

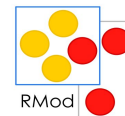
# Towards Object-centric Time-traveling Debuggers

Maximilian Ignacio Willebrinck Santander

Steven Costiou

Adrien Vanègue

Anne Etien



# Agenda

Towards  
Object-centric  
Time-traveling  
Debuggers

**I. Context** ←

**II. Proposition**

**III. Our Work**

# New debugging tools...

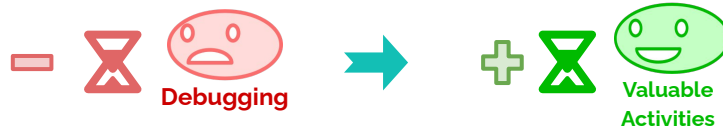
## Why?

- Debugging is a time-consuming task.
- Involves repetitive and tedious manual operations.

# New debugging tools...

## Why?

- Debugging is a time-consuming task.



- Involves repetitive and tedious manual operations.

# New debugging tools...

## Why?

- Debugging is a time-consuming task.

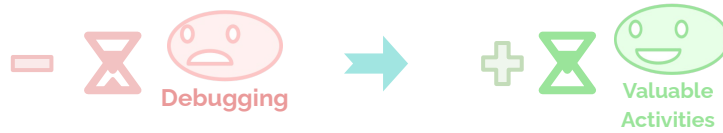


- Involves repetitive and tedious manual operations.

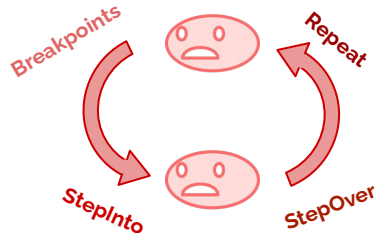
# New debugging tools...

## Why?

- Debugging is a time-consuming task.



- Involves repetitive and tedious manual operations.



# New debugging tools...

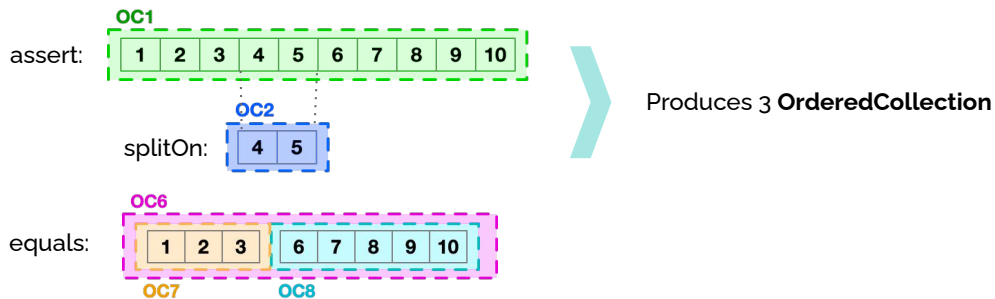
Why?

**Debugging is hard.**

We saw an opportunity to improve the debugging experience.

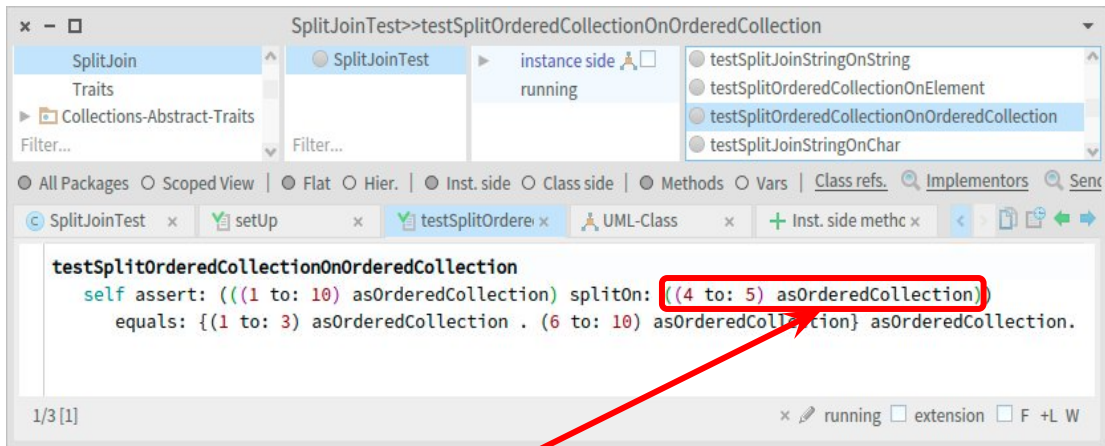
# Debugging Objects is Challenging: An Example

```
testSplitOrderedCollectionOnOrderedCollection
self assert: (((1 to: 10) asOrderedCollection) splitOn: ((4 to: 5) asOrderedCollection))
equals: {(1 to: 3) asOrderedCollection . (6 to: 10) asOrderedCollection} asOrderedCollection.
```





# Debugging Objects is Challenging: An Example



The screenshot shows an IDE window titled "SplitJoinTest>>testSplitOrderedCollectionOnOrderedCollection". The left sidebar shows a project structure with "SplitJoin" and "Traits". The main editor displays the following code:

```
testSplitOrderedCollectionOnOrderedCollection
  self assert: (((1 to: 10) asOrderedCollection) splitOn: ((4 to: 5) asOrderedCollection))
    equals: {(1 to: 3) asOrderedCollection . (6 to: 10) asOrderedCollection} asOrderedCollection.
```

