Improving Performance Through Object Lifetime Profiling: the DataFrame Case

Sebastian JORDAN MONTAÑO, Nahuel PALUMBO, Guillermo POLITO, Stéphane DUCASSE and Pablo TESONE

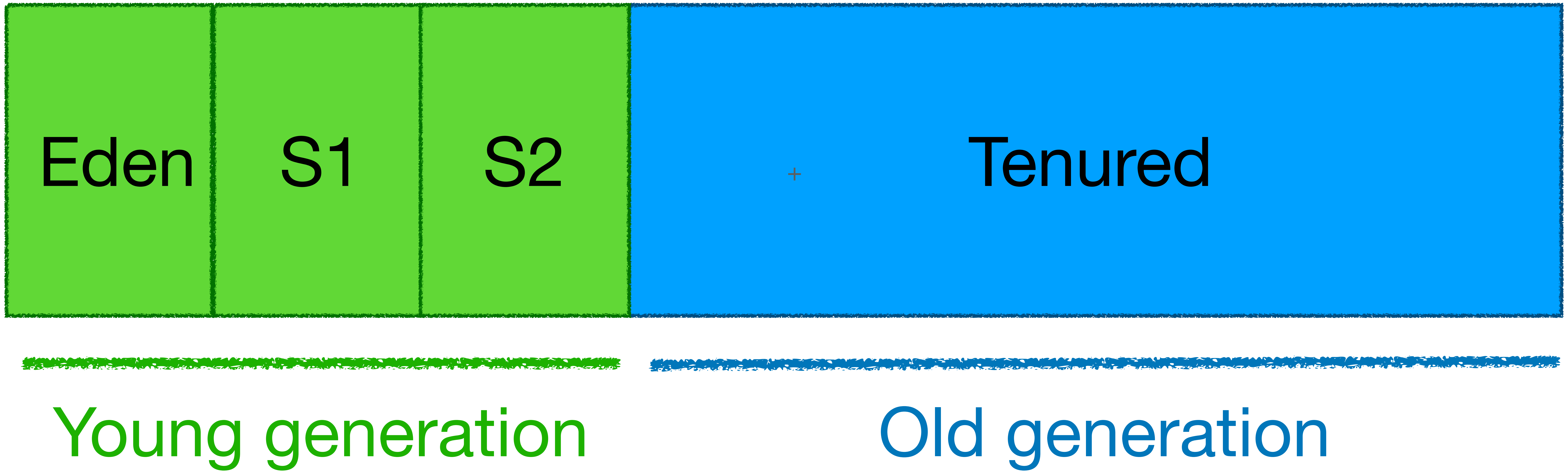
Inria, Univ. Lille, CNRS, Centrale Lille, UMR 9189 - CRIStAL

August 2023

Memory management in software

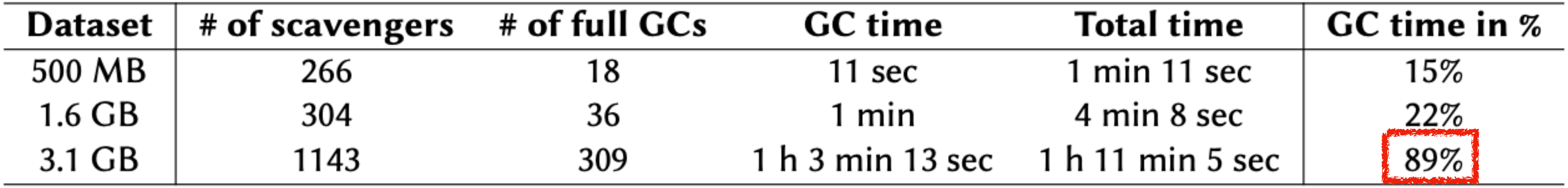
```
int* ptr = (int*) malloc(sizeof(int));
free(ptr);
```

Pharo’s garbage collector



GC parameters

Time spent on garbage collecting

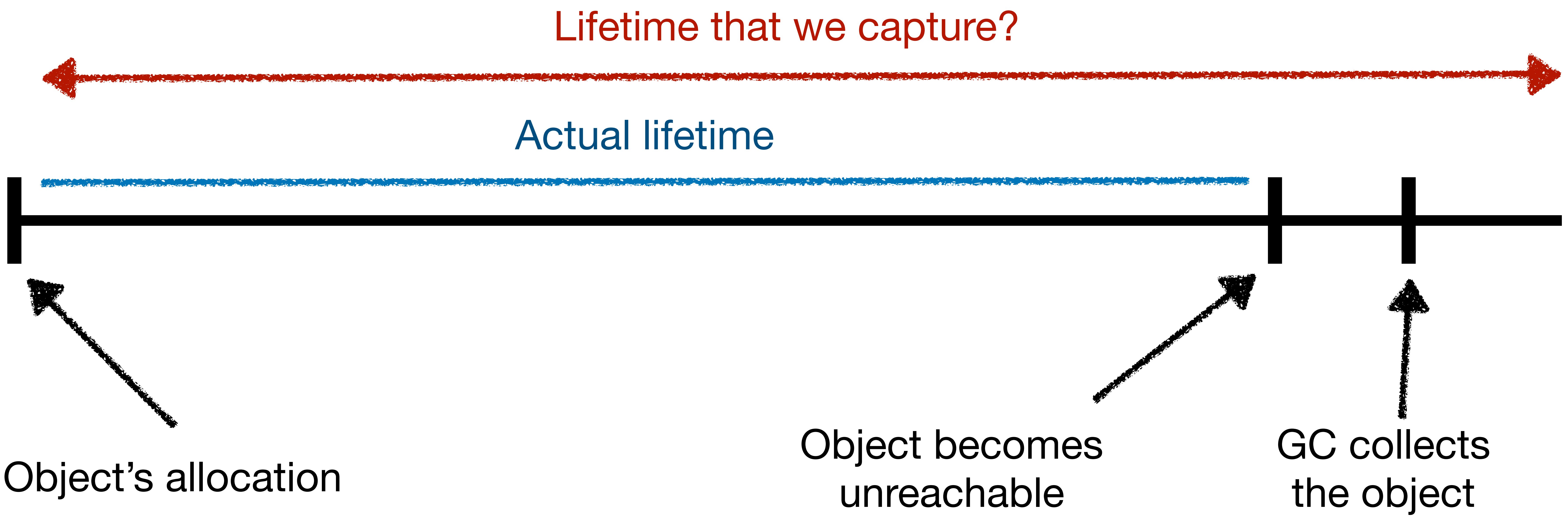


Dataset	# of scavengers	# of full GCs	GC time	Total time	GC time in %
500 MB	266	18	11 sec	1 min 11 sec	15%
1.6 GB	304	36	1 min	4 min 8 sec	22%
3.1 GB	1143	309	1 h 3 min 13 sec	1 h 11 min 5 sec	89%

Research question

- *How does approximate object lifetimes lead to GC performance improvements?*

An object’s lifetime



Lifetime that we capture?

