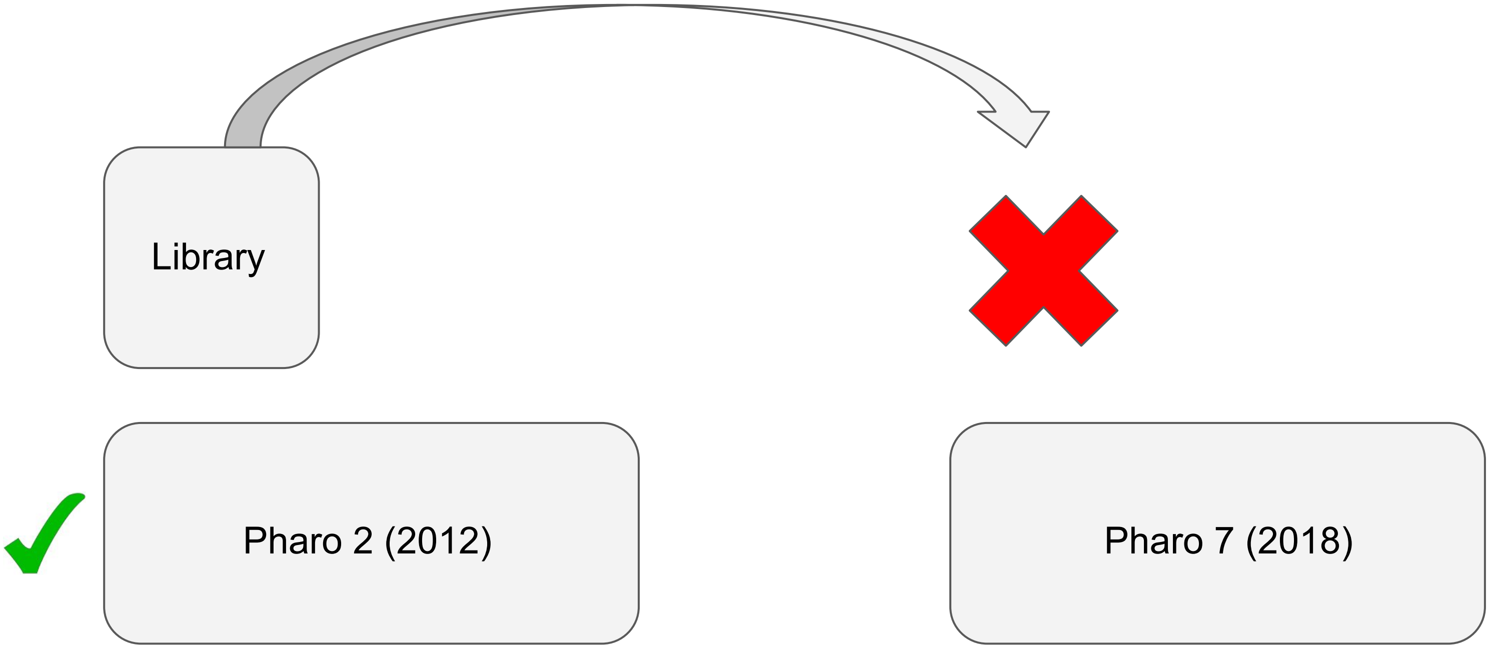
Towards easy program migration using language virtualization

Théo Rogliano, Pablo Tesone, Guille Polito

Agenda

1. Motivation: reusing old libraries in new versions of the language
2. Our approach: Virtualization-inspired language compatibility
3. Techniques for language virtualization: overcoming the challenges
4. Validating our virtualization approach
5. Future and conclusion

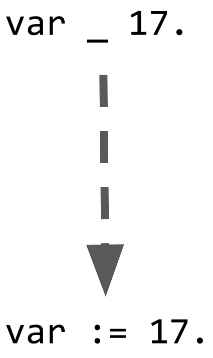
Reusing libraries between two versions of the language



Problems of old libraries in new language versions

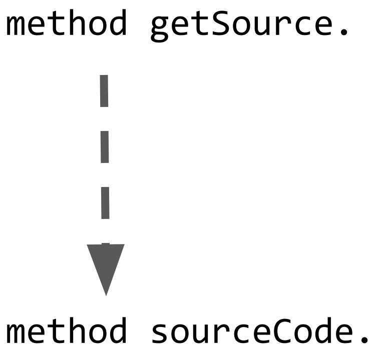
Examples of language changes that break programs:

- Syntax changes.
- Standard library changes (public classes, APIs).
- Other examples: Meta-model changes, compiler semantics changes.



```
var _ 17.
```

```
var := 17.
```

