Memory Profiling

Sebastian JORDAN MONTAÑO

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

sebastian.jordan@inria.fr








Octobre 2024

About me

- **I’m doing a PhD** on memory profiling
- **I like** Music (progressive rock, Charly SAY NO MORE LOCO), real languages, sports (lightweight baby!!)
- Average **Pharo** enjoyer
- **Professional hater**

* Real picture

Message-passing

Sending a message

In OOP, (almost) all computations are done by sending a message.

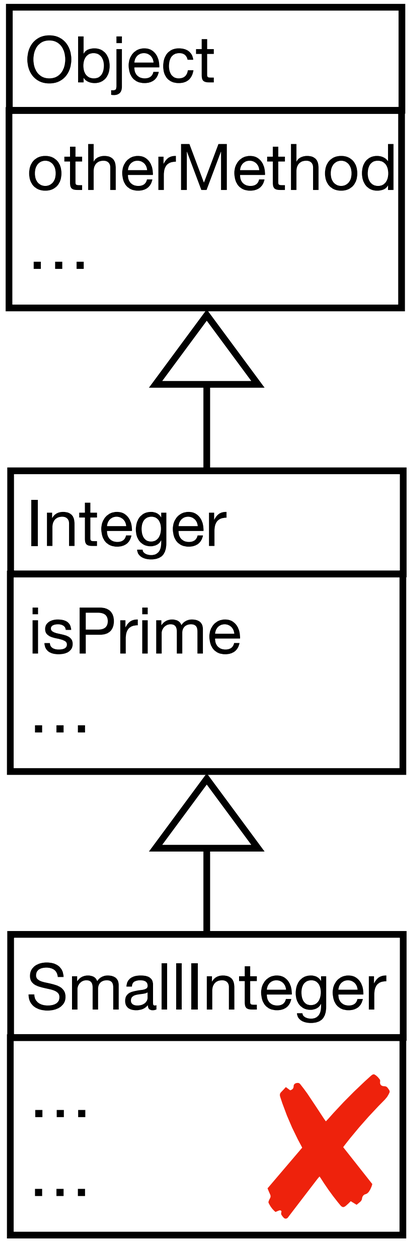
The lookup algorithm

```
179281 isPrime
```

Instance of `SmallInteger`

Pharo's VM

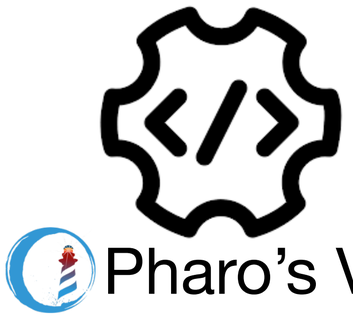
Searches the compiled method



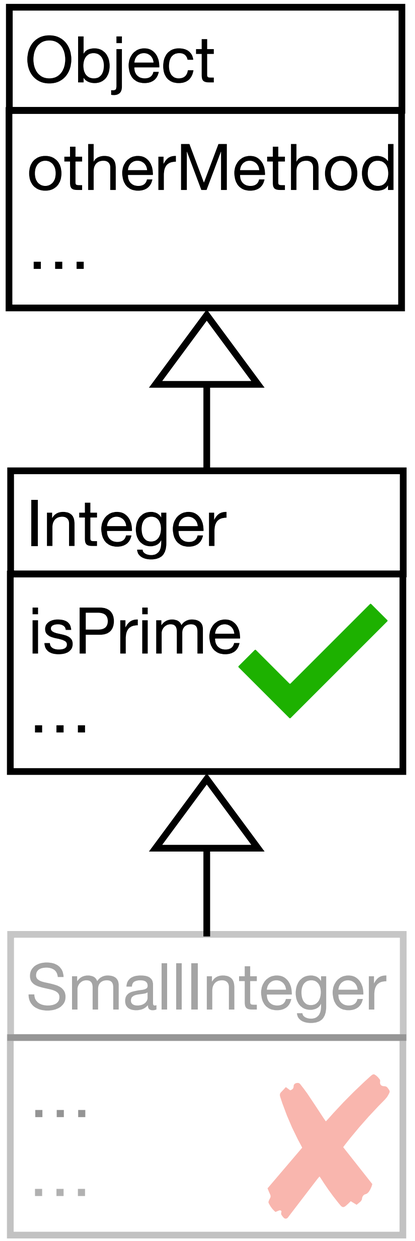
The lookup algorithm

```
179281 isPrime
```

Instance of `SmallInteger`



Searches the compiled method



Objects communicate through messages

1. The message `#method:` is sent to `aReceiver`

```
aReceiver method: args
```

Message-passing

2. Executes `#method:` into the object `aReceiver`

Message-passing control

Objects communicate through messages

1. The message `#method:` is sent to `aReceiver`

```
aReceiver method: args
```

Message-passing

2. Executes `#method:` into the object `aReceiver`

Pharo's VM

Message-passing control

The message is captured

1. The message `#method:` is sent to `aReceiver`

```
aReceiver method: args
```

2. A user-defined action is executed before the method's execution

#beforeAction

3. Executes `#method:` into the object `aReceiver`

Pharo's VM

4. An action is executed after the execution

#afterAction

Safe Message-Passing Control

- Meta-safe recursion
- Thread safety
- Safe handling of exceptions and non-local returns
- Uninstrumentation

MethodProxies

MethodProxies: A Safe and Fast Message-Passing Control Library

Sebastian Jordan Montaño[1], Juan Pablo Sandoval Alcocer[2], Guillermo Polito[1], Stéphane Ducasse[1] and Pablo Tesone[1]

