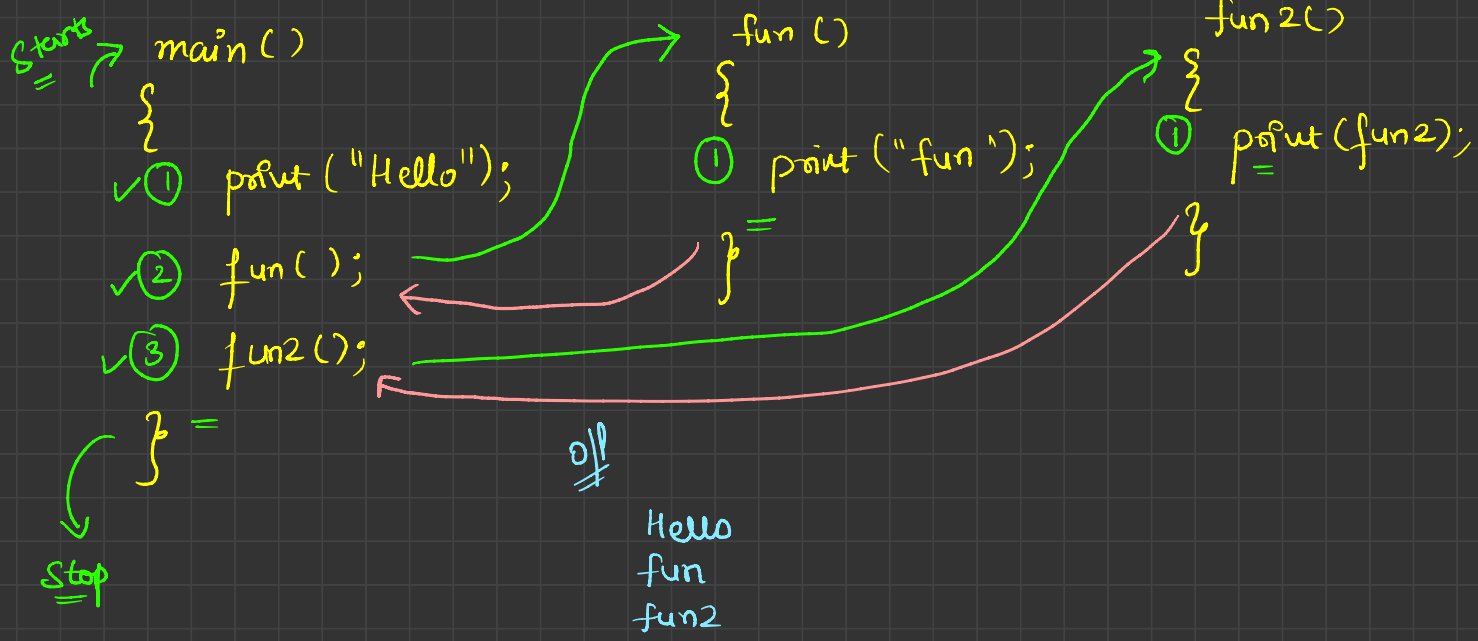
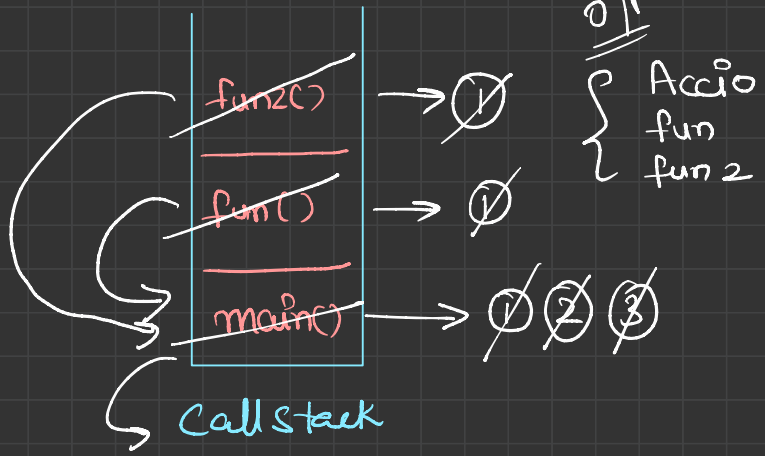
Recursion


Starts
main()
{
1 print("Hello");
2 fun();
3 fun2();
}
Stop
fun()
{
1 print("fun");
}
fun2()
{
1 print(fun2);
}
O/P
Hello
fun
fun2

```
main()
{
① printf("Accio");
② fun();
③ fun2();
}
```

```
void fun()
{
① printf("fun");
}  = return;
```

```
fun2()
{
① printf("fun2");
}  = return;
```



