Access to the global frame

What happens when this script is executed?

```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the squares “
    print(s + str(a) + “ “ + str(b) + “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```

The global frame is accessible

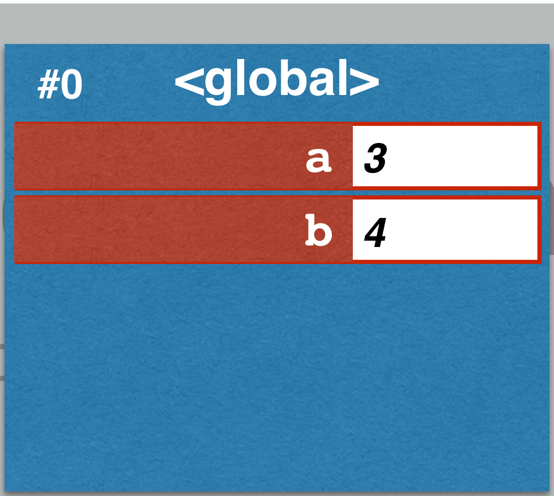
A function has access to globally-defined names:

- ➡ Python checks a function’s local frame for a name.
- ➡ If no slot in local frame, it *checks the global frame*.
- ➡ A function can call ***globally defined functions.***
- ➡ A function can use ***global variables***.
- ➡ If the function assigns a variable anywhere within its code (i.e. “locally”) then Python treats it as a ***local variable***.

Access to th me

What happens when this sc

```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the squares “
    print(s + str(a) + “ “ + str(b) + “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```

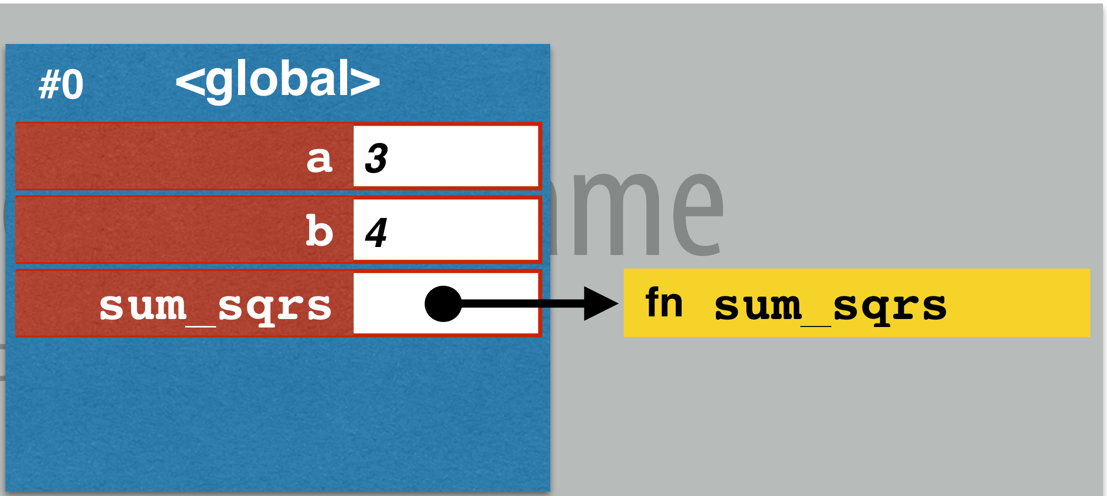


#0 <global>

a	3
b	4

Access to th me

What happens when this sc

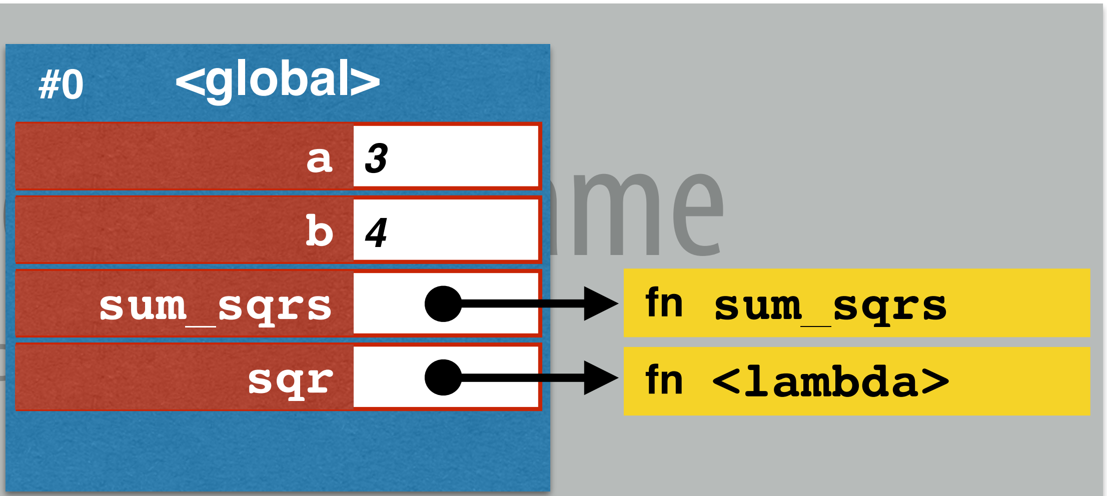


```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the squares “
    print(s + str(a) + “ “ + str(b) + “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```

Access to th me

What happens when this sc

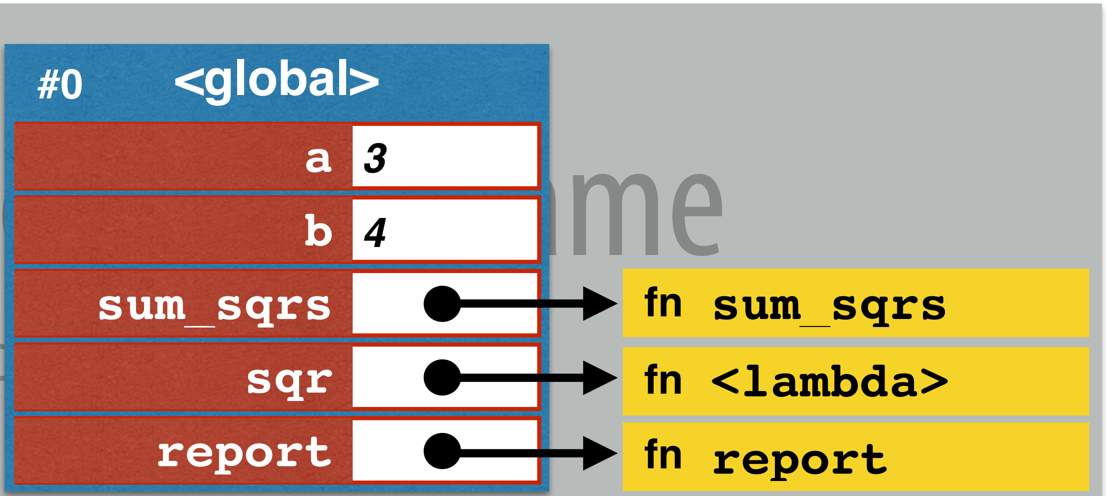
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the squares “
    print(s + str(a) + “ “ + str(b) + “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```



Access to th me

What happens when this sc

```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the squares “
    print(s + str(a) + “ “ + str(b) + “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```