[1]*Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL, Park Plaza, Parc scientifique de la Haute-Borne, 40 Av. Halley Bât A, 59650 Villeneuve-d'Ascq, France*

[2]*Department of Computer Science, School of Engineering, Pontificia Universidad Católica de Chile, Santiago, Chile*

Abstract

The injection of monitoring code allows for real-time observation of the program, which has proven instrumental in developing tools that assist developers with various programming tasks. In dynamic languages such as Pharo, renowned for their rich meta-programming capabilities and dynamic method dispatch, such monitoring capabilities are particularly valuable. Message-passing control techniques are commonly used to monitor program execution at the method level, involving the execution of specific code before and after each method invocation. Implementing message-passing control techniques, however, poses many challenges, notably in terms of instrumentation overhead. Additionally, it is crucial for the message-passing mechanism to be safe: *i.e.,* to accommodate recursive and reflective scenarios to ensure that it does not alter the execution of the monitored program, which could potentially lead to infinite loops or other unintended consequences.

Over the years, numerous techniques have been proposed to optimize message-passing control. This paper introduces MethodProxies, a message-passing instrumentation library that offers minimal overhead and is safe. We conduct a comparison between MethodProxies and two commonly used techniques implemented in the Pharo programming language: method substitution using the run:with:in: hook and source code modification. Our results demonstrate that MethodProxies offers significantly lower overhead compared to the other two techniques and is safe against infinite recursion.

Keywords

instrumentation, message-passing control, error handling, method compilation

- Implementation details of our (super) instrumentation library

/pharo-contributions/MethodProxies

Capturing allocations

Capture object allocation sites

In OOP, (almost) all computations are done by sending a message. In Pharo, this is also true when allocating objects.

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


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

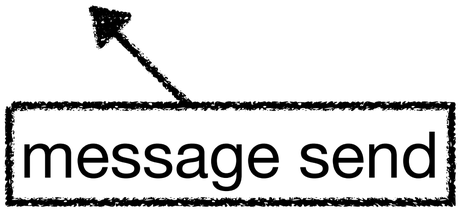
Meme time

Capture object allocation sites

In OOP, (almost) all computations are done by sending a message. In Pharo, this is also true when allocating objects.


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

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



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

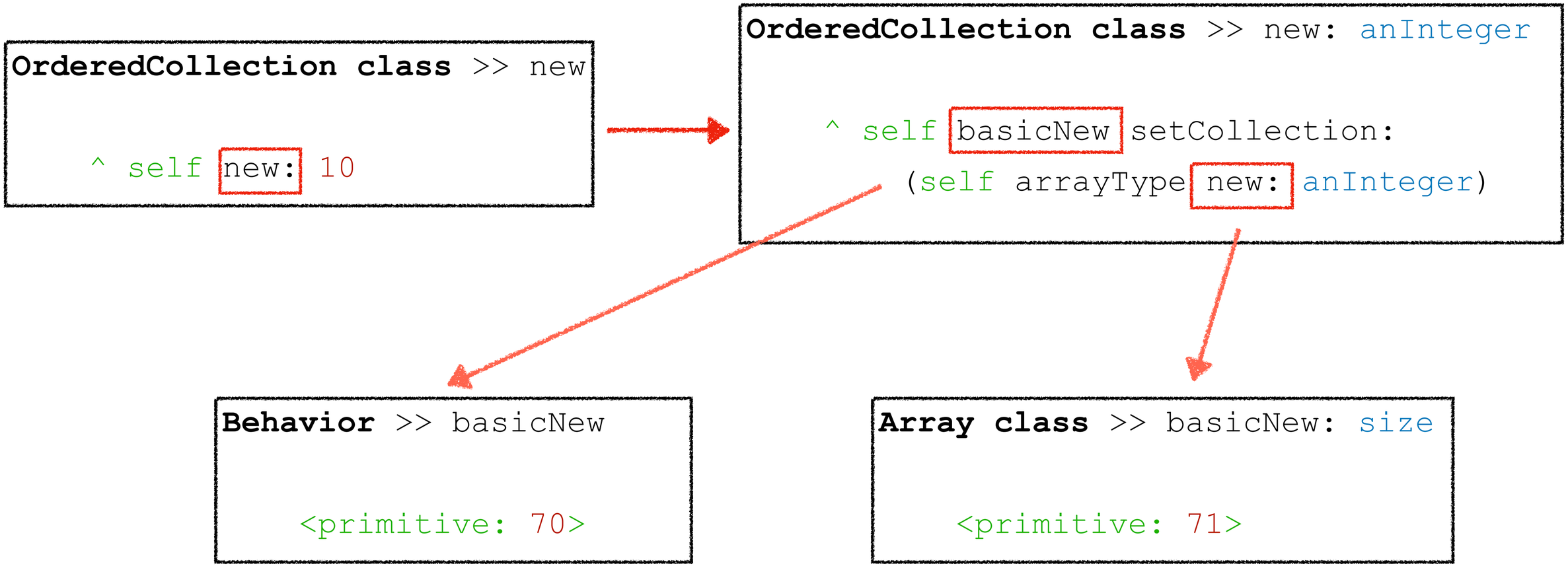
[1] **Memento Mori: Dynamic Allocation-Site-Based Optimizations**