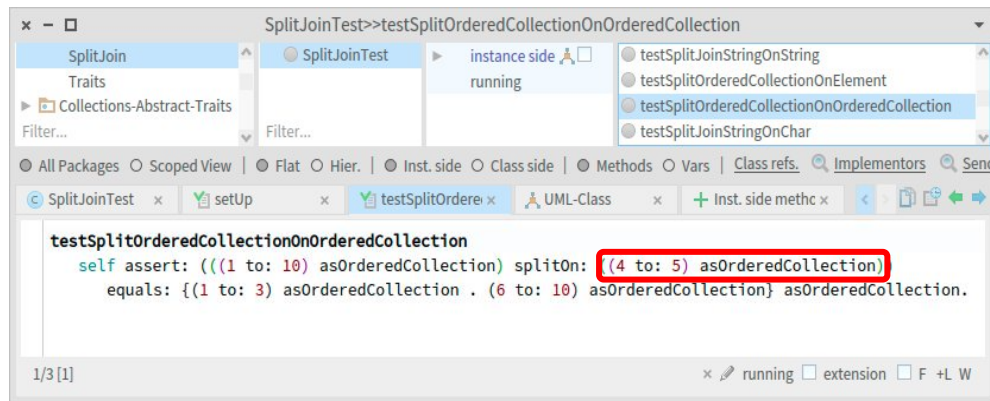
A red box highlights the argument `((4 to: 5) asOrderedCollection)`, and a red arrow points from this box towards the text below.

There is an `OrderedCollection` instantiated somewhere during that call.

*"I want to see how its instance variables evolve"*



# Debugging Objects is Challenging: An Example



## Conventional Debugging

There is an OrderedCollection instantiated somewhere during that call.

➔ **1. Find the object**



➔ To instantiation instruction

"I want to see how its instance variables evolve"

➔ **2. Specific object debugging operations**

➔ Many steps / uncomfortable conditional breakpoints

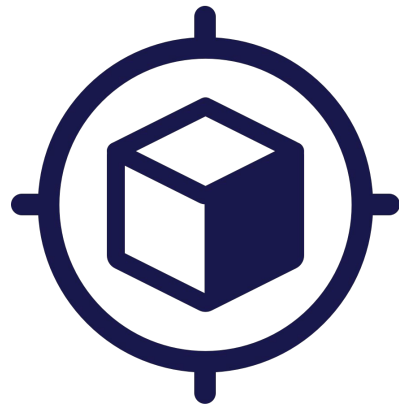
# Debugging Objects is Challenging: An Example

**1. Find the object**

**2. Specific object  
debugging operations**

*Are there special tools to make  
these task easier?*

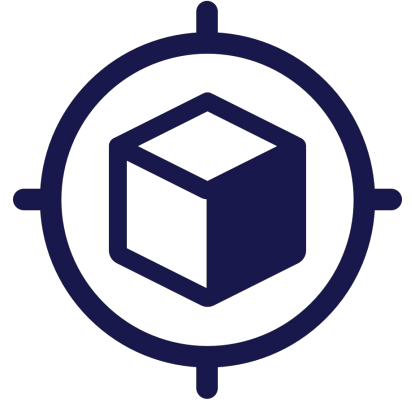
# Object-centric debugging?...



- What is Object-centric debugging [1]?
  - Makes **objects** the focus of debugging operations.
  - Debugging operations are defined to answer developer questions related to runtime behavior of objects.
  - Object-centric Debugging operators examples:
    - Halt on read
    - Halt on write
    - Halt on call

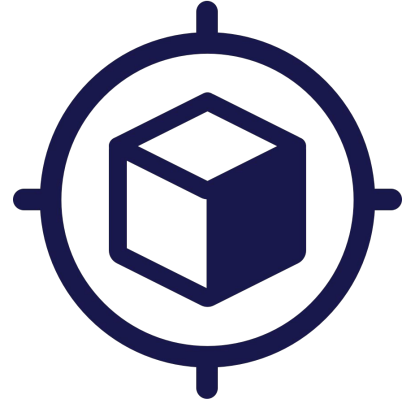
# Object-centric

## A promising debugging approach for OOP[1]

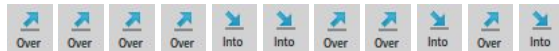


- It helps debugging when:
  - Developers debugging *questions* are closer to the objects which model the domain.
  - Developers want to follow an object behavior, avoiding traditional breakpoint management tediousness. *e.g: "I want to see how its instance variables evolve"*
- A prototype was presented and shown to be more effective supporting typical debugging tasks than traditional stack-oriented debugger.

# The problem...

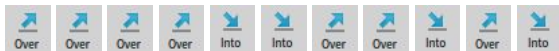


- **Scoping debugging operations (such as breakpoints) on specific object is difficult!**
  - *Devs must initially find such object identity first.*
  - *Manually step to the object of interest, or step through breakpoints hits to find it.*

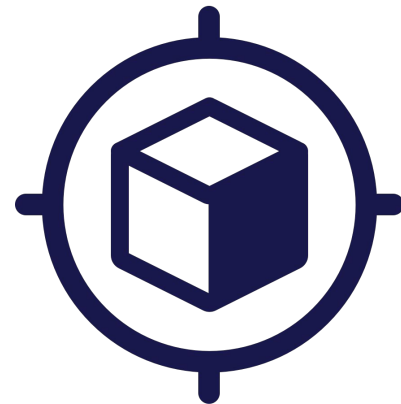


# The problem...

- **Scoping debugging operations (such as breakpoints) on specific object is difficult!**
  - *Devs must initially find such object identity first.*
  - *Manually step to the object of interest, or step through breakpoints hits to find it.*



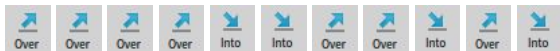
**TEDIOUS AND  
ERROR-PRONE MANUAL  
WORK!**



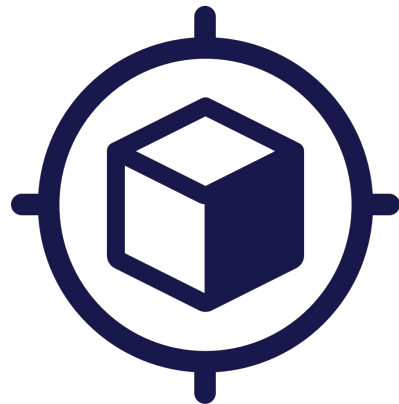
# The problem...

