ORACLE®

JavaOne™

ORACLE®

Faster Ruby and JavaScript with GraalVM

Chris Seaton
Research Manager
Oracle Labs
September 20, 2016

Java
Your
(Next)

Safe Harbor Statement

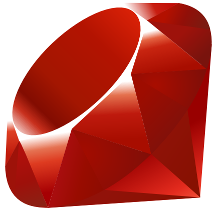
The following is intended to provide some insight into a line of research in Oracle Labs. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing 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.

The One VM Concept

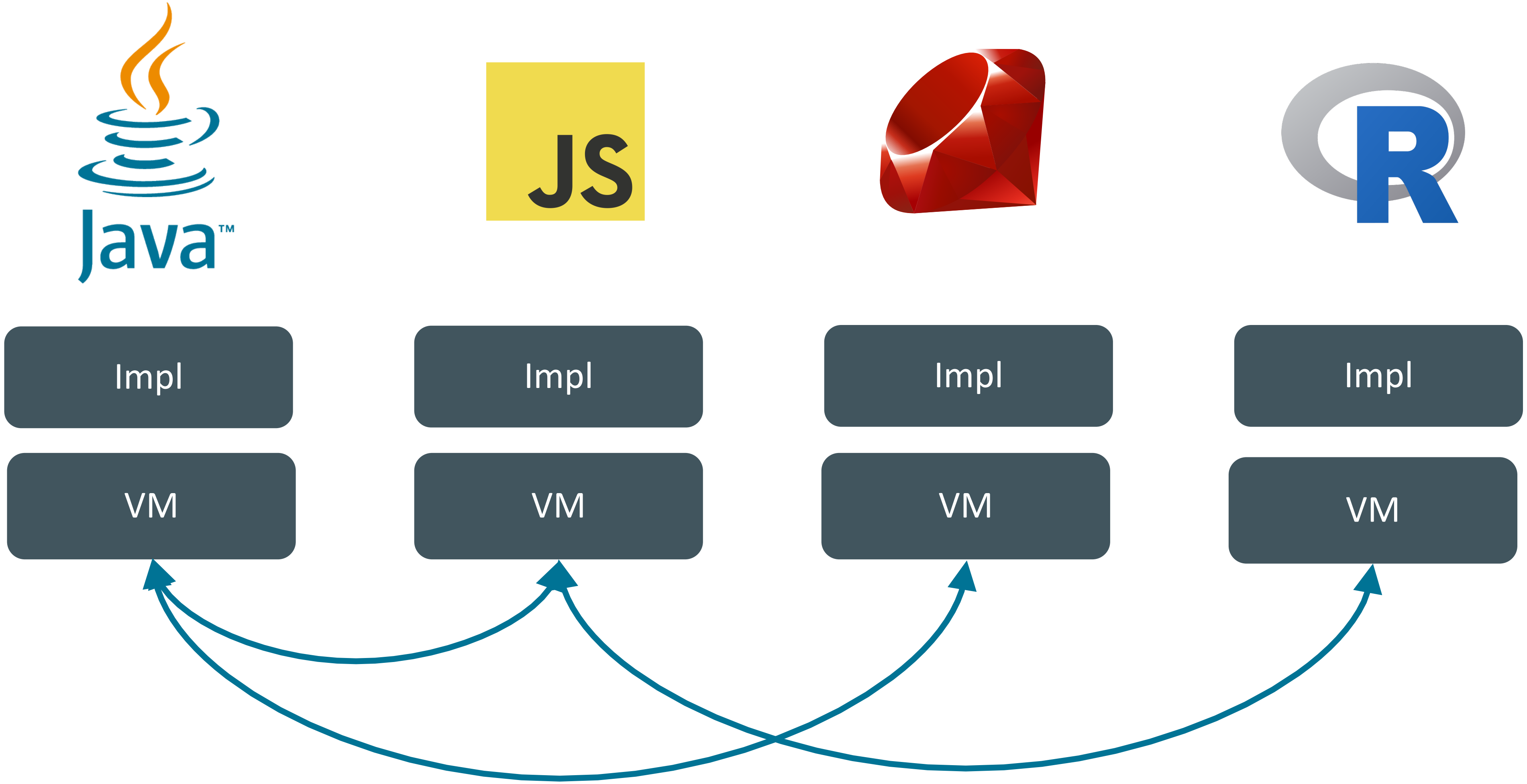
High performance polyglot virtual machine

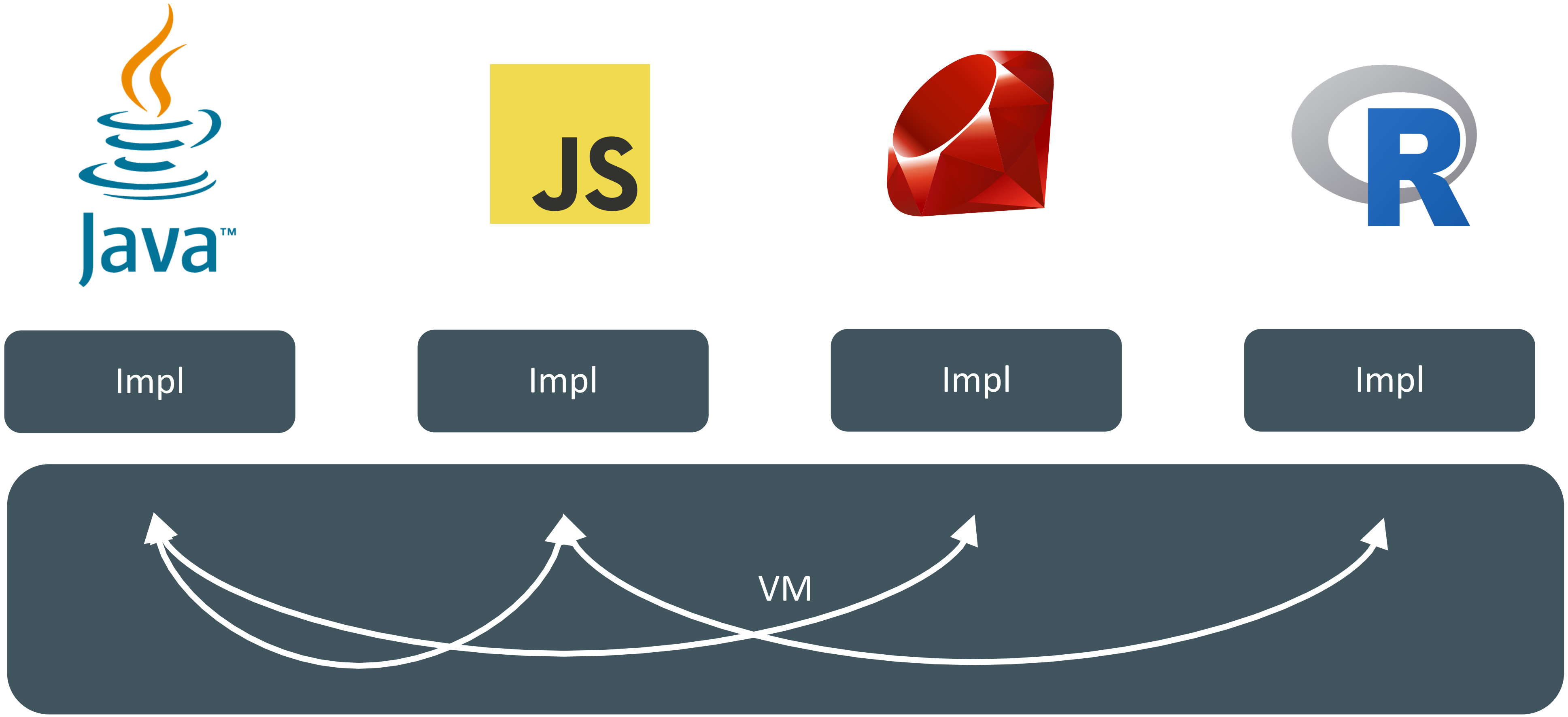
Java™

JS



R


Java™
JS
Impl
Impl
Impl
Impl
VM
VM
VM
VM



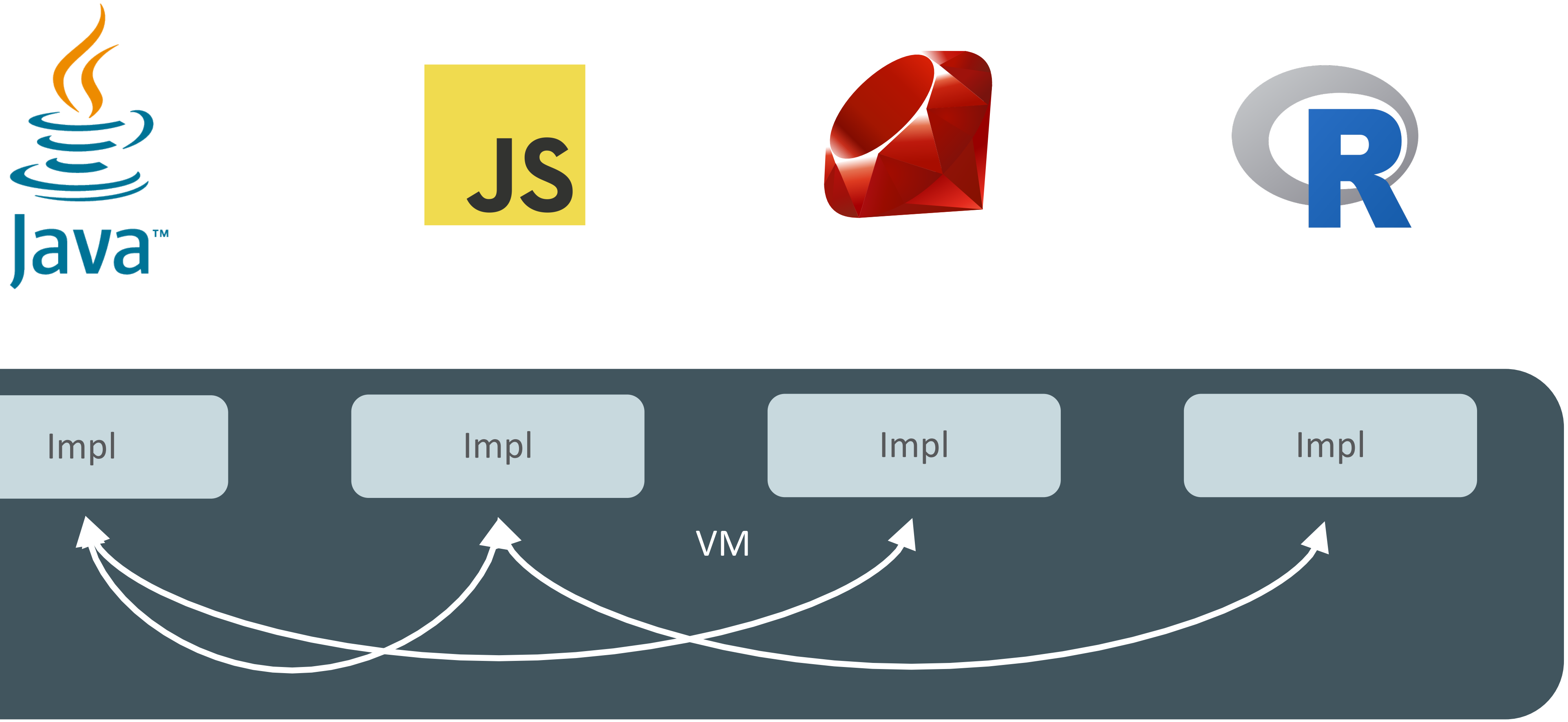
Impl

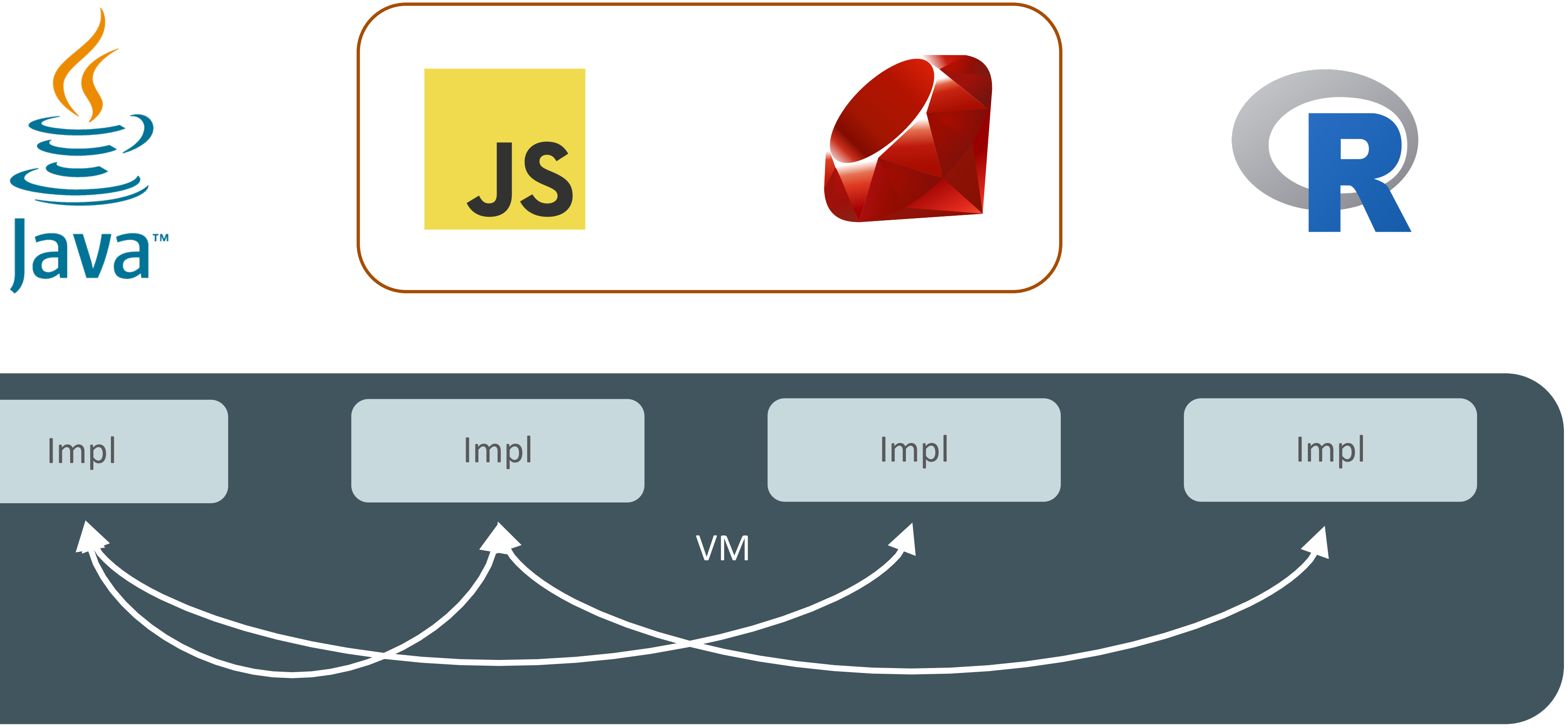
Impl

Impl

Impl

VM


Java
JS
R
Impl
Impl
Impl
Impl
VM



Impl

Impl

Impl

Impl

VM

JavaScript in GraalVM

Completeness

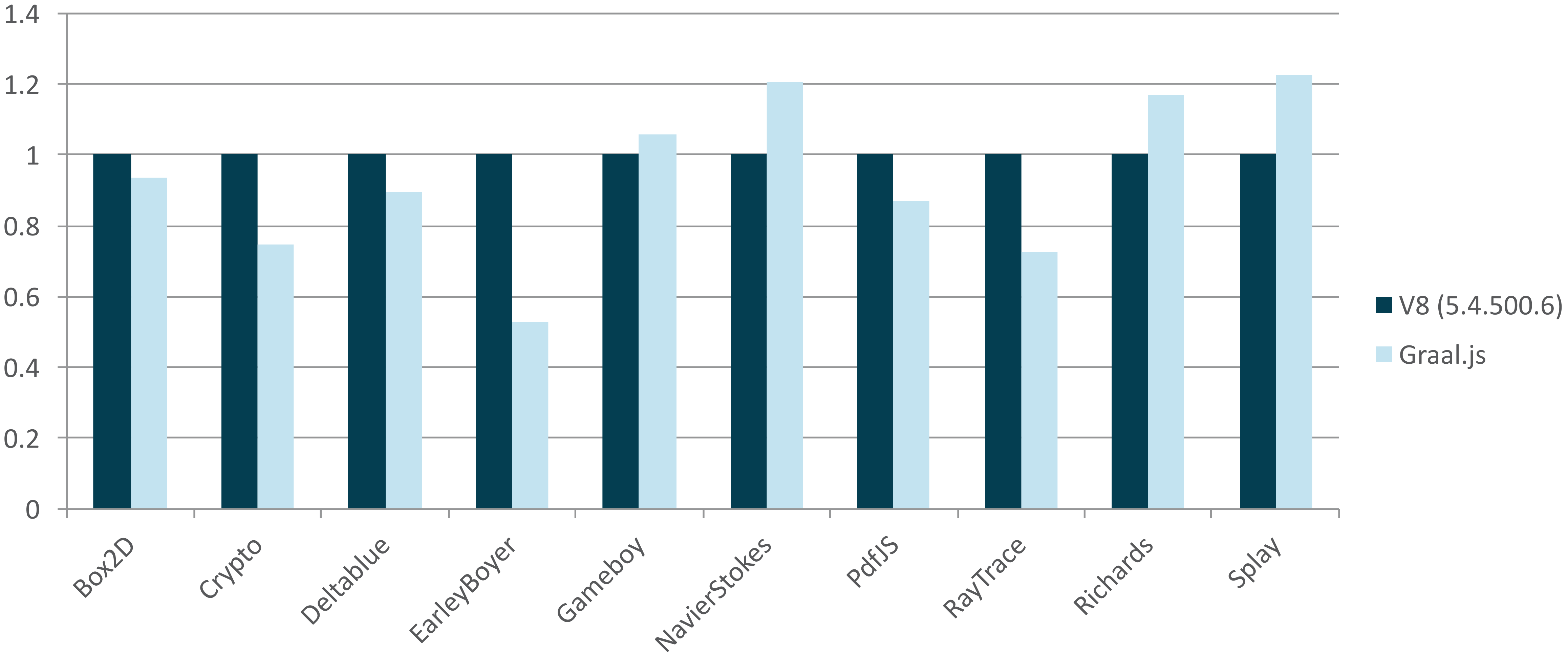
ECMAScript 2015 (ES6)

- Graal passes 99.3% (16298 of 16417 tests)
- Failing tests are to a large part Unicode Regular Expressions

ECMAScript 2016 (ES7)

- Graal passes 93.4% (20785 of 22260 tests)
- V8 (5.4.500.6) passes 91.1%
- Graal supports ES7 (exponentiation operator, Array.prototype.includes)
- Fails due to new block-level function declaration and corner-case tests of the spread operator

Classic research benchmarks – roughly level with V8



Ruby in GraalVM

Completeness – language and core library

99%

Ruby language

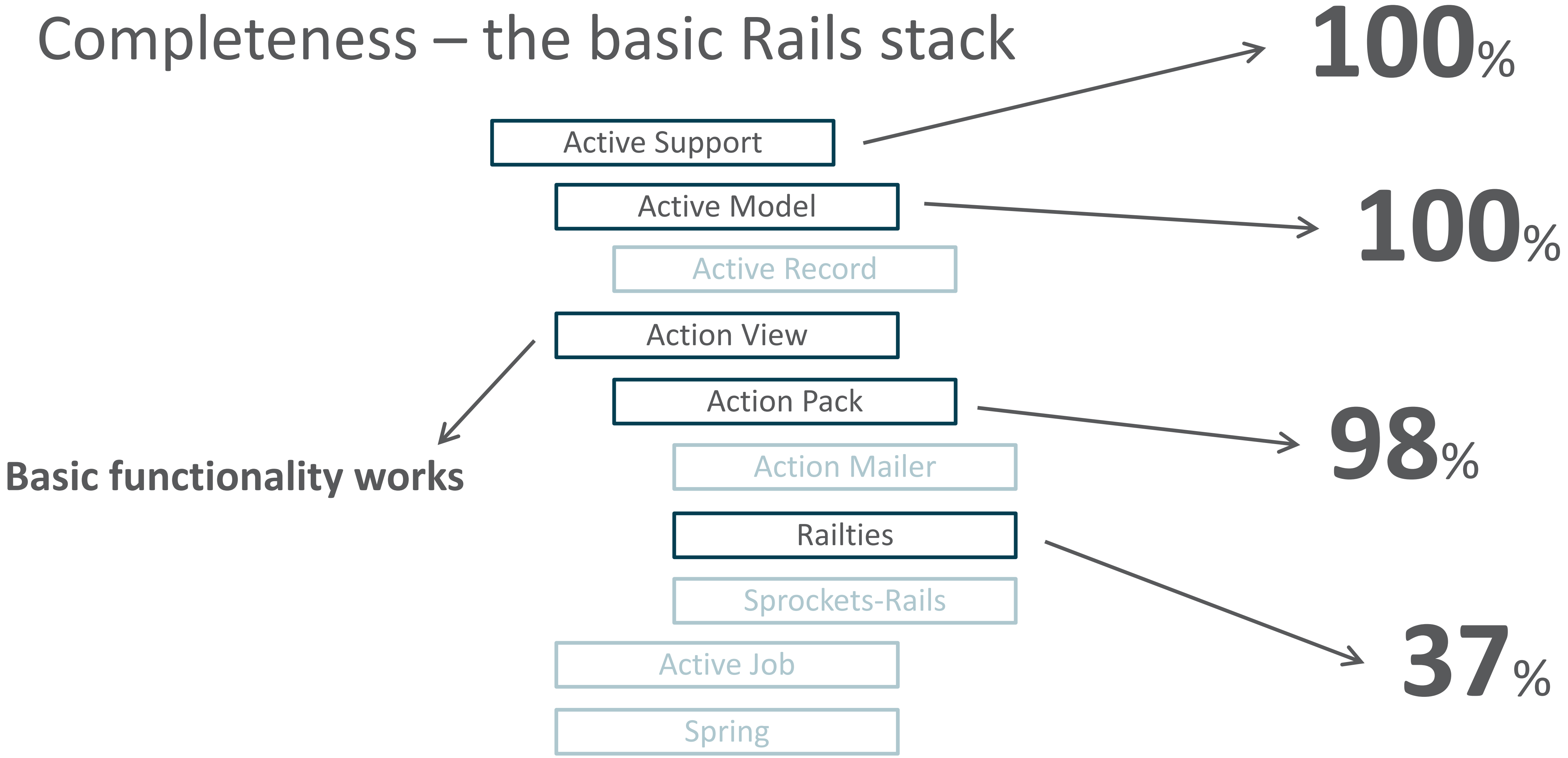
JRuby passes 94%

96%

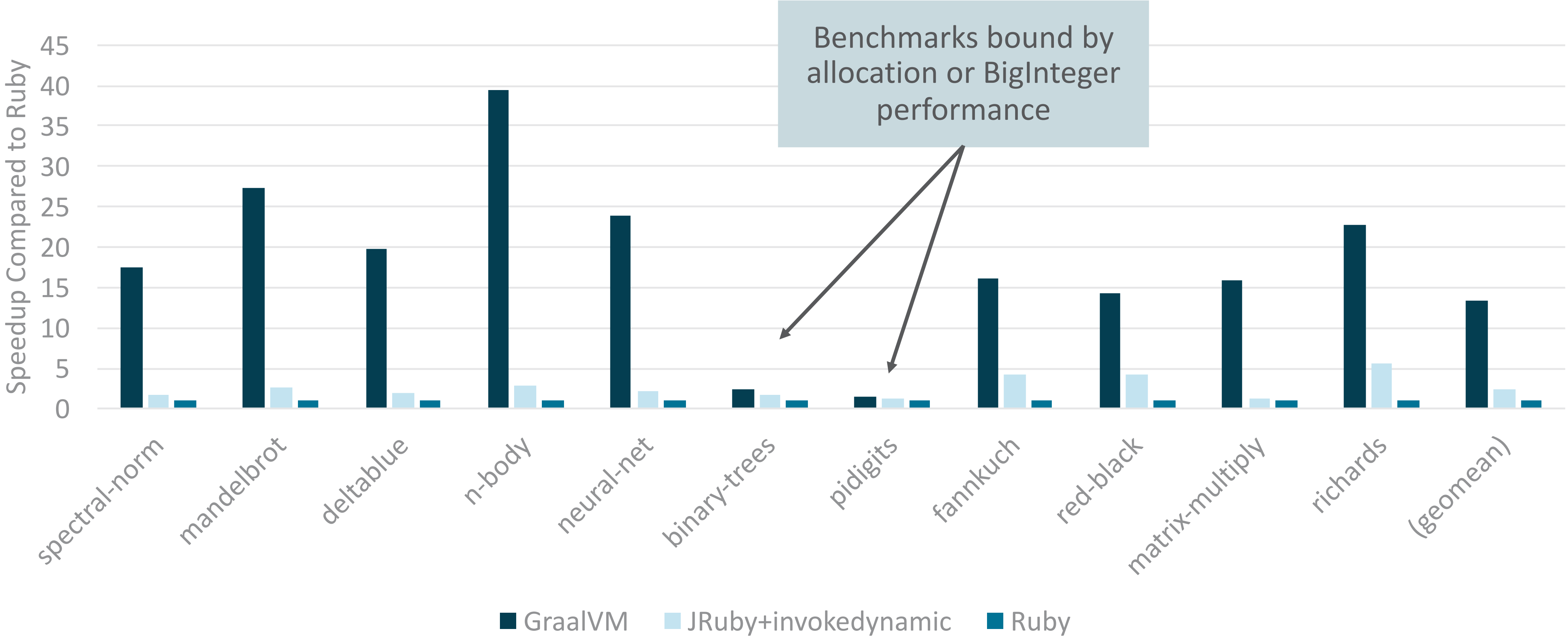
Ruby core library

JRuby passes 95%

Completeness – the basic Rails stack

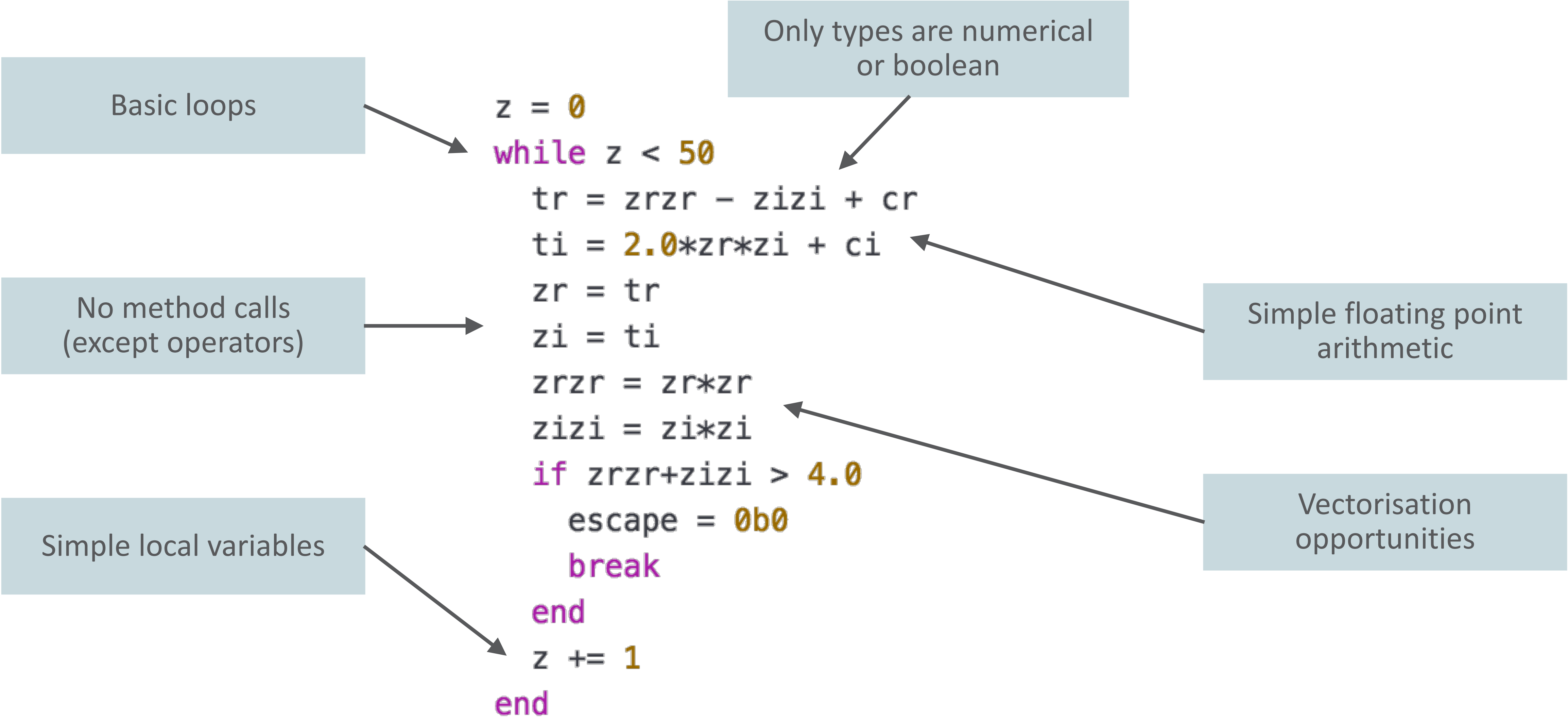


Classic research benchmarks – 10-20x faster



Benchmarks bound by allocation or BigInteger performance

‘But it’s easy to optimise that kind of code!’



