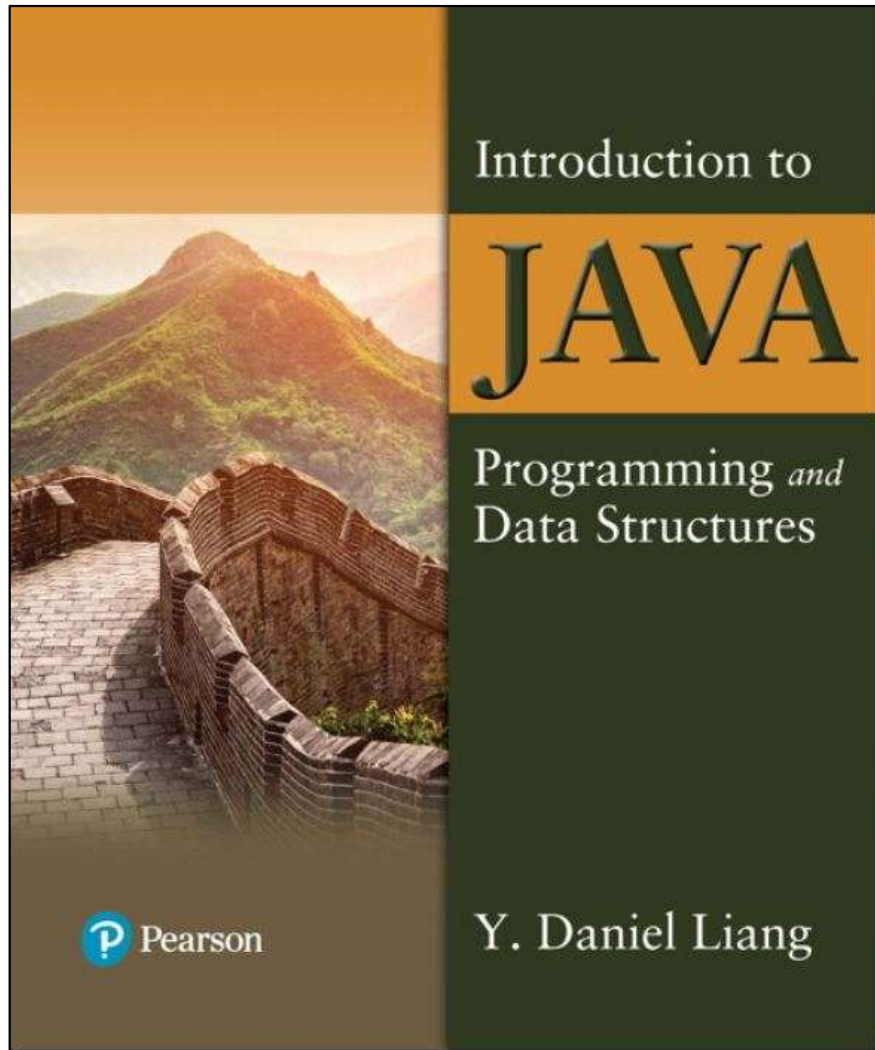


Introduction to Java Programming and Data Structures

Twelfth Edition



Chapter 18

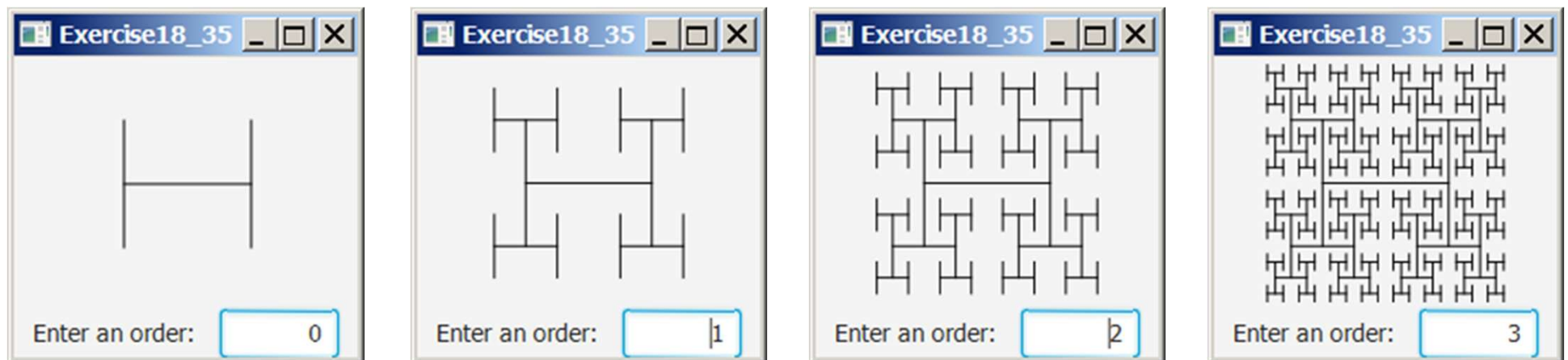
Recursion

Motivations (1 of 2)

Suppose you want to find all the files under a directory that contains a particular word. How do you solve this problem? There are several ways to solve this problem. An intuitive solution is to use recursion by searching the files in the subdirectories recursively.

Motivations (2 of 2)

H-trees, depicted in Figure 18.1, are used in a very large-scale integration (VLSI) design as a clock distribution network for routing timing signals to all parts of a chip with equal propagation delays. How do you write a program to display H-trees? A good approach is to use recursion.



Computing Factorial (1 of 11)

Mathematic notation:

$$n! = n * (n-1)!, n > 0$$

$$0! = 1$$

Function:

$$\text{factorial}(0) = 1;$$

$$\text{factorial}(n) = n * \text{factorial}(n-1); n > 0$$

[ComputeFactorial](#)

Computing Factorial (2 of 11)

$\text{factorial}(0) = 1;$

$\text{factorial}(n) = n * \text{factorial}(n - 1);$

$\text{factorial}(4)$

Computing Factorial (3 of 11)

$\text{factorial}(0) = 1;$

$\text{factorial}(n) = n * \text{factorial}(n - 1);$

$\text{factorial}(4) = 4 * \text{factorial}(3)$

Computing Factorial (4 of 11)

$\text{factorial}(0) = 1;$

$\text{factorial}(n) = n * \text{factorial}(n - 1);$

$$\begin{aligned}\text{factorial}(4) &= 4 * \text{factorial}(3) \\ &= 4 * 3 * \text{factorial}(2))\end{aligned}$$

Computing Factorial (5 of 11)

$$\text{factorial}(0) = 1;$$

$$\text{factorial}(n) = n * \text{factorial}(n - 1);$$

$$\begin{aligned}\text{factorial}(4) &= 4 * \text{factorial}(3) \\ &= 4 * 3 * \text{factorial}(2) \\ &= 4 * 3 * (2 * \text{factorial}(1))\end{aligned}$$

