OpenJDK 对于 RISC-V 的支持现状以及路线图

报告人：张定立

1 为什么需要移植JVM?

2 RISC-V 移植路线图
- ***OpenJDK/Hotspot：Zero-Assembler Project***
- ***JIKES RESEARCH VM***
- ***OpenJDK/OpenJ9***

3 目前在做 OpenJDK 往 RISCV 上移植的团队/个人

PART 1 为什么需要移植JVM?

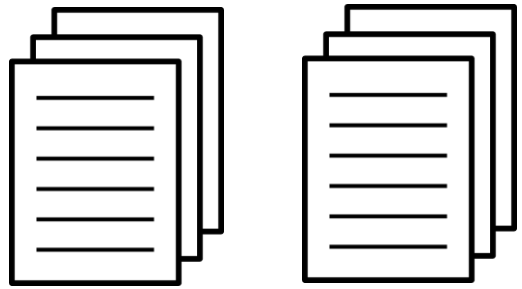
OpenJDK /Hotspot JVM

High -performance production JVM

运行Java应用，比如Hadoop，Apache Spark（目标为移植高性能、高可用型生产型JVM）

- OpenJDK/Hotspot JVM

Jikes Research VM



Easy -to -modify research JVM

学术研究，例如为托管语言提供硬件支持（目标为移植易修改的研究型JVM）

- Jike RVM（Jikes研究虚拟机）

PART 1 为什么需要移植JVM?

RISC-V Git仓库-关于Java的software列表

Name	Links	License	Maintainers
Maxine VM (Java Virtual Machine)	Upstream	GPLv2	Maxine team
Jikes RVM (Java Virtual Machine)	Upstream	Eclipse Public License (EPL)	Martin Maas (University of California, Berkeley)
OpenJDK/HotSpot (Java Virtual Machine)	?	?	Alexey Baturo, Michael Knysnek, Martin Maas
OpenJDK/OpenJ9 (Java Virtual Machine)	Upstream	Eclipse Public License 2.0 (EPLv2) with ClassPath Exception & Apache 2.0	Cheng Jin

参考资料:
[1] https://github.com/riscv/riscv-software-list

PART 2　RISC-V 移植路线图

两个初步想法：

- **Sean Halle（from Intensivate）：**

Just modify the backend code generator.
Start with the MIPS backend and modify it into RISC-V.

- **Andrew Haley（Java Platform Lead Engineer From Red Hat）：**

Two engineers with deep knowledge of HotSpot could get a bare-bones port done in a year, one doing the assembler, C1, and template interpreter, and the other doing C2. Both would work on the shared runtime.

参考资料：
[1] https://groups.google.com/a/groups.riscv.org/forum/#!msg/sw-dev/cxsJpen-caA/6Kv73ZuDAgAJ

PART 2　RISC-V 移植路线图

一、OpenJDK/Hotspot：Zero-Assembler Project

OpenJDK Releases

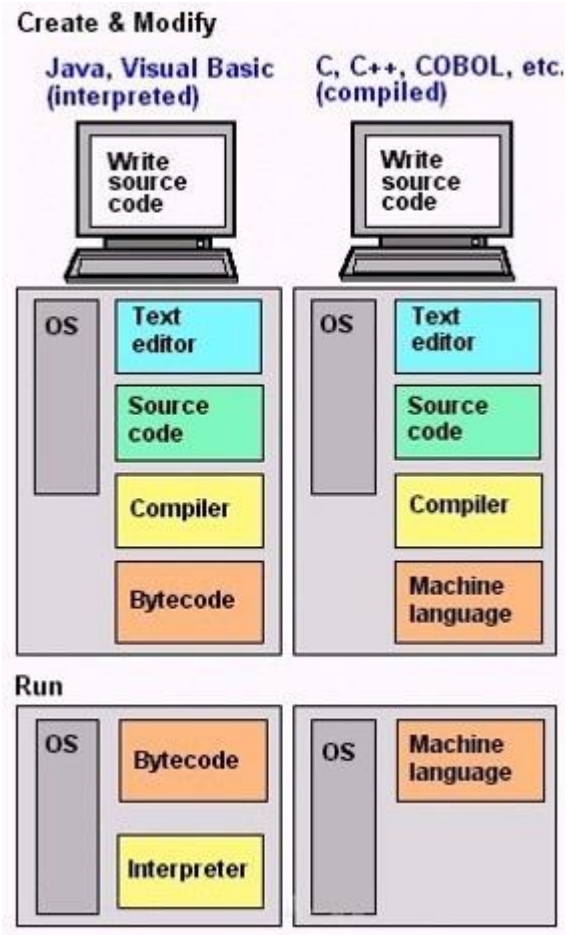
- 10 (GA 2018/03/20)
- 11 (GA 2018/09/25)
- 12 (GA 2019/03/19)
- 13 (GA 2019/09/17)
- 14 (GA 2020/03/17)
- 15 (in development)
- 16 (in development)

