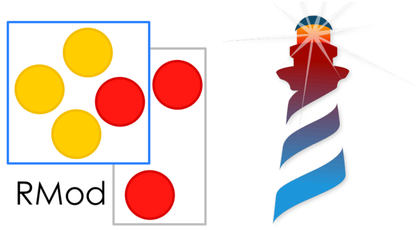


Profiling Code Cache Behaviour via Events

Work In Progress Paper - MPLR 2021

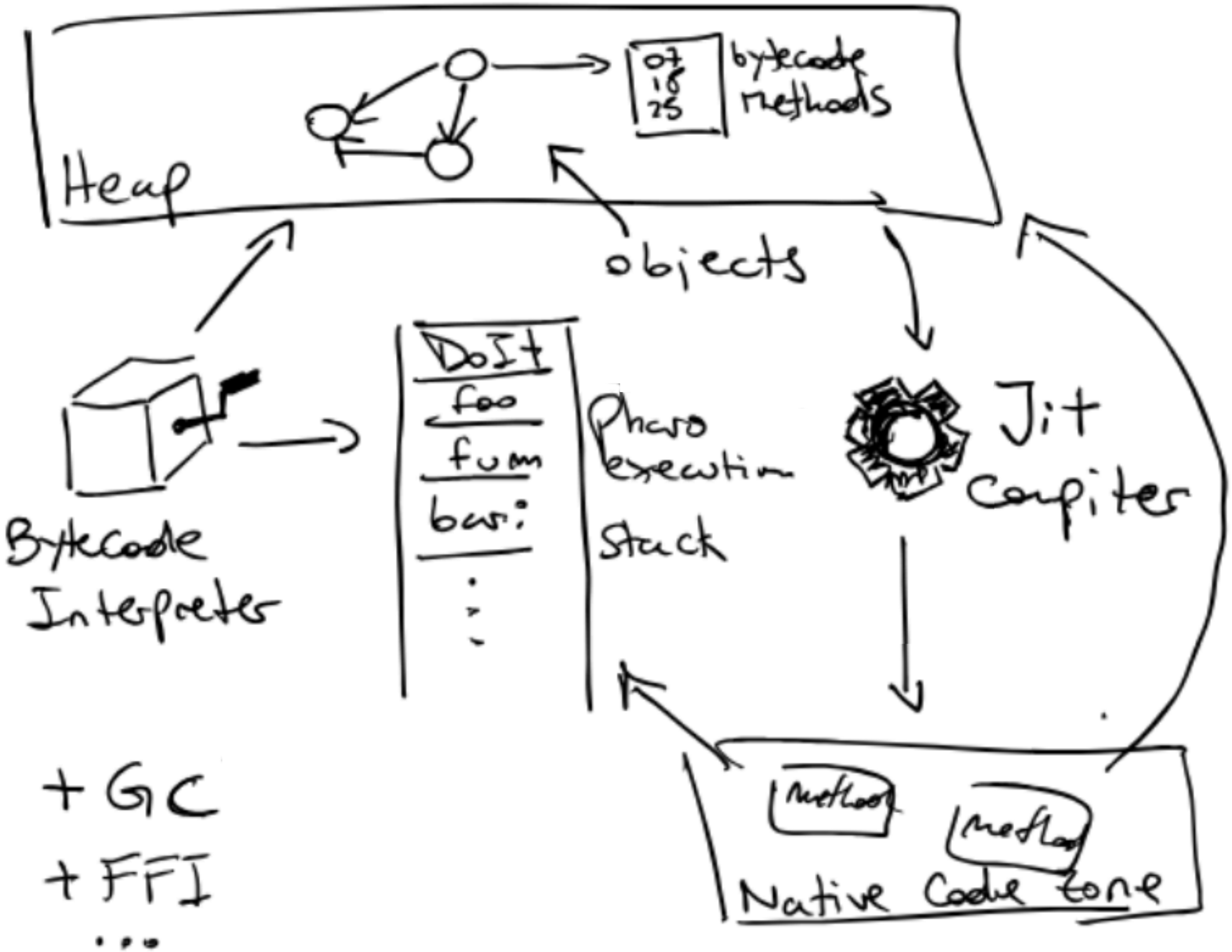
Pablo Tesone, Guillermo Polito, Stéphane Ducasse
Univ. Lille, Inria, CNRS, Centrale Lille, UMR 9189 CRIStAL
Pharo Consortium

What is Pharo

A Programming Language + IDE

- Dynamically-typed: no type annotations, no static type checks
- General purpose!
- Object-oriented + Classes
- Open Source - MIT License
- Used for teaching, research and in the industry

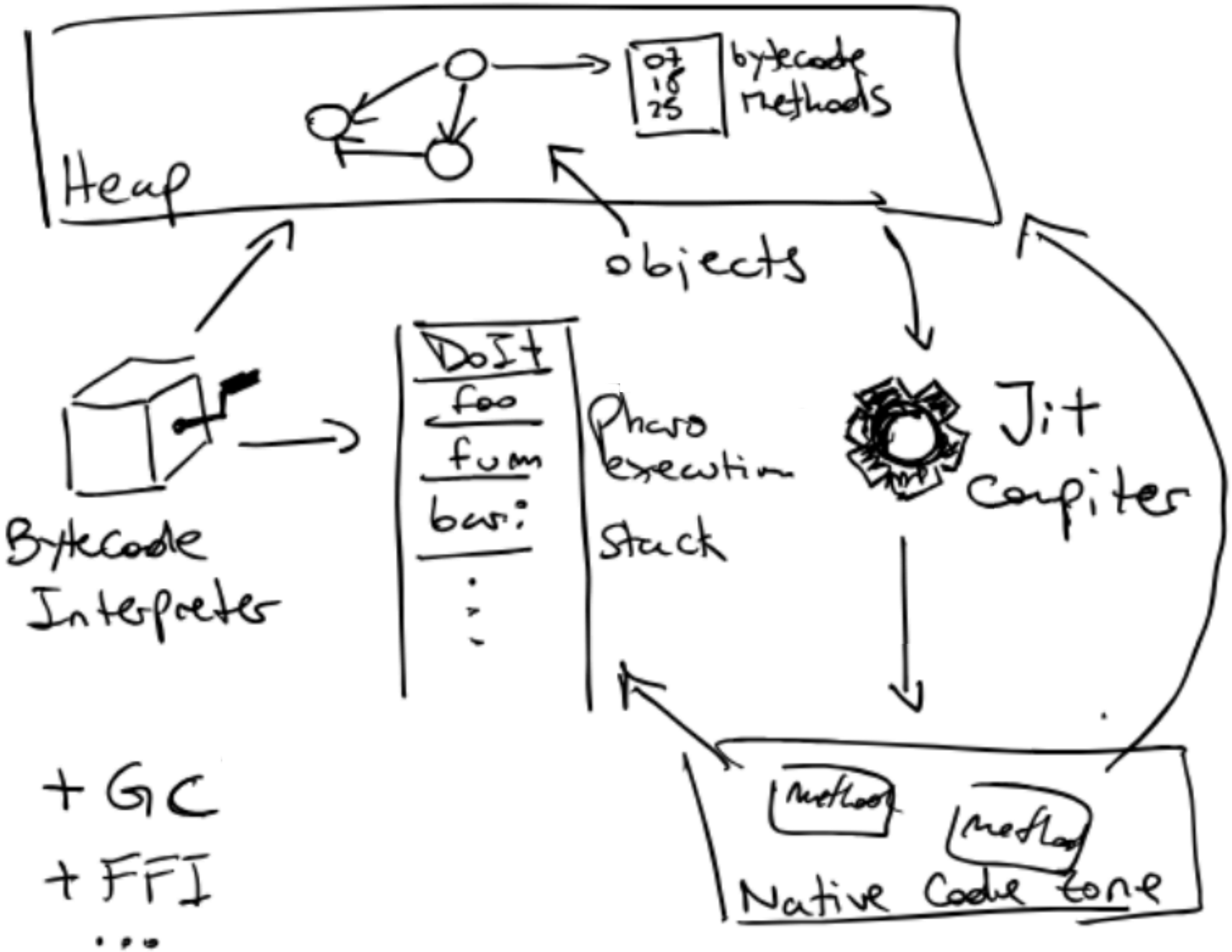
Pharo VM Components



Pharo VM Components

Different Components That interact

Interactions are not as clear to the user…



Performance Tuning of an Application

- Different Parameters to Tune Up (e.g., Memory Size, Code Cache Size, etc)
- Parameters depends on the application (e.g., method working set, object creation rate)
- Parameters are related with each other (e.g., modifying one may affect negative other)

