CISC 3115 TY3

C21a: Recursion

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

- Motivation: finding file
- Base case and recursive subproblem
- Simple examples using math functions
- Call stacks during recursive method calls

Motivations: Finding File

- Problem: write a program that finds all the files under a directory that contains a particular word.

Motivations: Finding File: Observation 1: java.io.File

- Problem: write a program that finds all the files under a directory that contains a particular word.
- Observation
  - What have discussed about File I/O?
    - java.io.File
  - The directory may contain a subdirectory that may contain a subdirectory that may contain …

The java.io.File API

java.io.File	
`+File(pathname: String)`	Creates a `File` object for the specified path name. The path name may be a directory or a file.
`+File(parent: String, child: String)`	Creates a `File` object for the child under the directory parent. The child may be a file name or a subdirectory.
`+File(parent: File, child: String)`	Creates a `File` object for the child under the directory parent. The parent is a `File` object. In the preceding constructor, the parent is a string.
`+exists(): boolean`	Returns true if the file or the directory represented by the `File` object exists.
`+canRead(): boolean`	Returns true if the file represented by the `File` object exists and can be read.
`+canWrite(): boolean`	Returns true if the file represented by the `File` object exists and can be written.
`+isDirectory(): boolean`	Returns true if the `File` object represents a directory.
`+isFile(): boolean`	Returns true if the `File` object represents a file.
`+isAbsolute(): boolean`	Returns true if the `File` object is created using an absolute path name.
`+isHidden(): boolean`	Returns true if the file represented in the `File` object is hidden. The exact definition of *hidden* is system-dependent. On Windows, you can mark a file hidden in the File Properties dialog box. On Unix systems, a file is hidden if its name begins with a period(.) character.
`+getAbsolutePath(): String`	Returns the complete absolute file or directory name represented by the `File` object.
`+getCanonicalPath(): String`	Returns the same as `getAbsolutePath()` except that it removes redundant names, such as "." and "..", from the path name, resolves symbolic links (on Unix), and converts drive letters to standard uppercase (on Windows).
`+getName(): String`	Returns the last name of the complete directory and file name represented by the `File` object. For example, new `File("c:\\book\\test.dat").getName()` returns `test.dat`.
`+getPath(): String`	Returns the complete directory and file name represented by the `File` object. For example, new `File("c:\\book\\test.dat").getPath()` returns `c:\book\test.dat`.
`+getParent(): String`	Returns the complete parent directory of the current directory or the file represented by the `File` object. For example, new `File("c:\\book\\test.dat").getParent()` returns `c:\book`.
`+lastModified(): long`	Returns the time that the file was last modified.
`+length(): long`	Returns the size of the file, or 0 if it does not exist or if it is a directory.
`+listFile(): File[]`	Returns the files under the directory for a directory `File` object.
`+delete(): boolean`	Deletes the file or directory represented by this `File` object.The method returns true if the deletion succeeds.
`+renameTo(dest: File): boolean`	Renames the file or directory represented by this `File` object to the specified name represented in dest. The method returns true if the operation succeeds.
`+mkdir(): boolean`	Creates a directory represented in this `File` object. Returns true if the the directory is created successfully.
`+mkdirs(): boolean`	Same as `mkdir()` except that it creates directory along with its parent directories if the parent directories do not exist.

Motivations: Finding File: First Try

- List files

```
File[] fileArray = dir.listFiles();
for (File f: fileArray) {
   if (containWord(f, "for")) {
      System.out.println(f.toString());
   }
}
```

Motivations: Finding File: Observation 2: Subdirectories

- Problem: write a program that finds all the files under a directory that contains a particular word.
- Observation
  - What have discussed about File I/O?
    - java.io.File
  - The directory may contain a subdirectory
  - What if we also want to list all the files that contains the word under the subdirectory

Motivation: Finding File: Second Try

- List files also in the subdirectory

```
File[] fileArray = dir.listFiles();
for (File f: fileArray) {
  if (f.isDirectory()) {
    listFilesNextLevel(f, target);
  } else {
    if (containWord(f, target)) {
      System.out.println(f.toString());
    }
  }
}
```

Motivations: Finding File: Observation 3: Subdirectories

- Problem: write a program that finds all the files under a directory that contains a particular word.
- Observation
  - What have discussed about File I/O?
    - java.io.File
  - The directory may contain a subdirectory that may contain a subdirectory that may contain a subdirectory that may …
  - What if we also want to list all the files that contains the word under any of those subdirectories.

listFile and listFileNextLevel: Observation (Observation 4)?

listFile

```
File[] fileArray = dir.listFiles();
for (File f: fileArray) {
     if (f.isDirectory()) {
       listFilesNextLevel(f, target);
     } else {
       if (containWord(f, target)) {
         System.out.println(f.toString());
       }
     }
}
```

listFileNextLevel

```
File[] fileArray = dir.listFiles();
for (File f: fileArray) {
     if (f.isDirectory()) {
       // do nothing
     } else {
       if (containWord(f, target)) {
         System.out.println(f.toString());
       }
     }
}
```

Finding Files: Solution using Recursion

- listFile and listFileNextFile are in effect identical: we should apply the same method to all the directories, which is actually an example of recursion.
- Finding files using recursion in the nut shell

```
void listFiles(File dir, String target) {
  File[] fileArray = dir.listFiles();
  for (File f: fileArray) {
     if (f.isDirectory()) {
        listFiles (f, target);
     } else {
        // deal with file
     }
  }
}
```

Finding Files: Solution using Recursion: Implementation

- An example implementation

```
void listFiles(File dir, String target) throws FileNotFoundException {
    File[] fileArray = dir.listFiles();
    for (File f: fileArray) {
      if (f.isDirectory()) {
        listFiles(f, target);
      } else {
        if (f.canRead() && containWord(f, target)) {
          System.out.println(f.toString());
        }
      }
    }
  }
```

Concept of Recursion

- Concept of recursion
  - To use recursion here is to program using recursive method, i.e., to use methods that invoke themselves
  - Example
    - listFiles invokes listFiles itself

