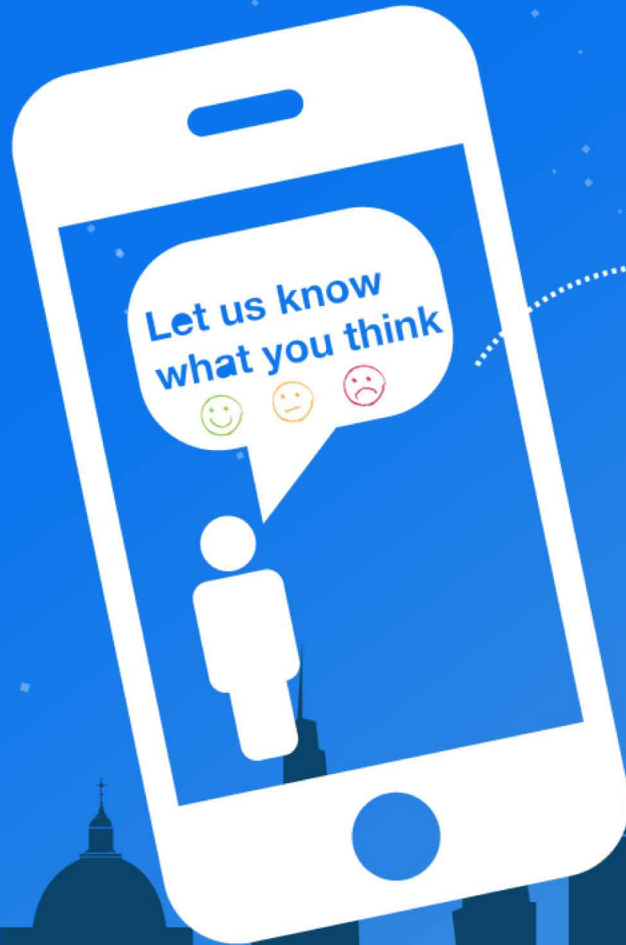


# Turning the JVM into a Polyglot VM with Graal

*Speaker Chris Seaton*





**Click 'Rate Session'  
to rate session  
and ask questions.**

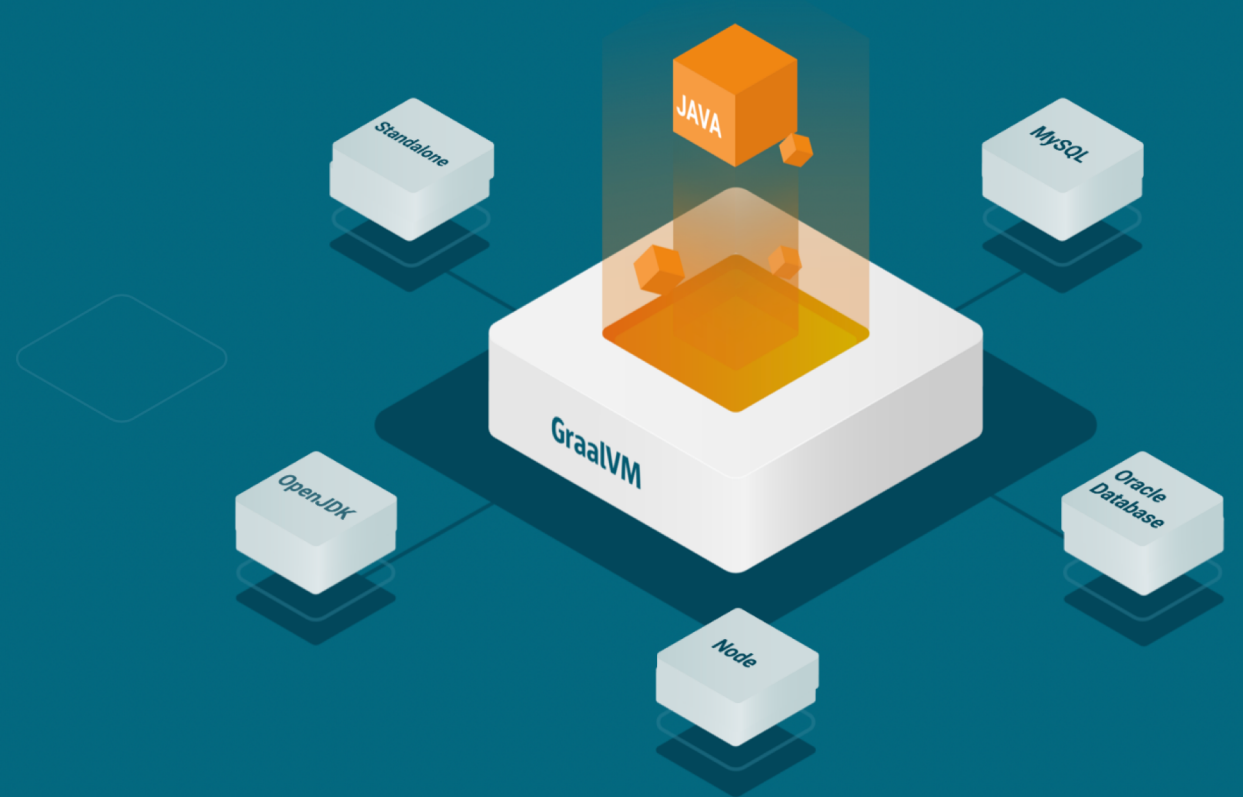


## Safe Harbor Statement

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.

