

Recursion

Tiziana Ligorio
tligorio@hunter.cuny.edu

Today's Plan

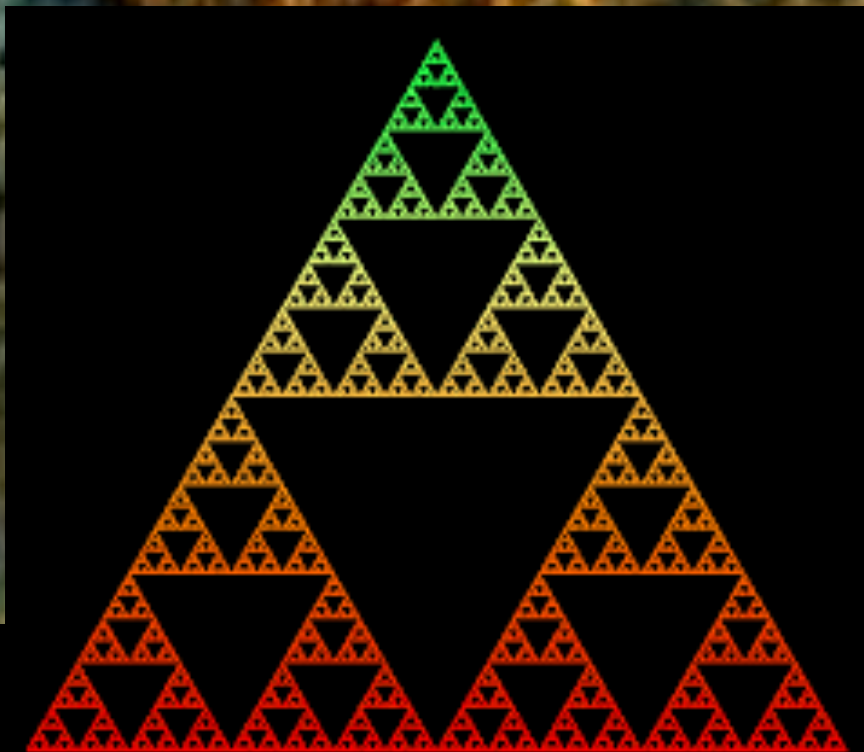


Announcements

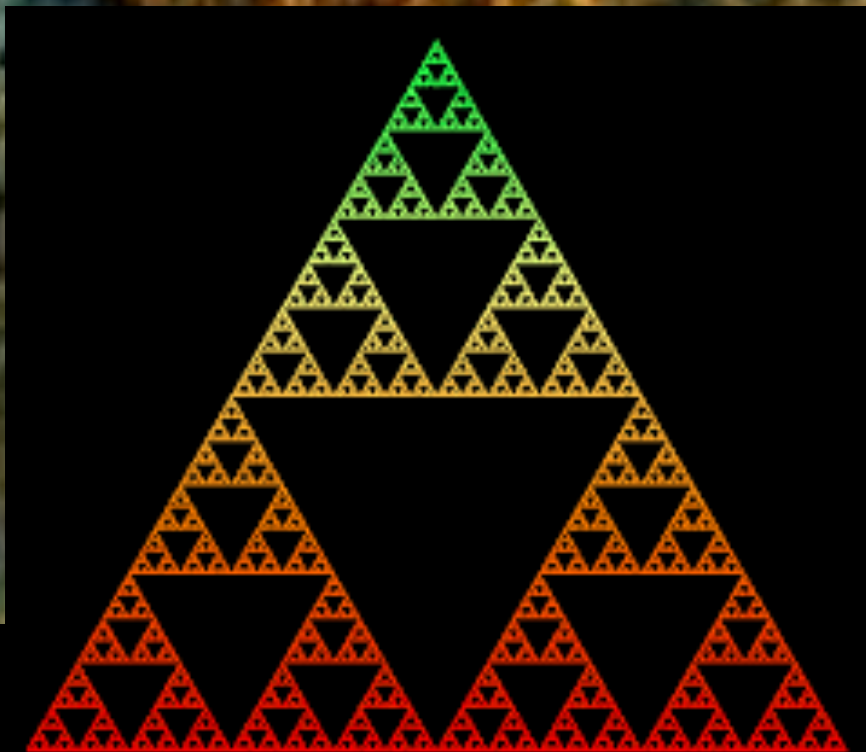
Recursion

Announcements

What do these images have in common



They contain a **SMALLER** copy of **THEMSELVES**



Print String Backwards

"Hello"

Print String Backwards

"Hello"

Procedure:

If there are characters to print

*Print the last character and **reverse the rest***



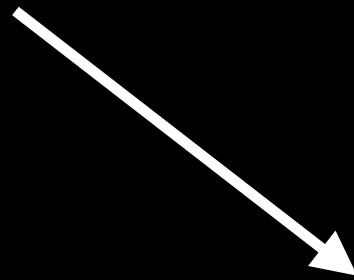
Recursive Call
Notice it's the last thing it does