#0 <global>

a 3

b 4

sum_sqrs → fn sum_sqrs

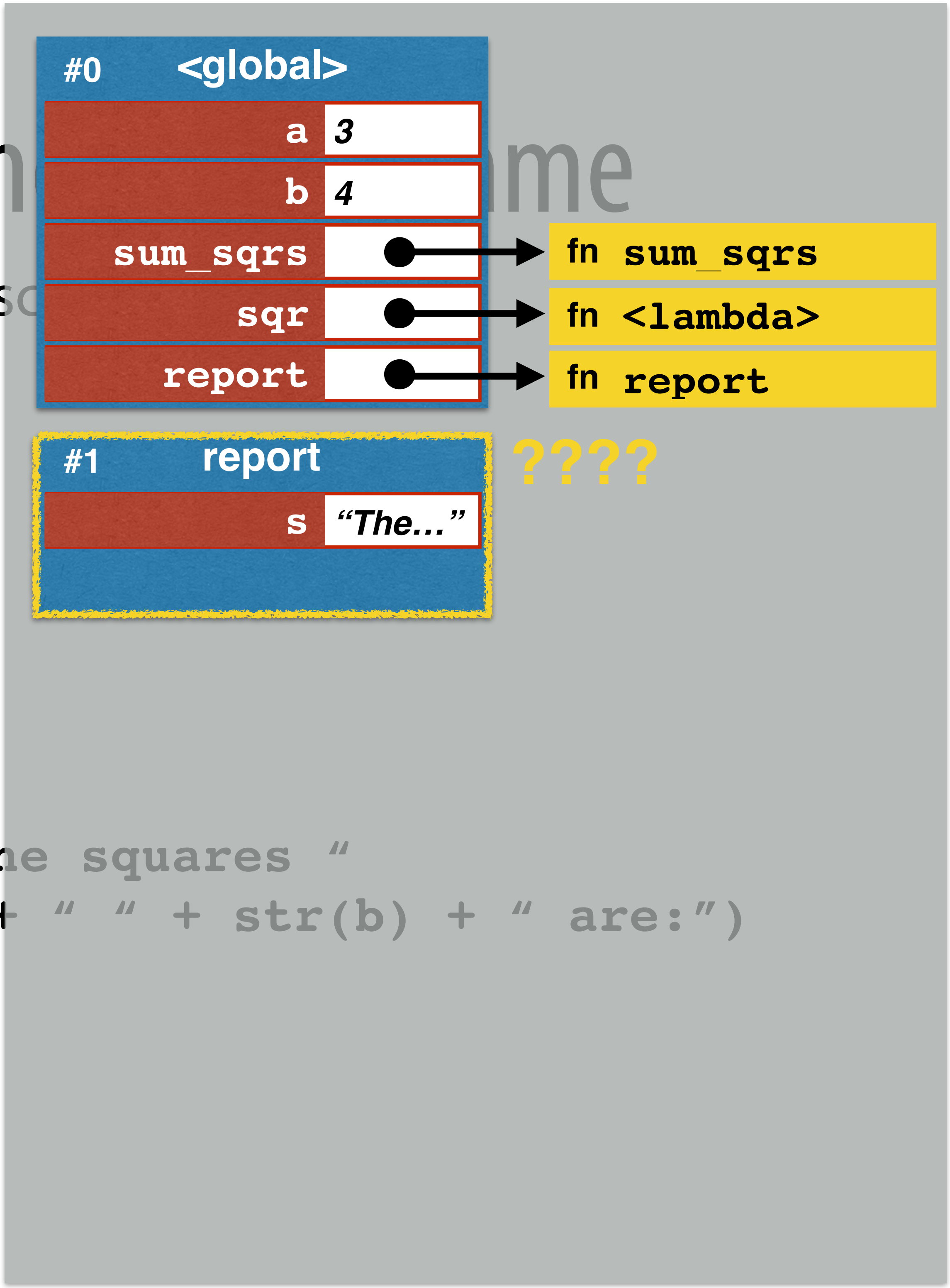
sqr → fn <lambda>

report → fn report

Access to th me

What happens when this sc

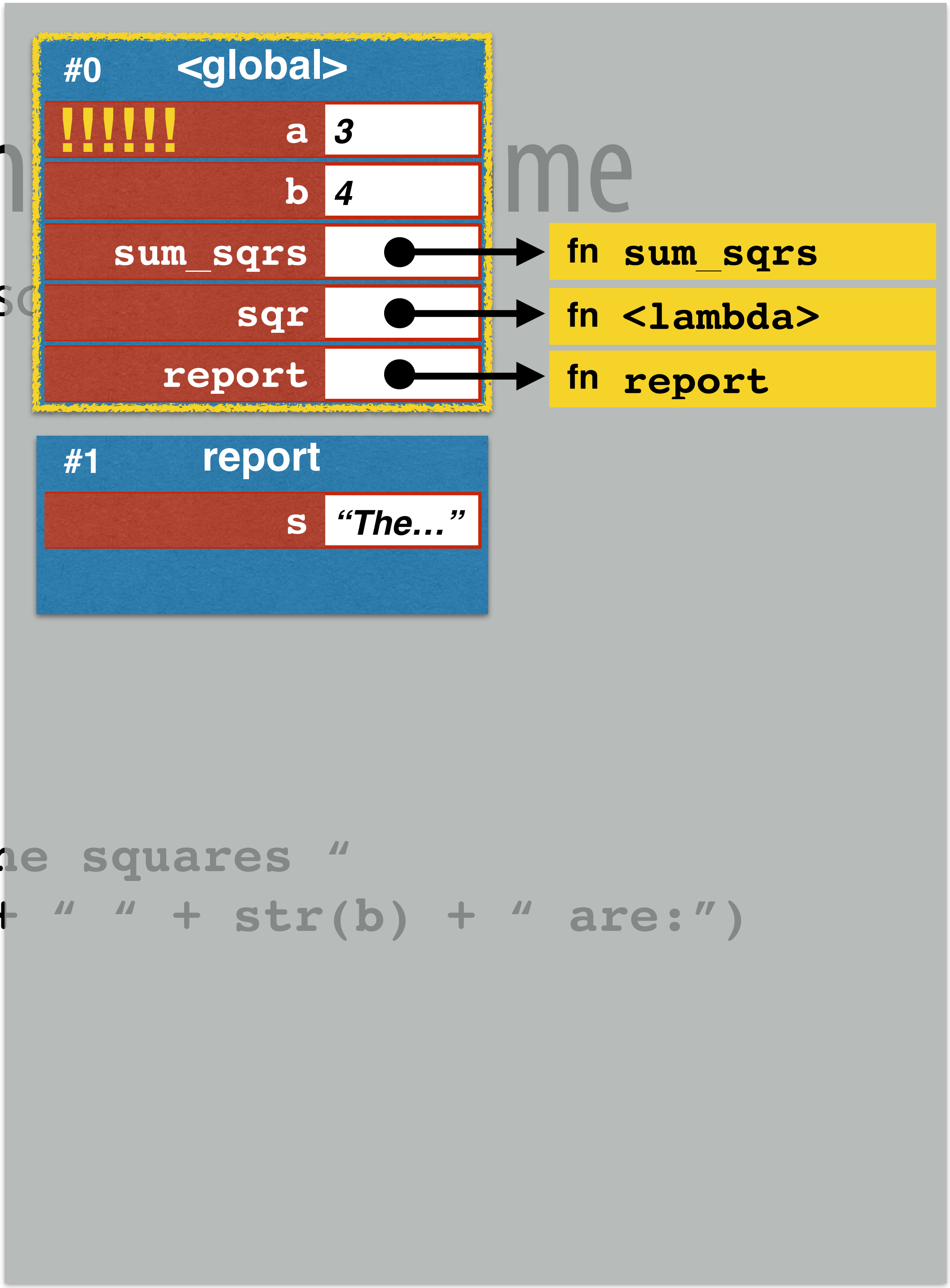
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the squares “
    print(s + str(a) + “ “ + str(b) + “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```



Access to th me

What happens when this sc

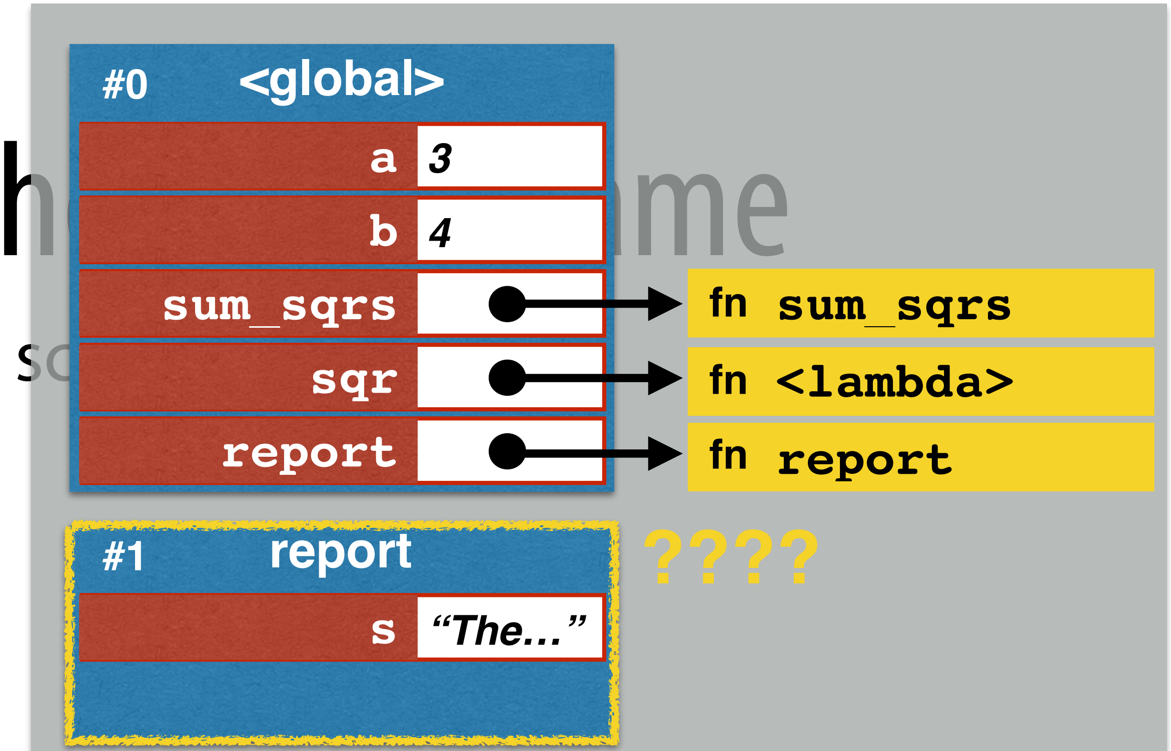
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the squares “
    print(s + str(a) + “ “ + str(b) + “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```



