Challenges in Debugging Bootstraps of Reflective Kernels

Carolina Hernández Phillips
INRIA Lille Nord Europe
IMT Lille Douai

Stéphane Ducasse
INRIA Lille Nord Europe

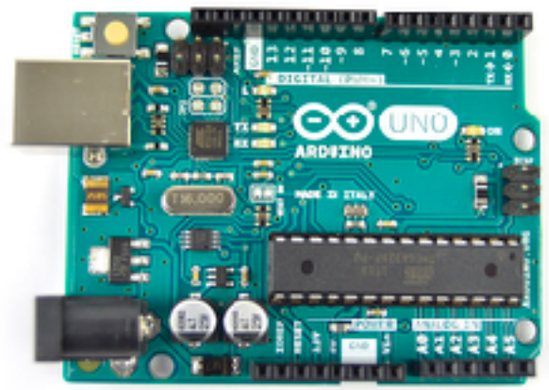
Luc Fabresse
IMT Lille Douai

Noury Bouraqadi
IMT Lille Douai

Guille Polito
Univ. Lille, CNRS, CRIStAL

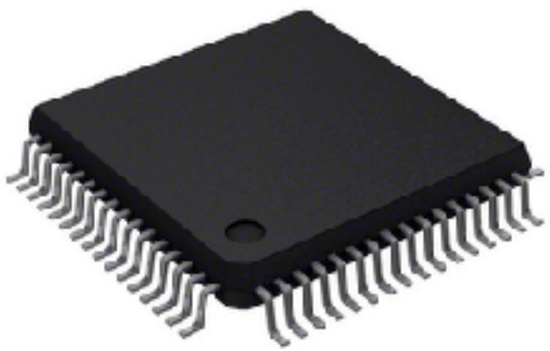
Pablo Tesone
Pharo Consortium

Why generating custom application runtimes for IoT?



Small Hardware requires software

Limited processing capabilities, storage, battery



Existing approaches: Generating lightweight implementations of Languages from scratch

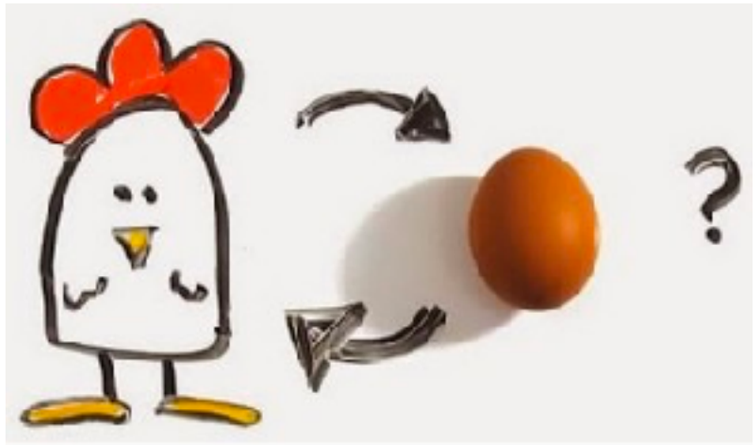
Implement from scratch: VM, base libraries, compiler

Implies complex low level implementation

Requires high expertise to develop!

Our high level approach: Bootstrapping reflective kernels

- Bootstrapping is to **generate a system using a previous version of the system** that is being generated



- Therefore we can use the **high level abstractions** and the **reflective capabilities** of both systems during the bootstrap

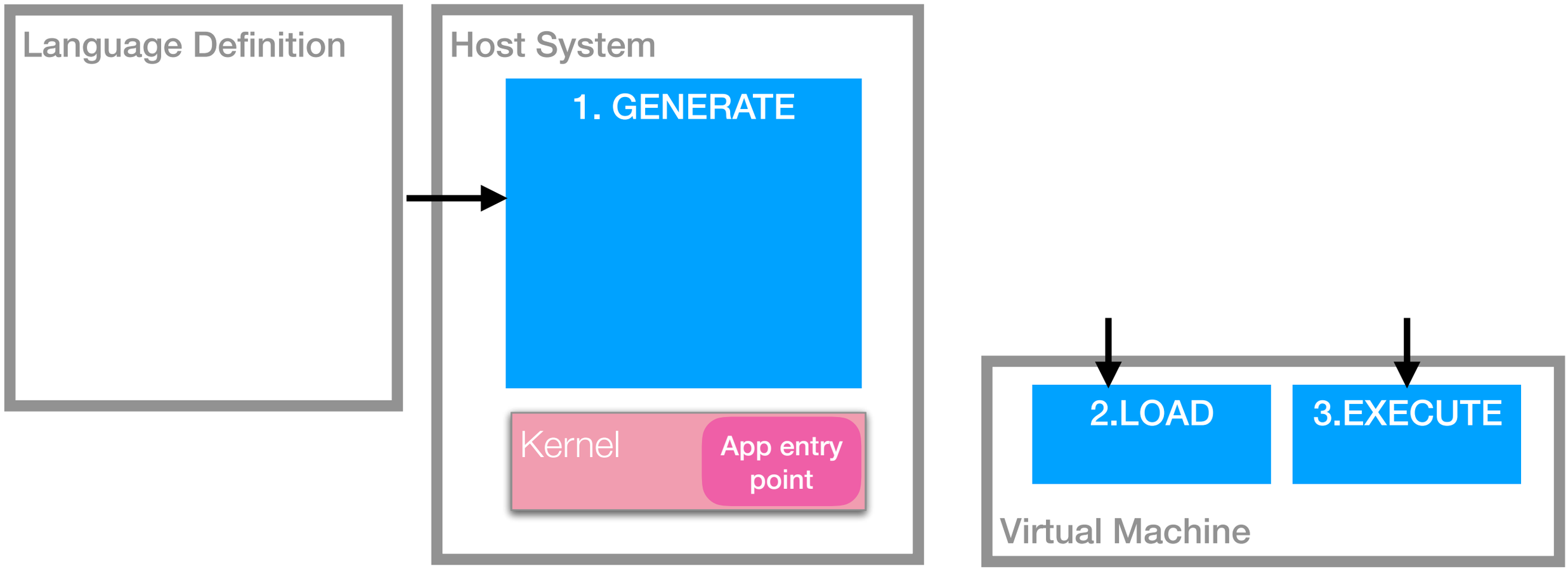
- The result is a **small Kernel** (an image in the case of Pharo) which can be executed by the same VM that executes its previous version

Demo

Let’s Bootstrap PharoCandle

(a Pharo micro kernel)

