CS 61A
Discussion 2

Recursion

Raymond Chan
Discussion 121
UC Berkeley

Announcements

- Project 1 Hog due tonight
  - tinyurl.com/61a-unstuck
- Guerrilla Section on Recursion 2/7 10am-noon
- CSM small group tutoring sections sign ups
  - csmscheduler.herokuapp.com

Recursion

- A recursive function is a function that calls itself.
- Three common steps
  - Figure our your base case(s)
  - Make the problem smaller and make a recursive call with that simpler argument
  - Use your recursive call to solve the full problem

Recursion

- Base cases are there to stop the recursion.
- No base case —> continue making recursive calls forever

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

Recursion

- Find a smaller problem for the recursive call.
- Make sure the problem is getting smaller **toward** the base case.
- Call the recursive function with this smaller argument.

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

Recursion

- Take the ***leap of faith*** and trust that your recursive function is correct on the smaller argument.

- Knowing that the recursive call returns what you want, how can you solve the bigger problem?

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

Recursion

factorial(5)

Recursion

factorial(5)

↓

5 * factorial(4)

Recursion

factorial(5)

↓

5 * factorial(4)

↓

4 * factorial(3)

Recursion

factorial(5)

↓

5 * factorial(4)

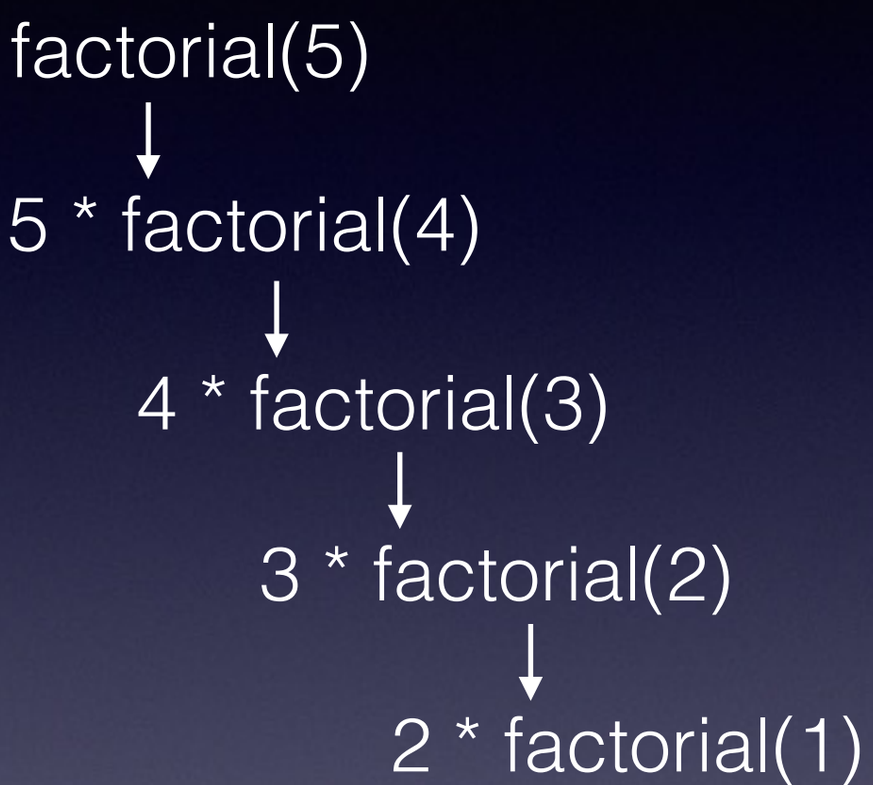
↓

4 * factorial(3)

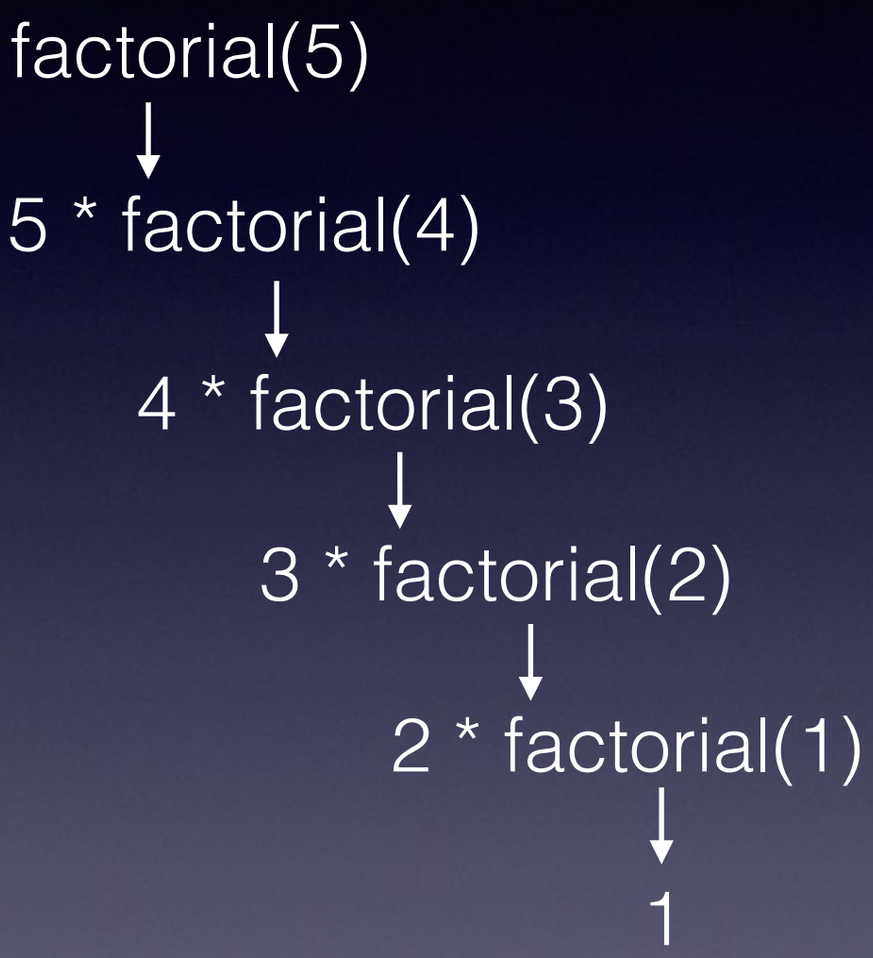
↓

3 * factorial(2)