Basic loops

No method calls (except operators)

Simple local variables

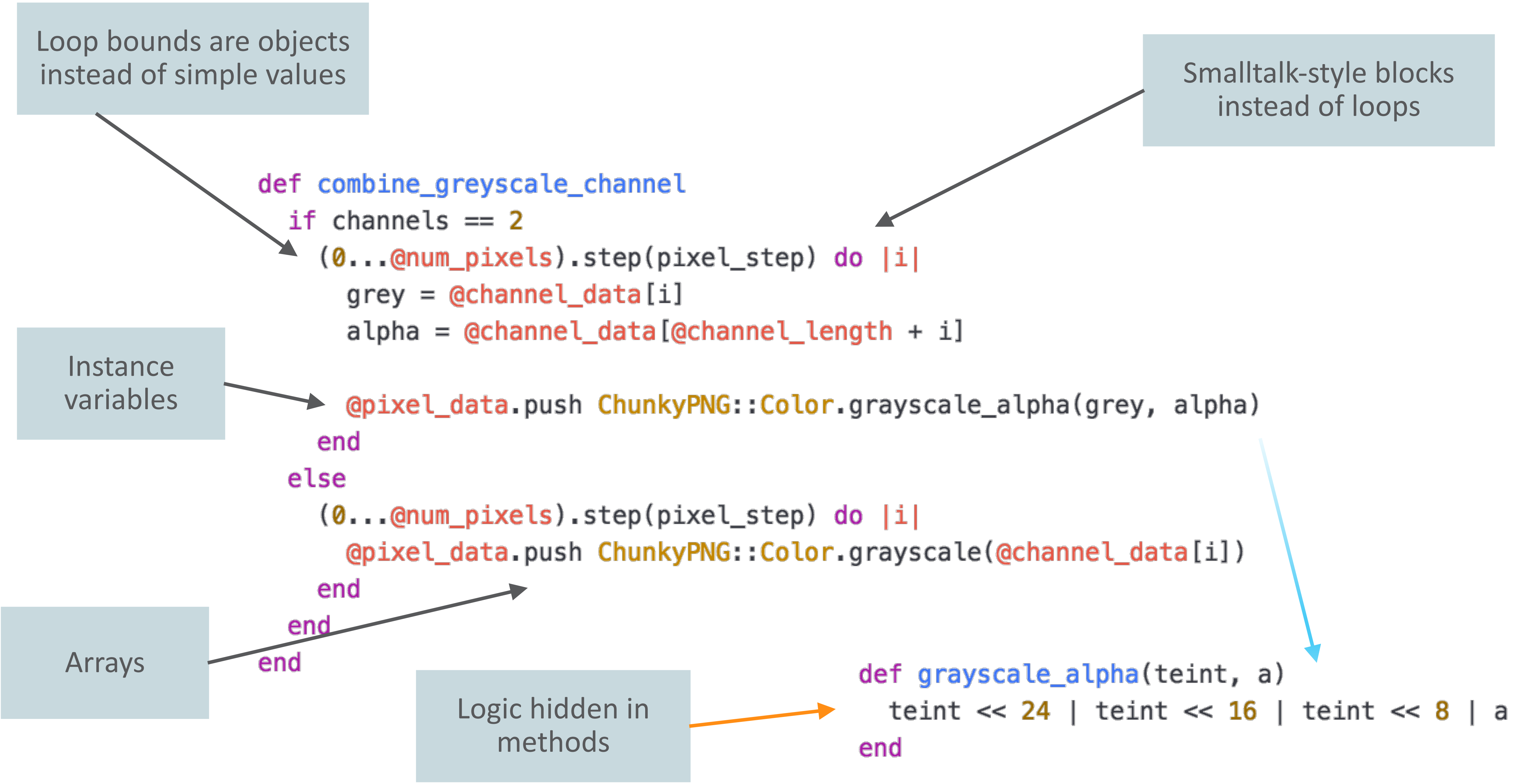
Only types are numerical or boolean

Simple floating point arithmetic

Vectorisation opportunities

```
z = 0
while z < 50
  tr = zrzr - zizi + cr
  ti = 2.0*zr*zi + ci
  zr = tr
  zi = ti
  zrzr = zr*zr
  zizi = zi*zi
  if zrzr+zizi > 4.0
    escape = 0b0
    break
  end
  z += 1
end
```

‘Real Ruby is much more complex!’



Loop bounds are objects instead of simple values

Smalltalk-style blocks instead of loops

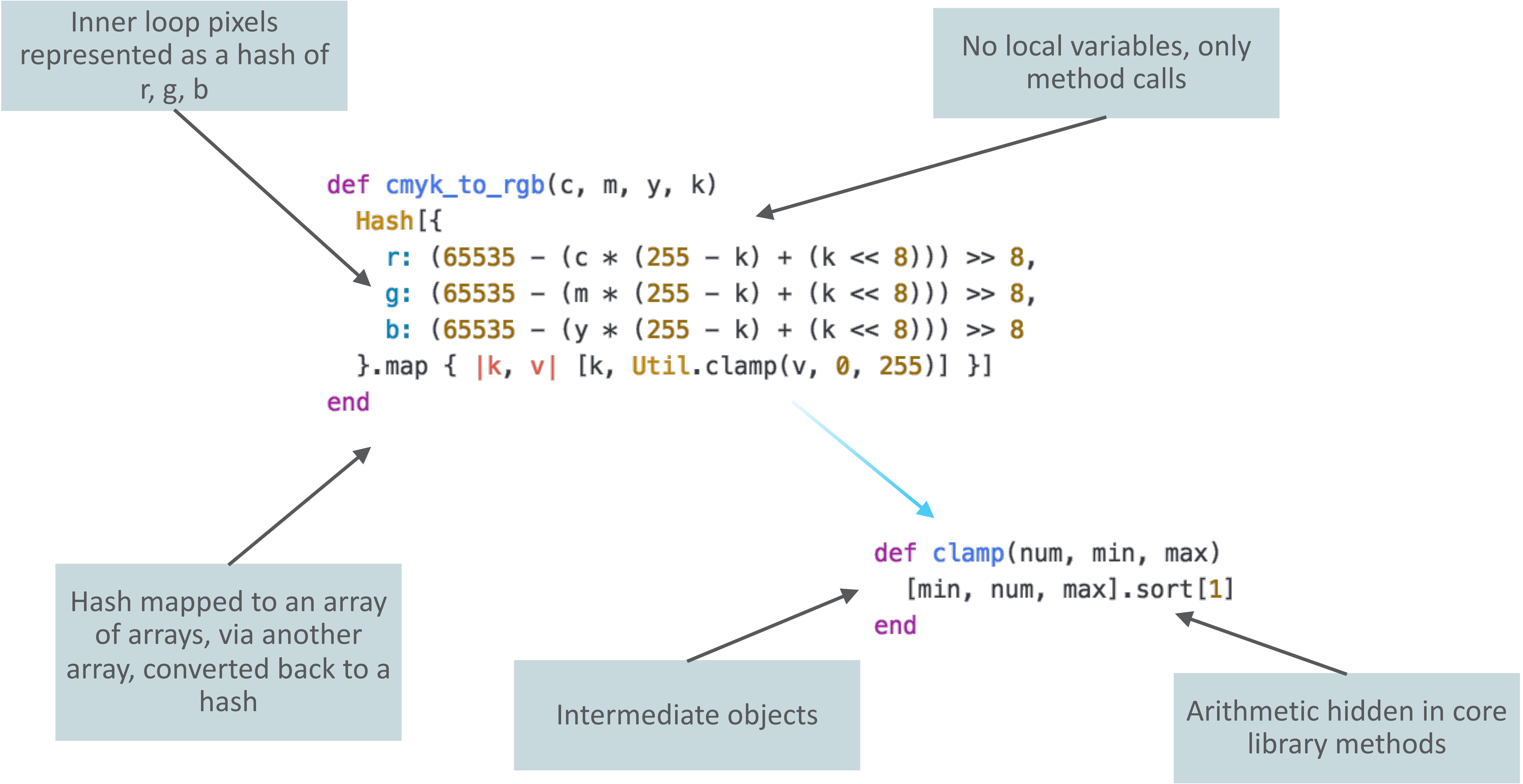
Instance variables

Arrays

Logic hidden in methods

```
def combine_greyscale_channel
  if channels == 2
    (0...@num_pixels).step(pixel_step) do |i|
      grey = @channel_data[i]
      alpha = @channel_data[@channel_length + i]

      @pixel_data.push ChunkyPNG::Color.grayscale_alpha(grey, alpha)
    end
  else
    (0...@num_pixels).step(pixel_step) do |i|
      @pixel_data.push ChunkyPNG::Color.grayscale(@channel_data[i])
    end
  end
end
```

```
def grayscale_alpha(teint, a)
  teint << 24 | teint << 16 | teint << 8 | a
end
```



Inner loop pixels represented as a hash of r, g, b

No local variables, only method calls

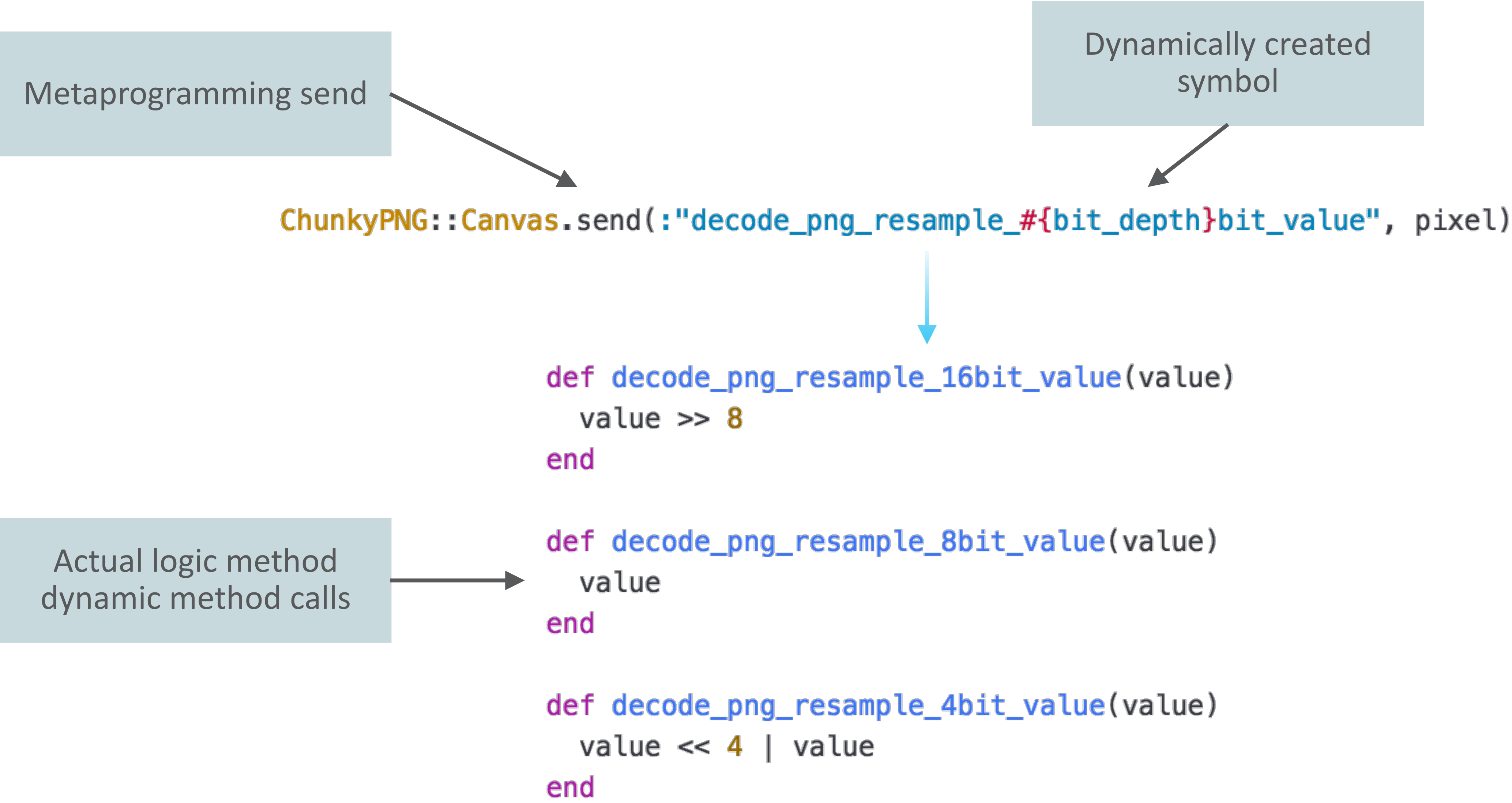
```ruby
def cmyk_to_rgb(c, m, y, k)
  Hash[{
    r: (65535 - (c * (255 - k) + (k << 8))) >> 8,
    g: (65535 - (m * (255 - k) + (k << 8))) >> 8,
    b: (65535 - (y * (255 - k) + (k << 8))) >> 8
  }.map { |k, v| [k, Util.clamp(v, 0, 255)] }]
end
```

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

Hash mapped to an array of arrays, via another array, converted back to a hash

Intermediate objects

Arithmetic hidden in core library methods



Metaprogramming send

Dynamically created symbol

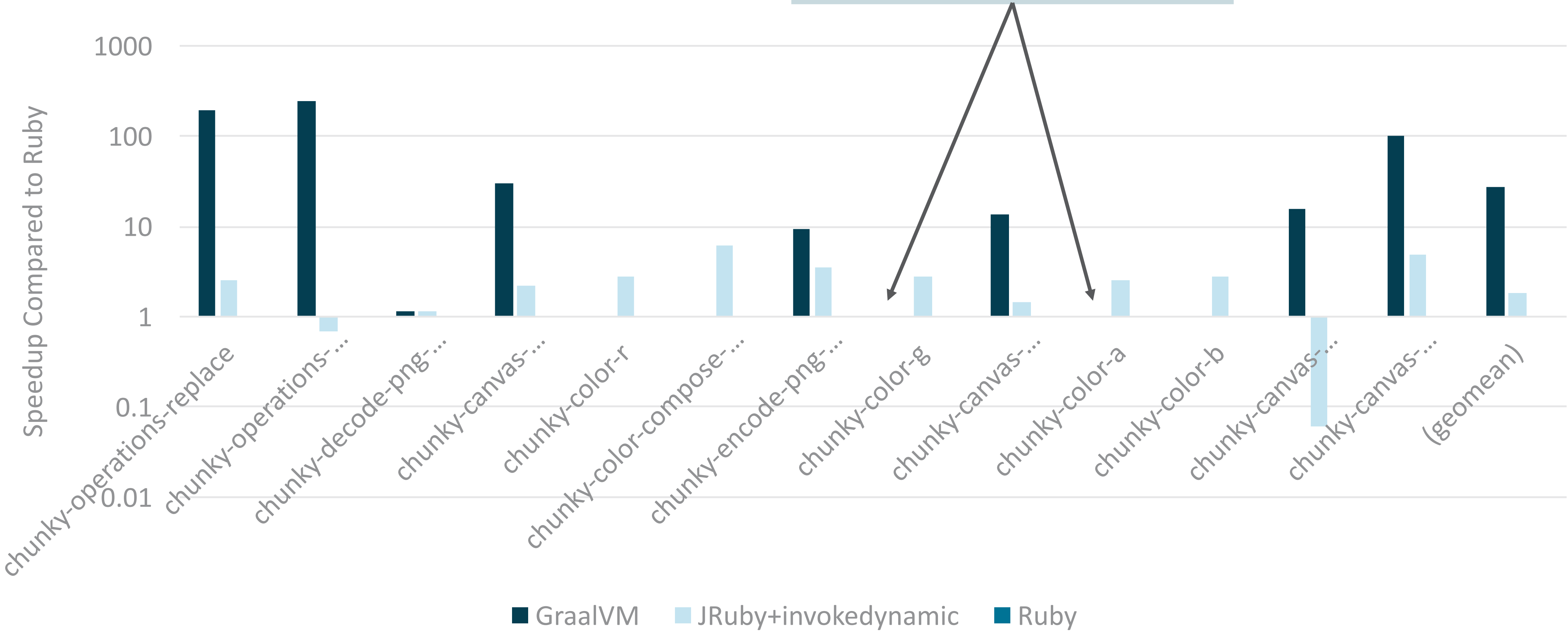
```ruby
ChunkyPNG::Canvas.send(:"decode_png_resample_#{bit_depth}bit_value", pixel)
```

Actual logic method dynamic method calls

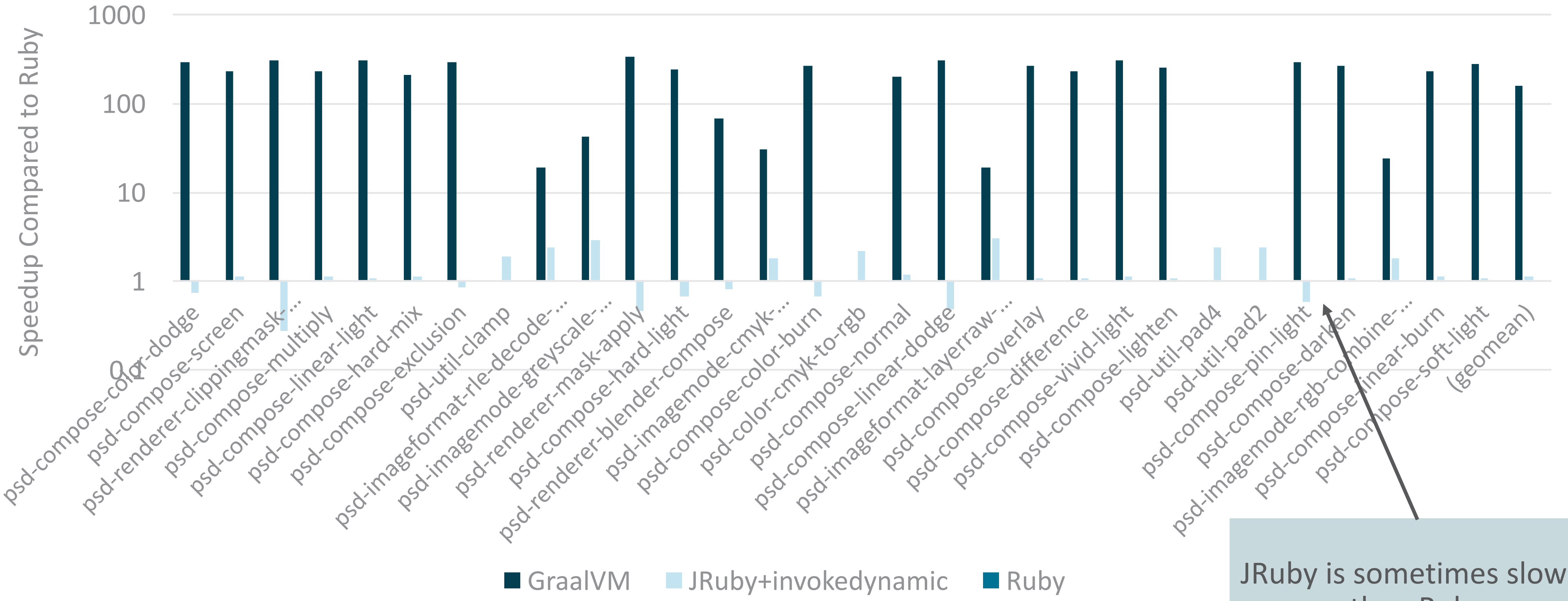
```ruby
def decode_png_resample_16bit_value(value)
  value >> 8
end

def decode_png_resample_8bit_value(value)
  value
end

def decode_png_resample_4bit_value(value)
  value << 4 | value
end
```

