PyCon China
Hangzhou 2018

GraalPython
— a new Python runtime

Feng Li (李枫)
hkli2013@126.com
Nov 4，2018

Agenda

I. GraalVM
- Why Python
- Python & Java
- Overview of GraalVM

II. Dive Into GraalVM & GraalPython
- Truffle & Graal
- GraalPython

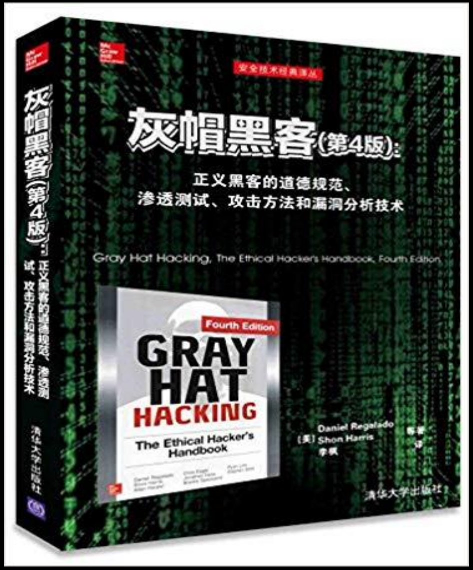
III. Rethinking Python Runtime
- Accelerating Python
- Redesign

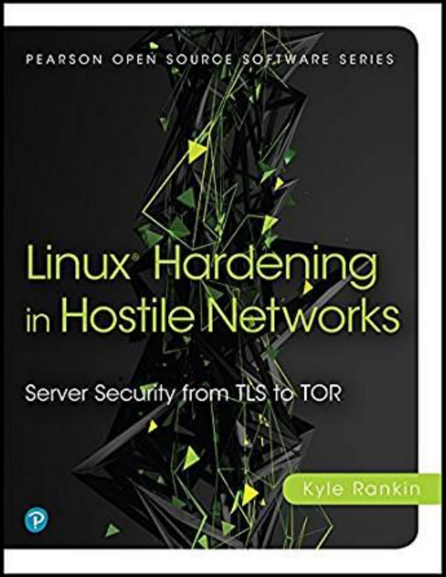
IV. GraalPython on ARM
- Development Env
- My Practice

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»


灰帽黑客(第4版):
正义黑客的道德规范、
渗透测试、攻击方法和漏洞分析技术
Gray Hat Hacking, The Ethical Hackers Handbook, Fourth Edition
Fourth Edition
GRAY HAT HACKING
The Ethical Hacker's Handbook
Daniel Regalado
[美] Shon Harris
清华大学出版社

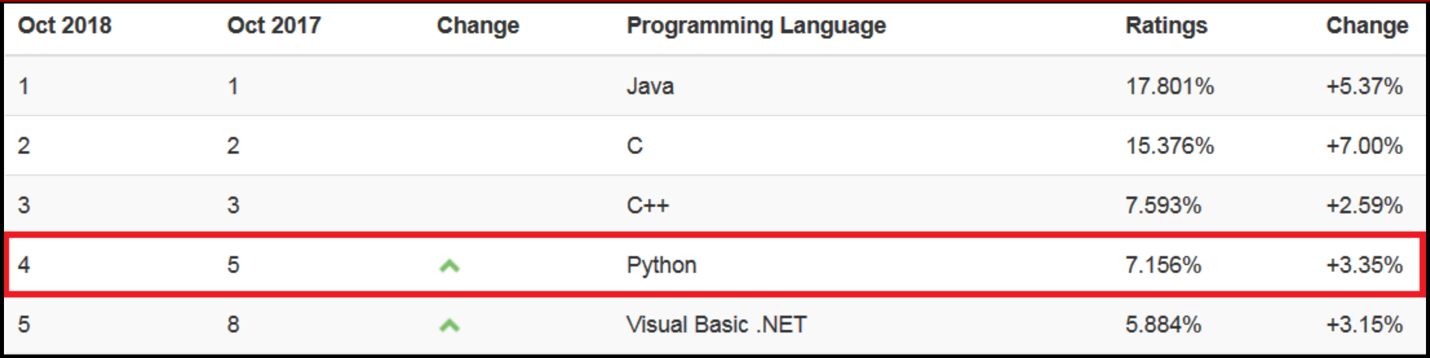

PEARSON OPEN SOURCE SOFTWARE SERIES
Linux Hardening in Hostile Networks
Server Security from TLS to TOR
Kyle Rankin

- Gave presentation at more than 10 Technology Conferences by now, especially for PyCon (e.g. PyCon China Hangzhou 2014…)
- In addition, took part in nearly 20 offline technical activities in Open Source Community which covers Chip, OS, Toolchain, Security, Blockchain, AI, and so on

I. GraalVM

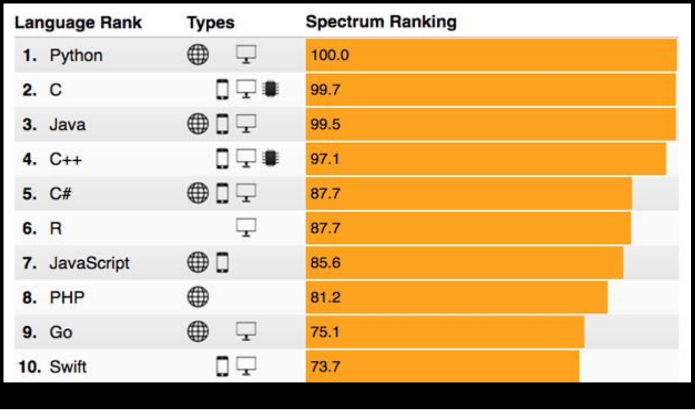
1) Why Python

- https://www.tiobe.com/tiobe-index/



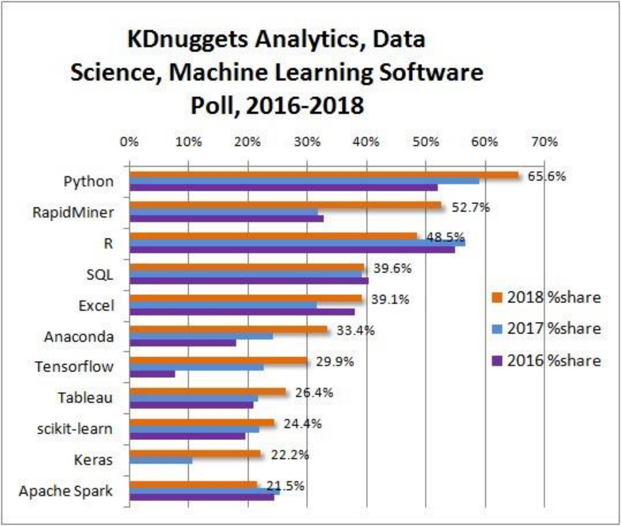
Oct 2018	Oct 2017	Change	Programming Language	Ratings	Change
1	1		Java	17.801%	+5.37%
2	2		C	15.376%	+7.00%
3	3		C++	7.593%	+2.59%
4	5	▲	Python	7.156%	+3.35%
5	8	▲	Visual Basic .NET	5.884%	+3.15%

- http://pypl.github.io/PYPL.html
- https://spectrum.ieee.org/computing/software/the-2017-top-programming-languages



Language Rank	Types	Spectrum Ranking
1. Python	🌐 🖥	100.0
2. C	📱 🖥 ▪	99.7
3. Java	🌐 📱 🖥	99.5
4. C++	📱 🖥 ▪	97.1
5. C#	🌐 📱 🖥	87.7
6. R	🖥	87.7
7. JavaScript	🌐 📱	85.6
8. PHP	🌐	81.2
9. Go	🌐 🖥	75.1
10. Swift	📱 🖥	73.7

- https://www.kdnuggets.com/2018/05/poll-tools-analytics-data-science-machine-learning-results.html



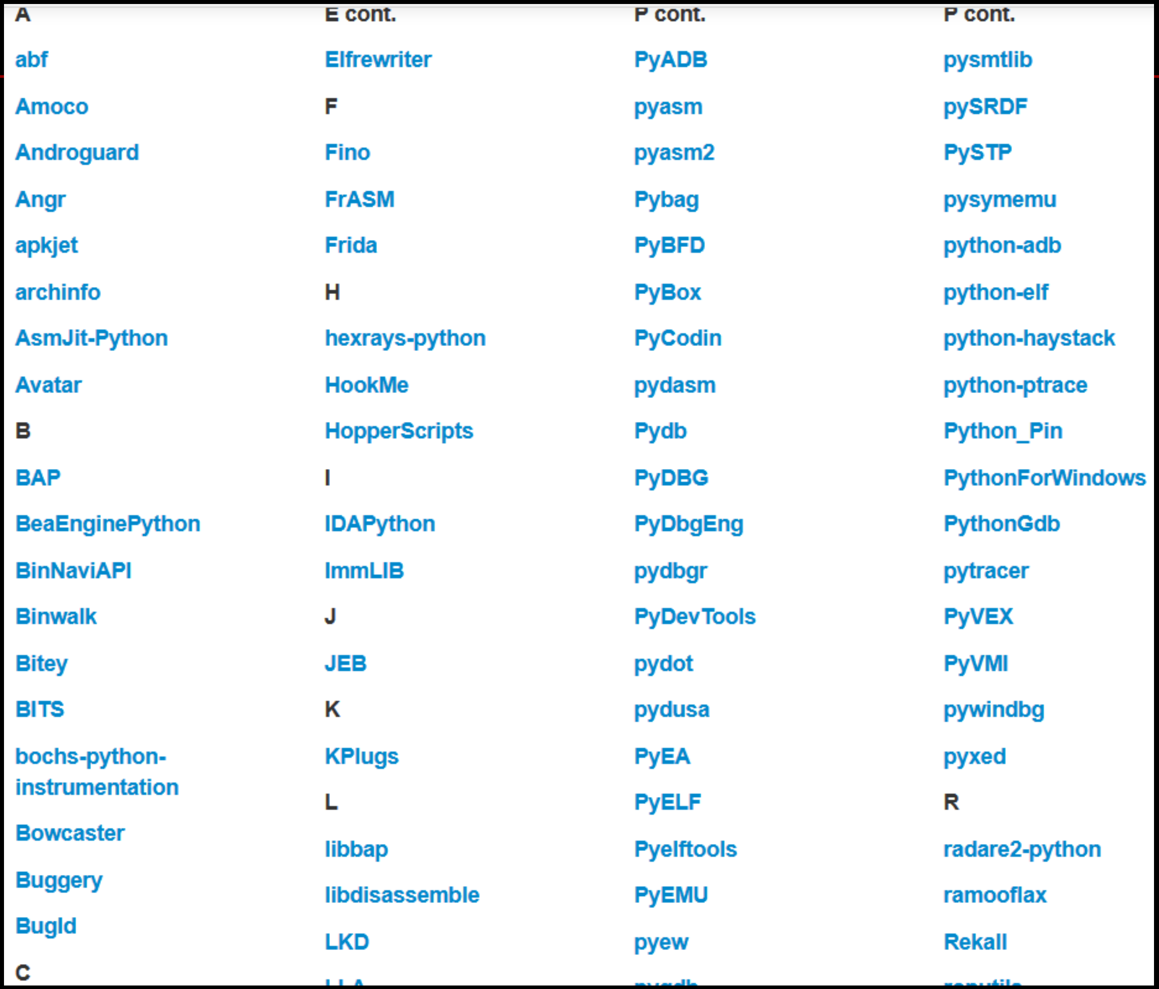
- **Famous Python projects**

  Build: **Meson, SCons…** **DevOps:** **Ansible, SaltStack…**
  Web: **Django, web2py, Flask, Tornado, Pylons, TurboGears, Quixote…**
  — Youtube, Quora, Reddit…
  AI: **PyTorch, Keras, Theano…**
  Big Data: **PyData, PySpark…**
  Science: **Scipy, Sage…**
  HPC: **Anaconda, PyCUDA…**
  Cloud/DataCenter: **OpenStack…**

  …

Security

- a Swiss Army Knife for hackers…
- http://www.pythonarsenal.com/



A

abf

Amoco

Androguard

Angr

apkjet

archinfo

AsmJit-Python

Avatar

B

BAP

BeaEnginePython

BinNaviAPI

Binwalk

Bitey

BITS

bochs-python-instrumentation

Bowcaster

Buggery

BugId

C

E cont.

Elfrewriter

F

Fino

FrASM

Frida

H

hexrays-python

HookMe

HopperScripts

I

IDAPython

ImmLIB

J

JEB

K

KPlugs

L

libbap

libdisassemble

LKD

LLA

P cont.

PyADB

pyasm

pyasm2

Pybag

PyBFD

PyBox

PyCodin

pydasm

Pydb

PyDBG

PyDbgEng

pydbgr

PyDevTools

pydot

pydusa

PyEA

PyELF

Pyelftools

PyEMU

pyew

pygdb

P cont.

pysmtlib

pySRDF

PySTP

pysymemu

python-adb

python-elf

python-haystack

python-ptrace

Python_Pin

PythonForWindows

PythonGdb

pytracer

PyVEX

PyVMI

pywindbg

pyxed

R

radare2-python

ramooflax

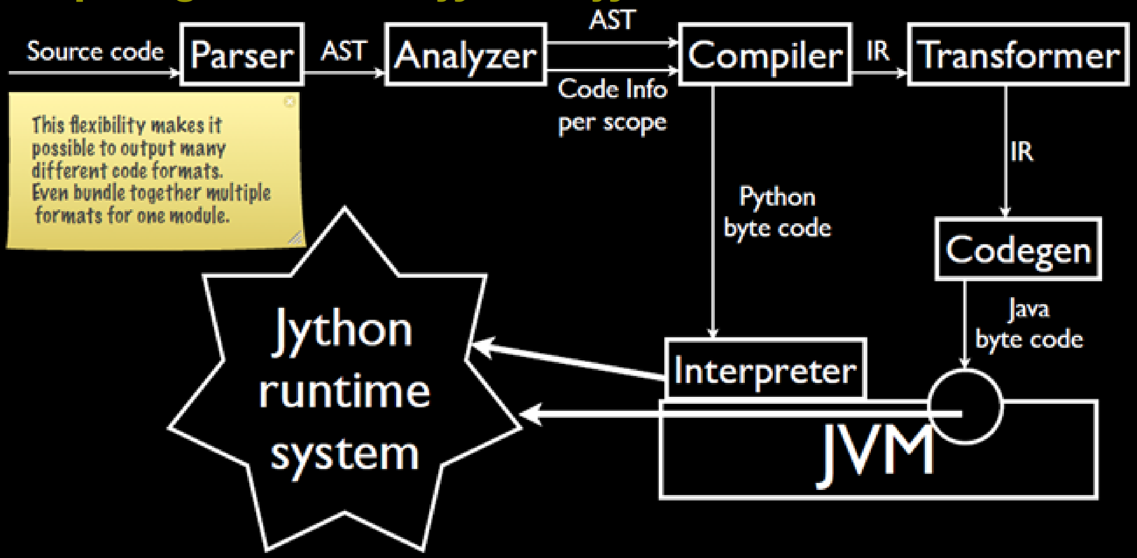
Rekall

…

2) Python & Java

Jython

- http://www.jython.org //No new release since 2015…
- https://github.com/jythontools/jython.git
- https://github.com/jython/jython3



This flexibility makes it possible to output many different code formats. Even bundle together multiple formats for one module.

