Graal: Not just a new JIT for the JVM.

Duncan MacGregor
Consulting Member of Technical Staff
Oracle Labs
March 4th 2019

Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a
commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle's products may change and remains at the sole discretion of Oracle Corporation.

The problem

Why do we need a new JIT

We'd like to write code like this

```
public int testMethod() {
   return Arrays.stream(myArray)
      .map(x -> x + 1)
      .map(x -> x + 2)
      .map(x -> x + 5)
      .reduce(0, Integer::sum);
}
```

It has lots of good features

- It’s easy to write
- It’s easily to compose stream functions together, or decompose this method
- It separates intent and strategy so it can easily change for large arrays, or more complex map operations

So why don’t we write code like this?

- Performance
- If you write a micro benchmark in this style and compare it to performing the operations using loops and arrays you’ll see a substantial performance difference
- BUT it’s much easier to make mistakes when writing the array code

So what's going on in this code?

- Arrays.stream creates a spliterator
  - And then creates a stream from that…
- The map operation creates a stream from that…
- Ditto for the next two map operations…
- Then finally reduce creates a terminal operation

What does a JIT need to do for this to be fast?

- Inlining
- Escape analysis
- Turn it into a simple loop

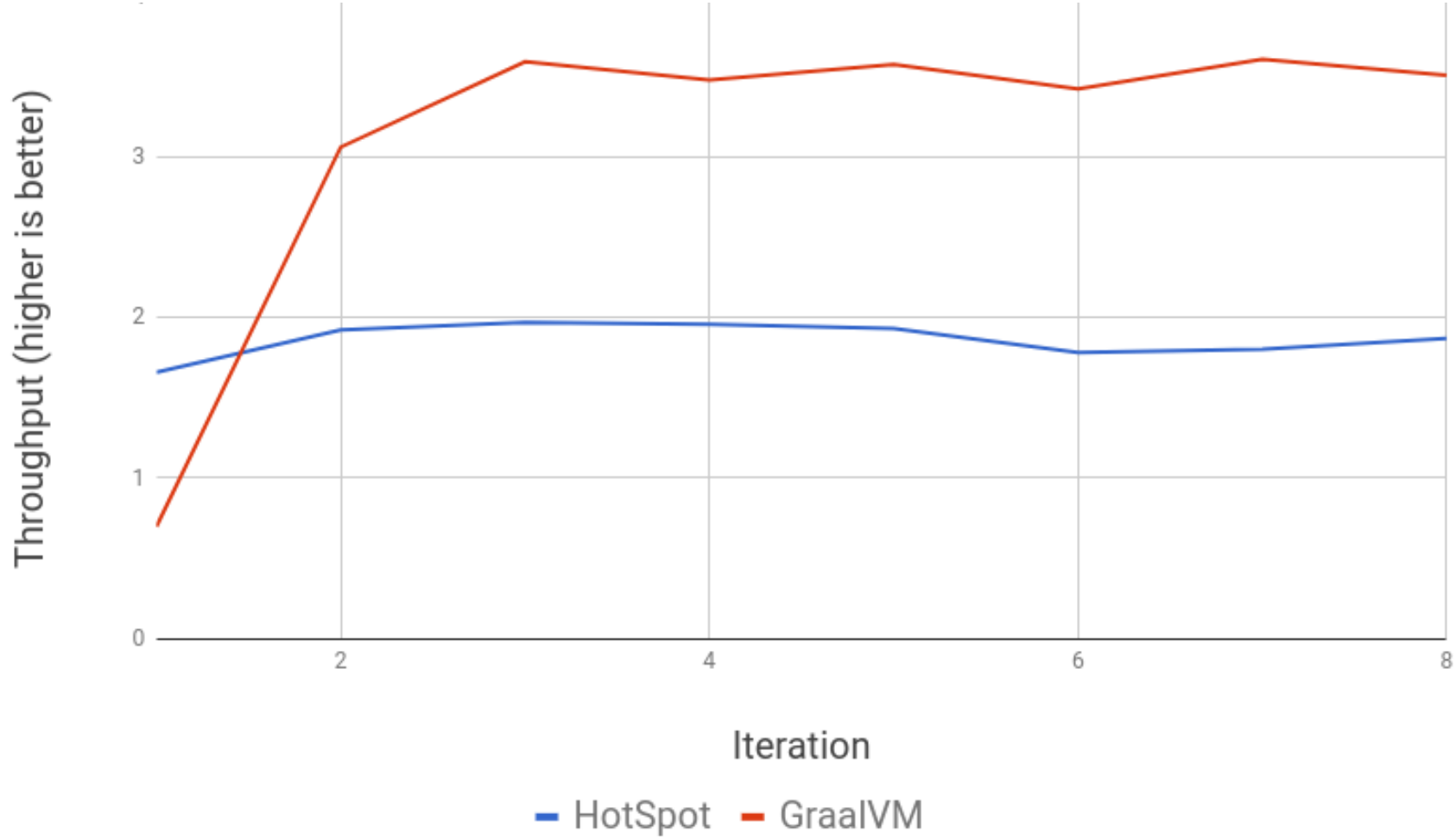
How well does C2 do at this?

- Inlining
  - It actually manages to inline a lot of this work
- Escape analysis
  - It still creates several of the intermediate objects

How well does Graal do at this?

- Inlining
  - Far more is inlined into the method which means…
- Escape analysis
  - Much more has been removed!
- It manages to extract a simple loop

What effect does that have?



https://medium.com/graalvm/stream-api-performance-with-graalvm-be6cfe7fbb52

So what's stopping us using this everywhere?

- Notice the time it took go get fast
- Graal is written in Java
  - We have to JIT our JIT
  - We use space on the heap
  - We may pollute type profiles in your code

You can try this at home

- Graal is already in OpenJDK 11
  - You can enable it with
    `-XX:+UnlockExperimentalVMOptions`
    `-XX:+UseJVMCICompiler`
    added to your java command line
  - You can also see how long it takes to compile itself by adding
    `-XX:+BootstrapJVCMI`

