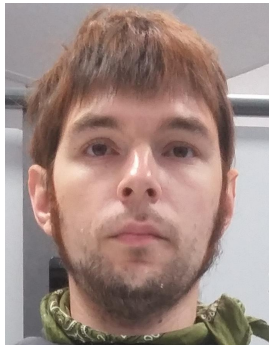
Active research on advanced debugging tools

Thomas Dupriez
Steven Costiou
RMoD
Inria Lille - Nord Europe

Who Are We?



Thomas Dupriez

PhD student (1st year)

RMoD

University of Lille,

Inria Lille Nord Europe

Steven Costiou

Researcher

RMoD

Inria Lille Nord Europe

What are “advanced debugging tools”?

- Debuggers are hard to build
- Debuggers are hard to understand and to use
- Lots of tools from research work aim at solving a specific problem

What are “advanced debugging tools”?

- Debuggers are hard to build
- Debuggers are hard to understand and to use
- Lots of tools from research work aim at solving a specific problem

We want to give you tools to build your own debugging tools!

What are “advanced debugging tools”?

- Why building your own debugging tools?
  - You need tools for your day-to-day problems
  - You need the right level of abstraction
  - You need to adapt the level of abstraction to the debugging context

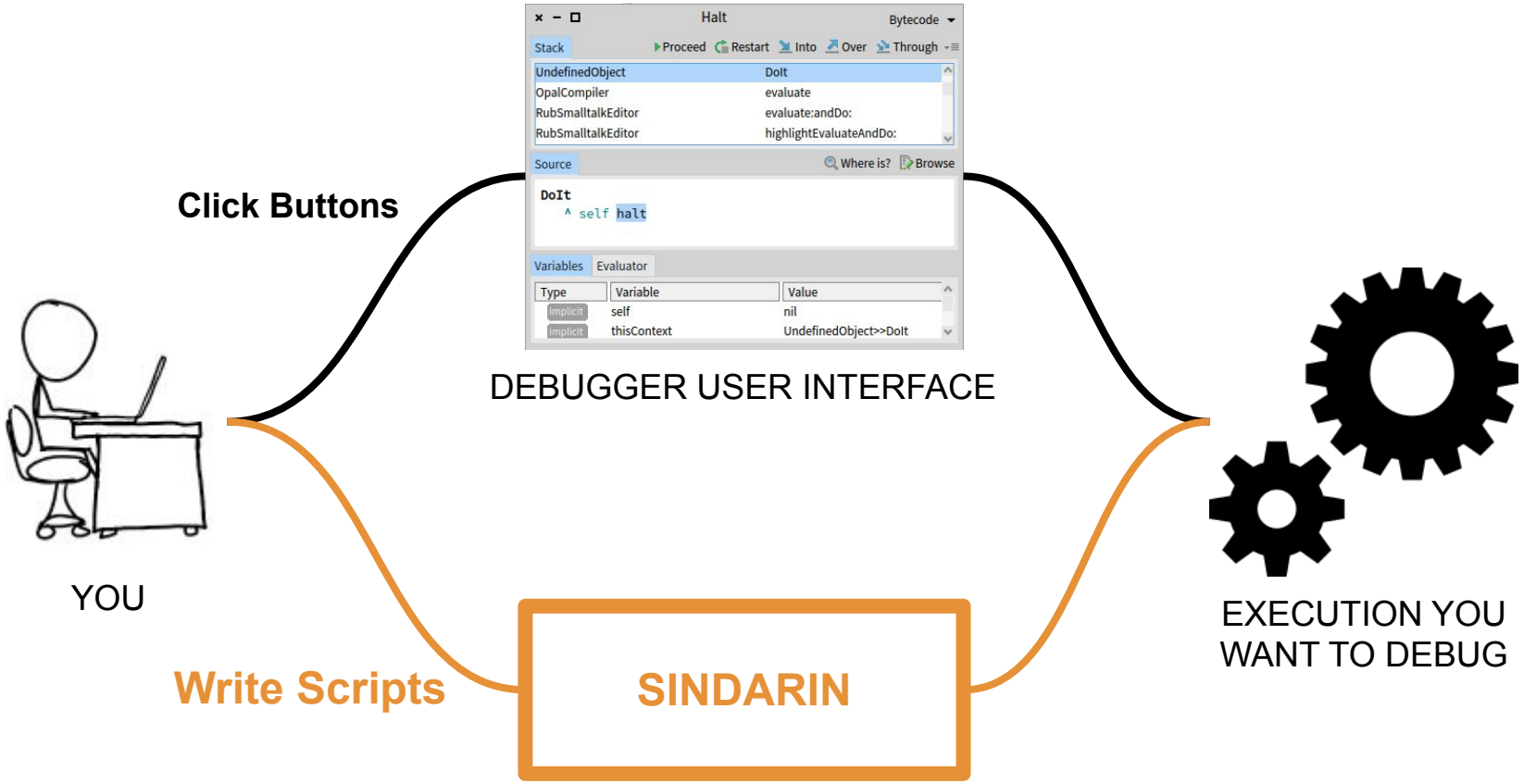
What are “advanced debugging tools”?

