开源社
kaiyuanshe

使用Java进行eBPF编程

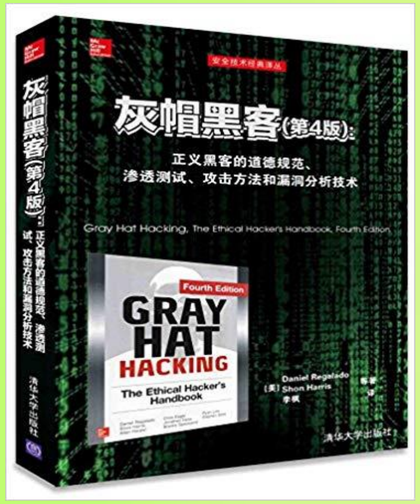
Programming eBPF with Java

李枫（Feng Li）

独立开发者

hkli2013@126.com

Who am I



An indie developer from China.

- **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(~11 years on Mobile dev).**
- **Actively participating Open Source Communities: https://github.com/XianBeiTuoBaFeng2015/MySlides**
- **Recently, focus on infrastructure of Cloud/Edge Computing, AI, IoT, Programming Languages & Runtimes, Network, Virtualization, RISC-V, EDA, 5G/6G…**

Agenda

I. Background
- An overview of xBPF
- Testbeds

II. Project hello-ebpf
- Overview
- Architecture and design
- Internals
- Write eBPF programs in pure Java

III. Discussion on GraalVM-based eBPF development
- uBPF on GraalVM
- Native support for eBPF in GraalVM?

IV. Wrap-up

I. Background

开源社 kaiyuanshe

1) An overview of xBPF

1.1 eBPF

1.1.1 Overview

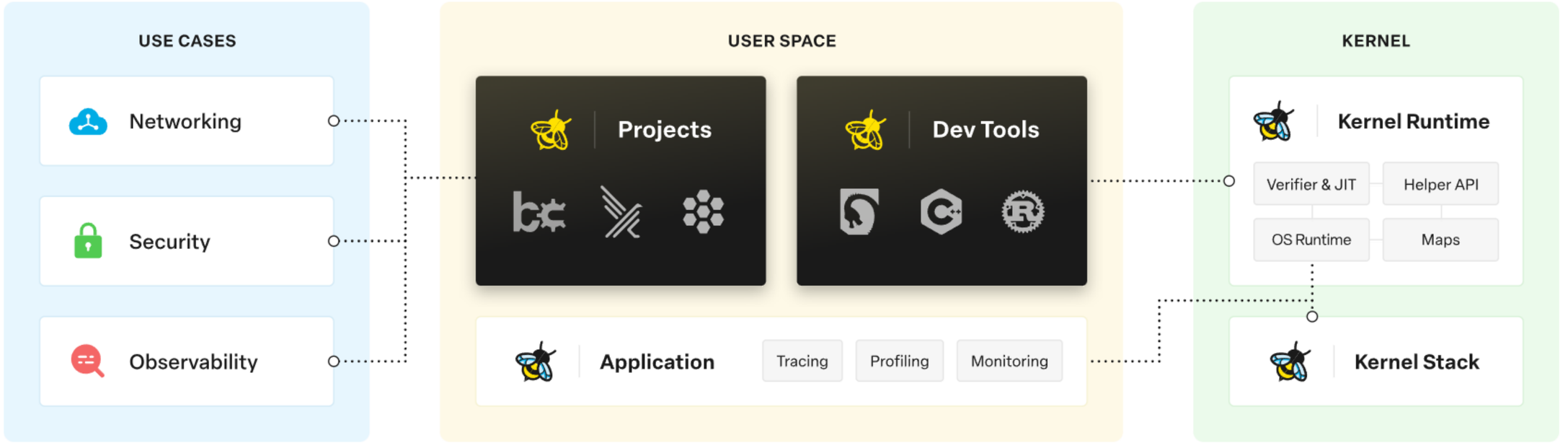
- https://en.wikipedia.org/wiki/Berkeley_Packet_Filter

> eBPF [edit]
>
> *Main article: eBPF*
>
> Since version 3.18, the Linux kernel includes an extended BPF virtual machine with ten 64-bit registers, termed eBPF. It can be used for non-networking purposes, such as for attaching eBPF programs to various tracepoints.[14][15][16] Since kernel version 3.19, eBPF filters can be attached to sockets,[17][18] and, since kernel version 4.1, to traffic control classifiers for the ingress and egress networking data path.[19][20] The original and obsolete version has been retroactively renamed to *classic BPF* (*cBPF*). Nowadays, the Linux kernel runs eBPF only and loaded cBPF bytecode is transparently translated into an eBPF representation in the kernel before program execution.[21] All bytecode is verified before running to prevent denial-of-service attacks. Until Linux 5.3, the verifier prohibited the use of loops, to prevent potentially unbounded execution times; loops with bounded execution time are now permitted in more recent kernels.[22]

- https://en.wikipedia.org/wiki/EBPF
- aims at being a universal in-kernel virtual machine
- dynamically programs the Kernel for efficient networking, observability, tracing, security, and more
- drives the User Space and Kernel Space repartition
- nearly every subsystem of Kernel is or will be effected by eBPF. especially for network and security
- for more details, you may refer to our previous talks and the LTS slides "eBPF: the Past, Present, and Future“
  (https://github.com/XianBeiTuoBaFeng2015/MySlides/blob/master/LTS/eBPF--Past%2C%20Present%2C%20and%20Future__20220922p.pdf)

I. Background

How it works



Source: https://ebpf.io/

I. Background

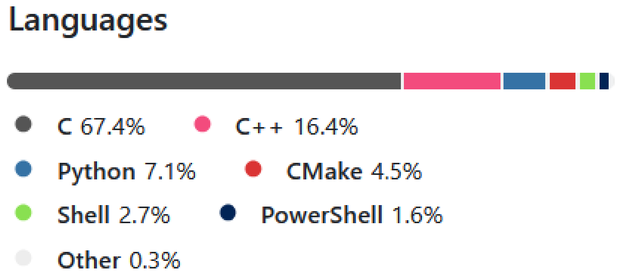
1.1.2 uBPF (userspace BPF)

- Another user-mode interpreter is *uBPF*, which supports JIT and eBPF (without cBPF). Its code has been reused to provide eBPF support in non-Linux systems.[6] Microsoft's *eBPF on Windows* builds on uBPF and the PREVAIL formal verifier.[7] *rBPF*, a Rust rewrite of uBPF, is used by the Solana blockchain platform as the execution engine.[8]

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

- **https://stackoverflow.com/questions/65904948/why-is-having-an-userspace-version-of-ebpf-interesting: mainly applied to Networking , Blockchain and more.**

- **https://github.com/iovisor/ubpf**



I. Background

开源社 kaiyuanshe

1.1.3 xBPF

- From our point of view, **xBPF** stands for **eBPF/uBPF** and all of their **derivatives**, together with the technologies that base on them.
- For more details, you may refer to our previous talk **"Revisiting the eBPF-centric new approach for Hyper-Converged Infrastructure & Edge Computing"** at K+ Summit 2021(Shanghai) and the follow-ups.

I. Background

2) Testbeds

2.1 X86

Intel NUC X15

- 

LAPAC71H(32GB DDR5) with Fedora 40(Linux Kernel 6.11.5).

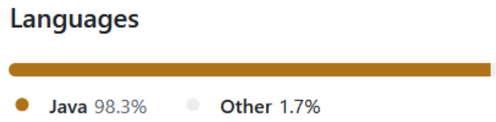
```
[mydev11@koonuc15x-1 boot]$ cat config-6.11.5-200.fc40.x86_64 |grep BPF
CONFIG_BPF=y
CONFIG_HAVE_EBPF_JIT=y
CONFIG_ARCH_WANT_DEFAULT_BPF_JIT=y
# BPF subsystem
CONFIG_BPF_SYSCALL=y
CONFIG_BPF_JIT=y
CONFIG_BPF_JIT_ALWAYS_ON=y
CONFIG_BPF_JIT_DEFAULT_ON=y
CONFIG_BPF_UNPRIV_DEFAULT_OFF=y
CONFIG_BPF_PRELOAD=y
CONFIG_BPF_PRELOAD_UMD=m
CONFIG_BPF_LSM=y
# end of BPF subsystem
CONFIG_CGROUP_BPF=y
CONFIG_IPV6_SEG6_BPF=y
CONFIG_NETFILTER_BPF_LINK=y
CONFIG_NETFILTER_XT_MATCH_BPF=m
CONFIG_NET_CLS_BPF=m
CONFIG_NET_ACT_BPF=m
CONFIG_BPF_STREAM_PARSER=y
CONFIG_LWTUNNEL_BPF=y
CONFIG_BPF_LIRC_MODE2=y
# HID-BPF support
CONFIG_HID_BPF=y
# end of HID-BPF support
CONFIG_BPF_EVENTS=y
CONFIG_TEST_BPF=m
[mydev11@koonuc15x-1 boot]$
```

II. Project hello-ebpf

开源社 kaiyuanshe

1) Overview

- https://github.com/parttimenerd/hello-ebpf/



Hello eBPF world! Hello Java world! Let's discover eBPF together and write Java user-land library along the way.

There are user land libraries for eBPF that allow you to write eBPF applications in C++, Rust, Go, Python and even Lua. But there are none for Java, which is a pity. So... I decided to write my own, which allows you to write eBPF programs directly in Java.

This is still in the early stages, but you can already use it for developing small tools and more coming in the future.

Prerequisites

These might change in the future, but for now, you need the following:

Either a Linux machine with the following:

- Linux 64-bit (or a VM)
- Java 22 or later
- libbpf and bpf-tool
  - e.g. `apt install libbpf-dev linux-tools-common linux-tools-$(uname -r)` on Ubuntu
- root privileges (for executing the eBPF programs)

- On Mac OS, you can use the Lima VM (or use the `hello-ebpf.yaml` file as a guide to install the prerequisites):

```
limactl start hello-ebpf.yaml --mount-writable
limactl shell hello-ebpf sudo bin/install.sh
limactl shell hello-ebpf

# You'll need to be root for most of the examples
sudo -s PATH=$PATH
```

The scheduler examples require a patched 6.11 kernel with the scheduler extensions, you can get it from here. You might also be able to run CachyOS and install a patched kernel from there.