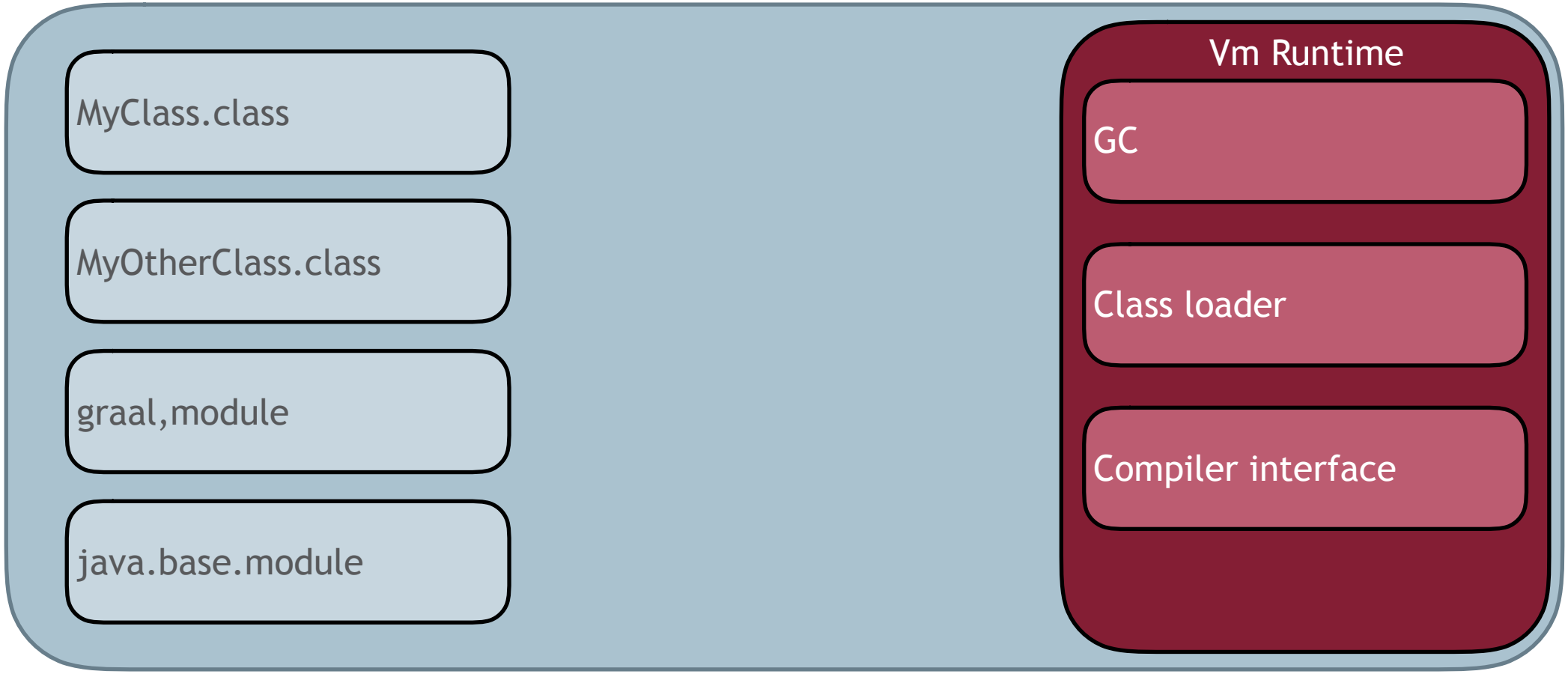
So now we have a new problem

How can we have Graal without the downsides

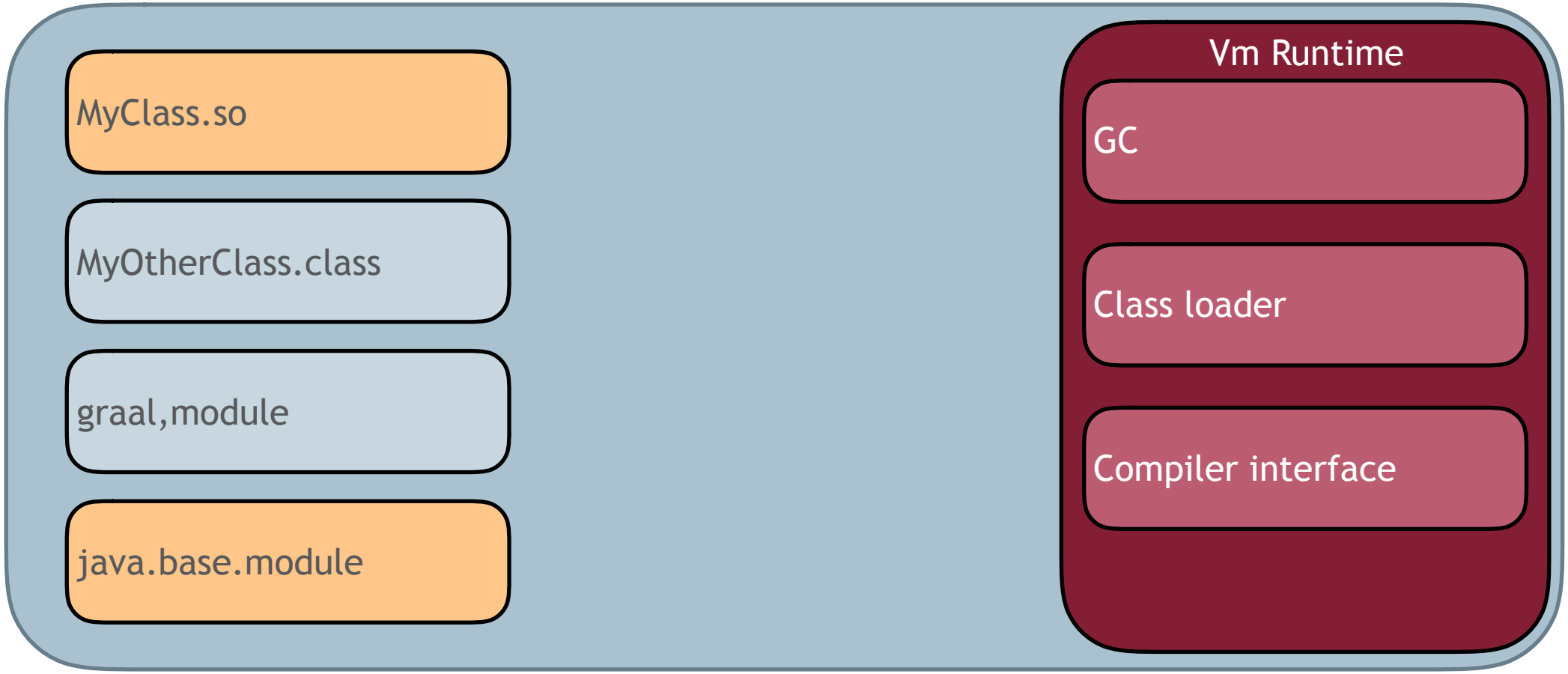
Graal isn’t just a JIT

- Many parts of a compiler are the same whether you do the compilation ahead of time or just in time
- You can do ahead of time compilation using **`jaotc`**
- But what does ahead of compilation mean for Java?