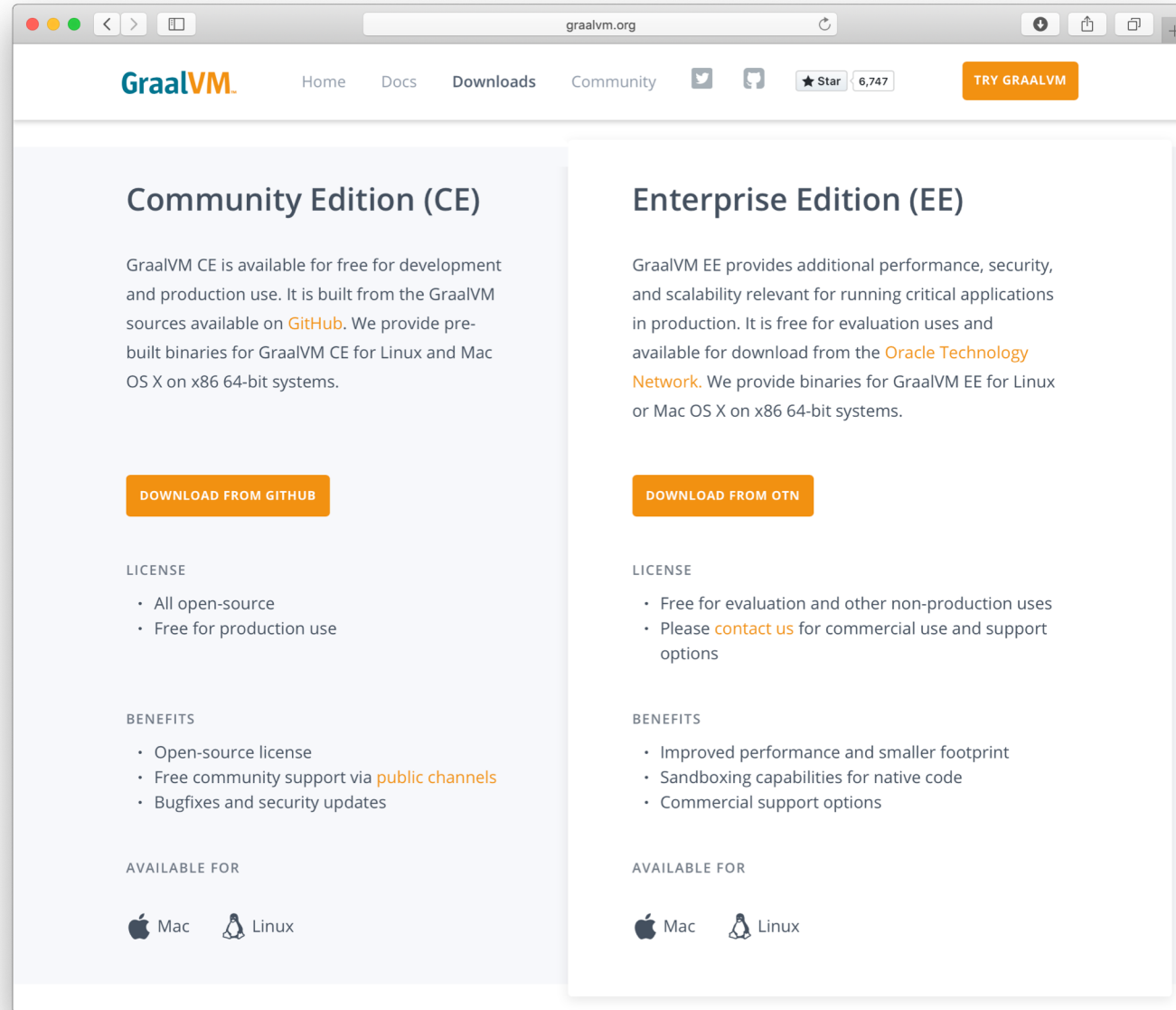
# GraalVM™

Run Programs Faster Anywhere





- What, in concrete terms, is GraalVM?
- What can I practically do with it?
- What do all these things have to do with each other?
- What is the big idea?
- What kind of change is this going to enable?





# GraalVM for high performance Java

# Using GraalVM as your JDK

- You can use GraalVM as a drop-in replacement for OpenJDK
- 1.8 at the moment, will be updated to the 11 LTS soon
- Includes all the same commands, flags, options and so on



# Add it to your \$PATH

```
$ export PATH=graalvm-ee-1.0.0-rc8/Contents/Home/bin:$PATH
```

```
public static void main(String[] args) {  
    Arrays.stream(args)  
        .flatMap(TopTen::fileLines)  
        .flatMap(line -> Arrays.stream(line.split("\\b")))  
        .map(word -> word.replaceAll("[^a-zA-Z]", ""))  
        .filter(word -> word.length() > 0)  
        .map(word -> word.toLowerCase())  
        .collect(Collectors.groupingBy(Function.identity(), Collectors.counting()))  
        .entrySet().stream()  
        .sorted((a, b) -> -a.getValue().compareTo(b.getValue()))  
        .limit(10)  
        .forEach(e -> System.out.format("%s = %d%n", e.getKey(), e.getValue()));  
}
```

# Compile and run as normal


```
$ javac TopTen.java  
$ time java TopTen large.txt  
...  
real 0m18.905s
```



This demo is  
run with the  
EE version

# Compare to standard OpenJDK

```
$ time java -XX:-UseJVMCICompiler TopTen large.txt  
...  
real 0m23.102s
```



I will explain  
this flag  
shortly...



# What is going on?

- The Graal just-in-time compiler is one part of GraalVM
- It replaces (or runs as a tier above) the existing JIT compilers like C2
- It's written in Java, which we think lets us improve it more easily, so it achieves better performance than C2
- Here we're getting 20% faster performance on a benchmark
- Twitter see 18% faster in production on real Scala applications, using only the CE version – EE not needed for high performance
- Is it odd that a JIT compiler for Java is written in Java?

```
synchronized (object) {  
    ...A  
}
```

```
synchronized (object) {  
    ...B  
}
```

```
synchronized (object) {  
    ...A  
    ...B  
}
```

```

for (MonitorExitNode monitorExitNode : graph.getNodes(MonitorExitNode.TYPE)) {
    FixedNode next = monitorExitNode.next();
    if ((next instanceof MonitorEnterNode || next instanceof RawMonitorEnterNode)) {
        // should never happen, osr monitor enters are always direct successors of the graph
        // start
        assert !(next instanceof OSRMonitorEnterNode);
        AccessMonitorNode monitorEnterNode = (AccessMonitorNode) next;
        if (isCompatibleLock(monitorEnterNode, monitorExitNode)) {
            /*
             * We've coarsened the lock so use the same monitor id for the whole region,
             * otherwise the monitor operations appear to be unrelated.
             */
            MonitorIdNode enterId = monitorEnterNode.getMonitorId();
            MonitorIdNode exitId = monitorExitNode.getMonitorId();
            if (enterId != exitId) {
                enterId.replaceAndDelete(exitId);
            }
            GraphUtil.removeFixedWithUnusedInputs(monitorEnterNode);
            GraphUtil.removeFixedWithUnusedInputs(monitorExitNode);
        }
    }
}

```

JVMCI is the interface that lets you plug in a new JIT

```
$ time java -XX:-UseJVMCICompiler TopTen large.txt  
...  
real 0m23.102s
```





Also works on OpenJDK 11

# Graal also works on standard OpenJDK 11

- Graal (the JIT compiler part) is also included in Graal
- As an experimental, unsupported option, hidden behind flags
- Older version, due to the release cycle
- We'd recommend using the GraalVM package to experiment with

# Enable Graal in OpenJDK 11

On AMD64, on  
macOS and  
Linux

```
$ java -XX:+UnlockExperimentalVMOptions \
      -XX:+EnableJVMCI \
      -XX:+UseJVMCICompiler \
```

...

Doesn't mention Graal, does  
it? JVMCI does service  
discovery and automatically  
finds Graal as the only JVMCI  
compiler available



# GraalVM for low footprint, fast startup



# What about shorter running applications or functions?

- The JVM typically has a relatively slow time to start
  - Compared to simpler VMs, like Python or Ruby
  - Compared to native executables like those produced from Go or Rust
  - JRuby 'hello world' startup time is an order of magnitude worse than standard Ruby
- The JVM typically takes up a relatively large amount of disk space
  - Can be helped with `jlink` – down to tens of MB
- The JVM typically takes up a relatively large amount of RAM
  - Interpreter, compiler, classfile parser, verifier etc all take up space

# Run as normal

```
$ time java TopTen small.txt
```

```
...
```

```
real 0m0.408s
```

# Compile to native using GraalVM

```
$ native-image TopTen  
...  
$ time ./topten small.txt  
...  
real 0m0.112s
```

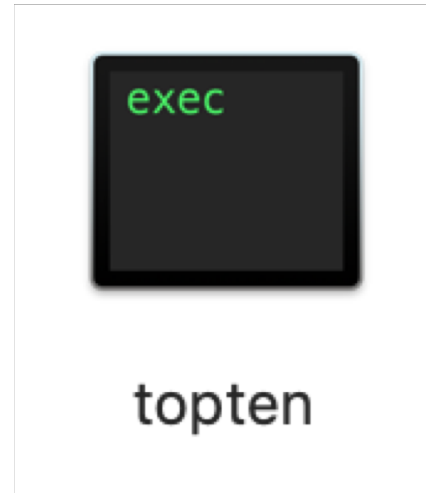


topten



topten

```
$ du -h topten  
8.8M topten
```



```
$ otool -L topten  
topten:  
/usr/lib/libSystem.B.dylib  
/usr/lib/libz.1.dylib  
/System/Library/CoreFoundation
```

topten.hop

mov add

If(b) r0;

