



CodeVash  
2019



## Seeking the holy Graal

Presenter: David Lucas

L  
S  
E





# Who am I ?

- Over 25 years in software industry
- Working with Java since 1998
- IntelliJ / JetBrains introduced me to Kotlin
- Google made me realize ...

**Kotlin is now the new Java**

- I am a Kotlin Enthusiast
- Extensive production deployments
- Experience with Make, Ant, Maven, and new to Gradle



David Lucas  
Lucas Software Engineering, Inc.  
[www.lse.com](http://www.lse.com)  
[ddlucas@lse.com](mailto:ddlucas@lse.com)  
[@DavidDLucas](https://twitter.com/DavidDLucas)



# My Agenda

- Not a big fan of polyglot  
(but do realize data scientists care)
- Not a big fan of stored procedures  
(but do realize big data cares)
- My goal is to shrink resource usage for  
microservices (jar -> exec)
- Alternative for Kotlin Native ?



# Goals

- Introduce GraalVM
- Demo some capabilities (js, R, rb, py)
- Demo node polyglot (js, java, R)
- Demo LLVM Interpreter
- Demo JDK 11 Nashorn vs GraalJS
- Demo Script Inspection
- Demo Jar to Native Image
- Summary



# GraalVM Intro

ORACLE®

- Graal is a highly optimized AOT / JIT compiler
- GraalVM is the combination of many Oracle projects over the last decade
- Focus was on improving performance of resource usage
- Platforms: JVM, Node.js, Native
- True Polyglot Runtime (shared data and functions)
- “Make development more productive and run programs faster anywhere” —@graalvm



# GraalVM Intro

ORACLE®

- Languages: Java, JavaScript  
R, Ruby, Python
- LLVM bit code can be executed
- Generates shared libraries and executables
- Oracle has had a JVM in their database since 2009 for Java Stored Procedures
- Adding support in MySQL
- Twitter in PROD, tweet service in 2017  
(built own includes GraalVM)





# GraalVM Intro

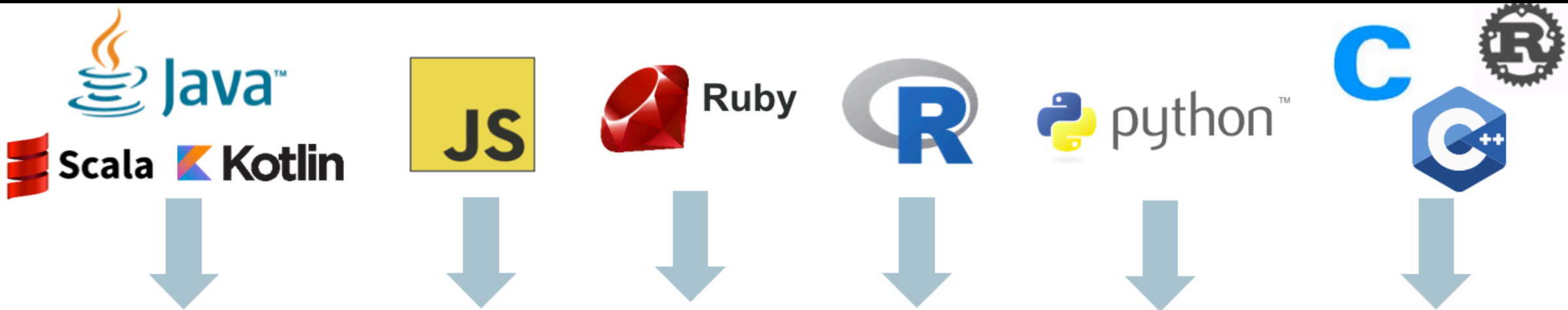
ORACLE®

- Ahead-Of-Time (AOT) Compilation (static) into Intermediate Representation (IR)
- Convert IR to native in more optimized fashion
  - Speculates results and references
  - De-optimizes and Re-optimizes
  - Snippets (inlining)
- Performs advance escape analysis and initialization before execution
- Details on how it creates a smart IR:

[http://lafo.ssw.uni-linz.ac.at/papers/2013\\_Onward\\_OneVMToRuleThemAll.pdf](http://lafo.ssw.uni-linz.ac.at/papers/2013_Onward_OneVMToRuleThemAll.pdf)

<http://www.graalvm.org/docs/why-graal/>

7 L  
S  
E



Automatic transformation of interpreters to compiler

# GraalVM™

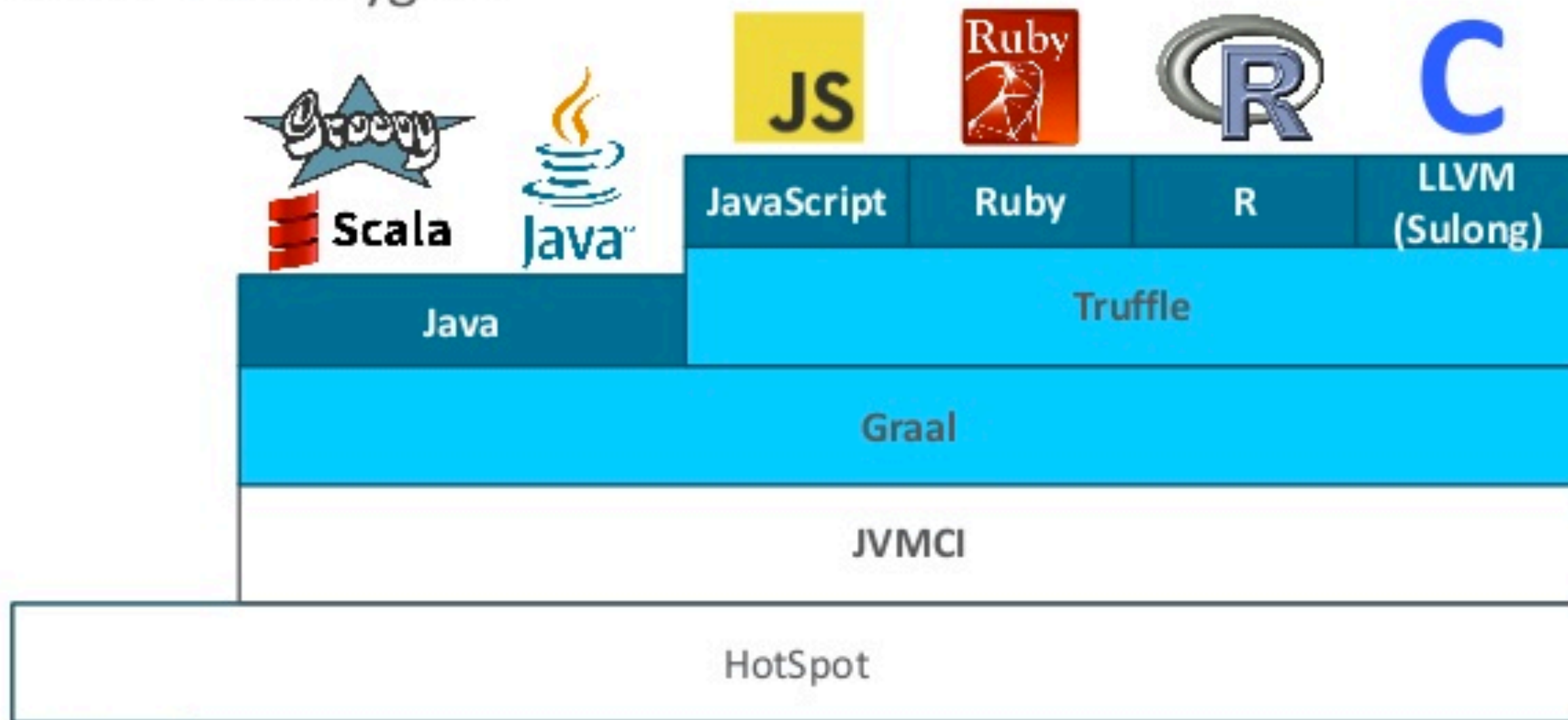
Embeddable in native or managed applications