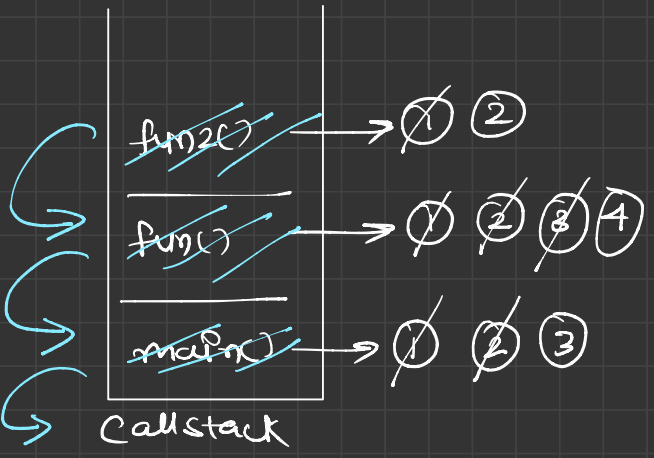
O/P

Accio
fun
fun 2

```
static void main()
{
1  print("Accio");
2  fun();
3  return;
}
```

```
static void fun()
{
1  print("fun");
2  fun2();
3  print("end");
4  return;
}
```

```
static void fun2()
{
1  print("fun2");
2  return;
}
```



o/p

Accio
fun
fun 2
end

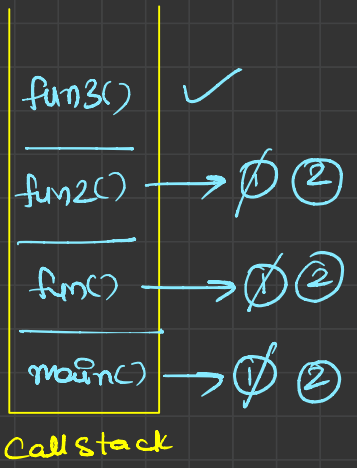
```
main()
{
1  print("I'm Main");
2  fun();
}
```

```
fun()
{
1  print(fun);
2  fun2();
}
```

```
fun2()
{
1  print(fun);
2  fun3();
}
```

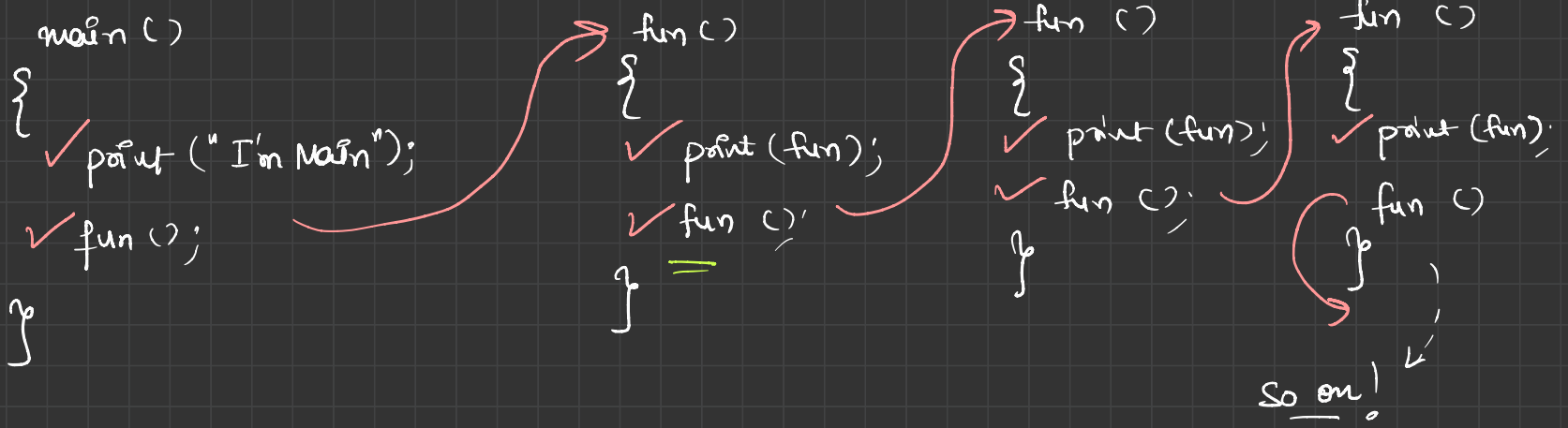
```
fun3()
{
   print(fun);
   fun4()
}
```

So on!

