при поддержке
{а} вижн

Лекция 7

Перевод и адаптация материалов Джона ДеНиро (John DeNero). Используется с разрешения автора.

Порядок рекурсивных вызовов

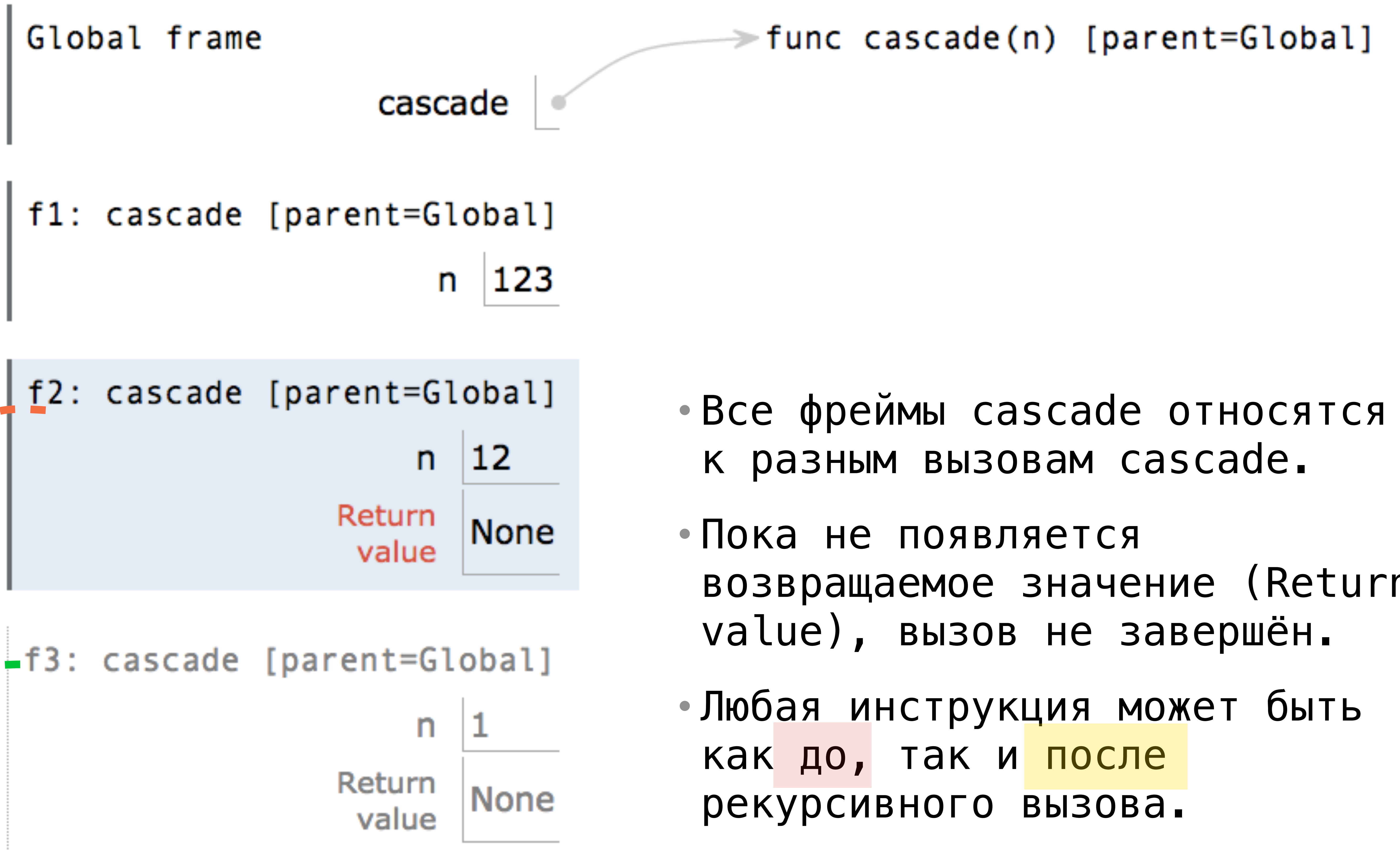
Функция «Каскад»

(Пример)

```
1  def cascade(n):
2      if n < 10:
3          print(n)
4      else:
5          print(n)
6          cascade(n//10)
7          print(n)
8
9  cascade(123)
```

Program output:

```
123
12
1
12
```



- Все фреймы cascade относятся к разным вызовам cascade.
- Пока не появляется возвращаемое значение (Return value), вызов не завершён.
- Любая инструкция может быть как до, так и после рекурсивного вызова.

