Scala Native

Denys Shabalin

EPFL

Scala Native

- Ahead-of-time compiler for Scala
- Originally announced on May 2016
- First release March 2017

Scala Native

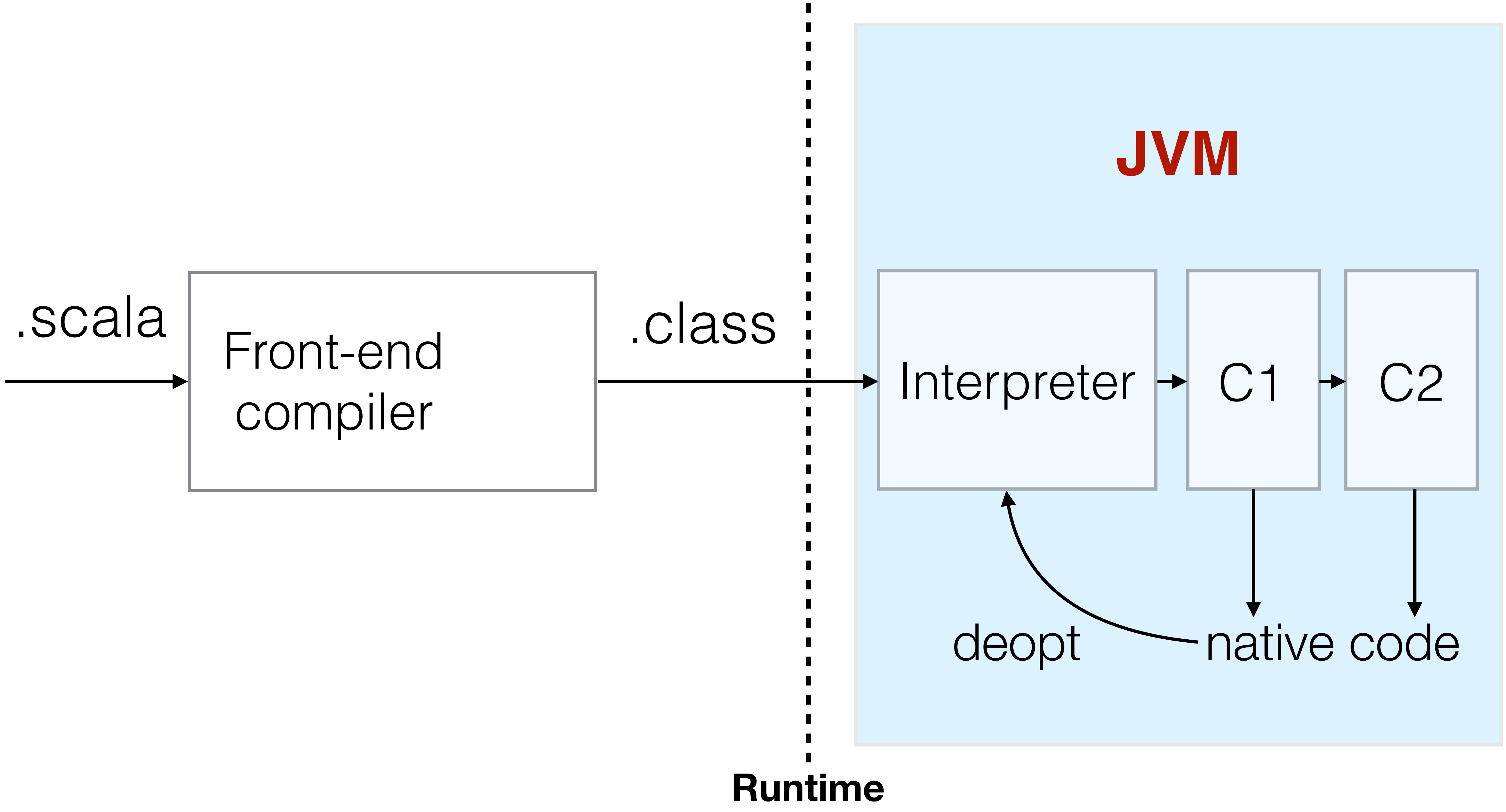
- Emits machine-dependent native code
- Build on top of LLVM compiler infrastructure
- Does whole-program optimization