Daniel Clifford Hannes Payer Michael Stanton Ben L. Titzer

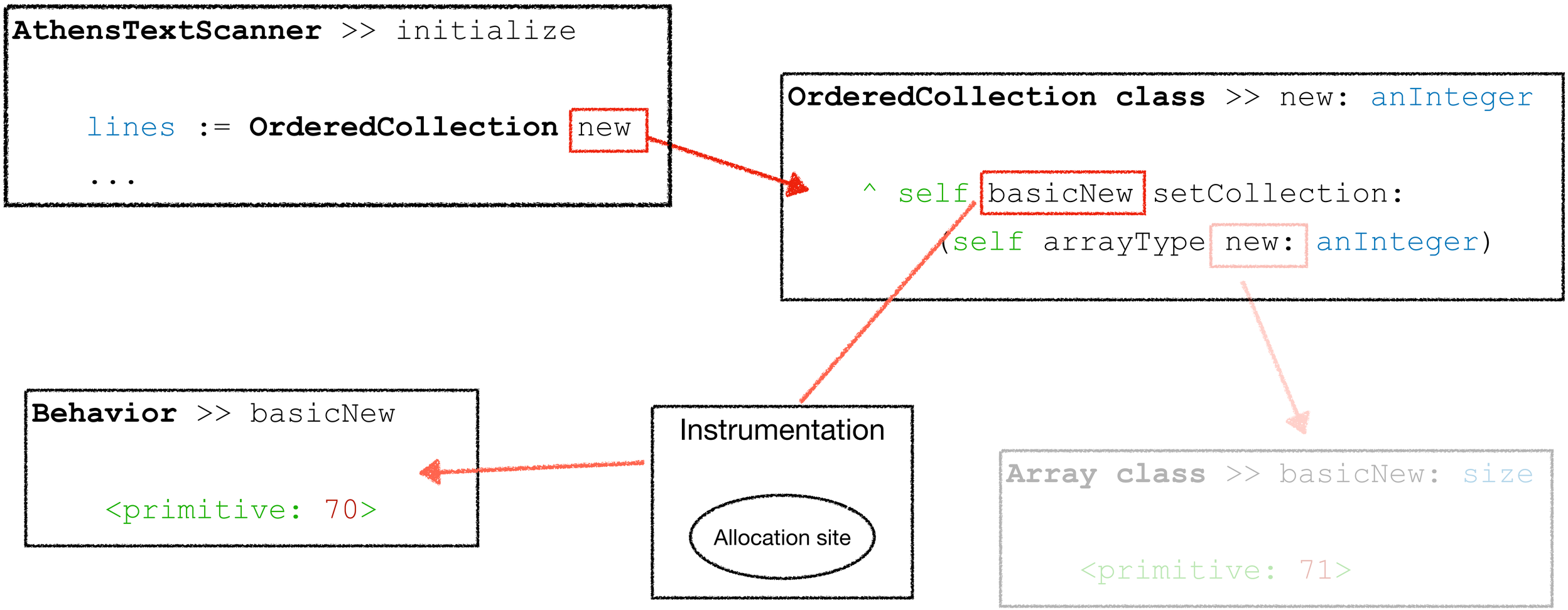
Allocating an object

```
OrderedCollection new
```

Allocating an object



Capture object allocation sites



Allocating an object

```
AthensTextScanner >> initialize

    lines := OrderedCollection new
    ...
```

```
OrderedCollection class >> new: anInteger

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

```
Behavior >> basicNew

    <primitive: 70>
```

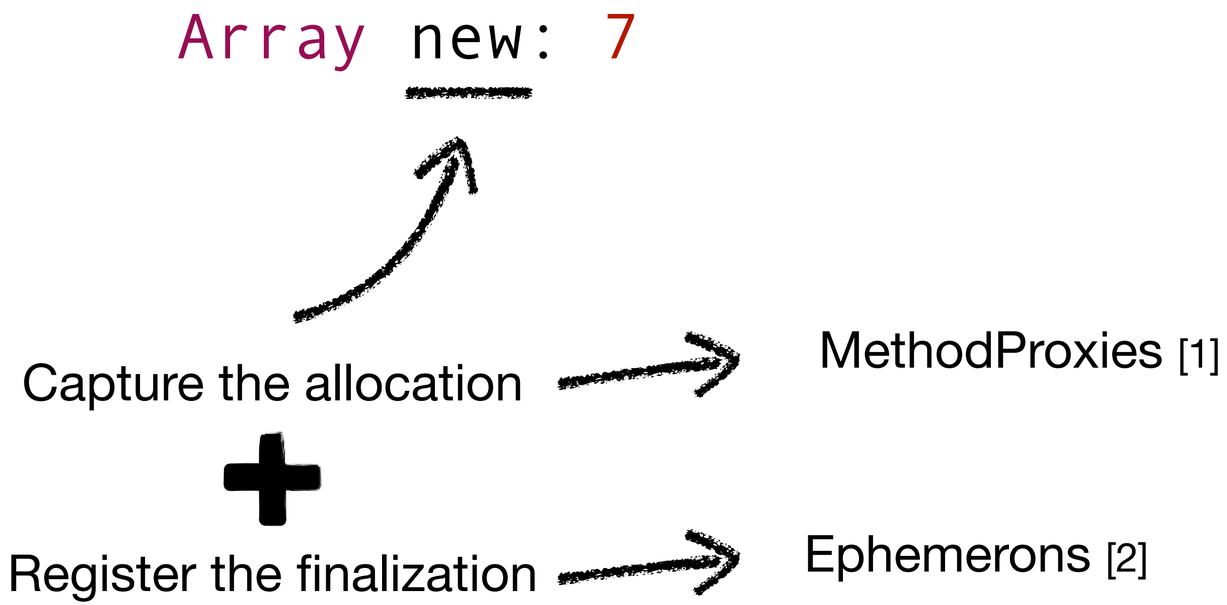
```
Array class >> basicNew: size

