

How and why
GraalVM is
quickly becoming
relevant for you

GraalVM™



ACEs@home – 17th June 2020

Lucas Jellema, CTO & Architect AMIS, Oracle ACE Director & Groundbreaker Ambassador





Demo

Demo

bit.ly/graalvm-handson

Native Binary
Executables

Lean Run Time

Custom DSL

Serverless Java

JavaScript, Python, Ruby, R, C/C++

Next Gen JIT Compiler

Polyglot

Oh boy, we will be in so much trouble...



What is GraalVM?



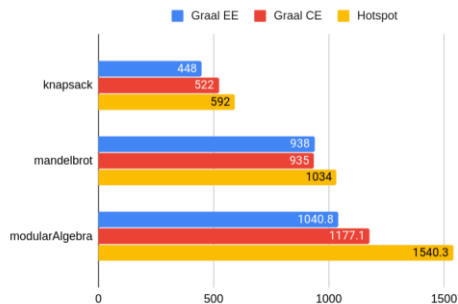
Graal Compiler

JVM Compiler Interface (JVMCI) JEP 243

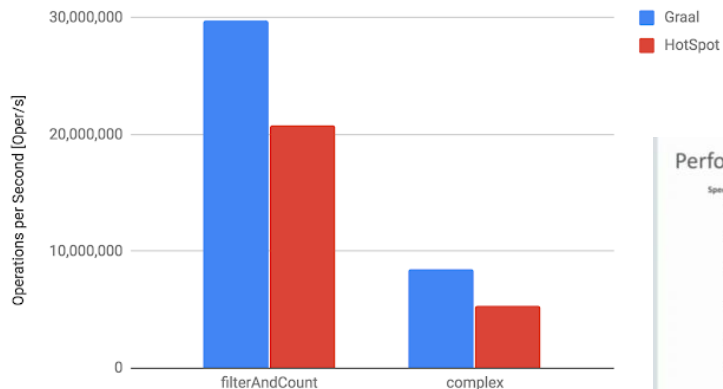
Java HotSpot VM

Performance and Resource Consumption of GraalVM vs C2

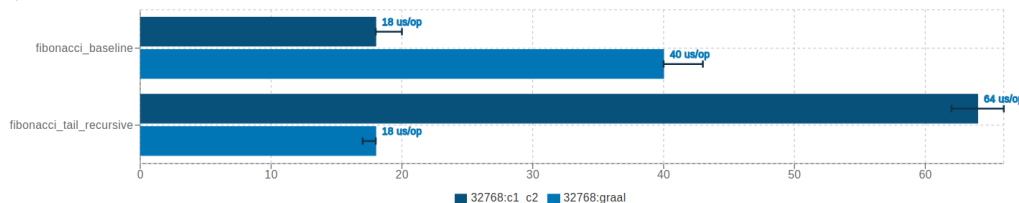
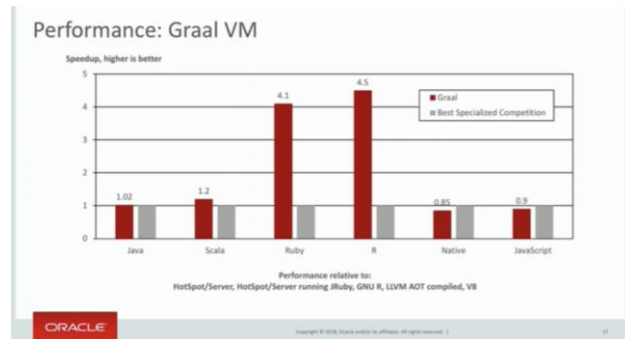
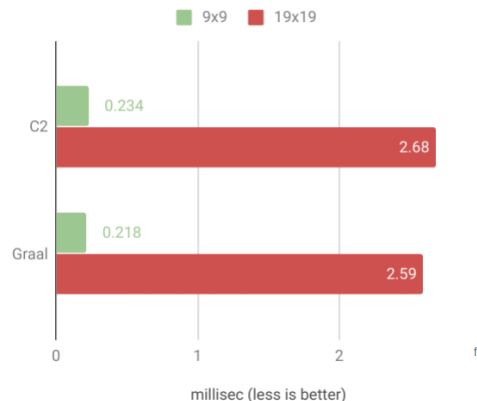
Kotlin Compilers (ms. lower is better) - Continuous Run



Stream Performance: Graal vs. HotSpot (Higher is Better)



Game of Go



What is GraalVM - beyond a better JIT compiler?

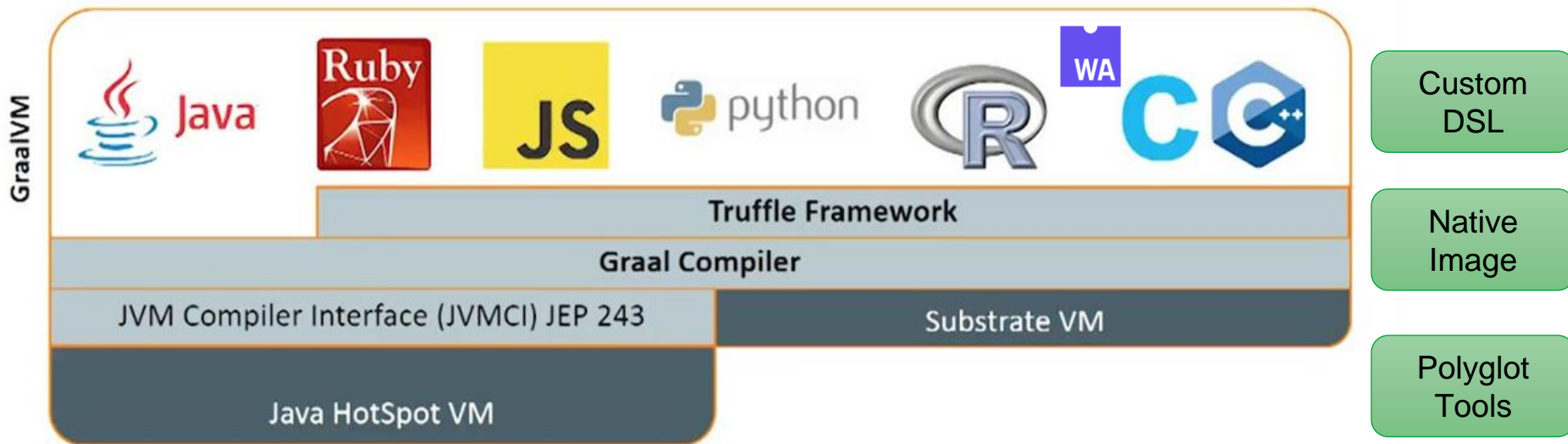


Graal Compiler

JVM Compiler Interface (JVMCI) JEP 243

Java HotSpot VM

What is GraalVM - beyond a better JIT compiler?



Introducing

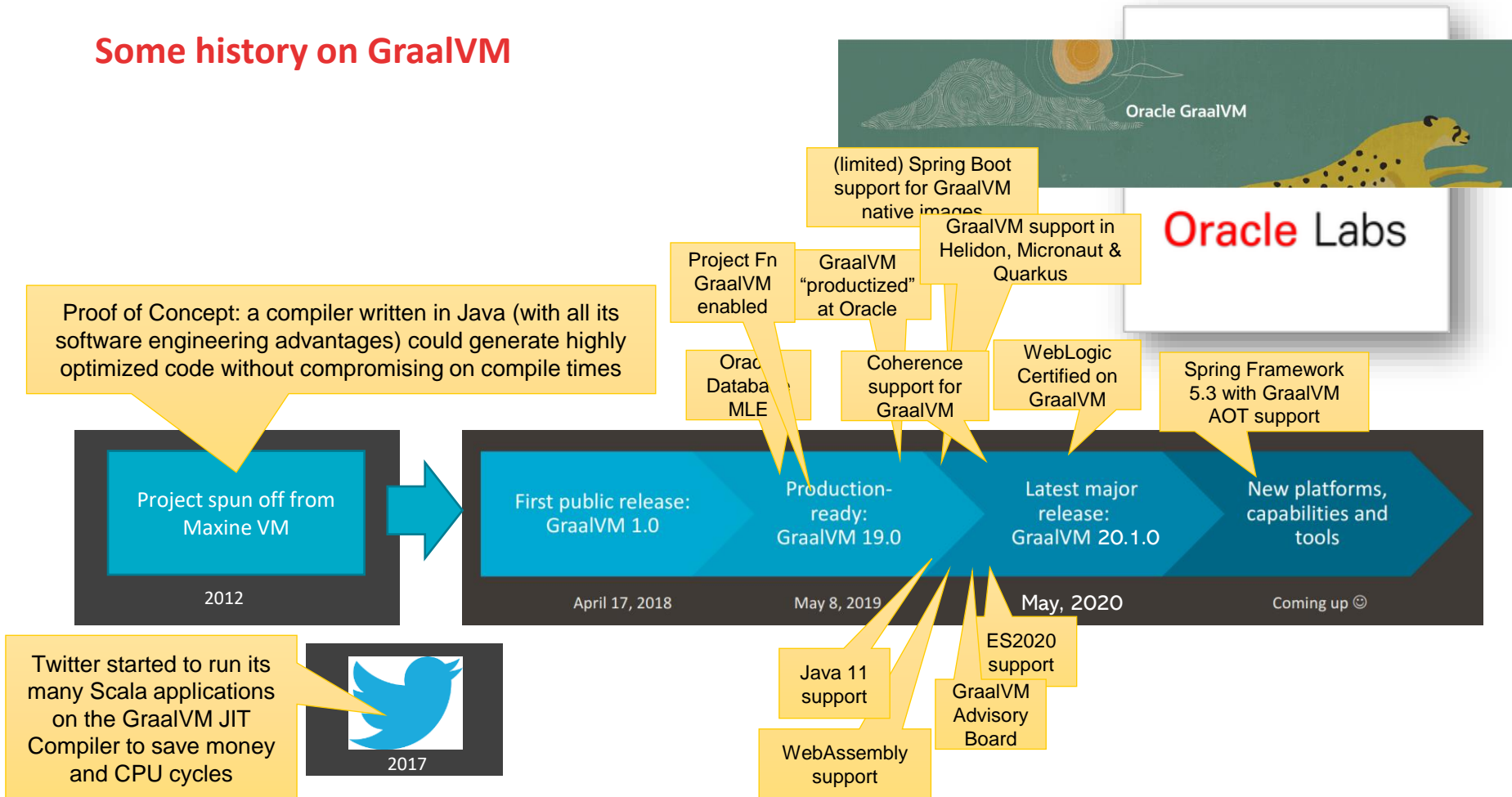
- JIT Compiler for running Java applications faster (matching native languages for speed)
 - Embeddable on Hotspot VM or other JVMs
- Polyglot VM – for running applications in any language
 - JavaScript, Ruby, R, Python, C/C++
- Ahead-of-Time compiler for creating native binaries
 - Simple, lean runtime with fast startup and minimal memory and process footprint
- Interoperability platform that allows applications in one technology to interact with modules in other technologies (“Nashorn on super steroids”)
- Framework for compiling, running, interoperating and natively compiling applications written in your own DSL
- Set of cross-language programming tools

