Memory Profiling in Pharo

Sebastian JORDAN MONTAÑO

sebastian.jordan@inria.fr

Inria, Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL

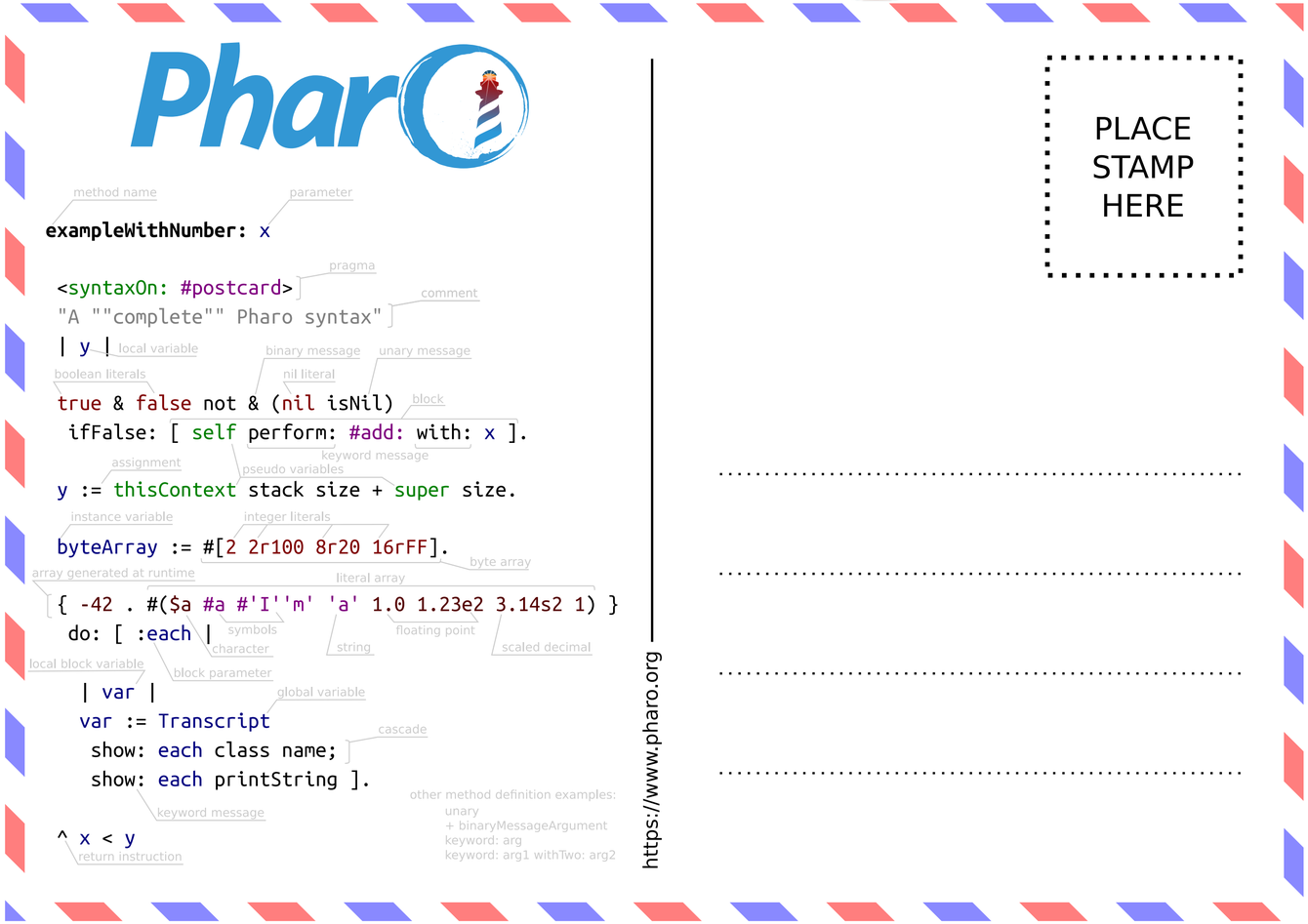
November 2023

About me

- **Bachelor's:** Software Engineer, UCB, Bolivia
- **Master's:** Software Engineering, UL, France
- **PhD:** Started in 2023 in memory profiling
- **Interests:** Music (progressive rock, Charly), (real) languages, sports, Pharo

Pharo: a programming language

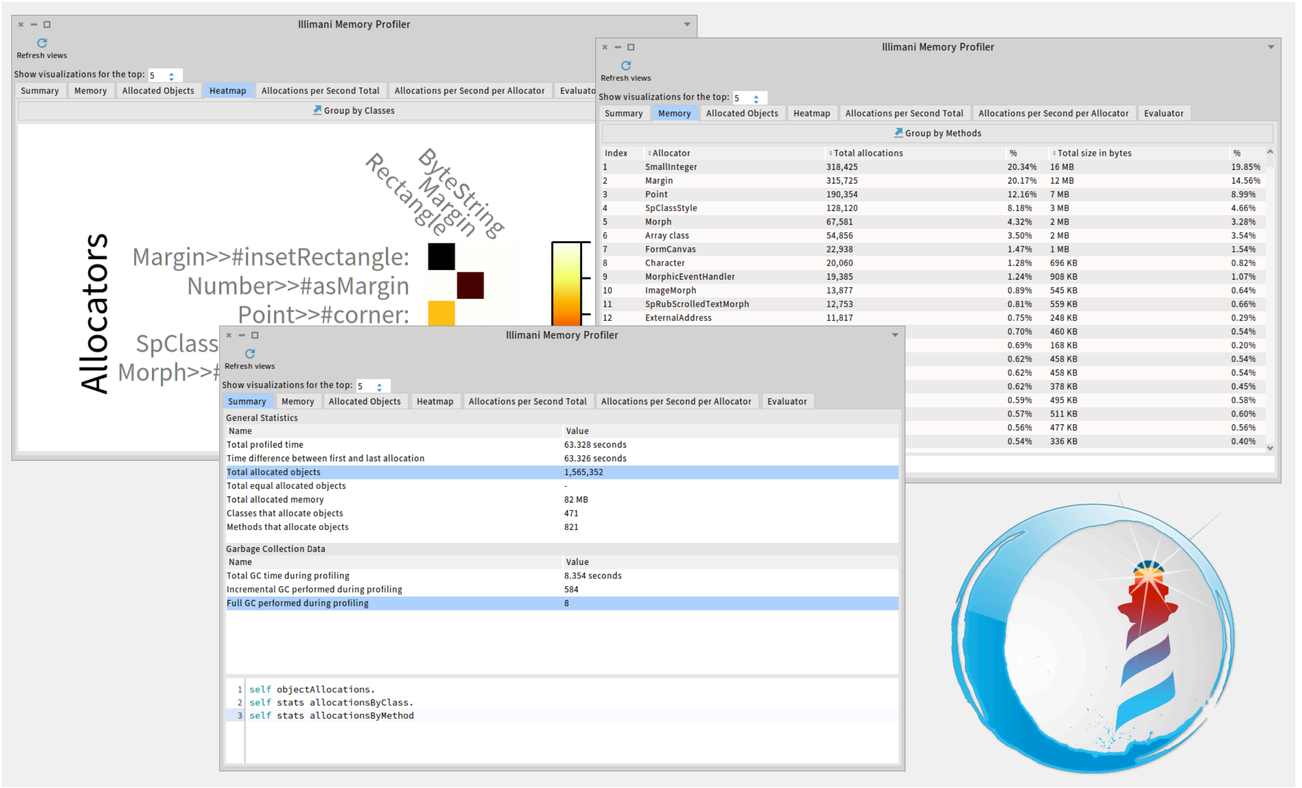
- Advanced run time reflection
- Pure object-oriented approach
- Software as objects
- (Super) simple syntax
- Pharo, and its VM, are written in Pharo
- MIT open-source license
- Live inspectable objects
- Easy stack manipulation



pharo.org

Illimani: a Pharo memory profiler

- Open-source MIT license
- Detects object allocation sites
- Tracks object lifetimes
- Allocation matrix
- Unmodified VM
- Density chart
- Memory consumption tables
- Rich object-oriented model



github.com/jordanmontt/illimani-memory-profiler

Object allocation sites

We define an object allocation site as the textual location in the source code where the object was created [1]

```
AthensTextScanner >> initialize

    lines := OrderedCollection new
    ...
```

Allocation site

[1] **Memento Mori: Dynamic Allocation-Site-Based Optimizations**

Daniel Clifford Hannes Payer Michael Stanton Ben L. Titzer

Capture object allocation sites

In Pharo, almost all computations are done by sending a message. This is also true when allocating objects.