    <primitive: 71>
```

Instrumentation

Allocation site

Capturing the allocations

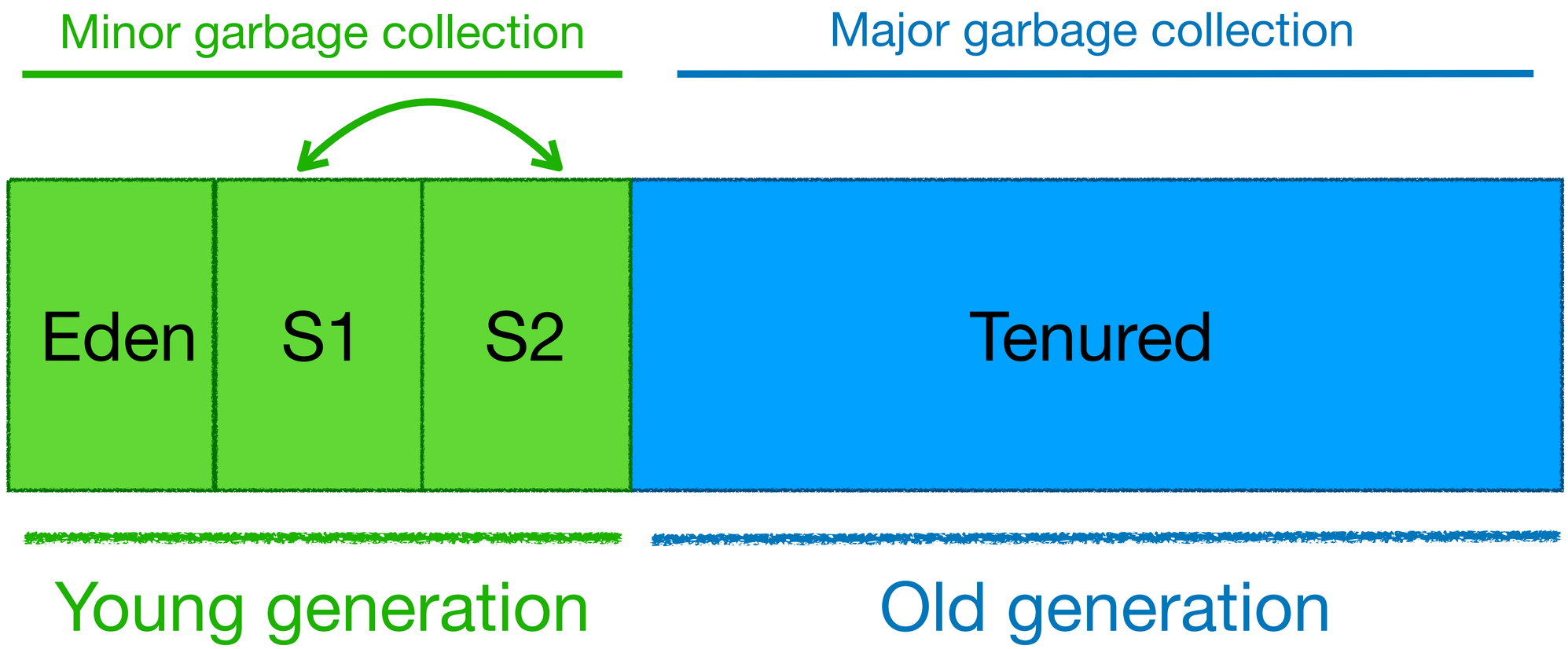


[1] github.com/pharo-contributions/MethodProxies

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

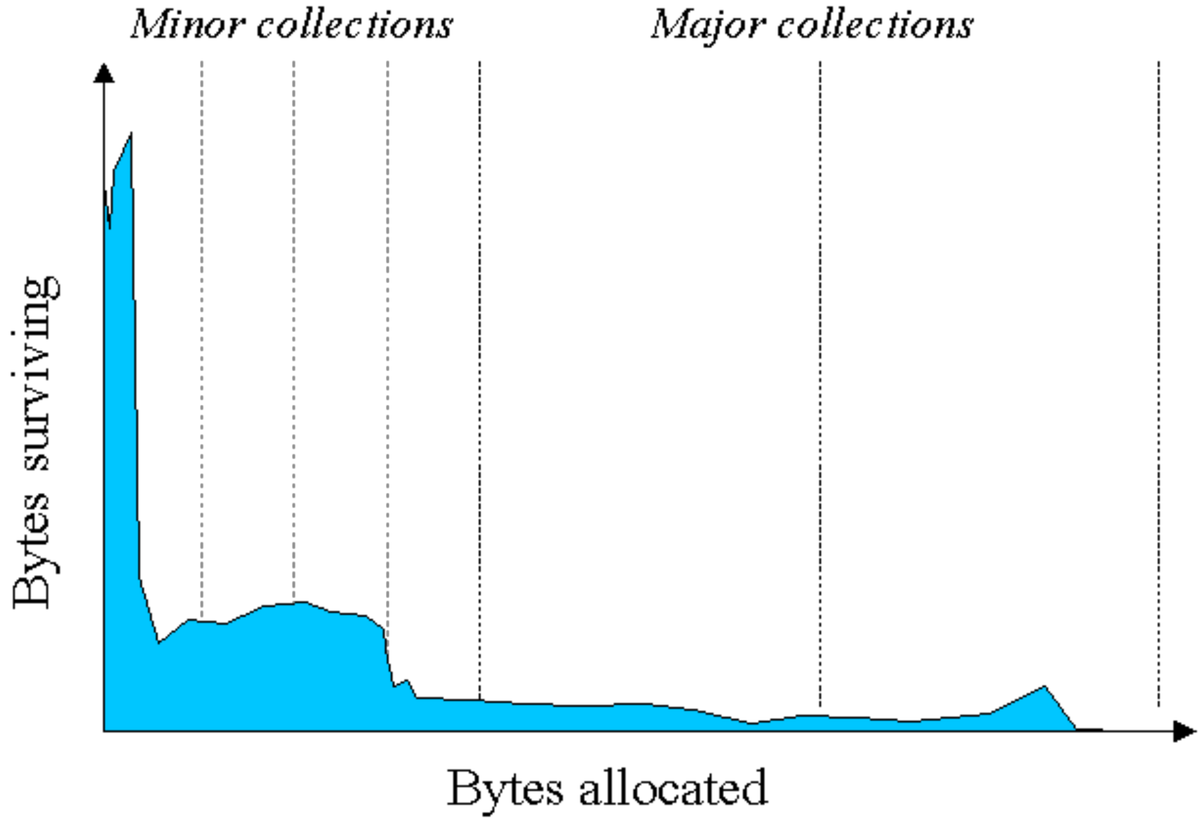
Object lifetime profiling

Generational Garbage Collectors



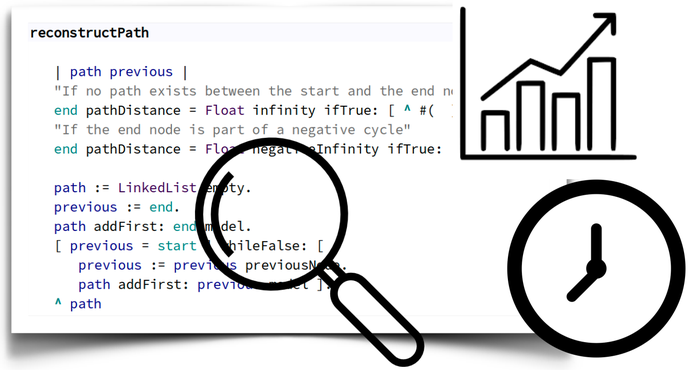
[1] Generation Scavenging: A Non-Disruptive High Performance Storage Reclamation Algorithm, Ungar et al., 1984

Common lifetimes distribution for reference



Source: oracle.com

Advantages of Object Lifetime Profiling



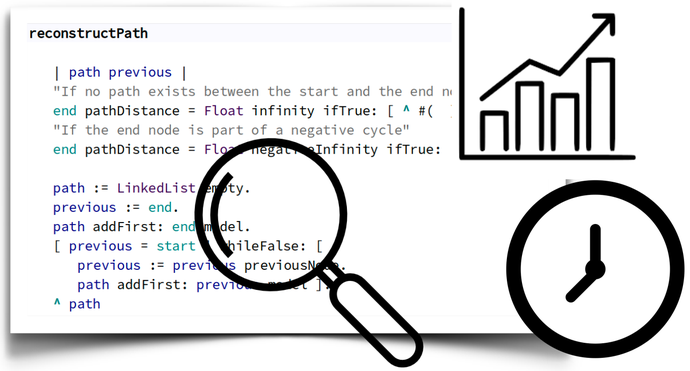
Object lifetime profiler

Useful for

Memory optimizations

- Pretenuring
- GC tuning

Profling Object Lifetimes with Finalization



Object lifetime profiler

It can be implemented using

Finalization mechanisms

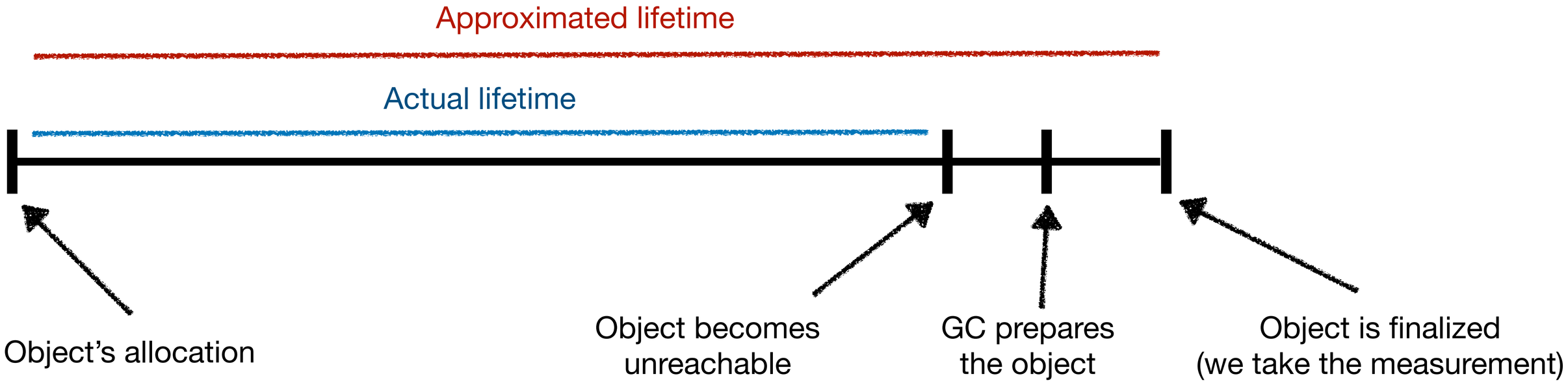
Finalization-Based Lifetime Profiling

$$lifetime = t_{collection} - t_{allocation}$$

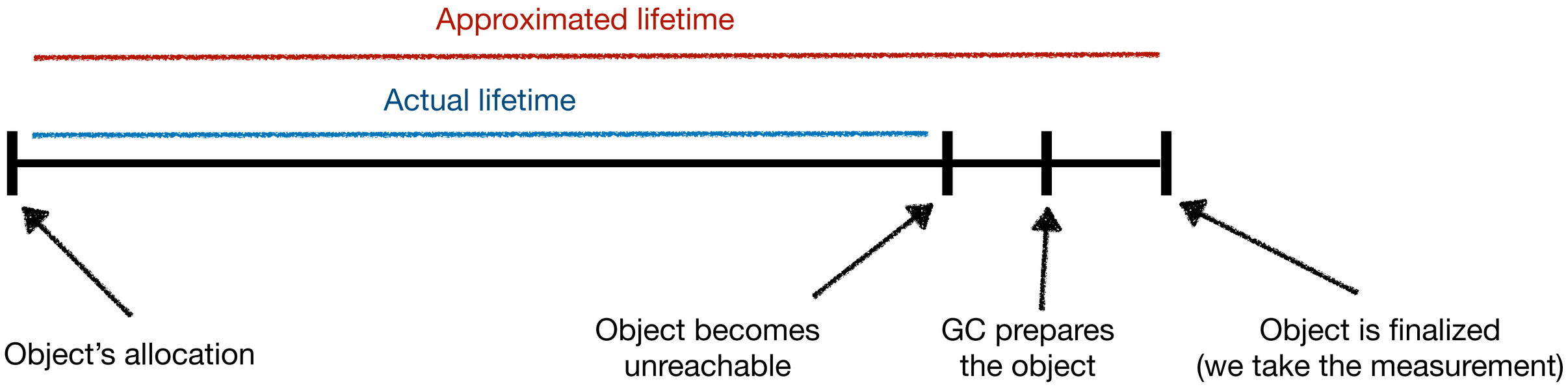
Object Finalization