Chunky PNG kernels

Many of these benchmarks are optimised away entirely by GraalVM



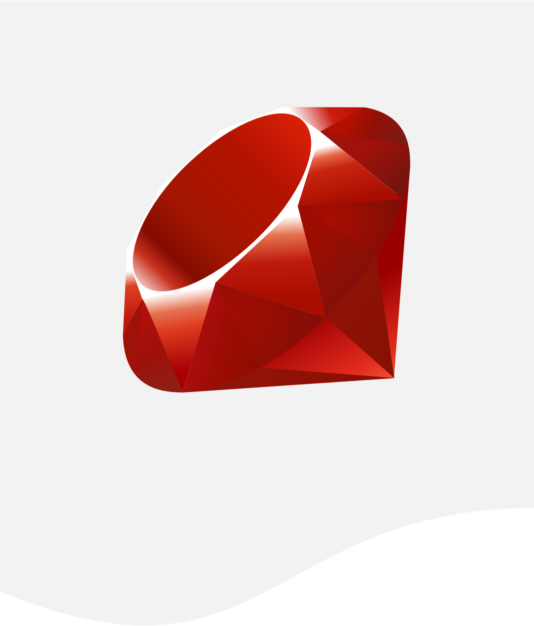
PSD.rb kernels

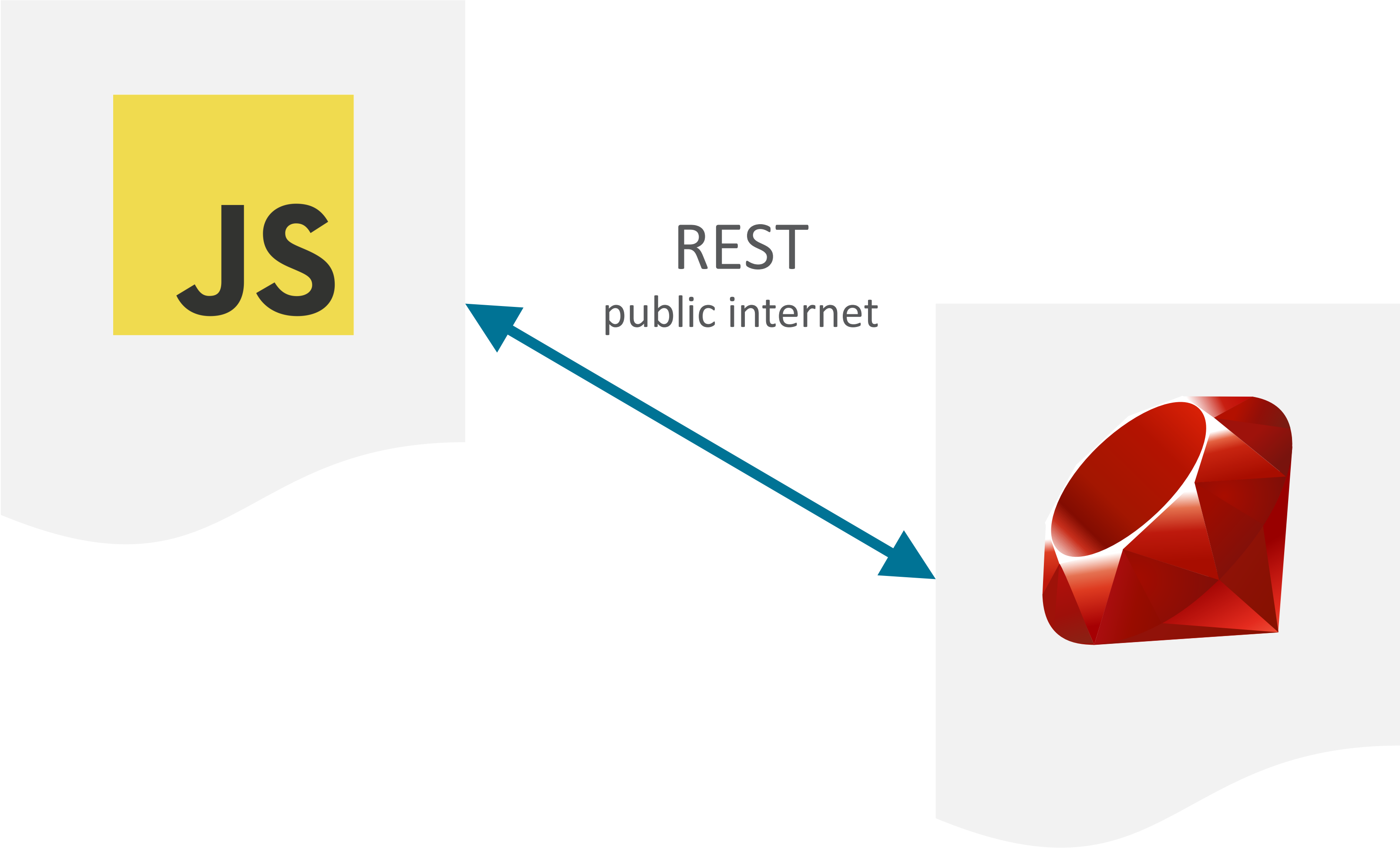


JRuby is sometimes slower than Ruby

Polyglot

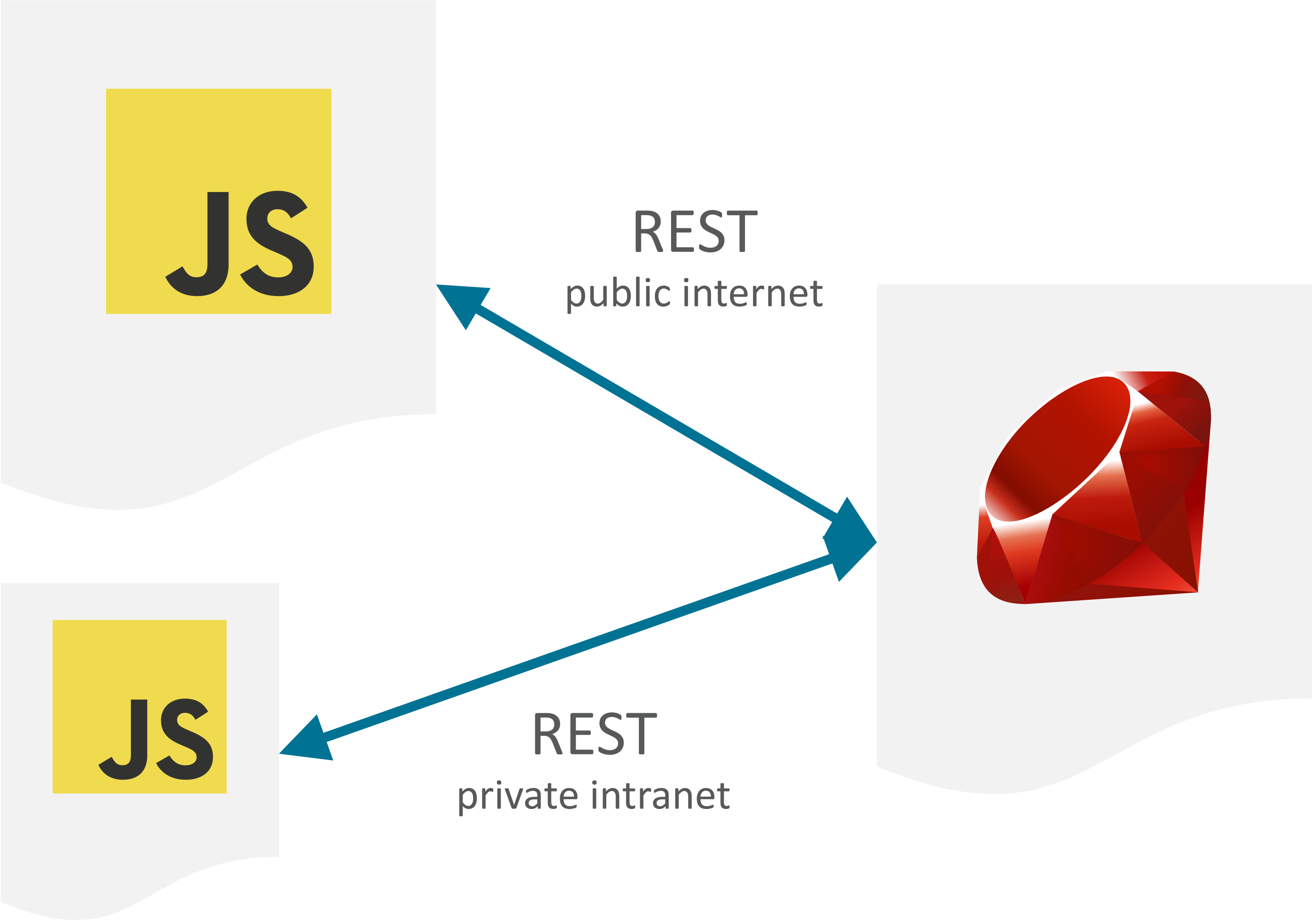
JS





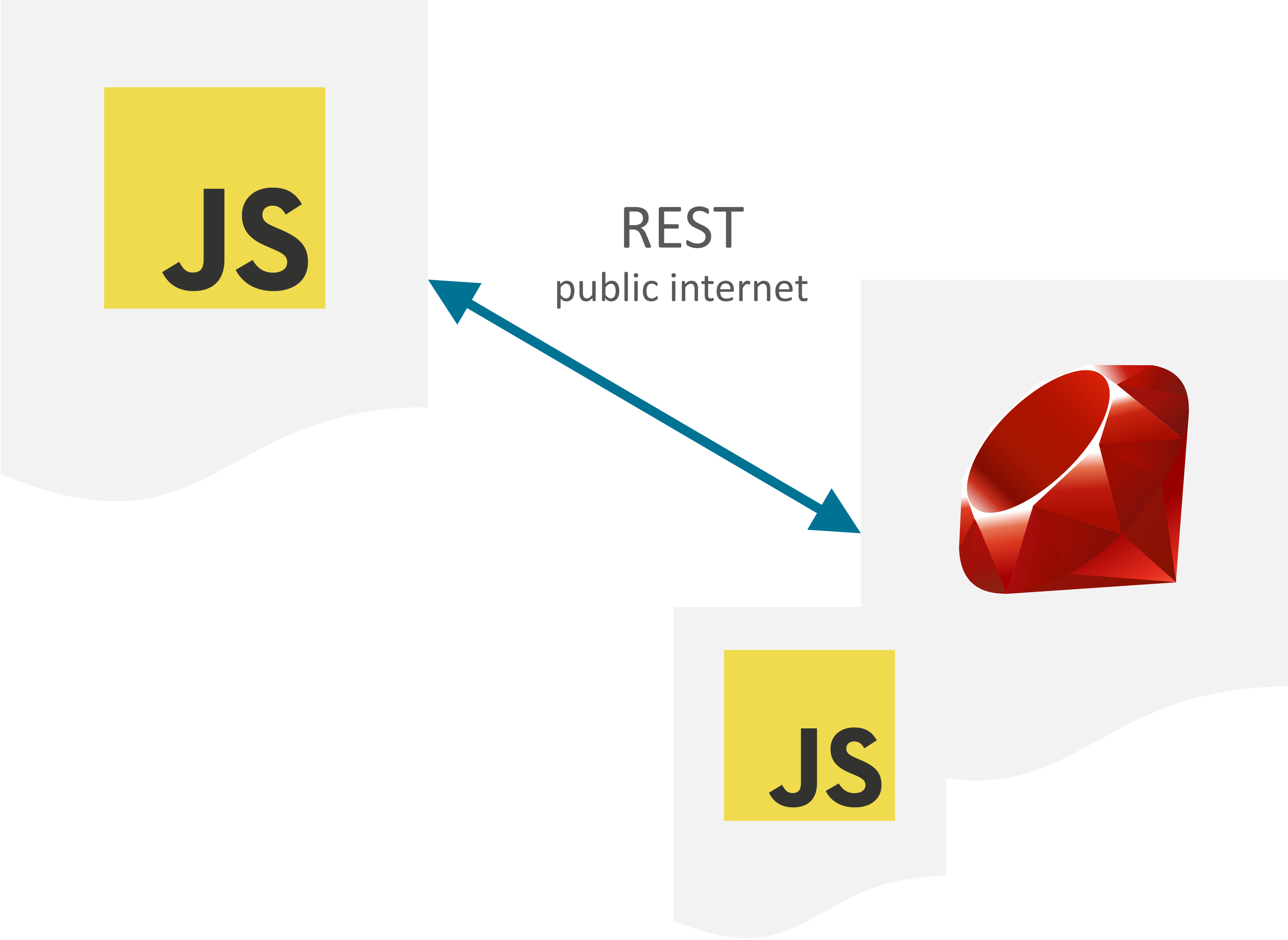
REST

public internet



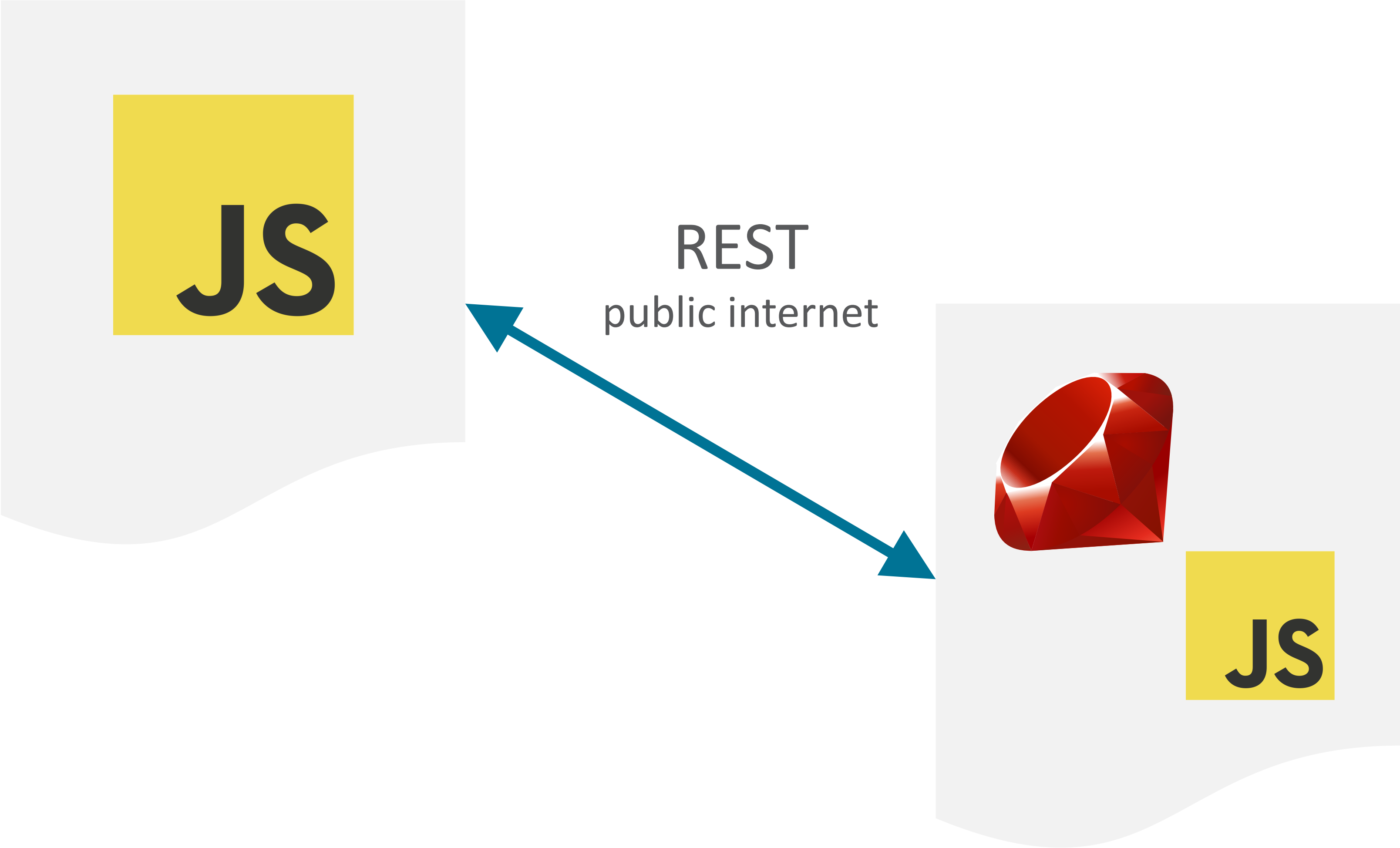
REST
public internet

REST
private intranet

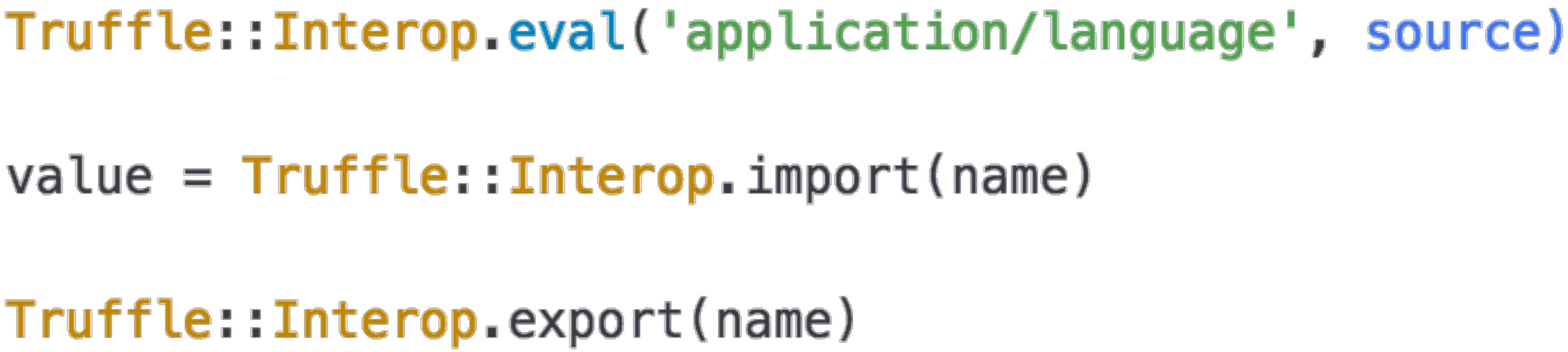


REST

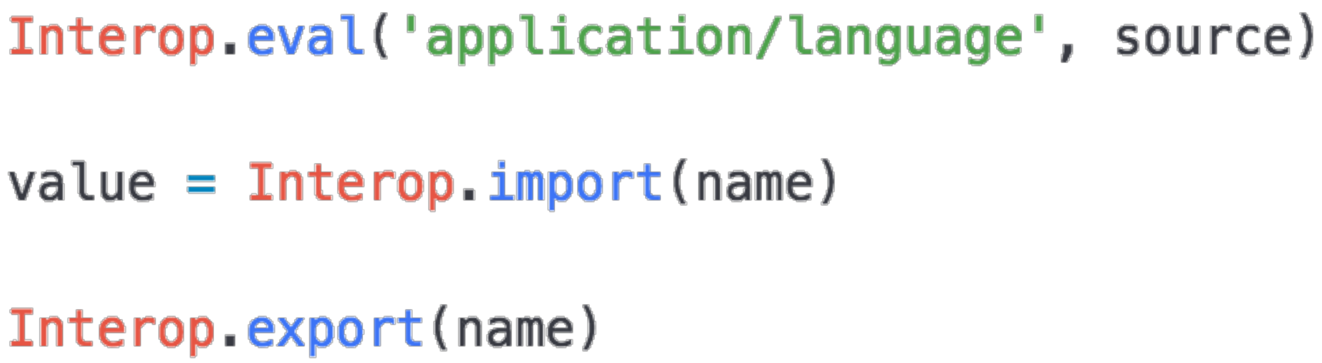
public internet


JS
REST
public internet
JS

How we do polyglot in GraalVM



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

value = Truffle::Interop.import(name)

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



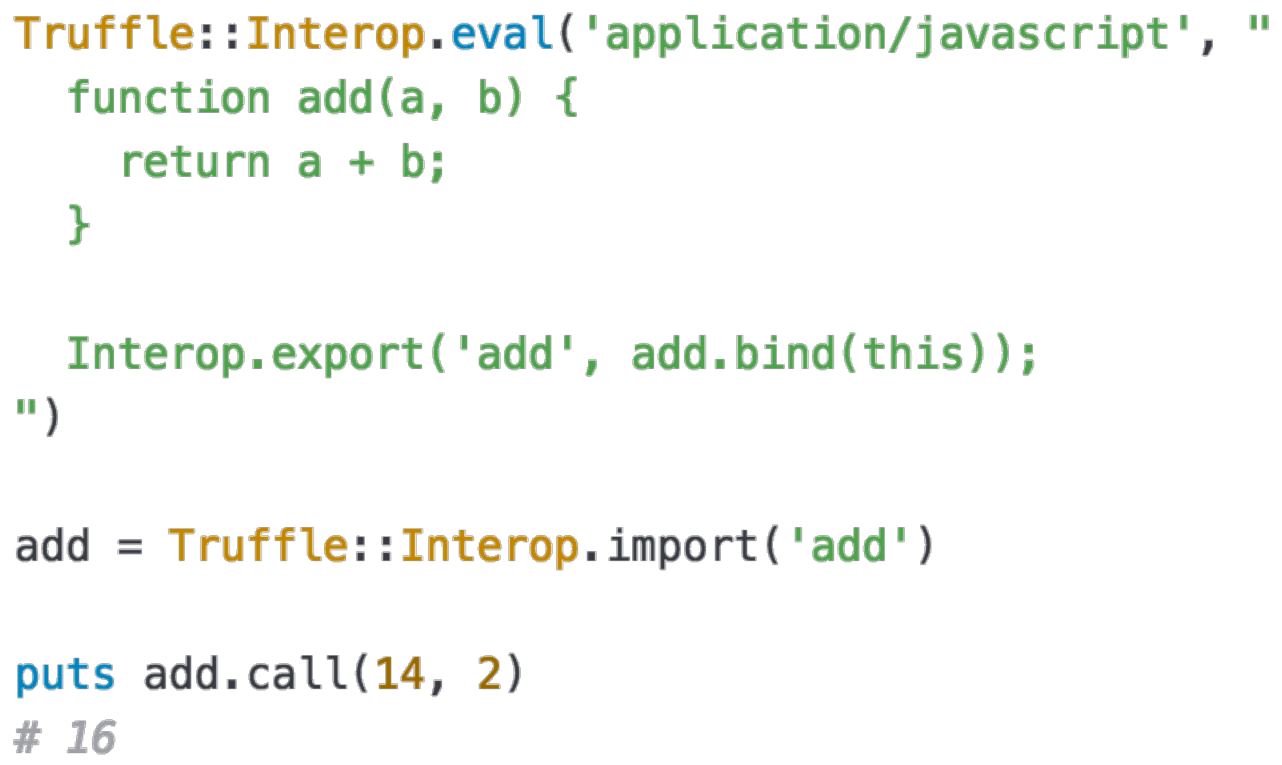
```
Interop.eval('application/language', source)

value = Interop.import(name)

Interop.export(name)
```

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

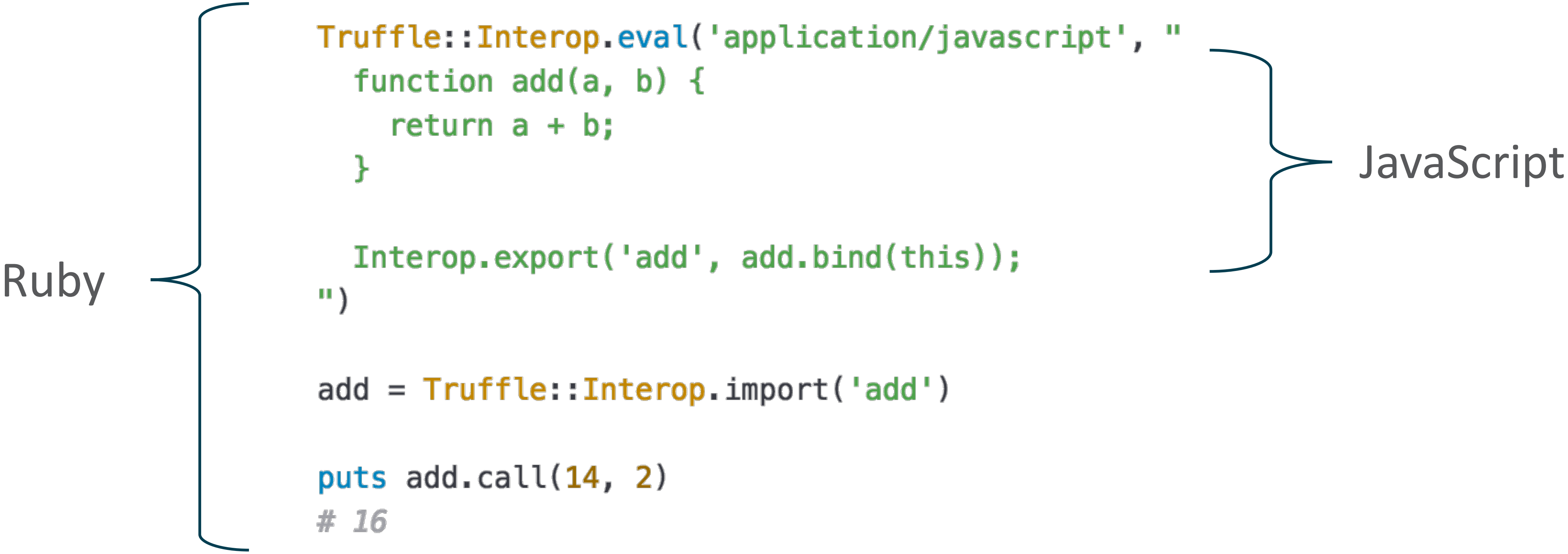
JavaScript

Ruby

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



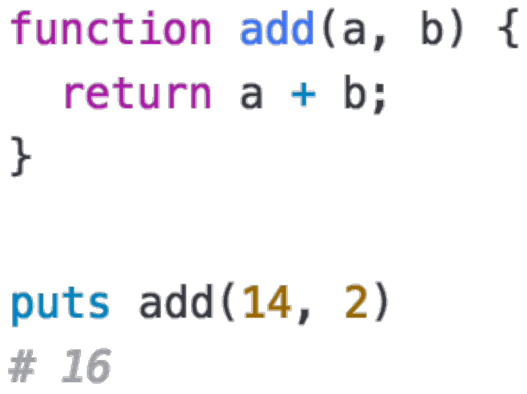
```
Truffle::Interop.eval('application/javascript', "
  function add(a, b) {
    return a + b;
  }

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

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

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



Ruby

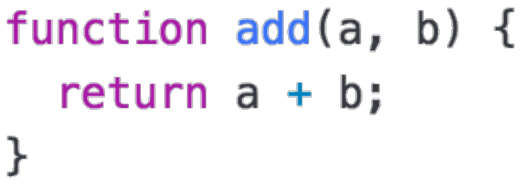
JavaScript

```
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
```



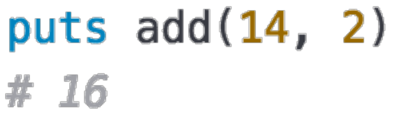
```
function add(a, b) {
  return a + b;
}

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

JavaScript

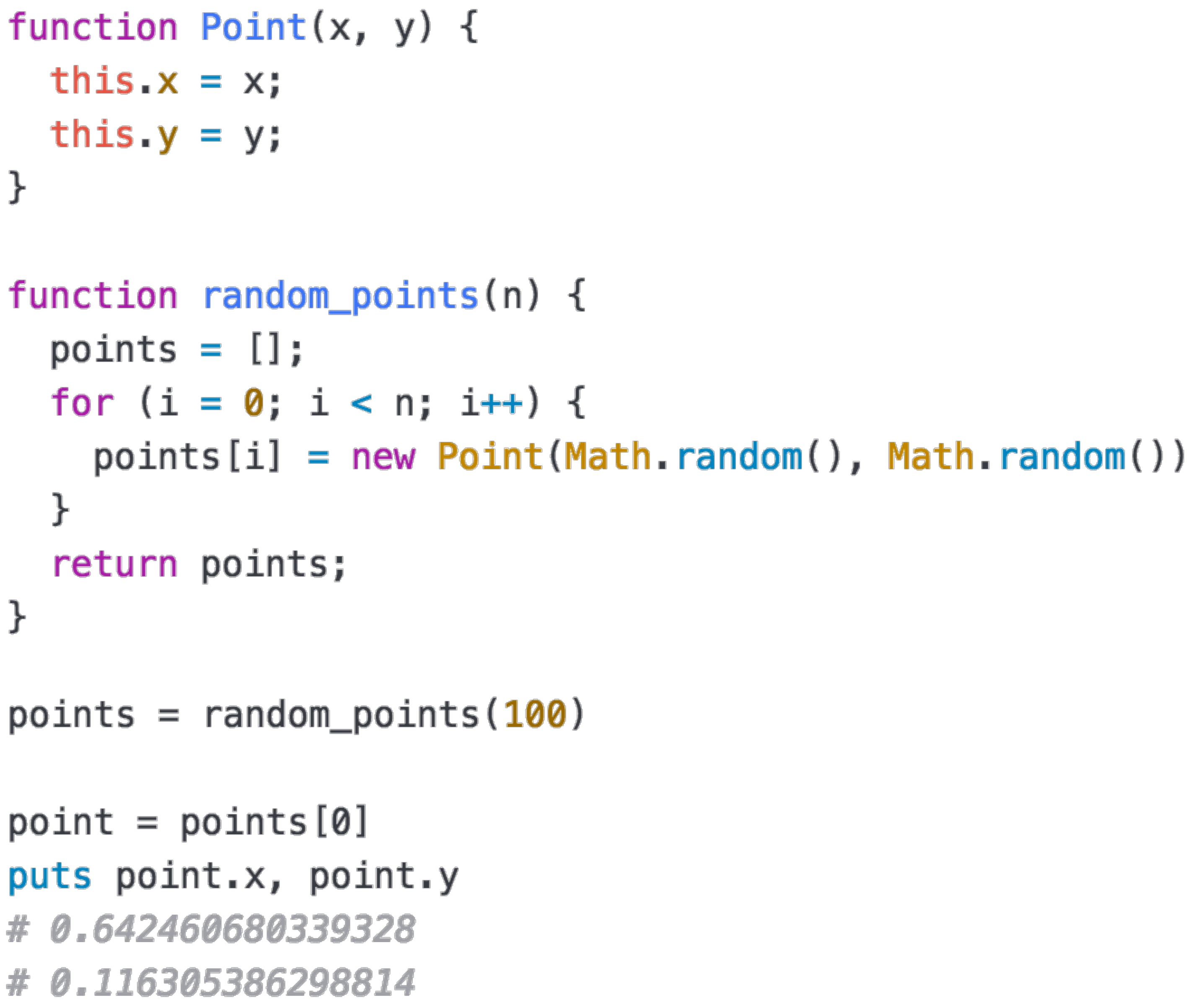


```
function add(a, b) {
  return a + b;
}
```

