SELF

the power of simplicity

Rolph Recto + Jonathan DiLorenzo

Great Works in PL

April 30, 2019

SELF: The Power of Simplicity

David Ungar, Stanford
Randall B. Smith, Xerox PARC

OOPSLA 87

1967	**Simula67**	Dahl and Nygaard
1980	**Smalltalk-80**	Kay, Ingalls, and Goldberg
1985	**C++**	Stroustrup
1987	**Self**	Ungar and Smith
1991	**Java**	Gosling, Sheridan, and Naughton
1995	**Javascript**	Eich

> Is JavaScript popular? It’s hard to say. Some Ajax developers profess (and demonstrate) love for it. Yet many curse it, including me. **I still think of it as a quickie love-child of C and Self.**

Brendan Eich

https://brendaneich.com/2008/04/popularity/

principles of Self

everything is an object

prototypes, not classes

all interactions are message passing

everything is an object

Smalltalk

primitive values

methods and closures

control structures

classes

```
((4 fac) between: 10 And: 100) ifTrue: “Hi!” False: ”Bye!”
```

call “**fac**” method on **4**, return **24**

call “**between:And:**” on **24** with args **10** and **100**, return **true**

call “**ifTrue:False:**” on **true** with args **“Hi!”** and **“Bye!”**, return **“Hi!”**

