Kaitlin Hoffmann

Office Hours:

SH 243 MR 11:00 - 12:30 PM via appointment https://calendly.com/hoffmank4/15min

Email: hoffmank4@newpaltz.edu

For TA Office Hours and Email – Please see syllabus

NEW! Supplemental Instruction: Sign up at my.newpaltz.edu

RECURSION

COMPUTER SCIENCE I

OBJECTIVES

- Recursive Methods

WHAT IS RECURSION?

- A recursive method is a method that calls **itself**.
- It is an alternative to using loops (for, while, etc), however, it's not always the best way to solve problems.
- Most of the time, loops will do the job just fine, however, some data structures, such as binary trees (you'll learn this in CS2), are better suited for recursive methods.
- Let's see an example...

LOOP VS RECURSION

- Say we have a problem that asks us to **print out the numbers from 1 to n**. We can do this using a loop (iterative process) or using recursion:

Iterative (loop)

```java
public static void printLoop(int n) {
    for(int i = 1; i<=n; i++) {
        System.out.print(i + " ");
    }
}
```

Recursive

```java
public static void printRecursively(int n) {
    if(n == 0) {
        return;
    }
    printRecursively( n: n-1);
    System.out.print(n + " ");
}
```

RECURSIVE METHODS

- The **base case** is going to tell us when to stop making recursive calls.
- The **general case** will include a recursive call and possibly some other computations.
- Usually in recursion we need to find a **pattern** to develop our method. The pattern is going to involve some information about the previous step or computation.

```
public static void printRecursively(int n) {
    if(n == 0) {
        return;
    }
    printRecursively( n: n-1);
    System.out.print(n + " ");
}
```

Base Case

General Case

Output:

```
1 2 3 4 5 6 7 8
```

BASE CASE

- The **base case** is going to tell us when to ***stop*** making recursive calls.
- If we want to print 1 through n, we want to **stop** making recursive methods when n reaches 0. This is our base case:

if(n == 0) { …

- With void types, we can just use the keyword ***return;*** by itself to stop the calls.

```
public static void printRecursively(int n) {
    if(n == 0) {
        return;
    }
    printRecursively( n: n-1);
    System.out.print(n + " ");
}
```

Base Case

General Case

Output:

1 2 3 4 5 6 7 8

BASE CASE

- The **general case** will include a ***recursive*** call and possibly some other computations.
- If we want to make n decrease by 1 every time and print n:

printRecursively(n-1);

System.out.print(n + " ");

```
public static void printRecursively(int n) {
    if(n == 0) {
        return;
    }
    printRecursively( n: n-1);
    System.out.print(n + " ");
}
```

Base Case

General Case

Output:

1 2 3 4 5 6 7 8

HOW DOES RECURSION WORK?

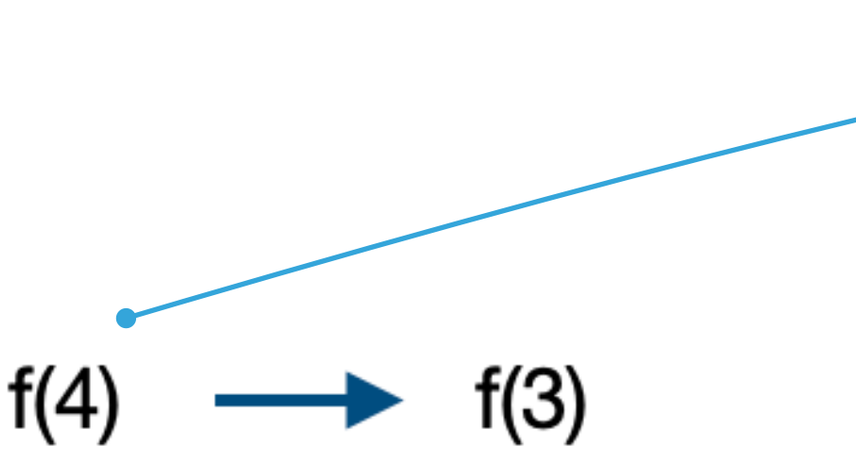
- When a method calls itself, it adds the code of the **general case** onto a data structure called a **stack**. It will keep "stacking" until the base case is hit. Once hit, it will remove each from the stack to perform the code.

- Let's see an example...

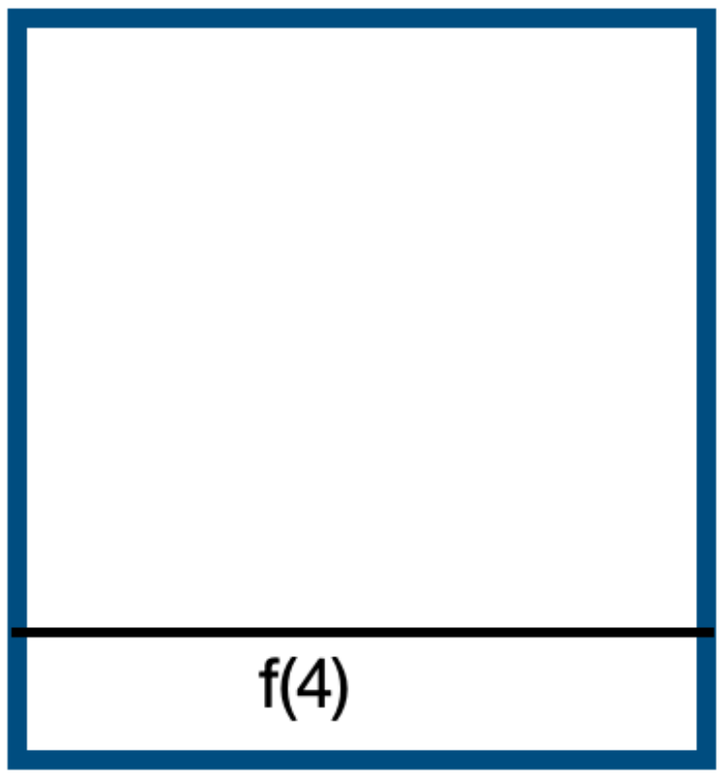
HOW DOES RECURSION WORK?


```
public static void f(int n) {
    if(n <= 0) {
        return;
    }
    f( n: n-1);
    System.out.print(n + " ");
}
```

