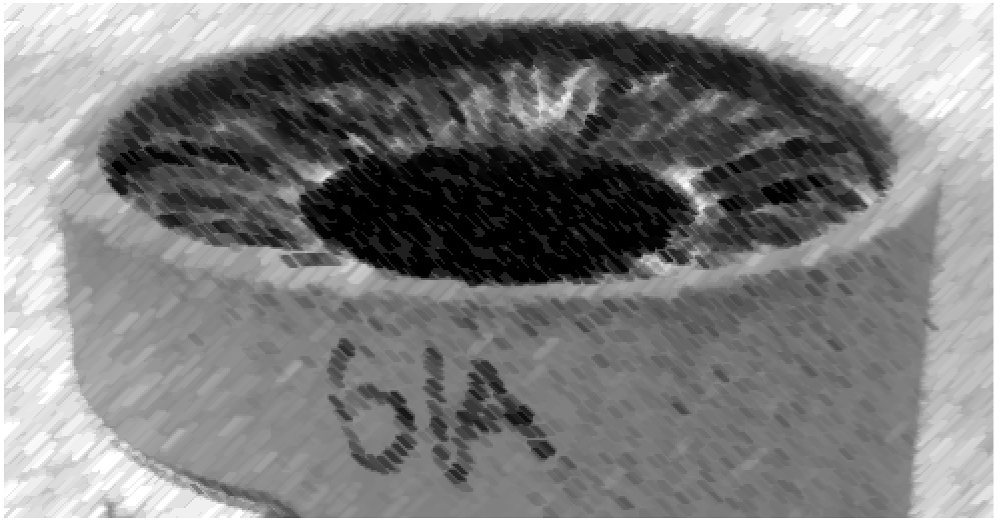


CS61A Lecture 11

Amir Kamil
UC Berkeley
February 15, 2013

Announcements

- HW4 due Wednesday at 11:59pm
- Hog contest deadline next week
  - Completely optional, opportunity for extra credit
  - See website for details

Fibonacci Sequence

The Fibonacci sequence is defined as

$$\mathrm{fib}(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ \mathrm{fib}(n-1) + \mathrm{fib}(n-2), & n > 1 \end{cases}$$

Example: http://goo.gl/DZbRG

Fibonacci Sequence

The Fibonacci sequence is defined as

$$\mathrm{fib}(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ \mathrm{fib}(n-1) + \mathrm{fib}(n-2), & n > 1 \end{cases}$$

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

Example: http://goo.gl/DZbRG

Fibonacci Sequence

The Fibonacci sequence is defined as

$$\mathrm{fib}(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ \mathrm{fib}(n-1) + \mathrm{fib}(n-2), & n > 1 \end{cases}$$

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

Example: http://goo.gl/DZbRG

Fibonacci Sequence

The Fibonacci sequence is defined as

$$\mathrm{fib}(n) = \begin{cases} 0, & n = 0 \\ 1, & n = 1 \\ \mathrm{fib}(n-1) + \mathrm{fib}(n-2), & n > 1 \end{cases}$$

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

Two recursive calls!

Example: http://goo.gl/DZbRG

Tree recursion

Tree recursion

Executing the body of a function may entail more than one recursive call to that function

Tree recursion

Executing the body of a function may entail more than one recursive call to that function

This is called *tree recursion*

Tree recursion

Executing the body of a function may entail more than one recursive call to that function

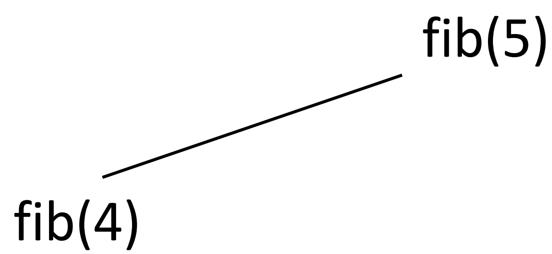
This is called *tree recursion*

fib(5)

Tree recursion

Executing the body of a function may entail more than one recursive call to that function

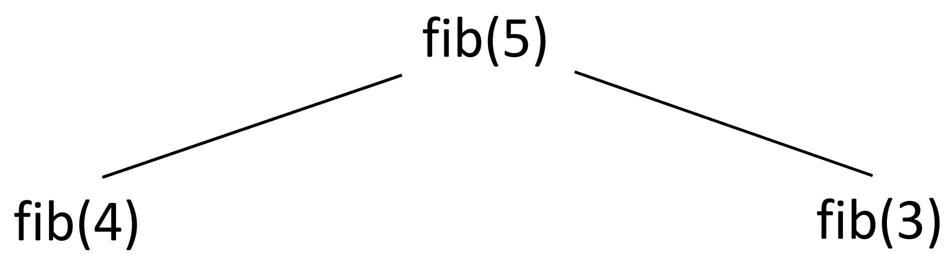
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

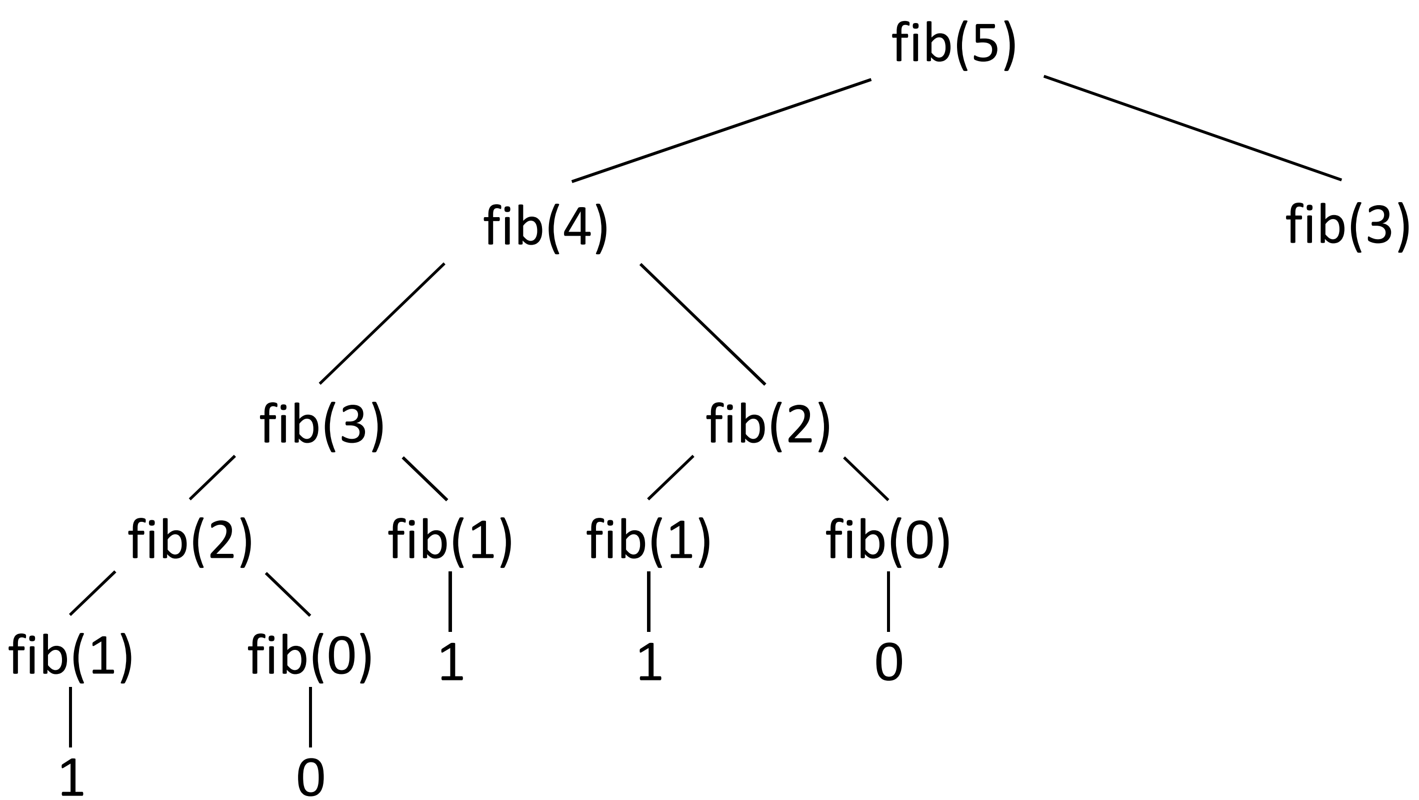
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

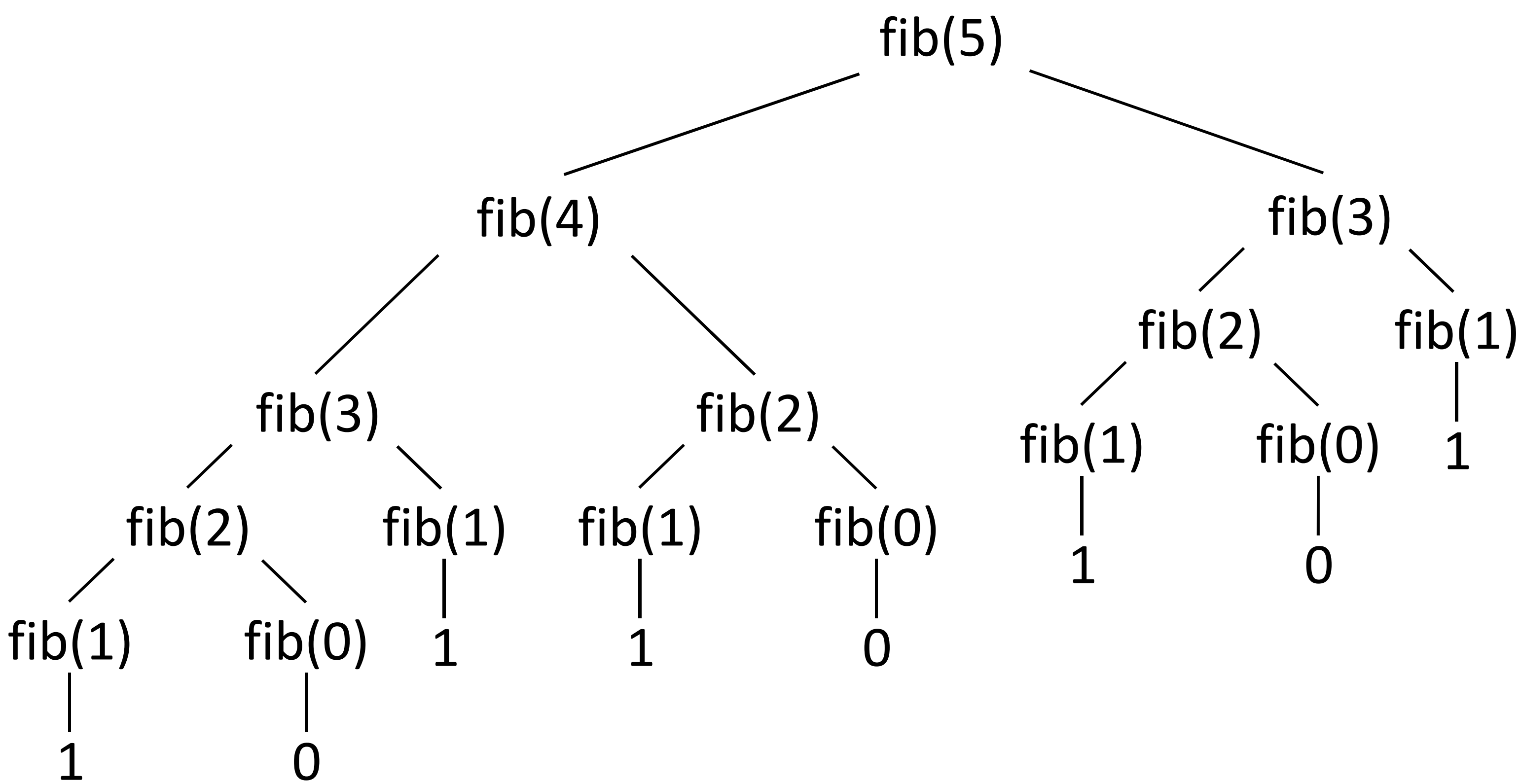
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

