

## Lecture #3: Environments

- Substitution is not as simple as it might seem.
- For example:

```
def f(x):  
    def g(x):  
        return x + 10  
    return g(5)  
f(3)
```

- When we call `f(3)`, we should **not** substitute 3 for the `x`s in `g`!
- And there are other difficulties...

# Names

- Evaluating expressions that are literals is easy: the literal's text gives all the information needed.
- But how did I evaluate names like `add`, `mul`, or `print`?
- How do I explain *assignment*? Substitution inadequate.

```
x = 3
```

```
print(x)
```

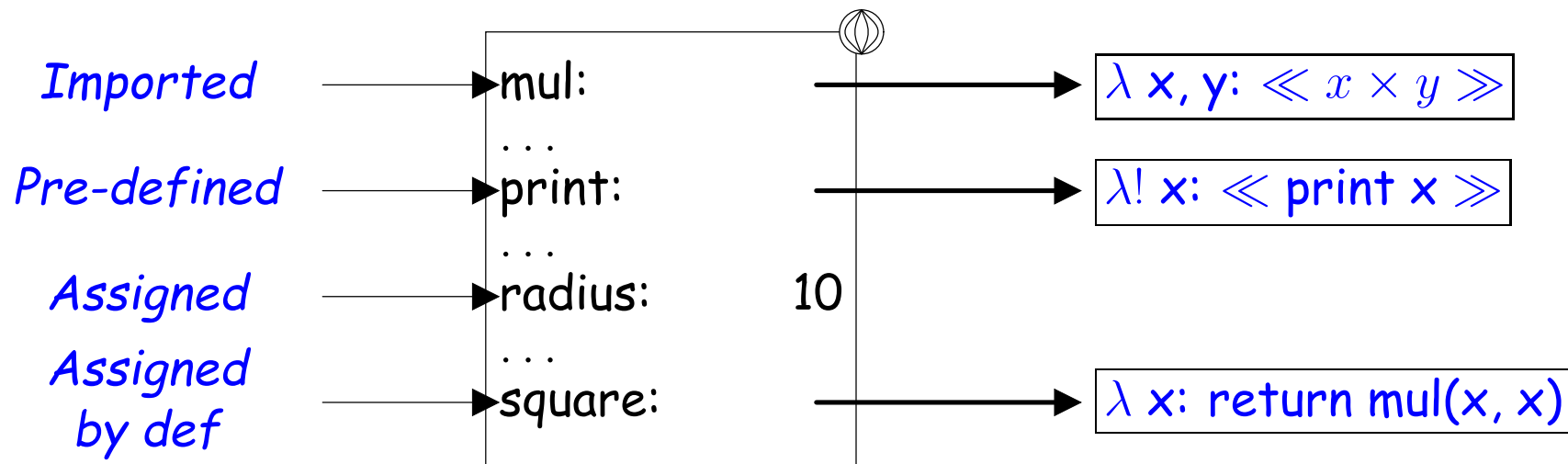
```
x = 4
```

```
print(x)    # After x = 3, does this x change to 3??!
```

- Deduction: there must be another source of information.
- We'll use the concept of an *environment* to explain it.

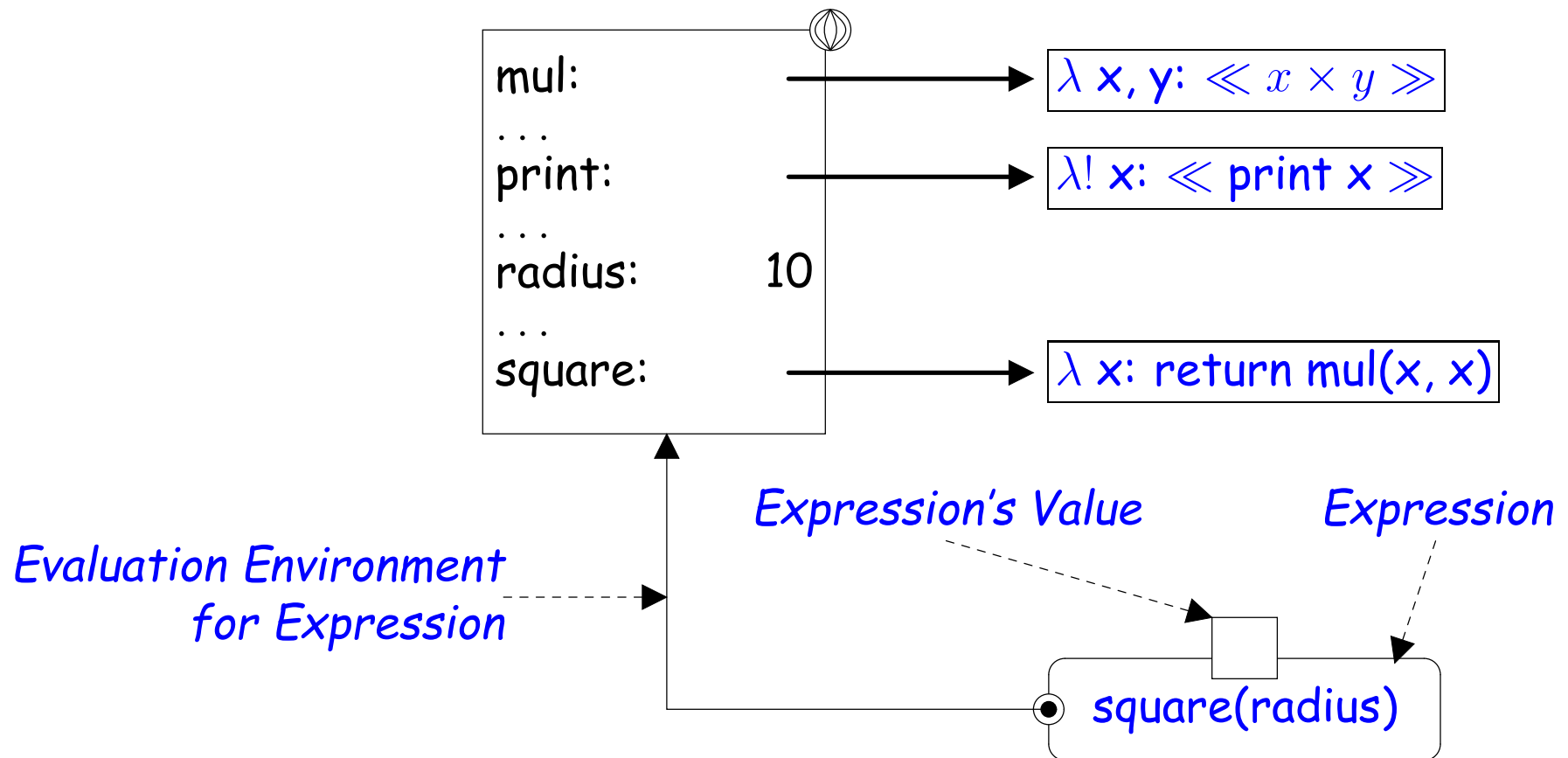
# Environments

- An **environment** is a mapping from names to values.
- We say that a name is **bound to** a value in this environment.
- Every expression is evaluated **in an environment**, which supplies the meanings of any names in it.
- Simplest environment consists of a single *global environment frame*:



