CS 161
Intro to CS I

Recursion

Oregon State University

Recursion

- What is it?
  - Function that calls itself 1 or more times (directly or indirectly)
  - Has 1 or more base case for stopping
  - Inductive reasoning: general case must eventually be reduced to a base case

Example: Drawing Rectangles

- Iterative Solution:

```
void draw_rect(int i) {
    for( ; i > 0; i--){
      cout << "*******“ << endl;
      cout << "*          *“ << endl;
      cout << "*******“ << endl << endl;
    }
}
```

Oregon State University

Example: Drawing Rectangles

- Recursive Solution

```
void draw_rect(int i) {
      if(i>0){      //Base case
        draw_rect(--i);    //Recursive call
        cout << "*******“ << endl;
        cout << "*          *“ << endl;
        cout << "*******“ << endl << endl;
      }
  }
```

What is different when we call after?

- Recursive Solution

```
void draw_rect(int i) {
     if(i>0){      //Base case
       cout << "******“ << endl;
       cout << "*          *“ << endl;
       cout << "******“ << endl << endl;
       draw_rect(--i);    //Recursive call
     }
}
```

Example: Factorial

- Definition

  $0! = 1;$

  $n! = n * (n-1) * \ldots * (n-(n-1)) * 1 = n * (n-1)! \; ; \; n > 0$

Iterative Factorial

factorial(0) = 1;

factorial(n) = n*n-1*n-2*…*n-(n-1)*1;

```
long factorial(int n) {
        long fact;
        if(n==0)
            fact=1;
        else
           for(fact=n; n > 1; n--)
              fact=fact*(n-1);
       return fact;
}
```

Recursive Factorial

factorial(0) = 1;
factorial(n) = n*factorial(n-1);

```
long factorial(int n) {
    if (n == 0)      // Base case
      return 1;
    else
      return n * factorial(n - 1);    // Recursive call
 }
```

OSU Oregon State University

Computing Factorial Iteratively

factorial(4)

factorial(0) = 1;
factorial(n) = n*(n-1)*…*2*1;

Computing Factorial Iteratively

factorial(0) = 1;
factorial(n) = n*(n-1)*…*2*1;

factorial(4) = 4 * 3

Computing Factorial Iteratively

factorial(0) = 1;
factorial(n) = n*(n-1)*…*2*1;

factorial(4) = 4 * 3
= 12 * 2

Computing Factorial Iteratively

factorial(0) = 1;
factorial(n) = n*(n-1)*...*2*1;

factorial(4) = 4 * 3
= 12 * 2
= 24 * 1

Computing Factorial Iteratively

factorial(0) = 1;
factorial(n) = n*(n-1)*...*2*1;

factorial(4) = 4 * 3
= 12 * 2
= 24 * 1
= 24

Computing Factorial Recursively

```
factorial(0) = 1;
factorial(n) = n*factorial(n-1);
```

factorial(4)

Computing Factorial Recursively

factorial(0) = 1;
factorial(n) = n*factorial(n-1);

factorial(4) = 4 * factorial(3)

Computing Factorial Recursively

factorial(0) = 1;
factorial(n) = n*factorial(n-1);

factorial(4) = 4 * factorial(3)
= 4 * (3 * factorial(2))

Computing Factorial Recursively

factorial(0) = 1;
factorial(n) = n*factorial(n-1);

factorial(4) = 4 * factorial(3)
= 4 * (3 * factorial(2))
= 4 * (3 * (2 * factorial(1)))

Computing Factorial Recursively

```
factorial(0) = 1;
factorial(n) = n*factorial(n-1);
```

factorial(4) = 4 * factorial(3)

= 4 * (3 * factorial(2))

= 4 * (3 * (2 * factorial(1)))

= 4 * (3 * (2 * (1 * factorial(0))))

Computing Factorial Recursively

factorial(0) = 1;
factorial(n) = n*factorial(n-1);

factorial(4) = 4 * factorial(3)
= 4 * (3 * factorial(2))
= 4 * (3 * (2 * factorial(1)))
= 4 * (3 * (2 * (1 * factorial(0))))
= 4 * (3 * (2 * (1 *1)))

Computing Factorial Recursively

factorial(0) = 1;
factorial(n) = n*factorial(n-1);

factorial(4) = 4 * factorial(3)
= 4 * (3 * factorial(2))
= 4 * (3 * (2 * factorial(1)))
= 4 * (3 * (2 * (1 * factorial(0))))
= 4 * (3 * (2 * (1 *1)))
= 4 * (3 * (2 * 1))

Computing Factorial Recursively

```
factorial(0) = 1;
factorial(n) = n*factorial(n-1);
```

factorial(4) = 4 * factorial(3)
= 4 * (3 * factorial(2))
= 4 * (3 * (2 * factorial(1)))
= 4 * (3 * (2 * (1 * factorial(0))))
= 4 * (3 * (2 * (1 *1)))
= 4 * (3 * (2 * 1))
= 4 * (3 * 2)

Computing Factorial Recursively

```
factorial(0) = 1;
factorial(n) = n*factorial(n-1);
```

factorial(4) = 4 * factorial(3)
= 4 * (3 * factorial(2))
= 4 * (3 * (2 * factorial(1)))
= 4 * (3 * (2 * (1 * factorial(0))))
= 4 * (3 * (2 * (1 *1)))
= 4 * (3 * (2 * 1))
= 4 * (3 * 2)
= 4 * 6

Computing Factorial Recursively

```
factorial(0) = 1;
factorial(n) = n*factorial(n-1);
```

factorial(4) = 4 * factorial(3)
= 4 * (3 * factorial(2))
= 4 * (3 * (2 * factorial(1)))
= 4 * (3 * (2 * (1 * factorial(0))))
= 4 * (3 * (2 * (1 *1)))
= 4 * (3 * (2 * 1))
= 4 * (3 * 2)
= 4 * 6
= 24