VOC

- https://github.com/pybee/voc/
- A transpiler that converts Python code into Java bytecode

…

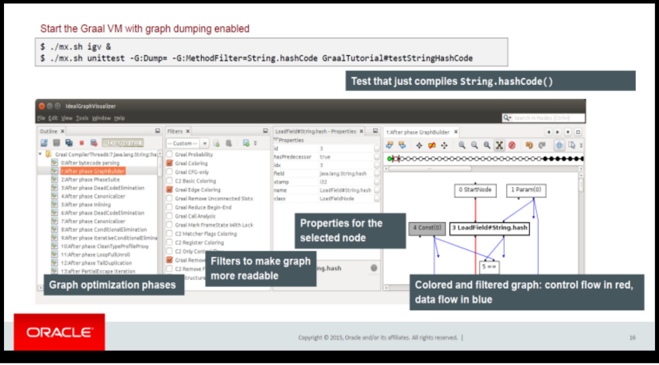
Python for JDK development

Mercurial

- http://hg.openjdk.java.net/
- https://blogs.oracle.com/kto/entry/mercurial_openjdk_questions
- ~~On December 2007, Sun moved the revision control of OpenJDK from~~ TeamWare to Mercurial…

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



Start the Graal VM with graph dumping enabled

```
$ ./mx.sh igv &
$ ./mx.sh unittest -G:Dump= -G:MethodFilter=String.hashCode GraalTutorial#testStringHashCode
```

Test that just compiles String.hashCode()

Properties for the selected node

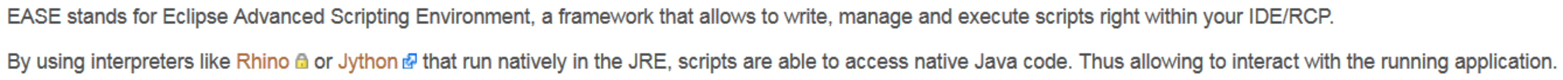
Filters to make graph more readable

Graph optimization phases

Colored and filtered graph: control flow in red, data flow in blue

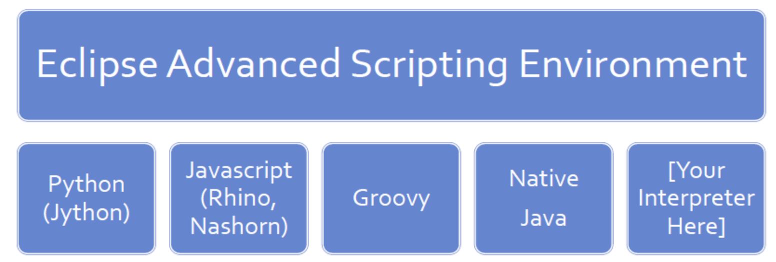
Eclipse Advanced Scripting Environment

- https://wiki.eclipse.org/EASE

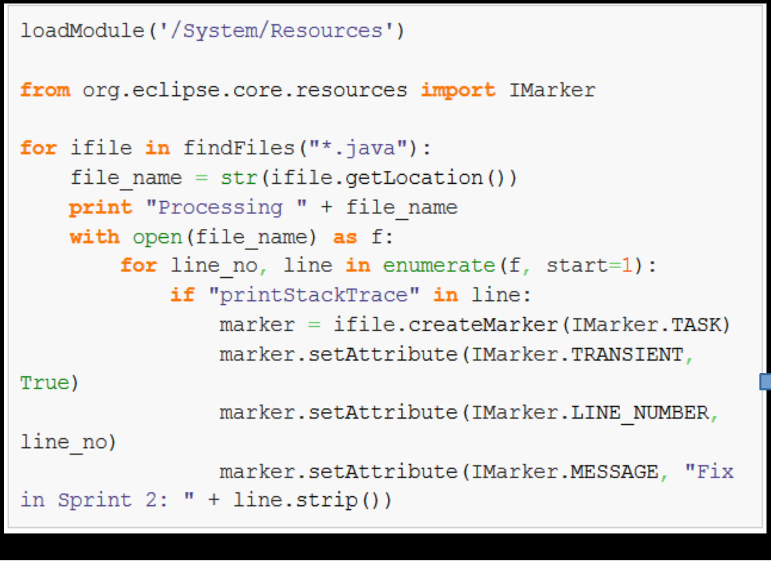


EASE stands for Eclipse Advanced Scripting Environment, a framework that allows to write, manage and execute scripts right within your IDE/RCP.

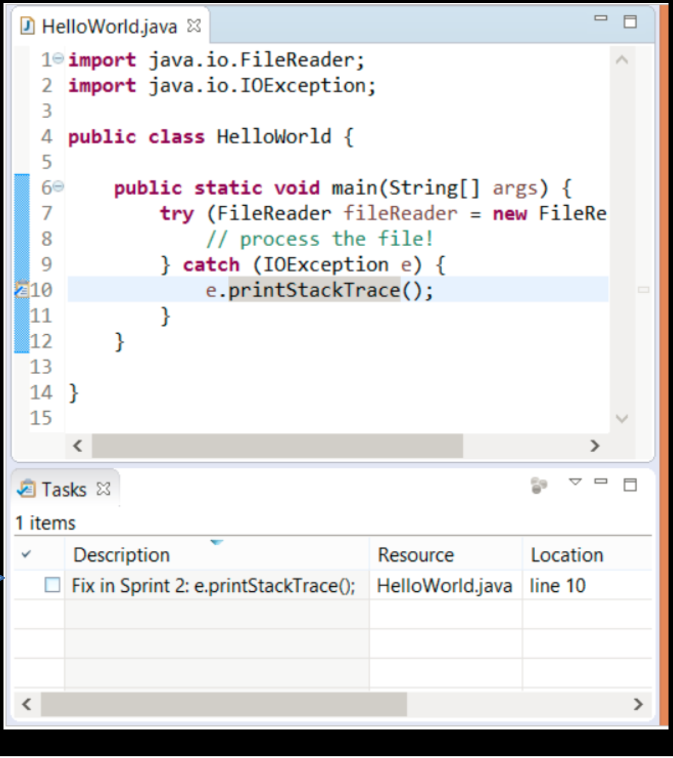
By using interpreters like Rhino or Jython that run natively in the JRE, scripts are able to access native Java code. Thus allowing to interact with the running application.


Eclipse Advanced Scripting Environment
Python (Jython)
Javascript (Rhino, Nashorn)
Groovy
Native Java
[Your Interpreter Here]

- Hack your IDE with Python & EASE



```
loadModule('/System/Resources')

from org.eclipse.core.resources import IMarker

for ifile in findFiles("*.java"):
    file_name = str(ifile.getLocation())
    print "Processing " + file_name
    with open(file_name) as f:
        for line_no, line in enumerate(f, start=1):
            if "printStackTrace" in line:
                marker = ifile.createMarker(IMarker.TASK)
                marker.setAttribute(IMarker.TRANSIENT,
True)
                marker.setAttribute(IMarker.LINE_NUMBER,
line_no)
                marker.setAttribute(IMarker.MESSAGE, "Fix
in Sprint 2: " + line.strip())
```



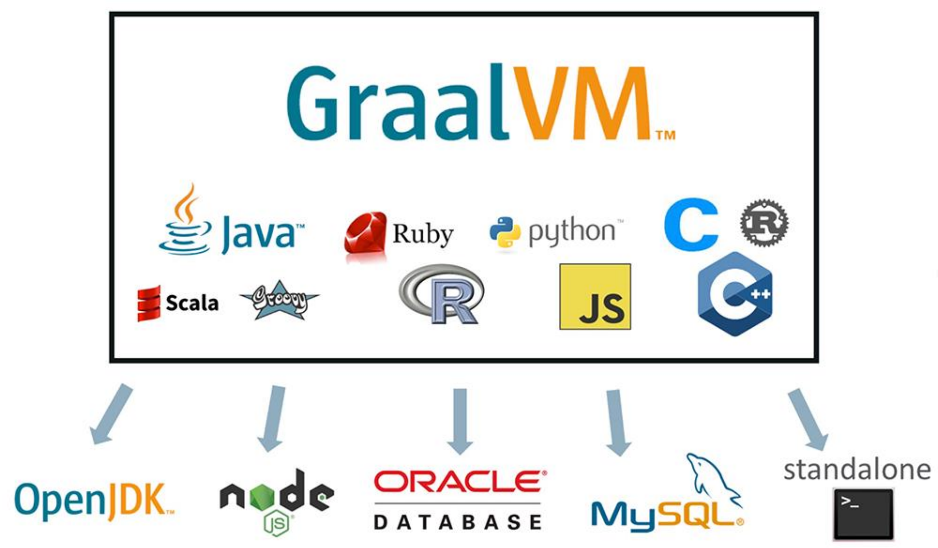
```
import java.io.FileReader;
import java.io.IOException;

public class HelloWorld {

    public static void main(String[] args) {
        try (FileReader fileReader = new FileRe
            // process the file!
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

}
```

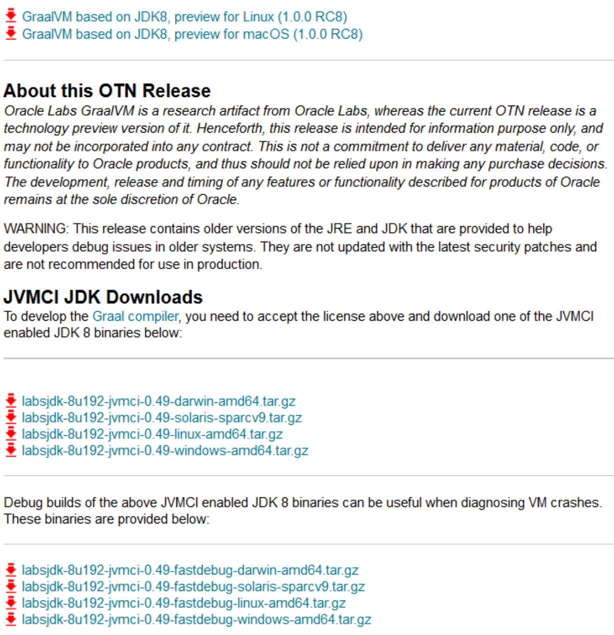
1 items

✓	Description	Resource	Location
□	Fix in Sprint 2: e.printStackTrace();	HelloWorld.java	line 10

3) Overview of GraalVM

- https://www.graalvm.org/
- http://www.oracle.com/technetwork/oracle-labs/program-languages/overview/index.html
- https://blogs.oracle.com/developers/announcing-graalvm





- GraalVM based on JDK8, preview for Linux (1.0.0 RC8)
- GraalVM based on JDK8, preview for macOS (1.0.0 RC8)

About this OTN Release

Oracle Labs GraalVM is a research artifact from Oracle Labs, whereas the current OTN release is a technology preview version of it. Henceforth, this release is intended for information purpose only, and may not be incorporated into any contract. This is not a commitment to deliver any material, code, or functionality to Oracle products, and thus should not be relied upon in making any purchase decisions. The development, release and timing of any features or functionality described for products of Oracle remains at the sole discretion of Oracle.

WARNING: This release contains older versions of the JRE and JDK that are provided to help developers debug issues in older systems. They are not updated with the latest security patches and are not recommended for use in production.

JVMCI JDK Downloads

To develop the Graal compiler, you need to accept the license above and download one of the JVMCI enabled JDK 8 binaries below:

- labsjdk-8u192-jvmci-0.49-darwin-amd64.tar.gz
- labsjdk-8u192-jvmci-0.49-solaris-sparcv9.tar.gz
- labsjdk-8u192-jvmci-0.49-linux-amd64.tar.gz
- labsjdk-8u192-jvmci-0.49-windows-amd64.tar.gz

Debug builds of the above JVMCI enabled JDK 8 binaries can be useful when diagnosing VM crashes. These binaries are provided below:

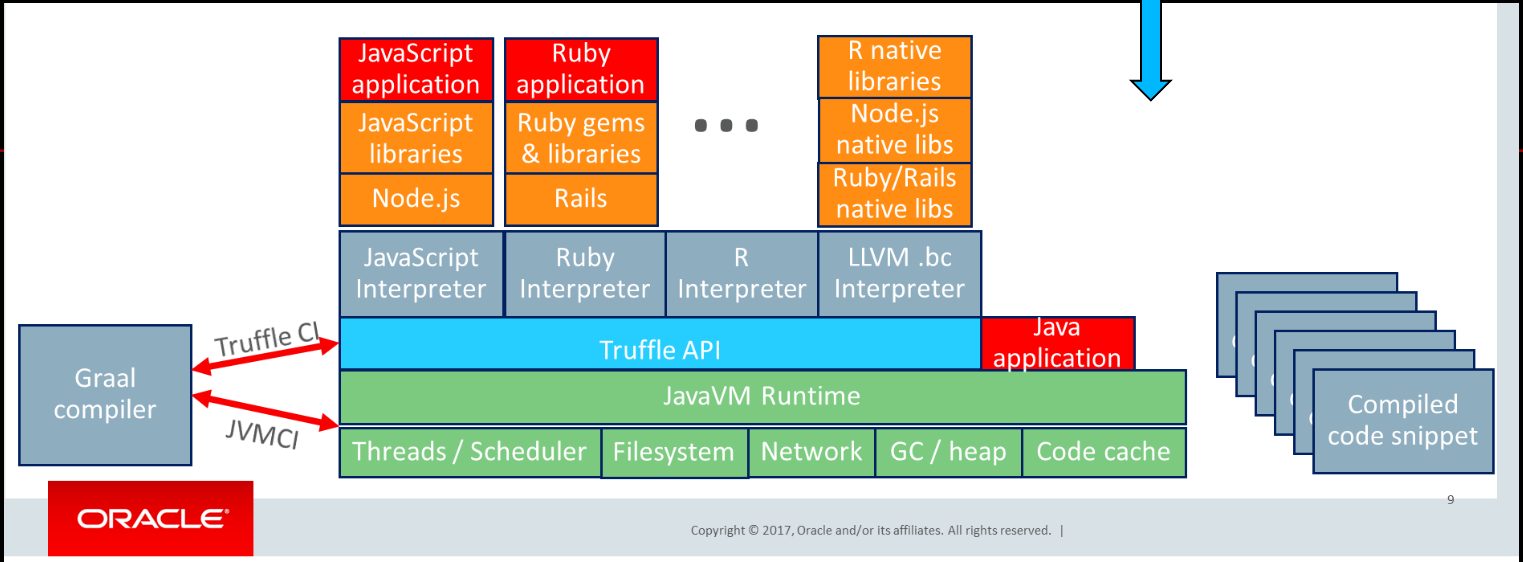
- labsjdk-8u192-jvmci-0.49-fastdebug-darwin-amd64.tar.gz
- labsjdk-8u192-jvmci-0.49-fastdebug-solaris-sparcv9.tar.gz
- labsjdk-8u192-jvmci-0.49-fastdebug-linux-amd64.tar.gz
- labsjdk-8u192-jvmci-0.49-fastdebug-windows-amd64.tar.gz

