Nested function parenting

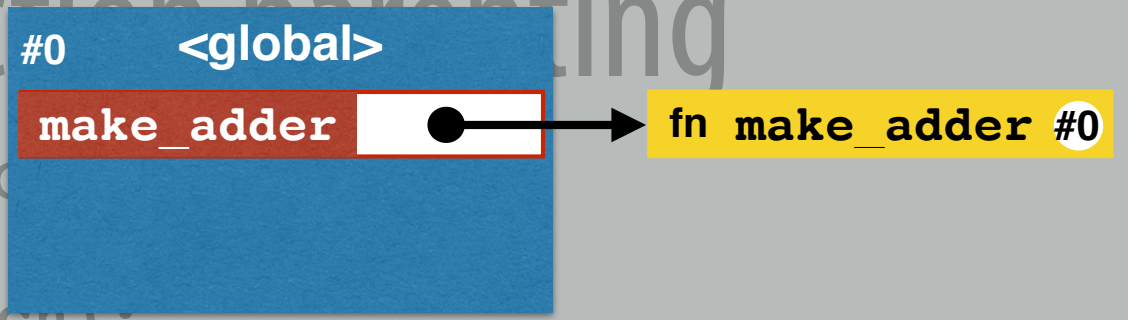
What happens when this script is executed?

```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhowmuch
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

Nested function parenting

What happens when this s

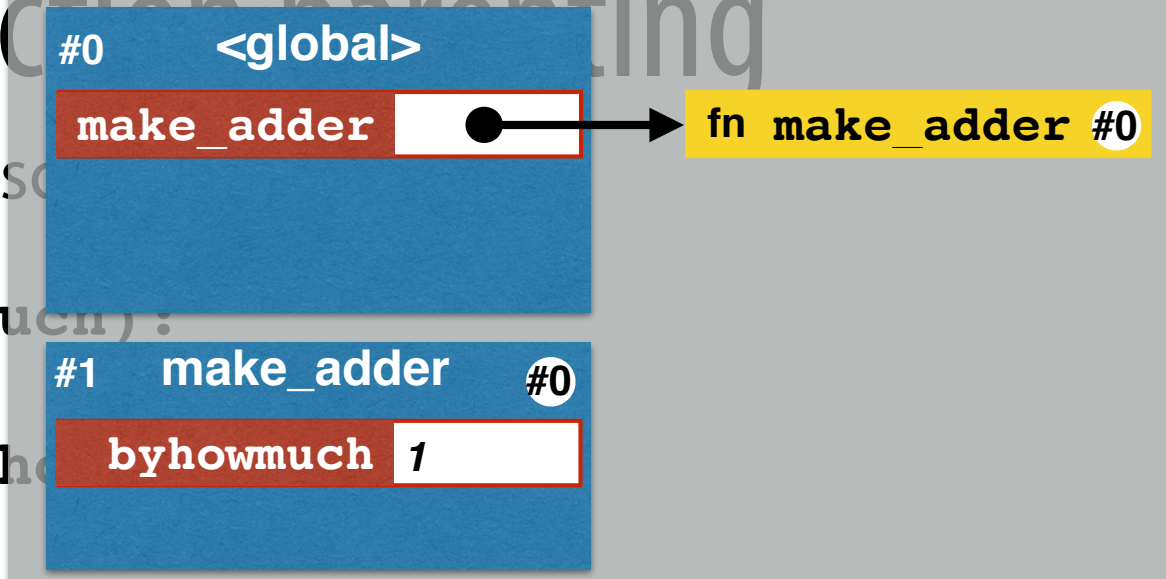
```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhowmuch
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```



Nested function parenting

What happens when this s

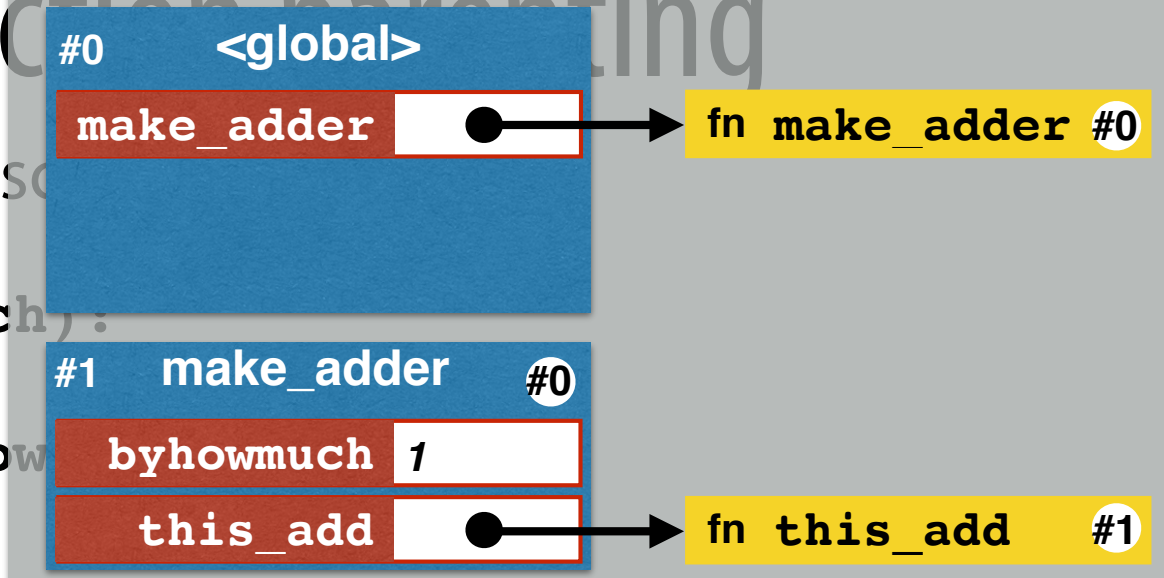
```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byh
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```



Nested function parenting

What happens when this s

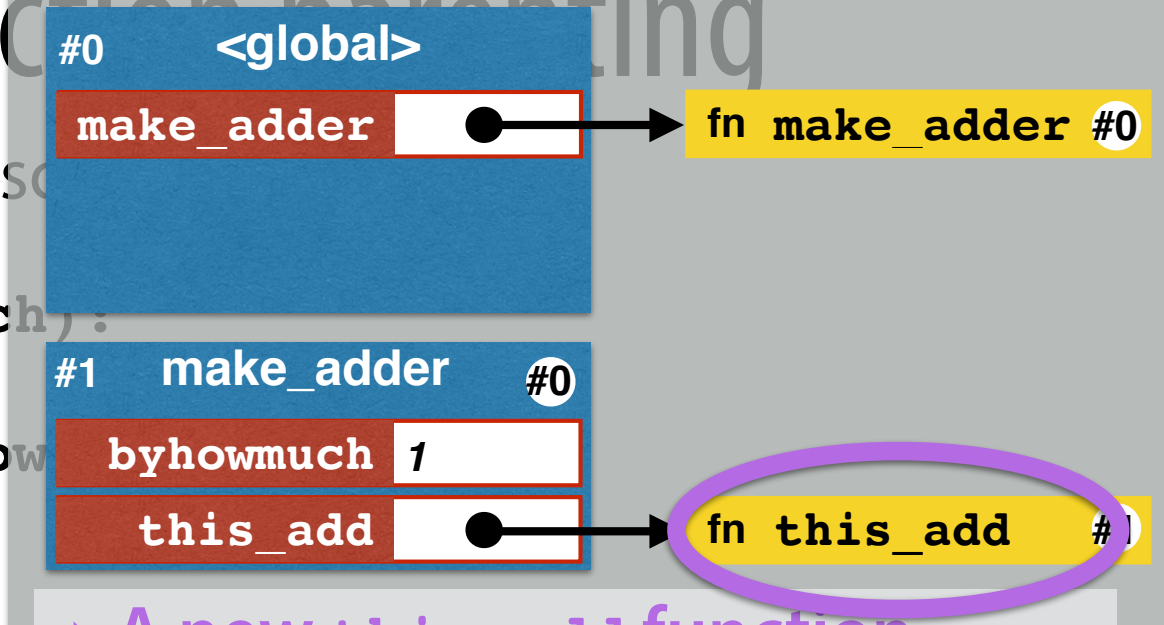
```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhow
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```