GraalVM™

Run Programs Faster Anywhere

- ✓ Increase application throughput and reduce latency
- ✓ Compile applications into small self-contained native binaries
- ✓ Seamlessly use multiple languages and libraries

Some history on GraalVM



GraalVM: Two Editions

Oracle Labs

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](#).

DOWNLOAD FROM GITHUB

LICENSE

- [Open Source Licenses](#)
- Free for development and production use

BENEFITS

- Open-source license
- Free community support via [public channels](#)
- Presence of all enterprise components
- Bug fixes and enhancements

AVAILABLE FOR



macOS



Linux



Windows

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](#).

DOWNLOAD FROM OTN

LICENSE

- [Oracle Master License Agreement](#)
- Free for evaluation and non-production use
- [Contact us](#) for commercial use and support options

BENEFITS

- Faster performance and smaller footprint
- Enhanced security features
- Managed capabilities for native code
- Premier 24x7x365 support via [MOS](#)

AVAILABLE FOR



macOS

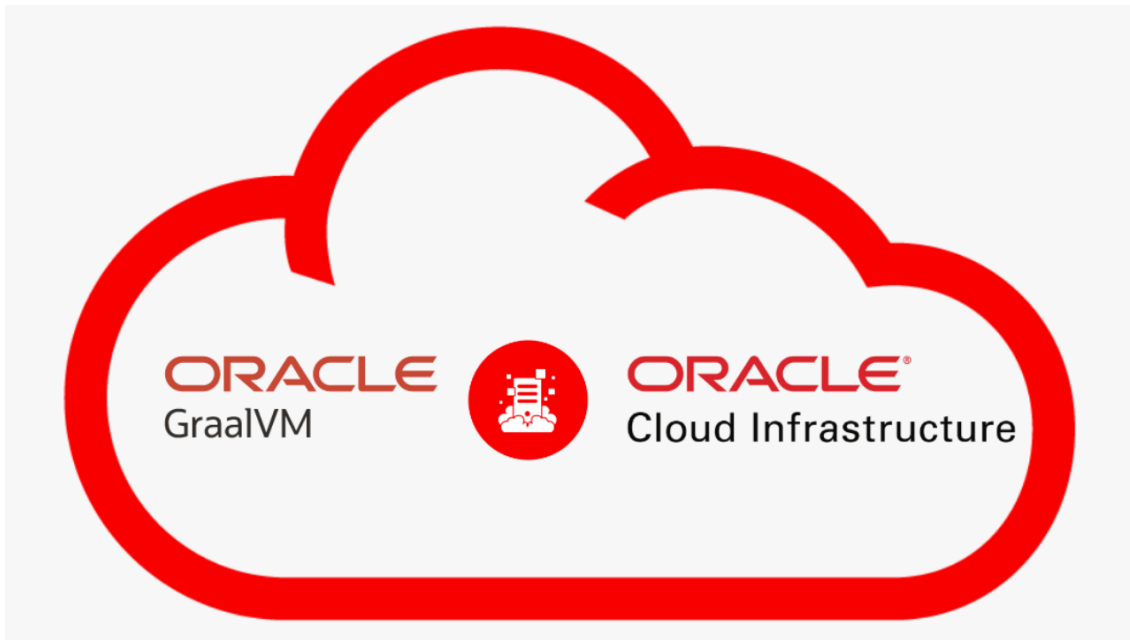


Linux



Windows

GraalVM EE included in all OCI subscriptions



GraalVM Enterprise is included in all Oracle Cloud Infrastructure (OCI) subscriptions so you can use it for no additional charge (Compute, OKE, Functions)

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](#).

[DOWNLOAD FROM OTN](#)

LICENSE

- [Oracle Master License Agreement](#)
- Free for evaluation and non-production use
- [Contact us](#) for commercial use and support options

BENEFITS

- Faster performance and smaller footprint
- Enhanced security features
- Managed capabilities for native code
- Premier 24x7x365 support via [MOS](#)

AVAILABLE FOR



macOS



Linux



Windows

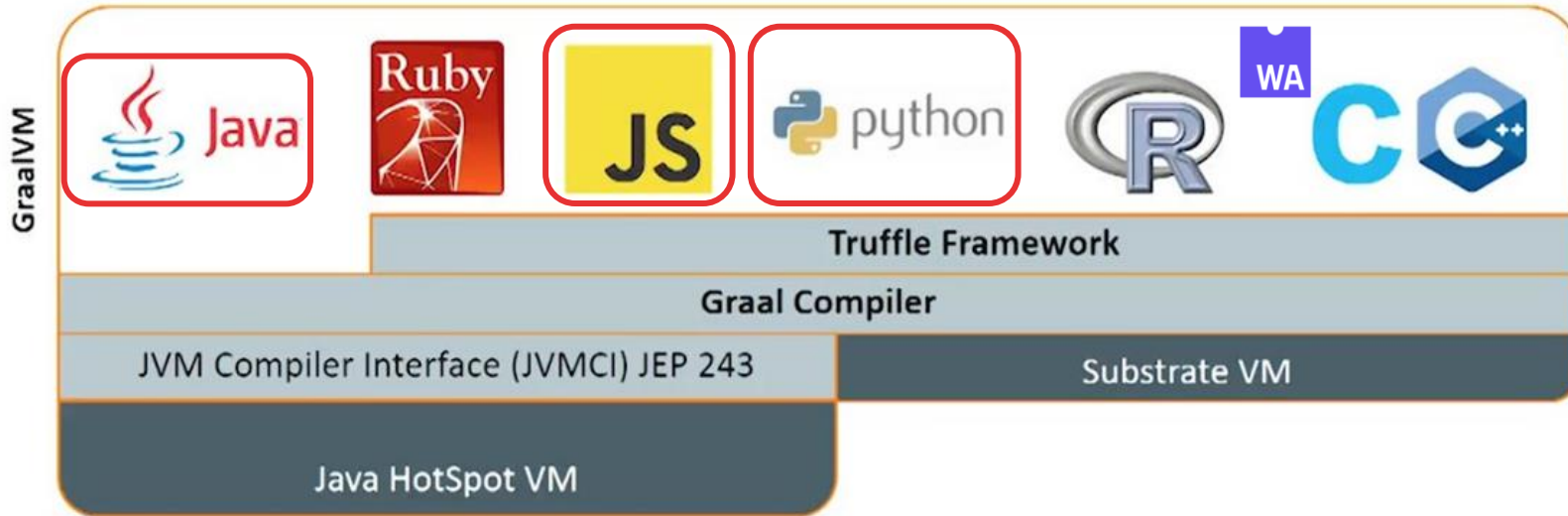
Today, we will focus on

- JIT Compiler for running Java applications faster (matching native languages)
 - on Hotspot VM or other traditional VMs
- **Polyglot VM – for running applications in any language**
 - JavaScript, Ruby, R, Python, C/C++
- **Ahead-of-Time compiler for creating native binaries**
 - **Simple, lean runtime with fast startup and minimal memory and process footprint**
- **Interoperability platform that allows applications in one technology to interact with with modules in other technologies (“Nashorn on super steroids”)**
- Framework for compiling, running, interoperating and natively compiling applications written in your own DSL
- Set of cross-language programming tools

GraalVM: Polyglot Runtime



Polyglot Run Time



Why do we have so many languages?

