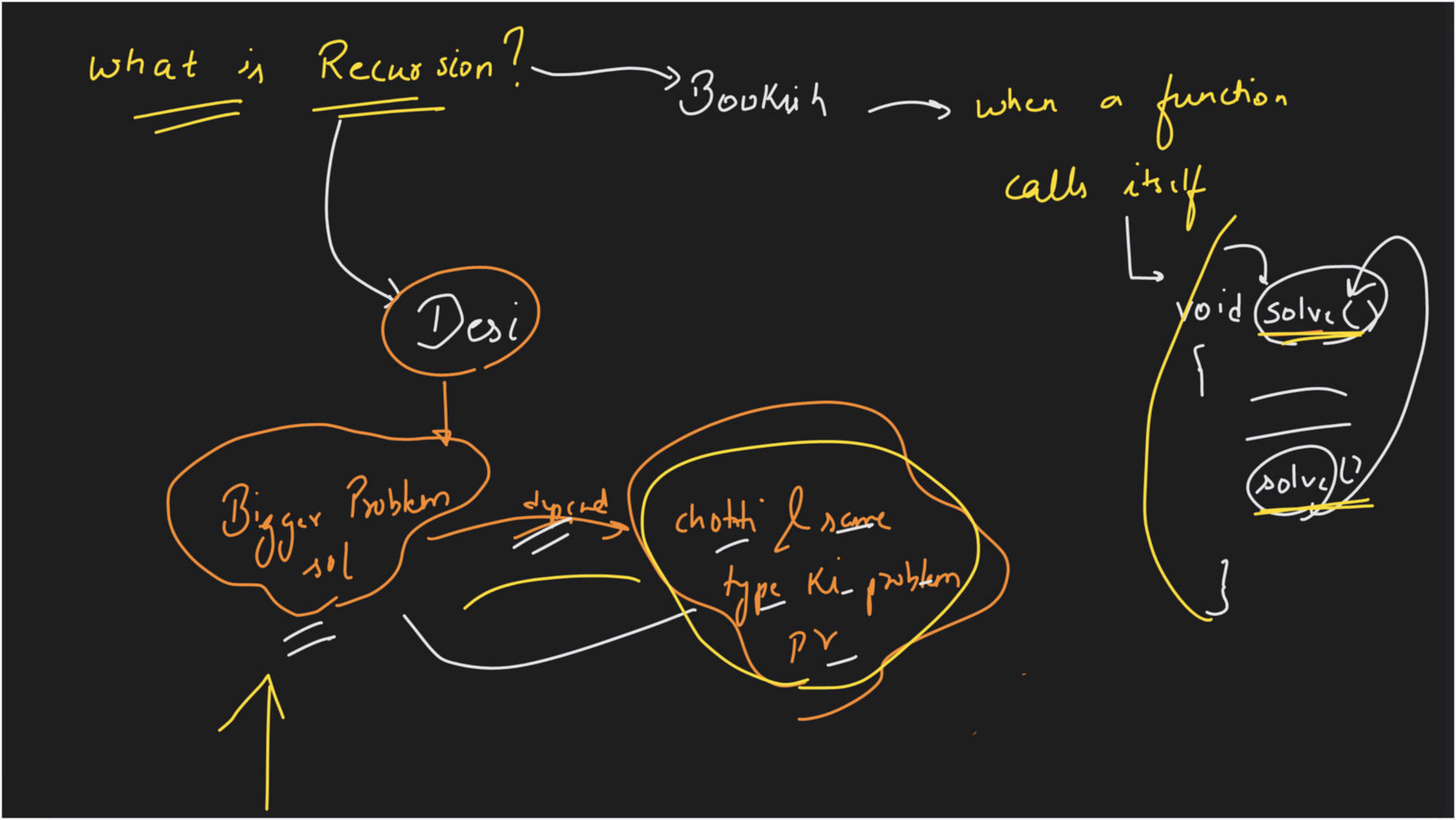
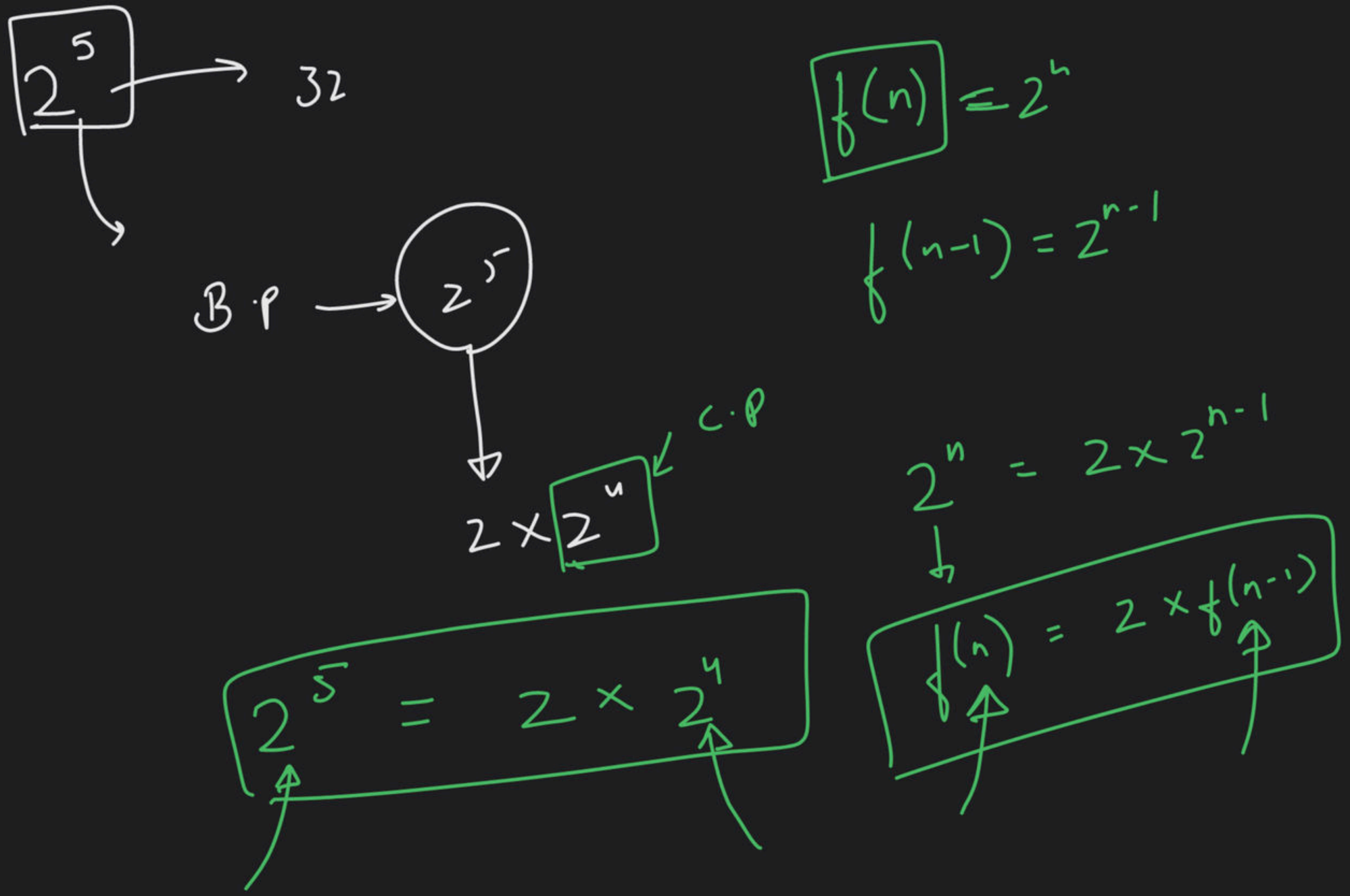
Recursion - Level 1

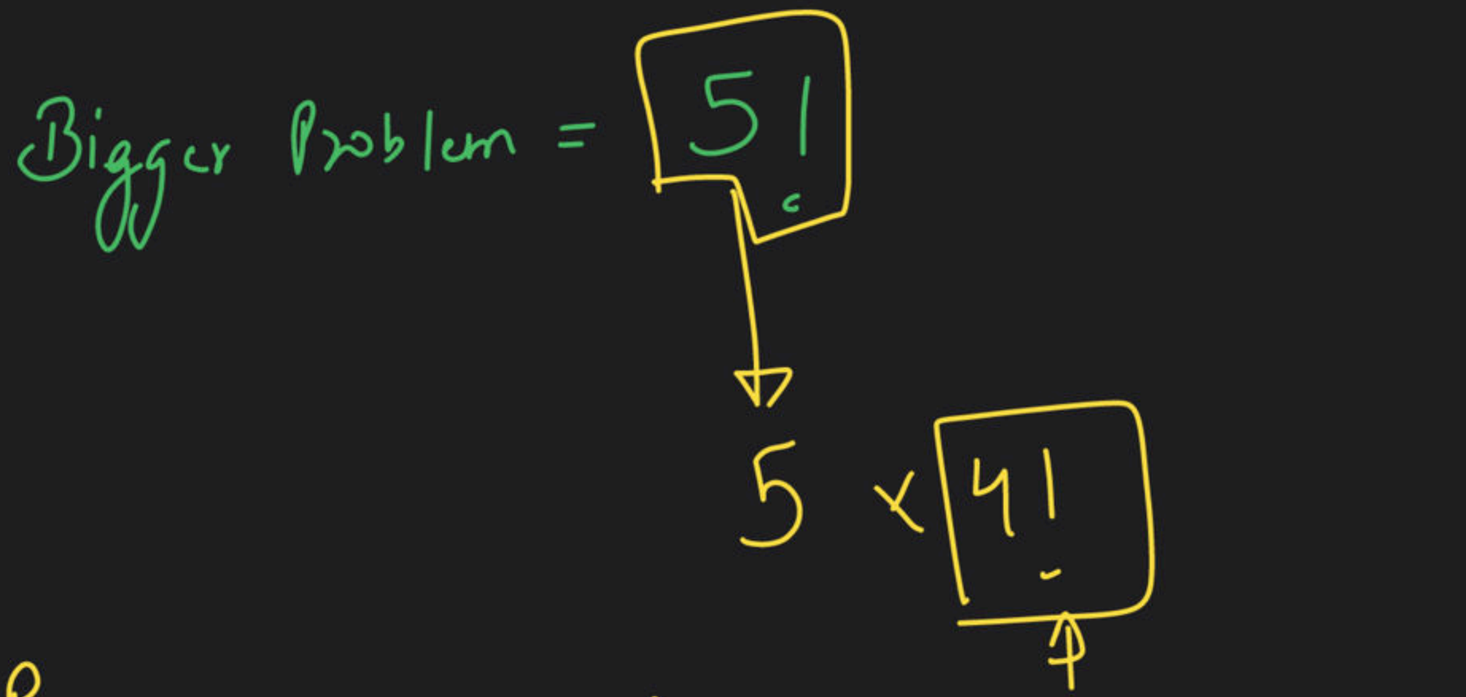
Special class

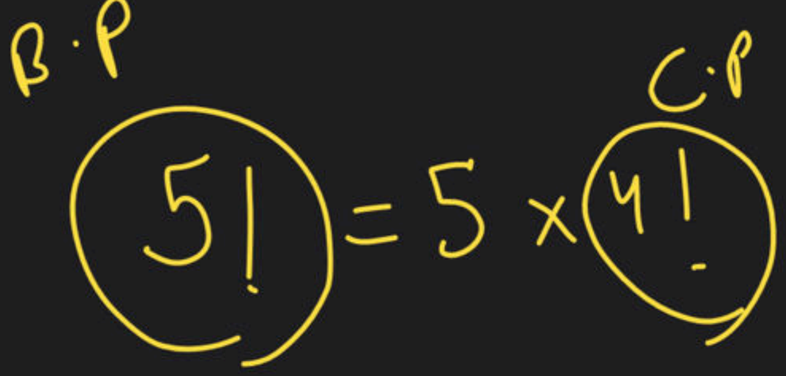
Love Babbar • Mar 10, 2023

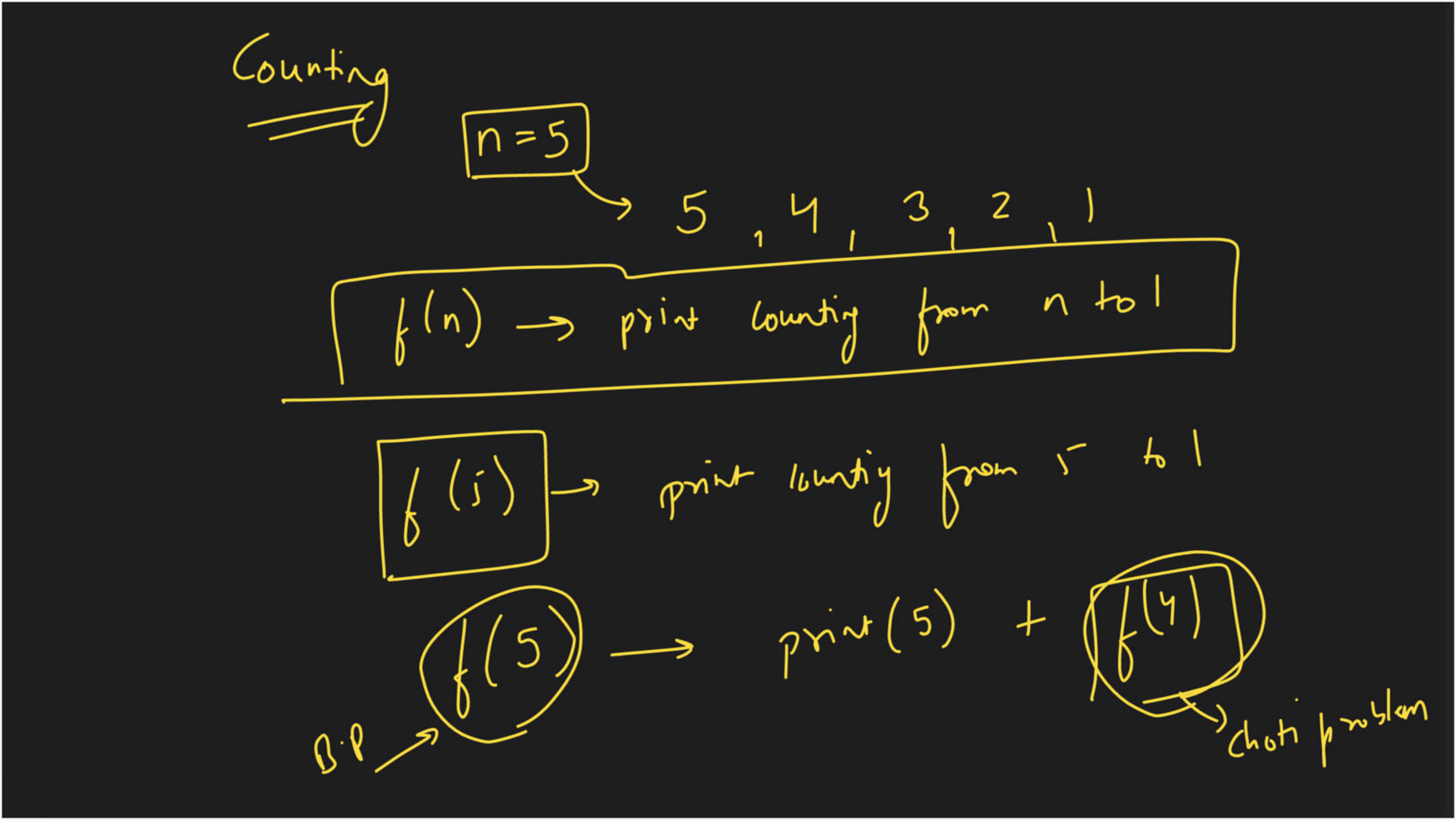

What is Recursion?
Bookish
when a function calls itself
Desi
Bigger Problem sol
depend
chotti si same type ki problem pr
void solve()
{
solve()
}