- Scoping debugging operations (such as breakpoints) on specific object is difficult!

- *Devs must initially find such object identity first.*
- *Manually step to the object of interest, or step through breakpoints hits to find it.*



*If only there was a way to fix this ...*



**TEDIOUS AND  
ERROR-PRONE MANUAL  
WORK!**





# Time-traveling debuggers



- Main features: reverse a program execution and deterministic replay.
  - With these debuggers, any stepping error can be amended by stepping back. (Stepped too far? No need to restart, just take a step back).
  - Developers want to check a past state of the program? No need to restart, just reverse it. *e.g: reverse to an object instantiation.*

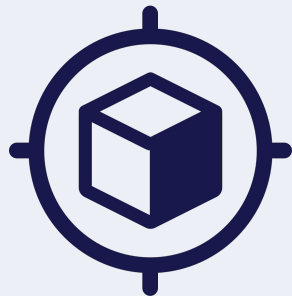
# There is another problem...



- **To the best of our knowledge, time-traveling solutions don't provide object-centric debugging operators.**

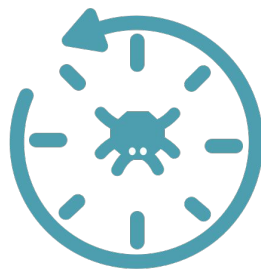
# Context summary

## Object Centric Debuggers



- Improves debugging experience in OOP.
- **Sill tedious and error prone.**

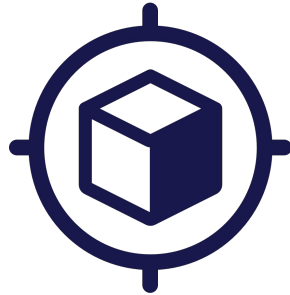
## Time-Traveling Debuggers



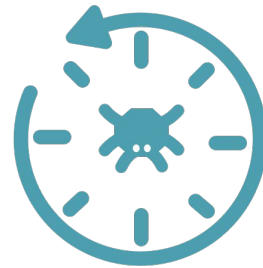
- Improves tediousness, and debugging/stepping mistakes are less costly.
- **So far, no support for object-centric debugging.**

# Context summary

Object Centric  
Debuggers



Time-Traveling  
Debuggers



- Improves debugging experience in OOP.
- Sill tedious and error prone.
- Improves tediousness, and debugging/stepping mistakes are less costly.
- So far, no support for object-centric debugging.



# Agenda

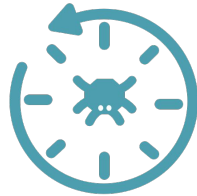
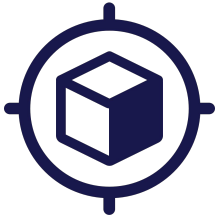
Towards  
Object-centric  
Time-traveling  
Debuggers

I. Context

II. Proposition ←

III. Our Work

# Our proposition

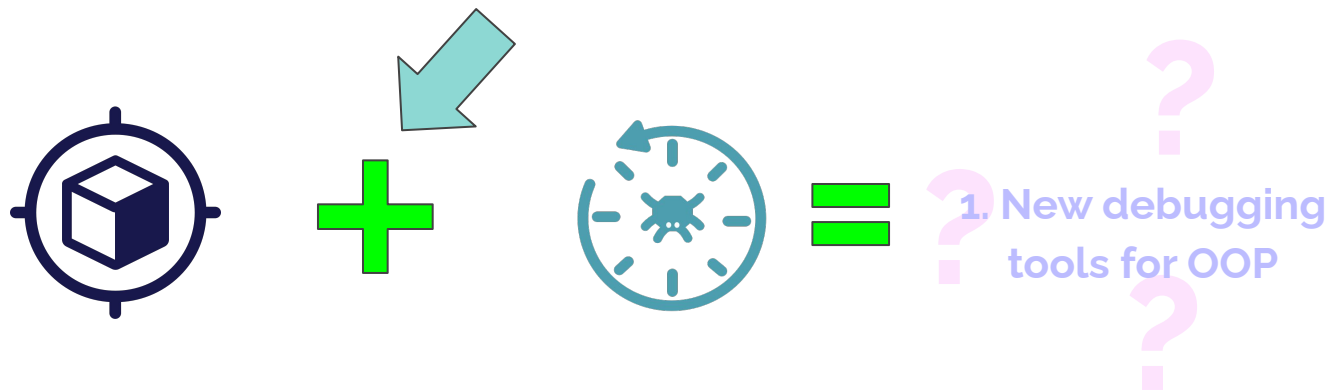


1. New debugging  
tools for OOP



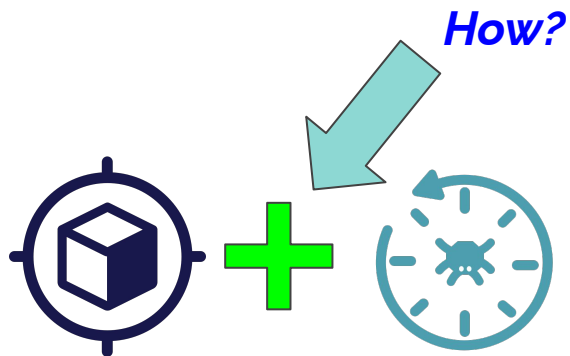
# Our proposition

2. How to combine them?



Second general question:  
How to combine them?