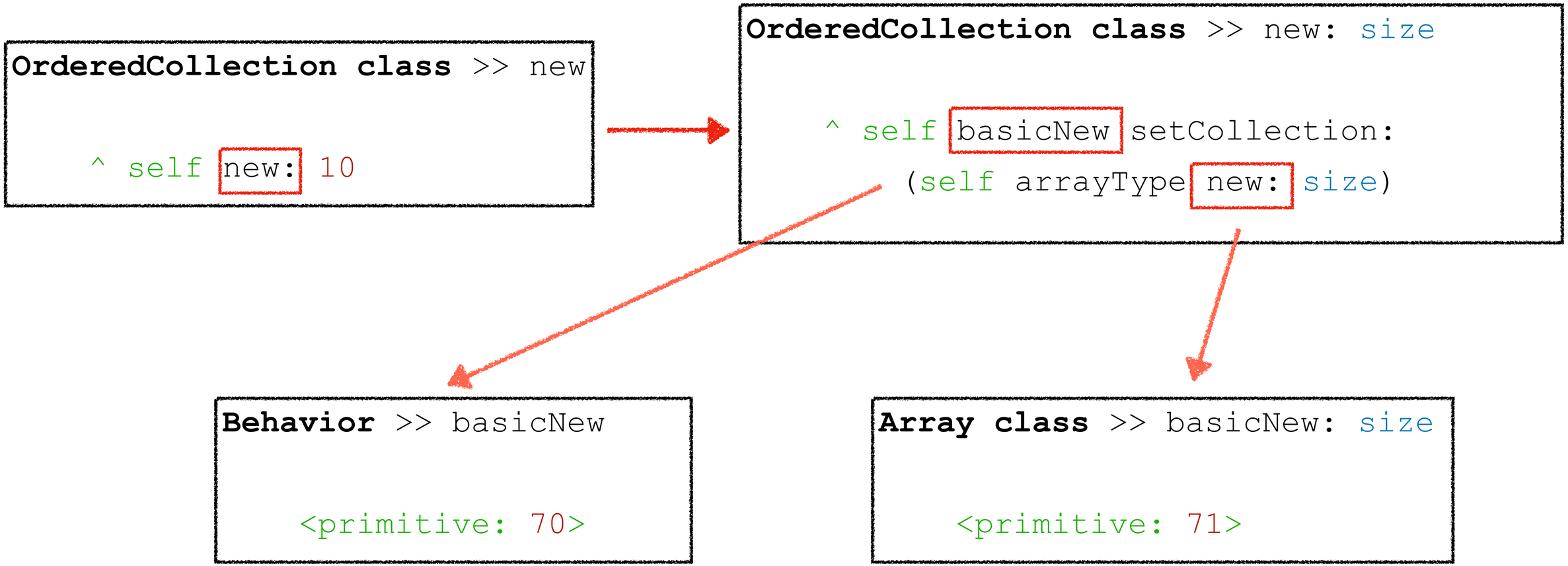
```
collection := OrderedCollection new.
```


```
list = new ArrayList();
```

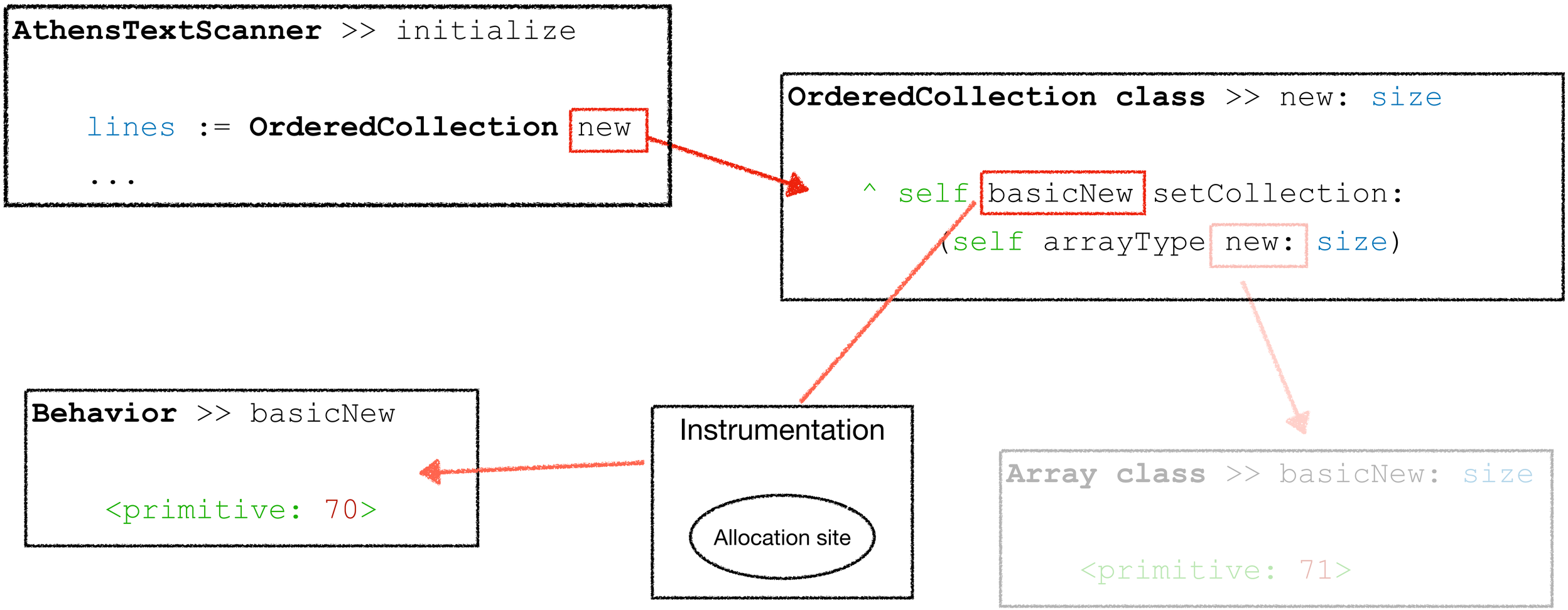
Allocating an object

```
OrderedCollection new
```

Allocating an object



Capture object allocation sites



Allocating an object

```
AthensTextScanner >> initialize

    lines := OrderedCollection new
    ...
```

```
OrderedCollection class >> new: size

    ^ self basicNew setCollection:
        (self arrayType new: size)
```

```
Behavior >> basicNew

    <primitive: 70>
```

```
Array class >> basicNew: size

    <primitive: 71>
```

