

ORACLE®

Turning the JVM into a Polyglot VM with Graal

Chris Seaton
Research Manager
Oracle Labs
April 2017

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.

Programming languages



stackoverflow

Questions

Tags

Tour

Users

Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required.

Why can't there be an “ultimate” programming language?



stackoverflow

Questions

Tags

Tour

Users

Stack Overflow is a question and answer site for professional and enthusiast programmers. It's 100% free, no registration required.

Why can't there be an “ultimate” programming language?

closed as not constructive by [Tim](#), [Bo Persson](#), [Devon_C_Miller](#), [Mark](#),
[Graviton](#) Jan 17 at 5:58

[JavaScript: One language to rule them all | VentureBeat](#)



venturebeat.com/2011/.../javascript-one-language-to-rule-them-all/ ▾

by Peter Yared - in 23 Google+ circles

Jul 29, 2011 - Why code in two different scripting languages, one on the client and one on the server? It's time for **one language to rule them all**. Peter Yared ...

[PDF] [Python: One Script \(Language\) to rule them all - Ian Darwin](#)

www.darwinsys.com/python/python4unix.pdf ▾

Another **Language?** ▶ Python was invented in 1991 by Guido van. Rossum. ▷ Named after the comedy troupe, not the snake. ▶ Simple. ▷ They **all** say that!

[Q & Stuff: One Language to Rule Them All - Java](#)

qstuff.blogspot.com/2005/10/one-language-to-rule-them-all-java.html ▾

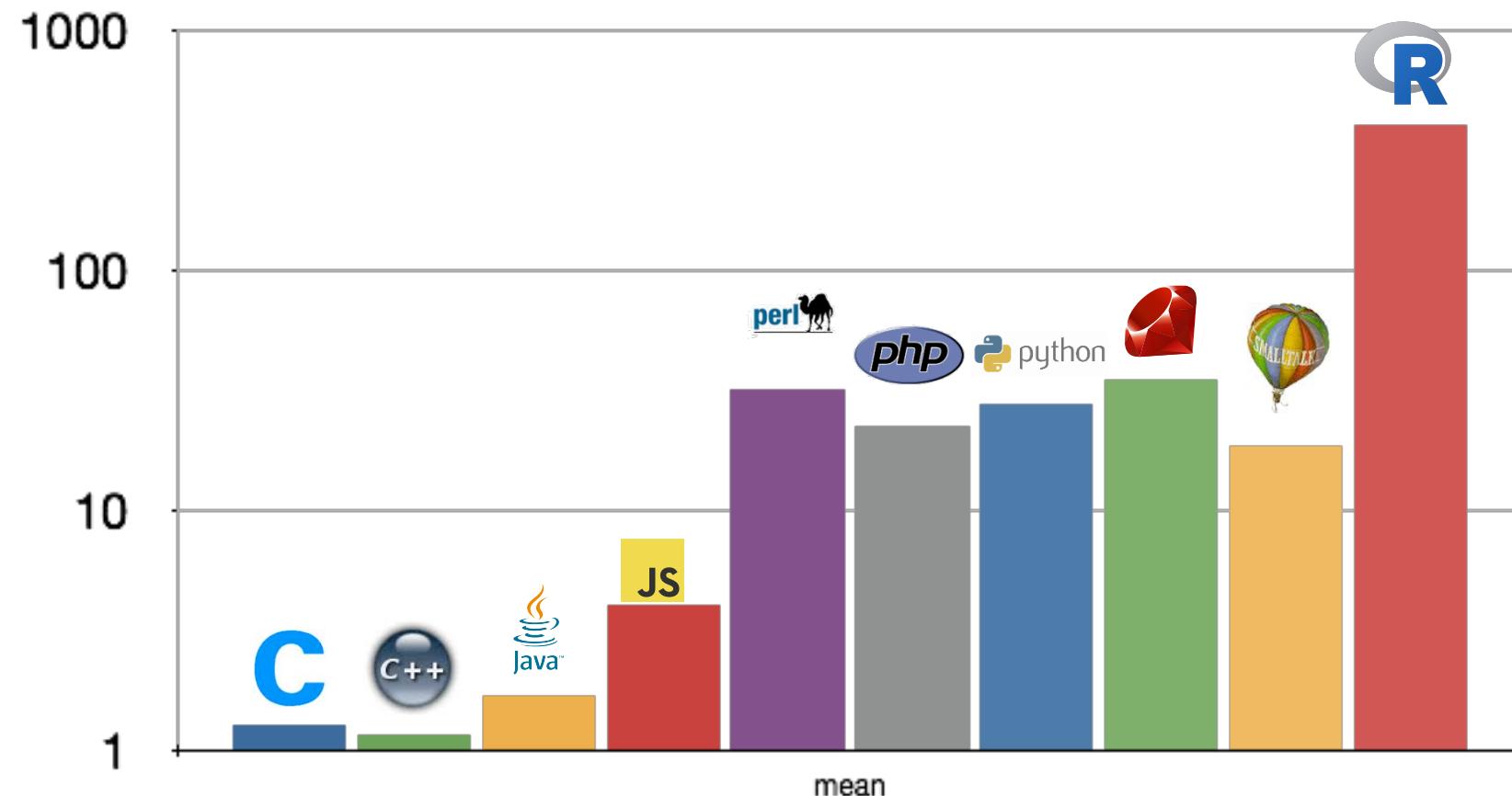
Oct 10, 2005 - **One Language to Rule Them All - Java**. For a long time I'd been hoping to add a scripting language to LibQ, to use in any of my (or other ...

[Dart : one language to rule them all - MixIT 2013 - Slideshare](#)

fr.slideshare.net/sdeleuze/dart-mixit2013en ▾

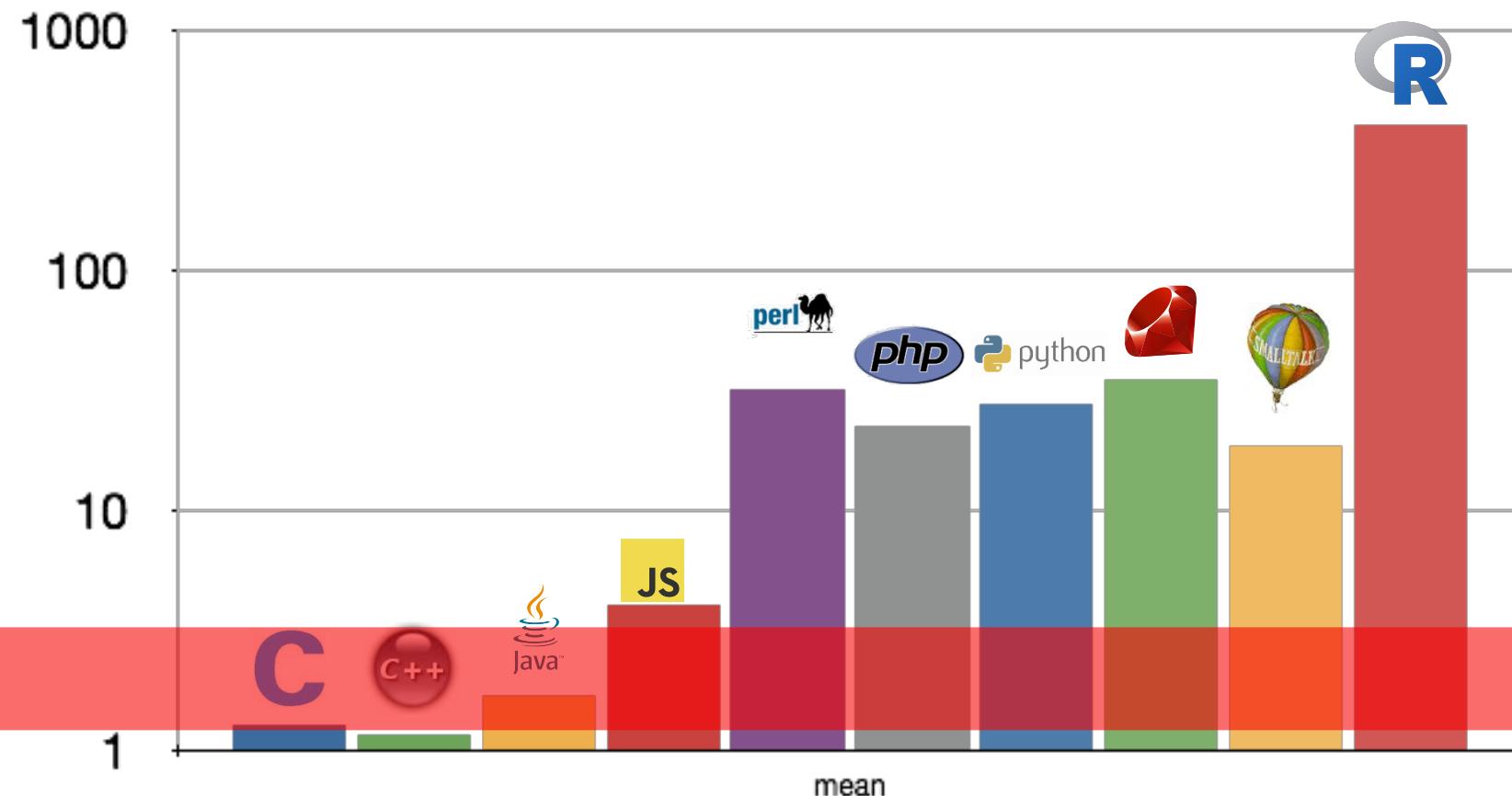
DartSébastien Deleuze - @sdeleuzeMix-IT 2013One language to rule them all ...

Computer Language Benchmarks Game



Computer Language Benchmarks Game

Goal:



Current situation

Prototype a new language

Parser and language work to build syntax tree (AST), AST Interpreter

Write a “real” VM

In C/C++, still using AST interpreter, spend a lot of time implementing runtime system, GC, ...

People start using it

People complain about performance

Define a bytecode format and write bytecode interpreter

Performance is still bad

Write a JIT compiler
Improve the garbage collector

Current situation

How it should be

Prototype a new language

Parser and language work to build syntax tree (AST), AST Interpreter

Write a “real” VM

In C/C++, still using AST interpreter, spend a lot of time implementing runtime system, GC, ...

People start using it

People complain about performance

Define a bytecode format and write bytecode interpreter

Performance is still bad

Write a JIT compiler
Improve the garbage collector

Prototype a new language in Java

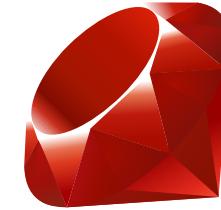
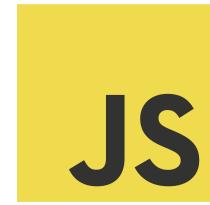
Parser and language work to build syntax tree (AST)
Execute using AST interpreter

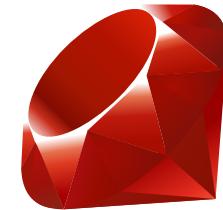
People start using it

And it is already fast

The GraalVM concept

The Ruby Logo is Copyright (c) 2006, Yukihiro Matsumoto. It is licensed under the terms of the Creative Commons Attribution-ShareAlike 2.5 agreement
JS Logo Copyright (c) 2011 Christopher Williams <chris@iterativedesigns.com>, MIT licence
You can distribute the R logo under the terms of the Creative Commons Attribution-ShareAlike 4.0 International license (CC-BY-SA 4.0) or (at your option) the GNU General Public License version 2 (GPL-2).





Impl

Impl

Impl

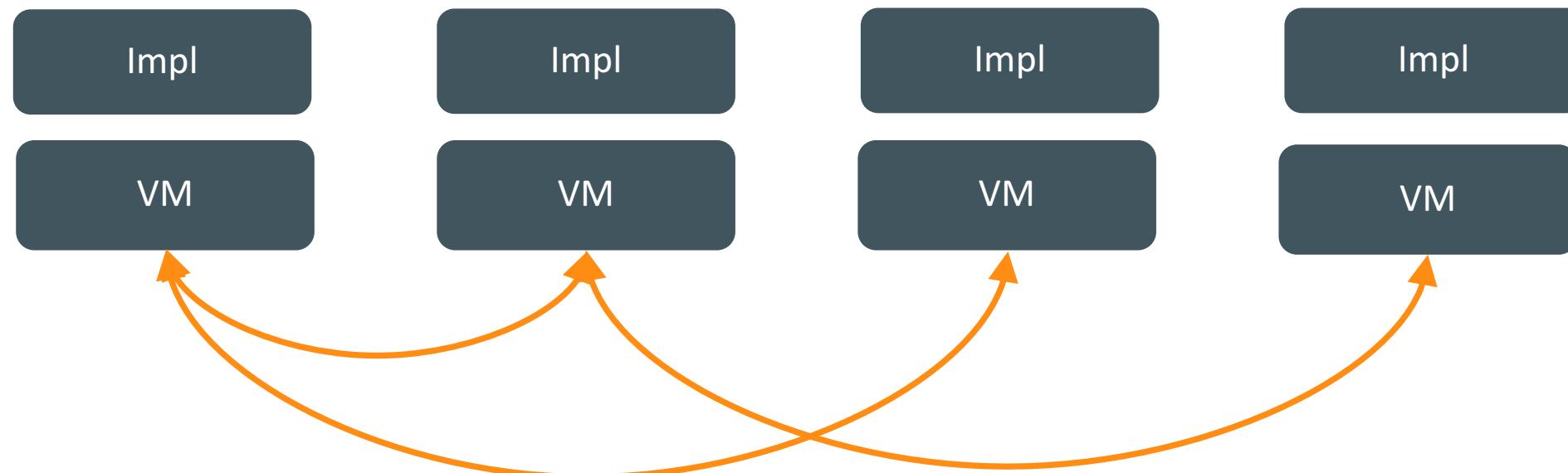
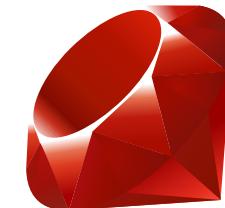
Impl

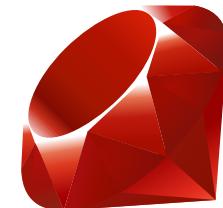
VM

VM

VM

VM





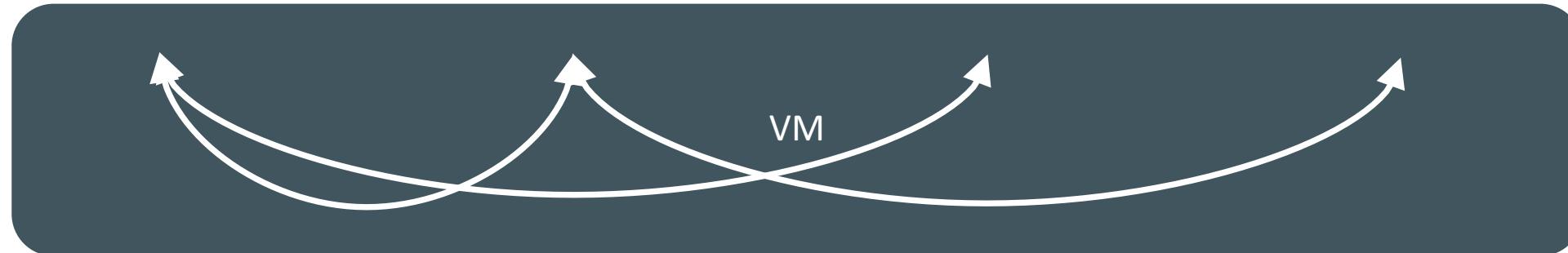
Impl

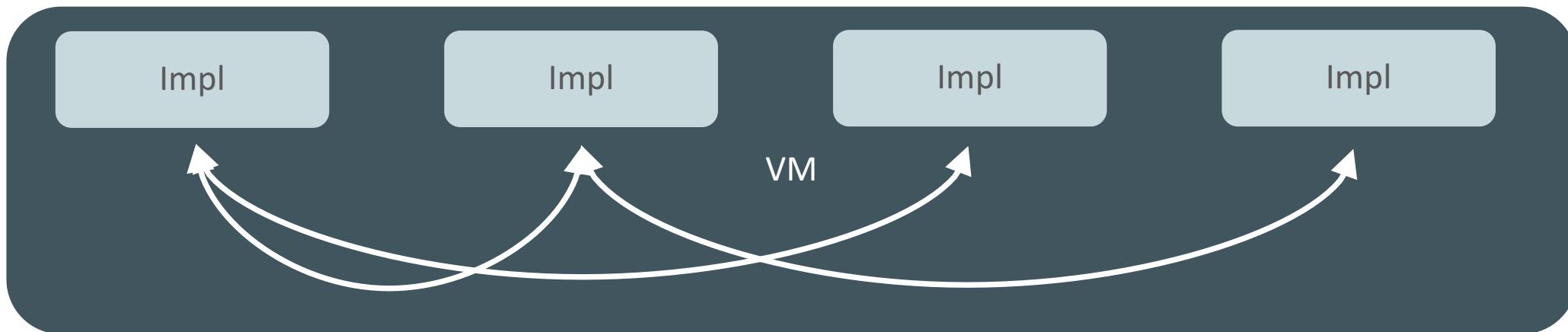
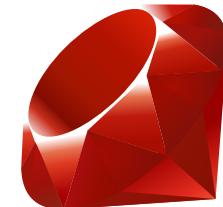
Impl