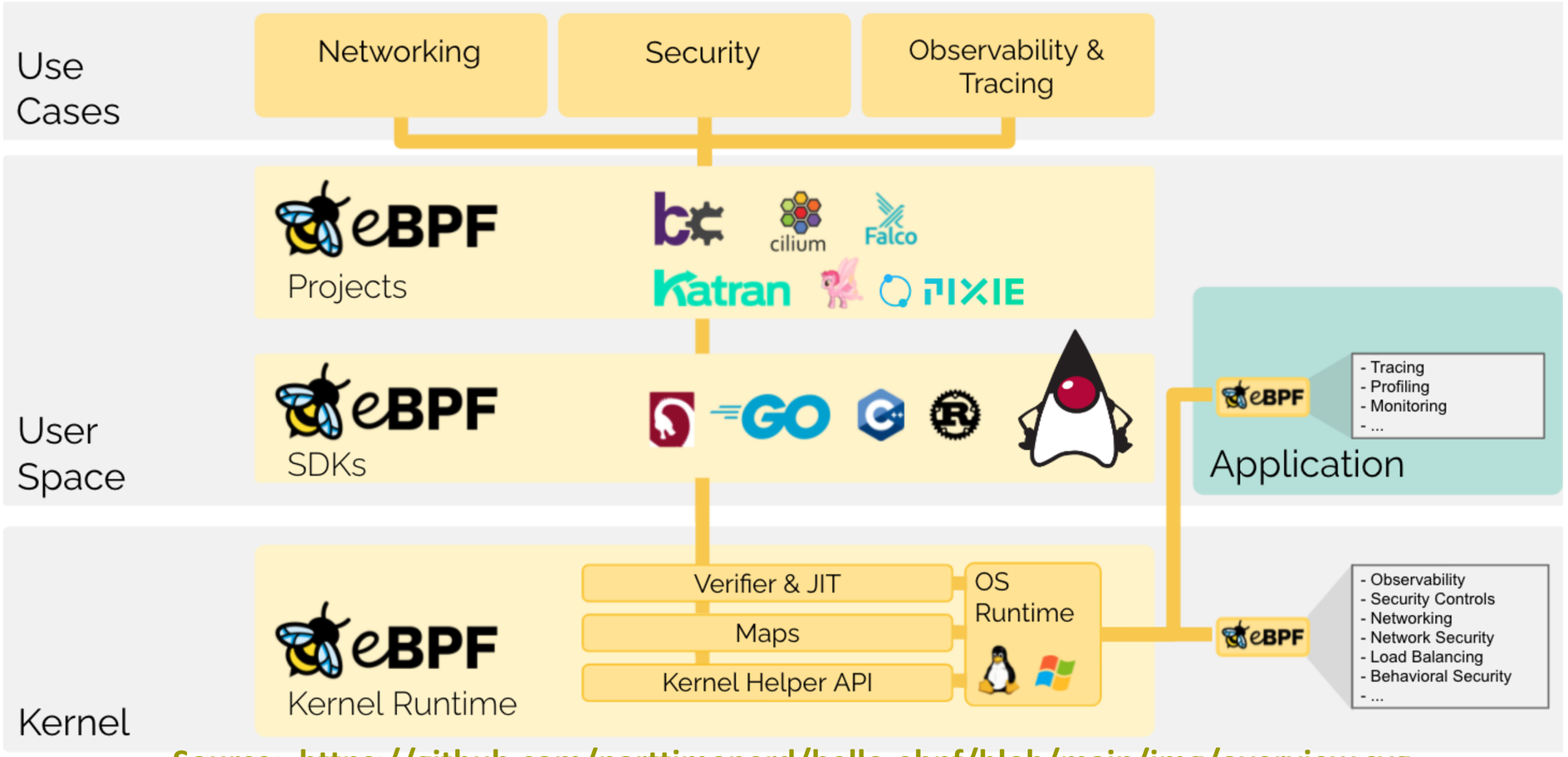
II. Project hello-ebpf

开源社 kaiyuanshe

Goals

- Provide a library (and documentation) for Java developers to explore eBPF and write their own eBPF programs, like firewalls, directly in Java, using the libbpf under the hood.

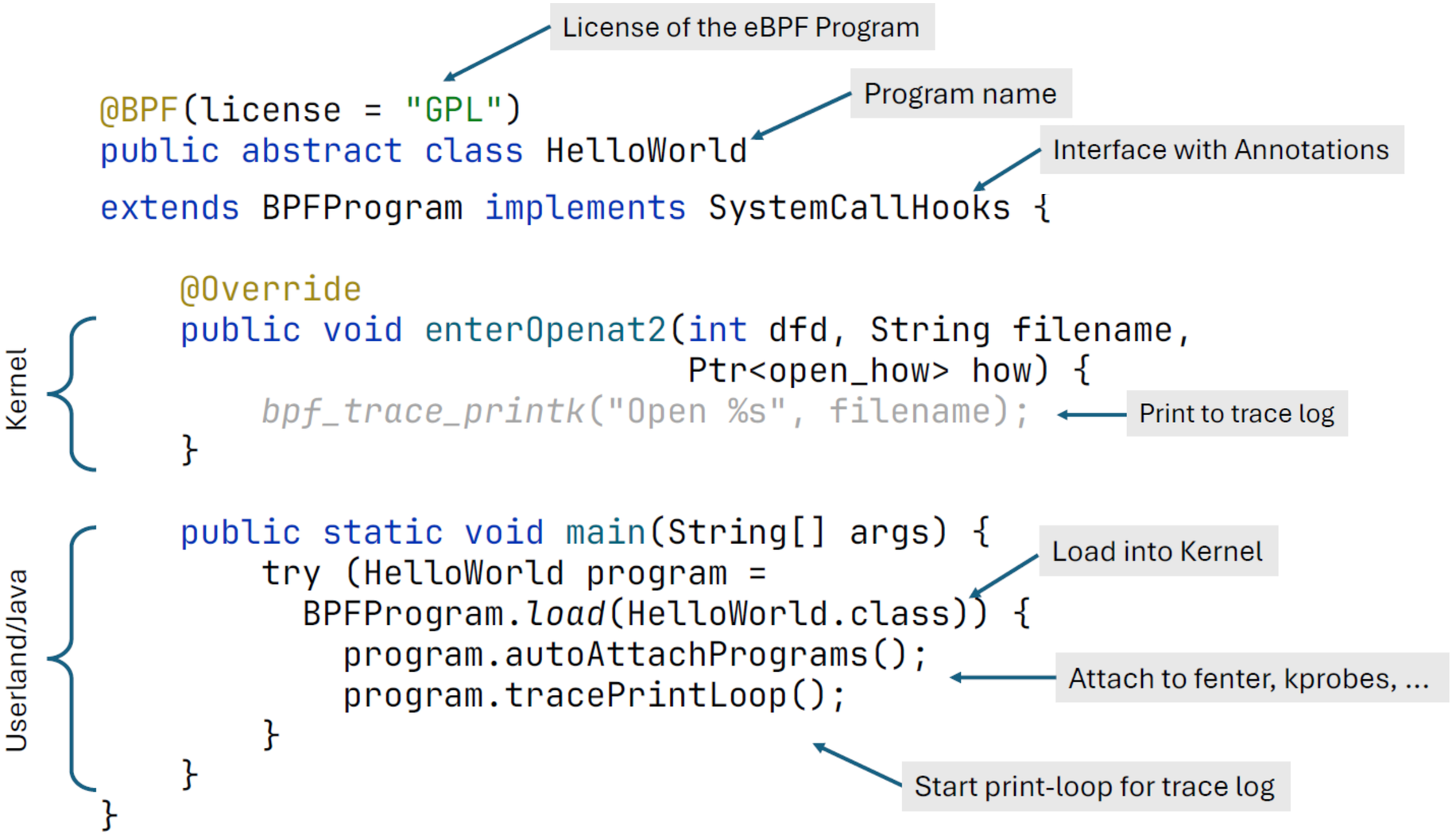
The goal is neither to replace existing eBPF libraries nor to provide a higher abstractions.



Source: https://github.com/parttimenerd/hello-ebpf/blob/main/img/overview.svg

II. Project hello-ebpf

HelloWorld



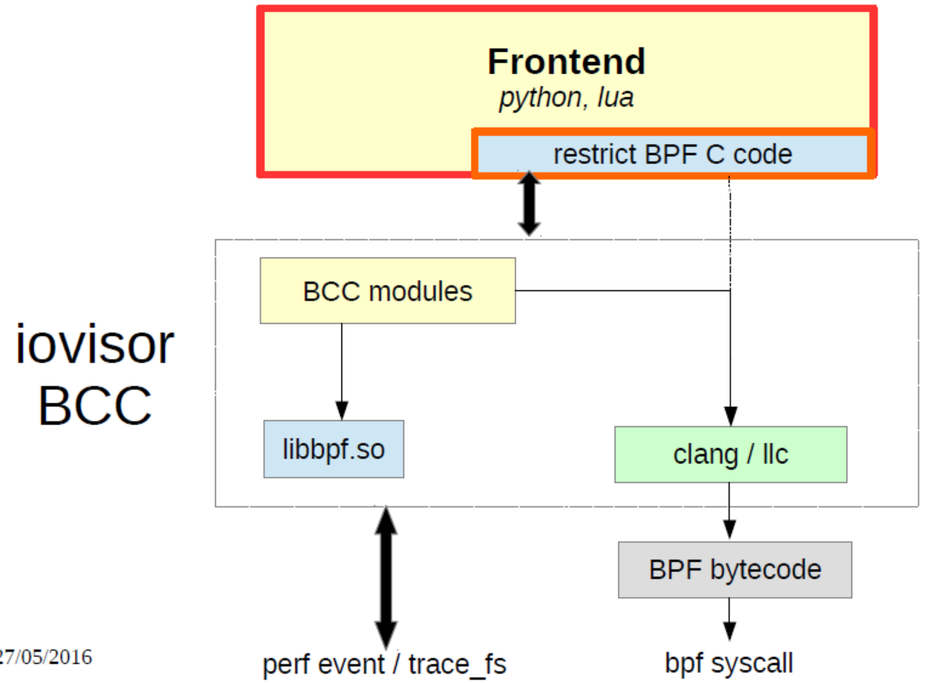

```
@BPF(license = "GPL")
public abstract class HelloWorld
extends BPFProgram implements SystemCallHooks {

    @Override
    public void enterOpenat2(int dfd, String filename,
                             Ptr<open_how> how) {
        bpf_trace_printk("Open %s", filename);
    }

    public static void main(String[] args) {
        try (HelloWorld program =
          BPFProgram.load(HelloWorld.class)) {
            program.autoAttachPrograms();
            program.tracePrintLoop();
        }
    }
}
```

Source: “hello-ebpf: Writing eBPF programs directly in Java”, Johannes Bechberger, LPC 2024.