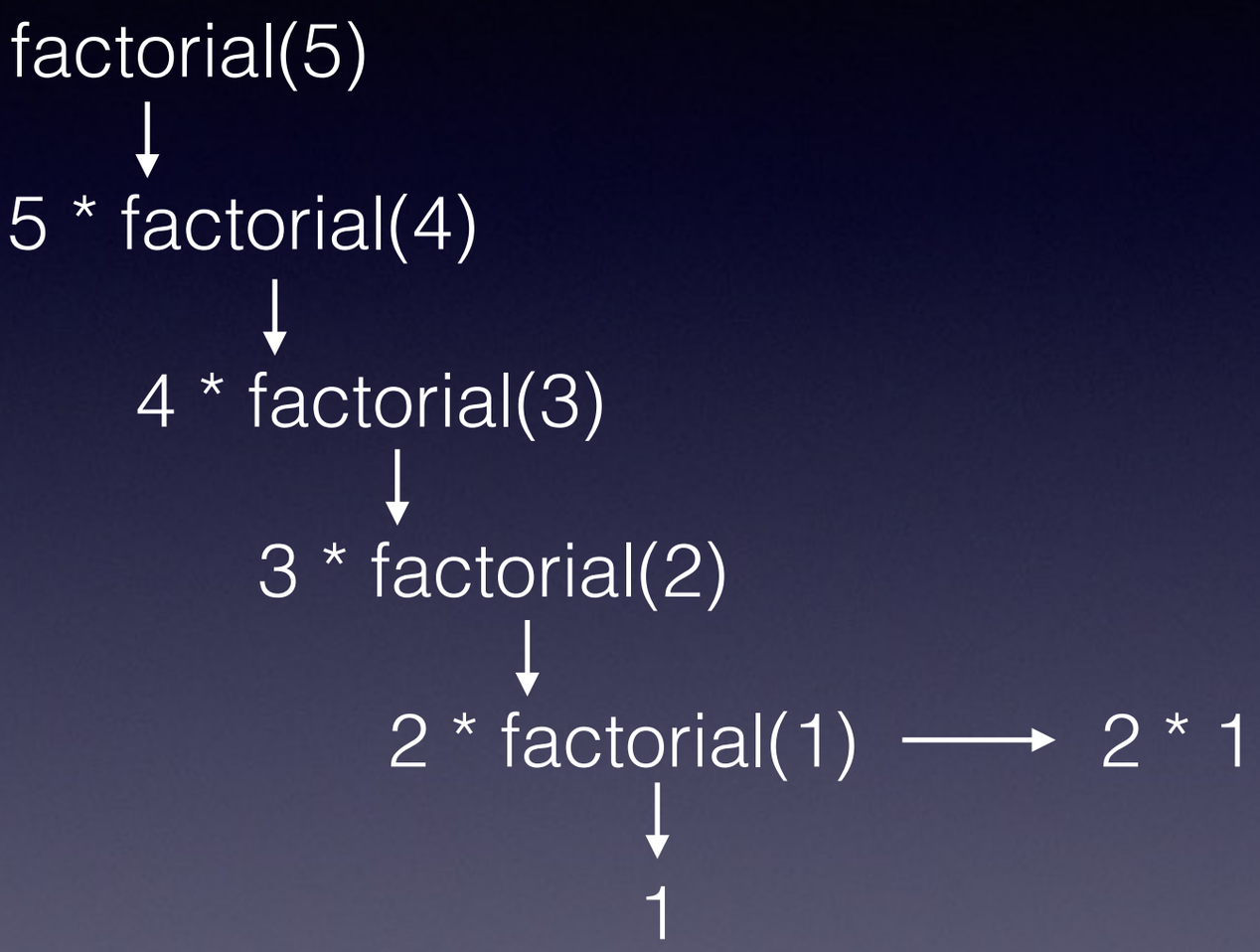
Recursion


factorial(5)
5 * factorial(4)
4 * factorial(3)
3 * factorial(2)
2 * factorial(1)

