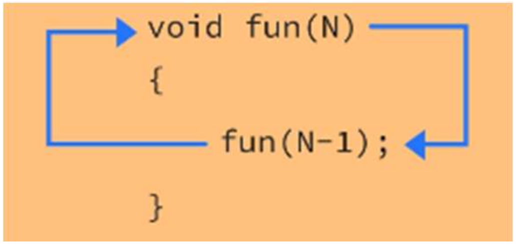
What is recursion

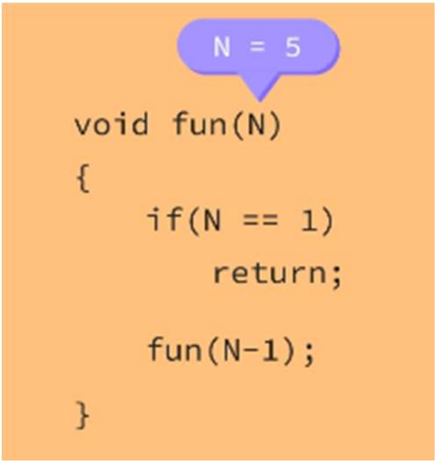
Recursion is the process for the function to call itself

Basic rules of recursion are

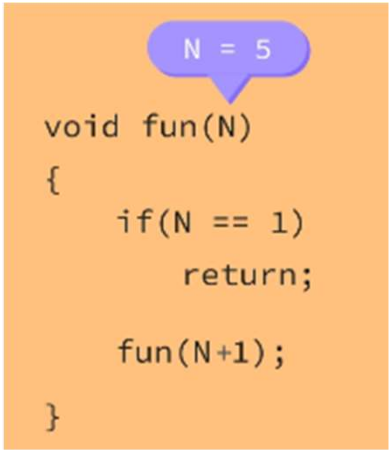
1. function should call itself
2. There should be a base case define where the function should stop calling itself
3. Recursive call should align towards base case in order to avoid infinite function calling.



```
void fun(N)
{
    fun(N-1);
}
```



```
void fun(N)
{
    if(N == 1)
        return;

    fun(N-1);
}
```



```
void fun(N)
{
    if(N == 1)
        return;

    fun(N+1);
}
```

How it work

Factorial using recursion

Factorial (N)= N * (N-1) * (N-2) * (N-3). 1

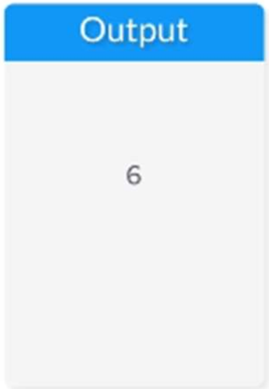
Recursive call : N * func(N-1)

Base case : N==1

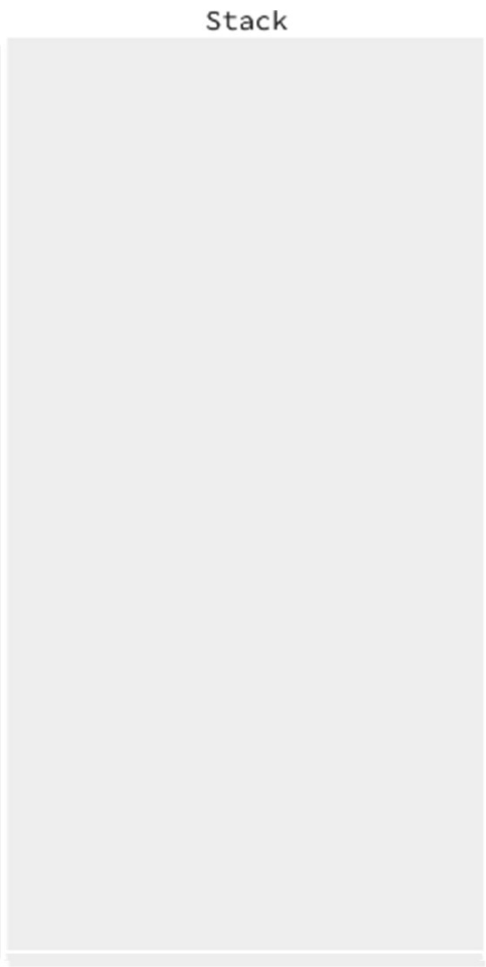
```
int factorial( int n ){
      if( n==1 ){
              return n;
      }
      return n * factorial ( n-1 );
}
```

