Objects

Edward Z. Yang

What are the central ideas of OO?

dynamic dispatch • encapsulation • subtyping • inheritance

How can we understand OO with the tools of this class?

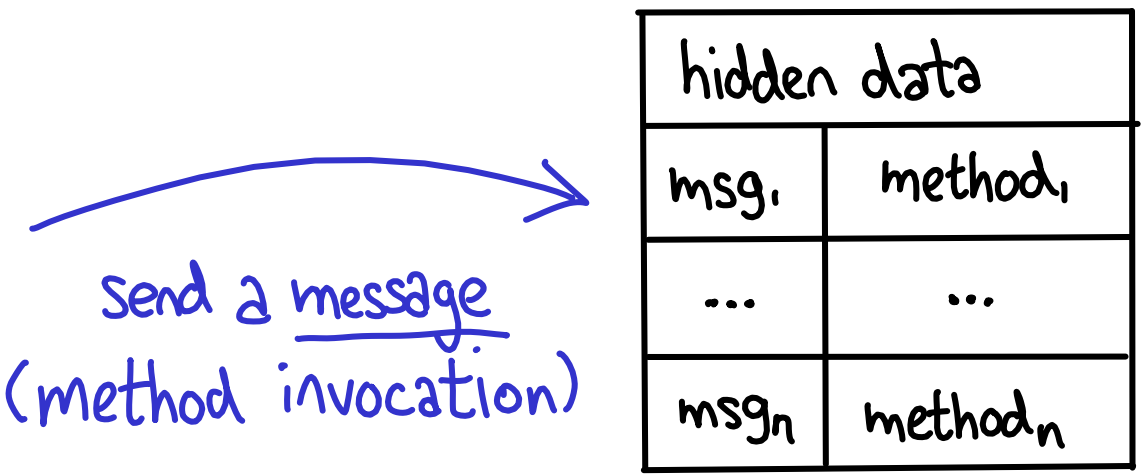
by reducing objects to known concepts (Simula)

by simplifying objects to a core idea (Smalltalk, Self)

What are the central ideas of OO?

dynamic dispatch • encapsulation • subtyping • inheritance

Anatomy of an object



compare with ADTs: behavioral rather than structural

Dynamic dispatch

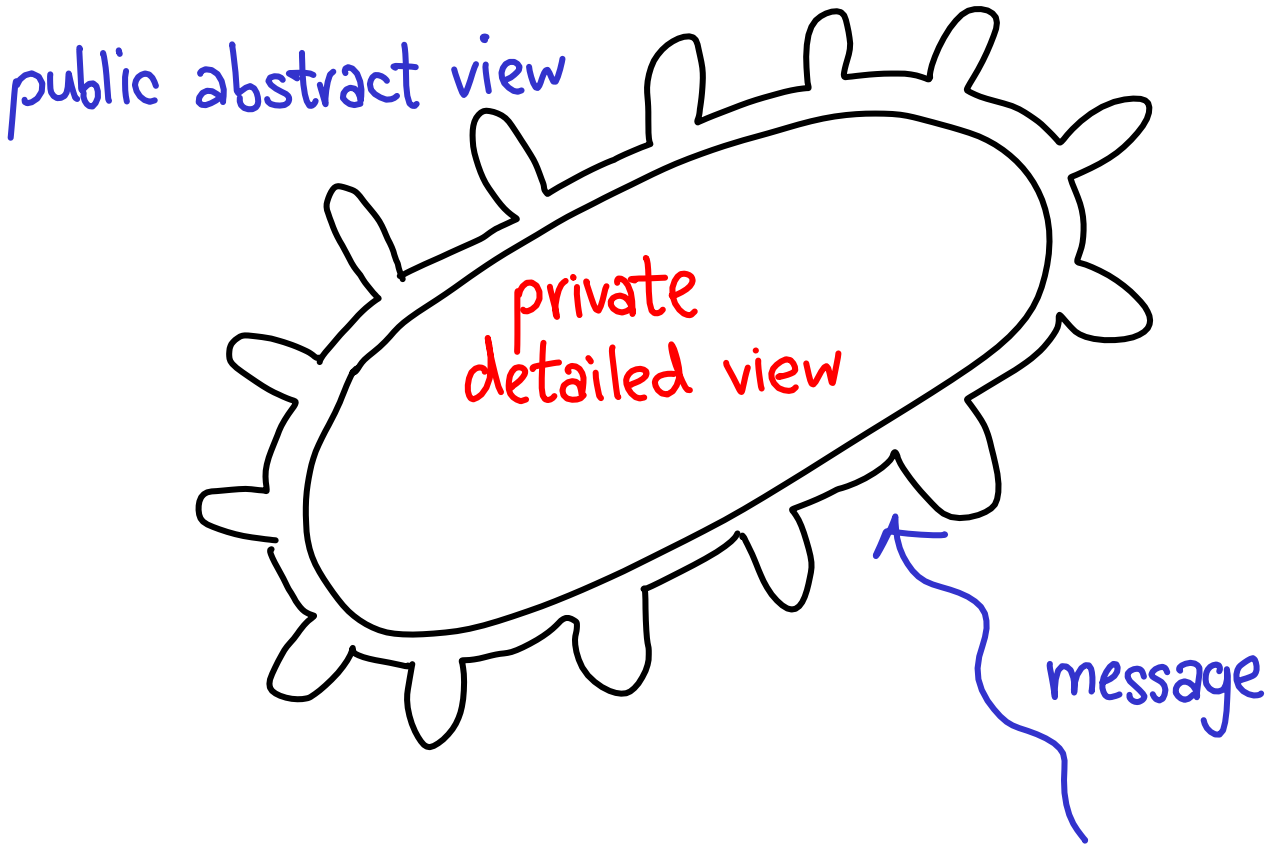
object→message(arguments) $\xrightarrow{\text{dynamic}}$ object & message

operation(arguments) $\xrightarrow{\text{static}}$ operation

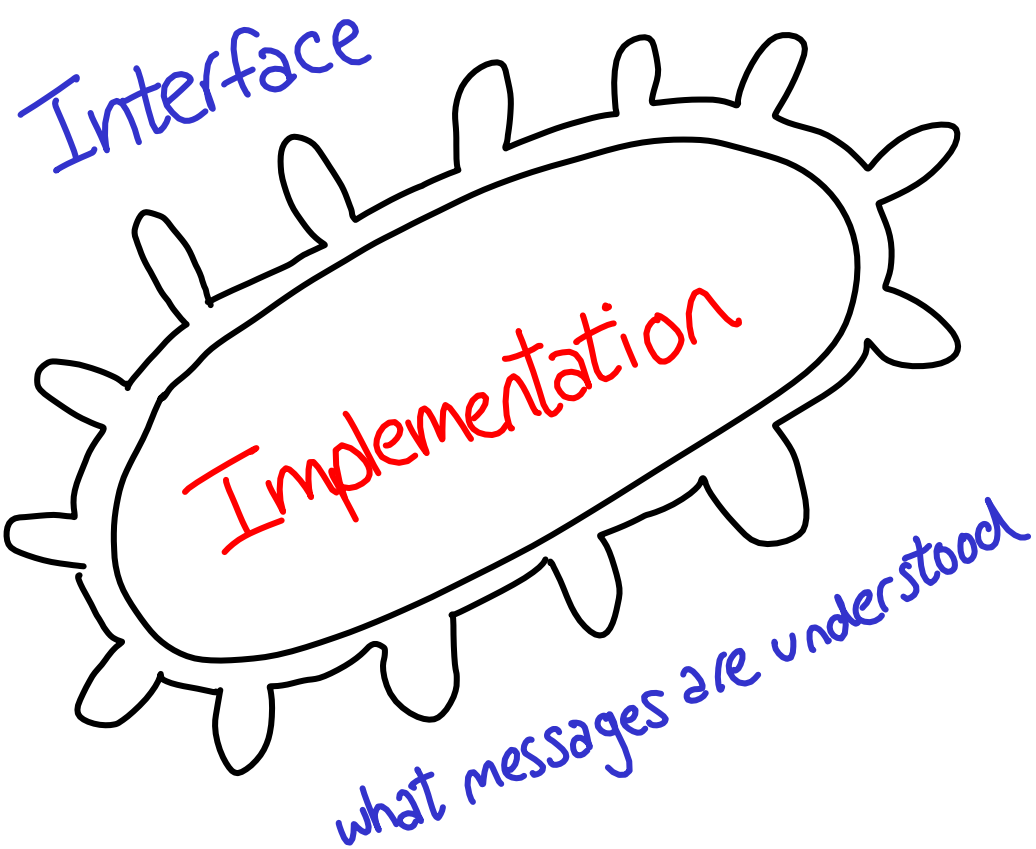
compare: n1→add(n2) versus add(n1,n2)

(first-class functions = dynamic dispatch)

Encapsulation



Encapsulation



Subtyping

relation between interfaces

ColoredPoint
- x-coord
- y-coord
- color
- move
- change-color

$<:$

subtype of

Point
- x-coord
- y-coord
- move

if interface A contains all of interface B,
then A objects can be used as B objects

Inheritance

relation between implementation

```
class ColoredPoint extends Point {
    // reuse implementation of move
    void changeColor(...) { ... }
    ...
}
```

Subtyping $\neq$ Inheritance

interfaces

implementation

OO principle:

