IO Visor @SCaLE 14x

Brenden Blanco

Jan. 23, 2016

Agenda

A bit of history and project motivation

An introduction to eBPF in the Linux kernel

An introduction to the BCC toolkit

Show how Clang/LLVM is integrated into BCC

Demo how to use IOVisor to build functional network applications

Demo how to use IOVisor to debug a live system

Q+A

Founding Members

What we want

Started with building networking applications for SDN

An SDK to extend low-level infrastructure

But…

Don’t want to become a kernel developer

Compare to a server app framework (e.g. Node.js)

Recognize that writing multithreaded apps is hard

Syntax that mirrors thought process, not the CPU arch (events vs threads)

Don’t sacrifice performance (v8 jit)

Make it easy to get code from the devs to deployment (npm)

Foster a community via sharing of code

What do you need to write infrastructure apps

High performance access to data

Reliability...it must never crash

In-place upgrades

Debug tools

A programming language abstraction

But there are restrictions

No custom kernels

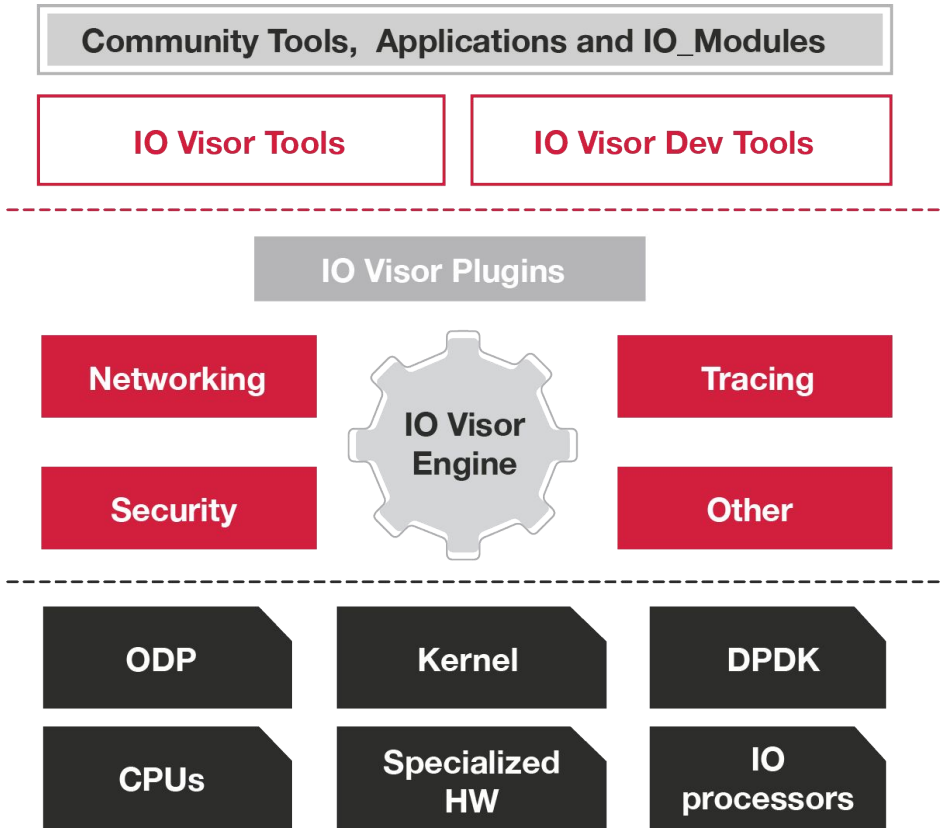
No custom kernel modules

No kernels with debug symbols

No reboots

(some of these are nice-to-haves)

IO Visor Project, What is in it?



- **IO Visor Engine** is an abstraction of an IO execution engine
- A set of development tools, **IO Visor Dev Tools**
- A set of **IO Visor Tools** for management and operations of the IO Visor Engine
- A set of Applications, Tools and open **IO Modules** build on top of the IO Visor framework
- A set of possible use cases & applications like **Networking, Security, Tracing & others**

Hello, World! Demo

```
#!/usr/bin/python
import bcc
b = bcc.BPF(text="""
void kprobe__sys_clone(void *ctx) {
  bpf_trace_printk("Hello, World!\\n");
}
""")
b.trace_print()
```