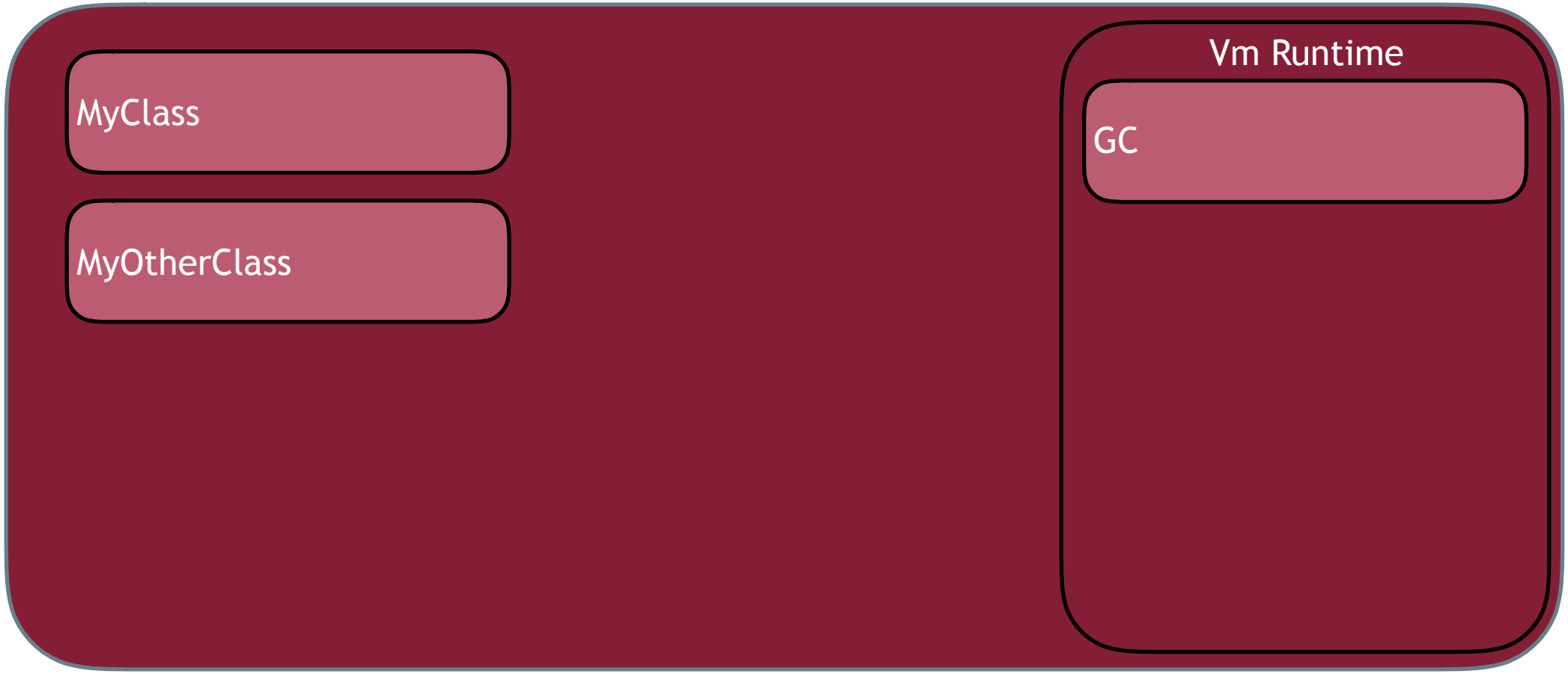
Just in time compilation means



Compiling a class to a shared library



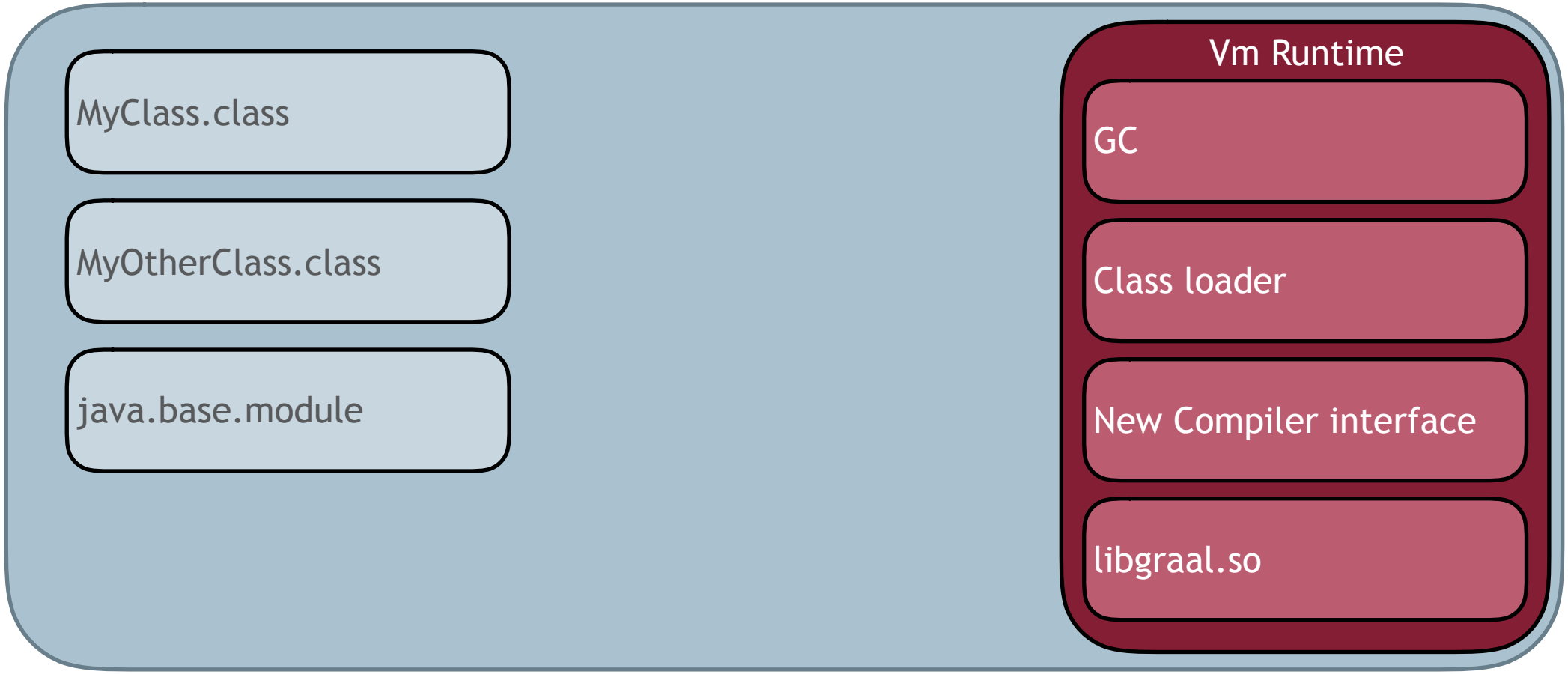
Building a standalone executable



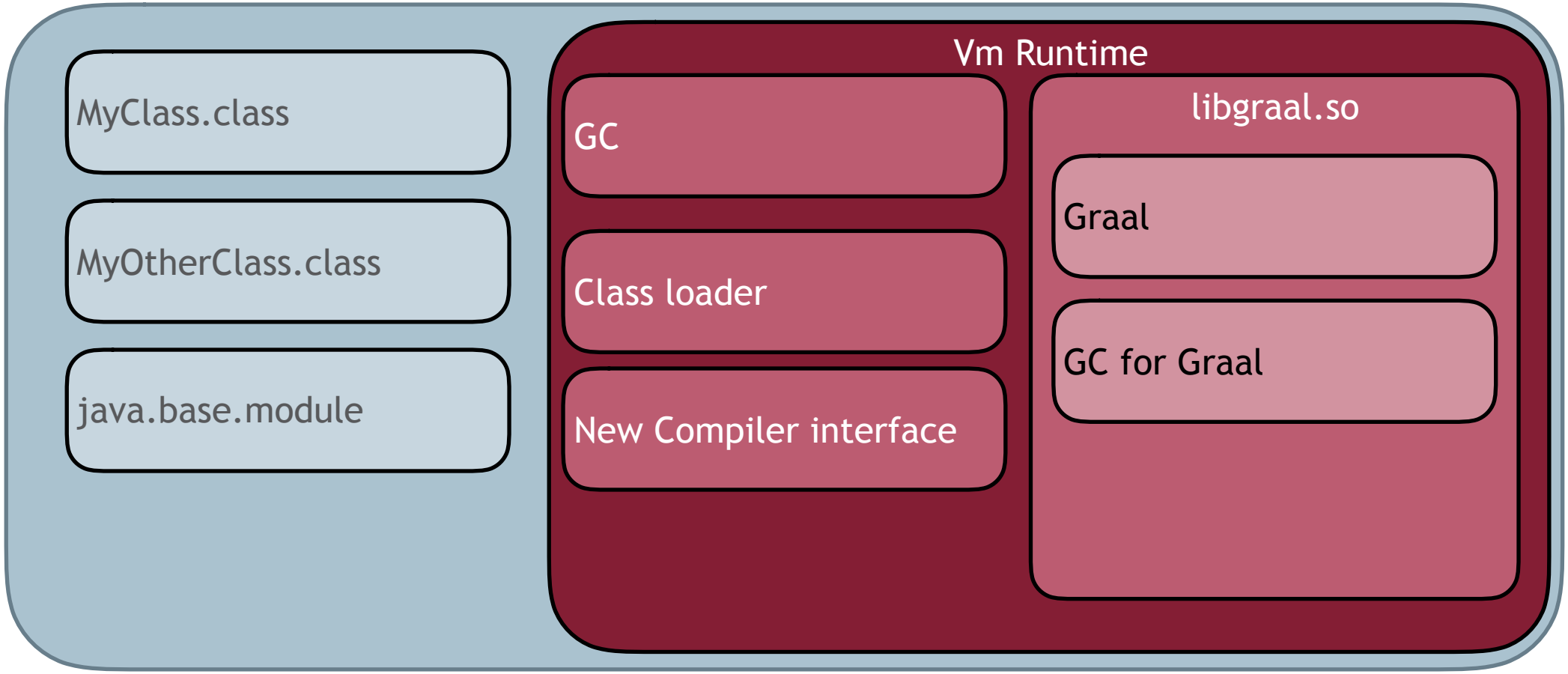
Does either of these really help with Graal?