Nested function parenting

What happens when this s

```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhow
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

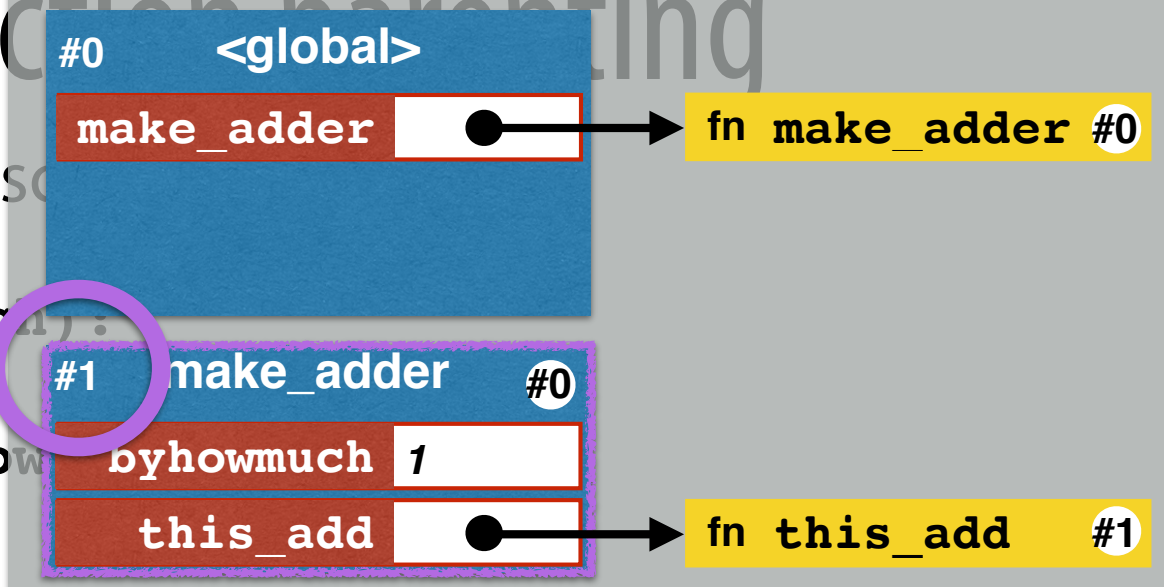


➡ A new `this_add` function object is constructed during this call.

Nested function parenting

What happens when this s

```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhow
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

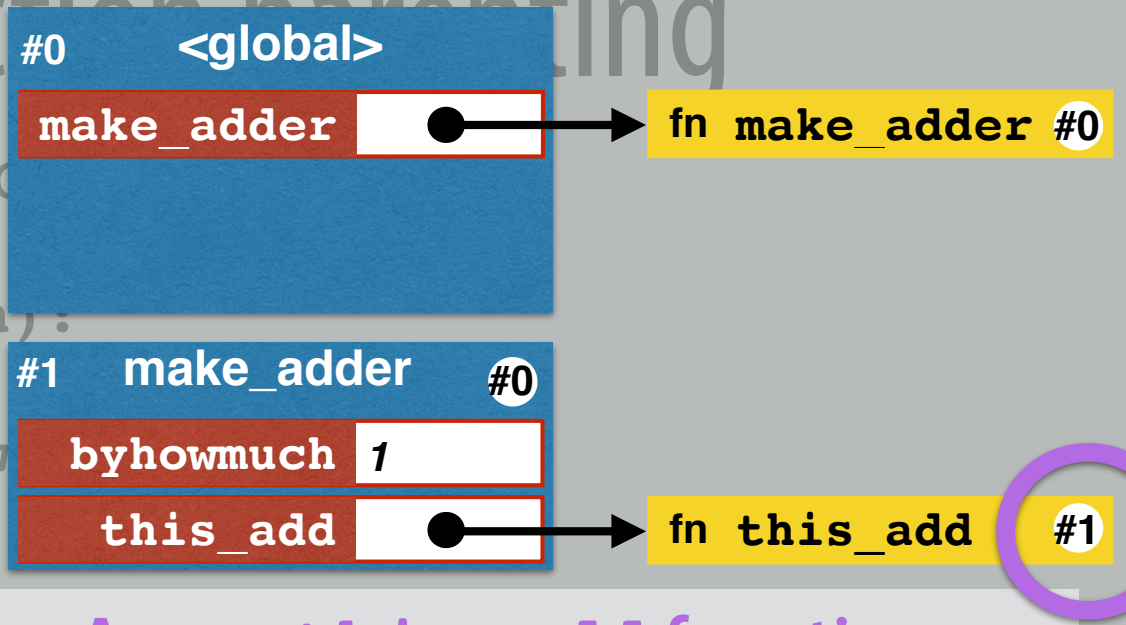


- A new `this_add` function object is constructed during this call.
- The *def* is executed within the context of frame #1.

Nested function parenting

What happens when this s

```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhow
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

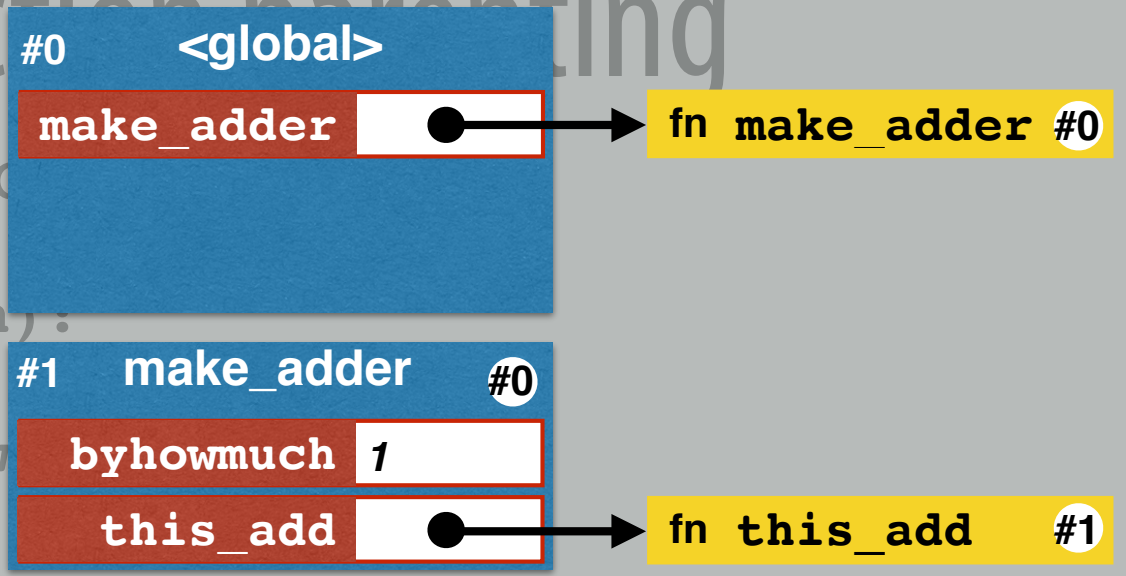


- A new `this_add` function object is constructed during this call.
- The *def* is executed within the context of frame #1.
- That new function's parent frame is set to #1.

Nested function parenting

What happens when this s

