OSDT China
Beijing 2018

GraalVM
— One VM to Rule Them All

Feng Li (李枫)
hkli2013@126.com
Dec 8，2018

Agenda

I. GraalVM
- Overview
- ARM

II. Dive Into GraalVM
- Truffle & Graal
- Sulong
- SVM

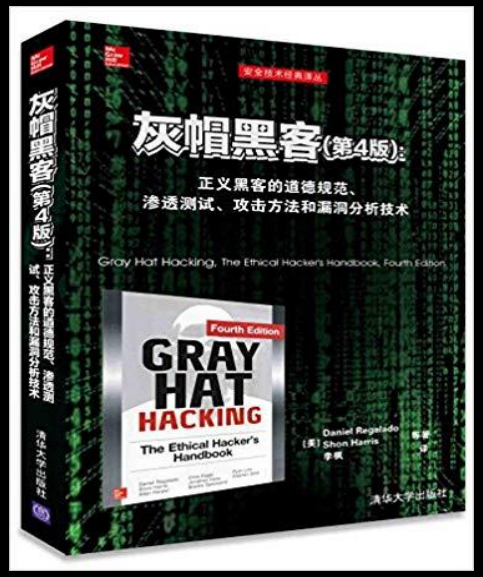
III. GraalVM on ARM
- Development Env
- My Practice

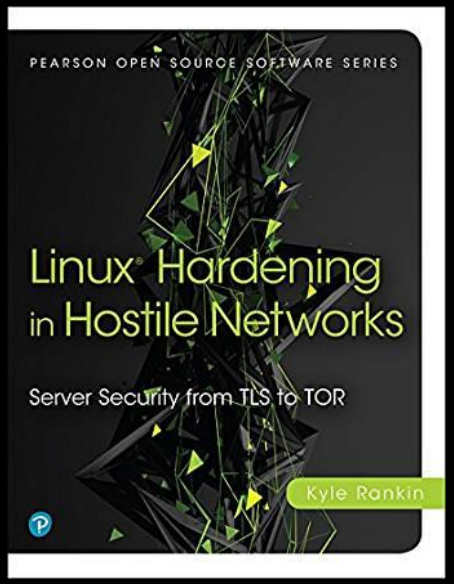
IV. Runtime
- Rethinking
- Redesign

V. Wrap-up

Who Am I

- The main translator of the book «Gray Hat Hacking The Ethical Hacker's Handbook, Fourth Edition» (ISBN：9787302428671) & «Linux Hardening in Hostile Networks, First Edition»



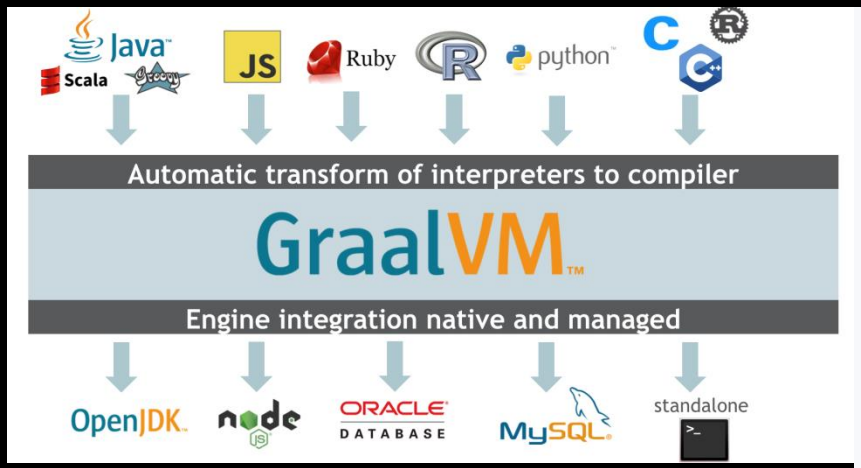


- Gave presentations at more than 10 technology conferences by now, which covers Languages, Cloud, Linux…
- Next, plan to participate in more technology conferences on DataCenter, OS, Toolchain, Security, Blockchain, AI, Chip/EDA…

I. GraalVM

1) Overview

- https://www.graalvm.org/



Community Edition (CE)

GraalVM CE is available for free for development and production use. It is built from the GraalVM sources available on GitHub. We provide pre-built binaries for GraalVM CE for Linux and Mac OS X on x86 64-bit systems.

DOWNLOAD FROM GITHUB

LICENSE

- All open-source
- Free for production use

BENEFITS

- Open-source license
- Free community support via public channels
- Bugfixes and security updates

AVAILABLE FOR

Mac Linux

Enterprise Edition (EE)

GraalVM EE provides additional performance, security, and scalability relevant for running critical applications in production. It is free for evaluation uses and available for download from the Oracle Technology Network. We provide binaries for GraalVM EE for Linux or Mac OS X on x86 64-bit systems.

DOWNLOAD FROM OTN

LICENSE

- Free for evaluation and other non-production uses
- Please contact us for commercial use and support options

BENEFITS

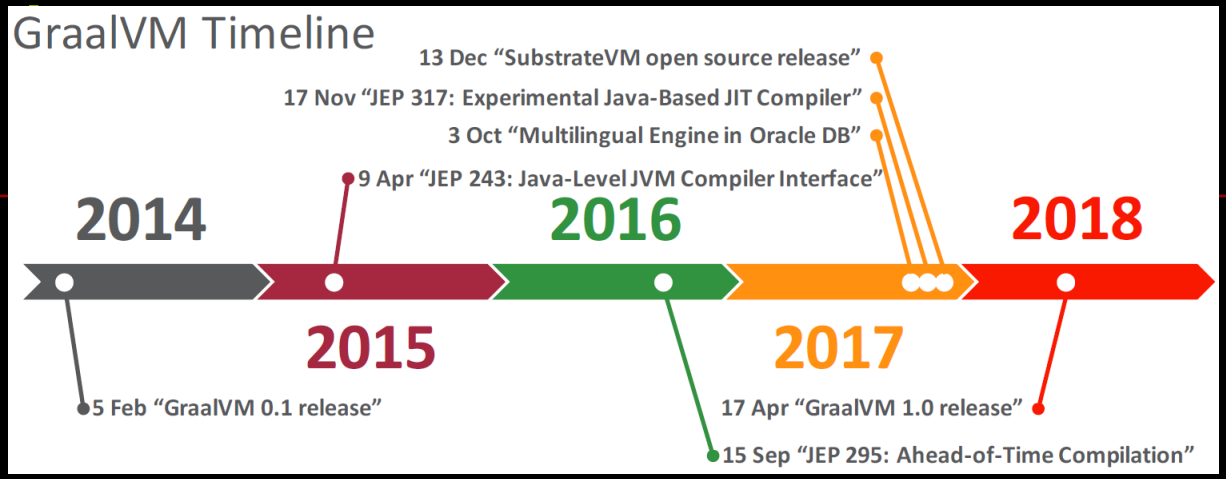
- Improved performance and smaller footprint
- Sandboxing capabilities for native code
- Commercial support options

AVAILABLE FOR

Mac Linux

- High-Performance Polyglot VM
- A meta-runtime for Language-Level Virtualization
- Currently base an Oracle Labs JDK 8 with JVMCI support
- http://www.graalvm.org/docs/reference-manual/

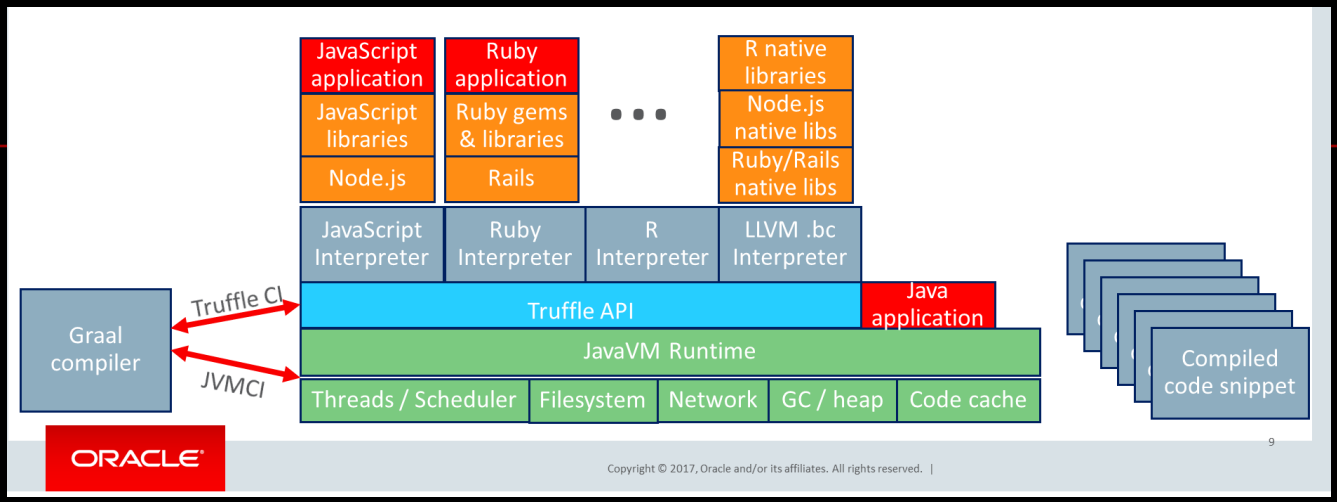
Timeline



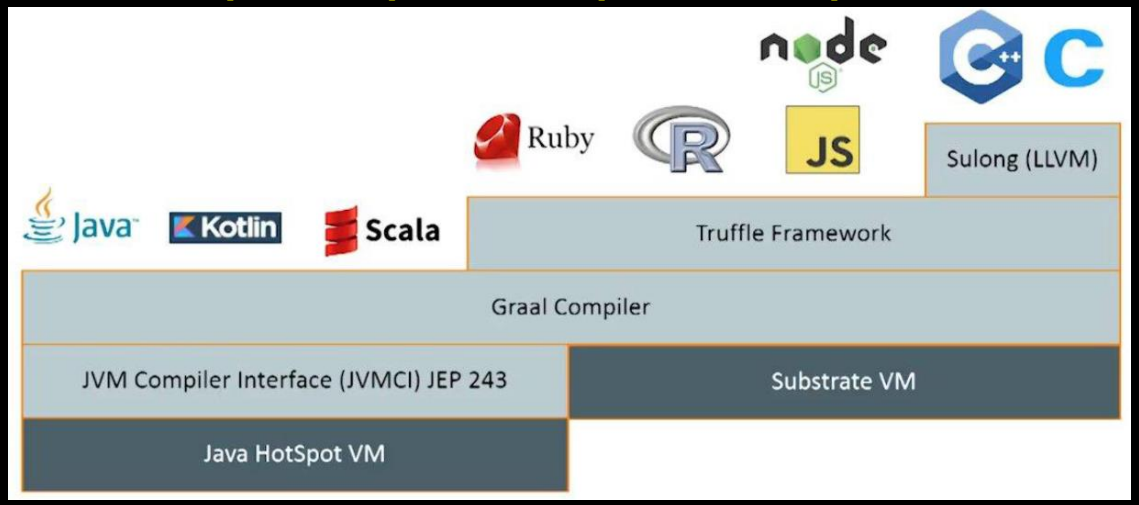
Source: https://doc.huodongjia.com/detail-7162.html

Arch

- A hybrid of static & dynamic runtimes



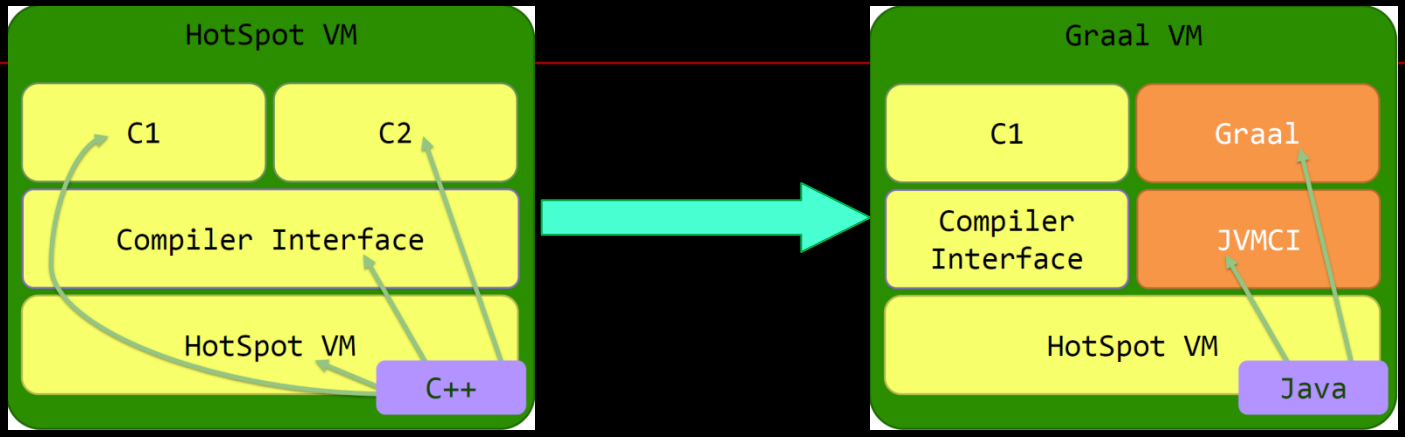
Source: https://ics.psu.edu/wp-content/uploads/2017/02/GraalVM-PSU.pptx



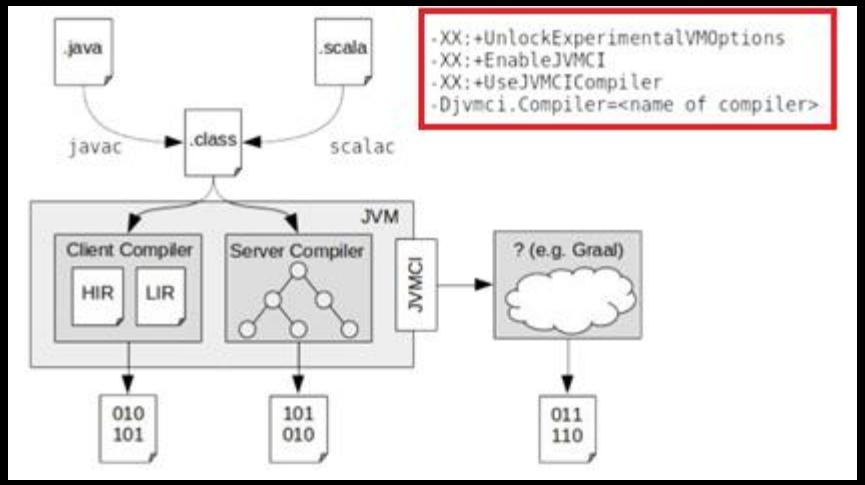
Source: "GraalVM and MicroProfile", Code One 2018

JVMCI

- Java-Level JVM Compiler Interface
- http://openjdk.java.net/jeps/243: experimental in JDK 9
-

