Modularity by Construction

Guille Polito

ESUG'26 - Plovdiv, Bulgaria

guillermo.polito@inria.fr
@guillep

Quick About Me: Guille

guillermo.polito@inria.fr
@guillep

- Pronounced *giʃe* (guichet in FR, ~ghisheh in EN?)
- **Now:** Researcher at Inria - Lille
- Pharo Contributor since ~2010

- **Keywords:** compilers, testing, test generation
- **Interests:** tooling, benchmarking, 日本語, board games, batman, concurrency, chess

If any of that interests you, come talk to me!

You Probably Know Me From…

The Pharo Module System

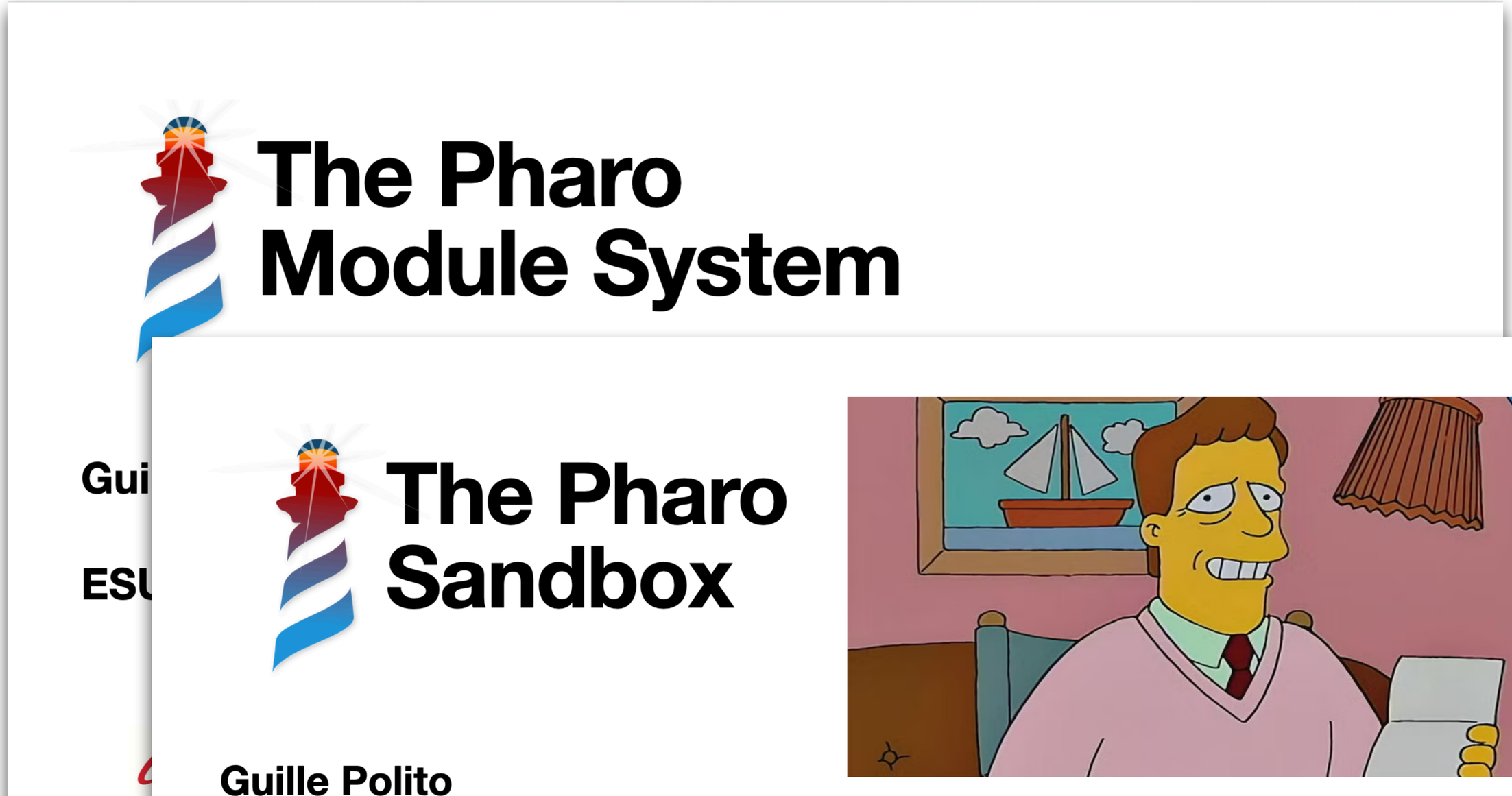
Guille Polito

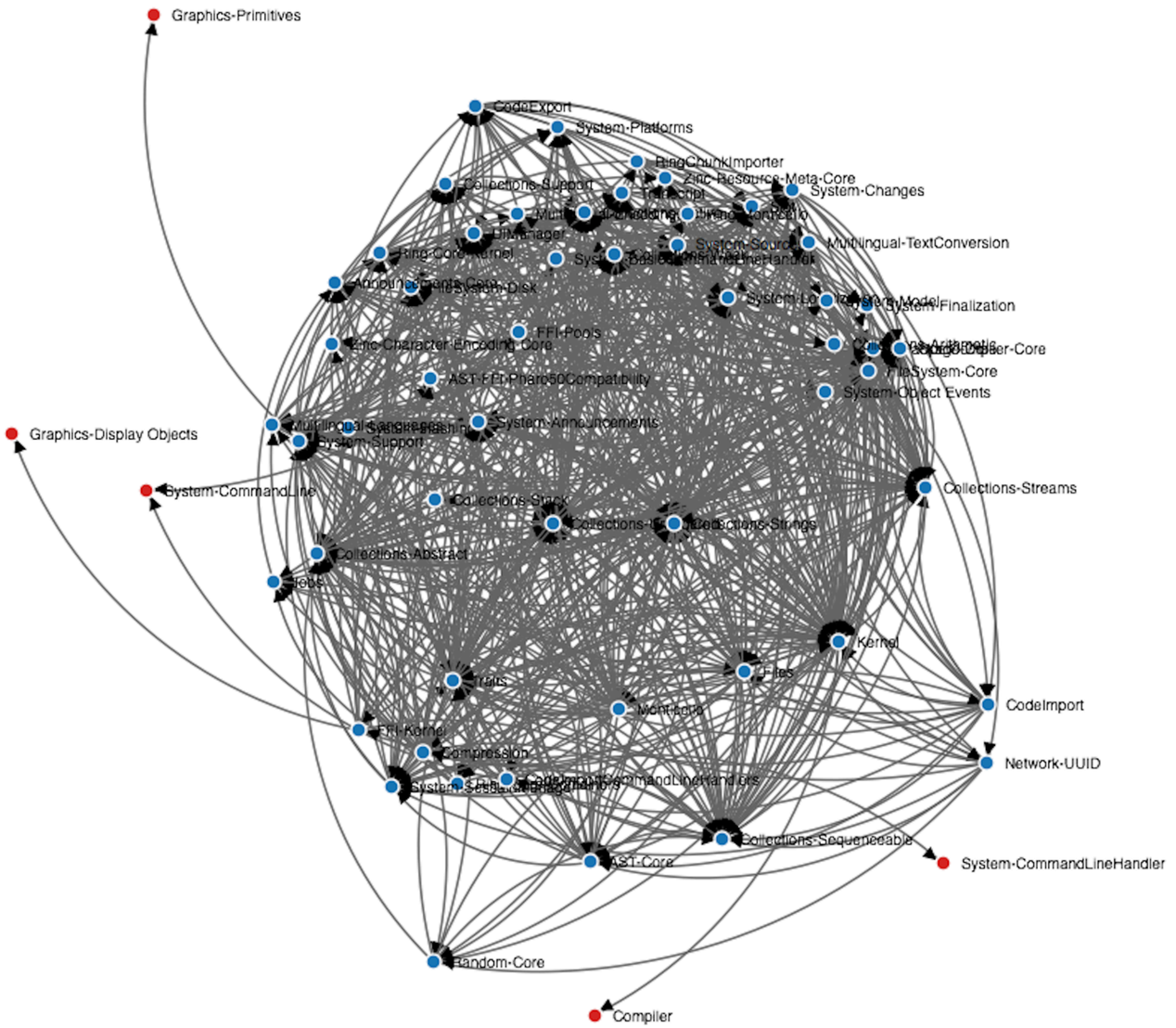
ESUG'26 - Plovdiv, Bulgaria

guillermo.polito@inria.fr
@guillep

Inria · Evref · cnrs · CRIStAL Centre de Recherche en Informatique, Signal et Automatique de Lille · Université de Lille