- Compiling to a library
  - Avoids having to JIT the compiler
  - Still uses the heap
- Compiling to an a standalone executable
  - Doesn’t leave us with a JIT we can use in the JVM
- Is there a useful middle way?

Compiling to a self contained shared library



Compiling to a self contained shared library

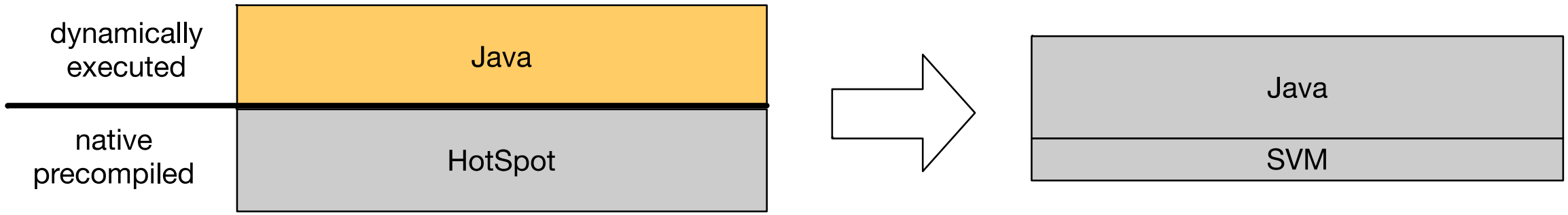


Third problem

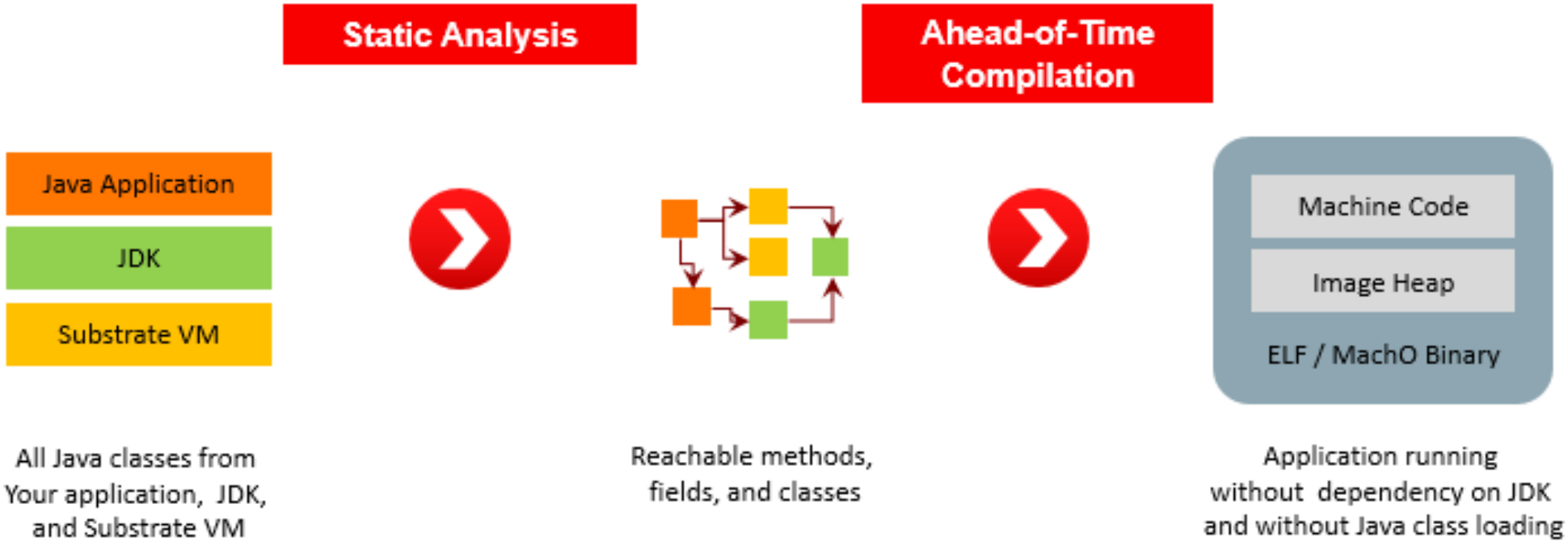
How do we turn a Java library into something we can use?

Native images

- Full AOT compilation to machine code
- Works with memory management
- Secure execution (e.g., bounds checks)
- **Embeddable with native applications**