Instrumentation

Allocation site

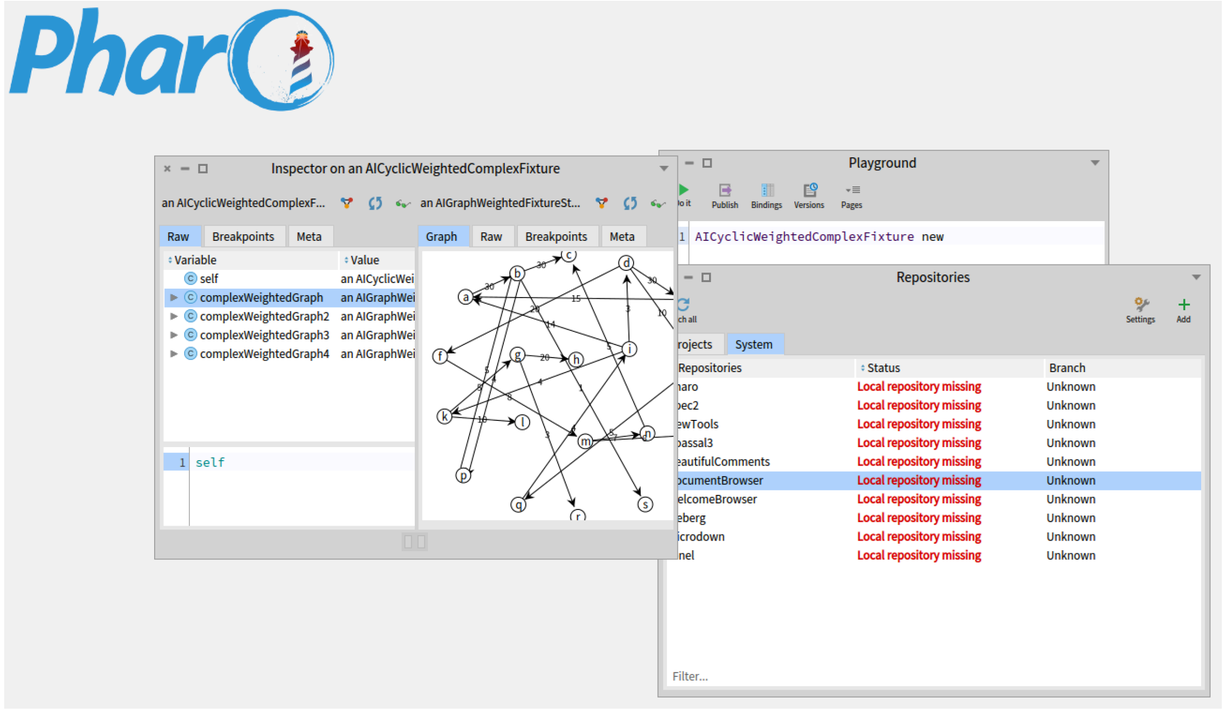
Instrumenting object allocation sites in Pharo

We've instrumented 7 methods that allocate objects to capture when they will be called and then filter the execution stack.

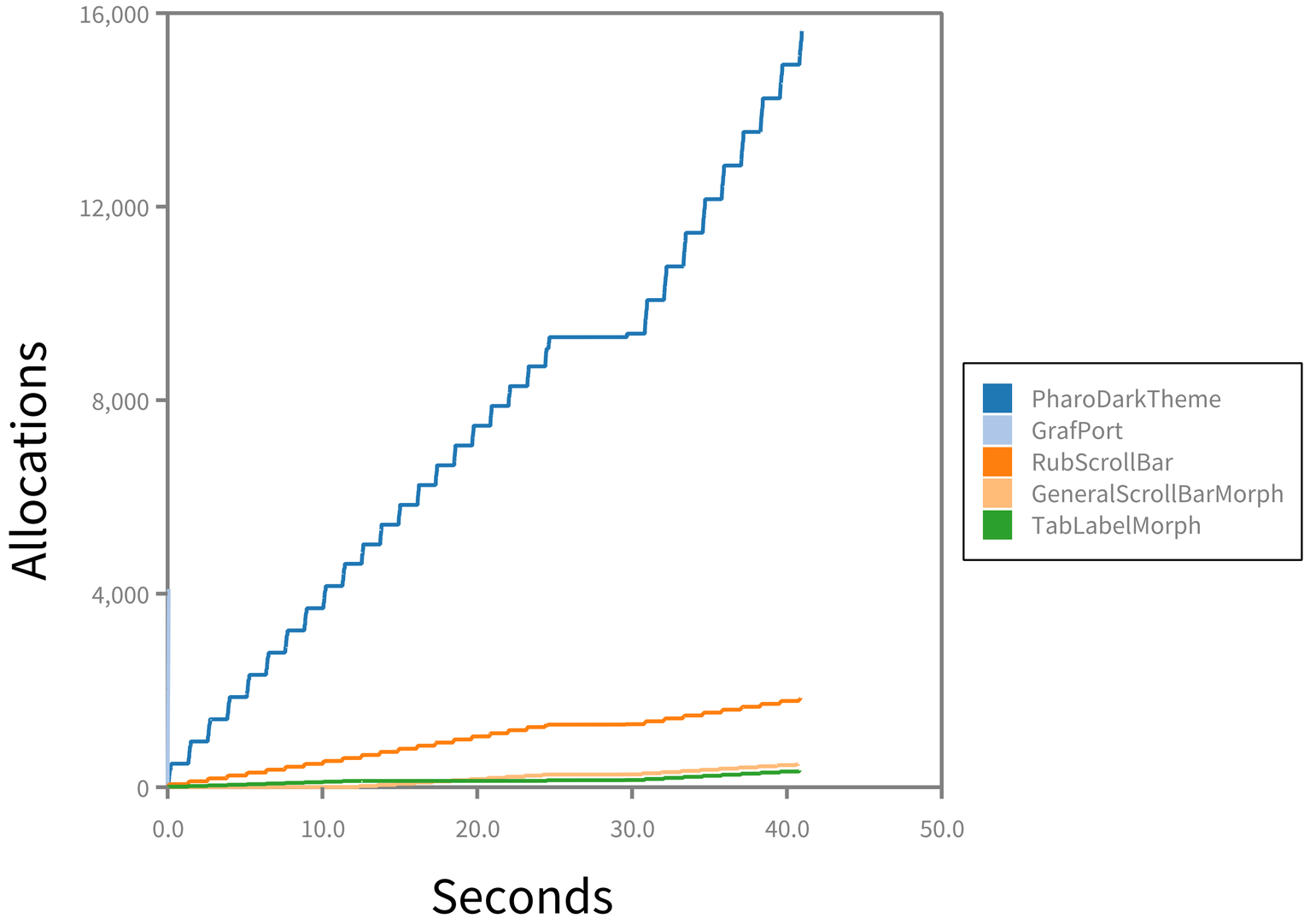
- `Behavior >> basicNew`
- `Behavior >> basicNew:`
- `Behavior >> basicPinned`
- `Behavior >> basicPinned:`
- `Behavior >> basicOld`
- `Behavior >> basicOld:`
- `Array class >> new:`

Case study 1: Morphic

- We profiled the allocations of type **`Color`**
- We opened 30 Pharo tools (Inspector, Playground and Iceberg) and we let them render for 100 rendering cycles



