Mission Pharo Kernel

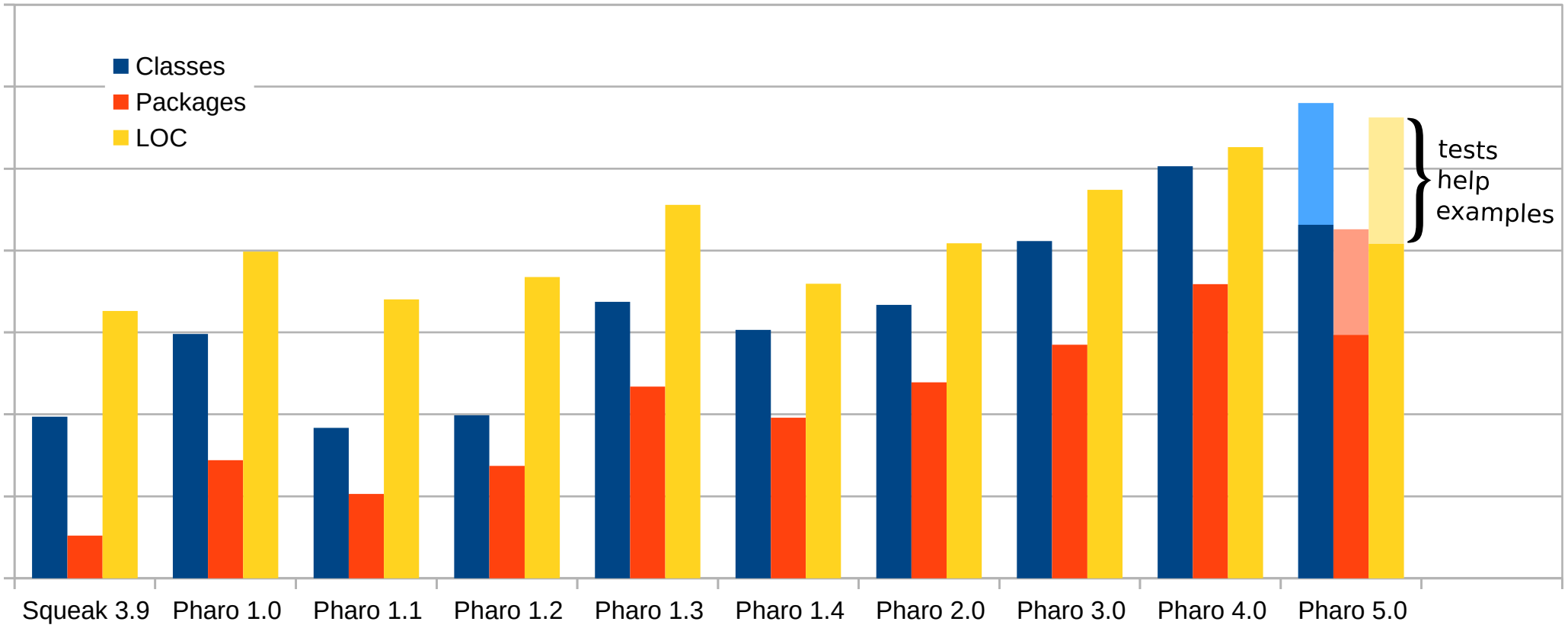
ESUG 2016, Prague

Pavel Krivanek and Christophe Demarey

Pharo is growing

New libraries
New tools
New tests
More documentation

Pharo evolution

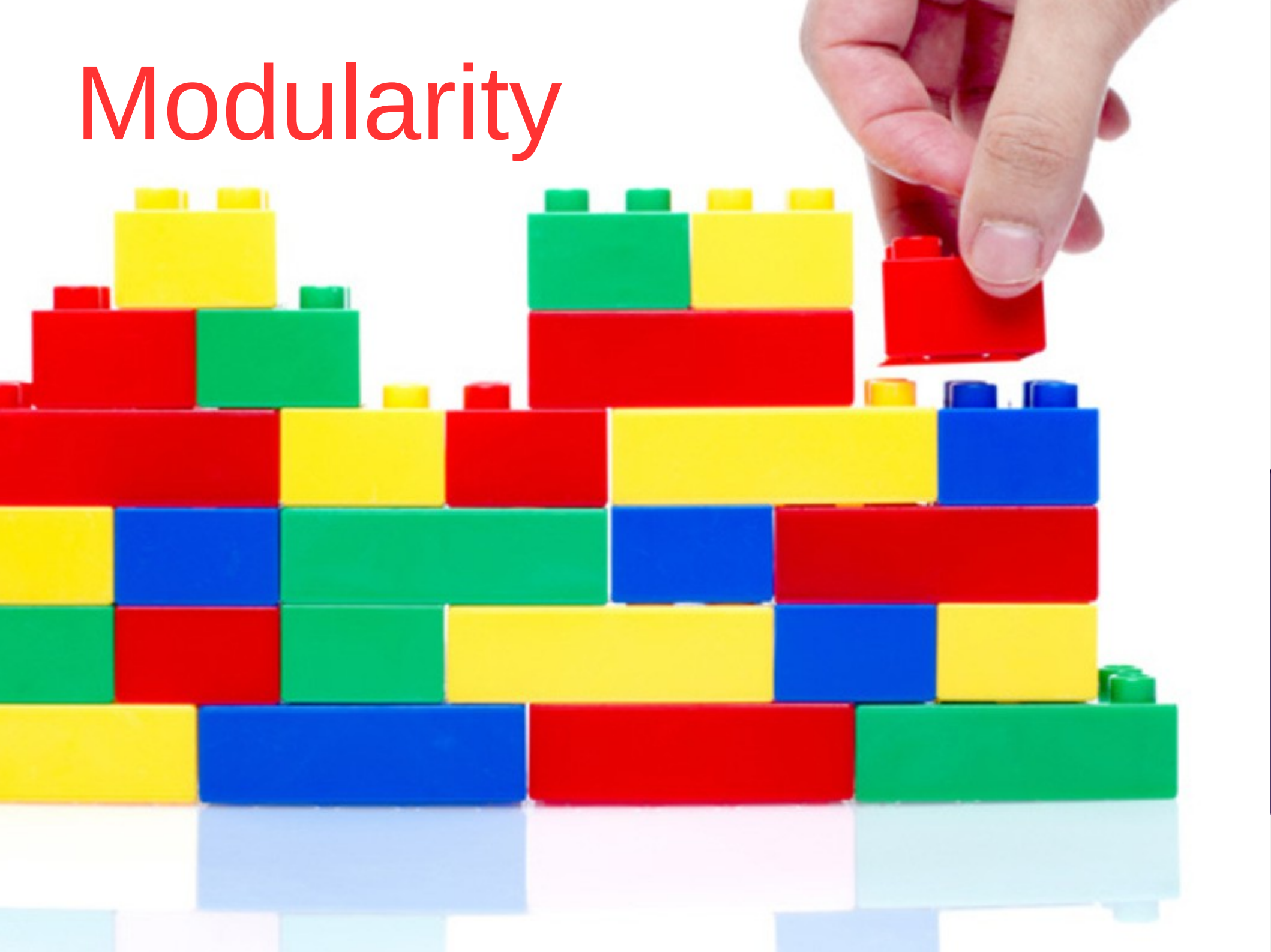



Code for future use
Duplicities
Obsolete code
DO-NOT-TOUCH code

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

Small
Simple
Stable
Understandable
Tested
Documented

From top

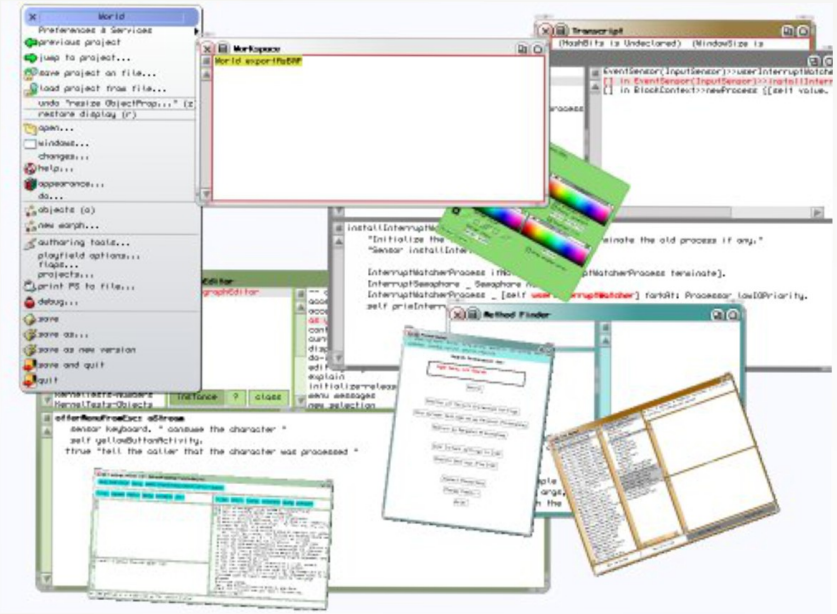
- cleaning
- shrinking
- reloading

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!
works again
BROKEN!
works again
BROKEN!
works again
BROKEN!
works again
BROKEN!