Group data and code together

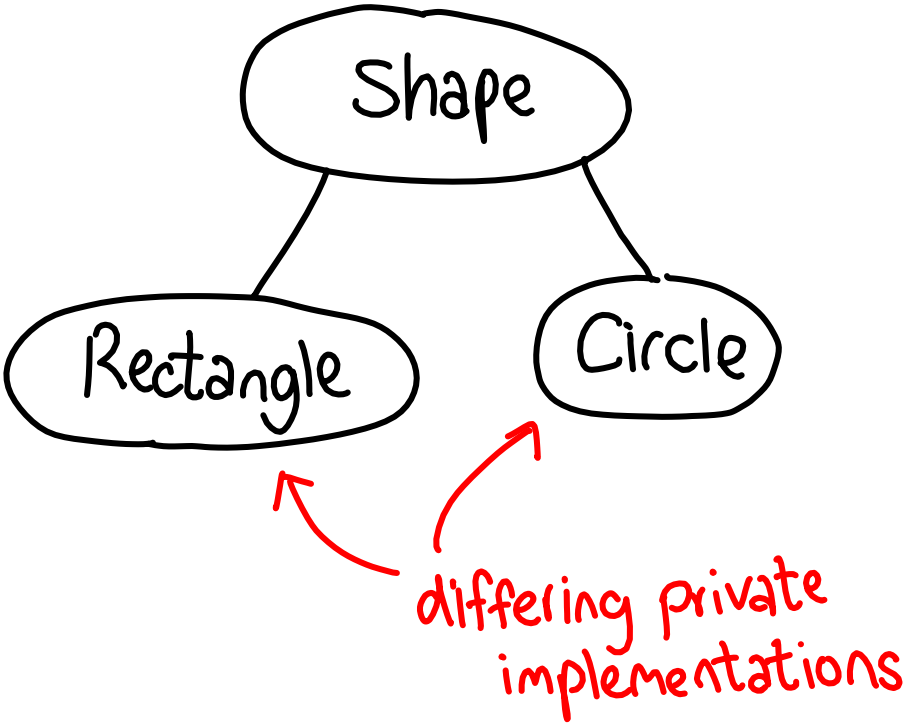
Comparative example: Shapes

Haskell

```
data Shape = Square Point Point
           | Circle Point Length

center :: Shape -> Point
move :: Shape -> Point -> Shape
render :: Shape -> IO ()
```

Comparative example: Shapes

Objects



Shape
- center
- move
- render

How can we understand OO with the tools of this class?

by reducing objects to known concepts (Simula)

by simplifying objects to a core idea (Smalltalk, Self)

Simula

objects as activation records

Algol 60

+classes & subtyping

Simula 67

a language for simulation

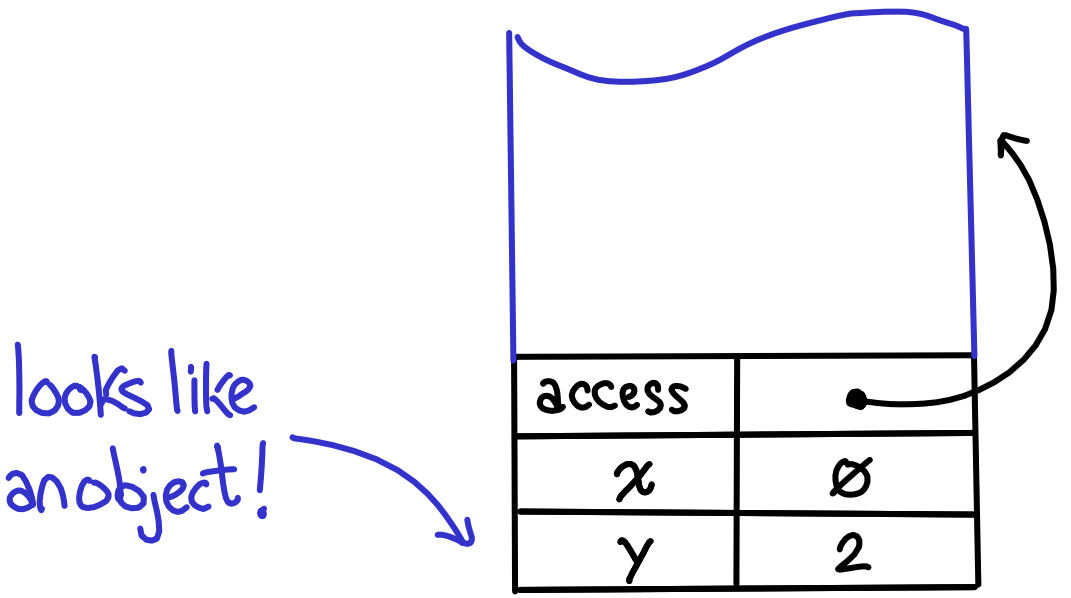
Nygaard & Dahl

Smalltalk

C++

...

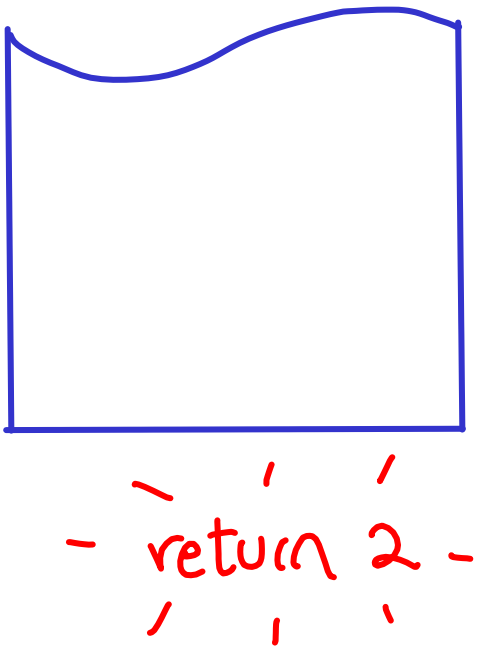
Recall: Activation Records



(Note: access link points to the "textually enclosing block instance")

Recall: Activation Records

Similar situation in manually memory managed languages (C, C++)

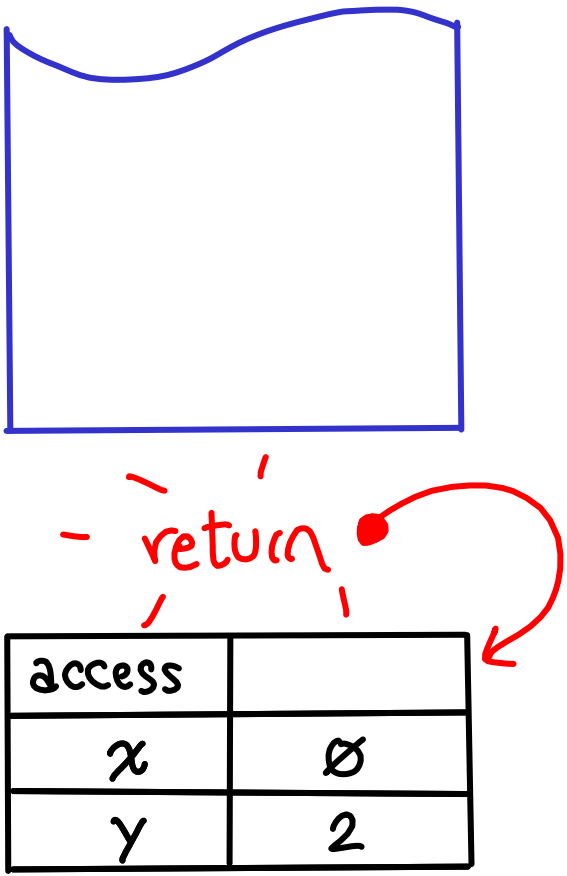


Recall: Activation Records

Simula called these "blocks"

http://web.cecs.pdx.edu/~black/publications/O-JDahl.pdf

"Objects already existed: they just needed to be freed from the stack discipline."



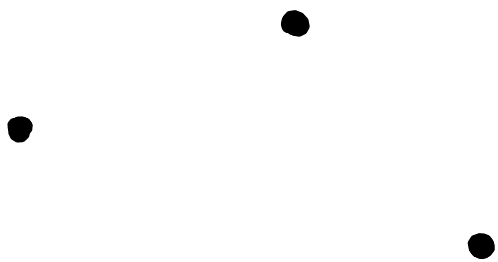
Need a GC!

Objects in Simula

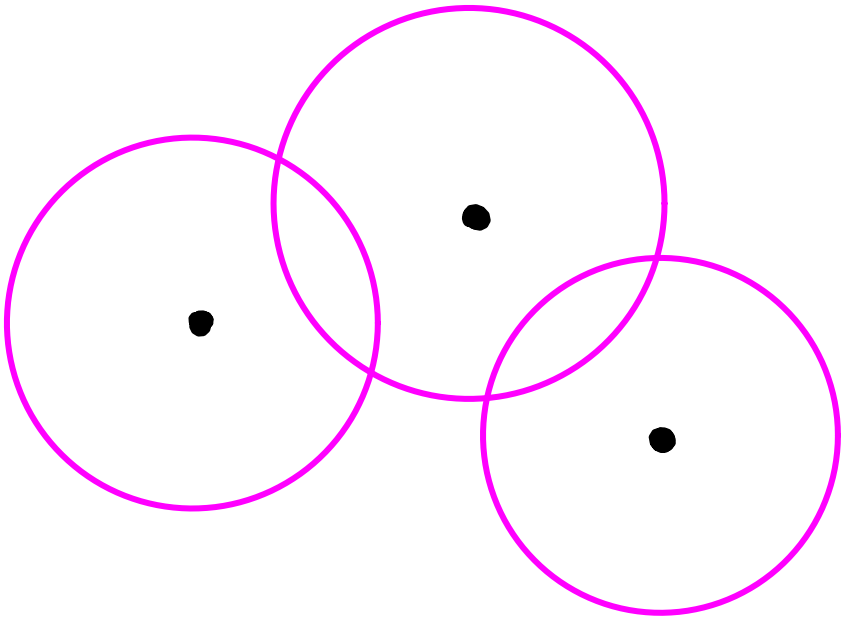
statically typed!