Recursion: Remark

- Recursion is a problem solving approach
  - a divide-and-conquer approach
  - where we divide a large problem into problems of the same nature but smaller size.
- Example
  - Problem: find a word in files recursively in a directory
    - The directory may contain a subdirectory. To find the word in files in the subdirectory is identical to find a word in the directory
    - The subdirectory may contain another subdirectory …

Questions

- Motivation for recursion
- Concept of recursion and recursive method
- More examples follow …

Problem Solving Using Recursion

- Many problems can be divided into smaller but the same problem of smaller size.
- Examples
  - Recursive mathematical functions, e.g., Factorials, Fibonacci Numbers
  - Sorting
  - Searching
  - …

Case Studies: Recursive Mathematical Functions

- Take a look at two recursive mathematical functions
  - Factorials
  - Fibonacci numbers
- Important concepts
  - Base case
  - Recursive calls and call stack
  - Performance implications

Factorials

- Factorials
  - f(n) = n! = n (n-1) (n-2) … 1
- which can be written as a recursive mathematical function
  - f(n) = n! = n (n-1)! = n f(n-1)
- That is
  - f(n) = n f(n-1)
  - f(0) = 1
- where f(0) = 1 (or 0! = 1) is the base case.

Base Case

- Base case is important
- Otherwise, where do we stop (without the base case)? e.g., consider
- f(3) = 3! = 3 f(2) = 3 2 f(1) = 3 2 1 f(0) = 3 2 1 0 f(-1) = 3 2 1 0 -1 f(-2) …
- The base case makes sure that we stop the recursive process somewhere.

Design Factorial Recursive Method

- Observe:
  - Recursive function: f(n) = n * f(n-1) when n > 0
  - Bae case: f(0) = 1
- Design method Factorial(n: int):
  - f(n) = n * f(n-1): when computing f(n), we invoke Factorial(n) where we compute n * Factorial(n-1) where we invoke the Factorial method recursively.
  - f(0) = 1: we stop invoking the Factorial method when n is 0.

Fibonacci Numbers

- Fibonacci numbers
  - Recurrence function

    f(n) = f(n-1) + f(n-2) when n>=2
  - Base case

    f(0) = 0

    f(1) = 1

Design Fibonacci Recursive Method

- fibonacci(n) is computed as fibonacci(n-1)*fibonacci(n-2) when n>=2 based on
  - Recurrence function

    f(n) = f(n-1) + f(n-2) when n>=2
- fibonacci(0) should return 0 and fibonacci(1) should return 1 according to
  - Base case

    f(0) = 0

    f(1) = 1

Recursive Calls and Call Stack

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

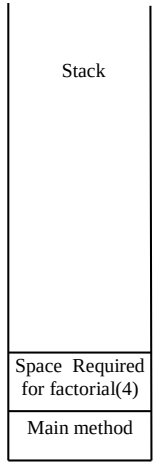
Observe the animation from the publisher and the author of the textbook.

animation

Trace Recursive factorial

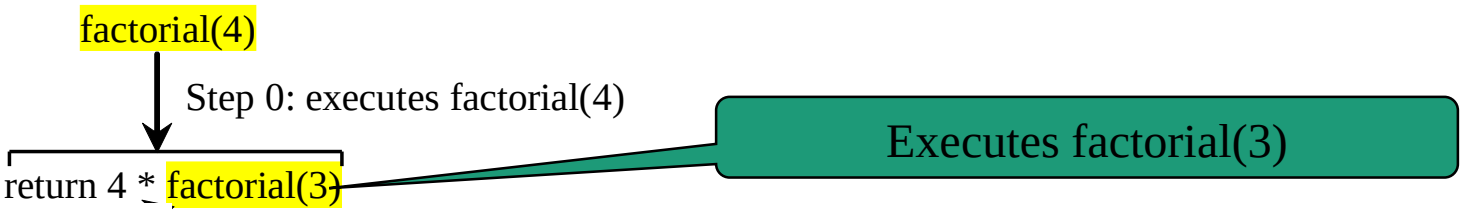
Executes factorial(4)

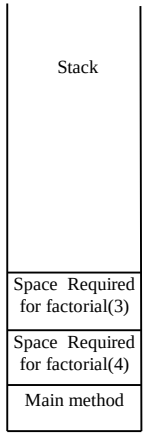
factorial(4)