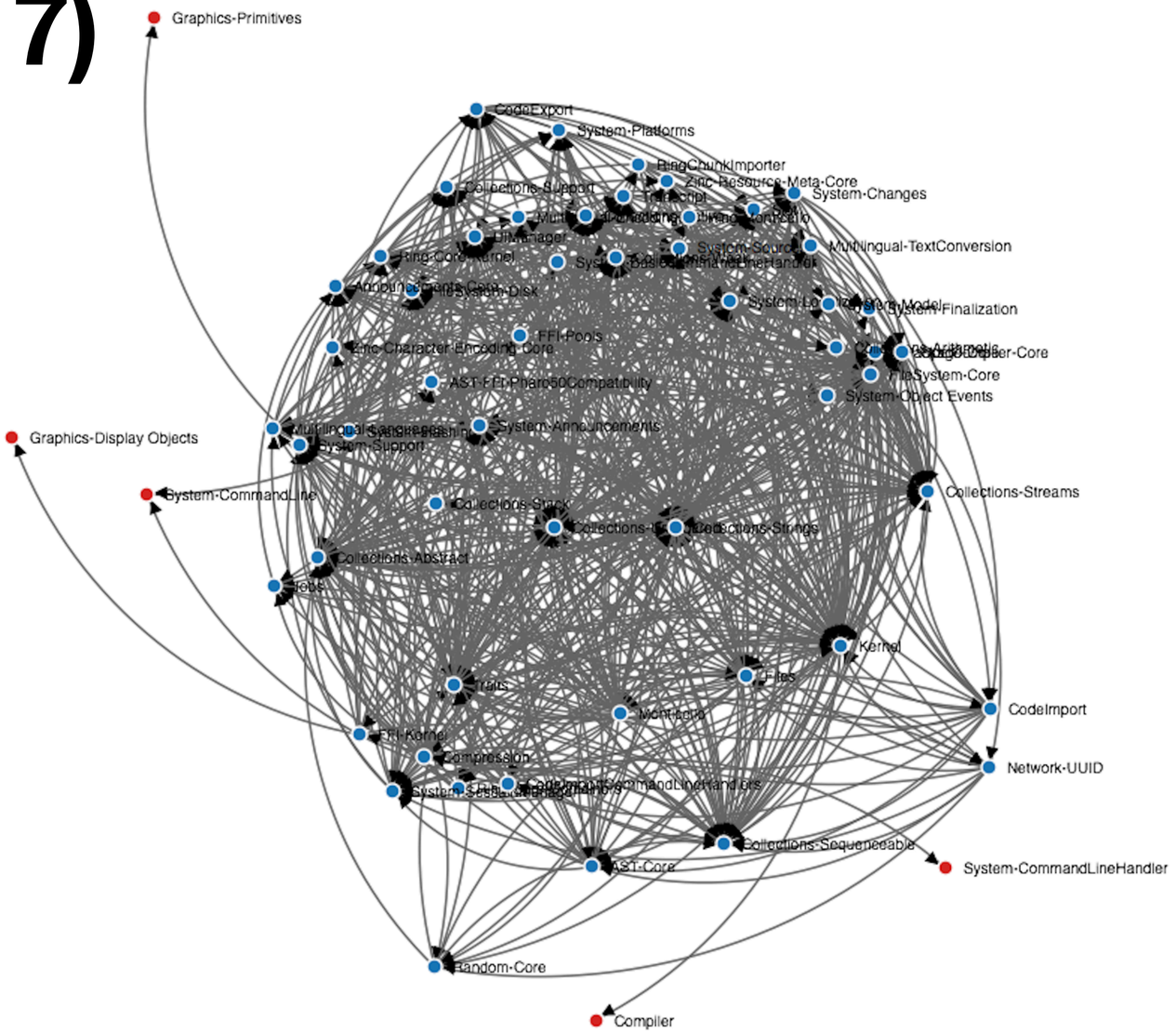
You Probably Know Me From…




Graphics-Primitives
CodeExport
System-Platforms
RingChunkImporter
Zinc-Resource-Meta-Core
System-Changes
Collections-Support
Multilingual-TextConversion
System-Finalization
FFI-Pools
Zinc-Character-Encoding-Core
FileSystem-Core
System-Object Events
Graphics-Display Objects
System-Announcements
System-Support
System-CommandLine
Collections-Streams
Collections-Strings
Collections-Abstract
Kernel
Files
CodeImport
FFI-Kernel
Network-UUID
Compression
System-CommandLineHandlers
Collections-Sequenceable
AST-Core
System-CommandLineHandler
Random-Core
Compiler

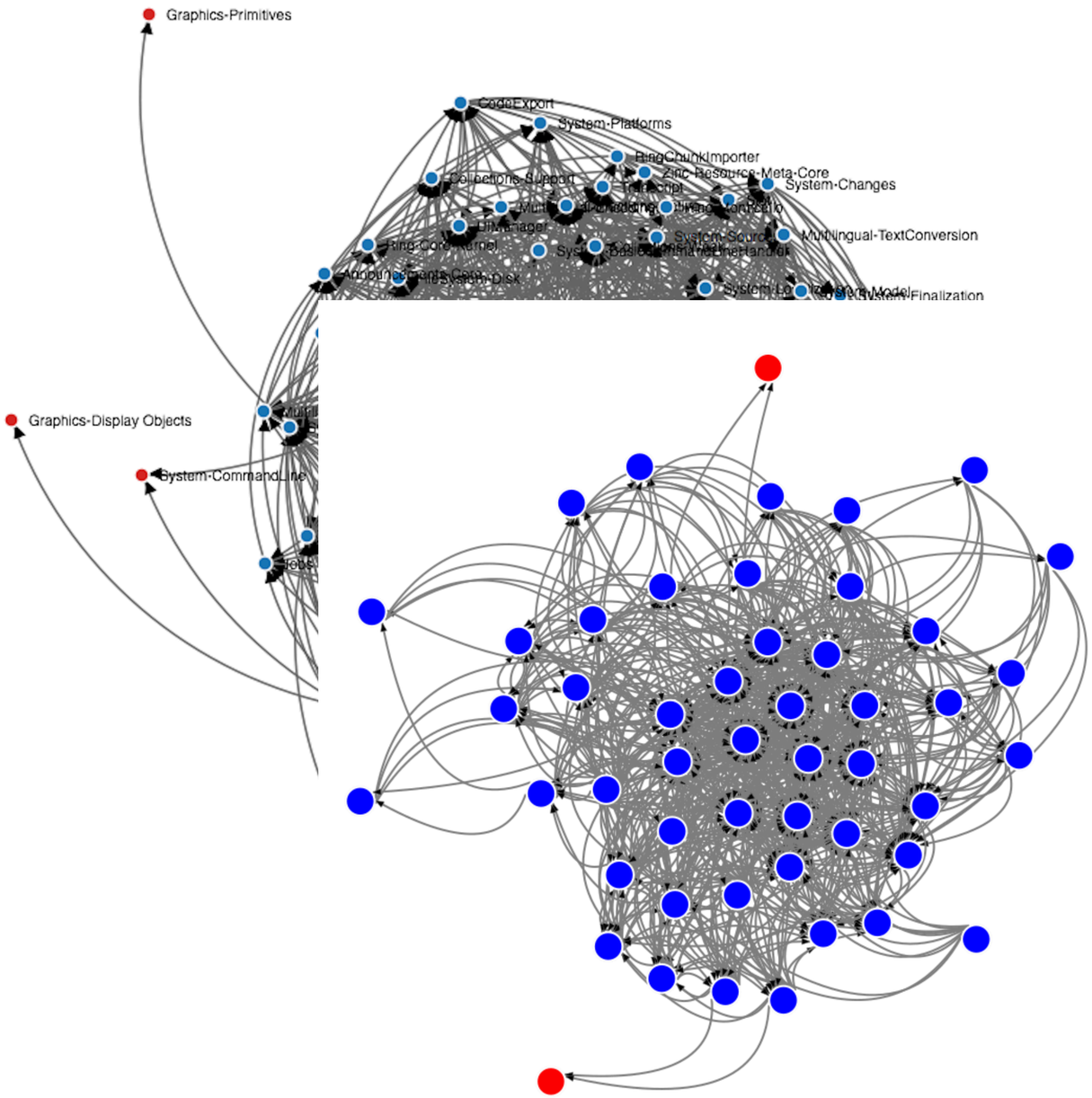
Pharo Kernel (2016-17)