# How to combine them?



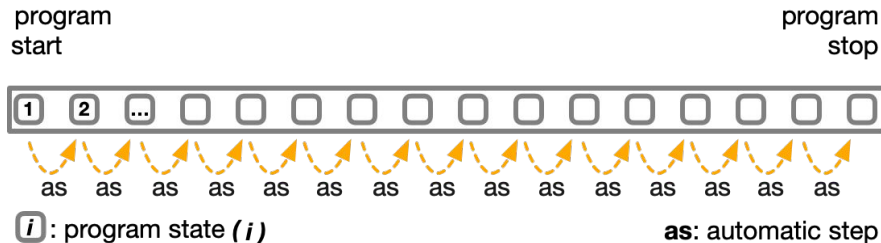
Our answer:

## Time-Traveling Queries



# Combining both techniques

## Time-Traveling Queries\*



*What is the value of this variable during execution?*

*On step 1 value changed from nil to 100*  
*On step 4 value changed from 100 to 200*  
*On step 40 value changed from 200 to 0*

...

(\*) M. Willebrinck, S. Costiou, A. Etien, S. Ducasse. Time-Traveling Debugging Queries: Faster Program Exploration. *International Conference on Software Quality, Reliability, and Security*, Dec 2021, Hainan Island, China.

# Combining both techniques

## Time-Traveling Queries\*

Do you have a debugging question?

Select a Time-Traveling Query from the Queries Menu!

Find execution data and explore your execution conveniently from the Query Results

The screenshot shows the debugger interface for a test named 'testLinksDo'. The 'Stack' pane on the left lists the current execution context. The main editor shows the source code of the test. The 'Queries Menu' is open, displaying a list of query categories: Messages, Messages - Object Centric, Instances Creations, Assignments - Object Centric, and Assignments - General. The 'Messages' category is selected, and a sub-menu is visible with options: All Message Sends, All Message Sends with selected selector, and All Received messages. A green callout box labeled 'Debugger' points to the top of the interface, and another green callout box labeled 'Queries Menu' points to the menu itself.

The screenshot shows the 'Query Results' window. At the top, there is a 'Stepping Control' section with buttons for 'Back 1', 'Adv. 1', 'Adv. Statement', 'Prev. Statement', 'Reset', 'To End', and 'STOP'. Below this is a 'Query' section with a 'Scripting' tab. The main area displays a table of results for the query 'Query for All Message Sends with selected selector : (add:)'.

Step	Msg Receiver	Oid	Msg Selector	
1	56	list (DoubleLinkedList)	8	add:
2	104	links (OrderedCollection)	18	add:
3	138	list (DoubleLinkedList)	8	add:
4	191	links (OrderedCollection)	18	add:
5	225	list (DoubleLinkedList)	8	add:
6	278	links (OrderedCollection)	18	add:
7	317	list (DoubleLinkedList)	8	add:

Below the table, there is a 'Filter...' input field. At the bottom of the window, it says 'Showing 20 results, fetched in: 282ms.' and 'ExecutedBytecode: 56 (3.06% of known execution)'. A green callout box labeled 'Step number (timestamp)' points to the 'Step' column, and another green callout box labeled 'Query results' points to the table.

(\* M. Willebrinck, S. Costiou, A. Etien, S. Ducasse. Time-Traveling Debugging Queries: Faster Program Exploration. International Conference on Software Quality, Reliability, and Security, Dec 2021, Hainan Island, China.

# Combining both techniques

With **Time-Traveling Queries**

We developed *SeekerOC* and an  
*Enhanced Pharo Inspector* to improve the  
debugging experience.

Made for Pharo 10.



# Agenda

Towards  
Object-centric  
Time-traveling  
Debuggers

**I. Context**

**II. Proposition**

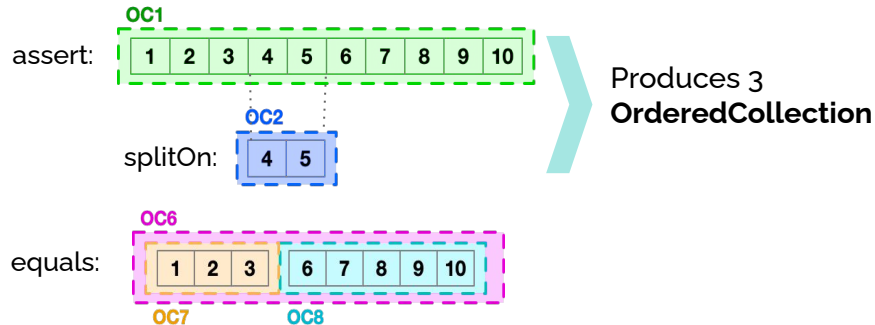
**III. Our Work** ←

# Improving Debugging Challenges: Back to the example

```
testSplitOrderedCollectionOnOrderedCollection
self assert: ((1 to: 10) asOrderedCollection) splitOn: ((4 to: 5) asOrderedCollection)
equals: {(1 to: 3) asOrderedCollection . (6 to: 10) asOrderedCollection} asOrderedCollection.
```

