Evaluating Finalization-Based Object Lifetime Profiling

Sebastian JORDAN MONTAÑO, Guillermo POLITO, Stéphane DUCASSE, Pablo TESONE

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

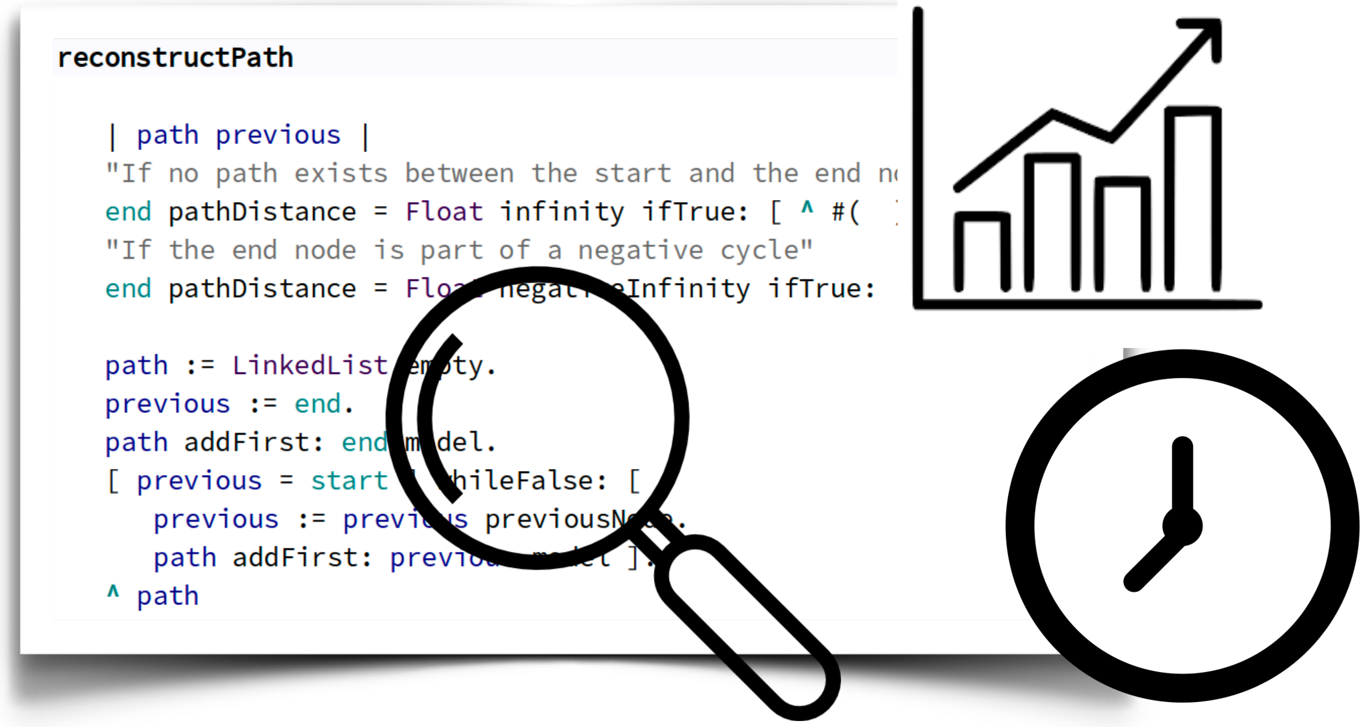
sebastian.jordan@inria.fr

International Symposium on Memory Management (ISMM) 2024

June 2024

Context and description

Advantages of Object Lifetime Profiling