Differences

- Pros
  - Readability
- Cons
  - Efficiency
  - Memory

Recursive Factorial

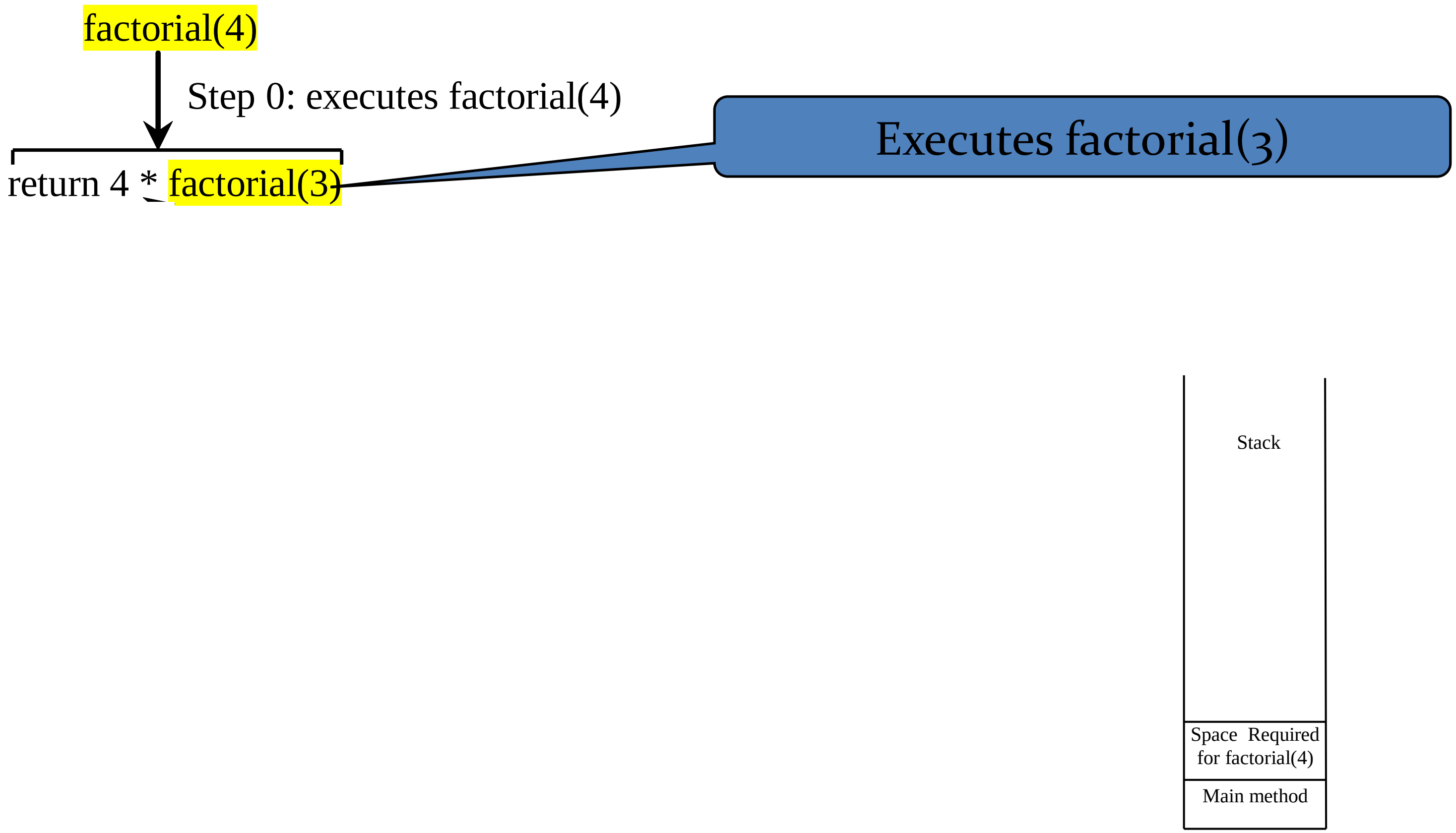
Executes factorial(4)

factorial(4)