AMIS | CONCLUSION



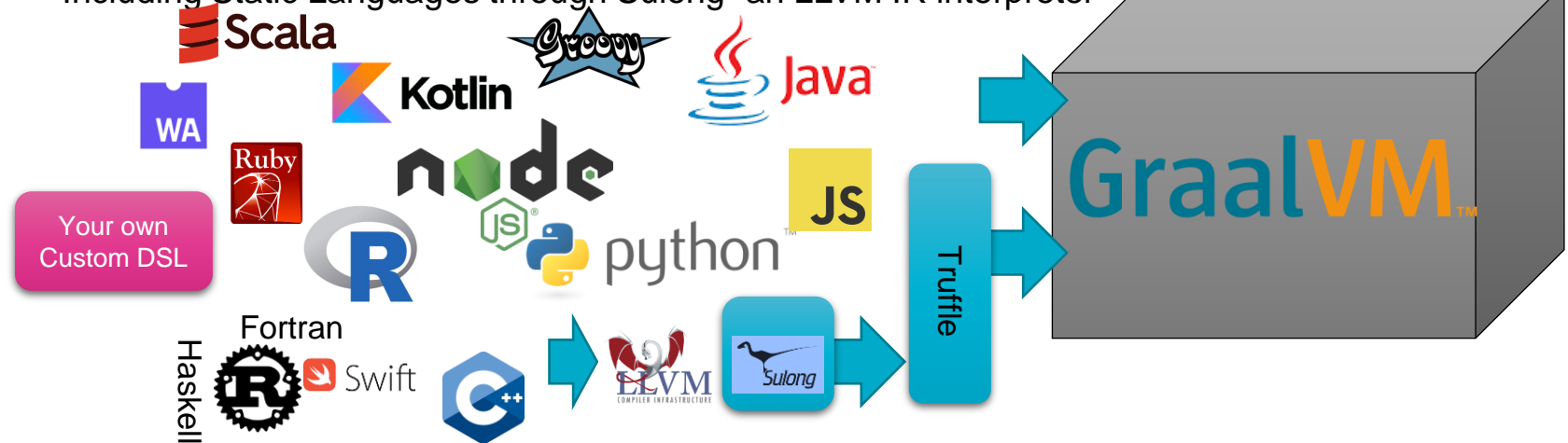
Polyglots – speak and understand many languages

AI | CONCLUSION



GraalVM Polyglot Runtime

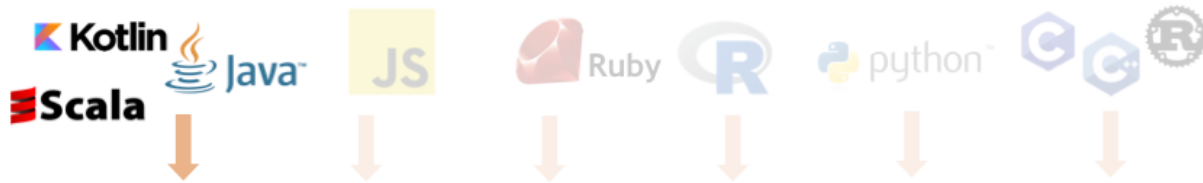
- GraalVM can run (on the mature, manageable, observable and run time optimized JVM)
 - Java
 - JVM Languages
 - Many other languages
 - Through Truffle produced intermediate code
 - Including Static Languages through Sulong - an LLVM IR interpreter



GraalVM: Ahead-of-Time compiler



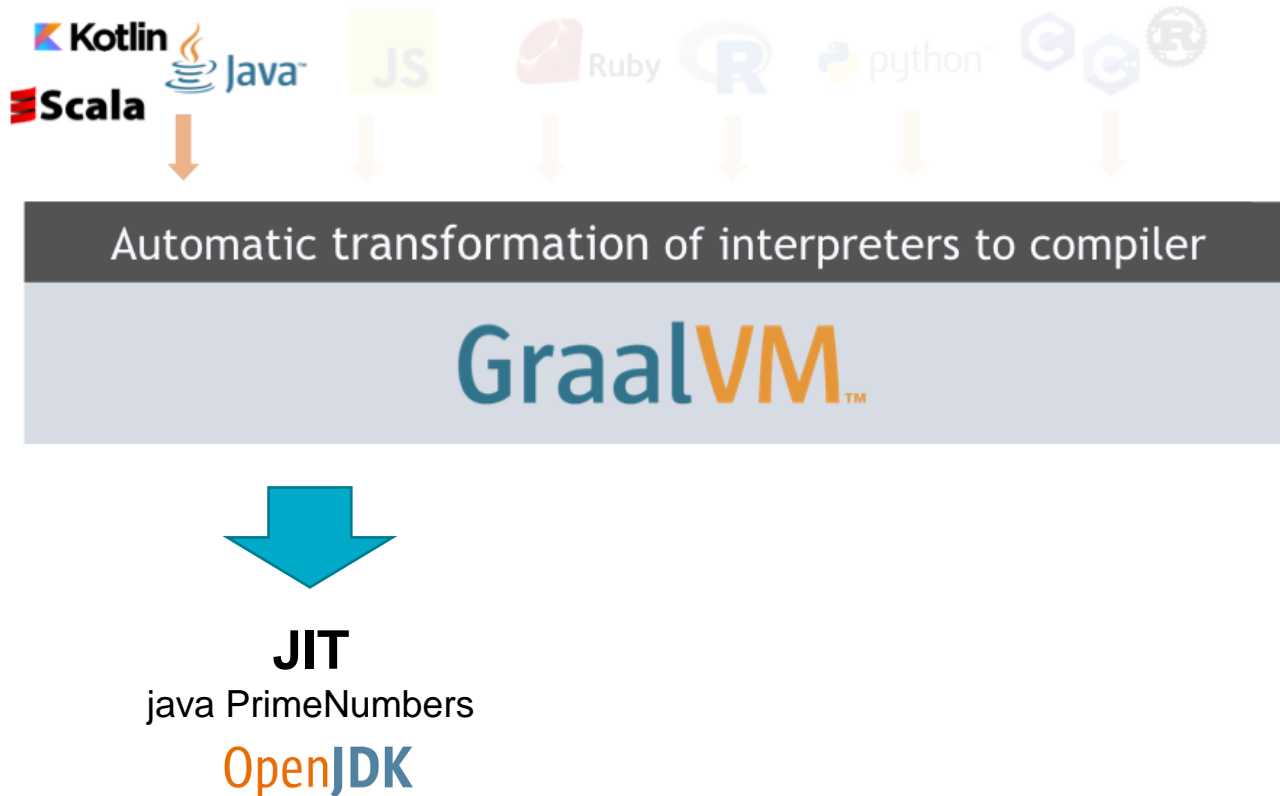
GraalVM compile modes



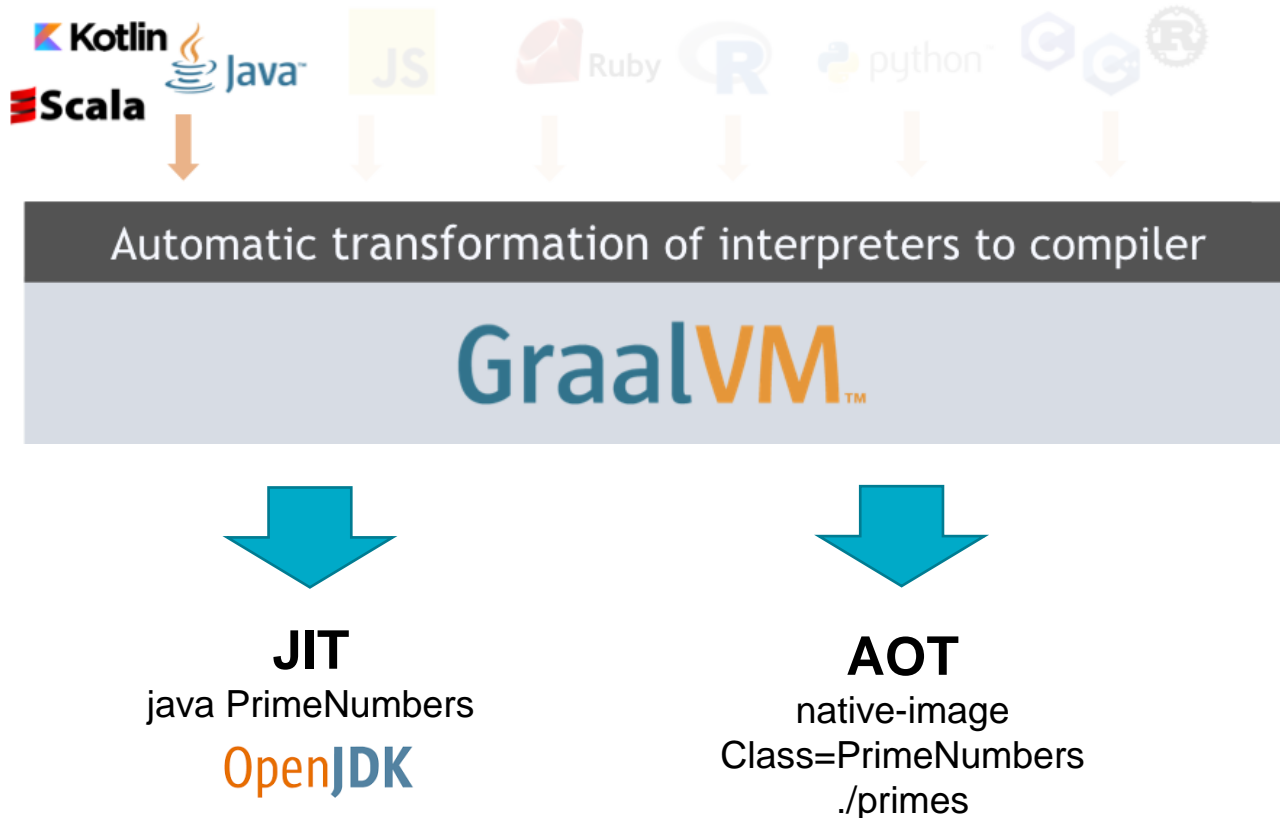
Automatic transformation of interpreters to compiler

GraalVMTM

GraalVM compile modes



GraalVM compile modes




```
bash-4.2# $GRAALVM_HOME/bin/native-image --verbose -H:Name=primes -H:Class=PrimeNumbers
```

```
StartServer [
/opt/graalvm-ce-java8-20.1.0/jre/bin/java \
-Xbootclasspath/a:/opt/graalvm-ce-java8-20.1.0/jre/lib/boot/
ne.jar:/opt/graalvm-ce-java8-20.1.0/jre/lib/boot/truffleruby
-cp \
/opt/graalvm-ce-java8-20.1.0/jre/lib/svm/builder/pointsto.je]
shadowed.jar:/opt/graalvm-ce-java8-20.1.0/jre/lib/svm/builder/
.jar:/opt/graalvm-ce-java8-20.1.0/jre/lib/svm/builder/javacp
pt/graalvm-ce-java8-20.1.0/jre/lib/svm/builder/llvm-wrapper-
ar:/opt/graalvm-ce-java8-20.1.0/jre/lib/jvmci/graal.jar:/opt
ava8-20.1.0/jre/lib/jvmci/graal-truffle-jfr-impl.jar:/opt/gr
.1.0/jre/lib/resources.jar \
-XX:+UseParallelGC \
-XX:+UnlockExperimentalVMOptions \
```

```
-H:Name=primes
```

```
-H:Class=PrimeNumbers
```

```
-H:CLibraryPath=/opt/graalvm-ce-java8-20.1.0/jre/lib/svm/clibraries/linux-amd64
```