486f78  
786572  
204469

Labels Proc. Str ☆ ●

toLowerCase

Tag Scope

Address	Type	Name
0x10009c030	P	_java.lang.Character.toLowerCase(int)int
0x10009d600	P	_java.lang.CharacterData00.toLowerCa...
0x1000a1310	P	_java.lang.CharacterData01.toLowerCa...
0x1000a2000	P	_java.lang.CharacterData02.toLowerCa...
0x1000a2de0	P	_java.lang.CharacterData0E.toLowerCa...
0x1000a3930	P	_java.lang.CharacterDataLatin1.toLowe...
0x1000a3fd0	P	_java.lang.CharacterDataPrivateUse.to...
0x1000a4270	P	_java.lang.CharacterDataUndefined.toL...
0x1000a9de0	P	_java.lang.ConditionalSpecialCasing.to...
0x1000b94f0	P	_java.lang.String.toLowerCase(java.util...
0x10009d180	P	_java.lang.CharacterData00.isOtherLo...
0x1000a1140	P	_java.lang.CharacterData01.isOtherLo...
0x1000a1bb0	P	_java.lang.CharacterData02.isOtherLo...
0x1000a2990	P	_java.lang.CharacterData0E.isOtherLo...
0x1000a3070	P	_java.lang.CharacterData.isOtherLower...
0x1000a3670	P	_java.lang.CharacterDataLatin1.isOther...
0x100301e70	P	_sun.nio.cs.UTF_8\$Decoder.xflow(java....
0x100301f70	P	_sun.nio.cs.UTF_8\$Decoder.xflow(java....
0x1001445d0	P	_java.security.UnresolvedPermission.n...
0x100017490	P	_com.oracle.svm.core.deopt.Deoptimiz...
0x10002e630	P	_com.oracle.svm.core.genscavenge.gr...
0x10004fe90	P	_com.oracle.svm.core.posix.headers.S...
0x1000553c0	P	_com.oracle.svm.core.stack.ThreadSta...
0x1000555f0	P	_com.oracle.svm.core.stack.ThreadSta...
0x100055800	P	_com.oracle.svm.core.stack.ThreadSta...

49157 labels

☒ Remove HI/LO macros

☒ Remove potentially dead code

☒ Remove NOPs

☐ No code duplicati...

```
int _java.lang.Character.toLowerCase(int)int(int arg0) {  
    rdi = arg0;  
    rsp = rsp - 0x18;  
    if ((rdi & 0xffffffff00) != 0x0) {  
        rsi = rdi >> 0x10;  
        if (rsi < 0xe) {  
            if (rsi <= 0x2) {  
                if (rsi < 0x2) {  
                    if (rsi != 0x0) {  
                        if (rsi != 0x1) {  
                            rsi = *qword_value_4298892832;  
                        }  
                        else {  
                            rsi = *qword_value_4298892816;  
                        }  
                    }  
                    else {  
                        rsi = *qword_value_4298892848;  
                    }  
                }  
                else {  
                    rsi = *qword_value_4298892864;  
                }  
            }  
            else {  
                rsi = *qword_value_4298892832;  
            }  
        }  
        else {  
            if (rsi <= 0x10) {  
                if ((rsi < 0x10) && (rsi == 0xe)) {  
                    rsi = *qword_value_4298892880;  
                }  
                else {  
                    rsi = *qword_value_4298892784;  
                }  
            }  
            else {  
                rsi = *qword_value_4298892832;  
            }  
        }  
    }  
}
```

> dataflow analysis of procedures in segment \_\_LINKEDIT  
> dataflow analysis of procedures in segment External Symbols  
> Analysis pass 9/10: remaining prologs search  
> Analysis pass 10/10: searching contiguous code area  
> Last pass done  
Background analysis ended in 0'46

>>> Python Command

File Information

Path: /Users/chrisseaton/Documents/orac

Loader: Mach-O

CPU: intel/x86\_64

☐ Branch always stops procedures

CPU Syntax Variant: AT&T

Calling Convention: System V

Address Information

Type: Procedure

Prolog Heuristic: Not a procedure prol

Prolog Mode: Use Heuristic

Navigation History

0x100008d30 (\_com.oracle.svm.core.code.CEntryF  
0x1000019e0 (\_TopTen.main(java.lang.String[])voic  
0x1000b0430 (\_java.lang.Math.scalb(double,int)dc  
0x1000b0190 (\_java.lang.Math.log(double,double)  
0x1000b05a0 (\_ java.lan.Math.toIntExact(long)lnt)

Clear Navigation Stack

Graphic Views

Procedure

21 basic blocks  
int func(int)

Edit

Calling Convention: File default (System...

Call graph

Type	Callers
Direct	_java.util.regex.Pattern\$SliceU.matc...
Direct	_java.lang.String\$CaseInsensitiveCo...
Direct	_java.lang.String.regionMatches(boole...
Direct	iava.lan.Strin.toLowerCase(iava.u...

Type	At	Method Called
------	----	---------------

# So what does this have to do with a JIT?

- Graal is written in Java
- So it can be used as a library from other Java code
- We realized that we could write a program to use it ahead-of-time, to build and ahead-of-time compiler
- This is what the `native-image` tool is – a Java application that uses Graal as a library



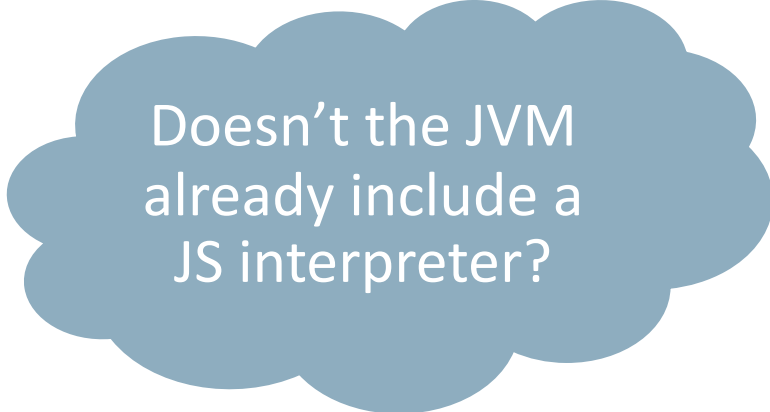


# GraalVM to run other languages

# GraalVM includes a new JavaScript interpreter

```
$ js -version  
Graal JavaScript 1.0 (GraalVM CE Native 1.0.0-rc8)
```

```
$ js  
> print("hello");  
hello
```



Doesn't the JVM  
already include a  
JS interpreter?

