

ISOVALENT

Who has an
Android phone?

Raise your 🙋



ISOVALENT

Is eBPF everywhere?



ISOVALENT



FOSDEM
Feb 1, 2025

An intro to **eBPF** with Go: The Foundation of Modern Kubernetes Networking



Donia Chaiehloudj | @doniacld

Software Engineer, Isovalent at Cisco

Co-author of the book "Learn Go with Pocket-Sized Projects"

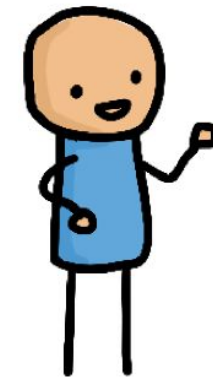
Without eBPF

Application Developer:

i want this new feature
to observe my app



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

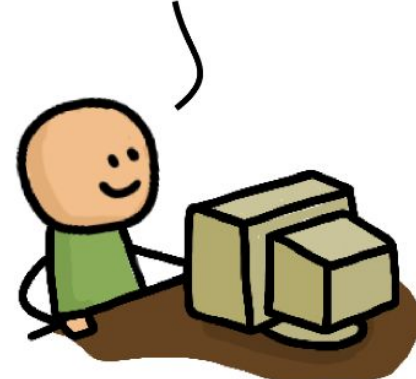


OK! Just give me a year to convince
the entire community that this is
good for everyone.

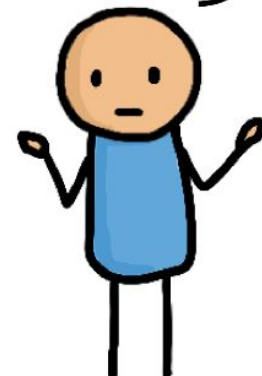


1 year later...

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



But i need this in
my Linux distro



5 year later...

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



OK but my requirements
have changed since...



What is  eBPF ?

Makes the Linux kernel
programmable

With eBPF

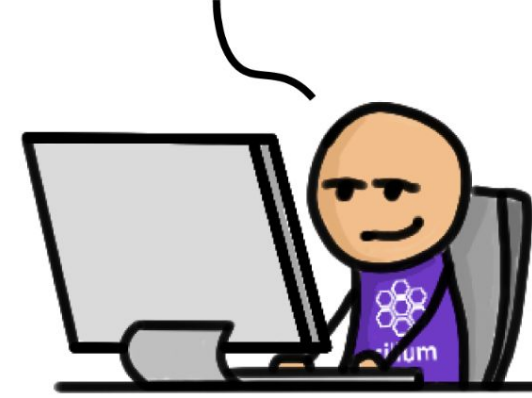
Application Developer:

I want this new feature to observe my app



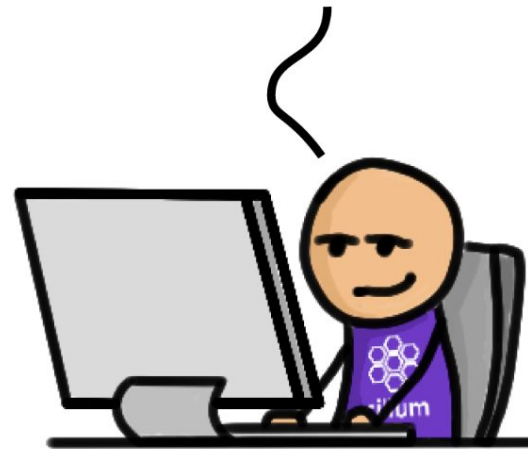
eBPF Developer:

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

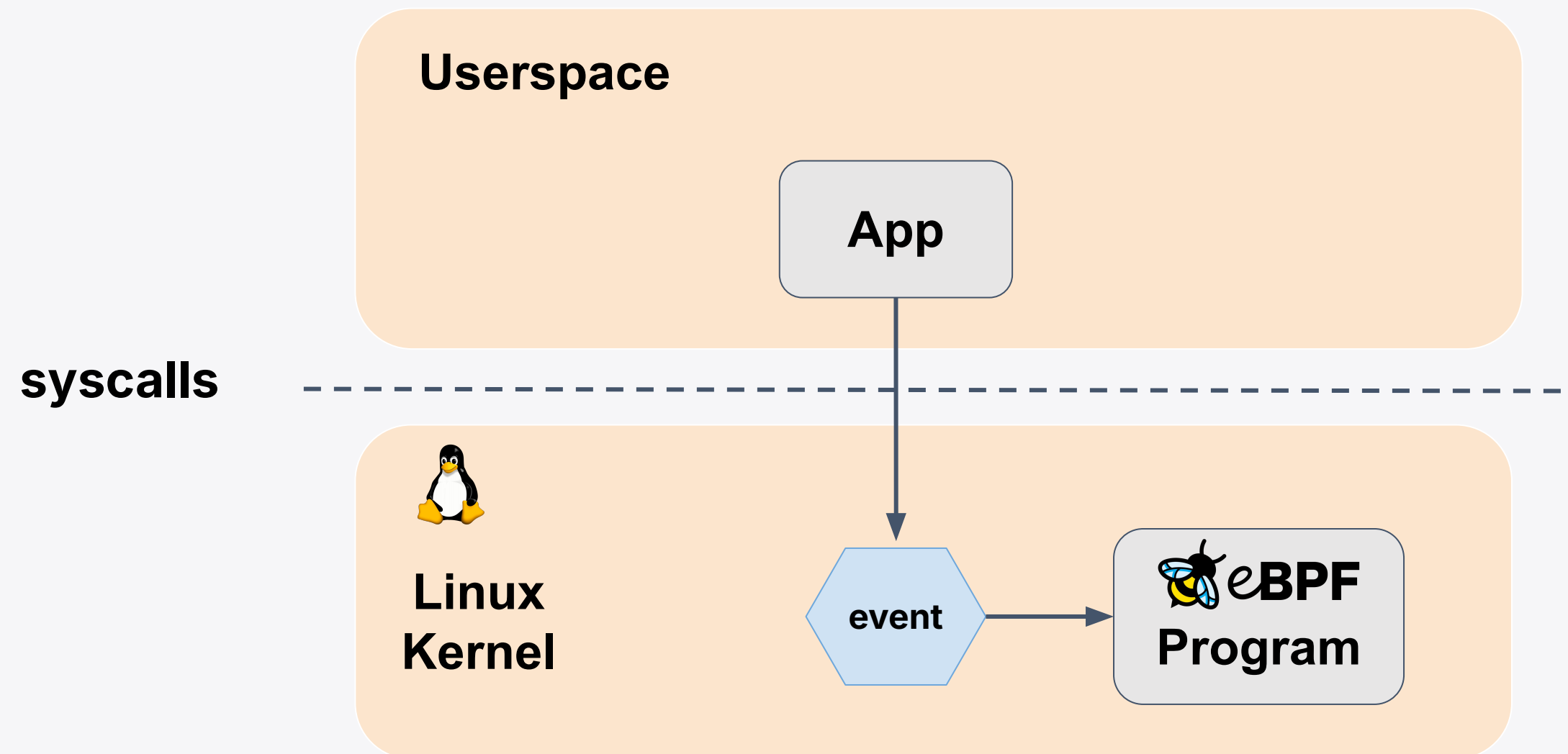


A couple of days later...

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



Overview of eBPF



Open File Tracing

<https://github.com/doniacld/ebpf-4-gophers>

Example: Trace when files are open

```
#include <linux/bpf.h>           // BPF header for definitions.
#include <bpf/bpf_helpers.h>      // BPF helper functions.

// Attach to sys_enter_openat tracepoint.
SEC("tracepoint/syscalls/sys_enter_openat")
int trace_openat(struct trace_event_raw_sys_enter* ctx) {
    char msg[] = "File opening..."; // Message to log.
    bpf_trace_printk(msg, sizeof(msg)); // Log message to trace pipe.

    return 0;
}

char LICENSE[] SEC("license") = "GPL";
```

Example: Trace when files are open

```
#include <linux/bpf.h>           // BPF header for definitions.
#include <bpf/bpf_helpers.h>      // BPF helper functions.

// Attach to sys_enter_openat tracepoint.
SEC("tracepoint/syscalls/sys_enter_openat")
int trace_openat(struct trace_event_raw_sys_enter* ctx) {
    char msg[] = "File opening..."; // Message to log.
    bpf_trace_printk(msg, sizeof(msg)); // Log message to trace pipe.

    return 0;
}

char LICENSE[] SEC("license") = "GPL";
```

