ORACLE

ORACLE

Polyglot programming in GraalVM™

with Tooling, AOT compilation, and Embedding

Petr Chalupa
Principal Member of Technical Staff
Oracle Labs

November 28, 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.

GraalVM Native Image technology (including SubstrateVM) is early adopter technology. It is available only under an early adopter license and remains subject to potentially significant further changes, compatibility testing and certification.

Program agenda

1. GraalVM
2. Truffle
3. Polyglot
4. Embedding and native image generation
5. Instrumentation and Tooling
6. Language status

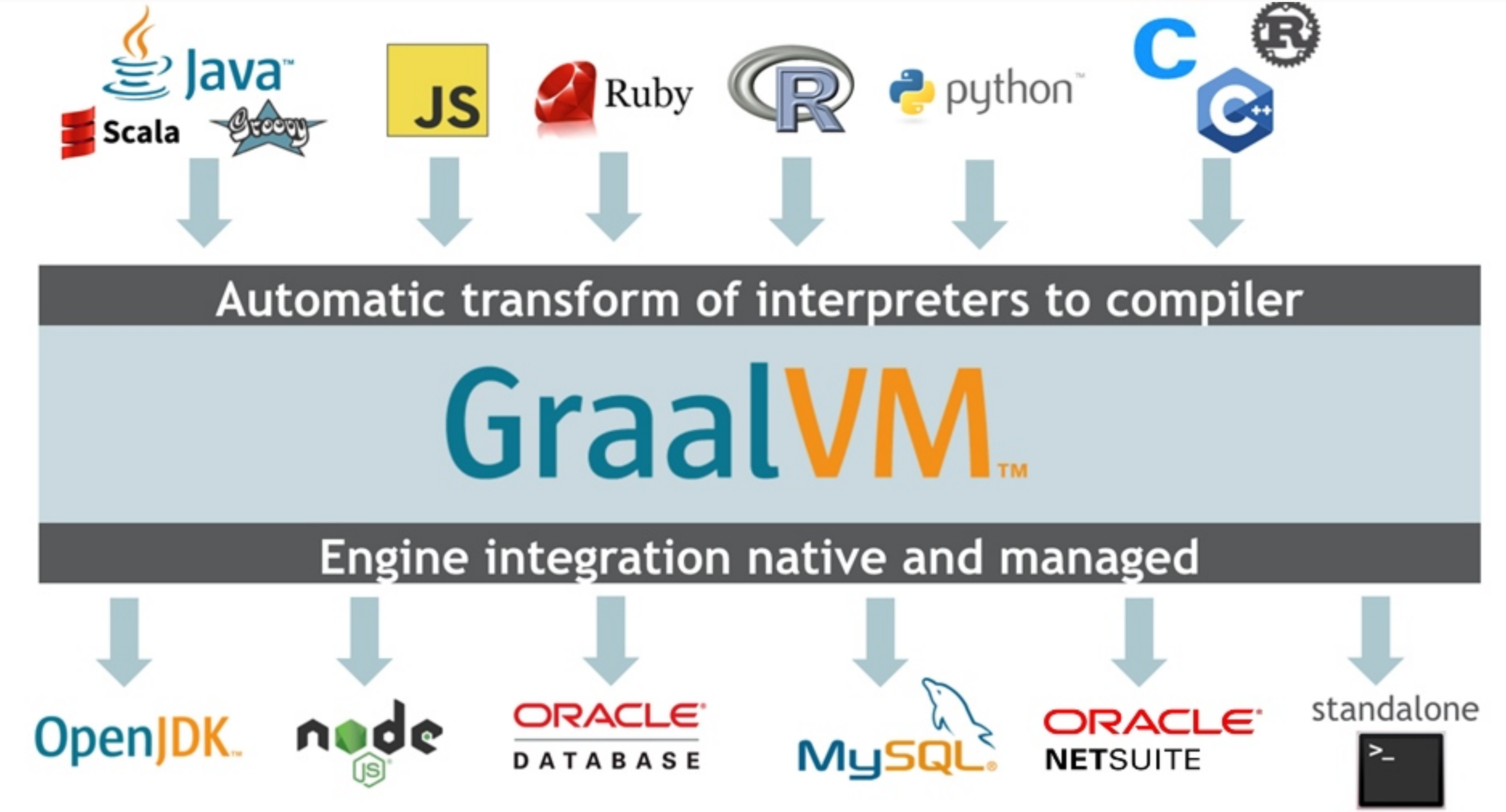
GraalVM™

What is GraalVM?

- Drop-in replacement for Oracle Java 8 and Java 11
  - Run your Java application faster
- Ahead-of-time compilation for Java
  - Create standalone binaries with low footprint
- High-performance JavaScript, Python, Ruby, R, …
  - The first VM for true polyglot programming
  - Implement your own language or DSL

What is Graal?

- A Java just in time compiler
  - At runtime, translates Java bytecodes into machine code
- Modern design, implemented in Java
  - Maintainable
  - Extendable
- Fully Java compliant => production ready
- High performance

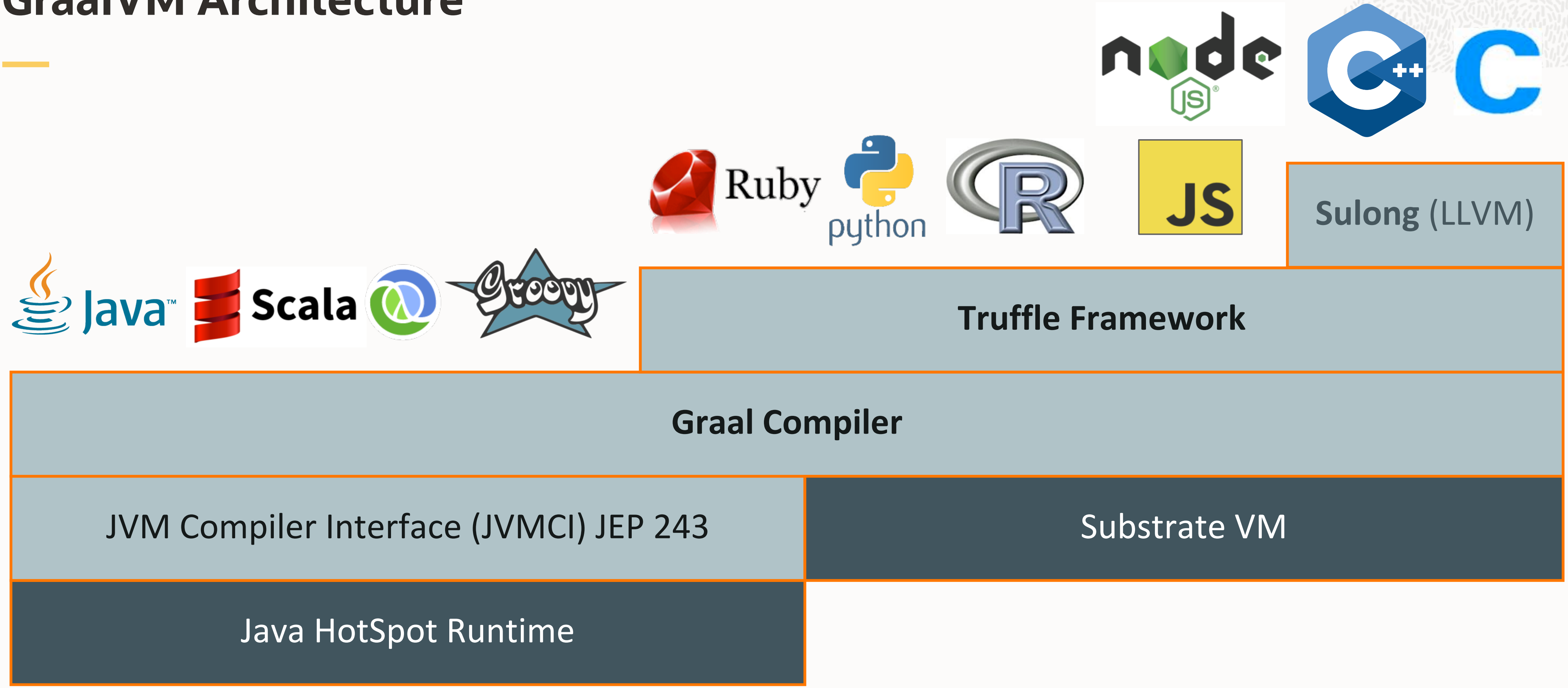


Automatic transform of interpreters to compiler

GraalVM

Engine integration native and managed

GraalVM Architecture



Sulong (LLVM)

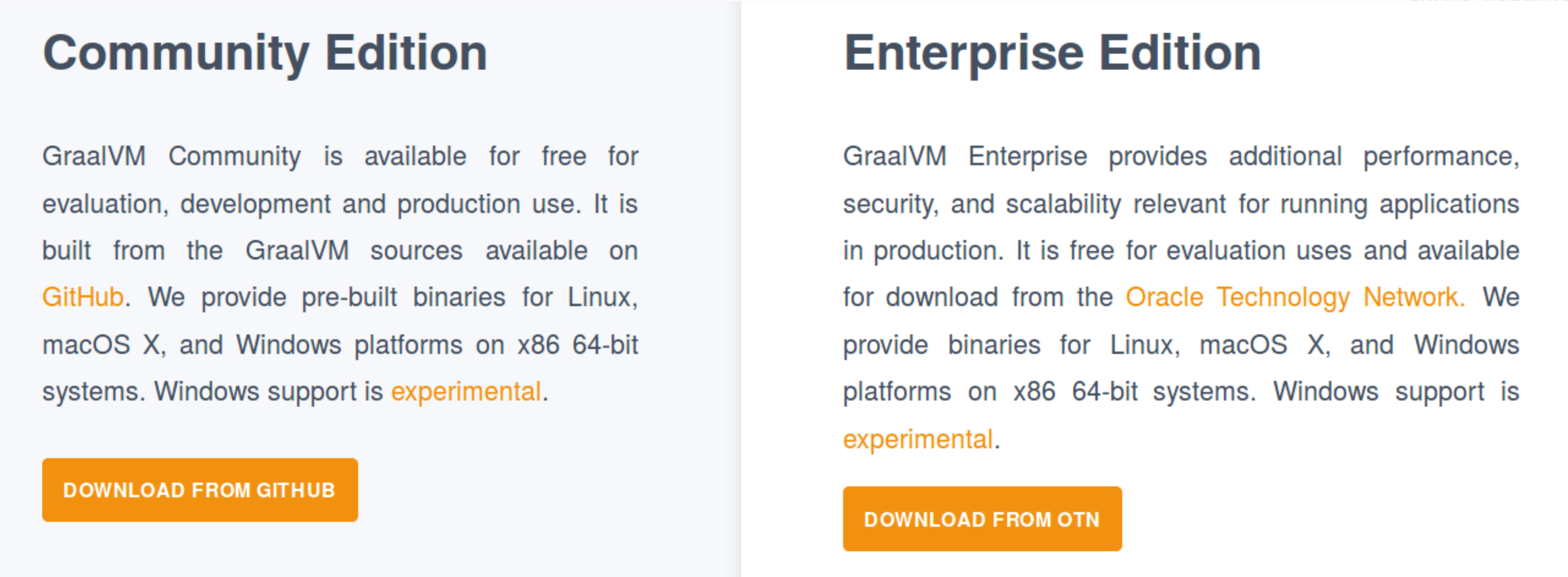
Truffle Framework

Graal Compiler

JVM Compiler Interface (JVMCI) JEP 243

Substrate VM

Java HotSpot Runtime



Community Edition

GraalVM Community is available for free for evaluation, development and production use. It is built from the GraalVM sources available on GitHub. We provide pre-built binaries for Linux, macOS X, and Windows platforms on x86 64-bit systems. Windows support is experimental.

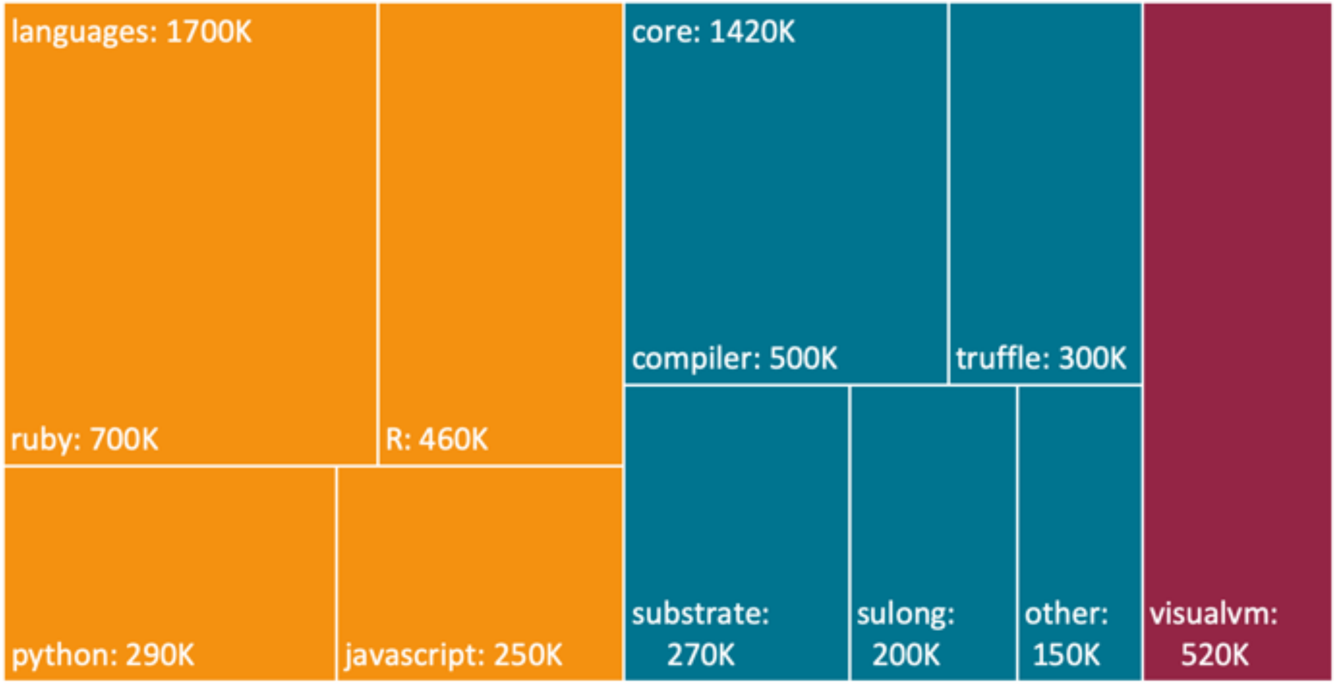
Enterprise Edition

GraalVM Enterprise provides additional performance, security, and scalability relevant for running applications in production. It is free for evaluation uses and available for download from the Oracle Technology Network. We provide binaries for Linux, macOS X, and Windows platforms on x86 64-bit systems. Windows support is experimental.

FREE on Oracle Cloud!

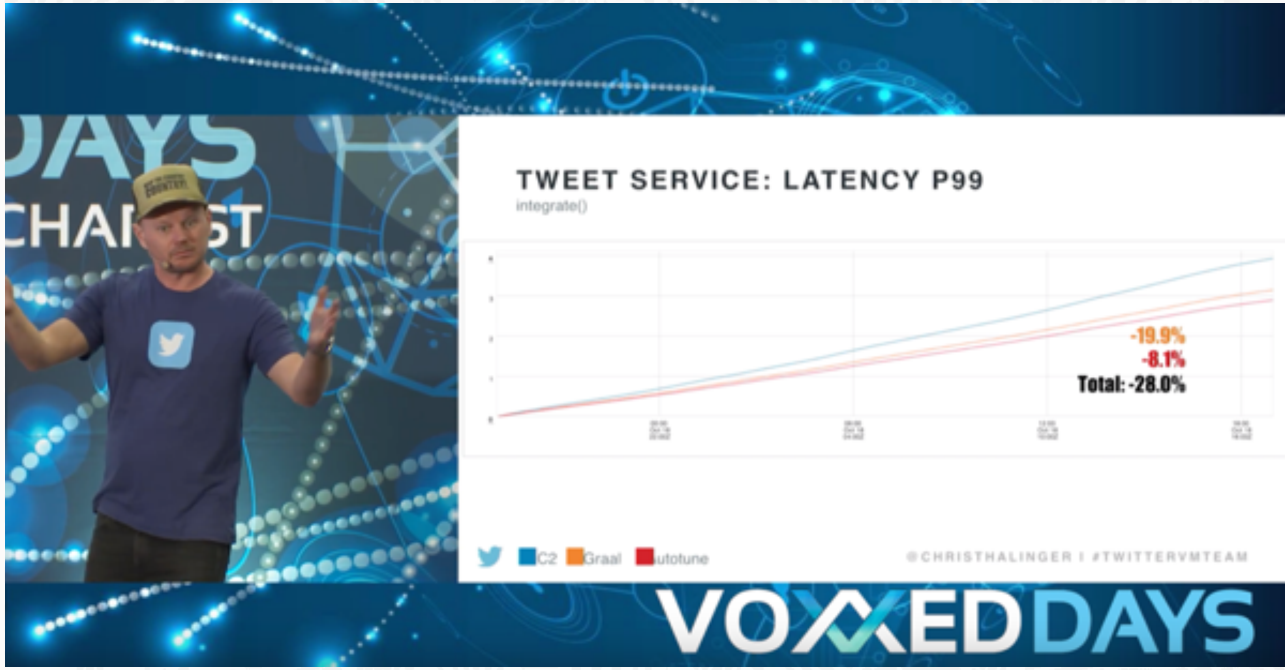
GraalVM Open Source

Open Source LOC actively maintained by GraalVM team



Total: 3,640,000 lines of code

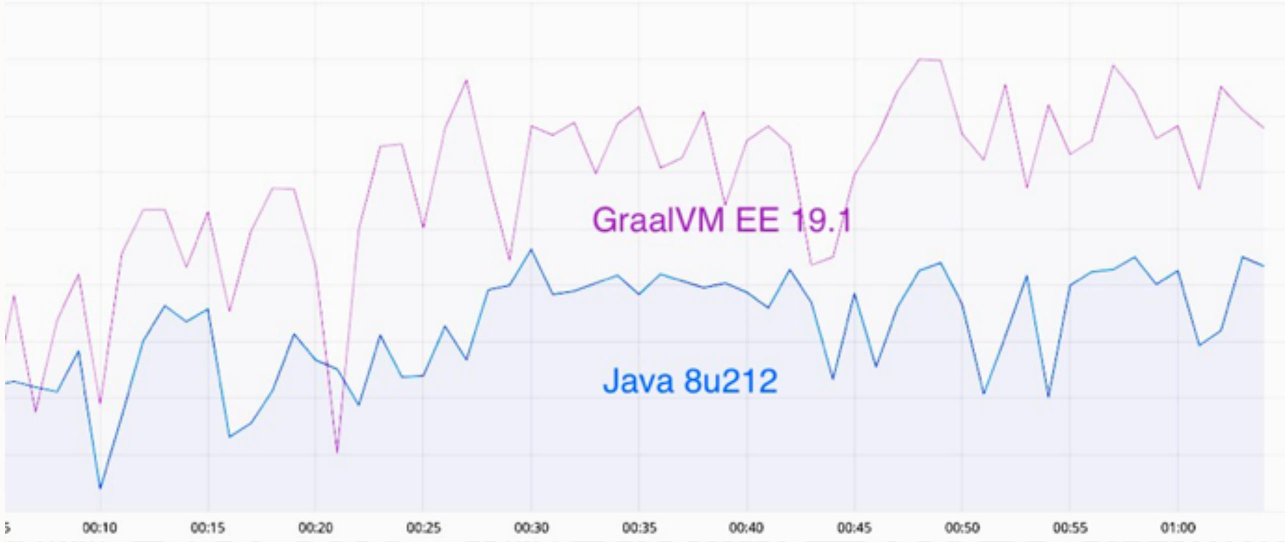
Twitter uses GraalVM compiler in production to run their Scala microservices



ORACLE

Cloud Infrastructure

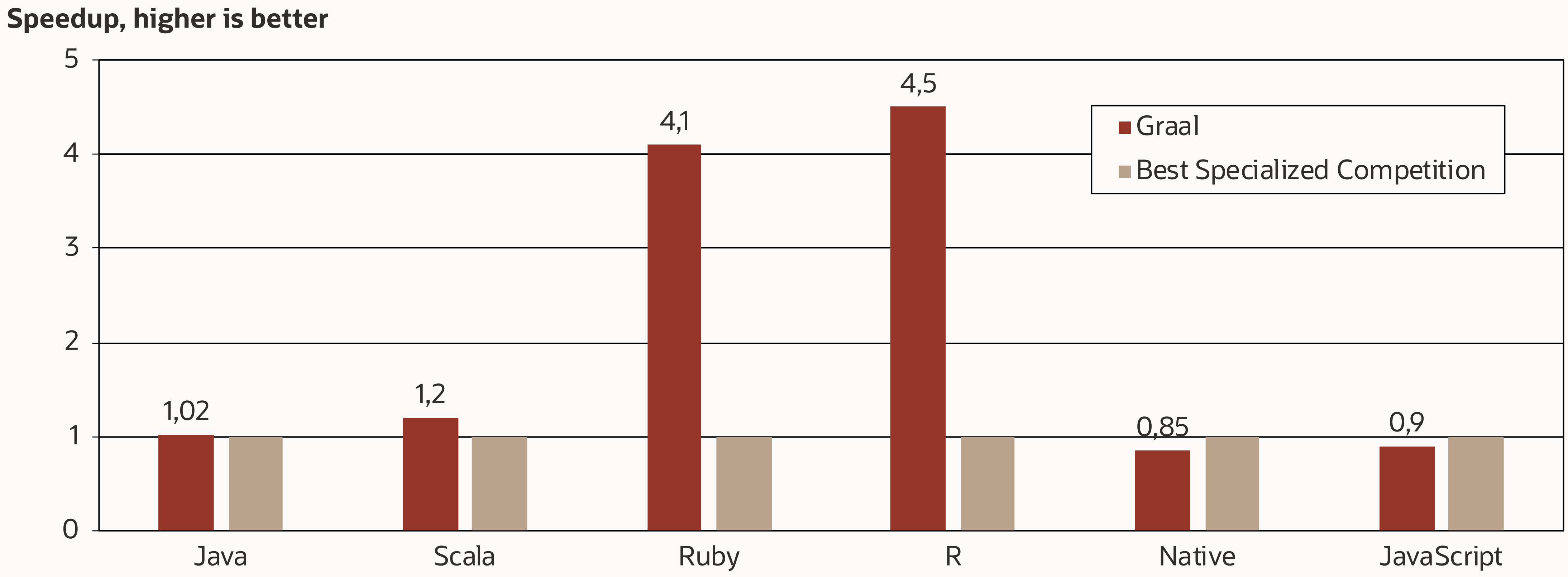
- Peak performance: +10%
- Garbage collection time: -25%
- Seamless migration



The rich ecosystem of CUDA-X libraries is now available for GraalVM applications.