How it work

```
int factorial(int n) {
    if (n == 1) {
        return n;
    }
    return n * factorial(n - 1);
}

public static void main(String[] args) {
    int n, ans;
    n=sc.nextInt();
    if (n >= 0) {
        ans = factorial(n);
        System.out.println(ans);
    }
}
```



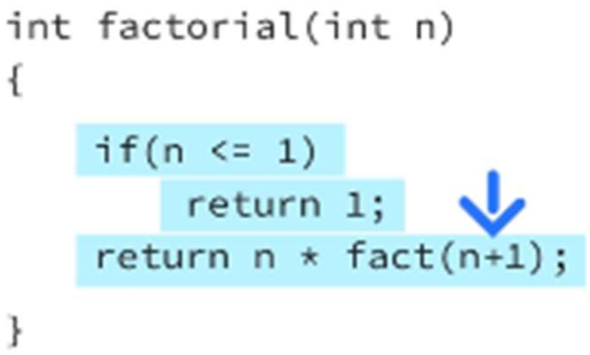
```
6
```



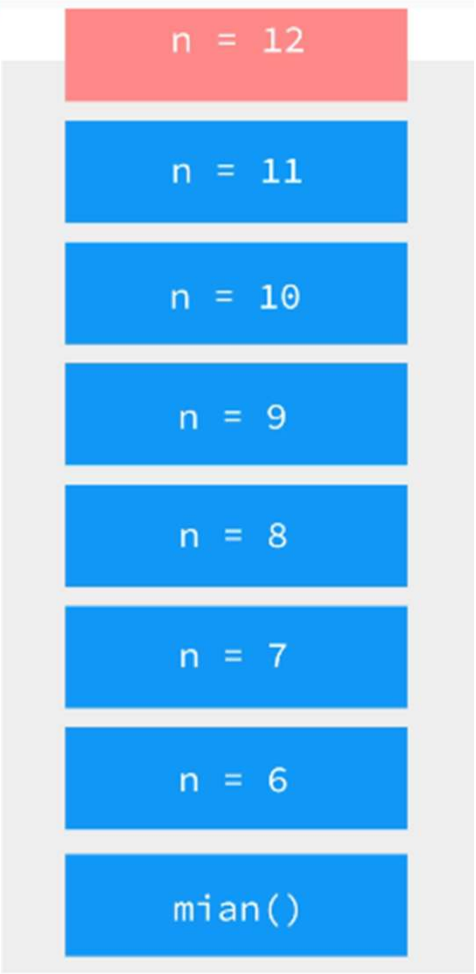
Stack Overflow

Reasons for stack overflow

1. Absence of Base Case
2. Recursive call doesn’t align towards the base case



```
int factorial(int n)
{
    if(n <= 1)
        return 1;
    return n * fact(n+1);
}
```



