Tree Recursion

Announcements

Tree Recursion

Tree Recursion

Tree-shaped processes arise whenever executing the body of a recursive function makes more than one recursive call

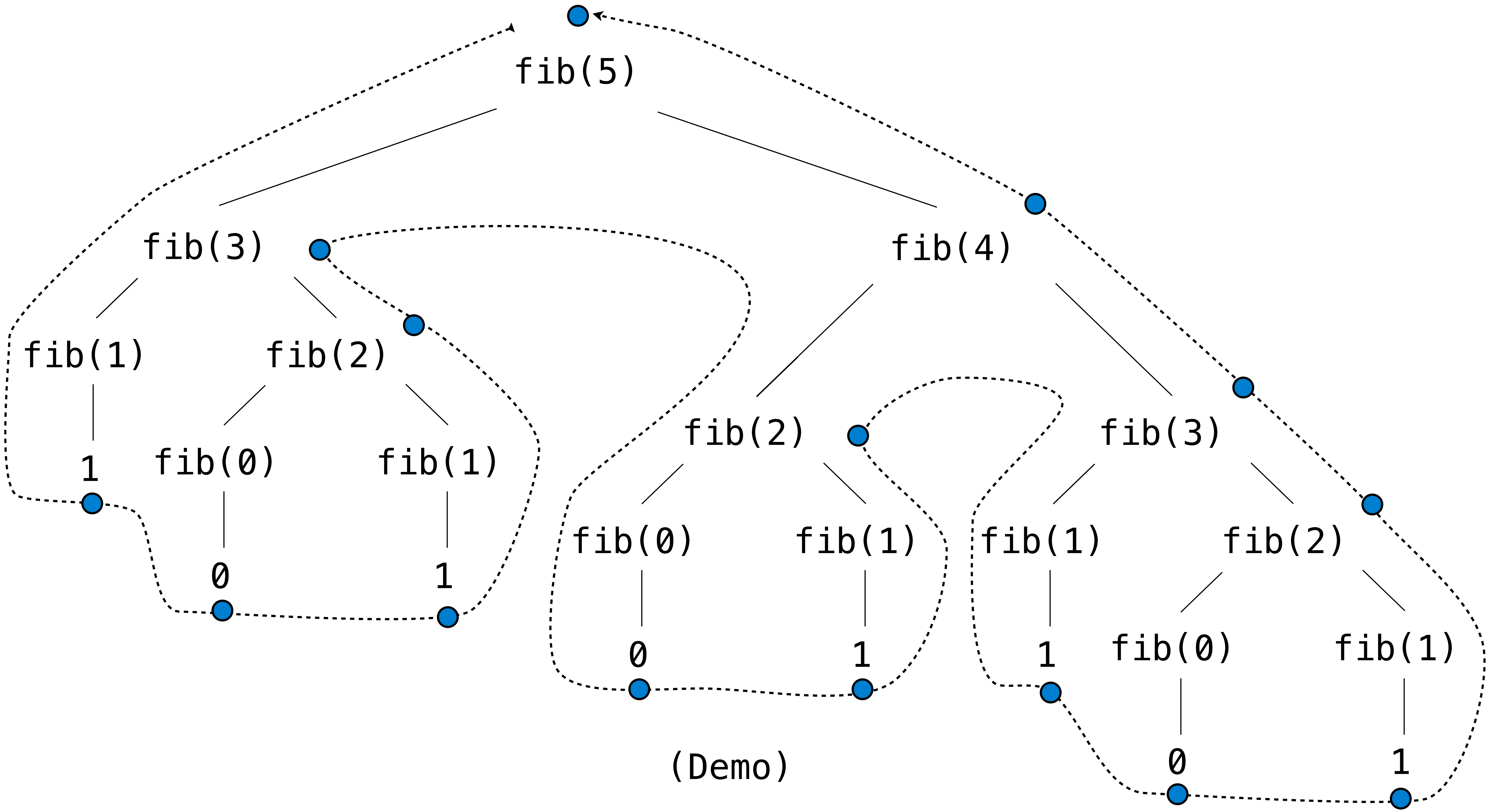
n:	0,	1,	2,	3,	4,	5,	6,	7,	8,	... ,	35
fib(n):	0,	1,	1,	2,	3,	5,	8,	13,	21,	... ,	9,227,465