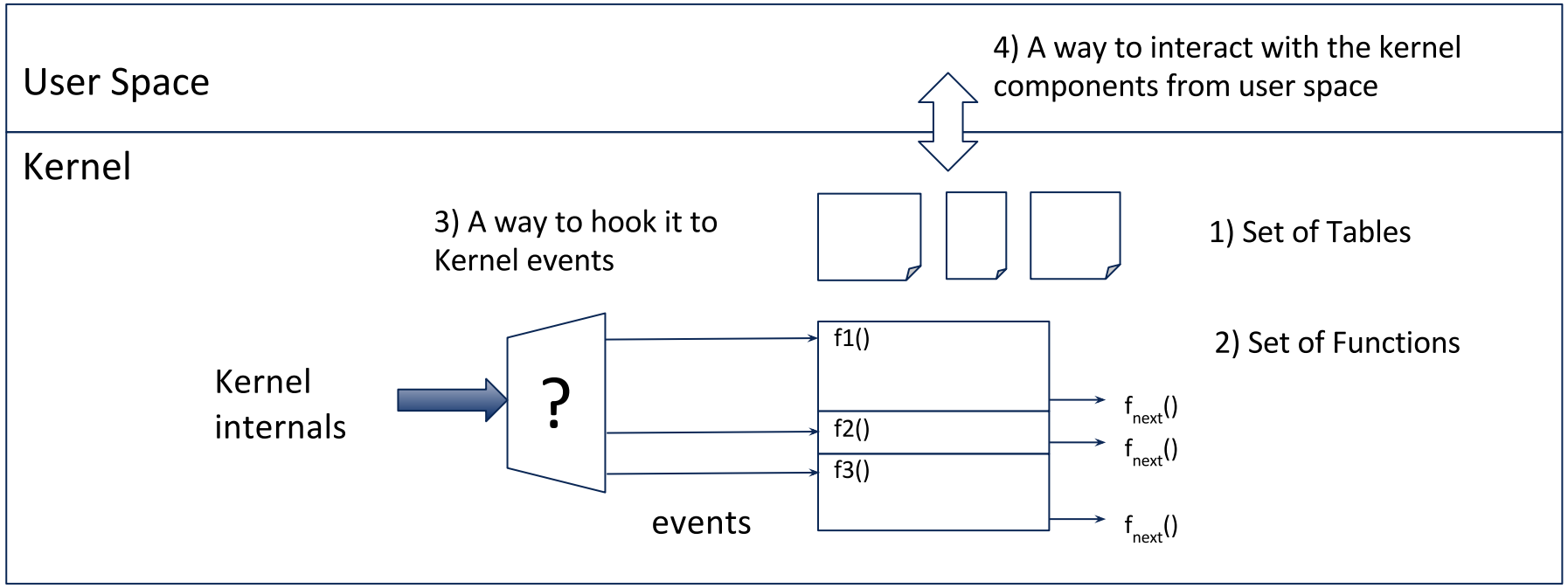
BPF

What are BPF Programs?

In a very simplified way:

A safe, runtime way to extend Linux kernel capabilities

Functions, Maps, Attachment Points, Syscall



More on BPF Programs

Berkeley Packet Filters around since 1990, extensions started Linux 3.18

Well, not really a program (no pid)...an event handler

- A small piece of code, executed when an event occurs
- In-kernel virtual machine executes the code
- Assembly instruction set

See ‘man 2 bpf’ for details

The eBPF Instruction Set

Instructions

- 10x 64bit registers
- 512B stack
- 1-8B load/store
- conditional jump
- arithmetic
- function call

Helper functions

- forward/clone/drop packet
- load/store packet data
- load/store packet metadata
- checksum (incremental)
- push/pop vlan
- access kernel mem (kprobes)

Data structures

- lookup/update/delete
  - in-kernel or from userspace
- hash, array, ...

BPF Kernel Hook Points

A program can be attached to:

- kprobes or uprobes
- socket filters
  - TAP or RAW (original tcpdump use case)
  - PACKET_FANOUT: loadbalance packets to sockets
- seccomp
- tc filters or actions, either ingress or egress

BPF Verifier

A program is declared with a type (kprobe, filter, etc.)

Only allows permitted helper functions

Kernel parses BPF instructions into a DAG

Disallows: back edges, unreachable blocks, illegal insns, finite execution

No memory accesses from off-stack, or from unverified source

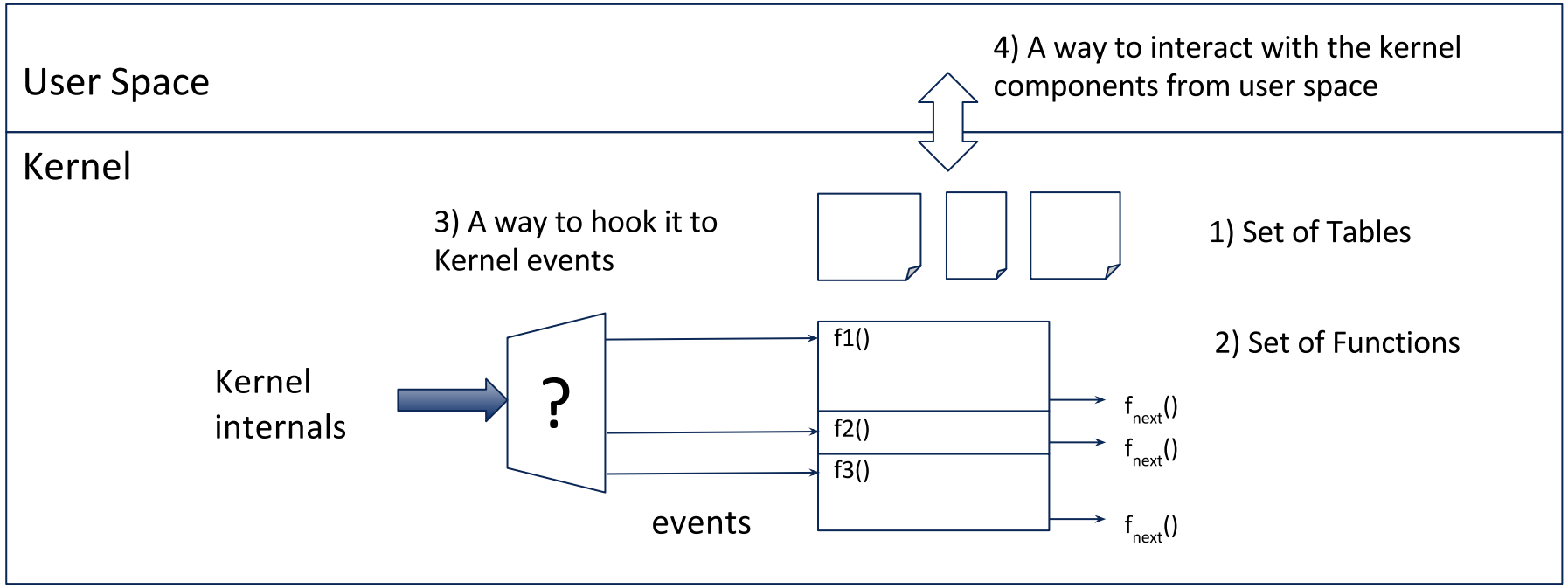
Program ok? => JIT compile to native instructions (x86_64, arm64, s390)

What are BPF Programs?

