

# A Practical Guide to eBPF Project Licensing

How I learned to stop worrying and love the GPL



Jef Spaleta & Bill Mulligan - Isovalent

Disclaimer:

We are **NOT** lawyers

This is **NOT** legal advice

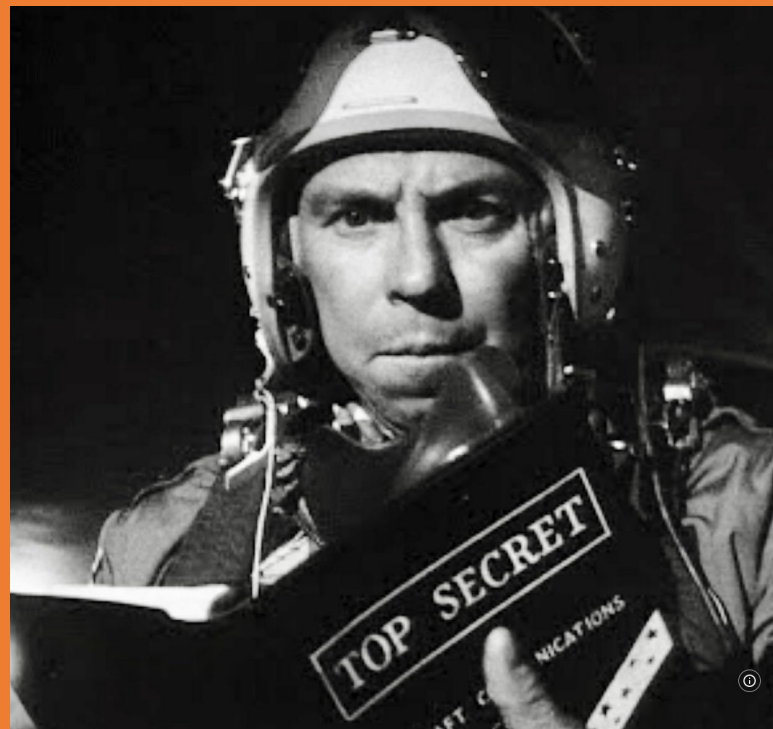
This is a summary of *intent*

# What's in This Talk

- **Motivation: Why eBPF is worth all all the hassle**
- **A quick overview of eBPF licensing considerations**
- **A review of history the "exceptional" relationship between the CNCF and GPL licensing**
- **Practical guidance for**
  - **Choosing your project's eBPF licensing**
  - **Structuring your project code**

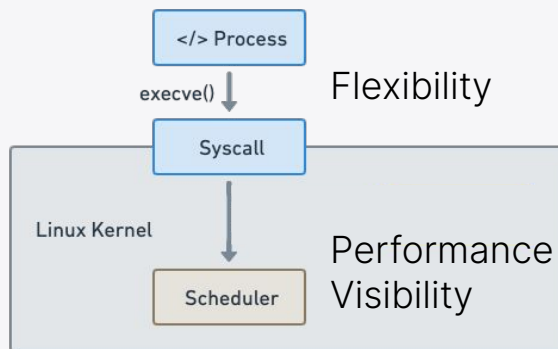
# Motivation

It's no secret  
eBPF is eating the world



# Kernel Space/User Space “Paradox”

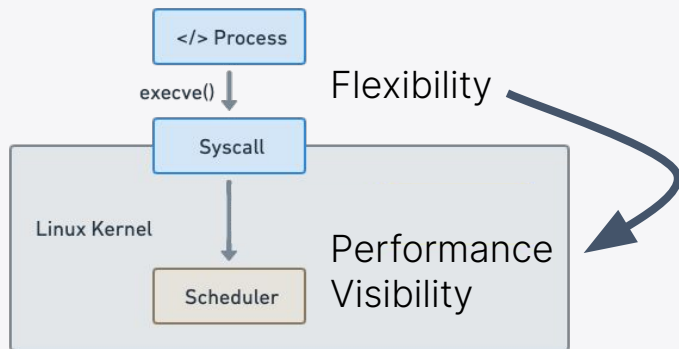
- **Kernel:** System awareness, but lacks flexibility
- **User space:** Programmable, but no direct access to kernel structures, resources
- **Kernel modules:** Difficult, unsafe, not stable



# Kernel Space/User Space “Paradox”

Can we have **programmability** in the kernel?

How can this **benefit cloud native** environments?

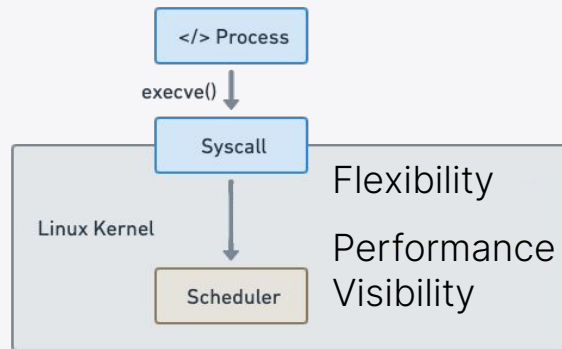


# eBPF in Kernel Space

In the kernel but **flexible**

Safety, performance, observability,  
and programmability

Available by default



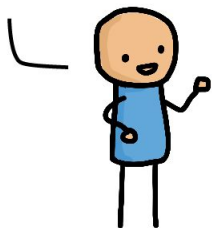
# ISOVALENT

Application Developer:

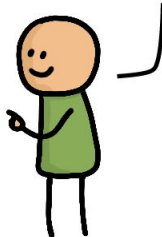
I want this new feature to observe my app



Hey kernel developer! Please add this new feature to the Linux kernel

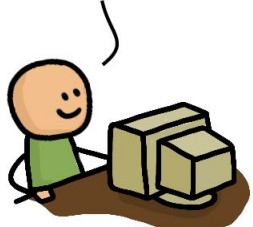


OK! Just give me a year to convince the entire community that this is good for everyone.

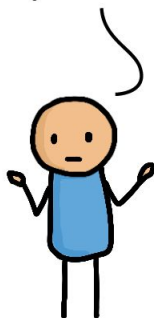


1 year later...

I'm done. The upstream kernel now supports this.



But I need this in my Linux distro

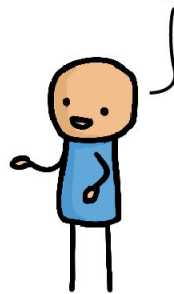


5 years later...

Good news. Our Linux distribution now ships a kernel with your required feature



OK but my requirements have changed since...





# ISOVALENT

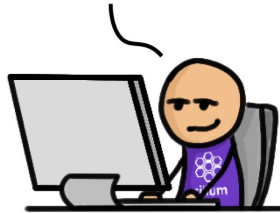
Application Developer:

i want this new feature  
to observe my app



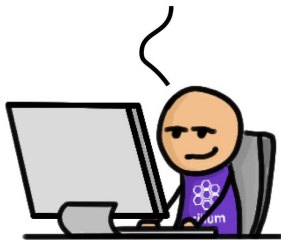
eBPF Developer:

OK! The kernel can't do this so let  
me quickly solve this with eBPF.



A couple of days later...

Here is a release of our eBPF project that has this feature  
now. BTW, you don't have to reboot your machine.





# **“Superpowers for Linux”**

– Brendan Gregg

ISOVALENT

# eBPF is the Cloud Native App Store



# So You Want to Use eBPF in Your Project?



# So You Want to Use eBPF in Your Project

## What you need to provide

- **eBPF bytecode**
  - Runs inside kernel virtual machine
  - Has access to internal kernel structures and functions covered by the GPL via use of helper functions
  - Usually needs to be licensed compatible with the GPLv2\*
- **userspace application**
  - Accesses kernel's syscall API
  - Privileged operation\*
  - May bundle pre-compiled eBPF bytecode payload
  - Licensing? We'll get to that

## What the Linux kernel provides

- **syscall API**
  - Used by application to load eBPF byte code into kernel
  - Used by application to create and access eBPF maps
  - Understood to be the licensing boundary between "userspace" and "kernel"
- **optional eBPF JIT compiler**
  - Translates bytecode into instructions
- **eBPF verifier**
  - Checks for safe execution of bytecode
- **eBPF virtual machine**
- **eBPF helper functions**
  - These impact bytecode licensing

# So You Want to Use eBPF in Your Project

## What you need to provide

- **eBPF bytecode**
  - Runs inside kernel virtual machine
  - Has access to internal kernel structures and functions covered by the GPL via use of helper functions
  - Usually needs to be licensed compatible with the GPLv2\*
- **userspace application**
  - Accesses kernel's syscall API
  - Privileged operation\*
  - May bundle pre-compiled eBPF bytecode payload
  - Licensing? We'll get to that

## What the Linux kernel provides

- **syscall API**
  - Used by application to load eBPF byte code into kernel
  - Used by application to create and access eBPF maps
  - Understood to be the licensing boundary between "userspace" and "kernel"
- **optional eBPF JIT compiler**
  - Translates bytecode into instructions
- **eBPF verifier**
  - Checks for safe execution of bytecode
- **eBPF virtual machine**
- **eBPF helper functions**
  - These impact bytecode licensing