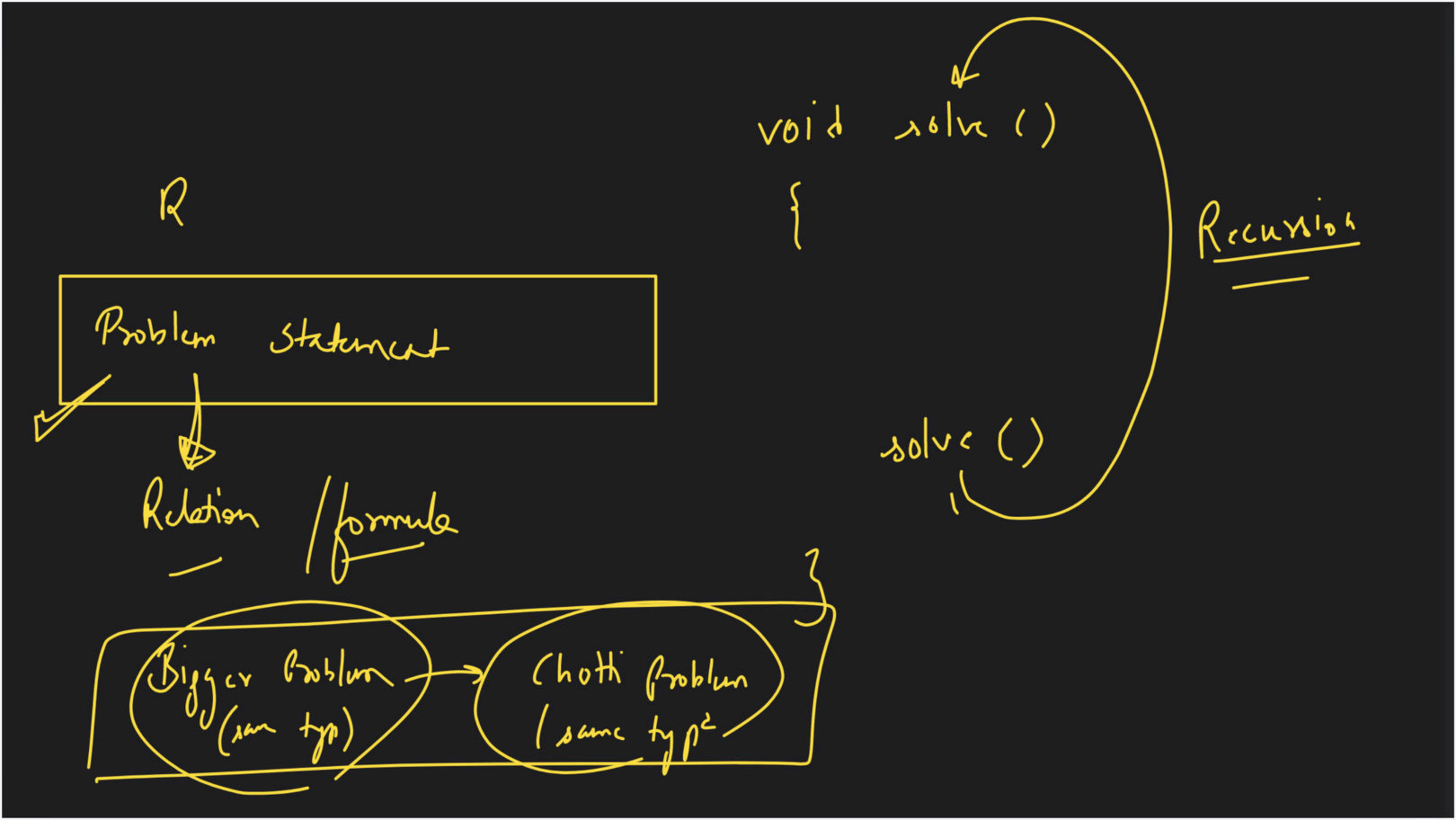

2^5
32
B.P
2^5
C.P
2×2^4
$2^5 = 2 \times 2^4$
$f(n) = 2^n$
$f(n-1) = 2^{n-1}$
$2^n = 2 \times 2^{n-1}$
$f(n) = 2 \times f(n-1)$

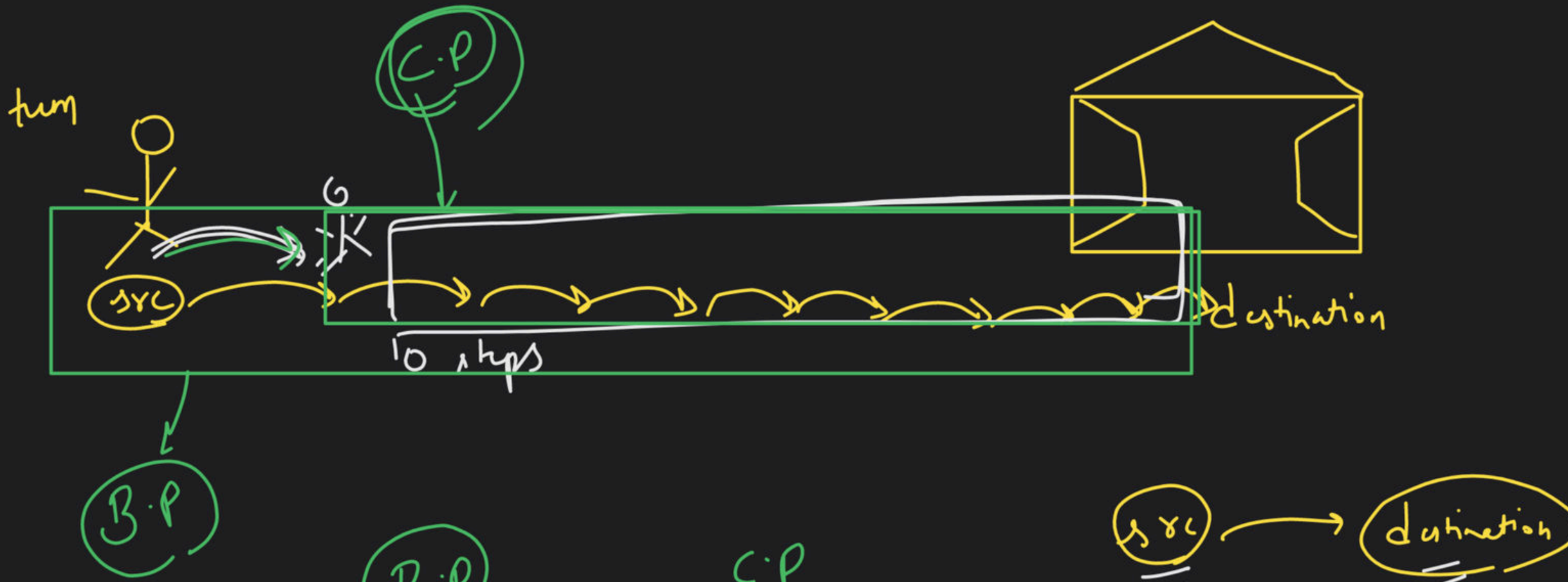
Factorial





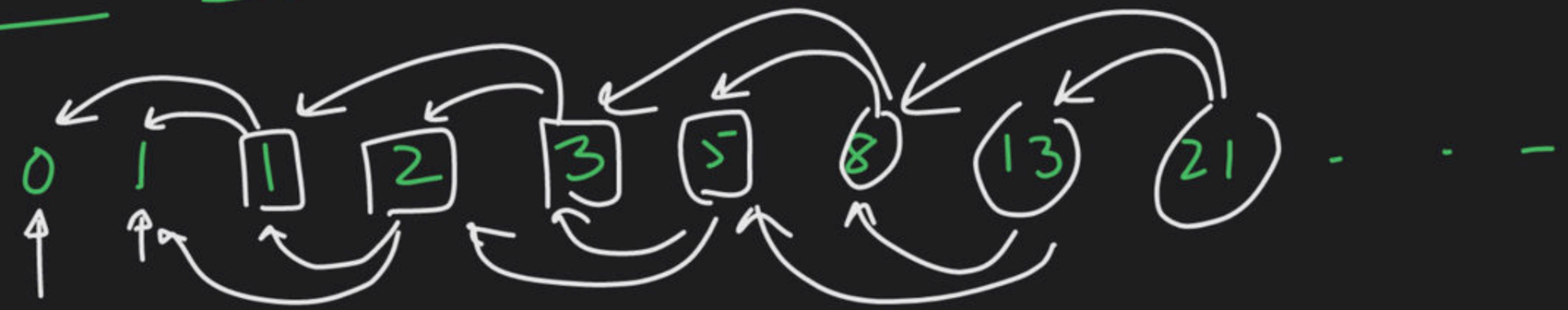

Counting
n = 5
5 , 4 , 3 , 2 , 1
f(n) → print counting from n to 1
f(i) → print counting from i to 1
f(5) → print(5) + f(4)
B.P
Choti problem


void solve ()
{
solve ()
Recursion
R
Problem Statement
Relation // formula
Bigger Problem (same type)
Chotti Problem (same type)



B·P
total no of step = 1 ± C·P total no of step

Fibonacci Series



n^{th} term = $(n-1)^{th}$ term + $(n-2)^{th}$ term

$$f(n) = f(n-1) + f(n-2)$$

D.P

C.P

C.P

Problem Statement

- i/p → n
- o/p → print 2^n

Biggest Problem → 2 to the power of n → 2^n

$2^n = 2 \times 2 \times 2 \times 2 \times \cdots \times 2$ (n times)

formula → $2^n = 2 \times 2^{n-1}$

Ex- n = 3

o/p → $2^3 = 8$

n = 6

o/p → 2^6 → 64

n = 10

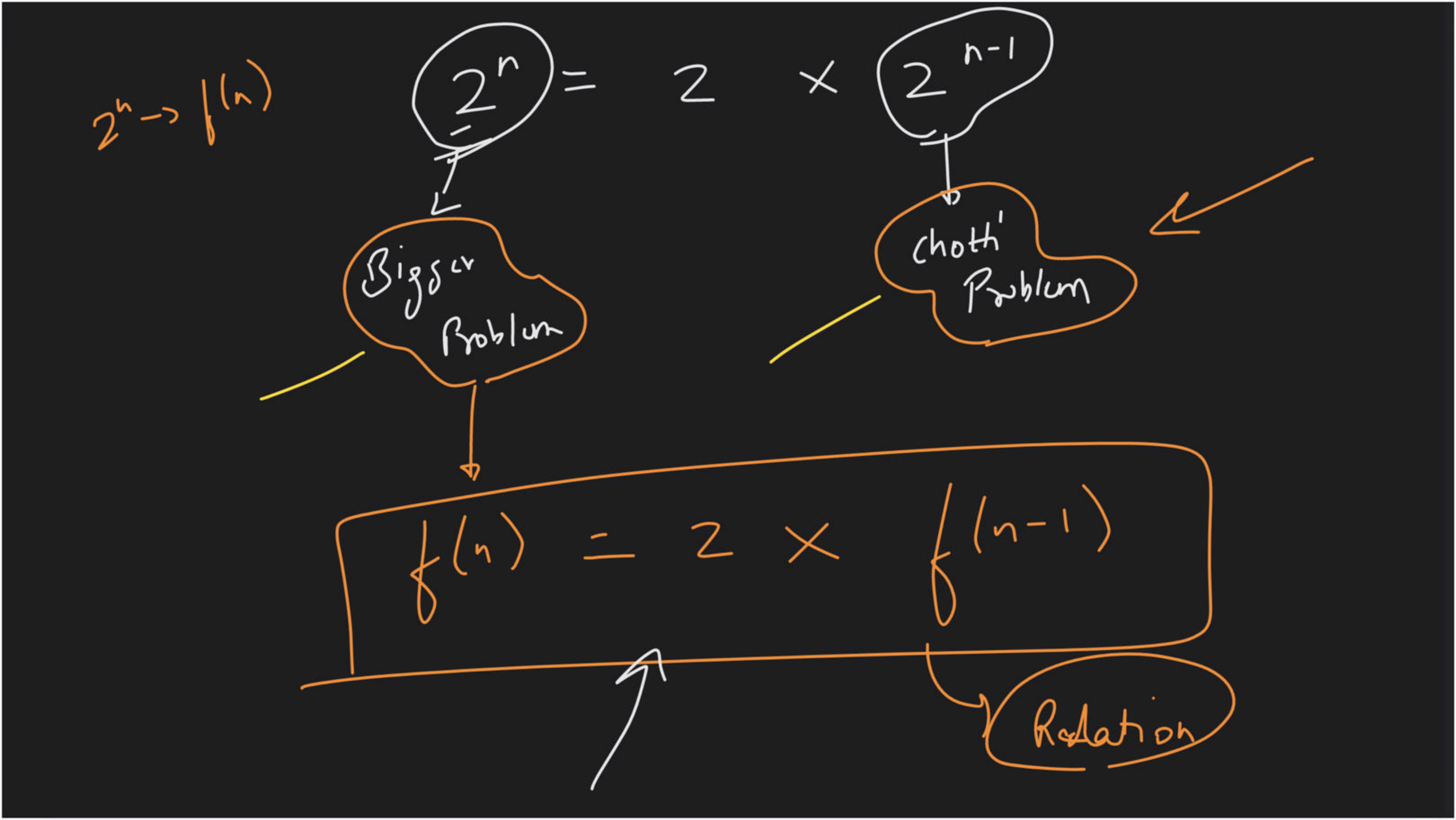
o/p → 2^{10} → 1024

$2 \times 2 = 2^2$

$2^2 \times 2^3 = 2^5$

$2^5 \times 2^{12} = 2^{22}$

$2^a \times 2^b = 2^{a+b}$


$2^n \rightarrow f(n)$
$2^n = 2 \times 2^{n-1}$
Bigger Problem
chotti Problem
$f(n) = 2 \times f(n-1)$
Relation

Problem statement →

i/p → n

o/p → n!

