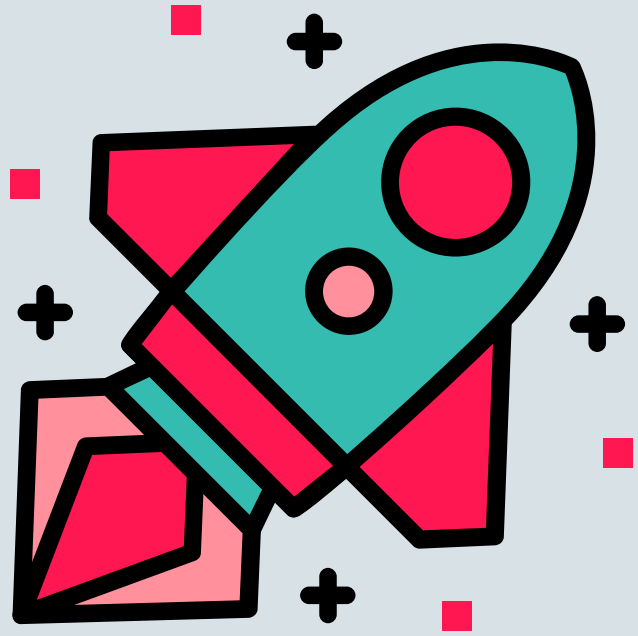

ORACLE®

An Introduction to TruffleRuby

High performance Ruby on GraalVM

Chris Seaton
Research Manager
Oracle Labs
March 2019

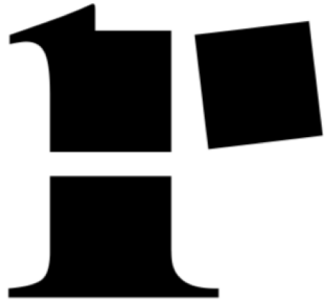


ORACLE

Safe Harbor Statement

The following is intended to provide some insight into a line of research in Oracle Labs. 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. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

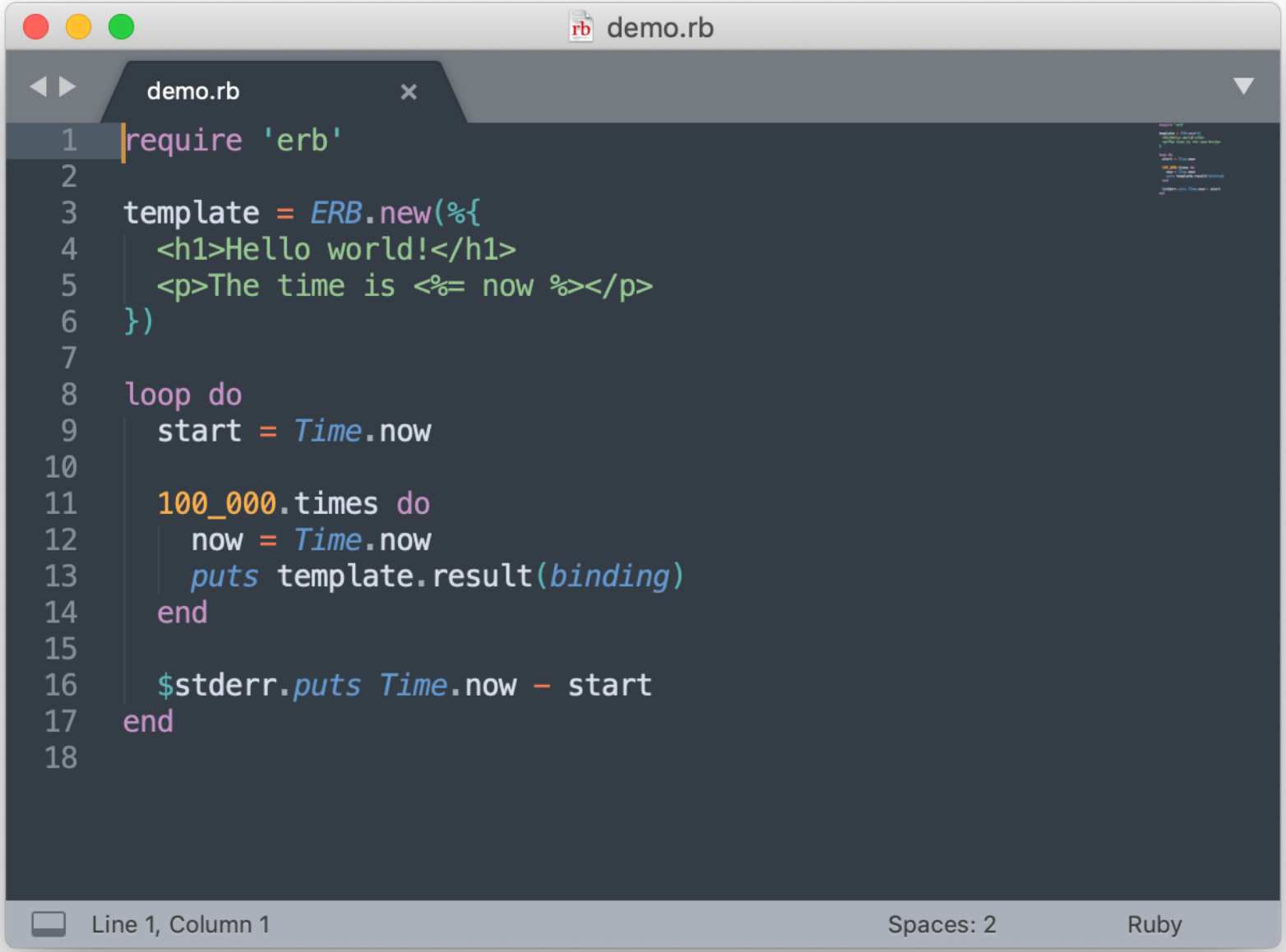
TruffleRuby basics



We wanted to build a Ruby that

- Runs idiomatic Ruby code faster
- Runs Ruby code in parallel
- Boots Ruby applications in less time
- Executes C extensions in a managed environment
- Adds fast and low-overhead interoperability with languages like Java, JavaScript, Python and R
- Provides new tooling such as debuggers and monitoring
- All while maintaining very high compatibility with the standard implementation of Ruby

```
$ rbenv install truffleruby-1.0.0-rc14
$ rbenv shell truffleruby-1.0.0-rc14
$ ruby -v
truffleruby 1.0.0-rc14, like ruby 2.6.2, GraalVM CE Native [x86_64-darwin]
```



TruffleRuby compared to MRI

Like MRI, TruffleRuby

- runs standard Ruby code
- uses RubyGems
- can be distributed as a binary tarball
- runs in Ruby version managers like rbenv, RVM, chruby
- doesn’t need a JVM to run
- starts instantly
- supports C extensions
- is open source

Compared to MRI, TruffleRuby has

- a concurrent, compacting GC
- an optimizing just-in-time native code compiler (a JIT)
- runs that run in parallel
- optimised data structures
- a graphical debugger
- more of the implementation in Ruby rather than C
- support for interop with Java, JavaScript, Python, R, and other languages