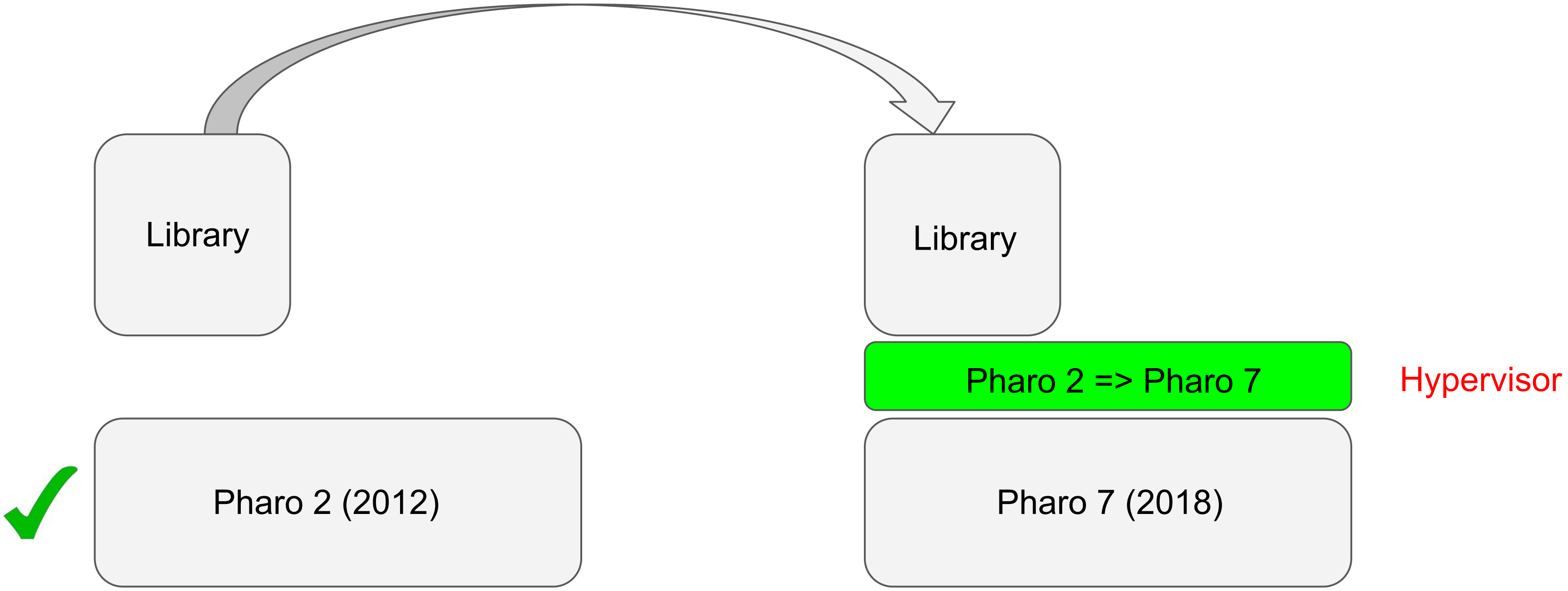


```
method getSource.
```

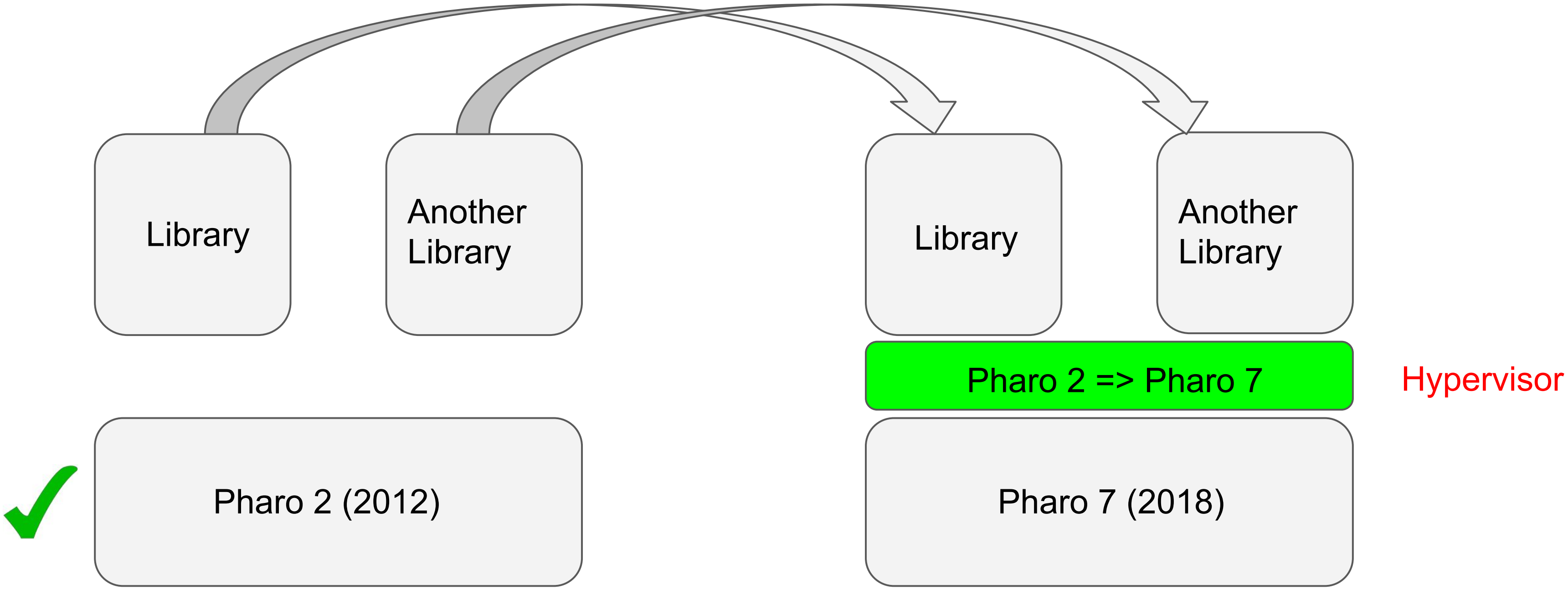
```
method sourceCode.
```

Our approach

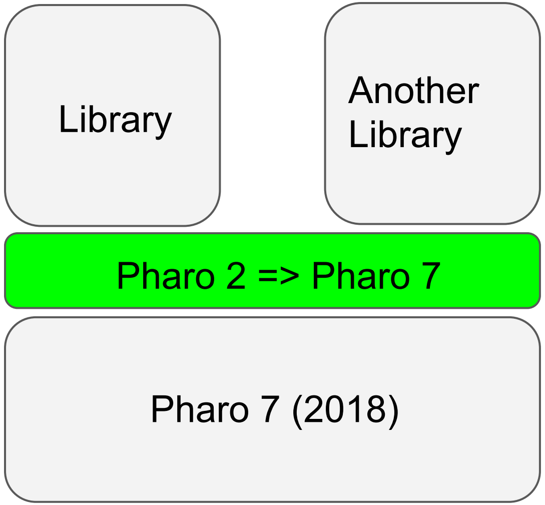
Virtualization-inspired language compatibility