# So You Want to Use eBPF in Your Project

## What you need to provide

- **eBPF bytecode**
  - Runs inside kernel virtual machine
  - Has access to internal kernel structures and functions covered by the GPL via use of helper functions
  - Usually needs to be licensed compatible with the GPLv2\*
- **userspace application**
  - Accesses kernel's syscall API
  - Privileged operation\*
  - May bundle pre-compiled eBPF bytecode payload
  - Licensing? We'll get to that

## What the Linux kernel provides

- **syscall API**
  - Used by application to load eBPF byte code into kernel
  - Used by application to create and access eBPF maps
  - Understood to be the licensing boundary between "userspace" and "kernel"
- **optional eBPF JIT compiler**
  - Translates bytecode into instructions
- **eBPF verifier**
  - Checks for safe execution of bytecode
- **eBPF virtual machine**
- **eBPF helper functions**
  - These impact bytecode licensing

# So You Want to Use eBPF in Your Project

## What you need to provide

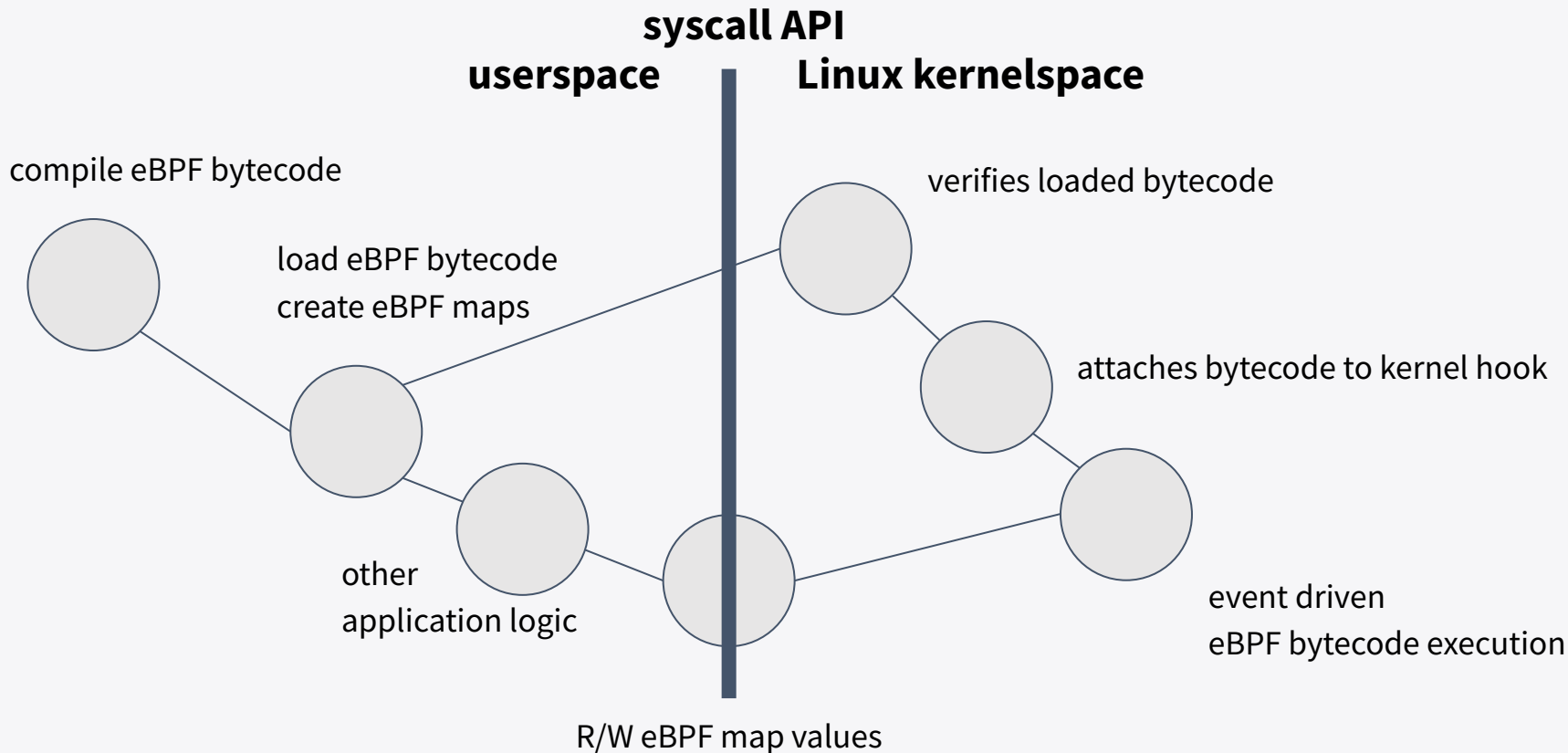
- **eBPF bytecode**
  - Runs inside kernel virtual machine
  - Has access to internal kernel structures and functions covered by the GPL via use of helper functions
  - Usually needs to be licensed compatible with the GPLv2\*
- **userspace application**
  - Accesses kernel's syscall API
  - Privileged operation\*
  - May bundle pre-compiled eBPF bytecode payload
  - Licensing? We'll get to that

## What the Linux kernel provides

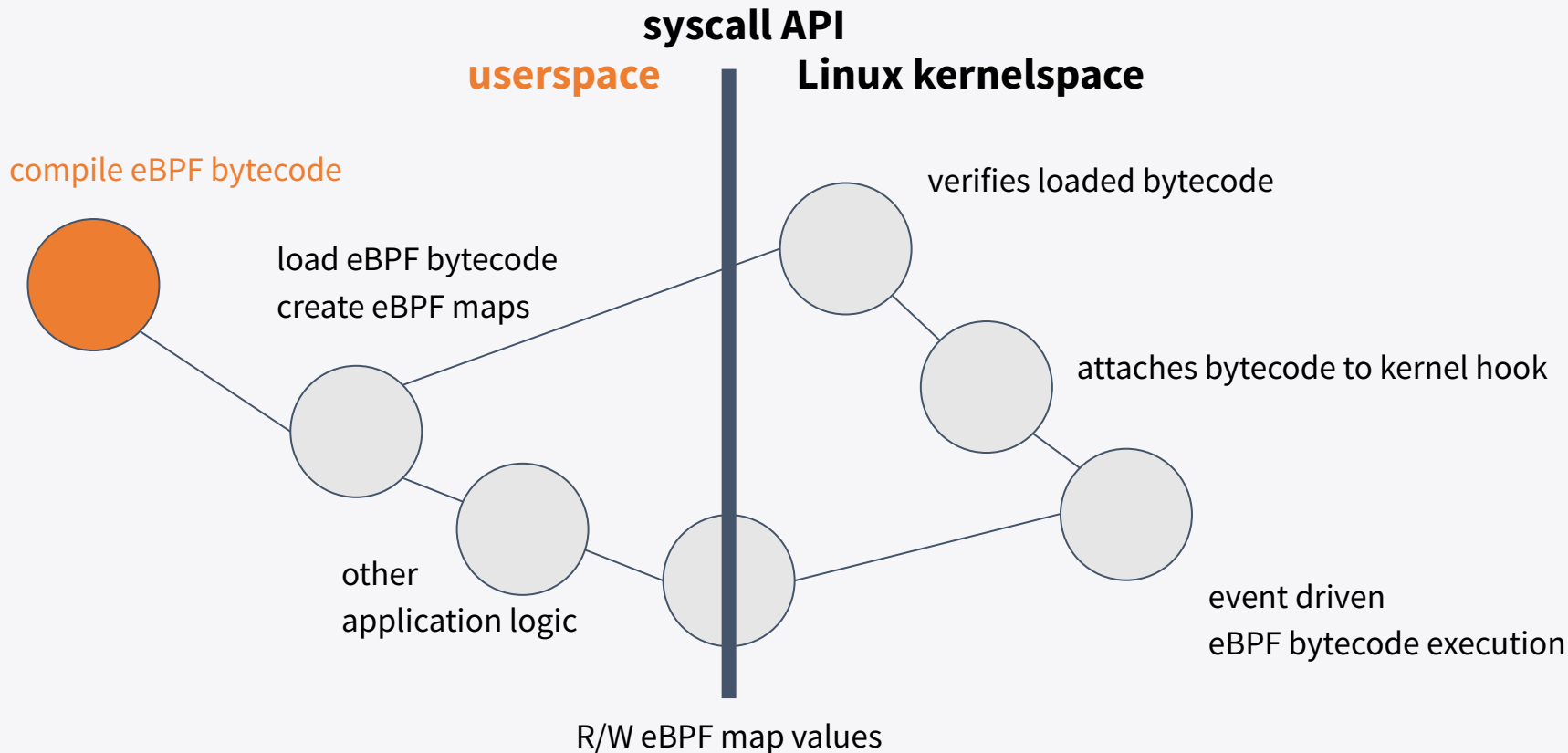
- **syscall API**
  - Used by application to load eBPF byte code into kernel
  - Used by application to create and access eBPF maps
  - Understood to be the licensing boundary between "userspace" and "kernel"
- **optional eBPF JIT compiler**
  - Translates bytecode into instructions
- **eBPF verifier**
  - Checks for safe execution of bytecode
- **eBPF virtual machine**
- **eBPF helper functions**
  - These impact bytecode licensing