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

A Tree-Recursive Process

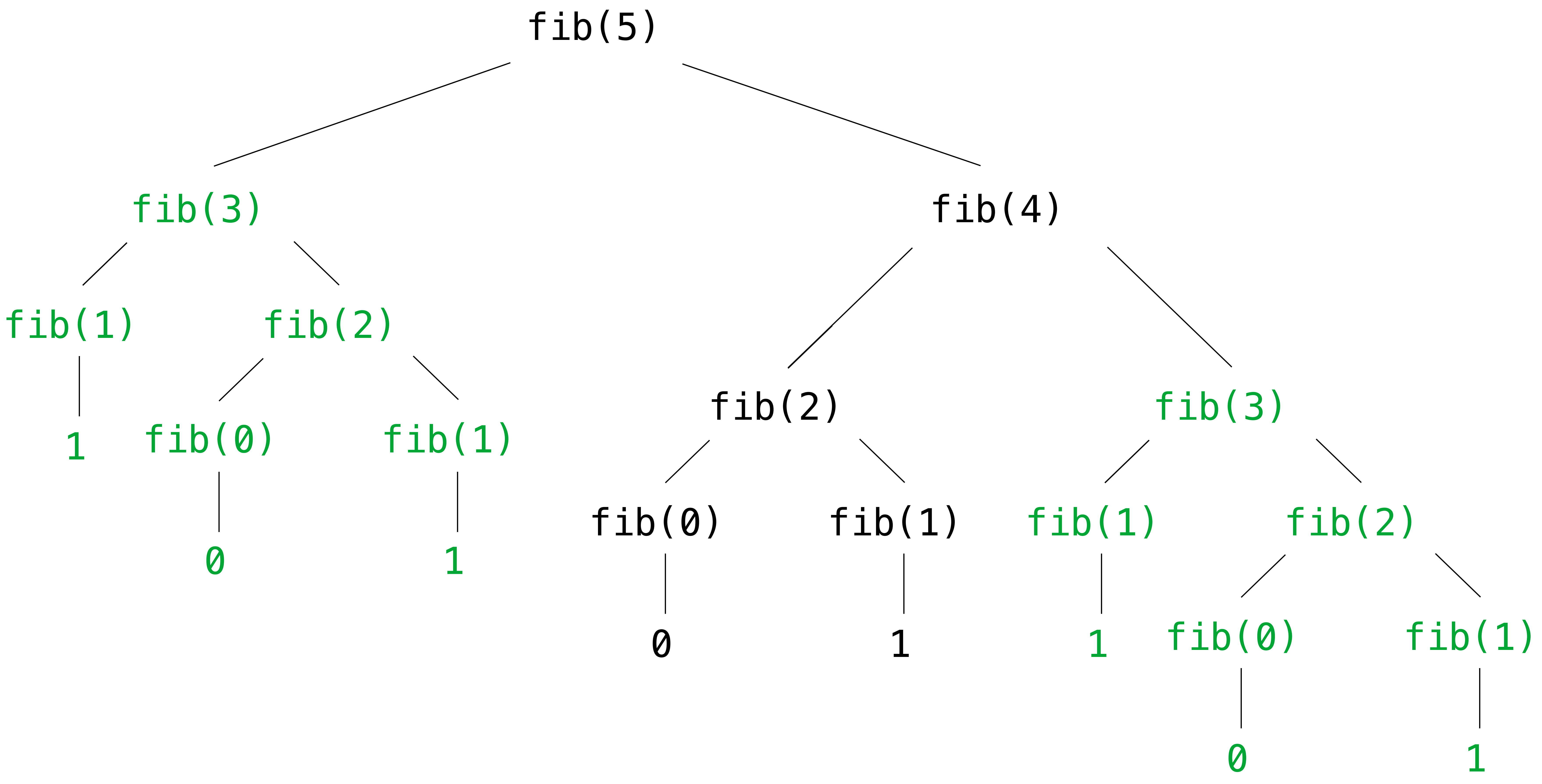
The computational process of fib evolves into a tree structure



(Demo)

Repetition in Tree-Recursive Computation

This process is highly repetitive; fib is called on the same argument multiple times

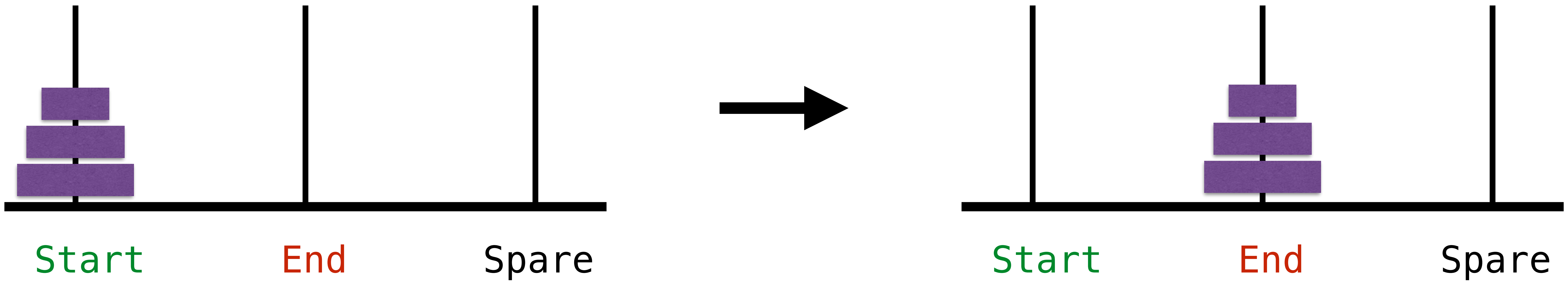


(We will speed up this computation dramatically in a couple weeks by remembering results)

Example: Towers of Hanoi

Towers of Hanoi

A puzzle asking that we move a stack of n **discs** from a **start peg** to a **end peg,** given a third **spare peg.**