Access to th me

What happens when this sc

```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the squares “
    print(s + str(a) + “ “ + str(b) + “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```



#0	<global>
a	3
b	4
sum_sqrs	fn sum_sqrs
sqr	fn <lambda>
report	fn report

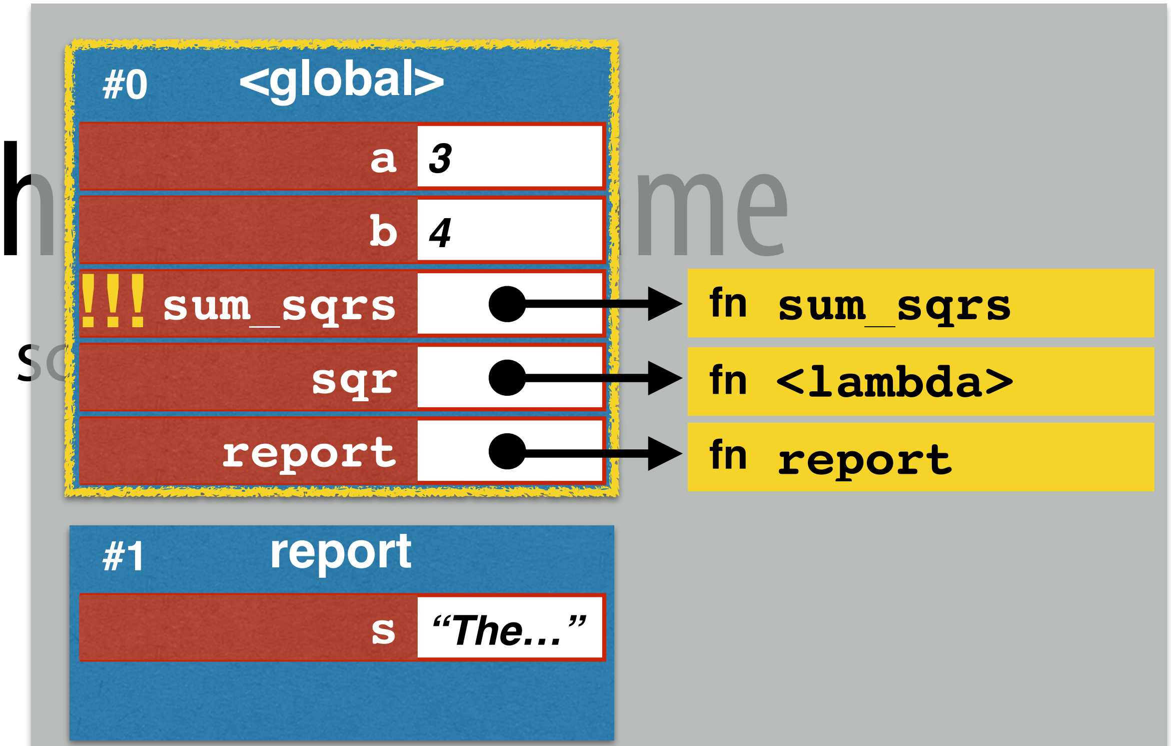
#1	report
s	“The…”

????

Access to th me

What happens when this sc

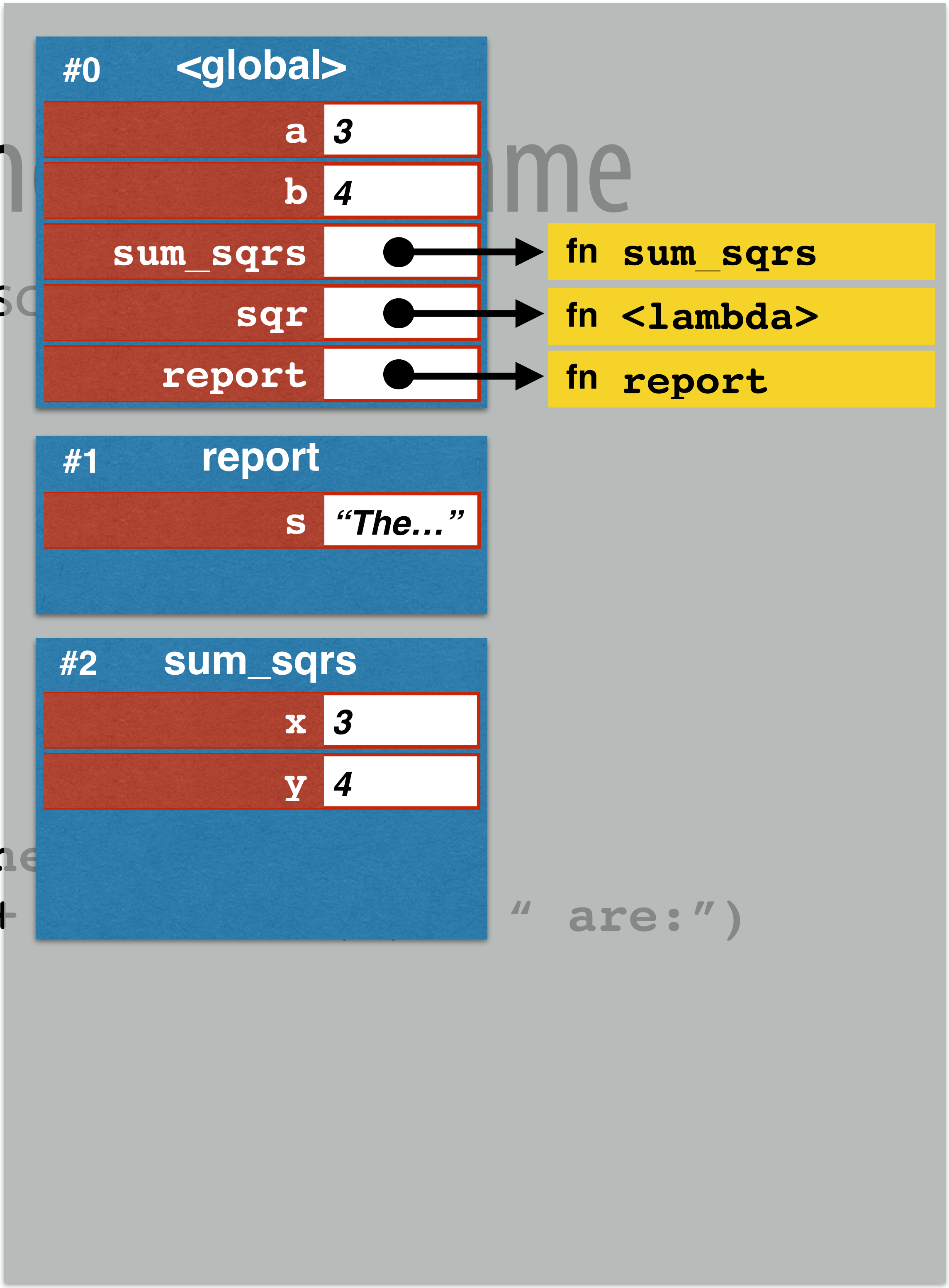
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the squares “
    print(s + str(a) + “ “ + str(b) + “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```



Access to th me

What happens when this sc

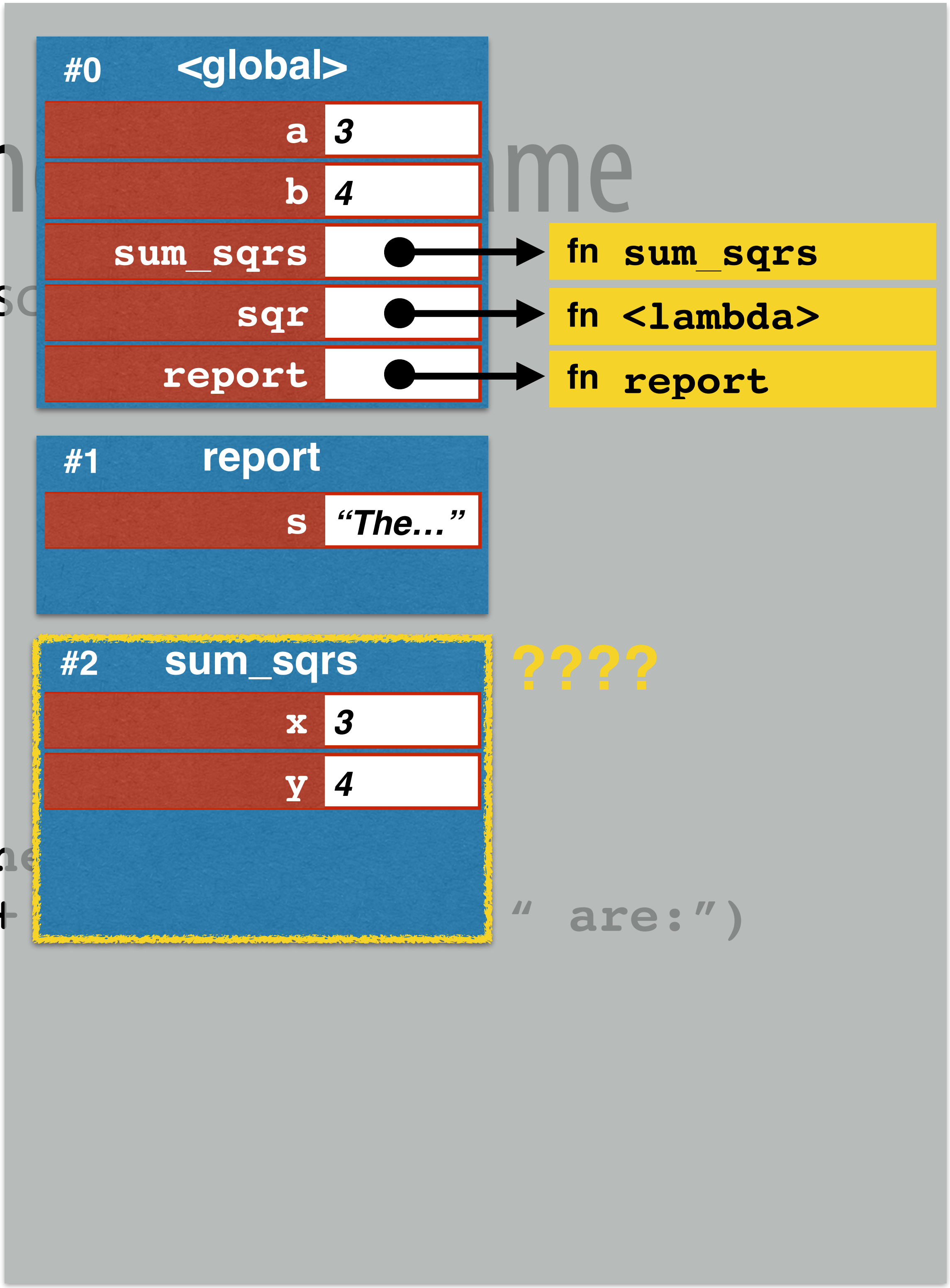
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the
    print(s + str(a) +                    “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```