Trace for **f(4)**



Didn’t reach base case, so 4 – 1…

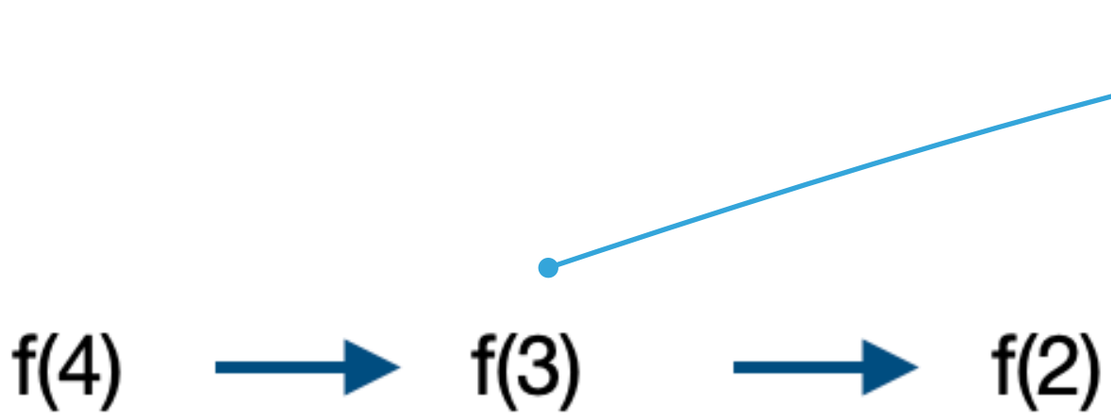


Stack

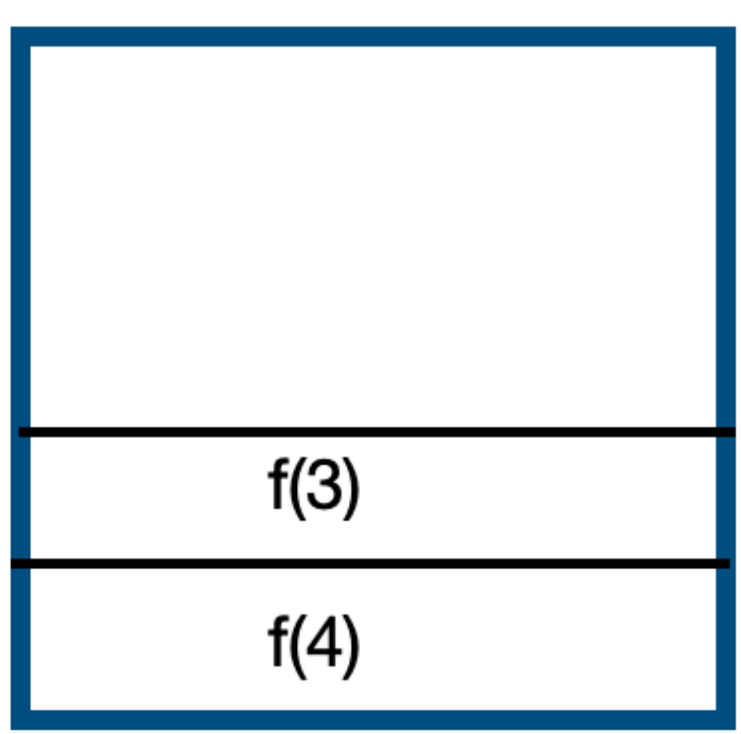
HOW DOES RECURSION WORK?


```
public static void f(int n) {
    if(n <= 0) {
        return;
    }
    f( n: n-1);
    System.out.print(n + " ");
}
```

Trace for **f(4)**



Didn’t reach base case, so 3 – 1…

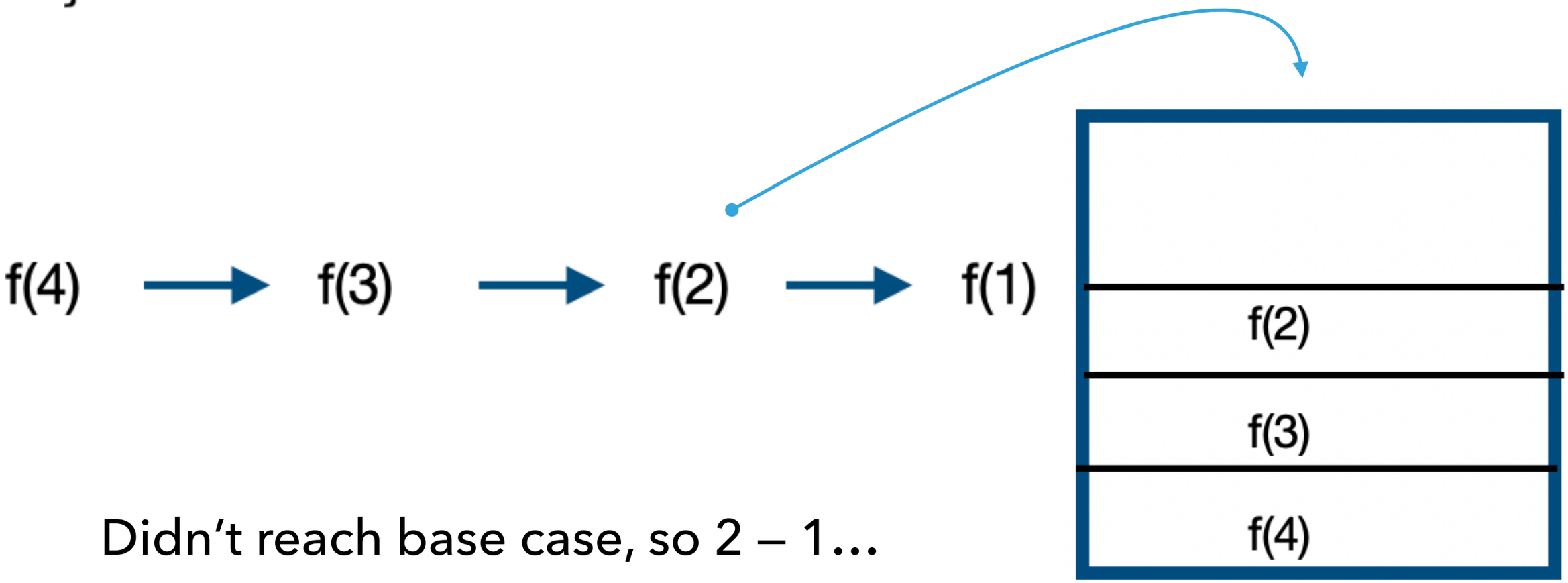


Stack

HOW DOES RECURSION WORK?


```
public static void f(int n) {
    if(n <= 0) {
        return;
    }
    f( n: n-1);
    System.out.print(n + " ");
}
```

Trace for **f(4)**

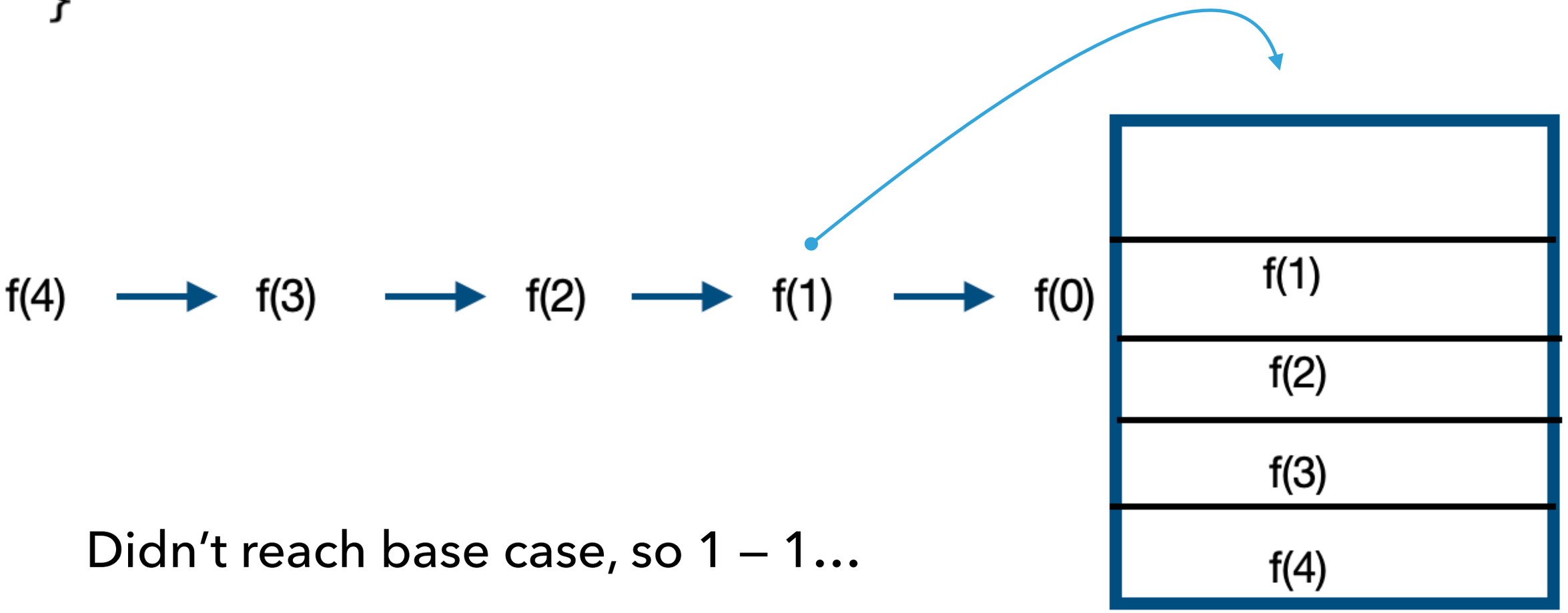


Didn’t reach base case, so 2 – 1…

HOW DOES RECURSION WORK?


```
public static void f(int n) {
    if(n <= 0) {
        return;
    }
    f( n: n-1);
    System.out.print(n + " ");
}
```

