Recursion

Lecture 12

What is Recursion

- A method of programming in which a function refers to itself in order to solve a problem

- Never necessary
  - In some situations, results in simpler and/or easier-to-write code
  - Can often be more expensive in terms of memory + time

Example

Consider the **factorial** function

$$n! = \prod_{k=1}^{n} k = 1 * 2 * 3 * \ldots * n$$

Exercise

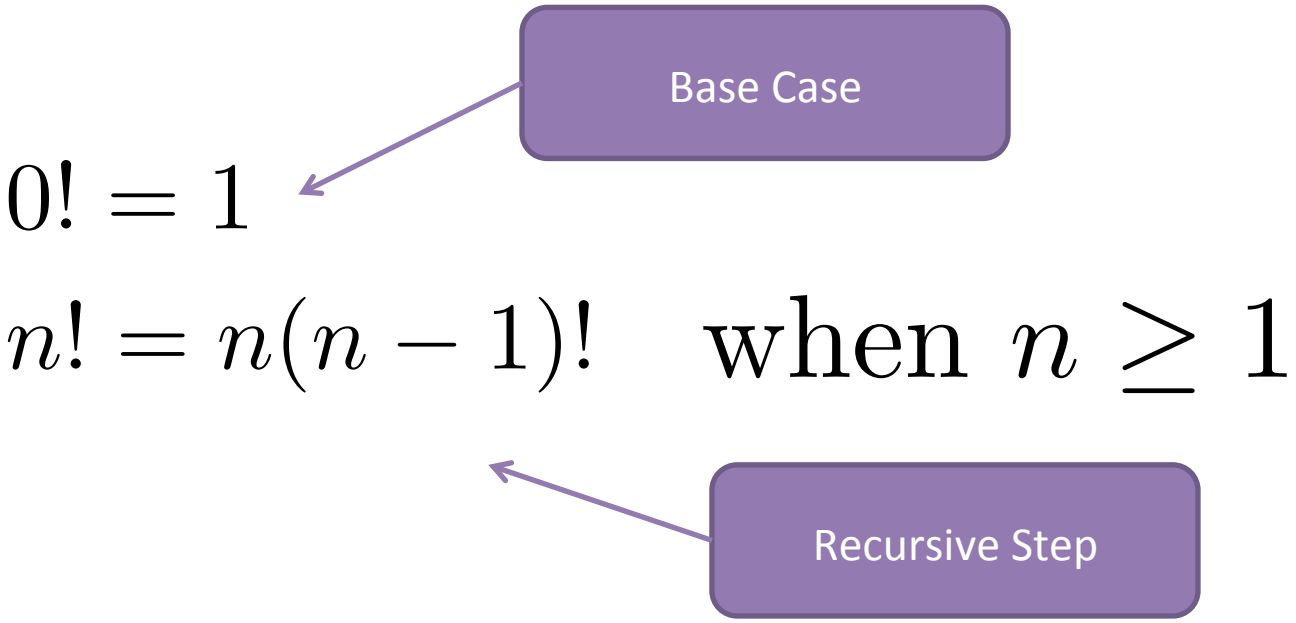
Write a **`factorial`** function that takes as input an integer and returns as an integer the result.

Answer

```
int factorial(int n)
{
  int result = 1;
  for ( int i=2; i<=n; i++ )
    result *= i;

  return result;
}
```