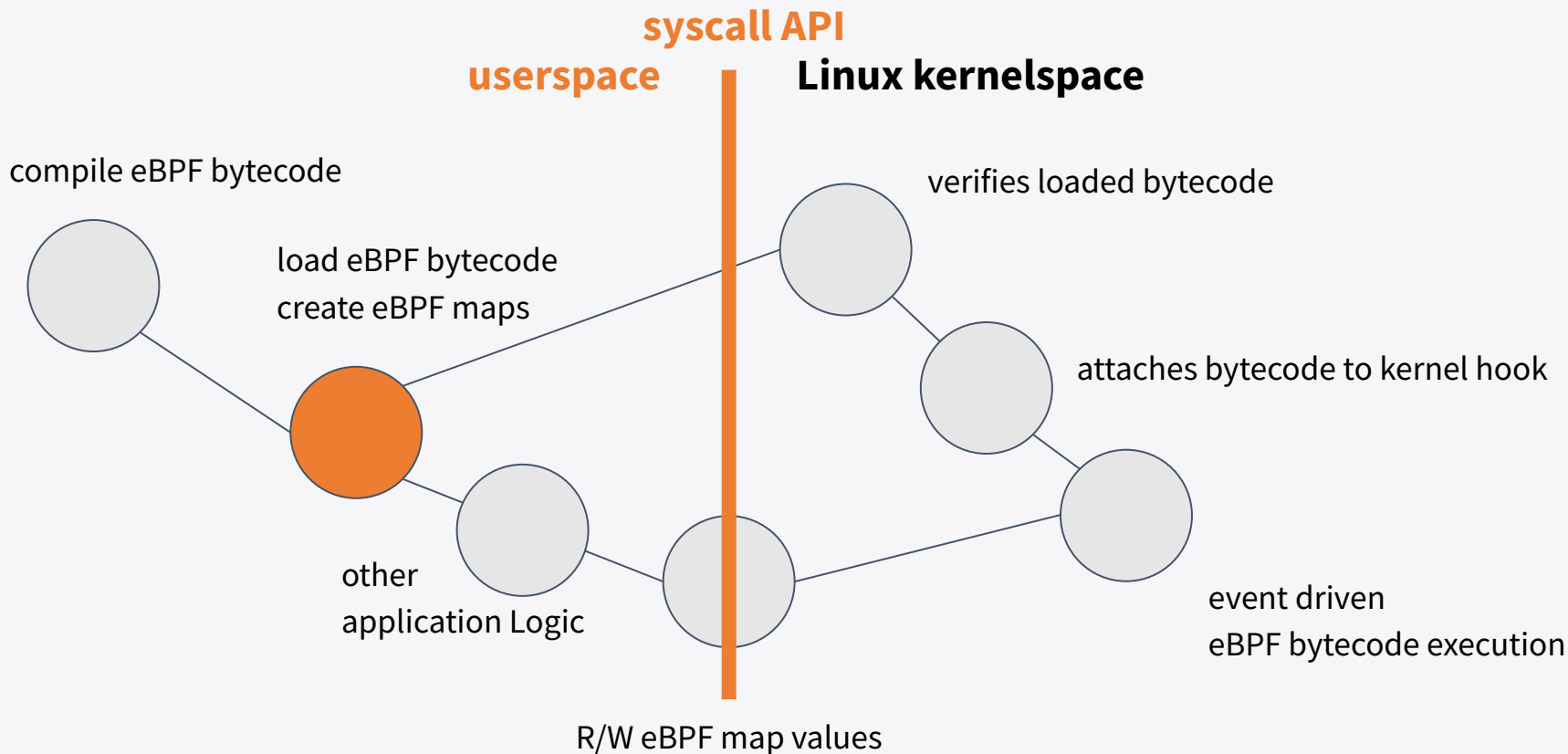
# Simplified eBPF Program Lifecycle



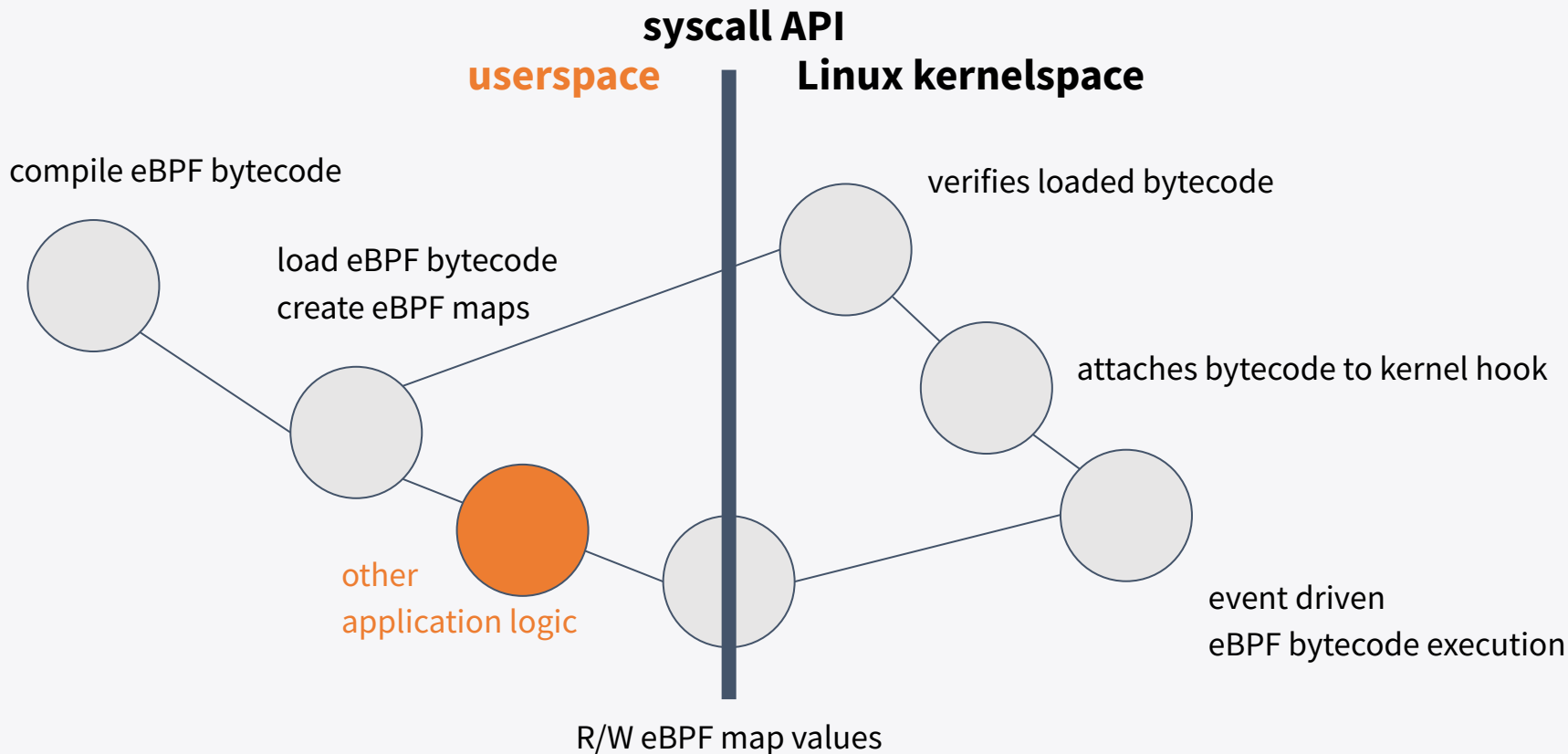
# Simplified eBPF Program Lifecycle



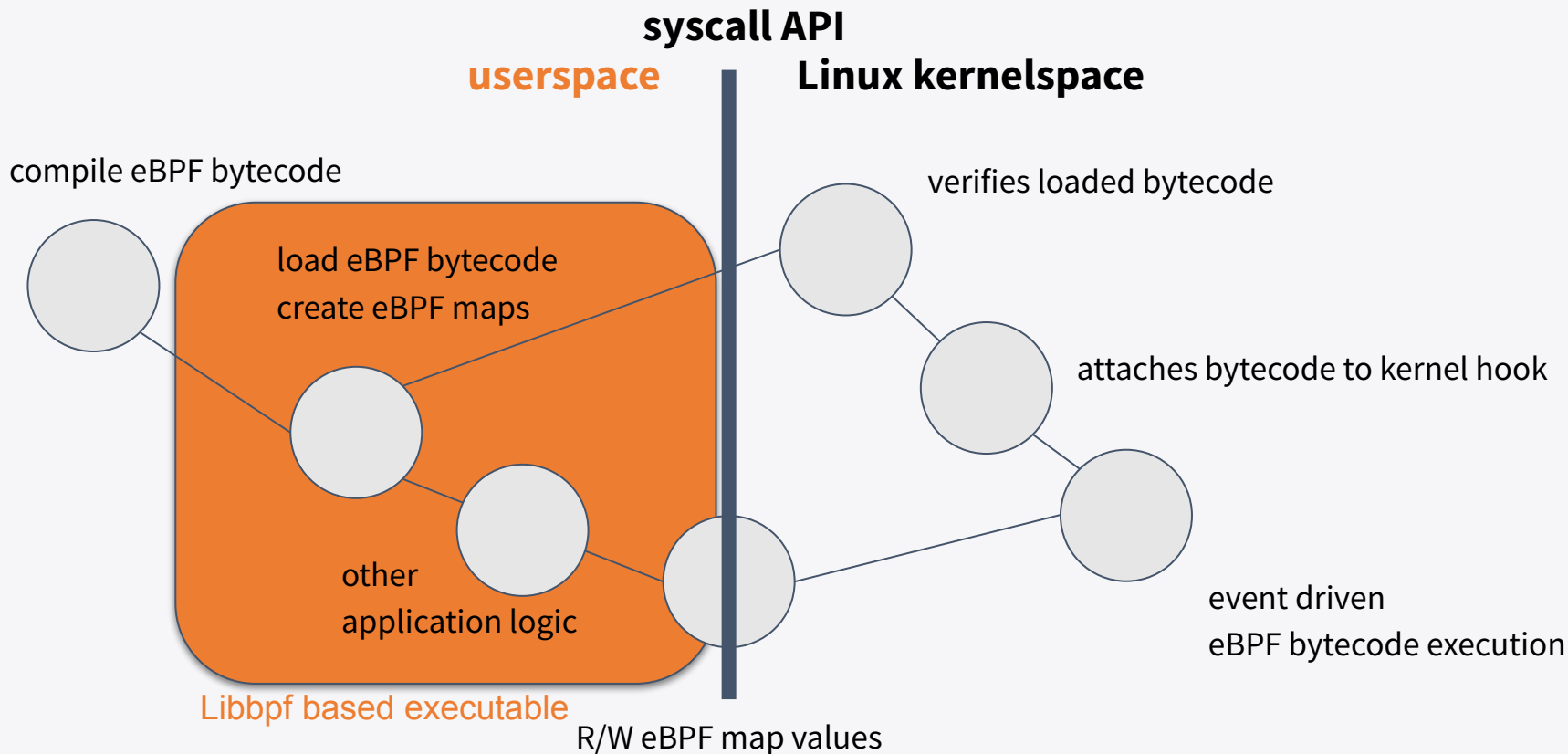
# Simplified eBPF Program Lifecycle



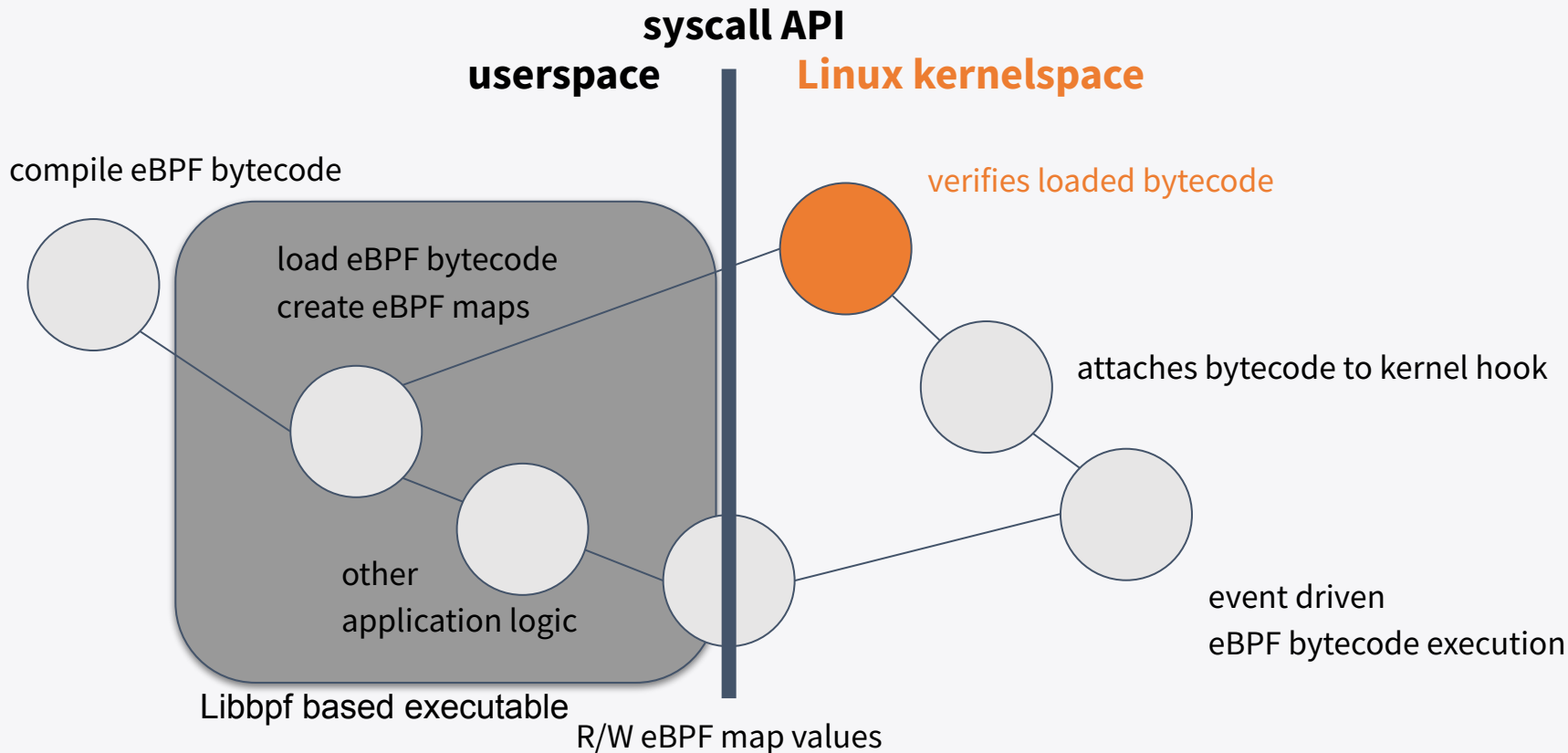
# Simplified eBPF Program Lifecycle



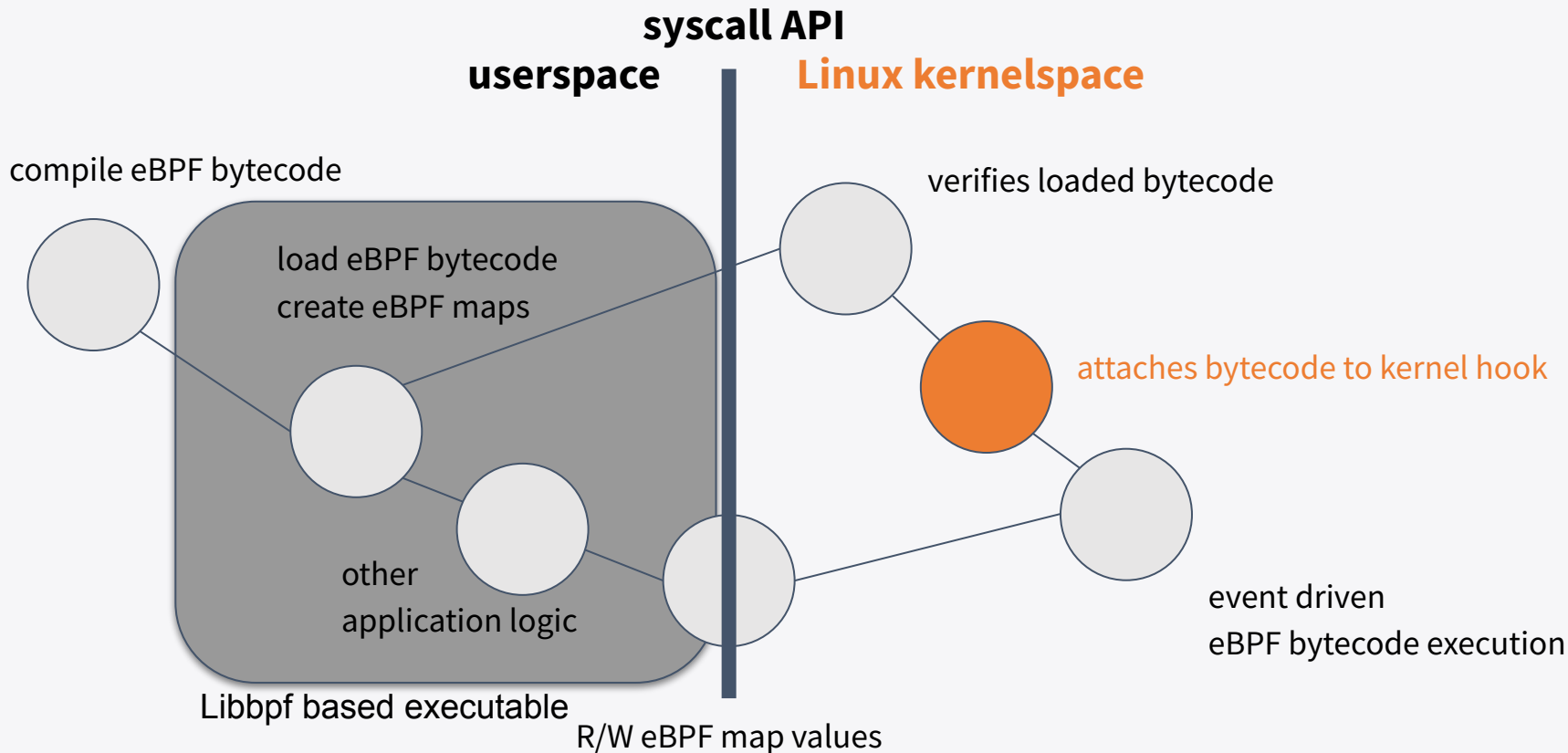
# Simplified eBPF Program Lifecycle



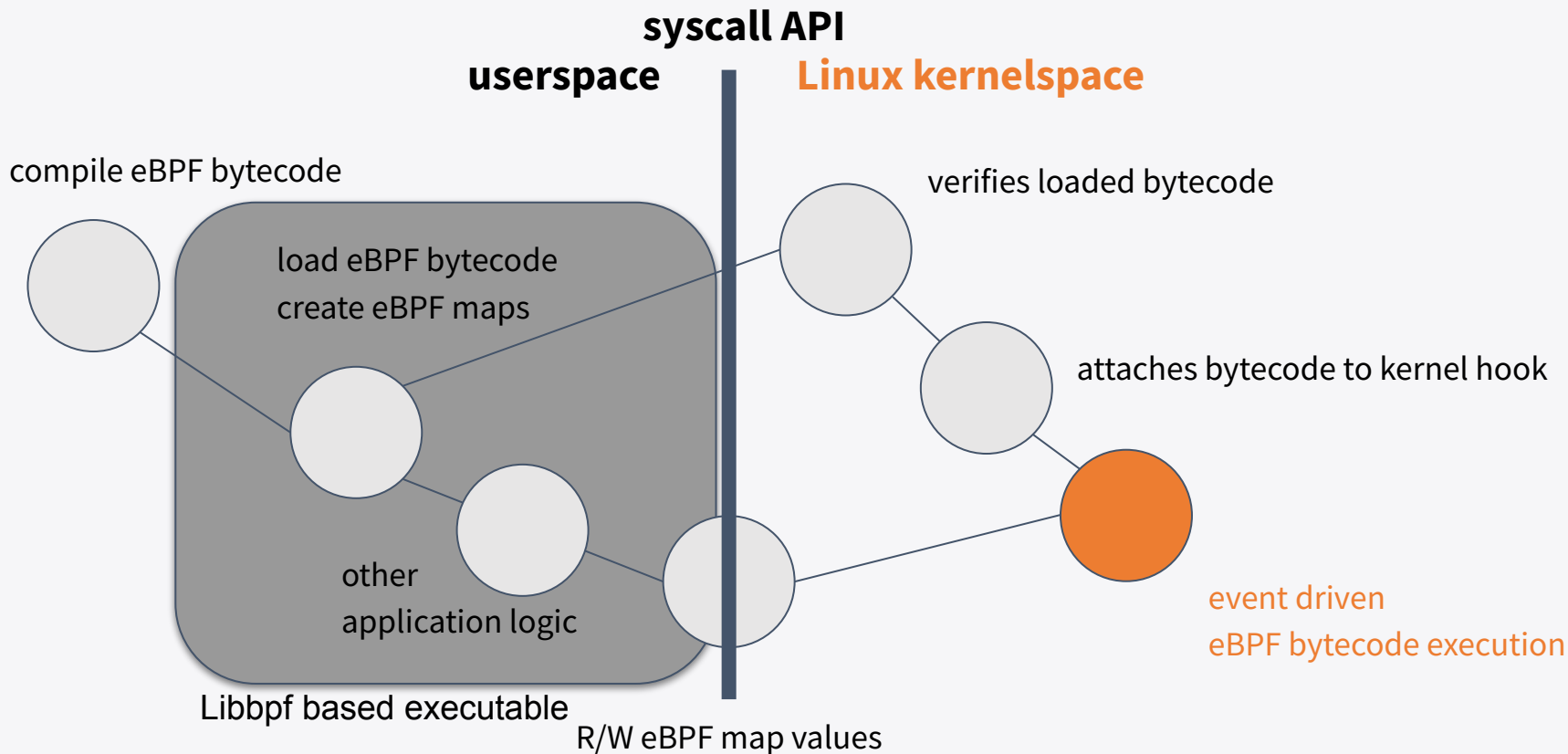
# Simplified eBPF Program Lifecycle



# Simplified eBPF Program Lifecycle

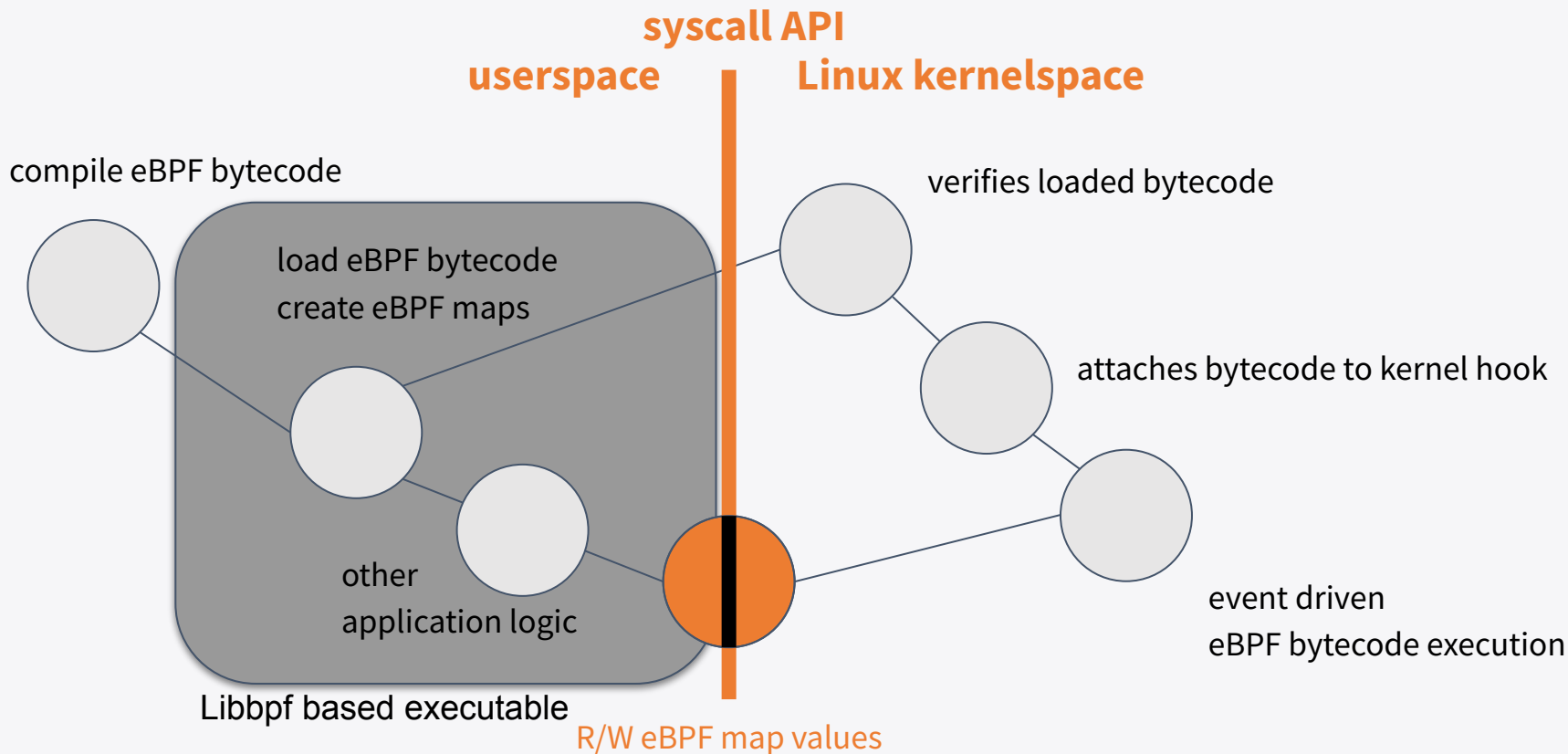


# Simplified eBPF Program Lifecycle





# Simplified eBPF Program Lifecycle



# eBPF Licensing Overview

Why are we even  
talking about the GPL?