Bootstrap

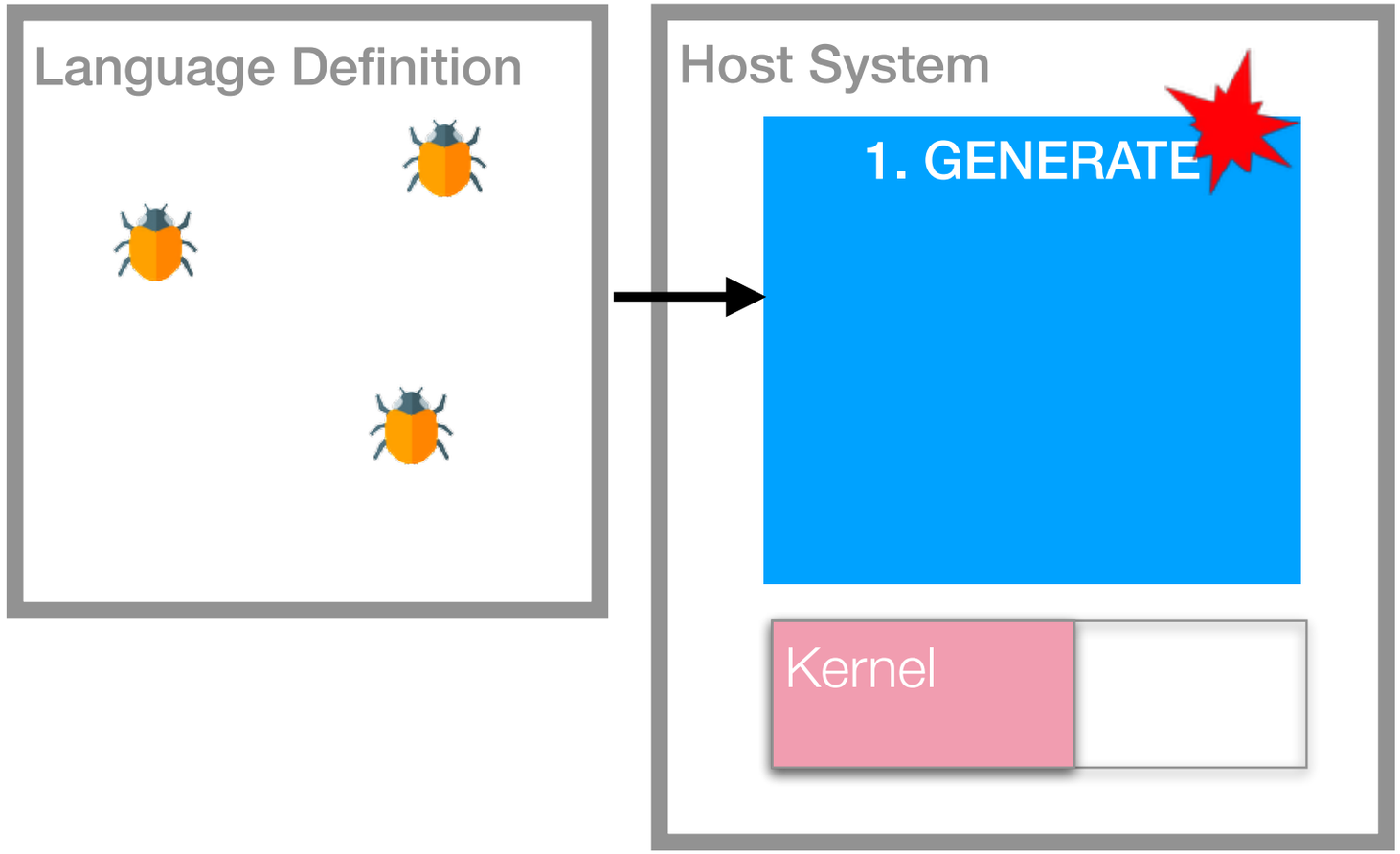

Language Definition
Host System
1. GENERATE
Kernel
App entry point
2.LOAD
3.EXECUTE
Virtual Machine