Class	Function that returns pointer to its activation record
Object	Activation record produced by call to class
Object access	Dot operator to access variables in record

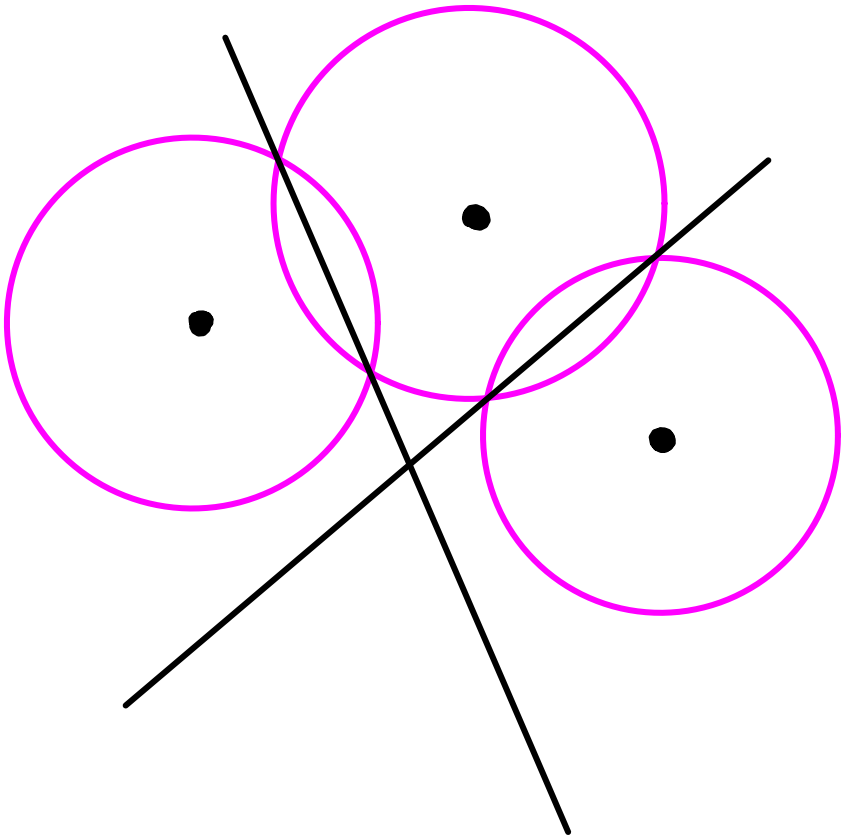
Example: Find radius & center of circle passing through three distinct points



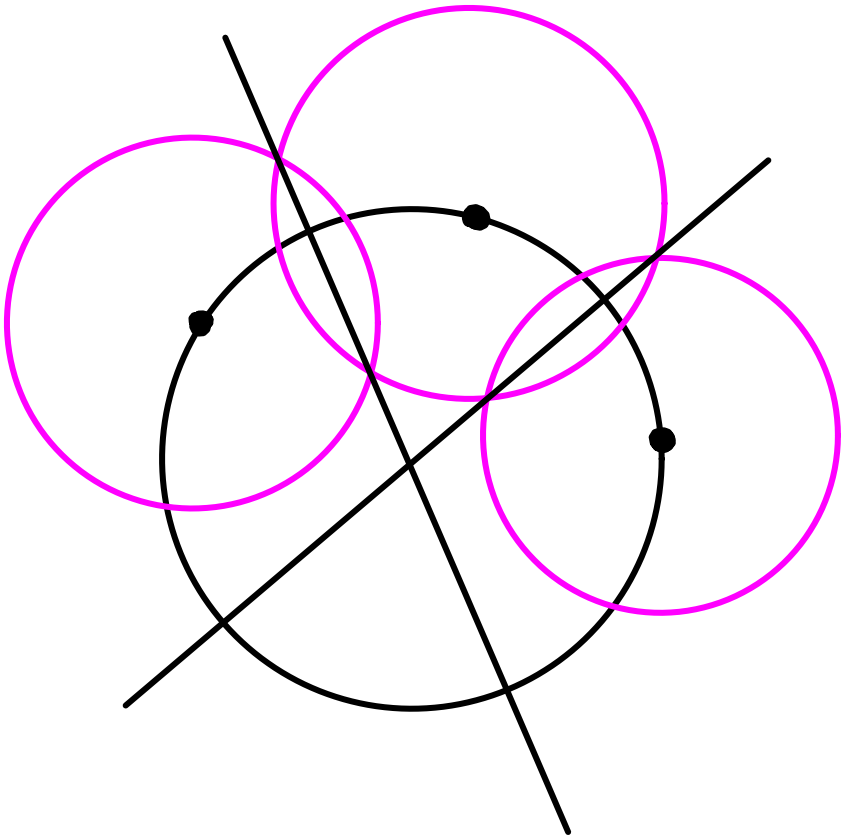
Example: Find radius & center of circle passing through three distinct points



Example: Find radius & center of circle passing through three distinct points



Example: Find radius & center of circle passing through three distinct points



Plan: Points, lines, circles ⇝ objects

Point
- equals(aPoint) : boolean
- distance(aPoint) : real

Line
- parallelto(aLine) : boolean
- meets(aLine) : REF(Point)

Circle
- intersects(aCircle) : REF(Line)

Plan: Points, lines, circles ⇝ objects

Point
- equals(aPoint) : boolean
- distance(aPoint) : real

Line
- parallelto(aLine) : boolean
- meets(aLine) : REF(Point)

Circle
- intersects(aCircle) : REF(Line)

references to object

```
class Point(x,y); real x,y;
begin
  boolean procedure equals(p); ref(Point) p;
    if p =/= none then
      equals := abs(x - p.x) + abs(y - p.y) < 0.00001
  real procedure distance(p); ref(Point) p;
    if p == none then error else
      distance := sqrt(( x - p.x )**2 + (y - p.y) ** 2);
end ***Point***

p :- new Point(1.0, 2.5);
q :- new Point(2.0, 3.5);
if p.distance(q) > 2 then ...
```

types of parameters
(a "fictitious" block to
store the arguments)

```
class Point(x,y); real x,y;
begin
  boolean procedure equals(p); ref(Point) p;
    if p =/= none then
      equals := abs(x - p.x) + abs(y - p.y) < 0.00001
  real procedure distance(p); ref(Point) p;
    if p == none then error else
      distance := sqrt(( x - p.x )**2 + (y - p.y) ** 2);
end ***Point***

p :- new Point(1.0, 2.5);
q :- new Point(2.0, 3.5);
if p.distance(q) > 2 then ...
```

```
class Point(x,y); real x,y;
begin
  boolean procedure equals(p); ref(Point) p;
    if p =/= none then
      equals := abs(x - p.x) + abs(y - p.y) < 0.00001
  real procedure distance(p); ref(Point) p;
    if p == none then error else
      distance := sqrt( x - p.x )**2 + (y - p.y) ** 2);
end ***Point***

p :- new Point(1.0, 2.5);
q :- new Point(2.0, 3.5);
if p.distance(q) > 2 then ...
```

return the value by assigning it

(some still-alive langs still have this syntax; e.g. Matlab, Fortran)

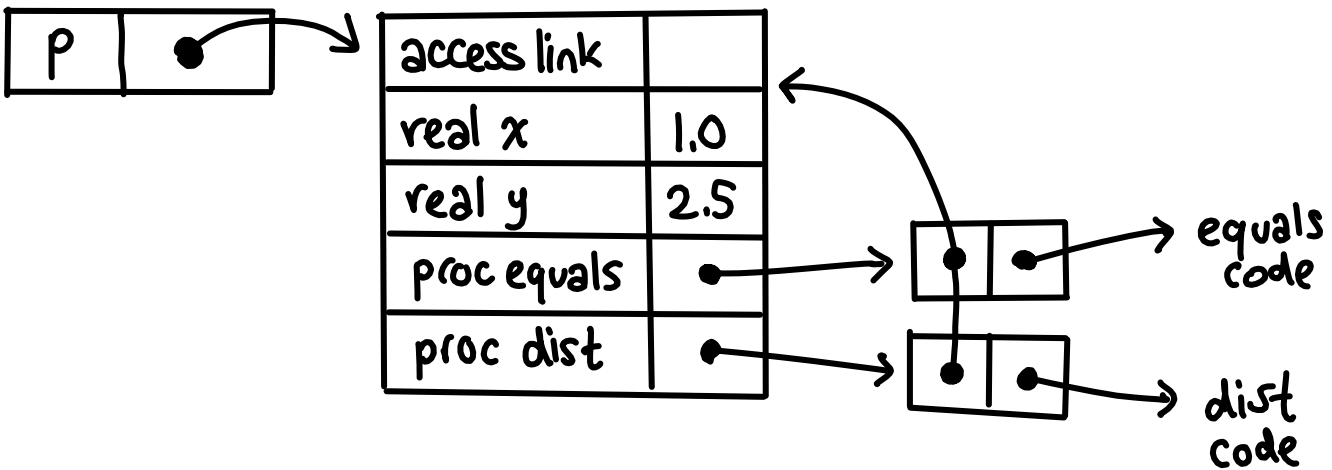
Special operator for reference assignment

new lexical scope

```
class Point(x,y); real x,y;
begin
  boolean procedure equals(p); ref(Point) p;
    if p =/= none then
      equals := abs(x - p.x) + abs(y - p.y) < 0.00001
  real procedure distance(p); ref(Point) p;
    if p == none then error else
      distance := sqrt(( x - p.x )**2 + (y - p.y) ** 2);
end ***Point***
```

```
p :- new Point(1.0, 2.5);
q :- new Point(2.0, 3.5);
if p.distance(q) > 2 then ...
```

Object representation



NB: Simula did not support closures; environment pointer for procedures defined in class "hard-coded" in spec. Like Algol 60, functions could be passed but not returned.

```
class Line(a,b,c); real a,b,c;
begin
  boolean procedure parallelto(l); ref(Line) l;
    if l =/= none then  parallelto := ...
  ref(Point) procedure meets(l); ref(Line) l;
    begin real t;
      if l =/= none and ~parallelto(l) then ...
    end;
  real d;
  d := sqrt(a**2 + b**2);
  if d = 0.0 then error else
    begin
      d := 1/d;
      a := a*d;  b := b*d;  c := c*d;
    end;
end *** Line***
```

local variables

```
class Line(a,b,c); real a,b,c;
begin
  boolean procedure parallelto(l); ref(Line) l;
    if l =/= none then  parallelto := ...
  ref(Point) procedure meets(l); ref(Line) l;
    begin real t;
      if l =/= none and ~parallelto(l) then ...
    end;
  real d;
  d := sqrt(a**2 + b**2);
  if d = 0.0 then error else
    begin
      d := 1/d;
      a := a*d;  b := b*d;  c := c*d;
    end;
end *** Line***
```

procedures

local variables

```
class Line(a,b,c); real a,b,c;
begin
  boolean procedure parallelto(l); ref(Line) l;
    if l =/= none then  parallelto := ...
  ref(Point) procedure meets(l); ref(Line) l;
    begin real t;
      if l =/= none and ~parallelto(l) then ...
    end;
  real d;
  d := sqrt(a**2 + b**2);
  if d = 0.0 then error else
    begin
      d := 1/d;
      a := a*d;  b := b*d;  c := c*d;
    end;
end *** Line***
```

procedures

body of class is just a function!