Print String Backwards

Hello

o

Active functions



Program Stack

Print String Backwards

Hello
o
→ Hell

Active functions

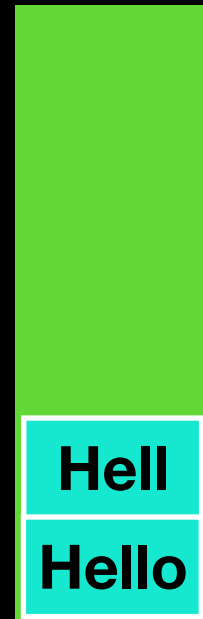


Program Stack

Print String Backwards

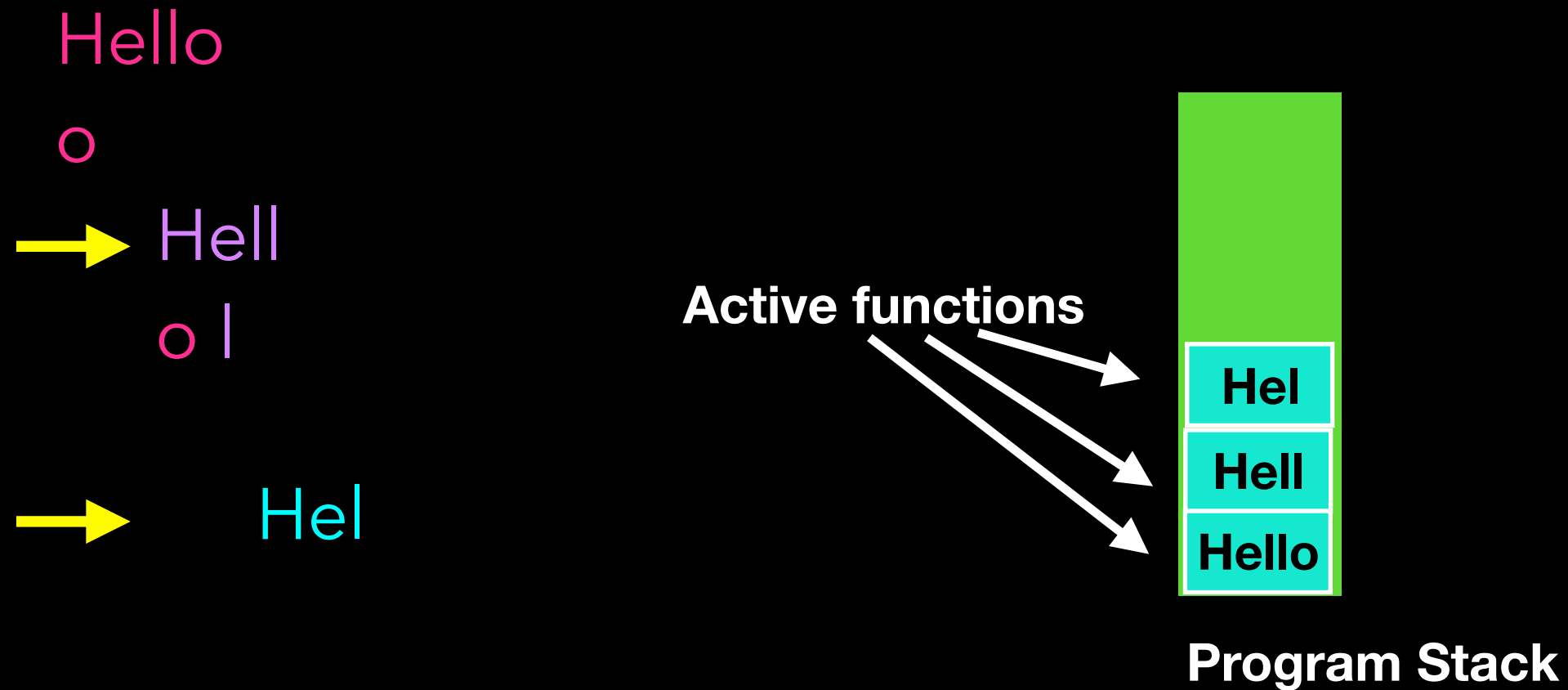
Hello
o
→ Hell
o |

Active functions

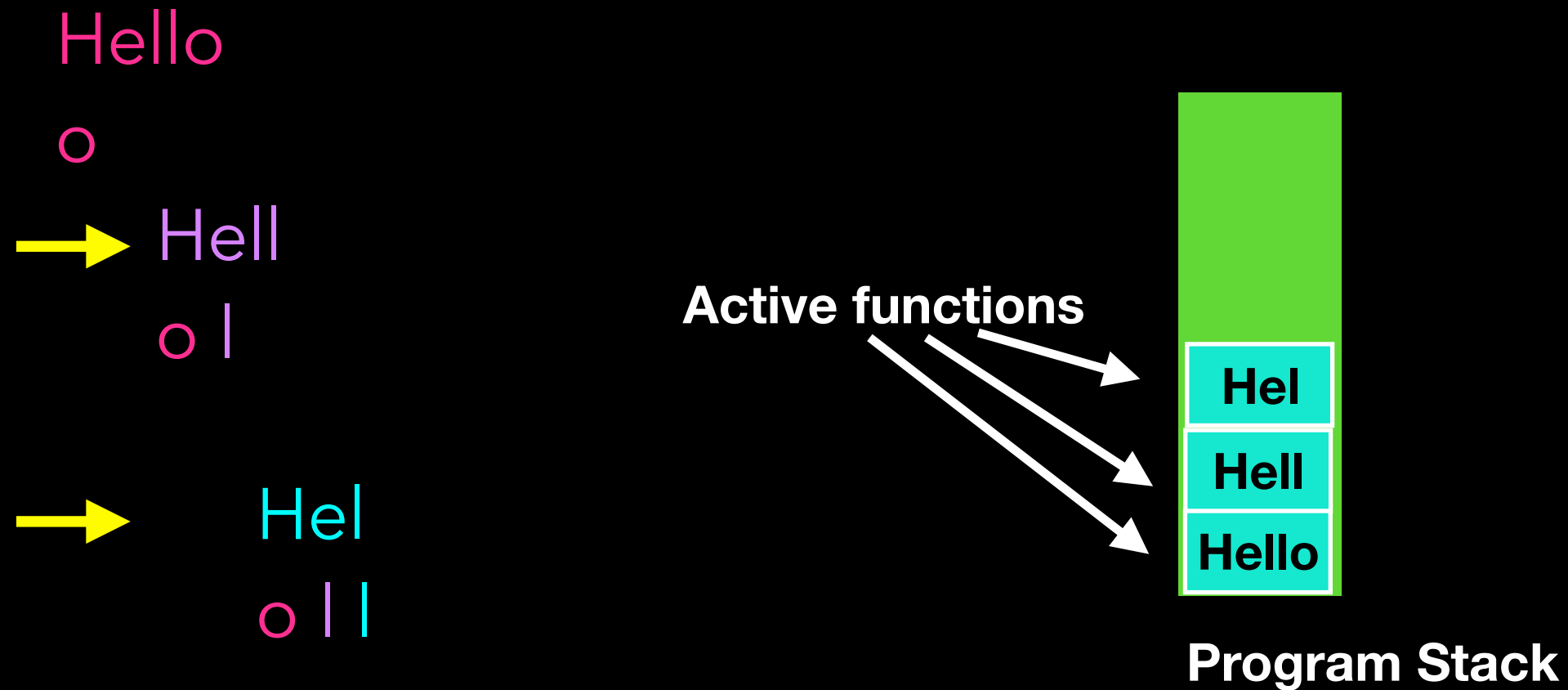


Program Stack

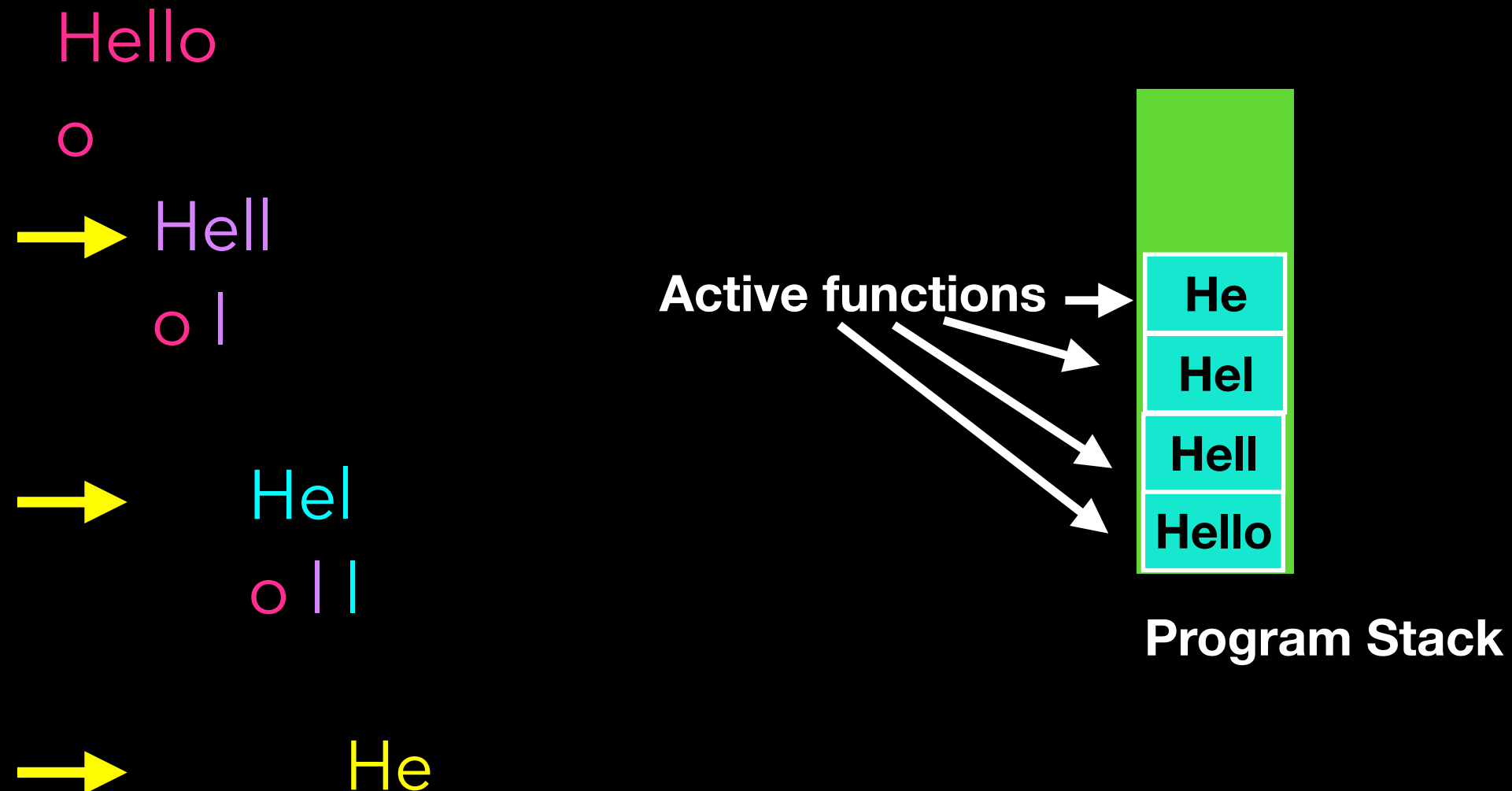
Print String Backwards



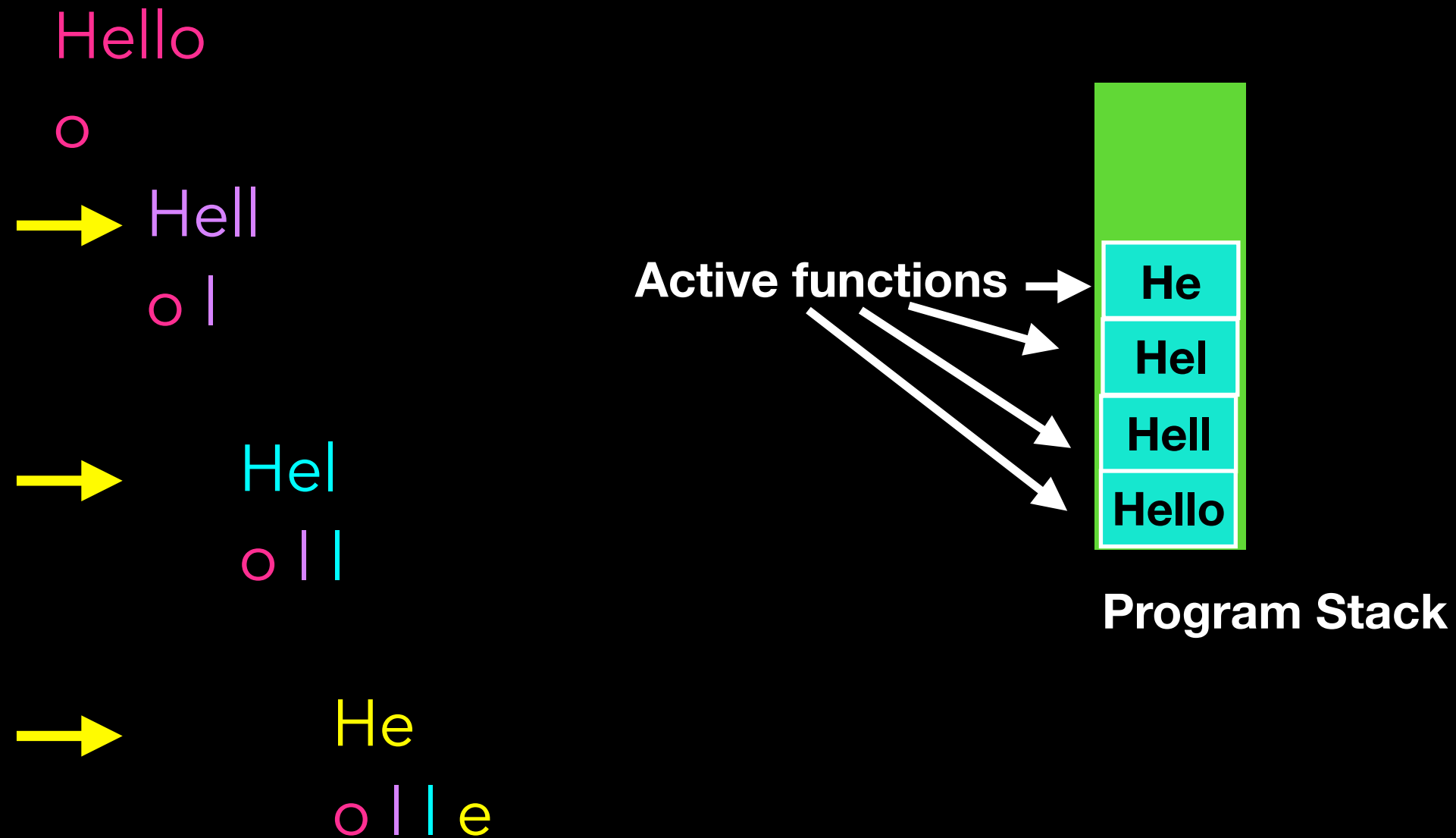
Print String Backwards



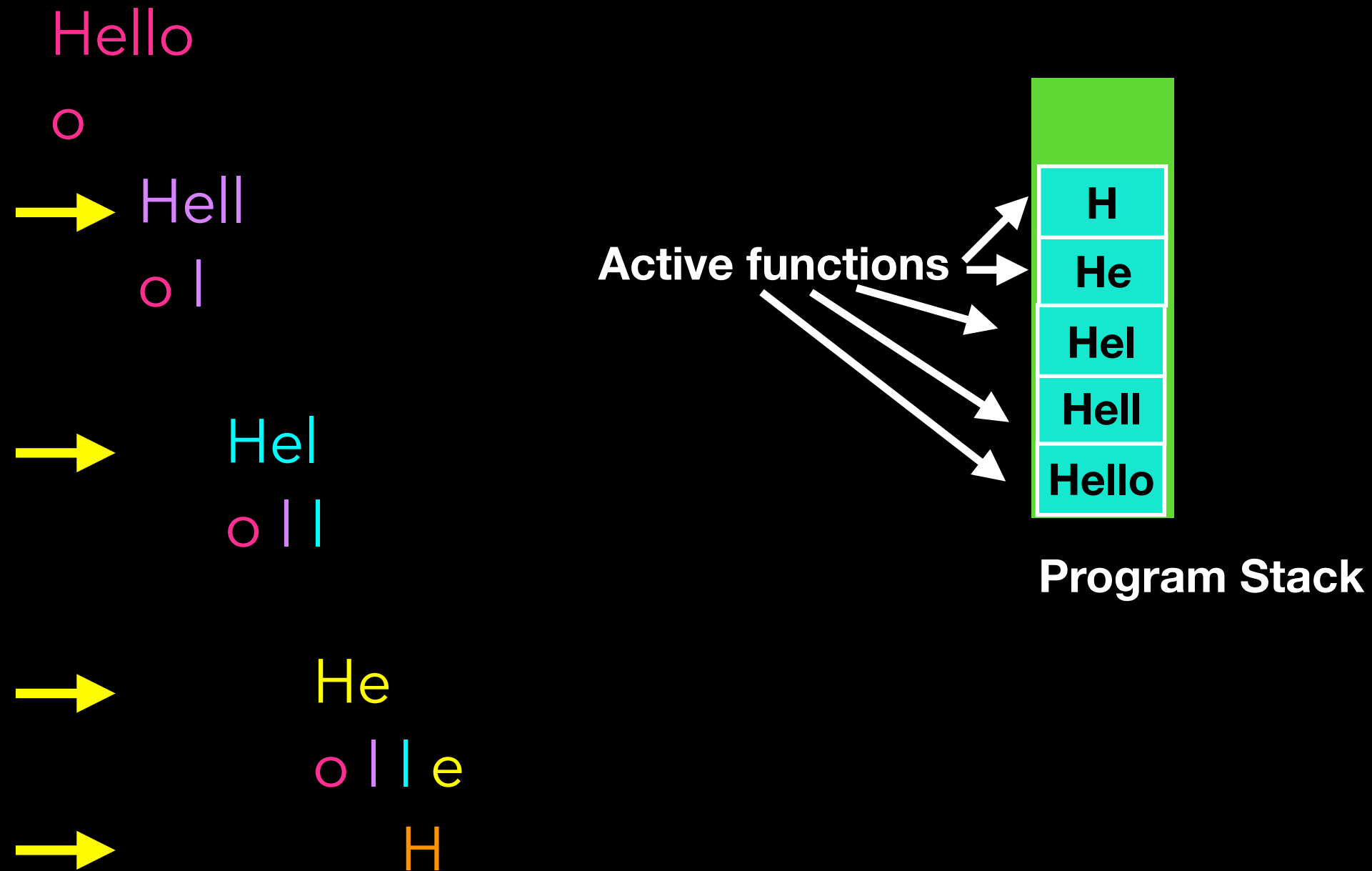
Print String Backwards



