CS 161
Intro to CS I

Recursion

Example: Factorial

- Definition

  0!=1;

  n!=n * (n-1) * … * (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
  }
```

Computing Factorial Iteratively

factorial(4)

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

Computing Factorial Iteratively

factorial(4) = 4 * 3

```
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
= 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(4)

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

Computing Factorial Recursively

factorial(4) = 4 * factorial(3)

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

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

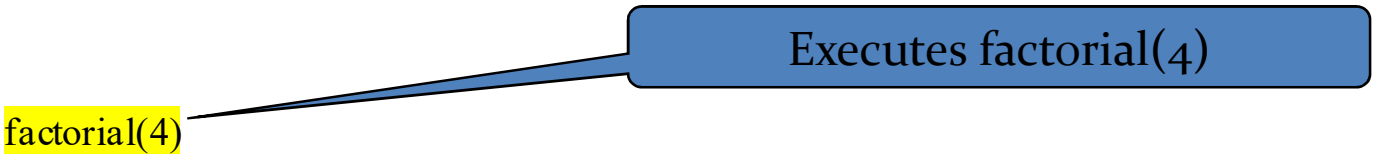
```
factorial(0) = 1;
factorial(n) = n*factorial(n-1);
```

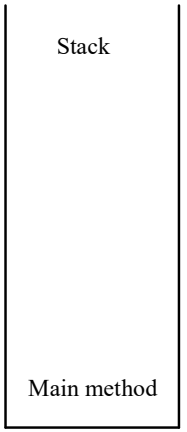
factorial(4) = 4 * factorial(3)
= 4 * (3 * factorial(2))
= 4 * (3 * (2 * factorial(1)))
= 4 * (3 * (2 * (1 * factorial(0))))
= 4 * (3 * (2 * (1 *1)))
= 4 * (3 * (2 * 1))
= 4 * (3 * 2)
= 4 * 6
= 24

Differences

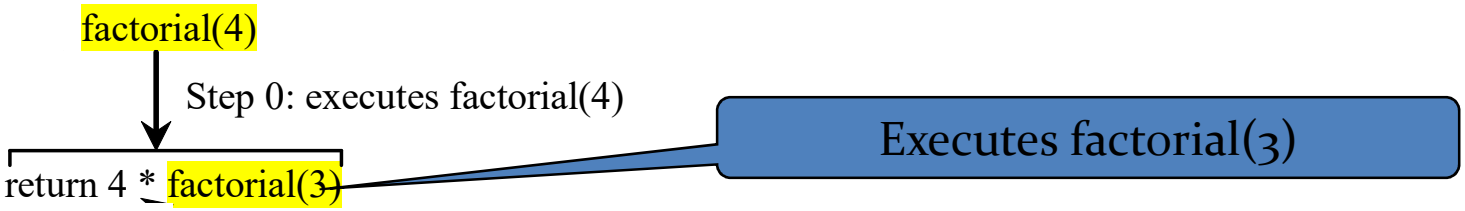
- Pros
  - Readability
- Cons
  - Efficiency
  - Memory

