第二届 eBPF开发者大会

www.ebpftravel.com

独立于操作系统的eBPF组件架构（eBPFCA）

分享人：胡庆伟

时间：2024年4月

中国·西安

第二届 eBPF开发者大会

www.ebpftravel.com

① eBPFCA概述

中国·西安

① eBPFCA概述

eBPF：eBPF是Linux内核一种革命性的技术，起源于Linux内核，可以在内核中运行沙盒程序。由于内核具有监控和控制整个系统的特权，它通常是实现可观测性、安全性和网络功能的理想场所。

① eBPFCA概述

eBPF应用：探测、追踪、与内核合作提高效率

eBPF优势：

安全、高性能、扩展性

应用程序可移植性强、工具链完善

① eBPFCA概述

1. 探测、可观测行往往在系统成熟后会考虑；
2. eBPF的可移植性不只体现在不同Linux版本的适配；
3. 不同操作系统更方便的支持eBPF

独立于操作系统的eBPF组件架构（eBPFCA）

① eBPFCA概述

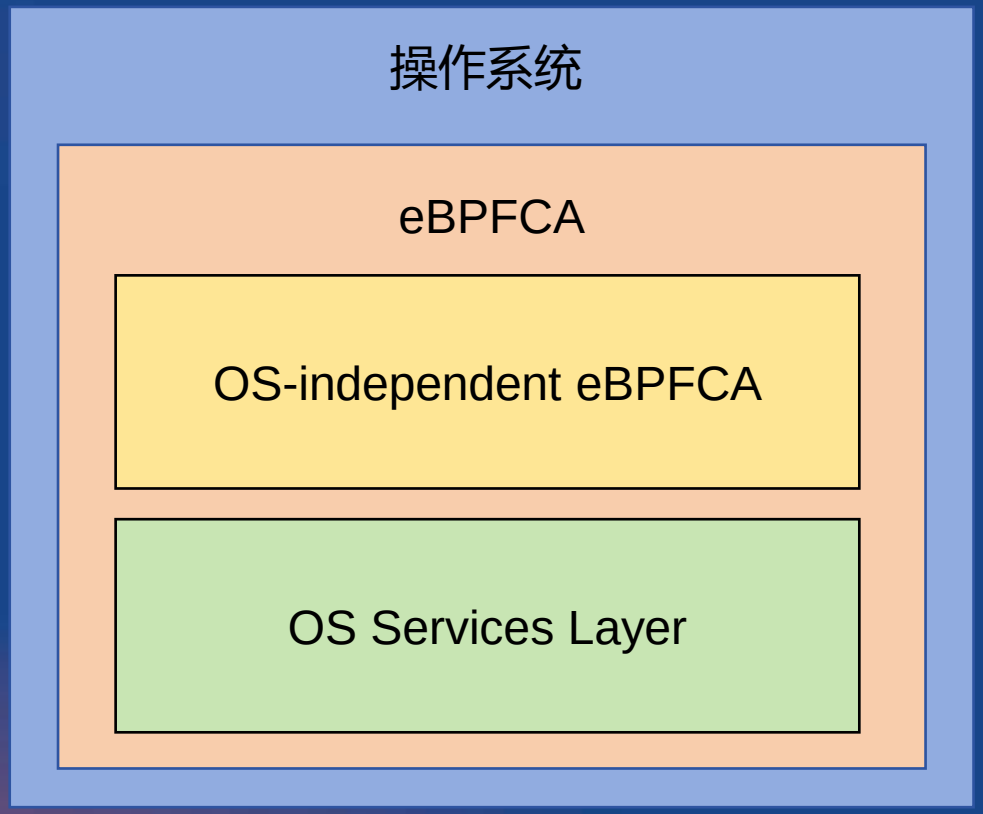
eBPFCA实现内核中eBPF的通用部分，向上为用户提供类似或相同于eBPF的接口。其中包括eBPF程序load、attach、字节码翻译等操作。

