

# Mission Pharo Kernel

ESUG 2016, Prague

Pavel Krivanek and Christophe Demarey

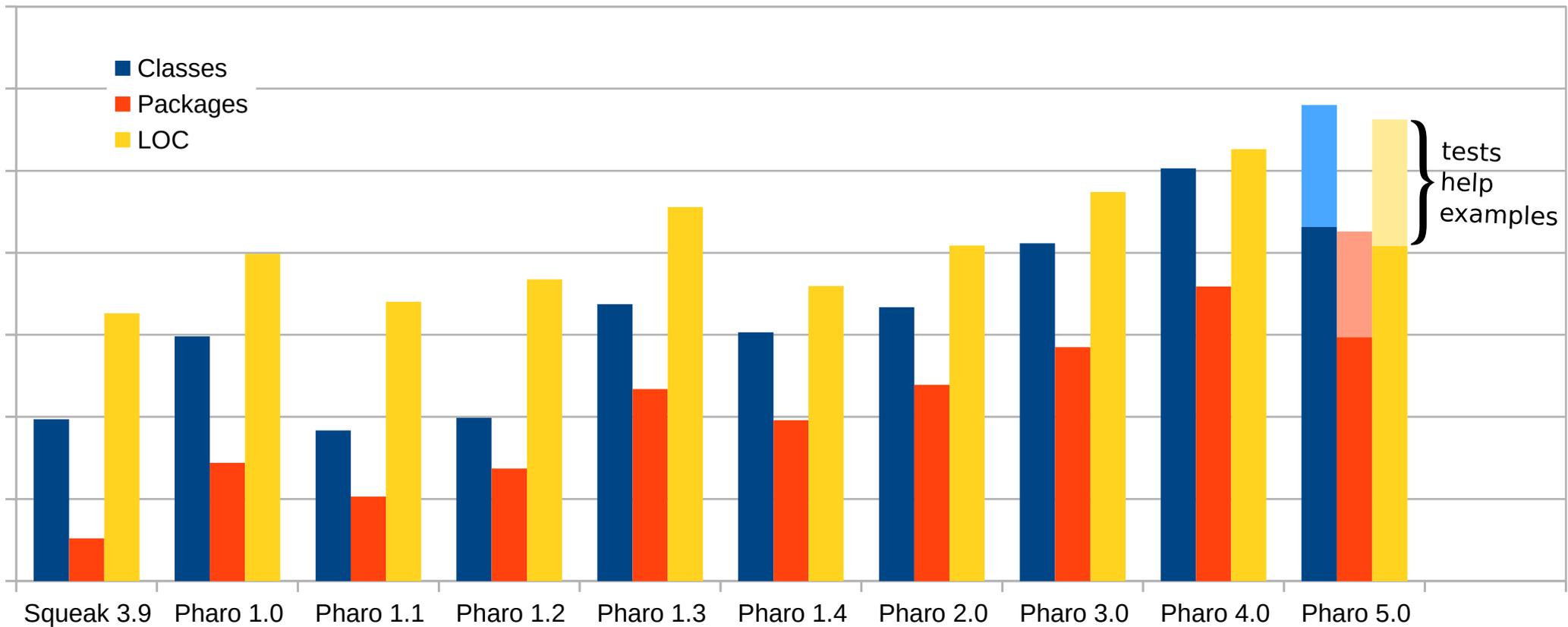


# Pharo is growing

New libraries  
New tools  
New tests  
More documentation



# Pharo evolution





Code for future use  
Duplicities  
Obsolete code  
DO-NOT-TOUCH code



# Pharo ZEN

- Easy to understand, easy to learn from, easy to change.

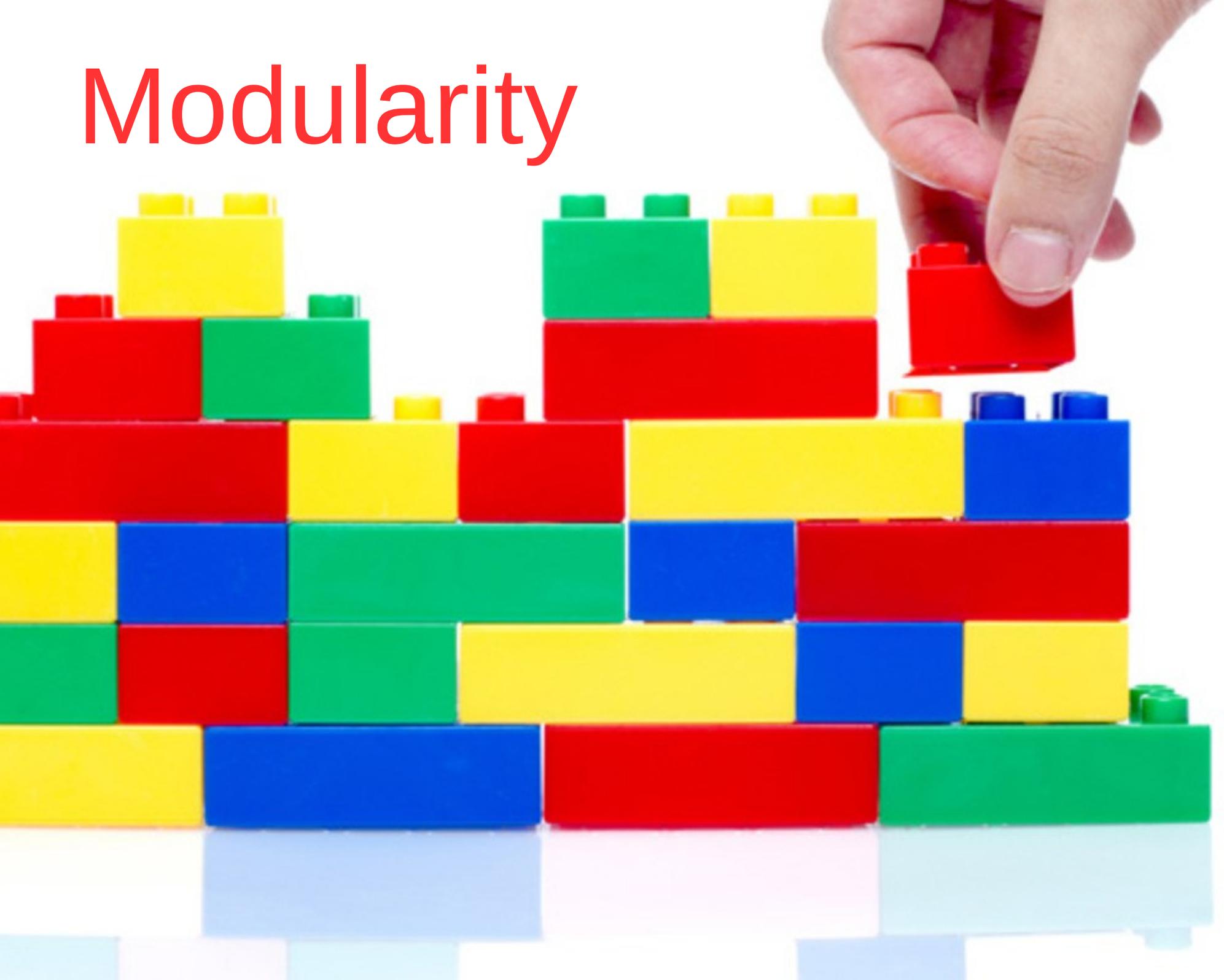
- Objects all the way down.
- Examples to learn from.
- Fully dynamic and malleable.
- Beauty in the code, beauty in the comments.