find factorial of n

$$n! = n \times \underbrace{(n-1) \times (n-2) \times (n-3) \cdots 3 \times 2 \times 1}_{(n-1)!}$$

$$n! = n \times (n-1)!$$

Ex

n = 5

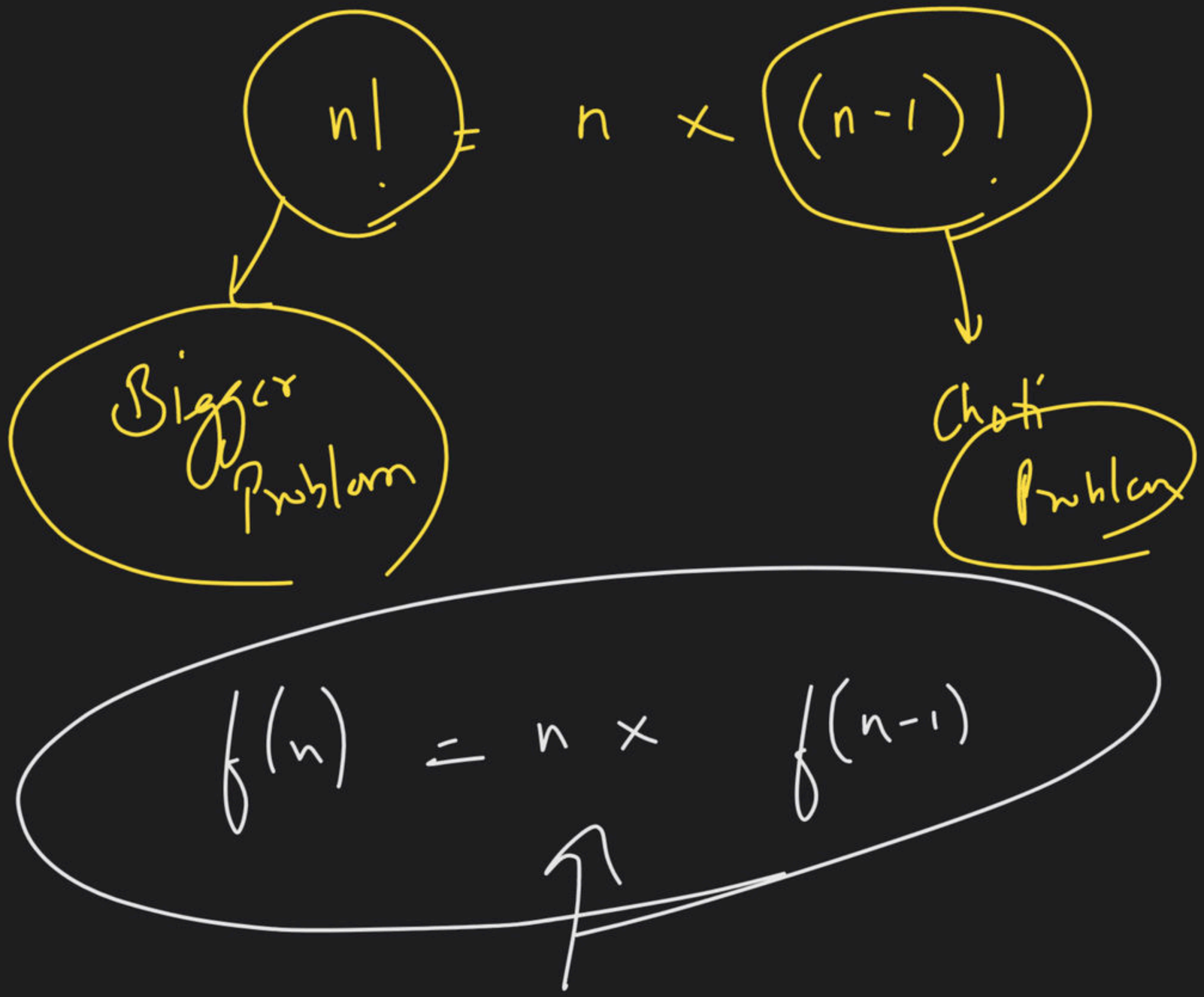
o/p → 5! → 120

n = 3

o/p → 3! → 6

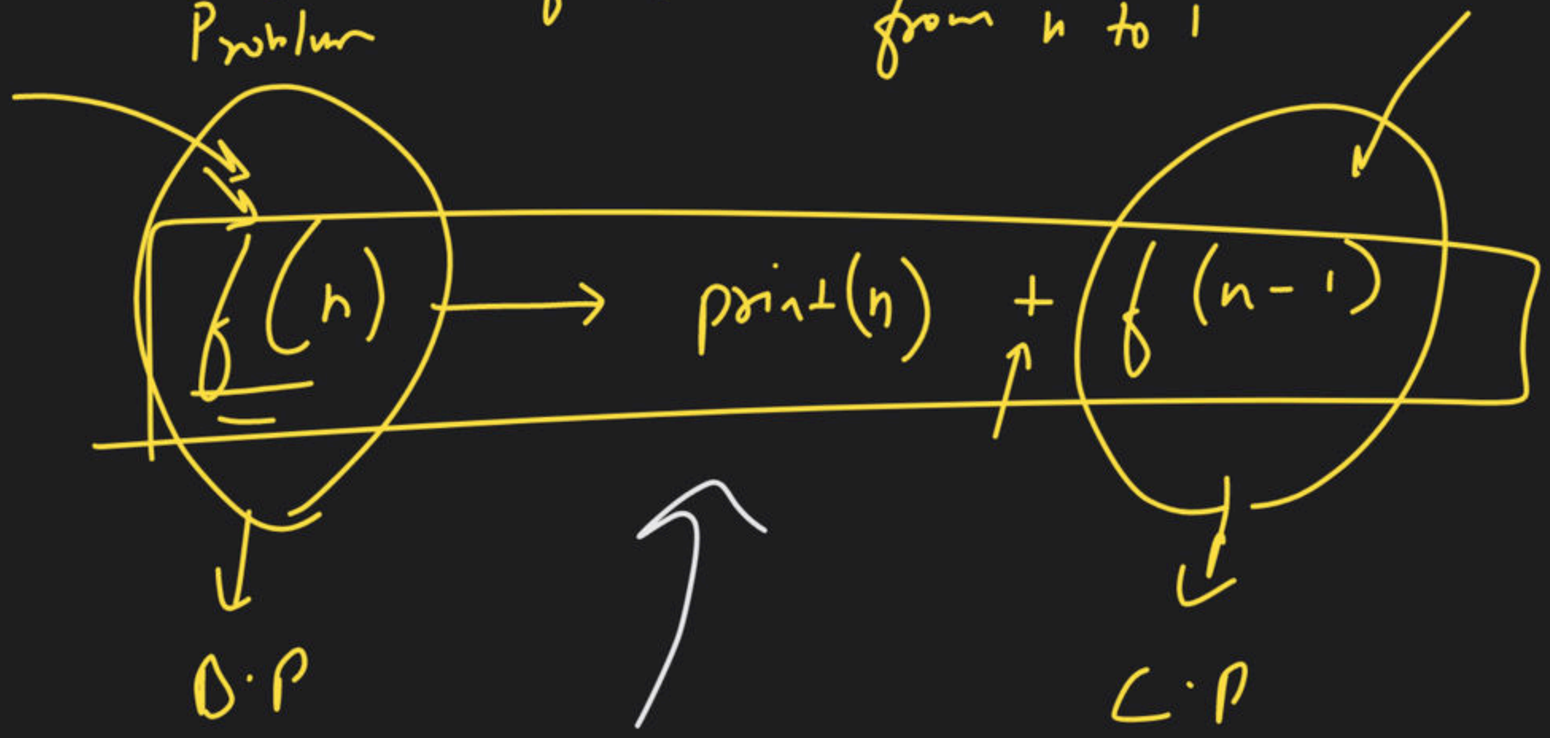
n = 4

o/p → 4! → 24


n! = n x (n-1)!
Bigger Problem
Choti Problem
f(n) = n x f(n-1)