TruffleRuby compared to JRuby

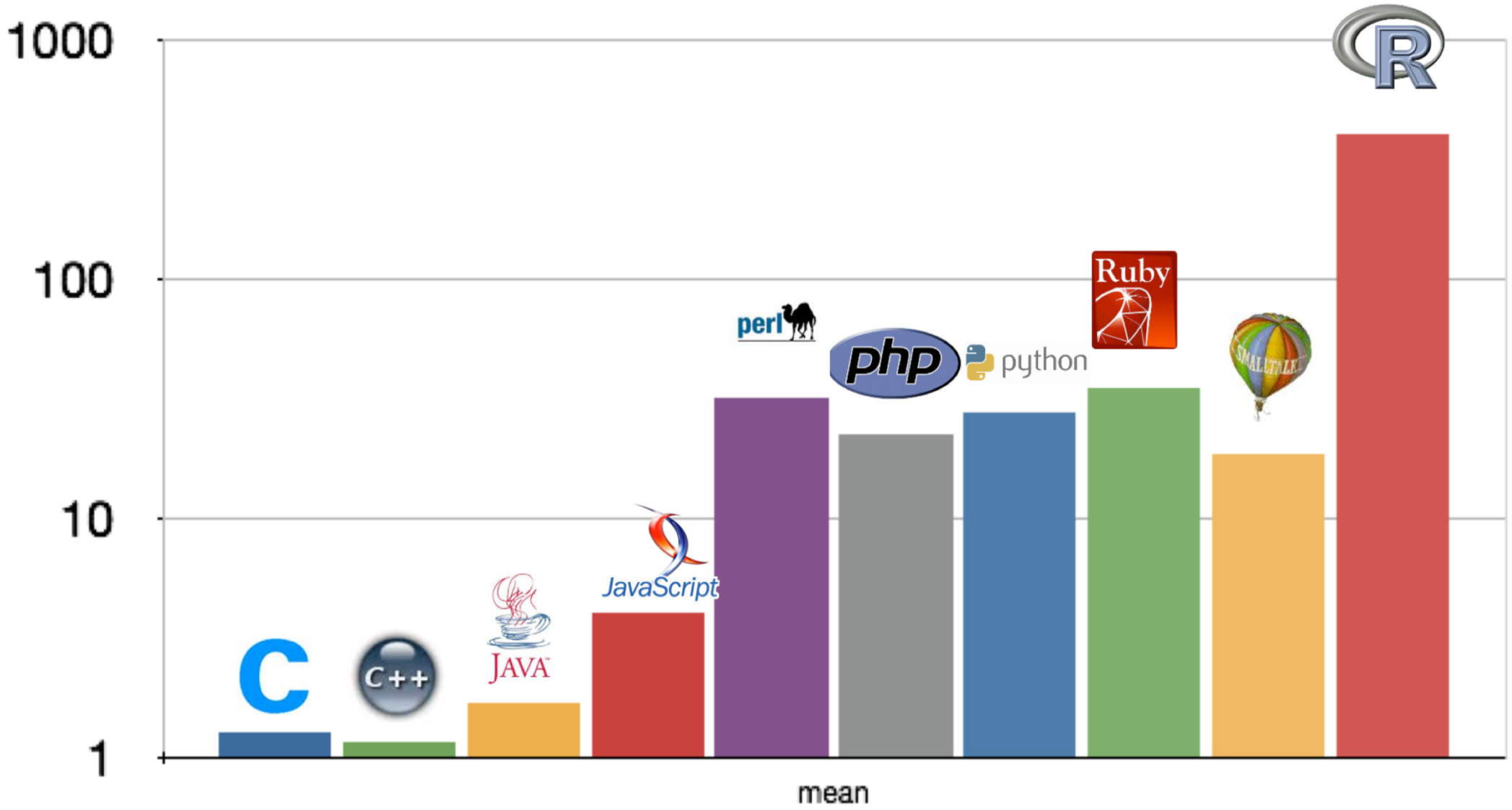
Like JRuby, TruffleRuby

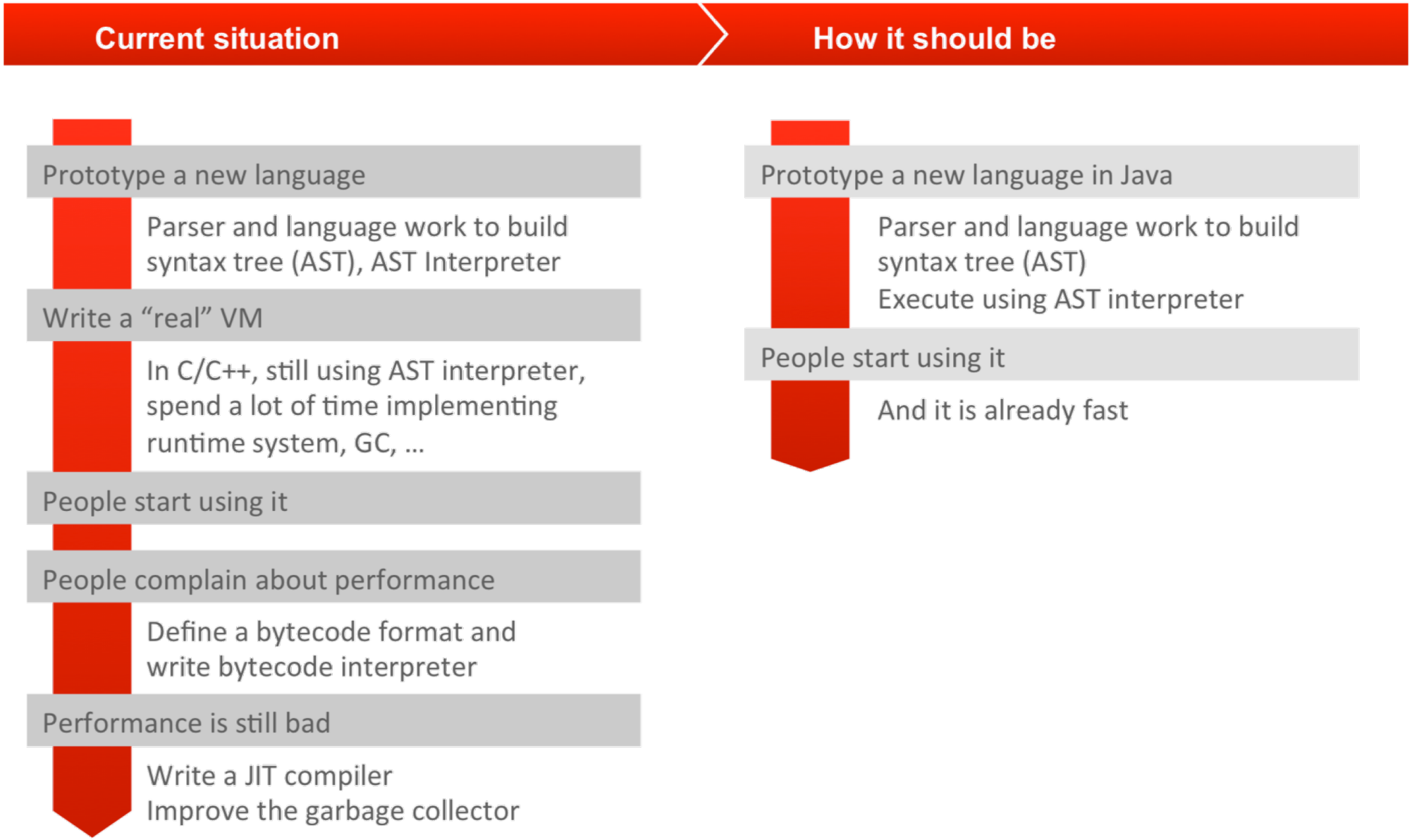
- can run on the JVM
- uses the GC and JIT from the JVM, but in a different way
- runs threads in parallel
- is partially implemented in Java
- can use Java libraries
- is open source

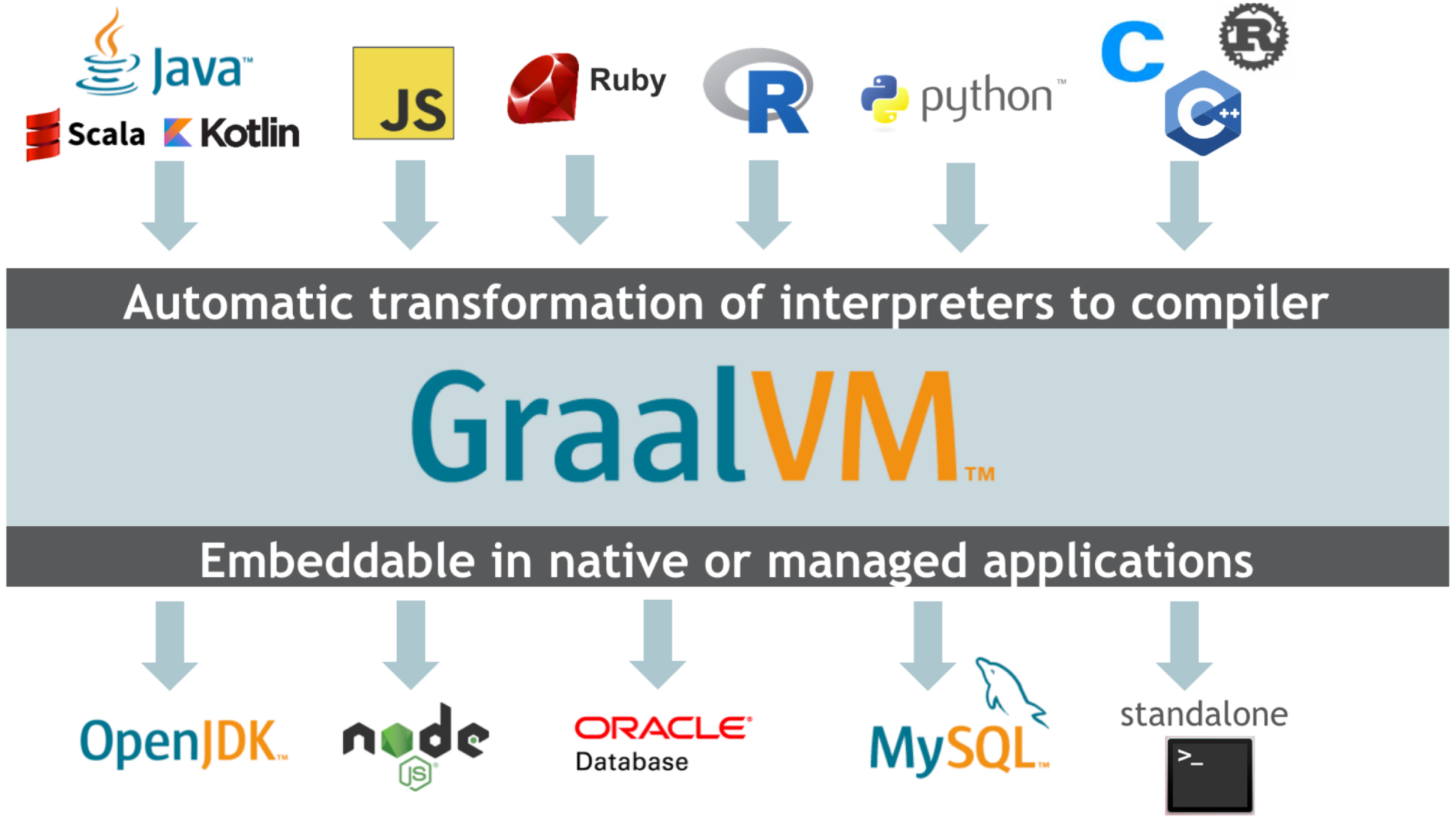
Compared to JRuby, TruffleRuby

- can run without the JVM
- can start instantly
- can be embedded in a native application
- has optimized data structures
- can run C extensions
- has a more powerful JIT
- has more of the implementation in Ruby rather than Java
- supports a standard Ruby stack