Performance Tuning of an Application

- Different Parameters to Tune Up (e.g., Memory Size, Code Cache Size, etc)
- Parameters depends on the application (e.g., method working set, object creation rate)
- Parameters are related with each other (e.g., modifying one may affect negative other)

We need correct information to set them up

Current VM Performance Indicators

- Pharo VM exposes some basic statistics about runtime, e.g. :
  - Number of GC (Scavenger / Full GC)
  - Time in GC (Scavenger / Full GC)
  - Total Execution Time
  - …

Current VM Performance Indicators

- Pharo VM exposes some basic statistics about runtime, e.g. :
  - Number of GC (Scavenger / Full GC)
  - Time in GC (Scavenger / Full GC)
  - Total Execution Time
  - …

They present basic total information and don’t identify steady state

It is not enough to identify performance issues in an application

Perfomance Indicators

Requirements

- **Precise Information**: We collect all events occurring.
- **App Execution Identification:** we need to identify when the app starts.
- **Time Correlated**: Events should have timestamps.
- **Events Expressing VM Behaviour**: counters and indicators should expose the behaviour of VM components.
- **Scalable:** it should handle long time running applications.
- **Usable:** presenting information relevant / accessible to the user.

Perfomance Indicators

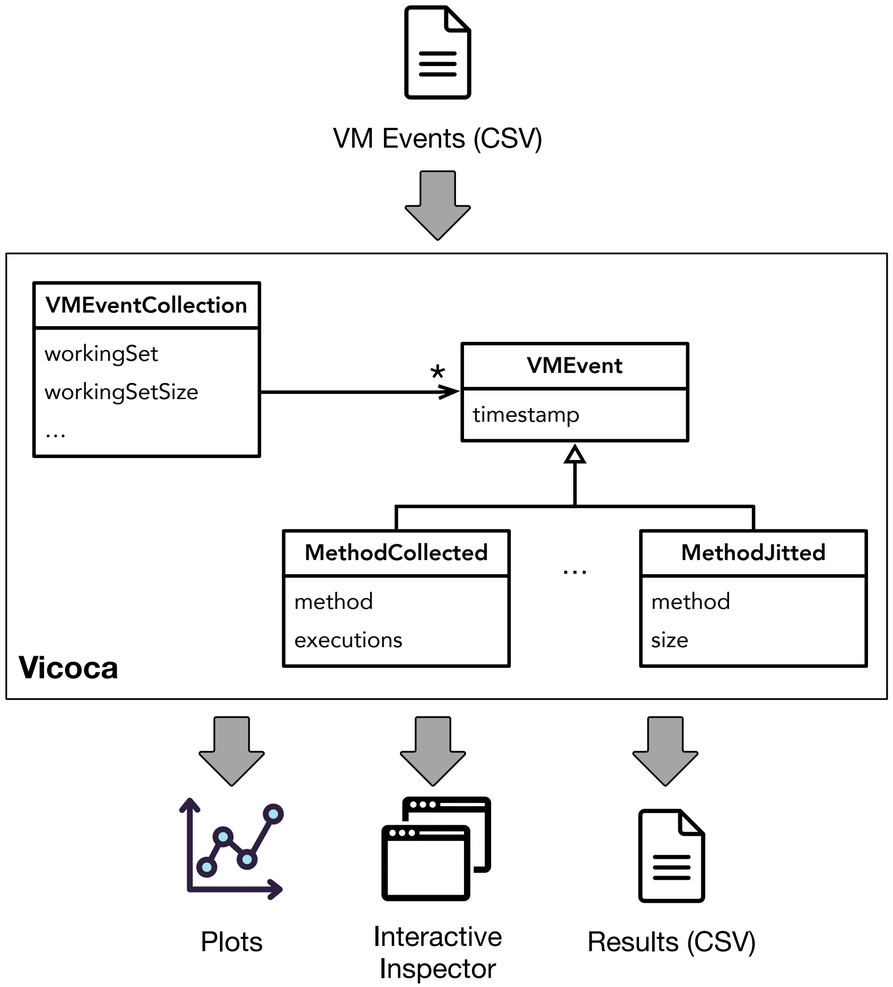
Requirements

We want to relate and analyse events from different one or many different executions

- **Precise Information**: We collect all events occurring.
- **App Execution Identification:** we need to identify when the app starts.
- **Time Correlated**: Events should have timestamps.
- **Events Expressing VM Behaviour**: counters and indicators should expose the behaviour of VM components.
- **Scalable:** it should handle long time running applications.
- **Usable:** presenting information relevant / accessible to the user.