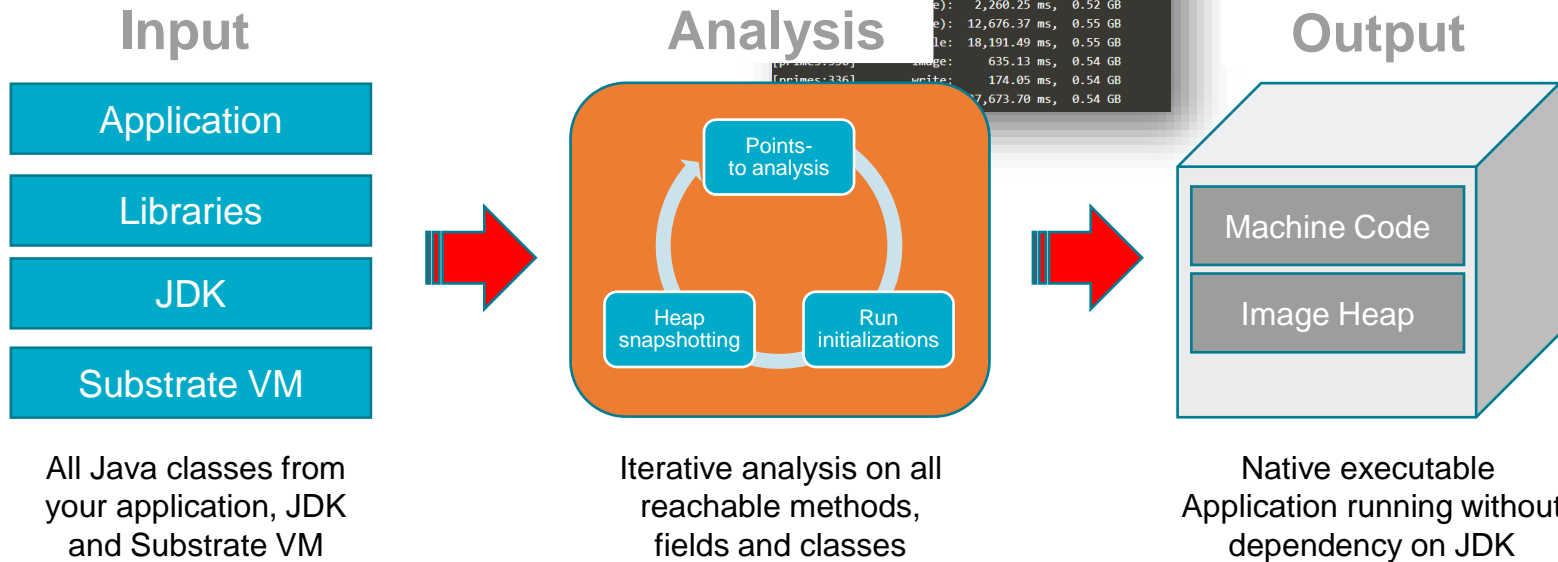
```
[primes:336] classlist: 3,228.97 ms, 0.56 GB
[primes:336] (cap): 796.80 ms, 0.56 GB
[primes:336] setup: 3,020.37 ms, 0.56 GB
[primes:336] (clinit): 182.89 ms, 0.54 GB
[primes:336] (typeflow): 6,634.87 ms, 0.54 GB
[primes:336] (objects): 4,350.83 ms, 0.54 GB
[primes:336] (features): 190.84 ms, 0.54 GB
[primes:336] analysis: 11,587.30 ms, 0.54 GB
[primes:336] universe: 292.02 ms, 0.54 GB
[primes:336] (parse): 2,865.28 ms, 0.55 GB
[primes:336] (inline): 2,260.25 ms, 0.52 GB
[primes:336] (compile): 12,676.37 ms, 0.55 GB
[primes:336] compile: 18,191.49 ms, 0.55 GB
[primes:336] image: 635.13 ms, 0.54 GB
[primes:336] write: 174.05 ms, 0.54 GB
[primes:336] [total]: 37,673.70 ms, 0.54 GB
```

```
bash-4.2# ls -l
```

```
-rw-rw-rw- 1 root root 613 Jun 17 07:26 PrimeNumbers.java
```

```
-rwxr-xr-x 1 root root 3403272 Jun 17 07:35 primes
```


Ahead-of-Time Compilation



AOT vs JIT: Startup Time

JIT

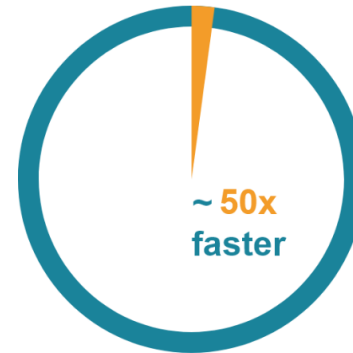
- Load JVM executable
- Load the classes from the file system
- Verify bytecode
- Start interpreting
- Run static initializers
- First tier compilation (C1)
- Run (suboptimal)
- Gather profiling feedback
- Second tier compilation (GraalVM or C2)
- Run with best machine code

AOT

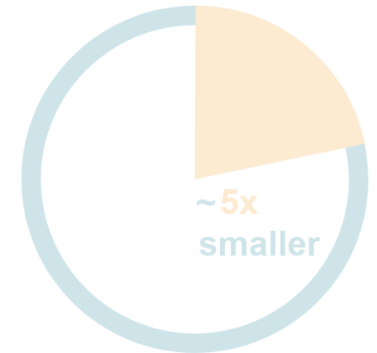
- Load executable with prepared heap
- Immediately start with best machine code

GraalVM for Microservices

Startup Time



Memory Footprint



AOT vs JIT: Memory Footprint

JIT

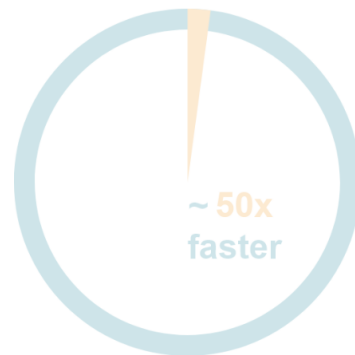
- Loaded JVM executable
- Application data
- Loaded bytecodes
- Reflection meta-data
- Code cache
- Profiling data
- JIT compiler data structures

AOT

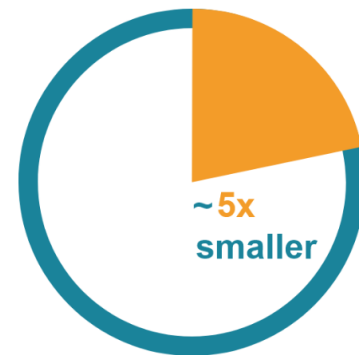
- Loaded application executable
- Application data

GraalVM for Microservices

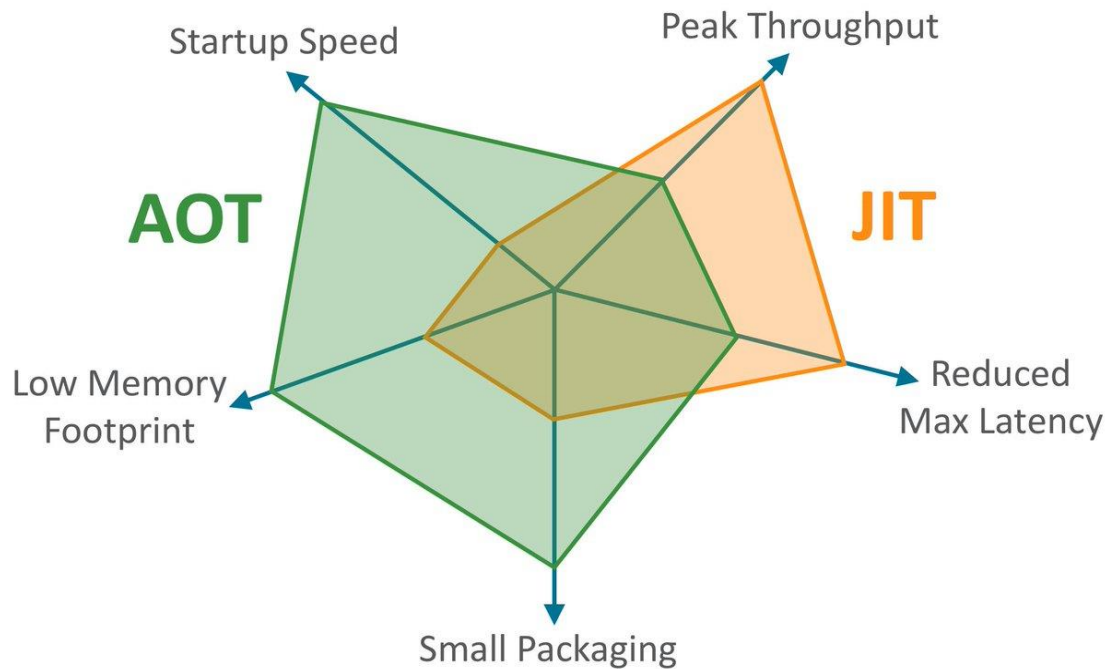
Startup Time



Memory Footprint



AOT vs JIT

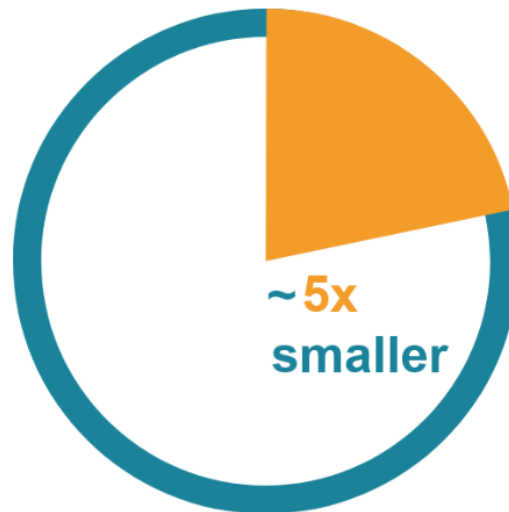


Microservices – Serverless – Containers – Native (AOT)