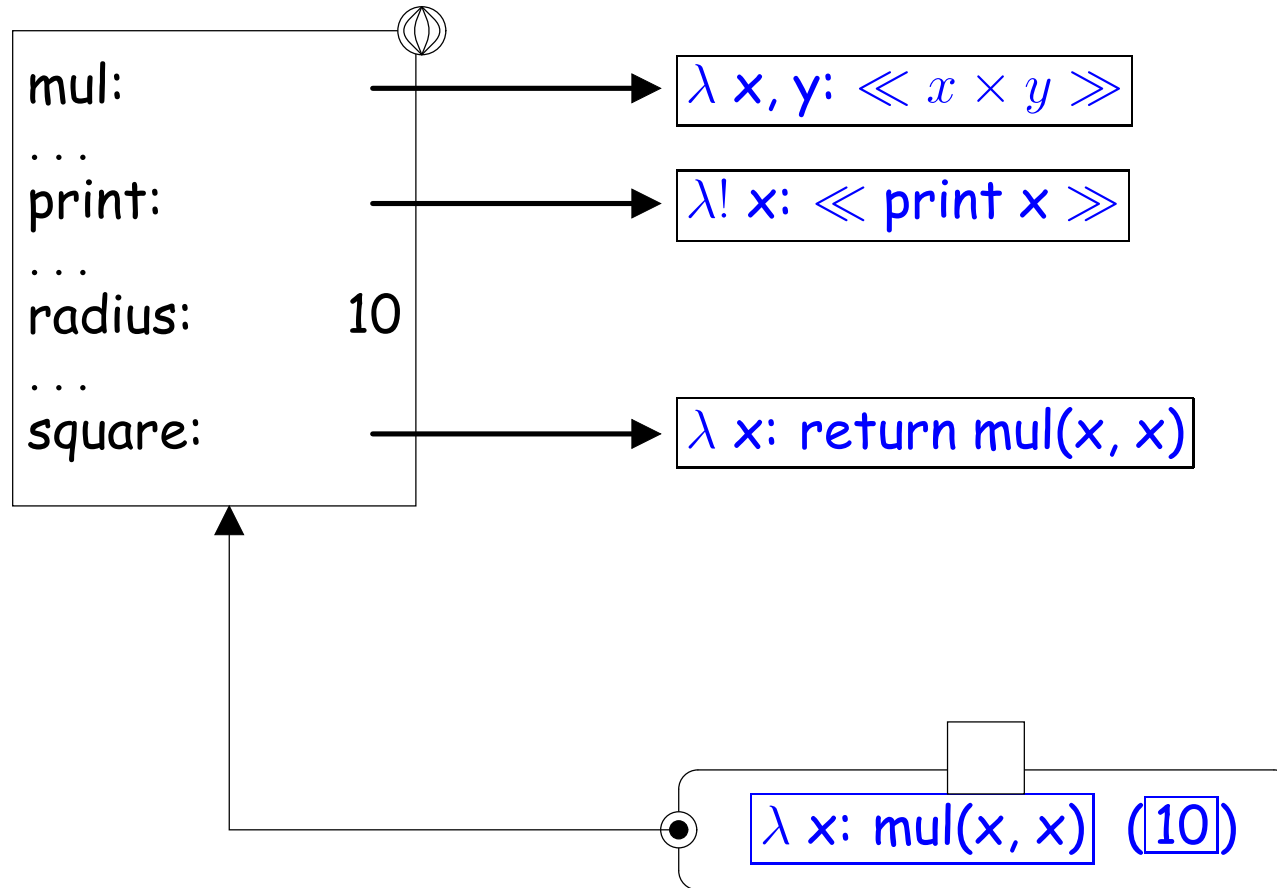
# Evaluation of Names

- To evaluate a name (identifier) in an environment, look for what that name *"is bound to"* in that environment.
- For example, in this situation...



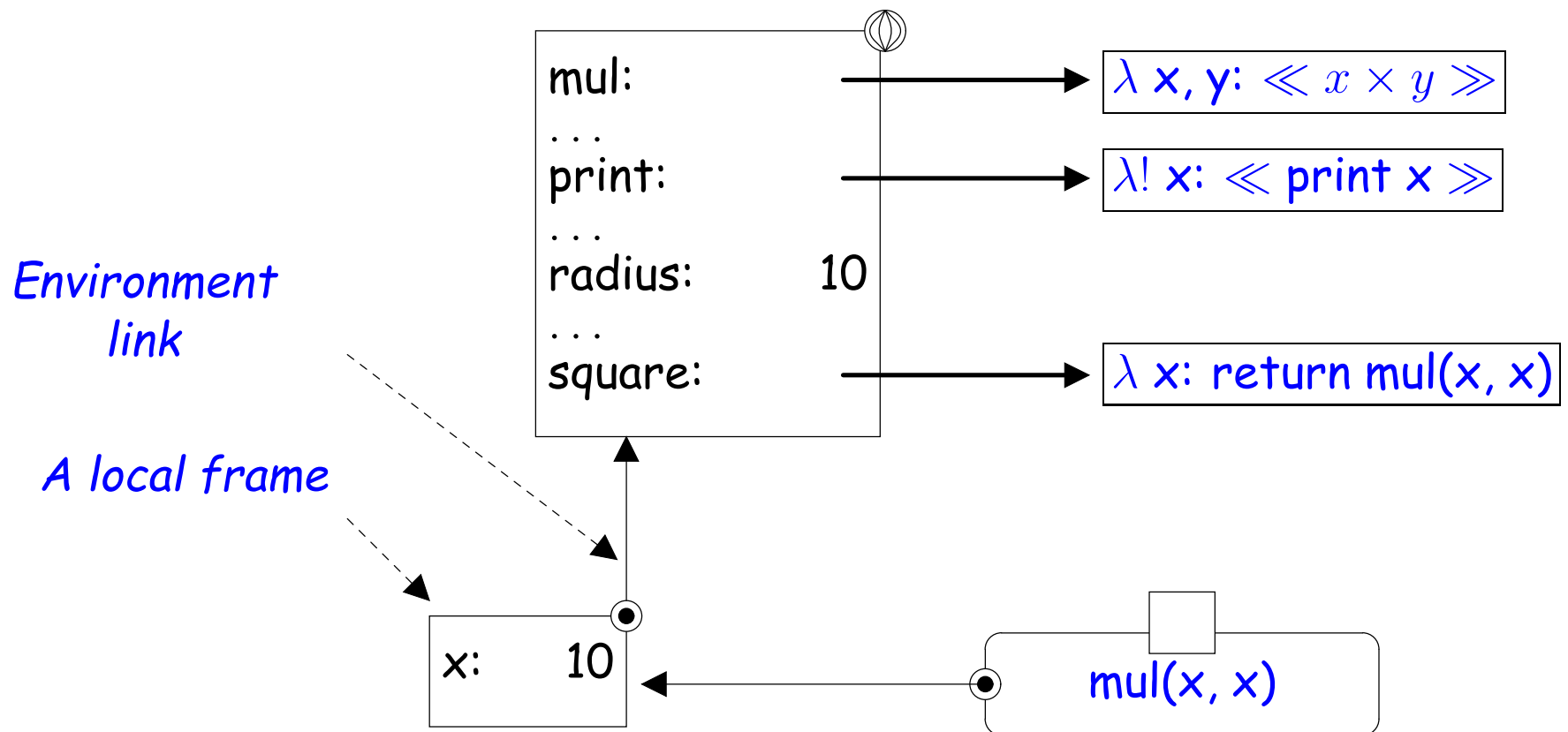
## Evaluation of Names (II)

... We find the values for **square** and **radius** in the global frame (the big box with the globe on its upper right).