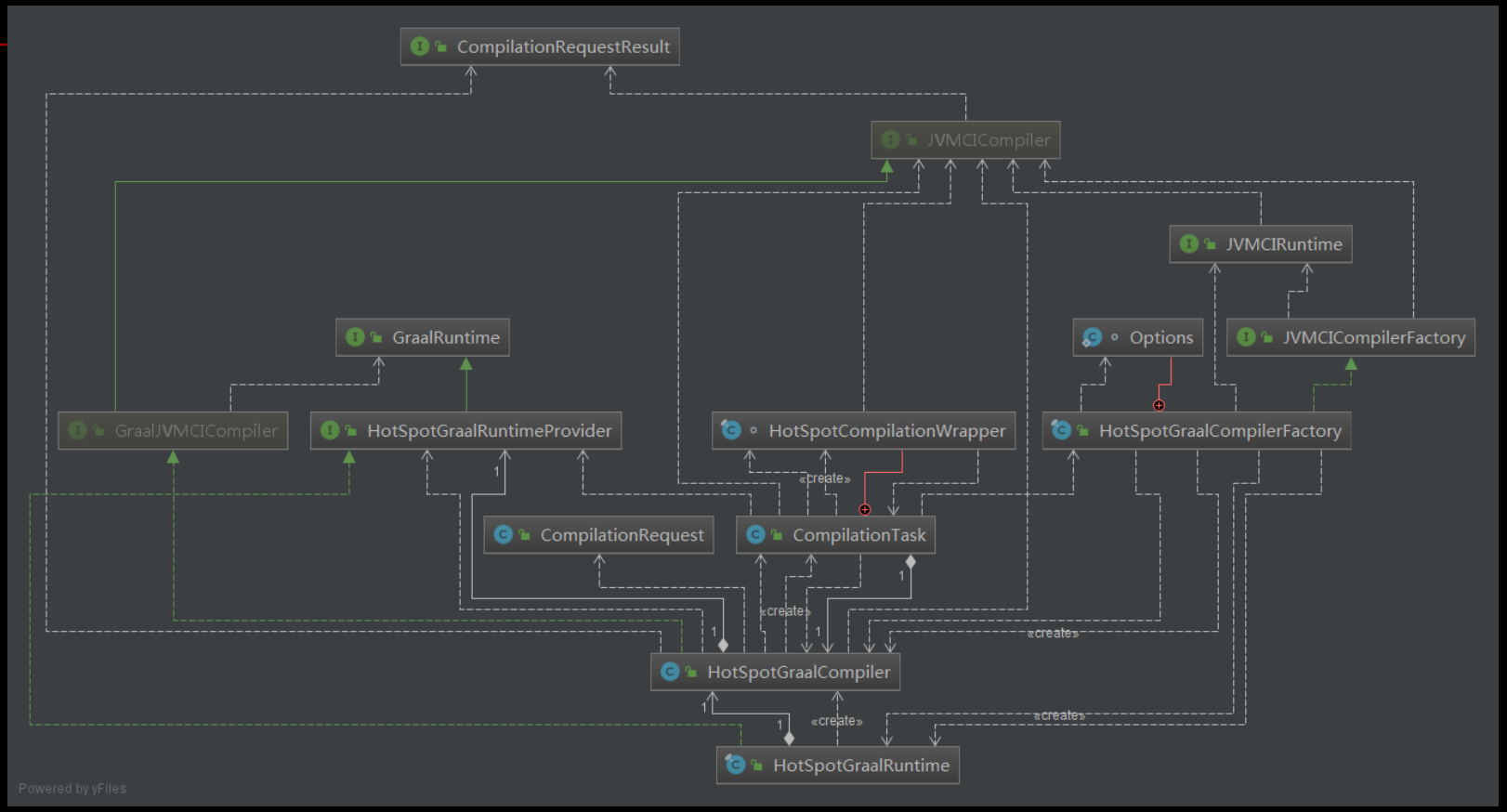


Source: https://www.slideshare.net/jyukutyo/jvmgraalopenj9



Source: https://www.dynatrace.com/news/blog/new-ways-introducing-compiled-code-java-9/

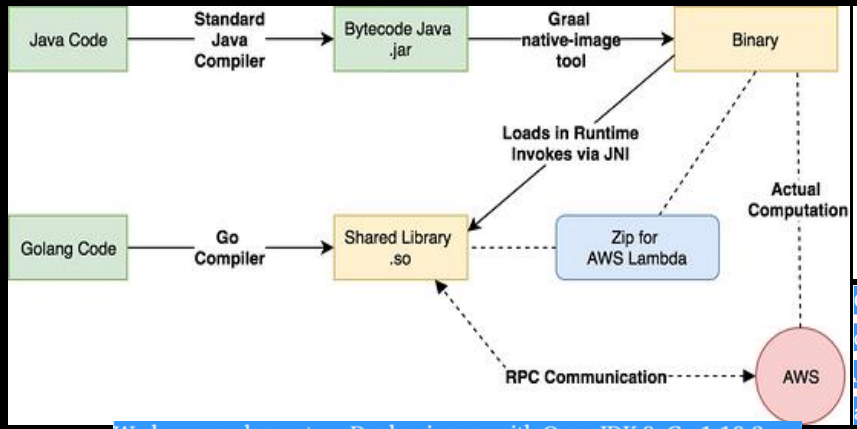
- $JDK12_SRC/src/jdk.internal.vm.ci/share/classes/
  $JDK12_SRC/src/jdk.internal.vm.compiler/share/classes
  $JDK12_SRC/src/hotspot/share/jvmci/



Ecosystem

- **Twitter Service (Production Env)**
  Runs 100% on Graal in production
  “Twitter's Quest for a Wholly Graal Runtime”, JavaOne 2017

- **Using GraalVM to run Native Java in AWS Lambda with Golang**



We have used a custom Docker image with OpenJDK 8, Go 1.10.2 and GraalVM 1.0 RC1.

Memory (MB)	Avg Duration (ms)	Max Duration (ms) java	Avg Graal + Go (ms)	Max Graal + Go (ms)
256	489	3179	992	1011
512	235	1426	486	529
1024	123	652	243	266
1536	85	443	162	173
2048	78	371	143	153

Graal + Go version was always slower, but the runtime duration was more consistent. It suffered no cold start, but it also consumed more memory. Plain Java version used 50–60 MB of memory but the Graal version used additional 110–290 MB of memory.

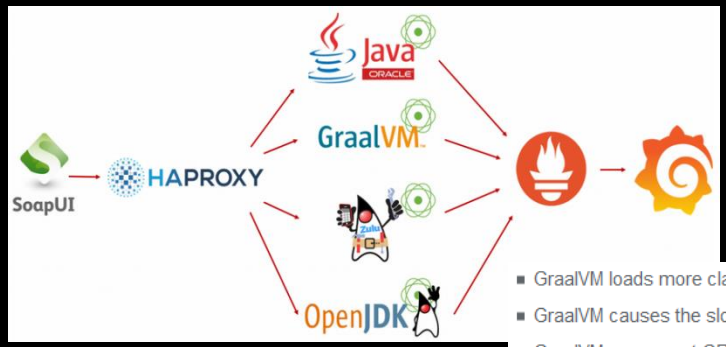
Source: https://engineering.opsgenie.com/run-native-java-using-graalvm-in-aws-lambda-with-golang-ba86e27930bf

Performance

- https://www.graalvm.org/docs/why-graal/
- https://www.graalvm.org/docs/examples/java-performance-examples/
- https://blog.linkerd.io/2018/06/04/announcing-the-linkerd-graalvm-working-group/

- …
- SPECjbb2015?

- https://technology.amis.nl/2018/11/23/comparing-jvm-performance-zulu-openjdk-openjdk-oracle-jdk-graalvm-ce/

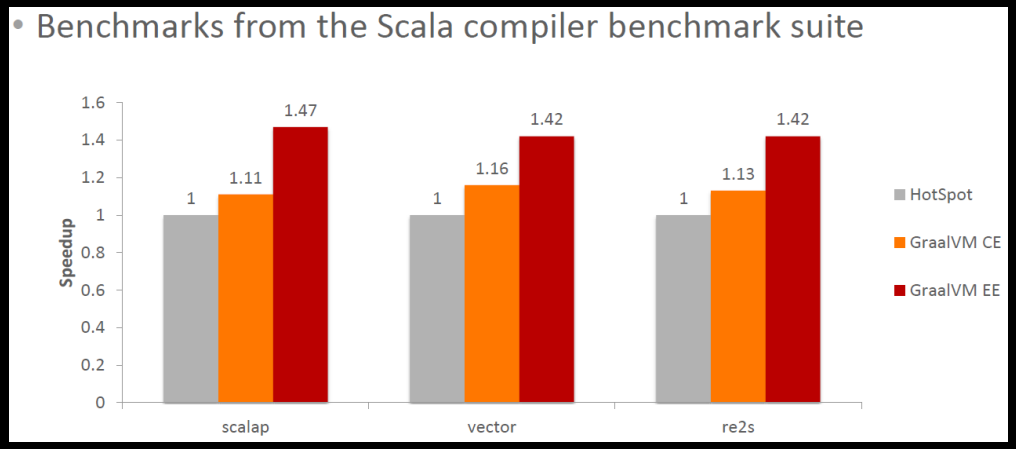


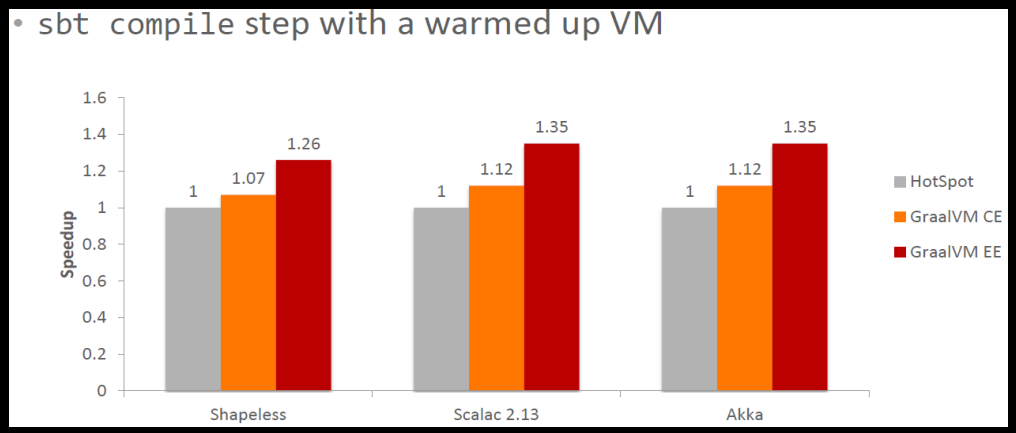
- GraalVM CE rc9 (8u192)
- OpenJDK 8u191
- Zulu 8u192
- Oracle JDK 8u181

- GraalVM loads more classes for the same application
- GraalVM causes the slowest response times for the application
- GraalVM uses most CPU (to achieve the slowest response time)
- GraalVM uses most time on garbage collection
- Zulu OpenJDK uses least memory of the compared JVMs. Zulu OpenJDK and GraalVM are more stable in their memory usage when comparing to Oracle JDK and OpenJDK.

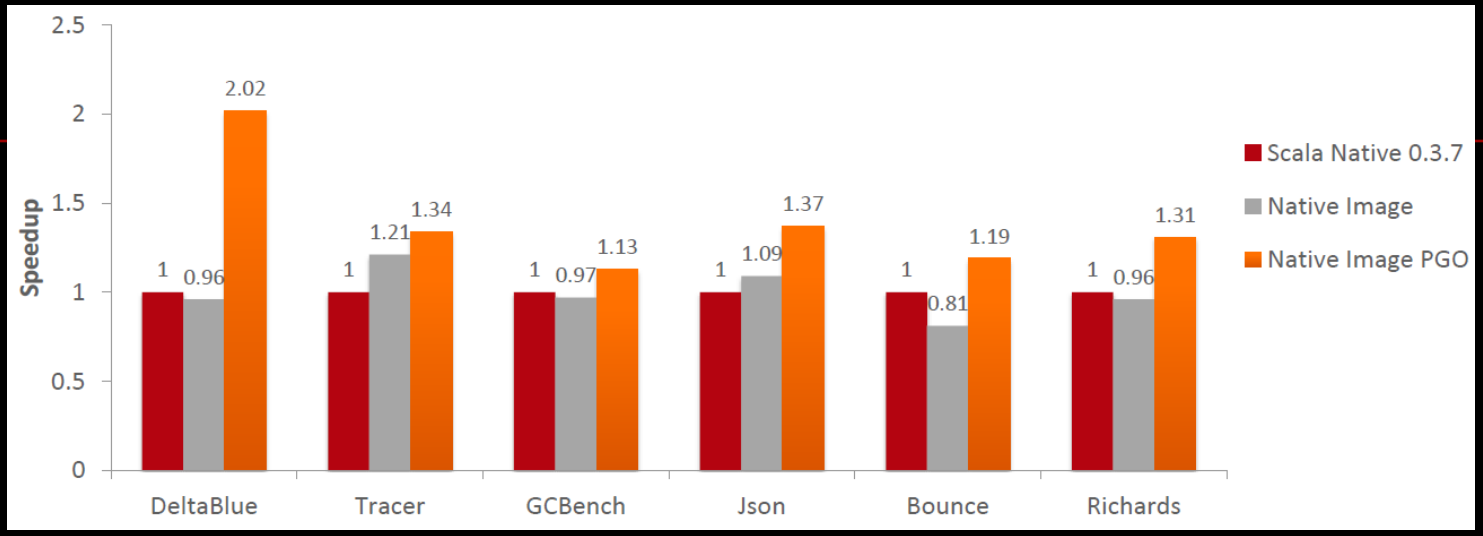
Scala

- https://scala-lang.org
- Famous Scala projects:
  BigData(Spark), Server(Twitter), (Akka), EDA(Chisel)…
- “Compiling ScalaFaster with GraalVM”, Scala Days 2018





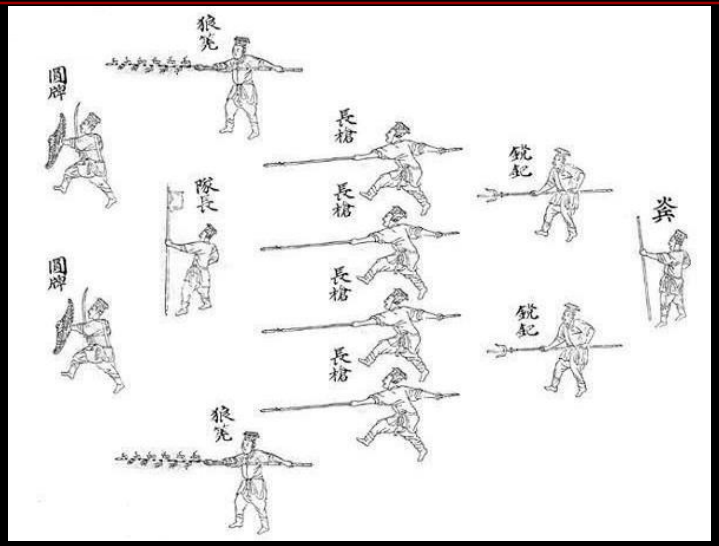
- http://www.scala-native.org/
- Scala Native Benchmarks running with the immix GC



Source: "Compiling Scala Faster with GraalVM", Scala Days 2018

Mixed-Language Programming

- https://en.wikipedia.org/wiki/Polyglot_(computing)
- Polyglot — use the best tool for the right jobs: high performance, scripting, web, functional programming, etc



- $GRAALVM_SRC/sdk/src/org.graalvm.polyglot
  http://www.graalvm.org/docs/reference-manual/polyglot/

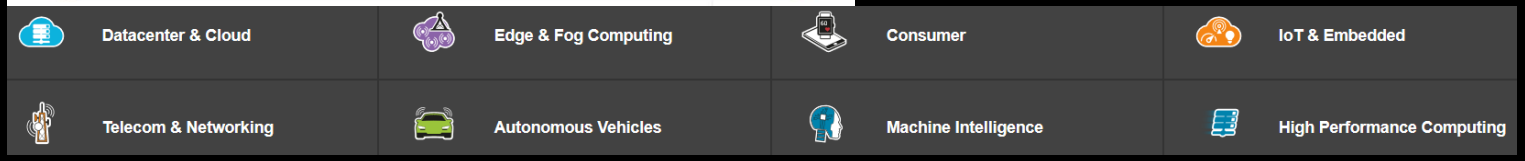
```
import polyglot
array = polyglot.eval(language="js", string="[1,2,42,4]")
print(array[2])
```

```
$ graalpython --polyglot --jvm polyglot.py
42
```

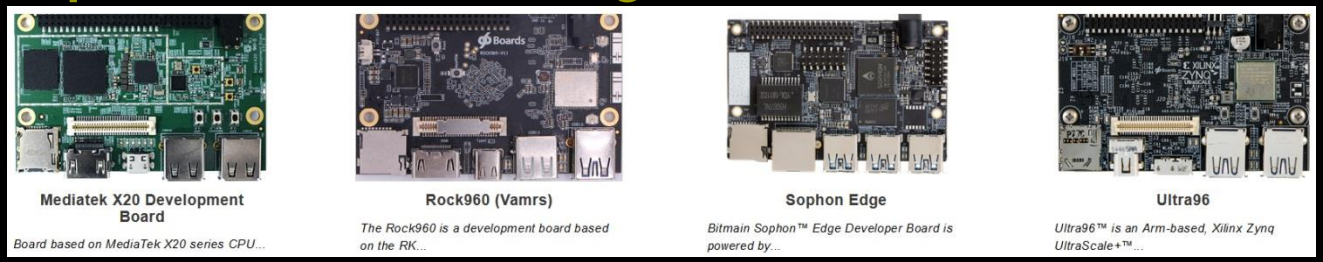
- JavaOne → Oracle Code One (2018)

2) ARM

- https://en.wikipedia.org/wiki/ARM_architecture
- https://www.arm.com/
- http://www.linaro.org



- https://www.96boards.org/



Trends for 2018

NXP ← Q ← Qualcomm

CAVIUM THUNDERX2 ← Broadcom ← MARVELL

applied micro ← MACOM

X-Gene 3
16nm FINFET

AMPERE

HUAWEI

Huawei Hi16xx Family

	Hi1620	Hi1616	Hi1612	Hi1610
Announced	2018	2017	2016	2015
Cores	24 to 64	32	32	16
Architecture	Ares	Cortex-A72	Cortex-A57	Cortex-A57
Frequency (GHz)	2.4 to 3.0	2.4 GHz	2.1 GHz	2.1 GHz
L1	64 KB L1-I 64 KB L1-D	48 KB L1-I 32 KB L1-D	48 KB L1-I 32 KB L1-D	48 KB L1-I 32 KB L1-D
L2	512 KB Private	1MB/4 cores	1MB/4 cores	1MB/4 cores
L3	1MB/core Shared	32MB CCN	32MB CCN	16MB CCN
Memory	8x DDR4-3200	4x DDR4-2400	4x DDR4-2133	2x DDR4-1866
Interconnect	Up to 4S 240 Gbps/port	Up to 2S 96 Gbps/port	?	?
IO	40 PCIe 4.0 2 x 100 GE	46 PCIe 3.0 8 x 10GE	16 PCIe 3.0	16 PCIe 3.0
Process	TSMC 7nm	TSMC 16nm	TSMC 16nm	TSMC 16nm
Power	100 to 200 W	85W	?	?