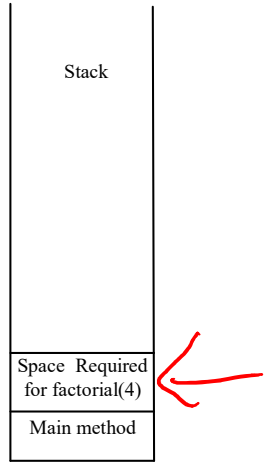
Recursive Factorial



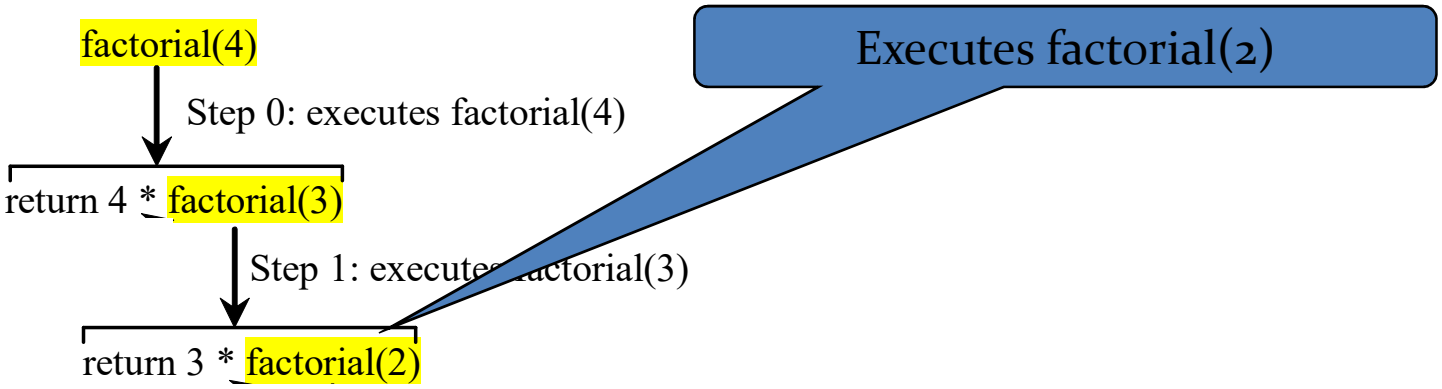