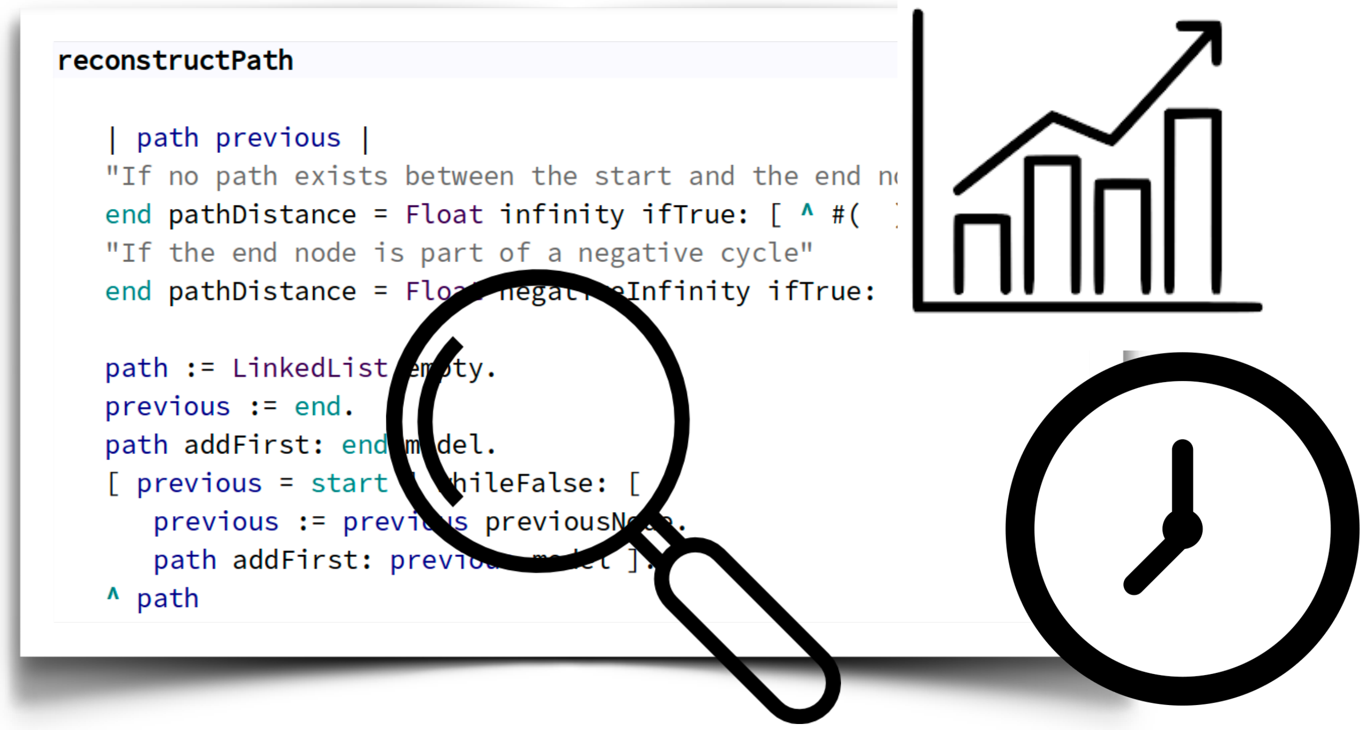
Object lifetime profiler

Useful for

Memory optimizations

- Pretenuring
- GC tuning

Profling Object Lifetimes with Finalization



Object lifetime profiler

It can be implemented using

Finalization mechanisms

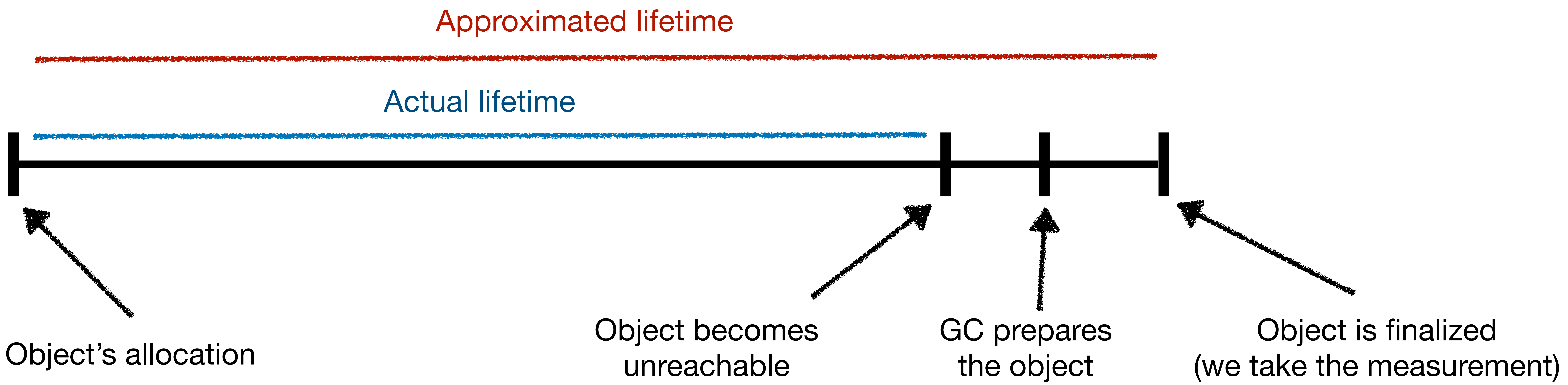
Finalization-Based Lifetime Profiling

$$\texttt{lifetime} = \texttt{t}_{\texttt{collection}} - \texttt{t}_{\texttt{allocation}}$$