works again
BROKEN!
works again

Broken again... [18.8.2016]

Why so hard and long?

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

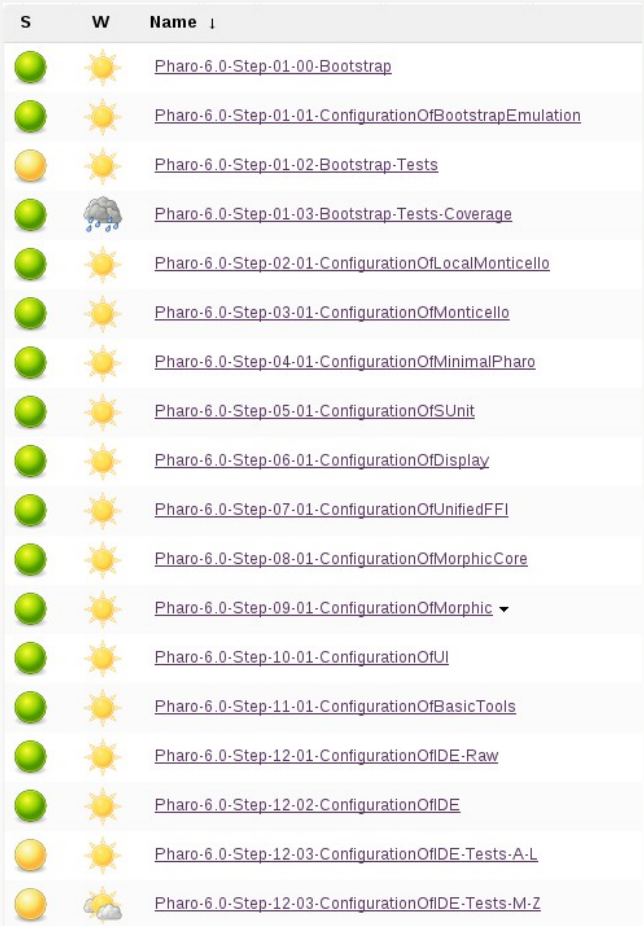
Kernel image (shrinked / bootstrapped)

+ Monticello
+ Network support
+ Remote repositories support
+ Metacello

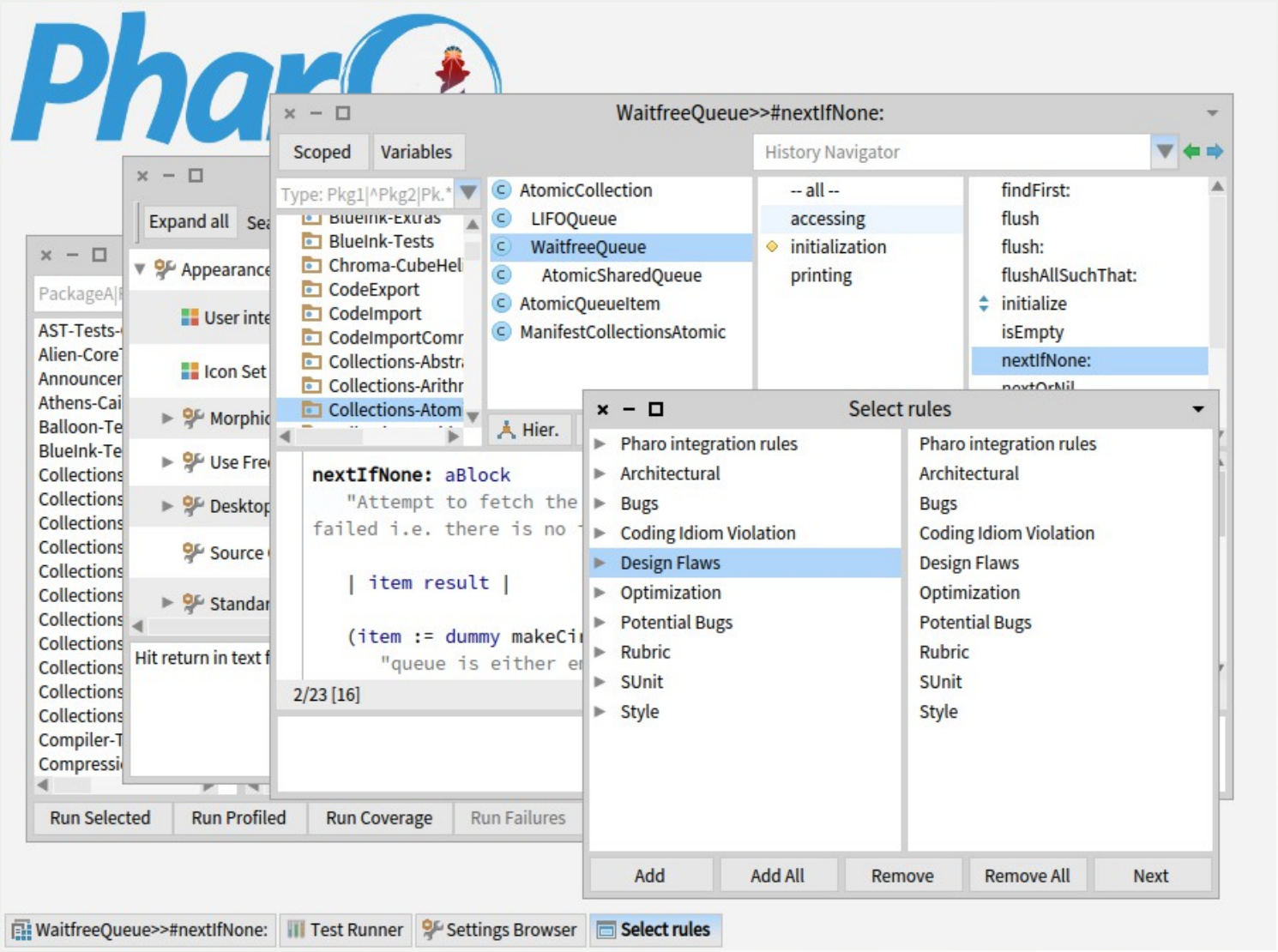
= minimal Pharo

+ SUnit, Display support, UFFI
+ Morphic core, Morphic
+ UI, Basic tools, IDE

= Pharo



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



Bootstrapped & reloaded from GIT

Let's talk about BOOTSTRAP

Bootstrap

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

Why do we need a bootstrap ?