```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhow
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

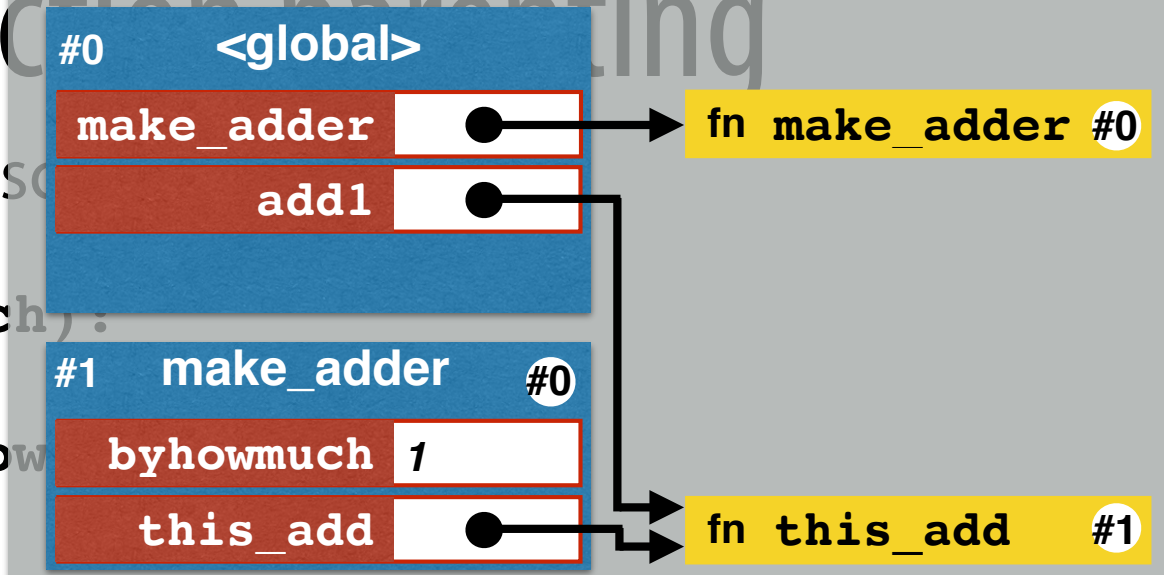


- A new `this_add` function object is constructed during this call.
- The *def* is executed within the context of frame #1.
- That new function's parent frame is set to #1.

Nested function parenting

What happens when this s

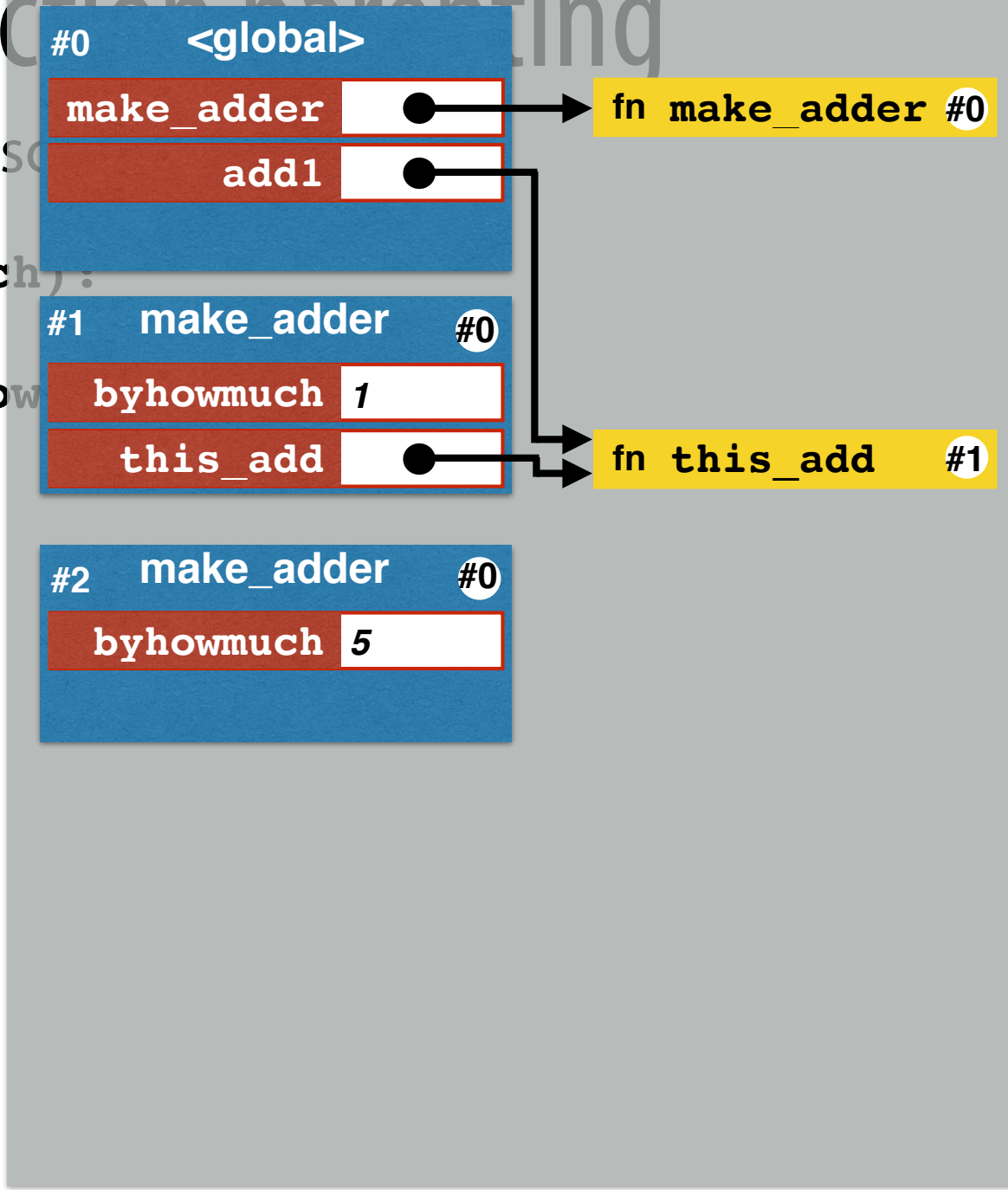
```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhow
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```



Nested function parenting

What happens when this s

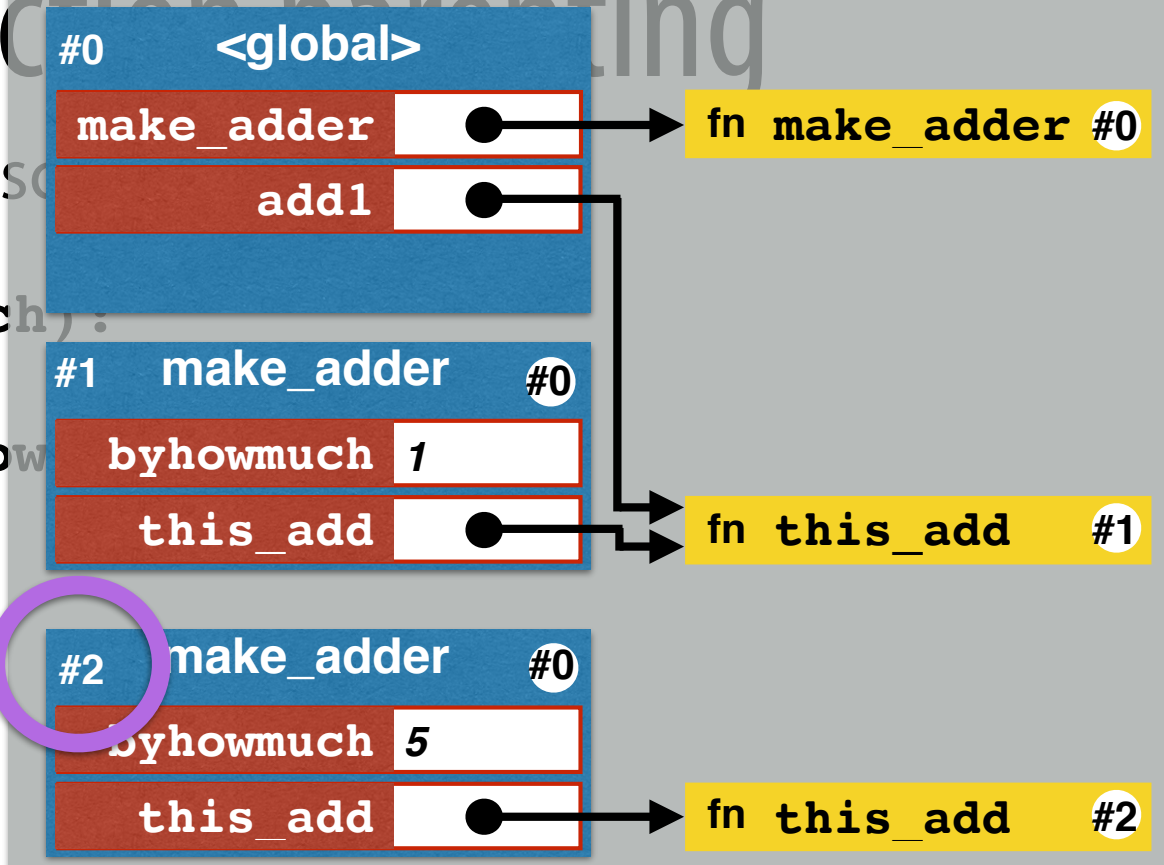
```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhow
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```



