RECURSION

RECURSIVE FUNCTIONS

Recursion

- A method/procedure where the solution to a problem depends on solutions to smaller instances of the same problem
- So we break the task into smaller subtasks
- The approach can be applied to many types of problems and recursion is one of the central ideas of computer science
- We have to define base cases in order to avoid infinite loops
- We can solve problems with recursion or with iteration

We want to add the first **N** numbers:

Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int iterationSum(int N){

   int result = 0;

   for(int i=1;i<N;++i){
            result = result + i;
   }

   return result;
}
```

ITERATION

```
public int recursionSum(int N){

   if( N == 1 ) return 1;

   return N + recursionSum(N-1);
}
```

RECURSION

Head VS tail recursion

- If the recursive call occurs at the end of a method → it is called a tail recursion
- The tail recursion is similar to a loop
- The method executes all the statements before jumping into the next recursive call
- If the recursive call occurs at the beginning of a method, it is called a head recursion.
- The method saves the state before jumping into the next recursive call

```
public void tail(int N) {
  if( N == 1 ) return;
  System.out.println(N);
  tail(N-1);
}
```

TAIL RECURSION

```
public void head(int N) {
  if( N == 1 ) return;
  head(N-1);
  System.out.println(N);
}
```

HEAD RECURSION

Stack with recursion

- We have to track during recursion who called the given method and what arguments are to be handed over
- **AND WE HAVE TO TRACK THE PENDING CALLS !!!**
- We just need a single stack data structure: the operating system does everything for us
- These important information are to be pushed to the stack
- Values are popped from the stack

We want to add the first **N** numbers:

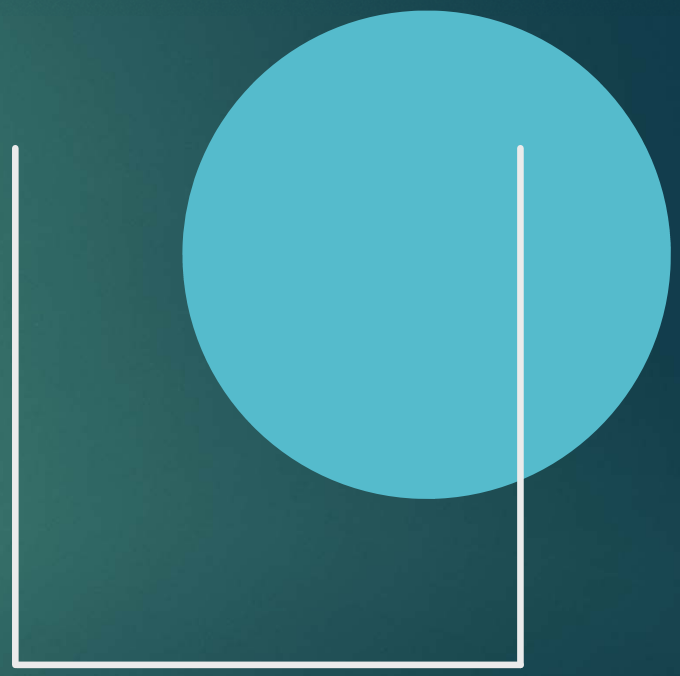
Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int recursionSum(int N){

  if( N == 1 ) return 1;

  return N + recursionSum(N-1);
}
```

RECURSION



STACK

We want to add the first **N** numbers:

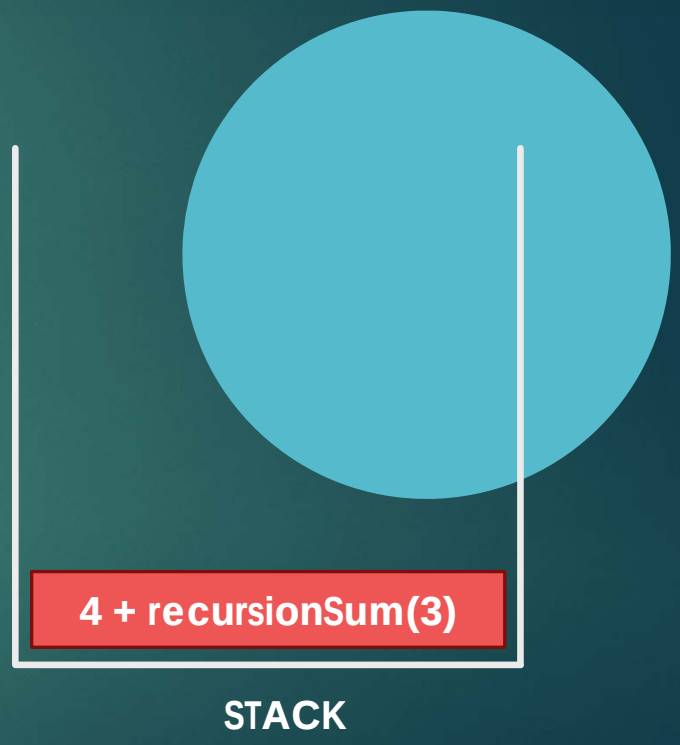
Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int recursionSum(int N){