Example: Trace when files are open

```
#include <linux/bpf.h>           // BPF header for definitions.
#include <bpf/bpf_helpers.h>      // BPF helper functions.

// Attach to sys_enter_openat tracepoint.
SEC("tracepoint/syscalls/sys_enter_openat")
int trace_openat(struct trace_event_raw_sys_enter* ctx) {
    char msg[] = "File opening..."; // Message to log.
    bpf_trace_printk(msg, sizeof(msg)); // Log message to trace pipe.

    return 0;
}

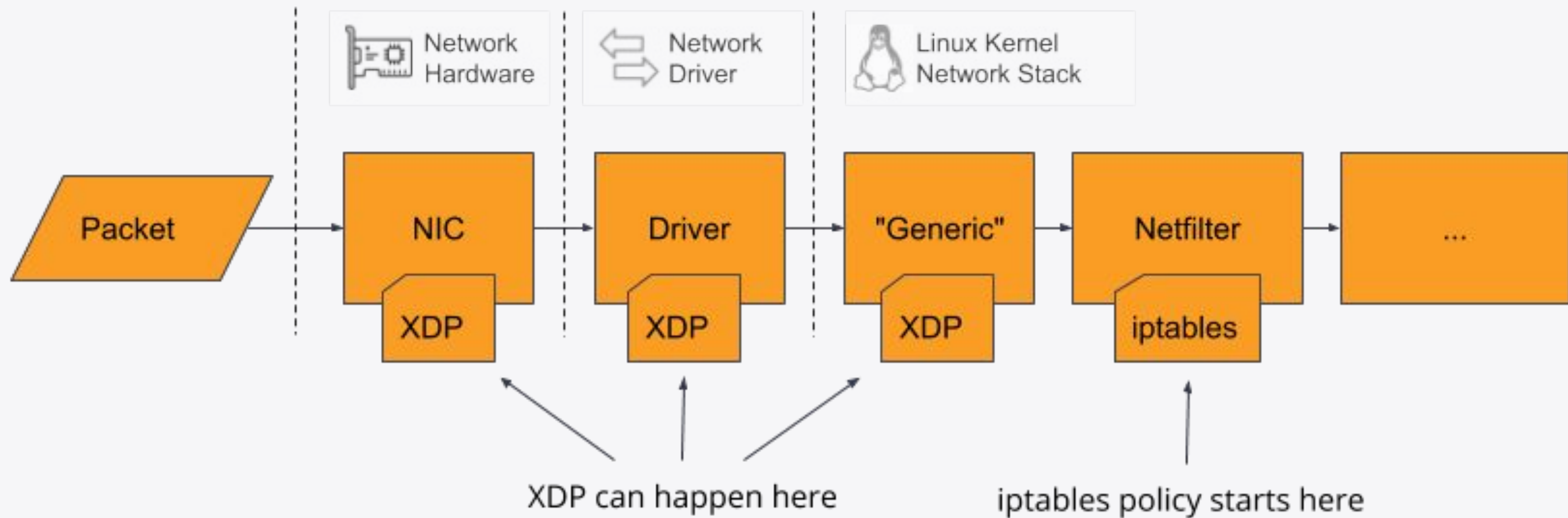
char LICENSE[] SEC("license") = "GPL";
```

```
$ sudo cat /sys/kernel/debug/tracing/trace_pipe
systemd-journal-326 [000] ...21 53765.037027: bpf_trace_printk: File opening...
systemd-journal-326 [000] ...21 53765.037048: bpf_trace_printk: File opening...
      cat-35252 [000] ...21 53765.037948: bpf_trace_printk: File opening...
      cat-35252 [000] ...21 53765.037966: bpf_trace_printk: File opening...
      cat-35252 [000] ...21 53765.038107: bpf_trace_printk: File opening...
```

