

# Pharo: the best system for object-oriented \*thinking\*

<http://stephane.ducasse.free.fr>

<http://www.pharo.org>



**Reflecting on our daily  
practice and experience**

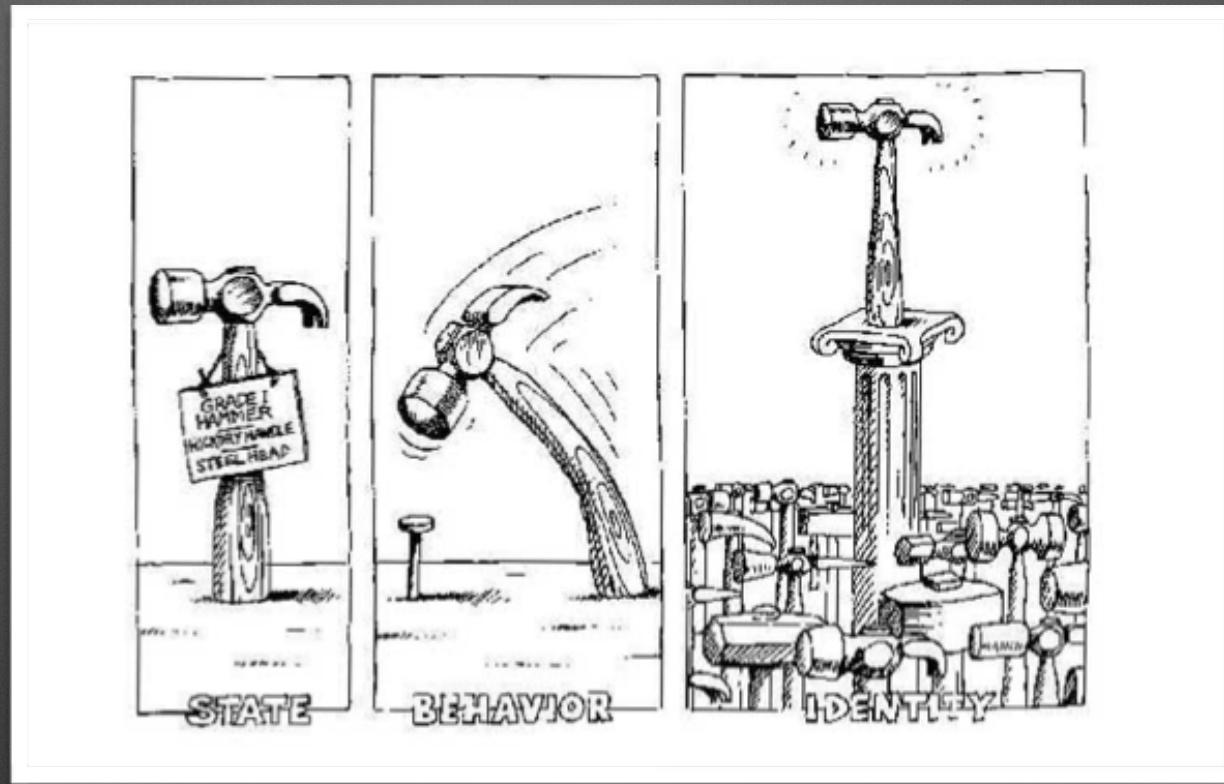
# Properties we see

- Turtles all the way down
- Uniform and elegant
- Simple oo model
- But powerful to build really large applications
- Open system
- Learn from the system itself