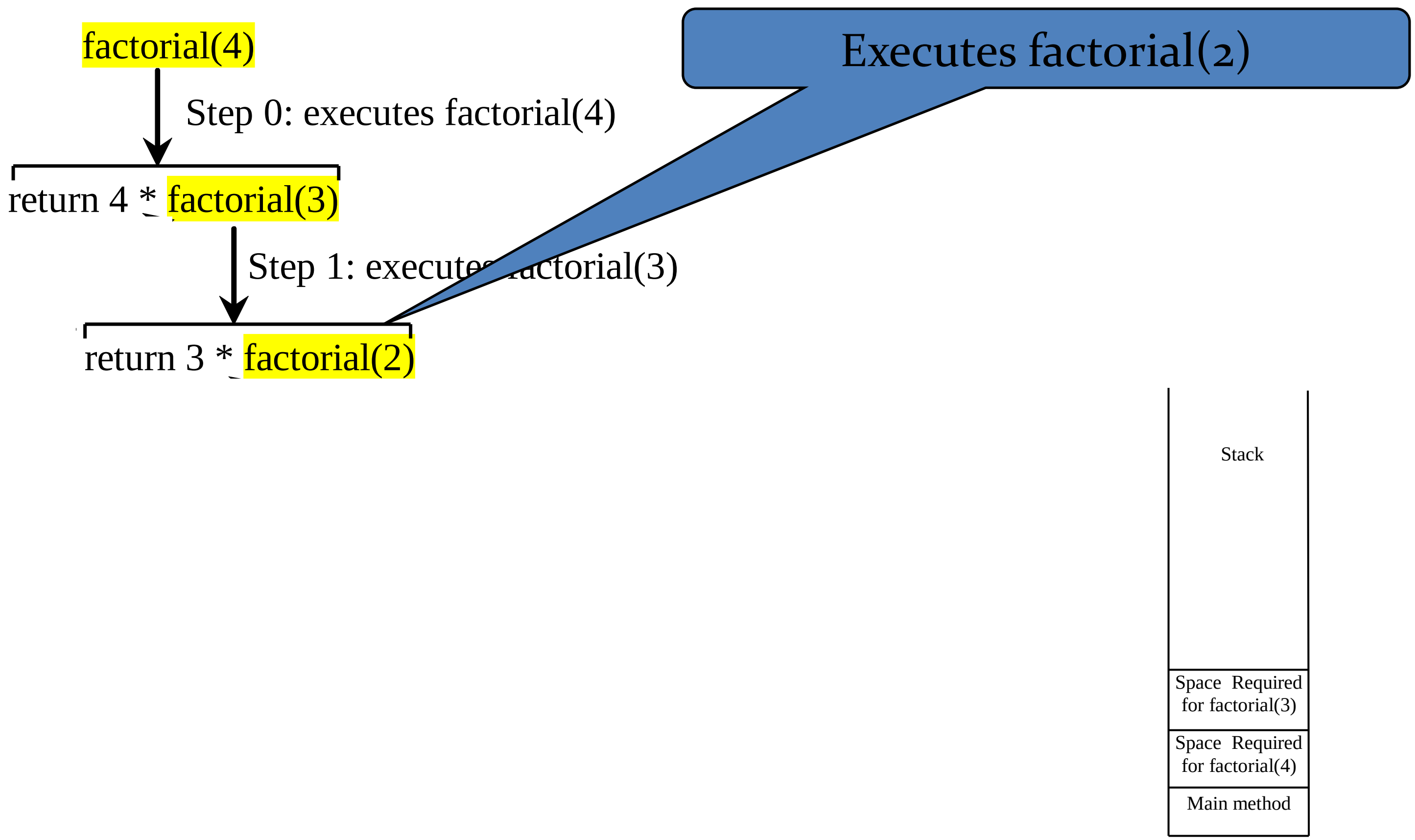
Stack

Main method

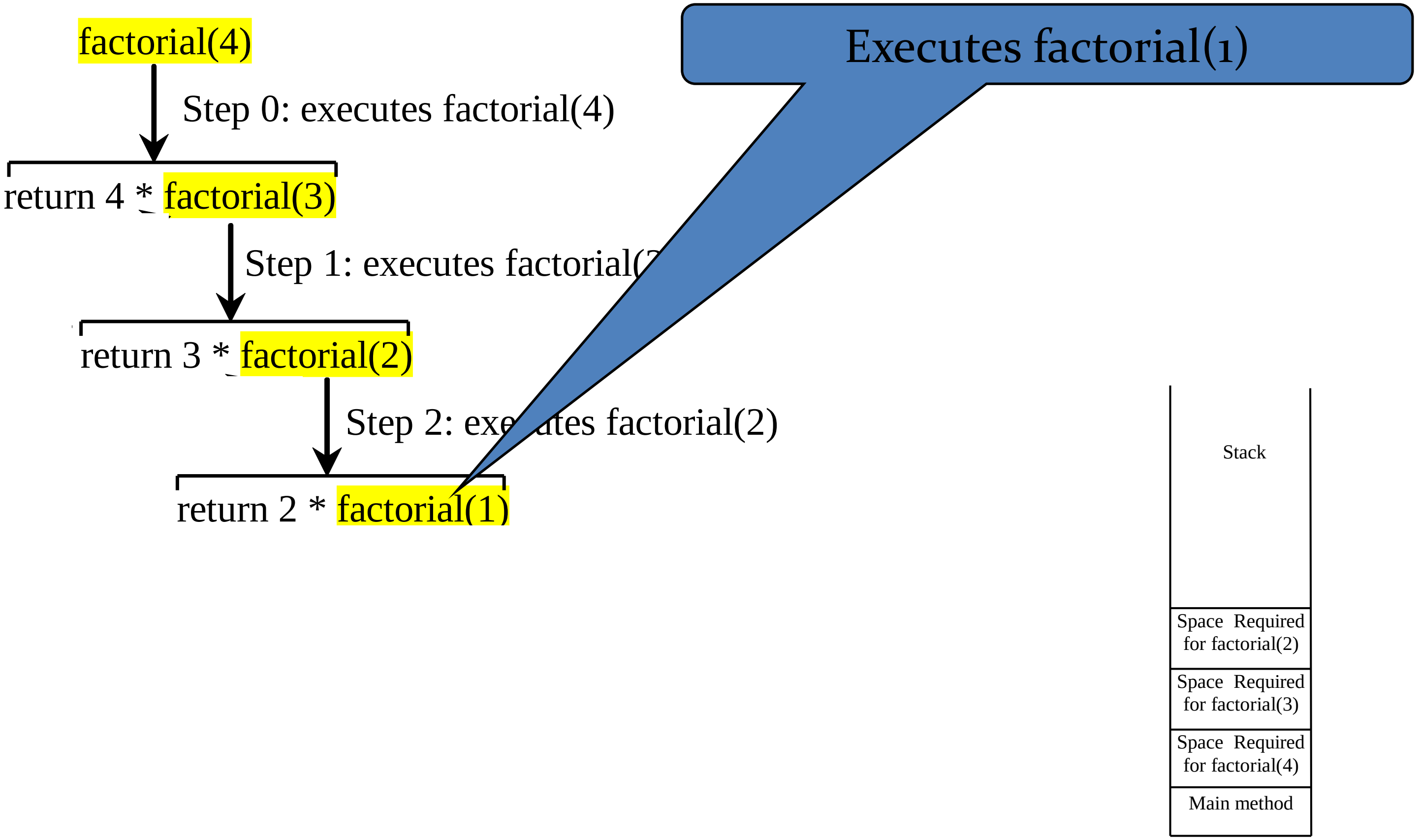
Recursive Factorial



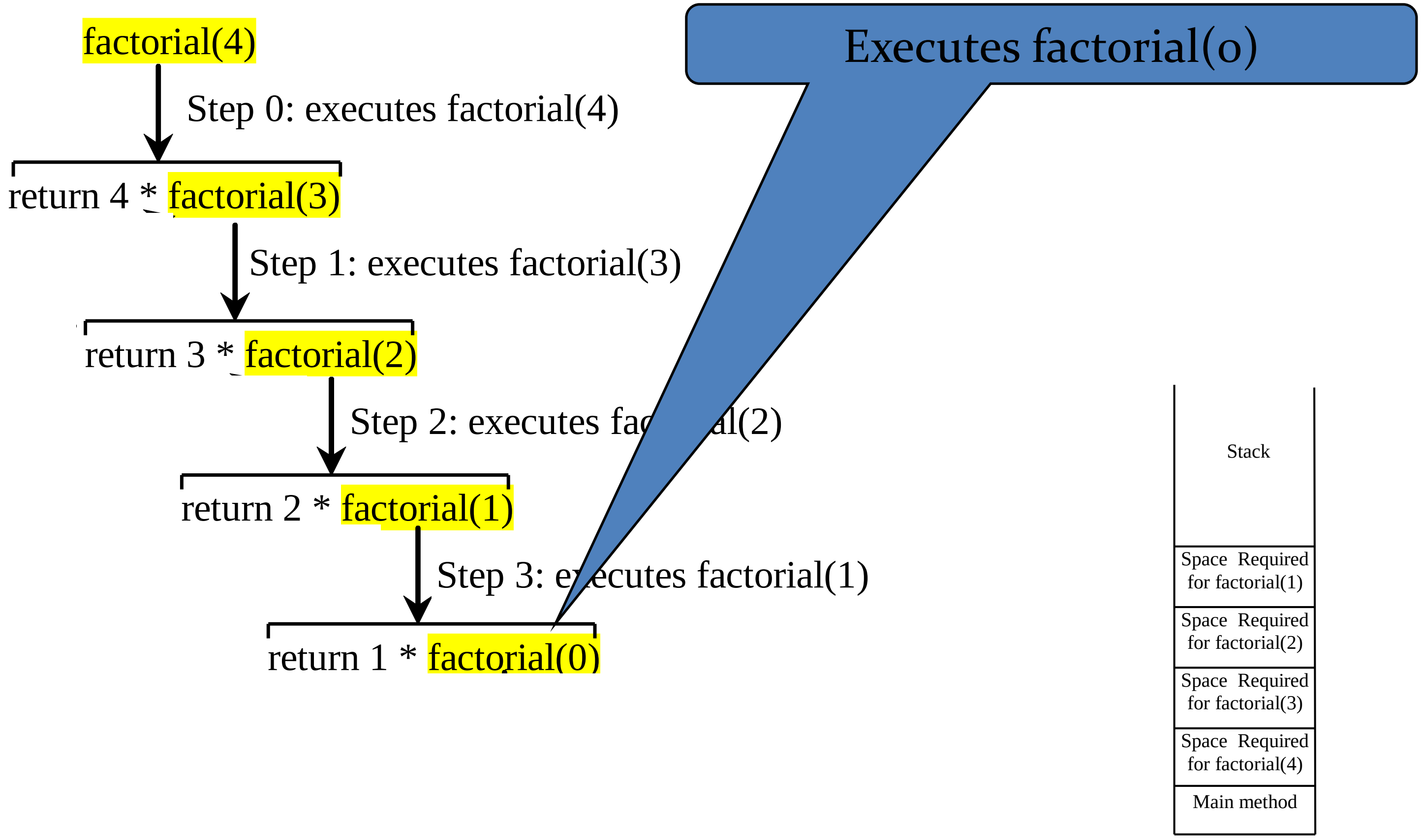