Object Finalization

Code Instrumentation

Delayed Lifetime Observation

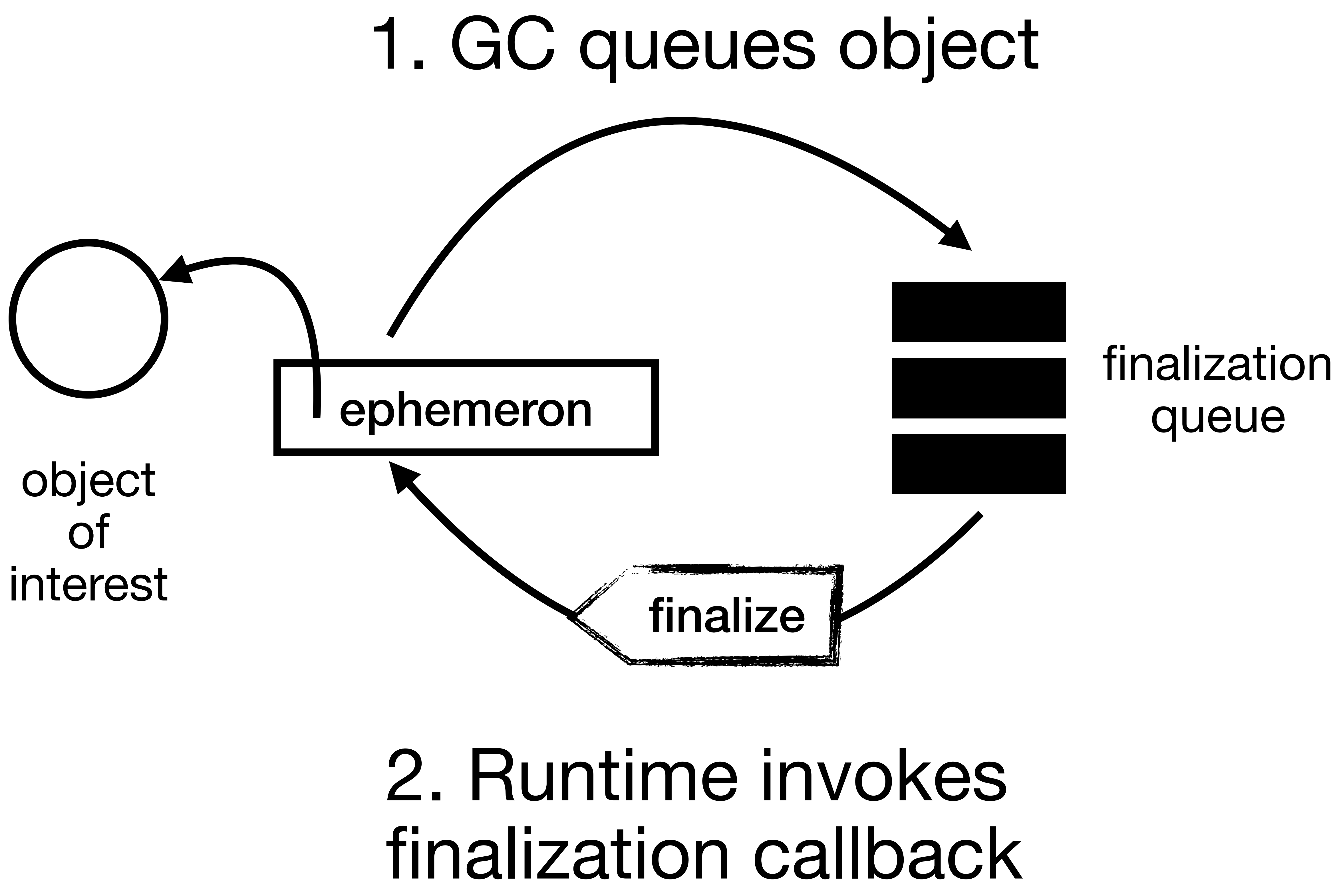


Lifetime **observation** time is **delayed**

Ephemeron Finalization and Contamination

- Finalization notifies object collection
- Ephemerons track objects of interest
- GC queues ephemeron when object is candidate for collection
- The execution engine dequeues and invokes finalization callback

Ephemeron contaminate the heap, adding extra GC pressure



Ephemerons: A New Finalization Mechanism
Hayes. OOPSLA'97

Evaluating Finalization-Based Lifetime Profiling

- **Goal**: *understanding the impact* of
    - ephemeron contamination
    - delayed observation times

FiLiP: Finalization Lifetime Profiler

We built a profiler for experimentation: **FiLiP**

- Tracks allocation time by **instrumenting allocations**
- Tracks collection time with ***ephemeron finalization***
  - *optionally pre-tenured for experimentation*
- Built-in **sampling support**: track a *configurable* subset of all allocations

Methodology and Results

Research Questions

- RQ1: Are lifetime profilers actionable?
- **RQ2: How does sampling impact the computed object lifetimes?**
- **RQ3: Does sampling reduce the execution time overhead?**
- RQ4: Does sampling reduce the memory overhead?

Benchmark selection