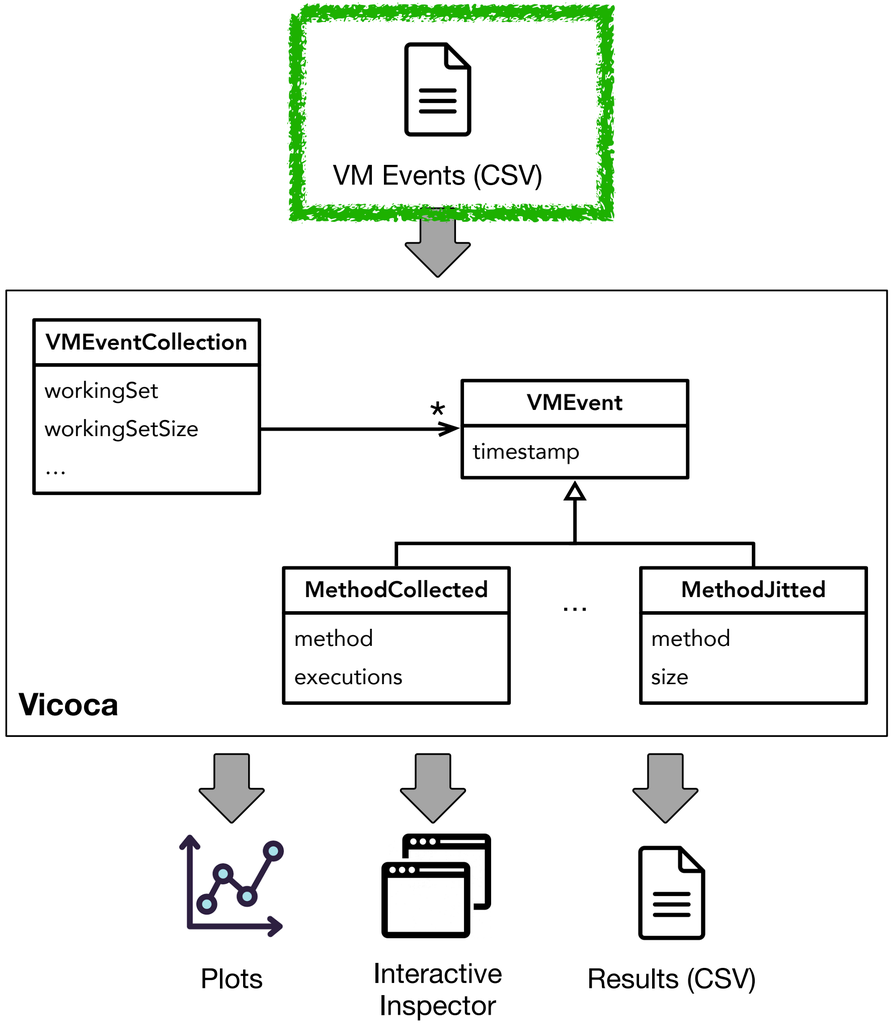
Vicoca

A tool for collecting and Analysing Events for the Pharo VM



Vicoca

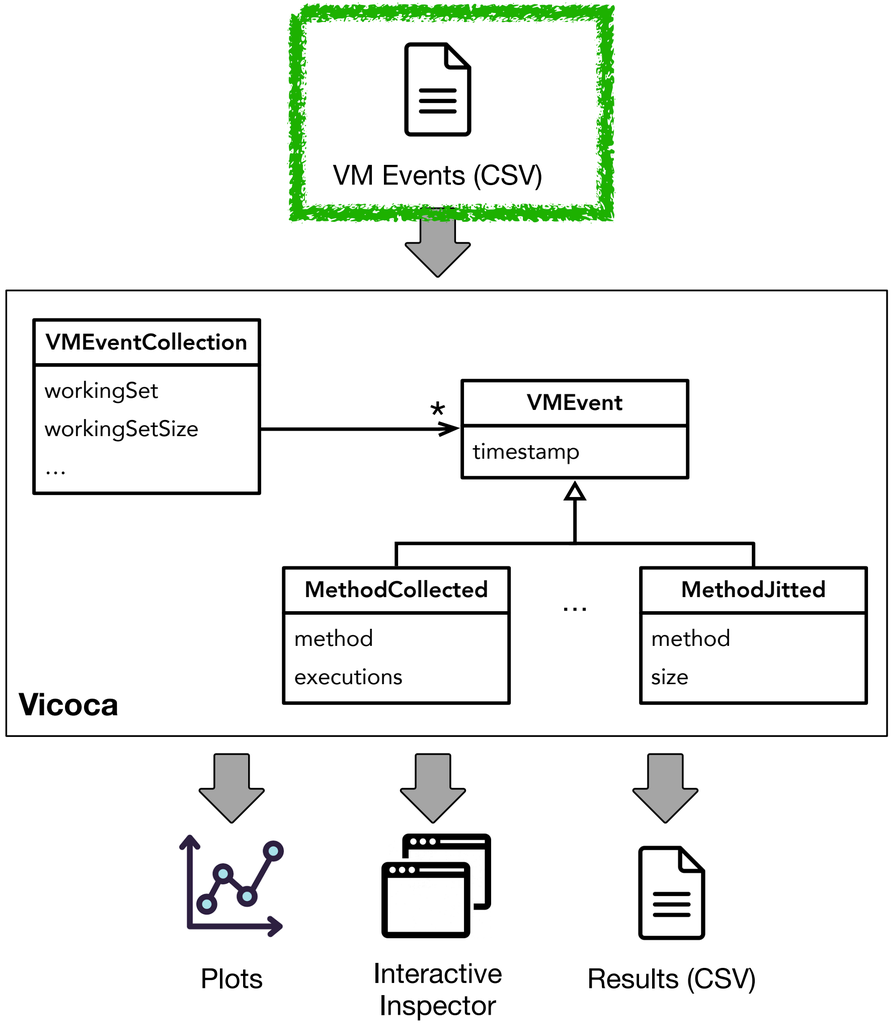
A tool for collecting and Analysing Events for the Pharo VM



VM extended with a event-based profiler. Capturing all events with time information

Vicoca

A tool for collecting and Analysing Events for the Pharo VM

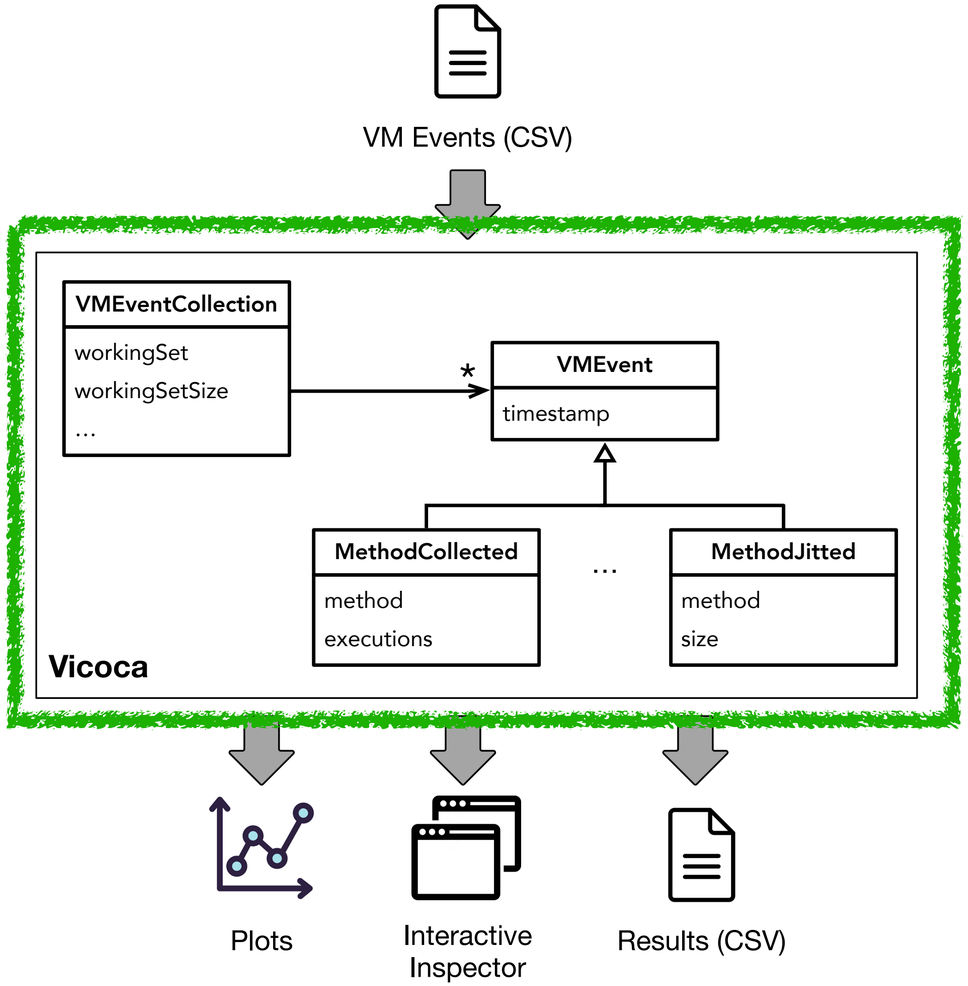


VM extended with a event-based profiler. Capturing all events with time information

Each event stores raw data. All events are recorded to later analysis.

Vicoca

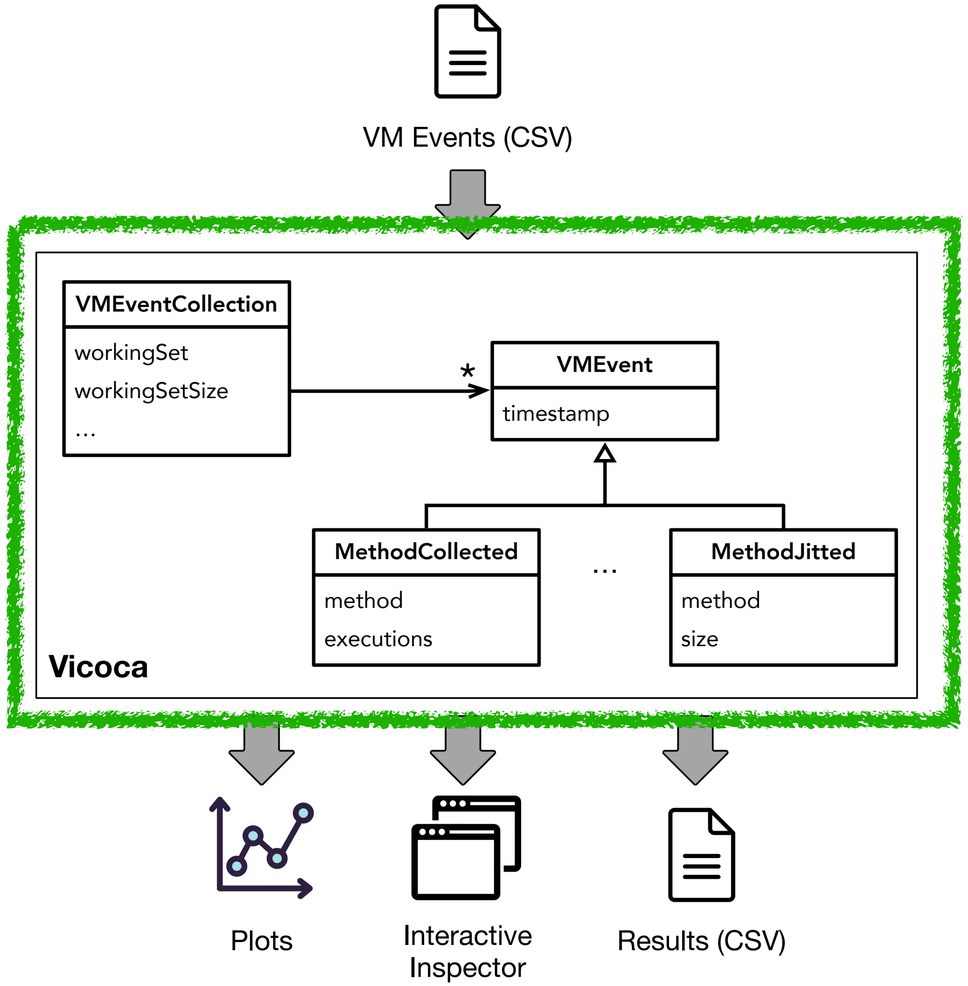
A tool for collecting and Analysing Events for the Pharo VM



After execution. Events are loaded in Vicoca to analyse them.