Code
Instrumentation

Delayed Lifetime Observation

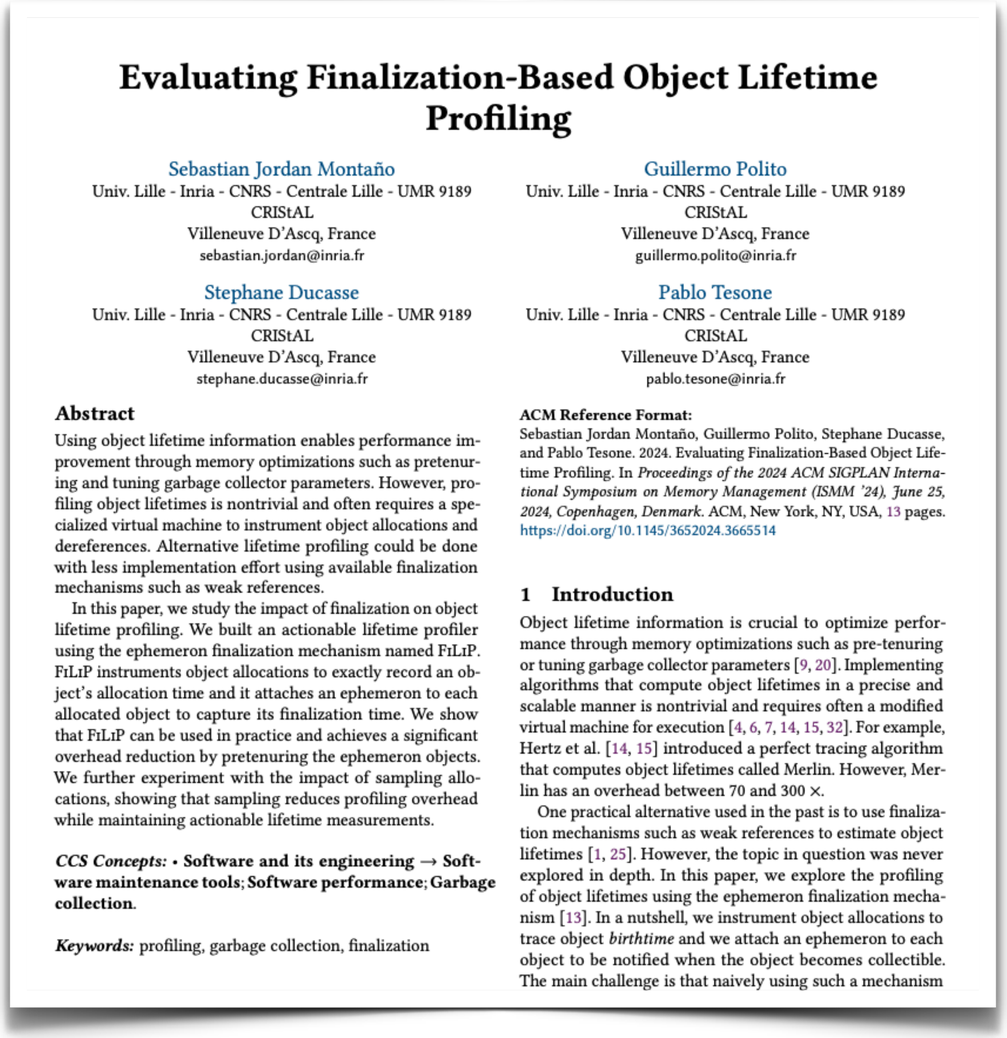


Delayed Lifetime Observation



Lifetime **observation** time is **delayed**

More in the paper!



Evaluating Finalization-Based Object Lifetime Profiling

Sebastian Jordan Montaño
Univ. Lille - Inria - CNRS - Centrale Lille - UMR 9189
CRIStAL
Villeneuve D'Ascq, France
sebastian.jordan@inria.fr

Guillermo Polito
Univ. Lille - Inria - CNRS - Centrale Lille - UMR 9189
CRIStAL
Villeneuve D'Ascq, France
guillermo.polito@inria.fr

Stephane Ducasse
Univ. Lille - Inria - CNRS - Centrale Lille - UMR 9189
CRIStAL
Villeneuve D'Ascq, France
stephane.ducasse@inria.fr

Pablo Tesone
Univ. Lille - Inria - CNRS - Centrale Lille - UMR 9189
CRIStAL
Villeneuve D'Ascq, France
pablo.tesone@inria.fr

Abstract

Using object lifetime information enables performance improvement through memory optimizations such as pretenuring and tuning garbage collector parameters. However, profiling object lifetimes is nontrivial and often requires a specialized virtual