Trace for **f(4)**

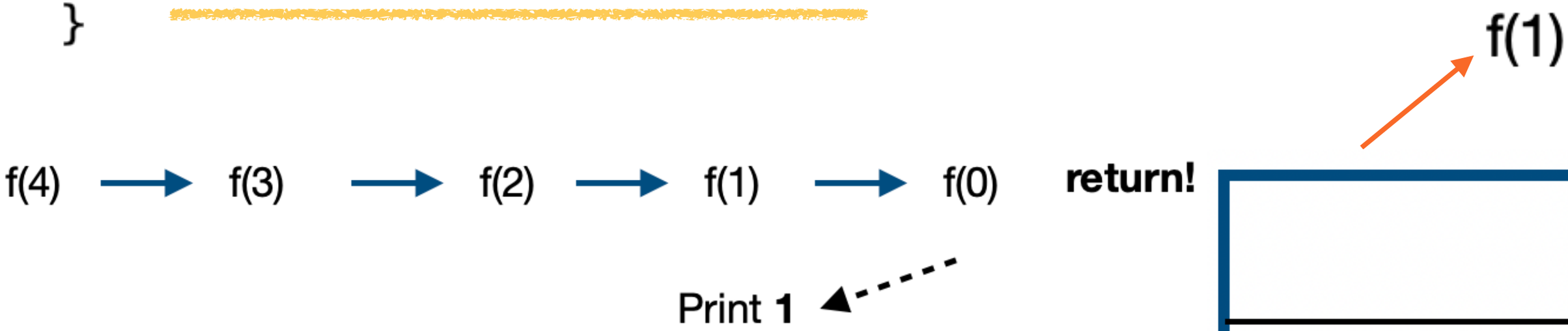


Didn’t reach base case, so 1 – 1…

HOW DOES RECURSION WORK?


```
public static void f(int n) {
    if(n <= 0) {
        return;
    }
    f( n: n-1);
    System.out.print(n + " ");
}
```

Trace for **f(4)**



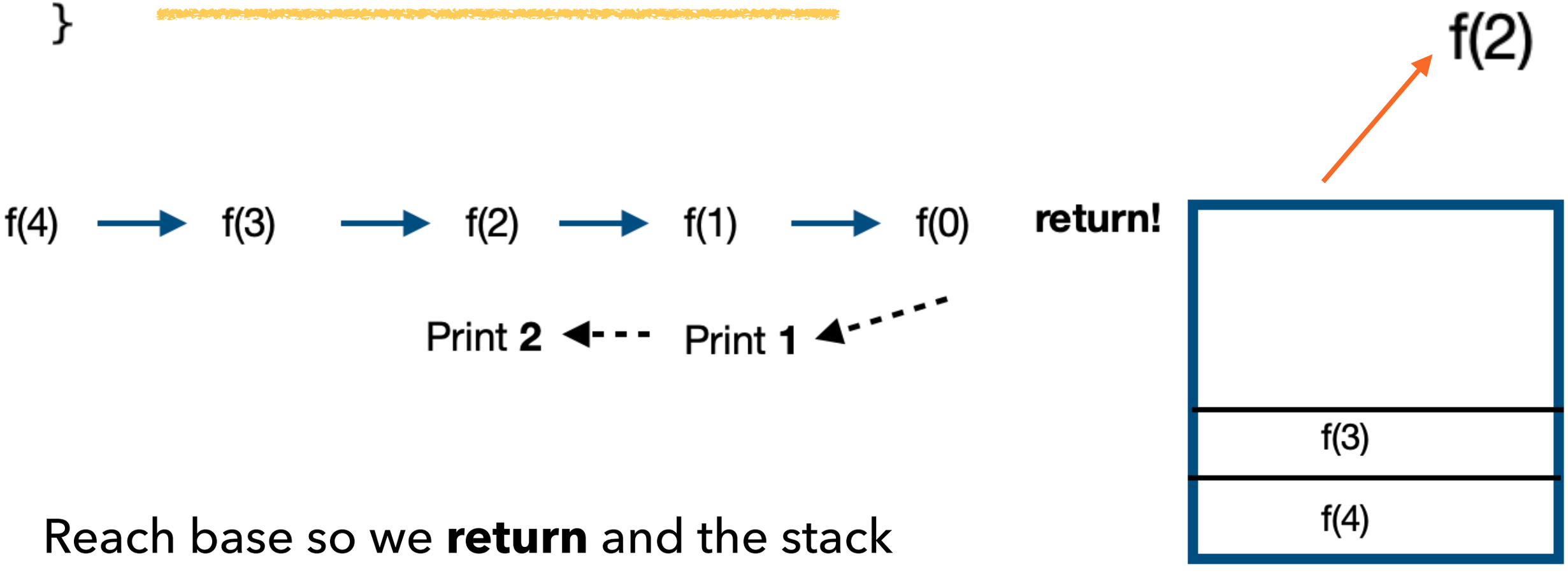
Reach base so we **return** and the stack empties one by one

Stack

HOW DOES RECURSION WORK?


```
public static void f(int n) {
    if(n <= 0) {
        return;
    }
    f( n: n-1);
    System.out.print(n + " ");
}
```

Trace for **f(4)**



Reach base so we **return** and the stack empties one by one

HOW DOES RECURSION WORK?

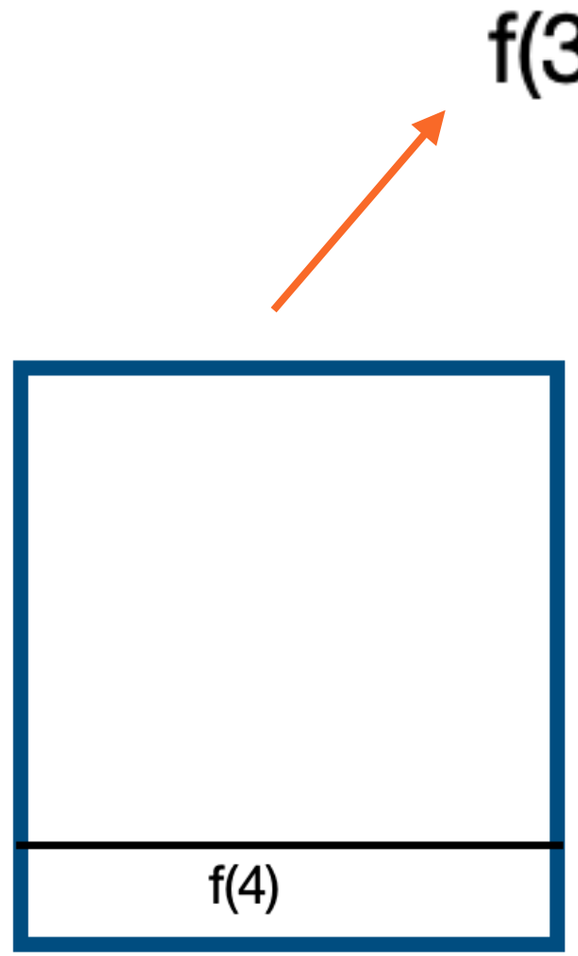

```
public static void f(int n) {
    if(n <= 0) {
        return;
    }
    f( n: n-1);
    System.out.print(n + " ");
}
```

Trace for **f(4)**

f(4) → f(3) → f(2) → f(1) → f(0) **return!**

Print **3** ◄- - - Print **2** ◄- - - Print **1** ◄- - - -

Reach base so we **return** and the stack empties one by one



HOW DOES RECURSION WORK?