Startup Time



Memory Footprint



Frameworks adopting GraalVM native image

Helidon SE (Oracle)

A set of Java Libraries for writing microservices



Micronaut

JVM-based framework for building light-weight modular applications (microservices)

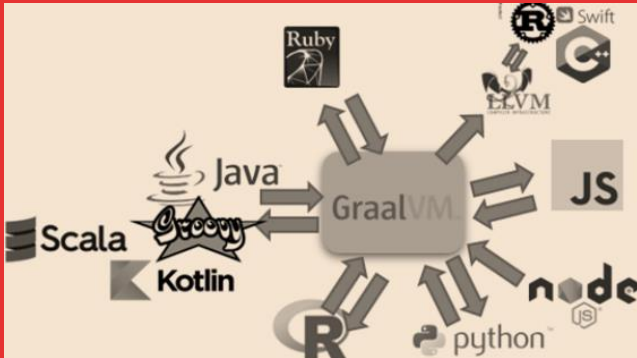


Quarkus (Red Hat)

Kubernetes-native Java framework tailored for GraalVM and Hotspot



Polyglot Platform with Language Interoperability



Polyglot *Interoperability*

- Establish frictionless interaction between language contexts
 - To benefit from the best of all wor(l)ds





KEEP
CALM
THERE'S
ALWAYS
MAÑANA



Polyglot Interoperability

- Establish frictionless interaction between language contexts
 - To benefit from the best of all wor(l)ds
- Call out to 'other world' in a native way
 - Receive response in interpretable format
 - Minimal hoops to jump through at design time
 - Minimal performance overhead at run time
 - Uncomplicated run time environment
 - Despite multiple runtime execution languages
- Basically: *polyglot with the ease of mono-glot*



Current interoperability on JVM

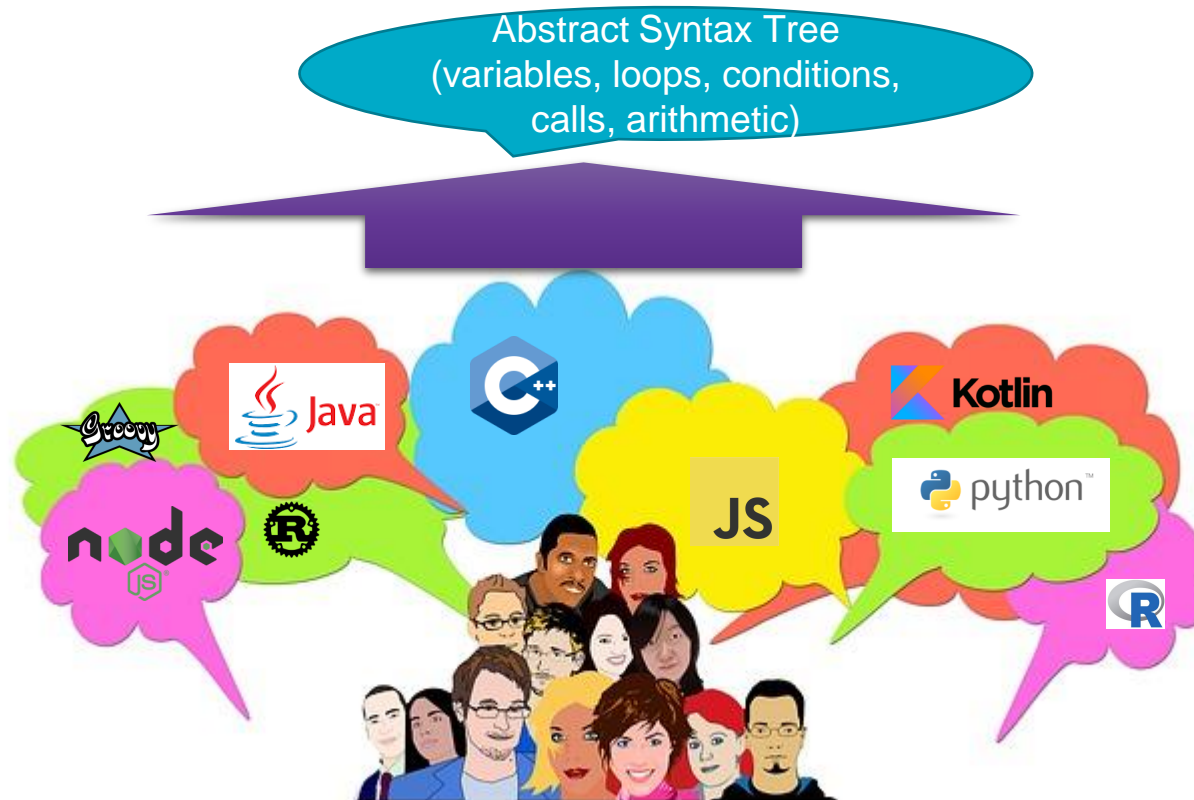
- Current interoperability options from Java:
 - file or queue
 - REST API
 - native inter
 - JSR-223 In
 - (and supp
 - intra-JVM,
 - Java/Groov
- Less than pe
- technical limi
- complex runt



All languages are used to express and communicate very similar concepts

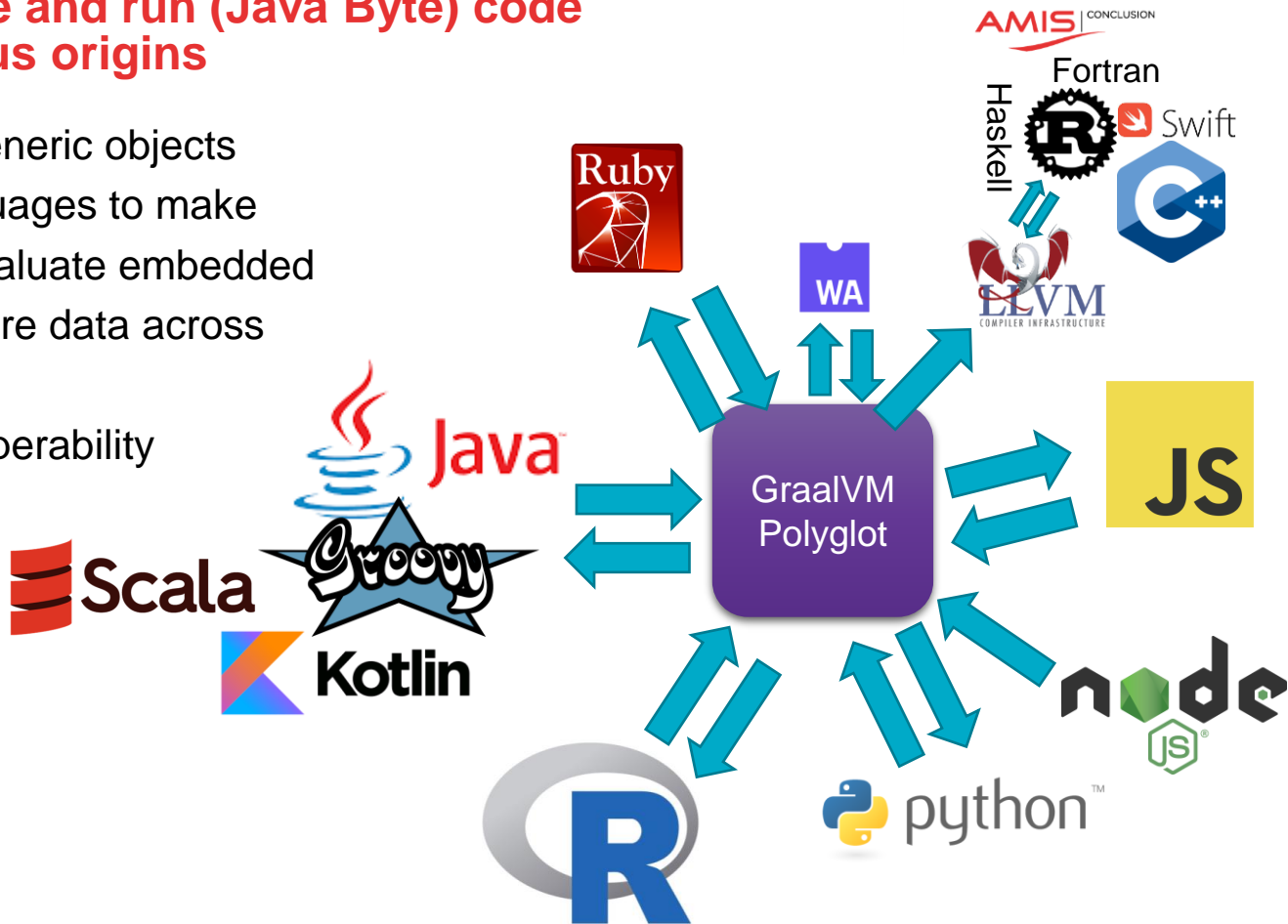


All programming languages are used to express and execute very similar concepts



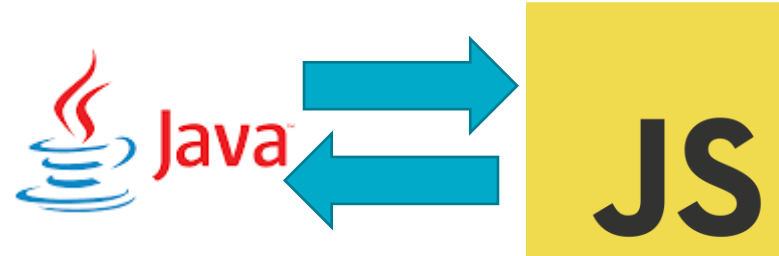
GraalVM can merge and run (Java Byte) code derived from various origins