animation

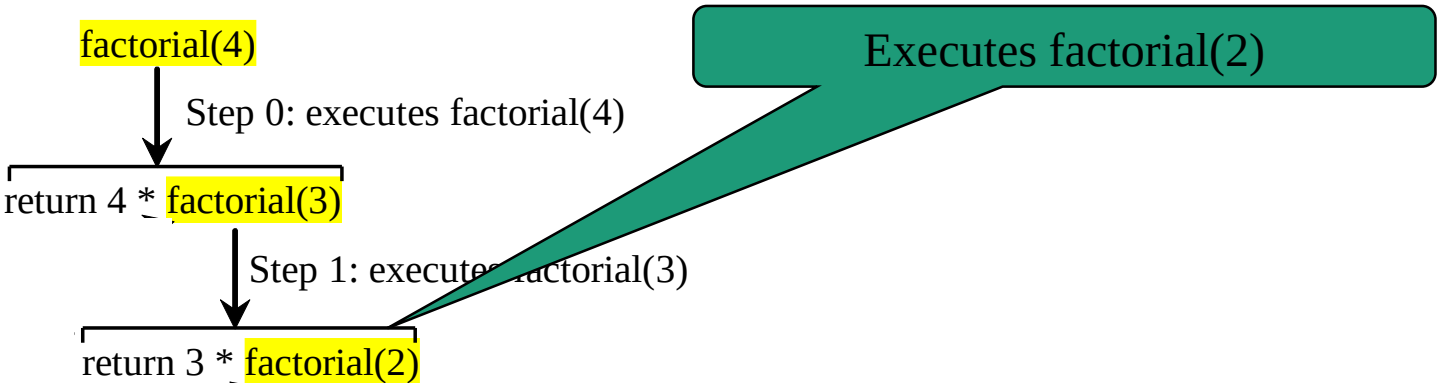
Trace Recursive factorial

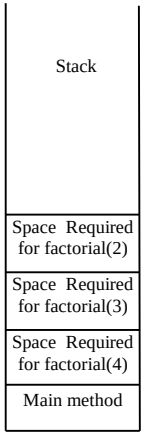




animation

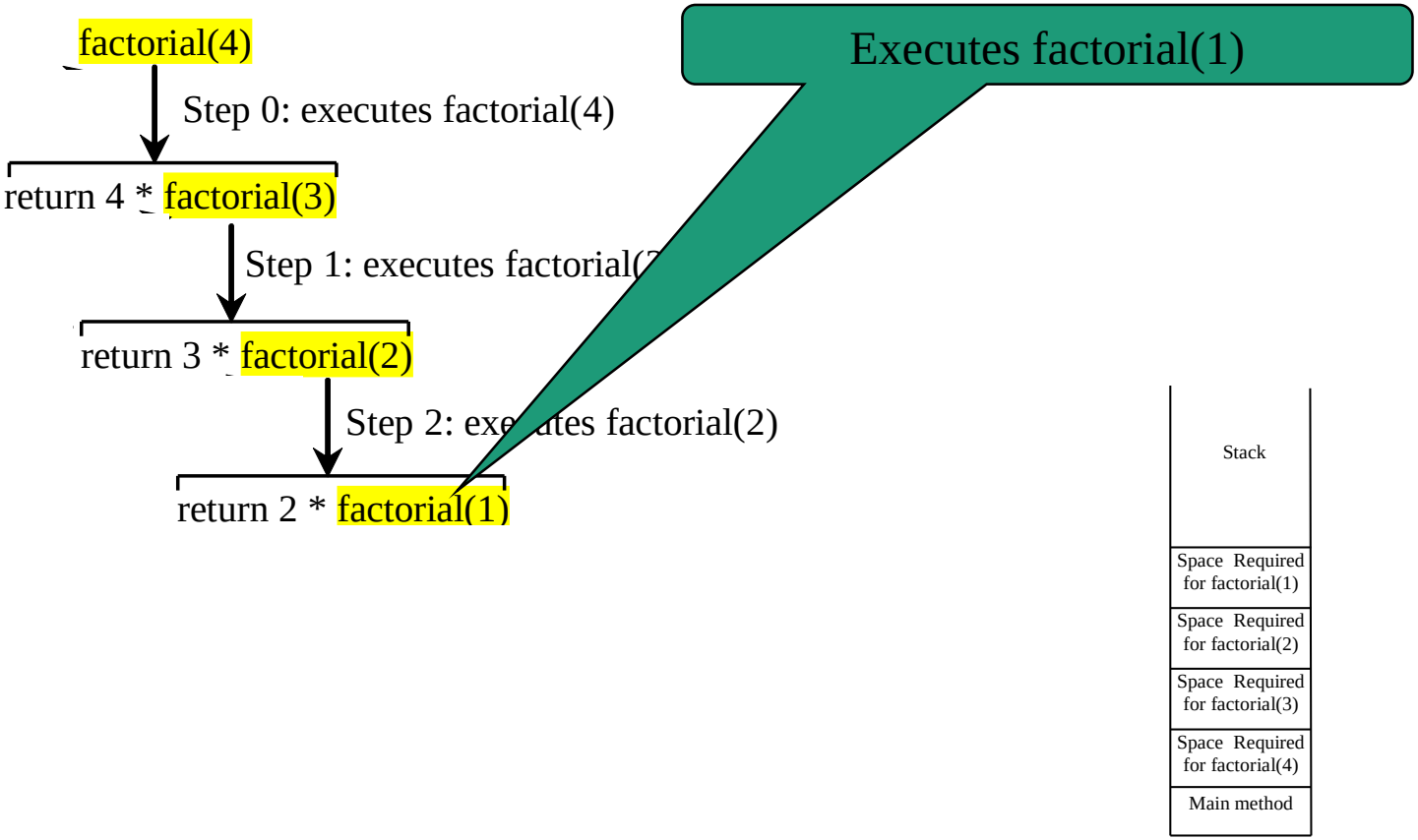