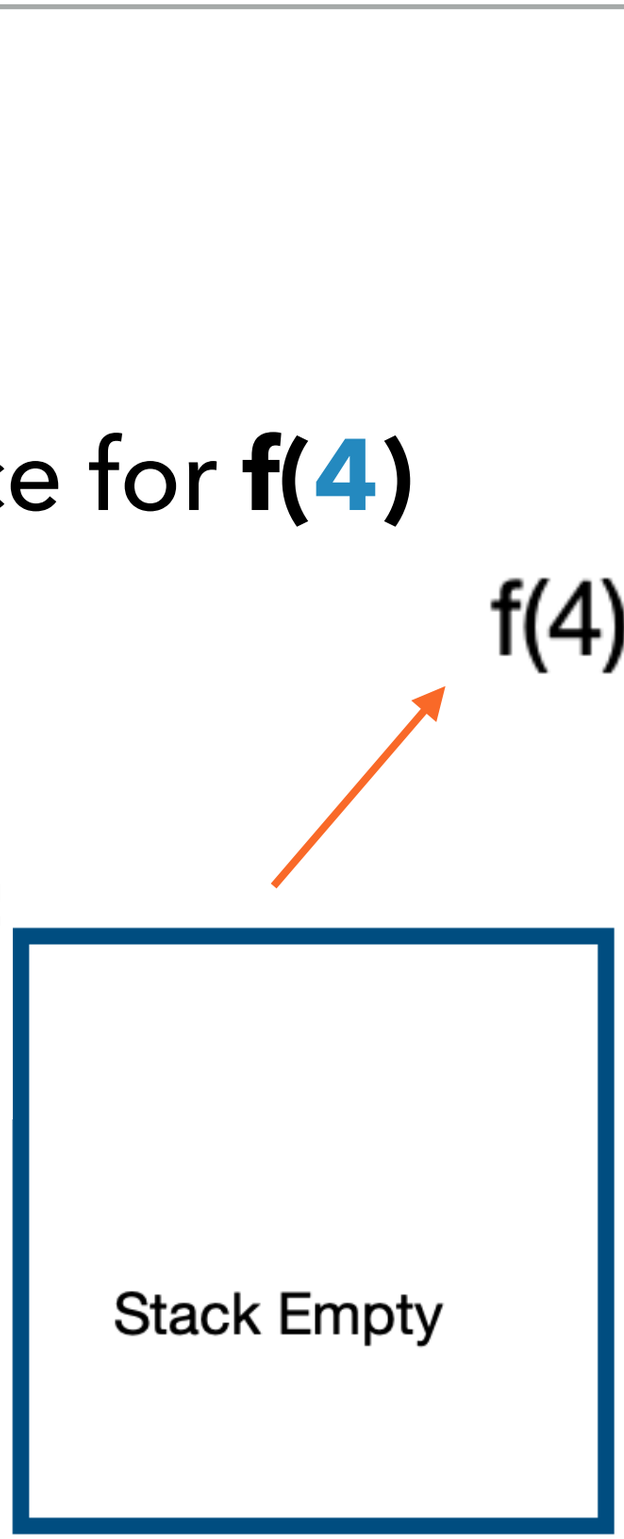

```
public static void f(int n) {
    if(n <= 0) {
        return;
    }
    f( n: n-1);
    System.out.print(n + " ");
}
```

Trace for **f(4)**

f(4) → f(3) → f(2) → f(1) → f(0) **return!**

Print **4** ◄- - - Print **3** ◄- - - Print **2** ◄- - - Print **1**

Stack is empty, so output is: **1 2 3 4**



Stack

EXAMPLE 1

- Printing 1 to n recursively. Let’s put this together now…

EXAMPLE 2

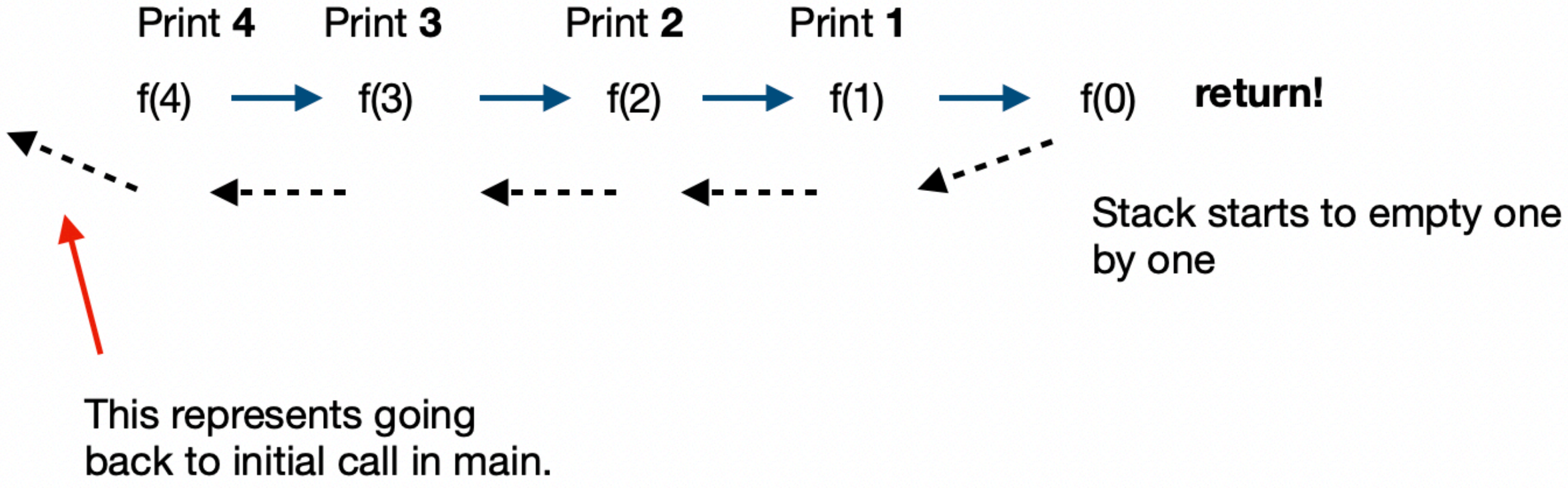
- Printing ***n to 1*** recursively.
- What is our **base** case (when to stop)? **n <= 0**

HOW DOES RECURSION WORK?


```
public static void f(int n) {
    if(n <= 0) {
        return;
    }
    System.out.print(n + " ");
    f( n: n-1);
}
```

Let's print **n to 1** (backwards from last exercise). Notice we just **switch** the print and method call.

Trace for **f(4)**



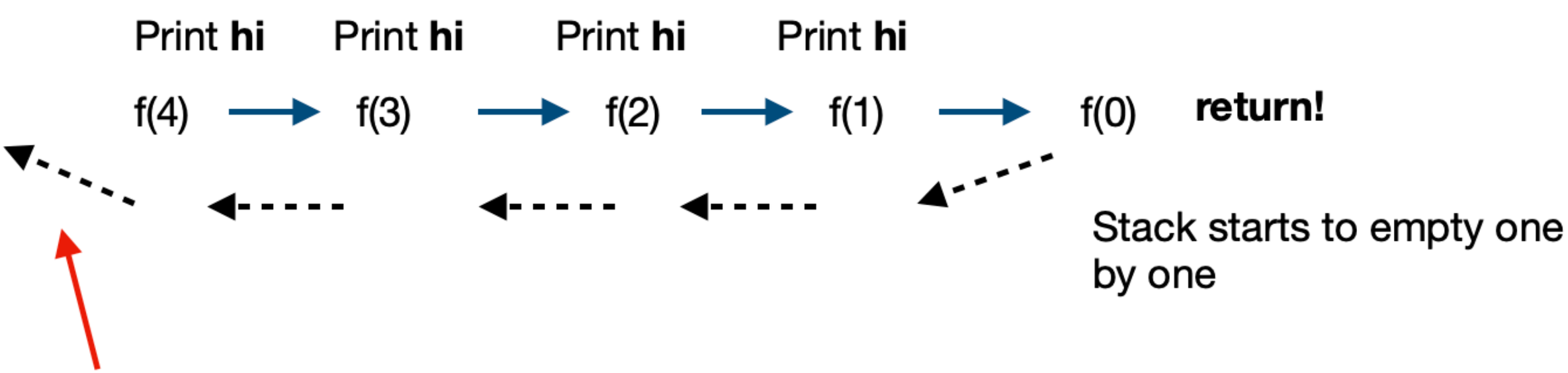
EXERCISE 1

- Create a recursive method that prints a String **n** amount of times. The String and n should be parameters.
- Make sure you determine the **base case** first (when to stop).
- Don’t overthink recursion! All you need to do is to subtract 1 from n every call.