Counting P.S → i/p → n
o/p → reverse counting → ulti giri
from n → 1

Bigger Problem → f(n) → reverse counting from n to 1



Ex → n=5

o/p → 5 4 3 2 1

n

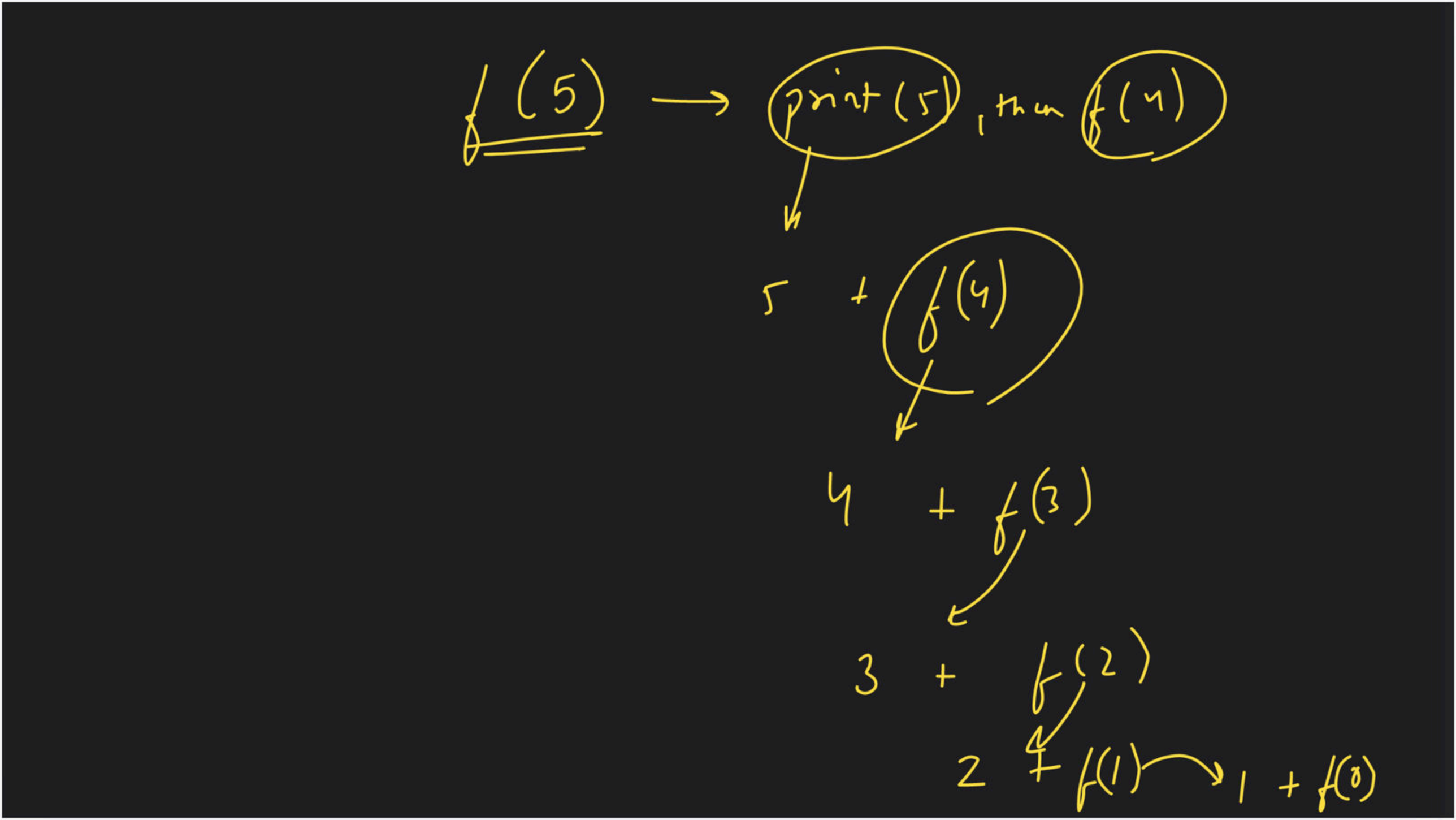
n-1

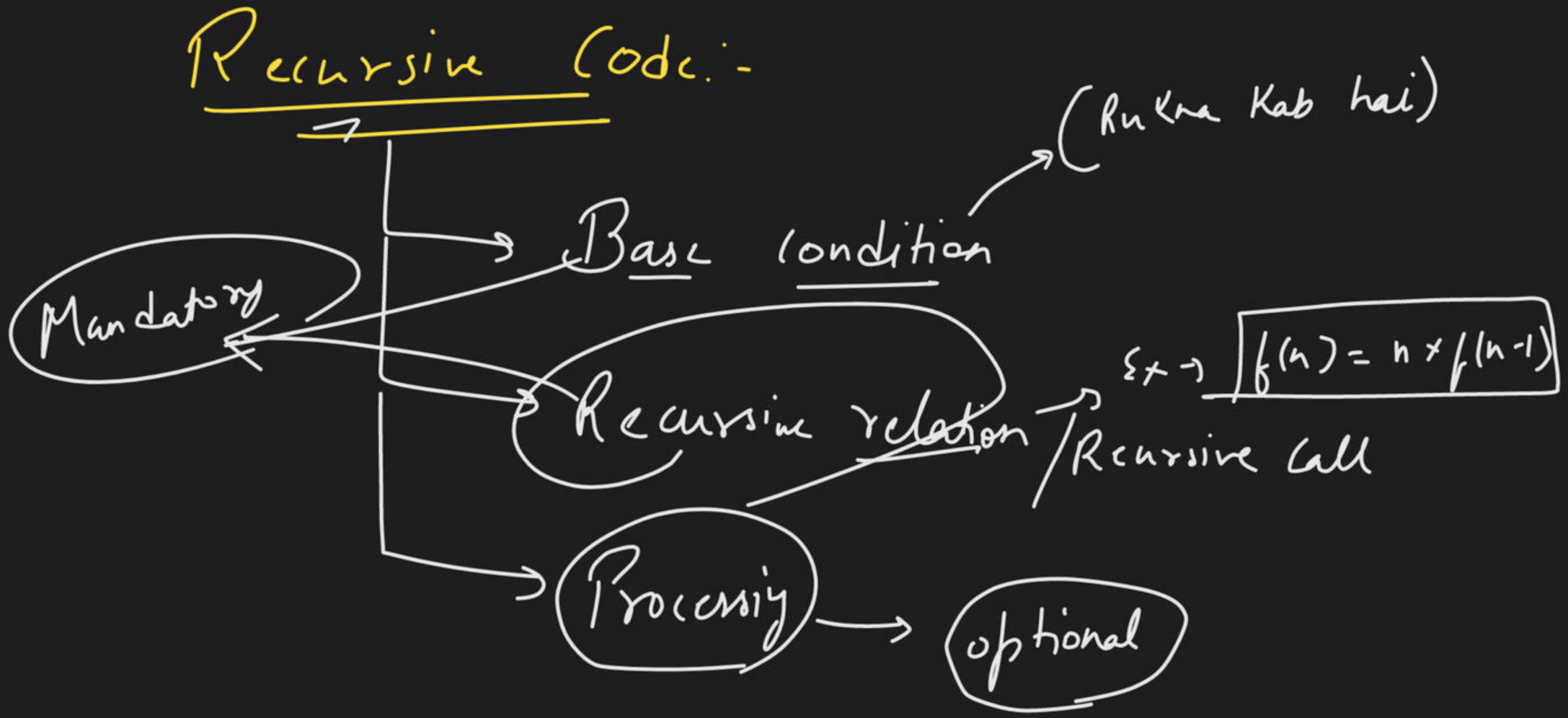
n = 2

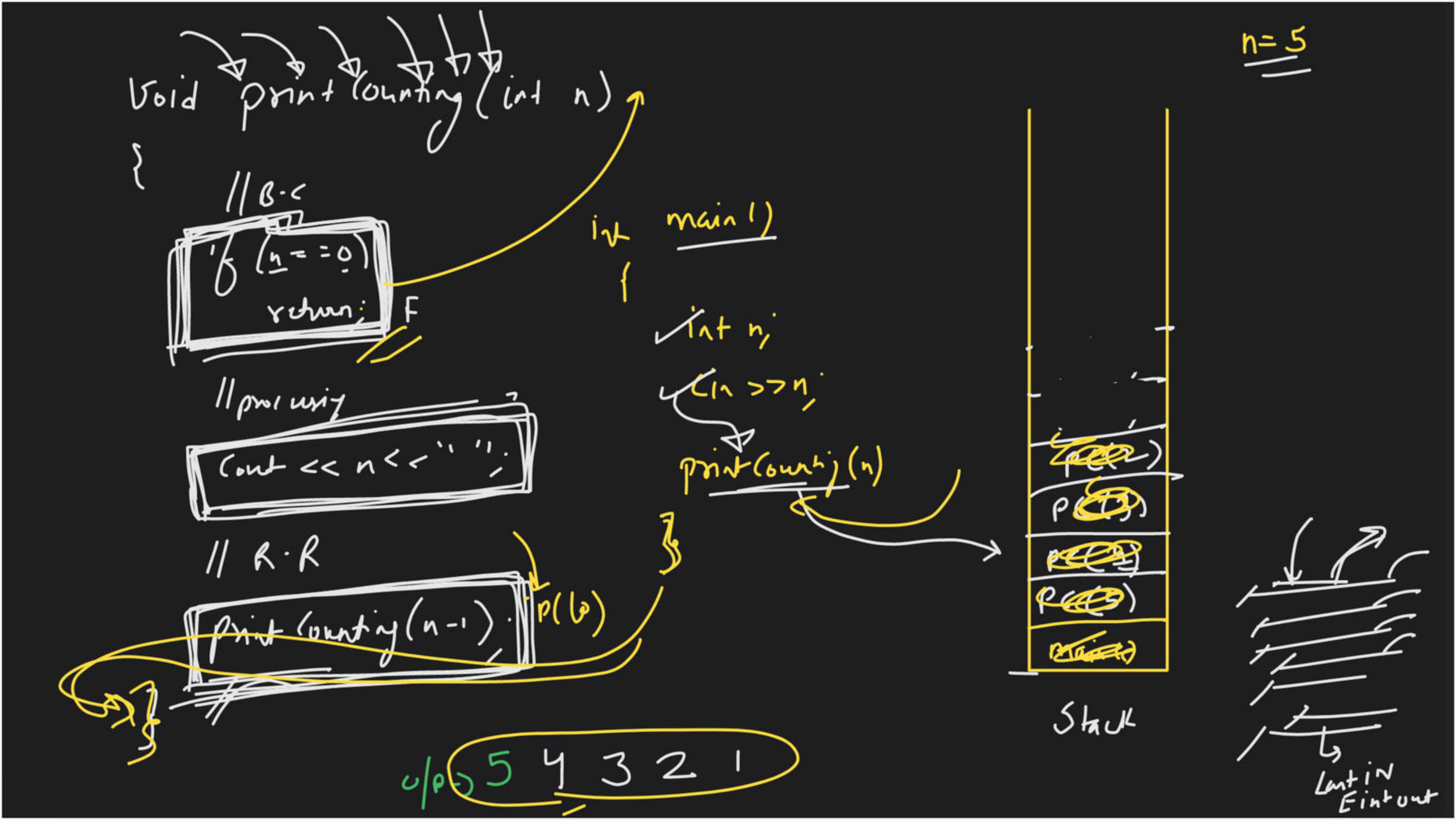
o/p → 2 1

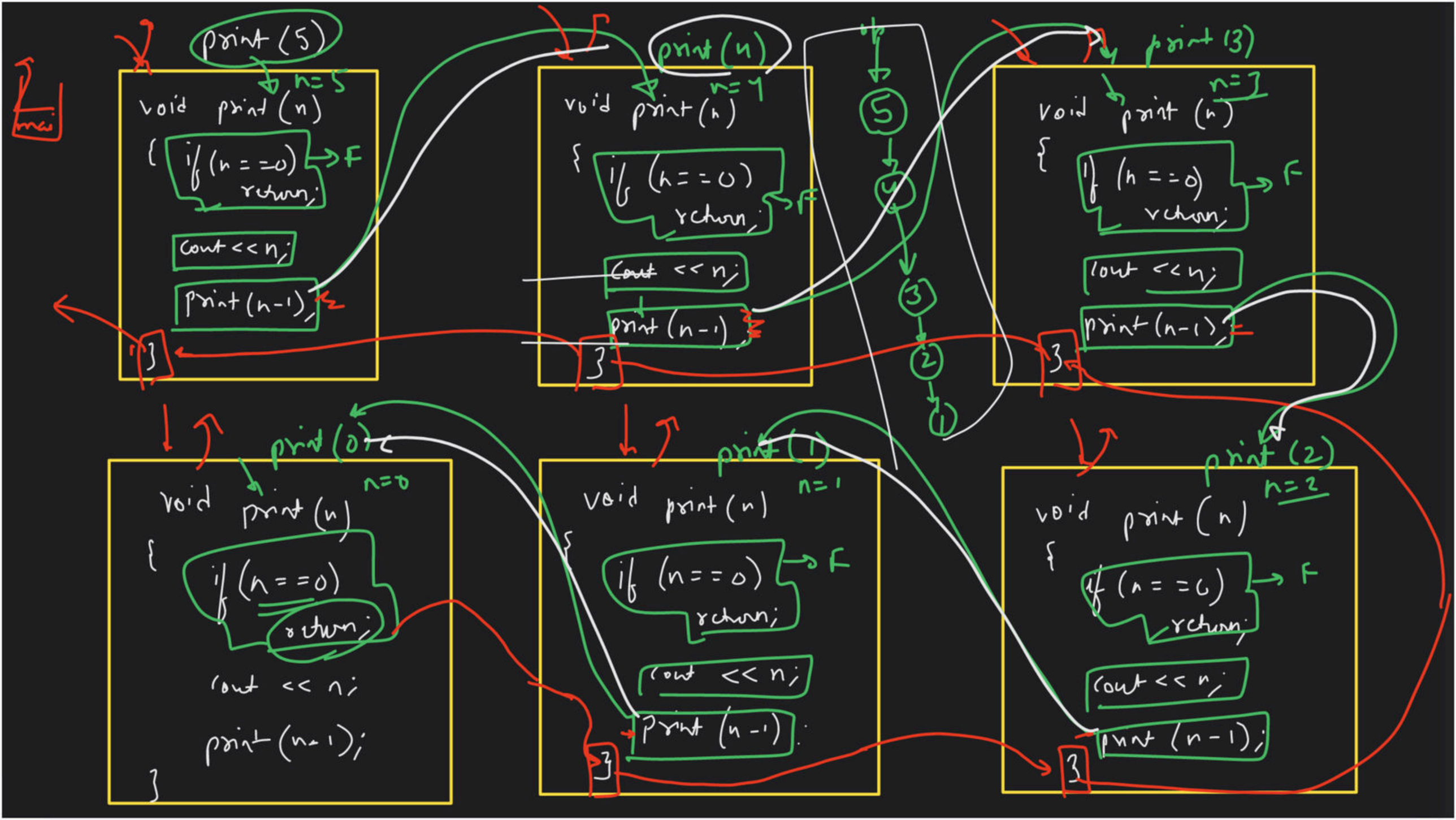
n = 10

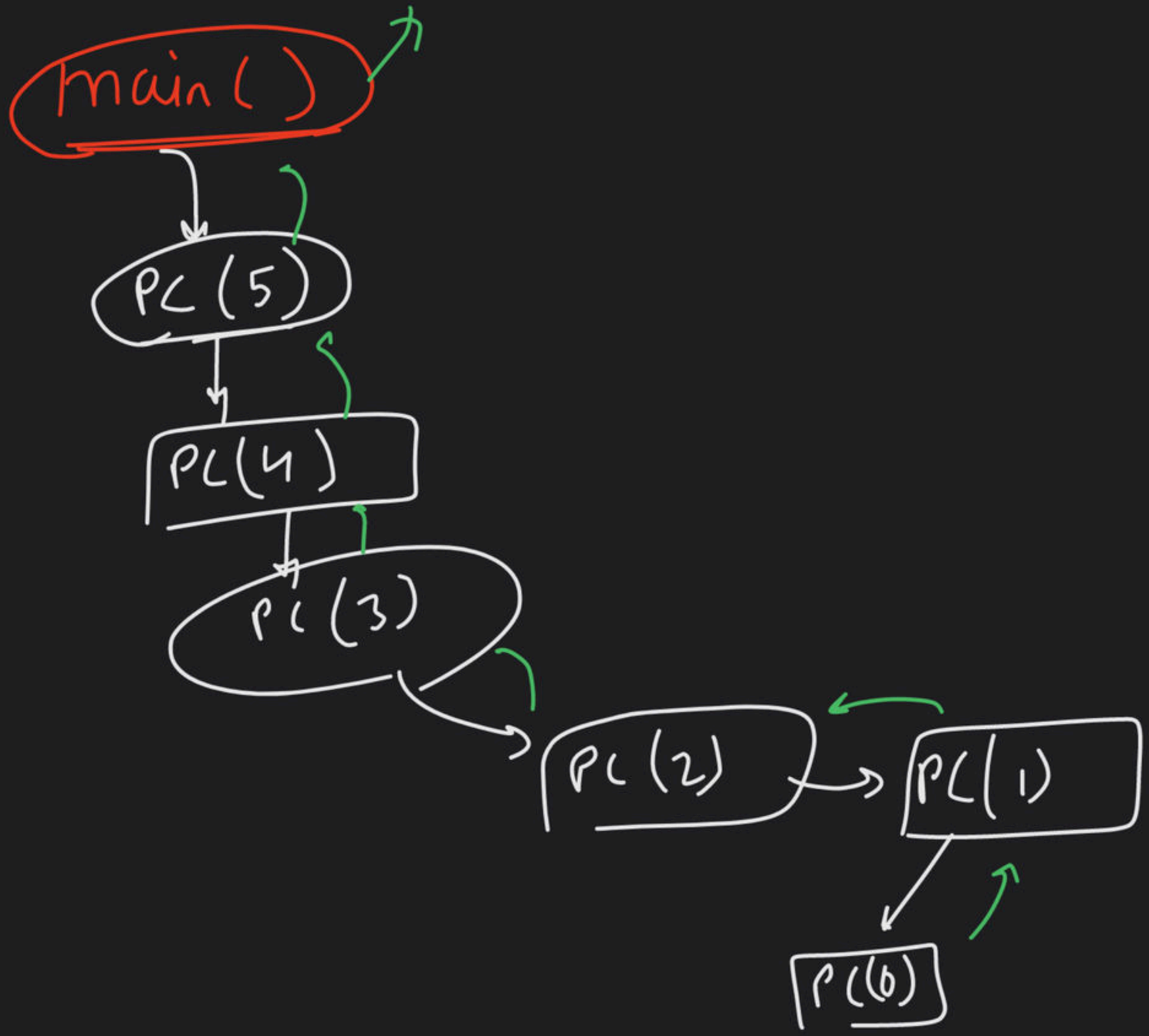
o/p → 10 9 8 7 6 5 4 3 2 1


f(5)
print(5)
then
f(4)
5 + f(4)
4 + f(3)
3 + f(2)
2 + f(1)
1 + f(0)


Recursive Code :-
Base condition
(Rukna Kab hai)
Mandatory
Recursive relation / Recursive call
Ex → f(n) = n × f(n-1)
Processing
optional

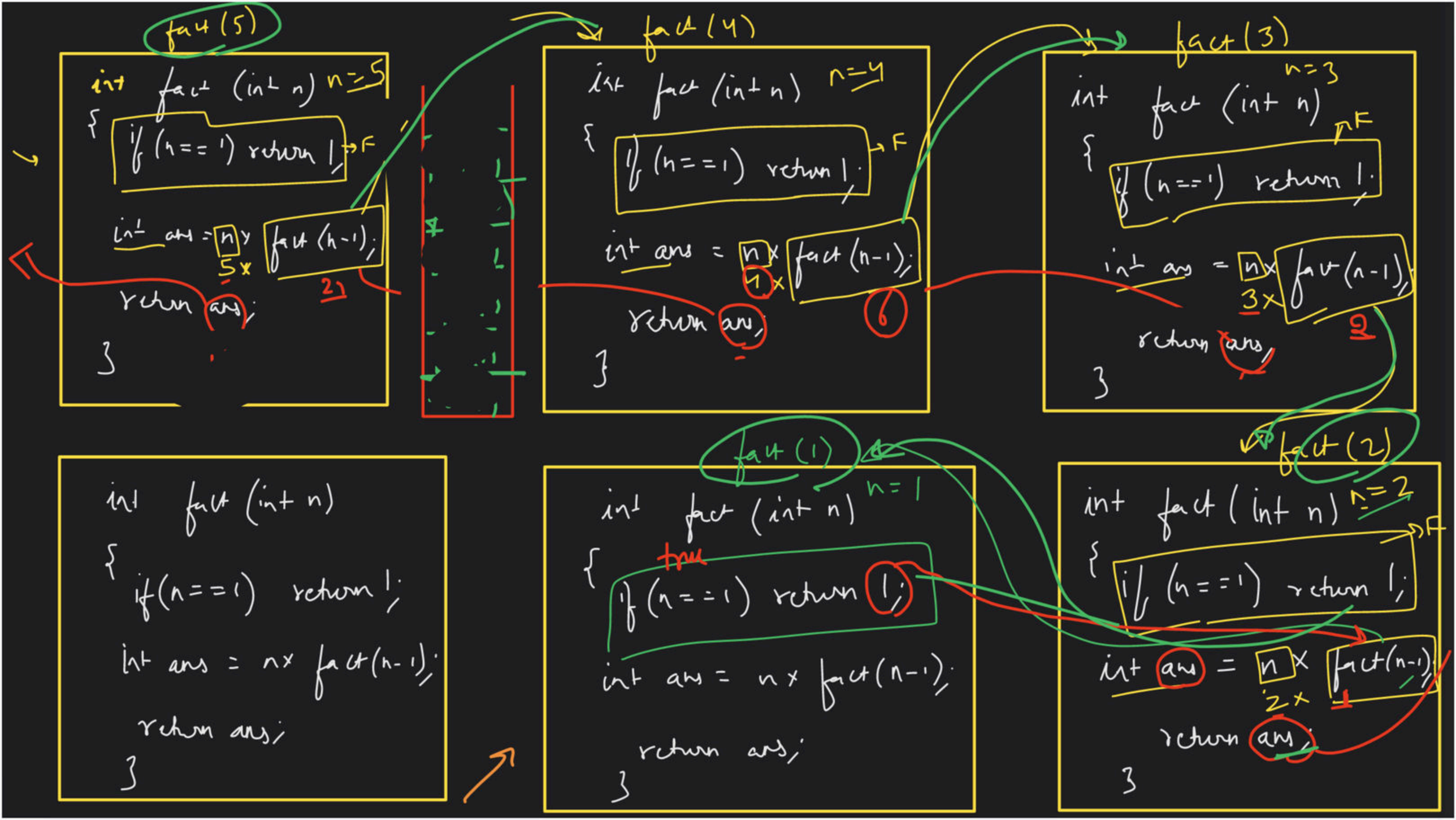

n=5
void print counting (int n)
{
// B.C
if (n==0)
return;
F
// processing
cout << n << " ";
// R.R
print counting (n-1);
P(bp)
int main()
{
int n;
cin >> n;
print counting (n)
Stack
o/p: 5 4 3 2 1
Int in
Eint out