- GraalVM provides generic objects in all supported languages to make Polyglot call-outs, evaluate embedded foreign code and share data across language contexts
- Thus enabling interoperability



The New Beast between Java and JavaScript

- Out go Rhino and Nashorn
- In comes GraalVM



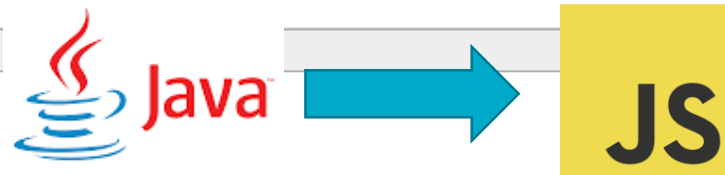
From Java to JavaScript

- Evaluate JS snippets
 - Load and call JavaScript sources
 - Exchange data and objects back and forth
 - Allow JavaScript to callback to Java objects
 - Run multiple JavaScript threads in parallel
-
- Note: what applies to JavaScript by and large applies to all languages that can run on GraalVM through Truffle
 - Including Python, R, Ruby, WebAssembly, LLVM (C/C++, Rust, Swift,...)
 - Your own DSL



From Java to JavaScript

Evaluate a Simple Code Snippet



```
*HelloWorld.java ✕
1 package nl.amis.java2js;
2
3 import org.graalvm.polyglot.*;
4
5 public class HelloWorld {
6     public static void main(String[] args) {
7         Context polyglot = Context.create();
8         polyglot.eval("js", "print('Hello JavaScript!')");
9     }
10 }
```

Import Graal
Polyglot package

Create Polyglot
context

Evaluate
Snippet

Problems Console ✕

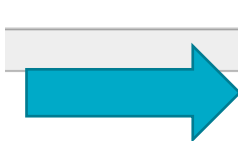
<terminated> HelloWorld [Java Application] /usr/lib/jvm/graalvm-ce-19.2.1/bin/java (Oct 24, 2019, 5:09:23 AM)

Hello JavaScript!

From Java to JavaScript Evaluate a Simple Code Snippet



Java



*HelloWorld.java

```
1 package nl.amis.java2js;  
2  
3 import org.graalvm.polyglot.*;  
4  
5 public class HelloWorld {  
6     public static void main(String[] args) {  
7         Context polyglot = Context.create();  
8         polyglot.eval("js", "print('Hello JavaScript!')");  
9     }  
10 }
```

```
Value helloWorldFunction= polyglot.eval("js"  
    , "(function(name) { return `Hello ${name}, welcome to the world of JavaScript` })");  
// Use the function  
String greeting = helloWorldFunction.execute("John Doe").asString();  
System.out.println(greeting);
```

Import Graal
Polyglot package

Create Polyglot
context

Evaluate
Snippet

Evaluate Snippet =>
instantiate function

Execute function

Problems Console

```
<terminated> HelloWorld [Java Application] /usr/lib/jvm/graalvm-ce-19.2.1/bin/java (Oct 24, 2019, 5:09:23 AM)  
Hello JavaScript!  
Hello John Doe, welcome to the world of JavaScript
```

From Java to JavaScript Evaluate a Simple Code Snippet



```
*HelloWorld.java
1 package nl.amis.java2js;
2
3 import org.graalvm.polyglot.*;
4
5 public class HelloWorld {
6     public static void main(String[] args) {
7         Context polyglot = Context.create();
8         polyglot.eval("js", "print('Hello JavaScript!')");
9
10        Value helloWorldFunction= polyglot.eval("js"
11            , "(function(name) { return `Hello ${name}, welcome to the world of JavaScript` })");
12
13        // Use the function
14        String greeting = helloWorldFunction.execute("John Doe").asString();
15        System.out.println(greeting);
16
17        // Handle Exception Thrown in JavaScript
18        try {
19            polyglot.eval("js", "print('Hello JavaScript!'); throw 'I do not feel like executing this';");
20        } catch (PolyglotException e) {
21            System.out.println("Exception caught from JavaScript Execution. Original Exception: "+e.getMessage());
22        }
23    }
24 }
```

Problems Console

```
<terminated> HelloWorld [Java Application] /usr/lib/jvm/graalvm-ce-19.2.1/bin/java (Oct 24, 2019, 5:09:23 AM)
Hello JavaScript!
Hello John Doe, welcome to the world of JavaScript
Hello JavaScript!
Exception caught from JavaScript Execution. Original Exception: I do not feel like executing this
```

A Typical Java Challenge (aided by JavaScript interaction)

- Developing a Java application



- I need to perform validations on input data
 - Postal Code (various countries), Mobile Phone Numbers (many countries), Email Address, Credit Card Number etc.

- NPM Module Validator offers most of these OOTB



- But... it is written in JavaScript
 - How does that help me?



A Typical Java Challenge

- Developing a Java application

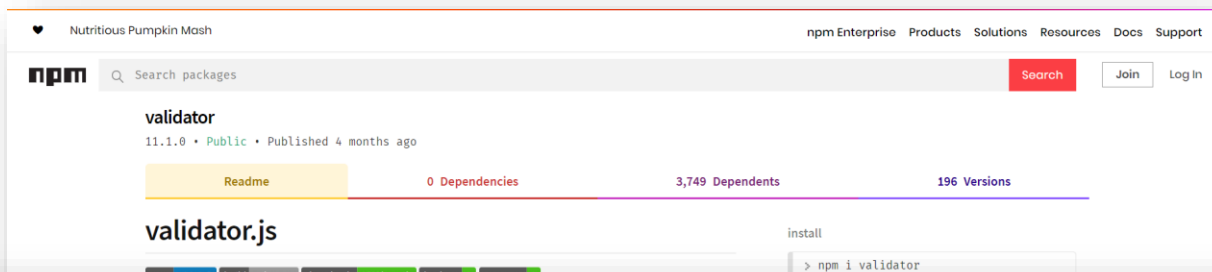


- I need to perform validations on input data
 - Postal Code (various countries), Mobile Phone Numbers (many countries), Email Address, Credit Card Number etc.
- In Pseudo Code:

```
public static void main(String[] args) {  
    Boolean postalCodeValidationResult = postalCodeValidator.execute("2716 NK","NL")  
    System.out.println("Postal Code Validation Result "+ postalCodeValidationResult);  
    postalCodeValidationResult = postalCodeValidator.execute("XX 27165 ","NL")  
    System.out.println("Postal Code Validation Result "+ postalCodeValidationResult);  
}
```

Community package in JavaScript for dozens of predefined validations

AMIS | CONCLUSION



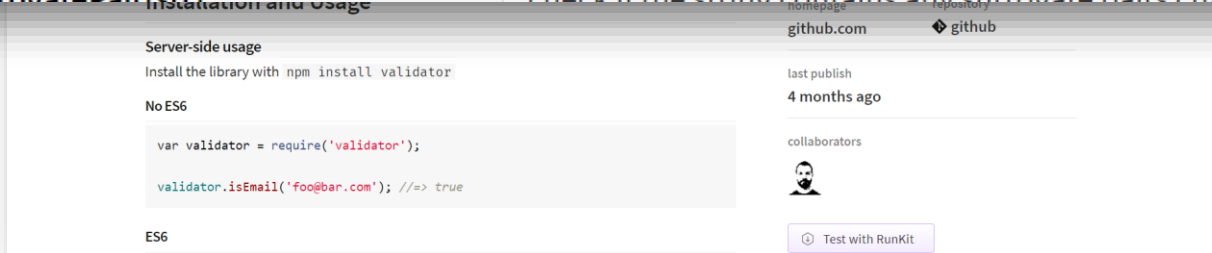
`isPostalCode(str, locale)`

check if the string is a postal code,

(locale is one of ['AD', 'AT', 'AU', 'BE',

`isSurrogatePair(str)`

check if the string contains any surrogate pairs chars



JavaScript Validator Module is compatible with GraalJS

AMIS | CONCLUSION



GraalVM™

The screenshot shows the GraalVM website's compatibility checker. The header includes the GraalVM logo, navigation links (Home, Docs, Downloads, Community), social media icons, a star count (10,670), and a 'TRY GRAALVM' button. The left sidebar contains a search bar and a list of links: Why GraalVM?, Getting Started, Examples, Compatibility (highlighted), Release Notes, Reference Manual, GraalVM as a Platform, Security Guide, Support, FAQ, and Search. The main content area has a heading 'Quickly check if an NPM module, Ruby gem, or R package is compatible with GraalVM.' Below this is a text input field containing 'validator', a clear button (x), and a 'CHECK!' button. The results are displayed under the heading 'Graal.js' in a table.