开源社 kaiyuanshe

2) Architecture and design

Look back on BCC

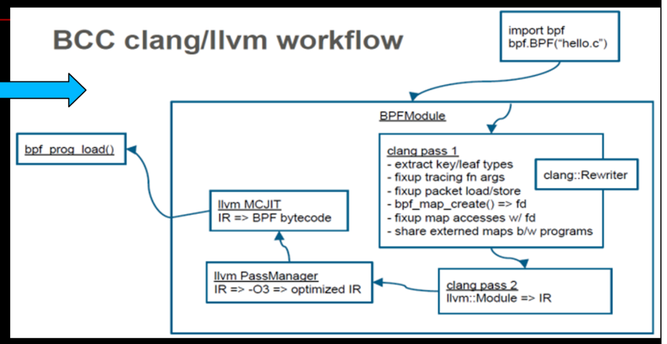


Source: http://www.slideshare.net/vh21/meet-cutebetweenebpfandtracing

A Sample

- https://lwn.net/Articles/747640/ //Some advanced BCC topics

```
#!/usr/bin/env python

from bcc import BPF
from time import sleep

program = """
    BPF_HASH(callers, u64, unsigned long);

    TRACEPOINT_PROBE(kmem, kmalloc) {
        u64 ip = args->call_site;
        unsigned long *count;
        unsigned long c = 1;

        count = callers.lookup((u64 *)&ip);
        if (count != 0)
            c += *count;

        callers.update(&ip, &c);

        return 0;
    }
"""
b = BPF(text=program)

while True:
    try:
        sleep(1)
        for k,v in sorted(b["callers"].items()):
            print ("%s %u" % (b.ksym(k.value), v.value))
        print
    except KeyboardInterrupt:
        exit()
```



Source: http://linuxplumbersconf.org/2015/ocw//system/presentations/3249/original/bpf_llvm_2015aug19.pdf

The output from this little program looks like:

```
# ./example.py
i915_sw_fence_await_dma_fence 4
intel_crtc_duplicate_state 4
SyS_memfd_create 1
drm_atomic_state_init 4
sg_kmalloc 7
intel_atomic_state_alloc 4
seq_open 504
SyS_bpf 22
```

Source: "Python for Linux Kernel Debugging", Feng Li, PyCon China(Hangzhou) 2019.

II. Project hello-ebpf

2.1 Hierarchical structure

-

```
[mydev11@koonuc15x-1 Hello-eBPF]$ tree -L 1 hello-ebpf-main/
hello-ebpf-main/
├── annotations
├── bin
├── bpf
├── bpf-compiler-plugin
├── bpf-compiler-plugin-test
├── bpf-gen
├── bpf-processor
├── bpf-runtime
├── bpf-samples
├── build.sh
├── debug_bpf.sh
├── debug.sh
├── hello-ebpf.yaml
├── img
├── LICENSE
├── mvnw
├── pom.xml
├── rawbpf
├── README.md
├── run_bpf.sh
├── run.sh
├── shared
└── testutil
```

```
[mydev11@koonuc15x-1 Hello-eBPF]$ cat hello-ebpf-main/bin/install.sh
#!/bin/bash
sudo bash -c "echo \"\\\$nrconf{restart} = 'a'\" >> /etc/needrestart/needrestart.conf"
sudo apt-get install -y apt-transport-https ca-certificates curl clang llvm
sudo apt-get install -y libelf-dev libpcap-dev libbfd-dev binutils-dev build-essential make
sudo apt-get install -y linux-tools-common linux-tools-$(uname -r)
sudo apt-get install -y libbpf-dev
sudo apt-get install -y python3-pip zsh tmux
sudo apt-get install -y zip unzip git
curl -s "https://get.sdkman.io" | bash
source "$HOME/.sdkman/bin/sdkman-init.sh"
sdk install java 22-sapmchn
sdk install maven[mydev11@koonuc15x-1 Hello-eBPF]$
```

```
[mydev11@koonuc15x-1 Hello-eBPF]$ tree hello-ebpf-main/testutil
hello-ebpf-main/testutil
├── bin
│   └── java
├── find_and_get_kernel.py
├── oci-lib.sh
├── README.md
├── run-in-container.sh
└── setup-and-run.sh
```

```
[mydev11@koonuc15x-1 Hello-eBPF]$ tree hello-ebpf-main/shared
hello-ebpf-main/shared
├── pom.xml
├── README.md
└── src
    └── main
        └── java
            └── me
                └── bechberger
                    └── ebpf
                        └── shared
                            ├── Constants.java
                            ├── Disassembler.java
                            ├── KernelFeatures.java
                            ├── LibC.java
                            ├── PanamaUtil.java
                            ├── Syscalls.java
                            ├── TraceLog.java
                            ├── util
                            │   ├── DiffUtil.java
                            │   └── LineReader.java
                            └── Util.java
```

II. Project hello-ebpf

开源社 kaiyuanshe

```
[mydev11@koonuc15x-1 Hello-eBPF]$ tree hello-ebpf-main/bpf-runtime
hello-ebpf-main/bpf-runtime
├── build.sh
├── CHANGELOG
├── deploy.sh
├── pom.xml
└── README.md
```

```
[mydev11@koonuc15x-1 Hello-eBPF]$ cat hello-ebpf-main/bpf-runtime/build.sh
#!/usr/bin/env sh

set -e

# get parent-parent folder, shell agnostic
cd "$(dirname "$0")"/.. || exit

(cd bpf-runtime; mvn clean)

mvn package -U && MAVEN_OPTS="-Xss1000m" time java -jar bpf-gen/target/bpf-gen.jar bpf-runtime/src/main/java/ bpf-gen/data/hel
per-defs.json

(cd bpf-runtime; MAVEN_OPTS="-Xss1000m" mvn package -U)
[mydev11@koonuc15x-1 Hello-eBPF]$
```

```
[mydev11@koonuc15x-1 Hello-eBPF]$ cat hello-ebpf-main/bpf-runtime/deploy.sh
#!/usr/bin/env sh

set -e

# get parent-parent folder, shell agnostic
cd "$(dirname "$0")"/.. || exit

(cd bpf-runtime; mvn clean; MAVEN_OPTS="-Xss1000m" mvn package deploy -U)[mydev11@koonuc15x-1 Hello-eBPF]$
[mydev11@koonuc15x-1 Hello-eBPF]$
```

II. Project hello-ebpf

```
[mydev11@koonuc15x-1 Hello-eBPF]$ tree hello-ebpf-main/bpf
hello-ebpf-main/bpf
├── pom.xml
├── README.md
└── src
    ├── main
    │   └── java
    │       └── me
    │           └── bechberger
    │               └── ebpf
    │                   └── bpf
    │                       ├── BPFError.java
    │                       ├── BPFJ.java
    │                       ├── BPFProgram.java
    │                       ├── CGroupHook.java
    │                       ├── GlobalVariable.java
    │                       ├── LSMHook.java
    │                       ├── map
    │                       │   ├── BPFArray.java
    │                       │   ├── BPFBaseMap.java
    │                       │   ├── BPFHashMap.java
    │                       │   ├── BPFLRUHashMap.java
    │                       │   ├── BPFMap.java
    │                       │   ├── BPFQueueAndStack.java
    │                       │   ├── BPFQueue.java
    │                       │   ├── BPFRingBuffer.java
    │                       │   ├── BPFStack.java
    │                       │   ├── FileDescriptor.java
    │                       │   └── MapTypeId.java
    │                       ├── NetworkUtil.java
    │                       ├── Scheduler.java
    │                       ├── TCHook.java
    │                       ├── Util.java
    │                       └── XDPHook.java
    └── test
        └── java
            └── me
                └── bechberger
                    └── ebpf
                        └── bpf
                            ├── ArrayMapTest.java
                            ├── DataTypeTest.java
                            ├── FEntryExitAutoAttachTest.java
                            ├── GlobalVariableTest.java
                            ├── HashMapTest.java
                            ├── HelloWorldTest.java
                            ├── MapGenerationTest.java
                            ├── QueueMapTest.java
                            ├── RingBufferTest.java
                            ├── TestUtil.java
                            ├── TypeLayoutTest.java
                            └── TypeProcessingTest.java
```

```
[mydev11@koonuc15x-1 Hello-eBPF]$ tree hello-ebpf-main/bpf-processor
hello-ebpf-main/bpf-processor
├── pom.xml
├── README.md
└── src
    ├── main
    │   └── java
    │       └── me
    │           └── bechberger
    │               ├── cast
    │               │   └── CAST.java
    │               └── ebpf
    │                   ├── bpf
    │                   │   └── processor
    │                   │       ├── AnnotationUtils.java
    │                   │       ├── BPFTypeLike.java
    │                   │       ├── CompilationCache.java
    │                   │       ├── CompilerErrorProcessor.java
    │                   │       ├── DefinedTypes.java
    │                   │       ├── Processor.java
    │                   │       └── TypeProcessor.java
    │                   ├── NameUtil.java
    │                   └── type
    │                       ├── BoxHelper.java
    │                       ├── BPFType.java
    │                       ├── Enum.java
    │                       ├── Ptr.java
    │                       ├── Struct.java
    │                       ├── TypedefBase.java
    │                       ├── Typedef.java
    │                       ├── TypedEnum.java
    │                       ├── TypeUtils.java
    │                       └── Union.java
    └── test
        └── java
            └── me
                └── bechberger
                    ├── cast
                    │   └── ASTTest.java
                    └── ebpf
                        └── bpf
                            └── processor
                                └── CompilerErrorProcessorTest.java
```

II. Project hello-ebpf

```
[mydev11@koonuc15x-1 Hello-eBPF]$ tree hello-ebpf-main/bpf-compiler-plugin
hello-ebpf-main/bpf-compiler-plugin
├── pom.xml
├── README.md
└── src
    └── main
        ├── java
        │   └── me
        │       └── bechberger
        │           └── ebpf
        │               └── bpf
        │                   └── compiler
        │                       ├── CompilerPlugin.java
        │                       ├── MethodHeaderTemplate.java
        │                       ├── MethodTemplateCache.java
        │                       ├── MethodTemplate.java
        │                       ├── NullHelpers.java
        │                       └── Translator.java
        └── resources
            └── META-INF
                └── services
                    └── com.sun.source.util.Plugin
```