- Sindarin
  - An API to interact with the Pharo debugger and to build custom debugging tools

- Applications of Sindarin

1) Scriptable debugger
2) Power Assert
3) Customization of the debugger

Application 1: Scriptable Debugger



With Sindarin, developers have an alternative way to interact with their debugger: scripting it

Warm-Up Example: Gold Digger

Demo

DEMO

Warm Up: Gold Digger

1) Manually step (a lot) through the rocks to find the gold

Class	Method
SindarinDemo_GoldDigger	rock6
SindarinDemo_GoldDigger	rock5
SindarinDemo_GoldDigger	rock4
SindarinDemo_GoldDigger	rock3
SindarinDemo_GoldDigger	rock2
SindarinDemo_GoldDigger	rock1

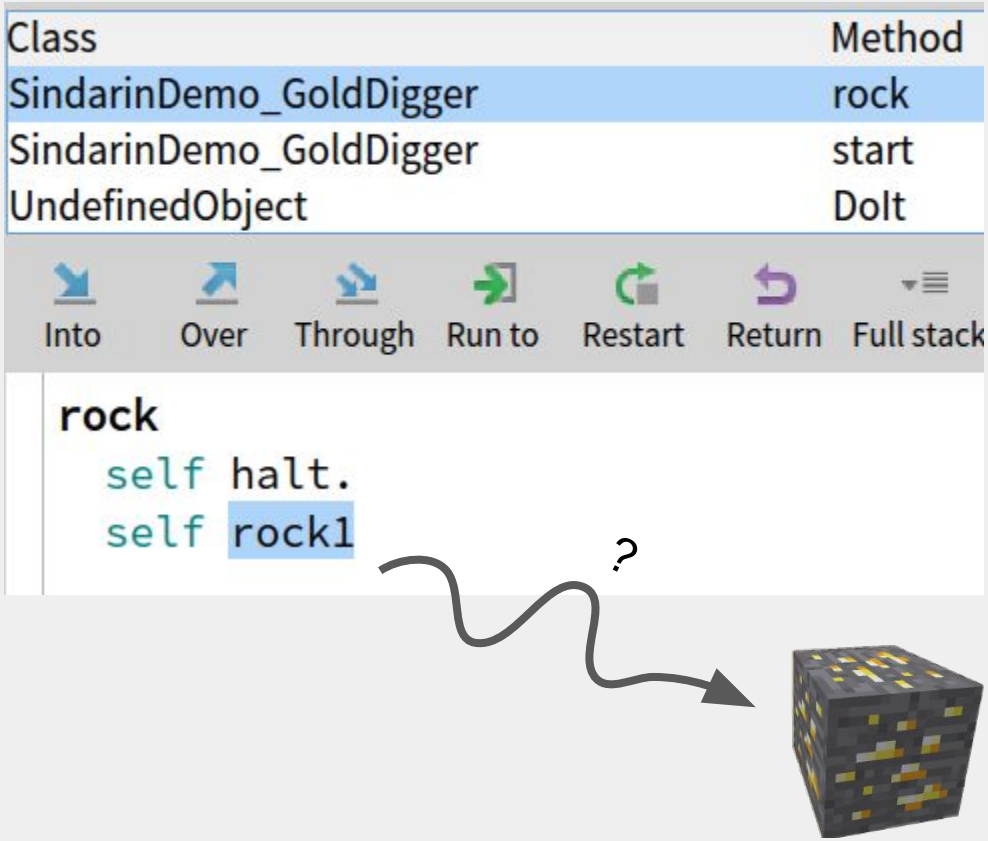
Into Over Through Run to Restart Return Full stack Where is? Proceed

```
rock6
  "Relevant point for the bug you are tracking"
  self gold.
```