NAME	VERSION	STATUS
validator	~> 9.0	100.00% tests pass
validator	~> 8.0	100.00% tests pass
validator	~> 7.0	100.00% tests pass
validator	~> 5.0	100.00% tests pass
validator	~> 10.9	100.00% tests pass
validator	~> 10.6	100.00% tests pass
validator	~> 10.5	100.00% tests pass
validator	~> 10.4	100.00% tests pass
validator	~> 10.11	100.00% tests pass
validator	~> 10.10	100.00% tests pass

A Typical Java Challenge – Resolved (with GraalVM)

- Embrace GraalVM as the Java runtime
- Install npm module *validator*
- Install and use webpack to create a single bundle for module *validator*
- Extend Java application
 - Import GraalVM Polyglot package
 - Create JavaScript Context
 - Load *validator_bundled.js* into context
 - Get desired JS functions from bindings map
 - Execute required JS functions as if they were Java methods

GraalVM™

npm



A Typical Java Challenge – Resolved (with GraalVM)

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

public class Java2JS {
    public static void main(String[] args) {
        Context c = Context.create("js");
        c.eval(Source.newBuilder("js", new File("./bin/validator_bundled.js")).build());
        System.out.println(c.getBindings("js").getMemberKeys());
        Value postalCodeValidator = c.getBindings("js").getMember("isPostalCode");
        Boolean postalCodeValidationResult = postalCodeValidator.execute("2716 NK", "NL").asBoolean();
        System.out.println("Postal Code Validation Result " + postalCodeValidationResult);
        postalCodeValidationResult = postalCodeValidator.execute("XX 27165 ", "NL").asBoolean();
        System.out.println("Postal Code Validation Result " + postalCodeValidationResult);
    }
}
```

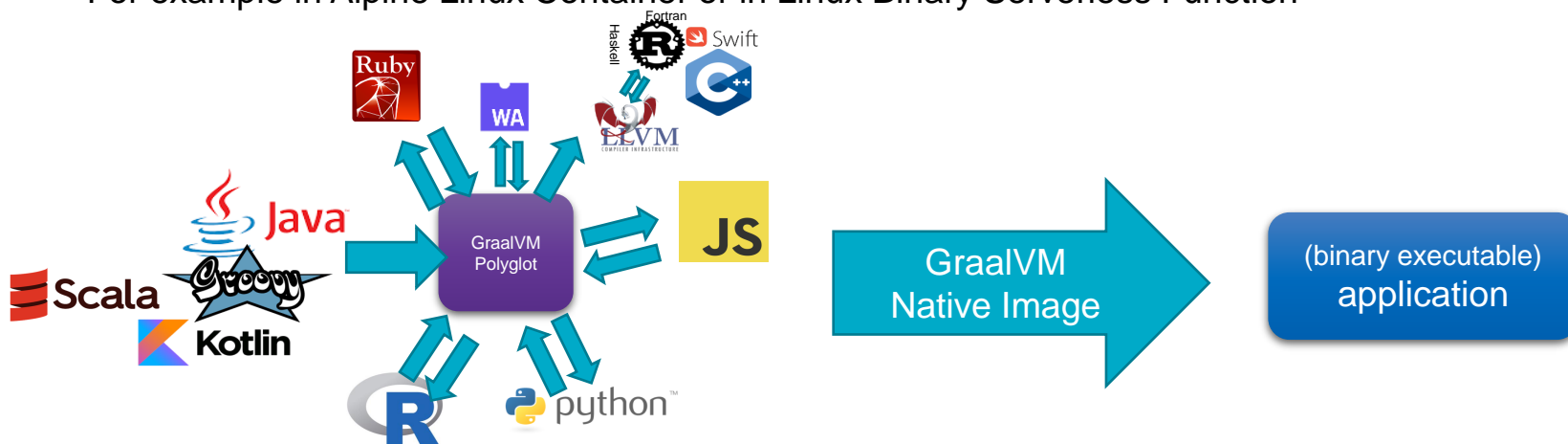


Problems Console

```
<terminated> ValidateThroughNPMValidator [Java Application] /usr/lib/jvm/graalvm-ce-19.2.1/bin/java (Oct 24, 2019, 3:59:00 AM)
All functions available from Java (as loaded into Bindings) [isISBN, isMongoId, squareRoot, isLength, isISSN, isISO8601, trim, toB
Postal Code Validation Result true
Postal Code Validation Result false
```

Polyglot and Ahead of Time Compiled Native Binary

- A multi-lingual or polyglot Java application can be turned into a Native Binary Executable using GraalVM Ahead of Time compilation
 - Non JVM-languages can be included but are not compiled ahead of time
 - The build process for the binary executable takes long & uses a lot of memory
- The outcome: a single, small[ish] file running on plain Linux
 - For example in Alpine Linux Container or in Linux Binary Serverless Function

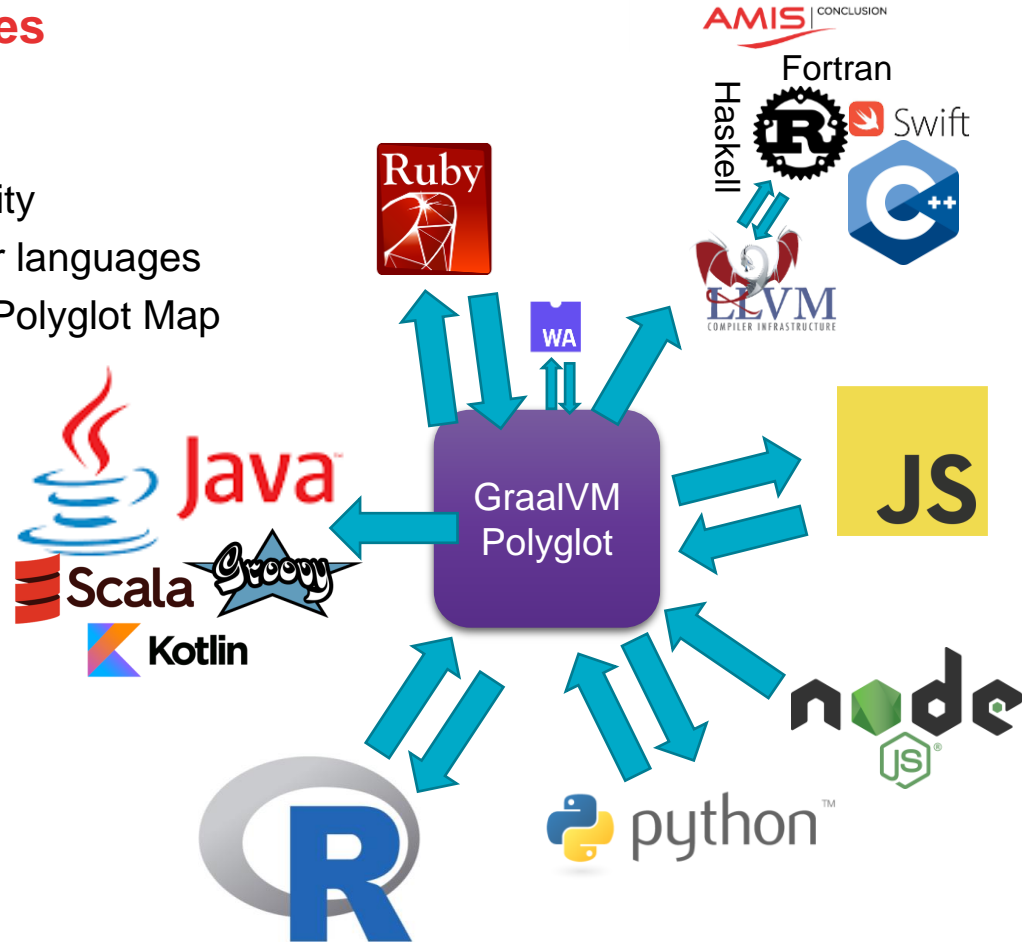


Polyglot
from Node | Python | Ruby | R | ...
to any language on GraalVM

The logo for GraalVM, featuring the text "GraalVM" in a blue and orange font, with a small "TM" trademark symbol.

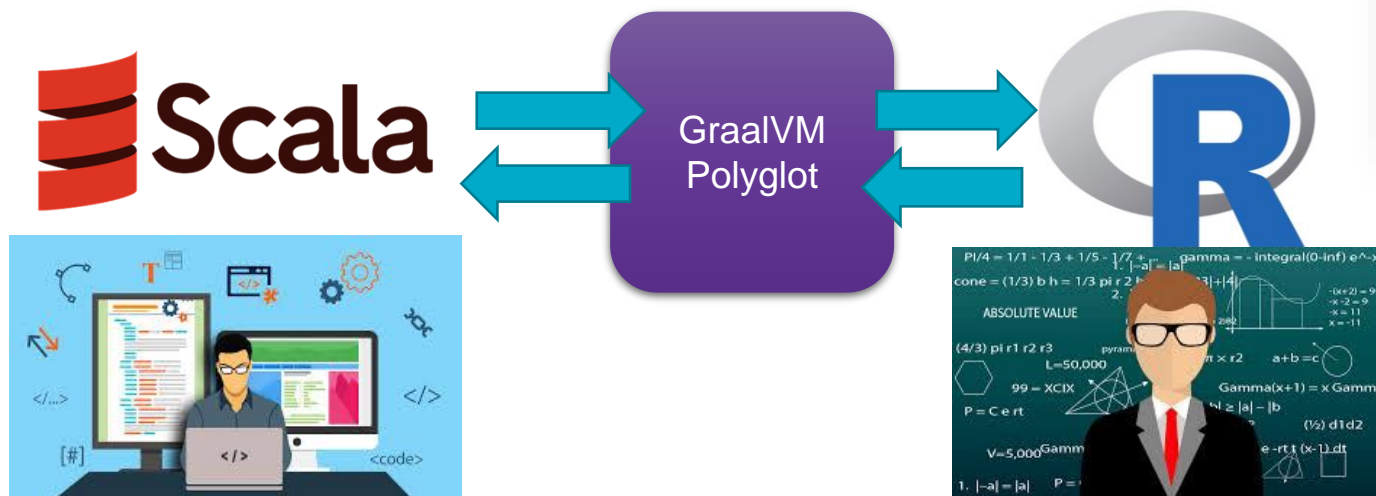
Polyglot from non-JVM languages

- Any language running on GraalVM can participate in Polyglot interoperability
 - Evaluate & Execute Snippets in other languages
 - Export data and objects through the Polyglot Map to other execution contexts
 - Import data and objects from the Polyglot Map from other language contexts
 - Read, write & execute
- Interacting with JVM based languages is slightly different
 - Use Java Type to construct native object based on a Java Class



Polyglot from non-JVM languages

Dutch National Police Case: Scala \Leftrightarrow (Fast)R on GraalVM

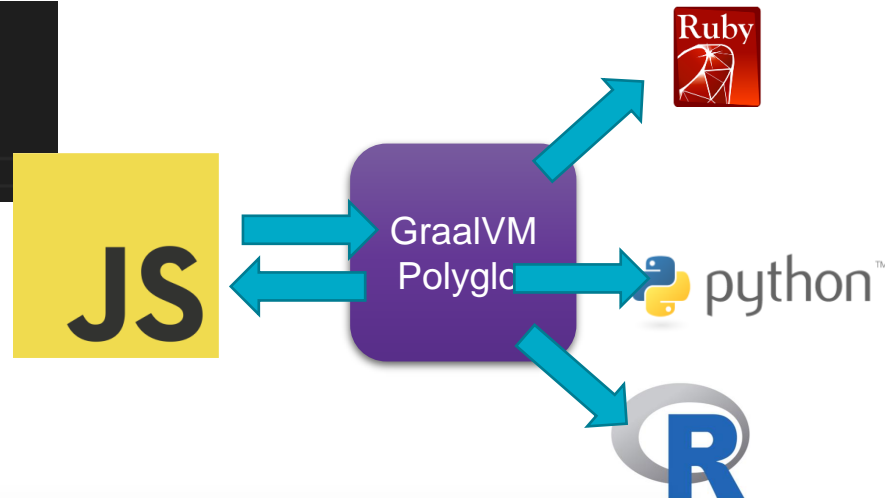


Polyglot from JavaScript to other Graal Truffle Languages

- Polyglot.eval can execute any language snippet
 - It returns data objects (to read/write) or functions (that can be executed)

JS