machine to instrument object allocations and dereferences. Alternative lifetime profiling could be done with less implementation effort using available finalization mechanisms such as weak references.

In this paper, we study the impact of finalization on object lifetime profiling. We built an actionable lifetime profiler using the ephemeron finalization mechanism named FiLiP. FiLiP instruments object allocations to exactly record an object's allocation time and it attaches an ephemeron to each allocated object to capture its finalization time. We show that FiLiP can be used in practice and achieves a significant overhead reduction by pretenuring the ephemeron objects. We further experiment with the impact of sampling allocations, showing that sampling reduces profiling overhead while maintaining actionable lifetime measurements.

CCS Concepts: • **Software and its engineering** → **Software maintenance tools**; **Software performance**; **Garbage collection**.

Keywords: profiling, garbage collection, finalization

ACM Reference Format:
Sebastian Jordan Montaño, Guillermo Polito, Stephane Ducasse, and Pablo Tesone. 2024. Evaluating Finalization-Based Object Lifetime Profiling. In *Proceedings of the 2024 ACM SIGPLAN International Symposium on Memory Management (ISMM '24), June 25, 2024, Copenhagen, Denmark.* ACM, New York, NY, USA, 13 pages.
https://doi.org/10.1145/3652024.3665514

1 Introduction

Object lifetime information is crucial to optimize performance through memory optimizations such as pre-tenuring or tuning garbage collector parameters [9, 20]. Implementing algorithms that compute object lifetimes in a precise and scalable manner is nontrivial and requires often a modified virtual machine for execution [4, 6, 7, 14, 15, 32]. For example, Hertz et al. [14, 15] introduced a perfect tracing algorithm that computes object lifetimes called Merlin. However, Merlin has an overhead between 70 and 300 ×.