# GraalVM also includes an implementation of Node.js

```
$ node --version  
v10.9.0
```

```
$ npm --version  
6.2.0
```

```
var express = require('express');
var app = express();

app.get('/', function (req, res) {
  res.send('<h1>Hello!</h1>');
});

app.listen(8080, function () {
  console.log('serving at http://localhost:8080')
});
```

```
$ npm install express
```

```
$ node hello-express.js  
serving at http://localhost:8080
```

# You can plug new languages into GraalVM

```
$ gu install ruby  
$ gu install python  
$ gu install R
```

# You can plug new languages into GraalVM

```
$ ruby --version  
truffleruby 1.0.0-rc8, like ruby 2.4.4, GraalVM CE Native [x86_64-darwin]
```

```
$ graalpython --version  
Graal Python 3.7.0 (GraalVM CE Native 1.0.0-rc8)
```

```
$ R --version  
R version 3.4.0 (FastR)
```

# So what does this have to do with a JIT?

- We realized instead of writing languages that emit bytecode at runtime (JRuby) we could write languages that use the Graal JIT directly
- But that's hard, so we realized we could write a framework, Truffle, to do that automatically, based on a simple interpreter
- Faster, as they use a more powerful JIT more directly
- Simpler, because a framework does most of the hard work, so easy to implement lots of languages
- Interopable (polyglot) because they all use the same system
- Using `native-image` they start quickly



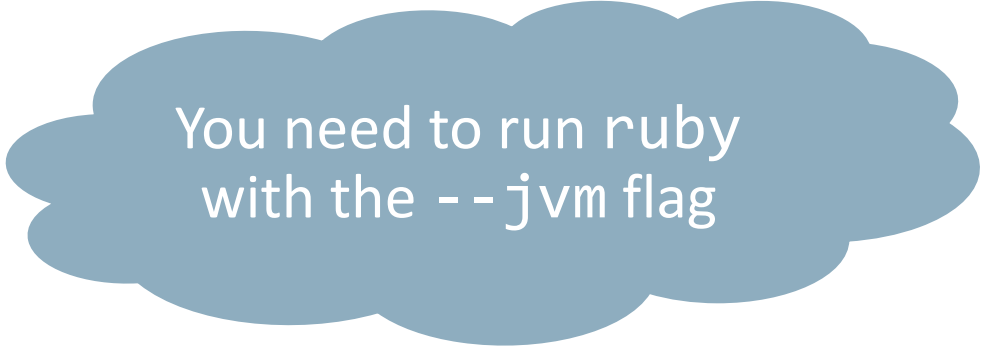


# GraalVM to run polyglot programs

# GraalVM is polyglot as well as multi-language

- Like many languages implemented on the JVM, our languages can use Java libraries
- Run `ruby-java.rb`

```
BigInteger = Java.type('java.math.BigInteger')  
puts BigInteger.valueOf(2).pow(100).toString
```

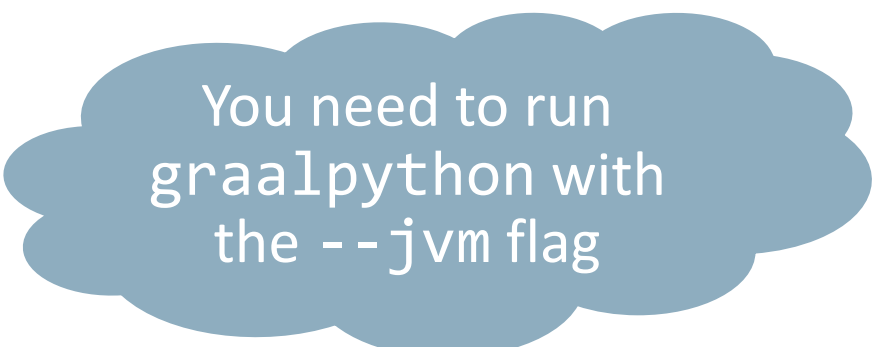


You need to run ruby  
with the `--jvm` flag

# GraalVM is polyglot as well as multi-language

- Run python-java.rb

```
import java
BigInteger = java.type('java.math.BigInteger')
print(BigInteger.valueOf(2).pow(100).toString())
```



You need to run  
graalpython with  
the --jvm flag

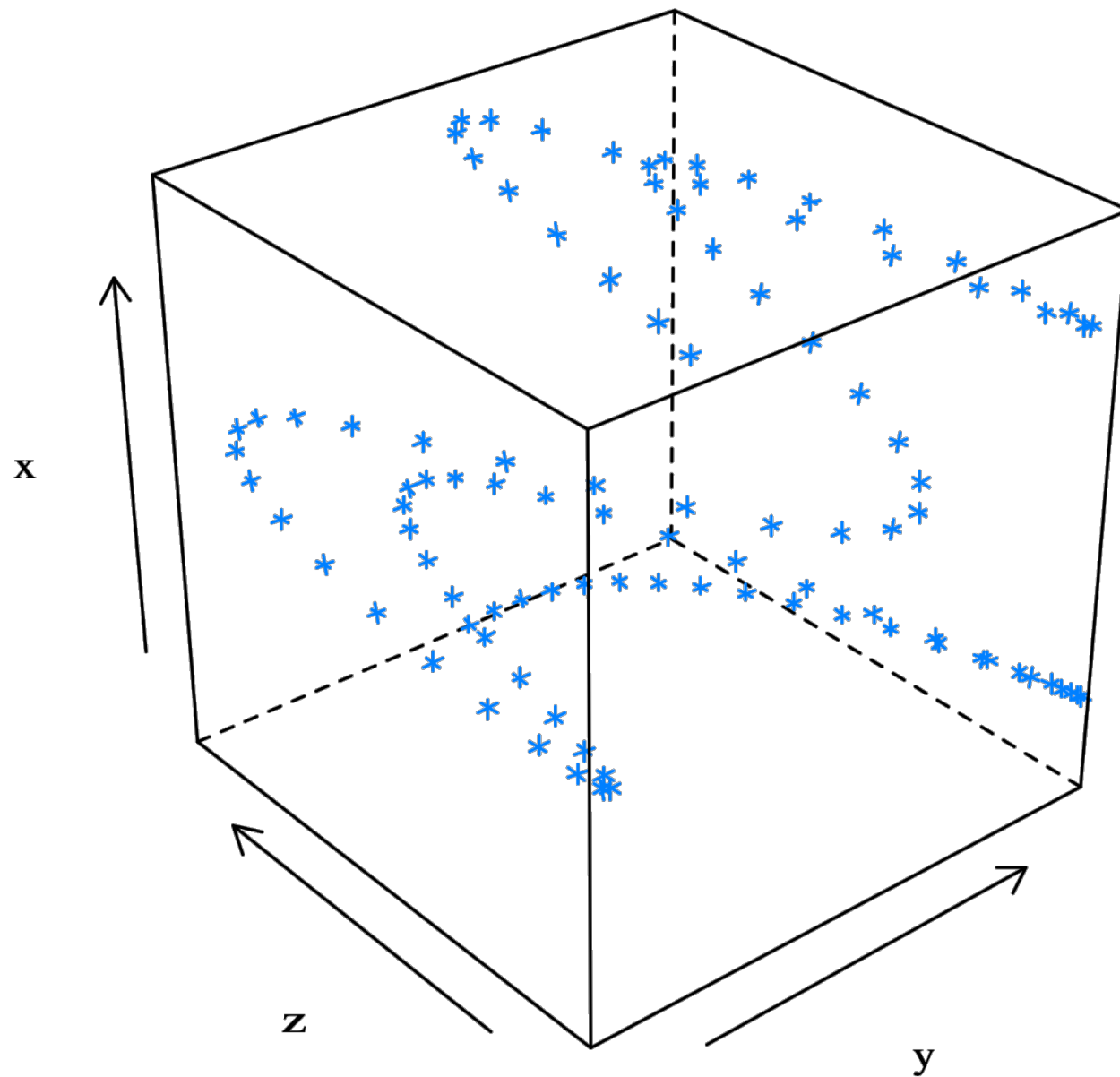
# Java is just another language in this case

```
const express = require('express');
const app = express();

app.get('/', function (req, res) {
  res.send(Interop.eval('R',
    `svg();
    require(lattice);
    x <- 1:100
    y <- sin(x/10)
    z <- cos(x^1.3/(runif(1)*5+10))
    print(cloud(x~y*z, main="cloud plot"))
    grDevices:::svg.off()
  `));
})

app.listen(8080, function () {
  console.log('serving at http://localhost:8080');
});
```