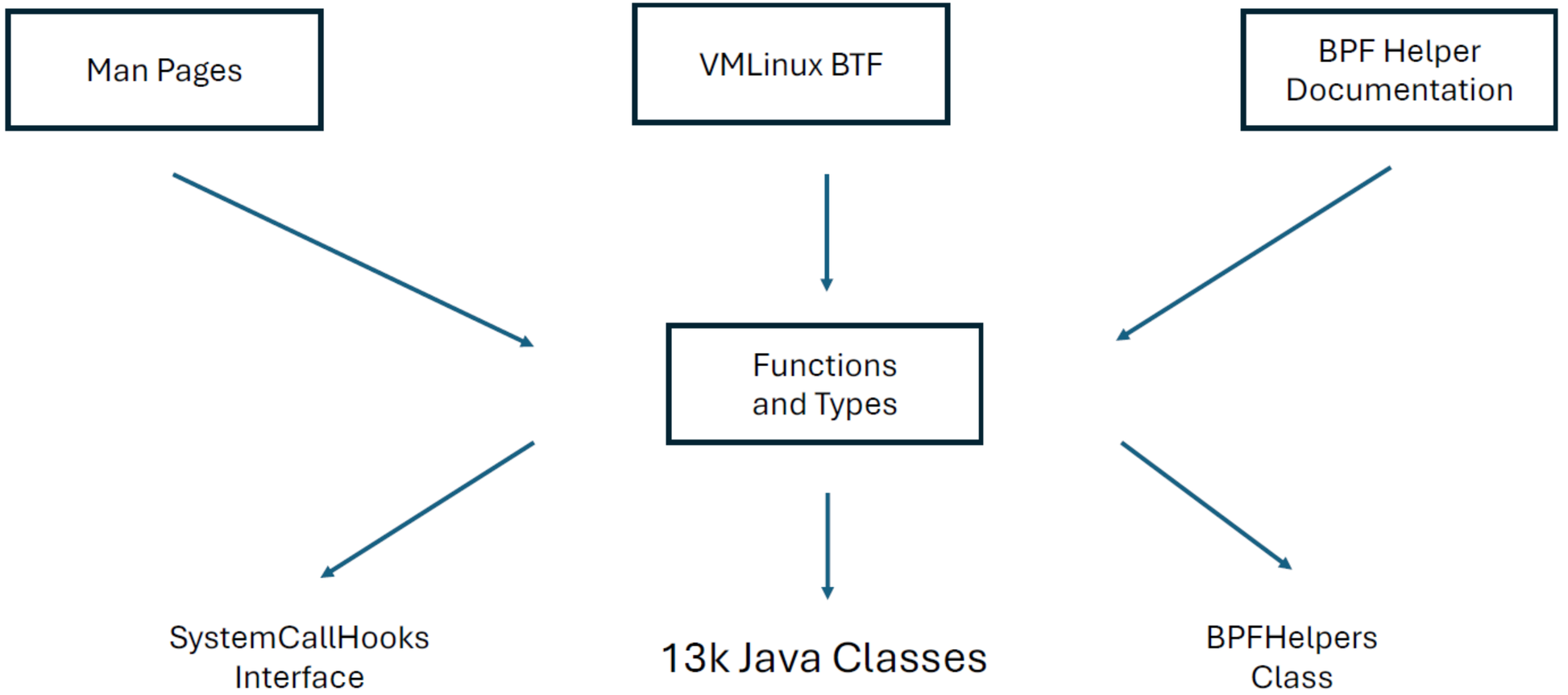
```
[mydev11@koonuc15x-1 Hello-eBPF]$ tree hello-ebpf-main/annotations
hello-ebpf-main/annotations
├── pom.xml
├── README.md
└── src
    └── main
        └── java
            └── me
                └── bechberger
                    └── ebpf
                        └── annotations
                            ├── AlwaysInline.java
                            ├── AnnotationInstances.java
                            ├── bpf
                            │   ├── BPFFunctionAlternative.java
                            │   ├── BPFFunction.java
                            │   ├── BPFImpl.java
                            │   ├── BPFInterface.java
                            │   ├── BPF.java
                            │   ├── BPFMapClass.java
                            │   ├── BPFMapDefinition.java
                            │   ├── BuiltinBPFFunction.java
                            │   ├── InternalBody.java
                            │   ├── InternalMethodDefinition.java
                            │   ├── MethodIsBPFRelatedFunction.java
                            │   ├── NotUsableInJava.java
                            │   ├── Properties.java
                            │   ├── PropertyDefinition.java
                            │   ├── PropertyDefinitions.java
                            │   ├── Property.java
                            │   └── Requires.java
                            ├── CustomType.java
                            ├── EnumMember.java
                            ├── Includes.java
                            ├── InlineUnion.java
                            ├── Offset.java
                            ├── OriginalName.java
                            ├── OriginalNames.java
                            ├── Size.java
                            ├── Sizes.java
                            ├── Type.java
                            └── Unsigned.java
```

II. Project hello-ebpf

开源社 kaiyuanshe

```
[mydev11@koonuc15x-1 Hello-eBPF]$ tree hello-ebpf-main/bpf-gen
hello-ebpf-main/bpf-gen
├── data
│   ├── helper-defs.json
│   └── README.md
├── pom.xml
├── README.md
└── src
    ├── main
    │   └── java
    │       └── me
    │           └── bechberger
    │               └── ebpf
    │                   └── gen
    │                       ├── BTF.java
    │                       ├── DeclarationParser.java
    │                       ├── Generator.java
    │                       ├── HelperJSONProcessor.java
    │                       ├── KnownTypes.java
    │                       ├── Main.java
    │                       ├── Markdown.java
    │                       └── SystemCallProcessor.java
    └── test
        └── java
            └── me
                └── bechberger
                    └── ebpf
                        └── gen
                            ├── DeclarationParserTest.java
                            ├── GeneratorTest.java
                            └── SystemCallProcessorTest.java
```

2.2 VMLinux

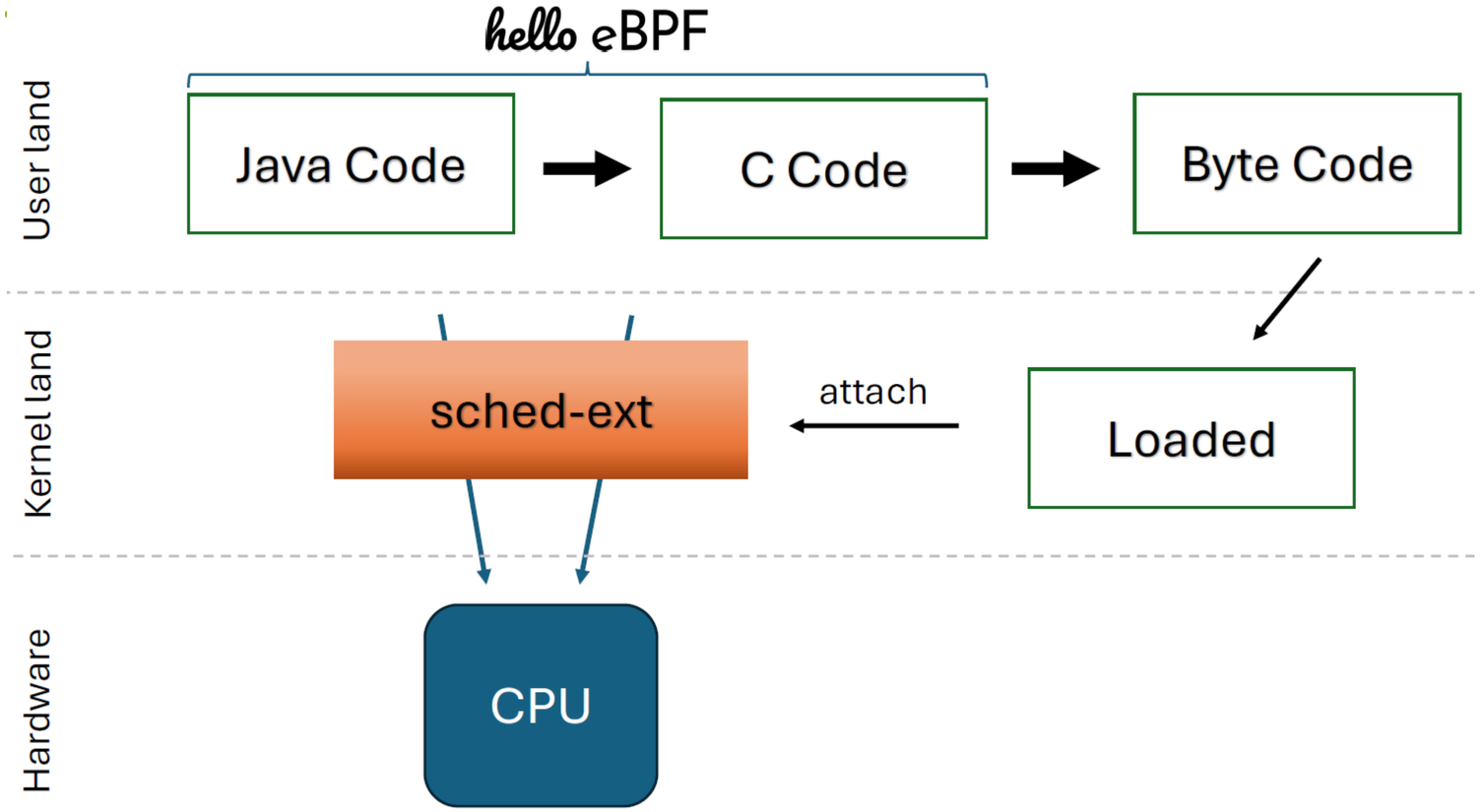


Source: “hello-ebpf: Writing eBPF programs directly in Java”, Johannes Bechberger, LPC 2024.