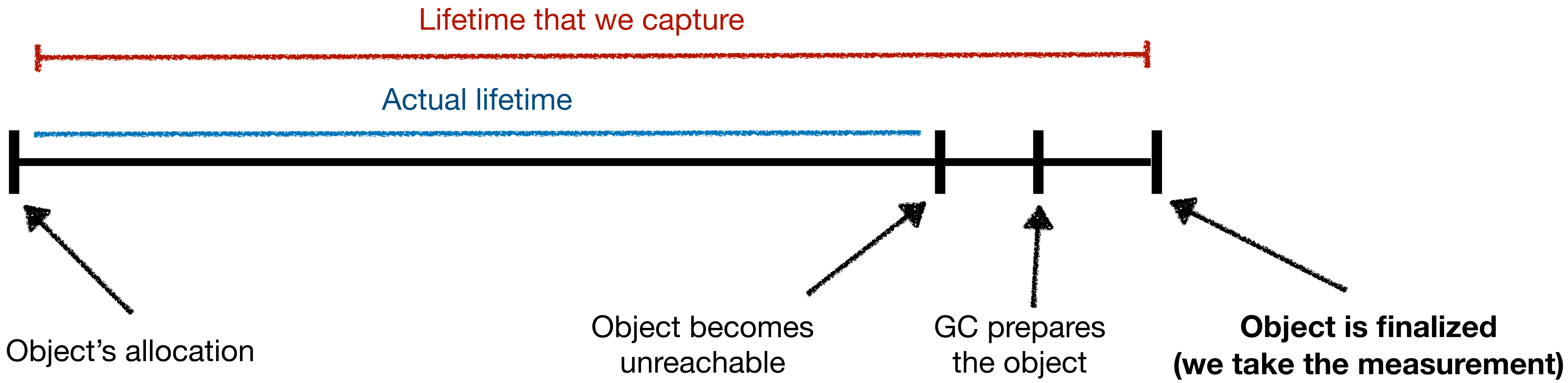
Actual lifetime

Object’s allocation

Object becomes unreachable

GC collects the object

An object’s approximated lifetime



Lifetime that we capture

Actual lifetime

Object’s allocation

Object becomes unreachable

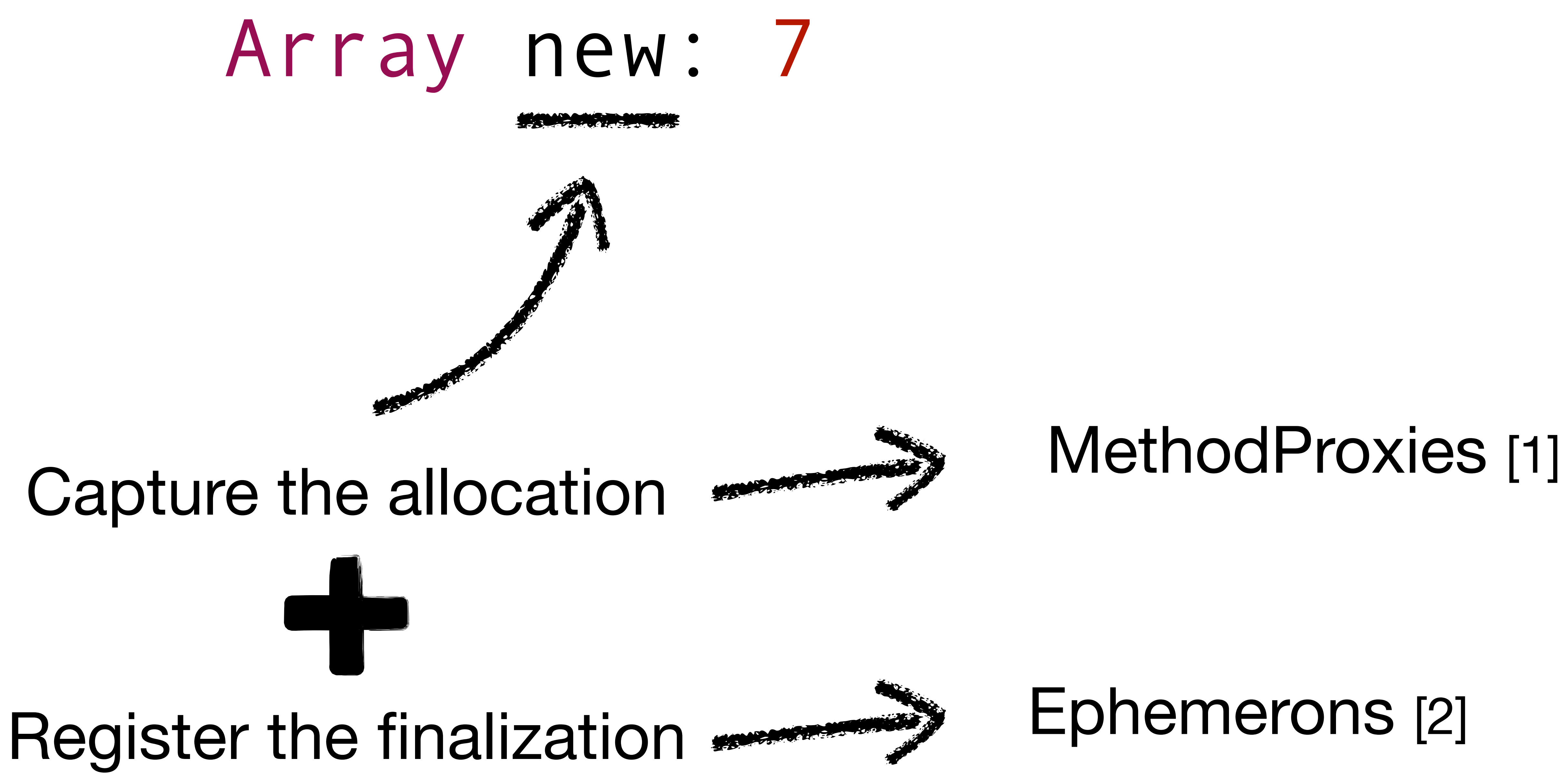
GC prepares the object

Object is finalized (we take the measurement)