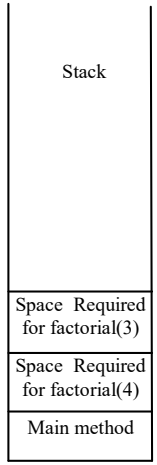
Recursive Factorial



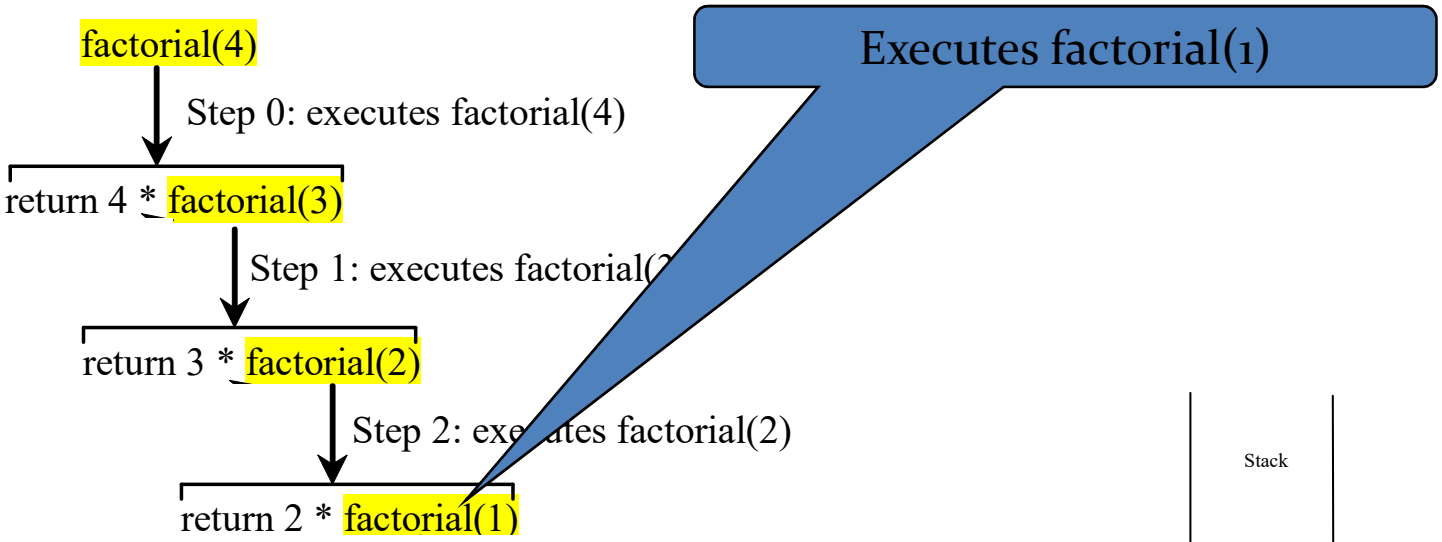