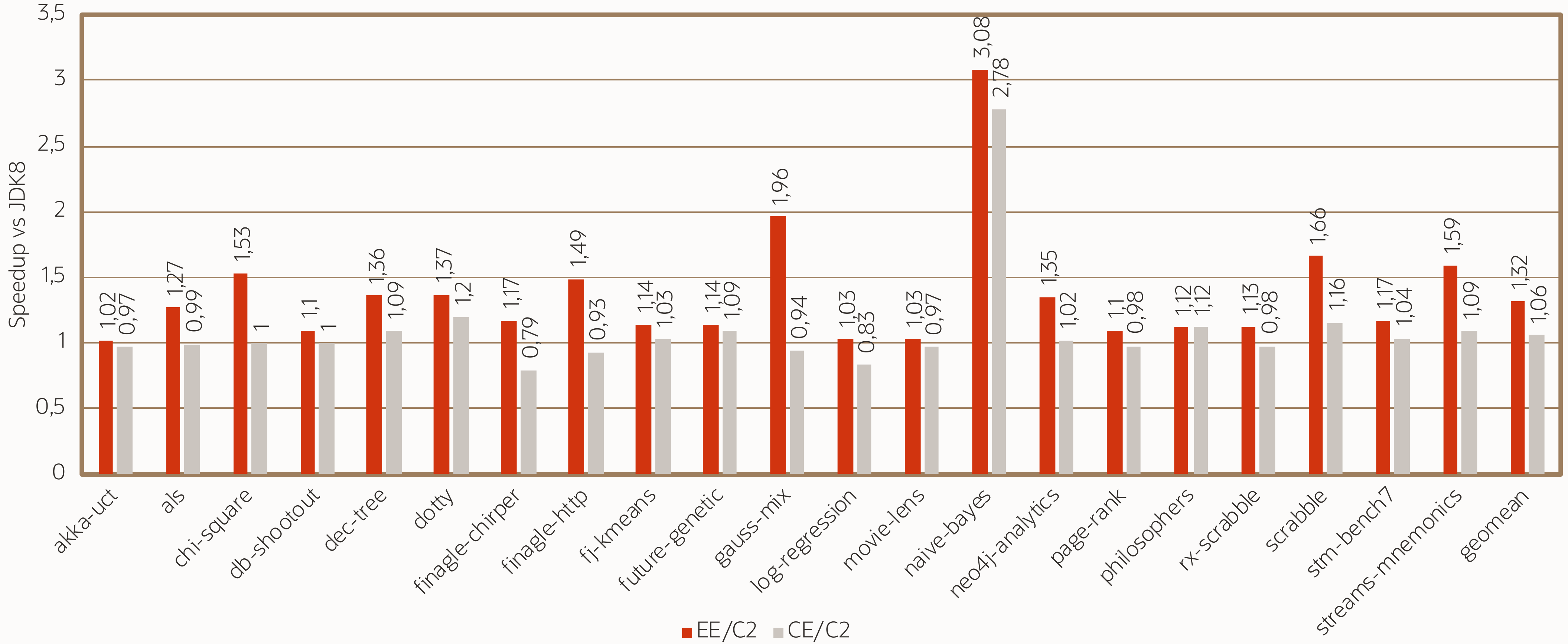
GPU kernels can be directly launched from GraalVM languages such as R, JavaScript, Scala and other JVM-based languages.

Performance: GraalVM Summary



Performance relative to:
HotSpot/Server, HotSpot/Server running JRuby, GNU R, LLVM AOT compiled, V8

GraalVM JIT Performance: Renaissance.dev

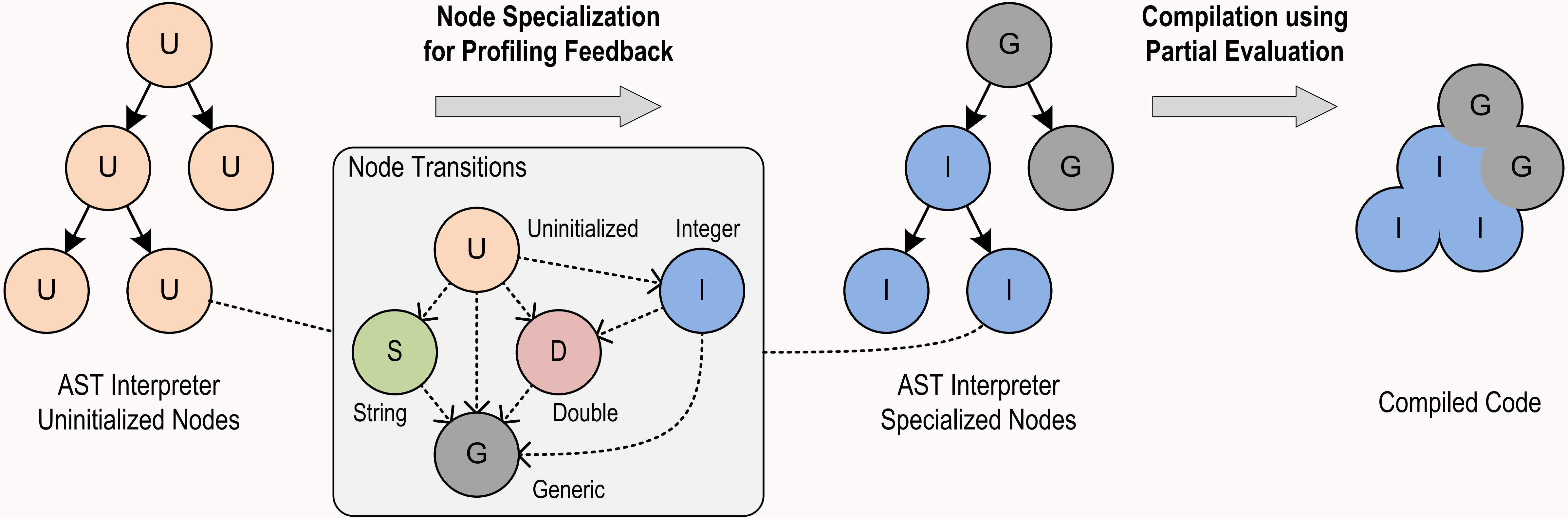


Truffle

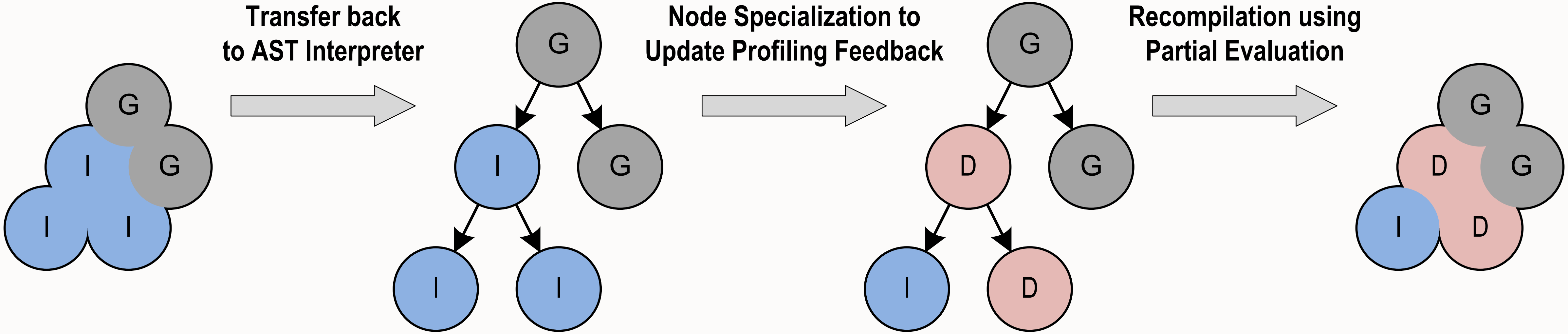
Truffle

- Self-optimizing abstract tree interpreter
  - Rewrites itself
- The simplest way to implement a language
- A Java library
  - Node classes
  - Execute methods

Speculate and Optimize ...



… and Transfer to Interpreter and Reoptimize!



Truffle

```java
@GenerateUncached
public abstract class IsFrozenNode extends RubyBaseWithoutContextNode {

    public static IsFrozenNode create() {
        return IsFrozenNodeGen.create();
    }

    public abstract boolean execute(Object object);

    @Specialization
    protected boolean isFrozen(boolean object) {
        return true;
    }

    @Specialization
    protected boolean isFrozen(int object) {
        return true;
    }

    @Specialization
    protected boolean isFrozen(long object) {
        return true;
    }

    @Specialization
    protected boolean isFrozen(double object) {
        return true;
    }

    @Specialization
    protected boolean isFrozen(
            DynamicObject object,
            @Cached ReadObjectFieldNode readFrozenNode) {
        return (boolean) readFrozenNode.execute(object, Layouts.FROZEN_IDENTIFIER, false);
    }

}
```

Polyglot

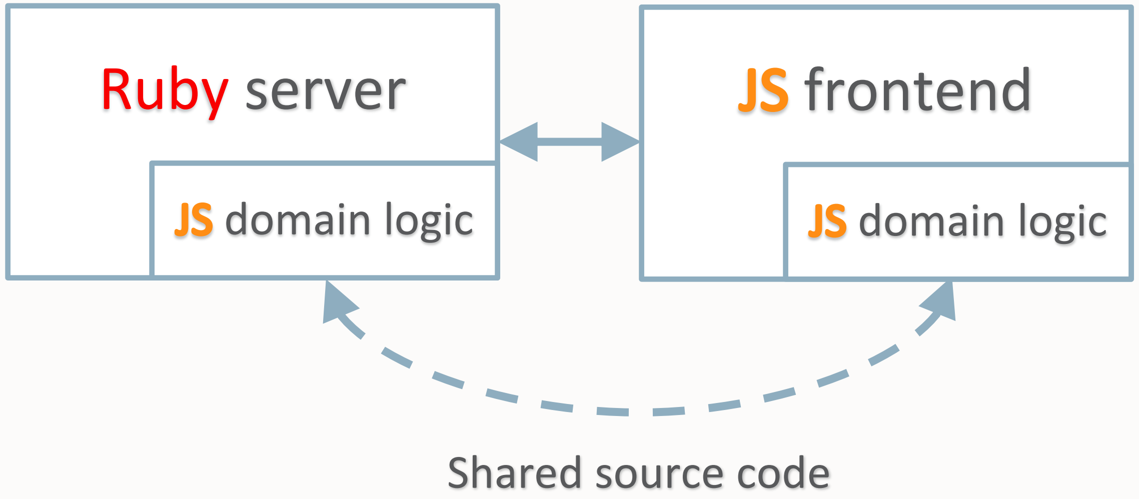
Use cases

Polyglot use-cases

- Missing library
  - You are not limited to libraries only written in your language of choice
- Legacy code
  - Migration of legacy project from language A to B can be gradual
- Sharing code
  - Share domain logic between frontend (JS) and server code (Java, Ruby, …)
- Execute on GPU
- User scripting
  - Run your user provided scaling rules
  - Run stored functions in DB
  - And more

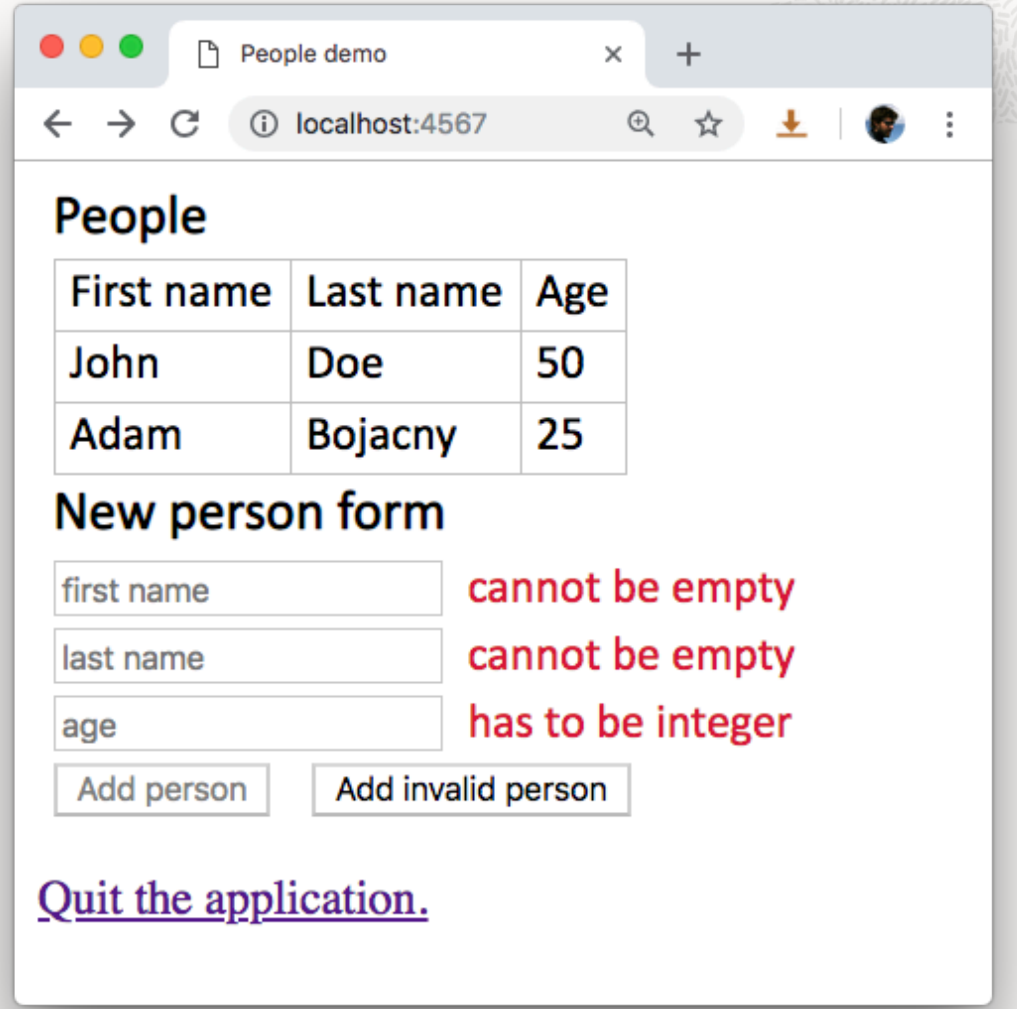
Web application demo – Ruby

- JavaScript frontend
- Ruby backend
  - In-memory database
- **Shared** domain logic written in JavaScript
  - No duplication
  - Identical behavior
  - The same validation code runs in the webserver and on the server



Web application demo – Ruby

- Single table of people
  - With first name, last name, and age
- A form to add new person
  - With immediate validations



People

First name	Last name	Age
John	Doe	50
Adam	Bojacny	25

New person form

first name cannot be empty

last name cannot be empty

age has to be integer

Quit the application.

Web application demo – The shared JS domain

```
function Person(firstName, lastName, age) {
    this.firstName = firstName;
    this.lastName  = lastName;
    this.age       = Number.parseInt(age);
    this.errors    = {};

    if (!this.firstName) { this.errors.firstName = 'cannot be empty'; }
    if (!this.lastName)  { this.errors.lastName  = 'cannot be empty'; }
    // ... age validation skipped (Integer, -1 > age < 151)

    Object.freeze(this);
}

Person.prototype.isInvalid = function () {
    return Object.keys(this.errors).length;
};
```

Web application demo – The Ruby server

```ruby
class PolyglotApp < Sinatra::Base
  # ... configuration skipped

  Polyglot.eval_file('js',
      File.join(File.dirname(__FILE__), 'public', 'person.js'))
  JsPerson = Polyglot.eval('js', 'Person')

  PEOPLE_DB = []
  def self.initialize_db
    PEOPLE_DB.clear
    PEOPLE_DB.push JsPerson.new('John', 'Doe', 50)
  end

  initialize_db
```

Load and evaluate JS file

Get the defined class by evaluation and store it in Ruby constant

Create new instance of Person defined in JS

Web application demo – The Ruby server

```
get '/' do
  File.read(File.join(self.class.public_folder, 'index.html'))
end

def js_person_as_hash(js_user)
  { firstName: js_user[:firstName],
    lastName:  js_user[:lastName],
    age:       js_user[:age] }
end

get '/people.json' do
  data = PEOPLE_DB.map { |js_user| js_person_as_hash(js_user) }
  data.to_json
end
```

Read fields from a JS object

Web application demo – The Ruby server

```
post '/person.json' do
  person_json          = JSON.parse request.body.read
  js_person            = JsPerson.fromObject OpenStruct.new person_json
  js_person_is_invalid = js_person.isInvalid

  if js_person_is_invalid > 0
    [403, 'Invalid user.']
  else
    PEOPLE_DB.push js_person
    [200, 'Ok']
  end
end
```

Call function on a JS object.
Creates new instance.

Call function on a JS object.
Checks the validity.

Polyglot

How does it work

Language independent interoperability messages

- Messages
  - Array like
    - HAS_SIZE, GET_SIZE, READ, WRITE
  - Members
    - HAS_KEYS, KEYS, INVOKE, READ, WRITE,
  - etc.
- Each language defines its behavior for these messages only
  - Otherwise there would be language combination explosion
  - Otherwise adding new languages would be hard

Idiomatic syntax translates to interoperability messages

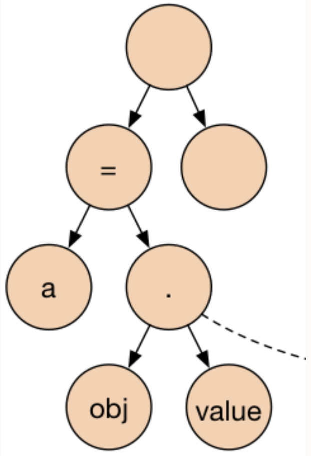
```
js_function = Polyglot.eval('js', <<-JS)
    function(arg) { return [1,2,3,5,8].slice(0,arg); }
JS
js_array    = js_function.call 4       # EXECUTE
puts js_array.size # => 4              # GET_SIZE
puts js_array[3]   # => 5              # READ


JavaIntArray      = Java.type 'int[]'
java_int_array    = JavaIntArray.new 1 # NEW
puts java_int_array.size  # => 1       # GET_SIZE
java_int_array[0] = 42                 # WRITE
puts java_int_array[0]    # => 42      # READ
```

Explicit message usage when necessary

- It’s not possible to get a size of an array in C
- Therefore there is always a way to call the messages explicitly
  - bool polyglot_has_array_elements(const void *value);
  - uint64_t polyglot_get_array_size(const void *array);
  - array[idx];

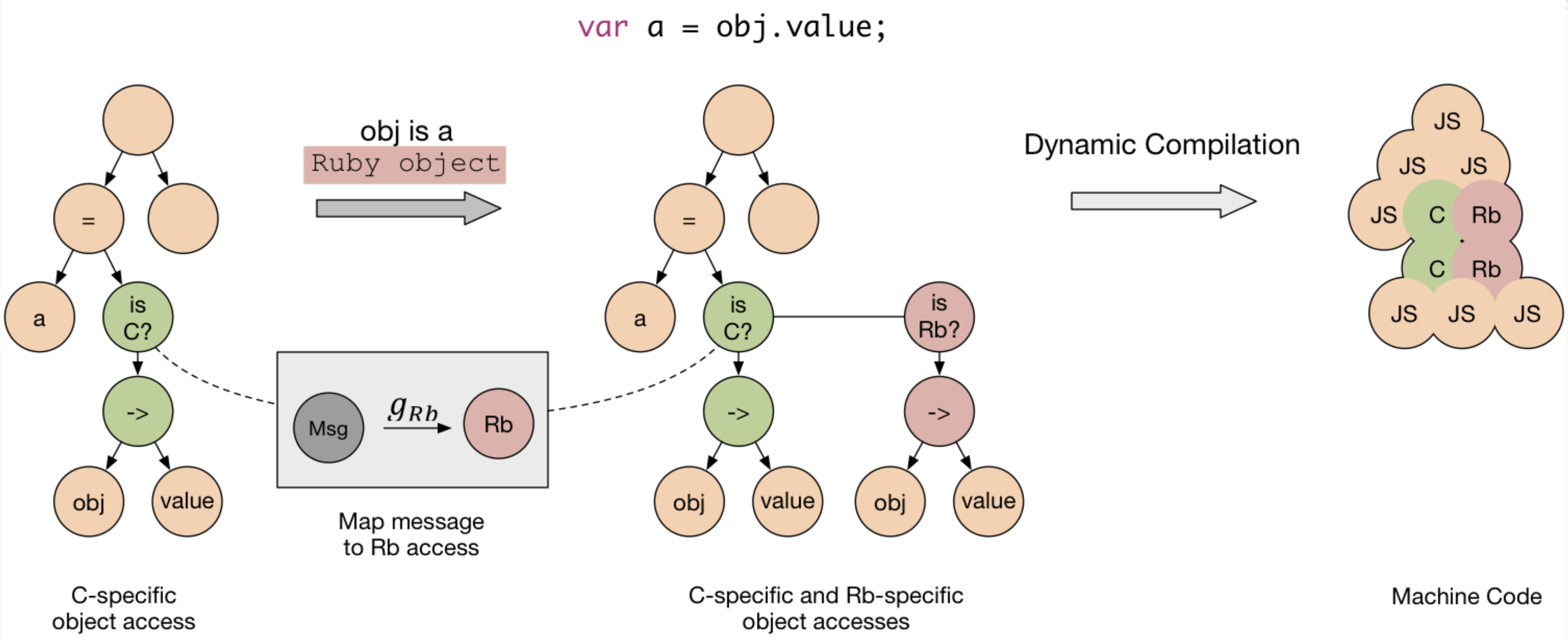
High-Performance Language Interoperability

```
var a = obj.value;
```



JS-specific object access

High-Performance Language Interoperability



Polyglot performance

- Uses the same AST rewriting framework
- Optimizes through the language barrier
- The foreign language nodes make no difference to Truffle
- It can optimize as well as if it were nodes from the host language
  - Same inline cache principle as for regular method calls

Polyglot

More examples

Using Numpy from Java – Calling into Numpy

```
try (Context context = Context.newBuilder()
      .option("python.PythonPath", "/path/to/numpy-1.16.4-py3.7-macosx-10.14-x86_64.egg")
      .allowAllAccess(true).build()) {
  Value geomean = context.eval("python", "import numpy\n" + "import math\n" +
      "lambda x: math.pow(numpy.array(x).prod(), 1/len(x))");
```

anonymous function that calculates the geometric mean using numpy and the math module

```
  double[] values = new double[] { 1, 5, 8, 3, 5, 8, 8, 7, 5, 6 };
  double mean = geomean.execute(values).asDouble();
  System.out.println(mean);
  // 4.905181164183902

}
```

Using CUDA to Access Nvidia GPUs

- Different binding libraries / APIs for CUDA in different programming languages
- Varying set of supported features
- Translation to/from unmanaged environment (in Java, C#, Python, etc.)