*There is an  
OrderedCollection  
instantiated somewhere  
during that call.*

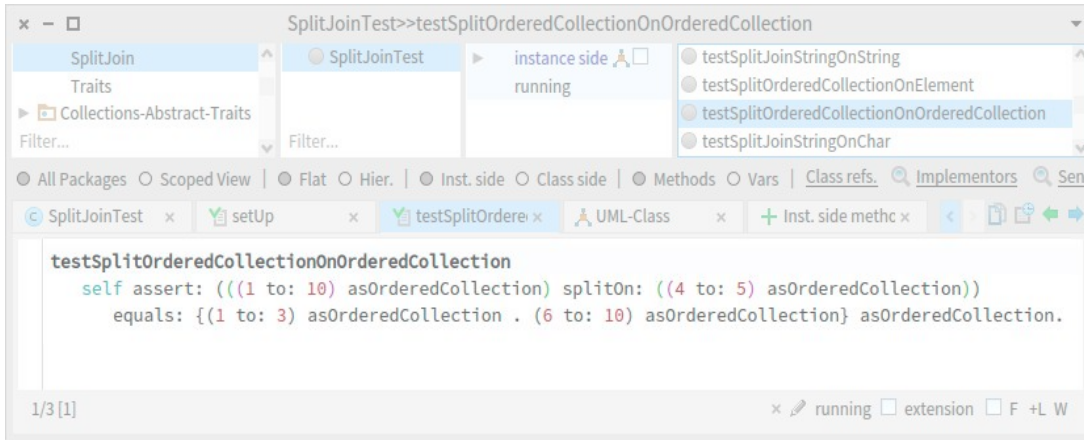
*"I want to see how its  
instance variables  
evolve"*



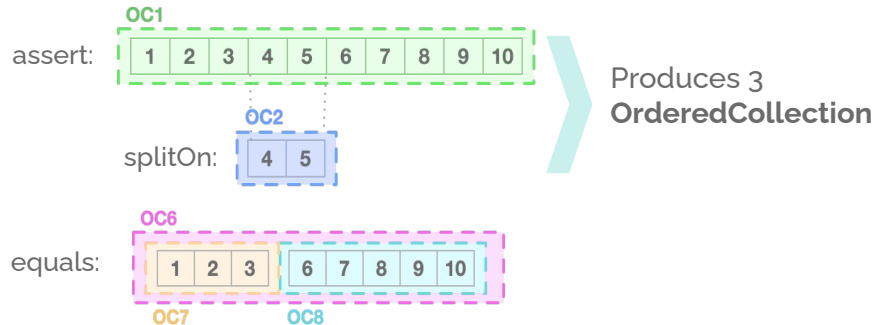
**1. Find the object**

**2. Specific object  
debugging operations**

# Improving Debugging Challenges: Back to the example



*We made tools  
to make this  
easier!*



**1. Find the object**

**2. Specific object  
debugging operations**

## NEW OOP DEBUGGING TOOL

# I. SeekerOC

### 1. Easy objects finding and identification using TTQs

The screenshot displays the SeekerOC debugging tool interface. The main window shows a test execution for `SplitJoinTest>>testSplitOrderedCollectionOnOrderedCollection`. The Stack pane lists the current method `testSplitOrderedCollectionOnOrderedCollection` in `SplitJoinTest` and its superclass `performTest` in `SUnit-Core`. The code editor shows the test method implementation, with a context menu open over the `asOrderedCollection` call. The menu includes options like `Do it`, `Inspect it`, `SeekerQueries`, `Find`, `Copy`, `Cut`, `Paste`, `Messages`, `Messages - Object Centric`, `Instances Creations`, `Assi All Instances Creation`, and `Assi All Instances Creation of class named as selection`. The `Instances Creations` option is highlighted, and a green arrow points to the `Assi All Instances Creation of class named as selection` option. The right pane shows the results of the query, listing 8 instances of `OrderedCollection` with their memory addresses and the class they were instantiated from. The query is `Query for All Instances Creation of class named as selection : (OrderedCollection)`. The results table has columns for Step, About to instantiate a:, and Sender's Class. The results show 8 instances of `OrderedCollection` at various memory addresses (67, 592, 760, 1136, 1501, etc.). The interface also shows stepping control buttons and a filter field.

Step	About to instantiate a:	Sender's Class
1	67	OrderedCollection [Collections-Sequence... OrderedColle
2	592	OrderedCollection [Collections-Sequence... OrderedColle
3	760	OrderedCollection [Collections-Sequence... OrderedColle
4	1136	OrderedCollection [Collections-Sequence... OrderedColle
5	1501	OrderedCollection [Collections-Sequence... OrderedColle

*Lists all OrderedCollection objects instantiated during the test*

## NEW OOP DEBUGGING TOOL

# I. SeekerOC

### 1. Easy objects finding and identification using TTQs

The screenshot displays the SeekerOC debugging tool interface. The main window shows the execution of the method `testSplitOrderedCollectionOnOrderedCollection` in the `SplitJoinTest` class. The code is visible in the editor, and the execution is paused at line 3. The `Stack` pane shows the current method call stack.