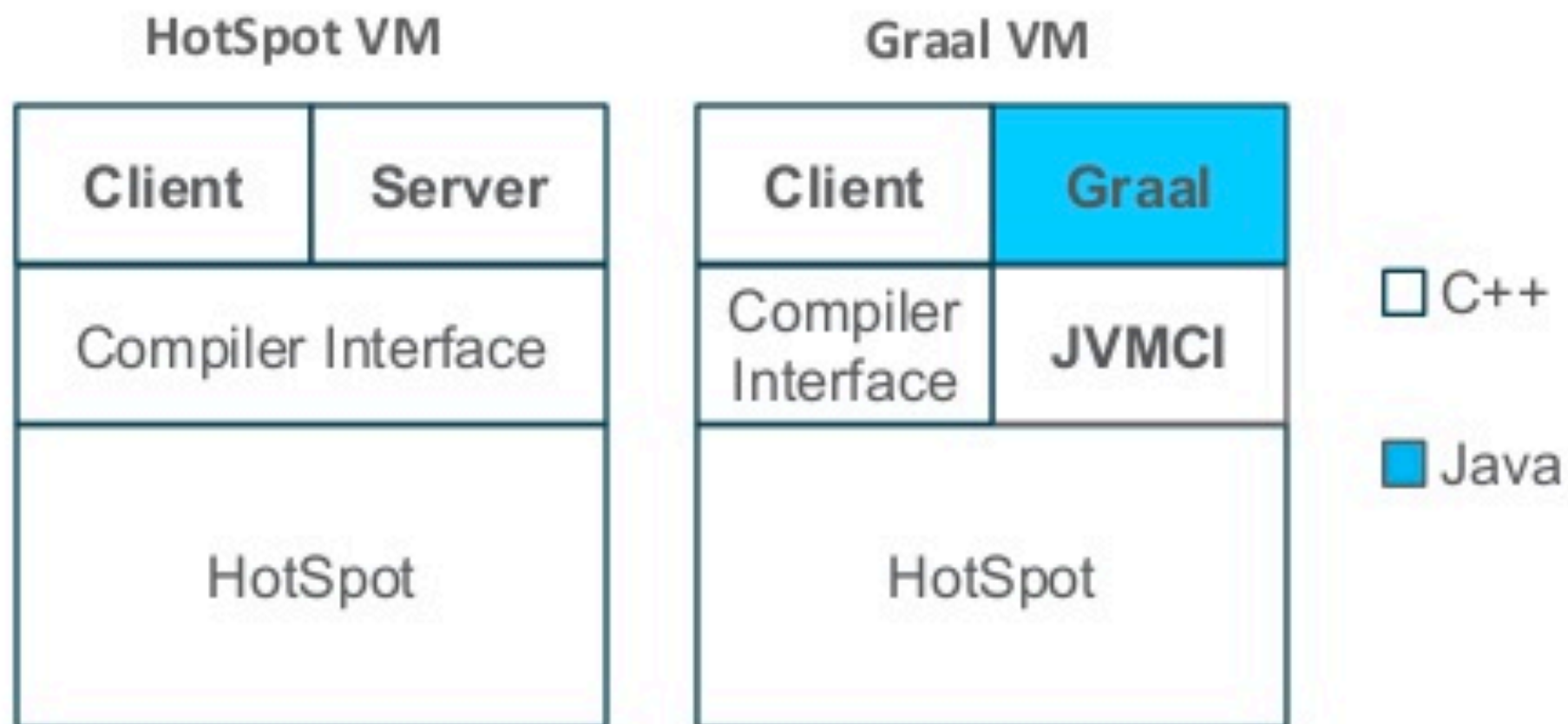




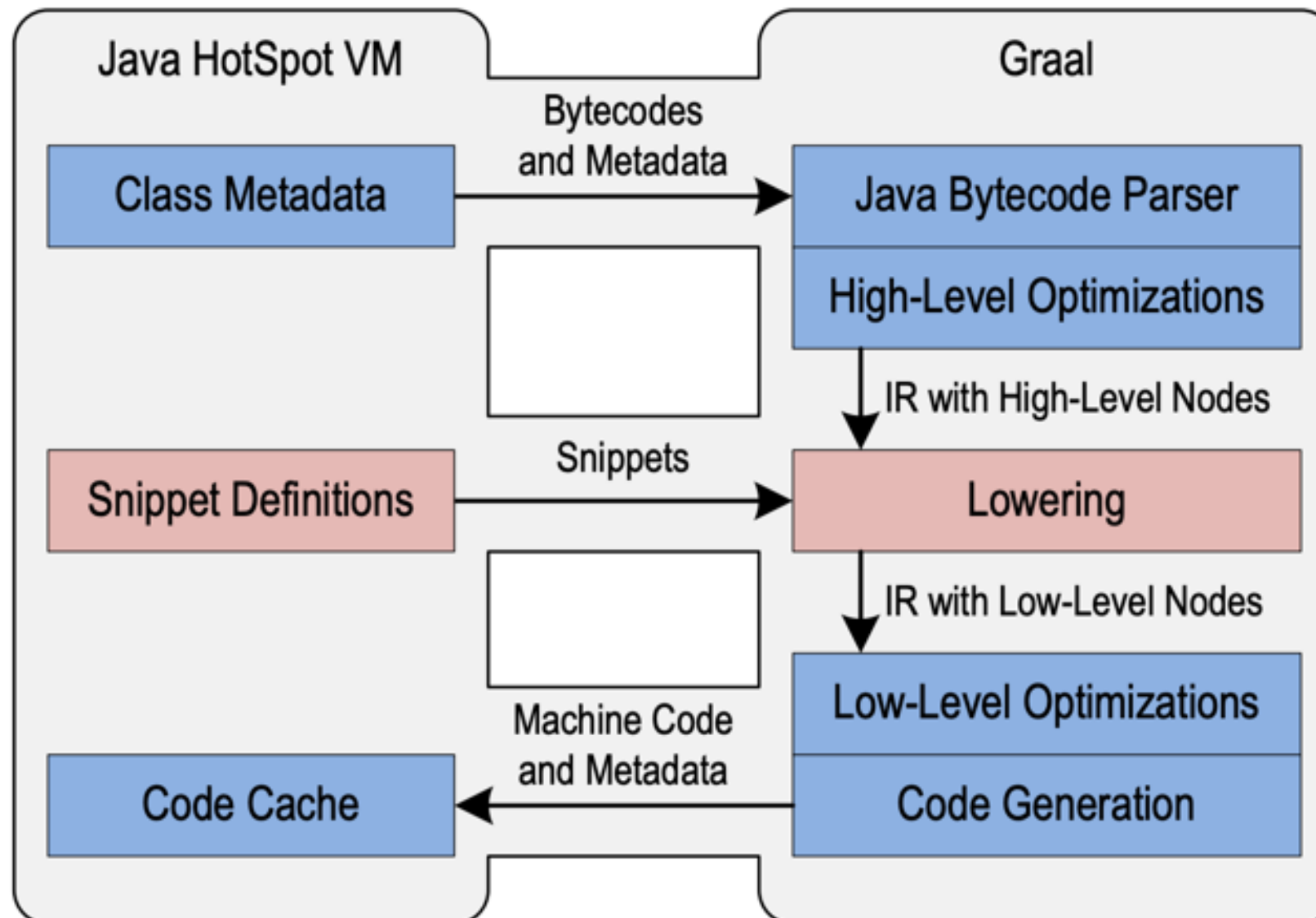
## Graal VM Polyglot



## Graal and Graal VM



# Compiler-VM Separation

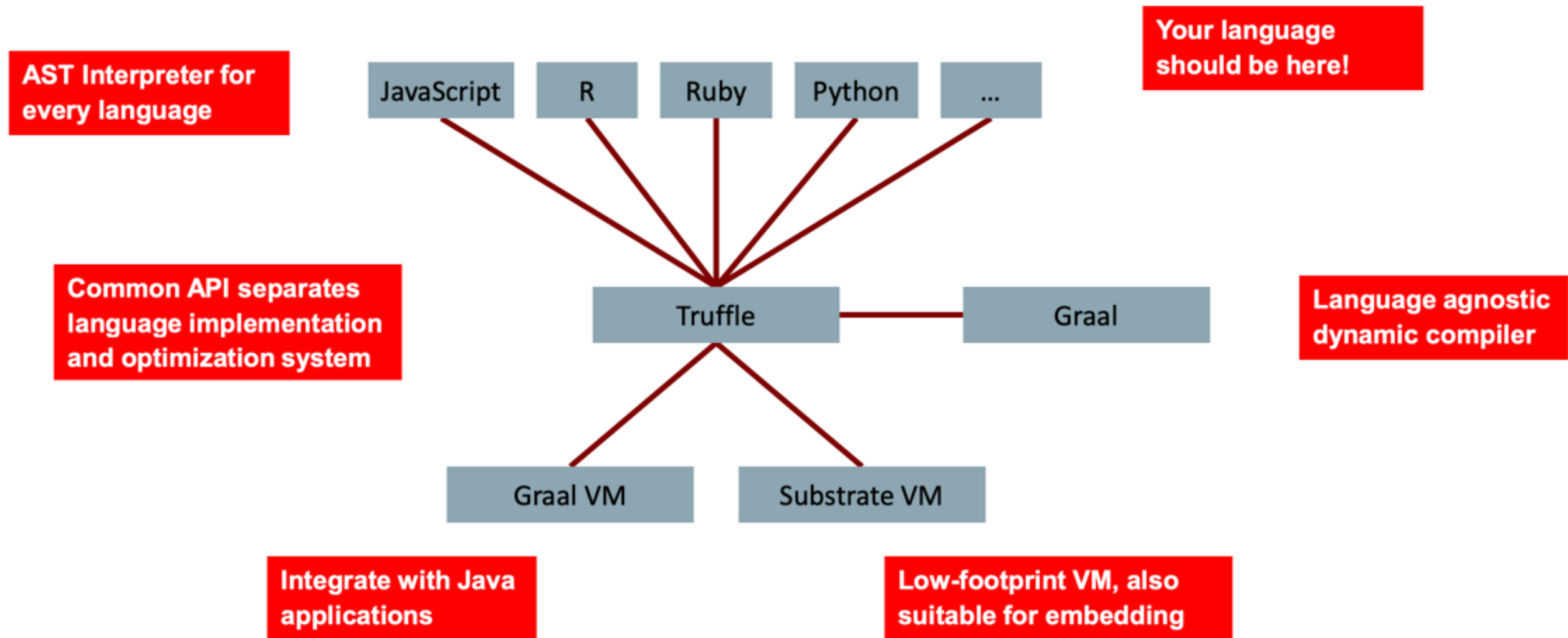




- Truffle API
  - Declarative
  - AST representation of source code
  - Convert AST into IR
  - Written in Java
  - Script Engines use Truffle to create AST
  - Truffle AST used to generate IR
  - IR used to create byte code or native code