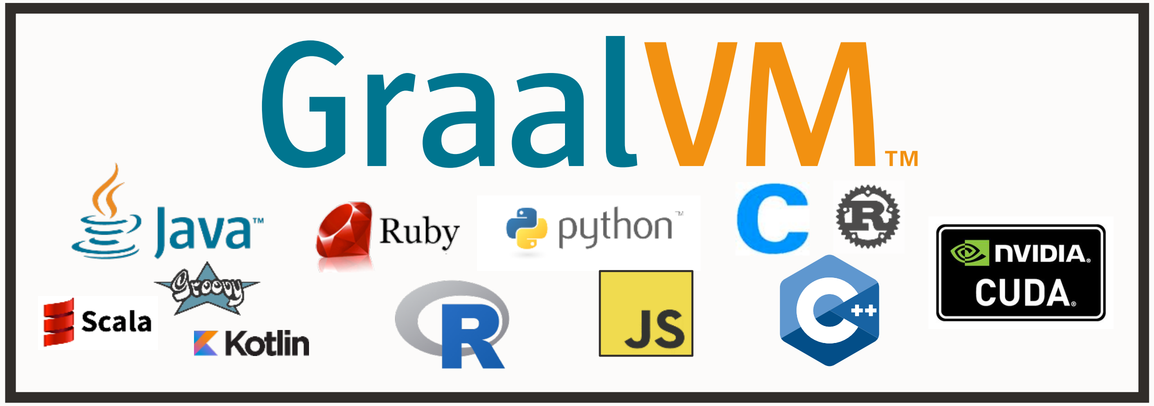
Python	Numba, cuPy, PyCUDA
Java	JCuda, jCUDA, CUDA4J
C / C++	CUDA C/C++ (language extension)
R	gpuR, indirectly through Rcpp
JS	gpu.js (WebGL), node-cuda, cuda-ts
C#	Hybridizer, ManagedCUDA, Alea GPU, ILGPU
Ruby	RbCUDA

Using grCUDA to Access Nvidia GPUs

- Efficient exchange of data between host language and GPU without burdening the programmer
- Expose GPU resources in ways that are native in the host language, e.g., as arrays
- Allow programmers to invoke existing GPU code from their host language
- Allow programmers to define new GPU kernels on the fly
- Polyglot interface: uniform bindings across all programming languages

- Implemented as a “Truffle Language”
  (although “CUDA” is a platform, not a language)

- Developed by NVIDIA in collaboration
  with Oracle Labs
- BSD 3-clause license



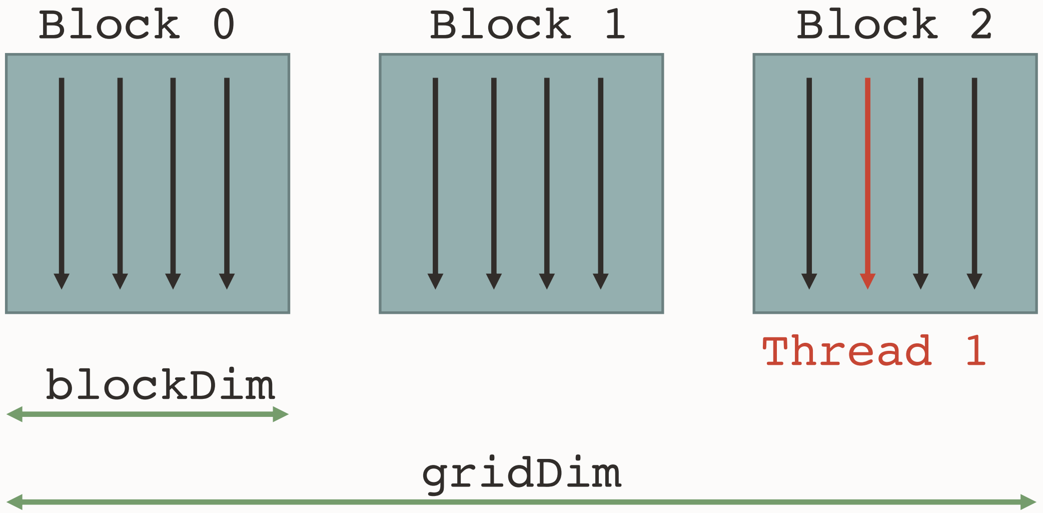
Creating and Using Device Arrays (Python)

```
import polyglot

# Get constructor function as callable
DeviceArray = polyglot.eval(language=‘grcuda’, string=‘DeviceArray’)

# Create 1D device array that can hold 1000 int values
dev_int_arr = DeviceArray(‘int’, 1000)

# Create 2D device array that can hold 1000 x 100 float values
dev_float_2d = DeviceArray(‘float’, 1000, 100)

# Setting array elements
for i in range(len(dev_int_array)):
    dev_int_arr[i] = i
for i in range(len(dev_float_2d)):
    for j in range(len(dev_float_2d[0])):
        dev_float_2d[i][j] = i + j
```

As fast or faster than native language constructs (e.g., R vectors) because of simpler semantics (e.g., no NA values)

GPU Kernels in CUDA C++

```
__global__ void inc_kernel(float *out_arr, const float *in_arr, size_t num_elements) {
  for (auto idx = blockIdx.x * blockDim.x + threadIdx.x; idx < num_elements;
       idx += gridDim.x * blockDim.x) {
    out_arr[idx] = in_arr[idx] + 1;
  }
}
```



Block 0

Block 1

Block 2

Thread 1

blockDim

gridDim

Number of blocks and threads can be configured in 3 dimensions (x, y, z)

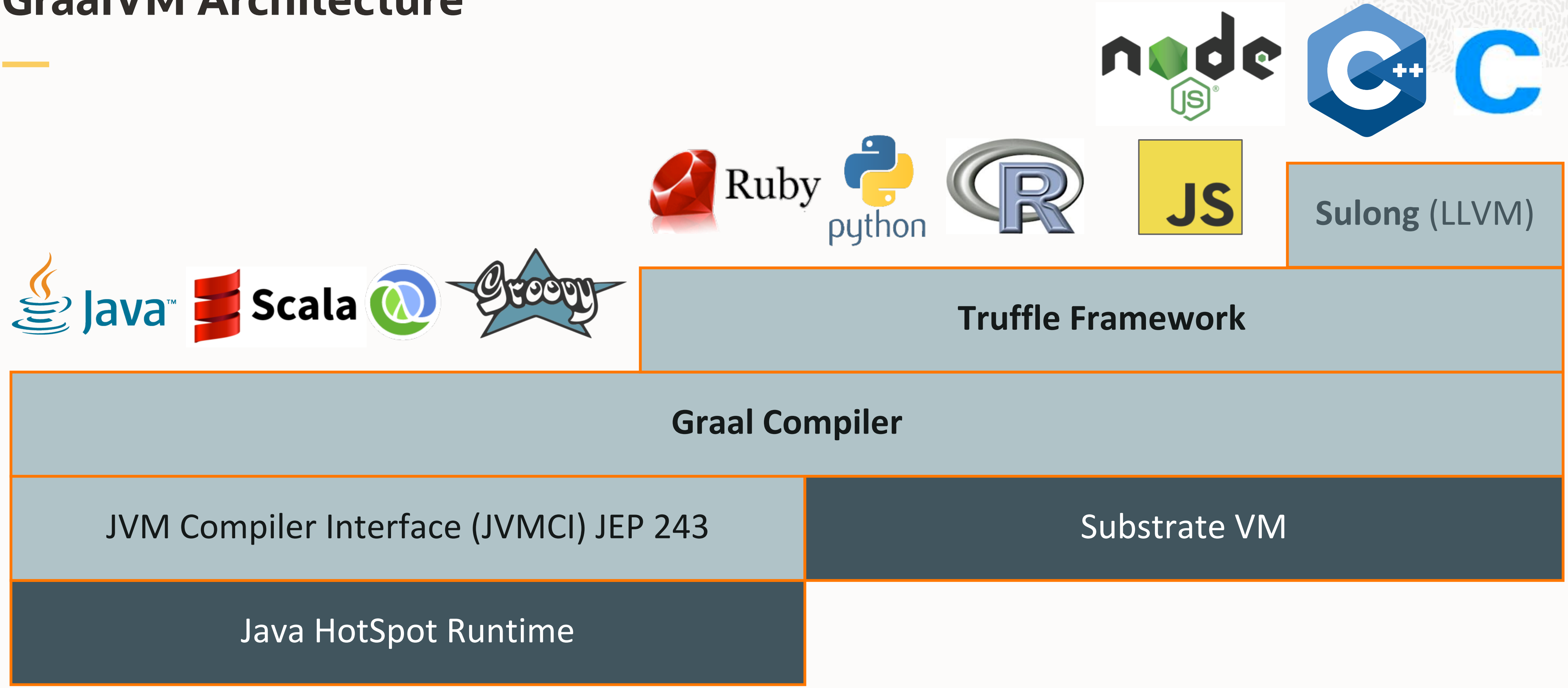
Launching GPU Kernels (JS)

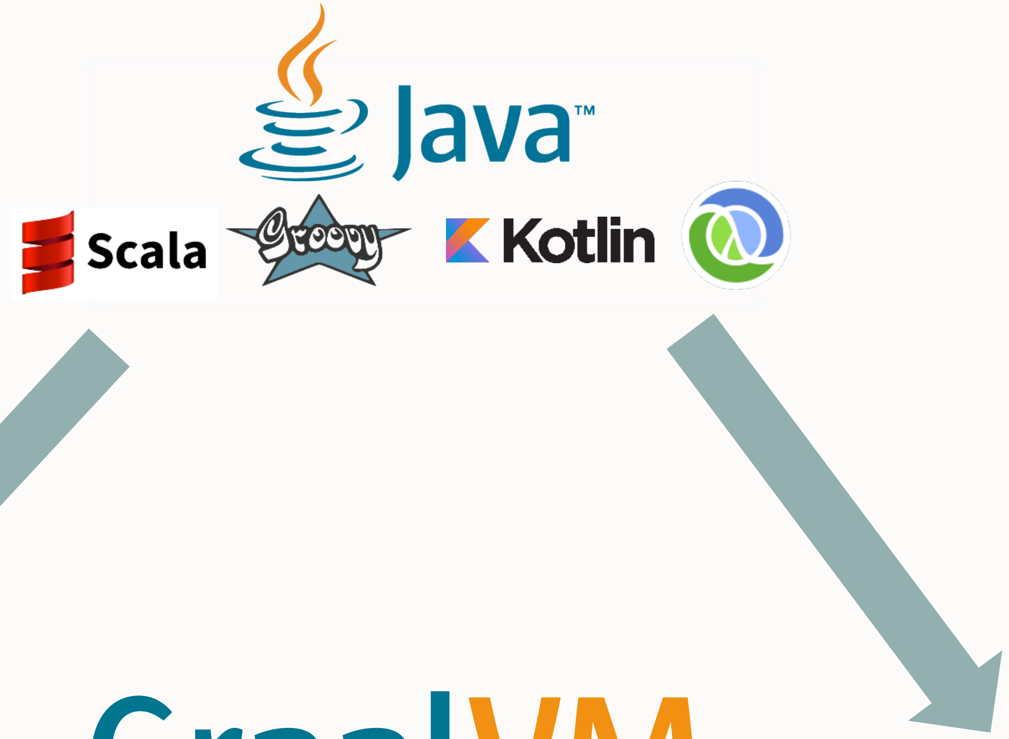
```
const DeviceArray = Polyglot.eval('grcuda', string='DeviceArray')
const N = 1000
const in_arr = DeviceArray('float', N)
const out_arr = DeviceArray('float', N)
for (let i = 0; i < N; i++)
  in_arr[i] = i
const code = '__global__ void inc_kernel(...) ...'
const buildkernel = Polyglot.eval('grcuda', string='buildkernel')
const incKernel = buildkernel(code, 'inc_kernel', 'pointer, pointer, uint64')

// Launch kernel in grid consisting of 160 blocks with 256 threads each
incKernel(160, 256)(out_arr, in_arr, N)

for (let i = 0; i < 10; i++) {
  console.log(out_arr[i]);
}
```

Device arrays `in_arr` and `out_arr` can be passed to GPU kernel

Embedding and native image generation

GraalVM Architecture





GraalVM™

JIT

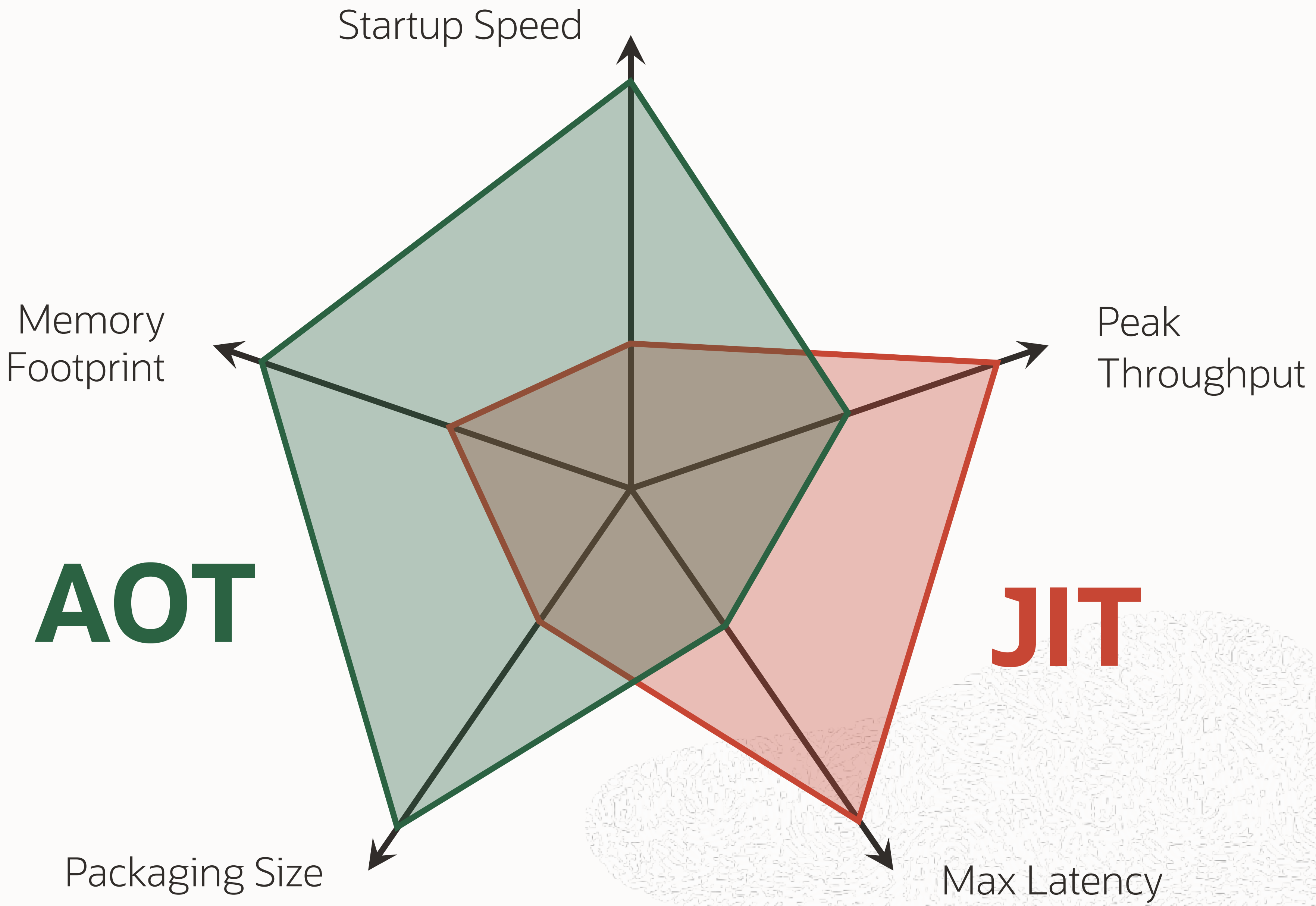
```
$ java MyMainClass
```

OpenJDK™

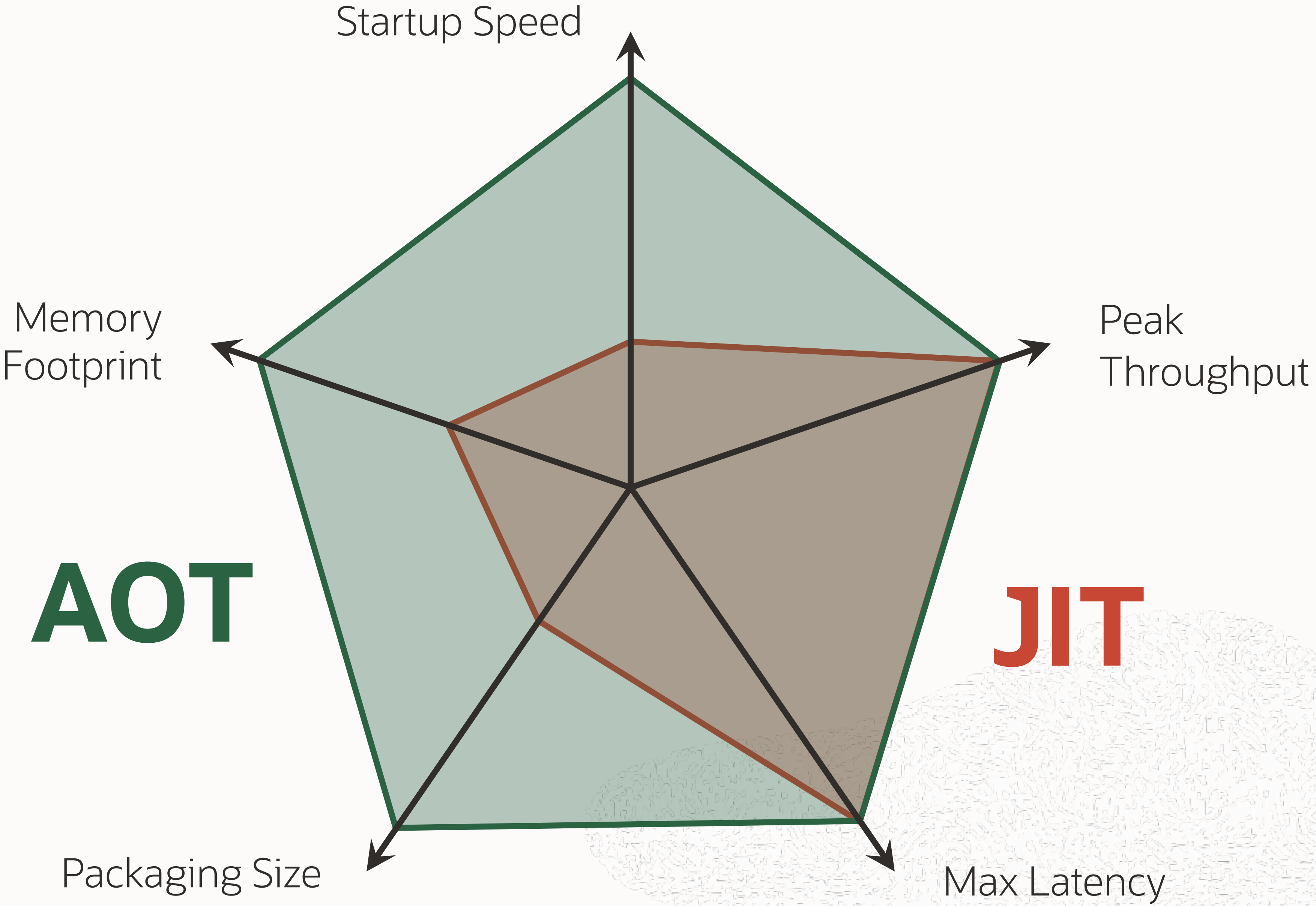
AOT

```
$ native-image MyMainClass
$ ./mymainclass
```

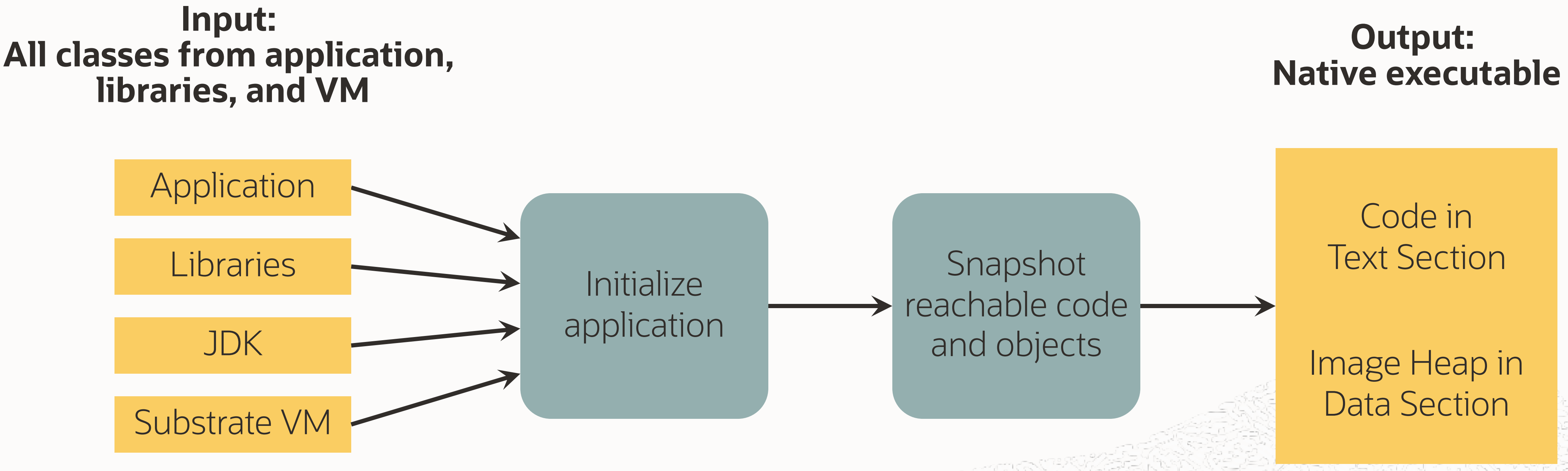
Currently


Startup Speed
Memory Footprint
Peak Throughput
AOT
JIT
Packaging Size
Max Latency

Goal


Startup Speed
Memory Footprint
Peak Throughput
AOT
JIT
Packaging Size
Max Latency