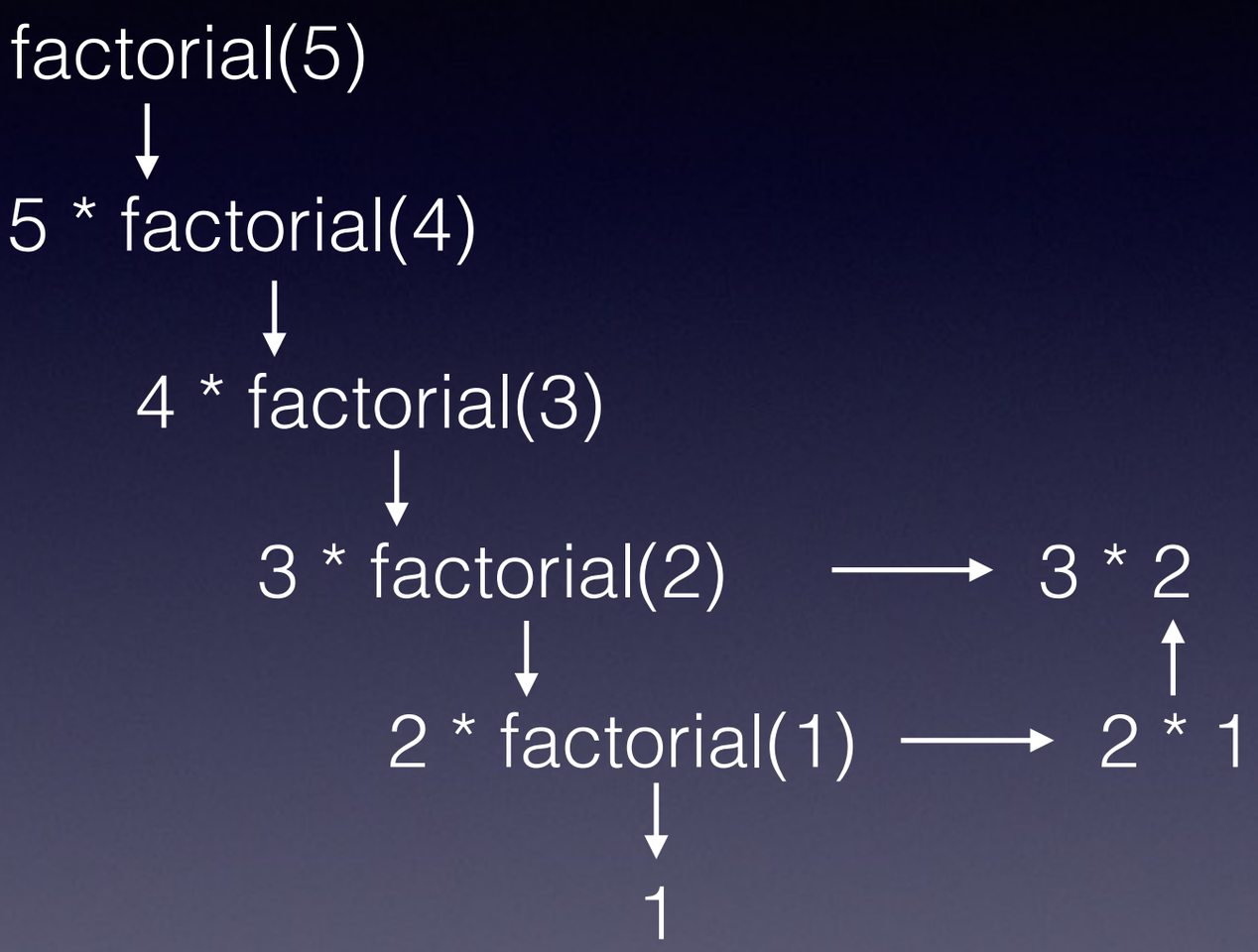
Recursion


factorial(5)
5 * factorial(4)
4 * factorial(3)
3 * factorial(2)
2 * factorial(1)
1