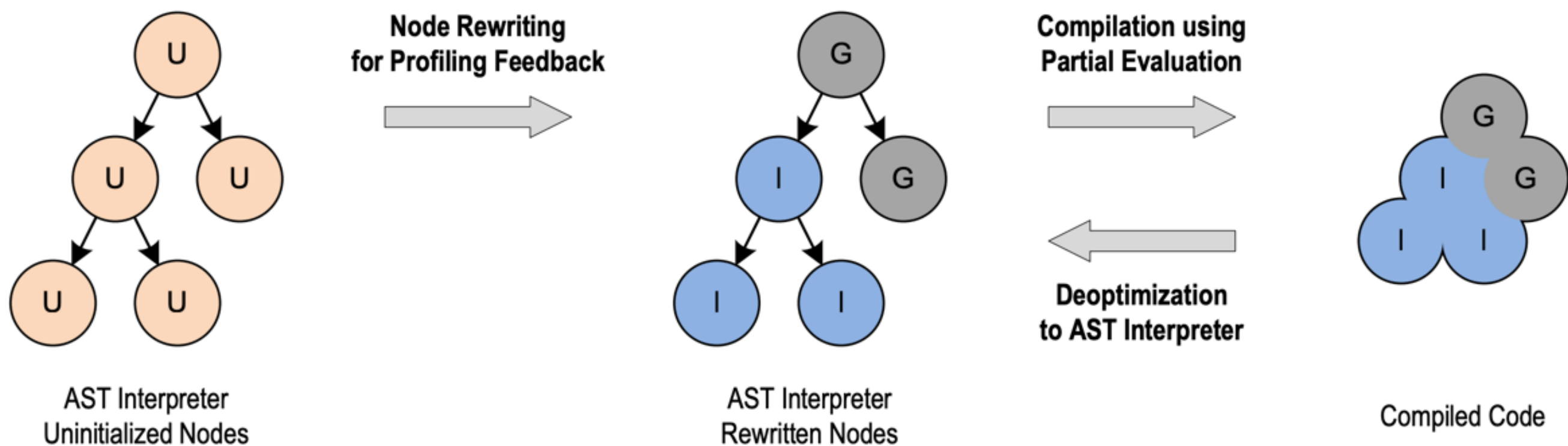
# Truffle System Structure







# Truffle Approach

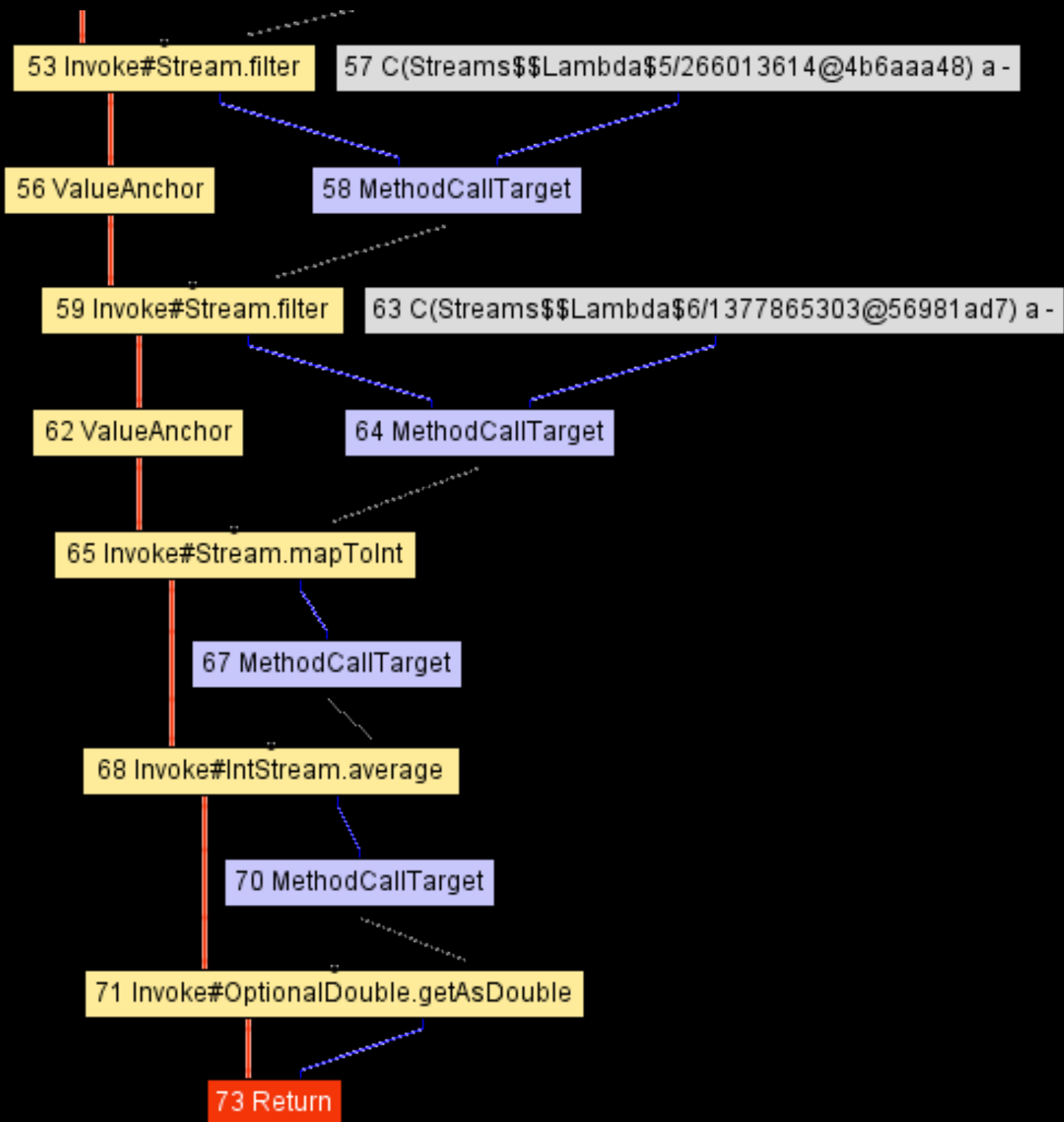




# GraalVM

## Intermediate Representation (IR)

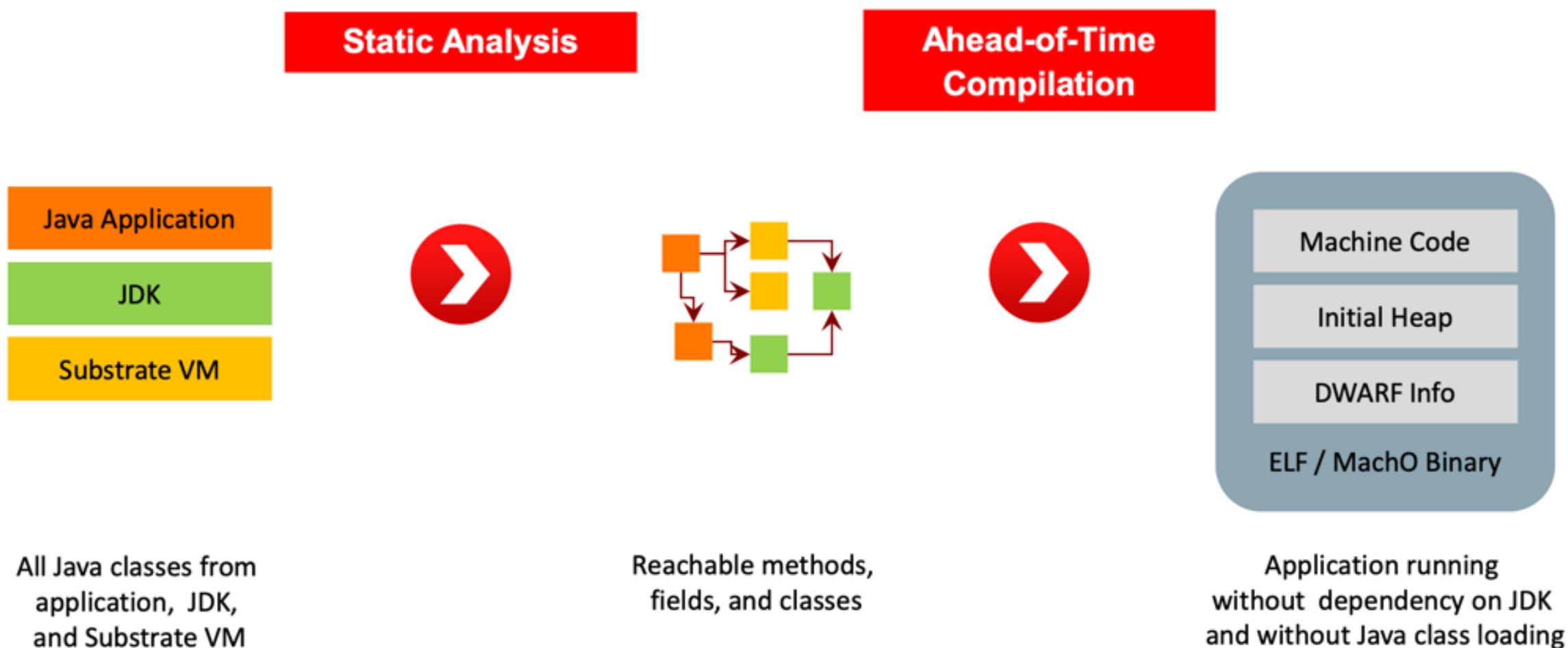
ORACLE®





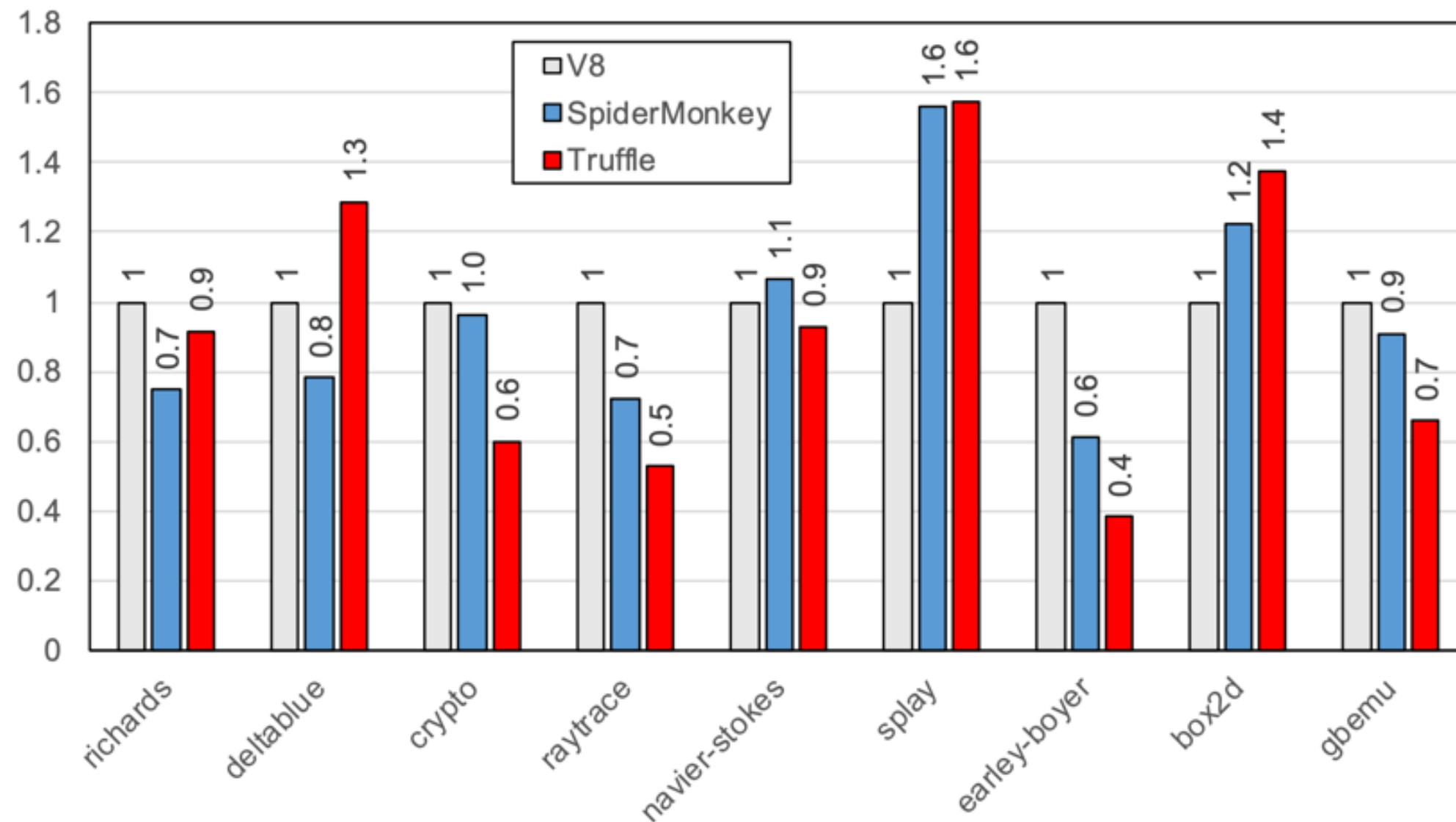