  if( N == 1 ) return 1;

  return N + recursionSum(N-1);
}
```

RECURSION



We want to add the first **N** numbers:

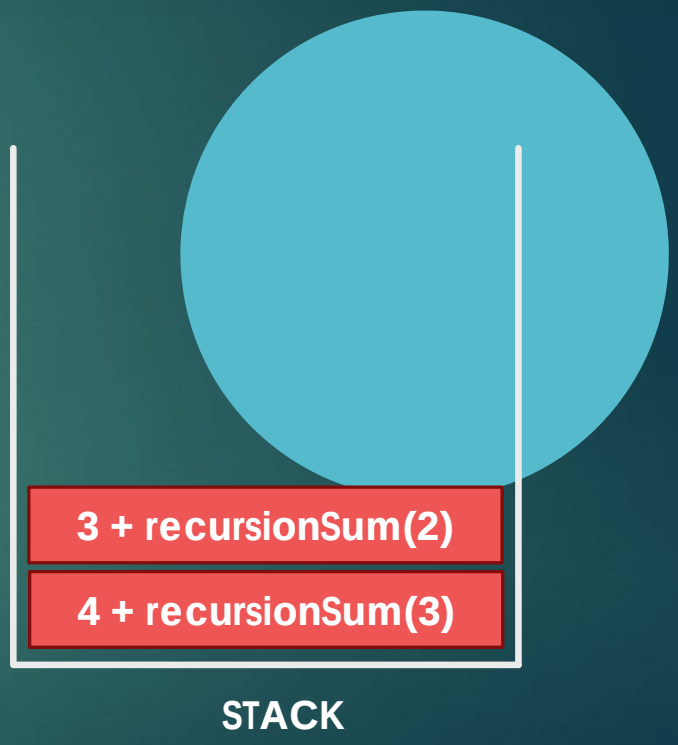
Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int recursionSum(int N){

  if( N == 1 ) return 1;

  return N + recursionSum(N-1);
}
```

RECURSION



We want to add the first **N** numbers:

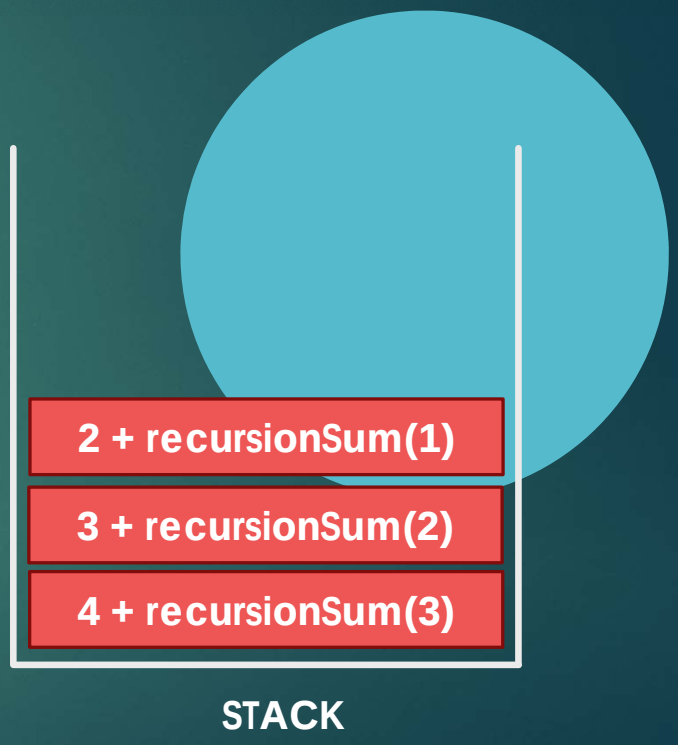
Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int recursionSum(int N){