Vicoca

A tool for collecting and Analysing Events for the Pharo VM

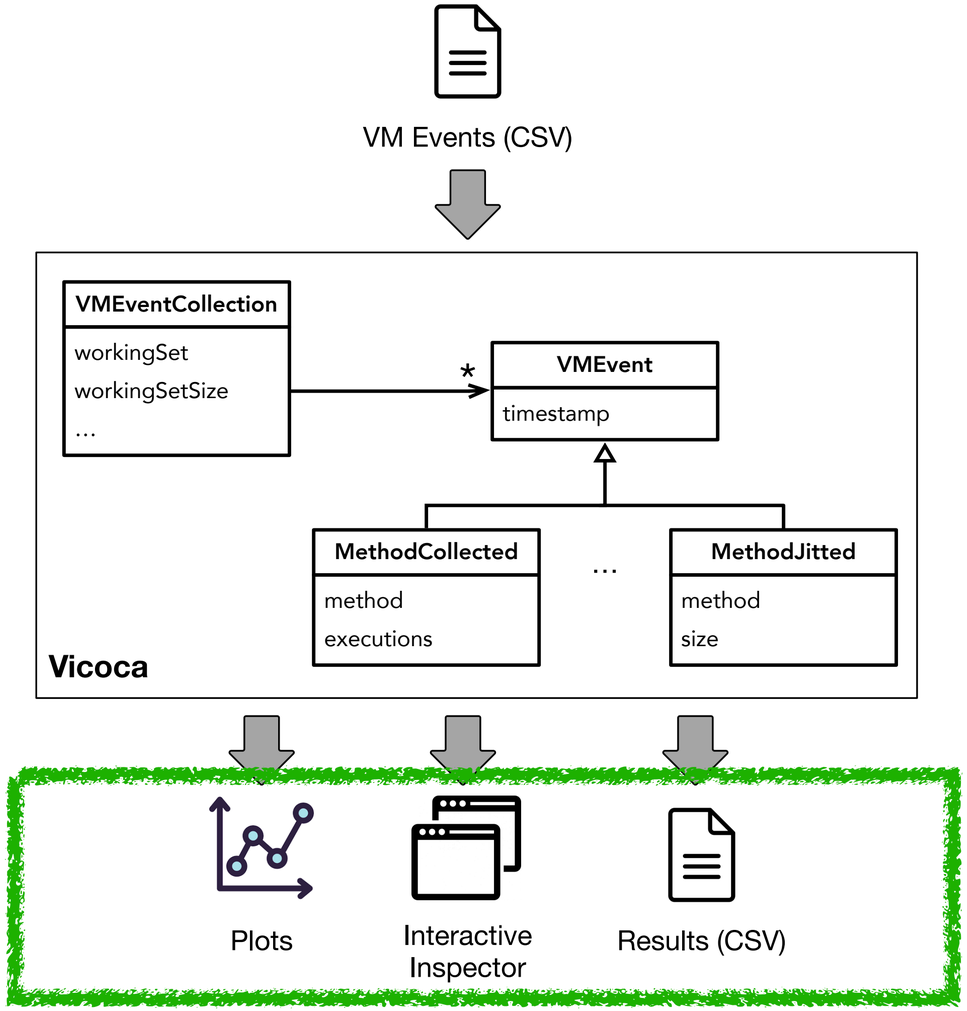


After execution. Events are loaded in Vicoca to analyse them.

Each event is converted into a rich object model, that is used by the tool. Also, relations are rebuilt and totals calculated

Vicoca

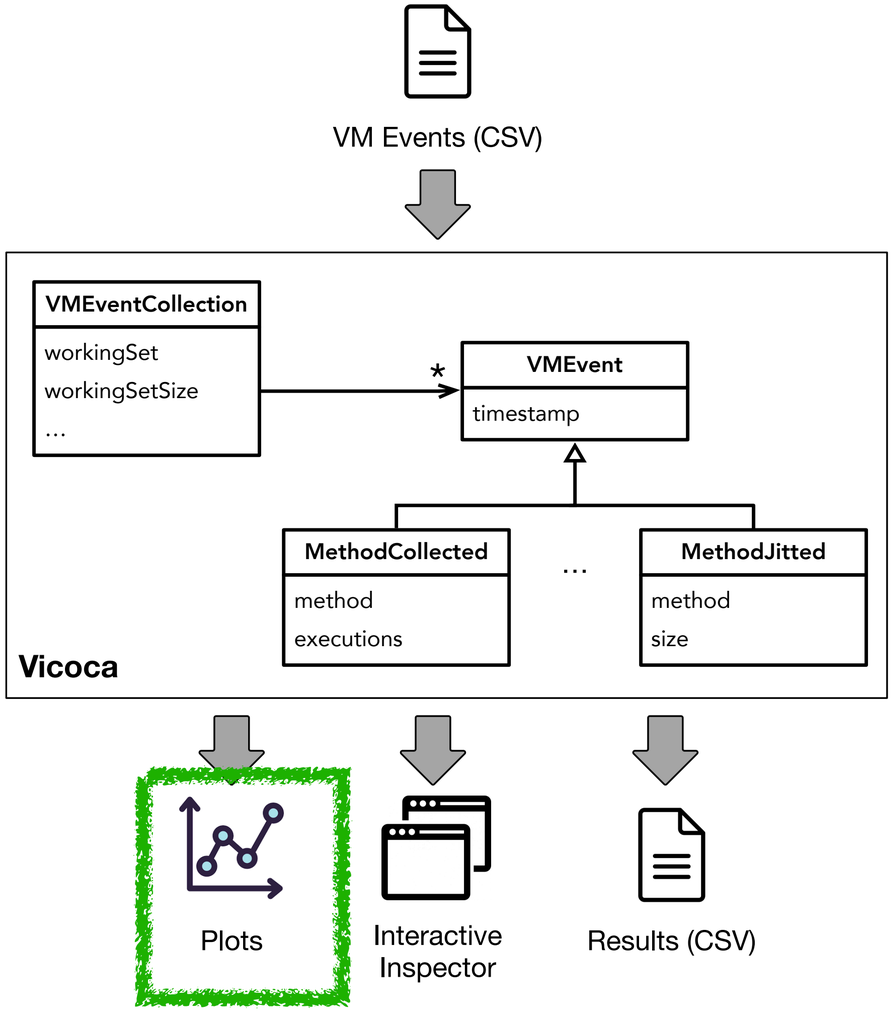
A tool for collecting and Analysing Events for the Pharo VM



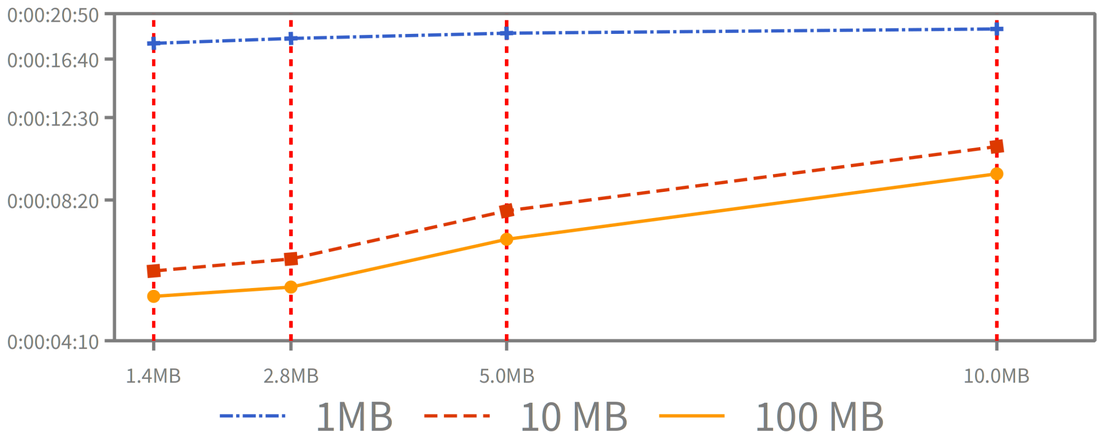
Model is used to generate output useful for the user.