Recursive Factorial



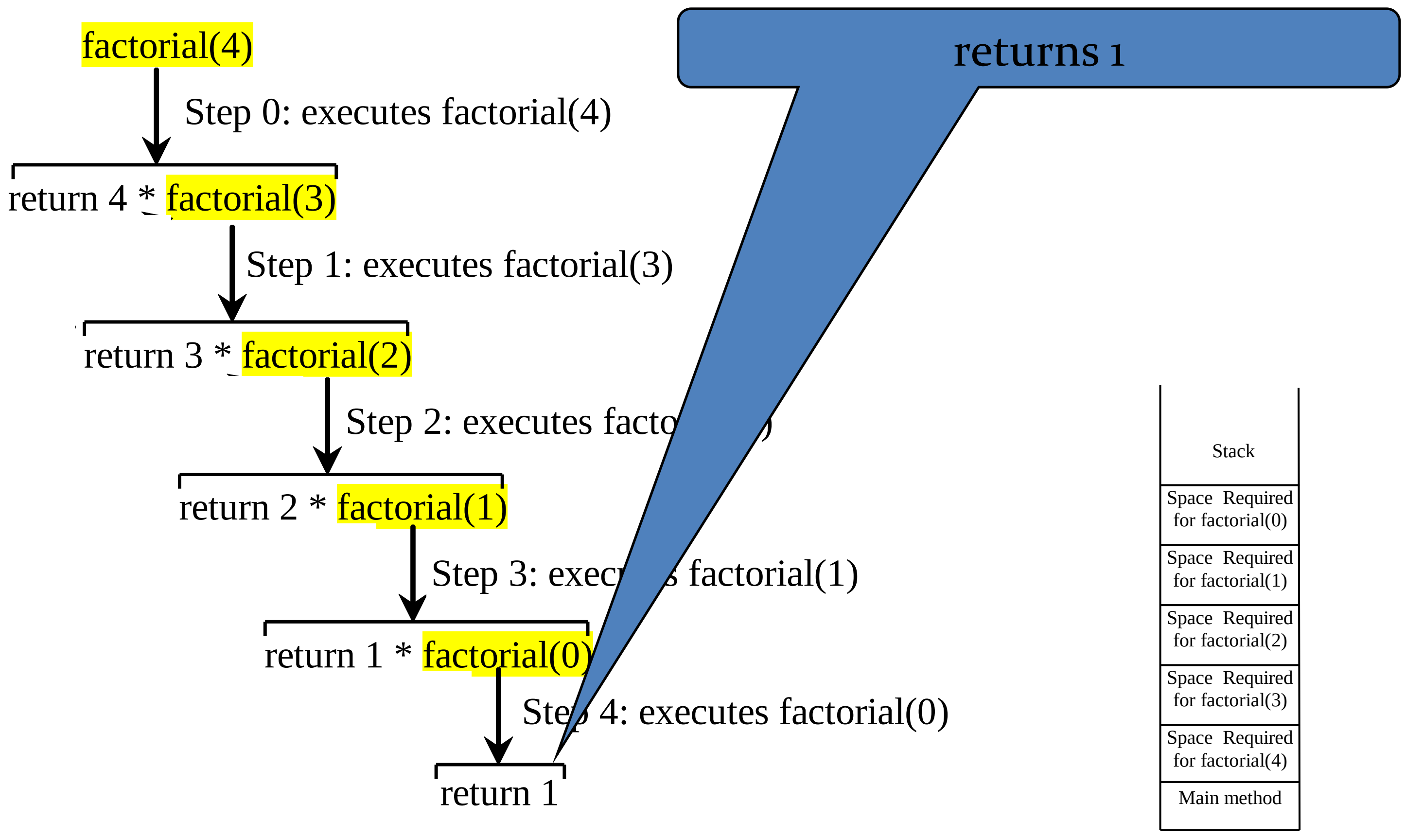
Recursive Factorial



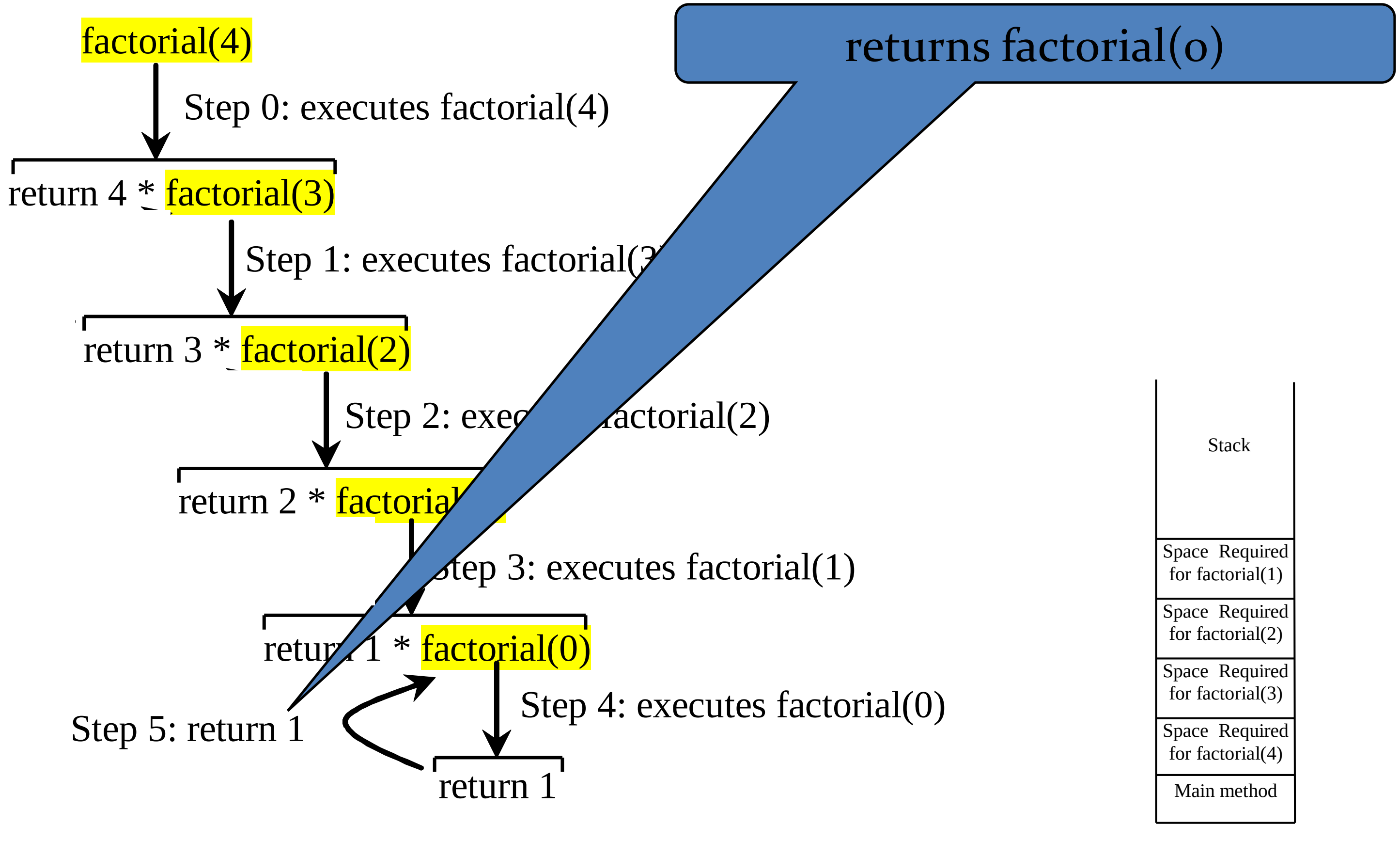
Recursive Factorial



