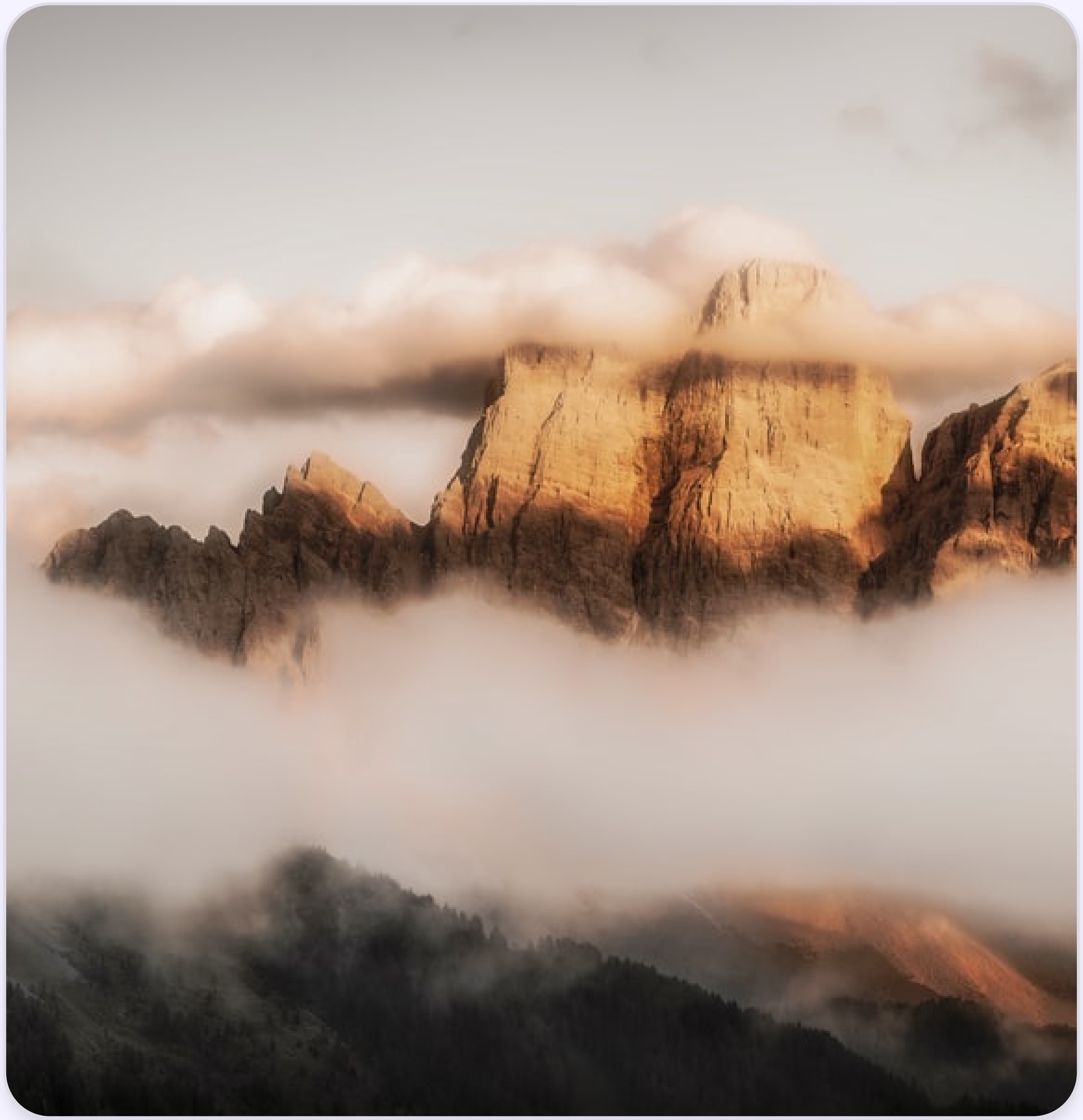
初探 GraalVM

Run Programs Faster Anywhere 🚀

Let's Begin →

Content



About GraalVM

GraalVM 前世

- 💥 起源于 Sun Micorsystems Maxine Virtual Machine(2005), 目标是用 Java 编写 Java Virtual Machine
- 🎯 希望从 C 开发的问题中解放出来, 并从元循环(Meta-circular)受益
- 🐌 当意识到到目标过于巨大后, 决定尽可能多的重用 Hotspot 运行时, 采用 Hook 的方式嵌入定制编译器