- Kernel = Minimal Image
- 53 packages



2016 vs 2026

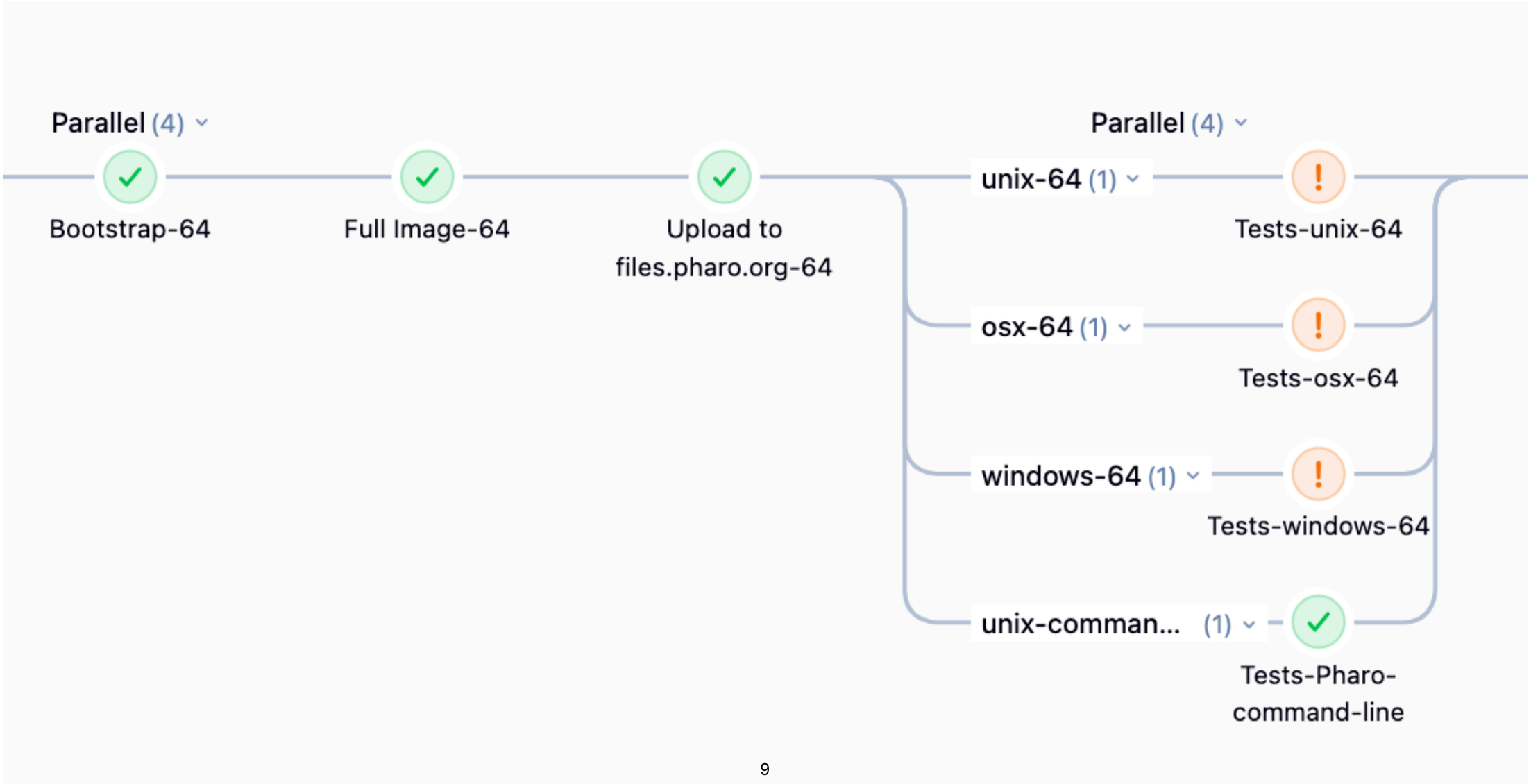
- Kernel = Minimal Image
- 2016 = 53 packages
- 2026 = 25 packages
  - The delta? loaded at later stages



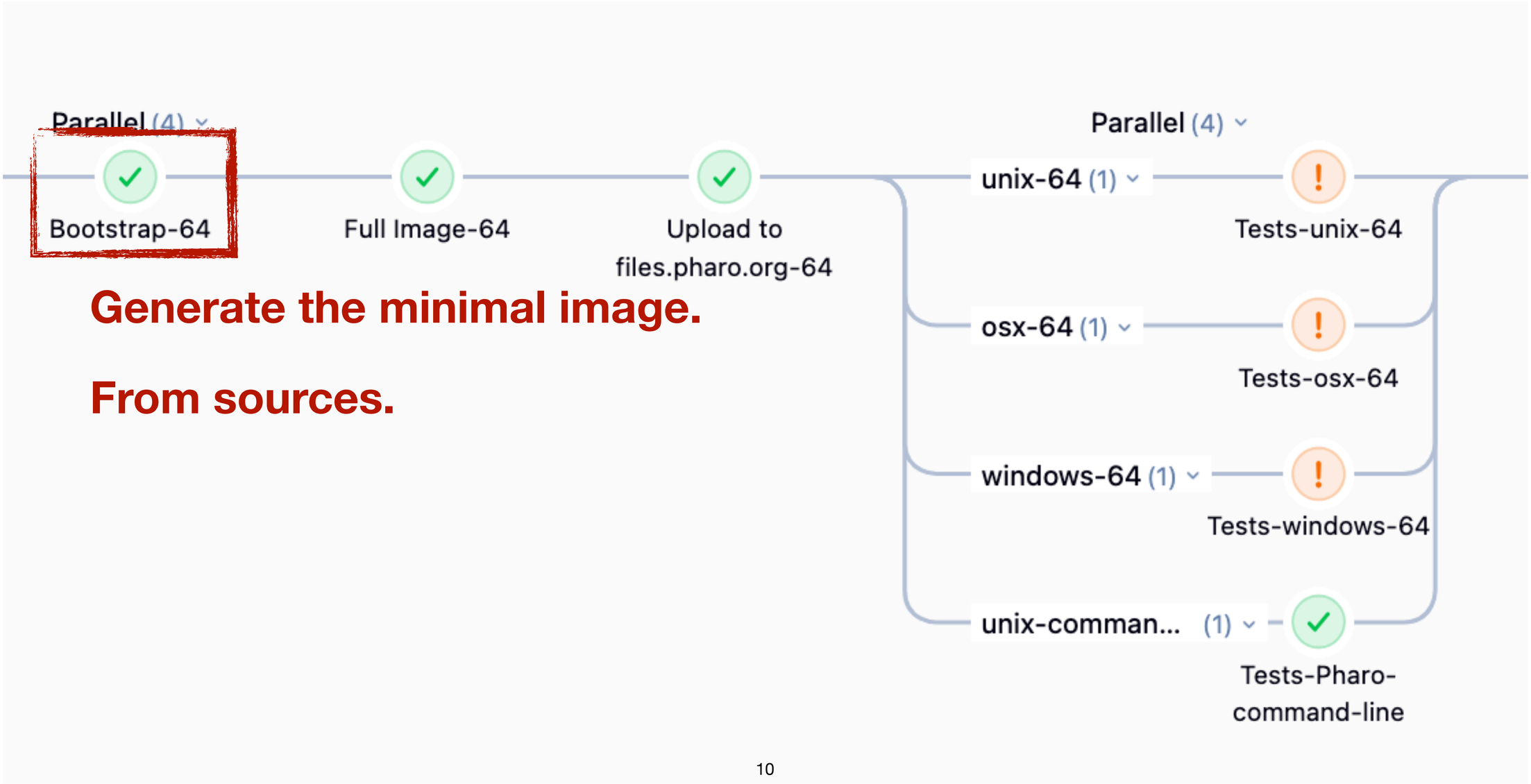
Behind the Scenes

- The effort of many people
- Analysing dependencies
- Refactoring
- Redesign
- Fight with compatibility