In a very simplified way:

A safe, runtime way to extend Linux kernel capabilities

Functions, Maps, Attachment Points, Syscall



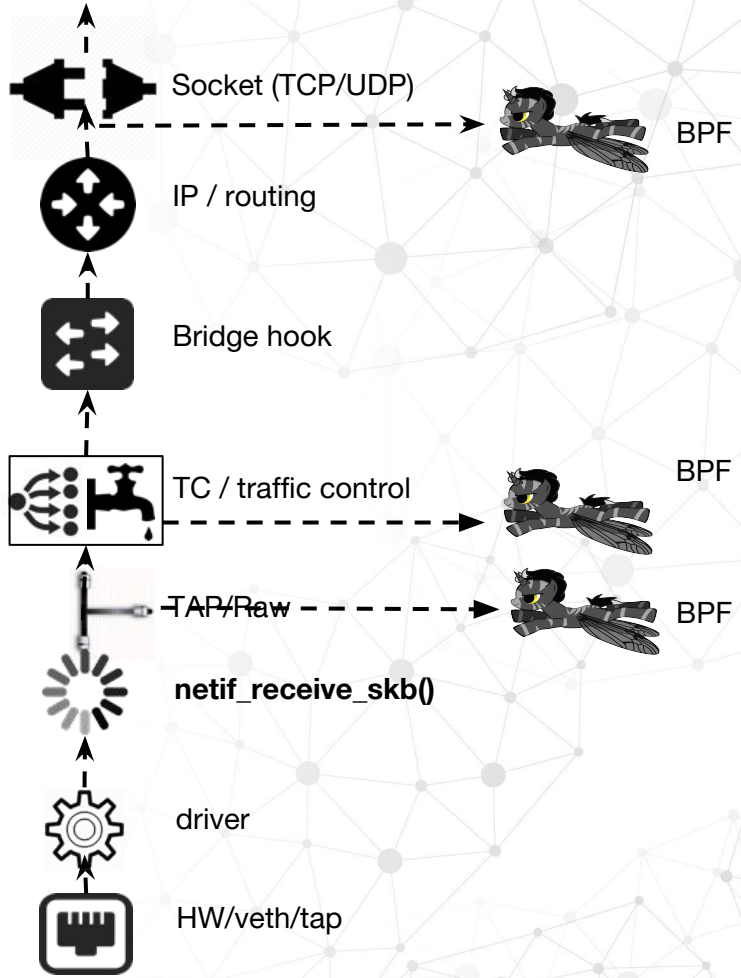
Developer Workflow

eBPF program written in C

Translated into eBPF instructions (LLVM)

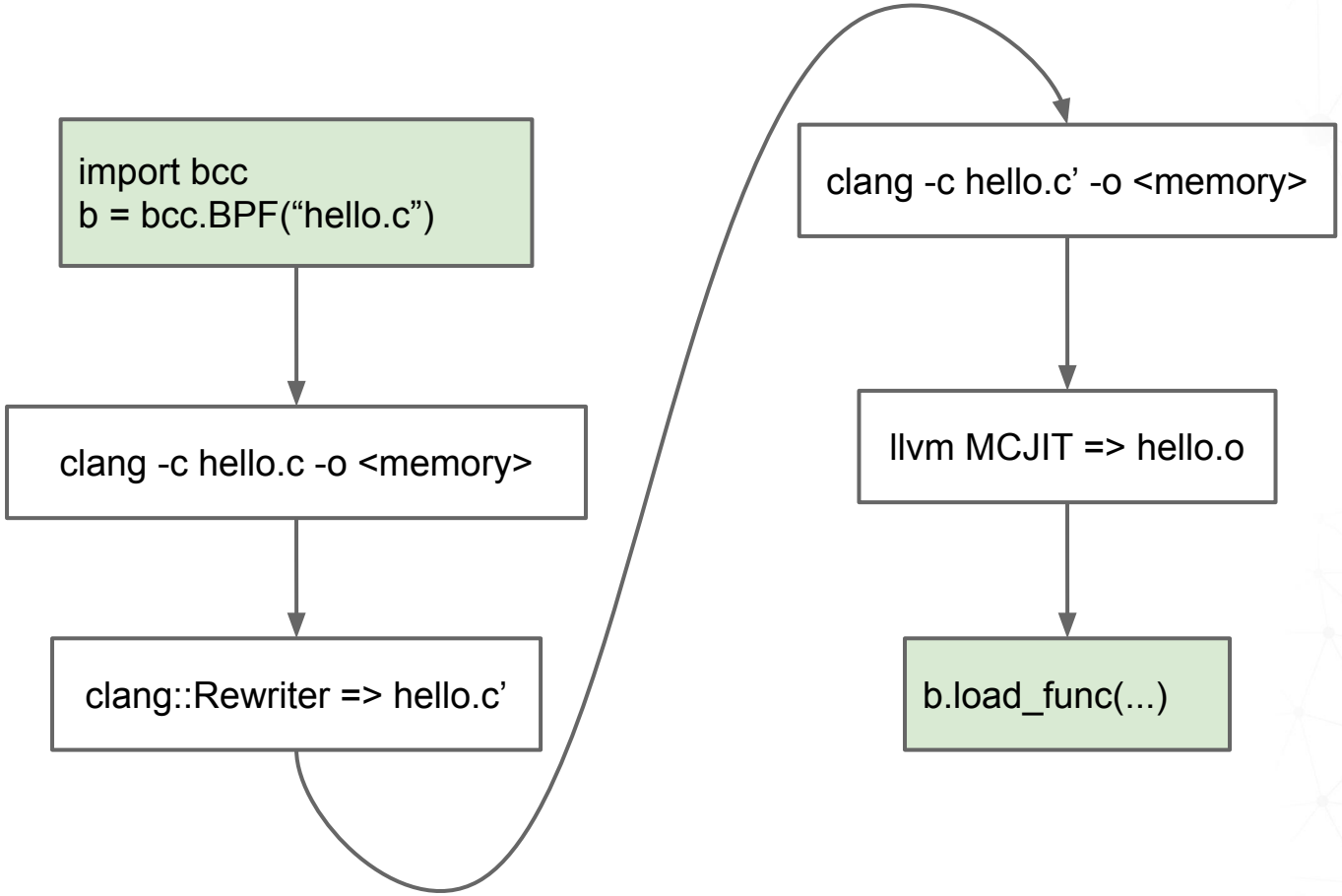
Loaded in kernel

Hooked at different levels of Linux Networking Stack (as an example)