process id

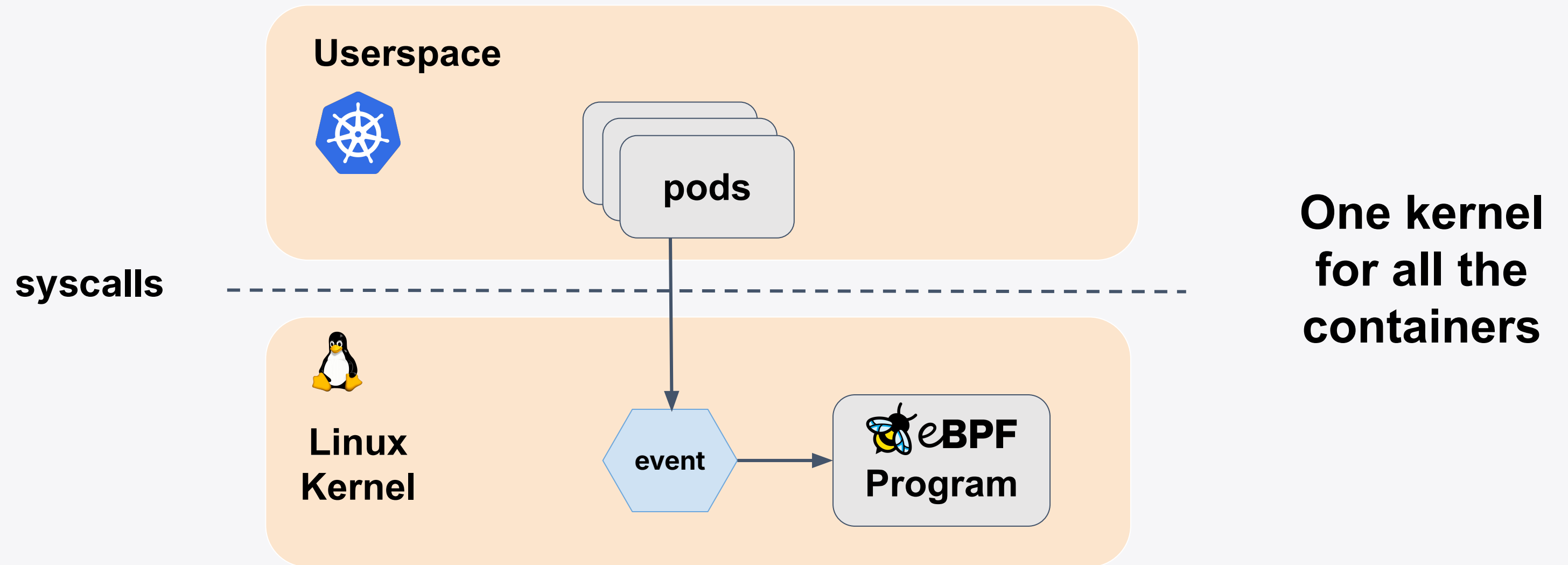
Tracing Demo

XDP hooks power

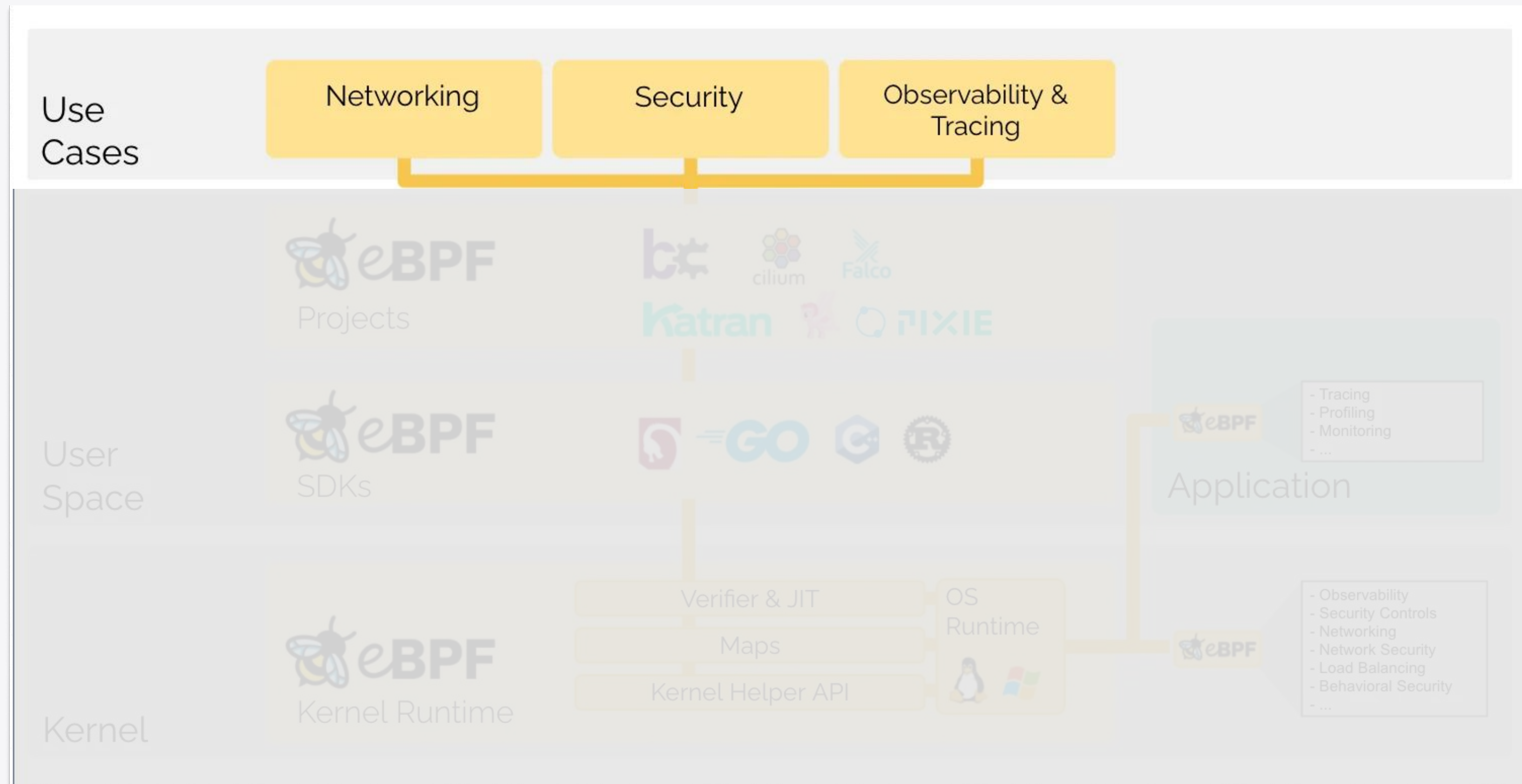


<https://blog.ippon.tech/ebpf-and-kernel-modules-whats-the-difference>

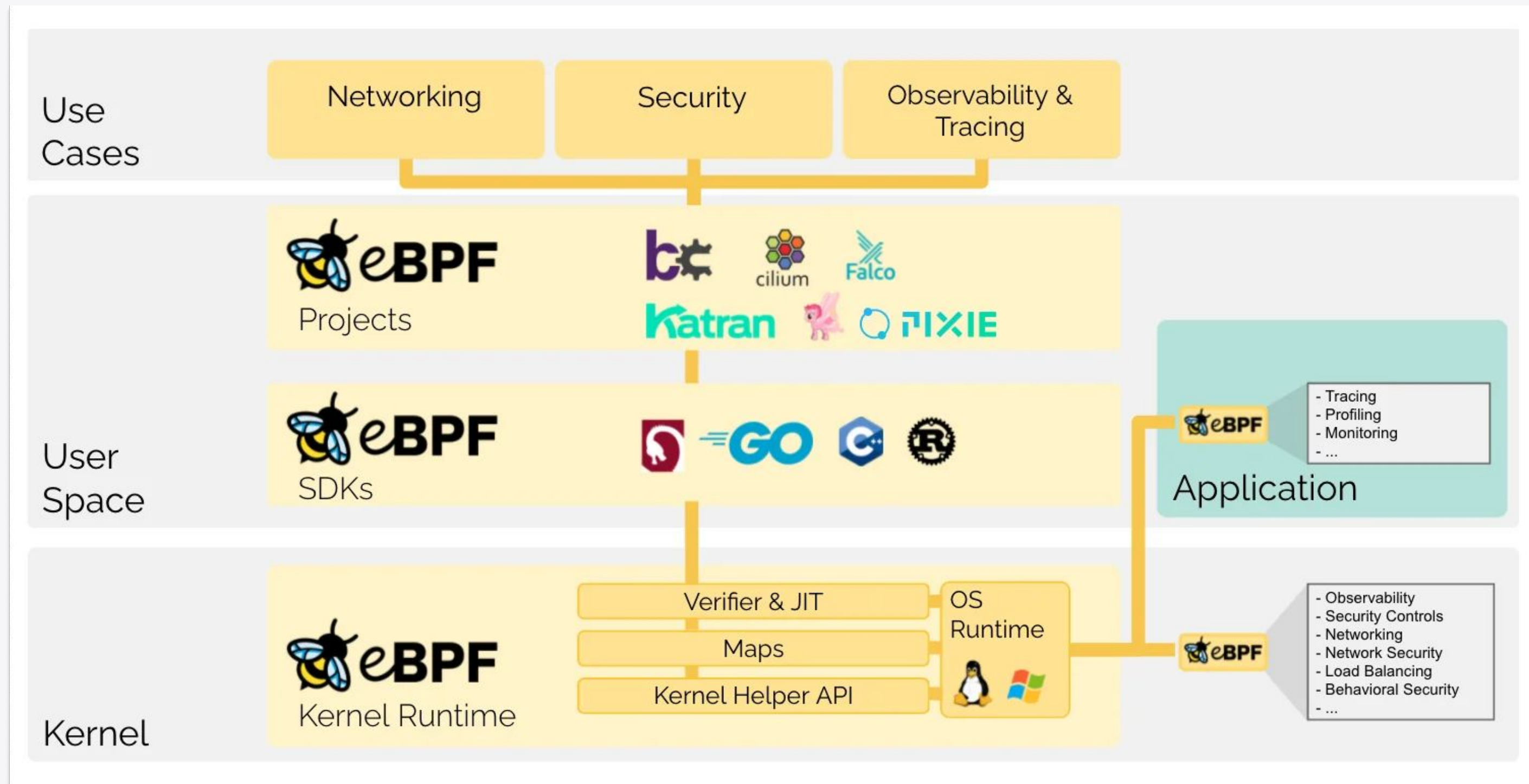
Why eBPF for Kubernetes?



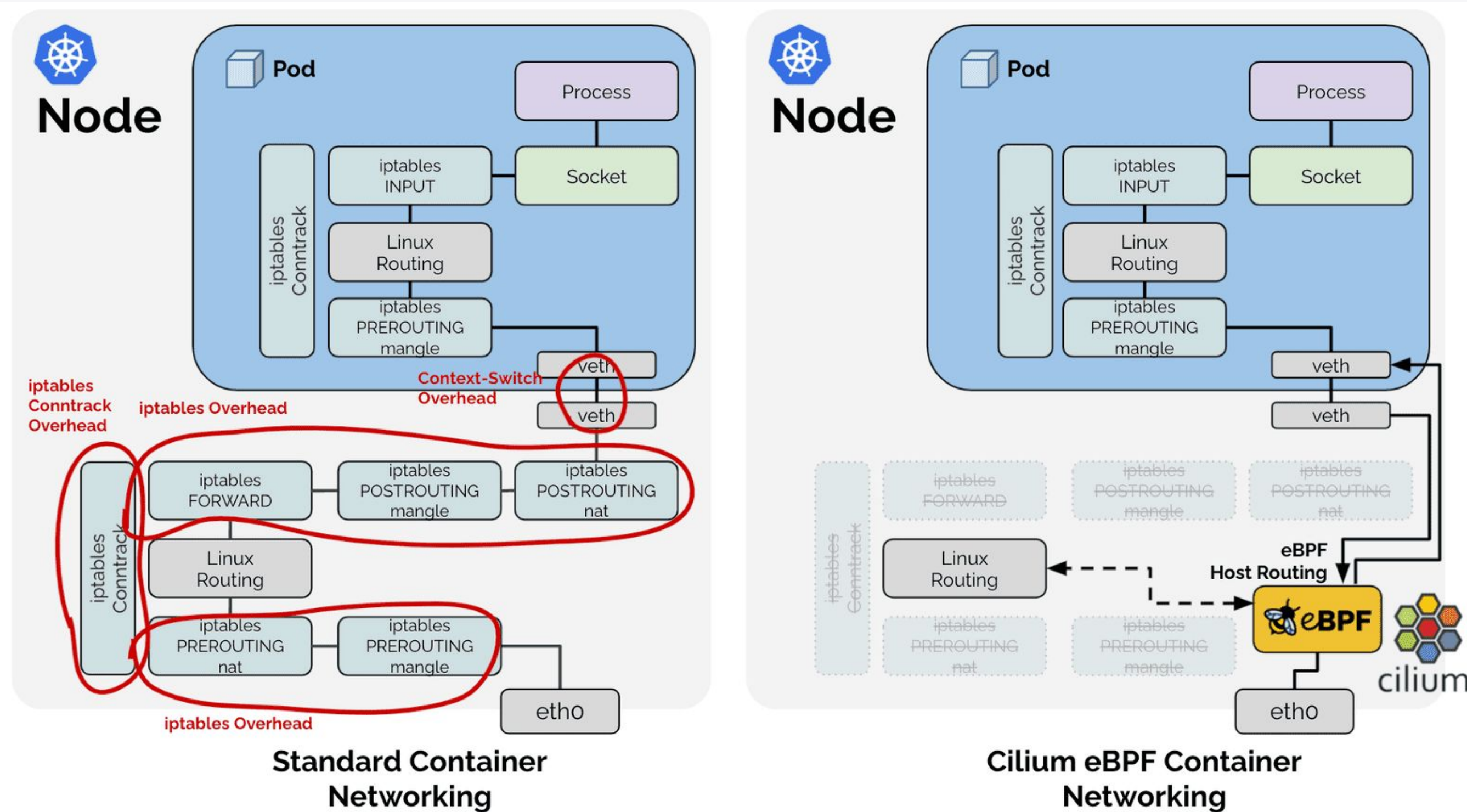
Why eBPF for Kubernetes?



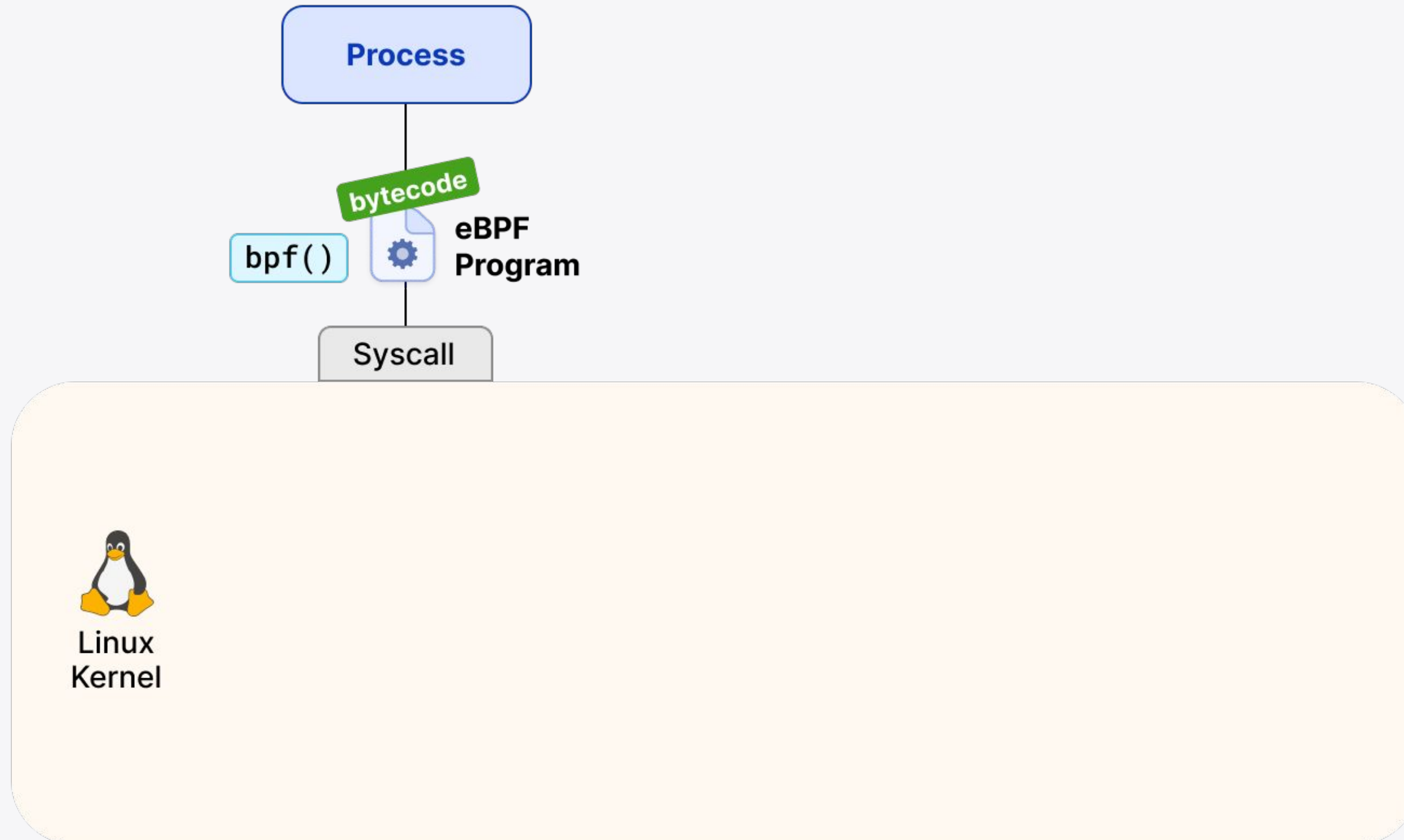
Why eBPF for Kubernetes?