# eBPF Is a Key Linux Kernel Technology

The bulk of this talk will be about licensing concerns working specifically with the Linux kernel eBPF implementation.

Because the Linux kernel is GPLv2 licensed, we'll be addressing the licensing concerns related the the Linux kernel's GPLv2 license.

# Do eBPF Programs Need to be GPL Licensed?

The most correct answer: **It depends**

The practical answer: **Yes!**

When an eBPF bytecode is loaded, the Linux kernel checks which helper functions it intends to use. If any function is marked as “GPL only,” the eBPF program has to have a **GPL-compatible license**. The kernel rejects the eBPF program if it does not have a compatible license.

Not all eBPF helper functions are marked as “GPL only”, so it is theoretically possible for some eBPF programs not to need GPL-compatible licensing. However, Alexei Staravoi, co-maintainer of eBPF in the kernel, has stated that **"all meaningful bpf programs are GPL"** because key helper functions are GPL-ed.

# The Licensing Impact of eBPF Helper Functions

One such GPL-ed helper function is `bpf_trace_printk` which is used to generate debug log output.

If a developer needs to add it to a non-GPL-licensed eBPF program for debugging purposes, she would have to change the license to get the kernel to accept the program.

Further, some eBPF program types are required to be GPL compatible even if they don't use "GPL only" helper functions directly, because they implicitly call `EXPORT_SYMBOL_GPL` kernel functions.