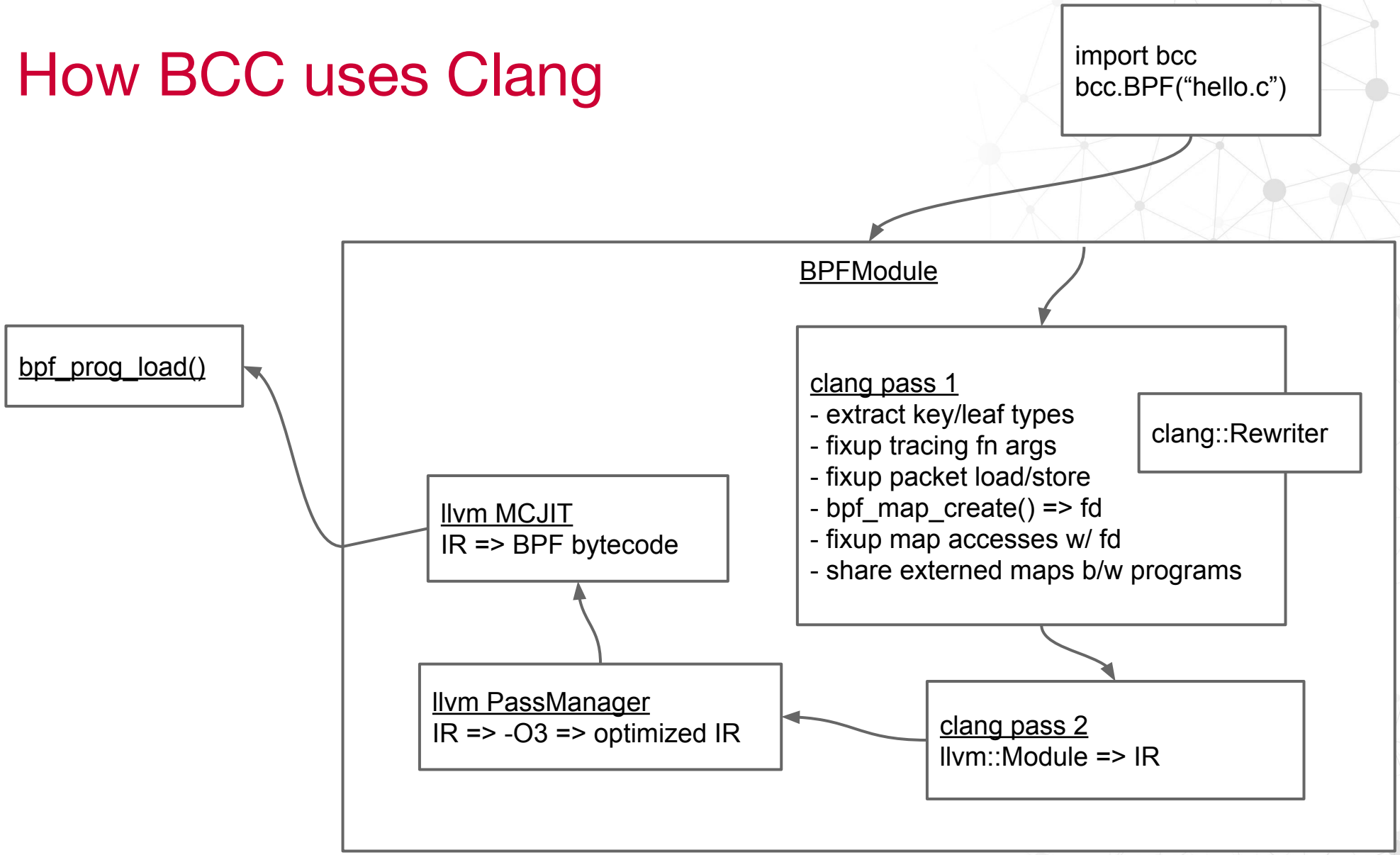


Using Clang and LLVM in BCC

How BCC uses Clang



How BCC uses Clang



Rewrite Sample #1

```
#include <uapi/linux/ptrace.h>
int do_request(struct pt_regs *ctx, int req) {
  bpf_trace_printk("req ptr: 0x%x\n", req);
  return 0;
}
```

```
#include <uapi/linux/ptrace.h>
int do_request(struct pt_regs *ctx, int req) {
  ({
    char _fmt[] = "req ptr: 0x%x\n";
    bpf_trace_printk_(_fmt, sizeof_(fmt), ((u64)ctx->di));
  });
  return 0;
}
```

Rewrite Sample #2

```
#include <linux/sched.h>
#include <uapi/linux/ptrace.h>

int count_sched(struct pt_regs *ctx,
                struct task_struct *prev) {
  pid_t p = prev->pid;
  return p != -1;
}
```

Rewrite Sample #2

```
#include <linux/sched.h>
#include <uapi/linux/ptrace.h>

int count_sched(struct pt_regs *ctx,
                struct task_struct *prev) {
  pid_t p = ({
    pid_t _val;
    memset(&_val, 0, sizeof(_val));
    bpf_probe_read(&_val, sizeof(_val),
                   ((u64)ctx->di) + offsetof(struct task_struct, pid));
    _val;
  });
  return p != -1;
}
```

Rewrite Sample #3

```
#include <bcc/proto.h>
struct IPKey { u32 dip; u32 sip; };
BPF_TABLE("hash", struct IPKey, int, mytable, 1024);
int recv_packet(struct __sk_buff *skb) {
  struct IPKey key;
  u8 *cursor = 0;
  struct ethernet_t *ethernet = cursor_advance(cursor, sizeof(*ethernet));
  struct ip_t *ip = cursor_advance(cursor, sizeof(*ip));
  key.dip = ip->dst;
  key.sip = ip->src;
  int *leaf = mytable.lookup(&key);
  if (leaf)
    *(leaf)++;
  return 0;
}
```