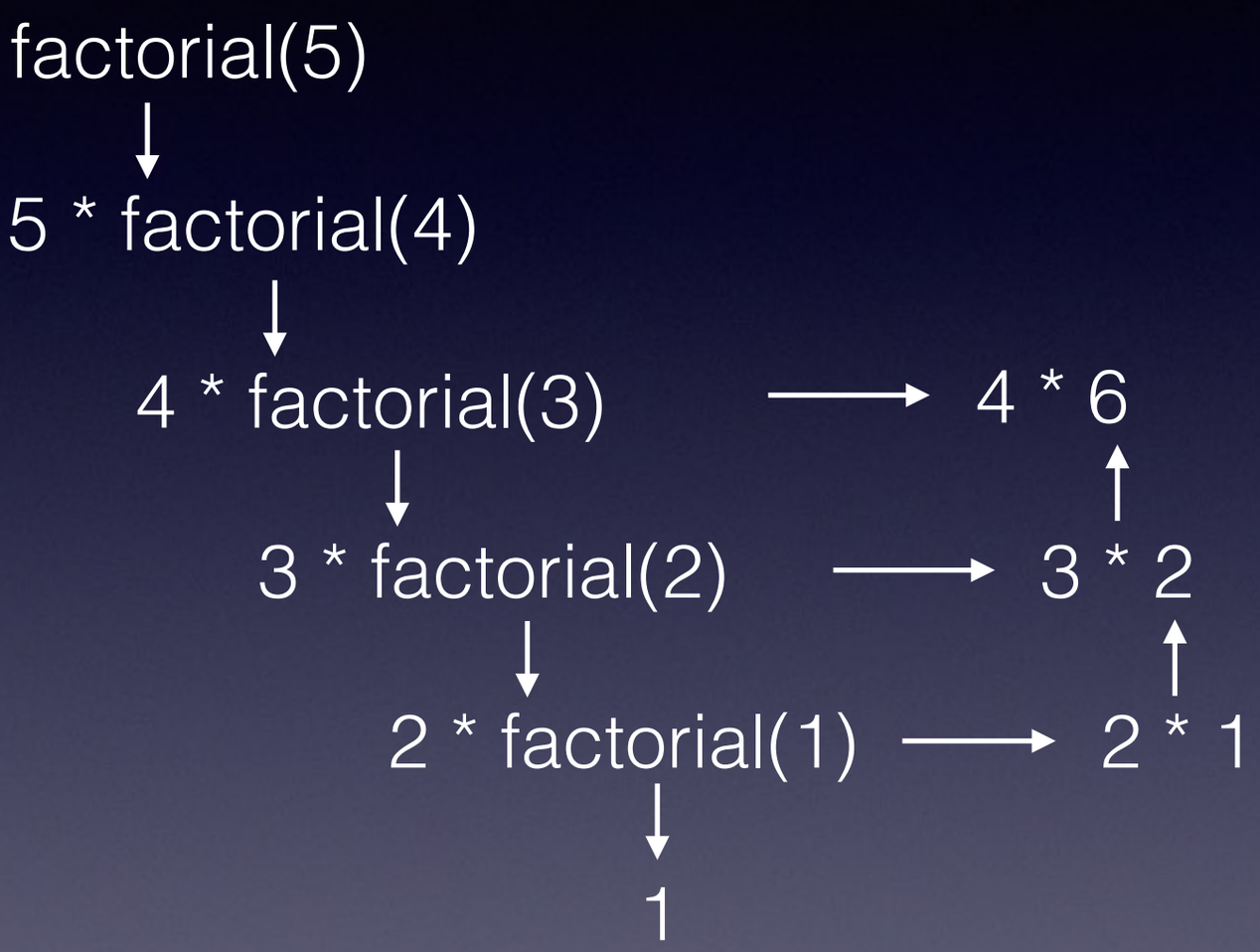
Recursion


factorial(5)
5 * factorial(4)
4 * factorial(3)
3 * factorial(2)
2 * factorial(1)
2 * 1
1

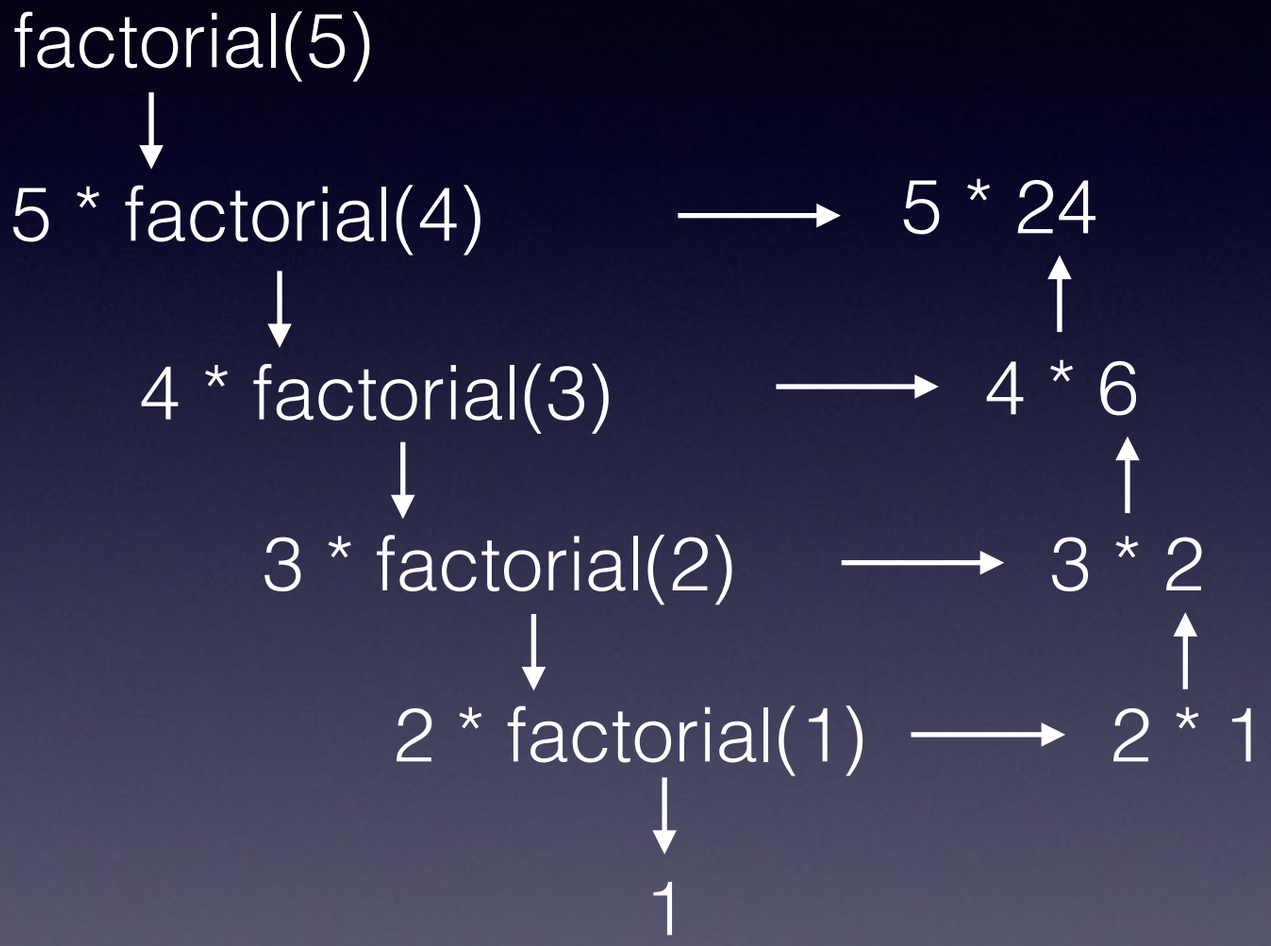
Recursion


factorial(5)
5 * factorial(4)
4 * factorial(3)
3 * factorial(2)
3 * 2
2 * factorial(1)
2 * 1
1