Cloud on ARM

- **Amazing! AWS re:Invent 2018**
  https://aws.amazon.com/ec2/instance-types/a1/

Model	vCPUs	Memory (GiB)	Instance Storage	Network Bandwidth (Gbps)	EBS Bandwidth (Mbps)
a1.medium	1	2	EBS-Only	Up to 10	Up to 3,500
a1.large	2	4	EBS-Only	Up to 10	Up to 3,500
a1.xlarge	4	8	EBS-Only	Up to 10	Up to 3,500
a1.2xlarge	8	16	EBS-Only	Up to 10	Up to 3,500
a1.4xlarge	16	32	EBS-Only	10	3,500

AWS Designed Processor: Graviton

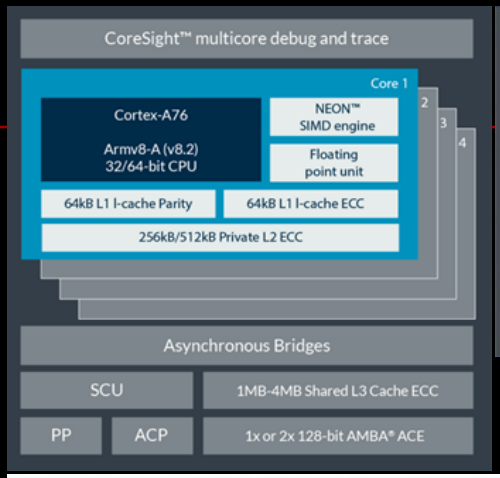
packet.net

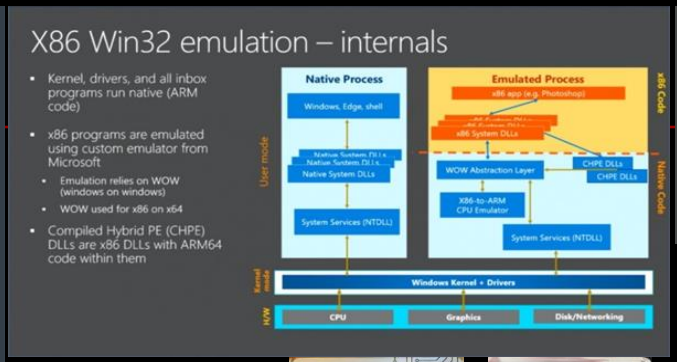
- **https://retout.co.uk/blog/2017/04/25/packet-net-arm64-servers**

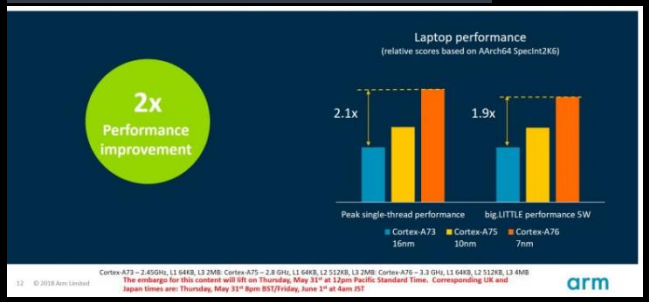
Packet.net offer an ARMv8 server with 96 cores for $0.50/hour.

Year 2019

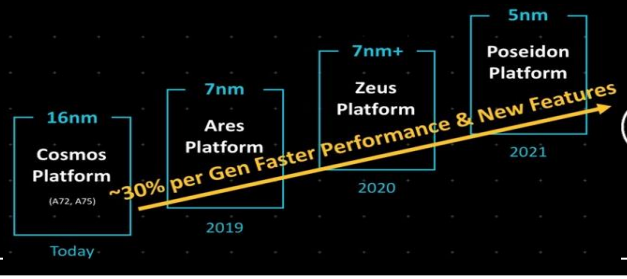
- ARM PC/Laptop







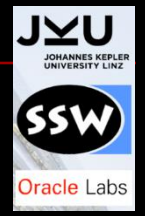
- ARM Neoverse



II. Dive Into GraalVM

Overview

- https://www.oracle.com/technetwork/oracle-labs/program-languages/overview/index.html
- http://ssw.jku.at/
- Src
  https://github.com/graalvm
  https://github.com/oracle/graal



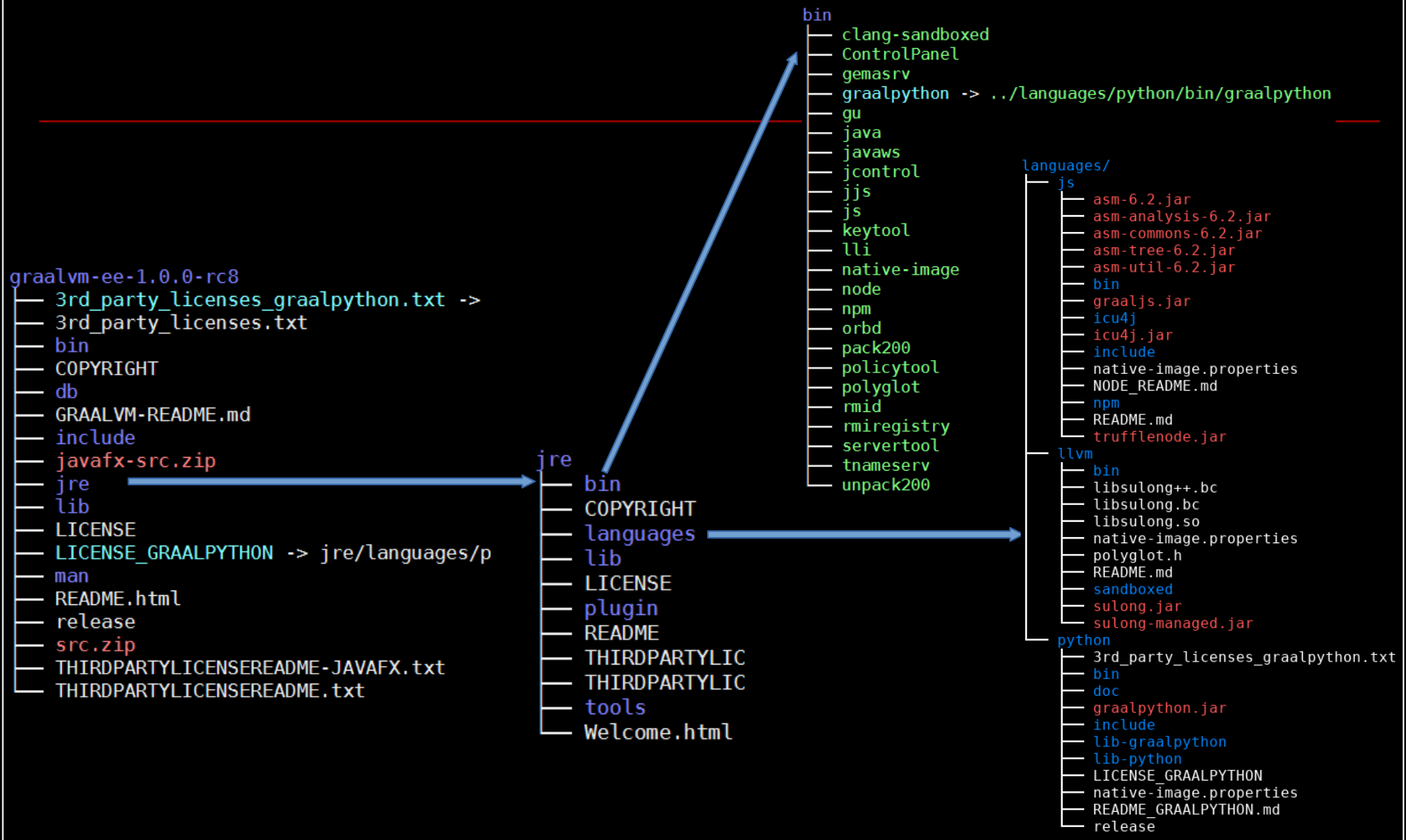
Repository Structure

The GraalVM main source repository includes the following components:

- Graal SDK contains long term supported APIs of GraalVM.
- Graal compiler written in Java that supports both dynamic and static compilation and can integrate with the Java HotSpot VM or run standalone.
- Truffle language implementation framework for creating languages and instrumentations for GraalVM.
- Tools contains a set of tools for GraalVM languages implemented with the instrumentation framework.
- Substrate VM framework that allows ahead-of-time (AOT) compilation of Java applications under closed-world assumption into executable images or shared objects.
- Sulong is an engine for running LLVM bitcode on GraalVM.
- TRegex is an implementation of regular expressions which leverages GraalVM for efficient compilation of automata.
- VM includes the components to build a modular GraalVM image.

- **Binary**

https://www.graalvm.org/docs/reference-manual/graal-updater (gu)



- **Src**

GraalVM CE 1.0.0 RC10

```
[mydev@myfedora GraalVM]$ cloc graal-vm-1.0.0-rc10
Unescaped left brace in regex is deprecated here (and will be fatal in Perl 5.32), p
{ <-- HERE |})/ at /usr/bin/cloc line 7801.
    8704 text files.
    8639 unique files.
     548 files ignored.

github.com/AlDanial/cloc v 1.72  T=62.67 s (130.3 files/s, 19862.0 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Java                          6464         135722         293043         700547
C                             1448           3788          44100          25495
Python                          30           1929           1769          12367
Markdown                        54           1388              0           4617
C++                             81            696           2884           4054
C/C++ Header                    43            712           2239           3480
XML                             16             35            152           3358
make                            16            213            484            557
Clojure                          1             29             31            300
Bourne Shell                     3             36            131            245
YAML                             1              8              3             98
HTML                             1             11             20             40
Bourne Again Shell               1              4              1             27
JSON                             1              0              0             12
Visualforce Component            2              0              0             10
JavaScript                       1              0             24              2
-------------------------------------------------------------------------------
SUM:                          8163         144571         344881         755209
-------------------------------------------------------------------------------
```

OpenJDK 12 master

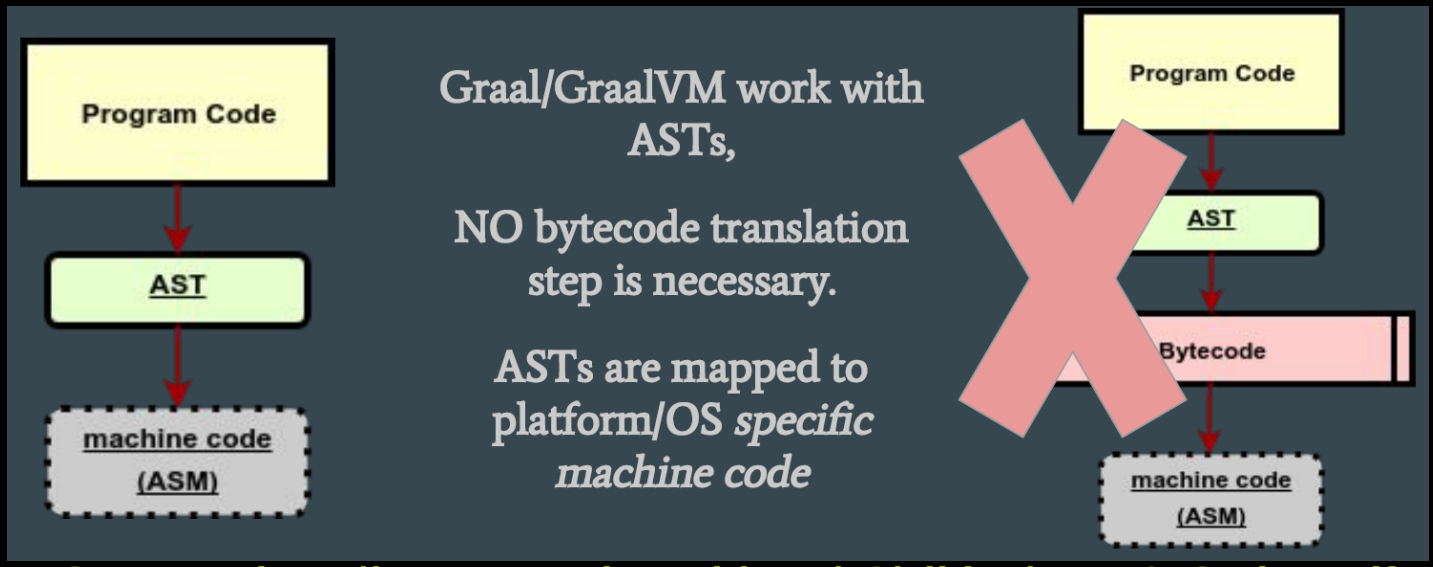
```
[mydev@myfedora OpenJDK]$ cloc openjdk12/
Unescaped left brace in regex is deprecated here (and will be fatal in Perl 5.32), p
E in m/({ <-- HERE |})/ at /usr/bin/cloc line 7801.
   65382 text files.
   63880 unique files.
    7045 files ignored.

github.com/AlDanial/cloc v 1.72  T=467.64 s (124.8 files/s, 28065.4 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
Java                         47532        1000866        3054397        5062506
XML                           1348           2776           9512        1189812
C++                           2929         155748         219676         818256
C                             1041          86050          94023         399968
C/C++ Header                  2629          77655         154212         317445
...
```

1) Truffle /Graal

AST

- https://en.wikipedia.org/wiki/Abstract_syntax_tree
- Graal/GraalVM: ASTs as first class citizen