Scala Native

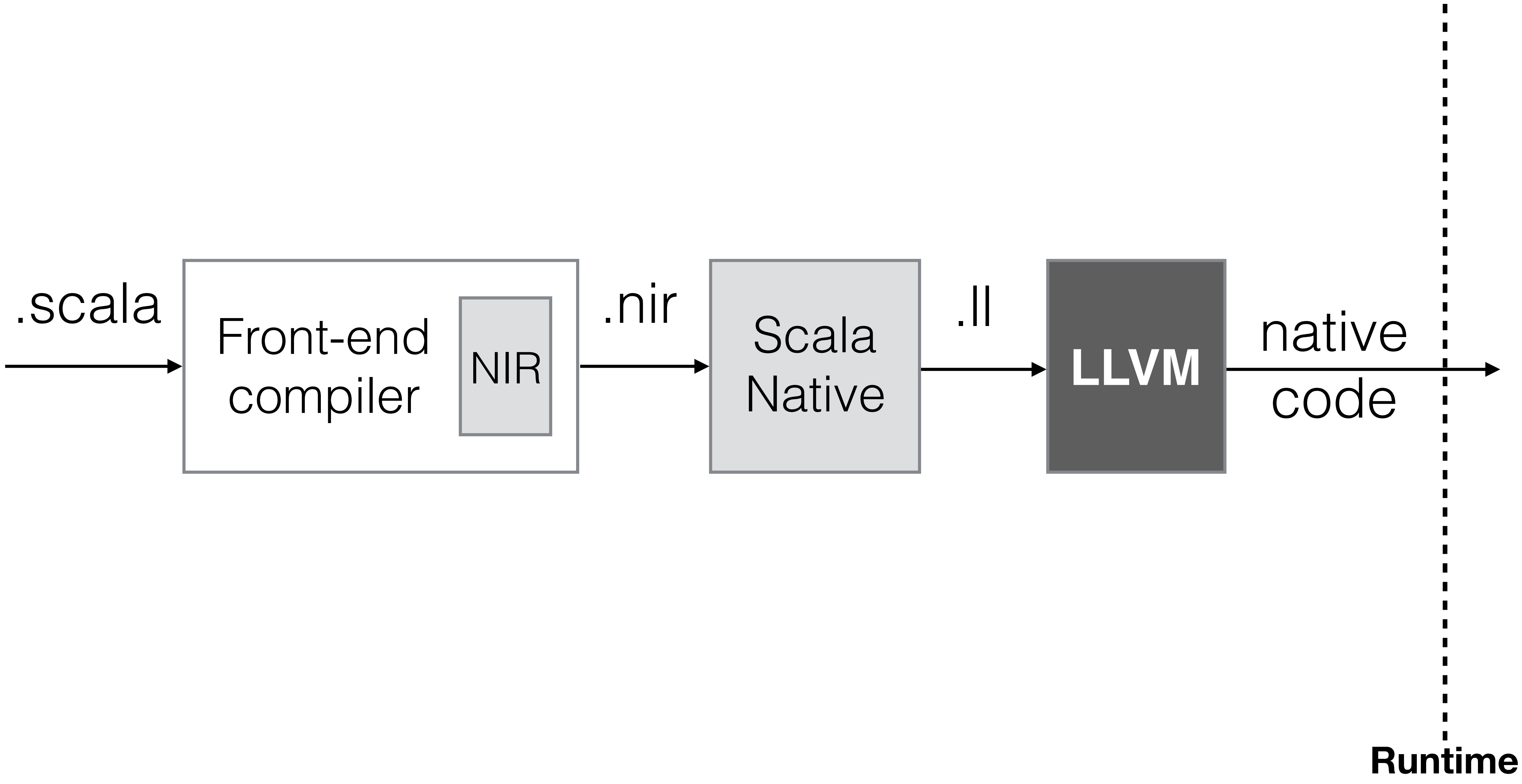
1. Compiled ahead-of-time
2. Highly compatible
3. Great interoperability
4. Predictable garbage collection