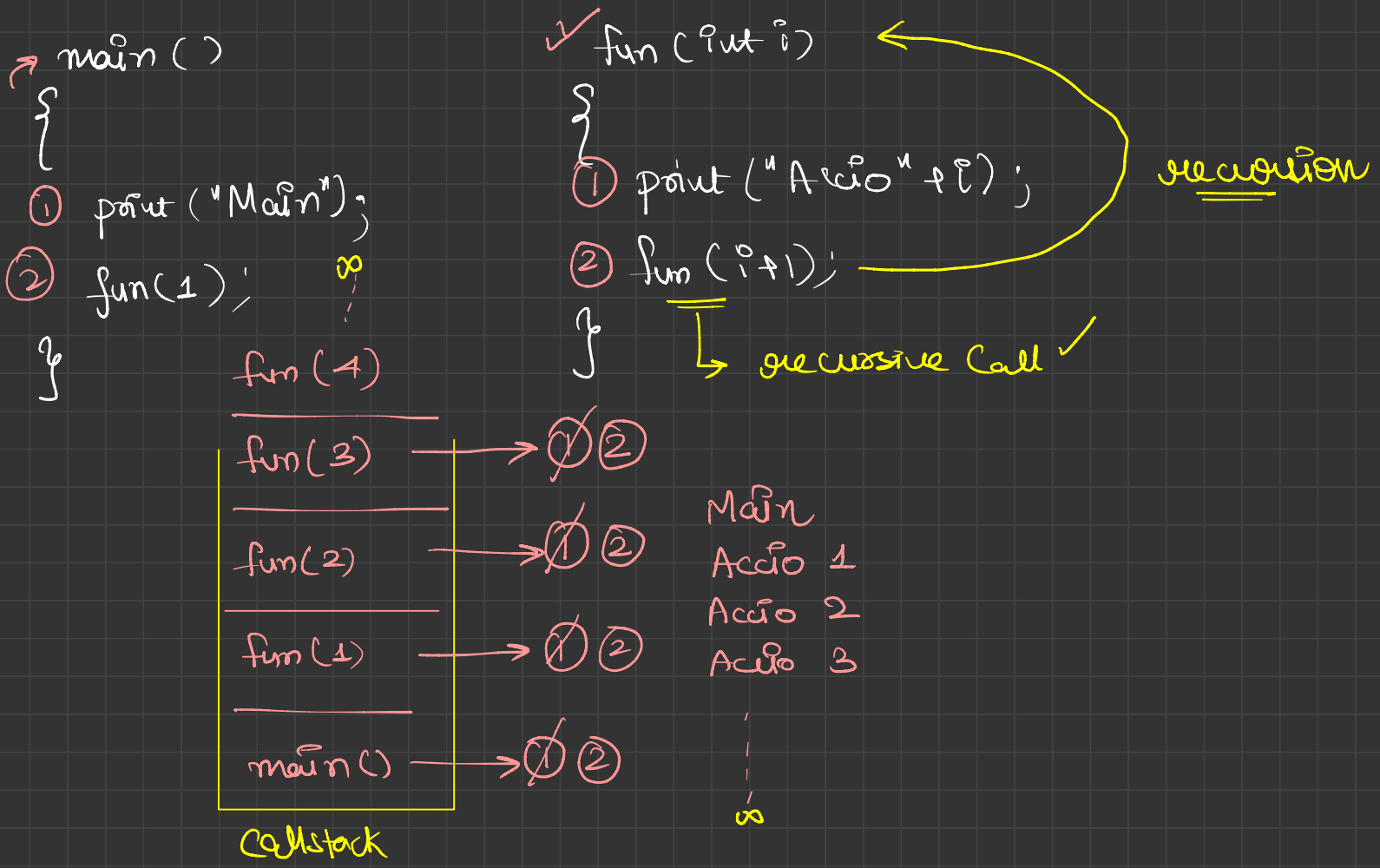


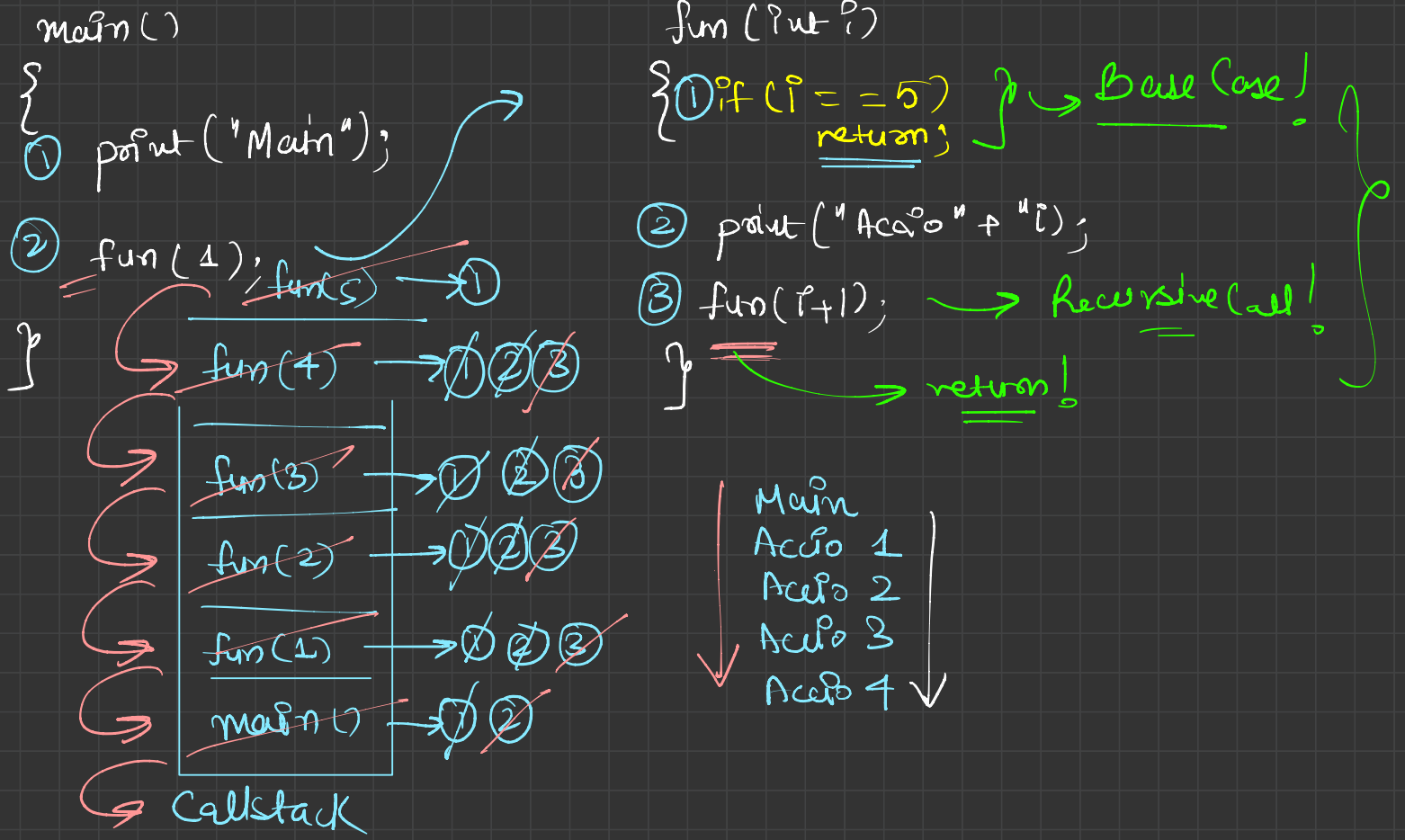
I'm Main
fun
fun

same work



I'm main
fun
fun
fun
.
.
.


main()
{
① print("Main");
② fun(1);
}
fun(int i)
{
① print("Accio"+i);
② fun(i+1);
}
recursion
recursive call
∞
fun(4)
fun(3)
fun(2)
fun(1)
main()
①②
Callstack
Main
Accio 1
Accio 2
Accio 3
∞


main()
{
① print("Main");
② fun(1);
}
fun(int i)
{ ① if(i==5) return; → Base Case!
② print("Accio" + i);
③ fun(i+1); → Recursive Call!
} → return!
fun(5) → ①
fun(4) → ① ② ③
fun(3) → ① ② ③
fun(2) → ① ② ③
fun(1) → ① ② ③
main() → ① ②
Callstack
Main
Accio 1
Accio 2
Accio 3
Accio 4

Recursion

1. base case
2. Recursive call
3. return statement

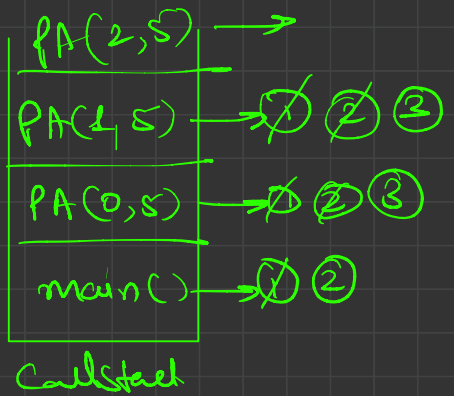
} Needed!