  if( N == 1 ) return 1;

  return N + recursionSum(N-1);
}
```

RECURSION



We want to add the first **N** numbers:

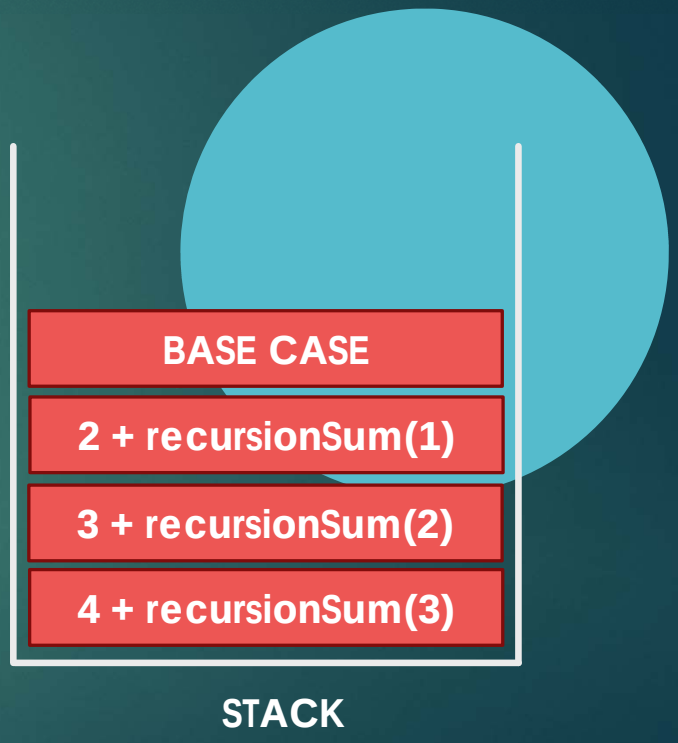
Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int recursionSum(int N){

  if( N == 1 ) return 1;

  return N + recursionSum(N-1);
}
```

RECURSION



We want to add the first **N** numbers:

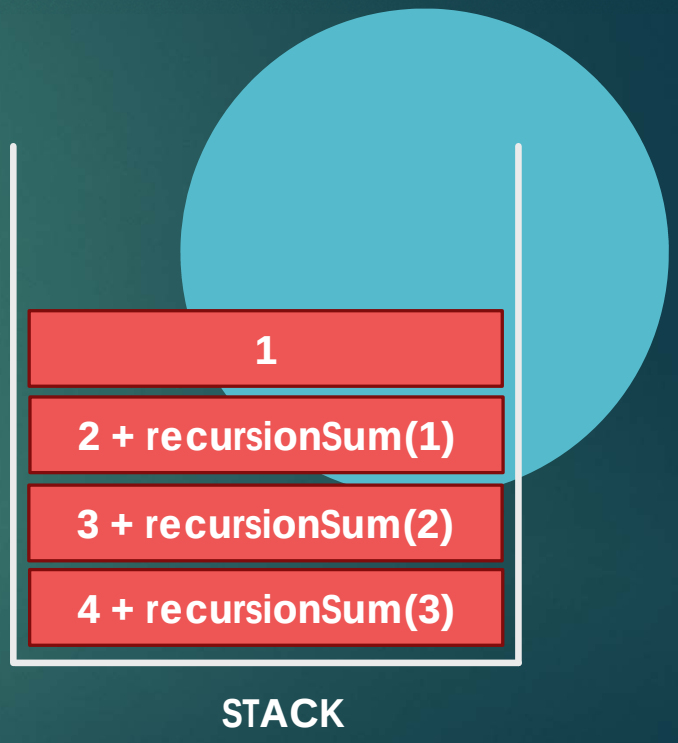
Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int recursionSum(int N){

  if( N == 1 ) return 1;