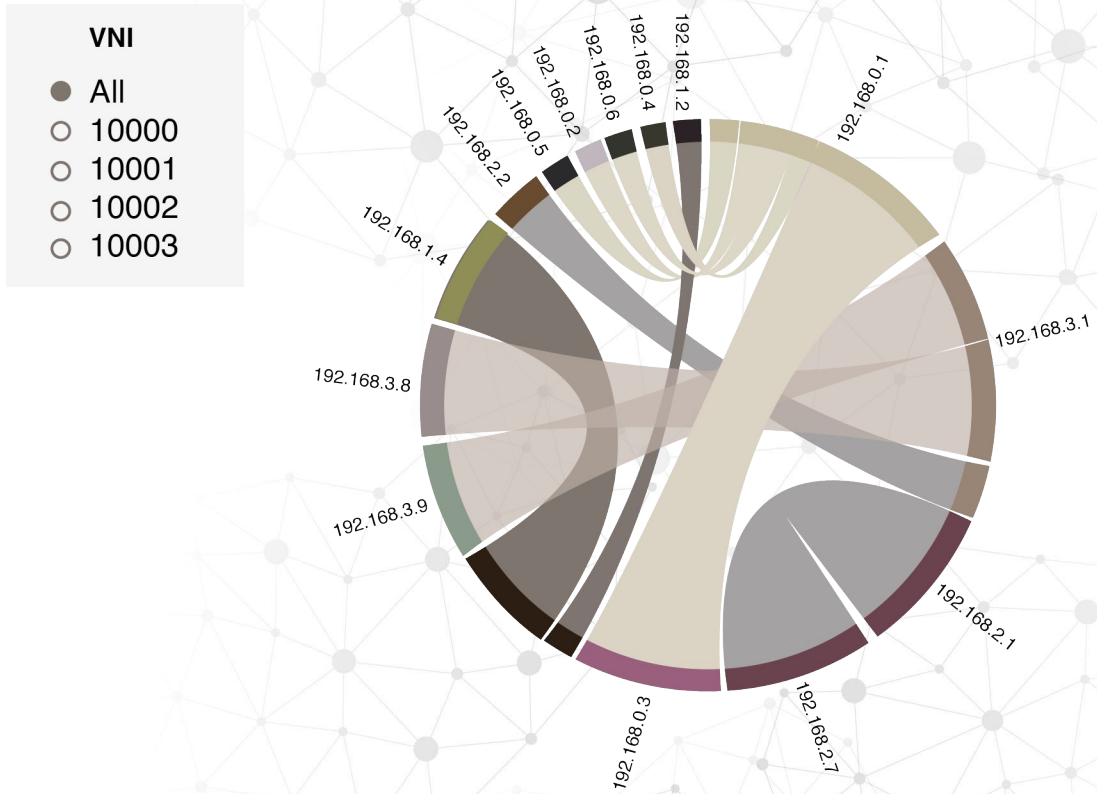
Rewrite Sample #3

```
#include <bcc/proto.h>
struct IPKey { u32 dip; u32 sip; };
BPF_TABLE("hash", struct IPKey, int, mytable, 1024);
int recv_packet(struct __sk_buff *skb) {
  struct IPKey key;
  u8 *cursor = 0;
  struct ethernet_t *ethernet = cursor_advance(cursor, sizeof(*ethernet));
  struct ip_t *ip = cursor_advance(cursor, sizeof(*ip));
  key.dip = bpf_dext_pkt(skb, (u64)ip+16, 0, 32);
  key.sip = bpf_dext_pkt(skb, (u64)ip+12, 0, 32);
  int *leaf = bpf_map_lookup_elem((void *)bpf_pseudo_fd(1, 3), &key);
  if (leaf)
    *(leaf)++;
  return 0;
}
```

