Luiss
Libera Università Internazionale degli Studi Sociali Guido Carli

Algorithms A.Y. 2022/2023

Lab – Mystery function

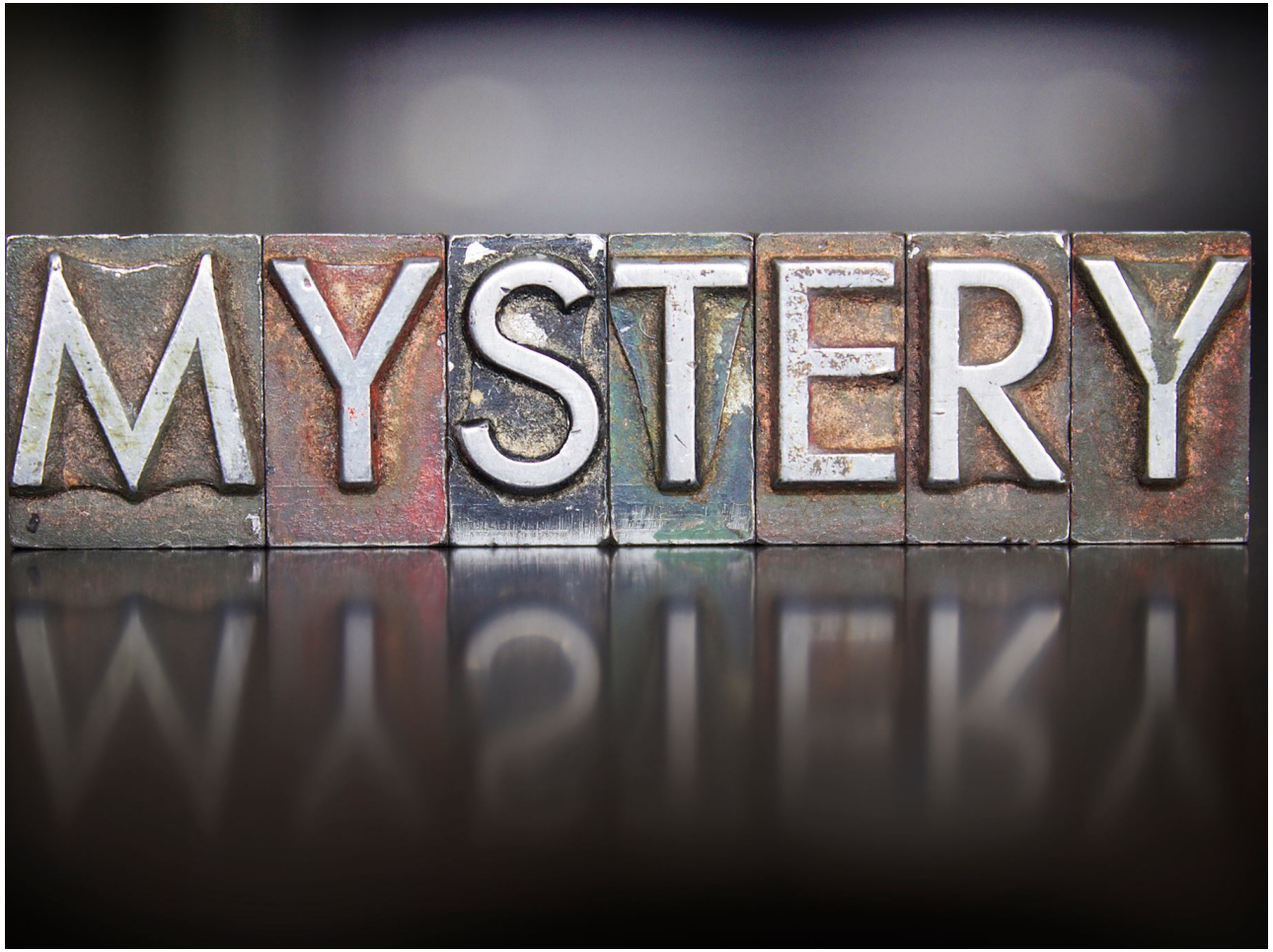
Irene Finocchi, Flavio Giorgi, Bardh Prenkaj
finocchi@luiss.it, fgiorgi@luiss.it, bprenkaj@luiss.it