  return N + recursionSum(N-1);
}
```

RECURSION



We want to add the first **N** numbers:

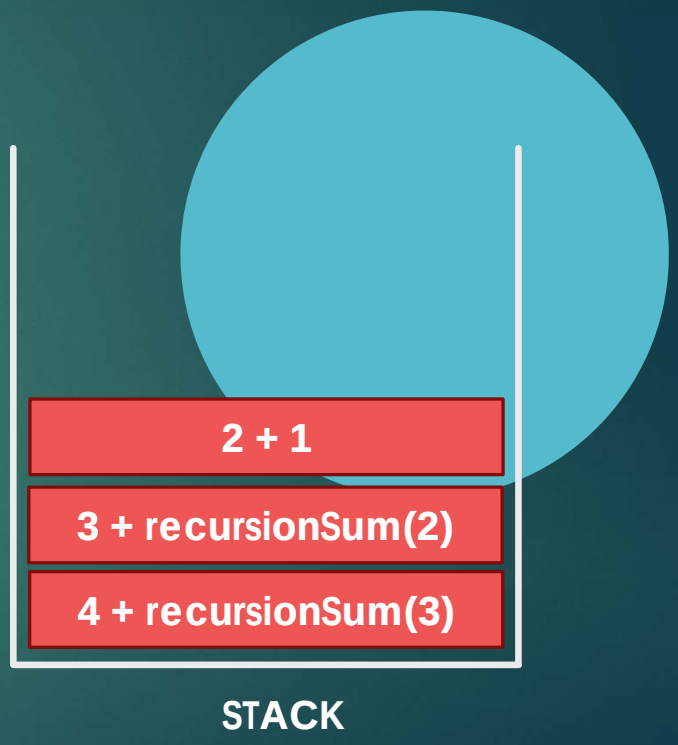
Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int recursionSum(int N){

  if( N == 1 ) return 1;

  return N + recursionSum(N-1);
}
```

RECURSION



We want to add the first **N** numbers:

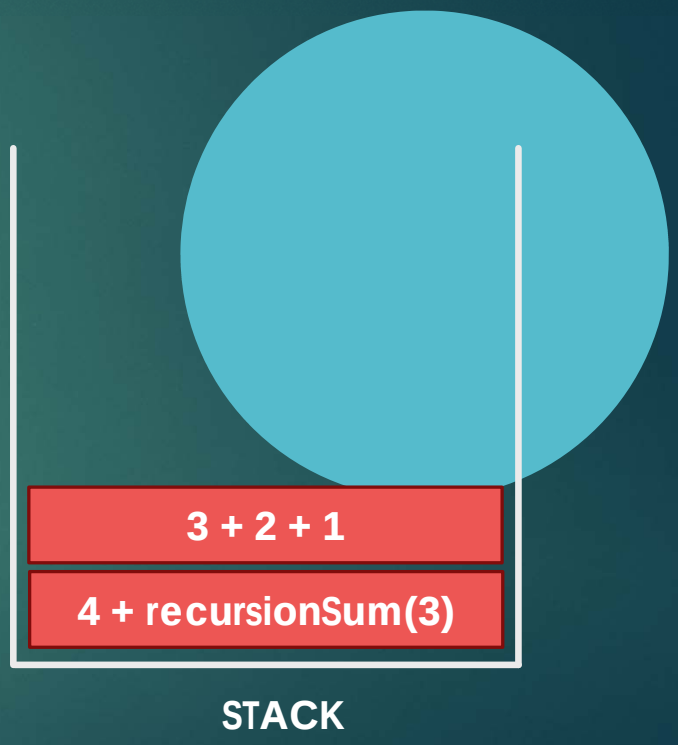
Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int recursionSum(int N){

  if( N == 1 ) return 1;

  return N + recursionSum(N-1);
}
```

RECURSION



We want to add the first **N** numbers:

Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int recursionSum(int N){

  if( N == 1 ) return 1;

  return N + recursionSum(N-1);
}
```

RECURSION

When we used **recursionSum(int N)** method:

```
recursionSum(4)
   recursionSum(3)
      recursionSum(2)
         recursionSum(1)
            return 1
         return 2+1
      return 3+2+1
  return 4+3+2+1
```

So these method calls and values are stored on the stack

Comparing recursive implementation against iterative implementation → recursion is at least twice slower because first we unfold recursive calls (pushing them on a stack) until we reach the base case and then we traverse the stack and retrieve all recursive calls