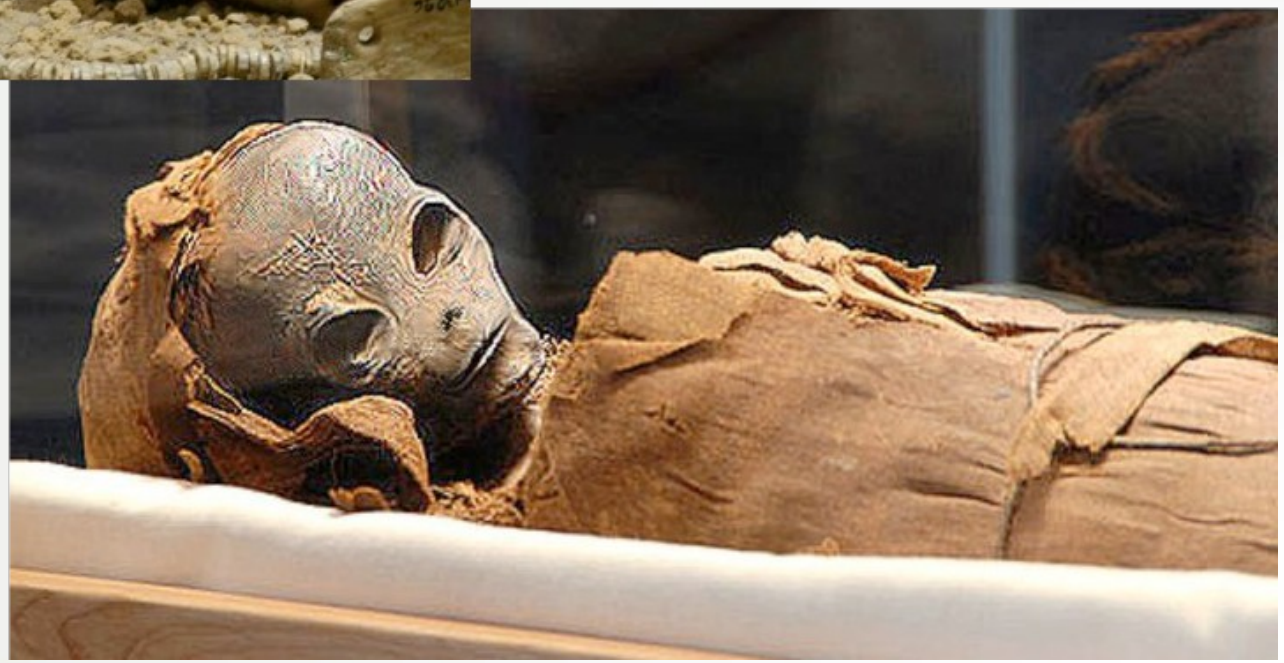
- Have a known initial state
- Be able to reproduce the state of a system
- Ensure we can reinitialize the system at any time
- Ease Kernel evolution
- Identify a small subset of the language allowing the definition of the language itself

Why bootstraping is difficult?

Archaelogy

Dead code!