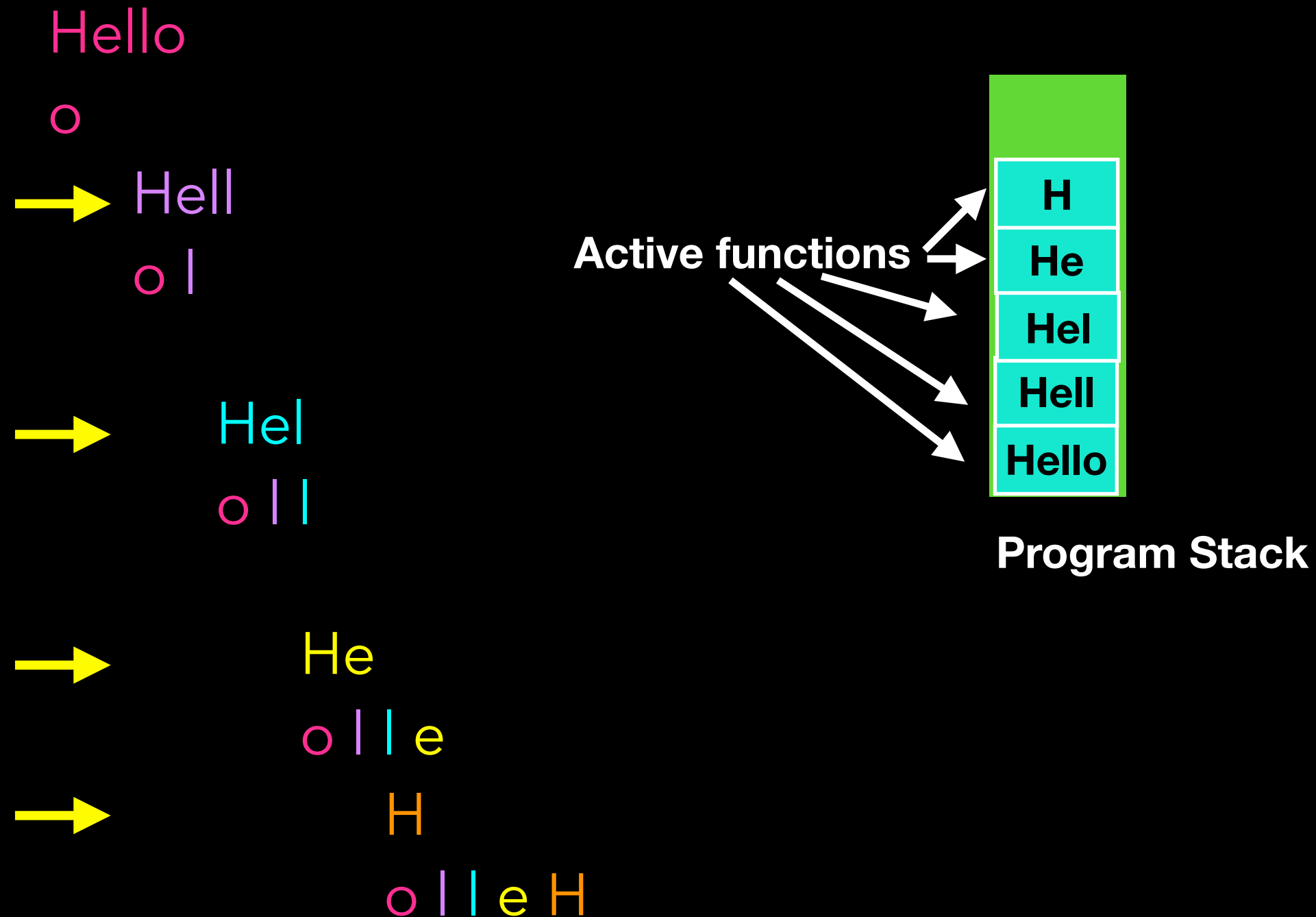
Print String Backwards



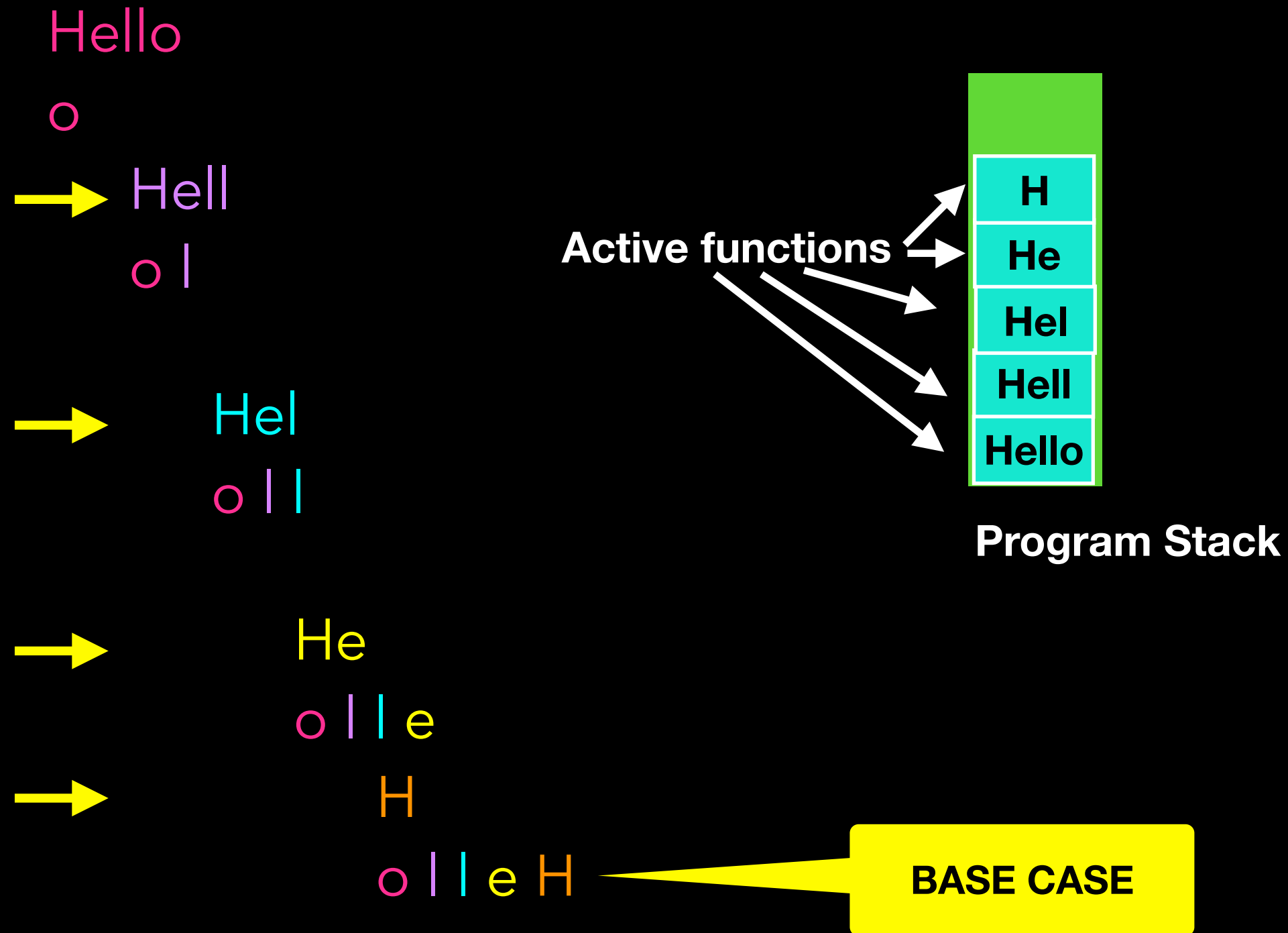
Print String Backwards



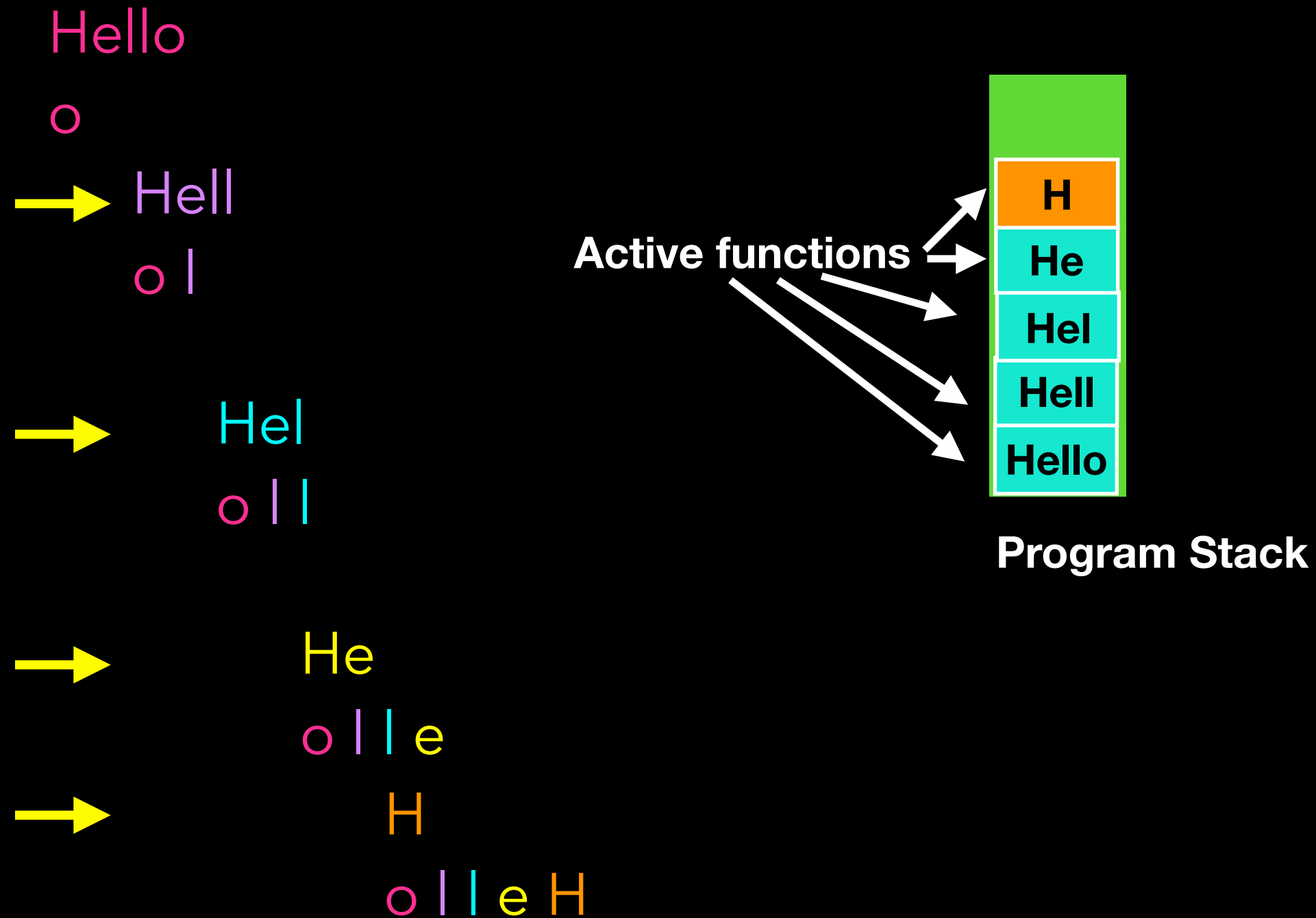
Print String Backwards



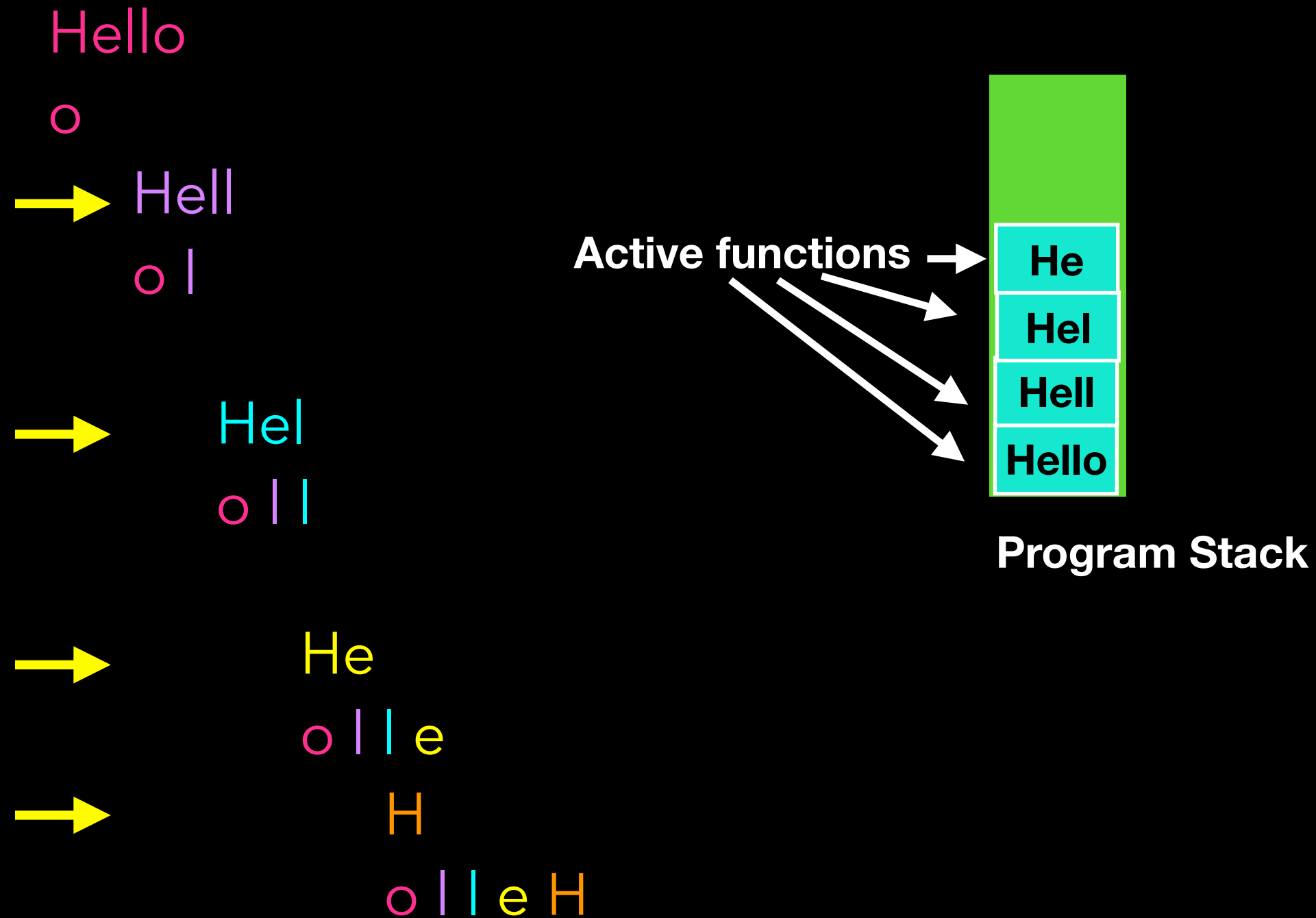
Print String Backwards



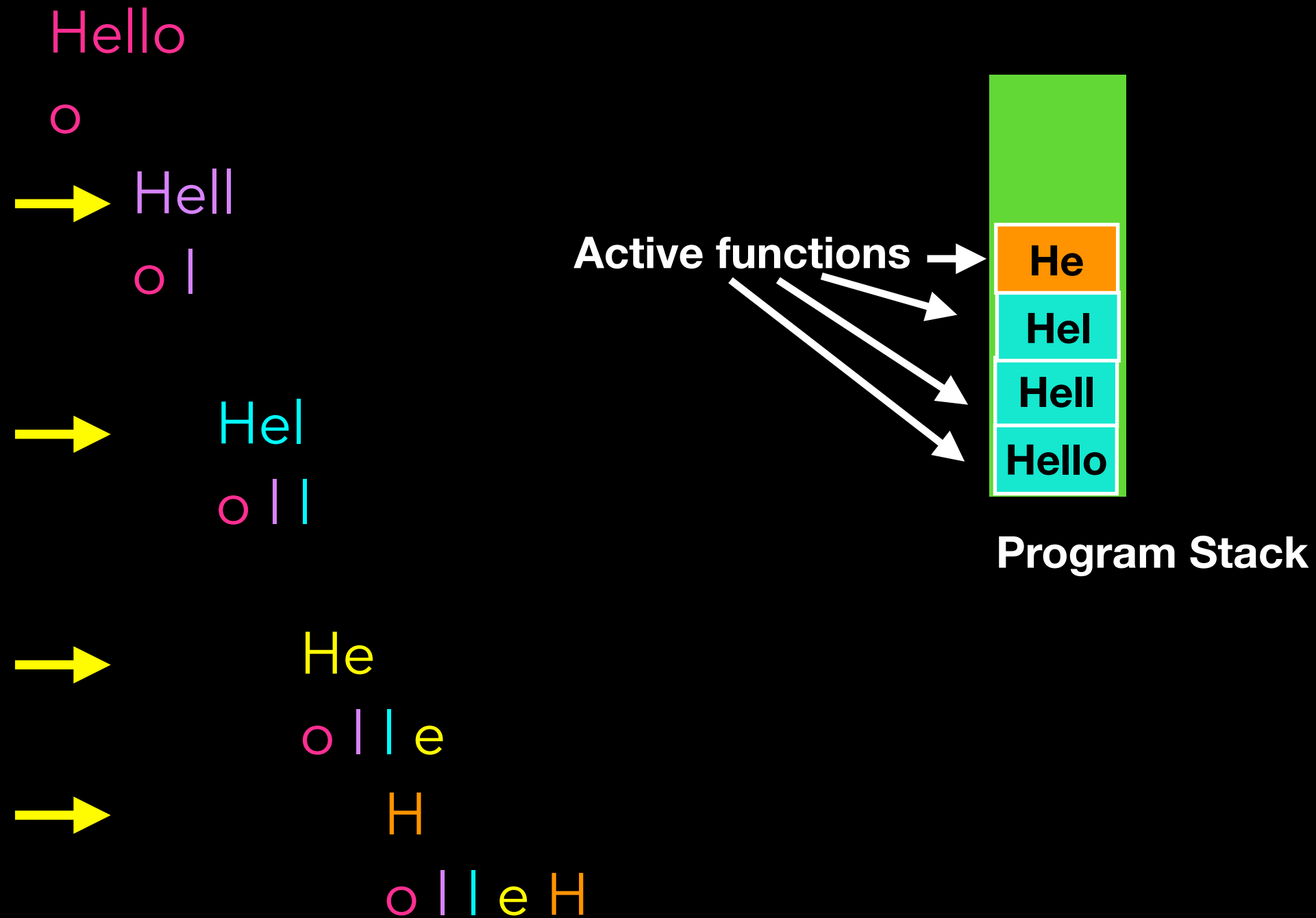
Print String Backwards



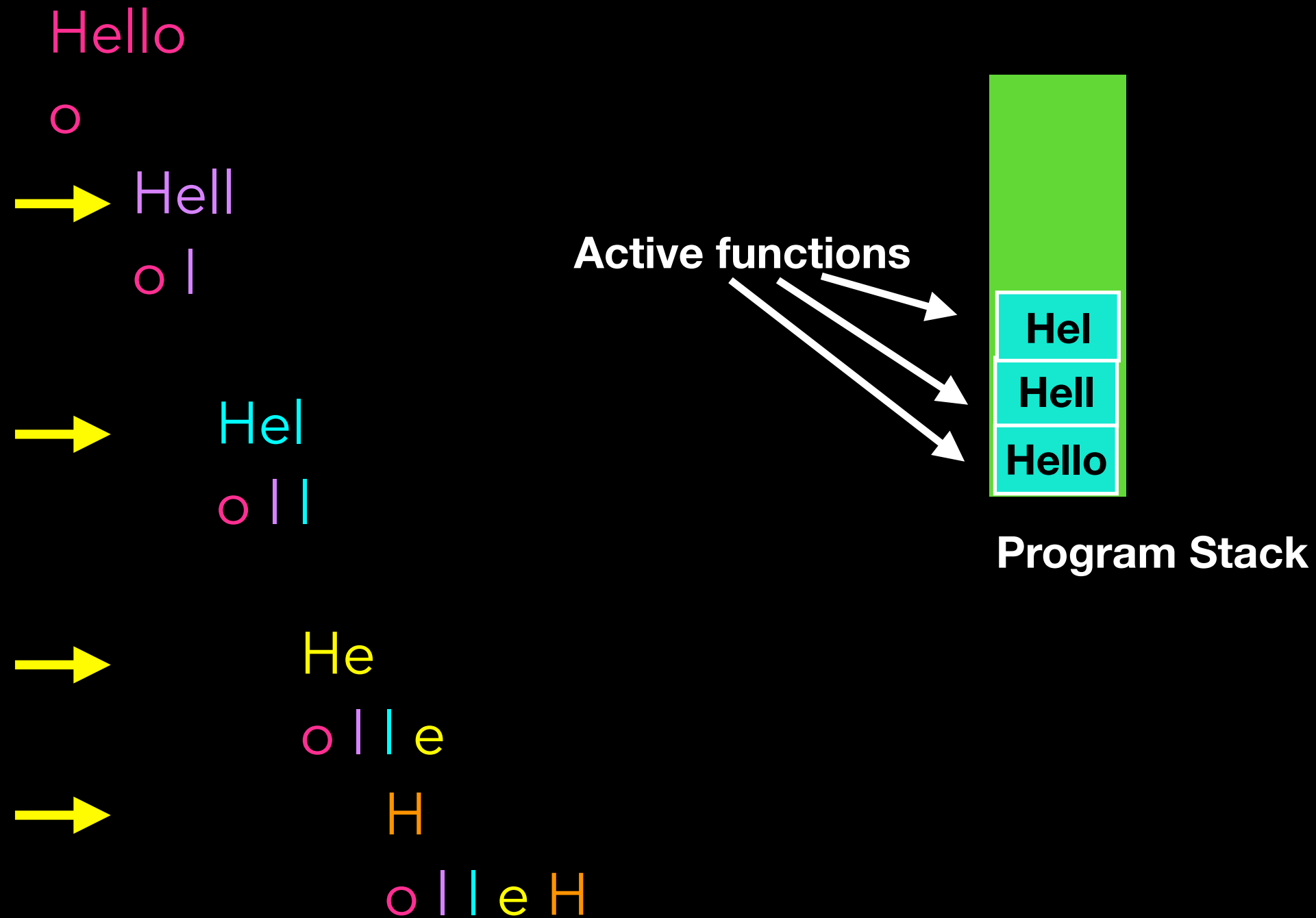
Print String Backwards



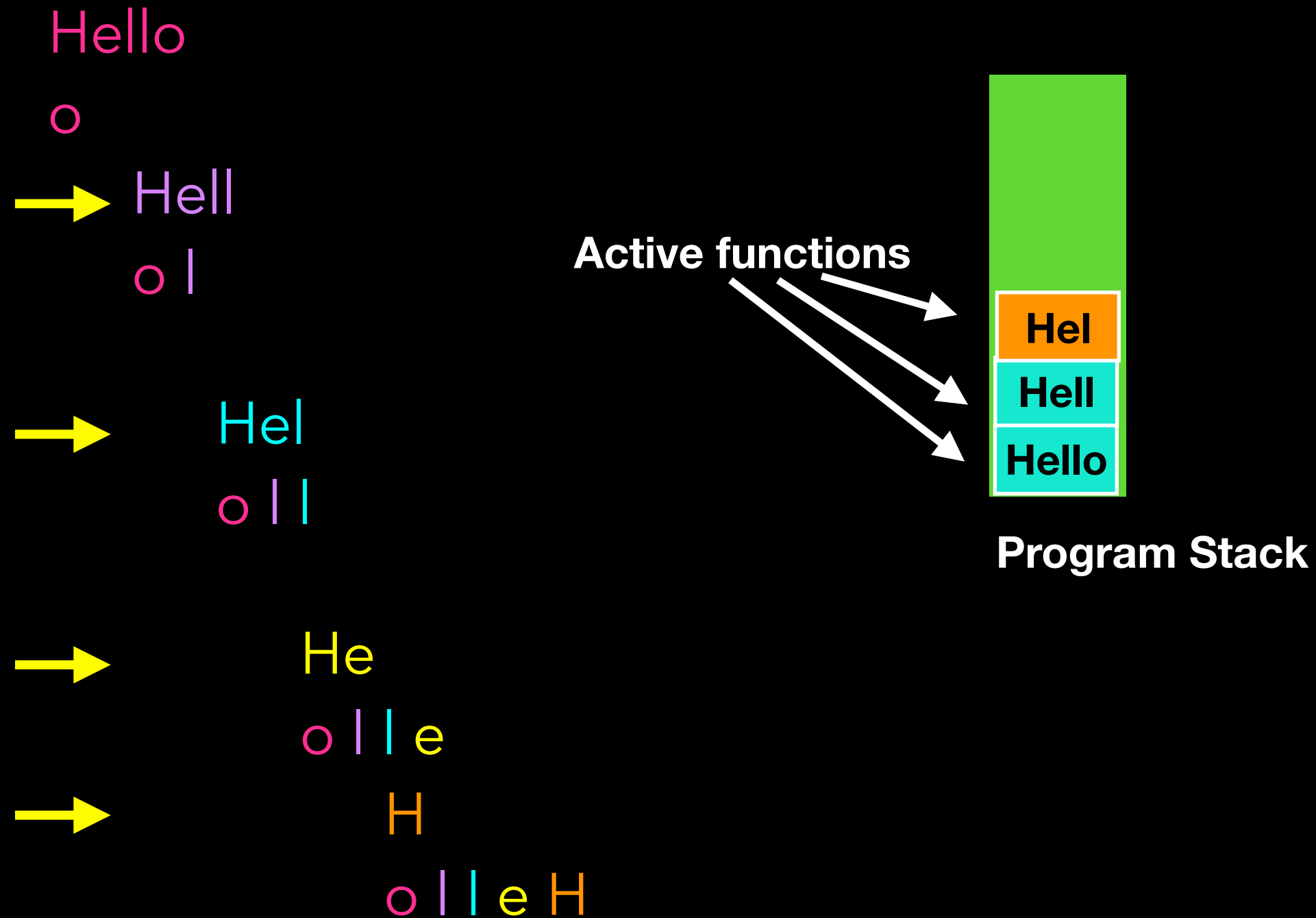
Print String Backwards



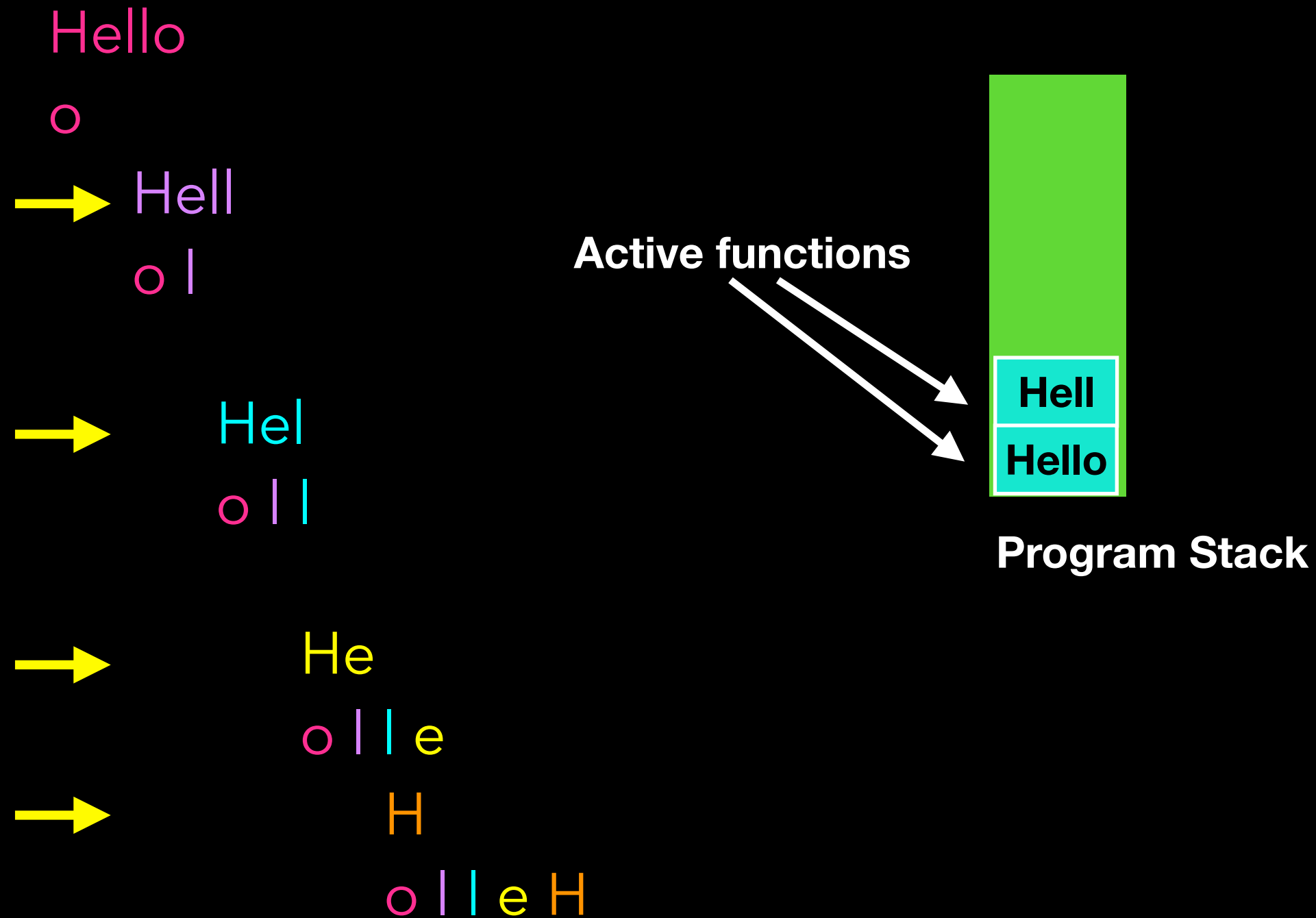
Print String Backwards