Access to th me

What happens when this sc

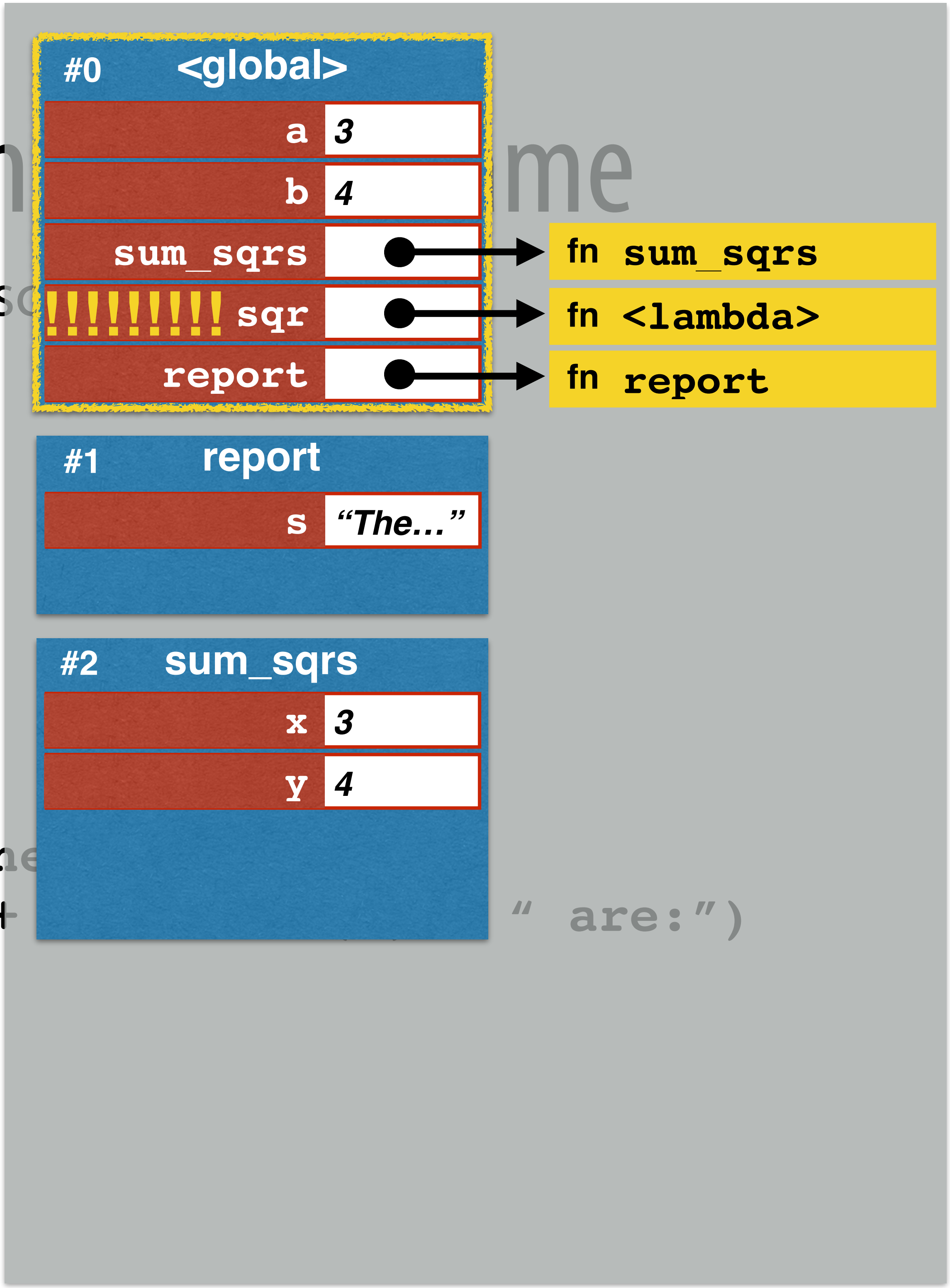
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the
    print(s + str(a) +                “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```



Access to th me

What happens when this sc

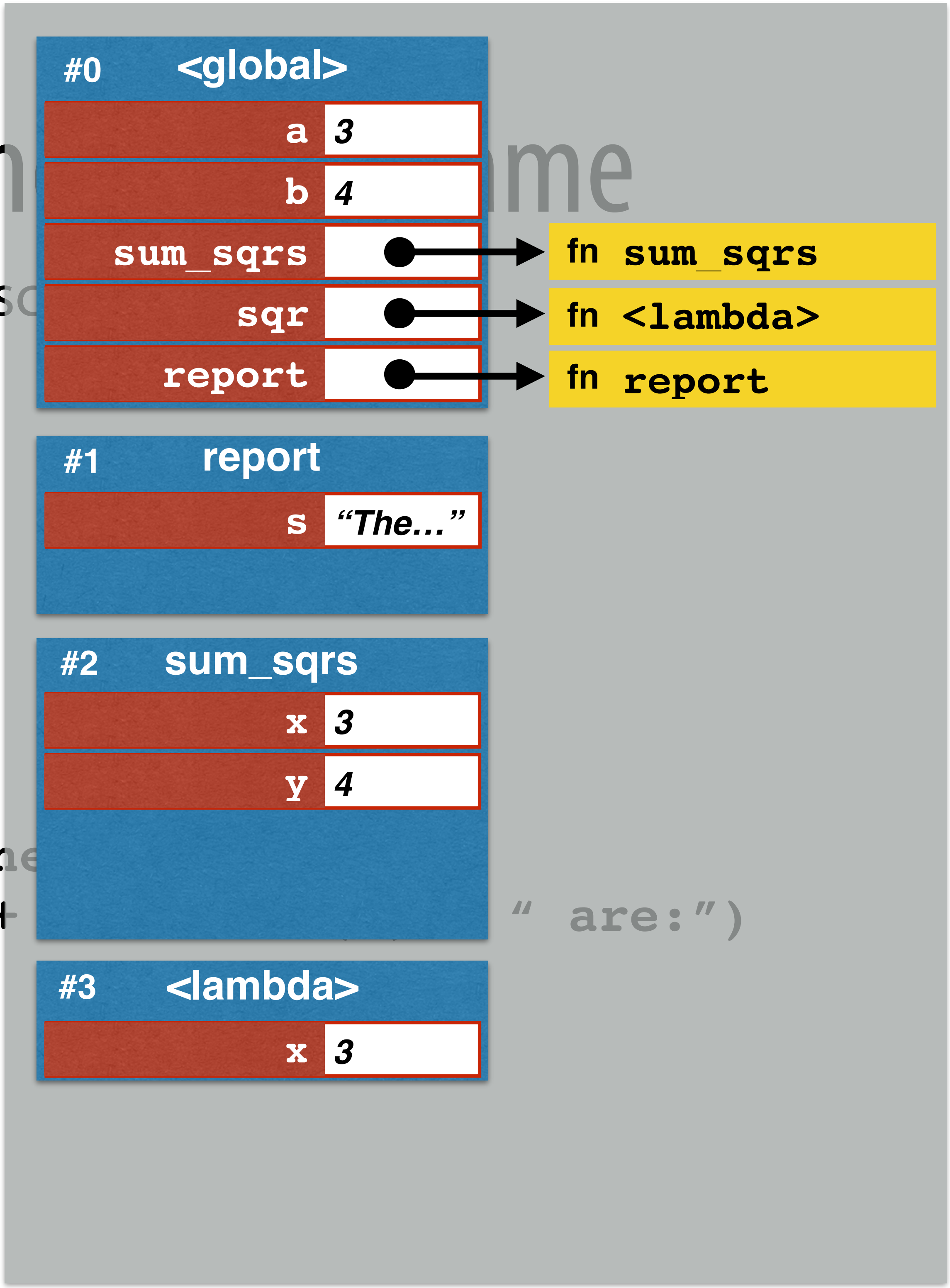
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the
    print(s + str(a) +                  “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```



Access to th me

What happens when this sc

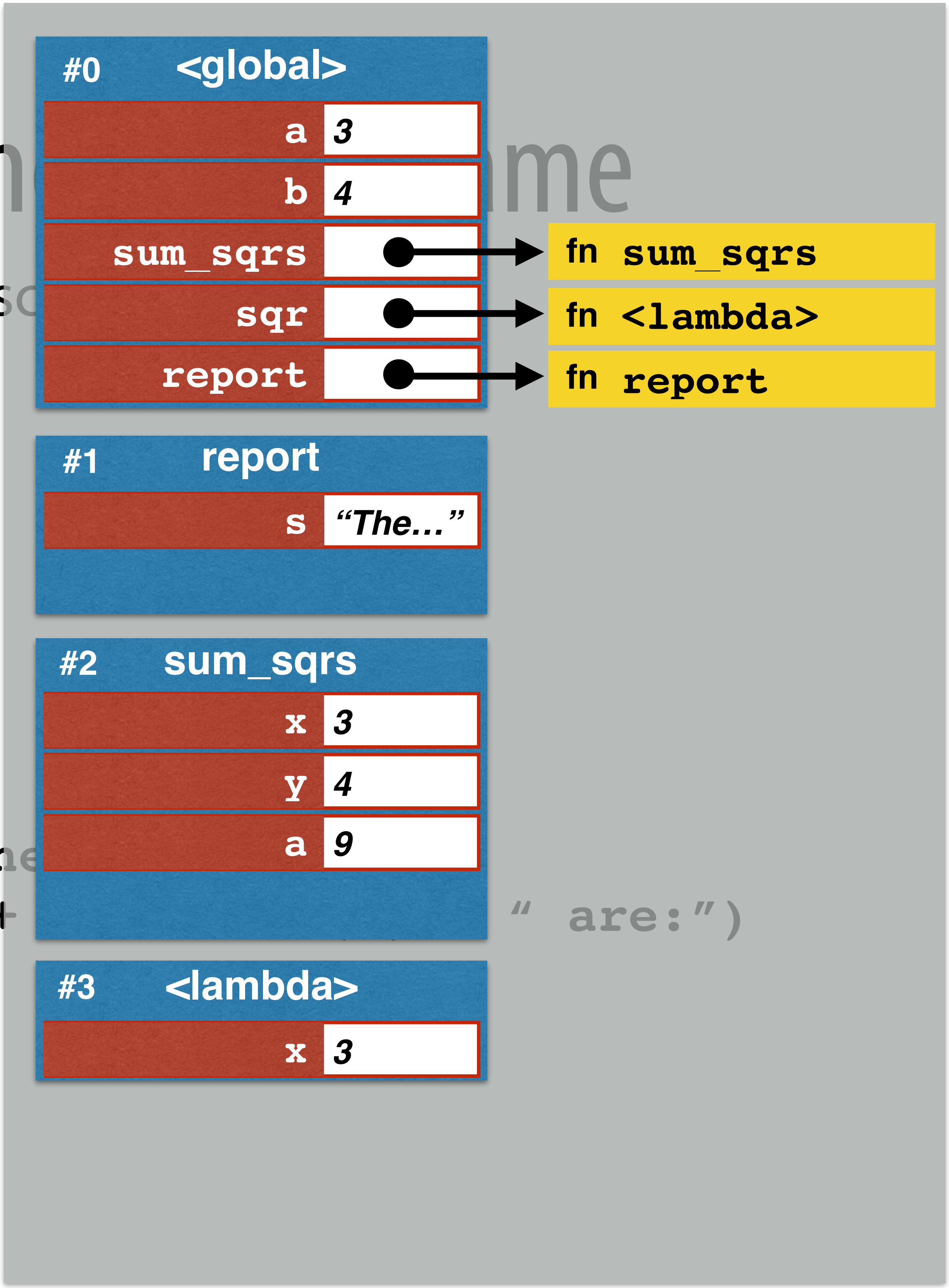
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the
    print(s + str(a) +           “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```



Access to th me

What happens when this sc