14 February 2023

Lab Lecture 3 - Mystery function

Lab lecture 3 overview:

- We implement three different algorithms to compute a mysterious number
- We compare their performance (time and memory)
- Q/A project

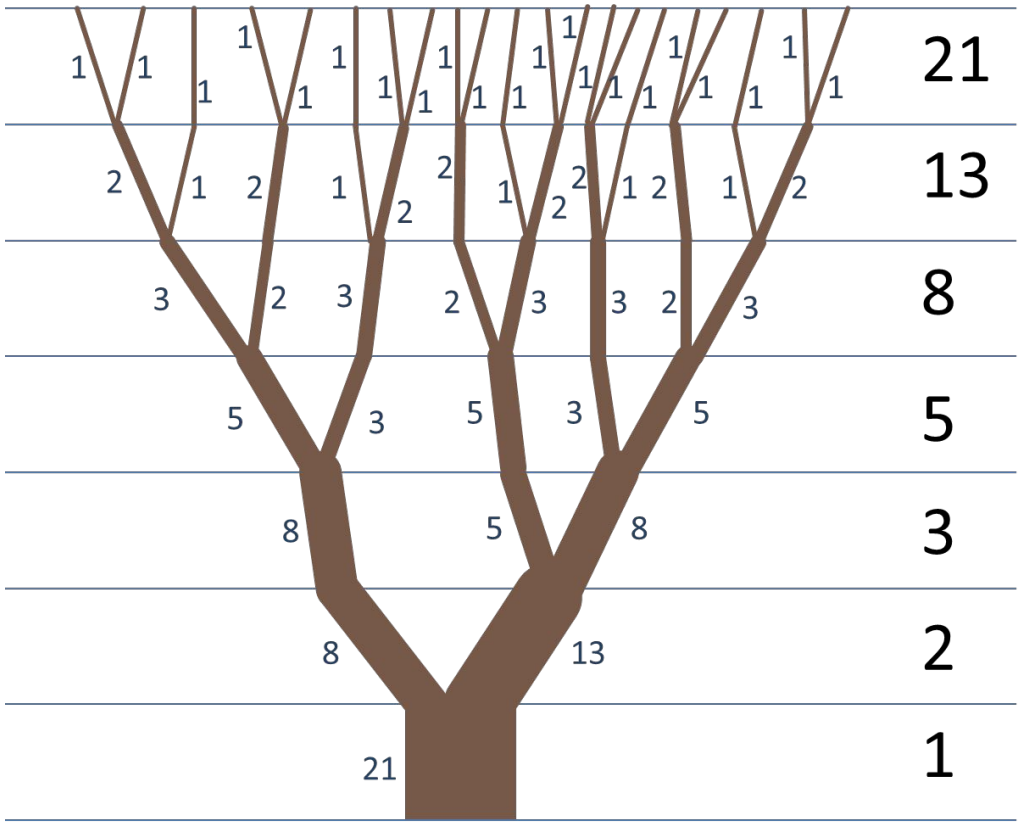


Lab Lecture 3 - Mystery function

What is this mysterious function that we want to implement?

Given a number x, we want to compute 2^x by relying on two recursive calls just as we've seen with Fibonacci numbers.

- What is the base case?
- How do the recursive calls look like?