Compiled ahead-of-time

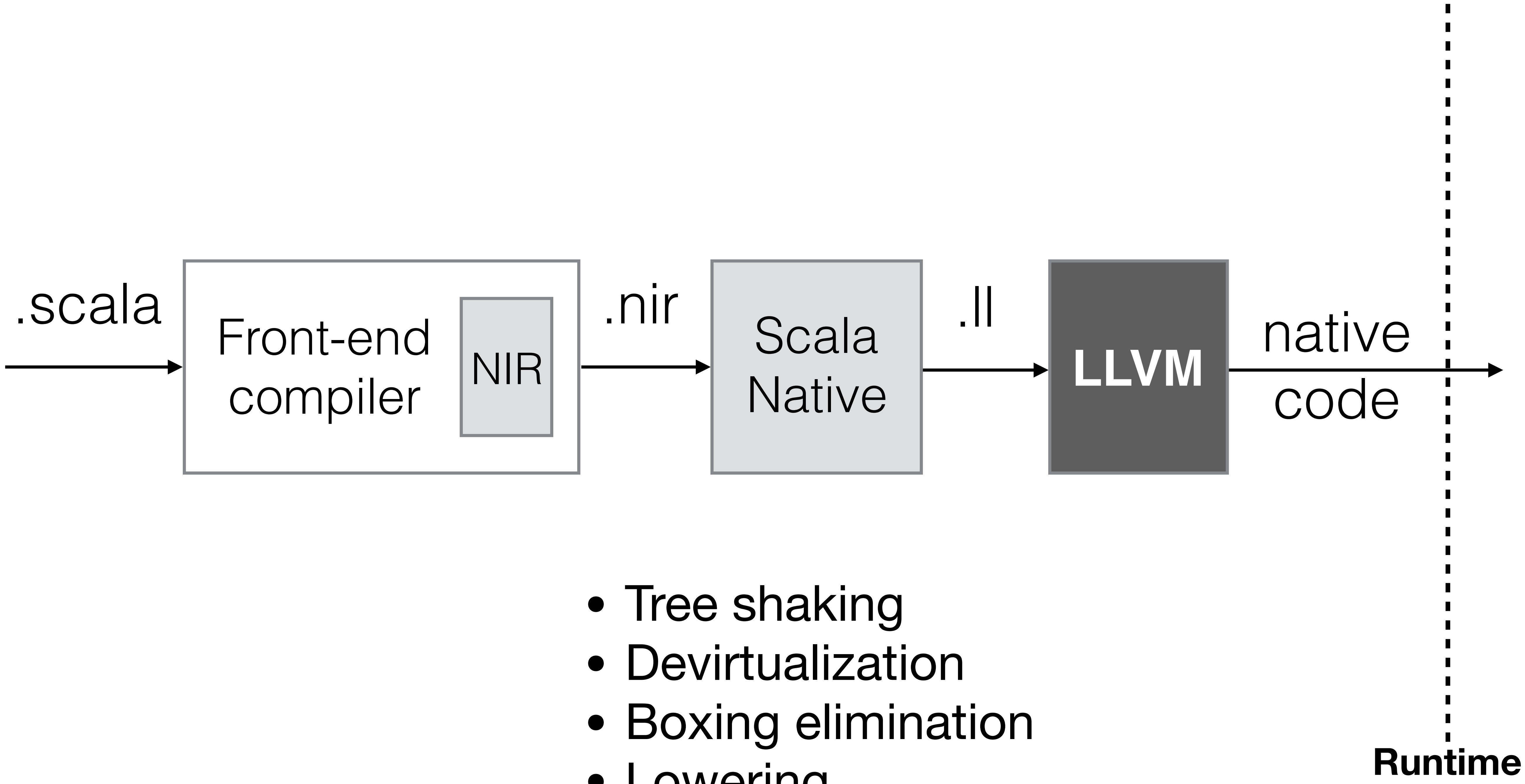
Just-in-time


.scala
Front-end compiler
.class
JVM
Interpreter
C1
C2
deopt
native code
Runtime

Ahead-of-time

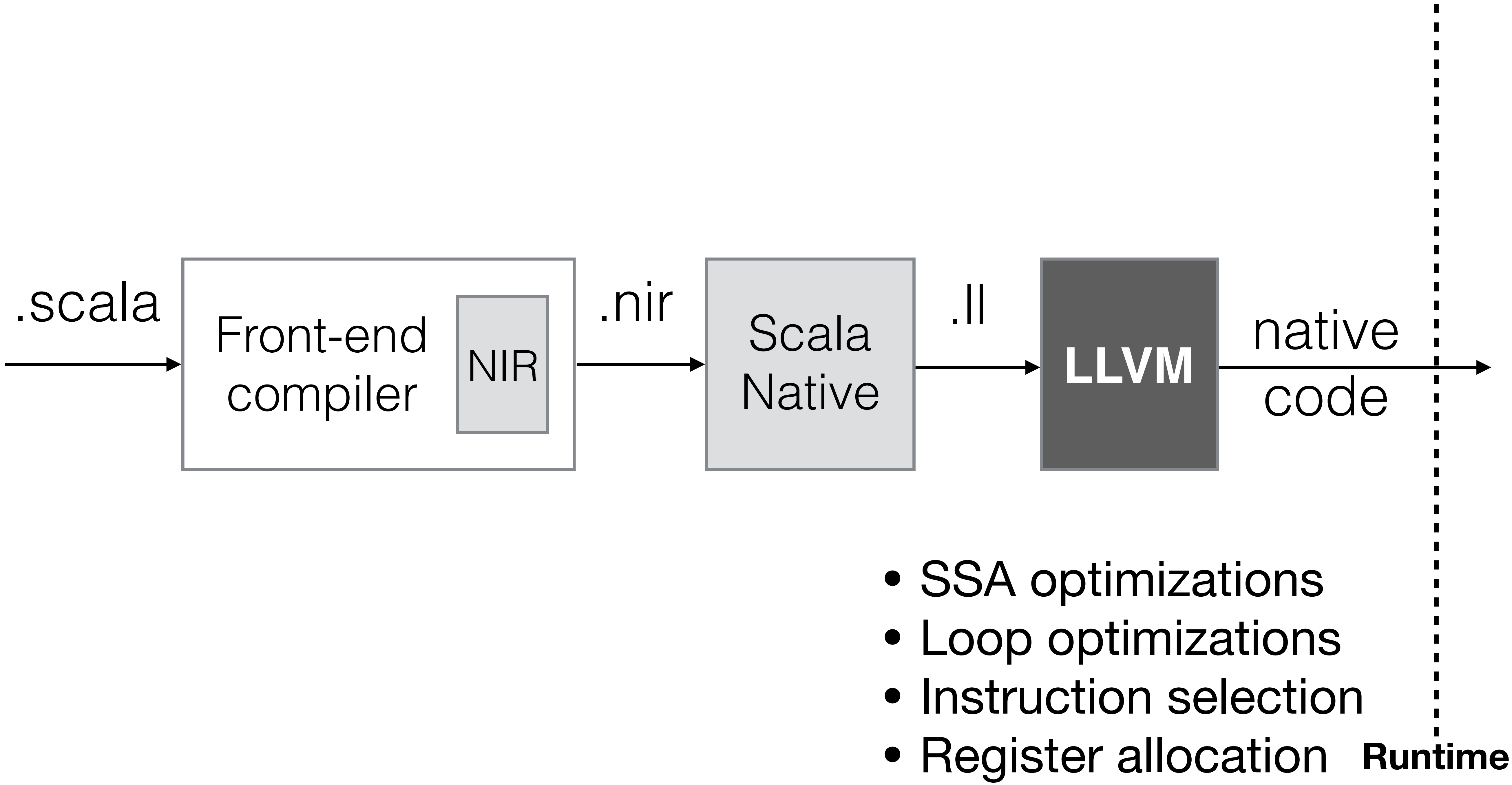


Ahead-of-time



- Tree shaking
- Devirtualization
- Boxing elimination
- Lowering

Ahead-of-time



- SSA optimizations
- Loop optimizations
- Instruction selection
- Register allocation
- Native code codegen