- Simplicity is the ultimate elegance.

- Better a set of small polymorphic classes than a large ugly one.
- Classes structure our vocabulary.
- Messages are our vocabulary.
- Polymorphism is our esperanto.
- Abstraction and composition are our friends.
- Tests are important but can be changed.
- Explicit is better than implicit.
- Magic only at the right place.
- One step at a time.
- There is no unimportant fix.
- Learning from mistakes.
- Perfection can kill movement.
- Quality is a emerging property.
- Simple processes to support progress.
- Communication is key.

- A system with robust abstractions that a single person can understand.

# Modularity



# Pharo Kernel

**S**mall

**S**imple

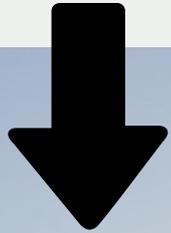
**S**table

**U**nderstandable

**T**ested

**D**ocumented

# From top



- cleaning
- shrinking
- reloading

# From bottom

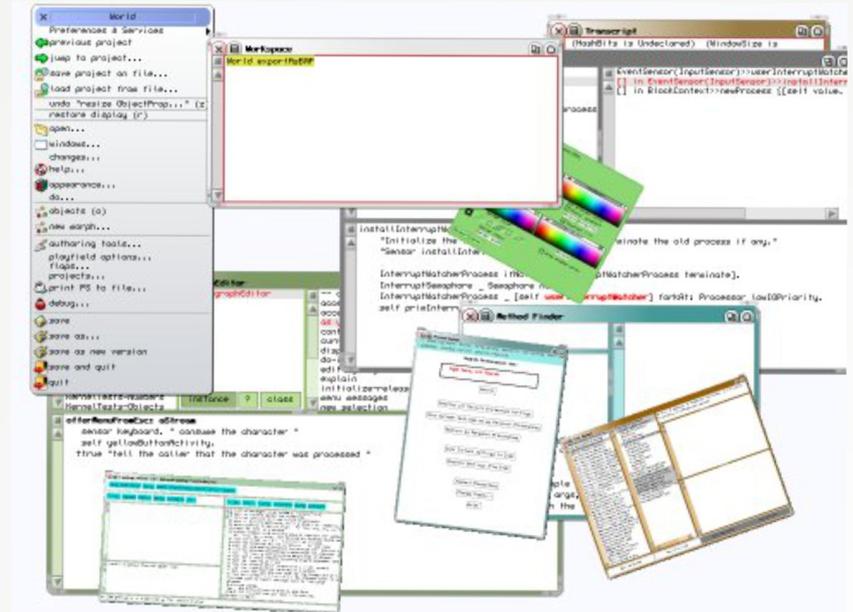


bootstrapping from zero -  
reloading -



# From top shrinking and modularization

- started before Pharo
- removing of code is easy
- clean removing is not easy
- reloading is even harder

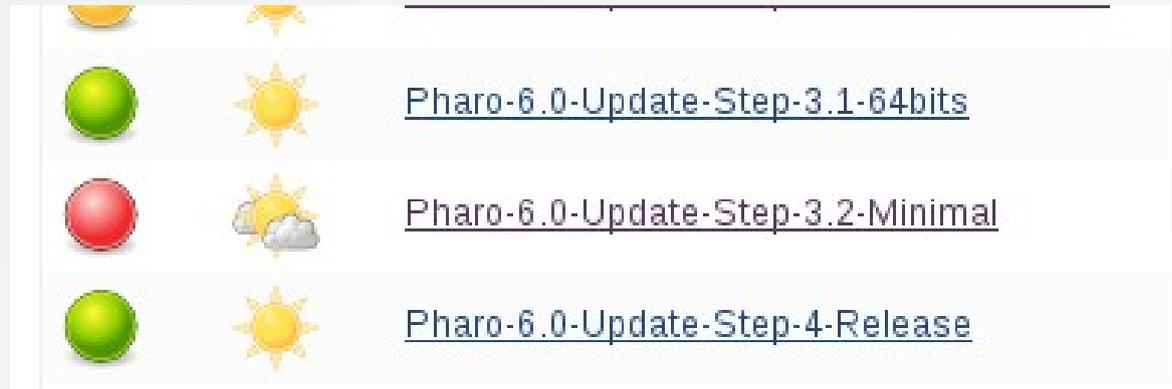


Morphic reloaded [22.07.2006]

# Kernel image evolution in shortcut

**BROKEN!**

works again



Broken again... [18.8.2016]

# Why so hard and long?

A man wearing a green t-shirt, blue jeans, and a tan cap is watering a young orange tree in a field. The tree has several ripe oranges hanging from its branches. The background shows a grassy field with a wooden fence and mountains in the distance under a cloudy sky.

Everyone needs to  
**take of care of  
modularity**

integrated in  
development process  
(tests, rules, CI jobs)

# CI jobs for Pharo modularization

- since Pharo 2.0
- shrink image
- increase granularity of reloaded modules
- tests
- coverage testing
- experiments (Tanker)