The big idea behind TruffleRuby





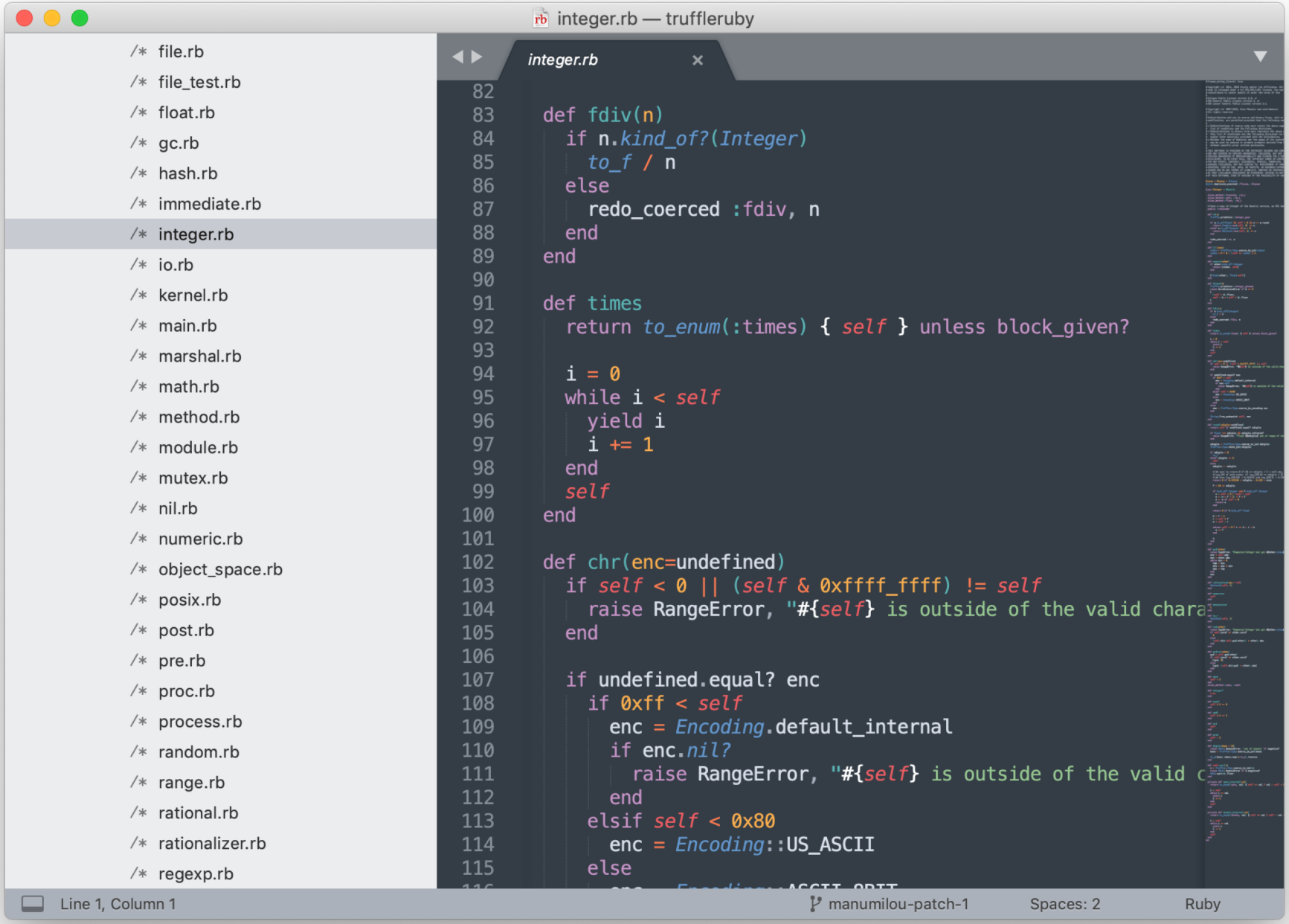


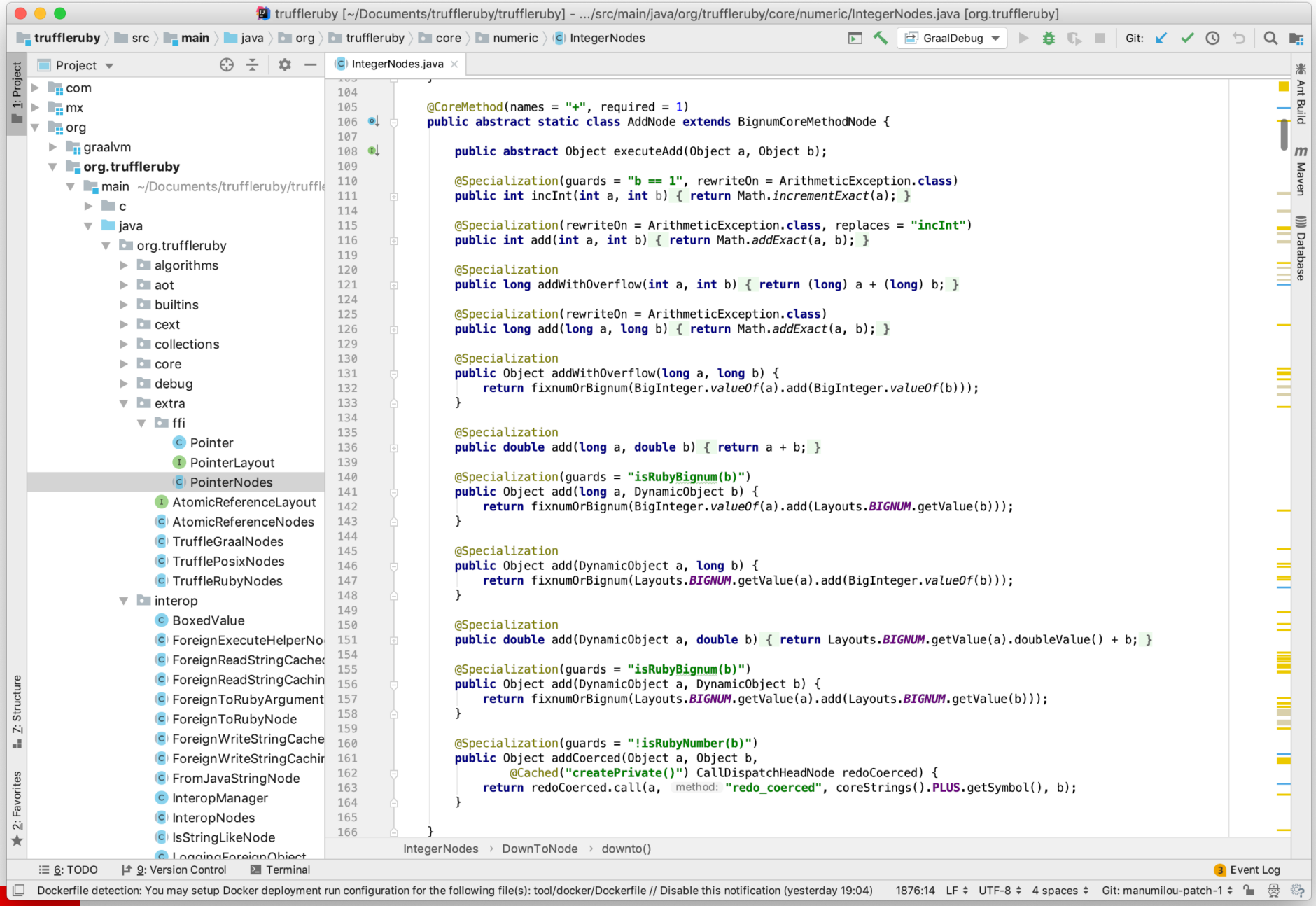
How TruffleRuby works

TruffleRuby is implemented with

- a core in Java providing basic functionality
- a layer on top of that implemented in Ruby for most functionality
- most of the standard library from MRI
- RubyGems, Bundler, etc unmodified

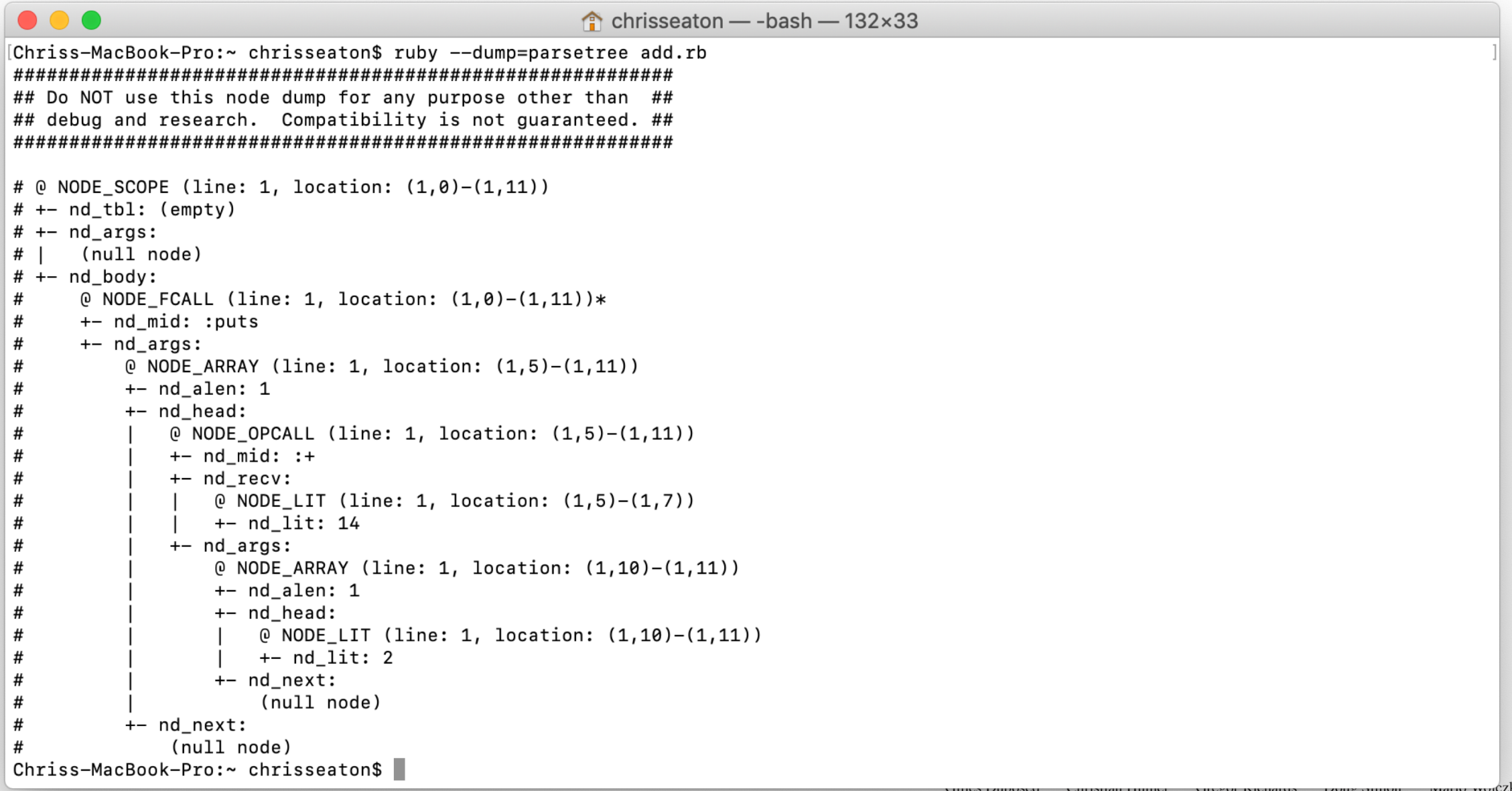
- All similar to the original idea of Rubinius





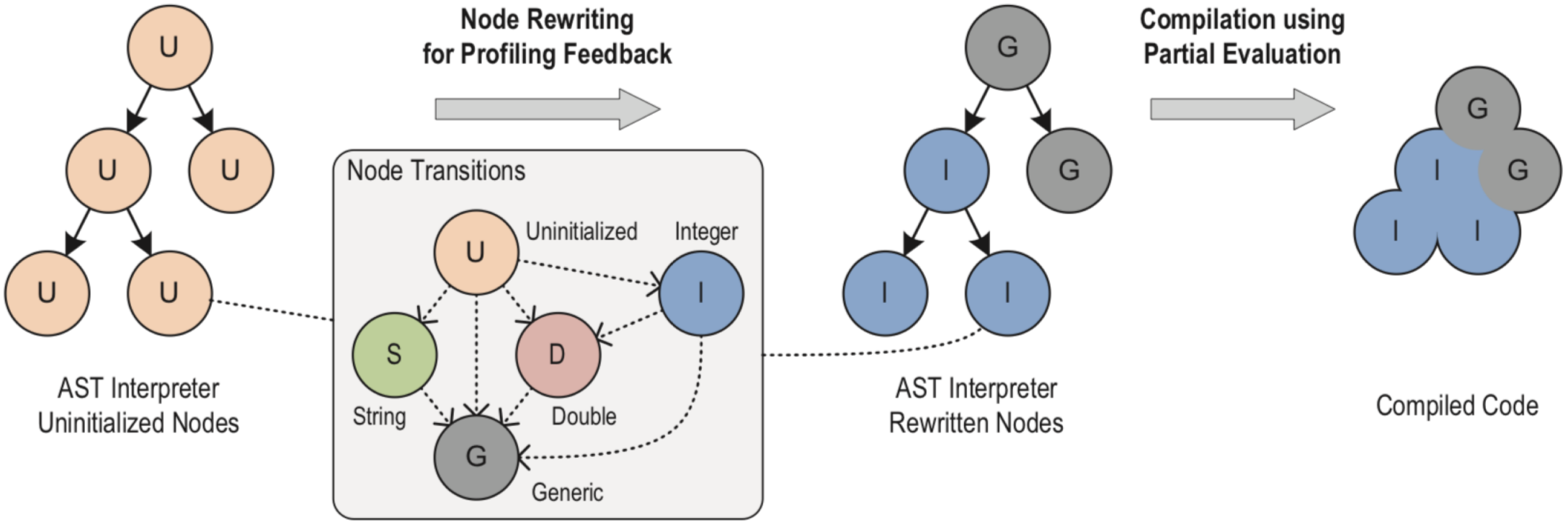
le Them All

Andreas Wöß† Lukas Stadler†
Gilles Duboscq† Christian Humer† Gregor Richards§ Doug Simon* Mario Wolczko*