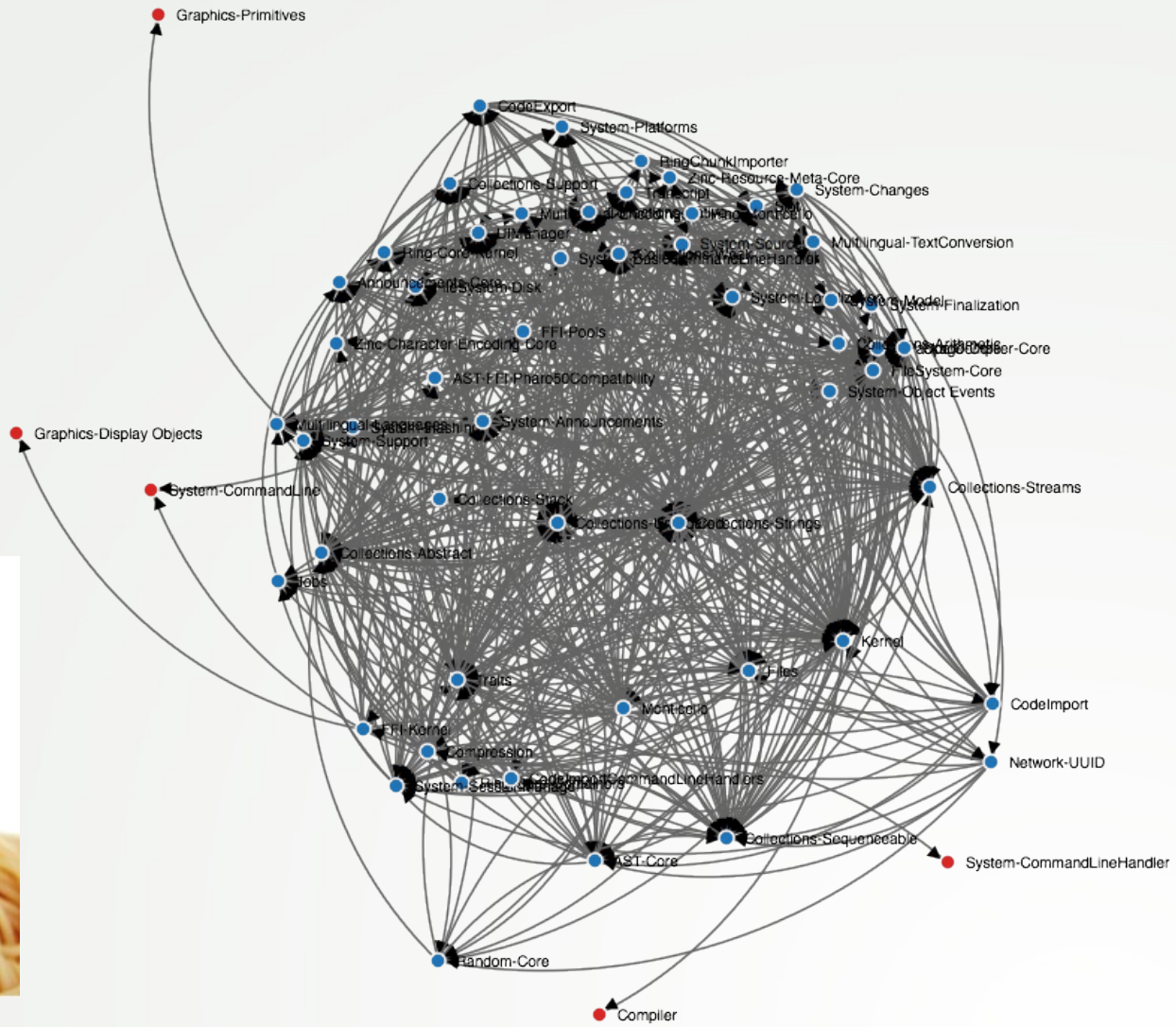
Strange things



cleanups

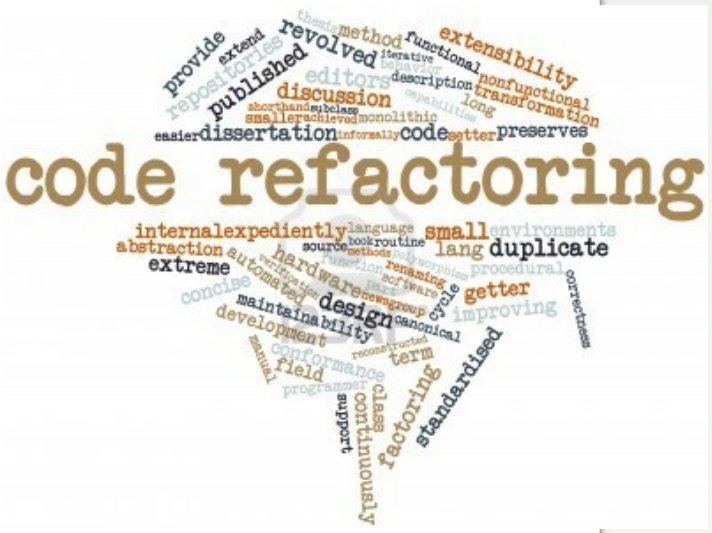
Missing code

The dependency hell



Cut dependencies

KEEP CALM AND REFACTOR CODE



How to fix bad dependencies?

Create a new package to isolate functionnalities

Move some methods as extensions to another package

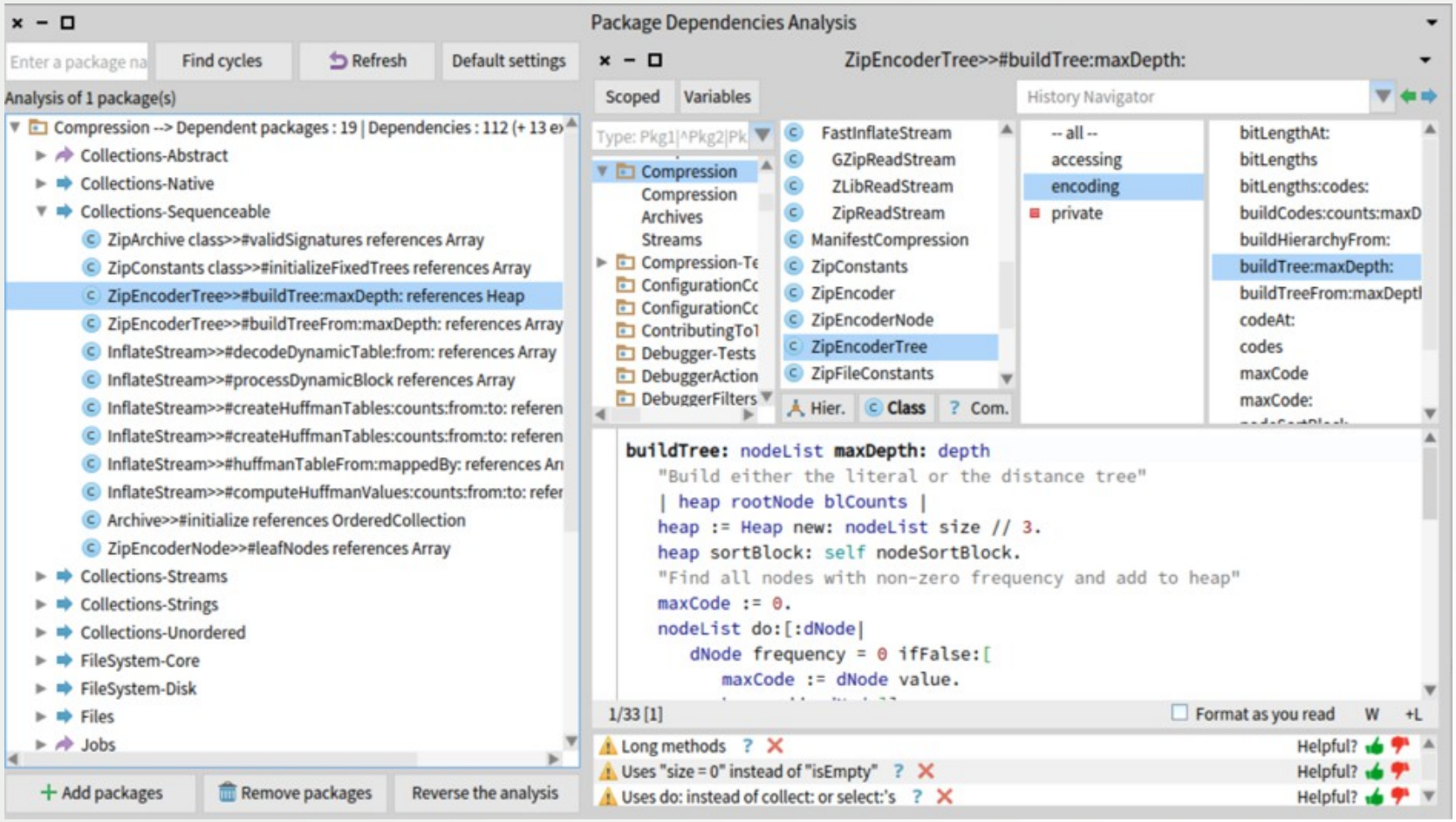
Re-design completely a functionality

- e.g. startup list

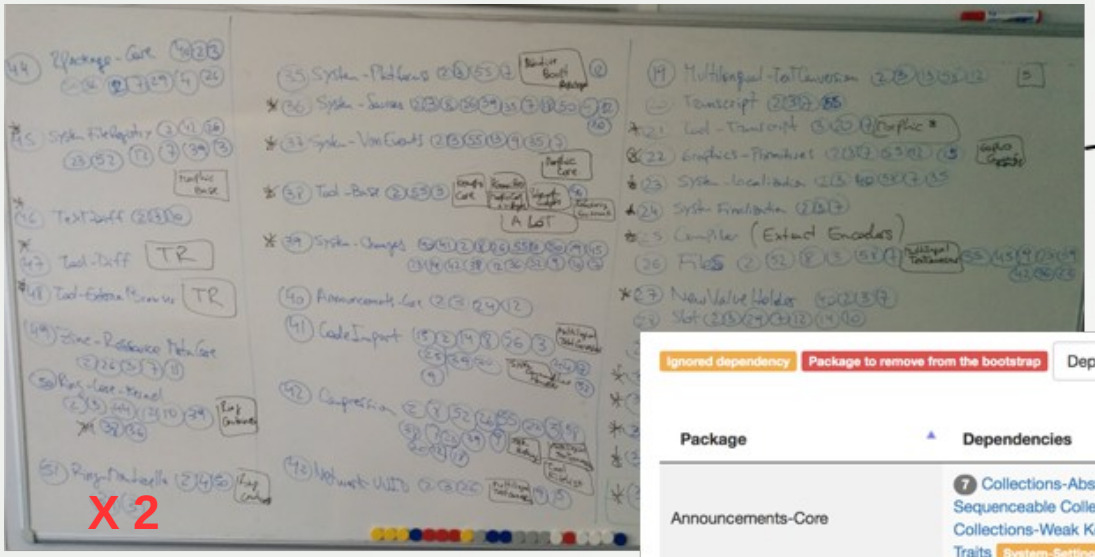
...