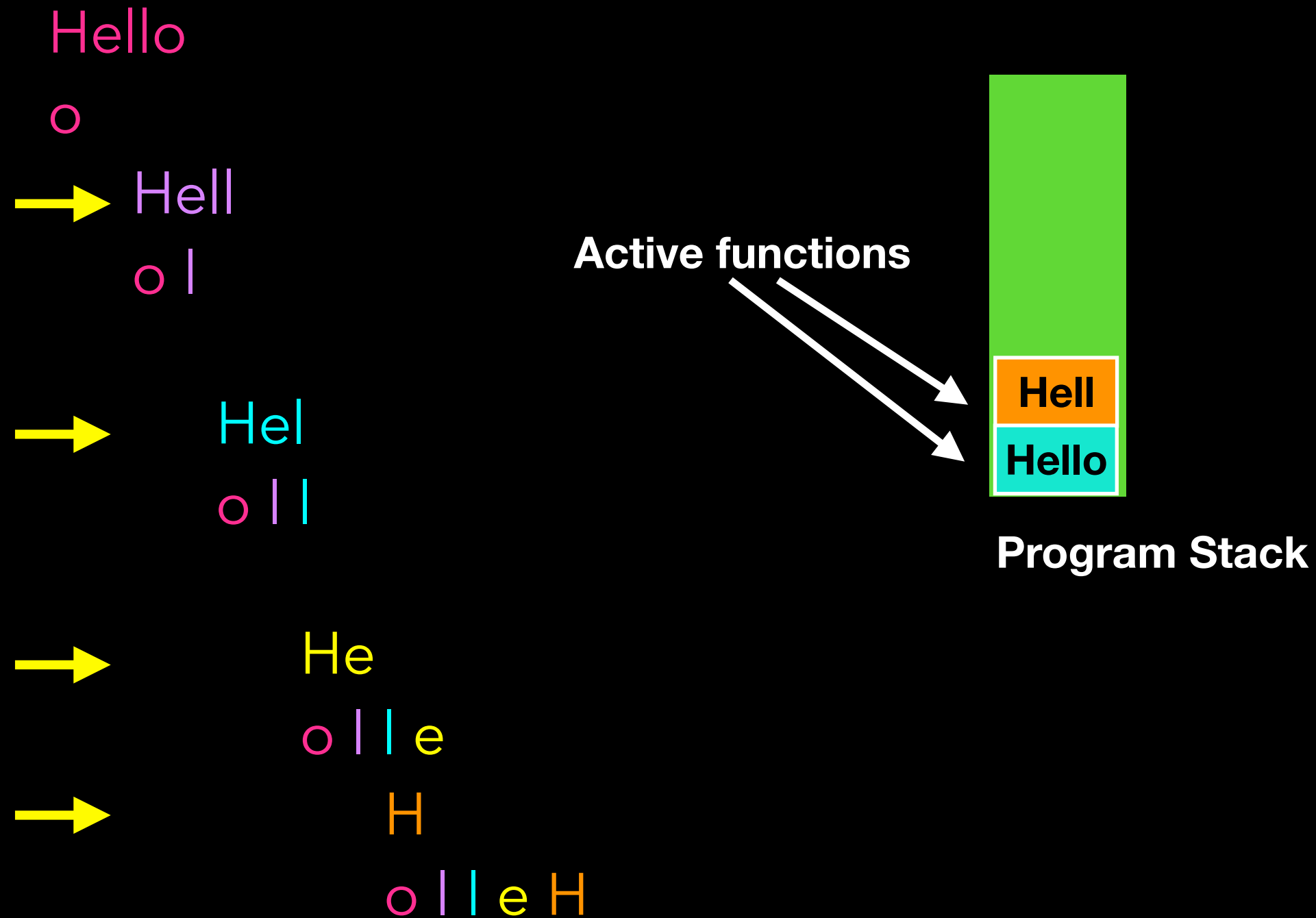
Print String Backwards



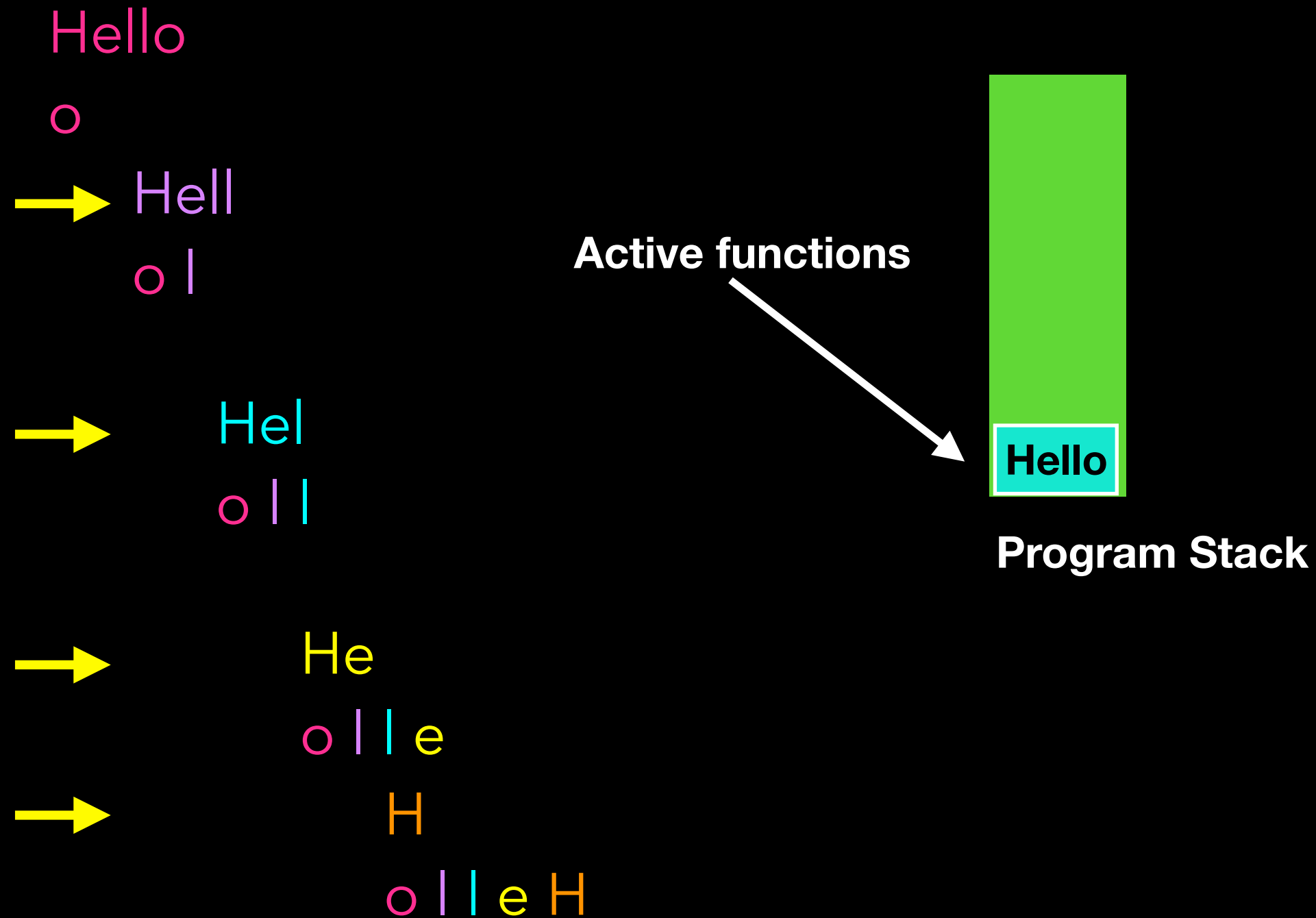
Print String Backwards



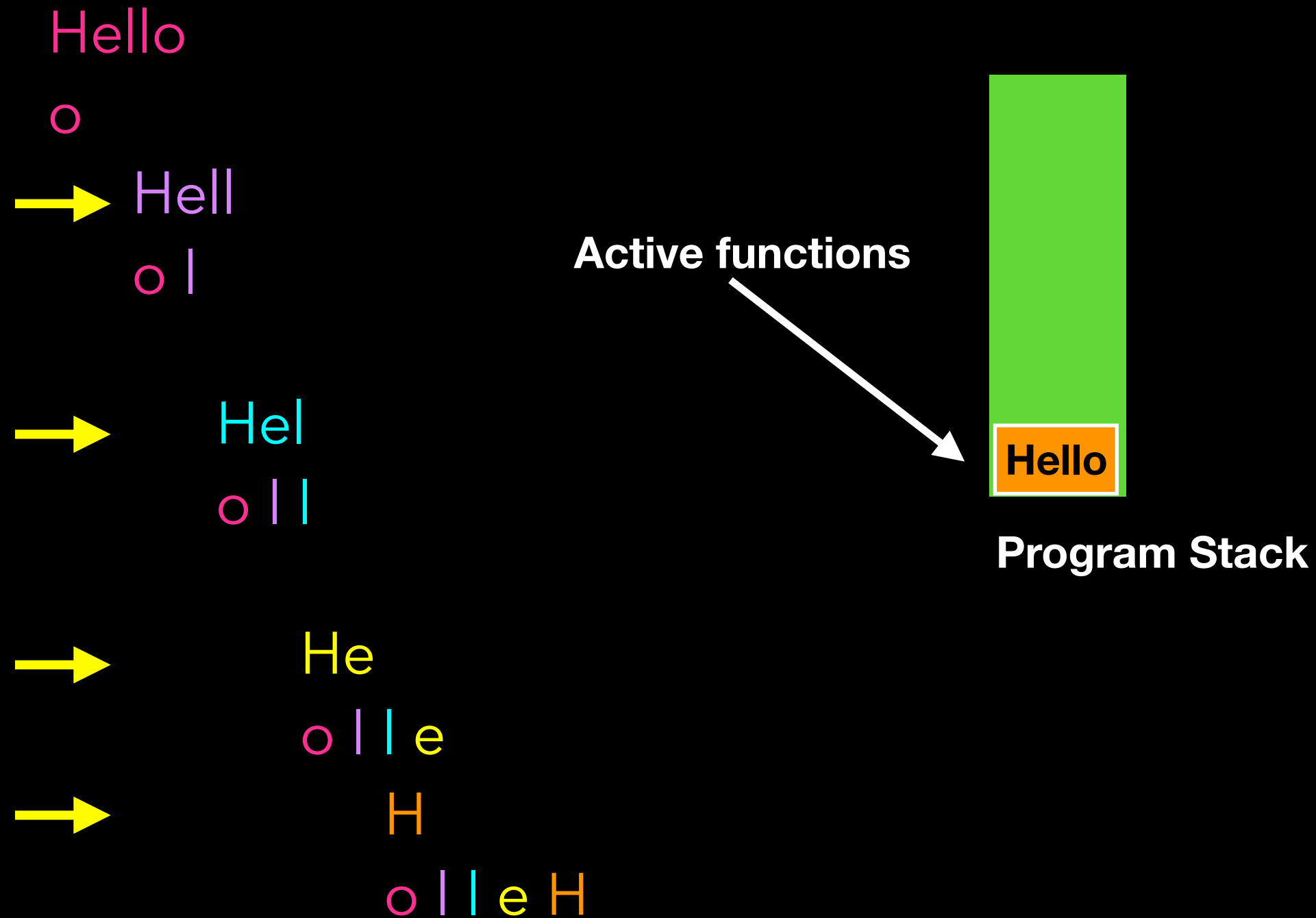
Print String Backwards



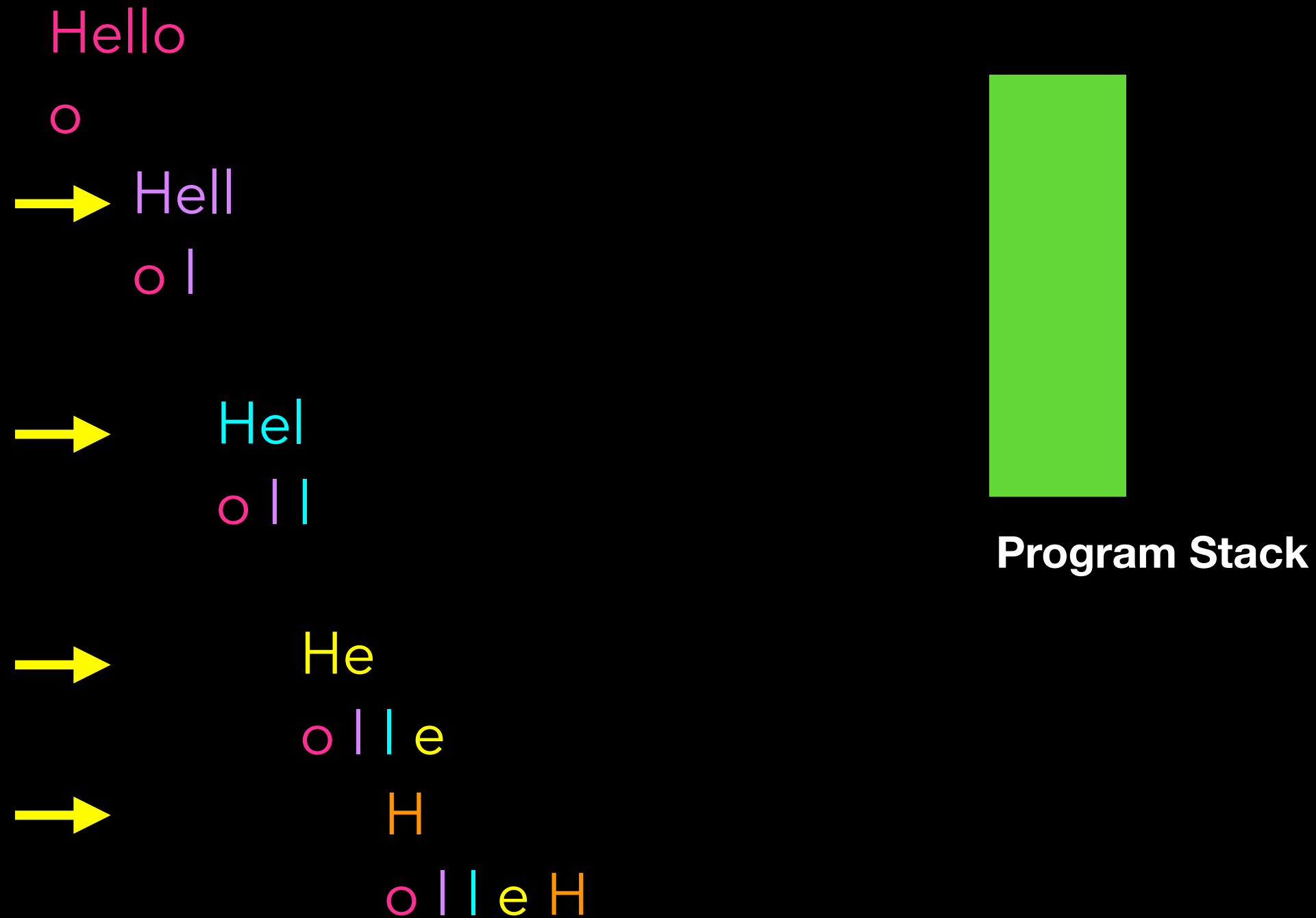
Print String Backwards



Print String Backwards



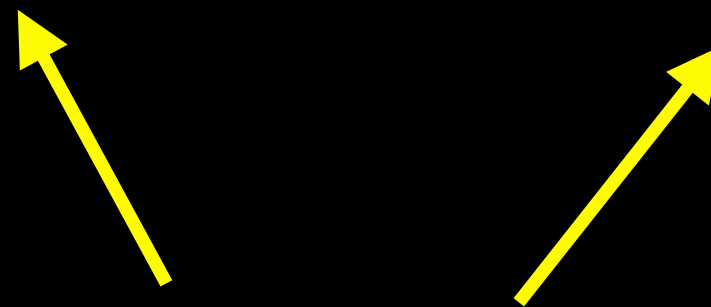
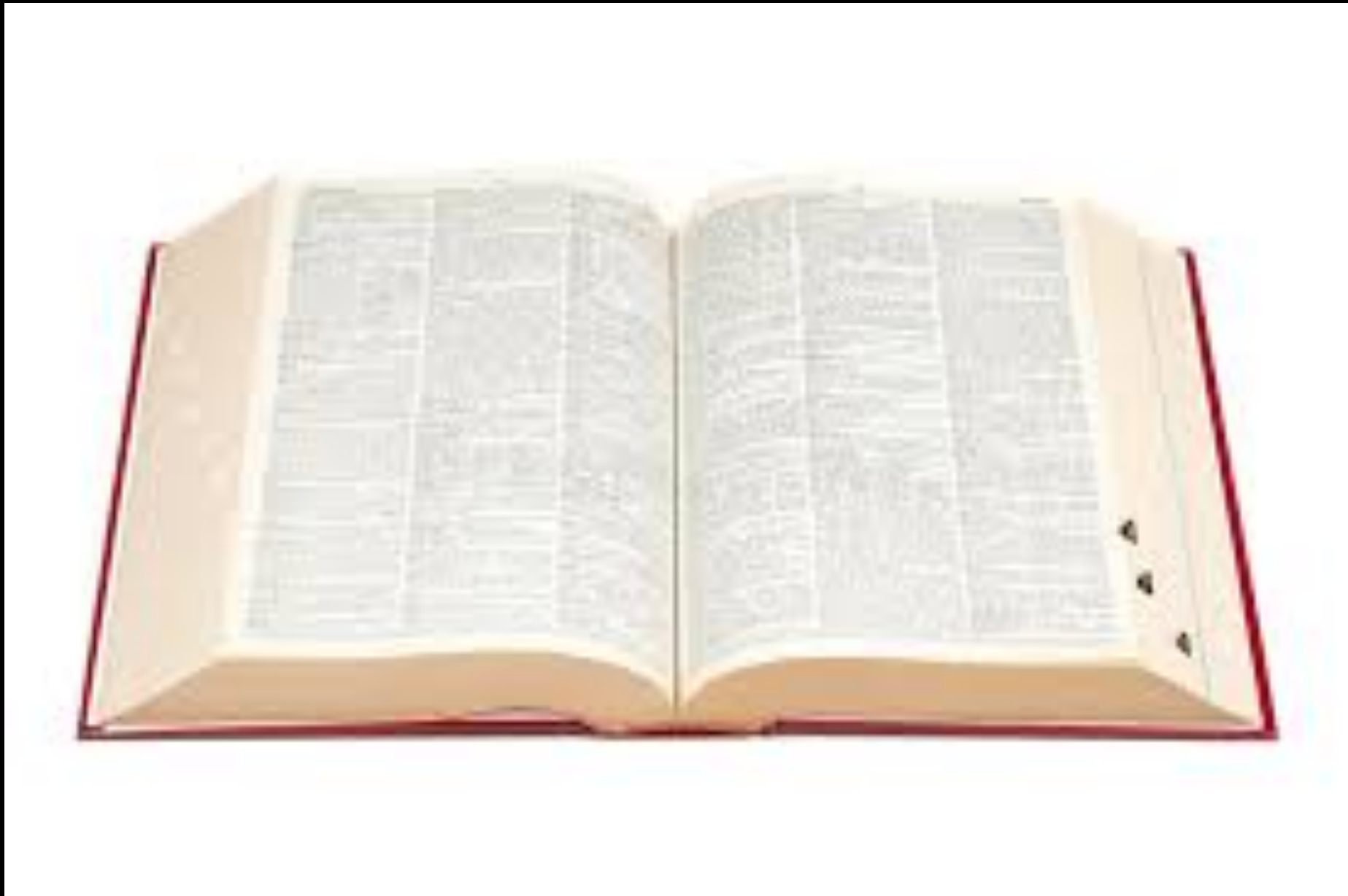
Print String Backwards



Lecture Activity

If I hand you a **printed** dictionary (an actual book) and ask you to find the word “Kalimba”, what do you do?