It's highly impractical to avoid licensing eBPF bytecode as GPL. Projects need to plan accordingly.

# How to tell the Linux Kernel your eBPF bytecode is GPL-Compatible

If your eBPF program wants to use a “GPL only” kernel function, the eBPF program has to have a GPL-compatible license from the documented licensing scheme (ref: <https://docs.kernel.org/process/license-rules.html#id1>).

The code for establishing this compatibility (`license_is_gpl_compatible()`) is defined in (ref: <https://github.com/torvalds/linux/blob/master/include/linux/license.h>)

eBPF programs use a line of code such as `char _license[] SEC("license") = "Dual BSD/GPL";` to convey the license string to the kernel.

**Note: This license string is not a substitute for the SPDX identifier in the source file which you need to provide for licensing auditing purposes.**

# Which GPL Compatible License to Use?

The generally accepted practise is to use dual license of GPLv2 and a permissive Open Source license such as:

- GPL-2.0 OR MIT
- GPL-2.0 OR BSD-2-Clause
- GPL-2.0 OR Apache-2.0

Dual licensing meets the stated *intent* of allowing proprietary application software that needs to reuse data structures defined in with eBPF sources while still meeting the GPL compliance obligations of the Linux kernel (when the bytecode is used with the Linux kernel).

If your an eBPF based project outside of the CNCF, dual-licensing your eBPF bytecode sources should suffice for you.

If your project is part of the CNCF.....well....it's been complicated, but things have recently changed.

# Do You Need to Distribute the Linux Kernel Sources with your eBPF program?

The most correct answer: **It depends**

The practical answer: **No, but it is nice to do**

Because eBPF bytecode is Compile Once - Run Anywhere (CO-RE) eBPF programs do not have to be compiled against any specific linux kernel binary. The eBPF loader converts the bytecode at runtime into the operational code for the specific kernel version and arch.

**This is a starkly different than linux module binaries, which must be built for specific linux kernel binaries, even if they are licensed as proprietary.**

Projects tend to provide reference Linux sources.



# Stuck Between a Rock a Hard Place

GPL-compatible dual licensing like this hasn't been acceptable for CNCF projects historically

What's been needed is a exception for all in-kernel eBPF programs, to allow flexibility for debugging purposes, and to avoid the overhead of detailed review of the needs of hundreds of individual eBPF programs.

**Don't panic! We'll talk more about the latest CNCF policy changes in the next section**

Let's finish up this section by reviewing the eBPF userspace executable licensing

# The Elephant in the Room: How to License Userspace Executables

Can a non-GPL'd user space executable bundle/load/interact with GPL licensed eBPF bytecode running inside the kernel?

The truest answer: Licensing is complicated. Contact a lawyer.

The practical answer: Yes! The *intent* is for this to be normal, reasonable practice. All the userspace interactions fall under a Linux kernel syscall API exception.

# A Glossary of Terms

The GPLv2 has a lot to say about when certain licensing terms apply. Let's review some important terms that help us better understand the Linux kernel developers' *intent* in relation to the GPLv2 licensing language as it related to software that interacts with the Linux kernel.

**Derived Work:** Modified copies of the original source code. This can include binary compilations of several sources. For example: Compiled eBPF bytecode is a derived work of your eBPF source code and the Linux kernel.

CopyLeft.org has a entire chapter of its GPL Guide dedicated to the finer details on how courts determine derivative status <https://copyleft.org/guide/comprehensive-gpl-guide.html>

## Linking:

Compiled source code can also make reference to externally provided code when executed. Theses external resources are either linked statically at build time or are dynamically linked at runtime. The GPL FAQ states the *intent* is that all forms of linking constitute a derived work under the GPLv2 <https://www.gnu.org/licenses/old-licenses/gpl-2.0-faq.html#LinkingWithGPL>

# A Glossary of Terms

## **Modular:**

Modular programs are pieces of a single application that may execute separately but depend on detailed communication of internal data structures or shared memory.

The GPLv2 FAQ states the *intent* is the entire combined program must be distributed in a GPL compatible manner if any module is GPL licensed

<https://www.gnu.org/licenses/old-licenses/gpl-2.0-faq.html#GPLModuleLicense>

## **Aggregate:**

An aggregate consists of a number of separate programs, distributed together that communicate with each other as if they were independent executables on the system using standard OS provided mechanisms such as pipes or file descriptors.

The GPLv2 FAQ states the *intent* to permit creation and distribution an aggregate, of GPL and non-GPL compatible code.

<https://www.gnu.org/licenses/old-licenses/gpl-2.0-faq.en.html#TOCMereAggregation>

**The line between modular and aggregate can be blurry. But in the case of eBPF the Linux developers have provided clarity via the syscall API licensing exception.**

# A Note About the Linux Kernel Syscall API

The Linux kernel developers have made a very deliberate effort to provide documented *intent* around the issue of what compromises a derived work of the linux kernel and what does not. They have designated a set of system calls as the syscall API that can be called from other programs without those programs being considered a derived work of the kernel.

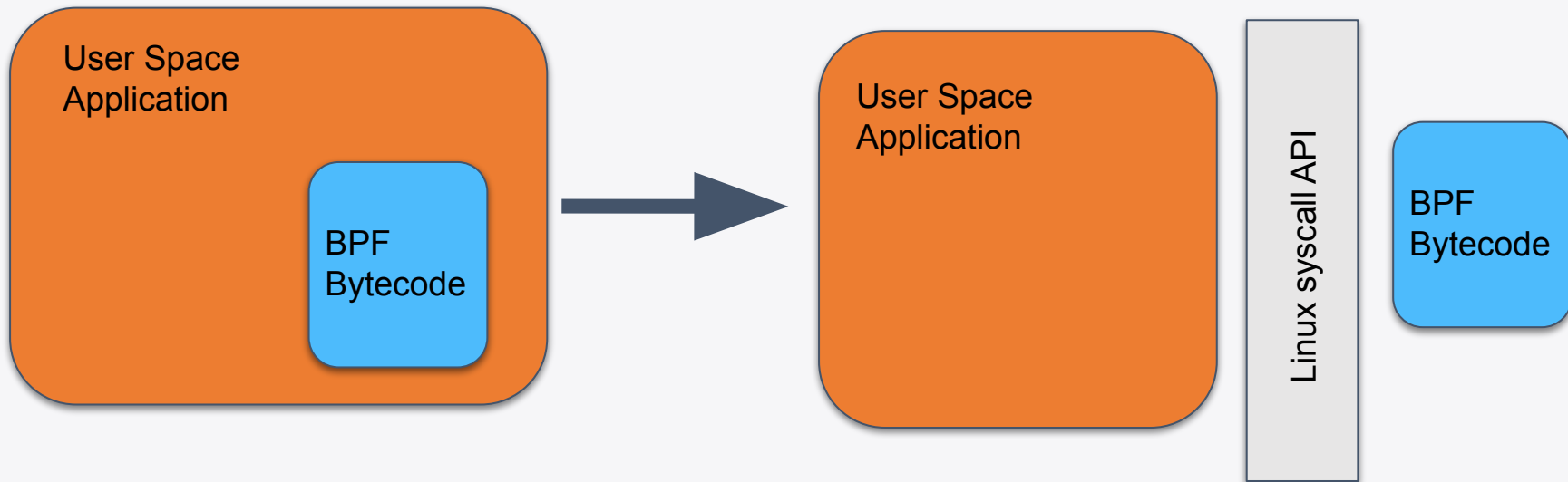
For the syscall API they have formalized a GPL licensing exception note to express that intent as part of the Linux codebase.

<https://github.com/torvalds/linux/blob/master/LICENSES/exceptions/Linux-syscall-note>

**The bpf syscalls used to load eBPF and interact with the eBPF maps are part of the kernel's syscall API.**

# Modular Versus Aggregate

A User-space executable that bundles and eBPF bytecode is generally considered an **aggregate** of the user-space executable and the eBPF bytecode, they are two separate works that interact via the Linux syscall API to get useful work done and can be packaged together without being considered modular parts of a single application derived from the Linux kernel.



# The Linux Developer View of eBPF Aggregates

"**Generally**, proprietary-licensed applications and GPL licensed BPF programs written for the Linux kernel in the same package can co-exist because they are separate executable processes."

ref: [https://docs.kernel.org/bpf/bpf\\_licensing.html](https://docs.kernel.org/bpf/bpf_licensing.html)

# eBPF Maps & Aggregates

The GPLv2 has a lot to say about how modular programs should be licensed and there is no distinct test to determine if a relationship between two executables that interact is an aggregation, or if they are modular parts of a combined whole.

**Does communication between userspace executable and the kernelspace executable via eBPF maps nullify aggregate status?**

**The truest answer: It's complicated. Contact a Lawyer.**

**The practical answer: No, all userspace map operations use the Linux kernel syscall API.**

The documented Linux kernel developer *intent* is that eBPF userspace programs using maps in practice should be thought of as aggregates. All eBPF map operations are covered by the Linux kernels syscall API licensing note.

It becomes a question of your project's *intent* with regard to how you treat the userspace and eBPF bytecode interaction. **Hint: Document your maps.**



# eBPF Maps Are Not Tied to Executables

For truly independent operation of the eBPF userspace and kernelspace executables, it should be possible to manipulate the eBPF maps created by the aggregate using an independently developed userspace executable.