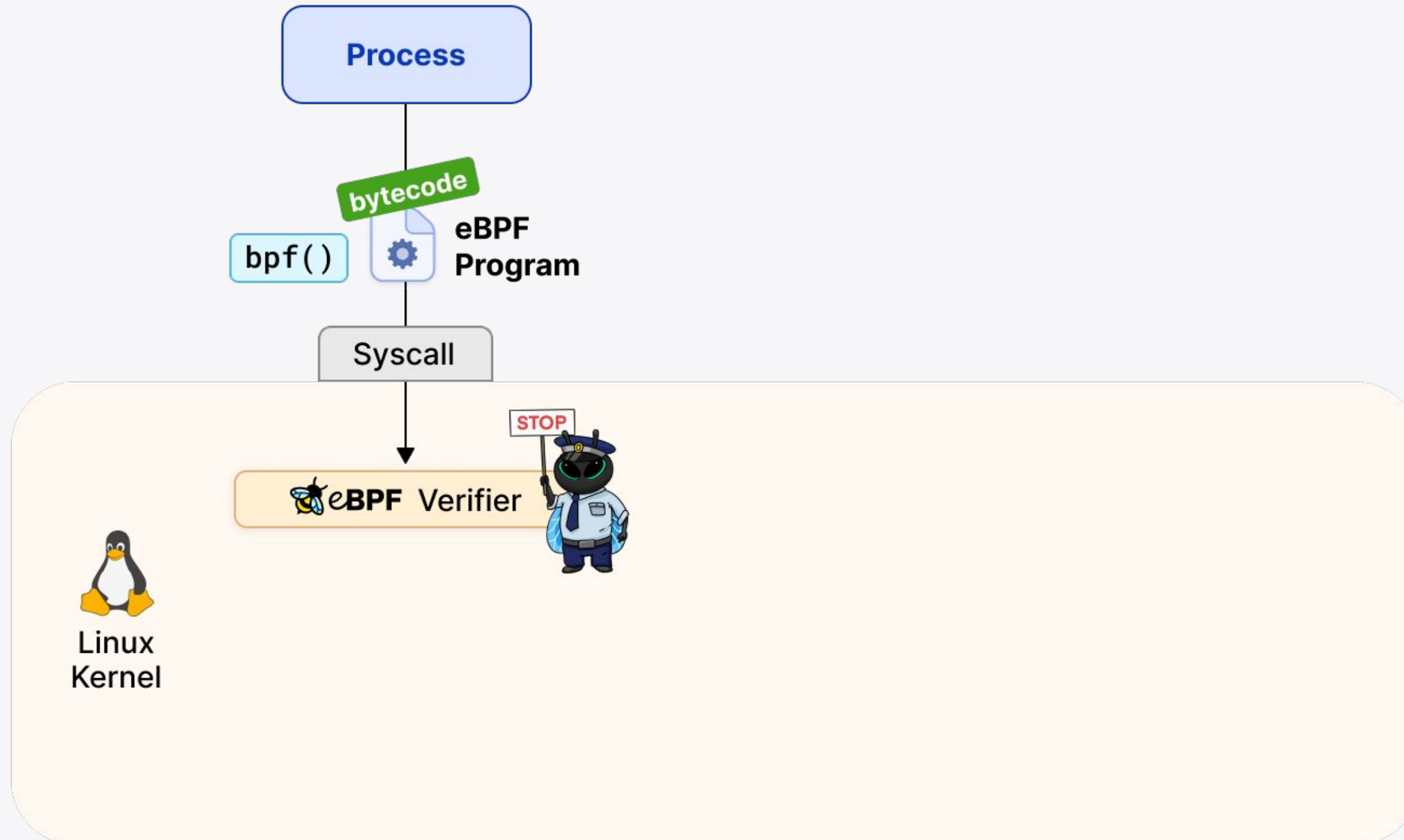
Efficient Networking: Cilium use of eBPF



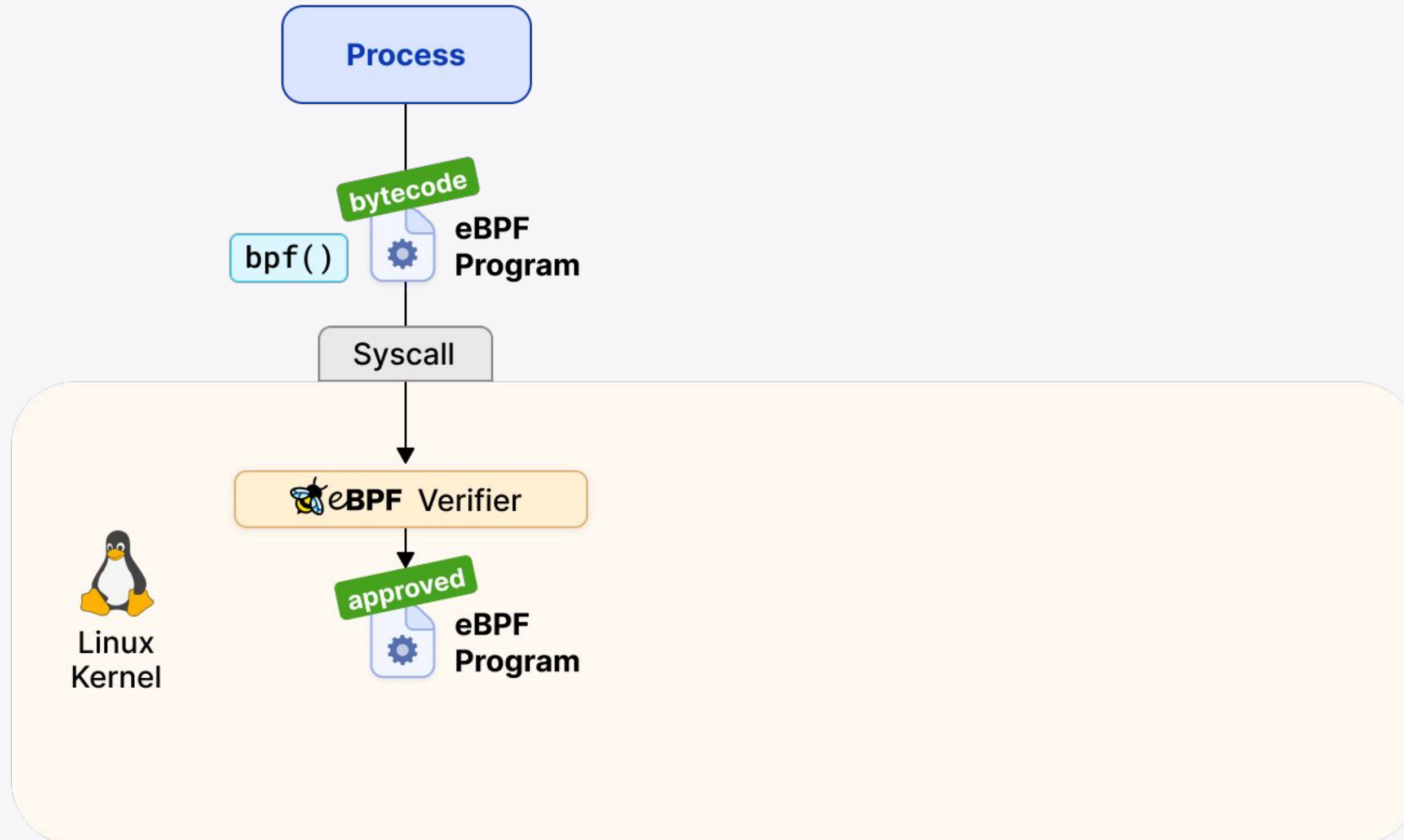
How eBPF works under the hood?