Tools support

Dependency Analyser



Dependency dashboard



Ignored dependency | Package to remove from the bootstrap | Dependency graph

Search:

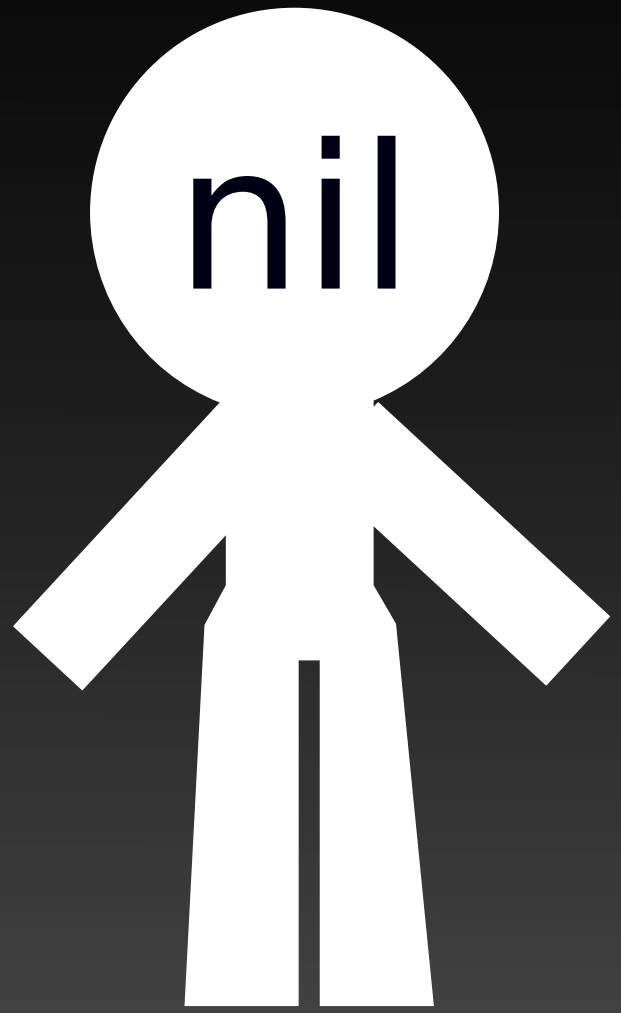
Package	Dependencies	Dependants
Announcements-Core	7 Collections-Abstract Collections-Sequenceable Collections-Unordered Collections-Weak Kernel System-Finalization Traits System-Settings	9 AST-Core Jobs Monticello RPackage-Core System-Announcements System-Changes System-Localization System-Model System-Sources
AST-Core	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 System-Settings	11 AST-FFI-Pharo50Compatibility CodeImport Collections-Strings FileSystem-Core Kernel Multilingual-Encodings Network-UUID OpalCompiler-Core System-Hashing Traits Zinc-Character-Encoding-Core
AST-FFI-Pharo50Compatibility	6 AST-Core Collections-Sequenceable Collections-Strings FFI-Kernel Kernel OpalCompiler-Core	2 AST-Core FFI-Kernel
CodeExport	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	6 Monticello OpalCompiler-Core System-Changes System-Hashing System-Sources Traits