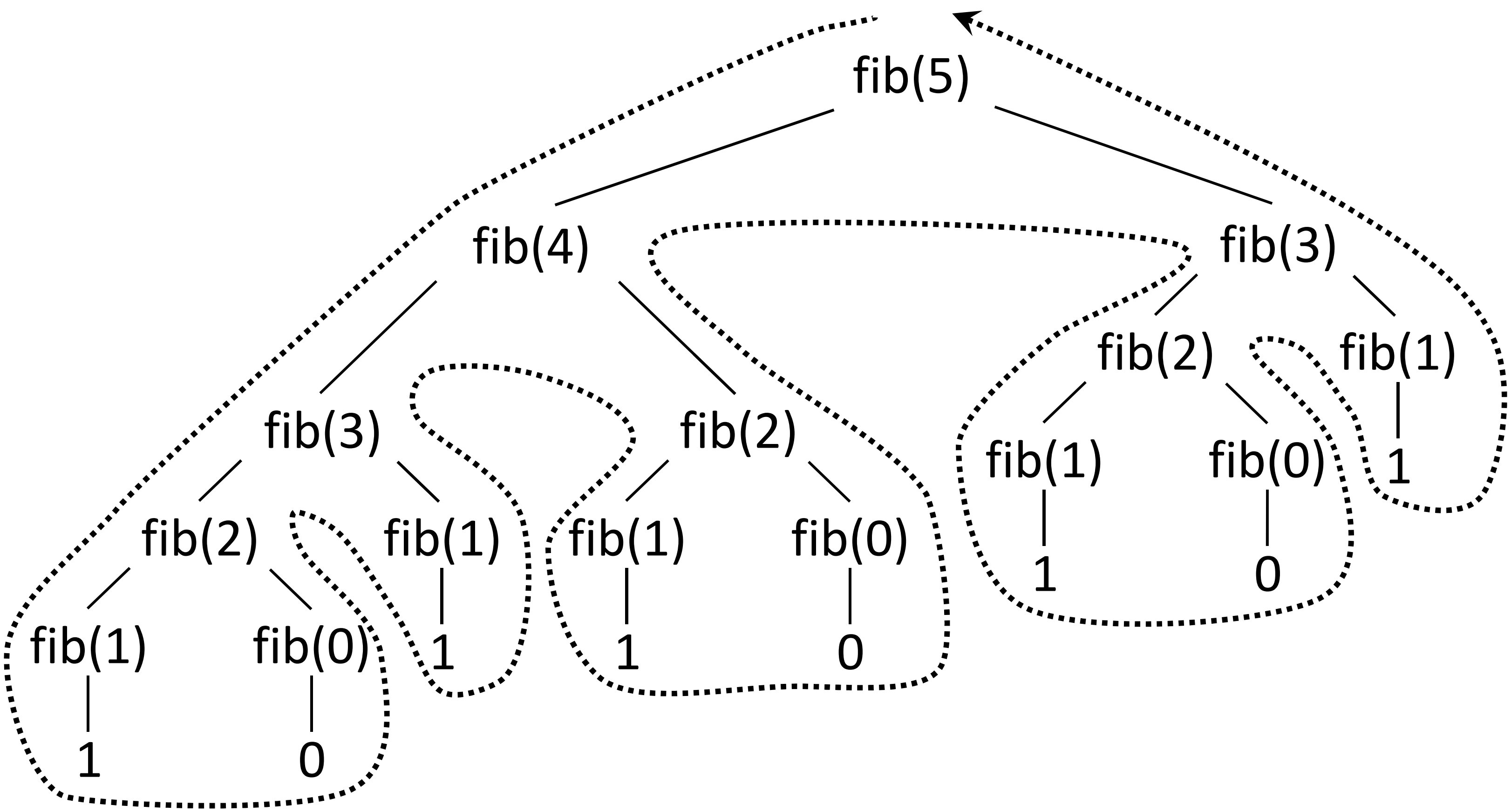
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

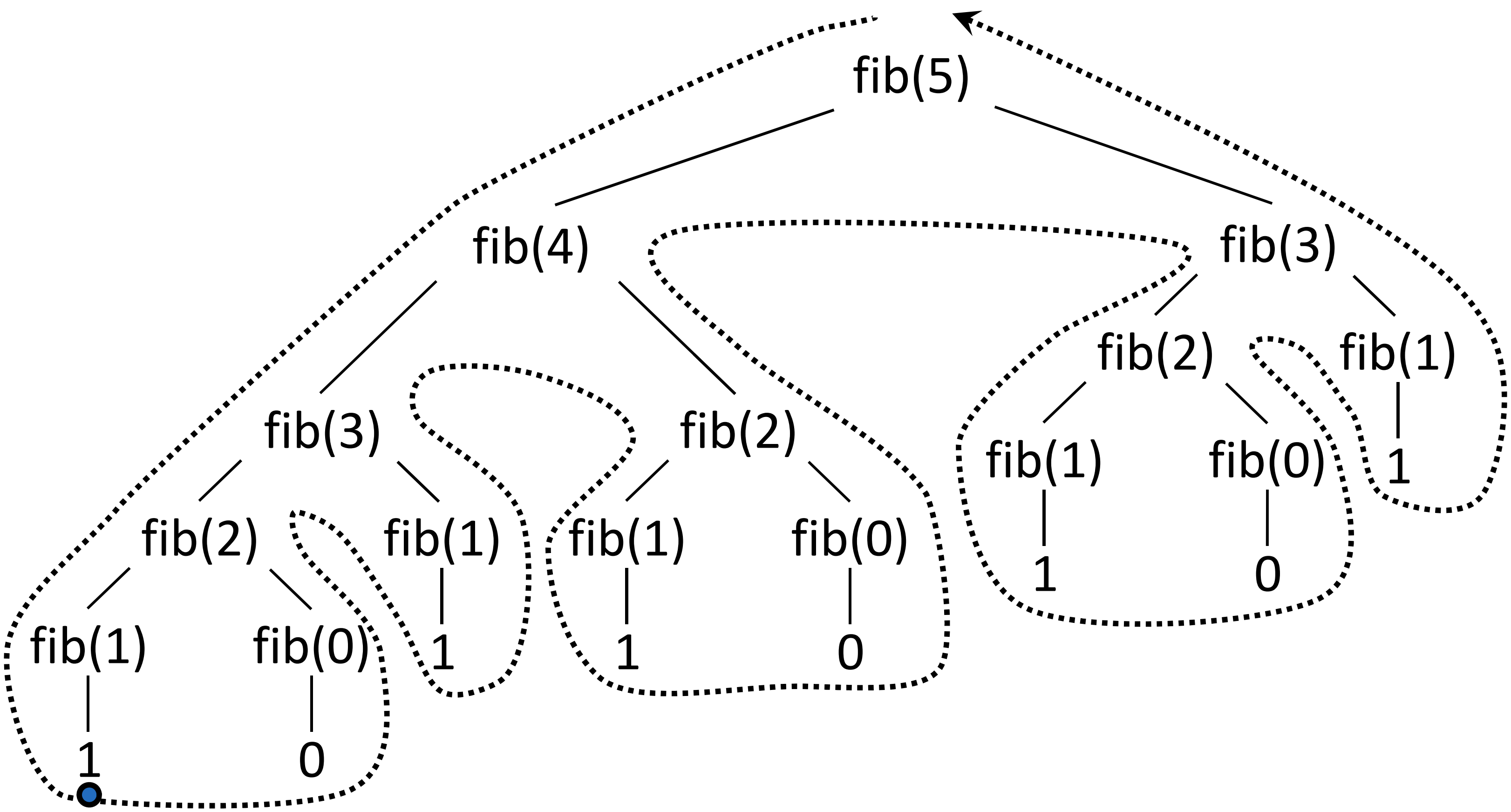
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

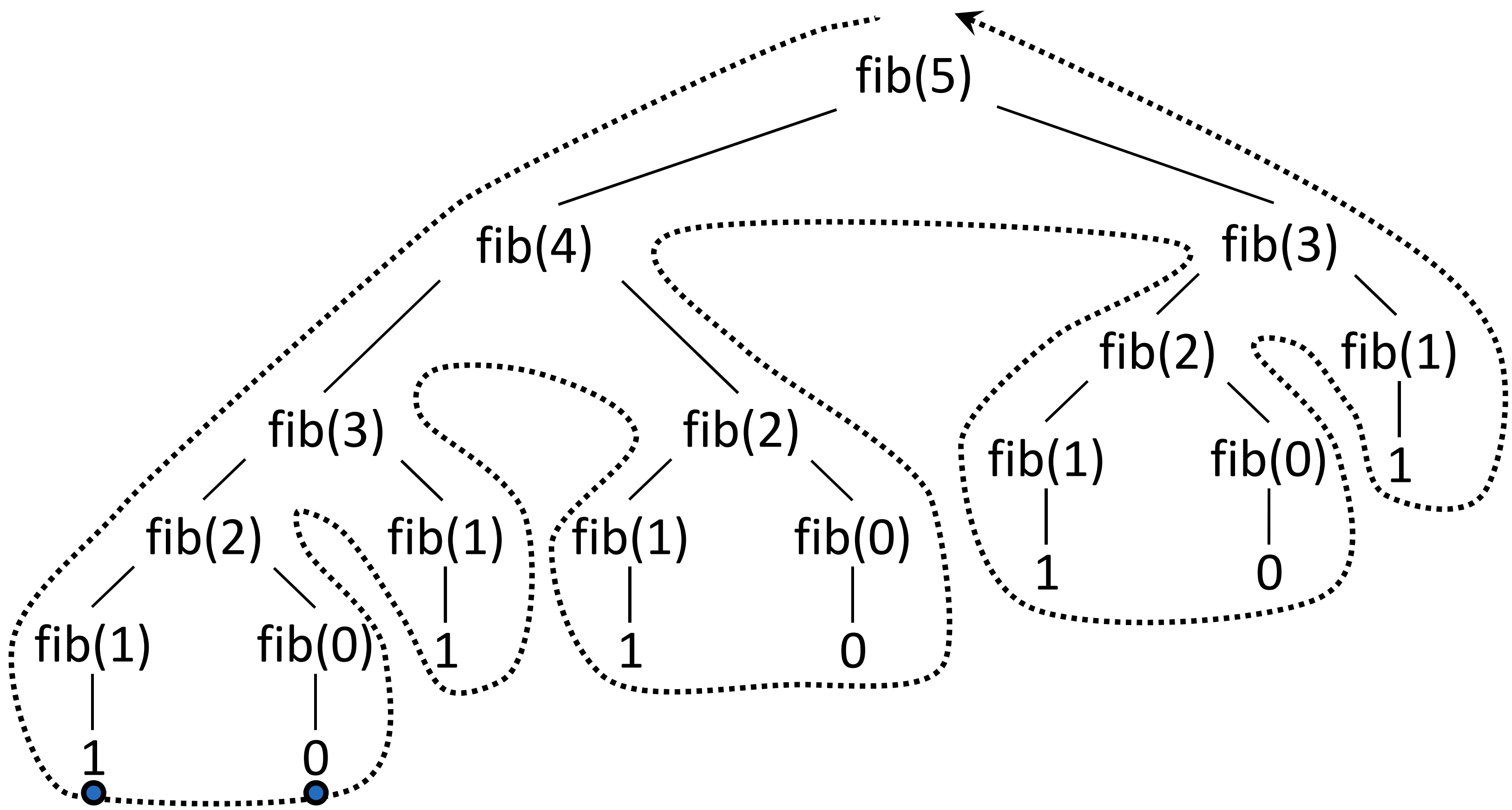
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

