GOTC

全球开源技术峰会

THE GLOBAL OPENSOURCE TECHNOLOGY CONFERENCE

OPEN SOURCE , OPEN WORLD

「开源云原生计算时代」专场

本期议题：**GraalVM-based unified runtime for eBPF & WASM**

李枫 2021年8月1日

Agenda

GOTC

I. Overview

- Runtime
- DSL
- eBPF
- LLVM
- WASM
- GraalVM
- Testbed

II. eBPF on GraalVM

- uBPF
- Implementations

III. WASM on GraalVM

- GraalVM-native Implementation
- Sulong-based Implementation

GOTC

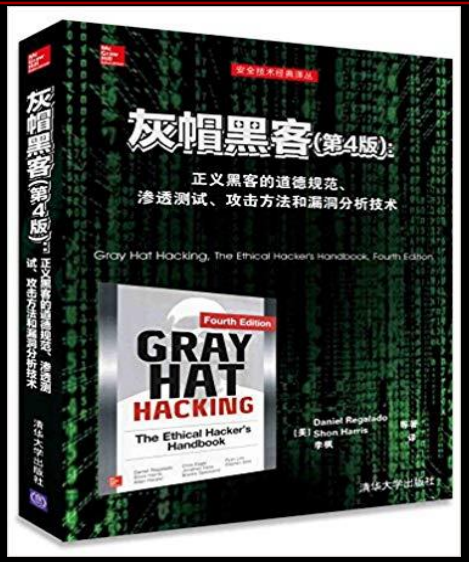
IV. Unified eBPF & WASM runtime for Cloud Native

- Rethinking Cloud Native
- Why is GraalVM
- Ideas for Unified eBPF & WASM runtime

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» (ISBN: 9787115544384)



- Pure software developmement for ~15 years
- Actively participate in various activities of the open source community
- https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/Conf
  https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/LTS
- Recently, focus on infrastructure of Cloud/Edge Computing, AI, Virtualization, Program Runtimes, Network, 5G, RISC-V, EDA…

I. Overview

1) Runtime

- https://en.wikipedia.org/wiki/Runtime_system
- https://en.wikipedia.org/wiki/Register_machine
- https://en.wikipedia.org/wiki/Stack_machine
- https://en.wikipedia.org/wiki/Intermediate_representation
- https://en.wikipedia.org/wiki/Bytecode
- https://en.wikipedia.org/wiki/Compiler
- https://en.wikipedia.org/wiki/Just-in-time_compilation
- https://en.wikipedia.org/wiki/Ahead-of-time_compilation
- https://en.wikipedia.org/wiki/Source-to-source_compiler
- https://en.wikipedia.org/wiki/Interpreter_(computing)
- …
- https://en.wikipedia.org/wiki/Polyglot_(computing)

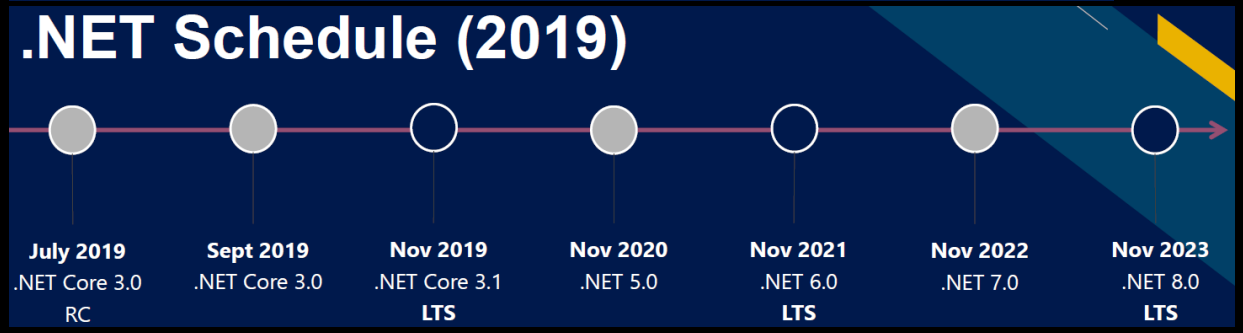
  Polyglot — use the best tool for the right jobs: high performance, scripting, web, functional programming, etc

1.1 The most important Polyglot Runtimes (GraalVM, .Net, WASM…)

GOTC

.Net

- https://en.wikipedia.org/wiki/.NET
- **Roadmap:**



Source: https://www.slideshare.net/marco.parenzan/mini-net-conf-2020-239489909

- https://www.infoworld.com/article/3608611/whats-new-in-microsoft-net-6.html
- https://devblogs.microsoft.com/dotnet/announcing-net-6-preview-6/

2) DSL

- https://en.wikipedia.org/wiki/Domain-specific_language

A **domain-specific language** (**DSL**) is a computer language specialized to a particular application domain. This is in contrast to a general-purpose language (GPL), which is broadly applicable across domains. There are a wide variety of DSLs, ranging from widely used languages for common domains, such as HTML for web pages, down to languages used by only one or a few pieces of software, such as MUSH soft code. DSLs can be further subdivided by the kind of language, and include domain-specific *markup* languages, domain-specific *modeling* languages (more generally, specification languages), and domain-specific *programming* languages. Special-purpose computer languages have always existed in the computer age, but the term "domain-specific language" has become more popular due to the rise of domain-specific modeling. Simpler DSLs, particularly ones used by a single application, are sometimes informally called **mini-languages**.

The line between general-purpose languages and domain-specific languages is not always sharp, as a language may have specialized features for a particular domain but be applicable more broadly, or conversely may in principle be capable of broad application but in practice used primarily for a specific domain. For example, Perl was originally developed as a text-processing and glue language, for the same domain as AWK and shell scripts, but was mostly used as a general-purpose programming language later on. By contrast, PostScript is a Turing complete language, and in principle can be used for any task, but in practice is narrowly used as a page description language.

- Pros & Cons:

Some of the advantages:[2][3]

- Domain-specific languages allow solutions to be expressed in the idiom and at the level of abstraction of the problem domain. The idea is that domain experts themselves may understand, validate, modify, and often even develop domain-specific language programs. However, this is seldom the case.[8]
- Domain-specific languages allow validation at the domain level. As long as the language constructs are safe any sentence written with them can be considered safe.[*citation needed*]
- Domain-specific languages can help to shift the development of business information systems from traditional software developers to the typically larger group of domain-experts who (despite having less technical expertise) have a deeper knowledge of the domain.[9]
- Domain-specific languages are easier to learn, given their limited scope.

Some of the disadvantages:

- Cost of learning a new language vs. its limited applicability
- Cost of designing, implementing, and maintaining a domain-specific language as well as the tools required to develop with it (IDE)
- Finding, setting, and maintaining proper scope.
- Difficulty of balancing trade-offs between domain-specificity and general-purpose programming language constructs.
- Potential loss of processor efficiency compared with hand-coded software.
- Proliferation of similar non-standard domain-specific languages, for example, a DSL used within one insurance company versus a DSL used within another insurance company.[10]
- Non-technical domain experts can find it hard to write or modify DSL programs by themselves.[8]
- Increased difficulty of integrating the DSL with other components of the IT system (as compared to integrating with a general-purpose language).
- Low supply of experts in a particular DSL tends to raise labor costs.
- Harder to find code examples.

GOTC

2.1 P4

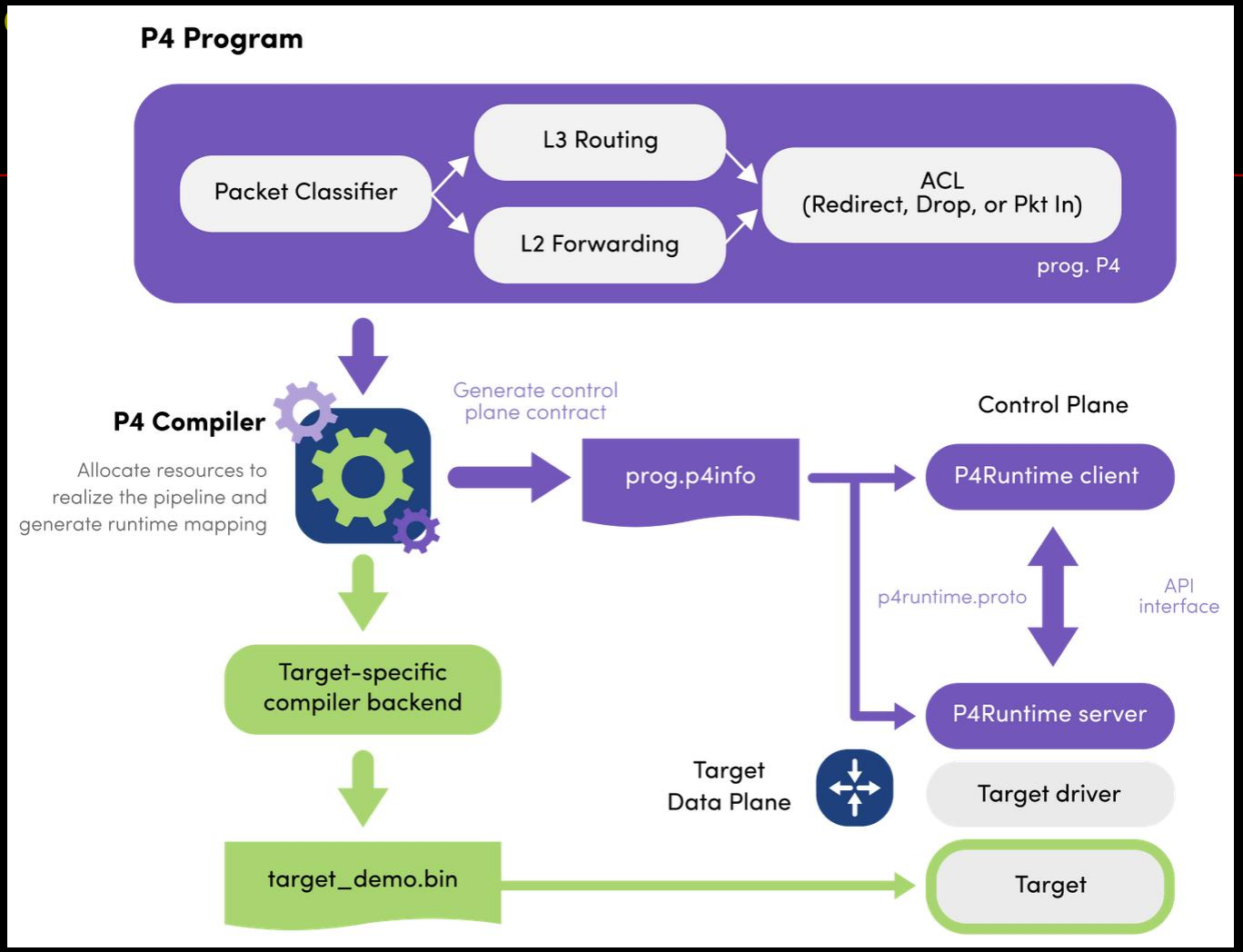
- https://en.wikipedia.org/wiki/P4_(programming_language)
- https://p4.org/

Programming Protocol-independent Packet Processors (P4) is a domain-specific language for network devices, specifying how data plane devices (switches, NICs, routers, filters, etc.) process packets.

```
table routing {
  key = { ipv4.dstAddr : lpm; }
  actions = { drop; route; }
  size : 204  8;
}
control ingress() {
  apply {
   routing.apply();
  }
}
```

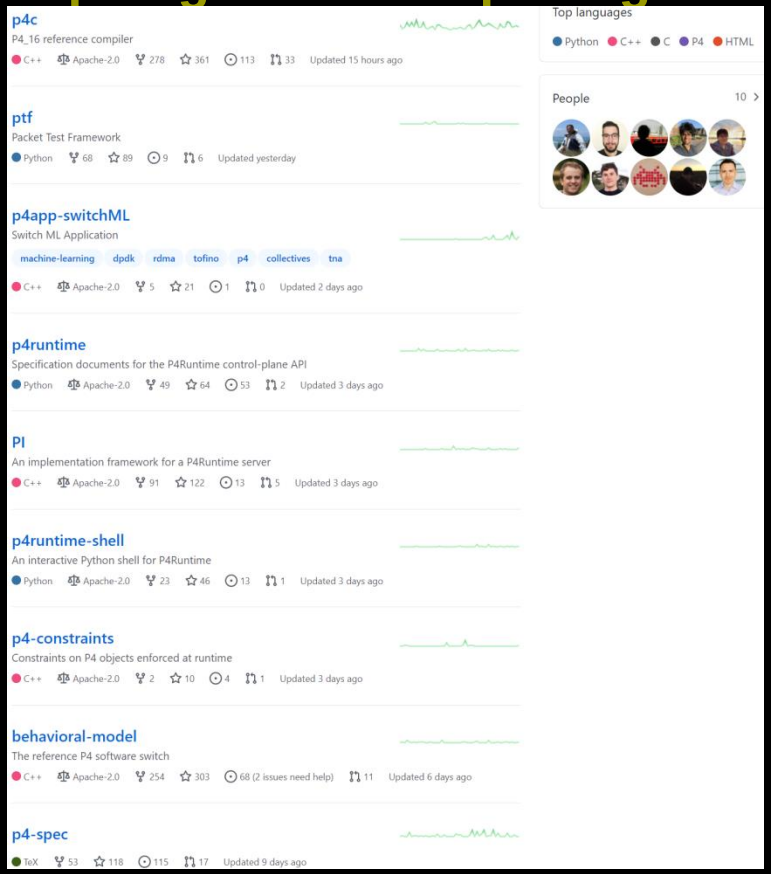
- Features:
  - C-like, strongly typed language
  - Type and memory-safe (no pointers)
  - Bounded execution (no loops)
  - Statically allocated (no malloc, no recursion)
- https://p4.org/specs/
- https://p4.org/learn/
- https://p4.org/ecosystem/

Workflow


P4 Program
Packet Classifier
L3 Routing
L2 Forwarding
ACL
(Redirect, Drop, or Pkt In)
prog. P4
P4 Compiler
Allocate resources to realize the pipeline and generate runtime mapping
Generate control plane contract
prog.p4info
Control Plane
P4Runtime client
p4runtime.proto
API interface
Target-specific compiler backend
P4Runtime server
Target Data Plane
Target driver
target_demo.bin
Target

Development

- https://github.com/p4lang/education/blob/master/GettingStarted.md
- https://github.com/p4lang/tutorials
- https://github.com/p4lang/



- ...

3) eBPF

3.1 Overview

- https://en.wikipedia.org/wiki/Berkeley_Packet_Filter

BPF (aka cBPF)

- Introduced in kernel 2.1.75 (1997)
- https://blog.cloudflare.com/bpf-the-forgotten-bytecode/
-

```
# tcpdump host 127.0.0.1 and port 22 -d
(000) ldh      [12]
(001) jeq      #0x800           jt 2    jf 18
(002) ld       [26]
(003) jeq      #0x7f000001      jt 6    jf 4
(004) ld       [30]
(005) jeq      #0x7f000001      jt 6    jf 18
(006) ldb      [23]
(007) jeq      #0x84            jt 10   jf 8
(008) jeq      #0x6             jt 10   jf 9
(009) jeq      #0x11            jt 10   jf 18
(010) ldh      [20]
(011) jset     #0x1fff          jt 18   jf 12
(012) ldxb     4*([14]&0xf)
(013) ldh      [x + 14]
[...]
```

Optimizes packet filter performance

2 x 32-bit registers & scratch memory

User-defined bytecode executed by an in-kernel sandboxed virtual machine

Steven McCanne and Van Jacobson, 1993

Source: https://www.slideshare.net/brendangregg/kernel-recipes-2017-performance-analysis-with-bpf

eBPF (extended BPF)

- Since Linux Kernel v3.15 and ongoing
- Aims at being a universal in-kernel virtual machine
- A simple way to extend the functionality of Kernel at runtime
- “DTrace(

) for Linux”

BPF VM

GOTC

Factor	Classic BPF	Extended BPF
Register count	2: A, X	10: R0–R9, plus R10 as a read-only frame pointer
Register width	32-bit	64-bit
Storage	16 memory slots: M[0–15]	512 bytes of stack space, plus infinite “map” storage
Restricted kernel calls	Very limited, JIT specific	Yes, via the bpf_call instruction
Event targets	Packets, seccomp-BPF	Packets, kernel functions, user functions, tracepoints, user markers, PMCs

Source: https://brendangregg.com/bpf-performance-tools-book.html

BPF Runtime Internals

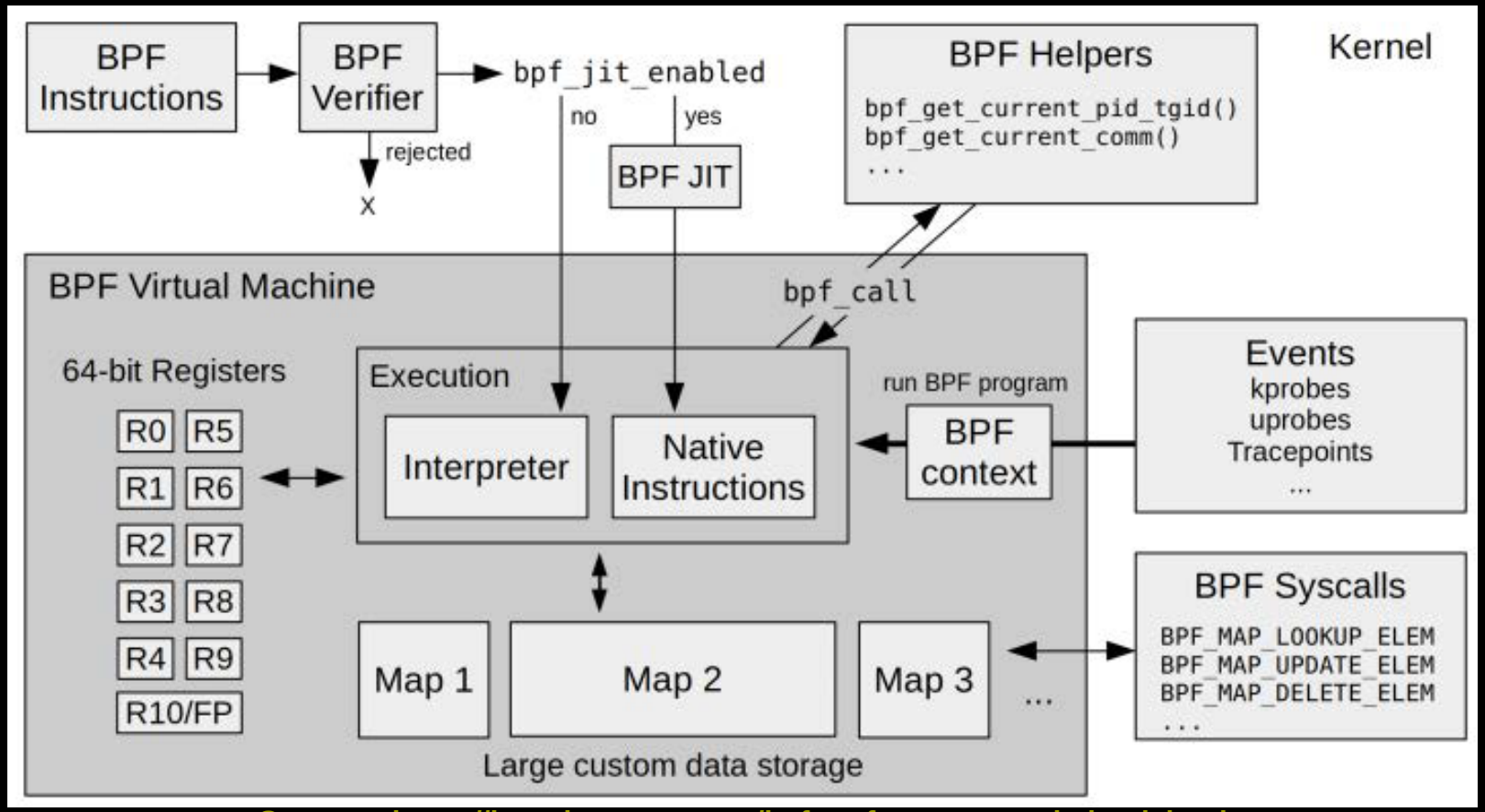
GOTC

- bpf() system call

  http://www.man7.org/linux/man-pages/man2/bpf.2.html

```
#include <linux/bpf.h>

int bpf(int cmd, union bpf_attr *attr, unsigned int size);
```

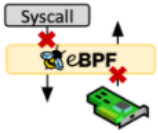
-



Source: https://brendangregg.com/bpf-performance-tools-book.html

- https://ebpf.io/

Security

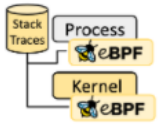


Building on the foundation of seeing and understanding all system calls and combining that with a packet and socket-level view of all networking operations allows for revolutionary new approaches to securing systems. While aspects of system call filtering, network-level filtering, and process context tracing have typically been handled by completely independent systems, eBPF allows for combining the visibility and control of all aspects to create security systems operating on more context with better level of control.