# Substrate VM

## Static Analysis and Ahead-of-Time Compilation using Graal





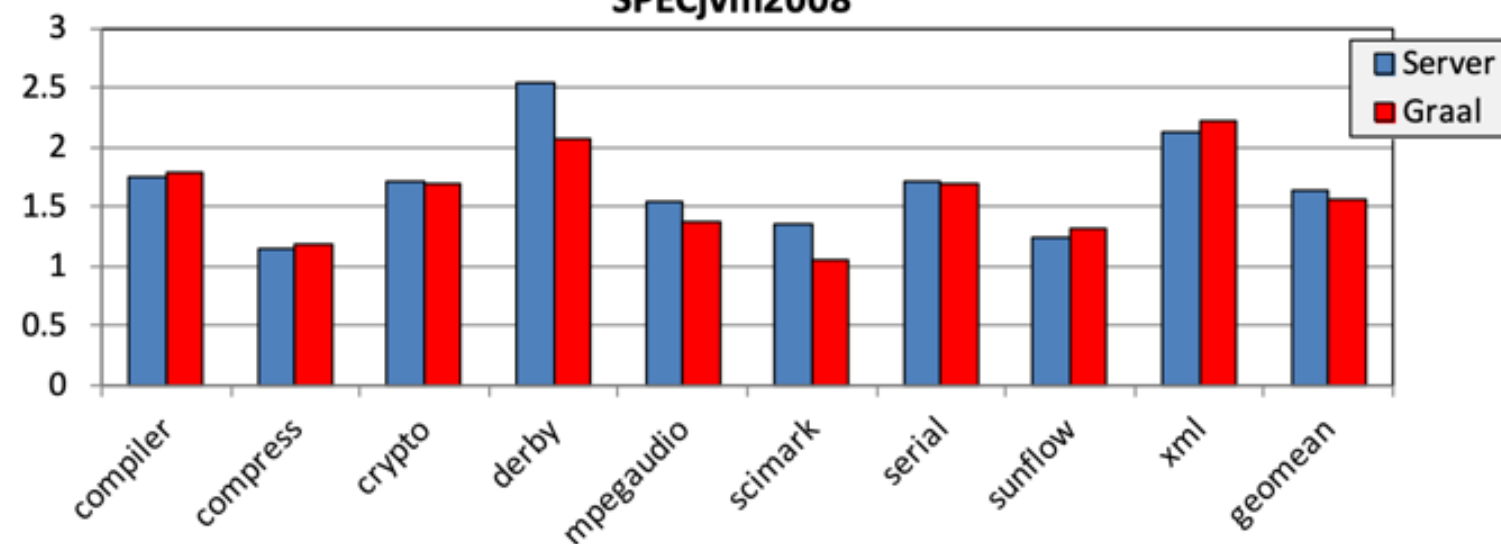
# Performance: JavaScript



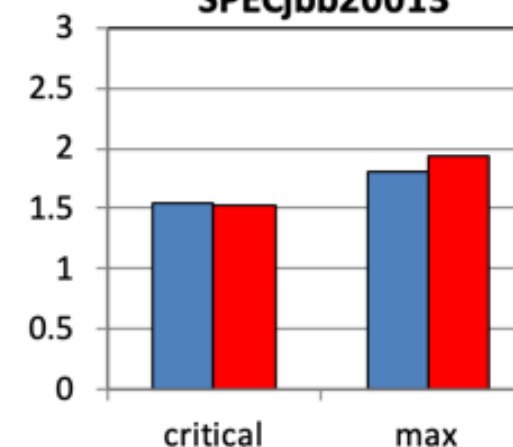


# Graal Benchmark Results

SPECjvm2008



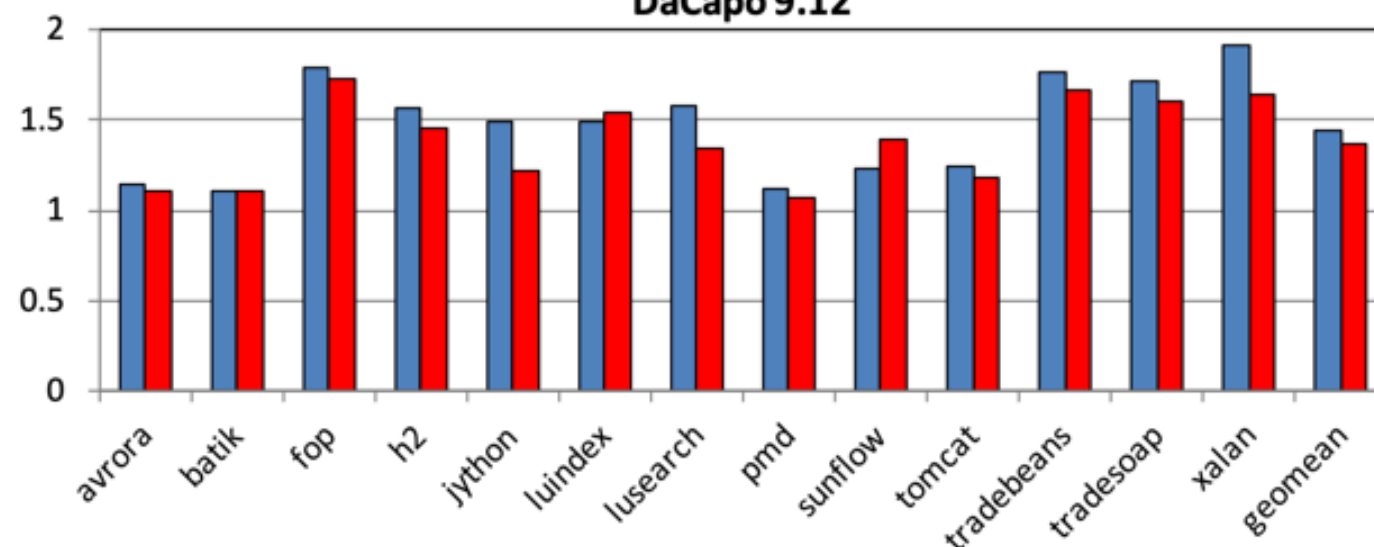
SPECjbb20013



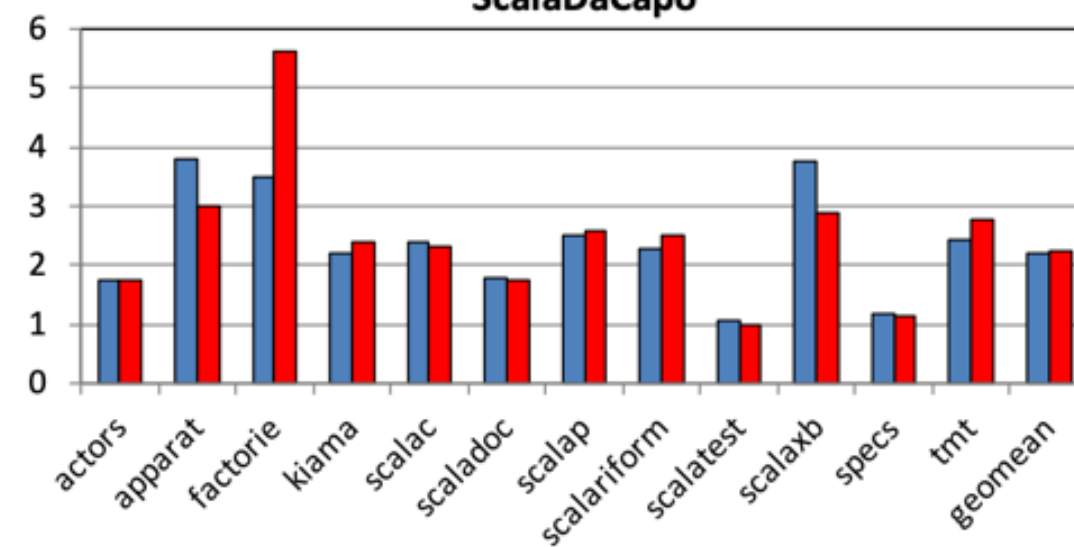
Higher is better,  
normalized to  