Derived classes (inheritance)

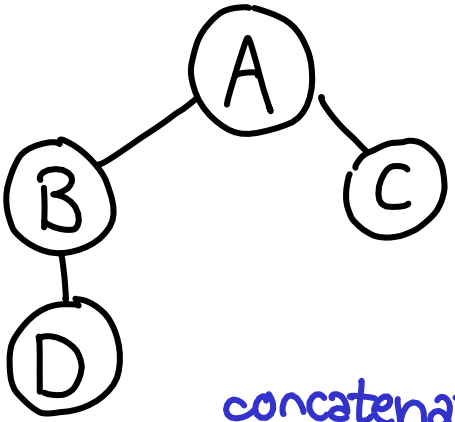
Hoare

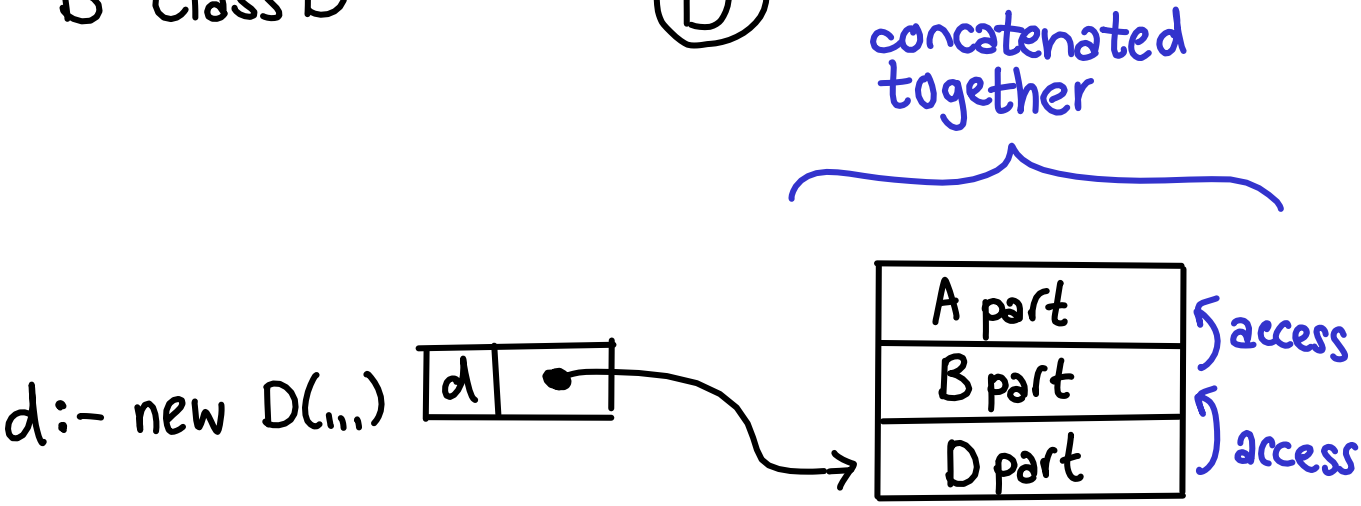
class A

A class B

A class C

B class D





Virtual versus activation records

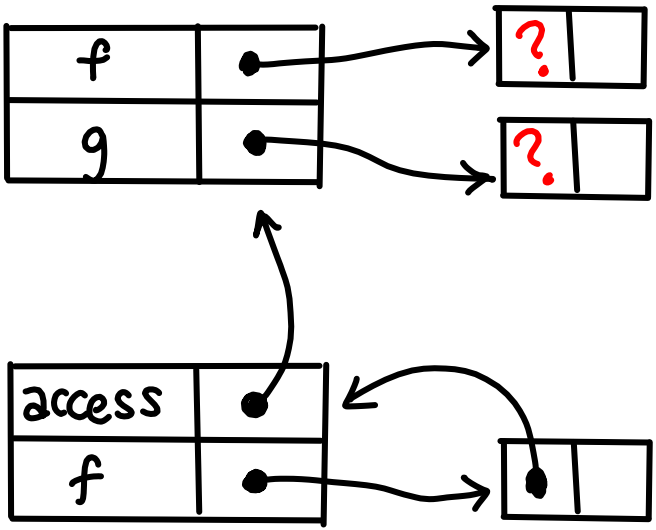
```
class A;
begin
  integer procedure f();
    begin f := 0 end;
  integer procedure g();
    begin g := f end;
end;
```

nullary function call

```
A class B;
begin
  integer procedure f();
    begin f := 1 end;
end;

b :- new B();
b.g % What is this value?
```

Virtual versus activation records

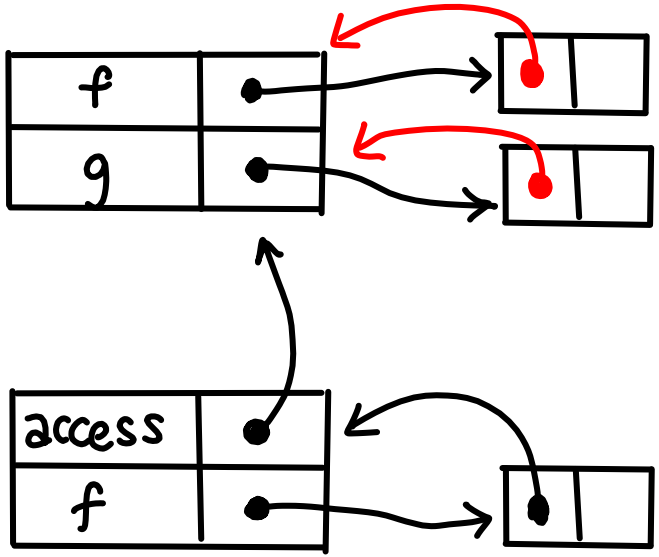
```
class A;
begin
  integer procedure f();
    begin f := 0 end;
  integer procedure g();
    begin g := f end;
end;

A class B;
begin
  integer procedure f();
    begin f := 1 end;
end;

b :- new B();
b.g % What is this value?
```



Virtual versus activation records

```
class A;
begin
  integer procedure f();
    begin f := 0 end;
  integer procedure g();
    begin g := f end;
end;

A class B;
begin
  integer procedure f();
    begin f := 1 end;
end;

b :- new B();
b.g % What is this value?
```

It's ∅!



Extra virtual keyword to get to the object itself.

Simula 67: Summary

Hugely influential language, with

- Classes
- Objects
- Inheritance
- Subtyping
- Virtual methods
- Inner (combine parent code)
- Inspect/Qua (instanceof/cast)

Features for simulation

Simula 67: Summary

Some missing things...

Encapsulation (added later)

Self/Super (Smalltalk)

(did have this(class))

Class variables (use globals instead)

Exceptions (whatever)

Simula 67: Summary

⇝ C++

Class	Function that returns pointer to its activation record
Object	Activation record produced by call to class
Subtyping	By class hierarchy
Inheritance	By prefixing

a shift... to dynamically typed
OO languages

Smalltalk

Everything is an object... even classes

Smalltalk

- Popularized objects
- Developed at Xerox PARC

"Take the hardest and most profound thing you need to do, make it great, and then build every easier thing out of it". That was the promise of LISP and the lure of lambda—[what it] needed was a better "hardest and most profound" thing. Objects should be it.

- Influenced by Simula, but very different
- All operations messages to objects

"The most powerful language in the world...in a page of code."