Интерактивная диаграмма

Два определения cascade

(Пример)

```
def cascade(n):
    if n < 10:
        print(n)
    else:
        print(n)
        cascade(n//10)
        print(n)
```

```
def cascade(n):
    print(n)
    if n >= 10:
        cascade(n//10)
        print(n)
```

- Из двух логически одинаковых реализаций обычно лучше та, что меньше.
- В данном случае, более длинная — более понятная (хотя бы для меня).
- При изучении рекурсивных функций, ставь простые случаи в начало.
- Обе реализации рекурсивны, хотя только первая имеет каноническую структуру.

Пример: обратный каскад

Обратный каскад

Напиши функцию, которая выводит обратный каскад:

```
1
12
123
1234
123
12
1
```

```
def inverse_cascade(n):
    grow(n)
    print(n)
    shrink(n)


def f_then_g(f, g, n):
    if n:
        f(n)
        g(n)


grow =   lambda n: f_then_g(                    )
shrink = lambda n: f_then_g(                    )
```

Древовидная рекурсия

Древовидная рекурсия

Древовидная рекурсия появляется в случае, когда из тела рекурсивной функции делается более одного рекурсивного вызова.

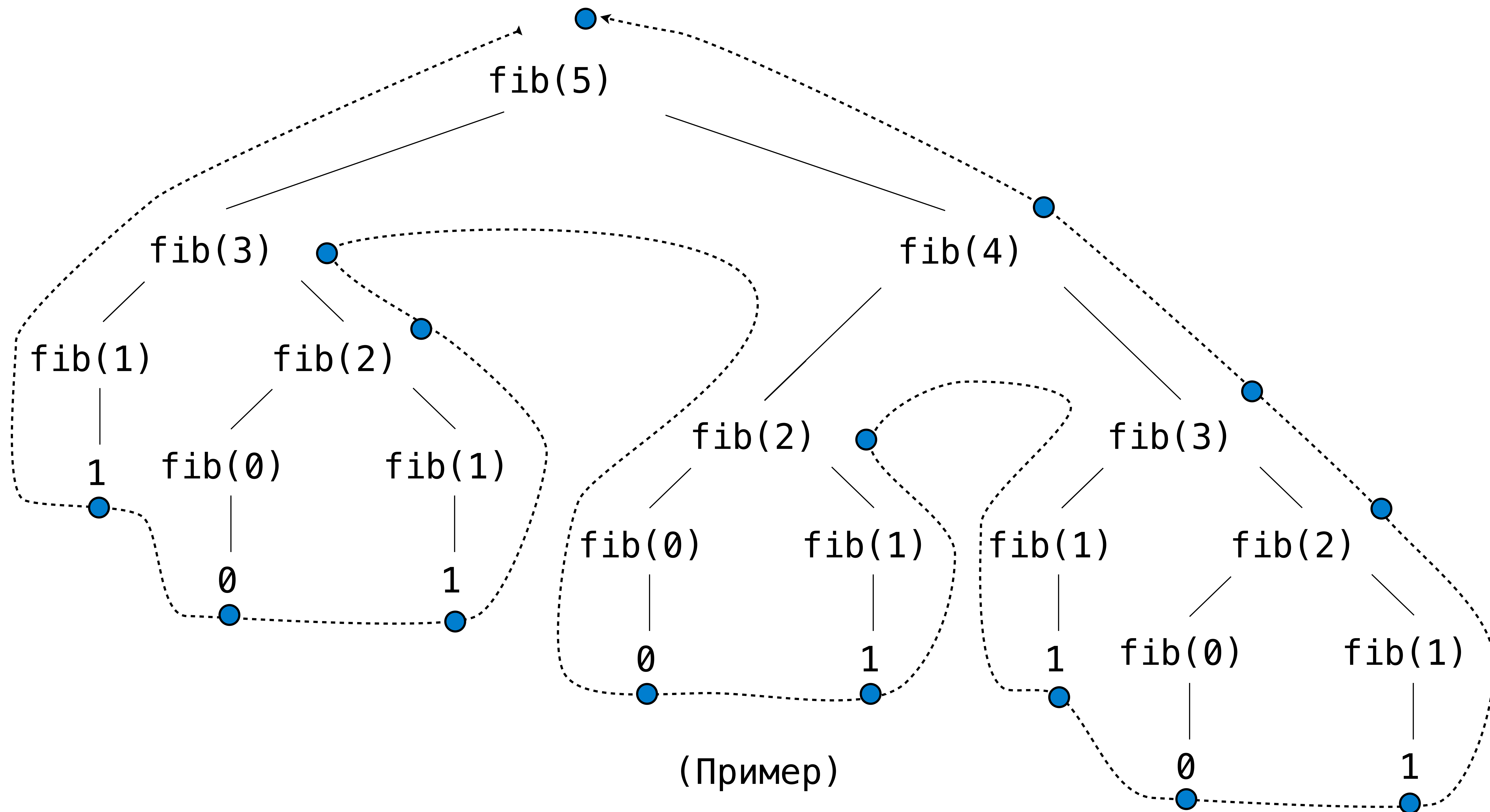
n:	0,	1,	2,	3,	4,	5,	6,	7,	8,	... ,	35
fib(n):	0,	1,	1,	2,	3,	5,	8,	13,	21,	... ,	9,227,465

```
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