**What is the essence of OOP?**

# objects + late binding

**objects!**



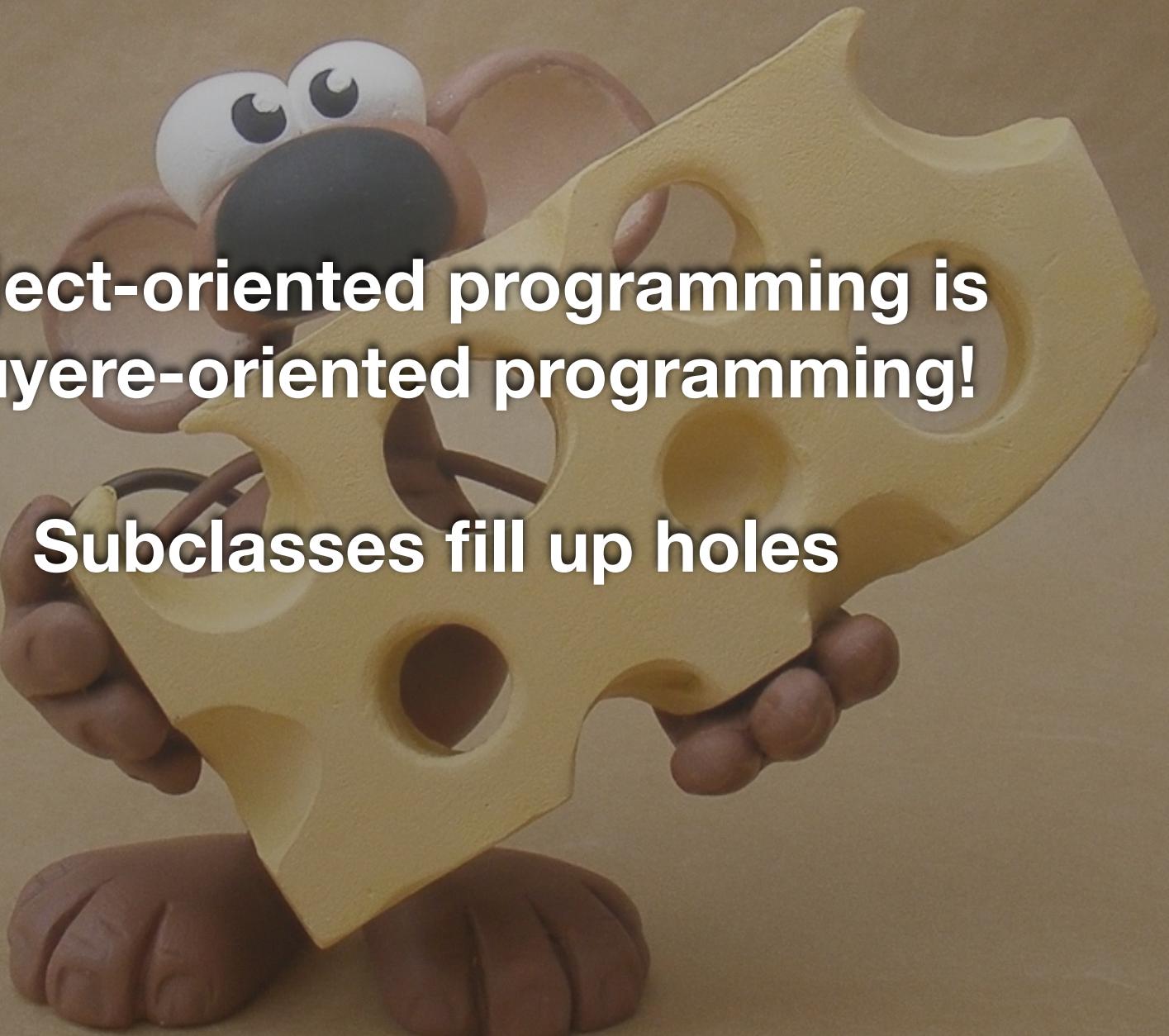
late binding

# Late binding

The fact that the “procedure” to be executed depends on the object on which the computation is requested.

**Sending a message is a choice operator**

**Classes embed choices**

A cartoon character with a large head, white hair, and a black face is holding a large wedge of yellow cheese with several holes. The character's hands are brown and are gripping the cheese from below. The background is a plain, light brown.

**Object-oriented programming is  
gruyere-oriented programming!**

**Subclasses fill up holes**

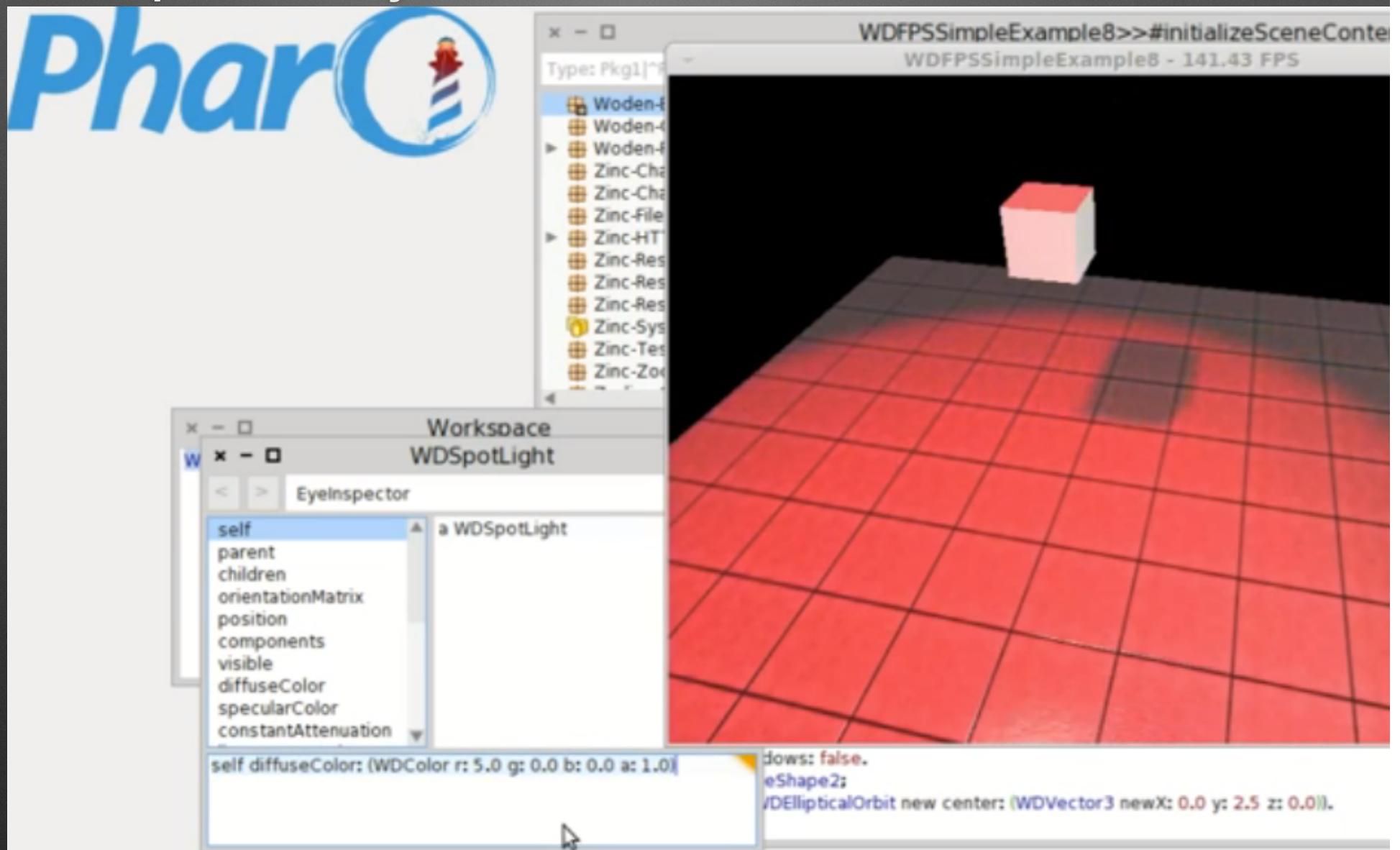
**How to convey that objects  
are living/interactive entities?**