Q Print AcadioJobs 'N' times, using recursion!

```
public static void main()
{
① int N = 5;
② printAccio(0, 5);
}
```

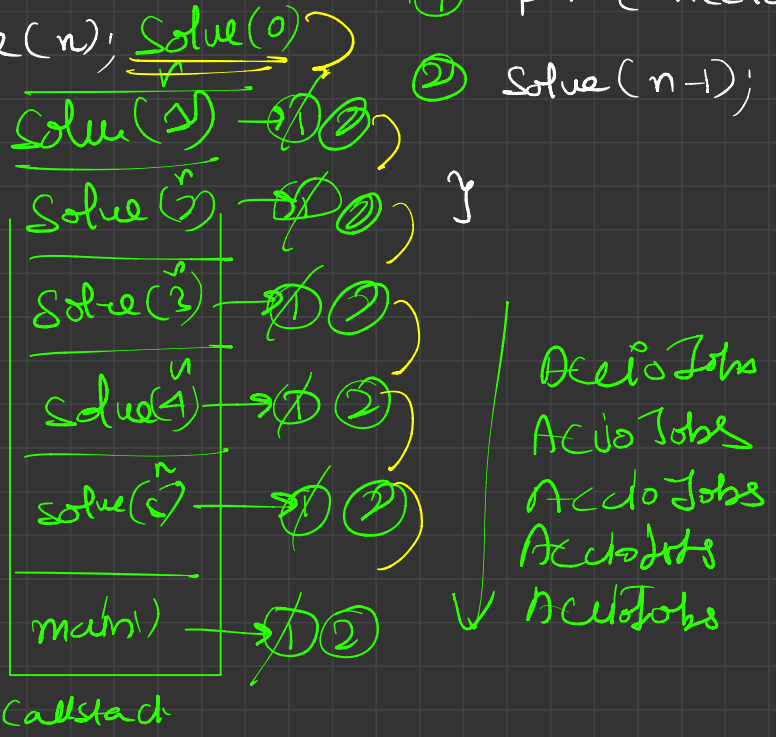
```
static void printAccio(int i, int N)
{ ① if(i == N) return;
  ② print(AcadioJobs);
  ③ printAccio(i+1, N);
}
```

AccioJobs
AccioJobs



```
main()
{
① int n = 5;
   solve(n);
}
```

```
solve(n)
{
   if(n==0) return;
① print("AccioJobs");
② solve(n-1);
}
```



AccioJobs
AccioJobs
AccioJobs
AccioJobs
AccioJobs

faith.