Write down precise steps (a procedure) as if someone who doesn't know what a dictionary is must follow your instructions.



Look in ?

LOOK FOR WORD "Kalimba" IN DICTIONARY

- Open dictionary at random page

- _ If "Kalimba" is on page FOUND!!!

- Else if "Kalimba" is lexicographically $<$ first word on page

LOOK FOR WORD "Kalimba" IN **LOWER** HALF ←

Recursive Call

- Else if "Kalimba" is lexicographically $>$ last word on page

LOOK FOR WORD "Kalimba" IN **UPPER** HALF ←

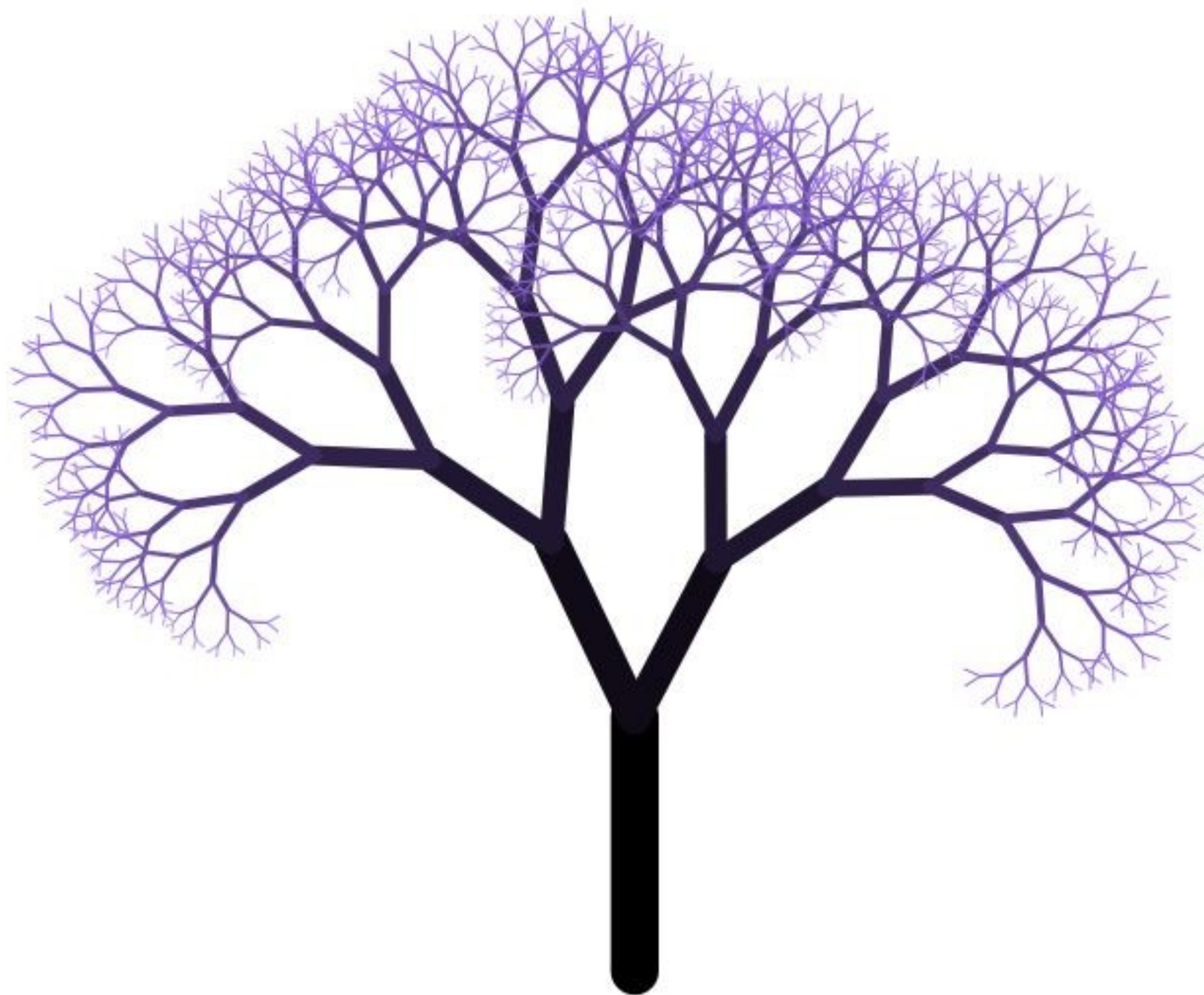
Recursive Call

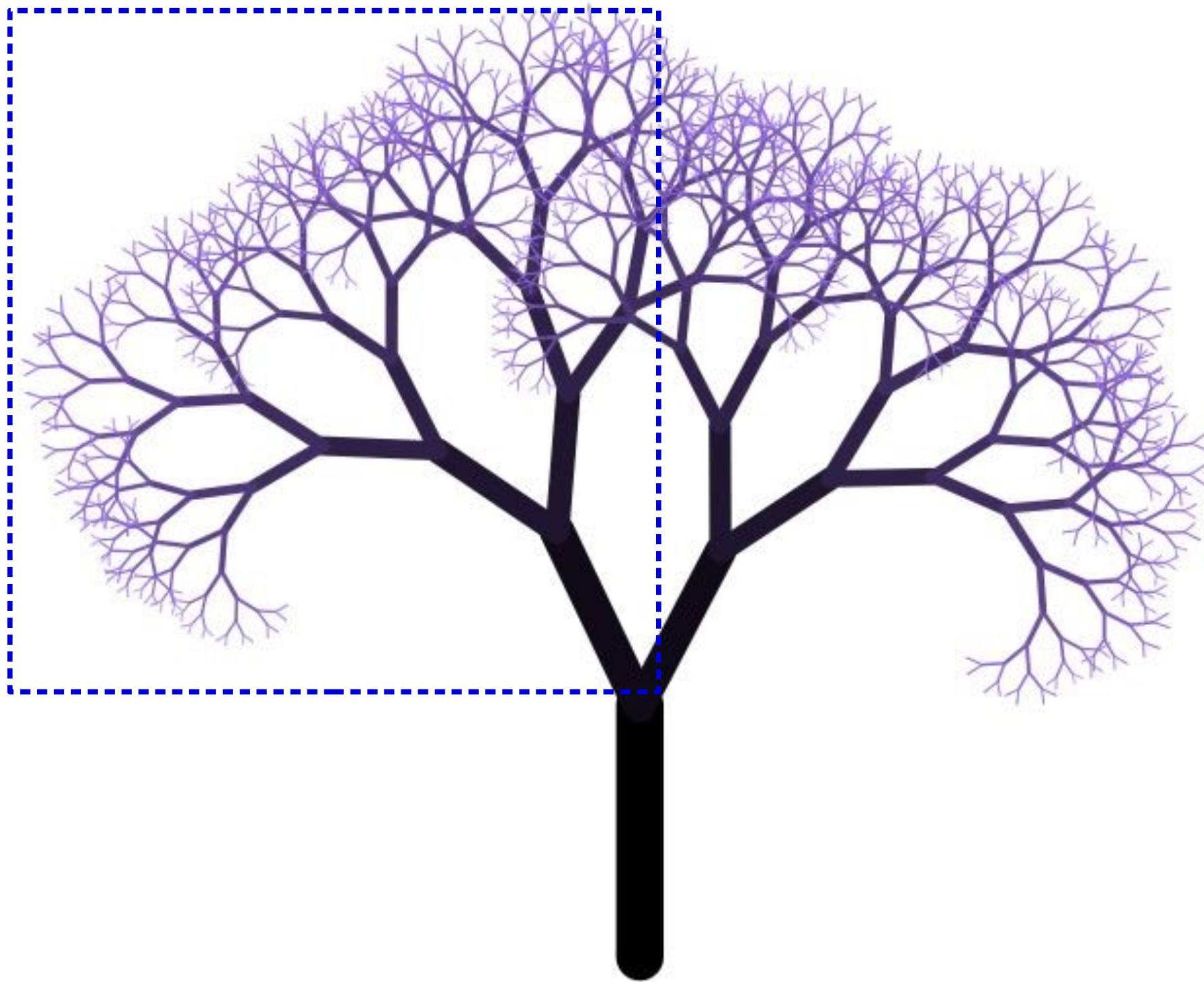
How is this different from recursive solution to print backwards?

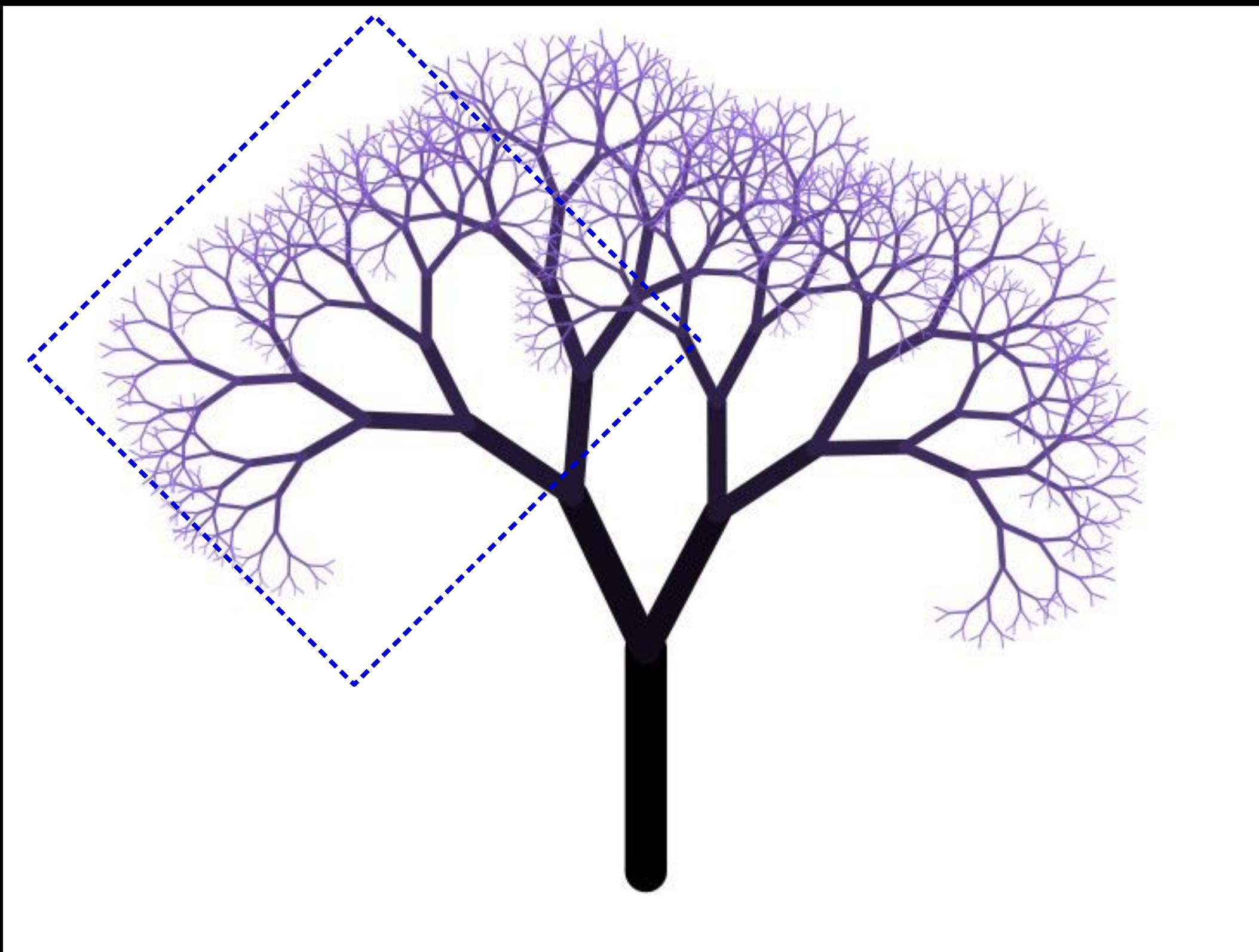
How is this different from recursive solution to print backwards?

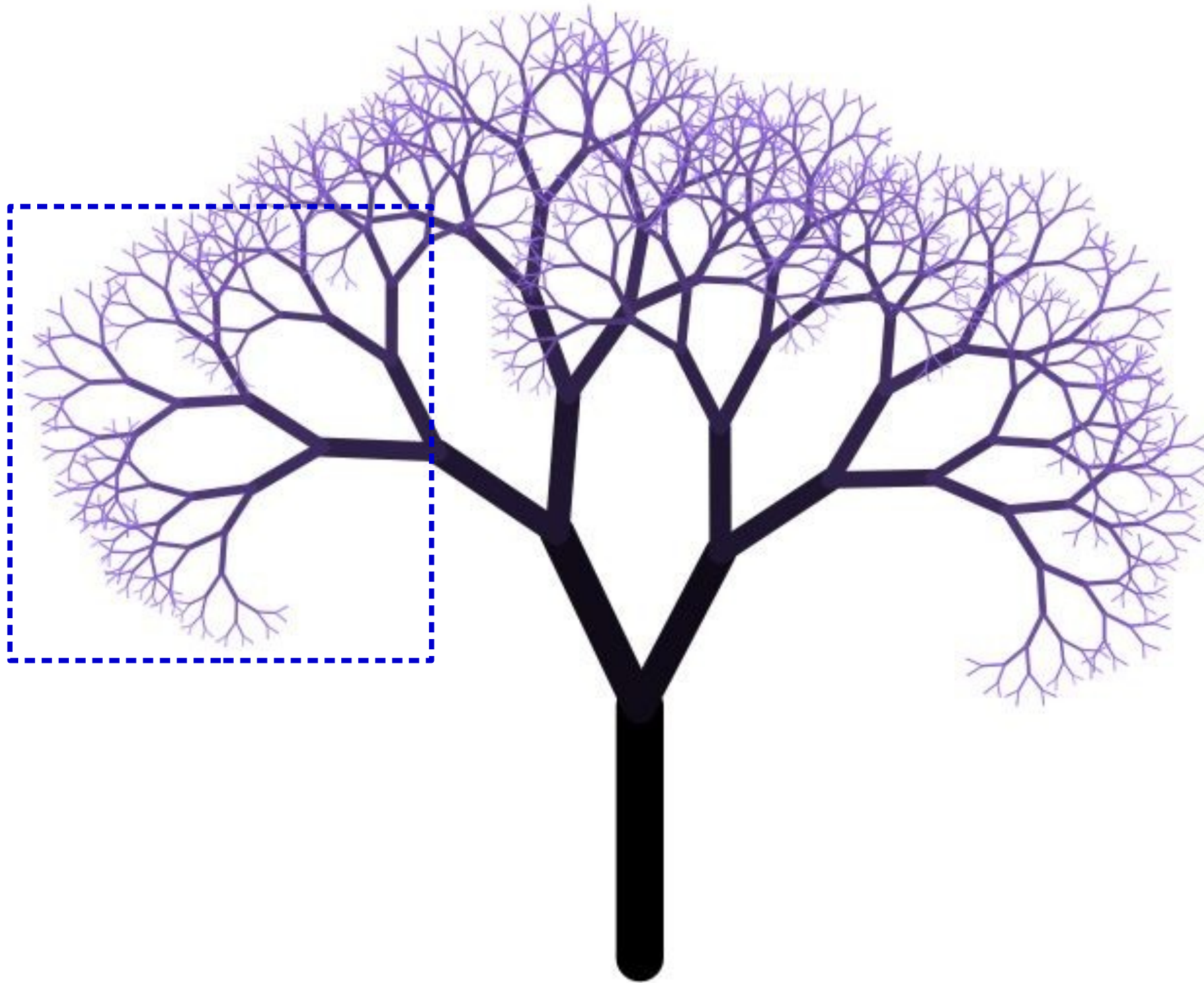
- Two recursive calls
- Execute either one or the other
- Cuts problem in 1/2

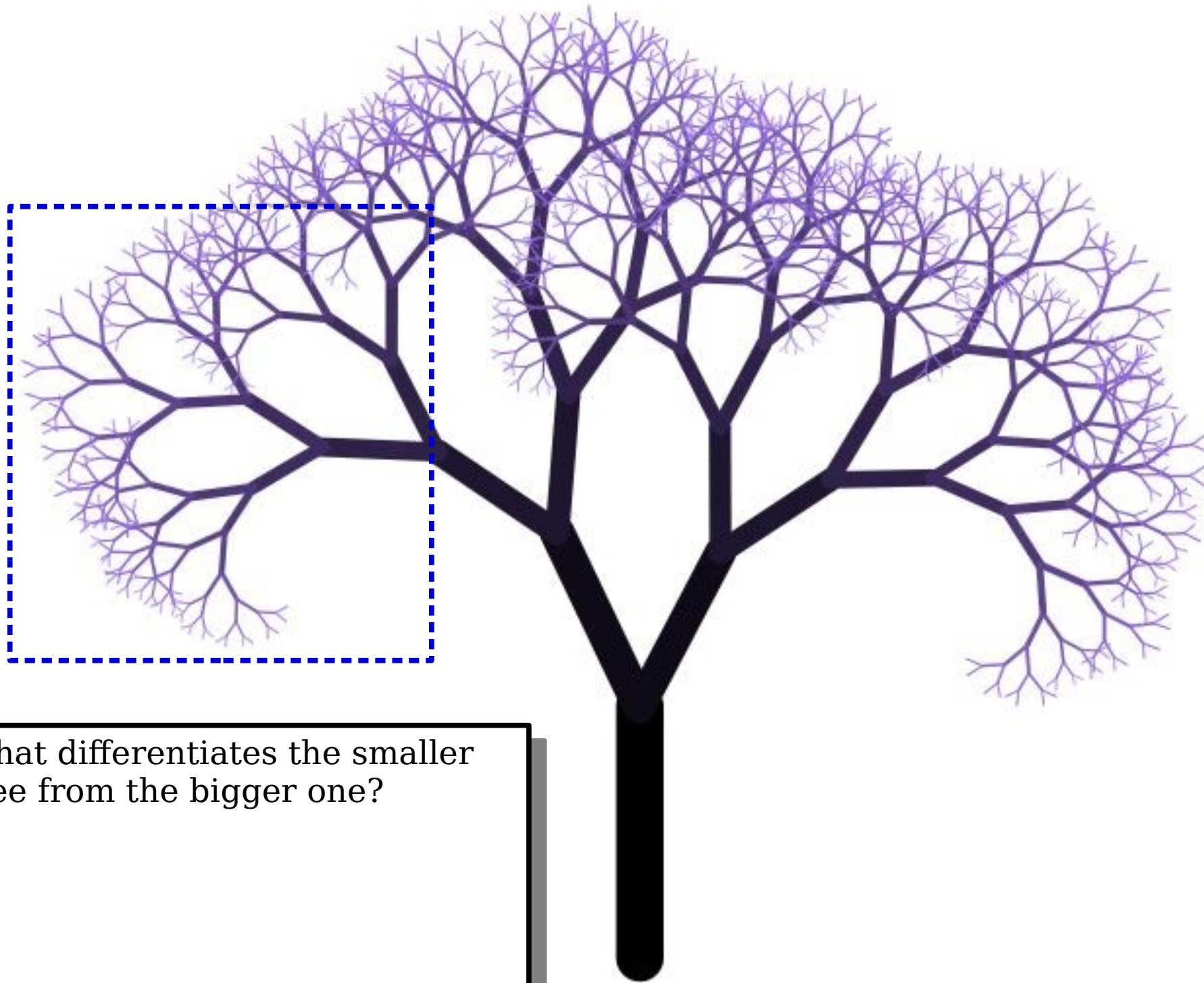
The images in the next slides were adapted from
Keith Schwarz at Stanford University