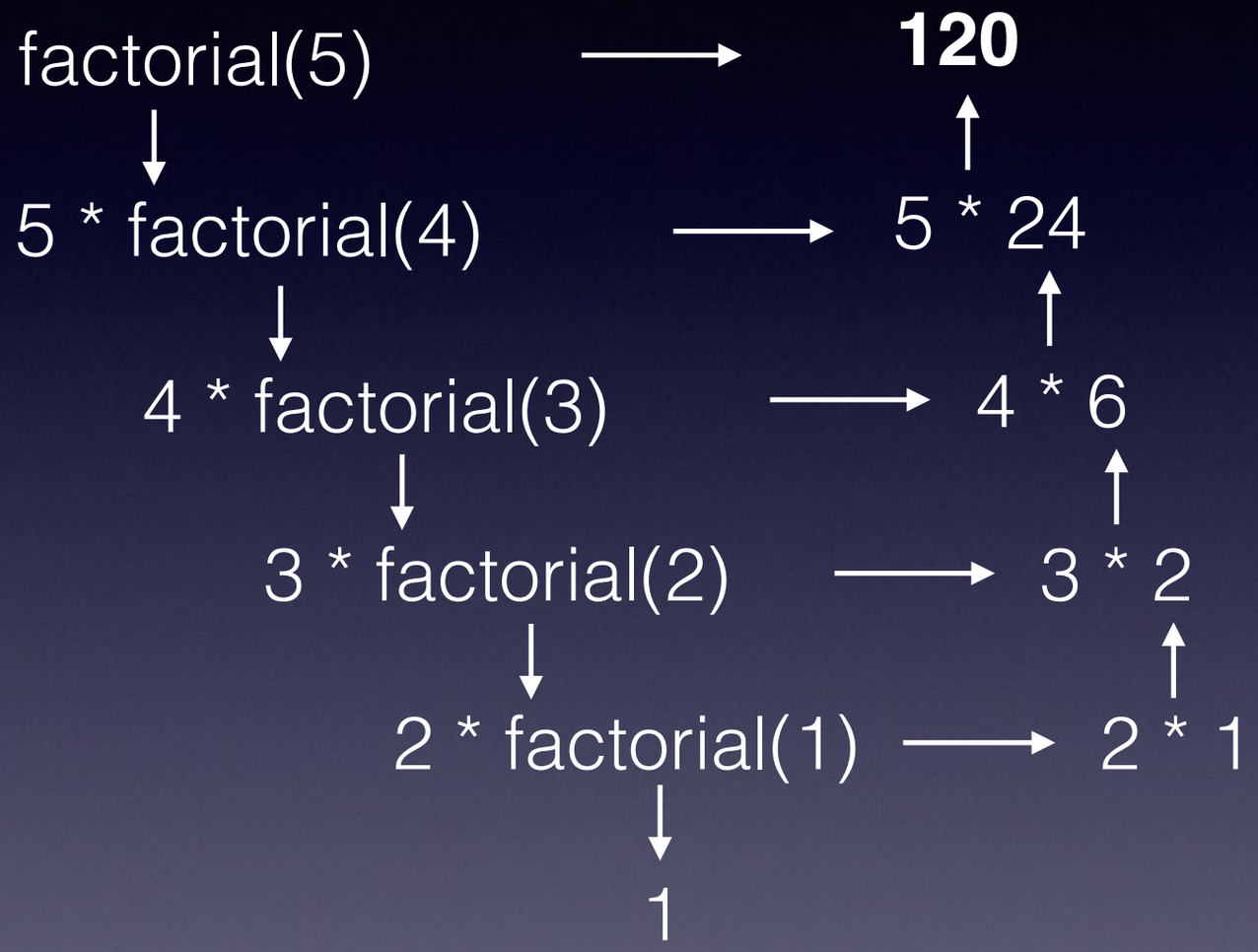
Recursion


factorial(5)
5 * factorial(4)
4 * factorial(3)
4 * 6
3 * factorial(2)
3 * 2
2 * factorial(1)
2 * 1
1

Recursion


factorial(5)
5 * factorial(4)
5 * 24
4 * factorial(3)
4 * 6
3 * factorial(2)
3 * 2
2 * factorial(1)
2 * 1
1

Recursion


factorial(5)
5 * factorial(4)
4 * factorial(3)
3 * factorial(2)
2 * factorial(1)
1
120
5 * 24
4 * 6
3 * 2
2 * 1

Tree Recursion

- Recursive functions that make more than one recursive call in its recursive case.