Recursion and Iteration

N

N, N-1, N-2, N-3, ... 1

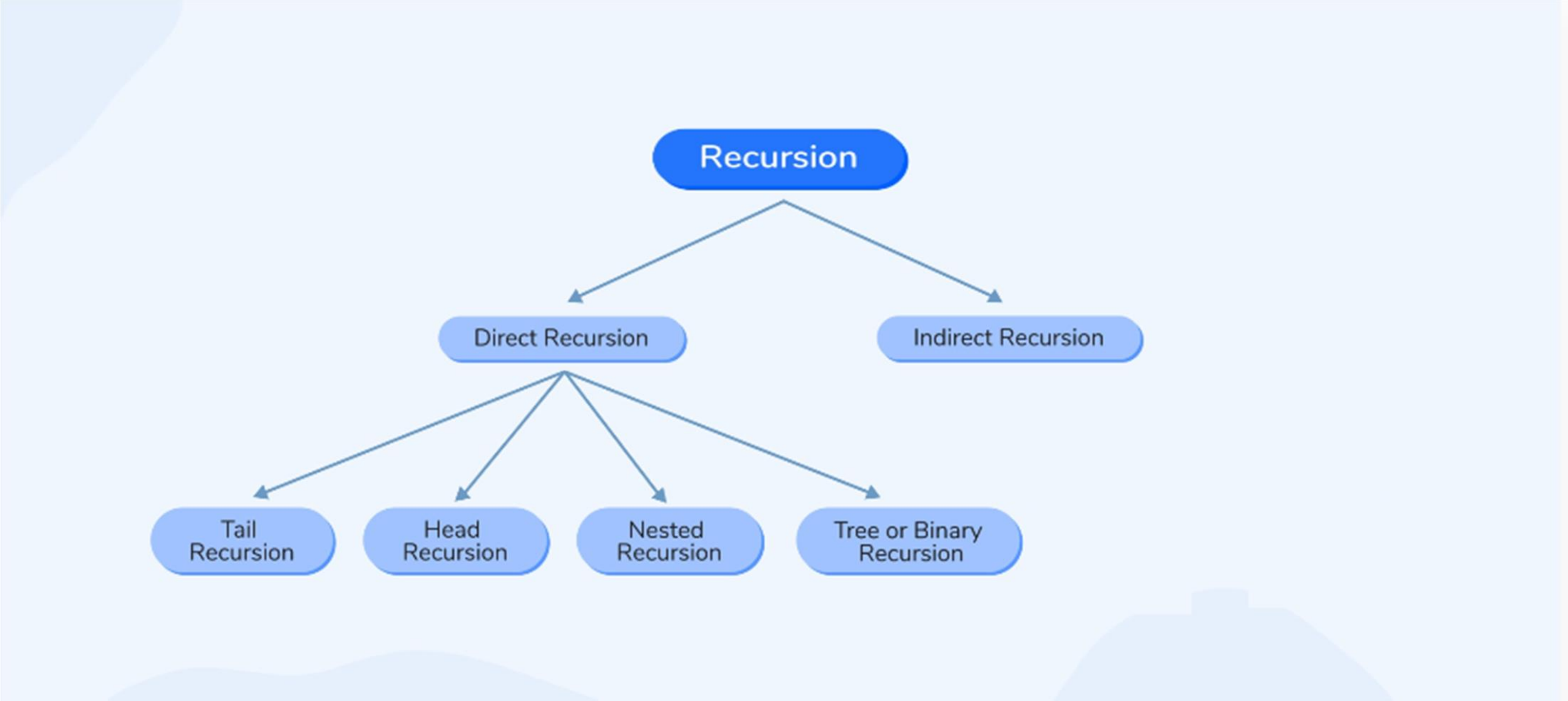
Iteration

```
void iteration(int n){
   while( n > 1 ){
       System.out.print( n );
       n - -;
    }
}
```

Recursion

```
void recursion(int n){
   if( n< 1 )
      return ;
    System.out.print( n );
    recursion( - - n );
}
```