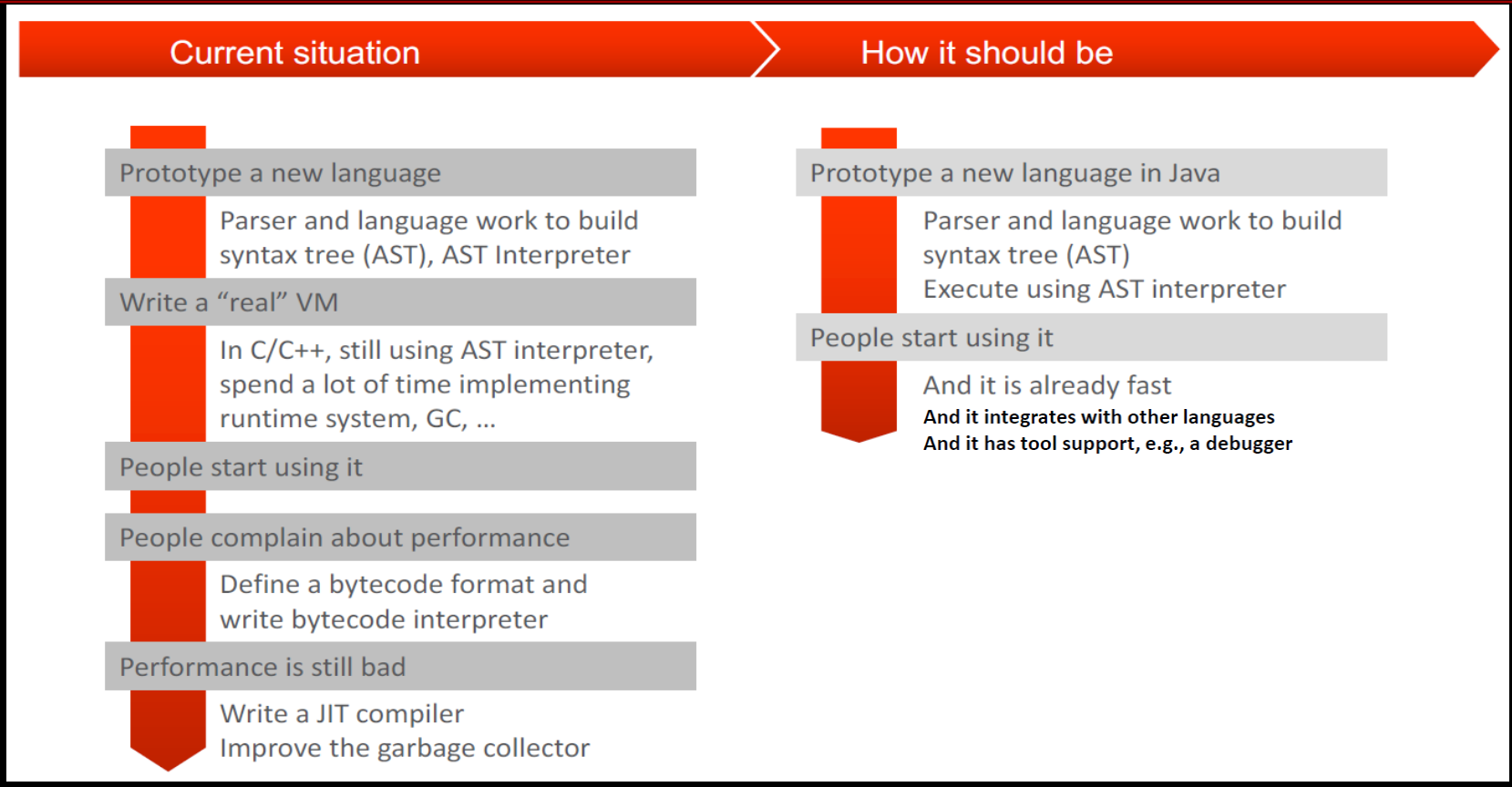


Source: http://crest.cs.ucl.ac.uk/cow/59/slides/cow59_Sarkar.pdf

1) Truffle & Graal

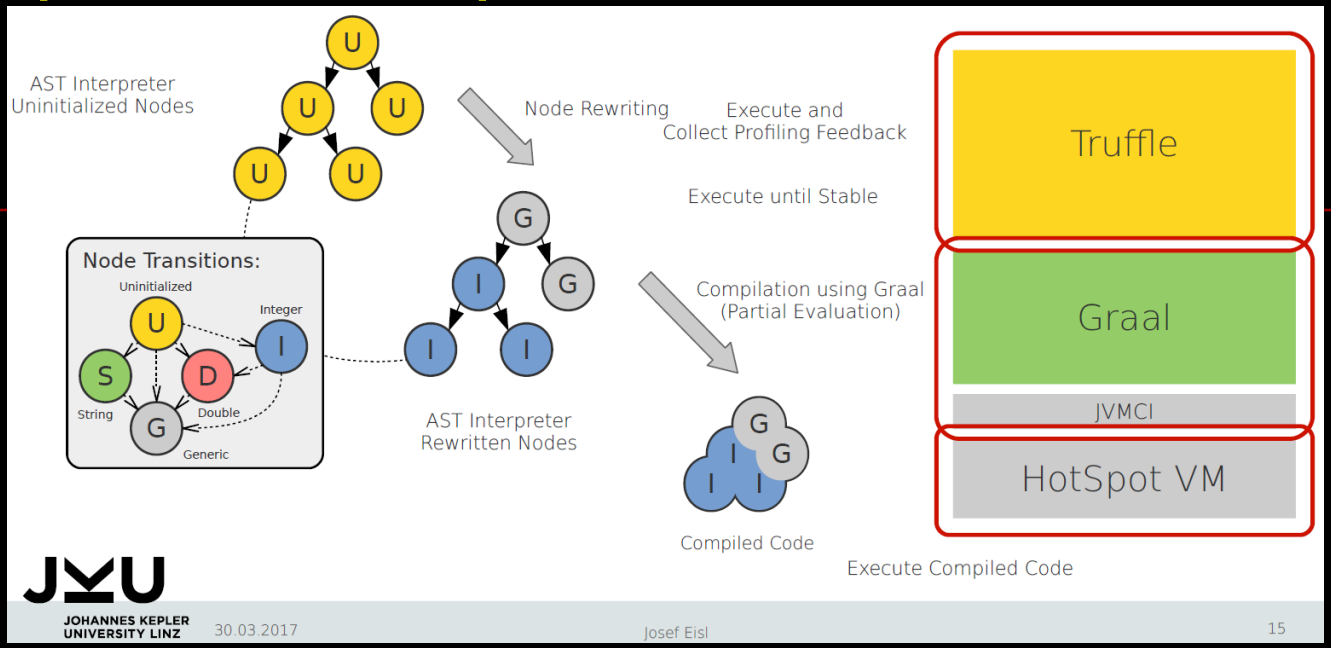
- https://github.com/neomatrix369/awesome-graal

Implement a new language runtime



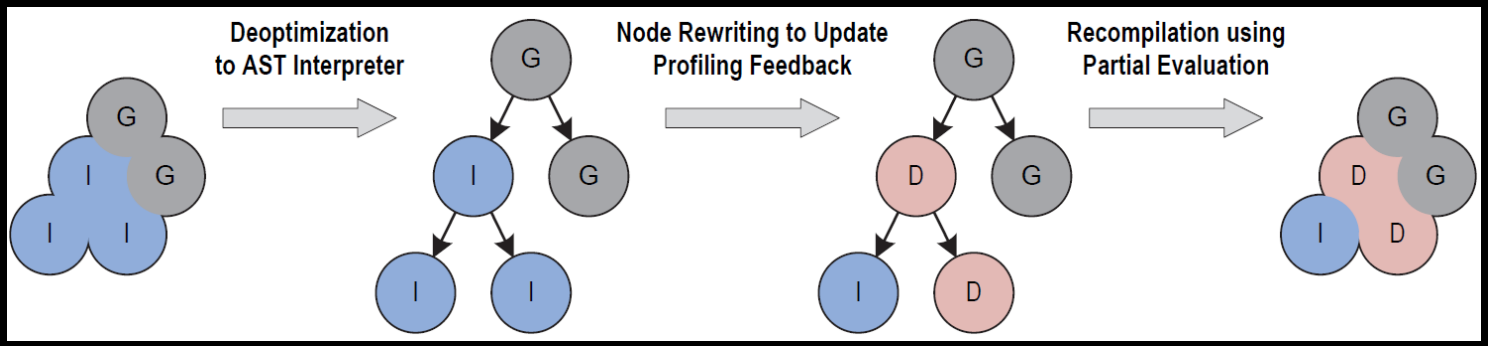
Source: "Turning the JVM into a Polyglot VM with Graal", Chris Seaton, Oracle Labs

Optimization and Speculation



Source: https://www.complang.tuwien.ac.at/lehre/ubvo/jku.pdf

Deoptimization



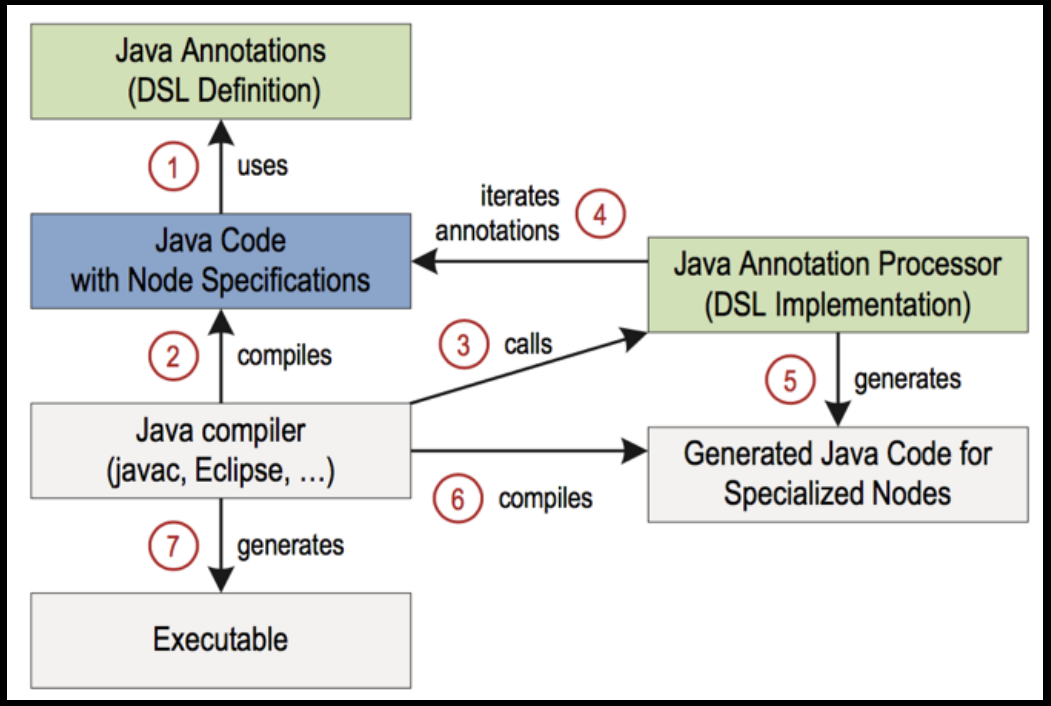
Source: https://qconnewyork.com/system/files/presentation-slides/qconf_final.pdf

Internal of Truffle

- **Truffle DSL (Specialization and Node Rewriting)**
  Java with compiler directives, annotations and an annotation processor

---

- **javac -processor AnnotationProcessor AnnotatedFile.java**
- **DSL processing flow**



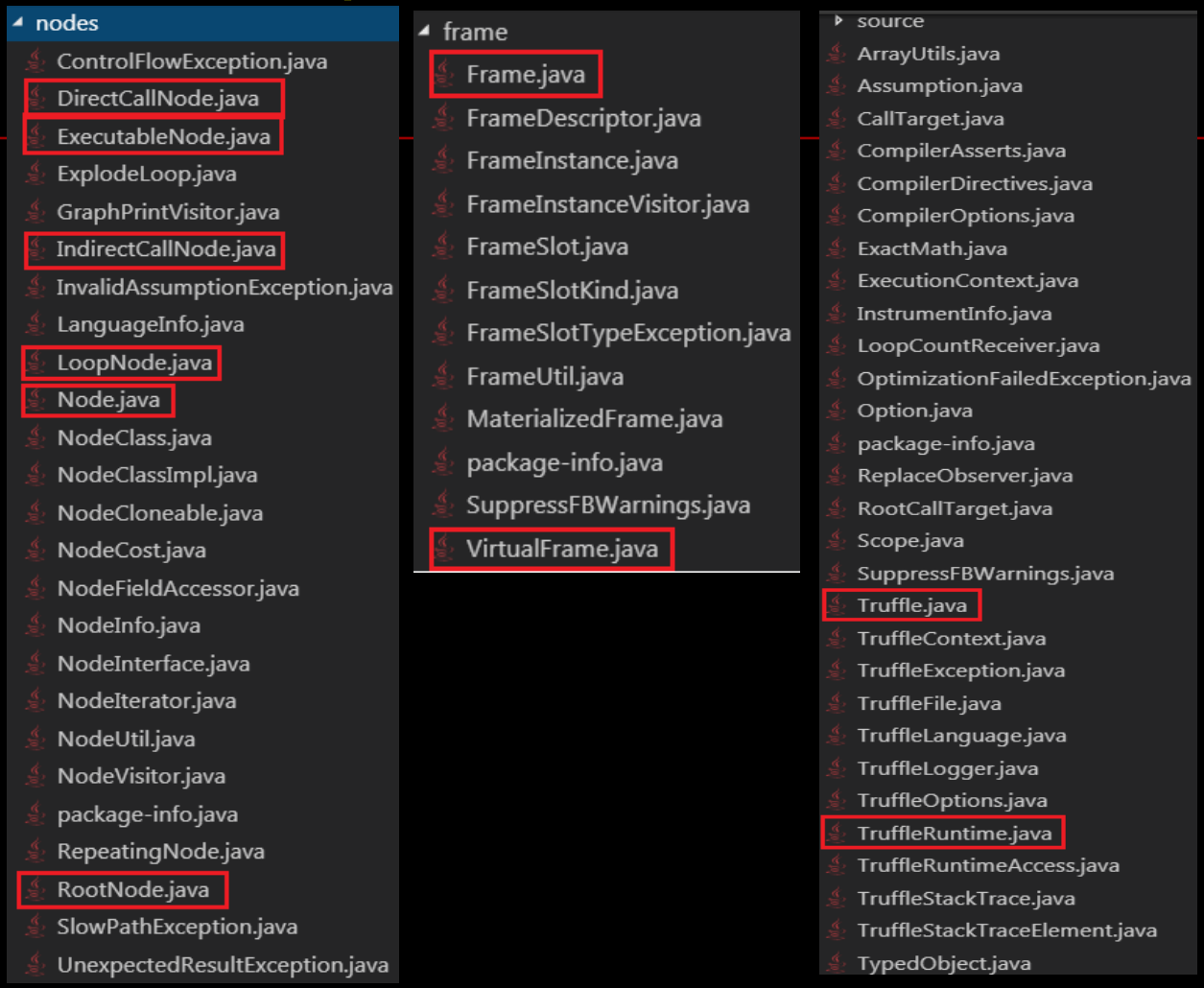

```java
@Specialization(rewriteOn=ArithmeticException.class)
int addInt(int a, int b) {
  return Math.addExact(a, b);
}

@Specialization
double addDouble(double a, double b) {
    return a + b;
}

@Generic
Object addGeneric(Frame f, Object a, Object b) {
  // Handling of String omitted for simplicity.
  Number aNum = Runtime.toNumber(f, a);
  Number bNum = Runtime.toNumber(f, b);
  return Double.valueOf(aNum.doubleValue() +
          bNum.doubleValue());
}
```

