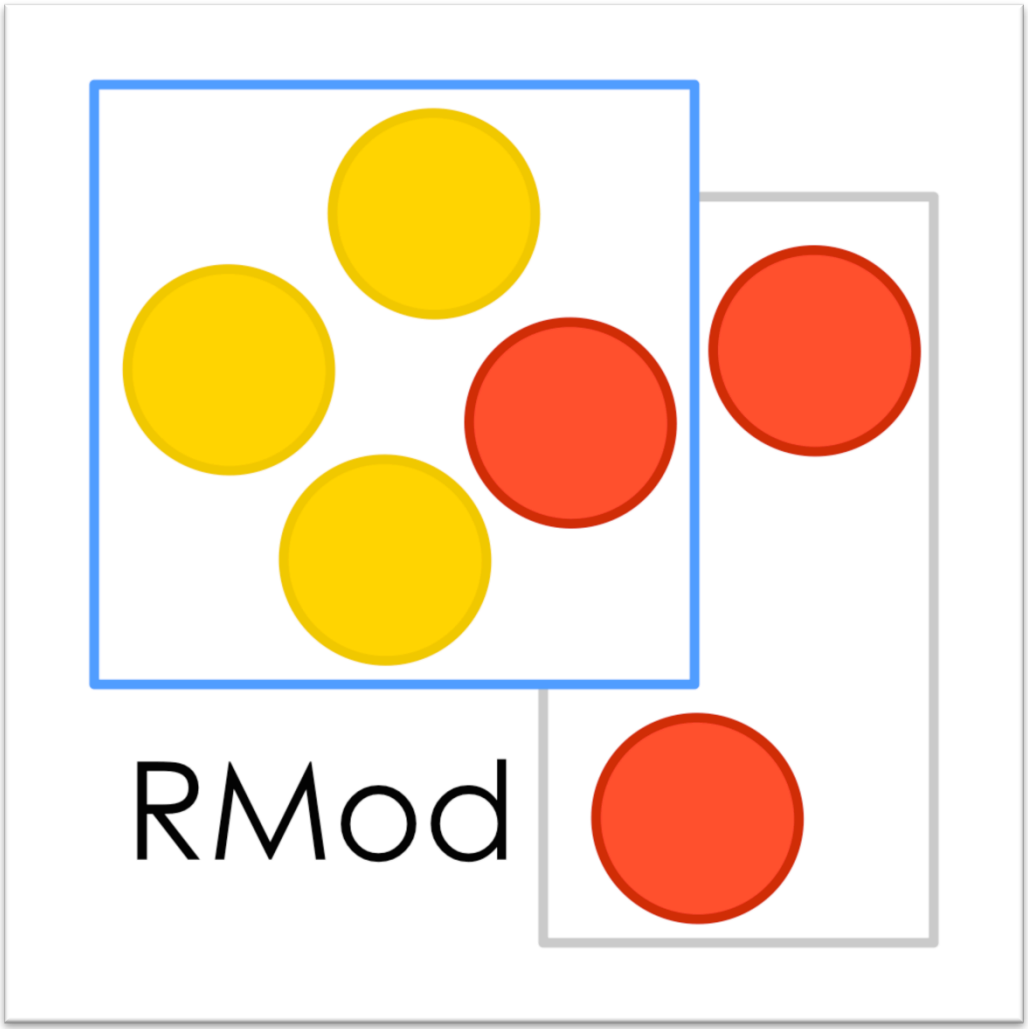
Modularity from the trenches

Stéphane Ducasse and Guillermo Polito

Modularity 2017
Brussels


RMod

A context: Large Systems

- Several thousands of classes
- Multiple Millions of LOC
- Long living systems ~ 15 to 25 years
- Successful systems

Software Entropy

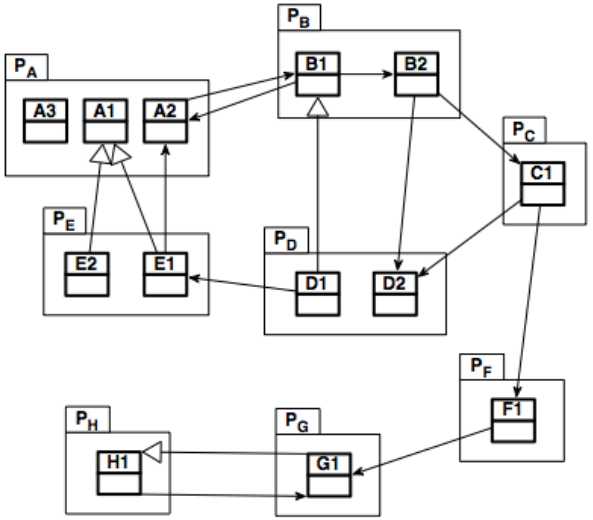
- You have success
- Your software grows
- Complexity too


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

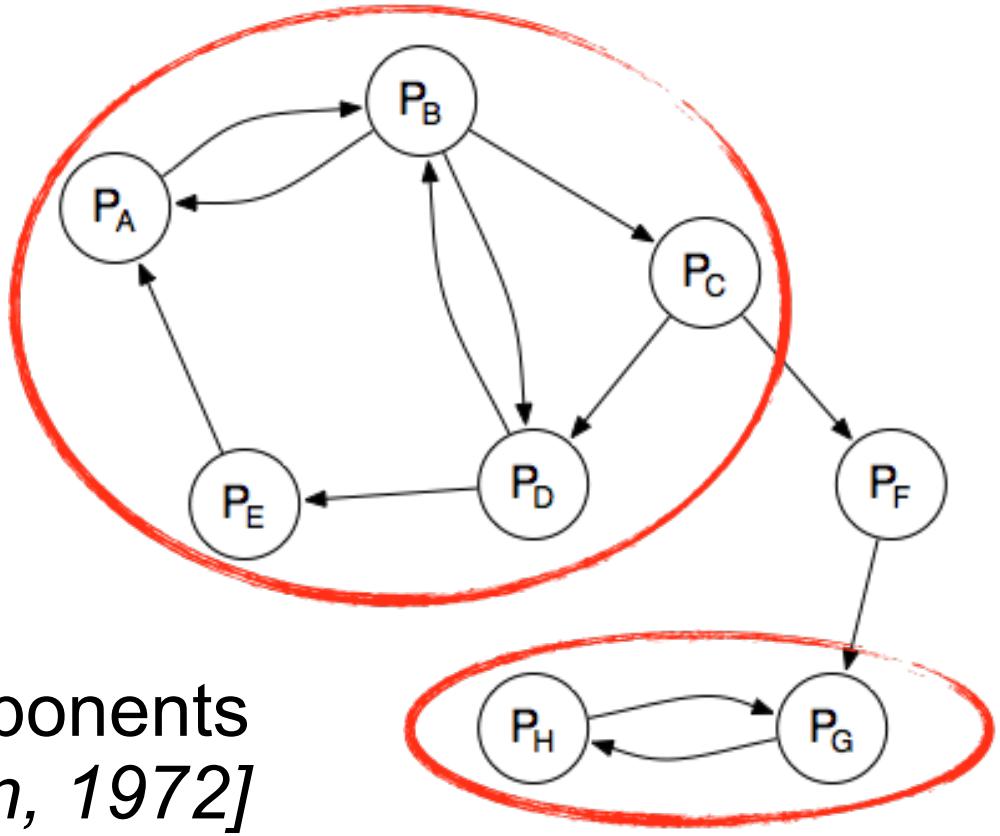
Roadmap

- **Analyses to help modularizing software**
  - PhD of H. Abdeen: using simulating annealing to repackage
  - PhD of J. Laval: characterising cycles and layers
- The story of the Pharo Kernel mission
- Removing vs. Bootstrap
  - PhD of G. Polito
- Towards go Module system

Identifying cyclic dependencies?



i. Graph abstraction



ii. Strongly Connected Components (SCC) detection *[Tarjan, 1972]*

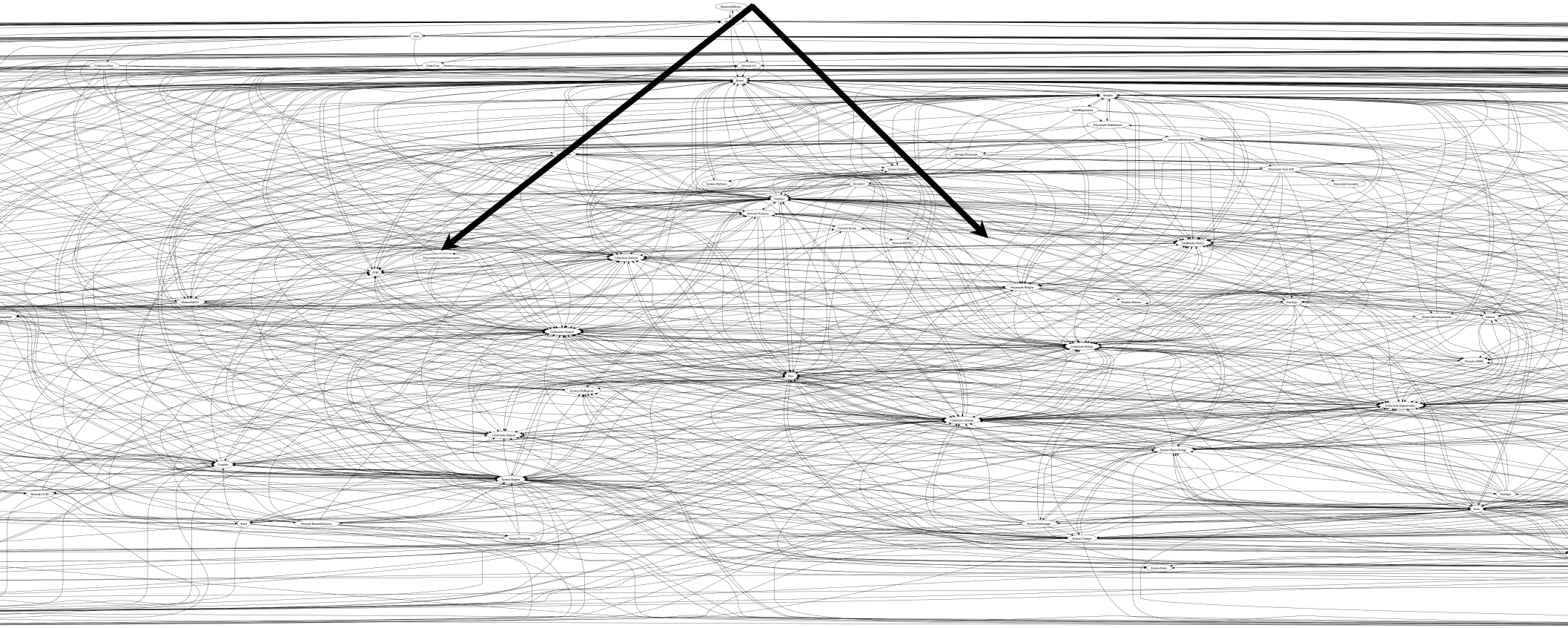
A large software application

Pharo Core 1.1, 115 packages

- One big SCC ☺

What are the differences between these dependencies ?