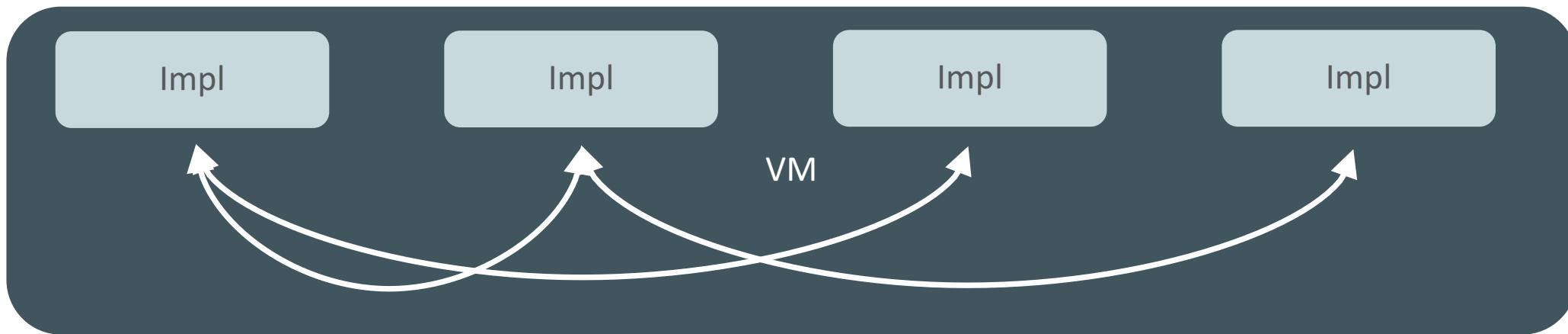
Impl

Impl

VM







How we do polyglot in GraalVM

```
Truffle::Interop.eval('application/language', source)
```

```
value = Truffle::Interop.import(name)
```

```
Truffle::Interop.export(name)
```

```
Interop.eval('application/language', source)
```

```
value = Interop.import(name)
```

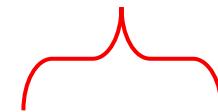
```
Interop.export(name)
```

```
puts Truffle::Interop.eval('application/javascript', '14 + 2')
# 16
```

Ruby

```
puts Truffle::Interop.eval('application/javascript', '14 + 2')  
# 16
```

JavaScript



```
Truffle::Interop.eval('application/javascript', "
    function add(a, b) {
        return a + b;
    }

    Interop.export('add', add.bind(this));
")

add = Truffle::Interop.import('add')

puts add.call(14, 2)
# 16
```

Ruby

```
Truffle::Interop.eval('application/javascript', "
  function add(a, b) {
    return a + b;
  }

  Interop.export('add', add.bind(this));
")

add = Truffle::Interop.import('add')

puts add.call(14, 2)
# 16
```

JavaScript

```
function add(a, b) {  
    return a + b;  
}  
  
puts add(14, 2)  
# 16
```

JavaScript

```
function add(a, b) {  
    return a + b;  
}
```

Ruby

```
puts add(14, 2)  
# 16
```

```
function Point(x, y) {
    this.x = x;
    this.y = y;
}

function random_points(n) {
    points = [];
    for (i = 0; i < n; i++) {
        points[i] = new Point(Math.random(), Math.random())
    }
    return points;
}

points = random_points(100)

point = points[0]
puts point.x, point.y
# 0.642460680339328
# 0.116305386298814
```

JS

```
function Point(x, y) {  
    this.x = x;  
    this.y = y;  
}  
  
function random_points(n) {  
    points = [];  
    for (i = 0; i < n; i++) {  
        points[i] = new Point(Math.random(), Math.random())  
    }  
    return points;  
}
```

Ruby

```
points = random_points(100)  
  
point = points[0]  
puts point.x, point.y  
# 0.642460680339328  
# 0.116305386298814
```

Performance

```
def clamp(num, min, max)
    [min, num, max].sort[1]
end

def cmyk_to_rgb(c, m, y, k)
    Hash[{
        r: (65535 - (c * (255 - k) + (k << 8))) >> 8,
        g: (65535 - (m * (255 - k) + (k << 8))) >> 8,
        b: (65535 - (y * (255 - k) + (k << 8))) >> 8
    }.map { |k, v| [k, clamp(v, 0, 255)] }]
end

benchmark do
    cmyk_to_rgb(rand(255), rand(255), rand(255), rand(255))
end
```

Warms up and then
reports iterations per
second

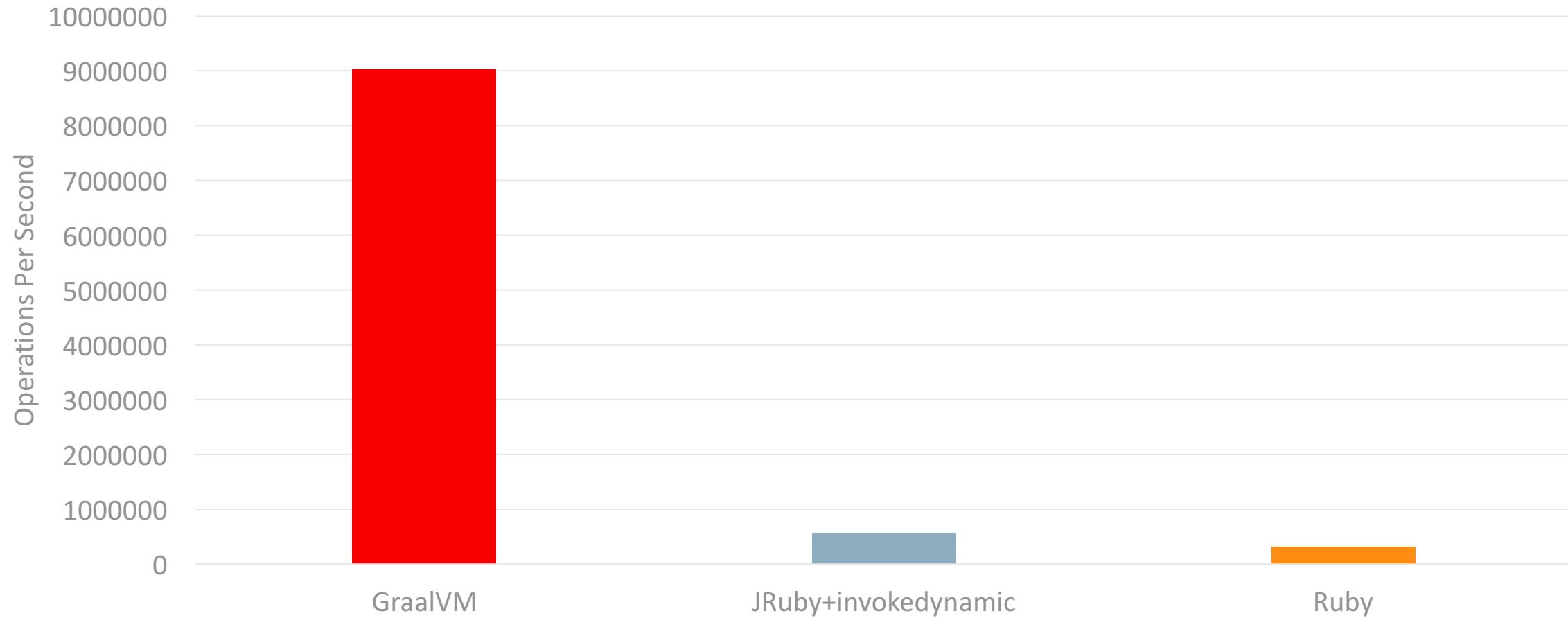
```
def clamp(num, min, max)
  [min, num, max].sort[1]
end

def cmyk_to_rgb(c, m, y, k)
  Hash[{
    r: (65535 - (c * (255 - k) + (k << 8))) >> 8,
    g: (65535 - (m * (255 - k) + (k << 8))) >> 8,
    b: (65535 - (y * (255 - k) + (k << 8))) >> 8
  }.map { |k, v| [k, clamp(v, 0, 255)] }]
end

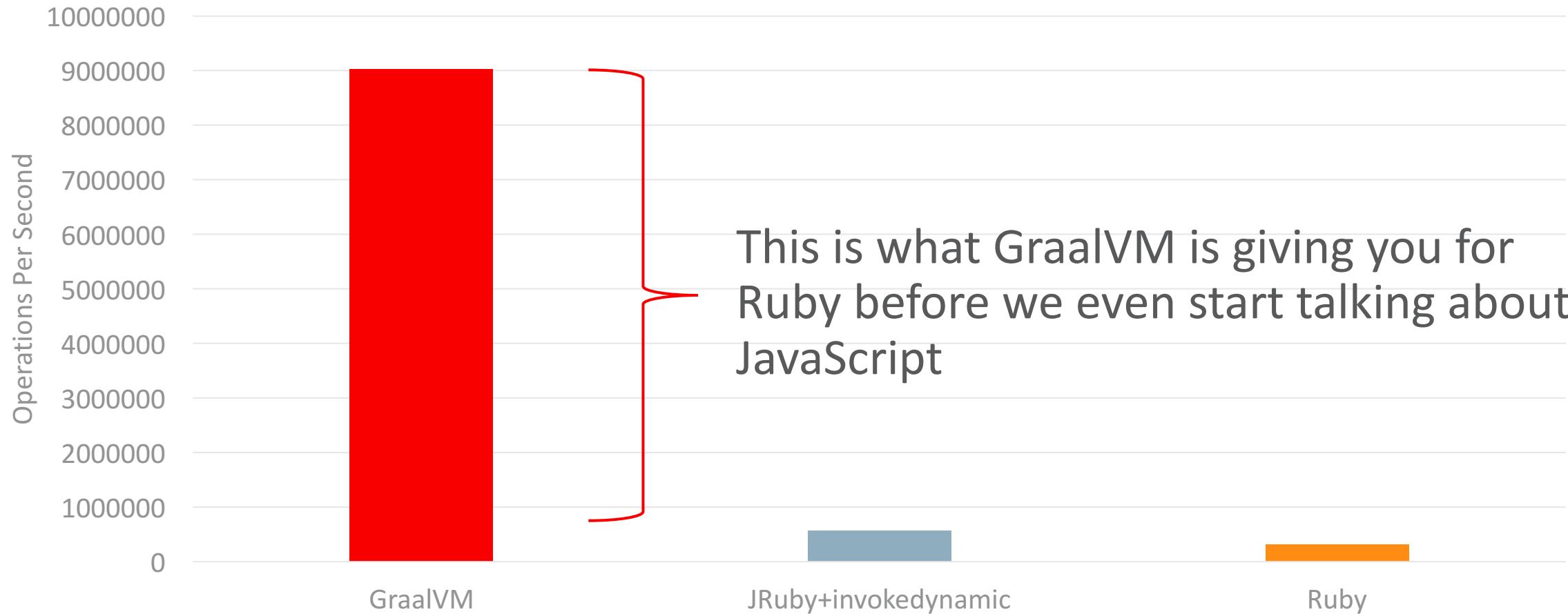
benchmark do
  cmyk_to_rgb(rand(255), rand(255), rand(255), rand(255))
end
```

Random inputs stop the
whole thing being totally
optimised away

clamp in Pure Ruby



clamp in Pure Ruby



```
require 'v8'

context = V8::Context.new

$clamp = context.eval("
  function clamp(num, min, max) {
    if (num < min) {
      return min;
    } else if (num > max) {
      return max;
    } else {
      return num;
    }
  }
  clamp;
")

def cmyk_to_rgb(c, m, y, k)
  Hash[{
    r: (65535 - (c * (255 - k) + (k << 8))) >> 8,
    g: (65535 - (m * (255 - k) + (k << 8))) >> 8,
    b: (65535 - (y * (255 - k) + (k << 8))) >> 8
  ].map { |k, v| [k, $clamp.call(v, 0, 255)] }
end
```

```
require 'v8'

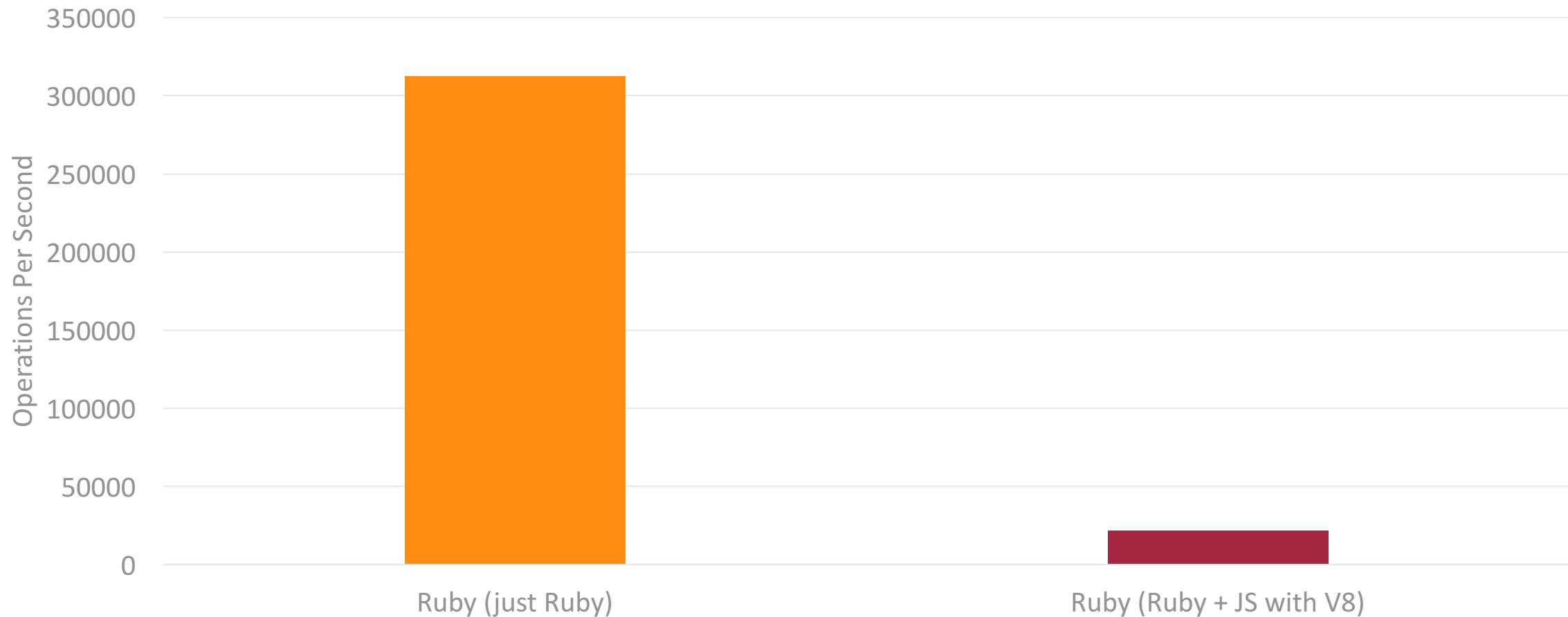
context = V8::Context.new

$clamp = context.eval("
  function clamp(num, min, max) {
    if (num < min) {
      return min;
    } else if (num > max) {
      return max;
    } else {
      return num;
    }
  }
  clamp;
")

def cmyk_to_rgb(c, m, y, k)
  Hash[{
    r: (65535 - (c * (255 - k) + (k << 8))) >> 8,
    g: (65535 - (m * (255 - k) + (k << 8))) >> 8,
    b: (65535 - (y * (255 - k) + (k << 8))) >> 8
  ].map { |k, v| [k, $clamp.call(v, 0, 255)] }
end
```