Source: http://lafo.ssw.uni-linz.ac.at/papers/2014_SPLASH_OneVMToRuleThemAll.pdf

- $GRAALVM_SRC/truffle/src/com.oracle.truffle.api/src/com/oracle/truffle/api/



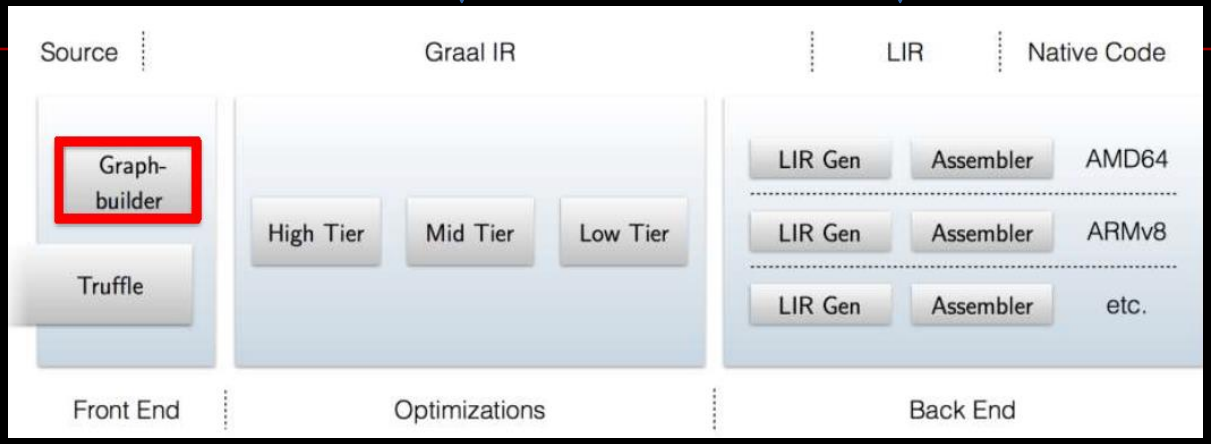
- https://github.com/graalvm/simplelanguage

Internal of Graal

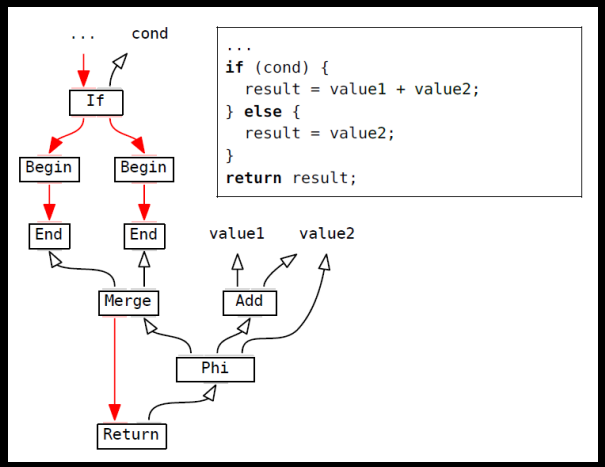
- IR

Graph-Based IR
Platform Independent

Graph-Based IR
Platform Dependent

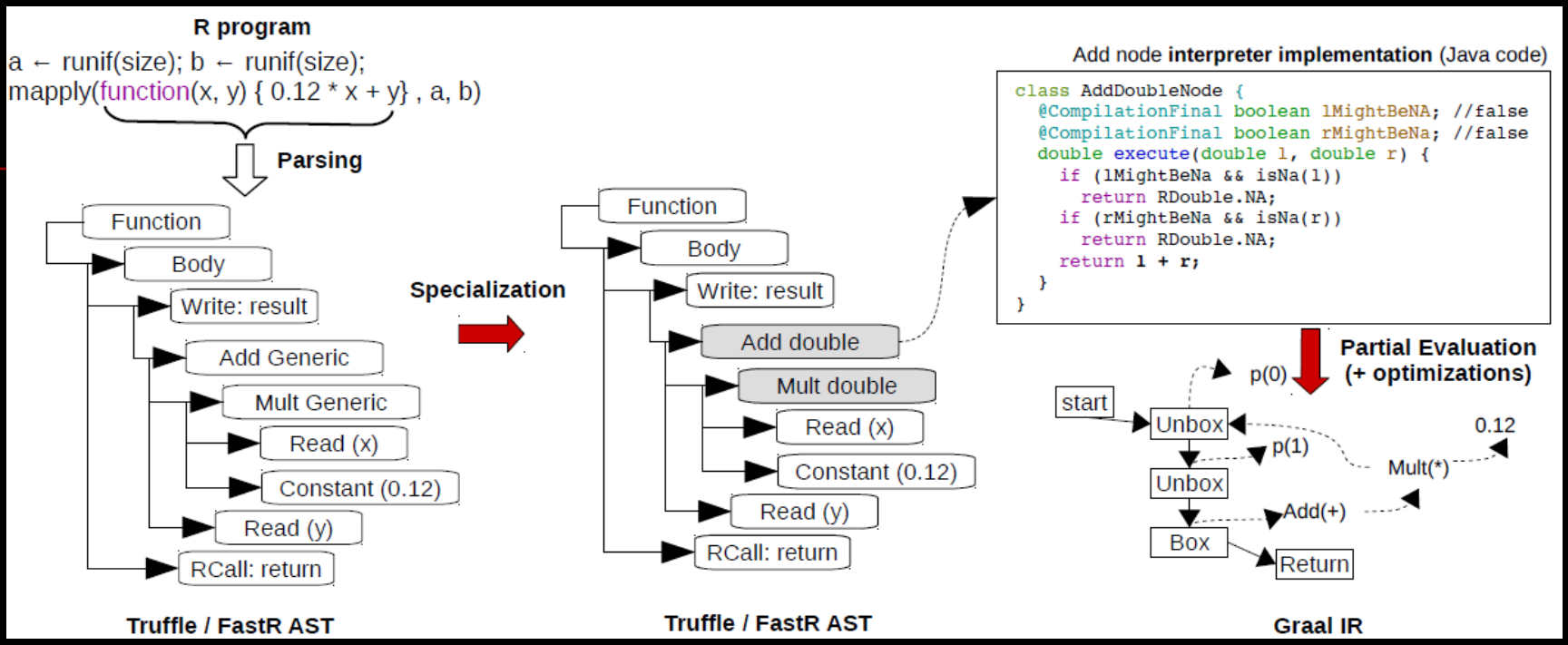


Source: http://ssw.jku.at/Research/Papers/Stadler14PhD/Thesis_Stadler_14.pdf



Source: http://ssw.jku.at/General/Staff/GD/APPLC-2013-paper_12.pdf

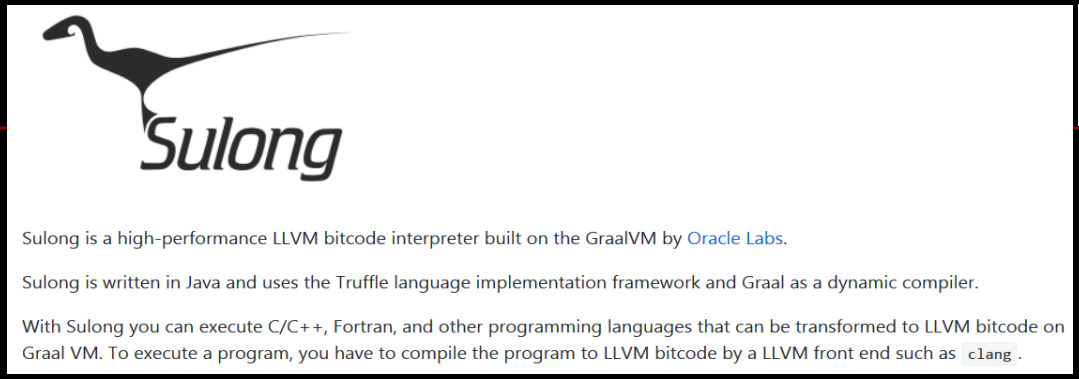
Execution of an R program in FastR via Truffle and Graal



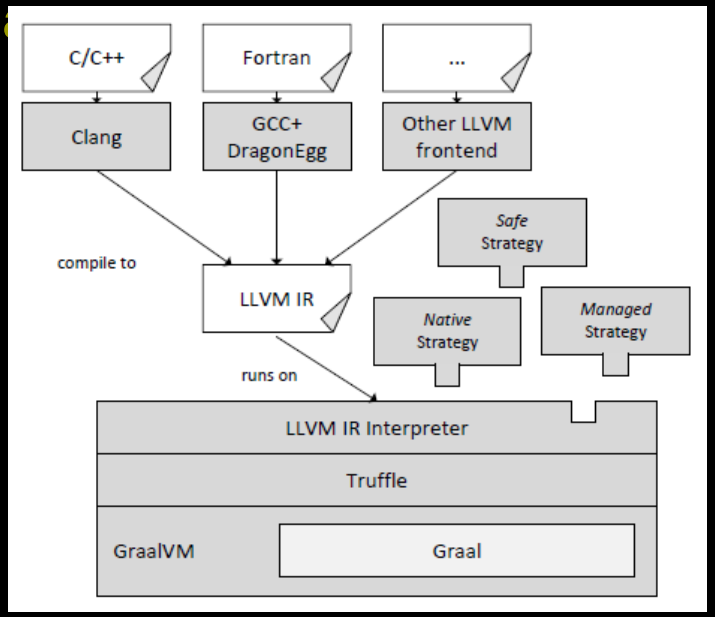
Source: https://michel.steuwer.info/files/publications/2017/VEE-2017.pdf

2) Sulong

- https://github.com/oracle/graal/tree/master/sulong



- Sulong: Chinese for velocisaurus
  - 速: fast, rapid
  - 龙: dragon



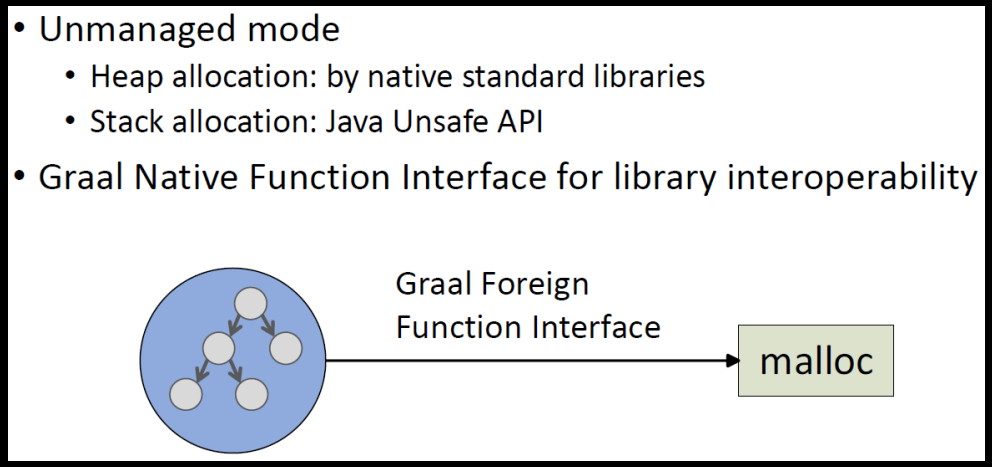
Source: http://ssw.jku.at/General/Staff/ManuelRigger/MoreVMs18.pdf

Internal

- Memory Safe and Efficient Execution of LLVM-Based Languages
- https://www.graalvm.org/docs/reference-manual/languages/llvm/
- Node ⟵ LLVMNode

  $GRAALVM_SRC/sulong/projects/com.oracle.truffle.llvm.nodes/src/com/oracle/truffle/llvm/nodes

  $GRAALVM_SRC/sulong/projects/com.oracle.truffle.llvm.runtime/src/com/oracle/truffle/llvm/runtime
- Memory



Source: https://www.llvm.org/devmtg/2016-01/slides/Sulong.pdf

3) SVM

- **JEP 295: Ahead-of-Time Compilation**
- **Self-contained native executables**

... an **embeddable** VM

with **fast startup** and **low footprint**

for, and written in, a **subset of Java**

optimized to **execute Truffle** languages

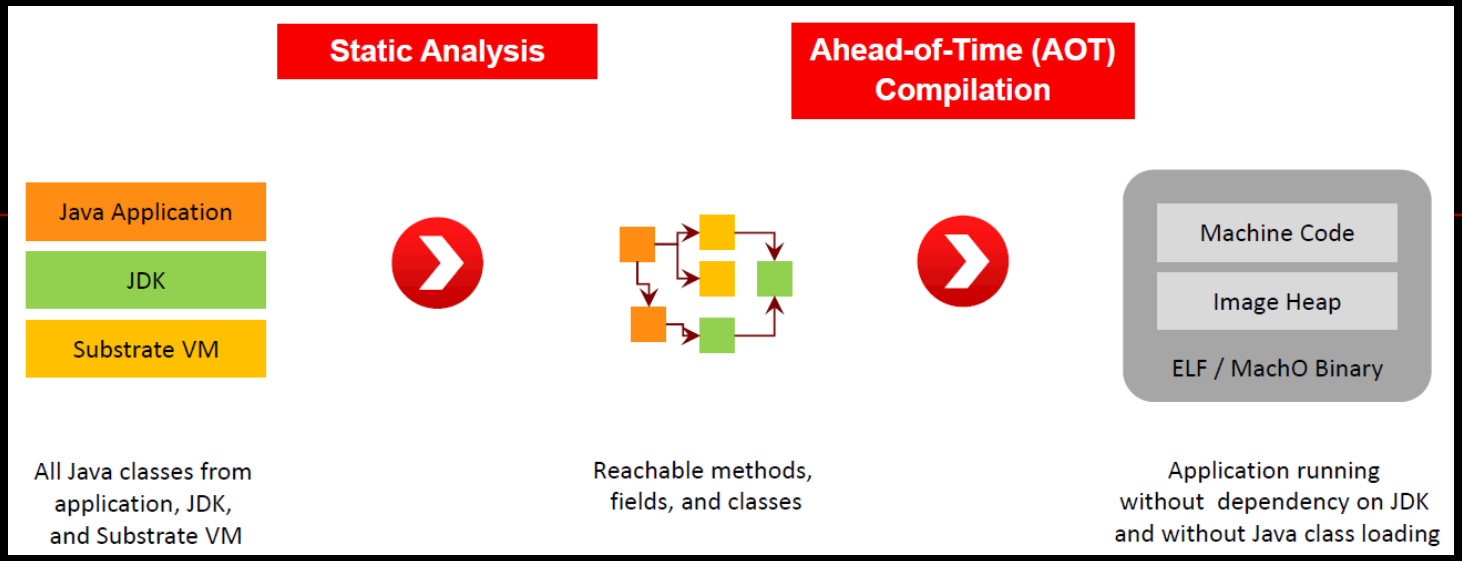
ahead-of-time compiled using Graal

integrating with **native development tools.**

- Type safety and memory safety of Java
  - Type checks, array bounds checks, null pointer checks
- Garbage collection
  - All Java memory is managed automatically
- JDK support
  - Most core and utility classes
- C code integration
  - SystemJava: access C functions and C data structures without performance overhead
- Multithreading (optional feature)
  - Everything in `java.util.concurrent` package
- Native tool support for debugging, profiling, ...
  - Standard DWARF debug information for ahead-of-time compiled code and dynamically compiled code

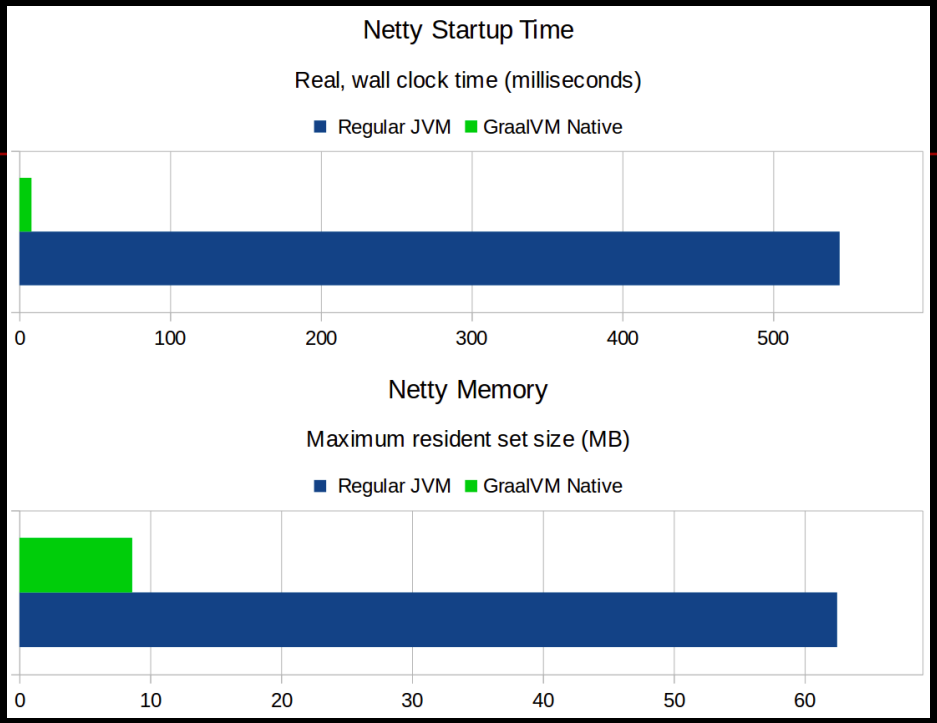
Source: https://www.complang.tuwien.ac.at/lehre/ubvo/substrate.pdf