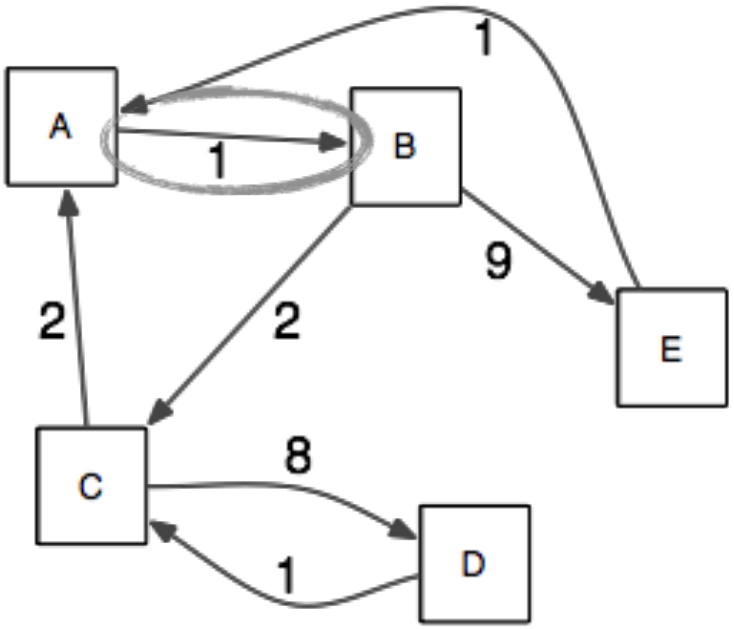
Packages in SCC



Pharo Core 1.1, 78 packages in cycle

- Cycles are not visible
- No information about dependencies

Building a DSM



weight = Inheritance + invocation + reference + extension

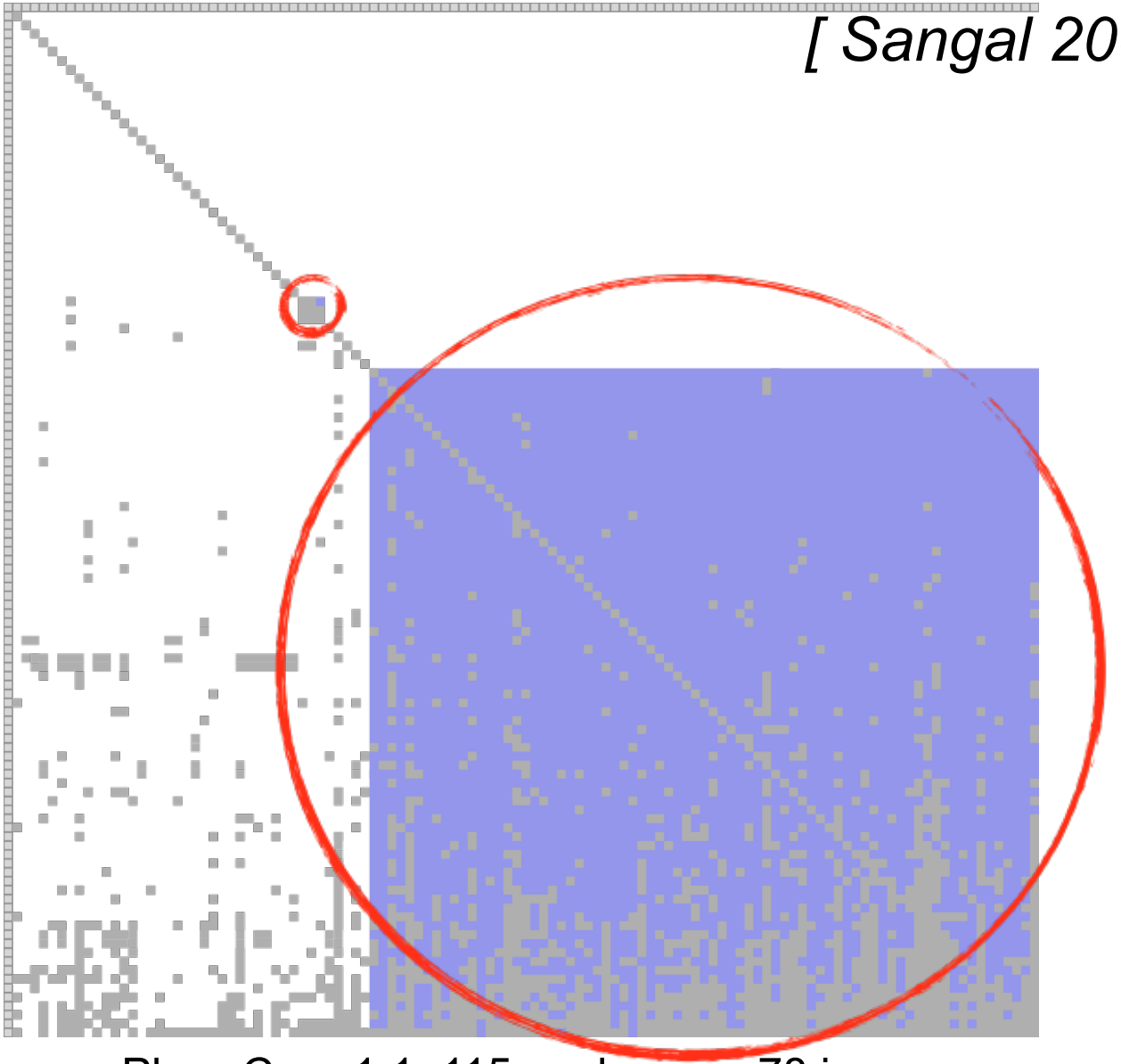
provides

depends

	A	B	C	D	E
A			X		X
B	X				
C		X		X	
D			X		
E		X			

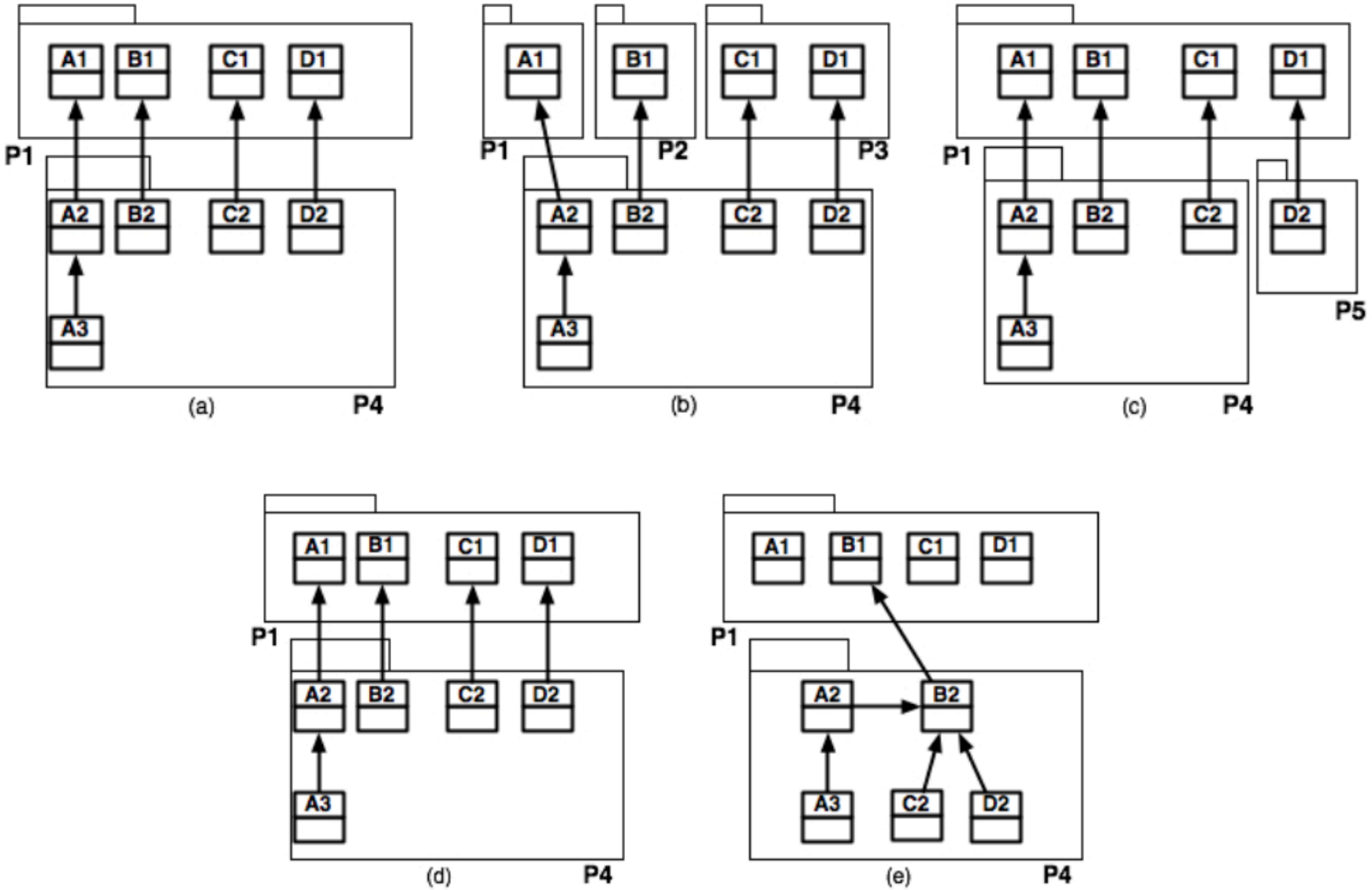
DSM for software architecture

[Sangal 2005]



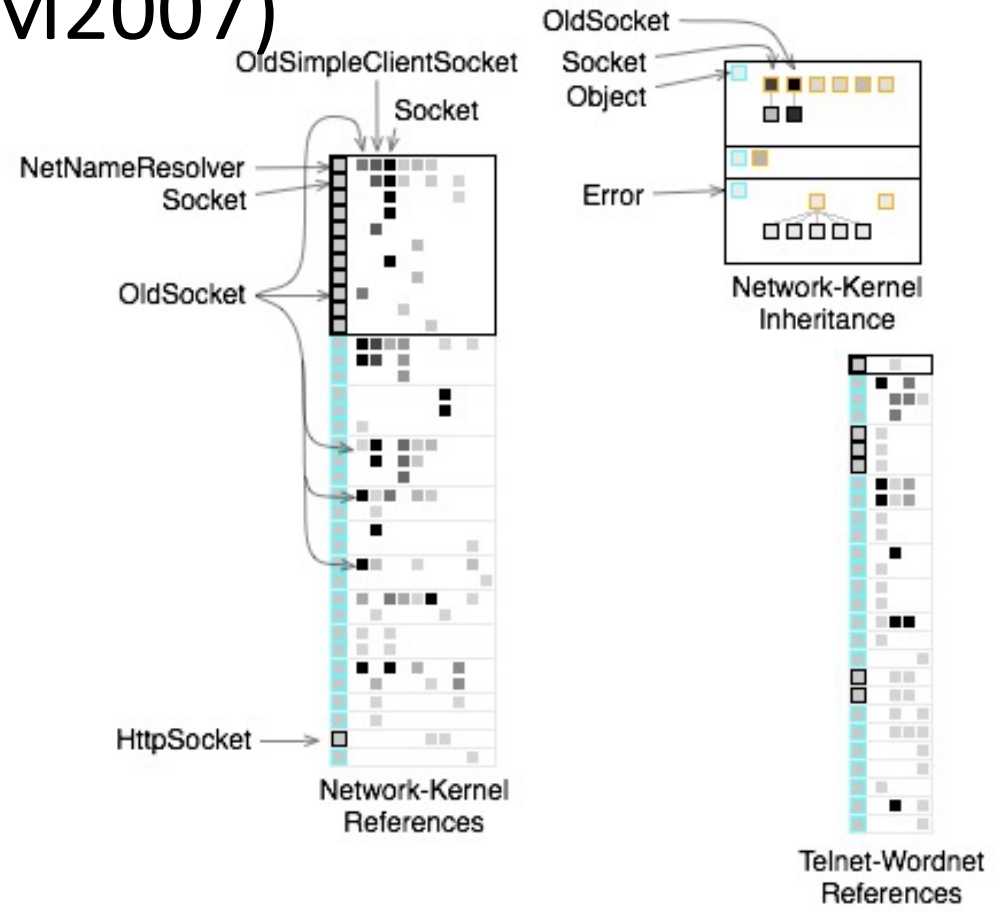
PharoCore 1.1, 115 packages - 78 in cycle

There is a need to understand package dependencies


A1
B1
C1
D1
P1
A2
B2
C2
D2
A3
(a)
P4
P2
P3
(b)
P5
(c)
(d)
(e)

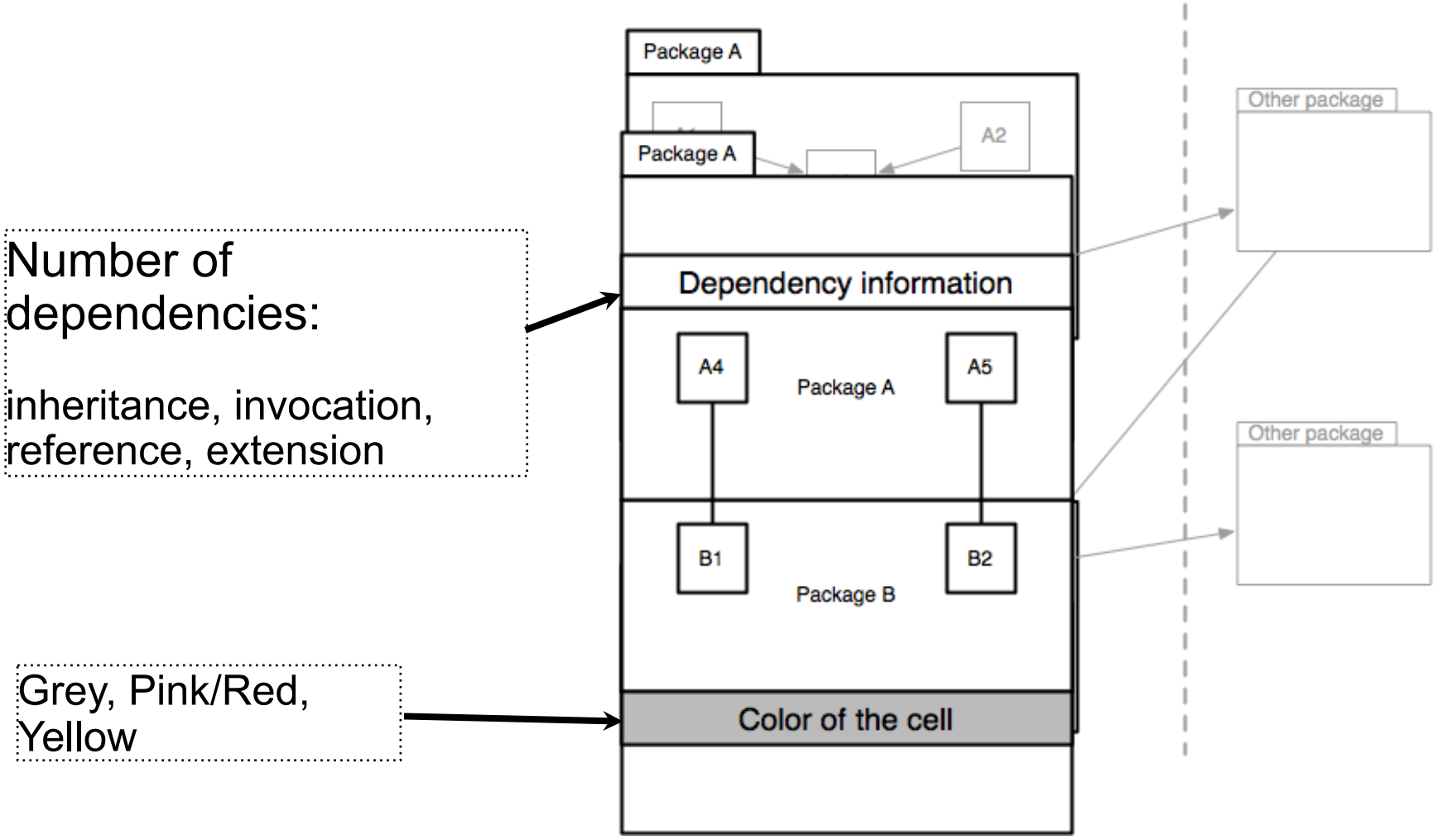
Some tools

- Package Blueprints (ICSM2007)
- Extended DSM



A cell expresses a dependency

- From Package A to Package B

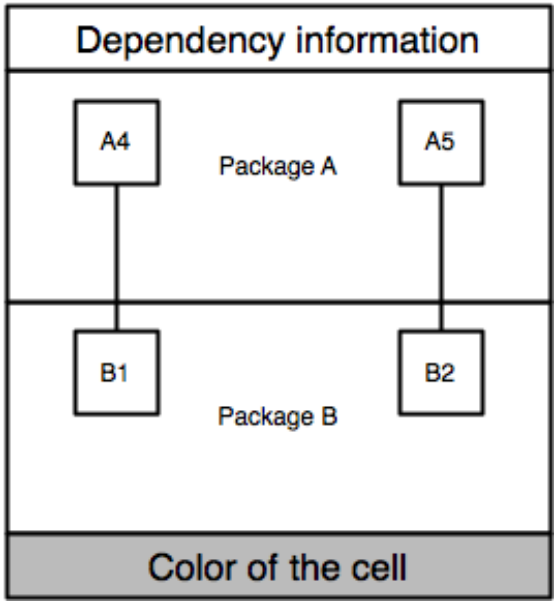


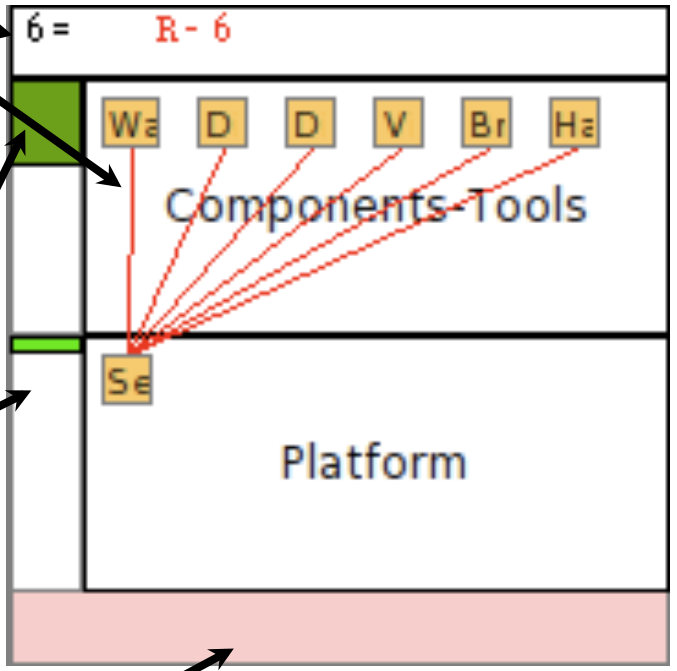
eCell example

- From Components-Tools to Platform

6 dependencies:

6 references (6 classes to 1 class)





rate of impacted classes

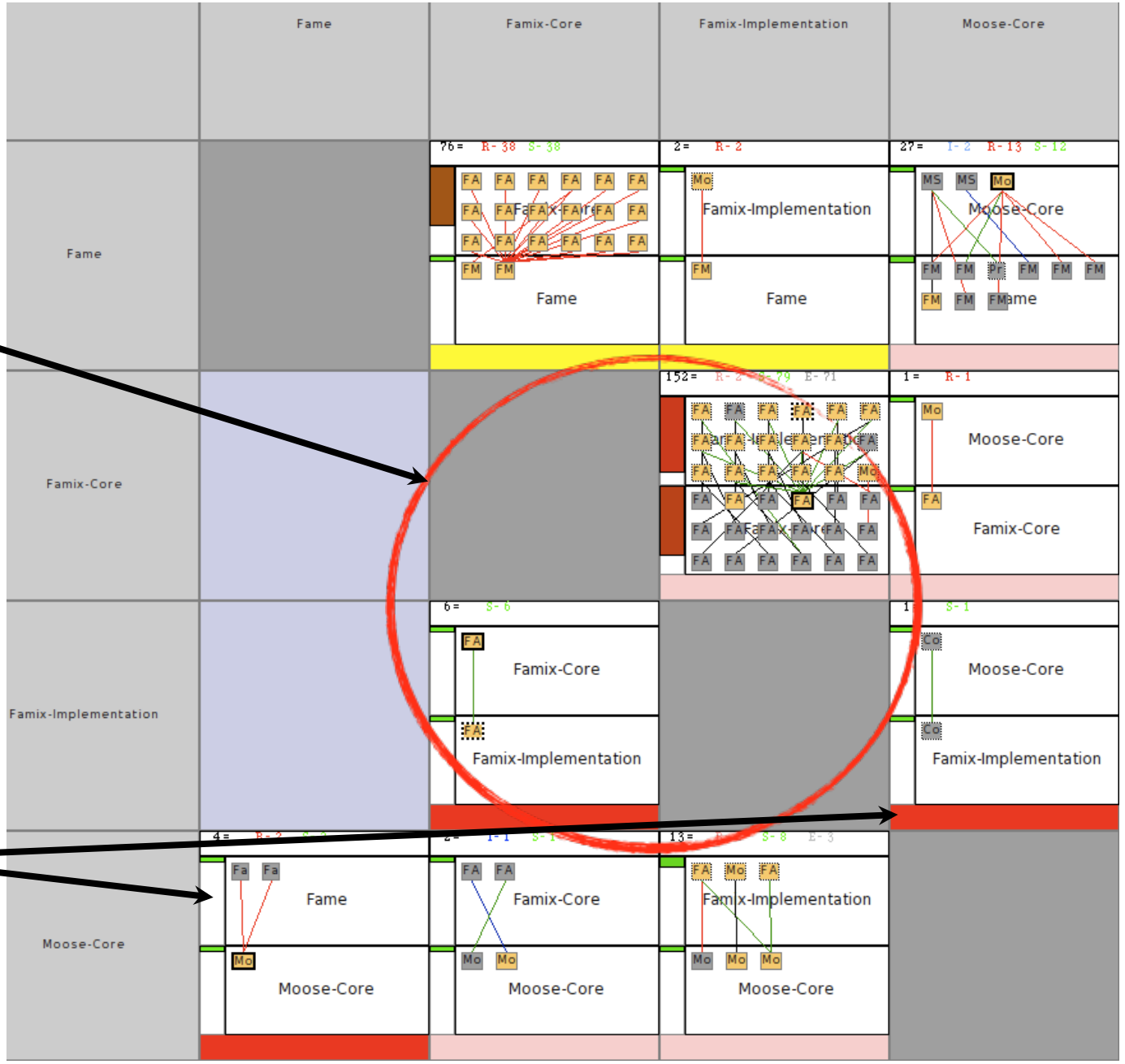
In a direct cycle

eDSM with eCell: Zoom on a SCC

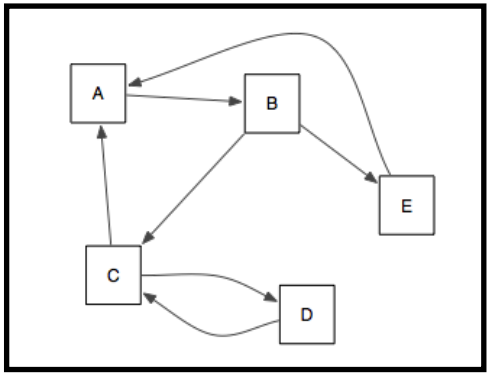
A direct cycle:

The red dependency seems better to remove

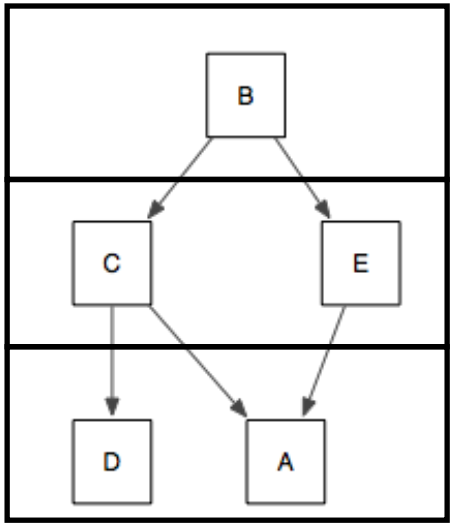
2 other candidate dependencies



Existing approaches to layer identification



- Lattix move each SCC in a layer
  - *[Lattix 2005]*
  - Does not work with undesired cycles
- Minimum Feedback Arc Set (MFAS) break the minimum of edges.
  - *[Melton 2007]*
  - *Does not take into account the semantics*
- Regression and Integration testing technics
  - *[Le Traon 2000, Da Veiga Cabral 2010]*
  - *=> Minimization of vertices vs minimization of arcs*



oZone strategy to identify layers

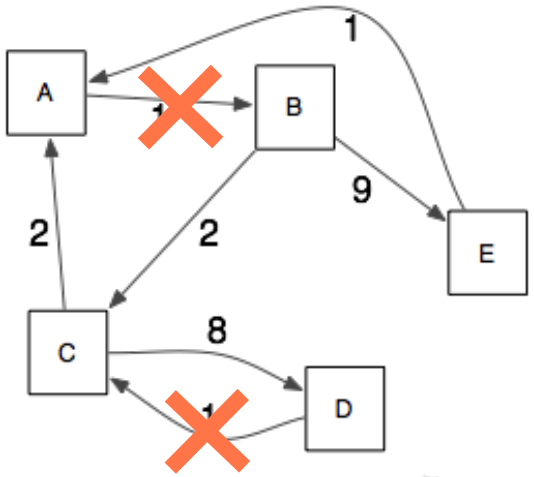
- Selecting edges that can break direct cycles.
- Selecting edges that are shared dependencies.
- User interaction to improve the results

Results are:

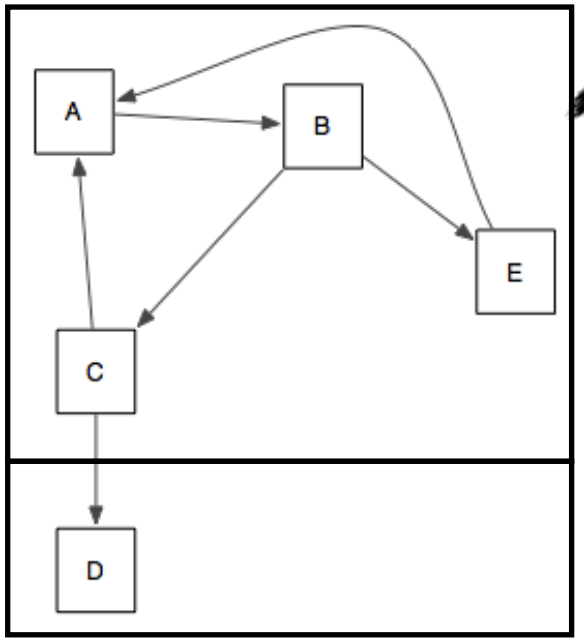
A list of dependencies that are interesting to remove.

A layered structure of the application to understand it.

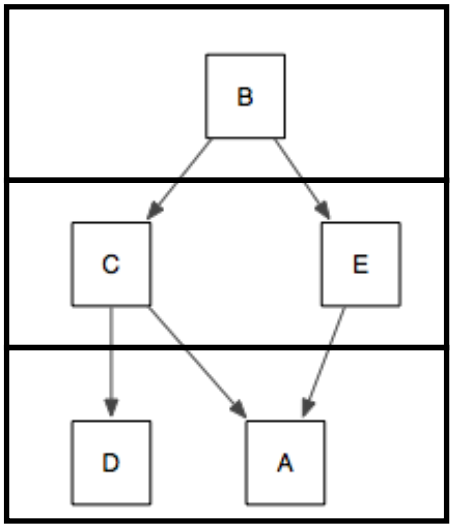
oZone: example



remove direct cycles
by ignoring a
dependency



remove other
cycles
by ignoring shared
dependencies



Roadmap

- Analyses to help modularizing software
  - PhD of H. Abdeen: using simulating annealing
  - PhD of J. Laval: characterising cycles and layers
- **The story of the Pharo Kernel mission**
- Removing vs. Bootstrap
  - PhD of G. Polito
- Towards go Module system

Pharo as a real experimental setup

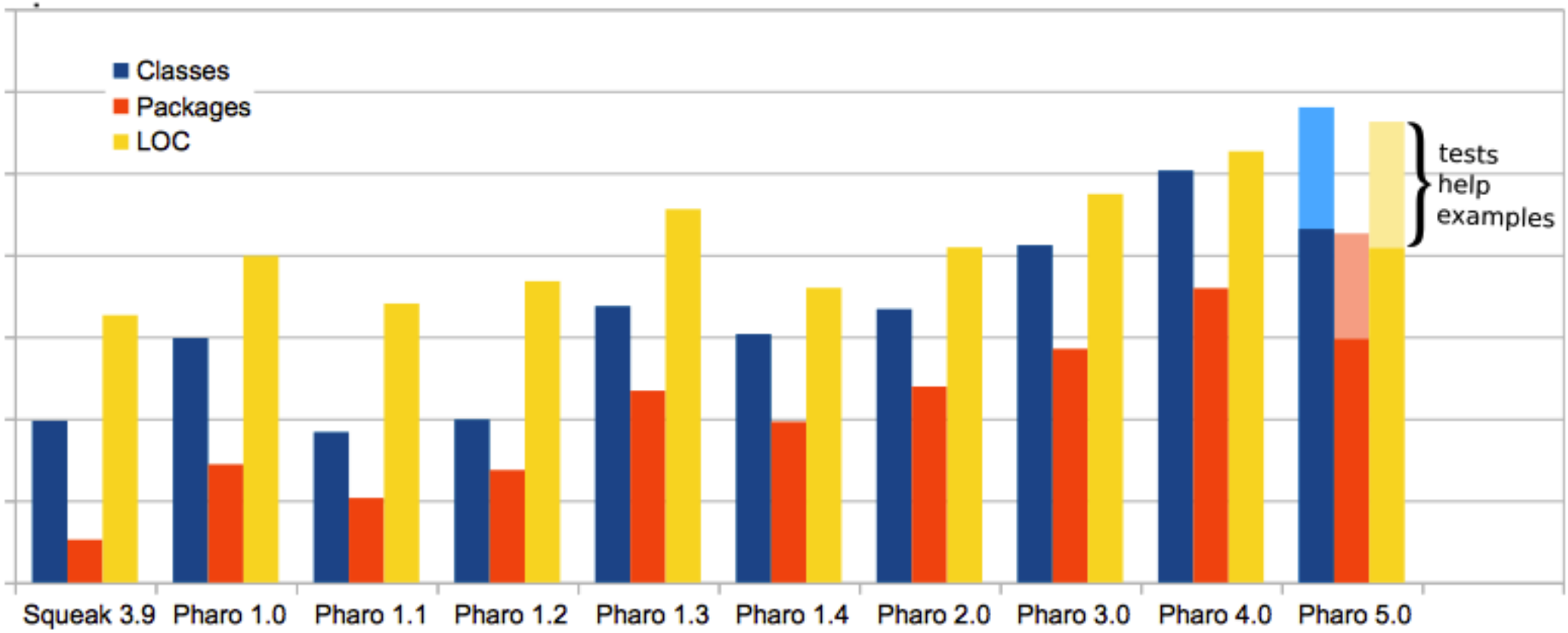
Pharo this is

- 465 packages
- more than 3000 thousands external projects
- 98 committers
- 30 universities teaching with Pharo
- 16 worldwide research groups