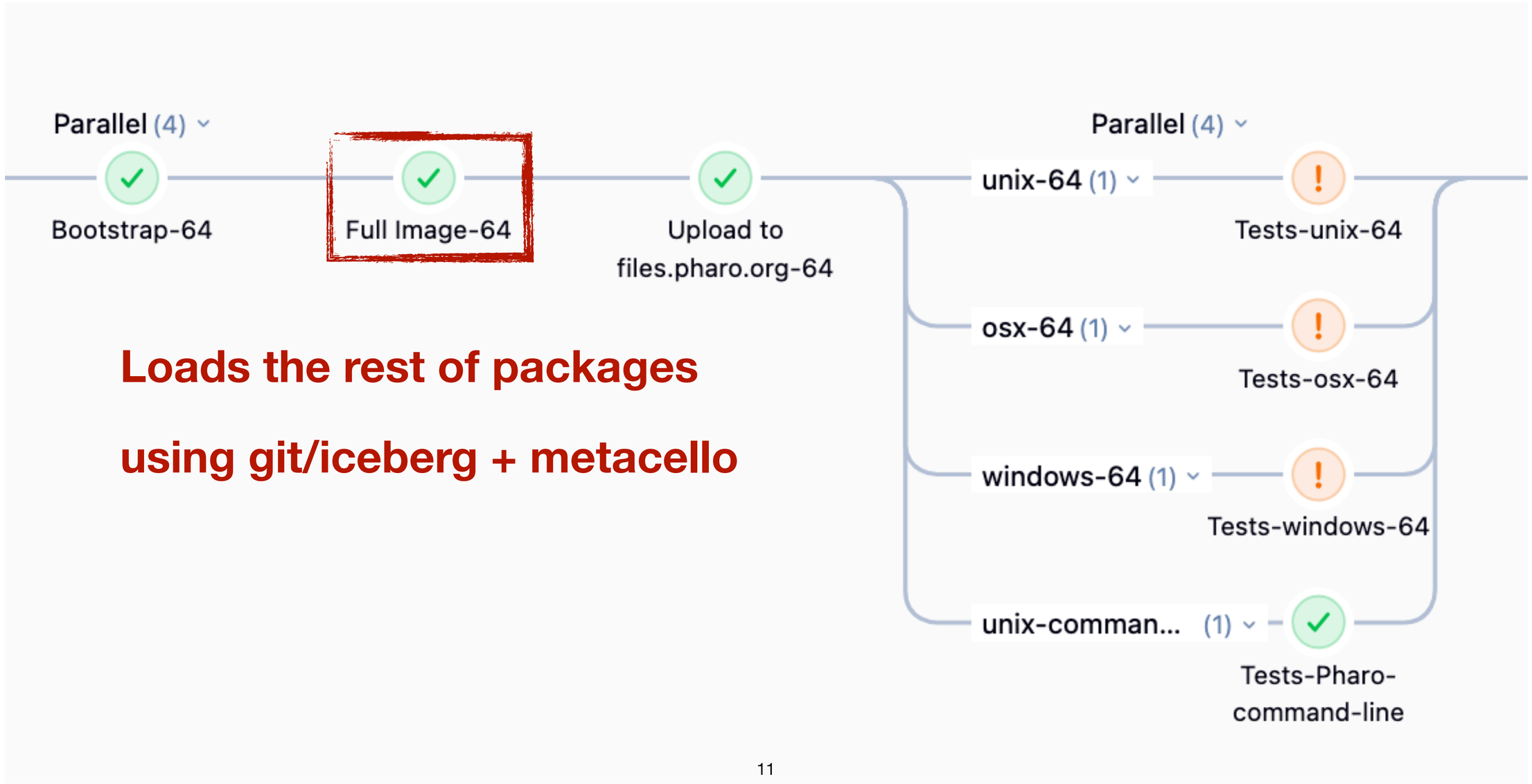
Pharo Build System



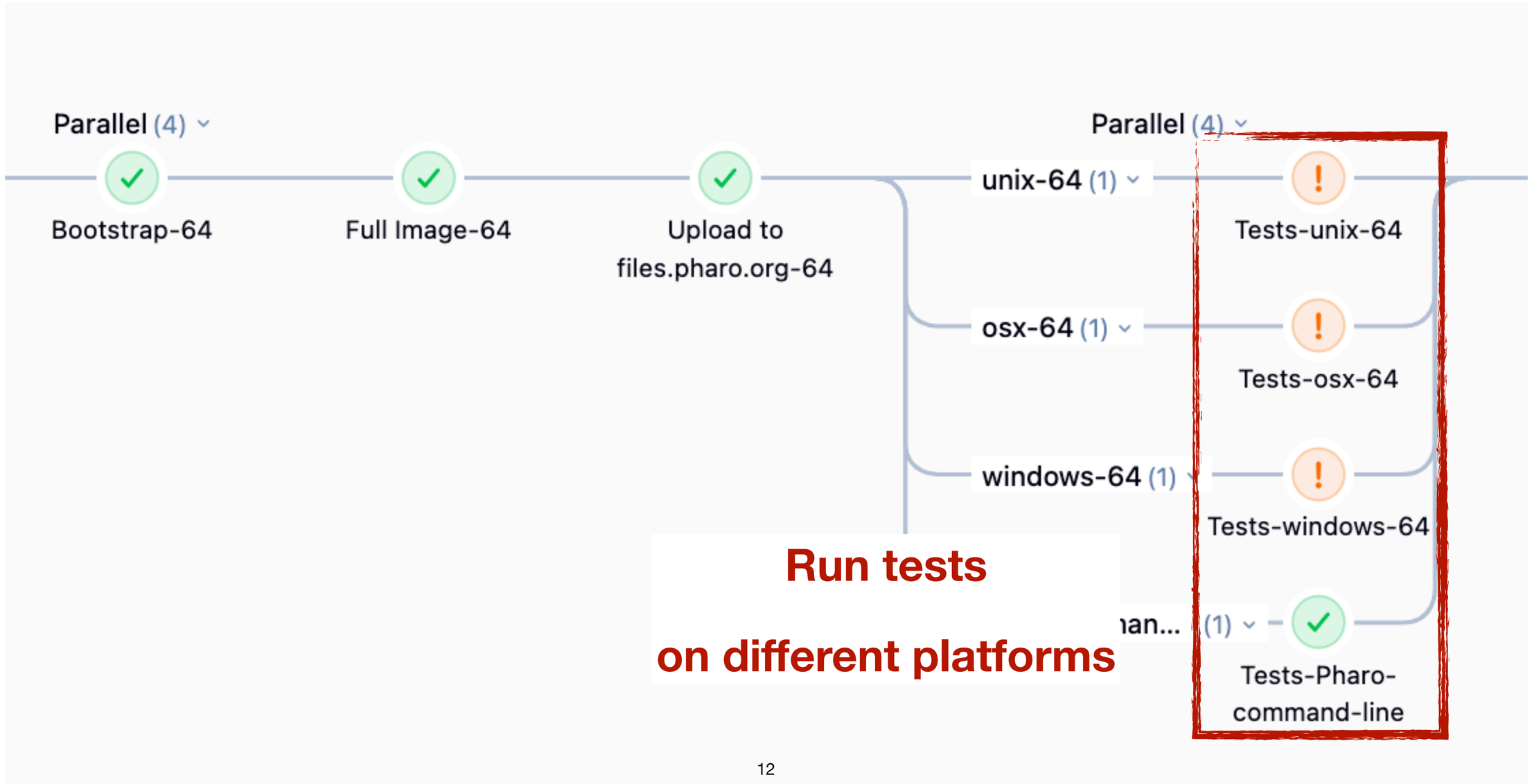
Pharo Build System



Pharo Build System

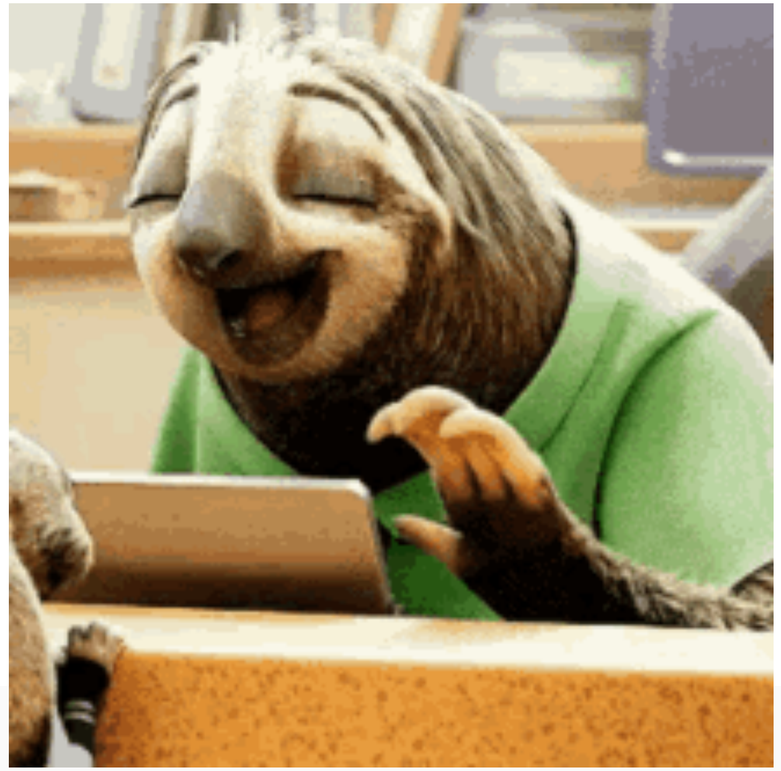


Pharo Build System



Not Very Agile

- Feedback arrives very late!
- Tests are run in the full system
  - Dependencies not well defined
  - Side effects affect other tests!