Nested function parenting

What happens when this s

```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhow
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

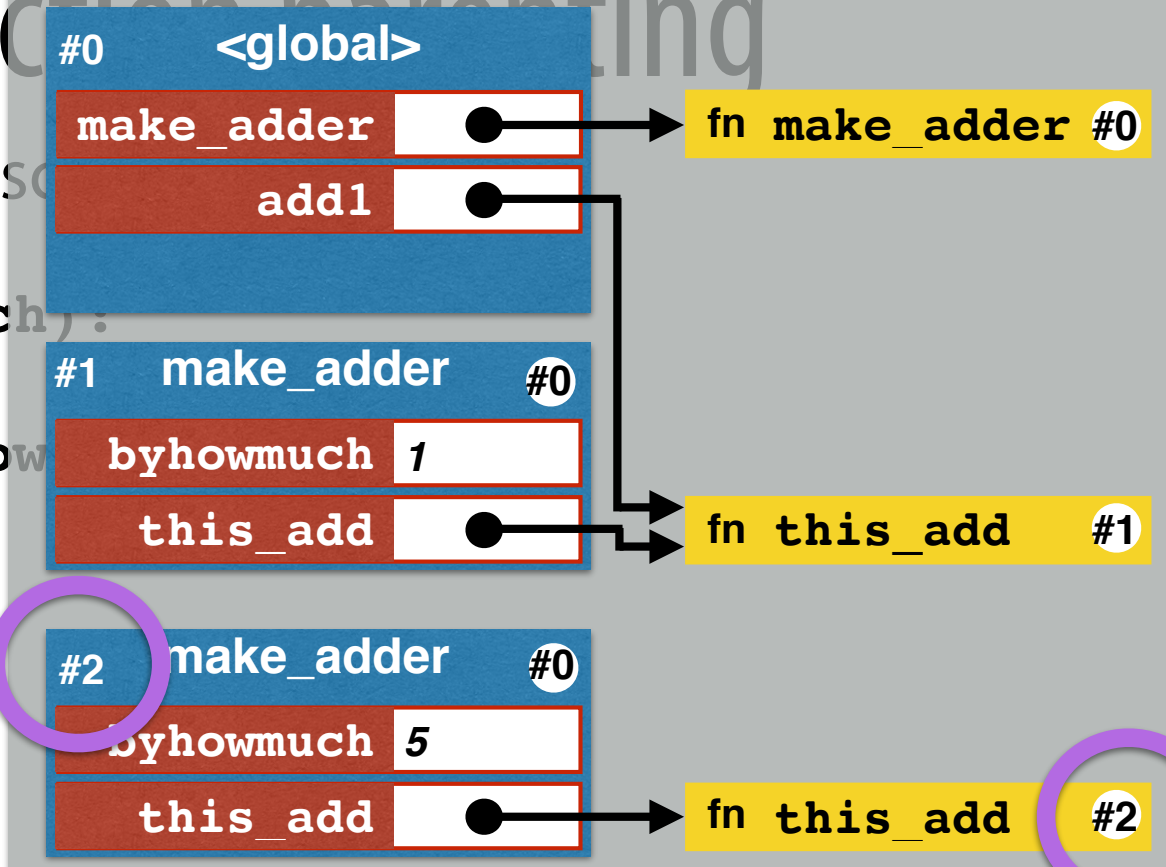


- A second `this_add` function object is constructed during this call with parameter 5.
- The *def* is executed within the context of frame #2.

Nested function parenting

What happens when this s

```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhow
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

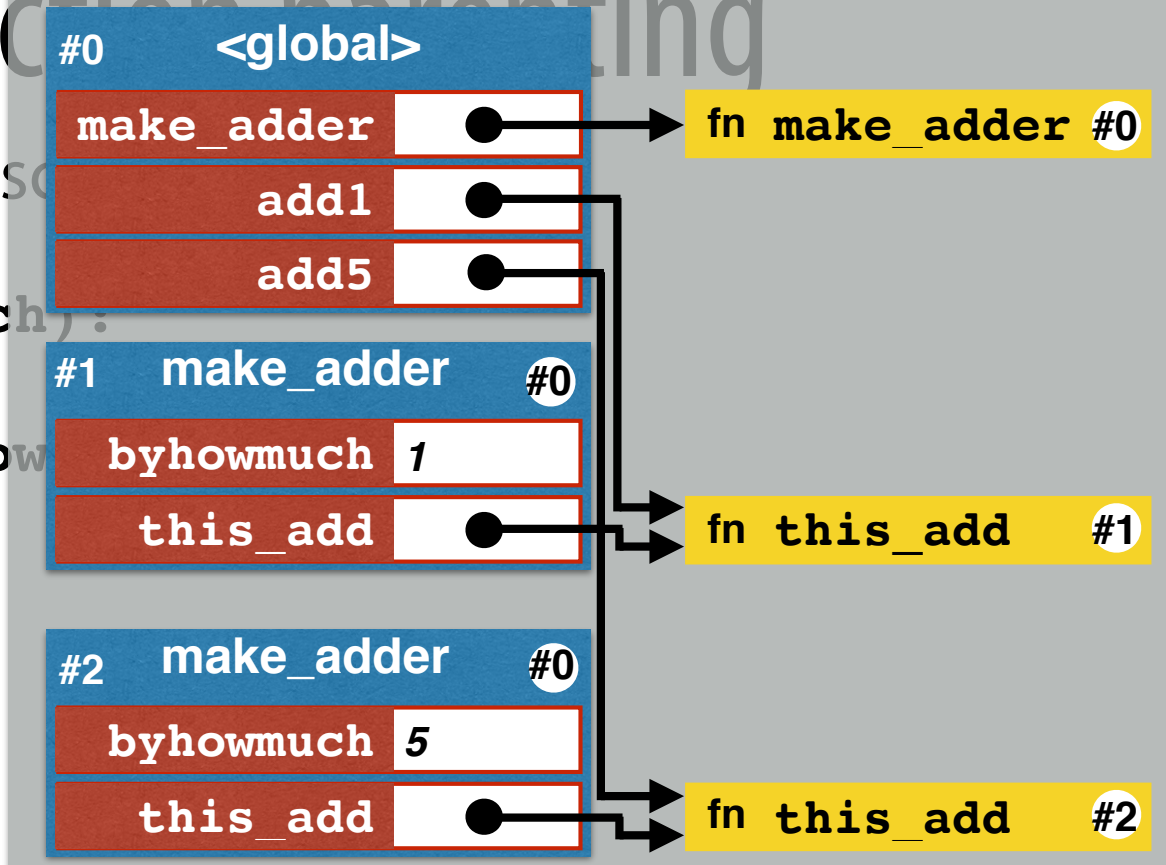


➡ The second `this_add` function object's parent frame is #2.