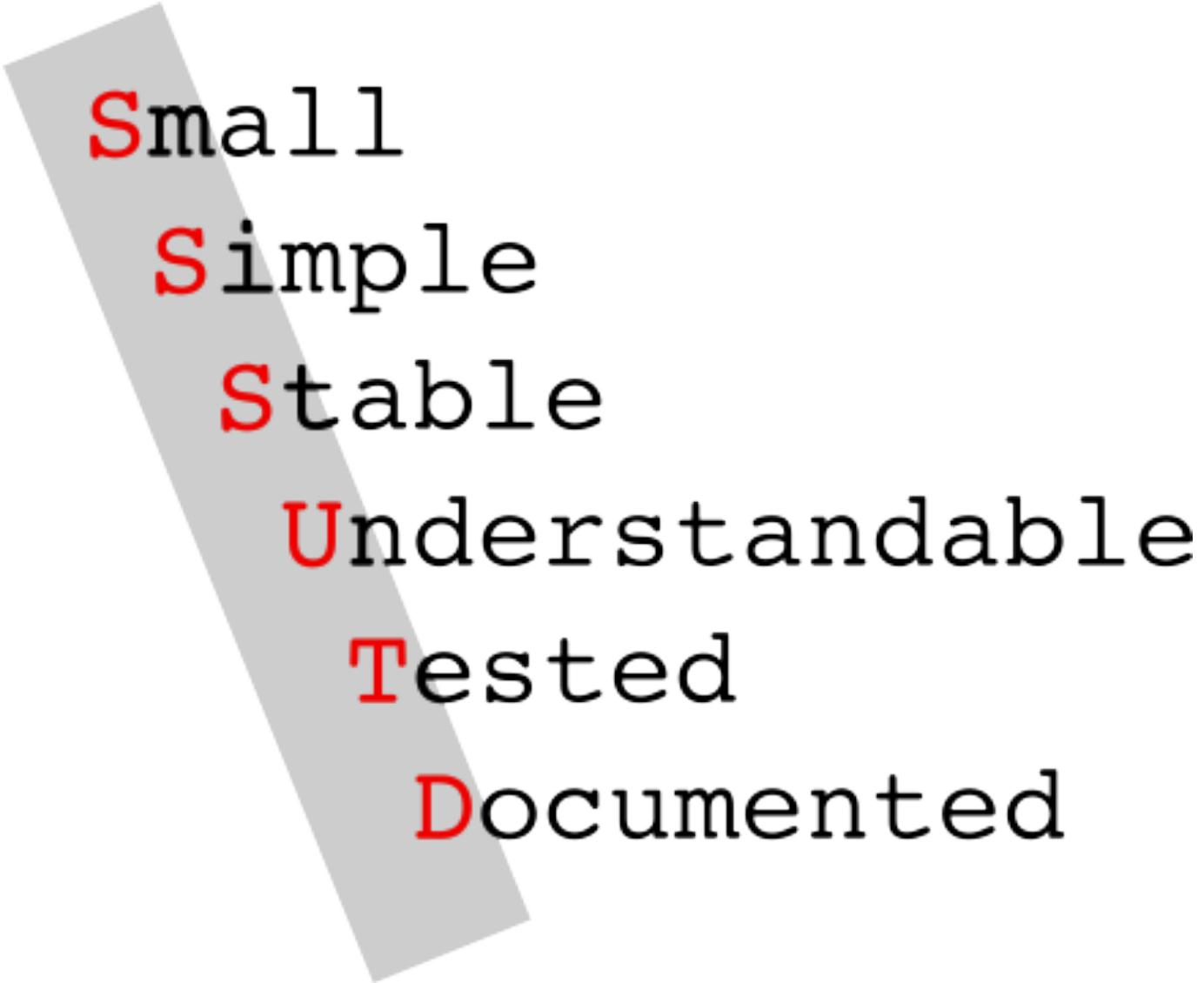
Pharo is growing

New libraries
New tools
New tests
More documentation

Pharo evolution



Pharo Kernel Mission



From top

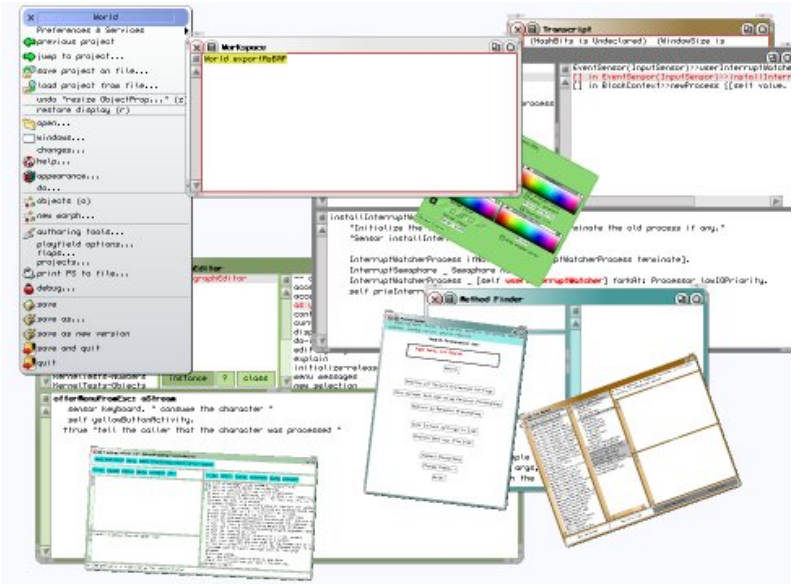
- cleaning
- shrinking
- reloading

From bottom

- bootstrapping
- 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
by Pavel Krivanel [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?
Everyone must
take of care of
modularity
Should be 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

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

Roadmap

- Analyses to help modularizing software
  - PhD of H. Abdeen: using simulating annealing
  - PhD of J. Laval: characterising cycles and layers
- The story of the Pharo Kernel mission
- **Removing vs. Bootstrap**
  - PhD of G. Polito
- Towards go Module system

Let's talk about BOOTSTRAP

Bootstrap

« The process that builds the minimal infrastructure of a langage 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 bootstraping is difficult?

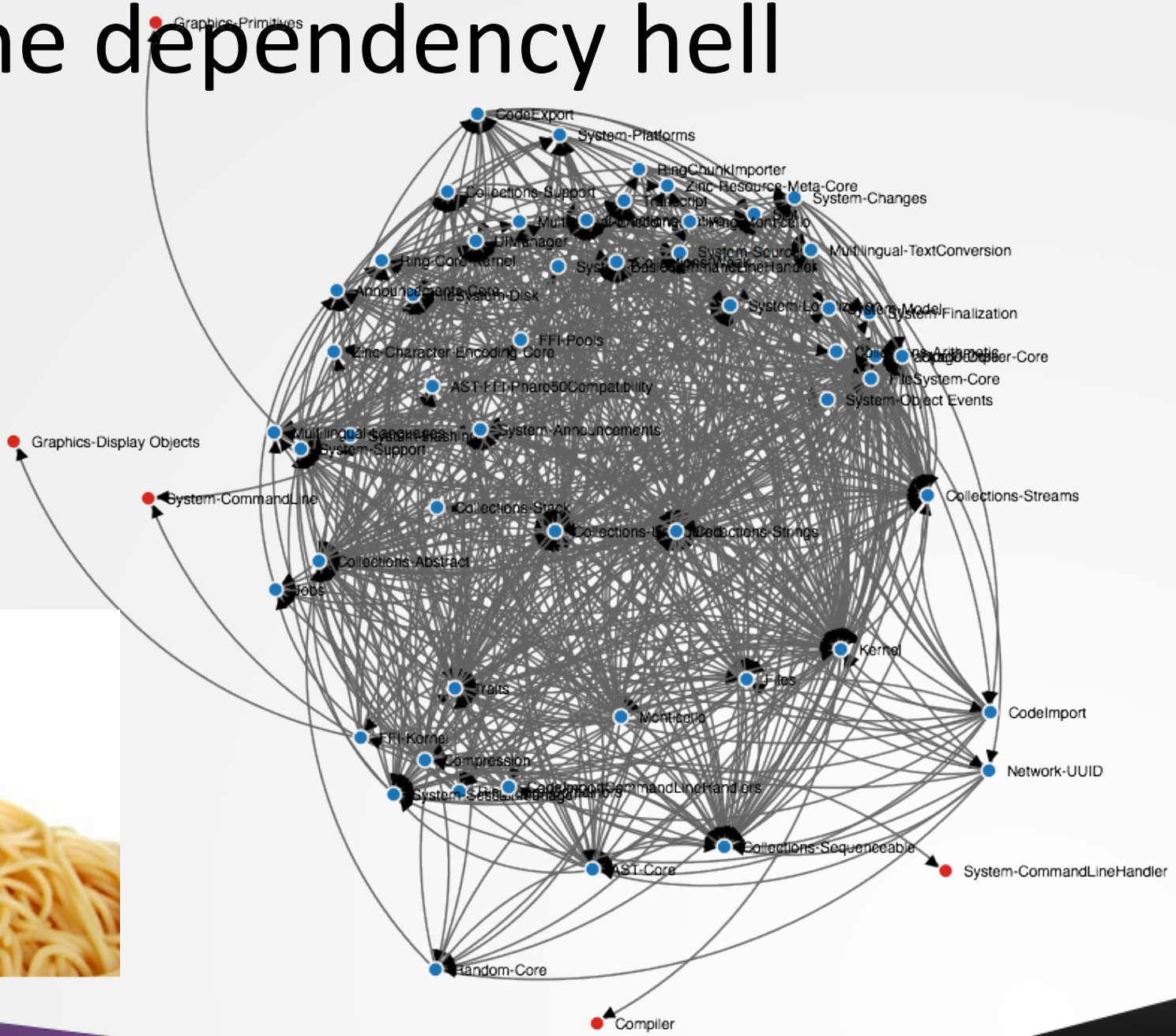

Archaelogy

Deadcode

Strange logic

The missing initialization

The dependency hell



KEEP

CALM

AND

REFACTOR

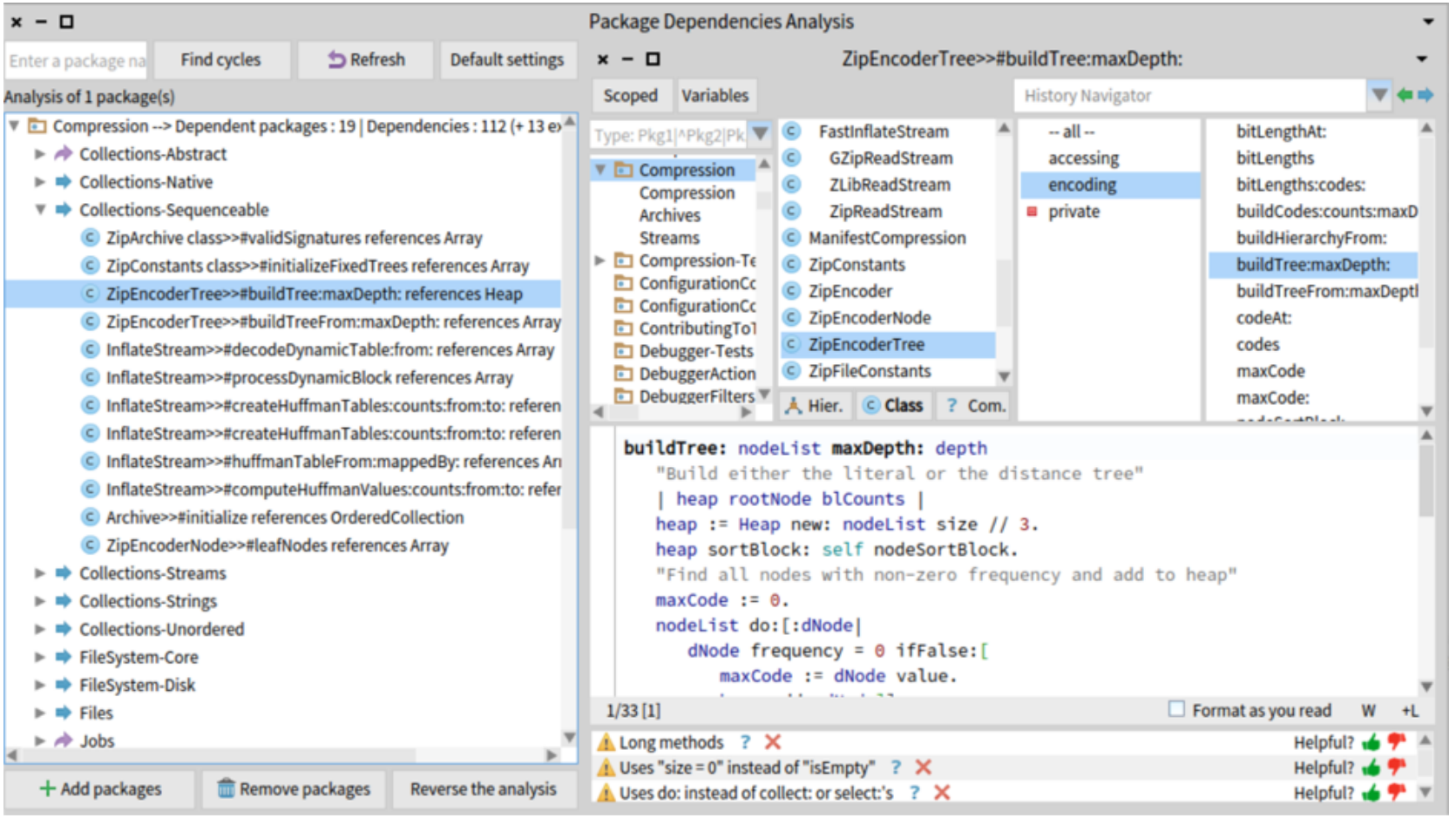
CODE

How to fix bad dependencies?