Procedure



Source: https://static.rainfocus.com/oracle/oow18/sess/1526041579721001pM2J/PF/-2018-10-24%20SubstrateVM%20CodeOne_1540478815946001 9swO.pdf

- https://medium.com/graalvm/instant-netty-startup-using-graalvm-nativeimage-generation-ed6f14ff7692

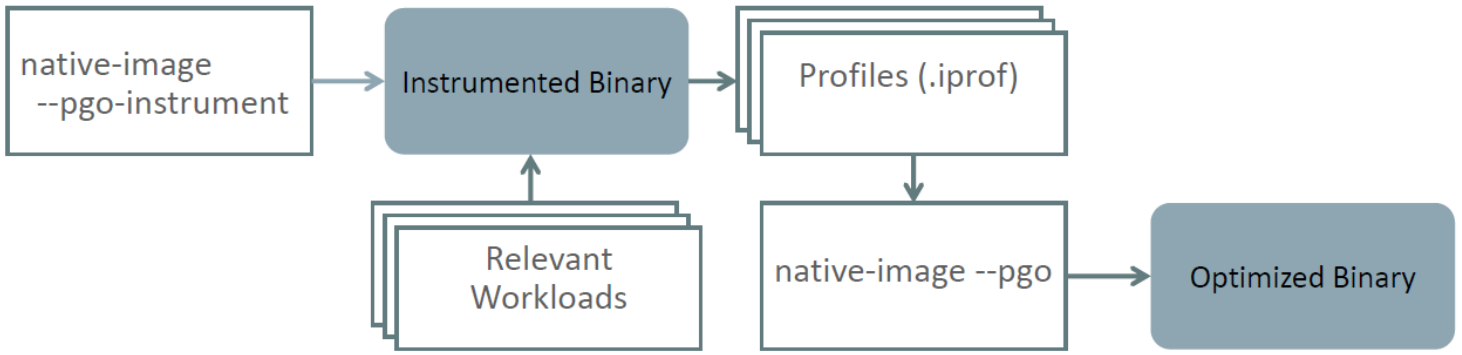


Source: "Compiling Java ahead-of-time with GraalVM", Jokerconf 2018

- https://github.com/cstancu/netty-native-demo
- https://github.com/taylorwood/lein-native-image
- https://spring.io/blog/2018/10/02/the-evolution-of-spring-fu
  https://jira.spring.io/browse/SPR-16991
- …

PGO (Profile-Guided Optimizations)

- Graal compiler is built ground-up with profiles in mind
  - Collecting profiles is essential for performance of native images
- PGO requires running relevant workloads before building an image



Source: "Compiling Scala Faster with GraalVM", Scala Days 2018

Why SVM & AOT?

- Different target group / alignment to HotSpot
  - Cloud, Container, Functions, IoT
  - Small & volatile
  - Jakarte EE / Desktop out of scope

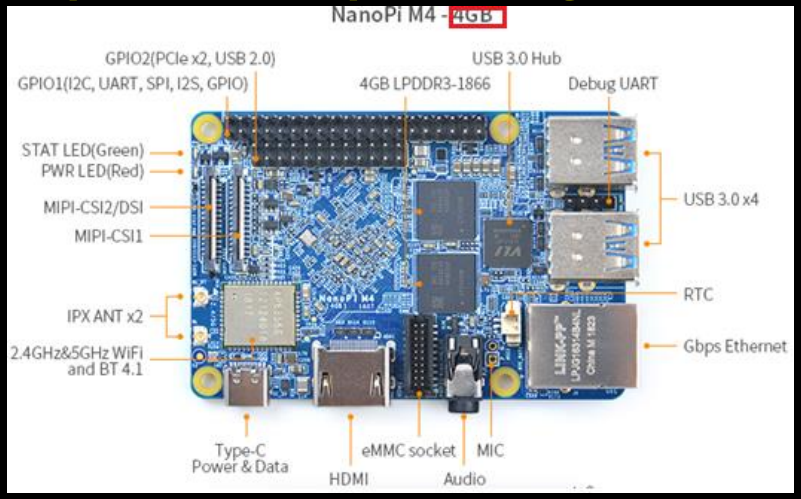
Source: https://www.slideshare.net/trivadis/techevent-graalvm-performance-interoperability?qid=e20f523f-c994-48a9-b308-a23783667631&v=&b=&from_search=14

III. GraalVM on ARM

1) Development Env

NanoPi M4

- http://wiki.friendlyarm.com/wiki/index.php/NanoPi_M4
- https://en.wikipedia.org/wiki/Rockchip



- Ubuntu 18.04 Desktop/Core & Android 8.1/7.1.2 for AARCH64

My Dev Env

- https://www.armbian.com/nanopi-m4 (upgrade to Ubuntu 18.10)

```
myrk3399@nanopim4:/$ uname -a
Linux nanopim4 4.4.162-rk3399 #41 SMP Fri Oct 26 14:03:47 CEST 2018 aarch64 aarch64 aarch64 GNU/Linux
```

2) My Practice

Technical Solution

- **http://openjdk.java.net/projects/jdk/11/**

Features

181: Nest-Based Access Control
309: Dynamic Class-File Constants
315: Improve Aarch64 Intrinsics
318: Epsilon: A No-Op Garbage Collector
320: Remove the Java EE and CORBA Modules
321: HTTP Client (Standard)
323: Local-Variable Syntax for Lambda Parameters
324: Key Agreement with Curve25519 and Curve448
327: Unicode 10
328: Flight Recorder
329: ChaCha20 and Poly1305 Cryptographic Algorithms
330: Launch Single-File Source-Code Programs
331: Low-Overhead Heap Profiling
332: Transport Layer Security (TLS) 1.3
333: ZGC: A Scalable Low-Latency Garbage Collector (Experimental)
335: Deprecate the Nashorn JavaScript Engine
336: Deprecate the Pack200 Tools and API

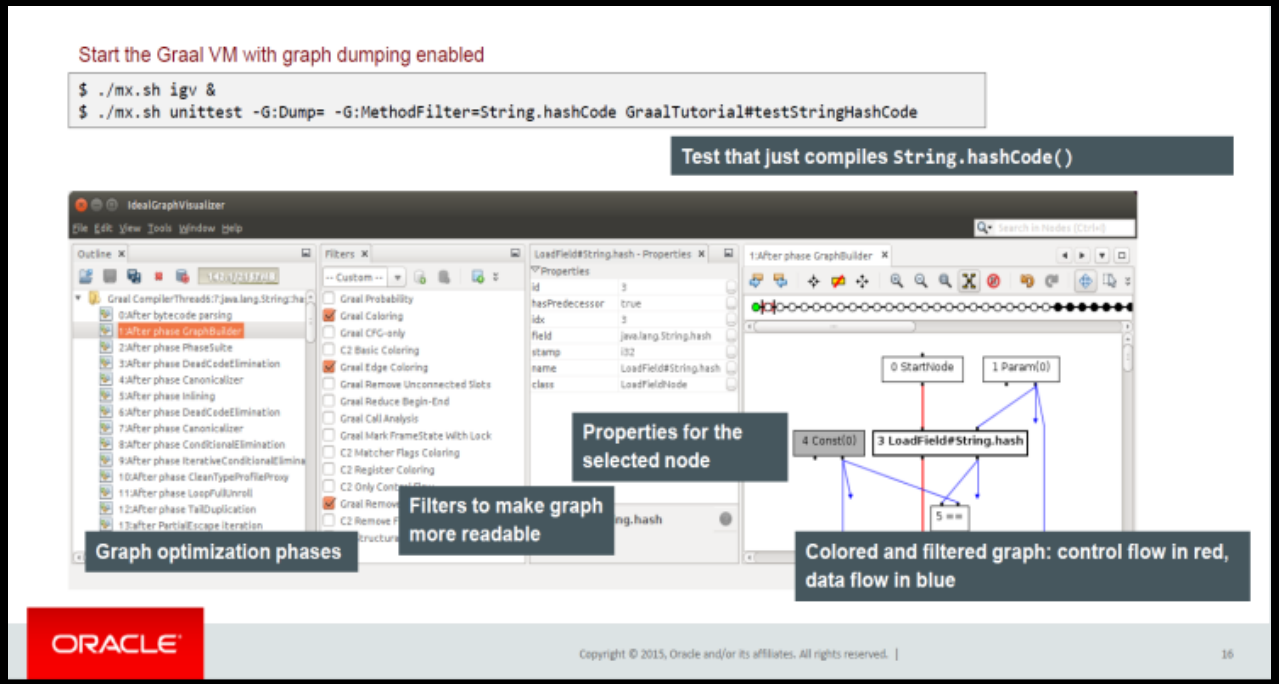
- **http://openjdk.java.net/projects/metropolis/**

- Experimental clone of JDK 11 (*not* for immediate release)
- Hosting work on AOT and the Graal compiler
- Definition of "System Java" for implementing HotSpot modules.
  - Experimentation with SVM-style deployment.
- Translation of discrete HotSpot modules into System Java.
- The Big One: Compilation of Graal as System Java for JIT
  - Replacement for C2, then C1, then stub and interpreter generators.
  - This will take a long time, but it's a necessary technology refresh.
- *Tomorrow's reference implementation!*

Source: http://cr.openjdk.java.net/~jrose/pres/201801-JIT-Metropolis.pdf

MX

- https://github.com/graalvm/mx
- mx is a command line based tool for managing the development of (primarily) Java code. It includes a mechanism for specifying the dependencies as well as making it simple to build, test, run, update, etc the code and built artifacts
- mx is written in Python (version 2.7) and is extensible
- mx –help
- IR Example: Ideal Graph Visualizer



OpenJDK 11 on ARM

- build OpenJDK 11 by yourself
  http://hg.openjdk.java.net/jdk-updates/jdk11u/
- https://github.com/AdoptOpenJDK/openjdk11-binaries/releases
  e.g., OpenJDK11U-jdk_aarch64_linux_hotspot_2018-11-26-07-41.tar.gz
- #export JDK_BOOT_DIR=$YOUR_OpenJDK11-AARCH64_HOME

- reserve at least 6GB disk space
- on NanoPi M4 with Ubuntu 18.10 (Kernel 4.4.162) + 

+ GCC 8.2.0-7 + jemalloc 5.1.0 + 6GB Memory (4GB DDR4 + 2GB Swap)

```
myrk3399@nanopim4:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM$ free -m
              total        used        free      shared  buff/cache   available
Mem:           3811         398        2754          17         658        3307
Swap:          1905           0        1905
```

- cd $YOUR_OPENJDK11_SRC_HOME and run the commands:
  bash configure --disable-warnings-as-errors
  make JOBS=6 images

build

- setup mx
- env variables

```
#export PYTHON_HOME=
#export PYTHONPATH=/home/mydev/.local/lib/python3.6/site-packages
export MX_HOME=/opt/MyWorkSpace/DevSW/Tools/Build/mx
#export JAVA_HOME=/opt/MyWorkSpace/DevSW/Java/JDK/Oracle/Std/11/jdk-11.0.1
#export GRAALVM_HOME=/opt/MyWorkSpace/DevSW/Java/JDK/Oracle/GraalVM/EE/graalvm-ee-1.0.0-rc10
export LLVM_HOME=/opt/MyWorkSpace/DevSW/Toolchain/LLVM/clang-llvm-7.0.0-aarch64-linux-gnu
export PATH=$MX_HOME:$LLVM_HOME/bin:$PATH
#export PATH=$JAVA_HOME/bin:$MX_HOME:$GRAALVM_HOME/jre/bin:$PATH
```

- build script

```
date
export JAVA_HOME=/opt/MyWorkSpace/DevSW/Java/JDK/Oracle/Std/11/jdk-11.0.1+13
cd graal/vm

echo -e "\n\n\n=============== start building GraalVM on ARM64 ==============="
mx -V --dy /substratevm,/tools build | gawk '{print strftime("%Y-%m-%d %H:%M:%S"),$0}'
#mx -V --dy /substratevm,/tools,/sulong build | gawk '{print strftime("%Y-%m-%d %H:%M:%S"),$0}'

echo -e "\n\n\n=============== end of building GraalVM on ARM64 ==============="
date
```

patching

Java compliance, build with JDK 11, Polyglot Native API, VisualVM (currently do not support ARM64) Sulong (need porting to AARCH64)…