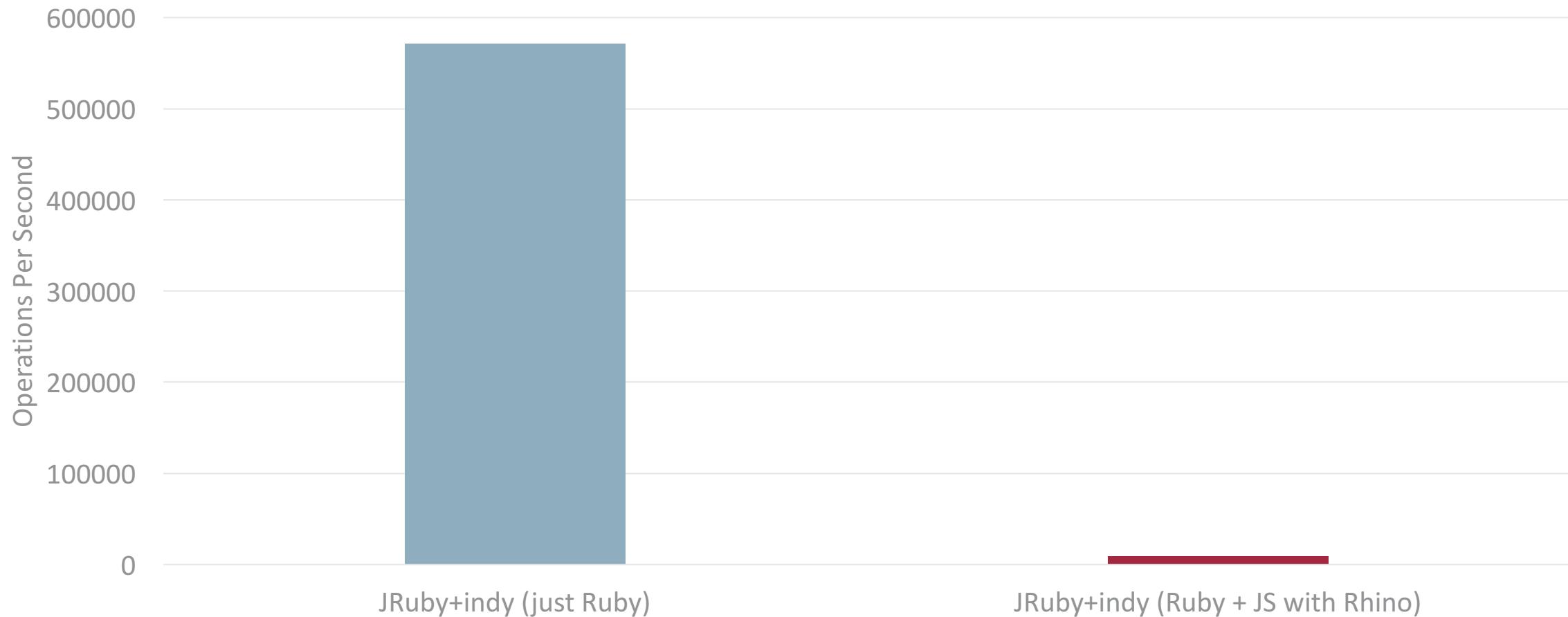
Not only have we rewritten
in JavaScript, but the
JavaScript code is simpler
than the Ruby

clamp in Ruby and JavaScript with V8



```
require 'rhino'  
  
context = Rhino::Context.new
```

clamp in Ruby and JavaScript with JRuby and Rhino

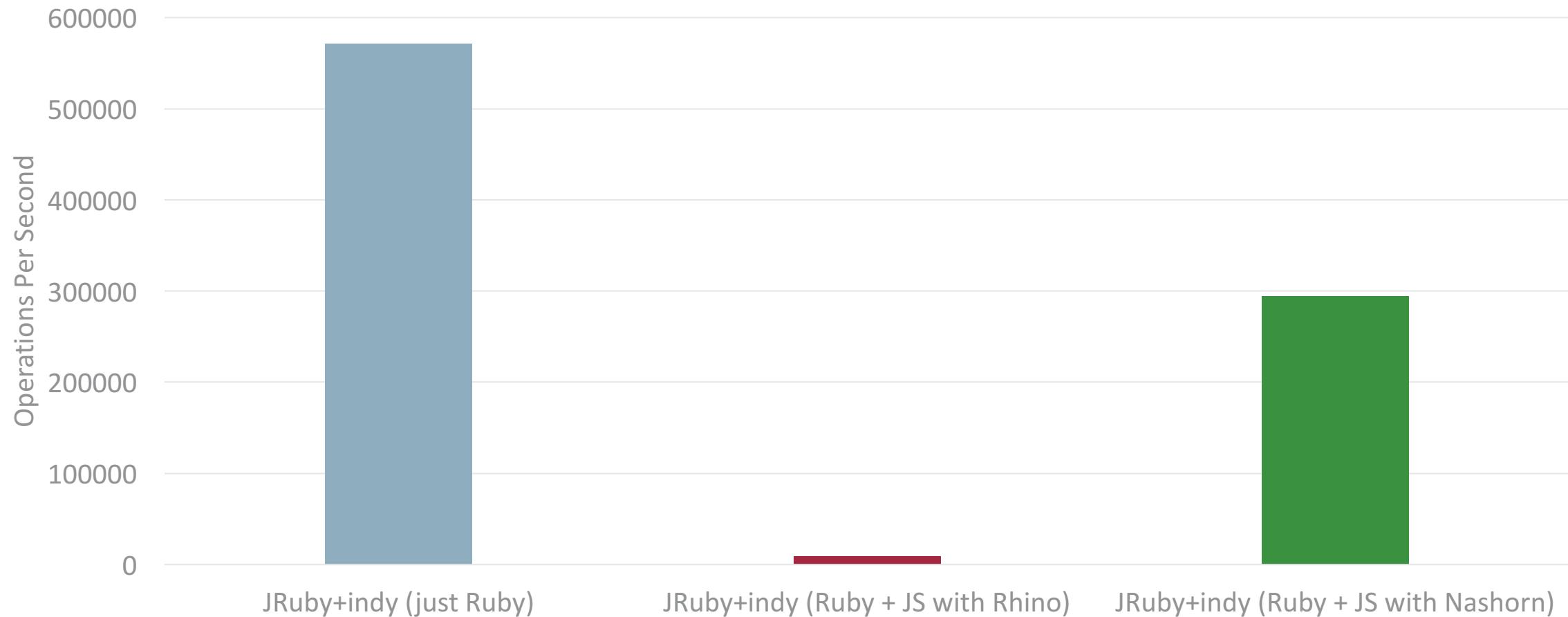


```
factory = javax.script.ScriptEngineManager.new
engine = factory.getEngineByName 'nashorn'
bindings = engine.createBindings

$clamp = engine.eval("
    function clamp(num, min, max) {
        if (num < min) {
            return min;
        } else if (num > max) {
            return max;
        } else {
            return num;
        }
    }
", bindings)

def cmyk_to_rgb(c, m, y, k)
  Hash[{
    r: (65535 - (c * (255 - k) + (k << 8))) >> 8,
    g: (65535 - (m * (255 - k) + (k << 8))) >> 8,
    b: (65535 - (y * (255 - k) + (k << 8))) >> 8
  ].map { |k, v| [k, $clamp.call(v, 0, 255)] }
end
```

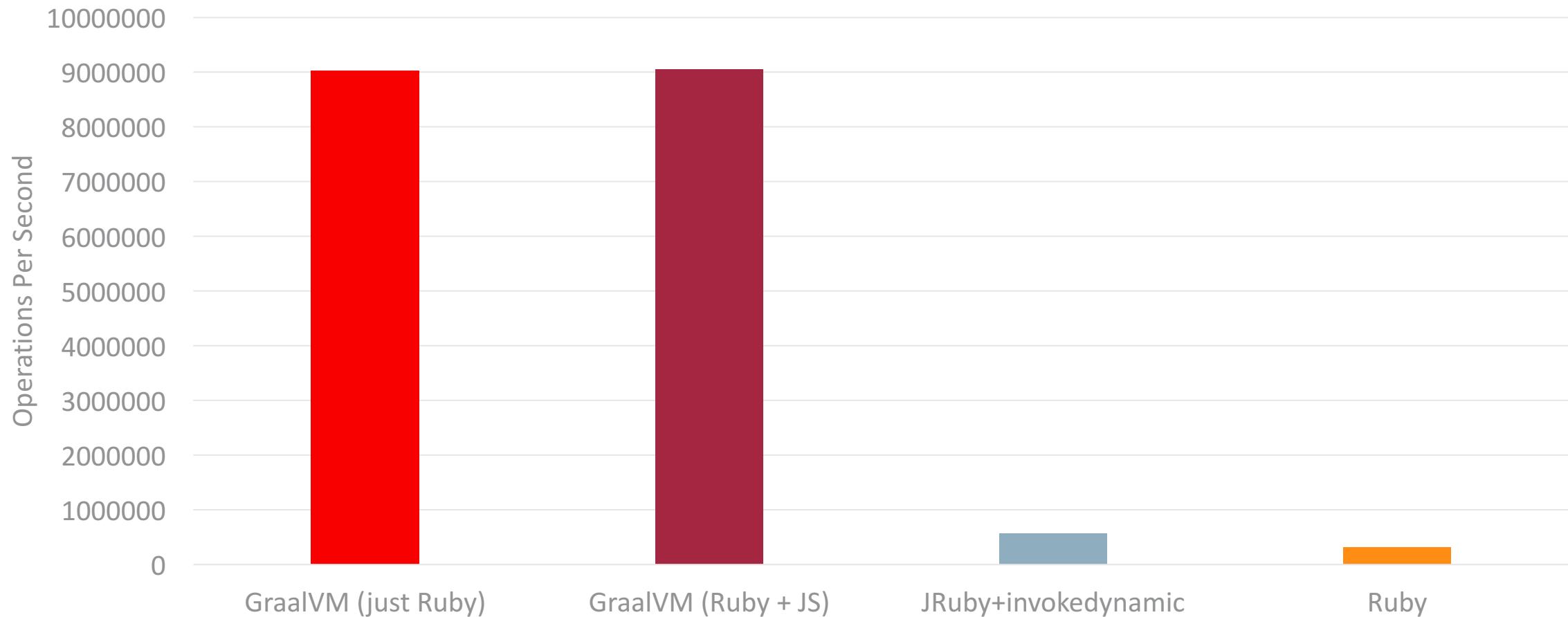
clamp in Ruby and JavaScript with JRuby and Nashorn



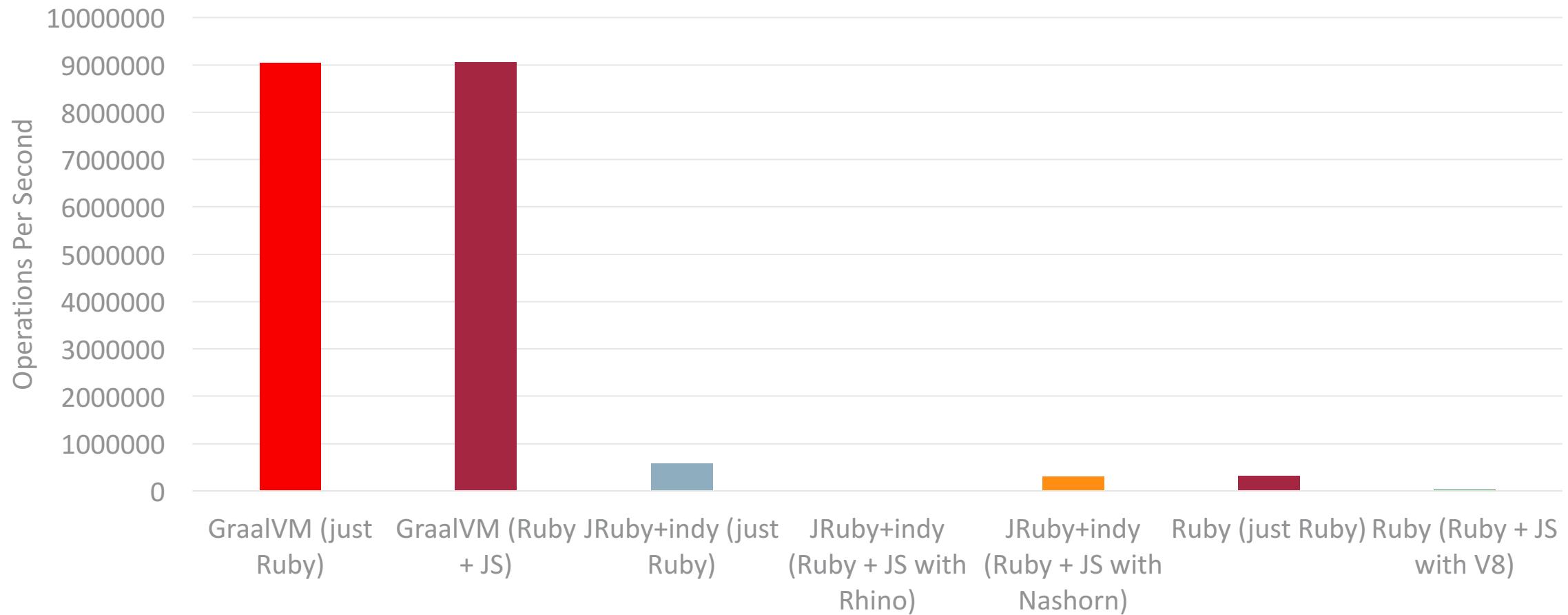
```
function clamp(num, min, max) {
    if (num < min) {
        return min;
    } else if (num > max) {
        return max;
    } else {
        return num;
    }
}

def cmyk_to_rgb(c, m, y, k)
    Hash[{
        r: (65535 - (c * (255 - k) + (k << 8))) >> 8,
        g: (65535 - (m * (255 - k) + (k << 8))) >> 8,
        b: (65535 - (y * (255 - k) + (k << 8))) >> 8
    ].map { |k, v| [k, clamp(v, 0, 255)] }
end
```