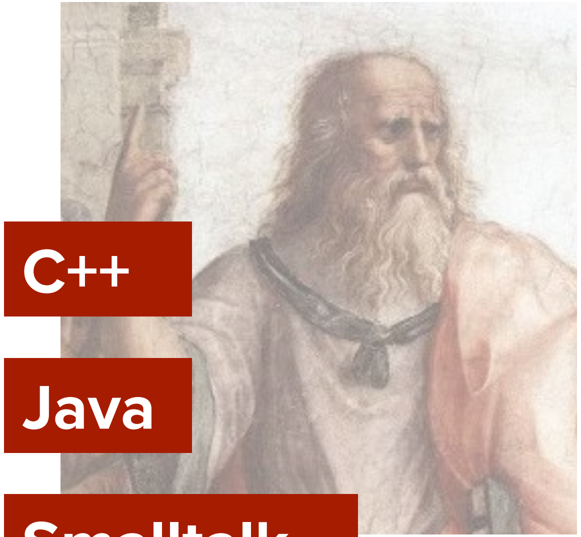
objects are instances of classes

Smalltalk

objects are clones of prototypes

Self

objects are instances of classes

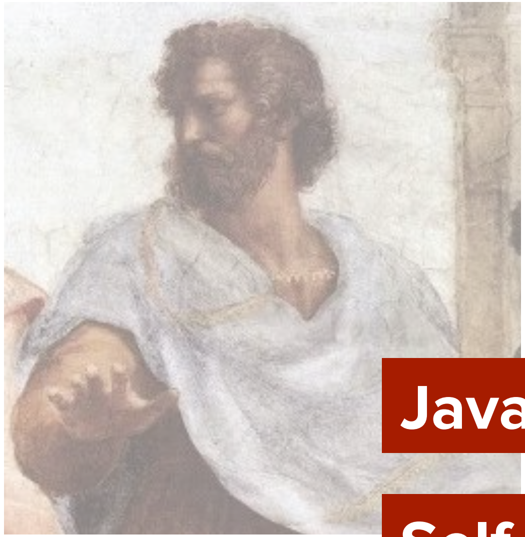


C++

Java

Smalltalk

objects are clones of prototypes

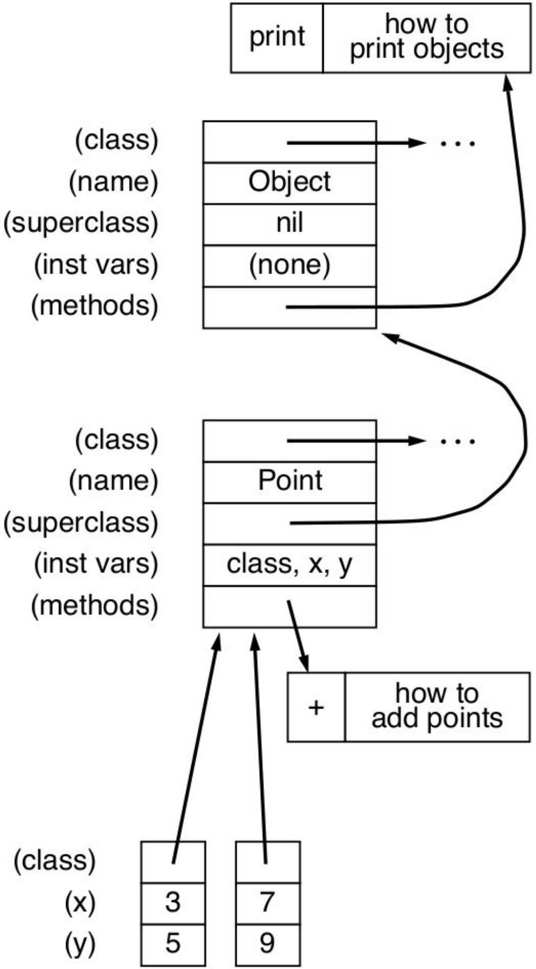


Javascript

Self

classes	prototypes
create objects by calling class constructor	create objects by cloning prototype
can modify methods only by subclassing	objects can have unique methods and fields
classes need metaclasses, etc. (infinite regress!)	no classes, no infinite regress

classes



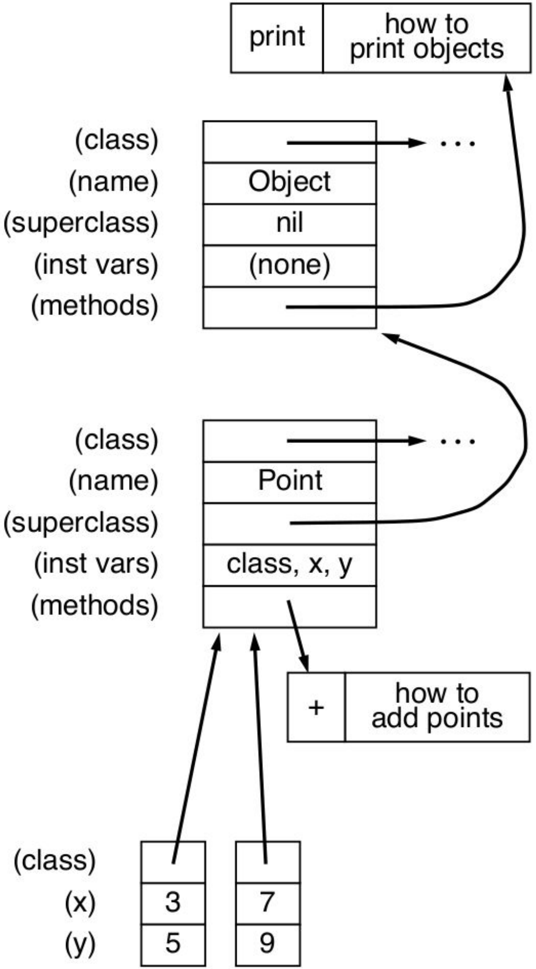
```
p := (Point new) x: 7 y: 9
p print
```

follow p’s class pointer, check if print is defined there

not defined there, so follow superclass pointer

found “print” in Object class! Invoke with receiver “p”