Allocations over time

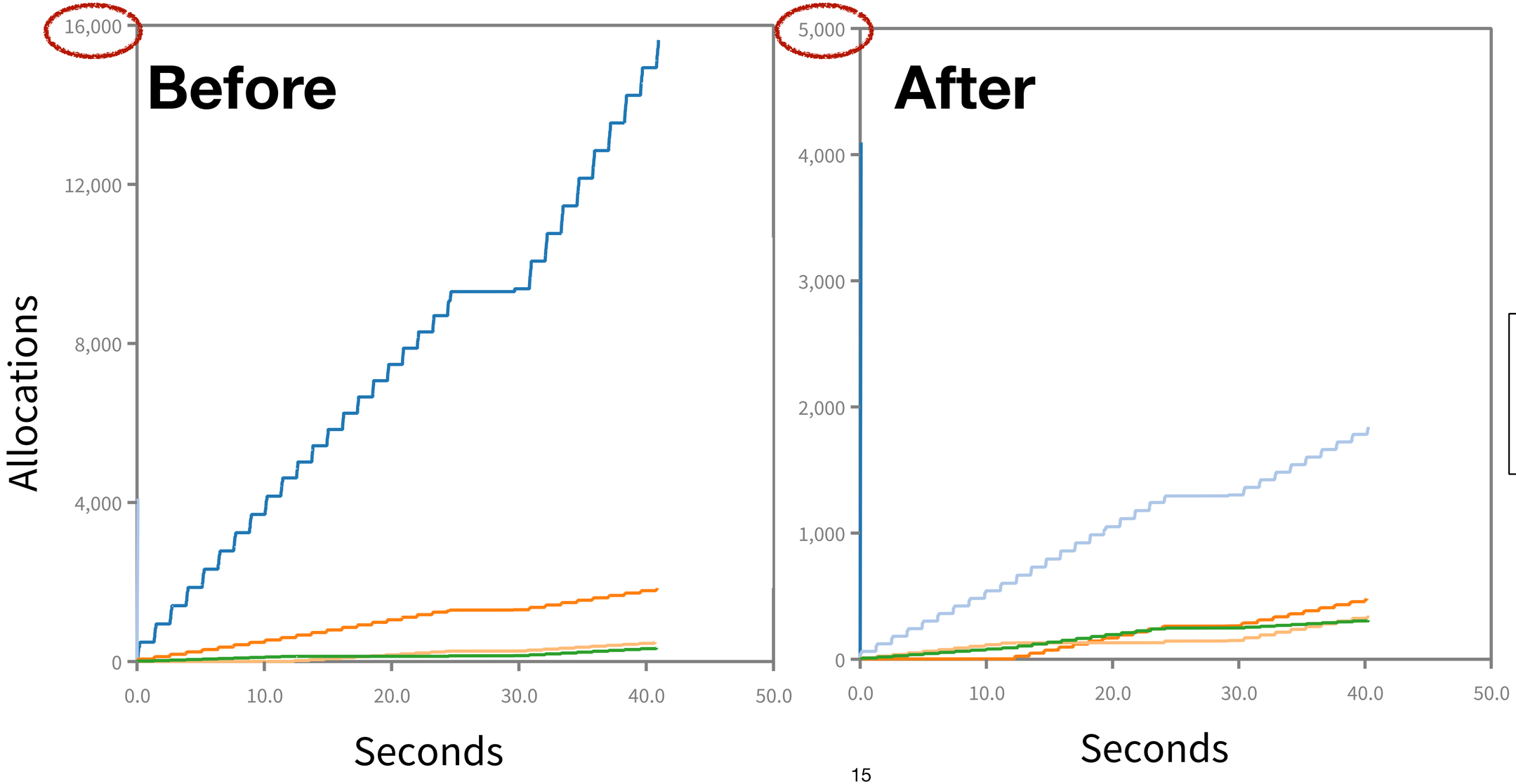


Morphic Analysis

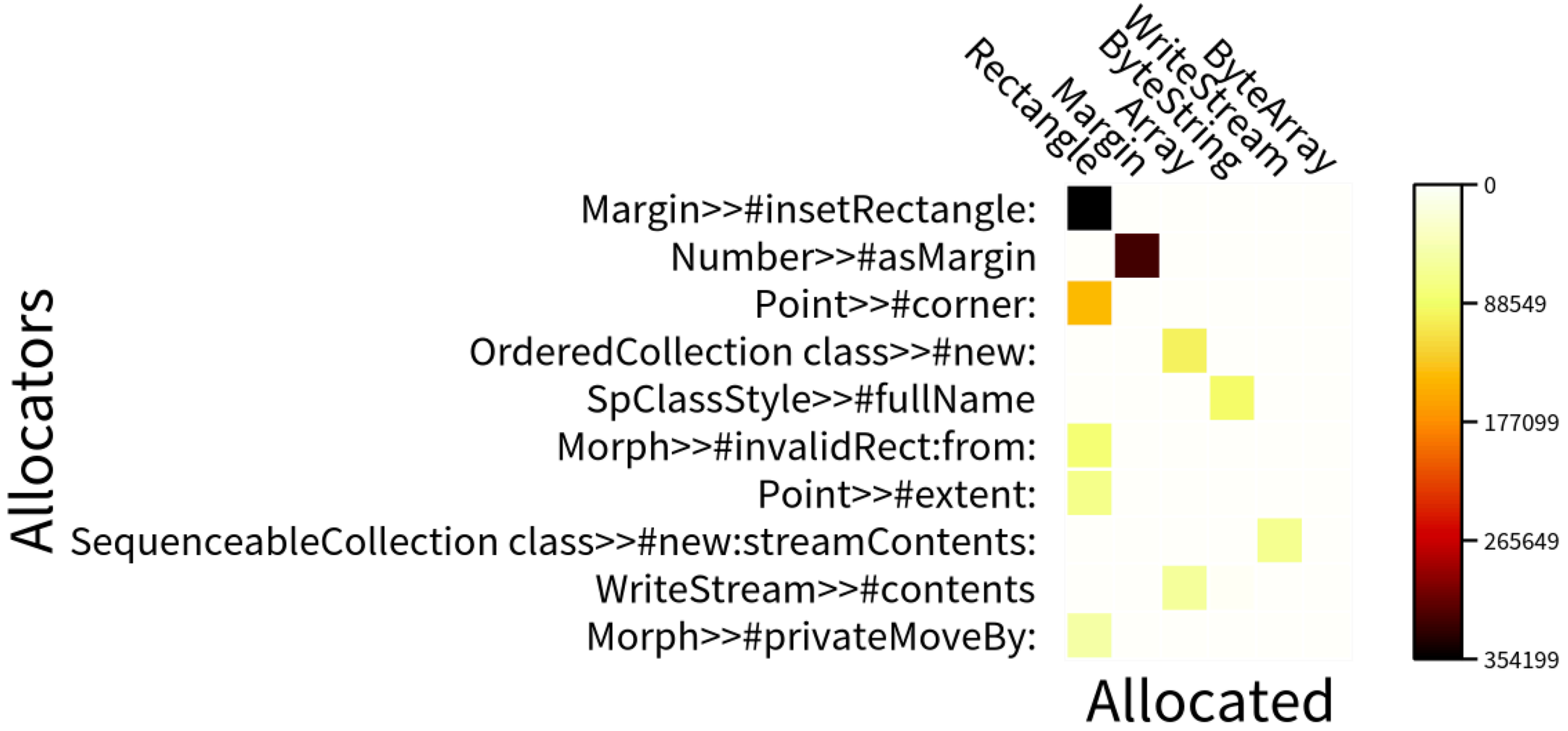
Allocator class	Allocated colors	%
PharoDarkTheme	15,629	66%
GrafPort	4,096	17%
RubScrollBar	1,842	8%
GeneralScrollBarMorph	480	2%
TabLabelMorph	346	1%
Rest of the classes	1293	2%

We have identified an object allocation site in the class `PharoDarkTheme` that allocated 66% of all the allocated colors with 99,9% redundant allocations.