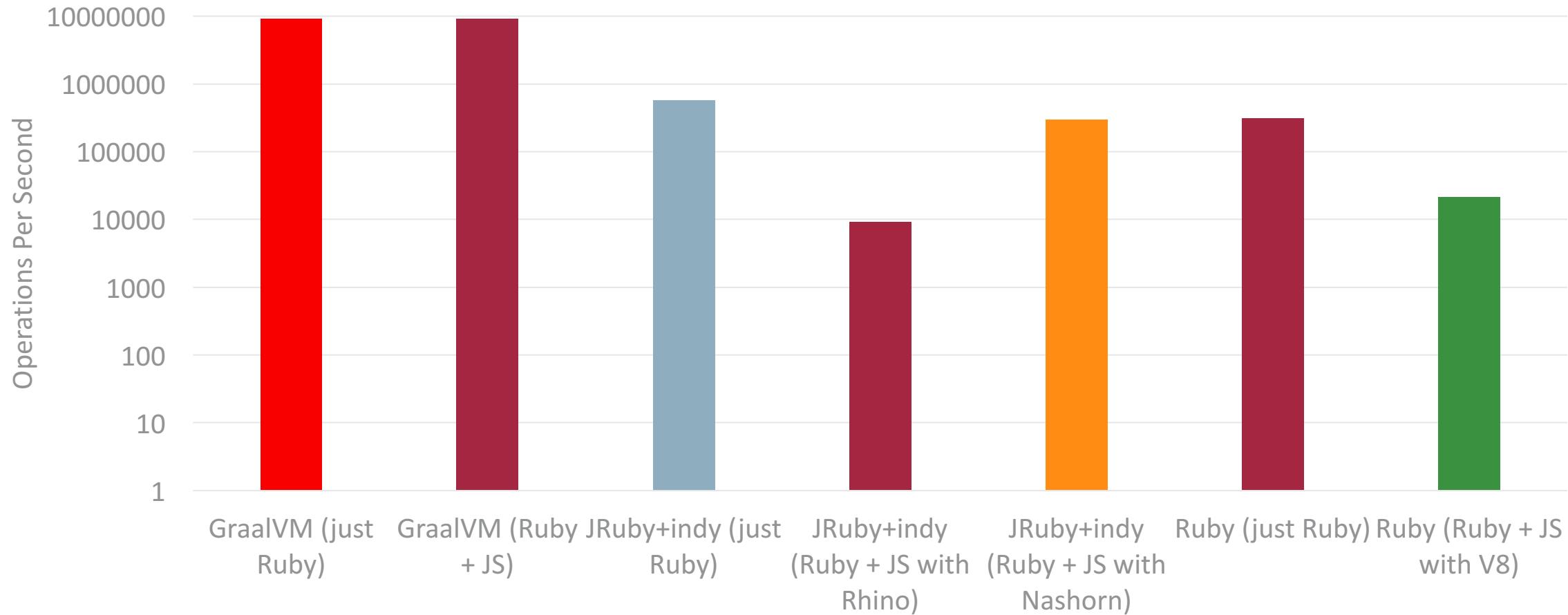
clamp in Ruby and JavaScript with GraalVM



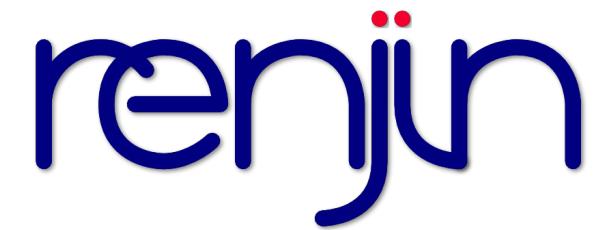
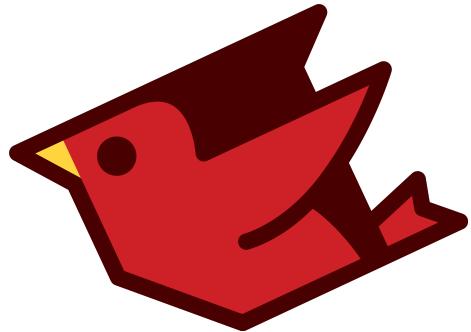
clamp in all configurations



clamp in all configurations



How Graal achieves this



Conventional JVM implementations of languages work by emitting
JVM bytecode – the same thing that the Java compiler does

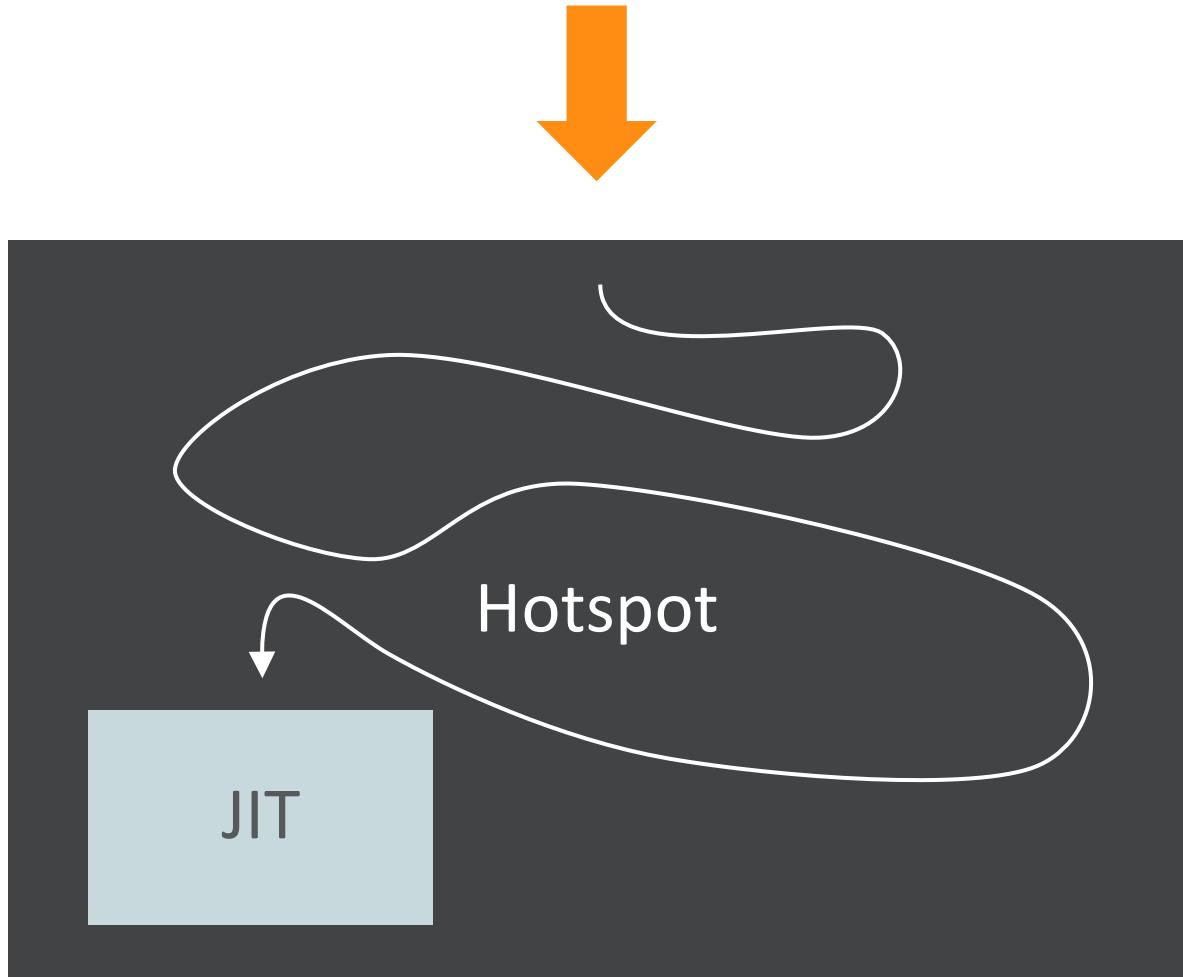


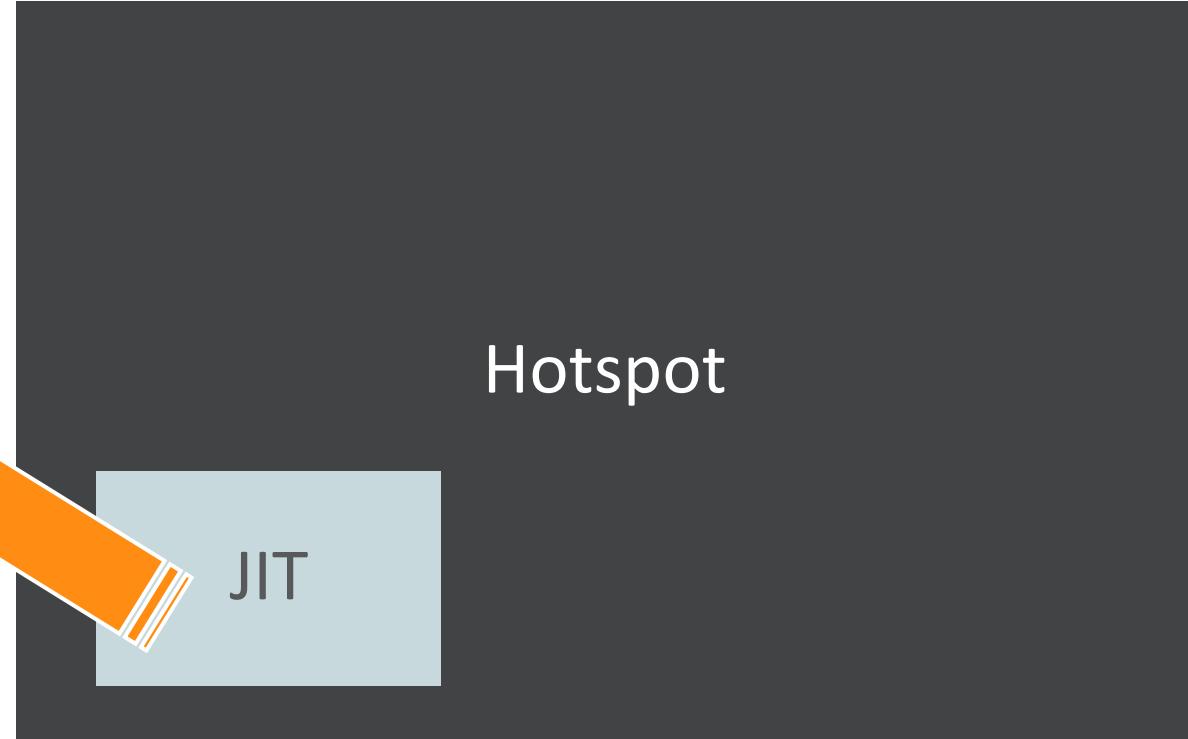
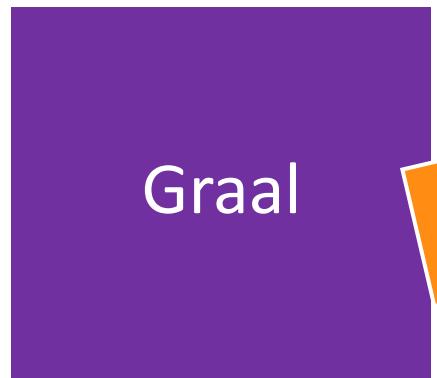
Hotspot



Hotspot







Graal

Hotspot

Truffle



Graal

Hotspot

Slightly confusing terminology...

- Graal is a new JIT compiler for the JVM
- Graal VM is the JVM, with Graal, Truffle, and our languages bundled in it
- Truffle uses Graal on your behalf

Guest Language



Bytecode

JVM

Guest Language



Compiler internal data
structures, optimisation passes,
machine code, ...

Graal

Guest Language



language interpreter

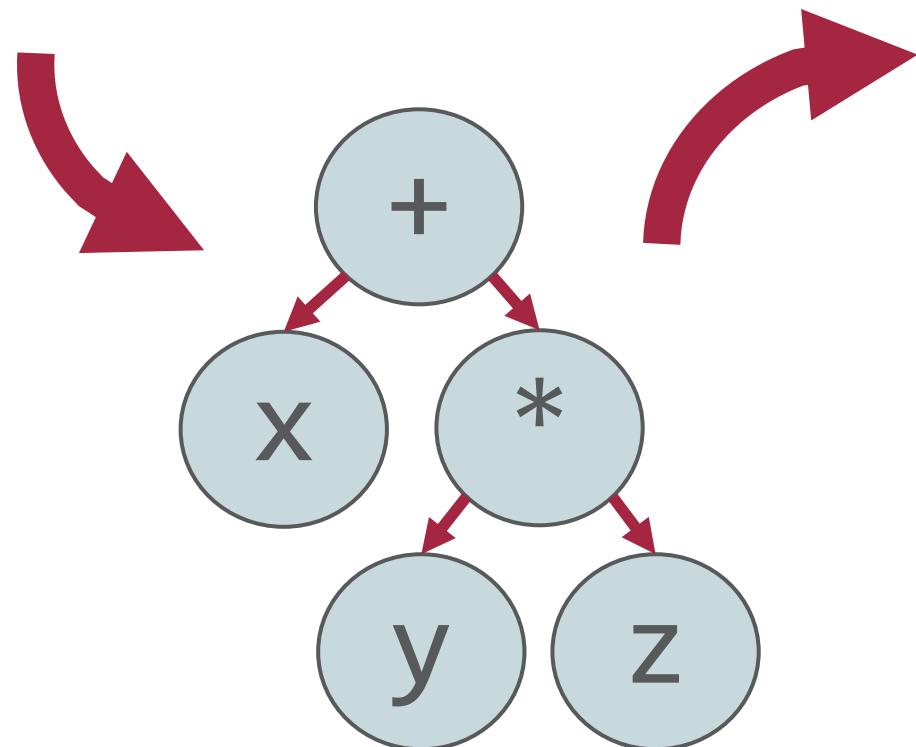
Truffle



Graal

The very basics of Truffle and Graal

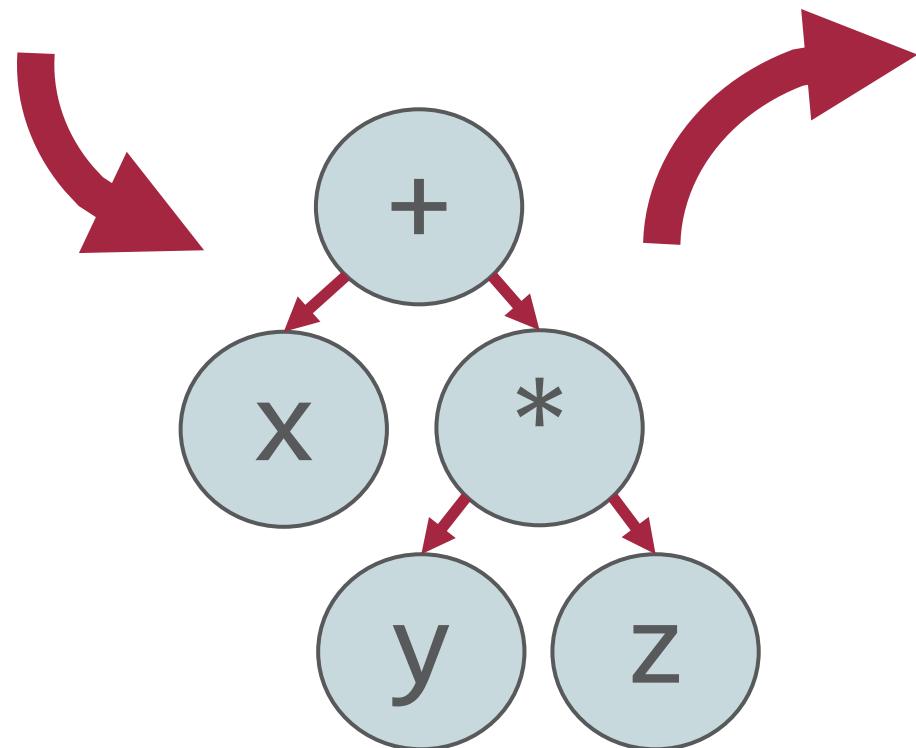
$x + y * z$



load_local x
load_local y
load_local z
call *
call +

pushq %rbp
movq %rsp, %rbp
movq %rdi, -8(%rbp)
movq %rsi, -16(%rbp)
movq %rdx, -24(%rbp)
movq -16(%rbp), %rax
movl %eax, %edx
movq -24(%rbp), %rax
imull %edx, %eax
movq -8(%rbp), %rdx
addl %edx, %eax
popq %rbp
ret