More history: http://worrydream.com/EarlyHistoryOfSmalltalk/

FIGURE 11.21 The Dynabook model

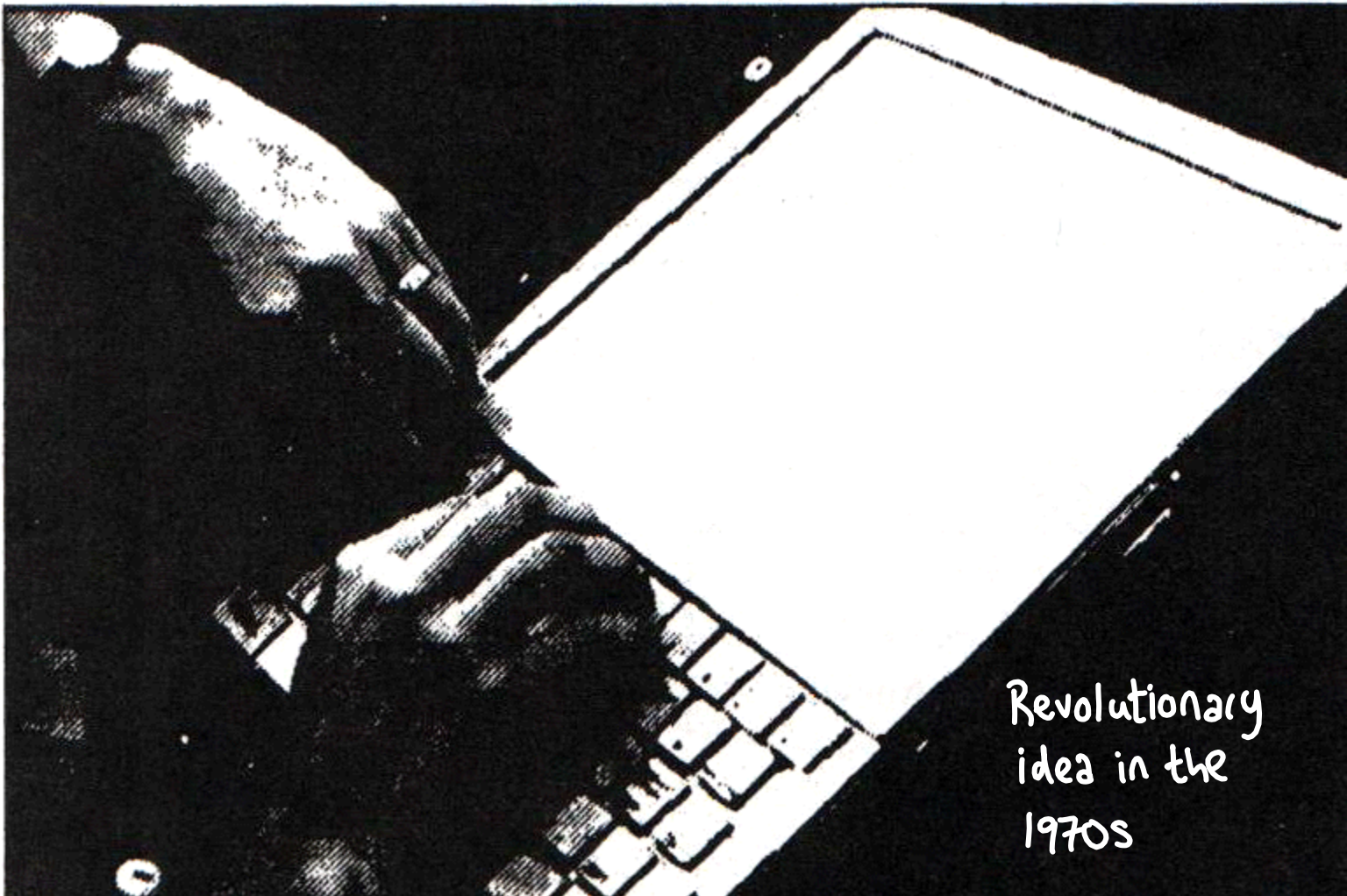
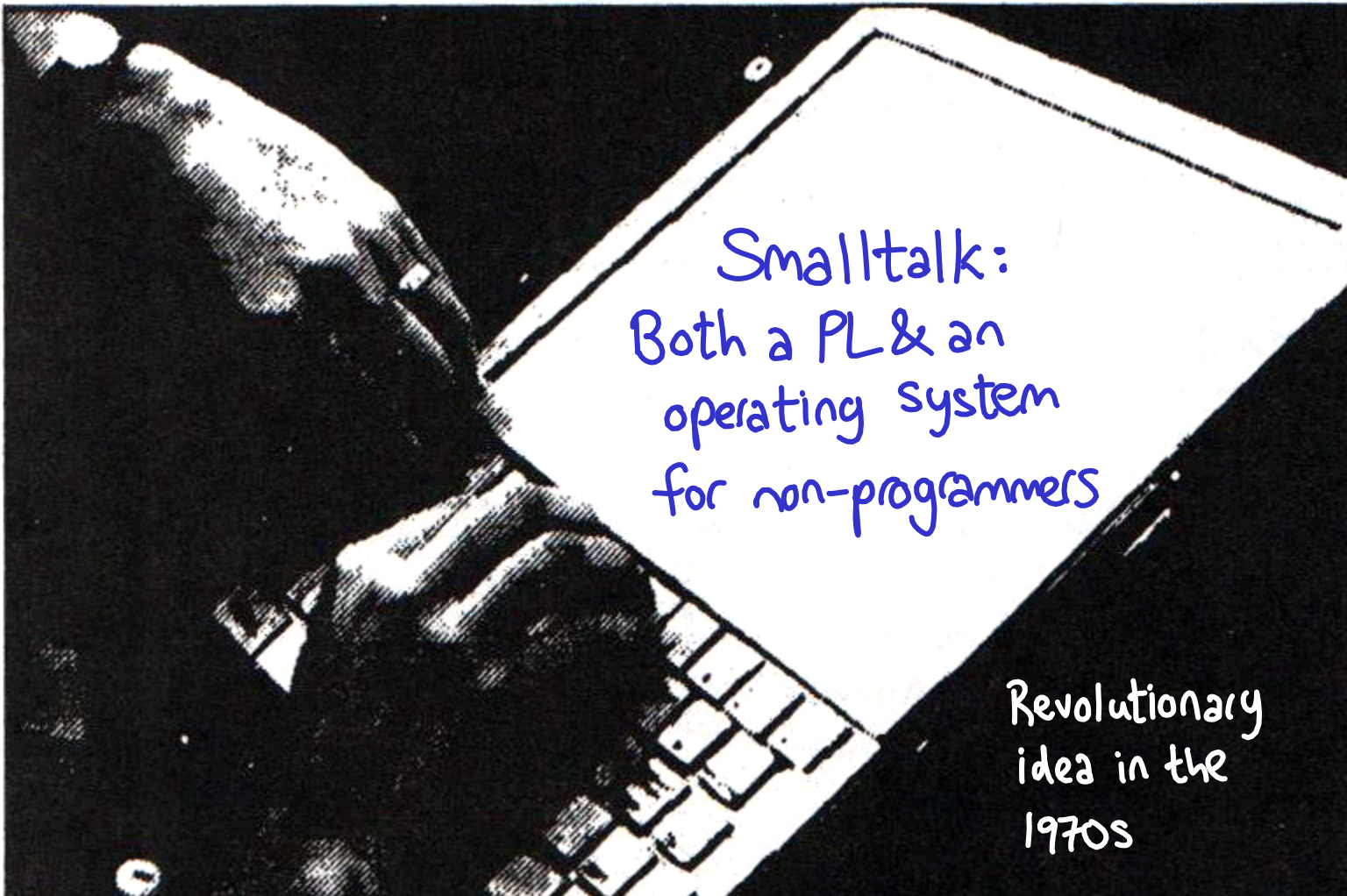


FIGURE 11.21 The Dynabook model



Smalltalk terminology

Object	Instance of a class
Class	Def$^{\underline{n}}$ of the behavior of objects
Selector	Name of message (method name)
Message	Selector + parameter values (could be forwarded)
Method	Code to respond to message
Instance variable	Data stored in object
Subclass	Incrementally modified parent class

Smalltalk semantics

- Everything is an object
- Objects communicate by sending/receiving messages.
- Objects have their own state
- Every object is an instance of a class
- A class provides behavior for its instances

Instance messages and methods

```
x: xcoord y: ycoord | |
   x <- xcoord
   y <- ycoord
moveDx: dx Dy: dy | |
   x <- dx + x
   y <- dy + y
x | | ^x
y | | ^y
draw | |
   ...code to draw point...
```

Selectors

x:y: is a mixfix operator

pt x:2 y:3

message

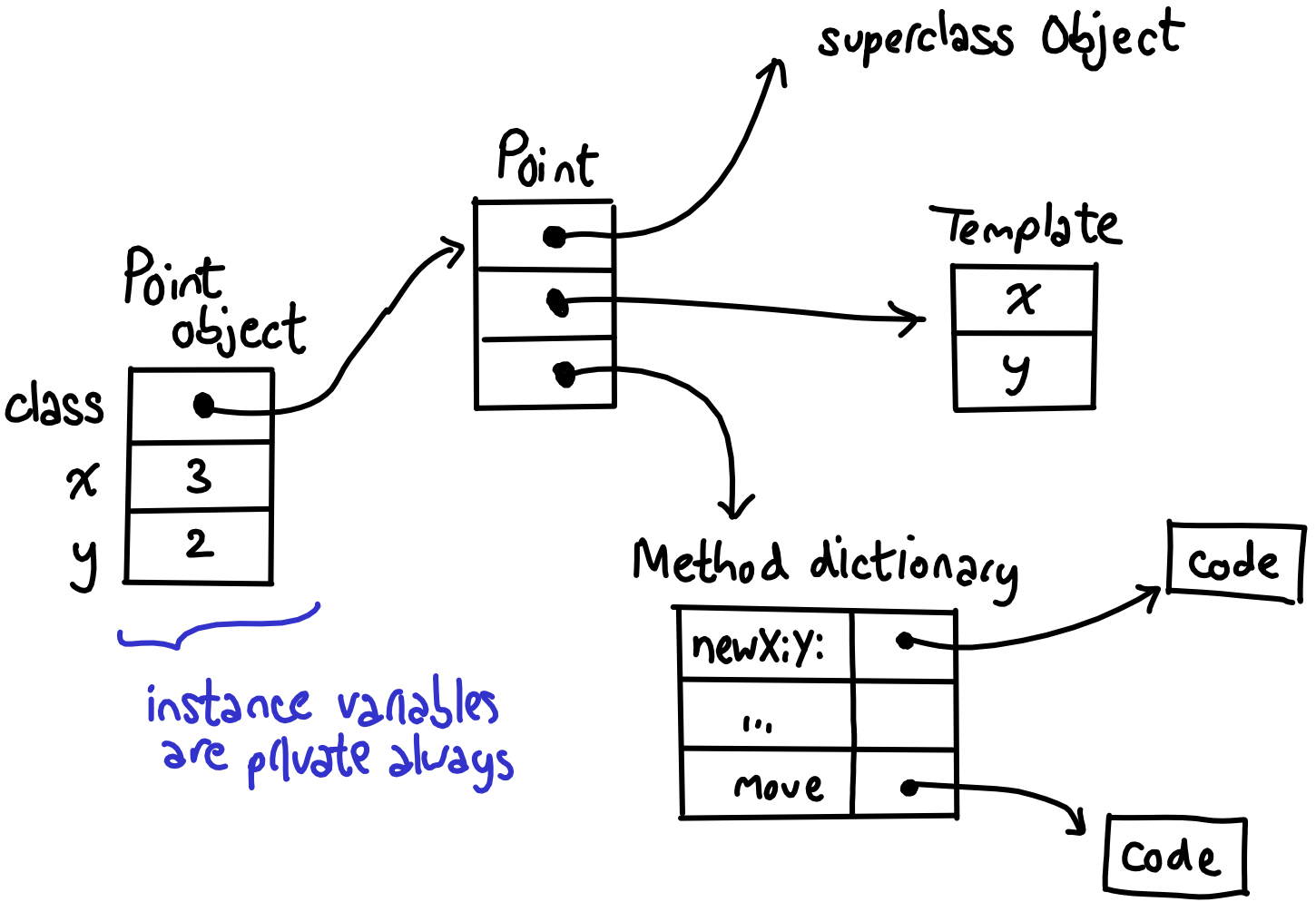
Instance messages and methods

```
x: xcoord y: ycoord | |
   x <- xcoord
   y <- ycoord
moveDx: dx Dy: dy | |
   x <- dx + x
   y <- dy + y
x | | ^x
y | | ^y
draw | |
   ...code to draw point...
```