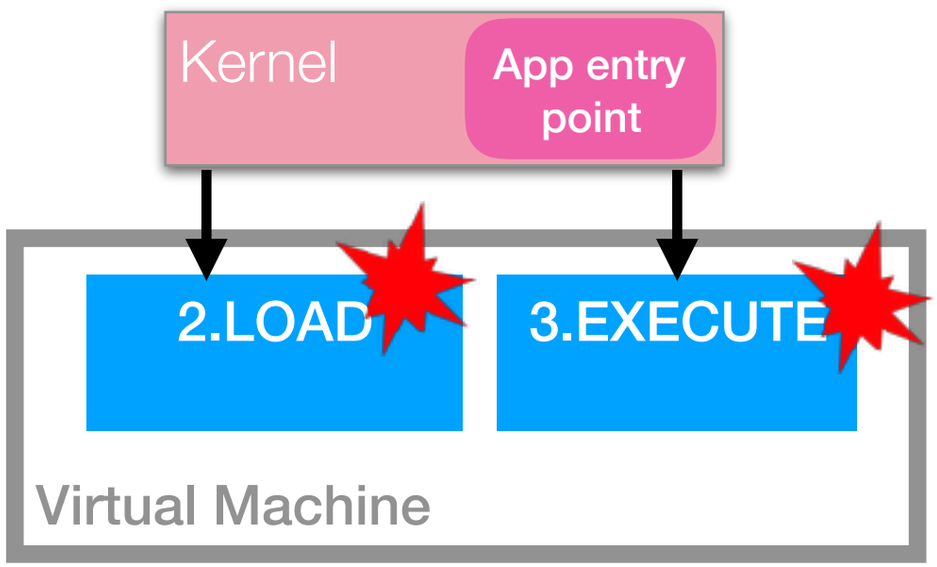
Defects and Failures

Defects & Failures

Defect: error in Language Definition

Failure: incorrect result during the Bootstrap





Defects Classification

Defect: error in Language Definition

Language Definition

- Structural Definitions of Classes
- Methods Definitions

Class **PCPoint**

superclass : PCObject,
instVars : { ‘x’, ’y’ },
type : ~~variable~~ fixed

Structural Defect

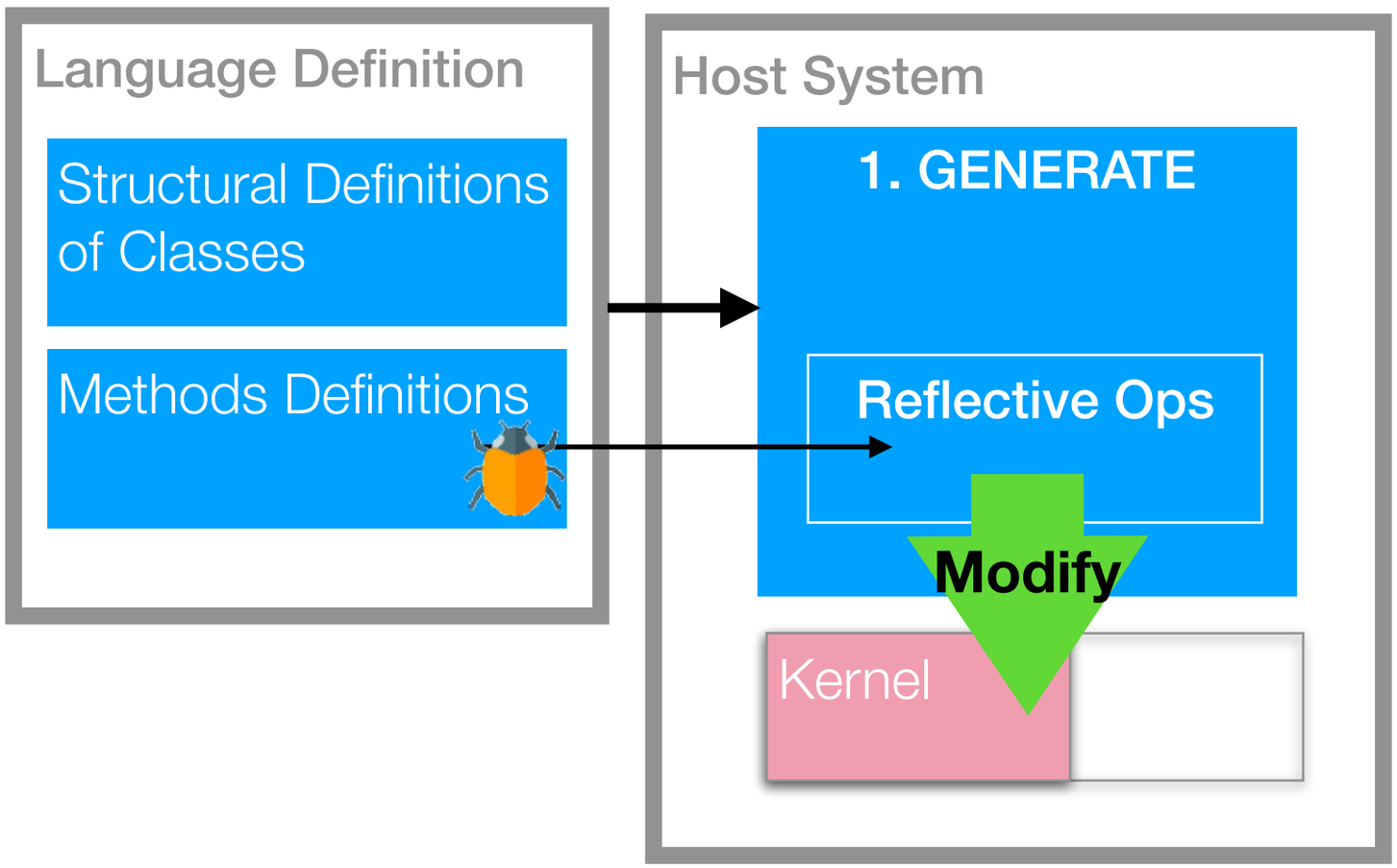
```
PCPoint >> + arg {
      ^ (x + arg x) @ (y + arg y)
}
PCPoint >> crossProduct: aPoint {
      ^ x * aPoint y - (y * aPoint x)
}
...
```

x

Semantic Defect

Semantic Defects are Dangerous

Defect: error in Language Definition



Semantic Defects in

reflective methods

modify the structural

definitions in the Kernel

```
PCClassBuilder >> installMethod: aCompiledMethod inClass: aClass {

   aClass methodDictionary add: aCompiledMethod

}
```

The why of defects

Defects

Defect: error in Language Definition

Language Definition

- Structural Definitions of Classes
- Methods Definitions

Class **PCPoint**

superclass : PCObject,
instVars : { ‘x’, ’y’ },
type : ~~variable~~ fixed

Structural Defect

```
PCPoint >> + arg {
      ^ (x + arg x) @ (y + arg y)
}
PCPoint >> crossProduct: aPoint {
      ^ x * aPoint y - (y * aPoint x)
}
...
```

x

Semantic Defect

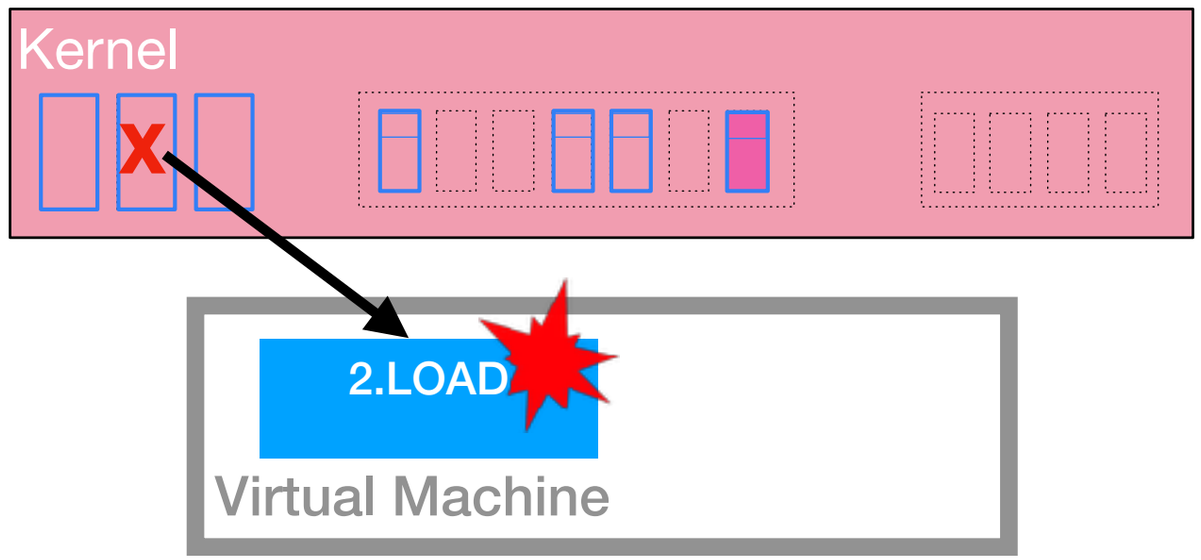
The why of Defects

- Virtual Machine requirements

```
Class PCArray

    superclass : PCObject,
    instVars : { },
    Type : variable
```