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

Arch

- A hybrid of static & dynamic runtimes

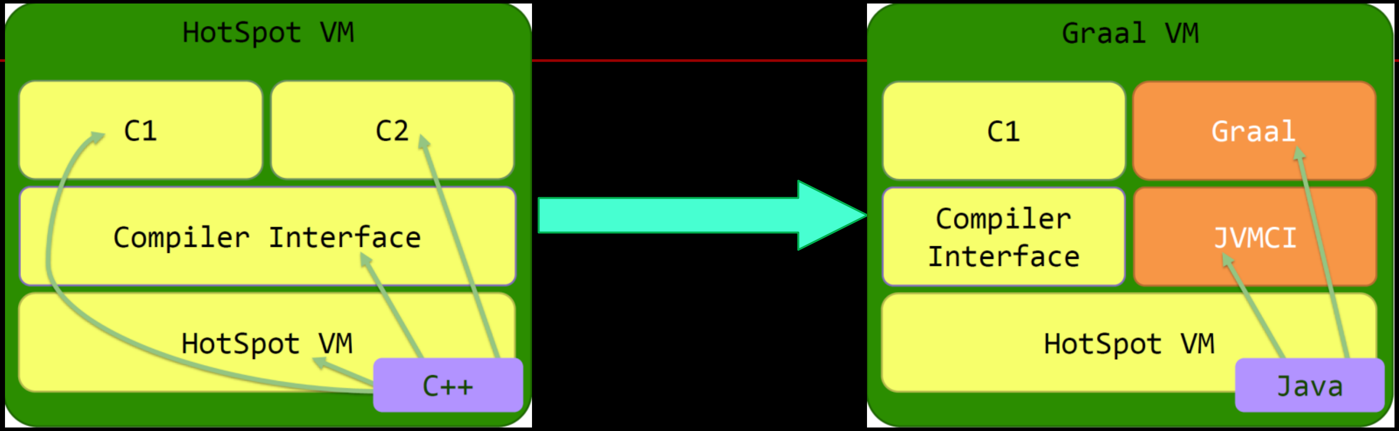
Substrate VM



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

JVMCI

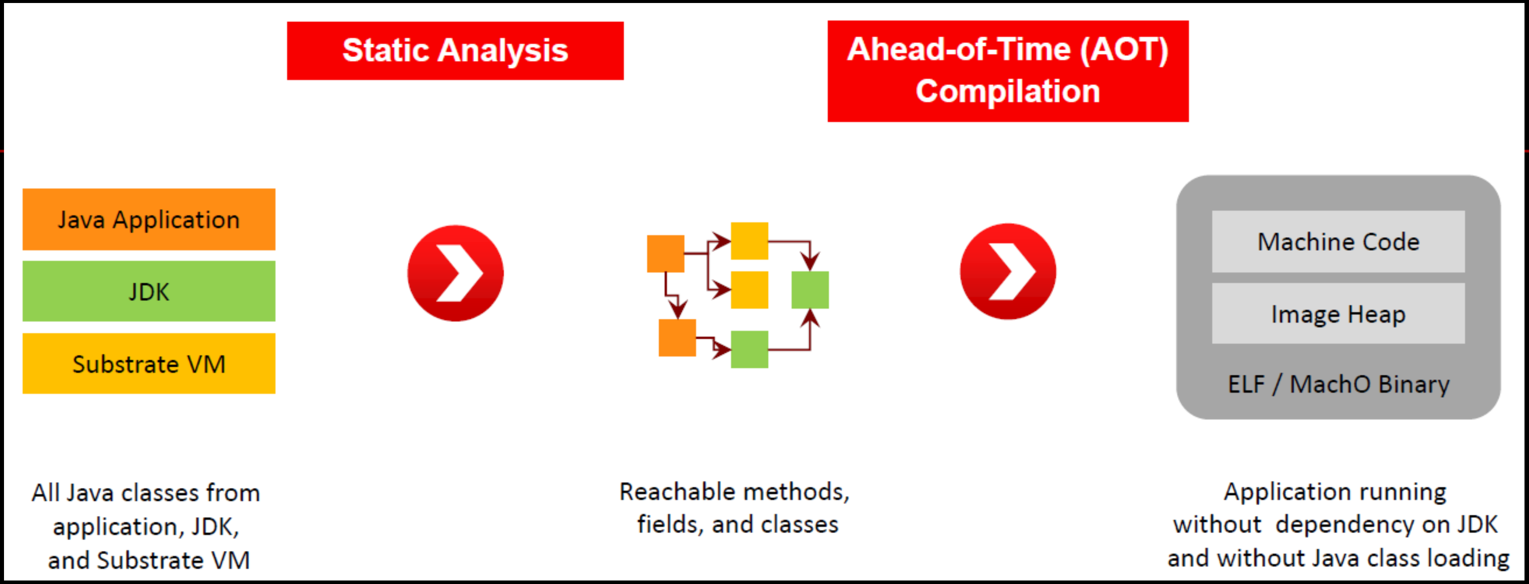
- Java-Level JVM Compiler Interface
- http://openjdk.java.net/jeps/243: experimental in JDK 9



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

Substrate VM

- Native Image Generation



Static Analysis

Ahead-of-Time (AOT) Compilation

Java Application

JDK

Substrate VM

Machine Code

Image Heap

ELF / MachO Binary

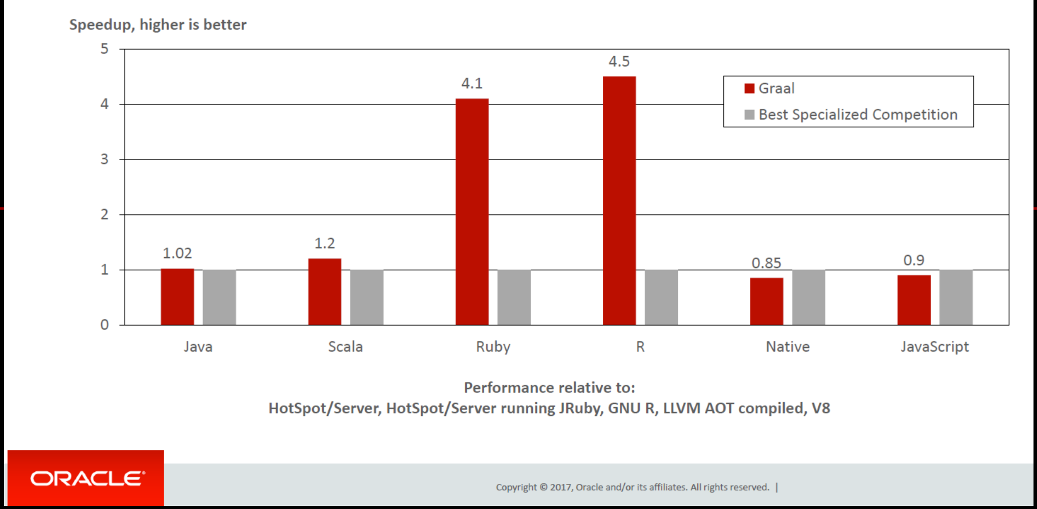
All Java classes from
application, JDK,
and Substrate VM

Reachable methods,
fields, and classes

Application running
without dependency on JDK
and without Java class loading

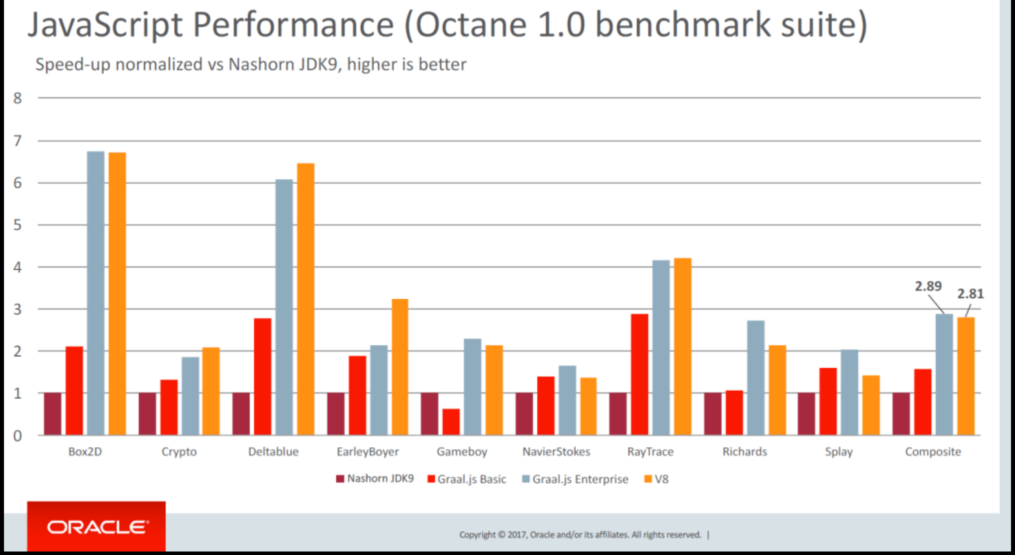
Source: https://static.rainfocus.com/oracle/oow18/sess/1526041579721001pM2J/PF/
-2018-10-24%20SubstrateVM%20CodeOne_1540478815946001 9swO.pdf

Performance



Performance relative to:
HotSpot/Server, HotSpot/Server running JRuby, GNU R, LLVM AOT compiled, V8

Source: http://lafo.ssw.uni-linz.ac.at/papers/2017_PLDI_GraalTutorial.pdf

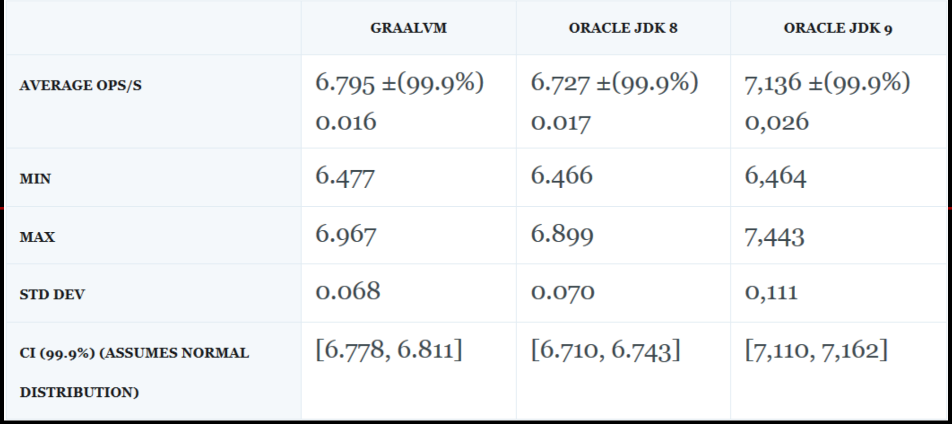


JavaScript Performance (Octane 1.0 benchmark suite)

Speed-up normalized vs Nashorn JDK9, higher is better

Source: http://dbpl2017.org/slides/DBPL-2017-s2.pdf

- **but for GraalVM 1.0.0 RC1**

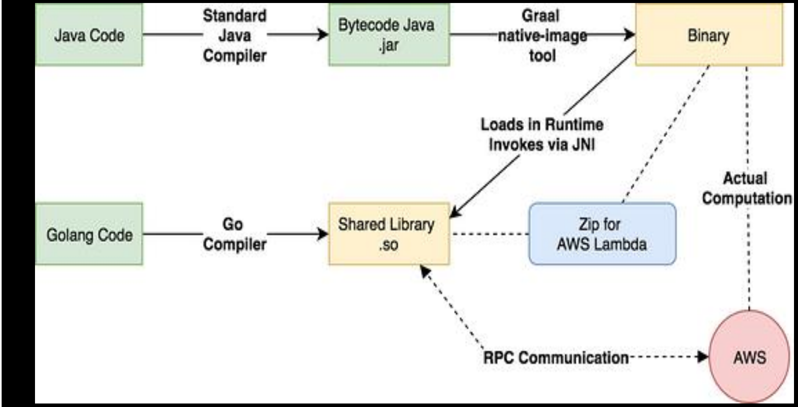


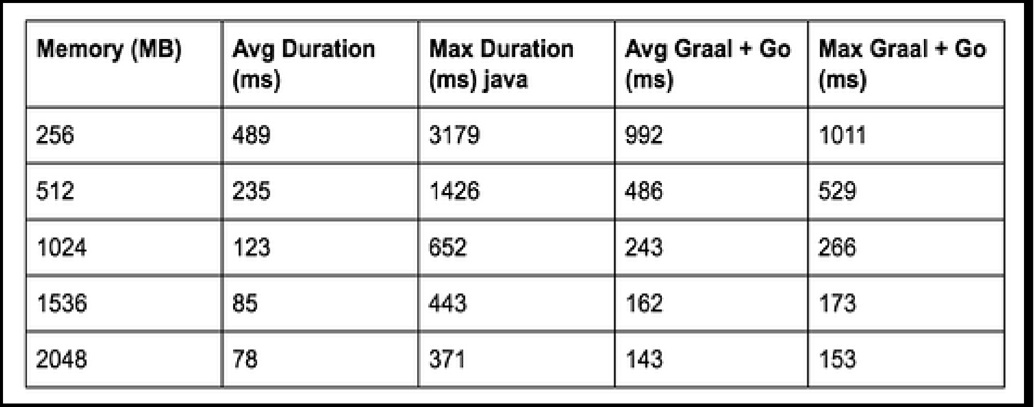
	GRAALVM	ORACLE JDK 8	ORACLE JDK 9
AVERAGE OPS/S	6.795 ±(99.9%) 0.016	6.727 ±(99.9%) 0.017	7,136 ±(99.9%) 0,026
MIN	6.477	6.466	6,464
MAX	6.967	6.899	7,443
STD DEV	0.068	0.070	0,111
CI (99.9%) (ASSUMES NORMAL DISTRIBUTION)	[6.778, 6.811]	[6.710, 6.743]	[7,110, 7,162]

Source: https://blog.frankel.ch/first-impressions-graalvm

- **Real World Apps:**

Using GraalVM to run Native Java in AWS Lambda with Golang



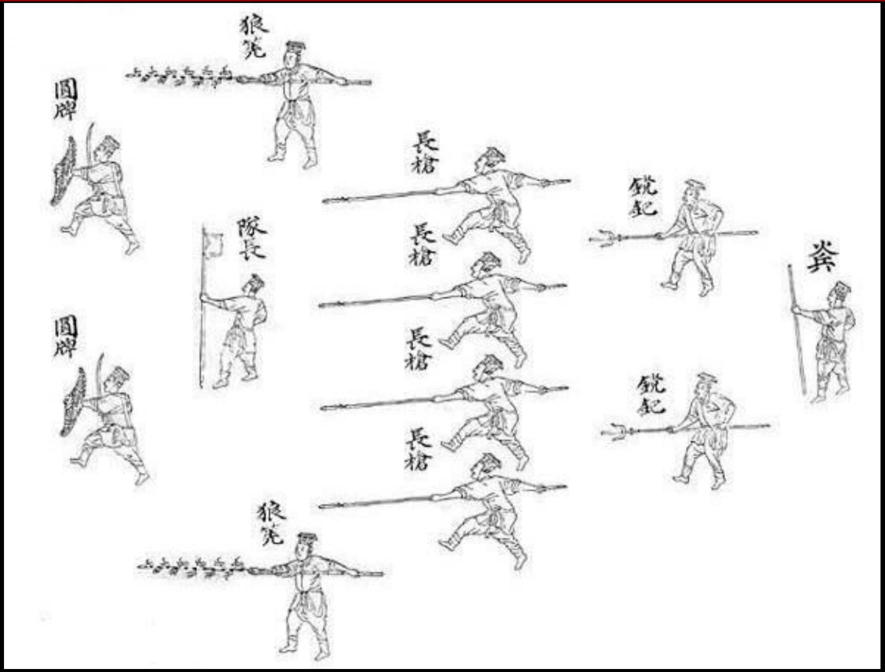


Memory (MB)	Avg Duration (ms)	Max Duration (ms) java	Avg Graal + Go (ms)	Max Graal + Go (ms)
256	489	3179	992	1011
512	235	1426	486	529
1024	123	652	243	266
1536	85	443	162	173
2048	78	371	143	153