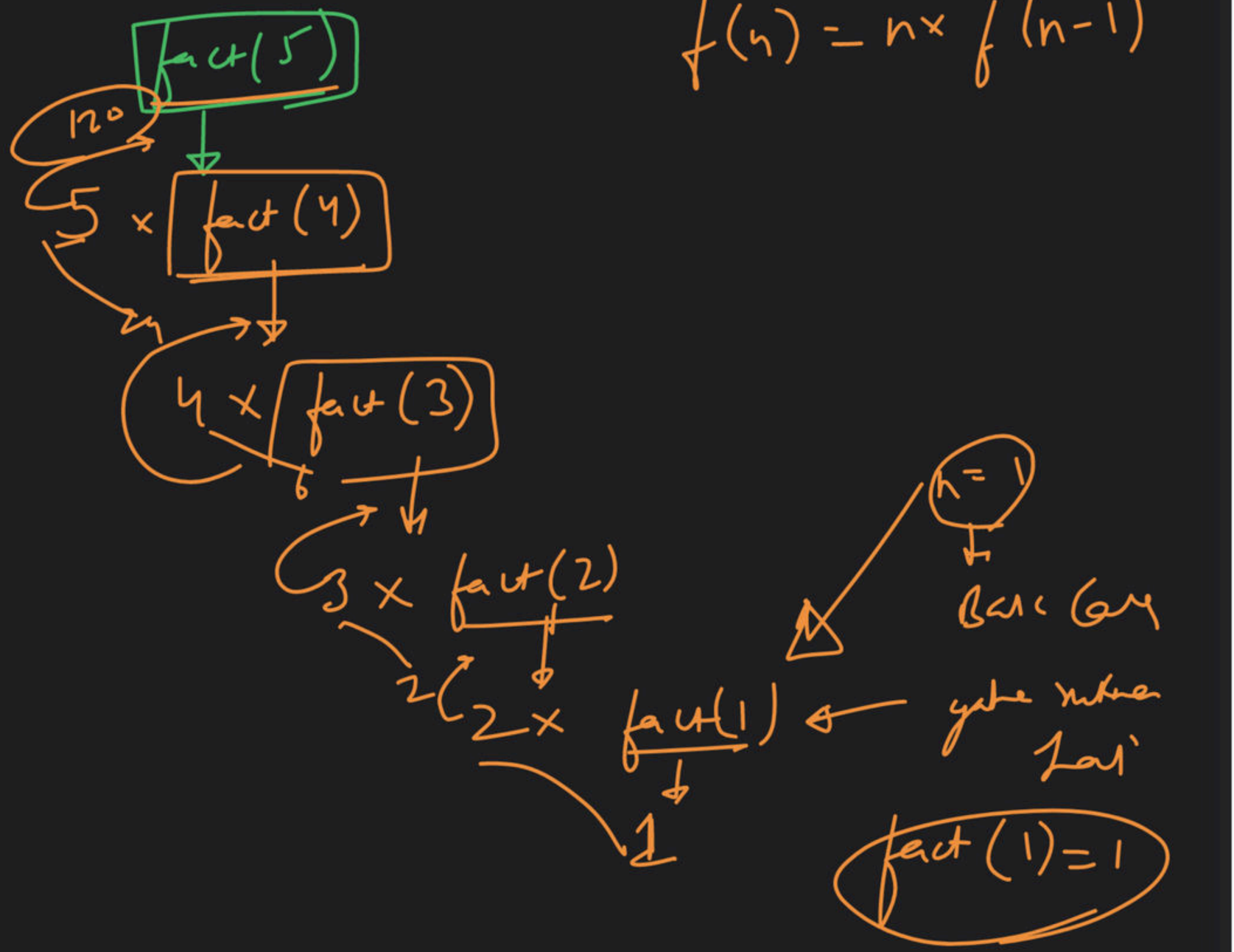

main
print(5)
n=5
void print(n)
{
if(n==0) return;
F
cout << n;
print(n-1);
}
print(4)
n=4
void print(n)
{
if(n==0) return;
F
cout << n;
print(n-1);
}
print(3)
n=3
void print(n)
{
if(n==0) return;
F
cout << n;
print(n-1);
}
5
4
3
2
1
print(0)
n=0
void print(n)
{
if(n==0) return;
cout << n;
print(n-1);
}
print(1)
n=1
void print(n)
{
if(n==0) return;
F
cout << n;
print(n-1);
}
print(2)
n=2
void print(n)
{
if(n==0) return;
F
cout << n;
print(n-1);
}


main()
PC(5)
PC(4)
PC(3)
PC(2)
PC(1)
PC(6)

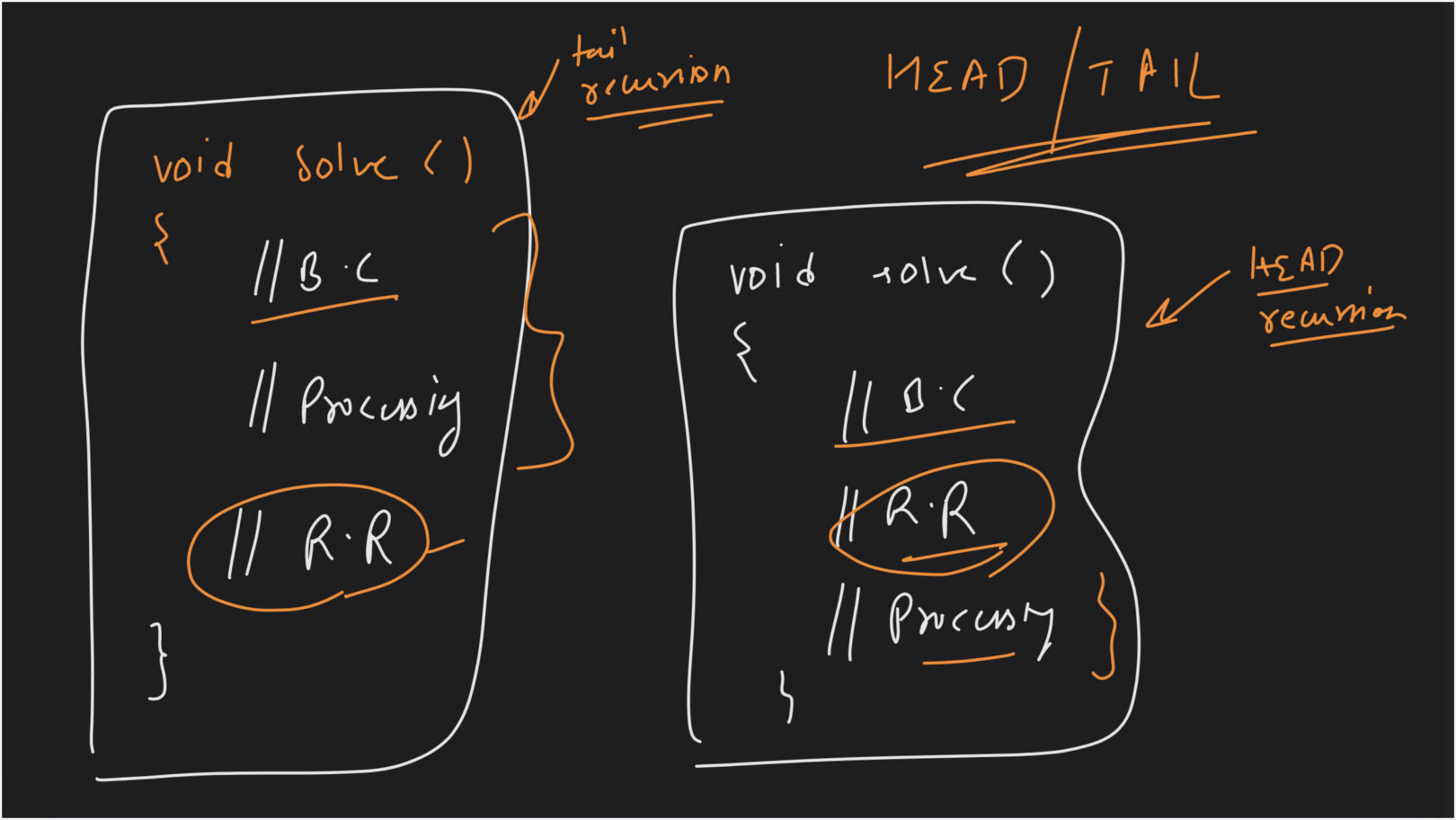
factorial

```
int fact (int n)
{
  // B.C
  if (n == 1)
      return 1;

  int ans = n * fact(n-1);

  return ans;
}
```

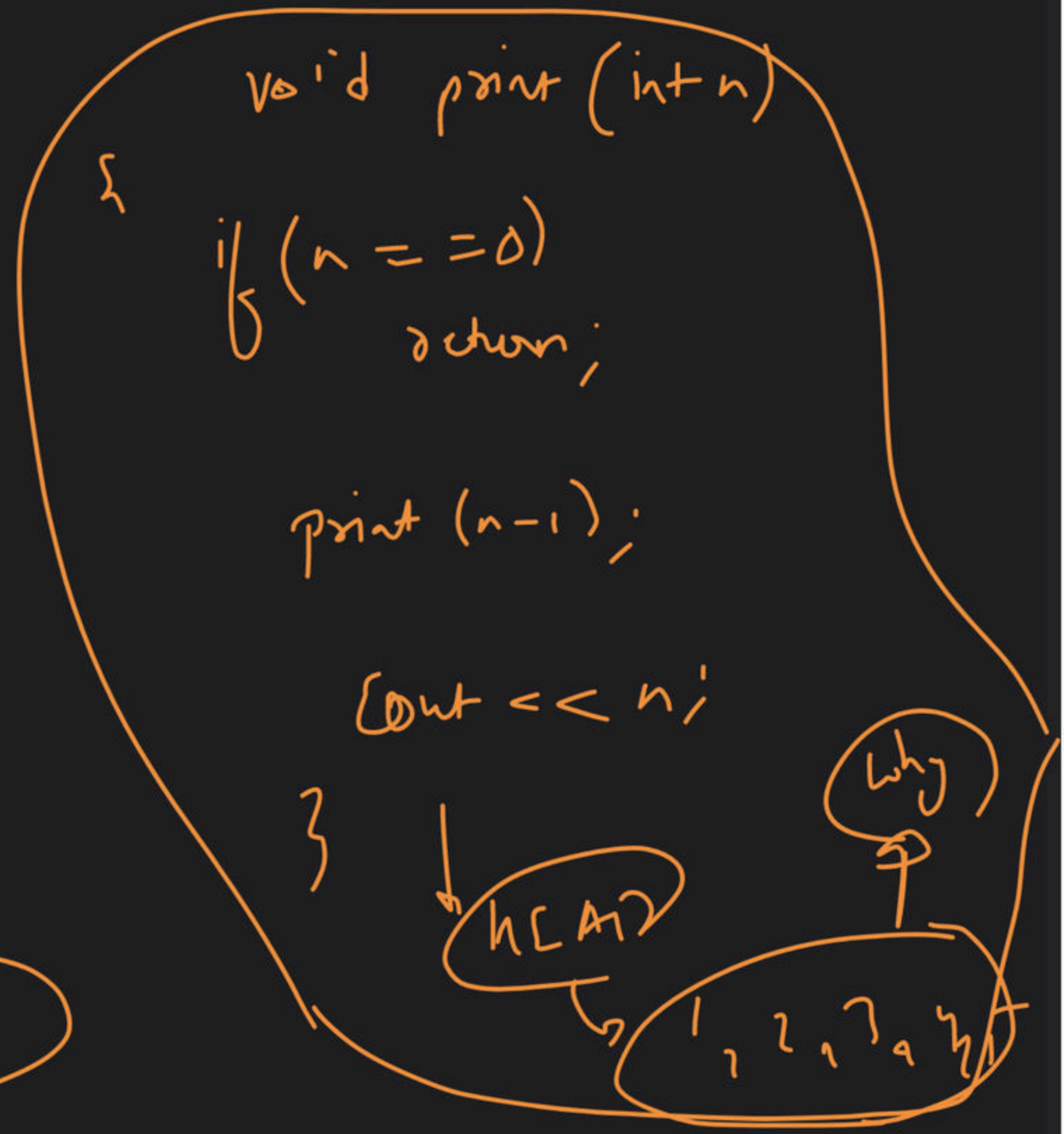

fact(5)
int fact (int n) n=5
{
if (n==1) return 1; F
int ans = n * fact(n-1);
5x
21
return ans;
}
fact(4)
int fact (int n) n=4
{
if (n==1) return 1; F
int ans = n * fact(n-1);
4x
6
return ans;
}
fact(3)
n=3
int fact (int n)
{
if (n==1) return 1; F
int ans = n * fact(n-1);
3x
2
return ans;
}
fact(2)
int fact (int n) n=2
{
if (n==1) return 1; F
int ans = n * fact(n-1);
2x
1
return ans;
}
fact(1)
int fact (int n) n=1
{
true
if (n==1) return 1;
int ans = n * fact(n-1);
return ans;
}
int fact (int n)
{
if(n==1) return 1;
int ans = n * fact(n-1);
return ans;
}

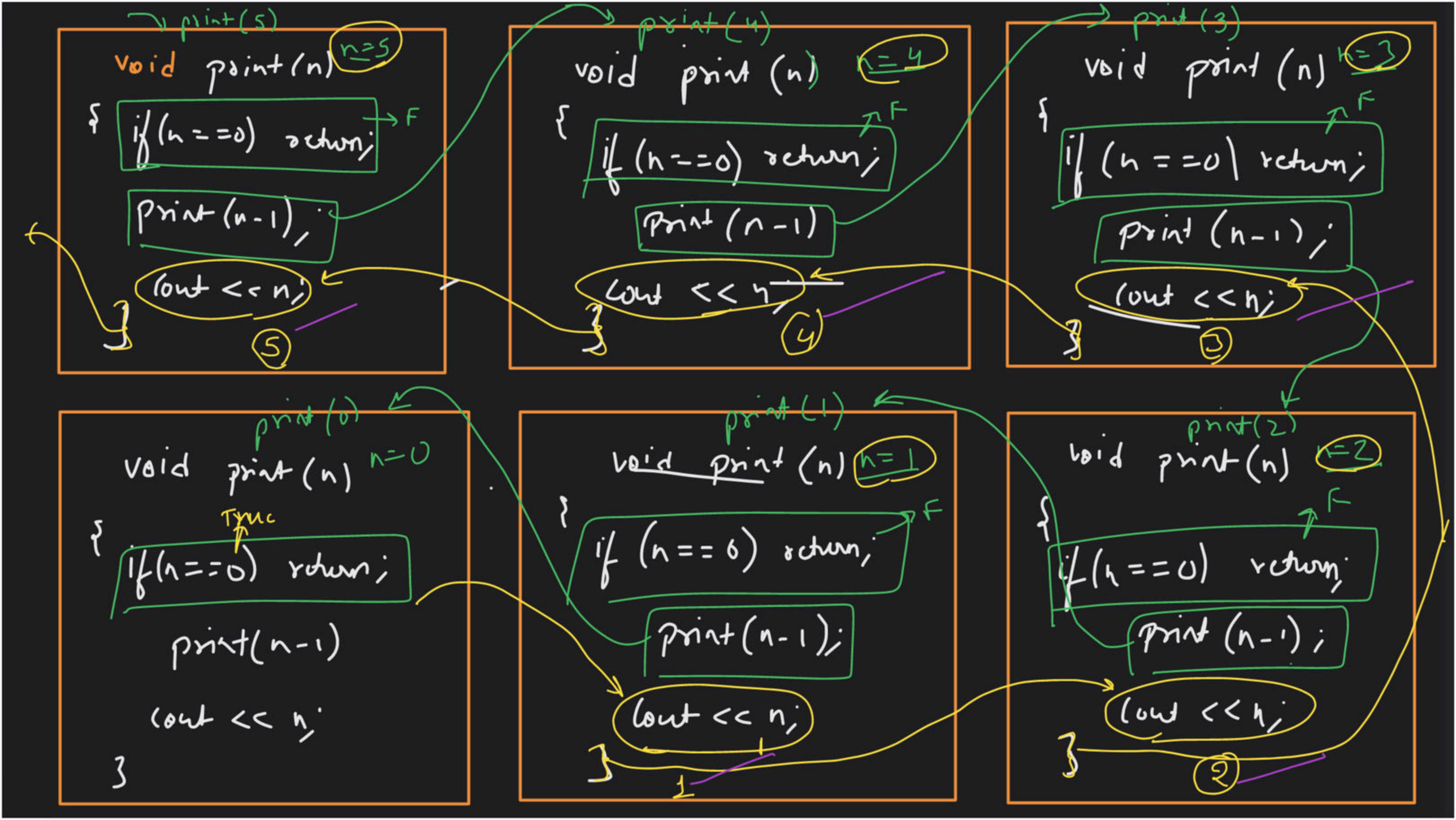


$$f(n) = n \times f(n-1)$$


tail recursion
HEAD / TAIL
void solve()
{
// B.C
// Processing
// R.R
}
void solve()
{
// B.C
// R.R
// Processing
}
HEAD recursion