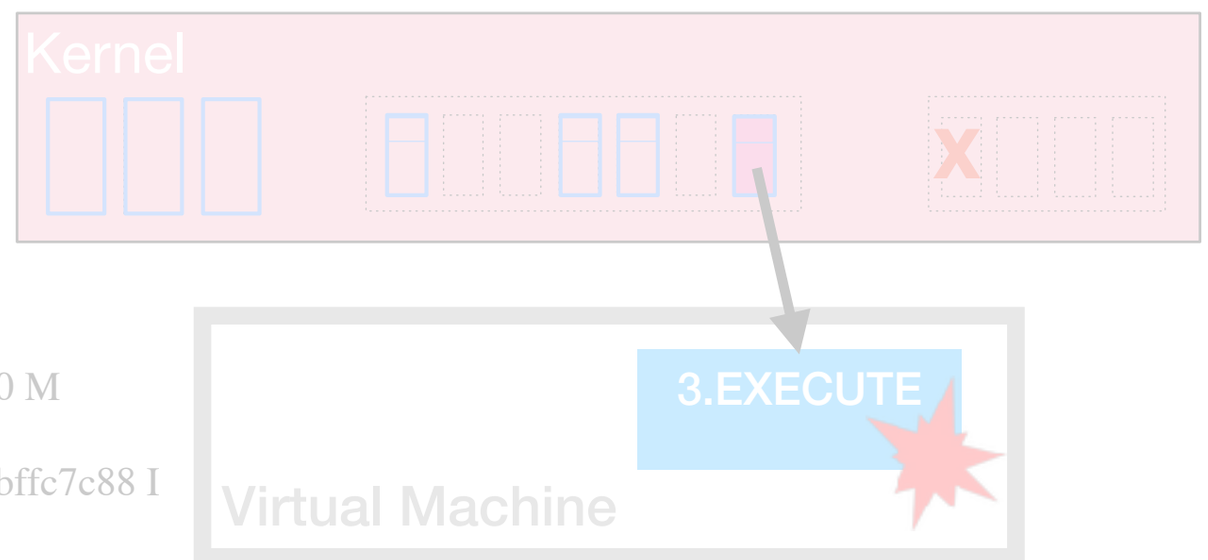
Segmentation Fault



- Application requirements

```
PCMainApplication >> entryPoint {
    PCMyClass doSomething
}
```

Smalltalk stack dump: 0xbffc8fd0 M
>species 0x6e4e350: a(n) bad class 0xbffc7c0c M
>copyReplaceFrom:to:with: 0x6e4e350: a(n) bad class 0xbffc7c30 M
>, 0x6e4e350: a(n) bad class 0xbffc7c5c I
>doesNotUnderstand: activeProcess 0x6e2f7c0: a(n) bad class 0xbffc7c88 I
>doesNotUnderstand: activeProcess 0x6e2f7c0: a(n) bad class



Why is it hard to find the defects back?

- We are **debugging the VM**

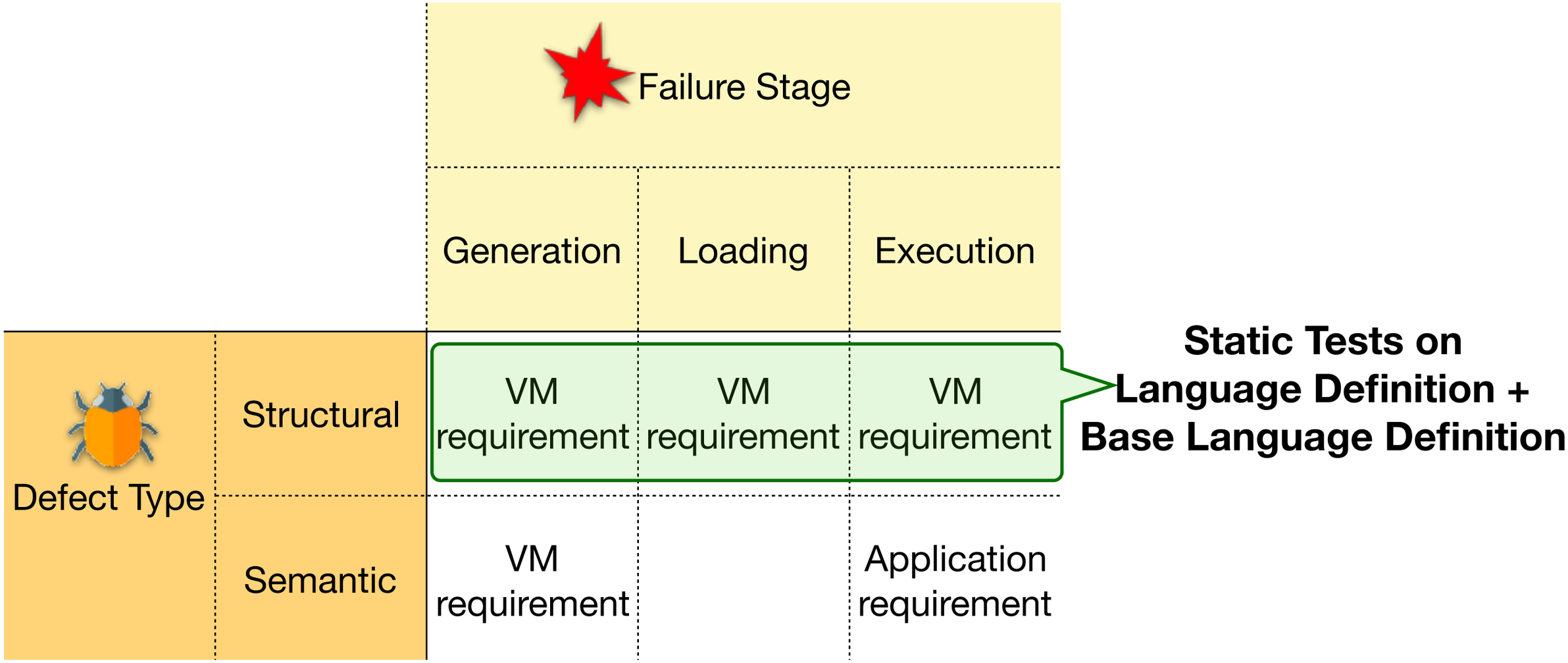
- We **lose great part of the abstractions** of the generated language