You need to run node  
with the `--jvm` flag



# So what does this have to do with a JIT?

- All the languages use the same JIT, and the same high-level implementation framework, so they can all work together
- Integration is at a higher level than with traditional bytecode implementation





# GraalVM to run native languages

# Is our approach only suited to Java and dynamic languages?

- There's nothing special about native languages
- C has the same `if` statements and `while` loops Ruby does
- C has pointers and `malloc`, but so does Ruby in its FFI module



# Is our approach only suited to Java and dynamic languages?

- Example – running `gzip` on the JVM
  - Not a clean piece of code
  - 8.6 k lines of C
  - Macros, pointer arithmetic, unions
  - We'll avoid the complexity of autotools and make by using a single-file version
  - <http://people.csail.mit.edu/smcc/projects/single-file-programs/>

```

if (more == (unsigned)EOF) {
    /* Very unlikely, but possible on 16 bit machine if strstart == 0
     * and lookahead == 1 (input done one byte at time)
     */
    more--;
} else if (strstart >= WSIZE+MAX_DIST) {
    /* By the IN assertion, the window is not empty so we can't confuse
     * more == 0 with more == 64K on a 16 bit machine.
     */
    Assert(window_size == (ulg)2*WSIZE, "no sliding with BIG_MEM");

    memcpy((char*)window, (char*)window+WSIZE, (unsigned)WSIZE);
    match_start -= WSIZE;
    strstart    -= WSIZE; /* we now have strstart >= MAX_DIST: */
    if (rsync_chunk_end != 0xFFFFFFFFUL)
        rsync_chunk_end -= WSIZE;

    block_start -= (long) WSIZE;

    for (n = 0; n < HASH_SIZE; n++) {
        m = head[n];
        head[n] = (Pos)(m >= WSIZE ? m-WSIZE : NIL);
    }
    for (n = 0; n < WSIZE; n++) {
        m = prev[n];
        prev[n] = (Pos)(m >= WSIZE ? m-WSIZE : NIL);
        /* If n is not on any hash chain, prev[n] is garbage but
         * its value will never be used.
         */
    }
    more += WSIZE;
}

```

```
$ clang -c -emit-llvm gzip.c  
$ gzip small.txt  
$ lli gzip.bc -d small.txt.gz
```

# So what does this have to do with a JIT?

- We can use the JIT that we use for Java, JavaScript, Ruby, Python, R and so on, for C as well
- Actually – any language that can target LLVM
- C, C++, Objective C, Swift, Fortran, Rust, etc
- Genuine potential for dynamic optimization
- Potential for sandboxing as well



# GraalVM to debug polyglot programs

# Tooling for these extra languages

- With other languages on the JVM you usually have to use a Java debugger
- Perhaps with source information in the bytecode for the guest language
- Some custom debuggers, but not for all languages

```
def fizzbuzz(n)
  if n % 3 == 0 && n % 5 == 0
    'FizzBuzz'
  elsif n % 3 == 0
    'Fizz'
  elsif n % 5 == 0
    'Buzz'
  else
    n
  end
end

(1..20).each do |n|
  puts fizzbuzz(n)
end
```



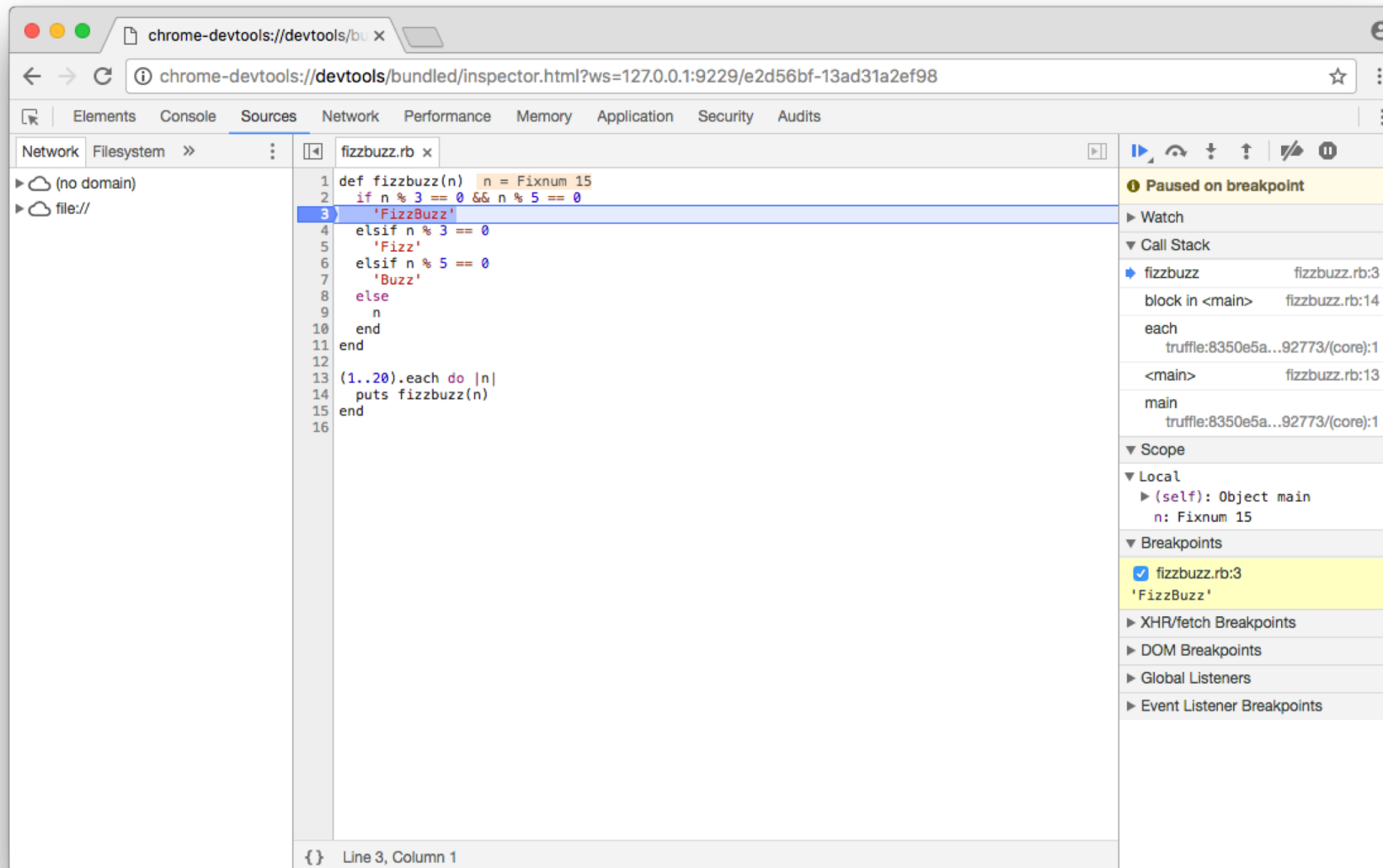
```
$ ruby fizzbuzz.rb
```

```
...
```

```
$ ruby --inspect fizzbuzz.rb
```

Debugger listening on port 9229.

To start debugging, open the following URL in Chrome:  
chrome-devtools://devtools/bundled/...