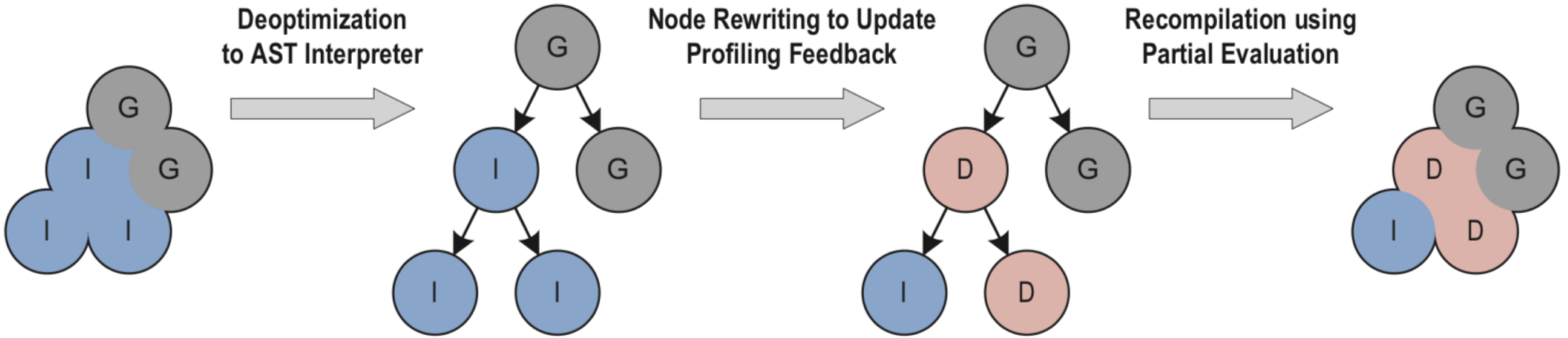
Giles Dubosq Christian Hulner Gregor Richards Doug Simon Mario Wolczko*

ORACLE



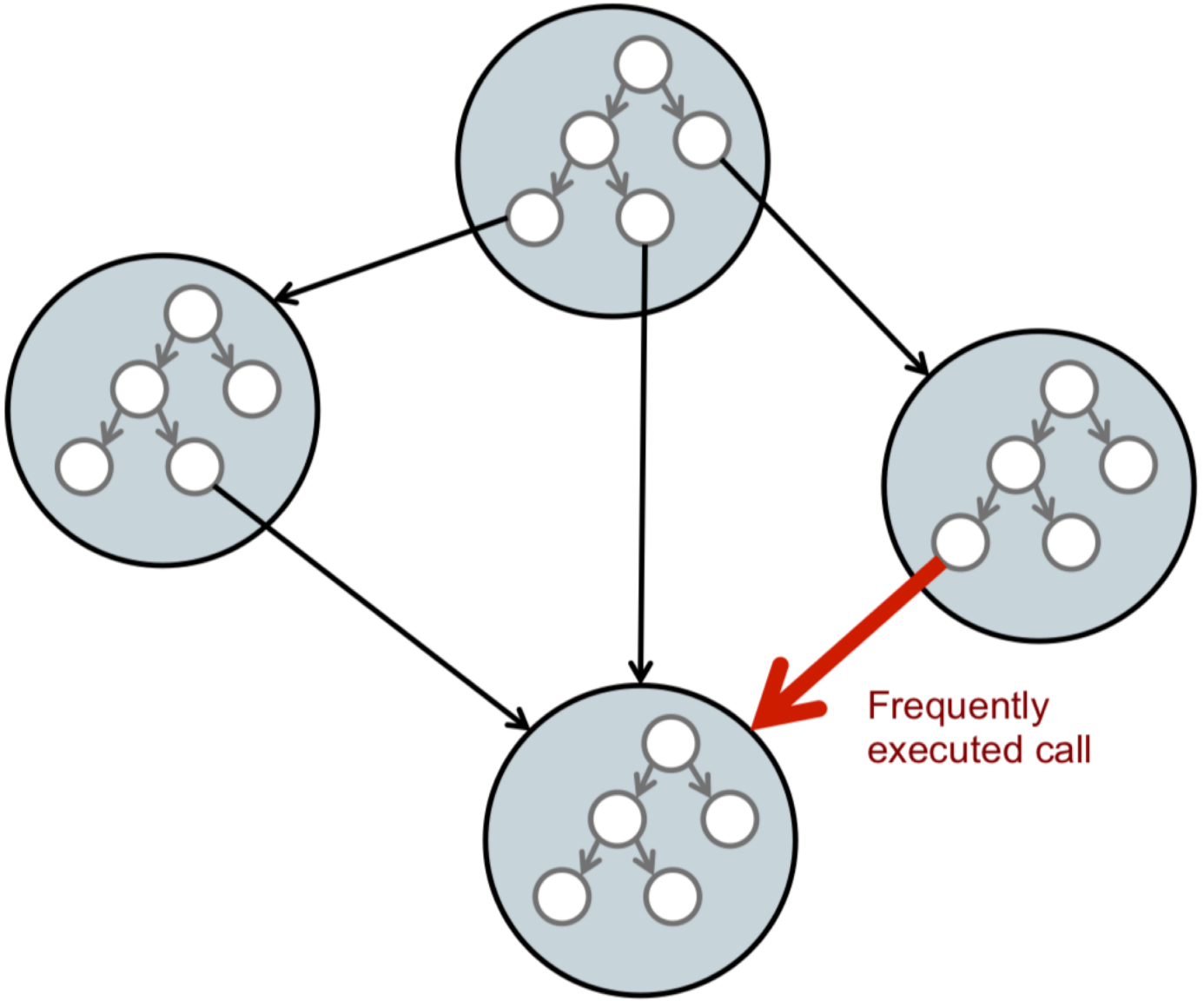
One VM to Rule Them All

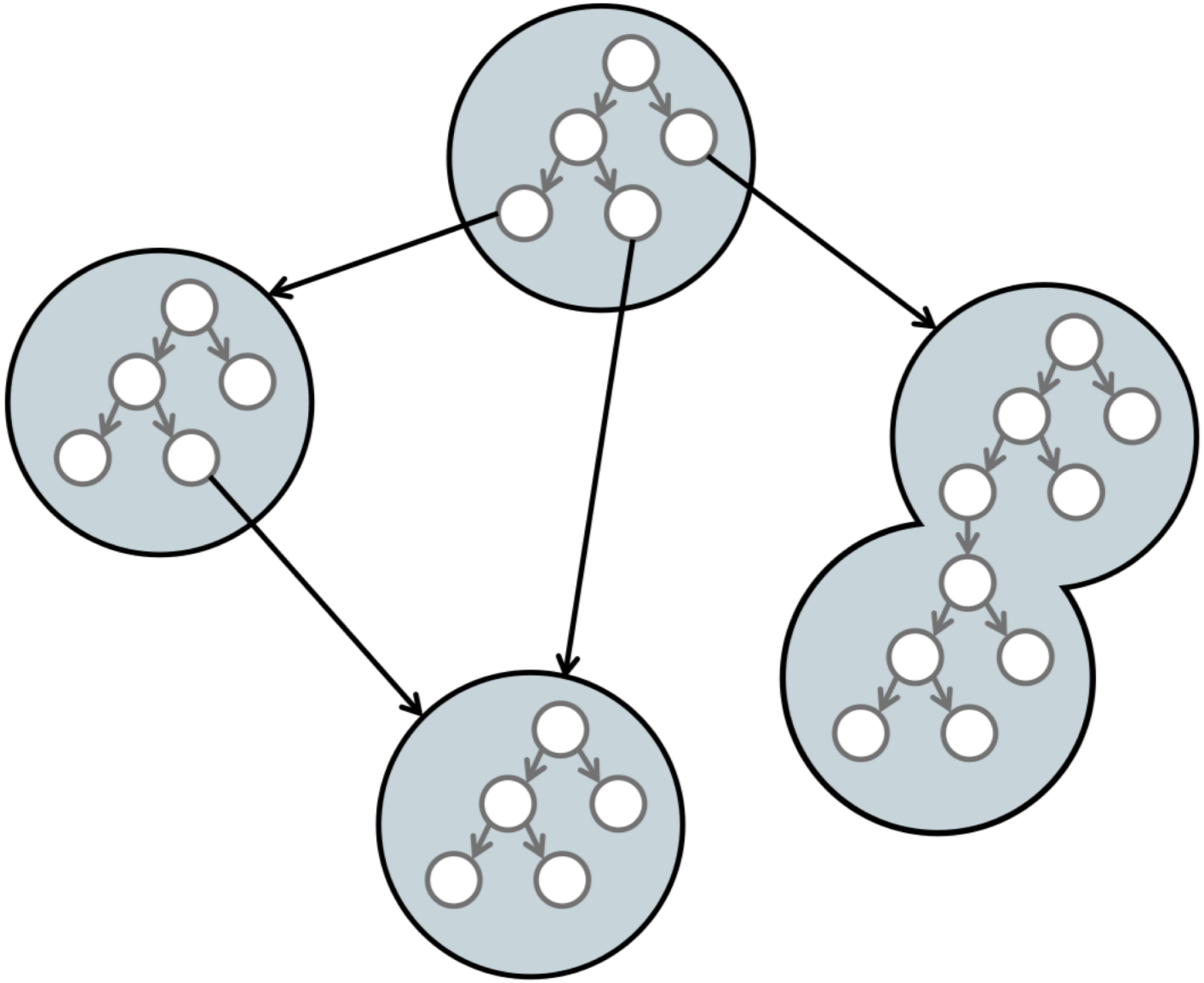
Thomas Würthinger* Christian Wimmer* Andreas Wöß† Lukas Stadler†
Gilles Duboscq† Christian Humer† Gregor Richards§ Doug Simon* Mario Wolczko*



One VM to Rule Them All

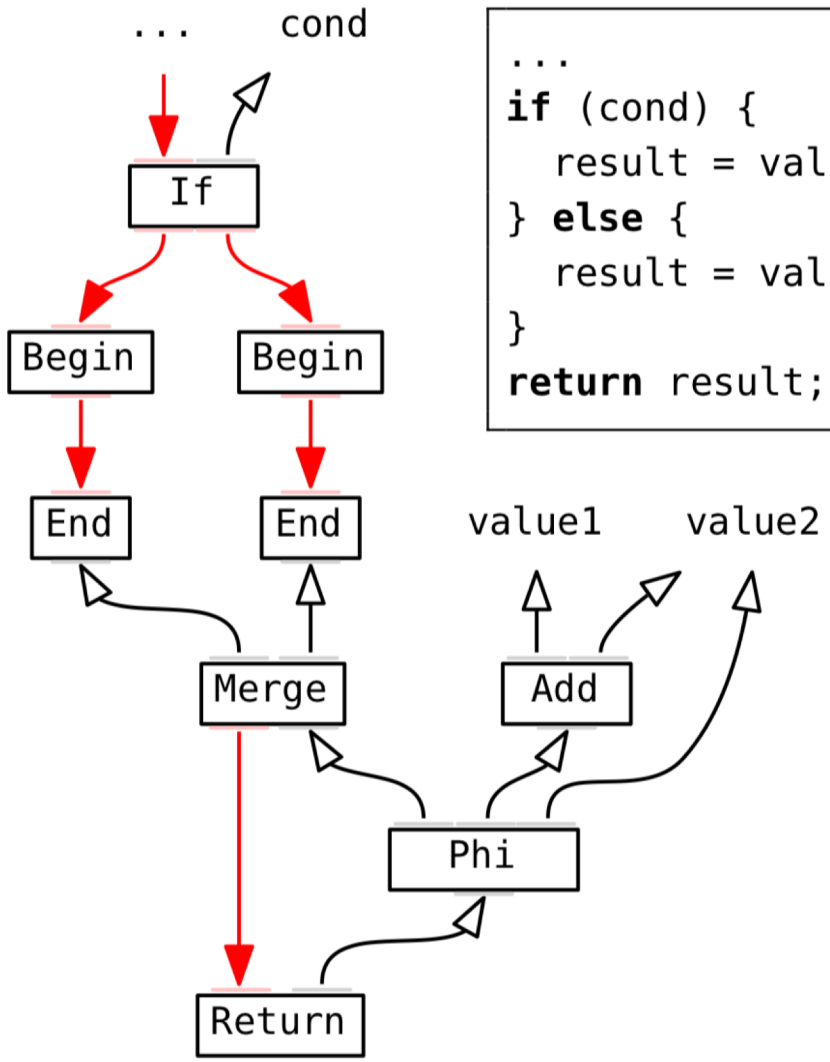
Thomas Würthinger* Christian Wimmer* Andreas Wöß† Lukas Stadler†
Gilles Duboscq† Christian Humer† Gregor Richards§ Doug Simon* Mario Wolczko*





What is Graal?