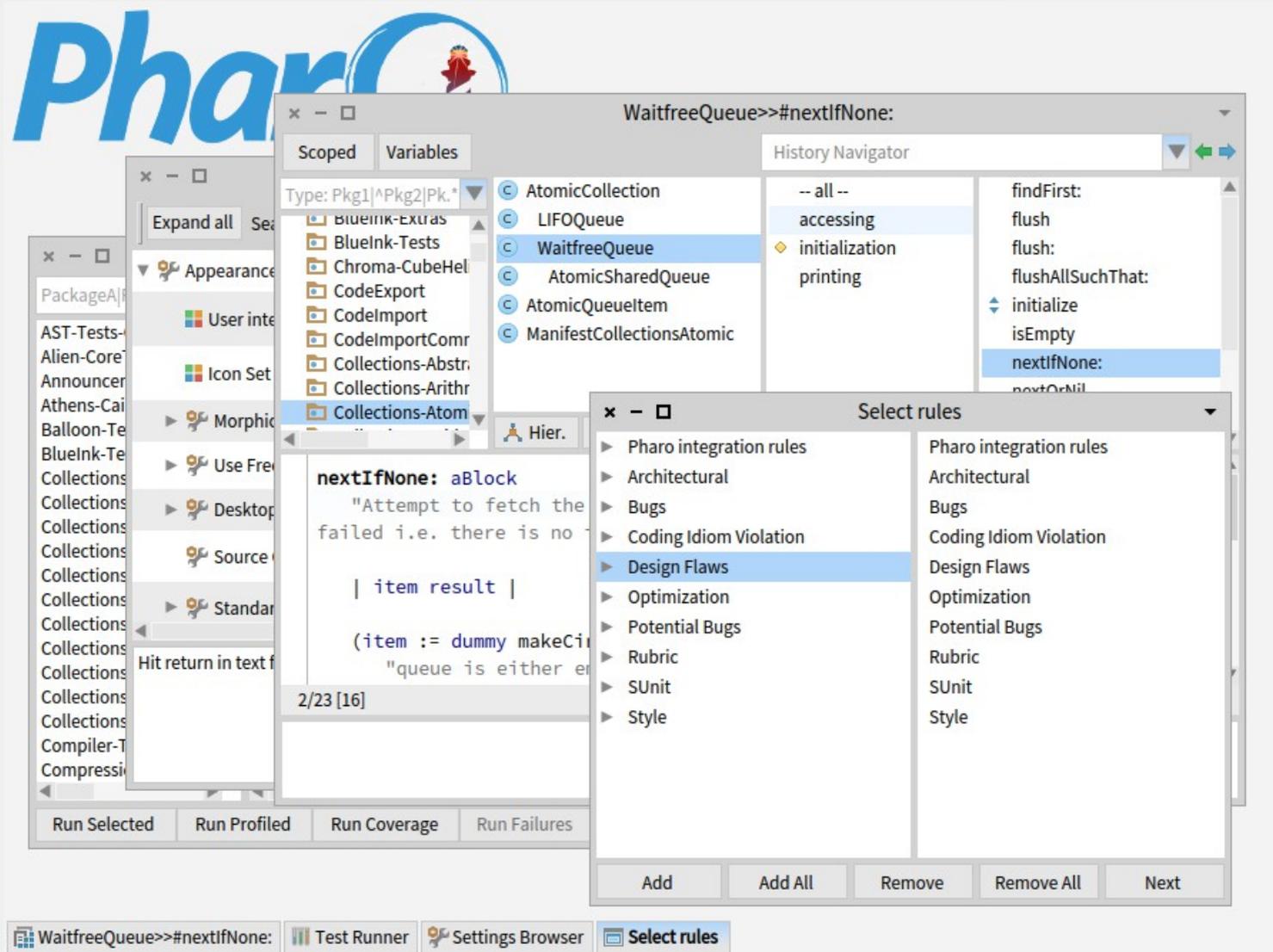
<https://ci.inria.fr/pharo/view/6.0-SysConf/>

# CI jobs for Pharo modularization

## Kernel image (shrunk / bootstrapped)

- + Monticello
- + Network support
- + Remote repositories support
- + Metacello
- = minimal Pharo
- + SUnit, Display support, UFFI
- + Morphic core, Morphic
- + UI, Basic tools, IDE
- = Pharo

S	W	Name
		<a href="#">Pharo-6.0-Step-01-00-Bootstrap</a>
		<a href="#">Pharo-6.0-Step-01-01-ConfigurationOfBootstrapEmulation</a>
		<a href="#">Pharo-6.0-Step-01-02-Bootstrap-Tests</a>
		<a href="#">Pharo-6.0-Step-01-03-Bootstrap-Tests-Coverage</a>
		<a href="#">Pharo-6.0-Step-02-01-ConfigurationOfLocalMonticello</a>
		<a href="#">Pharo-6.0-Step-03-01-ConfigurationOfMonticello</a>
		<a href="#">Pharo-6.0-Step-04-01-ConfigurationOfMinimalPharo</a>
		<a href="#">Pharo-6.0-Step-05-01-ConfigurationOfSUnit</a>
		<a href="#">Pharo-6.0-Step-06-01-ConfigurationOfDisplay</a>
		<a href="#">Pharo-6.0-Step-07-01-ConfigurationOfUnifiedFFI</a>
		<a href="#">Pharo-6.0-Step-08-01-ConfigurationOfMorphicCore</a>
		<a href="#">Pharo-6.0-Step-09-01-ConfigurationOfMorphic</a>
		<a href="#">Pharo-6.0-Step-10-01-ConfigurationOfUI</a>
		<a href="#">Pharo-6.0-Step-11-01-ConfigurationOfBasicTools</a>
		<a href="#">Pharo-6.0-Step-12-01-ConfigurationOfIDE-Raw</a>
		<a href="#">Pharo-6.0-Step-12-02-ConfigurationOfIDE</a>
		<a href="#">Pharo-6.0-Step-12-03-ConfigurationOfIDE-Tests-A-L</a>
		<a href="#">Pharo-6.0-Step-12-03-ConfigurationOfIDE-Tests-M-Z</a>



Bootstrapped & reloaded from GIT

# Let's talk about BOOTSTRAP



with  
broadcast signal is  
**def·i·ni·tion** n. 1.

The teacher gave de  
the new words.

## Bootstrap

« The process that builds the minimal infrastructure of a language reusable to define the language itself »

# Why do we need a bootstrap ?

- Have a known initial state
- Be able to reproduce the state of a system
- Ensure we can reinitialize the system at any time
- Ease Kernel evolution
- Identify a small subset of the language allowing the definition of the language itself

Why bootstrapping is difficult?

The bottom of the slide features abstract geometric shapes. On the left, there is a dark grey triangle pointing downwards. To its right, a large purple shape extends across the bottom, consisting of a dark purple triangle pointing upwards and a lighter purple trapezoidal area below it.

# Archaeology





# Dead code!



# Strange things



# cleanups



# Missing code





# Cut dependencies





# How to fix bad dependencies?

Create a new package to isolate functionalities

Move some methods as extensions to another package

Re-design completely a functionality  
e.g. startup list

...

# Tools support



# Dependency Analyser

The screenshot displays a software interface for Package Dependencies Analysis. The main window is titled "Package Dependencies Analysis" and shows a dependency tree for the package "ZipEncoderTree>>#buildTree:maxDepth:". The tree is organized into a left sidebar and a main content area.