- **DataFrame**: Load a synthetic dataset of 230 MB .
- **HoneyGinger**: Run a smoothed-particle hydrodynamics simulator.
- **ReMobidyc**: Run a multi-agent simulator of wolves chasing and eating goats.
- **Bloc**: Render moving figures that simulate the flocking behavior of birds.
- **Moose**: Load a software database of 13521 classes and 48087 methods into the Moose meta-model.

General methodology

- We profiled each benchmark using **4 different sampling rates**: 0.1%, 1%, 50%, and 100%
- Presented measured **lifetimes are relative** to total benchmark execution time.
    - To account for profiling overhead
- Execution time overhead ***with and without ephemeron pre-tenuring***

RQ2: Precision Methodology

- We compare lifetimes across all sampling rates to assess variation.
- In-depth analysis of the DataFrame benchmark.

RQ2: Precision

Sampling and lifetimes

	100% Sampling	50% Sampling	1% Sampling	0.1% Sampling	**Avg±stdev**
DataFrame					
Overall average lifetimes	15.24%	14.68%	14.09%	14.21%	14.55%±0.45
HoneyGinger					
Overall average lifetimes	0.09%	0.08%	0.05%	0.06%	0.07%±0.014
Re:Mobidyc					
Overall average lifetimes	0.5%	0.36%	0.46%	0.18%	0.37%±0.12
Bloc					
Overall average lifetimes	6.27%	6.27%	6.51%	6.63%	6.42%±0.15
Moose					
Overall average lifetimes	13.19%	13.33%	13.67%	12.82%	13.25%±0.3