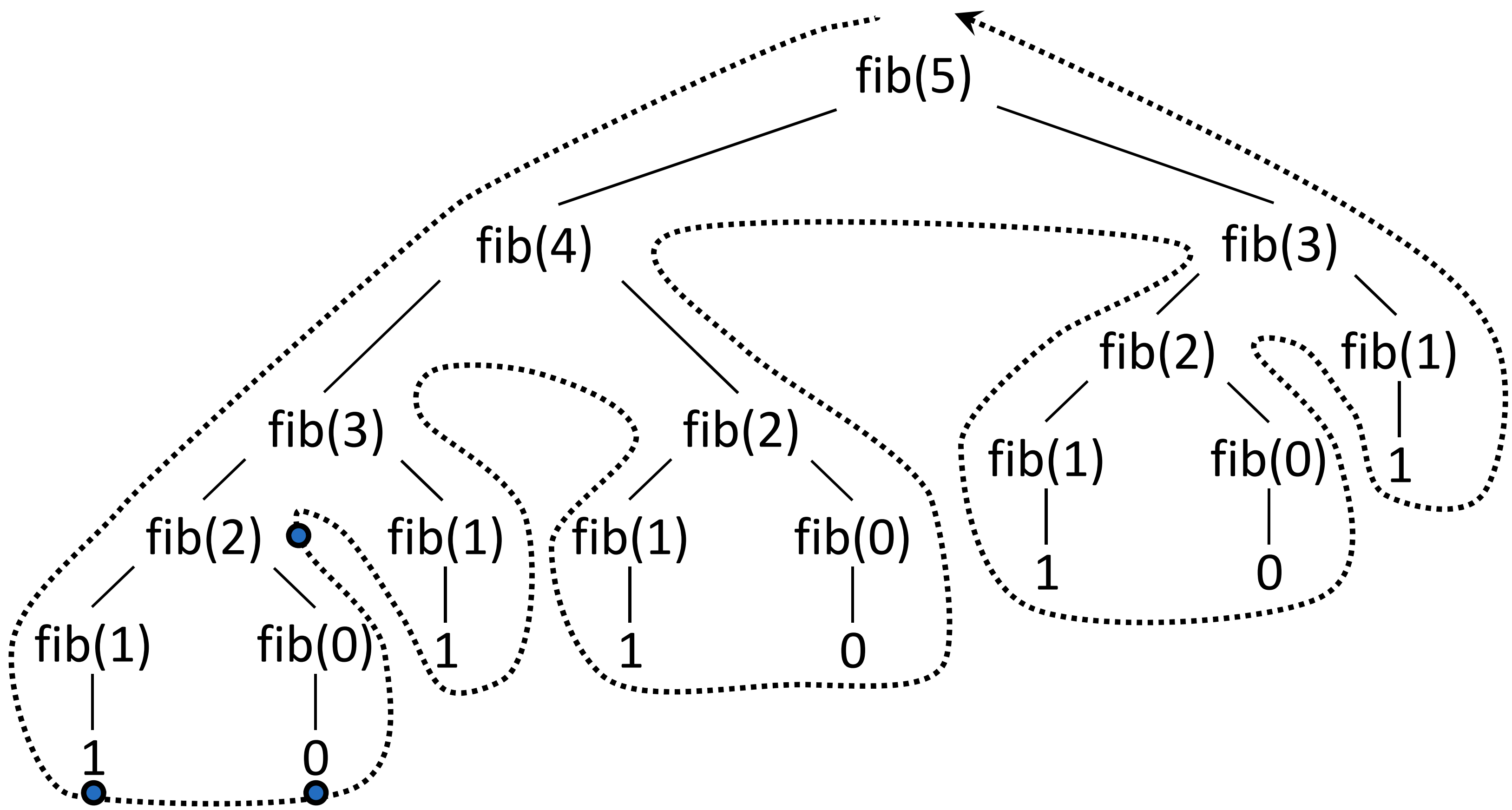
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

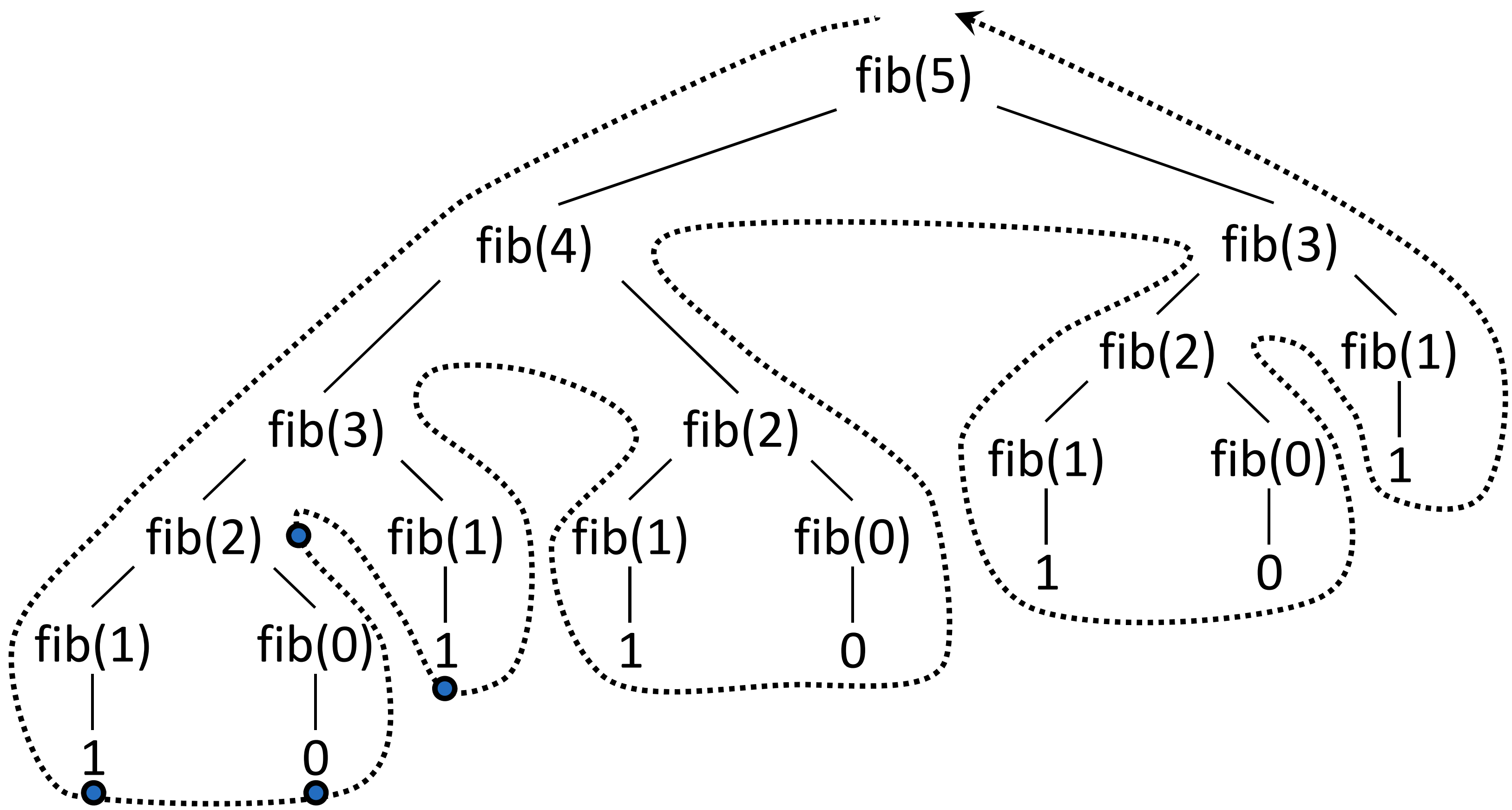
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

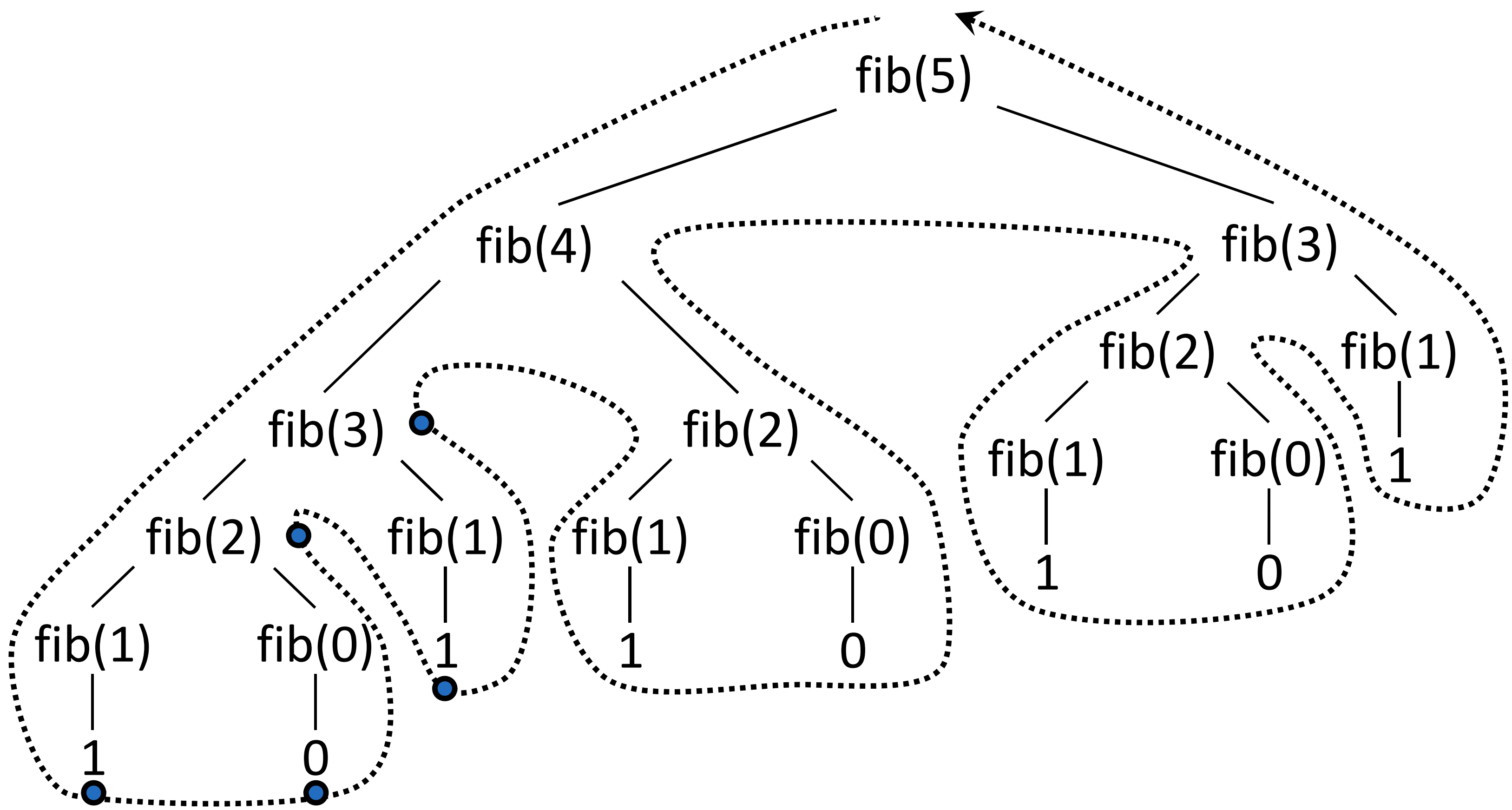
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

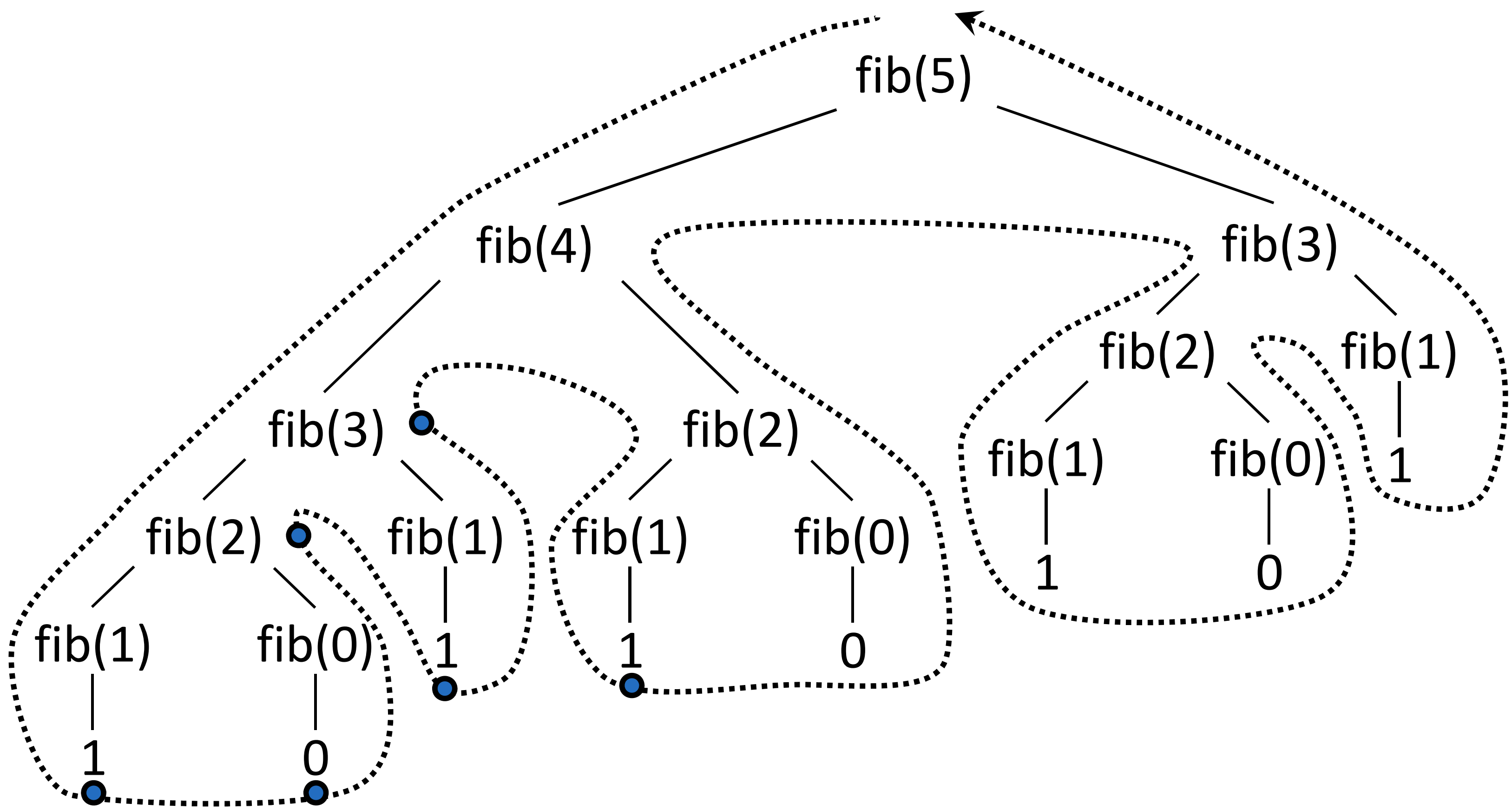
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