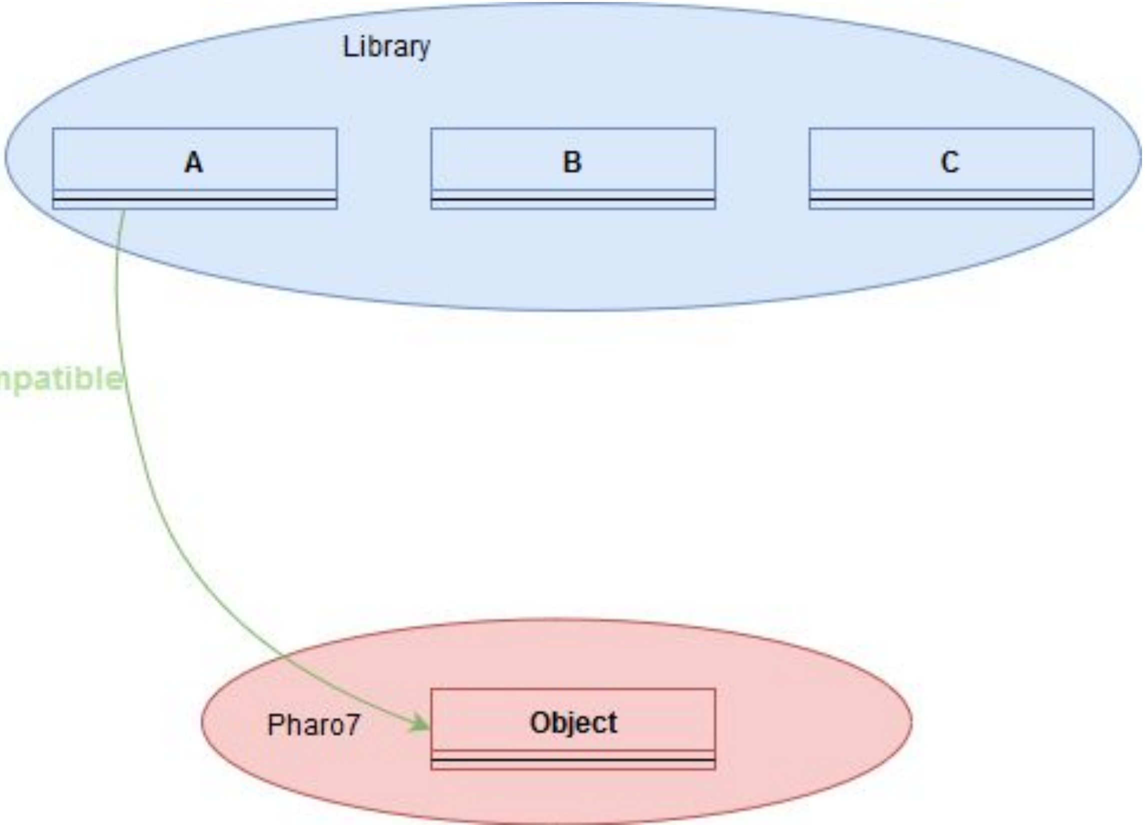
Reusable compatibility layer



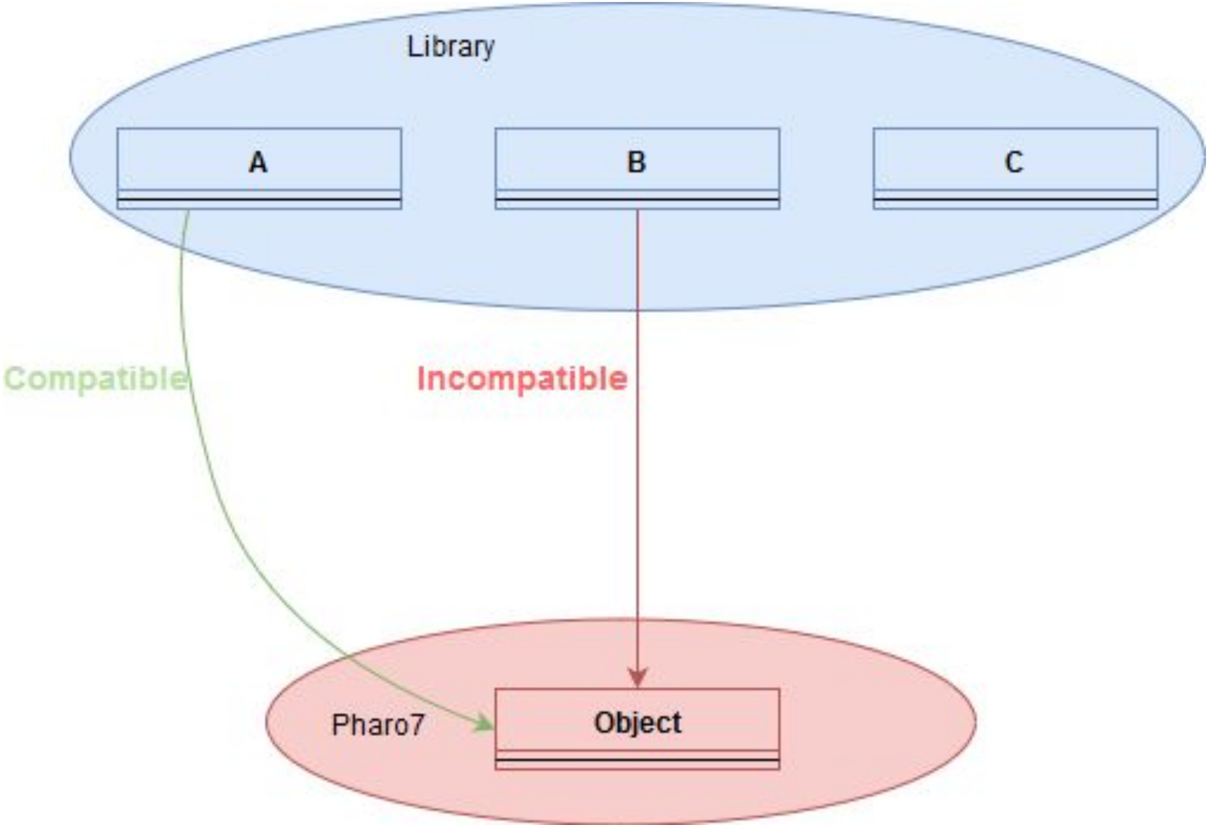
Research question: how do we build a compatibility layer ?