Recursive Factorial

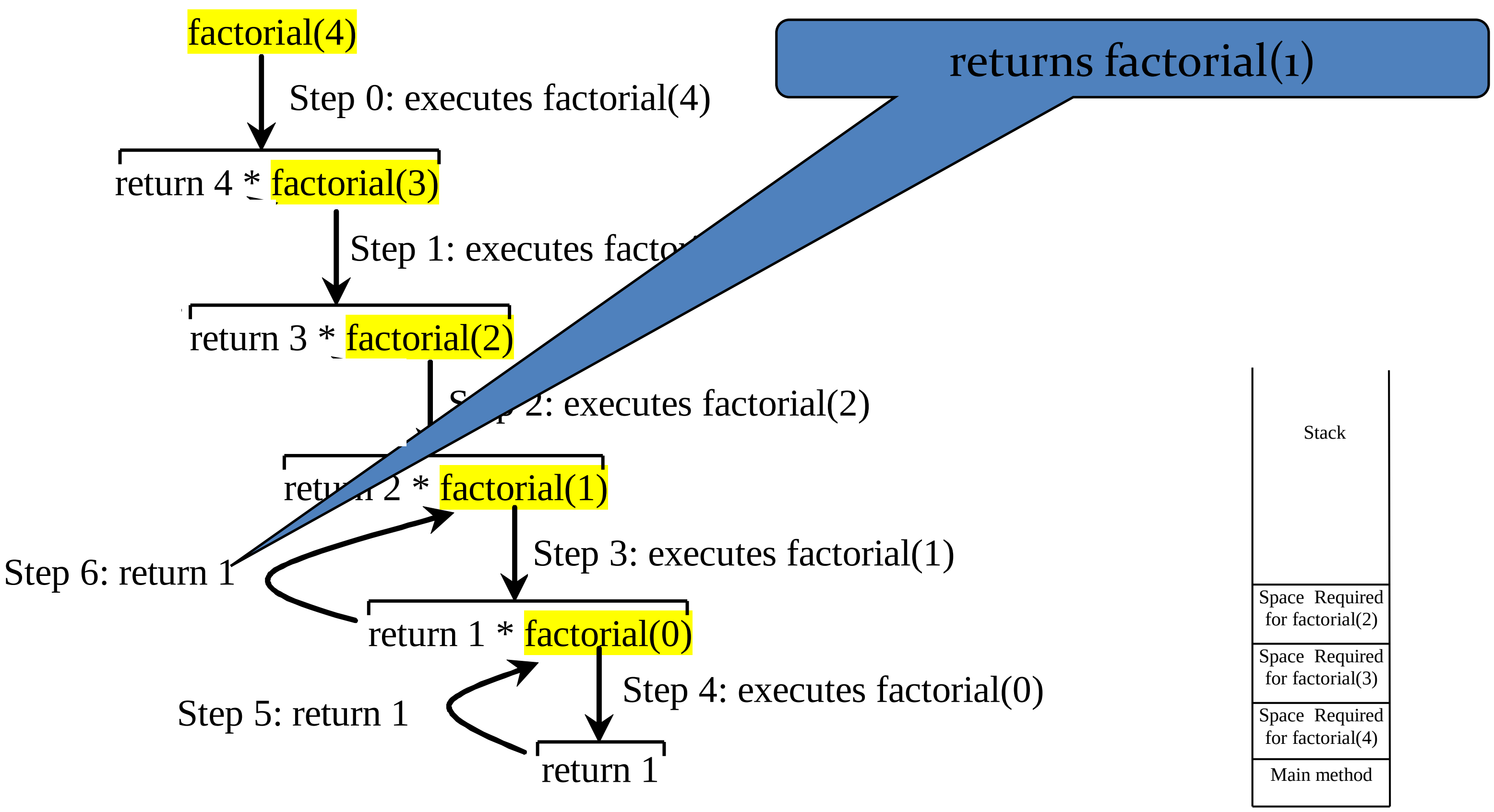


Recursive Factorial

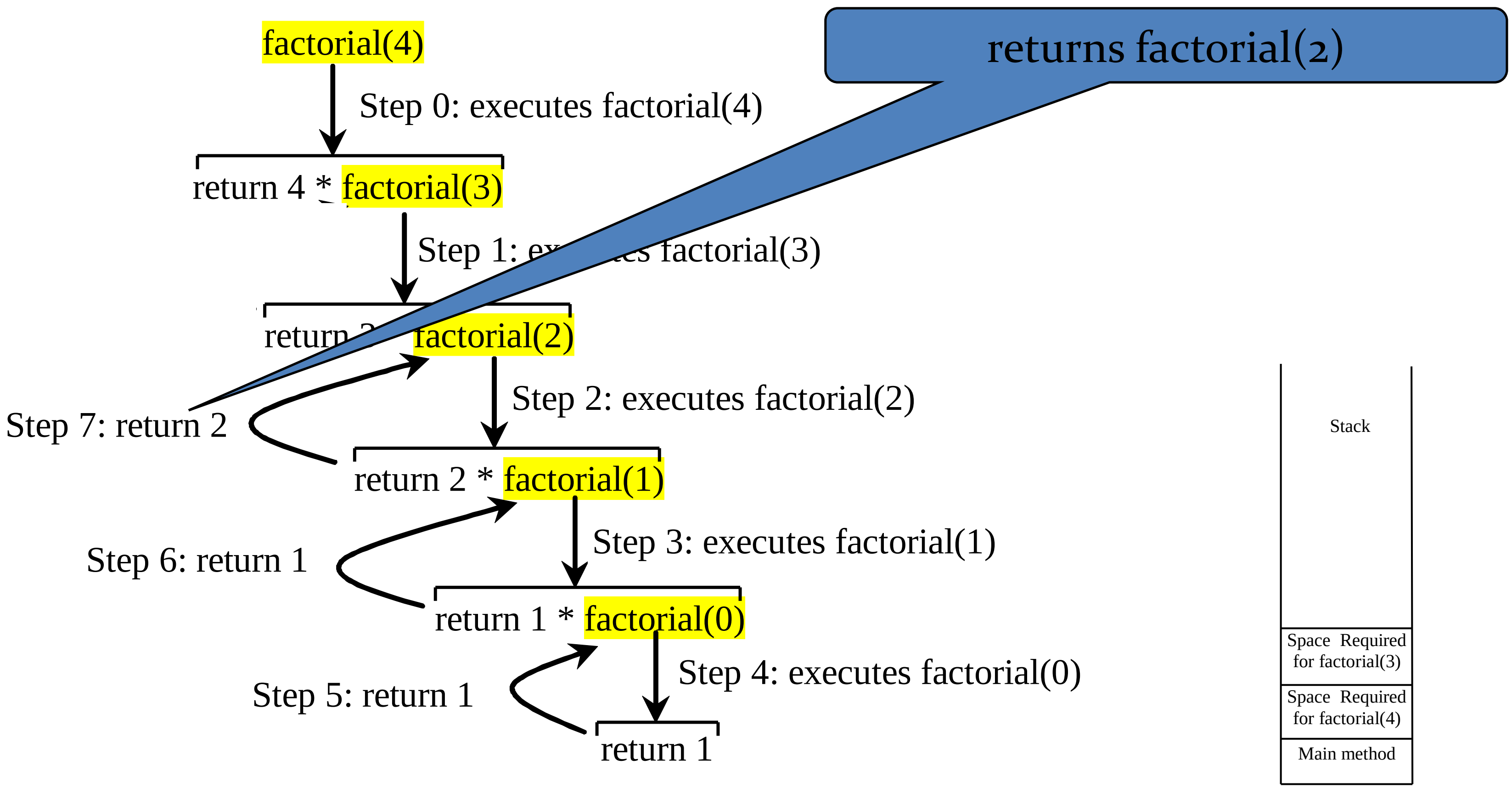


Oregon State University