We want to calculate the factorial for **N**

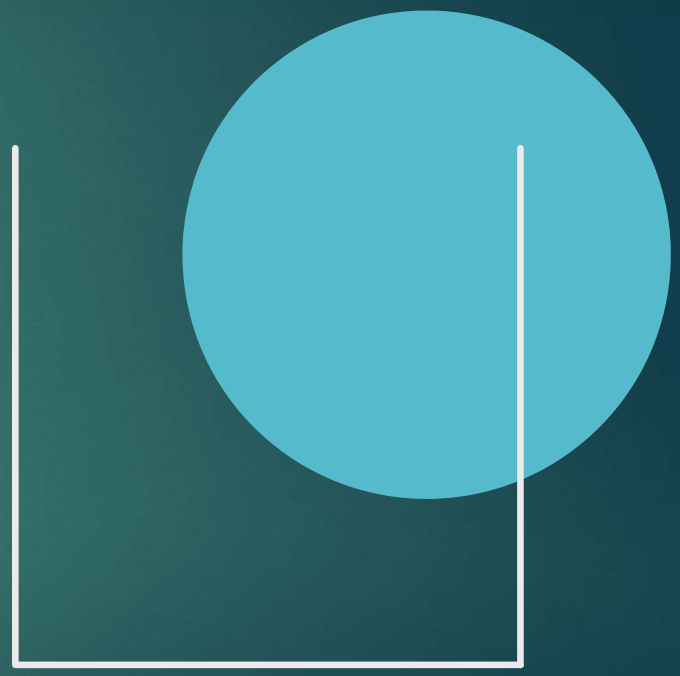
Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int factorial(int N){

  if( N == 1 ) return 1;

  return N * factorial(N-1);
}
```

RECURSION



STACK

We want to calculate the factorial for **N**

Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int factorial(int N){

  if( N == 1 ) return 1;

  return N * factorial(N-1);
}
```

RECURSION

We want to calculate the factorial for **N**

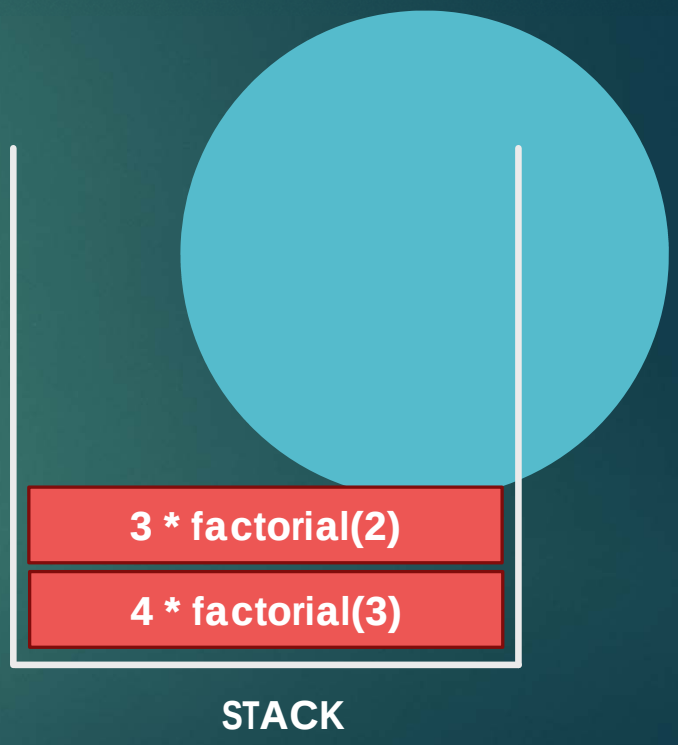
Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int factorial(int N){

  if( N == 1 ) return 1;