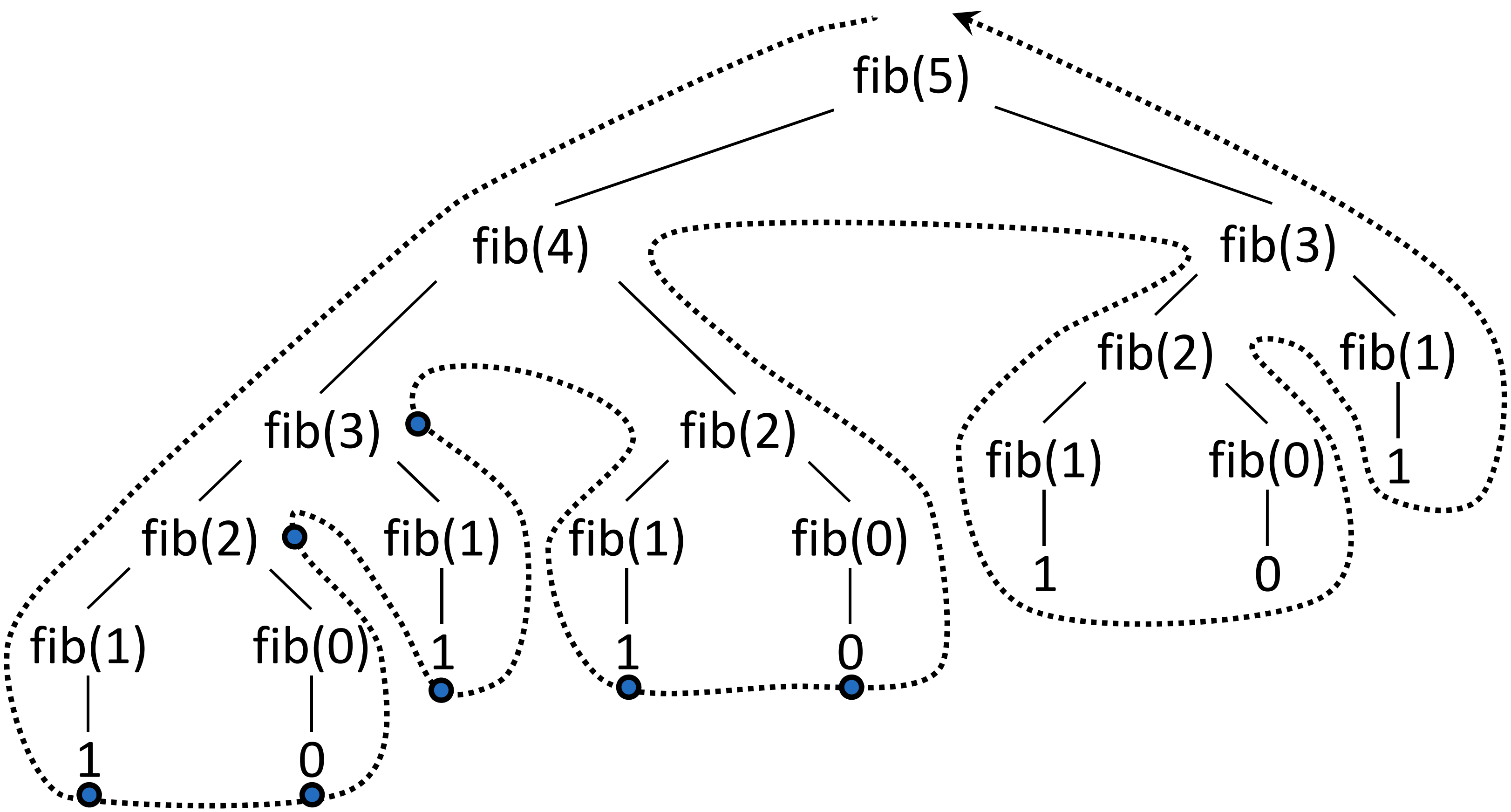
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

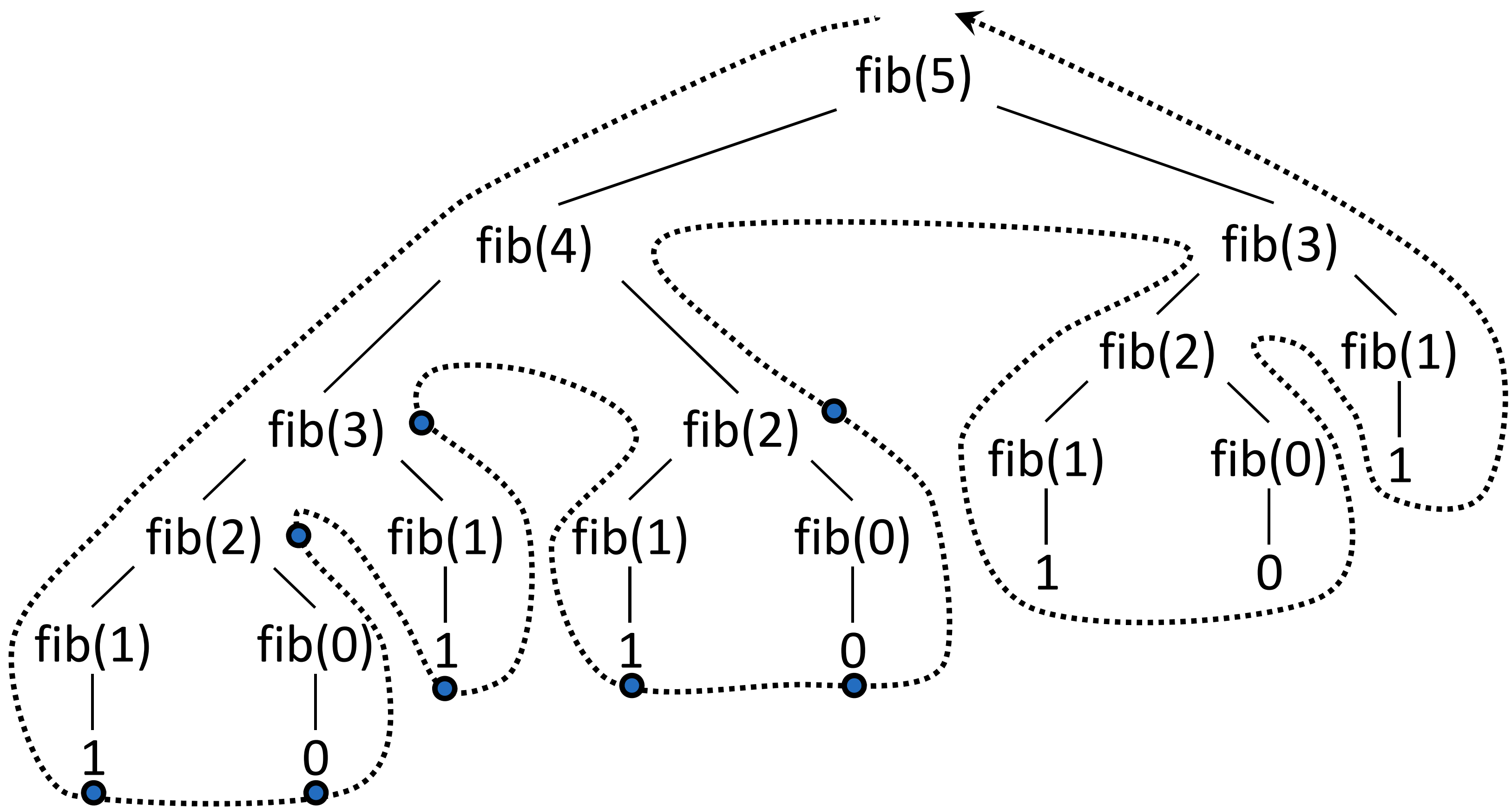
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

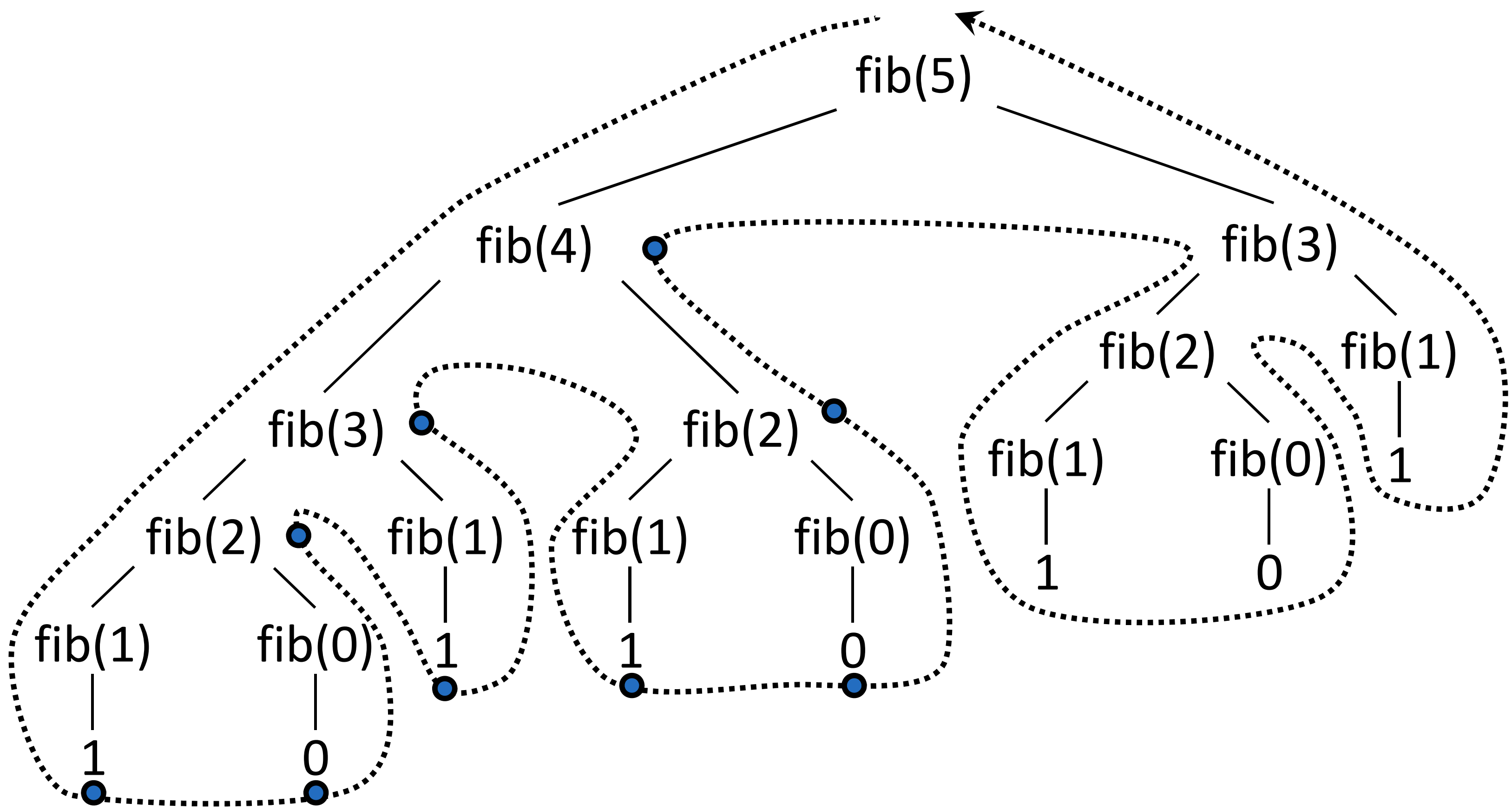
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