Computing Factorial (6 of 11)

$$\text{factorial}(0) = 1;$$

$$\text{factorial}(n) = n * \text{factorial}(n - 1);$$

$$\text{factorial}(4) = 4 * \text{factorial}(3)$$

$$= 4 * 3 * \text{factorial}(2)$$

$$= 4 * 3 * (2 * \text{factorial}(1))$$

$$= 4 * 3 * (2 * (1 * \text{factorial}(0)))$$

Computing Factorial (7 of 11)

$$\text{factorial}(0) = 1;$$

$$\text{factorial}(n) = n * \text{factorial}(n - 1);$$

$$\text{factorial}(4) = 4 * \text{factorial}(3)$$

$$= 4 * 3 * \text{factorial}(2)$$

$$= 4 * 3 * (2 * \text{factorial}(1))$$

$$= 4 * 3 * (2 * (1 * \text{factorial}(0)))$$

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

Computing Factorial (8 of 11)

$$\text{factorial}(0) = 1;$$

$$\text{factorial}(n) = n * \text{factorial}(n - 1);$$

$$\text{factorial}(4) = 4 * \text{factorial}(3)$$

$$= 4 * 3 * \text{factorial}(2)$$

$$= 4 * 3 * (2 * \text{factorial}(1))$$

$$= 4 * 3 * (2 * (1 * \text{factorial}(0)))$$

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

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

Computing Factorial (9 of 11)

$$\text{factorial}(0) = 1;$$

$$\text{factorial}(n) = n * \text{factorial}(n - 1);$$

$$\begin{aligned}\text{factorial}(4) &= 4 * \text{factorial}(3) \\ &= 4 * 3 * \text{factorial}(2) \\ &= 4 * 3 * (2 * \text{factorial}(1)) \\ &= 4 * 3 * (2 * (1 * \text{factorial}(0))) \\ &= 4 * 3 * (2 * (1 * 1)) \\ &= 4 * 3 * (2 * 1) \\ &= 4 * 3 * 2\end{aligned}$$

Computing Factorial (10 of 11)

$$\text{factorial}(0) = 1;$$

$$\text{factorial}(n) = n * \text{factorial}(n - 1);$$

$$\begin{aligned}\text{factorial}(4) &= 4 * \text{factorial}(3) \\ &= 4 * (3 * \text{factorial}(2)) \\ &= 4 * (3 * (2 * \text{factorial}(1))) \\ &= 4 * (3 * (2 * (1 * \text{factorial}(0)))) \\ &= 4 * (3 * (2 * (1 * 1))) \\ &= 4 * (3 * (2 * 1)) \\ &= 4 * (3 * 2) \\ &= 4 * (6)\end{aligned}$$

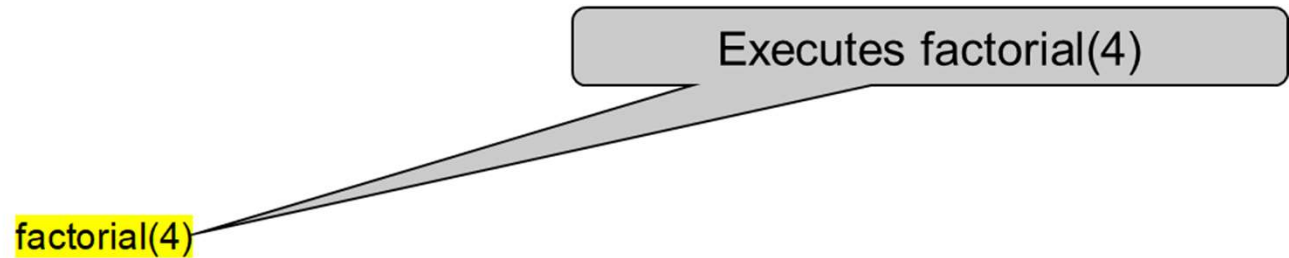
Computing Factorial (11 of 11)

$$\text{factorial}(0) = 1;$$

$$\text{factorial}(n) = n * \text{factorial}(n - 1);$$

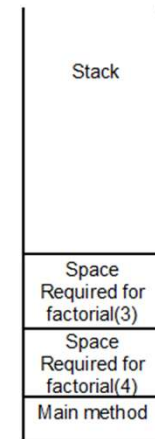
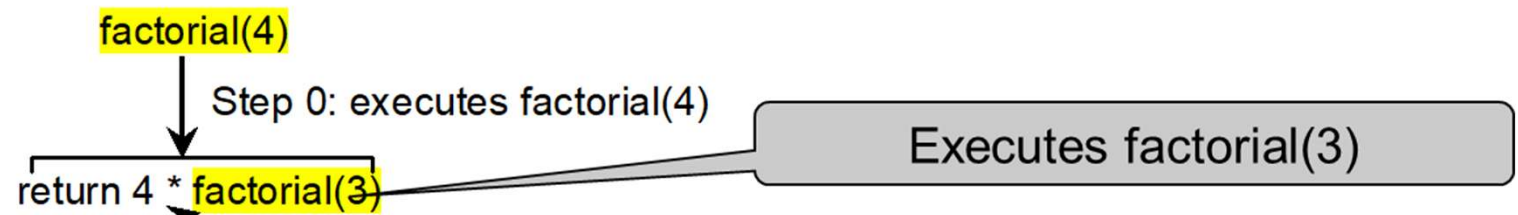
$$\begin{aligned}\text{factorial}(4) &= 4 * \text{factorial}(3) \\ &= 4 * (3 * \text{factorial}(2)) \\ &= 4 * (3 * (2 * \text{factorial}(1))) \\ &= 4 * (3 * (2 * (1 * \text{factorial}(0)))) \\ &= 4 * (3 * (2 * (1 * 1))) \\ &= 4 * (3 * (2 * 1)) \\ &= 4 * (3 * 2) \\ &= 4 * (6) \\ &= 24\end{aligned}$$

Trace Recursive Factorial (1 of 11)

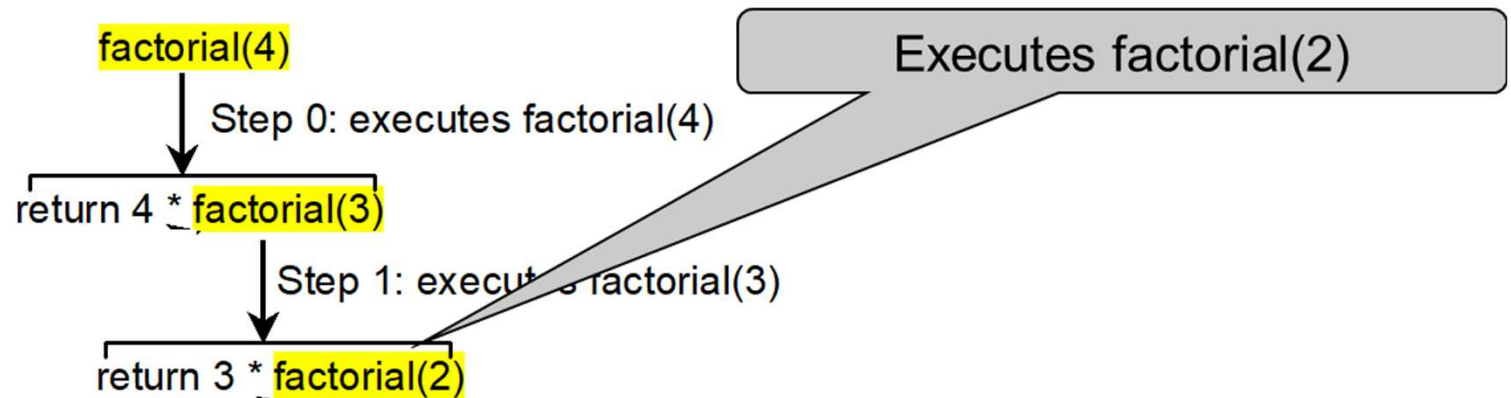


Stack
Space Required for factorial(4)
Main method

Trace Recursive Factorial (2 of 11)