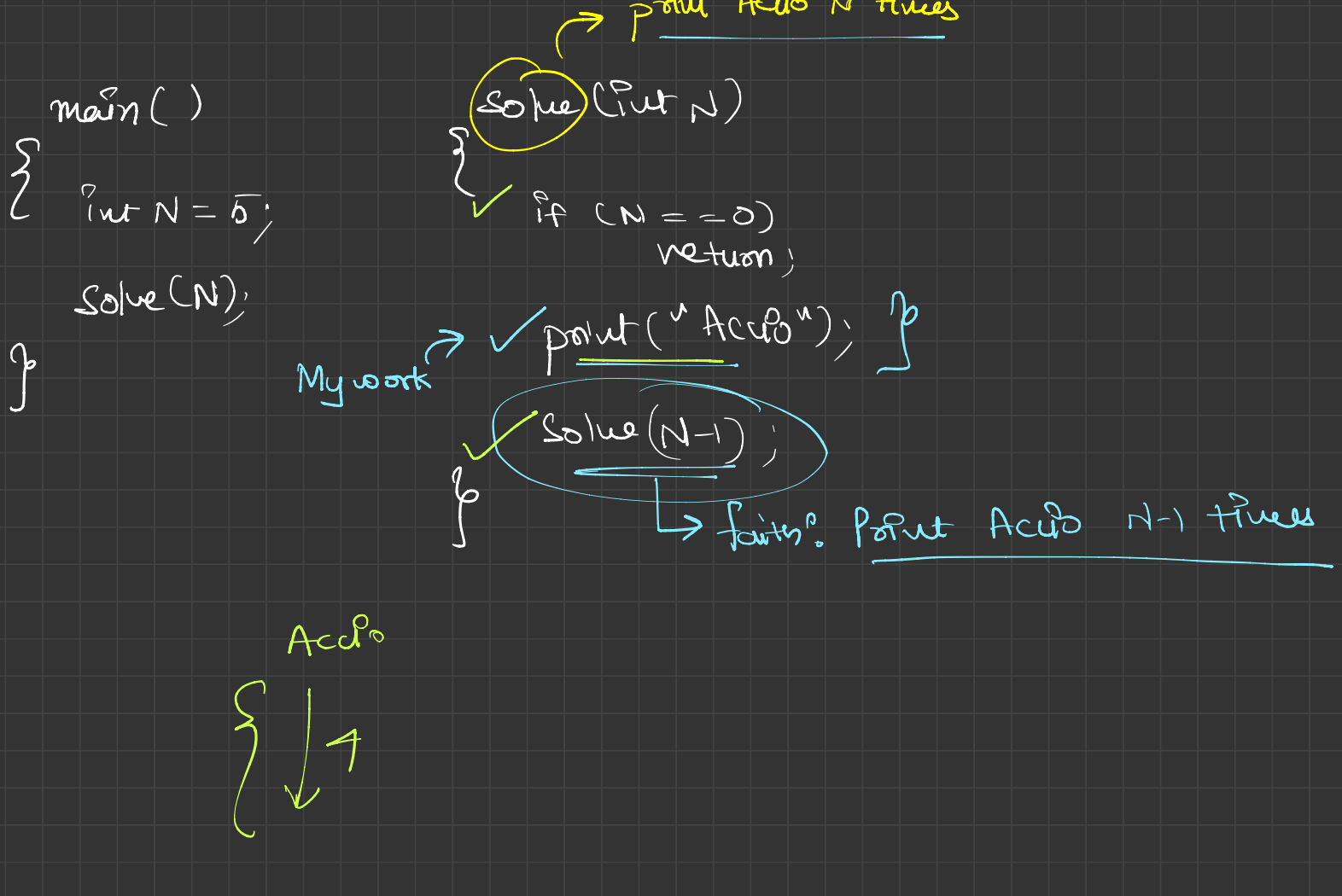
Recursion

↓

Breaking of Bigger Problems in Smaller Problem

↓

Always do your work, efficiency


print Accio N times
main()
{
int N = 5;
Solve(N);
}
Solve(int N)
{
if (N == 0)
return;
My work
print("Accio");
Solve(N-1);
}
Faith: Print Accio N-1 times
Accio
{
}

Q Print Numbers till N

N = 3

o/p 1 2 3

```java
public class Main {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n = sc.nextInt();

        printtillN(n);

    }

    public static void printtillN(int n) {
        //write your code
    }
}
```