Nested function parenting

What happens when this s

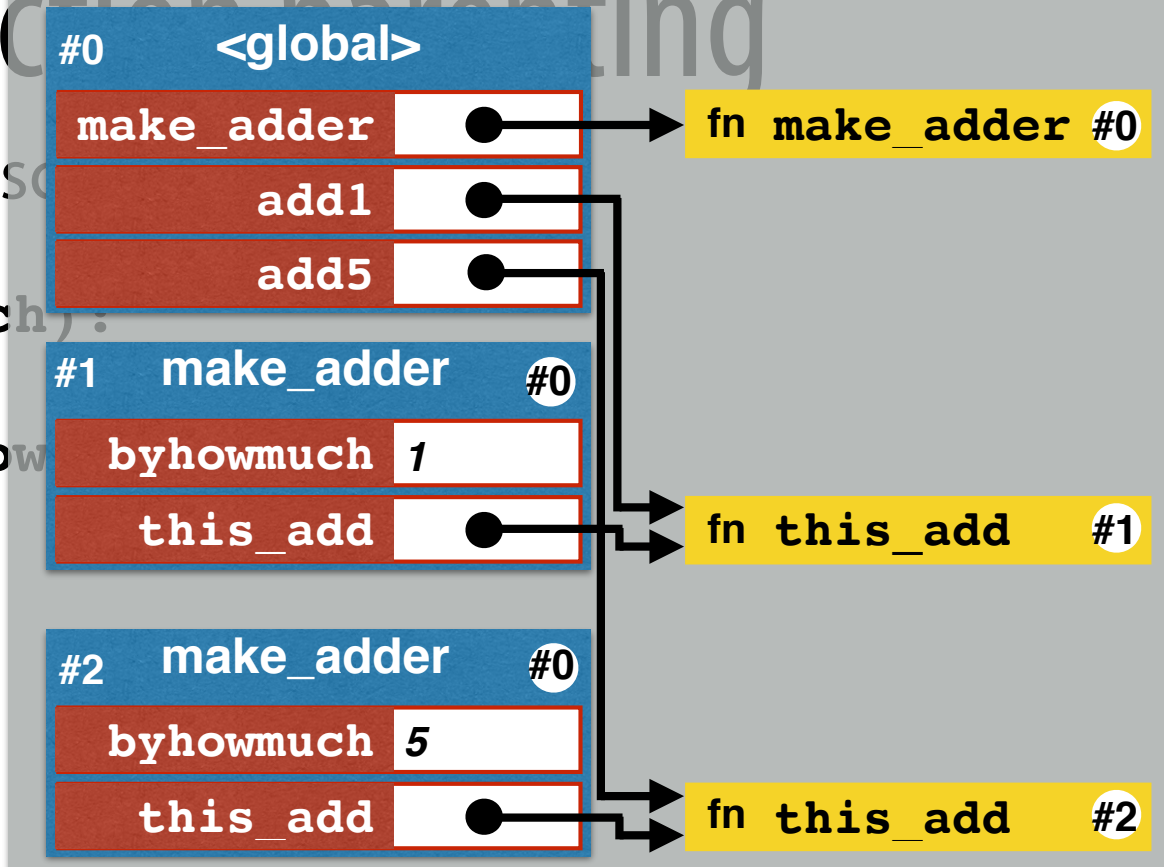
```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byho
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```



Nested function parenting

What happens when this s

```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhow
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