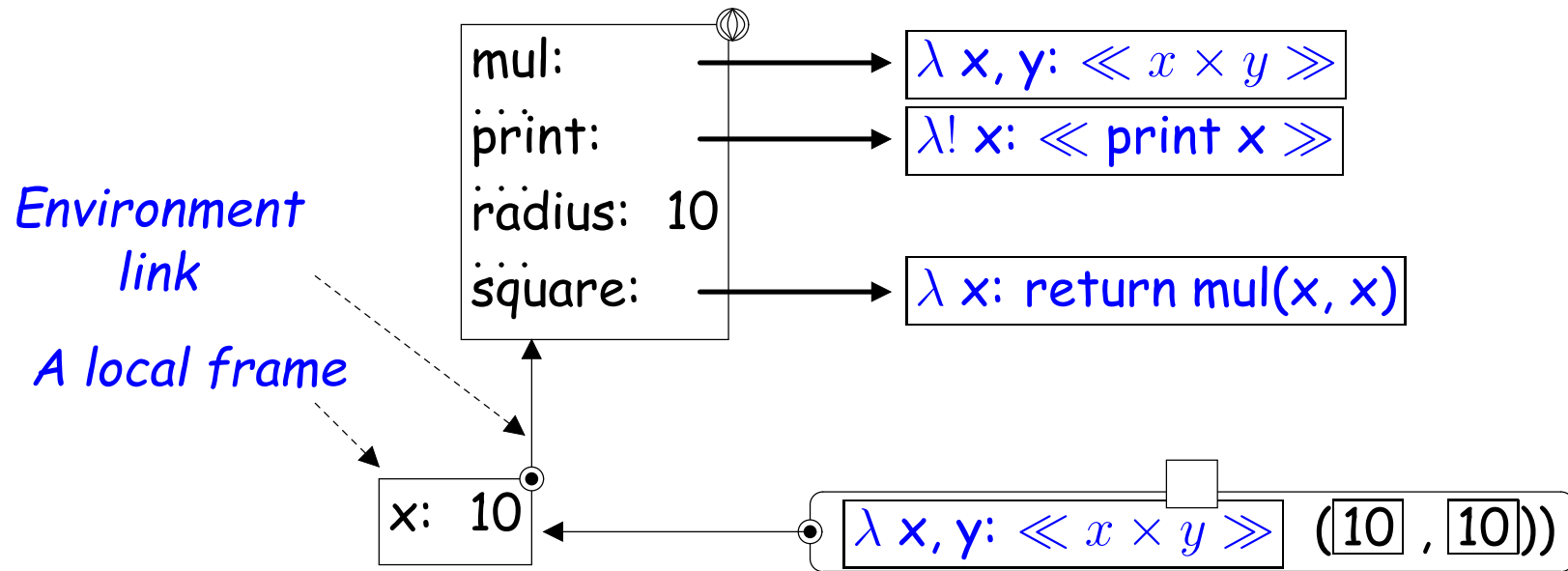


# Evaluation of Names: More Complicated Environments

- In general, as we'll see, environments consist of *chains* of frames.
- Here, we find the value of *x* in the small, "*local frame*"
- We don't find *mul*, there, so we must follow the "*environment link*" looking for it.



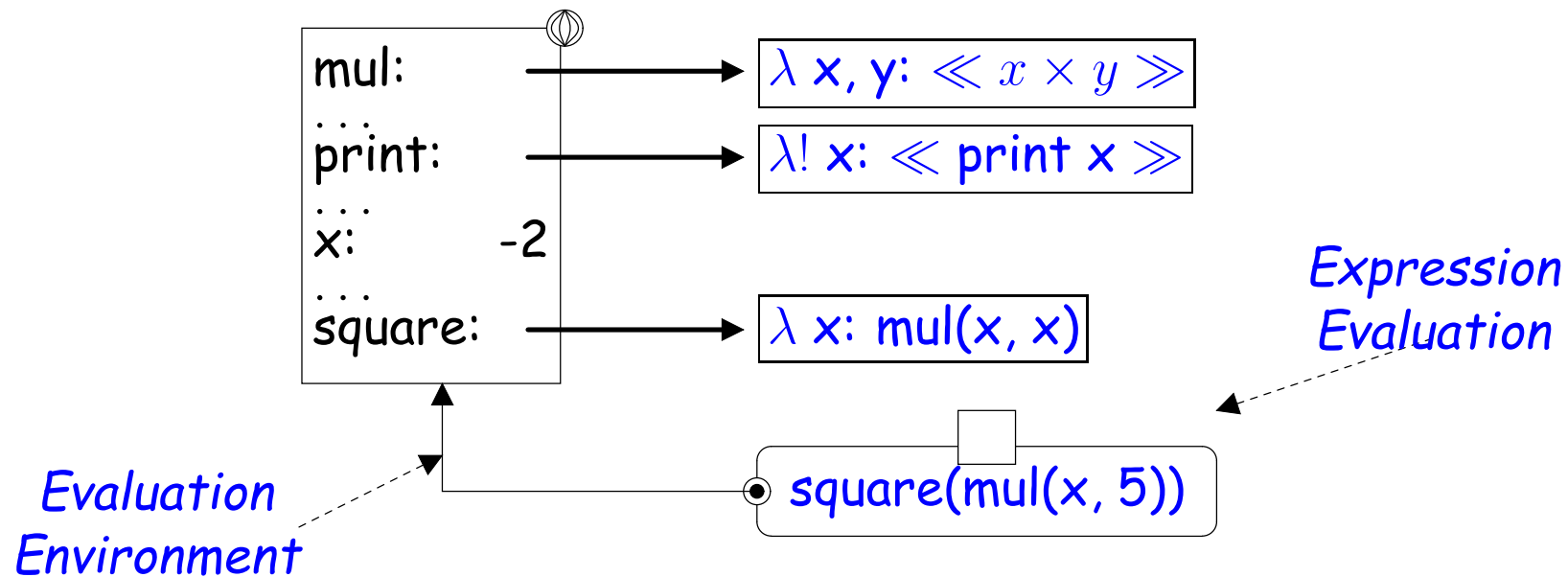
## More Complicated Environments (II)