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

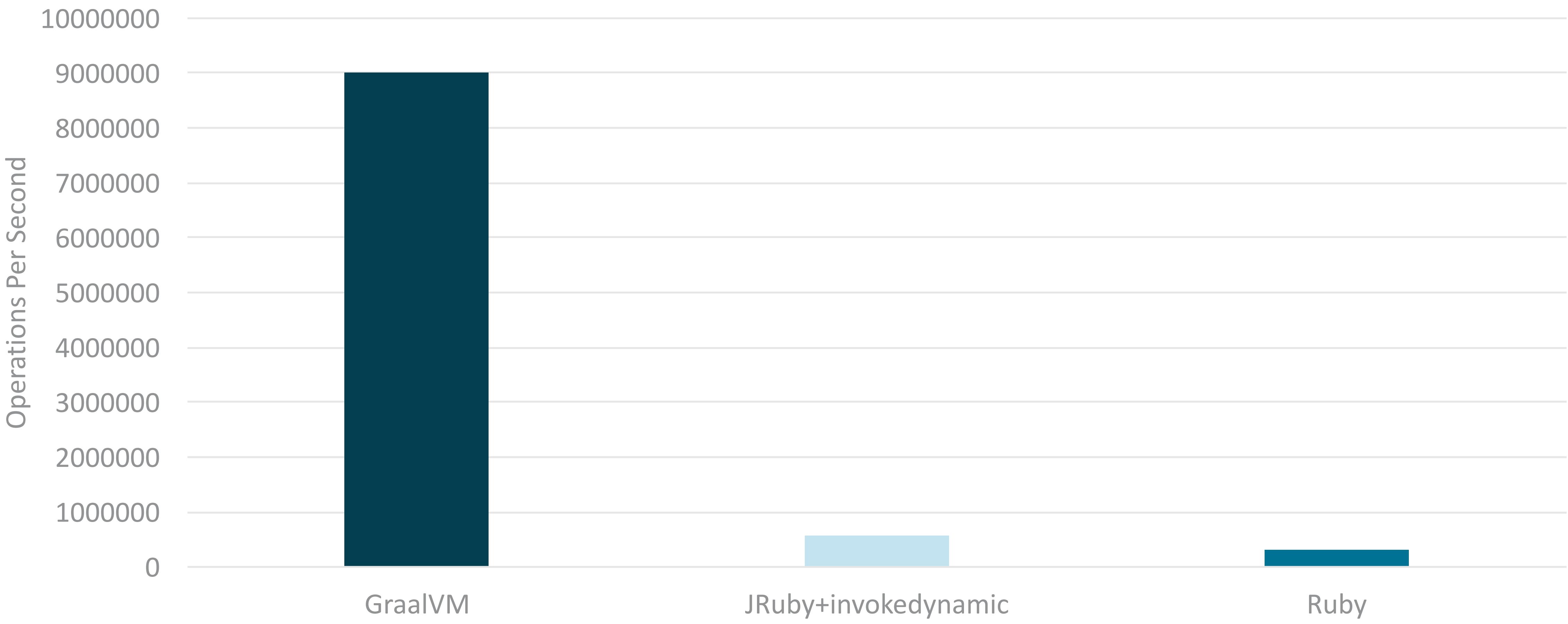
```
def clamp(num, min, max)
  [min, num, max].sort[1]
end

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
```

```
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
```

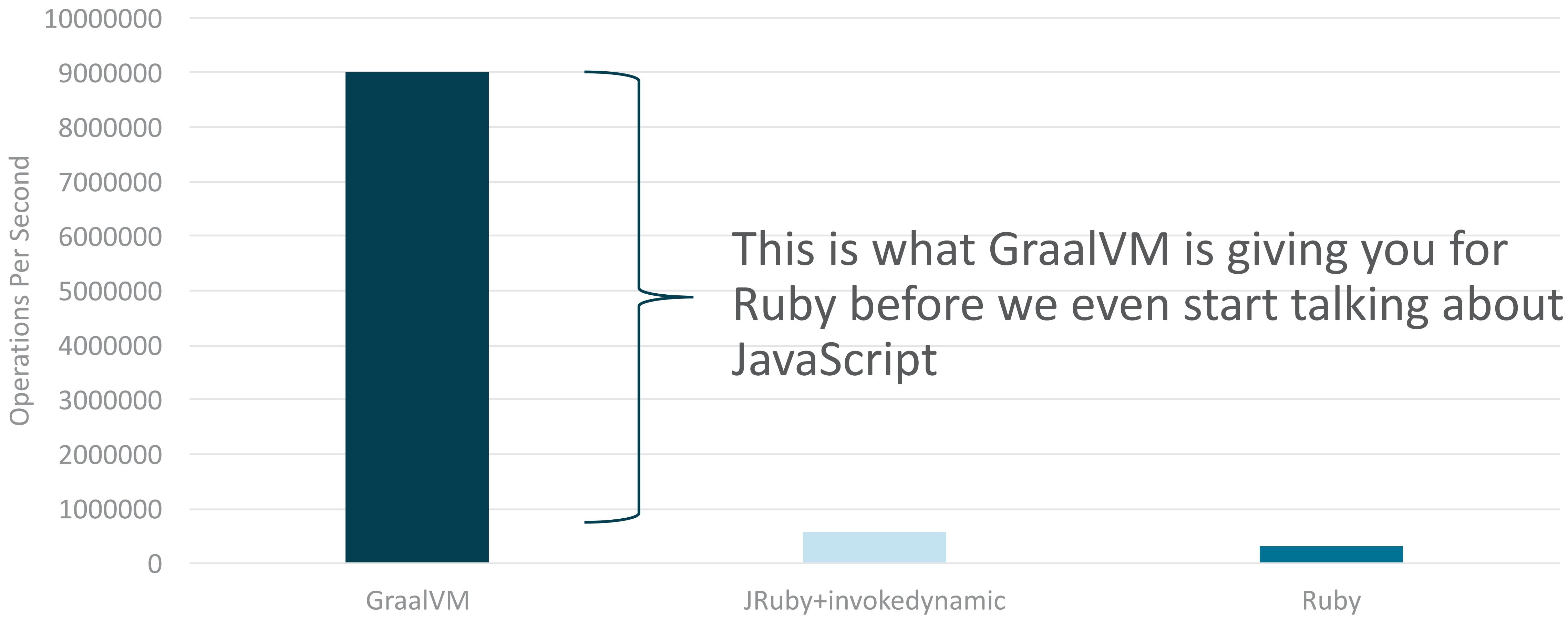
Warms up and then reports iterations per second

Random inputs stop the whole thing being totally optimised away

`clamp` in Ruby



`clamp` in Ruby



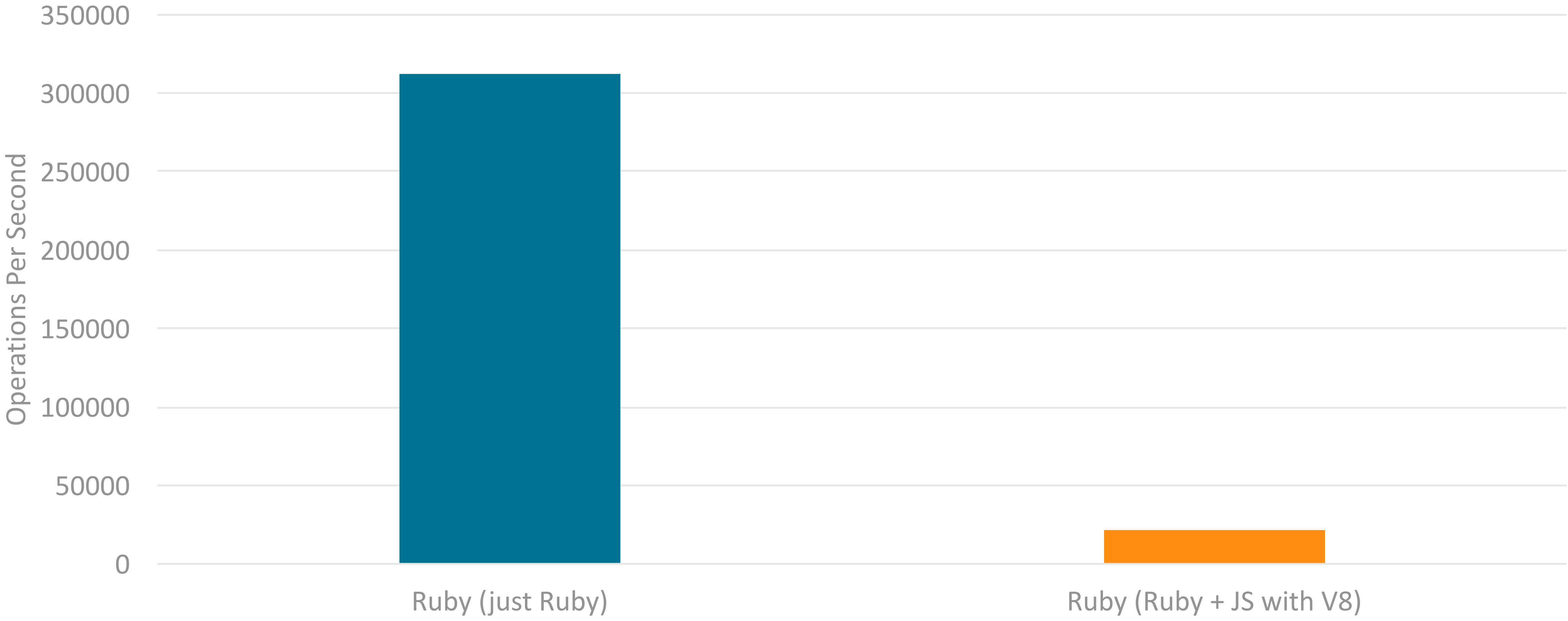
This is what GraalVM is giving you for Ruby before we even start talking about JavaScript

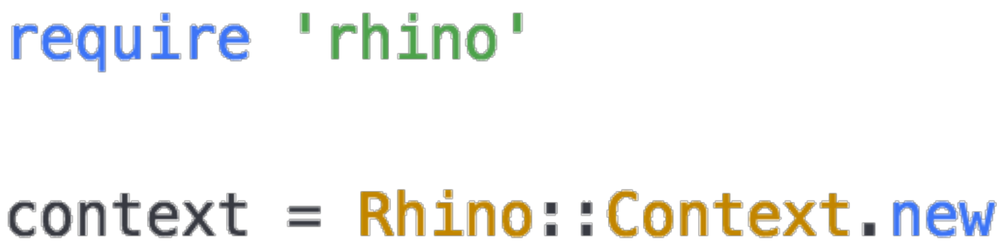
```
require 'v8'

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
```

```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
```

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

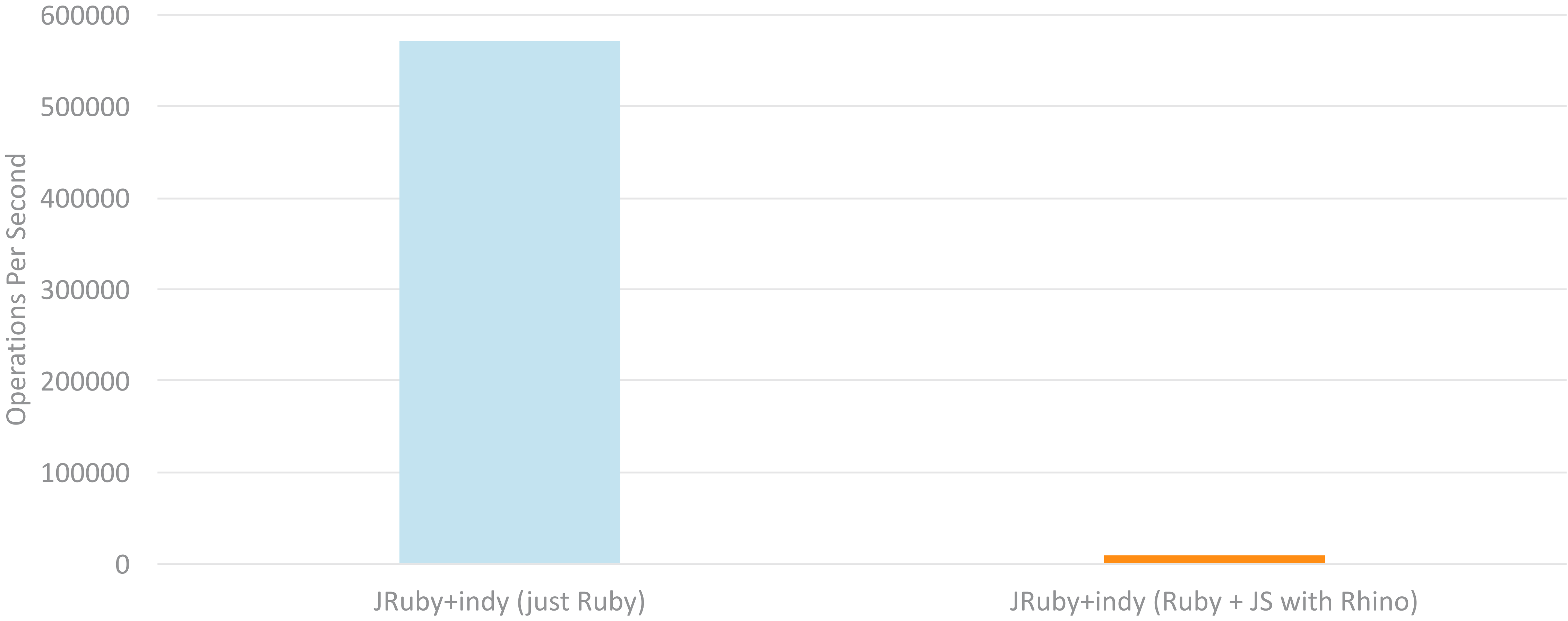
`clamp` in Ruby and JavaScript with V8

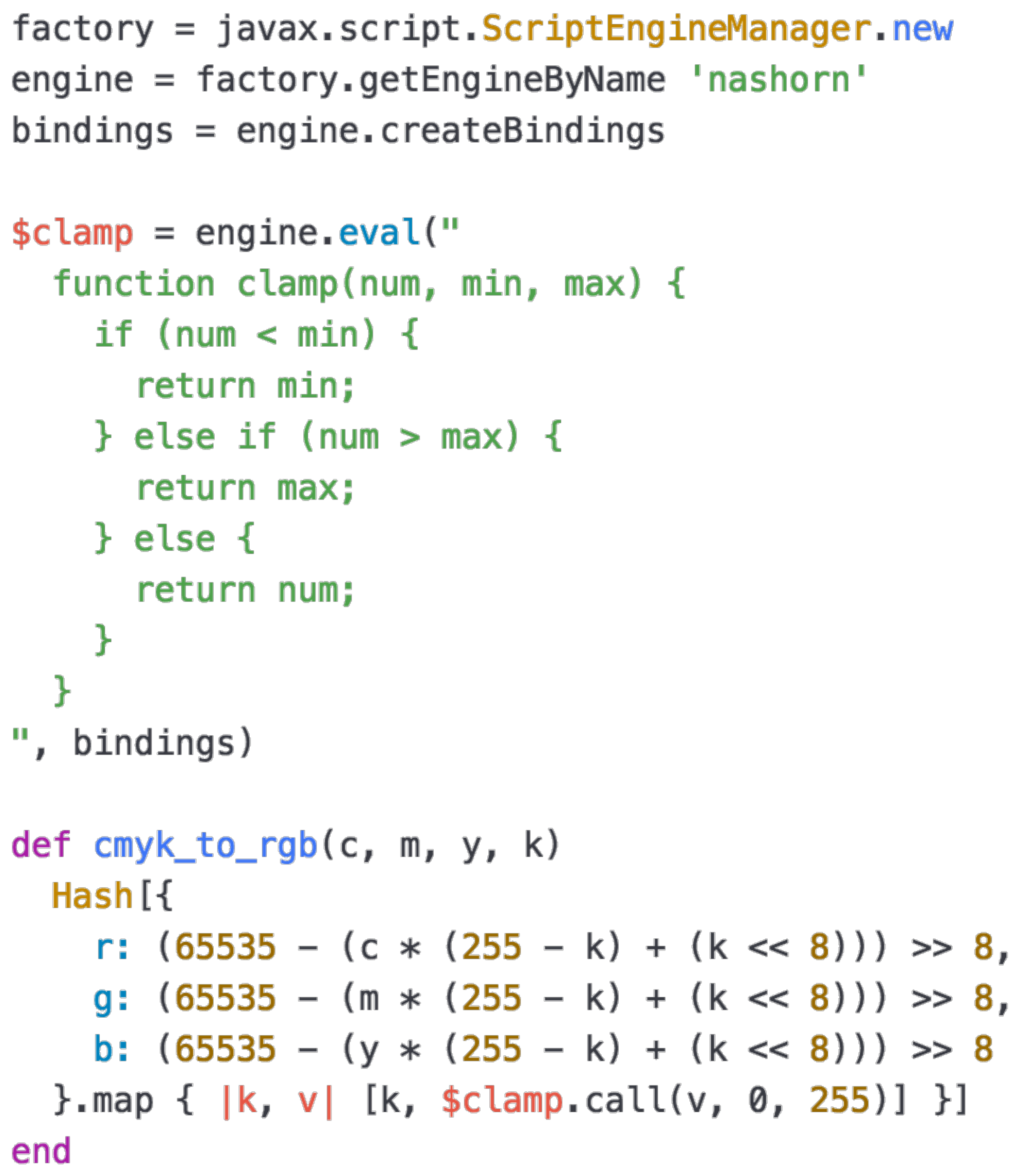




```
require 'rhino'

context = Rhino::Context.new
```

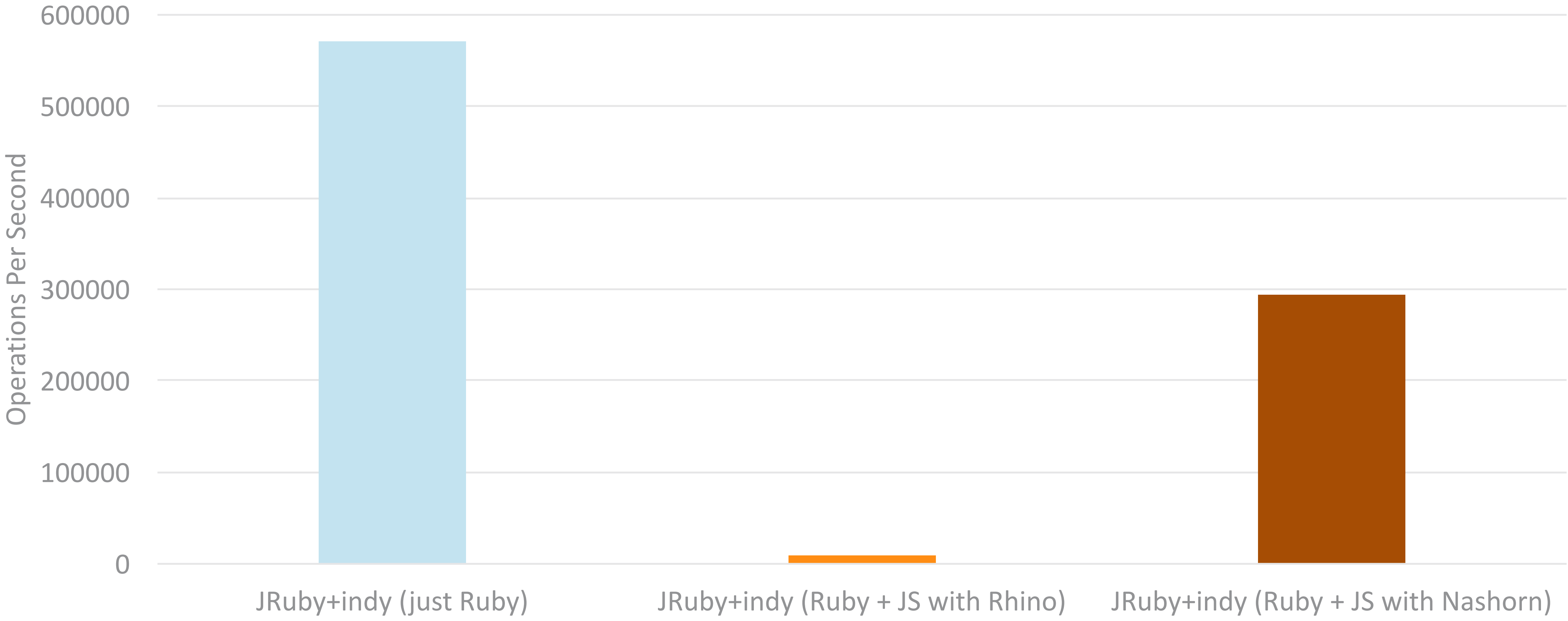
`clamp` in Ruby and JavaScript with JRuby and Rhino

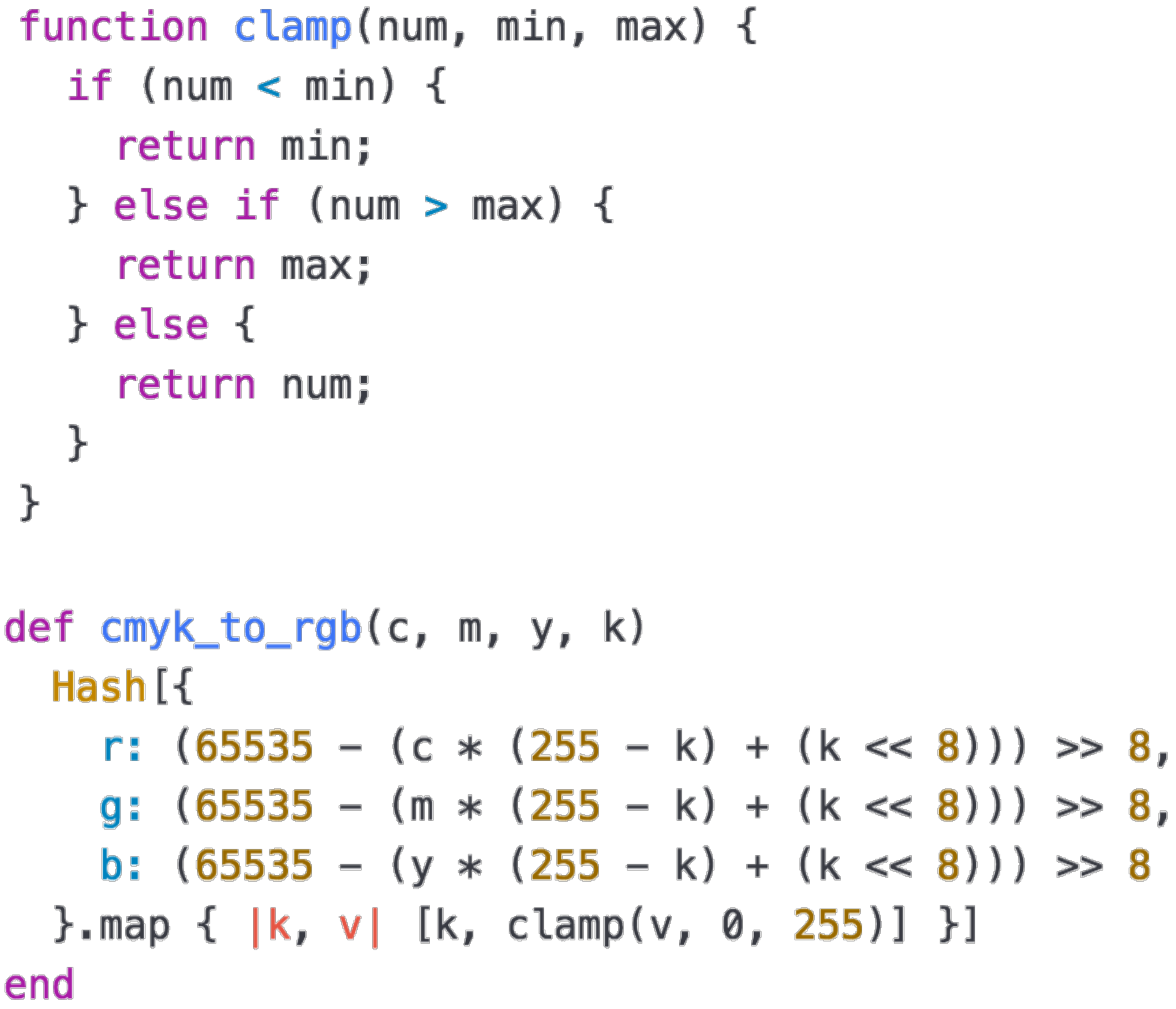




```
factory = javax.script.ScriptEngineManager.new
engine = factory.getEngineByName 'nashorn'
bindings = engine.createBindings

$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

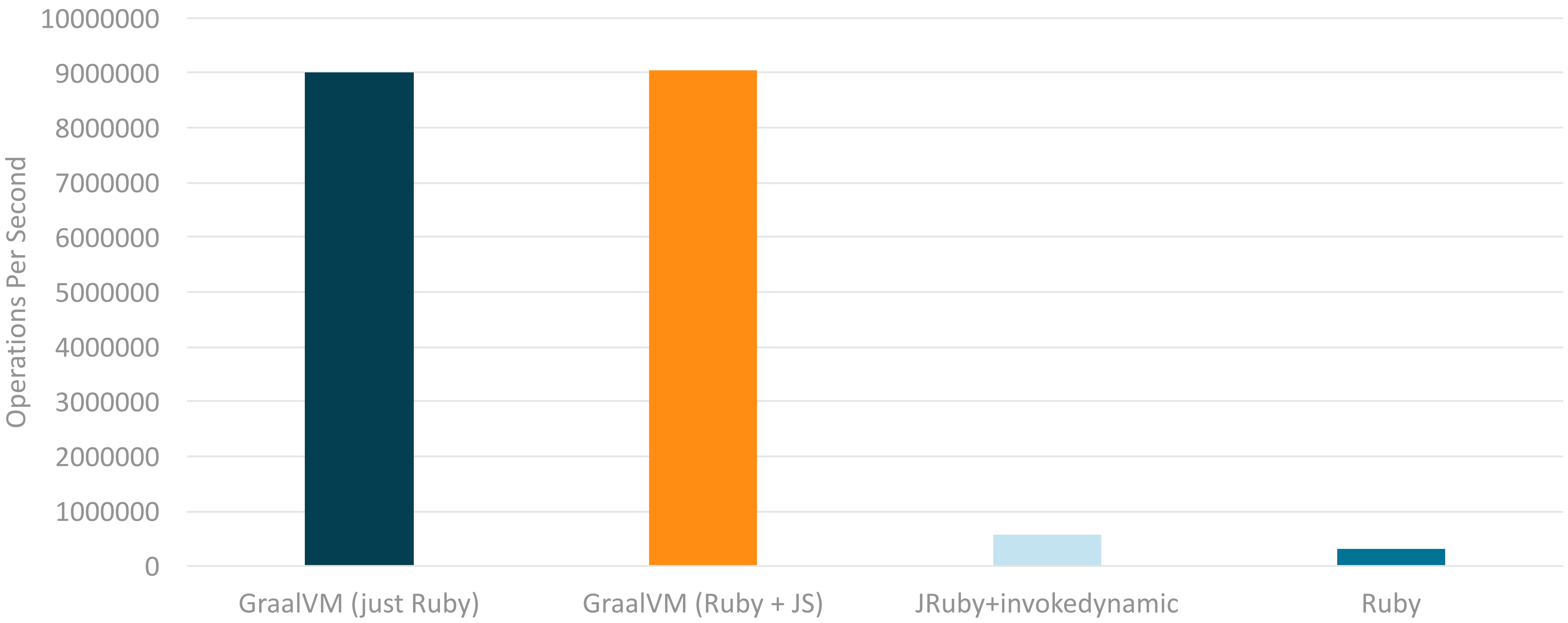




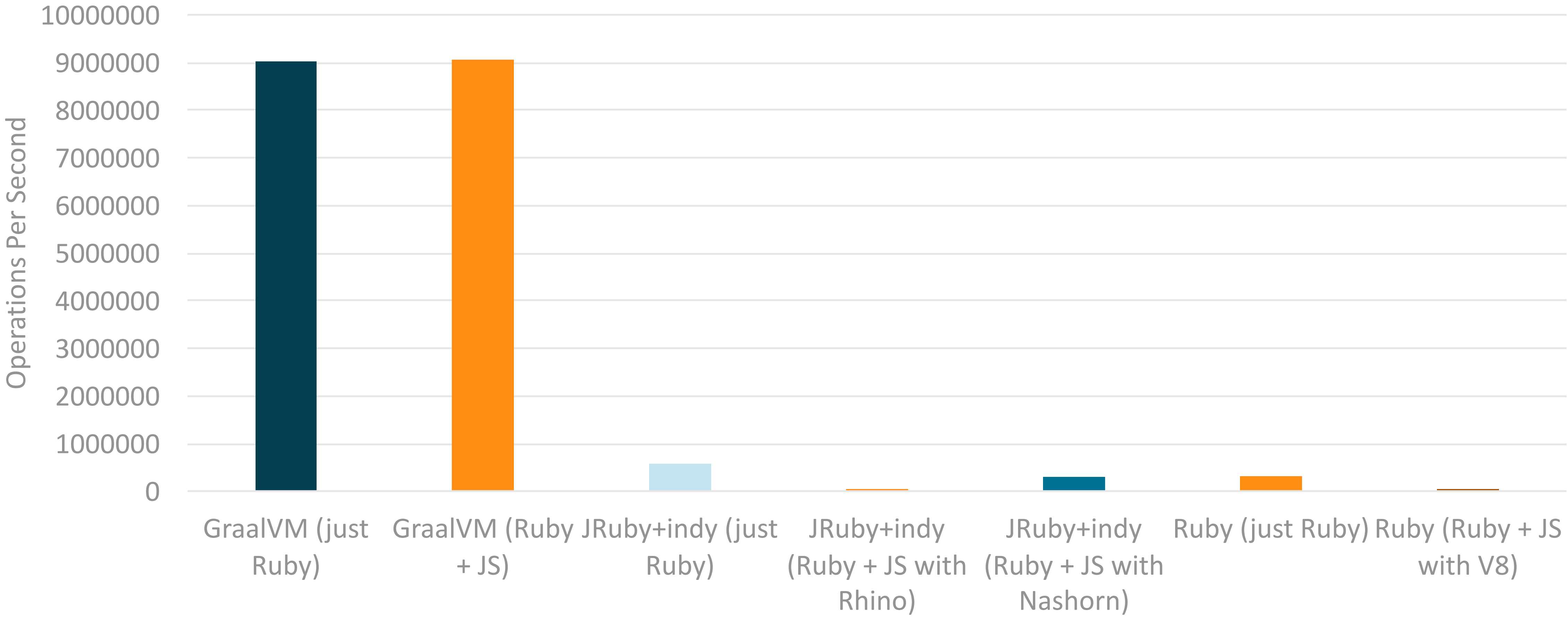
```
function clamp(num, min, max) {
  if (num < min) {
    return min;
  } else if (num > max) {
    return max;
  } else {
    return num;
  }
}
```