Taking a heap dump of  
a Java application

Graal VisualVM 20180227-unknown-revn

Applications x

- Local
  - VisualVM
    - TopTen (pid 24809)
      - [heapdump] 09:30:07
  - Remote
  - VM Coredumps
  - Snapshots

TopTen (pid 24809) x

Threads Sampler Profiler [heapdump] 09:30:07 x

### TopTen (pid 24809)

#### Heap Dump

Summary

Heap		Environment	
Size:	17,428,828 B	System	Mac OS X (10.13.4)
Classes:	5,555	Architecture:	x86_64 64bit
Instances:	303,497	Java Home:	0.0-rc1/Contents/Home/jre
Classloaders:	90	Java Version:	1.8.0_161
GC Roots:	1,797	Java Name:	al-jvmci-0.42, mixed mode)
Objects Pending for Finaliz	0	Java Vendor:	Oracle Corporation
		JVM Uptime:	0 min 10 sec

#### System Properties [ show ]

Classes by Instances Count [ view all ]			Classes by Instances Size [ view all ]		
char[]	56,768	(18.7%)	char[]	4,228,030 B	(24.3%)
java.lang.String	56,533	(18.6%)	java.lang.String	1,582,924 B	(9.1%)
java.lang.Object[]	18,461	(6.1%)	java.lang.Object	1,317,456 B	(7.6%)
java.util.HashMap	15,843	(5.2%)	java.lang.reflect	1,148,436 B	(6.6%)
java.lang.Class[]	11,871	(3.9%)	java.util.HashM	697,092 B	(4%)

Instances by Size [ view all ]			Dominators by Retained Size [ view all ]		
java.util.HashMa	131,096 B	(0.8%)	Retained sizes must be computed first:		
java.util.concurre	65,560 B	(0.4%)	<a href="#">Compute Retained Sizes</a>		

# Monitoring for these extra languages

- VisualVM and other similar tools let you monitor the JVM
- Non-JVM languages often don't have this kind of tool
- With other languages on the JVM, then often show the underlying Java objects, rather than the guest language objects

```
$ ruby render.rb
```

```
$ jvisualvm
```

Change the format of  
the heap dump here  
to show it as Ruby

Graal VisualVM 20180227-unknown-revn

Applications x Ruby (pid 24813) x

Local  
VisualVM  
Remote  
VM Coredumps  
Snapshots

Threads Sampler Profiler [heapdump] 09:30:30 x

### Ruby (pid 24813)

Heap Dump

Summary

Heap		Environment	
Size:	766,488 B	Language:	Ruby (version 2.3.7)
Types:	44	Platform:	darwin x86_64
Objects:	6,936		

Types by Objects Count [ view all ]			Types by Objects Size [ view all ]		
String	3,733	(0.3%)	String	358,368 B	(0.5%)
Symbol	1,110	(0.1%)	Symbol	106,560 B	(0.2%)
Class	834	(0.1%)	Class	80,344 B	(0.1%)
Array	481	(0%)	Array	73,624 B	(0.1%)
Proc	201	(0%)	Hash	55,392 B	(0.1%)

Objects by Size [ view all ]			Dominators by Retained Size [ view all ]	
Hash#5552 : sha	16,848 B	(0%)	Retained sizes must be computed first: <button>Compute Retained Sizes</button>	
Hash#6538 : sha	9,232 B	(0%)		
Hash#1388 : sha	4,368 B	(0%)		
Hash#4991 : sha	4,368 B	(0%)		
Hash#1916 : sha	2,320 B	(0%)		

# So what does this have to do with a JIT?

- All the languages are implemented in the same framework, so the debugger can understand them all via that framework
- The Graal JIT has support for deoptimization, so can debug optimized code running in production



# GraalVM to run Java as a native library




# Java code as a native library

- The Java ecosystem is phenomenal
- Often more and better libraries than available in other languages
- In the examples so far, it's always been the Java code that has owned the process
- Can we run Java code inside another application that we already have?

Apache SIS™

About▼Project Documentation▼ASF▼

APACHECON  
North America  
September 24-27, 2018  
Montréal, Canada



The Apache SIS™ library

Apache Spatial Information System (SIS) is a free software, Java language library for developing geospatial applications. SIS provides data structures for geographic features and associated metadata along with methods to manipulate those data structures. The library is an implementation of [GeoAPI 3.0](#) interfaces and can be used for desktop or server applications.

The SIS metadata module forms the base of the library and enables the creation of metadata objects which comply with the model of international standards. The SIS referencing module enable the construction of geodetic data structures for geospatial referencing such as axis, projection and coordinate reference system definitions, along with the associated operations which enable the conversion or transformation of coordinates between different systems of reference. The SIS storage modules will provide a common approach to the reading and writing of metadata, features and coverages.

Some Apache SIS features are:

- Geographic metadata (ISO 19115-1:2014)
  - Read from or written to ISO 19139 compliant XML documents.
  - Read from netCDF, GeoTIFF, Landsat, GPX and Moving Feature CSV encoding.
  - Automatic conversions between the model published in 2003 and the revision published in 2014.
- Referencing by coordinates (ISO 19111:2007)
  - Well Known Text (WKT) version 1 and 2 (ISO 19162:2015).
  - Geographic Markup Language (GML) version 3.2 (ISO 19136:2007).
  - [EPSG geodetic dataset](#) for geodetic definitions and for coordinate operations. See the list of [supported coordinate reference systems](#).
  - Mercator, Transverse Mercator, Lambert Conic Conformal, stereographic and more map projections. See the list of [supported operation methods](#).
  - Optional [bridge to Proj.4](#) as a complement to Apache SIS own referencing engine.
- Referencing by identifiers (ISO 19112:2003)
  - Geohashes (a simple encoding of geographic coordinates into short strings of letters and digits).
  - Military Grid Reference System (MGRS), also used for some civilian uses.
- Units of measurement
  - Implementation of [JSR-363](#) with parsing, formating and unit conversion functionalities.

Home

License

Mailing Lists

Project Team

PROJECT

DOCUMENTATION

Developer guide

Online Javadoc

Downloads

Source Code

Code patterns

FAQ

Issue Tracker

ASF

The Foundation

Donate

Thanks

Security

ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved. |

```
import org.apache.sis.distance.DistanceUtils;

public class Distance {

    public static void main(String[] args) {
        final double aLat    = Double.parseDouble(args[0]);
        final double aLong    = Double.parseDouble(args[1]);
        final double bLat     = Double.parseDouble(args[2]);
        final double bLong    = Double.parseDouble(args[3]);
        System.out.printf("%f km%n", DistanceUtils.getHaversineDistance(aLat, aLong, bLat, bLong));
    }
}
```

```
$ javac -cp sis.jar -parameters Distance.java
$ java -cp sis.jar:. Distance \
    51.507222 -0.1275 40.7127 -74.0059
5570.25 km
```

```
$ native-image -cp sis.jar:. Distance
```

```
...
```

```
$ ./distance 51.507222 -0.1275 40.7127 -74.0059  
5570.25 km
```

```
...
import org.graalvm.nativeimage.IsolateThread;
import org.graalvm.nativeimage.c.function.CEntryPoint;

public class Distance {

    ...

    @CEntryPoint(name = "distance")
    public static double distance(IsolateThread thread,
        double a_lat, double a_long,
        double b_lat, double b_long) {
        return DistanceUtils.getHaversineDistance(a_lat, a_long, b_lat, b_long);
    }

    ...
}
```

```
$ native-image -cp sis.jar:. -H:Kind=SHARED_LIBRARY \  
-H:Name=libdistance
```

```
#include <stdlib.h>
#include <stdio.h>

#include <libdistance.h>

int main(int argc, char **argv) {
    graal_isolate_t *isolate = NULL;
    graal_isolatethread_t *thread = NULL;

    if (graal_create_isolate(NULL, &isolate) != 0 || (thread = graal_current_thread(isolate)) == NULL) {
        fprintf(stderr, "initialization error\n");
        return 1;
    }

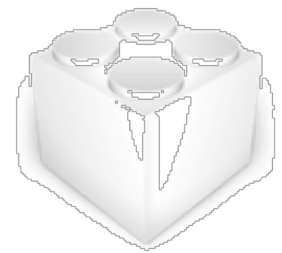
    double a_lat   = strtod(argv[1], NULL);
    double a_long  = strtod(argv[2], NULL);
    double b_lat   = strtod(argv[3], NULL);
    double b_long  = strtod(argv[4], NULL);

    printf("%f km\n", distance(thread, a_lat, a_long, b_lat, b_long));

    return 0;
}
```



```
$ clang -I. -L. -ldistance distance.c -o distance
$ otool -L distance
distance:
    libdistance.dylib
    /usr/lib/libSystem.B.dylib
$ ./distance 51.507222 -0.1275 40.7127 -74.0059
5570.25 km
```