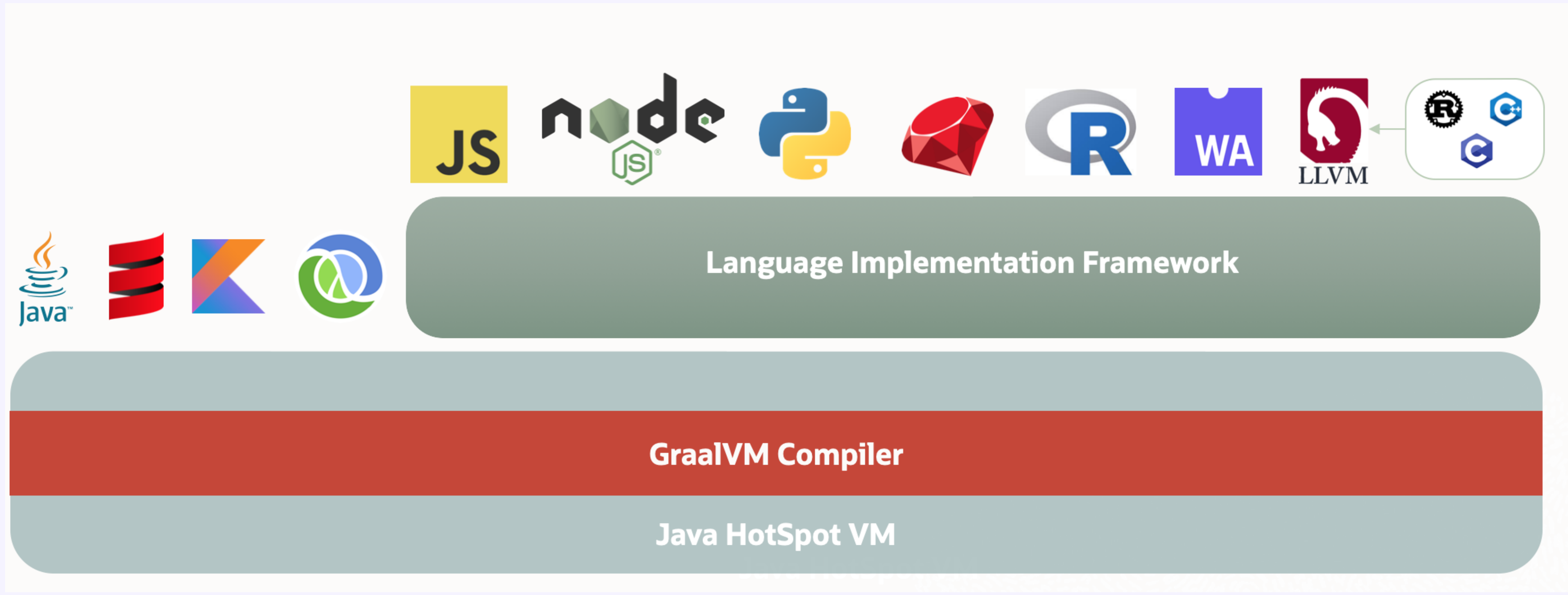
GraalVM 今生(Project Goals)

- 💰 Oracle Labs, 第一个生产就绪版本 GraalVM 19.0 已于 2019 年 5 月发布
- 🚄 一个高效的 Java 运行时, 对标 Native languages
- 🔥 为多语言设计消除语言间的隔离, 允许在单个程序中混合使用多种编程语言
- 🔉 支持多种基础环境运行: Oracle Database, OpenJDK, Nodejs, Android/iOS
- 🌈 减少应用的启动时间 (AOT), 是实现程序微服务的理想方式

What is GraalVM?