**Left Sidebar (Analysis of 1 package(s)):**

- Compression --> Dependent packages : 19 | Dependencies : 112 (+ 13 e)
  - Collections-Abstract
  - Collections-Native
  - Collections-Sequenceable
    - ZipArchive class>>#validSignatures references Array
    - ZipConstants class>>#initializeFixedTrees references Array
    - ZipEncoderTree>>#buildTree:maxDepth: references Heap**
    - ZipEncoderTree>>#buildTreeFrom:maxDepth: references Array
    - InflateStream>>#decodeDynamicTable:from: references Array
    - InflateStream>>#processDynamicBlock references Array
    - InflateStream>>#createHuffmanTables:counts:from:to: referen
    - InflateStream>>#createHuffmanTables:counts:from:to: referen
    - InflateStream>>#huffmanTableFrom:mappedBy: references An
    - InflateStream>>#computeHuffmanValues:counts:from:to: refer
    - Archive>>#initialize references OrderedCollection
    - ZipEncoderNode>>#leafNodes references Array
  - Collections-Streams
  - Collections-Strings
  - Collections-Unordered
  - FileSystem-Core
  - FileSystem-Disk
  - Files
  - Jobs

**Main Content Area (ZipEncoderTree>>#buildTree:maxDepth:):**

The main content area shows a list of packages and classes. The "Compression" package is expanded, showing its sub-packages and classes. The "ZipEncoderTree" class is selected, and its code snippet is displayed in the bottom right pane.

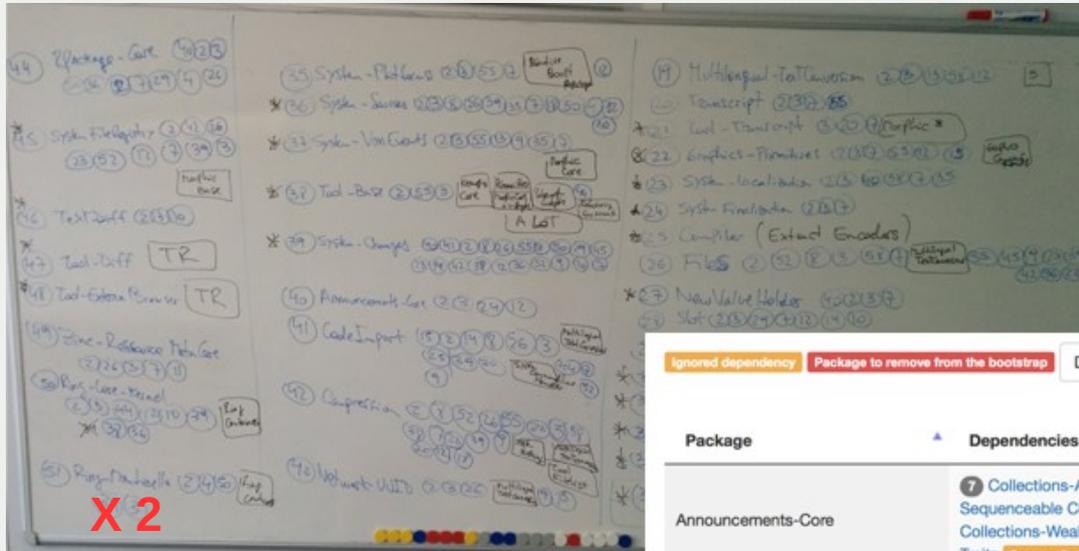
**Code Snippet:**

```
buildTree: nodeList maxDepth: depth
"Build either the literal or the distance tree"
| heap rootNode blCounts |
heap := Heap new: nodeList size // 3.
heap sortBlock: self nodeSortBlock.
"Find all nodes with non-zero frequency and add to heap"
maxCode := 0.
nodeList do:[:dNode]
    dNode frequency = 0 iffFalse:[
        maxCode := dNode value.
```

**Bottom Panel (Warnings):**

- Long methods ? X Helpful? 🟢🔴
- Uses "size = 0" instead of "isEmpty" ? X Helpful? 🟢🔴
- Uses do: instead of collect: or select:'s ? X Helpful? 🟢🔴

# Dependency dashboard



Ignored dependency Package to remove from the bootstrap Dependency graph

Search:

Package	Dependencies	Dependants
Announcements-Core	<ul style="list-style-type: none"> <li>7 Collections-Abstract Collections-Sequenceable Collections-Unordered Collections-Weak Kernel System-Finalization Traits <b>System-Settings</b></li> </ul>	<ul style="list-style-type: none"> <li>9 AST-Core Jobs Monticello RPackage-Core System-Announcements System-Changes System-Localization System-Model System-Sources</li> </ul>
AST-Core	<ul style="list-style-type: none"> <li>15 AST-FFI-Pharo50Compatibility Announcements-Core Collections-Abstract Collections-Native Collections-Sequenceable Collections-Streams Collections-Strings Collections-Unordered Collections-Weak Kernel OpalCompiler-Core System-Announcements System-SessionManager Traits Transcript <b>System-Settings</b></li> </ul>	<ul style="list-style-type: none"> <li>11 AST-FFI-Pharo50Compatibility CodeImport Collections-Strings FileSystem-Core Kernel Multilingual-Encodings Network-UUID OpalCompiler-Core System-Hashing Traits Zinc-Character-Encoding-Core</li> </ul>
AST-FFI-Pharo50Compatibility	<ul style="list-style-type: none"> <li>6 AST-Core Collections-Sequenceable Collections-Strings FFI-Kernel Kernel OpalCompiler-Core</li> </ul>	<ul style="list-style-type: none"> <li>2 AST-Core FFI-Kernel</li> </ul>
CodeExport	<ul style="list-style-type: none"> <li>15 Collections-Abstract Collections-Streams Collections-Strings Collections-Unordered FileSystem-Core FileSystem-Disk Files Kernel Multilingual-TextConversion RPackage-Core Slot System-Localization System-Support Traits</li> </ul>	<ul style="list-style-type: none"> <li>6 Monticello OpalCompiler-Core System-Changes System-Hashing System-Sources Traits</li> </ul>

# Dependency visualization

<https://ci.inria.fr/pharo/job/Pharo-6.0-DependencyAnalysis/ws/bootstrap-dependency-report-graph.html>