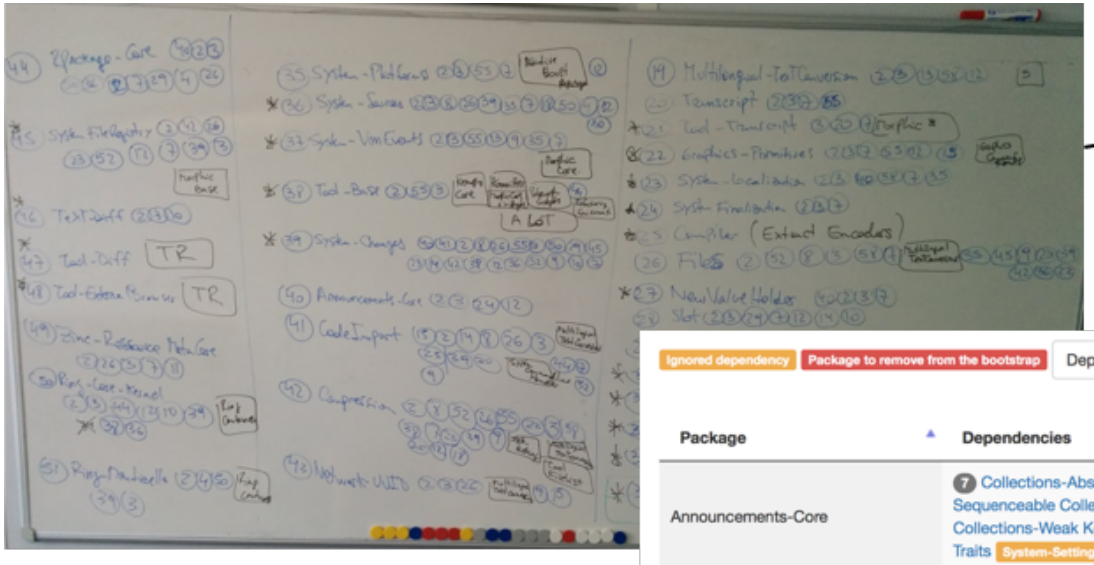
- Create a new package to isolate functionnalities
- Move methods as extensions to another package
- Kill facades (there are global thinking)
- Components made to be customized
  - Settings
- Registration mechanism
- Re-design completely a functionality
  - e.g. startup list
- ...

Tools support

Dependencies analyser



Dependency Dashboard



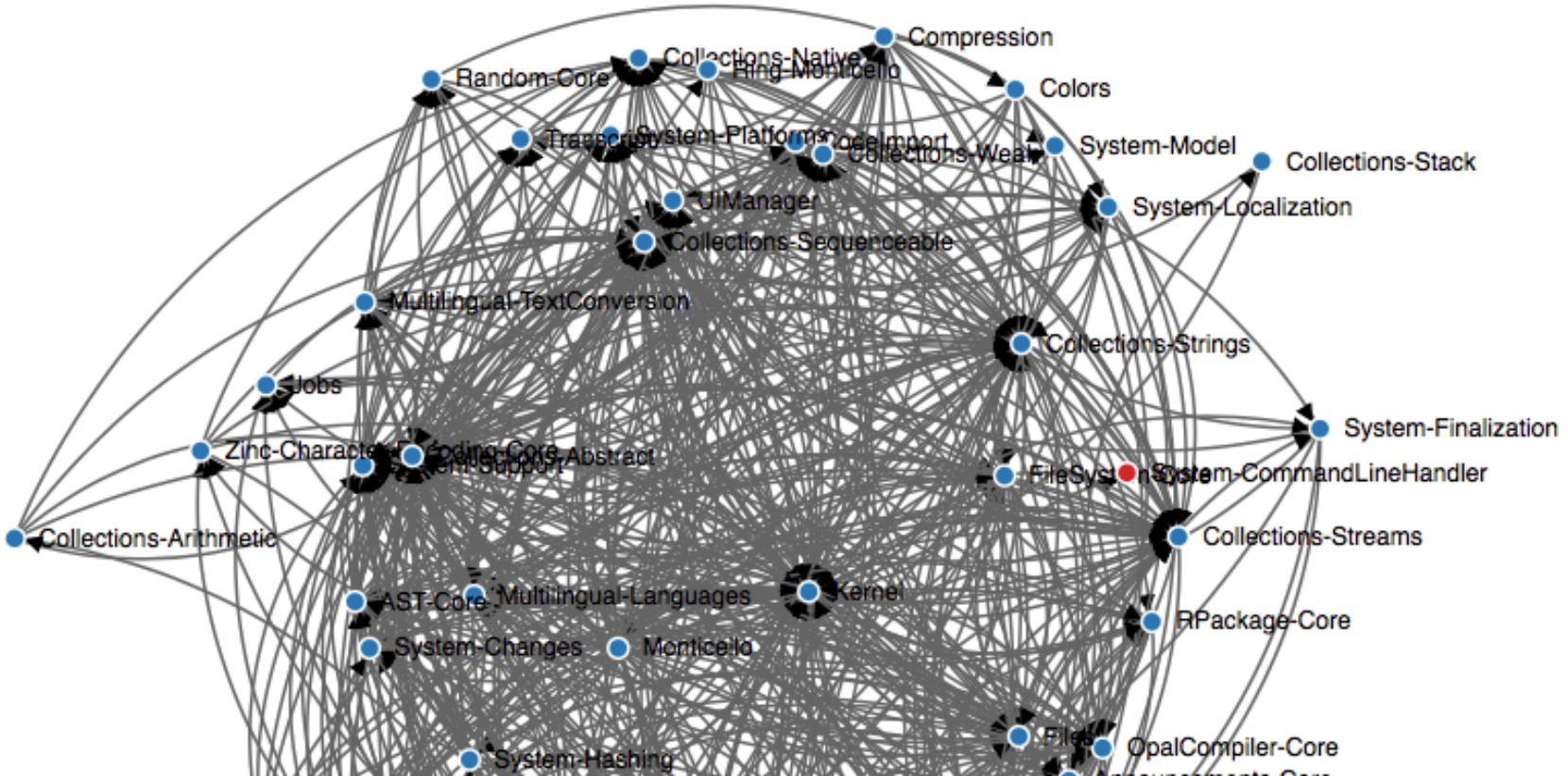
Ignored dependency | Package to remove from the bootstrap | Dependency graph

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-

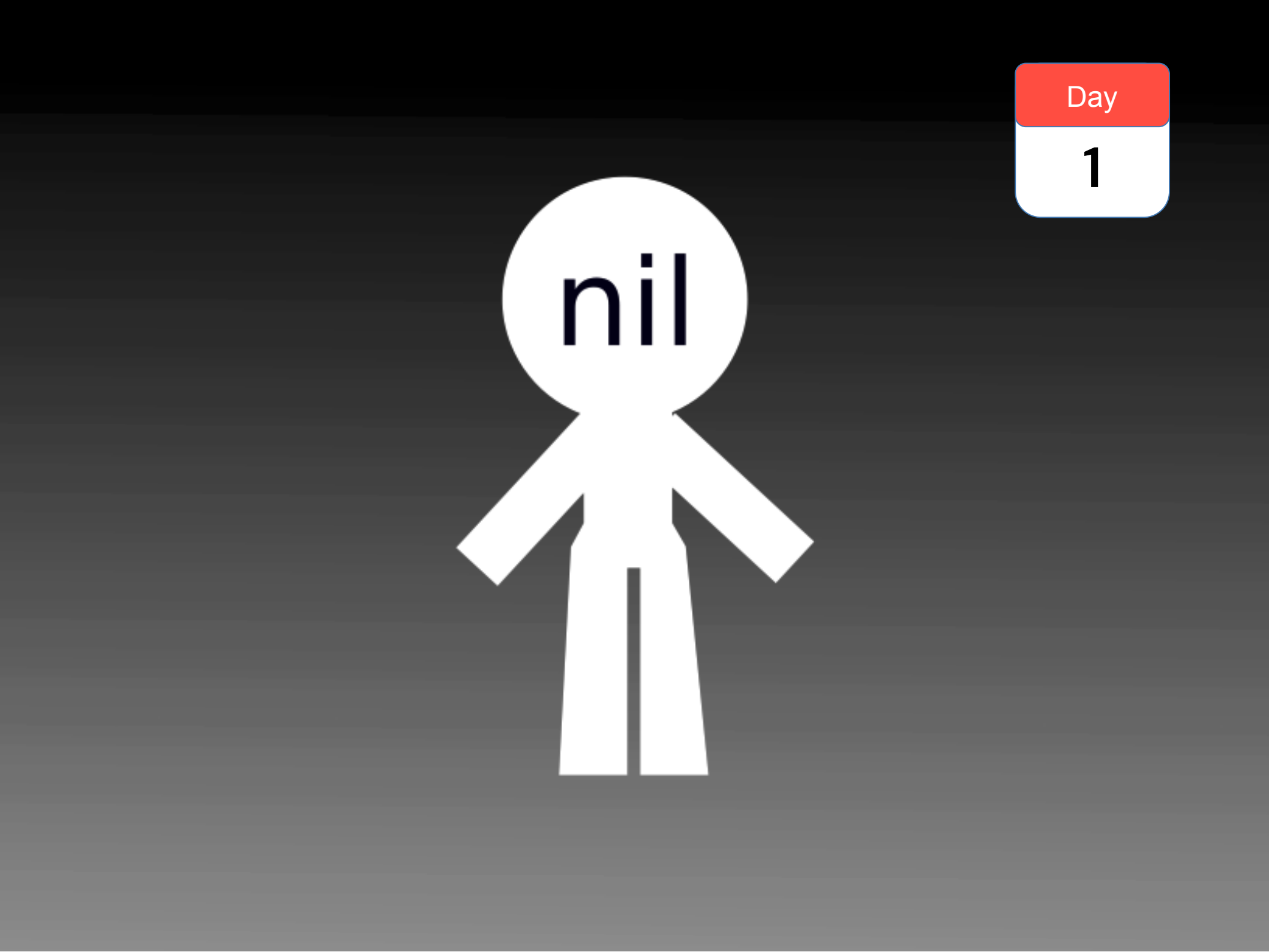


The bootstrap process

Bootstrap process insight

1) creation of stub objects : nil, false, true
2) definition of classes and metaclasses
3) compilation of methods
4) creation of the initial process
5) system serialization