Networking

The combination of programmability and efficiency makes eBPF a natural fit for all packet processing requirements of networking solutions. The programmability of eBPF enables adding additional protocol parsers and easily program any forwarding logic to meet changing requirements without ever leaving the packet processing context of the Linux kernel. The efficiency provided by the JIT compiler provides execution performance close to that of natively compiled in-kernel code.

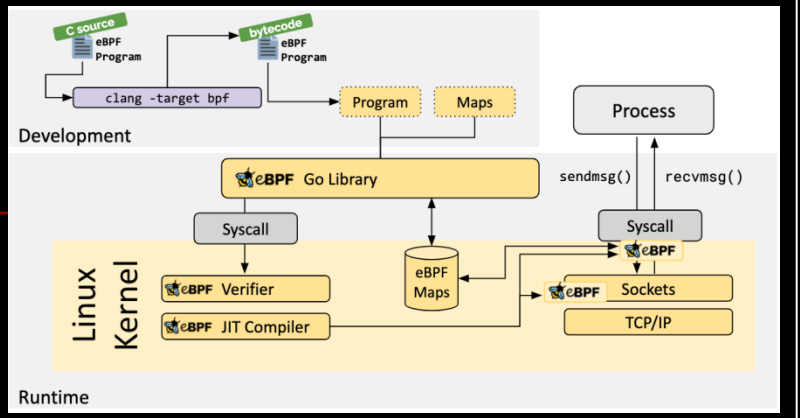
Tracing & Profiling



The ability to attach eBPF programs to trace points as well as kernel and user application probe points allows unprecedented visibility into the runtime behavior of applications and the system itself. By giving introspection abilities to both the application and system side, both views can be combined, allowing powerful and unique insights to troubleshoot system performance problems. Advanced statistical data structures allow to extract meaningful visibility data in an efficient manner, without requiring the export of vast amounts of sampling data as typically done by similar systems.

Observability & Monitoring

Instead of relying on static counters and gauges exposed by the operating system, eBPF enables the collection & in-kernel aggregation of custom metrics and generation of visibility events based on a wide range of possible sources. This extends the depth of visibility that can be achieved as well as reduces the overall system overhead significantly by only collecting the visibility data required and by generating histograms and similar data structures at the source of the event instead of relying on the export of samples.

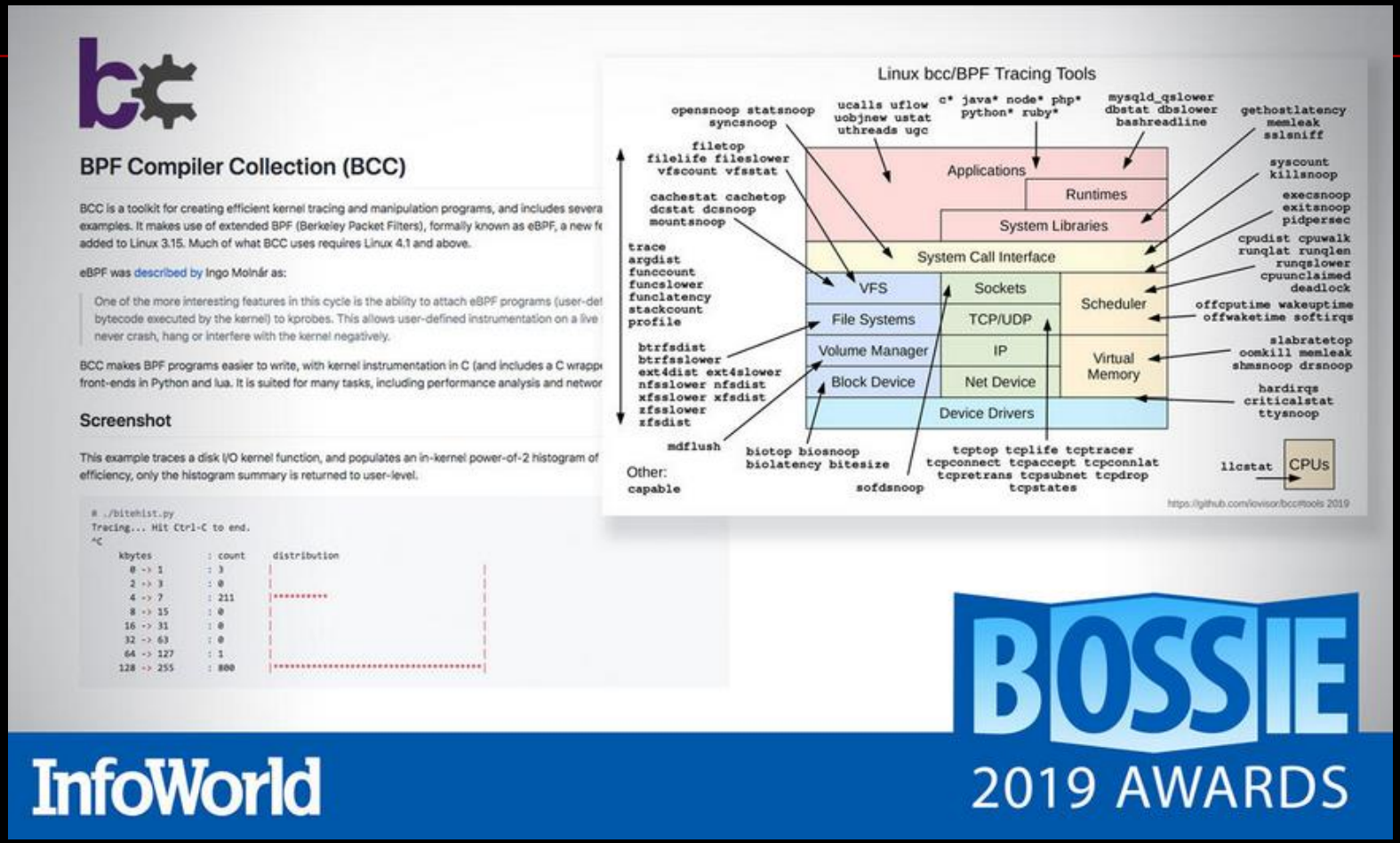


https://ebpf.io/what-is-ebpf/

https://ebpf.io/projects/

3.2 BCC (BPF Compiler Collection)

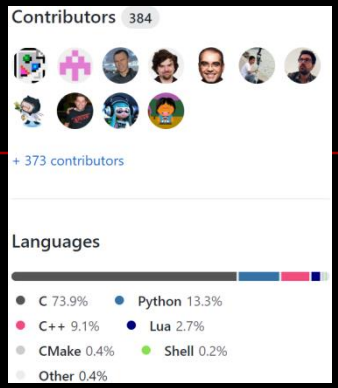
- https://www.infoworld.com/article/3444198/the-best-open-source-software-of-2019.html



What is it

GOTC

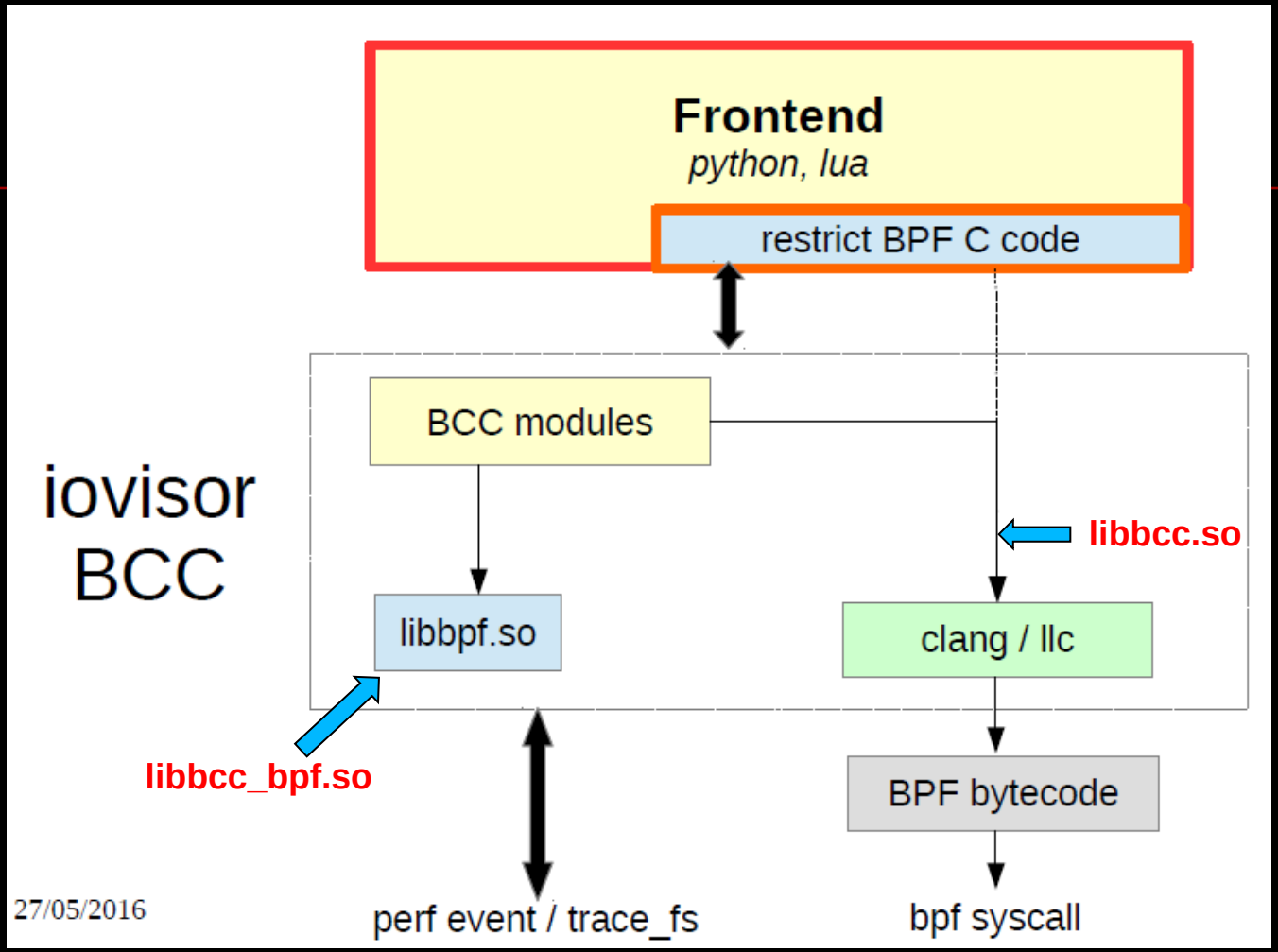
- https://github.com/iovisor/bcc/



a toolkit with Python/Lua frontend for compiling, loading, and executing BPF programs, which allows user-defined instrumentation on a live kernel image:

- compile BPF program from C source
- attach BPF program to kprobe/uprobe/tracepoint/USDT/socket
- poll data from BPF program
- framework for building new tools or one-off scripts
- contains a P4 compiler for BPF targets
- additional projects to support Go, Rust, and DTrace-style frontend
- https://github.com/iovisor/bcc/blob/master/docs/kernel-versions.md
- …

Architecture



Source: http://www.slideshare.net/vh21/meet-cutebetweenebpfandtracing

- https://github.com/libbpf

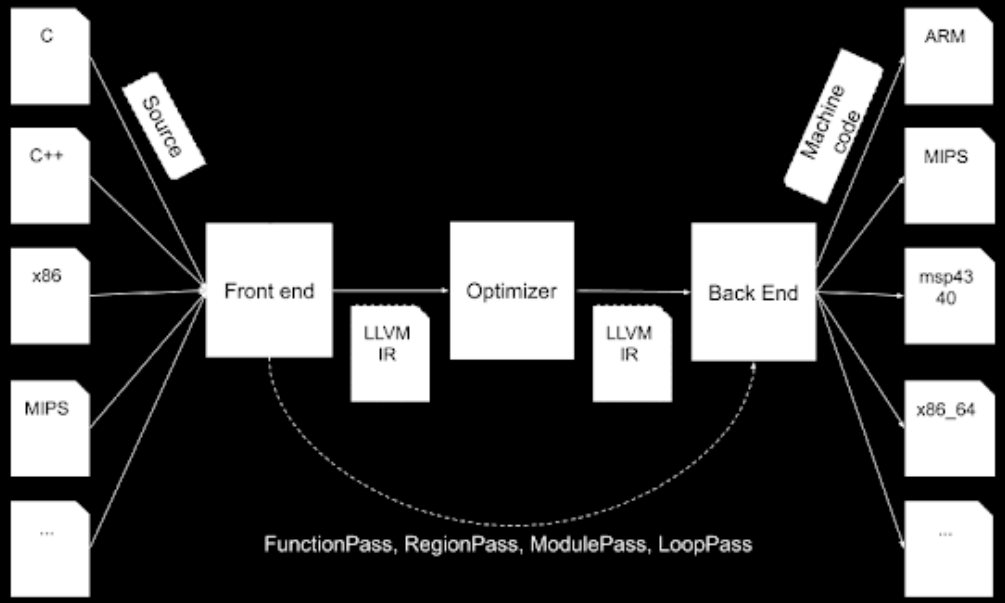
4) LLVM

4.1 Overview

- https://en.wikipedia.org/wiki/LLVM

LLVM is a set of compiler and toolchain technologies,[5] which can be used to develop a front end for any programming language and a back end for any instruction set architecture. LLVM is designed around a language-independent intermediate representation (IR) that serves as a portable, high-level assembly language that can be optimized with a variety of transformations over multiple passes.[6]

- https://llvm.org
- http://clang.llvm.org/
-



Source: http://blog.k3170makan.com/2020/04/learning-llvm-i-introdution-to-llvm.html

- https://www.llvm.org/ProjectsWithLLVM/

IR

The core of LLVM is the intermediate representation (IR), a low-level programming language similar to assembly. IR is a strongly typed reduced instruction set computing (RISC) instruction set which abstracts away most details of the target. For example, the calling convention is abstracted through *call* and *ret* instructions with explicit arguments. Also, instead of a fixed set of registers, IR uses an infinite set of temporaries of the form %0, %1, etc. LLVM supports three equivalent forms of IR: a human-readable assembly format, an in-memory format suitable for frontends, and a dense bitcode format for serializing. A simple "Hello, world!" program in the IR format:[32]

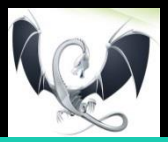
```
@.str = internal constant [14 x i8] c"hello, world\0A\00"

declare i32 @printf(i8*, ...)

define i32 @main(i32 %argc, i8** %argv) nounwind {
entry:
    %tmp1 = getelementptr [14 x i8], [14 x i8]* @.str, i32 0, i32 0
    %tmp2 = call i32 (i8*, ...) @printf( i8* %tmp1 ) nounwind
    ret i32 0
}
```

- https://llvm.org/docs/LangRef.html
- MLIR:
  https://mlir.llvm.org/

- Bitcode:
  https://llvm.org/docs/BitCodeFormat.html
  https://llvm.org/docs/CommandGuide/llvm-bcanalyzer.html
  https://zhuanlan.zhihu.com/p/308201373

GOTC

LLVM vs GCC



GPL v3	UIUC, MIT
Front-end: CC1 / CPP	Front-end: Clang
ld.bfd / ld.gold	lld / mclinker
gdb	lldb
as / objdump	MC layer
glibc	llvm-libc?
libstdc++	libc++
libsupc++	libc++abi
libgcc	libcompiler-rt
libgccjit	libLLVMMCJIT
. . .	ORC JIT, Coroutines, Clangd, libclc, Falcon...

- http://lld.llvm.org/
- https://llvm.org/docs/Proposals/LLVMLibC.html

4.2 LLVM support for eBPF

- eBPF backend firstly introduced in LLVM 3.7 release
- http://llvm.org/docs/CodeGenerator.html#the-extended-berkeley-packet-filter-ebpf-backend
-
  - Enabled by default with all major distributions
    - Registered targets: `llc --version`
    - `llc`'s BPF `-march` options: bpf, bpfeb, bpfel
    - `llc`'s BPF `-mcpu` options: generic, v1, v2, probe
  - Assembler output through `-S` supported
  - `llvm-objdump` for disassembler and code annotations (via DWARF)
  - Annotations correlate directly with kernel verifier log
  - Outputs ELF file with maps as relocation entries
    - Processed by BPF loaders (e.g. iproute2) and pushed into kernel

  Source: https://ossna2017.sched.com/event/BCsg/making-the-kernels-networking-data-path-programmable-with-bpf-and-xdp-daniel-borkmann-covalent
- $LLVM_SRC/lib/Target/BPF
- http://cilium.readthedocs.io/en/latest/bpf/
- LLVM 9.0 Released With Ability To Build The Linux x86_64

5) WASM

5.1 Overview

- https://en.wikipedia.org/wiki/WebAssembly

C source code and corresponding WebAssembly

C source code	WebAssembly `.wat` text format	WebAssembly `.wasm` binary format

`int factorial(int n) {`
`  if (n == 0)`
`  return 1;`
`  else`
`  return n * factorial(n-1);`
`}`
	`(func (param i64) (result i64)`
	`  local.get 0`
	`  i64.eqz`
	`  if (result i64)`
	`  i64.const 1`
	`  else`
	`  local.get 0`
	`  local.get 0`
	`  i64.const 1`
	`  i64.sub`
	`  call 0`
	`  i64.mul`
	`  end)`
		`00 61 73 6D 01 00 00 00`
		`01 00 01 60 01 73 01 73 06`
		`03 00 01 00 02`
		`0A 00 01`
		`00 00`
		`20 00`
		`50`
		`04 7E`
		`42 01`
		`05`
		`20 00`
		`20 00`
		`42 01`
		`7D`
		`10 00`
		`7E`
		`0B`
		`0B 15 17`

- https://webassembly.org/

  WebAssembly (abbreviated *Wasm*) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable compilation target for programming languages, enabling deployment on the web for client and server applications.

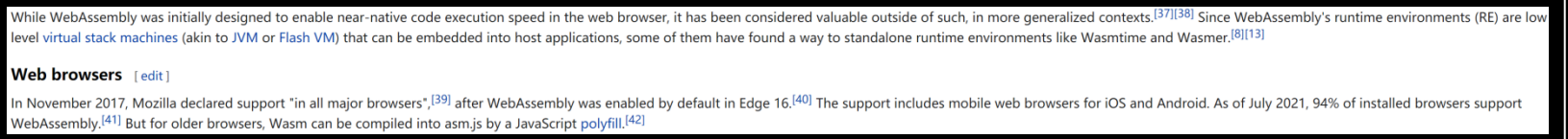
- https://github.com/WebAssembly/design

- WebAssembly 1.0 has shipped in 4 major browser engines.