LUISS

Lab Lecture 3 - Mystery function

Lab Lecture 3 - Mystery function

```
def mystery(x):
    if x = 1 then return 2
    return mystery(x-1) + mystery(x-1)
```

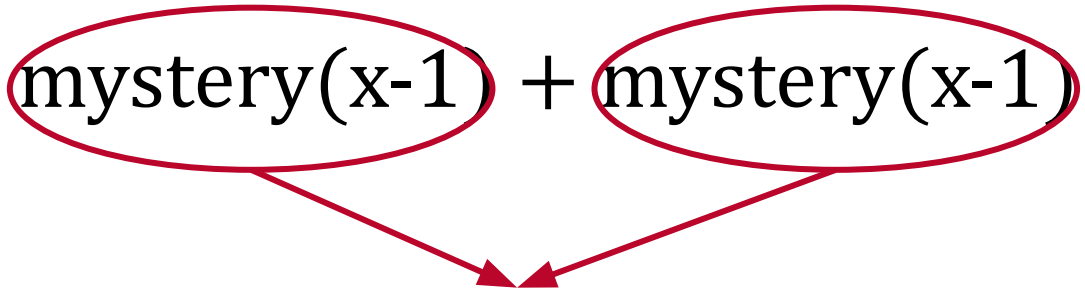
Lab Lecture 3 - Mystery function

```
def mystery(x):
    if x = 1 then return 2
    return mystery(x-1) + mystery(x-1)
```

because $2^0 = 1$, we can stop one recursive level before

Lab Lecture 3 - Mystery function

```
def mystery(x):
    if x = 1 then return 2
    return mystery(x-1) + mystery(x-1)
```



Lab Lecture 3 - Mystery function

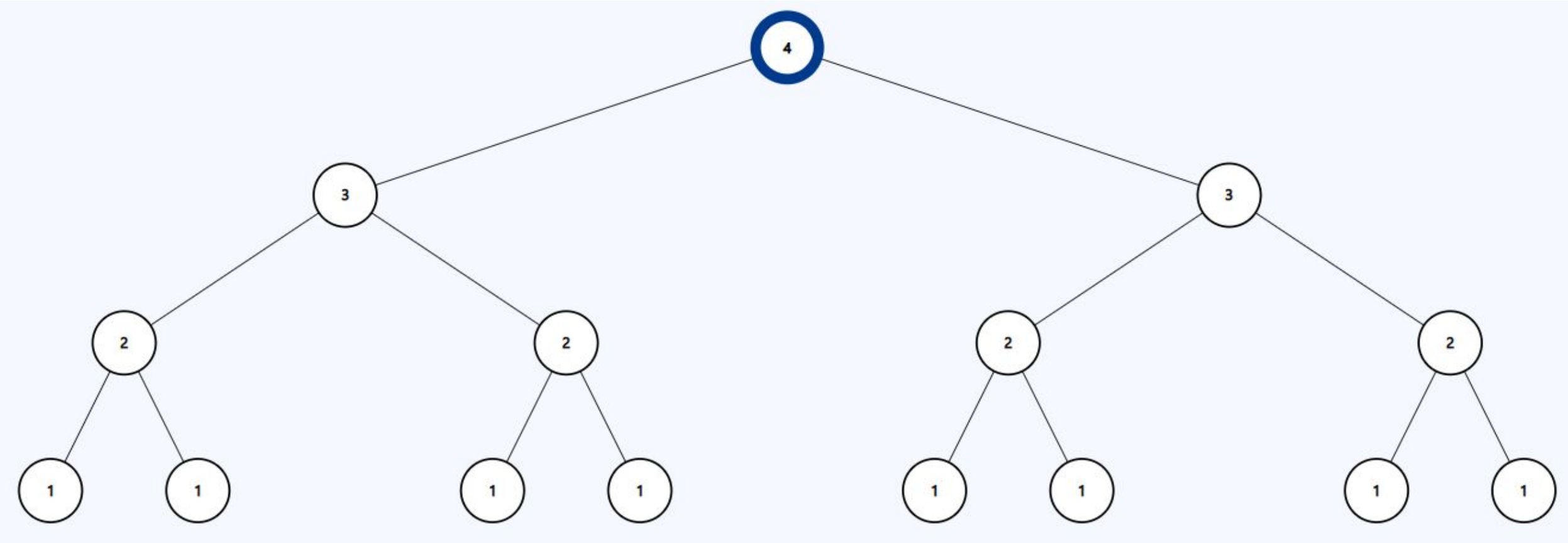
Draw the call stack for x = 4

Given a number x, we want to compute 2^4 according to the pseudocode in the previous slide.