Morphic after the fix

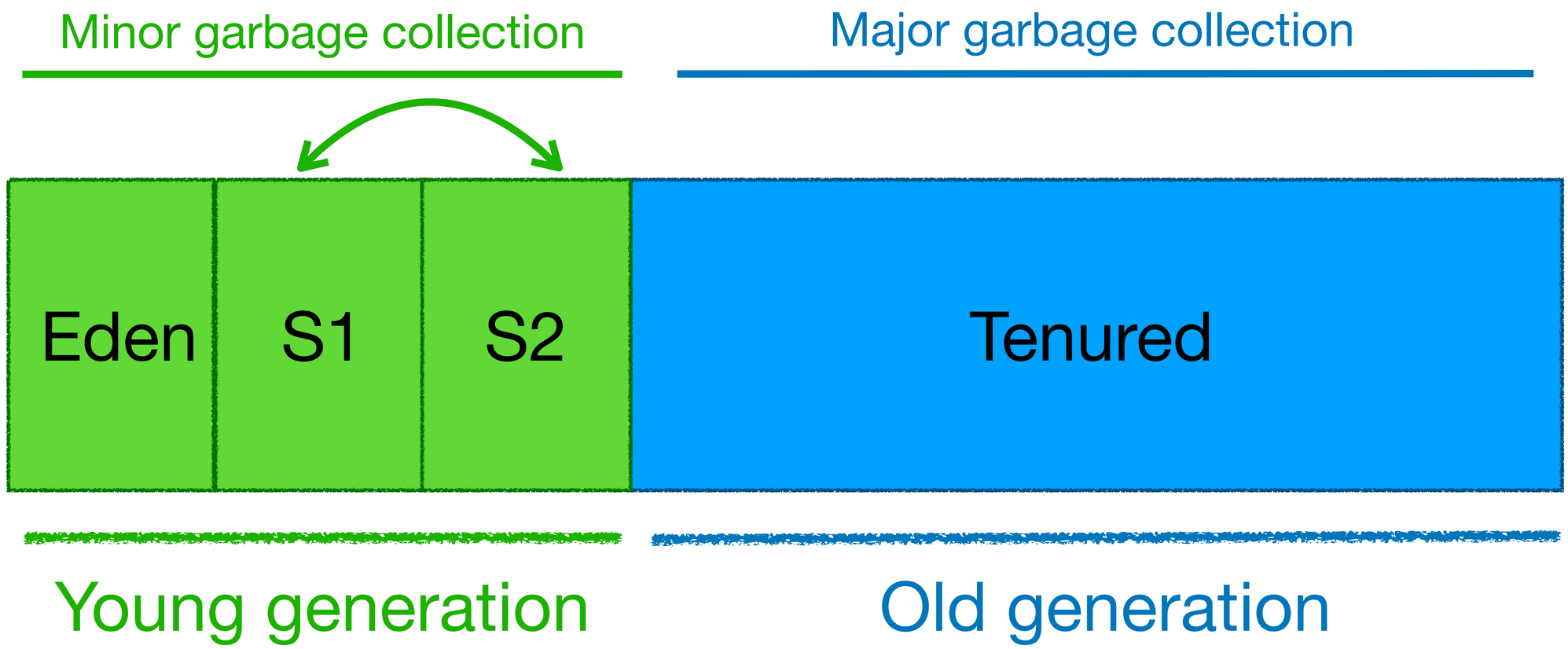


Detecting other allocation sites



Case study 2: Object lifetimes

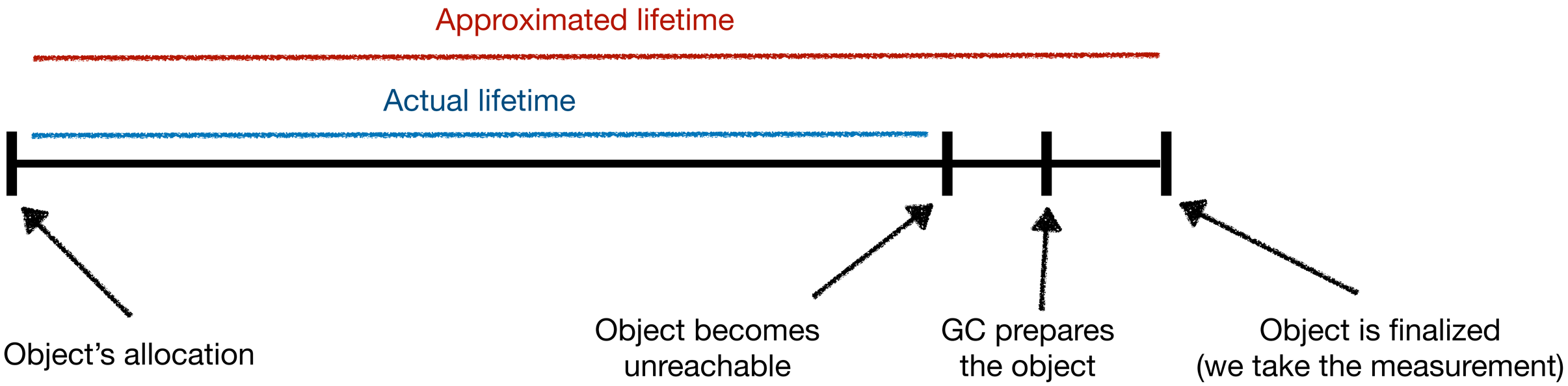
Pharo’s garbage collector



An object’s (approximated) lifetime

object’sLifetime = finalizationTime - allocationTime