**libdistance.dylib**

# GraalVM for polyglot in the database

# Demo using the Oracle Database MLE

- Multi-lingual (polyglot) edition
- Available as a Docker image
- Subject to the Oracle Technology Network license agreement, so you need to accept that and download it yourself

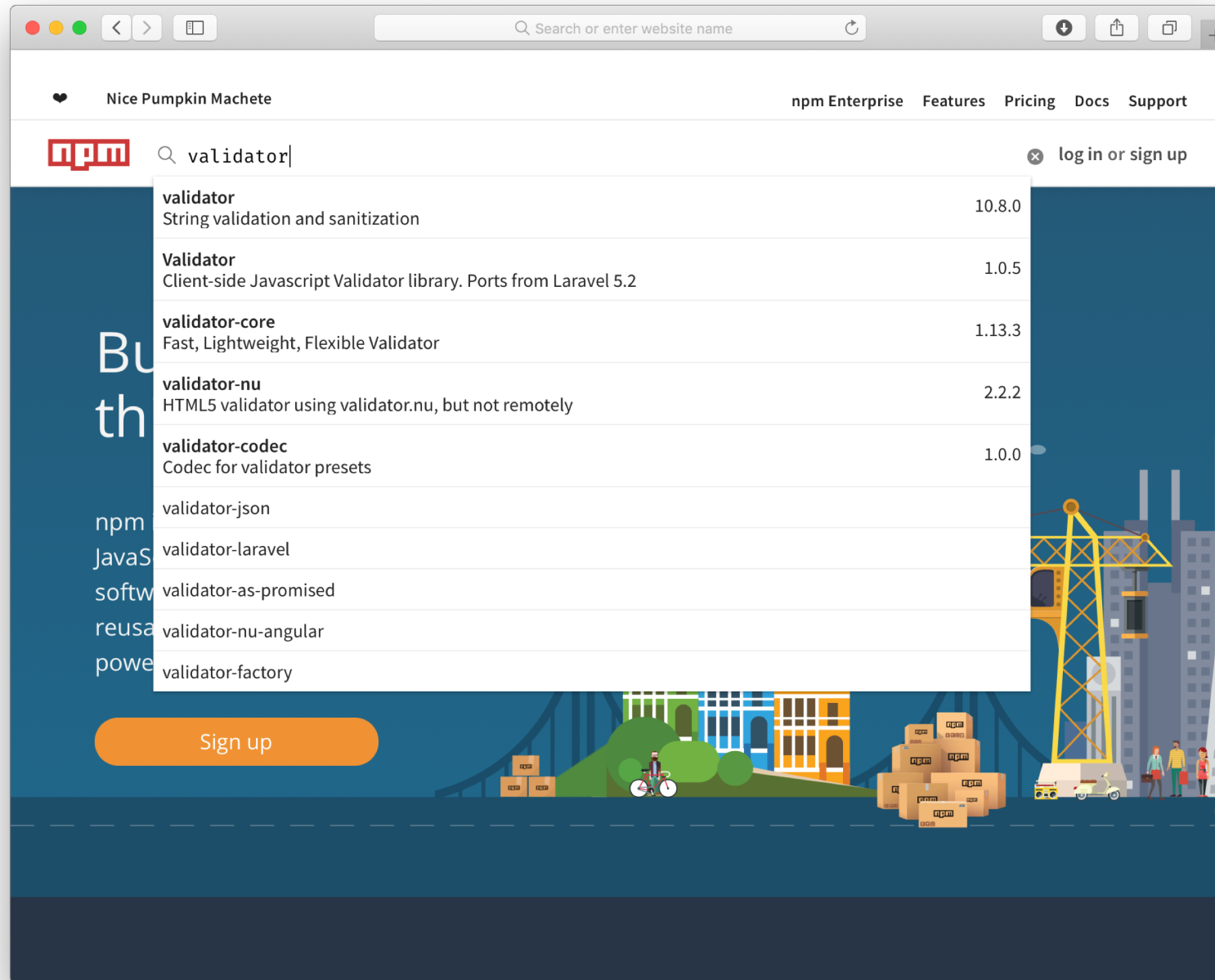
<https://oracle.github.io/oracle-db-mle/releases/0.2.7/docker/>

```
$ docker load --input mle-docker-0.2.7.tar.gz # takes a while  
  
$ docker run mle-docker-0.2.7 # takes a while  
$ docker ps  
$ docker exec -ti <container_id> bash -li
```

# JavaScript in the client and frontend, Oracle in the backend



**ORACLE®**  
DATABASE



validator.js

npm

v10.8.0

build

passing

downloads

6M/m

A library of string validators and sanitizers.

Strings only

This library validates and sanitizes strings only.

If you're not sure if your input is a string, coerce it using `input + ''`. Passing anything other than a string is an error.

Installation and Usage

Server-side usage

Install the library with `npm install validator`

No ES6

```
var validator = require('validator');  
  
validator.isEmail('foo@bar.com'); //=> true
```

ES6



```
import validator from 'validator';
```

install

```
> npm i validator
```

± weekly downloads

827,075

version	license
10.8.0	MIT
open issues	pull requests
31	2
homepage	repository
github.com	 github
last publish	
a month ago	
collaborators	
	
<div>Test with RunKit</div>	
<div>Report a vulnerability</div>	

ORACLE®

Copyright © 2018, Oracle and/or its affiliates. All rights reserved. |

```
$ echo "{}" > package.json
$ npm install validator
$ npm install @types/validator

$ dbjs deploy -u scott -p tiger -c localhost:1521/ORCLCDB validator

$ sqlplus scott/tiger@localhost:1521/ORCLCDB
```