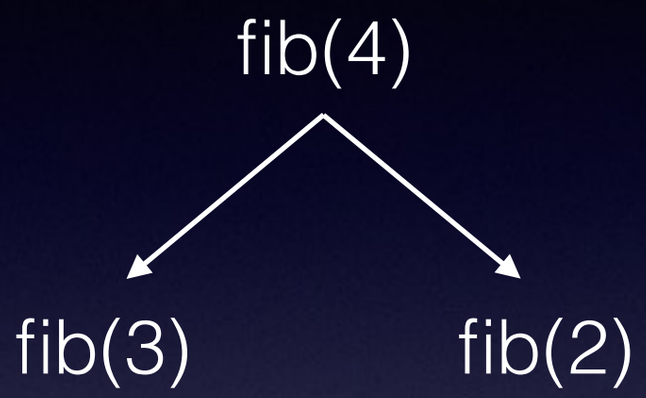
- Example: fibonacci sequence

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

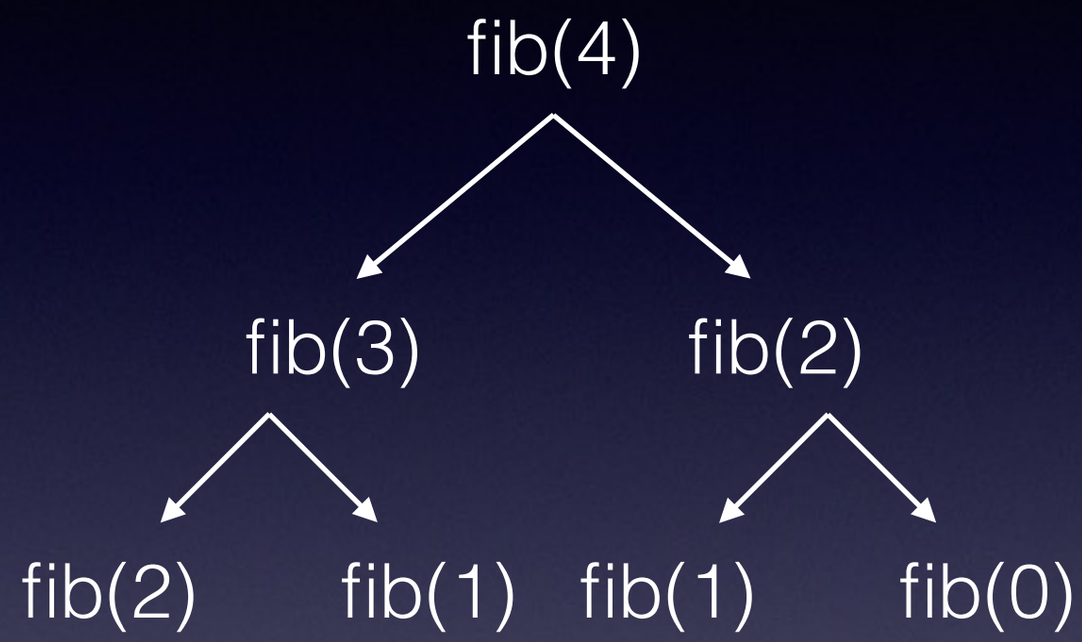
Tree Recursion

fib(4)