Zero-Assembler Project

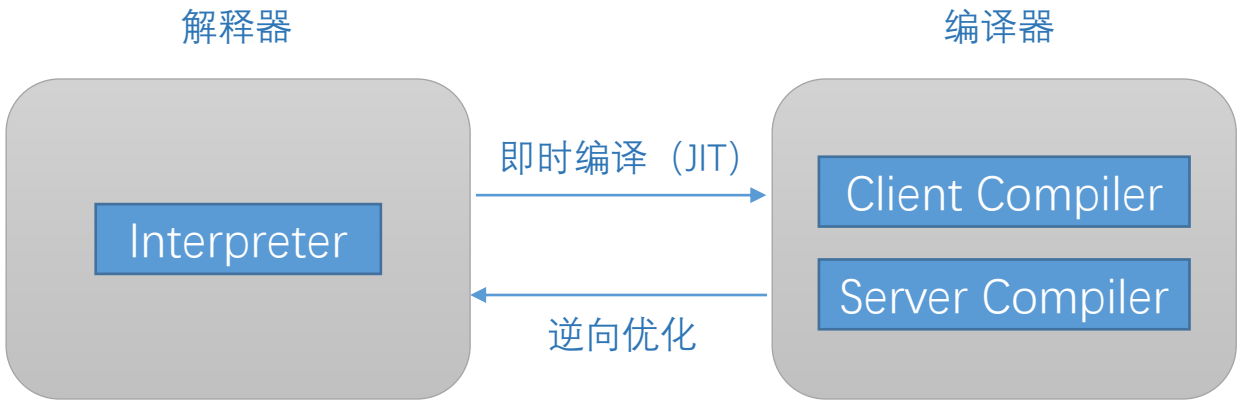
- Zero是一种OpenJDK的无汇编移植，因此可以轻易的移植到其他系统之上。这个项目的目标是能够不用增加过多额外的工作来在不同的架构上构建TCK-compliant的OpenJDK。Zero的解释器部分现在已经可以在PowerPC (32- and 64-bit), x86-64, IA-64, ARM and zSeries等平台上运行
- Zero本身没有JIT，而基于LLVM 的Shark可以为它实现JIT。

PART 2　RISC-V 移植路线图

一、OpenJDK/Hotspot：Zero-Assembler Project



JIT：Just In Time Compiler，即时编译器，这是是针对解释型语言而言的，而且并非虚拟机必须，是一种优化手段，Java虚拟机标准对JIT的存在没有作出任何规范，所以这是虚拟机实现的自定义优化技术。



参考资料：

[1] https://blog.csdn.net/qq_36627886/article/details/80402959

PART 2　RISC-V 移植路线图

一、OpenJDK/Hotspot：Zero-Assembler Project

HotSpot虚拟机

HotSpot基本上是用C ++编写的，但它的最关键的代码中约10,000行是用汇编语言编写的，所以为了使OpenJDK真正具有可移植性，很明显需要替换此部分核心汇编语言。

默认情况下，HotSpot以**混合模式***运行*。hotspot默认使用解释+编译混合(mixed mode)的方式执行代码，它首先使用模板解释器对字节码进行解释，当发现一段代码是热点的时候，就使用JIT进行优化编译再执行，这也是它的名字“热点”(hotspot)的由来。

```
linux@linux-virtual-machine:/opt/jdk-13.0.2/bin$ ./java -Xint -version
openjdk version "13.0.2" 2020-01-14
OpenJDK Runtime Environment (build 13.0.2+8)
OpenJDK 64-Bit Server VM (build 13.0.2+8, interpreted mode, sharing)
linux@linux-virtual-machine:/opt/jdk-13.0.2/bin$ ./java -Xcomp -version
openjdk version "13.0.2" 2020-01-14
OpenJDK Runtime Environment (build 13.0.2+8)
OpenJDK 64-Bit Server VM (build 13.0.2+8, compiled mode, sharing)
linux@linux-virtual-machine:/opt/jdk-13.0.2/bin$ ./java  -version
openjdk version "13.0.2" 2020-01-14
OpenJDK Runtime Environment (build 13.0.2+8)
OpenJDK 64-Bit Server VM (build 13.0.2+8, mixed mode, sharing)
```