# How to convey the object feel?

- Live interaction
- Using Inspectors
  - Talk and interact with objects
- Programs are objects too

# Scripting live

<https://www.youtube.com/watch?v=1Nze9tnwYxY>



<http://youtu.be/CuimMwuZiGA>

The image shows a software interface with two windows. The left window is titled "Workspace" and contains the following code:

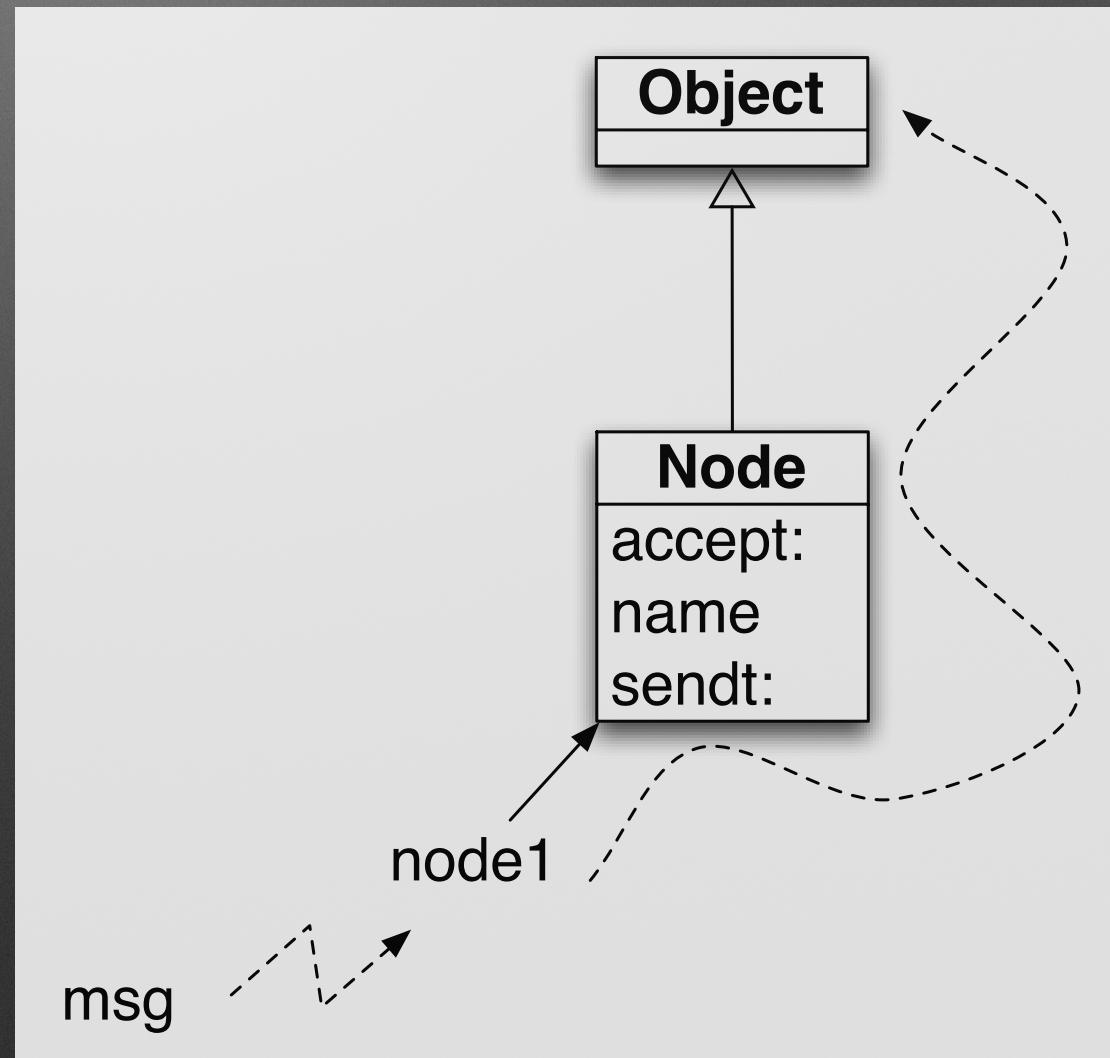
```
| elements lay |  
  
elements := (1 to: 5) collect: [ :ob |  
  (R3CubeShape new) elementOn: ob ].  
    
lay := R3WallLayout new.  
lay on: elements.  
  
UberPresenter present: elements
```

The right window is titled "Uber Presenter" and displays a 3D perspective view of a wall layout consisting of five gray cubes arranged in a staggered pattern.

# Even with the vocabulary

- We send messages to objects.
- We do not invoke methods!

# Sending a message implies looking up for a method



# Simplicity and elegance

- Syntax fits on a postcard!
- Simple but complete object model

# Complete Syntax on a Postcard

exampleWithNumber: x

“A method that illustrates every part of Smalltalk method syntax”

<menu>

| y |

true & false not & (nil isNil) ifFalse: [self halt].

y := self size + super size.