DEMO

Warm Up: Gold Digger

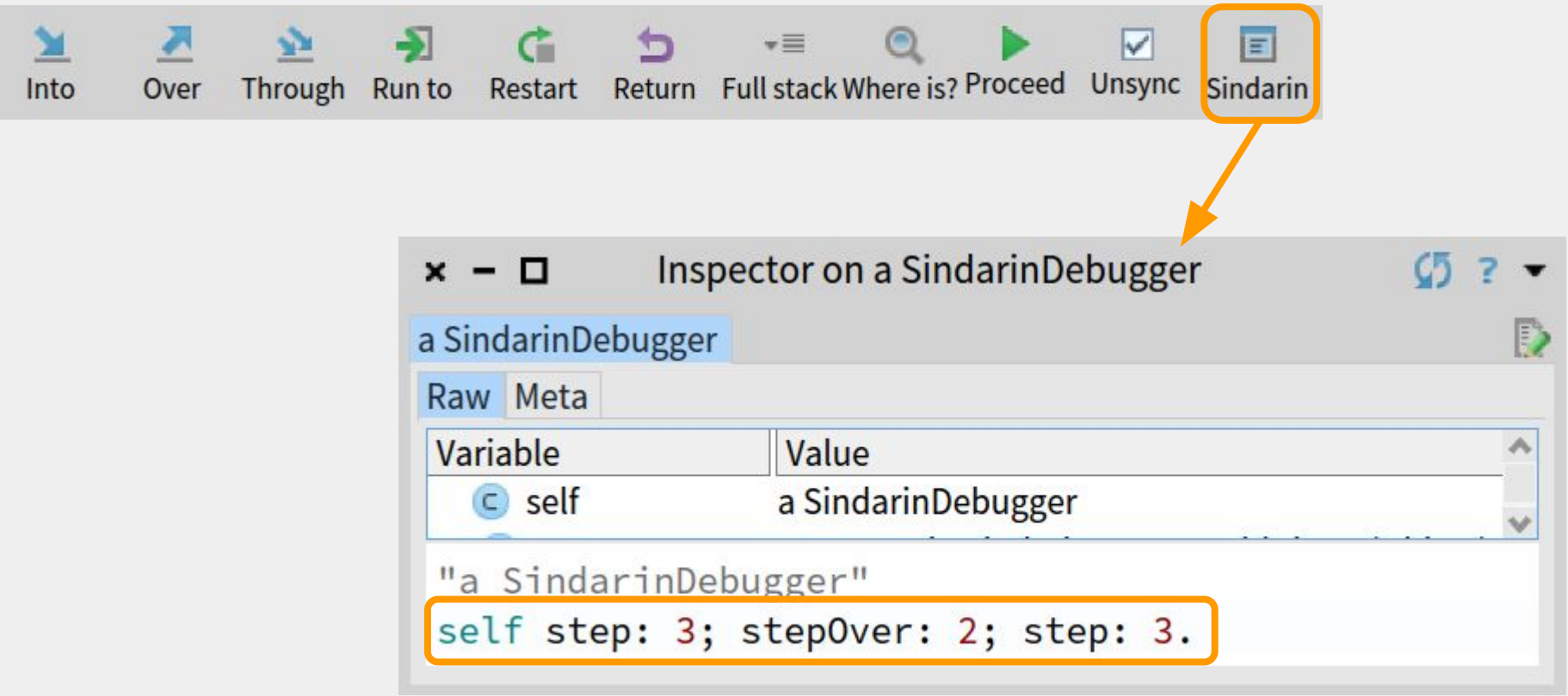
1) Manually step (a lot) through the rocks to find the gold
2) After restarting the execution, we would like to get back to the gold