参考资料：

[1] https://community.oracle.com/docs/DOC-983740

PART 2　RISC-V 移植路线图

一、OpenJDK/Hotspot：Zero-Assembler Project

C1/C2 & 解释模式/编译模式

```
public class LoopTest{
    public static int j=0;
    public static void main(String[] args) {
        long start = System.nanoTime()/1000000L;
        spendTime();
        long end = System.nanoTime()/1000000L;
        System.out.println("Java代码运行时间: "+String.valueOf(end-start)+"ms");
    }
    private static void spendTime(){
        //int j=0;  //局部变量会被优化，效果见图2
        for (int i =500000000;i>0;i--) {
            j++;    //如果是空循环的话在编译模式下会被优化，见图3
        }
        System.out.println(System.getProperty("java.vm.name")); //获取JVM名字和类型
        System.out.println(System.getProperty("java.vm.info")); //获取JVM的工作模式
    }
}
```

循环测试代码

-Xint代表解释模式(interpreted mode)，会强制JVM以解释方式执行所有的字节码，当然这会降低运行速度，通常低10倍或更多。

-Xcomp代表编译模式(compiled mode)，与（-Xint）正好相反，JVM在第一次使用时会把所有的字节码编译成本地代码，从而带来最大程度的优化。

```
linux@linux-virtual-machine:~/code/test$
OpenJDK 64-Bit Server VM
interpreted mode, sharing
Java代码运行时间: 7172ms
linux@linux-virtual-machine:~/code/test$
OpenJDK 64-Bit Server VM
interpreted mode, sharing
Java代码运行时间: 7495ms
linux@linux-virtual-machine:~/code/test$
OpenJDK 64-Bit Server VM
interpreted mode, sharing
Java代码运行时间: 6996ms
```

-Xint 运行结果

```
linux@linux-virtual-machine:~/code/test$
OpenJDK 64-Bit Server VM
compiled mode, sharing
Java代码运行时间: 35ms
linux@linux-virtual-machine:~/code/test$
OpenJDK 64-Bit Server VM
compiled mode, sharing
Java代码运行时间: 33ms
linux@linux-virtual-machine:~/code/test$
OpenJDK 64-Bit Server VM
compiled mode, sharing
Java代码运行时间: 34ms
```

-Xcomp 运行结果

PART 2　RISC-V 移植路线图

一、OpenJDK/Hotspot：Zero-Assembler Project

解释器
HotSpot实际上包含两个解释器：模板解释器和C ++解释器，精简汇编语言层支持C ++代码。

堆栈操作
HotSpot本身就是一个应用程序，它具有自己的堆栈，单独的C和C ++函数在其中存储自己的调用者信息和其他数据。该堆栈的格式是特定于CPU和OS的，其布局被定义为特定平台的应用程序二进制接口（ABI）的一部分。

Shark
使用与HotSpot特定于平台的编译器相同的接口，因此无需改动HotSpot即可接入。在混合模式下运行时，HotSpot的编译器调度程序会定位热方法并调用Shark对其进行编译。Shark将这些方法的Java字节码转换为LLVM IR，并调用LLVM的JIT生成本机代码。然后将本机代码安装在VM中，以替换该方法的解释版本，然后控制权返回到编译器调度程序。