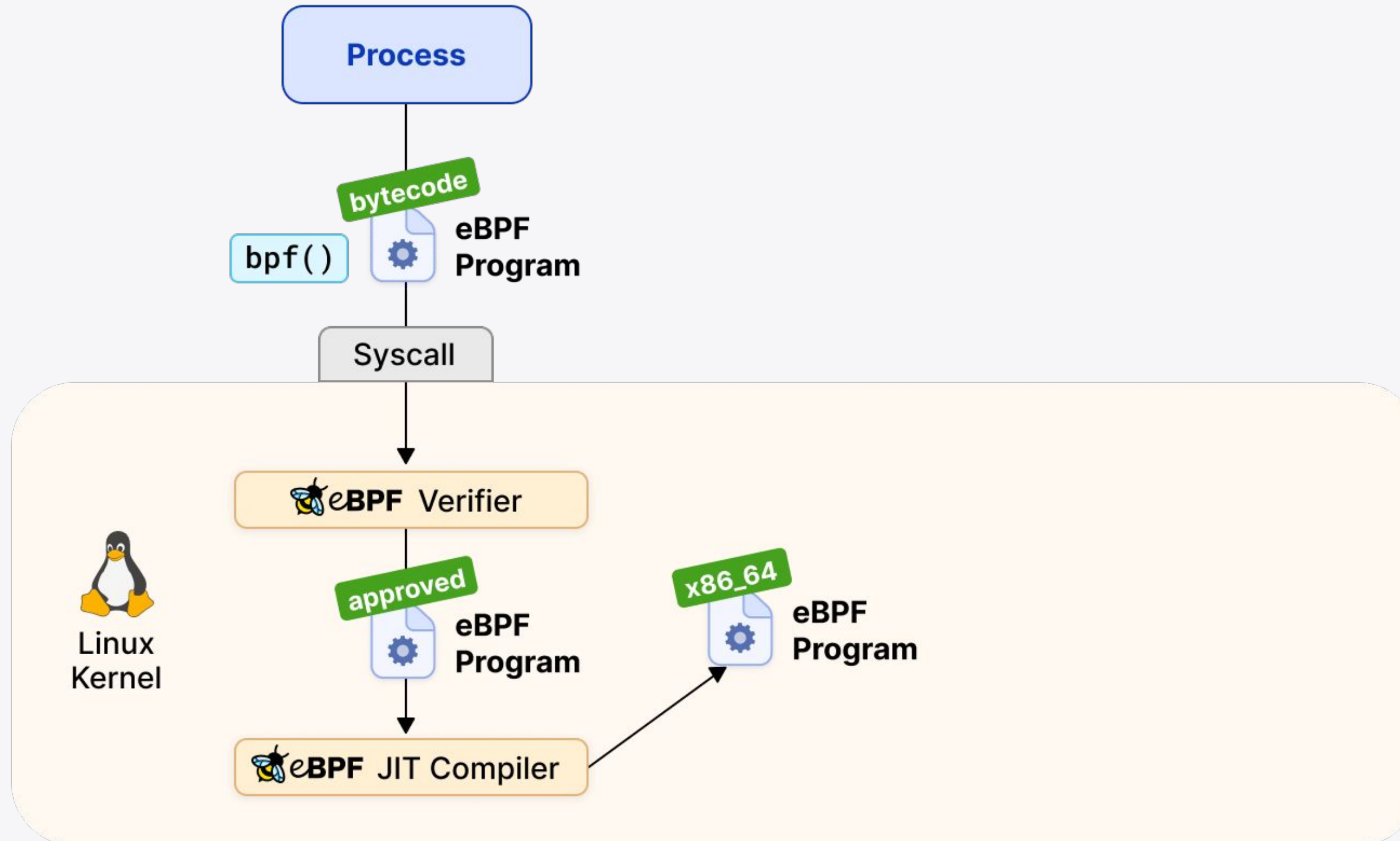
How eBPF works under the hood?



How eBPF works under the hood?



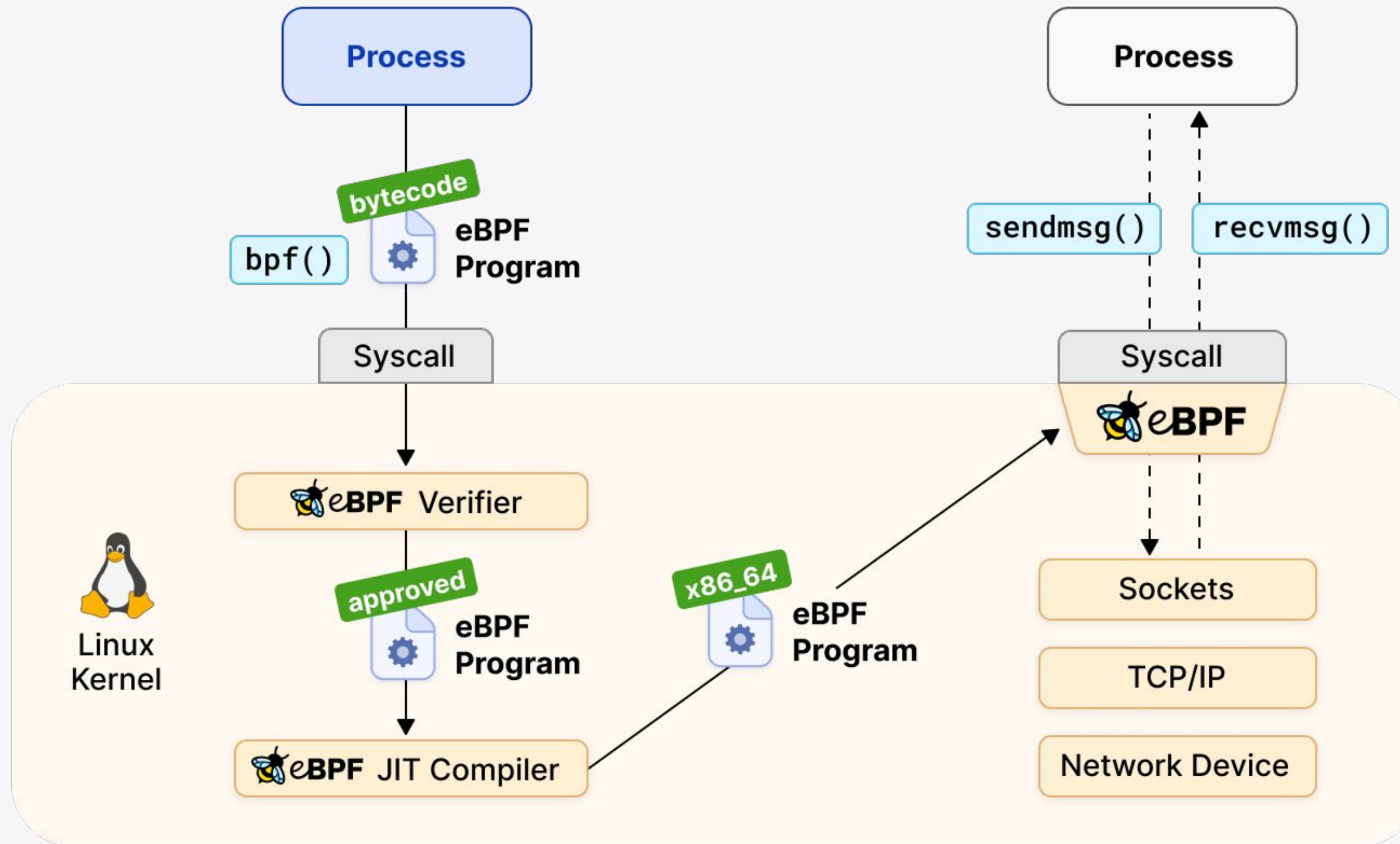
How eBPF works under the hood?



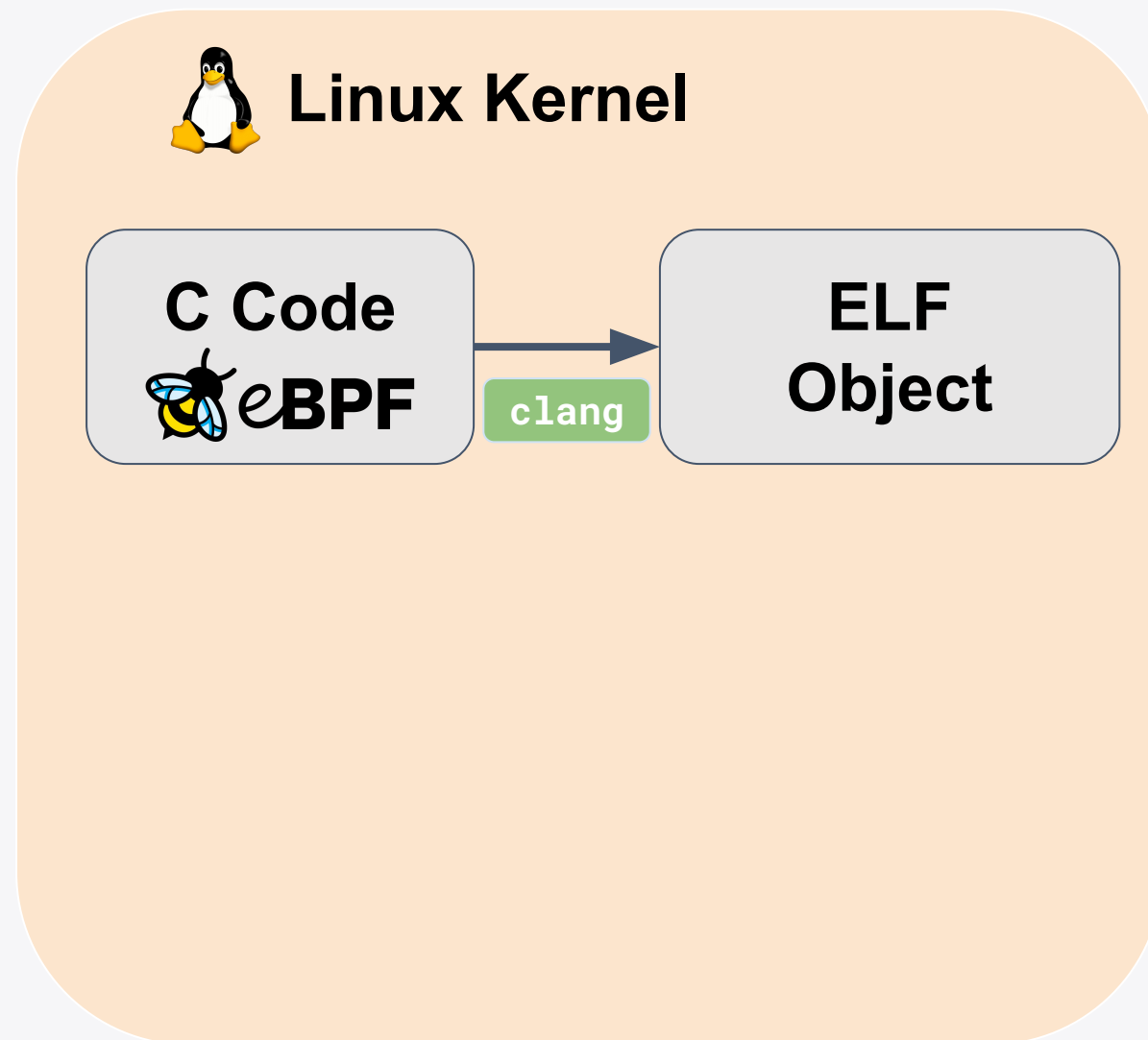


@doniacld

How eBPF works under the hood?