GraalVM Architecture

- 🕹️ Java HotSpot VM
- 🍭 GraalVM compiler (JIT Compiler)
- 🚉 Truffle Language Implemention Framework



Runtime Mode

- JVM Runtime Mode
- Native Image
- Java On Truffle (experimental)
  - `java -truffle [options] class`
  - `java -truffle --java.JavaHome=/path/to/java/home -version`

GraalVM Components

Available Distributions

- 社区版: GraalVM CE Base on OpenJDK
- 企业版: GraalVM EE Base on Oracle JDK
- Support JDK Version: 8/11/16

Components List

- Core Components
  - Runtimes: Java HotSpot VM, JavaScript runtime, LLVM runtime
  - Libraries: GraalVM compiler, Polyglot API
  - Utilites: GraalVM Updater
- Additional Components
  - Tools/Utilites: Native Image, Java on Truffle
  - Others Runtimes: Node.js, Python, Ruby and so on

Demo

Why GraalVM?

For Java

- 更佳的峰值性能
  - Twitter 服务性能提升 `22%`[1] (GraalVM EE)
  - 美团 `TP9999` 由 `60ms↓50ms`, 幅度 `16.7%`
- 更少的资源消耗, CPU 以及内存
  - User CPU Time `↓11%`
  - Old Gen Used `↓40MB`
  - PS Scavenge Cycles `↓2.5%~2.7%`
- 异构语言的扩展性[2]
- 构建机器可直接执行的应用程序 (Native Image)

```java
import org.graalvm.polyglot.*;
{
  Context context = Context.create("js");
  Value parse = context.eval("js", "JSON.parse");
  Value parsedValue = parse.execute("{\"foo\":\"bar\"}");
  Value memberValue = parsedValue.getMember("foo");
  System.out.println(memberValue.asString());
}
```