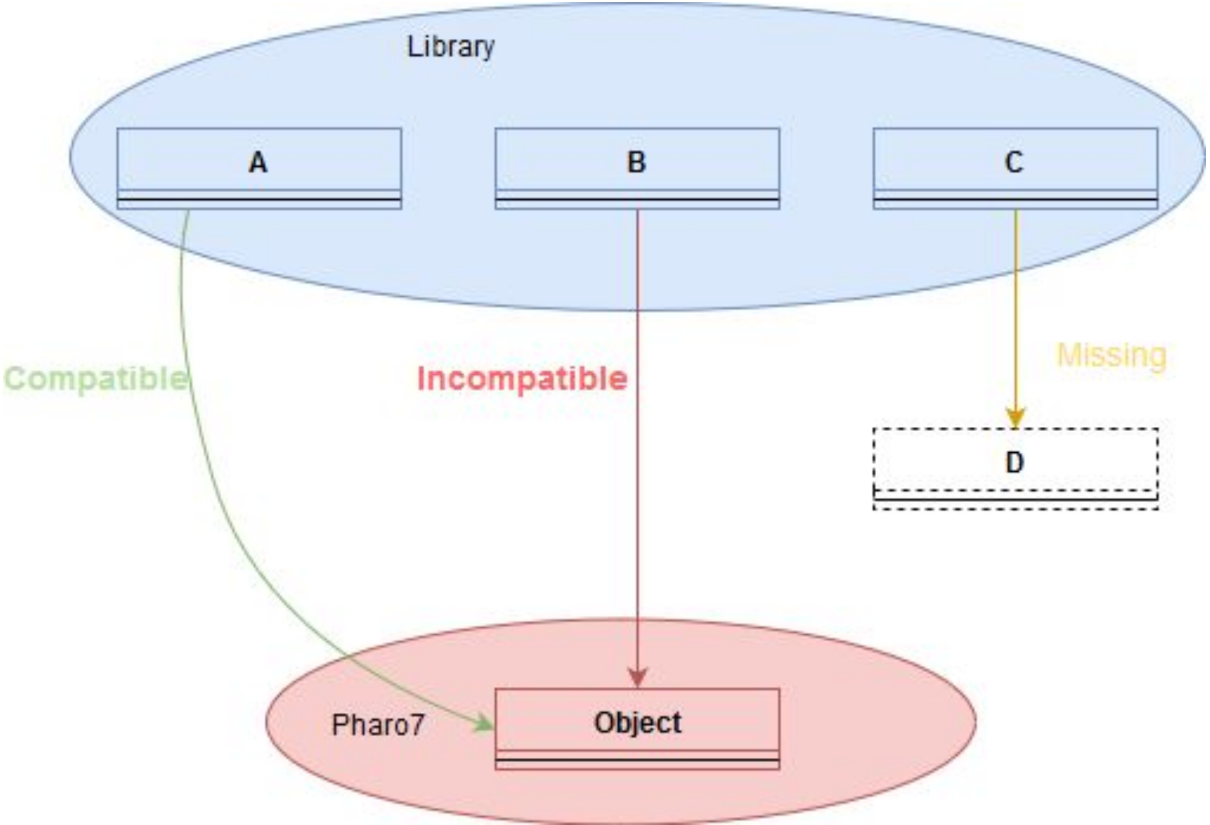
Challenges of language virtualization



Challenges of language virtualization



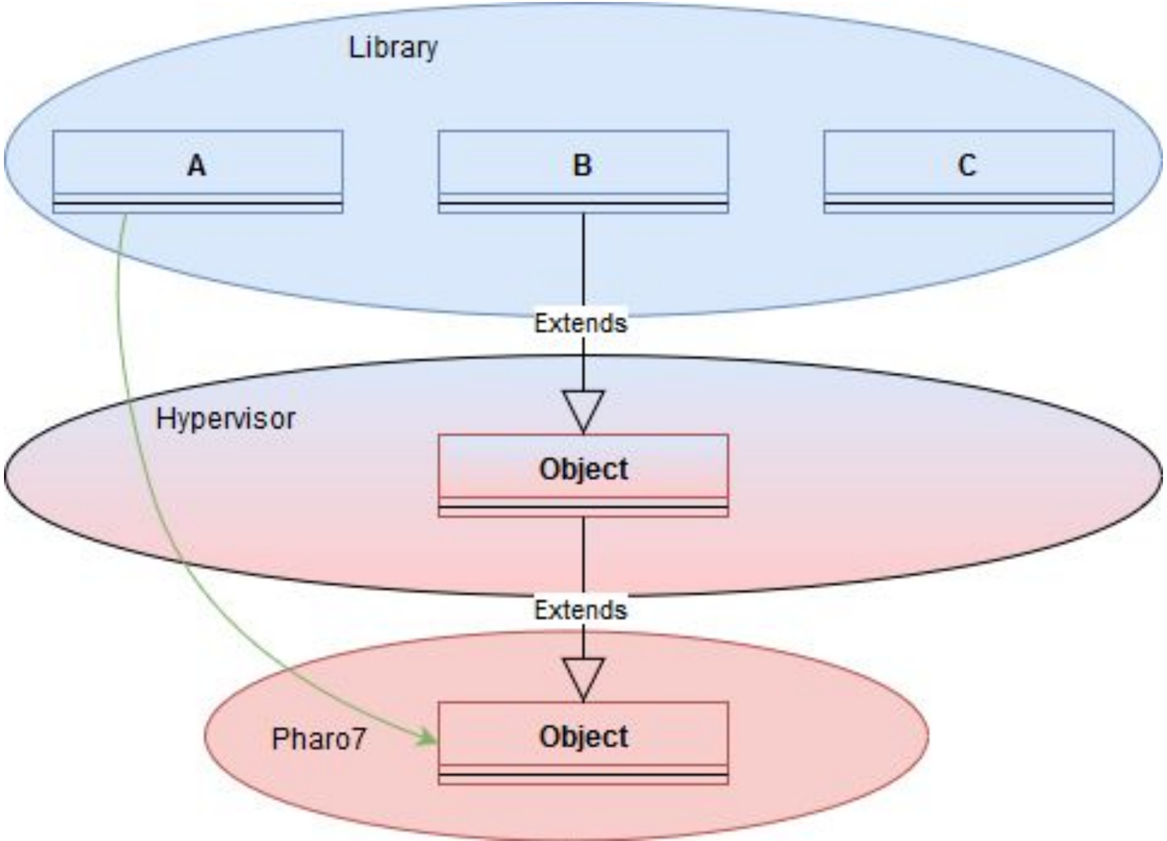