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

    cout << n;

    print(n-1);
}
```



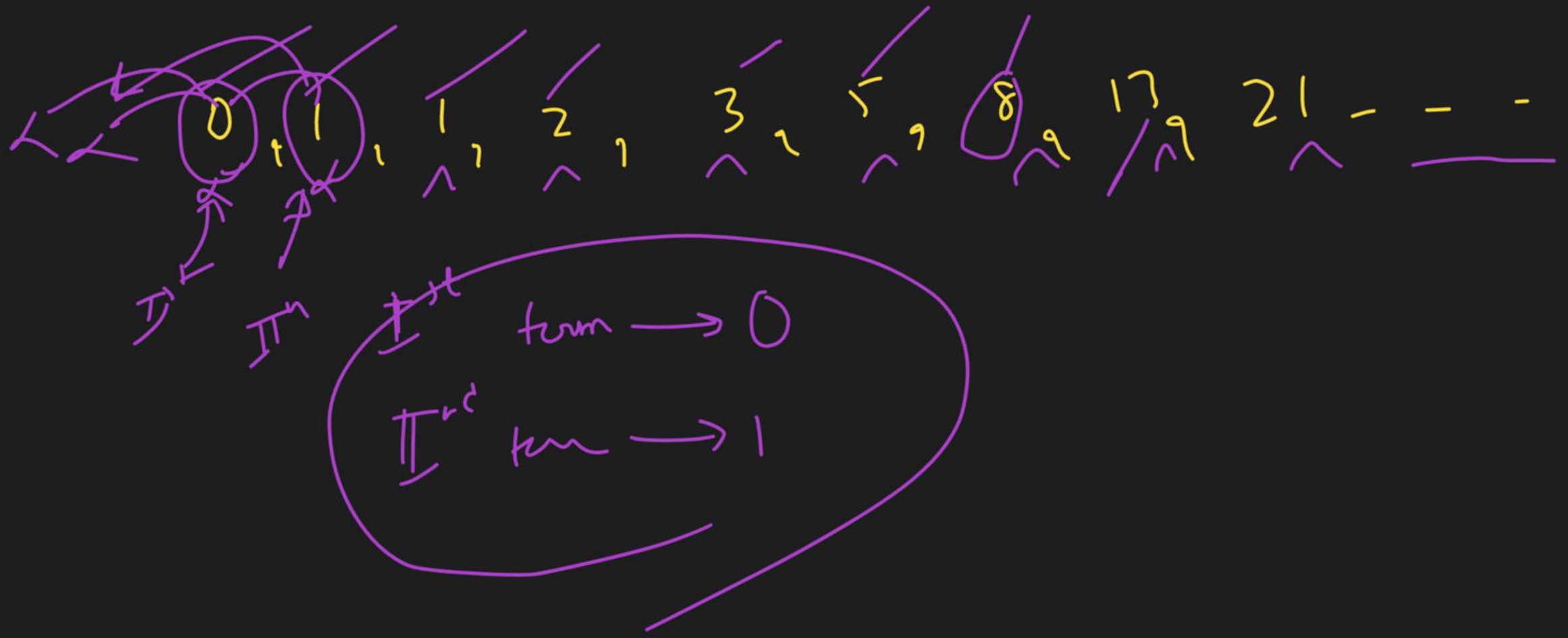

print(5)
void print(n)
n=5
{
if(n==0) return;
F
print(n-1);
cout << n;
5
print(4)
void print(n)
n=4
{
if(n==0) return;
F
print(n-1)
cout << n;
4
print(3)
void print(n)
n=3
{
if(n==0) return;
F
print(n-1);
cout << n;
3
print(0)
void print(n)
n=0
Time
{
if(n==0) return;
print(n-1)
cout << n;
}
print(1)
void print(n)
n=1
{
if(n==0) return;
F
print(n-1);
cout << n;
}
1
print(2)
void print(n)
n=2
{
if(n==0) return;
F
print(n-1);
cout << n;
}
2

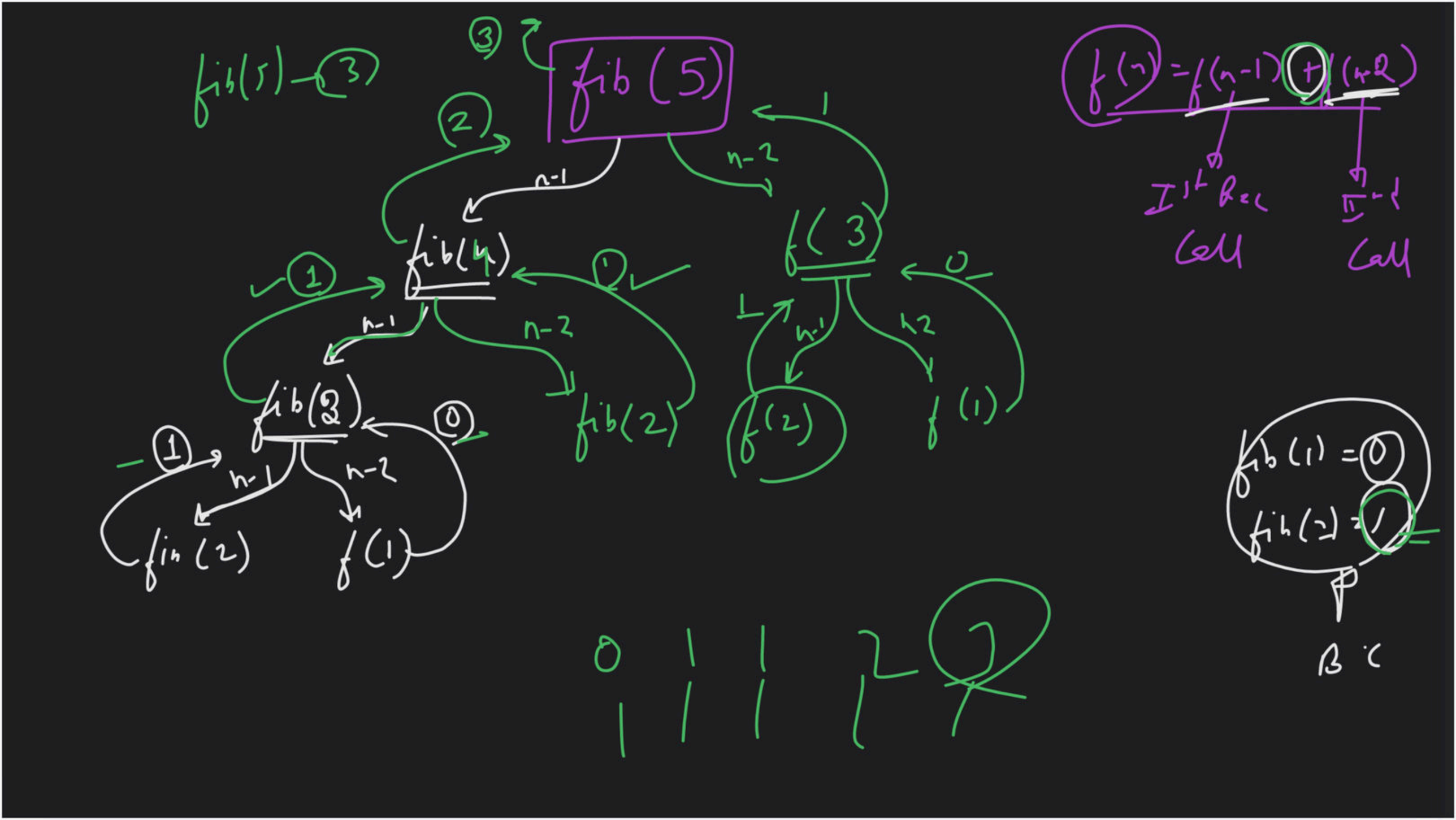
→ Fibonacci Series

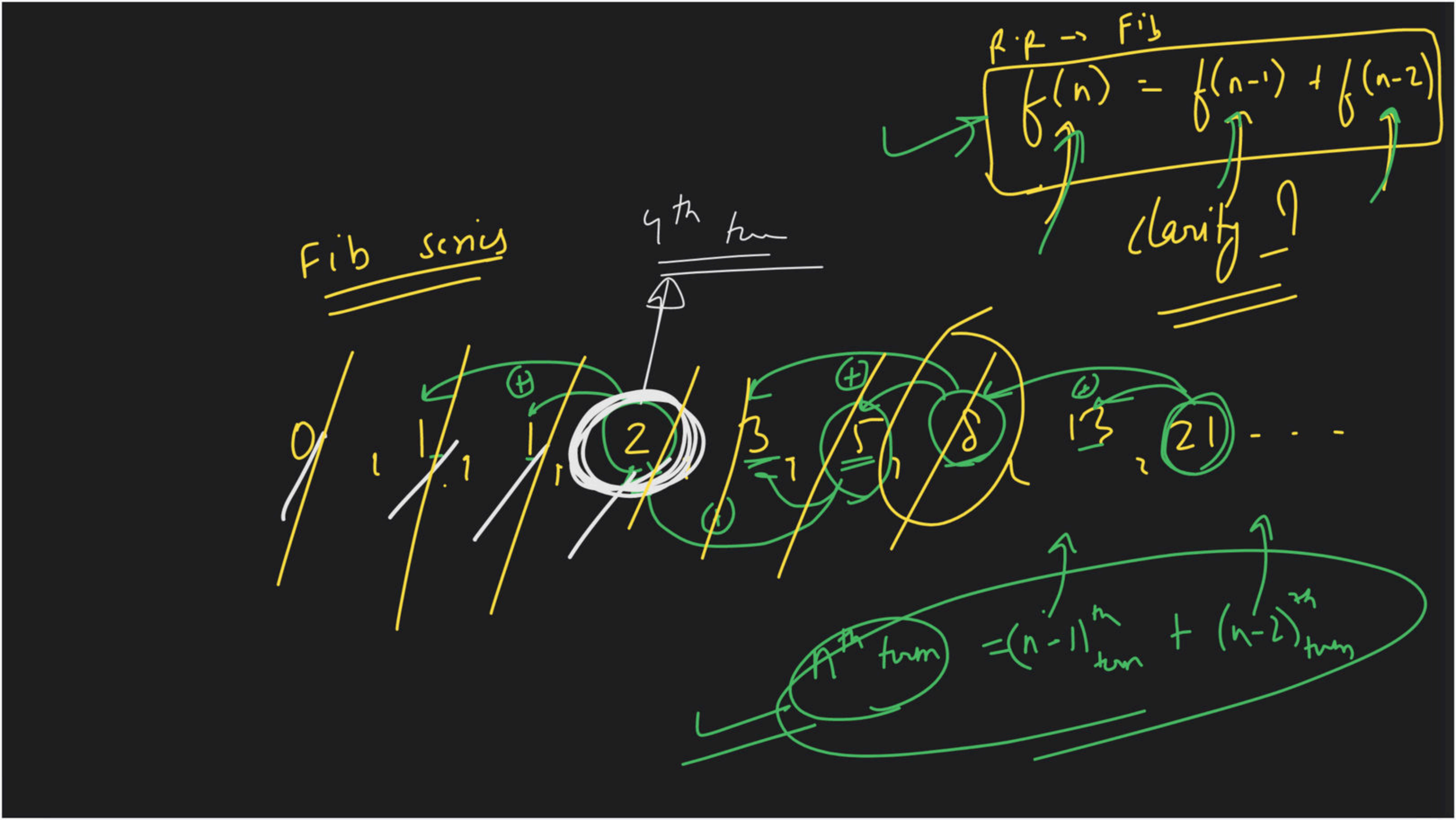
n^{th} term = $(n-1)^{th}$ term + $(n-2)^{th}$ term

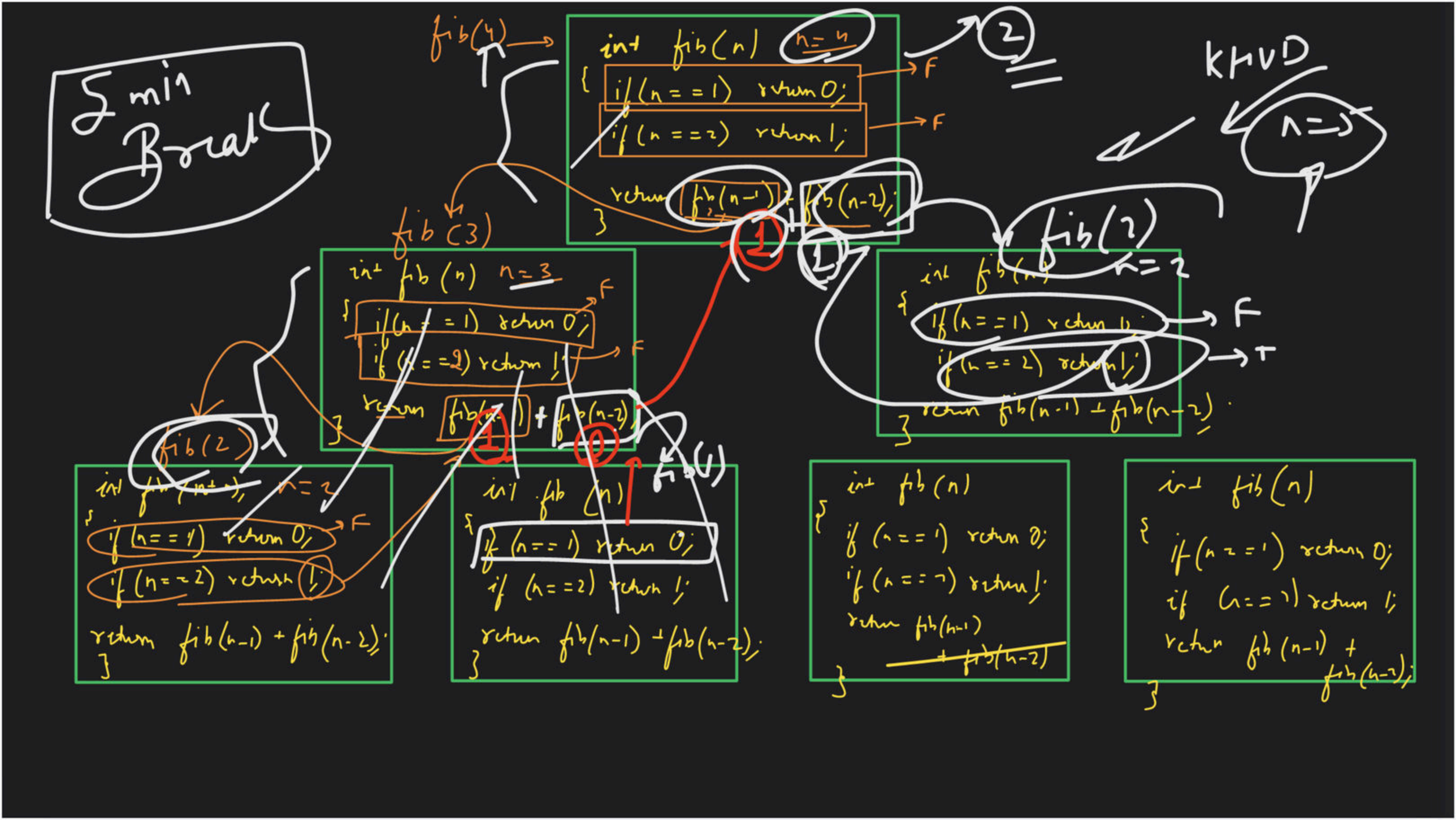
$$f(n) = f(n-1) + f(n-2)$$

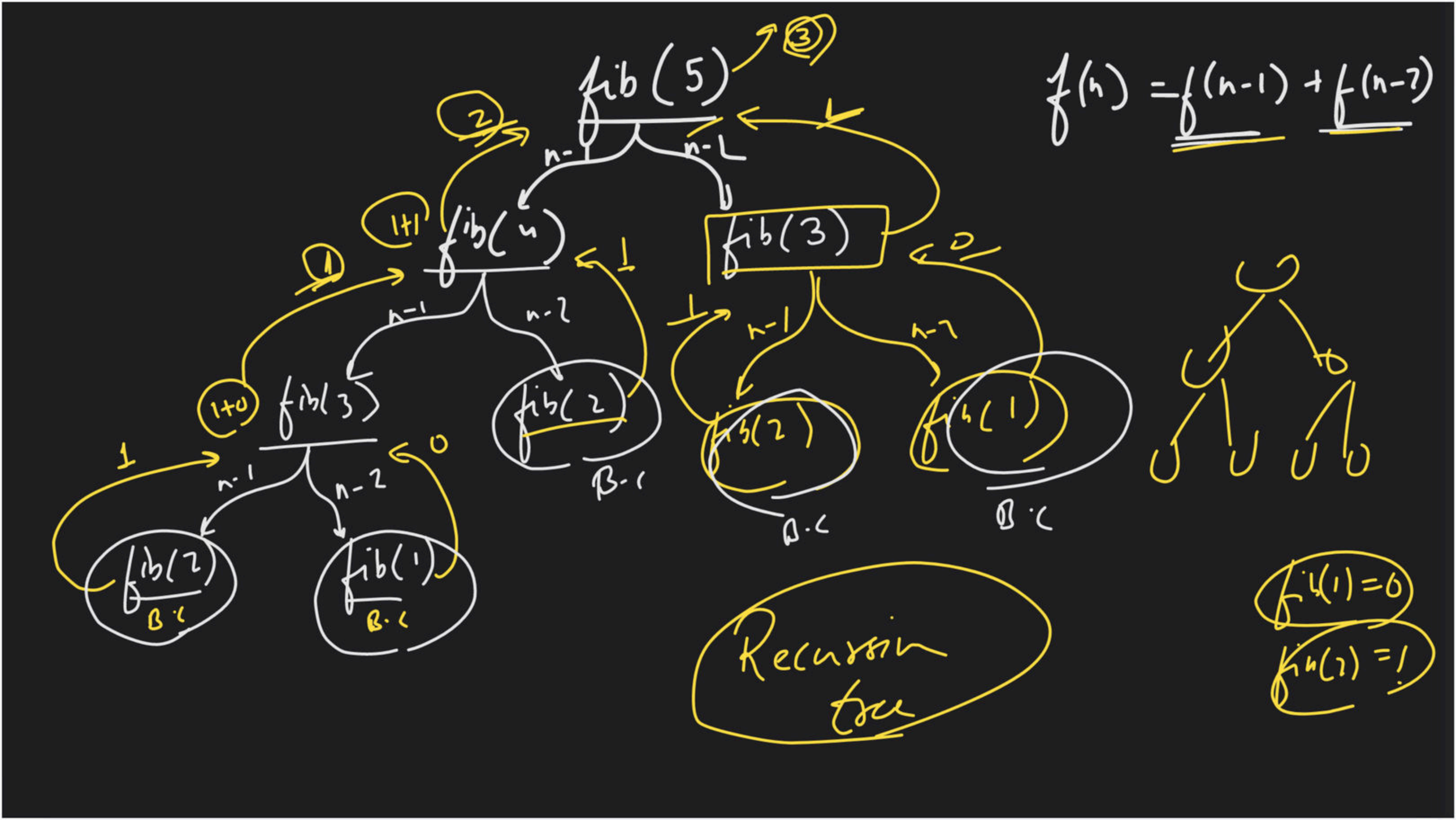
→ Base case → ?

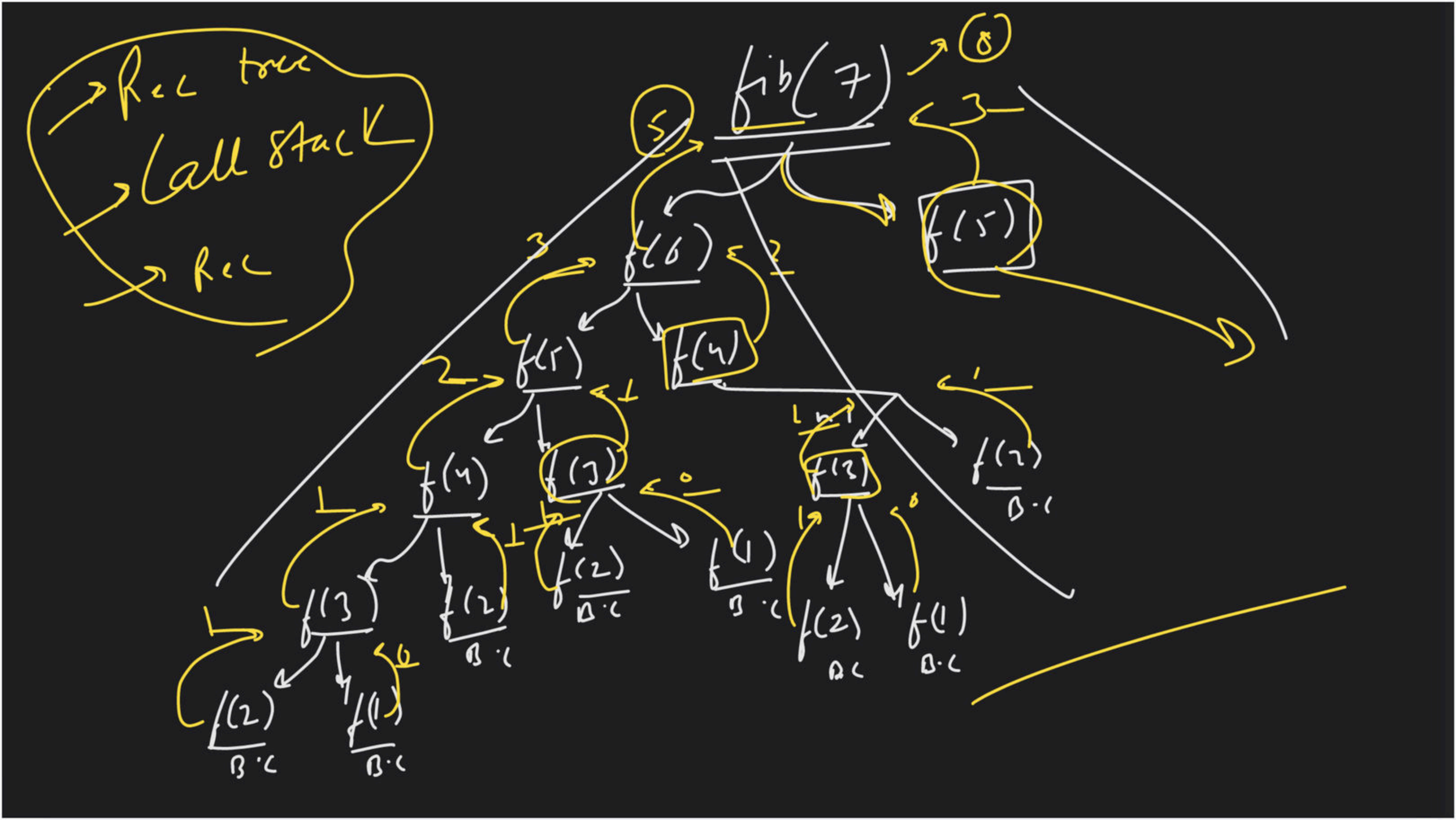

0
1
1
2
3
5
8
13
21
1st