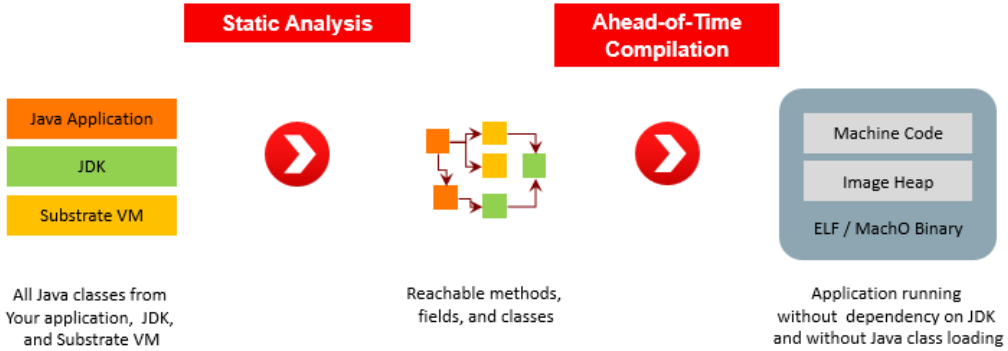


Native image generation



Native image lifecycle

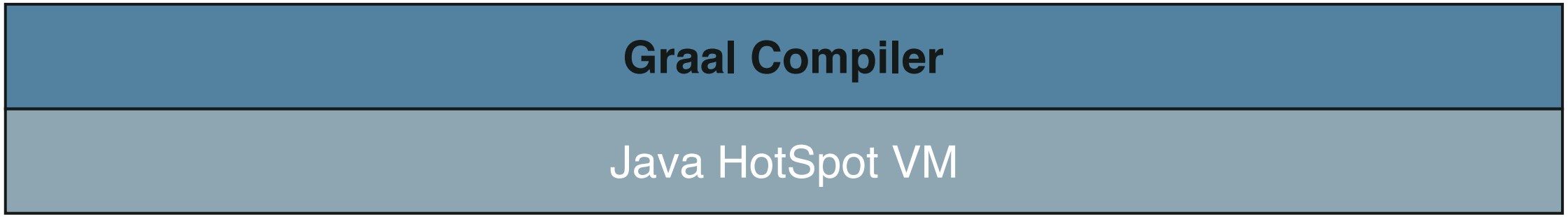
Image build time



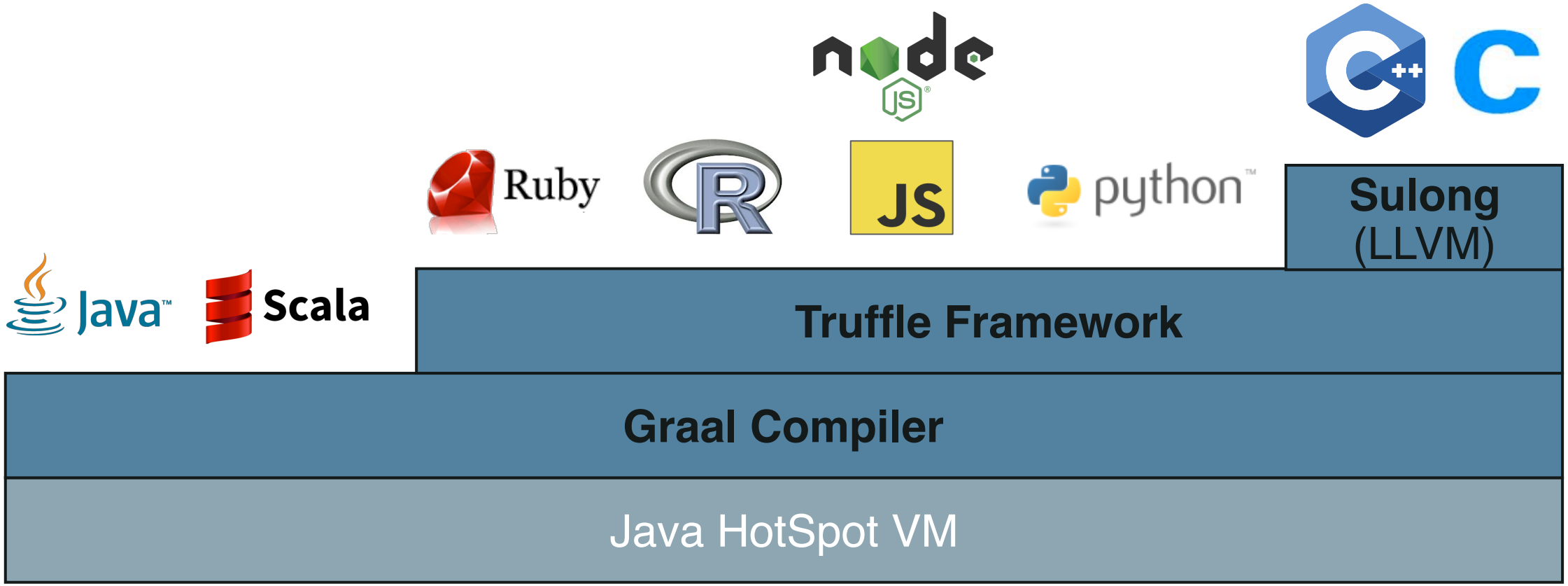
Runtime

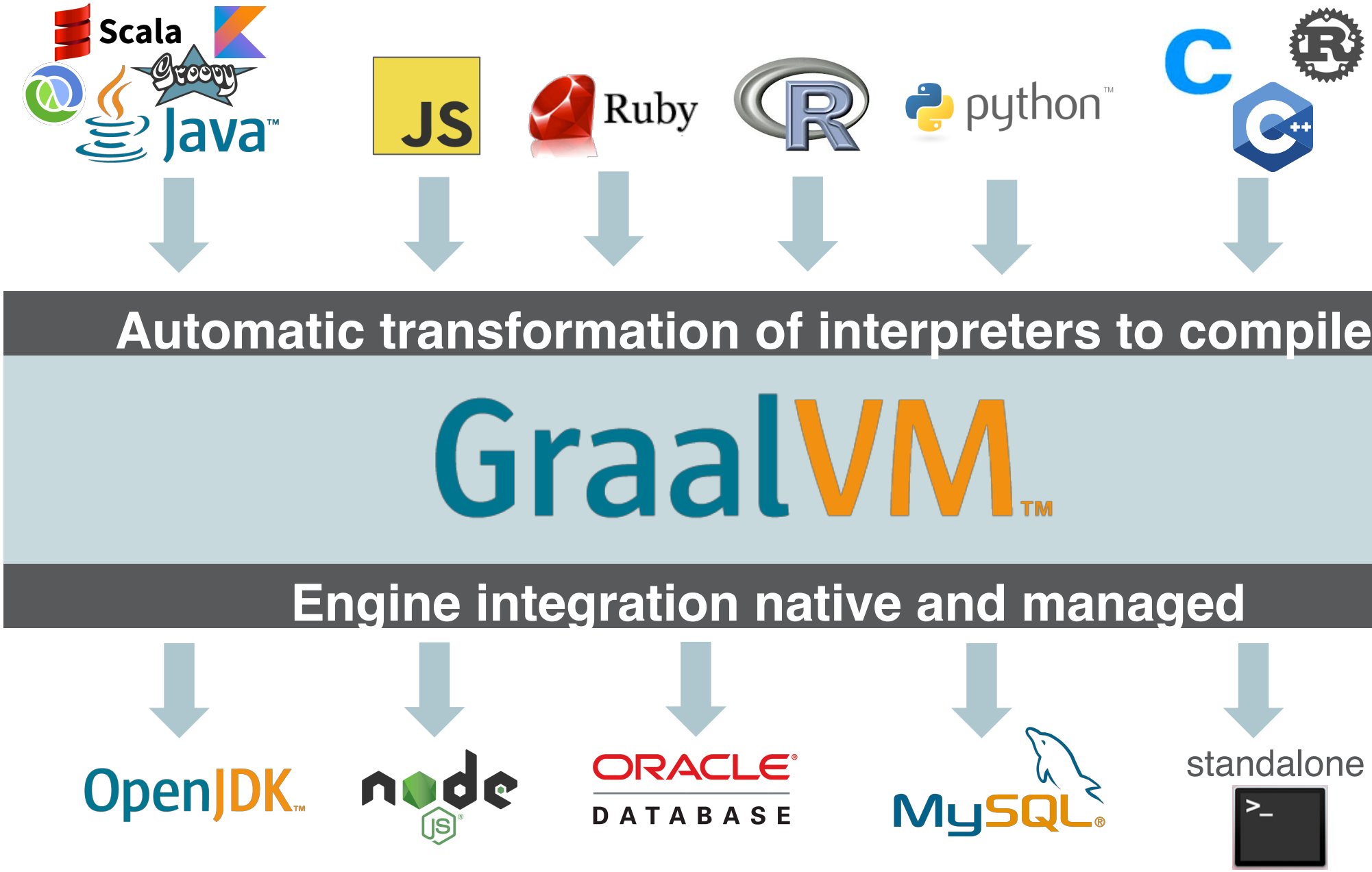
GraalVM™

Graal VM on the JVM Architecture



Graal VM on the JVM Architecture





oracle / graal

Unwatch 250 | Unstar 4,931 | Fork 265

Code | Issues 63 | Pull requests 5 | Insights

GraalVM: Run Programs Faster Anywhere 🚀 https://www.graalvm.org

polyglot vm java javascript python r ruby c

24,763 commits | **8** branches | **90** releases | **84** contributors

Branch: **master** | **New pull request** | **Create new file** | **Upload files** | **Find file** | **Clone or download**

cstancu [GR-10052] Reset lazily initialized cache fields of collection classes. ··· Latest commit `f85f8b4` an hour ago

ci_includes	Build Graph I/O API Javadoc explicitly	5 months ago
compiler	[GR-9933] Compilation fails with a Stackoverflow error.	7 hours ago
docs	Moved readme to the top-level directory	a month ago
examples	added Classpath Exception to mx files	19 days ago
regex	TRegex: removed some duplicated code from array buffer helper classes	7 days ago
sdk	Added ability configure caching per Source.	4 days ago
substratevm	Reset lazily initialized cache fields of collection classes.	an hour ago
tools	Make source hashCode deterministic again.	4 days ago

Community Edition (CE)

GraalVM CE is available for free for development and production use. It is built from the GraalVM sources available on GitHub. We provide pre-built binaries for GraalVM CE for Linux on x86 64-bit systems.