Challenges of language virtualization



Techniques for language virtualization

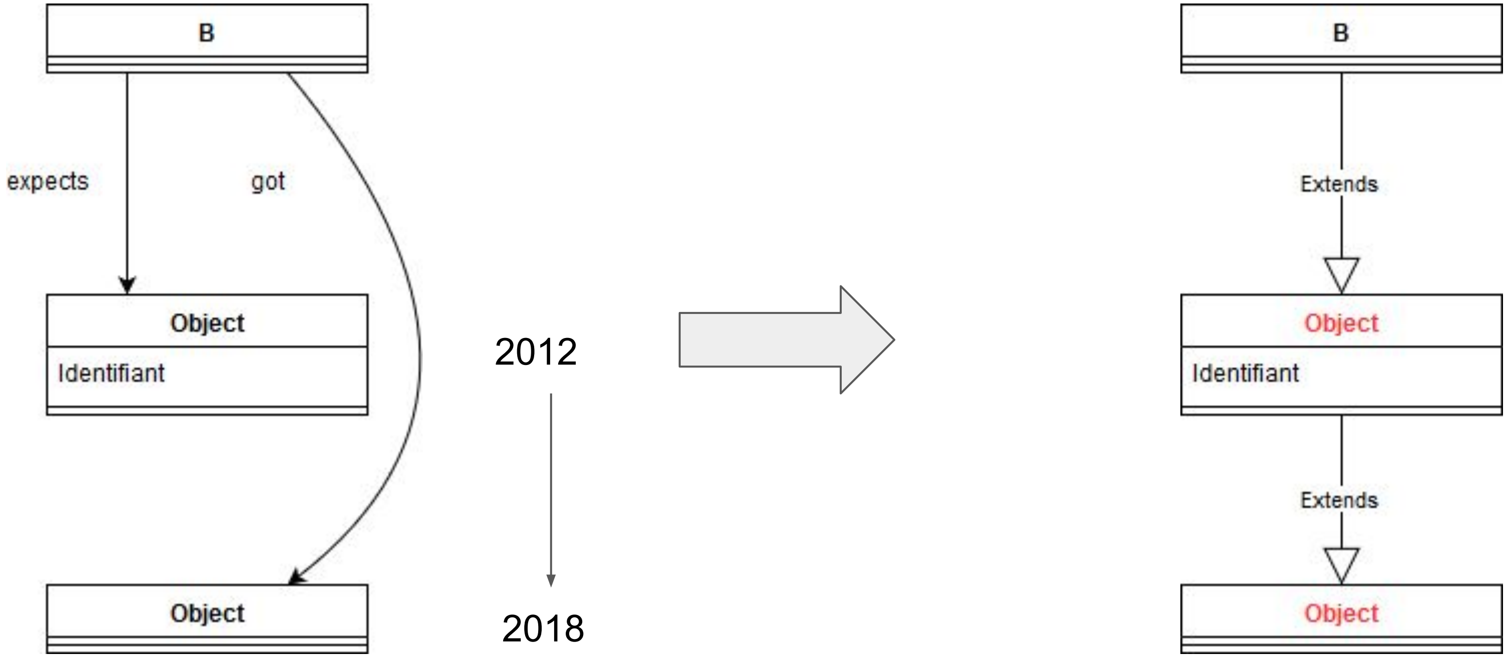
- Kernel indirection.
- Dynamic code rewriting
- Modules for isolation