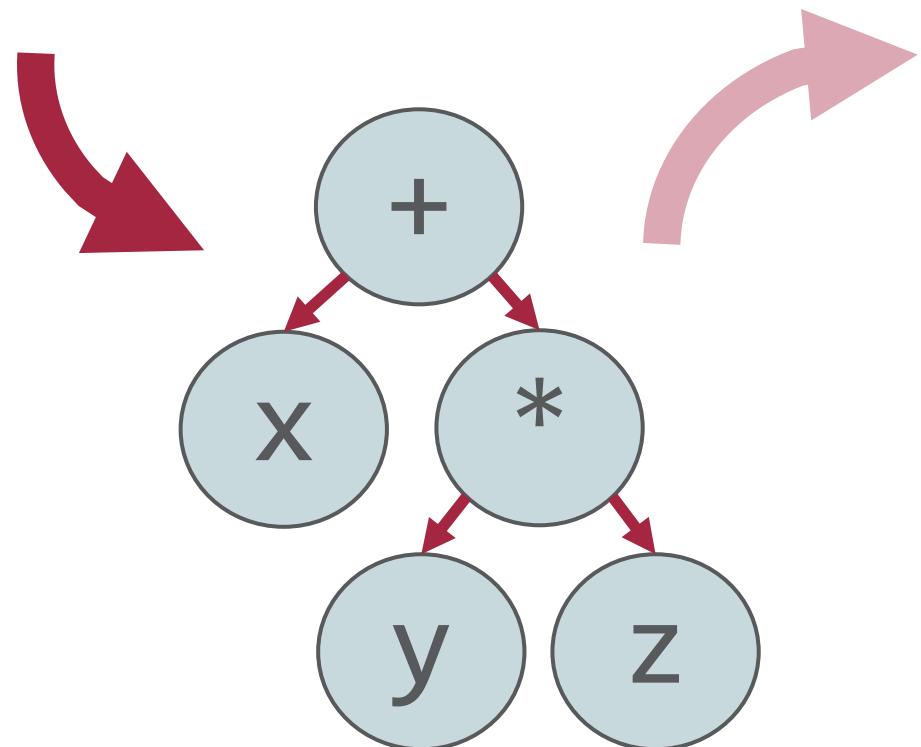
$x + y * z$



load_local x
load_local y
load_local z
call *
call +

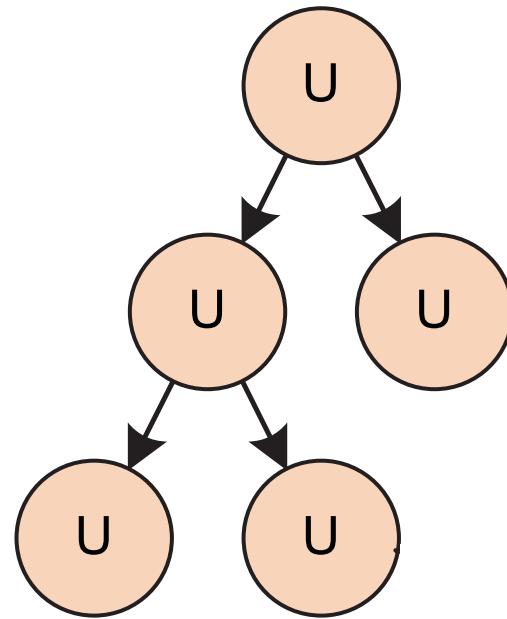
pushq %rbp
movq %rsp, %rbp
movq %rdi, -8(%rbp)
movq %rsi, -16(%rbp)
movq %rdx, -24(%rbp)
movq -16(%rbp), %rax
movl %eax, %edx
movq -24(%rbp), %rax
imull %edx, %eax
movq -8(%rbp), %rdx
addl %edx, %eax
popq %rbp
ret

$x + y * z$



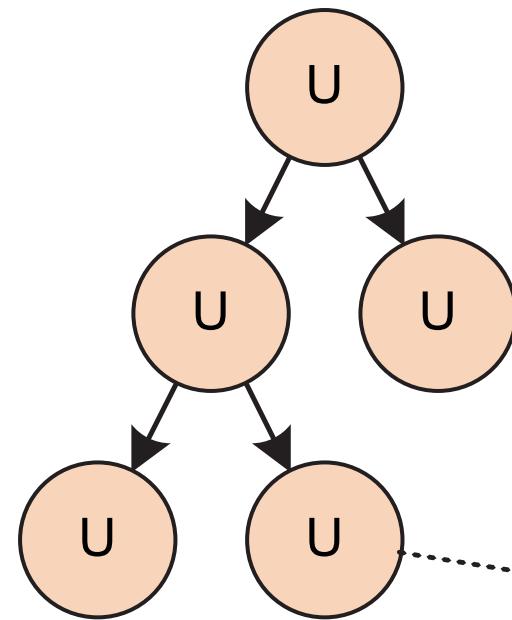
load_local x
load_local y
load_local z
call *
call +

pushq %rbp
movq %rsp, %rbp
movq %rdi, -8(%rbp)
movq %rsi, -16(%rbp)
movq %rdx, -24(%rbp)
movq -16(%rbp), %rax
movl %eax, %edx
movq -24(%rbp), %rax
imull %edx, %eax
movq -8(%rbp), %rdx
addl %edx, %eax
popq %rbp
ret



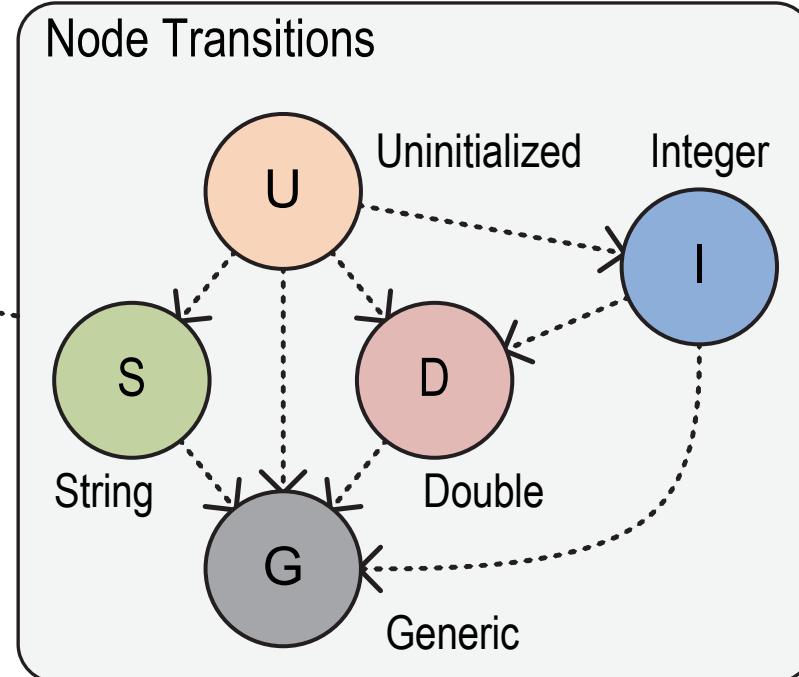
AST Interpreter
Uninitialized Nodes

T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.

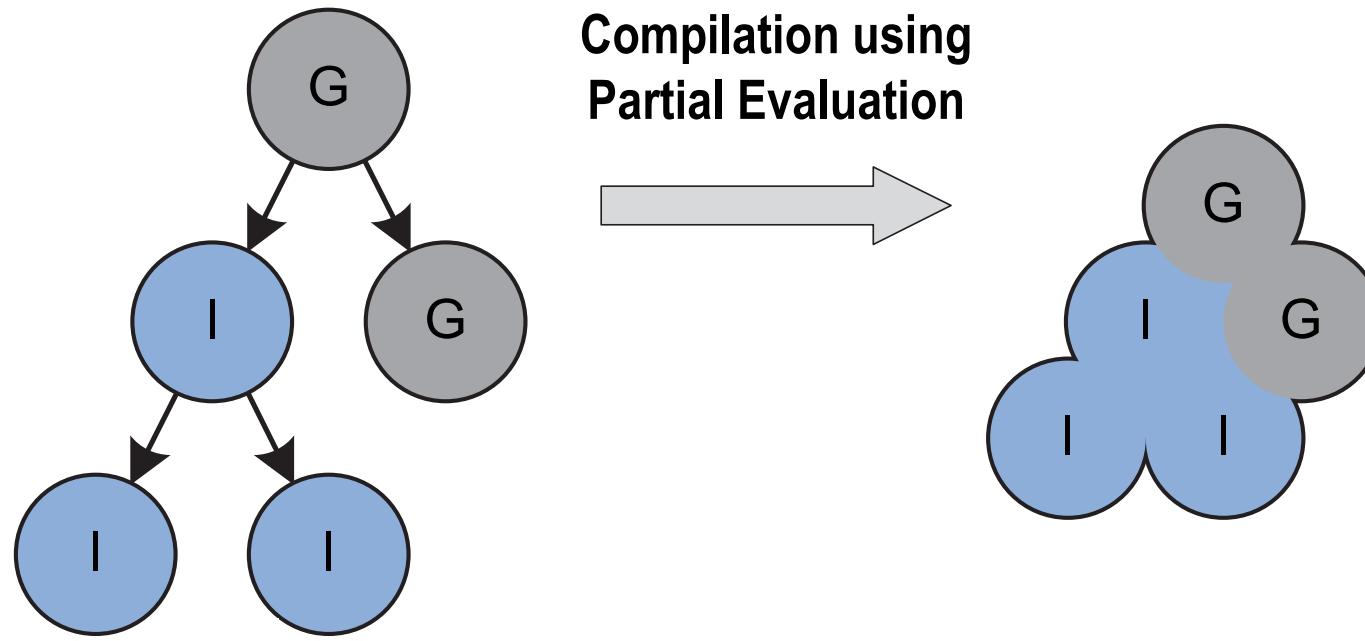


AST Interpreter
Uninitialized Nodes

Node Rewriting for Profiling Feedback



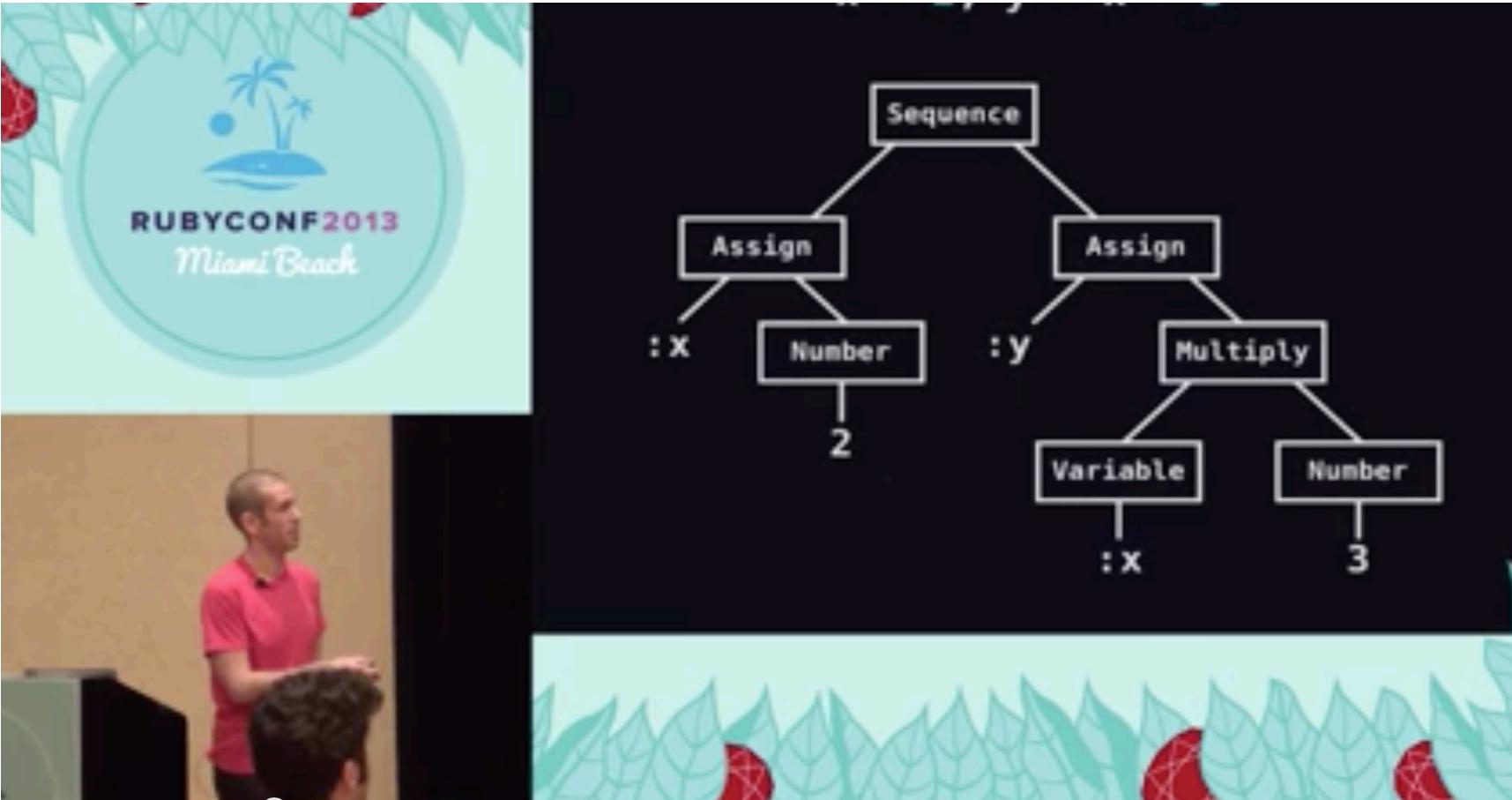
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.



AST Interpreter
Rewritten Nodes

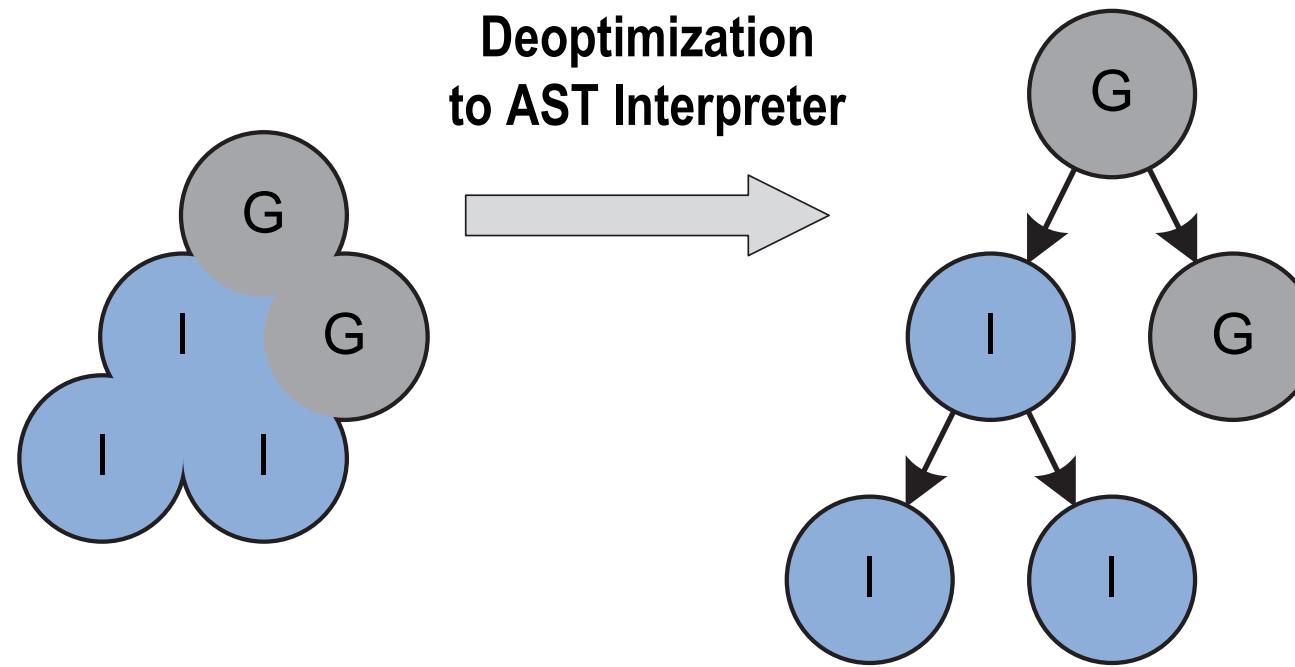
Compiled Code

T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.