Trace Recursive factorial





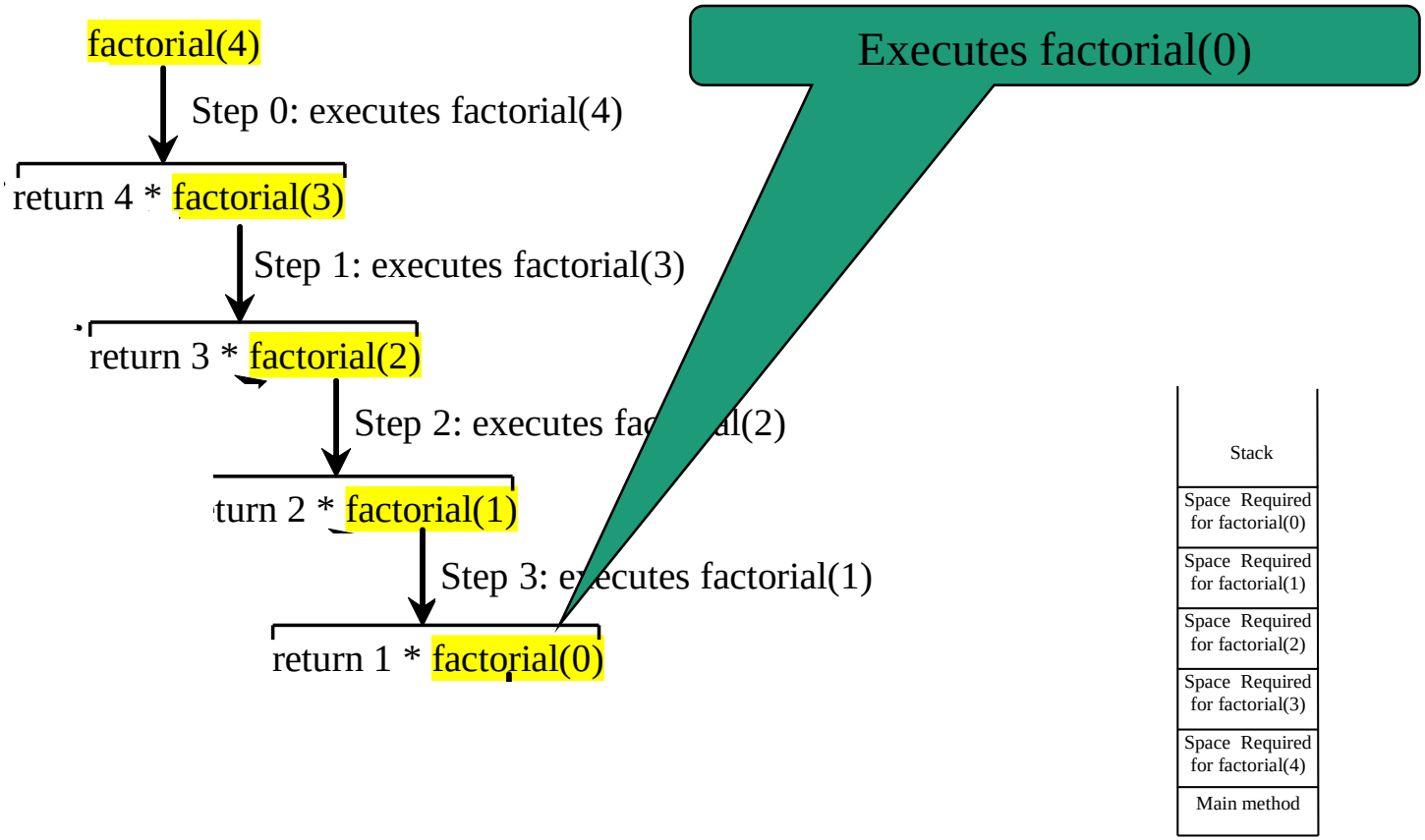
animation

Trace Recursive factorial



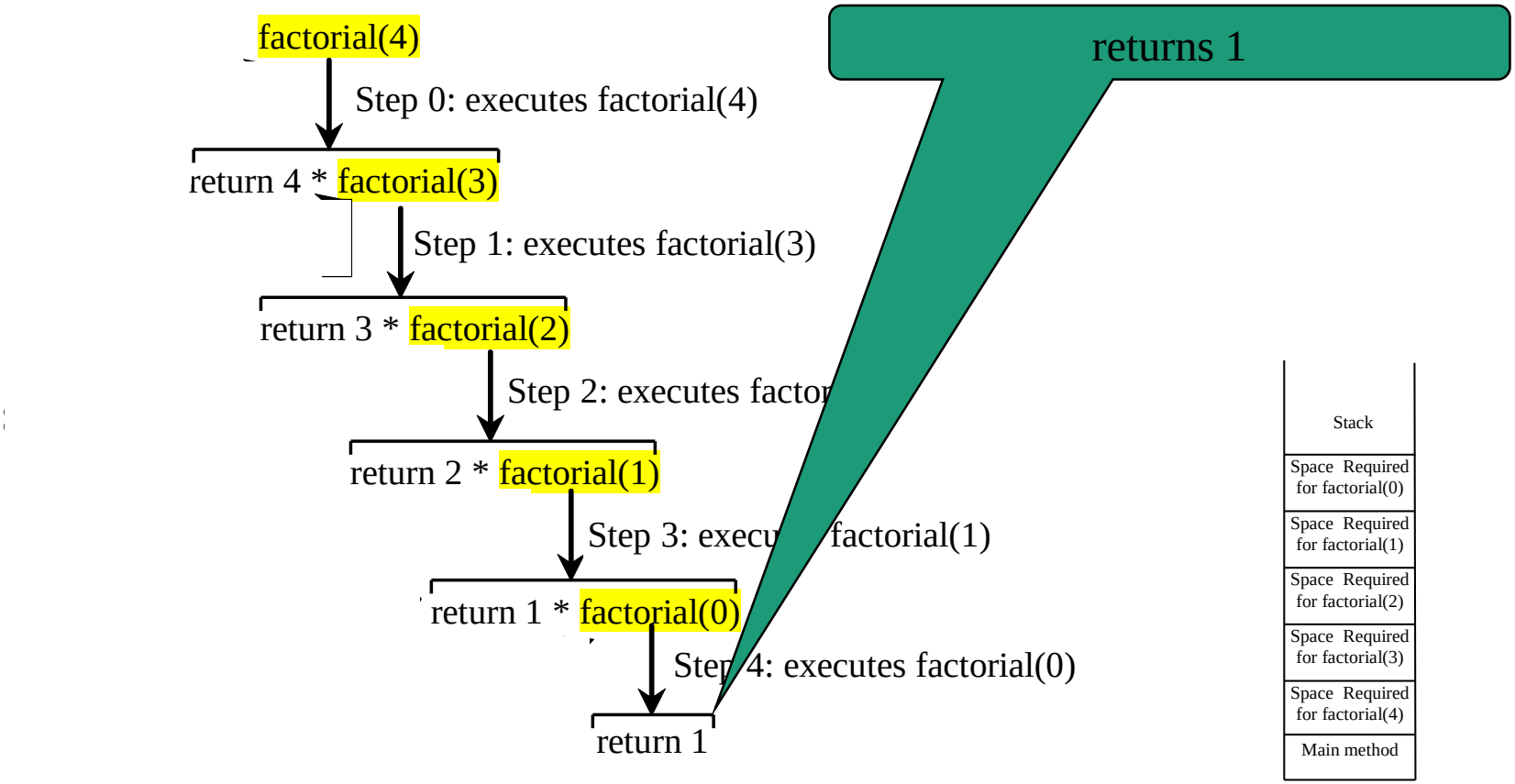
animation