Kernel Indirection

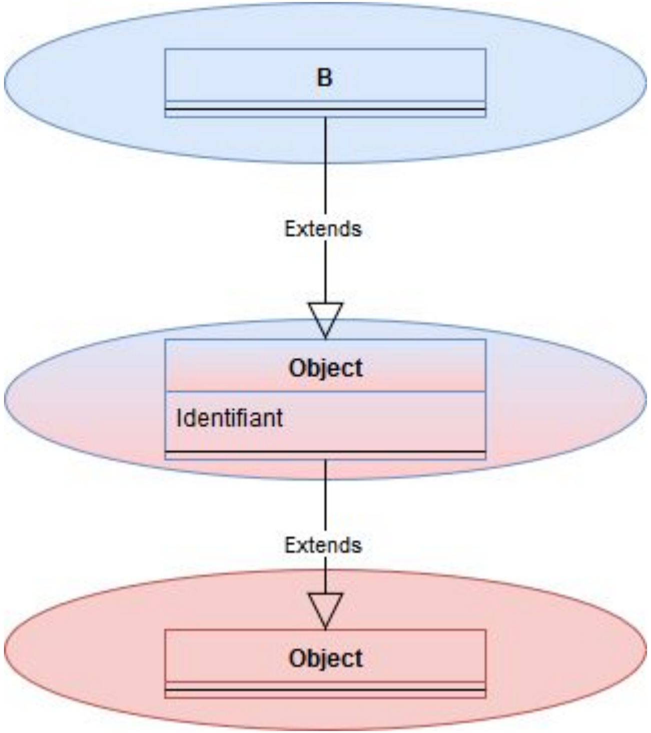


Entity with the same name we expect exists, we reuse it with inheritance.

Solving incompatibilities with inheritance

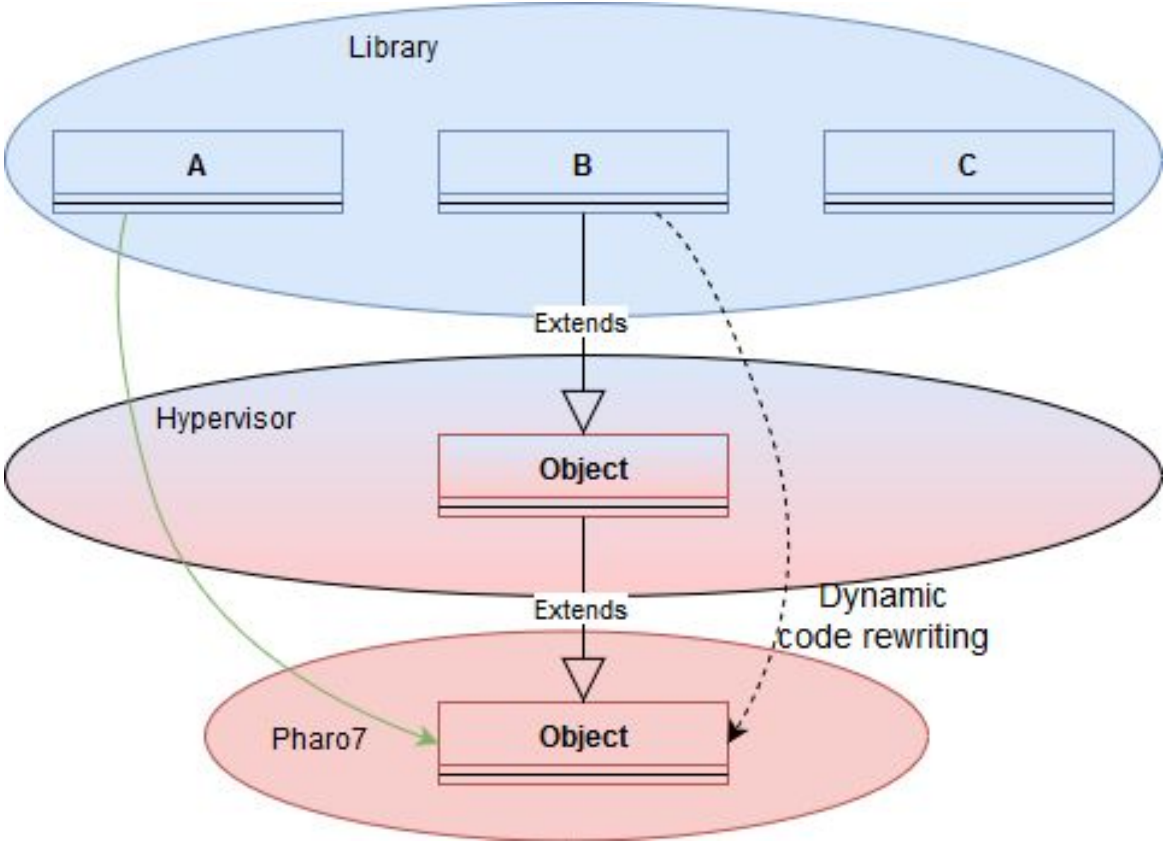


Solving name conflicts with modules



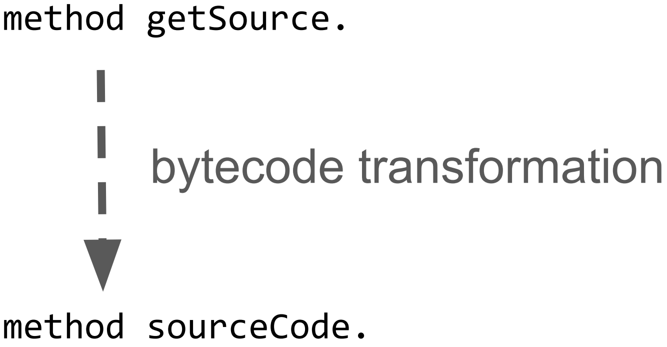
Same name allowed in different modules.

Dynamic code rewriting



Kernel indirection is not enough.