Native Image: Principle



Input:
All classes from application, libraries, and VM

Application

Libraries

JDK

Substrate VM

Initialize application

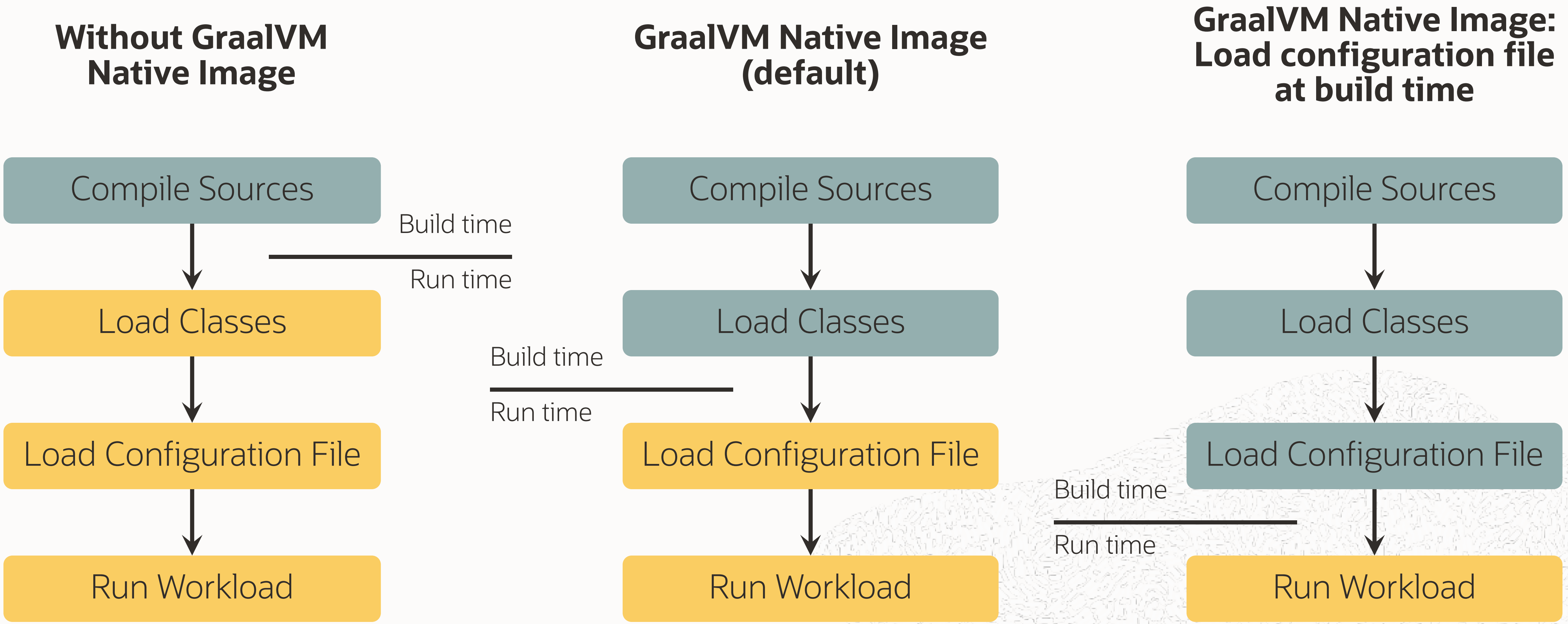
Snapshot reachable code and objects

Output:
Native executable

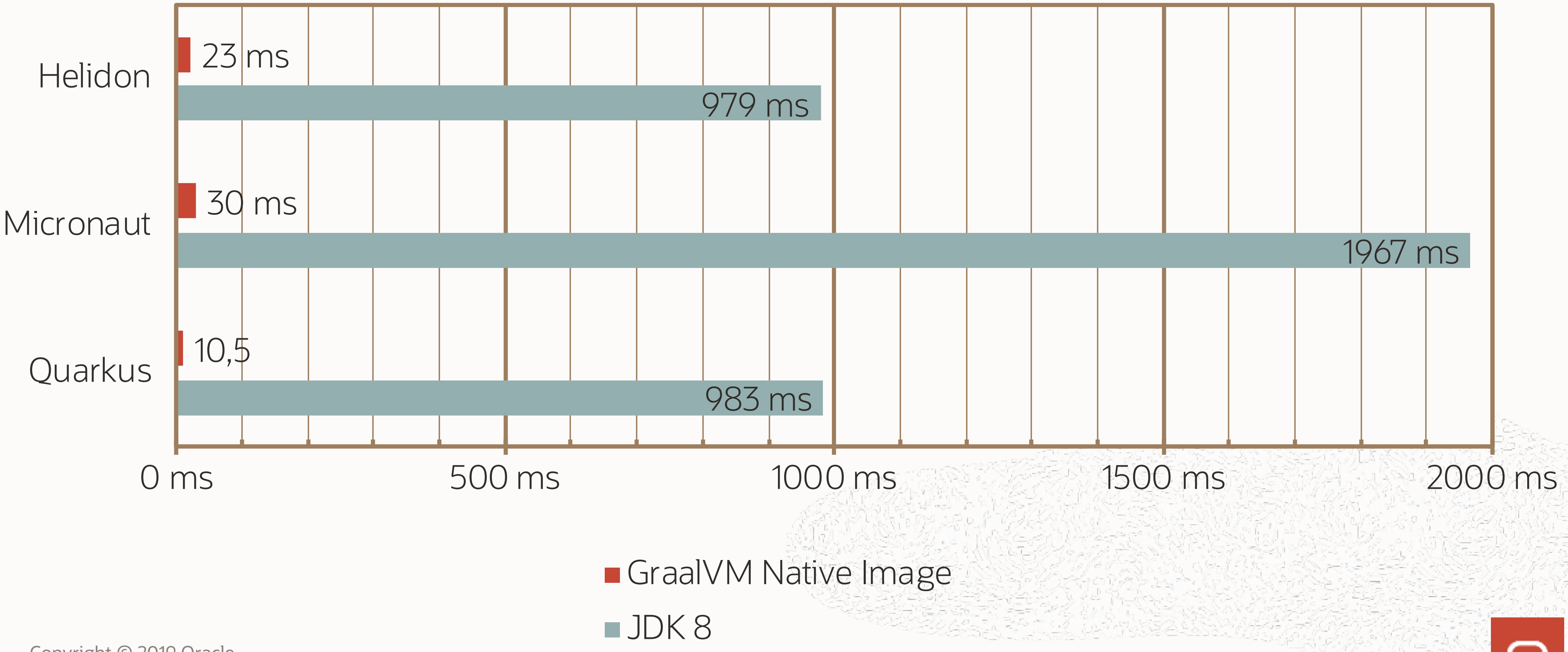
Code in Text Section

Image Heap in Data Section

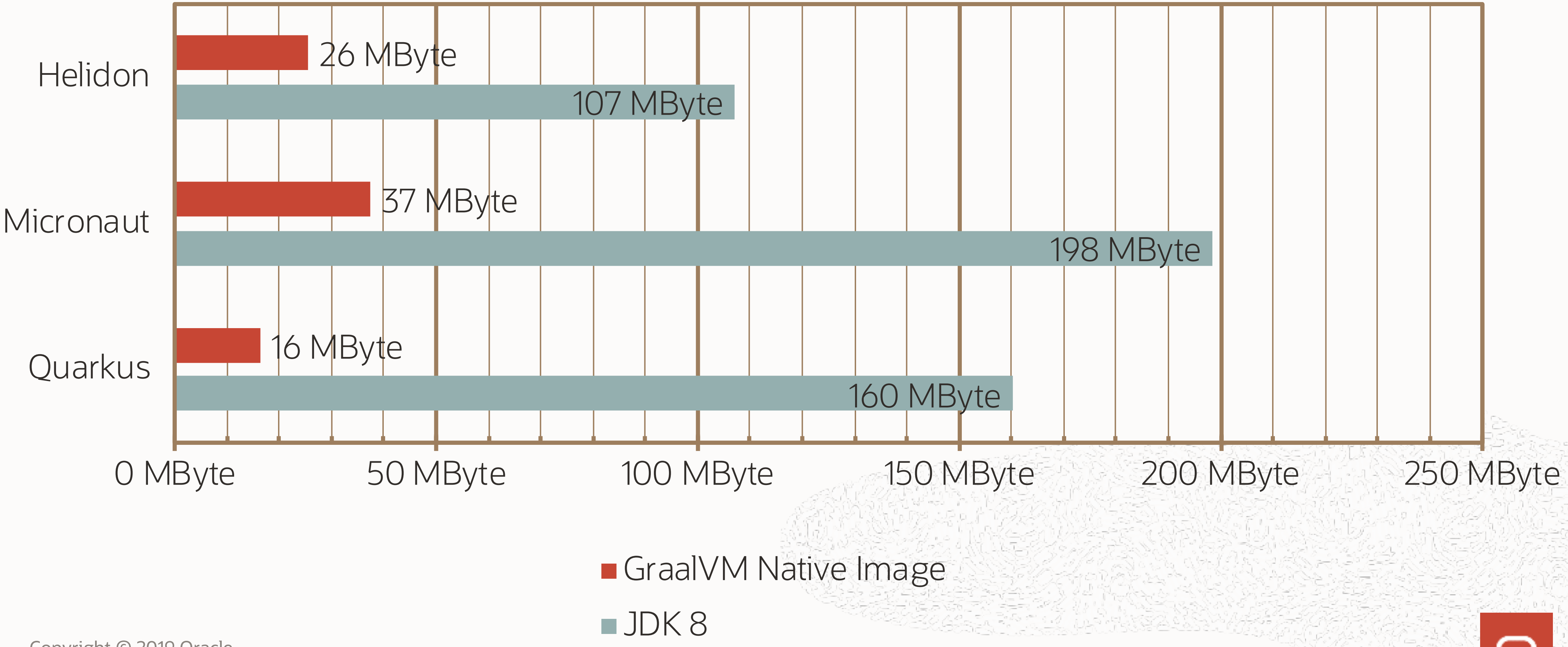
Benefits of the Image Heap



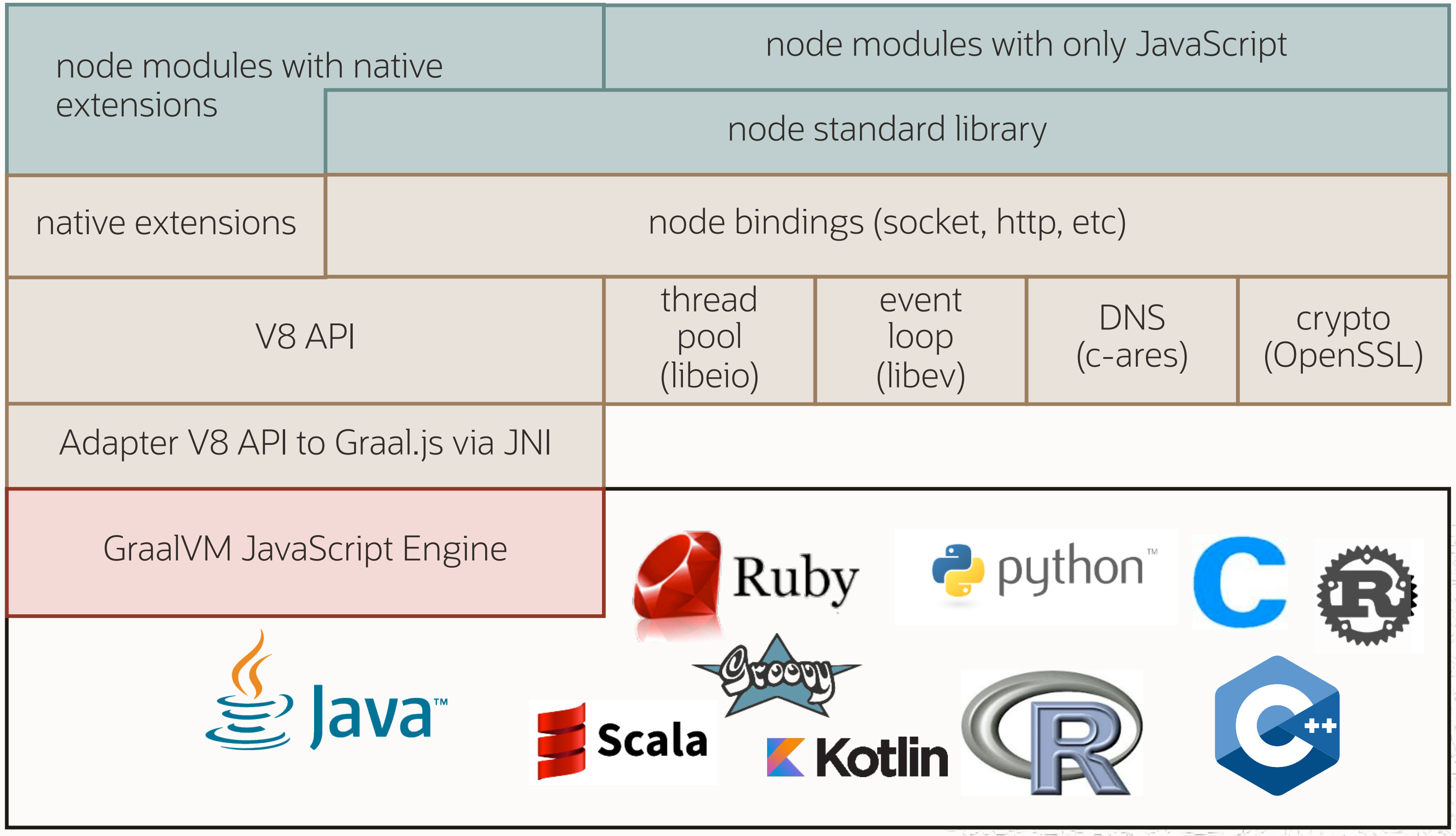
Startup Time of Java Microservice Frameworks



Memory Footprint



Architecture of Node.js running via GraalVM

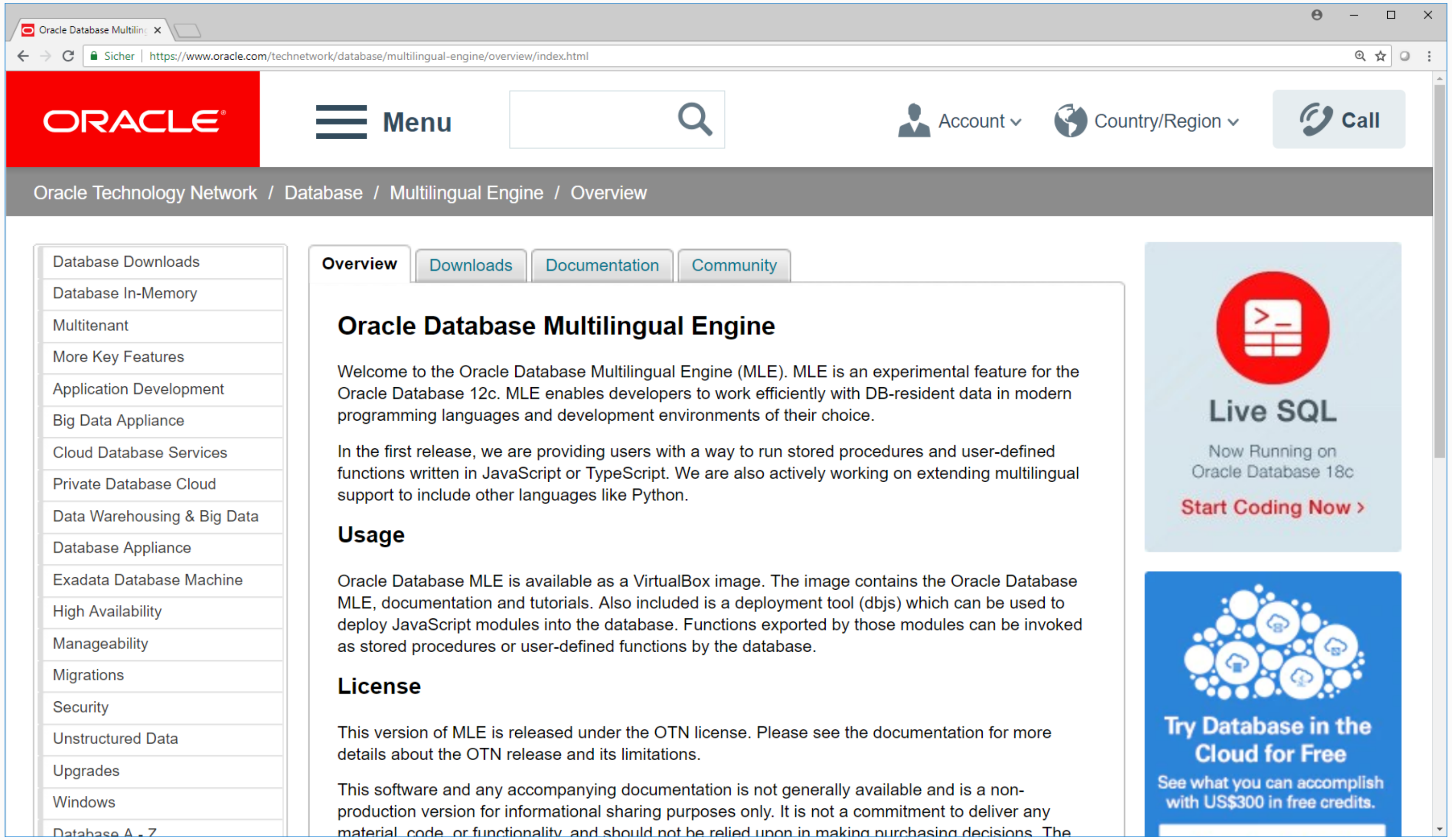

node modules with native extensions
node modules with only JavaScript
node standard library
native extensions
node bindings (socket, http, etc)
V8 API
thread pool (libeio)
event loop (libev)
DNS (c-ares)
crypto (OpenSSL)
Adapter V8 API to Graal.js via JNI
GraalVM JavaScript Engine
Ruby
python™
Java™
Scala
Groovy
Kotlin

🟦 JavaScript

🟫 C++

🟥 Java

Embedding Graal: MLE (OracleDB)



Oracle Database Multilingual Engine

Welcome to the Oracle Database Multilingual Engine (MLE). MLE is an experimental feature for the Oracle Database 12c. MLE enables developers to work efficiently with DB-resident data in modern programming languages and development environments of their choice.

In the first release, we are providing users with a way to run stored procedures and user-defined functions written in JavaScript or TypeScript. We are also actively working on extending multilingual support to include other languages like Python.

Usage

Oracle Database MLE is available as a VirtualBox image. The image contains the Oracle Database MLE, documentation and tutorials. Also included is a deployment tool (dbjs) which can be used to deploy JavaScript modules into the database. Functions exported by those modules can be invoked as stored procedures or user-defined functions by the database.

License

This version of MLE is released under the OTN license. Please see the documentation for more details about the OTN release and its limitations.

This software and any accompanying documentation is not generally available and is a non-production version for informational sharing purposes only. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The

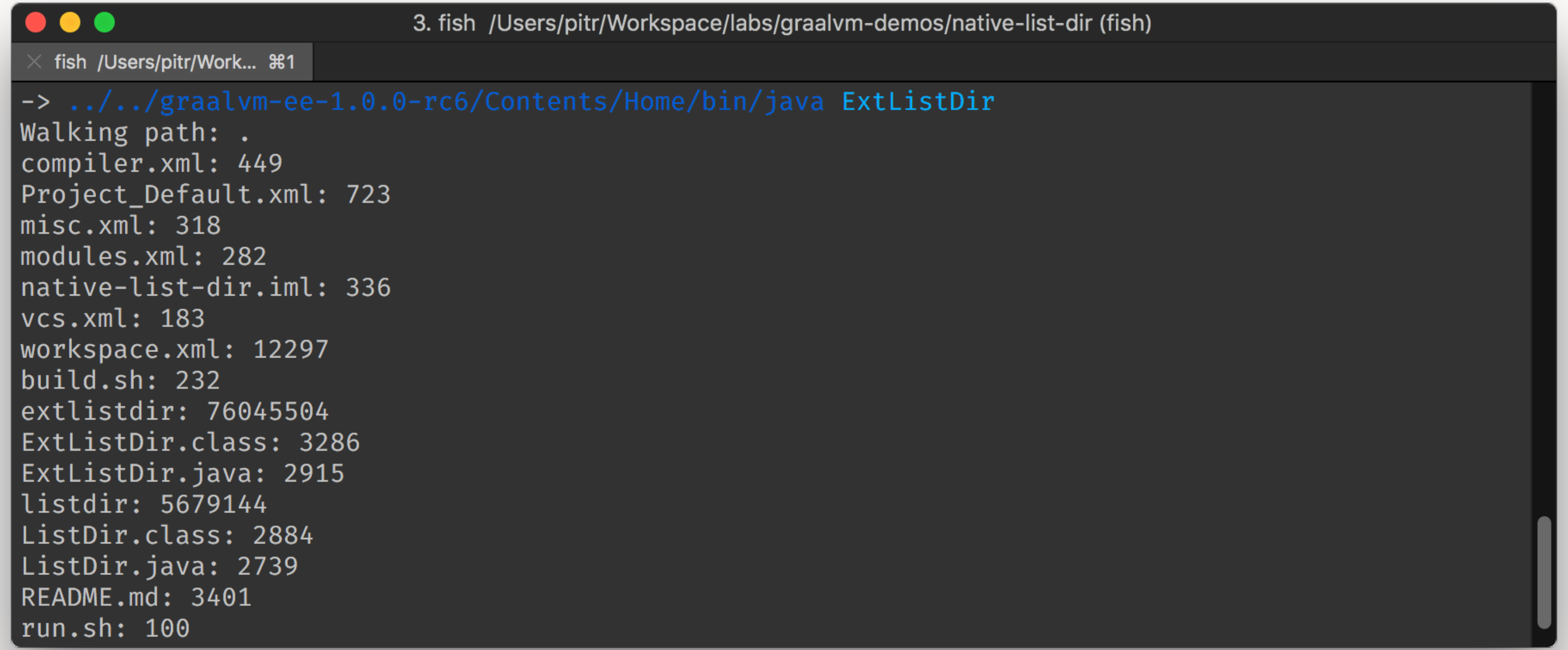
```
module.exports.helloworld =
  function () {
    return "Hello World";
  }

export function helloworld():string;

shell> dbjs deploy helloworld.js

sql> SELECT helloworld() from dual;
```

https://www.oracle.com/technetwork/database/multilingual-engine/

Embedding small JS snippet

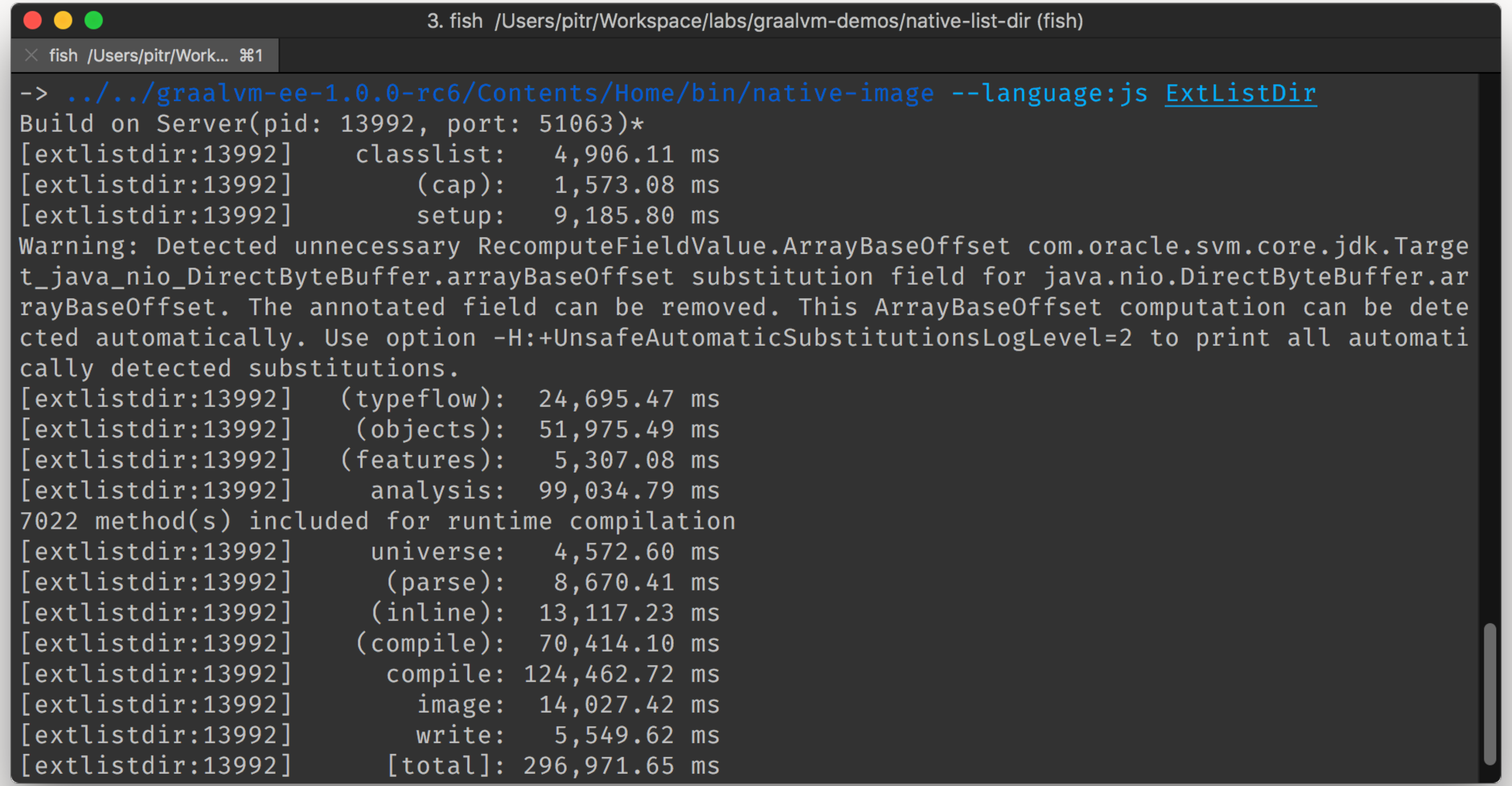
```java
import org.graalvm.polyglot.*;
// ...

public static void main(String[] args) throws java.io.IOException {
    System.out.println("Walking path: " + Paths.get(args.length > 0 ? args[0] : "."));
    final Context context = Context.create("js");
    final Value jsFn = context.eval("js",
            "function(name, size) { return name + ': ' + size}");
    try (Stream<Path> paths = Files.walk(Paths.get("."))) {
        paths.filter(Files::isRegularFile).
                forEach((Path p) -> {
                    File f = p.toFile();
                    Value v = jsFn.execute(f.getName(), f.length());
                    System.out.println(v);
                });
    }
}
```