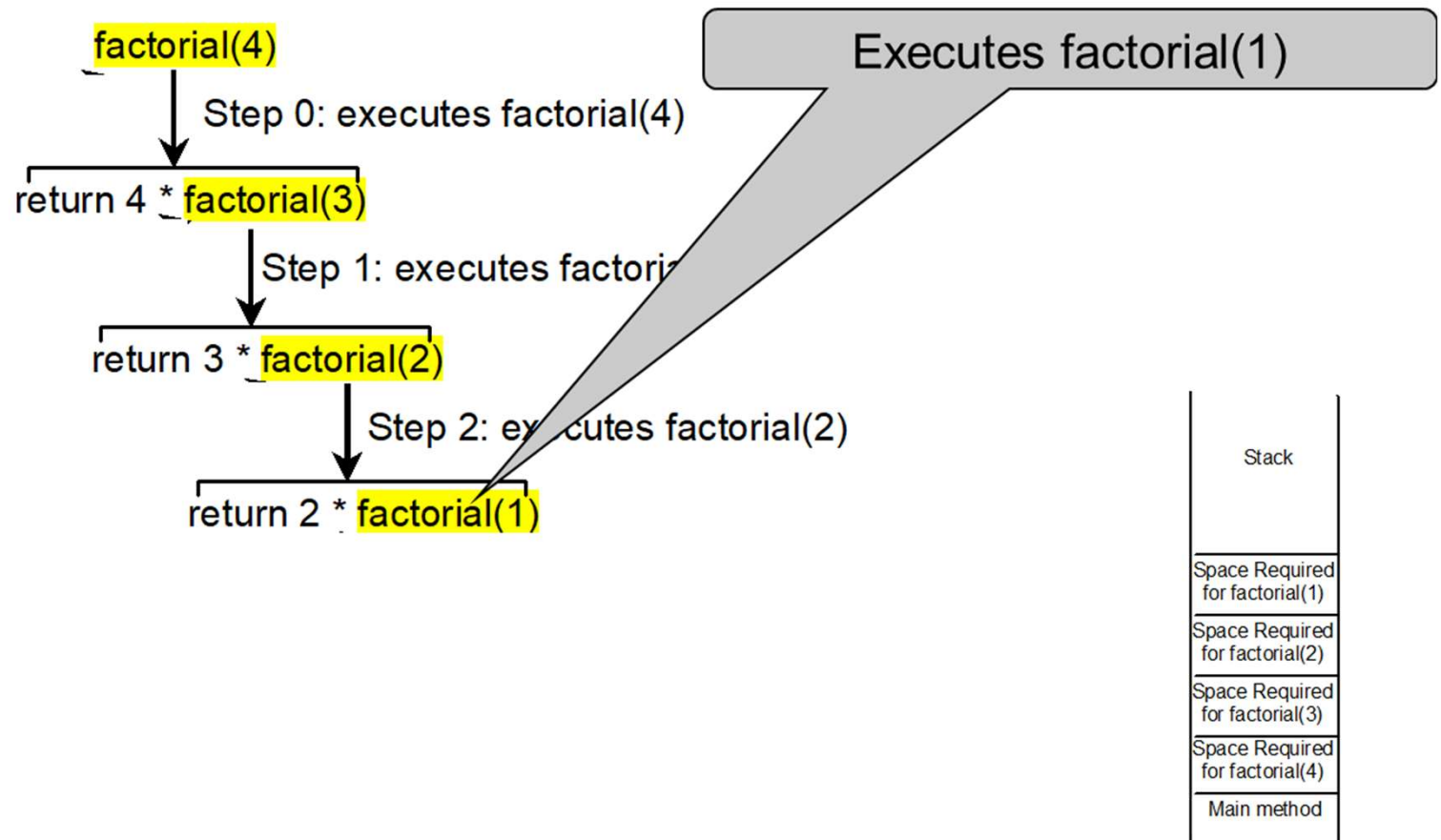


Trace Recursive Factorial (3 of 11)

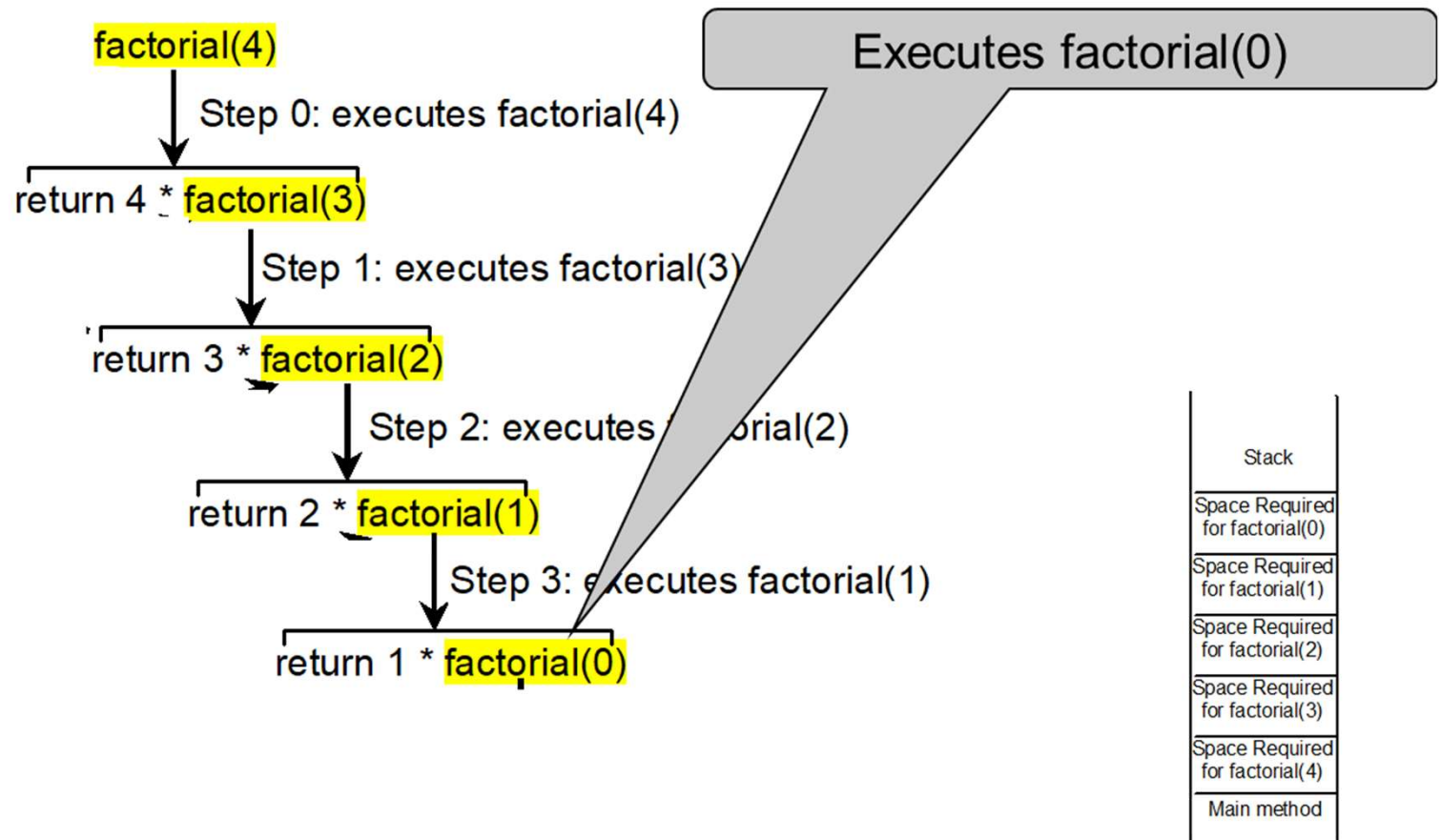


Stack
Space Required for factorial(2)
Space Required for factorial(3)
Space Required for factorial(4)
Main method