A Vision

Don’t break
my toys,
I’ll not break yours

~~Not~~ Very Agile

- Feedback arrives very late!

  Check as early as possible!

- Tests are run in the full system

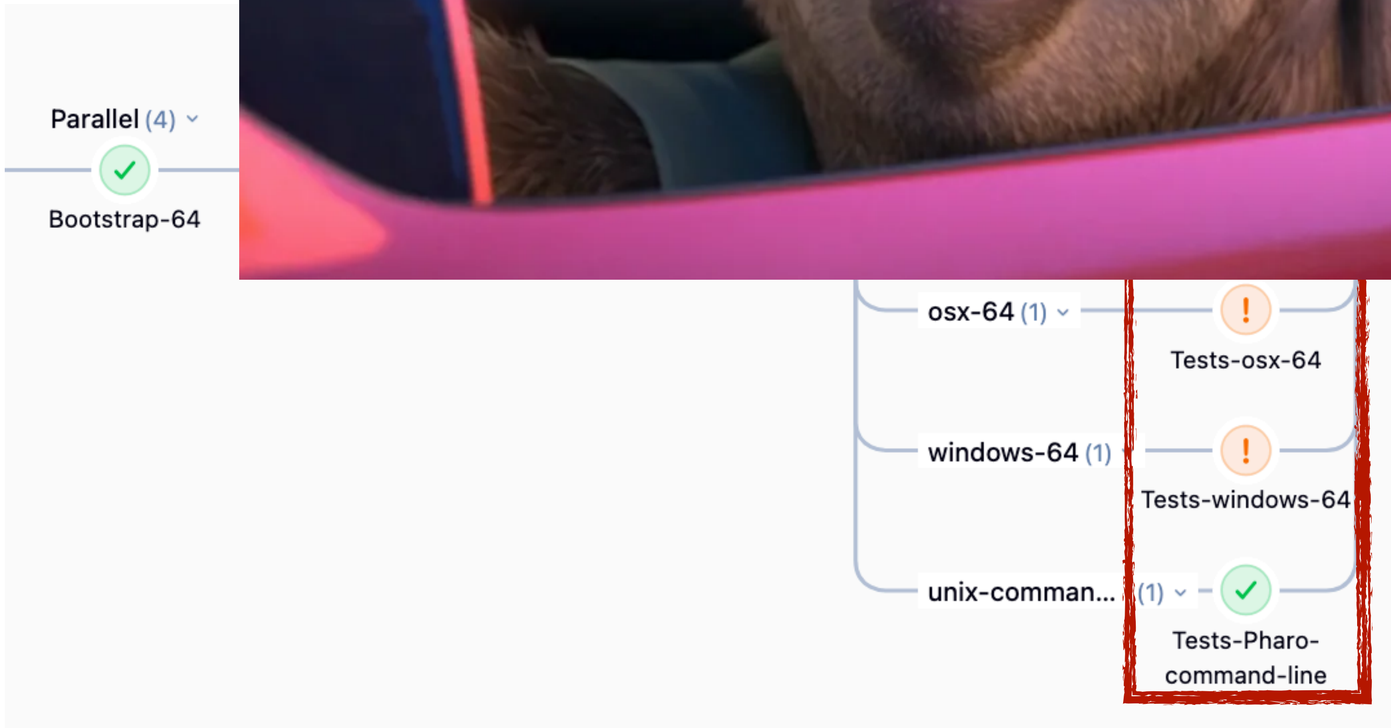
  Controlled Dependencies!

  - Dependencies not well defined

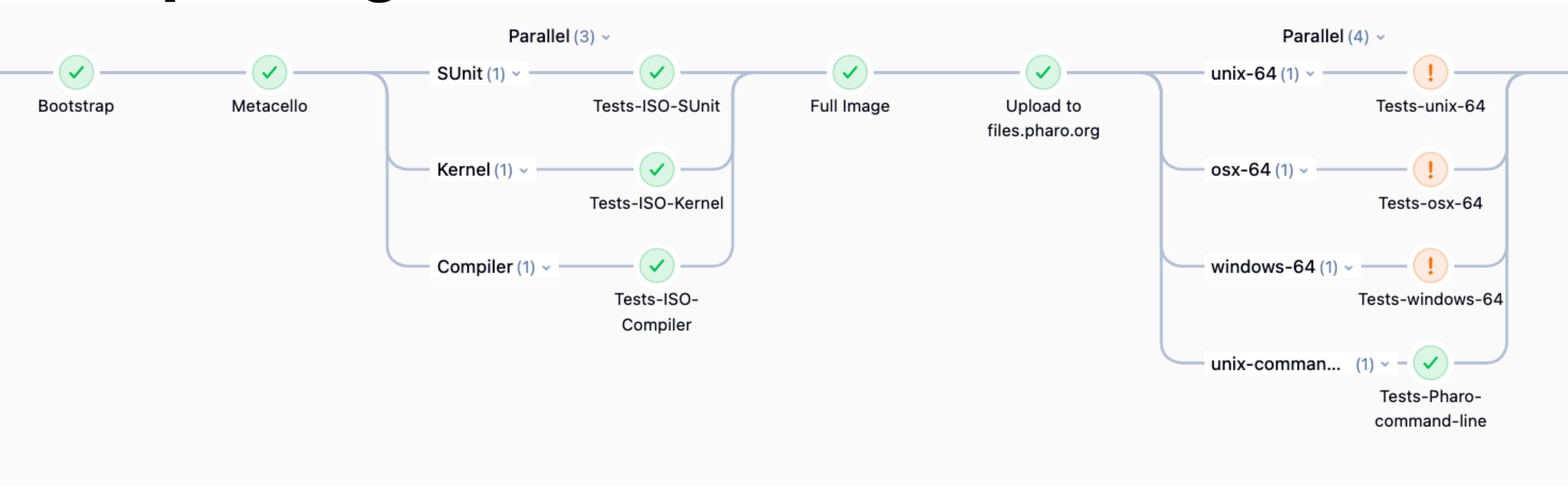
    Strict loading mode!

  - Side effects affect other tests!

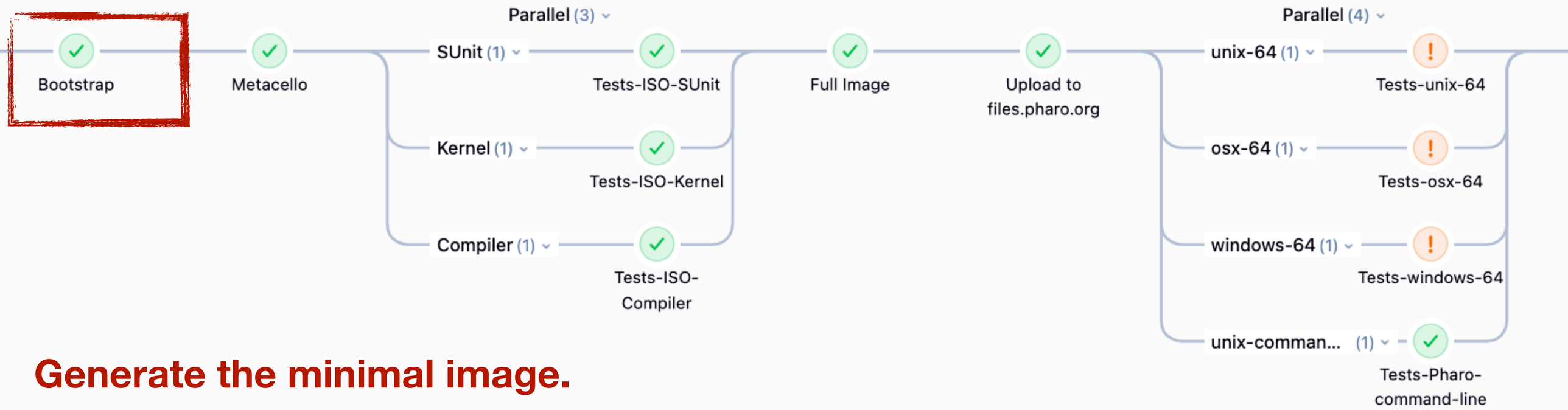
    Sandboxing!