Vicoca

A tool for collecting and Analysing Events for the Pharo VM

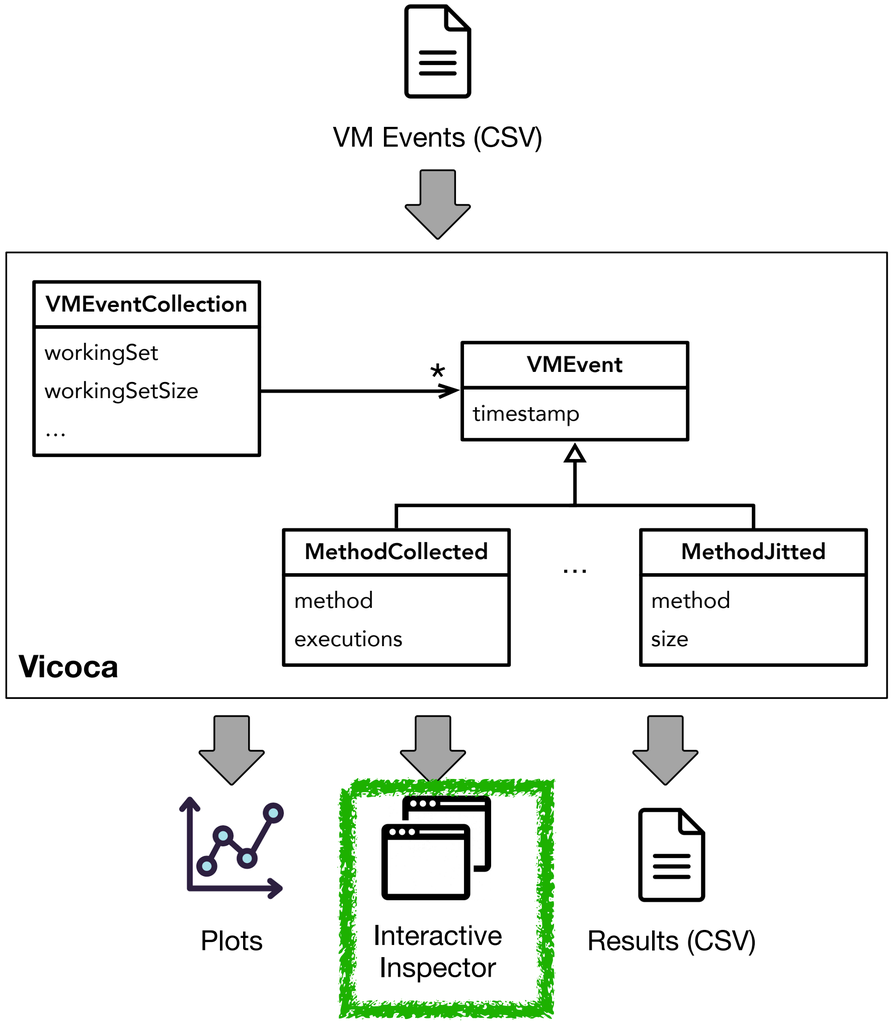


Predefined plots, support for plot scripting using Pharo plotting library (Roassal)



Vicoca

A tool for collecting and Analysing Events for the Pharo VM



The object model is navigable using custom extensible inspectors

Working Set	Raw	Breakpoints	Meta
Method			Executions
ProtoObject >> class			2676530993
ArrayedCollection >> size			2011757467
SmallInteger >> =			1495640219
Symbol >> =			1341766248
SmallInteger >> \\			1160873709
Object >> enclosedElement			1145750258
Object >> =			816070216
Object >> at:put:			764810930
Object >> basicAt:			711668845
WriteStream >> nextPut:			695757809
CompiledCode >> objectAt:			685686635

A Case Study

Analysing the installation of Moose

- Pharo is an image based language, all code and objects is stored in a binary format.
- Pharo Code is installed from the source, it is compiled to be loaded in the image.
- Moose (https://modularmoose.org/) is a software analysis tool developed on Pharo.
- Loading Moose compiles 1,662 classes and 51,053 methods.
- It takes 15 minutes without any performance tuning.

A Case Study

Analysing the installation of Moose

- Two initial suspects:
  - Code Cache Trashing
  - Excessive number of full GC executions

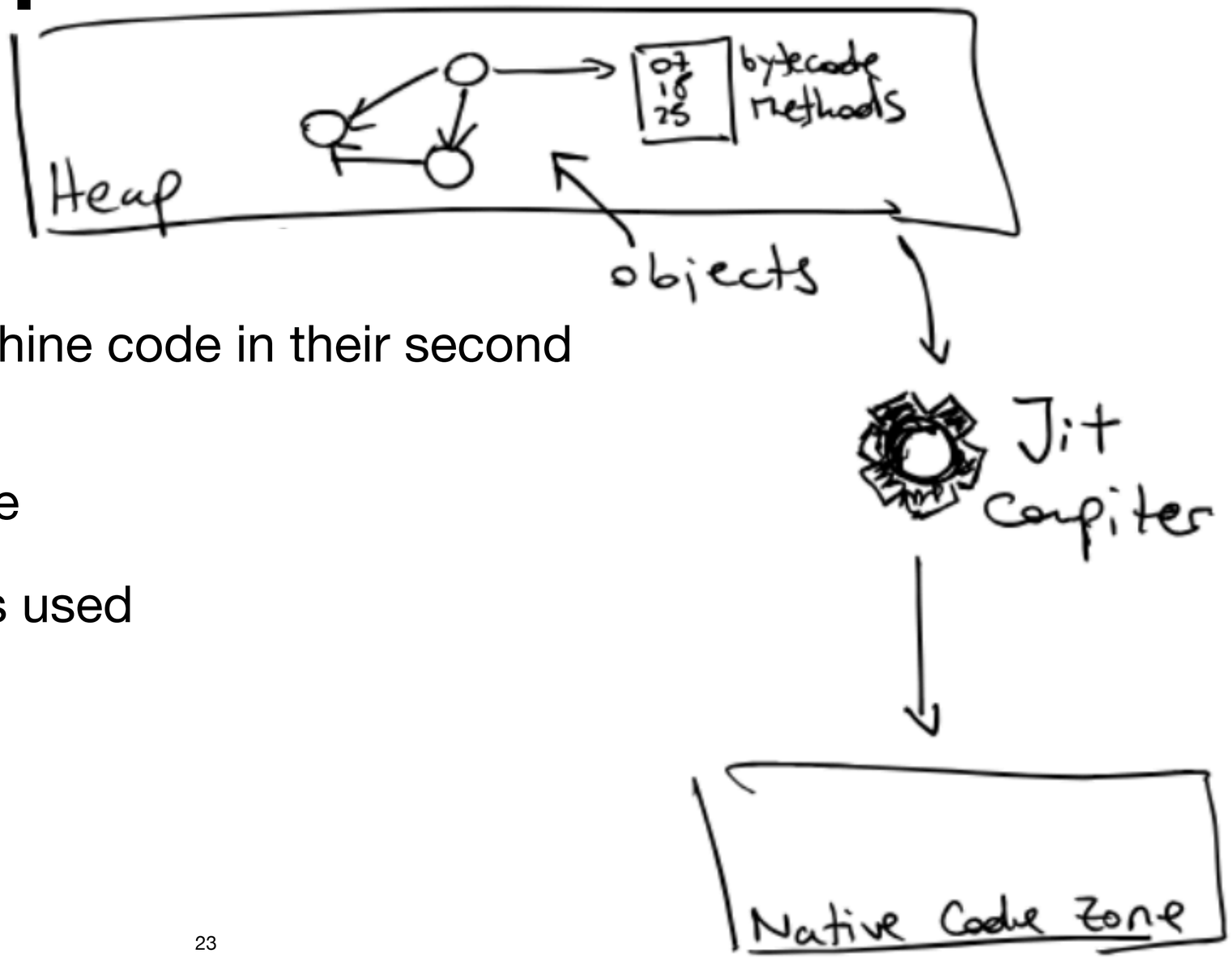
A Generational Garbage Collector

- Object space divided by generations
- Old and Permanent objects are kept in the old space
- New objects are kept in the young space.