## **bpftool has entered the chat**

bpftool is a general purpose diagnostic utility for interacting with the Linux bpf syscalls from userspace, including the bpf map syscalls.

It can even be used to access and manipulate the maps created by other userspace programs.

Is that enough to demonstrate strict independent operation? Not exactly.

Map data can be opaque. Projects should document their maps as if they were interfaces between different independently developed projects.

# Use BTF to Document Your eBPF Maps

The complexity of the eBPF map objects is primarily controlled by the eBPF kernel developers, not by the userspace application developers for *most* available map types.

The map type you have to be most diligent about is the general purpose `BPF_MAP_TYPE_HASH` Which allows for arbitrary structs for both keys and values.

This flexibility could easily be abused to obfuscate communication between the userspace and kernelspace executables contrary to the *intent* of considering them as independent executables.

It turns out modern Compile Once - Run Everywhere (CO-RE) eBPF applications using libbpf, make use of BPF Type Format (BTF) and gives you a level of map documentation sufficient to debug map structures using bpftool.

# Licensing Recap



## **eBPF userspace application sources:**

License however ( Be nice and document your maps! )

For CNCF projects that means use Apache 2.0 and document your maps!

## **eBPF bytecode sources:**

Dual Licensed with GPL and permissive Open Source license

For CNCF projects...ask for an exception...let's talk about that

ISOVALENT

# The CNCF & The GPL

An exceptional relationship



# The CNCF Charter



## 11. IP Policy

- (c) All new inbound code contributions to the CNCF shall be (i) accompanied by a Developer Certificate of Origin sign-off (<https://developercertificate.org>) and (ii) made under the Apache License, Version 2.0 (available at <https://www.apache.org/licenses/LICENSE-2.0>), such license to be in addition to, and shall not supersede, obligations undertaken under the contribution license agreement(s) provided for in (b) above.
- (d) All outbound code will be made available under the Apache License, Version 2.0.
- (e) All projects evaluated for inclusion in the CNCF shall be completely licensed under an OSI-approved open source license. If the license for a project included in CNCF is not Apache License, Version 2.0, approval of the Governing Board shall be required.
- (g) If an alternative inbound or outbound license is required for compliance with the license for a leveraged open source project or is otherwise required to achieve the CNCF's mission, the Governing Board may approve the use of an alternative license for inbound or outbound contributions on an exception basis.

ref: <https://github.com/cncf/foundation/blob/main/charter.md#11-ip-policy>

Using eBPF in a CNCF project has historically been complicated by the *exceptional* status of the GPL inside CNCF's IP policy

# There's No Stopping Cloud Native eBPF

**Current CNCF projects integrating eBPF features:**

**Cilium, Falco, Blix, Istio, Pixie, Inspektor Gadget,  
and even Kubernetes!**

**How many projects are considering eBPF right now?**

**Many more!**

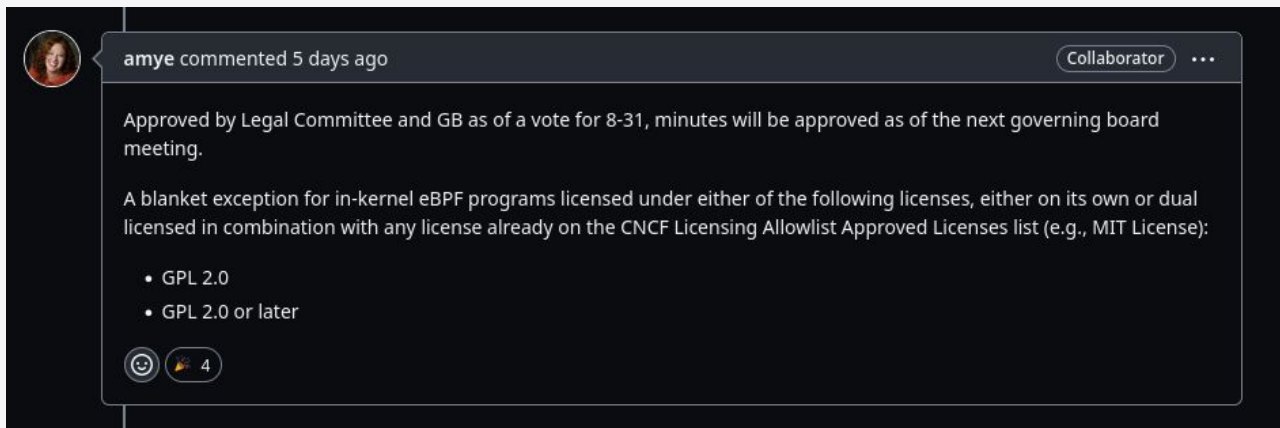
**How many future projects will be using eBPF?**

**All of them?**



# Big News!

## Blanket eBPF Licensing Exception Approved!



Ref: <https://github.com/cncf/foundation/issues/474#issuecomment-1739796978>

Note: This exception applies to the "in-kernel" ebpf source and bytecode. The userspace code should still be Apache 2.0 as per IP Policy



ISOVALENT

# Practical Guidance To Manifest *Intent*

Keep calm and code on



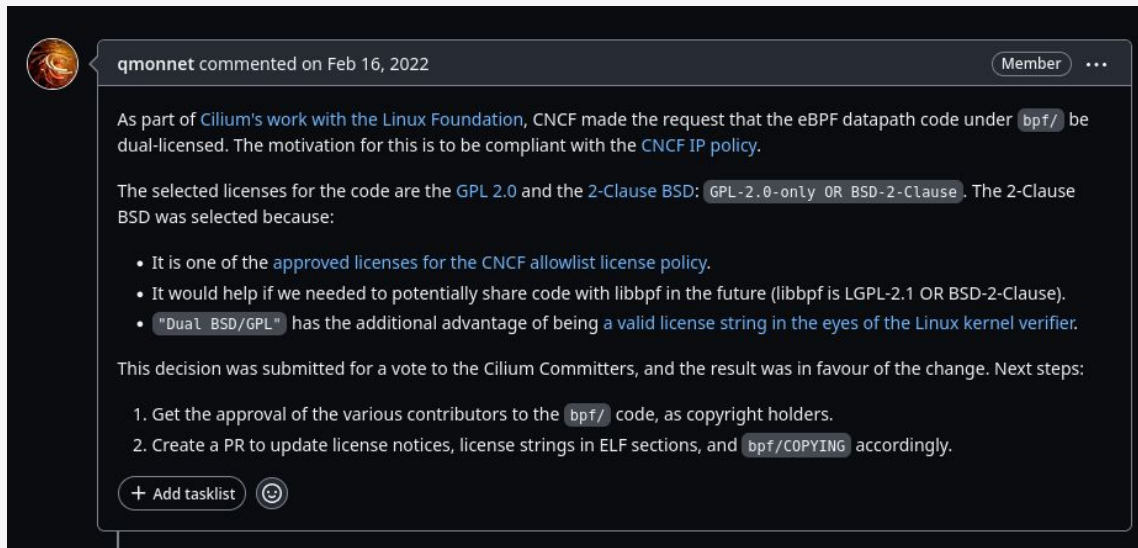
These are *personal opinions* about what you can do to manifest *intent* in your eBPF project to best match the *intent* expressed by the Linux developers.


This isn't legal advice

This isn't even my employer's opinions

This might not even my co-presenter's opinions(sorry Bill)

# We can use the Cilium project as a guide



 qmonnet commented on Feb 16, 2022 Member ...


As part of [Cilium's work with the Linux Foundation](#), CNCF made the request that the eBPF datapath code under `bpf/` be dual-licensed. The motivation for this is to be compliant with the [CNCF IP policy](#).

The selected licenses for the code are the [GPL 2.0](#) and the [2-Clause BSD](#): `GPL-2.0-only OR BSD-2-Clause`. The 2-Clause BSD was selected because:

- It is one of the [approved licenses for the CNCF allowlist license policy](#).
- It would help if we needed to potentially share code with libbpf in the future (libbpf is LGPL-2.1 OR BSD-2-Clause).
- `"Dual BSD/GPL"` has the additional advantage of being a [valid license string in the eyes of the Linux kernel verifier](#).

This decision was submitted for a vote to the Cilium Committers, and the result was in favour of the change. Next steps:

1. Get the approval of the various contributors to the `bpf/` code, as copyright holders.
2. Create a PR to update license notices, license strings in ELF sections, and `bpf/COPYING` accordingly.