Improving the CI Flow



Improving the CI Flow

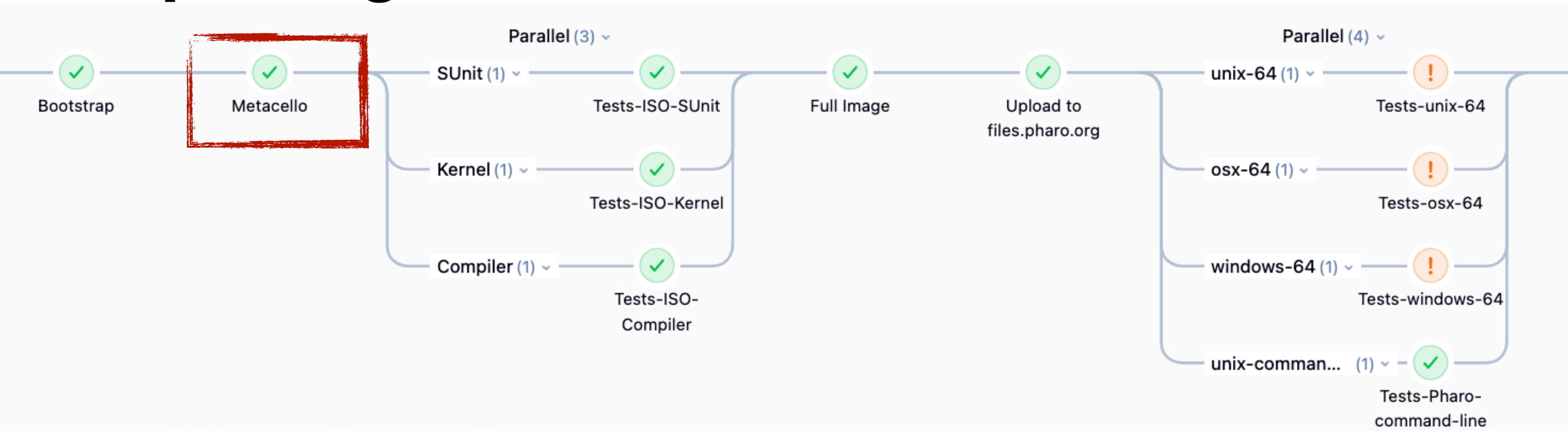


Generate the minimal image.

From sources.

Same as before.

Improving the CI Flow

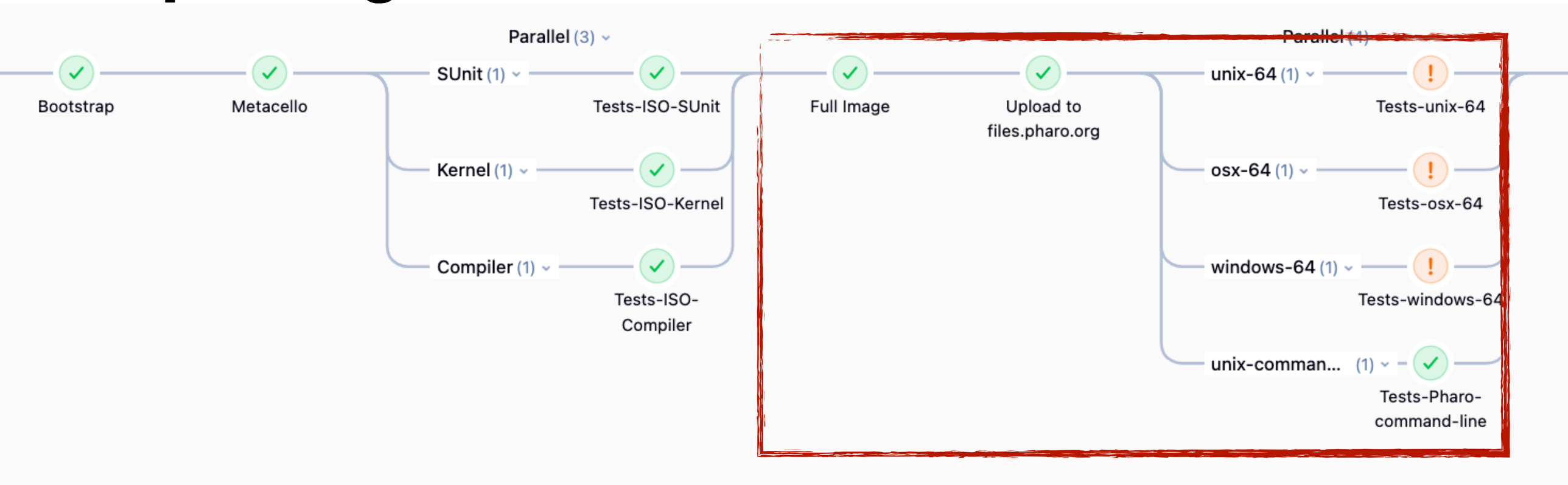


Minimal Metacello installation.

Split from the previous “Full image”.

Opens the door to minimal image testing!

Improving the CI Flow

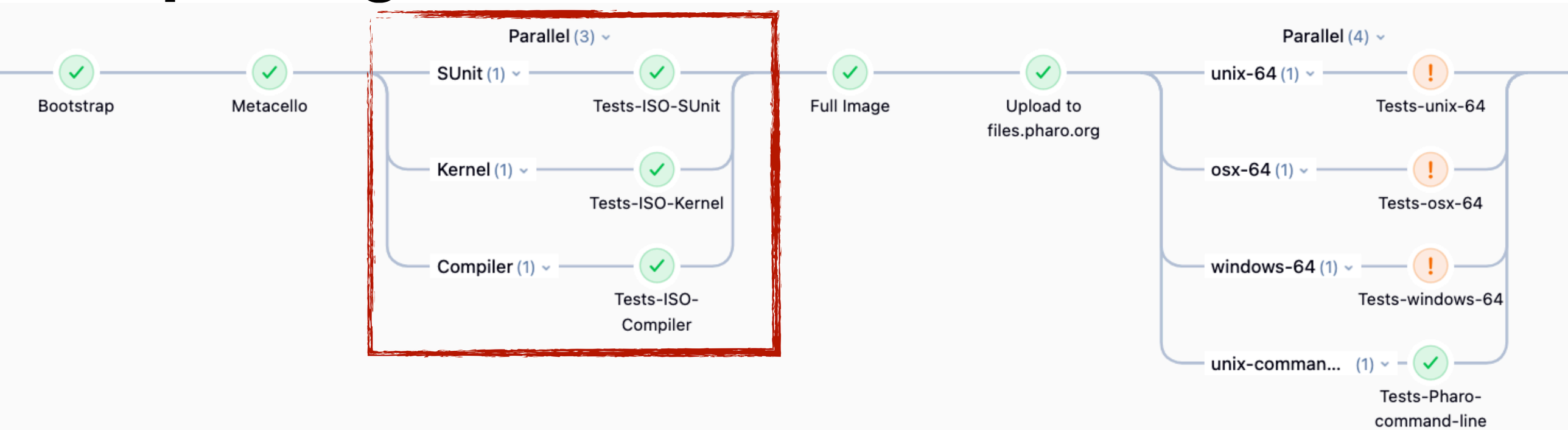


Same as before:

+ finish loading all packages

+ run all tests

Improving the CI Flow



Package verification in *isolation!!*

Package Verification in Isolation

```
./pharo metacello.image metacello install --strict SUnit --groups Core

./pharo metacello.image metacello install --strict MyProject --groups Tests


./pharo metacello.image test --project-name MyProject
```

Package Verification in Isolation

Metacello command line installs SUnit + the library + its tests

```
./pharo metacello.image metacello install --strict SUnit --groups Core
./pharo metacello.image metacello install --strict MyProject --groups Tests
```

```
./pharo metacello.image test --project-name MyProject
```

Package Verification in Isolation

```
./pharo metacello.image metacello install --strict SUnit --groups Core

./pharo metacello.image metacello install --strict MyProject --groups Tests

./pharo metacello.image test --project-name MyProject
```

Test command line

Package Verification in Isolation

NEW! Strict mode. Rejects broken code

```
./pharo metacello.image metacello install --strict SUnit --groups Core

./pharo metacello.image metacello install --strict MyProject --groups Tests
```

```
./pharo metacello.image test --project-name MyProject
```

Package Verification in Isolation

NEW! Strict mode. Rejects broken code

```
./pharo metacello.image metacello install --strict SUnit --groups Core

./pharo metacello.image metacello install --strict MyProject --groups Tests

./pharo metacello.image test --project-name MyP
```

```
Metacello new
    baseline: 'MyProject';
    repository: '...';
    beStrict;
    load.
```

Also in Metacello scripts

Package Verification in Isolation

NEW! Project Convention!