#(\$a #a ‘a’ 1 1.0)

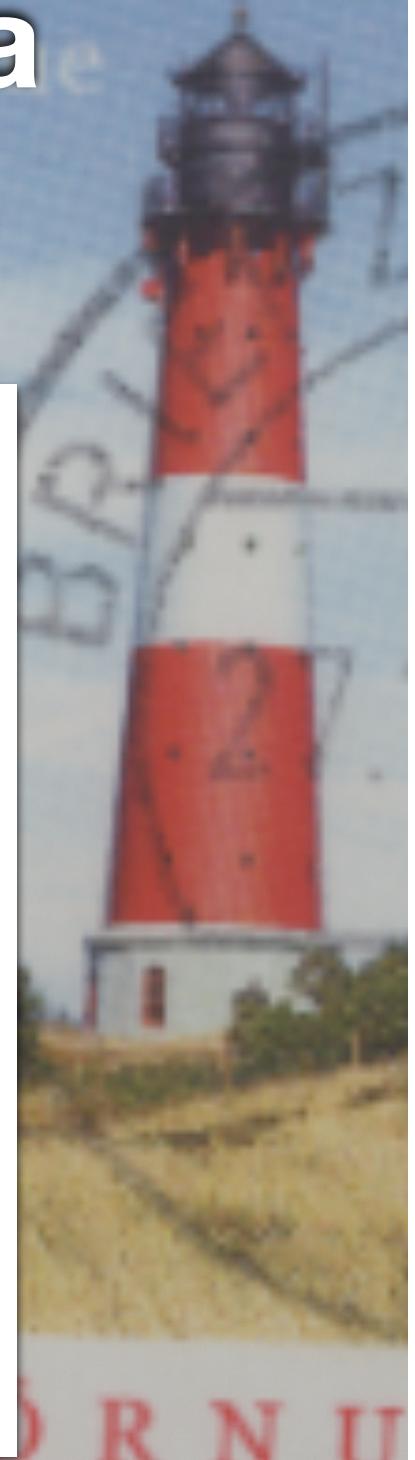
do: [ :each | Transcript

show: (each class name);

show: (each printString);

show: ‘ ’].

^ x < y



# A Pure OO World

# Only objects!

mouse, booleans, arrays, numbers, strings, windows, scrollbars, canvas, files, trees, compilers, sound, url, socket, fonts, text, collections, stack, shortcut, streams, ...

# Less is more!

No constructors, no static methods, no operators

No type declaration, no primitive types,

No interfaces, no need for factory

No packages/private/protected modifiers

No parametrized types

No boxing/unboxing

# Still powerful

# Object Model

- Dynamically typed
- **\*\*Everything\*\*** is an instance of a class
- All methods are public and virtual
- Attributes are protected
- Single Inheritance

**Everything is an  
object**

**Objects are instances of  
Classes**

# Objects are instances of Classes

(10@200)

# Objects are instances of Classes

(10@200) class

# Objects are instances of Classes

(10@200) class

Point

# Classes are objects too

# Classes are objects too

Point selectors

# Classes are objects too

## Point selectors

```
an IdentitySet(#eightNeighbors #+ #isZero #sortsBefore: #degrees #printOn: #sideOf:  
#fourNeighbors #hash #roundUpTo: #min: #min:max: #max #adaptToCollection:andSend:  
#quadrantOf: #crossProduct: #= #nearestPointOnLineFrom:to: #bitShiftPoint: #* #guarded  
#insideTriangle:with:with: #grid: #truncateTo: #y #setR:degrees: #normal #directionToLineFrom:to:  
#truncated #nearestPointAlongLineFrom:to: #theta #scaleTo: #encodePostscriptOn: #> #asPoint  
#extent: #r #roundTo: #max: #interpolateTo:at: #triangleArea:with: #angleWith: #dotProduct:  
#isSelfEvaluating #'<=' #to:intersects:to: #'//' #isInsideCircle:with:with: #< #scaleFrom:to: #corner:  
#to:sideOf: #x #'>=' #roundDownTo: #onLineFrom:to:within: #transposed #ceiling #angle  
#basicType #translateBy: #asFloatPoint #'\\' #adaptToNumber:andSend: #abs #negated #octantOf:  
#asIntegerPoint #flipBy:centerAt: #scaleBy: #floor #onLineFrom:to: #isPoint #reflectedAbout: #/  
#dist: #asNonFractionalPoint #bearingToPoint: #reciprocal #rotateBy:centerAt: #rotateBy:about:  
#rounded #setX:setY: #squaredDistanceTo: #normalized #veryDeepCopyWith: #- #storeOn: #rect:  
#deepCopy #isIntegerPoint #min #adhereTo: #adaptToString:andSend:)
```

# Classes are objects too

Point instVarNames

# Classes are objects too

Point instVarNames

```
>#('x' 'y')
```

**Methods are public**

**Methods are all late-bound**

**Instance variables are  
protected**

# Single Inheritance

# Messages + Objects

$2 + 5$

$> 7$

Yes + is a message sent to 2

1

1 class

1 class

> SmallInteger

1 class maxVal

> 1073741823

# The largest small integer!

1 class maxVal

> 1073741823

(1 class maxVal + 1)

(1 class maxVal + 1)

> 1073741824

# The smallest large integer!

(1 class maxVal + 1)

> 1073741824

$(1 \text{ class maxVal} + 1) \text{ class}$

(1 class maxVal + 1) class

> LargePositiveInteger

1000 factorial

>

## 1000 factorial

1000 factorial / 999 factorial

1000

x - □

## Implementors of factorial [1]

Integer (mathematical functions) factorial [Kernel]

Browse

Senders

Implementors

Version

Source



### factorial

"Answer the factorial of the receiver."

```
self = 0 ifTrue: [^ 1].  
self > 0 ifTrue: [^ self * (self - 1) factorial].  
self error: 'Not valid for negative integers'
```

Learning from the  
system...

# Three questions

- Question 1: how to implement not
- Question 2: how to implement or or ifTrue:ifFalse:
- Question 3: Why these questions?

# Booleans

&, |, not, or: ,and:, xor:

ifTrue:ifFalse:, ifFalse:ifTrue:

3 > 0

ifTrue: ['positive']

ifFalse: ['negative']

-> ‘positive’

ifTrue:ifFalse is sent to an object: a boolean!