EXERCISE 1 — SOLUTION


```
public static void printString(String s, int n) {
    if(n <= 0) {
        return;
    }
    System.out.println(s);
    printString(s, n: n-1);
}
```

Trace for **printString(“hi”, 4)**



EXAMPLE 3

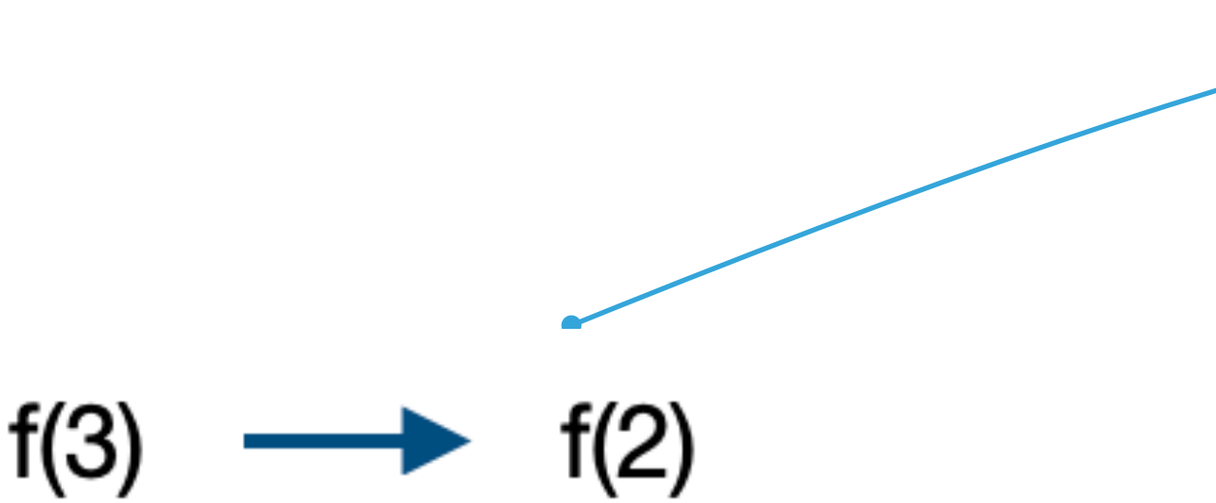
- Let's find the sum of **1 to n** recursively.
- We can't use a variable to hold the sum since it will re-initialize it to 0 every time the method is called.
- That's okay! We can continuously **add to the stack**, allowing the sum to be stored.
- First, what should be our **base case** and what should we return in the base case?

n <= 0 ***return*** 0;

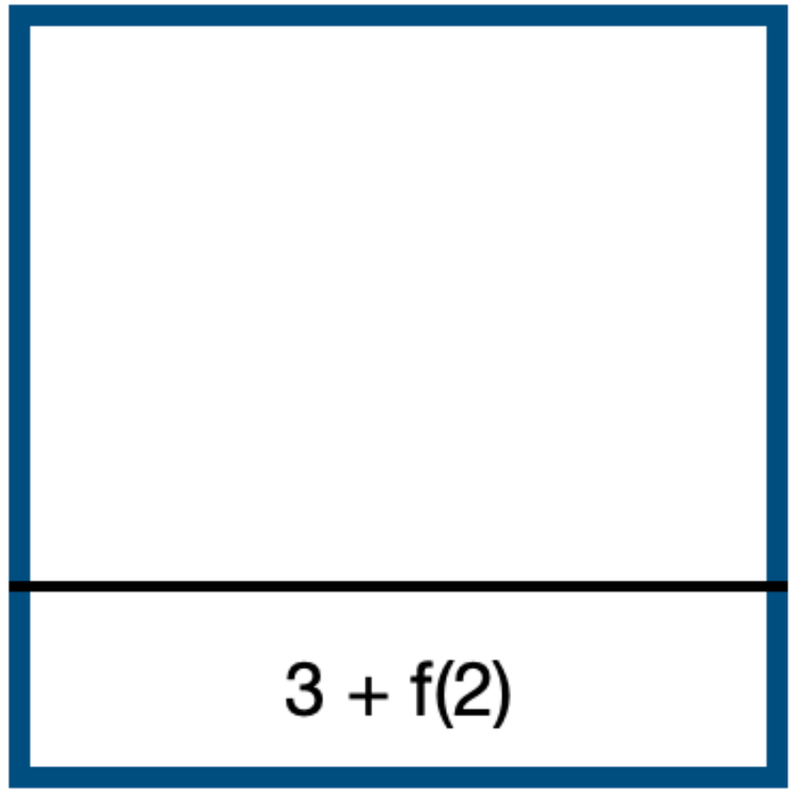
HOW DOES RECURSION WORK?


```
public static int findSum(int n) {
    if(n <= 0) {
        return 0;
    }
    return n + findSum( n: n - 1);
}
```

Trace for **f(3)**



Didn’t reach base case, so 3 – 1…



Stack

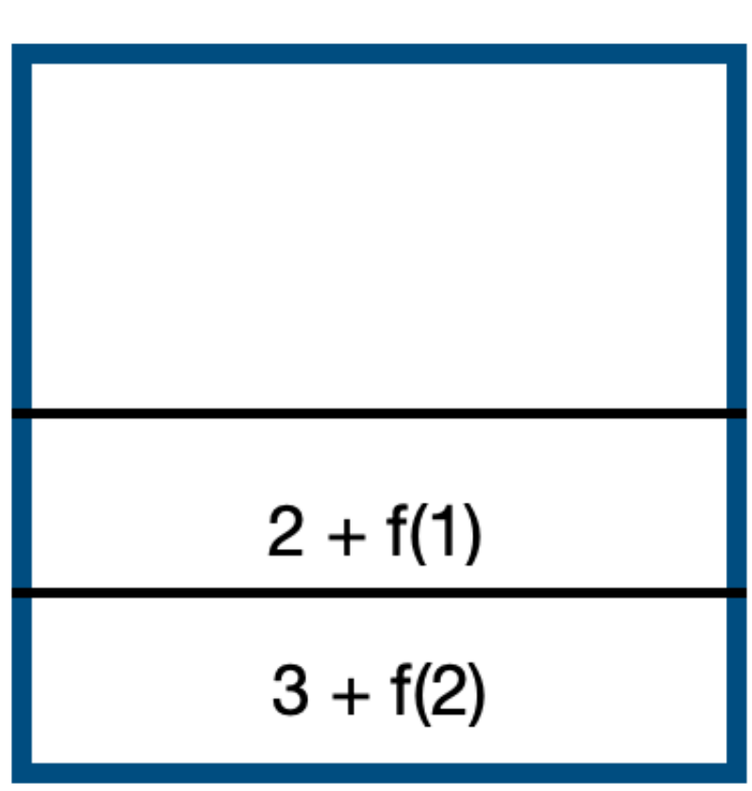
HOW DOES RECURSION WORK?


```
public static int findSum(int n) {
    if(n <= 0) {
        return 0;
    }
    return n + findSum( n: n - 1);
}
```

Trace for **f(3)**

f(3) → f(2) → f(1)