开源社 kaiyuanshe

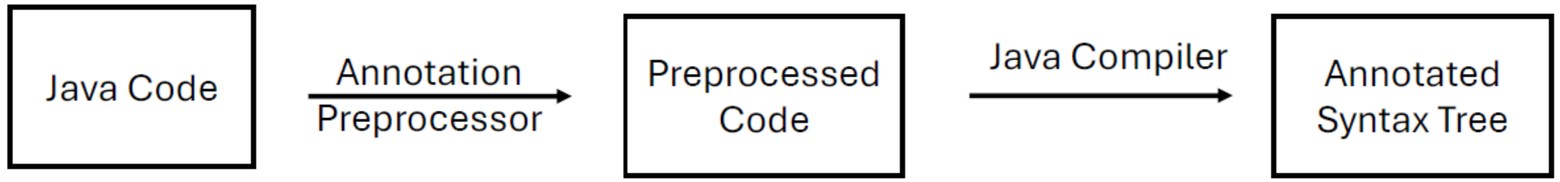
3) Internals

Workflow



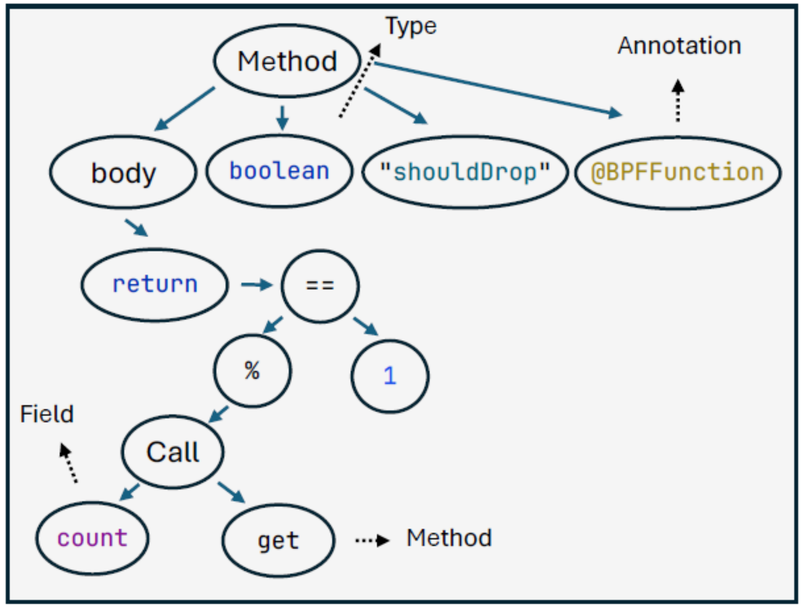
Source: "hello-ebpf: Writing eBPF programs directly in Java", Johannes Bechberger, LPC 2024.

II. Project hello-ebpf

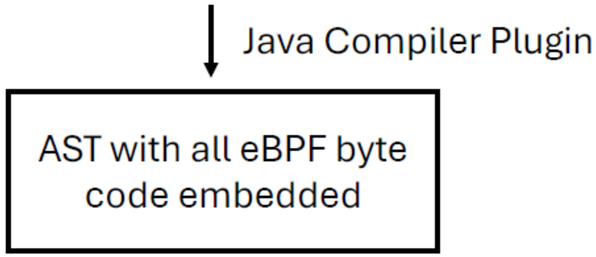



```
final GlobalVariable
 <@Unsigned Integer> count
  = ...;

@BPFFunction
public boolean shouldDrop() {
    return count.get() % 3
           == 1;
}
```

```
static final String
   EBPF_PROGRAM = """
        // license, ...
        u32 count SEC(".data");
        """;


@BPFFunction
public boolean shouldDrop() {
    return count.get() % 3
           == 1;
}
```




```
@BPF(license = "GPL")
public abstract class XDPDropEveryThirdPacket
extends BPFProgram implements XDPHook {

    final GlobalVariable<@Unsigned Integer> count = new GlobalVariable<>(0);

    @BPFFunction
    public boolean shouldDrop() {
        return count.get() % 3 == 1;
    }

    @Override
    public xdp_action xdpHandlePacket(Ptr<xdp_md> ctx) {
        count.set(count.get() + 1);
        return shouldDrop() ? xdp_action.XDP_DROP : xdp_action.XDP_PASS;
    }

    // ...
}
```



Source: “hello-ebpf: Writing eBPF programs directly in Java”, Johannes Bechberger, LPC 2024.

II. Project hello-ebpf

开源社 kaiyuanshe

Data structures

- **Structs**

```
@Type
struct Event {
    @Unsigned int pid;
    @Size(256) String f;
}

@BPFFunction
int access(       Event event) {
    event.pid = 5;
    return event.pid;
}
```

```
struct Event {
    u32 pid;
    u8 filename[256];
};


s32 access(struct Event event) {
    event.pid = 5;
    return event.pid;
}
```

- **Unions**

```
@Type
static class SampleUnion extends Union {
    @Unsigned
    int ipv4;
    long count;
}
```

Source: “hello-ebpf: Writing eBPF programs directly in Java”, Johannes Bechberger, LPC 2024.

II. Project hello-ebpf

- **Pointers** Ptr<T>

```java
public class Ptr<T> {

    @BuiltinBPFFunction("(*($this))")
    public T val() {}

    @BuiltinBPFFunction("&($arg1)")
    public static <T> Ptr<T> of(@Nullable T value){}

    @BuiltinBPFFunction("((void*)0)")
    public static Ptr<?> ofNull() {}

    @BuiltinBPFFunction("(($T1*)$this)")
    public <S> Ptr<S> cast() {}
    // ...
}
```

```java
@BPFFunction
public int refAndDeref() {
    int value = 3;
    Ptr<Integer> ptr = Ptr.of(value);
    return ptr == Ptr.ofNull() ? 1 : 0;
}
```

```java
s32 refAndDeref() {
    s32 value = 3;
    s32*          ptr = &value;
    return ptr == ((void*)0)   ? 1 : 0;
}
```

Source: “hello-ebpf: Writing eBPF programs directly in Java”, Johannes Bechberger, LPC 2024.

II. Project hello-ebpf

- **Maps**

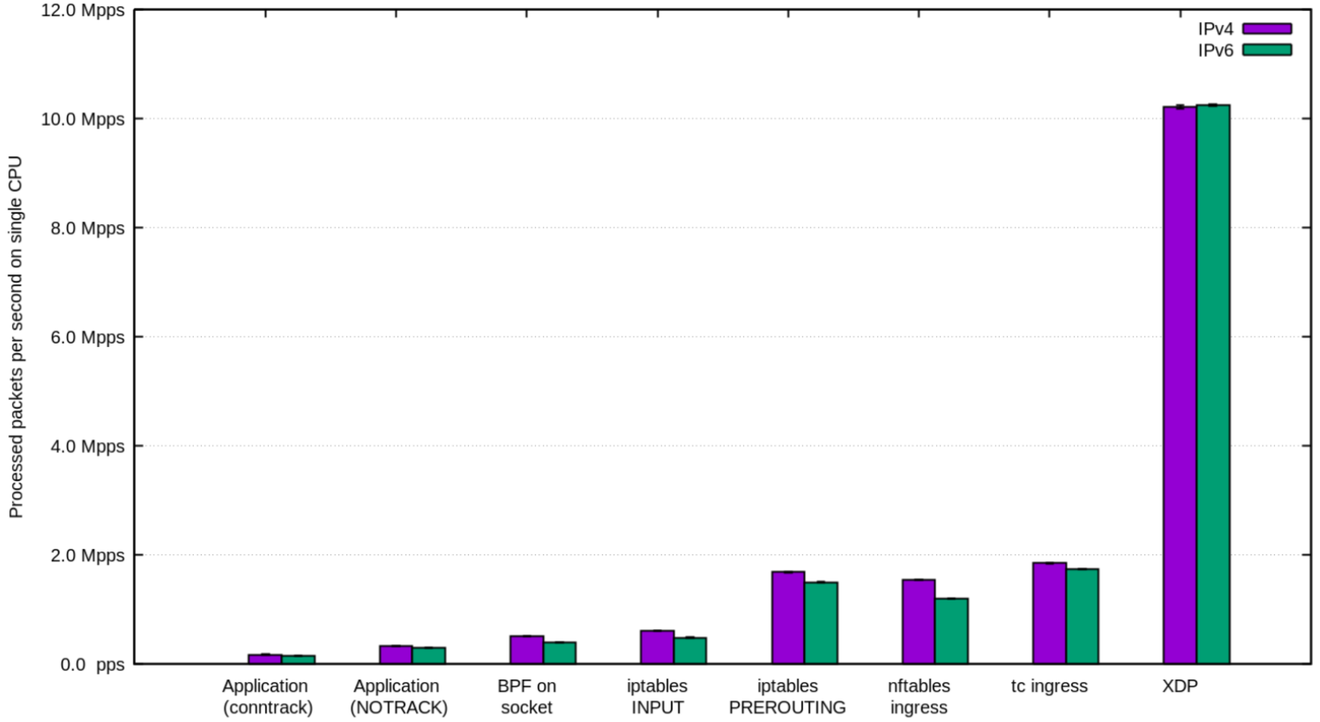
```
@BPFMapDefinition(maxEntries = 100 * 1024)
BPFHashMap<@Size(STRING_SIZE) String, Entry> map;
// in eBPF
String key = ...;
map.bpf_get(key); map.put(key, entry);
// in Java
program.map.get(key)
program.map.put(key, value)
```

Source: “hello-ebpf: Writing eBPF programs directly in Java”, Johannes Bechberger, LPC 2024.

4) Write eBPF programs in pure Java

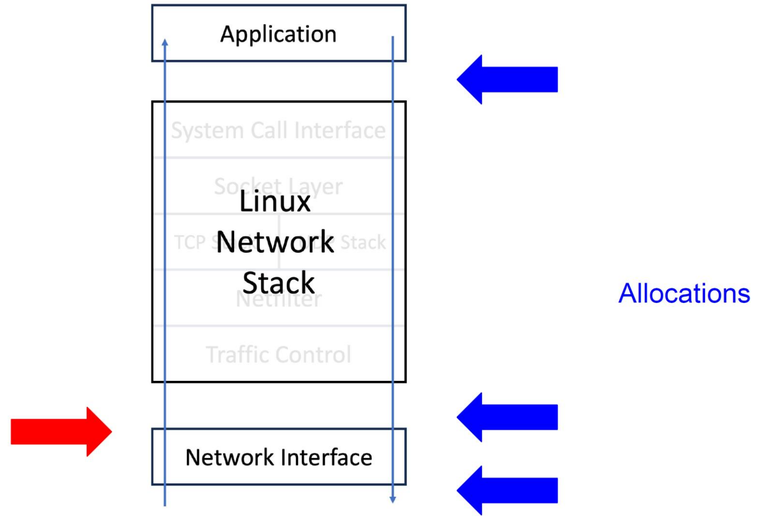
4.1 Building a firewall

Packet dropping performance



Source: https://blog.cloudflare.com/how-to-drop-10-million-packets/

XDP



Source: "Building a Lightning Fast Firewall in Java & eBPF", Johannes Bechberger, Javazone 2024.