```
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
```

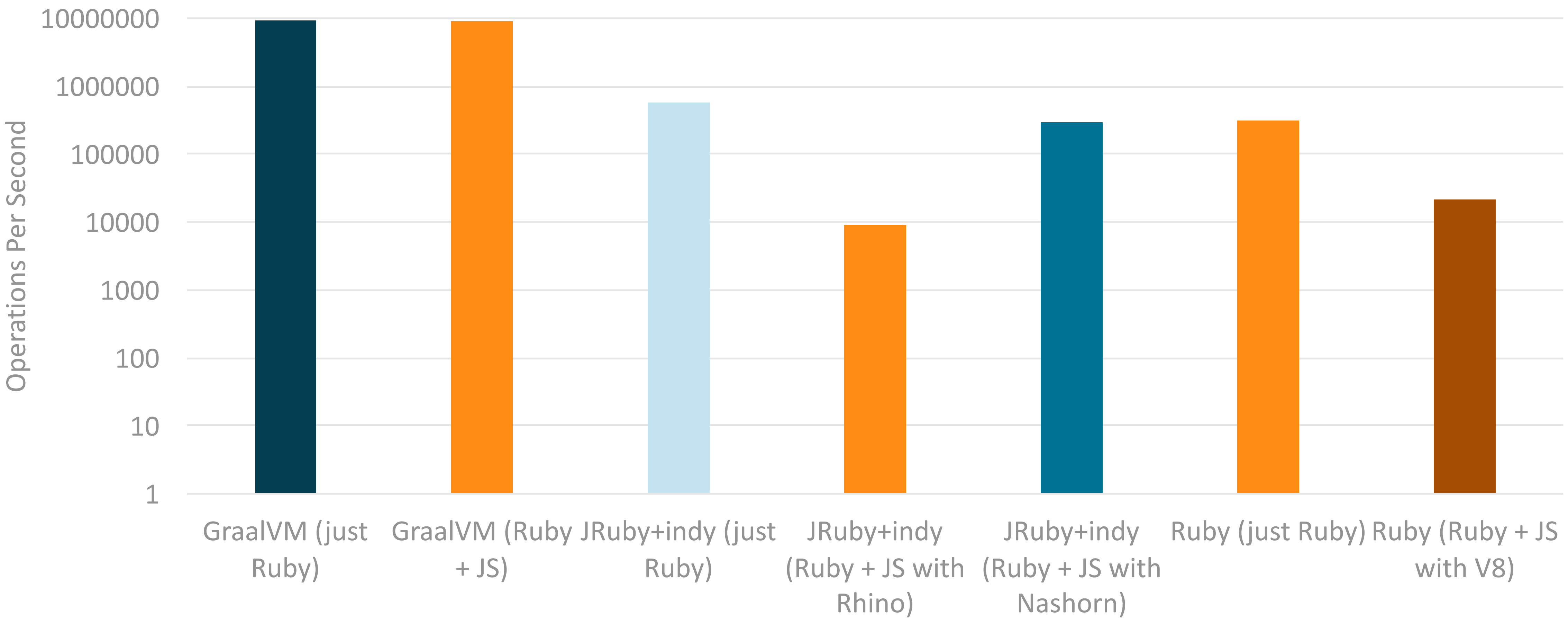
`clamp` in Ruby and JavaScript with GraalVM



`clamp` in all configurations

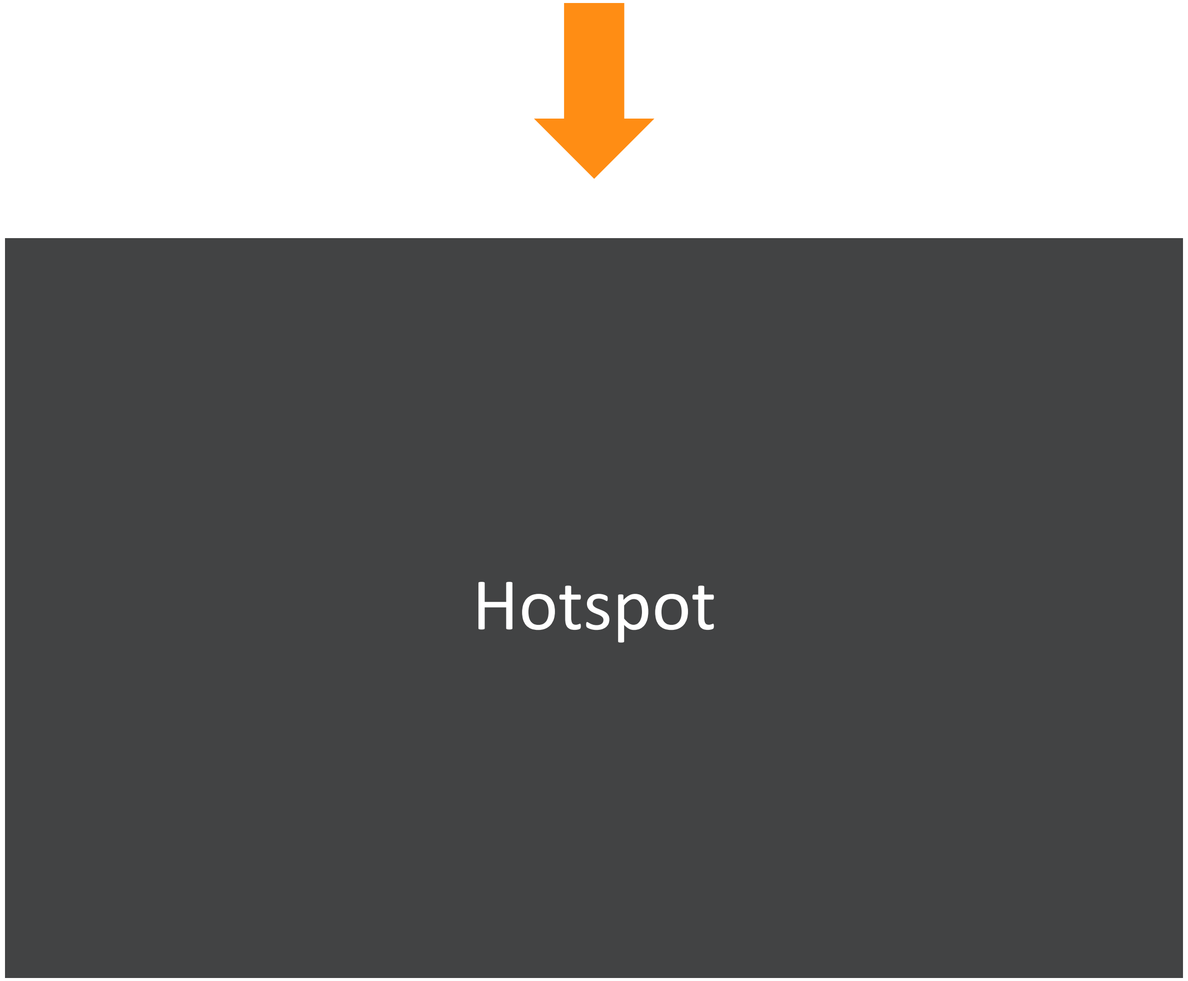


`clamp` in all configurations

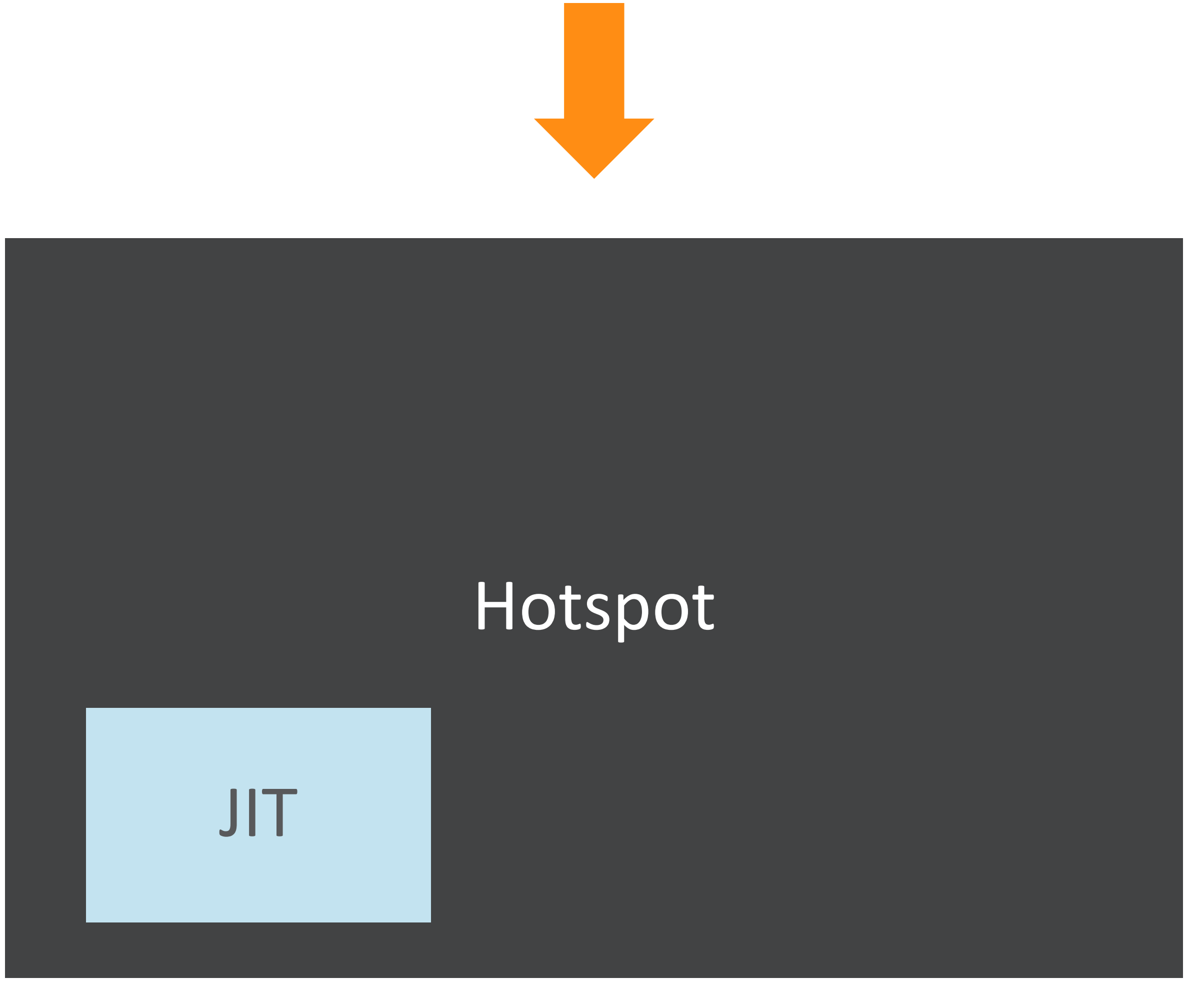


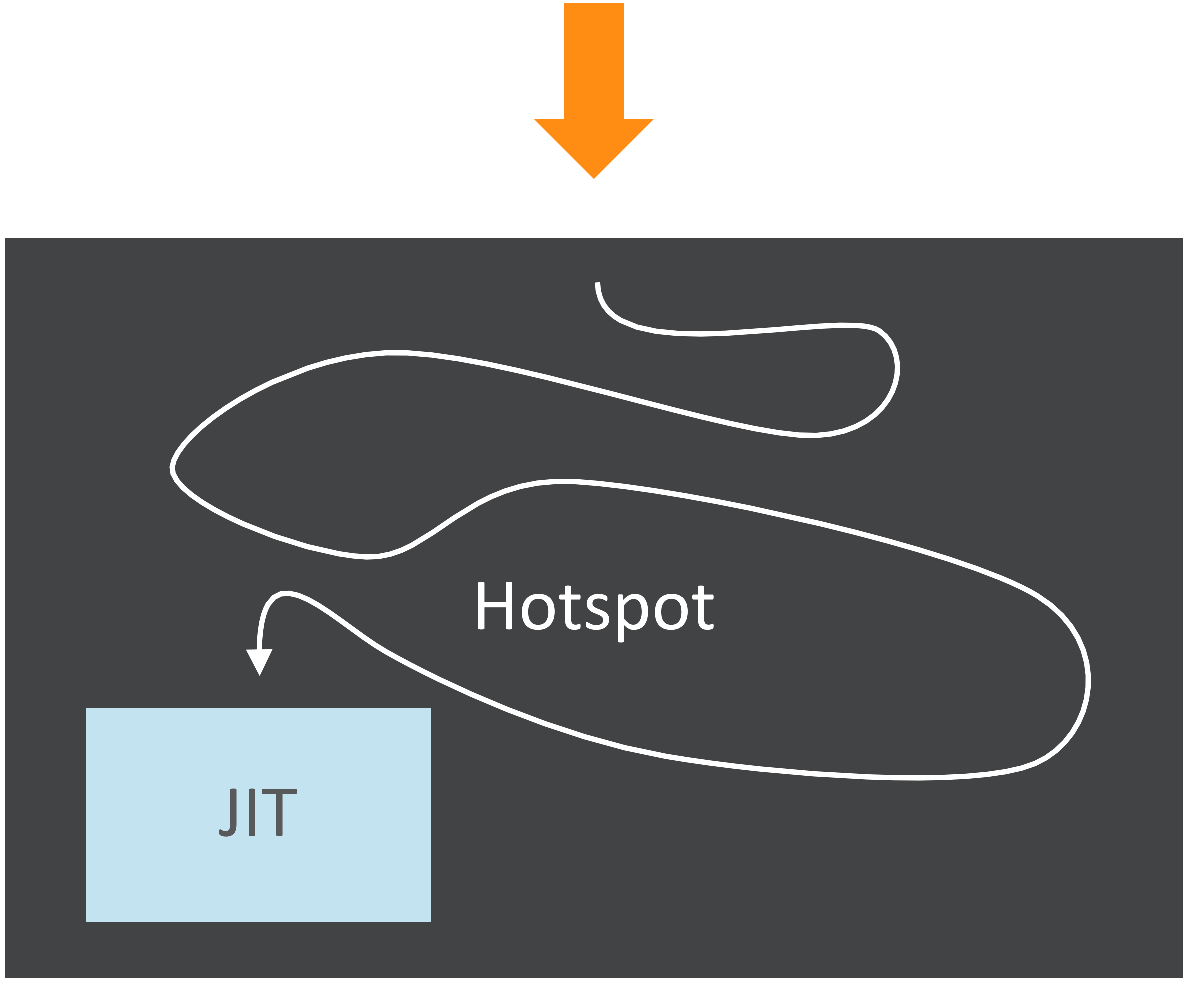
How Graal achieves this

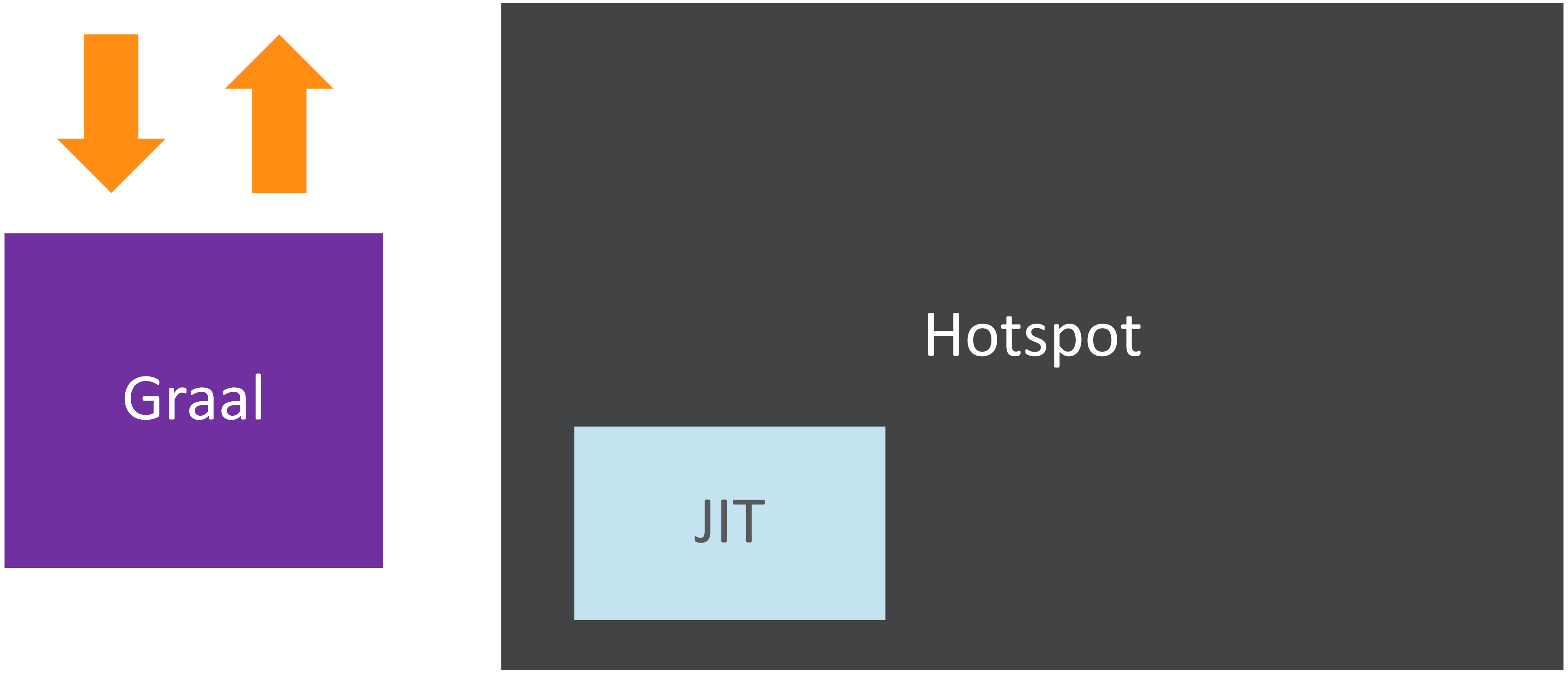
Hotspot

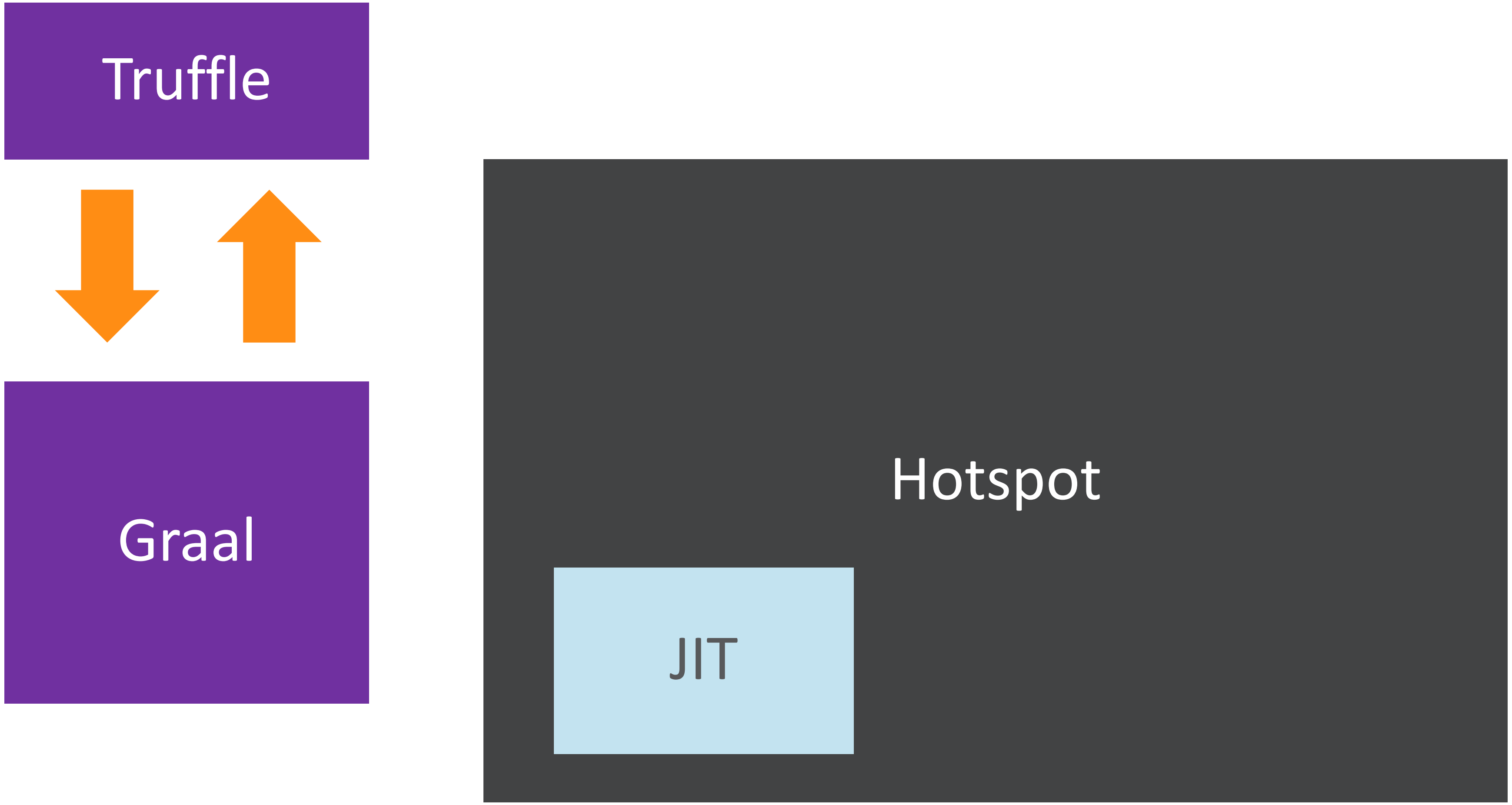


Hotspot


Hotspot
JIT


Hotspot
JIT

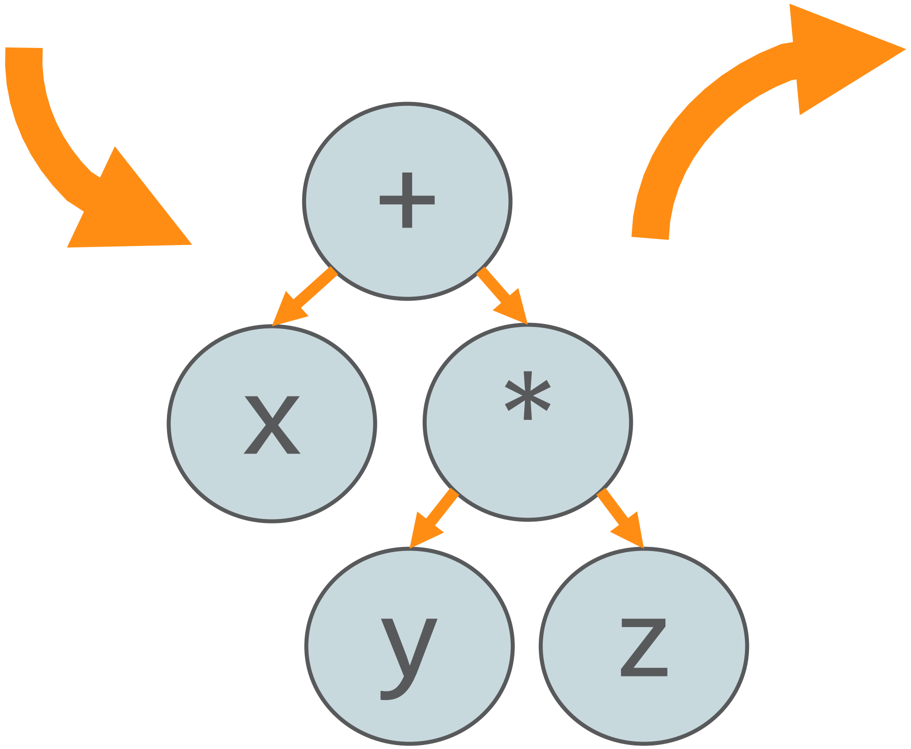

Graal
Hotspot
JIT


Truffle
Graal
Hotspot
JIT

The very basics of Truffle and Graal

- Common representation of programs
- Keep it rich enough to not have to throw away meaning
- Common optimisation of the representation

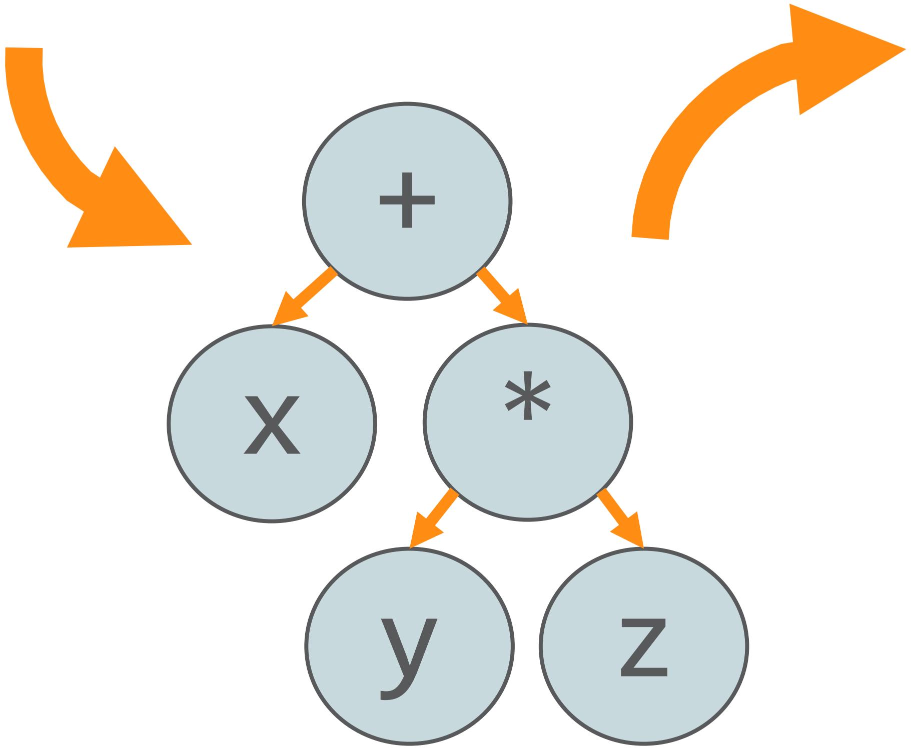
x + y * z