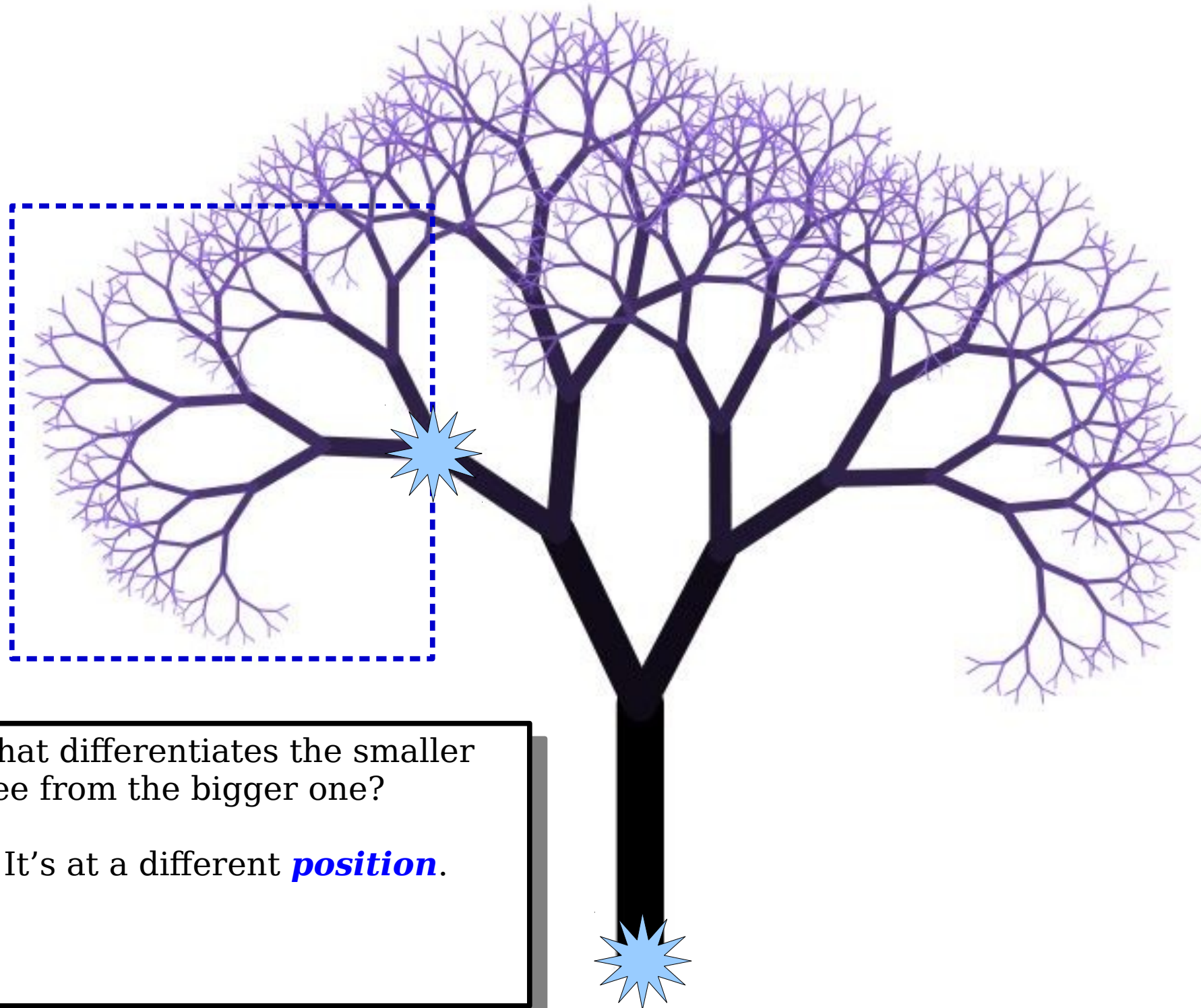






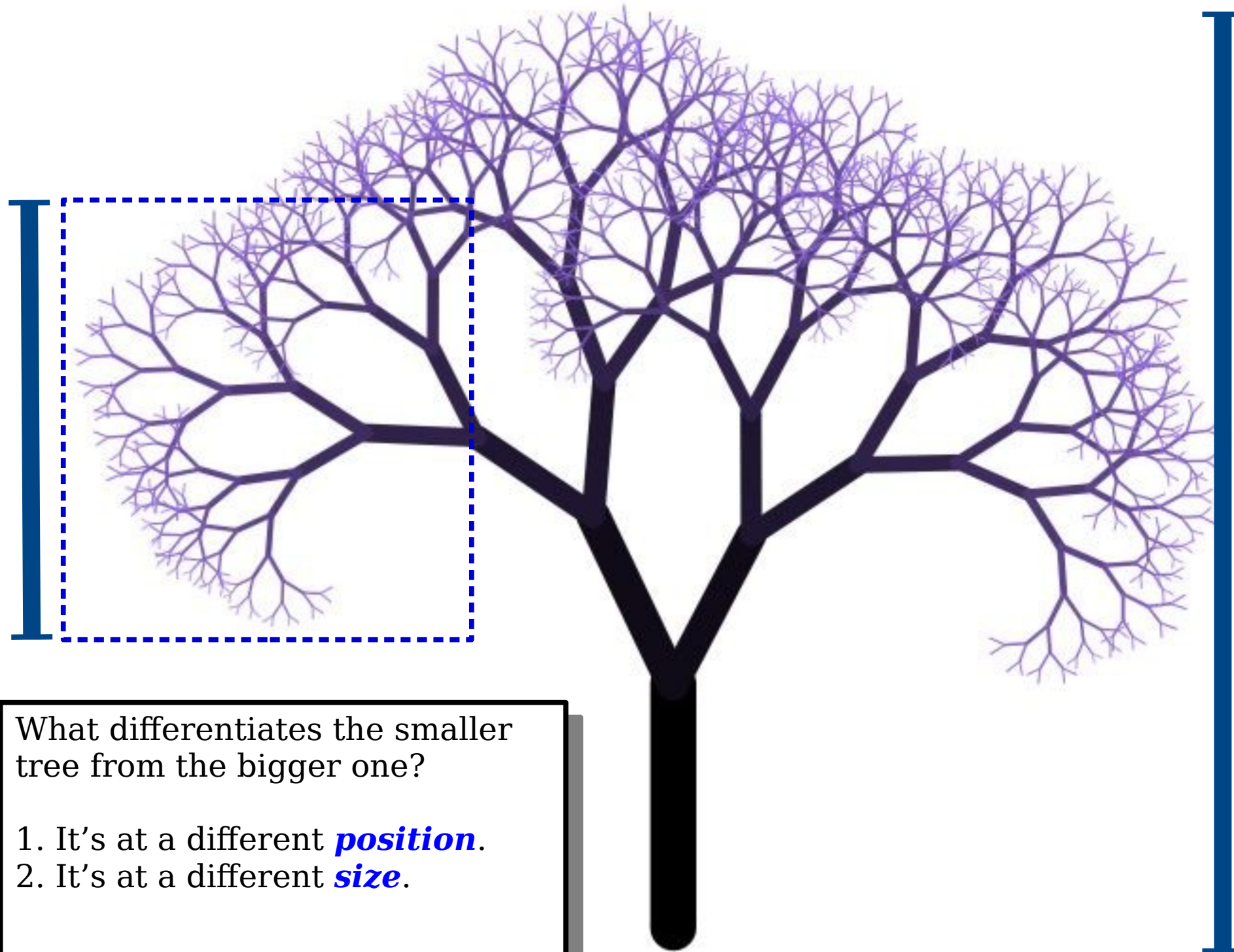


What differentiates the smaller tree from the bigger one?



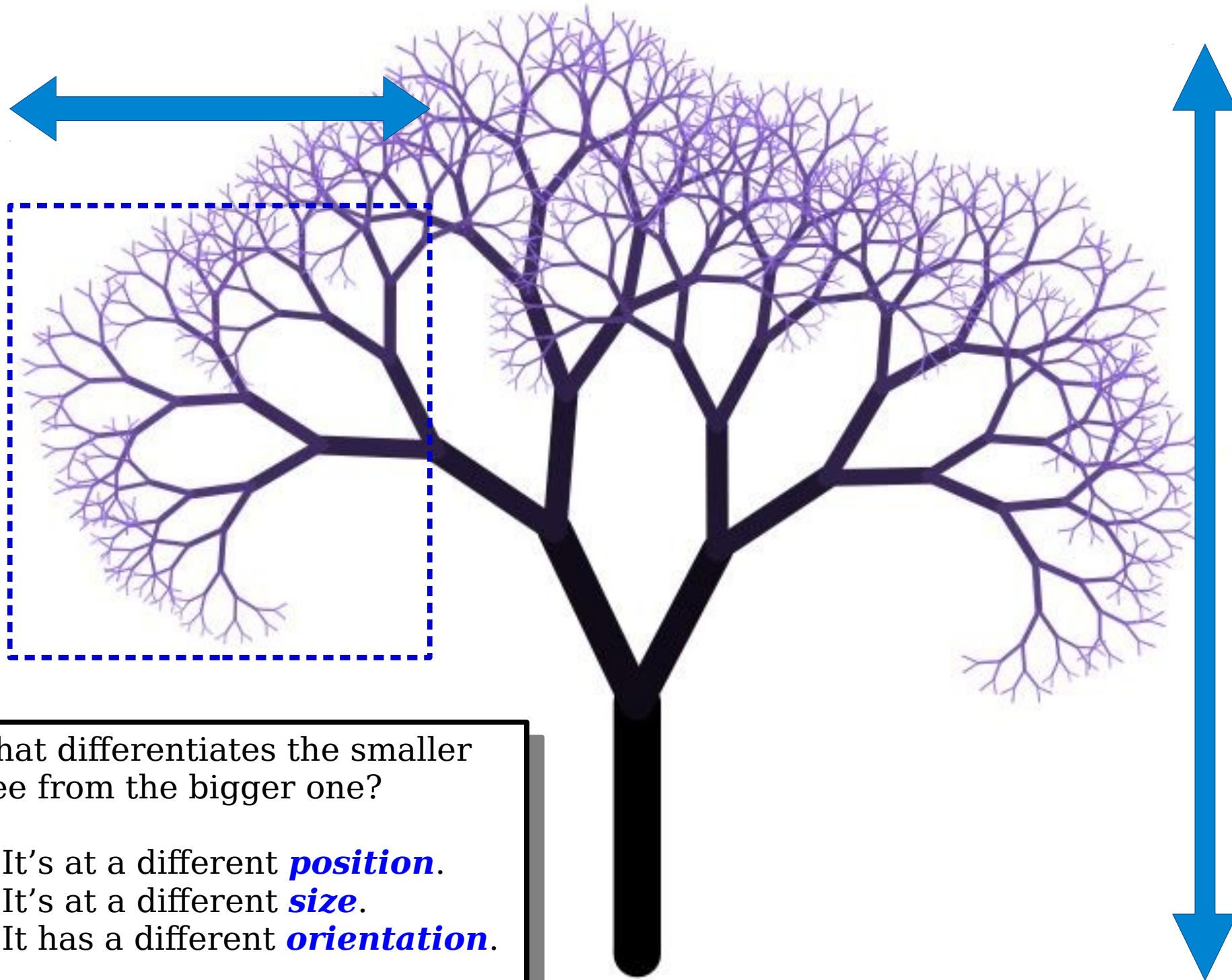
What differentiates the smaller tree from the bigger one?

1. It's at a different ***position***.



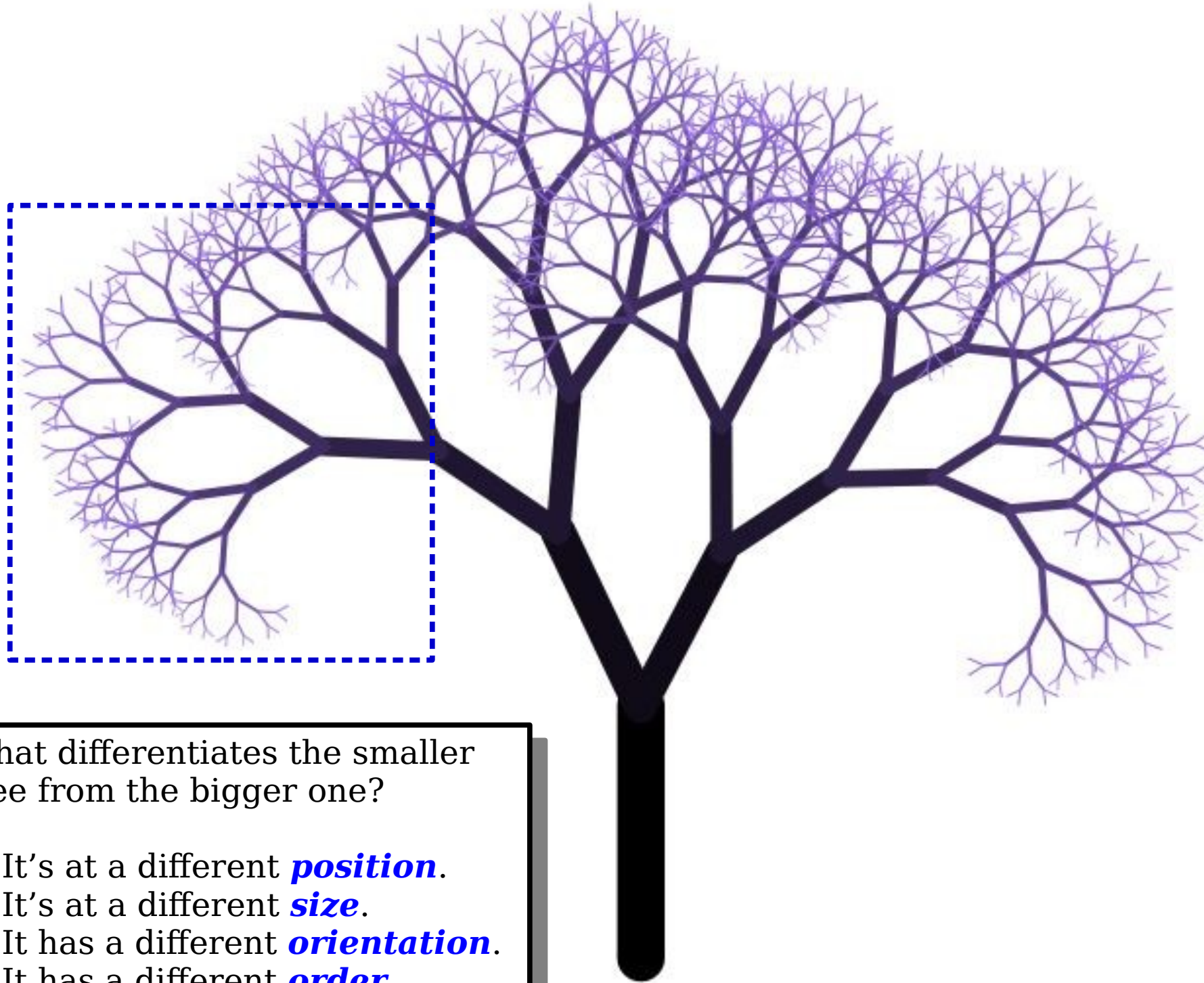
What differentiates the smaller tree from the bigger one?

1. It's at a different **position**.
2. It's at a different **size**.



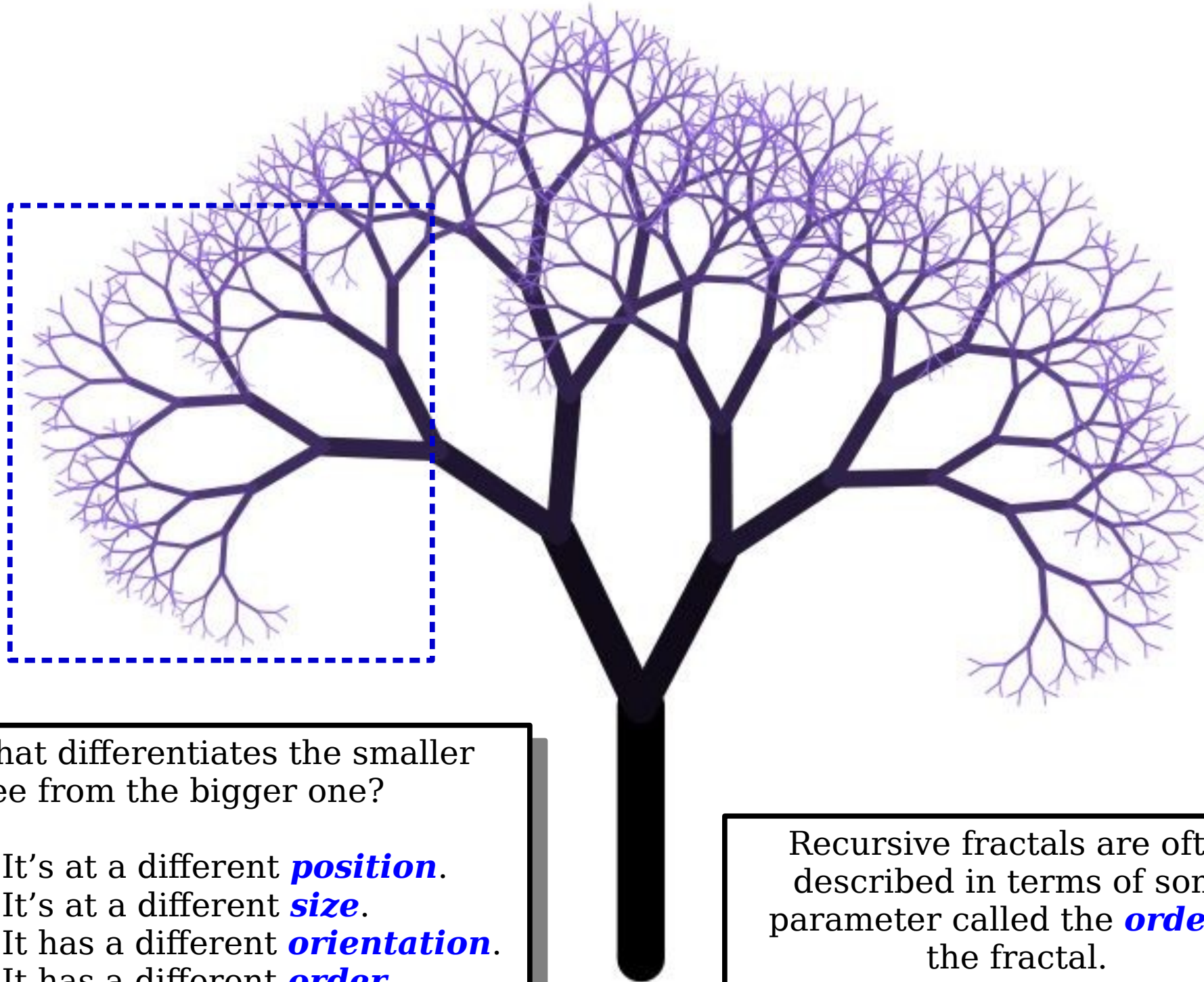
What differentiates the smaller tree from the bigger one?