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

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

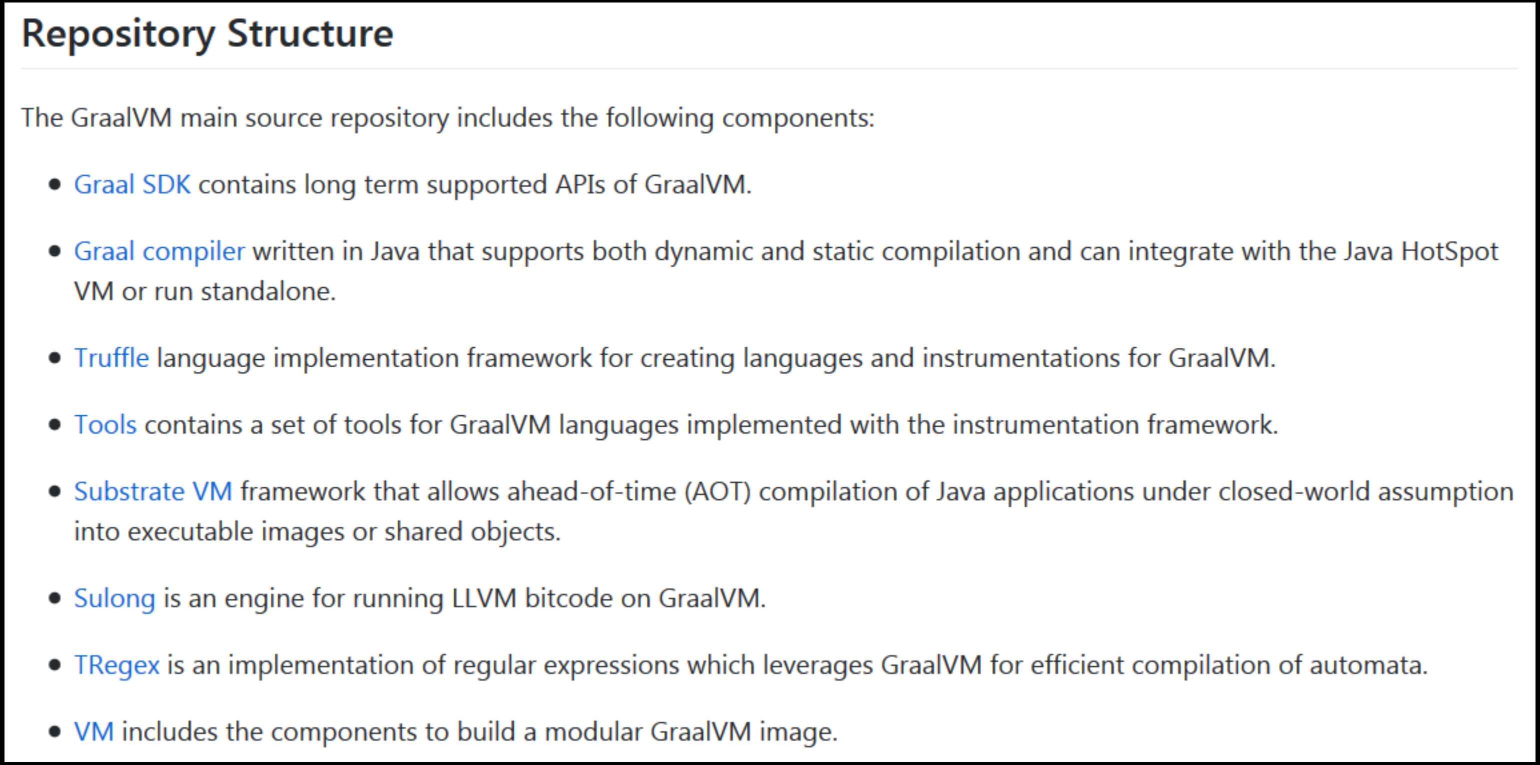


- JavaOne → Oracle Code One (2018)

II. Dive Into GraalVM & GraalPython

1) Truffle & Graal

- https://github.com/neomatrix369/awesome-graal
- http://ssw.jku.at/
- https://github.com/oracle/graal



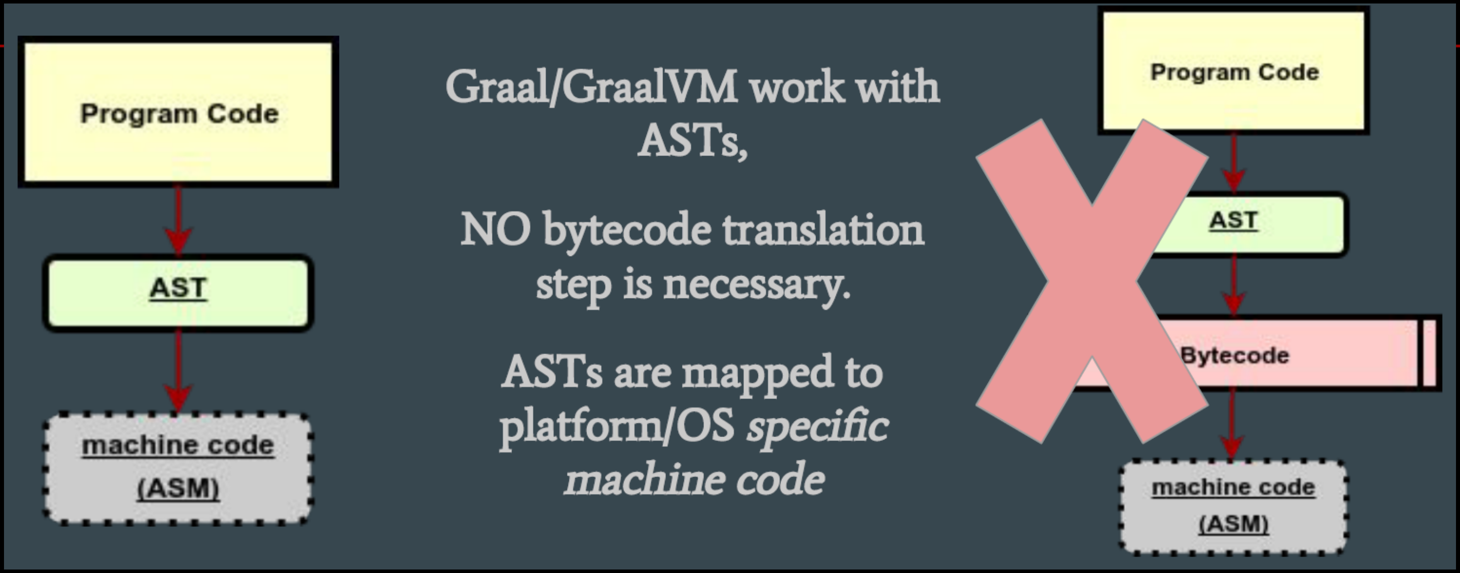
Repository Structure

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.

AST

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



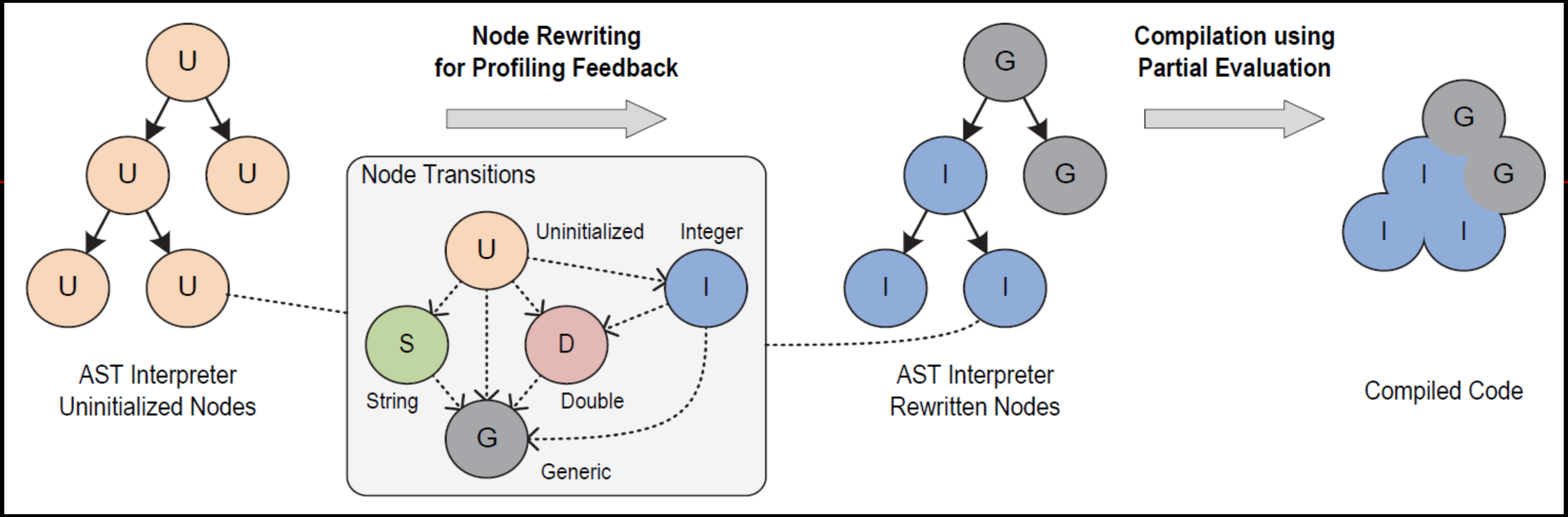
Graal/GraalVM work with ASTs,

NO bytecode translation step is necessary.

ASTs are mapped to platform/OS *specific machine code*

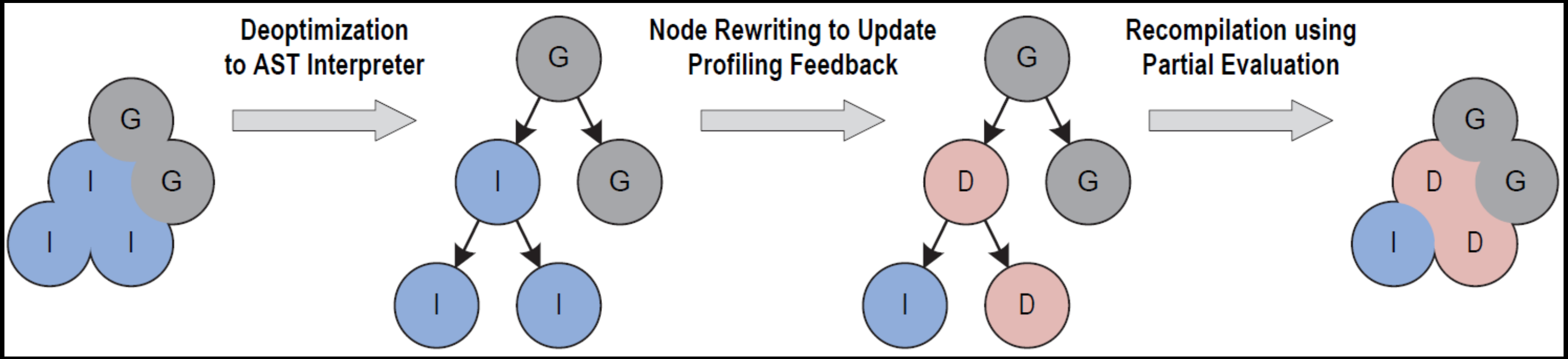
Source: http://crest.cs.ucl.ac.uk/cow/59/slides/cow59_Sarkar.pdf

Optimization and Speculation



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

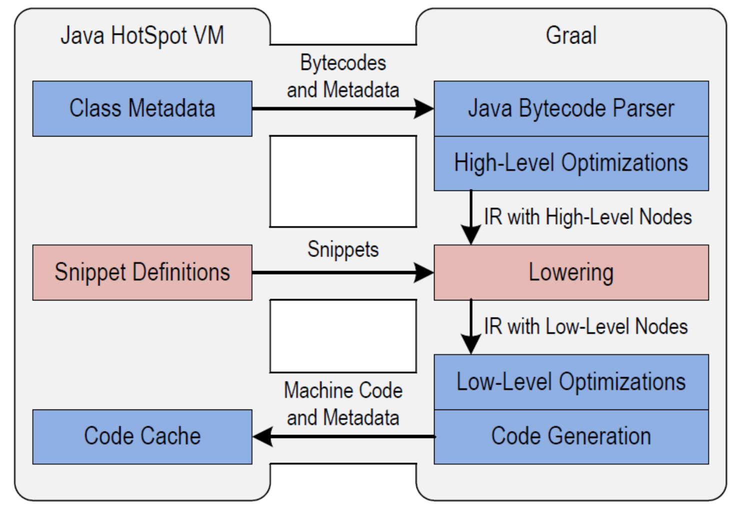
Deoptimization



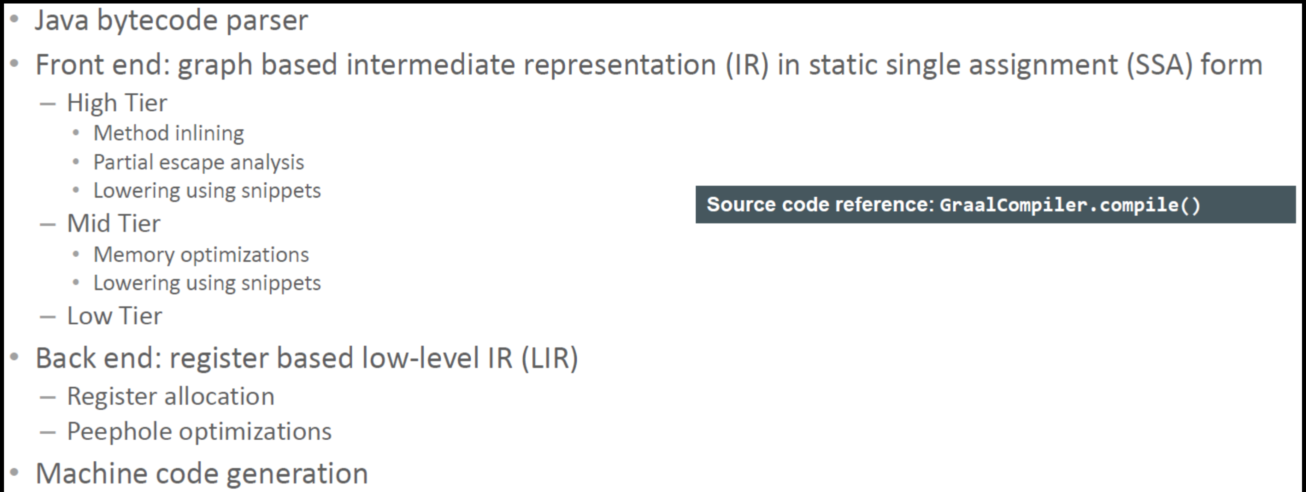
Source: https://qconnewyork.com/system/files/presentation-slides/qconf_final.pdf

Internal of Graal

- http://lafo.ssw.uni-linz.ac.at/papers/2017_PLDI_GraalTutorial.pdf



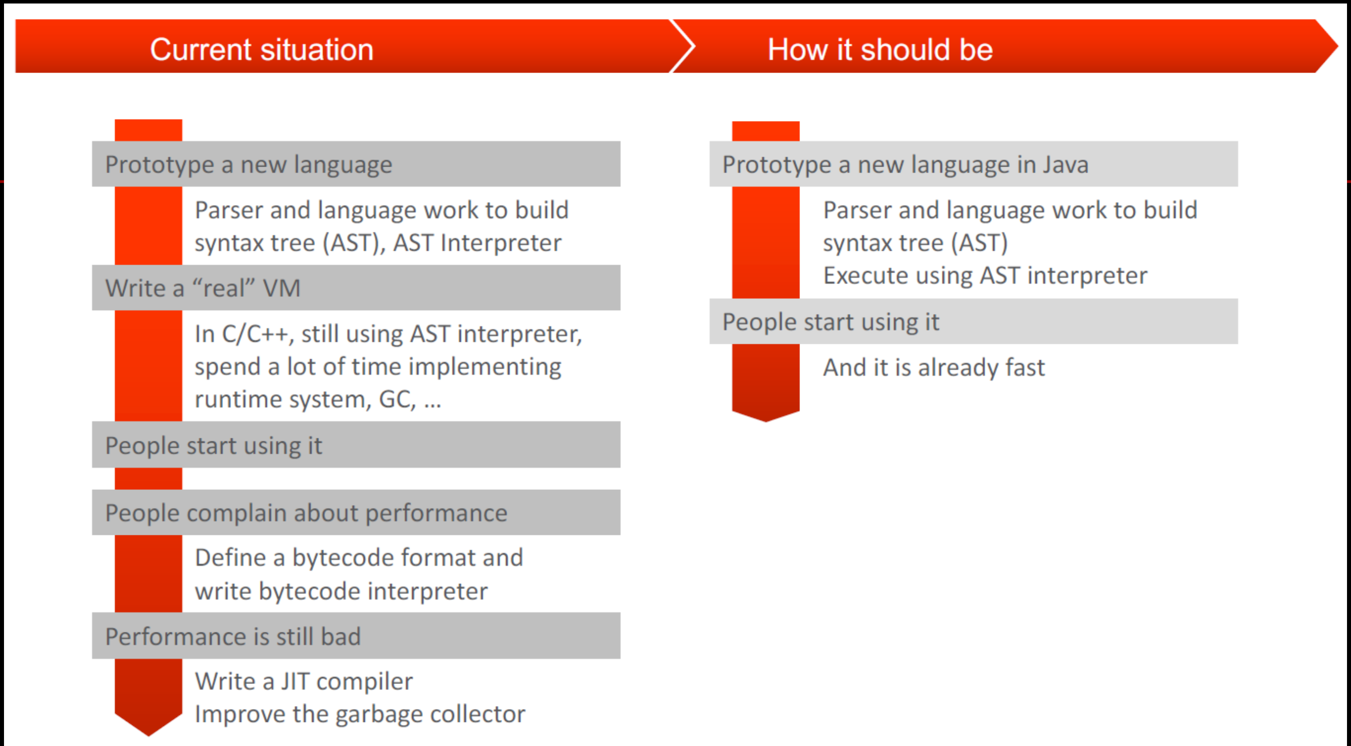
- Default Compilation Pipeline



- Java bytecode parser
- Front end: graph based intermediate representation (IR) in static single assignment (SSA) form
  - High Tier
    - Method inlining
    - Partial escape analysis
    - Lowering using snippets
  - Mid Tier
    - Memory optimizations
    - Lowering using snippets
  - Low Tier
- Back end: register based low-level IR (LIR)
  - Register allocation
  - Peephole optimizations
- Machine code generation