Mutable assignment

Instance variables (private)

Return


superclass Object
Point
Point object
class
x
3
y
2
Template
x
y
Method dictionary
newX:y:
...
move
Code
Code
instance variables are private always

Point

class name	Point
super class	Object
class var	pi
instance var	x y
class messages and methods	
<...names and code for methods...>	
instance messages and methods	
<...names and code for methods...>	

Class messages and methods

```
newX:xvalue Y:yvalue  | |
^ self new x: xvalue
           y: yvalue

newOrigin | |
^ self new x: 0
           y: 0

initialize | |
    pi <- 3.14159
```

Classes are objects too!

Send the "new" message to self object (Point class is object)

```
newX:xvalue Y:yvalue  | |
^ self new x: xvalue
           y: yvalue

newOrigin | |
^ self new x: 0
           y: 0

initialize | |
    pi <- 3.14159
```

To newly created Point object, send (mixfix) message x:y:

self could be overloaded: always points to actual object

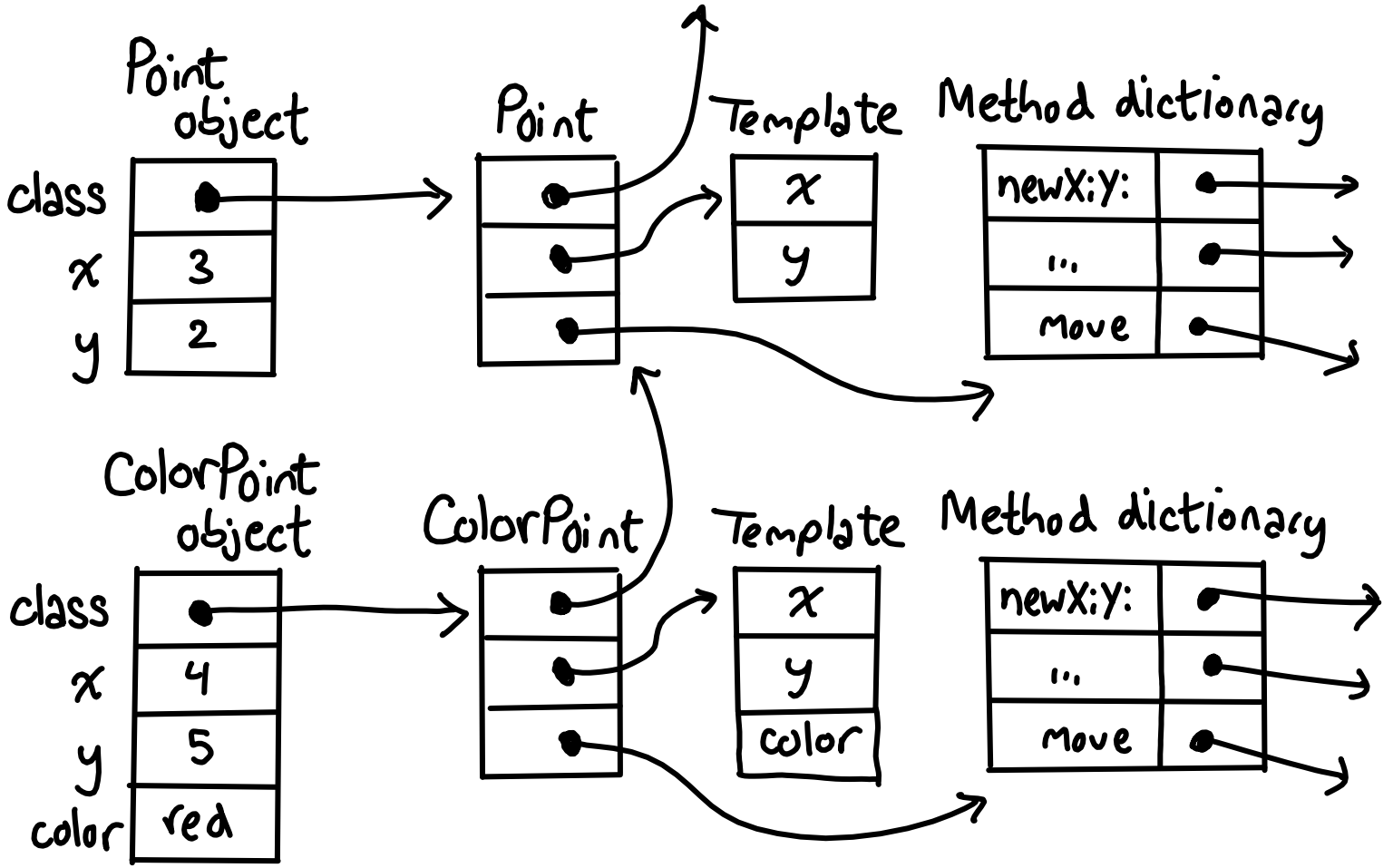
Inheritance

class name	ColorPoint
super class	Point
class var	
instance var	color
class messages and methods	
newX:Y:C:	<... code...>
instance messages and methods	
color	|| ^ color
draw	<.. code ..>

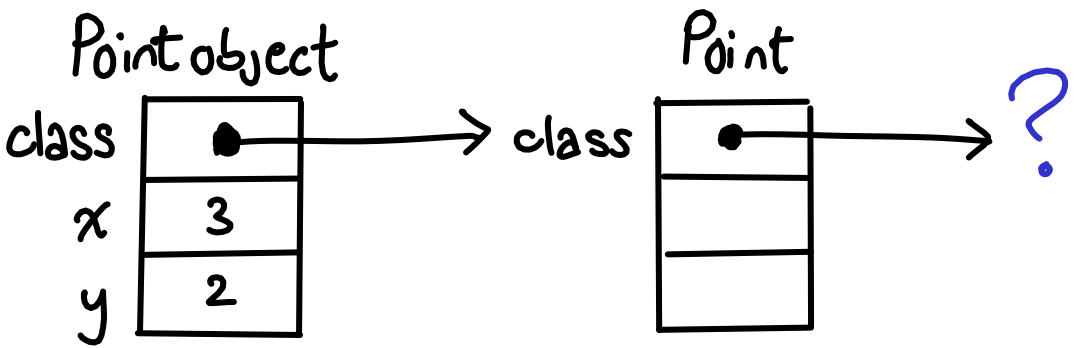
new instance variable

new method

override

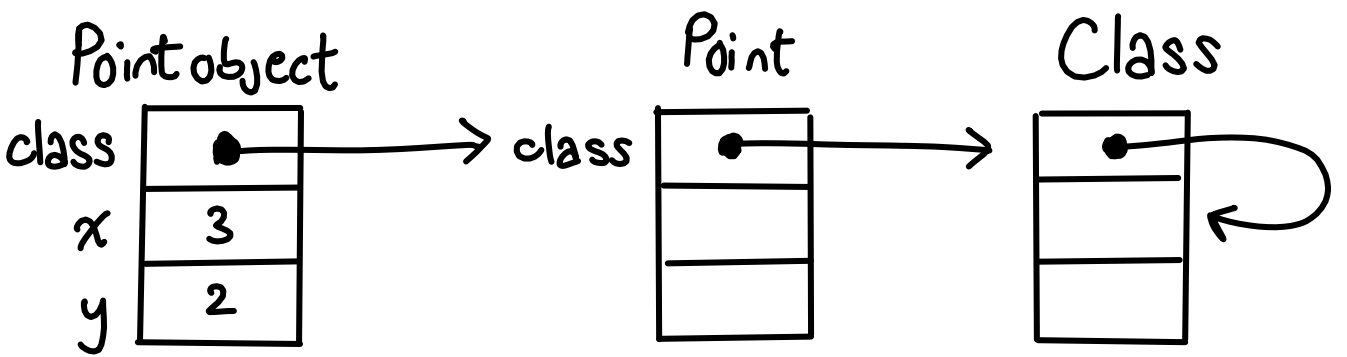

Point object
class
x
3
y
2
Point
Template
x
y
Method dictionary
newX:y:
...
move
ColorPoint object
class
x
4
y
5
color
red
ColorPoint
Template
x
y
color
Method dictionary
newX:y:
...
move

If a class is an object...



What's the class of a class?