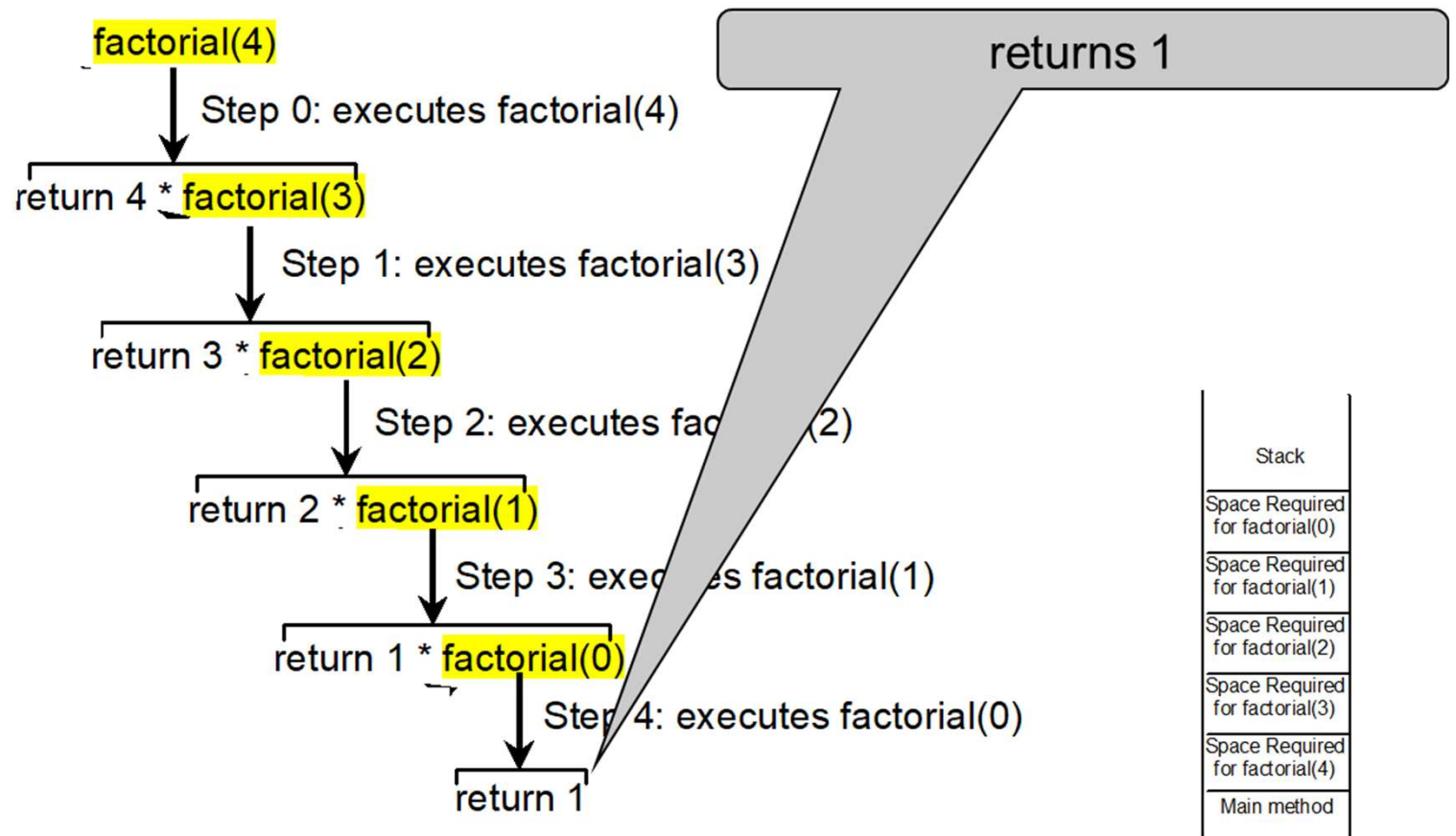
Trace Recursive Factorial (4 of 11)



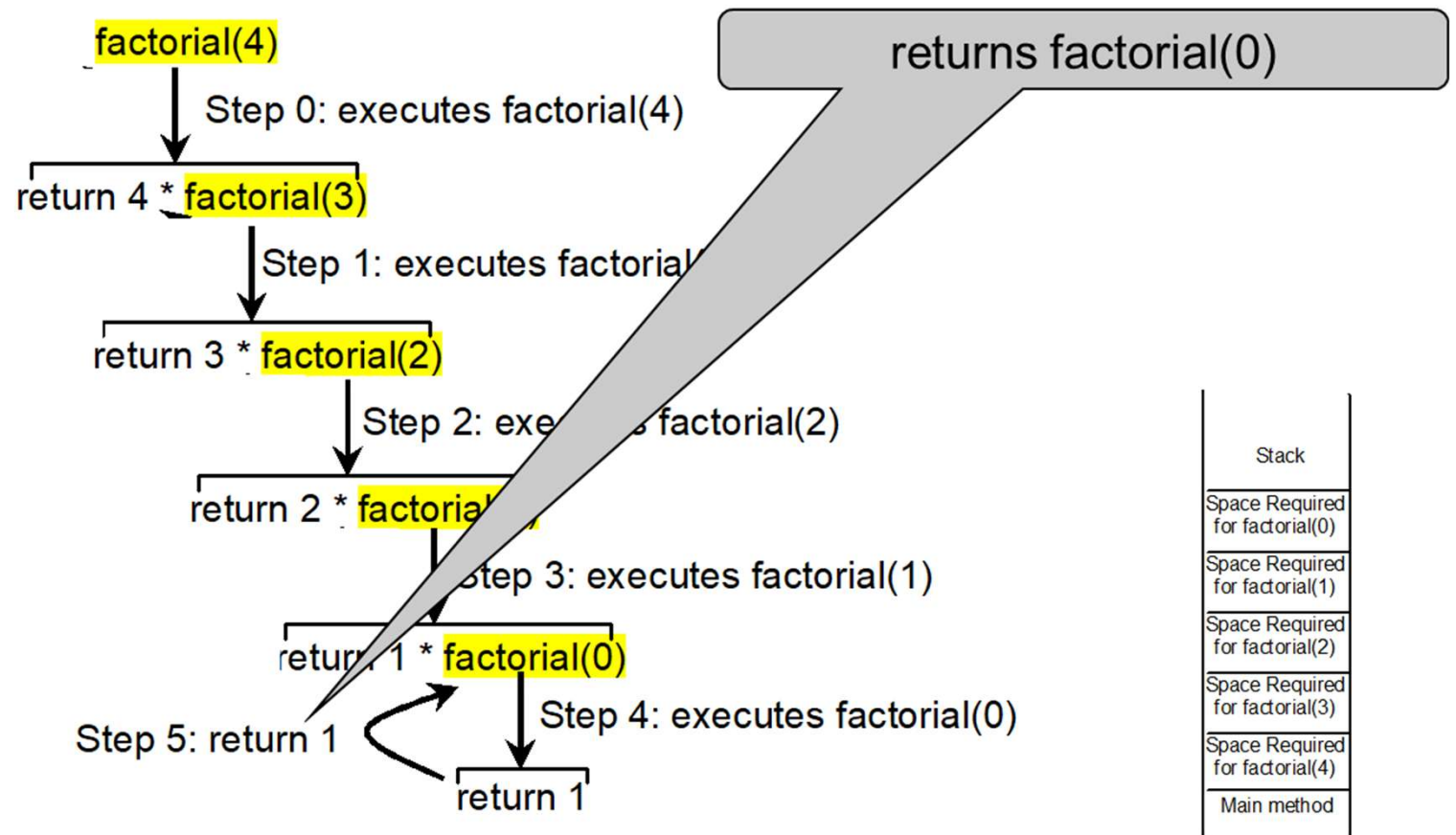
Trace Recursive Factorial (5 of 11)