- https://webassembly.org/getting-started/developers-guide/

Implementations

GOTC

- 

While WebAssembly was initially designed to enable near-native code execution speed in the web browser, it has been considered valuable outside of such, in more generalized contexts.[37][38] Since WebAssembly's runtime environments (RE) are low level virtual stack machines (akin to JVM or Flash VM) that can be embedded into host applications, some of them have found a way to standalone runtime environments like Wasmtime and Wasmer.[8][13]

Web browsers [edit]

In November 2017, Mozilla declared support "in all major browsers",[39] after WebAssembly was enabled by default in Edge 16.[40] The support includes mobile web browsers for iOS and Android. As of July 2021, 94% of installed browsers support WebAssembly.[41] But for older browsers, Wasm can be compiled into asm.js by a JavaScript polyfill.[42]

- **https://github.com/appcypher/awesome-wasm-langs**
- **https://github.com/appcypher/awesome-wasm-runtimes**

- **WASM & Rust**
  https://www.rust-lang.org/what/wasm
  https://rustwasm.github.io/book
  https://github.com/rustwasm

Compilers

GOTC

Because WebAssembly executables are precompiled, it is possible to use a variety of programming languages to make them.[43] This is achieved either through direct compilation to Wasm, or through implementation of the corresponding virtual machines in Wasm. There have been around 40 programming languages reported to support Wasm as a compilation target.[44]

Emscripten compiles C and C++ to Wasm[32] using the Binaryen and LLVM as backend.[45]

As of version 8, a standalone Clang can compile C and C++ to Wasm.[46]

Its initial aim is to support compilation from C and C++,[47] though support for other source languages such as Rust, .NET languages[48][49][44] and AssemblyScript[50] (TypeScript-like) is also emerging. After the MVP release, there are plans to support multithreading and garbage collection[51][52] which would make WebAssembly a compilation target for garbage-collected programming languages like C# (supported via Blazor), F# (supported via Bolero[53] with help of Blazor), Python, and even JavaScript where the browser's just-in-time compilation speed is considered too slow. A number of other languages have some support including Python,[54] Java,[55] Julia,[56][57][58] Zig,[59] and Ruby,[60] as well as Go.[61]

Limitations

GOTC

1. In general, WebAssembly does not allow direct interaction with the DOM. All interaction must flow through JavaScript interop.
2. Multithreading (although there are plans to address this.)
3. Garbage collection (although there are plans to address this.)
4. Security considerations (discussed below)

WebAssembly is supported on desktops, and mobile, but on the latter, in practice (for non-small memory allocations, such as with Unity game engine) there are "grave limitations that make many applications infeasible to be *reliably* deployed on mobile browsers [..] Currently allocating more than ~300MB of memory is not reliable on Chrome on Android without resorting to Chrome-specific workarounds, nor in Safari on iOS."[62]

All major web browsers allow WebAssembly if Content-Security-Policy is not specified, or if "unsafe-eval" is used, but otherwise the major web browsers behave differently.[63] In practice WebAssembly can't be used on Chrome without "unsafe-eval",[64][65] while a worker thread workaround is available.[66]

5.2 Beyond the browser

- https://webassembly.org/docs/non-web/
- https://www.zdnet.com/article/microsoft-google-back-bytecode-alliance-to-move-webassembly-beyond-the-browser/
- …

WASI

- **WebAssembly System Interface**

WebAssembly System Interface (WASI) is a simple interface (ABI and API) designed by Mozilla intended to be portable to any platform.[77] It provides POSIX-like features like file I/O constrained by capability-based security.[78][79] There are also a few other proposed ABI/APIs.[80][81]

WASI is influenced by CloudABI and Capsicum.

Solomon Hykes, a co-founder of Docker, wrote in 2019, "If WASM+WASI existed in 2008, we wouldn't have needed to create Docker. That's how important it is. WebAssembly on the server is the future of computing."[82] Wasmer, out in version 1.0, provides "software containerization, we create universal binaries that work anywhere without modification, including operating systems like Linux, macOS, Windows, and web browsers. Wasm automatically sandboxes applications by default for secure execution".[82]

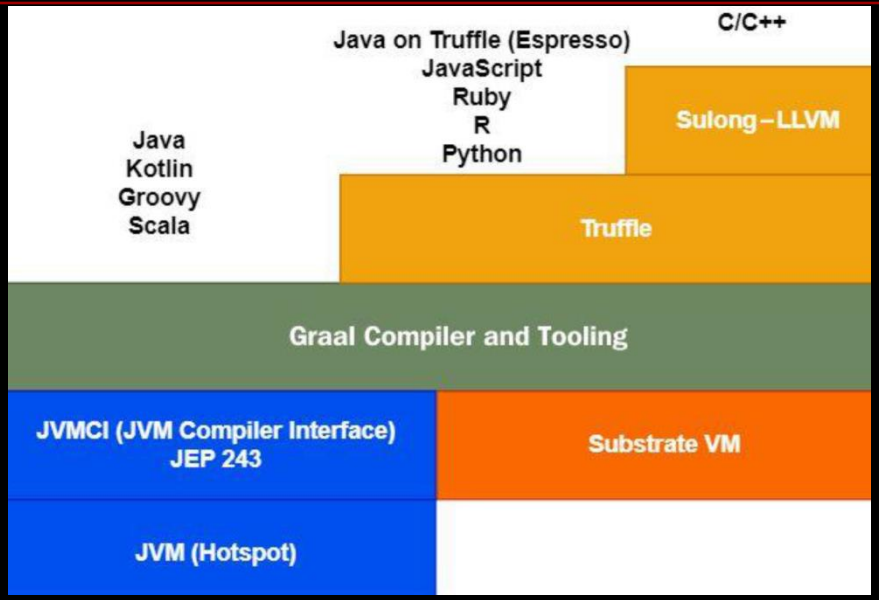
- https://wasi.dev/
- https://github.com/WebAssembly/WASI

- https://github.com/bytecodealliance/wasmtime/blob/main/docs/WASI-documents.md
- https://hacks.mozilla.org/2019/03/standardizing-wasi-a-webassembly-system-interface/
- https://training.linuxfoundation.org/announcements/wasi-bringing-webassembly-way-beyond-browsers/

6) GraalVM

6.1 Overview

- https://en.wikipedia.org/wiki/GraalVM
- https://www.graalvm.org/
-

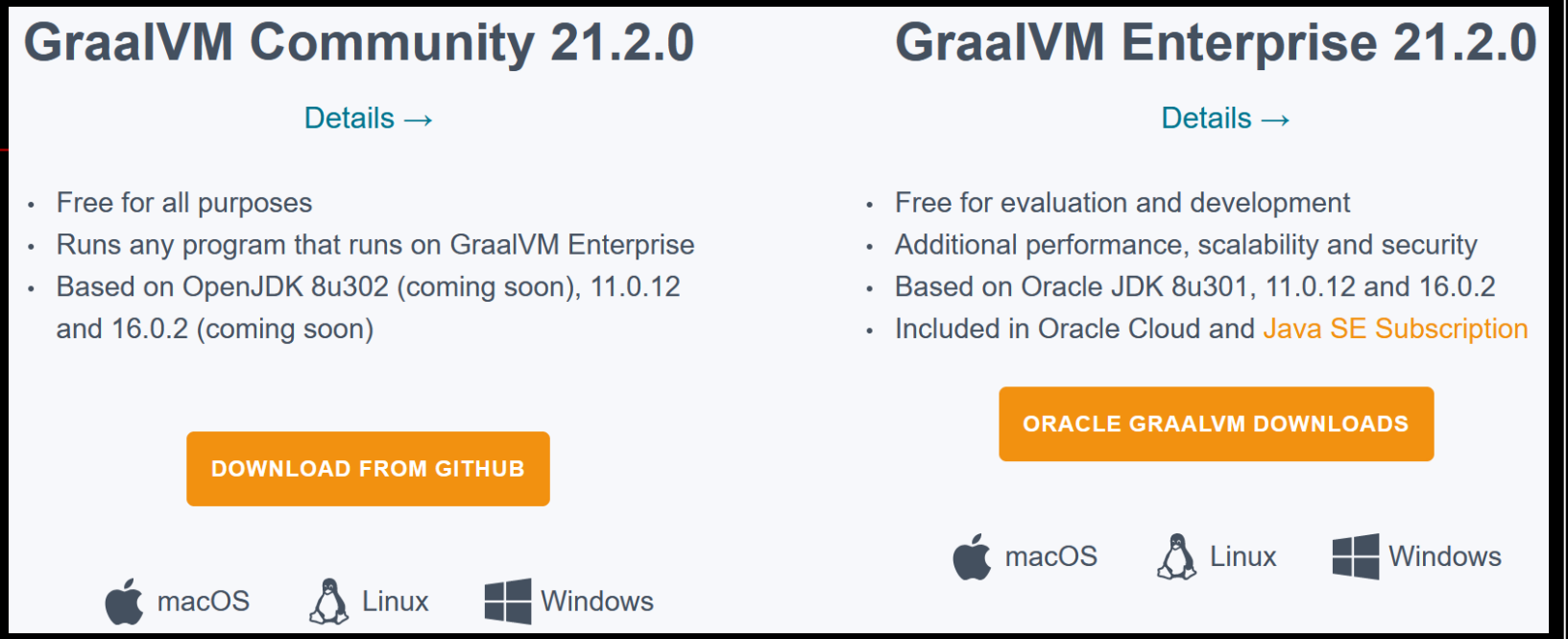


Source: https://static.packt-cdn.com/downloads/9781800564909_ColorImages.pdf

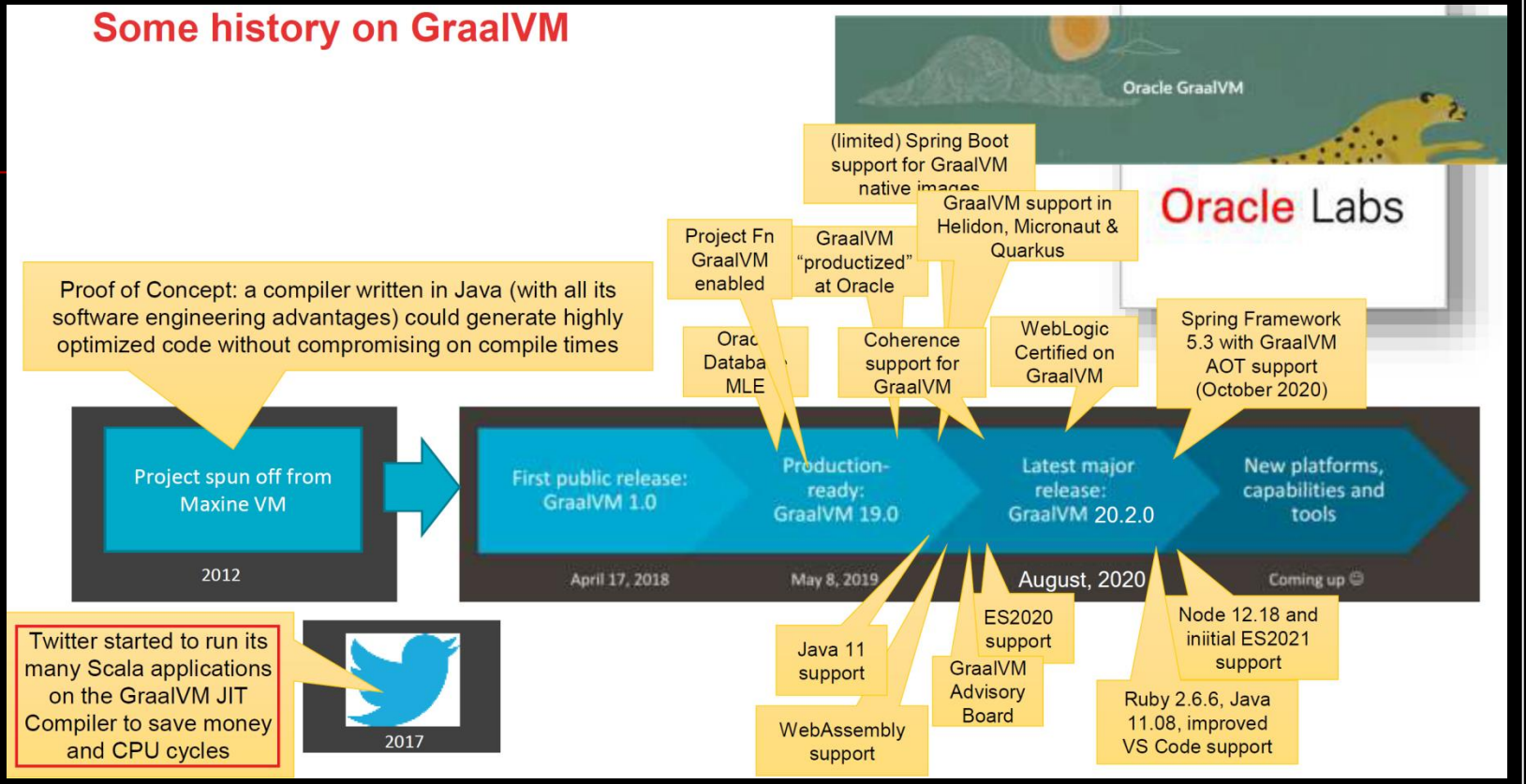
- A Universal High-Performance Polyglot VM
- A meta-runtime for Language-Level Virtualization
- Currently base an Oracle Labs JDK 8, 11, 16 with JVMCI support
- https://www.graalvm.org/docs/introduction/
- https://github.com/oracle/graal/blob/master/docs/

Editions

- CE(Community Edition) vs EE(Enterprise Edition)



History

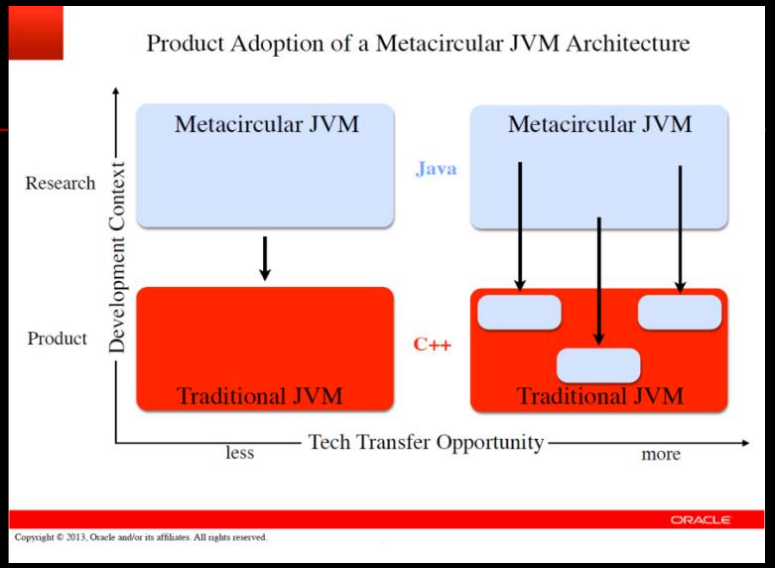


Source: "How and why GraalVM is quickly becoming relevant for you", Lucas Jellema, DOAG 2020.

GOTC

Meta-Circular

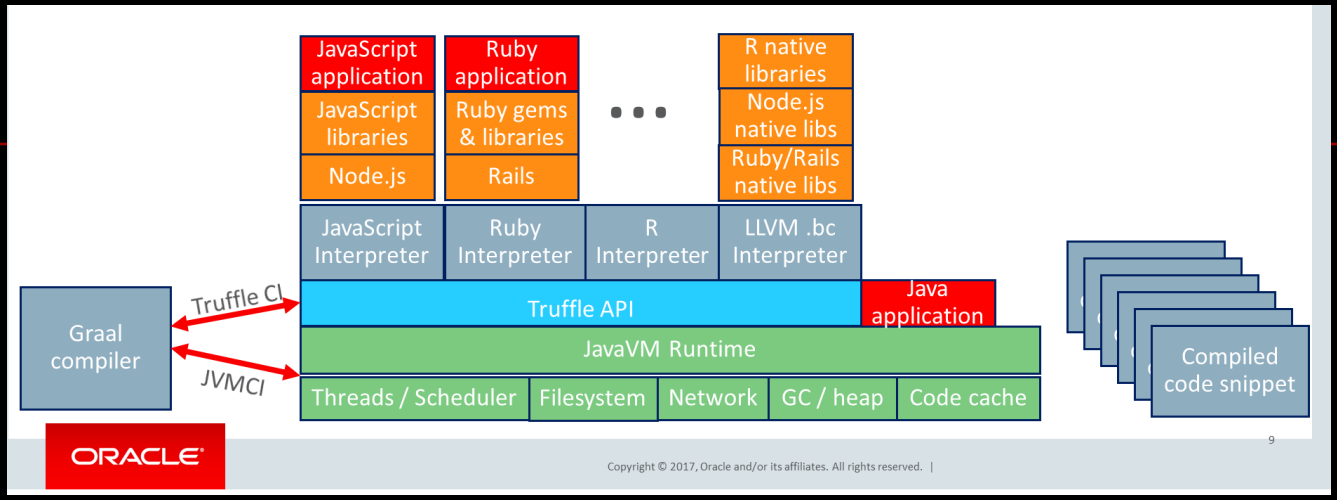
- Maxine → GraalVM



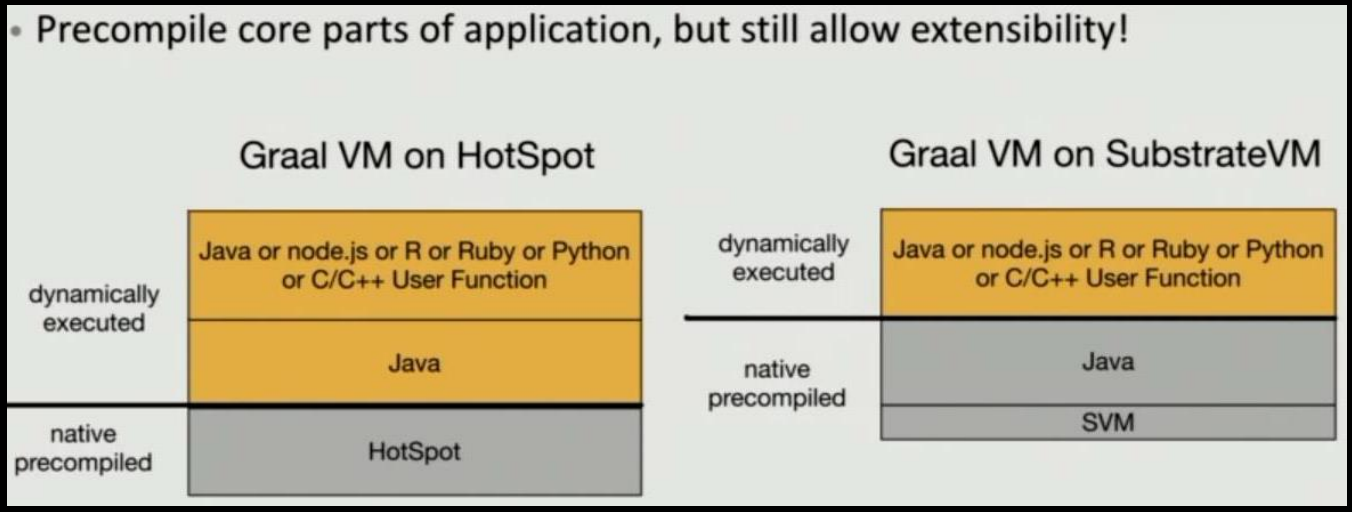
Source: https://chrisseaton.com/truffleruby/jokerconf17/

Architecture

- **A hybrid of static & dynamic runtimes**



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

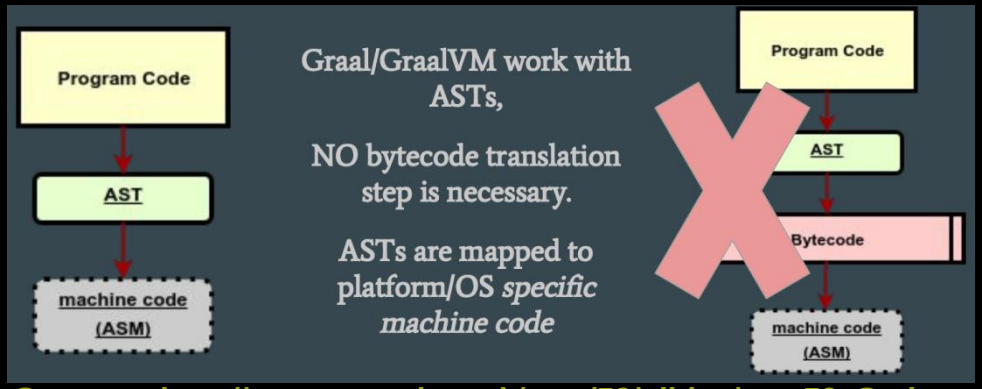


Source: "Adopting Java for the Serverless world", Vadym Kazulkin, JUG London 2020.