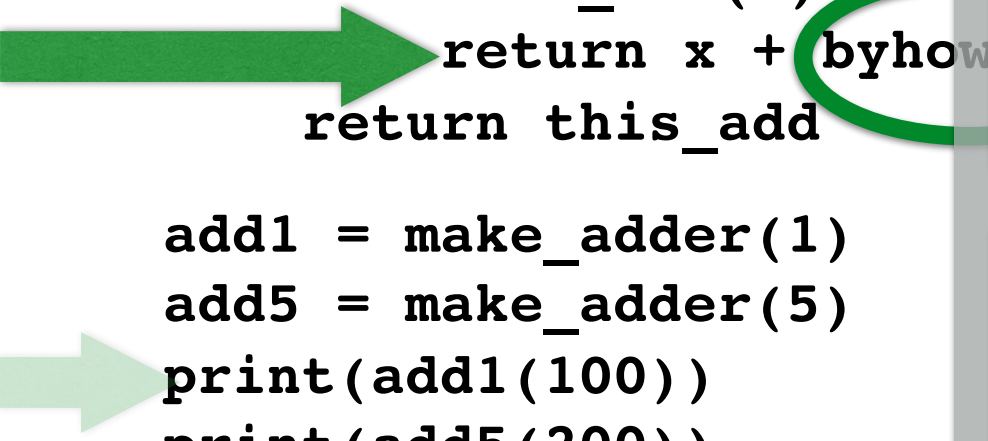


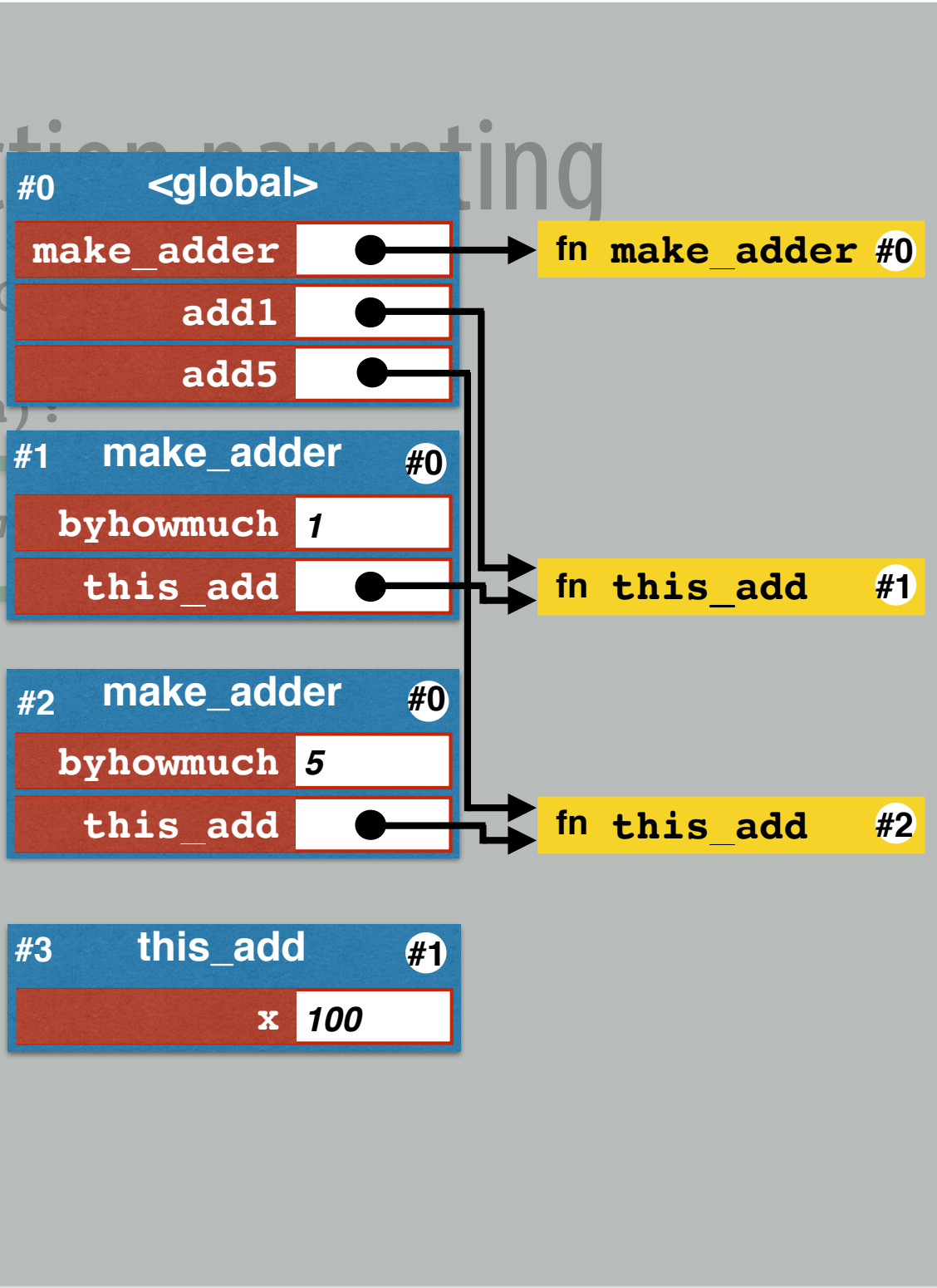
- Now the top-level code has access to two different adding functions `add1` and `add5`.
- Each is a different closure.

Nested function parenting

What happens when this s

```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhow
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

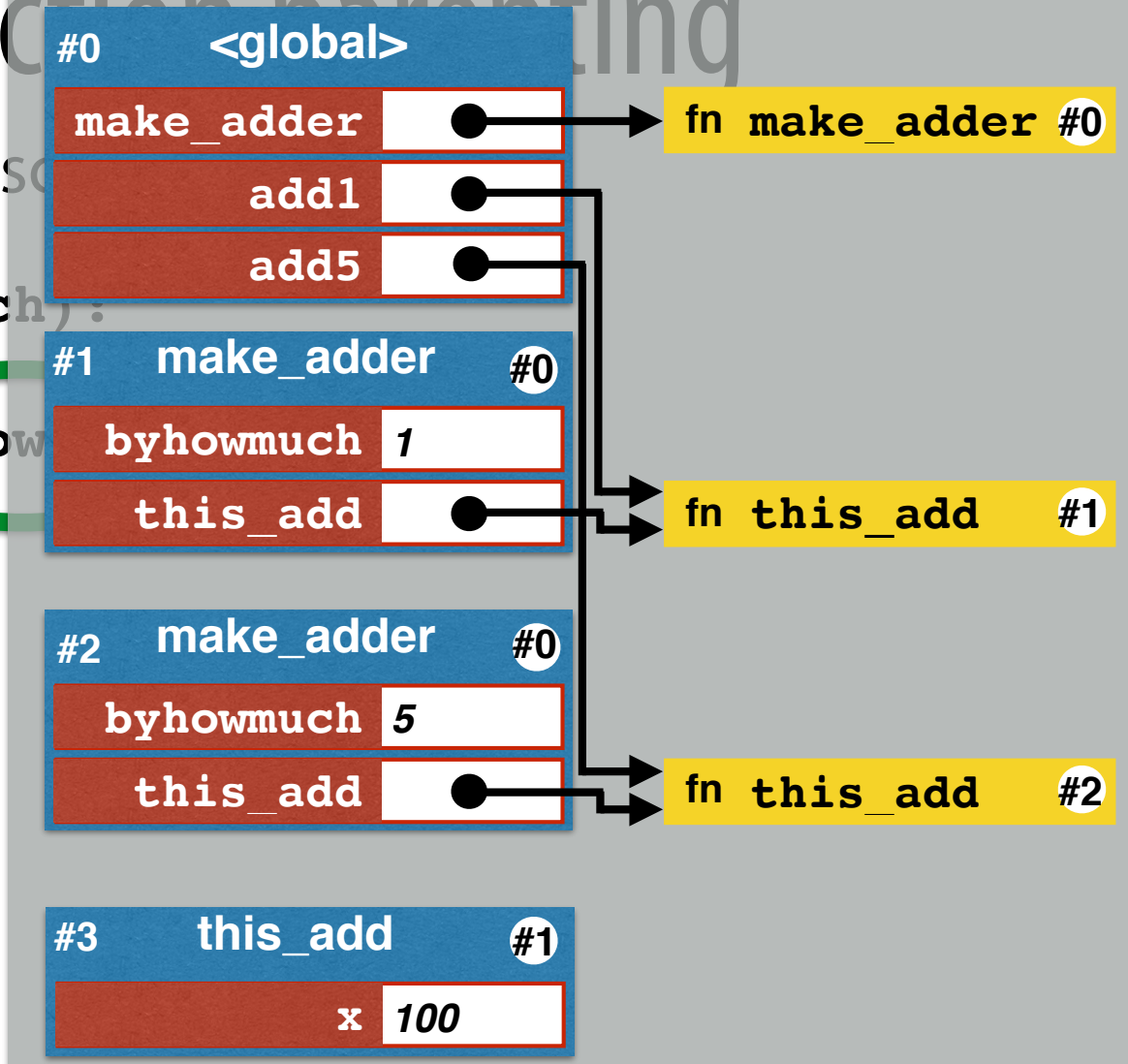




Nested function parenting

What happens when this s

```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhow
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