Dependency visualization

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

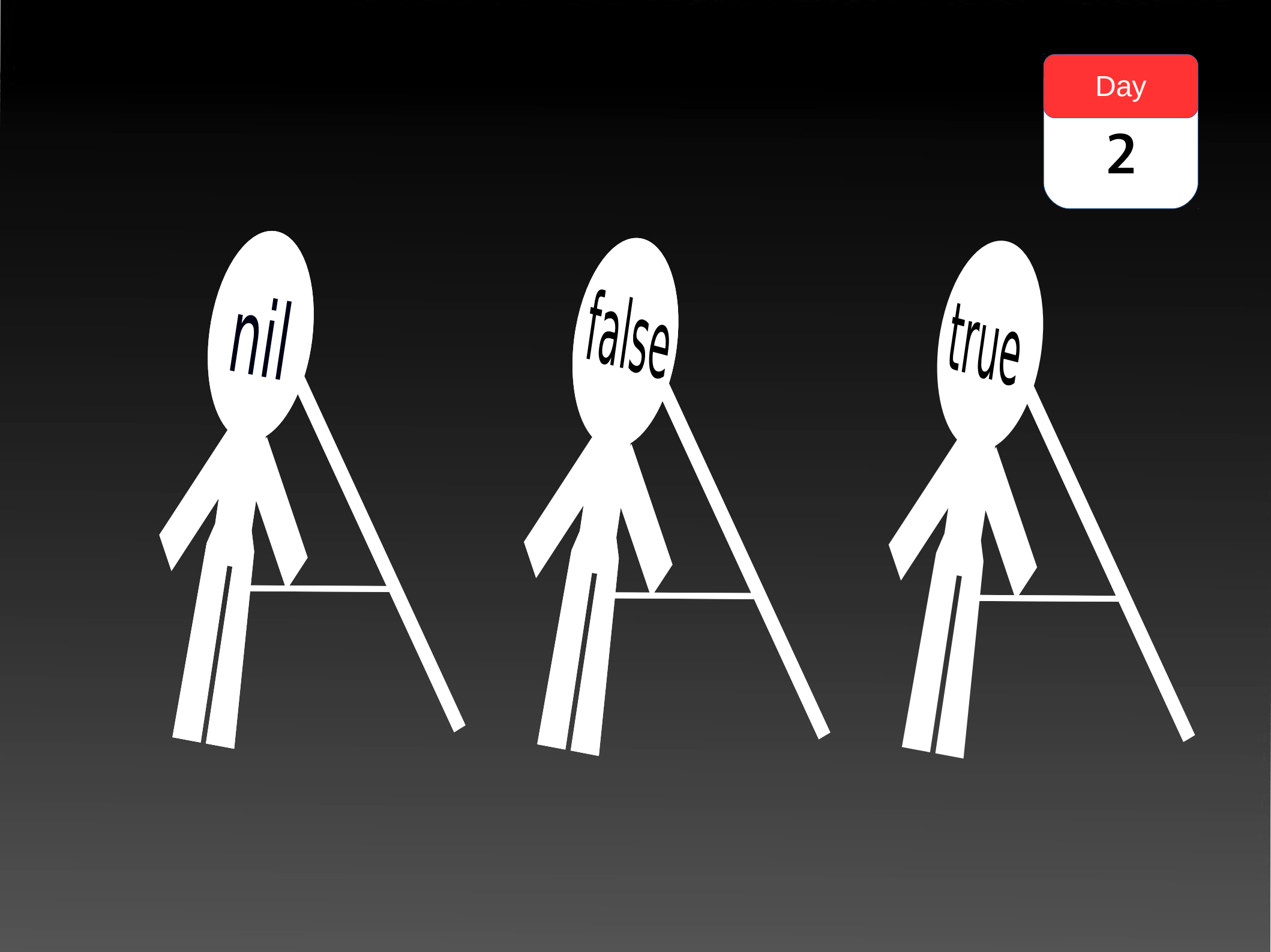
The bootstrap process

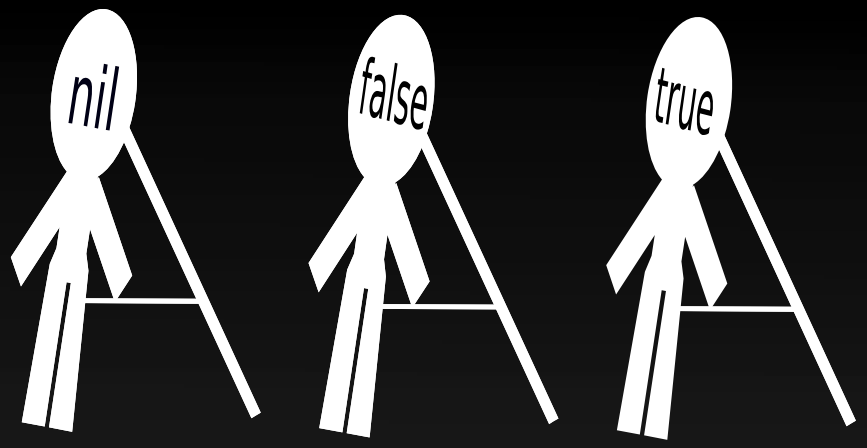
nothing


nil

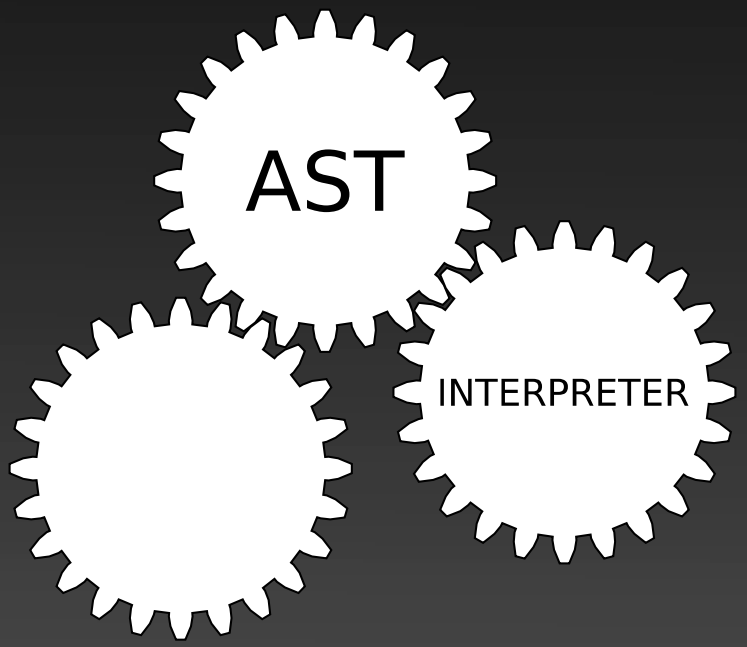

Day
1

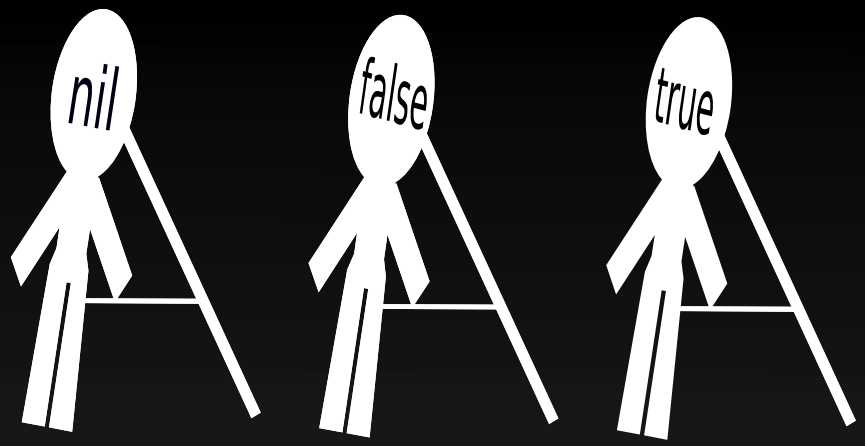

nil


Day
2
nil
false
true



classes

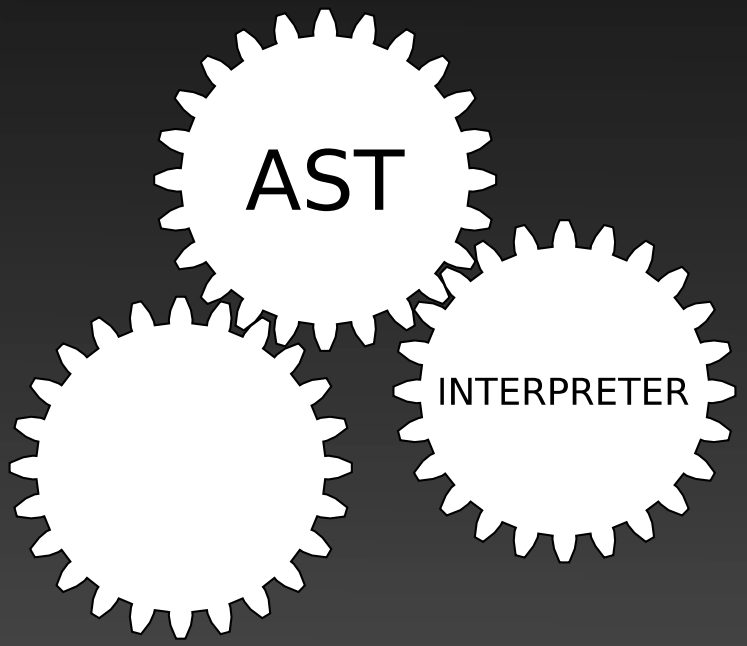