Embedding small JS snippet


3. fish /Users/pitr/Workspace/labs/graalvm-demos/native-list-dir (fish)
fish /Users/pitr/Work... ⌘1

```
-> ../../graalvm-ee-1.0.0-rc6/Contents/Home/bin/java ExtListDir
Walking path: .
compiler.xml: 449
Project_Default.xml: 723
misc.xml: 318
modules.xml: 282
native-list-dir.iml: 336
vcs.xml: 183
workspace.xml: 12297
build.sh: 232
extlistdir: 76045504
ExtListDir.class: 3286
ExtListDir.java: 2915
listdir: 5679144
ListDir.class: 2884
ListDir.java: 2739
README.md: 3401
run.sh: 100
```

Embedding small JS snippet

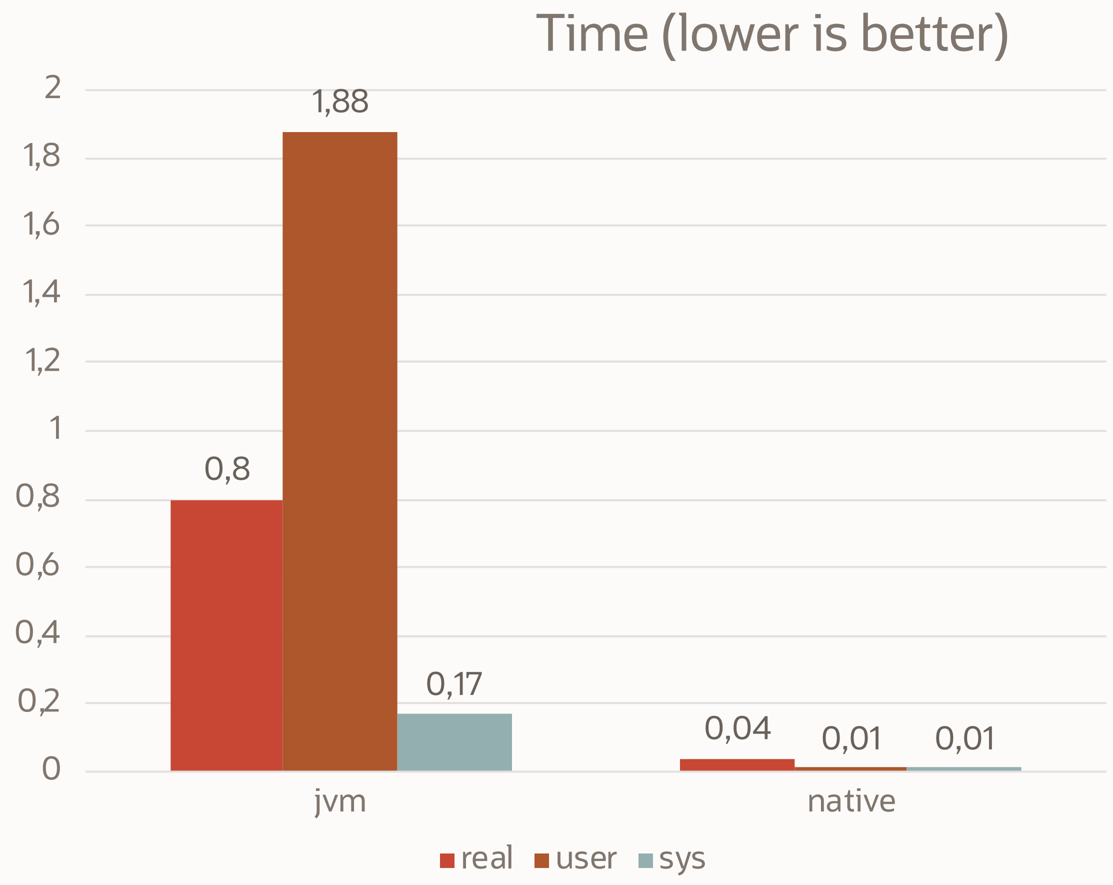


3. fish /Users/pitr/Workspace/labs/graalvm-demos/native-list-dir (fish)
fish /Users/pitr/Work... ⌘1

```
-> ../../graalvm-ee-1.0.0-rc6/Contents/Home/bin/native-image --language:js ExtListDir
Build on Server(pid: 13992, port: 51063)*
[extlistdir:13992]    classlist:   4,906.11 ms
[extlistdir:13992]        (cap):   1,573.08 ms
[extlistdir:13992]        setup:   9,185.80 ms
Warning: Detected unnecessary RecomputeFieldValue.ArrayBaseOffset com.oracle.svm.core.jdk.Targe
t_java_nio_DirectByteBuffer.arrayBaseOffset substitution field for java.nio.DirectByteBuffer.ar
rayBaseOffset. The annotated field can be removed. This ArrayBaseOffset computation can be dete
cted automatically. Use option -H:+UnsafeAutomaticSubstitutionsLogLevel=2 to print all automati
cally detected substitutions.
[extlistdir:13992]   (typeflow):  24,695.47 ms
[extlistdir:13992]    (objects):  51,975.49 ms
[extlistdir:13992]   (features):   5,307.08 ms
[extlistdir:13992]     analysis:  99,034.79 ms
7022 method(s) included for runtime compilation
[extlistdir:13992]     universe:   4,572.60 ms
[extlistdir:13992]      (parse):   8,670.41 ms
[extlistdir:13992]     (inline):  13,117.23 ms
[extlistdir:13992]    (compile):  70,414.10 ms
[extlistdir:13992]      compile: 124,462.72 ms
[extlistdir:13992]        image:  14,027.42 ms
[extlistdir:13992]        write:   5,549.62 ms
[extlistdir:13992]      [total]: 296,971.65 ms
```

Embedding small JS snippet

- `time graalvm-ee-1.0.0-rc6/.../bin/java ExtListDir`
- `time ./extlistdir`



Embedding small JS snippet

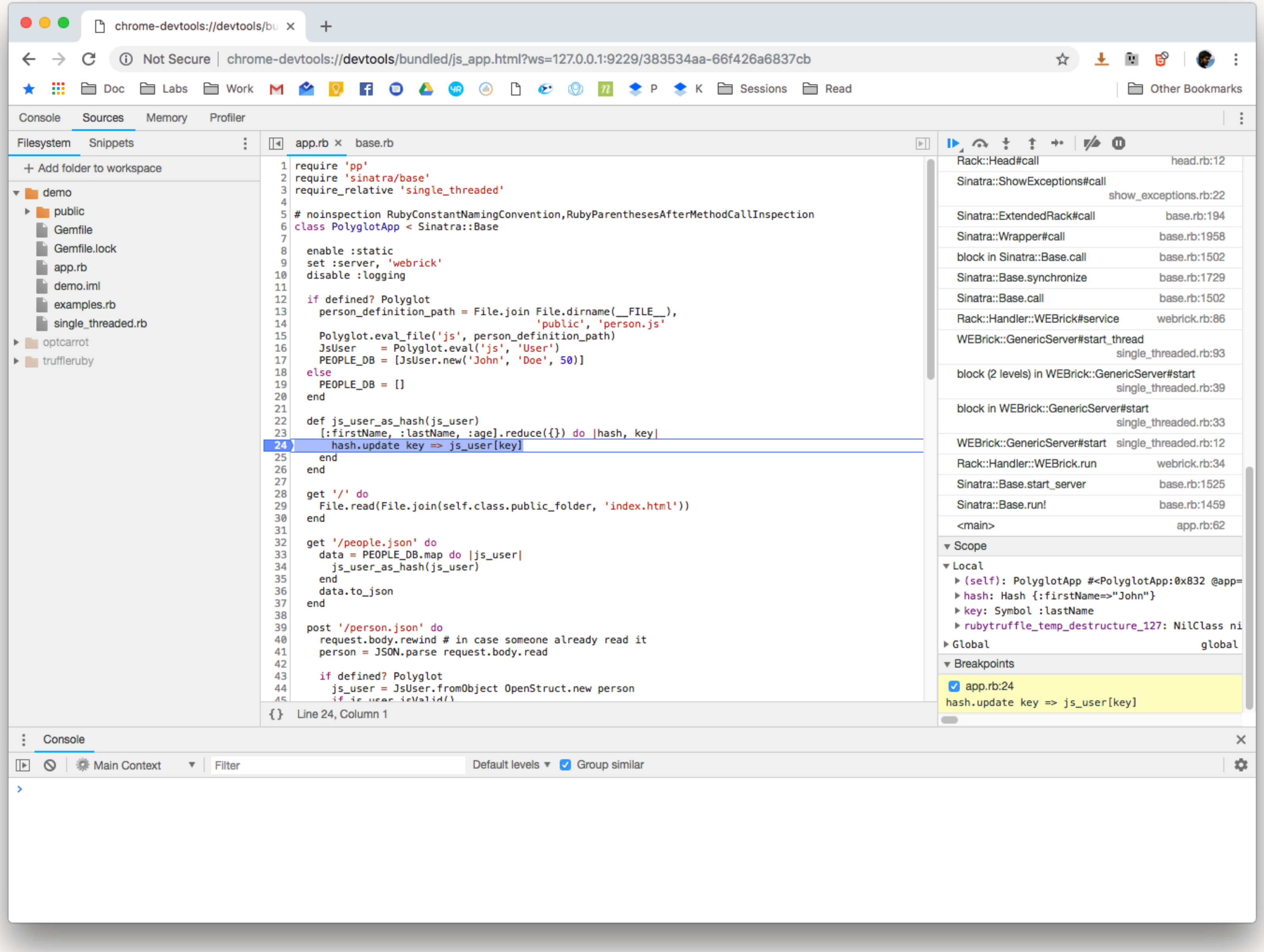
- `extlistdir` is just 72MB
- It is a native image without dependencies on JVM
- It has the JS engine in it
- It executed almost as fast as native `ls` command

Instrumentation and Tooling

Debugging

Debugging with Chrome DevTools

- Run the server with *--inspect* option
    - `ruby --polyglot --experimental-options --ruby.single-threaded --inspect app.rb`
- Follow the instructions and open the *chrome-devtools:* link
    - Step through the languages



```ruby
require 'pp'
require 'sinatra/base'
require_relative 'single_threaded'

# noinspection RubyConstantNamingConvention,RubyParenthesesAfterMethodCallInspection
class PolyglotApp < Sinatra::Base

  enable :static
  set :server, 'webrick'
  disable :logging

  if defined? Polyglot
    person_definition_path = File.join File.dirname(__FILE__),
                                       'public', 'person.js'
    Polyglot.eval_file('js', person_definition_path)
    JsUser    = Polyglot.eval('js', 'User')
    PEOPLE_DB = [JsUser.new('John', 'Doe', 50)]
  else
    PEOPLE_DB = []
  end

  def js_user_as_hash(js_user)
    [:firstName, :lastName, :age].reduce({}) do |hash, key|
      hash.update key => js_user[key]
    end
  end

  get '/' do
    File.read(File.join(self.class.public_folder, 'index.html'))
  end

  get '/people.json' do
    data = PEOPLE_DB.map do |js_user|
      js_user_as_hash(js_user)
    end
    data.to_json
  end

  post '/person.json' do
    request.body.rewind # in case someone already read it
    person = JSON.parse request.body.read

    if defined? Polyglot
      js_user = JsUser.fromObject OpenStruct.new person
      if js_user.isValid()
```

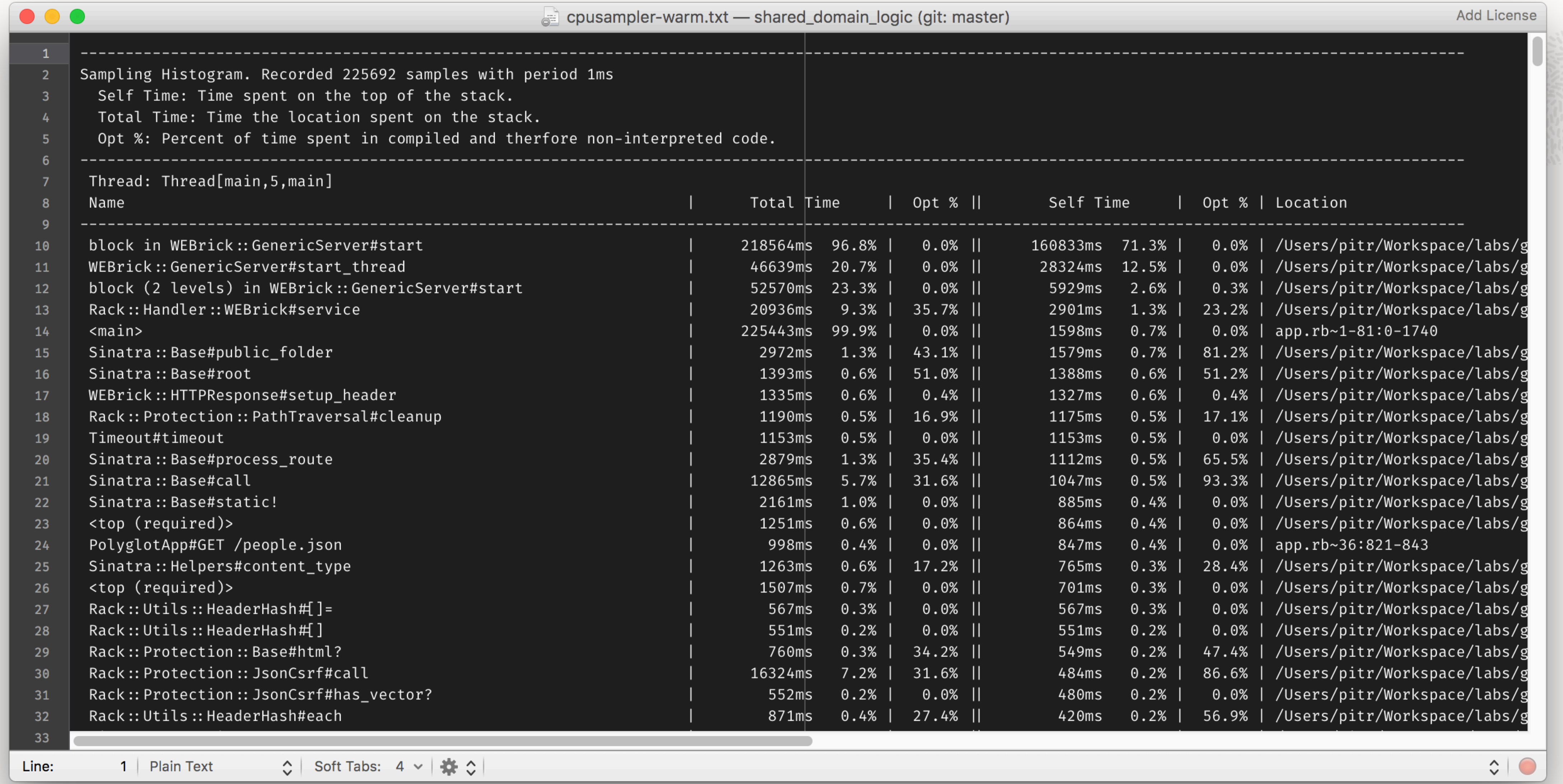
```
Rack::Head#call                                    head.rb:12
Sinatra::ShowExceptions#call
                                     show_exceptions.rb:22
Sinatra::ExtendedRack#call                        base.rb:194
Sinatra::Wrapper#call                            base.rb:1958
block in Sinatra::Base.call                      base.rb:1502
Sinatra::Base.synchronize                        base.rb:1729
Sinatra::Base.call                               base.rb:1502
Rack::Handler::WEBrick#service                  webrick.rb:86
WEBrick::GenericServer#start_thread
                                      single_threaded.rb:93
block (2 levels) in WEBrick::GenericServer#start
                                      single_threaded.rb:39
block in WEBrick::GenericServer#start
                                      single_threaded.rb:33
WEBrick::GenericServer#start          single_threaded.rb:12
Rack::Handler::WEBrick.run                      webrick.rb:34
Sinatra::Base.start_server                       base.rb:1525
Sinatra::Base.run!                               base.rb:1459
<main>                                            app.rb:62
▼ Scope
▼ Local
  ▶ (self): PolyglotApp #<PolyglotApp:0x832 @app=
  ▶ hash: Hash {:firstName=>"John"}
  ▶ key: Symbol :lastName
  ▶ rubytruffle_temp_destructure_127: NilClass ni
▶ Global                                          global
▼ Breakpoints
☑ app.rb:24
hash.update key => js_user[key]
```

Instrumentation and Tooling

Profiling

CPU-Sampler

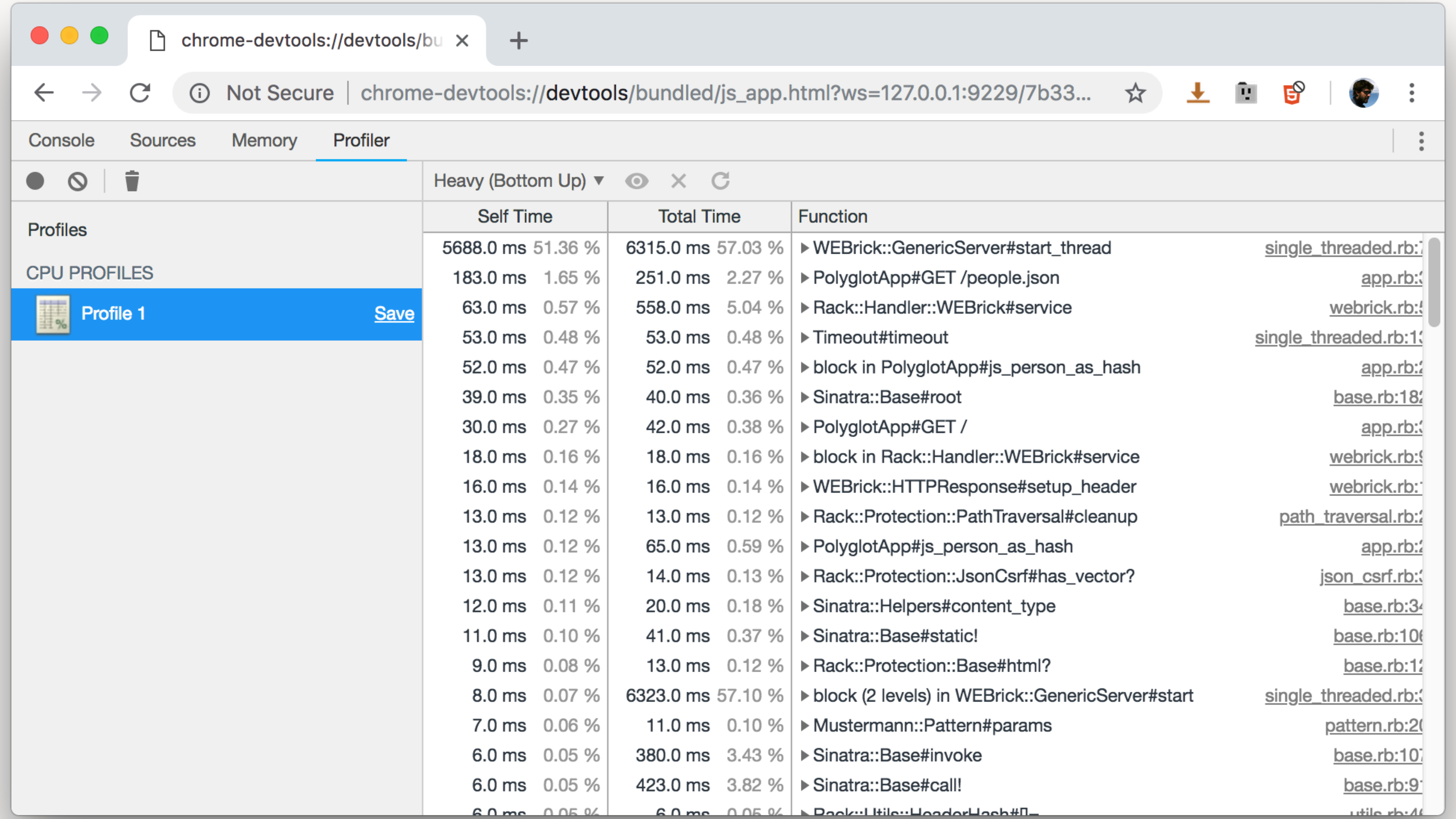
- Let’s introduce extra slowness affecting `/people.js` request
- CPU-Sampler can tell us which methods take the most time
  - `ruby --jvm --polyglot -Xsingle_threaded --cpusampler app.rb`



```
-------------------------------------------------------------------------------------------------------------------------------------------------------
Sampling Histogram. Recorded 225692 samples with period 1ms
  Self Time: Time spent on the top of the stack.
  Total Time: Time the location spent on the stack.
  Opt %: Percent of time spent in compiled and therfore non-interpreted code.
-------------------------------------------------------------------------------------------------------------------------------------------------------
 Thread: Thread[main,5,main]
 Name                                                          |      Total Time     |  Opt % ||       Self Time     |  Opt % | Location
-------------------------------------------------------------------------------------------------------------------------------------------------------
 block in WEBrick::GenericServer#start                         |     218564ms  96.8% |   0.0% ||     160833ms  71.3% |   0.0% | /Users/pitr/Workspace/labs/g
 WEBrick::GenericServer#start_thread                           |      46639ms  20.7% |   0.0% ||      28324ms  12.5% |   0.0% | /Users/pitr/Workspace/labs/g
 block (2 levels) in WEBrick::GenericServer#start              |      52570ms  23.3% |   0.0% ||       5929ms   2.6% |   0.3% | /Users/pitr/Workspace/labs/g
 Rack::Handler::WEBrick#service                                |      20936ms   9.3% |  35.7% ||       2901ms   1.3% |  23.2% | /Users/pitr/Workspace/labs/g
 <main>                                                        |     225443ms  99.9% |   0.0% ||       1598ms   0.7% |   0.0% | app.rb~1-81:0-1740
 Sinatra::Base#public_folder                                   |       2972ms   1.3% |  43.1% ||       1579ms   0.7% |  81.2% | /Users/pitr/Workspace/labs/g
 Sinatra::Base#root                                            |       1393ms   0.6% |  51.0% ||       1388ms   0.6% |  51.2% | /Users/pitr/Workspace/labs/g
 WEBrick::HTTPResponse#setup_header                            |       1335ms   0.6% |   0.4% ||       1327ms   0.6% |   0.4% | /Users/pitr/Workspace/labs/g
 Rack::Protection::PathTraversal#cleanup                       |       1190ms   0.5% |  16.9% ||       1175ms   0.5% |  17.1% | /Users/pitr/Workspace/labs/g
 Timeout#timeout                                               |       1153ms   0.5% |   0.0% ||       1153ms   0.5% |   0.0% | /Users/pitr/Workspace/labs/g
 Sinatra::Base#process_route                                   |       2879ms   1.3% |  35.4% ||       1112ms   0.5% |  65.5% | /Users/pitr/Workspace/labs/g
 Sinatra::Base#call                                            |      12865ms   5.7% |  31.6% ||       1047ms   0.5% |  93.3% | /Users/pitr/Workspace/labs/g
 Sinatra::Base#static!                                         |       2161ms   1.0% |   0.0% ||        885ms   0.4% |   0.0% | /Users/pitr/Workspace/labs/g
 <top (required)>                                              |       1251ms   0.6% |   0.0% ||        864ms   0.4% |   0.0% | /Users/pitr/Workspace/labs/g
 PolyglotApp#GET /people.json                                  |        998ms   0.4% |   0.0% ||        847ms   0.4% |   0.0% | app.rb~36:821-843
 Sinatra::Helpers#content_type                                 |       1263ms   0.6% |  17.2% ||        765ms   0.3% |  28.4% | /Users/pitr/Workspace/labs/g
 <top (required)>                                              |       1507ms   0.7% |   0.0% ||        701ms   0.3% |   0.0% | /Users/pitr/Workspace/labs/g
 Rack::Utils::HeaderHash#[]=                                   |        567ms   0.3% |   0.0% ||        567ms   0.3% |   0.0% | /Users/pitr/Workspace/labs/g
 Rack::Utils::HeaderHash#[]                                    |        551ms   0.2% |   0.0% ||        551ms   0.2% |   0.0% | /Users/pitr/Workspace/labs/g
 Rack::Protection::Base#html?                                  |        760ms   0.3% |  34.2% ||        549ms   0.2% |  47.4% | /Users/pitr/Workspace/labs/g
 Rack::Protection::JsonCsrf#call                               |      16324ms   7.2% |  31.6% ||        484ms   0.2% |  86.6% | /Users/pitr/Workspace/labs/g
 Rack::Protection::JsonCsrf#has_vector?                        |        552ms   0.2% |   0.0% ||        480ms   0.2% |   0.0% | /Users/pitr/Workspace/labs/g
 Rack::Utils::HeaderHash#each                                  |        871ms   0.4% |  27.4% ||        420ms   0.2% |  56.9% | /Users/pitr/Workspace/labs/g
```