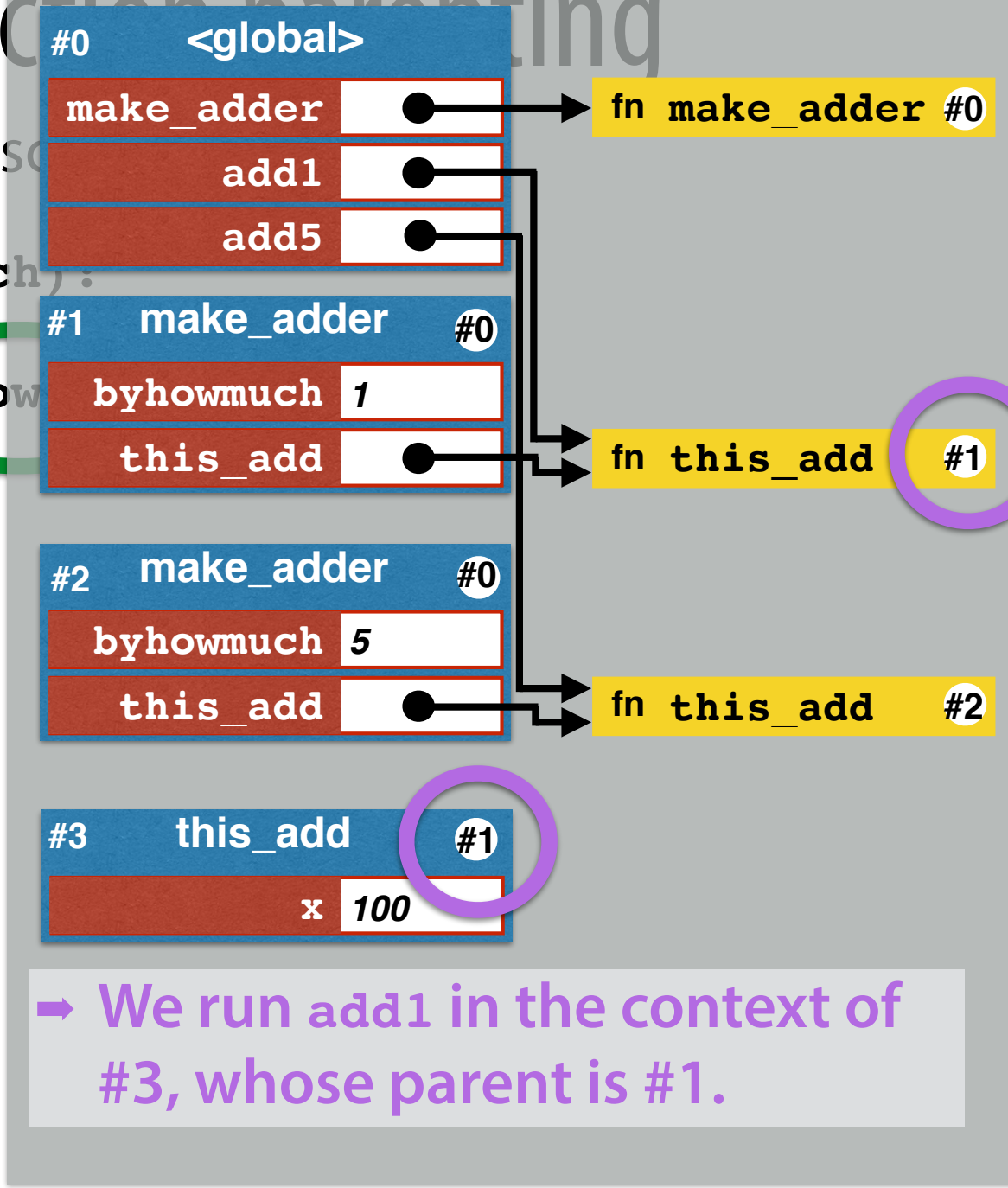


➡ We run `add1` in the context of #3, whose parent is #1.

Nested function parenting

What happens when this s

```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhow
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

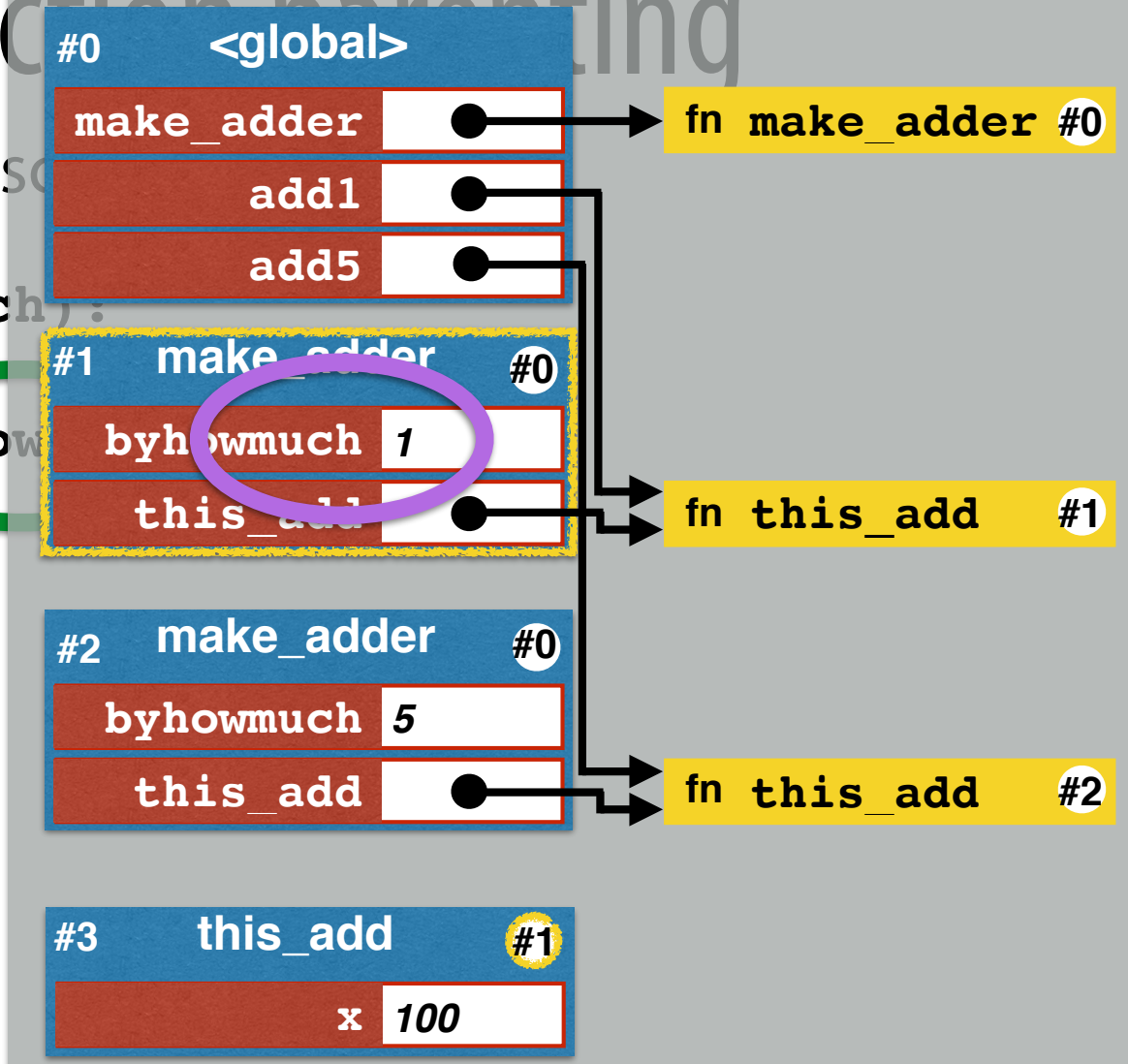


➡ We run `add1` in the context of #3, whose parent is #1.

Nested function parenting

What happens when this s

```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byho
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```