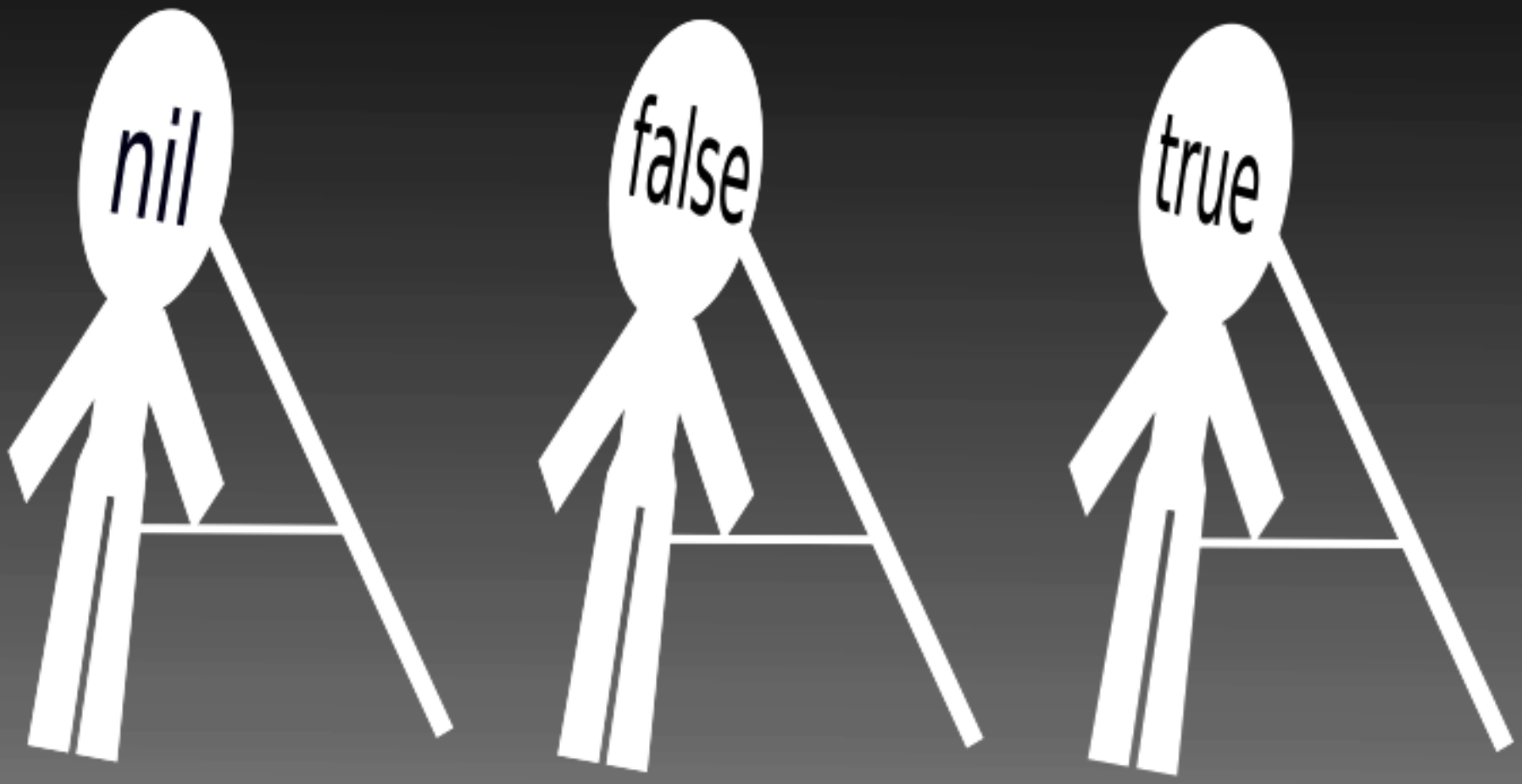
nothing

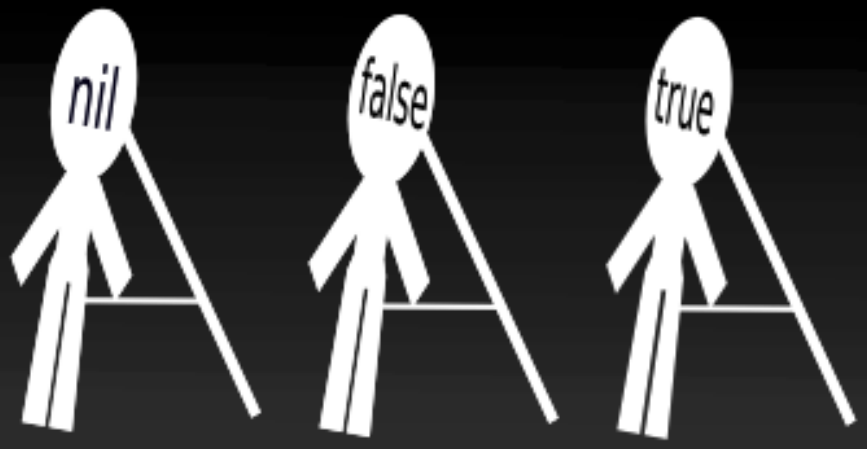

Day
1
nil


nil

Day

2

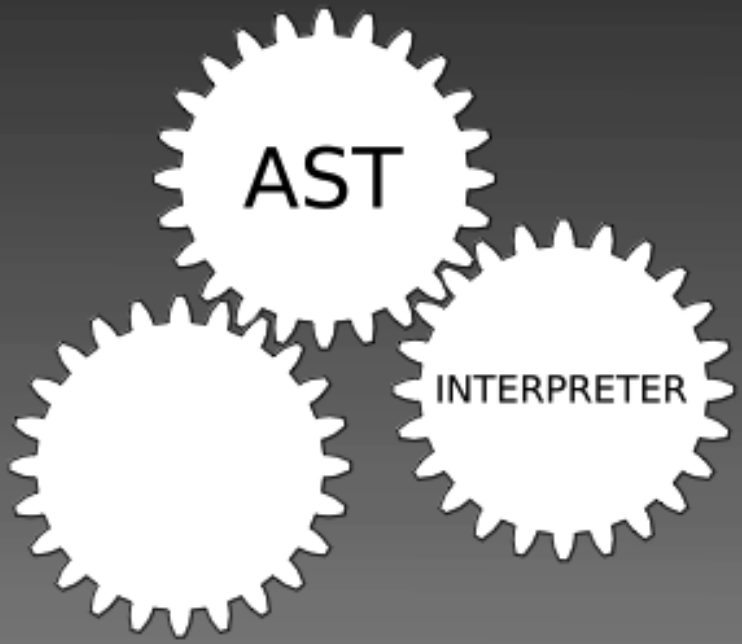




classes

Day

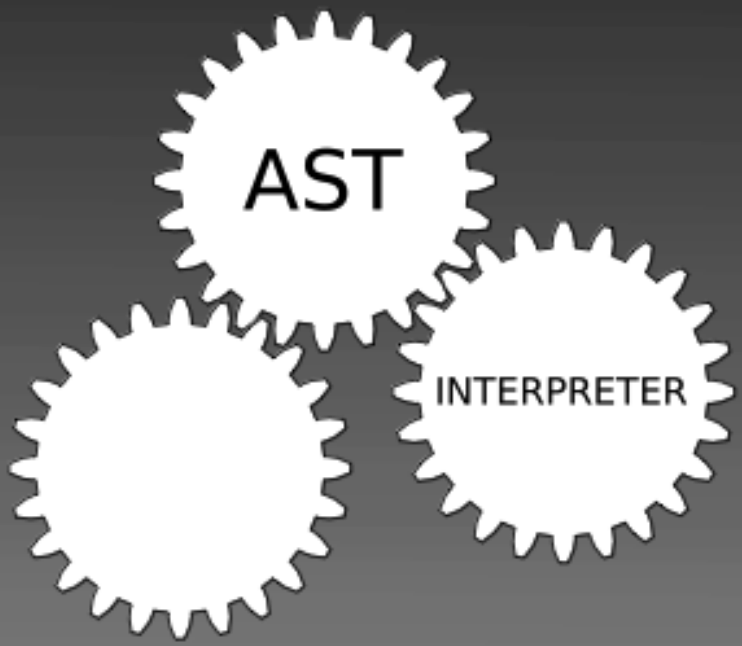
3



Day
4

classes

methods



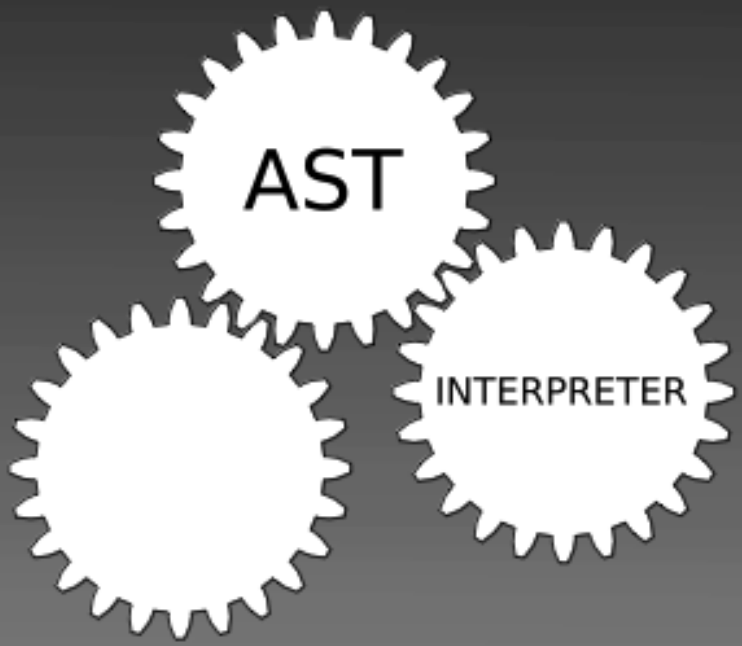
Day

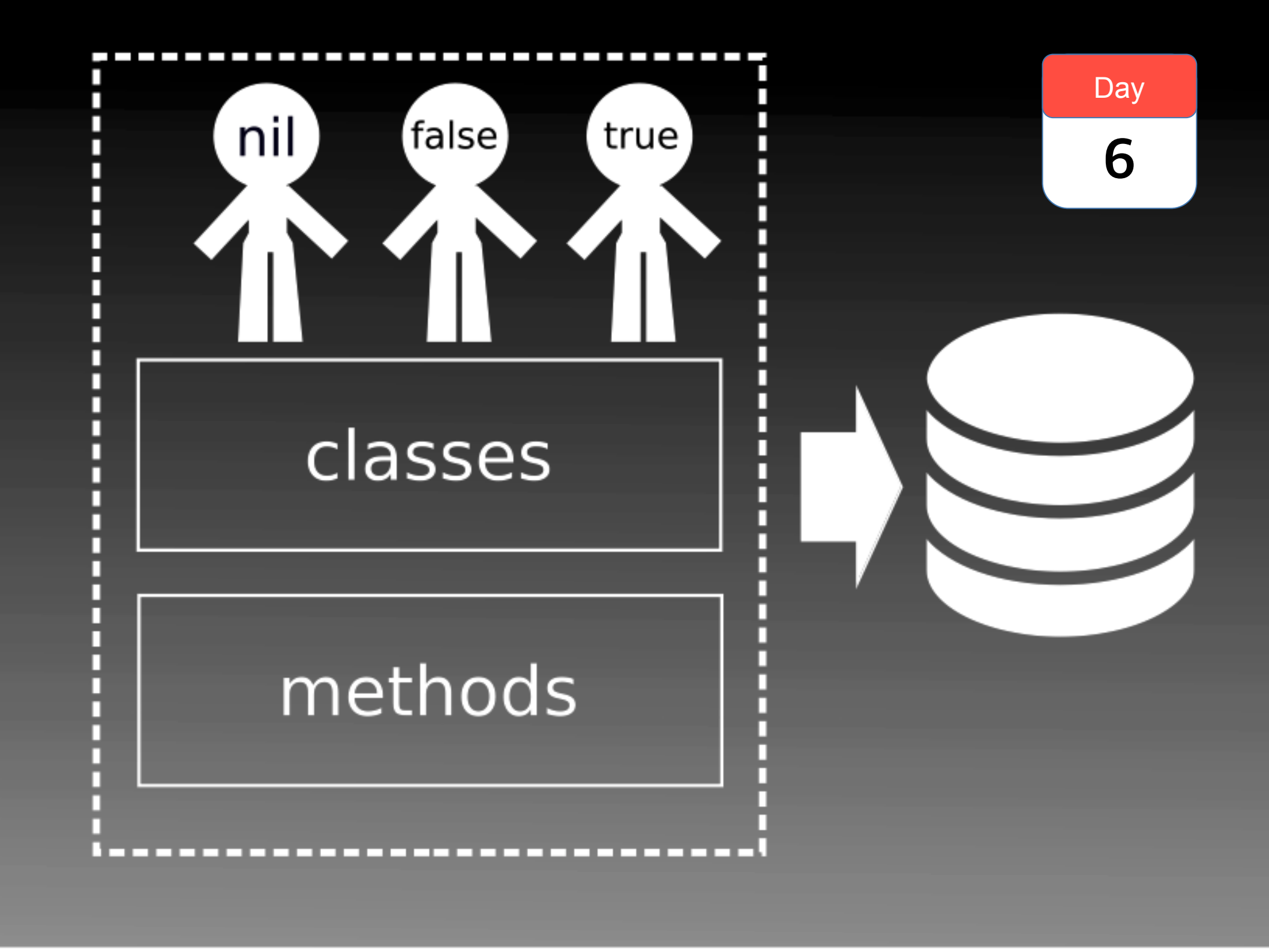
5

classes

methods

process

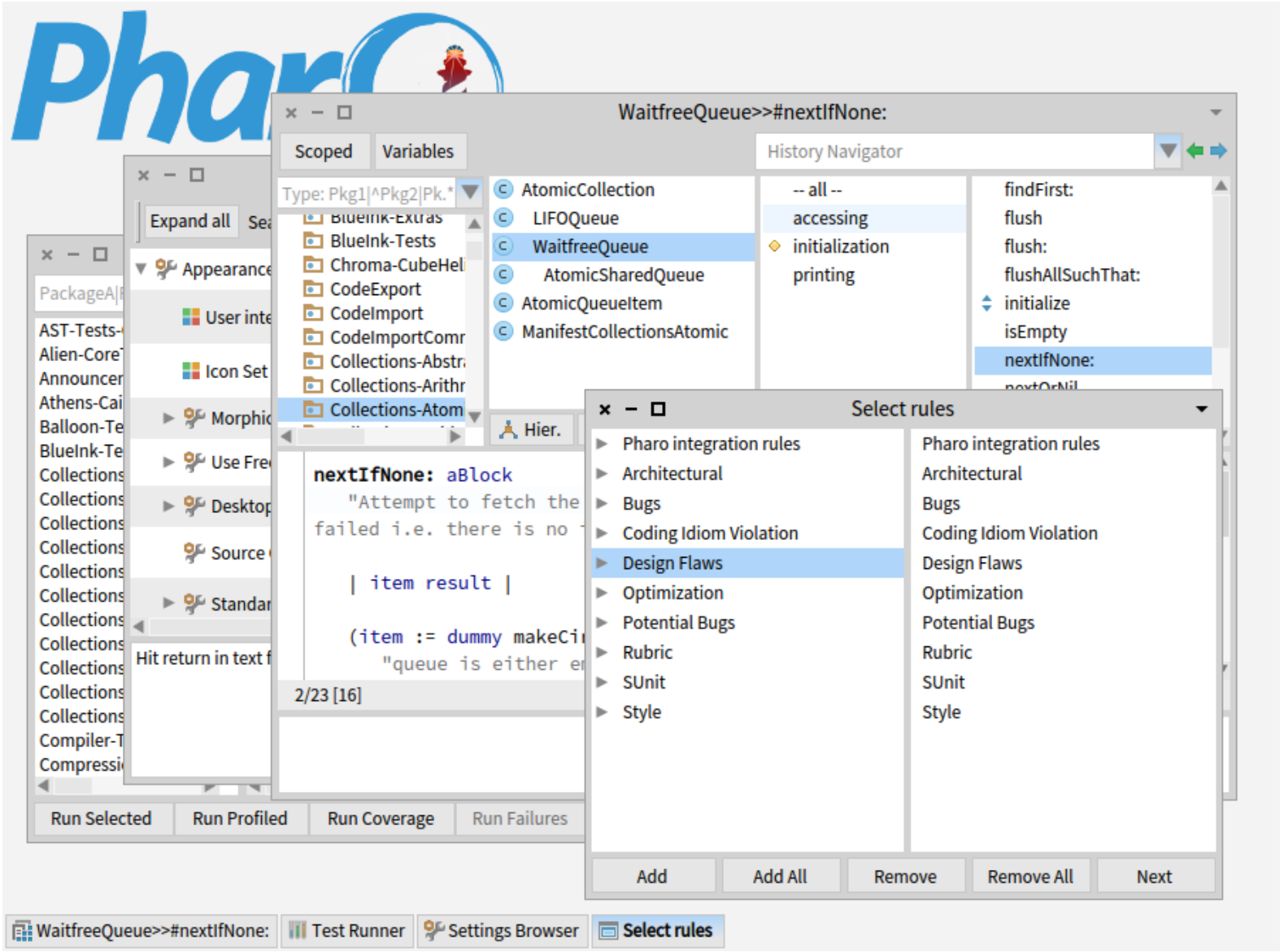



nil
false
true
classes
methods
Day
6

CI jobs for Pharo modularization

- Kernel (shrinked / bootstrapped)
- Monticello
- Network support
- Remote repositories support
- Metacello
- = minimal Pharo
- SUnit, Display support, UFFI
- Morphic core, Morphic
- UI, Basic tools, IDE
- = Pharo