classes



```
p := (Point new) x: 1 y: 10
p print
```

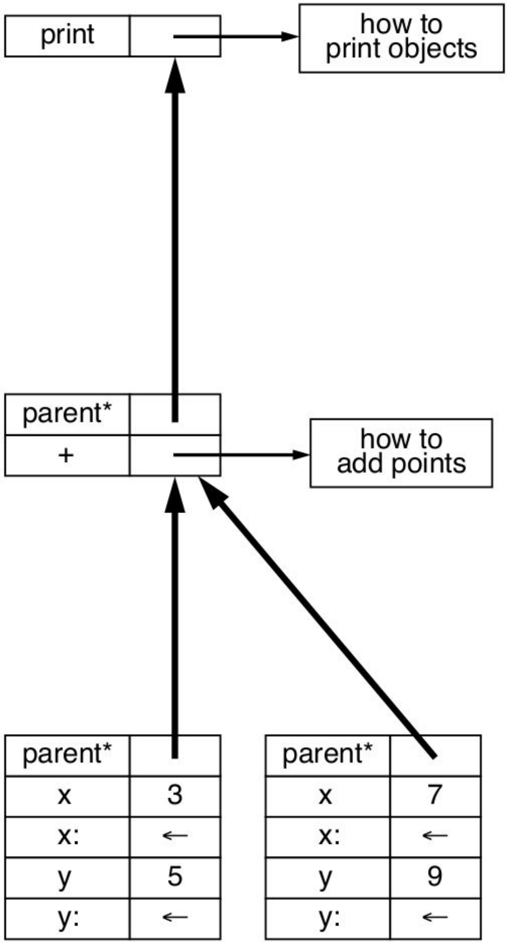
follow p’s class pointer, check if print is defined there

not defined there, so follow superclass pointer

found “print” in Object class! Invoke with receiver “p”

to have different print method, need to create Point subclass

prototypes



print	

how to print objects

parent*	
+	

how to add points

parent*	
x	3
x:	←
y	5
y:	←

parent*	
x	7
x:	←
y	9
y:	←

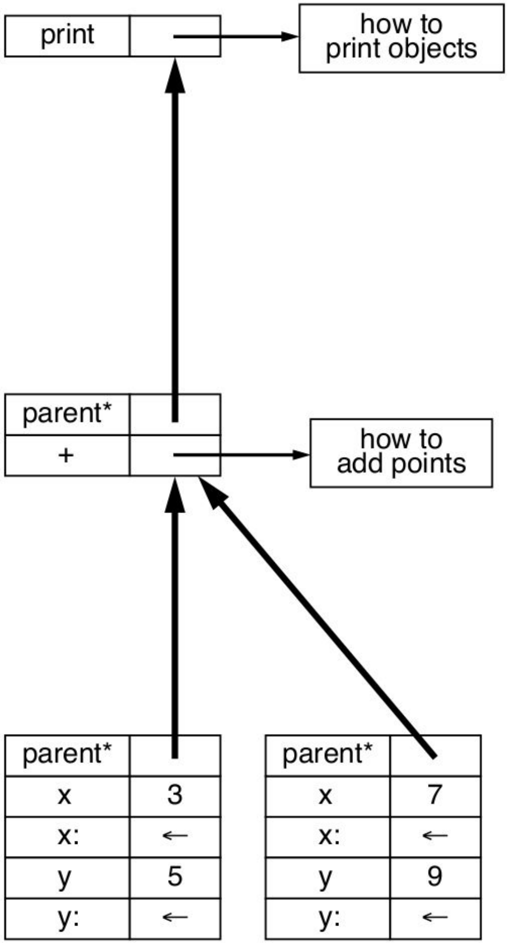
```
p:= (point clone) x: 7 y: 9
p print
```