Types of Recursion



Recursion

Direct Recursion

Indirect Recursion

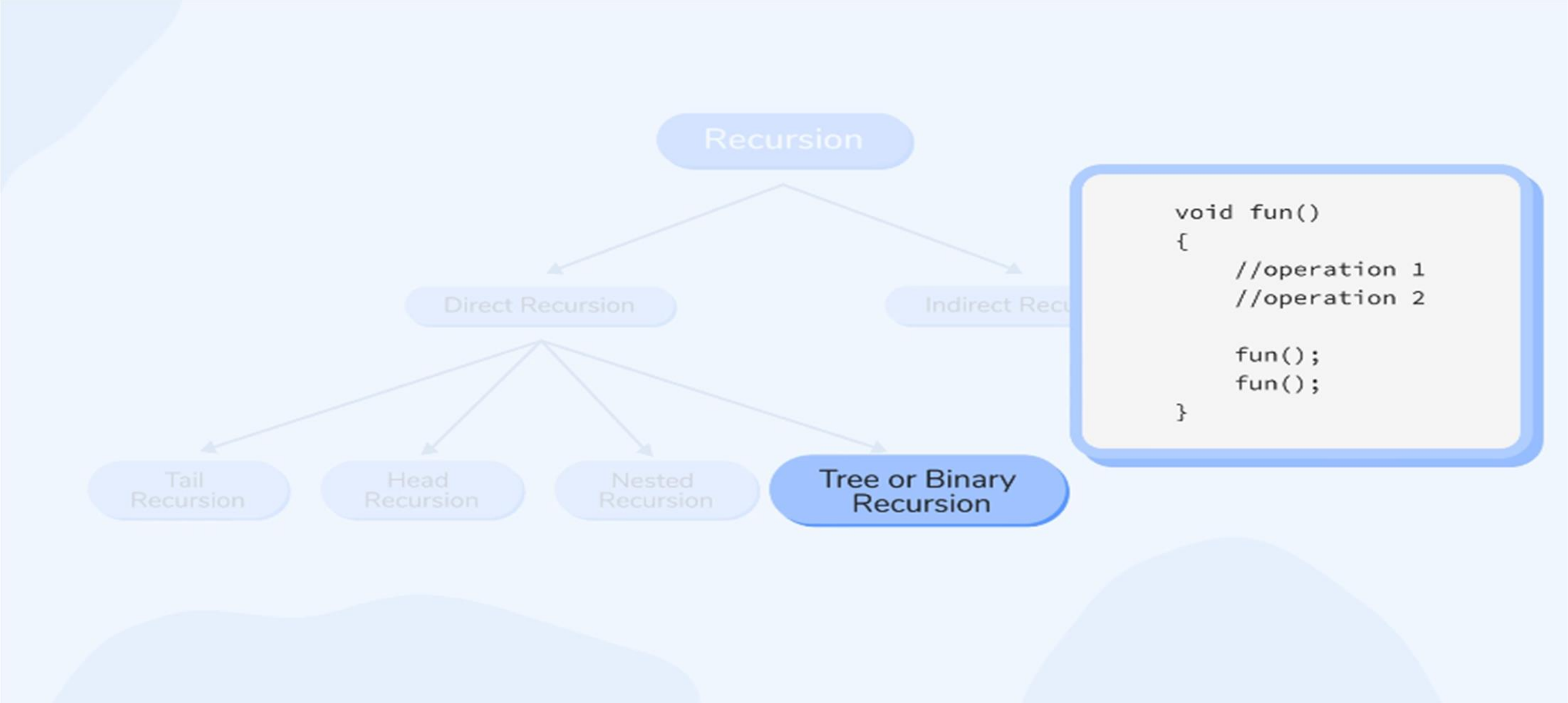
Tail Recursion

Head Recursion

Nested Recursion

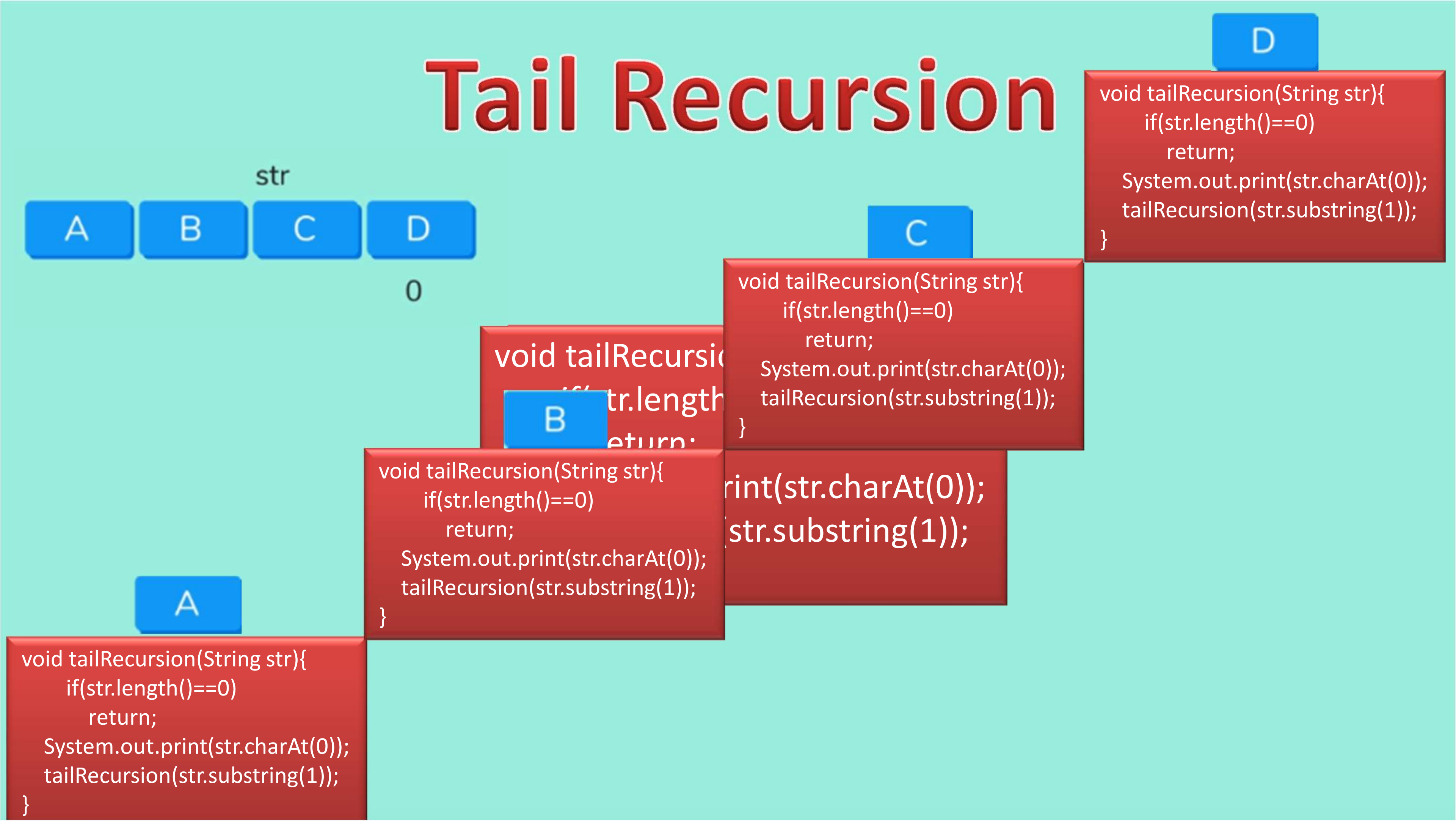
Tree or Binary Recursion

Types of Recursion



```
void fun()
{
    //operation 1
    //operation 2

    fun();
    fun();
}
```

Tail Recursion



```
void tailRecursion(String str){
        if(str.length()==0)
            return;
    System.out.print(str.charAt(0));
    tailRecursion(str.substring(1));
}
```