GOTC

Truffle & Graal

- https://github.com/neomatrix369/awesome-graal
- AST
  https://en.wikipedia.org/wiki/Abstract_syntax_tree
  Graal/GraalVM: ASTs as first class citizen

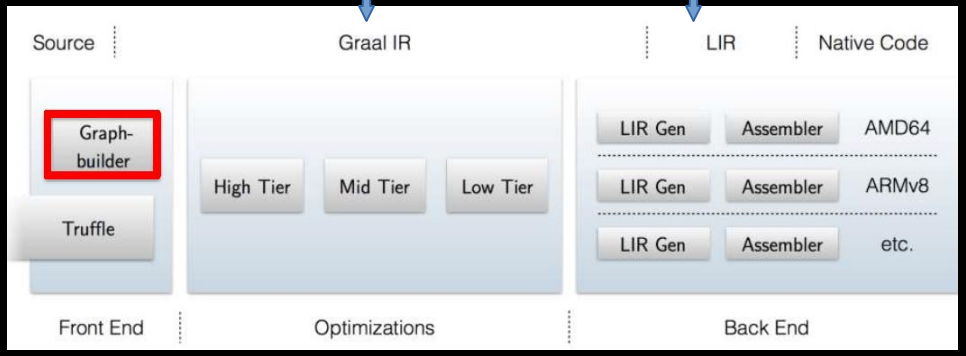


Source: http://crest.cs.ucl.ac.uk/cow/59/slides/cow59_Sarkar.pdf

- IR

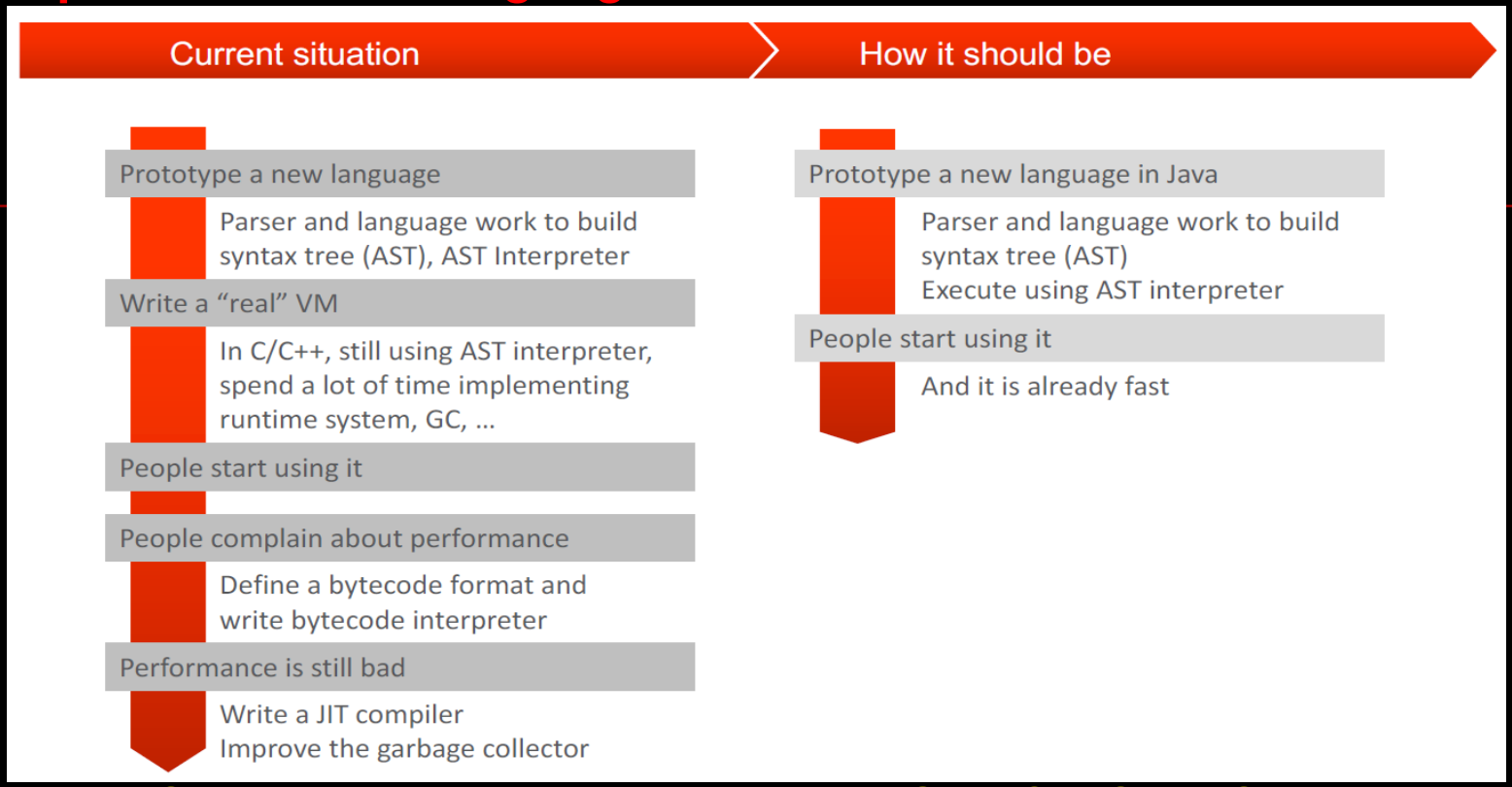
Graph-Based IR
Platform Independent

Graph-Based IR
Platform Dependent



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

- **Implement a new language runtime**



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

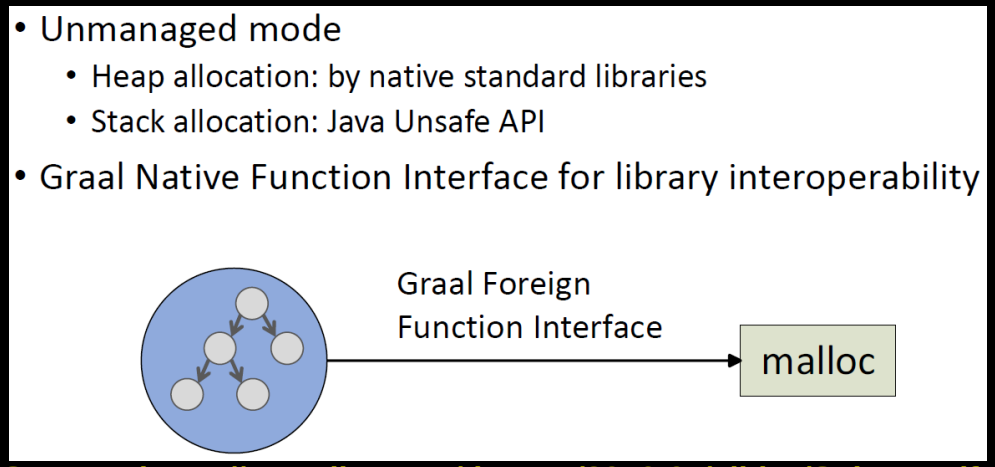
- https://www.graalvm.org/graalvm-as-a-platform/language-implementation-framework/
- https://github.com/oracle/graal/blob/master/truffle/docs/Languages.md

Sulong

GOTC

- https://github.com/oracle/graal/tree/master/sulong
- 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.runtime/src/com/oracle/truffle/llvm/runtime/nodes
- Memory



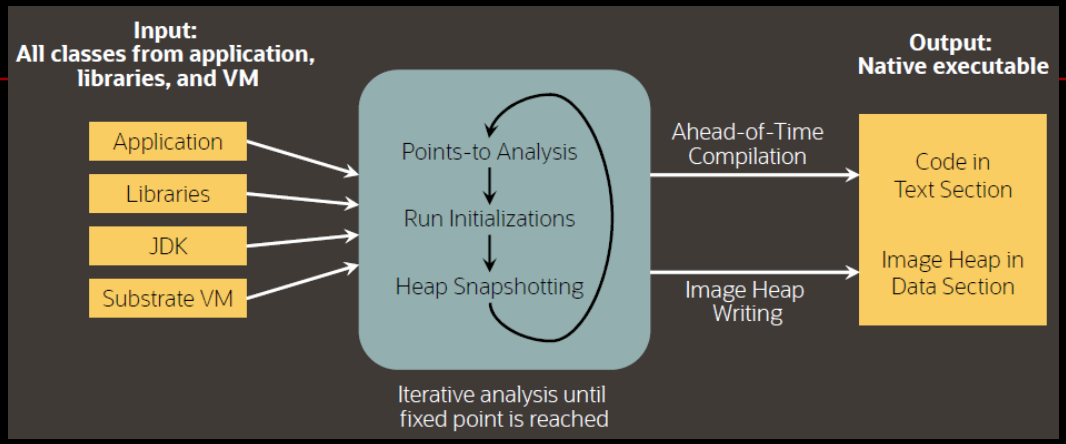
Source: https://www.llvm.org/devmtg/2016-01/slides/Sulong.pdf

- https://www.graalvm.org/reference-manual/llvm/
- https://www.graalvm.org/reference-manual/llvm/Compiling/
- lli (directly execute programs from LLVM bitcode)

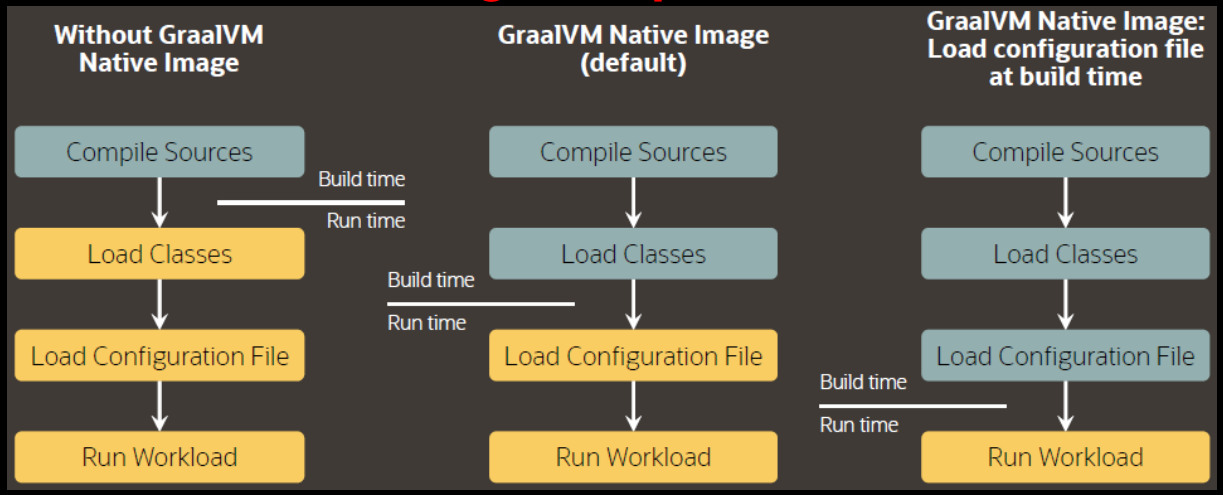
  https://llvm.org/docs/CommandGuide/lli.html

Native Image

- https://www.graalvm.org/reference-manual/native-image/
- https://www.graalvm.org/examples/native-image-examples/
-



- Benefits of the Image Heap



Source: https://www.jug-gr.de/downloads/JDK_14_und_GraalVM_im_Java-%C3%96kosystem_WW.pdf

GOTC

Substrate VM(SVM)

https://llvJEP 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

GOTC

Src

- https://github.com/oracle/graal

Repository Structure

The GraalVM main source repository includes the components listed below. The documentation for each component includes developer instructions for the component.

- GraalVM SDK contains long term supported APIs of GraalVM.
- GraalVM 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.
- GraalWasm is an engine for running WebAssembly programs 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.
- VS Code provides extensions to Visual Studio Code that support development of polyglot applications using GraalVM.

- https://github.com/graalvm
- https://github.com/oracle/graalpython
- …

GOTC

Getting started

- https://www.graalvm.org/docs/getting-started/
- https://www.graalvm.org/reference-manual/graalvm-updater/
- Assets:

e.g.,https://github.com/graalvm/graalvm-ce-builds/releases/tag/vm-21.2.0

- espresso-installable-java11-darwin-amd64-21.2.0.jar
- espresso-installable-java11-darwin-amd64-21.2.0.jar.sha256
- espresso-installable-java11-linux-aarch64-21.2.0.jar
- espresso-installable-java11-linux-aarch64-21.2.0.jar.sha256
- espresso-installable-java11-linux-amd64-21.2.0.jar
- espresso-installable-java11-linux-amd64-21.2.0.jar.sha256
- espresso-installable-java11-windows-amd64-21.2.0.jar
- espresso-installable-java11-windows-amd64-21.2.0.jar.sha256
- espresso-installable-java8-linux-amd64-21.2.0.jar
- espresso-installable-java8-linux-amd64-21.2.0.jar.sha256
- espresso-installable-java8-windows-amd64-21.2.0.jar
- espresso-installable-java8-windows-amd64-21.2.0.jar.sha256
- graalvm-ce-java11-darwin-amd64-21.2.0.tar.gz
- graalvm-ce-java11-darwin-amd64-21.2.0.tar.gz.sha256
- graalvm-ce-java11-linux-aarch64-21.2.0.tar.gz
- graalvm-ce-java11-linux-aarch64-21.2.0.tar.gz.sha256
- graalvm-ce-java11-linux-amd64-21.2.0.tar.gz
- graalvm-ce-java11-linux-amd64-21.2.0.tar.gz.sha256
- graalvm-ce-java11-windows-amd64-21.2.0.zip
- graalvm-ce-java11-windows-amd64-21.2.0.zip.sha256
- graalvm-ce-java16-darwin-amd64-21.2.0.tar.gz
- graalvm-ce-java16-darwin-amd64-21.2.0.tar.gz.sha256
- graalvm-ce-java16-linux-aarch64-21.2.0.tar.gz
- graalvm-ce-java16-linux-aarch64-21.2.0.tar.gz.sha256
- graalvm-ce-java16-linux-amd64-21.2.0.tar.gz
- graalvm-ce-java16-linux-amd64-21.2.0.tar.gz.sha256
- graalvm-ce-java16-windows-amd64-21.2.0.zip
- graalvm-ce-java16-windows-amd64-21.2.0.zip.sha256
- graalvm-ce-java8-linux-amd64-21.2.0.tar.gz
- graalvm-ce-java8-linux-amd64-21.2.0.tar.gz.sha256
- graalvm-ce-java8-windows-amd64-21.2.0.zip
- graalvm-ce-java8-windows-amd64-21.2.0.zip.sha256
- llvm-toolchain-installable-java11-darwin-amd64-21.2.0.jar
- llvm-toolchain-installable-java11-darwin-amd64-21.2.0.jar.sha256
- llvm-toolchain-installable-java11-linux-aarch64-21.2.0.jar
- llvm-toolchain-installable-java11-linux-aarch64-21.2.0.jar.sha256
- llvm-toolchain-installable-java11-linux-amd64-21.2.0.jar
- llvm-toolchain-installable-java11-linux-amd64-21.2.0.jar.sha256
- llvm-toolchain-installable-java16-darwin-amd64-21.2.0.jar
- llvm-toolchain-installable-java16-darwin-amd64-21.2.0.jar.sha256
- llvm-toolchain-installable-java16-linux-aarch64-21.2.0.jar
- llvm-toolchain-installable-java16-linux-aarch64-21.2.0.jar.sha256
- llvm-toolchain-installable-java16-linux-amd64-21.2.0.jar
- llvm-toolchain-installable-java16-linux-amd64-21.2.0.jar.sha256
- llvm-toolchain-installable-java8-darwin-amd64-21.2.0.jar
- llvm-toolchain-installable-java8-darwin-amd64-21.2.0.jar.sha256
- llvm-toolchain-installable-java8-linux-amd64-21.2.0.jar
- llvm-toolchain-installable-java8-linux-amd64-21.2.0.jar.sha256
- native-image-installable-svm-java11-darwin-amd64-21.2.0.jar
- native-image-installable-svm-java11-darwin-amd64-21.2.0.jar.sha256
- native-image-installable-svm-java11-linux-aarch64-21.2.0.jar
- native-image-installable-svm-java11-linux-aarch64-21.2.0.jar.sha256
- native-image-installable-svm-java11-linux-amd64-21.2.0.jar
- native-image-installable-svm-java11-linux-amd64-21.2.0.jar.sha256
- native-image-installable-svm-java11-windows-amd64-21.2.0.jar
- native-image-installable-svm-java11-windows-amd64-21.2.0.jar.sha256
- native-image-installable-svm-java16-darwin-amd64-21.2.0.jar
- native-image-installable-svm-java16-darwin-amd64-21.2.0.jar.sha256
- native-image-installable-svm-java16-linux-aarch64-21.2.0.jar
- native-image-installable-svm-java16-linux-aarch64-21.2.0.jar.sha256
- native-image-installable-svm-java16-linux-amd64-21.2.0.jar
- native-image-installable-svm-java16-linux-amd64-21.2.0.jar.sha256
- native-image-installable-svm-java16-windows-amd64-21.2.0.jar
- native-image-installable-svm-java16-windows-amd64-21.2.0.jar.sha256
- native-image-installable-svm-java8-linux-amd64-21.2.0.jar
- native-image-installable-svm-java8-linux-amd64-21.2.0.jar.sha256
- native-image-installable-svm-java8-windows-amd64-21.2.0.jar
- native-image-installable-svm-java8-windows-amd64-21.2.0.jar.sha256
- wasm-installable-svm-java11-darwin-amd64-21.2.0.jar
- wasm-installable-svm-java11-darwin-amd64-21.2.0.jar.sha256
- wasm-installable-svm-java11-linux-aarch64-21.2.0.jar
- wasm-installable-svm-java11-linux-aarch64-21.2.0.jar.sha256
- wasm-installable-svm-java11-linux-amd64-21.2.0.jar
- wasm-installable-svm-java11-linux-amd64-21.2.0.jar.sha256
- wasm-installable-svm-java11-windows-amd64-21.2.0.jar
- wasm-installable-svm-java11-windows-amd64-21.2.0.jar.sha256
- wasm-installable-svm-java16-darwin-amd64-21.2.0.jar
- wasm-installable-svm-java16-darwin-amd64-21.2.0.jar.sha256
- wasm-installable-svm-java16-linux-aarch64-21.2.0.jar
- wasm-installable-svm-java16-linux-aarch64-21.2.0.jar.sha256
- wasm-installable-svm-java16-linux-amd64-21.2.0.jar
- wasm-installable-svm-java16-linux-amd64-21.2.0.jar.sha256
- wasm-installable-svm-java16-windows-amd64-21.2.0.jar
- wasm-installable-svm-java16-windows-amd64-21.2.0.jar.sha256
- wasm-installable-svm-java8-linux-amd64-21.2.0.jar
- wasm-installable-svm-java8-linux-amd64-21.2.0.jar.sha256
- wasm-installable-svm-java8-windows-amd64-21.2.0.jar
- wasm-installable-svm-java8-windows-amd64-21.2.0.jar.sha256
- graalpython-21.2.0-linux-amd64.tar.gz
- graalpython-21.2.0-linux-amd64.tar.gz.sha256
- graalpython-21.2.0-macos-amd64.tar.gz
- graalpython-21.2.0-macos-amd64.tar.gz.sha256
- python-installable-svm-java11-darwin-amd64-21.2.0.jar
- python-installable-svm-java11-darwin-amd64-21.2.0.jar.sha256
- python-installable-svm-java11-linux-amd64-21.2.0.jar
- python-installable-svm-java11-linux-amd64-21.2.0.jar.sha256
- python-installable-svm-java16-darwin-amd64-21.2.0.jar
- python-installable-svm-java16-darwin-amd64-21.2.0.jar.sha256
- python-installable-svm-java16-linux-amd64-21.2.0.jar
- python-installable-svm-java16-linux-amd64-21.2.0.jar.sha256
- python-installable-svm-java8-linux-amd64-21.2.0.jar
- python-installable-svm-java8-linux-amd64-21.2.0.jar.sha256