does p have print method? no, so follow parent pointer to delegate

does Point delegate have “print”? no, so follow parent pointer to delegate

does Object delegate have print? yes, invoke with “p” as receiver

prototypes



```
p:= (point clone) x: 1 y: 10
p print
```

does p have print method? no, so follow parent pointer to delegate

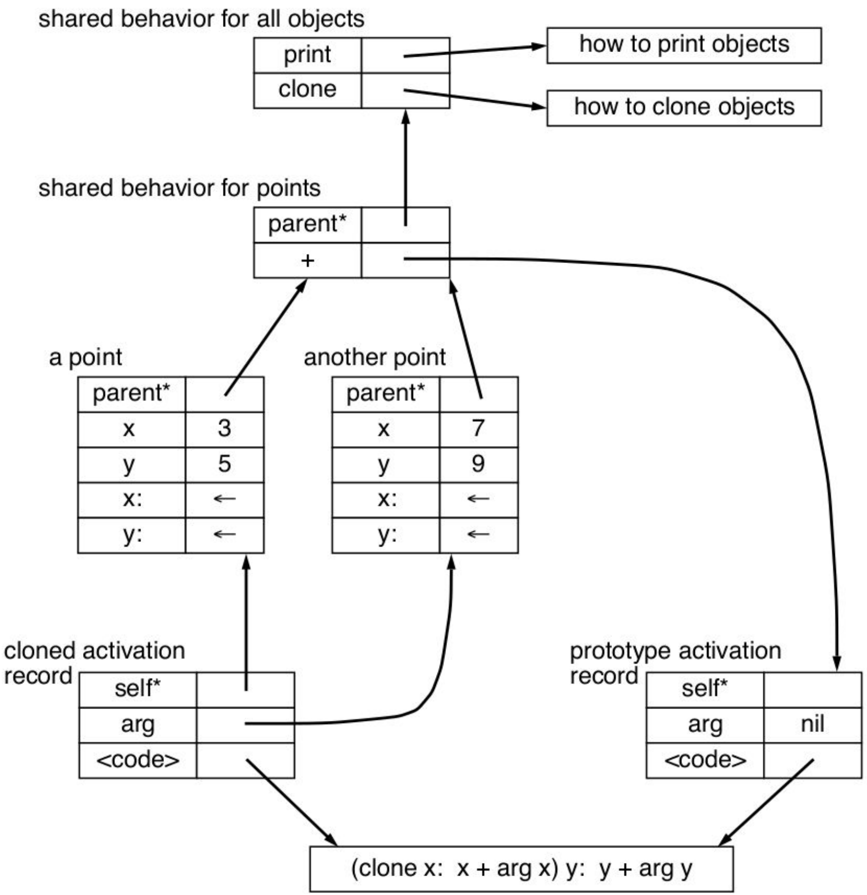
does Point delegate have “print”? no, so follow parent pointer to delegate

does Object delegate have print? yes, invoke with “p” as receiver

to have special print method for p, define new slot in p -- no subclass needed!

activation as cloning

method invocation clones prototype activation record



shared behavior for all objects

print	
clone	

how to print objects

how to clone objects

shared behavior for points

parent*	
+	

a point

parent*	
x	3
y	5
x:	←
y:	←

another point

parent*	
x	7
y	9
x:	←
y:	←

cloned activation record

self*	
arg	
<code>	

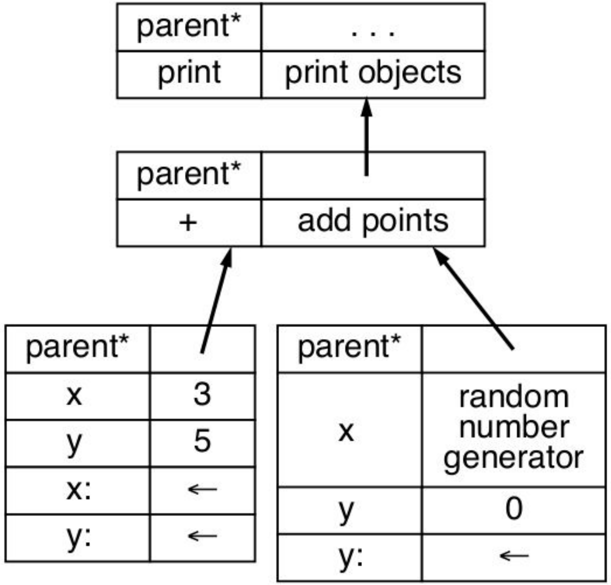
prototype activation record

self*	
arg	nil
<code>	

(clone x: x + arg x) y: y + arg y

state as behavior

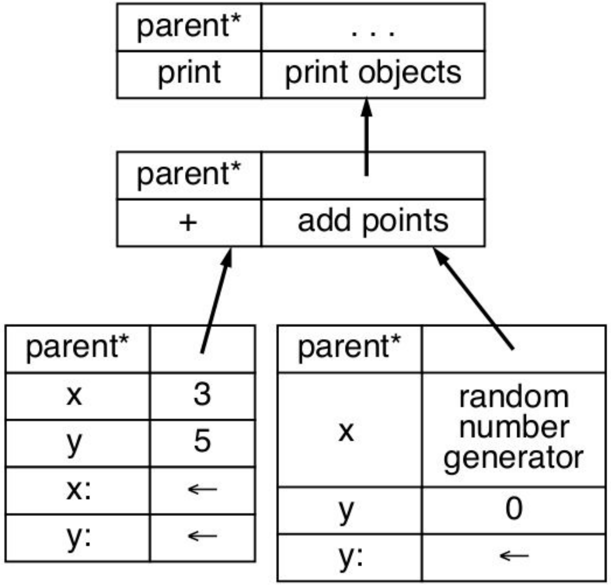
field access and assignment are messages to current receiver (self)