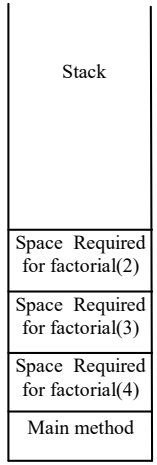
Recursive Factorial



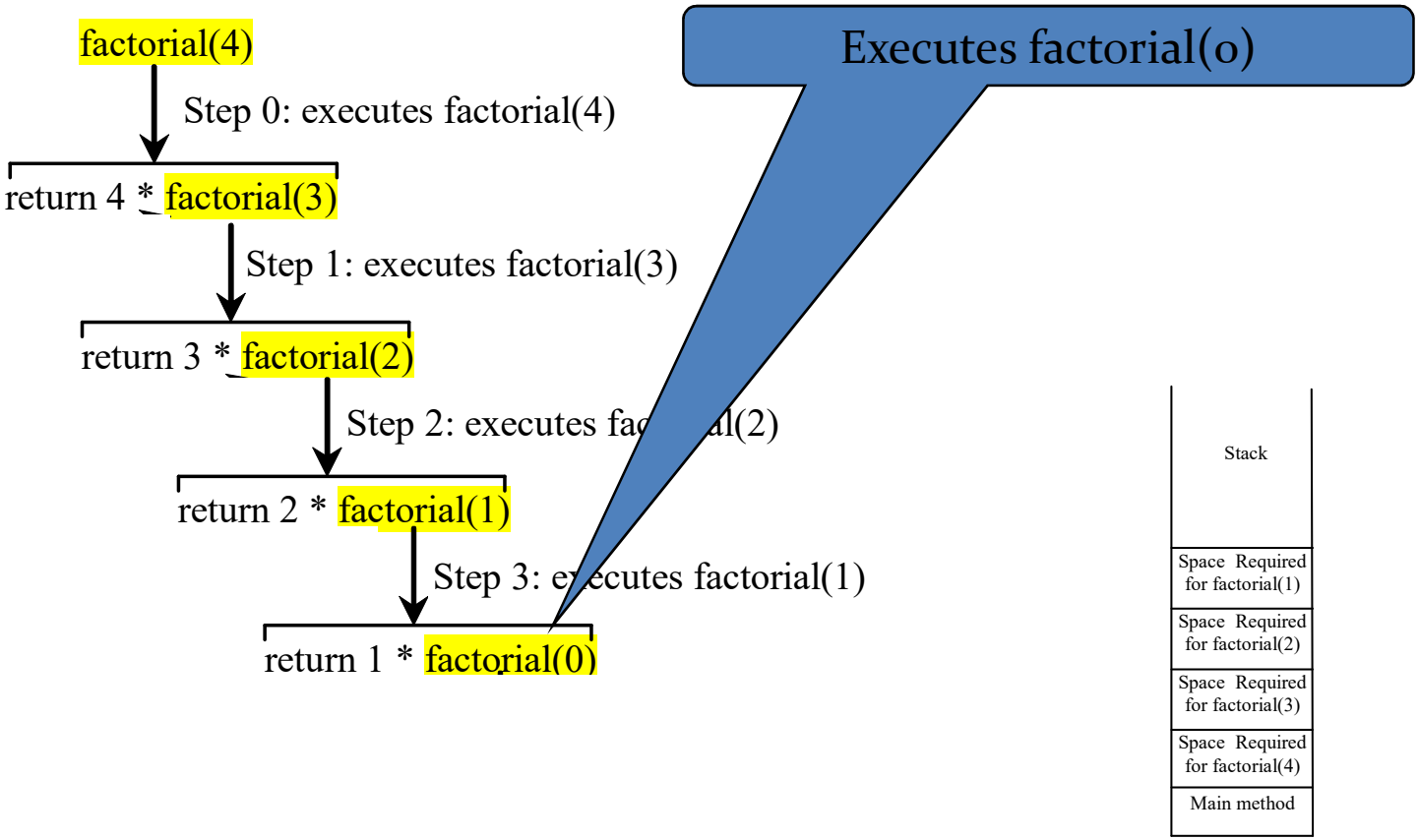