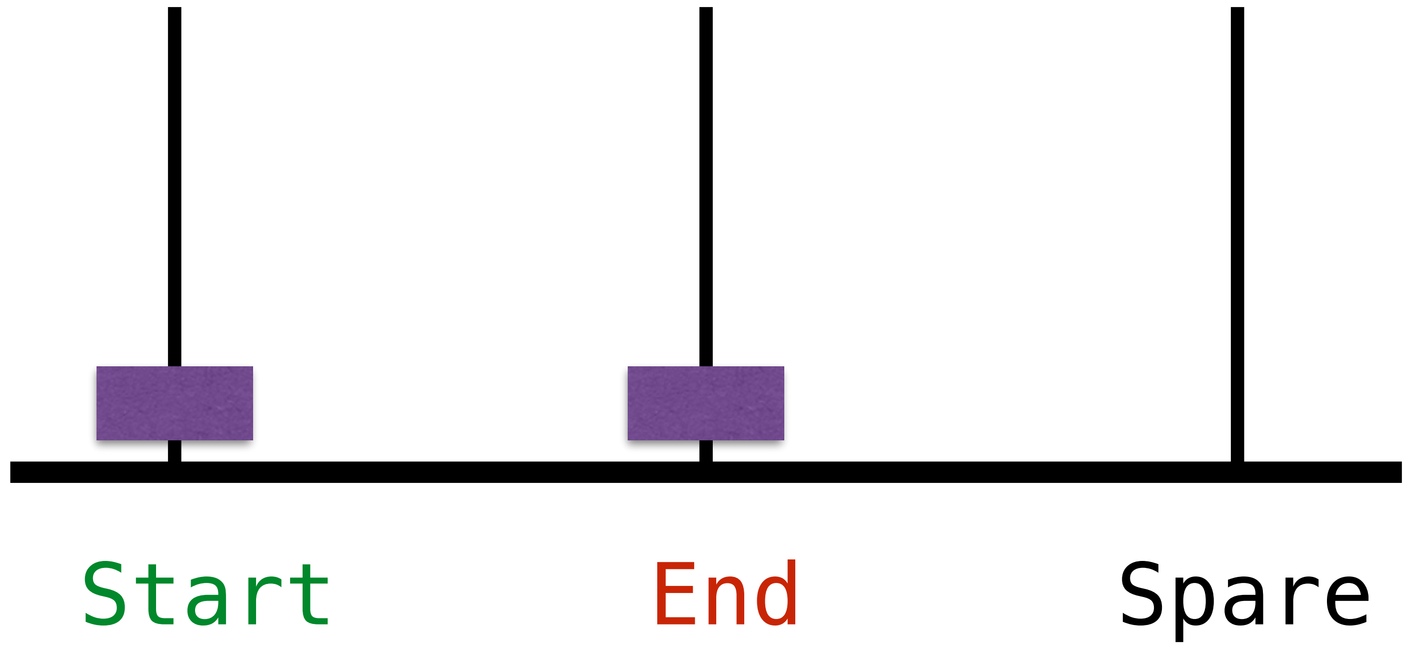


Demo: https://www.mathsisfun.com/games/towerofhanoi.html

Towers of Hanoi Strategy

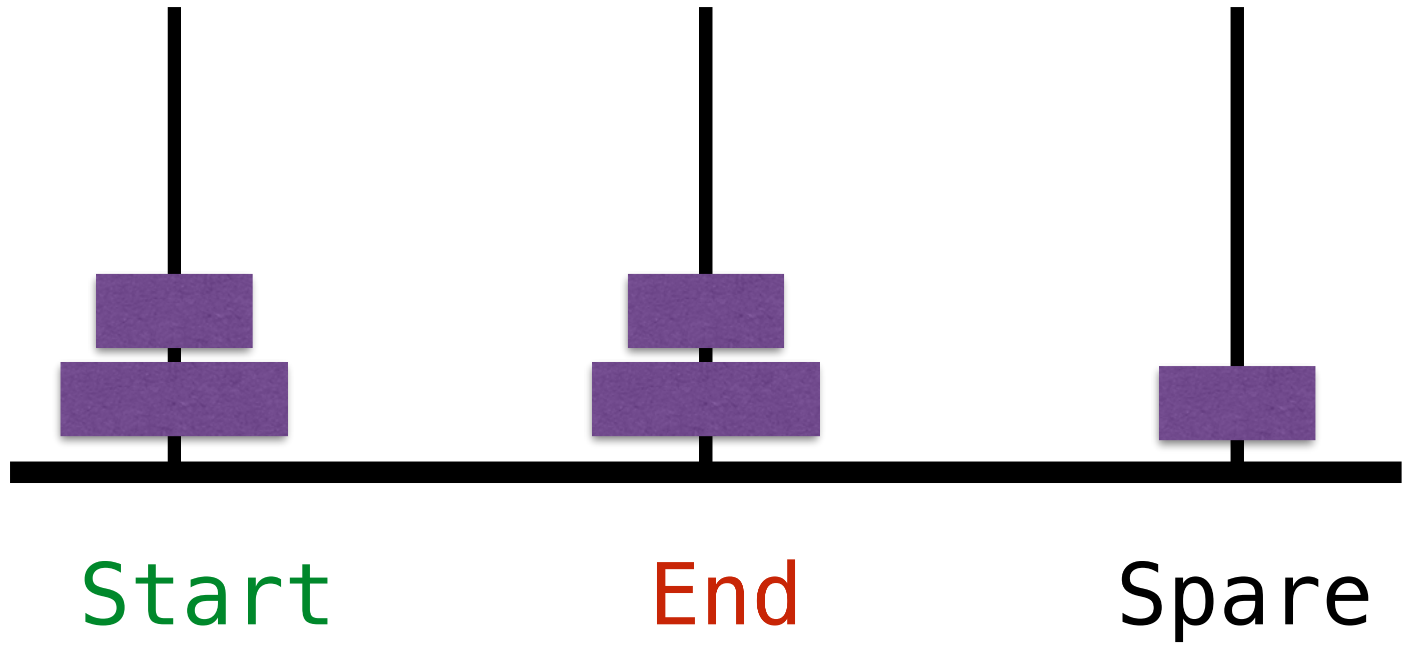
n = 1:

Move disc from Peg 1 to Peg 2



n = 2:

Move disc from Peg 1 to Peg 3
Move disc from Peg 1 to Peg 2
Move disc from Peg 3 to Peg 2