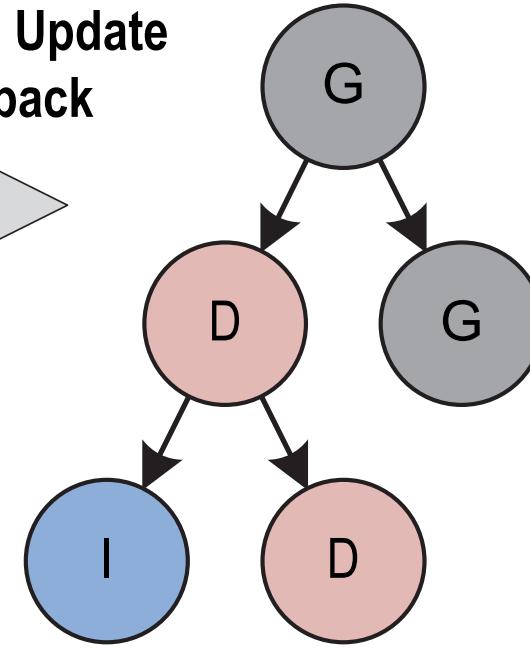
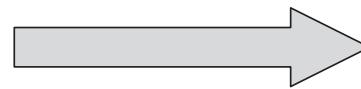
codon.com/compilers-for-free

Presentation, by Tom Stuart, licensed under a Creative Commons Attribution ShareAlike 3.0

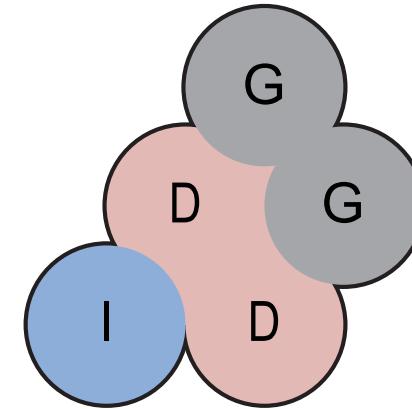
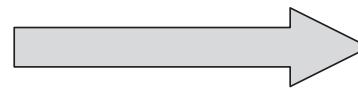


T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.

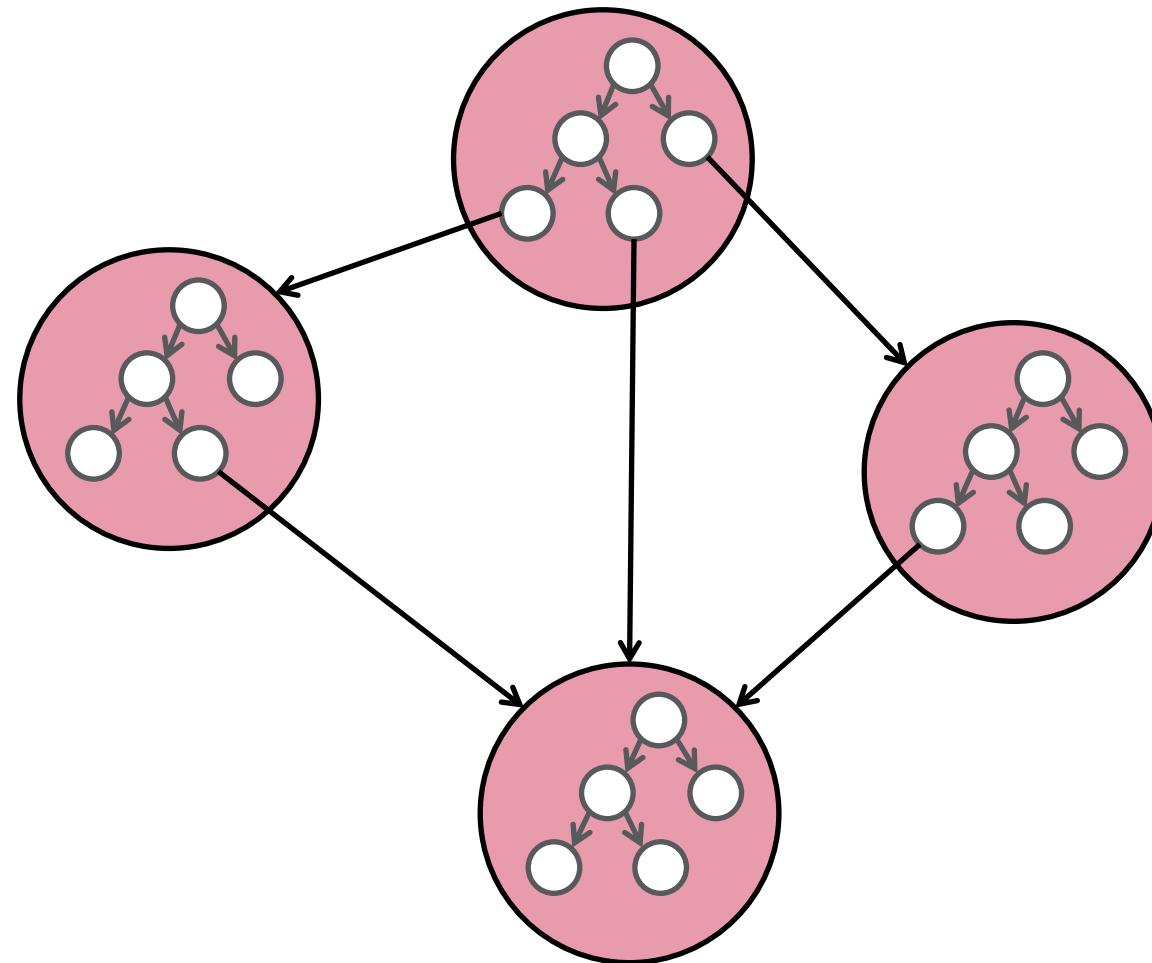
Node Rewriting to Update Profiling Feedback