1. It's at a different **position**.
2. It's at a different **size**.
3. It has a different **orientation**.



What differentiates the smaller tree from the bigger one?

1. It's at a different **position**.
2. It's at a different **size**.
3. It has a different **orientation**.
4. It has a different **order**.



What differentiates the smaller tree from the bigger one?

1. It's at a different **position**.
2. It's at a different **size**.
3. It has a different **orientation**.
4. It has a different **order**.

Recursive fractals are often described in terms of some parameter called the **order** of the fractal.

An order-0 tree.

What differentiates the smaller tree from the bigger one?

1. It's at a different **position**.
2. It's at a different **size**.
3. It has a different **orientation**.
4. It has a different **order**.

Recursive fractals are often described in terms of some parameter called the **order** of the fractal.

An order-1 tree.

What differentiates the smaller tree from the bigger one?

1. It's at a different **position**.
2. It's at a different **size**.
3. It has a different **orientation**.
4. It has a different **order**.



Recursive fractals are often described in terms of some parameter called the **order** of the fractal.

An order-2 tree.

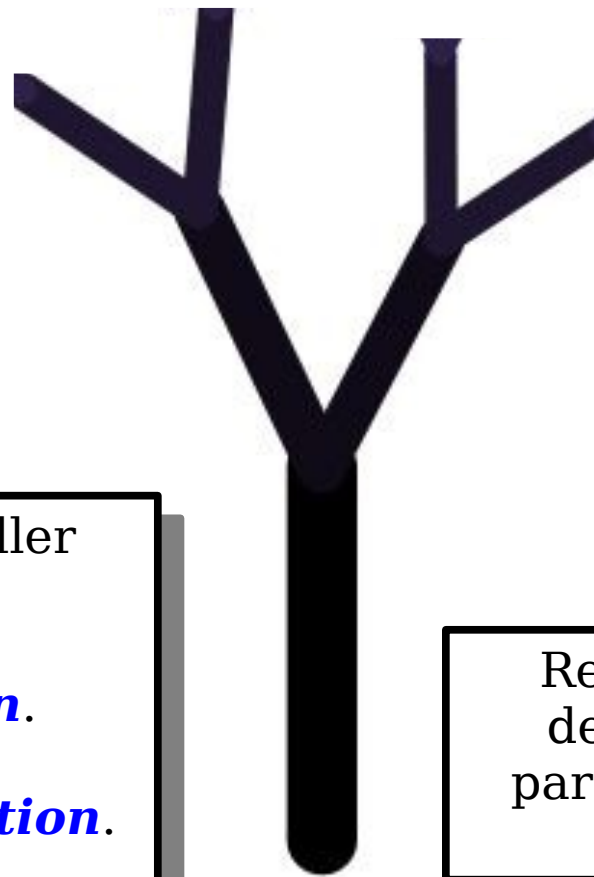


What differentiates the smaller tree from the bigger one?

1. It's at a different **position**.
2. It's at a different **size**.
3. It has a different **orientation**.
4. It has a different **order**.

Recursive fractals are often described in terms of some parameter called the **order** of the fractal.

An order-3 tree.

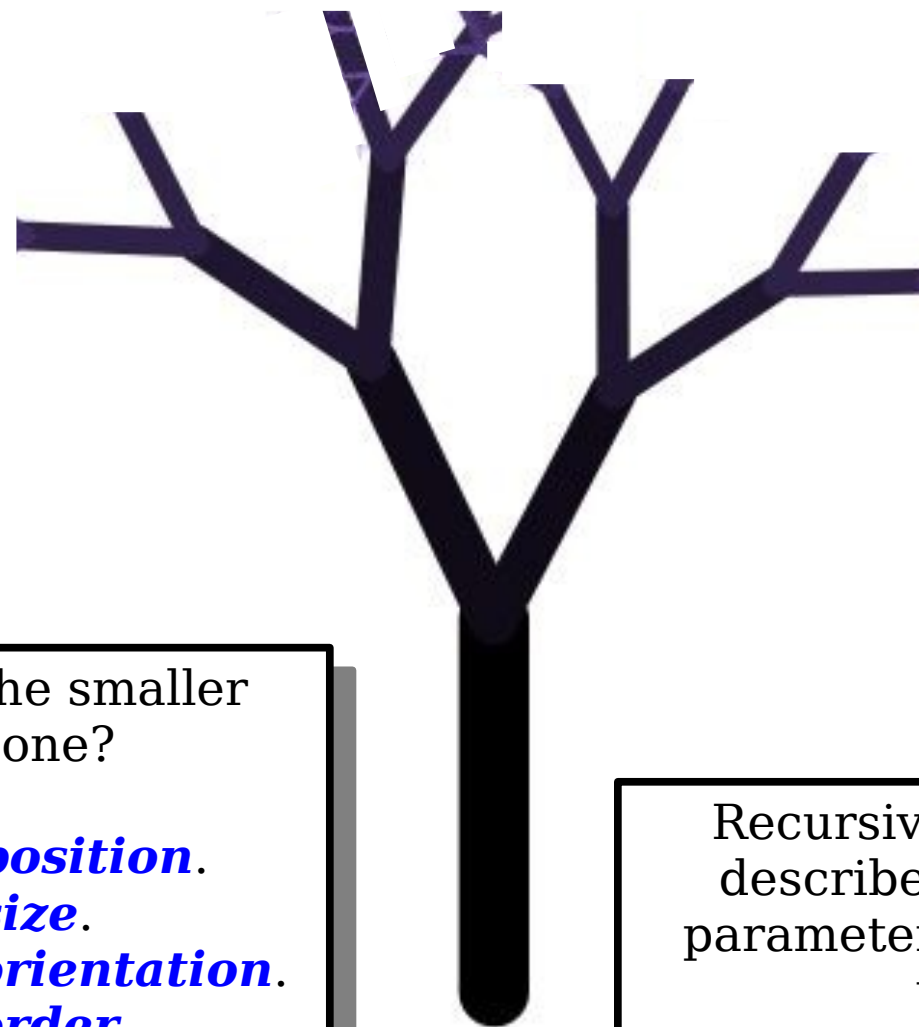


What differentiates the smaller tree from the bigger one?

1. It's at a different **position**.
2. It's at a different **size**.
3. It has a different **orientation**.
4. It has a different **order**.

Recursive fractals are often described in terms of some parameter called the **order** of the fractal.

An order-4 tree.

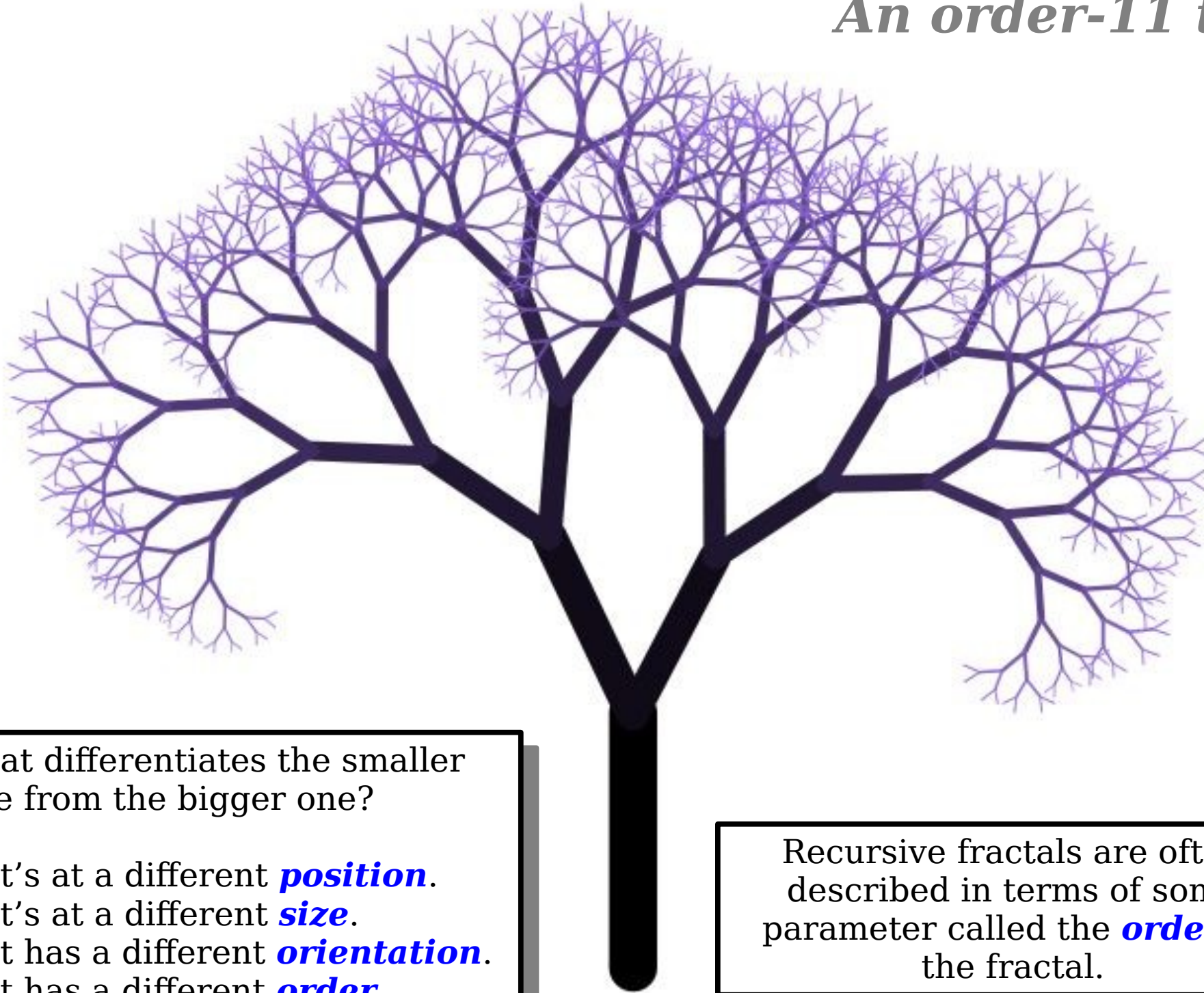


What differentiates the smaller tree from the bigger one?

1. It's at a different **position**.
2. It's at a different **size**.
3. It has a different **orientation**.
4. It has a different **order**.

Recursive fractals are often described in terms of some parameter called the **order** of the fractal.

An order-11 tree.

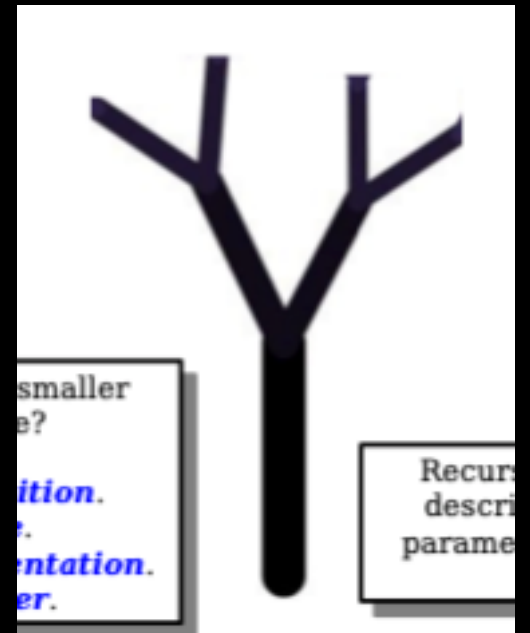


What differentiates the smaller tree from the bigger one?

1. It's at a different **position**.
2. It's at a different **size**.
3. It has a different **orientation**.
4. It has a different **order**.

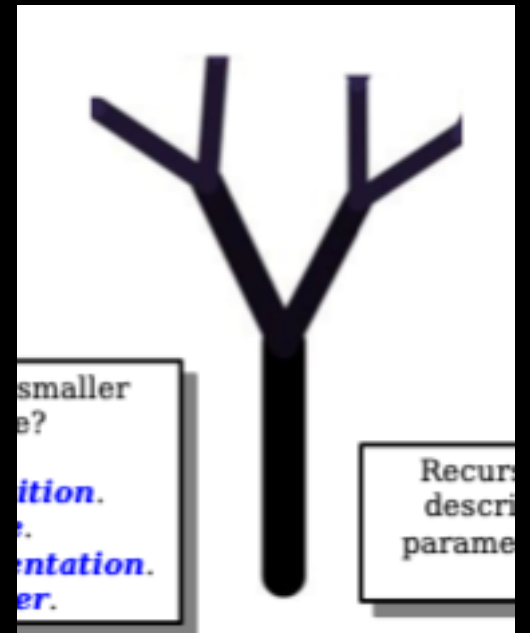
Recursive fractals are often described in terms of some parameter called the **order** of the fractal.

Lecture Activity



Give a sequence precise instructions in English (algorithm) to DRAW an order-3 fractal tree

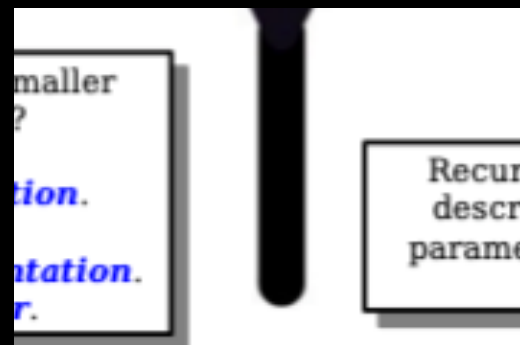
Lecture Activity



Give a sequence precise instructions in English (algorithm) to DRAW an order-3 fractal tree

Hint:

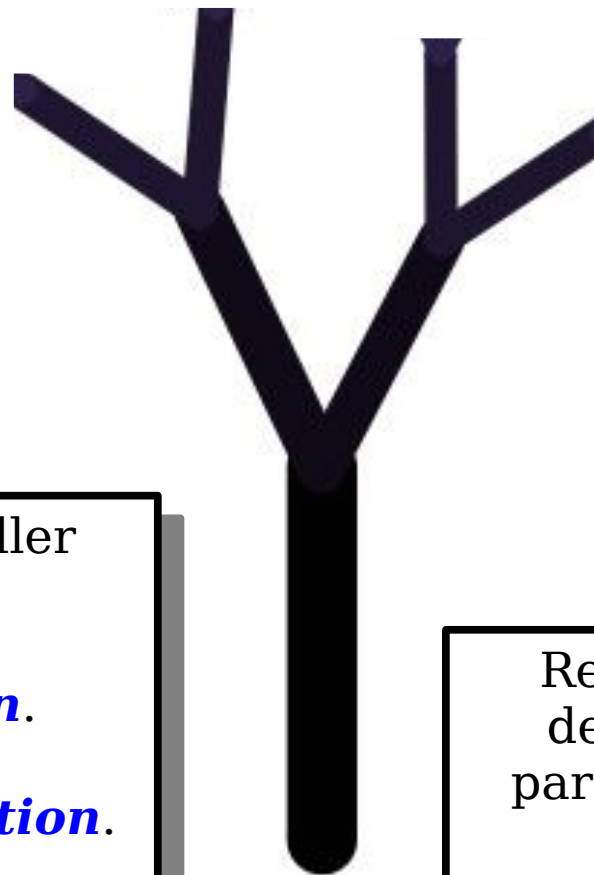
Draw a line and then



An order-3 tree.

An order-0 tree is nothing at all.

An order- n tree is a line with two smaller order- $(n-1)$ trees starting at the end of that line.



What differentiates the smaller tree from the bigger one?

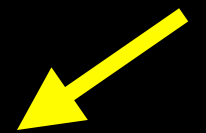
1. It's at a different **position**.
2. It's at a different **size**.
3. It has a different **orientation**.
4. It has a different **order**.

Recursive fractals are often described in terms of some parameter called the **order** of the fractal.