DEMO

Warm Up: Gold Digger

1) Manually step (a lot) through the rocks to find the gold
2) After restarting the execution, we would like to get back to the gold
3) Instead of repeatedly clicking debugger buttons, you can write a Sindarin script to get you back to the gold in an instant



The Sindarin API

sender

receiver

Contexts

temporaries

step

stepOver

Stepping

method

arguments

proceed

skip

AST

node

messageReceiver

messageSelector

Helpers

setBreakpoint

assignmentValue

Breakpoints

assignmentVariableName

remove

whenHit:

And More...

Script Example: Step-to-next-Iteration

Demo

DEMO

Script Example: Step-to-next-Iteration

```
ctx := self context.
[ self context == ctx ] whileTrue: [ self stepOver ].
[ self receiver isKindOf: Array ] whileTrue: [ self step ].
```

Script Example: Skip-Next-Exception

Demo

DEMO

Script Example: Skip-Next-Exception

```
[self node isMessage and:
    [(self messageSelector = #signal:) and:
        [ Exception allSubclasses includes: self
    messageReceiver ]]]
        whileFalse: [ self step ].
self skip.
```

Application 2: PowerAssert

Like #assert: , but records intermediate results, and can replay faulty executions

Demo

DEMO

PowerAssert

```
testPowerAsserterReplayerExample
   | divisorsOfSixty |
   divisorsOfSixty := { 0. 1. 2. 3. 4. 5. 6. 10. 12. 15. 20. 30. 60 }.
   PowerAsserter new assert: [ (60 / (divisorsOfSixty at: divisorsOfSixty size atRandom)) isInteger]
```

When assertion fails

PowerAssert Visualisation

[(60 / (divisorsOfSixty at: divisorsOfSixty size atRandom)) isInteger]

Eval Order	AST Node	Value
1	RBLiteralValueNode(60)	60
2	RBTemporaryNode(divisorsOfSixty)	#(0 1 2 3 4 5 6 10 12 15 20 30 60)
3	RBTemporaryNode(divisorsOfSixty)	#(0 1 2 3 4 5 6 10 12 15 20 30 60)
4	RBMessageNode(divisorsOfSixty size)	13
5	RBMessageNode(divisorsOfSixty size atRandom)	1
6	RBMessageNode((divisorsOfSixty at: divisorsOfSixty size atRandom))	0
7	RBMessageNode((60 / (divisorsOfSixty at: divisorsOfSixty size atRandom)))	0
8	RBReturnNode(^ (ZeroDivide dividend: self) signal)	
9	RBReturnNode(^ (ZeroDivide dividend: self) signal)	
10	RBMessageNode((60 / (divisorsOfSixty at: divisorsOfSixty size atRandom)) isInteger)	false

Replay

AST nodes, with their value

Replay faulty execution

DEMO

PowerAssert - Replay

PowerAssert Replayer

[(60 / (divisorsOfSixty at: divisorsOfSixty size atRandom)) isInteger] in PowerAsserterExamples>>testPowerA