- How many leaf nodes are there?
- How many internal nodes are there?
- What's the relationship between the number of internal and leaf nodes?

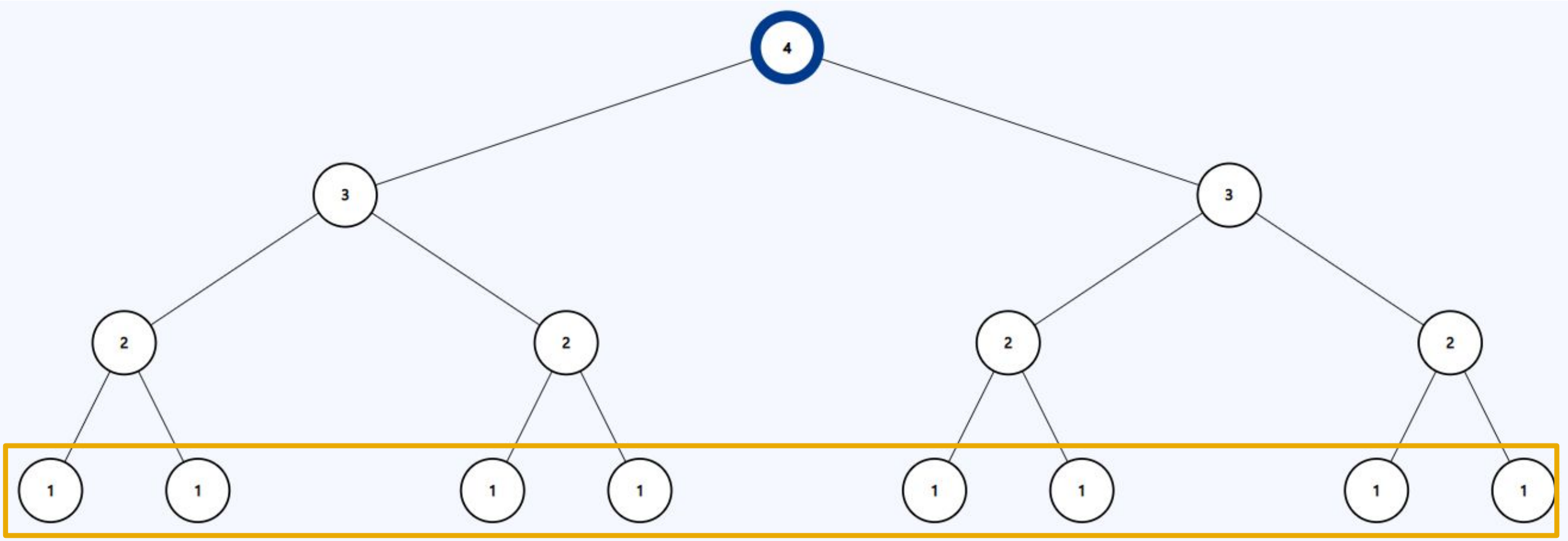
Lab Lecture 3 - Mystery function

Mystery function – “call stack”



https://tree-visualizer.netlify.app/

Mystery function – “call stack”