```
var arrayR = Polyglot.eval("R", "c(1,2,42,4)")  
var arrayPython = Polyglot.eval("python", "[11,21,84,41]")  
var arrayRuby = Polyglot.eval("ruby", "[1,2,42,4]")  
console.log("R says: "+arrayR[2]);  
console.log("Python says: "+arrayPython[3]);  
console.log("Ruby says: "+arrayRuby[1]);
```





```
R says: 42  
Python says: 41  
Ruby says: 2
```


Polyglot from JavaScript to other Graal Truffle Languages

- Polyglot.eval can execute any language snippet
 - It returns data objects (to read/write) or functions (that can be executed)

JS

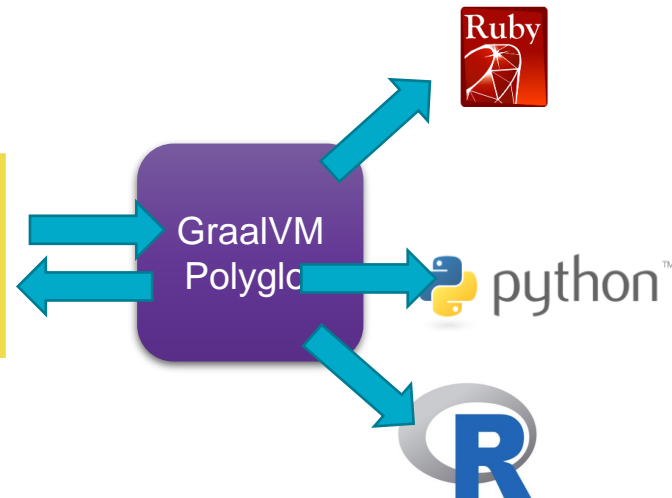
```
// load a Python Library that defines a Function called Fibonacci;
Polyglot.evalFile("python", "library.py");
fibFunc = Polyglot.eval("python", `Fibonacci`)  python
print("Fibonacci(9) - called from JS, executed in Python "+fibFunc(9))

// create a lambda function in Python, return it to JS and invoke it from JS
var square = Polyglot.eval("python", `(lambda x: x*x)`);  python
var x = 5
print(`Square of ${x} = ${square(x)} - according to Python`)

logInR = Polyglot.eval("R", `function(input) { 
|   print( paste('Message from R : ', input))
| }`)
logInR("Message from JavaScript")
```

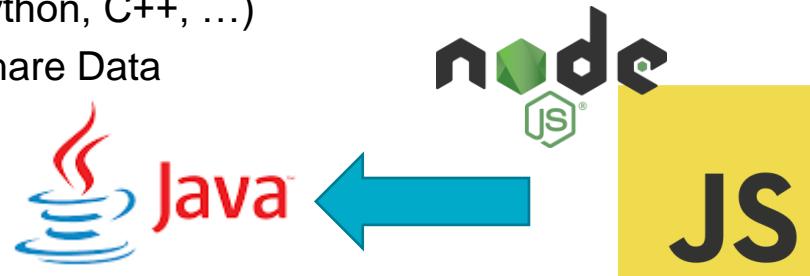
```
Fibonacci(9) - called from JS, executed in Python 21
Square of 5 = 25 - according to Python
[1] "Message from R : Message from JavaScript"
```

JS



Node Application calling out to Java (and others)

- GraalVM can run Node & JS applications
 - GraalJS /GraalVM can replace V8 as interpreter & execution engine
 - GraalJS runs Java Byte code on JVM
 - This engine is a Java application that works on any Java 8+ implementation
 - Can run faster and better scalable to big memory structures
 - However: it may need some warmup time to reach peak performance
 - GraalVM 20.1: based on Node 12.15 and ECMAScript 2020 compliant
- Node application running with GraalVM can leverage GraalVMs Polyglot – with Java as well as other languages (R, Ruby, Python, C++, ...)
 - Evaluate Code Snippets, Instantiate Objects, Share Data



Node Application calling out to Java

```
js joker2.js x
js2java > js joker2.js > ...
6
7  const javaJokerClass = Java.type('nl.amis.js2java.Joker');
8  const javaJoker1 = new javaJokerClass()
9
10 function crackJoke() {
11     print(getJoke())
12 }
13
14 function getJoke() {
15     return javaJoker1.getJoke()
16 }
17
18 var i=1
19 while (i++<10) {
20     crackJoke()
21 }
```

```
node --jvm --vm.cp application-bundle.jar joker2.js
```



GraalJS

Java 11 APIs

JVM & GraalVM

application-bundle.jar

Joker.class

node
Node Application

This bloke said to me: 'I'm going to attack you with the neck of a guitar.' I said: 'Is that a fret?' I have kleptomania. But when it gets bad, I take something for it.
I had a dream last night that I was cutting carrots with the Grim Reaper – dicing with death.
I went to buy some camouflage trousers the other day, but I couldn't find any.
I'd like to start with the chimney jokes – I've got a stack of them. The first one is on the house.
I had a dream last night that I was cutting carrots with the Grim Reaper – dicing with death.

Node Application calling out to Java

Get hold of Java Class and
instantiate Java Object

```
const javaJokerClass = Java.type('nl.amis.js2java.Joker');  
const javaJoker1 = new javaJokerClass()
```

Invoke method on Object
and get String returned

```
function getJoke() {  
  return javaJoker1.getJoke()  
}
```

```
node --jvm --vm.cp application-bu
```

```
Joker.java  
1 package nl.amis.js2java;  
6 public class Joker  
7  
8   String[] jokes = {"What's the best thing about Switzerland? I don't know, but  
9                     "As a scarecrow, people say I'm outstanding in my field. But  
10                    "The best thing about my car is that it's the only one in the world  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22   public String getJoke()  
23     String joke = jokes[new Random().nextInt(jokes.length)];  
24     return joke;  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39 }
```

Joker.class

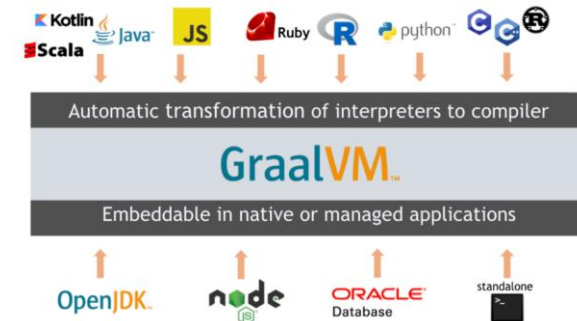
node
Node Application

Conclusion



Quick Recap – What is GraalVM?

- Thomas Wuerthinger: “a new virtual machine that is trying to execute many languages faster”
- JIT Foundation for the Long Term Evolution of HotSpot JVM
 - Stable, observable, manageable and maintainable JVM with optimizations for modern Java and JVM language patterns
- Serverless Java
 - Small footprint, rapid startup time, simple & small run time
- Interoperability platform aka Polyglot VM
 - Run and interact between many languages
- Bonus features
 - Custom DSL
 - Native interaction between C/C++ and Java
 - Polyglot Tools: Debugger (with Chrome Dev Tools), Profiler, GraalVM VisualVM, Ideal Graph Visualizer, Visual Studio Code extension for GraalVM



Adopting GraalVM in your team - your first steps



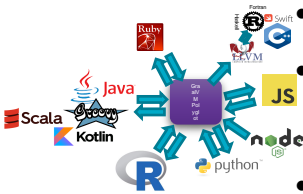
- Start exploring GraalVM – download & install, play and do a PoC
 - Plenty of resources available (code samples, blog articles, tutorials, ..)
- Hybrid: have some containers running GraalVM (for Java applications)

instead of your traditional JVM brand

- to try out, check if it is stable and performant and (better) manageable
- Start leveraging polyglot



- single runtime for different programming languages)
- interoperability (benefit in Java from 3rd party libraries in JavaScript or Python [or vice versa] and/or make better use of the skills in your team)
 - better start creating standards and guidelines around when and how to polyglot!
- AOT/native - use GraalVM to build Java applications as native executables as last step in the build process and deliver that executable instead of JAR files
 - create Java based Serverless Functions with small footprint and very quick startup
 - benefit from ultra fast and super small frameworks (Quarkus, Helidon, Micronaut, Vert.x, Spring, ...)



Thank you for your attention

Slides:

<https://bit.ly/acesathome-graalvm>

Live Handson Environments

bit.ly/graalvm-handson



lucas.jellema@amis.nl | technology.amis.nl |  @lucasjellema |  lucas-jellema





GraalVM™