DOWNLOAD FROM GITHUB

Enterprise Edition (EE)

GraalVM EE provides additional performance, security, and scalability relevant for running critical applications in production. It is free for evaluation uses and available for download from the Oracle Technology Network. We provide binaries for GraalVM EE for Linux or Mac OS X on x86 64-bit systems.

DOWNLOAD FROM OTN

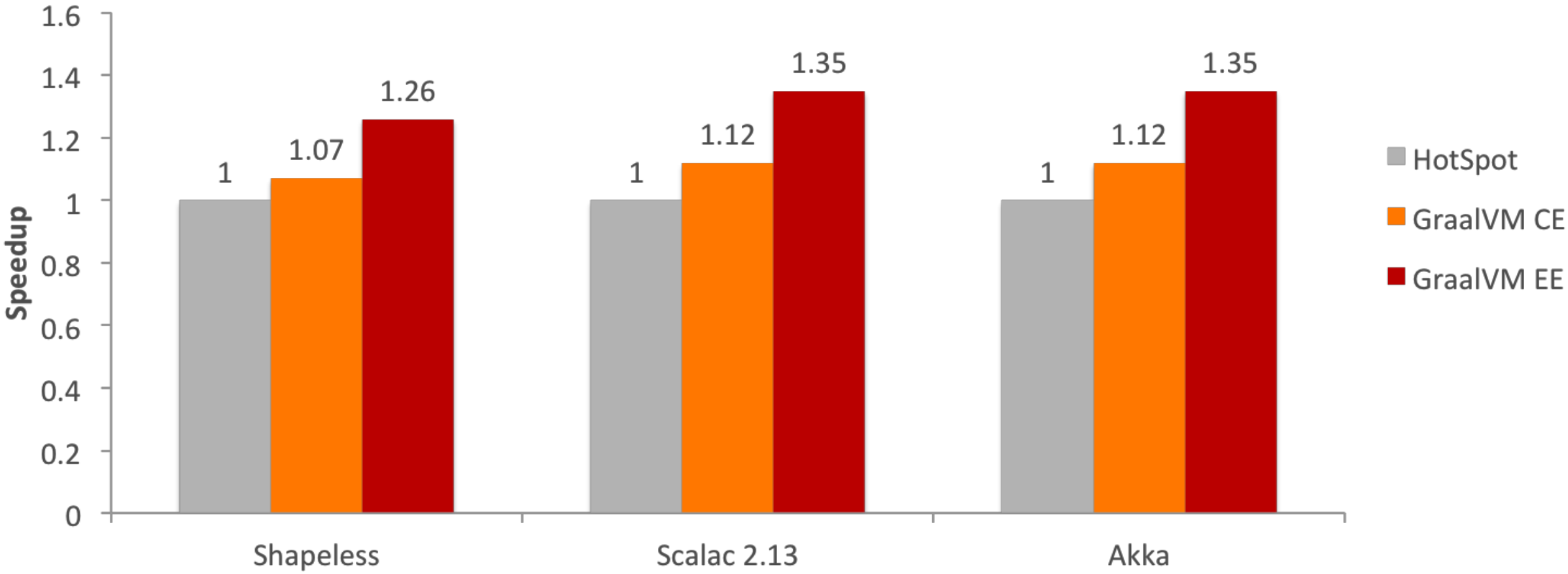
Practical Partial Evaluation for High-Performance Dynamic Language Runtimes

Thomas Würthinger* Christian Wimmer* Christian Humer* Andreas Wöß*
Lukas Stadler* Chris Seaton* Gilles Duboscq* Doug Simon* Matthias Grimmer†

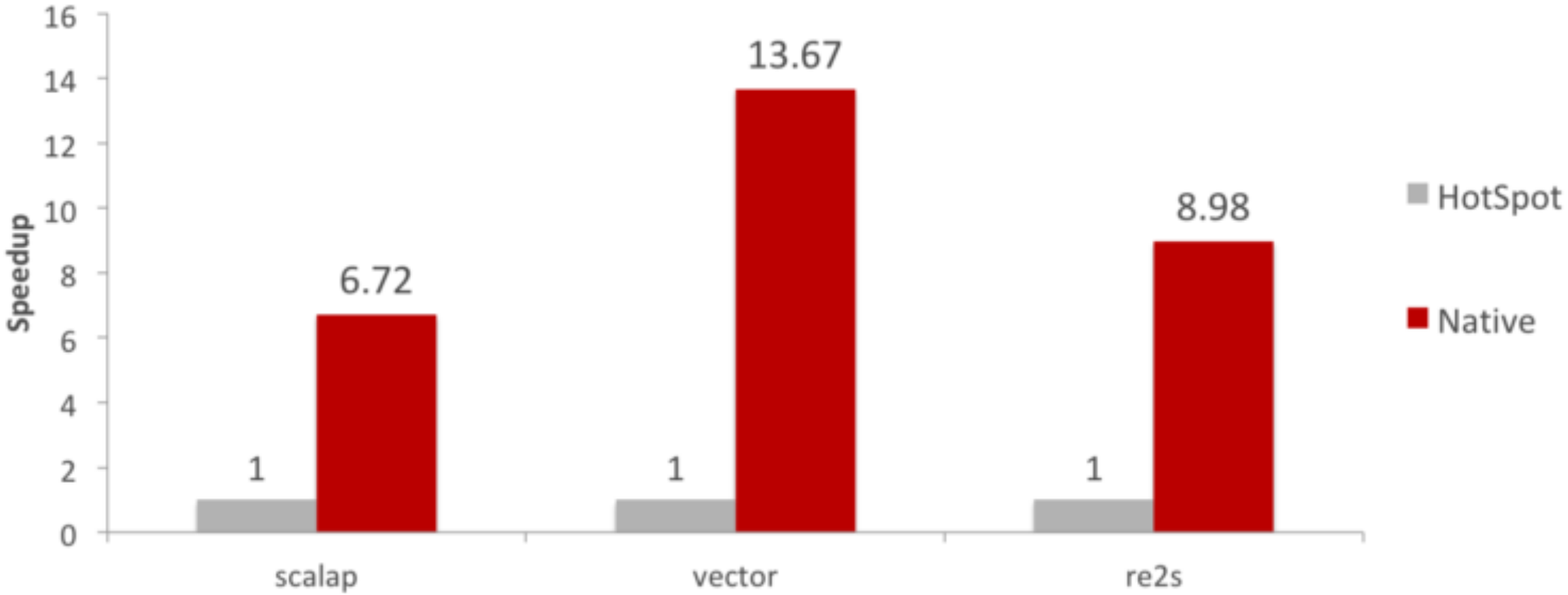
*Oracle Labs †Institute for System Software, Johannes Kepler University Linz, Austria
{thomas.wuerthinger, christian.wimmer, christian.humer, andreas.woess, lukas.stadler, chris.seaton,
gilles.m.duboscq, doug.simon}@oracle.com matthias.grimmer@jku.at

http://chrisseaton.com/rubytruffle/pldi17-truffle/pldi17-truffle.pdf

sbt > clean; compile;