Ahead-of-time

1. Instant startup time
2. Predictable performance
3. Small self-contained binaries

Highly compatible

Compatibility

- The same language as the reference implementation of Scala on top of the JVM
- All of the language features are supported (yes, even the obscure parts like structural types)
- Standard distribution ships a compatible subset of the core libraries

Library Compatibility

- Most of Scala Library works out of the box

- `java.{lang, util, io, nio, net}`
  coverage is growing with every release
  http://www.scala-native.org/en/latest/lib/javalib.html

- Build-in bindings for libc and POSIX

Build Compatibility

- Sbt is an officially supported build tool via a plugin
- Supports cross publishings against JVM, JS and Native
- `sbt new scala-native/scala-native.g8`

Easy interoperability

Interoperability

Calling C code is as simple as:

```
import scalanative.native._

@extern
object libfoo {
  def foo(data: Ptr[Int]): Unit = extern
}

val data = stdlib.malloc(size)
libfoo.foo(data)
```

Interoperability

- Low-level types:
  `Ptr, CStructN[T1, …, TN], CArray[T, N]`

- Easy forward-declaration-based calls to C code

- Zero performance overhead

Predictable
Garbage Collection

Garbage Collection

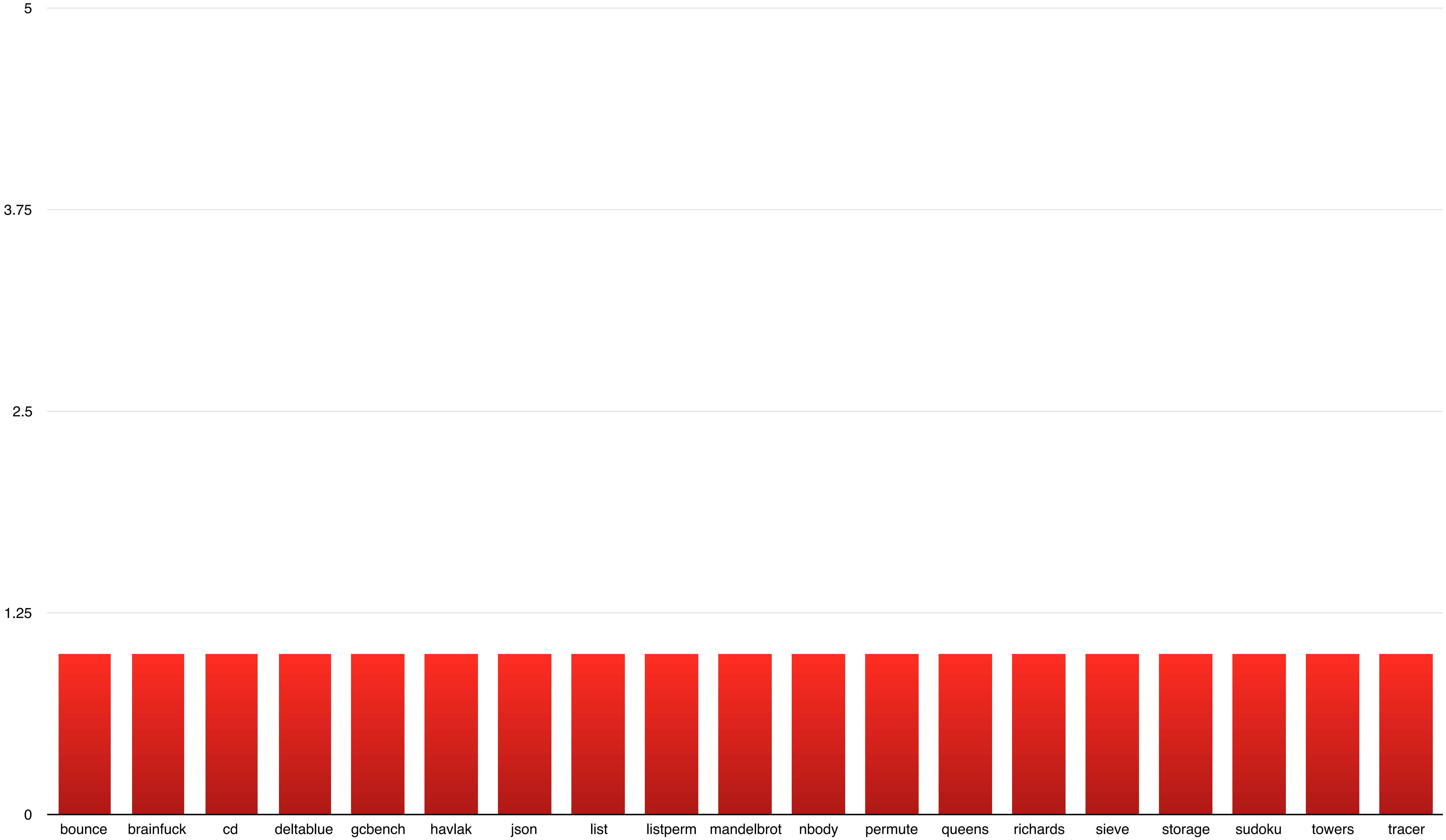
Three garbage collection options:

- Boehm
- None
- Immix

Boehm GC

- Our first garbage collector
- Fully conservative GC, originally designed for C/C+
- http://www.hboehm.info/gc/

Boehm GC


5
3.75
2.5
1.25
0
bounce
brainfuck
cd
deltablue
gcbench
havlak
json
list
listperm
mandelbrot
nbody
permute
queens
richards
sieve
storage
sudoku
towers
tracer

No GC

- You can switch the garbage collector off completely
- Memory gets allocated but never freed
- Zero garbage collection overhead

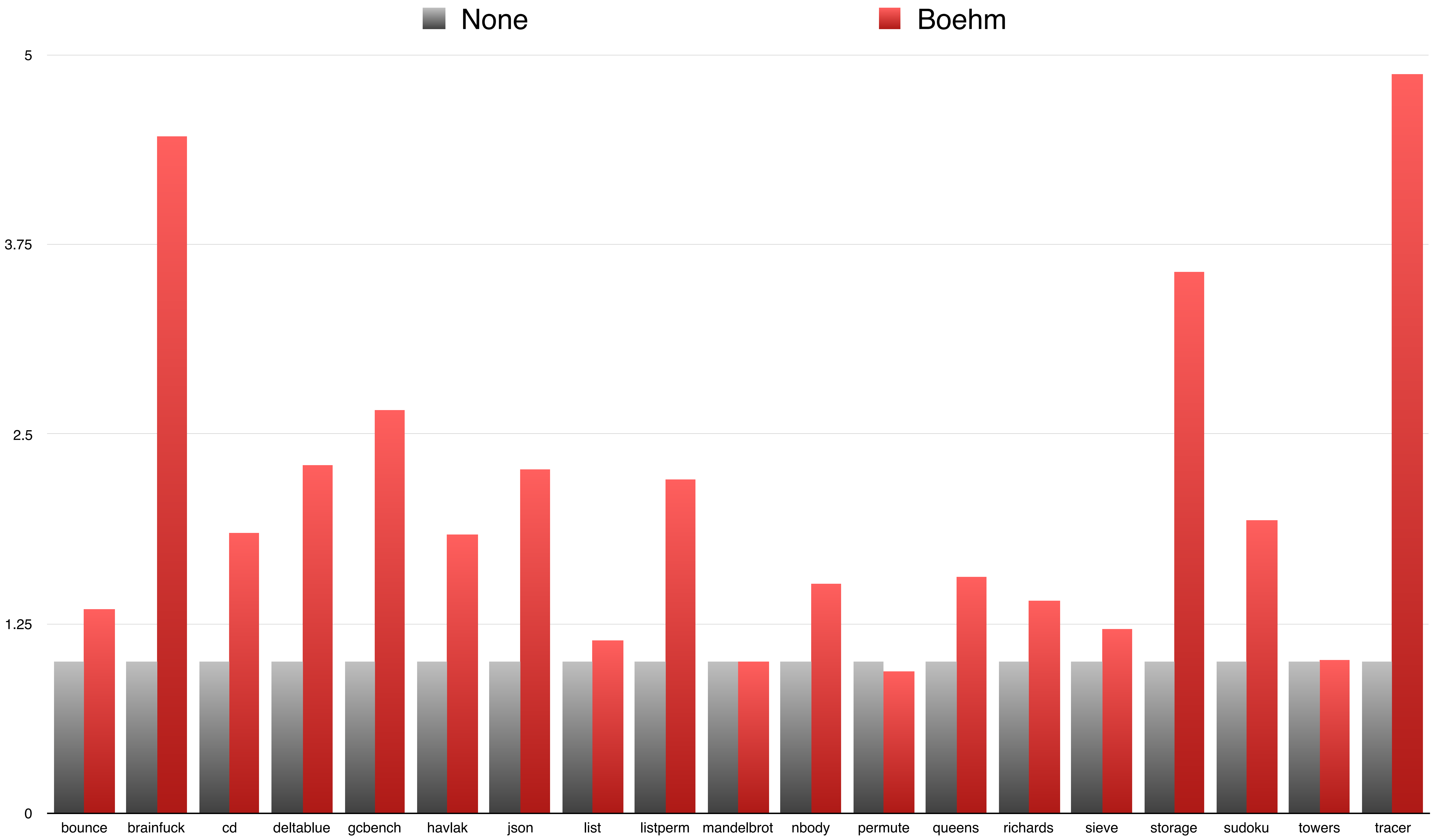
From: k...@rational.com (Kent Mitchell)
Subject: Re: Does memory leak?
Date: 1995/03/31
newsgroups: comp.lang.ada

This sparked and interesting memory for me. **I was once working with a customer who was producing on-board software for a missile.** In my analysis of the code, I pointed out that they had a number of problems with storage leaks. Imagine my surprise when the customers chief software engineer said "Of course it leaks". He went on to point out that they had calculated the amount of memory the application would leak in the total possible flight time for the missile and then doubled that number. **They added this much additional memory to the hardware to "support" the leaks. Since the missile will explode when it hits it's target or at the end of it's flight, *the ultimate in garbage collection* is performed without programmer intervention.**

```
--
Kent Mitchell                    | One possible reason that things aren't
Technical Consultant             | going according to plan is .....
Rational Software Corporation    | that there never *was* a plan!
```

https://groups.google.com/forum/message/raw?msg=comp.lang.ada/E9bNCvDQ12k/1tezW24ZxdAJ