Installation

https://www.graalvm.org/reference-manual/graalvm-updater/#component-installation

e.g., on X64 Laptop:

```
[mydev@myfedora ~]$ tree -L 1 /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0
/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0
├── bin
├── conf
├── GRAALVM-README.md
├── include
├── jmods
├── languages
├── legal
├── lib
├── LICENSE.txt
├── release
├── THIRD_PARTY_LICENSE.txt
└── tools
```

```
[mydev@myfedora ~]$ tree -L 2 /opt/MyWorkSpace/DevSW_Install/Java/JDK/GraalVM/CE/java11-21.2.0
/opt/MyWorkSpace/DevSW_Install/Java/JDK/GraalVM/CE/java11-21.2.0
├── Espresso
│   ├── espresso-installable-java11-linux-amd64-21.2.0.jar
│   └── espresso-installable-java11-linux-amd64-21.2.0.jar.sha256
├── graalvm-ce-java11-linux-amd64-21.2.0.tar.gz
├── graalvm-ce-java11-linux-amd64-21.2.0.tar.gz.sha256
├── llvm-toolchain-installable-java11-linux-amd64-21.2.0.jar
├── llvm-toolchain-installable-java11-linux-amd64-21.2.0.jar.sha256
├── native-image-installable-svm-java11-linux-amd64-21.2.0.jar
├── native-image-installable-svm-java11-linux-amd64-21.2.0.jar.sha256
├── Python
│   ├── graalpython-21.2.0-linux-amd64.tar.gz
│   ├── graalpython-21.2.0-linux-amd64.tar.gz.sha256
│   ├── python-installable-svm-java11-linux-amd64-21.2.0.jar
│   └── python-installable-svm-java11-linux-amd64-21.2.0.jar.sha256
├── wasm-installable-svm-java11-linux-amd64-21.2.0.jar
└── wasm-installable-svm-java11-linux-amd64-21.2.0.jar.sha256
```

```
[mydev@myfedora java11-21.2.0]$ gu list
ComponentId              Version             Component name                Stability                     Origin
------------------------------------------------------------------------------------------------------------------
graalvm                  21.2.0              GraalVM Core                  -
js                       21.2.0              Graal.js                      Supported
[mydev@myfedora java11-21.2.0]$
[mydev@myfedora java11-21.2.0]$
[mydev@myfedora java11-21.2.0]$ gu available
Downloading: Component catalog from www.graalvm.org
ComponentId              Version             Component name                Stability                     Origin
------------------------------------------------------------------------------------------------------------------
espresso                 21.2.0              Java on Truffle               Experimental                  github.com
llvm-toolchain           21.2.0              LLVM.org toolchain            Supported                     github.com
native-image             21.2.0              Native Image                  Early adopter                 github.com
nodejs                   21.2.0              Graal.nodejs                  Supported                     github.com
python                   21.2.0              Graal.Python                  Experimental                  github.com
R                        21.2.0              FastR                         Experimental                  github.com
ruby                     21.2.0              TruffleRuby                   Experimental                  github.com
wasm                     21.2.0              GraalWasm                     Experimental                  github.com
[mydev@myfedora java11-21.2.0]$
```

GOTC

```
[mydev@myfedora ~]$ which gu
/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/bin/gu
[mydev@myfedora ~]$
[mydev@myfedora ~]$ pushd /opt/MyWorkSpace/DevSW_Install/Java/JDK/GraalVM/CE/java11-21.2.0
/opt/MyWorkSpace/DevSW_Install/Java/JDK/GraalVM/CE/java11-21.2.0 ~
[mydev@myfedora java11-21.2.0]$
[mydev@myfedora java11-21.2.0]$ gu -L install  llvm-toolchain-installable-java11-linux-amd64-21.2.0.jar
Processing Component archive: llvm-toolchain-installable-java11-linux-amd64-21.2.0.jar
Installing new component: LLVM.org toolchain (org.graalvm.llvm-toolchain, version 21.2.0)
[mydev@myfedora java11-21.2.0]$
[mydev@myfedora java11-21.2.0]$ gu -L install native-image-installable-svm-java11-linux-amd64-21.2.0.jar
Processing Component archive: native-image-installable-svm-java11-linux-amd64-21.2.0.jar
Installing new component: Native Image (org.graalvm.native-image, version 21.2.0)
[mydev@myfedora java11-21.2.0]$
[mydev@myfedora java11-21.2.0]$ gu -L install wasm-installable-svm-java11-linux-amd64-21.2.0.jar
Processing Component archive: wasm-installable-svm-java11-linux-amd64-21.2.0.jar
Installing new component: GraalWasm (org.graalvm.wasm, version 21.2.0)

IMPORTANT NOTE:
---------------
Set of GraalVM components that provide language implementations have changed. The Polyglot native image and polyglot native C library may be out of sync:
- new languages may not be accessible
- removed languages may cause the native binary to fail on missing resources or libraries.
To rebuild and refresh the native binaries, use the following command:
        /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/bin/gu rebuild-images

[mydev@myfedora java11-21.2.0]$ gu -L install python/python-installable-svm-java11-linux-amd64-21.2.0.jar
Error: Missing file: python/python-installable-svm-java11-linux-amd64-21.2.0.jar
[mydev@myfedora java11-21.2.0]$
[mydev@myfedora java11-21.2.0]$ gu -L install Python/python-installable-svm-java11-linux-amd64-21.2.0.jar
Processing Component archive: Python/python-installable-svm-java11-linux-amd64-21.2.0.jar
Installing new component: Graal.Python (org.graalvm.python, version 21.2.0)

IMPORTANT NOTE:
---------------
Set of GraalVM components that provide language implementations have changed. The Polyglot native image and polyglot native C library may be out of sync:
- new languages may not be accessible
- removed languages may cause the native binary to fail on missing resources or libraries.
To rebuild and refresh the native binaries, use the following command:
        /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/bin/gu rebuild-images

[mydev@myfedora java11-21.2.0]$
```

```
[mydev@myfedora java11-21.2.0]$ which native-image
/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/bin/native-image
[mydev@myfedora java11-21.2.0]$
[mydev@myfedora java11-21.2.0]$ which graalpython
/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/bin/graalpython
[mydev@myfedora java11-21.2.0]$ graalpython -V
Python 3.8.5 (GraalVM CE Native 21.2.0)
[mydev@myfedora java11-21.2.0]$
[mydev@myfedora java11-21.2.0]$ which wasm
/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/bin/wasm
[mydev@myfedora java11-21.2.0]$ wasm --version
WebAssembly (GraalVM CE Native 21.2.0)
[mydev@myfedora java11-21.2.0]$
[mydev@myfedora java11-21.2.0]$ which java
/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/bin/java
[mydev@myfedora java11-21.2.0]$
[mydev@myfedora java11-21.2.0]$ /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/bin/lli --print-toolchain-path
/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/languages/llvm/native/bin
[mydev@myfedora java11-21.2.0]$
```

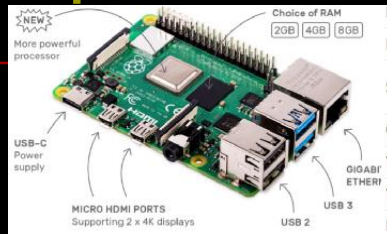
GOTC

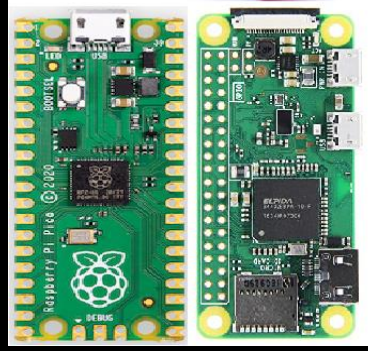
```
[mydev@myfedora /]$ gu list
ComponentId              Version             Component name                Stability                     Origin
------------------------------------------------------------------------------------------------------------------
graalvm                  21.2.0              GraalVM Core                  -
js                       21.2.0              Graal.js                      Supported
llvm-toolchain           21.2.0              LLVM.org toolchain            Supported
native-image             21.2.0              Native Image                  Early adopter
python                   21.2.0              Graal.Python                  Experimental
wasm                     21.2.0              GraalWasm                     Experimental
[mydev@myfedora /]$
```

7) Testbed

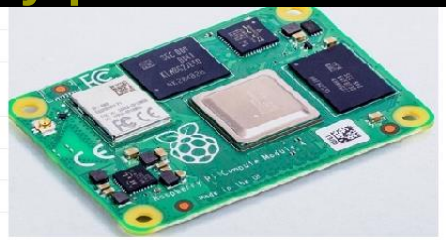
7.1 Raspberry Pi

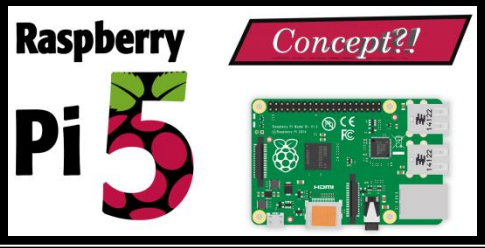
- https://www.raspberrypi.org/products/raspberry-pi-4-model-b/





Features/Specs	Raspberry Pi 4B	Raspberry Pi 3 B+
Release date	24th June 2019	14th March 2018
SoC	Broadcom BCM2711 quad-core Cortex-A72 @ 1.5 GHz	Broadcom BCM2837B0 quad-core Cortex-A53 @ 1.4 GHz
GPU	VideoCore VI with OpenGL ES 1.1, 2.0, 3.0	VideoCore IV with OpenGL ES 1.1, 2.0
Video Decode	H.265 4Kp60, H.264 1080p60	H.264 & MPEG-4 1080p30
Video Encode	H.264 1080p30	
Memory	1GB, 2GB, or 4GB LPDDR4	1GB LPDDR2
Storage	microSD card	
Video & Audio Output	2x micro HDMI ports up to 4Kp60 3.5mm AV port (composite + audio) MIPI DSI connector	1x HDMI 1.4 port up to 1080p60 3.5mm AV port (composite + audio) MIPI DSI connector
Camera	MIPI CSI connector	
Ethernet	Native Gigabit Ethernet	Gigabit Ethernet over USB (300 Mbps max.)
WiFi	Dual band 802.11 b/g/n/ac	
Bluetooth	Bluetooth 5.0 + BLE	Bluetooth 4.2 + BLE
USB	2x USB 3.0 + 2x USB 2.0	4x USB 2.0
Expansion	40-pin GPIO header	
Power Supply	5V via USB type-C up to 3A 5V via GPIO header up to 3A Power over Ethernet via PoE HAT	5V via micro USB up to 2.5A 5V via GPIO header up to 3A Power over Ethernet via PoE HAT
Dimensions	85×56 mm	
Default OS	Raspbian (after June 24, 2019)	Raspbian (after March 2018)
Price	$35 (1GB RAM), $45 (2GB RAM), $55 (4GB RAM)	$35 (1GB RAM)



- 

7.2 X86 Laptop

- on an Dell Laptop with Fedora 34 + Kernel 5.13.4 + 14GB Memory(6GB DDR4 + 8GB Swap)

GOTC

Fedora

- A Linux distribution developed by the community-supported Fedora Project which is sponsored primarily by Red Hat
- https://en.wikipedia.org/wiki/Fedora_(operating_system)

- https://getfedora.org/
- https://alt.fedoraproject.org/alt/
- https://spins.fedoraproject.org/
- https://fedoraproject.org/wiki/Architectures/ARM
- https://fedoramagazine.org/
- https://silverblue.fedoraproject.org/
- Developer friendly!

II. eBPF on GraalVM

1) uBPF

1.1 Overview

- Userspace BPF
- https://stackoverflow.com/questions/65904948/why-is-having-an-userspace-version-of-ebpf-interesting
  applied to Networking & Blockchain…

2) Implementations

Dev Env

- **X64 Laptop:**

```
[mydev@myfedora /]$ uname -a
Linux myfedora 5.13.4-200.fc34.x86_64 #1 SMP Tue Jul 20 20:27:29 UTC 2021 x86_64 x86_64 x86_64 GNU/Linux
[mydev@myfedora /]$
[mydev@myfedora /]$ java --version
openjdk 11.0.12 2021-07-20
OpenJDK Runtime Environment GraalVM CE 21.2.0 (build 11.0.12+6-jvmci-21.2-b08)
OpenJDK 64-Bit Server VM GraalVM CE 21.2.0 (build 11.0.12+6-jvmci-21.2-b08, mixed mode, sharing)
[mydev@myfedora /]$
[mydev@myfedora /]$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/x86_64-redhat-linux/11/lto-wrapper
OFFLOAD_TARGET_NAMES=nvptx-none
OFFLOAD_TARGET_DEFAULT=1
Target: x86_64-redhat-linux
Configured with: ../configure --enable-bootstrap --enable-languages=c,c++,fortran,objc,obj-c++,ada,go,d,lto --prefix=/usr --mandir=/usr/share/man --infodir=/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-shared --enable-threads=posix --enable-checking=release --enable-multilib --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --enable-gnu-unique-object --enable-linker-build-id --with-gcc-major-version-only --with-linker-hash-style=gnu --enable-plugin --enable-initfini-array --with-isl=/builddir/build/BUILD/gcc-11.1.1-20210531/obj-x86_64-redhat-linux/isl-install --enable-offload-targets=nvptx-none --without-cuda-driver --enable-gnu-indirect-function --enable-cet --with-tune=generic --with-arch_32=i686 --build=x86_64-redhat-linux
Thread model: posix
Supported LTO compression algorithms: zlib zstd
gcc version 11.1.1 20210531 (Red Hat 11.1.1-3) (GCC)
[mydev@myfedora /]$
[mydev@myfedora /]$ clang -v
clang version 12.0.0 (Fedora 12.0.0-2.fc34)
Target: x86_64-unknown-linux-gnu
Thread model: posix
InstalledDir: /bin
Found candidate GCC installation: /bin/../lib/gcc/x86_64-redhat-linux/11
Found candidate GCC installation: /usr/lib/gcc/x86_64-redhat-linux/11
Selected GCC installation: /usr/lib/gcc/x86_64-redhat-linux/11
Candidate multilib: .;@m64
Candidate multilib: 32;@m32
Selected multilib: .;@m64
[mydev@myfedora /]$
[mydev@myfedora /]$ mvn -v
Apache Maven 3.6.3 (Red Hat 3.6.3-8)
Maven home: /usr/share/maven
Java version: 11.0.12, vendor: GraalVM Community, runtime: /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "5.13.4-200.fc34.x86_64", arch: "amd64", family: "unix"
[mydev@myfedora /]$
[mydev@myfedora /]$ rust -V
-bash: rust: command not found
[mydev@myfedora /]$ rustc -V
rustc 1.53.0 (Fedora 1.53.0-1.fc34)
[mydev@myfedora /]$
[mydev@myfedora /]$ go version
go version go1.16.5 linux/amd64
[mydev@myfedora /]$
```

- **pip3 install –user wheel nose nose2 coveralls cpp-coveralls pyelftools parcon**

2.1 BPF-Graal-Truffle

GOTC

- https://github.com/mattmurante/bpf-graal-truffle
- a VM for (e)BPF built using the Truffle framework and utilizing Graal JIT compilation

This is a Maven project based off of GraalVM's simple-language that uses the Truffle framework to implement an (e)BPF VM. This repository also includes sample programs that can be compiled to BPF and run on the VM for testing.

To rebuild the project, first ensure the $JAVA_HOME is pointing to the GraalVM Contents/Home directory. Then, in the root directory, execute the command:

```
mvn package
```

to recreate the bpf language jar, the native-image executable, and the bpf component which can be installed into GraalVM. To add the component to GraalVM, execute:

```
gu -L install {path to bpf-component.jar}
```

GraalVM releases can be found at https://github.com/graalvm/graalvm-ce-builds/releases. Graal 20.1.0 for Java 8 was used for this project.

The code for simplelanguage can be found at https://github.com/graalvm/simplelanguage.

Potential Improvements:
- Add (e)BPF call instruction
- Improve use of Truffle framework and interaction with Graal compiler for improved performance

- X64 only

```
[mydev@myfedora Bak]$ tree -L 2 bpf-graal-truffle-master/native/sources/jdk/vm/ci
bpf-graal-truffle-master/native/sources/jdk/vm/ci
├── amd64
│   ├── AMD64.java
│   └── AMD64Kind.java
```

...

Failed to be built with GraalVM CE Java11-21.2.0

```
2021-07-28 00:48:53 [bpfnative:21454]    classlist:   4,705.50 ms,  0.96 GB
2021-07-28 00:48:57 [bpfnative:21454]        (cap):   2,502.03 ms,  0.96 GB
2021-07-28 00:49:00 [bpfnative:21454]        setup:   6,854.94 ms,  0.96 GB
2021-07-28 00:50:28 [bpfnative:21454]     (clinit):     763.88 ms,  2.09 GB
2021-07-28 00:50:28 [bpfnative:21454]   (typeflow):  33,718.01 ms,  2.09 GB
2021-07-28 00:50:28 [bpfnative:21454]    (objects):  46,195.27 ms,  2.09 GB
2021-07-28 00:50:28 [bpfnative:21454]   (features):   3,958.40 ms,  2.09 GB
2021-07-28 00:50:28 [bpfnative:21454]     analysis:  86,239.67 ms,  2.09 GB
2021-07-28 00:50:30 [bpfnative:21454]     universe:   1,853.08 ms,  2.09 GB
2021-07-28 00:50:30 1915 method(s) included for runtime compilation
2021-07-28 00:50:31
2021-07-28 00:50:31 === Found 1 compilation blacklist violations ===
2021-07-28 00:50:31
2021-07-28 00:50:31 Blacklisted method
2021-07-28 00:50:31   org.graalvm.compiler.code.SourceStackTraceBailoutException$1.fillInStackTrace()
2021-07-28 00:50:31 called from
2021-07-28 00:50:31   java.lang.Throwable.<init>(Throwable.java:255) -> java.lang.Exception.<init>(Exception.java:54) -> java.lang.RuntimeException.<init>(RuntimeException.java:51) -> com.oracle.truffle.bpf.nodes.util.NotYetImplemented.<init>(NotYetImplemented.java:6) -> com.oracle.truffle.bpf.nodes.other.ByteswapExpressionNode.byteSwap(ByteswapExpressionNode.java:18)
2021-07-28 00:50:31   com.oracle.truffle.bpf.nodes.other.BE.operation(BE.java:22)
2021-07-28 00:50:31   com.oracle.truffle.bpf.nodes.other.BENodeGen.executeBoolean(BENodeGen.java:38)
2021-07-28 00:50:31   com.oracle.truffle.bpf.nodes.ProgramNode.execute(ProgramNode.java:49)
2021-07-28 00:50:31   org.graalvm.compiler.truffle.runtime.OptimizedCallTarget.executeRootNode(OptimizedCallTarget.java:632)
2021-07-28 00:50:31   org.graalvm.compiler.truffle.runtime.OptimizedCallTarget.profiledPERoot(OptimizedCallTarget.java:603)
2021-07-28 00:50:31
2021-07-28 00:50:31 [bpfnative:21454]      [total]: 102,365.07 ms,  2.09 GB
2021-07-28 00:50:31 # Printing build artifacts to: /opt/MyWorkSpace/MyProjs/Runtime/GraalVM/eBPF/uBPF/BPF-Graal-Truffle/bpf-graal-truffle-master_gcc/native/bpfnative.build_artifacts.txt
2021-07-28 00:50:31 [ERROR] Command execution failed.
2021-07-28 00:50:31 org.apache.commons.exec.ExecuteException: Process exited with an error: 1 (Exit value: 1)
2021-07-28 00:50:31     at org.apache.commons.exec.DefaultExecutor.executeInternal (DefaultExecutor.java:404)
```

...

```
2021-07-28 00:50:31 [INFO] ------------------------------------------------------------------------
2021-07-28 00:50:31 [INFO] Reactor Summary for bpflanguage-parent 20.1.0:
2021-07-28 00:50:31 [INFO]
2021-07-28 00:50:31 [INFO] bpflanguage-parent ............................. SUCCESS [  0.006 s]
2021-07-28 00:50:31 [INFO] bpflanguage .................................... SUCCESS [01:33 min]
2021-07-28 00:50:31 [INFO] launcher ....................................... SUCCESS [  2.089 s]
2021-07-28 00:50:31 [INFO] bpflanguage-graalvm-native ..................... FAILURE [01:46 min]
2021-07-28 00:50:31 [INFO] bpflanguage-graalvm-component .................. SKIPPED
2021-07-28 00:50:31 [INFO] ------------------------------------------------------------------------
2021-07-28 00:50:31 [INFO] BUILD FAILURE
2021-07-28 00:50:31 [INFO] ------------------------------------------------------------------------
2021-07-28 00:50:31 [INFO] Total time:  03:22 min
2021-07-28 00:50:31 [INFO] Finished at: 2021-07-28T00:50:31-07:00
2021-07-28 00:50:31 [INFO] ------------------------------------------------------------------------
2021-07-28 00:50:31 [ERROR] Failed to execute goal org.codehaus.mojo:exec-maven-plugin:3.0.0:exec (make_native) on project bpflanguage-graalvm-native: Command execution failed.: Process exited with an error: 1 (Exit value: 1) -> [Help 1]
2021-07-28 00:50:31 [ERROR]
2021-07-28 00:50:31 [ERROR] To see the full stack trace of the errors, re-run Maven with the -e switch.
2021-07-28 00:50:31 [ERROR] Re-run Maven using the -X switch to enable full debug logging.
2021-07-28 00:50:31 [ERROR]
2021-07-28 00:50:31 [ERROR] For more information about the errors and possible solutions, please read the following articles:
2021-07-28 00:50:31 [ERROR] [Help 1] http://cwiki.apache.org/confluence/display/MAVEN/MojoExecutionException
2021-07-28 00:50:31 [ERROR]
2021-07-28 00:50:31 [ERROR] After correcting the problems, you can resume the build with the command
2021-07-28 00:50:31 [ERROR]   mvn <args> -rf :bpflanguage-graalvm-native
```