为了eBPFCA能够方便的与操作系统结合，eBPFCA将所有依赖操作系统的操作集中在操作系统服务层（OS Server Layer，简称OSL）。

① eBPFCA概述

1. 提供基本eBPF服务的操作系统独立层（OS-independent）；

2. 提供eBPFCA与特定操作系统的操作系统服务层（OSL）。



第二届 eBPF开发者大会

www.ebpftravel.com

② eBPFCA分析

中国·西安

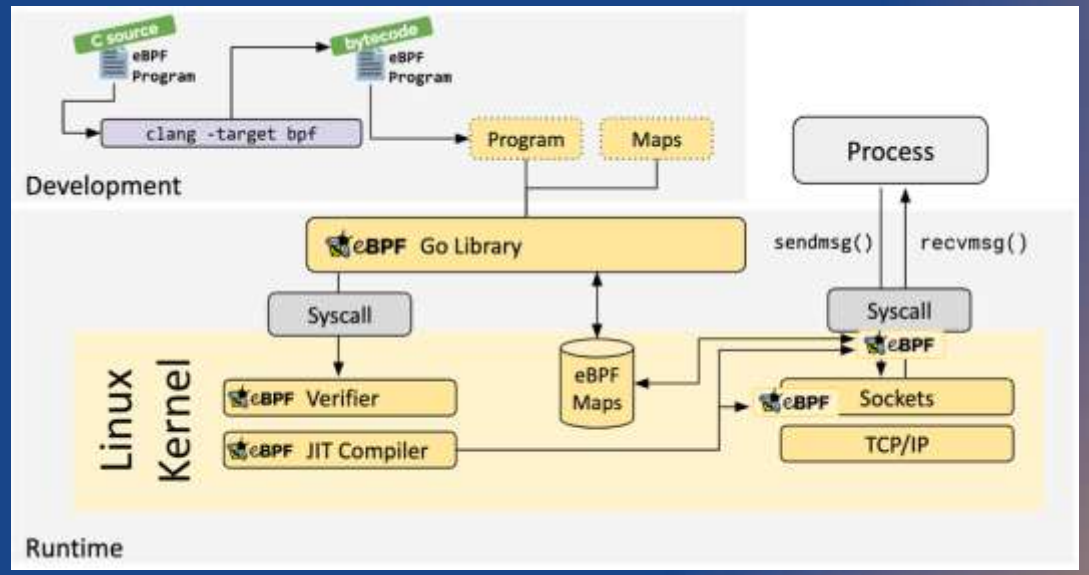
② eBPFCA分析

为用户提供与Linux中eBPF相同的使用方法。

Linux中eBPF的cmd：

- BPF_MAP_CREATE
- BPF_MAP_LOOKUP_ELEM
- BPF_PROG_LOAD
- BPF_OBJ_GET
- BPF_PROG_ATTACH
- ……

处理过程与linux现有eBPF的处理过程相同。



② eBPFCA分析

使用方法和处理过程的特点：

- eBPF的使用和处理过程中大多是和操作系统其他机制无关的。
- 分析操作系统其他机制有关的部分，仅有附加操作需要与操作系统进行配合。