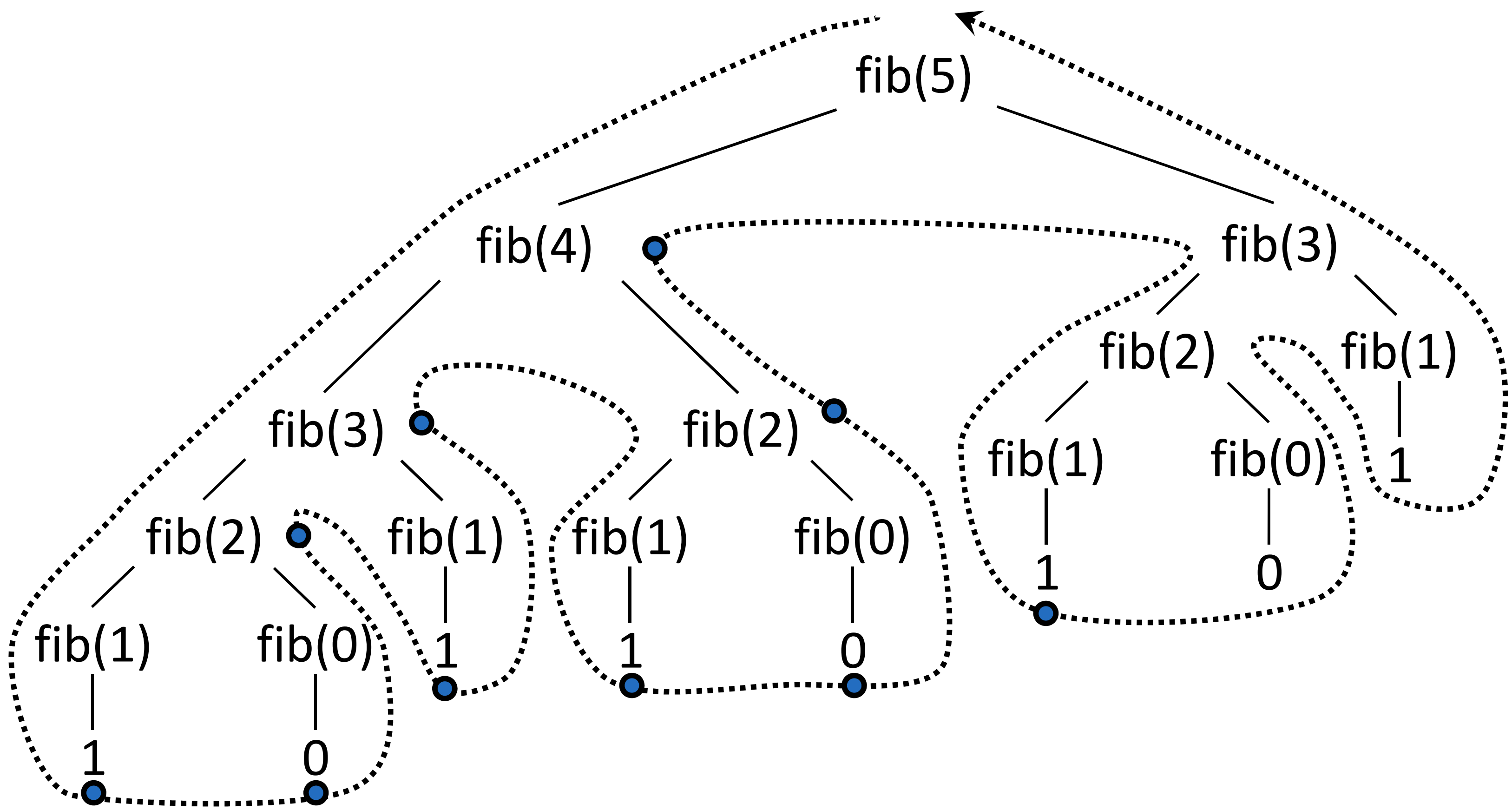
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

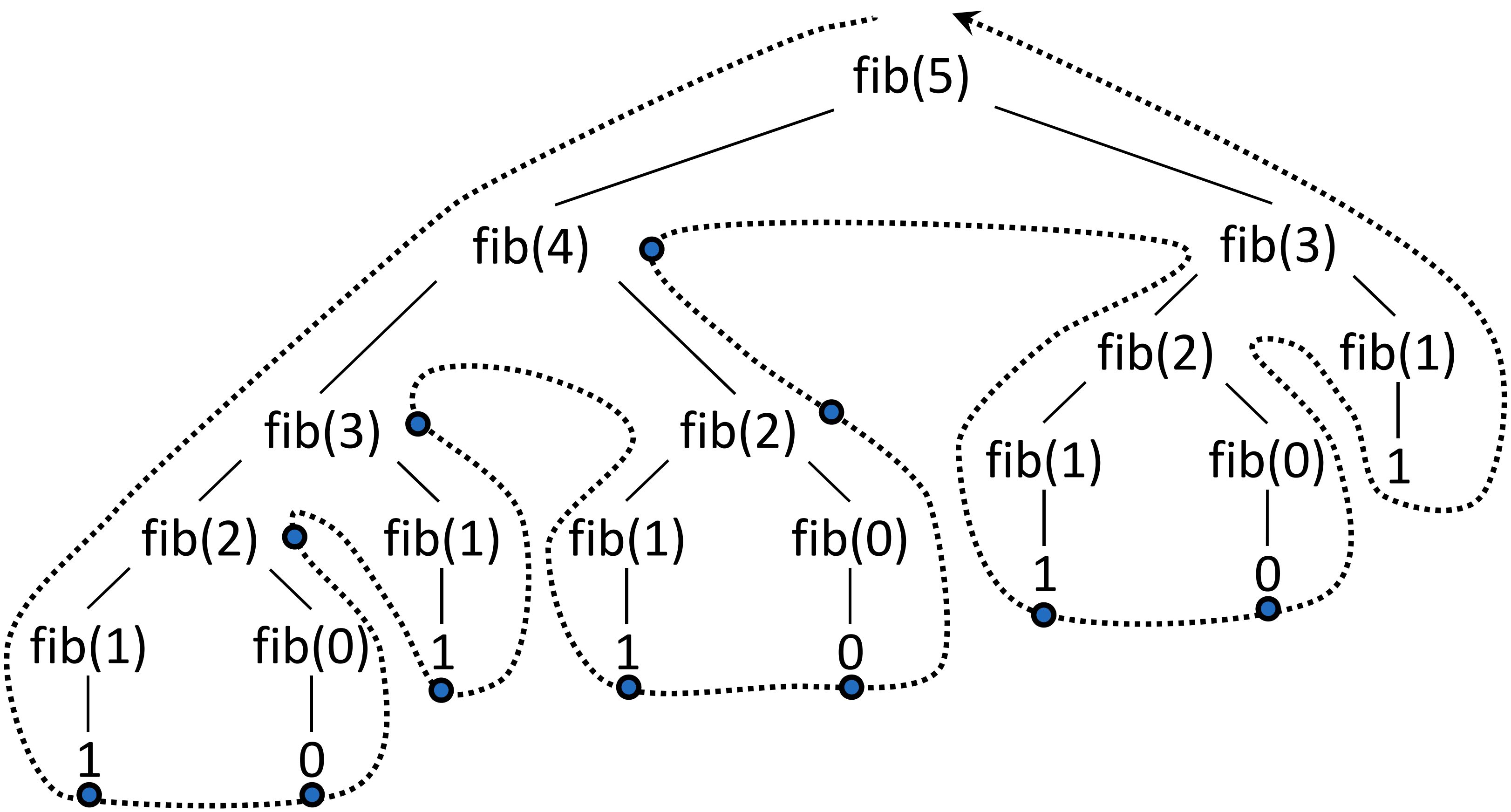
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

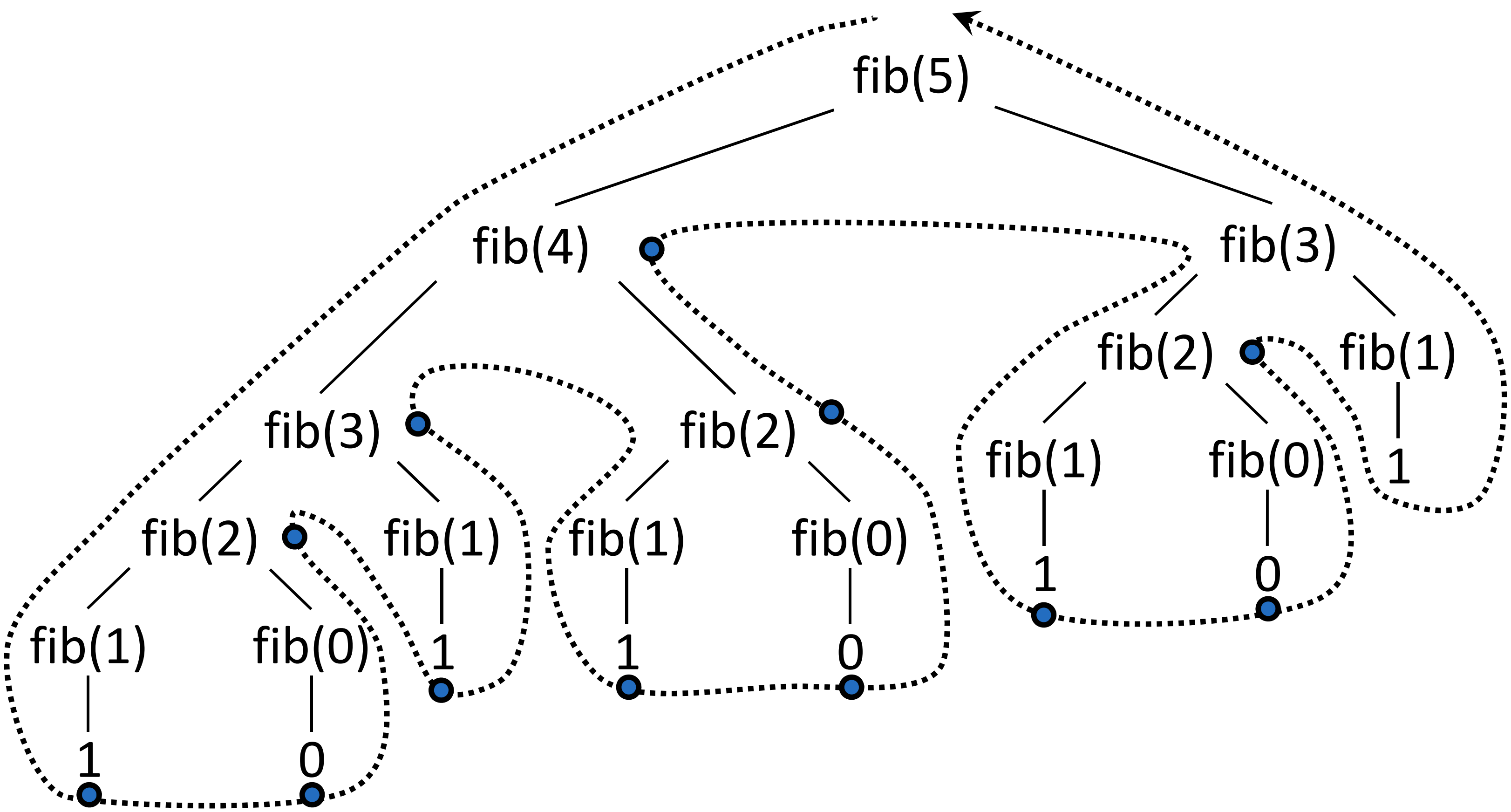
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