Capturing the allocations

```
Array new: 7
```

Capturing the allocations



```
Array new: 7
```

Capture the allocation → MethodProxies [1]

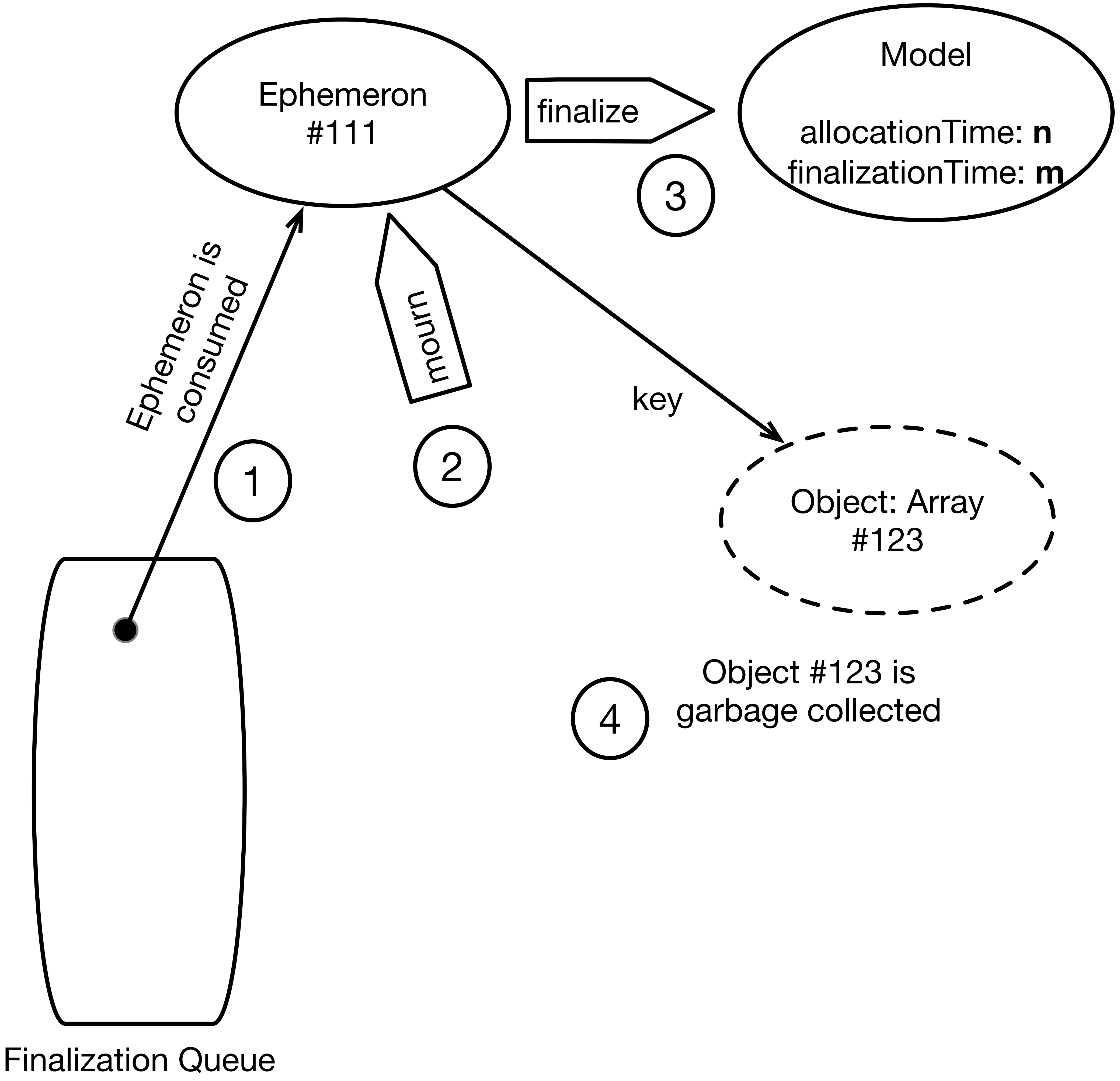
+

Register the finalization → Ephemerons [2]

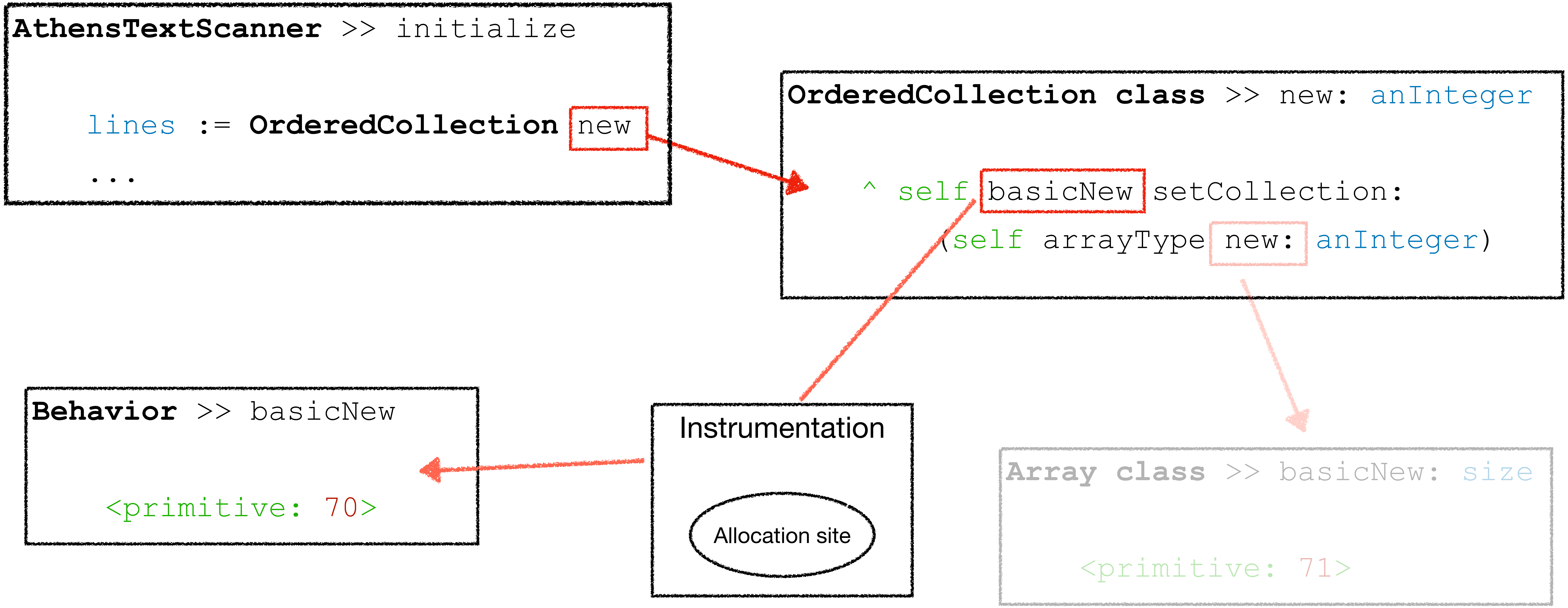
[1] github.com/pharo-contributions/MethodProxies