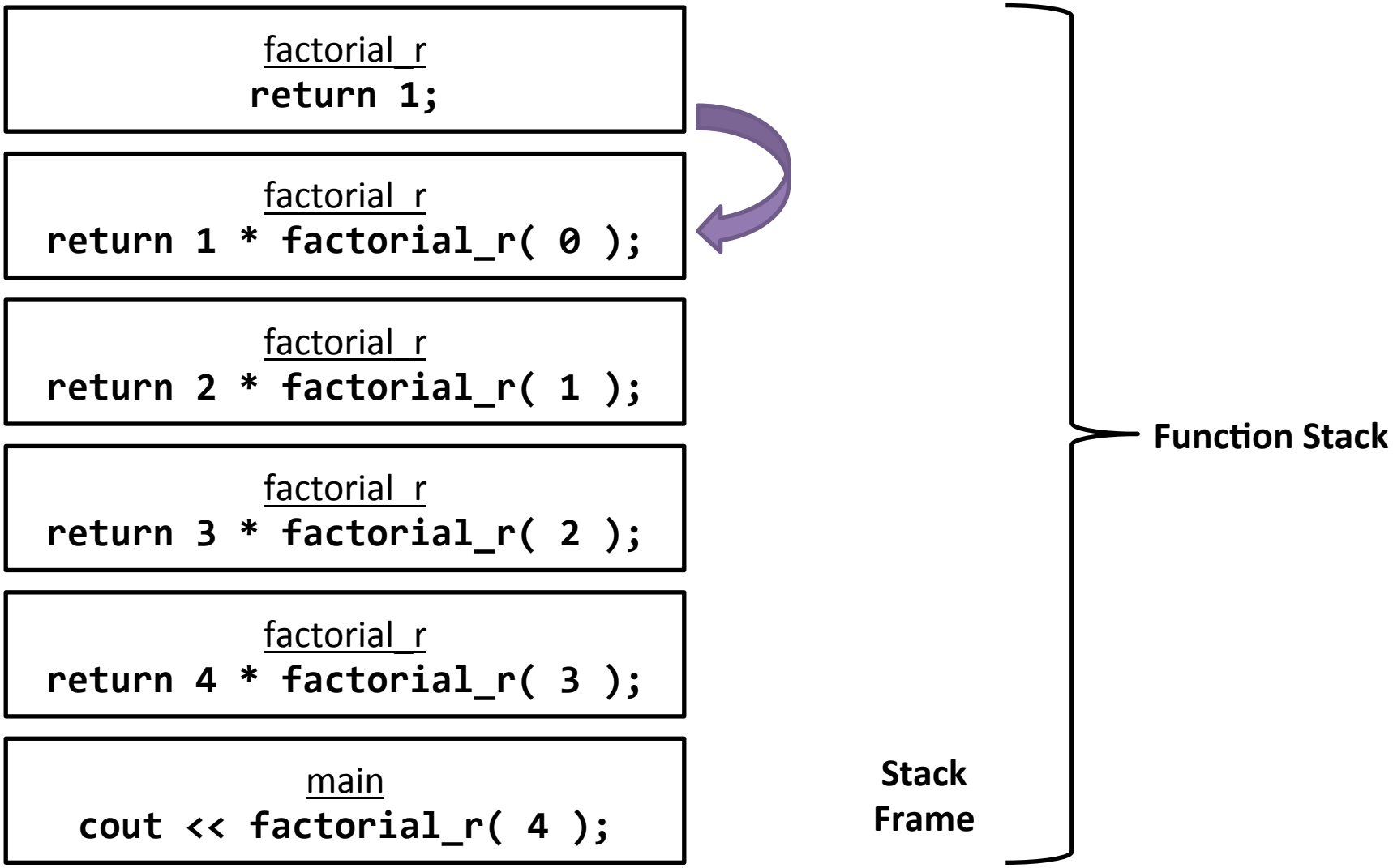
Consider a Recursive Definition



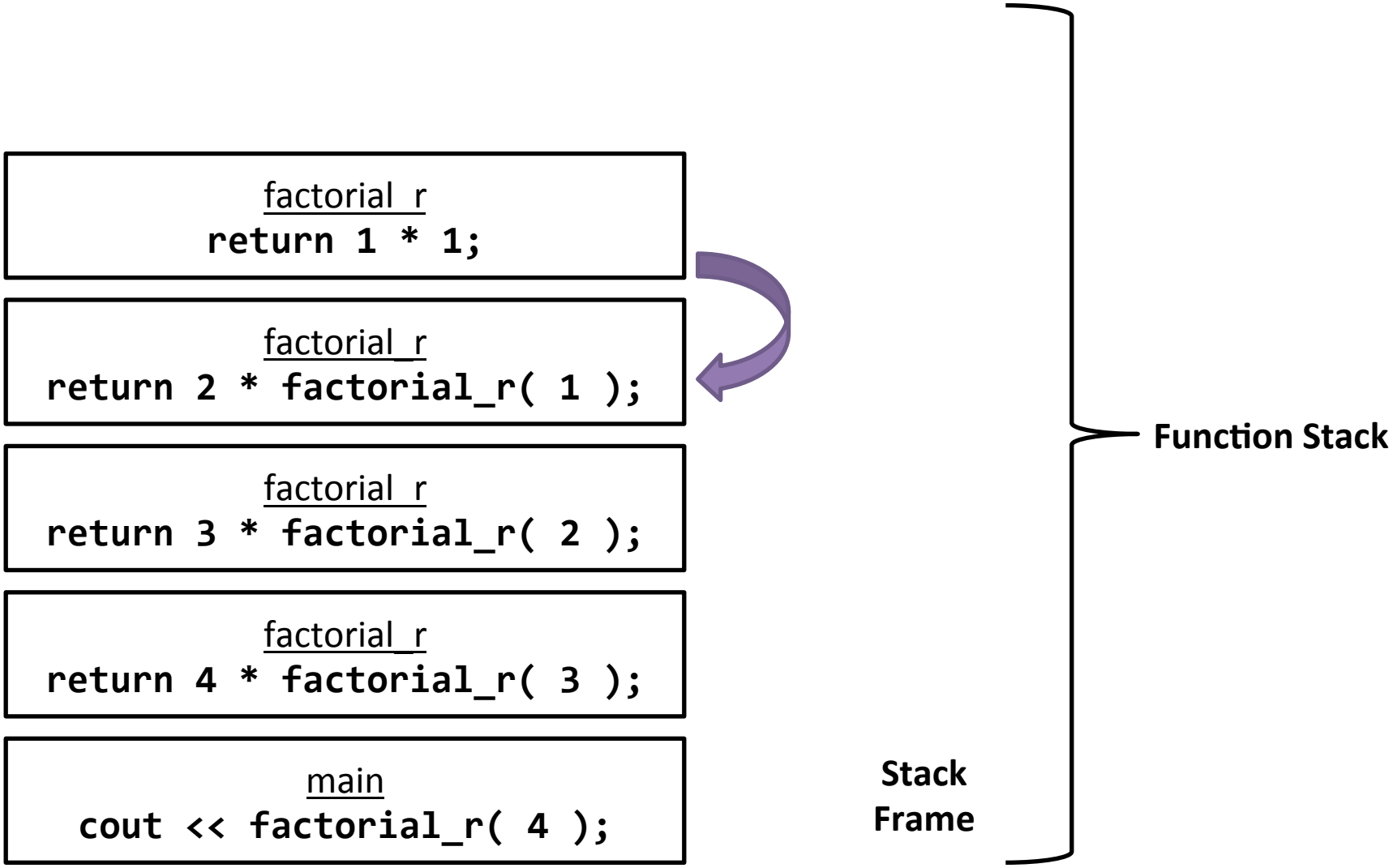
Conversion to Code

```
int factorial_r(int n)
{
   if ( n == 0 )
      return 1;
   else
      return ( n * factorial_r( n-1 ) );
}
```