# Evaluating User-Defined Function Calls

- Consider the expression `square(mul(x, x))` in

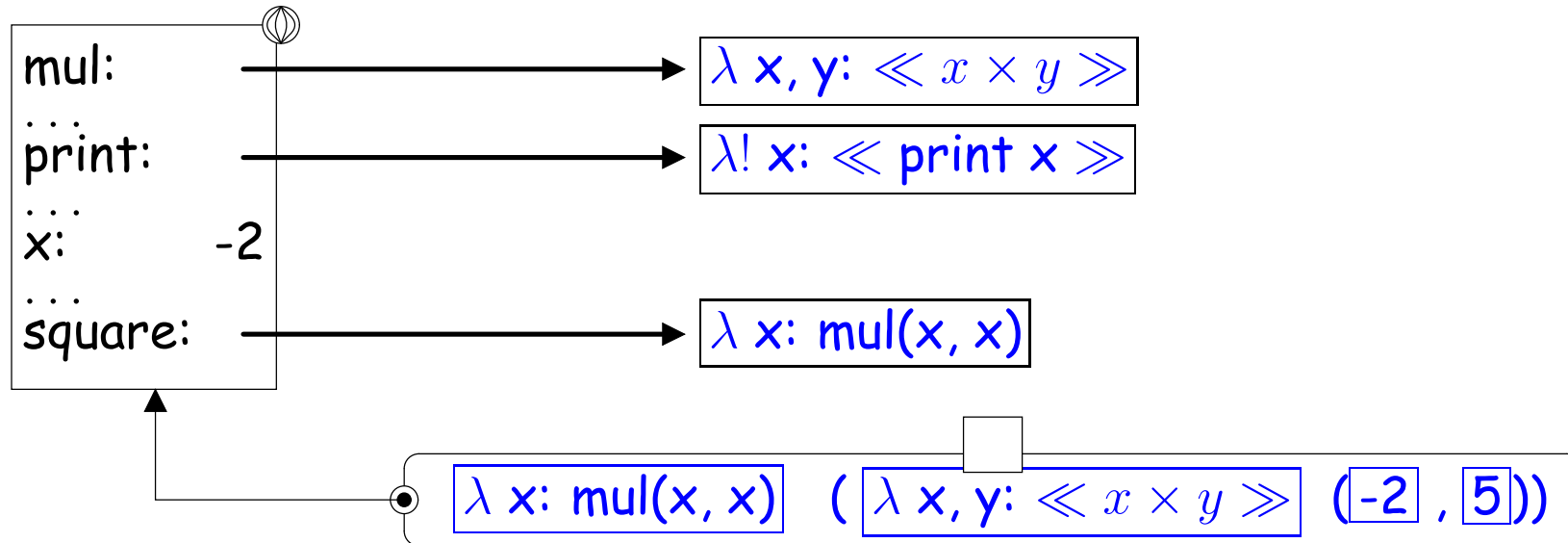
```
from operator import mul
def square(x):
    return mul(x, x)
x = -2
print(square(mul(x, 5)))
```





## Evaluating User-Defined Function Calls (II)

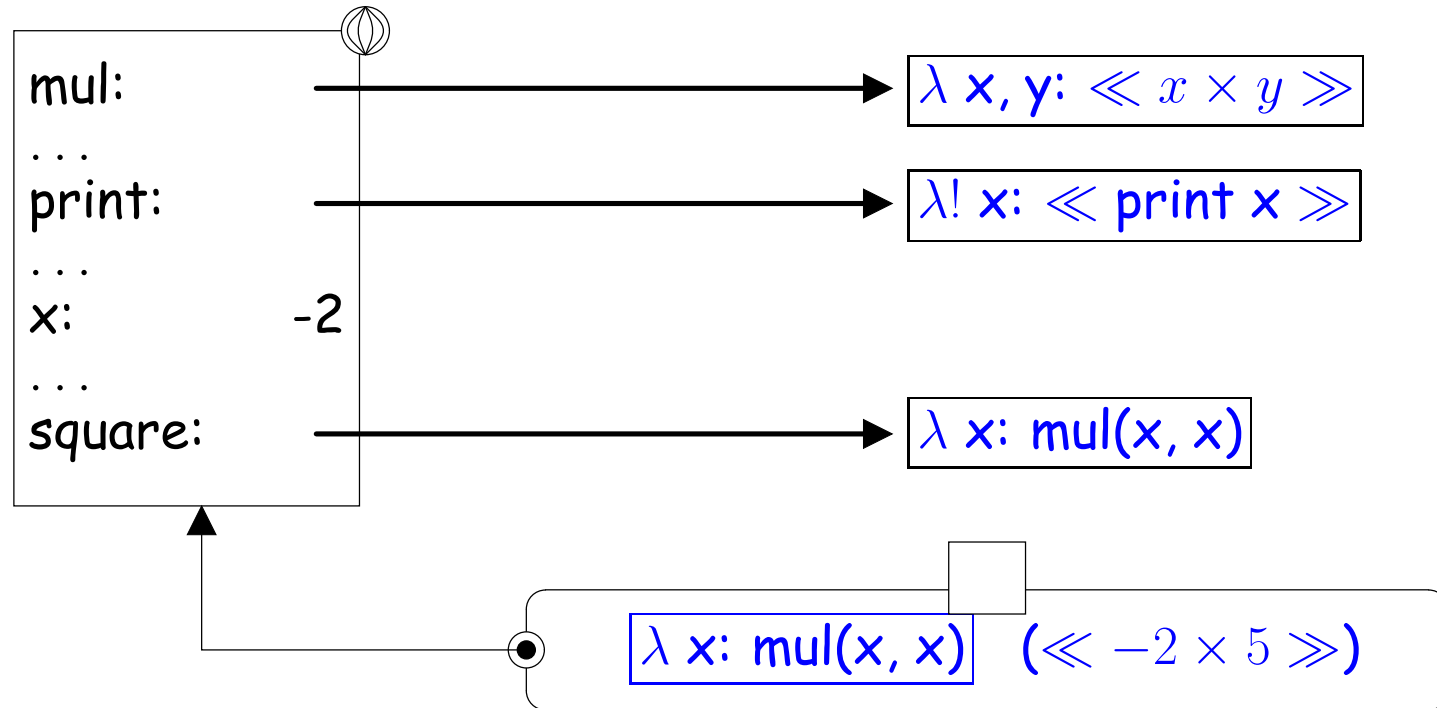
- First evaluate the subexpressions of `square(mul(x, x))` in the global environment:



- Evaluating subexpressions `x`, `mul`, and `square` takes values from the expression's environment.