参考资料：
[1] https://community.oracle.com/docs/DOC983740

PART 2　RISC-V 移植路线图

一、OpenJDK/Hotspot：Zero-Assembler Project

OpenJDK 6开始包含了ZERO模块，但默认是安装Server VM，需要自己编译安装。

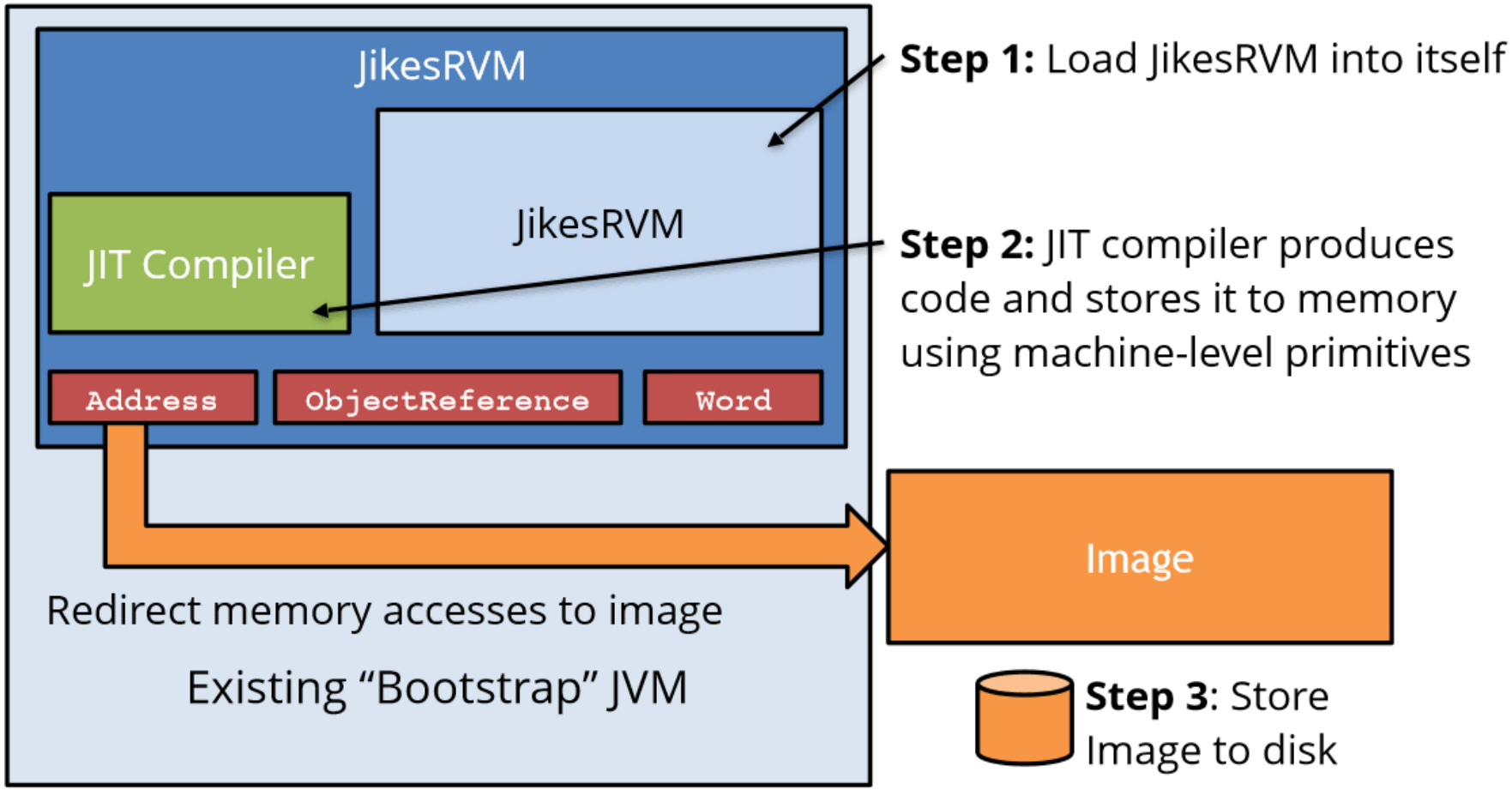
```
bash configure \
--with-debug-level=slowdebug \  #默认值为release,如果我们要调试的话，设定为fastdebug或者slowdebug
--enable-dtrace \               #开启一个性能工具
--with-jvm-features=zero \      #设置使用 c++ 实现的 zero 解释器
--with-jvm-variants=zero \
--with-target-bits=64           #指定编译64位的虚拟机

make all
```

```
linux@linux-virtual-machine:~/Desktop/linux-x86_64-zero-slowdebug/jdk/bin$ ./java -version
openjdk version "14-internal" 2020-03-17
OpenJDK Runtime Environment (slowdebug build 14-internal+0-adhoc.linux.jdk14-6c954123ee8d)
OpenJDK 64-Bit Zero VM (slowdebug build 14-internal+0-adhoc.linux.jdk14-6c954123ee8d, interpreted mode)
linux@linux-virtual-machine:~/Desktop/linux-x86_64-zero-slowdebug/jdk/bin$ ./java /home/linux/code/test/LoopTest.java
81866
linux@linux-virtual-machine:~/Desktop/linux-x86_64-zero-slowdebug/jdk/bin$ ./java -Xcomp -version
openjdk version "14-internal" 2020-03-17
OpenJDK Runtime Environment (slowdebug build 14-internal+0-adhoc.linux.jdk14-6c954123ee8d)
OpenJDK 64-Bit Zero VM (slowdebug build 14-internal+0-adhoc.linux.jdk14-6c954123ee8d, interpreted mode)
```