Cost of Boehm GC

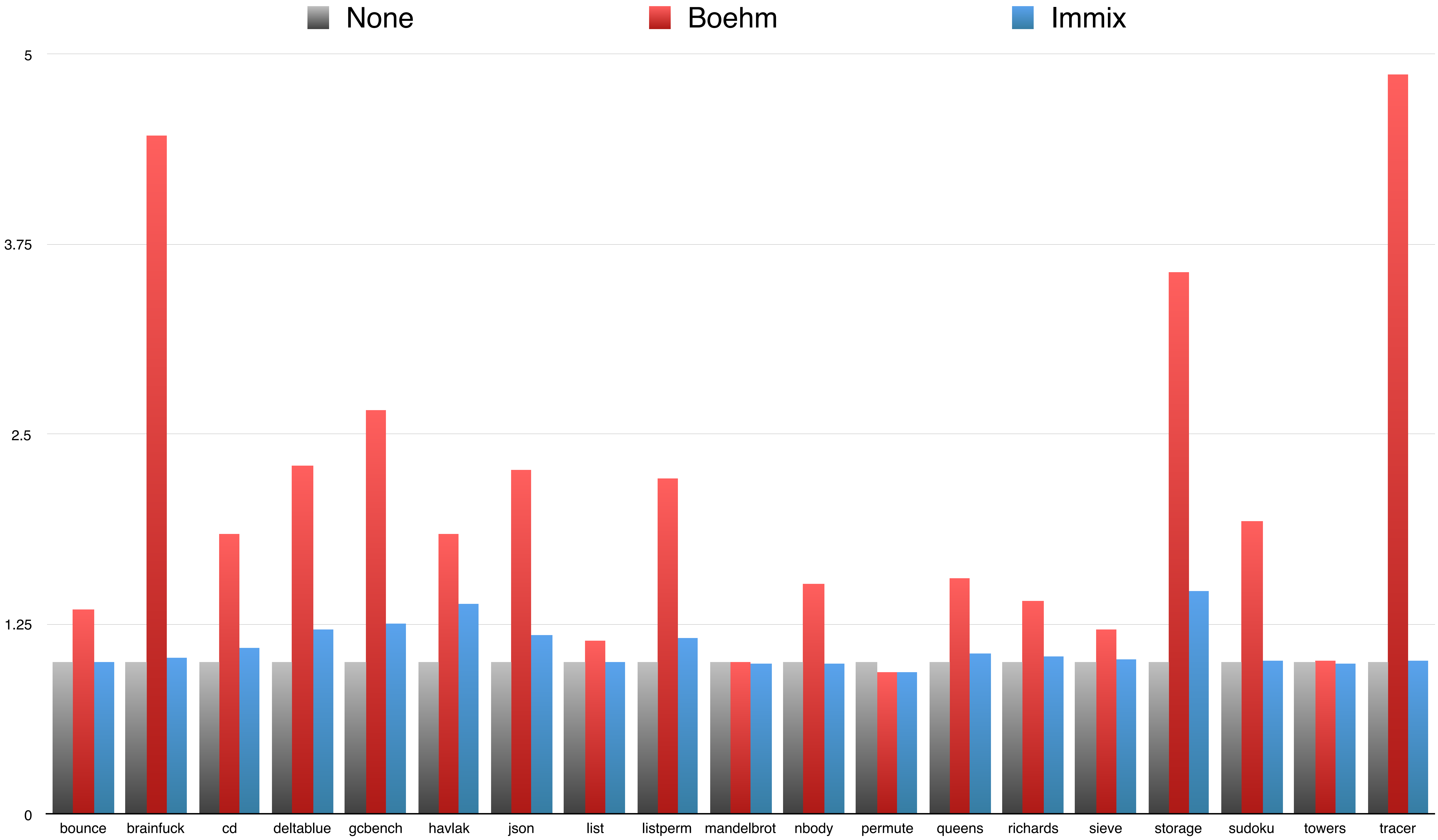


Performance evaluation and original implementation by Lukas Kellenberger.