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

An object’s finalization at a time m



An object’s allocation at a time n



```
AthensTextScanner >> initialize

    lines := OrderedCollection new
    ...
```

```
OrderedCollection class >> new: anInteger

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

```
Behavior >> basicNew

    <primitive: 70>
```

Instrumentation

Allocation site

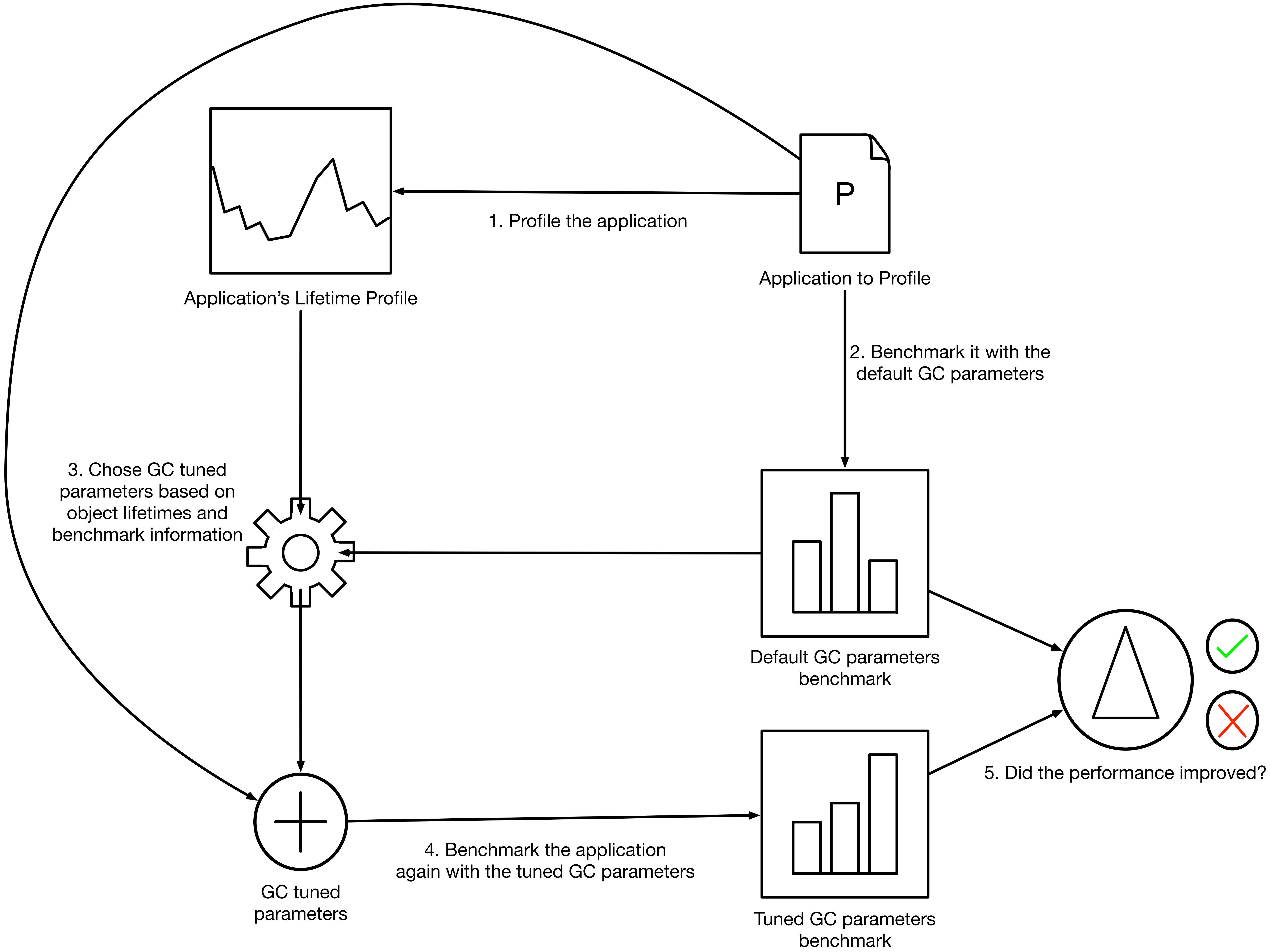
```
Array class >> basicNew: size