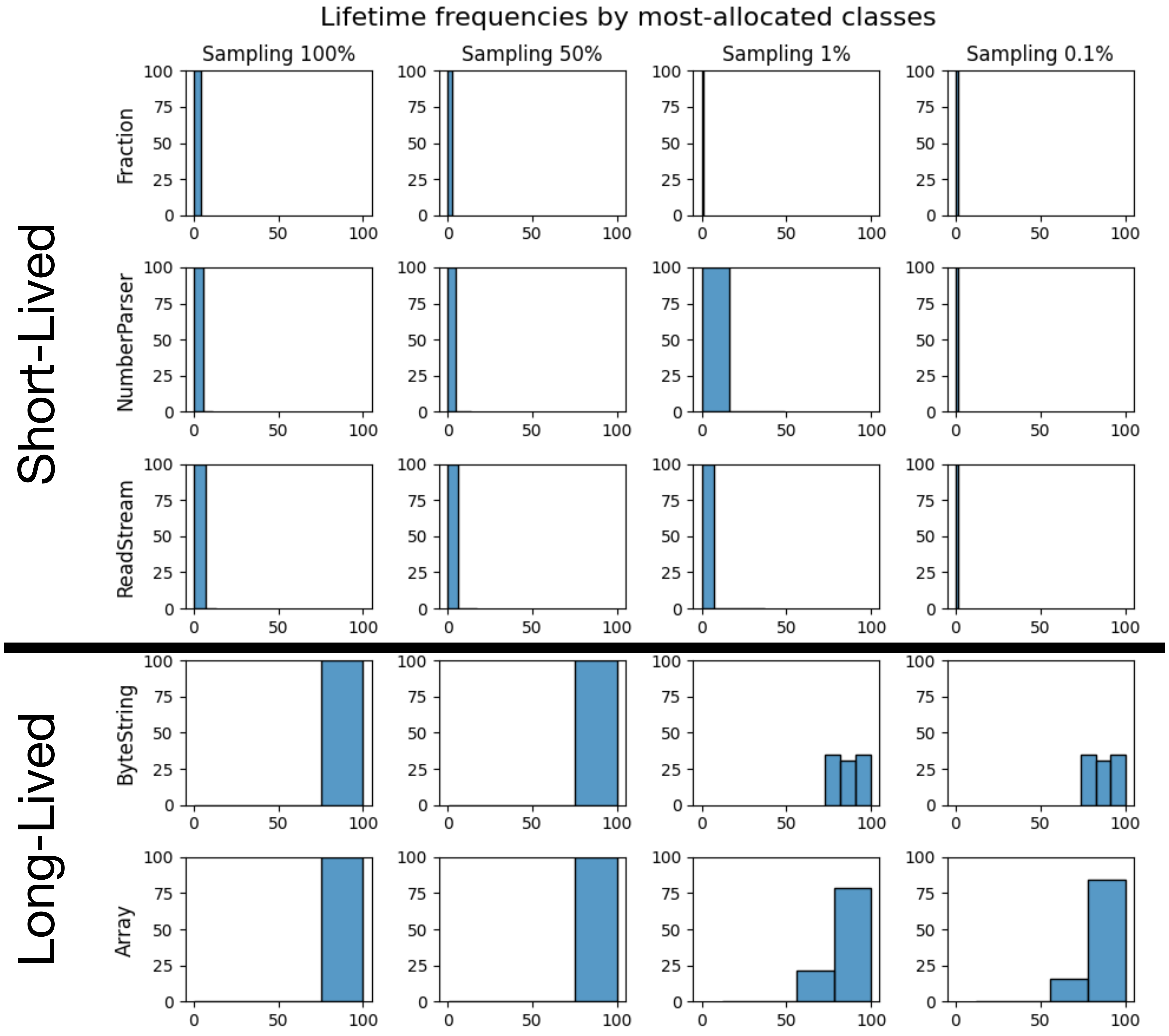
RQ2: Precision

Sampling and lifetimes

	100% Sampling	50% Sampling	1% Sampling	0.1% Sampling	**Avg±stdev**
DataFrame					
Overall average lifetimes	15.24%	14.68%	14.09%	14.21%	14.55%±0.45
HoneyGinger					
Overall average lifetimes	0.09%	0.08%	0.05%	0.06%	0.07%±0.014
Re:Mobidyc					
Overall average lifetimes	0.5%	0.36%	0.46%	0.18%	0.37%±0.12
Bloc					
Overall average lifetimes	6.27%	6.27%	6.51%	6.63%	6.42%±0.15
Moose					
Overall average lifetimes	13.19%	13.33%	13.67%	12.82%	13.25%±0.3