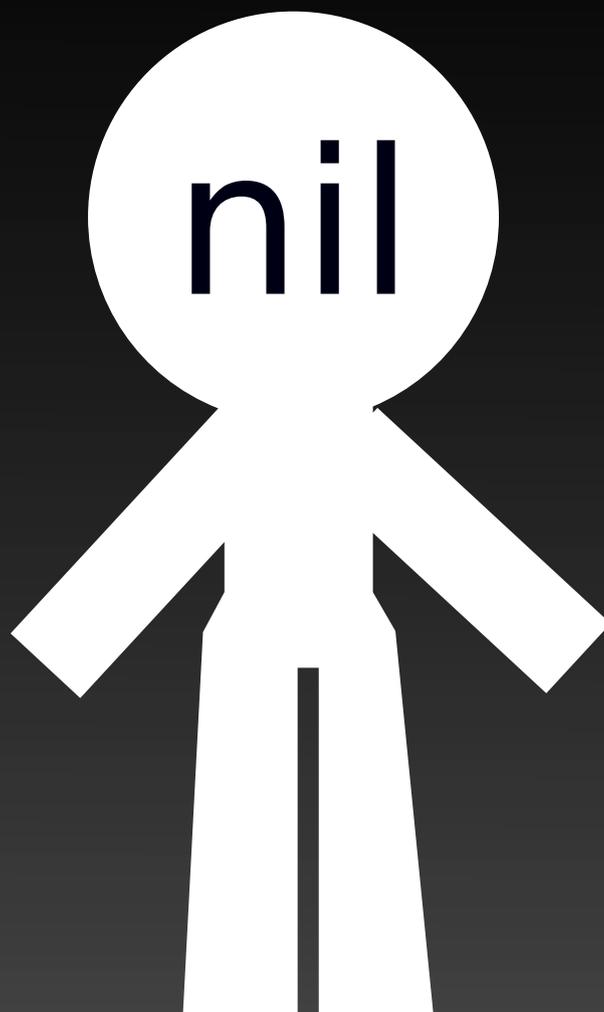
# The bootstrap process

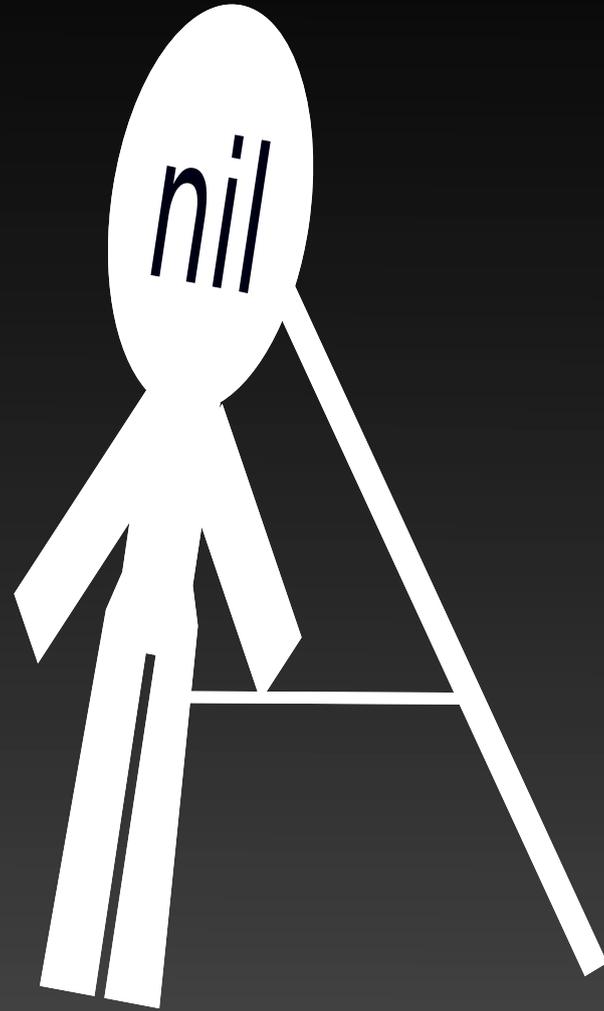




Day

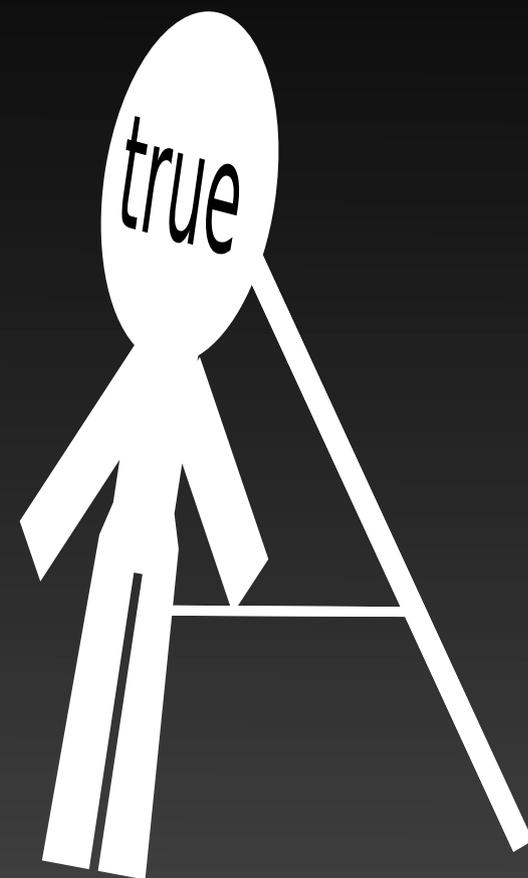
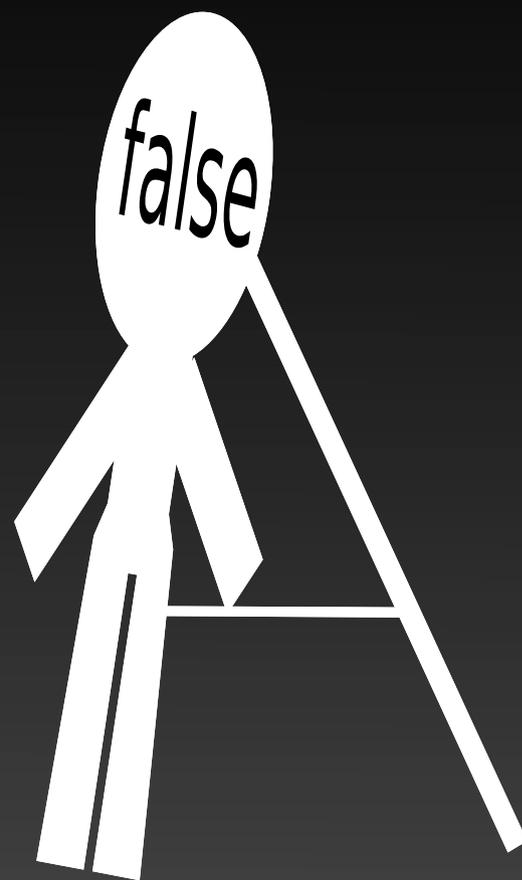
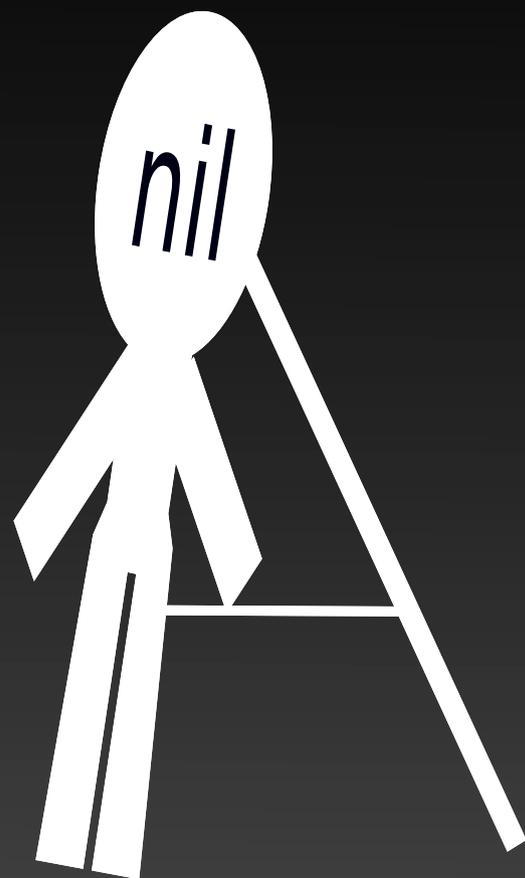
1