---

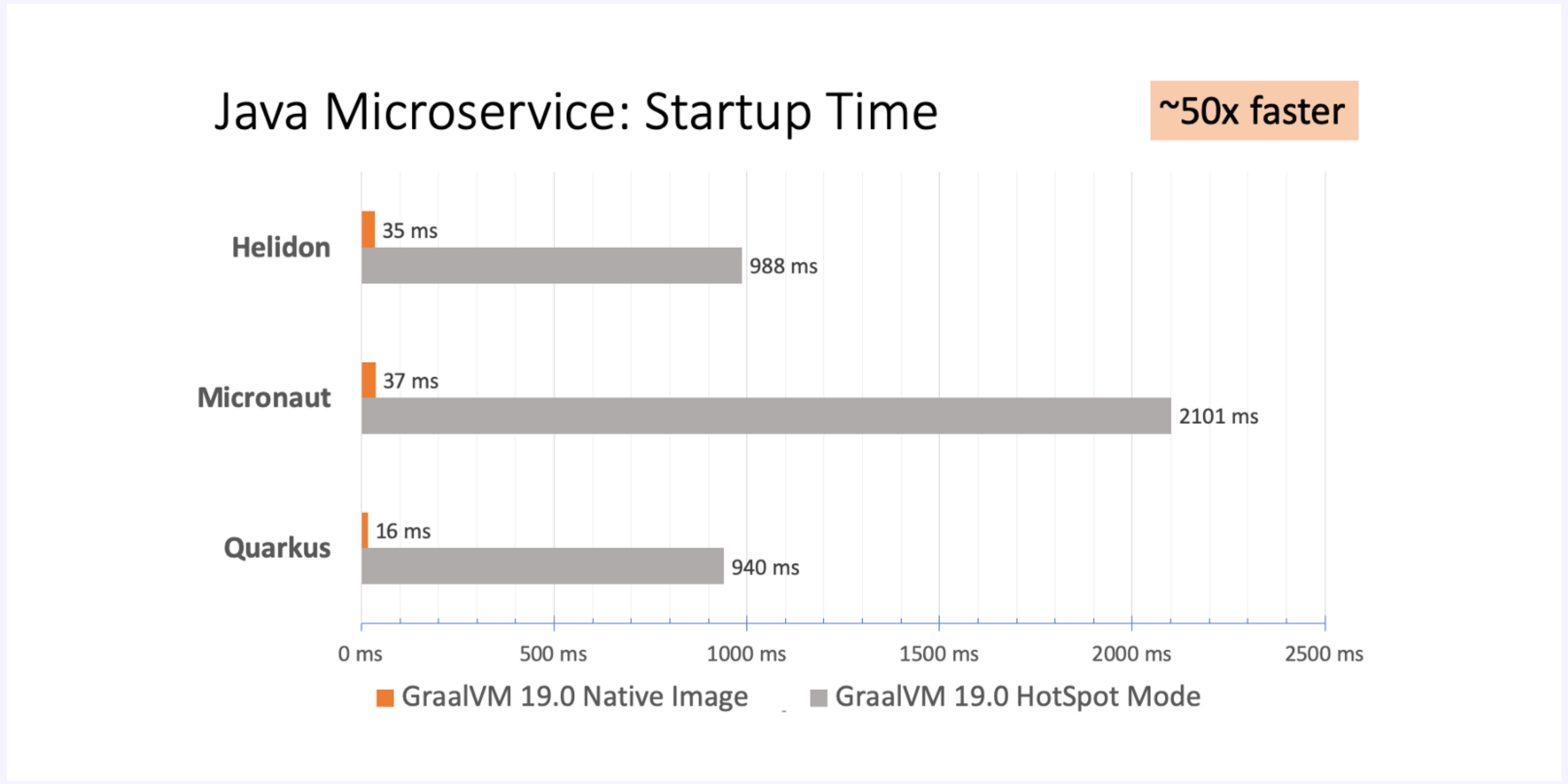
1. Twitter's quest for a wholly Graal runtime ↩