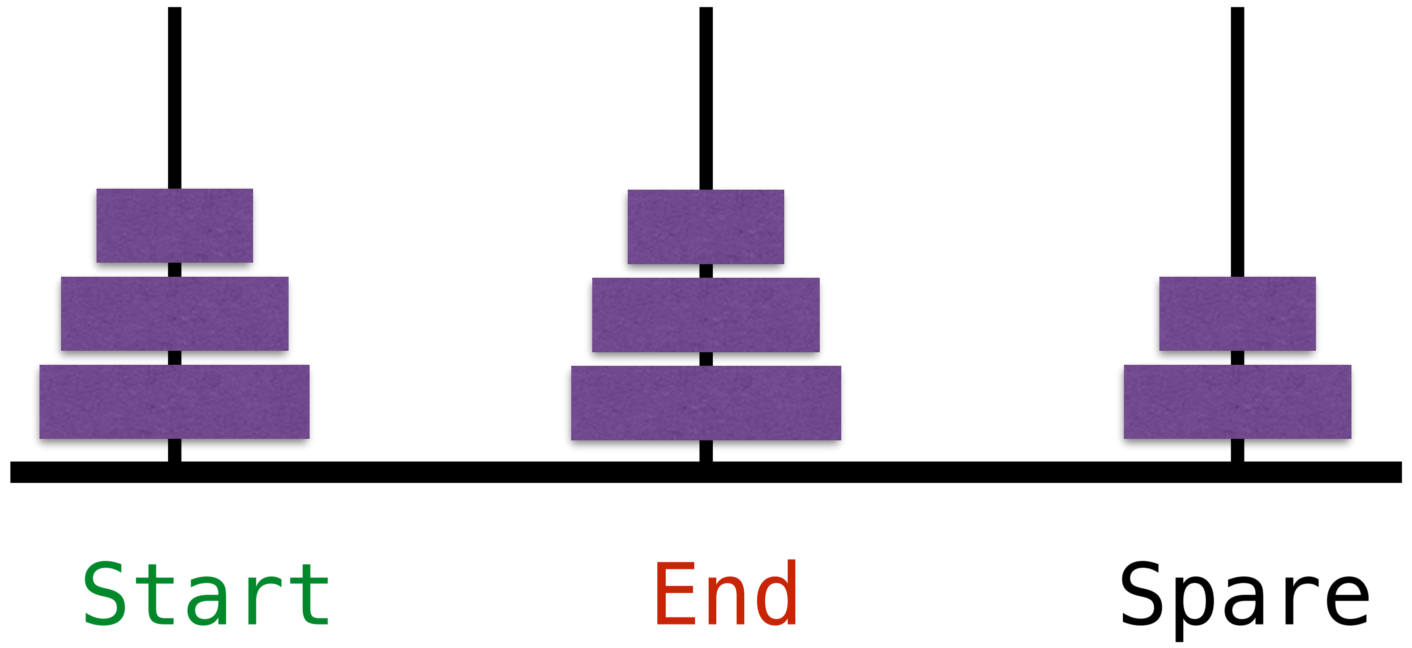
In general, we now know how to move 2 discs from any peg to any other peg.

Towers of Hanoi General Strategy

Let’s remember that we know how to move 2 discs from any peg to any other peg.

n = 3:

Move top 2 discs from Peg 1 to Peg 3
Move bottom disc from Peg 1 to Peg 2
Move top 2 discs from Peg 3 to Peg 2



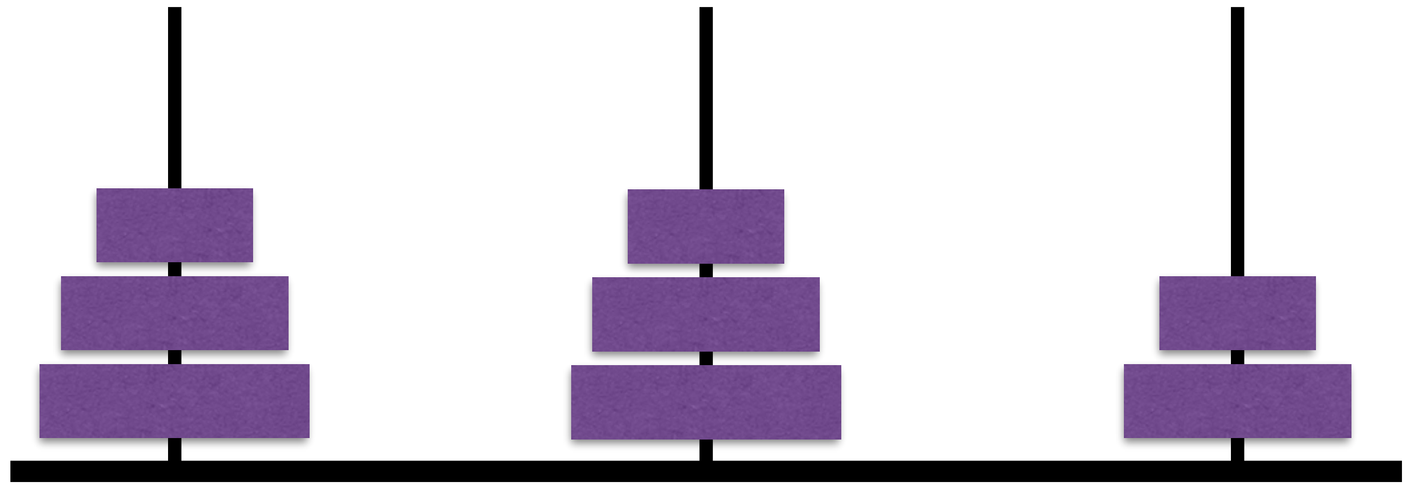
Can we generalize this to any n discs?