- A compiler from Java bytecode* to native machine code
  - Not a compiler from Java source code to Java bytecode
  - ‘Java bytecode’ could come from Truffle
- It’s a compiler library rather than being an executable, like GCC
- Some things I didn’t tie Graal down to there:
  - I didn’t say it was specifically a just-in-time compiler
  - I didn’t say the bytecode had to come directly from a program as written

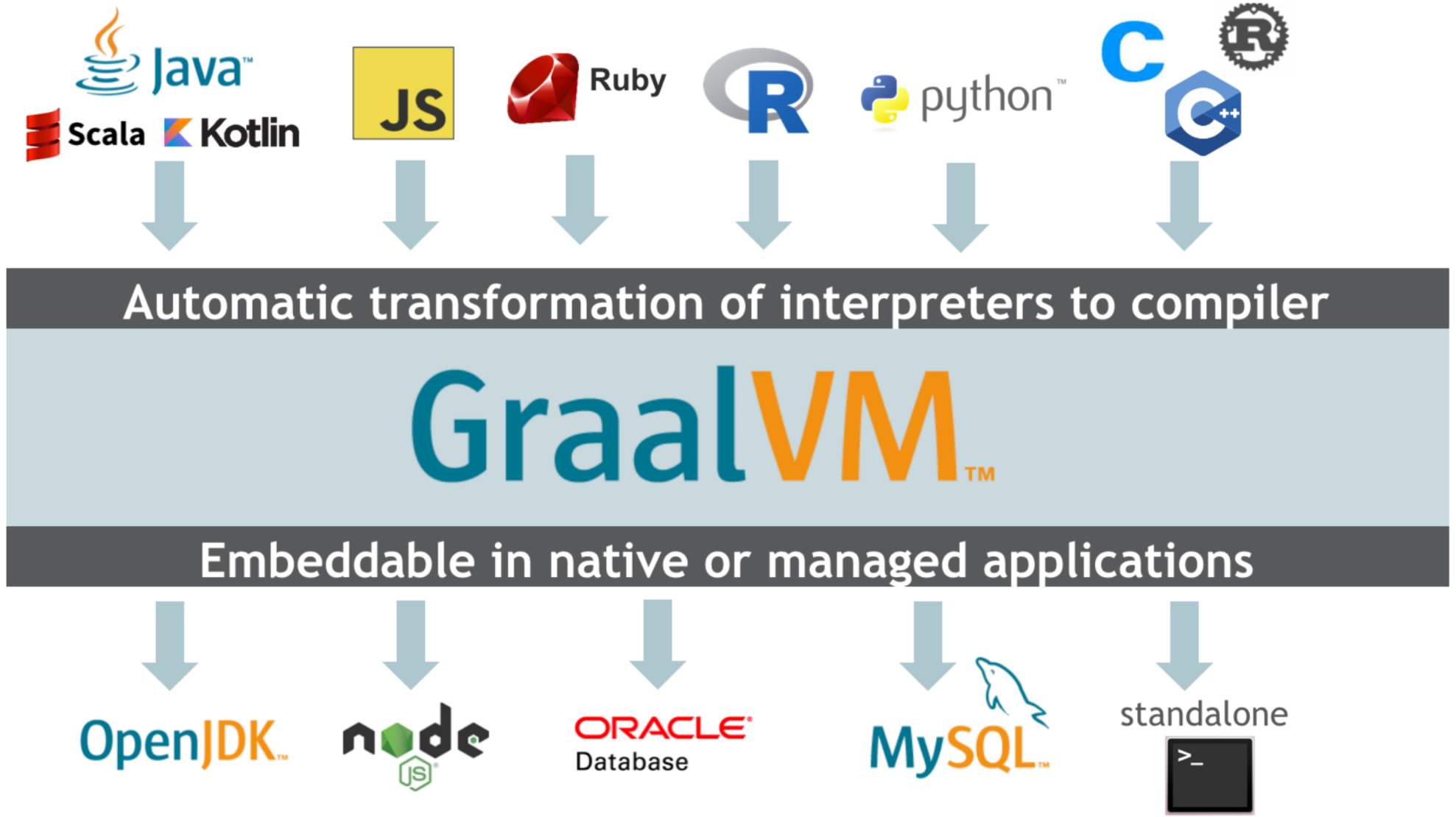


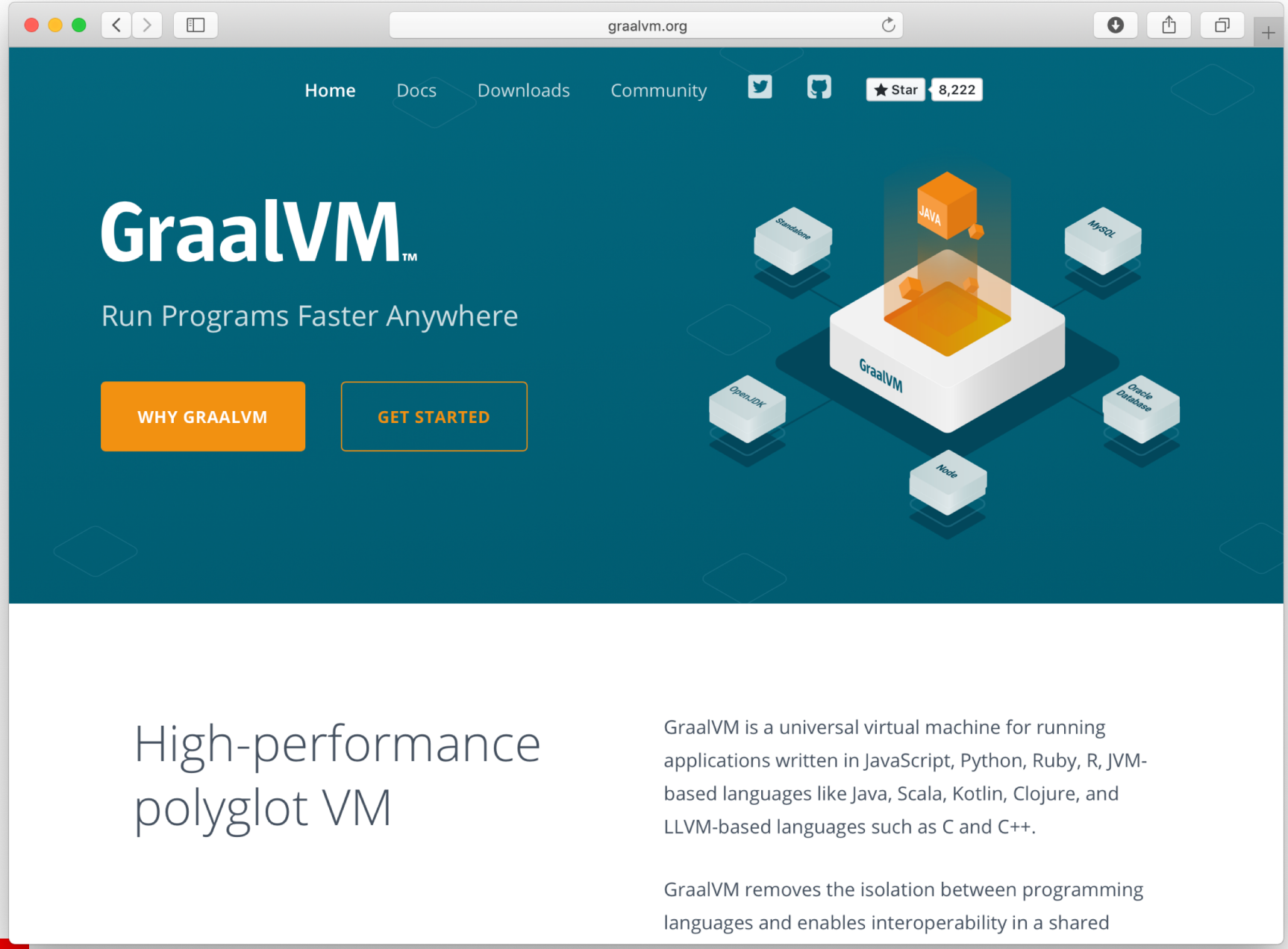

```
...
if (cond) {
  result = value1 + value2;
} else {
  result = value2;
}
return result;
```

Graal IR: An Extensible Declarative Intermediate Representation

Gilles Duboscq* Lukas Stadler* Thomas Würthinger†
Doug Simon† Christian Wimmer† Hanspeter Mössenböck*