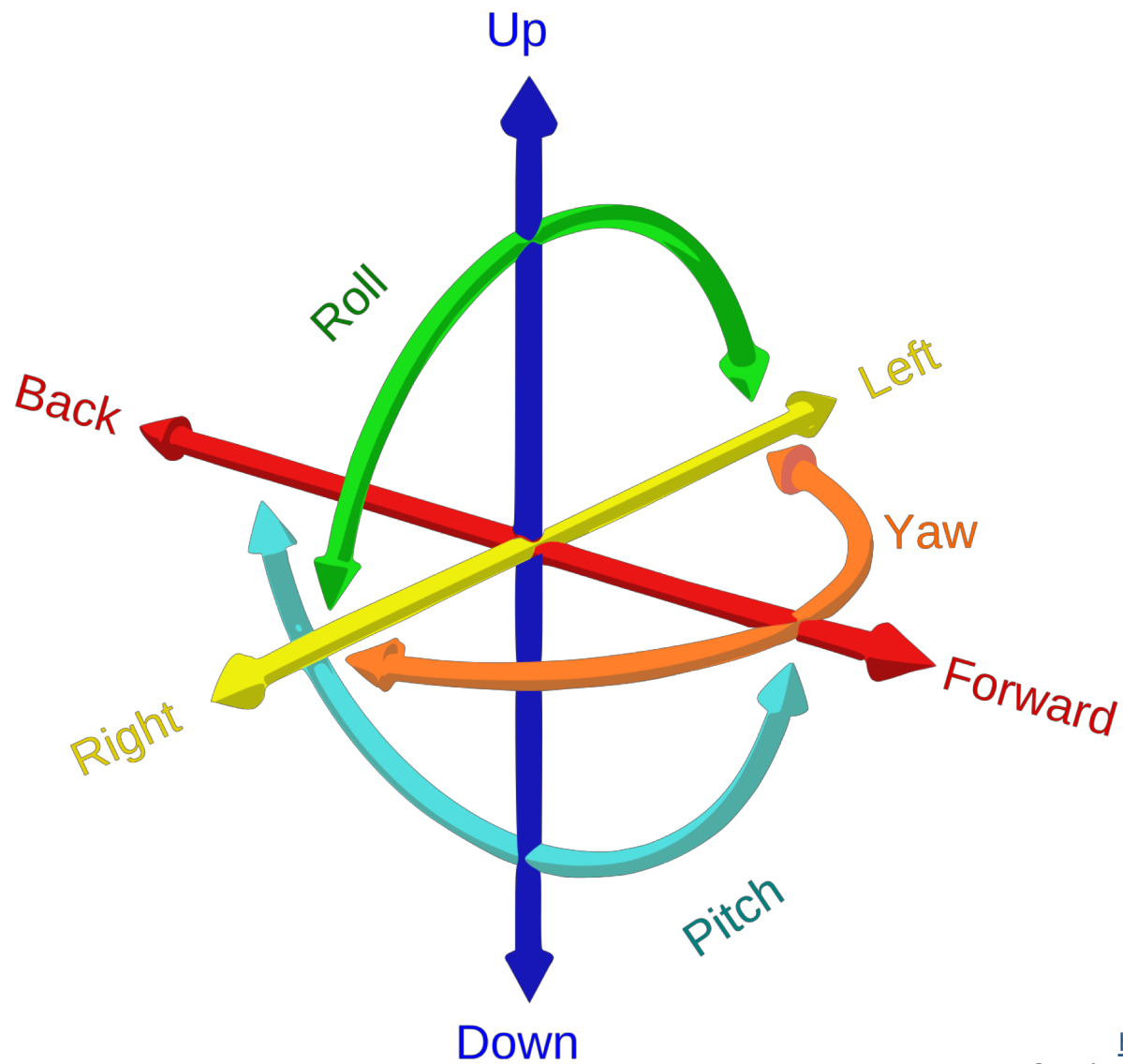


```
SQL> select validator.isEmail('oleg.selaev@oracle.com') from dual;  
SQL> select validator.isEmail('oleg.selaev') from dual;
```

# How does this bring it all together?

- This is a...
  - JavaScript interpreter, implemented in Java, using our framework
  - Using the polyglot interface to talk to the query language
  - Compiled ahead-of-time using Graal into a native library
  - Which can be linked into the database
  - Including Graal as a JIT within that library for high performance

# Wrap up



<https://commons.wikimedia.org/wiki/File:6DOF.svg>  
Creative Commons Attribution-Share Alike 4.0 International

# I think it's about giving people degrees of freedom

- Let people run the language they want
- With the ecosystem of libraries they want
- On the JVM or on native
- Embedded or embedding
- With the tooling they want
- With the performance they want
- 'One compiler to rule them all'

# Get in touch with us

[graalvm.org](http://graalvm.org)

[twitter.com/ChrisGSeaton](https://twitter.com/ChrisGSeaton)

[gitter.im/graalvm/graal-core](https://gitter.im/graalvm/graal-core)

# Team

## Oracle

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

## Oracle (continued)

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

## Oracle (continued)

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

## Red Hat

Andrew Dinn  
Andrew Haley

## Intel

Michael Berg

## Twitter

Chris Thalinger

## Oracle Interns

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

## Oracle Alumni

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

## JKU Linz

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

## TU Berlin:

Volker Markl  
Andreas Kunft  
Jens Meiners  
Tilman Rabl

## University of Edinburgh

Christophe Dubach  
Juan José Fumero Alfonso  
Ranjeet Singh  
Toomas Remmelg

## LaBRI

Floréal Morandat

## University of California, Irvine

Michael Franz  
Yeoul Na  
Mohaned Qunaibit  
Gulfem Savrun Yeniceri  
Wei Zhang

## Purdue University

Jan Vitek  
Tomas Kalibera  
Petr Maj  
Lei Zhao

## T. U. Dortmund

Peter Marwedel  
Helena Kotthaus  
Ingo Korb

## University of California, Davis

Duncan Temple Lang  
Nicholas Ulle

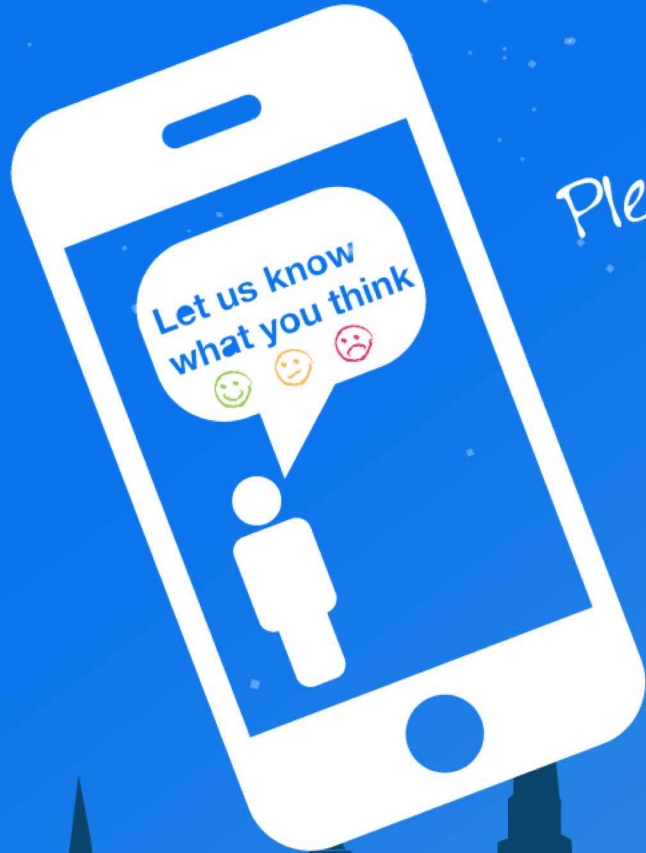
## University of Lugano, Switzerland

Walter Binder  
Sun Haiyang

## 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.





*Please*

**Remember to  
rate this session**

*Thank you!*



Did you **remember**  
**to rate** the previous  
session ?



**goto;**  
copenhagen

 Follow us @gotocph