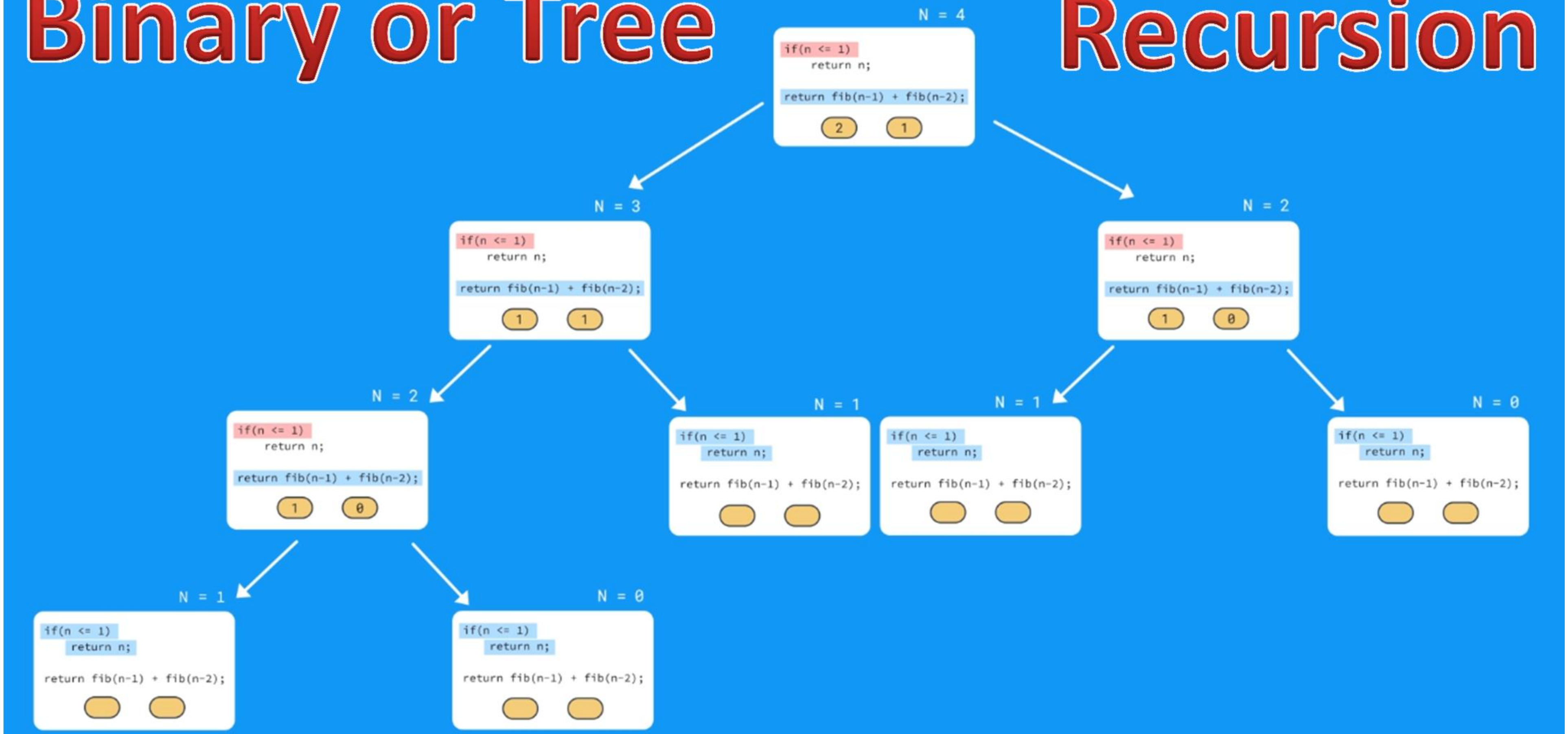
```
void tailRecursion(String str){
        if(str.length()==0)
            return;
    System.out.print(str.charAt(0));
    tailRecursion(str.substring(1));
}
```

```
void tailRecursi
        tr.length
         eturn;
rint(str.charAt(0));
(str.substring(1));
```

```
void tailRecursion(String str){
        if(str.length()==0)
            return;
    System.out.print(str.charAt(0));
    tailRecursion(str.substring(1));
}
```

```
void tailRecursion(String str){
        if(str.length()==0)
            return;
    System.out.print(str.charAt(0));
    tailRecursion(str.substring(1));
}
```