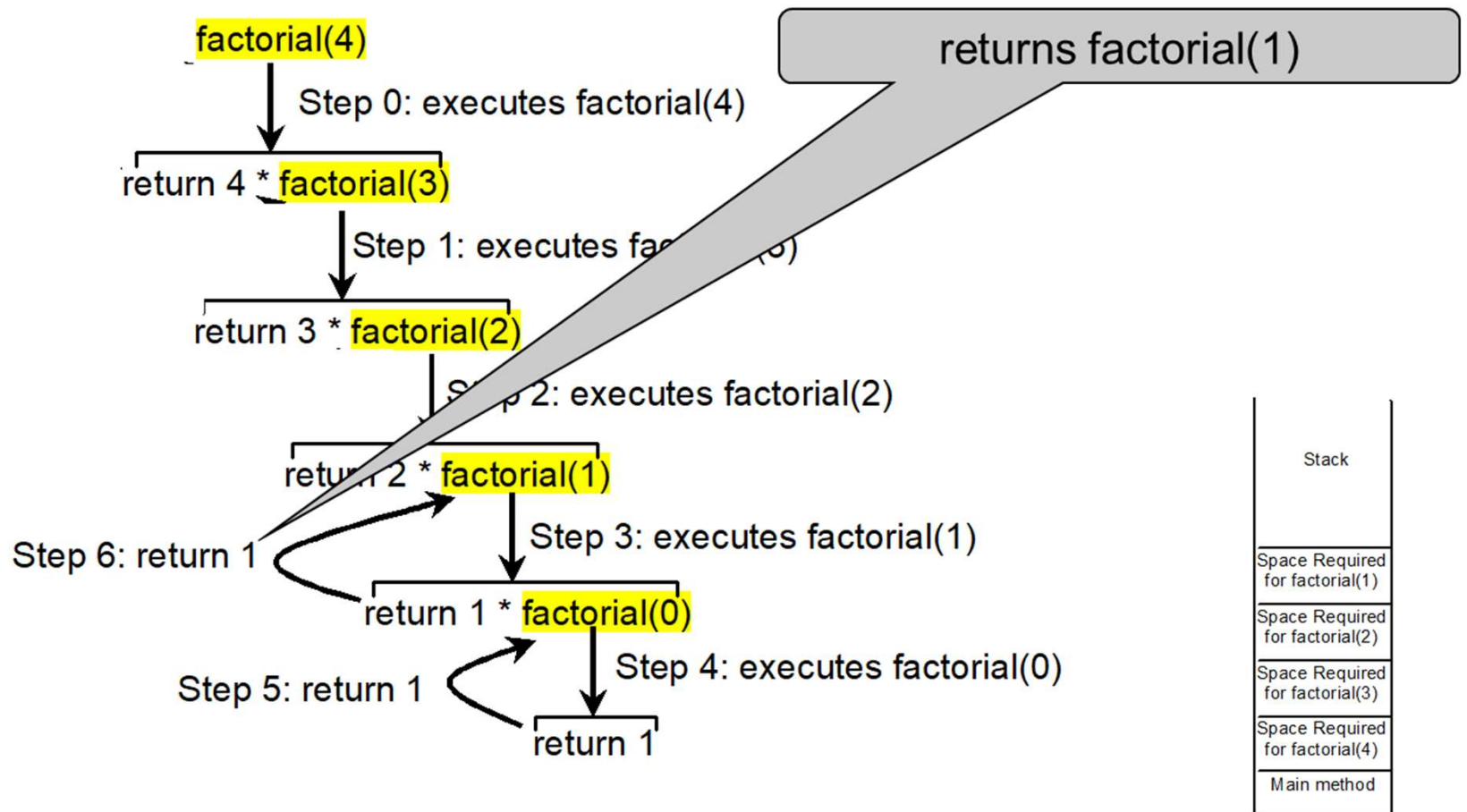
Trace Recursive Factorial (6 of 11)



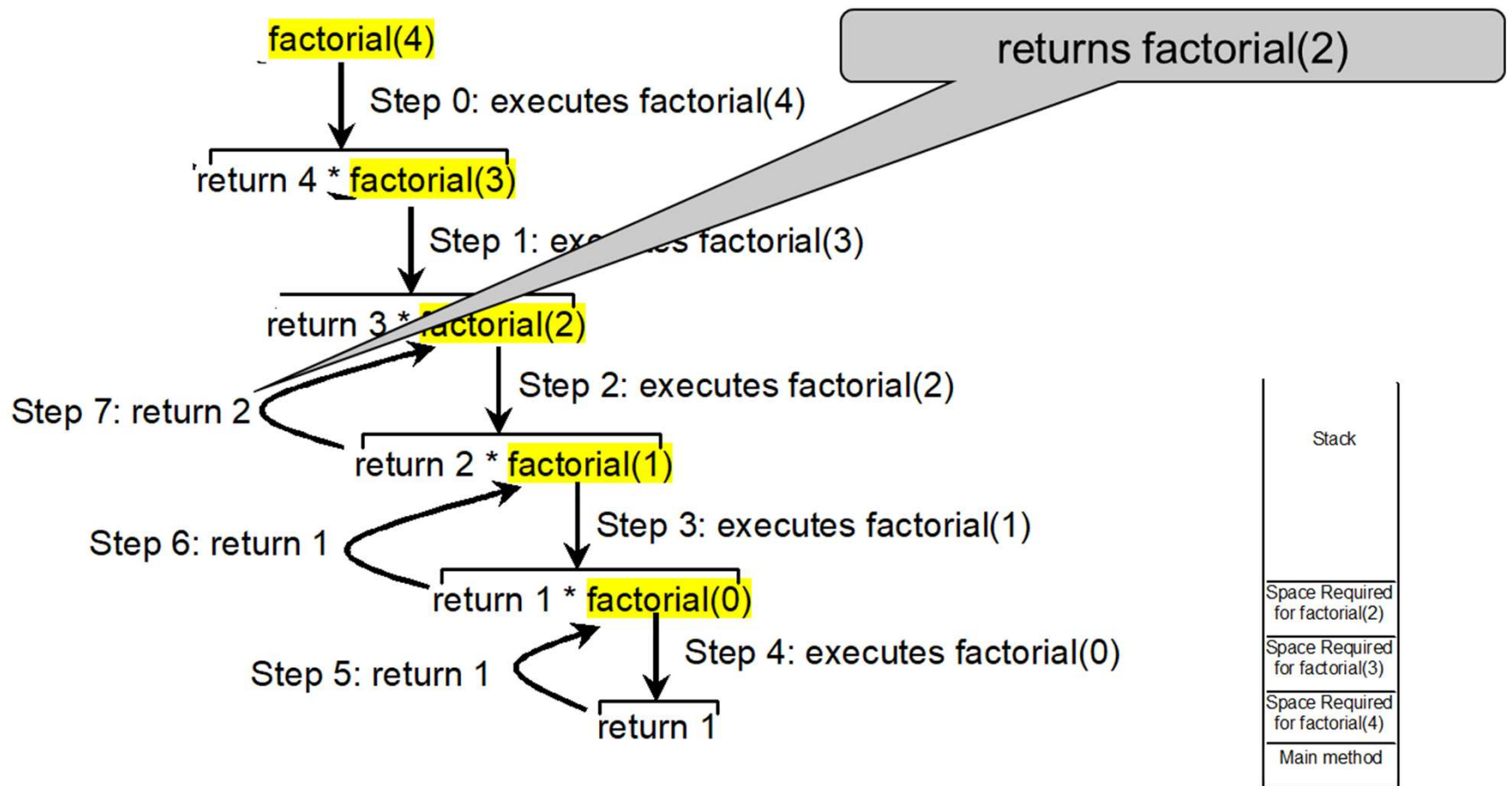
Trace Recursive Factorial (7 of 11)