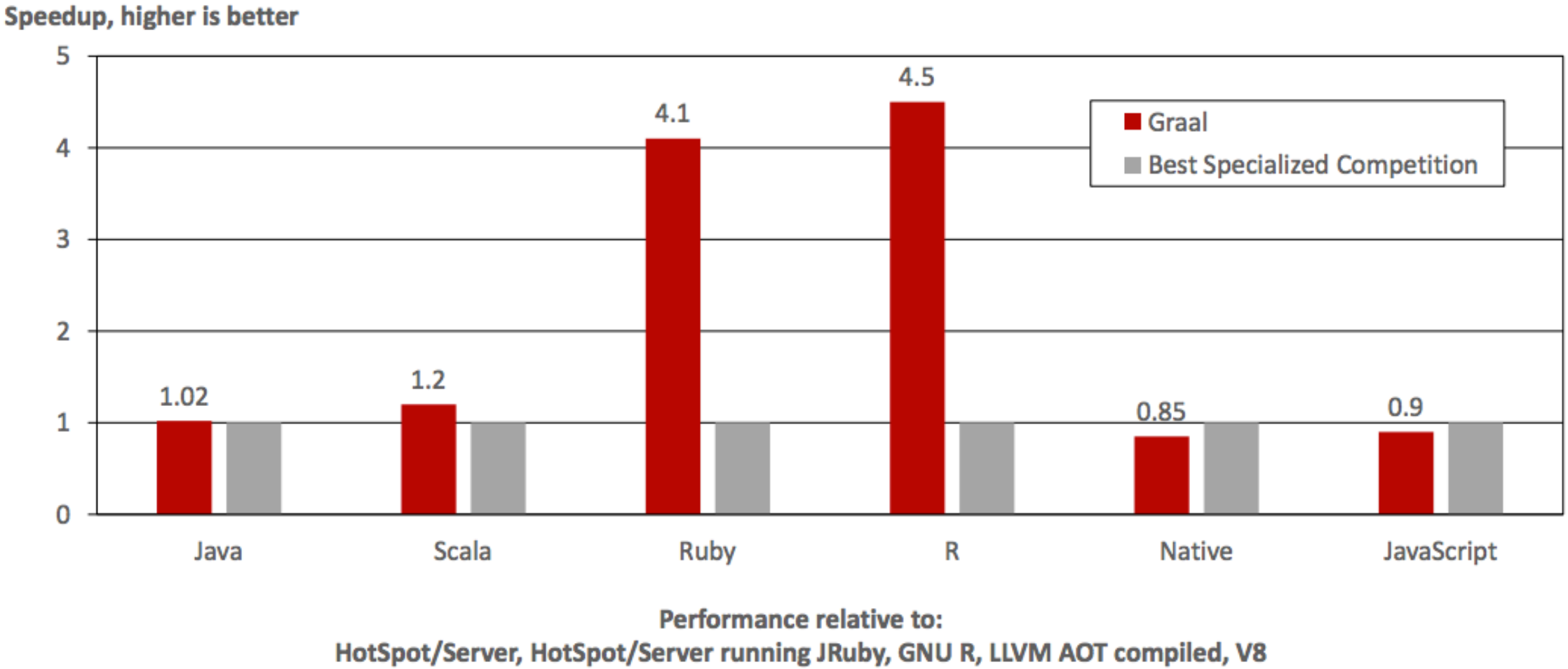


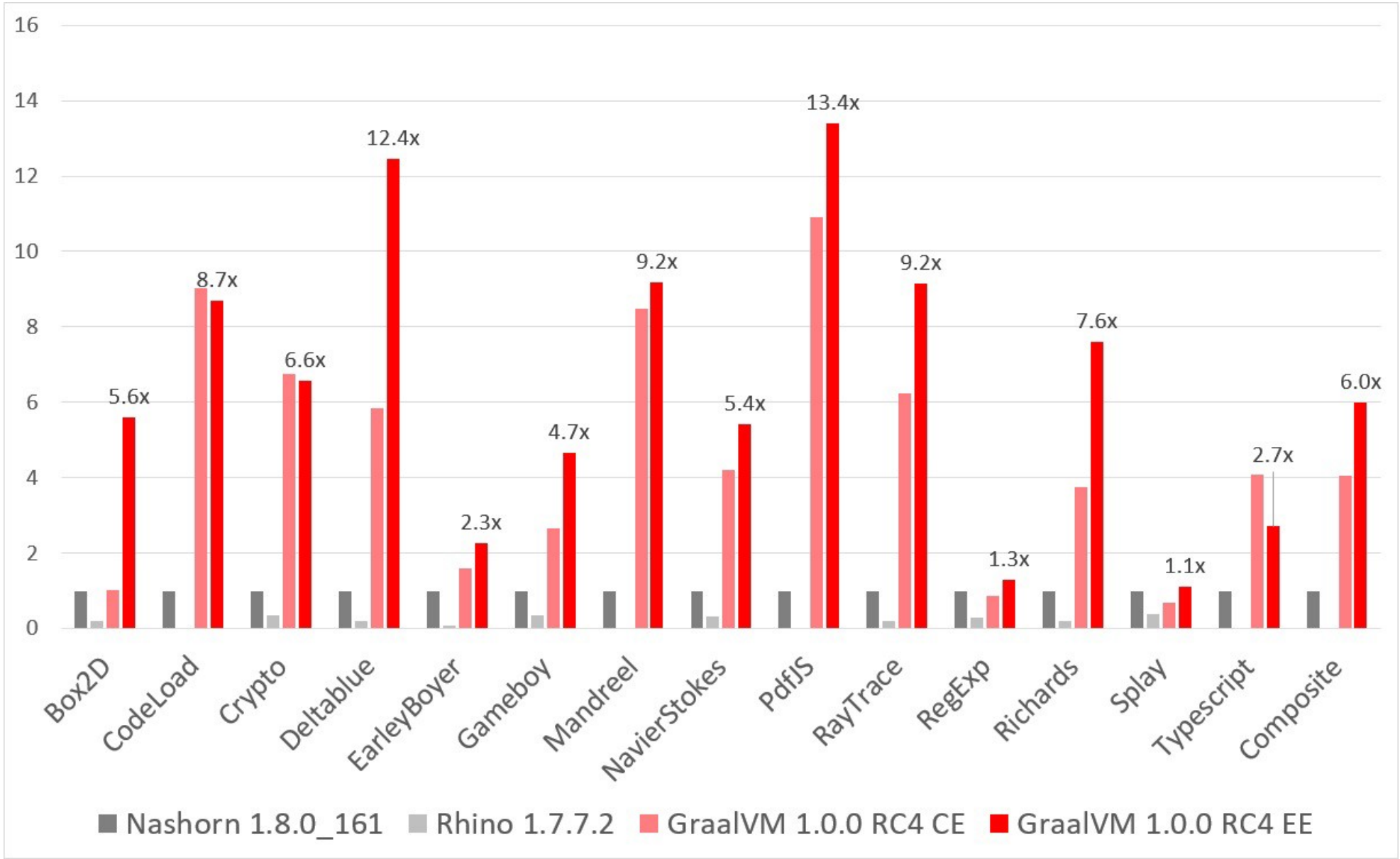
https://medium.com/graalvm/compiling-scala-faster-with-graalvm-86c5c0857fa3