Towers of Hanoi General Strategy

Let's remember that we know how to move 2 discs from any peg to any other peg.

n = 3:

Move top 2 discs from Peg 1 to Peg 3
Move bottom disc from Peg 1 to Peg 2
Move top 2 discs from Peg 3 to Peg 2



Can we generalize this to any n discs?

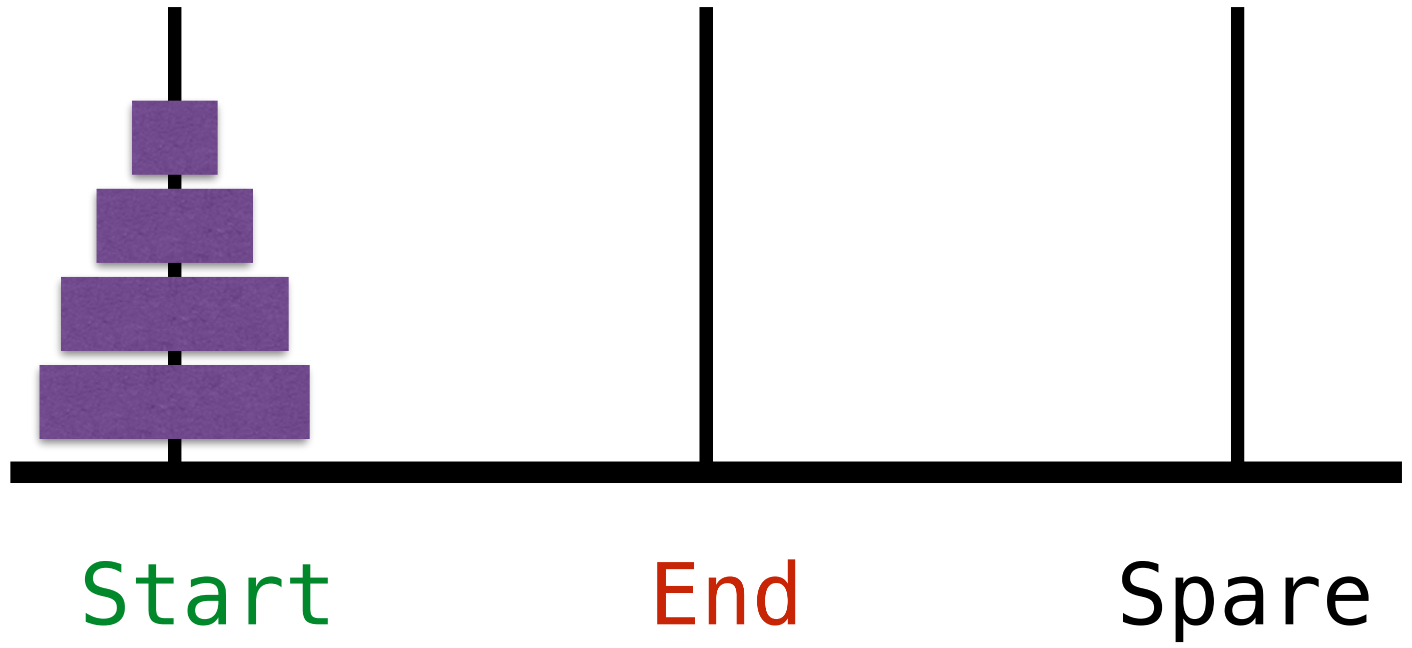
Yes!

Towers of Hanoi General Strategy

Let’s remember that we know how to move 2 discs from any peg to any other peg.

n = ?:

Move top n-1 discs from Peg 1 to Peg 3
Move bottom disc from Peg 1 to Peg 2
Move top n-1 discs from Peg 3 to Peg 2



Can we generalize this to any n discs?

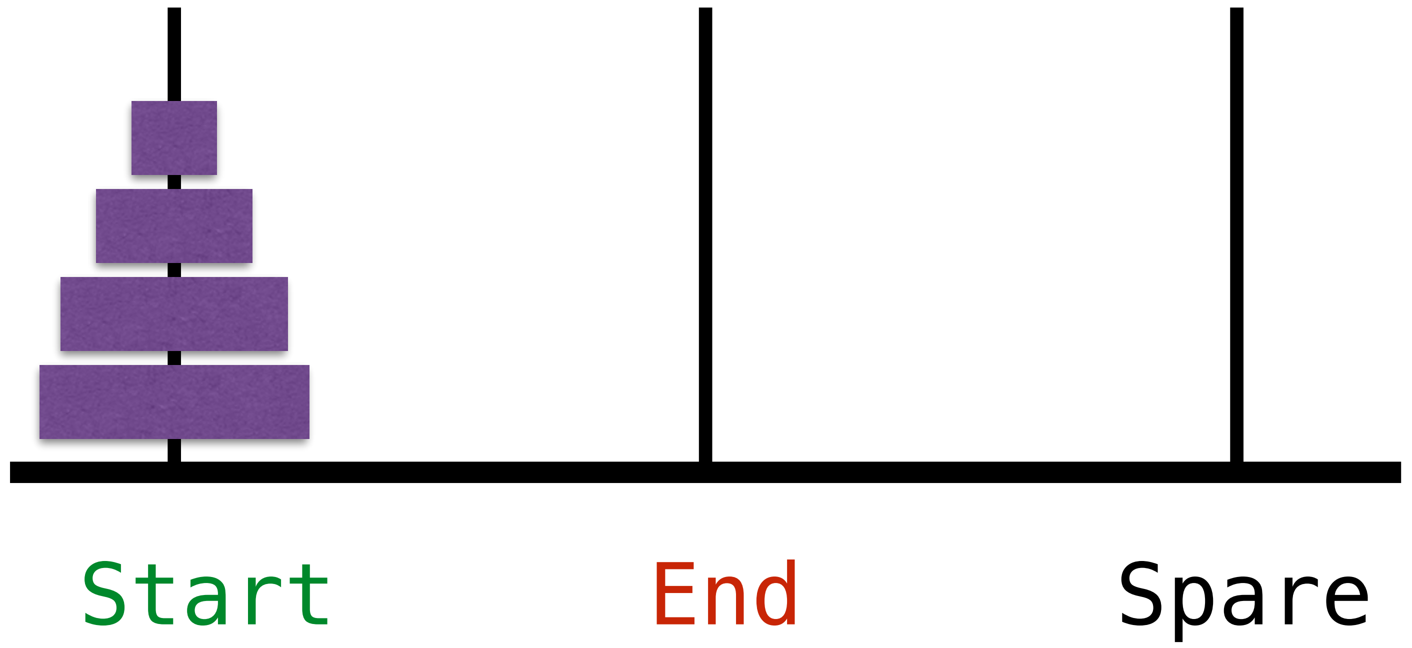
Yes!