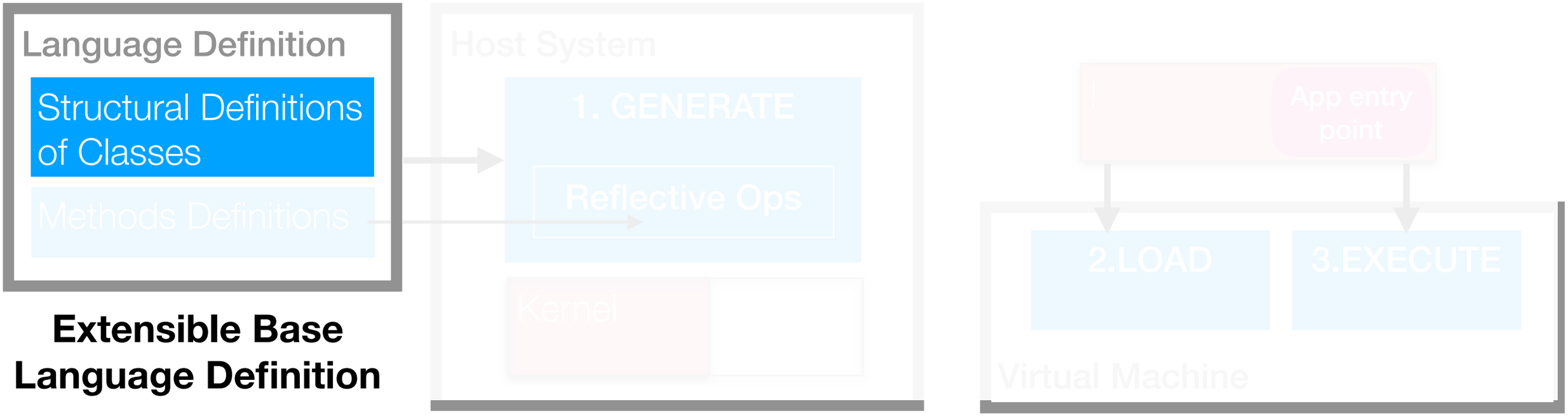
Taxonomy of Errors and proposed Solutions

Taxonomy of Errors and Solutions

		Failure Stage		
		Generation	Loading	Execution
Defect Type	Structural			
	Semantic			