Didn’t reach base case, so 2 – 1…



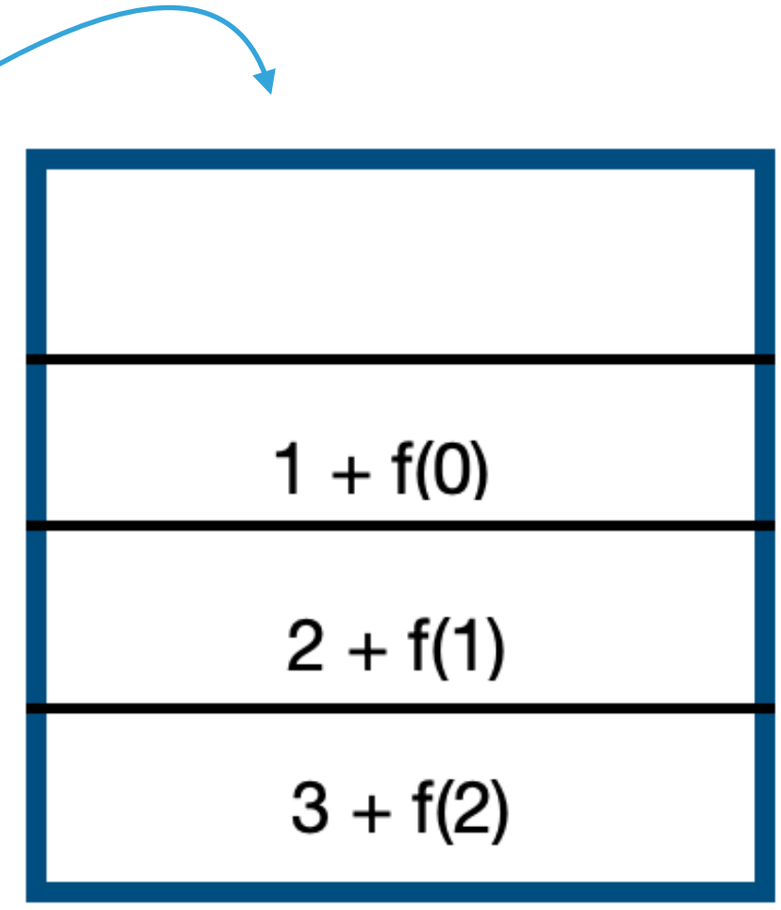
Stack

HOW DOES RECURSION WORK?


```
public static int findSum(int n) {
    if(n <= 0) {
        return 0;
    }
    return n + findSum( n: n - 1);
}
```

Trace for **f(3)**

f(3) → f(2) → f(1) → f(0) **return 0!**



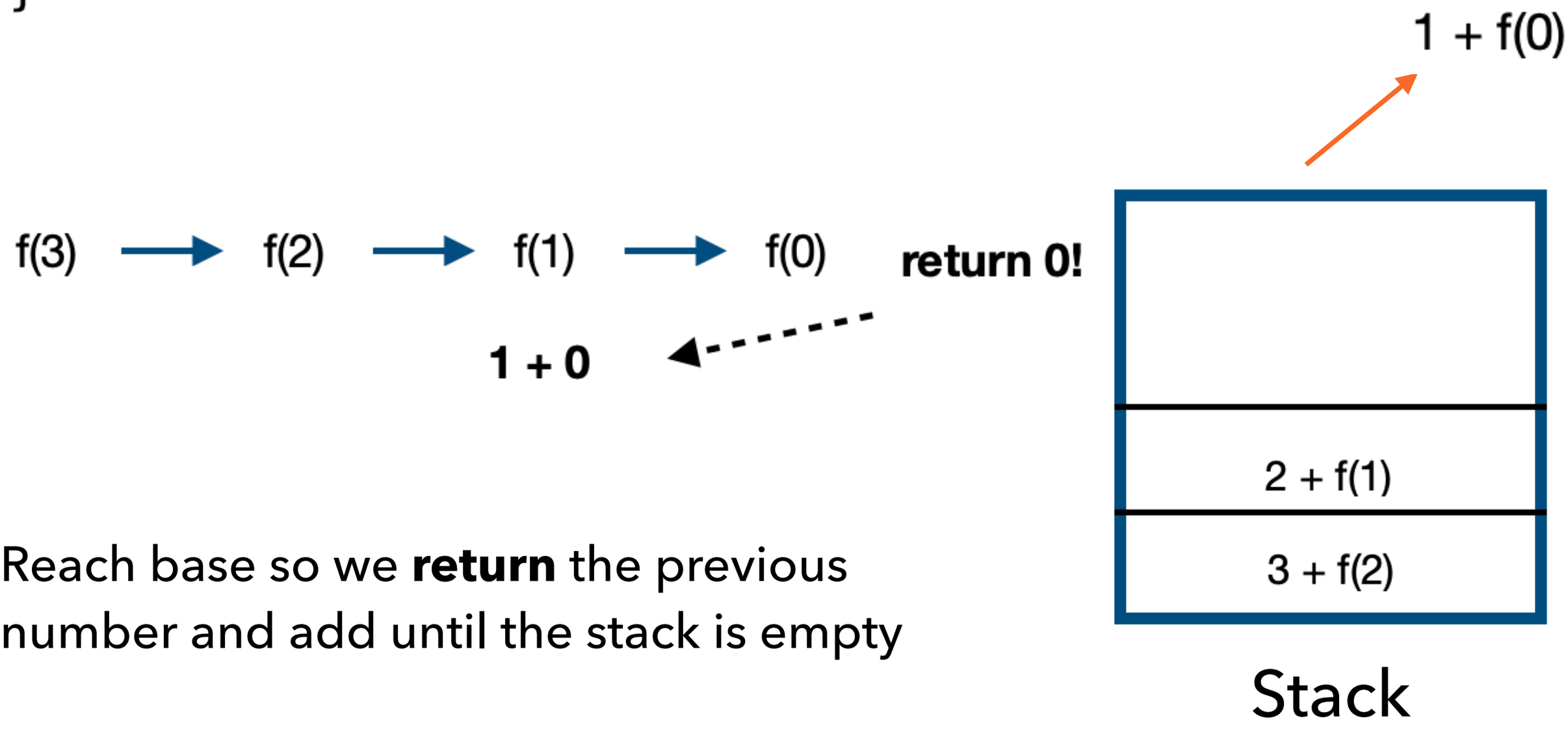
Stack

Didn’t reach base case, so 1 – 1…

HOW DOES RECURSION WORK?


```
public static int findSum(int n) {
    if(n <= 0) {
        return 0;
    }
    return n + findSum( n: n - 1);
}
```

Trace for **f(3)**

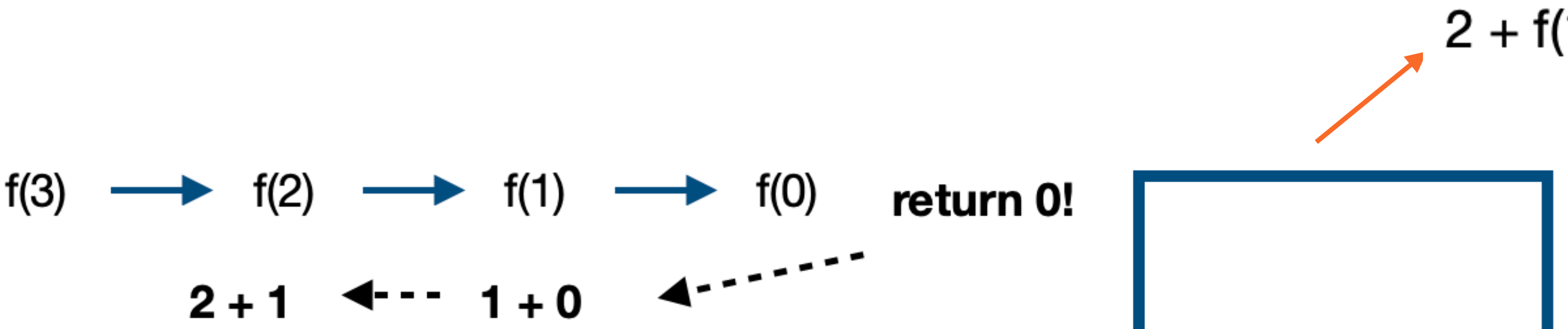


Reach base so we **return** the previous number and add until the stack is empty

HOW DOES RECURSION WORK?


```
public static int findSum(int n) {
    if(n <= 0) {
        return 0;
    }
    return n + findSum( n: n - 1);
}
```