Trace Recursive factorial



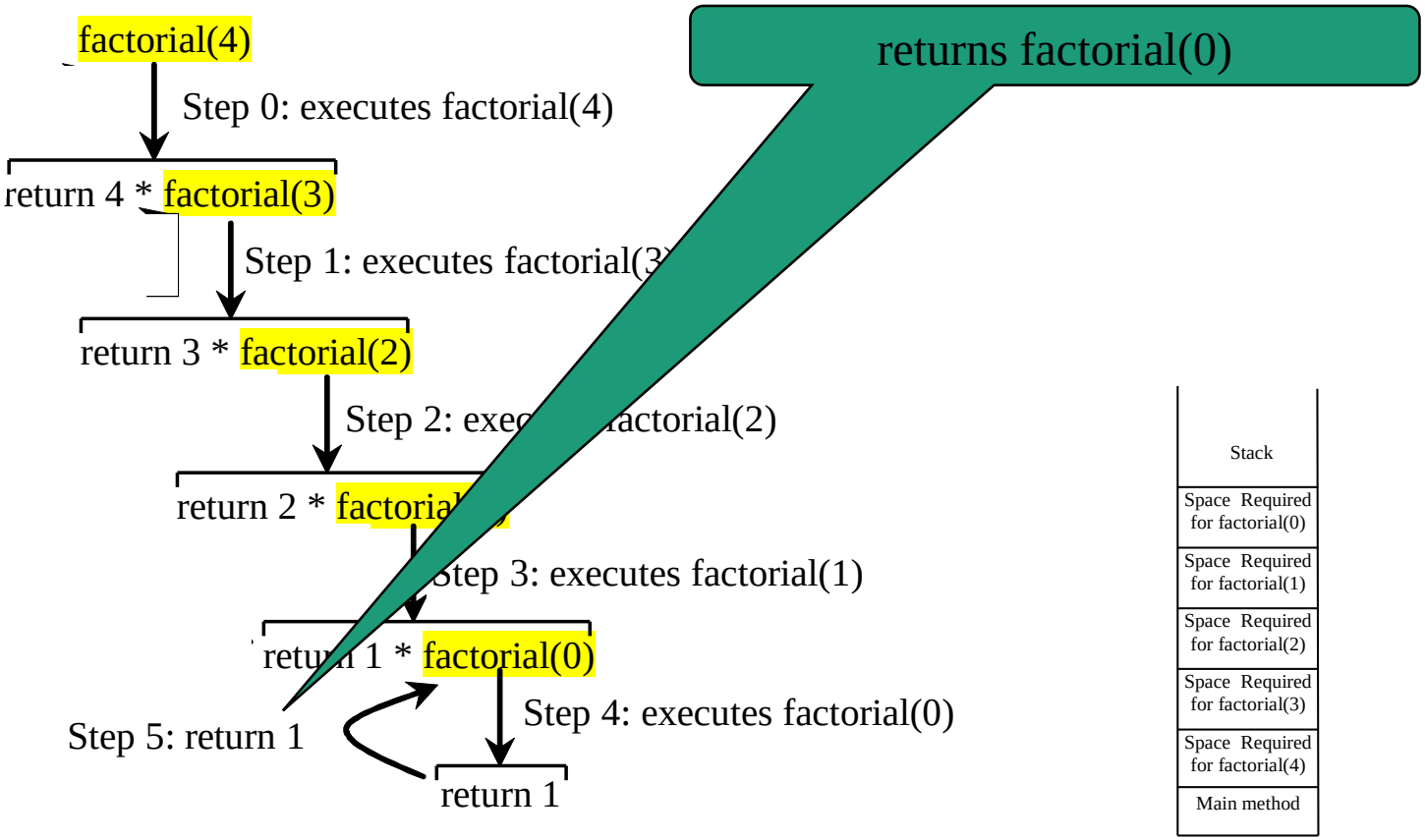
animation

Trace Recursive factorial



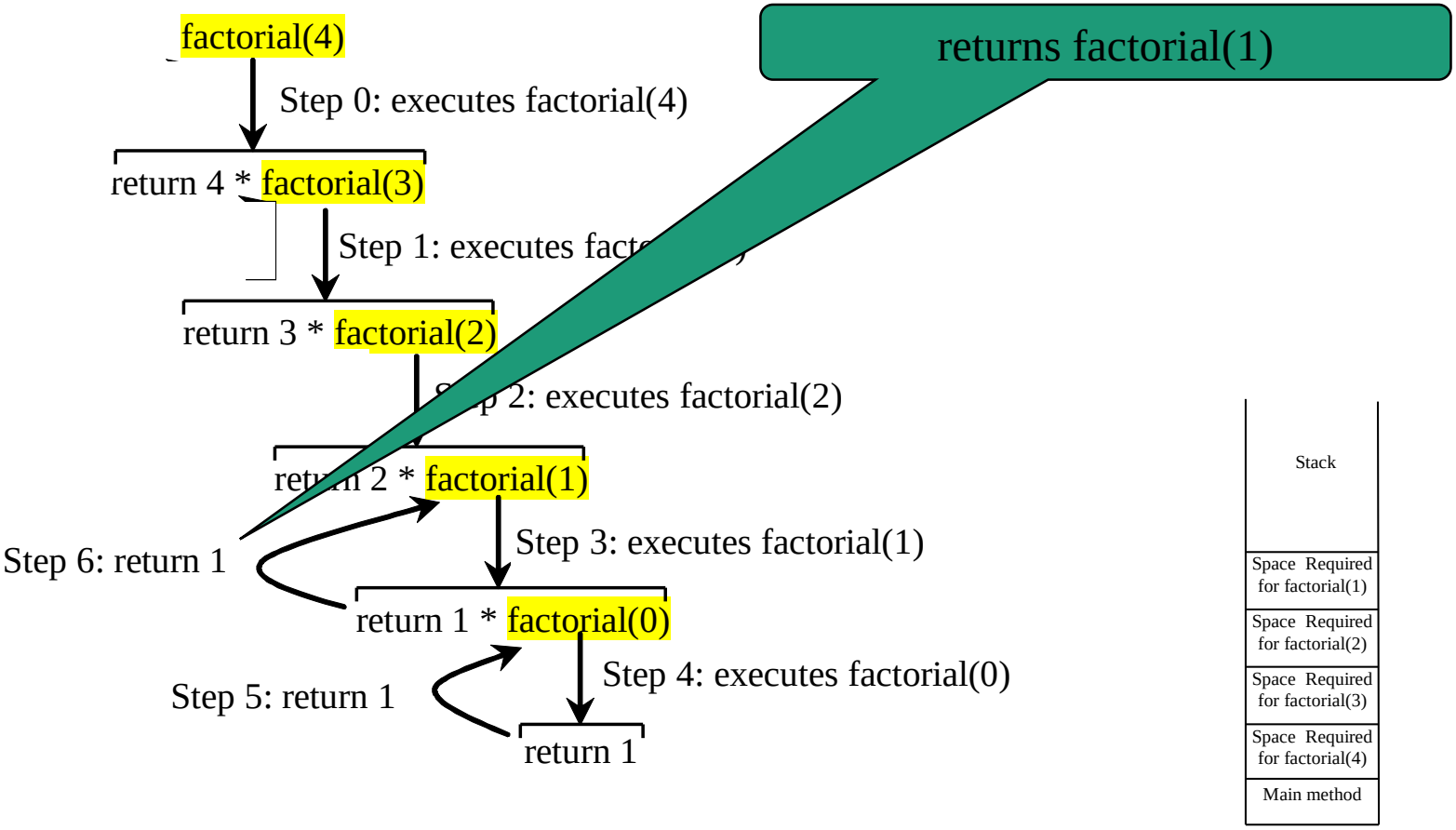
animation