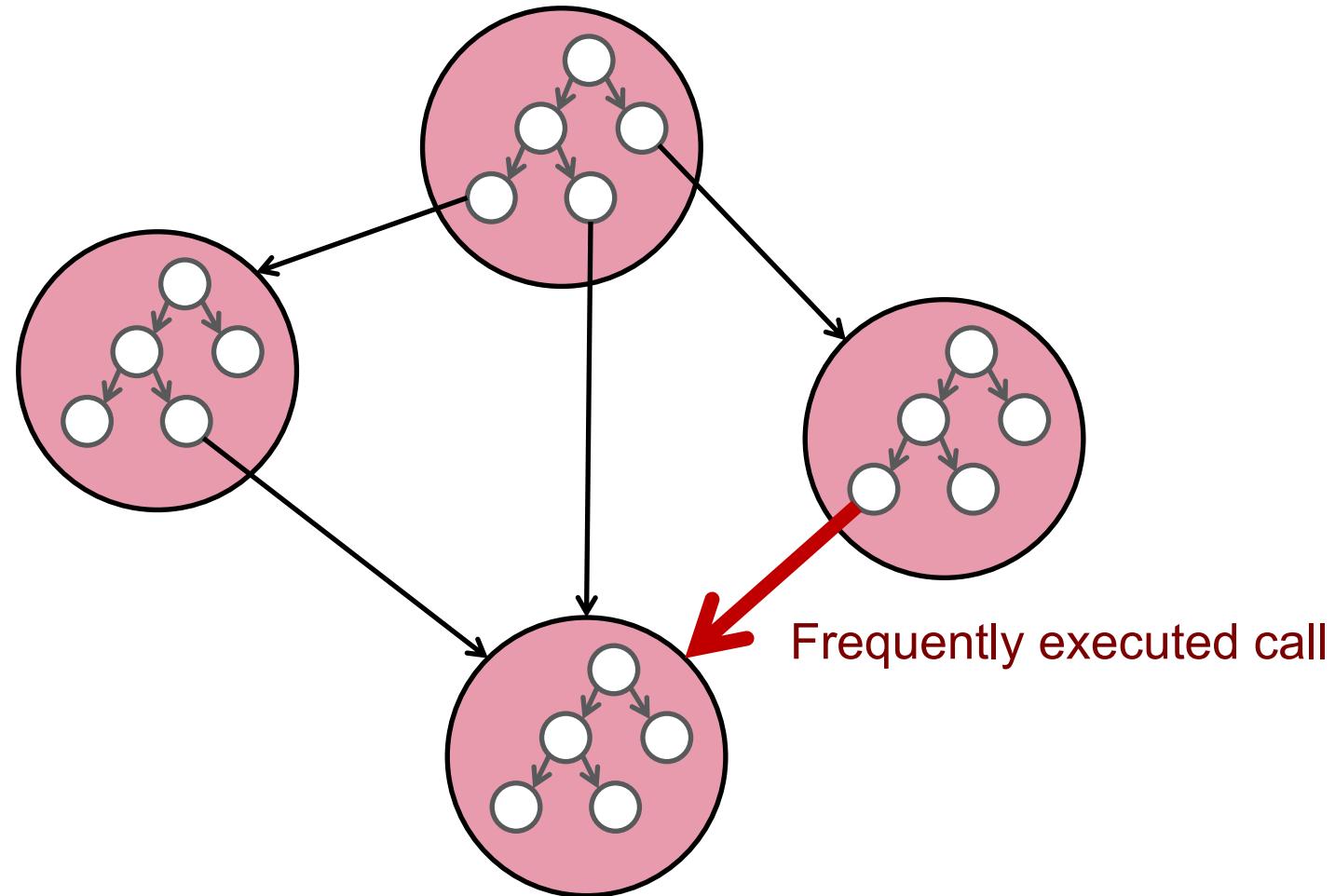


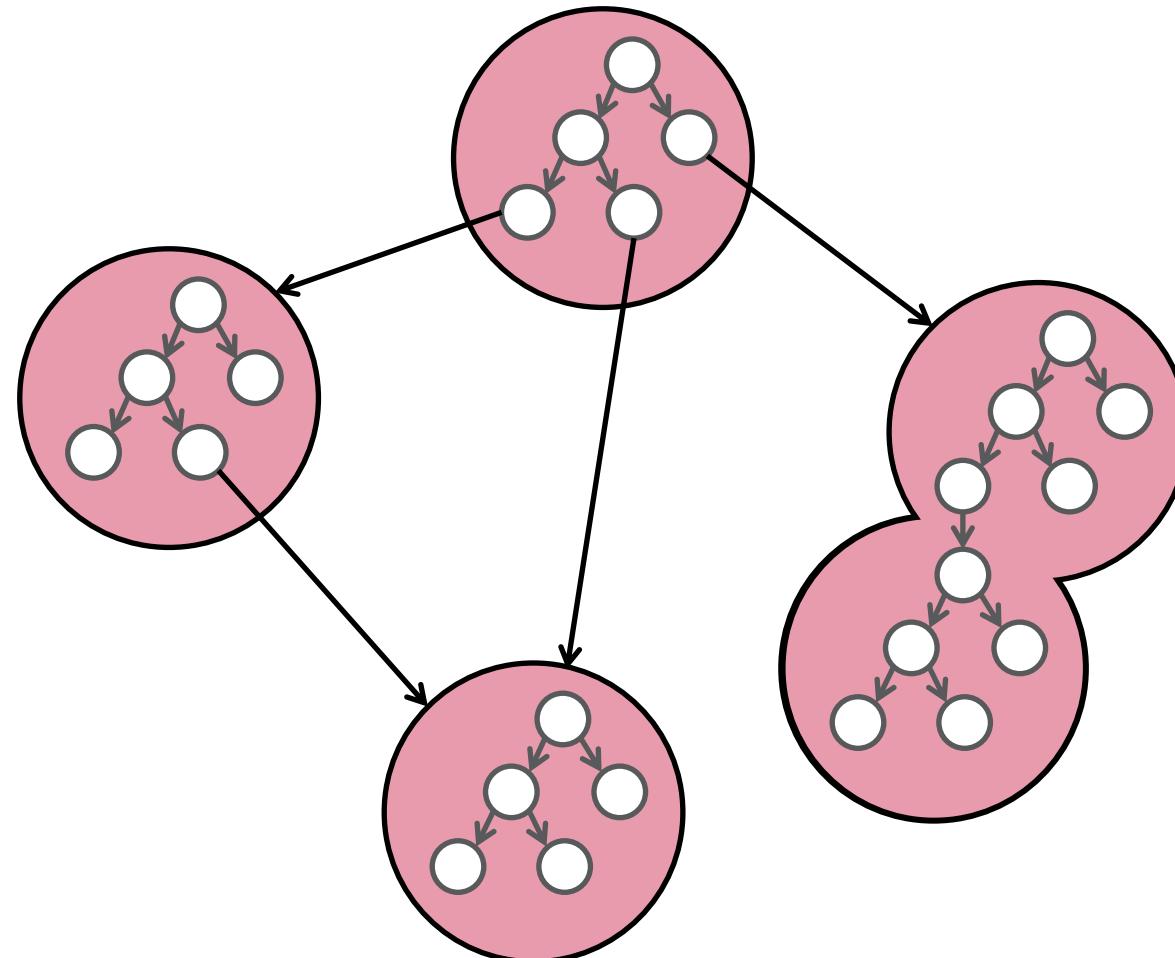
Recompilation using Partial Evaluation

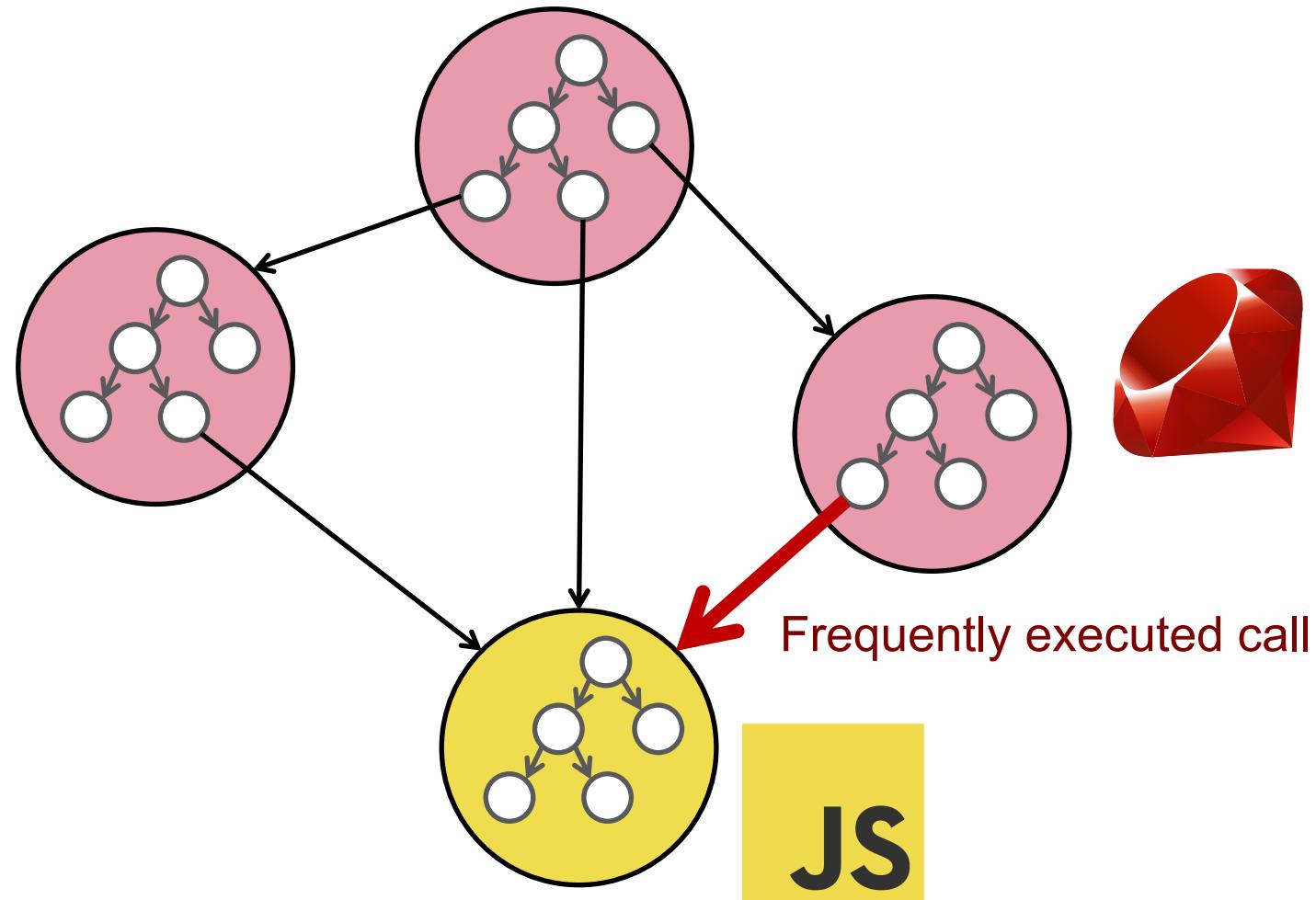


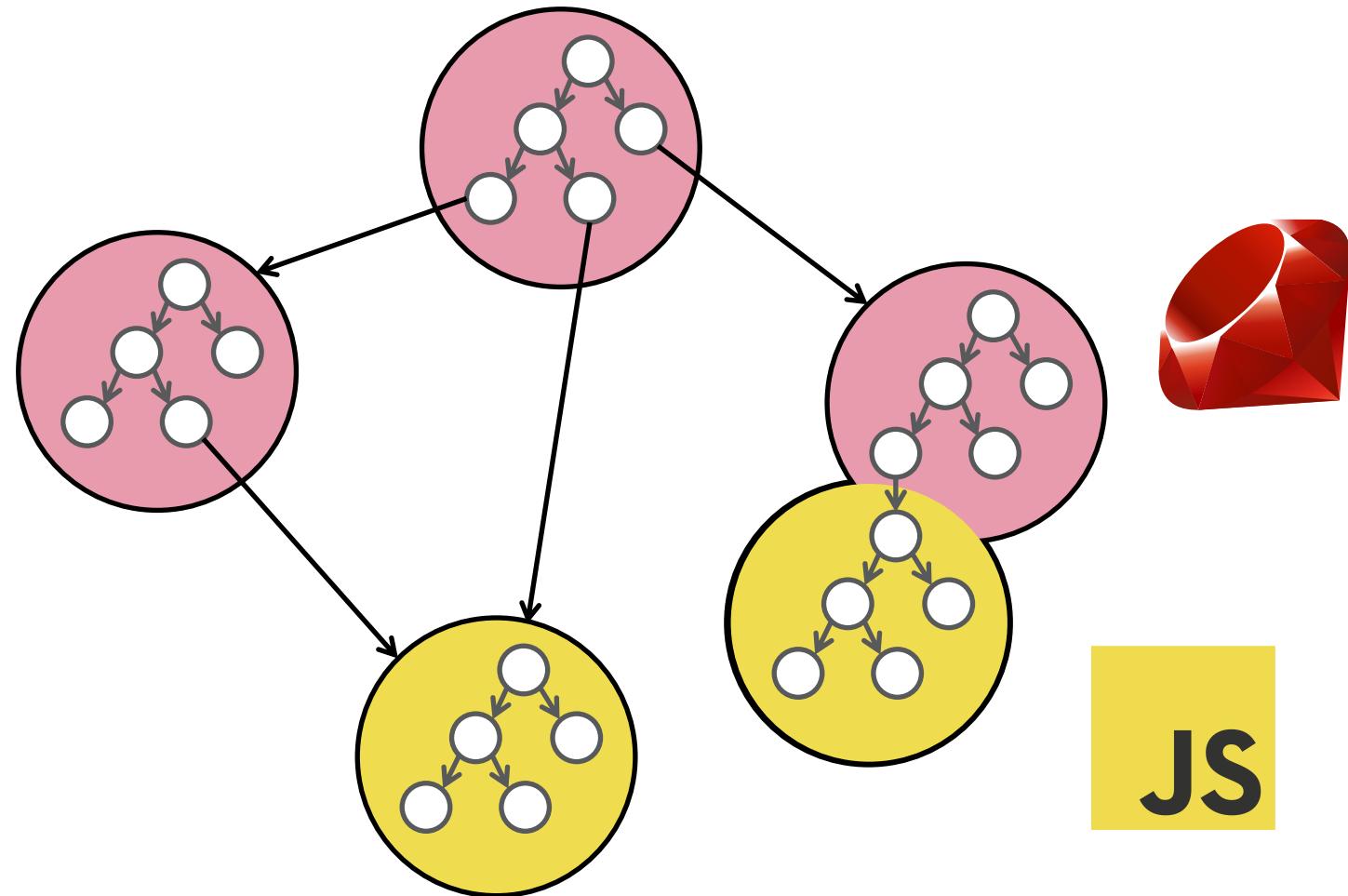
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.

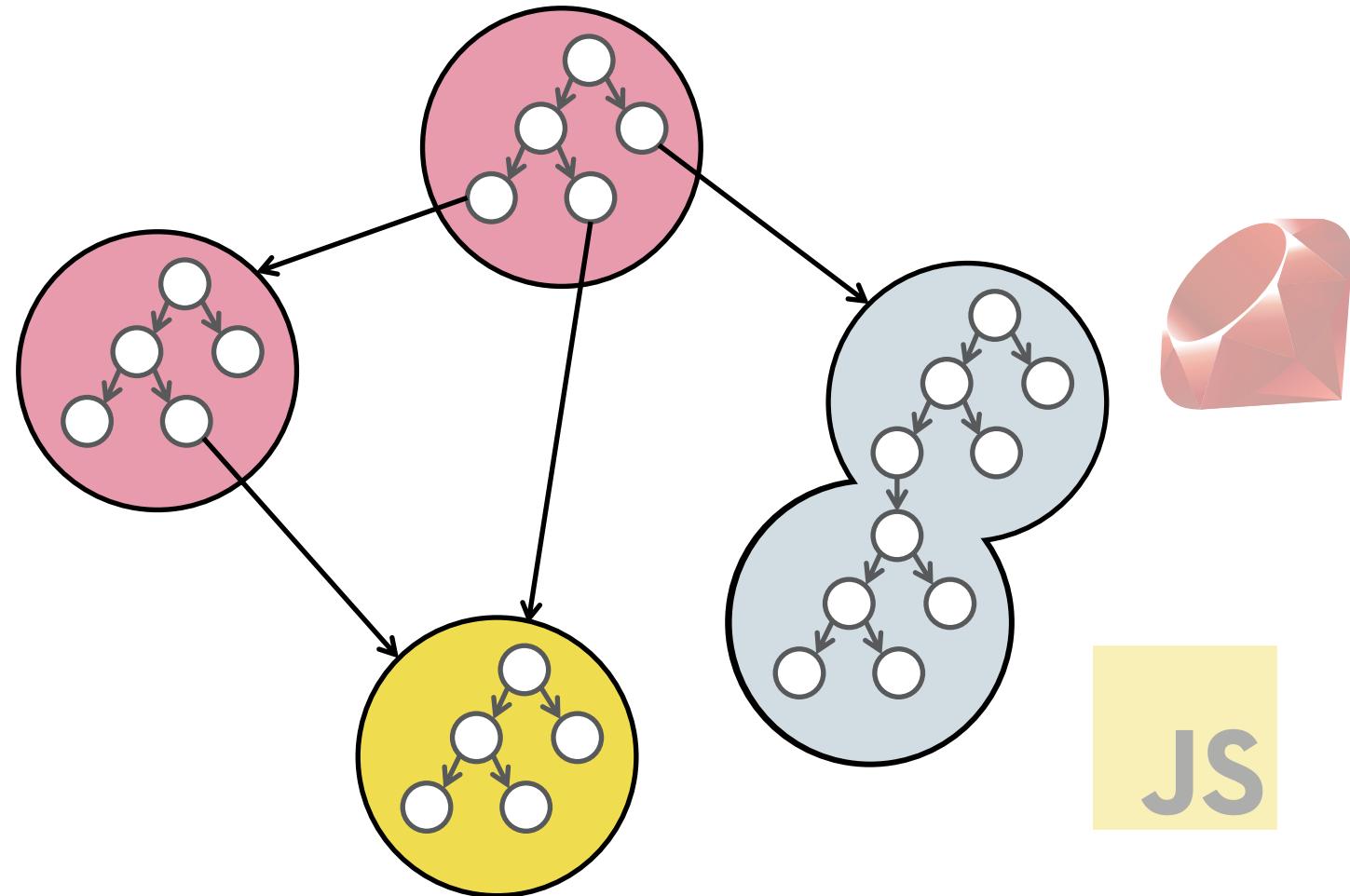












How effective is this in the extreme?

```
def sum(n)
  i = 0
  a = 0
  while i < n
    i += 1
    a += n
  end
  a
end

values = (1..100).to_a

loop do
  values.each do |v|
    sum(v)
  end
end
```

```
function sum(n) {
  var i = 0;
  var a = 0;
  while (i < n) {
    i += 1;
    a += n;
  }
  return a;
}

values = (1..100).to_a

loop do
  values.each do |v|
    sum(v)
  end
end
```

```
def sum(n)
  i = 0
  a = 0
  while i < n
    i += 1
    a += n
  end
  a
end
```

```
values = (1..100).to_a
```

```
loop do
  values.each do |v|
    sum(v)
  end
end
```

Looking at these loops here

```
function sum(n) {
  var i = 0;
  var a = 0;
  while (i < n) {
    i += 1;
    a += n;
  }
  return a;
}
```

```
values = (1..100).to_a
```

```
loop do
  values.each do |v|
    sum(v)
  end
end
```

```
def sum(n)
  i = 0
  a = 0
  while i < n
    i += 1
```

```
0x00000001118dfa30: mov    esi,edi
0x00000001118dfa32: add    esi,r9d
0x00000001118dfa35: jo     0x00000001118dfb62
0x00000001118dfa3b: inc    ecx
0x00000001118dfa3d: mov    edi,esi
0x00000001118dfa3f: cmp    r9d,ecx
0x00000001118dfa42: jg    0x00000001118dfa30
```

```
loop do
  values.each do |v|
    sum(v)
  end
end
```

```
function sum(n) {
  var i = 0;
  var a = 0;
  while (i < n) {
    i += 1;
```

```
0x000000010ca4ad90: mov    eax,r11d
0x000000010ca4ad93: add    eax,r14d
0x000000010ca4ad96: jo     0x000000010ca4ae68
0x000000010ca4ad9c: inc    r10d
0x000000010ca4ad9f: mov    r11d,eax
0x000000010ca4ada2: cmp    r14d,r10d
0x000000010ca4ada5: jg    0x000000010ca4ad90
```

```
loop do
  values.each do |v|
    sum(v)
  end
end
```

```
def add(a, b)
  a + b
end

def sum(n)
  i = 0
  a = 0
  while i < n
    i += 1
    a = add(a, n)
  end
  a
end
```

```
function add(a, b) {
  return a + b;
}

def sum(n)
  i = 0
  a = 0
  while i < n
    i += 1
    a = add(a, n)
  end
  a
end
```

```
def add(a, b)
  a + b
end
```

```
0x0000000103a7dc70: mov    esi,edi
0x0000000103a7dc72: add    esi,r9d
0x0000000103a7dc75: jo     0x0000000103a7dda2
0x0000000103a7dc7b: inc    ecx
0x0000000103a7dc7d: mov    edi,esi
0x0000000103a7dc7f: cmp    r9d,ecx
0x0000000103a7dc82: jg     0x0000000103a7dc70
```

```
a = add(a, n)
end
a
end
```

```
function add(a, b) {
  return a + b;
}
```

```
0x000000010aadb1f0: mov    esi,edi
0x000000010aadb1f2: add    esi,r9d
0x000000010aadb1f5: jo     0x000000010aadb322
0x000000010aadb1fb: inc    ecx
0x000000010aadb1fd: mov    edi,esi
0x000000010aadb1ff: cmp    r9d,ecx
0x000000010aadb202: jg     0x000000010aadb1f0
```

```
a = add(a, n)
end
a
end
```

```
def add(a, b)
    a + b
end
```

```
0x0000000103a7dc70: mov    esi,edi
0x0000000103a7dc72: add    esi,r9d
0x0000000103a7dc75: jo     0x0000000103a7dda2
0x0000000103a7dc7b: inc    ecx
0x0000000103a7dc7d: mov    edi,esi
0x0000000103a7dc7f: cmp    r9d,ecx
0x0000000103a7dc82: jg    0x0000000103a7dc70
```

```
end
```

```
a
```

```
end
```

```
function add(a, b) {
    return a + b;
}
```

```
1
```

```
esi,edi
esi,r9d
0x00000010aadb322
ecx
edi,esi
r9d,ecx
0x00000010aadb1f0
```

```
(a, n)
```

```
end
```

```
a
```

```
end
```

```
def add(a, b)
    a + b
end
```

```
0x0000000103a7dc70: mov    esi,edi
0x0000000103a7dc72: add    esi,r9d
0x0000000103a7dc75: jo     0x0000000103a7dda2
0x0000000103a7dc7b: inc    ecx
0x0000000103a7dc7d: mov    edi,esi
0x0000000103a7dc7f: cmp    r9d,ecx
0x0000000103a7dc82: jg    0x0000000103a7dc70
```

```
end
```

```
a
```

```
end
```

```
function add(a, b) {
    return a + b;
}
```

```
1
```

```
esi,edi
esi,r9d
0x00000010aadb322
ecx
edi,esi
r9d,ecx
0x00000010aadb1f0
```

```
(a, n)
```

```
end
```

```
a
```

```
end
```

What is this for?