PART 2　RISC-V 移植路线图

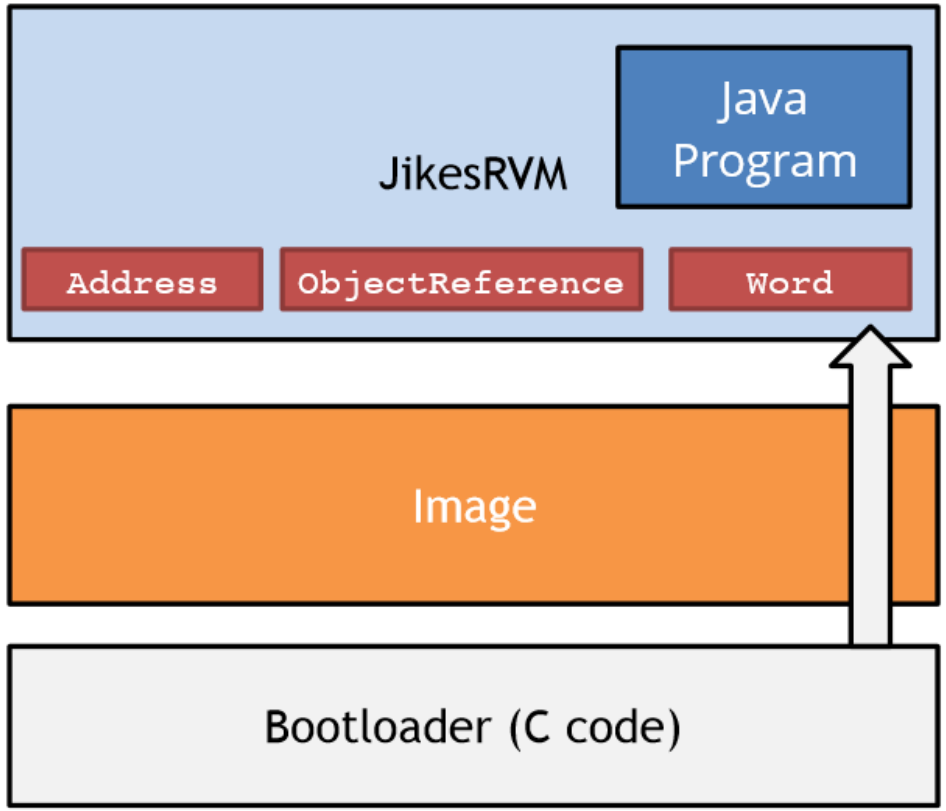
二、JIKES RESEARCH VM-运行示意



参考资料：
[1] https://riscv.org/2017/12/7th-risc-v-workshop-proceedings/

PART 2　RISC-V 移植路线图

二、JIKES RESEARCH VM-运行示意无bootstrap JVM



In JIT-generated code, primitives map to actual operations

Resume at “boot” function.

Loads boot image into memory implements native calls, fault handlers, etc.

注：Maxine VM也是用java来实现java vm，（java in java）达成“元循环”的思想，内部没有编译器只能依赖解模式执行，性能问题未得到解决。