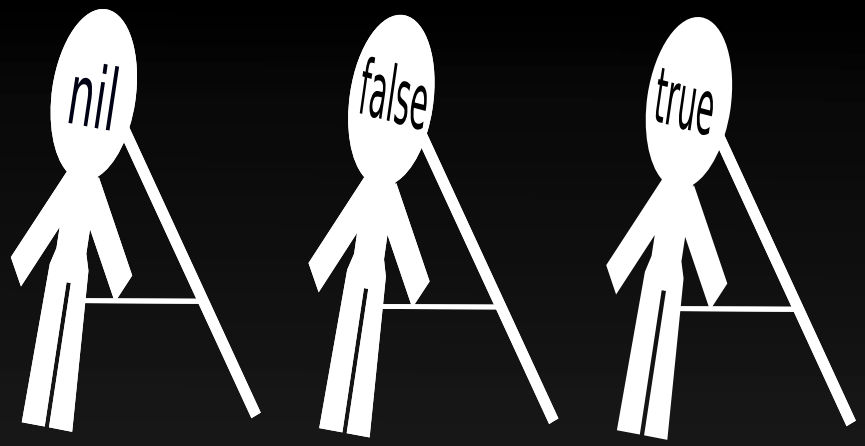


Day
4

classes

methods





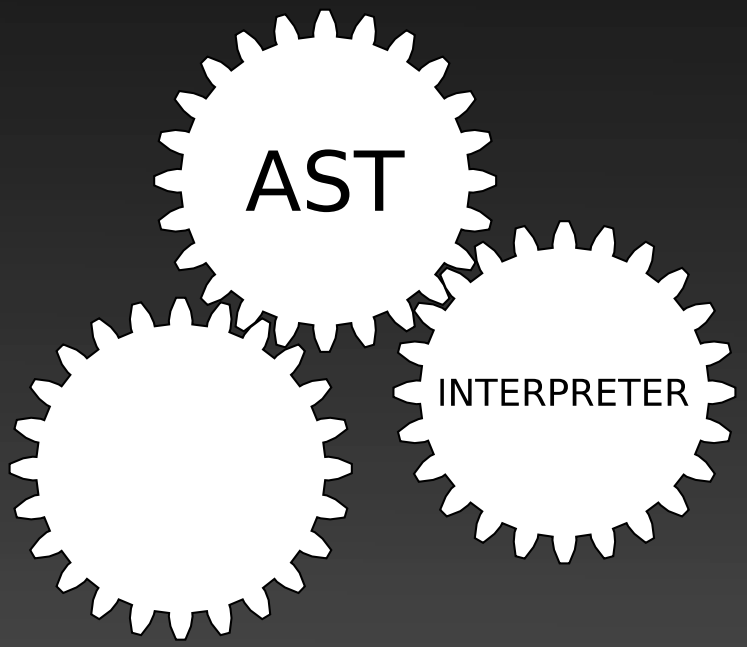
Day

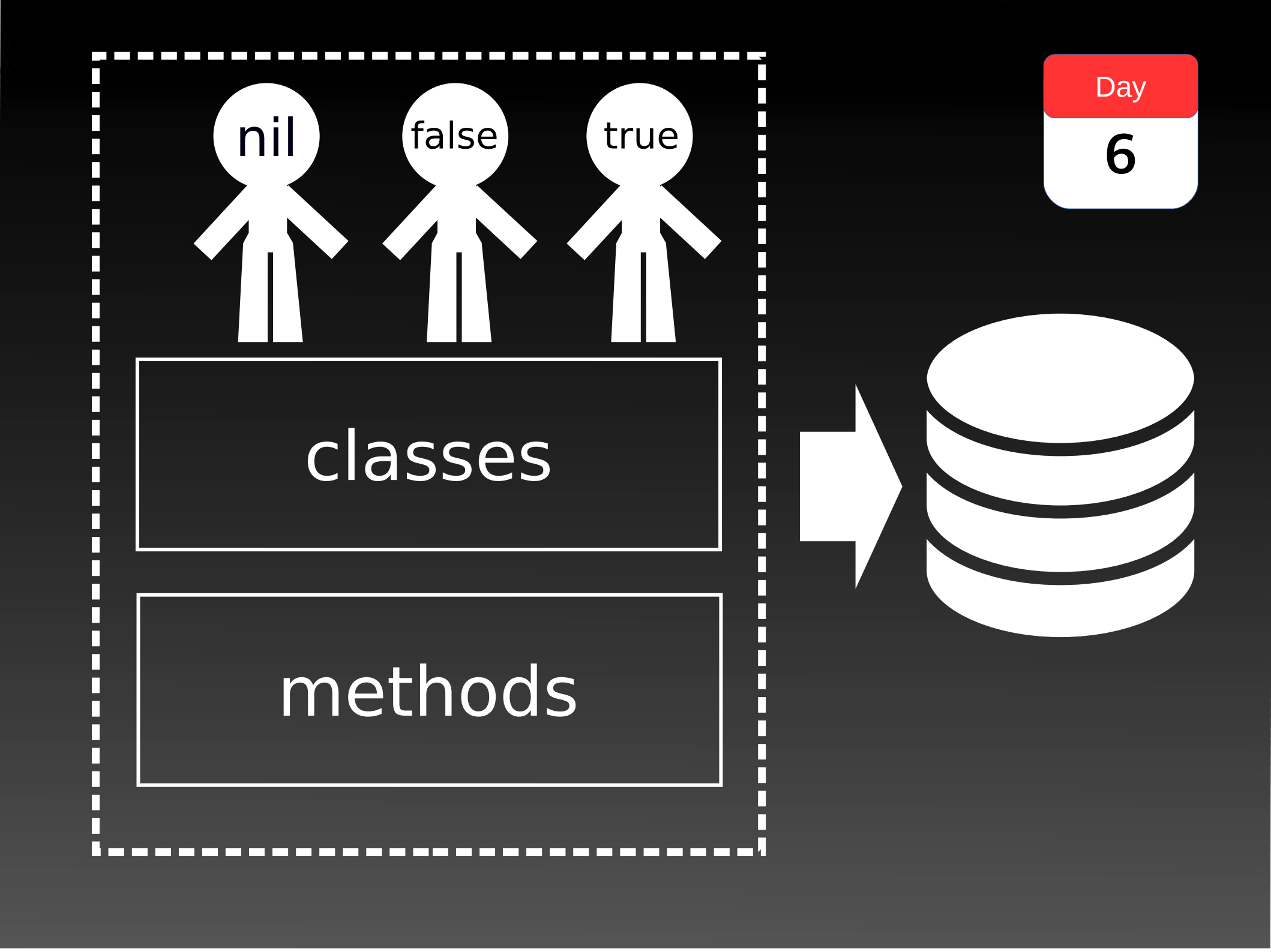
5

classes

methods

process




nil
false
true
classes
methods
Day
6

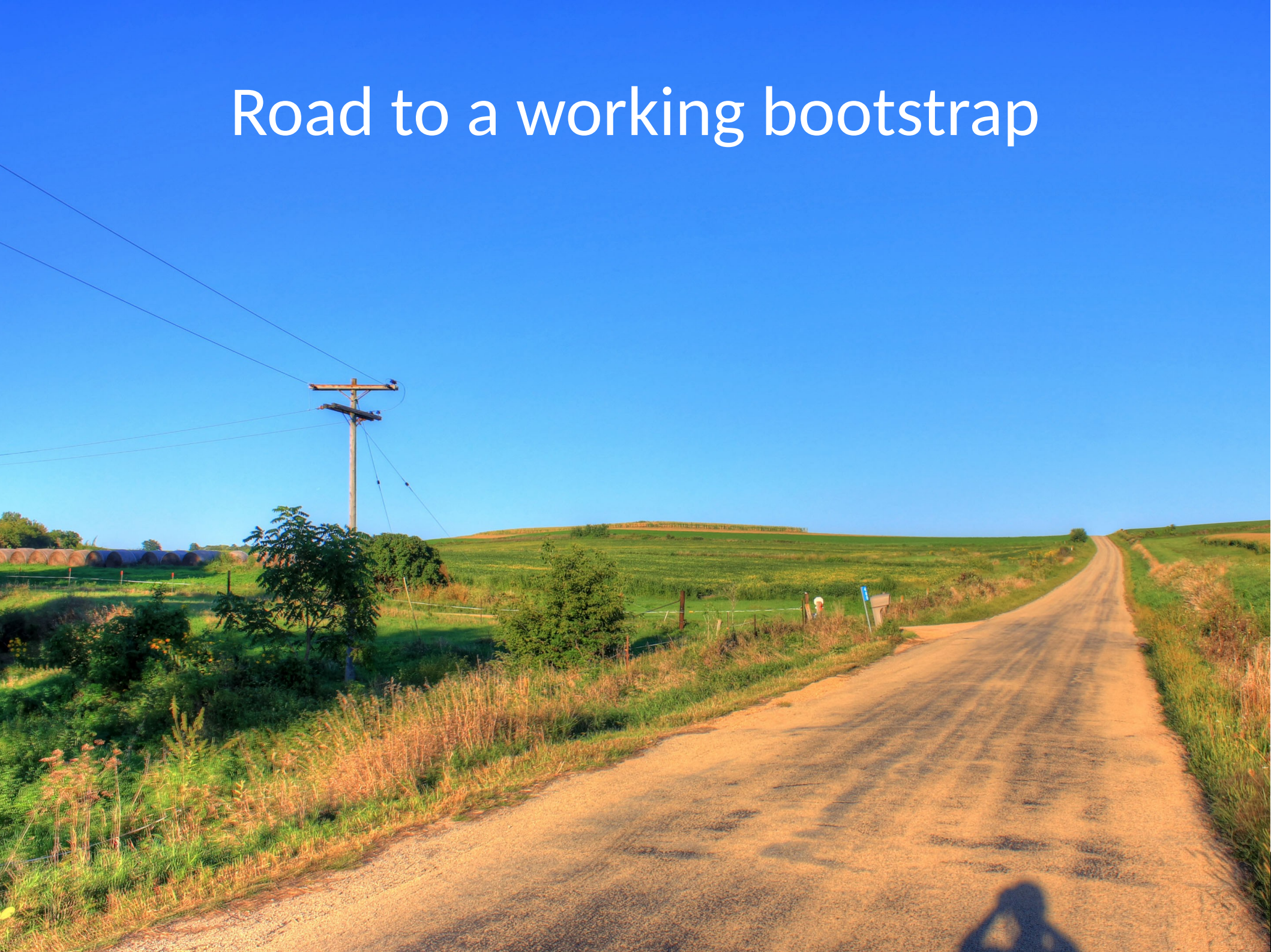

Day
7

More details

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

Story #2