```
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
```

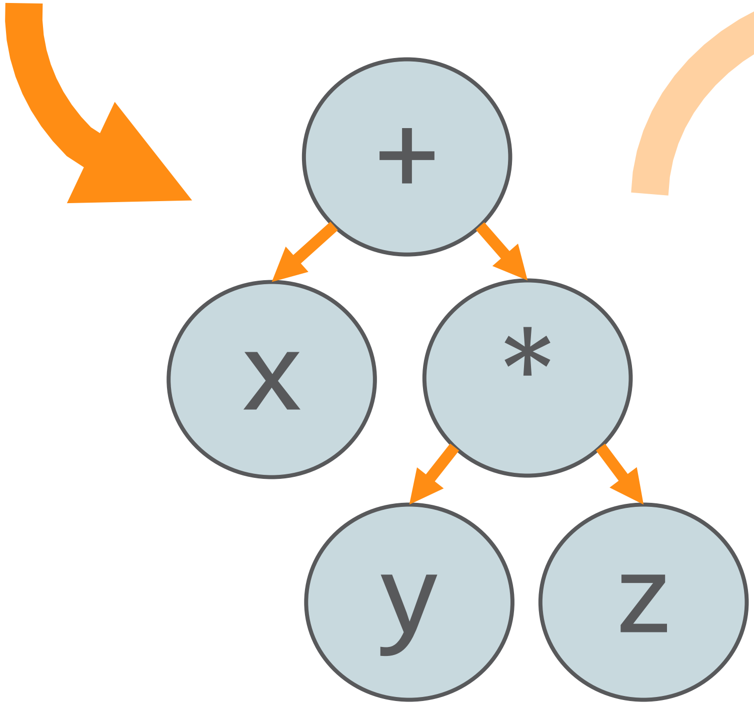
```
x + y * z
```



```
load_local x
load_local y
load_local z
call *
call +
```

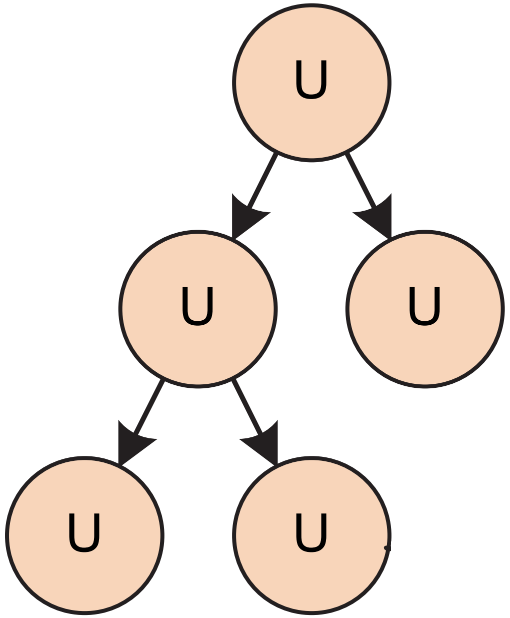
```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  %rdx, -24(%rbp)
movq  -16(%rbp), %rax
movl  %eax, %edx
movq  -24(%rbp), %rax
imull %edx, %eax
movq  -8(%rbp), %rdx
addl  %edx, %eax
popq  %rbp
ret
```

```
x + y * z
```



```
load_local x
load_local y
load_local z
call *
call +
```

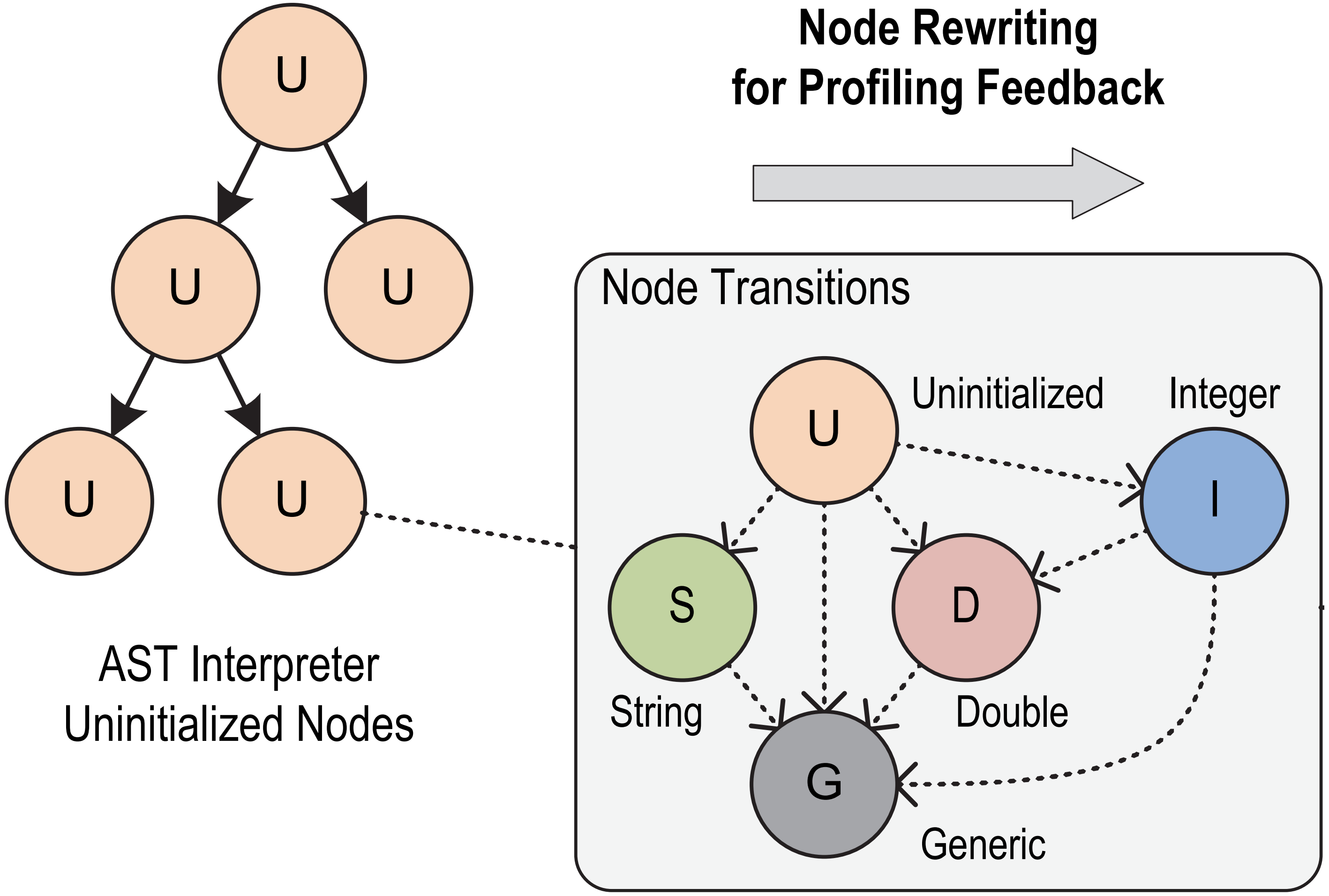
```
pushq %rbp
movq  %rsp, %rbp
movq  %rdi, -8(%rbp)
movq  %rsi, -16(%rbp)
movq  %rdx, -24(%rbp)
movq  -16(%rbp), %rax
movl  %eax, %edx
movq  -24(%rbp), %rax
imull %edx, %eax
movq  -8(%rbp), %rdx
addl  %edx, %eax
popq  %rbp
ret
```



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.

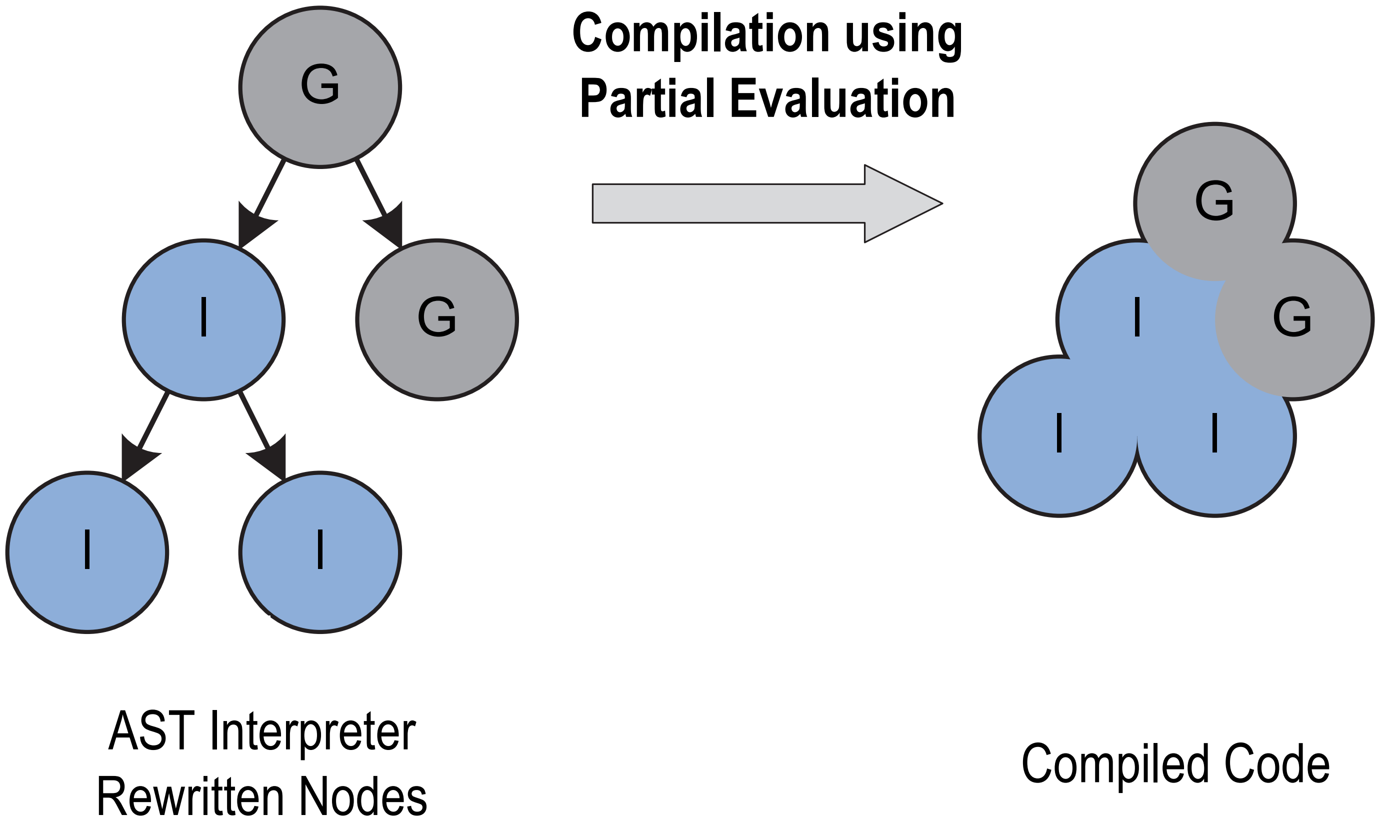
Node Rewriting for Profiling Feedback



AST Interpreter
Uninitialized Nodes

Node Transitions

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.

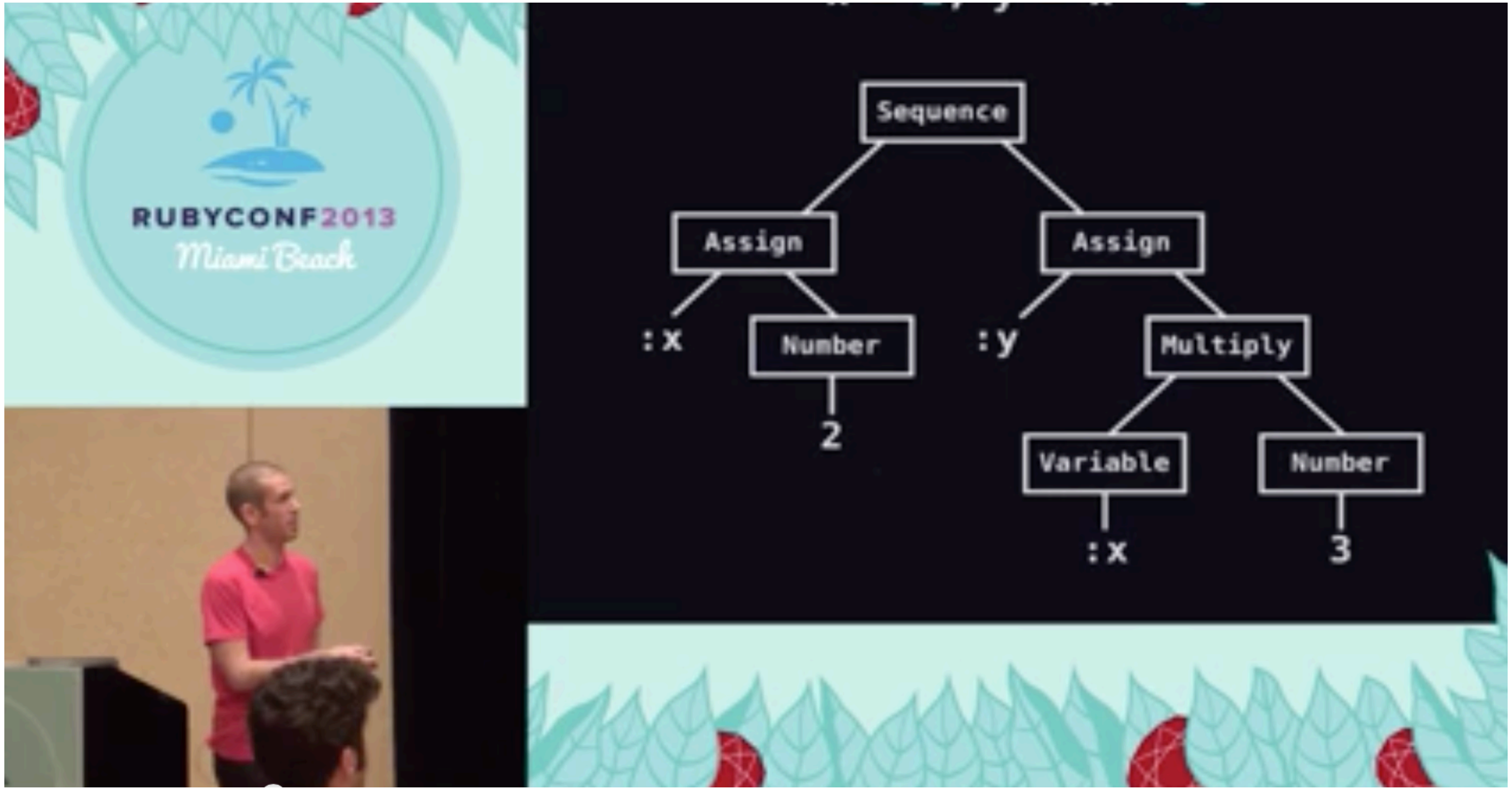


Compilation using Partial Evaluation

AST Interpreter
Rewritten Nodes

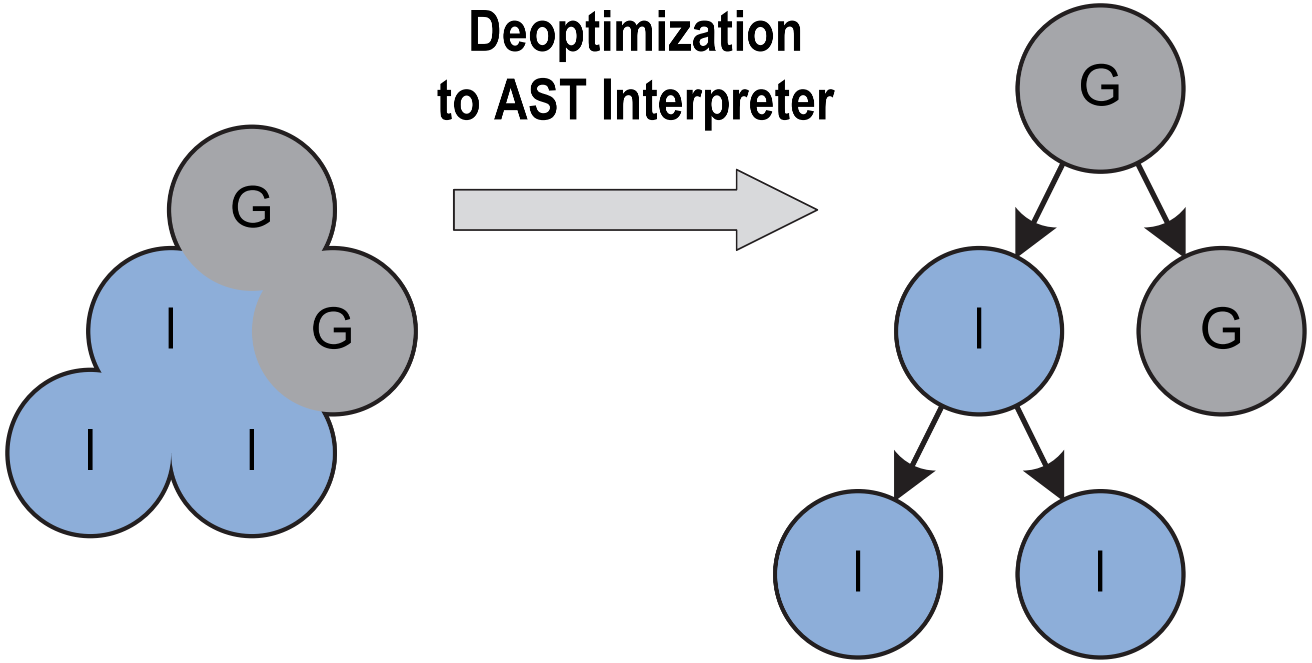
Compiled Code

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.



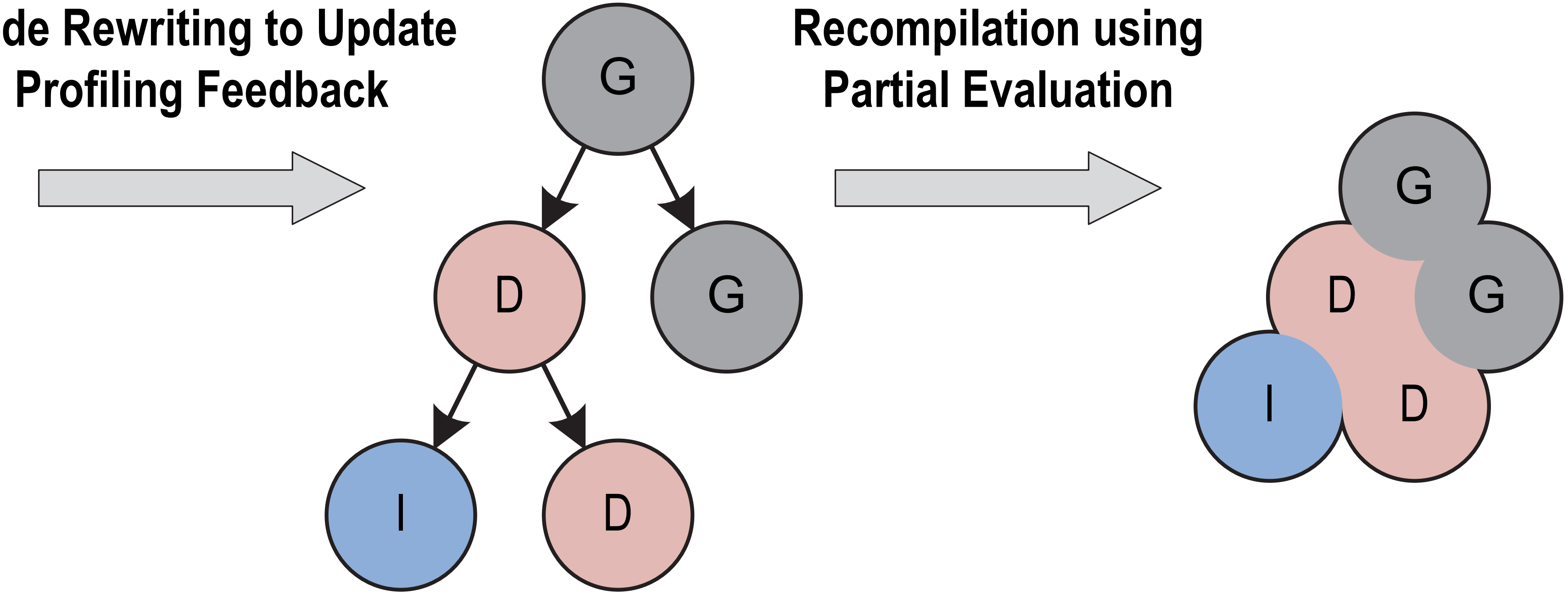
codon.com/compilers-for-free

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



Deoptimization to AST Interpreter

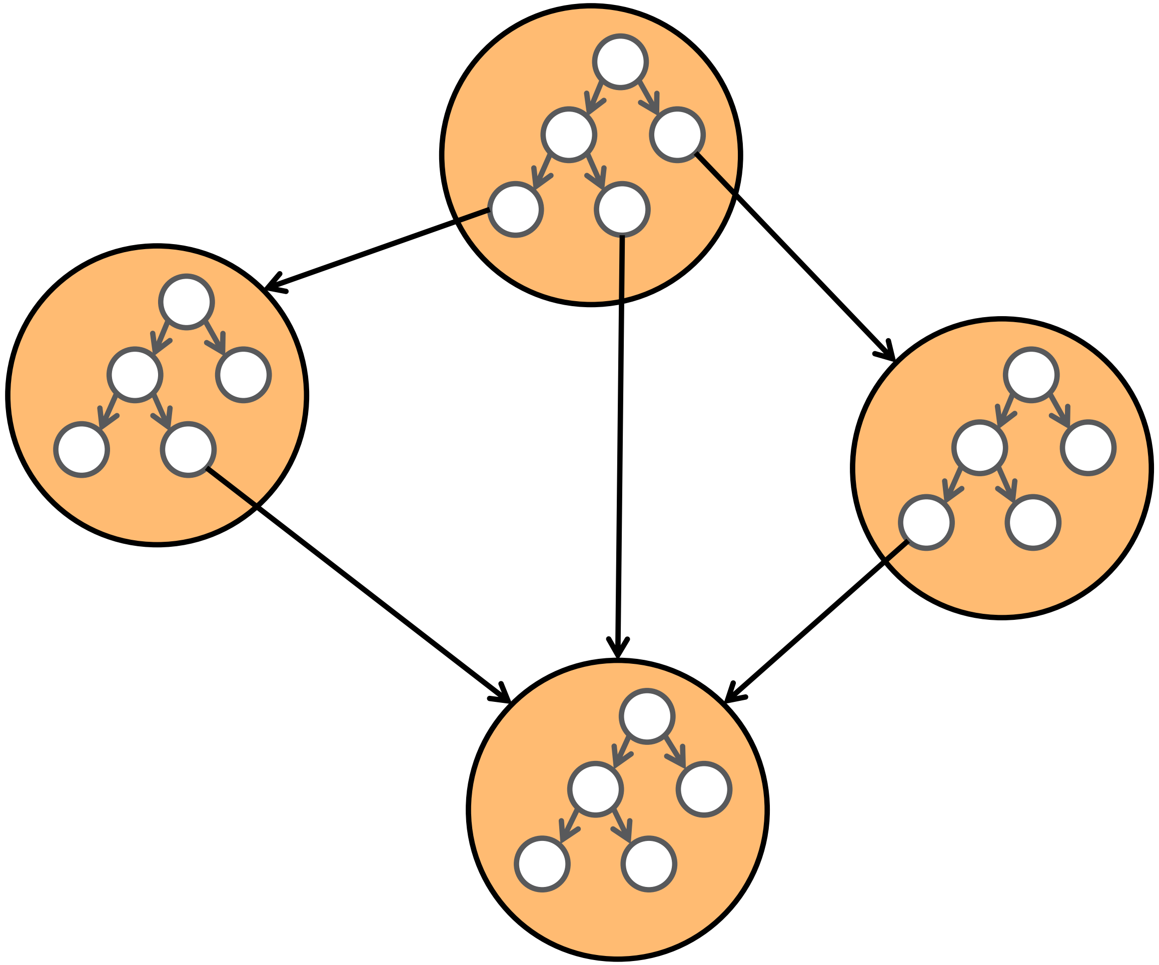
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.

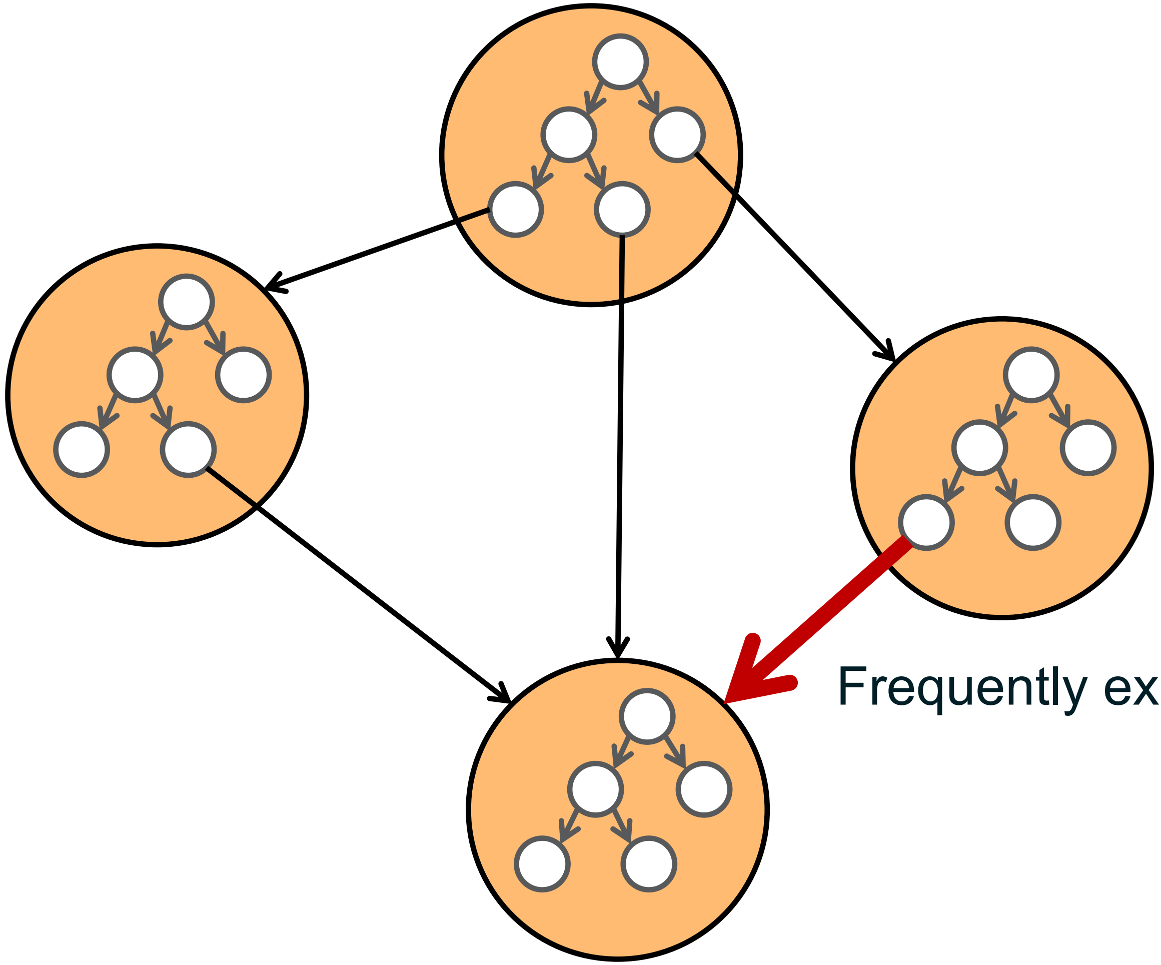


Node Rewriting to Update Profiling Feedback

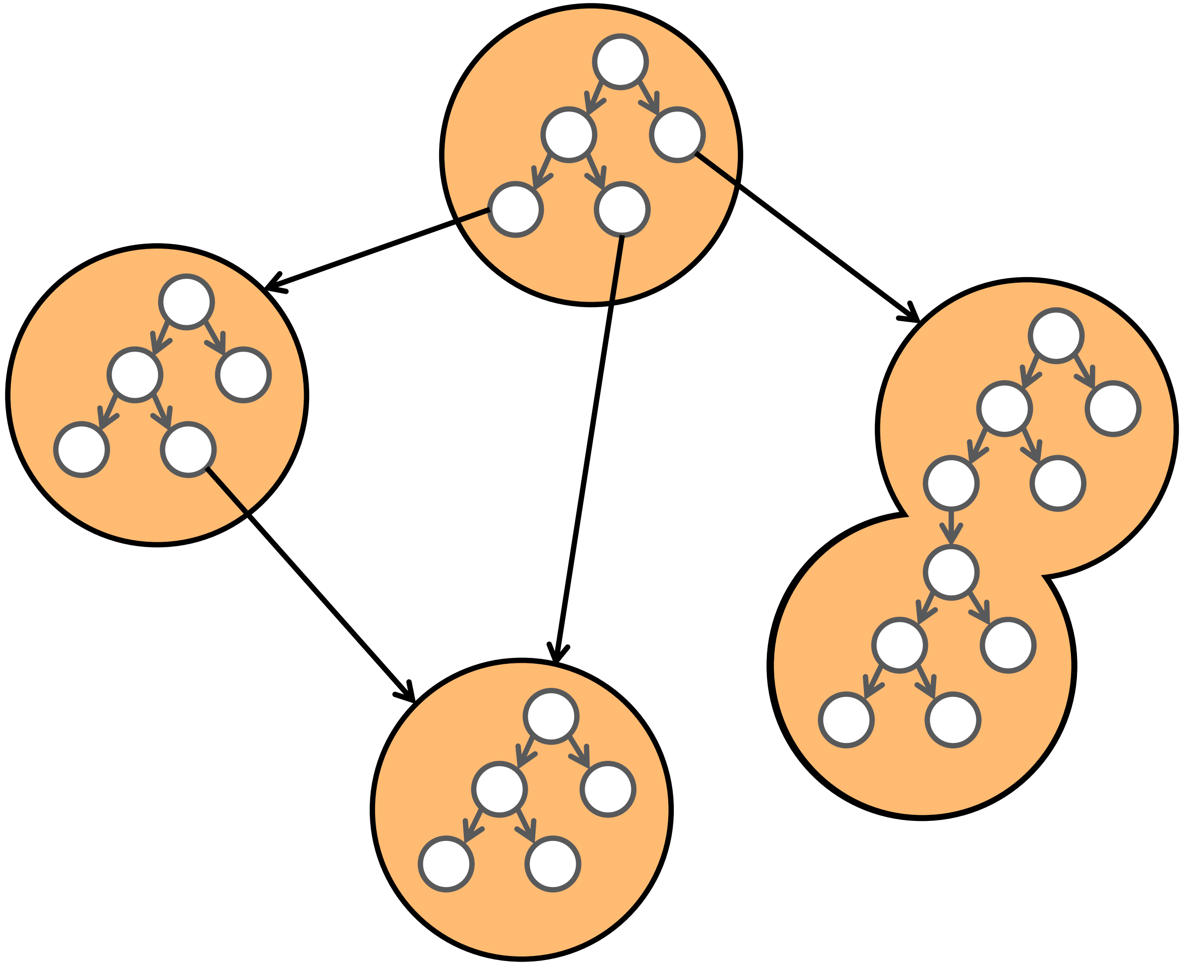
Recompilation using Partial Evaluation

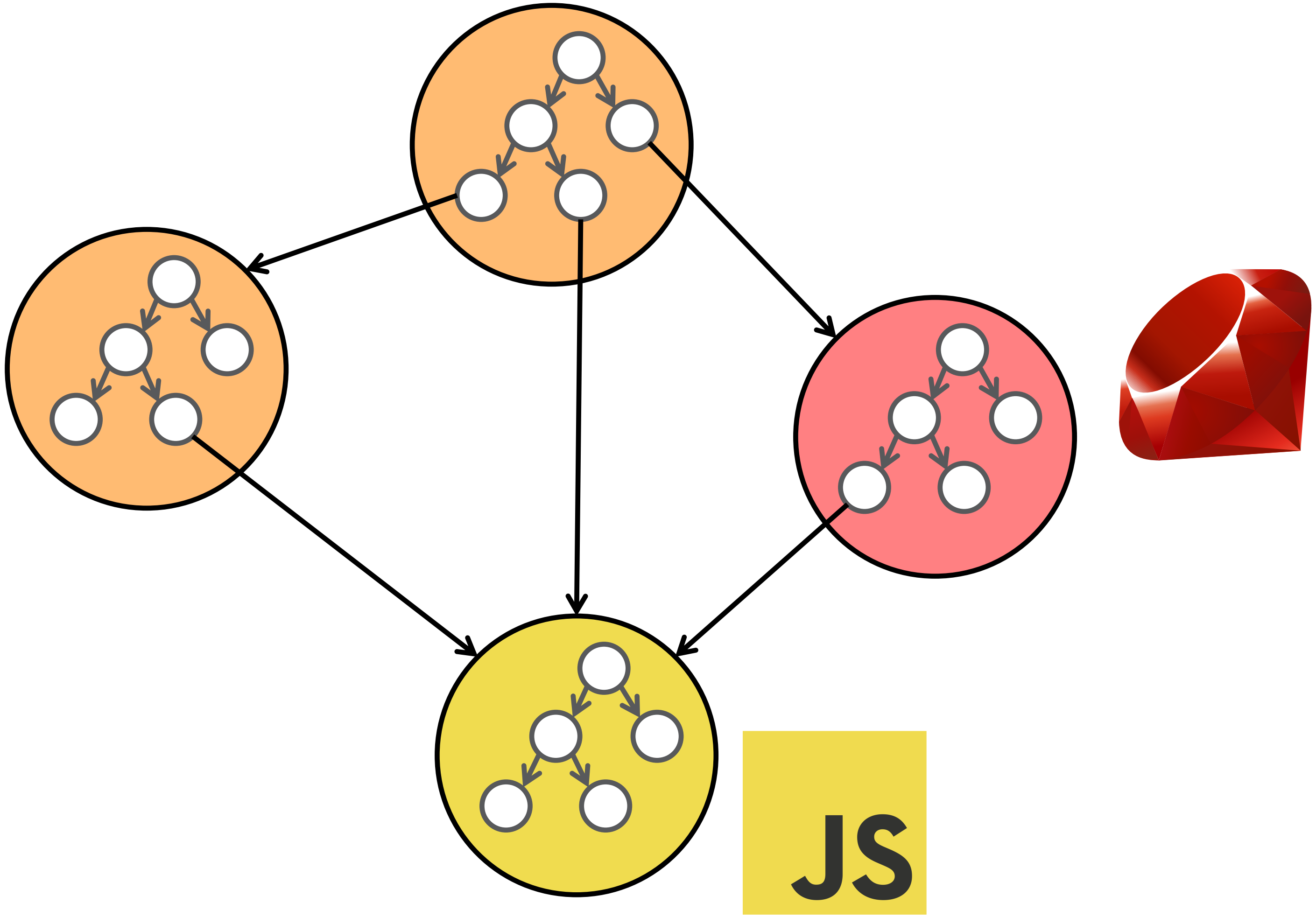
T. Würthinger, C. Wimmer, A. Wöß, L. Stadler, G. Duboscq, C. Humer, G. Richards, D. Simon, and M. Wolczko. One VM to rule them all. In Proceedings of Onward!, 2013.

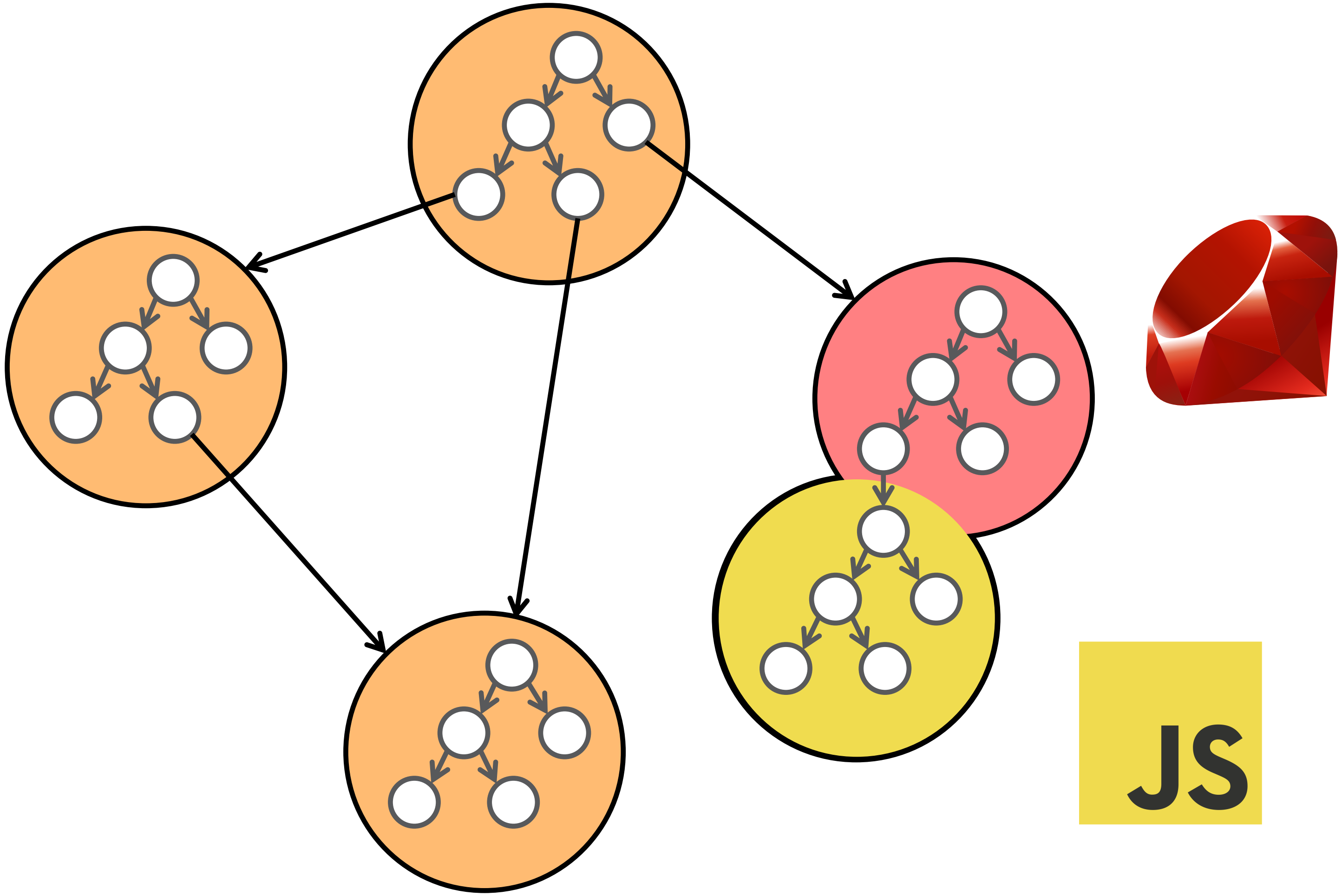


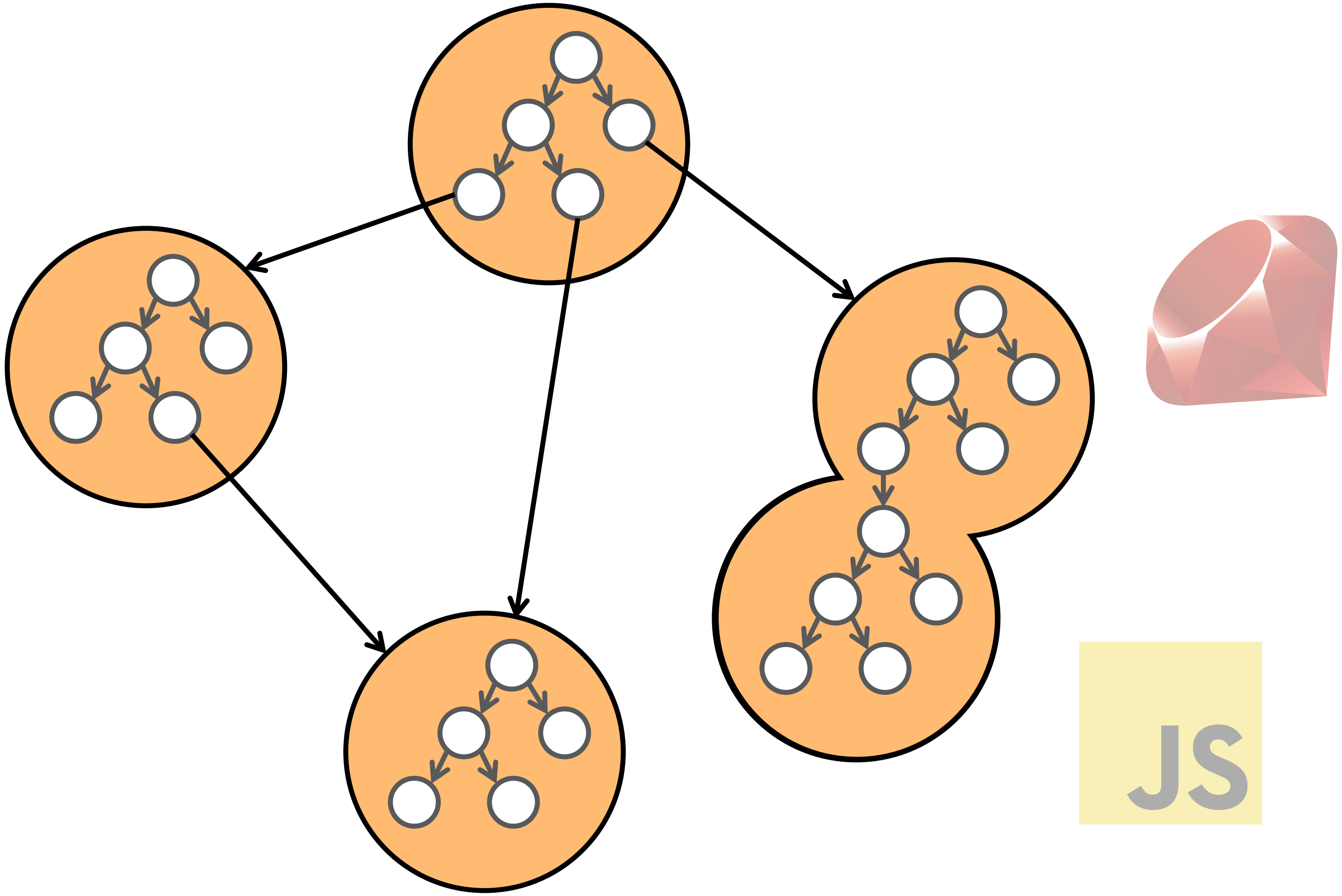


Frequently executed call




JS


JS


JS

Looking at how effective this is