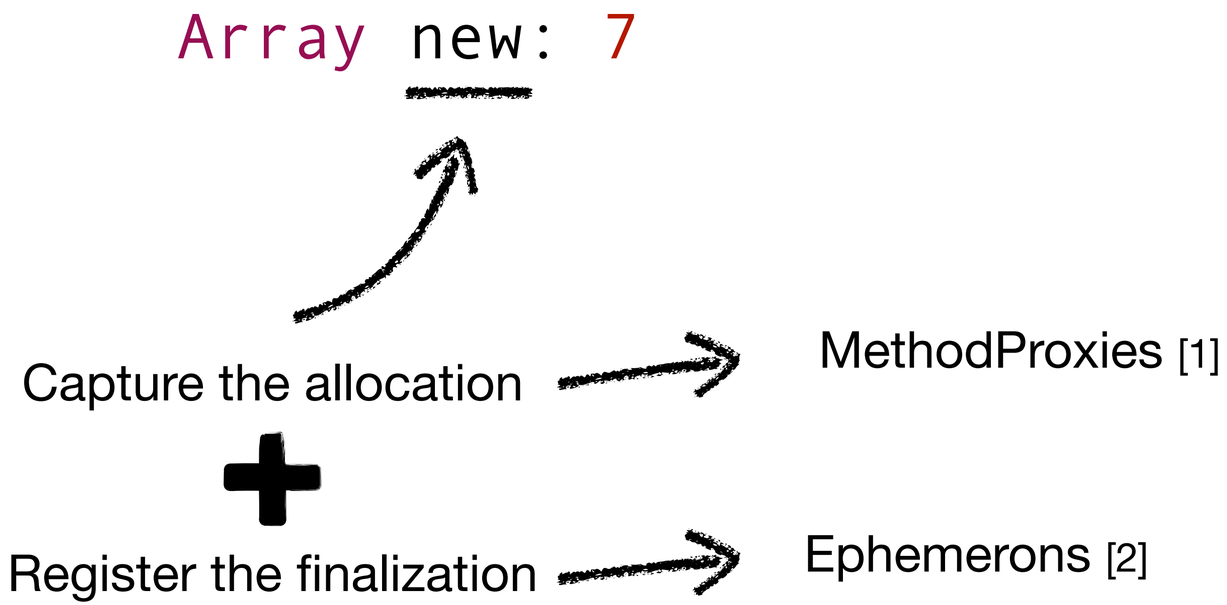
An object’s lifetime



Capturing the allocations

```
Array new: 7
```

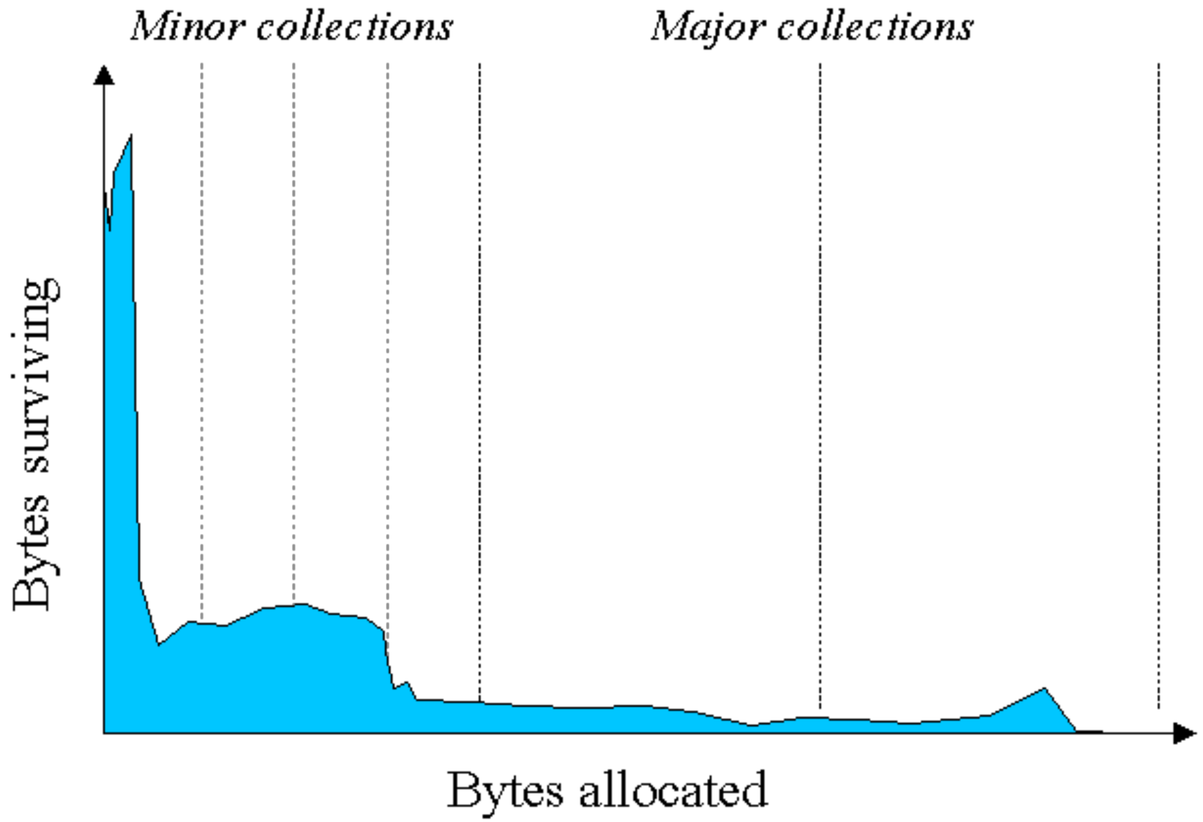
Capturing the allocations



[1] github.com/pharo-contributions/MethodProxies

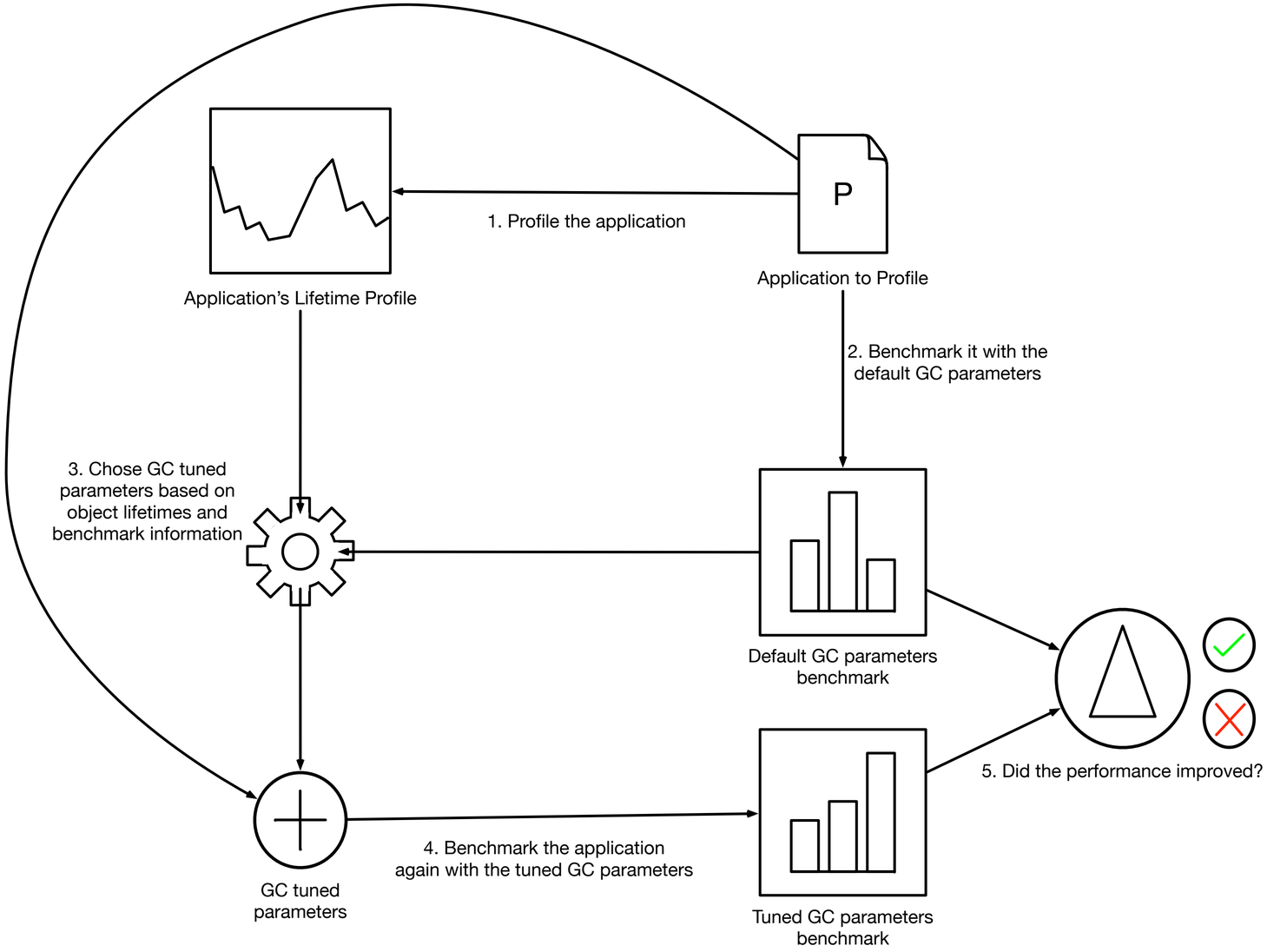
[2] github.com/pharo-project/pheps/blob/main/phep-0003.md