state as behavior

```
p x          // p.x

p x: 2       // p.x = 2
```



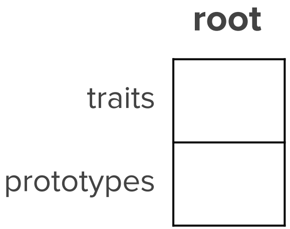
parent*	. . .
print	print objects

parent*	
+	add points

parent*	
x	3
y	5
x:	←
y:	←

parent*	
x	random number generator
y	0
y:	←

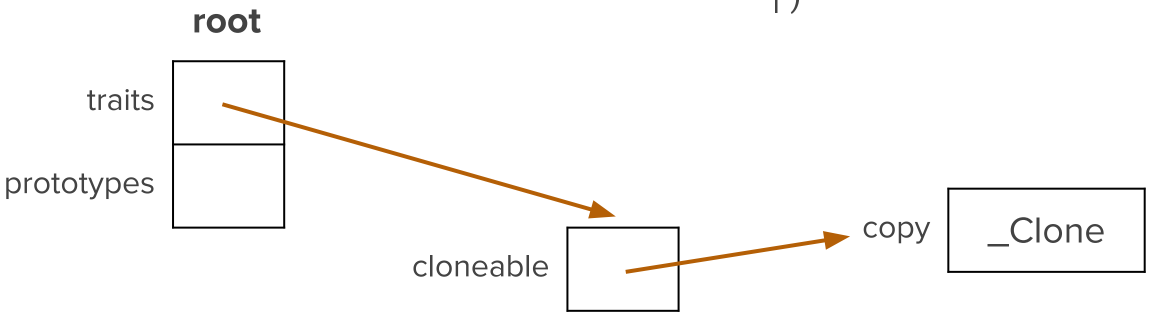
example: points



```
_AddSlotsIfAbsent: (|
  traits = ().
  prototypes = ().
|)
```

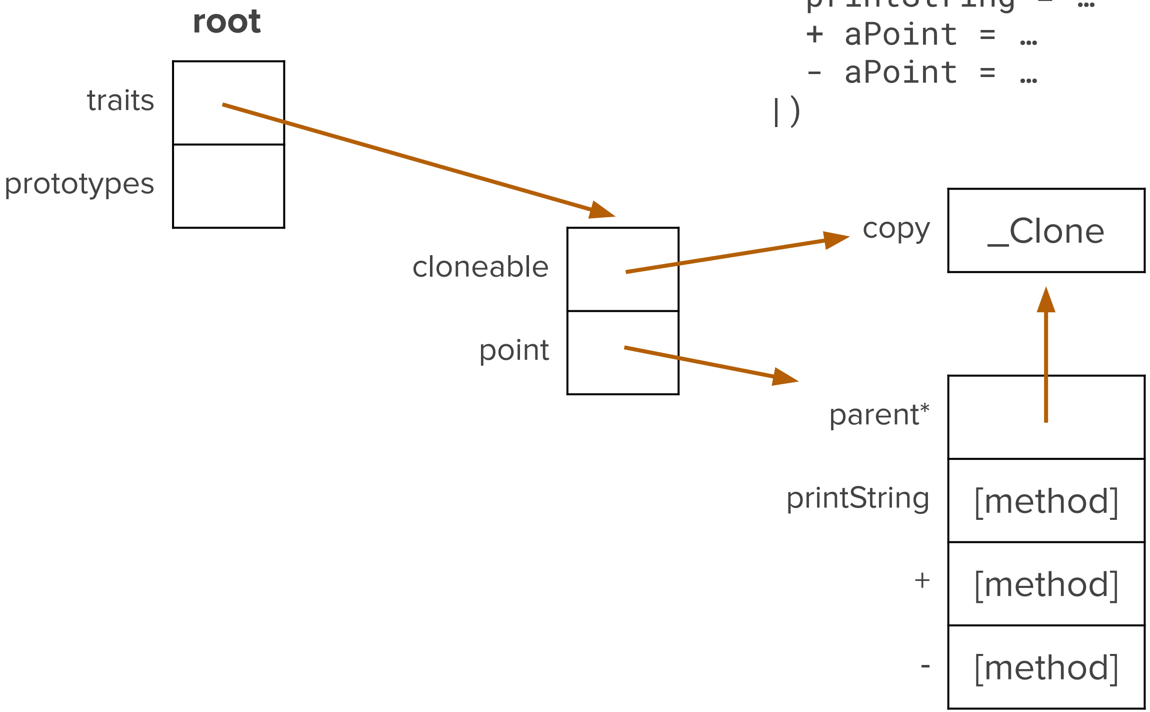
example: points

```
traits _AddSlotsIfAbsent:(|cloneable=()|)
traits cloneable _Define:(|
  copy = (_Clone).
|)
```