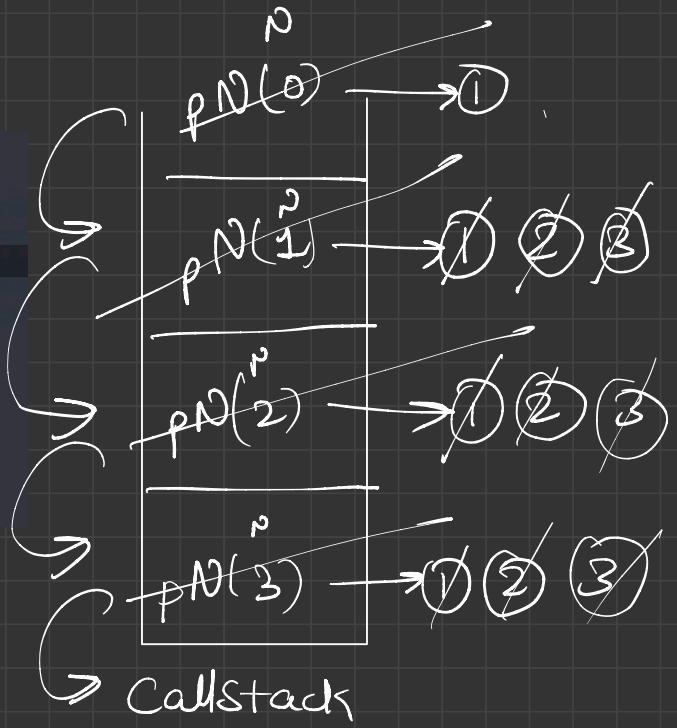
print no. --- 1 ---- n

1 ---- n-1

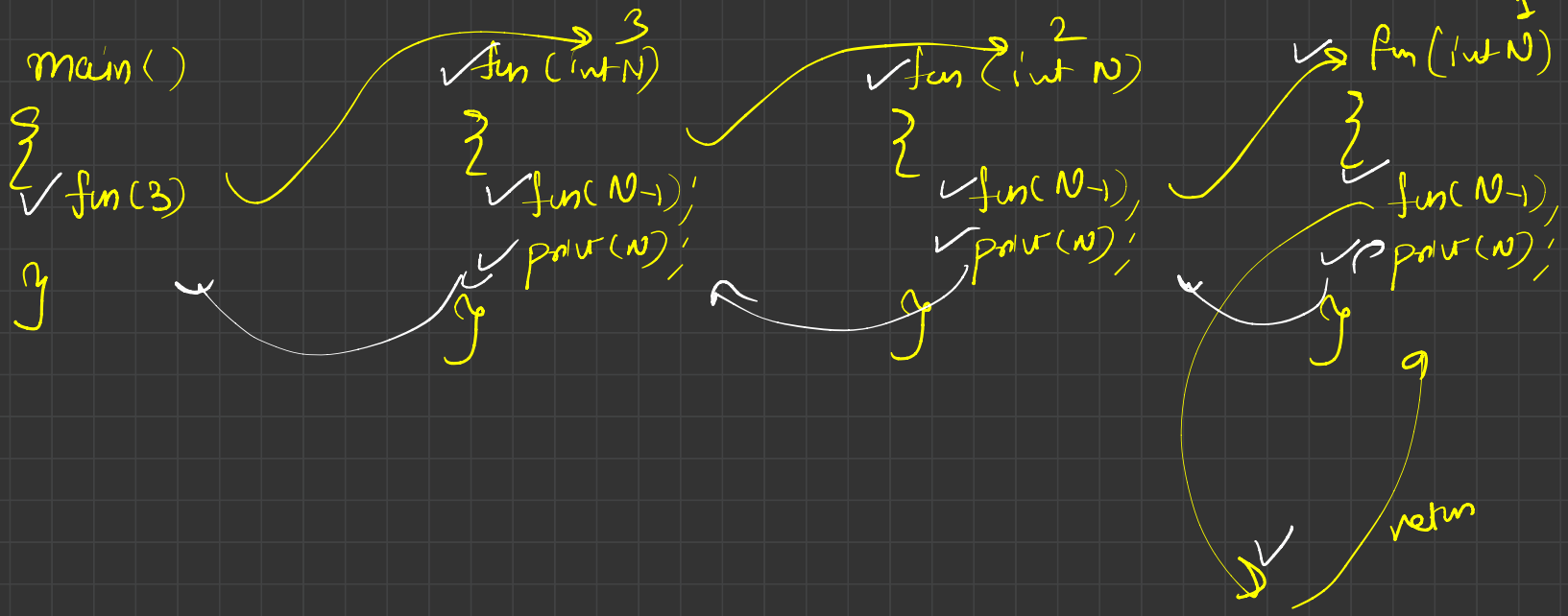
0

printtillN(int n)
{
if(n==0) return;
printtillN(n-1)
print(n);
}

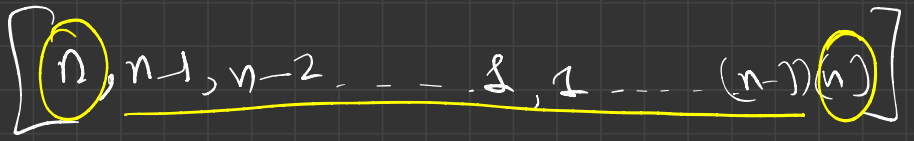
```java
// Faith: print number from 1...n
public static void printtillN(int n) {
    // base case
    if (n == 0) return;

    // prints number from 1......(n - 1)
    printtillN(n - 1);

    // your work
    System.out.print(n + " ");
}
```



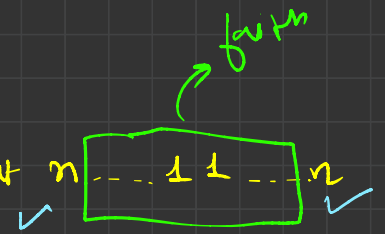
1 2 3

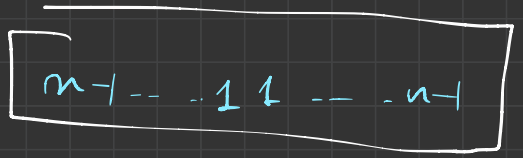

main()
{
fun(3)
}
fun(int N)
3
{
fun(N-1);
print(N);
}
fun(int N)
2
{
fun(N-1);
print(N);
}
fun(int N)
1
{
fun(N-1);
print(N);
}
return
1 2 3

Print Dec. & Inc.