Develop with eBPF



Write eBPF programs

Bytecode 1011

Who wants to do that? 🤪

C

- Huge ecosystem
- Handle precisely low level resources

 Libraries

- [libbpf](#)

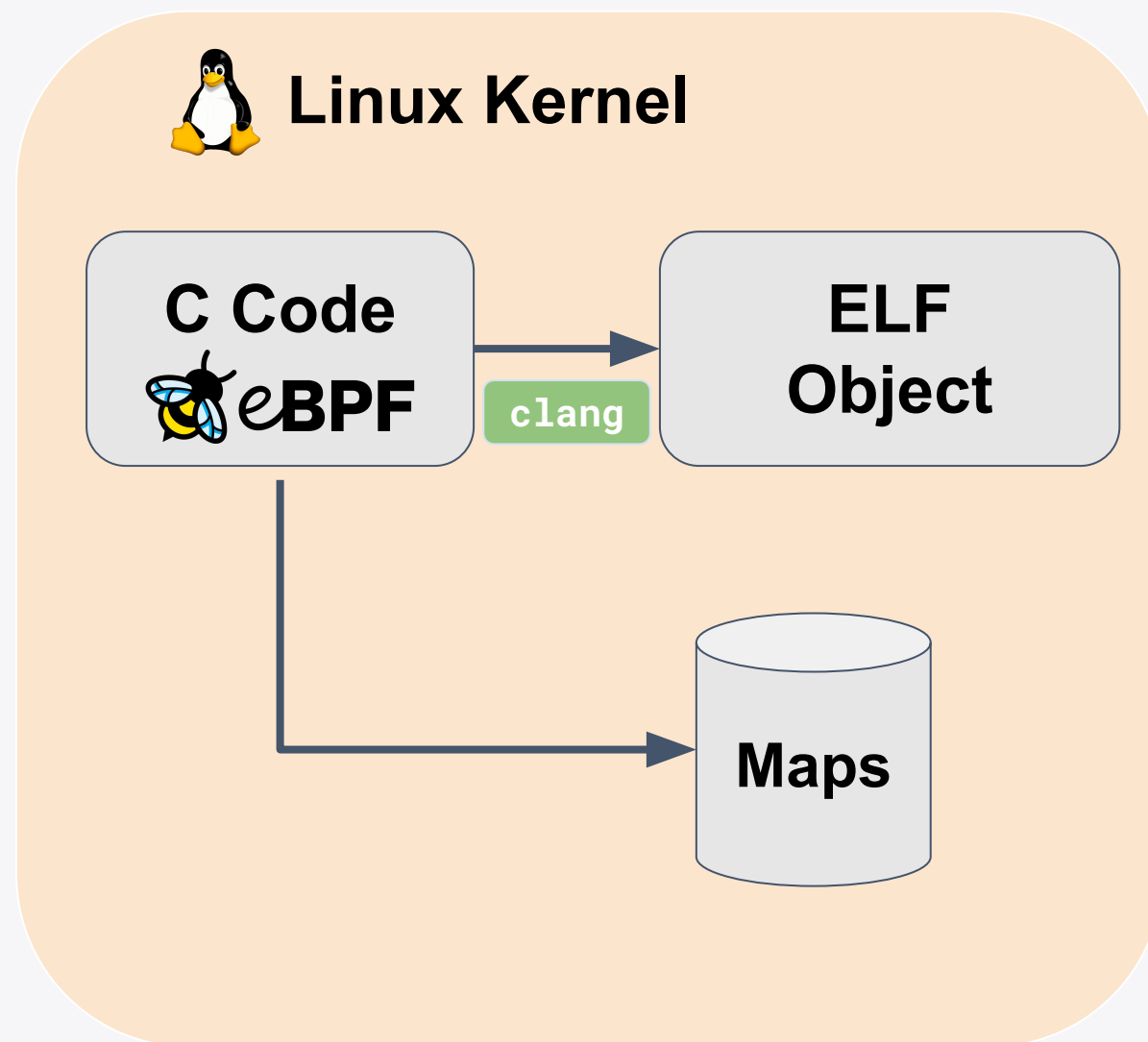
Rust

- Memory safety
- Concurrency made easy
- Ecosystem in maturation
- Verifier replaced

 Libraries

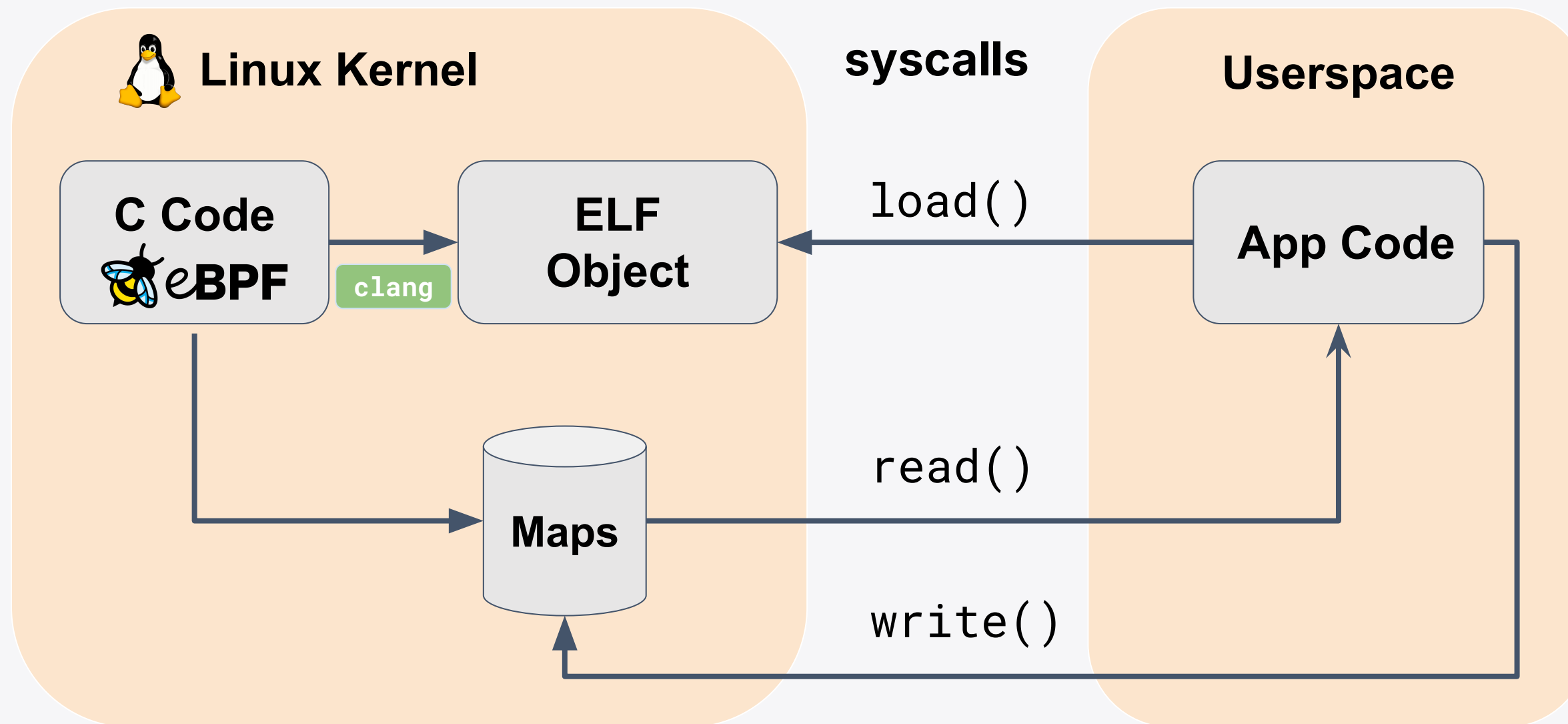
- [libbpf/libbpf-rs](#)
- [aya](#)

Maps in eBPF



BPF_MAP_TYPE_HASH
BPF_MAP_TYPE_ARRAY
BPF_MAP_TYPE_PERCPU_HASH
BPF_MAP_TYPE_PERCPU_ARRAY
BPF_MAP_TYPE_QUEUE
BPF_MAP_TYPE_STACK

Write programs in userspace loading eBPF



Write programs in userspace loading eBPF

C 

 painful to run in high level
app

Write programs in userspace loading eBPF

C 

 painful to run in high level app

Python 

-  Easy to start with high level language
- Wrappers for functions and maps

Write programs in userspace loading eBPF

C 

 painful to run in high level app

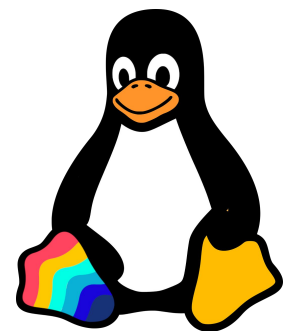
Python 

-  Easy to start with high level language
- Wrappers for functions and maps

Go 

-  Simplicity: easy syntax
-  Performance: efficient concurrency
- Wrappers for functions and maps