Binary or Tree Recursion



N = 4

```
if(n <= 1)
    return n;

return fib(n-1) + fib(n-2);
```

N = 3

```
if(n <= 1)
    return n;

return fib(n-1) + fib(n-2);
```

N = 2

```
if(n <= 1)
    return n;

return fib(n-1) + fib(n-2);
```

N = 2

```
if(n <= 1)
    return n;

return fib(n-1) + fib(n-2);
```

N = 1

```
if(n <= 1)
    return n;

return fib(n-1) + fib(n-2);
```

N = 1

```
if(n <= 1)
    return n;

return fib(n-1) + fib(n-2);
```

N = 0

```
if(n <= 1)
    return n;

return fib(n-1) + fib(n-2);
```

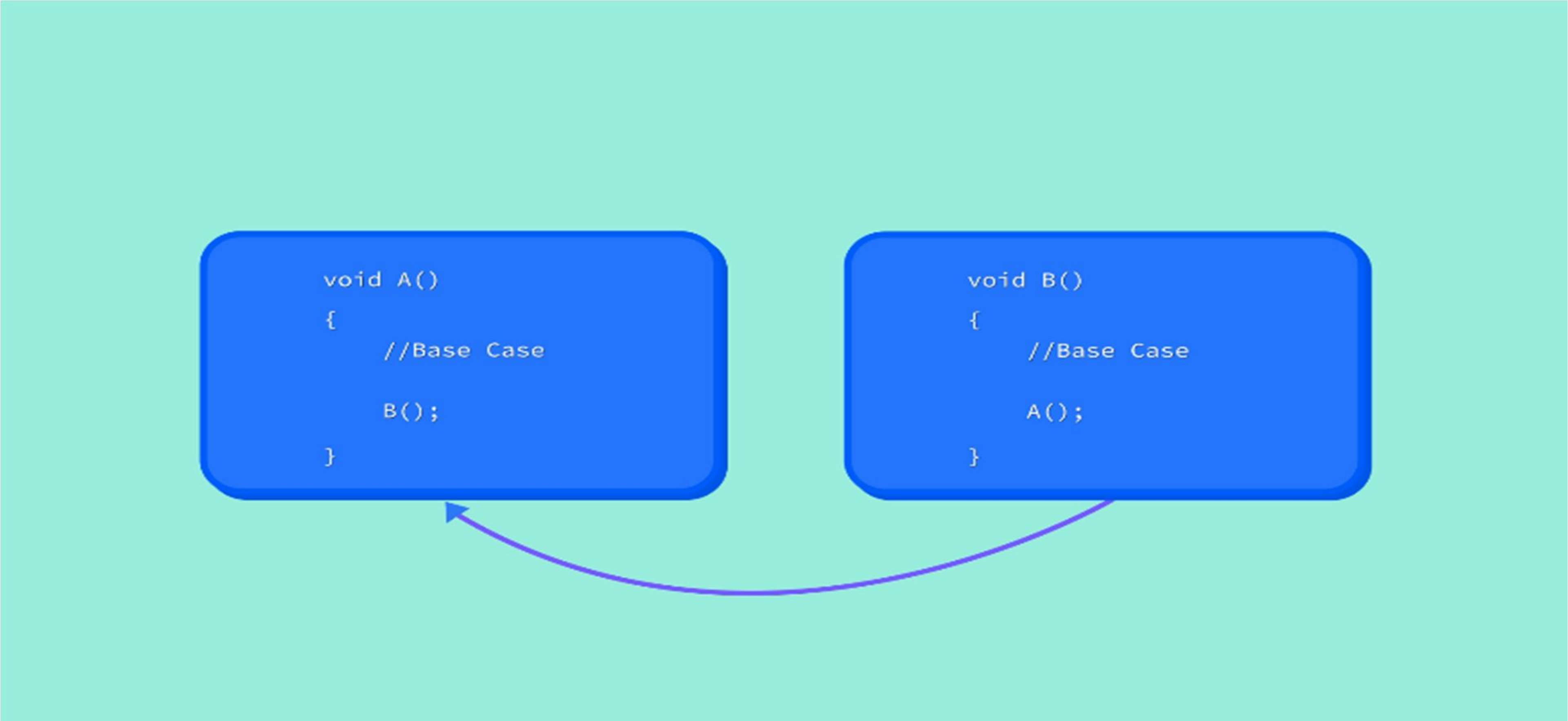
N = 1

```
if(n <= 1)
    return n;

return fib(n-1) + fib(n-2);
```

N = 0

```
if(n <= 1)
    return n;

return fib(n-1) + fib(n-2);
```

Indirect Recursion



```
void A()
{
    //Base Case

    B();

}
```

```
void B()
{
    //Base Case

    A();

}
```