    <primitive: 71>
```

Paper’s contributions

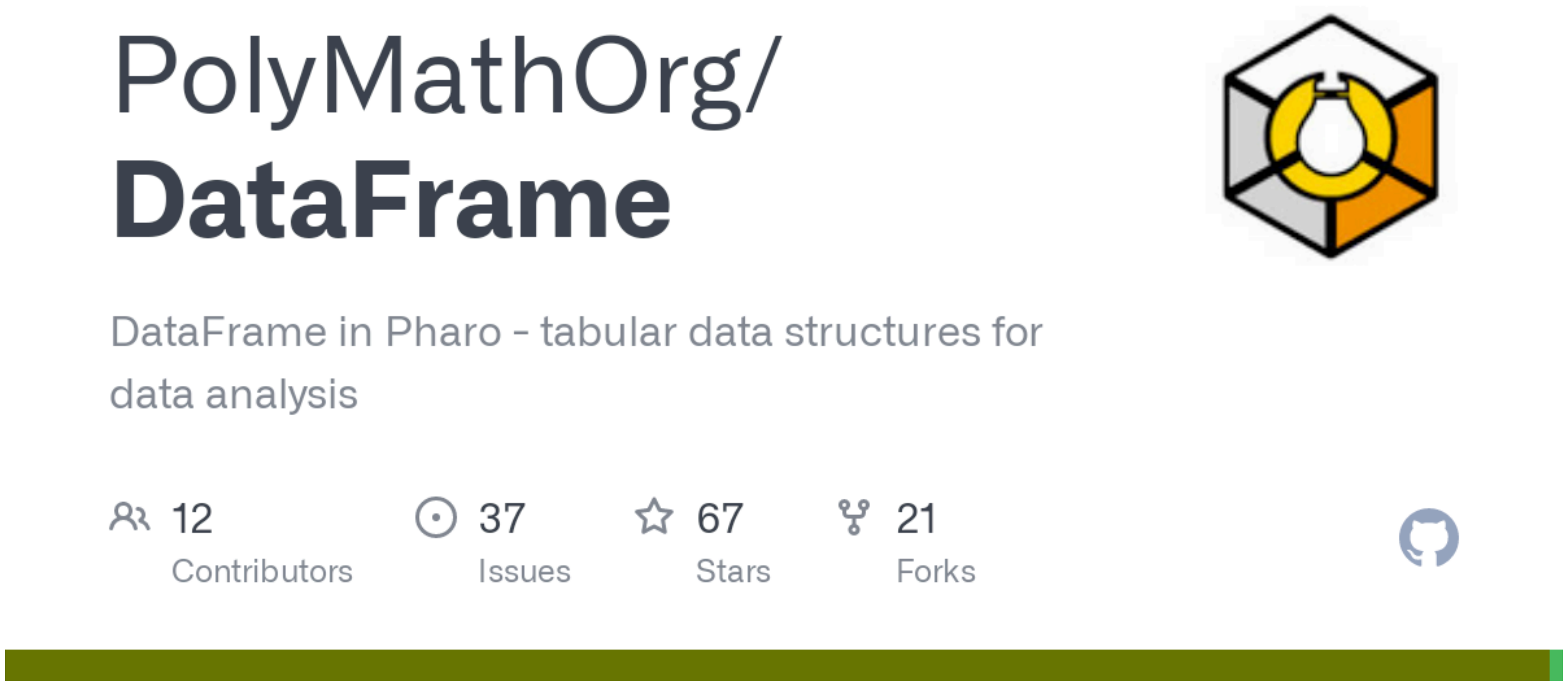
- *Challenges* of lifetime profiling
- **Illimani:** a lifetime profiler on stock Pharo VM

Methodology



P
Application to Profile
1. Profile the application
Application's Lifetime Profile
2. Benchmark it with the default GC parameters
3. Chose GC tuned parameters based on object lifetimes and benchmark information
Default GC parameters benchmark
GC tuned parameters
4. Benchmark the application again with the tuned GC parameters
Tuned GC parameters benchmark
5. Did the performance improved?

The target application



PolyMathOrg/
DataFrame

DataFrame in Pharo - tabular data structures for data analysis

12 Contributors

37 Issues

67 Stars