The `Seeker` pane is active, showing a list of objects found during the execution. The objects are listed in a table with columns for Step, About to instantiate a, and Sender's Class. The objects are:

Step	About to instantiate a	Sender's Class
1	67	OrderedCollection [Collections-Sequence... OrderedColle
2	592	Or Inspect the object about to be instantiated
3	760	Or List all messages send to the object
4	1136	Or List all assignments of the instance variables of the object
5	1501	Or Inspect the query result item
		Or Inspect the query result collection

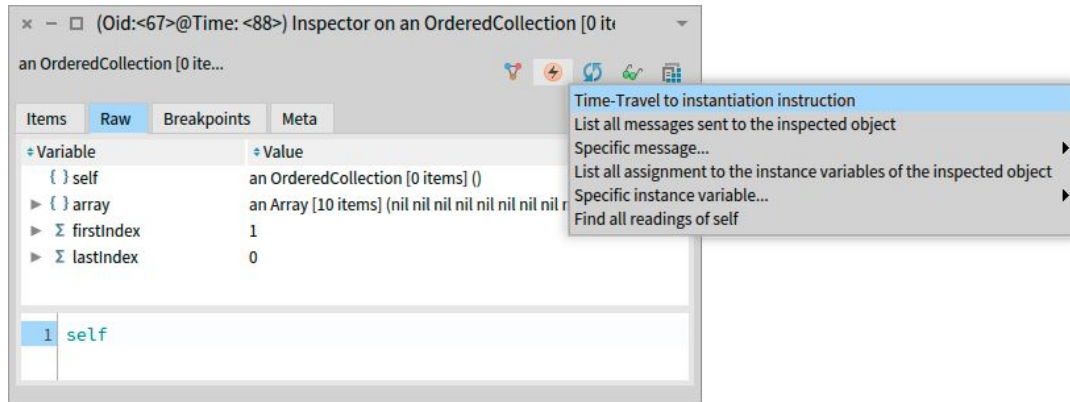
The interface also includes a `Stepping Control` section with buttons for `Back 1`, `Adv. 1`, `Adv. Statement`, `Prev. Statement`, `Reset`, `To End`, and `STOP`. There is also a `Query` section with a `Scripting` tab and a `Filter...` input field.

*The listed objects can be directly inspected.*



# II. Enhanced Pharo Inspector

## 2. Easy object and variables tracking, using Object-Centric TTQs.



## II. Enhanced Pharo Inspector

### 2. Easy object and variables tracking, using Object-Centric TTQs.

The screenshot shows the Pharo Inspector interface. The main window displays an `OrderedCollection` object with its variables: `self`, `array`, `firstIndex`, and `lastIndex`. A context menu is open over the `array` variable, listing several actions. The `Specific instance variable...` option is selected, which has opened a sub-menu showing the `array` variable and its `lastIndex` property.

The screenshot shows the Query window in the Pharo Inspector. It displays an `All Assignments Query` with 11 results. The results are shown in a table with columns for Step, Variable, Current Value, and To be Assigned. The `lastIndex` variable is shown being assigned values from 0 to 9 across the steps.

Step	Variable	Current Value	To be Assigned
1	87	lastIndex	nil
2	153	lastIndex	0
3	192	lastIndex	1
4	231	lastIndex	2
5	270	lastIndex	3
6	309	lastIndex	4
7	348	lastIndex	5
8	387	lastIndex	6
9	426	lastIndex	7
10	465	lastIndex	8
11	504	lastIndex	9

Showing 11 results, fetched in: 181ms.  
ExecutedBytecode: 88 (2.85% of known execution)

# Implementation, *Requirements*

***For SeekerOC and Enhanced Pharo Inspector.***

- *Time-traveling back end providing deterministic reverse and replay.*
- *Support for Time-Traveling Queries.*
- *Express (new) Time-Traveling Queries.*

Repository url: <https://github.com/Willembinck/SeekerOC-2022>

Implementation

# SeekerOC

Finding objects with TTQs

Queries Menu

Query Results

The screenshot shows the Seeker tool interface within an IDE. The main window title is "SplitJoinTest>>testSplitOrderedCollectionOnOrderedCollection". The interface is divided into several sections:

- Stack:** A table showing the current stack of frames.
- Stepping Control:** A set of buttons for navigating through the code execution (Back 1, Adv. 1, Adv. Statement, Prev. Statement, Repeat, To End, STOP).
- Query:** A section for defining and executing queries. The current query is "Query for All Instances Creation of class named as selection : (0)".
- Query Results:** A table displaying the results of the query.
- Queries Menu:** A context menu that is open, showing various options for finding objects, including "All Instances Creation of class named as selection".

Step	About to instantiate a:	Sender's Class
1	67	OrderedCollection [Collections-Sequence...
2	592	OrderedCollection [Collections-Sequence...
3	760	OrderedCollection [Collections-Sequence...
4	1136	OrderedCollection [Collections-Sequence...
5	1501	OrderedCollection [Collections-Sequence...

Showing 8 results, fetched in: 258ms.  
ExecutedBytecode: 34 (1.10% of known execution)

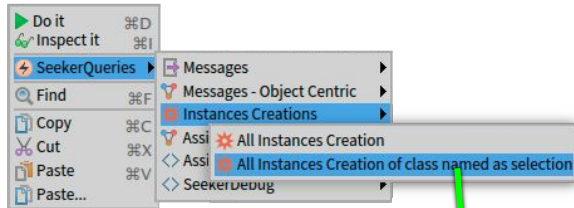
*How it works?*

## Implementation

# SeekerOC

## Finding objects with TTQs

Queries Context Menu



**#OrderedCollection**

Time-traveling debugger



**Iterable collection of  
ProgramState**

```
makeQueryFindInstancesOfClassNamed: aSymbol from: programStates
  ^ TimeTravelingQuery from: programStates
    select: [ :state |
      state isInstantiationMessage and:
        [ state messageReceiver class name = aSymbol ] ]
    collect: [ :state |
      ResultItem new
        step: state bytecodeIndex;
        aboutToInstantiate: state messageReceiver name;
        senderClass: state context receiver class;
        yourself ]
```