A Generational Garbage Collector

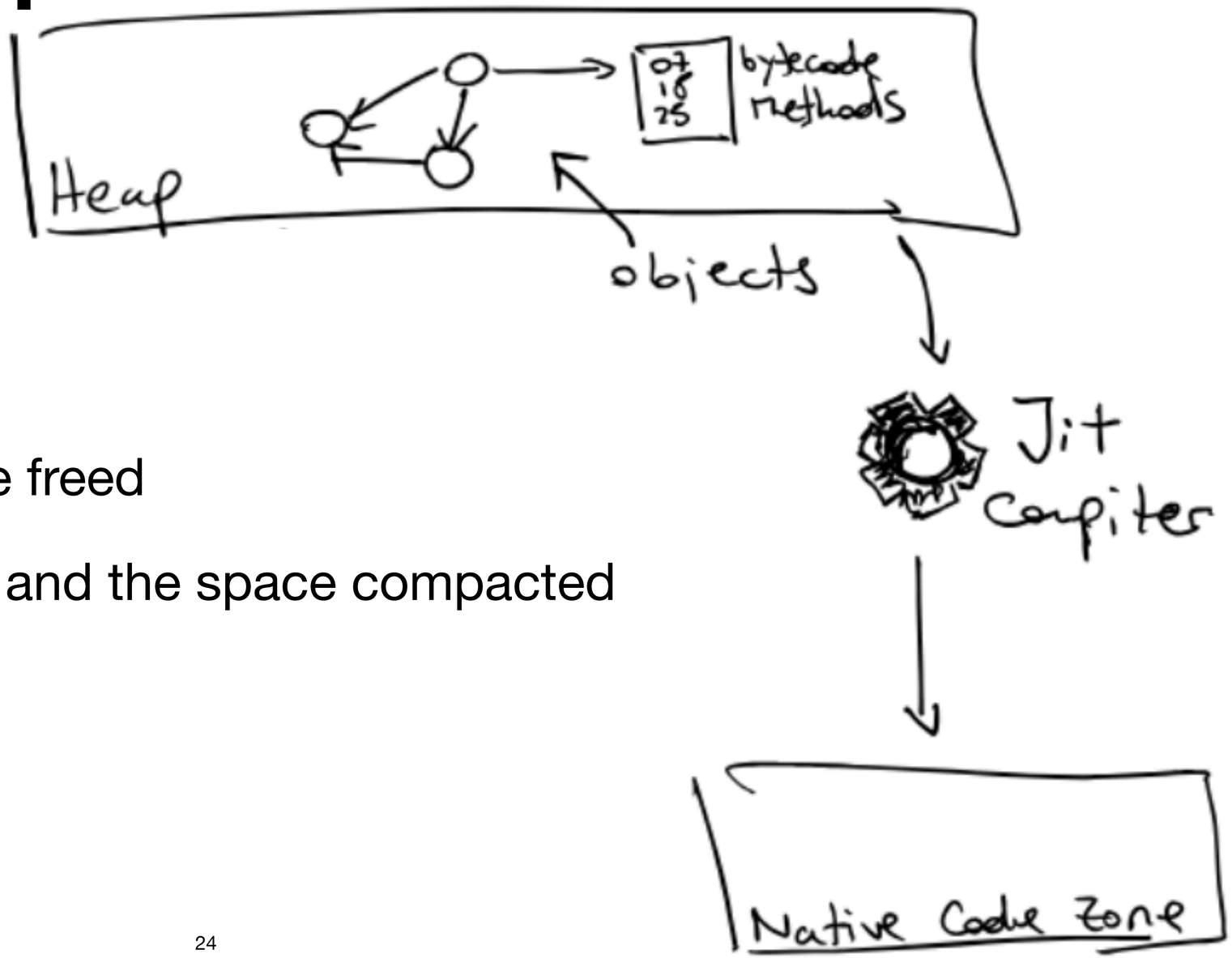
- Younger Generations use Copy Collector Scavenger
- Older Generations use Mark and Compact.

Baseline JIT Compiler



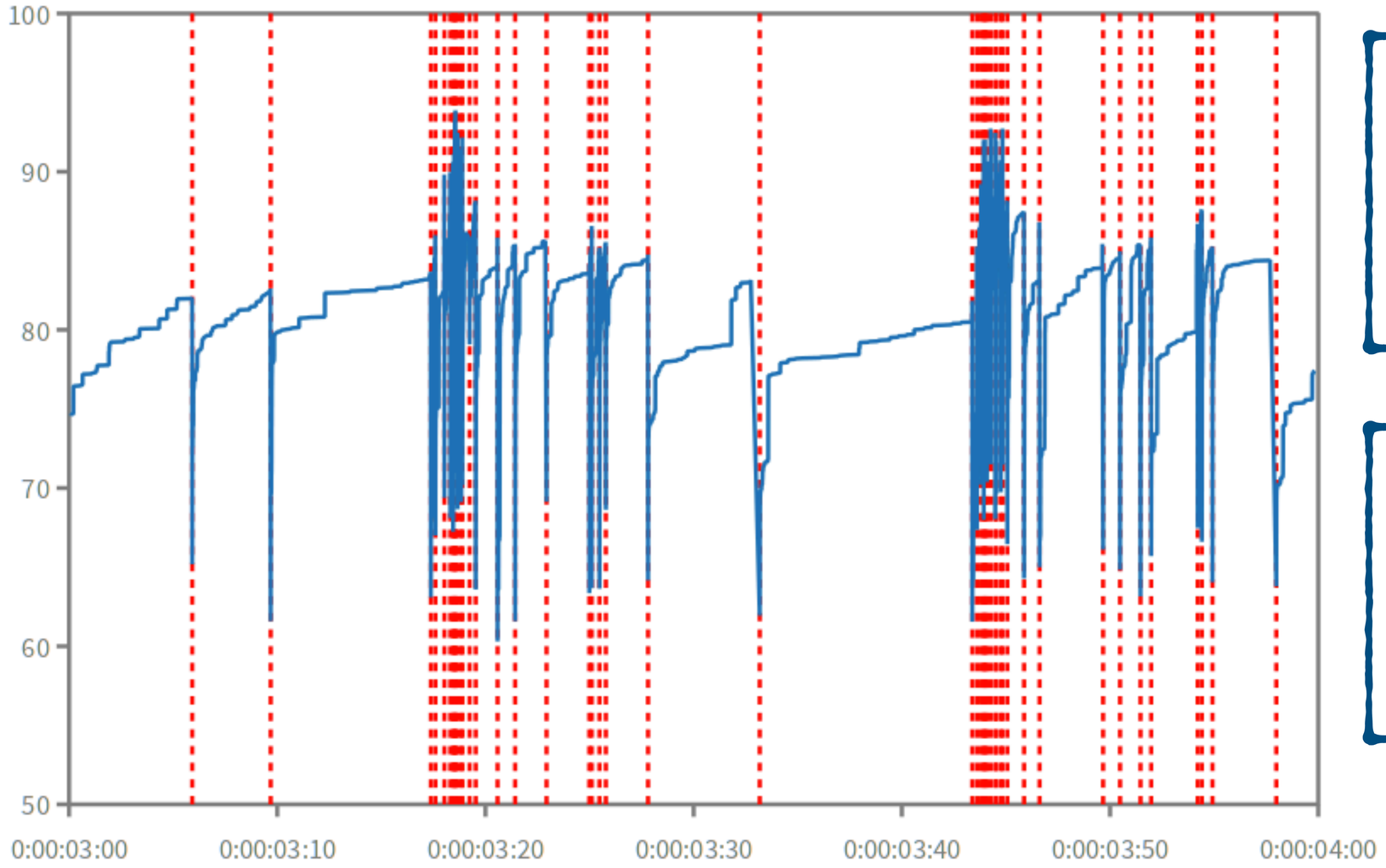
- Methods are compiled to machine code in their second execution in a row
- Stored in a Native Code Cache
- Then, machine code version is used