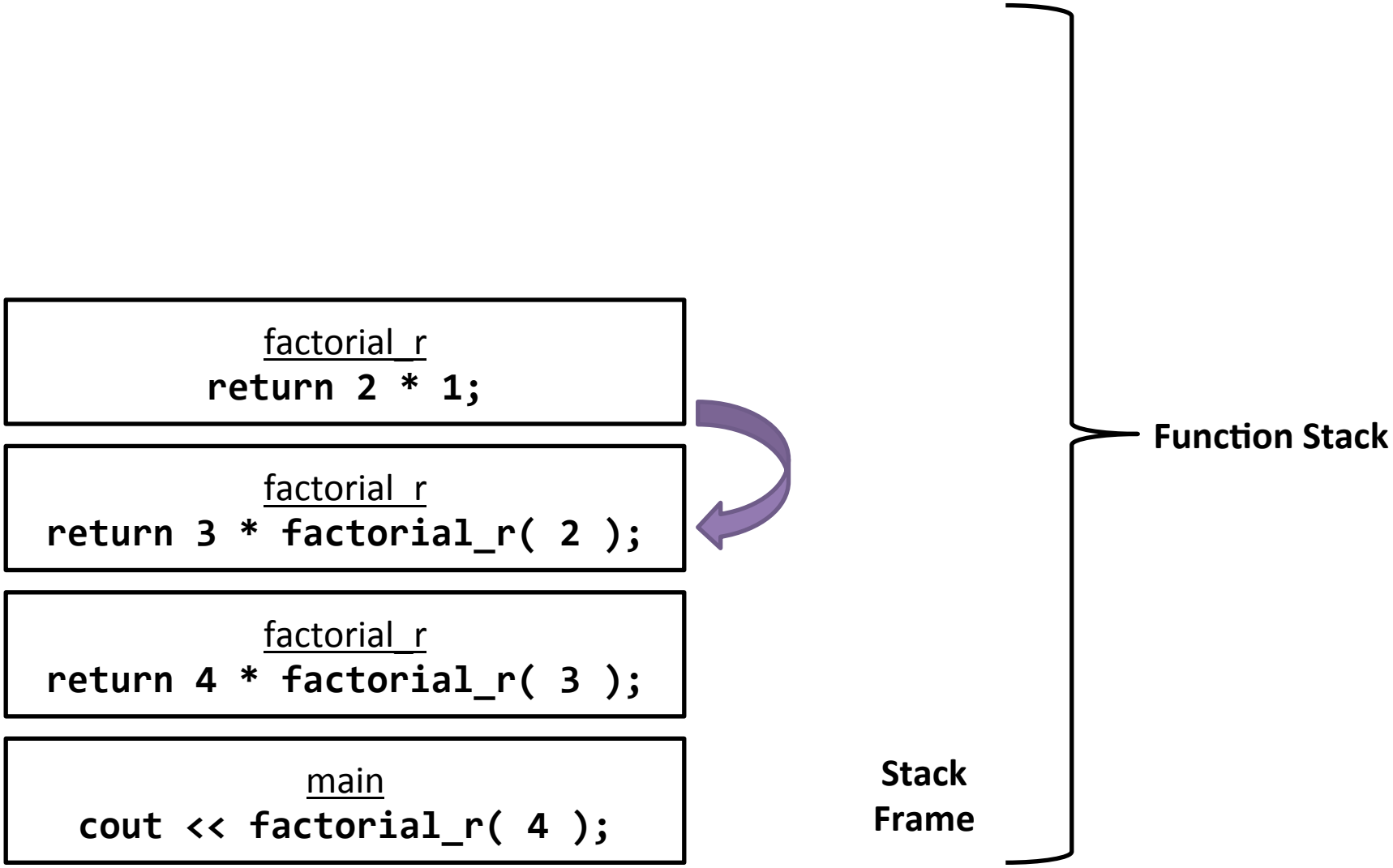
How the Code Executes



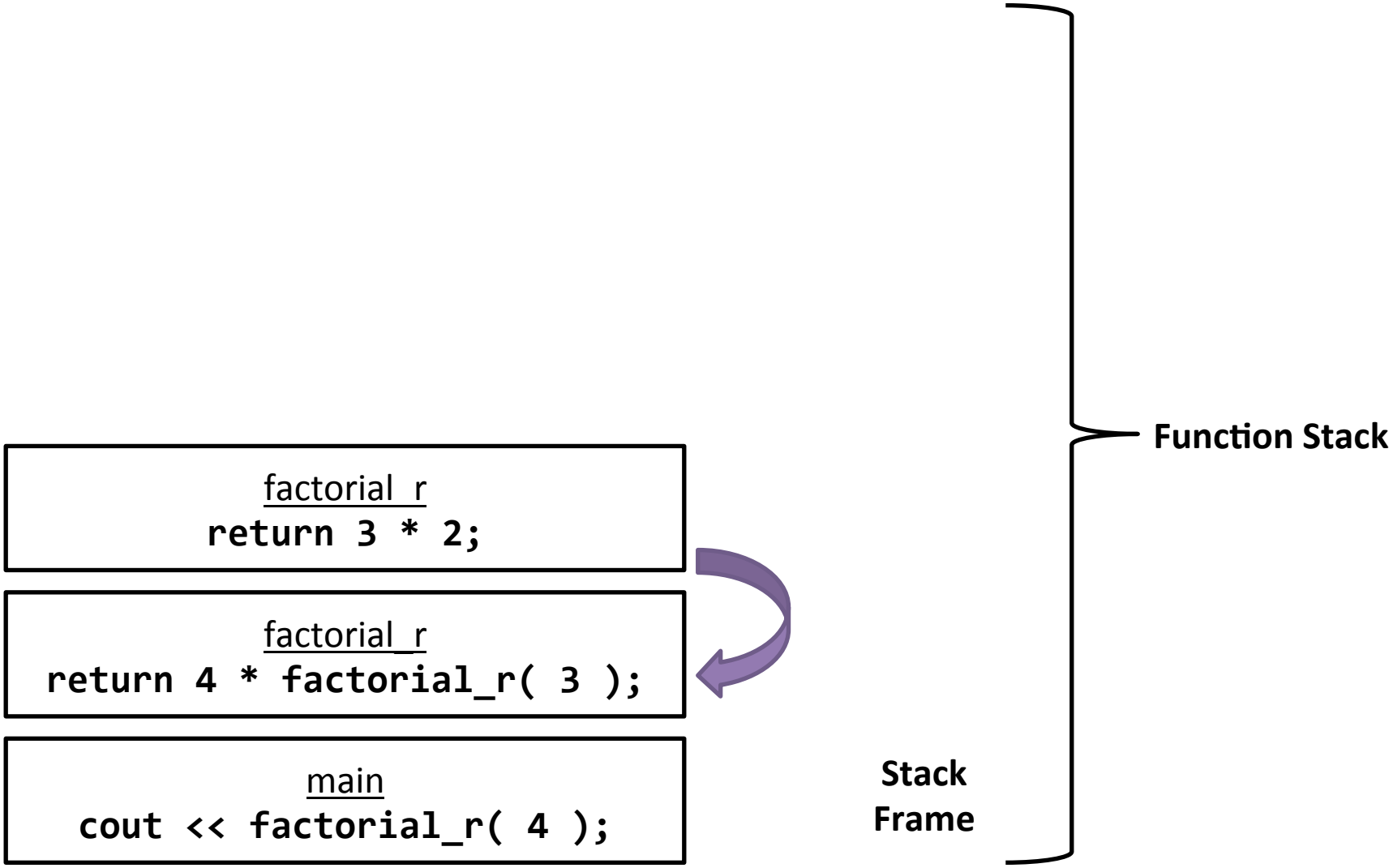
How the Code Executes



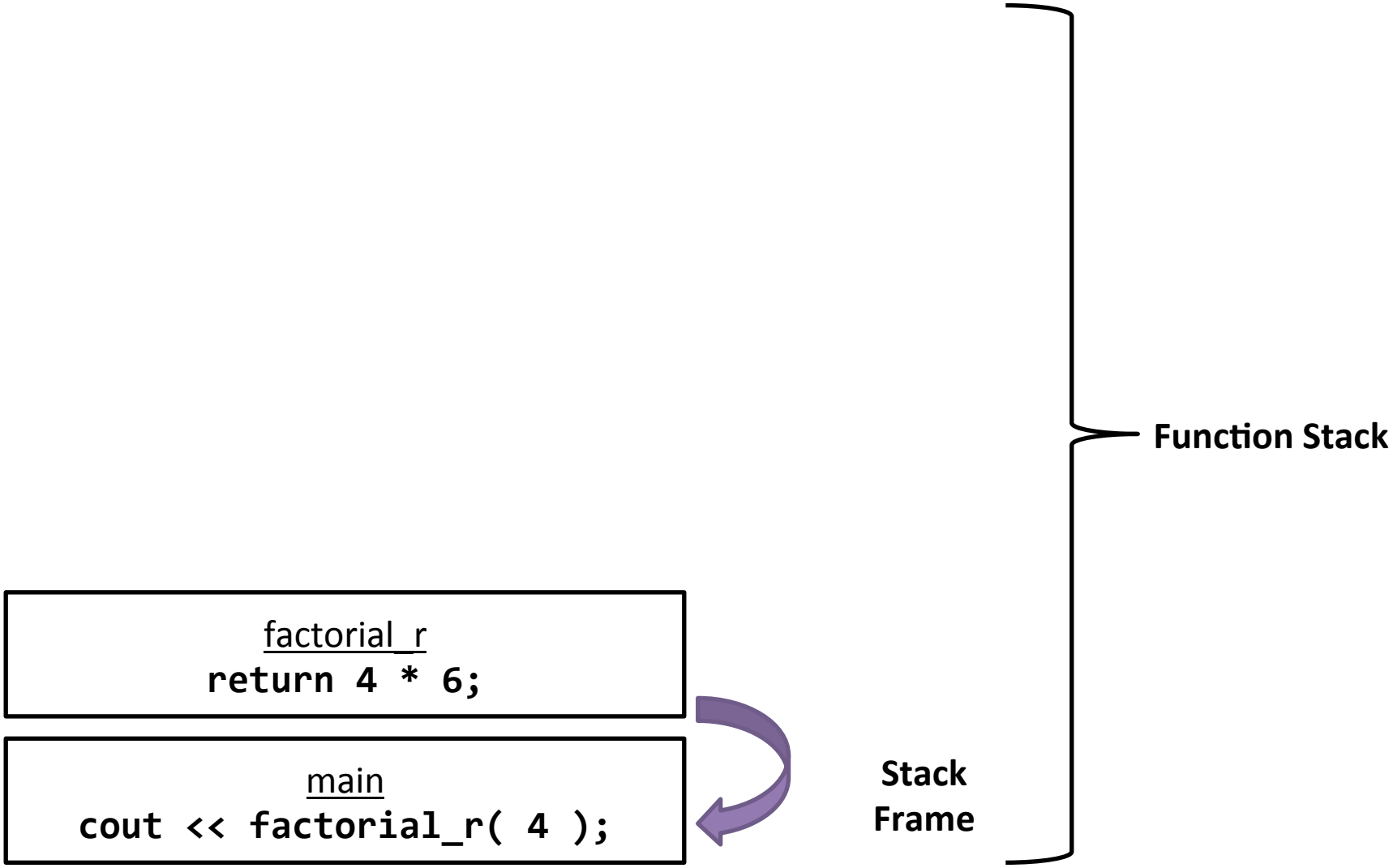
How the Code Executes



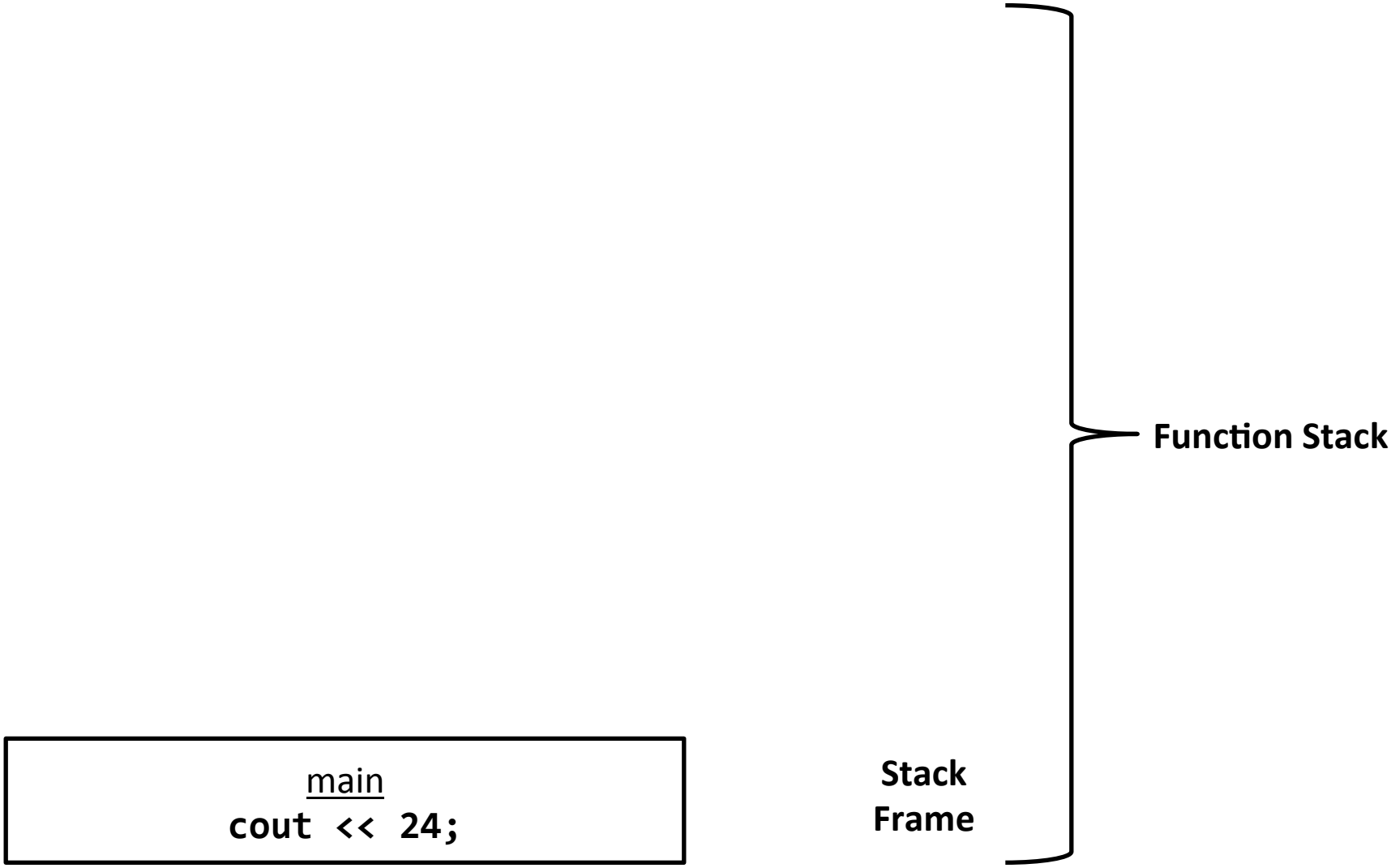
How the Code Executes



How the Code Executes



How the Code Executes



Exercise

Write a recursive function **`power`** that takes in two integer arguments (**`base`**, **`exponent`**) and returns $base^{exponent}$ using no libraries. Assume exponent will be non-negative.

Answer