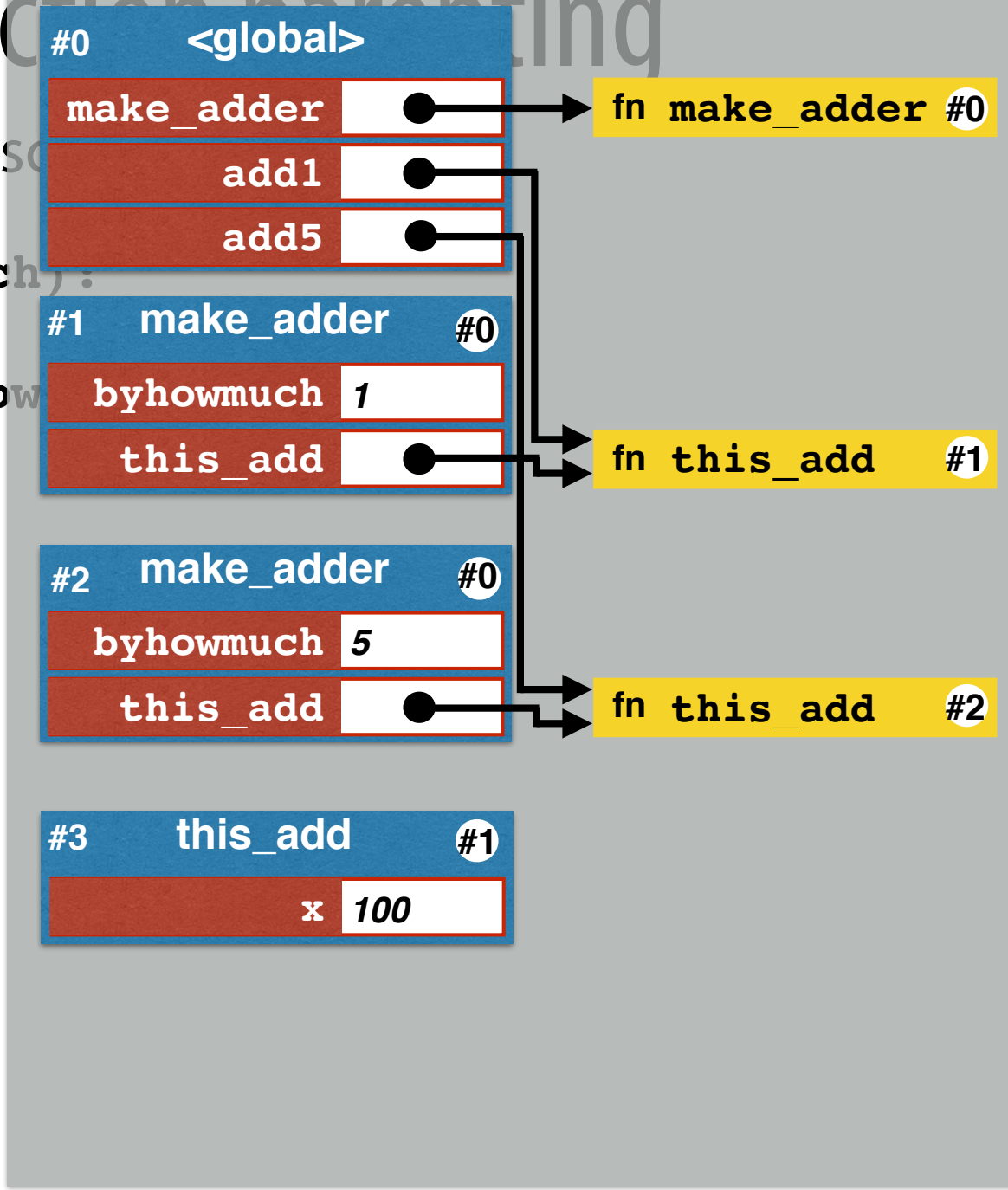
➡ We resolve `byhowmuch` by its value when add1 was built.

Nested function parenting

What happens when this s

```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byhow
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

101



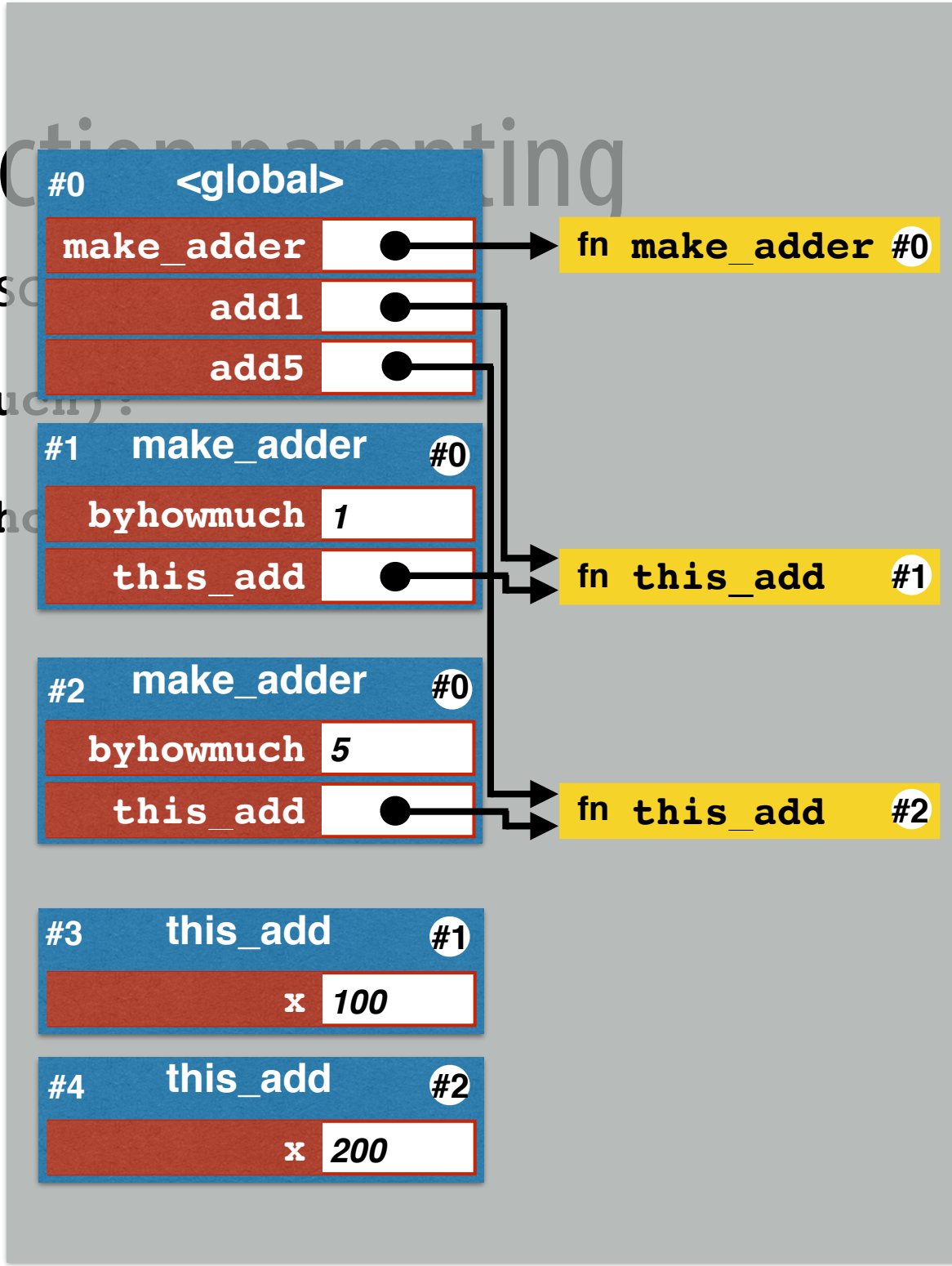
Nested function parenting

What happens when this s

```
def make_adder(byhowmuch):
    def this_add(x):
        return x + byh
    return this_add

add1 = make_adder(1)
add5 = make_adder(5)
print(add1(100))
print(add5(200))
```

Output to the console:

```
101
205
```



Following the parent chain

When a name is accessed in a statement:

- Python checks the active local frame.
- If not there, Python checks its parent frame.
- If not there, Python checks its parent's parent frame.
- Etc.
- Eventually this could hit the global frame, raise an error.

This is the ***execution environment*** of that statement.

Frames can get complicated.

What happens when this script is executed?

```
apply2 = (lambda f: (lambda x: f(f(x))))

def make_adder(byhowmuch):
    return (lambda x: x + byhowmuch)

add1 = make_adder(1)
add5 = make_adder(5)

this = apply2(add1)
that = apply2(add5)

this(1000)
that(1000)
```

Environment Example

Show this code's execution with an environment diagram:

```
def thing_maker(a,b,c):
    x = 35+b
    c = c+1
    def thing(v,w):
        print(x)
        print(a)
        u = v*b + w
        return u
    x = x - 30
    return thing

thingA = thing_maker(3,4,5)
r = thingA(10,11)
print(r)
```