```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the
    print(s + str(a) +          “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```



#0 <global>

a	3
b	4
sum_sqrs	→ fn sum_sqrs
sqr	→ fn <lambda>
report	→ fn report

#1 report

s	“The…”

#2 sum_sqrs

x	3
y	4
a	9

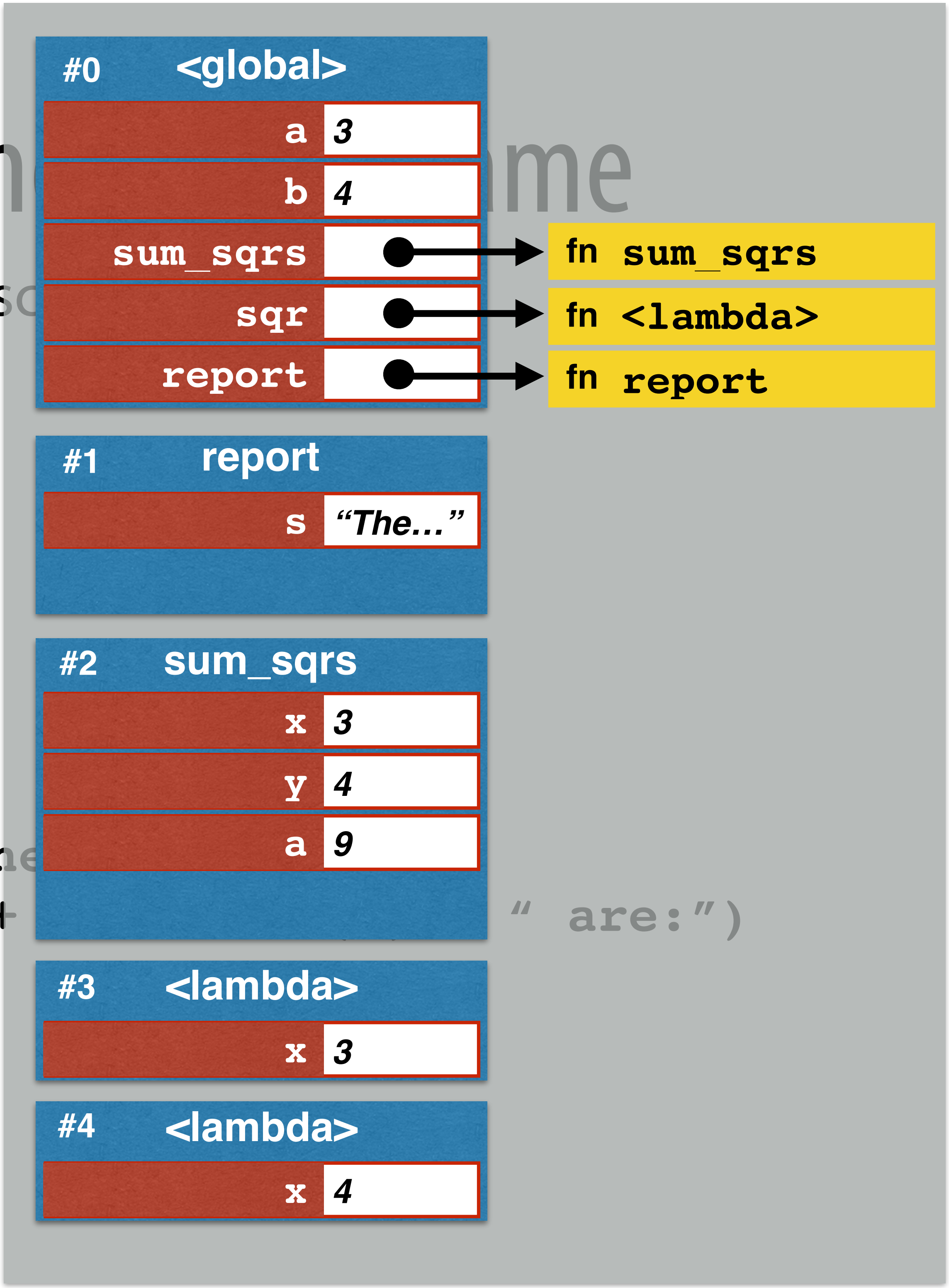
#3 <lambda>

x	3

Access to th me

What happens when this sc

```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the
    print(s + str(a) +                      “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```



#0 <global>

a 3

b 4

sum_sqrs → fn sum_sqrs

sqr → fn <lambda>

report → fn report

#1 report

s “The…”

#2 sum_sqrs

x 3

y 4

a 9

#3 <lambda>

x 3

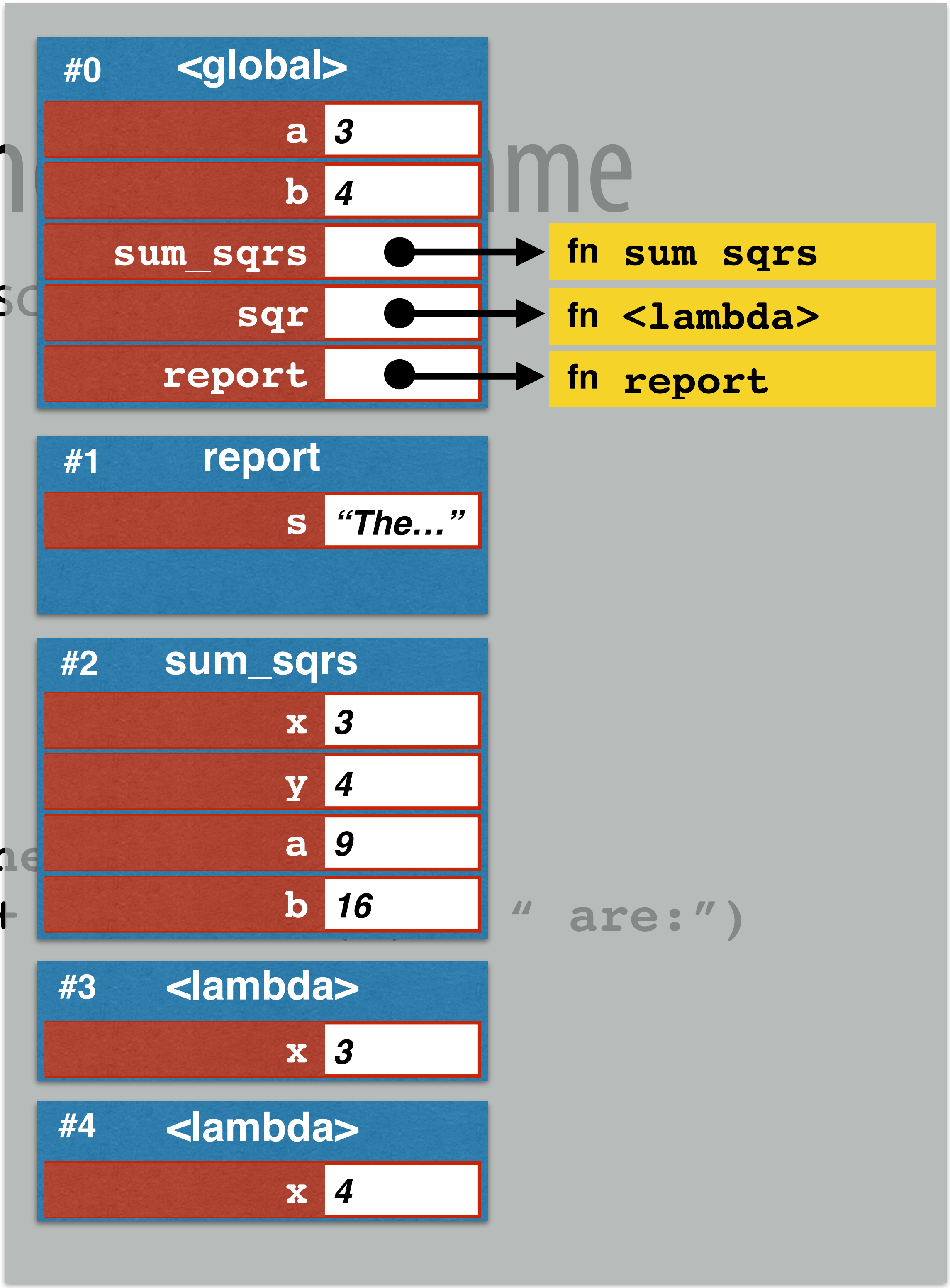
#4 <lambda>

x 4

Access to th me

What happens when this sc

```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the
    print(s + str(a) +                 “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```