Solving incompatibilities through code rewritings



```
method getSource.
```

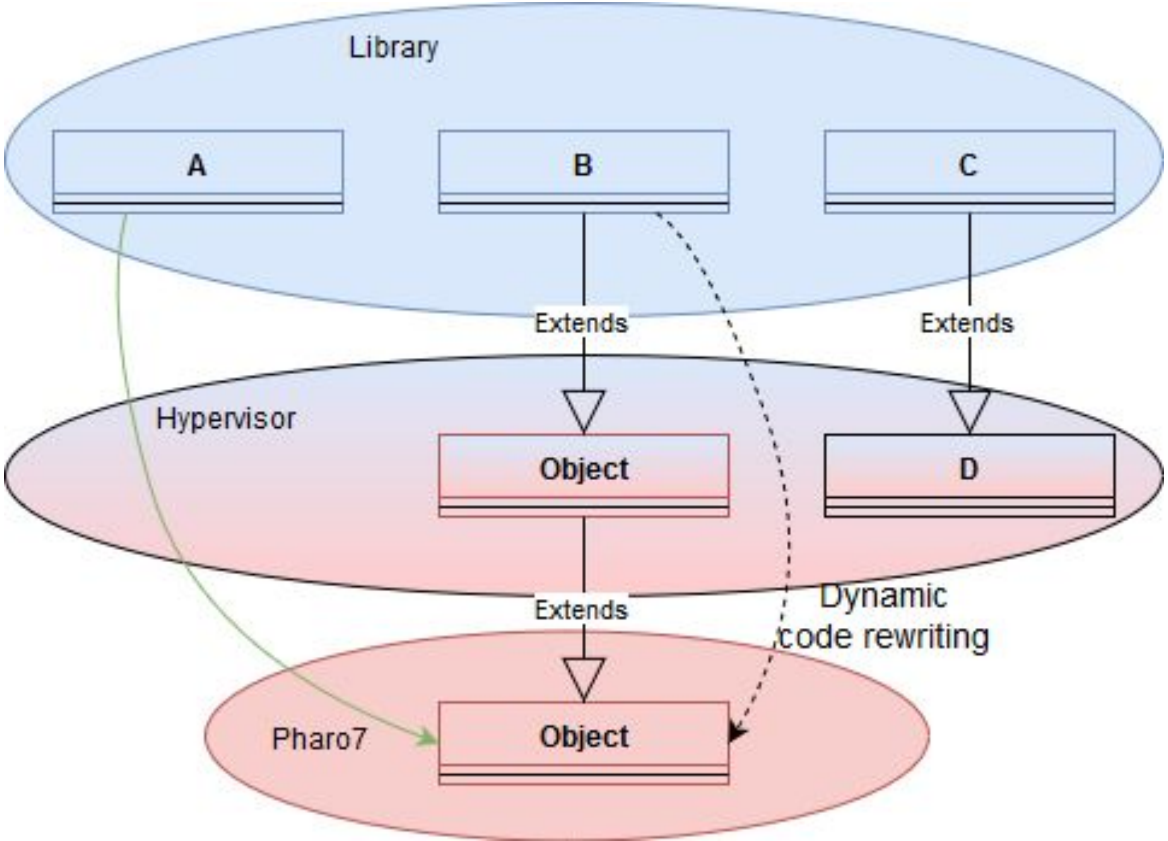
bytecode transformation

```
method sourceCode.
```

Transparent for the library.

Rewriting done through AST annotations.

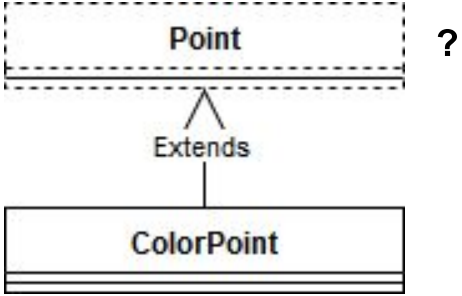
Retrieve missing behavior



-Retrieve old behavior if we have an archive.

-Assume behavior and code it (TDD).

Solving missing classes with late class creation



ColorPoint cannot be created without its superclass Point.

Introduced a class AST to analyse it before class creation

-Detection of missing references.

-React to the missing references (modify the class creation).

Validation

We execute old Pharo programs in a newer version of Pharo with a hypervisor.

Hypothesis: the program had all tests passing in the old version.

Goal: make those tests pass in the newer version with the hypervisor.

/!\ It does not mean the program is working (low test coverage)

Scenario1: Mutalk

Virtualizer strategies created with this scenario.