  return N * factorial(N-1);
}
```

RECURSION



We want to calculate the factorial for **N**

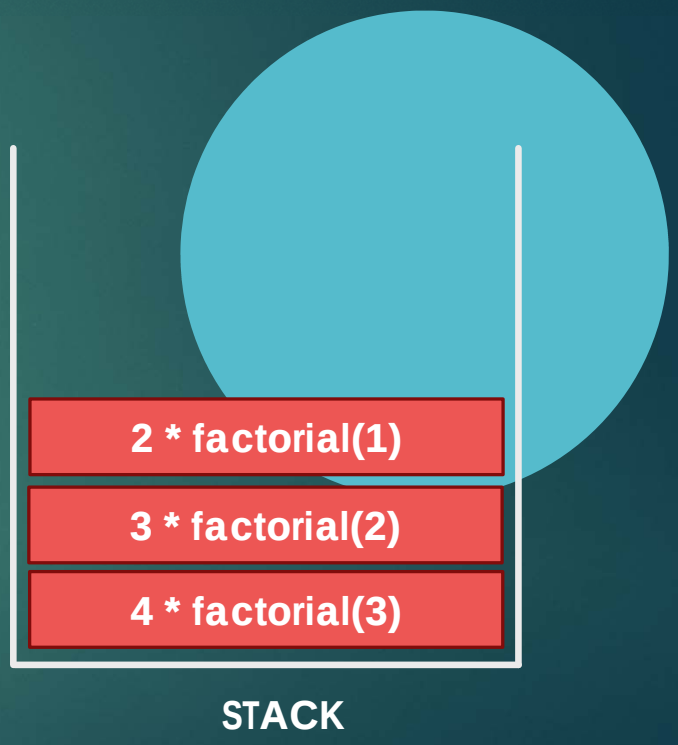
Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int factorial(int N){

  if( N == 1 ) return 1;

  return N * factorial(N-1);
}
```

RECURSION



We want to calculate the factorial for **N**

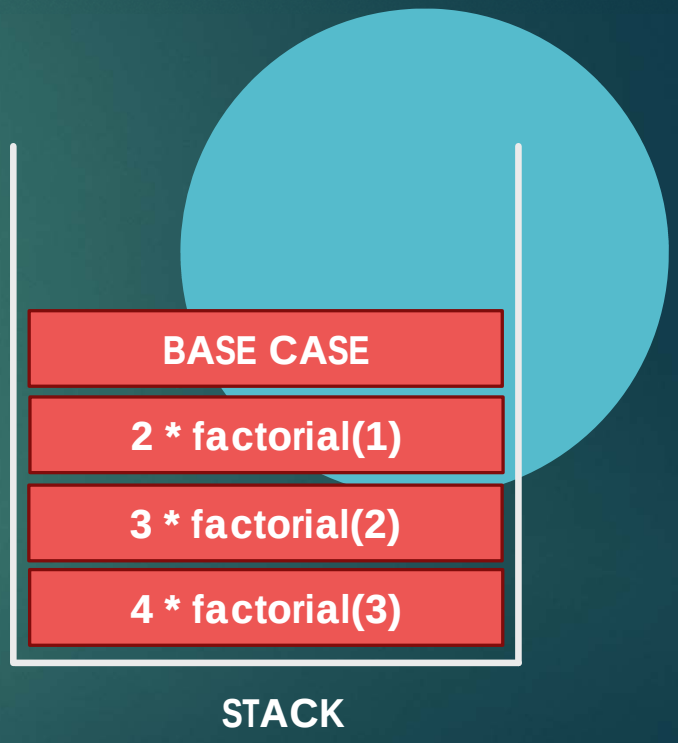
Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int factorial(int N){

  if( N == 1 ) return 1;

  return N * factorial(N-1);
}
```

RECURSION



We want to calculate the factorial for **N**

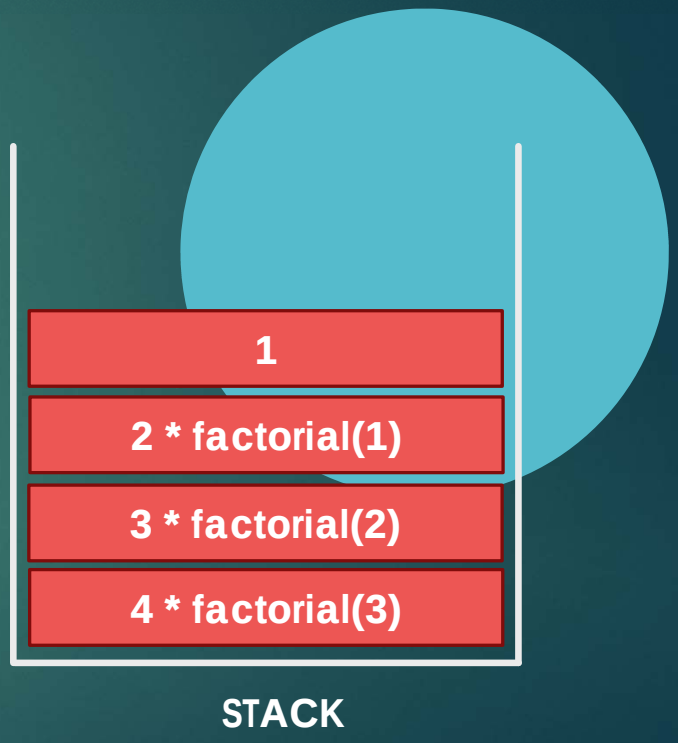
Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int factorial(int N){

  if( N == 1 ) return 1;

  return N * factorial(N-1);
}
```