Recursive Factorial

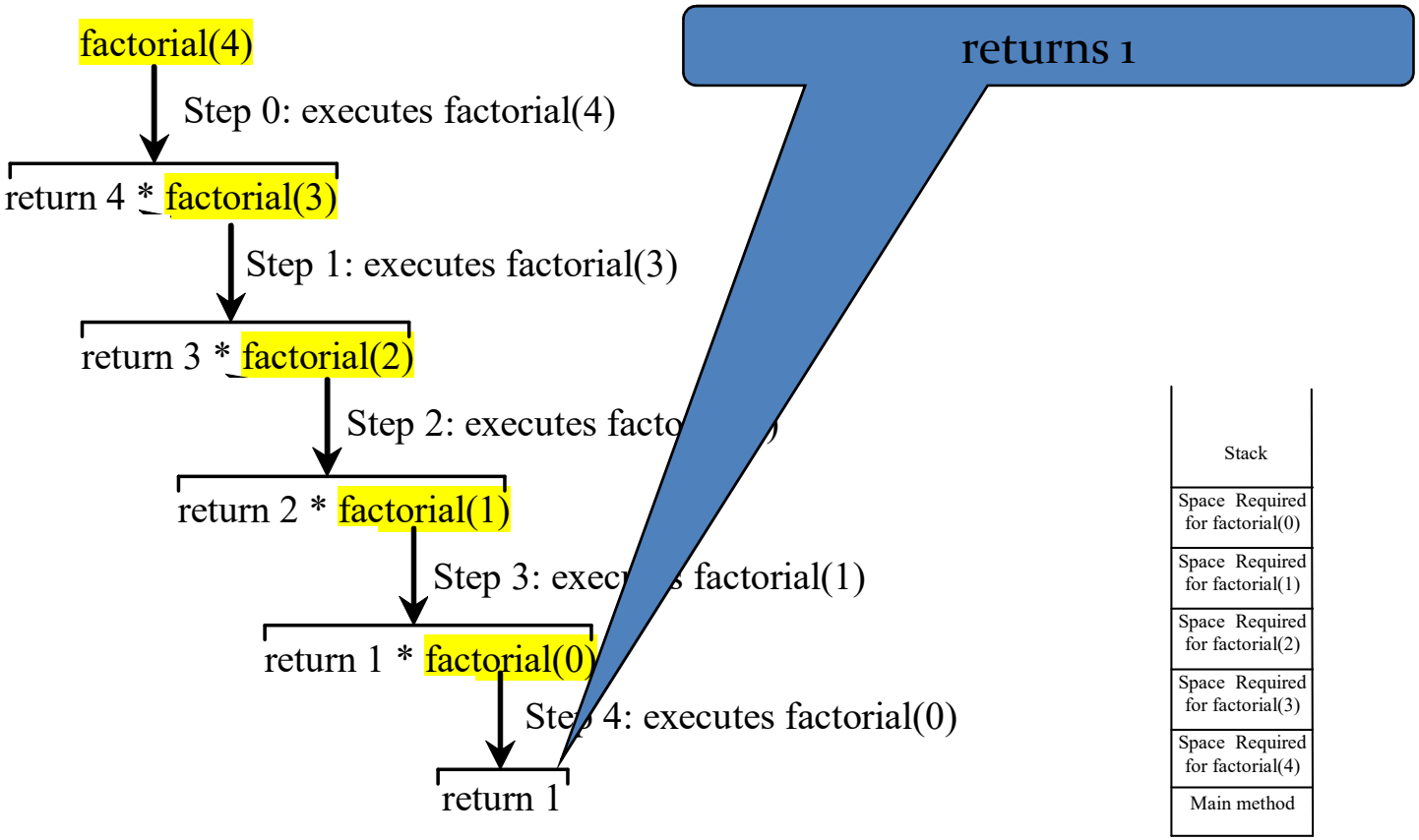