term → 0
2nd
term → 1

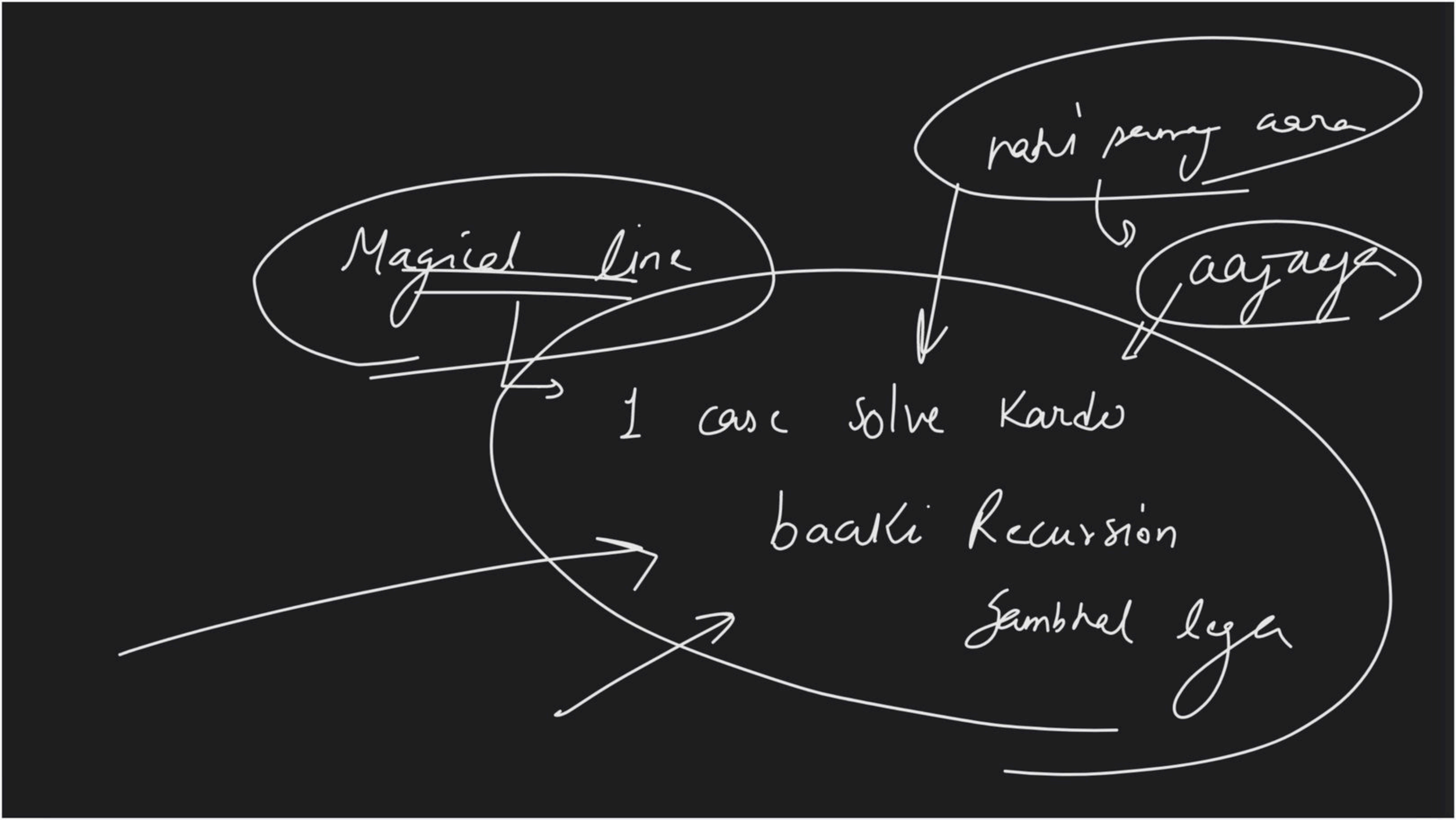

fib(5) → 3
3
fib (5)
2
n-1
n-2
1
fib(4)
fib(3)
1
1
0
n-1
n-2
fib(2)
fib(2)
fib(1)
1
0
n-1
n-2
fib(2)
fib(1)
f(n) = f(n-1) + f(n-2)
Ist Rec Call
IInd Rec Call
fib(1) = 0
fib(2) = 1
Base
0 1 1 2 3

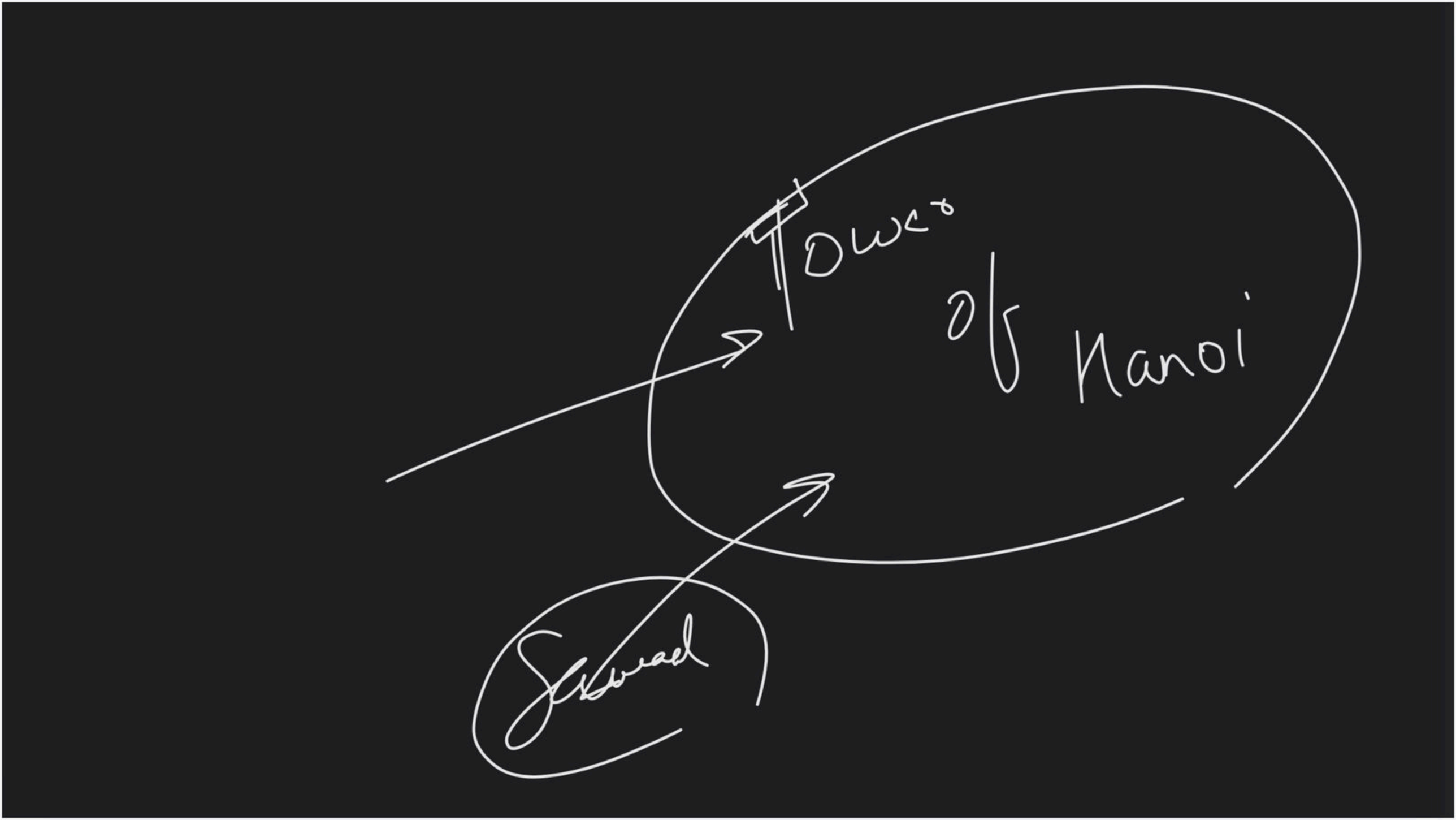

R·R → Fib
f(n) = f(n-1) + f(n-2)
Clarity ?
Fib series
4th term
0 1 1 2 3 5 8 13 21 . . .
nth term = (n-1)th term + (n-2)th term

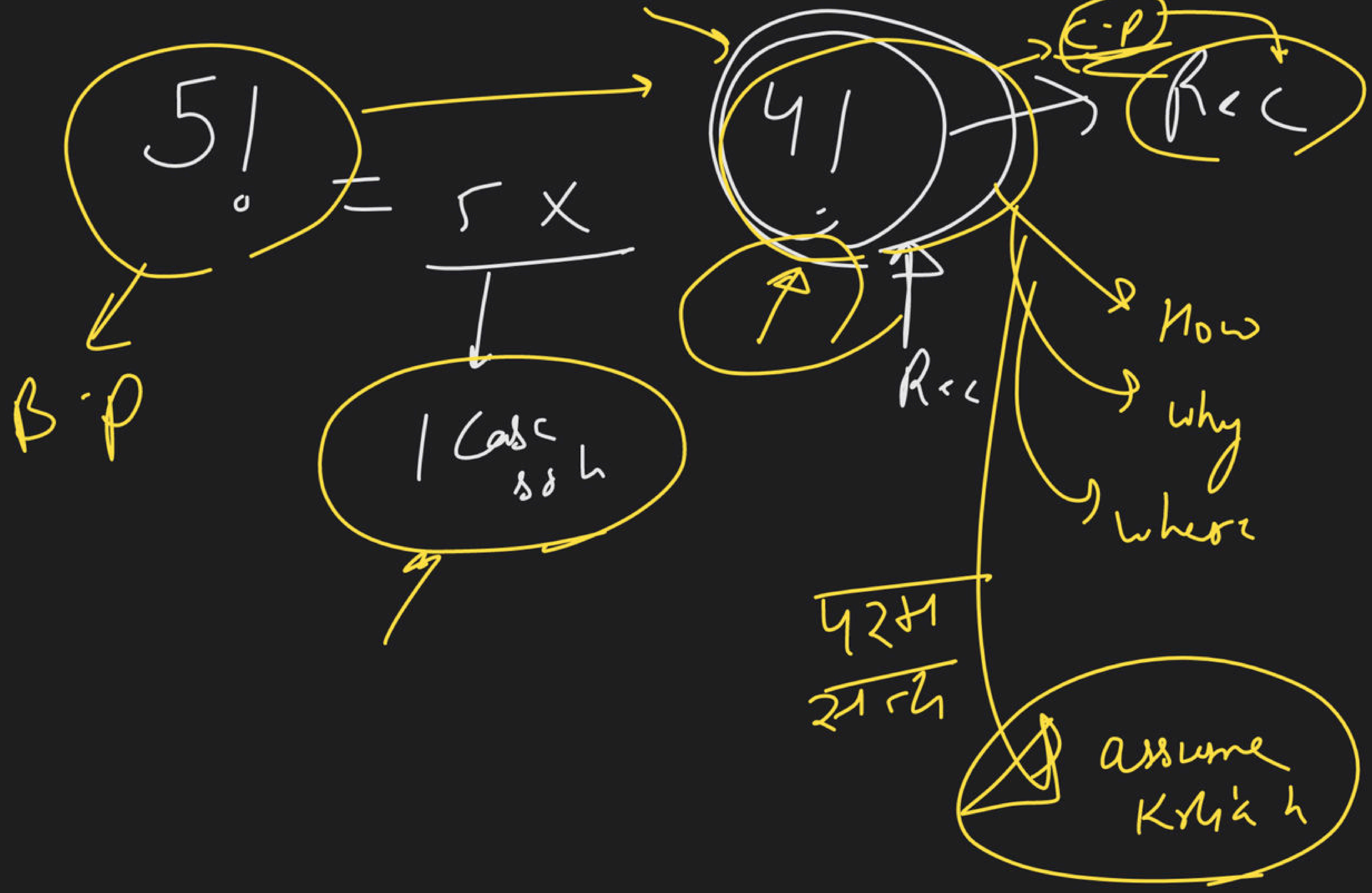

5 min Break
fib(4)
int fib(n) n=4
{
if(n==1) return 0;
if(n==2) return 1;
return fib(n-1) + fib(n-2);
}
F
F
2
fib(3)
int fib(n) n=3
{
if(n==1) return 0;
if(n==2) return 1;
return fib(n-1) + fib(n-2);
}
F
F
1
1
fib(2)
int fib(n) n=2
{
if(n==1) return 0;
if(n==2) return 1;
return fib(n-1) + fib(n-2);
}
F
1
int fib(n)
{
if(n==1) return 0;
if(n==2) return 1;
return fib(n-1) + fib(n-2);
}
fib(2)
int fib(n) n=2
{
if(n==1) return 0;
if(n==2) return 1;