```
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 this loop 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
```

```ruby
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
```

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

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

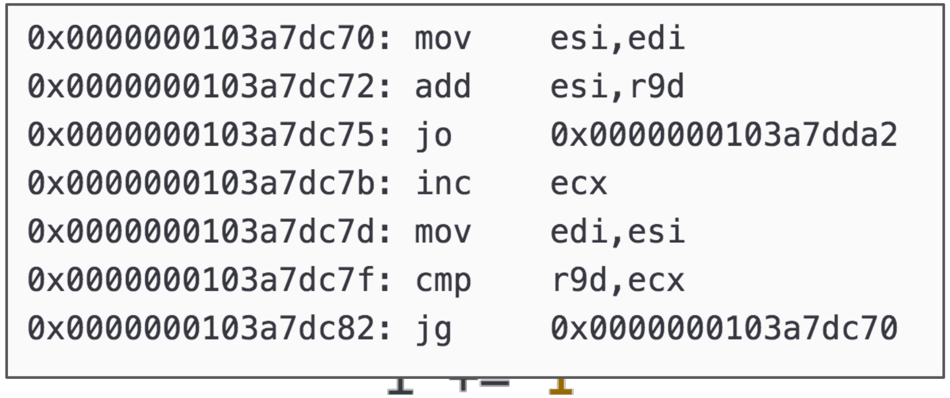
```
0x000000010ca4ad90: mov    eax,r11d
0x000000010ca4ad93: add    eax,r14d
0x000000010ca4ad96: jo     0x000000010ca4ae68
0x000000010ca4ad9c: inc    r10d
0x000000010ca4ad9f: mov    r11d,eax
0x000000010ca4ada2: cmp    r14d,r10d
0x000000010ca4ada5: jg     0x000000010ca4ad90
```

```ruby
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
```

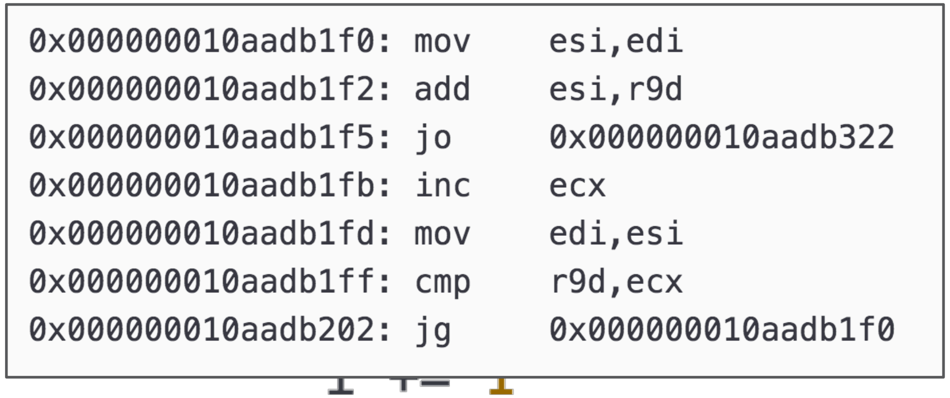
```
def add(a, b)
  a + b
end
```



```
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
```

```
    i += 1
    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
```

```
    i += 1
    a = add(a, n)
  end
  a
end
```

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

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

```
0x0000000103a7dc
0x0000000103a7dc
0x0000000103a7dc
0x0000000103a7dc
0x0000000103a7dc
0x0000000103a7dc
0x0000000103a7dc
```

```
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
```

```
esi,edi
esi,r9d
0x000000010aadb322
ecx
edi,esi
r9d,ecx
0x000000010aadb1f0
```

```
(a, n)
```

```
  end
  a
end
```

```
  end
  a
end
```

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

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

```
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
```

```
0x0000000103a7dc
0x0000000103a7dc
0x0000000103a7dc
0x0000000103a7dc
0x0000000103a7dc
0x0000000103a7dc
0x0000000103a7dc
```

```
esi,edi
esi,r9d
0x000000010aadb322
ecx
edi,esi
r9d,ecx
0x000000010aadb1f0
```