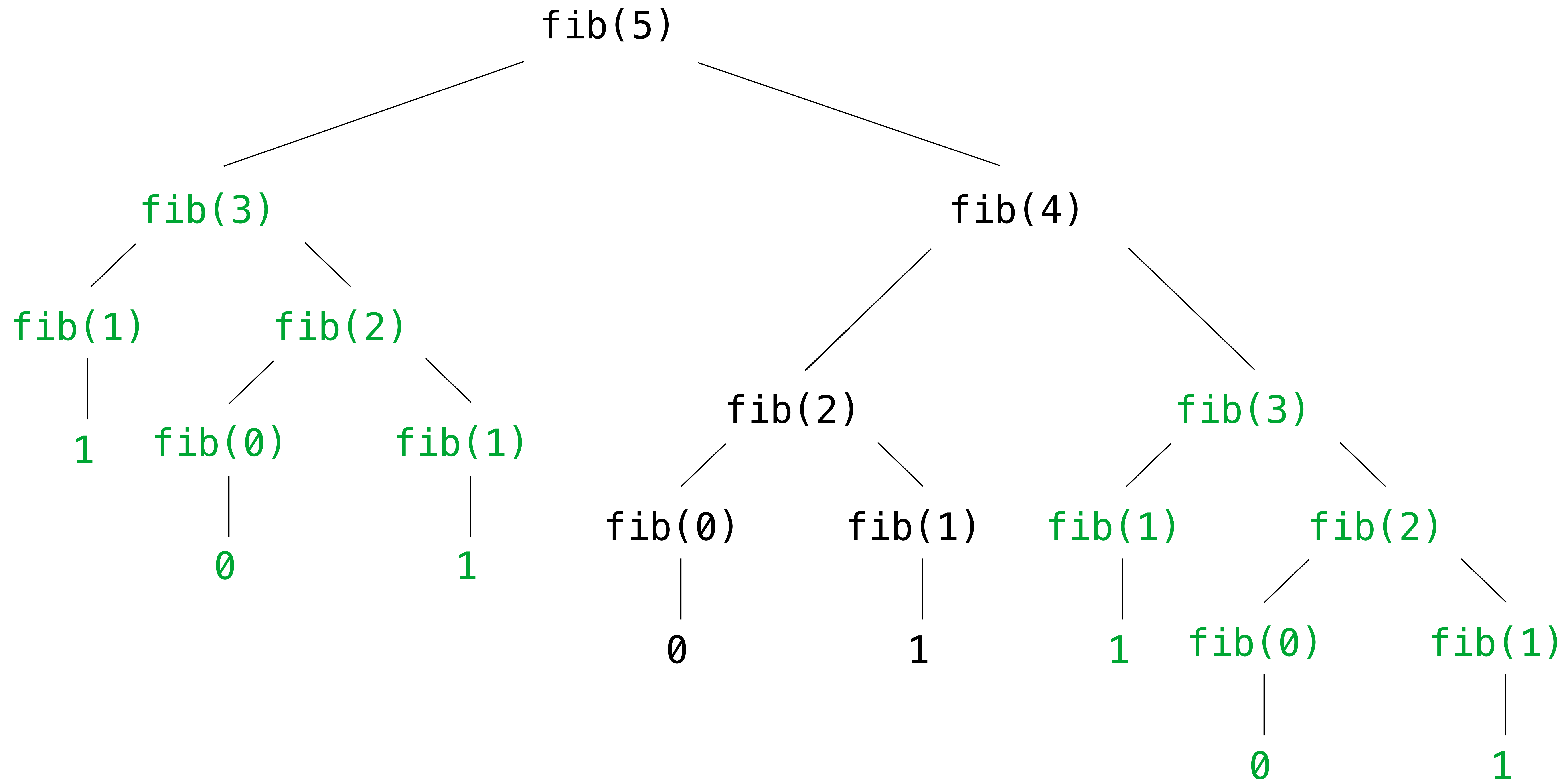
Древовидная рекурсия

Процесс вычисление fib(5) можно изобразить в виде дерева.



Повторения в древовидной рекурсии

Этот процесс сильно избыточен; fib вызывается с одинаковыми аргументами множество раз.



Можно существенно ускорить этот процесс запоминая промежуточные результаты.

Пример: подсчёт разбиений

Подсчет разбиений

Разбиением называется представление натурального числа n в виде суммы натуральных слагаемых, а сами слагаемые — частями разбиения. Порядок слагаемых не играет роли. Будем записывать разбиения перечисляя их части в возрастающем порядке. Пусть части разбиения не превышают m.