Immix GC

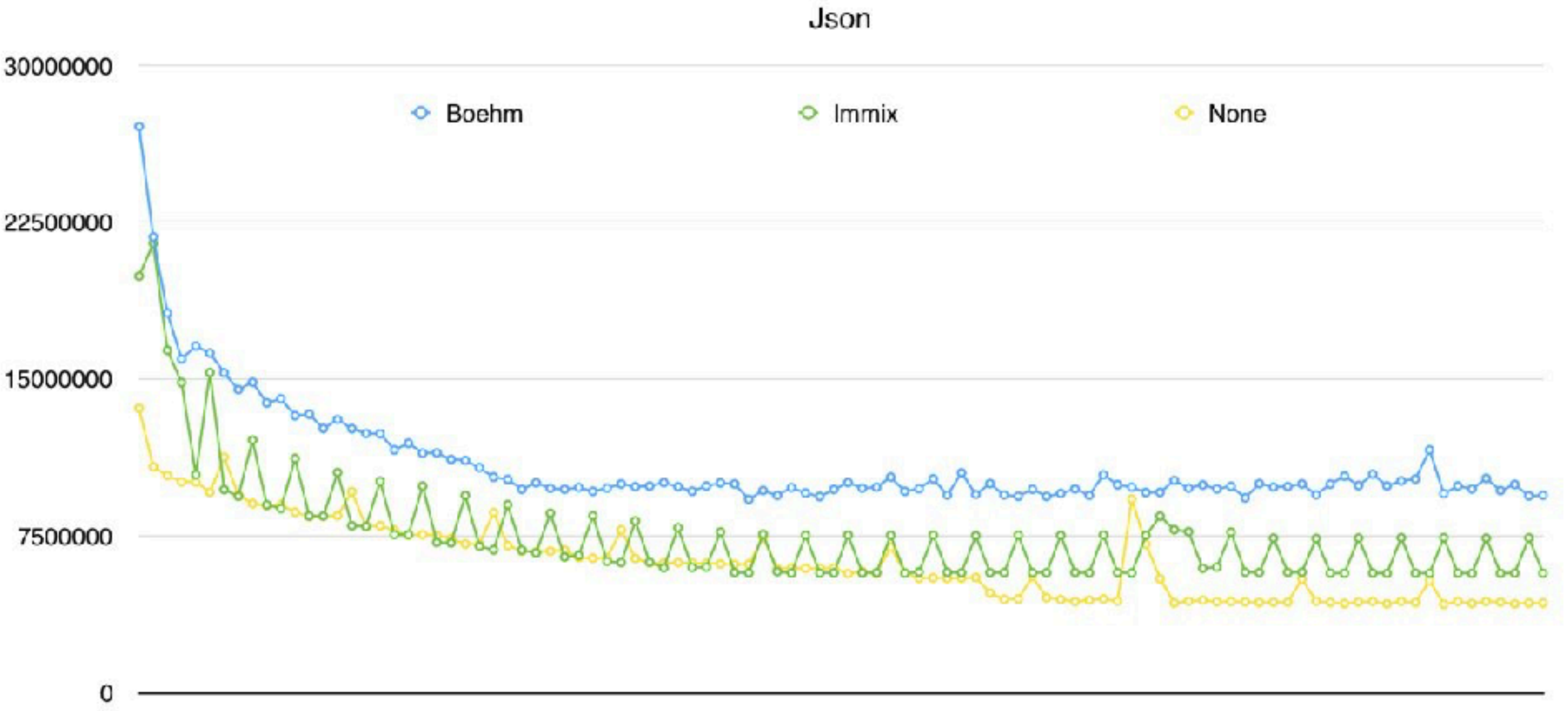
- Our artisanally crafted garbage collector, contributed by Lukas Kellenberger
- Precise heap, conservative stack
- Based on original work “*Immix: A Mark-Region Garbage Collector with Space Efficiency, Fast Collection, and Mutator Performance*” by Stephen M. Blackburn and Kathryn S. McKinley

Immix Throughput



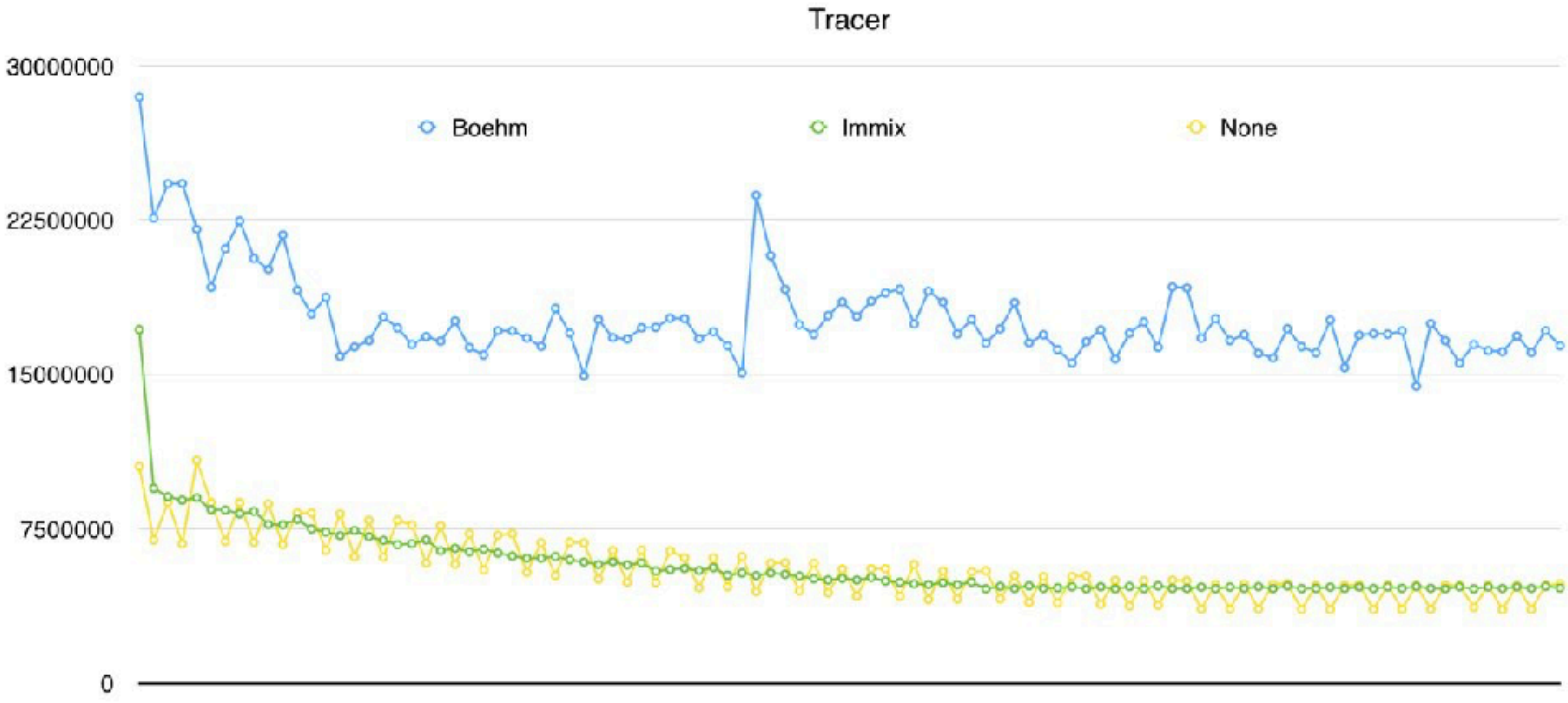
Performance evaluation and original implementation by Lukas Kellenberger.