One practical alternative used in the past is to use finalization mechanisms such as weak references to estimate object lifetimes [1, 25]. However, the topic in question was never explored in depth. In this paper, we explore the profiling of object lifetimes using the ephemeron finalization mechanism [13]. In a nutshell, we instrument object allocations to trace object *birthtime* and we attach an ephemeron to each object to be notified when the object becomes collectible. The main challenge is that naively using such a mechanism

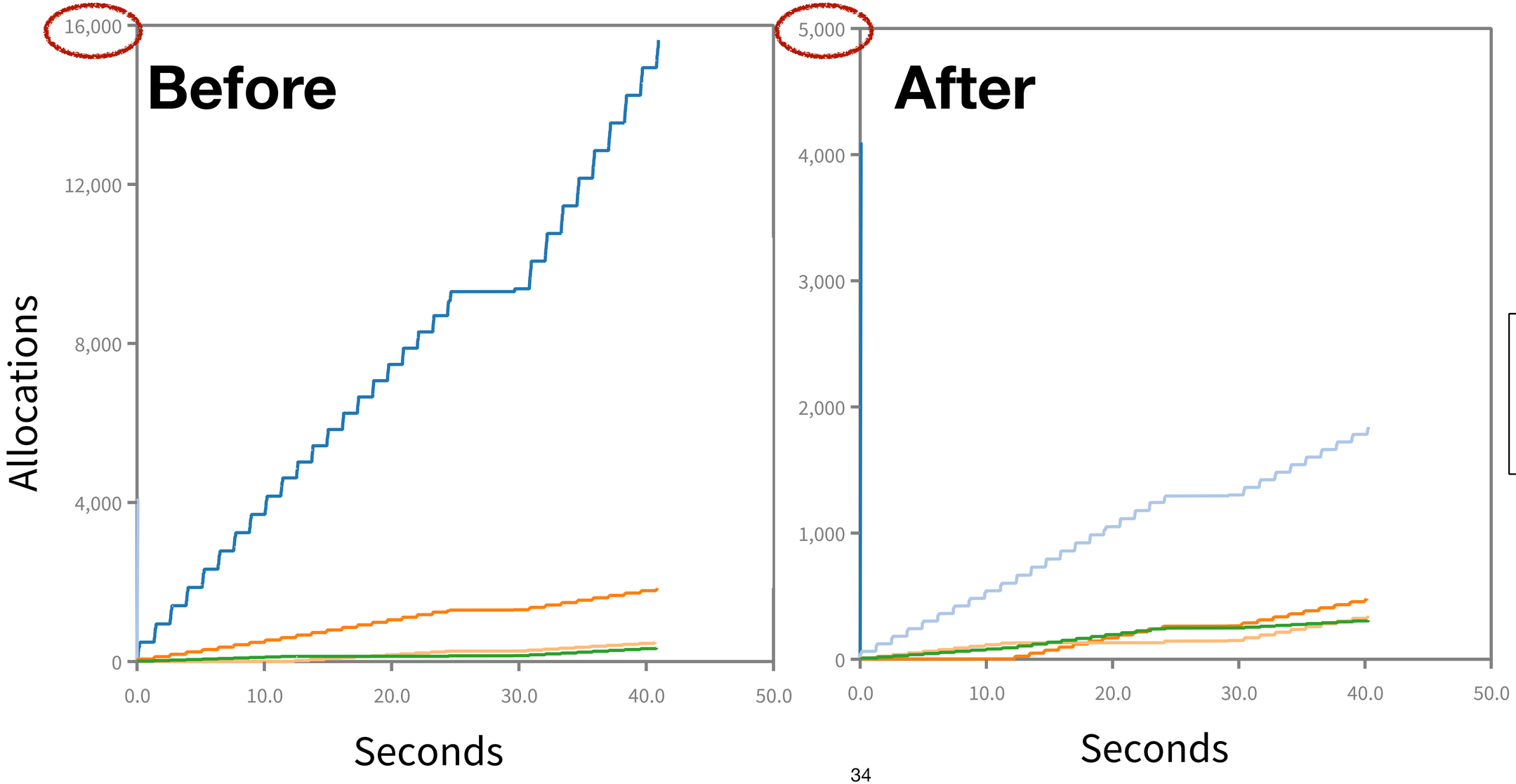
- Evaluation of the estimation of object lifetimes by using finalisation mechanisms

It works

- Yeah

Some numbers

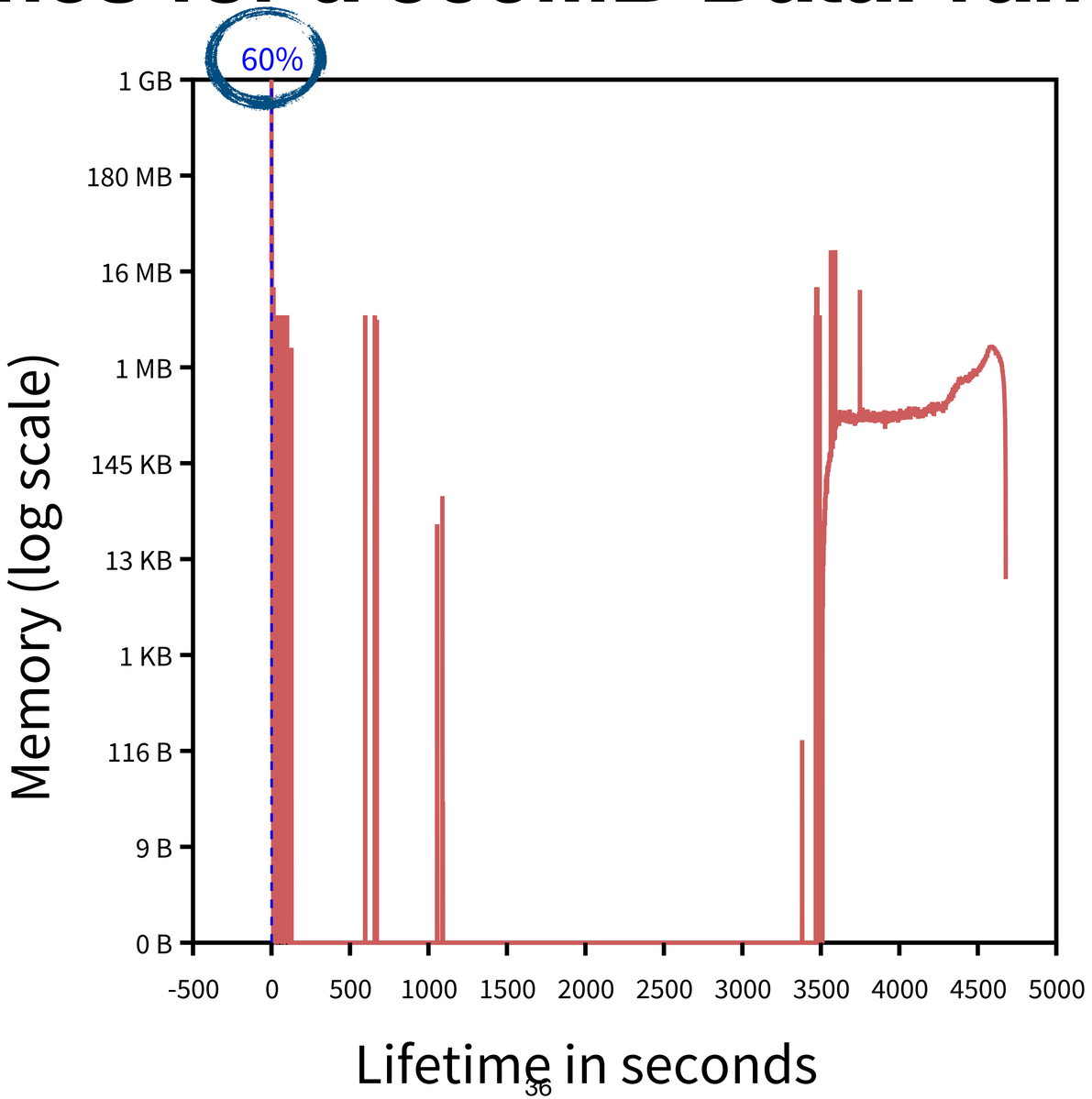
Morphic after the fix



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)



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×

Memory Profiling

- **Pharo** is cool
- MethodProxies is a **safe message-passing** control library that is cool
- **Finalization** profilers can be precise and have low overhead, which is cool
- **Memory profiling** is cool

/jordanmontt/illimani-memory-profiler

Sebastian JORDAN MONTAÑO

jordanmontt.fr