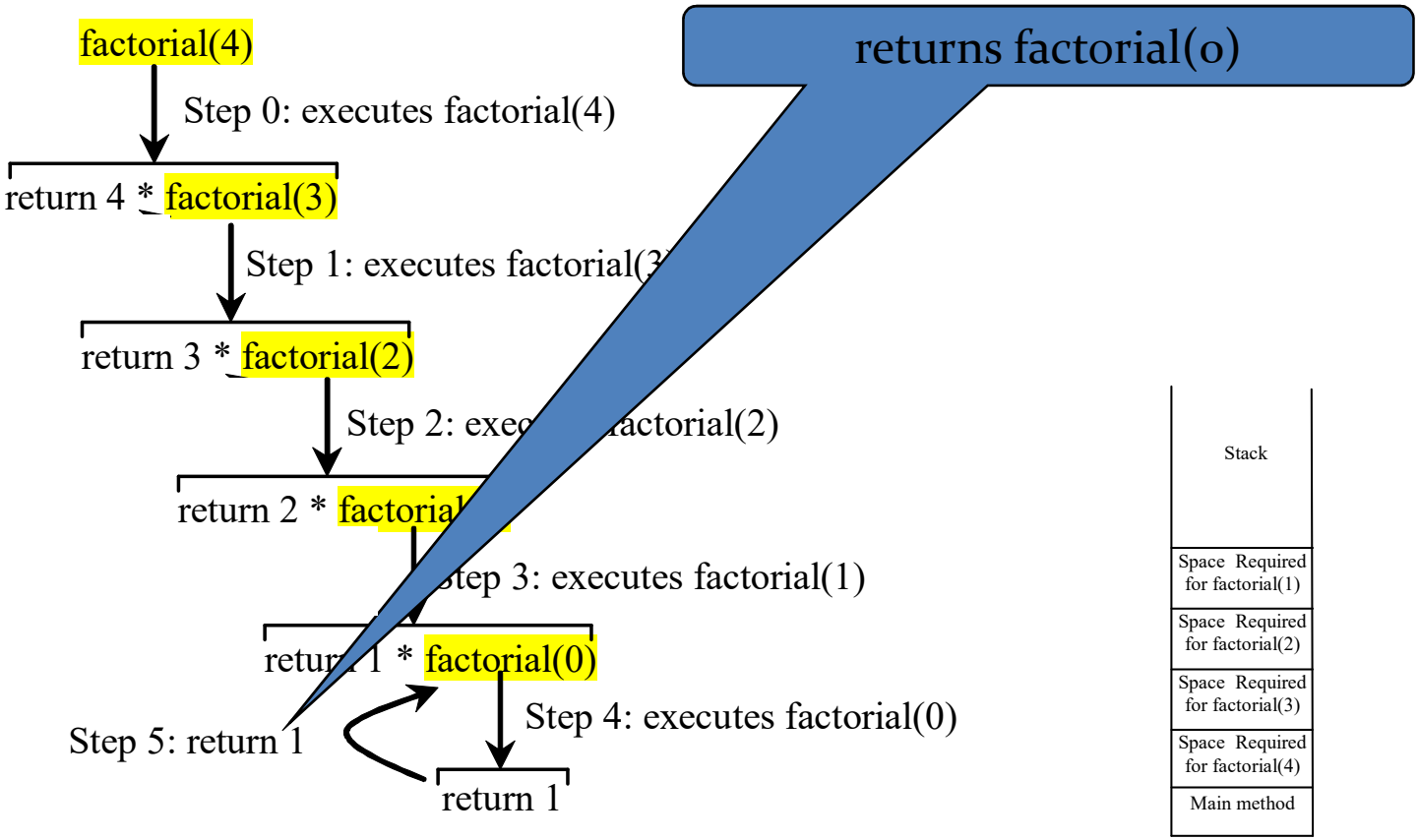
Recursive Factorial



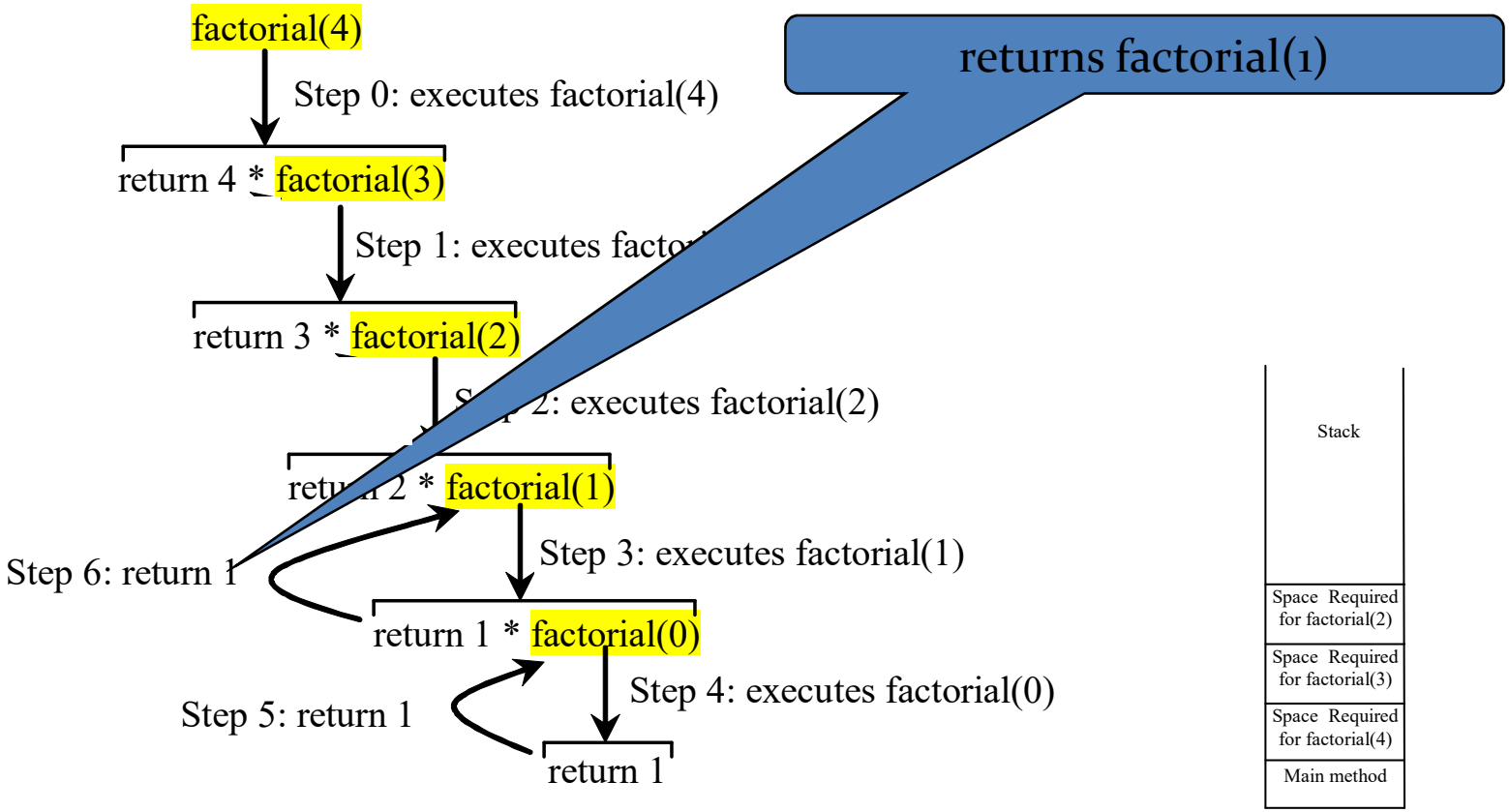
Recursive Factorial