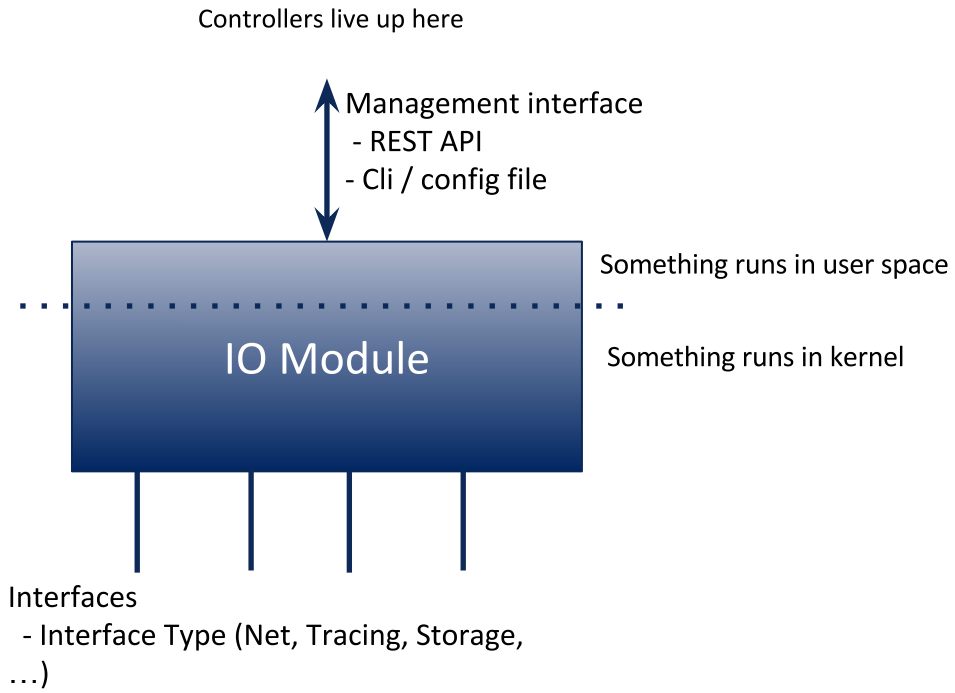
IO Modules for Networking

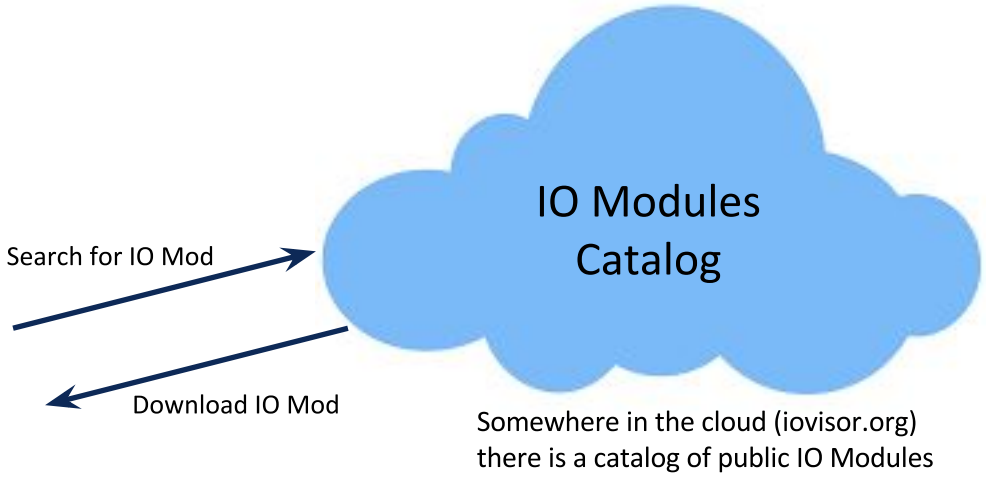
Network Analytics Demo

- IO Visor is used to build a **real-time, distributed analytics platform** that monitors the health of a VXLAN tunneling infrastructure
- Data plane component is inserted dynamically in the kernel and leveraged by the application to report information to the user
- Example here https://github.com/iovisor/bcc/tree/master/examples/networking/tunnel_monitor