RQ2: Precision

- DataFrame case
- Segregating short and long lived classes



RQ2: Conclusion

- Consistent lifetime frequencies **suggest precise measurements.**

- **Recommendation for developers:**
  - Start with a small sampling rate
  - Gradually incrementing it if needed
  - We obtained **precise results using a 0.1% sampling rate**

RQ3: Execution Time Overhead Methodology

- Comparison across **5 configurations**:
  - Baseline: without the profiler
  - Profiler active with sampling rates of 100%, 50%, 1% and 0.1%.

- Effect of **pretenuring the ephemerons**

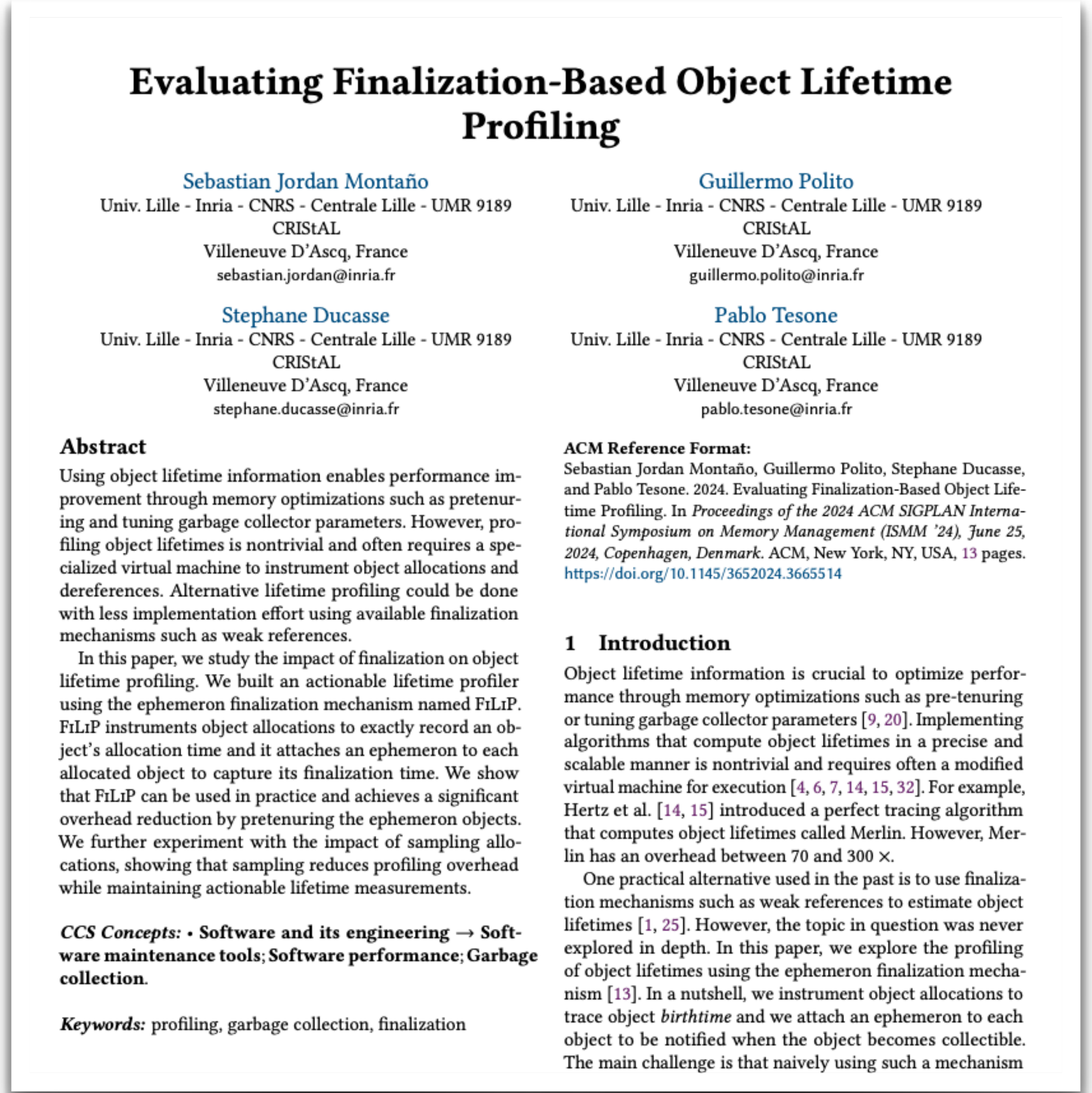
RQ3: Execution time overhead

Sampling rate	100%	50%	1%	0.1%
DataFrame - pretenuring	5.8 × ±0.017	3.8 × ±0.019	2.25 × ±0.019	2.27 × ±0.012
DataFrame + pretenuring	2.29 × ±0.03	1.64 × ±0.01	1.08 × ±0.001	1.12 × ±0.001
ΔPretenuring	2.53×	2.32×	2.08×	2.03×
HoneyGinger - pretenuring	9.2 × ±0.01	8.91 × ±0.03	8.4 × ±0.6	7.8 × ±0.04
HoneyGinger + pretenuring	3.2 × ±0.10	2.5 × ±0.10	2.2 × ±0.06	1.8 × ±0.05
ΔPretenuring	2.88×	3.56×	3.82×	4.33×
Bloc - pretenuring	1.04 × ±0.002	1.04 × ±0.001	1.04 × ±0.0008	1.04 × ±0.0006
Bloc + pretenuring	1.03 × ±0.001	1.03 × ±0.001	1.03 × ±0.0007	1.03 × ±0.0007
ΔPretenuring	1.01×	1.01×	1.01×	1.01×
Moose - pretenuring	3.2 × ±0.02	2.6 × ±0.008	2.02 × ±0.007	2.00 × ±0.005