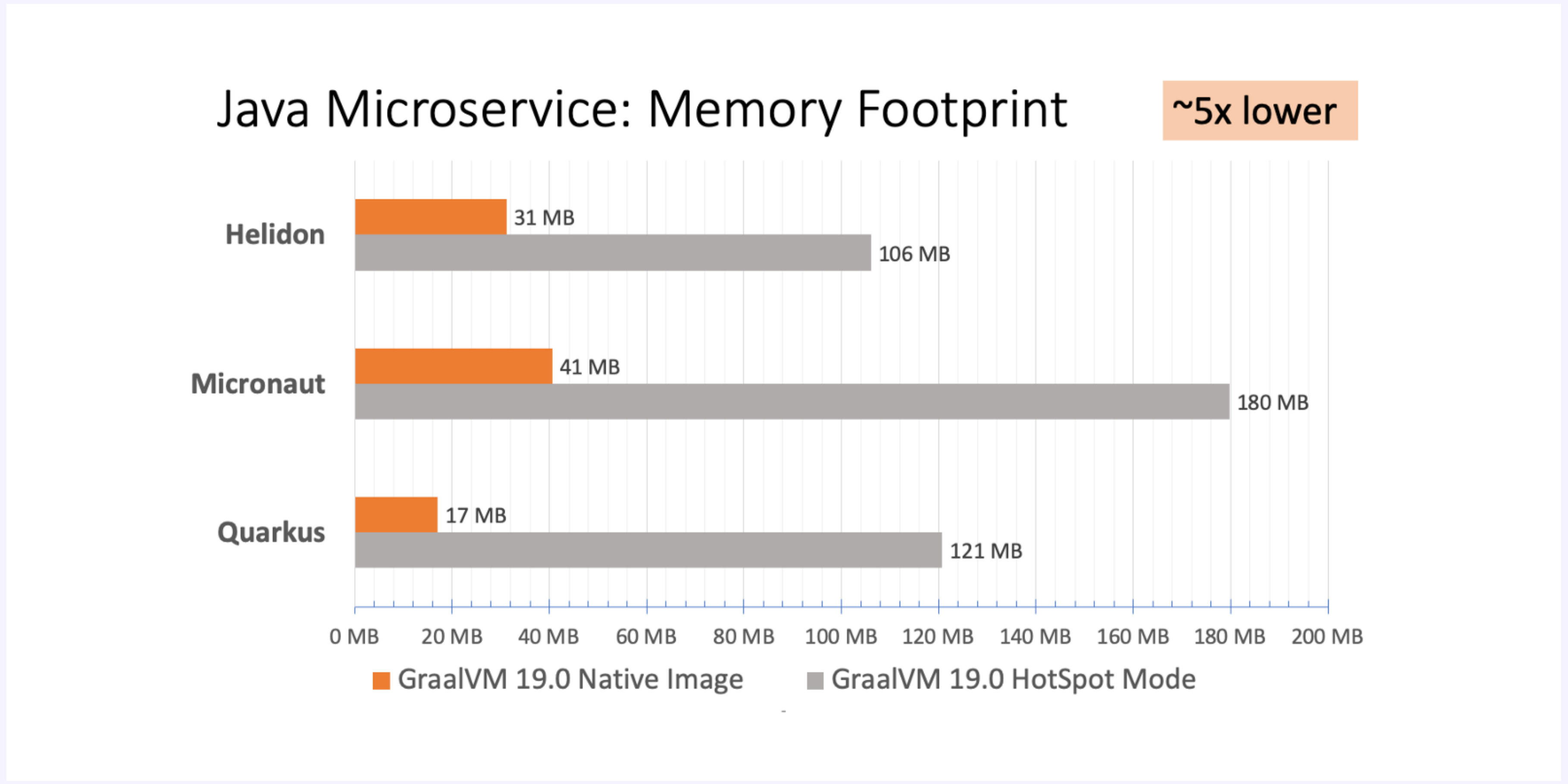
2. PrettyPrintJSON.java ↩

Why GraalVM?

For Micorservices Frameworks

- Native Image technology
- the best way for deploying cloud native applications
- accepted projects: Quarkus, Micronaut, Helidon, Spring Native (beta)





What GraalVM Compiler?

GraalVM Complier

- 纯 Java 编写的动态 JIT Compiler (transforms bytecode into machine code)
- 与 Java HotSpot VM 集成, 基于 JVMCI