✗ -45 missing entities

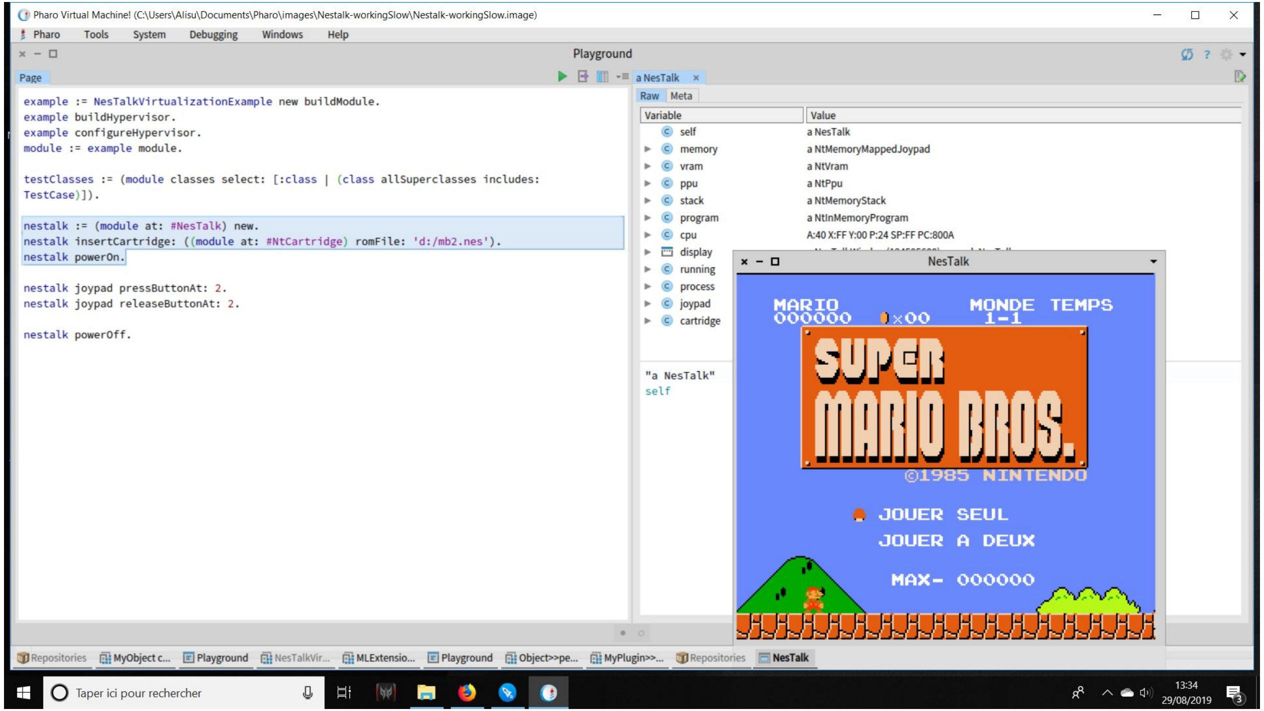
⇨ -37 reimplemented in hypervisor

✓ -355/362 tests passing

-8 remainings missing entities are for graphical behaviors or not tested

-7 failing tests are linked to the 8 remainings missing entities

Scenario (Bonus): NesTalk



Pharo Virtual Machine! (C:\Users\Alisu\Documents\Pharo\images\Nestalk-workingSlow\Nestalk-workingSlow.image)
Pharo
Tools
System
Debugging
Windows
Help
Playground
Page
a NesTalk
Raw
Meta

```
example := NesTalkVirtualizationExample new buildModule.
example buildHypervisor.
example configureHypervisor.
module := example module.

testClasses := (module classes select: [:class | (class allSuperclasses includes:
TestCase)]).

nestalk := (module at: #NesTalk) new.
nestalk insertCartridge: ((module at: #NtCartridge) romFile: 'd:/mb2.nes').
nestalk powerOn.

nestalk joypad pressButtonAt: 2.
nestalk joypad releaseButtonAt: 2.

nestalk powerOff.
```

Variable	Value
self	a NesTalk
memory	a NtMemoryMappedJoypad
vram	a NtVram
ppu	a NtPpu
stack	a NtMemoryStack
program	a NtInMemoryProgram
cpu	A:40 X:FF Y:00 P:24 SP:FF PC:800A
display	
running	
process	
joypad	
cartridge	

```
"a NesTalk"
self
```

Scenario 2: Fuel

✗ -79 missing entities

→ -67 reimplemented in hypervisor + stream compatibility

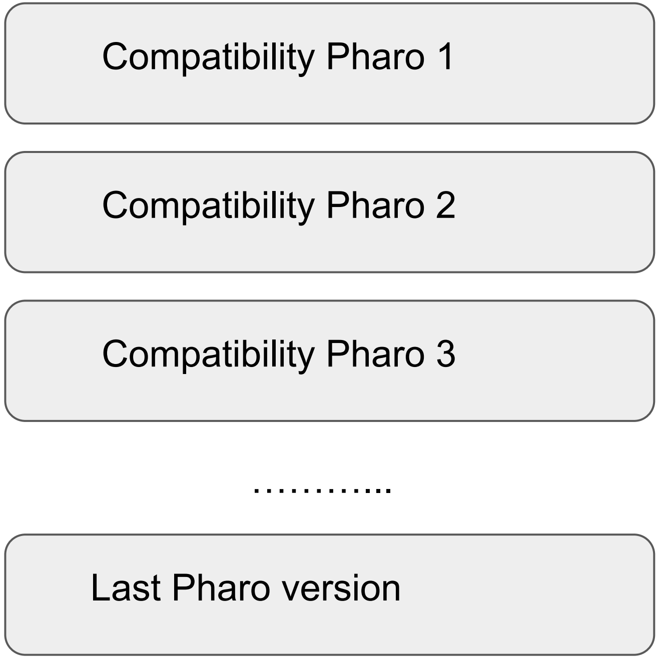
✓ -19/239 tests passing

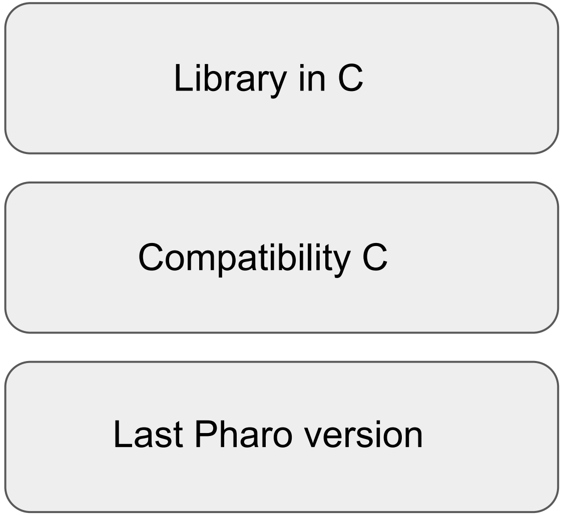
We encountered new challenges:

- Fuel assumes a single global environment
- The compatibility layer is not hidden to reflective operations
- Extension methods needs to be scoped to the compatibility layer or library

Future work

Compatibility layer superposition?





A compatibility layer with another language?

Relation with PharoGs?

Conclusion

Language changes cause compatibility problems.

We propose a compatibility layer between different pharo versions.

We validate our approach by running old applications in new versions.

We discover new challenges to overcome.

Questions ?

Théo Rogliano