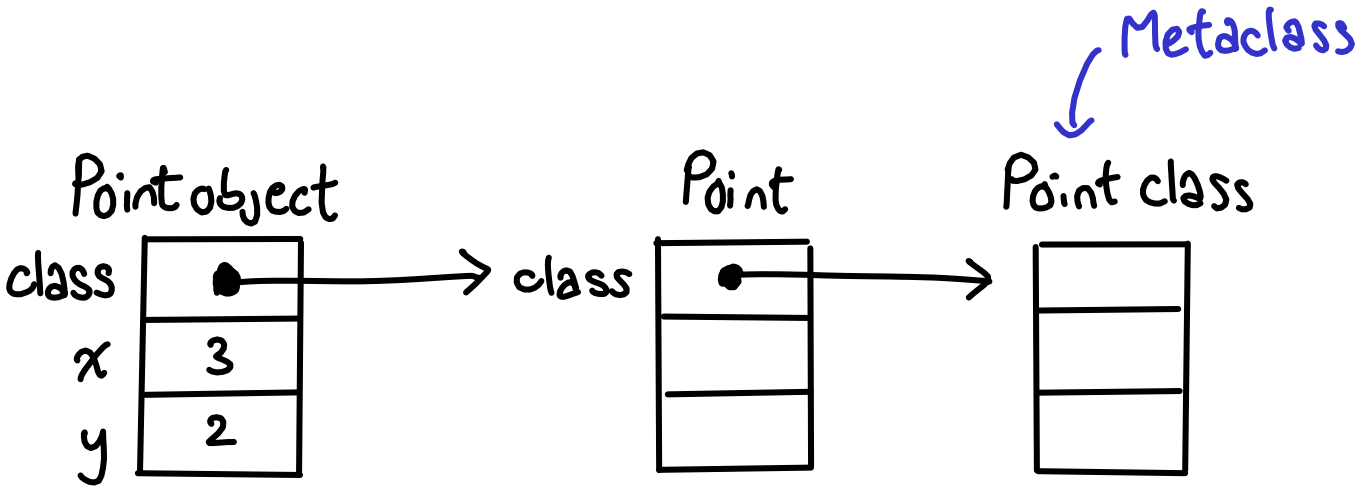
What's the class of a class?



Smalltalk-76: "Class"

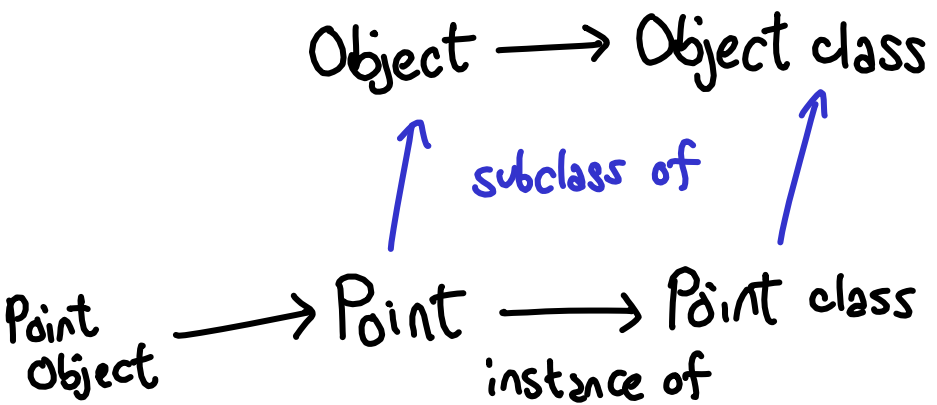
Trouble: all class methods (e.g. method to handle message to Point class must be put in Class)

What's the class of a class?



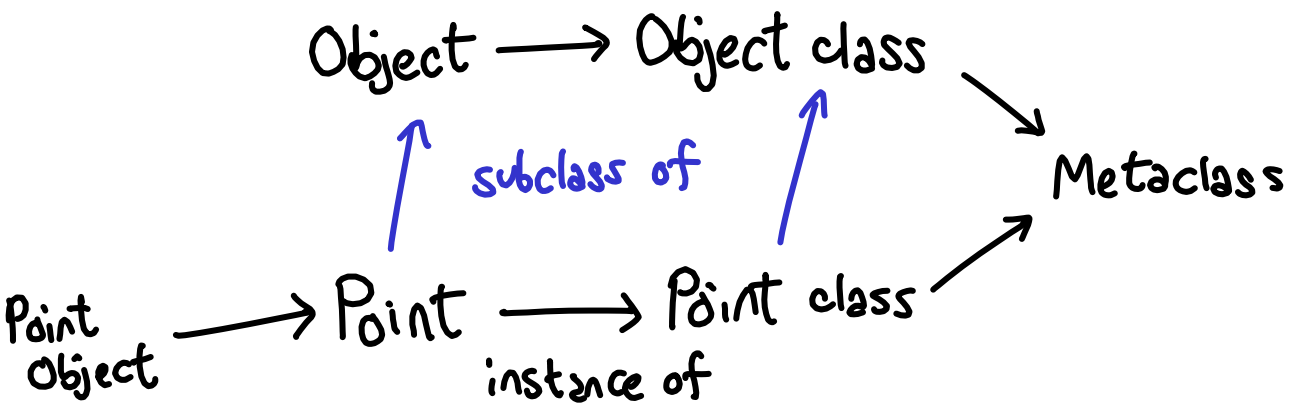
Smalltalk-80: Metaclasses

What's the class of a class?



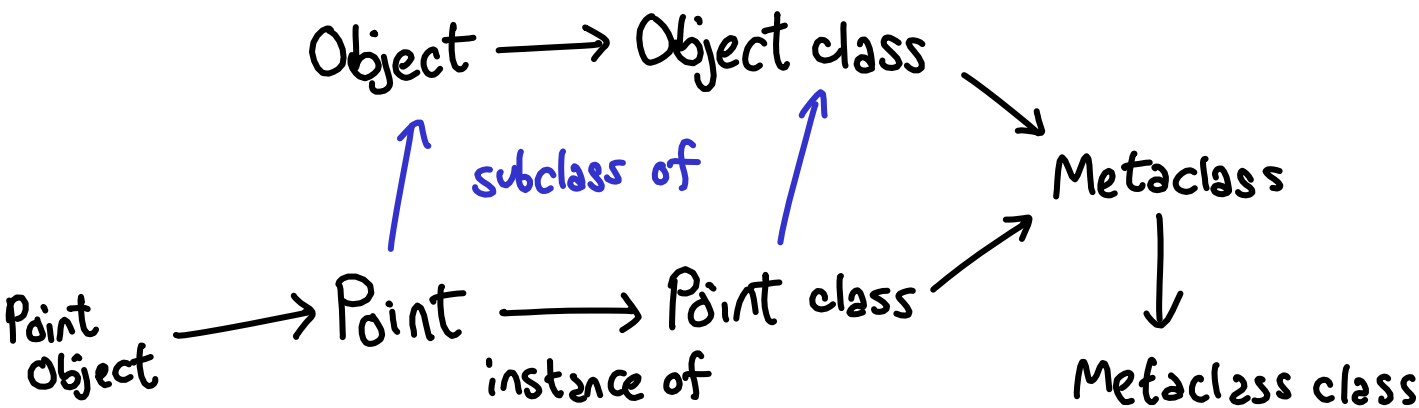
Smalltalk-80: Metaclasses

What's the class of a class?



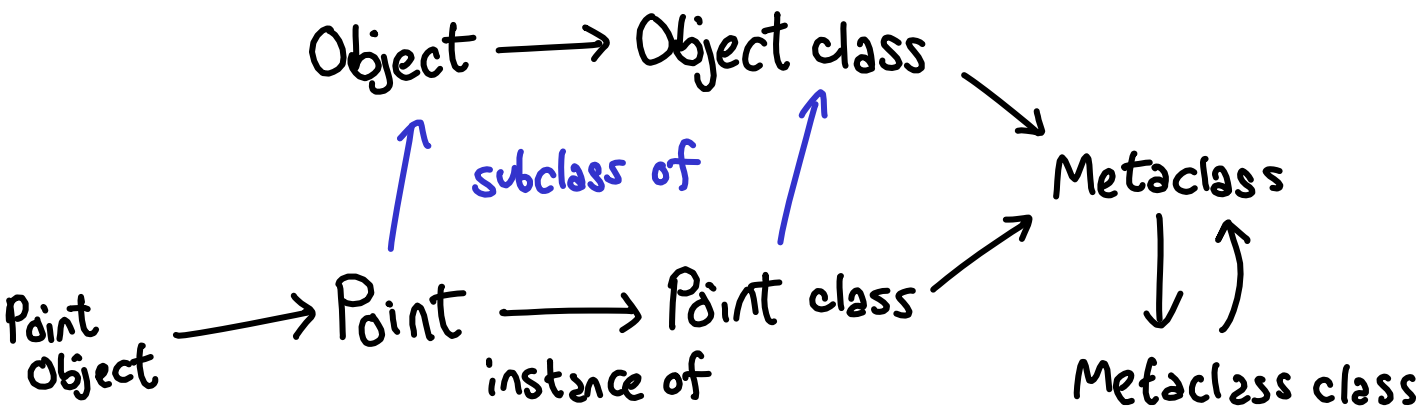
Smalltalk-80: Metaclasses

What's the class of a class?



