Back to Basics

What is an object?

What is a class?

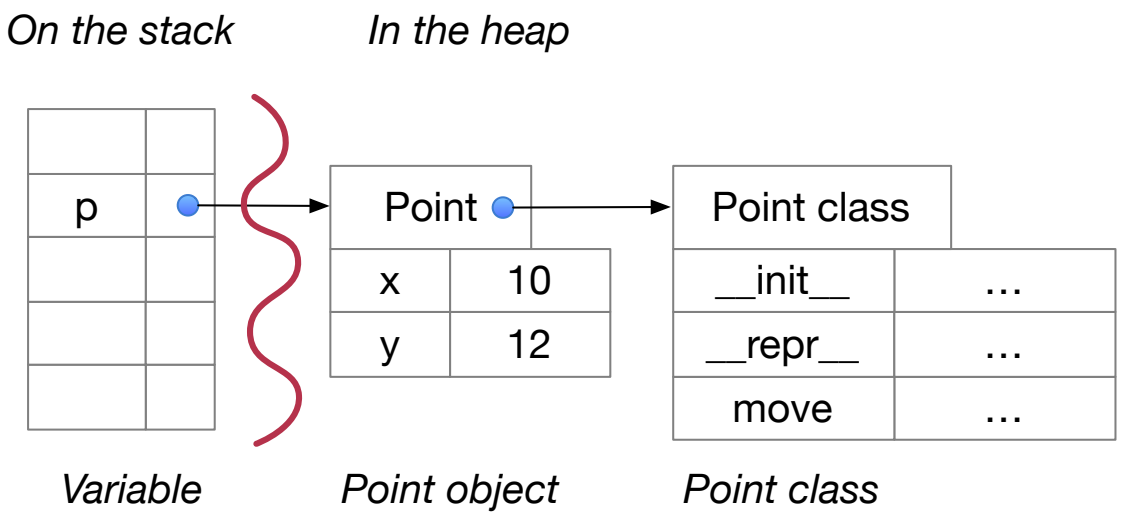
How does inheritance work?

How does overriding work?

What happens when I call foo.m(x,y)?

Some review here …

We looked at this in week 1 and 2 …



… but it’s time for a deeper dive

What's a class, really?

Essentially a table (dict) mapping method names to their definitions

The table includes inherited methods as well as methods defined directly in the class

```
>>> dir(Tile)
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__',
'__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '__str__',
'__subclasshook__', '__weakref__', 'add_listener', 'attend', 'could_be', 'eliminate',
'notify_all', 'set_value', 'unattend']
```

What's an object, really?

Essentially a table mapping names to field values.

Including a special field: __class__ is a reference to the class of the object

tile object

__class__	
row	2
col	2
candidates	set(["6"])
value	"6"

Tile class

__class__	<type>
__hash__	…
eliminate	…
attend	…
__str__	…

How does inheritance work?