Taxonomy of Errors and Solutions

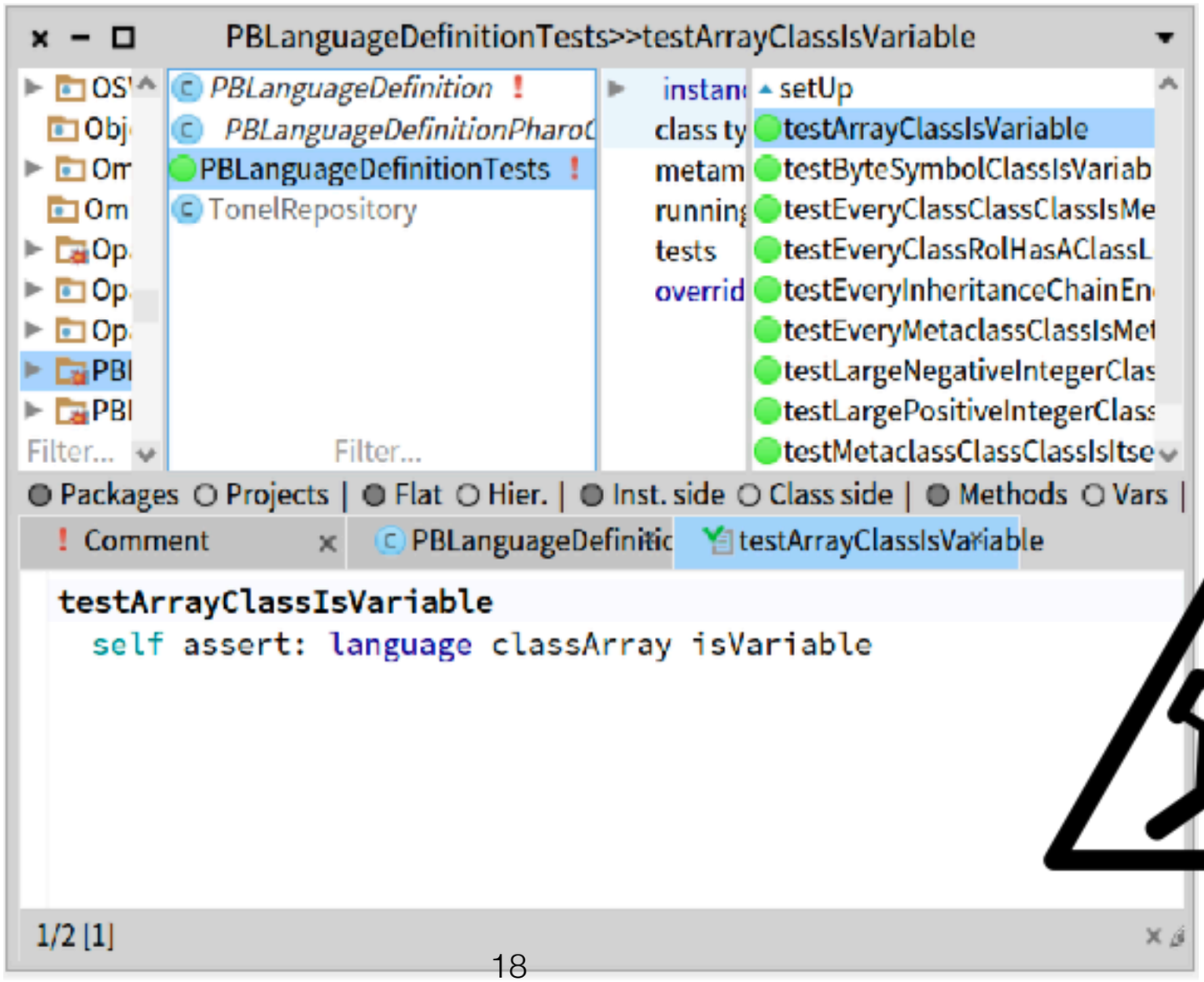




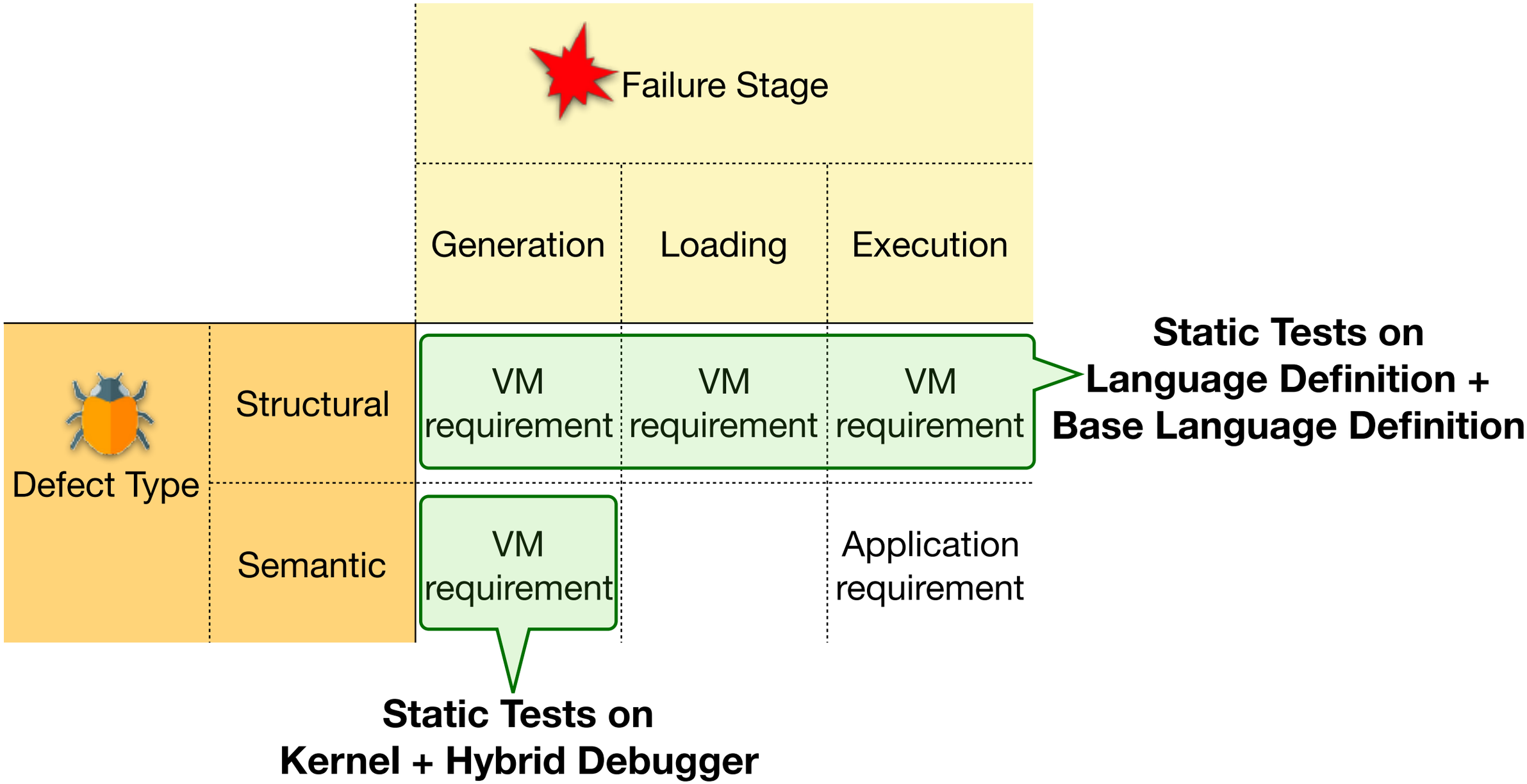
Extensible Base Language Definition

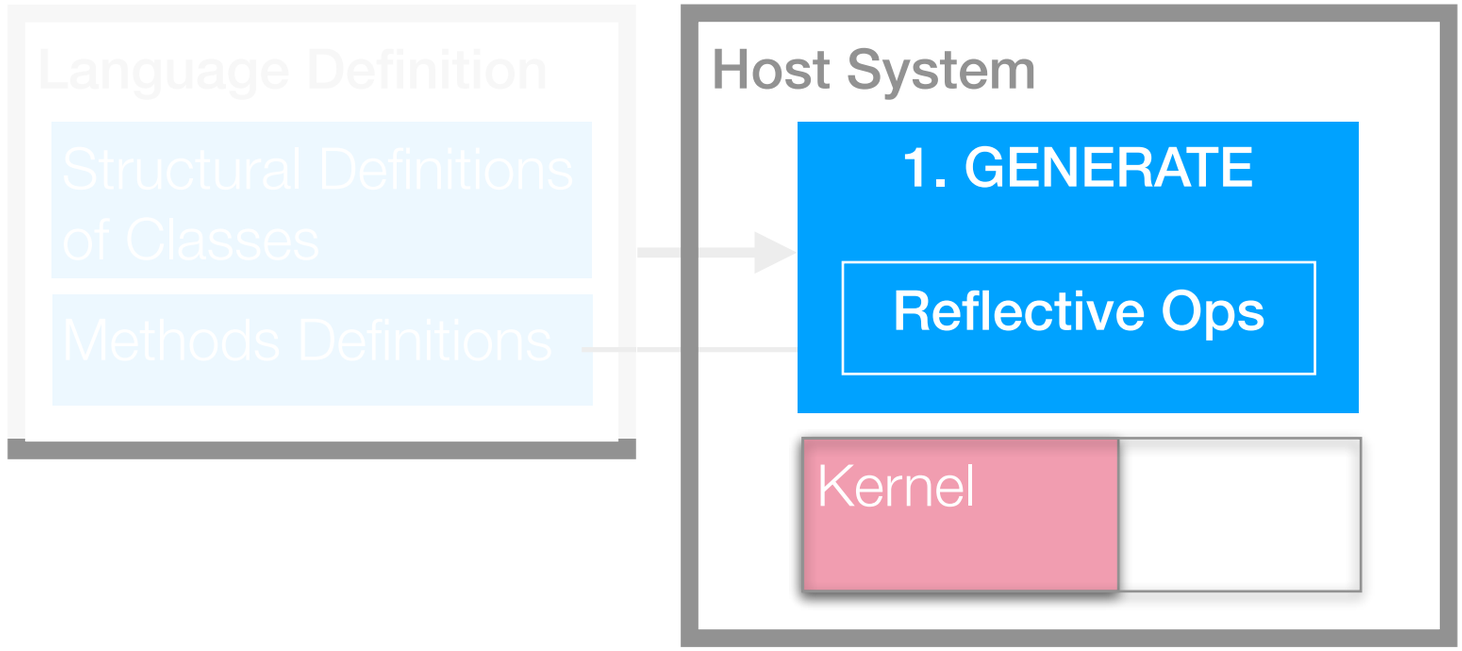
Static Tests on Language Definition

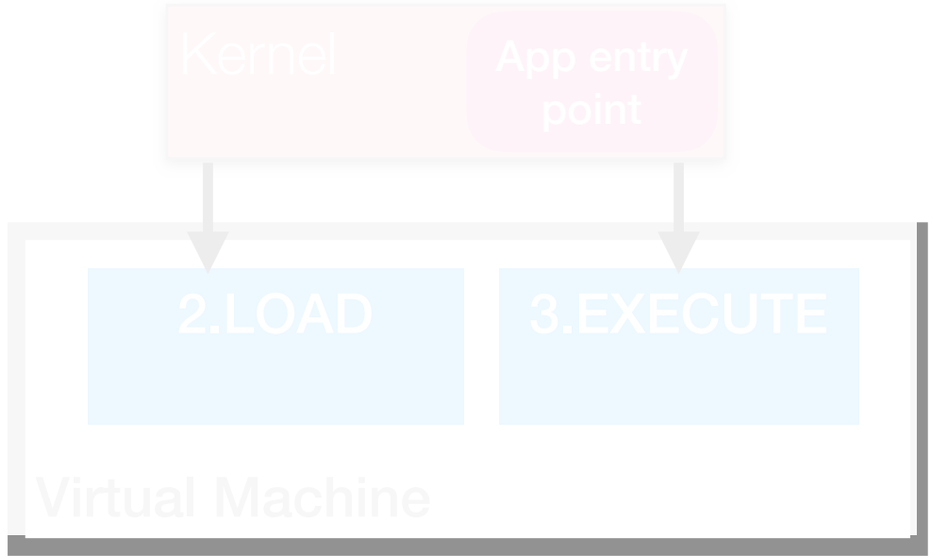
(They reify the VM requirements)