Solving Towers of Hanoi

```
def print_move(origin, destination):
    print("Move the top disk from rod", origin, "to rod", destination)

def move_stack(n, start, end):
    if n == 1:
        print_move(start, end)
    else:
        spare_peg = 6 - start - end
        move_stack(n-1, start, spare_peg)
        print_move(start, end)
        move_stack(n-1, spare_peg, end)
```

(Demo)



Example: Counting Partitions

Counting Partitions

The number of partitions of a positive integer n, using parts up to size m, is the number of ways in which n can be expressed as the sum of positive integer parts up to m in increasing order.

count_partitions(6, 4)

2 + 4 = 6

1 + 1 + 4 = 6

3 + 3 = 6

1 + 2 + 3 = 6

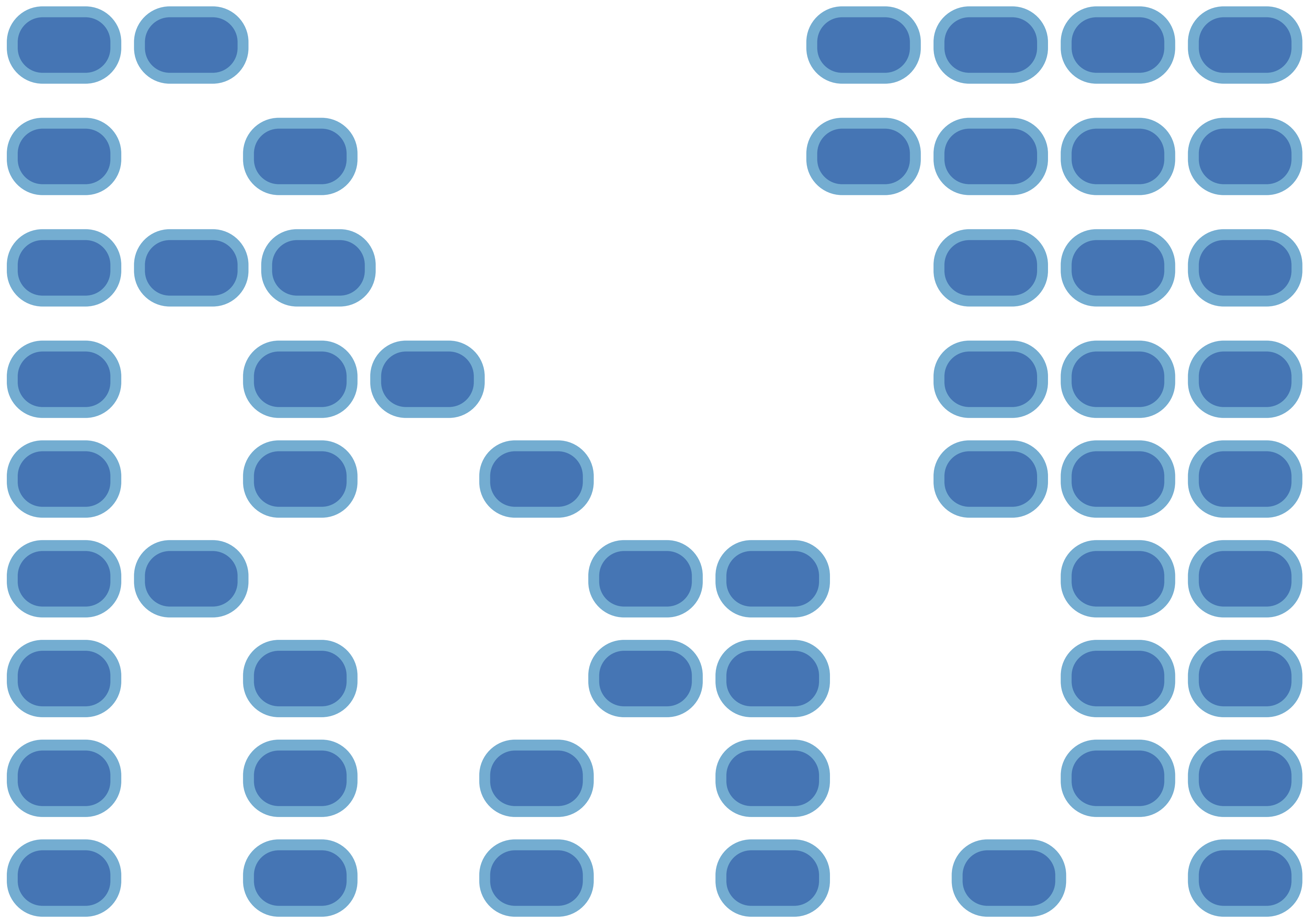
1 + 1 + 1 + 3 = 6

2 + 2 + 2 = 6

1 + 1 + 2 + 2 = 6

1 + 1 + 1 + 1 + 2 = 6

1 + 1 + 1 + 1 + 1 + 1 = 6

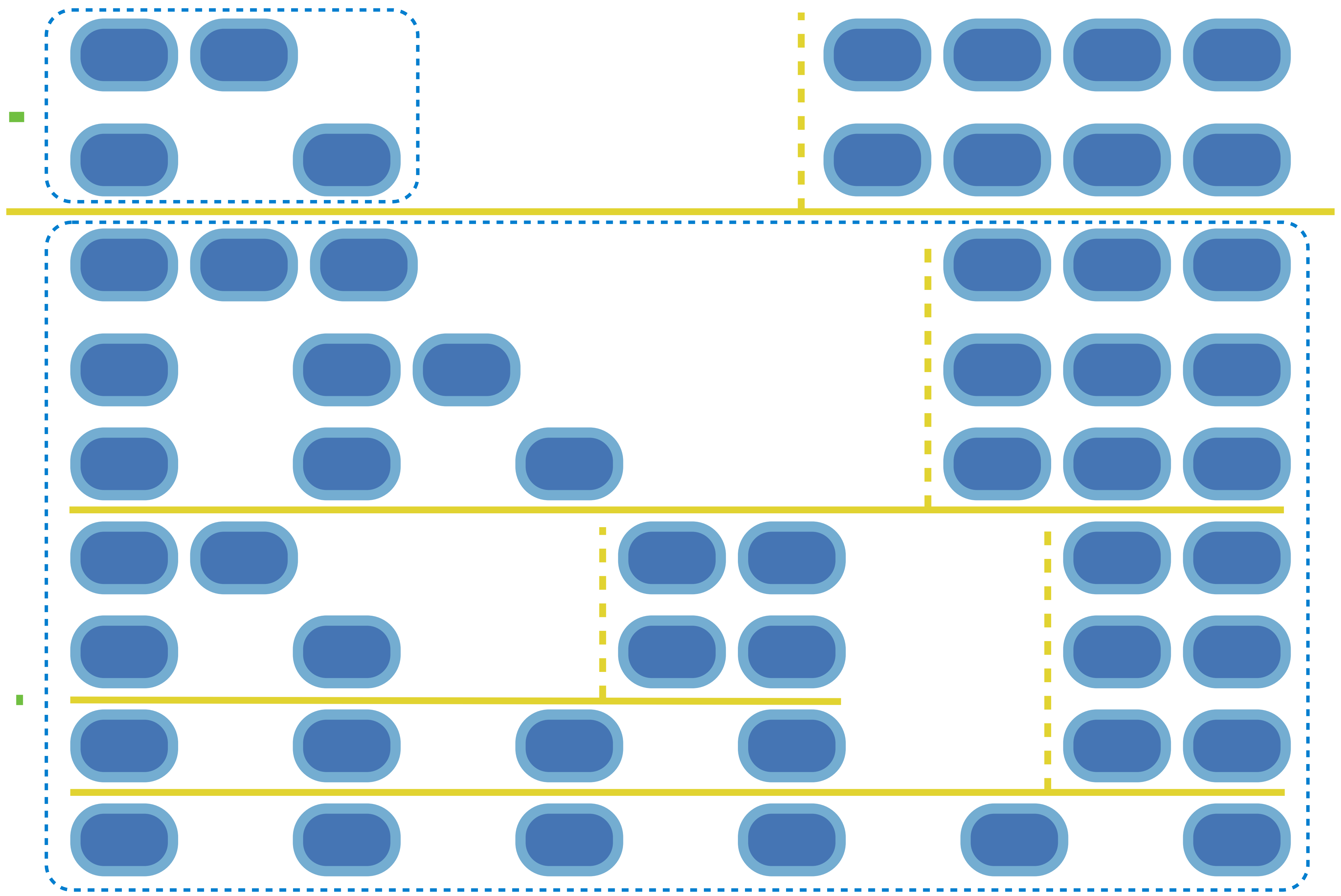


Counting Partitions

The number of partitions of a positive integer n, using parts up to size m, is the number of ways in which n can be expressed as the sum of positive integer parts up to m in non-decreasing order.

`count_partitions(6, 4)`

- Recursive decomposition: finding simpler instances of the problem.
- Explore two possibilities:
  - Use at least one 4
  - Don't use any 4
- Solve two simpler problems:
  - count_partitions(2, 4)
  - count_partitions(6, 3)
- Tree recursion often involves exploring different choices.



Counting Partitions

The number of partitions of a positive integer n, using parts up to size m, is the number of ways in which n can be expressed as the sum of positive integer parts up to m in increasing order.

- Recursive decomposition: finding simpler instances of the problem.
- Explore two possibilities:
  - Use at least one 4
  - Don't use any 4
- Solve two simpler problems:
  - count_partitions(2, 4)
  - count_partitions(6, 3)
- Tree recursion often involves exploring different choices.