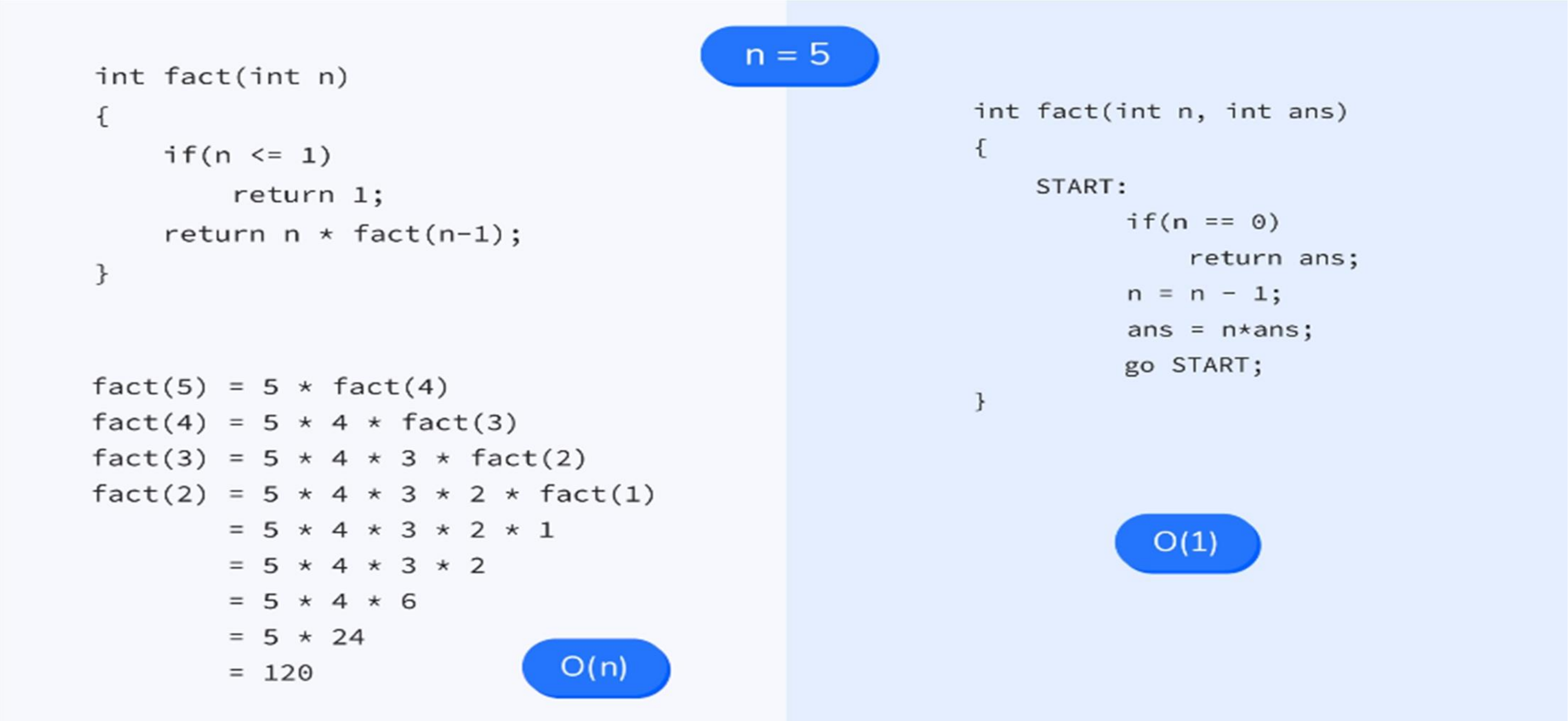
Optimization of Tail Recursion

```
void show(int n){
   If(n<1)
      return;
   System.out.println(n);
   show(--n);
}
```

Optimization

```
void show(int n){
START:
   If(n<1)
      return;
   System.out.println(n);
   n=n-1;
goto START;
}
```

Why Tail Recursion is efficient



```
int fact(int n)
{
    if(n <= 1)
        return 1;
    return n * fact(n-1);
}


fact(5) = 5 * fact(4)
fact(4) = 5 * 4 * fact(3)
fact(3) = 5 * 4 * 3 * fact(2)
fact(2) = 5 * 4 * 3 * 2 * fact(1)
        = 5 * 4 * 3 * 2 * 1
        = 5 * 4 * 3 * 2
        = 5 * 4 * 6
        = 5 * 24
        = 120
```

```
int fact(int n, int ans)
{
    START:
         if(n == 0)
             return ans;
         n = n - 1;
         ans = n*ans;
         go START;
}
```