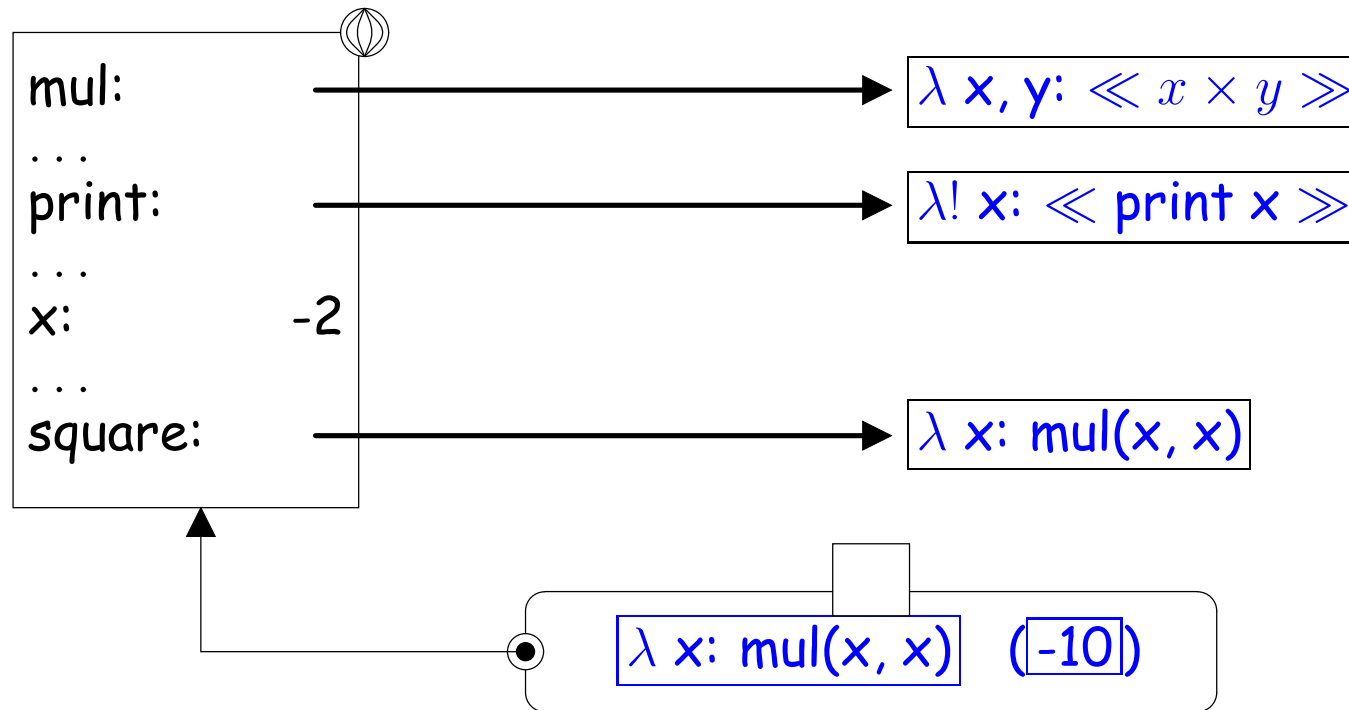
# Evaluating User-Defined Functions Calls (III)

- Then call the multiply function. Since this is primitive, let's just use the substitution model:



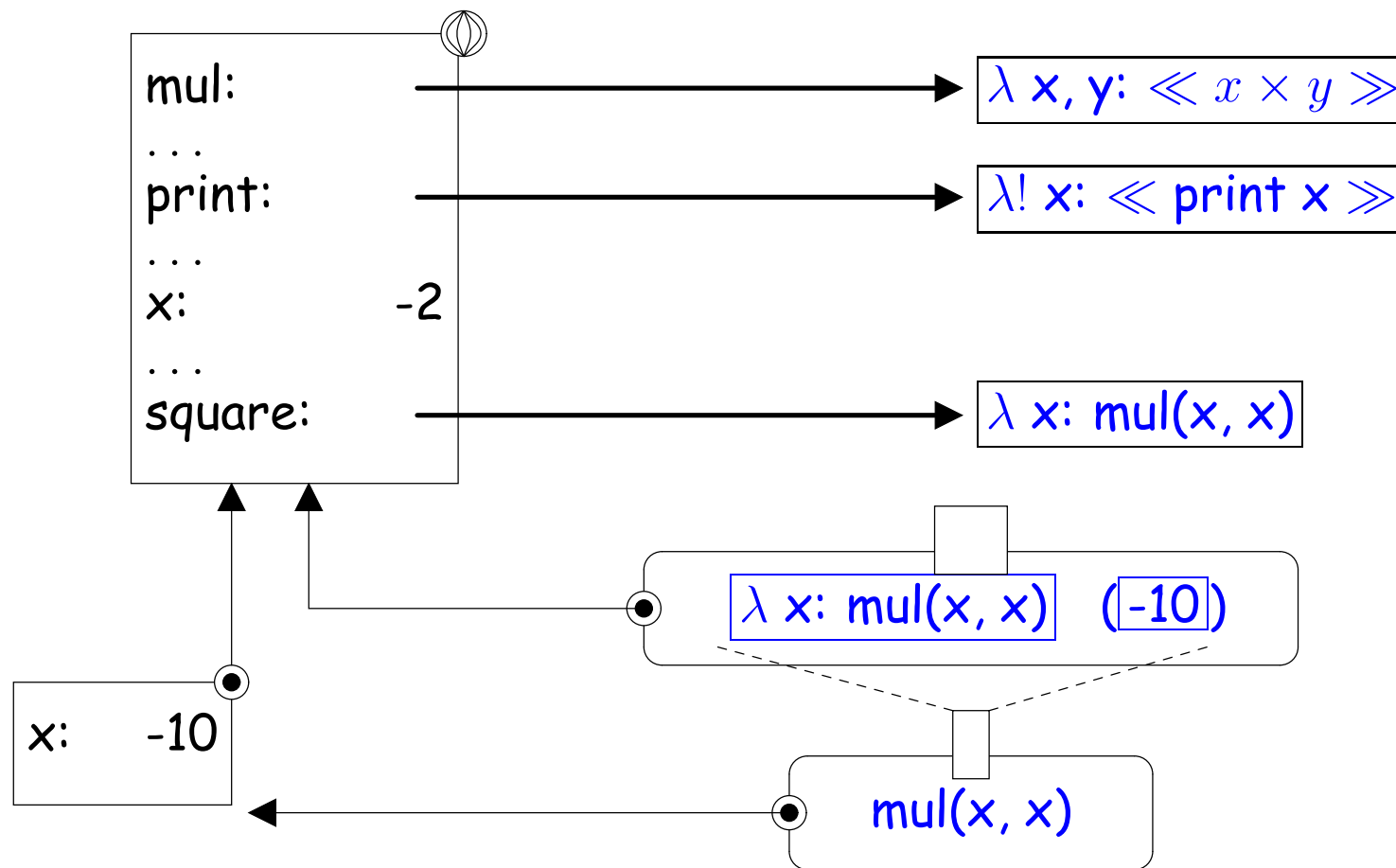
# Evaluating User-Defined Functions Calls (IV)

- Execute the primitive operation:



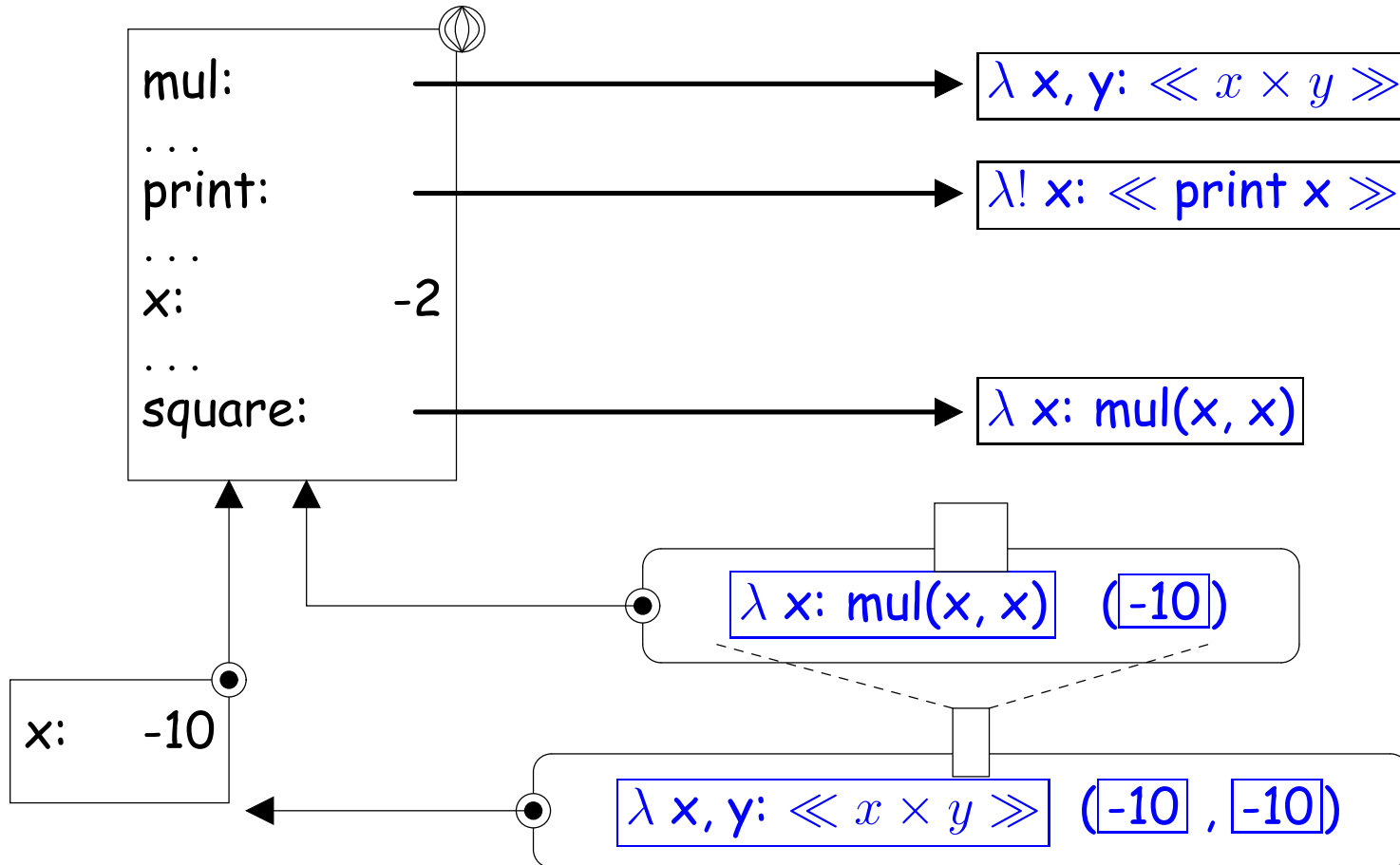
# Evaluating User-Defined Functions Calls (V)

- To evaluate the call to the user-defined function (*square*), start a new evaluation in a new *local environment frame*, attached to the frame where *square* was defined (the global frame here), and giving *x* the operand value.



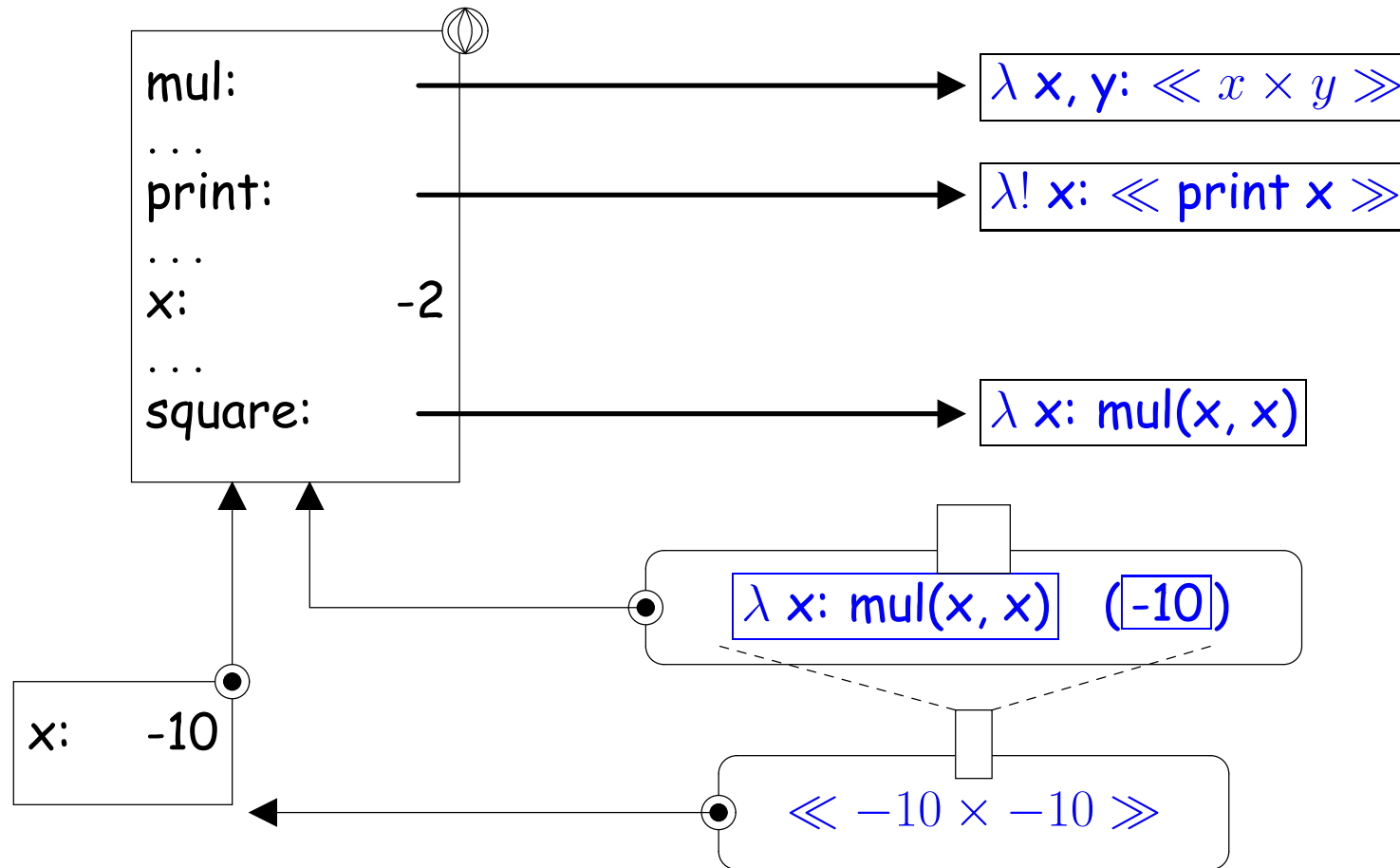
# Evaluating User-Defined Functions Calls (VI)