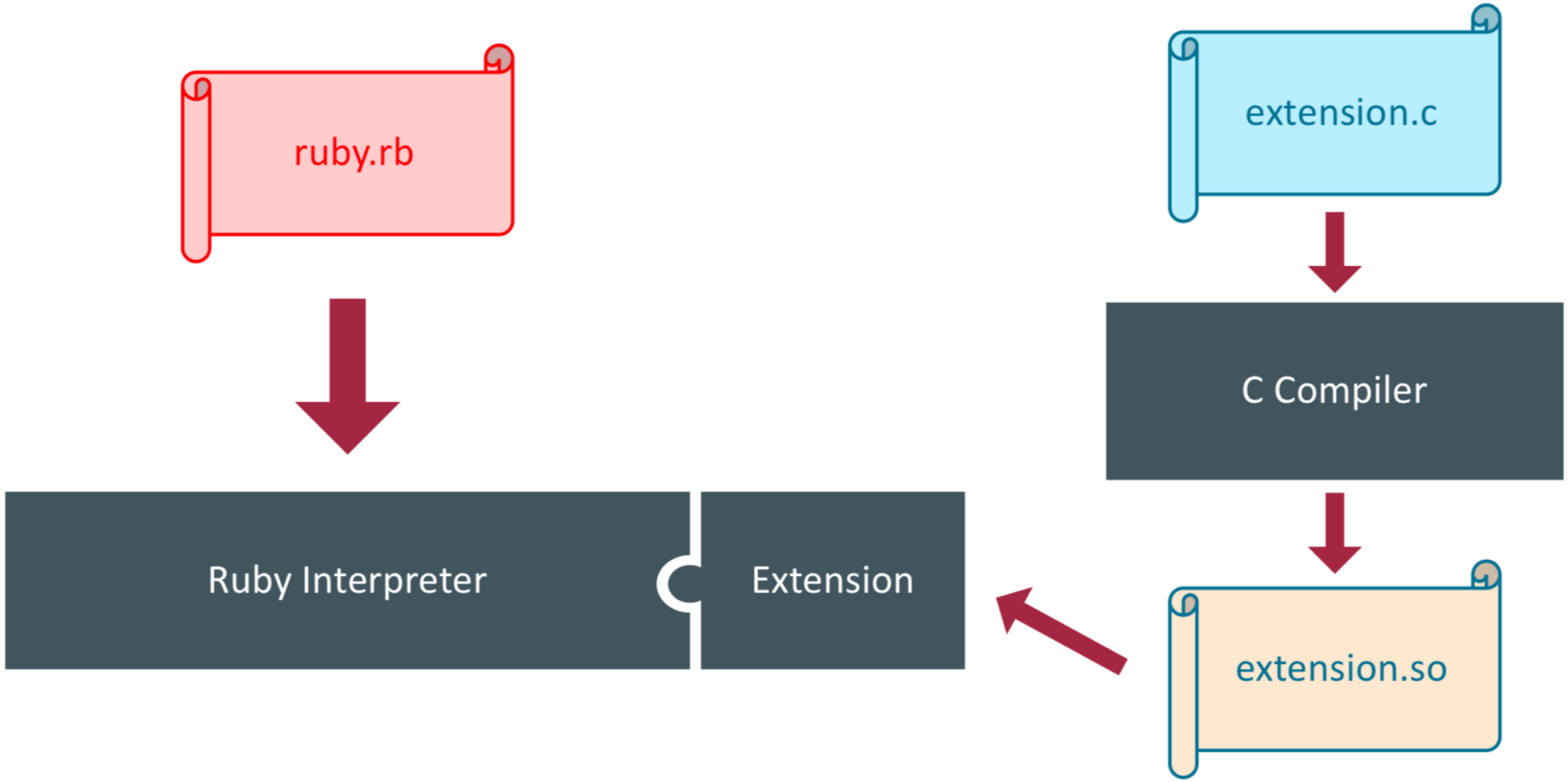
Polyglot

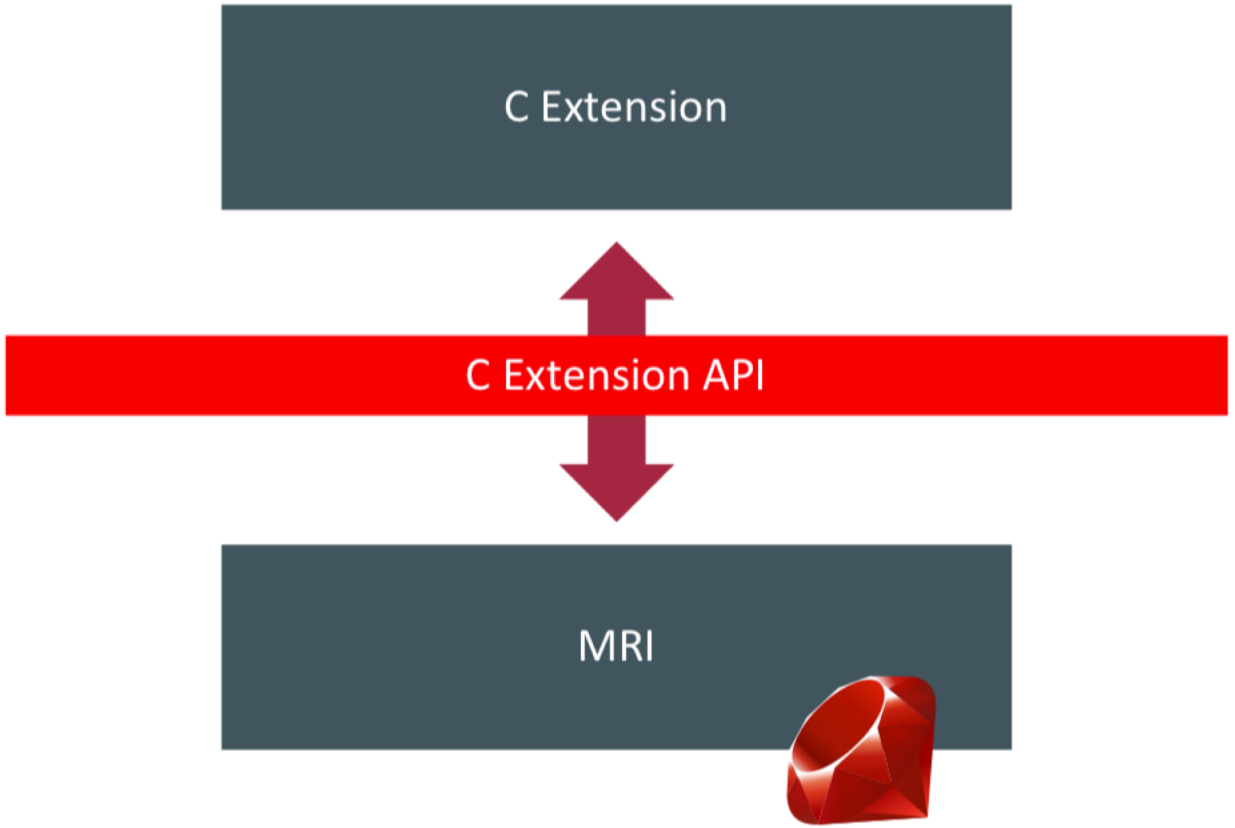


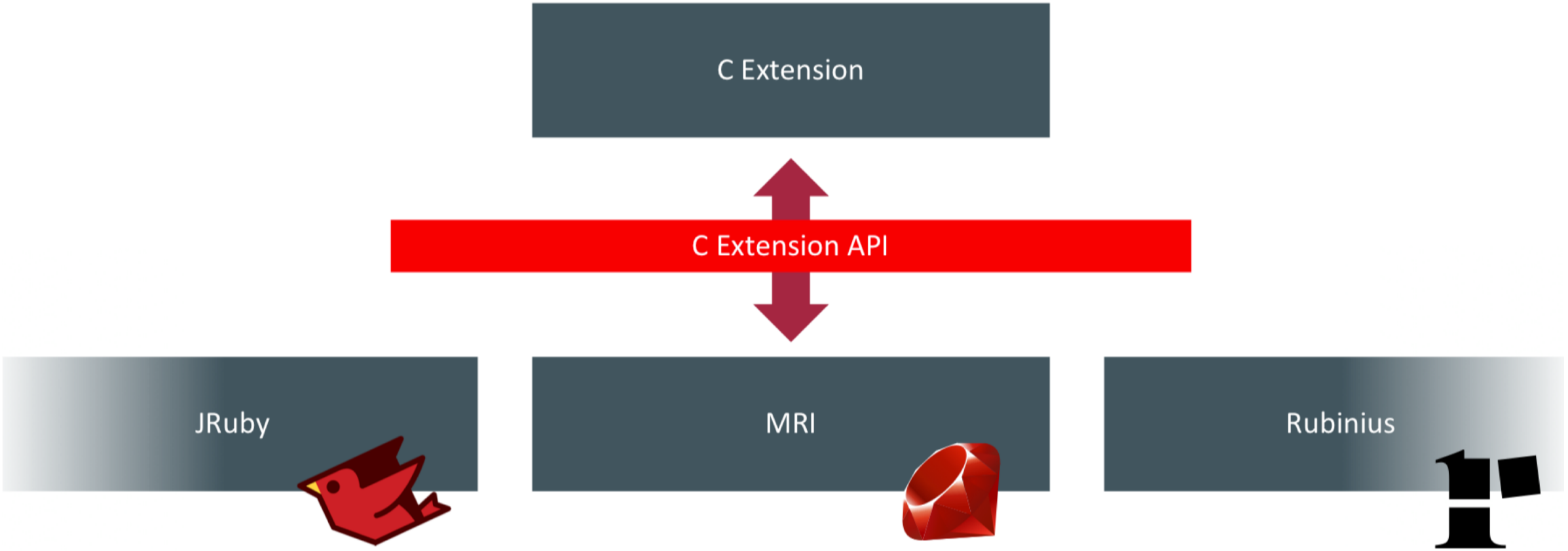


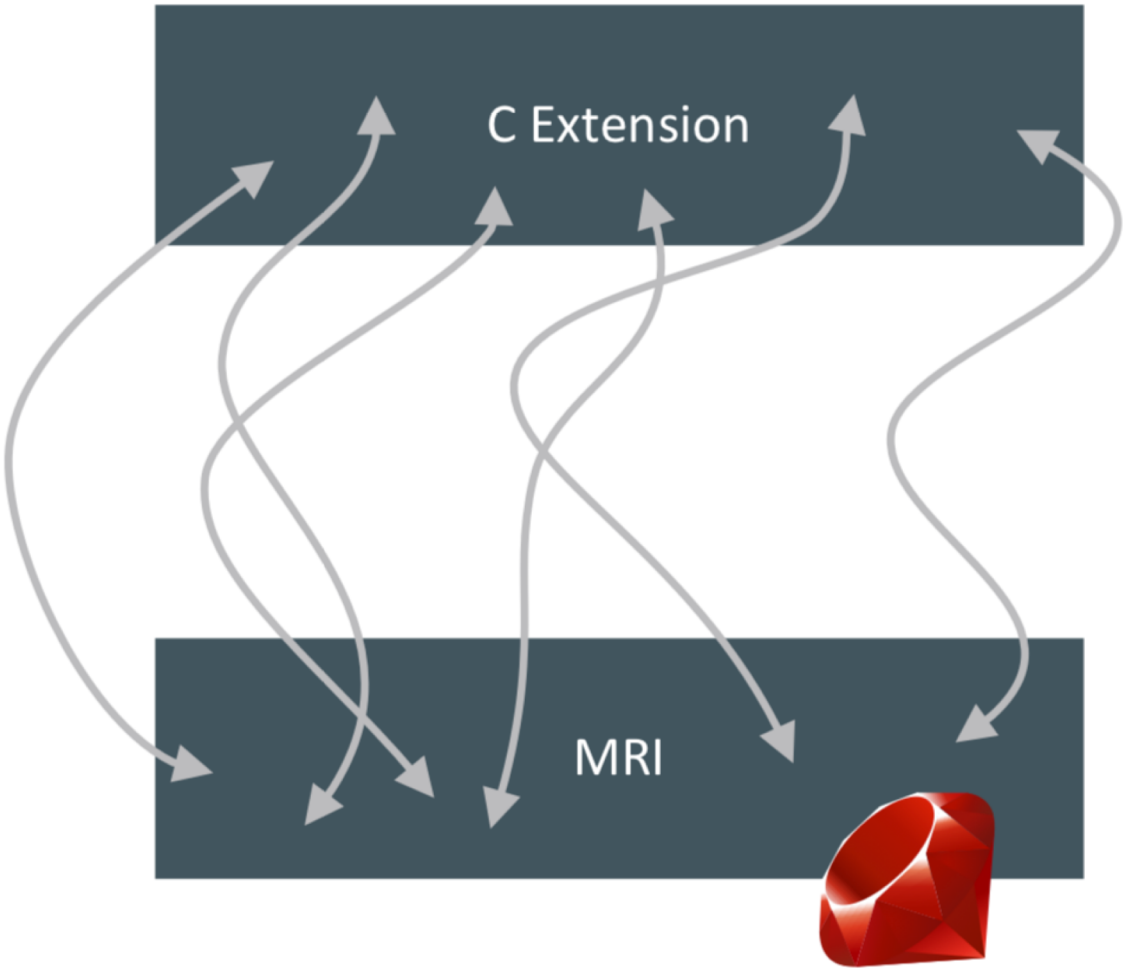
ORACLE

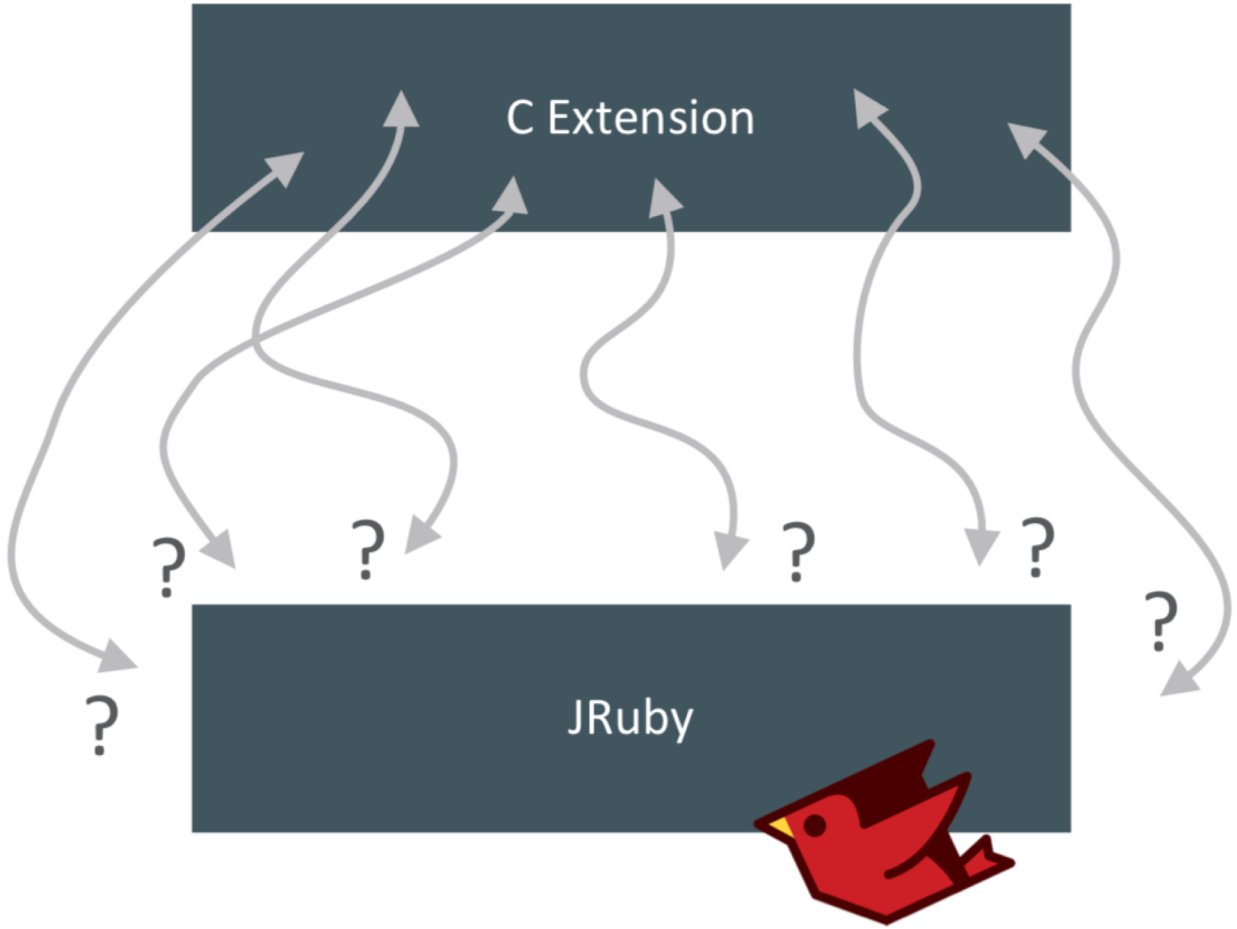
Running C extensions

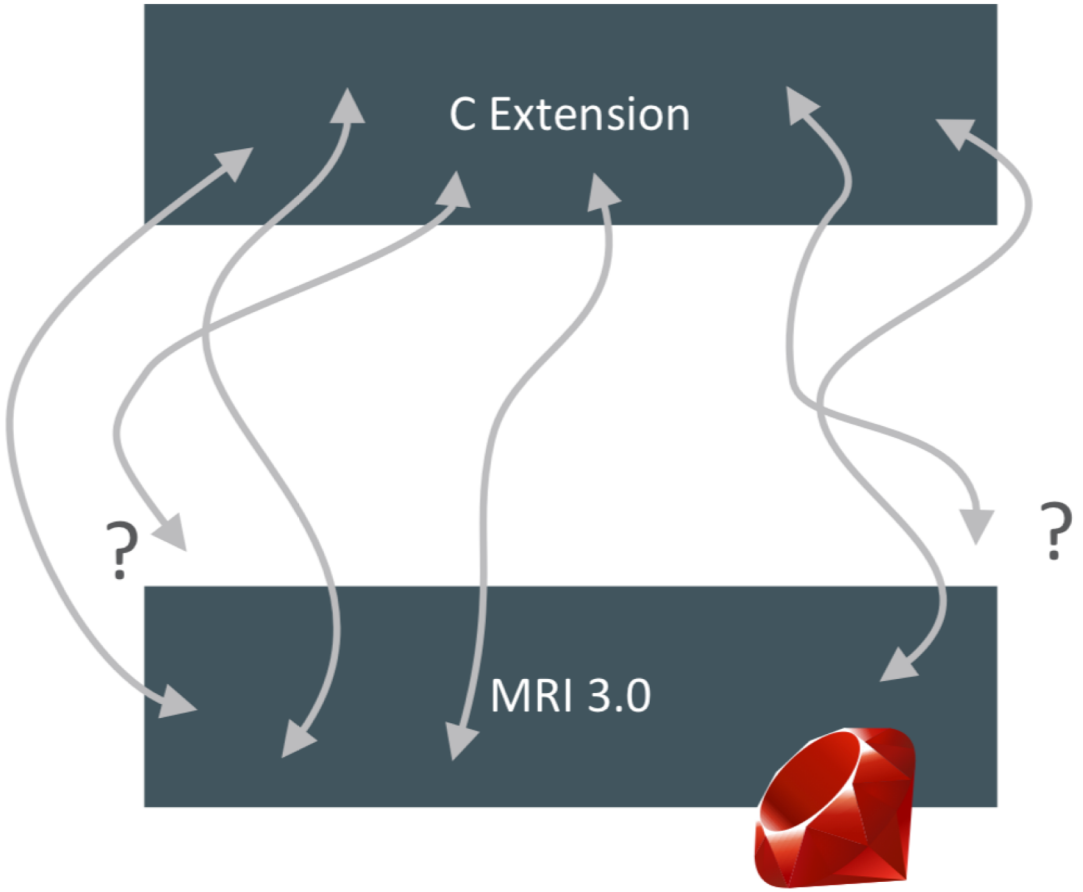


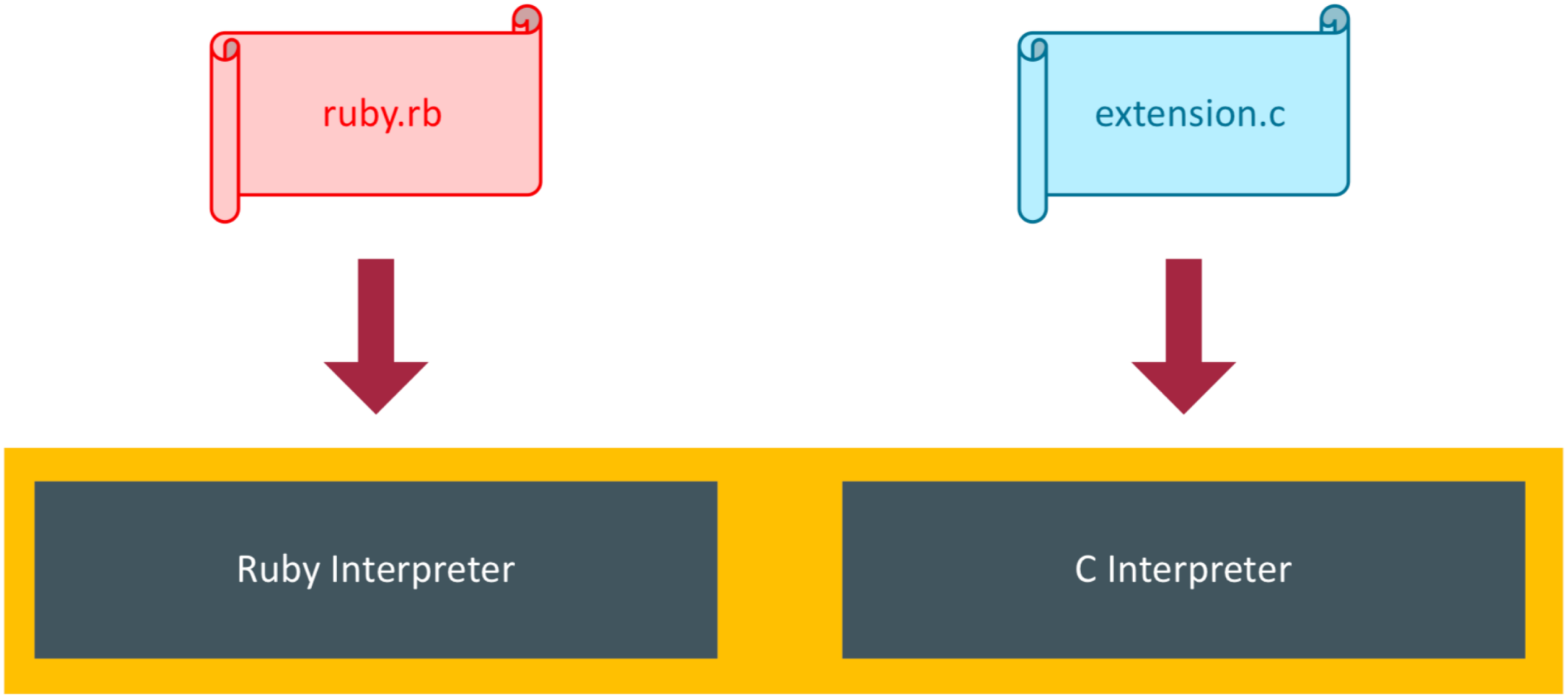


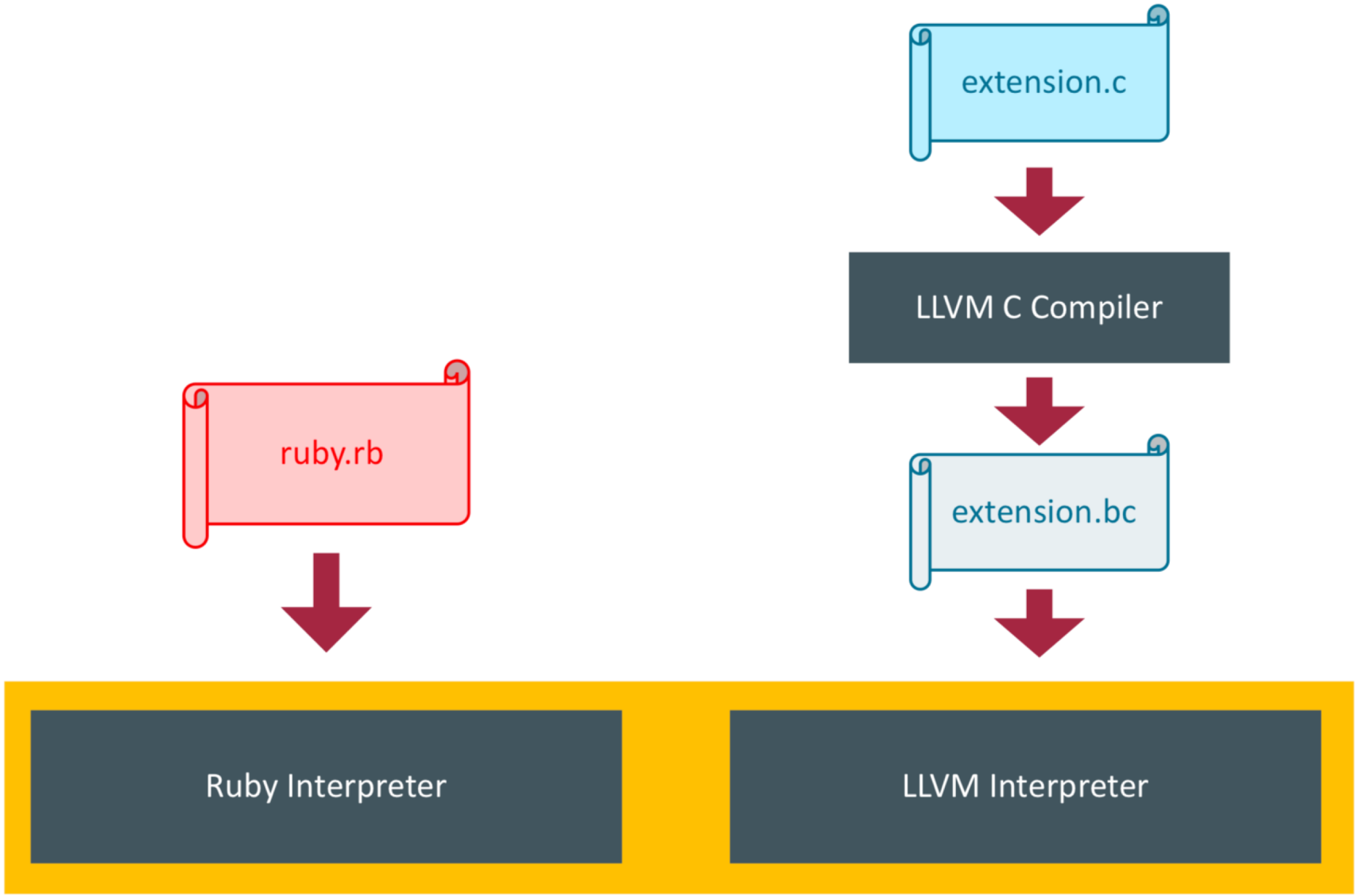


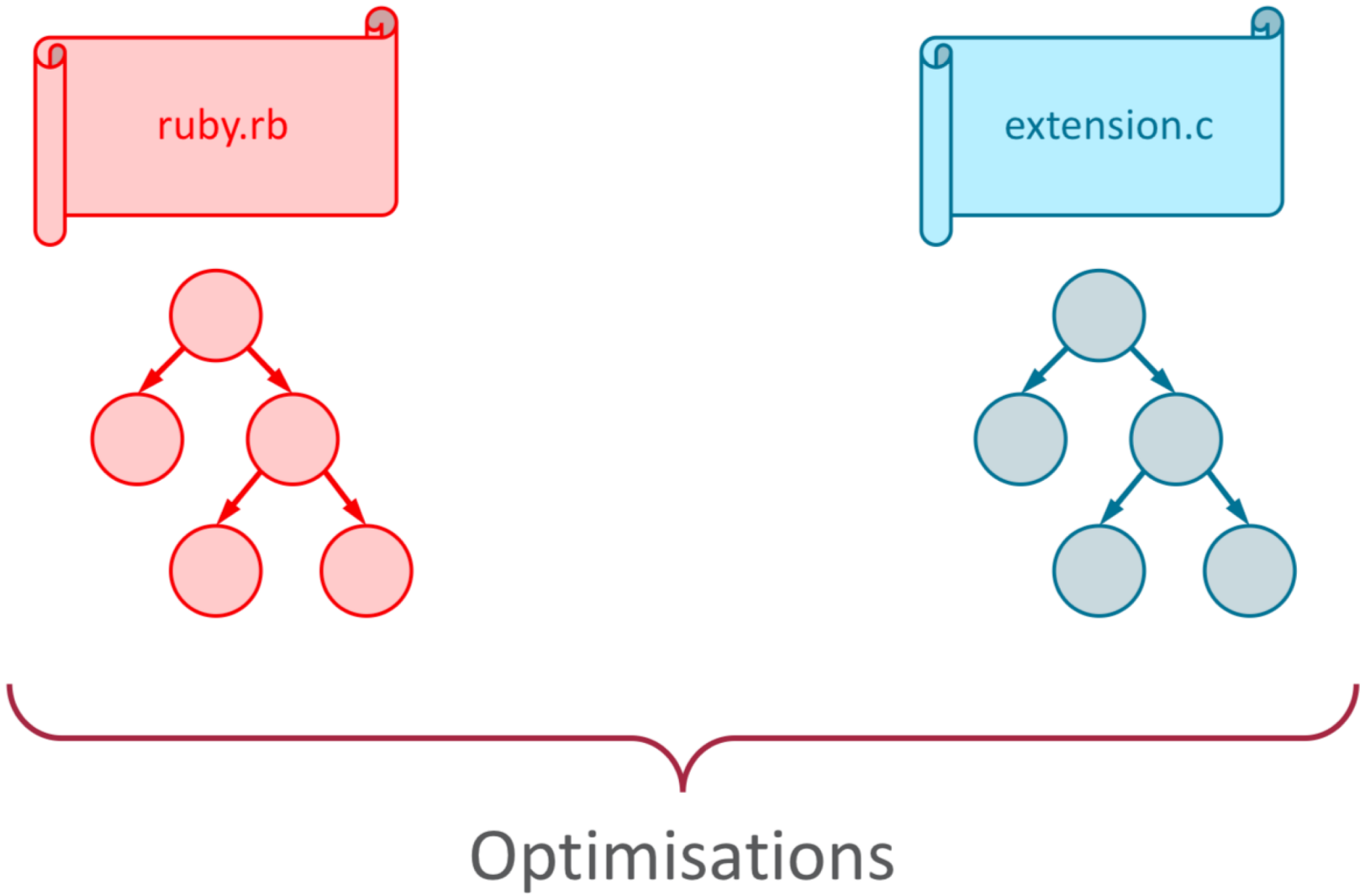


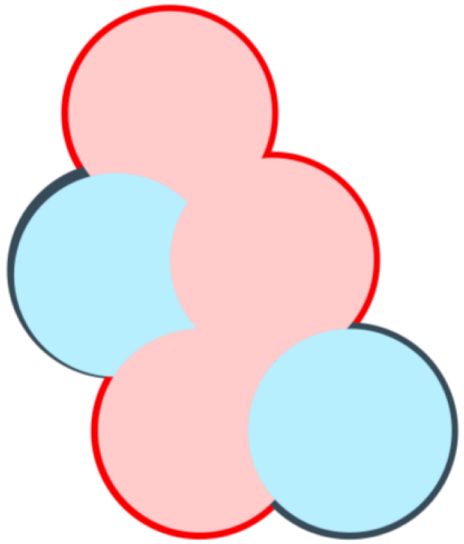












Optimisations

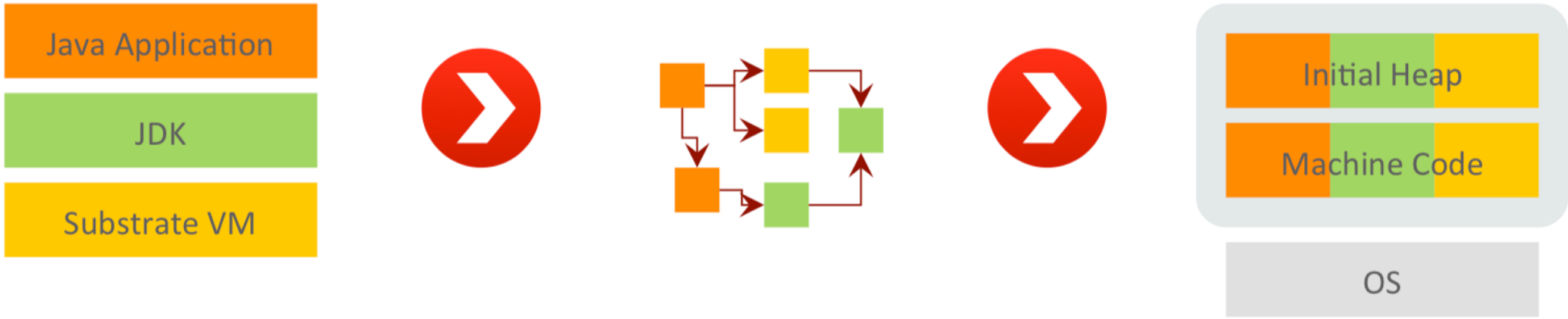
Native compilation

The JVM can be a bit of a burden

- required to run JRuby
- installed separately
- updated separately
- starts relatively slowly
- uses relatively large amount of memory

Static Analysis

Ahead-of-Time Compilation



All Java classes from application, JDK, and Substrate VM

Reachable methods, fields, and classes

Application running without compilation or class loading

Sum up

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

Questions

Safe Harbor Statement

The preceding is intended to provide some insight into a line of research in Oracle Labs. 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. Oracle reserves the right to alter its development plans and practices at any time, and the development, release, and timing of any features or functionality described in connection with any Oracle product or service remains at the sole discretion of Oracle. Any views expressed in this presentation are my own and do not necessarily reflect the views of Oracle.

Integrated Cloud

Applications & Platform Services


ORACLE®