Expr

__repr__	…
eval	

```
raise NotImplementedError("…")
```

Unop

__repr__	…
__init__	…
eval	

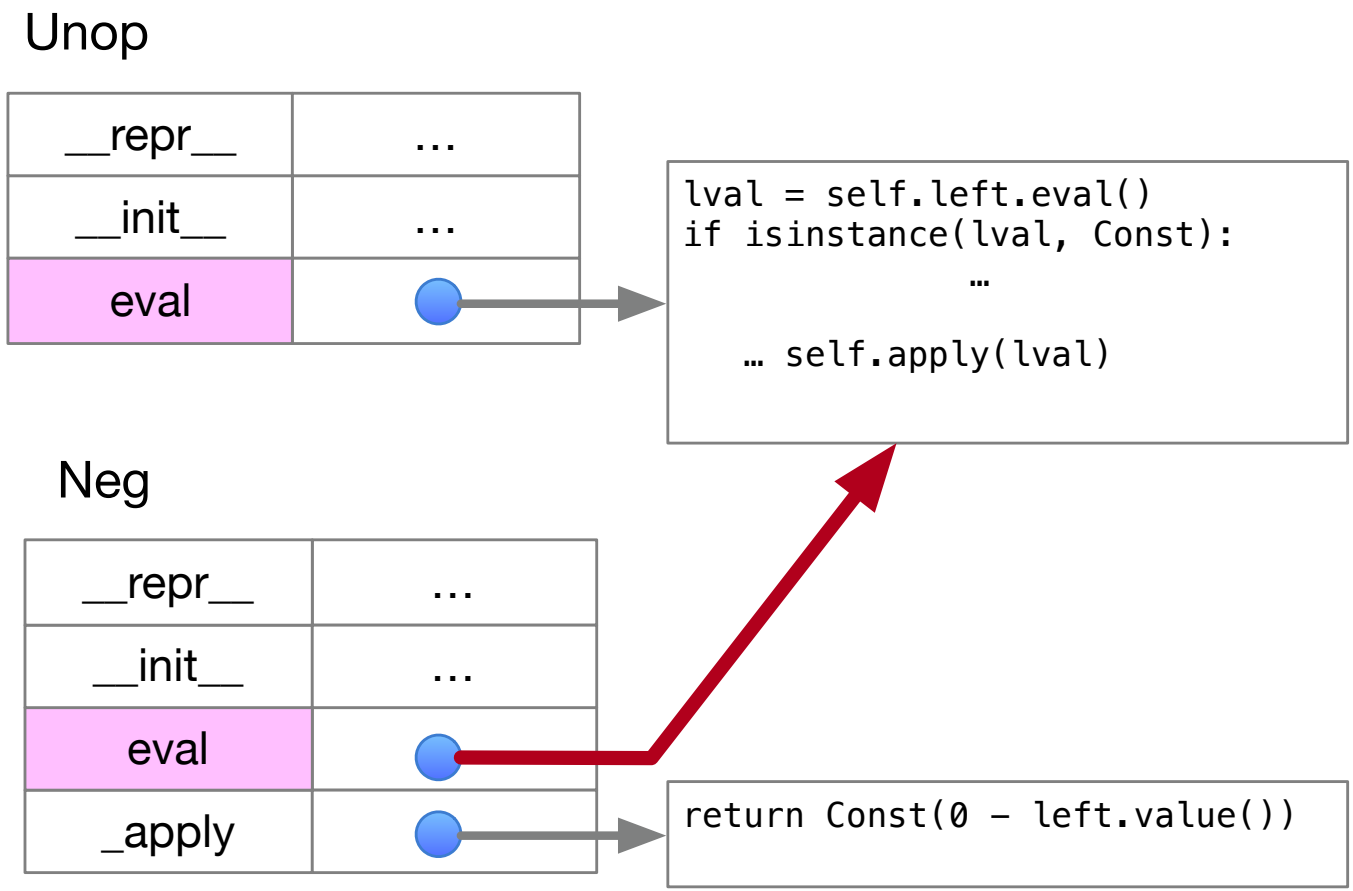
```
lval = self.left.eval()
if isinstance(lval, Const):
                …

   … self.apply(lval)
```

Neg

__repr__	…
__init__	…
eval	
_apply	

```
return Const(0 - left.value())
```

Inherit a method:



The method table of Neg references the inherited ”eval” method.
Inherited method references are just copied into subclasses.