A common object lifetime distribution



Source: oracle.com

Case study 2: methodology



Target application: DataFrame

PolyMathOrg/
DataFrame

DataFrame in Pharo - tabular data structures for data analysis

12 Contributors

37 Issues

67 Stars

21 Forks

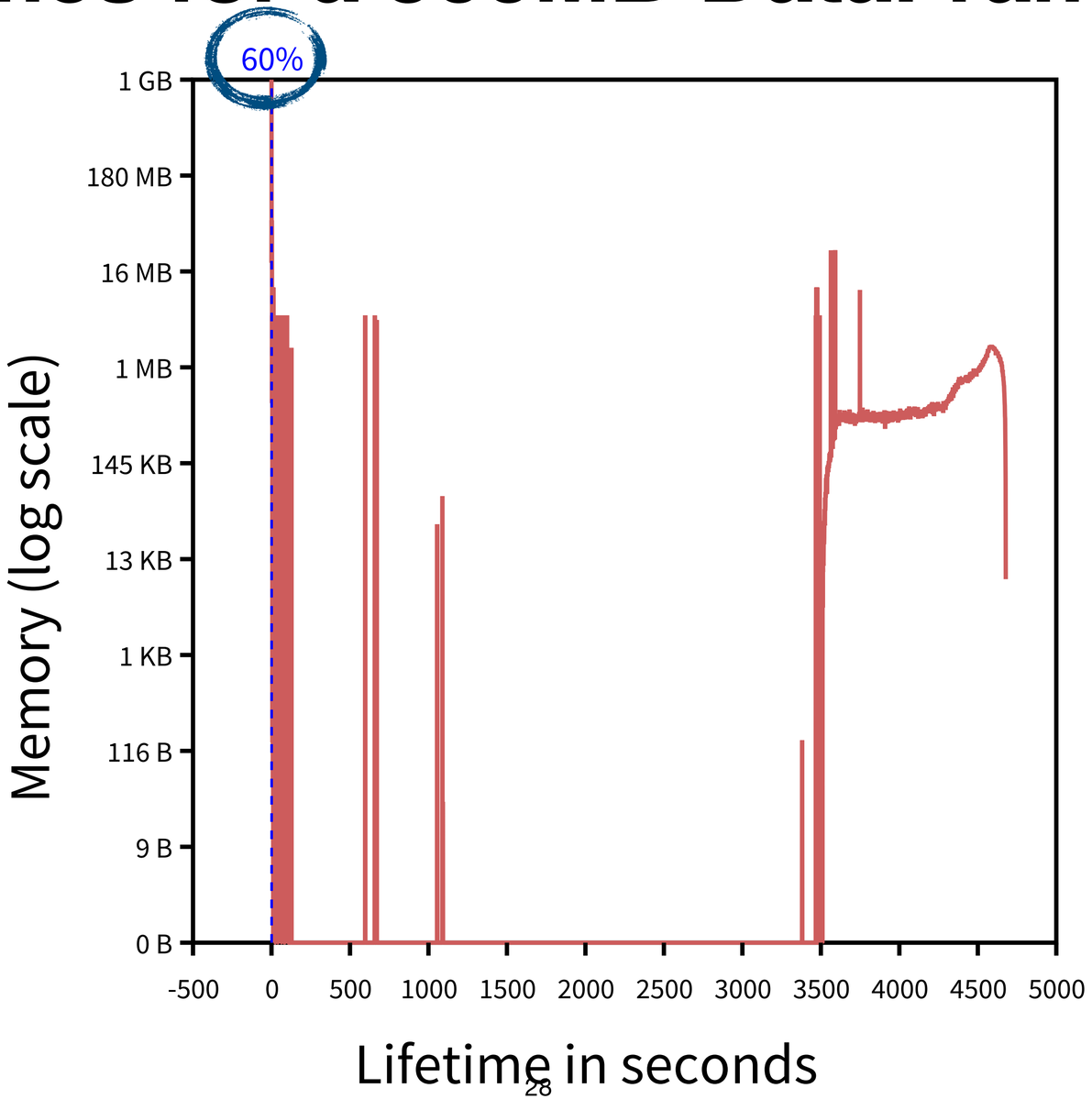
Benchmarking DataFrame

Dataset	# of scavengers	# of full GCs	GC time	Total time	GC time in %
500 MB	266	18	11 sec	71 sec	15%
1.6 GB	304	36	60 sec	248 sec	22%
3.1 GB	1143	309	3793 sec	4265 sec	89%

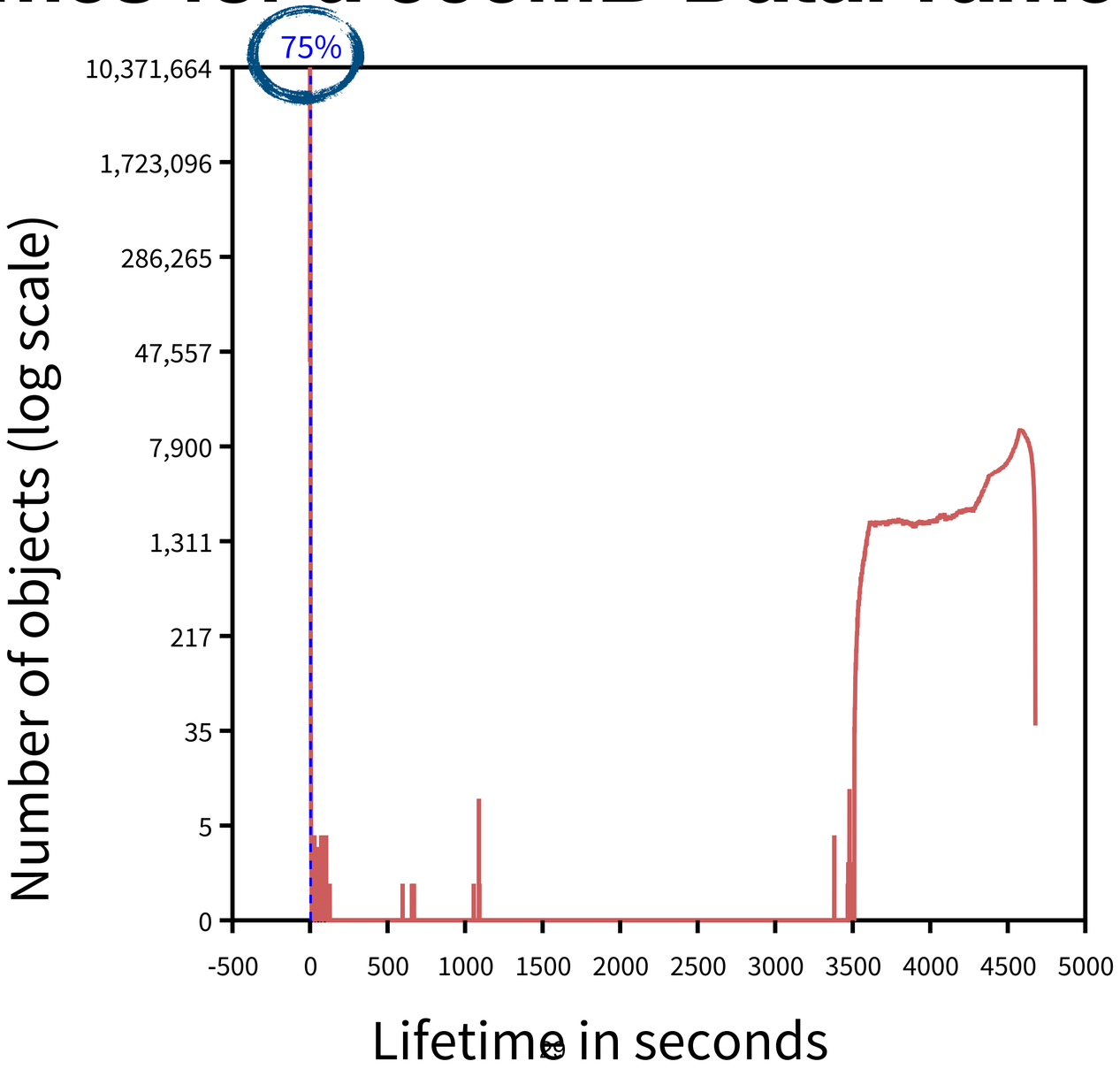
Benchmarking DataFrame

Dataset	# of scavengers	# of full GCs	GC time	Total time	GC time in %
500 MB	266	18	11 sec	71 sec	15%
1.6 GB	304	36	60 sec	248 sec	22%
3.1 GB	1143	309	3793 sec	4265 sec	89%