4.1.1 Building a Lightning Fast Firewall with Java & eBPF

Libraries

- **Implement a packet parsing library**

```
@BPFInterface
public interface BasePacketParser {

    int HTTP_PORT = 80;
    int HTTPS_PORT = 443;

    // ...

    @Type
    static class PacketInfo {
        public PacketDirection direction;
        public Protocol protocol;
        public IPAddress source;
        public IPAddress destination;
        public @Unsigned int destinationPort;
        public @Unsigned int sourcePort;
        public int length;
    }

    // ...

    @BPFFunction
    @AlwaysInline
    default boolean parsePacket2(@Unsigned int start,
      @Unsigned int end, Ptr<PacketInfo> info) {
        return parsePacket(Ptr.voidPointer(start), Ptr.voidPointer(e
    }

    /**
     * Parse a packet and extract the source and destination IP addr
     *
     * @param start start of the packet data
     * @param end   end of the packet data
     * @param info  output parameter for the extracted information
     * @return true if the packet is an IP packet and could be parse
     */
    @BPFFunction
    @AlwaysInline
    default boolean parsePacket(Ptr<?> start,
      Ptr<?> end, Ptr<PacketInfo> info) {

        // ...

        if (ethType == XDPHook.ETH_P_IP) {
            return parseIPPacket(
              start.add(offset).<runtime.iphdr>cast(), end, info);
        }
        if (ethType == XDPHook.ETH_P_IPV6) {
            return parseIPv6Packet(
              start.add(offset).<runtime.ipv6hdr>cast(), end, info);
        }
        return false;
    }
}
```

This library mechanism allows you to use the methods by simply implementing the interface in your BPF program, making it fairly easy to parse and filter packets (see BlockHTTP on GitHub):

```
@BPF(license = "GPL")
public abstract class BlockHTTP2 extends BPFProgram
  implements XDPHook, BasePacketParser {

    @Override
    public xdp_action xdpHandlePacket(Ptr<xdp_md> packet) {
        PacketInfo info = new PacketInfo();
        if (parsePacket2(packet.val().data,
                         packet.val().data_end,
                         Ptr.of(info))) {
            if (info.sourcePort == HTTP_PORT) {
                BPFJ.bpf_trace_printk("Dropping https packet\n");
                return xdp_action.XDP_DROP;
            }
        }
        return xdp_action.XDP_PASS;
    }

    // ...
}
```

The main advantage of implementing libraries as Java interfaces is that their properties align:

- Interfaces can't have instance variables, so defining global variables or maps is impossible. This constraint simplifies the implementation of the library mechanism, as the Java compiler plugin has to deal with less complexity.
- A program can implement and, therefore, use multiple interfaces.
- Finding the used libraries is easy, as they are all declared initially.

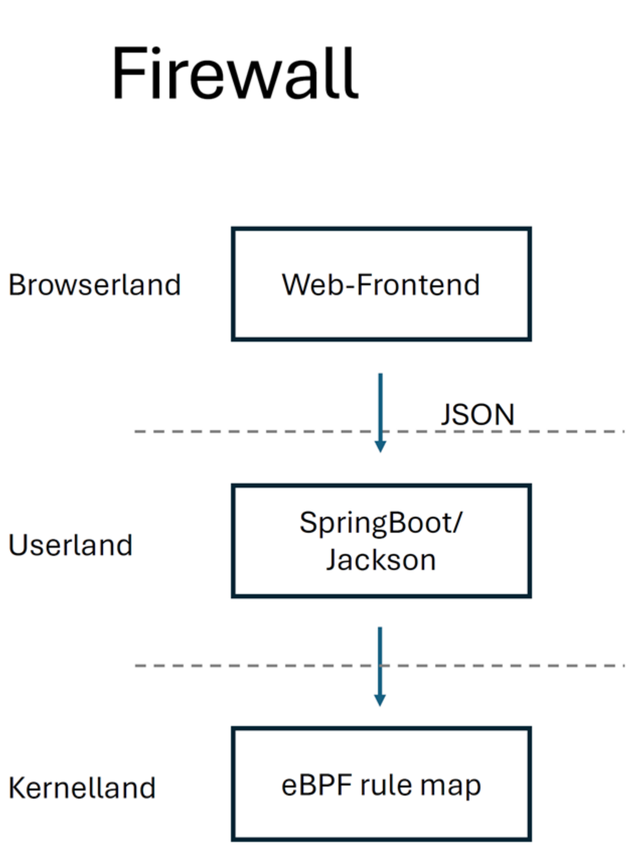
We can extend this into a firewall with rules. The implementation is split into the base firewall (GitHub) and the Spring based frontend (GitHub).

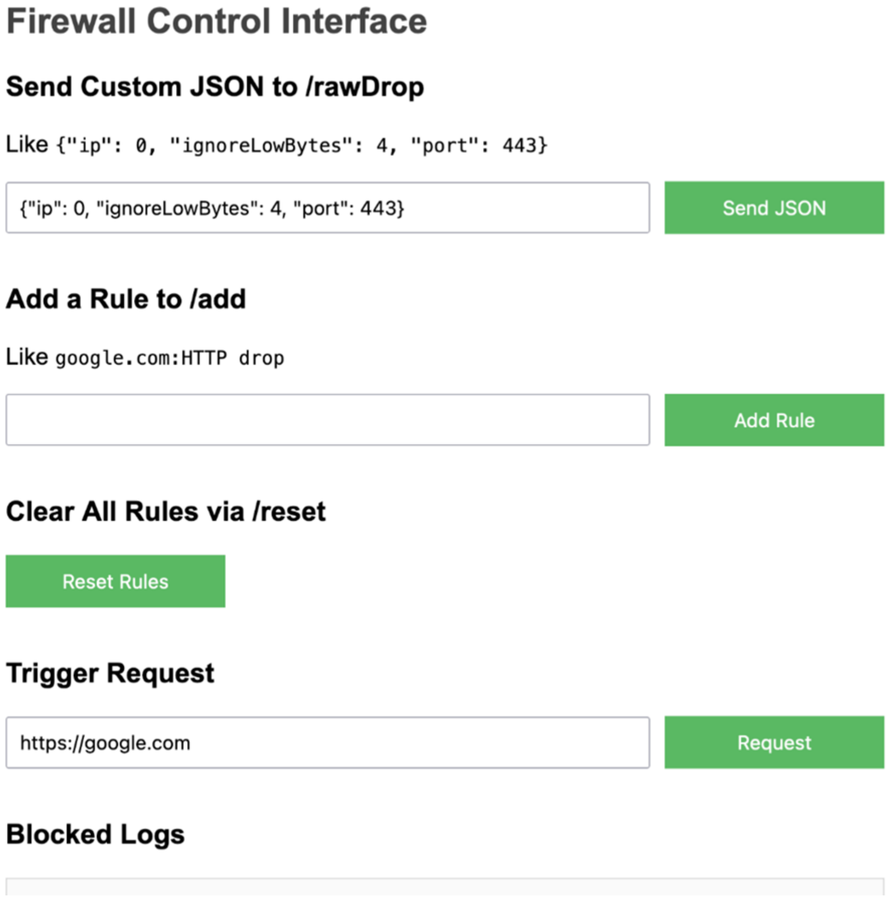
Source: https://mostlynerdless.de/blog/2024/08/27/hello-ebpf-building-a-lightning-fast-firewall-with-java-ebpf-14/

II. Project hello-ebpf

开源社 kaiyuanshe

Demo





Source: “hello-ebpf: Writing eBPF programs directly in Java”, Johannes Bechberger, LPC 2024.

4.2 Writing a Linux scheduler

4.2.1 sched-ext

- https://github.com/sched-ext/scx/

Sched_ext Schedulers and Tools

Languages

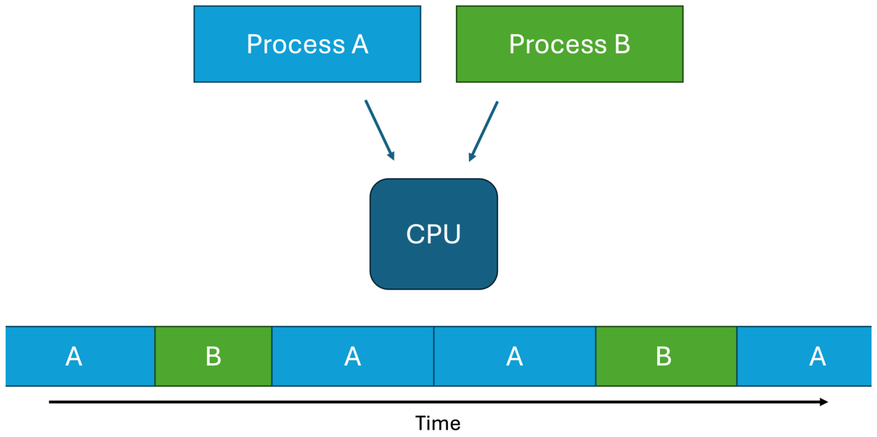
C 96.4% Rust 3.3% Other 0.3%

sched_ext is a Linux kernel feature which enables implementing kernel thread schedulers in BPF and dynamically loading them. This repository contains various scheduler implementations and support utilities.

sched_ext enables safe and rapid iterations of scheduler implementations, thus radically widening the scope of scheduling strategies that can be experimented with and deployed; even in massive and complex production environments.

You can find more information, links to blog posts and recordings, in the wiki. The following are a few highlights of this repository.

- The scx_layered case study concretely demonstrates the power and benefits of sched_ext.
- For a high-level but thorough overview of the sched_ext (especially its motivation), please refer to the overview document.
- For a description of the schedulers shipped with this tree, please refer to the schedulers document.
- The following video is the scx_rustland scheduler which makes most scheduling decisions in userspace Rust code showing better FPS in terraria while kernel is being compiled. This doesn't mean that scx_rustland is a better scheduler but does demonstrate how safe and easy it is to implement a scheduler which is generally usable and can outperform the default scheduler in certain scenarios.