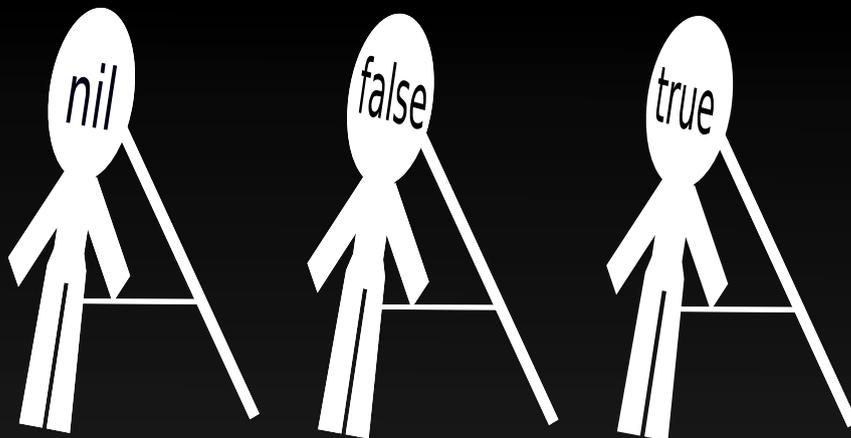
Day

2

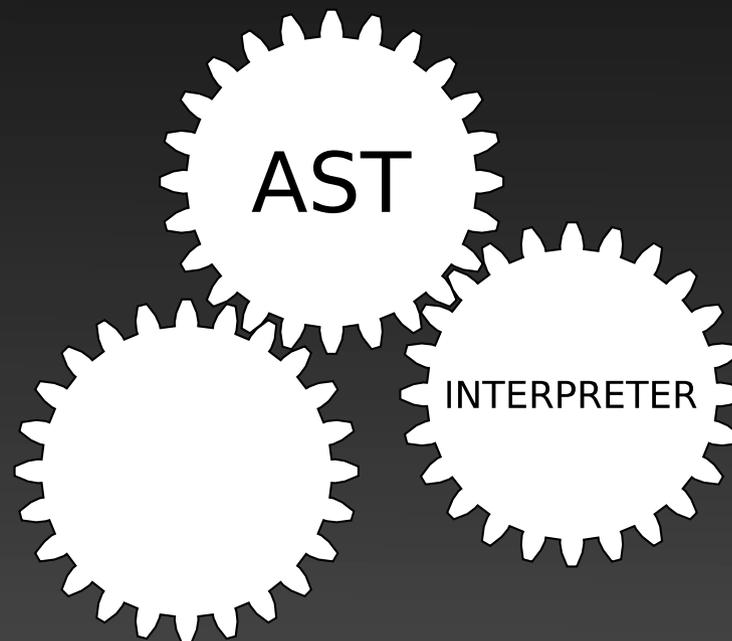


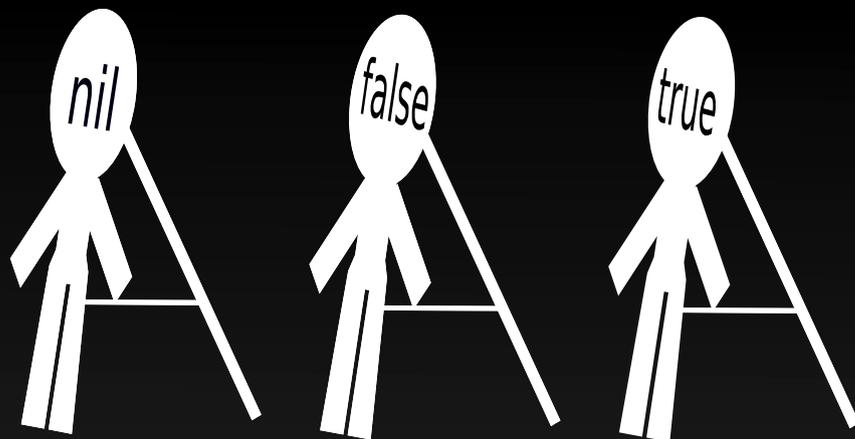
Day

3



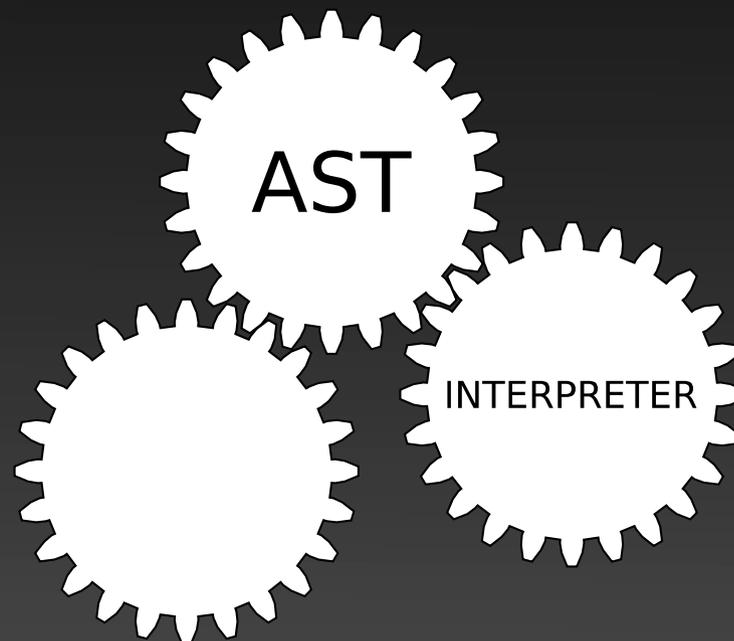
classes