#0 <global>

a	*3*
b	*4*
sum_sqrs	● → fn sum_sqrs
sqr	● → fn <lambda>
report	● → fn report

#1 report

s	*“The…”*

#2 sum_sqrs

x	*3*
y	*4*
a	*9*
b	*16*

#3 <lambda>

x	*3*

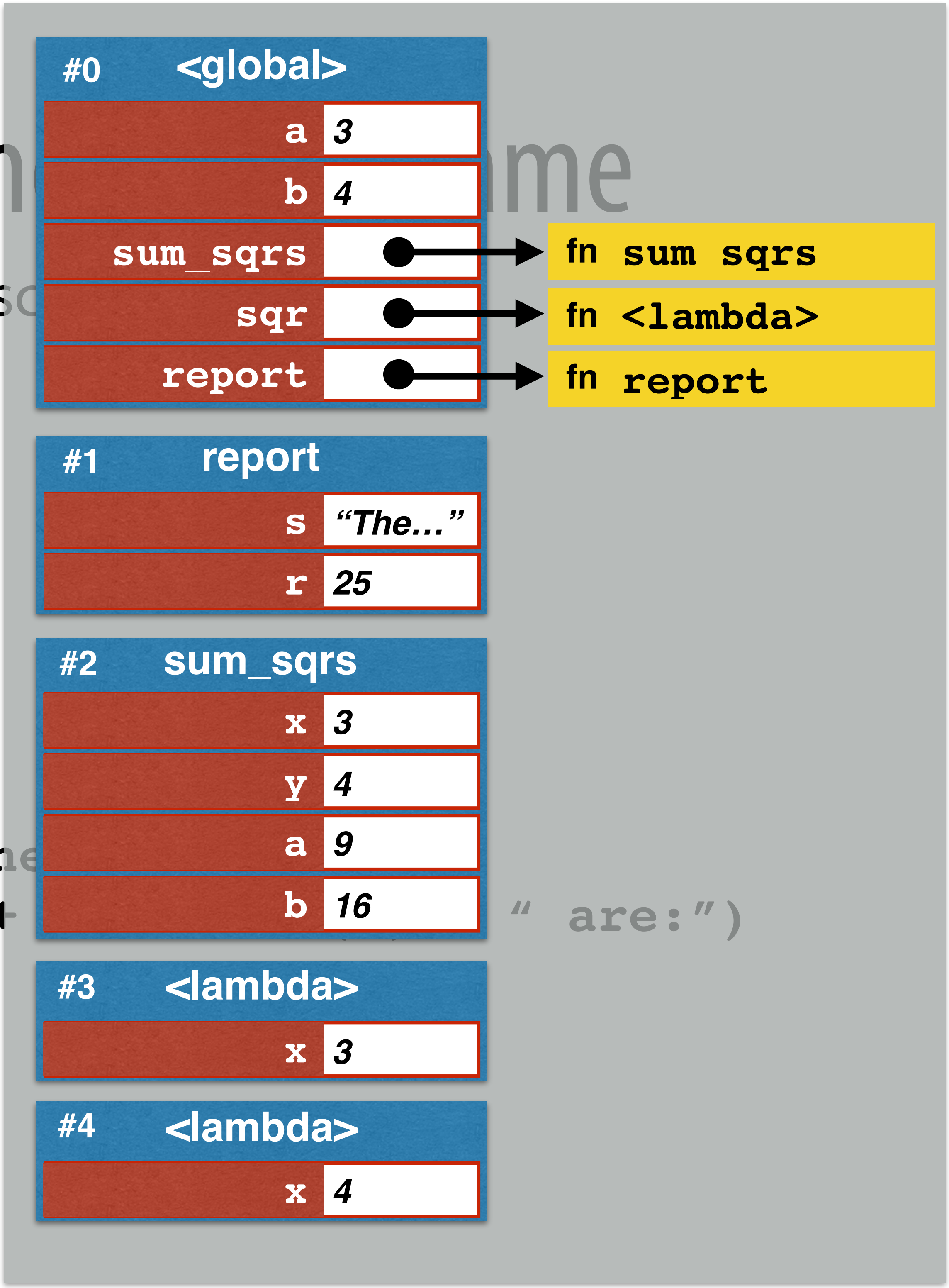
#4 <lambda>

x	*4*

Access to th me

What happens when this sc

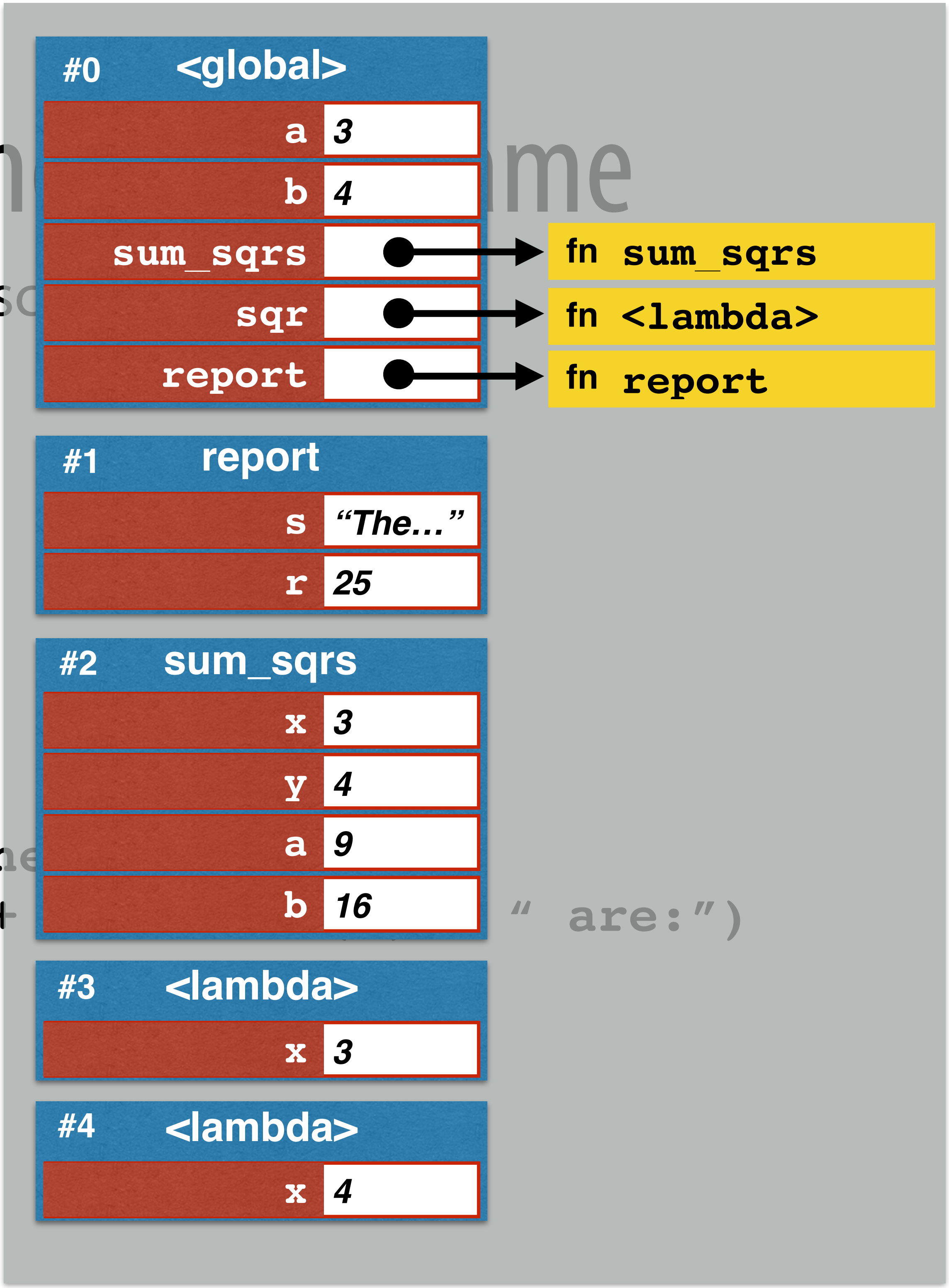
```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the
    print(s + str(a) +                  “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```



Access to th   me

What happens when this sc

```
a = 3
b = 4
def sum_sqrs(x,y):
    a = sqr(x)
    b = sqr(y)
    return a + b
sqr = lambda x: x * x
def report():
    s = “The sum of the
    print(s + str(a) +                          “ are:”)
    r = sum_sqrs(a,b)
    print(r)
report()
```



#0	<global>
a	3
b	4
sum_sqrs	→ fn sum_sqrs
sqr	→ fn <lambda>
report	→ fn report

#1	report
s	“The…”
r	25

#2	sum_sqrs
x	3
y	4
a	9
b	16

#3	<lambda>
x	3

#4	<lambda>
x	4

The global frame is accessible

A function has access to globally-defined names:

- ➡ When a variable name is used, Python checks local frame.
- ➡ If no binding in local frame, it checks the global frame.