Baseline JIT Compiler



- Code Cache has a fixed size
- When it is full, space has to be freed
- Methods have to be selected, and the space compacted

First Suspect: Analysing Code Cache

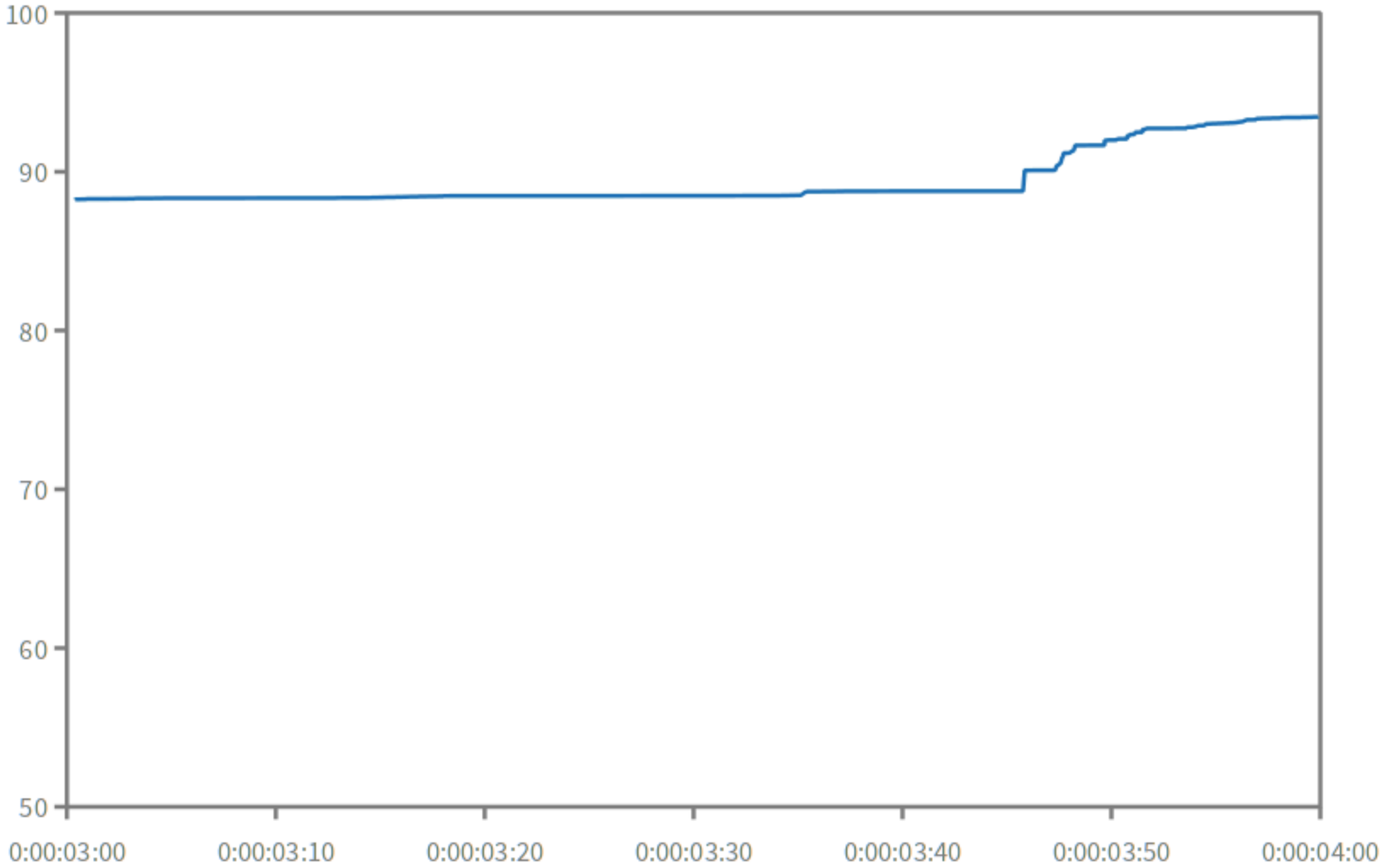


Analysing Events
We see trashing in the code cache

We need to increase the size of the code cache

Code Cache occupation rate when the application is in steady-state. In blue, the occupation rate of the code cache. In red, the compaction events. (1.44 MB)

First Suspect: Improving Code Cache



App Working Set fits in the code Cache.

Code Cache occupation rate when the application is in steady-state. In blue, the occupation rate of the code cache. In red, the compaction events. (10 MB)

Second Suspect: Reducing Full GC time

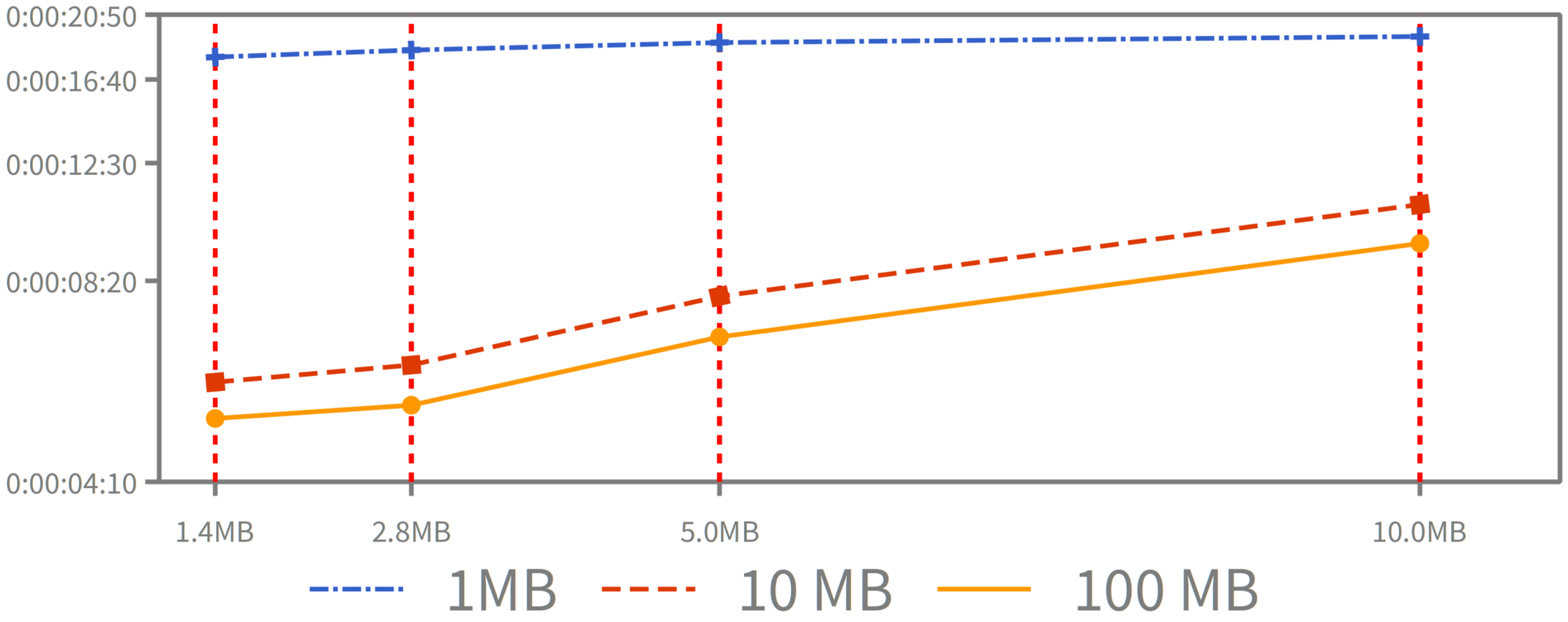
- For default Young Space size (1MB)
  - We have 26 Full GC executions
  - Execution Time is around 15 minutes.
- For 100MB of young space
  - We have 6 Full GC executions
  - Total Execution Time is around 5 minutes