```
(a, n)
```

```
  end
  a
end
```

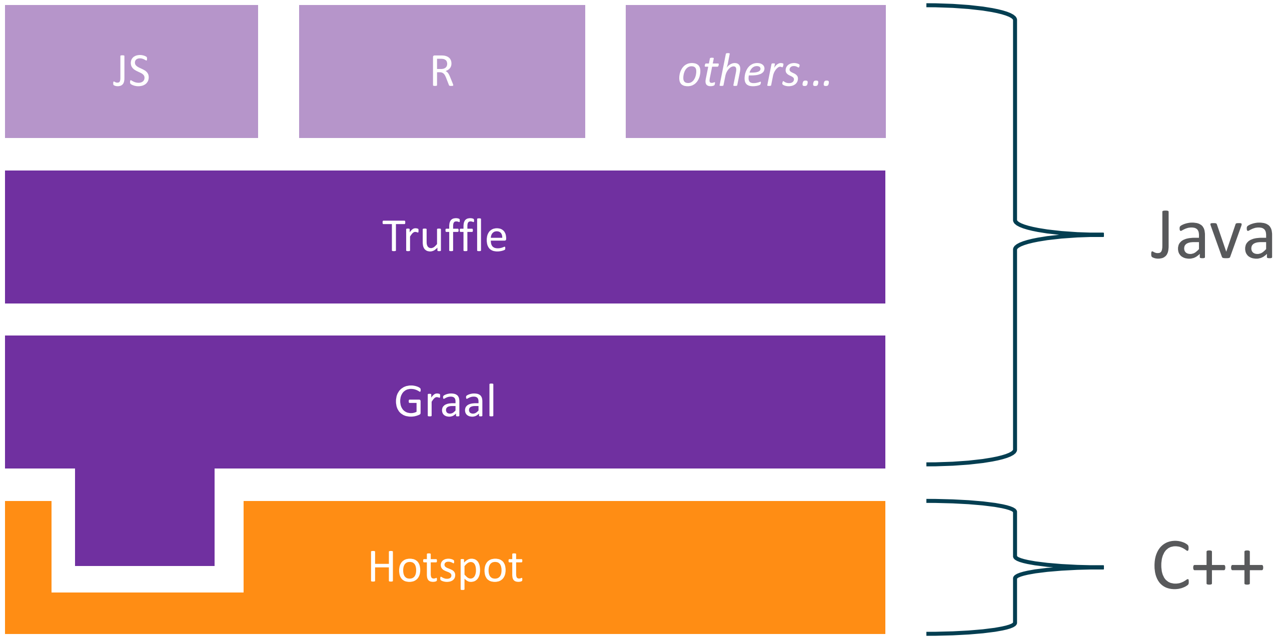
```
  end
  a
end
```

How to use GraalVM

GraalVM – everything in one package today

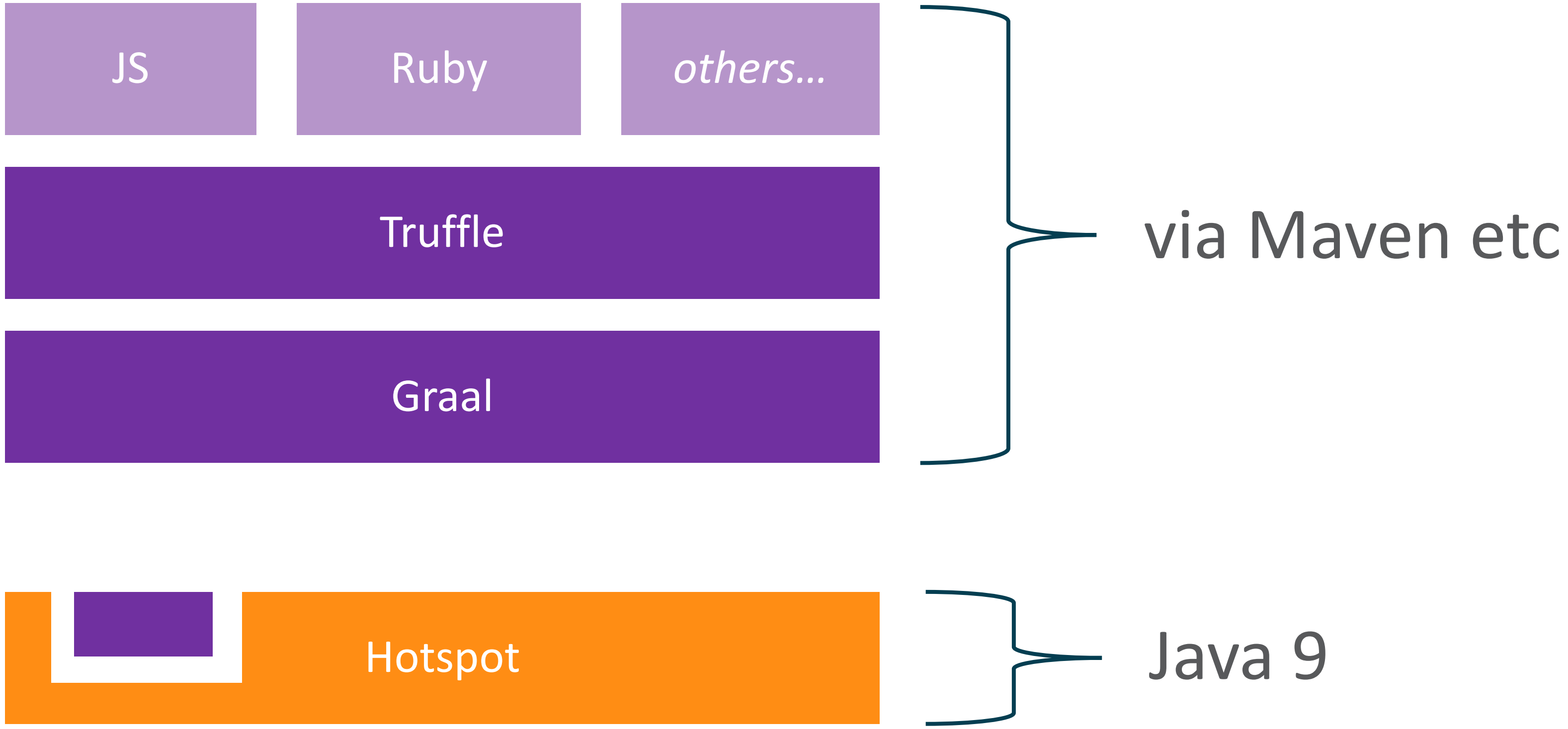
- Includes:
  - JVM (RE or DK)
  - Java
  - JavaScript
  - Ruby
  - R
  - More in the future
- Binary tarball release
- Mac or Linux

Java 9 – runs on an unmodified JVM

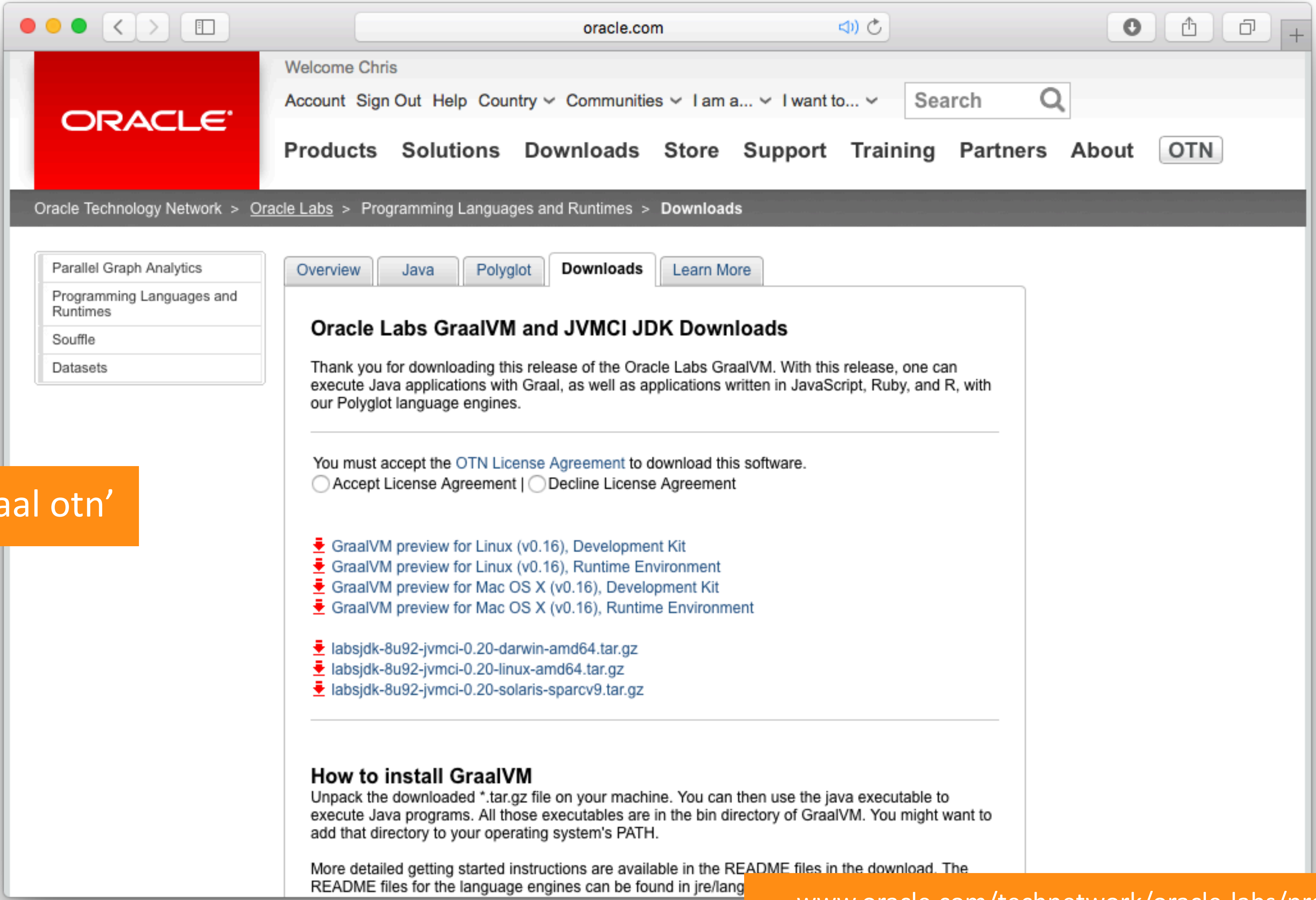


JVMCI
(JVM Compiler Interface)

Java 9 – runs on an unmodified JVM



Where to find more information



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

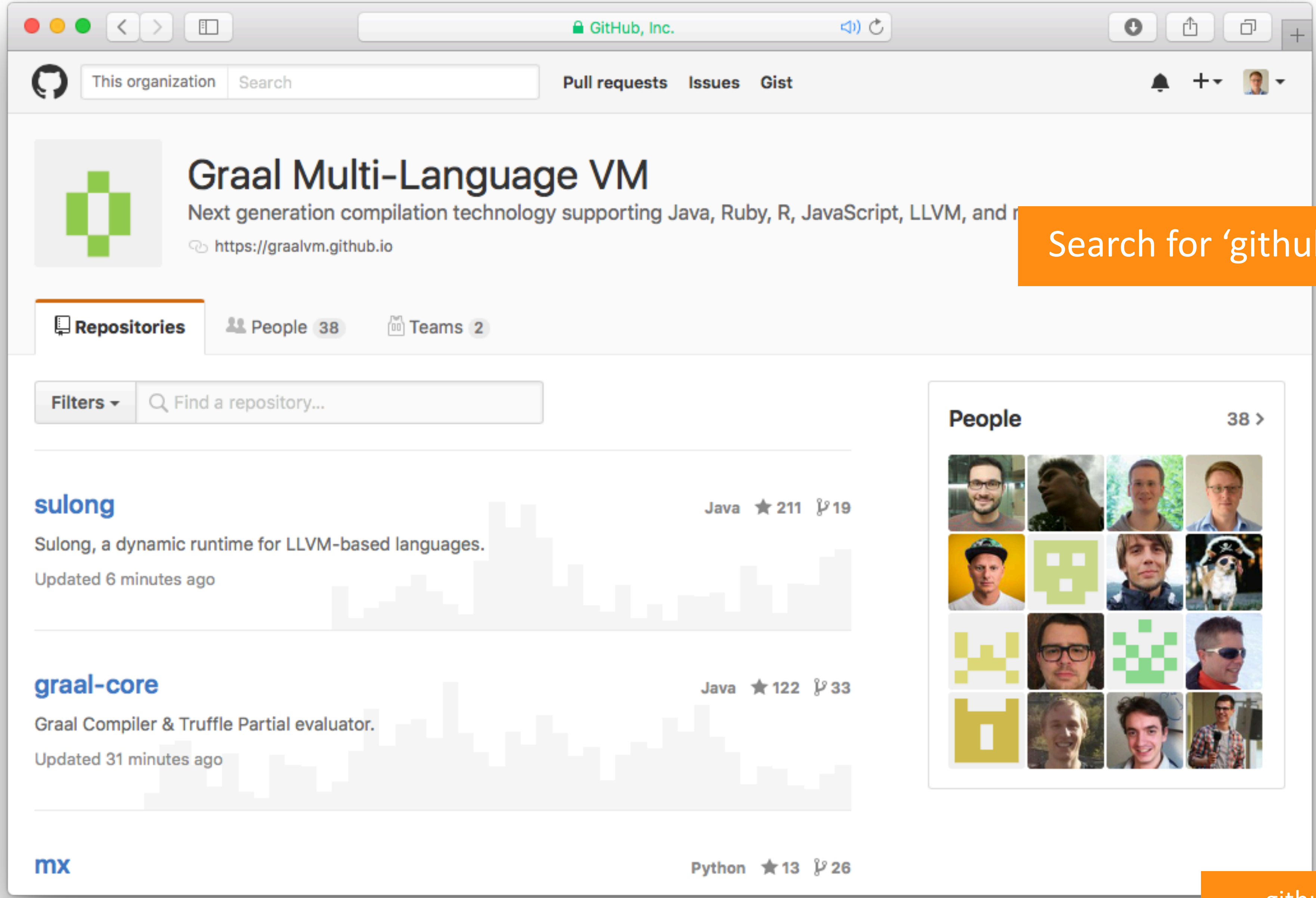
Unpack the downloaded *.tar.gz file on your machine. You can then use the java executable to execute Java programs. All those executables are in the bin directory of GraalVM. You might want to add that directory to your operating system's PATH.

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

Search for ’graal otn’

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

JavaOne™ ORACLE



GitHub, Inc.

This organization Search Pull requests Issues Gist

Graal Multi-Language VM

Next generation compilation technology supporting Java, Ruby, R, JavaScript, LLVM, and r

https://graalvm.github.io

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.

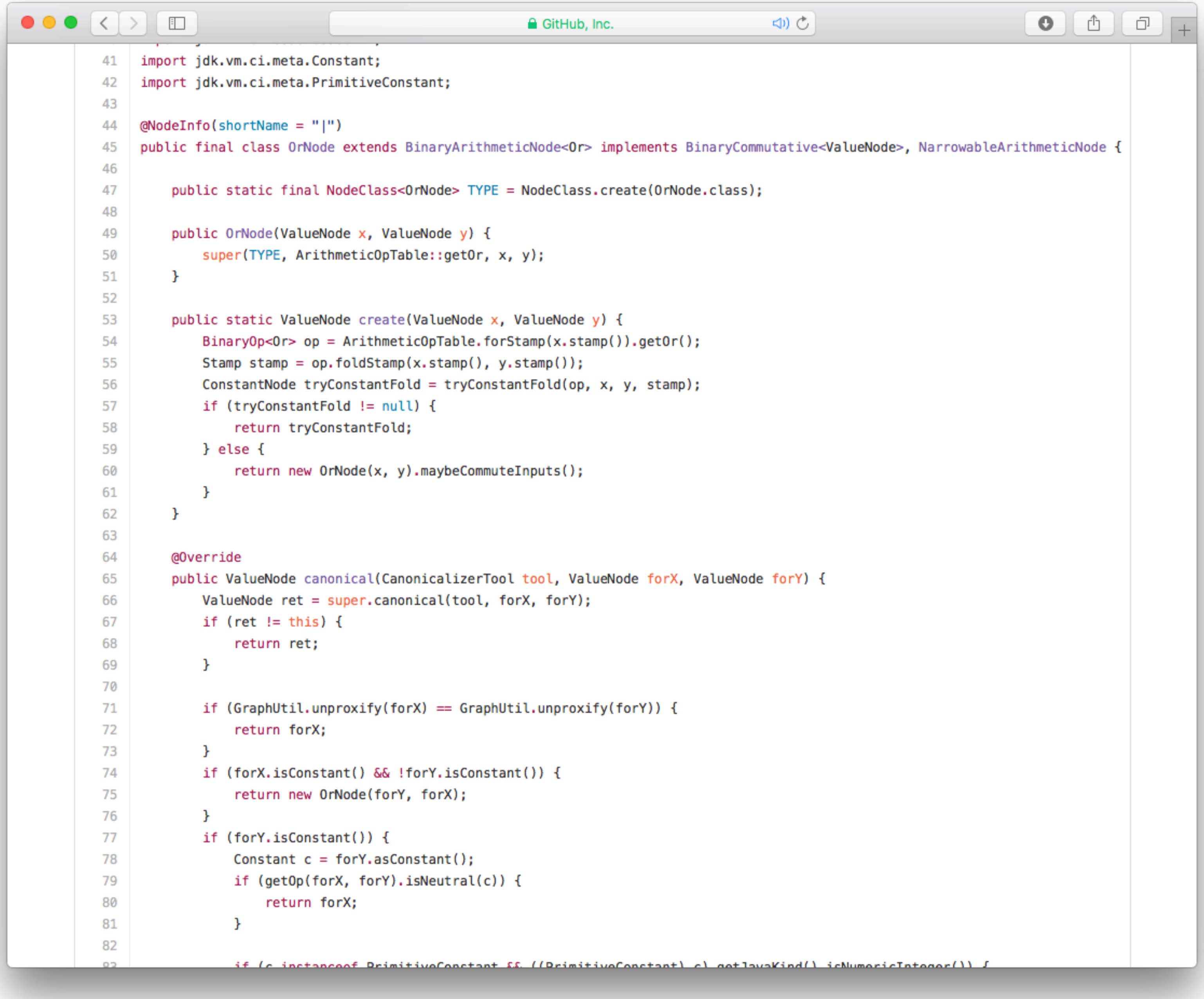
Updated 31 minutes ago

mx Python ★ 13 26

People 38 >

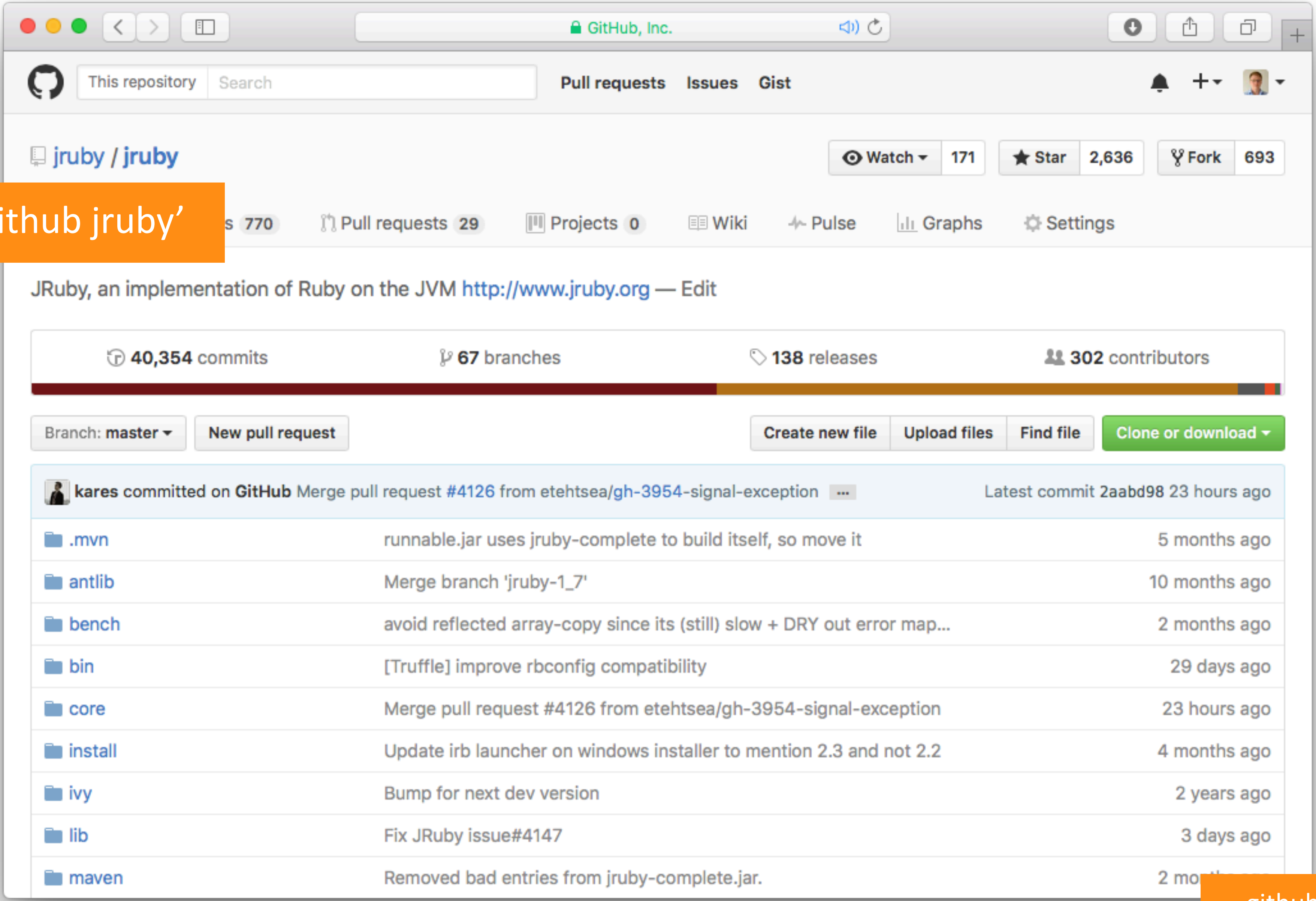
Search for ‘github graalvm’

github.com/graalvm



```java
import jdk.vm.ci.meta.Constant;
import jdk.vm.ci.meta.PrimitiveConstant;

@NodeInfo(shortName = "|")
public final class OrNode extends BinaryArithmeticNode<Or> implements BinaryCommutative<ValueNode>, NarrowableArithmeticNode {

    public static final NodeClass<OrNode> TYPE = NodeClass.create(OrNode.class);

    public OrNode(ValueNode x, ValueNode y) {
        super(TYPE, ArithmeticOpTable::getOr, x, y);
    }

    public static ValueNode create(ValueNode x, ValueNode y) {
        BinaryOp<Or> op = ArithmeticOpTable.forStamp(x.stamp()).getOr();
        Stamp stamp = op.foldStamp(x.stamp(), y.stamp());
        ConstantNode tryConstantFold = tryConstantFold(op, x, y, stamp);
        if (tryConstantFold != null) {
            return tryConstantFold;
        } else {
            return new OrNode(x, y).maybeCommuteInputs();
        }
    }

    @Override
    public ValueNode canonical(CanonicalizerTool tool, ValueNode forX, ValueNode forY) {
        ValueNode ret = super.canonical(tool, forX, forY);
        if (ret != this) {
            return ret;
        }

        if (GraphUtil.unproxify(forX) == GraphUtil.unproxify(forY)) {
            return forX;
        }
        if (forX.isConstant() && !forY.isConstant()) {
            return new OrNode(forY, forX);
        }
        if (forY.isConstant()) {
            Constant c = forY.asConstant();
            if (getOp(forX, forY).isNeutral(c)) {
                return forX;
            }

            if (c instanceof PrimitiveConstant && ((PrimitiveConstant) c).getJavaKind().isNumericInteger()) {
```

Search for ‘github jruby’



jruby / jruby

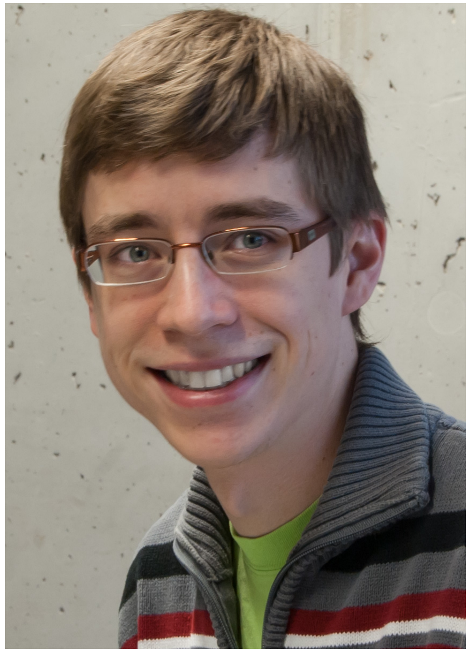
JRuby, an implementation of Ruby on the JVM http://www.jruby.org — Edit

40,354 commits | 67 branches | 138 releases | 302 contributors

kares committed on GitHub Merge pull request #4126 from etehtsea/gh-3954-signal-exception … Latest commit 2aabd98 23 hours ago

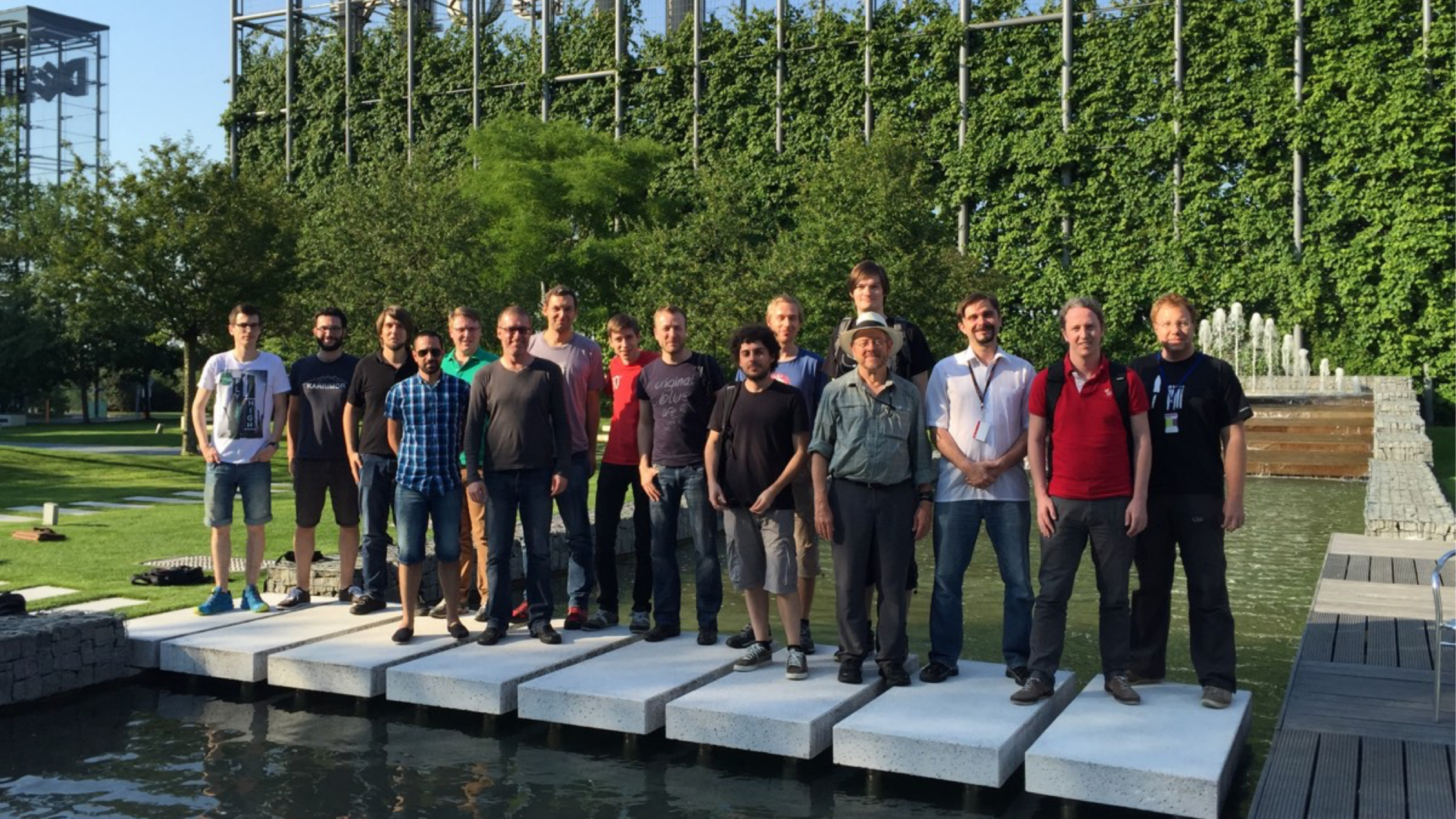
.mvn	runnable.jar uses jruby-complete to build itself, so move it	5 months ago
antlib	Merge branch 'jruby-1_7'	10 months ago
bench	avoid reflected array-copy since its (still) slow + DRY out error map…	2 months ago
bin	[Truffle] improve rbconfig compatibility	29 days ago
core	Merge pull request #4126 from etehtsea/gh-3954-signal-exception	23 hours ago
install	Update irb launcher on windows installer to mention 2.3 and not 2.2	4 months ago
ivy	Bump for next dev version	2 years ago
lib	Fix JRuby issue#4147	3 days ago
maven	Removed bad entries from jruby-complete.jar.	2 mo

github.com/jruby/jruby



Polyglot on the JVM with Graal [CON4553]

Tuesday, Sep 20, 5:30 p.m. - 6:30 p.m.
Hilton - Plaza Room A


KARRIMOR
FLY HIGH
original blue

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 Humer
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 Daloze
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.

JavaYour
(Next)

Integrated Cloud

Applications & Platform Services

JavaOne™

ORACLE®

ORACLE®