21 Forks

https://github.com/PolyMathOrg/DataFrame

Benchmark the loading of DataFrame

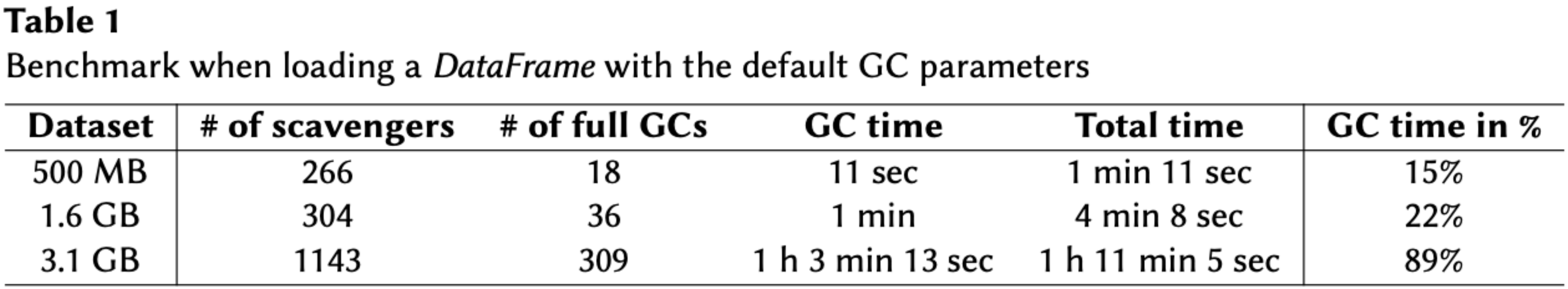


Table 1

Benchmark when loading a *DataFrame* with the default GC parameters

Dataset	# of scavengers	# of full GCs	GC time	Total time	GC time in %
500 MB	266	18	11 sec	1 min 11 sec	15%
1.6 GB	304	36	1 min	4 min 8 sec	22%
3.1 GB	1143	309	1 h 3 min 13 sec	1 h 11 min 5 sec	89%

Benchmark the loading of DataFrame

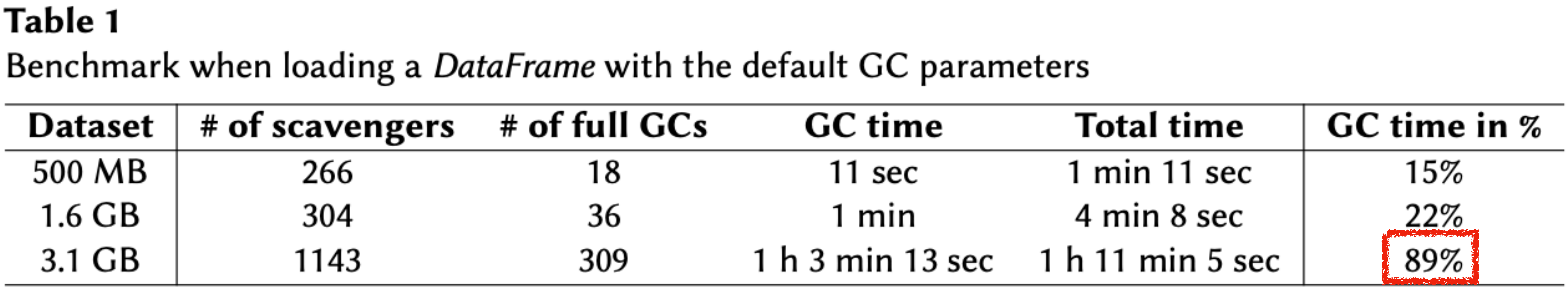
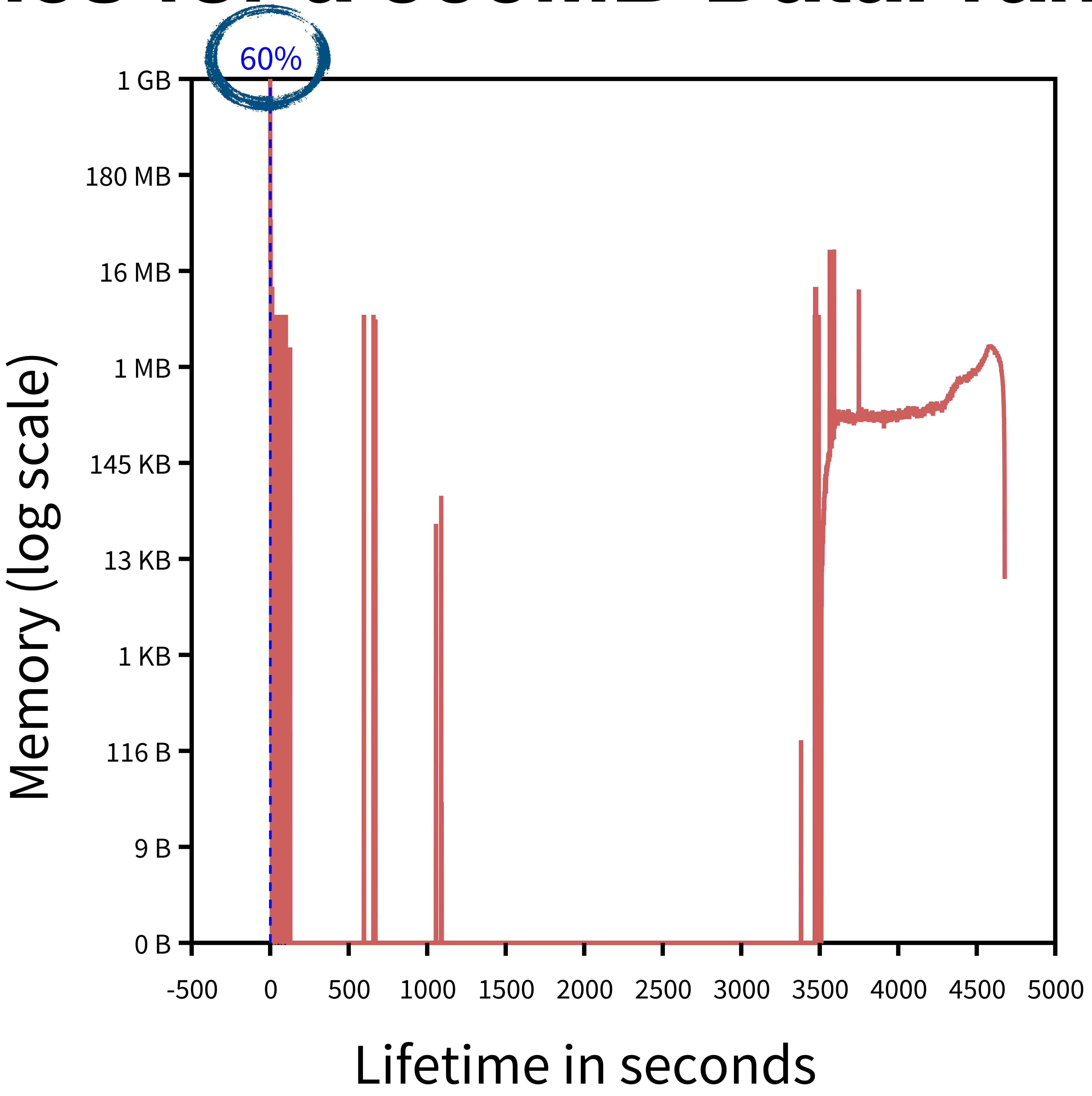


Table 1

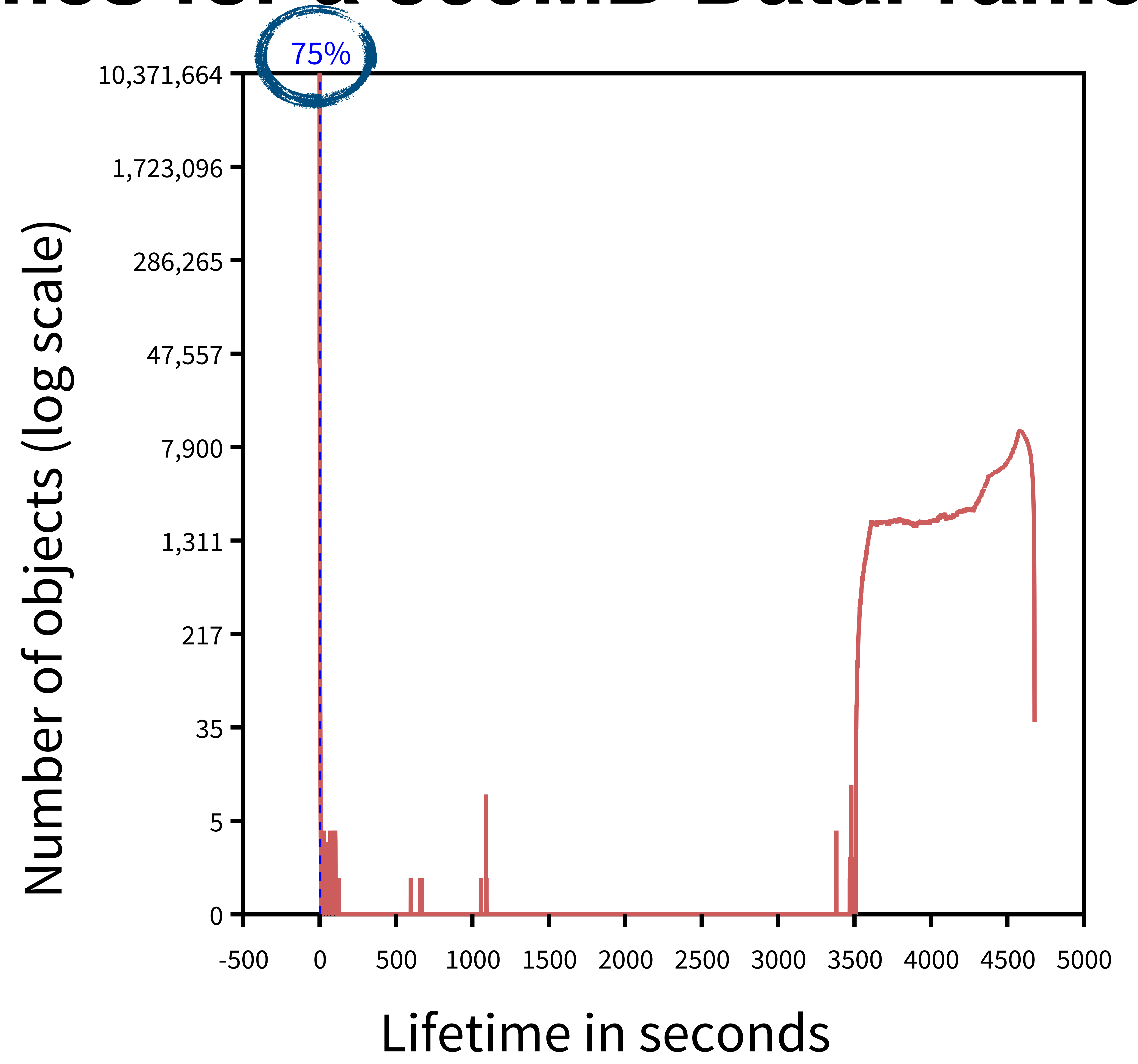
Benchmark when loading a *DataFrame* with the default GC parameters

Dataset	# of scavengers	# of full GCs	GC time	Total time	GC time in %
500 MB	266	18	11 sec	1 min 11 sec	15%