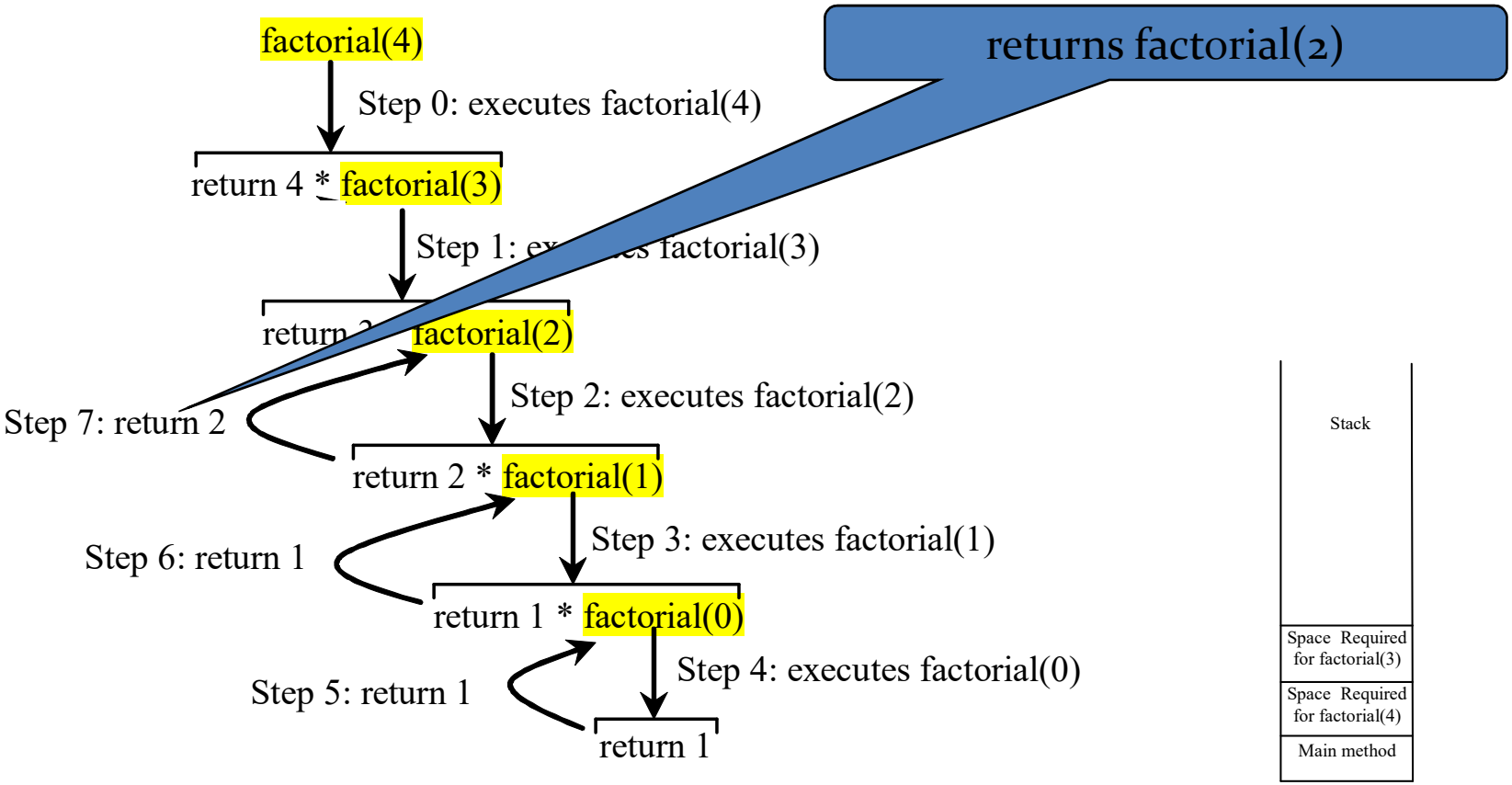
Recursive Factorial



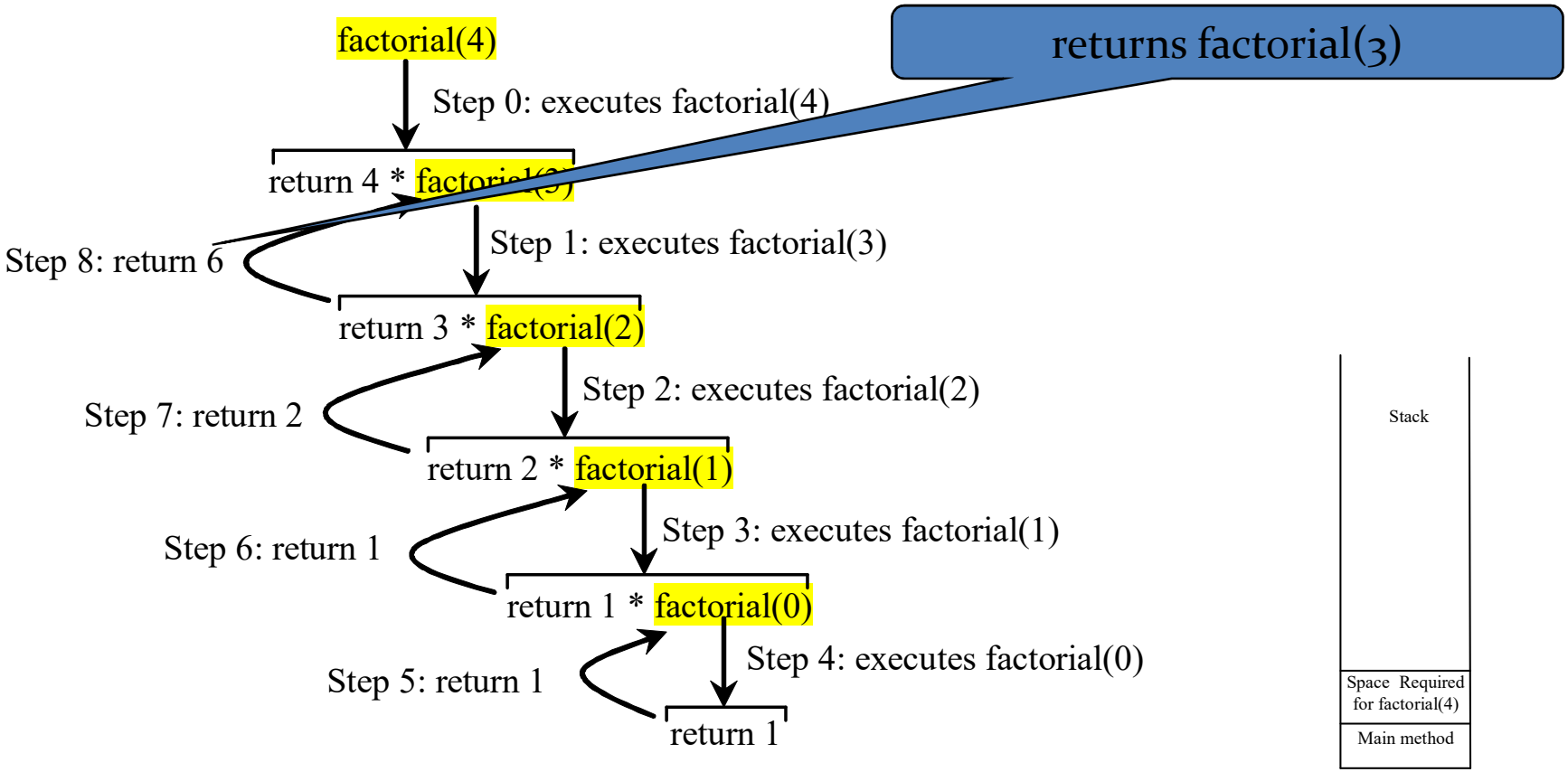
Recursive Factorial



Recursive Factorial



Recursive Factorial



Recursive Factorial