eBPF通用部分（eBPF General）

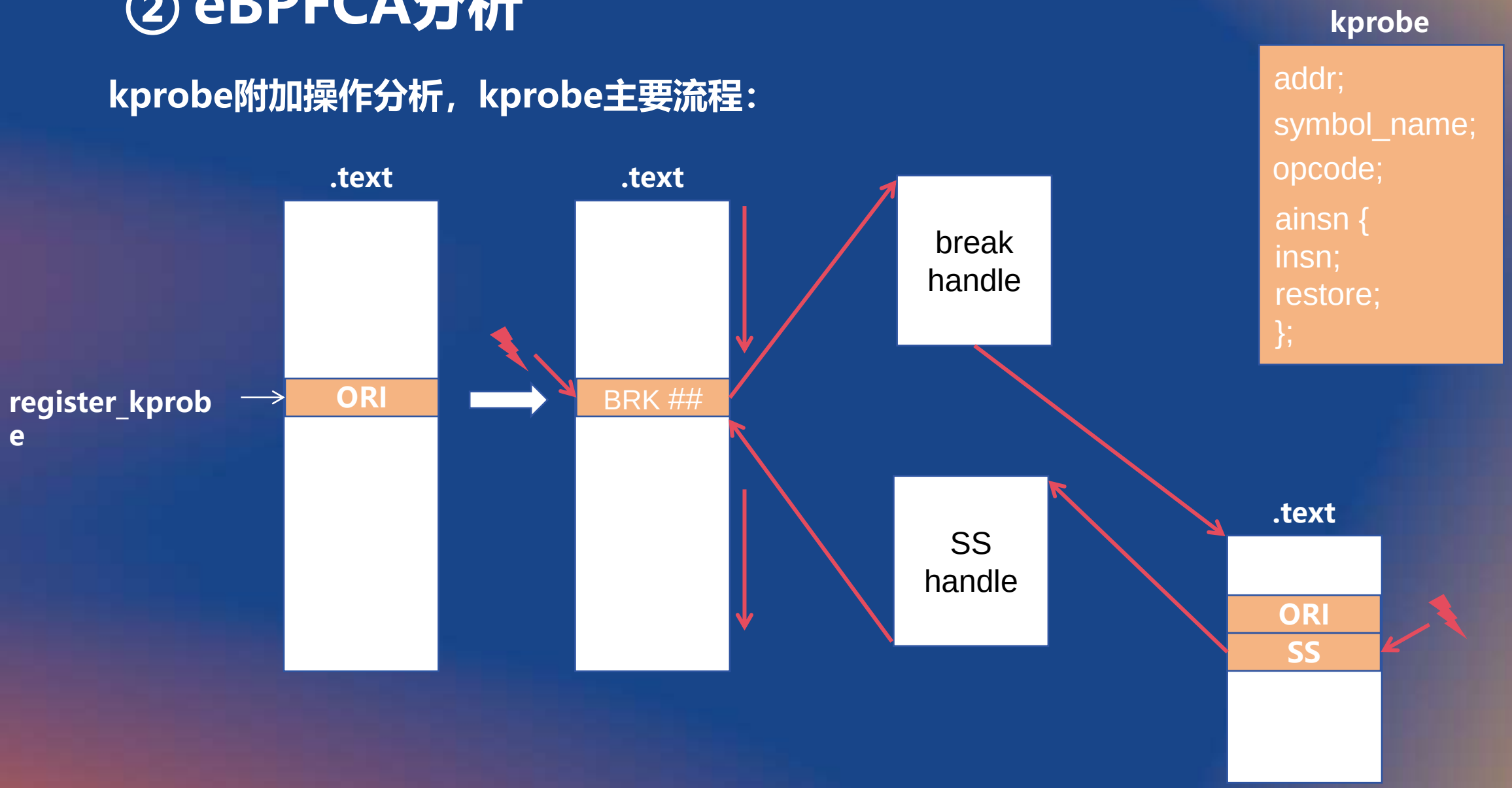
② eBPFCA分析

kprobe附加操作分析，kprobe主要流程：

1. 注册kprobe：获得探测点的地址，将探测点地址处的指令替换为break指令；准备一片可执行的内存，将原指令和一个SS单步调试指令写入；记录探测点地址的下一个指令地址。
2. 触发break异常：当系统运行到第1步替换的break指令时，会产生一个break异常，随后系统跳转到异常处理程序中进行break异常处理。在break异常处理中，修改异常返回地址为第1步准备的保存有原指令和一个SS单步调试指令的地址处。这样做的目的是为了让原指令得到运行。
3. 触发SS单步异常：当第2步的break异常处理完成后，由于修改了break异常的返回地址为保存有原指令和一个SS单步调试指令的地址处，所以break异常返回后，跳转到该处运行，执行完原指令后会触发一个SS单步调试指令，在单步调试指令中修改返回地址为探测点的下一条指令地址。SS单步调试异常返回后，就回到了正常的执行流中。