Immix Predictability



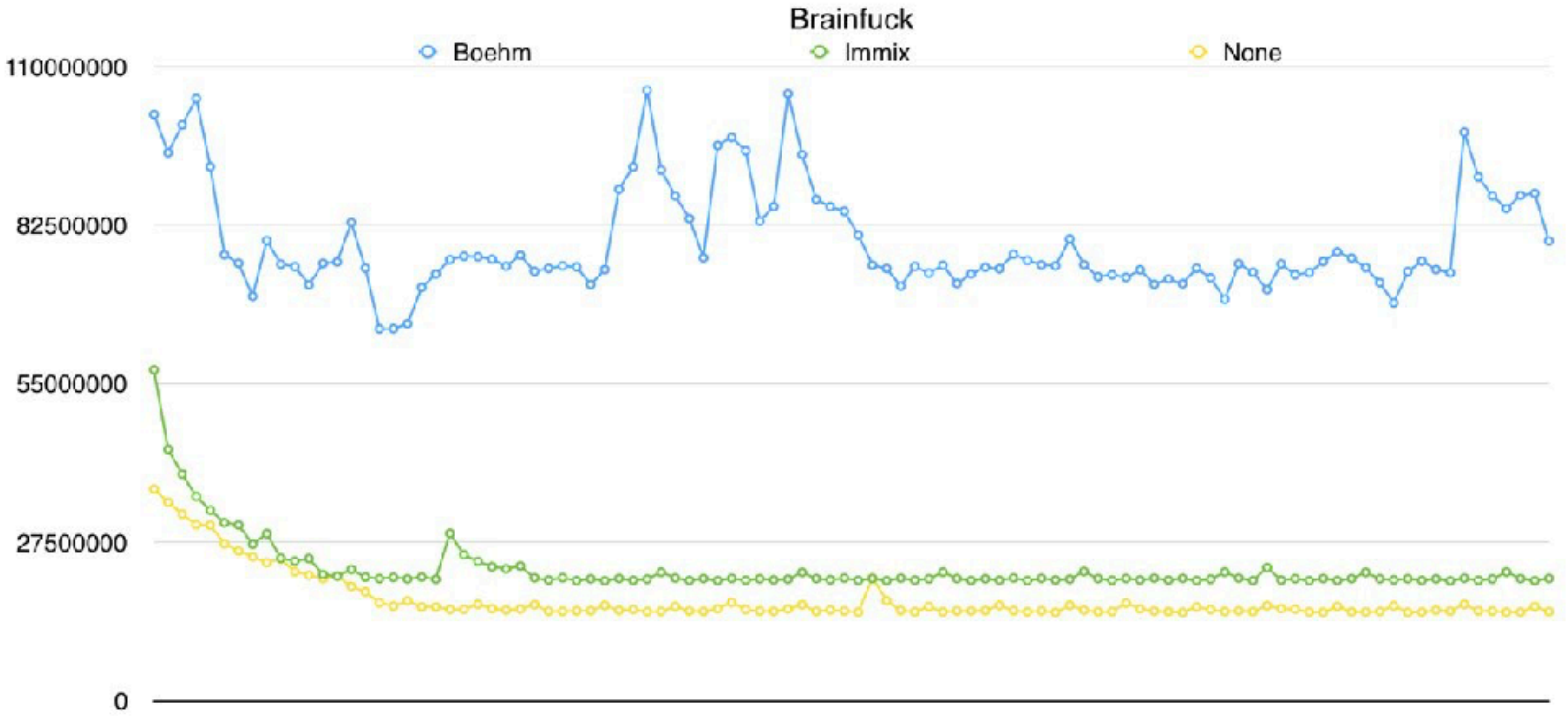
Performance evaluation and original implementation by Lukas Kellenberger.

Immix Predictability



Performance evaluation and original implementation by Lukas Kellenberger.

Immix Predictability



Performance evaluation and original implementation by Lukas Kellenberger.

Learn more

1. “Hands-on Scala-Native”
   by Guillaume Massé and Martin Duhemm

2. “Fast startup & low latency: pick two”
   by Denys Shabalin and Lukas Kellenberger

3. Official Website & Docs
   http://scala-native.org

4. Follow us on Twitter
   http://twitter.com/scala_native

Questions?