```
int power(int base, int exponent)
{
  if ( exponent == 0 )
    return 1;

  return base *
         power( base, exponent-1 );
}
```

Exercise

Write a recursive function **`vertical_digits`** that outputs each digit of an integer to the screen on its own line. For example:

```
vertical_digits( 1234 );
1
2
3
4
```

Answer

```
void vertical_digits(int n)
{
   if ( n < 10 )
   {
      cout << n << endl;
   }
   else
   {
      vertical_digits( n / 10 );
      cout << ( n % 10 ) << endl;
   }
}
```

Exercise

Write a recursive function **`vertical_digits2`** that outputs each digit of an integer to the screen on its own line. For example:

```
vertical_digits2( 1234 );
4
3
2
1
```

Answer

```
void vertical_digits2(int n)
{
   if ( n < 10 )
   {
      cout << n << endl;
   }
   else
   {
      cout << ( n % 10 ) << endl;
      vertical_digits2( n / 10 );
   }
}
```

Exercise

In mathematics, the Fibonacci sequence is a sequence of integers:

0, 1, 1, 2, 3, 5, 8, 13, 21, …

Or, more formally:

$$F_n = F_{n-1} + F_{n-2}$$

$$F_0 = 0, F_1 = 1$$

Write the recursive **`fibb`** function, which takes one integer argument.

Answer

```
int fibb(int n)
{
   if ( n == 0 )
      return 0;
   else if ( n == 1 )
      return 1;
   else
      return fibb( n-1 ) + fibb( n-2 );
}
```

Wrap Up

- Recursive functions refer to themselves
- Each recursive function should have one or more *base case*, as well as a recursive step
- Recursion is never necessary (it can always be implemented iteratively with a stack), but often leads to simpler, easier-to-read code