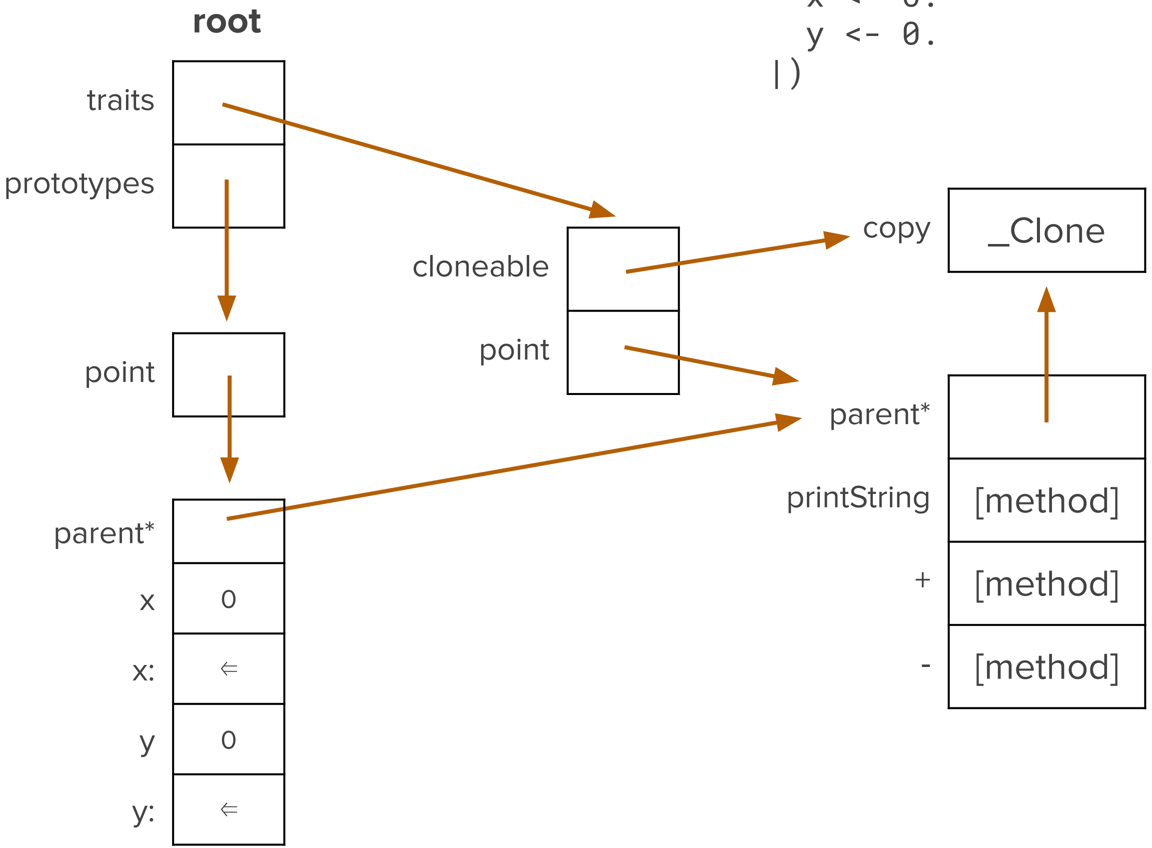
example: points

```
traits _AddSlotsIfAbsent: (|point=()|)
traits point _Define:(|
  parent* = traits cloneable.
  printString = …
  + aPoint = …
  - aPoint = …
|)
```