Road to a working bootstrap



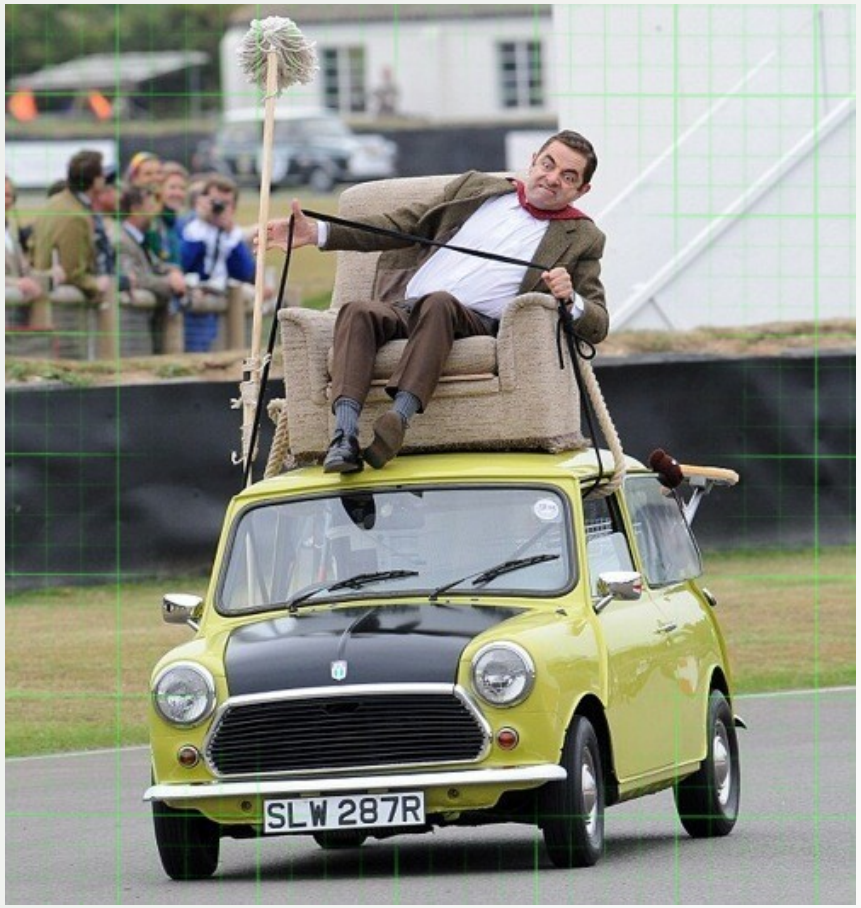
Bootstrap challenge
> language side bootstrap

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 boostrap
    e.g. Float

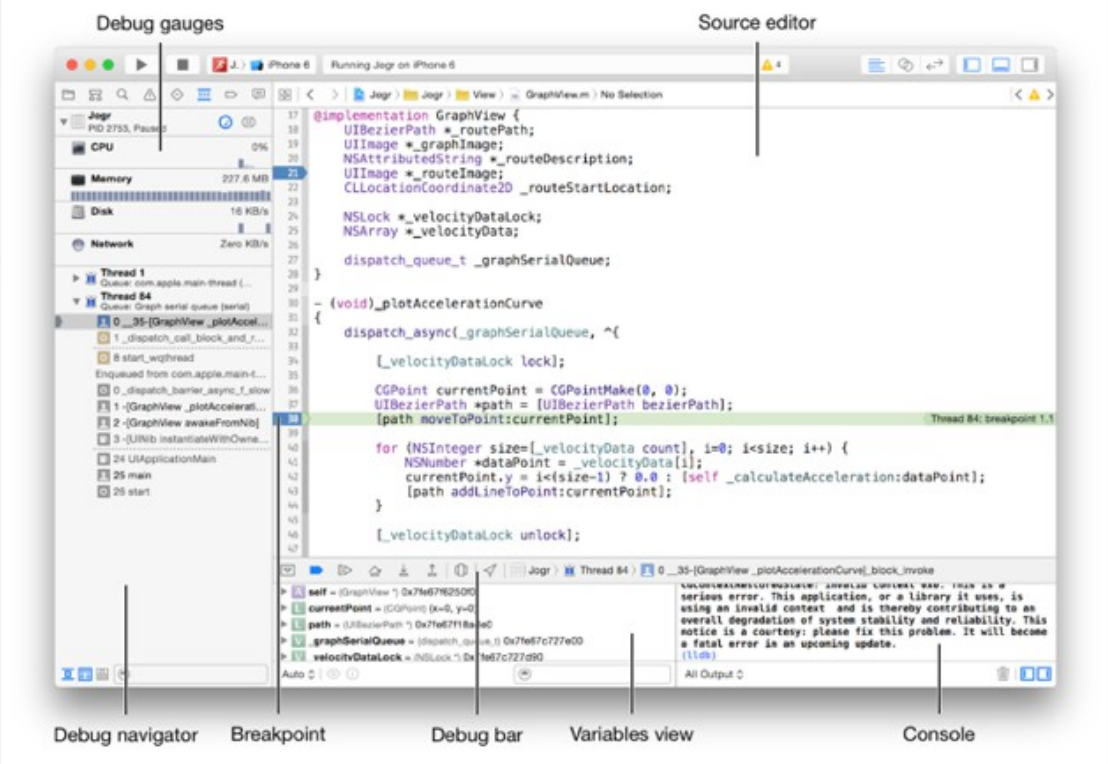
Super class not set

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 boostrap 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