Trace for **f(3)**

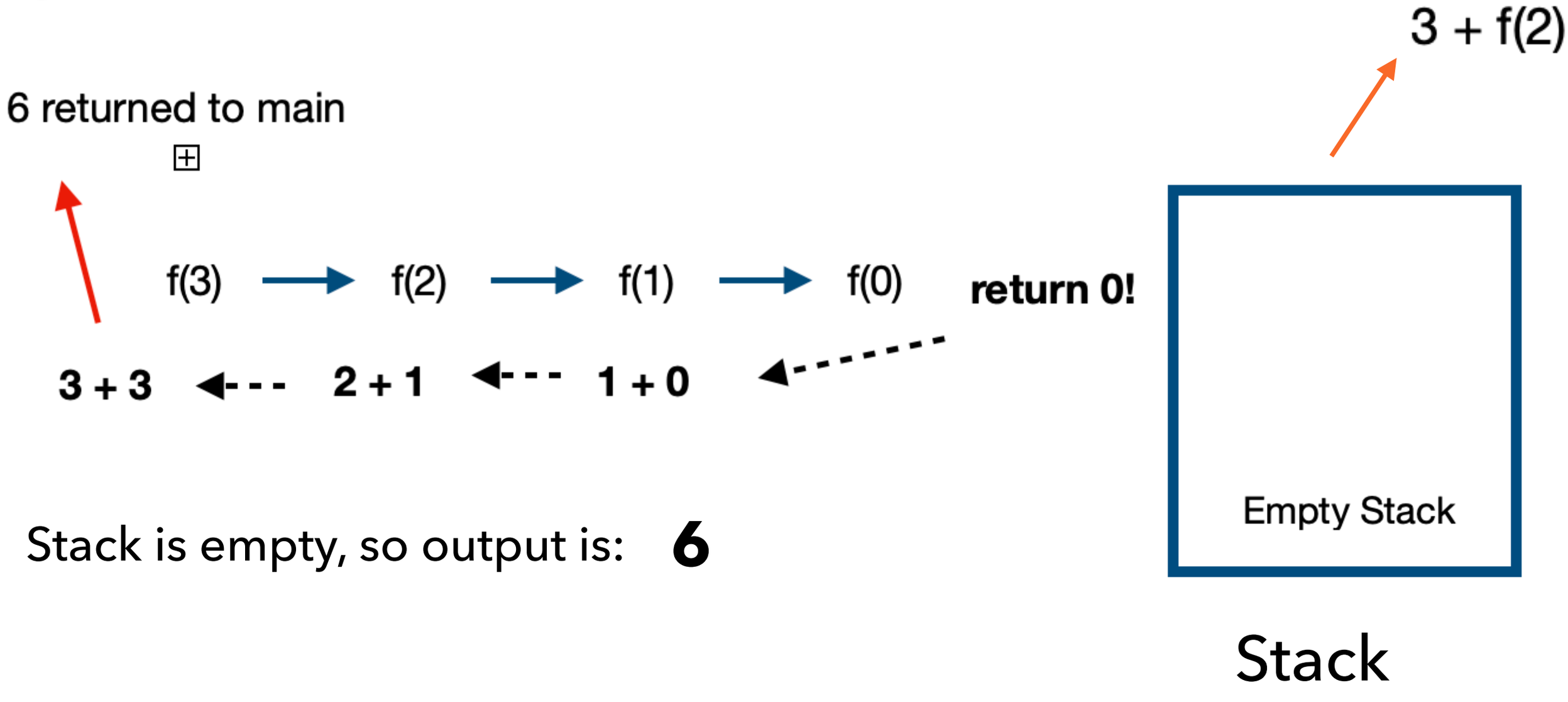


Reach base so we **return** the previous number and add until the stack is empty

HOW DOES RECURSION WORK?


```
public static int findSum(int n) {
    if(n <= 0) {
        return 0;
    }
    return n + findSum( n: n - 1);
}
```

Trace for **f(3)**



Stack is empty, so output is: **6**

EXAMPLE 3

- Let’s find the sum of **1 to n** recursively. Let’s do this now.

EXAMPLE 4

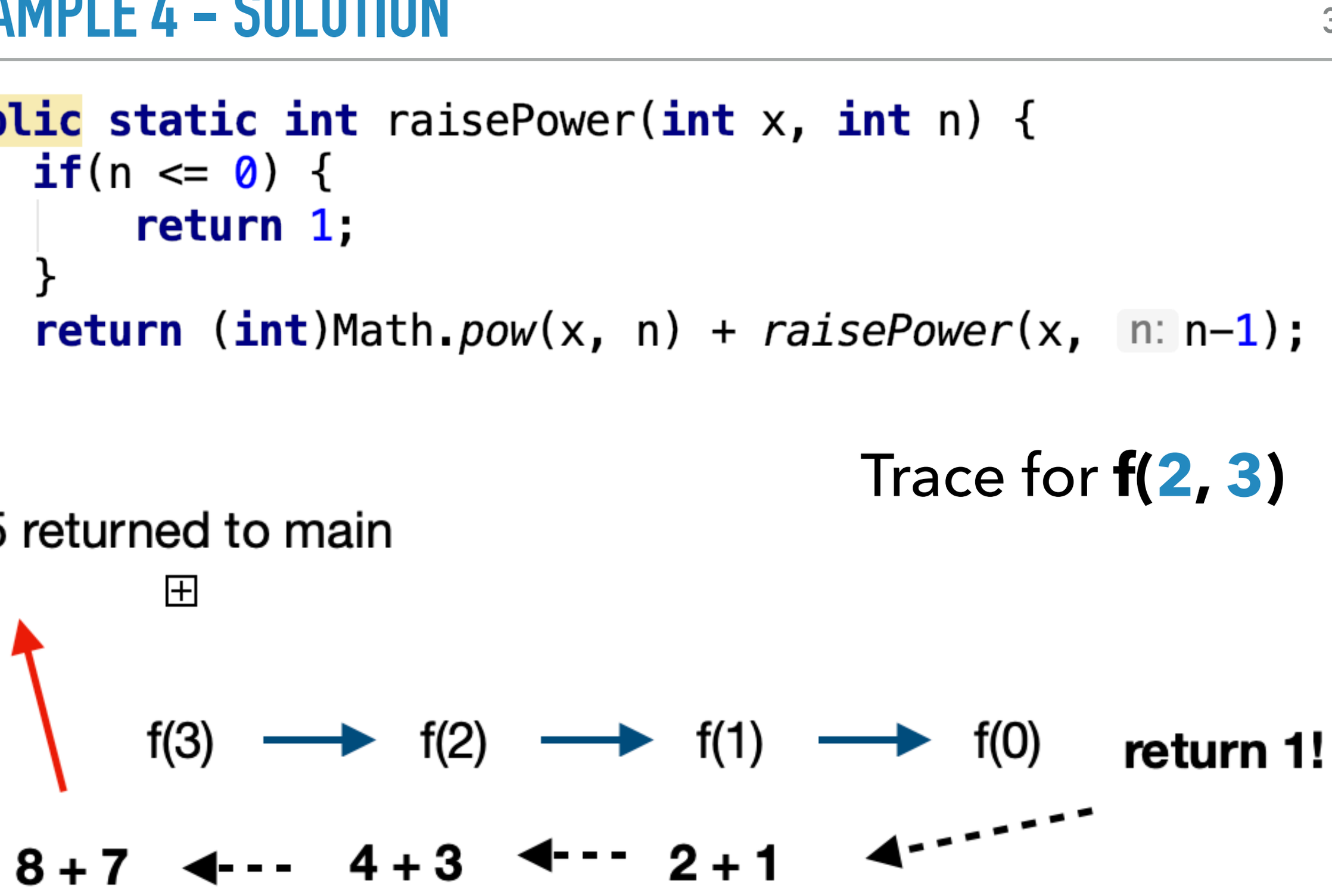
- Let's write a method to recursively sum ***x*** to the ***nth*** power ($\mathbf{x^n}$) starting with n = 0. Example: $\mathbf{x^0 + x^1 + x^2 + \ldots x^n}$
- What is our base case (when should we stop)? **n <= 0**