[self value. Processor terminateActive] in BlockClosure>>newProcess

Into | Over | Over (Replay) | Restart | Inspect Scriptable Debugger | Inspect Self

```
testPowerAsserterReplayerExample_clean
    | divisorsOfSixty |
    divisorsOfSixty := { 0. 1. 2. 3. 4. 5. 6. 10. 12. 15. 20. 30. 60 }.
    PowerAsserter new assert: [ (60 / (divisorsOfSixty at: divisorsOfSixty size atRandom)) isInteger]
```

Value Stack

#(0 1 2 3 4 5 6 10 12 15 20 30 60)
60
#(0 1 2 3 4 5 6 10 12 15 20 30 60)
#(0 1 2 3 4 5 6 10 12 15 20 30 60)

Replay value:

Raw | Breakpoints | Meta

Variable	Value
Σ self	13

Over (Replay)

PowerAssert Replayer

[(60 / (divisorsOfSixty at: divisorsOfSixty size atRandom)) isInteger] in PowerAsserterExamples>>testPowerA

[self value. Processor terminateActive] in BlockClosure>>newProcess

Into | Over | Over (Replay) | Restart | Inspect Scriptable Debugger | Inspect Self

```
testPowerAsserterReplayerExample_clean
    | divisorsOfSixty |
    divisorsOfSixty := { 0. 1. 2. 3. 4. 5. 6. 10. 12. 15. 20. 30. 60 }.
    PowerAsserter new assert: [ (60 / (divisorsOfSixty at: divisorsOfSixty size atRandom)) isInteger]
```

Value Stack

#(0 1 2 3 4 5 6 10 12 15 20 30 60)
60
#(0 1 2 3 4 5 6 10 12 15 20 30 60)
1

Replay value:

Raw | Breakpoints | Meta

Variable	Value
Σ self	0

PowerAssert

Sindarin Script performing the recording:

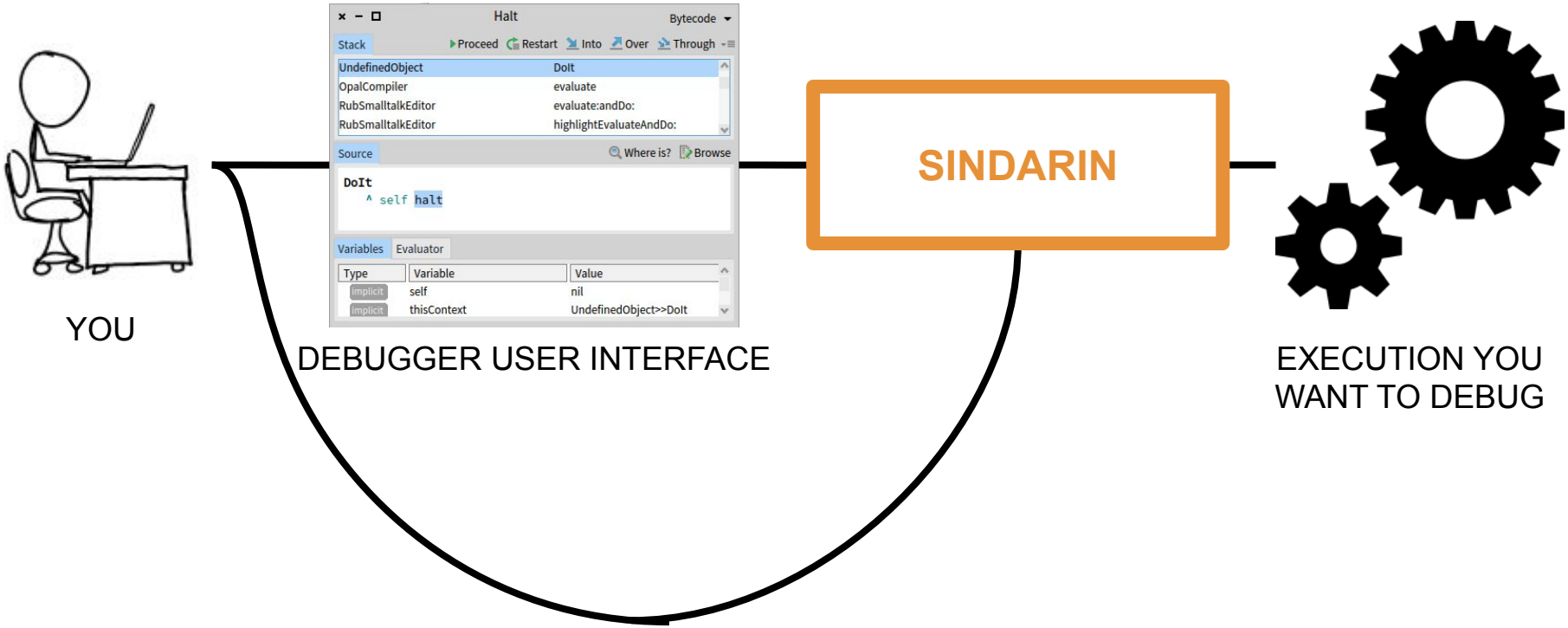
```
dbg := SindarinDebugger debug: aBlock.
evaluationData := OrderedCollection new.
blockNode := dbg context closure sourceNode.

[dbg currentNode == blockNode] whileFalse: [
 node := dbg currentNode.
 dbg stepOver.
 evaluationData add: {node. dbg context top}.
].
^ evaluationData.
```