Trace Recursive factorial



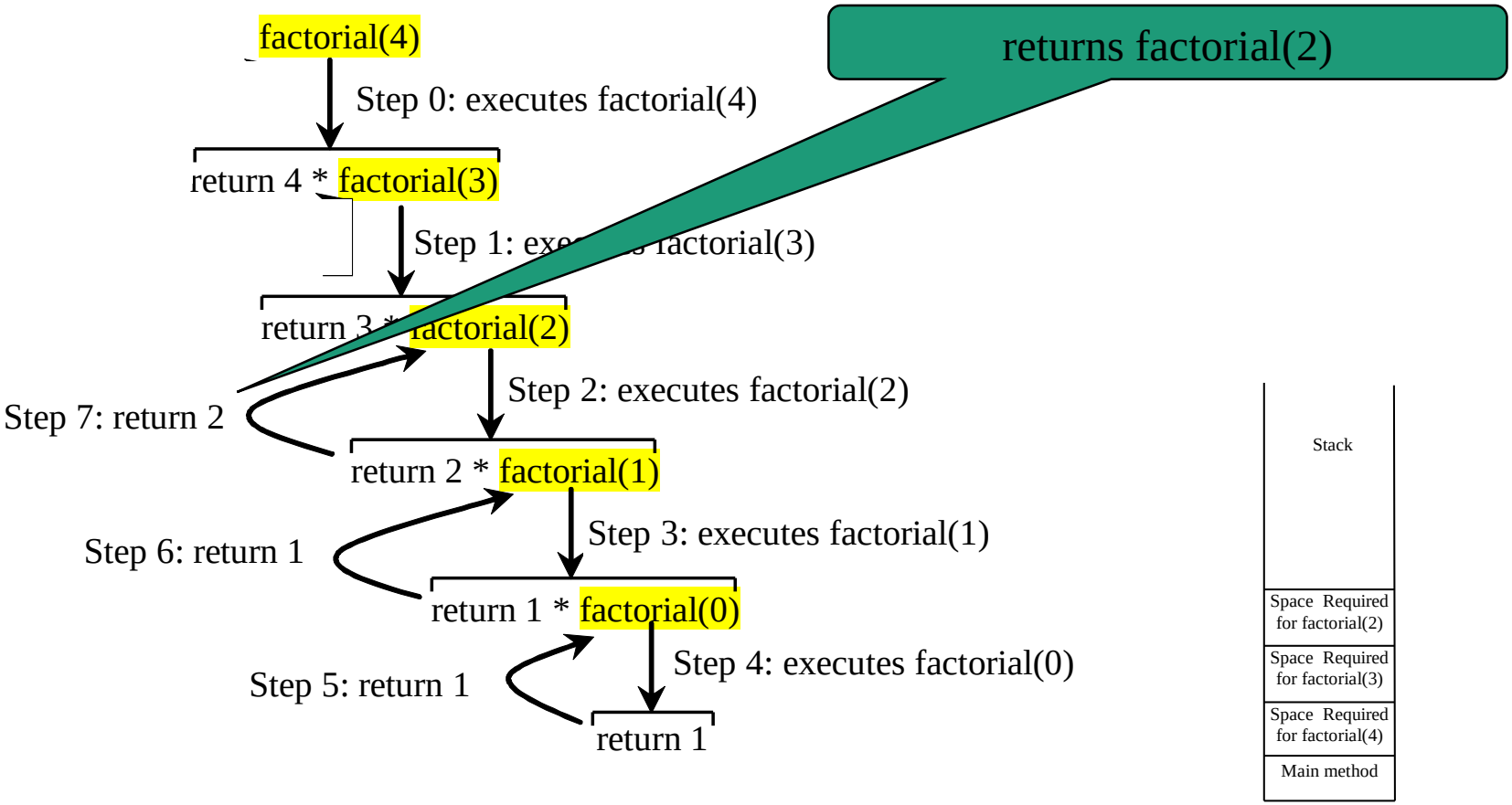
animation

Trace Recursive factorial



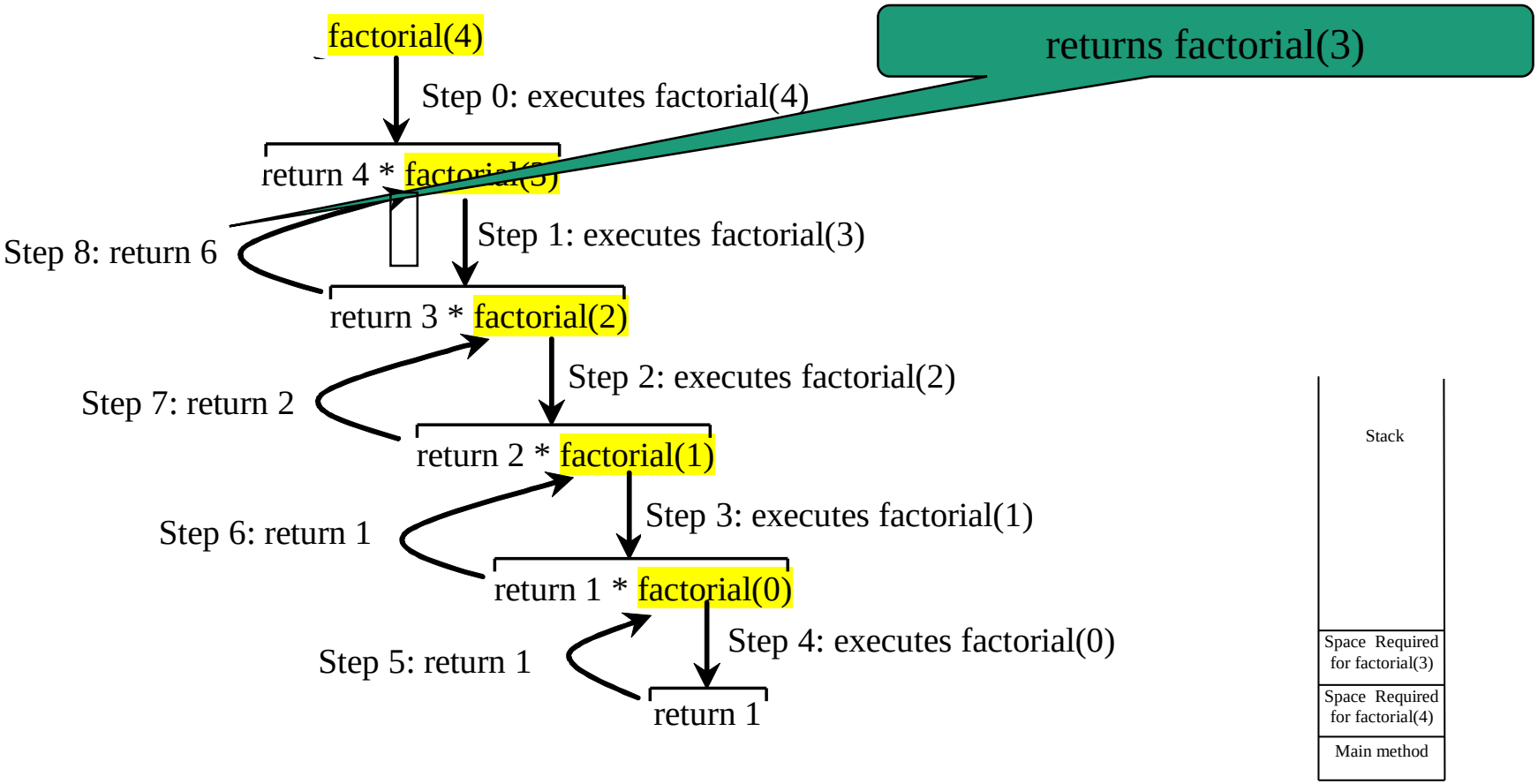
animation