EXAMPLE 4 – SOLUTION


```
public static int raisePower(int x, int n) {
    if(n <= 0) {
        return 1;
    }
    return (int)Math.pow(x, n) + raisePower(x,  n: n-1);
}
```

Trace for **f(2, 3)**

15 returned to main

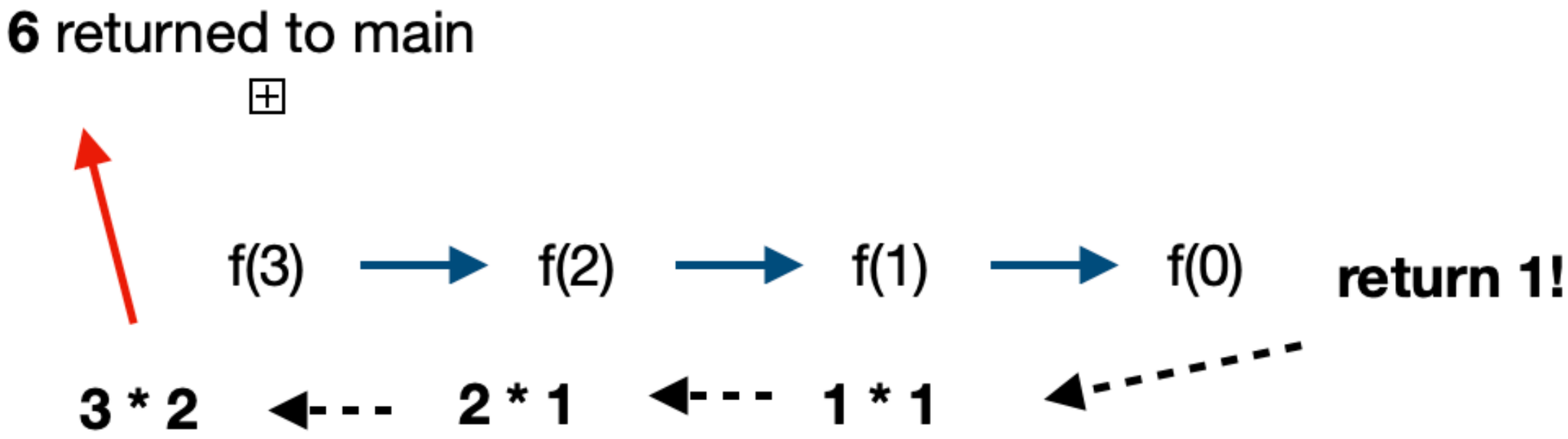


EXERCISE 2

- Create a recursive method that computes **factorial of n**.
  Example: **5!** = 1 * 2 * 3 * 4 * 5 = 120
- **NOTE: 0!** is **1** so what should be your base case?

EXERCISE 2


```java
public static int fact(int n) {
    if(n == 0) {
        return 1;
    }
    return n * fact( n: n-1);
}
```



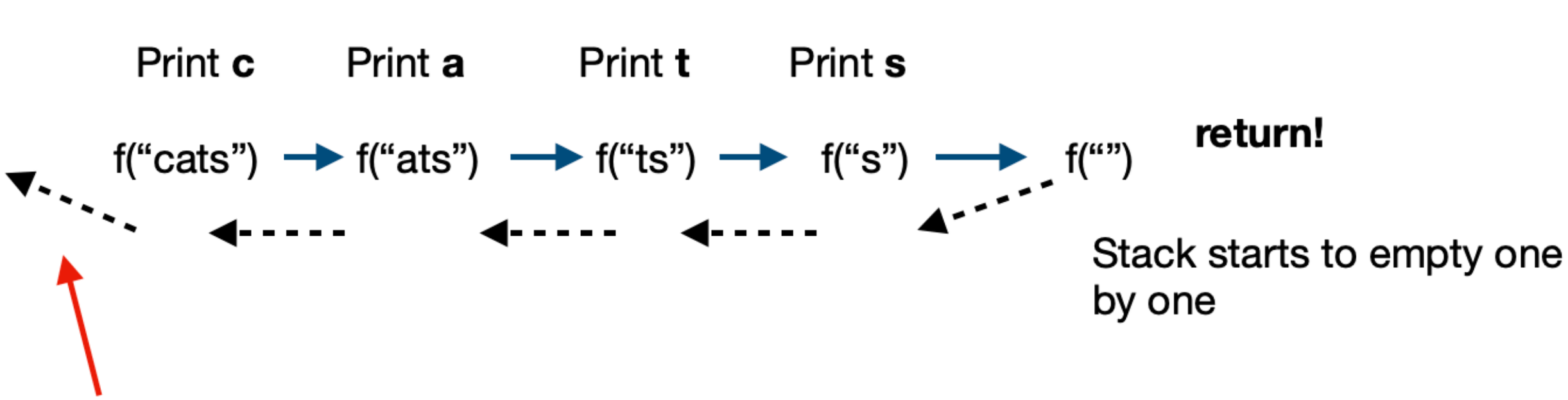
EXAMPLE 5

- Let's write a method to recursively display a string 1 character per line.
- We'll have to use the **substring** method to remove characters from our String every call. When our String has no more characters, we'll stop.
- What do you think our base case will be? **s.length() == 0**

EXAMPLE 5 — SOLUTION


```
public static void displayChars(String s) {
    if(s.length() == 0) {
        return;
    }
    System.out.println(s.charAt(0));
    displayChars(s.substring(1));
}
```

Trace for **f(“cats”)**



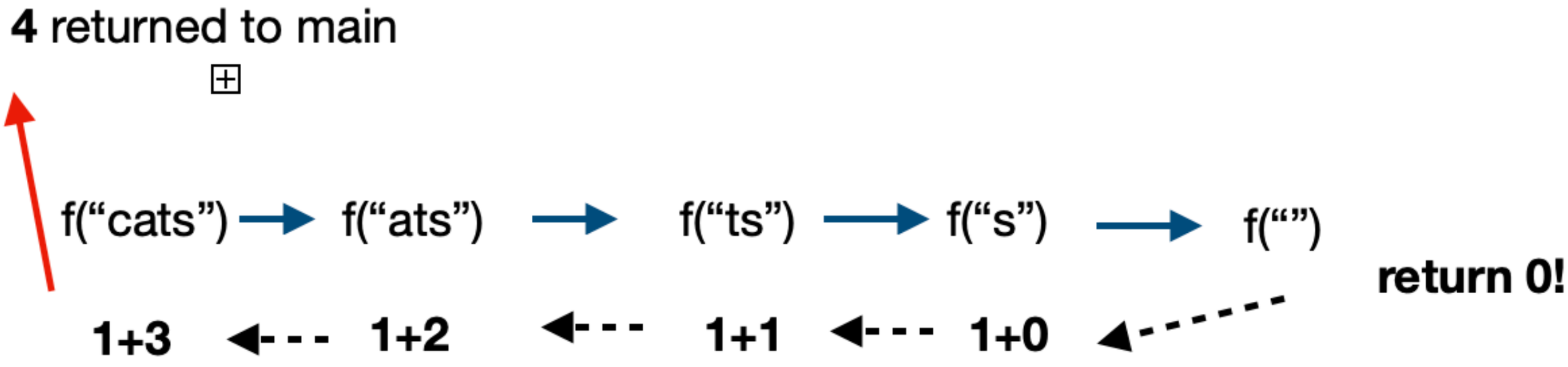
EXAMPLE 6

- Let's write a method to recursively compute the length of a string.
- What do you think our base case will be? **s.length() == 0**

EXAMPLE 6


```
public static int length(String s) {
    if(s.length() == 0) {
        return 0;
    }
    return 1 + length(s.substring(1));
}
```

Trace for **f(“cats”)**



EXERCISE 3

- Write a method to recursively reverse a string.

EXERCISE 3


```
public static void displayBackwards(String s) {
    if(s.length() == 0) {
        return;
    }
    System.out.println(s.charAt(s.length()-1));
    displayBackwards(s.substring(0, s.length()-1));
}
```

Trace for **f(“cats”)**