GraalVM Complier Advantages

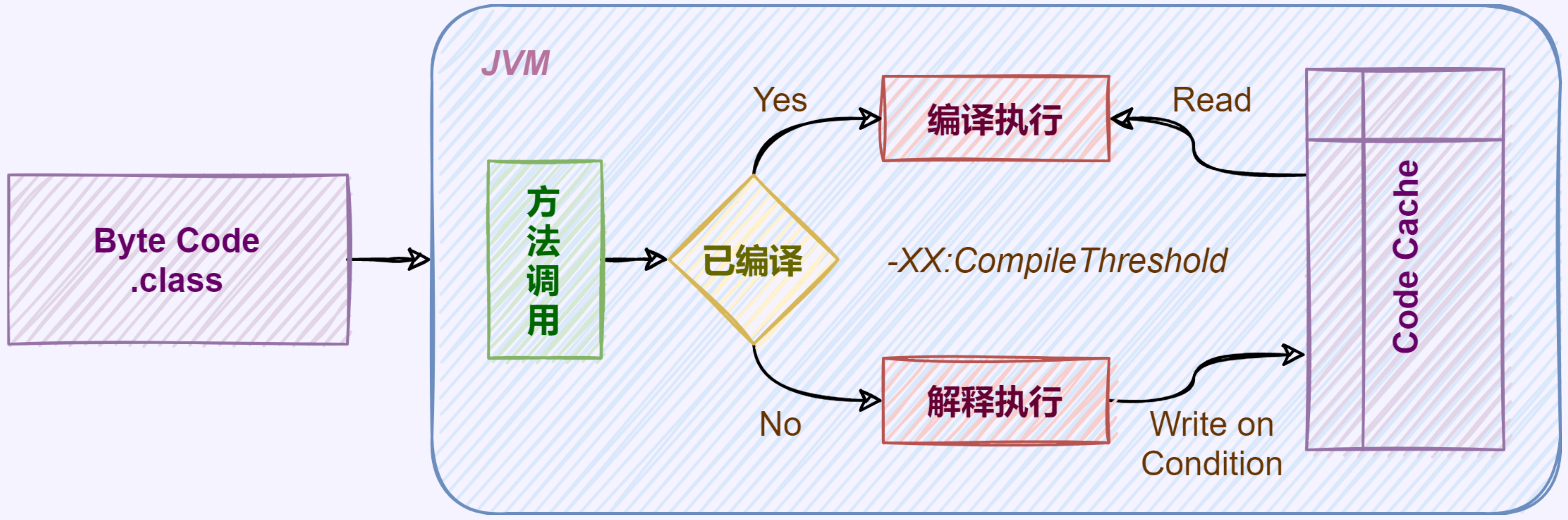
- 通过独特的代码分析和方法优化为运行在 JVM 上的程序提供性能优化
- 多种优化算法 (called Phases)[1]
  - 方法内联、部分逃逸分析、Global Value Numbering 等
  - GraalVM EE 拥有 62 优化阶段, 并拥有其中 27 项目专利
- 对新特性的优化更加友好, 如: Streams、Lambdas

1. Compiler Configuration ↩

Awesome JIT

Java JIT 触发优化的过程

- 阈值: 调用次数以及循环回边次数
  - `-XX:CompileThreshold`: 超出阈值将触发即时编译



Awesome JIT - Method Inlining

方法内联

- *将目标方法纳入编译方位之中, 取代原方法调用从而实现优化*
  - 方法调用栈: 栈帧 (包含: 局部变量表、操作数栈等)
- 热点方法(方法调用次数)
  - `-XX:MinInliningThreshold`
  - `-XX:InlineFrequencyCount`
- 被内联的方法体不宜过大(编译后的字节码)
  - `-XX:MaxInlineSize`
  - `-XX:FreqInlineSize`

```java
public class Inlining {
  public int sum(int a, int b, int c, int d) {
    return add(a, b) + add(c, d);
  }

  public int add(int x, int y) {
    return x + y;
  }
}
```

```java
public class Inlining {
  public int sum(int a, int b, int c, int d) {
    return a + b + c + d;
  }

  public int add(int x, int y) {
    return x + y;
  }
}
```

Awesome JIT - Partial Escape Analysis

Scalar replacement[1]

```java
public class ScalarReplacement {
  int x, y;
  boolean flag = randomBool();

  public void split(Blackhole blackhole) {
    for (int i = 0; i < 300; i++) {
      Value o = flag ? new Value(x) : new Value(y);
      blackhole.consume(o.x);
    }
  }
}
public class Value{
  public int x;
  Value(int x) {
    this.x = x;
  }
}
```

```java
public class ScalarReplacement {
  int x, y;
  boolean flag = randomBool();

  public void split(Blackhole blackhole) {
    for (int i = 0; i < 300; i++) {
      int val = flag ? x : y;
      blackhole.consume(val);
    }
  }
}
public class Value{
  public int x;
  Value(int x) {
    this.x = x;
  }
}
```