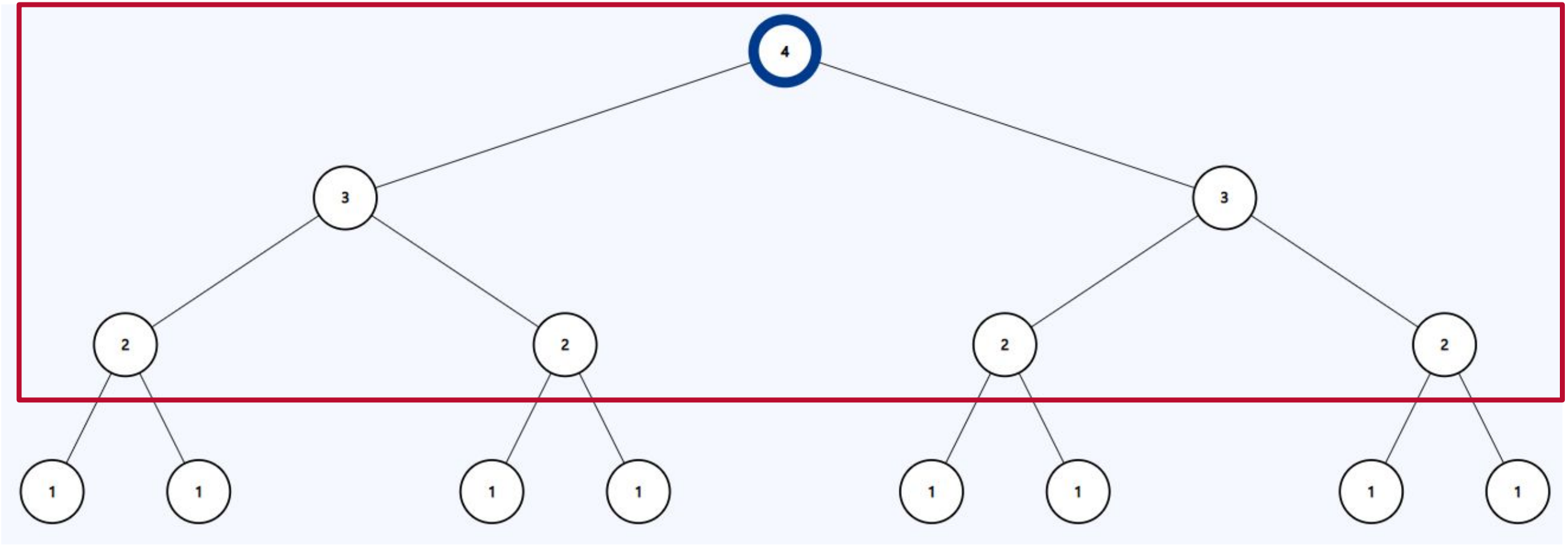


2^3 = 8 leaves

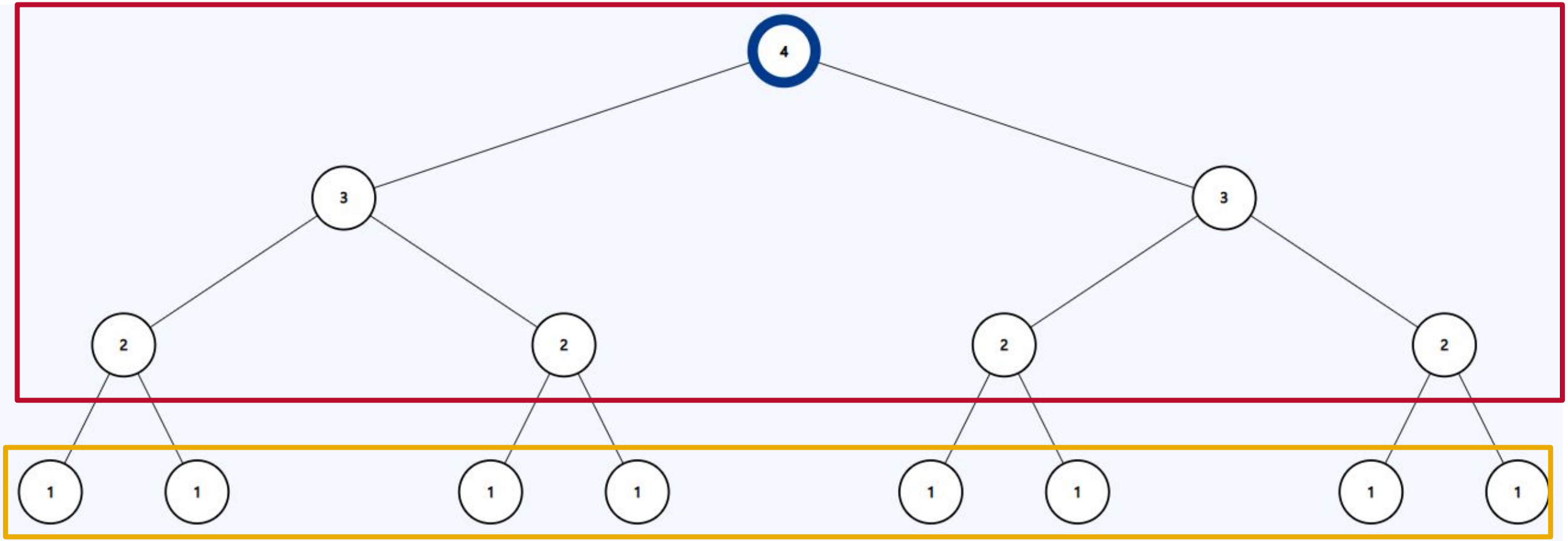
LUISS

Mystery function – “call stack”

2^3-1 = 7 internal nodes



Mystery function – “call stack”



leaves = # internal nodes + 1

Mystery function – “call stack”

What’s the relationship between leaves and internal nodes?

Given a number x, we have 2^{x-1} leaves and 2^{x-1} - 1 internal nodes (including the root node)

We’ll see a cool demonstration for this in the next laboratory class

For now, “convince” yourselves visually that the above relation holds.

Lab Lecture 3 - Think about how to improve mystery

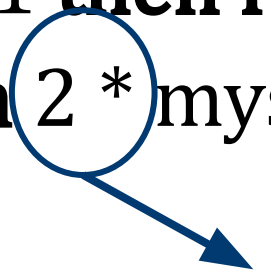
Lab Lecture 3 - Improved mystery function

```
def mystery(x):
    if x = 1 then return 2
    return mystery(x-1) + mystery(x-1)
```

why should we compute
the same thing twice?

Lab Lecture 3 - Mystery function

```
def mystery(x):
    if x = 1 then return 2
    return 2 * mystery(x-1)
```