# How to implement not?

- false not -> true
- true not -> false

**Without any IF statements!!!**

**Let the receiver decide!**

# Not implementation

Boolean>>not

self subclassResponsibility

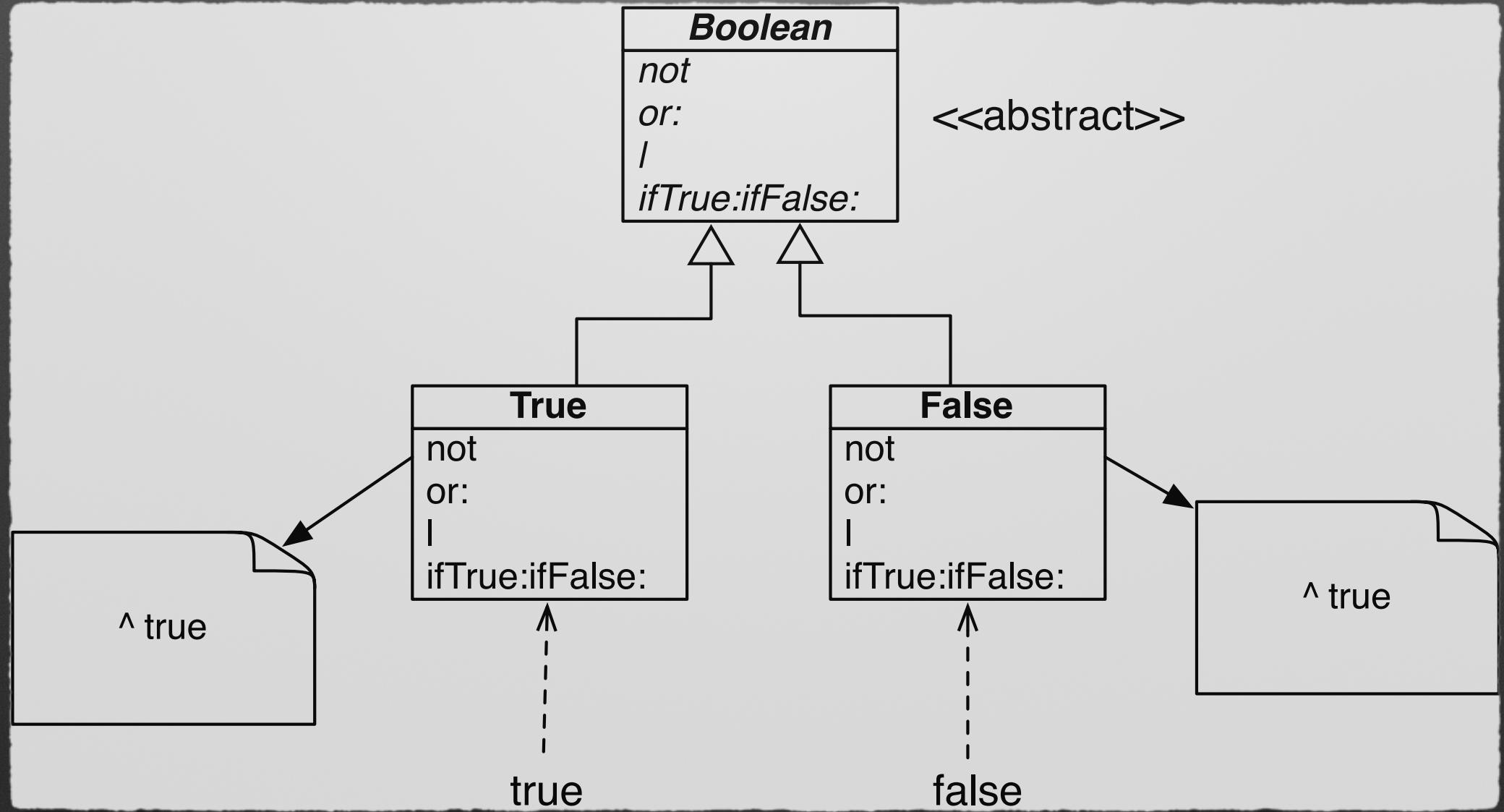
False>>not

^true

True>>not

^false

# Not implementation



# Check in the system

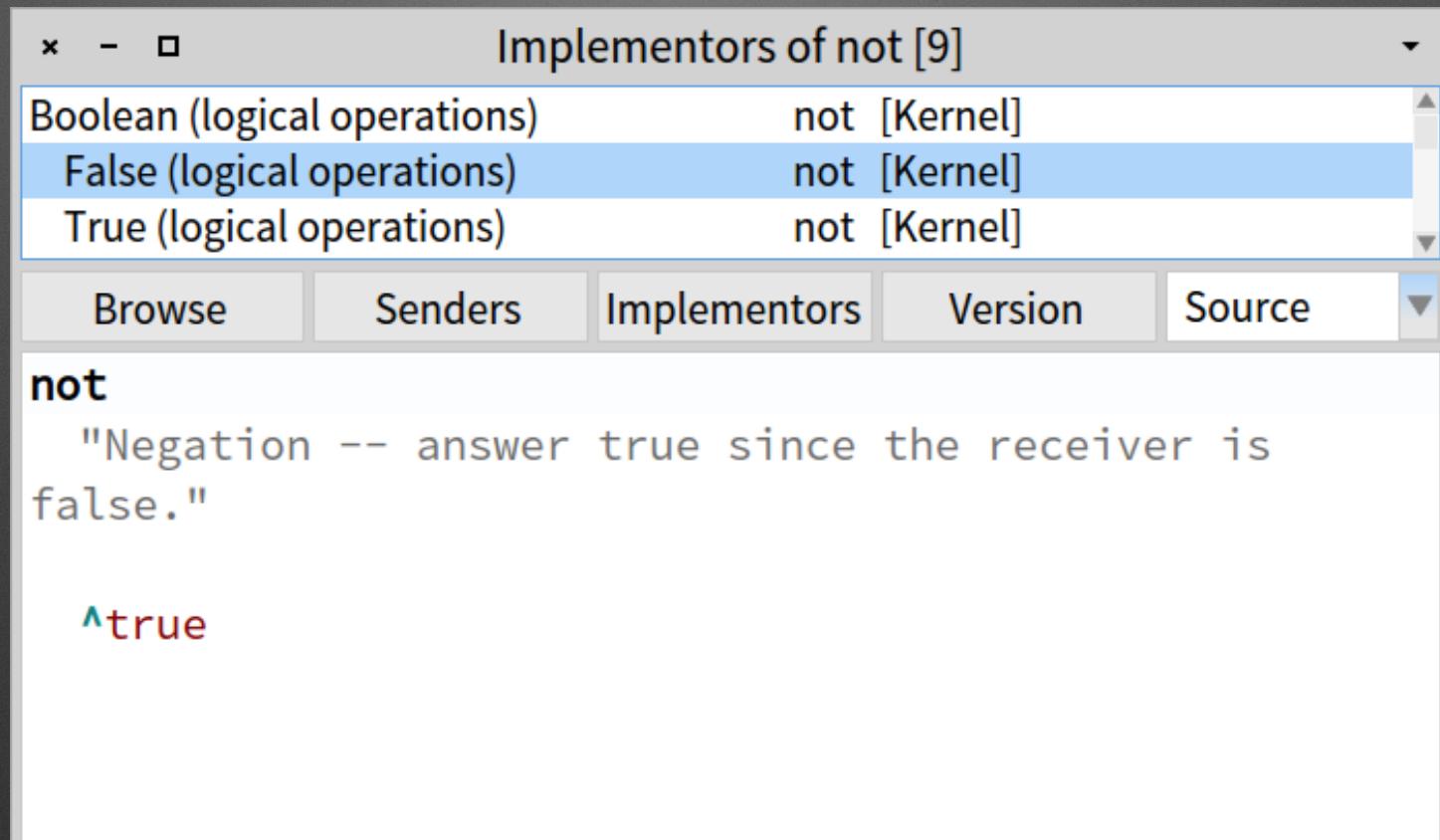
x - □ Implementors of not [9]

Boolean (logical operations)	not [Kernel]
False (logical operations)	not [Kernel]
True (logical operations)	not [Kernel]

Browse Senders Implementors Version Source

**not**  
"Negation -- answer true since the receiver is false."

**^true**



# How to implement or?

**true** | **true** -> **true**

**true** | **false** -> **true**

**true** | **anything** -> **true**

**false** | **true** -> **true**

**false** | **false** -> **false**

**false** | **anything** -> **anything**

## Implementors of | [12]

Boolean (logical operations)	[Kernel]