② eBPFCA分析

kprobe附加操作分析，kprobe主要流程：



② eBPFCA分析

提取kprobe处理流程中与操作系统系统相关的部分：

1. 获取探测点地址：kprobe通常使用函数名指定探测点，需要系统为kprobe提供符号所在的位置。

kprobe获得符号名可以作为OSL层的一个接口，为kprobe的OS-independent部分提供获取符号所在地址的功能。

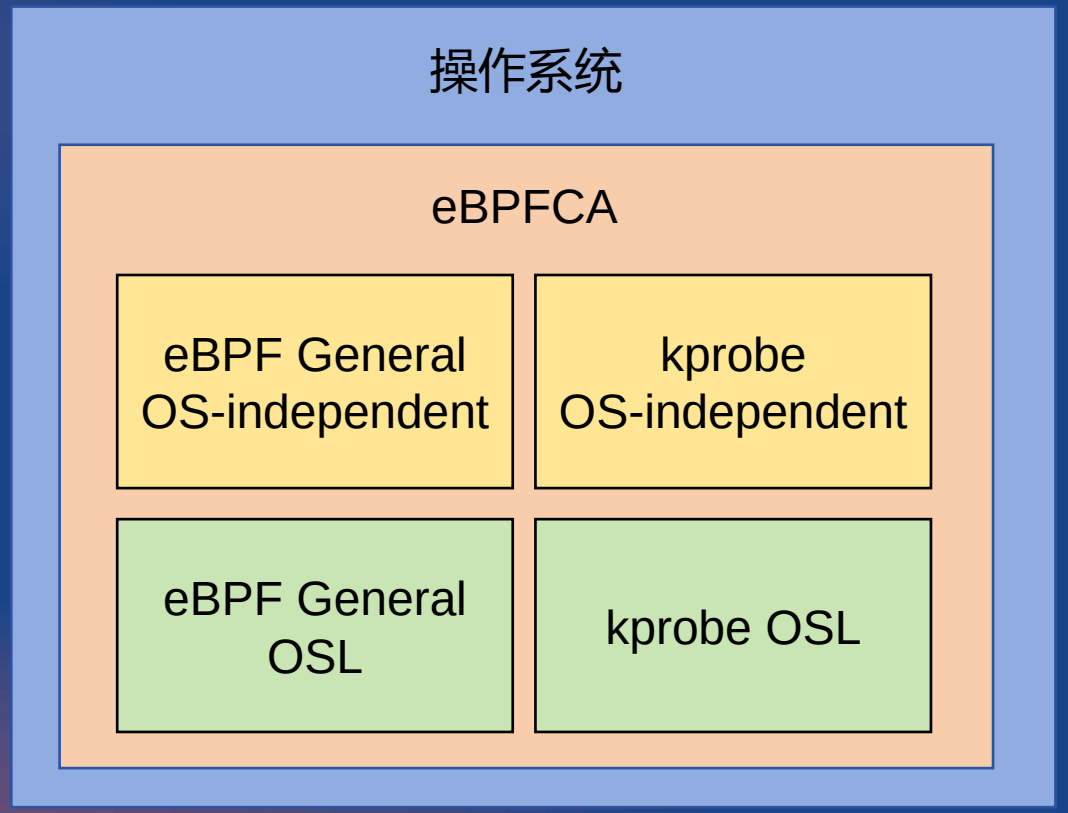
2. break异常处理：上述kprobe处理流程中的两个中断都是通过break异常进行的，两个异常的不同点仅在于操作码后几位不同，可以通过软件进行解析。

break异常处理中的主要处理逻辑可以作为OS-independent的部分，操作系统需要提供异常入口，并提供注册接口，将OS-independent的处理流程注册进异常处理中。