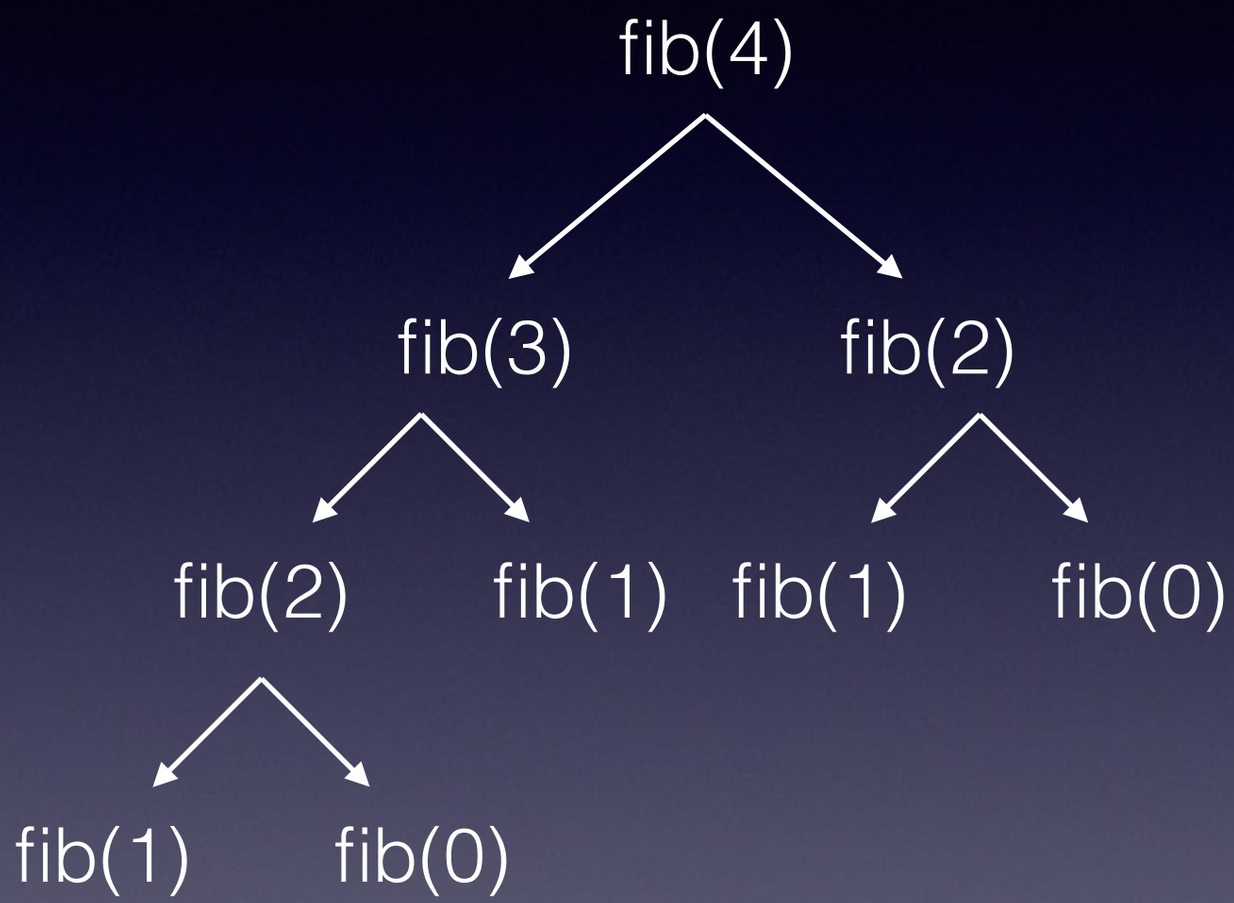
Tree Recursion



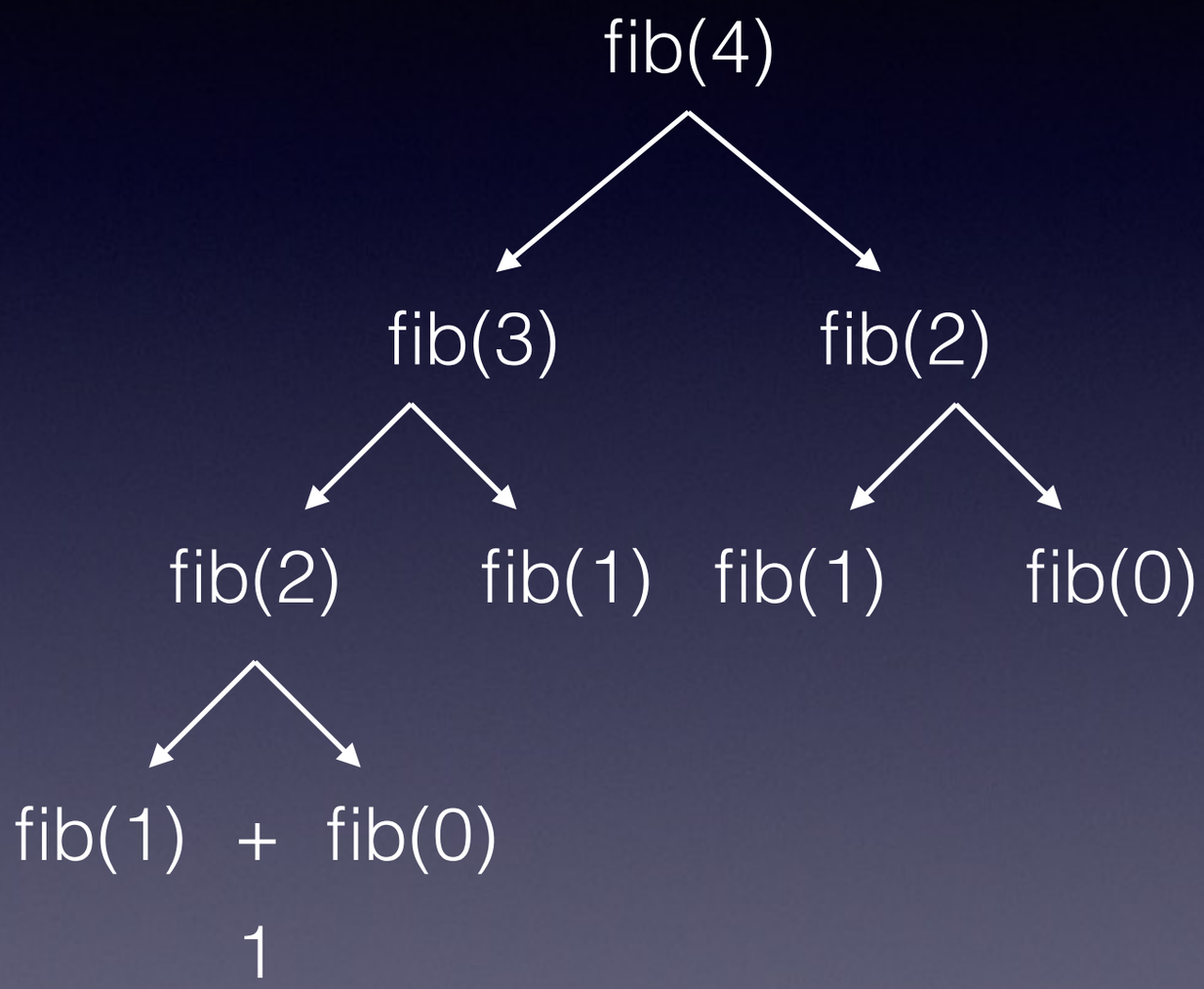
Tree Recursion



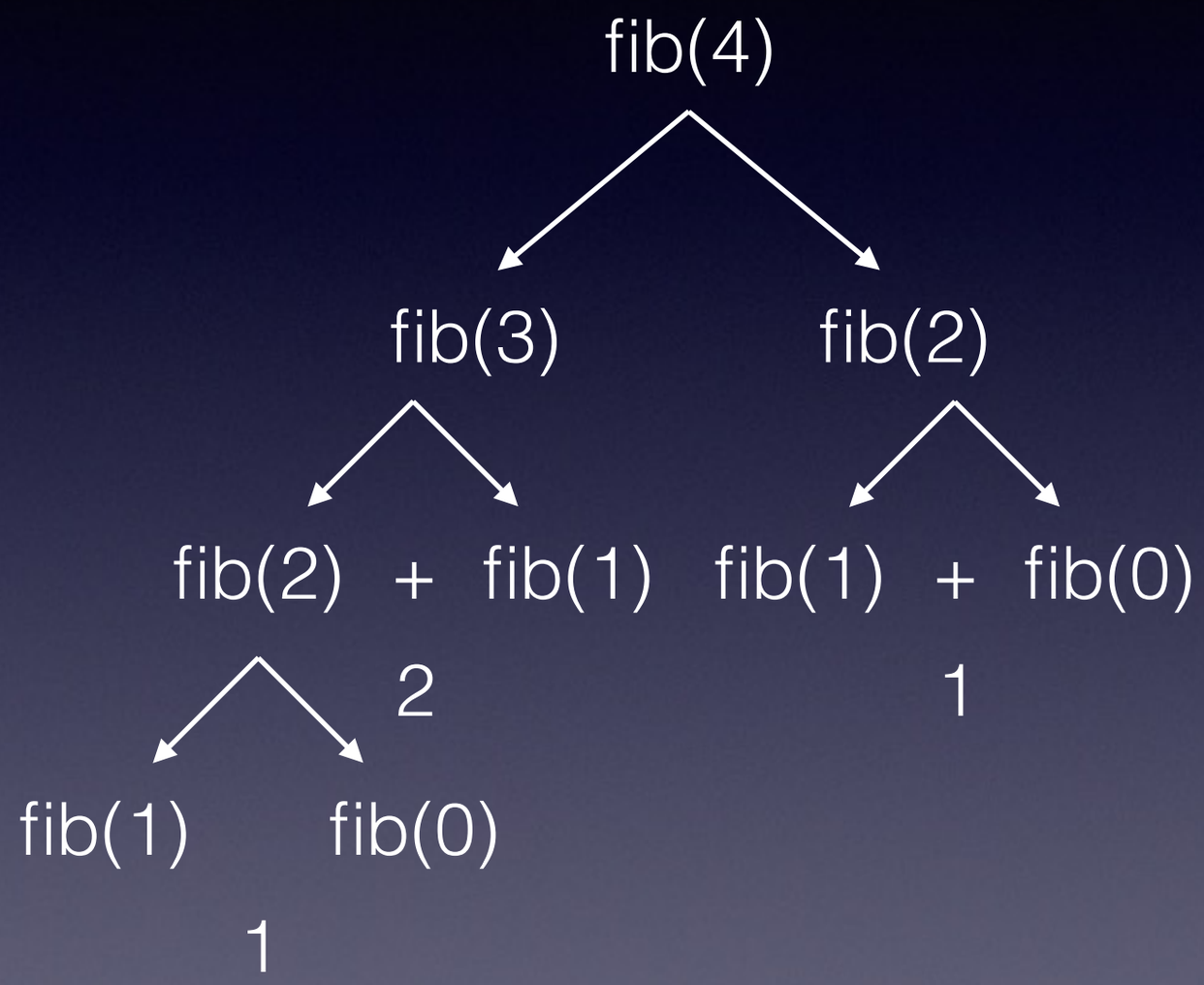
Tree Recursion



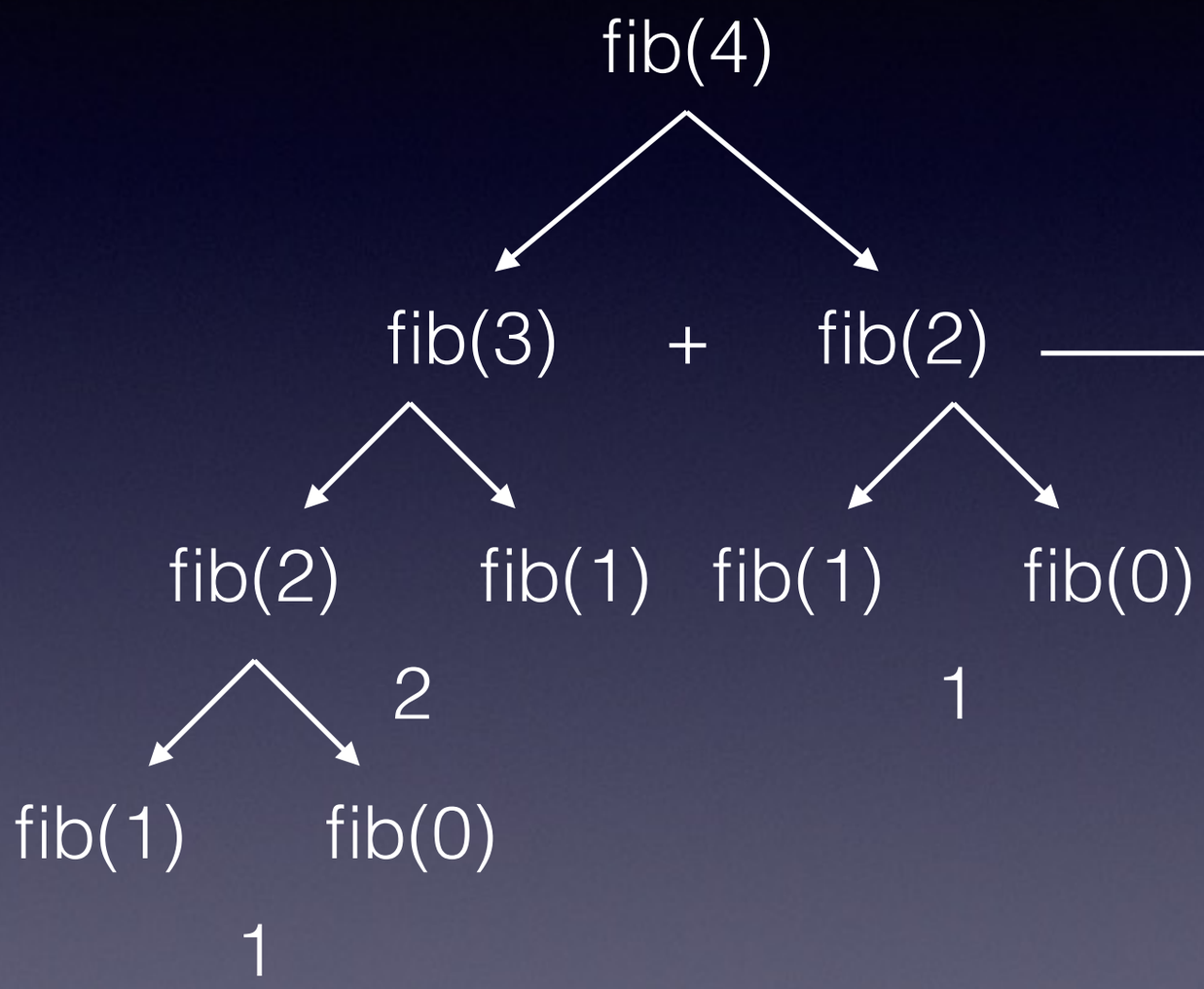
Tree Recursion



Tree Recursion



Tree Recursion



Worksheet

- 2.1 Cool recursion questions!
  - q1 - q2
- Tree recursion
  - q1

Recap

- Environment diagrams allow us to keep track of a variables and their values.
- Recursion functions call themselves.
- Tree recursive functions call themselves multiple times from one frame.