Recursive Factorial

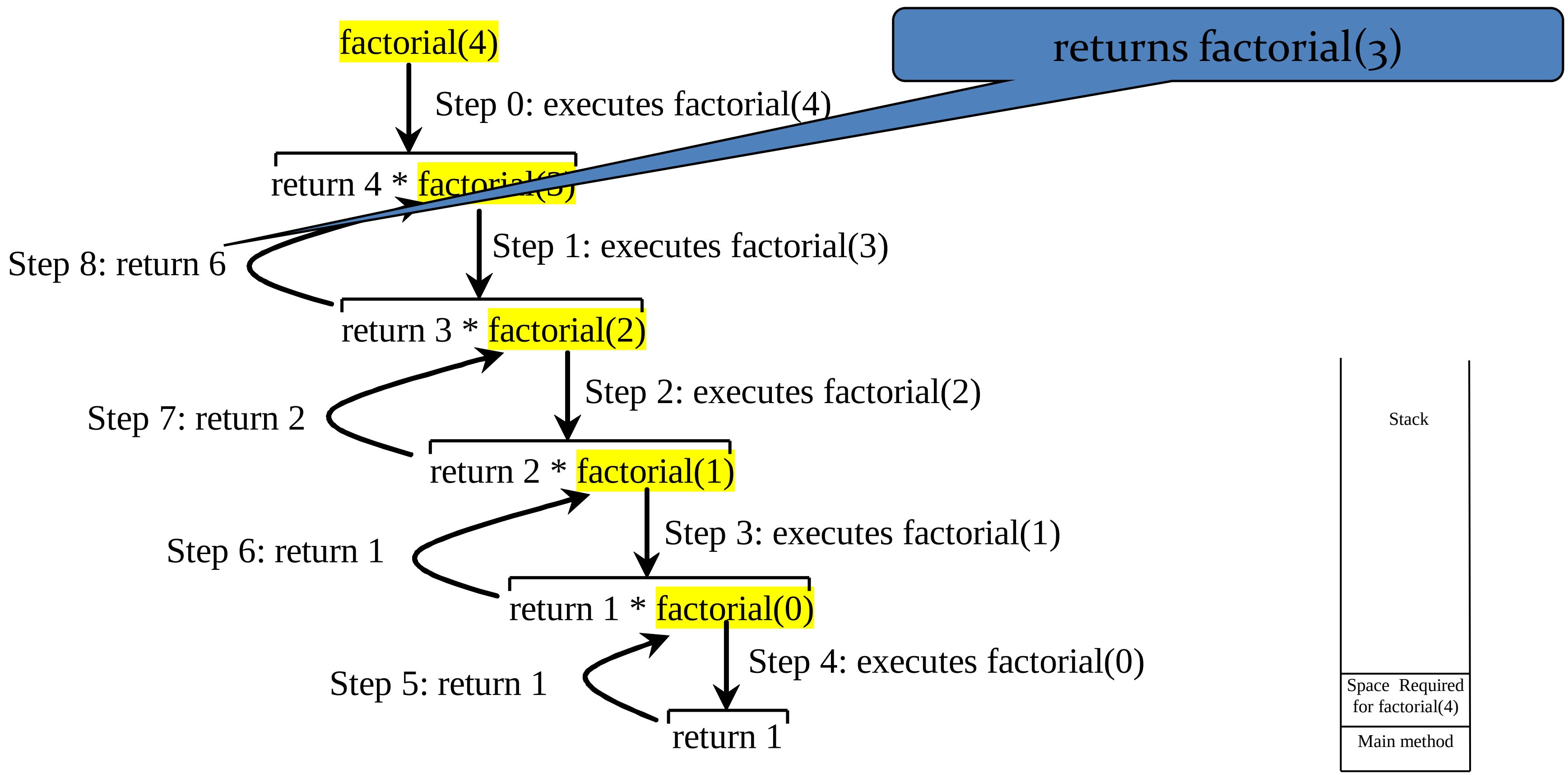


Recursive Factorial

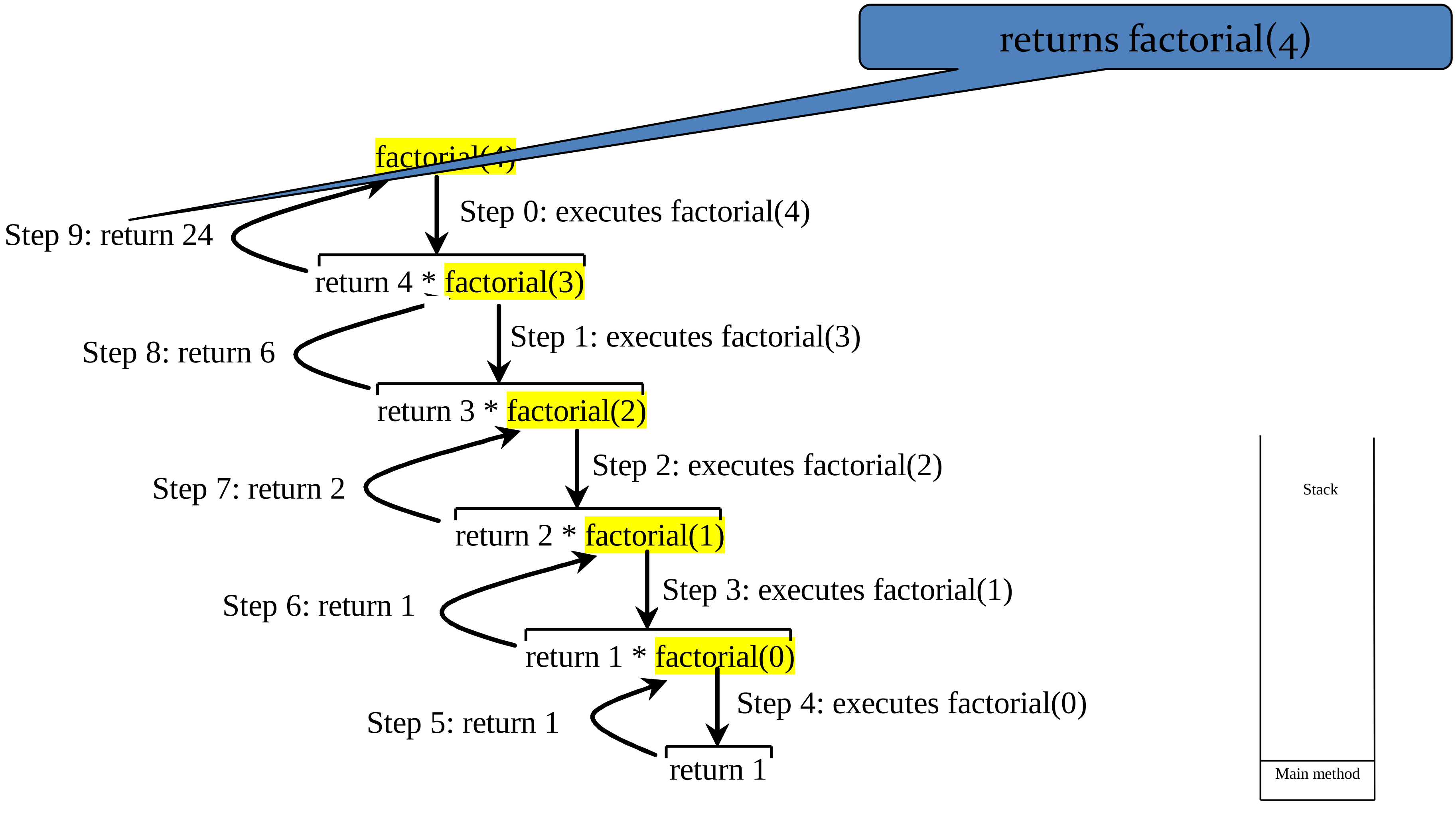


OSU Oregon State University

Recursive Factorial



Recursive Factorial



In-class Exercise #4

- Get into groups of 4 – 5.
- Write your own recursive *int pwr()* function that takes two integers as arguments and returns the integer result.
  - What does the function prototype look like?
  - Now, write the function definition…

Oregon State University