Client compiler.

Results are not SPEC  
compliant, but follow the  
rules for research use.

DaCapo 9.12



ScalaDaCapo







# GraalVM Setup

- GraalVM CE (Linux and Mac)  
<http://www.graalvm.org/downloads/>
- GraalVM EE (has Windows Preview)  
<https://www.oracle.com/technetwork/oracle-labs/program-languages/downloads/index.html>
- Follow installation instructions
- Add GRAALVM\_HOME to your PATH
- verify

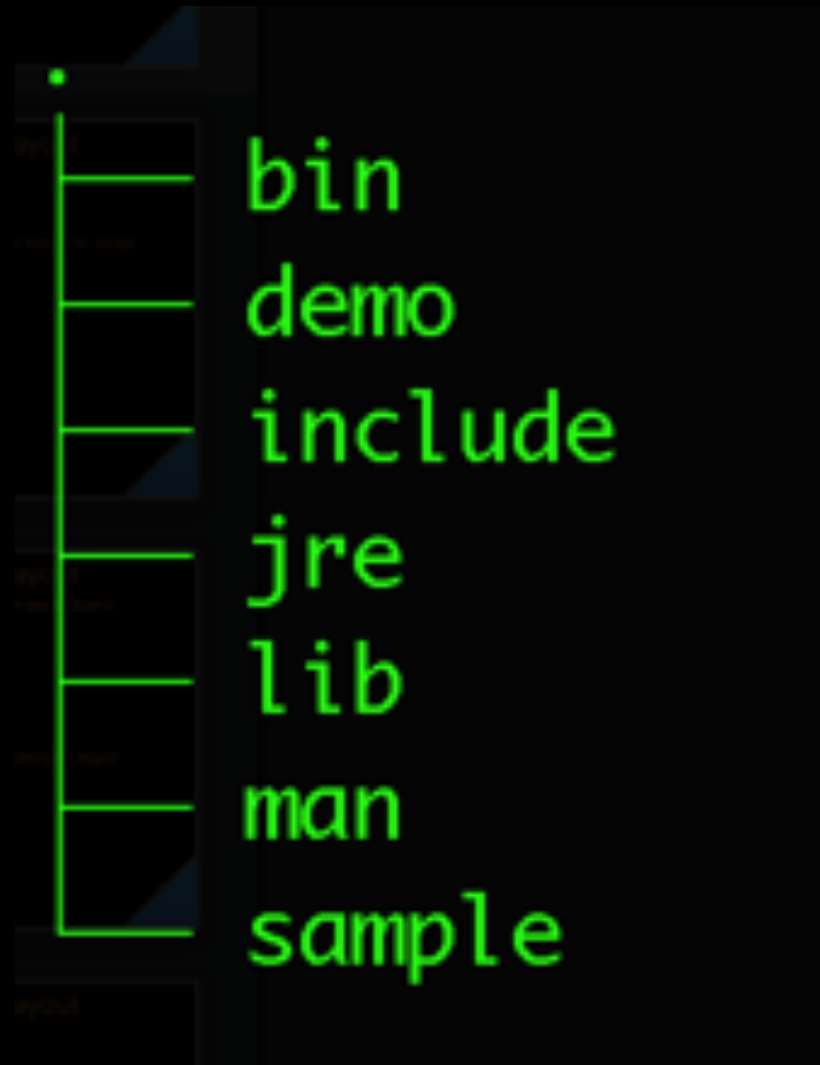
```
> java -version
openjdk version "1.8.0_192"
OpenJDK Runtime Environment (build 1.8.0_192-20181024123616.buildslave.jdk8u-src-tar--b12)
GraalVM 1.0.0-rc10 (build 25.192-b12-jvmci-0.53, mixed mode)
```