参考资料：
[1] https://riscv.org/2017/12/7th-risc-v-workshop-proceedings/

PART 2　RISC-V 移植路线图

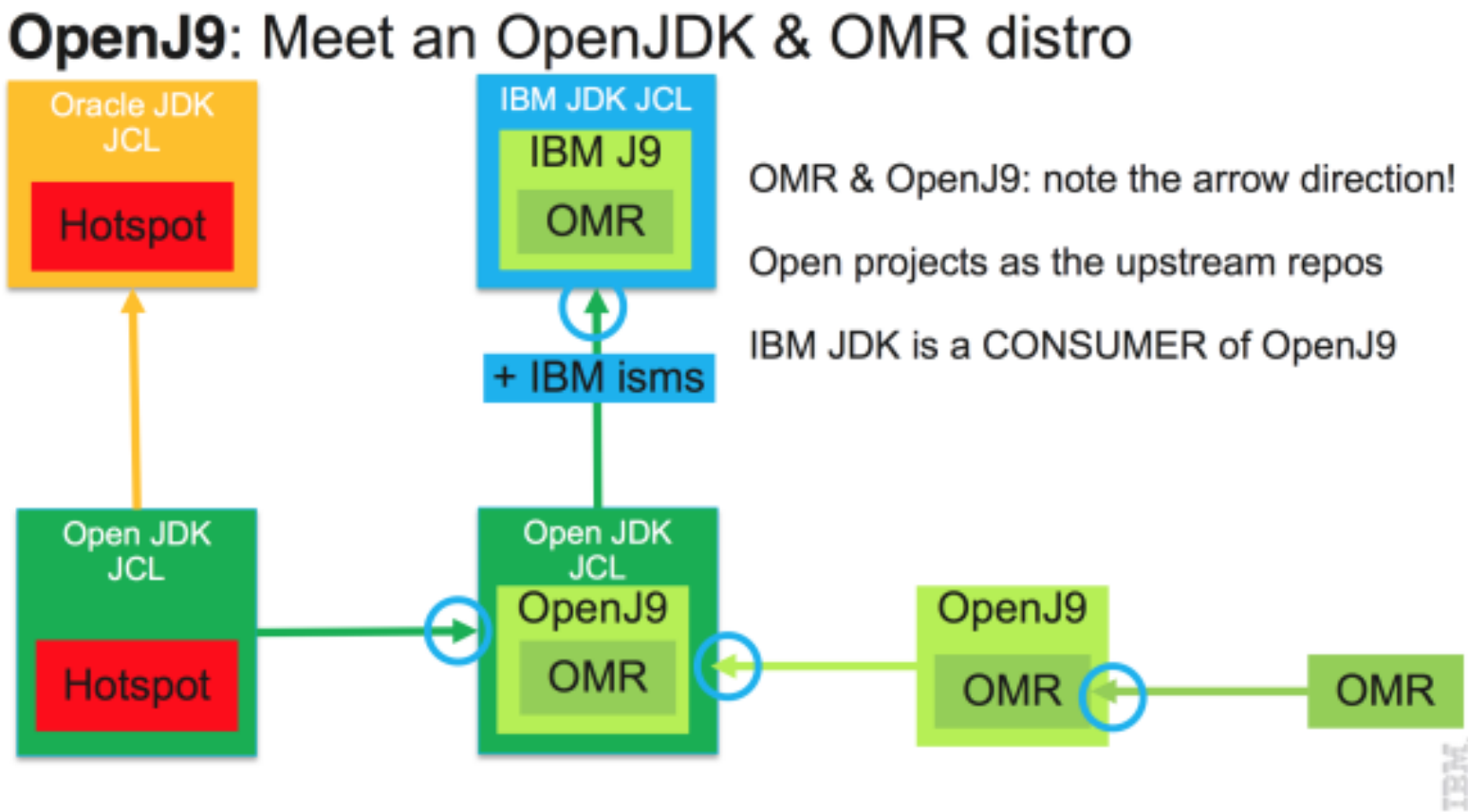
Michael A. Knyszek & Martin Maas的移植路线图

1. completed the interpreter portion
2. Baseline JIT and C1 port to RISC-V
3. porting HotSpot's C2 "Server" Optimizing Just-In-Time compiler to RISC-V