Bootstrapped & reloaded from GIT



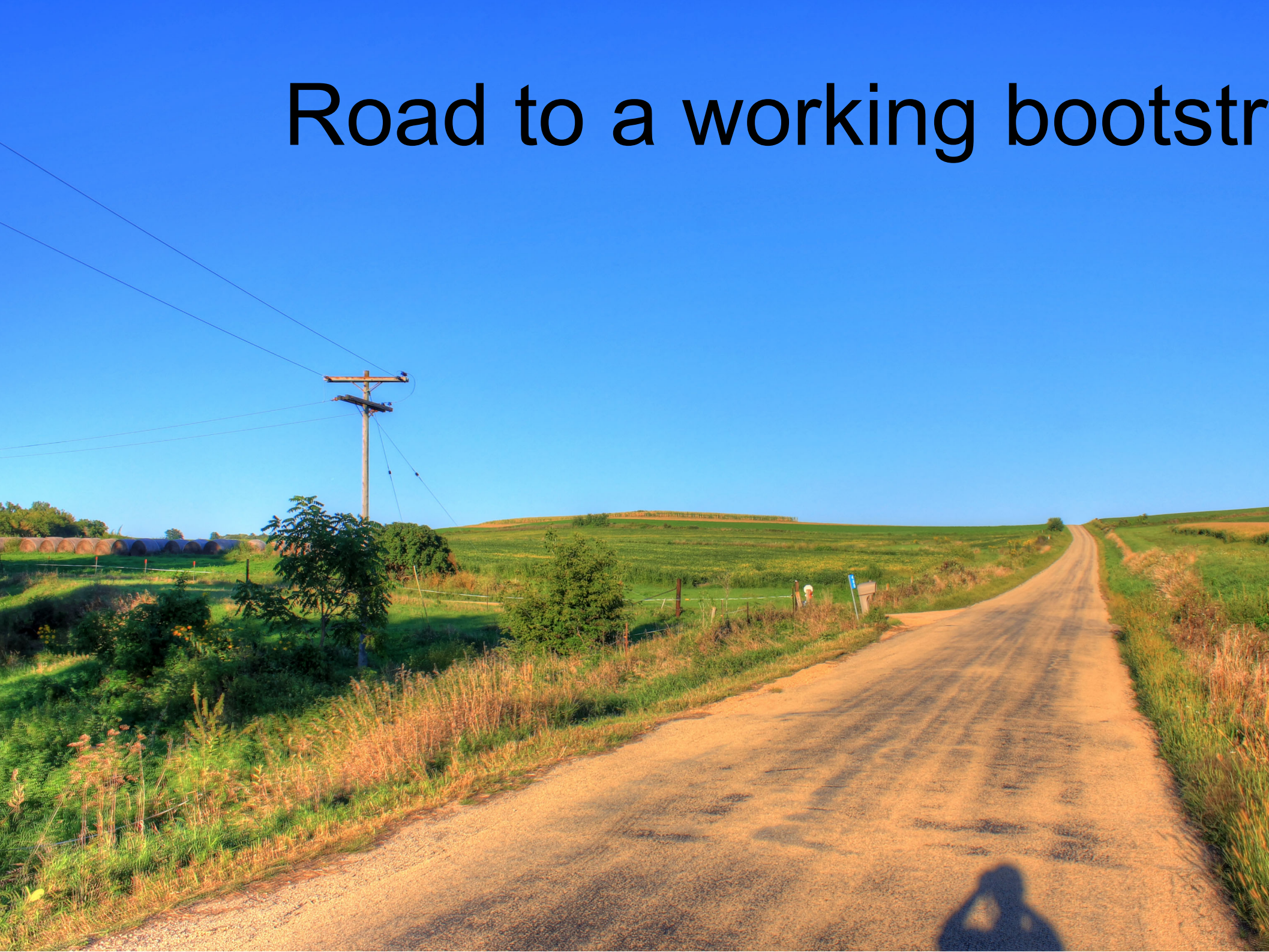

Day
7

More details

Cf PhD Guillermo Polito:
https://hal.inria.fr/tel-01251173

Story #2

Road to a working bootstr

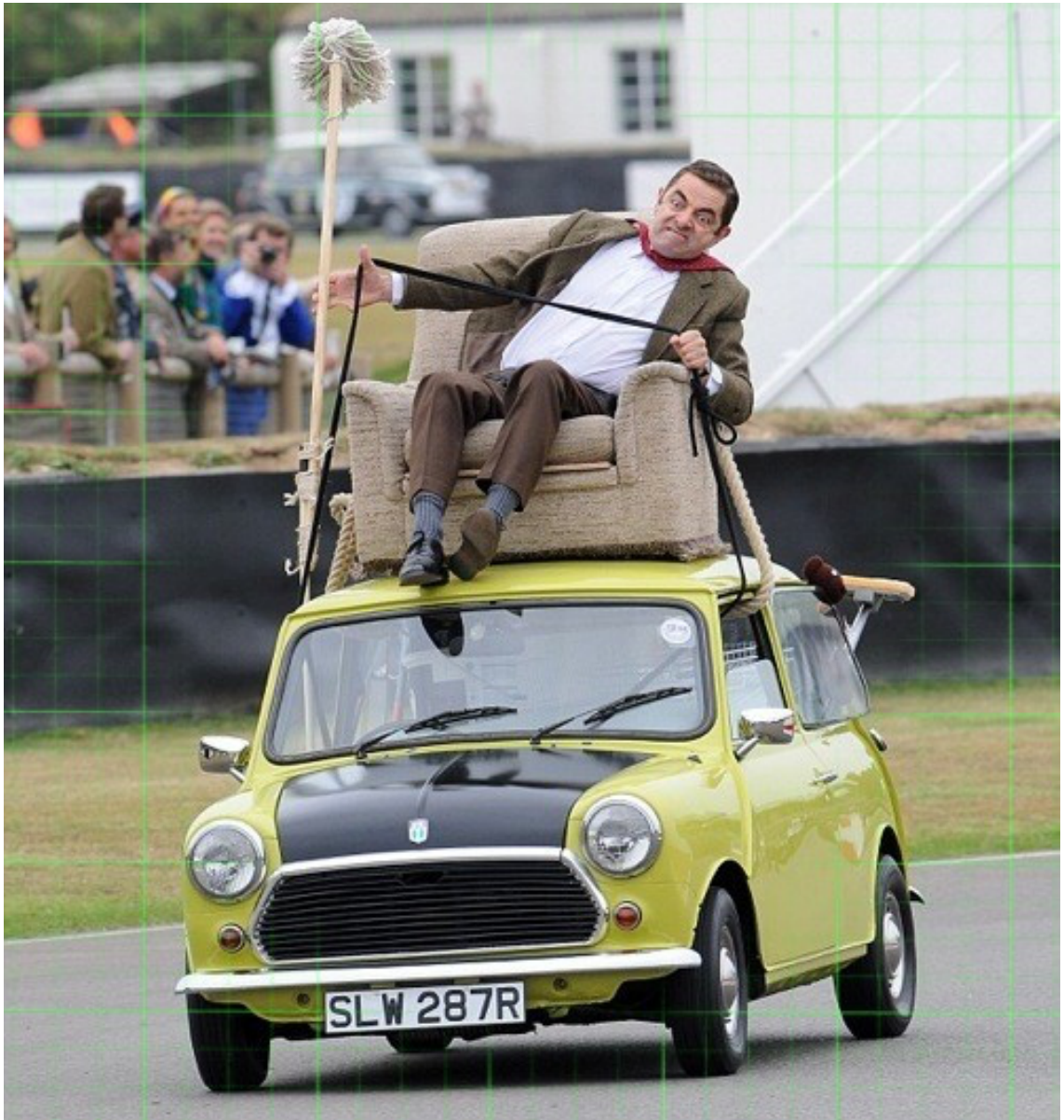


Bootstrap challenge
> language side bootstrap

- Language initialization generally done VM side

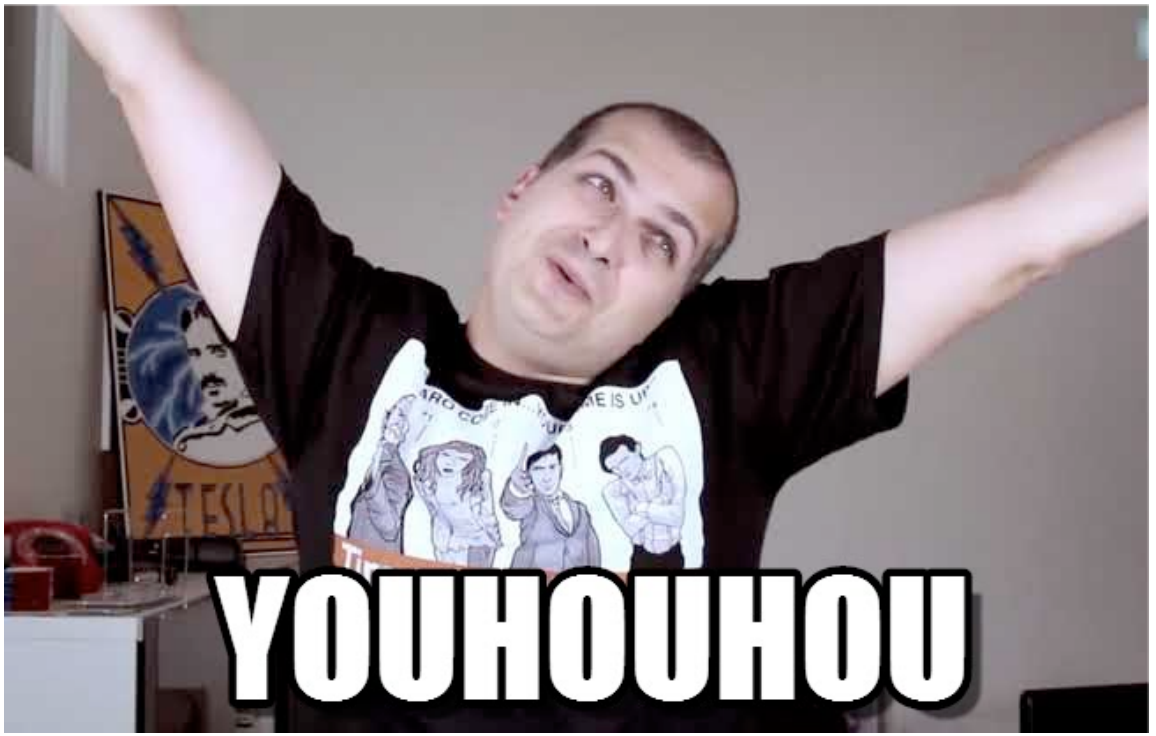
- We want to do it language side:
- Need to run code on top of a language under construction

Bootstrap challenge
> language side bootstrap



Road to a working bootstrap

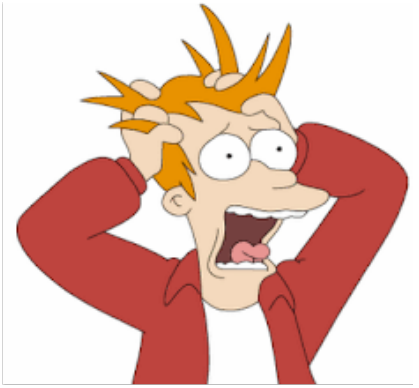
- First bootstrapped image!



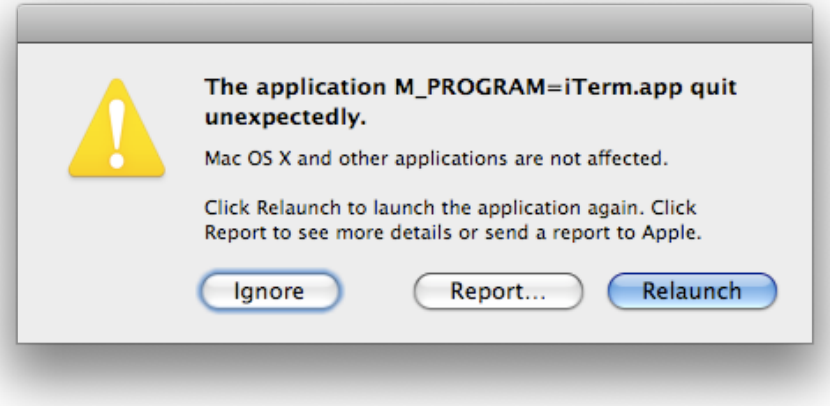
Road to a working bootstrap

Road to a working bootstrap

- Execute the system ….



the VM crashes



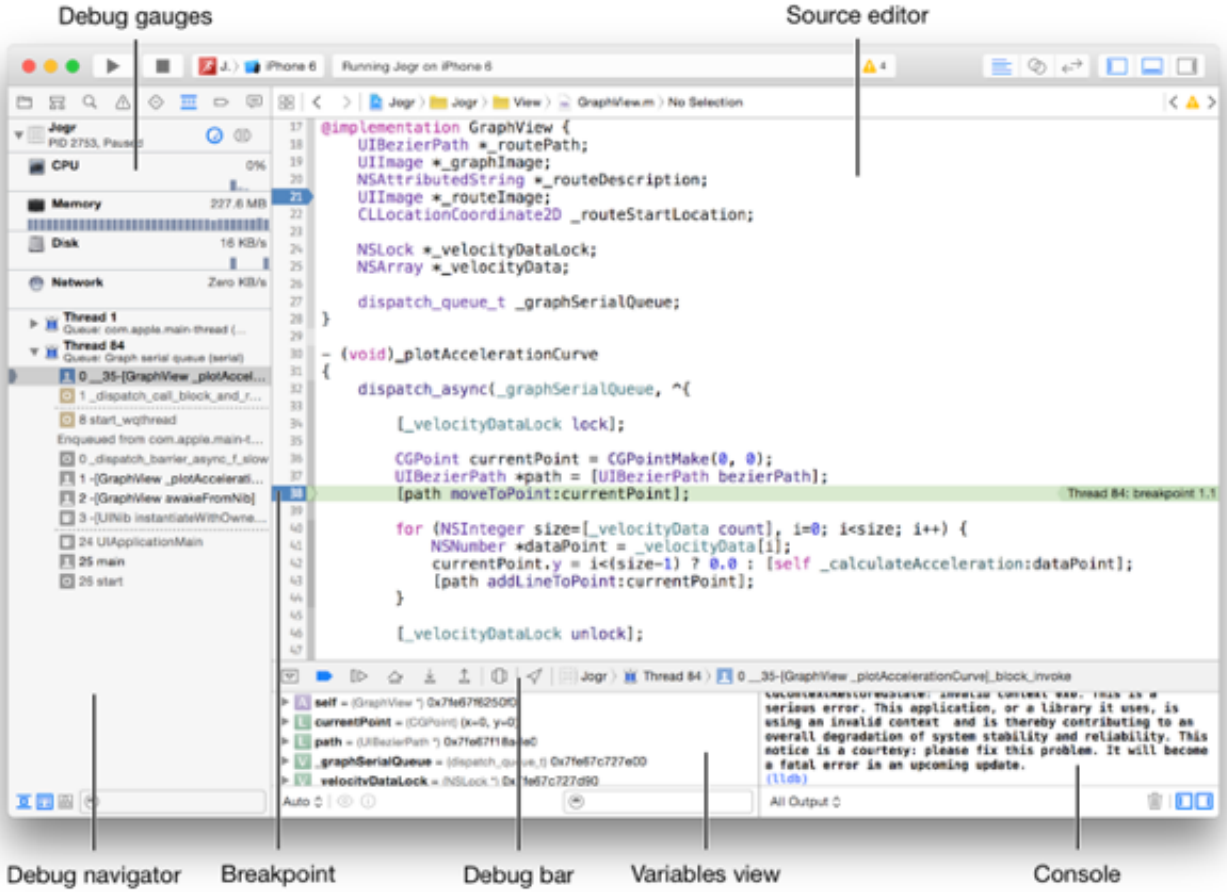
Road to a working bootstrap > some debugging examples

- Missing class in the boostrap
  e.g. Float
- superclass not set
- superclass set to a wrong value

Road to a working bootstrap

- Compile VM in debug mode

- Run bootstrapped system 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

Bootstrapped Pharo

Soon in production (Pharo 7.0)

All packages reloaded!