<http://www.graalvm.org/docs/getting-started/>



# GraalVM Layout

## Typical JDK Structure





# GraalVM Layout

## New Members

R	jarsigner	jhat	jstatd	policytool	serialver
Rscript	java	jinfo	jvisualvm	polyglot	servertool
appletviewer	javac	jjs	keytool	rake	testrb
extcheck	javadoc	jmap	lli	rdoc	tnameserv
gem	javah	jps	native-image	ri	truffleruby
graalpython	javap	jrunscript	native2ascii	rmic	unpack200
gu	jcmm	js	node	rmid	wsgen
idlj	jconsole	jsadebugd	npm	rmiregistry	wsimport
irb	jdb	jstack	orbd	ruby	xjc
jar	jdeps	jstat	pack200	schemagen	



# GraalVM Tooling

- Graal Updater: installs components (languages)
- Scripts can use Chrome Inspector (`—inspect`)
- Native Image: convert jar to exec or shared library
  - creates Substrate JVM
  - `native-image -H:ReflectionConfigurationFiles=graal_config.js on x.jar # provides explicit class loading`



# GraalVM DEMO

- examples/build.sh will pull down some git repos and place in tmp folder
- it also creates a docker image for Linux demo
- examples can run from Mac or Linux
- this is a subset of GraalVM samples





# GraalVM DEMO Verify

- verify GraalVM on MAC
- export PATH= ...
- ga install R ruby python
- cd examples/hello-poly
- javac HelloPolyglotWorld.java
- java HelloPolyglotWorld



# GraaIVM DEMO FizzBuzz

- Shell into Linux Docker Image
- `docker run -p3000:3000 -it graalvm-demo:0.01 bash`
- source environment  
  `./dockerjava/setup-gvm.env`
- add experimental languages  
  `ga install R ruby python`
- `cd /dockerjava/graalvm-ten-things`
- `which java && which R && which ruby && which`  
  `graalpython`
- `js fizzbuzz.js`
- `graalpython fizzbuzz.py`
- `Rscript fizzbuzz.r`
- `ruby fizzbuzz.rb`

# GraalVM DEMO Polyglot Node.js

- `node --jvm polyglot.js`
- `which npm`
- `npm init`
- `npm install`
- `node --jvm polyglot.js`
- Linux: `curl http://localhost:3000/` # hard to see
- Mac: `open http://localhost:3000/`



# GraaIVM DEMO LLVM

- `ls -l gzip.*`
- `clang -c -emit-llvm gzip.c`
- file `gzip.bc`
- `echo "HELLO WORLD" | lli gzip.bc -f -c - |`  
`zcat -`



# GraalVM DEMO Streams

- `cd ../streams-example`
- `/opt/graalvm-ce-1.0.0-rc10/bin/java -jar target/benchmarks.jar mapReduce -f1 -wi 4 -i4`
- `/usr/lib/jvm/java-11-openjdk-amd64/bin/java -jar target/benchmarks.jar mapReduce -f1 -wi 4 -i4`





# GraalVM DEMO JDK 11 JS

- `cd ../graal-js-jdk11-maven-demo/test`
- `mvn clean install`
- `mvn test`



# Adding some Ketchup



- So what about a complicated server ???
- Native can handle runtimes that have explicit resolution of types and dependencies
- Spring-FU project is working to create a functional API that avoids annotations, reflection, and runtime
- Working with Graal team to improve both
- <https://spring.io/blog/2018/10/02/the-evolution-of-spring-fu>



# Spring FU (kofu)

```
import org.springframework.fu.kofu.web.server
import org.springframework.fu.kofu.webApplication
import org.springframework.web.reactive.function.server.ServerRequest
import org.springframework.web.reactive.function.server.ServerResponse.ok

val app = webApplication {
    beans {
        bean<SampleService>()
        bean<SampleHandler>()
    }
    server {
        port = if (profiles.contains("test")) 8181 else 8080
        router {
            val handler = ref<SampleHandler>()
            GET("/", handler::hello)
            GET("/api", handler::json)
        }
        codecs {
            string()
            jackson()
        }
    }
}

data class Sample(val message: String)
class SampleService {
    fun generateMessage() = "Hello world!"
}

class SampleHandler(private val sampleService: SampleService) {
    fun hello(request: ServerRequest) = ok().syncBody(sampleService.generateMessage())
    fun json(request: ServerRequest) = ok().syncBody(Sample(sampleService.generateMessage()))
}

fun main() {
    app.run()
}
```



# GraaIVM DEMO kofu

- Mac shell:  
cd tmp/spring-fu/samples/kofu-reactive-minimal
- ./gradlew build
- java -jar build/libs/kofu-reactive-minimal.jar
- cat ./build.sh
- ./build.sh



# GraalVM Notes

- Static / declarative dependencies works
- Dynamic dependencies need work (logging)
- Definitely adds new scripting capabilities
- Brings a new AOT / JIT compiler to the table
- Static initialization and optimization saves in startup time
- Memory requirements are reduced



# GraalVM Notes

- Native works as long as static dependencies are identified
- Frameworks need to change how they do some things to work with Graal engine
- Spring FU is pushing to improve Graal
- Anyone can write a new scripting language and leverage the entire GraalVM ecosystem



# GraalVM Summary

- Technology to keep an eye on
- Depends on goals
  - If you are in to polyglot...
  - Alternative to Node.js
  - Multi-platform support
  - Ready for production, maybe for Twitter
- Extensible
- Flexible
- Open Source
- I get to use Kotlin more
- Enterprise Edition provides even more





# GraalVM Resources

- <https://www.graalvm.org>
- <https://www.slideshare.net/ThomasWuerthinger/2015-cgo-graal>
- <https://www.slideshare.net/ThomasWuerthinger/2014-0424-graal-modularity>
- <https://www.slideshare.net/ThomasWuerthinger/graal-truffle-ethdec2013>
- <https://medium.com/graalvm/stream-api-performance-with-graalvm-be6cfe7fbb52>
- <https://medium.com/graalvm/graalvm-ten-things-12d9111f307d>
- <https://medium.com/graalvm/under-the-hood-of-graalvm-jit-optimizations-d6e931394797>
- <https://github.com/oracle/graal>



# Questions ?

<https://github.com/lseinc/seeking-graal-cm19.git>

Don't forget to fill  
out the survey!



# Thank You !

<https://github.com/lseinc/seeking-graal-cm19.git>



David Lucas  
Lucas Software Engineering, Inc.  
[www.lse.com](http://www.lse.com)  
[ddlucas@lse.com](mailto:ddlucas@lse.com)  
[@DavidDLucas](https://twitter.com/DavidDLucas)

L  
S  
E