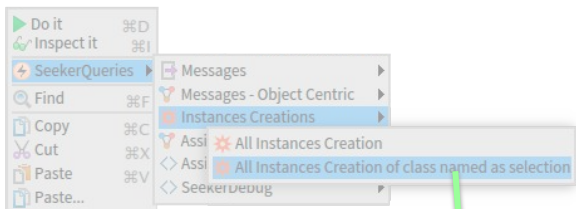
*The command  
executes a  
Time-Traveling  
Query*

## Implementation

# SeekerOC

## Finding objects with TTQs

Queries Context Menu



Time-traveling debugger



"Iterable" collection of  
ProgramState

#OrderedCollection

*From where to  
extract the data?*

*What program states  
are relevant?*

*What should be  
included in the results?*

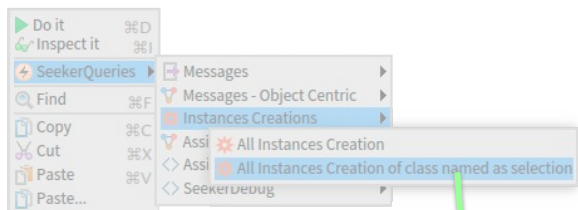
```
makeQueryFindInstancesOfClassNamed: aSymbol from: programStates
  ^ TimeTravelingQuery from: programStates
    select: [ :state |
      state isInstantiationMessage and:
      [ state messageReceiver class name = aSymbol ] ]
    collect: [ :state |
      ResultItem new
        step: state bytecodeIndex;
        aboutToInstantiate: state messageReceiver name;
        senderClass: state context receiver class;
        yourself ]
```

## Implementation

# SeekerOC

## Finding objects with TTQs

### Queries Context Menu



#OrderedCollection

### Time-traveling debugger



"Iterable" collection of

ProgramState

```
makeQueryFindInstancesOfClassNamed: aSymbol from: programStates
  ^ TimeTravelingQuery from: programStates
    select: [ :state ]
      state isInstantiationMessage and:
        [ state messageReceiver class name = aSymbol ] ]
    collect: [ :state ]
      ResultItem new
        step: state bytecodeIndex;
        aboutToInstantiate: state messageReceiver name;
        senderClass: state context receiver class;
        yourself ]
```

From where to extract the data?

What program states are relevant?

What should be included in the results?

### ProgramState:

API to access execution data.

API Example:

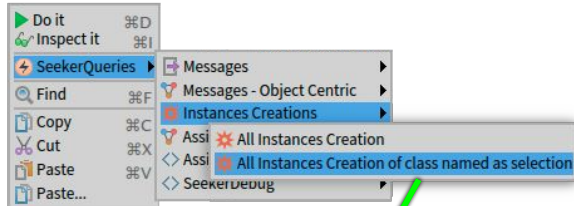
Method	Returns
context	Context
isMessageSend	Boolean
messageReceiver	Object

## Implementation

# SeekerOC

## Finding objects with TTQs

*Collected results are displayed in the Query Results table*



```
makeQueryFindInstancesOfClassNamed: aSymbol from: programStates
  ^ TimeTravelingQuery from: programStates
    select: [ :state |
      state isInstantiationMessage and:
      [ state messageReceiver class name = aSymbol ] ]
    collect: [ :state |
      ResultItem new
        step: state bytecodeIndex;
        aboutToInstantiate: state messageReceiver name;
        senderClass: state context receiver class;
        yourself ]
```

A screenshot of the Query Results table. The table has columns for 'Step', 'About to instantiate a:', and 'Sender's Class'. The results show a sequence of 'OrderedCollection' objects being instantiated. The table also includes a 'Filter...' dropdown and a status bar at the bottom indicating 'Showing 8 results, fetched in: 258ms.'.

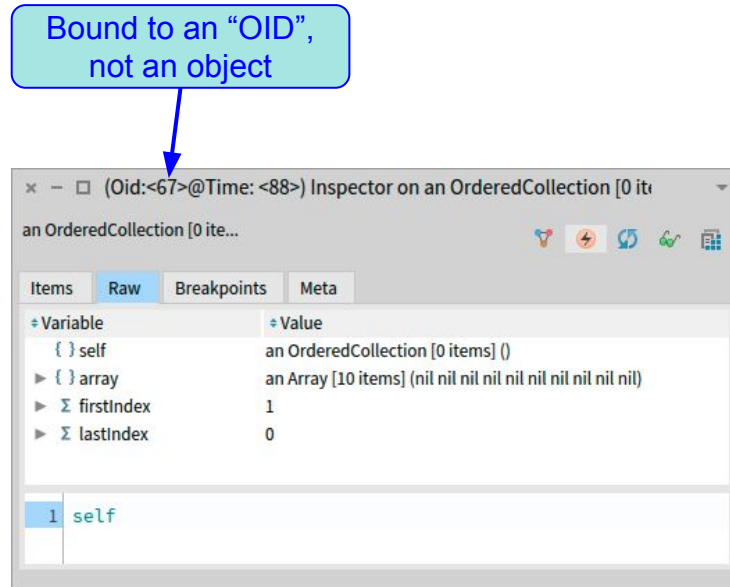
Step	About to instantiate a:	Sender's Class
1	OrderedCollection [Collections-Sequence...	OrderedColle
2	OrderedCollection [Collections-Sequence...	OrderedColle
3	OrderedCollection [Collections-Sequence...	OrderedColle
4	OrderedCollection [Collections-Sequence...	OrderedColle
5	OrderedCollection [Collections-Sequence...	OrderedColle

Showing 8 results, fetched in: 258ms.