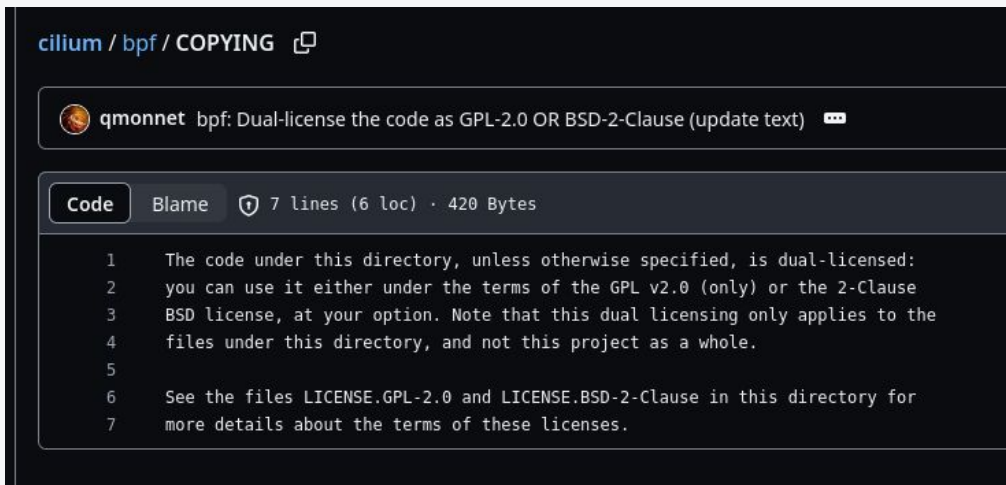
+ Add tasklist 


ref: <https://github.com/cilium/cilium/issues/18823>



# Intentionally Good Practices


Treat the eBPF sources as a separate work from your user space application code:

- Keep the eBPF source code compartmentalized in your project in a dedicated subtree.
  - Manifests *intent* to treat the two executables as an aggregate
- Choose pure GPL\* or permissively dual licensed GPL for eBPF sources.
  - Manifests *intent* to comply with Linux kernel licensing for linking with internal functions
  - Also required for CNCF blanket exception for eBPF use of GPL licensing.



```
cilium / bpf / COPYING 
```

 qmonnet bpf: Dual-license the code as GPL-2.0 OR BSD-2-Clause (update text) 

**Code** Blame  7 lines (6 loc) · 420 Bytes

```
1 The code under this directory, unless otherwise specified, is dual-licensed:
2 you can use it either under the terms of the GPL v2.0 (only) or the 2-Clause
3 BSD license, at your option. Note that this dual licensing only applies to the
4 files under this directory, and not this project as a whole.
5
6 See the files LICENSE.GPL-2.0 and LICENSE.BSD-2-Clause in this directory for
7 more details about the terms of these licenses.
```

# License Housekeeping

- Use SPDX Identifiers headers in sources such as:  
**// SPDX-License-Identifier: (GPL-2.0-only OR BSD-2-Clause)**  
This provides file level licensing auditing

```
▼ bpf/include/bpf/section.h
23 #endif
24
25 #ifndef BPF_LICENSE
26 # define BPF_LICENSE(NAME)
27     char ____license[] __section_license = NAME
28 #endif
29
▼ bpf/bpf_network.c
84     send_trace_notify(ctx, obs_point_to, 0, 0, 0,
85                       ctx->ingress_ifindex, reason, TRACE_PAYLOAD_LEN);
86
87     return ret;
88 }
89
90 BPF_LICENSE("Dual BSD/GPL");
```

```
cilium / bpf / bpf_network.c
Jack-R-lantern bpf: add __section_entry macro for tagging entrypoints
Code Blame 90 lines (75 loc) · 2.57 KB
1 // SPDX-License-Identifier: (GPL-2.0-only OR BSD-2-Clause)
2 /* Copyright Authors of Cilium */
3
4 #include <bpf/ctx/skb.h>
5 #include <bpf/api.h>
6
7 #include <node_config.h>
8 #include <netdev_config.h>
```

- Use BPF licensing section string such as:  
**"Dual BSD/GPL"**  
This is used by the Linux kernel
- SPDX and BPF licensing strings should be consistent in each file to avoid confusion

# Going Above and Beyond

- Use BPF Type Format (BTF) in your eBPF maps to aid in debugging maps using bpftool  
This reinforces the *intent* that the user space program and the ebpf program that are distributed together are an aggregate
- If you are writing a modern libbpf based application, you are already making use of BTF in order to benefit from Compile Once-Run Everywhere (CO-RE) eBPF bytecode
- When using BTF, bpftool in userspace can optional pretty print internal map values for complicated map structures, but it can't yet document contextual meaning of the keys and values.
- Maybe there's a future tooling enhancement opportunity here think about to help generate map documentation from the map specification to further align *intent*.

# Good Practices

Include a reference linux corresponding source

Open Question: Is there coopetition benefit in coordinating around a set of shared reference Linux kernel(s) for all CNCF projects?

ISOVALENT

# Future Considerations

Everything we just told you might  
need to be thrown out the  
Windows™





# Non-Linux eBPF Implementations Are Coming

It's possible to implement eBPF for other operating systems  
(including OSes meant for IoT microcontrollers and of course Windows!)

It's doubtful these alternative OS implementations will require  
GPL-compatible eBPF bytecode

Some of those implementations are going to be relevant to CNCF  
projects (maybe not the microcontroller OS, but Windows for sure)


Can CNCF get ahead of Windows eBPF licensing requirements before  
they start showing up as exceptional requests in projects?

(What licensing scheme allows for re-use of eBPF code for both Linux and Windows?)

ISOVALENT

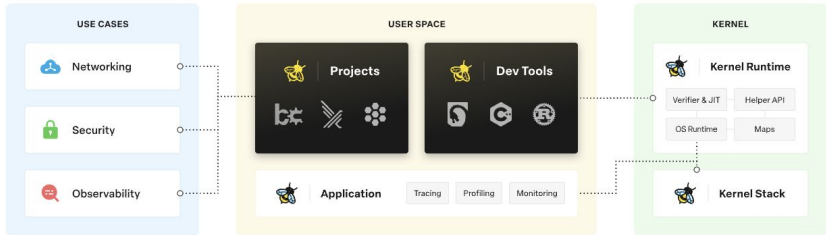
Want to  
Learn More?

ebpf.io


 [Learn](#) [Project Landscape](#) [Events](#) [Community](#) [Blog](#) [Foundation](#) [Eng](#)


## Dynamically program the kernel for efficient networking, observability, tracing, and security


[Project Landscape](#) [What is eBPF](#)




The diagram illustrates the eBPF architecture. It is divided into three main sections: USE CASES, USER SPACE, and KERNEL. USE CASES includes Networking, Security, and Observability. USER SPACE includes Projects, Dev Tools, and Application. Projects and Dev Tools are connected to the Application. The Application is connected to the Kernel. The Kernel includes Kernel Runtime and Kernel Stack. The Kernel Runtime includes Verifier & JIT, Helper API, OS Runtime, and Maps. The Kernel Stack is connected to the Kernel Runtime. The Application is connected to the Kernel Runtime. The Application also includes Tracing, Profiling, and Monitoring. The Application is connected to the Kernel Runtime. The Application is connected to the Kernel Runtime. The Application is connected to the Kernel Runtime.


 Programs are verified to safely execute

 Hook anywhere in the kernel to modify functionality

 JIT compiler for near native execution speed


 Add OS capabilities at runtime

### Organizations in every industry use eBPF in production




Google uses eBPF for security auditing, packet processing, and performance monitoring

VIDEO TALK



Netflix uses eBPF at scale for network insights

BLOG VIDEO



Android uses eBPF to monitor network usage, power, and memory profiling

DOCS

ISOVALENT

# ebpf.io en Français Italiano Português

## Programmation dynamique du noyau pour un trafic réseau, une observabilité, une trace et une sécurité efficaces

Paysage du projet

Qu'est-ce qu'eBPF



✓ Vérification des programmes pour une exécution sécurisée

✓ Branchement n'importe où dans le noyau pour une modification des fonctionnalités

✓ Compilateur JIT pour une vitesse d'exécution quasi native

✓ Accès aux fonctions bas niveau du système

## Des entreprises de tous type d'industries utilisent eBPF en production



Google utilise eBPF pour des audits sécurité, du traitement de paquets ainsi que le monitoring de performance.



Netflix utilise eBPF à grande échelle pour obtenir des informations sur le réseau.



Android utilise eBPF pour surveiller l'utilisation du réseau, pour l'alimentation et le profilage de la mémoire.

ISOVALENT

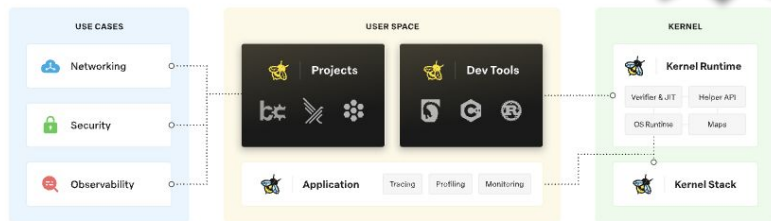
ebpf.io

## Programmation dynamique du noyau pour le réseau, une observabilité, une trace efficaces

Looking for help translating

Paysage du projet

Qu'est-ce qu'eBPF



✓ Vérification des programmes pour une exécution sécurisée

✓ Branchement n'importe où dans le noyau pour une modification des fonctionnalités

✓ Compilateur JIT pour une vitesse d'exécution quasi native

✓ Accès aux fonctions bas niveau du système

## Des entreprises de tous type d'industries utilisent eBPF en production

Google

Google utilise eBPF pour des audits sécurité, du traitement de paquets ainsi que le monitoring de performance.

NETFLIX

Netflix utilise eBPF à grande échelle pour obtenir des informations sur le réseau.

android

Android utilise eBPF pour surveiller l'utilisation du réseau, pour l'alimentation et le profilage de la mémoire.

ISOVALENT

## Book Signing at 1:30



ISOVALENT

# Thank you



ebpf.io



@jspaleta

@breakawaybilly



ebpf.io

Download from  
[isovalent.com](https://isovalent.com)