return fib(n-1) + fib(n-2);
}
F
T
int fib(n)
{
if(n==1) return 0;
if(n==2) return 1;
return fib(n-1) + fib(n-2);
}
int fib(n)
{
if(n==1) return 0;
if(n==2) return 1;
return fib(n-1) + fib(n-2);
}
KHUD
n==1


fib(5)
fib(4)
fib(3)
fib(3)
fib(2)
fib(2)
fib(1)
fib(2)
fib(1)
n-1
n-2
1+1
1+0
Recursion tree
f(n) = f(n-1) + f(n-2)
fib(1) = 0
fib(2) = 1

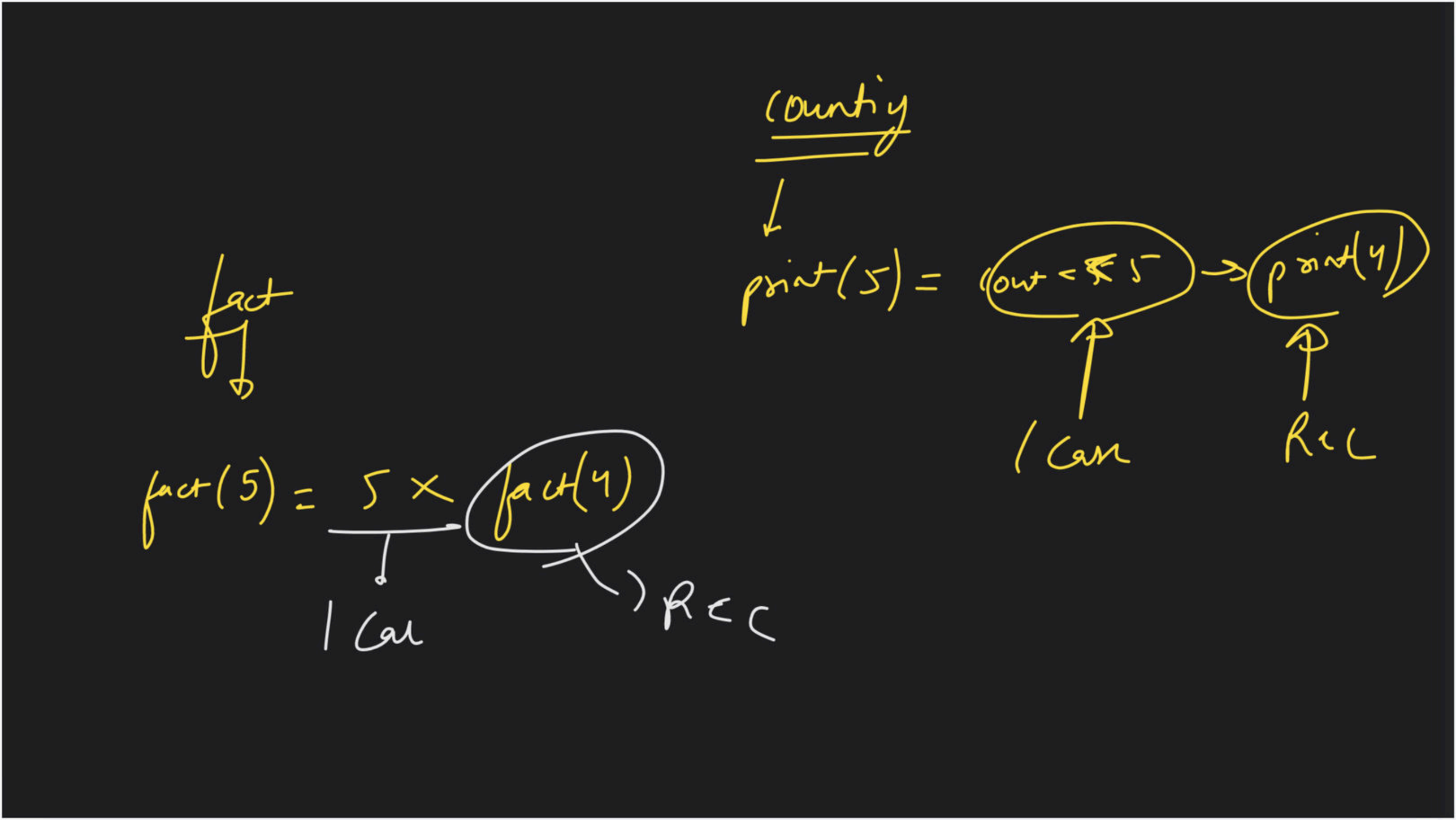

Rec tree
call stack
Rec
fib(7)
8
5
f(6)
f(5)
f(5)
f(4)
f(4)
f(3)
f(3)
f(2)
f(2)
f(1)
f(2)
f(1)
f(2)
f(1)
B.C
B.C
B.C
B.C
B.C
B.C
B.C


Magical line
1 case solve Kardo
baaki Recursion
Sambhal lega


Tower
of
Hanoi

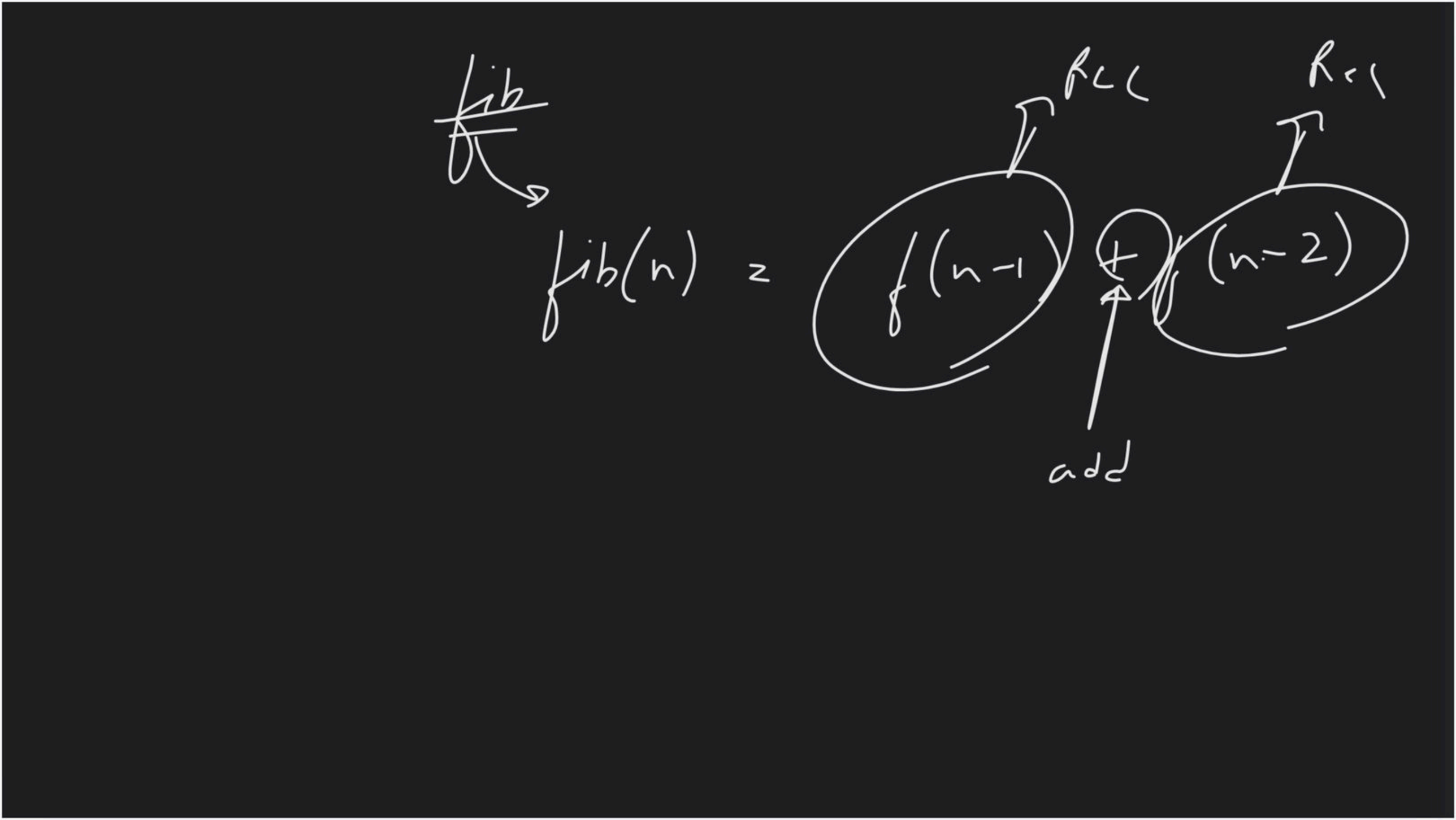


5!
= 5 x
B.P
1 case
4!
Rec
Rec
How
why
where
परम
assume


fact
fact(5) = 5 x fact(4)
1 Call
Rec
Countiy
print(5) =
Out << 5
print(4)
1 Call
Rec

$$2^n = 2 \times 2^{n-1}$$

$$f(n) = \underbrace{n \times}_{\text{1 call}} \overbrace{f(n-1)}^{\text{Rec}}$$


fib
fib(n) =
f(n-1)
+
f(n-2)
add