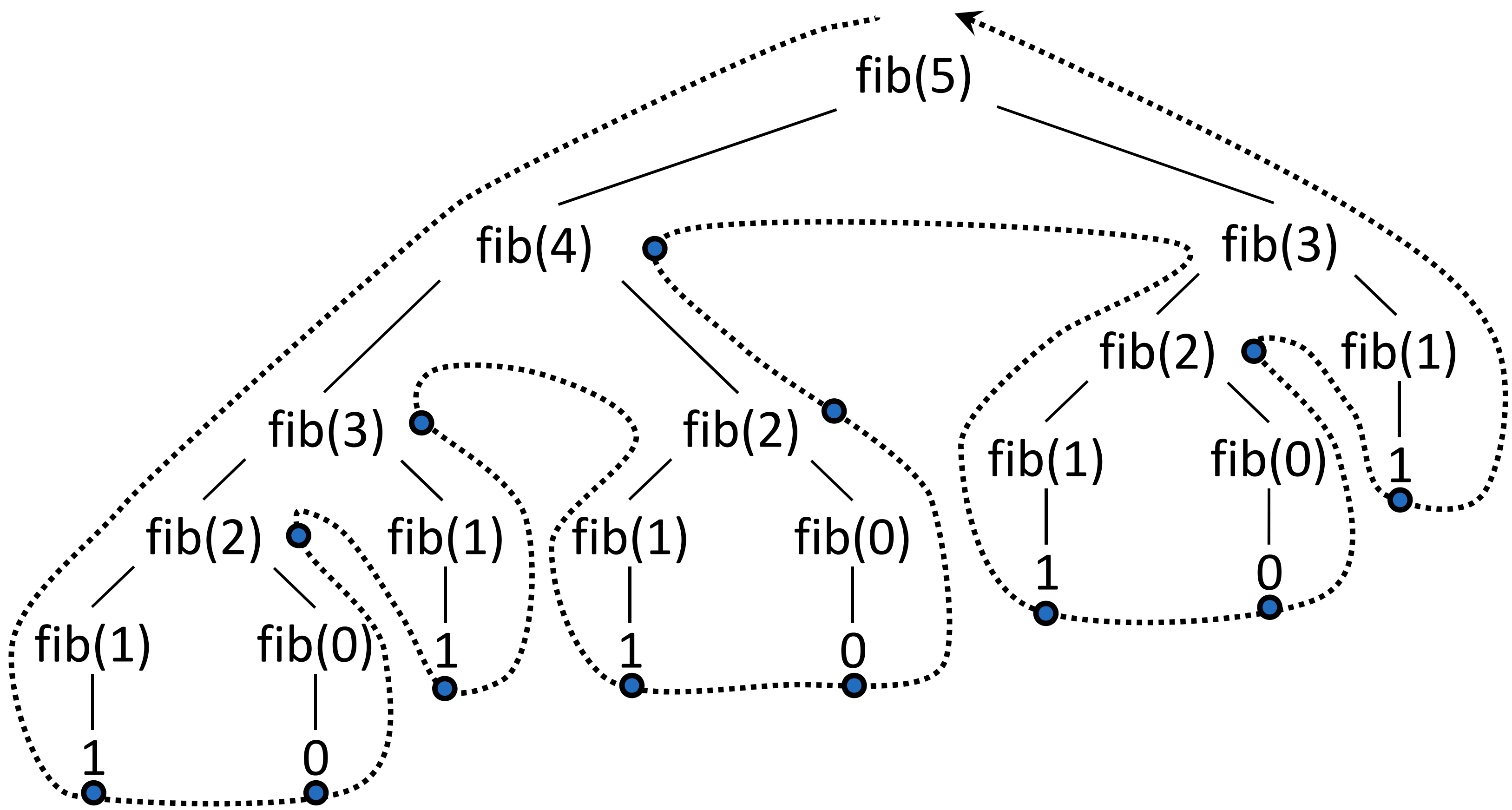
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

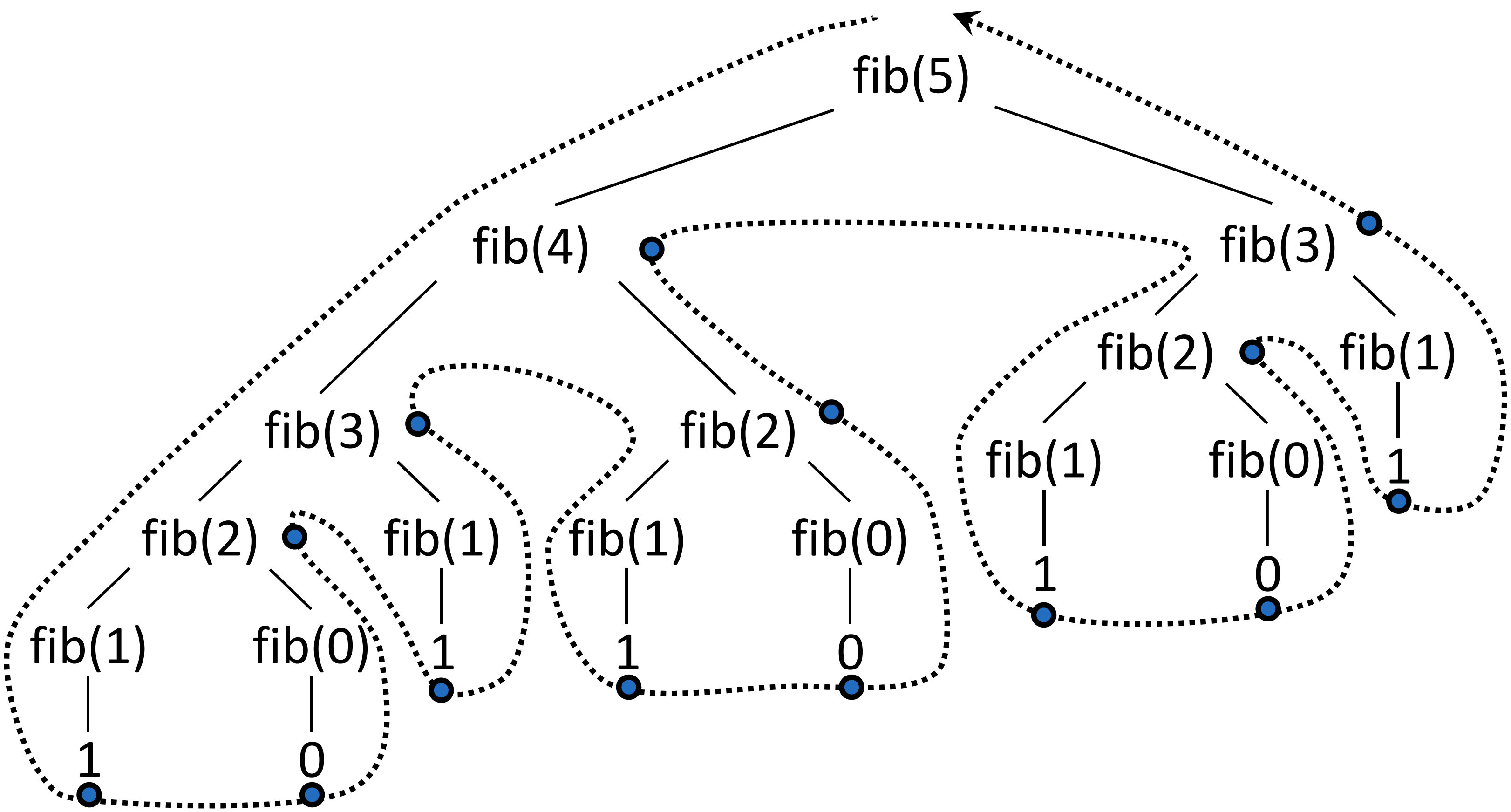
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

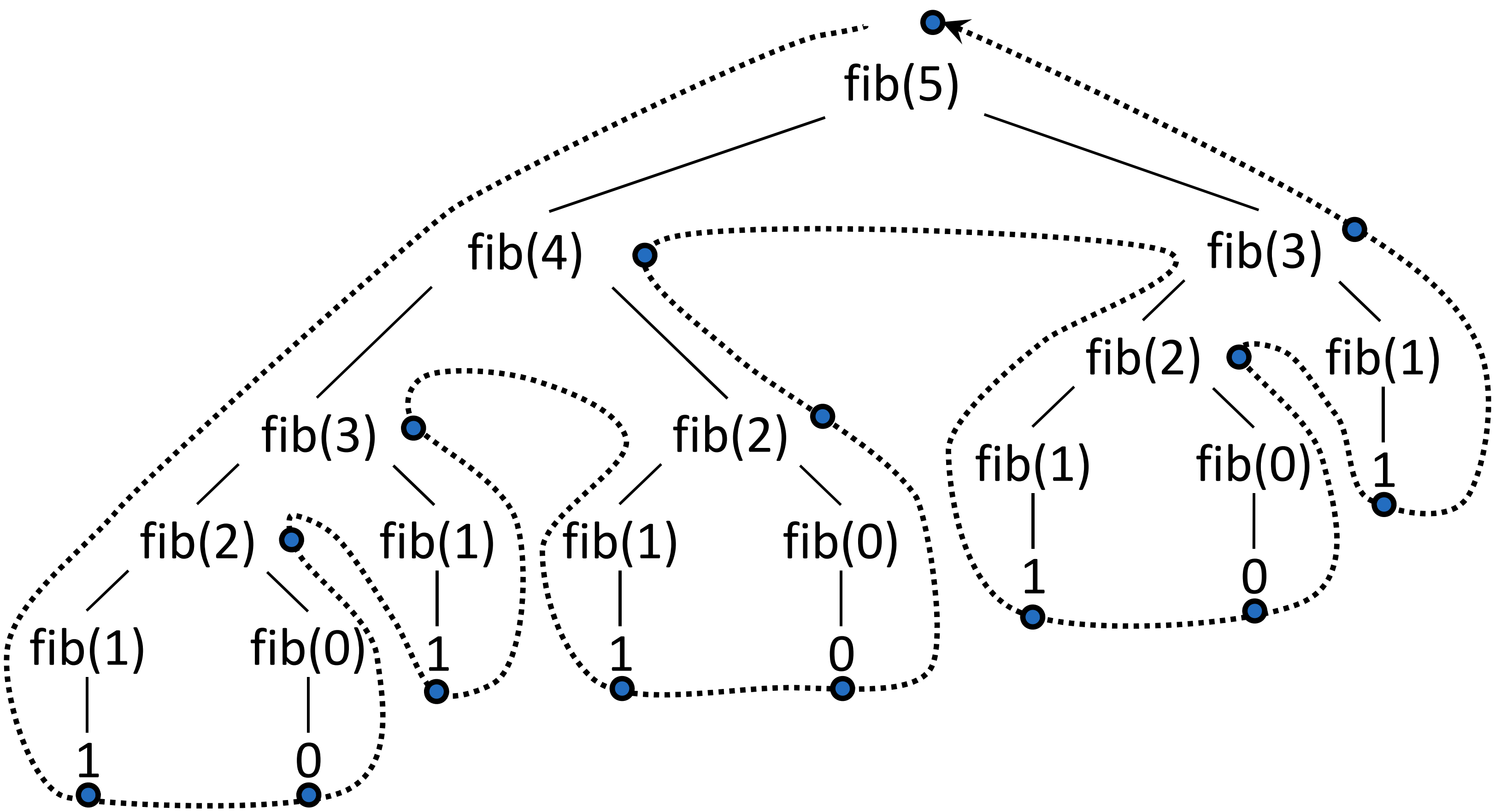
This is called *tree recursion*



Tree recursion

Executing the body of a function may entail more than one recursive call to that function

This is called *tree recursion*



Tracing the Order of Calls

Tracing the Order of Calls

We can use a higher-order function to see the order in which calls are made and complete

Tracing the Order of Calls

We can use a higher-order function to see the order in which calls are made and complete

```
def trace1(fn):
    """Return a function equivalent to fn that
    also prints trace output."""
    def traced(x):
        print('Calling', fn, '(', x, ')')
        res = fn(x)
        print('Got', res, 'from', fn, '(', x, ')')
        return res
    return traced
```

Tracing the Order of Calls

We can use a higher-order function to see the order in which calls are made and complete

```
def trace1(fn):
    """Return a function equivalent to fn that
    also prints trace output."""
    def traced(x):
        print('Calling', fn, '(', x, ')')
        res = fn(x)
        print('Got', res, 'from', fn, '(', x, ')')
        return res
    return traced

# Rebind the name fib to a traced version of fib
fib = trace1(fib)
```

Function Decorators

Function Decorators

```
@trace1
def triple(x):
    return 3 * x
```

Function Decorators

Function decorator

```
@trace1
def triple(x):
    return 3 * x
```

Function Decorators

Function decorator

```
@trace1
def triple(x):
    return 3 * x
```

Decorated function

Function Decorators

Function decorator

```
@trace1
def triple(x):
    return 3 * x
```

Decorated function

is identical to

Function Decorators

Function decorator

```
@trace1
def triple(x):
    return 3 * x
```

Decorated function

is identical to

```
def triple(x):
    return 3 * x
triple = trace1(triple)
```

Function Decorators

Function decorator

```
@trace1
def triple(x):
    return 3 * x
```

Decorated function

is identical to

Why not just use this?

```
def triple(x):
    return 3 * x
triple = trace1(triple)
```

The Recursive Leap of Faith

The Recursive Leap of Faith

```
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)
```

The Recursive Leap of Faith

```python
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)
```

Is factorial implemented correctly?