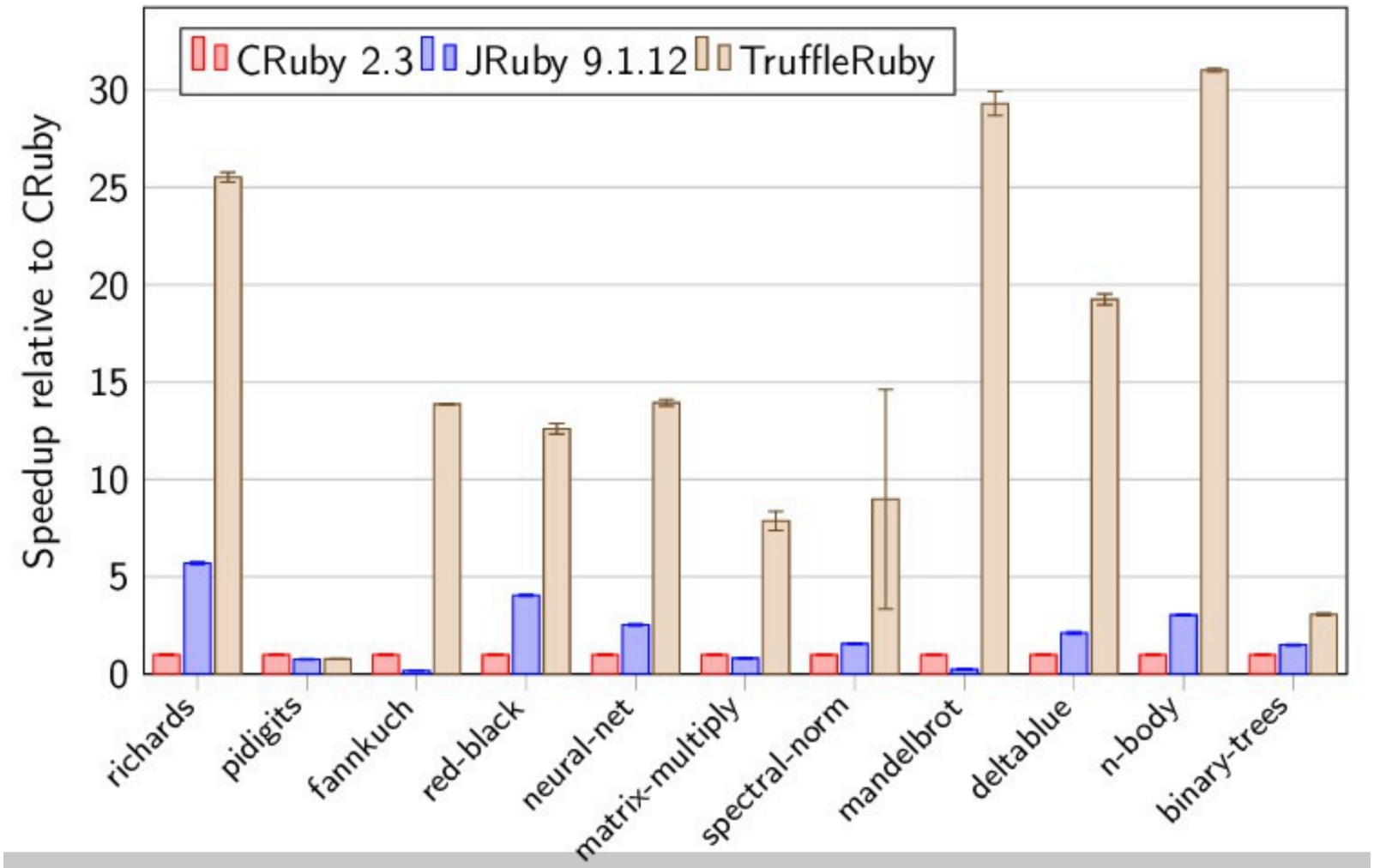
https://medium.com/graalvm/compiling-scala-faster-with-graalvm-86c5c0857fa3

Performance: Graal VM





The Computer Language Benchmarks Game



https://www.youtube.com/watch?v=mRKjWrNJ8DI

Try this out for yourselves

- Try using Graal in OpenJDK
  - Add
    `-XX:+UnlockExperimentalVMOptions`
    `-XX:+UseJVMCICompiler`
    to your java command line
- Try downloading GraalVM from graalvm.org
  - Look under graalvm.org/docs for Getting Started guides and examples
- Follow @graalvm on Twitter

Team

Oracle
Florian Angerer
Danilo Ansaloni
Stefan Anzinger
Martin Balin
Cosmin Basca
Daniele Bonetta
Dušan Bálek
Matthias Brantner
Lucas Braun
Petr Chalupa
Jürgen Christ
Laurent Daynès
Gilles Duboscq
Svatopluk Dědic
Martin Entlicher
Pit Fender
Francois Farquet
Brandon Fish
Matthias Grimmer
Christian Häubl
Peter Hofer
Bastian Hossbach
Christian Humer
Tomáš Hůrka
Mick Jordan

Oracle (continued)
Vojin Jovanovic
Anantha Kandukuri
Harshad Kasture
Cansu Kaynak
Peter Kessler
Duncan MacGregor
Jiří Maršík
Kevin Menard
Miloslav Metelka
Tomáš Myšík
Petr Pišl
Oleg Pliss
Jakub Podlešák
Aleksandar Prokopec
Tom Rodriguez
Roland Schatz
Benjamin Schlegel
Chris Seaton
Jiří Sedláček
Doug Simon
Štěpán Šindelář
Zbyněk Šlajchrt
Boris Spasojevic
Lukas Stadler
Codrut Stancu

Oracle (continued)
Jan Štola
Tomáš Stupka
Farhan Tauheed
Jaroslav Tulach
Alexander Ulrich
Michael Van De Vanter
Aleksandar Vitorovic
Christian Wimmer
Christian Wirth
Paul Wögerer
Mario Wolczko
Andreas Wöß
Thomas Würthinger
Tomáš Zezula
Yudi Zheng

Red Hat
Andrew Dinn
Andrew Haley

Intel
Michael Berg

Twitter
Chris Thalinger

Oracle Interns
Brian Belleville
Ondrej Douda
Juan Fumero
Miguel Garcia
Hugo Guiroux
Shams Imam
Berkin Ilbeyi
Hugo Kapp
Alexey Karyakin
Stephen Kell
Andreas Kunft
Volker Lanting
Gero Leinemann
Julian Lettner
Joe Nash
Tristan Overney
Aleksandar Pejovic
David Piorkowski
Philipp Riedmann
Gregor Richards
Robert Seilbeck
Rifat Shariyar

Oracle Alumni
Erik Eckstein
Michael Haupt
Christos Kotselidis
David Leibs
Adam Welc
Till Westmann

JKU Linz
Hanspeter Mössenböck
Benoit Daloze
Josef Eisl
Thomas Feichtinger
Josef Haider
Christian Huber
David Leopoldseder
Stefan Marr
Manuel Rigger
Stefan Rumzucker
Bernhard Urban

TU Berlin:
Volker Markl
Andreas Kunft
Jens Meiners
Tilmann Rabl

University of Edinburgh
Christophe Dubach
Juan José Fumero Alfonso
Ranjeet Singh
Toomas Remmelg

LaBRI
Floréal Morandat

University of California, Irvine
Michael Franz
Yeoul Na
Mohaned Qunaibit
Gulfem Savrun Yeniceri
Wei Zhang

Purdue University
Jan Vitek
Tomas Kalibera
Petr Maj
Lei Zhao

T. U. Dortmund
Peter Marwedel
Helena Kotthaus
Ingo Korb

University of California, Davis
Duncan Temple Lang
Nicholas Ulle

University of Lugano, Switzerland
Walter Binder
Sun Haiyang


ORACLE®