False (logical operations)	[Kernel]
True (logical operations)	[Kernel]

Browse

Senders

Implementors

Version

Source

| aBoolean

"Evaluating disjunction (OR) -- answer with the argument, aBoolean."

^aBoolean

## Implementors of | [12]

Boolean (logical operations)	[Kernel]
False (logical operations)	[Kernel]
True (logical operations)	[Kernel]

Browse

Senders

Implementors

Version

Source

| aBoolean

"Evaluating disjunction (OR) -- answer true since the receiver is true."

^self

# Ok so what?

- You will probably not implement another Boolean classes in your live
- So is it really that totally useless?

**Message sends act as case  
statements**

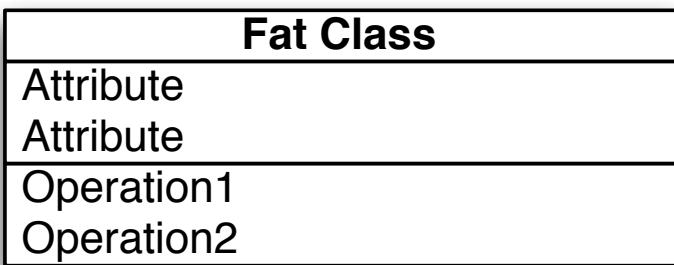
# Message sends act as case statements

- But dynamic and open
- Can add a branch (define a subclass)
- Branches can be defined in different packages