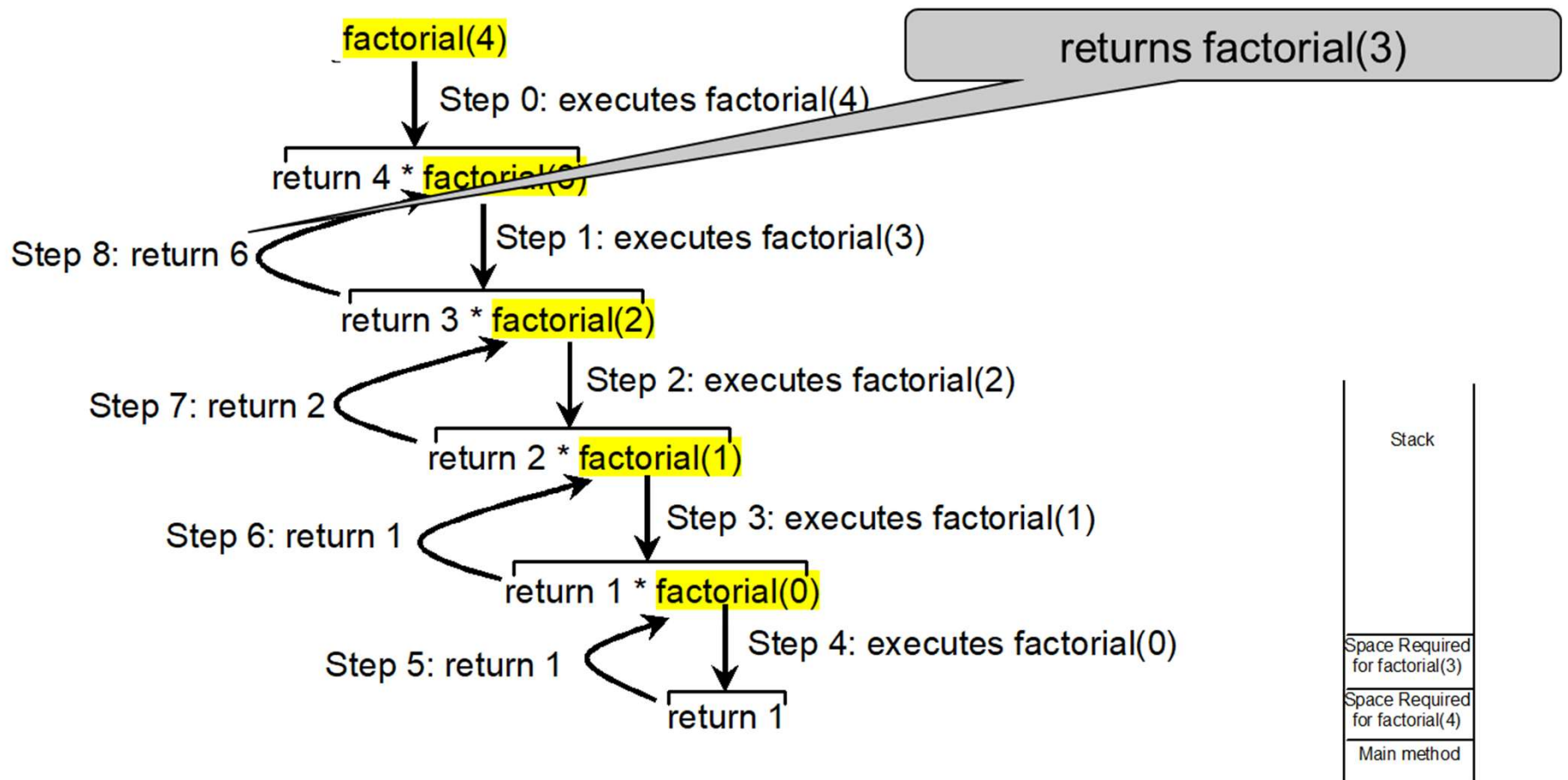
Trace Recursive Factorial (8 of 11)