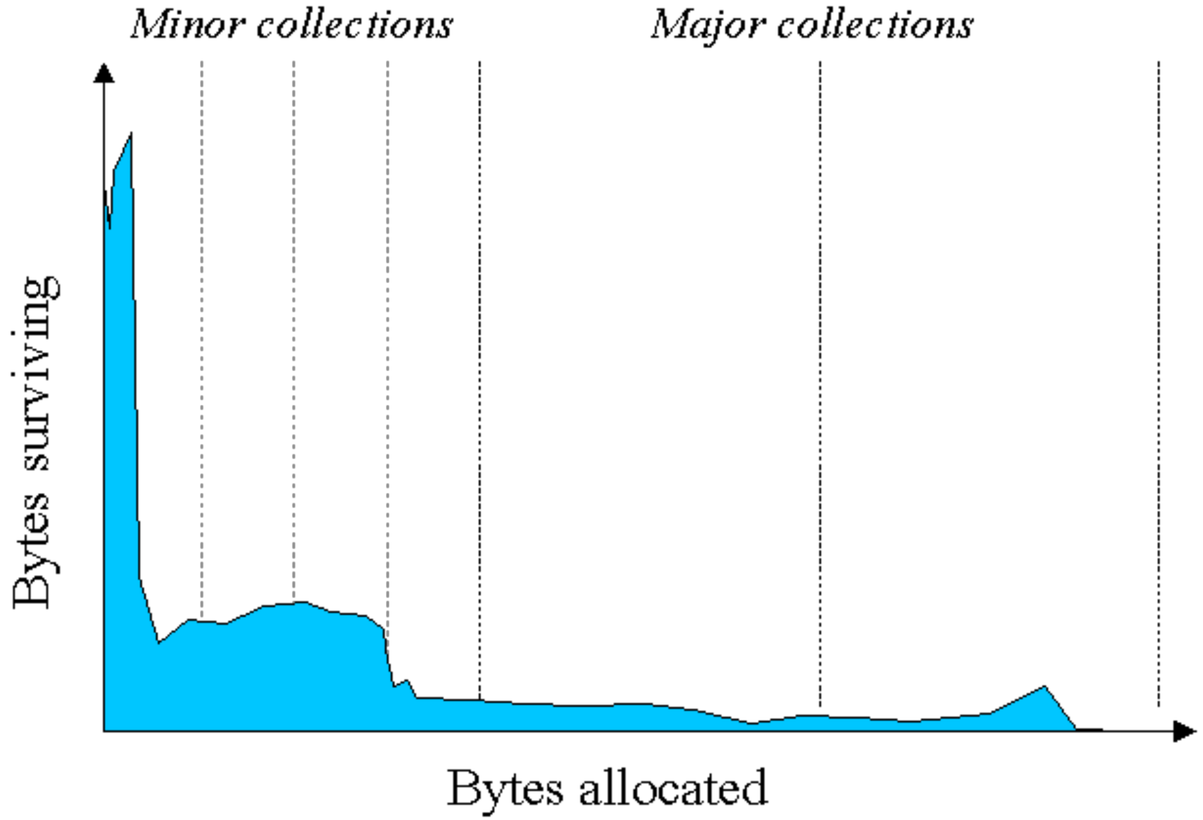
Object lifetimes for a 500MB DataFrame (memory)



Object lifetimes for a 500MB DataFrame (# objects)

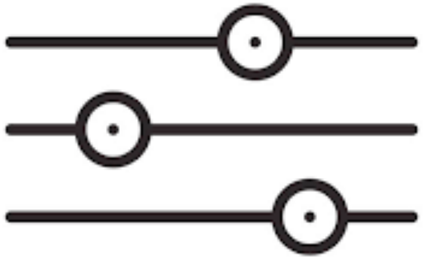


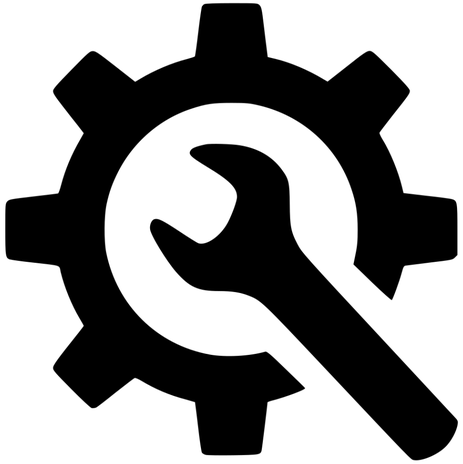
A common object lifetime distribution



Source: oracle.com

GC parameters





DataFrame performance improvements

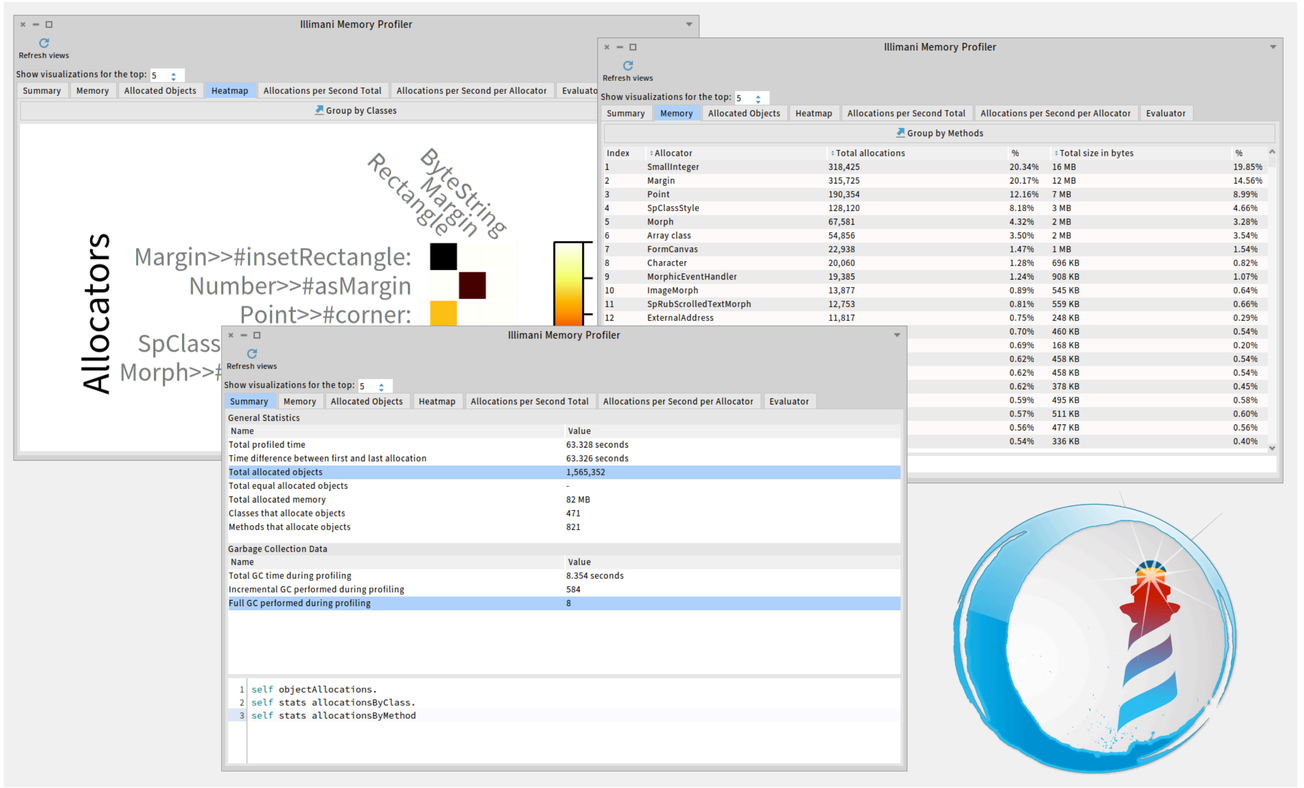
GC Configuration	GC spent time	Total execution time	Improved performance
Default	58 min 18 sec	1 hour 6 min 18 sec	1×
Configuration 1	9 min 41 sec	17 min 46 sec	3.7×
Configuration 2	4 min 57 sec	12 min 54 sec	5.1×
Configuration 3	5 min 8 sec	13 min 2 sec	5.1×
Configuration 4	2 min 42 sec	10 min 37 sec	6.2×
Configuration 5	1 min 47 sec	9 min 42 sec	6.8×

The future

- Study the precision of the approximated lifetimes.
- Dynamic optimizations based on allocation sites.

Illimani: a Pharo memory profiler

- Open-source MIT license
- Detects object allocation sites
- Tracks object lifetimes
- Allocation matrix
- Unmodified VM
- Density chart
- Memory consumption tables
- Rich object-oriented model



Sebastian JORDAN MONTAÑO

sebastian.jordan@inria.fr

github.com/jordanmontt/illimani-memory-profiler