Source code reference: `GraalCompiler.compile()`

Implement a new language runtime



Current situation

Prototype a new language

Parser and language work to build syntax tree (AST), AST Interpreter

Write a “real” VM

In C/C++, still using AST interpreter, spend a lot of time implementing runtime system, GC, …

People start using it

People complain about performance

Define a bytecode format and write bytecode interpreter

Performance is still bad

Write a JIT compiler
Improve the garbage collector

How it should be

Prototype a new language in Java

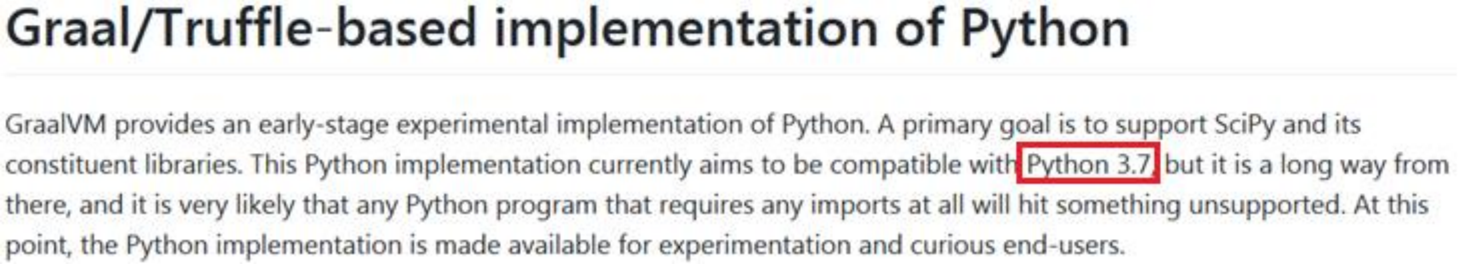
Parser and language work to build syntax tree (AST)
Execute using AST interpreter

People start using it

And it is already fast

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

2) GraalPython



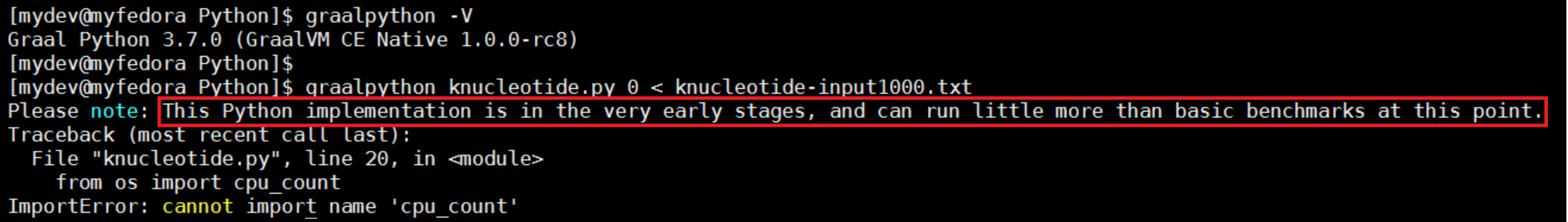
Graal/Truffle-based implementation of Python

GraalVM provides an early-stage experimental implementation of Python. A primary goal is to support SciPy and its constituent libraries. This Python implementation currently aims to be compatible with Python 3.7 but it is a long way from there, and it is very likely that any Python program that requires any imports at all will hit something unsupported. At this point, the Python implementation is made available for experimentation and curious end-users.

- https://github.com/graalvm/graalpython
- https://www.graalvm.org/docs/reference-manual/languages/python/
- https://benchmarksgame-team.pages.debian.net/benchmarksgame/faster/python.html

	Java 11.0.1	CPython 3.6.6	GraalPython ee-1.0.0-rc8
n-body	9.782s	13m8.269s	2m22.776s

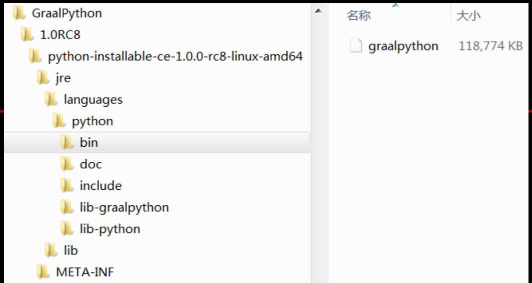
Test on Dell XPS 15z: i5-2410M@2.3Ghz, 6G RAM, Fedora 28 for X64 with Kernel 4.18.16



```
[mydev@myfedora Python]$ graalpython -V
Graal Python 3.7.0 (GraalVM CE Native 1.0.0-rc8)
[mydev@myfedora Python]$
[mydev@myfedora Python]$ graalpython knucleotide.py 0 < knucleotide-input1000.txt
Please note: This Python implementation is in the very early stages, and can run little more than basic benchmarks at this point.
Traceback (most recent call last):
  File "knucleotide.py", line 20, in <module>
    from os import cpu_count
ImportError: cannot import name 'cpu_count'
```

Integration

- https://github.com/graalvm/graalpython/releases/download/vm-1.0.0-rc8/python-installable-ce-1.0.0-rc8-linux-amd64.jar



```
GraalPython
  1.0RC8
    python-installable-ce-1.0.0-rc8-linux-amd64
      jre
        languages
          python
            bin
            doc
            include
            lib-graalpython
            lib-python
        lib
      META-INF
```

名称	大小
graalpython	118,774 KB

- GraalVM EE 1.0.0 RC8

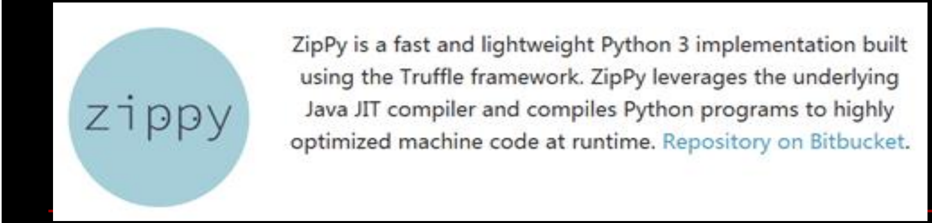
```
graalvm-ee-1.0.0-rc8
├── 3rd_party_licenses_graalpython.txt ->
├── 3rd_party_licenses.txt
├── bin
├── COPYRIGHT
├── db
├── GRAALVM-README.md
├── include
├── javafx-src.zip
├── jre
├── lib
├── LICENSE
├── LICENSE_GRAALPYTHON -> jre/languages/p
├── man
├── README.html
├── release
├── src.zip
├── THIRDPARTYLICENSEREADME-JAVAFX.txt
└── THIRDPARTYLICENSEREADME.txt
```

```
jre
├── bin
├── COPYRIGHT
├── languages
├── lib
├── LICENSE
├── plugin
├── README
├── THIRDPARTYLIC
├── THIRDPARTYLIC
├── tools
└── Welcome.html
```

```
bin
├── clang-sandboxed
├── ControlPanel
├── gemasrv
├── graalpython -> ../languages/python/bin/graalpython
├── gu
├── java
├── javaws
├── jcontrol
├── jjs
├── js
├── keytool
├── lli
├── native-image
├── node
├── npm
├── orbd
├── pack200
├── policytool
├── polyglot
├── rmid
├── rmiregistry
├── servertool
├── tnameserv
└── unpack200
```

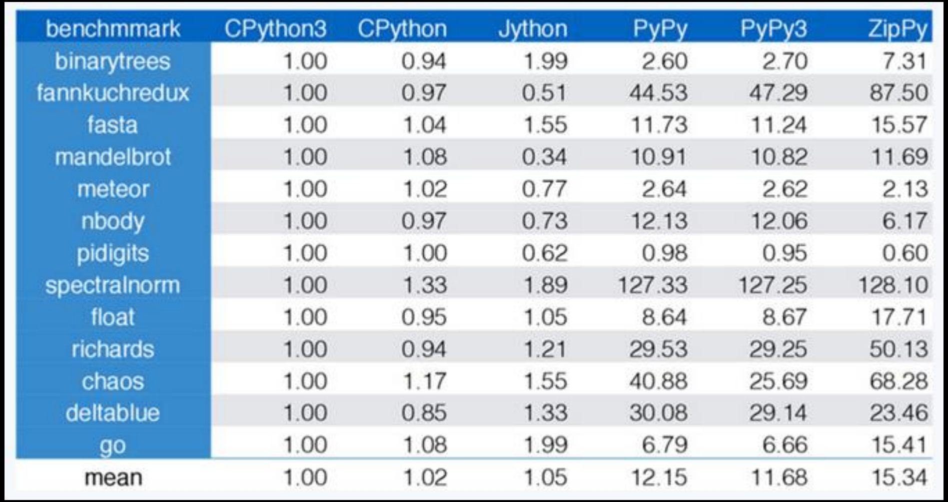
```
languages/
├── js
│   ├── asm-6.2.jar
│   ├── asm-analysis-6.2.jar
│   ├── asm-commons-6.2.jar
│   ├── asm-tree-6.2.jar
│   ├── asm-util-6.2.jar
│   ├── bin
│   ├── graaljs.jar
│   ├── icu4j
│   ├── icu4j.jar
│   ├── include
│   ├── native-image.properties
│   ├── NODE_README.md
│   ├── npm
│   ├── README.md
│   └── trufflenode.jar
├── llvm
│   ├── bin
│   ├── libsulong++.bc
│   ├── libsulong.bc
│   ├── libsulong.so
│   ├── native-image.properties
│   ├── polyglot.h
│   ├── README.md
│   ├── sandboxed
│   ├── sulong.jar
│   └── sulong-managed.jar
└── python
    ├── 3rd_party_licenses_graalpython.txt
    ├── bin
    ├── doc
    ├── graalpython.jar
    ├── include
    ├── lib-graalpython
    ├── lib-python
    ├── LICENSE_GRAALPYTHON
    ├── native-image.properties
    ├── README_GRAALPYTHON.md
    └── release
```

ZipPy

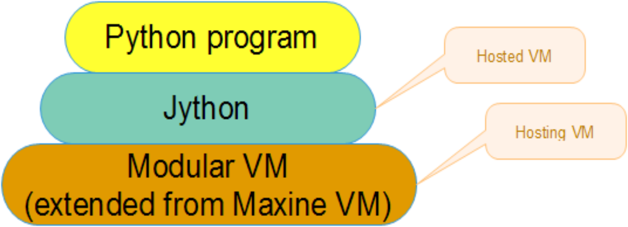


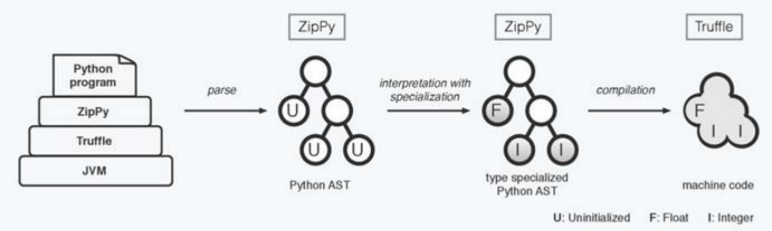
ZipPy is a fast and lightweight Python 3 implementation built using the Truffle framework. ZipPy leverages the underlying Java JIT compiler and compiles Python programs to highly optimized machine code at runtime. Repository on Bitbucket.

- http://thezhangwei.com/
- https://github.com/securesystemslab/zippy
- Optimizations
  - – Numeric Types, Type Specializations, Efficient Data Representation
  - – Control Flow Specializations, Generator Peeling, Optimizing Object Model and Calls



benchmark	CPython3	CPython	Jython	PyPy	PyPy3	ZipPy
binarytrees	1.00	0.94	1.99	2.60	2.70	7.31
fannkuchredux	1.00	0.97	0.51	44.53	47.29	87.50
fasta	1.00	1.04	1.55	11.73	11.24	15.57
mandelbrot	1.00	1.08	0.34	10.91	10.82	11.69
meteor	1.00	1.02	0.77	2.64	2.62	2.13
nbody	1.00	0.97	0.73	12.13	12.06	6.17
pidigits	1.00	1.00	0.62	0.98	0.95	0.60
spectralnorm	1.00	1.33	1.89	127.33	127.25	128.10
float	1.00	0.95	1.05	8.64	8.67	17.71
richards	1.00	0.94	1.21	29.53	29.25	50.13
chaos	1.00	1.17	1.55	40.88	25.69	68.28
deltablue	1.00	0.85	1.33	30.08	29.14	23.46
go	1.00	1.08	1.99	6.79	6.66	15.41
mean	1.00	1.02	1.05	12.15	11.68	15.34



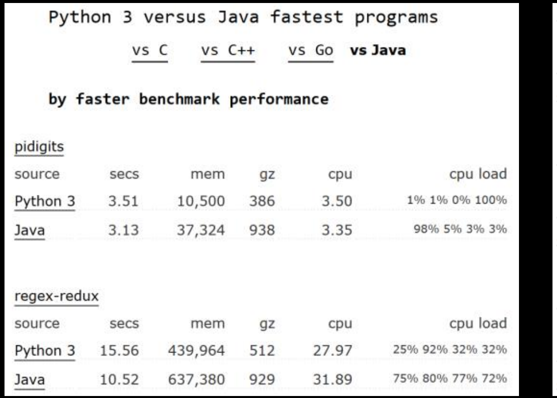


III. Rethinking Python Runtime

1) Accelerating Python

Why Python is Slow

- ~~dynamically typed~~
- no JIT support in the official CPython
- GIL (Global Interpreter Lock)
-