Go Libraries for eBPF



aquasecurity/libbpfgo

- A Go wrapper around libbpf
- Involves managing CGO and potential challenges with cross-compilation
- Provides access to the latest eBPF features and updates through libbpf



iovisor/gobpf

- Go bindings for the bcc framework
- Access to low-level routines
- Involves managing CGO



cilium/ebpf

- Load and attach eBPF programs
- Interact with eBPF maps for dynamic data management
- May not support the latest libbpf features

Monitoring Example

<https://github.com/doniacld/ebpf-4-gophers>

Packet counter: C eBPF program

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

```
struct {
    __uint(type, BPF_MAP_TYPE_ARRAY);
    __type(key, __u32);
    __type(value, __u64);
    __uint(max_entries, 1);
} pkt_count SEC(".maps");
```

```
// count_packets atomically increases a packet counter on every invocation.
```

```
SEC("xdp")
```

```
int count_packets() {
    __u32 key = 0;
    __u64 *count = bpf_map_lookup_elem(&pkt_count, &key);
    if (count) {
        __sync_fetch_and_add(count, 1);
    }

    return XDP_PASS;
}
```

```
char __license[] SEC("license") = "Dual MIT/GPL";
```

Packet counter: C eBPF program

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

struct {
    __uint(type, BPF_MAP_TYPE_ARRAY);
    __type(key, __u32);
    __type(value, __u64);
    __uint(max_entries, 1);
} pkt_count SEC(".maps");

// count_packets atomically increases a packet counter on every invocation.
SEC("xdp")
int count_packets() {
    __u32 key = 0;
    __u64 *count = bpf_map_lookup_elem(&pkt_count, &key);
    if (count) {
        __sync_fetch_and_add(count, 1);
    }

    return XDP_PASS;
}

char __license[] SEC("license") = "Dual MIT/GPL";
```

bpf2go: Compile to ELF objects

```
//go:generate go run github.com/cilium/ebpf/cmd/bpf2go counter counter.c
```

call clang
under the hood



Generated Go Objects

```
//go:generate go run github.com/cilium/ebpf/cmd/bpf2go counter counter.c
```

Objects to manipulate

```
// counterObjects contains all objects after they have been
// loaded into the kernel.
type counterObjects struct {
    counterPrograms
    counterMaps
    counterVariables
}
```

Generated Go Objects

```
//go:generate go run github.com/cilium/ebpf/cmd/bpf2go counter counter.c
```

Objects to manipulate

```
// counterObjects contains all objects after they have been
// loaded into the kernel.
type counterObjects struct {
    counterPrograms
    counterMaps
    counterVariables
}
```

eBPF program bytecodes

```
// counterPrograms contains all programs after they have
// been loaded into the kernel.
type counterPrograms struct {
    CountPackets *ebpf.Program `ebpf:"count_packets"`
}
```


Generated Go Objects

```
//go:generate go run github.com/cilium/ebpf/cmd/bpf2go counter counter.c
```

Objects to manipulate

```
// counterObjects contains all objects after they have been
// loaded into the kernel.
type counterObjects struct {
    counterPrograms
    counterMaps
    counterVariables
}
```

eBPF program bytecodes

```
// counterPrograms contains all programs after they have
// been loaded into the kernel.
type counterPrograms struct {
    CountPackets *ebpf.Program `ebpf:"count_packets"`
}
```

Maps to read/write

```
// counterMaps contains all maps after they have been loaded
// into the kernel.
type counterMaps struct {
    PktCount *ebpf.Map `ebpf:"pkt_count"`
}
```

Load the eBPF program

```
package main

import (
    // ...

    "github.com/cilium/ebpf/link"
    "github.com/cilium/ebpf/rlimit"
)

func main() {
    err := rlimit.RemoveMemlock()

    var objs counterObjects
    err := loadCounterObjects(&objs, nil)
    if err != nil { //... }
    defer objs.Close()

    ifname := "eth0"
    iface, err := net.InterfaceByName(ifname)
    if err != nil { //... }

    link, err := link.AttachXDP(link.XDPOptions{
        Program:    objs.CountPackets,
        Interface:  iface.Index,
    })
    defer link.Close()
}
```

Choose an interface

```
package main

import (
    // ...

    "github.com/cilium/ebpf/link"
    "github.com/cilium/ebpf/rlimit"
)

func main() {
    err := rlimit.RemoveMemlock()

    var objs counterObjects
    err := loadCounterObjects(&objs, nil)
    if err != nil { //... }
    defer objs.Close()

    ifname := "eth0"
    iface, err := net.InterfaceByName(ifname)
    if err != nil { //... }

    link, err := link.AttachXDP(link.XDPOptions{
        Program:  objs.CountPackets,
        Interface: iface.Index,
    })
    defer link.Close()
```

Attach the program to a hook

```
package main

import (
    // ...

    "github.com/cilium/ebpf/link"
    "github.com/cilium/ebpf/rlimit"
)

func main() {
    err := rlimit.RemoveMemlock()

    var objs counterObjects
    err := loadCounterObjects(&objs, nil)
    if err != nil { //... }
    defer objs.Close()

    ifname := "eth0"
    iface, err := net.InterfaceByName(ifname)
    if err != nil { //... }

    link, err := link.AttachXDP(link.XDPOptions{
        Program:  objs.CountPackets,
        Interface: iface.Index,
    })
    defer link.Close()

    ...
}
```

Read from the map

```
// Periodically fetch the packet counter from PktCount
tick := time.Tick(time.Second)
stop := make(chan os.Signal, 5)
signal.Notify(stop, os.Interrupt)
for {
    select {
    case <-tick:
        var count uint64
        err := objs.PktCount.Lookup(uint32(0), &count)
        if err != nil {
            log.Fatal("Map lookup:", err)
        }
        log.Printf("Received %d packets", count)
    case <-stop:
        log.Print("Received signal, exiting..")
        return
    }
}
```


Read from the map



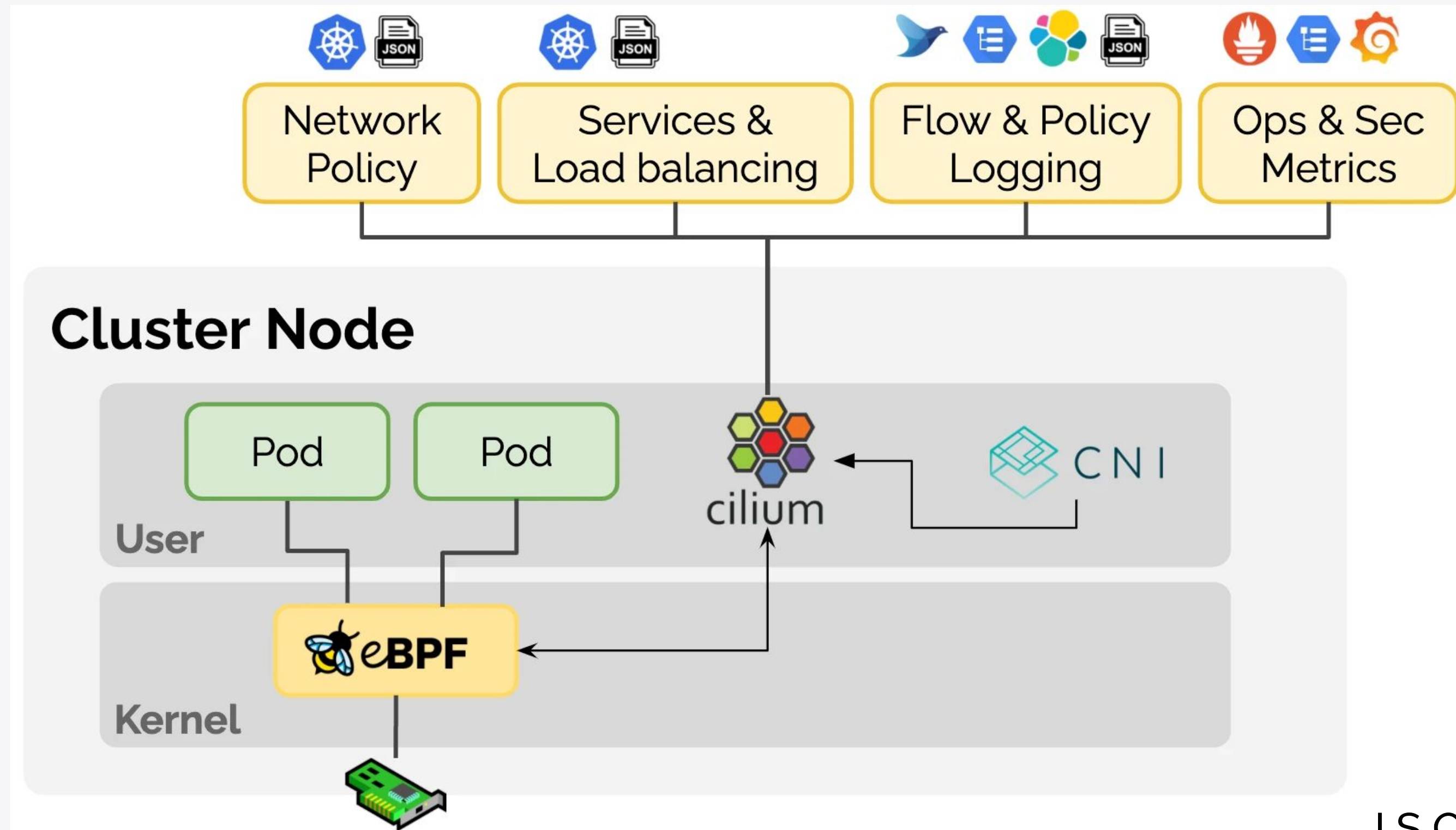
```
// Periodically fetch the packet counter from PktCount
tick := time.Tick(time.Second)
stop := make(chan os.Signal, 5)
signal.Notify(stop, os.Interrupt)
for {
    select {
    case <-tick:
        var count uint64
        err := objs.PktCount.Lookup(uint32(0), &count)
        if err != nil {
            log.Fatal("Map lookup:", err)
        }
        log.Printf("Received %d packets", count)
    case <-stop:
        log.Print("Received signal, exiting..")
        return
    }
}
```

Packet Counter Demo

Tool Example

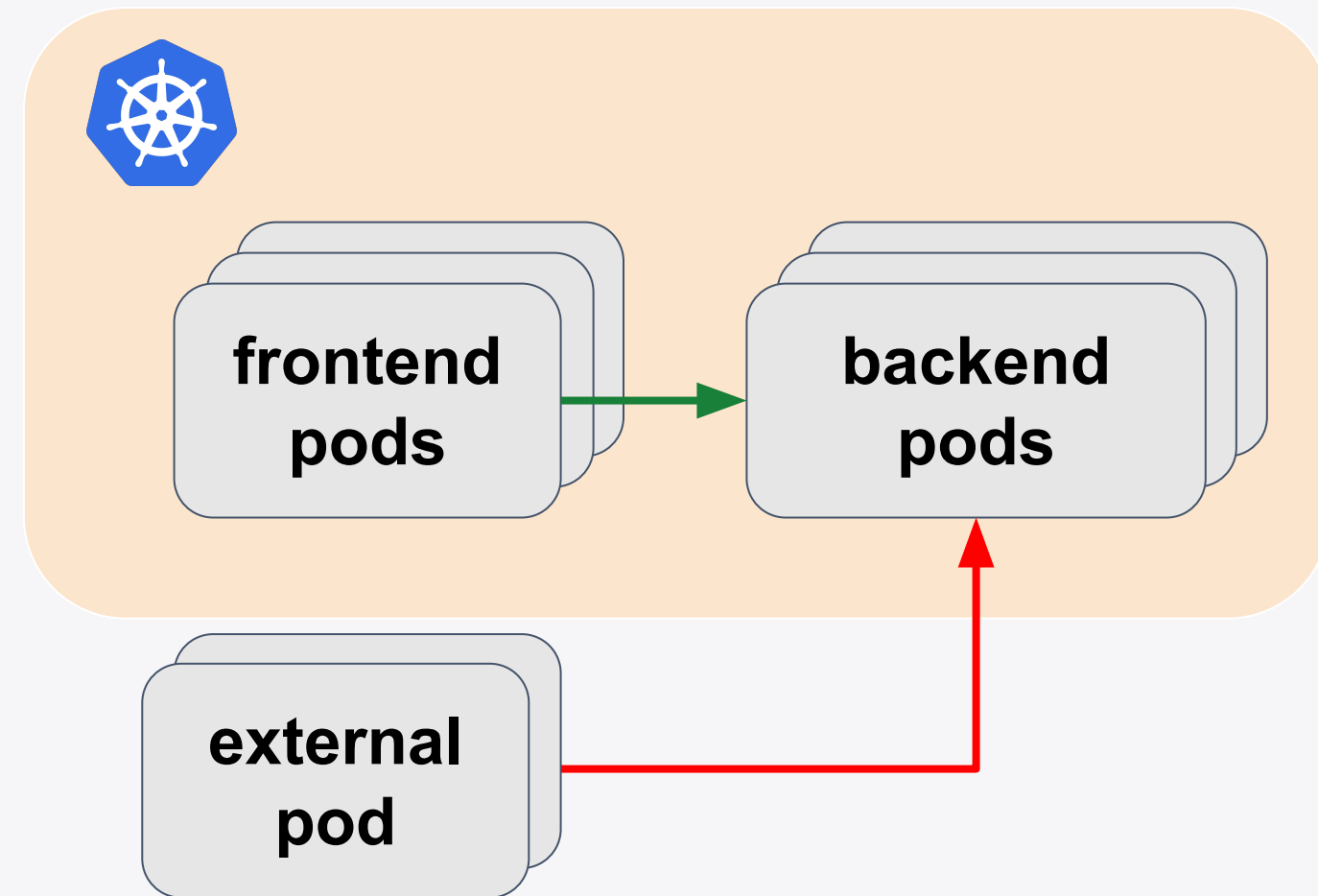
Cilium

Overview of Cilium



Cilium Network Policy