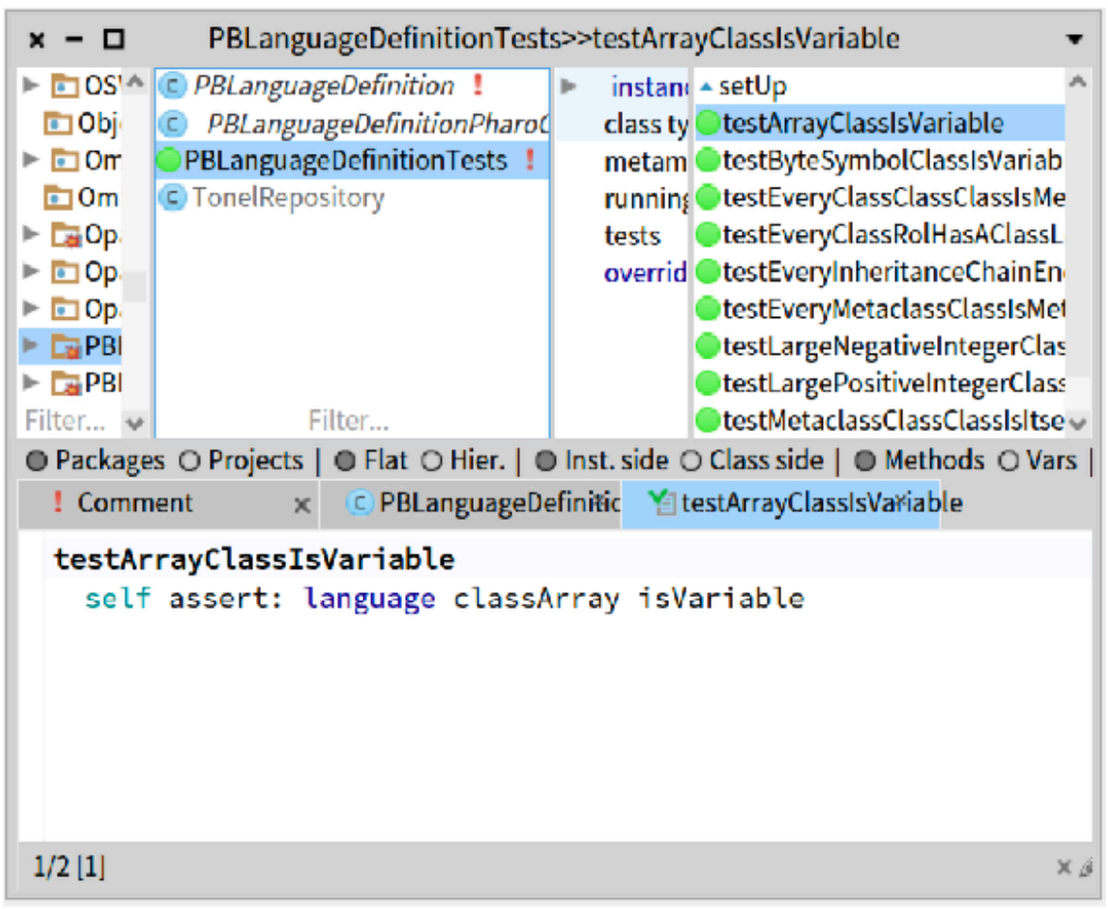
Taxonomy of Errors and Solutions







Static Tests on Language Kernel



+

Hybrid Debugger

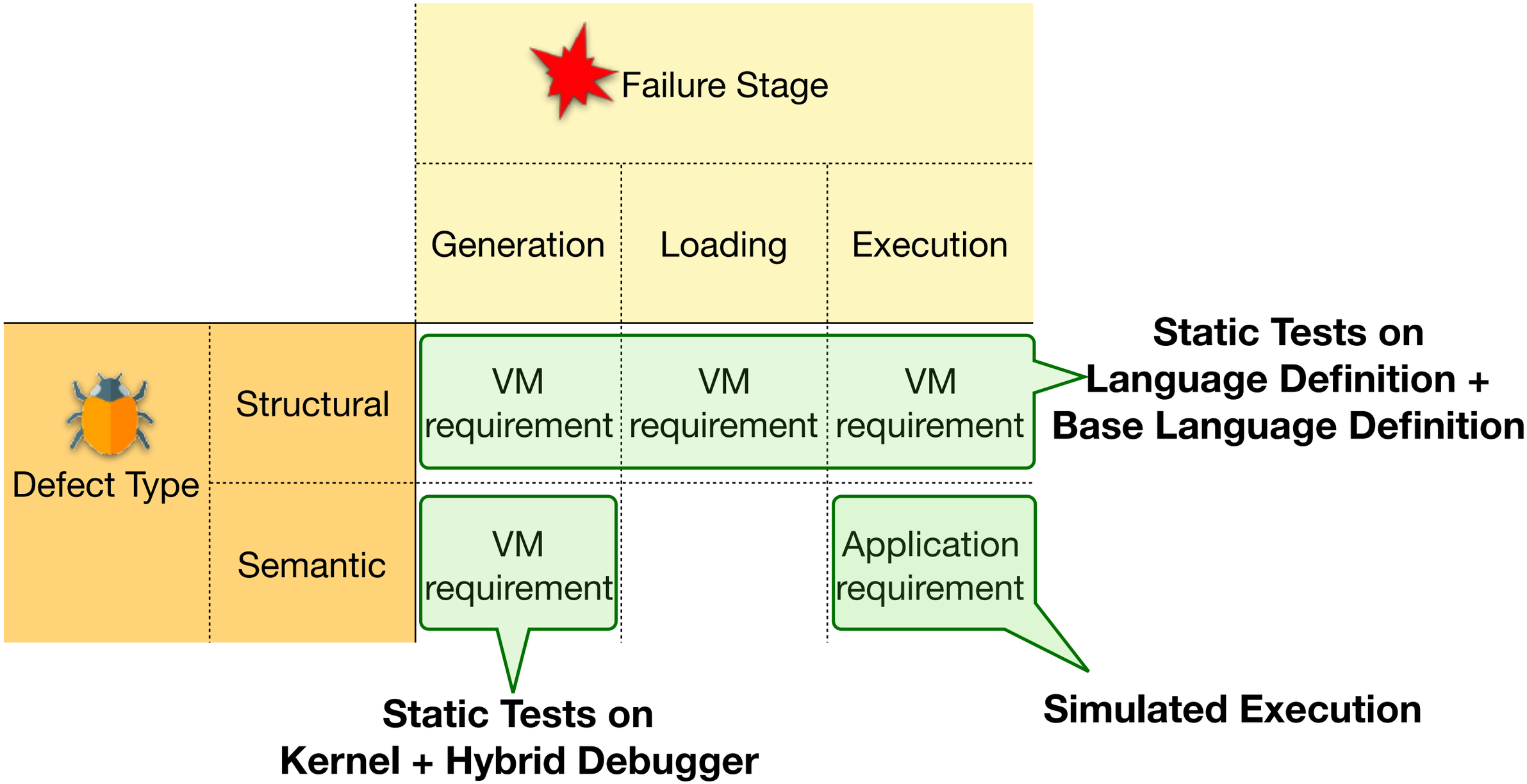
3 Execution levels:

- Language definition code
- Pharo code
- VM code

2 new Debugging Operations

- Step Down
- Step Up

Taxonomy of Errors and Solutions



Language Definition

Structural Definitions of Classes

Methods Definitions

App entry point

Lookup

Execution Simulator
AST interpreter
+
VM simulator

R/W objects

Kernel

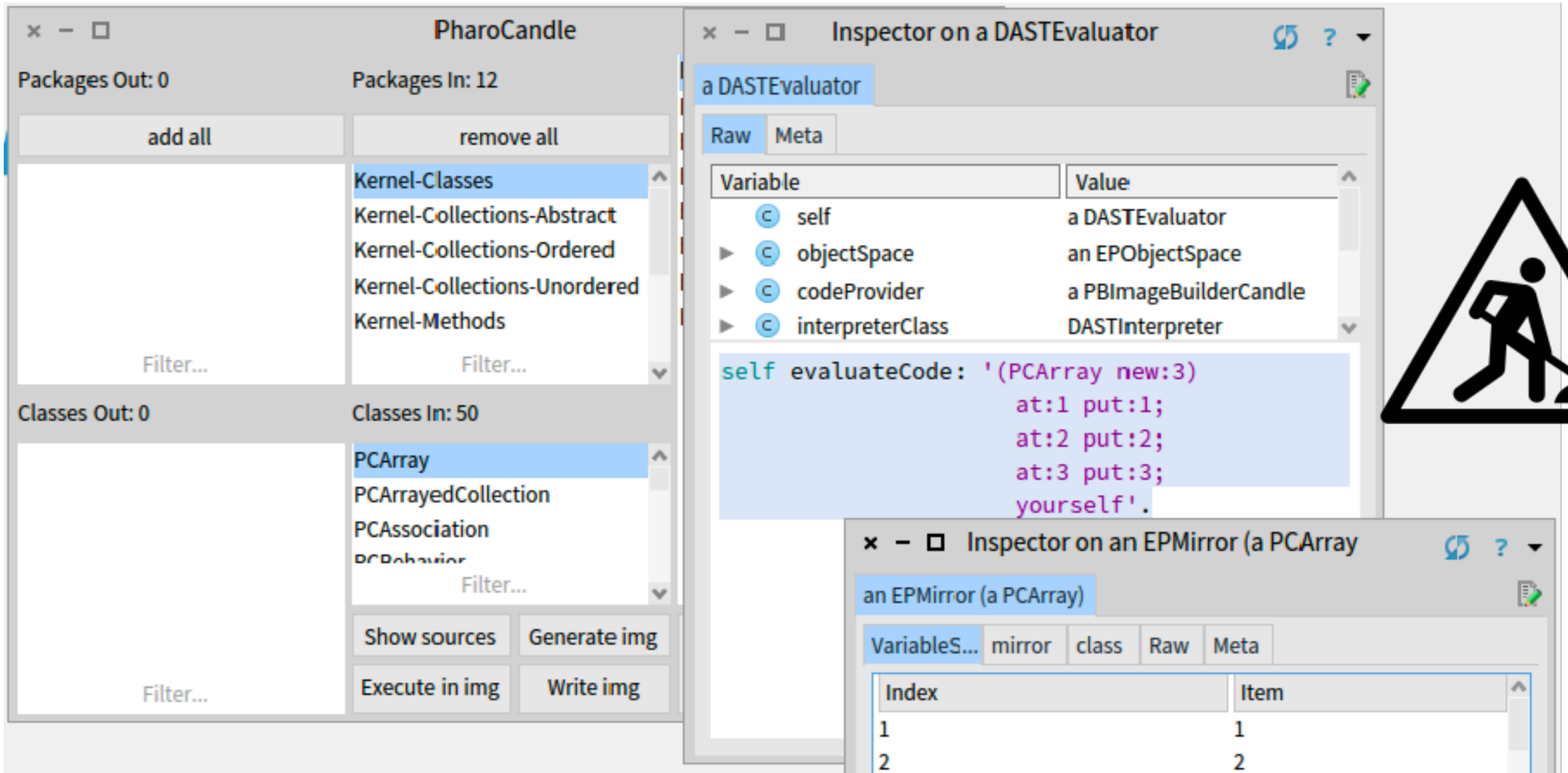
Kernel

App entry point

2.LOAD

3.EXECUTE

Virtual Machine



Taxonomy of Errors and Solutions

All these solutions can be used to debug the current Pharo bootstrap process!!

Research Directions

- Define the Pharo VM requirements, and model them for future modifications in future VM implementations

- Maximise the flexibility of the extensible base language definition, to maximise the range of languages that we can define from it

- Explore what is a good design for the hybrid debugger, so it contains the correct abstractions for debugging the bootstrap process

- Explore the limitations for the simulated execution environment

- Explore a way to debug failures hard to reproduce and which occur in production environment

- Shrinking the VM by removing unused plugins, which will be determined by dynamically analysing the simulated execution and its interaction with the VM simulator

Conclusions

- Analysis of Pharo Bootstrap process
- Taxonomy of Defects and Failures
- Proposed Solutions for each kind of error

Carolina Hernández Phillips
carolina.hernandez-phillips@inria.fr