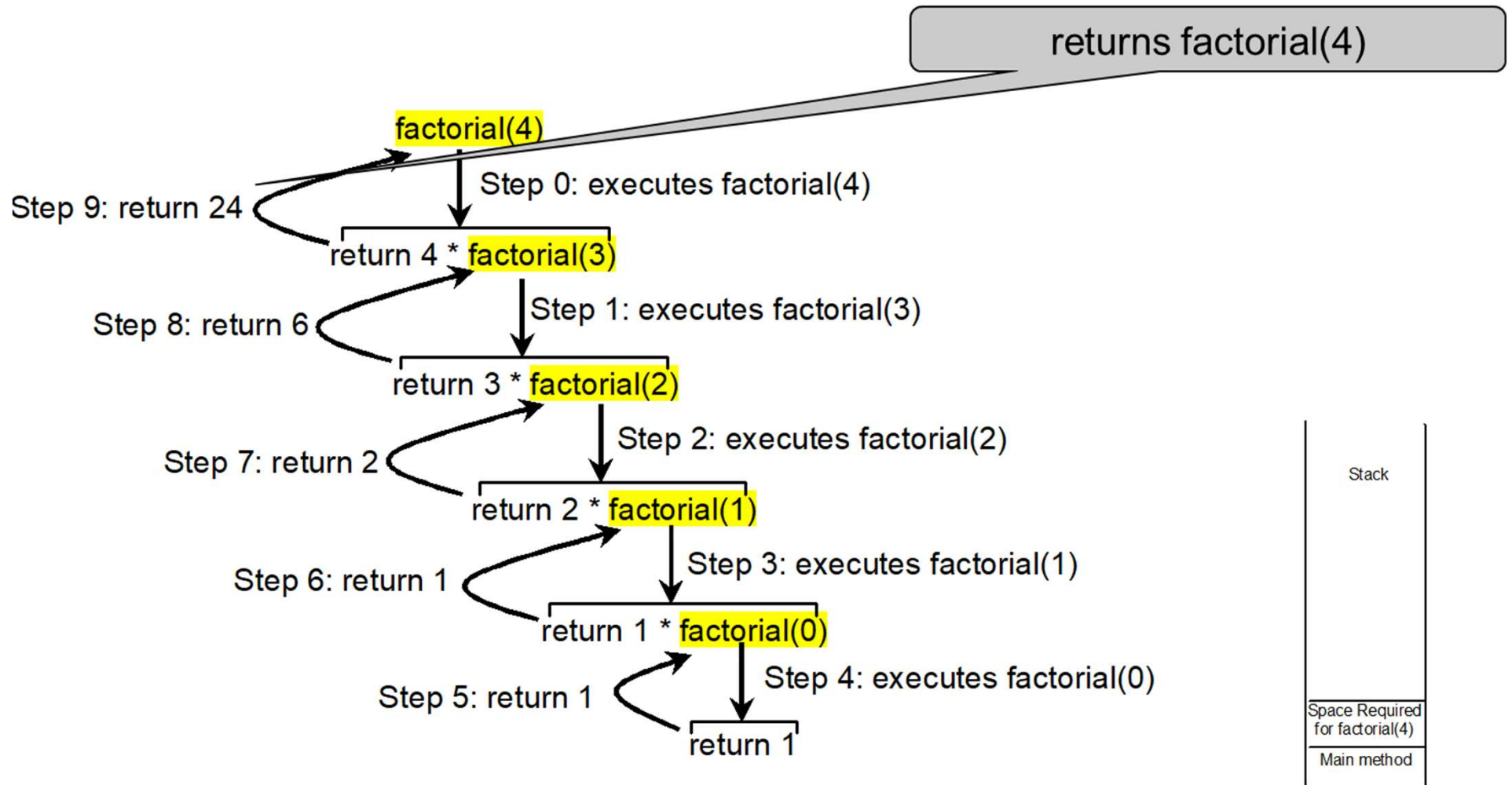
Trace Recursive Factorial (9 of 11)



Trace Recursive Factorial (10 of 11)



Trace Recursive Factorial (11 of 11)



Other Examples

$$f(0) = 0;$$

$$f(n) = n + f(n - 1);$$

Fibonacci Numbers (1 of 2)

Fibonacci series: 0 1 1 2 3 5 8 13 21 34 55
89...

indices: 0 1 2 3 4 5 6 7 8 9 10 11

$\text{fib}(0) = 0;$

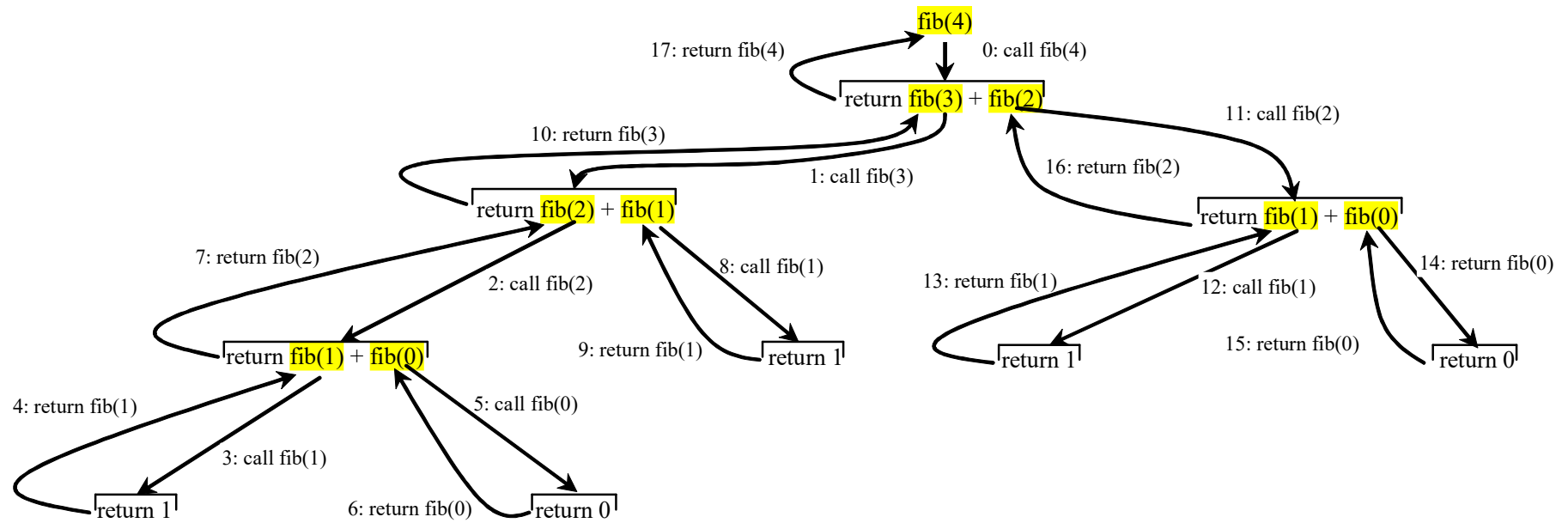
$\text{fib}(1) = 1;$

$\text{fib}(\text{index}) = \text{fib}(\text{index} - 1) + \text{fib}(\text{index} - 2); \text{index} \geq 2$

$\text{fib}(3) = \text{fib}(2) + \text{fib}(1) = (\text{fib}(1) + \text{fib}(0)) + \text{fib}(1) = (1 + 0)$
 $+ \text{fib}(1) = 1 + \text{fib}(1) = 1 + 1 = 2$

[ComputeFibonacci](#)

Fibonacci Numbers (2 of 2)



Problem Solving Using Recursion (1 of 2)

In general, to solve a problem using recursion, you break it into subproblems. If a subproblem resembles the original problem, you can apply the same approach to solve the subproblems recursively. A subproblem is almost the same as the original problem in nature with a smaller size.