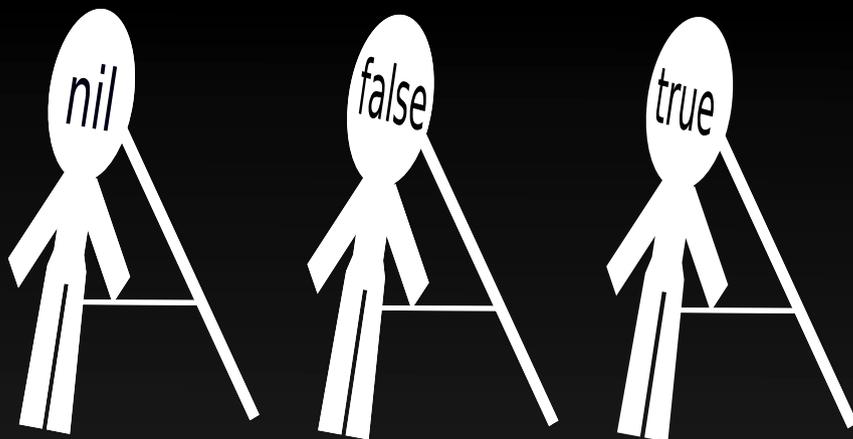
classes

methods



Day

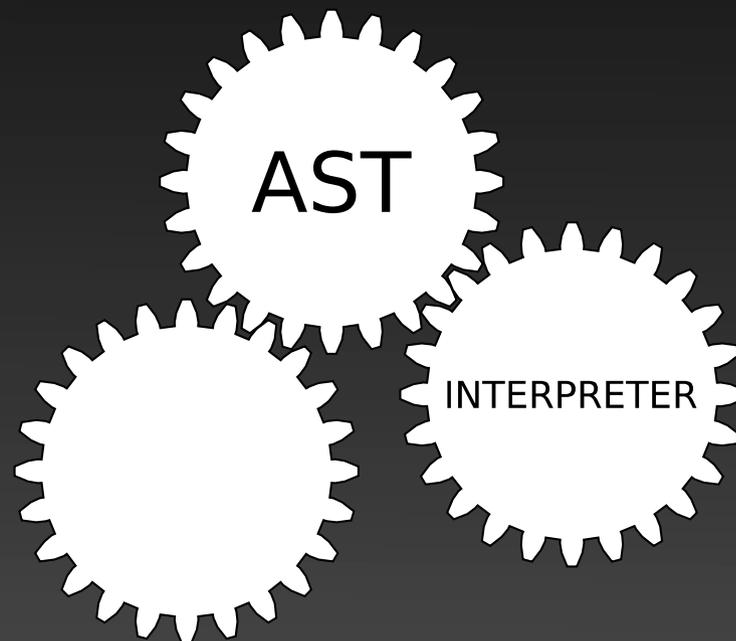
5

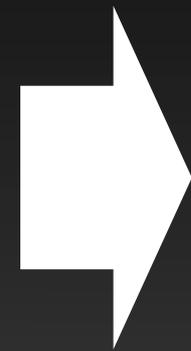
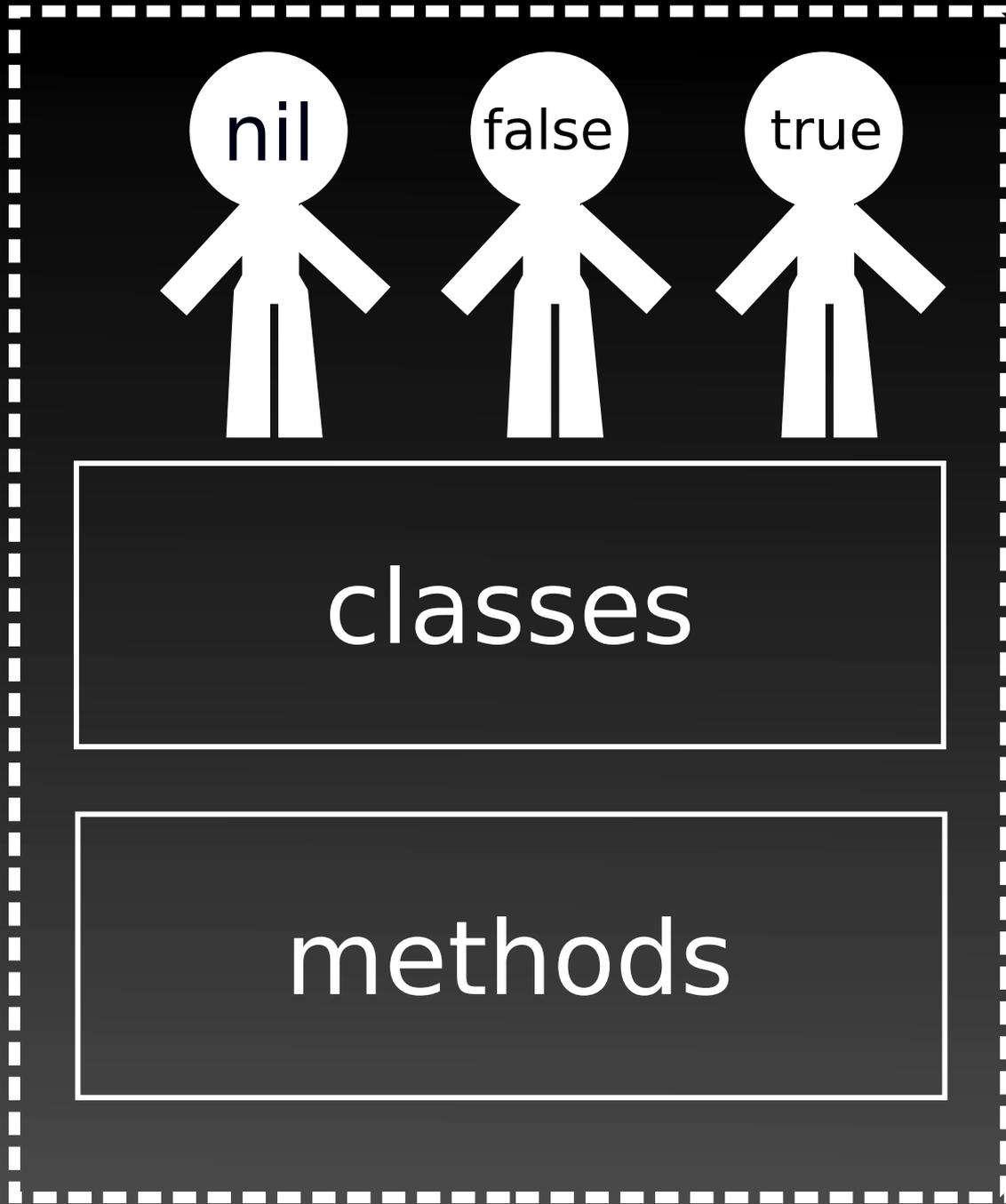


classes

methods

( process )