The Recursive Leap of Faith

```
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)
```

Is factorial implemented correctly?

1. Verify the base case.

The Recursive Leap of Faith

```
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)
```

Is factorial implemented correctly?

1. Verify the base case.
2. Treat **`factorial(n-1)`** as a functional abstraction.

The Recursive Leap of Faith

```
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)
```

Is factorial implemented correctly?

1. Verify the base case.
2. Treat `factorial(n-1)` as a functional abstraction.
3. Assume that `factorial(n-1)` is correct.

The Recursive Leap of Faith

```
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)
```

Is factorial implemented correctly?

1. Verify the base case.
2. Treat **`factorial(n-1)`** as a functional abstraction.
3. Assume that **`factorial(n-1)`** is correct.

Indiana Jones and The Last Crusade
© Lucasfilm, Ltd.

The Recursive Leap of Faith

```
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)
```

Is factorial implemented correctly?

1. Verify the base case.
2. Treat **`factorial(n-1)`** as a functional abstraction.
3. Assume that **`factorial(n-1)`** is correct.
4. Verify that **`factorial(n)`** is correct, assuming that **`factorial(n-1)`** is correct

Indiana Jones and The Last Crusade
© Lucasfilm, Ltd.

The Recursive Leap of Faith

```
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)
```

Oops!

Is factorial implemented correctly?

1. Verify the base case.
2. Treat **`factorial(n-1)`** as a functional abstraction.
3. Assume that **`factorial(n-1)`** is correct.
4. Verify that **`factorial(n)`** is correct, assuming that **`factorial(n-1)`** is correct

Indiana Jones and The Last Crusade
© Lucasfilm, Ltd.

The Recursive Leap of Faith

```python
def factorial(n):
    if n == 0:
        return 1
    return factorial(n-1)
```

Oops!

Is factorial implemented correctly?

1. Verify the base case.
2. Treat **`factorial(n-1)`** as a functional abstraction.
3. Assume that **`factorial(n-1)`** is correct.
4. Verify that **`factorial(n)`** is correct, assuming that **`factorial(n-1)`** is correct

Simpler problem

Indiana Jones and The Last Crusade
© Lucasfilm, Ltd.

Simplifying a Problem

Simplifying a Problem

Pig Latinization:

Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word

Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

smart → artsmay

Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

smart → artsmay

```
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))
```

Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

smart → artsmay

```
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))
```

smart

Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

smart → artsmay

```
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))
```

smart → marts

Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

smart → artsmay

```
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))
```

smart → marts → artsm

Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

smart → artsmay

```
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))
```

smart → marts → artsm → artsmay

Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

smart → artsmay

```
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))
```

smart → marts → artsm → artsmay

2 consonants to be moved

Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

smart → artsmay

```
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))
```

smart → marts → artsm → artsmay

2 consonants to be moved

1 consonant to be moved

Simplifying a Problem

Pig Latinization:

1. Move all beginning consonants to the end of the word
2. Add “ay” to the end of the word

smart → artsmay

```
def pig_latin(w):
    if starts_with_a_vowel(w):
        return w + 'ay'
    return pig_latin(rest(w) + first(w))
```

smart → marts → artsm → artsmay

2 consonants to be moved

1 consonant to be moved

Base case

Counting Change

Counting Change

\$1 = \$0.50 + \$0.25 + \$0.10 + \$0.10 + \$0.05

Counting Change

\$1 = \$0.50 + \$0.25 + \$0.10 + \$0.10 + \$0.05

\$1 = 1 half dollar, 1 quarter, 2 dimes, 1 nickel

Counting Change

\$1 = \$0.50 + \$0.25 + \$0.10 + \$0.10 + \$0.05

\$1 = 1 half dollar, 1 quarter, 2 dimes, 1 nickel

\$1 = 2 quarters, 2 dimes, 30 pennies

Counting Change

\$1 = \$0.50 + \$0.25 + \$0.10 + \$0.10 + \$0.05

\$1 = 1 half dollar, 1 quarter, 2 dimes, 1 nickel

\$1 = 2 quarters, 2 dimes, 30 pennies

\$1 = 100 pennies

Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1 half dollar, 1 quarter, 2 dimes, 1 nickel

$1 = 2 quarters, 2 dimes, 30 pennies

$1 = 100 pennies

How many ways are there to change a dollar?

Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1 half dollar, 1 quarter, 2 dimes, 1 nickel

$1 = 2 quarters, 2 dimes, 30 pennies

$1 = 100 pennies

How many ways are there to change a dollar?

How many ways to change $0.11?

Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1 half dollar, 1 quarter, 2 dimes, 1 nickel

$1 = 2 quarters, 2 dimes, 30 pennies

$1 = 100 pennies

How many ways are there to change a dollar?

How many ways to change $0.11?

Use a dime

Counting Change

\$1 = \$0.50 + \$0.25 + \$0.10 + \$0.10 + \$0.05

\$1 = 1 half dollar, 1 quarter, 2 dimes, 1 nickel

\$1 = 2 quarters, 2 dimes, 30 pennies

\$1 = 100 pennies

How many ways are there to change a dollar?

How many ways to change \$0.11?

Use a dime

Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1 half dollar, 1 quarter, 2 dimes, 1 nickel

$1 = 2 quarters, 2 dimes, 30 pennies

$1 = 100 pennies

How many ways are there to change a dollar?

How many ways to change $0.11?

Use a dime

10 1

Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1 half dollar, 1 quarter, 2 dimes, 1 nickel

$1 = 2 quarters, 2 dimes, 30 pennies

$1 = 100 pennies

How many ways are there to change a dollar?

How many ways to change $0.11?

Use a dime	**No dimes**
10 1	

Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1 half dollar, 1 quarter, 2 dimes, 1 nickel

$1 = 2 quarters, 2 dimes, 30 pennies

$1 = 100 pennies

How many ways are there to change a dollar?

How many ways to change $0.11?

Use a dime	**No dimes**	
	Use a nickel	
10 1		

Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1 half dollar, 1 quarter, 2 dimes, 1 nickel

$1 = 2 quarters, 2 dimes, 30 pennies

$1 = 100 pennies

How many ways are there to change a dollar?

How many ways to change $0.11?

Use a dime	**No dimes**
	Use a nickel
10 1	5 5 1

Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1 half dollar, 1 quarter, 2 dimes, 1 nickel

$1 = 2 quarters, 2 dimes, 30 pennies

$1 = 100 pennies

How many ways are there to change a dollar?

How many ways to change $0.11?

Use a dime	No dimes
	Use a nickel
10 1	5 5 1
	5 1 1 1 1 1 1

Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1 half dollar, 1 quarter, 2 dimes, 1 nickel

$1 = 2 quarters, 2 dimes, 30 pennies

$1 = 100 pennies

How many ways are there to change a dollar?

How many ways to change $0.11?

Use a dime	No dimes	
	Use a nickel	No nickles
10 1	5 5 1 5 1 1 1 1 1 1	

Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

$1 = 1 half dollar, 1 quarter, 2 dimes, 1 nickel

$1 = 2 quarters, 2 dimes, 30 pennies

$1 = 100 pennies

How many ways are there to change a dollar?

How many ways to change $0.11?

Use a dime	No dimes	
	Use a nickel	**No nickles**
10 1	5 5 1	1 1 1 1 1 1 1 1 1 1 1
	5 1 1 1 1 1 1	

Counting Change

$1 = $0.50 + $0.25 + $0.10 + $0.10 + $0.05

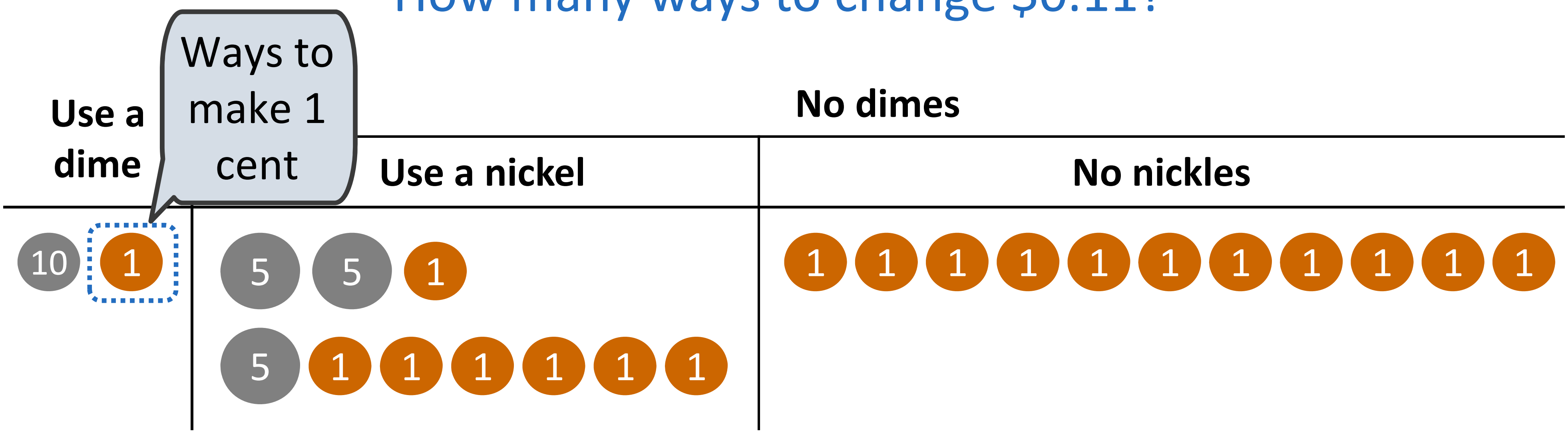
$1 = 1 half dollar, 1 quarter, 2 dimes, 1 nickel

$1 = 2 quarters, 2 dimes, 30 pennies

$1 = 100 pennies

How many ways are there to change a dollar?

How many ways to change $0.11?