- **Even modified the GraalVM version in the configuration files…**
- **Got the similar failure when built with GraalVM CE Java11-20.1.0**

Succesfully built with GraalVM CE Java8-20.1.0

```
2021-07-29 03:43:55 [bpfnative:573102]    classlist:   4,872.47 ms,  1.16 GB
2021-07-29 03:43:59 [bpfnative:573102]        (cap):   2,838.47 ms,  1.58 GB
2021-07-29 03:44:01 [bpfnative:573102]        setup:   6,170.50 ms,  1.58 GB
2021-07-29 03:44:53 [bpfnative:573102]     (clinit):     860.72 ms,  1.78 GB
2021-07-29 03:44:54 [bpfnative:573102]   (typeflow):  28,649.97 ms,  1.78 GB
2021-07-29 03:44:54 [bpfnative:573102]    (objects):  16,728.91 ms,  1.78 GB
2021-07-29 03:44:54 [bpfnative:573102]   (features):   2,404.48 ms,  1.78 GB
2021-07-29 03:44:54 [bpfnative:573102]     analysis:  49,877.24 ms,  1.78 GB
2021-07-29 03:44:55 [bpfnative:573102]     universe:   1,739.80 ms,  1.78 GB
2021-07-29 03:44:55 1355 method(s) included for runtime compilation
2021-07-29 03:45:05 [bpfnative:573102]      (parse):   8,905.54 ms,  1.70 GB
2021-07-29 03:45:12 [bpfnative:573102]     (inline):   6,844.52 ms,  1.78 GB
2021-07-29 03:46:00 [bpfnative:573102]    (compile):  48,052.51 ms,  1.81 GB
2021-07-29 03:46:02 [bpfnative:573102]      compile:  66,525.10 ms,  1.81 GB
2021-07-29 03:46:06 [bpfnative:573102]        image:   4,153.54 ms,  1.84 GB
2021-07-29 03:46:08 [bpfnative:573102]        write:   1,601.17 ms,  1.84 GB
2021-07-29 03:46:08 [bpfnative:573102]      [total]: 137,624.98 ms,  1.84 GB
2021-07-29 03:46:09 [INFO]
2021-07-29 03:46:09 [INFO] --------------< com.oracle:bpflanguage-graalvm-component >--------------
2021-07-29 03:46:09 [INFO] Building bpflanguage-graalvm-component 20.1.0                    [5/5]
2021-07-29 03:46:09 [INFO] --------------------------------[ pom ]---------------------------------
2021-07-29 03:46:09 [INFO]
2021-07-29 03:46:09 [INFO] --- exec-maven-plugin:3.0.0:exec (make_component) @ bpflanguage-graalvm-component ---
2021-07-29 03:46:17 [INFO] ------------------------------------------------------------------------
2021-07-29 03:46:17 [INFO] Reactor Summary for bpflanguage-parent 20.1.0:
2021-07-29 03:46:17 [INFO]
2021-07-29 03:46:17 [INFO] bpflanguage-parent ............................... SUCCESS [  0.003 s]
2021-07-29 03:46:17 [INFO] bpflanguage ...................................... SUCCESS [ 13.998 s]
2021-07-29 03:46:17 [INFO] launcher ......................................... SUCCESS [  1.164 s]
2021-07-29 03:46:17 [INFO] bpflanguage-graalvm-native ....................... SUCCESS [02:22 min]
2021-07-29 03:46:17 [INFO] bpflanguage-graalvm-component .................... SUCCESS [  7.593 s]
2021-07-29 03:46:17 [INFO] ------------------------------------------------------------------------
2021-07-29 03:46:17 [INFO] BUILD SUCCESS
2021-07-29 03:46:17 [INFO] ------------------------------------------------------------------------
2021-07-29 03:46:17 [INFO] Total time:  02:45 min
2021-07-29 03:46:17 [INFO] Finished at: 2021-07-29T03:46:17-07:00
2021-07-29 03:46:17 [INFO] ------------------------------------------------------------------------
```

```
[mydev@myfedora component]$ gu -L install bpf-component.jar
Processing Component archive: bpf-component.jar
Installing new component: BPF Language (com.oracle.truffle.bpf, version 20.1.0)

IMPORTANT NOTE:
---------------
Set of GraalVM components that provide language implementations have changed. The Polyglot native image and polyglot native C library may be out of sync:
- new languages may not be accessible
- removed languages may cause the native binary to fail on missing resources or libraries.
To rebuild and refresh the native binaries, use the following command:
        /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java8-20.1.0/bin/gu rebuild-images

[mydev@myfedora component]$
```

```
[mydev@myfedora /]$ gu list
ComponentId              Version             Component name      Origin
-------------------------------------------------------------------------------
com.oracle.truffle.bpf   20.1.0              BPF Language
graalvm                  20.1.0              GraalVM Core
llvm-toolchain           20.1.0              LLVM.org toolchain
native-image             20.1.0              Native Image
wasm                     20.1.0              GraalWasm
[mydev@myfedora /]$
```

Disable Native(AOT)

- Retried with GraalVM CE Java11-21.2.0 and build pass

```
[mydev@myfedora bpf-graal-truffle-master_gcc]$ git diff
diff --git a/pom.xml b/pom.xml
index 163d459..4670a05 100755
--- a/pom.xml
+++ b/pom.xml
@@ -41,7 +41,6 @@
        <modules>
                <module>language</module>
                <module>launcher</module>
-               <module>native</module>
                <module>component</module>
        </modules>
 </project>
```

```
2021-07-31 08:55:58 [INFO] Building bpflanguage-graalvm-component 20.1.0                    [4/4]
2021-07-31 08:55:58 [INFO] --------------------------------[ pom ]---------------------------------
2021-07-31 08:55:58 [INFO]
2021-07-31 08:55:58 [INFO] --- exec-maven-plugin:3.0.0:exec (make_component) @ bpflanguage-graalvm-component ---
2021-07-31 08:56:06 [INFO] ------------------------------------------------------------------------
2021-07-31 08:56:06 [INFO] Reactor Summary for bpflanguage-parent 20.1.0:
2021-07-31 08:56:06 [INFO]
2021-07-31 08:56:06 [INFO] bpflanguage-parent ............................. SUCCESS [  0.004 s]
2021-07-31 08:56:06 [INFO] bpflanguage .................................... SUCCESS [ 13.504 s]
2021-07-31 08:56:06 [INFO] launcher ....................................... SUCCESS [  0.876 s]
2021-07-31 08:56:06 [INFO] bpflanguage-graalvm-component .................. SUCCESS [  8.465 s]
2021-07-31 08:56:06 [INFO] ------------------------------------------------------------------------
2021-07-31 08:56:06 [INFO] BUILD SUCCESS
2021-07-31 08:56:06 [INFO] ------------------------------------------------------------------------
2021-07-31 08:56:06 [INFO] Total time:  23.038 s
2021-07-31 08:56:06 [INFO] Finished at: 2021-07-31T08:56:06-07:00
2021-07-31 08:56:06 [INFO] ------------------------------------------------------------------------
```

```
[mydev@myfedora bpf-graal-truffle-master_gcc]$ gu -L install component/bpf-component.jar
Processing Component archive: component/bpf-component.jar
Installation of component/bpf-component.jar failed: Component BPF Language requires older GraalVM 20.1.0, the current version is 21.2.0. Component cannot be installed.
Error: Component BPF Language requires older GraalVM 20.1.0, the current version is 21.2.0. Component cannot be installed.
[mydev@myfedora bpf-graal-truffle-master_gcc]$
[mydev@myfedora bpf-graal-truffle-master_gcc]$
```

- Repatched:

```
[mydev@myfedora bpf-graal-truffle-master_gcc]$ git diff
diff --git a/Polyglot Support Tutorial/simplelanguage-master/pom.xml b/Polyglot Support Tutorial/simplelanguage-master/pom.xml
index 6dca779..9e22a4f 100755
--- a/Polyglot Support Tutorial/simplelanguage-master/pom.xml
+++ b/Polyglot Support Tutorial/simplelanguage-master/pom.xml
@@ -48,7 +48,7 @@
     <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
     <m2e.apt.activation>jdt_apt</m2e.apt.activation>
     <!-- If you update this number make sure the VERSION value in ./sl matches -->
-    <graalvm.version>20.1.0</graalvm.version>
+    <graalvm.version>21.2.0</graalvm.version>
   </properties>
   <packaging>pom</packaging>
   <modules>
diff --git a/Polyglot Support Tutorial/simplelanguage-master/sl b/Polyglot Support Tutorial/simplelanguage-master/sl
index 81f7c39..cadf4d9 100755
--- a/Polyglot Support Tutorial/simplelanguage-master/sl
+++ b/Polyglot Support Tutorial/simplelanguage-master/sl
@@ -41,7 +41,7 @@
 #

 # If you update this number make sure the graalvm.version value in ./pom.xml matches
-VERSION="20.1.0"
+VERSION="21.2.0"

 MAIN_CLASS="com.oracle.truffle.sl.launcher.SLMain"
 SCRIPT_HOME="$(cd "$(dirname "$0")" && pwd -P)"
diff --git a/bpf b/bpf
index 90227a5..9199137 100755
--- a/bpf
+++ b/bpf
@@ -41,7 +41,7 @@
 #

 # If you update this number make sure the graalvm.version value in ./pom.xml matches
-VERSION="20.1.0"
+VERSION="21.2.0"

 MAIN_CLASS="com.oracle.truffle.bpf.launcher.VM"
 SCRIPT_HOME="$(cd "$(dirname "$0")" && pwd -P)"
```

```
diff --git a/language/.factorypath b/language/.factorypath
index 3ae1b5e..b74d455 100644
--- a/language/.factorypath
+++ b/language/.factorypath
@@ -1,8 +1,8 @@
 <factorypath>
-    <factorypathentry kind="VARJAR" id="M2_REPO/org/graalvm/truffle/truffle-api/20.1.0/truffle-api-20.1.0.jar" enabled="true" runInBatchMode="false"/>
-    <factorypathentry kind="VARJAR" id="M2_REPO/org/graalvm/sdk/graal-sdk/20.1.0/graal-sdk-20.1.0.jar" enabled="true" runInBatchMode="false"/>
-    <factorypathentry kind="VARJAR" id="M2_REPO/org/graalvm/truffle/truffle-dsl-processor/20.1.0/truffle-dsl-processor-20.1.0.jar" enabled="true" runInBatchMode="false"/>
-    <factorypathentry kind="VARJAR" id="M2_REPO/org/graalvm/truffle/truffle-tck/20.1.0/truffle-tck-20.1.0.jar" enabled="true" runInBatchMode="false"/>
-    <factorypathentry kind="VARJAR" id="M2_REPO/org/graalvm/sdk/polyglot-tck/20.1.0/polyglot-tck-20.1.0.jar" enabled="true" runInBatchMode="false"/>
+    <factorypathentry kind="VARJAR" id="M2_REPO/org/graalvm/truffle/truffle-api/21.2.0/truffle-api-21.2.0.jar" enabled="true" runInBatchMode="false"/>
+    <factorypathentry kind="VARJAR" id="M2_REPO/org/graalvm/sdk/graal-sdk/21.2.0/graal-sdk-21.2.0.jar" enabled="true" runInBatchMode="false"/>
+    <factorypathentry kind="VARJAR" id="M2_REPO/org/graalvm/truffle/truffle-dsl-processor/20.1.0/truffle-dsl-processor-21.2.0.jar" enabled="true" runInBatchMode="false"/>
+    <factorypathentry kind="VARJAR" id="M2_REPO/org/graalvm/truffle/truffle-tck/21.2.0/truffle-tck-21.2.0.jar" enabled="true" runInBatchMode="false"/>
+    <factorypathentry kind="VARJAR" id="M2_REPO/org/graalvm/sdk/polyglot-tck/21.2.0/polyglot-tck-21.2.0.jar" enabled="true" runInBatchMode="false"/>
     <factorypathentry kind="VARJAR" id="M2_REPO/org/antlr/antlr4-runtime/4.7/antlr4-runtime-4.7.jar" enabled="true" runInBatchMode="false"/>
 </factorypath>
diff --git a/pom.xml b/pom.xml
index 163d459..ed18846 100755
--- a/pom.xml
+++ b/pom.xml
@@ -35,13 +35,12 @@
                 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
                 <m2e.apt.activation>jdt_apt</m2e.apt.activation>
                 <!-- If you update this number make sure the VERSION value in ./sl matches -->
-                <graalvm.version>20.1.0</graalvm.version>
+                <graalvm.version>21.2.0</graalvm.version>
         </properties>
         <packaging>pom</packaging>
         <modules>
                 <module>language</module>
                 <module>launcher</module>
-                <module>native</module>
                 <module>component</module>
         </modules>
 </project>
[mydev@myfedora bpf-graal-truffle-master_gcc]$
```

Rebuild:

```
2021-07-31 09:25:25 [INFO] --------------< com.oracle:bpflanguage-graalvm-component >--------------
2021-07-31 09:25:25 [INFO] Building bpflanguage-graalvm-component 21.2.0                    [4/4]
2021-07-31 09:25:25 [INFO] --------------------------------[ pom ]---------------------------------
2021-07-31 09:25:25 [INFO]
2021-07-31 09:25:25 [INFO] --- exec-maven-plugin:3.0.0:exec (make_component) @ bpflanguage-graalvm-component ---
2021-07-31 09:25:34 [INFO] ------------------------------------------------------------------------
2021-07-31 09:25:34 [INFO] Reactor Summary for bpflanguage-parent 21.2.0:
2021-07-31 09:25:34 [INFO]
2021-07-31 09:25:34 [INFO] bpflanguage-parent ............................... SUCCESS [  0.005 s]
2021-07-31 09:25:34 [INFO] bpflanguage ...................................... SUCCESS [  9.329 s]
2021-07-31 09:25:34 [INFO] launcher ......................................... SUCCESS [  1.005 s]
2021-07-31 09:25:34 [INFO] bpflanguage-graalvm-component .................... SUCCESS [  8.670 s]
2021-07-31 09:25:34 [INFO] ------------------------------------------------------------------------
2021-07-31 09:25:34 [INFO] BUILD SUCCESS
2021-07-31 09:25:34 [INFO] ------------------------------------------------------------------------
2021-07-31 09:25:34 [INFO] Total time:  19.207 s
2021-07-31 09:25:34 [INFO] Finished at: 2021-07-31T09:25:34-07:00
2021-07-31 09:25:34 [INFO] ------------------------------------------------------------------------
```

Successfully install bpf-component.jar to GraalVM CE Java11-21.2.0

```
[mydev@myfedora bpf-graal-truffle-master_gcc]$ gu -L install component/bpf-component.jar
Processing Component archive: component/bpf-component.jar
Installing new component: BPF Language (com.oracle.truffle.bpf, version 21.2.0)

IMPORTANT NOTE:
---------------
Set of GraalVM components that provide language implementations have changed. The Polyglot native image and polyglot native C library may be out of sync:
- new languages may not be accessible
- removed languages may cause the native binary to fail on missing resources or libraries.
To rebuild and refresh the native binaries, use the following command:
       /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/bin/gu rebuild-images

[mydev@myfedora bpf-graal-truffle-master_gcc]$ gu list
ComponentId              Version             Component name                Stability                     Origin
-------------------------------------------------------------------------------------------------------------------
com.oracle.truffle.bpf   21.2.0              BPF Language                  -
graalvm                  21.2.0              GraalVM Core                  -
js                       21.2.0              Graal.js                      Supported
llvm-toolchain           21.2.0              LLVM.org toolchain            Supported
native-image             21.2.0              Native Image                  Early adopter
python                   21.2.0              Graal.Python                  Experimental
wasm                     21.2.0              GraalWasm                     Experimental
[mydev@myfedora bpf-graal-truffle-master_gcc]$
```

GOTC

Failed to start:

```
[mydev@myfedora /]$ which bpf
/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/bin/bpf
[mydev@myfedora /]$ bpf
Error parsing Graal options: Could not find option Vectorization
Error: A fatal exception has occurred. Program will exit.
[mydev@myfedora /]$
```

Patched again:

```
[mydev@myfedora bpf-graal-truffle-master_gcc]$ git diff bpf
diff --git a/bpf b/bpf
index 90227a5..d1f5f62 100755
--- a/bpf
+++ b/bpf
@@ -41,7 +41,7 @@
 #

 # If you update this number make sure the graalvm.version value in ./pom.xml matches
-VERSION="20.1.0"
+VERSION="21.2.0"

 MAIN_CLASS="com.oracle.truffle.bpf.launcher.VM"
 SCRIPT_HOME="$(cd "$(dirname "$0")" && pwd -P)"
@@ -109,7 +109,7 @@ if [[ "$GRAALVM_VERSION" != "" ]]; then
         ###ADD ON FOR PROFILING
 #       JAVA_ARGS+=("-XX:+PreserveFramePointer")
         ###ADD ON FOR VECTORIZATION
-        JAVA_ARGS+=("-Dgraal.Vectorization=true")
+        #JAVA_ARGS+=("-Dgraal.Vectorization=true")
         ###ADD ON FOR EXAMINING GRAAL'S COMPILATION ACTIONS
 #       JAVA_ARGS+=("-Dgraal.PrintCompilation=true" "-Dgraal.LogFile=/Users/mattmurante/Desktop/GraalOut.txt")
         ###ADD ON FOR EXAMINING COMPILED CODE
[mydev@myfedora bpf-graal-truffle-master_gcc]$
```

Succesfully run the “calculate the 10000th prime” test case:

```
[mydev@myfedora /]$ bpf /opt/MyWorkSpace/MyProjs/Runtime/GraalVM/eBPF/uBPF/BPF-Graal-Truffle/bpf-graal-truffle-master_gcc/samples/primes_10000.bpf
104729
```

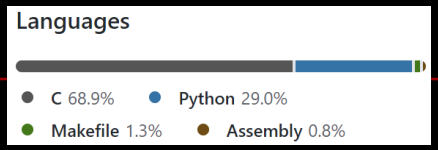
Not passed all the test cases at:

https://github.com/mattmurante/bpf-graal-truffle/tree/master/samples

Under investigation…

2.2 uBPF from IOVisor

- Userspace eBPF VM
- https://github.com/iovisor/ubpf




```
[mydev@myfedora UBPF-IOVisor]$ tree -L 2 ubpf
ubpf
├── bin
│   ├── ubpf-assembler
│   └── ubpf-disassembler
├── LICENSE-APACHE
├── README.md
├── requirements.txt
├── test_framework
│   ├── expand-testcase.py
│   ├── test_assembler.py
│   ├── testdata.py
│   ├── test_disassembler.py
│   ├── test_elf.py
│   ├── test_jit.py
│   ├── test_roundtrip.py
│   └── test_vm.py
└── tests
    ├── add.data
```

...

```
├── ubpf
│   ├── asm_parser.py
│   ├── assembler.py
│   ├── disassembler.py
│   └── __init__.py
└── vm
    ├── ebpf.h
    ├── inc
    ├── Makefile
    ├── test.c
    ├── ubpf_int.h
    ├── ubpf_jit_x86_64.c
    ├── ubpf_jit_x86_64.h
    ├── ubpf_loader.c
    └── ubpf_vm.c
```