Override a method

Expr

__repr__	…
eval	

```
raise NotImplementedError("…")
```

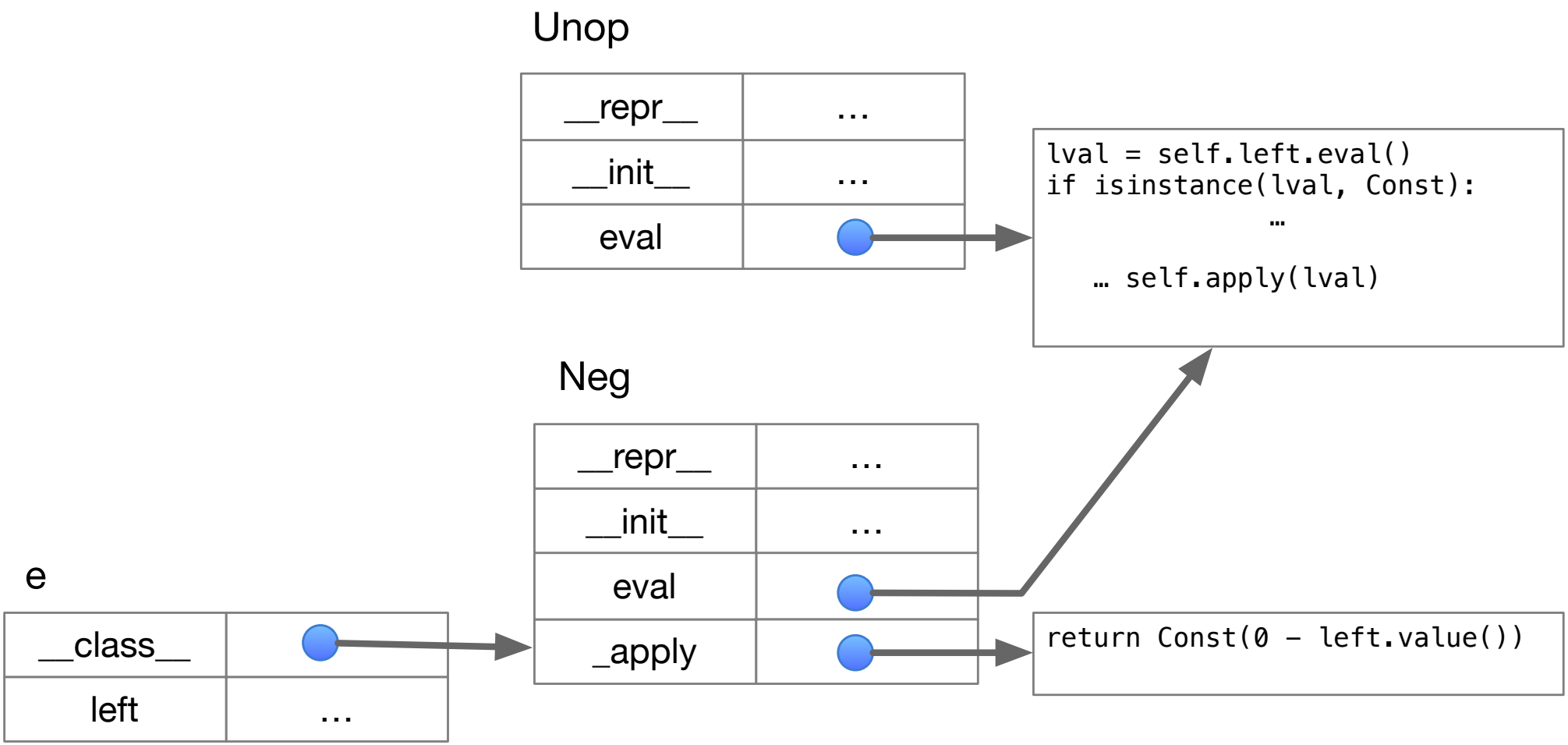
Unop

__repr__	…
__init__	…
eval	

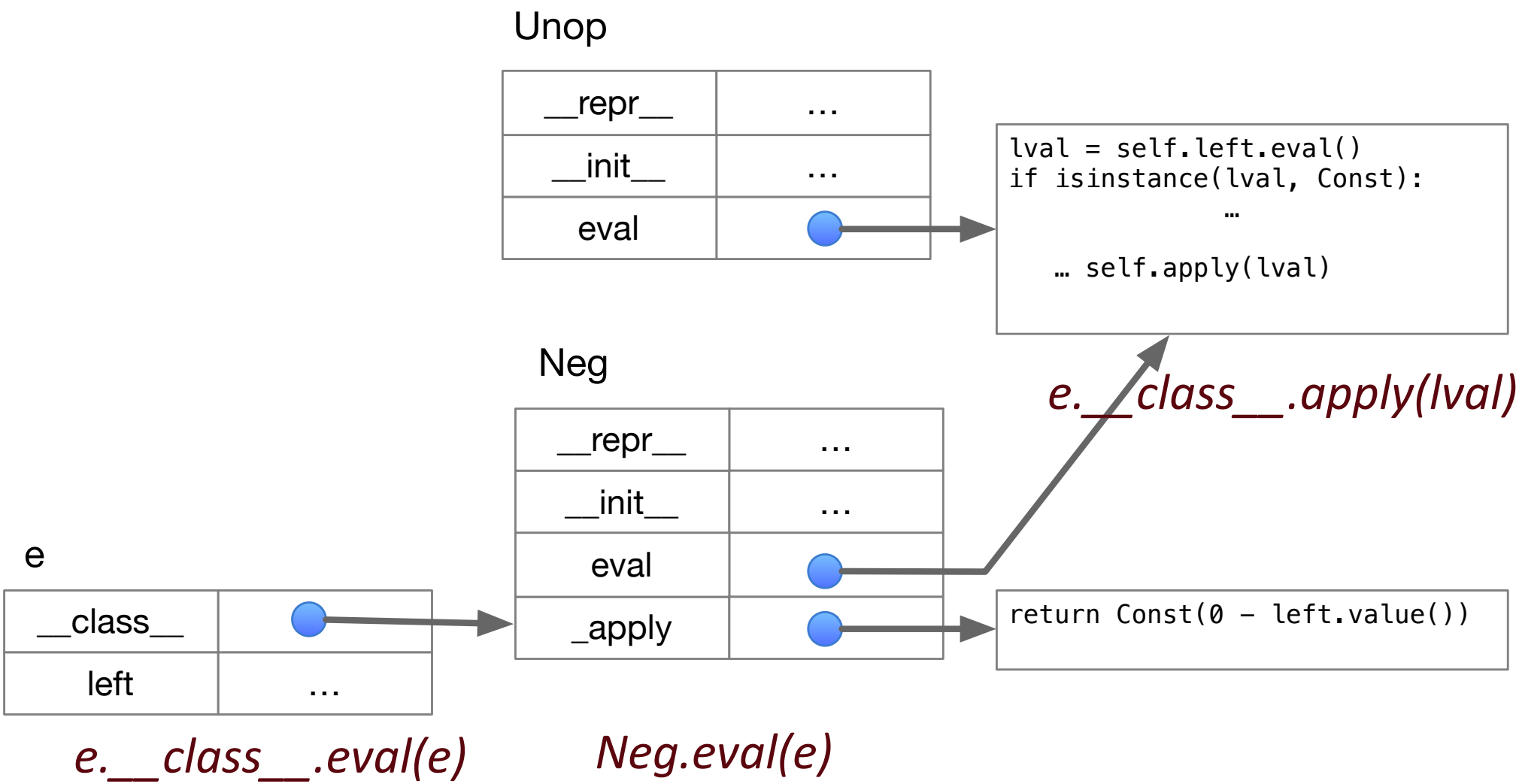
```
lval = self.left.eval()
if isinstance(lval, Const):
                …

   … self.apply(lval)
```

Subclass method table has an entry with the same name, but a different method body

Method call: e.eval()



Method call: e.eval()



Summary

Class is basically a table of functions

- Subclass inherits by copying the table
- Subclass overrides by replacing table entries

Object has a reference to the class

Method call o.foo() is really o.__class__.foo(o)

- (object o becomes the “self” argument)

And that’s really it! Very simple implementation. (Similar to Java, C++, ...)