Line: 1 Plain Text Soft Tabs: 4

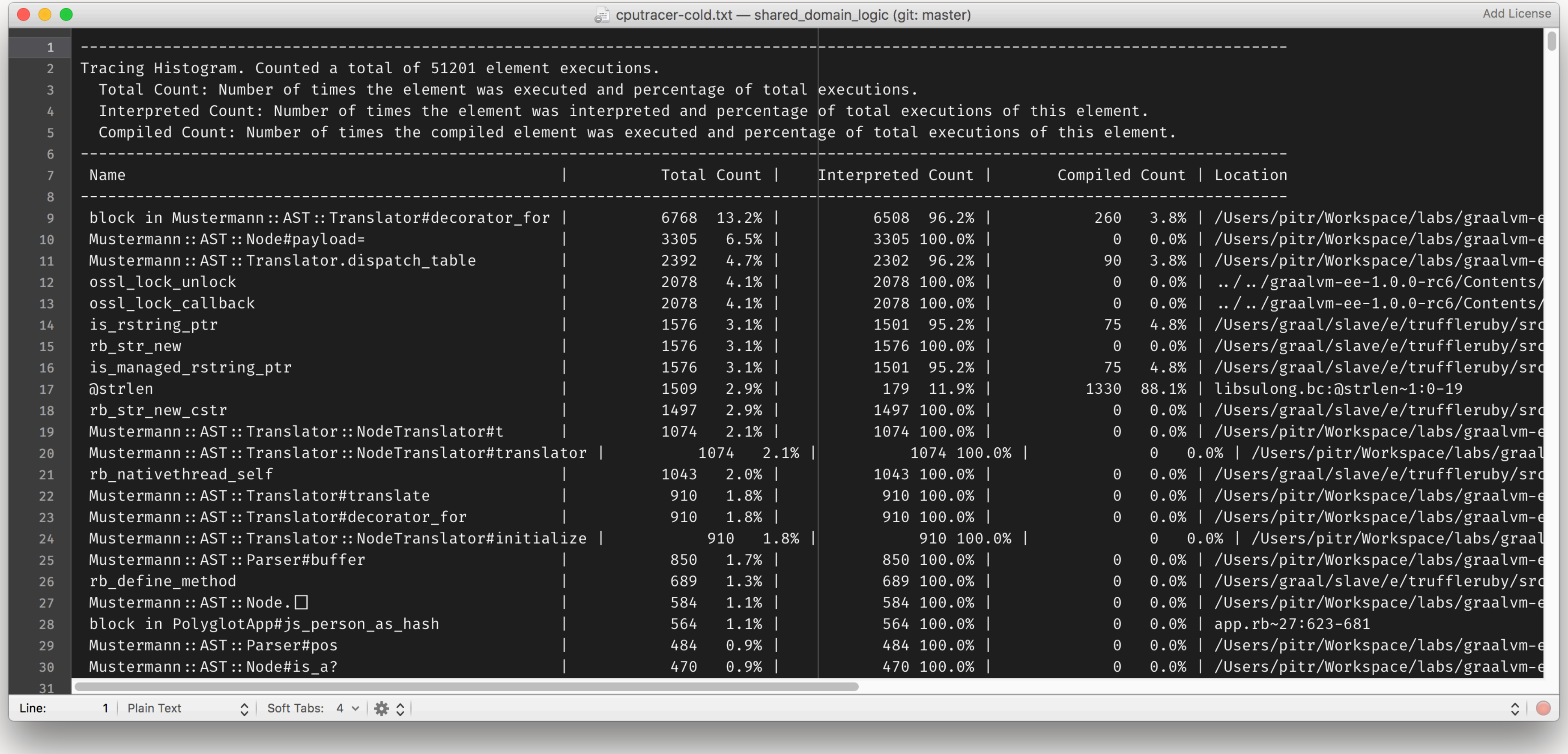
CPU-Sampler in Chrome DevTools



chrome-devtools://devtools/bu
Not Secure
chrome-devtools://devtools/bundled/js_app.html?ws=127.0.0.1:9229/7b33...
Console
Sources
Memory
Profiler
Heavy (Bottom Up) ▾
Profiles
CPU PROFILES
Profile 1
Save

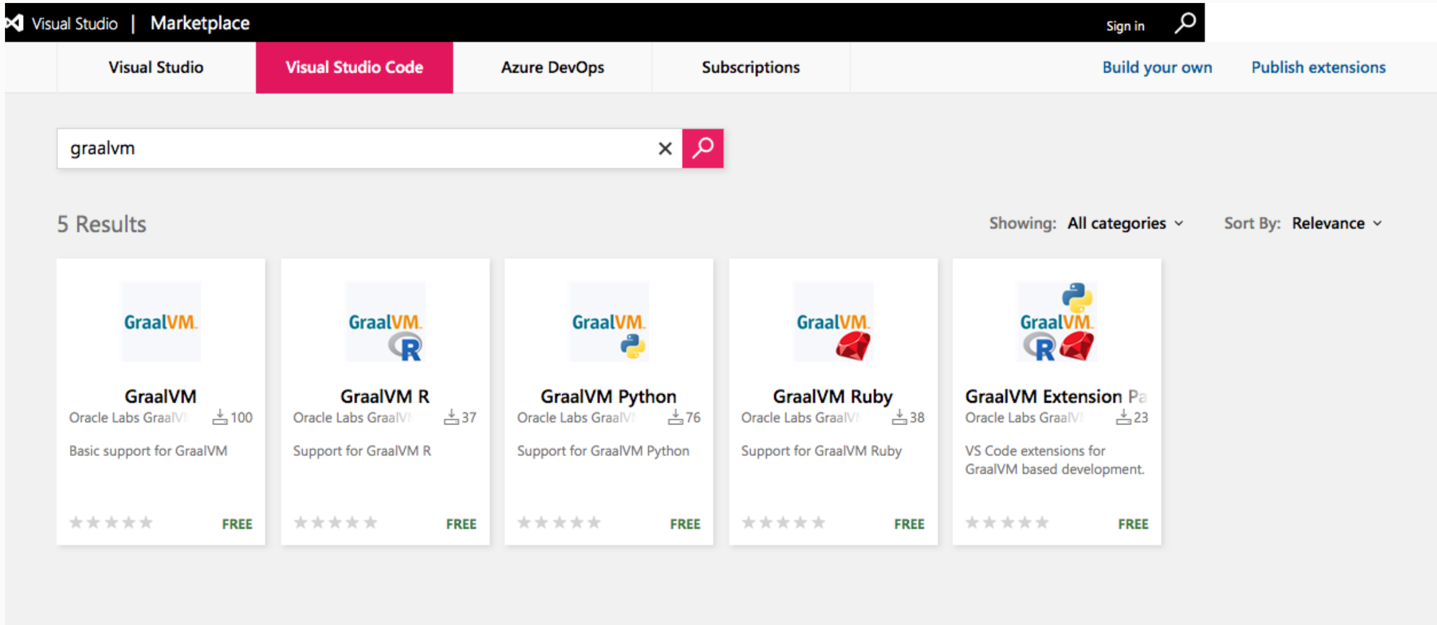
Self Time		Total Time		Function	
5688.0 ms	51.36 %	6315.0 ms	57.03 %	▶ WEBrick::GenericServer#start_thread	single_threaded.rb:7
183.0 ms	1.65 %	251.0 ms	2.27 %	▶ PolyglotApp#GET /people.json	app.rb:3
63.0 ms	0.57 %	558.0 ms	5.04 %	▶ Rack::Handler::WEBrick#service	webrick.rb:5
53.0 ms	0.48 %	53.0 ms	0.48 %	▶ Timeout#timeout	single_threaded.rb:13
52.0 ms	0.47 %	52.0 ms	0.47 %	▶ block in PolyglotApp#js_person_as_hash	app.rb:2
39.0 ms	0.35 %	40.0 ms	0.36 %	▶ Sinatra::Base#root	base.rb:182
30.0 ms	0.27 %	42.0 ms	0.38 %	▶ PolyglotApp#GET /	app.rb:3
18.0 ms	0.16 %	18.0 ms	0.16 %	▶ block in Rack::Handler::WEBrick#service	webrick.rb:9
16.0 ms	0.14 %	16.0 ms	0.14 %	▶ WEBrick::HTTPResponse#setup_header	webrick.rb:1
13.0 ms	0.12 %	13.0 ms	0.12 %	▶ Rack::Protection::PathTraversal#cleanup	path_traversal.rb:2
13.0 ms	0.12 %	65.0 ms	0.59 %	▶ PolyglotApp#js_person_as_hash	app.rb:2
13.0 ms	0.12 %	14.0 ms	0.13 %	▶ Rack::Protection::JsonCsrf#has_vector?	json_csrf.rb:3
12.0 ms	0.11 %	20.0 ms	0.18 %	▶ Sinatra::Helpers#content_type	base.rb:34
11.0 ms	0.10 %	41.0 ms	0.37 %	▶ Sinatra::Base#static!	base.rb:106
9.0 ms	0.08 %	13.0 ms	0.12 %	▶ Rack::Protection::Base#html?	base.rb:12
8.0 ms	0.07 %	6323.0 ms	57.10 %	▶ block (2 levels) in WEBrick::GenericServer#start	single_threaded.rb:3
7.0 ms	0.06 %	11.0 ms	0.10 %	▶ Mustermann::Pattern#params	pattern.rb:20
6.0 ms	0.05 %	380.0 ms	3.43 %	▶ Sinatra::Base#invoke	base.rb:107
6.0 ms	0.05 %	423.0 ms	3.82 %	▶ Sinatra::Base#call!	base.rb:91
6.0 ms	0.05 %	6.0 ms	0.05 %	▶ Rack::Utils::HeaderHash#[]=	utils.rb:4

CPU-Tracer



```
-----------------------------------------------------------------------------------------------------------------------------------
Tracing Histogram. Counted a total of 51201 element executions.
  Total Count: Number of times the element was executed and percentage of total executions.
  Interpreted Count: Number of times the element was interpreted and percentage of total executions of this element.
  Compiled Count: Number of times the compiled element was executed and percentage of total executions of this element.
-----------------------------------------------------------------------------------------------------------------------------------
 Name                                               |          Total Count |    Interpreted Count |       Compiled Count | Location
-----------------------------------------------------------------------------------------------------------------------------------
 block in Mustermann::AST::Translator#decorator_for |          6768  13.2% |          6508  96.2% |           260   3.8% | /Users/pitr/Workspace/labs/graalvm-e
 Mustermann::AST::Node#payload=                     |          3305   6.5% |          3305 100.0% |             0   0.0% | /Users/pitr/Workspace/labs/graalvm-e
 Mustermann::AST::Translator.dispatch_table         |          2392   4.7% |          2302  96.2% |            90   3.8% | /Users/pitr/Workspace/labs/graalvm-e
 ossl_lock_unlock                                   |          2078   4.1% |          2078 100.0% |             0   0.0% | ../../graalvm-ee-1.0.0-rc6/Contents/
 ossl_lock_callback                                 |          2078   4.1% |          2078 100.0% |             0   0.0% | ../../graalvm-ee-1.0.0-rc6/Contents/
 is_rstring_ptr                                     |          1576   3.1% |          1501  95.2% |            75   4.8% | /Users/graal/slave/e/truffleruby/src
 rb_str_new                                         |          1576   3.1% |          1576 100.0% |             0   0.0% | /Users/graal/slave/e/truffleruby/src
 is_managed_rstring_ptr                             |          1576   3.1% |          1501  95.2% |            75   4.8% | /Users/graal/slave/e/truffleruby/src
 @strlen                                            |          1509   2.9% |           179  11.9% |          1330  88.1% | libsulong.bc:@strlen~1:0-19
 rb_str_new_cstr                                    |          1497   2.9% |          1497 100.0% |             0   0.0% | /Users/graal/slave/e/truffleruby/src
 Mustermann::AST::Translator::NodeTranslator#t      |          1074   2.1% |          1074 100.0% |             0   0.0% | /Users/pitr/Workspace/labs/graalvm-e
 Mustermann::AST::Translator::NodeTranslator#translator |          1074   2.1% |          1074 100.0% |             0   0.0% | /Users/pitr/Workspace/labs/graal
 rb_nativethread_self                               |          1043   2.0% |          1043 100.0% |             0   0.0% | /Users/graal/slave/e/truffleruby/src
 Mustermann::AST::Translator#translate              |           910   1.8% |           910 100.0% |             0   0.0% | /Users/pitr/Workspace/labs/graalvm-e
 Mustermann::AST::Translator#decorator_for          |           910   1.8% |           910 100.0% |             0   0.0% | /Users/pitr/Workspace/labs/graalvm-e
 Mustermann::AST::Translator::NodeTranslator#initialize |           910   1.8% |           910 100.0% |             0   0.0% | /Users/pitr/Workspace/labs/graal
 Mustermann::AST::Parser#buffer                     |           850   1.7% |           850 100.0% |             0   0.0% | /Users/pitr/Workspace/labs/graalvm-e
 rb_define_method                                   |           689   1.3% |           689 100.0% |             0   0.0% | /Users/graal/slave/e/truffleruby/src
 Mustermann::AST::Node.□                            |           584   1.1% |           584 100.0% |             0   0.0% | /Users/pitr/Workspace/labs/graalvm-e
 block in PolyglotApp#js_person_as_hash             |           564   1.1% |           564 100.0% |             0   0.0% | app.rb~27:623-681
 Mustermann::AST::Parser#pos                        |           484   0.9% |           484 100.0% |             0   0.0% | /Users/pitr/Workspace/labs/graalvm-e
 Mustermann::AST::Node#is_a?                        |           470   0.9% |           470 100.0% |             0   0.0% | /Users/pitr/Workspace/labs/graalvm-e
```

Visual Studio Code Plugins



graalvm

5 Results

Showing: All categories

Sort By: Relevance

GraalVM
Oracle Labs GraalVM 100
Basic support for GraalVM
FREE

GraalVM R
Oracle Labs GraalVM 37
Support for GraalVM R
FREE

GraalVM Python
Oracle Labs GraalVM 76
Support for GraalVM Python
FREE

GraalVM Ruby
Oracle Labs GraalVM 38
Support for GraalVM Ruby
FREE

GraalVM Extension Pa
Oracle Labs GraalVM 23
VS Code extensions for GraalVM based development.
FREE

https://marketplace.visualstudio.com/search?term=graalvm&target=VSCode&category=All%20categories&sortBy=Relevance

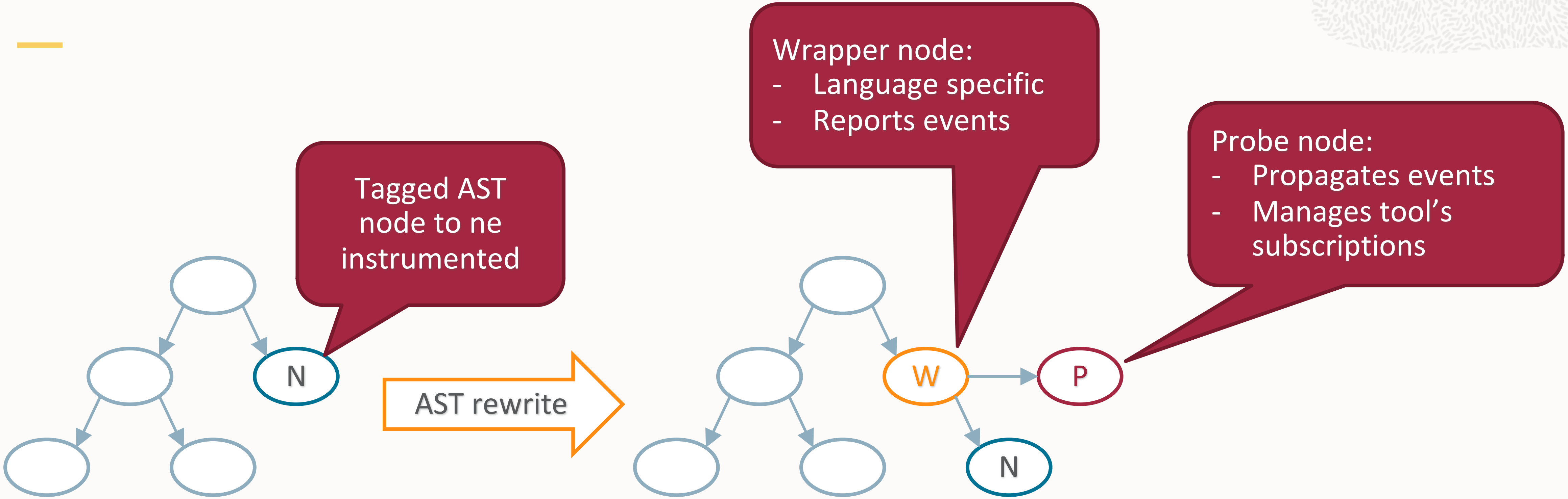
Instrumentation and Tooling

Instrumentation

Instrumentation

- One API built on top of rewriting AST
  - Languages need to only implement the instrumentation API
    - Minimal language support requirements
  - A language gets all the tools for **free**
  - Tools work for all languages
- Languages tags nodes
  - Expression, Call, Root, etc.
- Tools then can request to subscribe to events on tagged nodes
- Close to zero overhead when not used