1.6 GB	304	36	1 min	4 min 8 sec	22%
3.1 GB	1143	309	1 h 3 min 13 sec	1 h 11 min 5 sec	89%

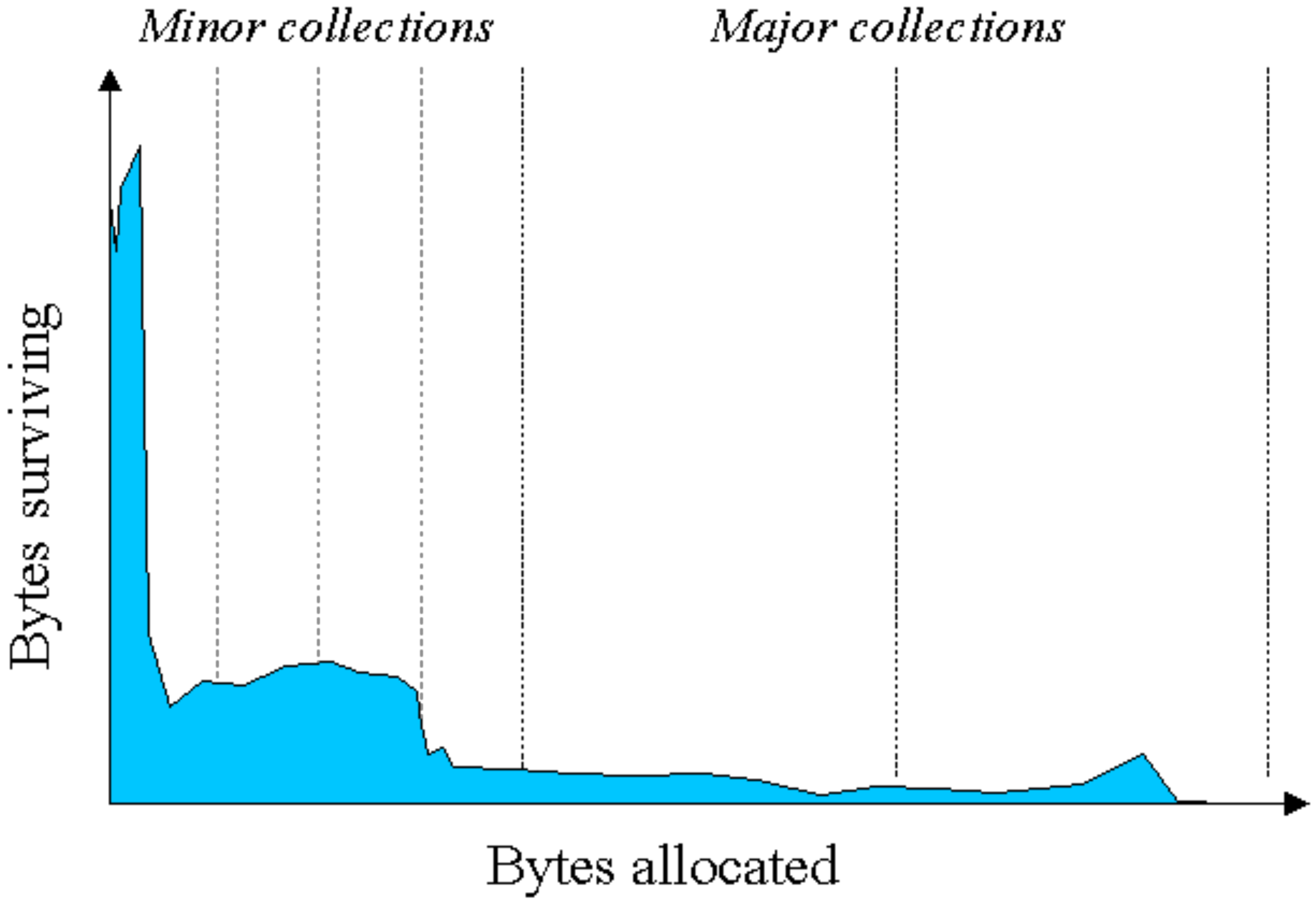
Object lifetimes for a 500MB DataFrame (memory)



Object lifetimes for a 500MB DataFrame (# objects)



Common object lifetime distribution



Source: oracle.com

GC custom parameters

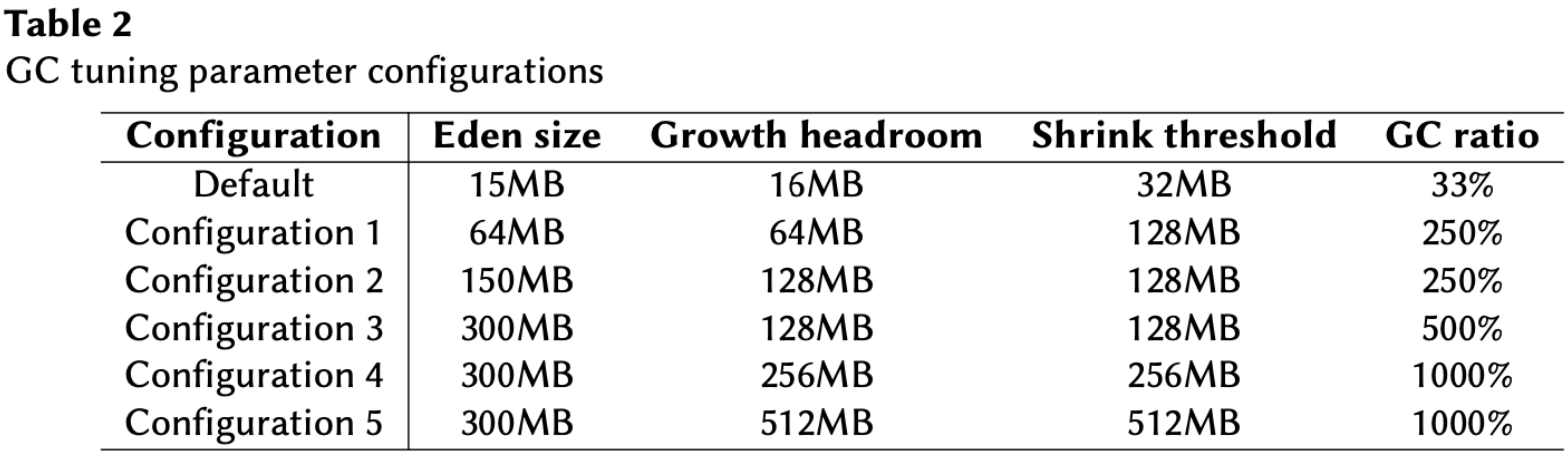
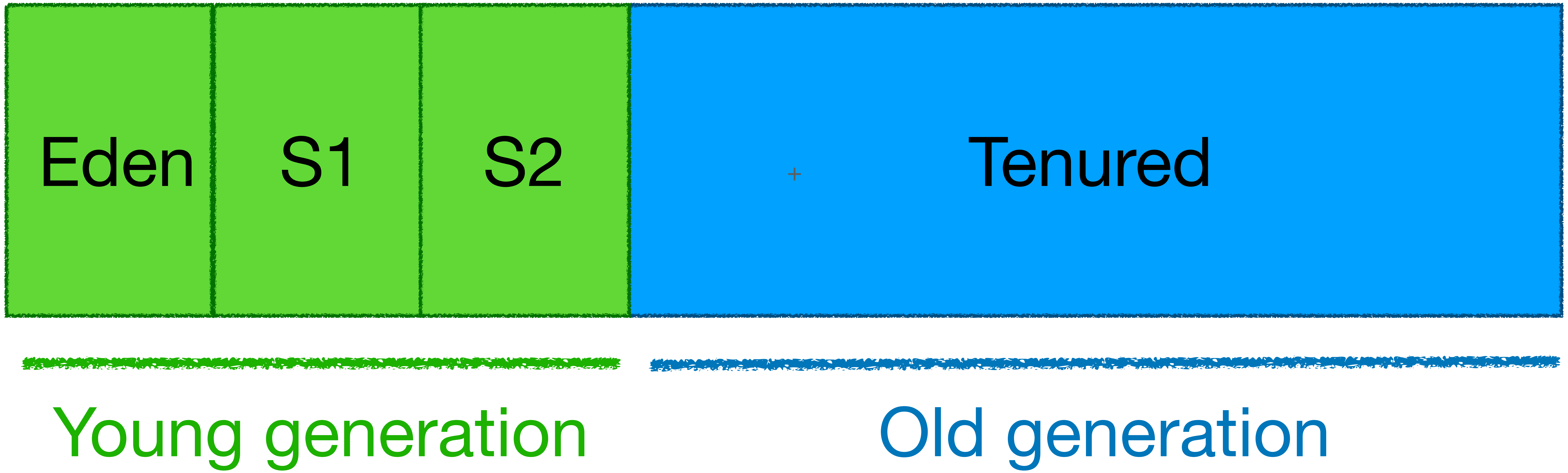


Table 2
GC tuning parameter configurations

Configuration	Eden size	Growth headroom	Shrink threshold	GC ratio
Default	15MB	16MB	32MB	33%
Configuration 1	64MB	64MB	128MB	250%
Configuration 2	150MB	128MB	128MB	250%
Configuration 3	300MB	128MB	128MB	500%
Configuration 4	300MB	256MB	256MB	1000%
Configuration 5	300MB	512MB	512MB	1000%