② eBPFCA分析

将eBPF General和kprobe结合起来，就形成了支持kprobe的eBPFCA。

综上，得到更细致的eBPFCA框架：



③ kprobe示例

③ kprobe示例

运行平台：

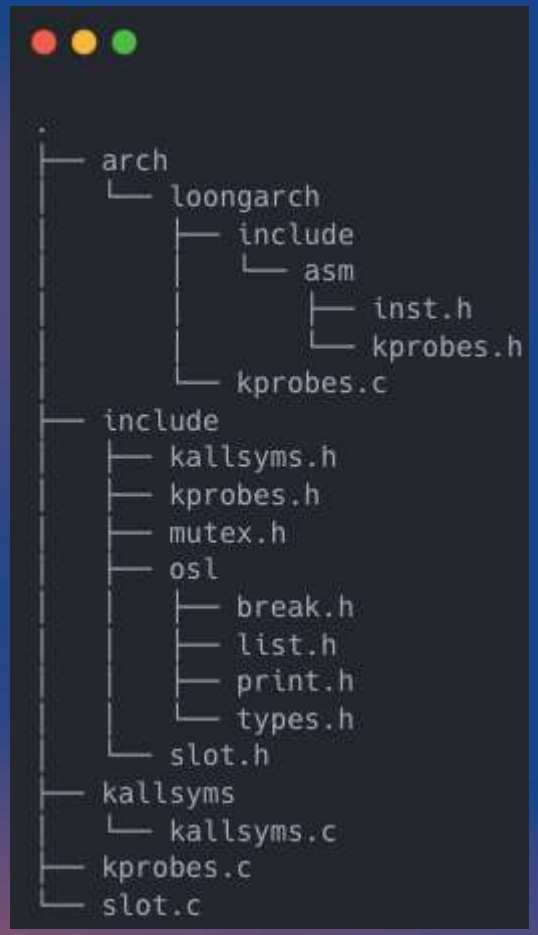
宿主机：ubuntu20.04

虚拟机：qemu-system-loongarch64（3A5000）

操作系统：小型操作系统（支持任务切换）

③ kprobe示例

源码结构：



```
.
├── arch
│   └── loongarch
│       ├── include
│       │   └── asm
│       │       ├── inst.h
│       │       └── kprobes.h
│       └── kprobes.c
├── include
│   ├── kallsyms.h
│   ├── kprobes.h
│   ├── mutex.h
│   ├── osl
│   │   ├── break.h
│   │   ├── list.h
│   │   ├── print.h
│   │   └── types.h
│   └── slot.h
├── kallsyms
│   └── kallsyms.c
├── kprobes.c
└── slot.c
```

使用示例：

```c
int handler_pre(struct kprobe *p, struct pt_regs *regs)
{
    printk("[kprobe] kthread_a will run\n");
    return 0;
}

struct kprobe kprobe_test = {
    .symbol_name = "kthread_a",
    .pre_handler = handler_pre,
};

void kthread_a(void *arg);

void init_all()
{
    int c = 0x10;
    thread_start("kthread_a", 31, kthread_a, "kthread_a");
    register_kprobe(&kprobe_test);
    intr_enable();
    while (1) {
        while (c--) ;
        printk("@@@@@: main kthread\n");
    }
}

void kthread_a(void *arg)
{
    char *str = arg;
    int a = 0x10;
    while (1) {
        while (a--) ;
        printk("@@@@@: %s\n", str);
    }
}
```

运行结果：

```
@@@@@: main kthread
@@@@@: main kthread
[kprobe] kthread_a will run
@@@@@: kthread_a
@@@@@: kthread_a
@@@@@: main kthread
@@@@@: main kthread
QEMU: Terminated
```