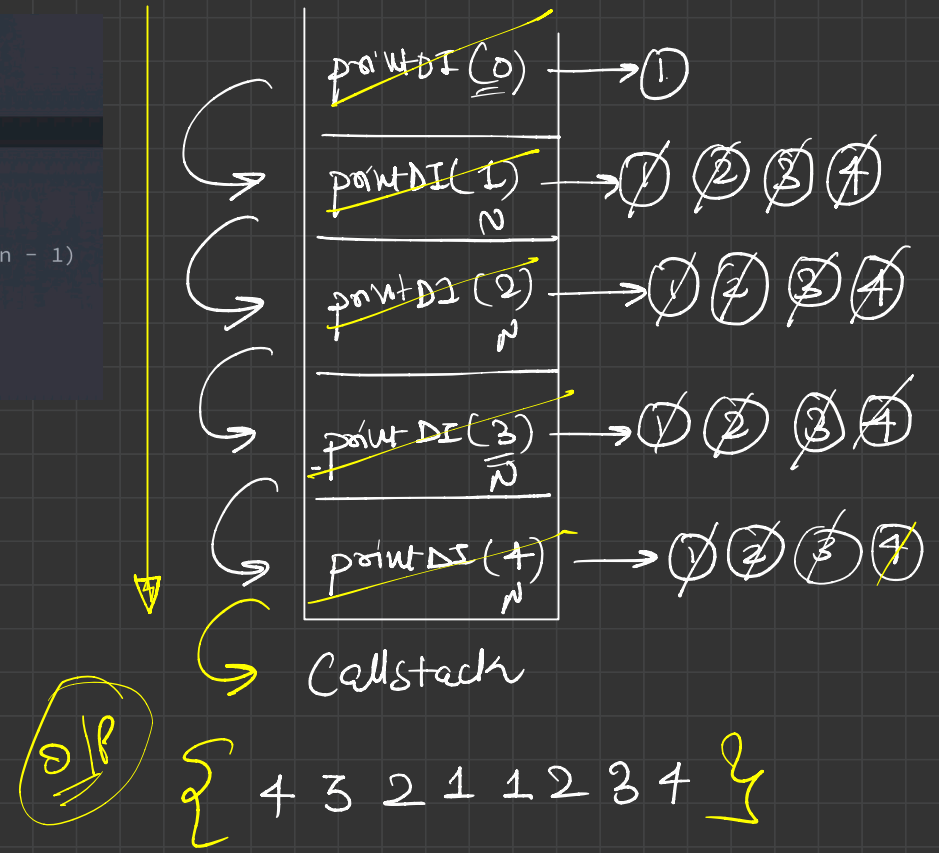

```java
public class Main {
  public static void main(String[] args){
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    printDI(n);
  }

  public static void printDI(int n) {
    // your code here
  }
}
```






```
printDI(int n)
{
  if (n == 0) return;
  print(n);
  printDI(n-1);
  print(n);
}
```

```java
// Faith: print number (n.....1 1.....n)
public static void printDI(int n) {
  // Base case
  if (n == 0) return;

  System.out.println(n);

  // print number (n - 1).....1 1......(n - 1)
  printDI(n - 1);

  System.out.println(n);
}
```



N^{th} fibonacci

0 1 1 2 3 5 8 13 21 ....

✓ 1st 2nd

fibo(N) = fibo(N-1) + fibo(N-2);

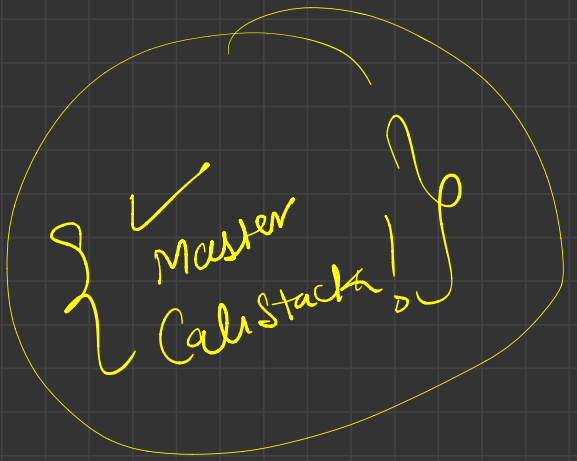
faith: returns Nth fibonacci

```
int fibo(int N)
{
    if(N==1) return 0;
    if(N==2) return 1;
    int a = fibo(N-1);
    int b = fibo(N-2);
    return a+b;
}
```

} → Base case

} → Recursive call

```java
// Faith: returns nth fibo
public static int fib(int n){
    // Base Case
    if (n == 1) {
        return 0;
    }
    if (n == 2) {
        return 1;
    }

    // returns (n-1)th fibo
    int a = fib(n - 1);

    // returns (n-2)th fibo
    int b = fib(n - 2);

    int nthFibo = a + b;
    return nthFibo;
}
```



```java
// Faith: returns nth fibo
public static int fib(int n){
    // Base Case
    if (n == 1) {
        return 0;
    }
    if (n == 2) {
        return 1;
    }

    // returns (n-1)th fibo
    int a = fib(n - 1);

    // returns (n-2)th fibo
    int b = fib(n - 2);

    int nthFibo = a + b;
    return nthFibo;
}
```