Source: https://mostlynerdless.de/blog/2024/09/10/hello-ebpf-writing-a-linux-scheduler-in-java-with-ebpf-15/

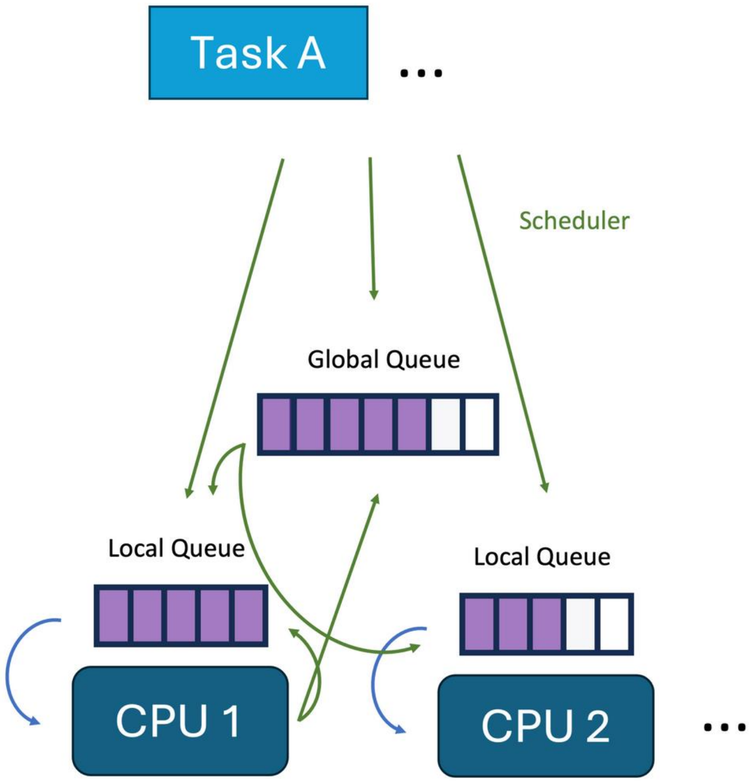
- https://www.kernel.org/doc/html/next/scheduler/sched-ext.html

4.2.2 Writing a Linux scheduler in Java with eBPF

Design

Basic Idea

Most basic schedulers have two sets of queues: local scheduling queues per CPU each CPU uses to get the next task to execute and global queues used for inter-CPU distribution of functions. The task of the scheduler is to move the functions between the queues based on its scheduling policy:



The Scheduler Interface

The primary data structure of sched-ext is the sched_ext_ops operations table. Implementing the Scheduler interface methods allows you to populate this table with custom methods. Let's take a look at the different methods in the following; we see later how we can implement them to create two small schedulers. I recommend reading the ext.c kernel source file, which defines all sched-ext related methods and structs and is well documented. The following description is a shortened version of the comments on sched_ext_ops. To avoid the confusing terms "thread" and "process," I'll be using "task" in the following, which is the process in the Linux kernel sense of the word and effectively means a thread in most applications.

Now to the description of the methods, albeit just of the subset that is required for the small demo sampler:

- **int init()**: Called when the scheduler is initialized, used typically to initialize scheduling queues and other data structures, might return an error code (as far as I understand).
- **void exit(Ptr<scx_exit_info> ei)**: Called when the scheduler exits, the exit info contains, for example, the exit reason.
- **int selectCPU(Ptr<task_struct> p, int prev_cpu, long wake_flags)**: Called to select the CPU for a task that is just woken up (and is soon to be scheduled). A task can be directly dispatched to an idle CPU by using the `scx_bpf_dispatch()` in this method, dispatching it to the returned CPU (as far as I understand). If the task is later dispatched to `SCX_DSQ_LOCAL`, then it will be dispatched to the scheduling CPU of the selected CPU.
- **void enqueue(Ptr<task_struct> p, long enq_flags)**: Called whenever a task is ready to be dispatched again if it is not already dispatched in `selectCPU()`. This method typically enqueues the task in a queue, see `selectCPU`, or dispatches the task directly to the selected CPU using the `scx_bpf_dispatch()` method.
- **void dispatch(int cpu, Ptr<task_struct> prev)**: Called when the CPU's local scheduling queue is empty to fill it by getting a task via `scx_bpf_consume()` from another scheduling queue or by directly dispatching a task via `scx_bpf_dispatch()`.
- **void running(Ptr<task_struct> p)**: Called when a task starts to run on its associated CPU.
- **void stopping(Ptr<task_struct> p, boolean runnable)**: Called when a task stops its execution on a CPU.
- **void enable(Ptr<task_struct> p)**: Called whenever a task starts to be scheduled by the currently implemented scheduler.

Source: https://mostlynerdless.de/blog/2024/09/10/hello-ebpf-writing-a-linux-scheduler-in-java-with-ebpf-15/

II. Project hello-ebpf

Implementations

FIFO Scheduler

The First-In-First-Out (FIFO) scheduler is the first scheduler we'll implement, based on the scx_simple from scx. This scheduler has a pretty simple policy: It schedules the tasks in order of their arrival for a time slice of 20 ms. The advantage is that scheduling decisions are made quickly.

But there is a major drawback: The scheduler isn't fair. Consider two tasks: a task `A` that just runs for a burst of 1 ms and then sleeps and a task B that runs continuously. Even though A uses only 5% of its allotted time slice every time it is scheduled, it will, on average, be scheduled the same number of times as `B`, thus `B` runs 20 times longer on the CPU. We see later why this is a problem.

But first, let's look at the implementation, omitting the user-land and logging code:

```
@BPF(license = "GPL")
public abstract class FIFOScheduler extends BPFProgram
                                    implements Scheduler {

    /**
     * A custom scheduling queue
     */
    static final long SHARED_DSQ_ID = 0;

    @Override
    public int init() {
        // init the scheduling queue
        return scx_bpf_create_dsq(SHARED_DSQ_ID, -1);
    }

    @Override
    public int selectCPU(Ptr<task_struct> p, int prev_cpu,
                         long wake_flags) {
        boolean is_idle = false;
        // let sched-ext select the best CPU
        int cpu = scx_bpf_select_cpu_dfl(p, prev_cpu,
                                         wake_flags,
                                         Ptr.of(is_idle));
        if (is_idle) {
            // directly dispatch to the CPU if it is idle
            scx_bpf_dispatch(p, SCX_DSQ_LOCAL.value(),
                             SCX_SLICE_DFL.value(),0);
        }
        return cpu;
    }

    @Override
    public void enqueue(Ptr<task_struct> p, long enq_flags) {
        // directly dispatch to the selected CPU's local queue
        scx_bpf_dispatch(p, SHARED_DSQ_ID,
                         SCX_SLICE_DFL.value(), enq_flags);
    }

    // ...

    public static void main(String[] args) {
        try (var program =
                 BPFProgram.load(FIFOScheduler.class)) {
            // ...
        }
    }
}
```

But what about a slightly more complex scheduler? Can we write a scheduler that is fairer to tasks that don't run for their whole time slice?

Weighted Scheduler

A simple solution to our fairness is prioritizing tasks according to their actual runtime on the CPU While we're at it, we can also factor in the priority (or weight) of a process by scaling the runtime accordingly. This policy is far better fairness-wise than the FIFO policy, but of course, there is still space improvement. There are reasons why proper schedulers are complex. To learn more, I recommend listening to the Tech Over Tea podcast episode with sched-ext developer David Vernet, which you can find on YouTube or Spotify.

Before I show you the scheduler implementation, I want to bring into focus the properties of the `sched_ext_entity` typed `scx` field of type of the `task_struct`, which are valuable for implementing the described policy (adapted from the C source):

```
@Type
class sched_ext_entity {

    // ...

    // priority of the task
    @Unsigned int weight;

    // ...
    /*
     * Runtime budget in nsecs. This is usually set through
     * scx_bpf_dispatch() but can also be modified directly
     * by the BPF scheduler. Automatically decreased by SCX
     * as the task executes. On depletion, a scheduling event
     * is triggered.
     *
     * This value is cleared to zero if the task is preempted
     * by %SCX_KICK_PREEMPT and shouldn't be used to determine
     * how long the task ran. Use p->se.sum_exec_runtime
     * instead.
     */
    @Unsigned long slice;

    /*
     * Used to order tasks when dispatching to the
     * vtime-ordered priority queue of a dsq. This is usually
     * set through scx_bpf_dispatch_vtime() but can also be
     * modified directly by the BPF scheduler. Modifying it
     * while a task is queued on a dsq may mangle the ordering
     * and is not recommended.
     */
    @Unsigned long dsq_vtime;

    // ...

}
```

Back to the scheduler implementation, the new scheduler is not too dissimilar to the FIFO scheduler but makes full use of the `enqueue` and `dispatch` methods:

```
@BPF(license = "GPL")
public abstract class WeightedScheduler extends BPFProgram
                                        implements Scheduler {

    // current vtime
    final GlobalVariable<@Unsigned Long> vtime_now =
            new GlobalVariable<>(0L);

    /*
     * Built-in DSQs such as SCX_DSQ_GLOBAL cannot be used as
     * priority queues (meaning, cannot be dispatched to with
     * scx_bpf_dispatch_vtime()). We therefore create a
     * separate DSQ with ID 0 that we dispatch to and consume
     * from. If scx_simple only supported global FIFO scheduling,
     * then we could just use SCX_DSQ_GLOBAL.
     */
    static final long SHARED_DSQ_ID = 0;

    @BPFFunction
    @AlwaysInline
    boolean isSmaller(@Unsigned long a, @Unsigned long b) {
        return (long)(a - b) < 0;
    }

    @Override
    public int selectCPU(Ptr<task_struct> p, int prev_cpu,
                         long wake_flags) {
        // same as before
        boolean is_idle = false;
        int cpu = scx_bpf_select_cpu_dfl(p, prev_cpu,
                wake_flags, Ptr.of(is_idle));
        if (is_idle) {
            scx_bpf_dispatch(p, SCX_DSQ_LOCAL.value(),
                    SCX_SLICE_DFL.value(),0);
        }
        return cpu;
    }

    @Override
    public void enqueue(Ptr<task_struct> p, long enq_flags) {
        // get the weighted vtime, specified in the stopping
        // method
        @Unsigned long vtime = p.val().scx.dsq_vtime;

        /*
         * Limit the amount of budget that an idling task can
         * accumulate to one slice.
         */
        if (isSmaller(vtime,
                      vtime_now.get() - SCX_SLICE_DFL.value())) {
            vtime = vtime_now.get() - SCX_SLICE_DFL.value();
        }
        /*
         * Dispatch the task to dsq_vtime-ordered priority
         * queue, which prioritizes tasks with smaller vtime
         */
        scx_bpf_dispatch_vtime(p, SHARED_DSQ_ID,
                               SCX_SLICE_DFL.value(), vtime,
                               enq_flags);
    }

    @Override
    public void dispatch(int cpu, Ptr<task_struct> prev) {
        scx_bpf_consume(SHARED_DSQ_ID);
    }

    @Override
    public void running(Ptr<task_struct> p) {
        /*
         * Global vtime always progresses forward as tasks
         * start executing. The test and update can be
         * performed concurrently from multiple CPUs and
         * thus racy. Any error should be contained and
         * temporary. Let's just live with it.
         */
        @Unsigned long vtime = p.val().scx.dsq_vtime;
        if (isSmaller(vtime_now.get(), vtime)) {
            vtime_now.set(vtime);
        }
    }

    @Override
    public void stopping(Ptr<task_struct> p, boolean runnable) {
        /*
         * Scale the execution time by the inverse of the weight
         * and charge.
         *
         * Note that the default yield implementation yields by
         * setting @p->scx.slice to zero and the following would
         * treat the yielding task
         * as if it has consumed all its slice. If this penalizes
         * yielding tasks too much, determine the execution time
         * by taking explicit timestamps instead of depending on
         * @p->scx.slice.
         */
        p.val().scx.dsq_vtime +=
            (SCX_SLICE_DFL.value() - p.val().scx.slice) * 100
            / p.val().scx.weight;
    }

    @Override
    public void enable(Ptr<task_struct> p) {
        /*
         * Set the virtual time to the current vtime, when the task
         * is about to be scheduled for the first time
         */
        p.val().scx.dsq_vtime = vtime_now.get();
    }

    public static void main(String[] args) {
        try (var program =
                 BPFProgram.load(WeightedScheduler.class)) {
            // ...
        }
    }
}
```

I merged the FIFO and the weighted scheduler implementations into the SampleScheduler class, which you can find on GitHub.