## Implementation

# Enhanced Pharo Inspector



OID to write Object centric Time-Traveling Queries

## Implementation

# Enhanced Pharo Inspector

## Object-centric TTQs, example

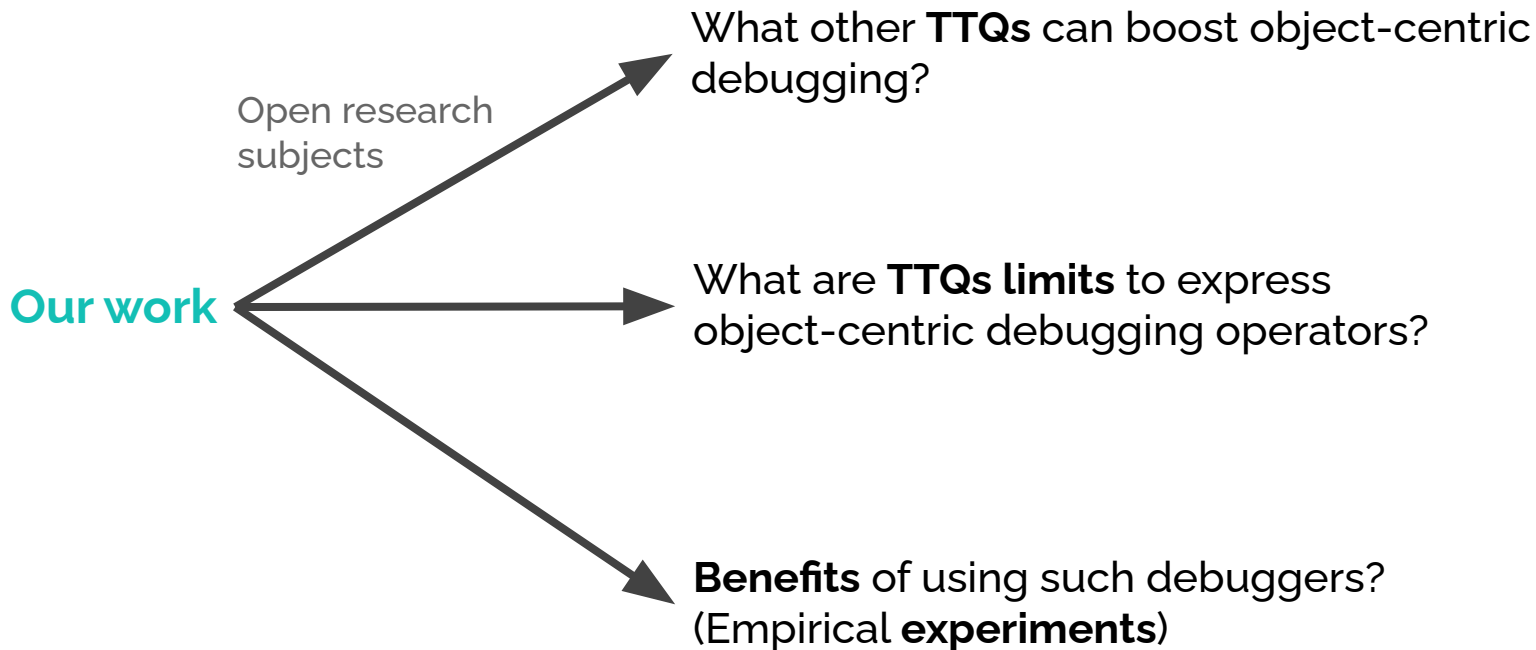
Bound to an "OID", not an object

Variable	Value
{ } self	an OrderedCollection [0 items] ()
▶ { } array	an Array [10 items] (nil nil nil nil nil nil nil nil nil nil nil nil)
▶ Σ firstIndex	1
▶ Σ lastIndex	0

Queries uses OID for selection predicate

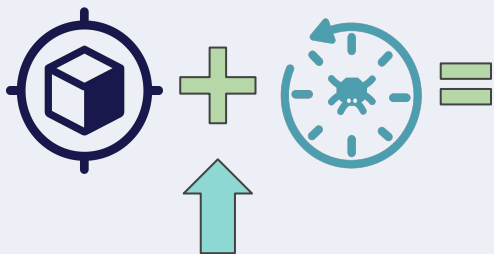
```
makeQueryForInstanceVariableName: aSymbol ofOID: aNumber from: programStates  
  
^ TimeTravelingQuery from: programStates  
  select: [ :state |  
    state nodeIsAssignment and:  
      [ state assignmentVariableName = aSymbol and:  
        [ state contextReceiverOID = aNumber ] ] ]  
  collect: [ :state |  
    ResultItem new  
      step: state bytecodeIndex;  
      variable: aSymbol;  
      oldValue: state assignmentCurrentValue;  
      newValue: state assignmentNewValue;  
      yourself ]
```

# Perspective



# Towards Object-centric Time-traveling debuggers

## Contribution



*Time-Traveling Queries* ✓

2. How to join both techniques

1. New OOP Debugging Tools

*SeekerOC* ✓  
&  
*Enhanced* ✓  
*Pharo*  
*Inspector*



Our work

Open research subjects

Other TTQs for OC debugging? ✓

TTQs limitations? ✓

Benefits? (Experiments) ✓

## Perspective