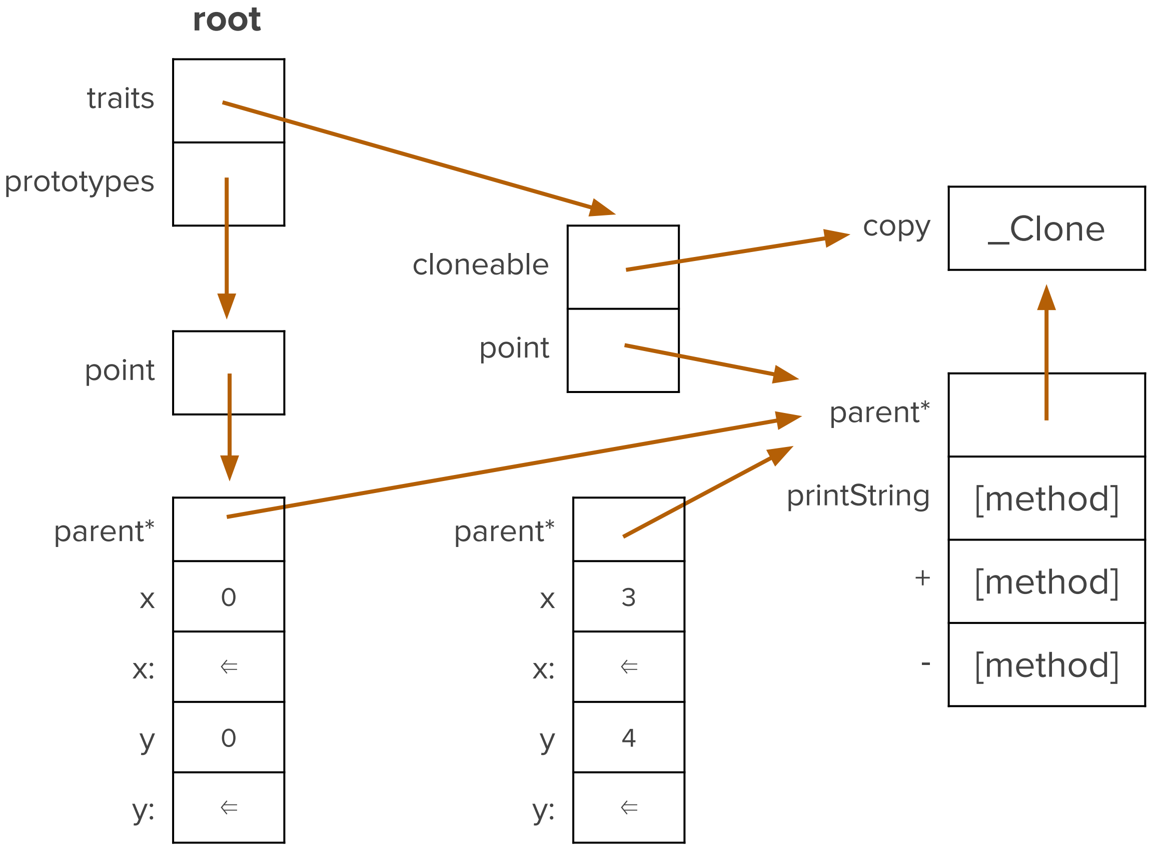
example: points

```
prototypes _AddSlotsIfAbsent (|point=()|)
prototypes point _Define:(|
  parent* = traits point.
  x <- 0.
  y <- 0.
|)
```



example: points

```
((prototypes point) copy) x: 3 y: 4
```



discussion

is Self a good influence on modern languages?

what are the tradeoffs of Self’s flexibility?

are there cases when simplicity should be abandoned?