Day

7



# More details



cf phd Guillermo Polito:  
<https://hal.inria.fr/tel-01251173>

# Story #2



# Road to a working bootstrap



# Bootstrap challenge

- > language side bootstrap

Language initialization generally done VM side

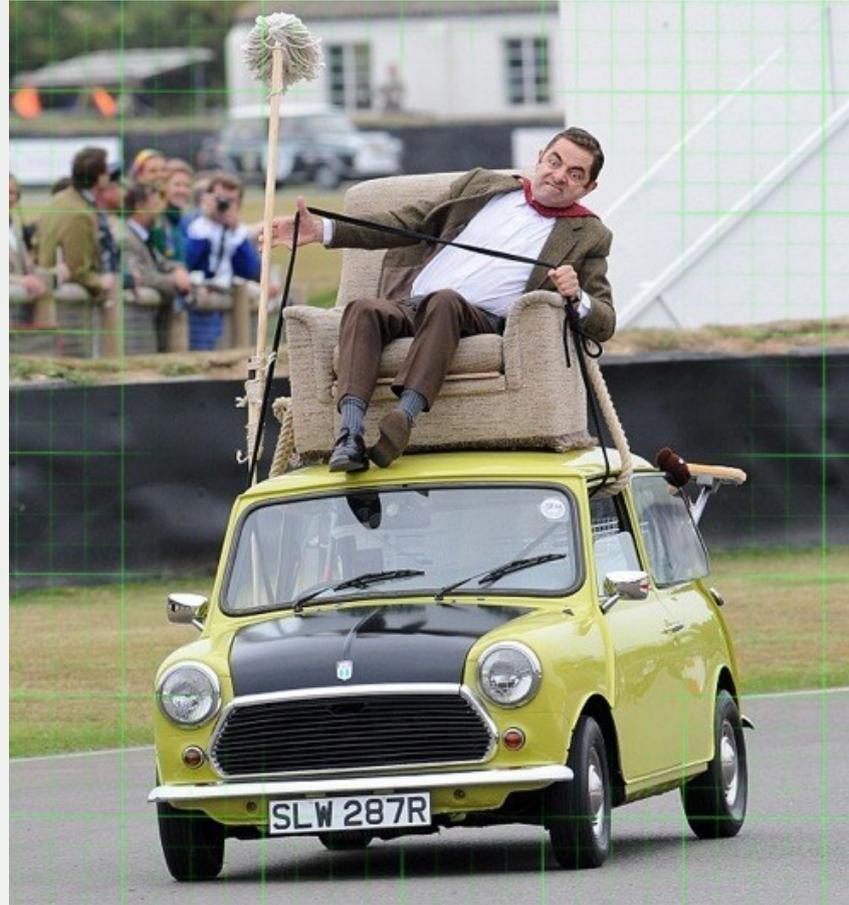
We want to do it language side (image side):

need to run code on top of a language under construction



# Bootstrap challenge

- > language side bootstrap



# Road to a working bootstrap

First bootstrapped image!



# Road to a working bootstrap



# Road to a working bootstrap

We run the image ...

... VM crash



# Road to a working bootstrap

- > some debugging examples

Missing class in the bootstrap  
e.g. Float

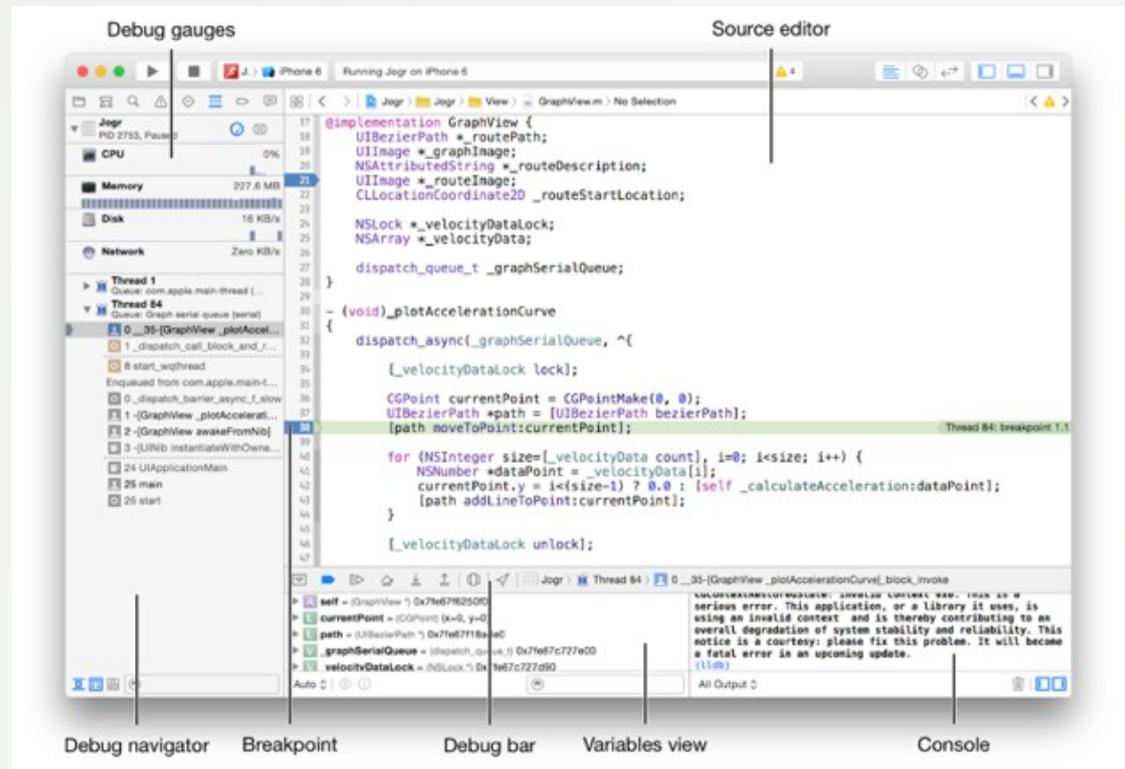
Super class not set

Super class set to a wrong value

# Road to a working bootstrap

Compile VM in debug mode

Run bootstrapped image through Xcode / LLDB



# Road to a working bootstrap

## > verifying the bootstrap

Rely on Pharo tests (>8 000 tests)

- Load SUnit
- Load test packages
- Run tests



# Wants to know more?

Bootstrap process hosted on Pharo CI server

<https://ci.inria.fr/pharo/view/Pharo%20bootstrap/>

GitHub repository

<https://github.com/guillep/PharoBootstrap>

# Conclusion

Having a modular system requires a lot of energy

- Easy to break
- Concern of everyone

We now have a working Pharo bootstrapped image.

We are able to load packages on top to build a full Pharo-image (UI, IDE, etc.)

# Roadmap

Make the bootstrap process more robust

Up-to-date package dependencies for the Pharo image + use of Cargo package manager

Build the official Pharo image on top of the bootstrap

Make the kernel smaller (e.g. kick out Unicode)



**« Always leave the campground  
cleaner than you found it. »**

*-Boy Scout rule, Uncle Bob*