- ➡ If a global function is called locally, binding is known
- ➡ If a global variable is accessed locally, binding is known.
- ➡ A local assignment within a function makes that name a new local variable. Any other mention of that name will be treated as a local access/update.

What about passing functions?

What happens when this script is executed?

```
def sqr(x):
    return x * x
def sum_apply(f,v1,v2):
    a = f(v1)
    b = f(v2)
    return a + b
r = sum_apply(sqr,3,4)
```

Passing functions

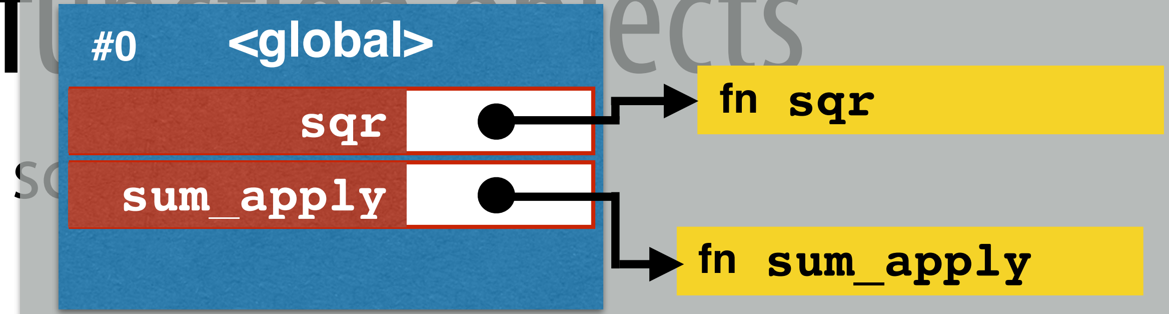
Suppose a function object is passed as a parameter.

- ➡ The local variable for that parameter *also refers* to that function object.
- ➡ The global name and local parameter name are ***aliases*** for that function object.
- ➡ When the function is called locally, that same function object is used.

Aliasing of function objects

What happens when this sc

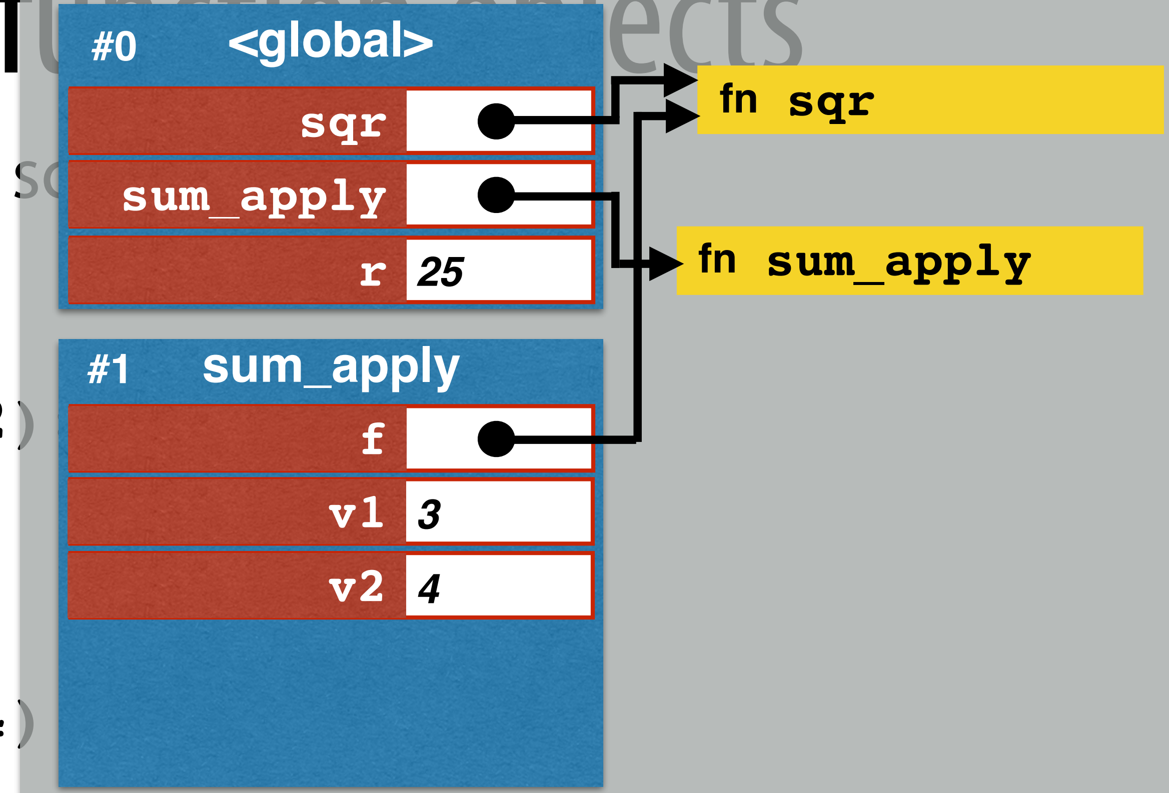
```
def sqr(x):
    return x * x
def sum_apply(f,v1,v2):
    a = f(v1)
    b = f(v2)
    return a + b
r = sum_apply(sqr,3,4)
```



Aliasing of function objects

What happens when this sc