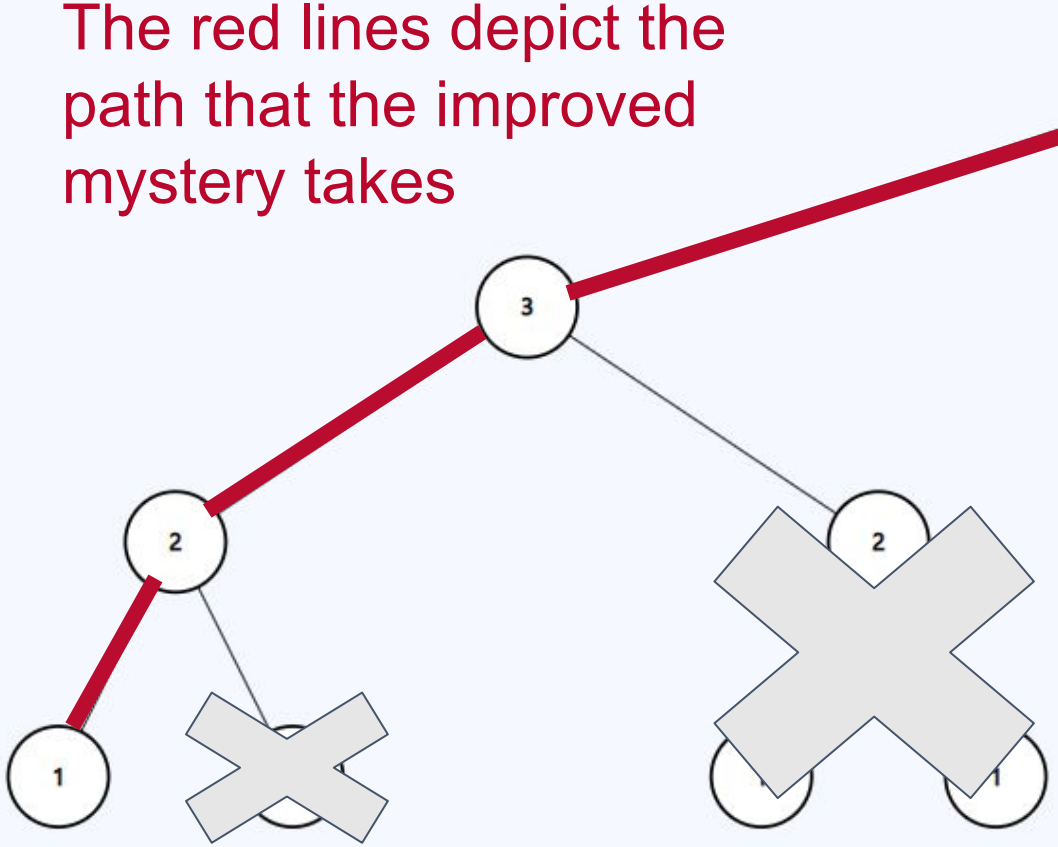
just multiply it by two

What's the running time of this function now?

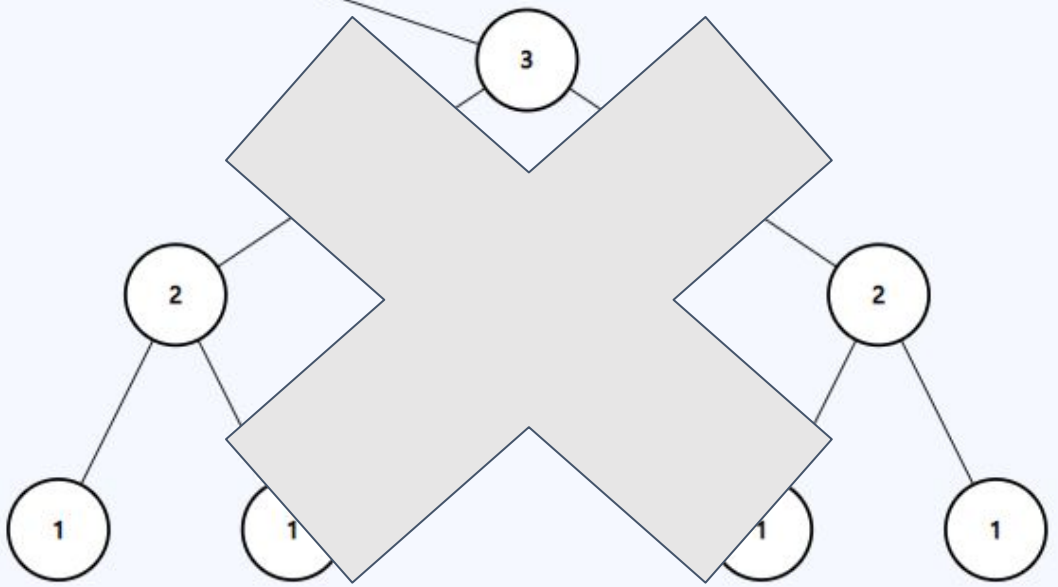
Is it faster than the non-improved version?

Mystery function with one recursive call

The red lines depict the path that the improved mystery takes



The rest of the call stack is ignored



Mystery function with one recursive call

We have a function that we can easily translate into a for loop