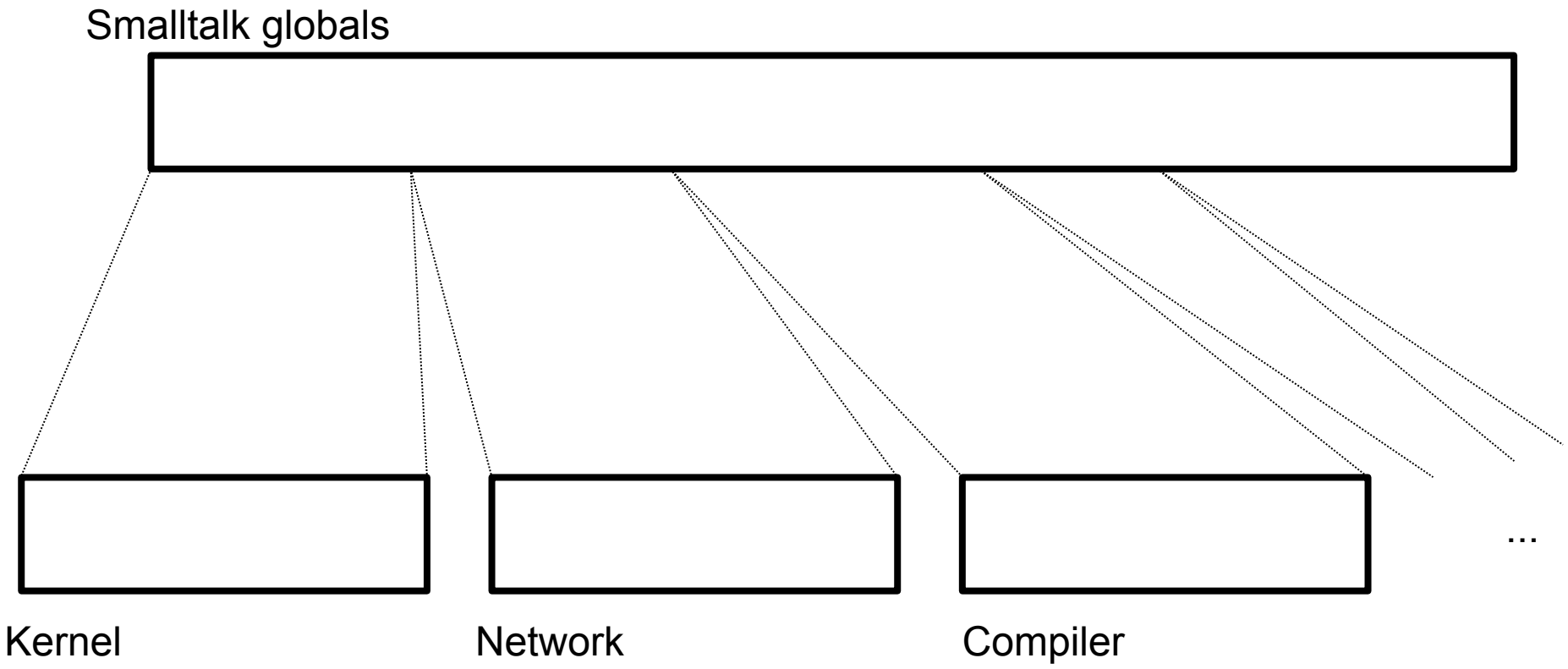
Architectural rules

Tools to the rescue

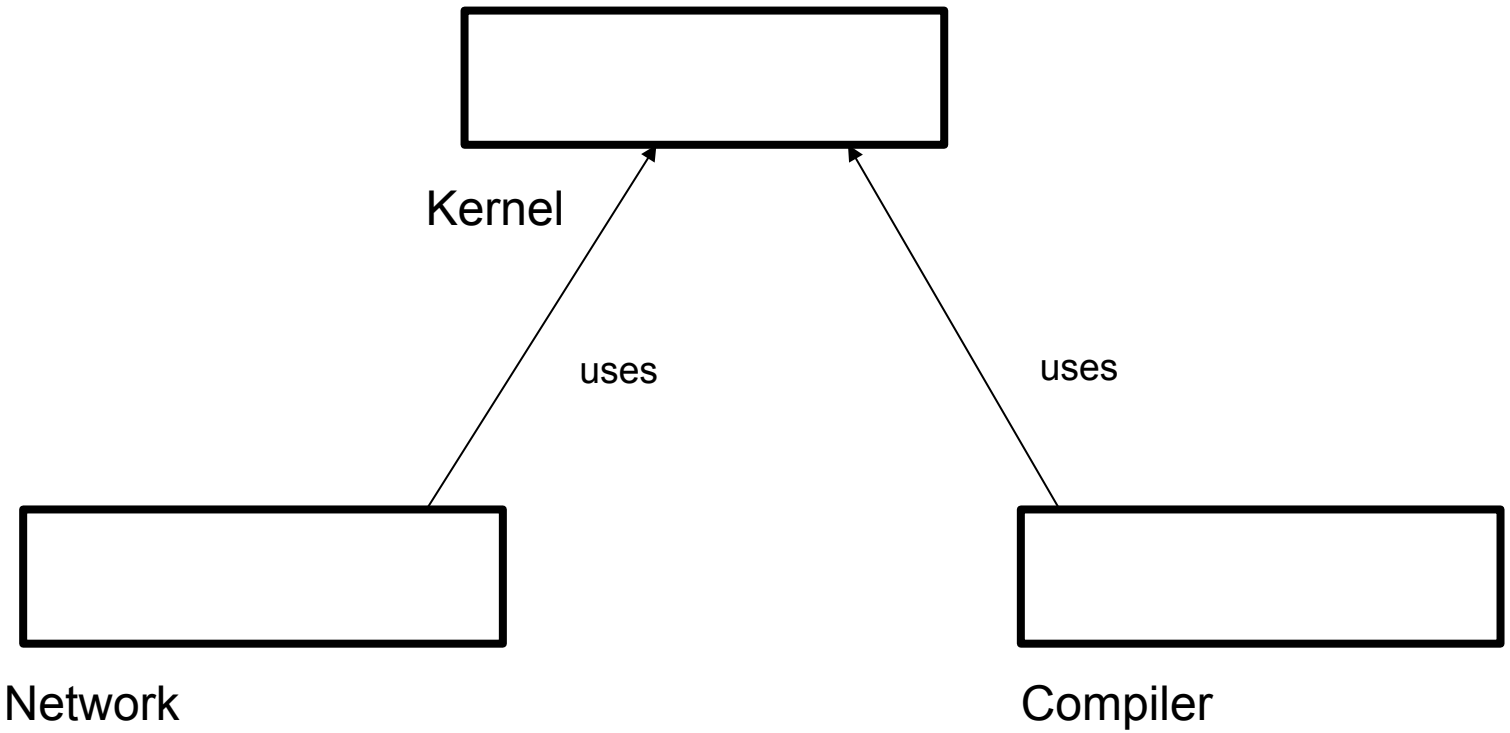
Metalo

- Designing a module system
- For live programming: programmer feel and feedback
- Should be usable by normal programmers
- Should be applicable and bootstrappable on plain Pharo

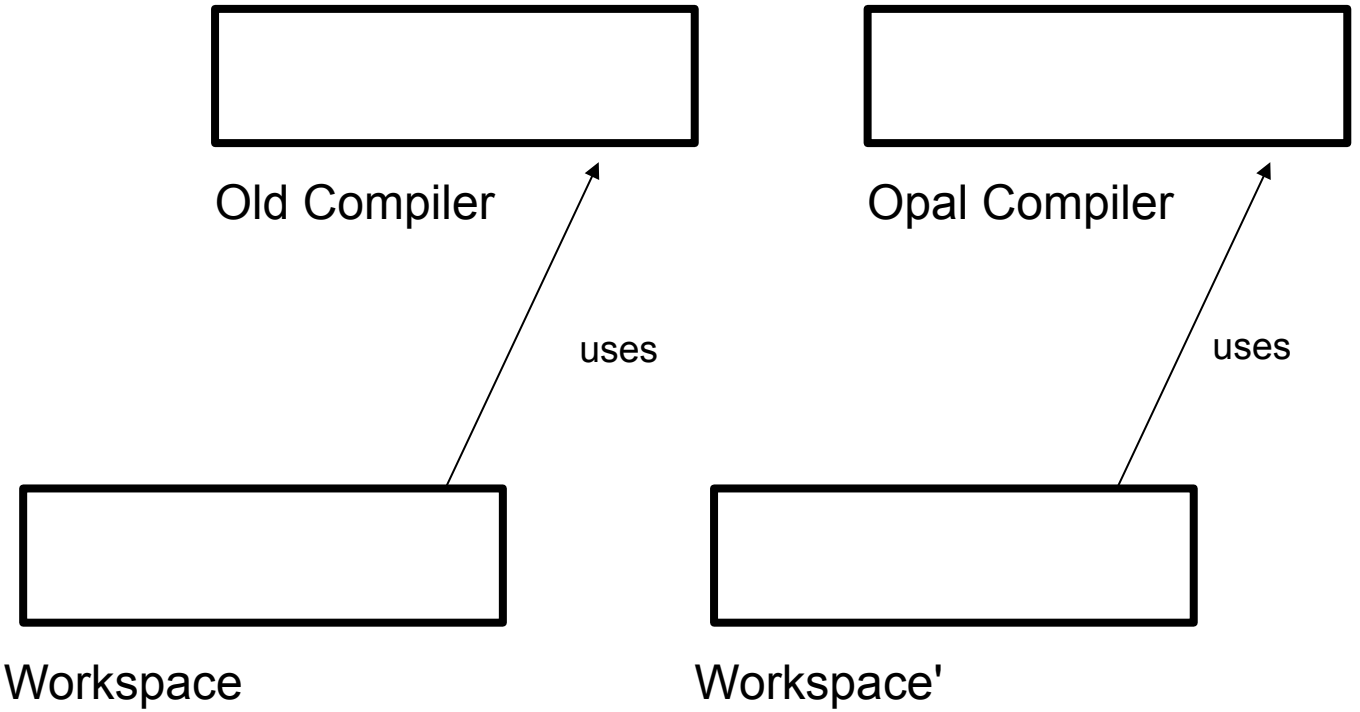
Objective 1 - Namespace



Objective 2
Better Dependency Management



Objective 3
Module testing



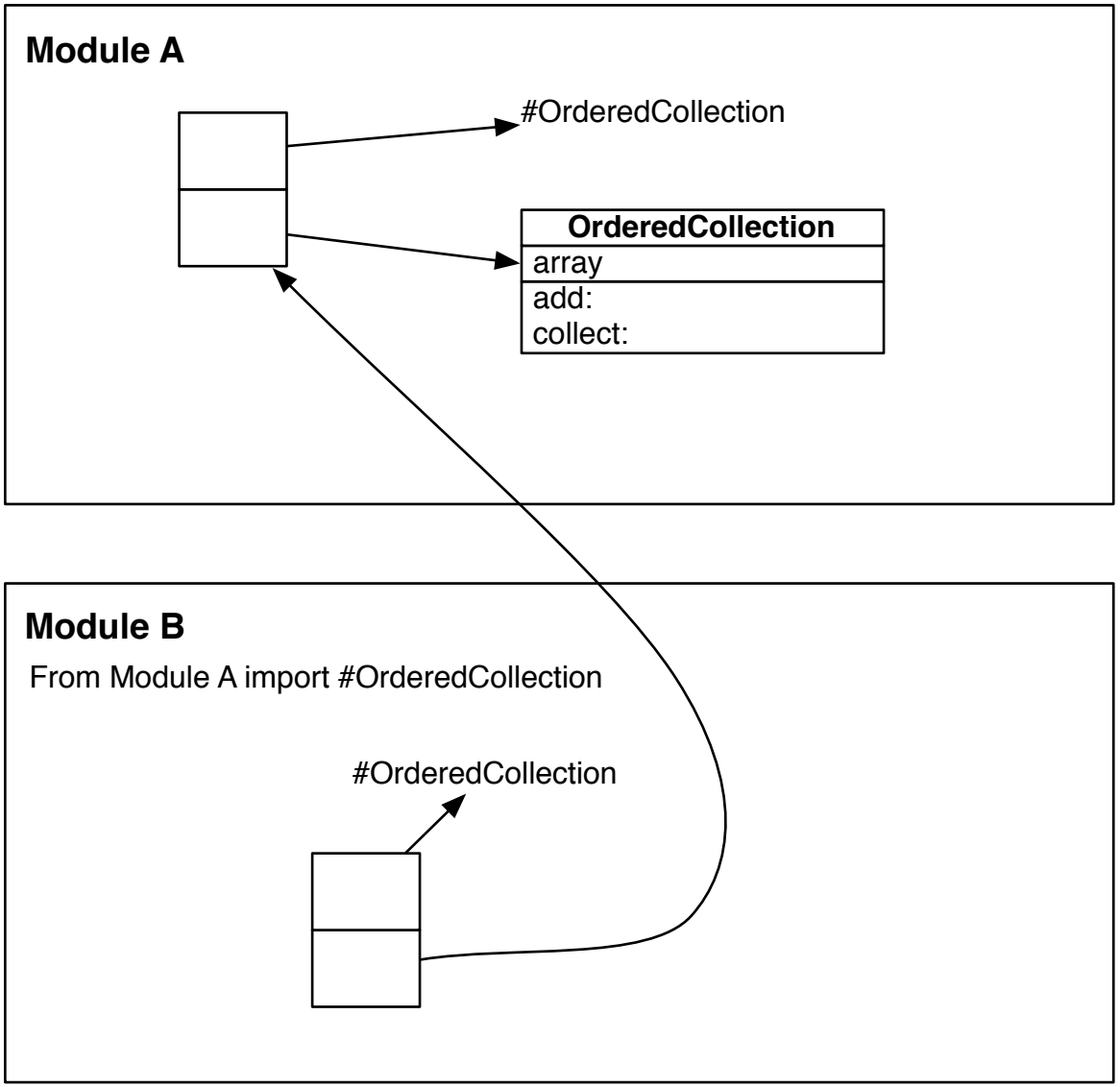
Current Design

- No nesting
- No name lookup
- Explicit import
- To be introduced incrementally (module and monolithic side by side for a while)
- Tools should support it

Current Design

- A module has its own namespace
- Shared bindings between modules as current

Shared Bindings



Tools

- All the tools should be module-aware
- Live programming
  - Workspace
  - Debugger
  - Inspector
  - Class browser
  - Refactorings

Conclusion

- Working with real system is rewarding
- Modularisation is a battle
- Tools are good weapon against entropy
- We are getting there!