参考资料：
[1] https://riscv.org/2017/12/7th-risc-v-workshop-proceedings/

PART 2　RISC-V 移植路线图

三、OpenJDK/OpenJ9

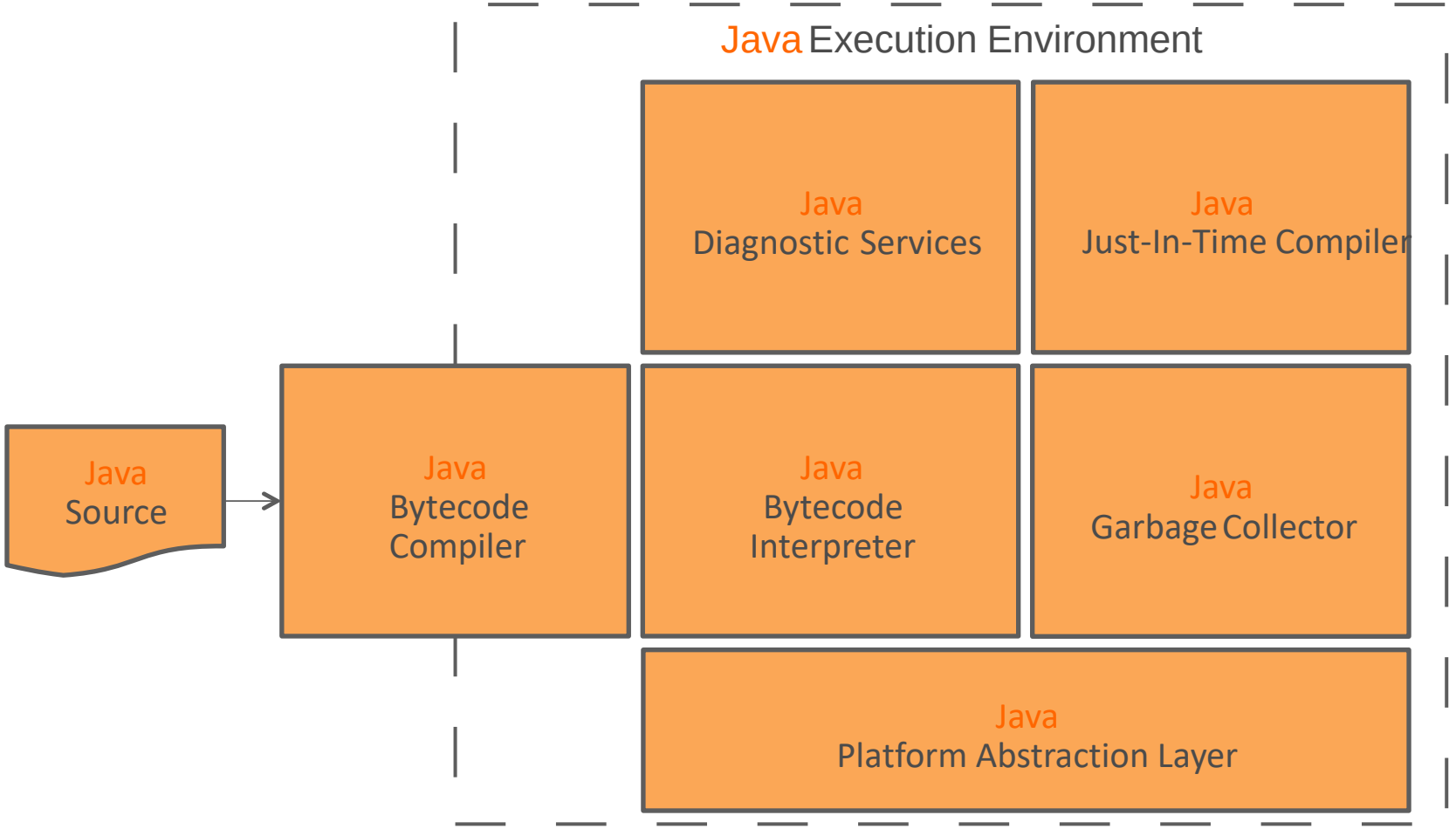


参考资料：

[1] https://github.com/eclipse/openj9

PART 2　RISC-V 移植路线图

三、**OpenJDK/OpenJ9-** Java Language Runtime

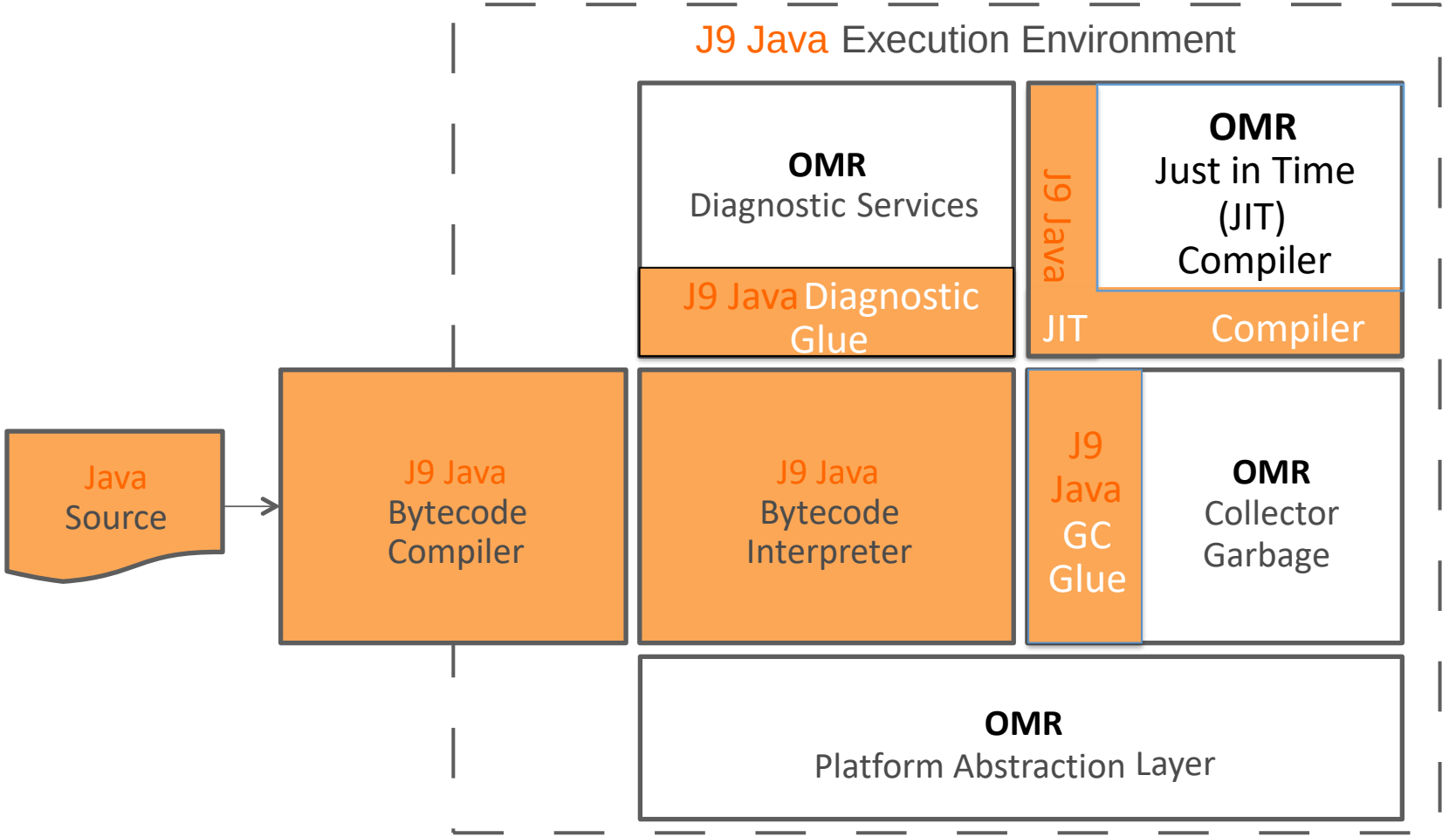


参考资料：

[1] https://www.slideshare.net/MarkStoodley/omr-a-modern-toolkit-for-building-language-runtimes

PART 2　RISC-V 移植路线图

三、OpenJDK/OpenJ9- J9中OMR与Java核心模块的组合



参考资料：

[1] https://www.slideshare.net/MarkStoodley/omr-a-modern-toolkit-for-building-language-runtimes

PART 2　RISC-V 移植路线图

三、OpenJDK/OpenJ9- OpenJ9移植路线图

- 在Linux上编译没有JIT的构建（更改所有请求的配置/设置以使其正常运行），所有更改将通过riscv-unknown-linux-gnu-gcc添加到编译中。
- 将用于RISC-V支持的FFI代码（libffi）导入OpenJ9
- 使用riscv-unknown-linux-gnu-gcc编译OpenJDK11 + OpenJ9 + OMR构建。
- 在主机上通过模拟器（例如QEMU）测试运行。

参考资料：
[1] https://github.com/eclipse/openj9/issues/5058

PART 3 目前在做 OpenJDK 往 RISCV 上移植的团队/个人

Name	Job	Base	Status
Edward Nevill	OpenJDK committer	ZERO	在OpenJDK官网上发起了一个为Zero添加RISC-V支持的issue
Michael A. Knyszek	Google	ZERO	基于ZERO后端的移植，但是还没有完成高性能的JIT编译移植
Cheng Jin	IBM OpenJ9 Software Developer	OpenJDK11 + OpenJ9 + OMR (without JIT)	通过简单的测试，编译了32位Linux交叉编译器，但编译器可能存在问题，正在添加64位支持