- We're not really suggesting that people routinely write alternate methods in different languages

- We're not really suggesting that people routinely write alternate methods in different languages
- More about removing the consideration of performance from the decision if you do want to combine languages

- Could make all library ecosystems available to all applications
- May be useful for unifying a front-end and back-end
- May be useful in handling legacy applications and incremental changes in implementation language

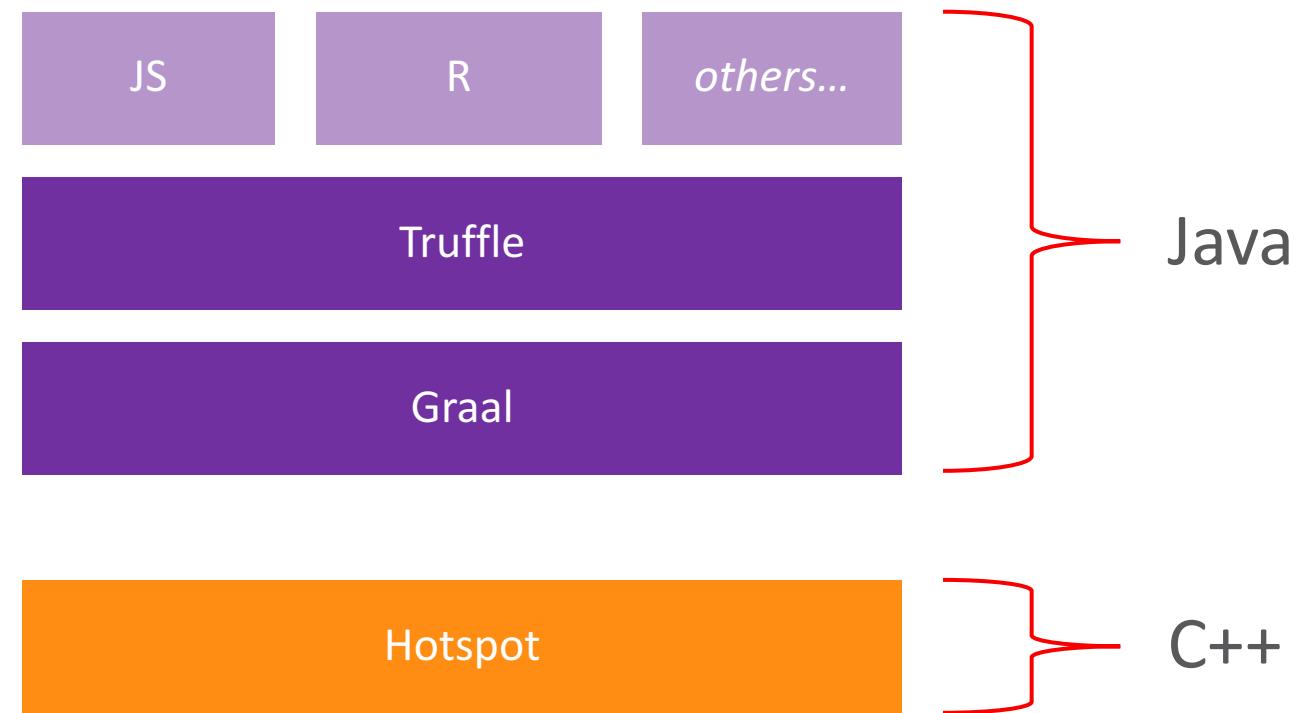
How to use GraalVM

GraalVM – everything in one package today

- Includes:
 - JVM (RE or DK)
 - Java
 - JavaScript
 - Ruby
 - R
 - More in the future
- Binary tarball release
- Mac or Linux

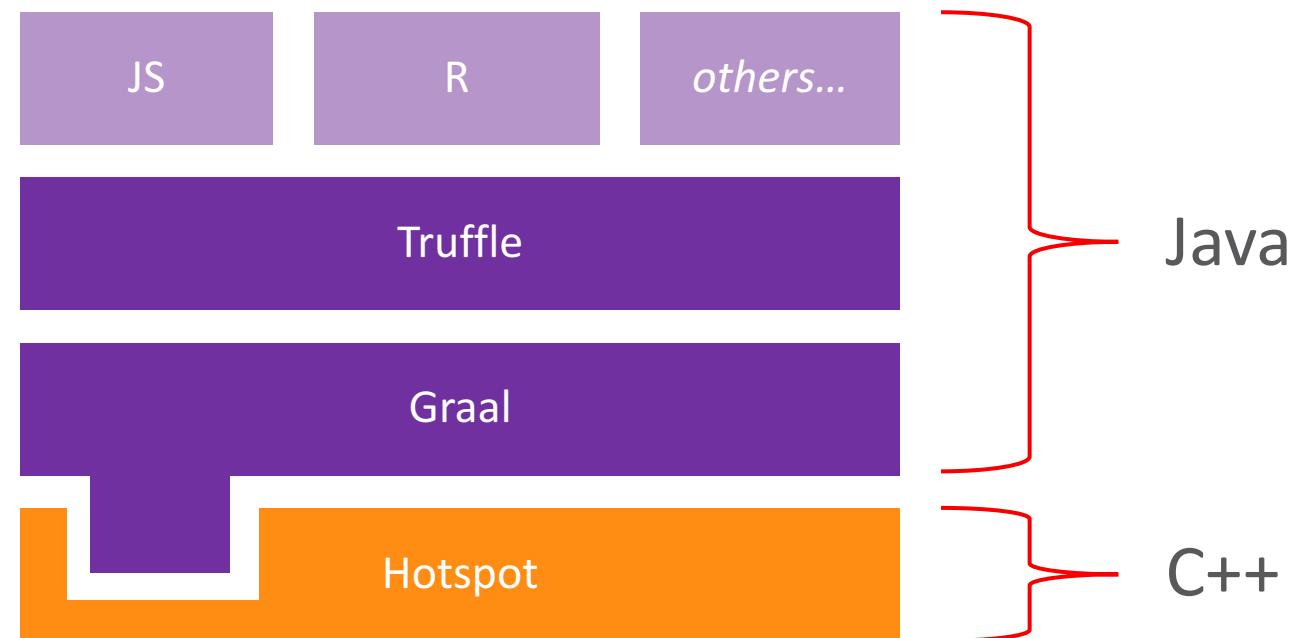


Java 9 – runs on an unmodified JVM

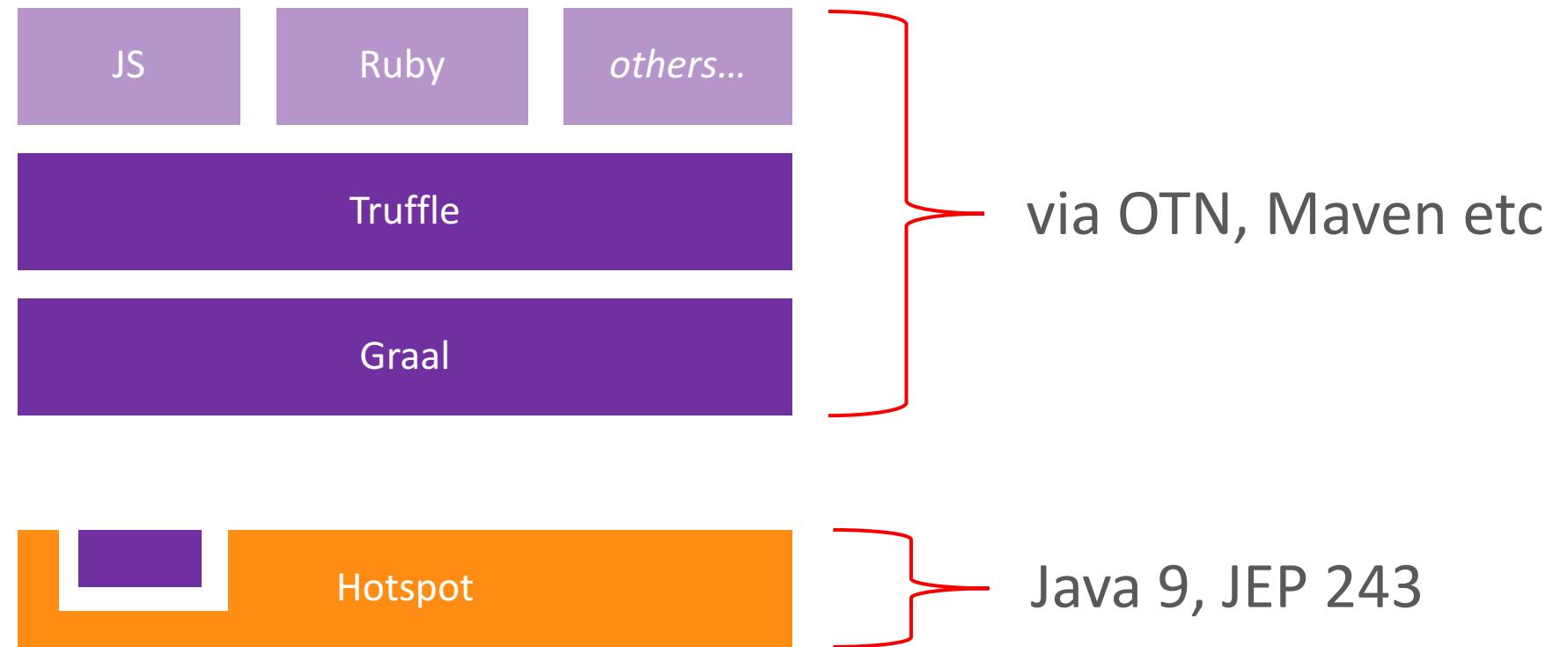


Java 9 – runs on an unmodified JVM

JVMCI
(JVM Compiler Interface)



Java 9 – runs on an unmodified JVM



Takeaways

- Oracle Labs is building Graal VM to support polyglot programs and programmers
- Extremely high performance for the languages on their own
- Completely unprecedented high performance for language interoperability
- Will work on an unmodified Java 9 JVM, or available as a bundle today
- Still at the research stage, but moving towards being something more than that

Where to find more information

Welcome Chris

Account Sign Out Help Country ▾ Communities ▾ I am a... ▾ I want to... ▾

Search

Products Solutions Downloads Store Support Training Partners About OTN

Oracle Technology Network > Oracle Labs > Programming Languages and Runtimes > Downloads

Parallel Graph Analytics

Programming Languages and Runtimes

Souffle

Datasets

Overview Java Polyglot **Downloads** Learn More

Oracle Labs GraalVM and JVMCI JDK Downloads

Thank you for downloading this release of the Oracle Labs GraalVM. With this release, one can execute Java applications with Graal, as well as applications written in JavaScript, Ruby, and R, with our Polyglot language engines.

You must accept the OTN License Agreement to download this software.

Accept License Agreement | Decline License Agreement

GraalVM preview for Linux (v0.16), Development Kit

GraalVM preview for Linux (v0.16), Runtime Environment

GraalVM preview for Mac OS X (v0.16), Development Kit

GraalVM preview for Mac OS X (v0.16), Runtime Environment

labsjdk-8u92-jvmci-0.20-darwin-amd64.tar.gz

labsjdk-8u92-jvmci-0.20-linux-amd64.tar.gz

labsjdk-8u92-jvmci-0.20-solaris-sparcv9.tar.gz

How to install GraalVM

Unpack the downloaded *.tar.gz file on your machine. You can then use the java executable to execute Java programs. All those executables are in the bin directory of GraalVM. You might want to add that directory to your operating system's PATH.

More detailed getting started instructions are available in the README files in the download. The README files for the language engines can be found in jre/lang

www.oracle.com/technetwork/oracle-labs/program-languages

 This organization Search

Pull requests Issues Gist

GitHub, Inc.

Search for 'github graalvm'

<https://graalvm.github.io>

Graal Multi-Language VM
Next generation compilation technology supporting Java, Ruby, R, JavaScript, LLVM, and more.

Repositories People 38 Teams 2

Filters Find a repository...

sulong Java ★ 211 ⚡ 19
Sulong, a dynamic runtime for LLVM-based languages.
Updated 6 minutes ago

graal-core Java ★ 122 ⚡ 33
Graal Compiler & Truffle Partial evaluator.
Updated 31 minutes ago

mx Python ★ 13 ⚡ 26

People 38 >



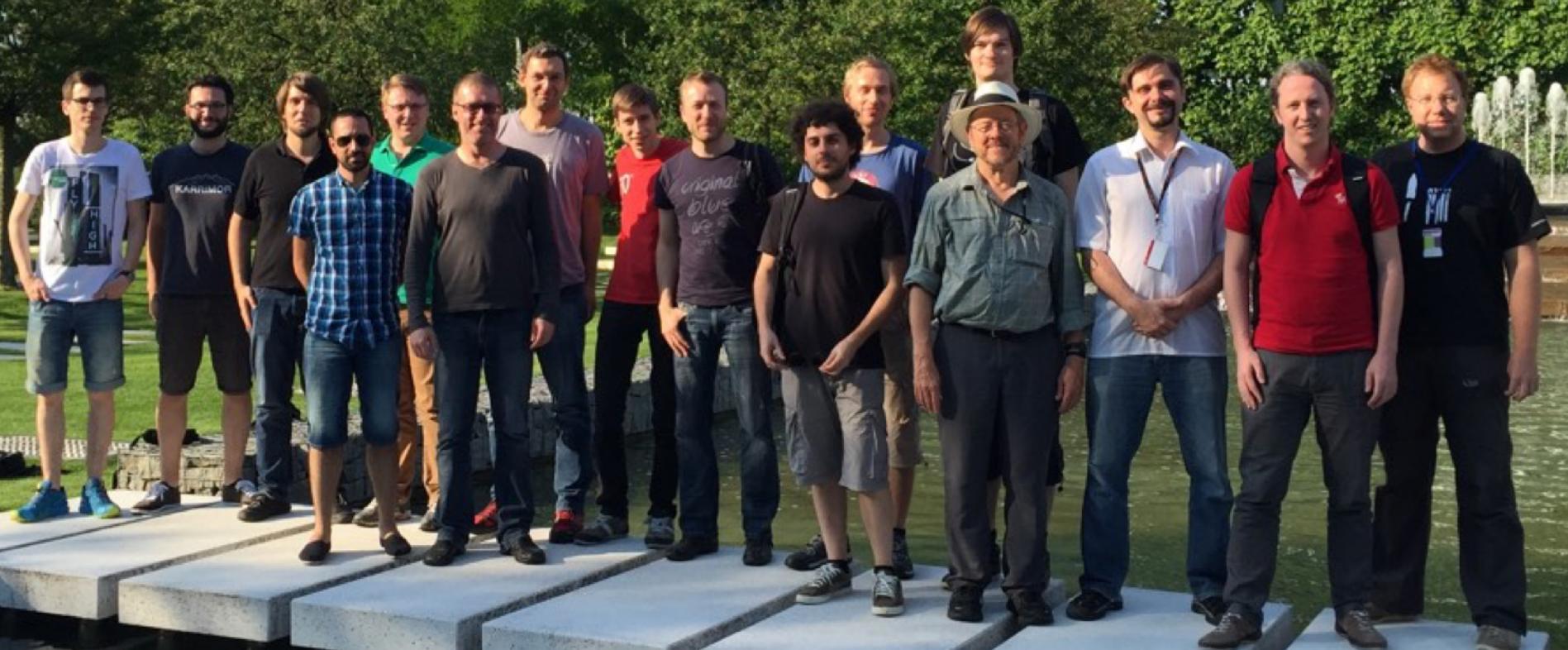
github.com/graalvm

@chrisgseaton

github.com/graalvm

gitter.im/graalvm/graal-core

Search ‘otn graalvm’



Acknowledgements

Oracle

Danilo Ansaloni
Stefan Anzinger
Cosmin Basca
Daniele Bonetta
Matthias Brantner
Petr Chalupa
Jürgen Christ
Laurent Daynès
Gilles Duboscq
Martin Entlicher
Brandon Fish
Bastian Hossbach
Christian Hummer
Mick Jordan
Vojin Jovanovic
Peter Kessler
David Leopoldseder
Kevin Menard
Jakub Podlešák
Aleksandar Prokopec
Tom Rodriguez

Oracle (continued)

Roland Schatz
Chris Seaton
Doug Simon
Štěpán Šindelář
Zbyněk Šlajchrt
Lukas Stadler
Codrut Stancu
Jan Štola
Jaroslav Tulach
Michael Van De Vanter
Adam Welc
Christian Wimmer
Christian Wirth
Paul Wögerer
Mario Wolczko
Andreas Wöß
Thomas Würthinger

Oracle Interns

Brian Belleville
Miguel Garcia
Shams Imam
Alexey Karyakin
Stephen Kell
Andreas Kunft
Volker Lanting
Gero Leinemann
Julian Lettner
Joe Nash
David Piorkowski
Gregor Richards
Robert Seilbeck
Rifat Shariyar

Alumni

Erik Eckstein
Michael Haupt
Christos Kotselidis
Hyunjin Lee
David Leibs
Chris Thalinger
Till Westmann

JKU Linz

Prof. Hanspeter Mössenböck
Benoit Daloz
Josef Eisl
Thomas Feichtinger
Matthias Grimmer
Christian Häubl
Josef Haider
Christian Huber
Stefan Marr
Manuel Rigger
Stefan Rumzucker
Bernhard Urban

University of Edinburgh

Christophe Dubach
Juan José Fumero Alfonso
Ranjeet Singh
Toomas Remmelg

LaBRI

Floréal Morandat

University of California, Irvine

Prof. Michael Franz
Gulfem Savrun Yeniceri
Wei Zhang

Purdue University

Prof. Jan Vitek
Tomas Kalibera
Petr Maj
Lei Zhao

T. U. Dortmund

Prof. Peter Marwedel
Helena Kotthaus
Ingo Korb

University of California, Davis

Prof. Duncan Temple Lang
Nicholas Ulle

University of Lugano, Switzerland

Prof. Walter Binder
Sun Haiyang
Yudi Zheng

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®