```
count_partitions(6, 4)
```

2 + 4 = 6

1 + 1 + 4 = 6

3 + 3 = 6

1 + 2 + 3 = 6

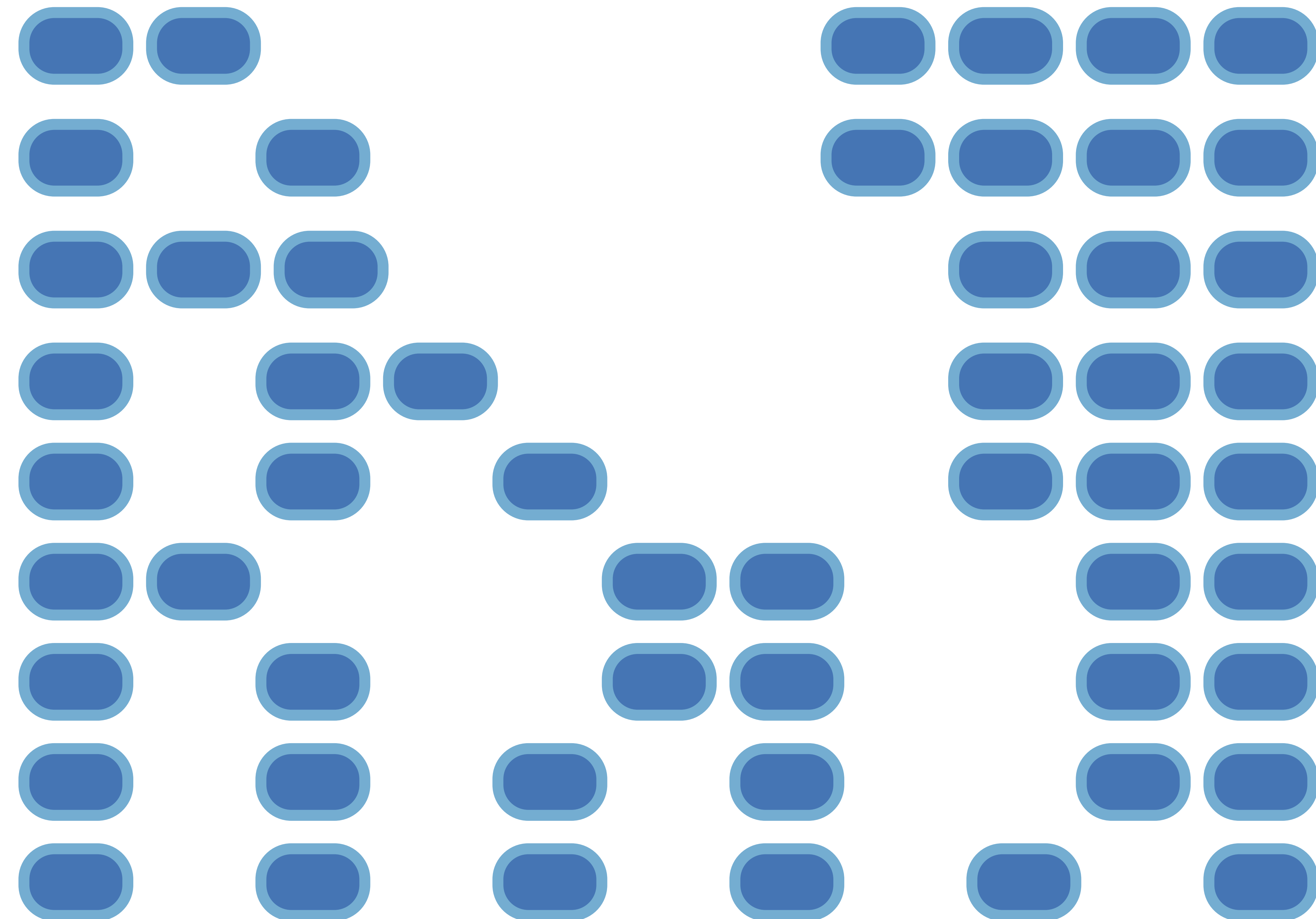
1 + 1 + 1 + 3 = 6

2 + 2 + 2 = 6

1 + 1 + 2 + 2 = 6

1 + 1 + 1 + 1 + 2 = 6

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

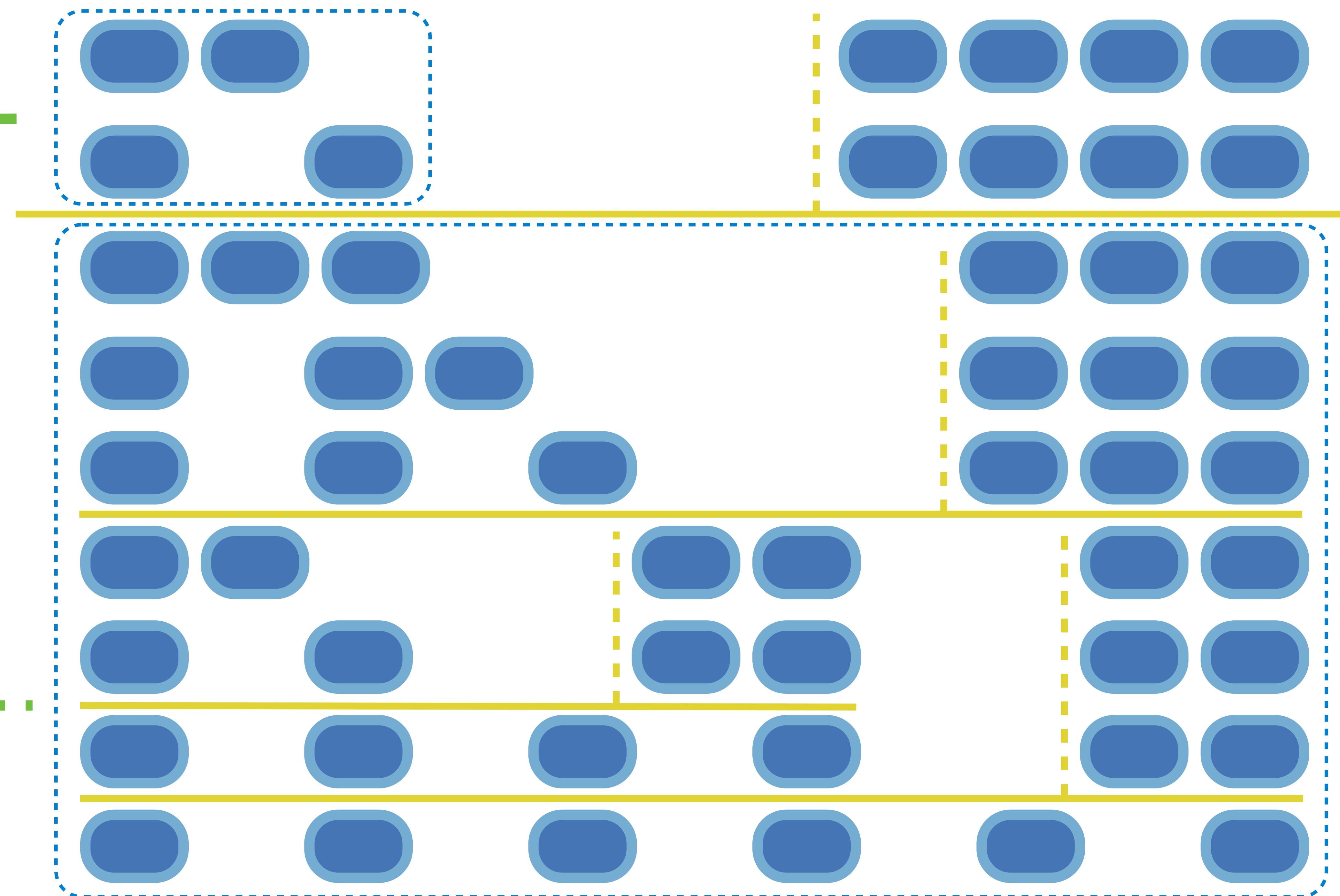


Подсчет разбиений

Разбиением называется представление натурального числа n в виде суммы натуральных слагаемых, а сами слагаемые — частями разбиения. Порядок слагаемых не играет роли. Будем записывать разбиения перечисляя их части в возрастающем порядке. Пусть часть разбиения не превышает m.

count_partitions(6, 4)

- Рекурсивная декомпозиция: поиск более простых случаев.
- Исследуем две возможности:
  - Хотя бы одна часть рана 4
  - Нет ни одной части равной 4
- Решаем две более простые задачи:
  - count_partitions(2, 4)
  - count_partitions(6, 3)
- Древовидная рекурсия зачастую предполагает исследование различных вариантов.



Подсчет разбиений

Разбиением называется представление натурального числа n в виде суммы натуральных слагаемых, а сами слагаемые — частями разбиения. Порядок слагаемых не играет роли. Будем записывать разбиения перечисляя их части в возрастающем порядке. Пусть часть разбиения не превышает m.

- Рекурсивная декомпозиция: поиск более простых случаев.
- Исследуем две возможности:
  - Хотя бы одна часть рана 4
  - Нет ни одной части равной 4
- Решаем две более простые задачи:
  - count_partitions(2, 4)
  - count_partitions(6, 3)
- Древовидная рекурсия зачастую предполагает исследование различных вариантов.

```
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
```

(Пример)

Интерактивная диаграмма