RECURSION



We want to calculate the factorial for **N**

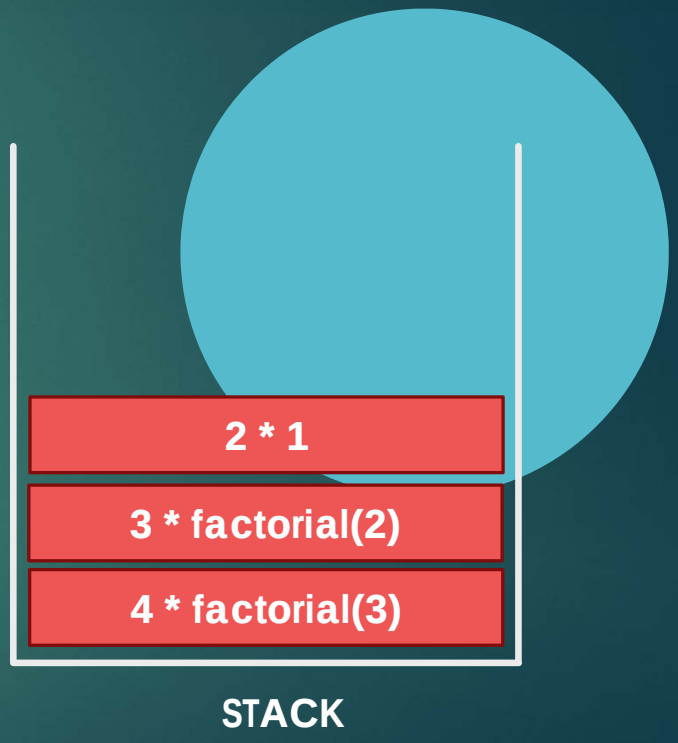
Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int factorial(int N){

  if( N == 1 ) return 1;

  return N * factorial(N-1);
}
```

RECURSION



We want to calculate the factorial for **N**

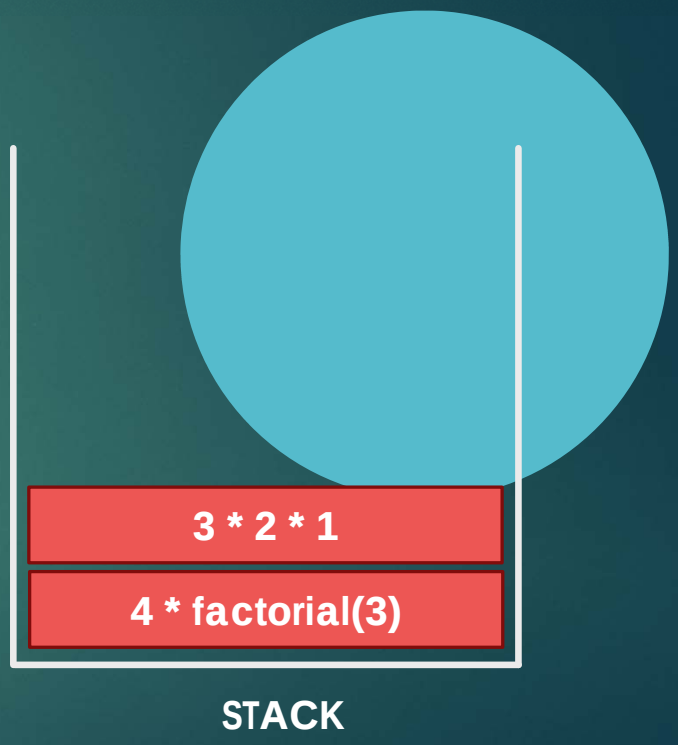
Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int factorial(int N){

  if( N == 1 ) return 1;

  return N * factorial(N-1);
}
```

RECURSION



We want to calculate the factorial for **N**

Usually we use a simple for / while loop but we can solve it with the help of recursive method calls

```
public int factorial(int N){

  if( N == 1 ) return 1;

  return N * factorial(N-1);
}
```

RECURSION