```
def sqr(x):
    return x * x
def sum_apply(f,v1,v2)
    a = f(v1)
    b = f(v2)
    return a + b
r = sum_apply(sqr,3,4)
```

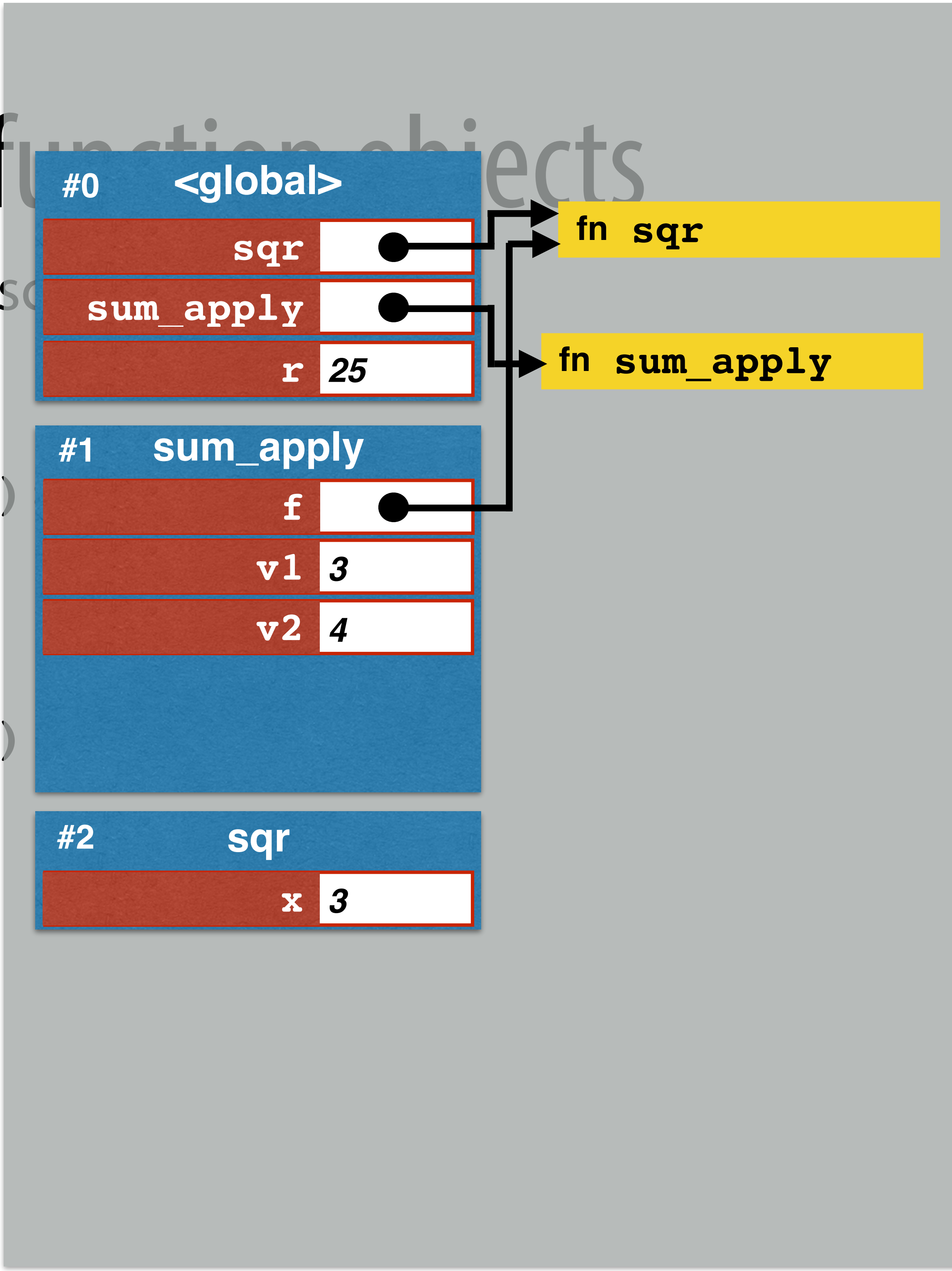


➡ Both the local name `f` and the global name `sqr` refer to the same function object.

Aliasing of function objects

What happens when this sc

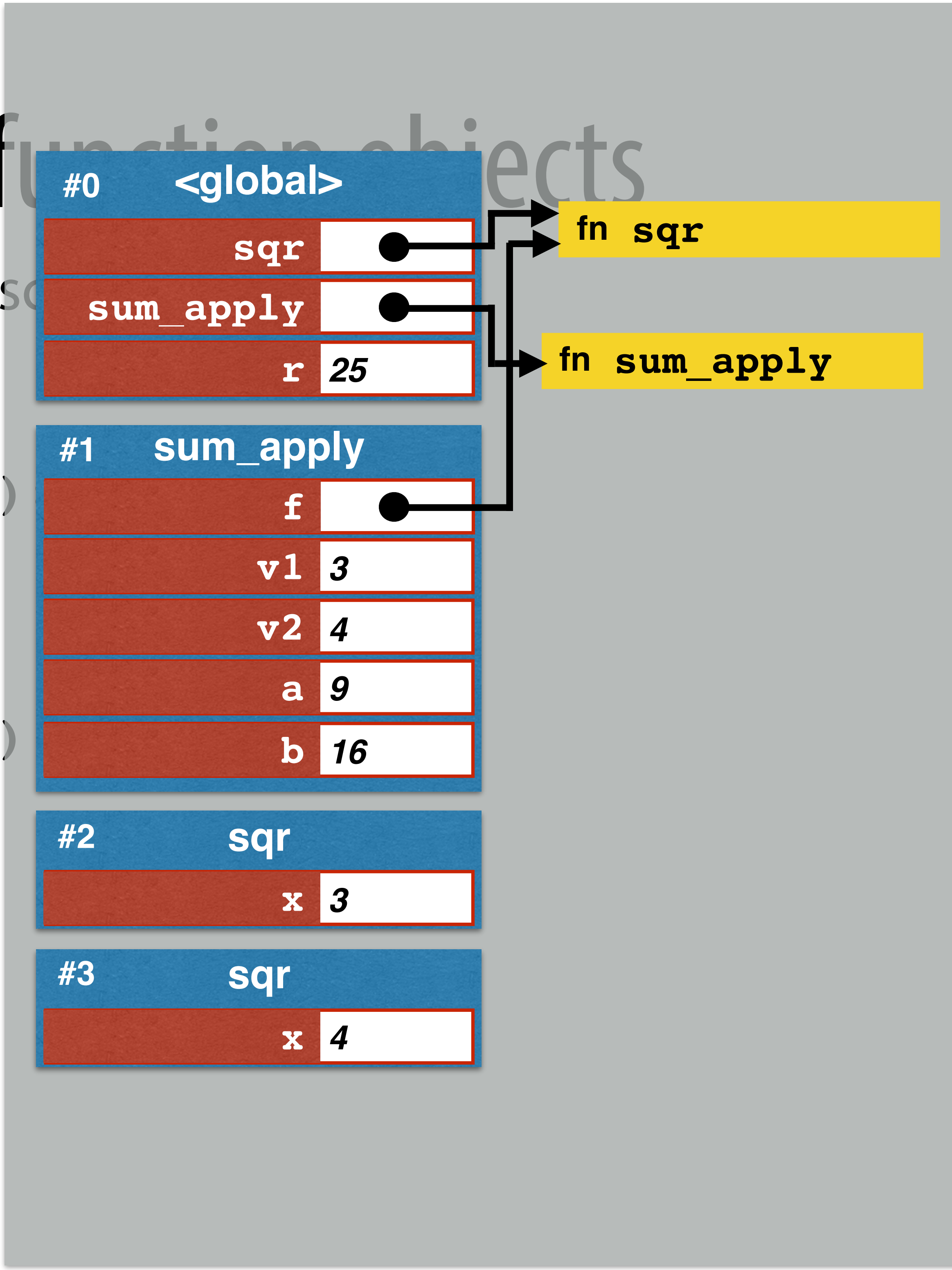
```
def sqr(x):
    return x * x
def sum_apply(f,v1,v2):
    a = f(v1)
    b = f(v2)
    return a + b
r = sum_apply(sqr,3,4)
```



Aliasing of function objects

What happens when this s

```
def sqr(x):
    return x * x
def sum_apply(f,v1,v2)
    a = f(v1)
    b = f(v2)
    return a + b
r = sum_apply(sqr,3,4)
```



#0 <global>

sqr	●
sum_apply	●
r	25

#1 sum_apply

f	●
v1	3
v2	4
a	9
b	16

#2 sqr

x	3

#3 sqr

x	4