1.ScalarReplacement Escape Analysis Details ↩

C2 VS Graal

Compiler	C2	Graal
Languages	C++	Java
For Developer Learning	学习曲线陡峭	模块化设计便于理解
Streams and Lambdas	不友好	Java 编写，更优的支持
Method Inlining	一般	激进的方法内联，更佳的峰值性能
Escape Analysis	不支持带控制流的逃逸分析	部分逃逸分析，支持带控制流的逃逸分析
New Major Releases	无	Java On Truffle, Native Image
Performance[1]	-	Better then C2, Low memory and CPU

1.Graal vs. C2: Battle of the JITs ↩

Native Image

Install Native Image

- GraalVM Updater
  - `` `gu install native-image` ``

Ahead-of-time Compilation

- *Native Image*
- translates Java and JVM-base code into native platform executable
- instant start, smaller, consume less resources
- ideal for cloud deployments and microservices

Demo

Native Reflection

Reflection

- Automatic Detection
    - `Class.forName(String)`
    - `Class.getField(String)`
- `-H:ReflectionConfigurationResources`

reflect-config.json

```json
[
  {
    "name": "Greeting",
    "allDeclaredFields": true,
    "allDeclaredConstructors": true,
    "allDeclaredMethods": true
  }
]
```

```java
public class ReflectionExample {
  public static void main(String[] args) throws Exception {
    Class<?> clazz = Class.forName("Greeting");
    Method method = clazz.getDeclaredMethod("sayHi", String.class);
    Object instance = clazz.getDeclaredConstructors()[0].newInstance();
    Object result = method.invoke(instance, "wuxin");
    System.out.println(result);
  }
}

class Greeting {
  public String sayHi(String name) {
    return String.format("Hi %s!", name);
  }
}
```

Spring Native (beta)

Dependency

- `spring-native:${version}`
- `spring-aot-maven-plugin:${version}`
- `spring-boot-maven-plugin:${version}`

Let’s you library support spring-native

- `spring-aot:${version}`
- `NativeConfiguration`

Supports

Annotations	Configuration File	说明
`@ResourceHint`	`resource-config.json`	资源相关
`@SerializationHint`	`serialization-config.json`	Java 序列化
`@TypeHint`, `@FieldHint`, `@MethodHint`	`relect-config.json`	反射相关
`@JdkProxyHint`, `@AopProxyHint`	`proxy-config.json`	代理相关

GraalVM is Production Ready?

Feature	Linux AMD64	Linux ARM64	MacOS	Windows
Native Image	stable	experimental	stable	experimental
LLVM runtime	stable	experimental	stable	not available
LLVM toolchain	stable	experimental	stable	not available
JavaScript	stable	experimental	stable	experimental
Node.js	stable	experimental	stable	experimental
Java on Truffle	experimental	not available	experimental	experimental
Python	experimental	not available	experimental	not available

Read more about GraalVM Community Edition 21

Reference

Documents

- ⭐ Java 即时编译器原理解析及实践: https://tech.meituan.com/2020/10/22/java-jit-practice-in-meituan.html
- JVM Compiler Interface(JVMCI): https://openjdk.java.net/jeps/243
- Partial Escape Analysis(PEA): http://www.ssw.uni-linz.ac.at/Research/Papers/Stadler14/Stadler2014-CGO-PEA.pdf
- Graal Vs C2: https://martijndwars.nl/2020/02/24/graal-vs-c2.html
- GraalVM at Facebook: https://medium.com/graalvm/graalvm-at-facebook-af09338ac519
- Twitter's quest for a wholly Graal runtime: https://www.youtube.com/watch?v=pR5NDkIZBOA
- JITWatch: https://github.com/AdoptOpenJDK/jitwatch

Thanks✨

愿作点亮你思绪的花火