Pharo’s garbage collector



Benchmarks results

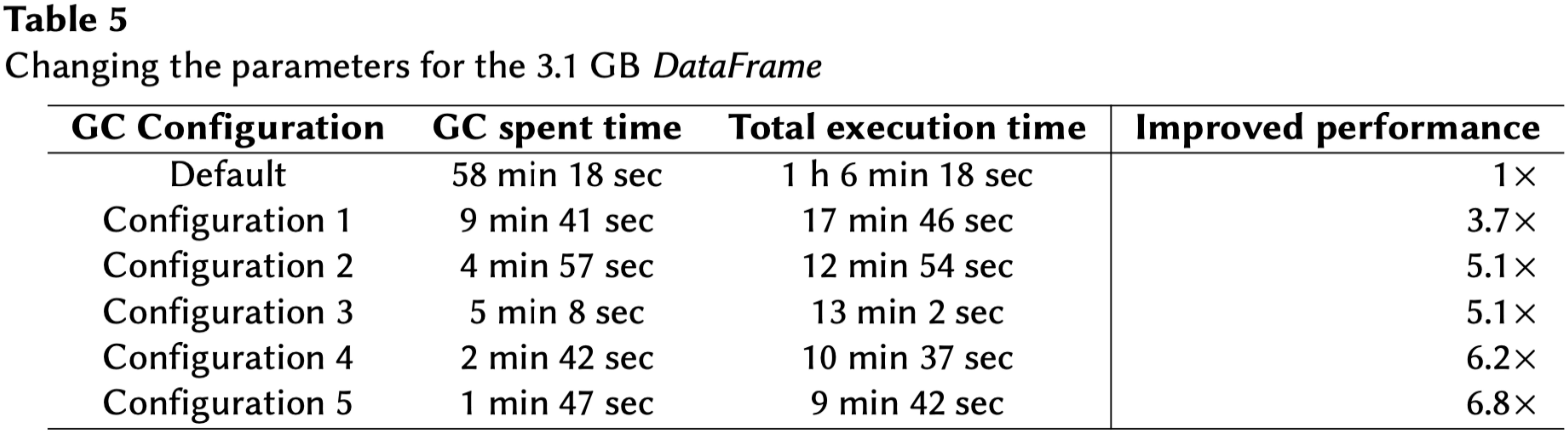


Table 5

Changing the parameters for the 3.1 GB *DataFrame*

GC Configuration	GC spent time	Total execution time	Improved performance
Default	58 min 18 sec	1 h 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×

Future work

- Measure the precision of our approximate object lifetimes
- Profiling at VM level to reduce the overhead
- Pre-tenuring

Summary

- We developed a lifetime profiler
- We profiled the object lifetimes and we validated our solution by observing how lifetimes relate to performance improvements when tuning the GC.

Sebastian JORDAN MONTAÑO

sebastian.jordan@inria.fr

github.com/jordanmontt/illimani-memory-profiler