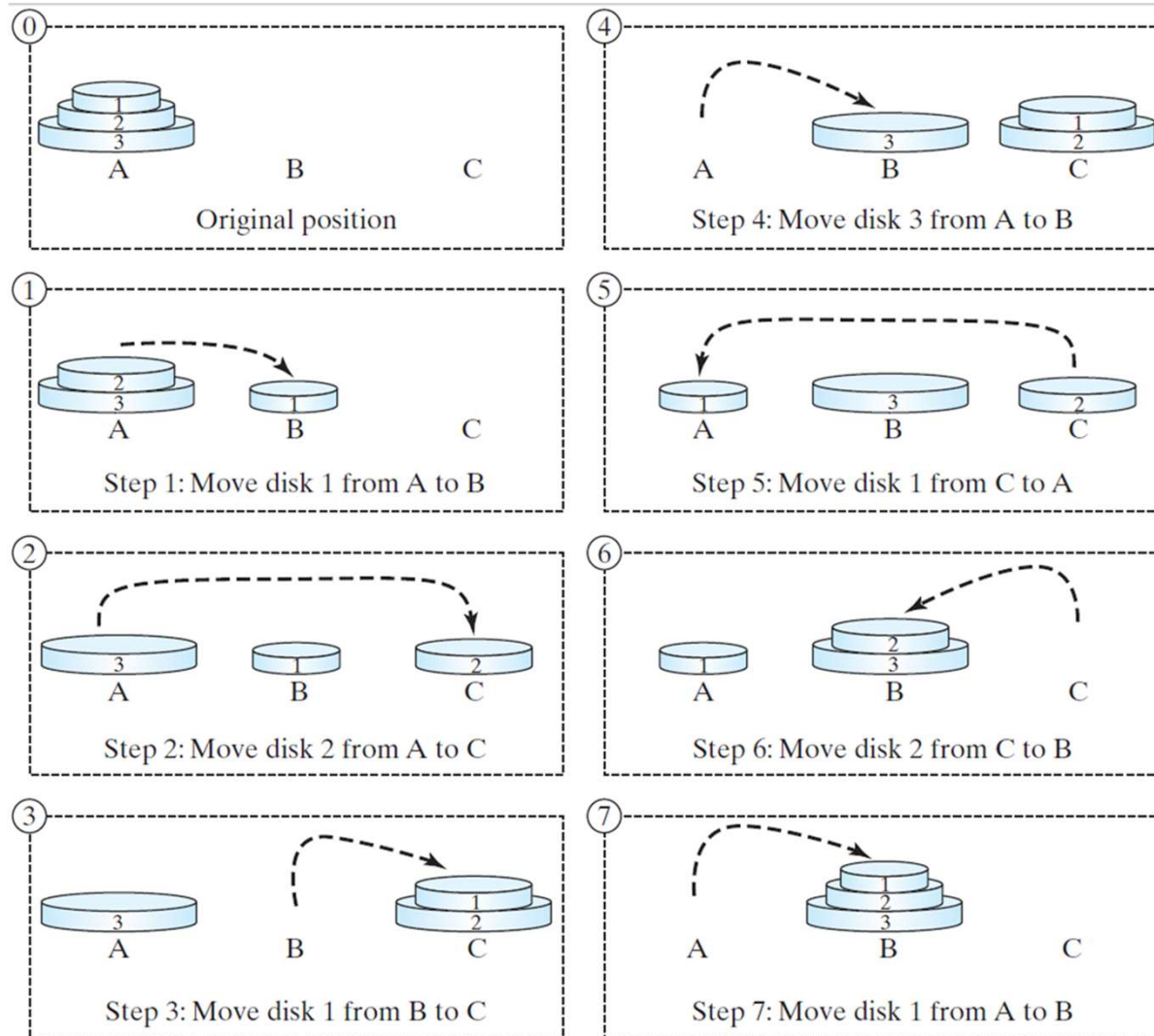
Characteristics of Recursion

- All recursive methods have the following characteristics:
 - The method is implemented using a conditional statement that leads to different cases.
 - One or more base cases (the simplest case) are used to stop recursion.
 - Every recursive call reduces the original problem, bringing it increasingly closer to a base case until it becomes that case.

Tower of Hanoi (1 of 2)

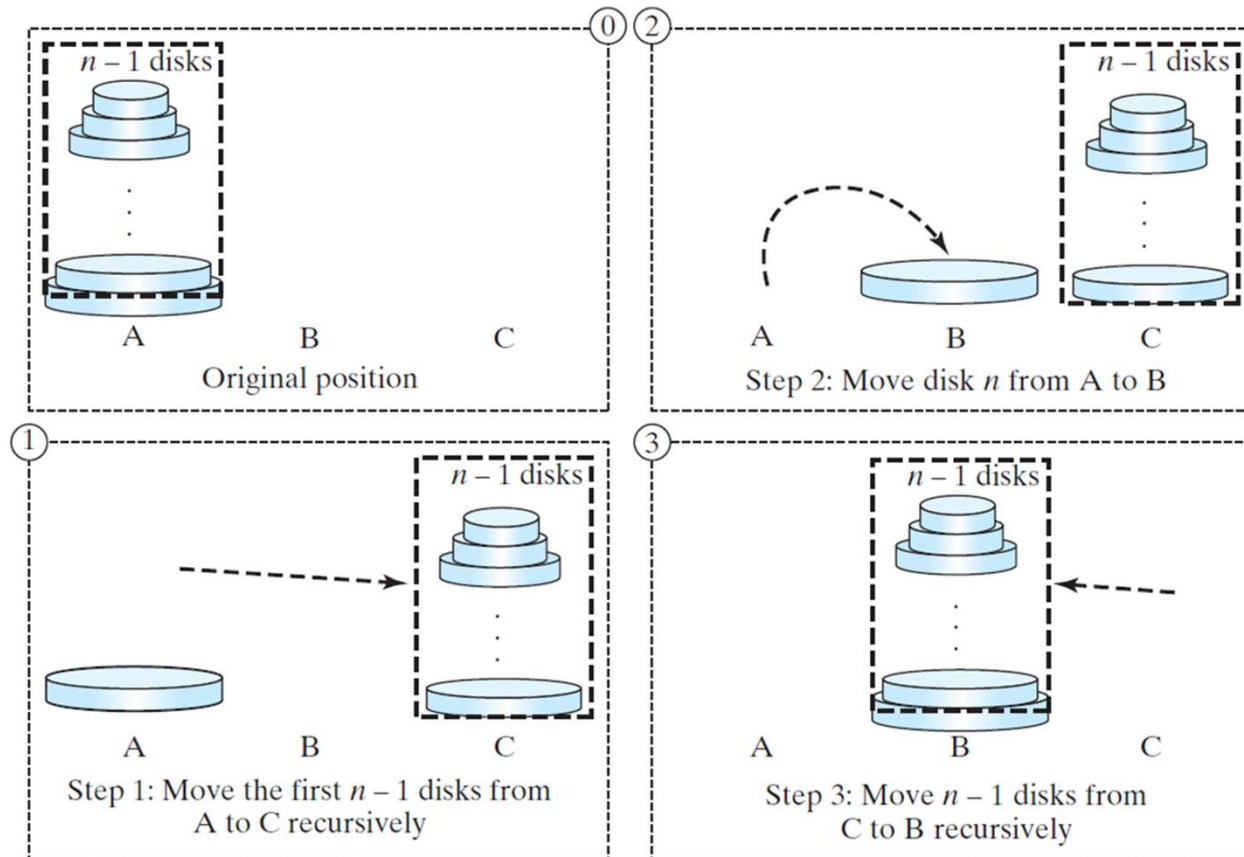
- There are n disks labeled 1, 2, 3, ..., n , and three towers labeled A, B, and C.
- No disk can be on top of a smaller disk at any time.
- All the disks are initially placed on tower A.
- Only one disk can be moved at a time, and it must be the top disk on the tower.

Tower of Hanoi (2 of 2)



Solution to Tower of Hanoi (1 of 2)

The Tower of Hanoi problem can be decomposed into three subproblems.



Solution to Tower of Hanoi (2 of 2)

- Move the first $n - 1$ disks from A to C with the assistance of tower B.
- Move disk n from A to B.
- Move $n - 1$ disks from C to B with the assistance of tower A.

[TowerOfHanoi](#)

Practice Now

- Factorial
- fibonacci

- Final Exam: **NO** online test option.
- Happy Thanksgiving.
- Last week, review everything from “Hello World”.