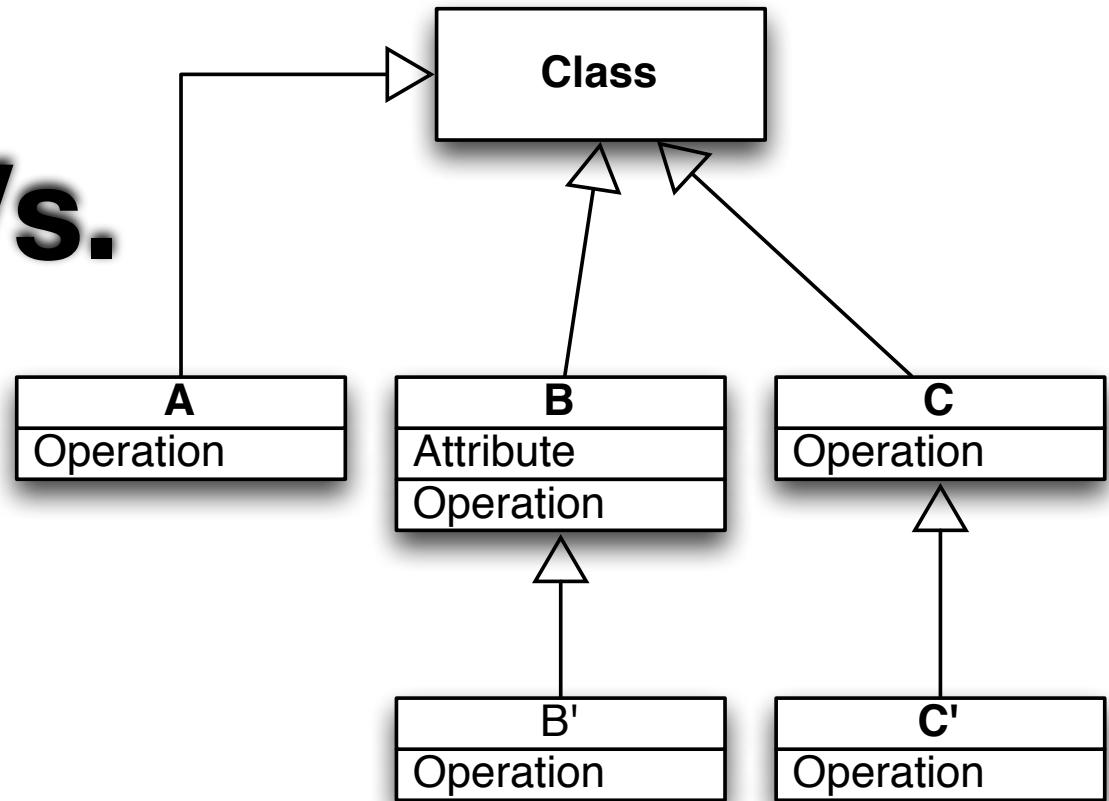
# OOP: the art of dispatching

Subclasses create your vocabulary

# To be able to select we need classes



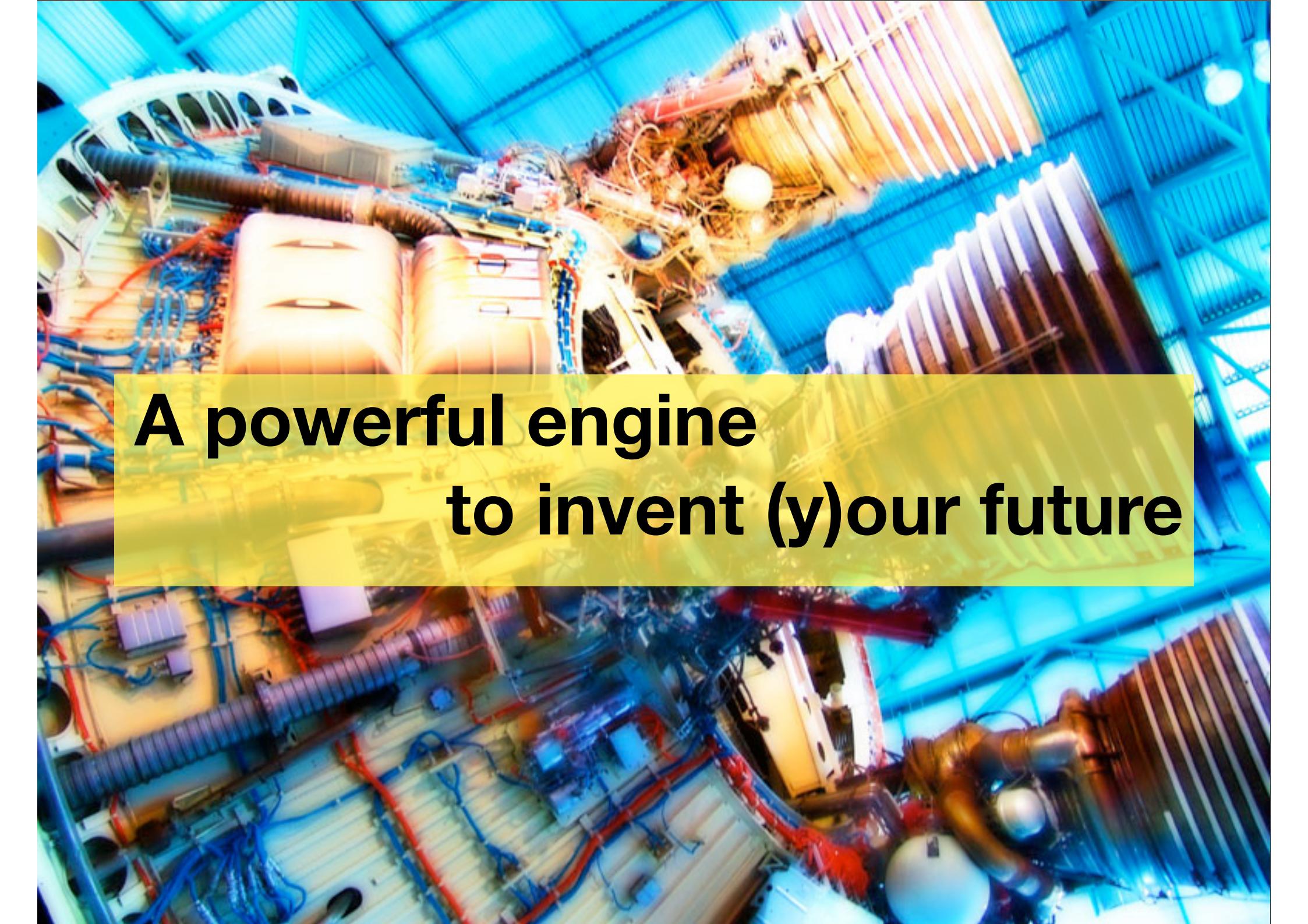
**Vs.**



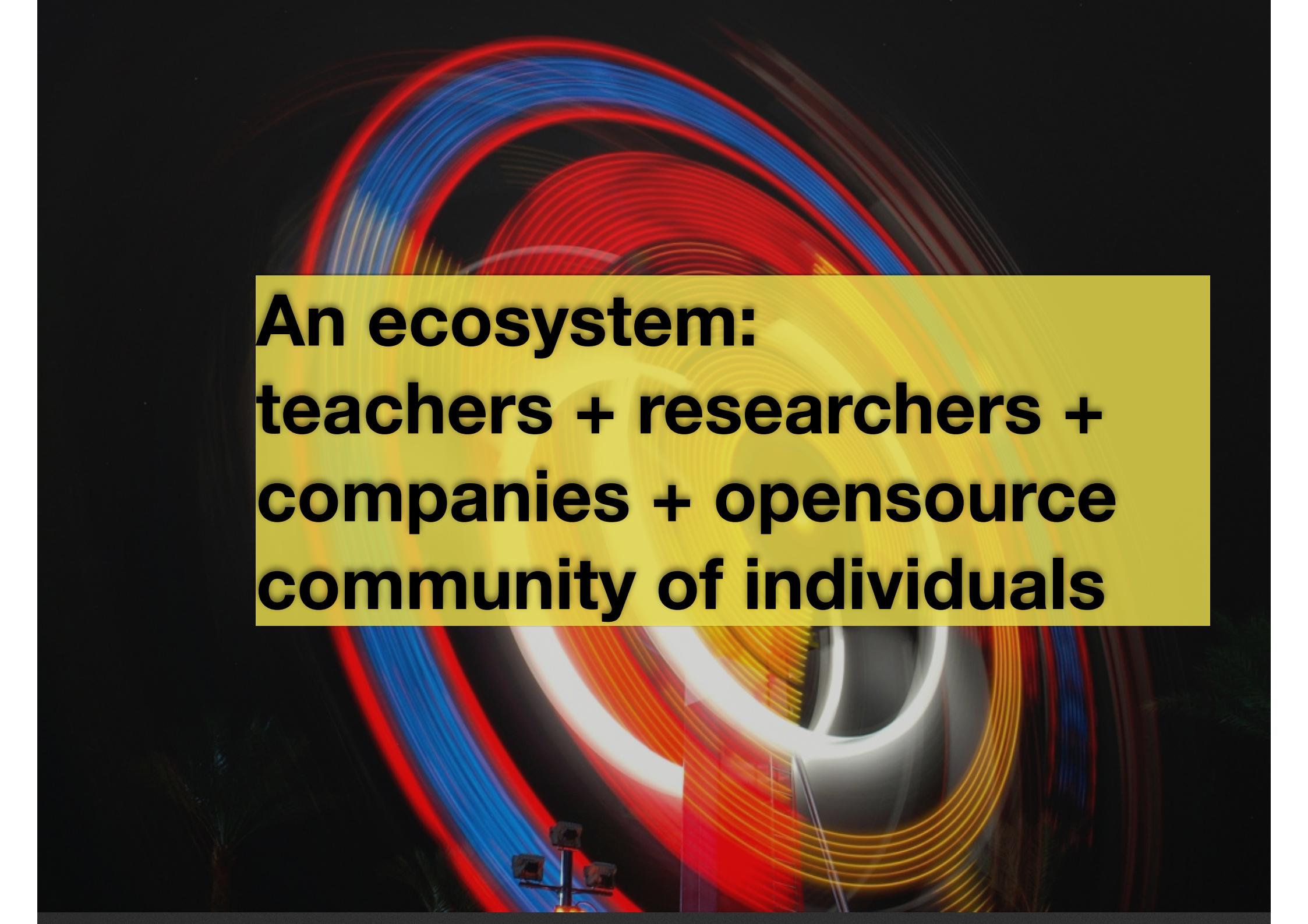
# Pharo

- <http://www.pharo.org>
- Pure object language
- Great community of active doers
- Powerful
- Elegant and fun to program
- Living system under your fingers





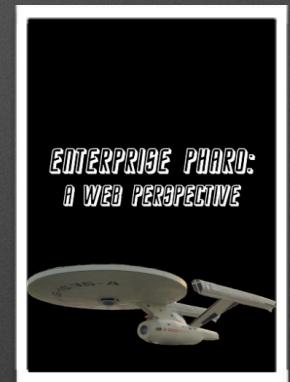
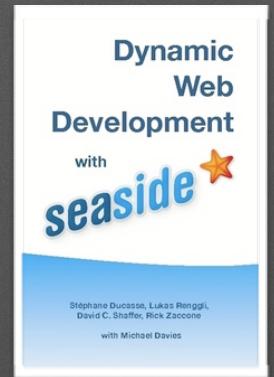
A powerful engine  
to invent (y)our future



**An ecosystem:  
teachers + researchers +  
companies + opensource  
community of individuals**

# Books

- ❖ Pharo by example <http://www.pharobyexample.org>
  - ❖ translated to french, merci! spanish, gracias!
  - ❖ translated to japanese, ありがとう!
- ❖ Dynamic web development with Seaside
- ❖ Deep into Pharo <http://www.deepintopharo.org>
- ❖ New 2015! Entreprise Pharo: a Web perspective
- ❖ Numerical Methods
- ❖ <https://github.com/SquareBracketAssociates/NumericalMethods/releases>
- ❖ New books in preparation
  - ❖ Updated Pharo by Example
  - ❖ Fun with Pharo



[www.pharo.org](http://www.pharo.org)

- Turtles all the way down
- Uniform and elegant
- Simple oo model
- But powerful to build really large applications
- Open system
- Learn from the system itself

# Latest news

- a Mooc is under preparation
- Pharo Starter Kit is ready to be released
  - all resources in one USB