- When we evaluate `mul(x, x)` in this new environment, we get the same value as before for `mul`, but the local value for `x`.



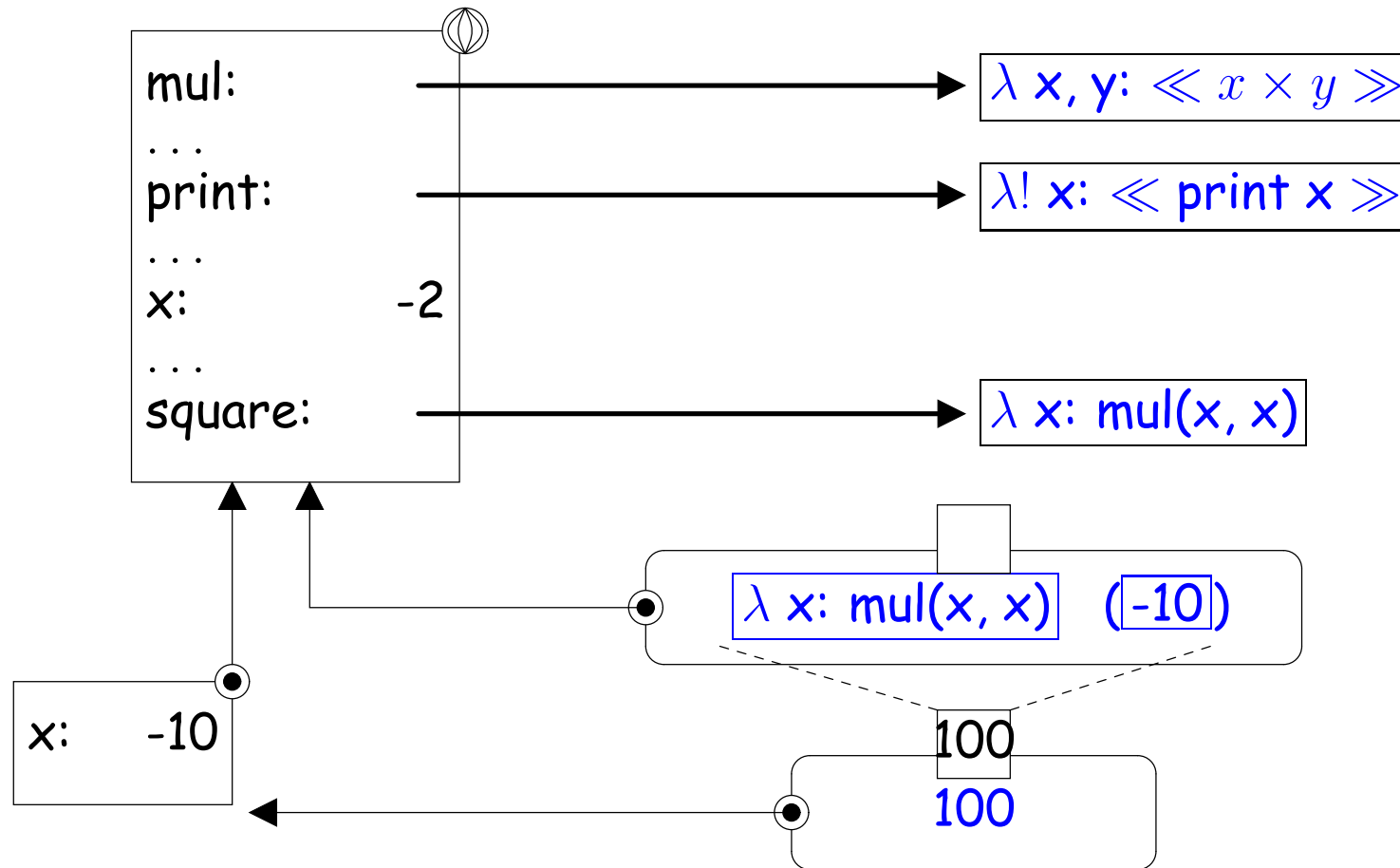
# Evaluating User-Defined Functions Calls (VII)

- Evaluate the primitive multiplication as before:



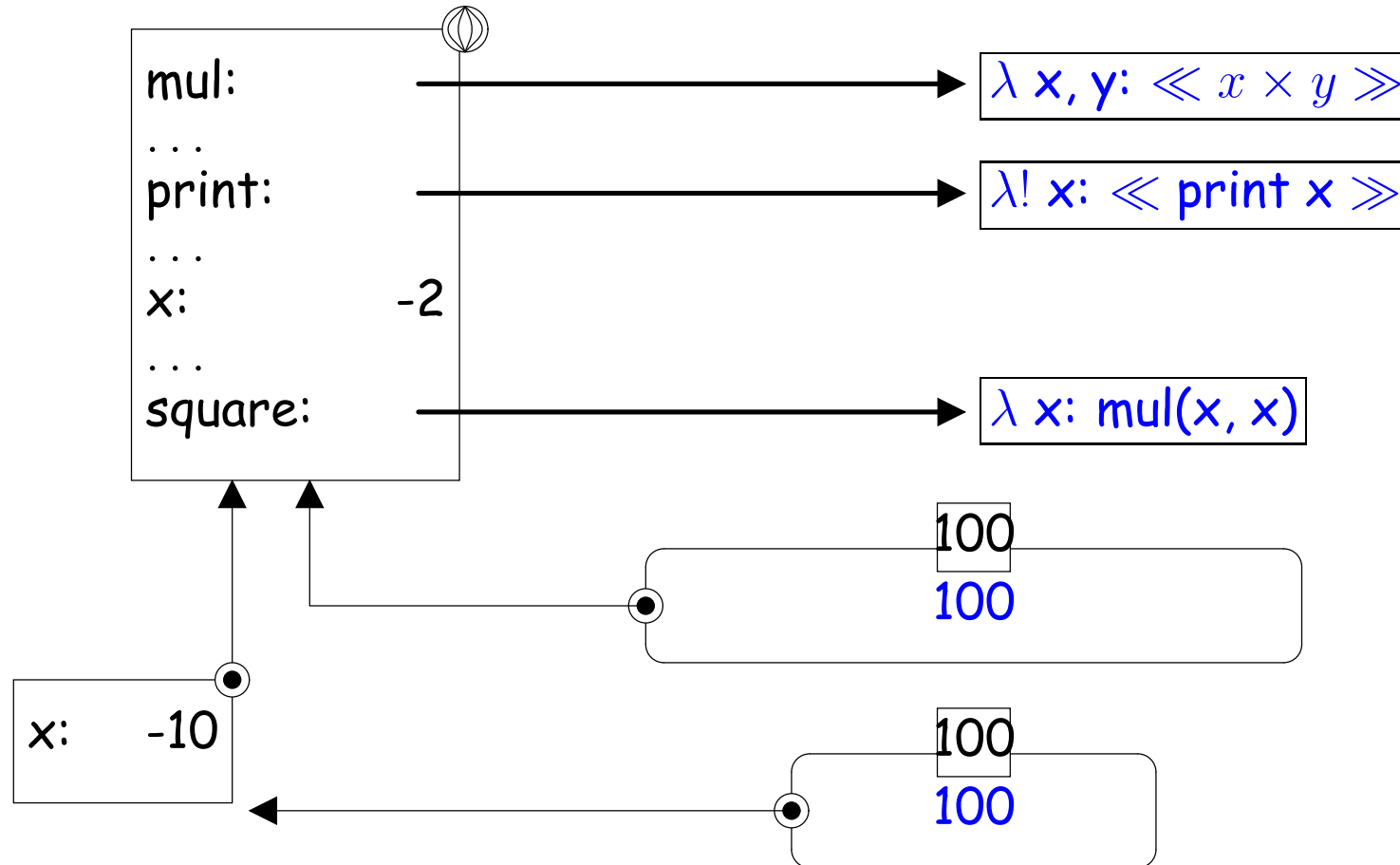
# Evaluating User-Defined Functions Calls (VIII)

- And return the finished value...



# Evaluating User-Defined Functions Calls (IX)

- ...replacing the call to the user-defined function and yielding the final value:

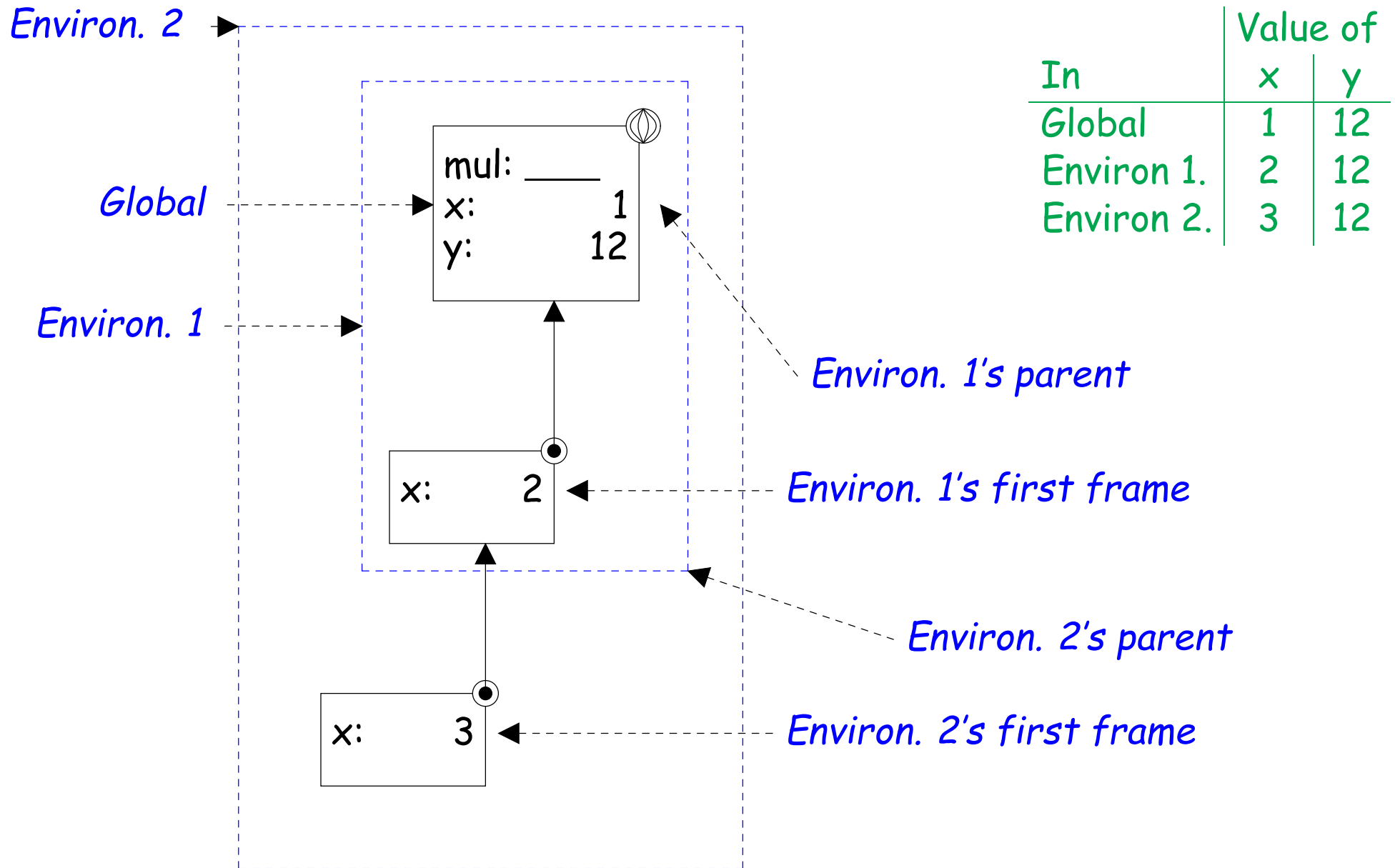




# Summary: Environments

- *Environments* map names to values.
- They consist of chains of *environment frames*.
- An environment is either a *global frame* or a first (local) frame chained to a *parent environment* (which is itself either a global frame or ...).
- We say that a name is *bound to* a value in a frame.
- The *value (or meaning) of a name* in an environment is the value it is bound to in the first frame, if there is one, ...
- ...or if not, the meaning of the name in the parent environment

# A Sample Environment Chain



# Environments: Binding and Evaluation

- Every expression and statement is evaluated (executed) in an environment, which determines the meaning of its names.
- Subexpressions (pieces) of an expression are evaluated in the same environment as the expression
- *Assigning* to a variable binds a value to it in (for now) the first frame of the environment in which the assignment is executed.
- *Def statements* bind a name to a function value in the first frame of the environment in which the *def* statement is executed.
- *Calling* a user-defined function creates a new local environment and binds the operand values in the call to the parameter names in that environment.

# Example: Evaluation of a Call: `sum_square(3,4)`