Python 3 versus Java fastest programs

vs C vs C++ vs Go **vs Java**

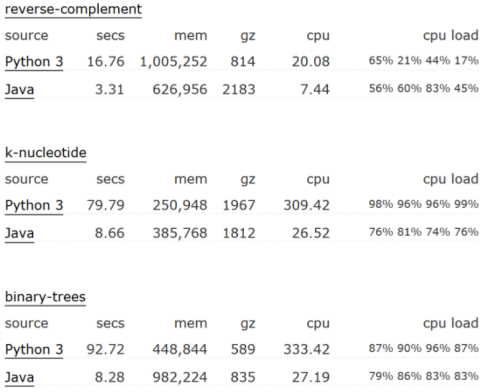
by faster benchmark performance

pidigits

source	secs	mem	gz	cpu	cpu load
Python 3	3.51	10,500	386	3.50	1% 1% 0% 100%
Java	3.13	37,324	938	3.35	98% 5% 3% 3%

regex-redux

source	secs	mem	gz	cpu	cpu load
Python 3	15.56	439,964	512	27.97	25% 92% 32% 32%
Java	10.52	637,380	929	31.89	75% 80% 77% 72%



reverse-complement

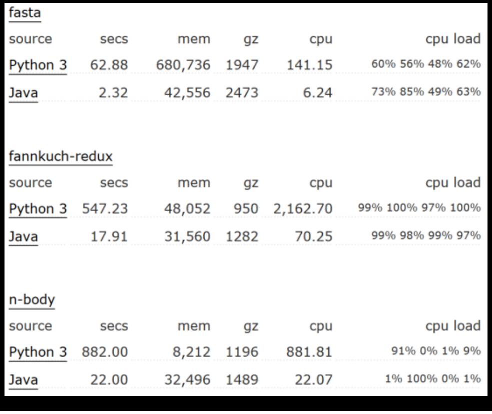
source	secs	mem	gz	cpu	cpu load
Python 3	16.76	1,005,252	814	20.08	65% 21% 44% 17%
Java	3.31	626,956	2183	7.44	56% 60% 83% 45%

k-nucleotide

source	secs	mem	gz	cpu	cpu load
Python 3	79.79	250,948	1967	309.42	98% 96% 96% 99%
Java	8.66	385,768	1812	26.52	76% 81% 74% 76%

binary-trees

source	secs	mem	gz	cpu	cpu load
Python 3	92.72	448,844	589	333.42	87% 90% 96% 87%
Java	8.28	982,224	835	27.19	79% 86% 83% 83%



fasta

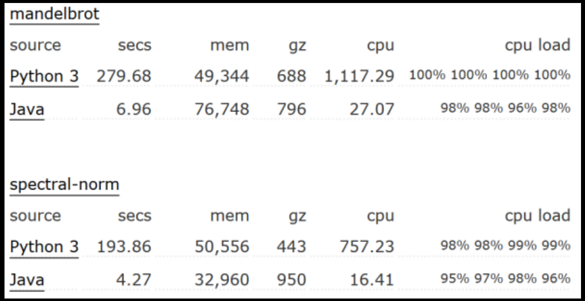
source	secs	mem	gz	cpu	cpu load
Python 3	62.88	680,736	1947	141.15	60% 56% 48% 62%
Java	2.32	42,556	2473	6.24	73% 85% 49% 63%

fannkuch-redux

source	secs	mem	gz	cpu	cpu load
Python 3	547.23	48,052	950	2,162.70	99% 100% 97% 100%
Java	17.91	31,560	1282	70.25	99% 98% 99% 97%

n-body

source	secs	mem	gz	cpu	cpu load
Python 3	882.00	8,212	1196	881.81	91% 0% 1% 9%
Java	22.00	32,496	1489	22.07	1% 100% 0% 1%

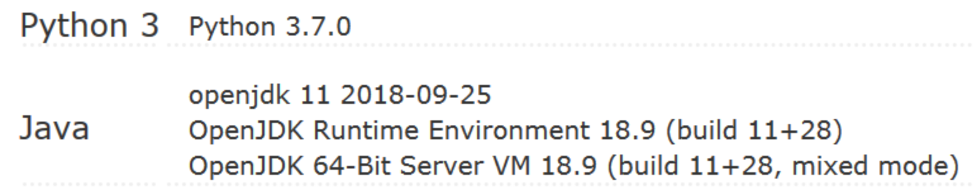


mandelbrot

source	secs	mem	gz	cpu	cpu load
Python 3	279.68	49,344	688	1,117.29	100% 100% 100% 100%
Java	6.96	76,748	796	27.07	98% 98% 96% 98%

spectral-norm

source	secs	mem	gz	cpu	cpu load
Python 3	193.86	50,556	443	757.23	98% 98% 99% 99%
Java	4.27	32,960	950	16.41	95% 97% 98% 96%

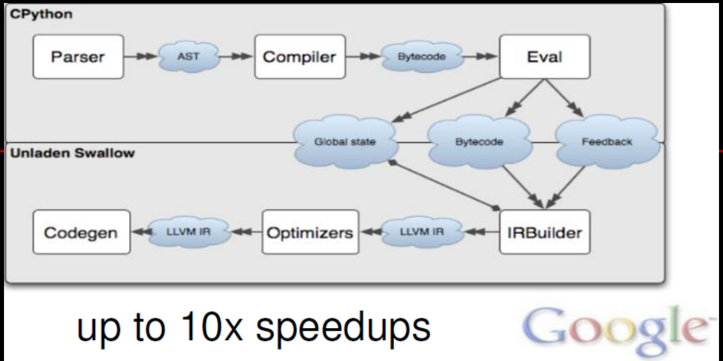


Python 3	Python 3.7.0
Java	openjdk 11 2018-09-25 OpenJDK Runtime Environment 18.9 (build 11+28) OpenJDK 64-Bit Server VM 18.9 (build 11+28, mixed mode)

https://benchmarksgame-team.pages.debian.net/benchmarksgame/faster/python.html

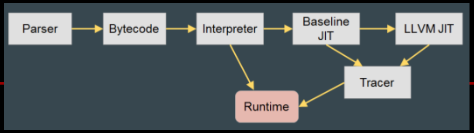
Runtimes

- LLVM-based (VMKit, MCJIT…)



up to 10x speedups

PySton



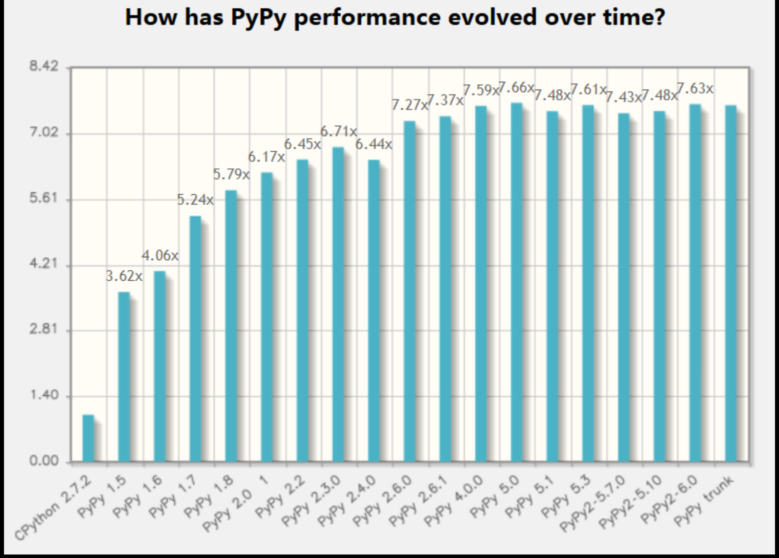
Dropbox

- pypy

RPython

Meta-tracing

…

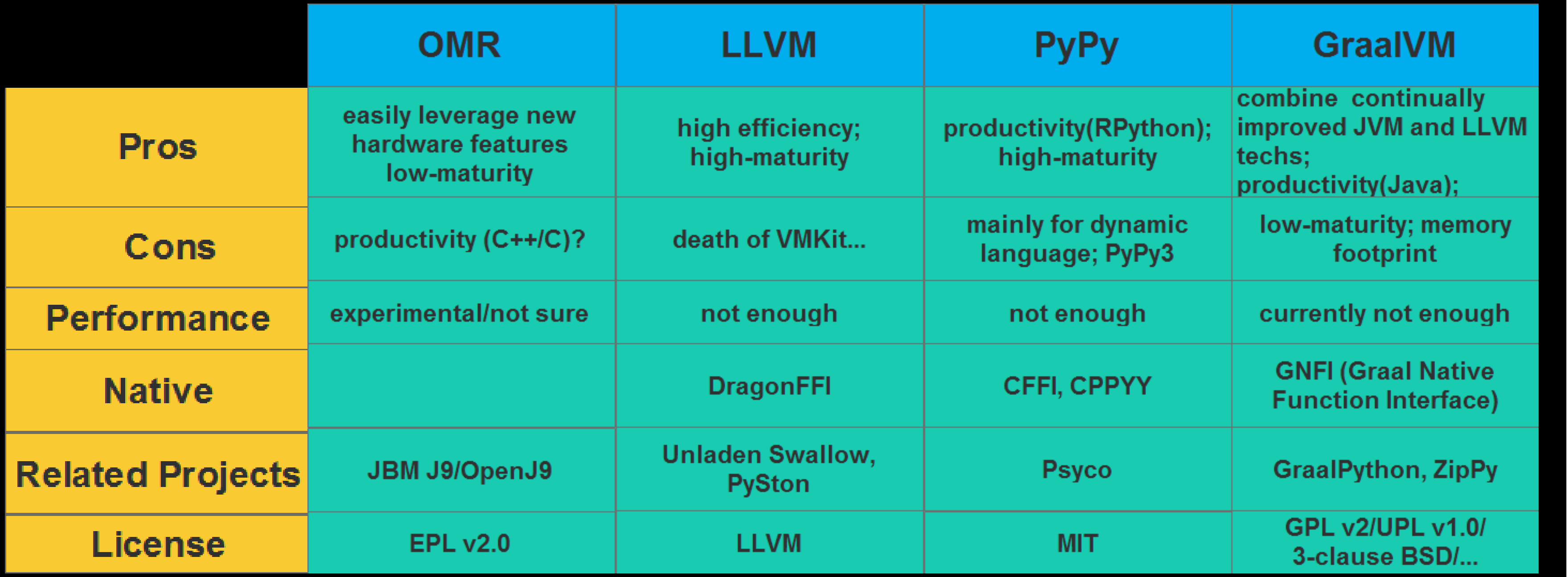


Source: http://speed.pypy.org/

2) Redesign

Rethinking of Python Runtime

- from my point of view, various Runtime Frameworks for Python implementation:



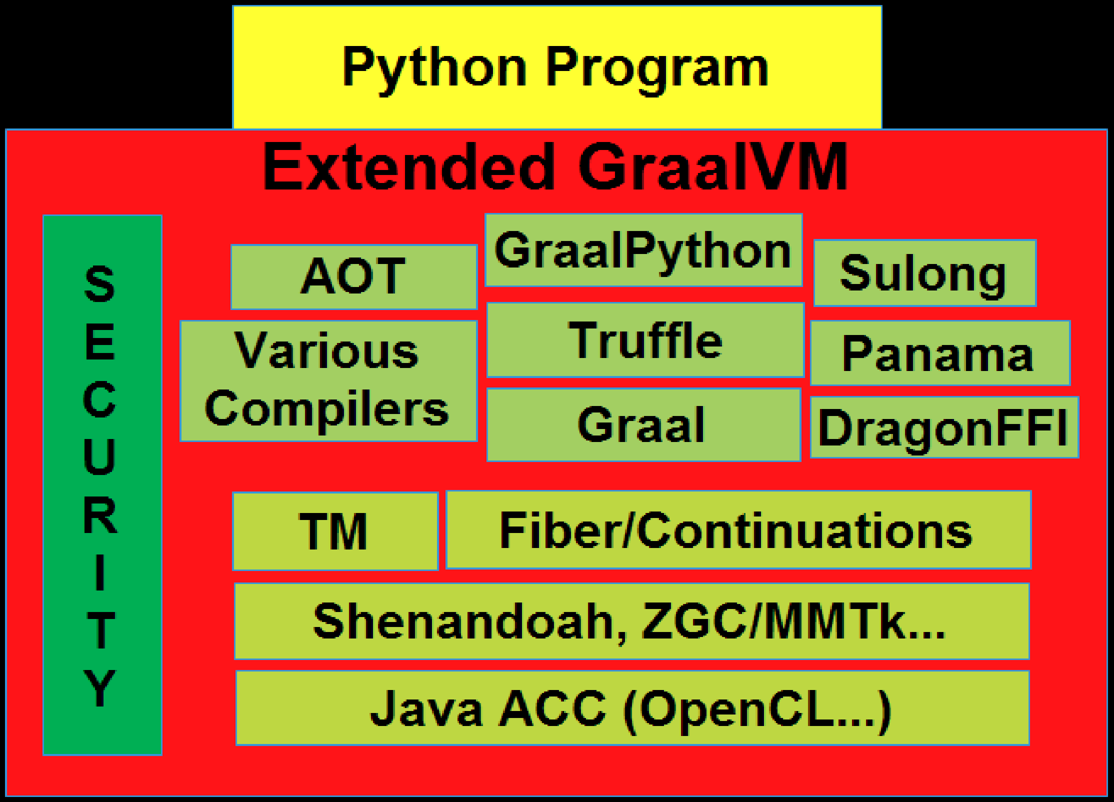
	OMR	LLVM	PyPy	GraalVM

Pros	easily leverage new hardware features
	low-maturity
		high efficiency;
		high-maturity
			productivity(RPython);
			high-maturity
				combine continually improved JVM and LLVM techs;
				productivity(Java);

Cons	productivity (C++/C)?	death of VMKit...	mainly for dynamic language; PyPy3	low-maturity; memory footprint
Performance	experimental/not sure	not enough	not enough	currently not enough
Native		DragonFFI	CFFI, CPPYY	GNFI (Graal Native Function Interface)
Related Projects	JBM J9/OpenJ9	Unladen Swallow, PySton	Psyco	GraalPython, ZipPy
License	EPL v2.0	LLVM	MIT	GPL v2/UPL v1.0/ 3-clause BSD/...

New Design

- http://openjdk.java.net/projects/metropolis/
- Base on JDK 12 — built-in support for ARM
- My redesign — extended GraalVM



Python Program

Extended GraalVM

SECURITY

AOT

GraalPython

Sulong

Various Compilers

Truffle

Panama

Graal

DragonFFI

TM

Fiber/Continuations

Shenandoah, ZGC/MMTk...

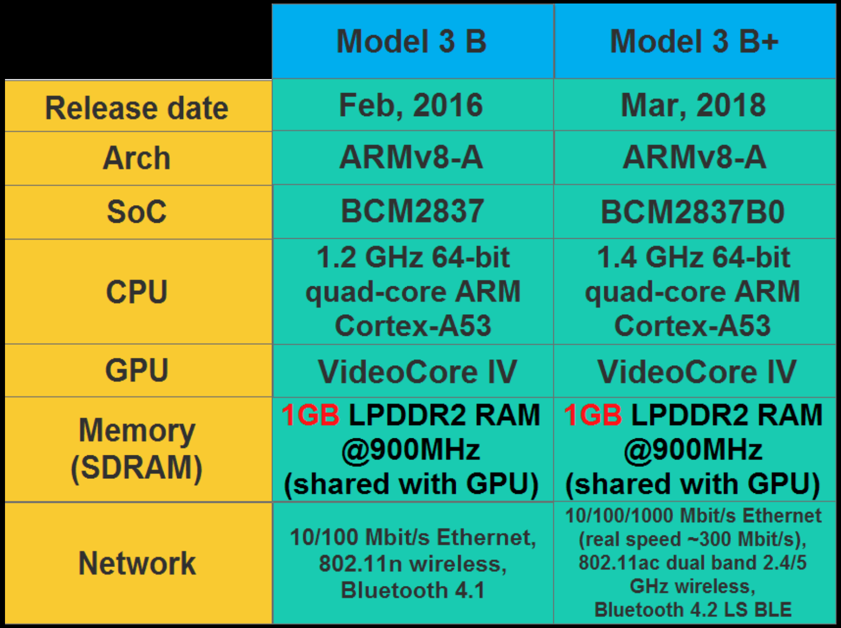
Java ACC (OpenCL...)