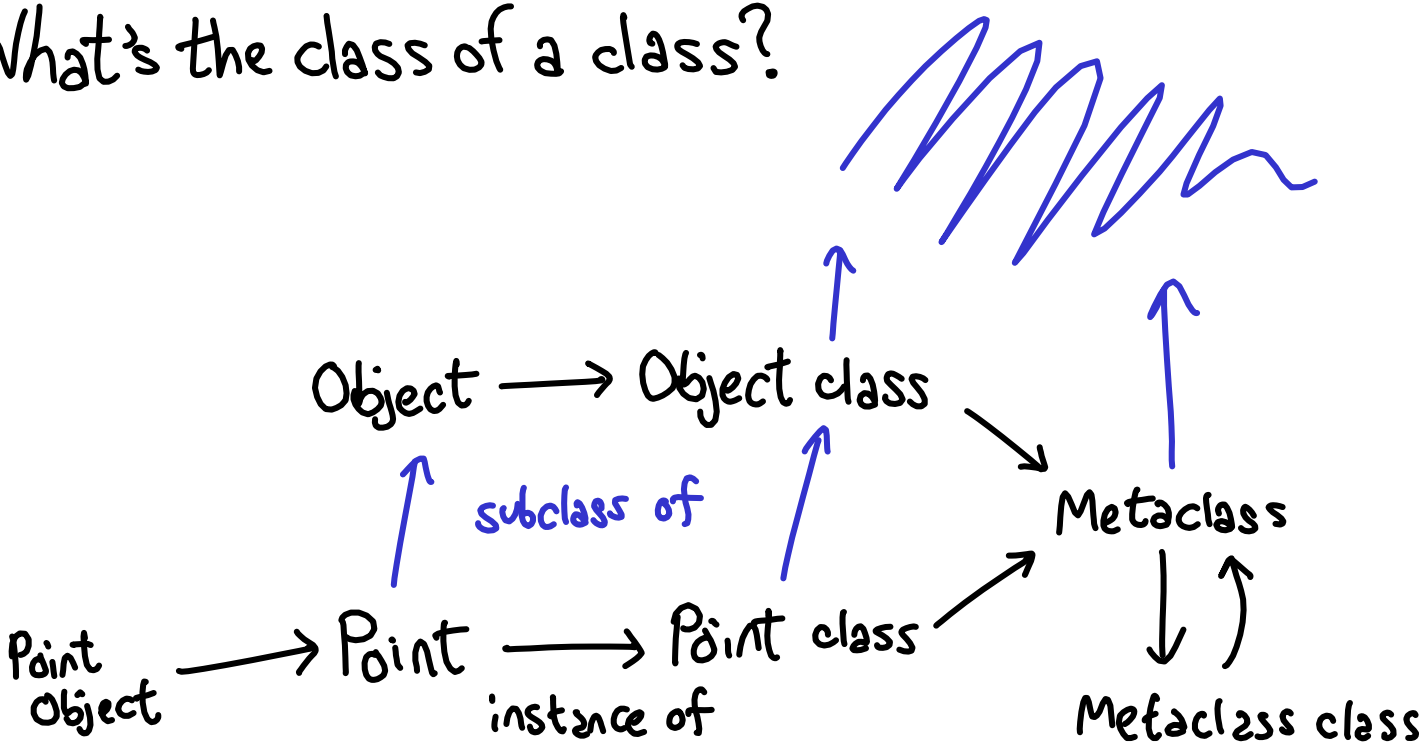
Smalltalk-80: Metaclasses

What's the class of a class?



Smalltalk-80: Metaclasses

What's the class of a class?



Smalltalk-80: Metaclasses

Smalltalk summary

Metaprogramming on crack

Class
- create objects that share methods
- internally records dictionary, parent, ...

Objects
- created by class, has private state

Encapsulation
- public methods, private state

Subtyping: implicit

Inheritance: subclasses, self, super

Self

Everything is an object; NO classes

Self

- Prototype-based OO language
- Randal Smith (Xerox PARC) and David Ungar (Stanford)
  - Successor to Smalltalk'80
  - "Self: The power of simplicity" OOPSLA'87
- Influence
  - JavaScript
  - Advances in compilation (esp. Java)

Self

- Everything is an object
- Everything done by messages
- No classes
- No variables

"A language for Smalltalk runtime structures"

Self semantics

- Clone
- Send message
- Add new slot
- Replace old slots
- Remove slots

Self semantics

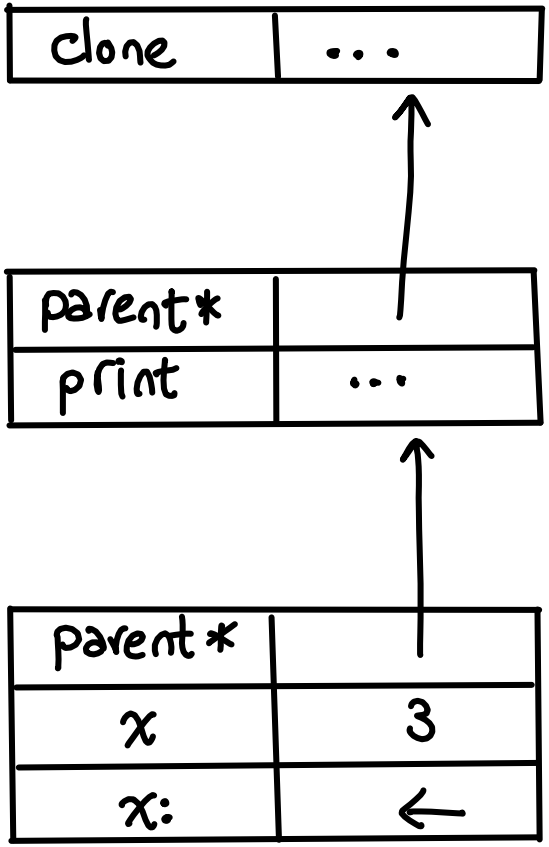
Objects consist of named slots

Slots contain code to do various things...

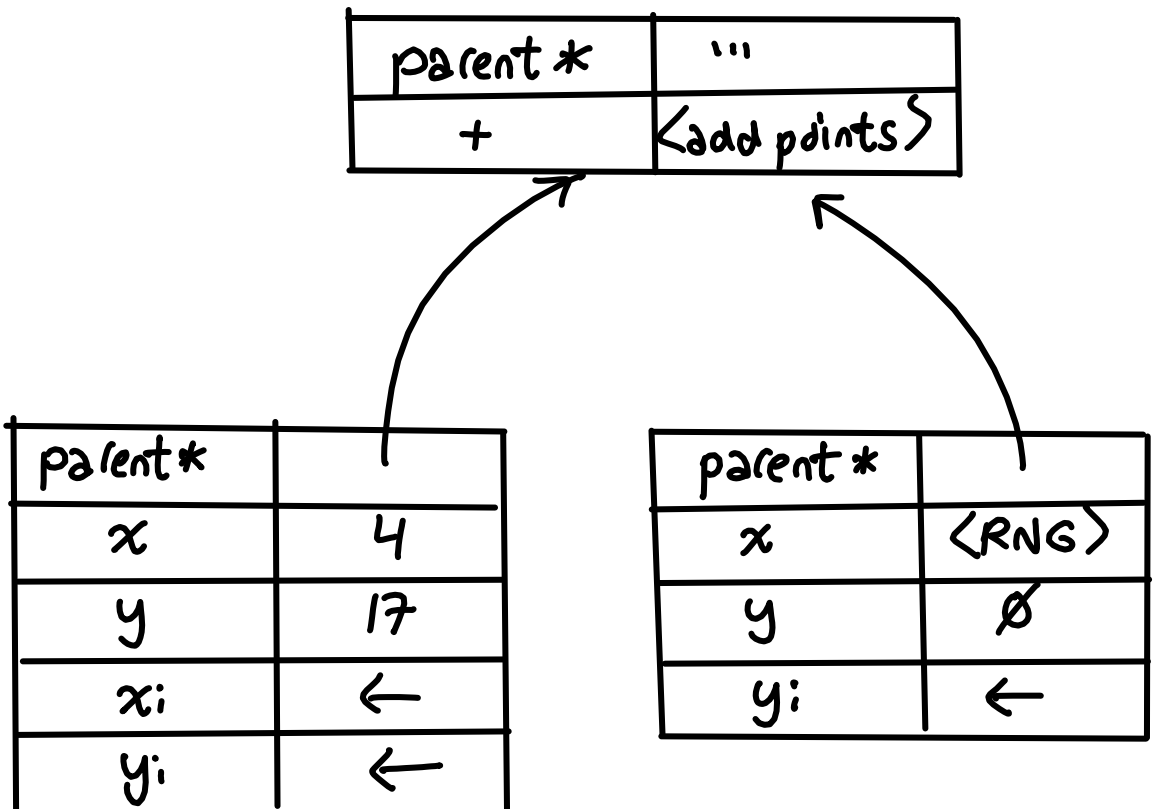
- Data — Return contents upon eval
- Assignment — Set value of slot
- Method — Code to run
- Parent — Inhelit slots from other object

Messages

- When message is sent, search for slot w/ name
- If not found, recursively search via *parent pointer
- When found, evaluate code in slot and return result
- <u>self</u> points to message receiver



"Variables" are not dumb!



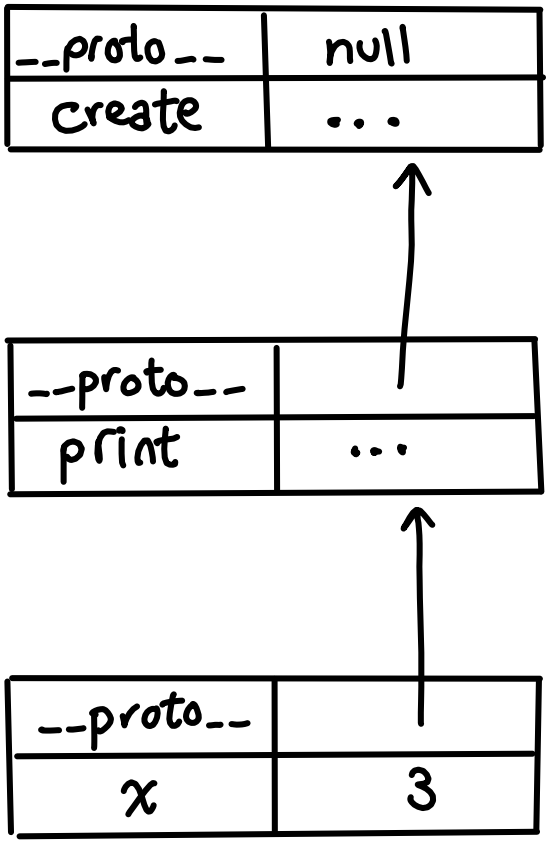
Object creation

- To create, copy old one (prototype)
- Add/modify/remove methods
- You can even change parent pointer

JavaScript prototypes

To create:

- new F();
  - __proto__ = F.prototype
- Object.create(p)
  - __proto__ = p



Not everything is a message

this keyword only defined when "received message"

Self versus classes

✓ Simpler! Avoids meta-classes

✗ Less structure; programmer discipline

(ES6 adding classes, which desugar to prototypal inheritance)

What are the central ideas of OO?

dynamic dispatch • encapsulation • subtyping • inheritance

How can we understand OO with the tools of this class?

by reducing objects to known concepts (Simula)

by simplifying objects to a core idea (Smalltalk, Self)