- X64 only

Build

- make –C vm
  https://github.com/iovisor/ubpf/blob/master/vm/Makefile

```
15  CFLAGS := -Wall -Werror -Iinc -O2 -g -Wunused-parameter
16  LDLIBS := -lm
17
18  INSTALL ?= install
19  DESTDIR =
20  PREFIX ?= /usr/local
21
22  ifeq ($(COVERAGE),1)
23  CFLAGS += -fprofile-arcs -ftest-coverage
24  LDFLAGS += -fprofile-arcs
25  endif
26
27  ifeq ($(ASAN),1)
28  CFLAGS += -fsanitize=address
29  LDFLAGS += -fsanitize=address
30  endif
31
32  all: libubpf.a test
33
34  ubpf_jit_x86_64.o: ubpf_jit_x86_64.c ubpf_jit_x86_64.h
35
36  libubpf.a: ubpf_vm.o ubpf_jit_x86_64.o ubpf_loader.o
37          ar rc $@ $^
38
39  test: test.o libubpf.a
40
41  install:
42          $(INSTALL) -d $(DESTDIR)$(PREFIX)/lib
43          $(INSTALL) -m 644 libubpf.a $(DESTDIR)$(PREFIX)/lib
44          $(INSTALL) -d $(DESTDIR)$(PREFIX)/include
45          $(INSTALL) -m 644 inc/ubpf.h $(DESTDIR)$(PREFIX)/include
46
47  clean:
48          rm -f test libubpf.a *.o
```

Samples

- https://klyr.github.io/posts/playing_with_ubpf/

```
[mydev@myfedora Test1]$ ccat hello.c
static int idouble(int a) {
        return (a * 2);
}

int bpf_prog(void *ctx) {
        int a = 1;
        a = idouble(a);

        return (a);
}
[mydev@myfedora Test1]$
[mydev@myfedora Test1]$ clang -O2 -target bpf -c hello.c -o hello.o
[mydev@myfedora Test1]$ file hello.o
hello.o: ELF 64-bit LSB relocatable, eBPF, version 1 (SYSV), not stripped
[mydev@myfedora Test1]$
[mydev@myfedora Test1]$ llvm-objdump --disassemble hello.o

hello.o:        file format elf64-bpf


Disassembly of section .text:

0000000000000000 <bpf_prog>:
       0:       b7 00 00 00 02 00 00 00 r0 = 2
       1:       95 00 00 00 00 00 00 00 exit
[mydev@myfedora Test1]$
[mydev@myfedora Test1]$ /opt/MyWorkSpace/MyProjs/Runtime/GraalVM/eBPF/uBPF/UBPF-IOVisor/ubpf-master_gcc/vm/test hello.o
0x2
[mydev@myfedora Test1]$
```

```
[mydev@myfedora Test1]$ ccat ./hello2.c
#include <stdint.h>

static uint32_t idouble(uint32_t a) {
        return (a * 2);
}

uint32_t bpf_prog(int32_t *arg) {
        uint32_t result = 0;
        result = idouble(*arg);

        return result;
}
[mydev@myfedora Test1]$
[mydev@myfedora Test1]$ clang -O2 -target bpf -c hello2.c -o hello2.o
[mydev@myfedora Test1]$ printf "%b" '\x03\x00\x00\x00' > integer_3.mem
[mydev@myfedora Test1]$
[mydev@myfedora Test1]$ /opt/MyWorkSpace/MyProjs/Runtime/GraalVM/eBPF/uBPF/UBPF-IOVisor/ubpf-master_gcc/vm/test hello2.o
uBPF error: out of bounds memory load at PC 0, addr (nil), size 4
mem (nil)/0 stack 0x7fffeb29a4b0/512
0xffffffffffffffff
[mydev@myfedora Test1]$ /opt/MyWorkSpace/MyProjs/Runtime/GraalVM/eBPF/uBPF/UBPF-IOVisor/ubpf-master_gcc/vm/test hello2.o --mem integer_3.mem
0x6
[mydev@myfedora Test1]$
[mydev@myfedora Test1]$ llvm-objdump --disassemble hello2.o

hello2.o:       file format elf64-bpf


Disassembly of section .text:

0000000000000000 <bpf_prog>:
       0:       61 10 00 00 00 00 00 00 r0 = *(u32 *)(r1 + 0)
       1:       67 00 00 00 01 00 00 00 r0 <<= 1
       2:       95 00 00 00 00 00 00 00 exit
[mydev@myfedora Test1]$
```

Run the uBPF IOVisor run on GraalVM

- Method: convert uBPF(IOVisor) to LLVM bitcode
- installed GraalVM LLVM toolchain

```
[mydev@myfedora /]$ lli --print-toolchain-path
/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/languages/llvm/native/bin
[mydev@myfedora /]$
[mydev@myfedora /]$ tree -L 1 /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/languages/llvm/native/bin
/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/languages/llvm/native/bin
├── ar -> graalvm-native-binutil
├── c++ -> graalvm-native-clang++
├── cc -> graalvm-native-clang
├── clang -> graalvm-native-clang
├── clang++ -> graalvm-native-clang++
├── g++ -> graalvm-native-clang++
├── gcc -> graalvm-native-clang
├── graalvm-clang -> graalvm-native-clang
├── graalvm-clang++ -> graalvm-native-clang++
├── graalvm-native-binutil
├── graalvm-native-clang
├── graalvm-native-clang++
├── graalvm-native-ld
├── ld -> graalvm-native-ld
├── ld64 -> graalvm-native-ld
├── ld.lld -> graalvm-native-ld
├── lld -> graalvm-native-ld
├── llvm-ar -> graalvm-native-binutil
├── llvm-nm -> graalvm-native-binutil
├── llvm-objcopy -> graalvm-native-binutil
├── llvm-objdump -> graalvm-native-binutil
├── llvm-ranlib -> graalvm-native-binutil
├── llvm-readelf -> graalvm-native-binutil
├── llvm-readobj -> graalvm-native-binutil
├── llvm-strip -> graalvm-native-binutil
├── nm -> graalvm-native-binutil
├── objcopy -> graalvm-native-binutil
├── objdump -> graalvm-native-binutil
├── ranlib -> graalvm-native-binutil
├── readelf -> graalvm-native-binutil
├── readobj -> graalvm-native-binutil
└── strip -> graalvm-native-binutil
```

```
[mydev@myfedora /]$ /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/languages/llvm/native/bin/clang -v
GraalVM wrapper script for clang
GraalVM version: 21.2.0
running: /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/lib/llvm/bin/clang -v
clang version 10.0.0 (GraalVM.org llvmorg-10.0.0-6-gad4288df64-bg263fb7a415 ad4288df6481f1720dc2b90495fa3cba4296f904)
Target: x86_64-unknown-linux-gnu
Thread model: posix
InstalledDir: /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/lib/llvm/bin
Found candidate GCC installation: /usr/lib/gcc/x86_64-redhat-linux/11
Selected GCC installation: /usr/lib/gcc/x86_64-redhat-linux/11
Candidate multilib: .;@m64
Candidate multilib: 32;@m32
Selected multilib: .;@m64
[mydev@myfedora /]$
```

- # Initial build result:

```
export GRAALVM_LLVM_TOOLCHAIN=$(lli --print-toolchain-path)
export PATH=$GRAALVM_LLVM_TOOLCHAIN:$PATH
```

```
[mydev@myfedora vm]$ file ./libubpf.a
./libubpf.a: current ar archive
[mydev@myfedora vm]$
[mydev@myfedora vm]$ file ./test.o
./test.o: LLVM IR bitcode
[mydev@myfedora vm]$ file ./test
./test: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, with debug_info, not stripped
[mydev@myfedora vm]$
```

- # Something wrong with the linker in current GraalVM LLVM toolchain?
- # My workaround(patched Makefile):

```
[mydev@myfedora ubpf-master_graalvm2]$ git diff
diff --git a/vm/Makefile b/vm/Makefile
index ee45188..5bb8e86 100644
--- a/vm/Makefile
+++ b/vm/Makefile
@@ -17,7 +17,7 @@ LDLIBS := -lm

 INSTALL ?= install
 DESTDIR =
-PREFIX ?= /usr/local
+PREFIX ?= /usr/local/ubpf/iovisor

 ifeq ($(COVERAGE),1)
 CFLAGS += -fprofile-arcs -ftest-coverage
@@ -29,20 +29,23 @@ CFLAGS += -fsanitize=address
 LDFLAGS += -fsanitize=address
 endif

-all: libubpf.a test
+all: ubpf-iovisor-graalvm

 ubpf_jit_x86_64.o: ubpf_jit_x86_64.c ubpf_jit_x86_64.h

-libubpf.a: ubpf_vm.o ubpf_jit_x86_64.o ubpf_loader.o
-       ar rc $@ $^
+ubpf_vm.o: ubpf_vm.c

-test: test.o libubpf.a
+ubpf_loader.o: ubpf_loader.c
+
+test.o: test.c
+
+ubpf-iovisor-graalvm: test.o ubpf_vm.o ubpf_jit_x86_64.o ubpf_loader.o
+       llvm-link test.o ubpf_vm.o ubpf_jit_x86_64.o ubpf_loader.o -o ubpf-iovisor-graalvm

 install:
-       $(INSTALL) -d $(DESTDIR)$(PREFIX)/lib
-       $(INSTALL) -m 644 libubpf.a $(DESTDIR)$(PREFIX)/lib
+       $(INSTALL) -d $(DESTDIR)$(PREFIX)/bin
        $(INSTALL) -d $(DESTDIR)$(PREFIX)/include
        $(INSTALL) -m 644 inc/ubpf.h $(DESTDIR)$(PREFIX)/include

 clean:
-       rm -f test libubpf.a *.o
+       rm -f ubpf-iovisor-graalvm test *.o
```

Rebuild and run the previous samples:

```
[mydev@myfedora ubpf-master_graalvm2]$ tree -L 1 vm
vm
├── ebpf.h
├── inc
├── Makefile
├── test.c
├── test.o
├── ubpf_int.h
├── ubpf-iovisor-graalvm
├── ubpf_jit_x86_64.c
├── ubpf_jit_x86_64.h
├── ubpf_jit_x86_64.o
├── ubpf_loader.c
├── ubpf_loader.o
├── ubpf_vm.c
└── ubpf_vm.o
```

```
[mydev@myfedora vm]$ file ./test.o
./test.o: LLVM IR bitcode
[mydev@myfedora vm]$
[mydev@myfedora vm]$ file ./ubpf-iovisor-graalvm
./ubpf-iovisor-graalvm: LLVM IR bitcode
```

```
[mydev@myfedora vm]$ /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/bin/lli ./ubpf-iovisor-graalvm
usage: /opt/MyWorkSpace/MyProjs/Runtime/GraalVM/eBPF/uBPF/UBPF-IOVisor/ubpf-master_graalvm2/vm/ubpf-iovisor-graalvm [-h] [-j|--jit] [-m|--mem PATH] BINARY

Executes the eBPF code in BINARY and prints the result to stdout.
If --mem is given then the specified file will be read and a pointer
to its data passed in r1.
If --jit is given then the JIT compiler will be used.

Other options:
  -r, --register-offset NUM: Change the mapping from eBPF to x86 registers
[mydev@myfedora vm]$
```

```
[mydev@myfedora vm]$ /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/languages/llvm/bin/lli ./ubpf-iovisor-graalvm /opt/MyWorkSpace/MyTest/eBPF/uBPF/Test1/hello.o
0x2
[mydev@myfedora vm]$
[mydev@myfedora vm]$ /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/languages/llvm/bin/lli ./ubpf-iovisor-graalvm /opt/MyWorkSpace/MyTest/eBPF/uBPF/Test1/hello2.o --mem /opt/MyWorkSpace/MyTest/eBPF/uBPF/Test1/integer_3.mem
0x6
[mydev@myfedora vm]$
```

Successfully run the samples with the built uBPF(IOVisor) with the GraalVM!

https://github.com/oracle/graal/blob/master/docs/reference-manual/llvm/README.md

2.3 Summary

Two ways to run uBPF with GraalVM:

GraalVM-native implementation via Truffle/Graal

- working on porting project **bpf-graal-truffle** to the latest GraalVM CE for **AArch64** with the re-implemented eBPF call instruction and so on
- keep the pace with the latest Linux Kernel 5.13+

Convert uBPF VM to LLVM bitcode

- working on a new version of GraalVM LLVM toolchain
- try to make **Rust**-based uBPF implementation run on GraalVM

III. WASM on GraalVM

GOTC

Dev Env(on X64 Laptop)

- https://emscripten.org/docs/getting_started/downloads.html

```
/opt/MyWorkSpace/MyProjs/Toolchain/WASM/EMSDK/emsdk-main/upstream/emscripten
├── AUTHORS
├── cache
├── ChangeLog.md
├── cmake
├── CONTRIBUTING.md
├── docs
├── em++
├── emar
├── emar.bat
├── emar.py
├── em++.bat
├── embuilder
├── embuilder.bat
├── embuilder.py
├── emcc
├── emcc.bat
├── emcc.py
├── emcmake
├── emcmake.bat
├── emcmake.py
├── em-config
├── em-config.bat
├── em-config.py
├── emconfigure
├── emconfigure.bat
├── emconfigure.py
├── emdump
├── emdump.bat
├── em-dwp
├── emdwp
├── em-dwp.bat
├── emdwp.bat
├── emmake
├── emmake.bat
├── emmake.py
├── emnm
├── emnm.bat
├── emprofile
├── emprofile.bat
├── em++.py
├── emranlib
├── emranlib.bat
├── emranlib.py
├── emrun
├── emrun.bat
├── emrun.py
├── emscons
├── emscons.bat
├── emscons.py
├── emscripten.py
├── emscripten-revision.txt
├── emscripten-version.txt
├── emsize
├── emsize.bat
├── emsize.py
├── LICENSE
├── media
├── node_modules
├── package.json
├── package-lock.json
├── __pycache__
├── README.md
├── requirements-dev.txt
├── src
├── system
├── tests
├── third_party
└── tools
```

```
/opt/MyWorkSpace/MyProjs/Toolchain/WASM/EMSDK/emsdk-main/upstream/bin
├── clang -> clang-13
├── clang++ -> clang
├── clang-13
├── llc
├── lld
├── llvm-addr2line -> llvm-symbolizer
├── llvm-ar
├── llvm-cxxfilt
├── llvm-dwarfdump
├── llvm-dwp
├── llvm-nm
├── llvm-objcopy
├── llvm-objdump
├── llvm-ranlib -> llvm-ar
├── llvm-readobj
├── llvm-size
├── llvm-strings
├── llvm-symbolizer
├── opt
├── wasm2js
├── wasm32-clang -> clang
├── wasm32-clang++ -> clang++
├── wasm32-wasi-clang -> clang
├── wasm32-wasi-clang++ -> clang++
├── wasm-as
├── wasm-ctor-eval
├── wasm-dis
├── wasm-emscripten-finalize
├── wasm-ld -> lld
├── wasm-metadce
├── wasm-opt
├── wasm-reduce
├── wasm-shell
└── wasm-split
```

- sudo dnf install wabt

- **mx**

https://github.com/graalvm/mx

Command-line tool used for the development of Graal projects.

```
[mydev@myfedora /]$ mx --help
usage: mx [-h] [-v] [-V] [--no-warning] [--quiet] [-y] [-n] [-p <path>] [--dbg <address>] [-d] [--attach <address>]
          [--backup-modified] [--exec-log <path>] [--cp-pfx <arg>] [--cp-sfx <arg>] [-J <arg> | --J @<args>]
          [-P <arg> | --Jp @<args>] [-A <arg> | --Ja @<args>] [--user-home <path>] [--java-home <path>]
          [--jacoco {off,on,append}] [--jacoco-whitelist-package <package>] [--jacoco-exclude-annotation <annotation>]
          [--jacoco-dest-file <path>] [--extra-java-homes <path>] [--strict-compliance] [--ignore-project <name>]
          [--kill-with-sigquit] [--suite <name>] [--suitemodel <arg>] [--primary] [--dynamicimports <name>]
          [--no-download-progress] [--version] [--mx-tests] [--jdk <tag:compliance>] [--jmods-dir <path>]
          [--version-conflict-resolution {suite,none,latest,latest_all,ignore}] [-c <cpus>] [--proguard-cp PROGUARD_CP]
          [--strip-jars] [--env <name>] [--trust-http] [--multiarch] [--dump-task-stats <path>] [--compdb <file>]
          [--timeout <secs>] [--ptimeout <secs>] [--vm VM] [--vmbuild VMBUILD]
          ...
```

for GraalVM development, pls refer to:

https://github.com/oracle/graal/tree/master/vm

```
[mydev@myfedora graal-master]$ find . -name "suite.py"
./vm/mx.vm/suite.py
./java-benchmarks/mx.java-benchmarks/suite.py
./sulong/mx.sulong/suite.py
./regex/mx.regex/suite.py
./wasm/mx.wasm/suite.py
./examples/mx.examples/suite.py
./substratevm/mx.substratevm/suite.py
./tools/mx.tools/suite.py
./compiler/mx.compiler/suite.py
./espresso/mx.espresso/suite.py
./truffle/mx.truffle/suite.py
./sdk/mx.sdk/suite.py
[mydev@myfedora graal-master]$
```

1) GraalVM-native Implementation

1.1 GraalWasm

- The official solution
- https://github.com/oracle/graal/tree/master/wasm

GraalWasm is a WebAssembly engine implemented in the GraalVM. It can interpret and compile WebAssembly programs in the binary format, or be embedded into other programs.

We are working hard towards making GraalWasm more stable and more efficient, as well as to implement various WebAssembly extensions. Feedback, bug reports, and open-source contributions are welcome!

-

```
[mydev@myfedora java11-21.2.0]$ find . -name "*.jar" |grep -i wasm
./lib/graalvm/wasm-launcher.jar
./languages/wasm/wasm.jar
```

Samples

- https://www.graalvm.org/reference-manual/wasm/

floyd.c:

```c
#include <stdio.h>

int main() {
  int number = 1;
  int rows = 10;
  for (int i = 1; i <= rows; i++) {
    for (int j = 1; j <= i; j++) {
      printf("%d ", number);
      ++number;
    }
    printf(".\n");
  }
  return 0;
}
```

```
[mydev@myfedora Test1]$ emcc -o floyd.wasm floyd.c
shared:INFO: (Emscripten: Running sanity checks)
[mydev@myfedora Test1]$
[mydev@myfedora Test1]$ which wasm
/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0/bin/wasm
[mydev@myfedora Test1]$
[mydev@myfedora Test1]$ wasm --Builtins=wasi_snapshot_preview1 floyd.wasm
1 .
2 3 .
4 5 6 .
7 8 9 10 .
11 12 13 14 15 .
16 17 18 19 20 21 .
22 23 24 25 26 27 28 .
29 30 31 32 33 34 35 36 .
37 38 39 40 41 42 43 44 45 .
46 47 48 49 50 51 52 53 54 55 .
[mydev@myfedora Test1]$
```

GOTC

Embedding WebAssembly Program

https://www.graalvm.org/reference-manual/wasm/
https://github.com/oracle/graal/tree/master/wasm
https://medium.com/graalvm/announcing-graalwasm-a-webassembly-engine-in-graalvm-25cd0400a7f2

prepare WasmPolyglotTest1.java(patched for above sample code):

```
[mydev@myfedora Polyglot]$ colordiff WasmPolyglotTest1.java.orgi WasmPolyglotTest1.java
5a6,7
> import java.io.File;
> import java.nio.file.Files;
9a12,13
>         File file = new File("main.wasm");
>         byte[] binary = Files.readAllBytes(file.toPath());
[mydev@myfedora Polyglot]$
```

```java
import org.junit.Test;
import static org.junit.Assert.*;
import org.graalvm.polyglot.*;
import org.graalvm.polyglot.io.ByteSequence;
import java.io.IOException;
import java.io.File;
import java.nio.file.Files;

public class WasmPolyglotTest1 {
    @Test
    public void test() throws IOException {
        File file = new File("main.wasm");
        byte[] binary = Files.readAllBytes(file.toPath());
        Context.Builder contextBuilder = Context.newBuilder("wasm");
        Source.Builder sourceBuilder = Source.newBuilder("wasm",
                                                         ByteSequence.create(binary),
                                                         "main");

        Source source = sourceBuilder.build();
        Context context = contextBuilder.build();
        context.eval(source);
        Value mainFunction = context.getBindings("wasm").getMember("main");
        Value result = mainFunction.execute();
        assertEquals(42, result.asInt());
    }
}
```