modified: compiler/mx.compiler/suite.py
modified: substratevm/mx.substratevm/mx_substratevm.py
modified: substratevm/mx.substratevm/suite.py
modified: tools/mx.tools/mx_tools.py
modified: tools/mx.tools/suite.py
modified: vm/mx.vm/mx_vm.py
modified: vm/mx.vm/suite.py

```
diff --git a/vm/mx.vm/mx_vm.py b/vm/mx.vm/mx_vm.py
index 97602eccbe3..61d59150c27 100644
--- a/vm/mx.vm/mx_vm.py
+++ b/vm/mx.vm/mx_vm.py
@@ -1707,11 +1707,11 @@ def check_versions(jdk_dir, jdk_version_regex, e

     out = subprocess.check_output([java, '-version'], stderr=subpr

-    match = jdk_version_regex.match(out)
-    if match is None:
-        mx.abort("'{}' has an unexpected version string:\n{}\ndoes
-    elif not match.group('jvm_version').startswith("1.8.0"):
-        mx.abort("GraalVM requires a JDK8 as base-JDK, while the s
'jvm_version'), out, check_env))
+    #match = jdk_version_regex.match(out)
+    #if match is None:
+    #    mx.abort("'{}' has an unexpected version string:\n{}\ndoes
+    #elif not match.group('jvm_version').startswith("1.8.0"):
+    #    mx.abort("GraalVM requires a JDK8 as base-JDK, while the s
('jvm_version'), out, check_env))
```

```
diff --git a/tools/mx.tools/suite.py b/tools/mx.tools/suite.py
index 776f47e5aee..1941860748c 100644
--- a/tools/mx.tools/suite.py
+++ b/tools/mx.tools/suite.py
@@ -144,26 +144,6 @@ suite = {
             "version" : "1.3.9",
         }
     },
-    "VISUALVM_COMMON" : {
-        "urls" : ["https://lafo.ssw.uni-linz.ac.at/pub/graal-external-deps/visualvm-632.tar.gz"],
-        "sha1" : "dab8a050f338b6df627a8d6a864c8d6498b3a0a6",
-    },
-    "VISUALVM_PLATFORM_SPECIFIC" : {
-        "os_arch" : {
-            "linux" : {
-                "amd64" : {
-                    "urls" : ["https://lafo.ssw.uni-linz.ac.at/pub/graal-external-deps/visualvm-632-linux-amd64.tar.gz"],
-                    "sha1" : "cc822c762985b49343138b9c68496845ee179dd1",
-                }
-            },
-            "darwin" : {
-                "amd64" : {
-                    "urls" : ["https://lafo.ssw.uni-linz.ac.at/pub/graal-external-deps/visualvm-632-macosx-x86_64.tar.gz"],
-                    "sha1" : "b32286a5ca9b59047f2d19a7f4646ab271a27980",
-                }
-            },
-        }
-    },
 },
```

```
diff --git a/substratevm/mx.substratevm/mx_substratevm.py b/substratevm/mx.substratevm/mx_substratevm.py
index 51e1f6c1fb3..0b97c821e5b 100644
--- a/substratevm/mx.substratevm/mx_substratevm.py
+++ b/substratevm/mx.substratevm/mx_substratevm.py
@@ -852,13 +852,11 @@ mx_sdk.register_graalvm_component(mx_sdk.GraalVmJreComponent(
     third_party_license_files=[],
     jar_distributions=['substratevm:POLYGLOT_NATIVE_API'],
     support_distributions=[
-        "substratevm:POLYGLOT_NATIVE_API_SUPPORT",
         "substratevm:POLYGLOT_NATIVE_API_HEADERS",
     ],
     polyglot_lib_build_args=[
         "-H:Features=org.graalvm.polyglot.nativeapi.PolyglotNativeAPIFeature",
         "-Dorg.graalvm.polyglot.nativeapi.libraryPath=<path:POLYGLOT_NATIVE_API_HEADERS>",
-        "-Dorg.graalvm.polyglot.nativeapi.nativeLibraryPath=<path:POLYGLOT_NATIVE_API_SUPPORT>",
         "-H:CStandard=C11",
         "-H:+SpawnIsolates",
     ],
@@ -866,7 +864,6 @@ mx_sdk.register_graalvm_component(mx_sdk.GraalVmJreComponent(
         "substratevm:POLYGLOT_NATIVE_API",
     ],
     polyglot_lib_build_dependencies=[
-        "substratevm:POLYGLOT_NATIVE_API_SUPPORT",
         "substratevm:POLYGLOT_NATIVE_API_HEADERS"
     ],
     has_polyglot_lib_entrypoints=True,
```

```
diff --git a/substratevm/mx.substratevm/suite.py b/substratevm/mx.substratevm/suite.py
index 90216dcdc43..9a109ae13f4 100644
--- a/substratevm/mx.substratevm/suite.py
+++ b/substratevm/mx.substratevm/suite.py
@@ -61,16 +61,7 @@ suite = {
             "workingSets": "SVM",
         },

-        "com.oracle.svm.core.jdk8": {
-            "subDir": "src",
-            "sourceDirs": ["src"],
-            "dependencies": ["com.oracle.svm.core"],
-            "javaCompliance": "8",
-            "checkstyle": "com.oracle.svm.core",
-            "workingSets": "SVM",
-        },
-
-        "com.oracle.svm.core.jdk9": {
+        "com.oracle.svm.core.jdk11": {
             "subDir": "src",
             "sourceDirs": ["src"],
             "dependencies": ["com.oracle.svm.core"],
@@ -590,8 +581,7 @@ suite = {
                 "com.oracle.svm.hosted",
                 "com.oracle.svm.truffle.nfi",
                 "com.oracle.svm.core",
-                "com.oracle.svm.core.jdk8",
-                "com.oracle.svm.core.jdk9",
+                "com.oracle.svm.core.jdk11",
                 "com.oracle.svm.core.posix",
                 "com.oracle.svm.core.windows",
                 "com.oracle.svm.core.genscavenge",
@@ -758,41 +748,6 @@ suite = {
         },

-        "POLYGLOT_NATIVE_API_SUPPORT" : {
-            "native" : True,
-            "platformDependent" : True,
-            "description" : "polyglot.nativeapi support distribution for the GraalVM",
```

```
diff --git a/vm/mx.vm/suite.py b/vm/mx.vm/suite.py
index b599764e26c..8046888ed94 100644
--- a/vm/mx.vm/suite.py
+++ b/vm/mx.vm/suite.py
@@ -90,13 +90,13 @@ suite = {
             "truffle:TRUFFLE_API",
         ],
         "checkstyleVersion" : "8.8",
-        "javaCompliance": "1.8",
+        "javaCompliance": "1.8+",
         "license": "GPLv2-CPE",
     },
     "org.graalvm.component.installer" : {
         "subDir" : "src",
         "sourceDirs" : ["src"],
-        "javaCompliance" : "1.8",
+        "javaCompliance" : "1.8+",
         "checkstyle": "com.oracle.graalvm.locator",
         "license" : "GPLv2-CPE",
     },
@@ -107,7 +107,7 @@ suite = {
             "org.graalvm.component.installer"
         ],
-        "javaCompliance" : "1.8",
+        "javaCompliance" : "1.8+",
         "checkstyle": "com.oracle.graalvm.locator",
         "license" : "GPLv2-CPE",
     },
```

```
-1483,29 +1453,16 @@ suite = {
    "workingSets" : "Graal,Truffle",
  },

  "org.graalvm.compiler.truffle.runtime.serviceprovider.jdk8" : {
  "org.graalvm.compiler.truffle.runtime.serviceprovider.jdk11" : {
    "subDir" : "src",
    "sourceDirs" : ["src"],
    "dependencies" : [
      "org.graalvm.compiler.truffle.runtime.serviceprovider",
    ],
    "overlayTarget" : "org.graalvm.compiler.truffle.runtime.serviceprovider",
    "checkstyle" : "org.graalvm.compiler.graph",
    "javaCompliance" : "8",
    "checkPackagePrefix" : "false",
    "workingSets" : "Graal,Truffle",
-   },
-
-   "org.graalvm.compiler.truffle.runtime.serviceprovider.jdk9" : {
-     "subDir" : "src",
-     "sourceDirs" : ["src"],
-     "dependencies" : [
-       "org.graalvm.compiler.truffle.runtime.serviceprovider",
-     ],
-     "checkstyle" : "org.graalvm.compiler.graph",
-     "javaCompliance" : "9+",
+     "javaCompliance" : "11+",
      "checkPackagePrefix" : "false",
-     "multiReleaseJarVersion" : "9",
+     "multiReleaseJarVersion" : "11",
      "workingSets" : "Graal,Truffle",
    },

@@ -1871,7 +1828,7 @@ suite = {
      "subDir" : "src",
      "dependencies" : [
        "org.graalvm.compiler.truffle.runtime.serviceprovider",
-       "org.graalvm.compiler.truffle.runtime.serviceprovider.jdk9",
+       "org.graalvm.compiler.truffle.runtime.serviceprovider.jdk11",
        "org.graalvm.compiler.truffle.runtime",
```

Still failed to build GraalVM with JDK 11 on ARM64

```
2018-12-04 10:19:04     XDG_SESSION_ID=2
2018-12-04 10:19:04     XDG_VTNR=7
2018-12-04 10:19:04 make \
2018-12-04 10:19:04  MX_VERBOSE=y \
2018-12-04 10:19:04  -f \
2018-12-04 10:19:04  /opt/MyWorkSpace/MyProjs/Runtime/GraalVM/graal/sulong/projects/com.oracle.truffle.llvm.libraries.bitcode/Makefil
e \
2018-12-04 10:19:04  -j \
2018-12-04 10:19:04  6
2018-12-04 10:19:04 [8443: started subprocess 8450: ['make', 'MX_VERBOSE=y', '-f', '/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/graal/su
long/projects/com.oracle.truffle.llvm.libraries.bitcode/Makefile', '-j', '6']]
2018-12-04 10:19:05 clang -c -emit-llvm -o bin/stat.noopt.bc /opt/MyWorkSpace/MyProjs/Runtime/GraalVM/graal/sulong/projects/com.oracl
e.truffle.llvm.libraries.bitcode/src/clock.c:37:3: error: invalid output constraint '=a' in asm
  __SYSCALL_2P(SYS_clock_gettime, clk_id, tp);
  ^
/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/graal/sulong/projects/com.oracle.truffle.llvm.libraries.bitcode/src/syscall.h:67:5: note: ex
panded from macro '__SYSCALL_2P'
    __SYSCALL_2(result, id, a1, a2);
                \
    ^
/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/graal/sulong/projects/com.oracle.truffle.llvm.libraries.bitcode/src/syscall.h:36:70: note: e
xpanded from macro '__SYSCALL_2'
#define __SYSCALL_2(result, id, a1, a2) __asm__ volatile("syscall" : "=a"(result) : "a"(id), "D"(a1), "S"(a2) : "memory", "rcx", "r11
");
                                                                     ^
/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/graal/sulong/projects/com.oracle.truffle.llvm.libraries.bitcode/src/exit.c:95:3: error: inva
lid output constraint '=a' in asm
  __SYSCALL_1(result, SYS_exit_group, status);
  ^
/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/graal/sulong/projects/com.oracle.truffle.llvm.libraries.bitcode/src/syscall.h:34:66: note: e
xpanded from macro '__SYSCALL_1'
#define __SYSCALL_1(result, id, a1) __asm__ volatile("syscall" : "=a"(result) : "a"(id), "D"(a1) : "memory", "rcx", "r11");
                                                                 ^
/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/graal/sulong/projects/com.oracle.truffle.llvm.libraries.bitcode/src/exit.c:97:5: error: inva
lid output constraint '=a' in asm
    __SYSCALL_1(result, SYS_exit_group, status);
    ^
/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/graal/sulong/projects/com.oracle.truffle.llvm.libraries.bitcode/src/syscall.h:34:1 error gen
erated.
66: note: expanded from macro '__SYSCALL_1'
#define __SYSCALL_1(result, id, a1) __asm__ volatile("syscall" : "=a"(result) : "a"(id), "D"(a1) : "memory", "rcx", "r11");
```

X64

```
modified:   compiler/mx.compiler/suite.py
modified:   substratevm/mx.substratevm/suite.py
modified:   sulong/mx.sulong/suite.py
modified:   sulong/projects/com.oracle.truffle.llvm.runtime/src/com/oracle/truffle/llvm/runtime/floating/LLVM80BitFloat.java
modified:   vm/mx.vm/mx_vm.py
modified:   vm/mx.vm/suite.py
```

```
[loading /modules/java.base/java/lang/Override.class]
[wrote /opt/MyWorkSpace/MyProjs/Java/GraalVM/Official/graal/sulong/mxbuild/projects/com.oracle.truffle.llvm.runtime/bin/com/oracle/truffle/llvm/runtime/nodes/api/LLVMObjectAccess.class]
[checking com.oracle.truffle.llvm.runtime.types.PrimitiveType]
    return _compile(pattern, flags).sub(repl, string, count)
  File "/opt/MyWorkSpace/DevSW/Tools/Build/mx/mx_subst.py", line 86, in <lambda>
    return re.sub(r'<([\w\-]+?)(:(.+?))?>', lambda m: self._replace(m, self.skip_unknown_substitutions, **kwArgs), string)
  File "/opt/MyWorkSpace/DevSW/Tools/Build/mx/mx_subst.py", line 75, in _replace
    return fn()
  File "/opt/MyWorkSpace/MyProjs/Java/GraalVM/Official/graal/vm/mx.vm/mx_vm.py", line 899, in _get_classpath
    return graalvm_home_relative_classpath(self.subject.native_image_jar_distributions, script_destination_directory, graal_vm=graal_vm)
  File "/opt/MyWorkSpace/MyProjs/Java/GraalVM/Official/graal/vm/mx.vm/mx_vm.py", line 973, in graalvm_home_relative_classpath
    jdk_location = relpath(_cp_entry.classpath_repr(jdk), jdk.home)
  File "/usr/lib64/python2.7/posixpath.py", line 428, in relpath
[loading /opt/MyWorkSpace/MyProjs/Java/GraalVM/Official/graal/truffle/mxbuild/dists/jdk11/truffle-api.jar(/com/oracle/truffle/api/nodes/Node$Child.class)]
    raise ValueError("no path specified")
ValueError: no path specified
[loading /modules/java.base/java/lang/SuppressWarnings.class]
```

[GR-12750] Ensure native-image image can be built with Java 11. ...

olpaw committed 20 hours ago

More Java 11 fixes for native-image building

olpaw committed 2 days ago

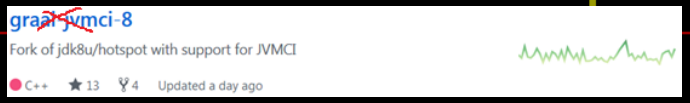
Provide Java 11 image building fixes ...

olpaw committed 8 days ago

...

Summary

- still not mature enough, hope to be much improved in 1.0 GA
- prone to break build while customizing
- remove restrictions on dependency of JDK 8 and moving to JDK 9+



troubleshooting…

- dynamically enable JVMCI and reload Graal compiler at runtime
- reduce build dependencies of Sulong, and upstreaming it

```
"libraries" : {
  "LLVM_TEST_SUITE" : {
    "packedResource" : True,
    "urls" : [
      "https://lafo.ssw.uni-linz.ac.at/pub/sulong-deps/test-suite-3.2.src.tar.gz",
      "http://llvm.org/releases/3.2/test-suite-3.2.src.tar.gz",
    ],
    "sha1" : "e370255ca2540bcd66f316fe5b96f459382f3e8a",
  },
  "GCC_SOURCE" : {
    "packedResource" : True,
    "urls" : [
      "https://lafo.ssw.uni-linz.ac.at/pub/sulong-deps/gcc-5.2.0.tar.gz",
      "http://gd.tuwien.ac.at/gnu/gcc/releases/gcc-5.2.0/gcc-5.2.0.tar.gz",
      "ftp://ftp.fu-berlin.de/unix/languages/gcc/releases/gcc-5.2.0/gcc-5.2.0.tar.gz",
      "http://mirrors-usa.go-parts.com/gcc/releases/gcc-5.2.0/gcc-5.2.0.tar.gz",
    ],
    "sha1" : "713211883406b3839bdba4a22e7111a0cff5d09b",
  },
```

- deprecated APIs

  https://docs.oracle.com/javase/10/docs/api/java.xml.bind-summary.html
- integrate novel debugging approaches into mx
- deal with mx, Java, JDK, Truffle/Graal, LLVM…
- make OpenJDK & GraalVM more developer/hacker friendly

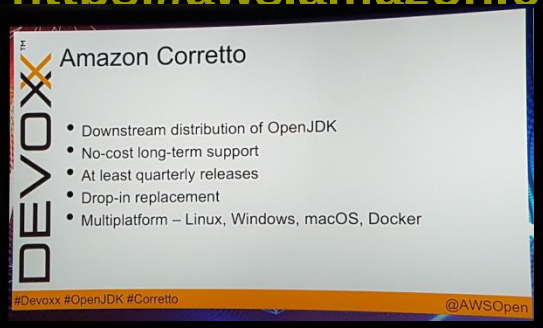
IV. Runtime

1) Rethinking

Customized

- https://software.intel.com/en-us/python-distribution

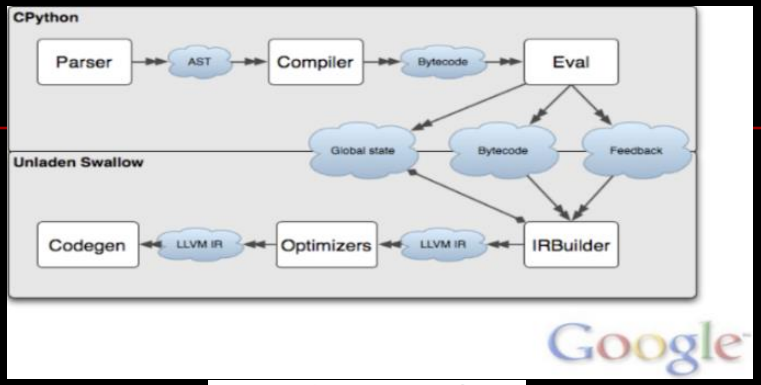
- https://aws.amazon.com/corretto/



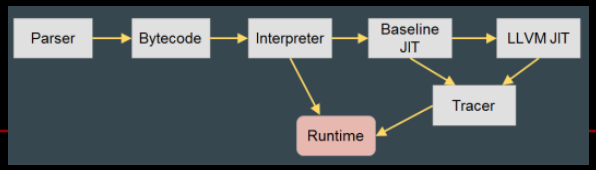
-

LLVM-based (VMKit, MCJIT…)