IV. GraalPython on ARM

1) Development Env

Raspberry Pi

- https://www.raspberrypi.org/



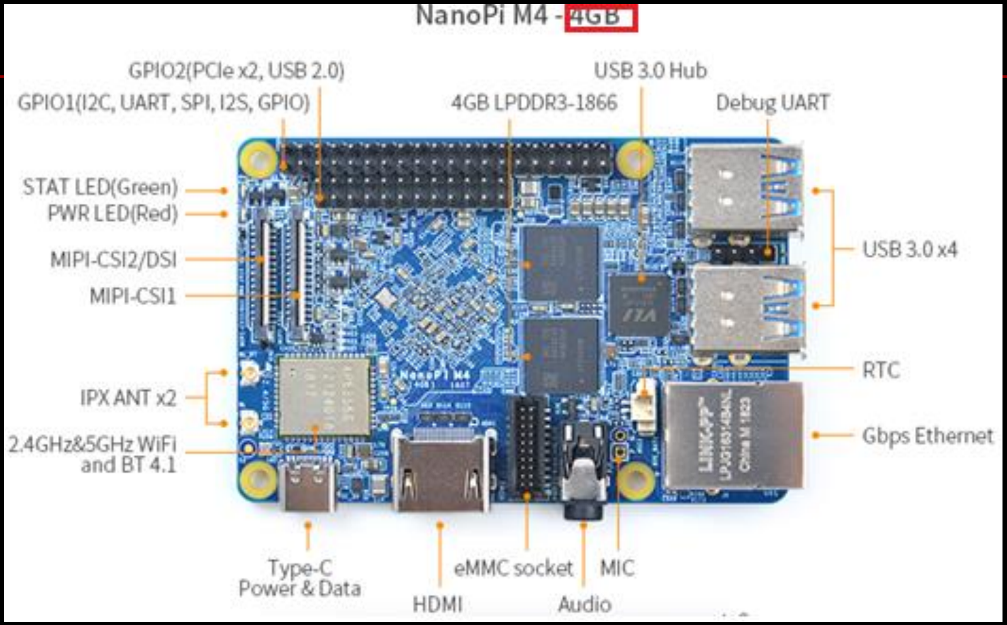
	Model 3 B	Model 3 B+
Release date	Feb, 2016	Mar, 2018
Arch	ARMv8-A	ARMv8-A
SoC	BCM2837	BCM2837B0
CPU	1.2 GHz 64-bit quad-core ARM Cortex-A53	1.4 GHz 64-bit quad-core ARM Cortex-A53
GPU	VideoCore IV	VideoCore IV
Memory (SDRAM)	1GB LPDDR2 RAM @900MHz (shared with GPU)	1GB LPDDR2 RAM @900MHz (shared with GPU)
Network	10/100 Mbit/s Ethernet, 802.11n wireless, Bluetooth 4.1	10/100/1000 Mbit/s Ethernet (real speed ~300 Mbit/s), 802.11ac dual band 2.4/5 GHz wireless, Bluetooth 4.2 LS BLE

- Official release (Raspbian with Linux Kernel 4.14 currently) still does not support AArch64
- Pls refer to my presentation "eBPF in Action" at LC3 Beijing (on Jun 25, 2018)

NanoPi M4

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

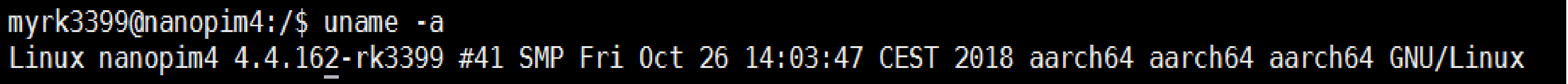
Rockchip RK3399



- 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)

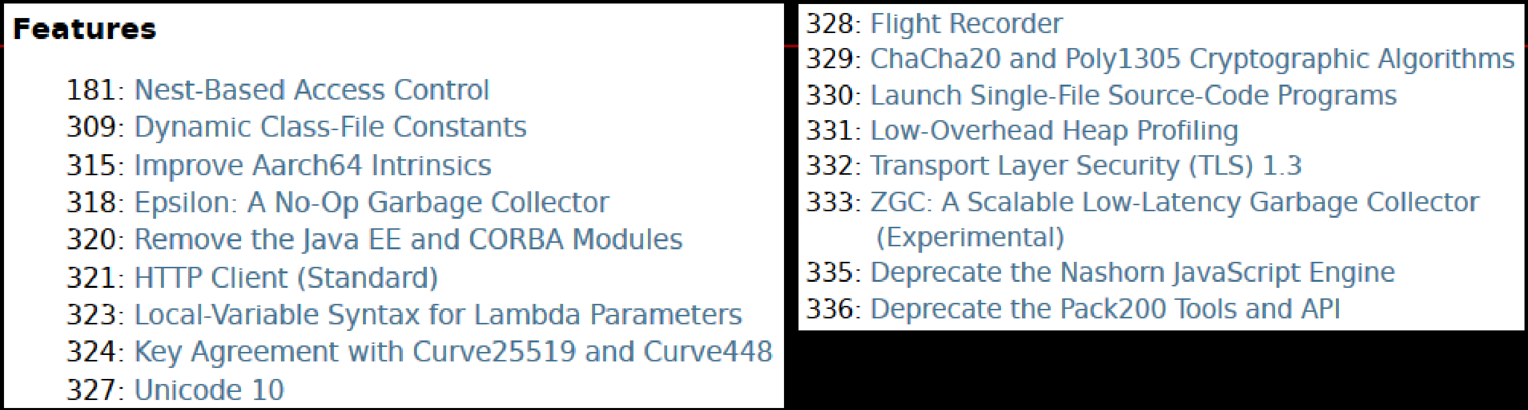


```
myrk3399@nanopim4:/$ uname -a
Linux nanopim4 4.4.162-rk3399 #41 SMP Fri Oct 26 14:03:47 CEST 2018 aarch64 aarch64 aarch64 GNU/Linux
```

2) My Practice

Technical Solution

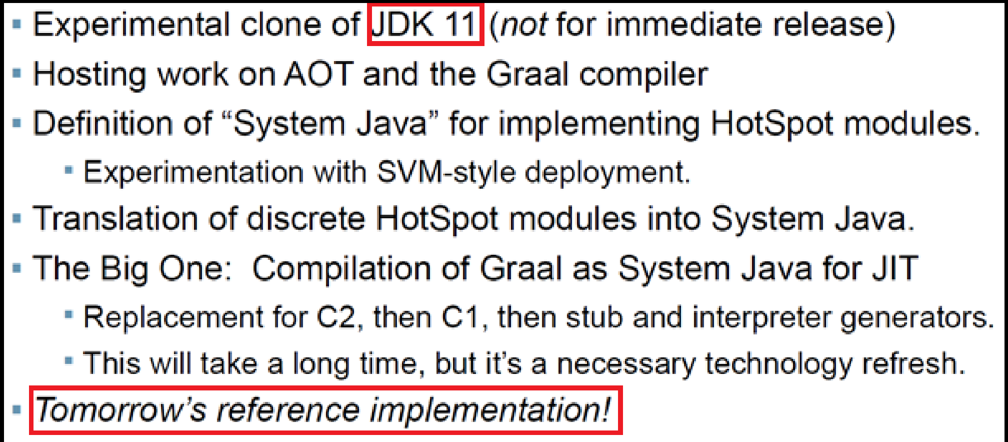
- http://openjdk.java.net/projects/jdk/11/



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/



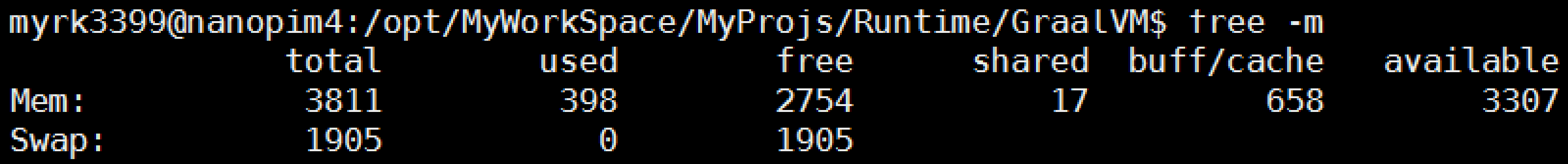
- Experimental clone of JDK 11 (*not* for immediate release)
- Hosting work on AOT and the Graal compiler
- Definition of “System Java” for implementing HotSpot modules.
  - Experimentation with SVM-style deployment.
- Translation of discrete HotSpot modules into System Java.
- The Big One: Compilation 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

OpenJDK 11 on ARM

- http://hg.openjdk.java.net/jdk-updates/jdk11u/
- #build OpenJDK 11 by yourself
- https://github.com/AdoptOpenJDK/openjdk11-binaries/releases
- #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
```

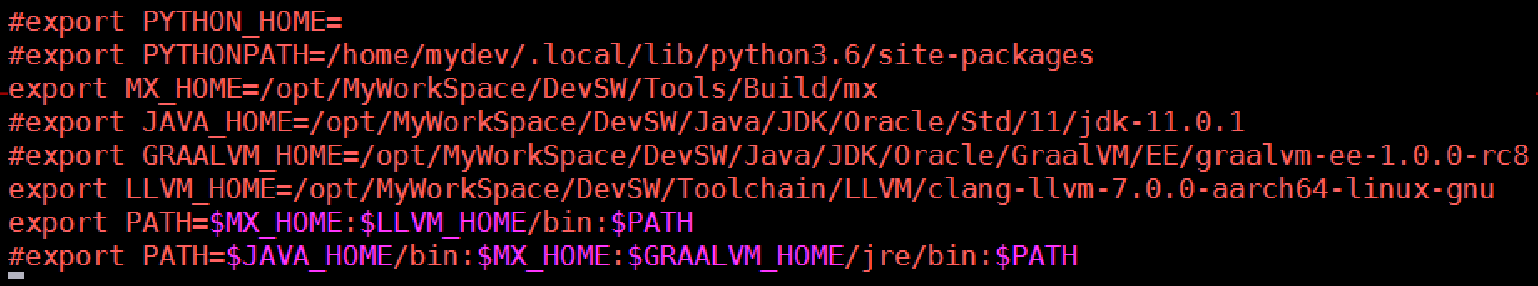
- cd $YOUR_OPENJDK11_SRCHOME and run the commands:

  bash configure --disable-warnings-as-errors

  make JOBS=6 images

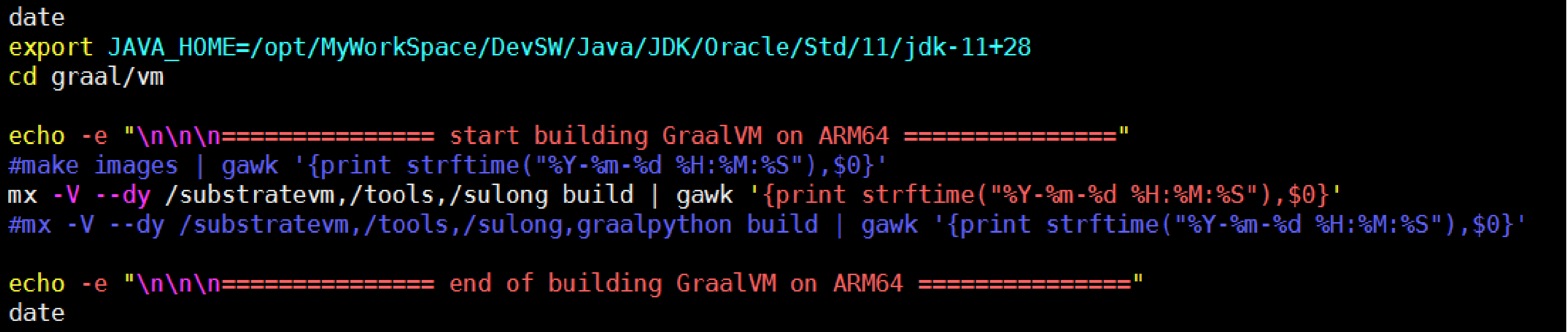
build GraalVM & GraalPython

- 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-rc8
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

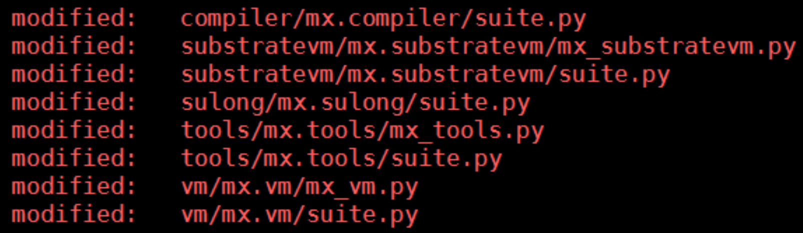


```
date
export JAVA_HOME=/opt/MyWorkSpace/DevSW/Java/JDK/Oracle/Std/11/jdk-11+28
cd graal/vm

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

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

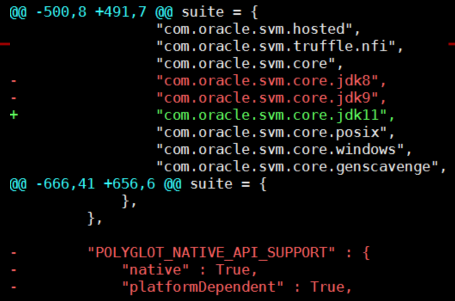
- patching for avoid various limitation

Java compliance, add support for JDK 11
Polyglot Native API, VisualVM (currently do not support AARCH64)
various build errors for sulong

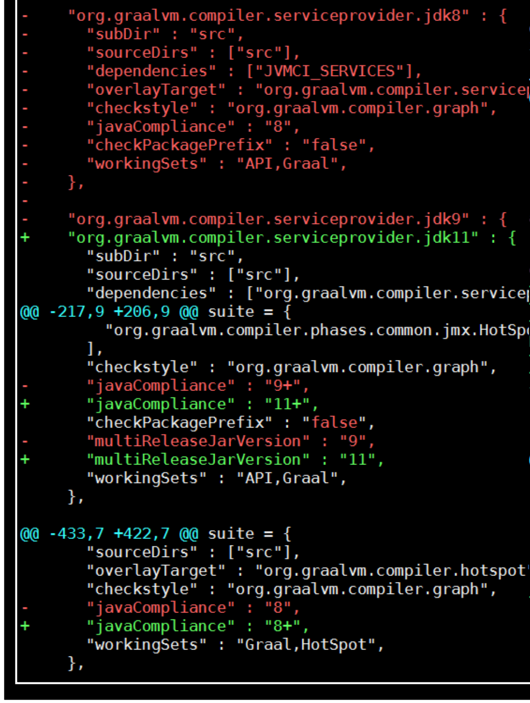
...