Source: https://mostlynerdless.de/blog/2024/09/10/hello-ebpf-writing-a-linux-scheduler-in-java-with-ebpf-15/

III. Discussion on GraalVM-based eBPF development

1) uBPF on GraalVM

Our previous research work

- Please refer to my previous talks and upcoming follow-ups:

1. "GraalVM-based unified runtime for eBPF & Wasm" at GOTC 2021 (Shenzhen), on GraalVM CE Java11-21.2.0:
   - Successfully ported project BPF-Graal-Truffle(https://github.com/mattmurante/bpf-graal-truffle except for AOT part and some test cases.
   - Successfully built uBPF(https://github.com/iovisor/ubpf) to LLVM bitcode and run on GraalVM.
   - Successfully run some basic tests with the official solution GraalWasm(https://github.com/oracle/graal/tree/master/wasm), but failed with the Polyglot case…

Issues

- Most of the uBPF implementations do not support ARM.
- The GraalVM LLVM toolchain does not work as expected for some cases.

III. Discussion on GraalVM-based eBPF development

2. "Revisiting GraalVM-based unified runtime for eBPF & Wasm" at OpenInfra Days China 2021(Beijing), on GraalVM CE Java11-21.3.0-dev:
   - Failed to run project rBPF(https://github.com/qmonnet/rbpf) together with Solana rBPF(https://github.com/solana-labs/rbpf) on GraalVM.
     May need to hack rustc/cargo as the blocking issue is that Rust projects are not built in a finely-grained way for generating the required LLVM bitcode file against each individual source code file.
   - Successfully make wasm3(https://github.com/wasm3/wasm3) to run on GraalVM.

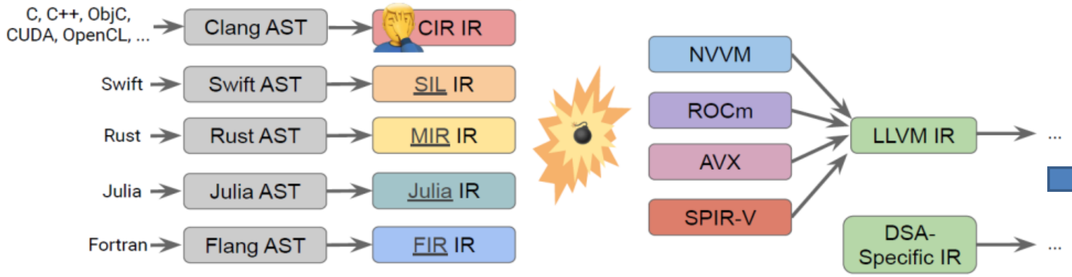
Issues

- Rust projects are not GraalVM friendly.
- Confirmed that there is something wrong with the GraalVM LLVM toolchain from the official GraalVM CE releases, especially the linker.

2) Native support for eBPF in GraalVM?

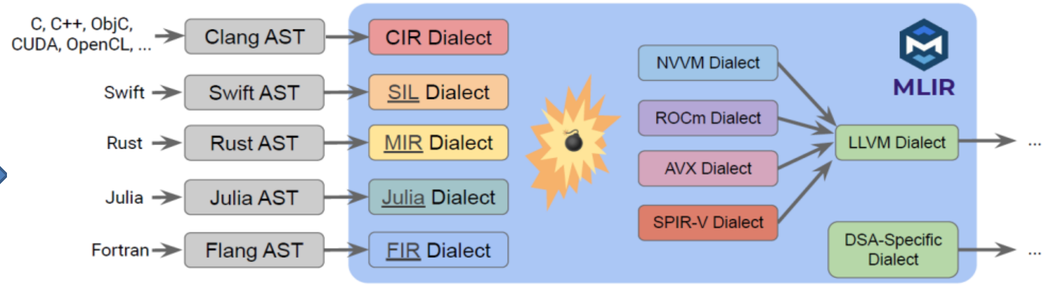
From LLVM to MLIR

- **MLIR: "Meta IR" and Compiler Infrastructure**



- Different back-ends demand customized IR for optimization
- DSAs even cannot use LLVM for generating back-end codes and demand their own IR for code generation

Severe Fragmentation: IRs have different implementations and "frameworks"



MLIR is a **"Meta IR"** and **compiler infrastructure** for:

- Design and implement **dialect**
- Optimization and transform inside of a **dialect**
- Conversion between different **dialects**
- Code generation of **dialect**

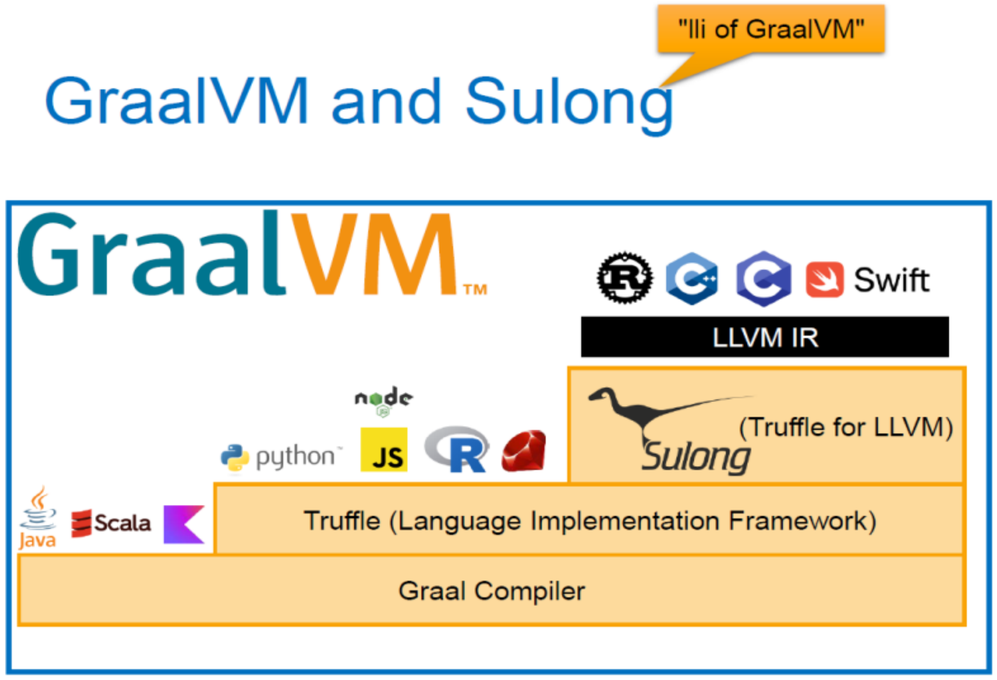
- **But how about a lightweight re-implementation of MLIR/LLVM?**

III. Discussion on GraalVM-based eBPF development

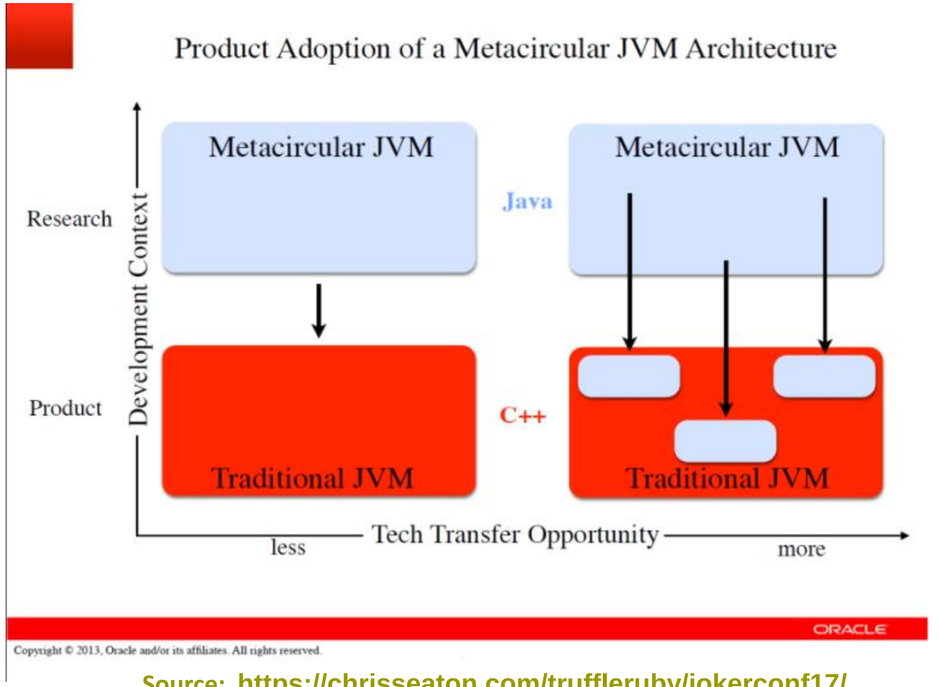
开源社 kaiyuanshe

Rethinking of GraalVM

- One VM to Rule Them All



Source: https://llvm.org/devmtg/2022-05/slides/2022EuroLLVM-ExtendingSulong.pdf



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

- Is it possible to re-implement MLIR/LLVM by GraalVM?

IV. Wrap-up

Ethos:

eBPF is the future of infrastructure

Source: “Building an Open Source Community One Friend at a Time”, Bill Mulligan, FOSDEM 2024.

“ eBPF is a crazy technology, it’s like putting Java~~Script~~ into the Linux kernel

– Brendan Gregg

https://www.facesofopensource.com/brendan-gregg/

Source: “hello-ebpf: Writing eBPF programs directly in Java”, Johannes Bechberger, LPC 2024.

- **A new GraalVM-based DSL for eBPF?**

- **You can look forward to our third-round and perhaps the forth-round discussion of "The eBPF-centric new approach for Hyper-Converged Infrastructure & Edge Computing" that will be divided into two series according to different technology roadmap:**
  - **"ARM + xPython + Rust + Lua + GraalVM + …" and**
  - **"RISC-V + xPython + D + Zig + Lua + SmartRuntime…"**

开源社
kaiyuanshe

开源社
kaiyuanshe

THANKS

Feng Li(李枫)

Indie developer

hkli2013@126.com