- Python

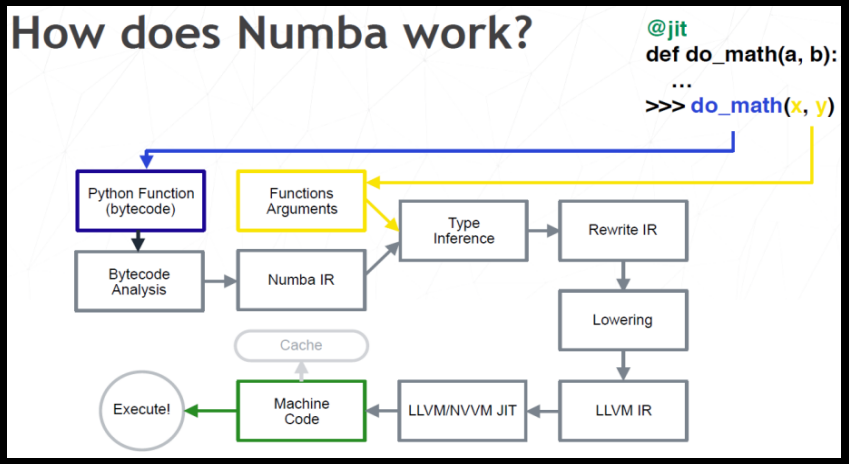


up to 10x speedups



PySton

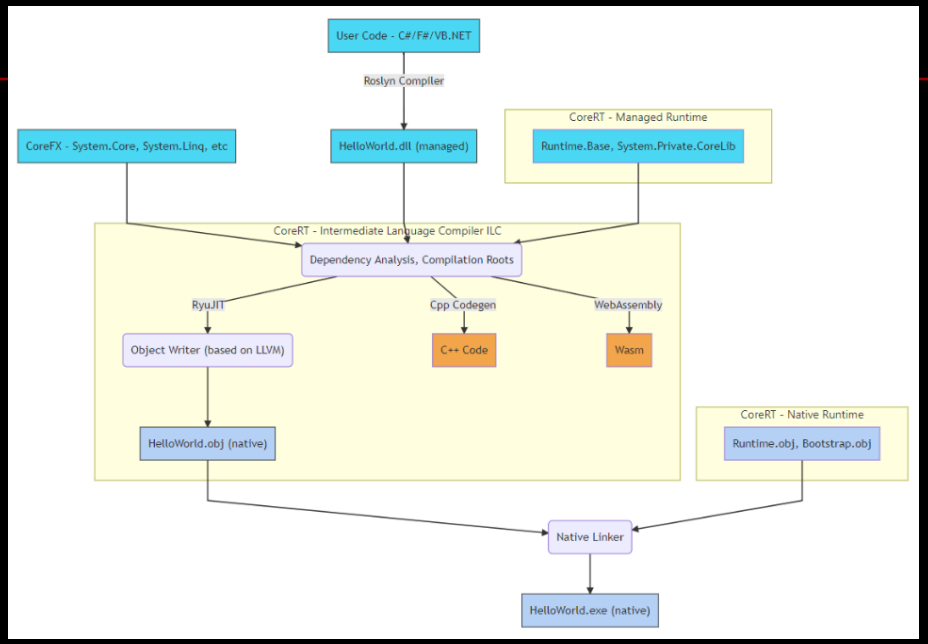
Dropbox

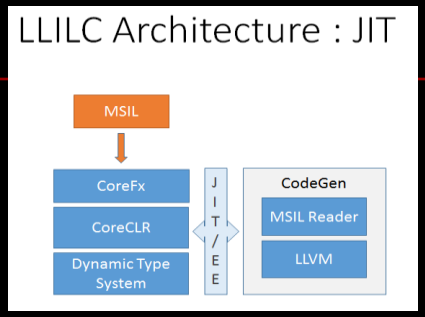


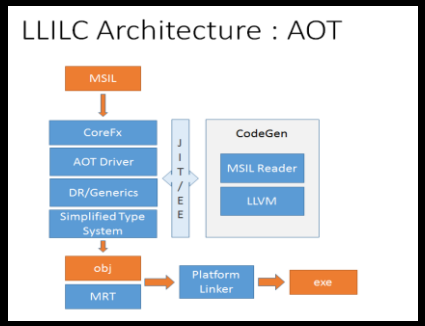
llvmlite is a project originally tailored for Numba's needs, using the following approach:

- A small C wrapper around the parts of the LLVM C++ API we need that are not already exposed by the LLVM C API.
- A ctypes Python wrapper around the C API.
- A pure Python implementation of the subset of the LLVM IR builder that we need for Numba.

- .Net
  https://github.com/dotnet/llilc
  https://github.com/dotnet/corert







- Java
  https://www.azul.com/products/zing/falcon-jit-compiler/

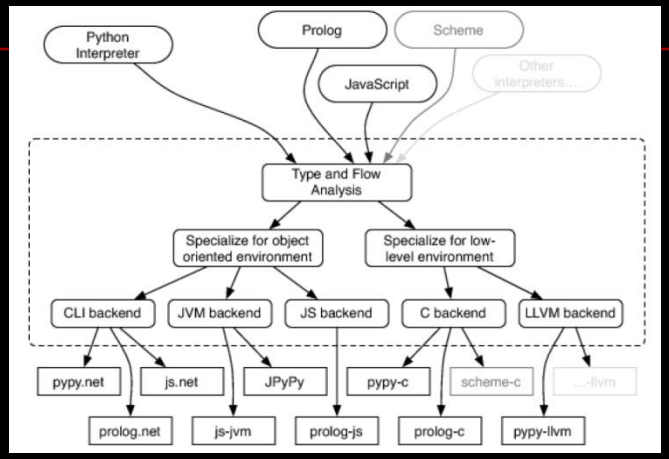
The performance advantages for Java workloads when running with Zing's new Falcon JIT compiler have significant business benefits:

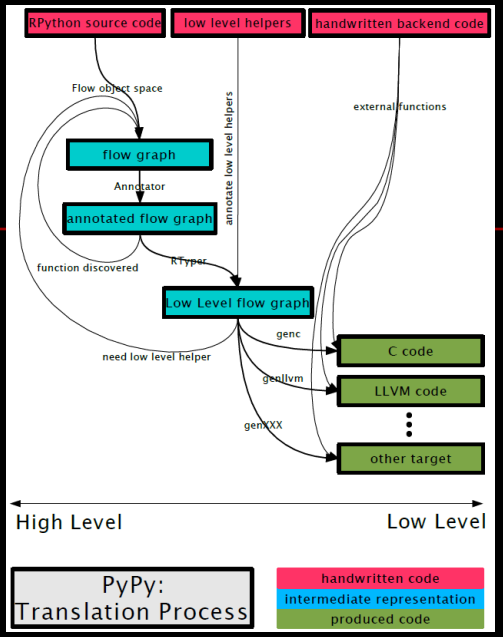
- Better application service level metrics: reduced latency, reduced timeouts, better consistency
- Better customer experience: reliably achieve customer expectations even under unpredictable load, improved customer satisfaction
- Lower operating costs: reduced memory footprint requirements, improved load carrying capacities on similarly sized instances in Cloud/AWS deployments
- Lower development costs: fewer engineering hours required to achieve desired performance, consistency and scale

Src2Src-based

- pypy

RPython

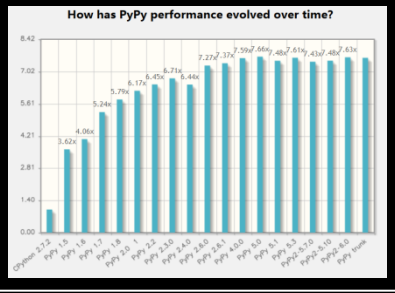




Meta-tracing

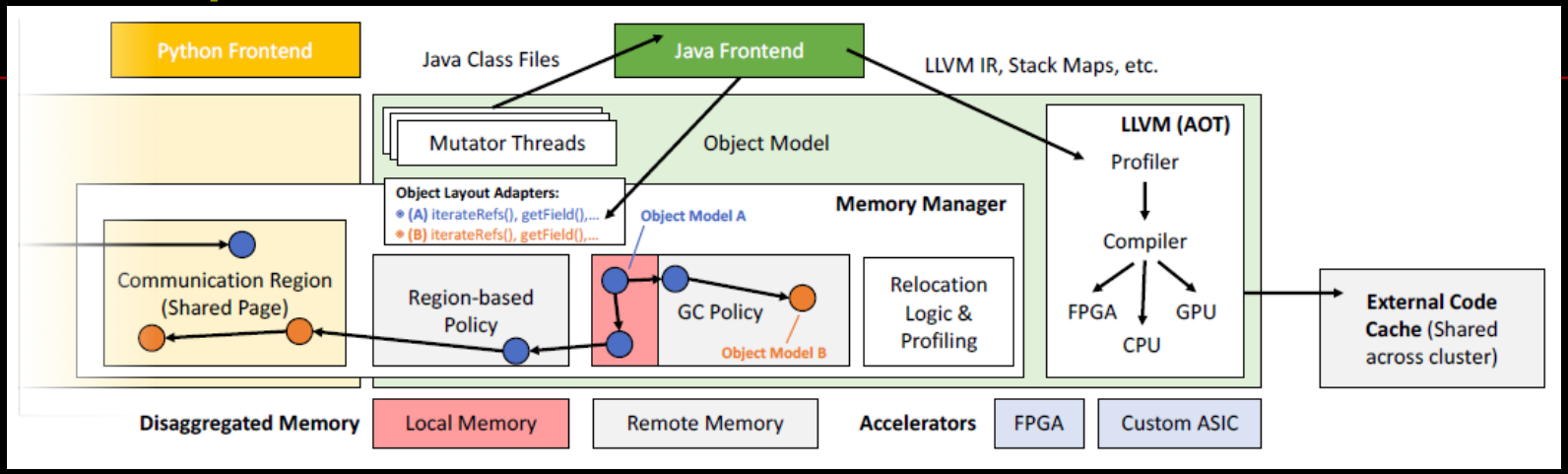
http://stefan-marr.de/papers/oopsla-marr-ducasse-meta-tracing-vs-partial-evaluation-artifacts/submitted-draft.pdf

speed.pypy.org



Cloud

- https://people.eecs.berkeley.edu/~maas/papers/maas-hotos17-cloud30.pdf



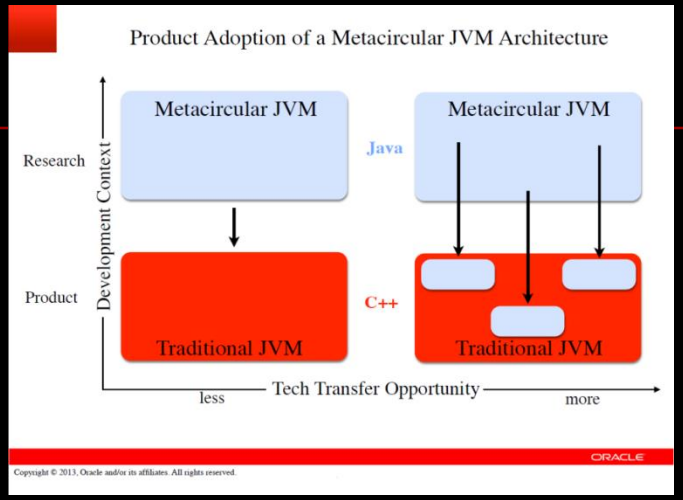
- Container-aware

  -XX:+UseContainerSupport (Linux only) since JDK 10

  OpenJ9 JIT memory auto-scaling based on container limits

Meta-Circular

- Maxine → GraalVM

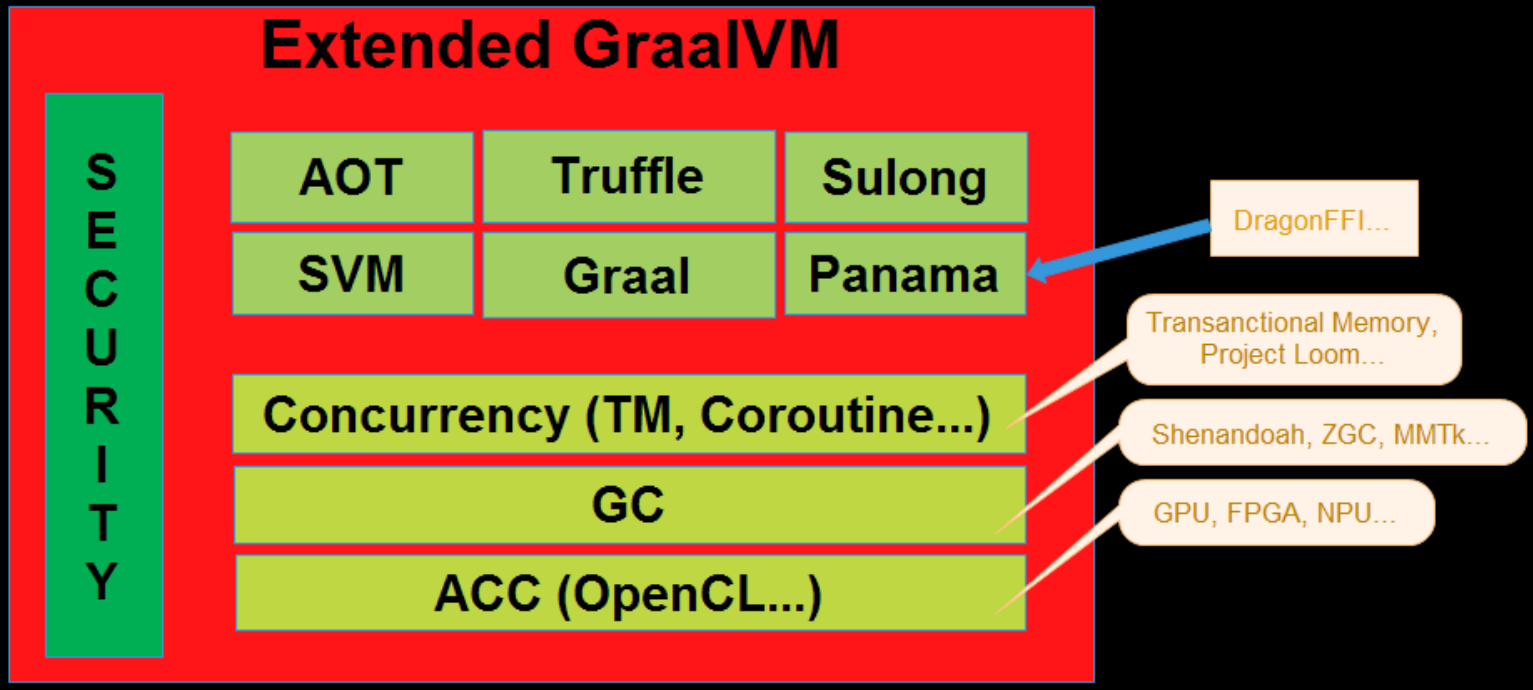


Source: https://chrisseaton.com/truffleruby/jokerconf17/

2) Redesign

- http://openjdk.java.net/projects/metropolis/
  Java on Java

- My redesign — extended GraalVM
  Base on JDK 12 — built-in support for ARM
  https://openjdk.java.net/jeps/8211652 //libgraal



V. Wrap-up

- Redesigning App Runtime -- User space/Kernel space Repartitioning & Unifying

  Rethinking the Language Runtime System for the Cloud 3.0 Era

VM Cloud 1.0 → Container Cloud 2.0 → LightVM/MicroVM Cloud 2.5 → Unikernel Cloud 3.0?

- Introducing Artificial Intelligence to Next Generation Runtime

Q & A

Thanks!

Reference

Slides/materials from many and varied sources:

- http://en.wikipedia.org/wiki/
- http://www.slideshare.net/

- https://medium.com/graalvm
- https://en.wikipedia.org/wiki/Java_virtual_machine
- https://chriswhocodes.com/hotspot_option_differences.html
- http://llvm.org/
- https://www.python.org
- https://github.com/dropbox/pyston
- https://en.wikipedia.org/wiki/Just-in-time_compilation
- https://en.wikipedia.org/wiki/Ahead-of-time_compilation
- https://en.wikipedia.org/wiki/Meta-circular_evaluator
- …