Lecture Activity

- draw a line
- tilt the canvas 45° left and draw an order-2 tree
- tilt the canvas 45° right and draw an order-2 tree

Recursive Call



Recursive Call



Lecture Activity

- draw a line

- tilt the canvas 45° left and draw an order-2 tree

- tilt the canvas 45° right and draw an order-2 tree

Lecture Activity

- draw a line
- tilt the canvas 45° left and draw an order-2 tree
 - draw a line
 - tilt the canvas 45° left and draw an order-1 tree
 - tilt the canvas 45° right and draw an order-1 tree
- tilt the canvas 45° right and draw an order-2 tree
 - draw a line
 - tilt the canvas 45° left and draw an order-1 tree
 - tilt the canvas 45° right and draw an order-1 tree

Lecture Activity

- draw a line
- tilt the canvas 45° left and draw an order-2 tree
 - draw a line
 - tilt the canvas 45° left and draw an order-1 tree
 - tilt the canvas 45° right and draw an order-1 tree
- tilt the canvas 45° right and draw an order-2 tree
 - draw a line
 - tilt the canvas 45° left and draw an order-1 tree
 - tilt the canvas 45° right and draw an order-1 tree

Lecture Activity

- draw a line

- tilt the canvas 45° left and draw an order-2 tree

- draw a line

- tilt the canvas 45° left and draw an order-1 tree

- draw a line

- tilt the canvas 45° left and draw an order-0 tree

- tilt the canvas 45° right and draw an order-0 tree

- tilt the canvas 45° right and draw an order-1 tree

- draw a line

- tilt the canvas 45° left and draw an order-0 tree

- tilt the canvas 45° right and draw an order-0 tree

- tilt the canvas 45° right and draw an order-2 tree

- draw a line

- tilt the canvas 45° left and draw an order-1 tree

- draw a line

- tilt the canvas 45° left and draw an order-0 tree

- tilt the canvas 45° right and draw an order-0 tree

- tilt the canvas 45° right and draw an order-1 tree

- draw a line

- tilt the canvas 45° left and draw an order-0 tree

- tilt the canvas 45° right and draw an order-0 tree

Lecture Activity

- draw a line
- tilt the canvas 45° left and draw an order-2 tree
 - draw a line
 - tilt the canvas 45° left and draw an order-1 tree
 - draw a line
 - tilt the canvas 45° left and draw an order-0 tree
 - tilt the canvas 45° right and draw an order-0 tree
 - tilt the canvas 45° right and draw an order-1 tree
 - draw a line
 - tilt the canvas 45° left and draw an order-0 tree
 - tilt the canvas 45° right and draw an order-0 tree
- tilt the canvas 45° right and draw an order-2 tree
 - draw a line
 - tilt the canvas 45° left and draw an order-1 tree
 - draw a line
 - tilt the canvas 45° left and draw an order-0 tree
 - tilt the canvas 45° right and draw an order-0 tree
 - tilt the canvas 45° right and draw an order-1 tree
 - draw a line
 - tilt the canvas 45° left and draw an order-0 tree
 - tilt the canvas 45° right and draw an order-0 tree

Nothing to draw at order 0

We stop!

BASE CASE

In general for n

- draw a line
- tilt the canvas 45° left and draw and order-(n-1) tree
- tilt the canvas 45° right and draw and order-(n-1) tree

Check This Out!!!

<http://recursivedrawing.com/>

Different Flavors of Recursion

Reverse String: write first character, reverse the remaining **single smaller string**

Dictionary: **either** inspect upper-half **or** lower-half

Fractal Tree: draw **both** the left order-($n-1$) and right order-($n-1$) trees

All solve a problem by breaking it up into one or more **smaller "similar" problems**

Recursive Problem-Solving

```
if(problem is sufficiently simple){  
    directly solve the problem  
    i.e. do something and/or return the solution  
}  
else{  
    split problem up into one or more smaller  
    problems with the same structure as the original  
  
    solve some or all of those smaller problems  
  
    do something or combine results to return  
    solution if necessary  
}
```

Recursive Problem-Solving

```
if(problem is sufficiently simple){
```

BASE CASE

```
    directly solve the problem
```

```
    i.e. do something and/or return the solution
```

```
} else{
```

```
    split problem up into one or more smaller  
    problems with the same structure as the original
```

```
    solve some or all of those smaller problems
```

```
    do something or combine results to return  
    solution if necessary
```

```
}
```

Why Recursion

An alternative to iteration

Not always practical (some compilers optimize tail-recursive algorithms)

Elegant and intuitive solution for some problems

Factorial

$$1 \times 2 \times 3 \times \dots \times n$$

$$n! = \prod_{k=1}^n k$$

For example:

$$0! = 1, 1! = 1, 2! = 2, 3! = 6, 4! = 24, 5! = 120$$

The empty product

But what if we start from n ?

$n! =$

But what if we start from n?

$$n! = n \times (n-1) \times (n-2) \times (n-3) \times \dots \dots \dots \dots 2 \times 1$$

What is this?

But what if we start from n?

$$n! = n \times \underbrace{(n-1) \times (n-2) \times (n-3) \times \dots \times 2 \times 1}_{(n-1)!}$$

But what if we start from n?

$$n! = n \times \underbrace{(n-1) \times (n-2) \times (n-3) \times \dots \dots \dots 2 \times 1}_{(n-1)!}$$

$$(n-1)! = (n-1) \times \underbrace{(n-2) \times (n-3) \times \dots \dots \dots 2 \times 1}$$

What is this?

But what if we start from n?

$$n! = n \times \underbrace{(n-1) \times (n-2) \times (n-3) \times \dots \dots \dots 2 \times 1}_{(n-1)!}$$

$$(n-1)! = (n-1) \times \underbrace{(n-2) \times (n-3) \times \dots \dots \dots 2 \times 1}_{(n-2)!}$$

Recursion that Returns a Value

$$n! = n \times (n-1)!$$
The diagram shows the mathematical formula for factorial, n! = n x (n-1)!. Two yellow arrows point from the text below to the exclamation marks in the formula. One arrow points to the 'n!' on the left, and the other points to the '(n-1)!' on the right, illustrating that the function calls itself with a smaller argument.

Same function being called within solution

Recursion that Returns a Value


$$n! = n \times (n-1)!$$

```
/** Computes the factorial of the nonnegative integer n.
  @pre:  n must be greater than or equal to 0.
  @post:  None.
  @return: The factorial of n; n is unchanged. */
int factorial(int n)
{
    if (n == 0)
        return 1;
    else // n > 0, so n-1 >= 0. Thus, fact(n-1) returns (n-1)!
        return n * factorial(n - 1); // n * (n-1)! is n!
} // end fact
```

Recursion that Returns a Value

$$n! = n \times (n-1)!$$

```
/** Computes the factorial of the nonnegative integer n.
  @pre:  n must be greater than or equal to 0.
  @post: None.
  @return: The factorial of n; n is unchanged. */
int factorial(int n)
{
    if (n == 0)
        return 1;
    else // n > 0, so n-1 >= 0. Thus, fact(n-1) returns (n-1)!
        return n * factorial(n - 1); // n * (n-1)! is n!
} // end fact
```

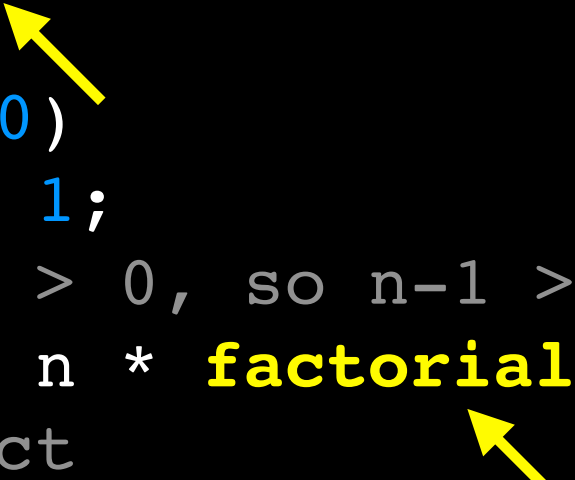


BASE CASE

Recursion that Returns a Value

$$n! = n \times (n-1)!$$

```
/** Computes the factorial of the nonnegative integer n.
  @pre:  n must be greater than or equal to 0.
  @post: None.
  @return: The factorial of n; n is unchanged. */
int factorial(int n)
{
    if (n == 0)
        return 1;
    else // n > 0, so n-1 >= 0. Thus, fact(n-1) returns (n-1)!
        return n * factorial(n - 1); // n * (n-1)! is n!
} // end fact
```



Recursion that Returns a Value

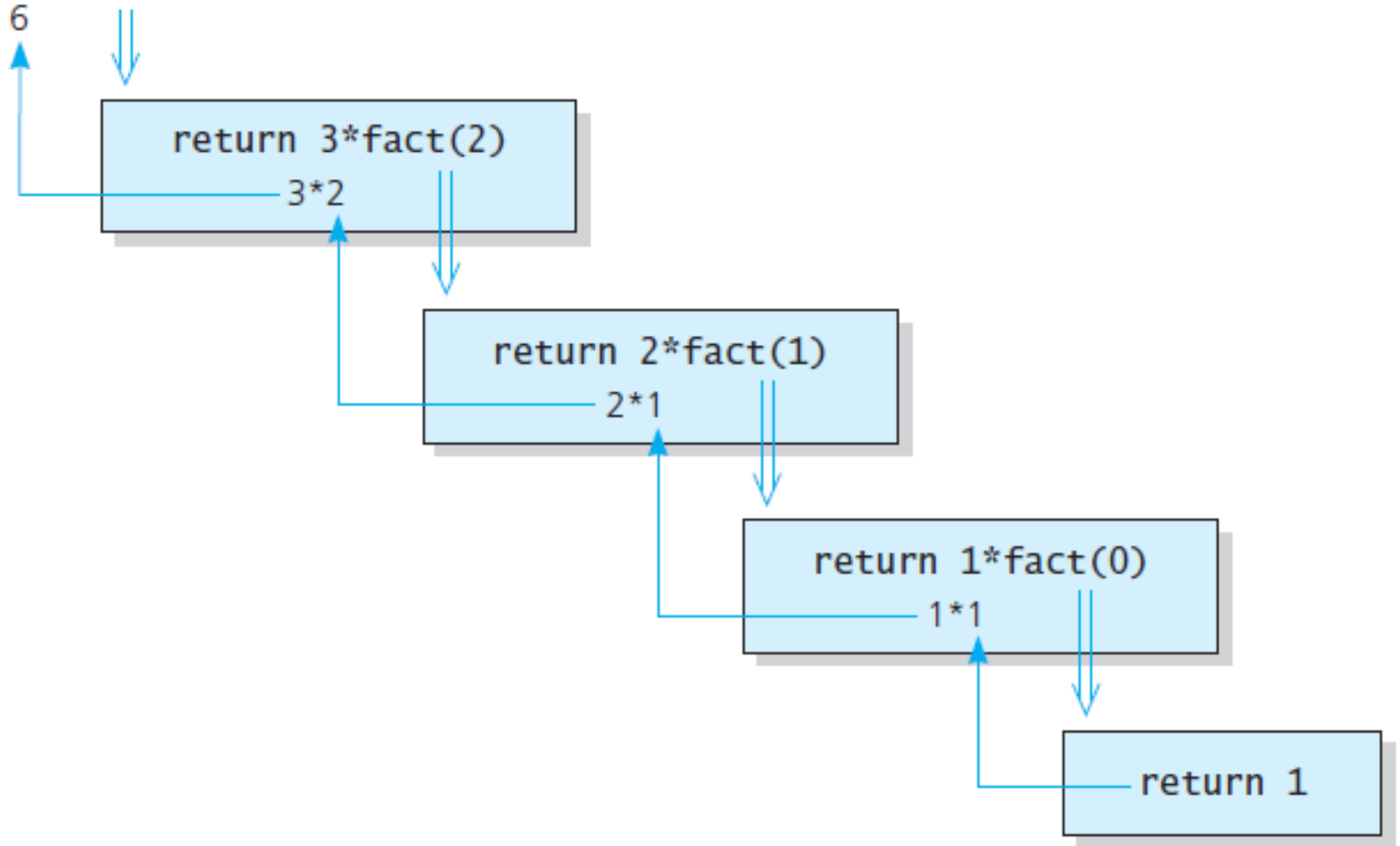
$$n! = n \times (n-1)!$$

```
/** Computes the factorial of the nonnegative integer n.
  @pre:  n must be greater than or equal to 0.
  @post: None.
  @return: The factorial of n; n is unchanged. */
int factorial(int n)
{
    if (n == 0)
        return 1;
    else // n > 0, so n-1 >= 0. Thus, fact(n-1) returns (n-1)!
        return n * factorial(n - 1); // n * (n-1)! is n!
} // end fact
```

BASE CASE

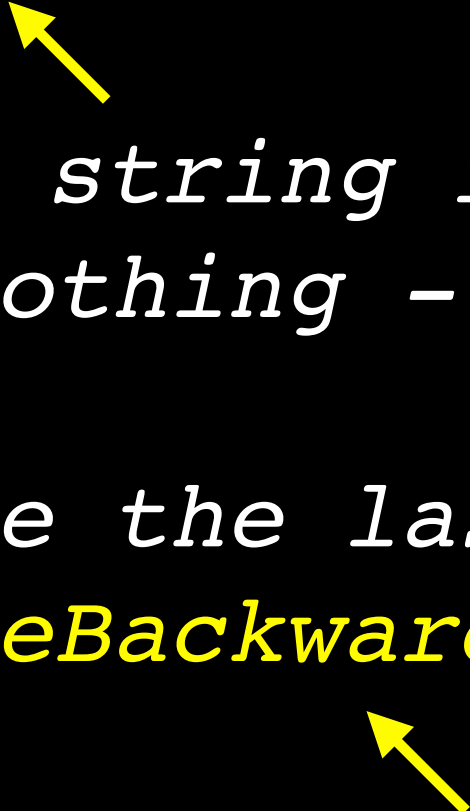
WILL LEAD TO
BASE CASE


```
cout << fact(3);
```



Writing a String Backwards

```
writeBackward(string s)
{
    if(the string is empty)
        Do nothing - this is the base case
    else
        Write the last character of s
        writeBackward(s minus the last char)
}
```

A diagram illustrating the recursive process. A yellow arrow points from the opening curly brace of the function to the 'base case' comment. Another yellow arrow points from the recursive call 'writeBackward(s minus the last char)' back to the opening curly brace, indicating the function calling itself.

Recursion that Performs an Action

```
/** Prints a string backward.  
  @post:  The string s is printed backwards  
  @param: s  The string to write backwards */  
  
void writeBackward(std::string s)  
{  
    size_t length = s.size(); // Length of string  
    if (length > 0) //implicit base case: if length ==0 do nothing  
    {  
        // Print the last character  
        std::cout << s.substr(length - 1, 1);  
  
        // Print the rest of the string backwards - recursive call  
        writeBackward(s.substr(0, length - 1));  
    } // end if  
  
    // length == 0 is the base case - do nothing  
} // end writeBackward
```

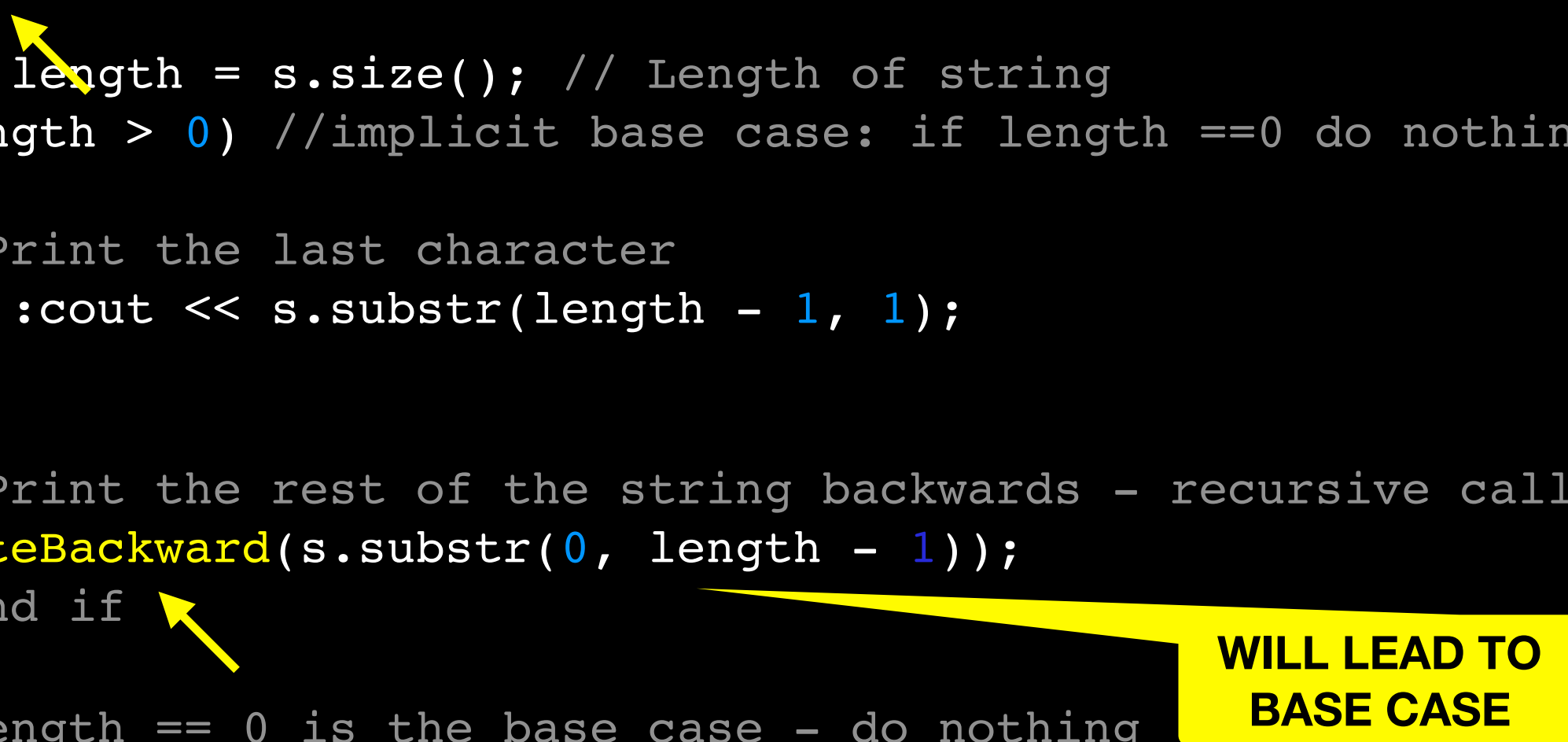
Recursion that Performs an Action

```
/** Prints a string backward.
  @post:  The string s is printed backwards
  @param: s  The string to write backwards */

void writeBackward(std::string s)
{
    size_t length = s.size(); // Length of string
    if (length > 0) //implicit base case: if length ==0 do nothing
    {
        // Print the last character
        std::cout << s.substr(length - 1, 1);

        // Print the rest of the string backwards - recursive call
        writeBackward(s.substr(0, length - 1));
    } // end if

    // length == 0 is the base case - do nothing
} // end writeBackward
```



The diagram illustrates the recursive process of the `writeBackward` function. A yellow arrow points from the `length` variable in the `if` condition to the `length` parameter in the recursive call `writeBackward(s.substr(0, length - 1))`. Another yellow arrow points from the `if` condition to the `length == 0` comment, which is highlighted by a yellow box labeled "WILL LEAD TO BASE CASE".

Write String Backwards

Hello

o

Hell

o l

Hel

o l l

He

o l l e

H

o l l e H

BASE CASE