Trace Recursive factorial



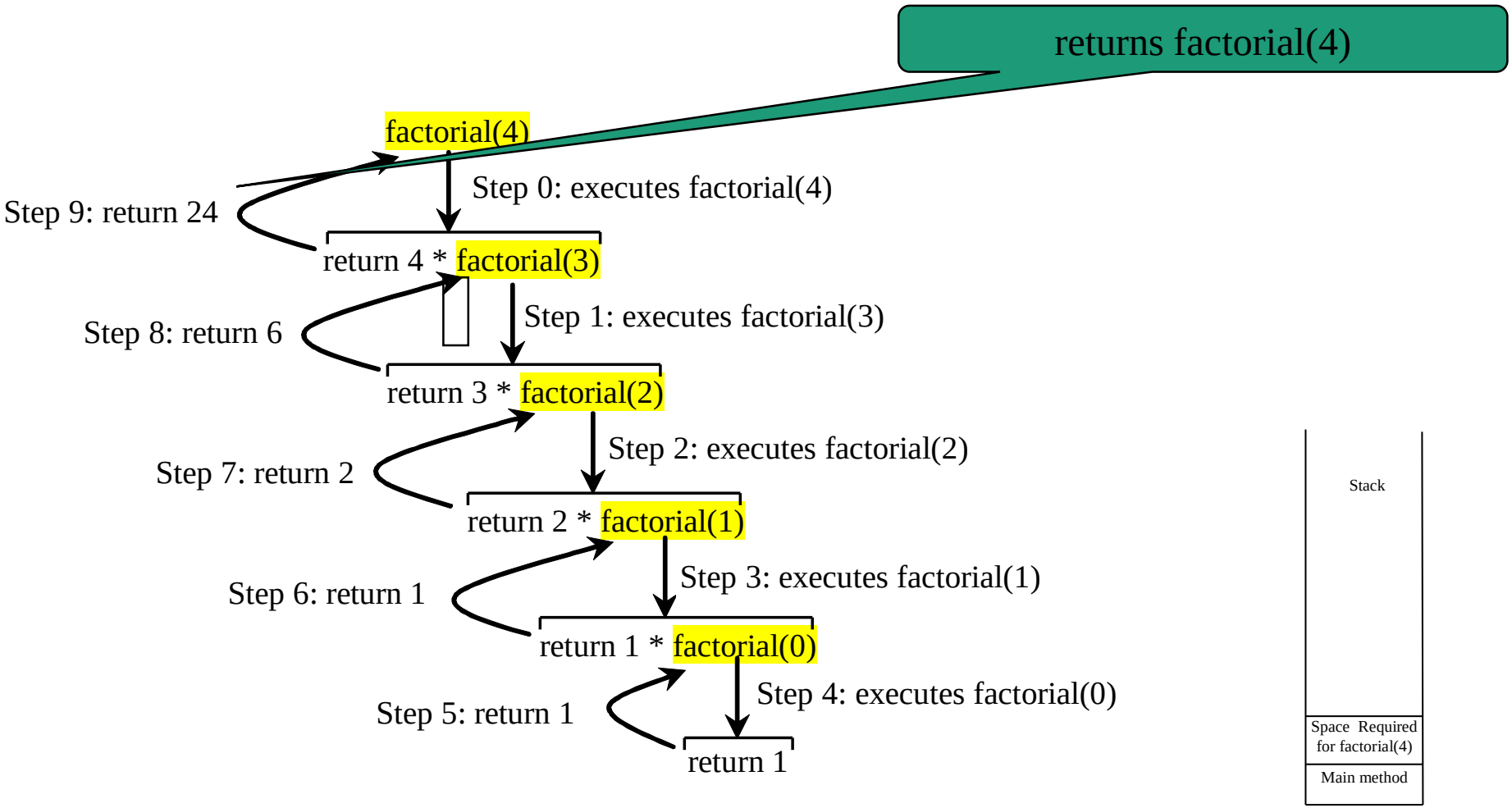
animation

Trace Recursive factorial

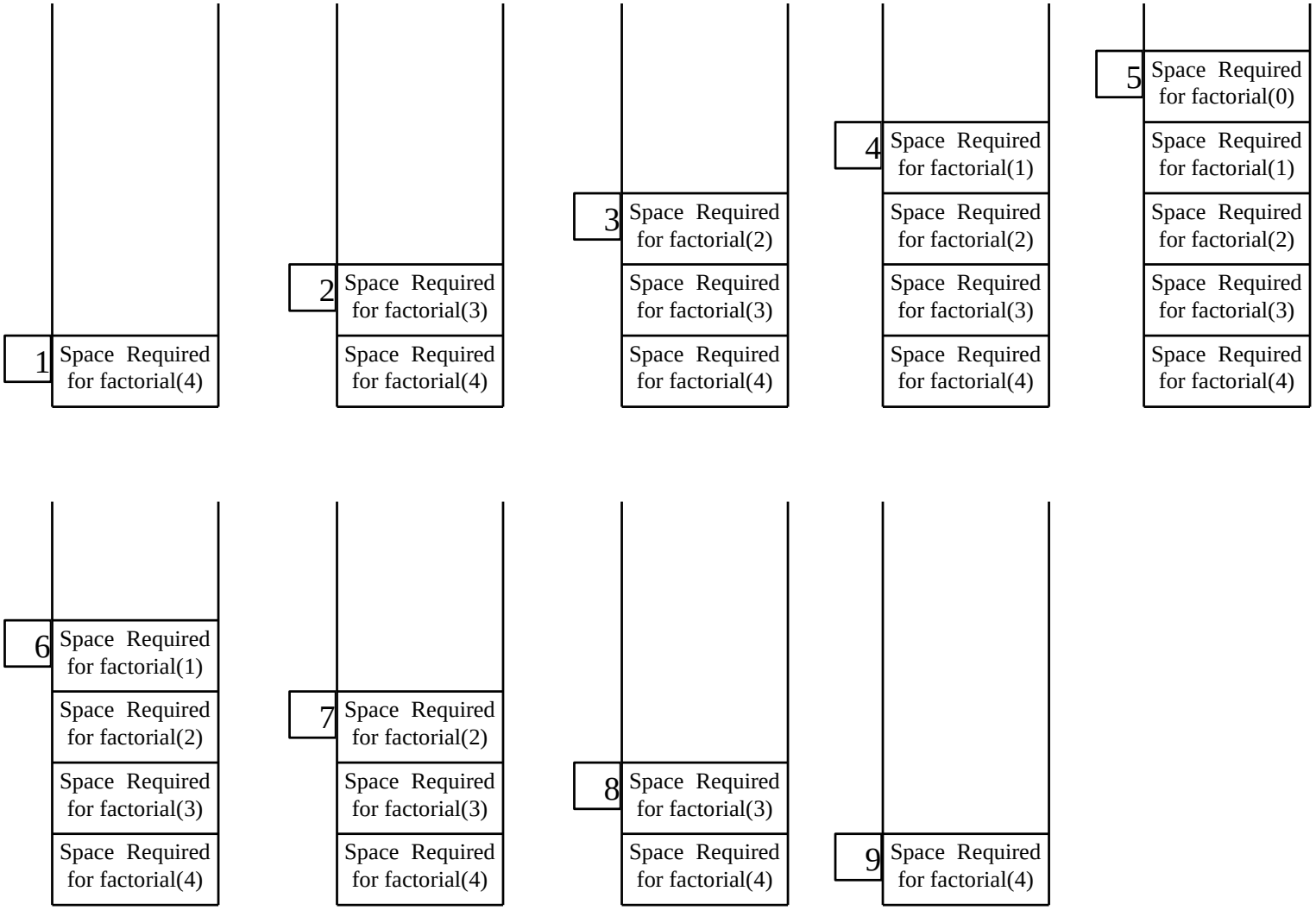


animation

Trace Recursive factorial



factorial(4) Stack Trace



Questions?

- Compute recursive mathematical functions using recursive methods
- Call stacks of recursive method invocation