Moose + pretenuring	2.02 × ±0.03	1.6 × ±0.007	1.1 × ±0.003	1.09 × ±0.002
ΔPretenuring	1.58×	1.63×	1.84×	1.84×
Re:Mobidyc - pretenuring	32.3 × ±2.2	13.8 × ±0.11	9.14 × ±0.13	9.01 × ±0.08
Re:Mobidyc + pretenuring	2.06 × ±0.005	1.5 × ±0.004	1.3 × ±0.01	1.25 × ±0.05
ΔPretenuring	15.68×	9.20×	7.03×	7.21×
Avg. + pretenuring	2.12×	1.65×	1.34×	1.26×

RQ3: Conclusion

- Pretenuring ephemerons
    - **reduces the overhead by 3.68x** on average
    - **average overhead of 1.59x** across all sampling rates
- Finalization profiling is a **practical alternative to estimating object lifetimes**

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

- All the research questions
- More detailed evaluation and methodology
- Detailed results
- Implementation details

Evaluating Finalization-Based Object Lifetime Profiling

- Finalization profilers can be **weakly actionable** and have **low overhead**

- **Pre-tenuring** ephemerons and **sampling** significantly reduces overhead

- **Sampling** provides **relevant object lifetime information**

Sebastian JORDAN MONTAÑO

sebastian.jordan@inria.fr

github.com/jordanmontt/illimani-memory-profiler

Inria
cnrs
Université de Lille
centralelille ÉCOLE CENTRALE DE LILLE
CRIStAL Centre de Recherche en Informatique, Signal et Automatique de Lille
Région Hauts-de-France
Evref