Instrumentation



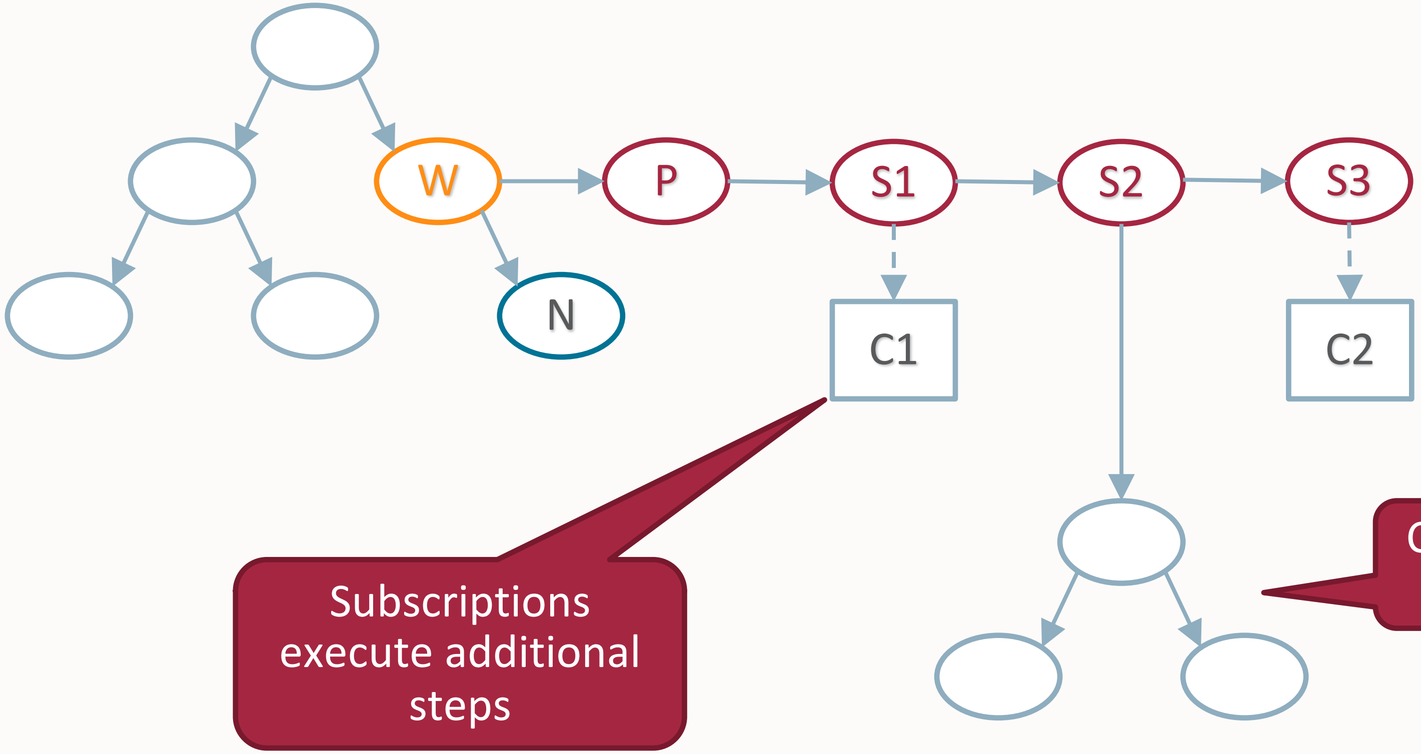
Tagged AST node to ne instrumented

AST rewrite

Wrapper node:
- Language specific
- Reports events

Probe node:
- Propagates events
- Manages tool’s subscriptions

Instrumentation



Subscriptions execute additional steps

Can also insert AST tree to execute

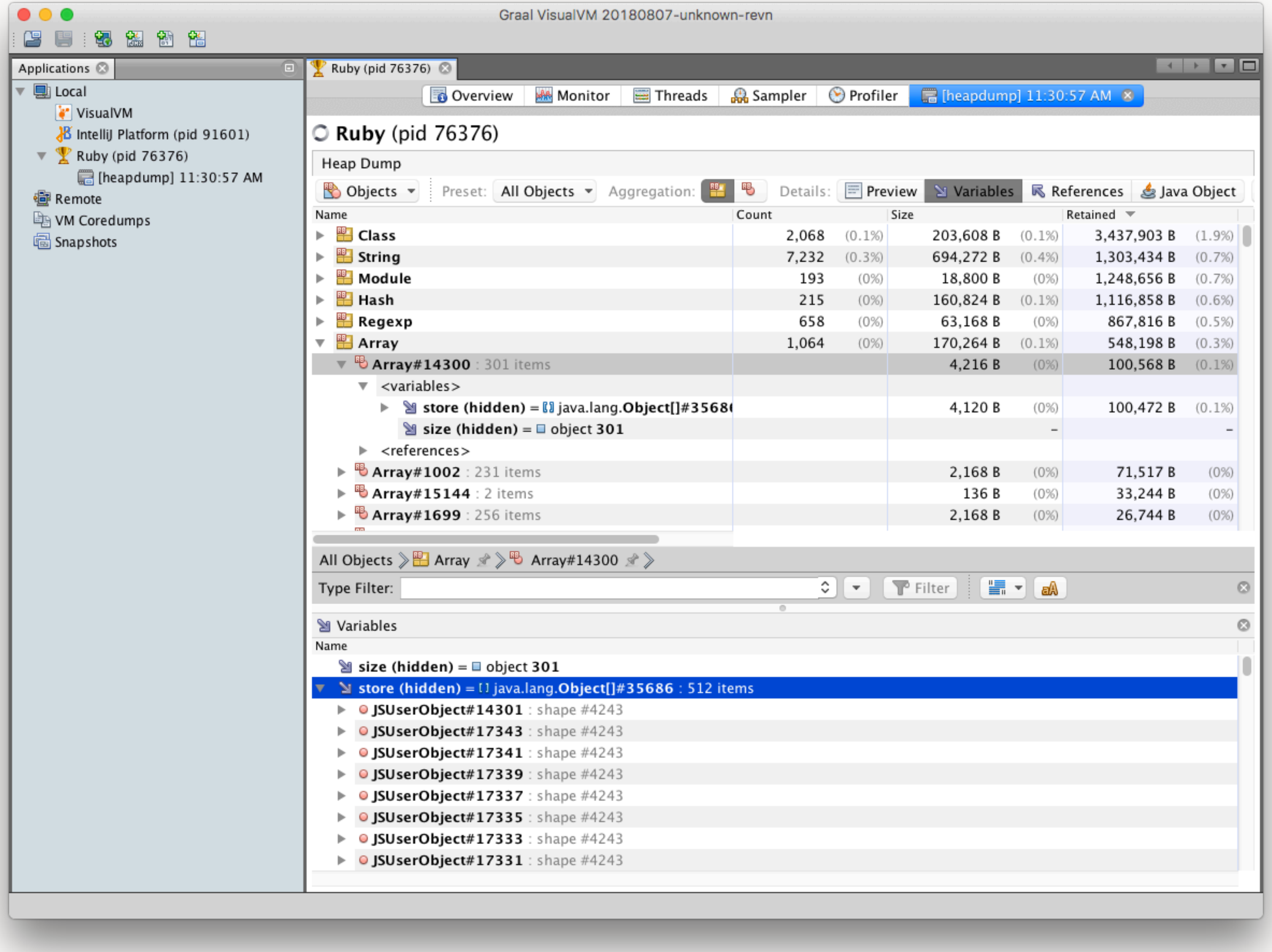
Implementing difficult language features

- Instrumentation is also used by the languages themselves
- R stepping
- Ruby `set_trace_func`
  - Attaches a function which gets executed on each: call, line, …

Graal VisualVM

Graal VisualVM

- Graal VisualVM has special support to understand Ruby, JS heap
- Compute retained sizes
  - The size including internal structures
    - String -> Rope -> byte[]
- Let’s create 300 extra people in the in-memory database
  - http://localhost:4567/add/300
  - We can lookup the array and its size (0.1%)



Graal VisualVM 20180807-unknown-revn

Ruby (pid 76376)

Heap Dump

Objects | Preset: All Objects | Aggregation: | Details: Preview | Variables | References | Java Object

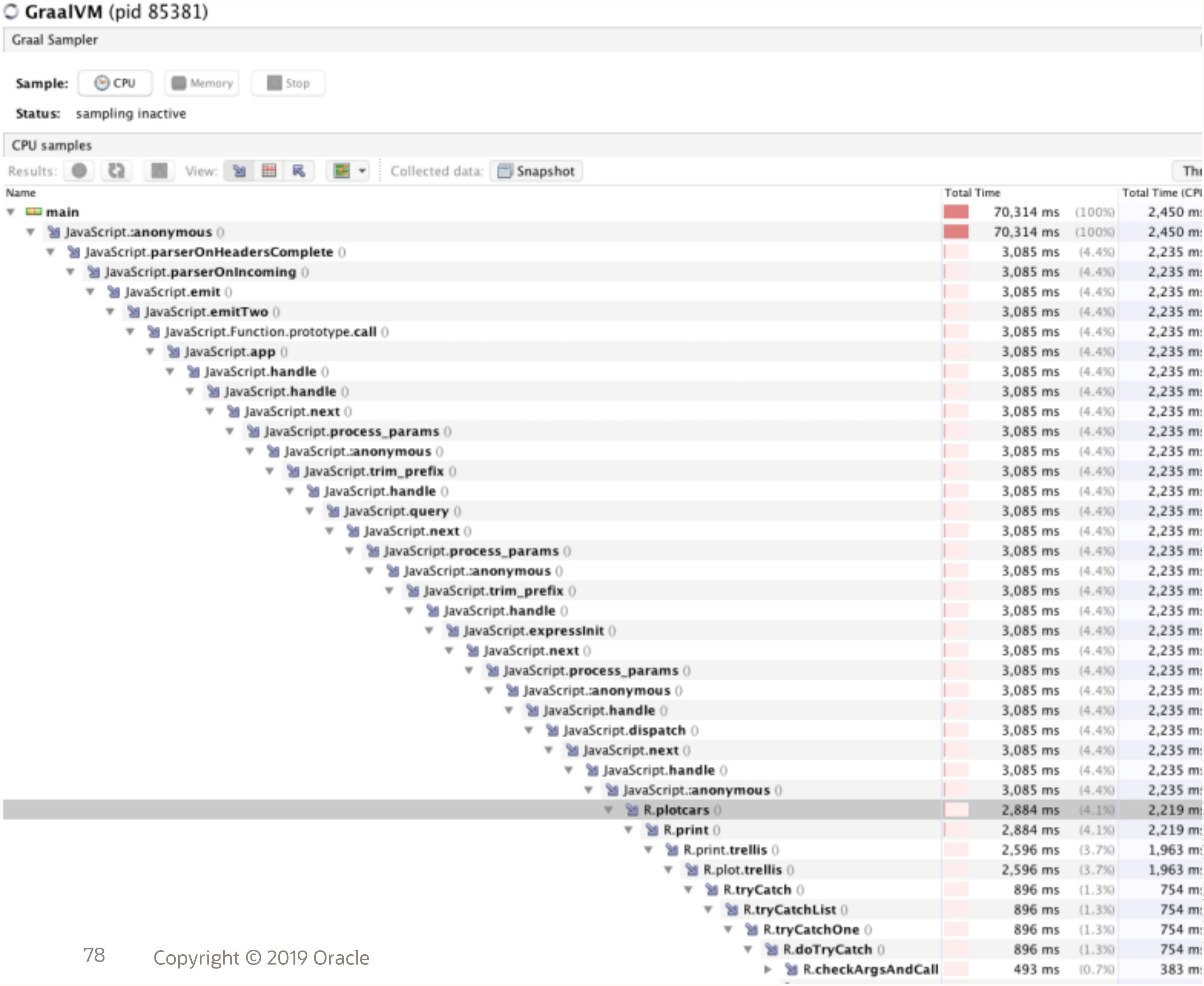
Name	Count		Size		Retained	
▶ Class	2,068	(0.1%)	203,608 B	(0.1%)	3,437,903 B	(1.9%)
▶ String	7,232	(0.3%)	694,272 B	(0.4%)	1,303,434 B	(0.7%)
▶ Module	193	(0%)	18,800 B	(0%)	1,248,656 B	(0.7%)
▶ Hash	215	(0%)	160,824 B	(0.1%)	1,116,858 B	(0.6%)
▶ Regexp	658	(0%)	63,168 B	(0%)	867,816 B	(0.5%)
▼ Array	1,064	(0%)	170,264 B	(0.1%)	548,198 B	(0.3%)
▼ Array#14300 : 301 items			4,216 B	(0%)	100,568 B	(0.1%)
▼ <variables>						
▶ store (hidden) = java.lang.Object[]#3568(			4,120 B	(0%)	100,472 B	(0.1%)
size (hidden) = object 301			–		–	
▶ <references>						
▶ Array#1002 : 231 items			2,168 B	(0%)	71,517 B	(0%)
▶ Array#15144 : 2 items			136 B	(0%)	33,244 B	(0%)
▶ Array#1699 : 256 items			2,168 B	(0%)	26,744 B	(0%)

All Objects > Array > Array#14300 >

Type Filter: | Filter

Variables

Name
size (hidden) = object 301
▼ store (hidden) = java.lang.Object[]#35686 : 512 items
▶ JSUserObject#14301 : shape #4243
▶ JSUserObject#17343 : shape #4243
▶ JSUserObject#17341 : shape #4243
▶ JSUserObject#17339 : shape #4243
▶ JSUserObject#17337 : shape #4243
▶ JSUserObject#17335 : shape #4243
▶ JSUserObject#17333 : shape #4243
▶ JSUserObject#17331 : shape #4243



GraalVM (pid 85381)

Graal Sampler

Sample: CPU Memory Stop

Status: sampling inactive

CPU samples

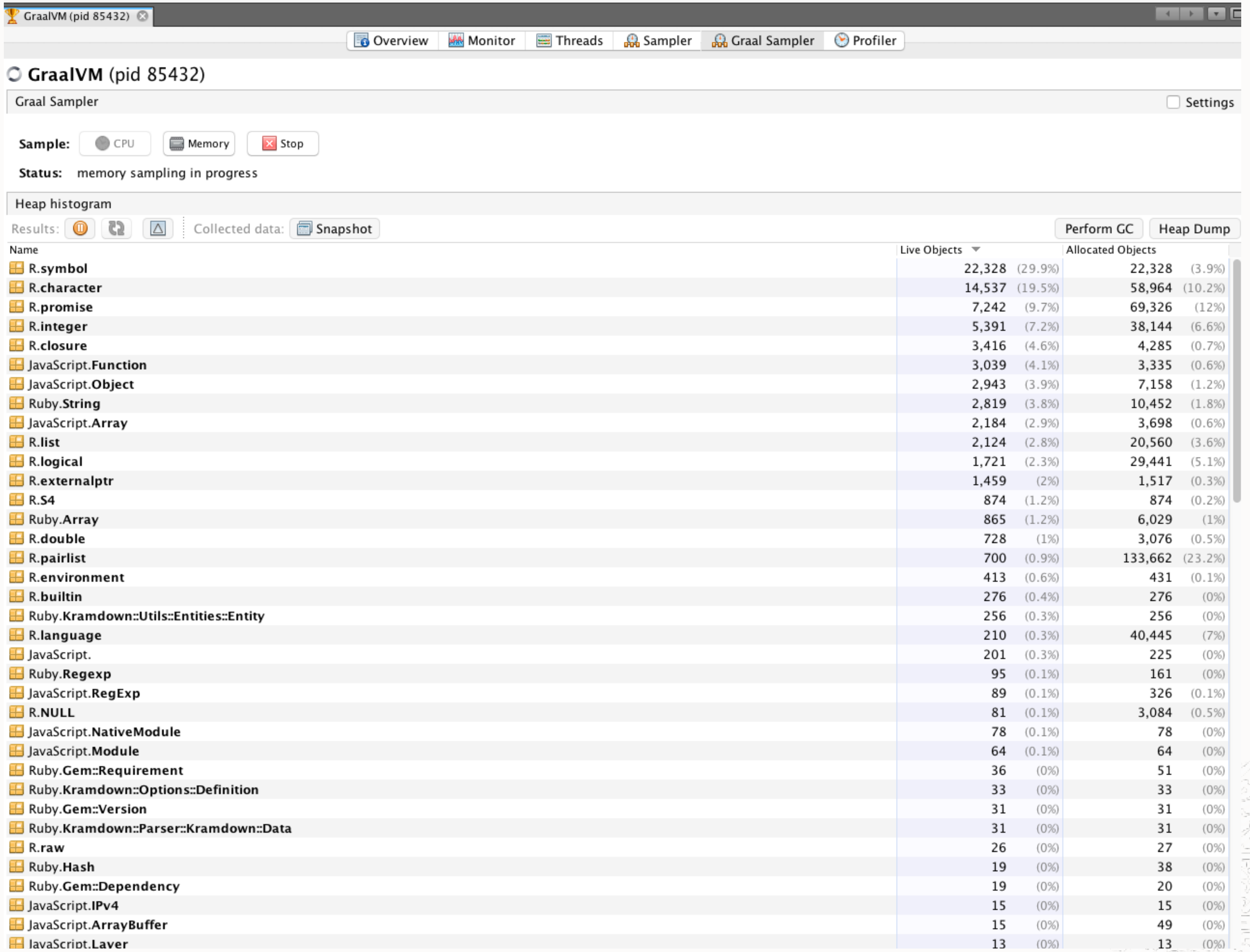
Results: View: Collected data: Snapshot Thr

```
Name                                                    Total Time              Total Time (CPI
▼ main                                                  70,314 ms  (100%)      2,450 m
 ▼ JavaScript.:anonymous ()                             70,314 ms  (100%)      2,450 m
  ▼ JavaScript.parserOnHeadersComplete ()                3,085 ms  (4.4%)      2,235 m
   ▼ JavaScript.parserOnIncoming ()                      3,085 ms  (4.4%)      2,235 m
    ▼ JavaScript.emit ()                                 3,085 ms  (4.4%)      2,235 m
     ▼ JavaScript.emitTwo ()                             3,085 ms  (4.4%)      2,235 m
      ▼ JavaScript.Function.prototype.call ()            3,085 ms  (4.4%)      2,235 m
       ▼ JavaScript.app ()                               3,085 ms  (4.4%)      2,235 m
        ▼ JavaScript.handle ()                           3,085 ms  (4.4%)      2,235 m
         ▼ JavaScript.handle ()                          3,085 ms  (4.4%)      2,235 m
          ▼ JavaScript.next ()                           3,085 ms  (4.4%)      2,235 m
           ▼ JavaScript.process_params ()                3,085 ms  (4.4%)      2,235 m
            ▼ JavaScript.:anonymous ()                   3,085 ms  (4.4%)      2,235 m
             ▼ JavaScript.trim_prefix ()                 3,085 ms  (4.4%)      2,235 m
              ▼ JavaScript.handle ()                     3,085 ms  (4.4%)      2,235 m
               ▼ JavaScript.query ()                     3,085 ms  (4.4%)      2,235 m
                ▼ JavaScript.next ()                     3,085 ms  (4.4%)      2,235 m
                 ▼ JavaScript.process_params ()          3,085 ms  (4.4%)      2,235 m
                  ▼ JavaScript.:anonymous ()             3,085 ms  (4.4%)      2,235 m
                   ▼ JavaScript.trim_prefix ()           3,085 ms  (4.4%)      2,235 m
                    ▼ JavaScript.handle ()               3,085 ms  (4.4%)      2,235 m
                     ▼ JavaScript.expressInit ()         3,085 ms  (4.4%)      2,235 m
                      ▼ JavaScript.next ()               3,085 ms  (4.4%)      2,235 m
                       ▼ JavaScript.process_params ()    3,085 ms  (4.4%)      2,235 m
                        ▼ JavaScript.:anonymous ()       3,085 ms  (4.4%)      2,235 m
                         ▼ JavaScript.handle ()          3,085 ms  (4.4%)      2,235 m
                          ▼ JavaScript.dispatch ()       3,085 ms  (4.4%)      2,235 m
                           ▼ JavaScript.next ()          3,085 ms  (4.4%)      2,235 m
                            ▼ JavaScript.handle ()       3,085 ms  (4.4%)      2,235 m
                             ▼ JavaScript.:anonymous ()  3,085 ms  (4.4%)      2,235 m
                              ▼ R.plotcars ()            2,884 ms  (4.1%)      2,219 m
                               ▼ R.print ()              2,884 ms  (4.1%)      2,219 m
                                ▼ R.print.trellis ()     2,596 ms  (3.7%)      1,963 m
                                 ▼ R.plot.trellis ()     2,596 ms  (3.7%)      1,963 m
                                  ▼ R.tryCatch ()          896 ms  (1.3%)        754 m
                                   ▼ R.tryCatchList ()     896 ms  (1.3%)        754 m
                                    ▼ R.tryCatchOne ()     896 ms  (1.3%)        754 m
                                     ▼ R.doTryCatch ()     896 ms  (1.3%)        754 m
                                      ▶ R.checkArgsAndCall 493 ms  (0.7%)        383 m
```

VisualVM

Polyglot Stack Trace

Polyglot Heap Dump



GraalVM (pid 85432)

Graal Sampler

□ Settings

Sample: CPU Memory Stop

Status: memory sampling in progress

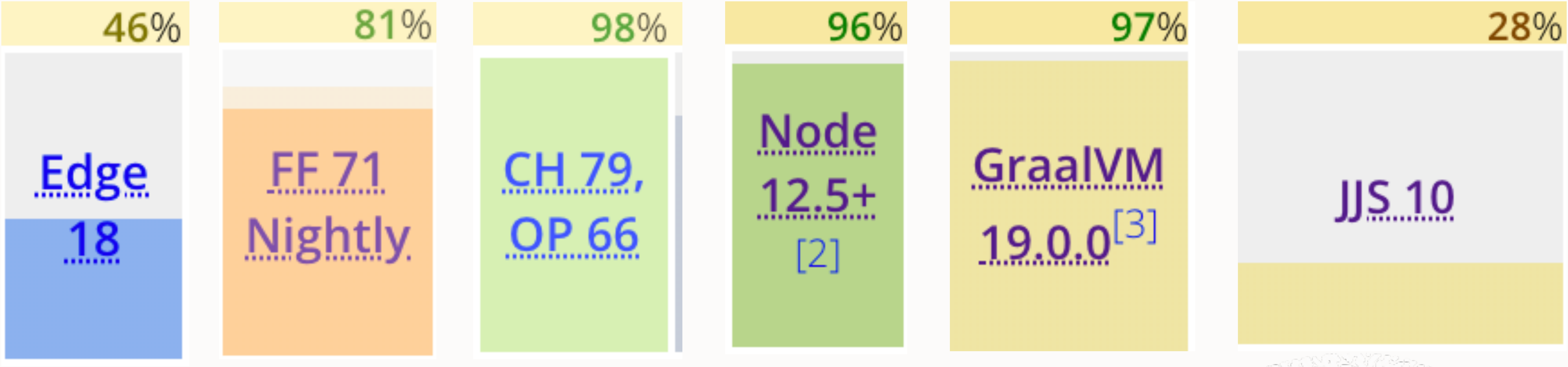
Heap histogram

Results: Collected data: Snapshot Perform GC Heap Dump

Name	Live Objects		Allocated Objects	
R.**symbol**	22,328	(29.9%)	22,328	(3.9%)
R.**character**	14,537	(19.5%)	58,964	(10.2%)
R.**promise**	7,242	(9.7%)	69,326	(12%)
R.**integer**	5,391	(7.2%)	38,144	(6.6%)
R.**closure**	3,416	(4.6%)	4,285	(0.7%)
JavaScript.**Function**	3,039	(4.1%)	3,335	(0.6%)
JavaScript.**Object**	2,943	(3.9%)	7,158	(1.2%)
Ruby.**String**	2,819	(3.8%)	10,452	(1.8%)
JavaScript.**Array**	2,184	(2.9%)	3,698	(0.6%)
R.**list**	2,124	(2.8%)	20,560	(3.6%)
R.**logical**	1,721	(2.3%)	29,441	(5.1%)
R.**externalptr**	1,459	(2%)	1,517	(0.3%)
R.**S4**	874	(1.2%)	874	(0.2%)
Ruby.**Array**	865	(1.2%)	6,029	(1%)
R.**double**	728	(1%)	3,076	(0.5%)
R.**pairlist**	700	(0.9%)	133,662	(23.2%)
R.**environment**	413	(0.6%)	431	(0.1%)
R.**builtin**	276	(0.4%)	276	(0%)
Ruby.**Kramdown::Utils::Entities::Entity**	256	(0.3%)	256	(0%)
R.**language**	210	(0.3%)	40,445	(7%)
JavaScript.	201	(0.3%)	225	(0%)
Ruby.**Regexp**	95	(0.1%)	161	(0%)
JavaScript.**RegExp**	89	(0.1%)	326	(0.1%)
R.**NULL**	81	(0.1%)	3,084	(0.5%)
JavaScript.**NativeModule**	78	(0.1%)	78	(0%)
JavaScript.**Module**	64	(0.1%)	64	(0%)
Ruby.**Gem::Requirement**	36	(0%)	51	(0%)
Ruby.**Kramdown::Options::Definition**	33	(0%)	33	(0%)
Ruby.**Gem::Version**	31	(0%)	31	(0%)
Ruby.**Kramdown::Parser::Kramdown::Data**	31	(0%)	31	(0%)
R.**raw**	26	(0%)	27	(0%)
Ruby.**Hash**	19	(0%)	38	(0%)
Ruby.**Gem::Dependency**	19	(0%)	20	(0%)
JavaScript.**IPv4**	15	(0%)	15	(0%)
JavaScript.**ArrayBuffer**	15	(0%)	49	(0%)
JavaScript.**Layer**	13	(0%)	13	(0%)