Other Neat Things One Can Do With Sindarin

- Step until a temporary variable with a given name is assigned a new value
- Step until the current context returns
- Record all messages being sent to a given object during an execution
- Step two similar (but not exactly identical) executions until they diverge

And More...

Application 3: Customization of the Debugger

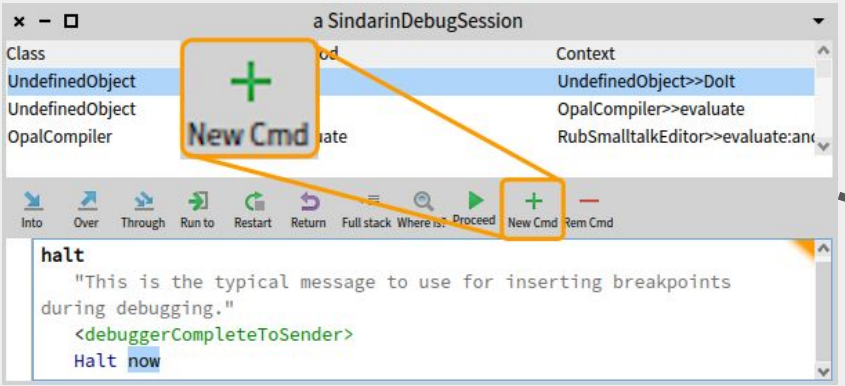


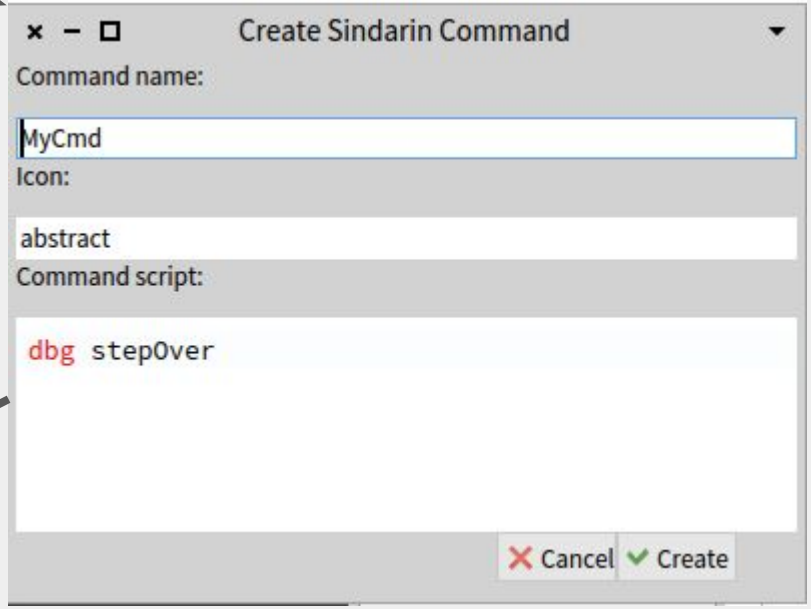
Customization of the Debugger

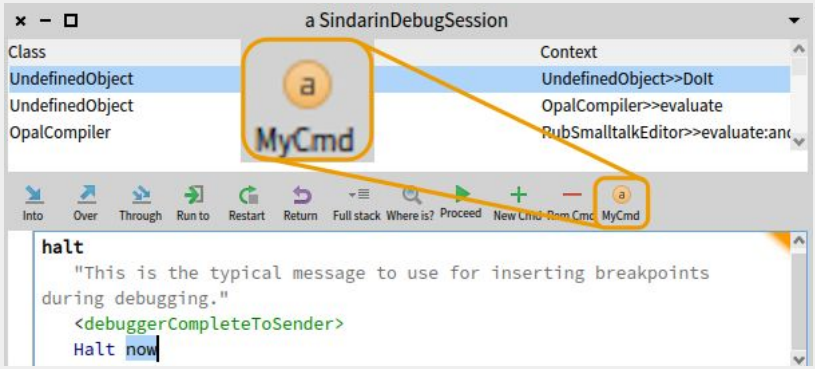
Demo

DEMO

Customization of the Debugger







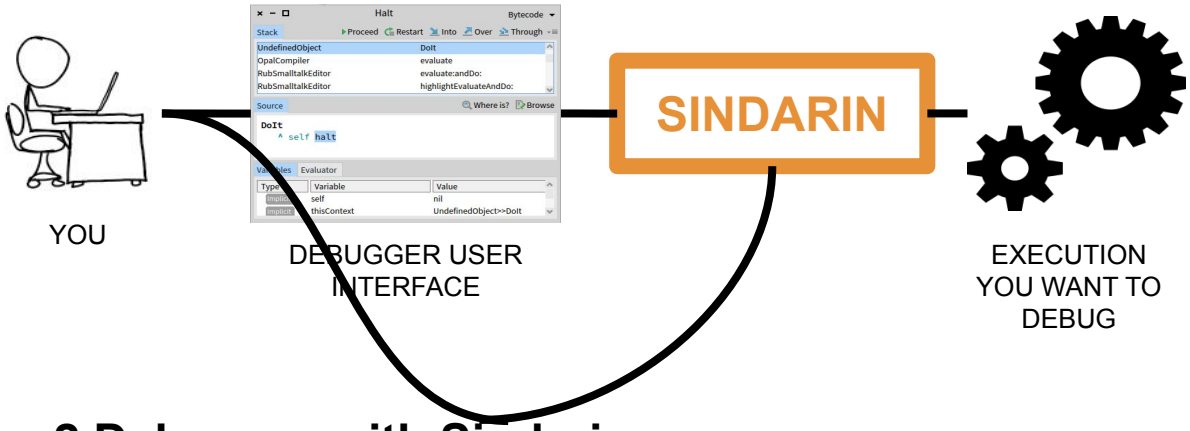
Planned

- Debugger recording the operations you perform and offering them to you as a Sindarin script later

- Sindarin API offering Object-Centric Debugging operations, like #haltOnCall, #haltOnWrite…

- Wishes? Come talk to us!

Thank you! Questions?



Spec2 Debugger with Sindarin

https://github.com/dupriezt/Spec2Debugger

Get it on Github!

```
Metacello new
    baseline: 'Spec2Debugger';
    repository: 'github://dupriezt/Spec2Debugger';
    load.
```

Just Sindarin (no-UI)

https://github.com/dupriezt/ScriptableDebugger

```
Metacello new
    baseline: 'Sindarin';
    repository: 'github://dupriezt/ScriptableDebugger';
    load.
```