```
apiVersion: "cilium.io/v2"
kind: CiliumNetworkPolicy
metadata:
  name: "deny-external"
spec:
  endpointSelector:
    matchLabels:
      app: "backend"
  ingress:
    - fromEndpoints:
      - matchLabels:
          app: "frontend"
```



From Endpoint to Identity

 Endpoint = Pod

```
type Endpoint struct {  
    ID          int  
    Labels      map[string]string  
    Identity    int  
}
```

Endpoints

```
1: {"app": "frontend"}  
2: {"app": "backend"}
```

```
type Policy struct {  
    SourceIdentity    int  
    DestinationIdentity int  
    Allowed           bool  
}
```

Policy

```
src: 1  
dst: 2  
true
```


Store Policies

eBPF map to store policies

```
struct bpf_map_def SEC("maps")
policy_map = {
    .type = BPF_MAP_TYPE_HASH,
    .key_size = sizeof(int),
    .value_size = sizeof(int),
    .max_entries = 1024,
};
```

Function to load policies into the eBPF map.

```
func loadPoliciesToMap(policies []Policy, policyMap *ebpf.Map) error {
    for _, policy := range policies {
        err := policyMap.Put(policy.SourceIdentity, policy.DestinationIdentity)
        if err != nil {
            return err
        }
    }
    return nil
}
```

eBPF program to enforce policies

```
// eBPF map to store policies.
struct bpf_map_def SEC("maps") policy_map = {
    .type = BPF_MAP_TYPE_HASH,
    .key_size = sizeof(int),
    .value_size = sizeof(int),
    .max_entries = 1024,
};

// eBPF program to enforce policies.
SEC("xdp")
int enforce_policy(struct xdp_md *ctx) {
    int src_identity = get_source_identity(ctx);
    int dst_identity = get_destination_identity(ctx);
    int *allowed = bpf_map_lookup_elem(&policy_map, &src_identity);
    if (allowed && *allowed == dst_identity) {
        // Allow the packet.
        return XDP_PASS;
    }
    // Drop the packet.
    return XDP_DROP;
}
```

Thank you!

ebpf.io

@ciliumproject

@isovalent

<https://isovalent.com/labs/ebpf-getting-started/>

Let's connect



Security Example

<https://github.com/doniacld/ebpf-4-gophers>

Packet drop egress

```
#define BLOCKED_IP 0x7f000001 // 127.0.0.1 in hex

SEC("xdp")
int drop_packet(struct xdp_md *ctx) {
    void *data = (void *)(long)ctx->data;
    void *data_end = (void *)(long)ctx->data_end;

    // Check if the packet's destination IP is blocked
    if (isBlockedIP(data, data_end)) {
        char msg[] = "Got ping packet";
        bpf_trace_printk(msg, sizeof(msg)); // Print to trace pipe
        return XDP_DROP;
    }

    return XDP_PASS; // Allow the packet to pass
}
```


Packet drop egress

```
static inline int isBlockedIP(void *data, void *data_end) {
    struct ethhdr *eth = data;
    struct iphdr *ip;

    // Ensure there is enough data for the Ethernet and IP headers
    if ((void *)(eth + 1) > data_end) {
        return XDP_PASS; // Not enough data for IP header
    }

    ip = (struct iphdr *)(eth + 1);

    // Ensure there is enough data for the IP header
    if ((void *)(ip + 1) > data_end) {
        return XDP_PASS; // Not enough data for IP header fields
    }

    // Check if the destination IP matches the blocked IP
    if (ip->daddr == __constant_htonl(BLOCKED_IP)) {
        return XDP_DROP; // Blocked IP found
    }

    return XDP_PASS; // Not blocked
}
```


Generated Go Objects

```
//go:generate go run github.com/cilium/ebpf/cmd/bpf2go xdp_drop xdp_drop.c
```

Objects to manipulate

```
// xdpdropObjects contains all objects after they have been  
loaded into the kernel.
```

```
type xdpdropObjects struct {  
    xdpdropPrograms  
    xdpdropMaps  
    xdpdropVariables  
}
```

eBPF program bytecodes

```
// xdpdropPrograms contains all programs after they have  
been loaded into the kernel.
```

```
type xdpdropPrograms struct {  
    XdpDrop *ebpf.Program `ebpf:"xdp_drop"`  
}
```

Choose an interface

```
func main() {
    // Allow the current process to lock memory for eBPF resources.
    // ...

    // Load the compiled eBPF program
    // ...

    // Get the network interface
    ifaceName := "lo"
    iface, err := net.InterfaceByName(ifaceName)
    if err != nil {
        log.Fatalf("Failed to get interface %s: %v", ifaceName, err)
    }

    // Attach the eBPF program to the interface using XDP
    l, err := link.AttachXDP(link.XDPOptions{
        Program:   objs.DropPacket,
        Interface: iface.Index,
        Flags:     link.XDPGenericMode,
    })
    if err != nil {
        log.Fatalf("Failed to attach XDP program: %v", err)
    }
    defer l.Close()
}
```

Attach the program to a hook

```
func main() {
    // Allow the current process to lock memory for eBPF resources.
    // ...

    // Load the compiled eBPF program
    // ...

    // Get the network interface
    ifaceName := "eth0"
    iface, err := net.InterfaceByName(ifaceName)
    if err != nil {
        log.Fatalf("Failed to get interface %s: %v", ifaceName, err)
    }

    // Attach the eBPF program to the interface using XDP
    l, err := link.AttachXDP(link.XDPOptions{
        Program:   objs.DropPacket,
        Interface:  iface.Index,
        Flags:      link.XDPGenericMode,
    })
    if err != nil {
        log.Fatalf("Failed to attach XDP program: %v", err)
    }
    defer l.Close()
}
```

Packet Drop Demo