Define a “*Tests*” group

```
./pharo metacello.image metacello install --strict SUnit --groups Core
./pharo metacello.image metacello install --strict MyProject --groups Tests

./pharo metacello.image test --project-name MyProject
```

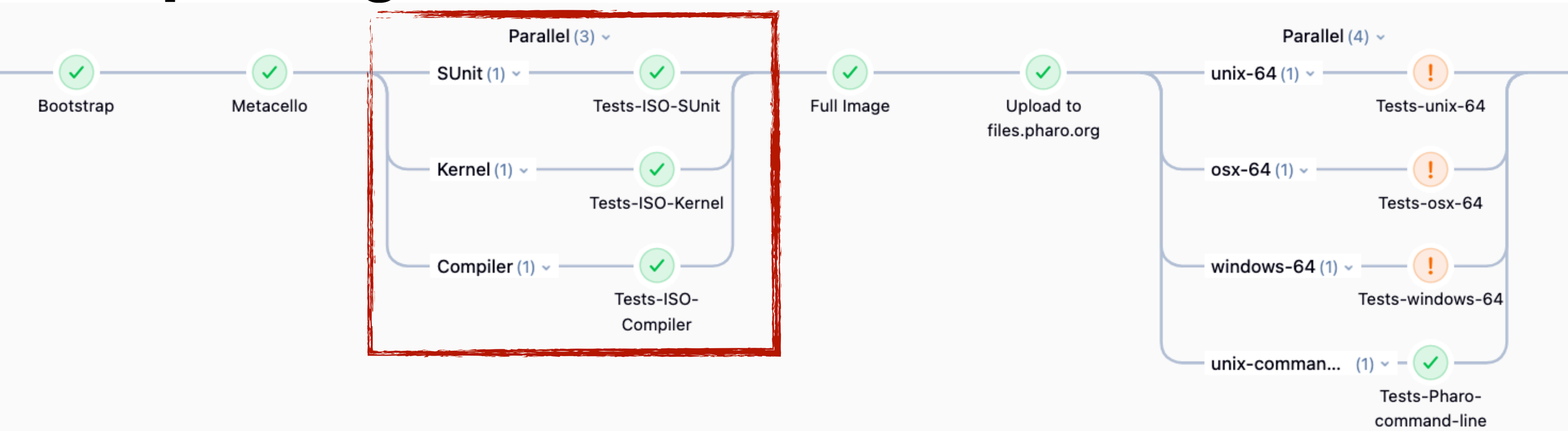
Package Verification in Isolation

```
./pharo metacello.image metacello install --strict SUnit --groups Core

./pharo metacello.image metacello install --strict MyProject --groups Tests

./pharo metacello.image test --project-name MyProject
```

NEW! Project Convention!

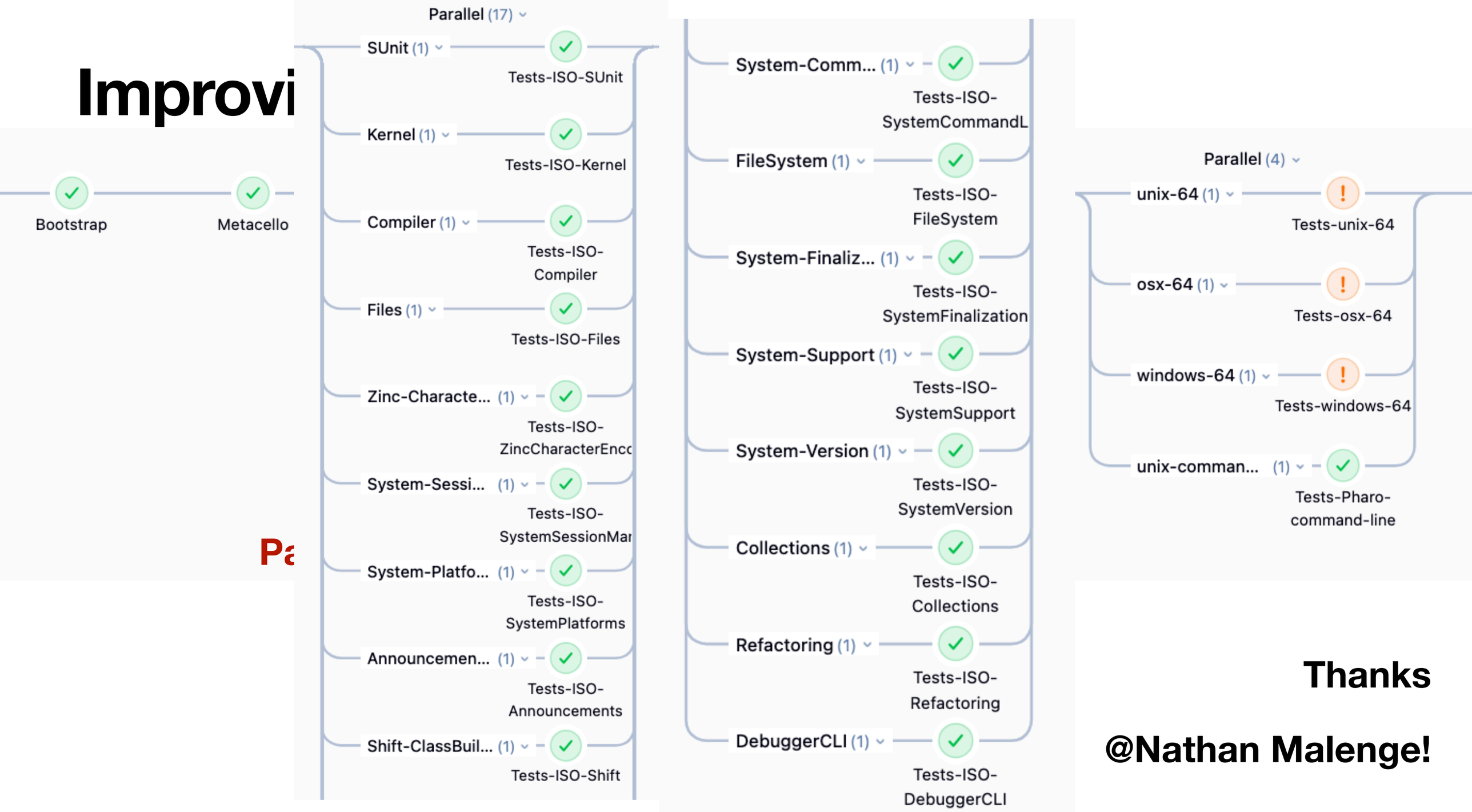
Define a “*Tests*” group, test are find from it!

Improving the CI Flow



Package verification in *isolation*

Improvi



Pa

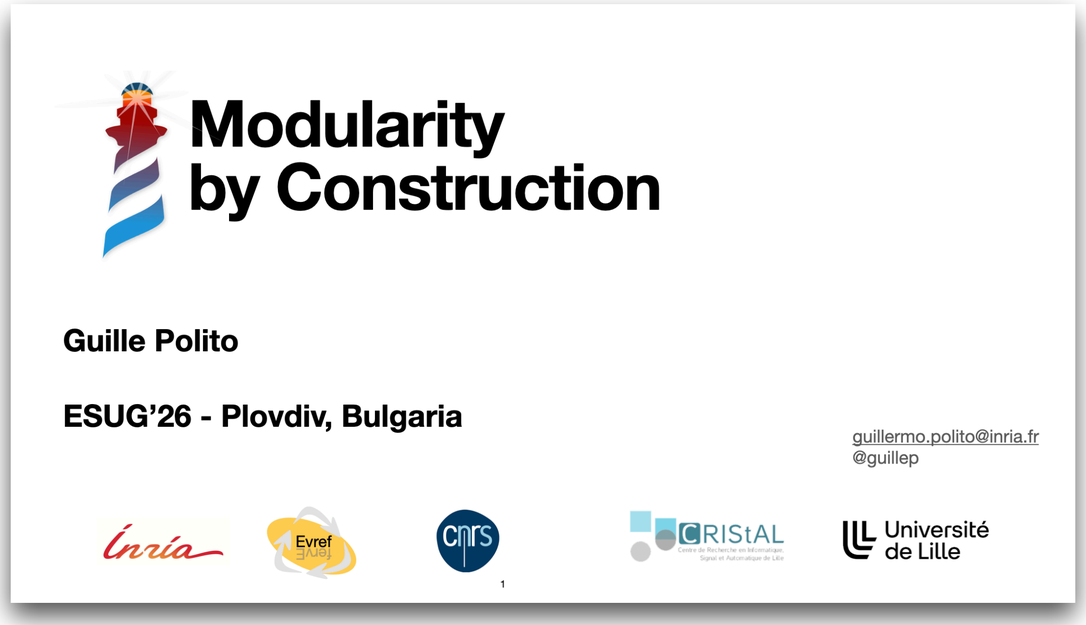
Thanks

@Nathan Malenge!