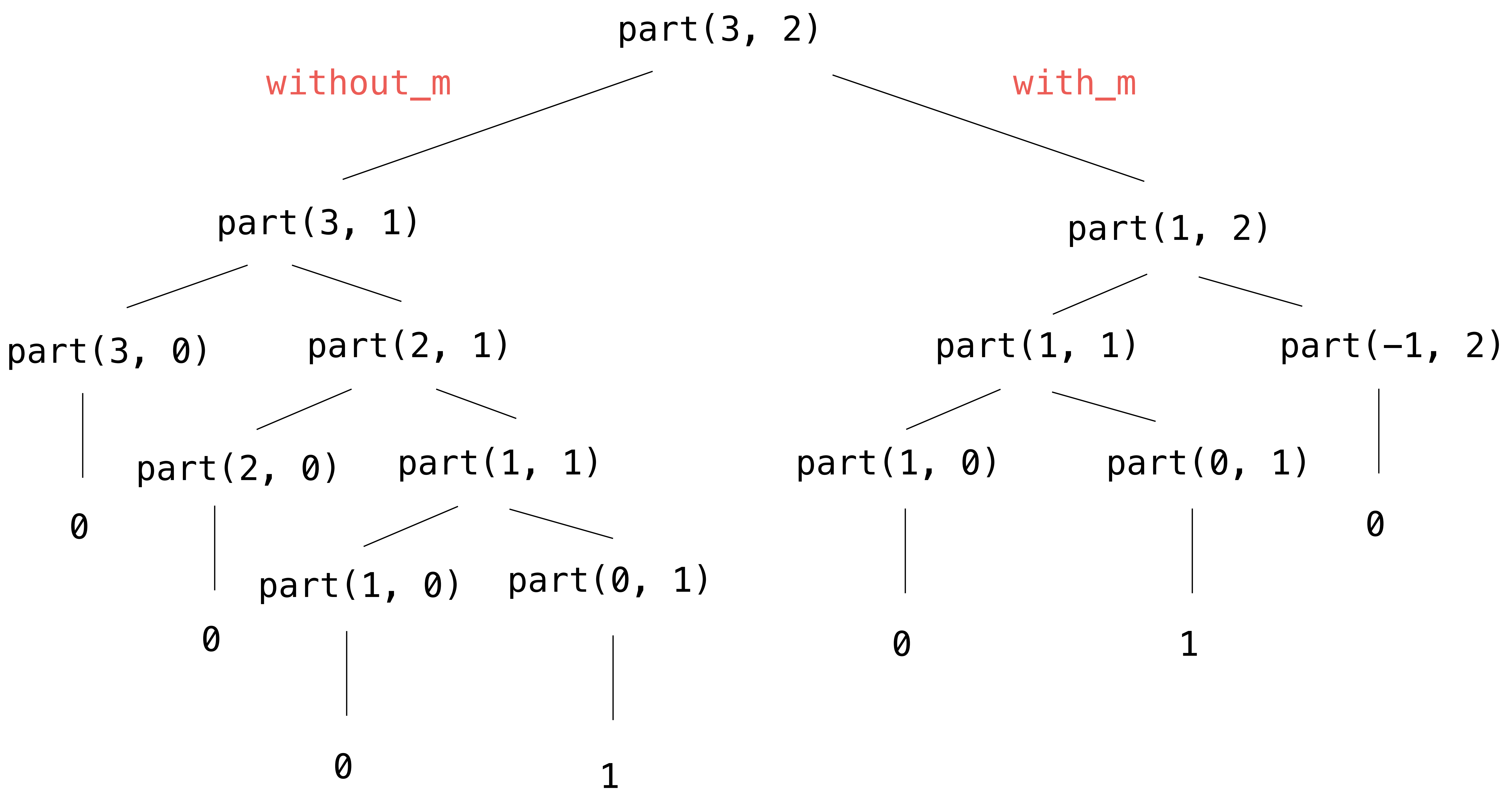
```
def count_partitions(n, m):
    if n == 0:
        return 1
    elif n < 0:
        return 0
    elif m == 0:
        return 0

    else:
        with_m = count_partitions(n-m, m)
        without_m = count_partitions(n, m-1)
        return with_m + without_m
```

(Demo)

pythontutor.com/composingprograms.html#code=def%20count_partitions%28n,%20m%29%3A%0A%20%20%20%20if%20n%20%3D%3D%200%3A%0A%20%20%20%20%20%20%20%20return%201%0A%20%20%20%20elif%20n%20<%200%3A%0A%20%20%20%20%20%20%20%20return%200%0A%20%20%20%20elif%20m%20%3D%3D%200%3A%0A%20%20%20%20%20%20%20%20return%200%0A%20%20%20%20else%3A%0A%20%20%20%20%20%20%20%20with_m%20%3D%20count_partitions%28n-m,
%20m%29%20%0A%20%20%20%20%20%20%20%20without_m%20%3D%20count_partitions%28n,
%20m-1%29%0A%20%20%20%20%20%20%20%20return%20with_m%20%2B%20without_m%0A%20%20%20%20%20%20%20%20%0Aresult%20%3D%20count_partitions%285,%203%29%0A%0A#%201%20%2B%201%20%2B%201%20%2B%201%20%2B%201%20%3D%205%0A#%201%20%2B%201%20%2B%201%20%2B%202%20%2B%20%20%20%3D%205%0A#%201%20%2B%202%20%2B%202%20%2B%20%20%20%20%20%20%20%3D%205%0A#%201%20%2B%201%20%2B%203%20%2B%20%20%20%20%20%20%20%3D%205%0A
20%2B%203%20%2B%20%20%20%20%20%20%20%20%20%20%20%20%20%3D%205&mode=display&origin=composingprograms.js&cumulative=false&py=3&rawInputLstJSON=[]&curInstr=0

Counting Partitions Call Tree



`part(3, 2) = 0 + 0 + 0 + 1 + 0 + 1 + 0 = 2`