④ 总结

④ 总结

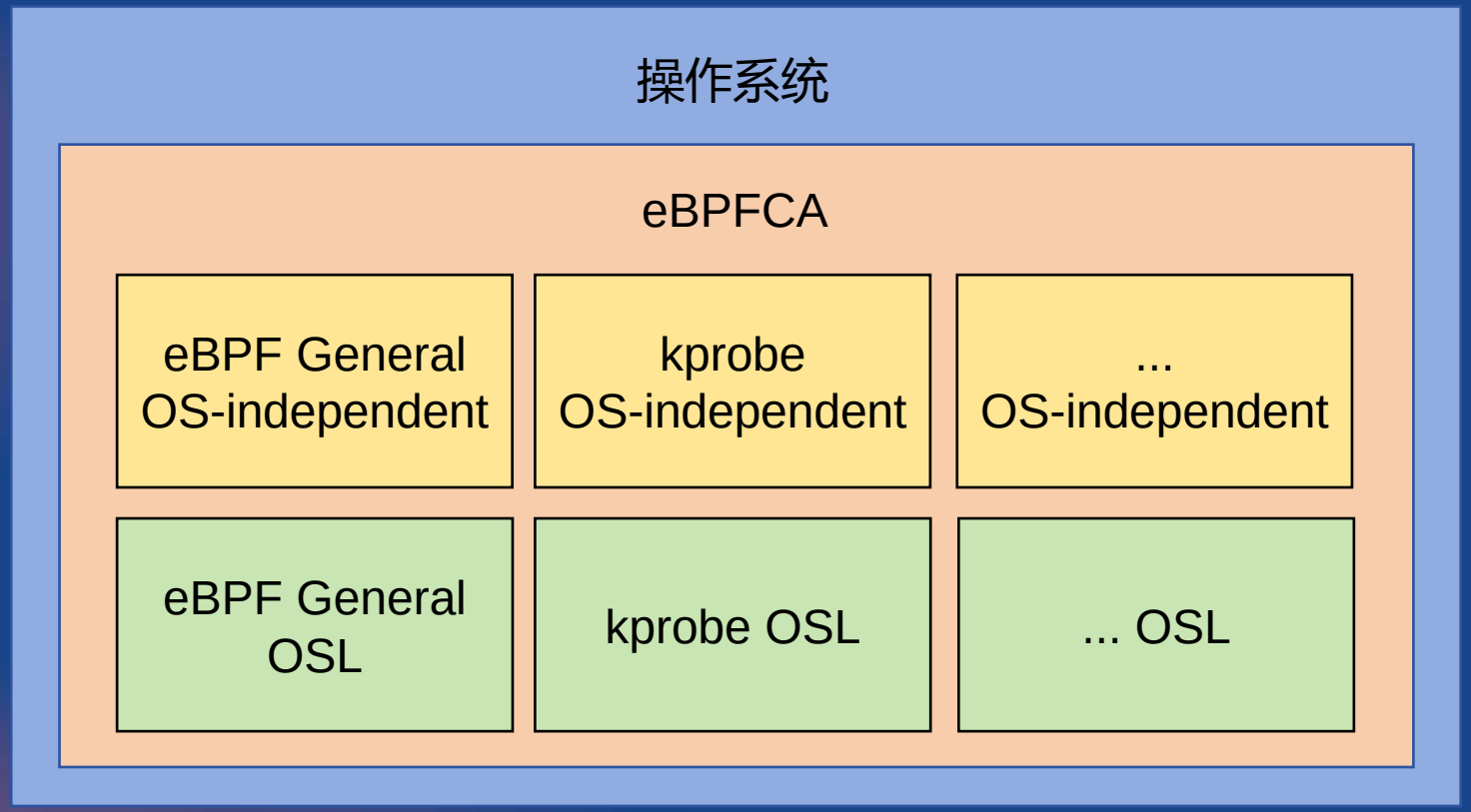
1. kprobe示例的实现，得到相对独立的kprobe。

2. eBPFCA分析中kprobe的分析，提供了一个如何将附加操作独立的例子：
    2. 分析kprobe的流程；
    3. 梳理kprobe的流程中操作系统无关和操作系统相关部分；
    4. 操作系统无关部分作为OS-independent，相关部分作为OSL；
    5. kprobe的OSL中依赖操作系统其他机制的部分需要制定协议。

3. kprobe提供一个示例，可推广至其他附加操作。

④ 总结

最终eBPFCA框架：



第二届 eBPF开发者大会

www.ebpftravel.com

感谢观看

分享人：胡庆伟　　时间：2024年4月

中国·西安