GOTC

prepare main.c and main.wasm:

```
[mydev@myfedora Polyglot]$ ccat main.c
#define WASM_EXPORT __attribute__((visibility("default")))

WASM_EXPORT
int main() {
  return 42;
}
[mydev@myfedora Polyglot]$
[mydev@myfedora Polyglot]$ emcc -o main.wasm main.c
```

put main.wasm and WasmPolyglotTest1.java to $GRAALVM_SRC/wasm

```
[mydev@myfedora wasm]$ git status
On branch master
Your branch is up to date with 'origin/master'.

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        src/org.graalvm.wasm.test/src/org/graalvm/wasm/test/WasmPolyglotTest1.java
        src/org.graalvm.wasm.test/src/org/graalvm/wasm/test/main.wasm
```

at $GRAALVM_SRC/wasm, run “mx --dy /truffle,/compiler build”
build pass! (GraalVM Java11-21.2.0)
then run “mx --dy /compiler,/truffle --jdk jvmci unittest WasmPolyglotTest1”

```
[mydev@myfedora wasm]$ mx --dy /compiler,/truffle --jdk jvmci unittest WasmPolyglotTest1
Updating/creating /opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Official/graal-master/compiler/mxbuild/linux-amd64/graaljdks/jdk11-cmp from /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java11-21.2.0 using
intermediate directory /opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Official/graal-master/compiler/mxbuild/linux-amd64/graaljdks/tmppJM4qi/jdk11-cmp since /opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Offici
al/graal-master/compiler/mxbuild/linux-amd64/graaljdks/jdk11-cmp does not exist
Traceback (most recent call last):
  File "/opt/MyWorkSpace/DevSW/Tools/Build/MX/mx.py", line 17707, in <module>
    main()
  File "/opt/MyWorkSpace/DevSW/Tools/Build/MX/mx.py", line 17670, in main
    d.post_init()
  File "/opt/MyWorkSpace/DevSW/Tools/Build/MX/mx_jardistribution.py", line 137, in post_init
    self._path = mx._make_absolute(self._default_path(), self.suite.dir)
  File "/opt/MyWorkSpace/DevSW/Tools/Build/MX/mx.py", line 5375, in _default_path
    self._extra_artifact_discriminant(), self.default_filename())
  File "/opt/MyWorkSpace/DevSW/Tools/Build/MX/mx_jardistribution.py", line 154, in _extra_artifact_discriminant
    compliance = self._compliance_for_build()
  File "/opt/MyWorkSpace/DevSW/Tools/Build/MX/mx_jardistribution.py", line 164, in _compliance_for_build
    jdk9 = mx.get_jdk('9+', cancel='No module-info will be generated for modular JAR distributions')
  File "/opt/MyWorkSpace/DevSW/Tools/Build/MX/mx.py", line 12473, in get_jdk
    jdk = factory.getJDKConfig()
  File "/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Official/graal-master/compiler/mx.compiler/mx_compiler.py", line 1019, in getJDKConfig
    return GraalJVMCIJDKConfig()
  File "/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Official/graal-master/compiler/mx.compiler/mx_compiler.py", line 1004, in __init__
    mx.JDKConfig.__init__(self, jdk.home, tag=_JVMCI_JDK_TAG)
  File "/opt/MyWorkSpace/DevSW/Tools/Build/MX/mx.py", line 13400, in __init__
    self.jar = self.exe_path('jar')
  File "/opt/MyWorkSpace/DevSW/Tools/Build/MX/mx.py", line 13490, in exe_path
    return exe_suffix(join(self.home, sub_dir, name))
  File "/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Official/graal-master/compiler/mx.compiler/mx_compiler.py", line 1011, in home
    return get_graaljdk().home
  File "/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Official/graal-master/compiler/mx.compiler/mx_compiler.py", line 901, in get_graaljdk
    graaljdk_dir, _ = _update_graaljdk(jdk)
  File "/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Official/graal-master/compiler/mx.compiler/mx_compiler.py", line 1467, in _update_graaljdk
    fp.write(unstrip_map)
...
```

try to run other tests and even without our new test case also got the same failure.

and build failed when switching to GraalVM Java8-21.2.0.

should be a JDK compliance issue, something wrong with the mx or my dev env? Still working on dig out the root cause…

…

- https://www.graalvm.org/reference-manual/polyglot-programming/

  not mature yet…

2) Sulong-based Implementation

2.1 XX

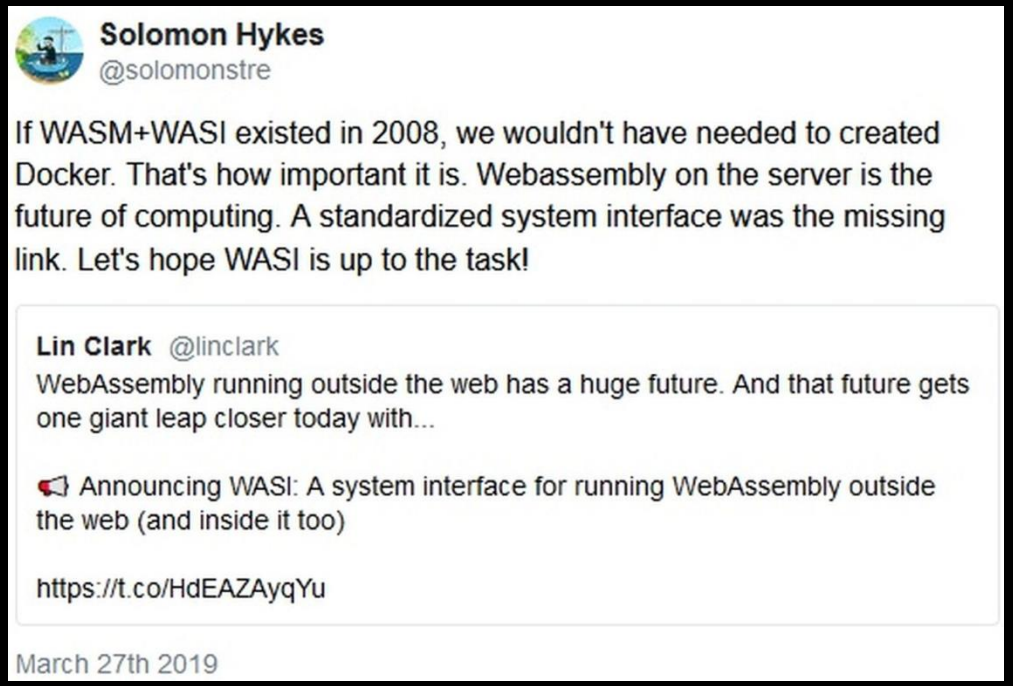
- Working on rebuild some awesome WASM implementations with GraalVM LLVM toolchain

GOTC

IV. Unified eBPF & WASM runtime for Cloud Native

1) Rethinking for Cloud Native

1.1 Replace docker with WASM?

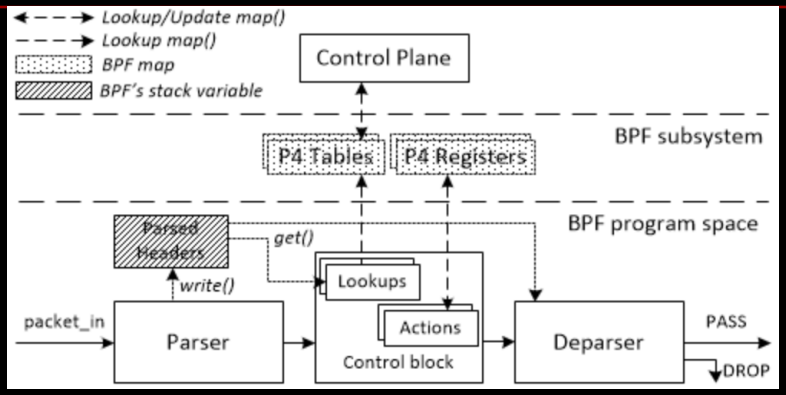
- 

- https://krustlet.dev/
  run WASM workloads in your Kubernetes cluster
- ...

1.2 DSLs

P4

- https://opennetworking.org/news-and-events/blog/p4c-ubpf-a-new-back-end-for-the-p4-compiler/



- https://github.com/p4lang/p4c/tree/main/backends/ubpf

The **p4c-ubpf** compiler allows to translate P4 programs into the uBPF programs. We use the uBPF implementation provided by the P4rt-OVS switch. The uBPF VM is based on the open-source implementation provided by IOVisor.

The P4-to-uBPF compiler accepts only the P4_16 programs written for the ubpf_model.p4 architecture model.

The backend for uBPF is mostly based on P4-to-eBPF compiler. In fact, it implements the same concepts, but generates C code, which is compatible with the user space BPF implementation.

```
                ------------                 ---------
P4_16 --->   | p4c-ubpf | ---> C ----> | clang | --> uBPF
                ------------                 ---------
```

- https://github.com/p4lang/p4c/blob/main/backends/ebpf/README.md

```
             --------------                 -------
P4 --->   | P4-to-eBPF | ---> C ----> | clang/BCC | --> eBPF
             --------------                 -------
```

...

How about a GraalVM-native implementation of P4-like DSL?

- …

1.3 EDA in the Cloud

- https://www.arm.com/company/news/2020/12/arm-moves-production-level-electronic-design-automation-to-the-cloud-with-the-help-of-aws
- FOSS EDA tools are written in various languages

---

you may refer to my previous talks as below:

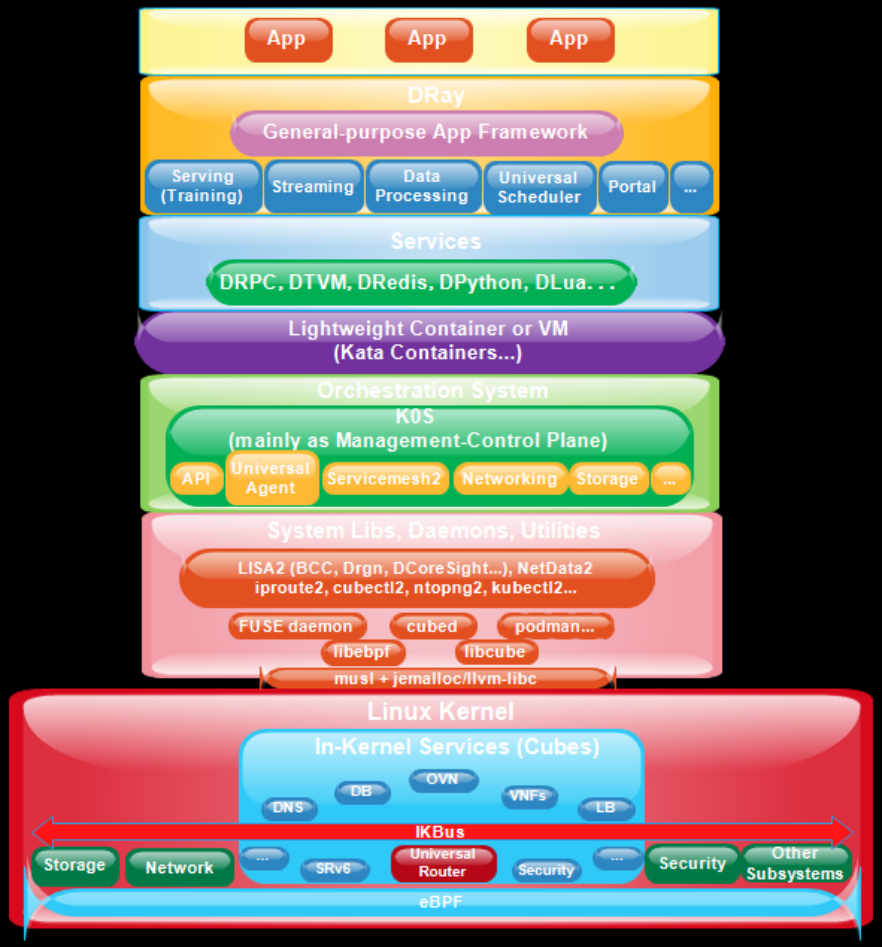
"Scala-based FOSS EDA on ARM" at the 5th China Functional Programming Meetup(Shanghai 2021)

"Python-based Open Source Toolchain for RISC-V Development" at the 1st RISC-V World Conference China(Shanghai 2021)

...

1.4 eBPF-centric infrastructure for edge computing

- Reconstructing nearly every aspect of Linux Kernel Networking & Security subsystem and much more.
- A new design

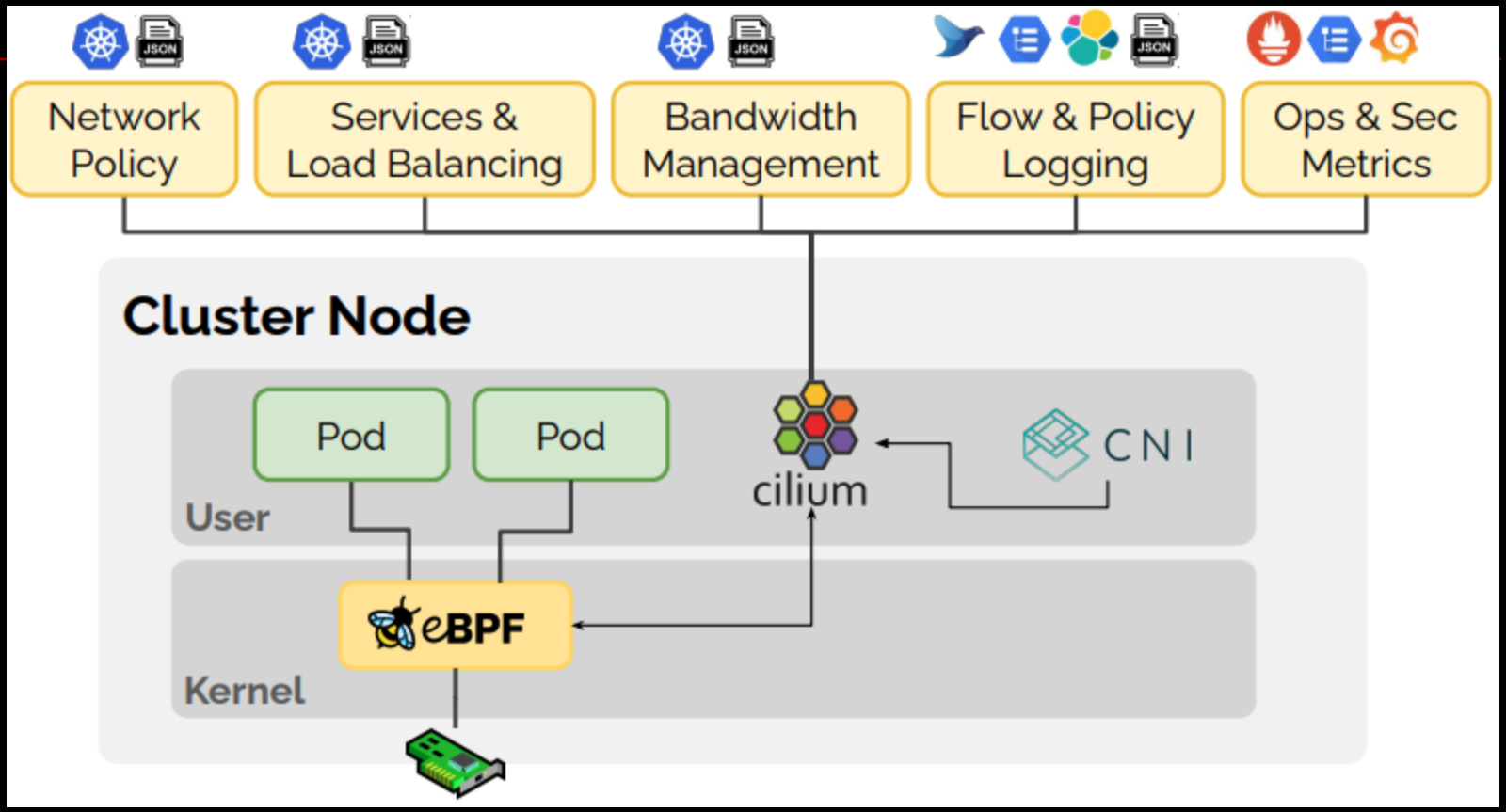


Source: "Rethinking Hyper-Converged Infrastructure for Edge Computing", Feng Li, OpenInfra Days 2019(Shanghai)

GOTC

Cilium

- https://cilium.io/
- eBPF-based Networking, Security, and Observability



- https://cilium.io/blog/2021/05/11/cni-benchmark

GOTC

Rethinking Cilium, Kubernetes, Service Mesh(Istio/Linkerd, Envoy)…

- More and more **lightweight Kubernetes**
- **Krustlet**-like new projects
- Extending **Envoy** with **WASM**

  https://thenewstack.io/wasm-modules-and-envoy-extensibility-explained-part-1/
- Cilium's ambition

…

2) Why is GraalVM

Best choice in current stage

- higher productivity for customizing a VM
- a more mature ecosystem

- many language implementations for reference
- official support for WASM
- less fragmentation and more sustainability
- “one VM to rule them all”
- how about a re-implementation of BCC by leveraging GraalVM?
  https://github.com/remkop/picocli

…

Still need to be further perfected

- further optimize code base
  e.g., OpenJDK 17
- further reduce the consumption of system resources
  e.g., Mandrel for Quarkus
- further improve GraalVM LLVM toolchain
  e.g., enhance the linker and add more tools
- provide a toolkit like VM generator for convenience
  e.g., automated generate a base VM according to an user defined configuration file

…

3) Ideas for Unified eBPF & WASM runtime

GOTC

SuperVM(Register-based)

- **A "superset" of eBPF & WASM**
  1. backward compatible with eBPF bytecode
  2. WASM bytecode could be converted to (similar to convert Java bytecode for Dalvik VM)
  3. hardware-friendly
  …

- **A superset of WASI**
  1. extended system interface for both uBPF & WASM
  2. better support heterogeneous parallel computing
  …

- **New DSLs for Super VM**
  …

In-Kernel Super VM

- One VM to run both eBPF and WASM program in Linux Kernel
- Written in Rust?

https://lwn.net/Articles/853423/ //Rust heads into the kernel?
https://lwn.net/Articles/852704/ //Rust in the Linux kernel
https://lwn.net/Articles/849849/ //Rust support hits linux-next
https://lwn.net/Articles/829858/ //Supporting Linux kernel development in Rust
…
https://www.zdnet.com/article/rust-in-the-linux-kernel-why-it-matters-and-whats-happening-next/
https://blogs.gartner.com/manjunath-bhat/2021/01/03/why-2021-will-be-a-rusty-year-for-system-programmers/
https://developers.slashdot.org/story/21/04/17/009241/linus-torvalds-says-rust-closer-for-linux-kernel-development-calls-c-a-crap-language
https://www.infoq.com/news/2021/04/rust-linux-kernel-development/
https://itwire.com/open-source/rust-support-in-linux-may-be-possible-by-5-14-release-torvalds.html
…
https://github.com/Rust-for-Linux
…
https://github.com/libbpf/libbpf-rs //Idiomatic rust wrapper around libbpf
https://github.com/foniod/redbpf //A Rust eBPF toolchain

V. Wrap-up

- DSLs are becoming increasingly more important.
- GraalVM, WASM and .Net are the key players to future of Polyglot Runtimes.
- eBPF is the key point in the infrastructure of tomorrow's edge computing.
- How about unified runtime for eBPF & WASM?
- Project like OVS(Open vSwitch) that code need to be run both in the user space and kernel space may adopt "a new design".
- Make the things above work on ARM & RISC-V.

GOTC

Q & A

GOTC

Thanks!

Reference

GOTC

Slides/materials from many and varied sources:

- http://en.wikipedia.org/wiki/
- http://www.slideshare.net/
- https://dotnet.microsoft.com/
- https://llvm.org/Users.html
- https://plvision.eu/expertise/sdn-nfv/p4
- https://stackoverflow.com/questions/68485302/is-ebpf-really-a-virtual-machine
- https://www.infoq.com/news/2020/05/java-leyden/
- https://www.infoq.cn/article/c6t2IL23O6EbdQgUpQhb
- https://cloudblogs.microsoft.com/opensource/2021/05/10/making-ebpf-work-on-windows/
- https://github.com/qmonnet/rbpf
- https://doc.dpdk.org/guides/prog_guide/bpf_lib.html
- https://www.infoq.com/articles/java-native-cli-graalvm-picocli/
- …