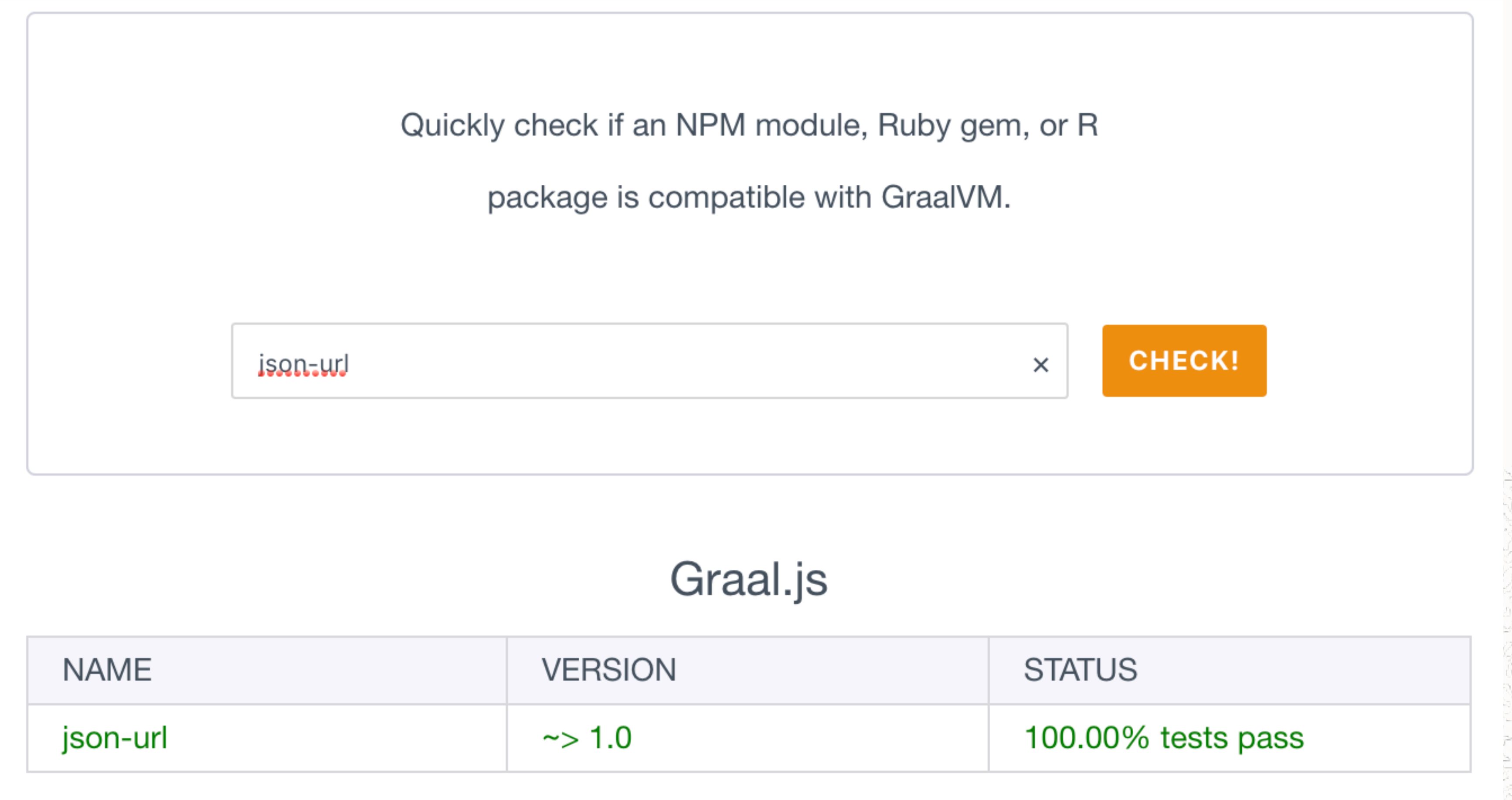
Language status

JS compatibility



46%	81%	98%	96%	97%	28%
Edge 18	FF 71 Nightly	CH 79, OP 66	Node 12.5+ [2]	GraalVM 19.0.0[3]	JJS 10

https://www.graalvm.org/docs/reference-manual/compatibility/



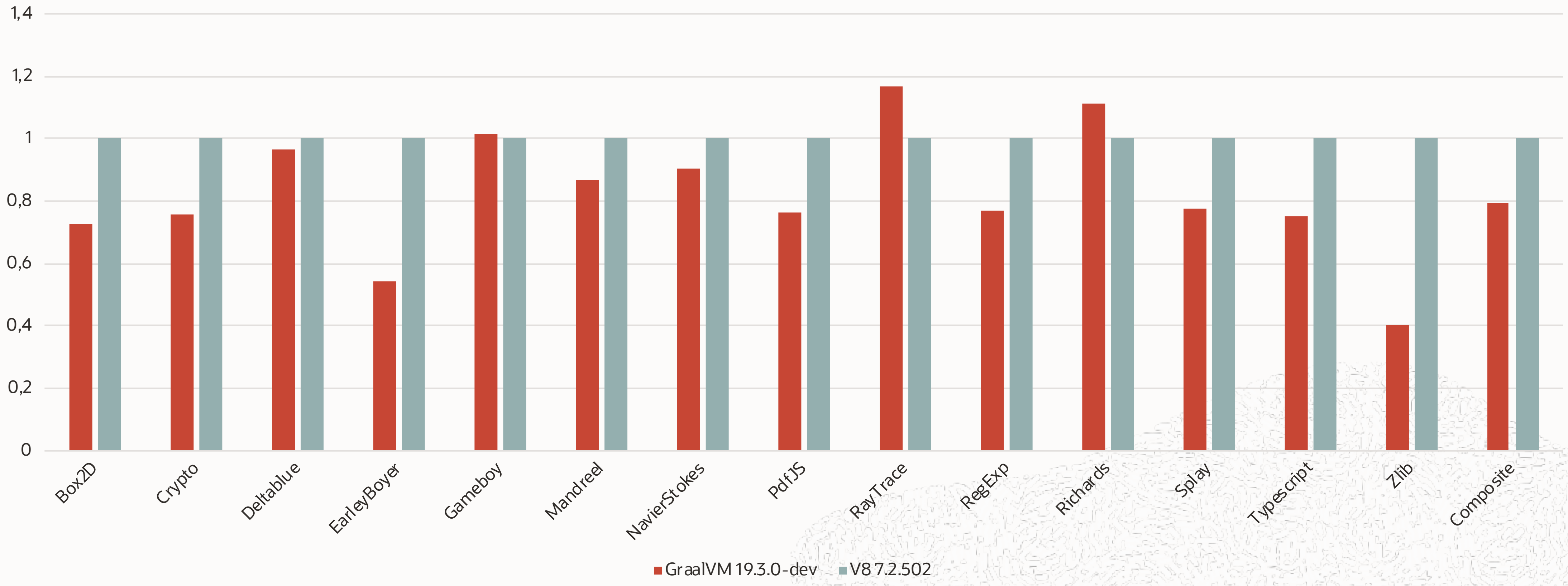
Quickly check if an NPM module, Ruby gem, or R package is compatible with GraalVM.

json-url

Graal.js

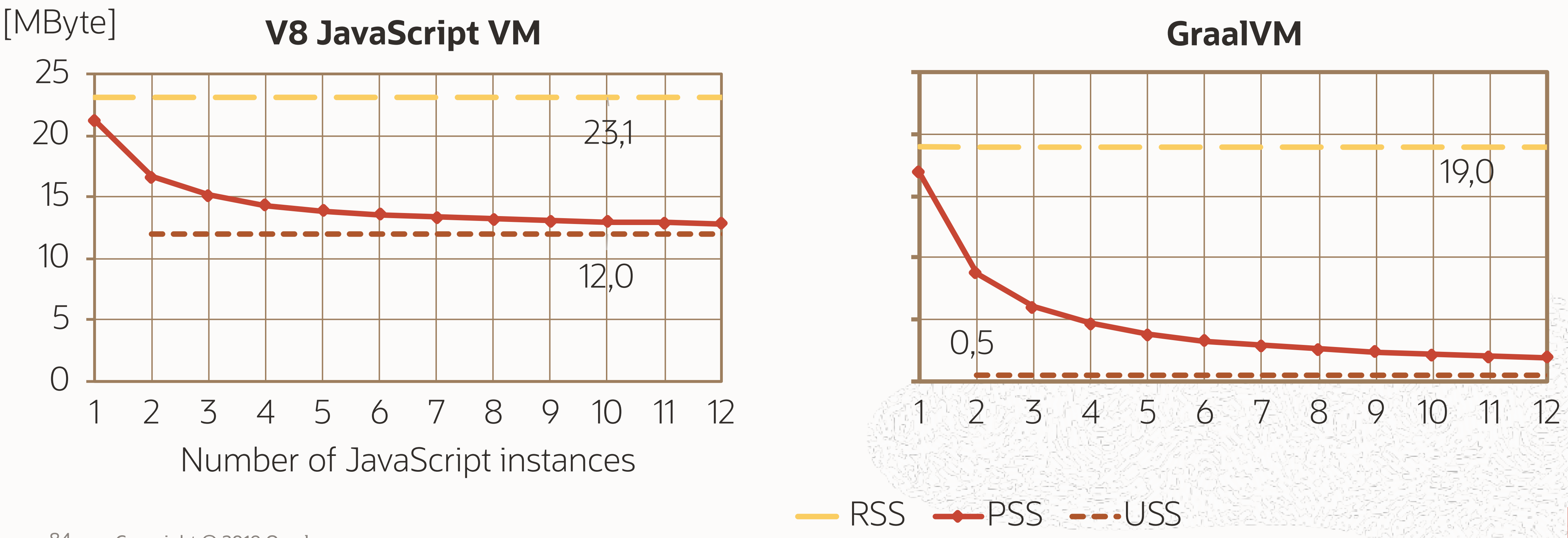
NAME	VERSION	STATUS
json-url	~> 1.0	100.00% tests pass

Graal.js Performance (versus V8)

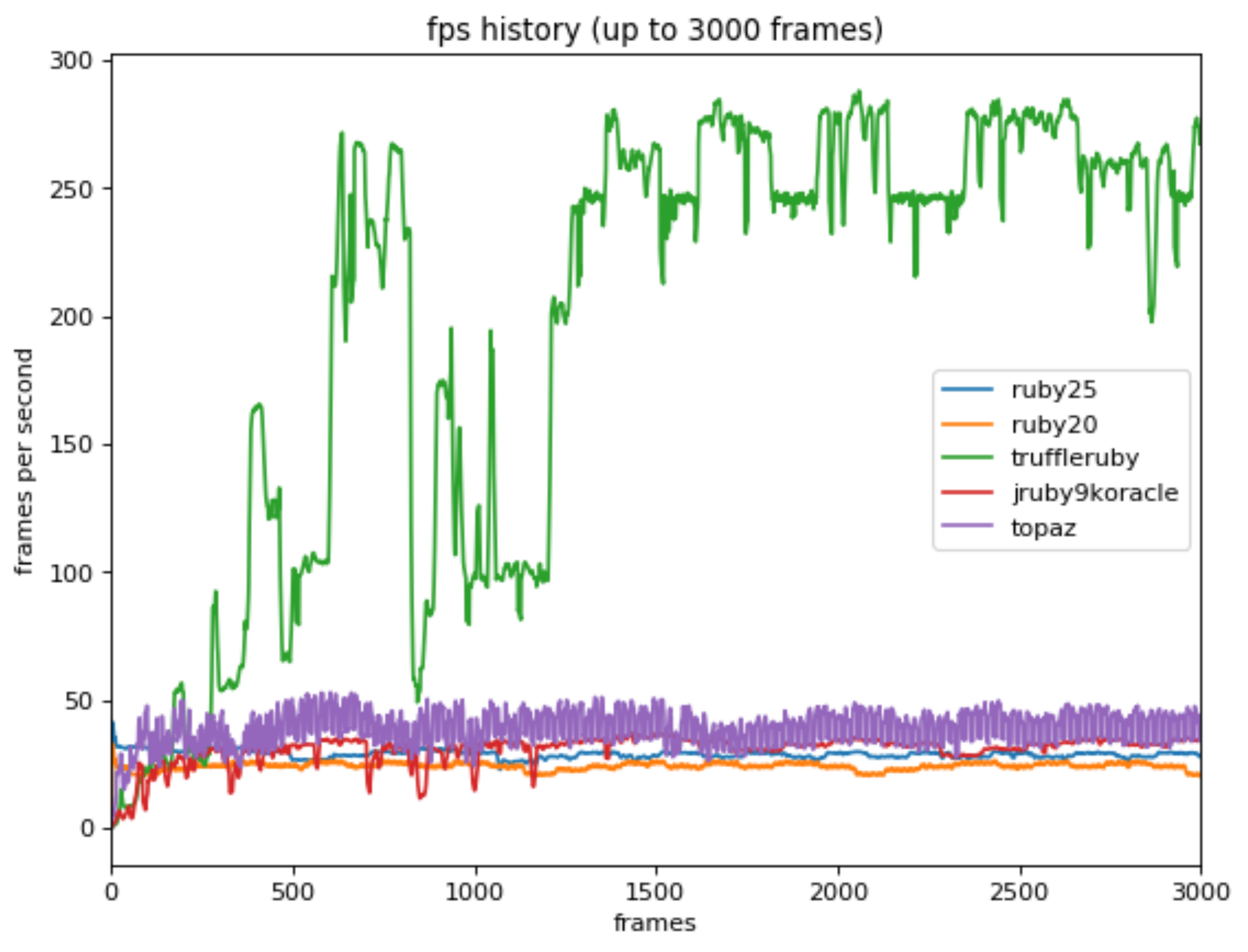


JavaScript Memory Footprint: V8 vs. GraalVM

- Memory for the first JavaScript instance: **23 MByte vs. 19 MByte**
- Memory for each additional JavaScript instance: **12 MByte vs. 0.5 MByte**

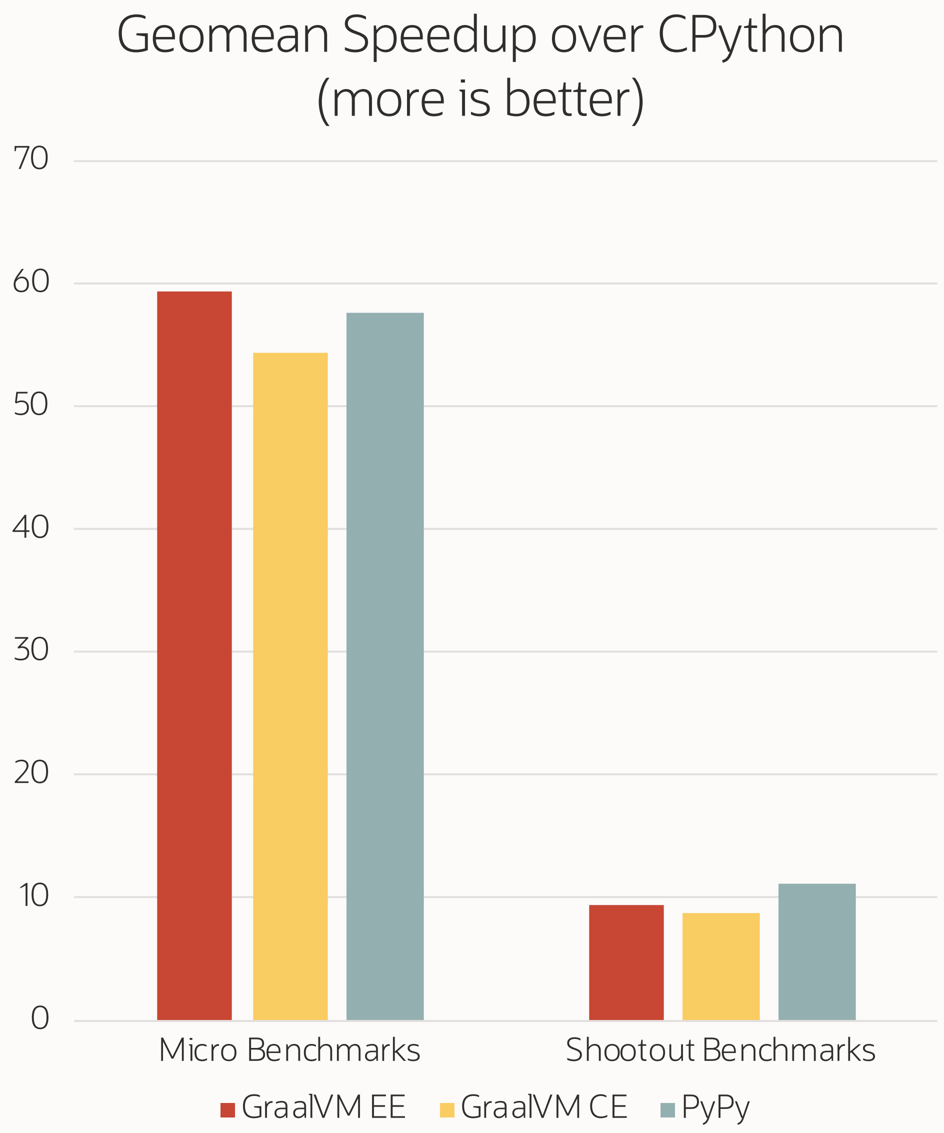


Optcarrot – TruffleRuby performance



Python Performance

Comparable to PyPy, the fastest alternative



Conclusion

Conclusion

For User

- Polyglot
- Tooling
- Performance
- Compatibility

For language implementer

- AST interpreter – simple implementation
- All the tooling from the start
  - Debugging
  - Profiling
- Performance from the beginning
- AOT compilation
  - Executable
  - Embedding

Production-Ready

Java

Scala, Groovy, Kotlin

JavaScript

Node.js

Native Image

VisualVM

Experimental

Ruby

R

LLVM Toolchain

Visionary

Python

VSCode Plugin

GPU Integration

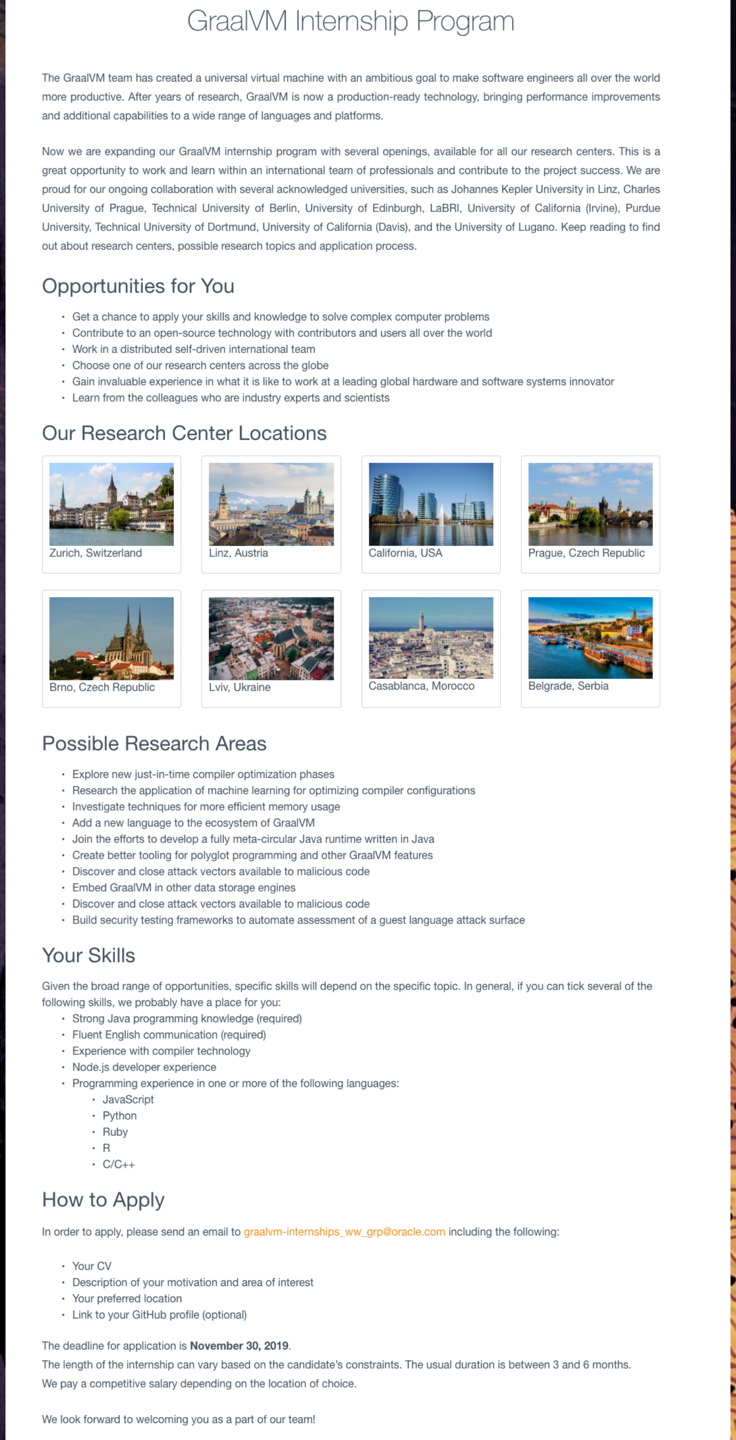
Webassembly

LLVM Backend

Internship

https://www.graalvm.org/community/internship/

The deadline for application is
November 30, 2019



GraalVM Internship Program

The GraalVM team has created a universal virtual machine with an ambitious goal to make software engineers all over the world more productive. After years of research, GraalVM is now a production-ready technology, bringing performance improvements and additional capabilities to a wide range of languages and platforms.

Now we are expanding our GraalVM internship program with several openings, available for all our research centers. This is a great opportunity to work and learn within an international team of professionals and contribute to the project success. We are proud for our ongoing collaboration with several acknowledged universities, such as Johannes Kepler University in Linz, Charles University of Prague, Technical University of Berlin, University of Edinburgh, LaBRI, University of California (Irvine), Purdue University, Technical University of Dortmund, University of California (Davis), and the University of Lugano. Keep reading to find out about research centers, possible research topics and application process.

Opportunities for You

- Get a chance to apply your skills and knowledge to solve complex computer problems
- Contribute to an open-source technology with contributors and users all over the world
- Work in a distributed self-driven international team
- Choose one of our research centers across the globe
- Gain invaluable experience in what it is like to work at a leading global hardware and software systems innovator
- Learn from the colleagues who are industry experts and scientists

Our Research Center Locations

Zurich, Switzerland

Linz, Austria

California, USA

Prague, Czech Republic

Brno, Czech Republic

Lviv, Ukraine

Casablanca, Morocco

Belgrade, Serbia

Possible Research Areas

- Explore new just-in-time compiler optimization phases
- Research the application of machine learning for optimizing compiler configurations
- Investigate techniques for more efficient memory usage
- Add a new language to the ecosystem of GraalVM
- Join the efforts to develop a fully meta-circular Java runtime written in Java
- Create better tooling for polyglot programming and other GraalVM features
- Discover and close attack vectors available to malicious code
- Embed GraalVM in other data storage engines
- Discover and close attack vectors available to malicious code
- Build security testing frameworks to automate assessment of a guest language attack surface

Your Skills

Given the broad range of opportunities, specific skills will depend on the specific topic. In general, if you can tick several of the following skills, we probably have a place for you:

- Strong Java programming knowledge (required)
- Fluent English communication (required)
- Experience with compiler technology
- Node.js developer experience
- Programming experience in one or more of the following languages:
  - JavaScript
  - Python
  - Ruby
  - R
  - C/C++

How to Apply

In order to apply, please send an email to graalvm-internships_ww_grp@oracle.com including the following:

- Your CV
- Description of your motivation and area of interest
- Your preferred location
- Link to your GitHub profile (optional)

The deadline for application is **November 30, 2019**.

The length of the internship can vary based on the candidate’s constraints. The usual duration is between 3 and 6 months.

We pay a competitive salary depending on the location of choice.

We look forward to welcoming you as a part of our team!

Thank you.

Questions?

https://www.graalvm.org/

Safe harbor statement

The preceding 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.

GraalVM Native Image technology (including SubstrateVM) is early adopter technology. It is available only under an early adopter license and remains subject to potentially significant further changes, compatibility testing and certification.

ORACLE