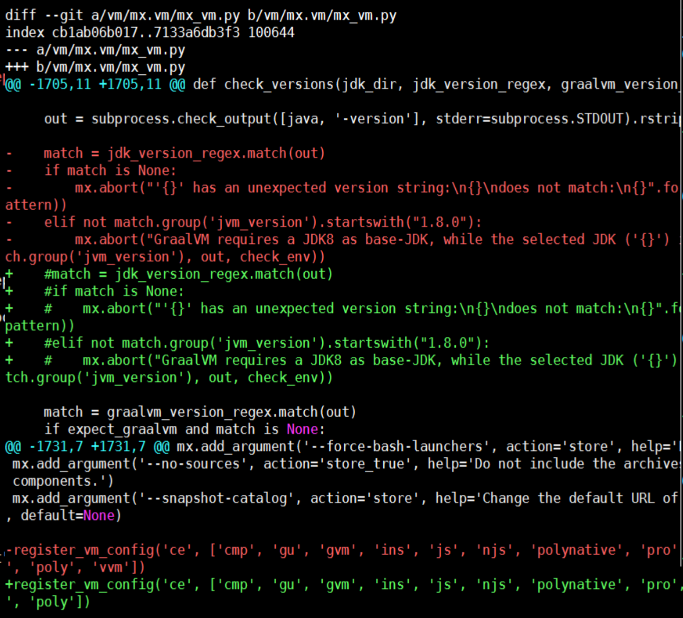
```
modified:   compiler/mx.compiler/suite.py
modified:   substratevm/mx.substratevm/mx_substratevm.py
modified:   substratevm/mx.substratevm/suite.py
modified:   sulong/mx.sulong/suite.py
modified:   tools/mx.tools/mx_tools.py
modified:   tools/mx.tools/suite.py
modified:   vm/mx.vm/mx_vm.py
modified: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



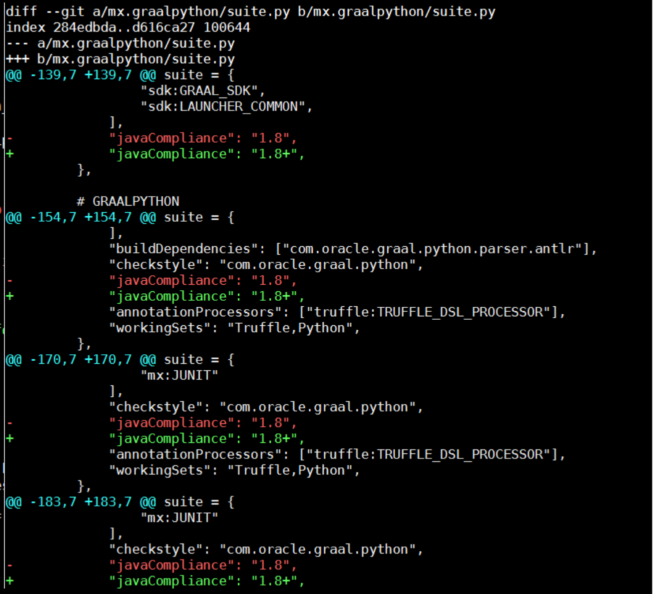
```
@@ -500,8 +491,7 @@ suite = {
                   "com.oracle.svm.hosted",
                   "com.oracle.svm.truffle.nfi",
                   "com.oracle.svm.core",
-                  "com.oracle.svm.core.jdk8",
-                  "com.oracle.svm.core.jdk9",
+                  "com.oracle.svm.core.jdk11",
                   "com.oracle.svm.core.posix",
                   "com.oracle.svm.core.windows",
                   "com.oracle.svm.core.genscavenge",
@@ -666,41 +656,6 @@ suite = {
                 },
             },

-           "POLYGLOT_NATIVE_API_SUPPORT" : {
-               "native" : True,
-               "platformDependent" : True,
```



```
-    "org.graalvm.compiler.serviceprovider.jdk8" : {
-      "subDir" : "src",
-      "sourceDirs" : ["src"],
-      "dependencies" : ["JVMCI_SERVICES"],
-      "overlayTarget" : "org.graalvm.compiler.service
-      "checkstyle" : "org.graalvm.compiler.graph",
-      "javaCompliance" : "8",
-      "checkPackagePrefix" : "false",
-      "workingSets" : "API,Graal",
-    },
-
-    "org.graalvm.compiler.serviceprovider.jdk9" : {
+    "org.graalvm.compiler.serviceprovider.jdk11" : {
       "subDir" : "src",
       "sourceDirs" : ["src"],
       "dependencies" : ["org.graalvm.compiler.service
@@ -217,9 +206,9 @@ suite = {
         "org.graalvm.compiler.phases.common.jmx.HotSp
       ],
       "checkstyle" : "org.graalvm.compiler.graph",
-      "javaCompliance" : "9+",
+      "javaCompliance" : "11+",
       "checkPackagePrefix" : "false",
-      "multiReleaseJarVersion" : "9",
+      "multiReleaseJarVersion" : "11",
       "workingSets" : "API,Graal",
     },

@@ -433,7 +422,7 @@ suite = {
       "sourceDirs" : ["src"],
       "overlayTarget" : "org.graalvm.compiler.hotspot
       "checkstyle" : "org.graalvm.compiler.graph",
-      "javaCompliance" : "8",
+      "javaCompliance" : "8+",
       "workingSets" : "Graal,HotSpot",
     },
```

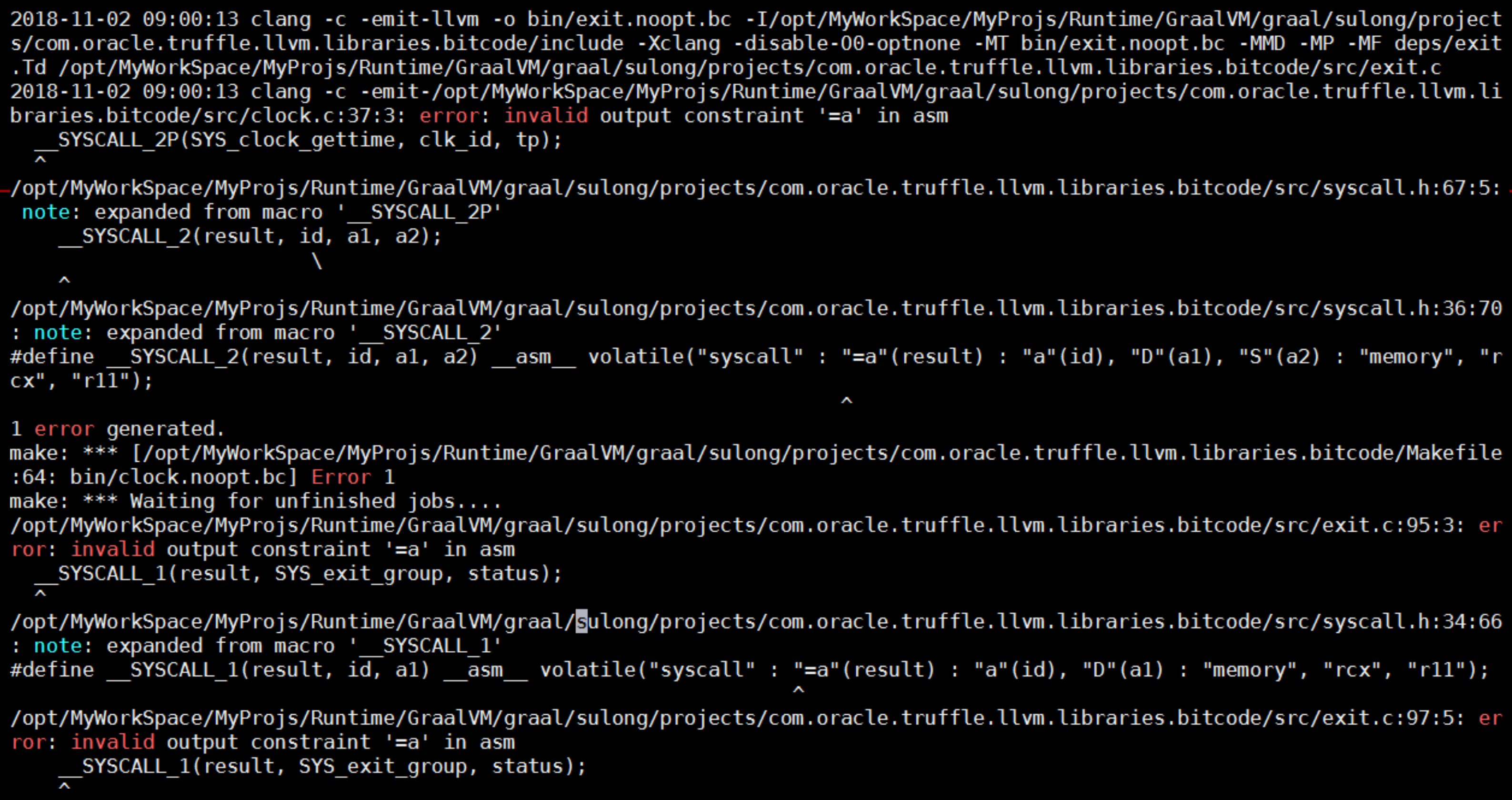


```
diff --git a/vm/mx.vm/mx_vm.py b/vm/mx.vm/mx_vm.py
index cb1ab06b017..7133a6db3f3 100644
--- a/vm/mx.vm/mx_vm.py
+++ b/vm/mx.vm/mx_vm.py
@@ -1705,11 +1705,11 @@ def check_versions(jdk_dir, jdk_version_regex, graalvm_version

     out = subprocess.check_output([java, '-version'], stderr=subprocess.STDOUT).rstri

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

     match = graalvm_version_regex.match(out)
     if expect_graalvm and match is None:
@@ -1731,7 +1731,7 @@ mx.add_argument('--force-bash-launchers', action='store', help='
 mx.add_argument('--no-sources', action='store_true', help='Do not include the archive
 components.')
 mx.add_argument('--snapshot-catalog', action='store', help='Change the default URL of
, default=None)

-register_vm_config('ce', ['cmp', 'gu', 'gvm', 'ins', 'js', 'njs', 'polynative', 'pro'
', 'poly', 'vvm'])
+register_vm_config('ce', ['cmp', 'gu', 'gvm', 'ins', 'js', 'njs', 'polynative', 'pro', 'rgx', 'slg', 'svm', 'tfl', 'libpoly
', 'poly'])
```



```
diff --git a/mx.graalpython/suite.py b/mx.graalpython/suite.py
index 284edbda..d616ca27 100644
--- a/mx.graalpython/suite.py
+++ b/mx.graalpython/suite.py
@@ -139,7 +139,7 @@ suite = {
                 "sdk:GRAAL_SDK",
                 "sdk:LAUNCHER_COMMON",
             ],
-            "javaCompliance": "1.8",
+            "javaCompliance": "1.8+",
         },

         # GRAALPYTHON
@@ -154,7 +154,7 @@ suite = {
             ],
             "buildDependencies": ["com.oracle.graal.python.parser.antlr"],
             "checkstyle": "com.oracle.graal.python",
-            "javaCompliance": "1.8",
+            "javaCompliance": "1.8+",
             "annotationProcessors": ["truffle:TRUFFLE_DSL_PROCESSOR"],
             "workingSets": "Truffle,Python",
         },
@@ -170,7 +170,7 @@ suite = {
                 "mx:JUNIT"
             ],
             "checkstyle": "com.oracle.graal.python",
-            "javaCompliance": "1.8",
+            "javaCompliance": "1.8+",
             "annotationProcessors": ["truffle:TRUFFLE_DSL_PROCESSOR"],
             "workingSets": "Truffle,Python",
         },
@@ -183,7 +183,7 @@ suite = {
                 "mx:JUNIT"
             ],
             "checkstyle": "com.oracle.graal.python",
-            "javaCompliance": "1.8",
+            "javaCompliance": "1.8+",
```

failed to build GraalVM with OpenJDK 11



```
2018-11-02 09:00:13 clang -c -emit-llvm -o bin/exit.noopt.bc -I/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/graal/sulong/project
s/com.oracle.truffle.llvm.libraries.bitcode/include -Xclang -disable-O0-optnone -MT bin/exit.noopt.bc -MMD -MP -MF deps/exit
.Td /opt/MyWorkSpace/MyProjs/Runtime/GraalVM/graal/sulong/projects/com.oracle.truffle.llvm.libraries.bitcode/src/exit.c
2018-11-02 09:00:13 clang -c -emit-/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/graal/sulong/projects/com.oracle.truffle.llvm.li
braries.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: expanded 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: expanded from macro '__SYSCALL_2'
#define __SYSCALL_2(result, id, a1, a2) __asm__ volatile("syscall" : "=a"(result) : "a"(id), "D"(a1), "S"(a2) : "memory", "r
cx", "r11");
                                                                     ^
1 error generated.
make: *** [/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/graal/sulong/projects/com.oracle.truffle.llvm.libraries.bitcode/Makefile
:64: bin/clock.noopt.bc] Error 1
make: *** Waiting for unfinished jobs....
/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/graal/sulong/projects/com.oracle.truffle.llvm.libraries.bitcode/src/exit.c:95:3: er
ror: invalid 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: expanded 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: er
ror: invalid output constraint '=a' in asm
    __SYSCALL_1(result, SYS_exit_group, status);
    ^
```

Sulong does not support AARCH64…

challenges

- 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+

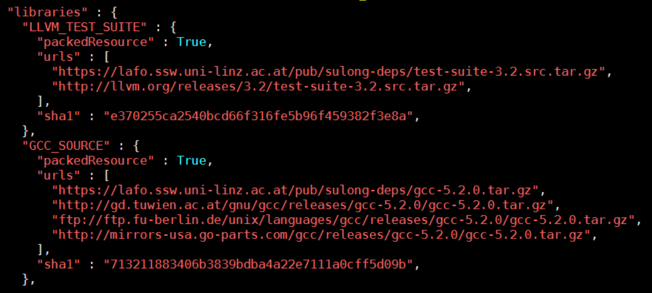
gra~~al-j~~vmci-8

Fork of jdk8u/hotspot with support for JVMCI

● C++ ★ 13 4 Updated a day ago

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",
  },
```

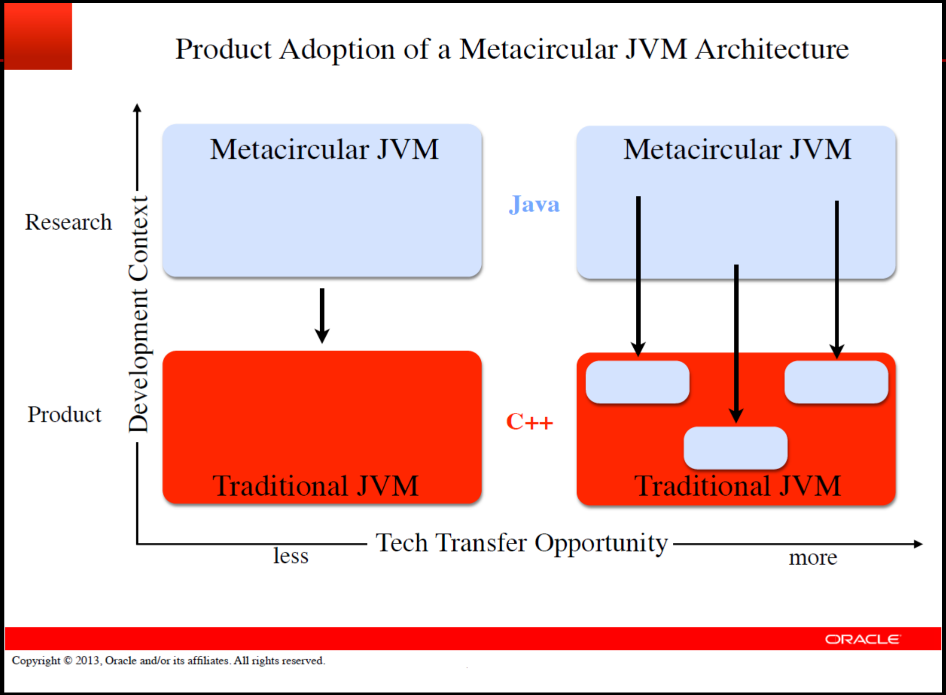
- deprecated APIs

https://docs.oracle.com/javase/10/docs/api/java.xml.bind-summary.html

- customize GraalPython to meet our need
- build native code of GraalVM & GraalPython by LLVM/Clang
- deal with Java, JDK, Truffle/Graal, LLVM, Python…
- make OpenJDK & GraalVM more developer/hacker friendly

V. Wrap-up

- Meta-circular VM



Product Adoption of a Metacircular JVM Architecture

- Is GraalVM/GraalPython the best choice for Python Runtime? Rethinking App Runtime…

Q & A

Thanks!

Reference

Slides/materials from many and varied sources:

- http://en.wikipedia.org/wiki/
- http://www.slideshare.net/
- https://www.python.org
- http://llvm.org
- https://en.wikipedia.org/wiki/Just-in-time_compilation
- https://github.com/dropbox/pyston
- …