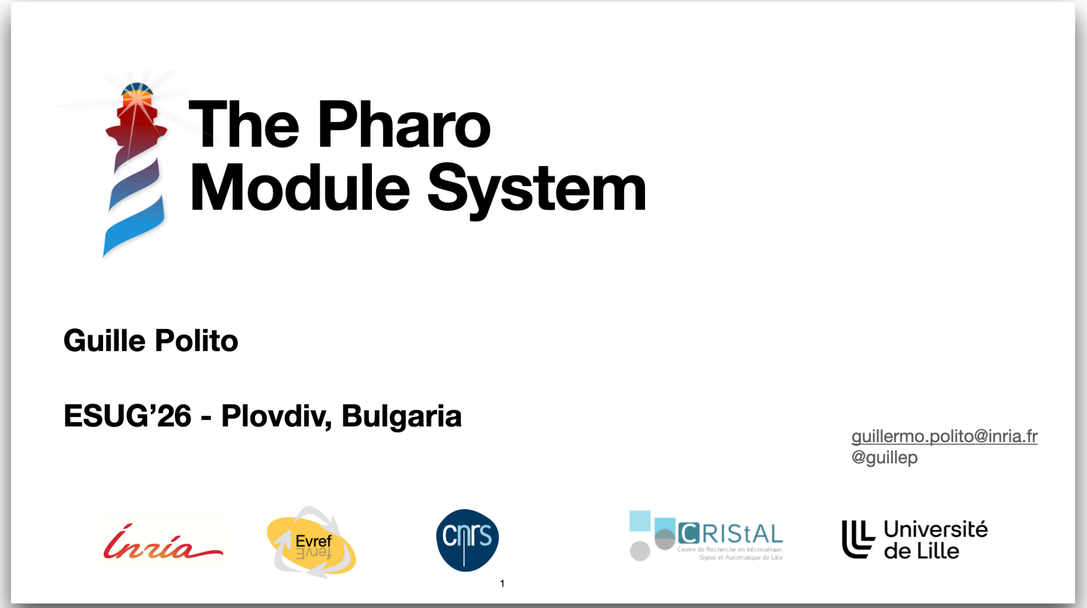
Behind the Scenes

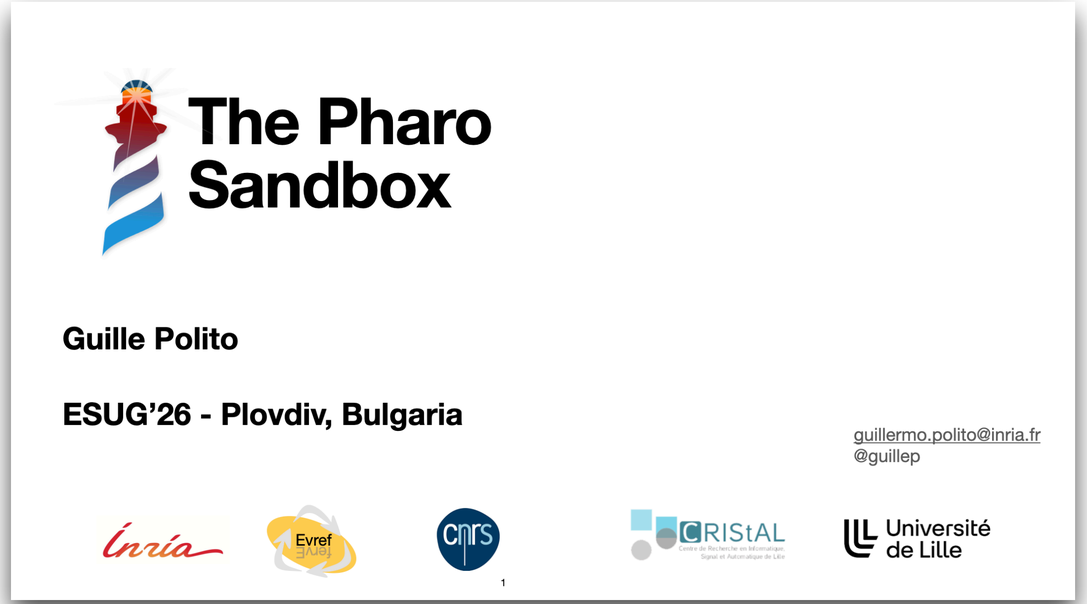
- The effort of many people
- Analysing dependencies
- Refactoring
- Redesign
- Fight with compatibility

Modularity Effort



+





The Sandbox

```
Sandbox do: [ :s |
    p := Point new.
    p setX: 17 setY: 11.
].
```

x	17
y	11

The Sandbox

```
Sandbox do: [ :s |
    p := Point new.
    p setX: 17 setY: 11.
].
```

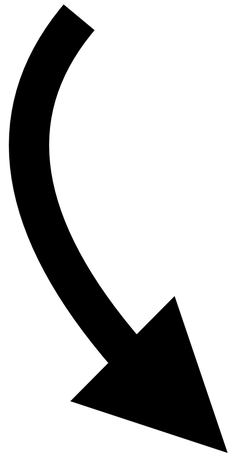
x	17
y	11

```
p := Point new.
Sandbox do: [ :s |
    p setX: 17 setY: 11.
].
```

The Sandbox

```
p := Point new.
Sandbox do: [ :s |
    p setX: 17 setY: 11.
].
```

```
setX: xValue setY: yValue

    x := xValue.
    y := yValue
```



```
writeSlot: slot on: object value: value
    (self owns: object) ifTrue: [
        ^ slot write: value to: object.
    ].

    self error
```

Sandboxing the Tests

```
runTestCaseUnderWatchdog: aTestCase

   [
      [aTestCase runCase] ensure: [
         "Terminated test is not considered as comp
for example)"
         mainTestProcess isTerminating ifFalse: [
            self handleCompletedTest ]]
   ] on: Exception do: [ :err |
         self handleException: err
   ]
```

```
runTestCaseUnderWatchdog: aTestCase

   [
      [

         Sandbox do: [ :sb |
               sb own: aTestCase.
               aTestCase runCase ]

      ] ensure: [
         mainTestProcess isTerminating
            ifFalse: [ self handleCompletedTest ] ] ]
      on: Exception
      do: [ :err | self handleException: err ]
```

The Goal

- **Module system + strict loading/linking**

  => Static dependency control and verification

- **Conventions**

  => Clear and simple rules to build complex software

- **Sandbox**

  => Runtime isolation guarantees

Some things I learned (or I reinforced)

- **“always relaxed” systems:**
  - helps yielding **fragile designs**
  - Technical debt cumulates

- **I want EVERYTHING:** “relaxed” at dev time, “strict” in production

- **About Feedback:** should be *fast* and *precise*

Some things I learned

- Sandbox exposes *design smells*
  - Writing to shared singletons
  - Strange designs: `aRegex matchingRangesIn: '…'` has side effects!!
  - Libraries writing to the global Morphic progress bar
  - Bad exception handling :)

Some things I learned (II)

- Emergence of lots of **interesting language design decisions**
- Lot of Pharo “system services” depend on side effects

  - #symbol, ‘string’ => should tests pollute the symbol table?
  - 1 second wait => should tests affect the scheduler?
  - FinalizationRegistry default add: … => should tests affect the finalization?
  - Class compile: ‘asd’ => installing code, undeclared, AST caches…

Some things I learned (III)

- Emergence of lots of **interesting language design decisions**
- What about I/O
  - Should FFI be sandboxed?
  - Files? Network?
  - Primitives? Which primitives?

What’s Next?

- **More Isolated Library Testing:** more redesign, dependency cutting…
- **More Redesign:** sandbox exposes design issues, both in runtime and libs
- **Better Debug Support:** command line debugger and profiler, @Nathan
- **Automatic Migration:** to new scoped extensions @Nathan

Modularity by Construction

- **Module system + strict loading/linking**

  => Static dependency control and verification

- **Conventions**

  => Clear and simple rules to build complex software

- **Sandbox**

  => Runtime isolation guarantees

guillermo.polito@inria.fr
@guillep