Counting Change

\$1 = \$0.50 + \$0.25 + \$0.10 + \$0.10 + \$0.05

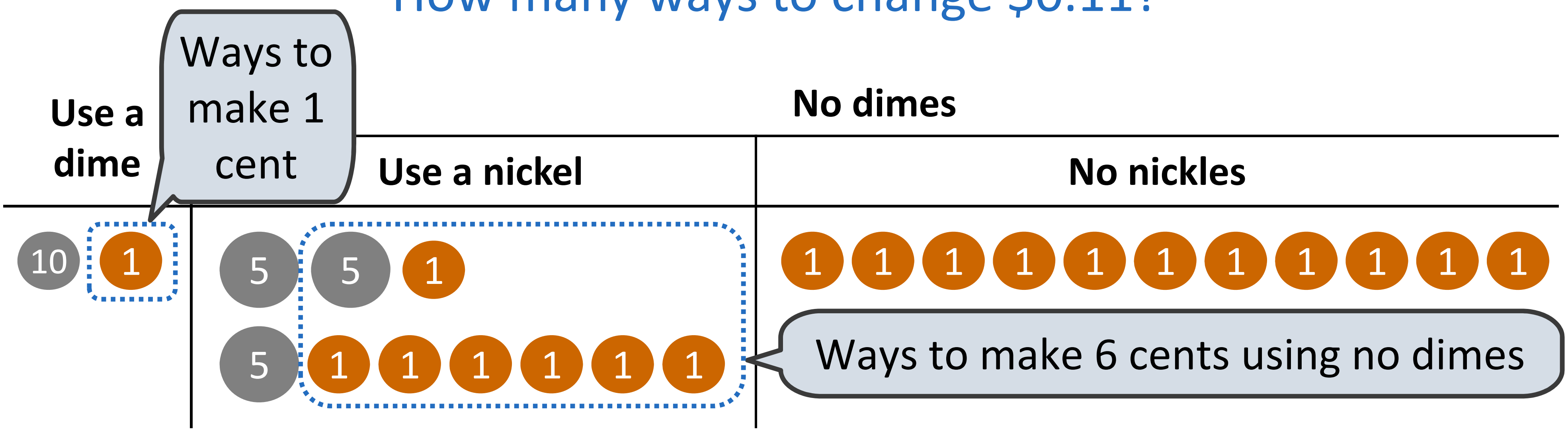
\$1 = 1 half dollar, 1 quarter, 2 dimes, 1 nickel

\$1 = 2 quarters, 2 dimes, 30 pennies

\$1 = 100 pennies

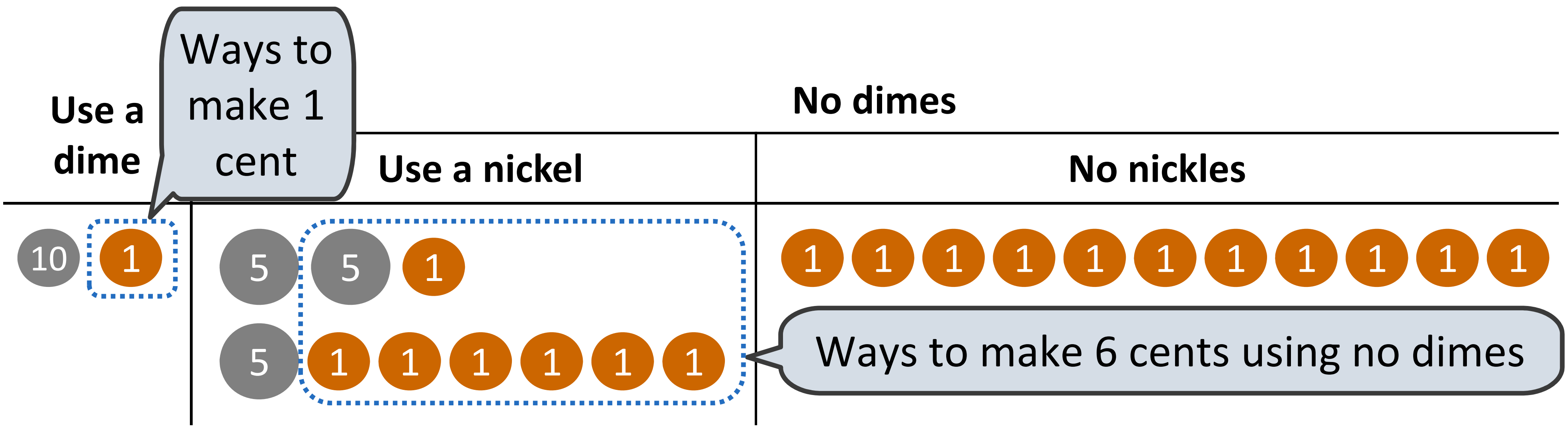
How many ways are there to change a dollar?

How many ways to change \$0.11?



Counting Change Recursively

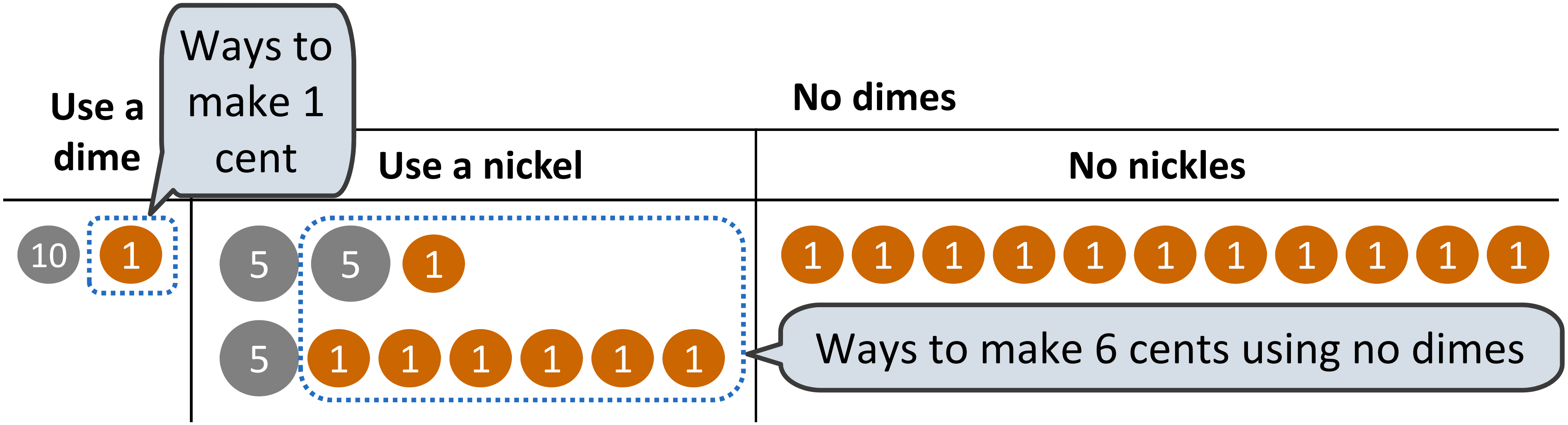
How many ways are there to change a dollar?



Counting Change Recursively

How many ways are there to change a dollar?

The number of ways to change an amount *a* using *n* kinds of coins is:

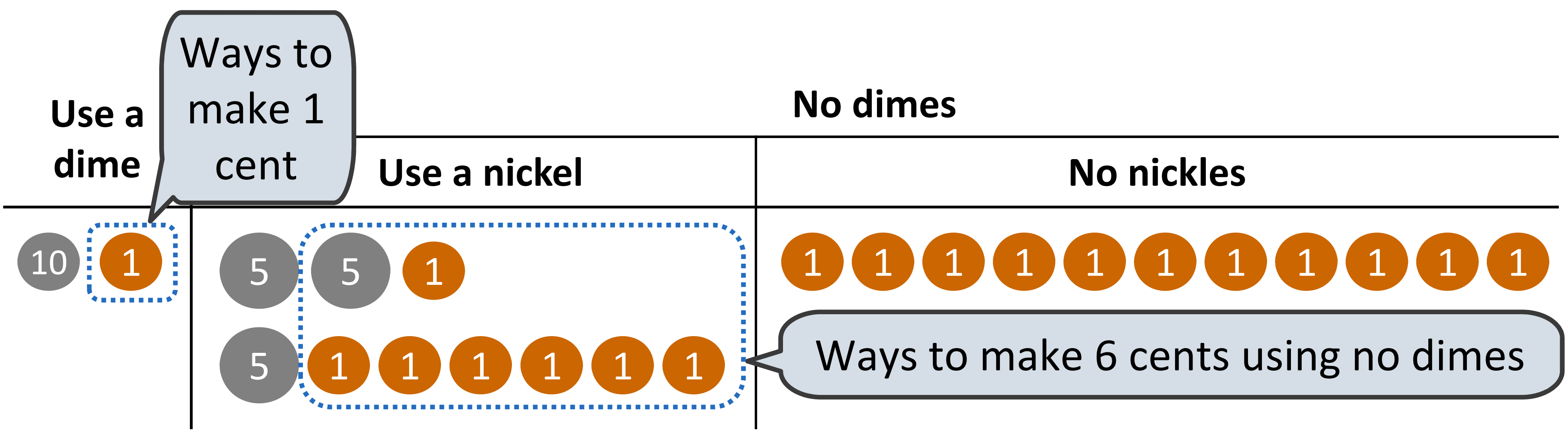


Counting Change Recursively

How many ways are there to change a dollar?

The number of ways to change an amount *a* using *n* kinds of coins is:

1. The number of ways to change *a-d* using all kinds, where *d* is the amount of the first kind of coin

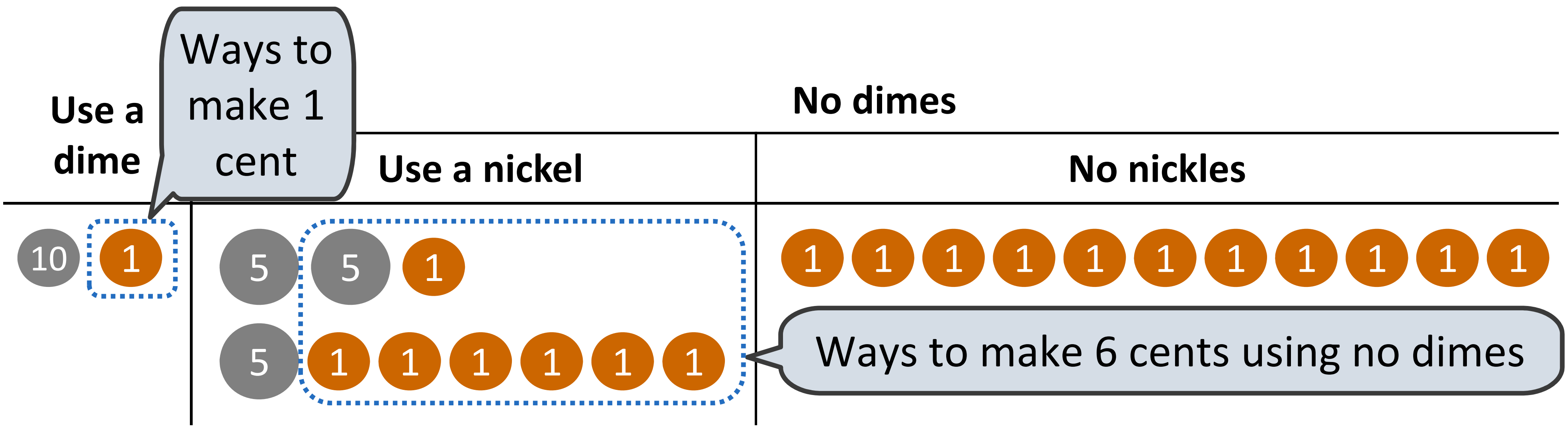


Counting Change Recursively

How many ways are there to change a dollar?

The number of ways to change an amount *a* using *n* kinds of coins is:

1. The number of ways to change *a-d* using all kinds, where *d* is the amount of the first kind of coin
2. The number of ways to change *a* using all but the first kind



Counting Change Recursively

How many ways are there to change a dollar?

The number of ways to change an amount *a* using *n* kinds of coins is:

1. The number of ways to change *a-d* using all kinds, where *d* is the amount of the first kind of coin
2. The number of ways to change *a* using all but the first kind

Counting Change Recursively

How many ways are there to change a dollar?

The number of ways to change an amount *a* using *n* kinds of coins is:

1. The number of ways to change *a-d* using all kinds, where *d* is the amount of the first kind of coin
2. The number of ways to change *a* using all but the first kind

```
def count_change(a, d):
    if a == 0:
        return 1
    if a < 0 or d == 0:
        return 0
    return (count_change(a-d, d) +
            count_change(a, next_coin(d)))
```

Counting Change Recursively

How many ways are there to change a dollar?

The number of ways to change an amount *a* using *n* kinds of coins is:

1. The number of ways to change *a-d* using all kinds, where *d* is the amount of the first kind of coin
2. The number of ways to change *a* using all but the first kind

```
def count_change(a, d):
    if a == 0:
        return 1
    if a < 0 or d == 0:
        return 0
    return (count_change(a-d, d) +
            count_change(a, next_coin(d)))
```

One way to make no amount

Counting Change Recursively

How many ways are there to change a dollar?

The number of ways to change an amount *a* using *n* kinds of coins is:

1. The number of ways to change *a-d* using all kinds, where *d* is the amount of the first kind of coin
2. The number of ways to change *a* using all but the first kind

```
def count_change(a, d):
    if a == 0:
        return 1
    if a < 0 or d == 0:
        return 0
    return (count_change(a-d, d) +
            count_change(a, next_coin(d)))
```

One way to make no amount

Can't make negative amount, or any amount with no coins

Counting Change Recursively

How many ways are there to change a dollar?

The number of ways to change an amount *a* using *n* kinds of coins is:

1. The number of ways to change *a-d* using all kinds, where *d* is the amount of the first kind of coin
2. The number of ways to change *a* using all but the first kind

```
def count_change(a, d):
    if a == 0:
        return 1
    if a < 0 or d == 0:
        return 0
    return (count_change(a-d, d) +
            count_change(a, next_coin(d)))
```

One way to make no amount

Can't make negative amount, or any amount with no coins

Functional abstraction to get next coin