Second Suspect: Reducing Full GC time

- For default Young Space size (1MB)
  - We have 26 Full GC executions
  - Execution Time is around 15 minutes.
- For 100MB of young space
  - We have 6 Full GC executions
  - Total Execution Time is around 5 minutes

We have improved the issue… let's apply both

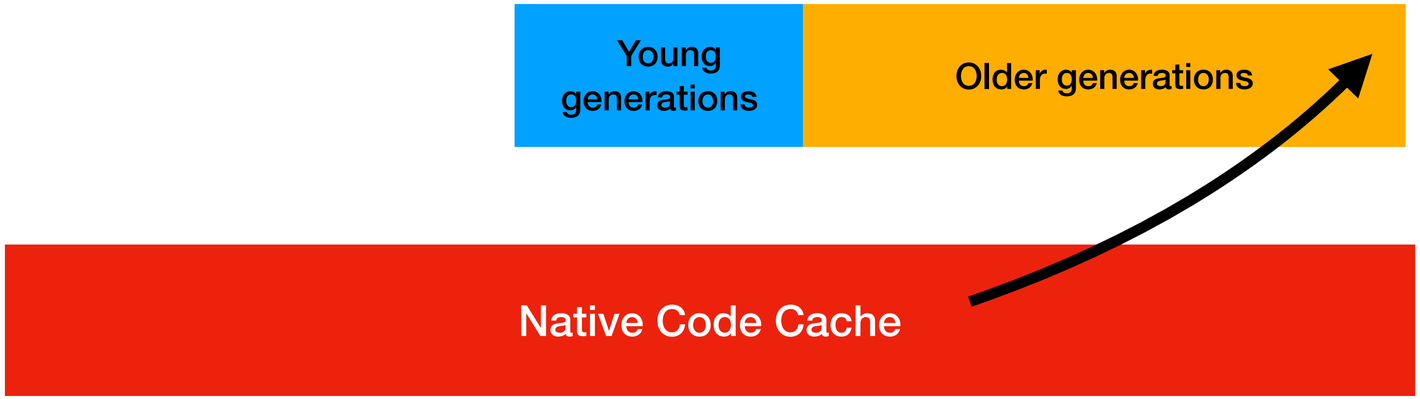
Applying Both… Unexpected results



Execution time for different Young Space size (1MB, 10MB, 100MB) and Cache Sizes (1.44MB, 2.8MB, 5MB, 10MB)

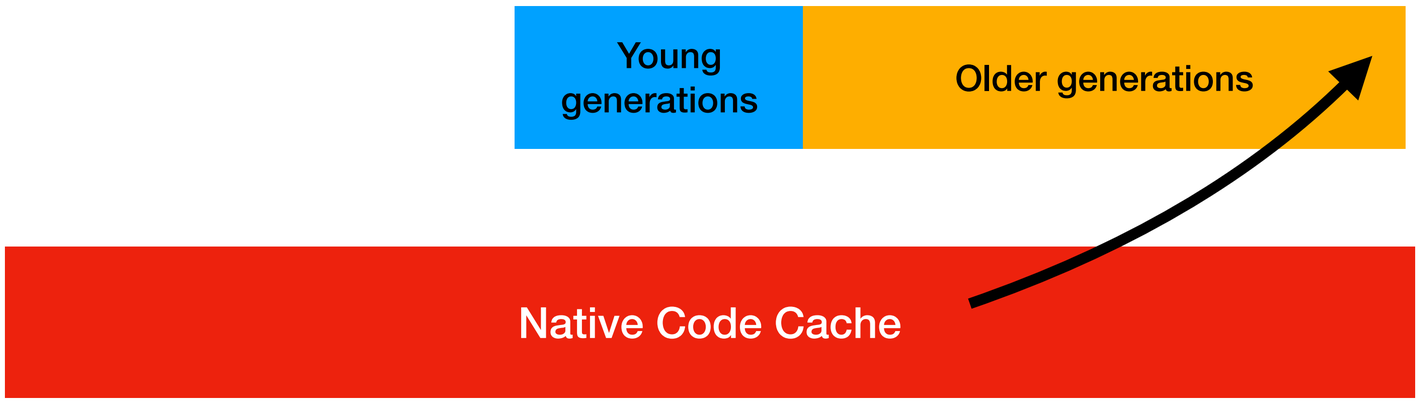
Code Zone Size / GC Time Relationship

- Native methods have inlined object references
  - native code is a root of the old space, it has to be traversed on Full GC
  - when objects move, native code is *scanned*, *decompiled* and *patched*



Code Zone Size / GC Time Relationship

- Native methods have inlined object references
  - native code is a root of the old space, it has to be traversed on Full GC
  - when objects move, native code is *scanned*, *decompiled* and *patched*

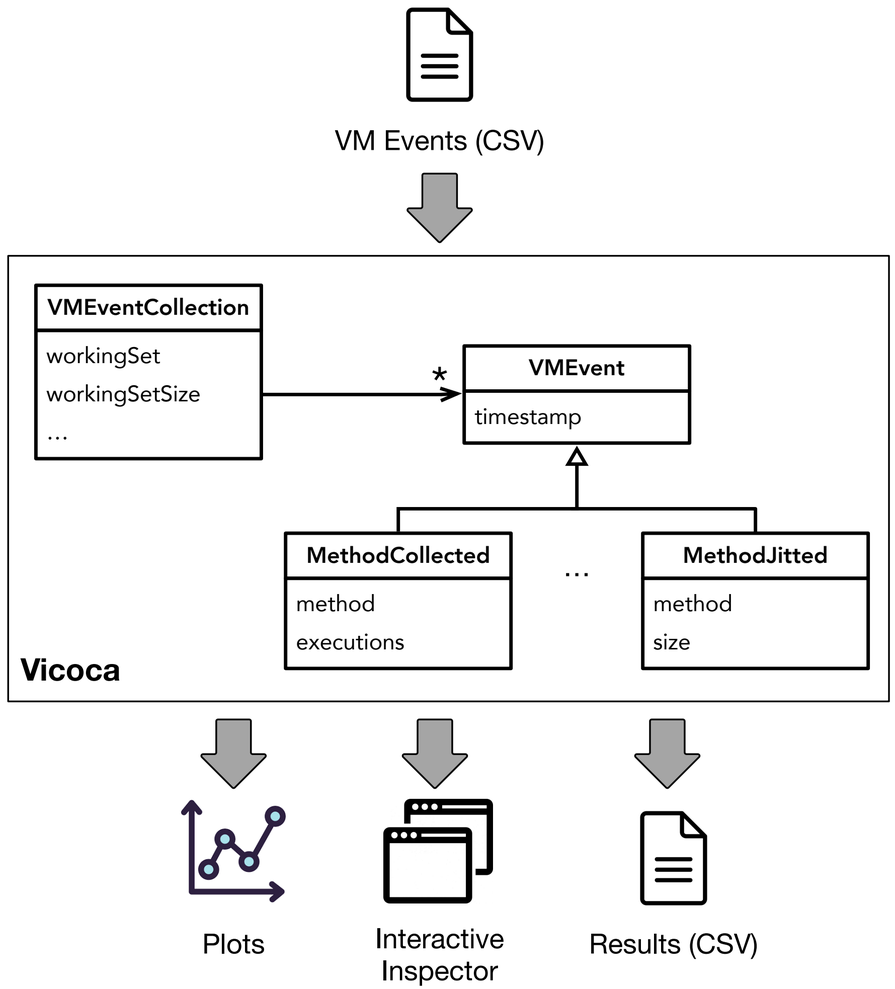


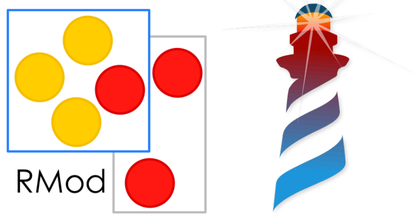
Bigger Code Cache… puts pressure on the GC

Future Work

Towards getting gold from the data

- Automatic Detection of Bottlenecks
- Application Behaviour Identification
- Automatic Performance Hinting / Proposals
- Improved visualisations and tooling





Profiling Code Cache Behaviour via Events

Work In Progress Paper - MPLR 2021

- Vicoca a tool for:
  - Capturing and analysing Events from the VM
  - Correlating events and behaviour
  - VM components oriented events
  - Presenting Events in a Usable way