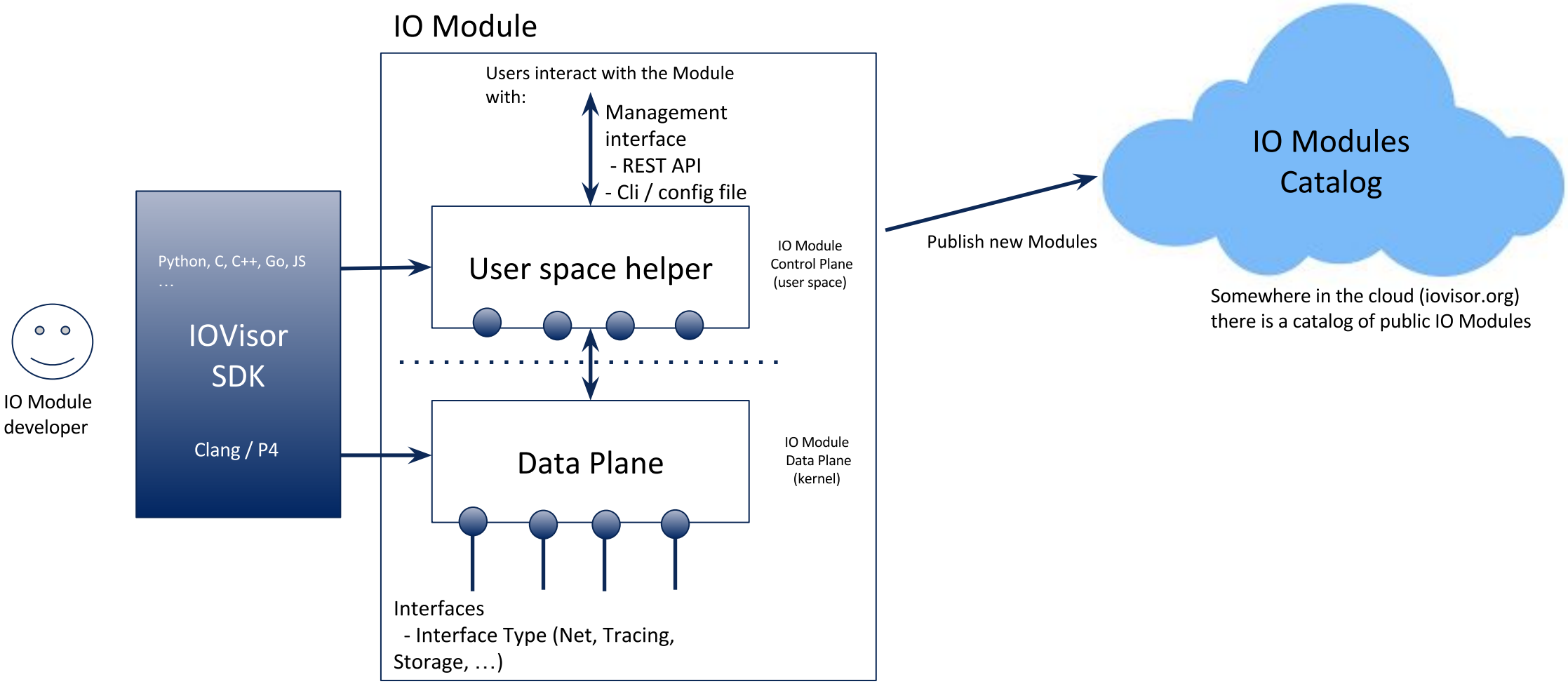


IO Module, users perspective



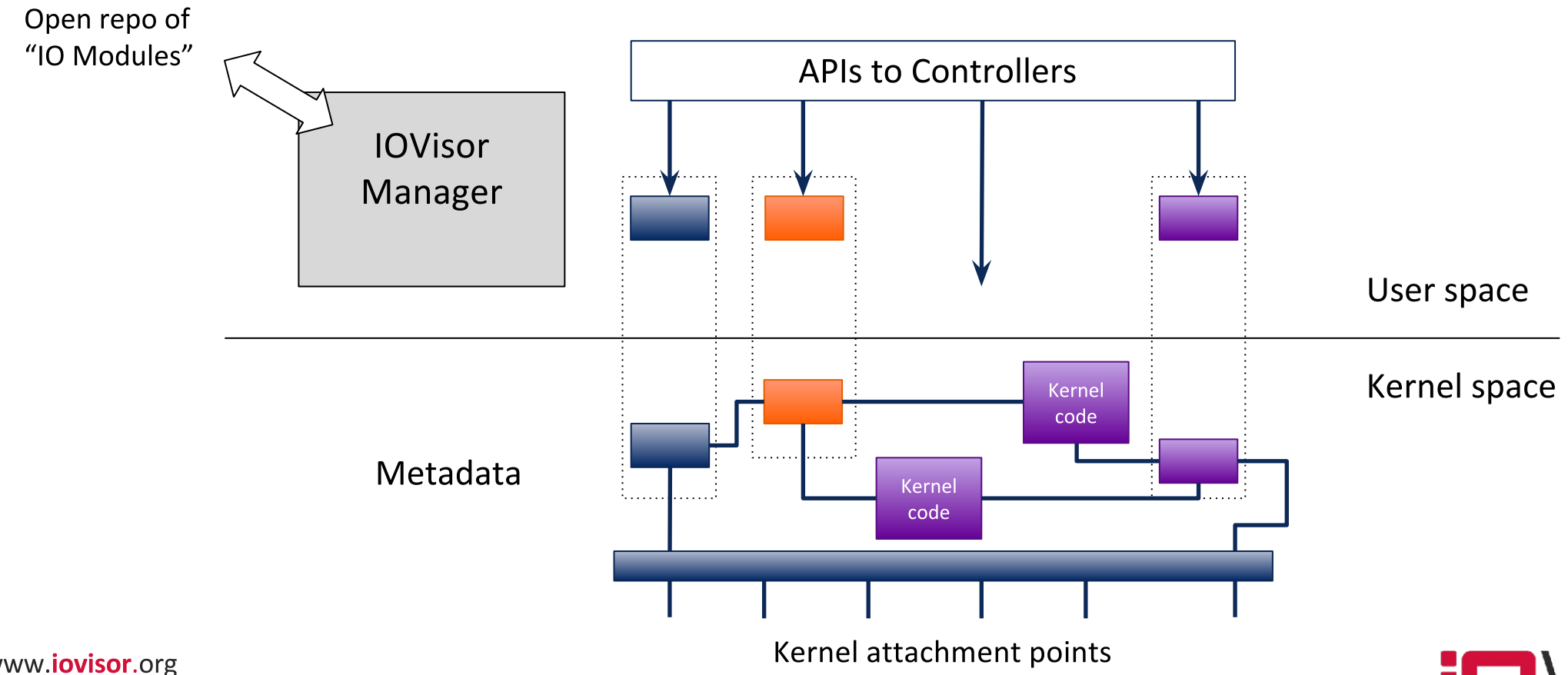


IO Module, developers perspective

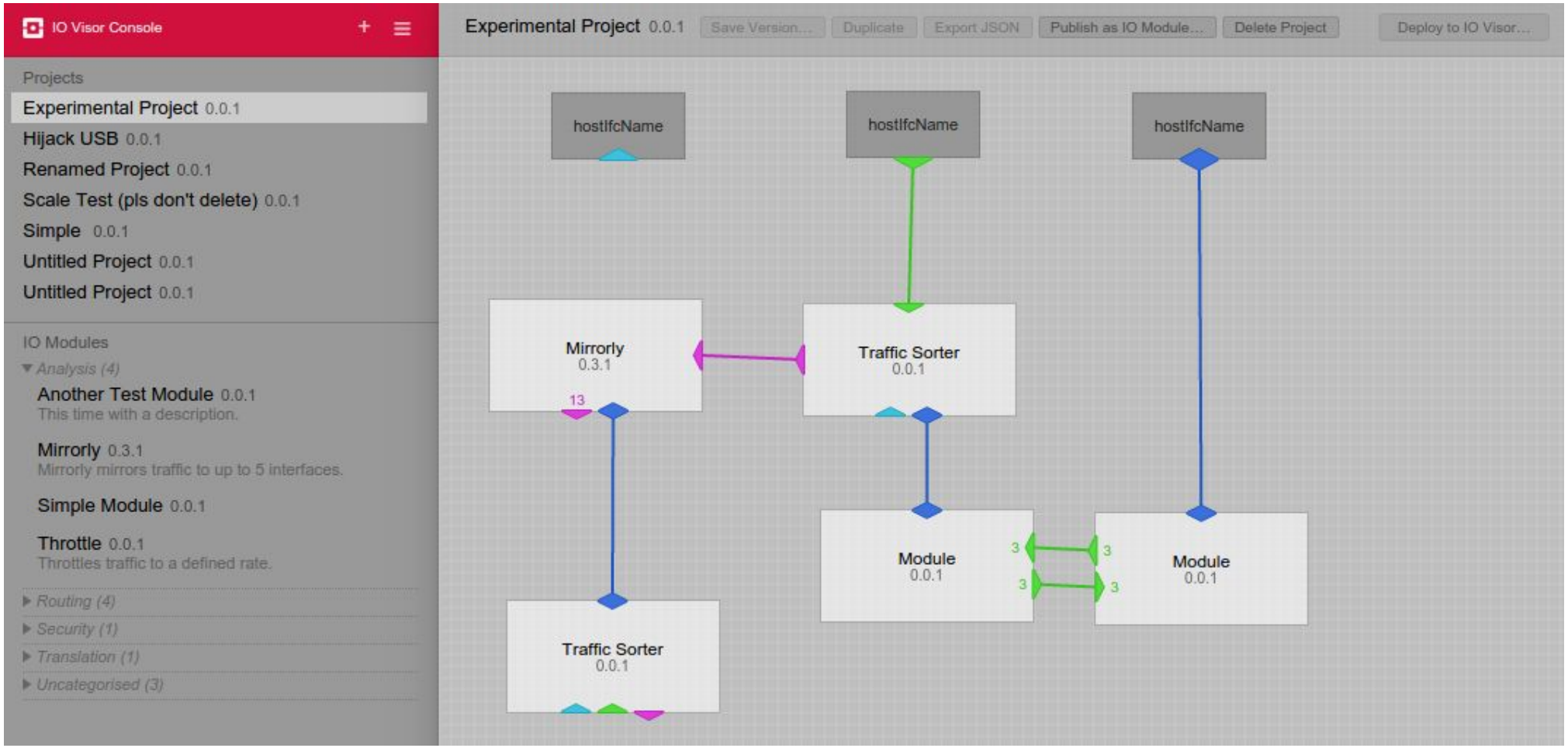


IO Module, graph composition

- extending Linux Kernel capabilities



Composing IO Modules



Using BCC for Tracing

Tracing Demo

https://github.com/iovisor/bcc

Linux eBPF Stack Trace Hack

Linux eBPF Off-CPU Flame Graph

Thank You!

Learn More and Contribute

https://iovisor.org

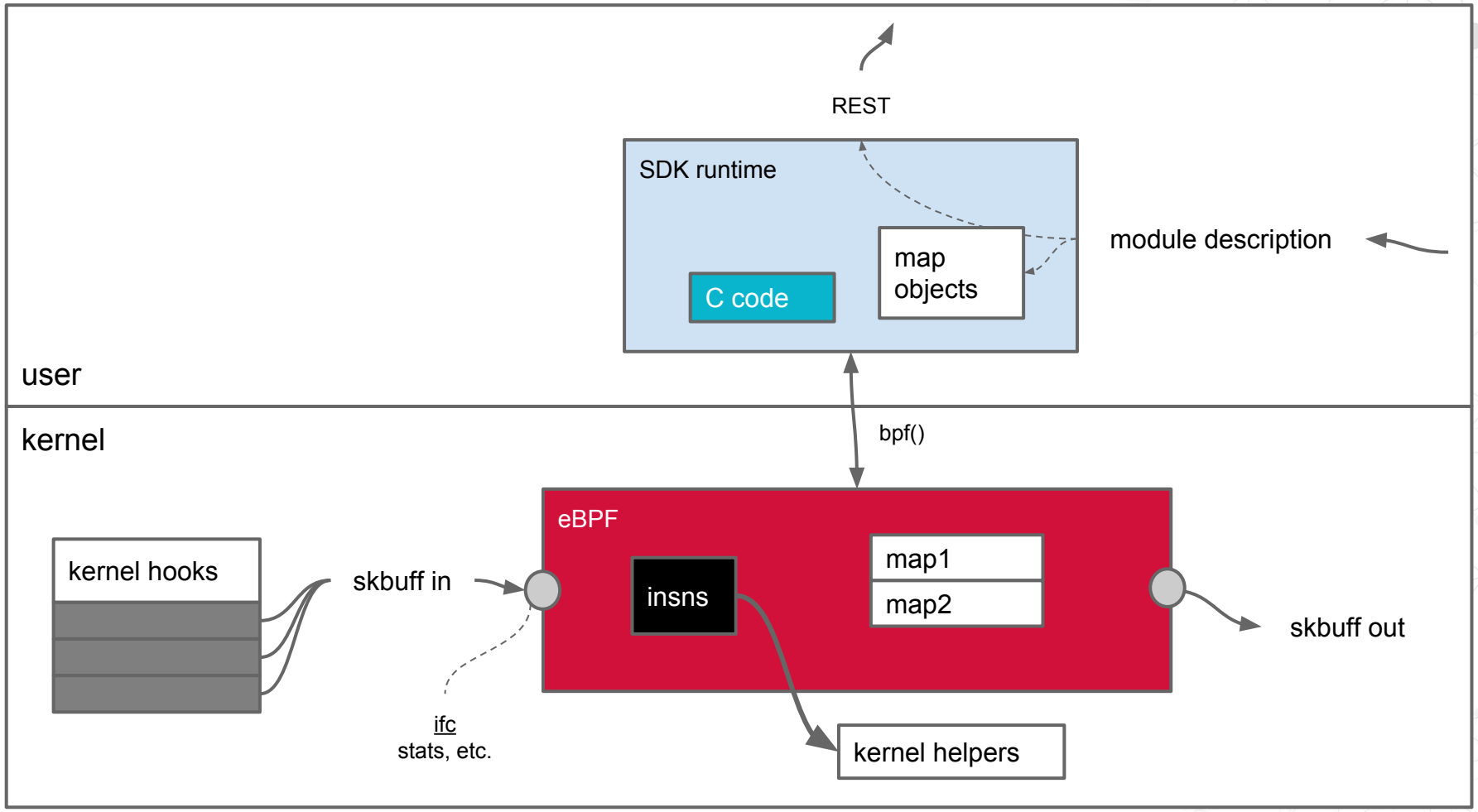
https://github.com/iovisor

#iovisor irc.oftc.net

@IOVisor

Backup Slides

Components of an IOV_Module



Introducing IO Visor Project

LINUX FOUNDATION
COLLABORATIVE PROJECTS

Evolution of Kernel BPF & eBPF
(Berkeley Packet Filter)

Led by initial contributions from PLUMgrid
(Upstreamed since Kernel 3.16)

Future of Linux Kernel IO for software defined services

*"IO Visor will work closely with the Linux kernel community to **advance universal IO extensibility for Linux**. This collaboration is critically important as virtualization is putting more demands on flexibility, performance and security.*

*Open source software and collaborative development are the ingredients for addressing massive change in any industry. **IO Visor will provide the essential framework for this work on Linux virtualization and networking**."*

Jim Zemlin, Executive Director, The Linux Foundation.

IO Visor Project: What?

1

Open Source & Community

- An open source project and a community of developers
- Enables a new way to Innovate, Develop and Share IO and Networking functions

2

Programmable Data Plane

- A programmable data plane and development tools to simplify the creation of new infrastructure ideas

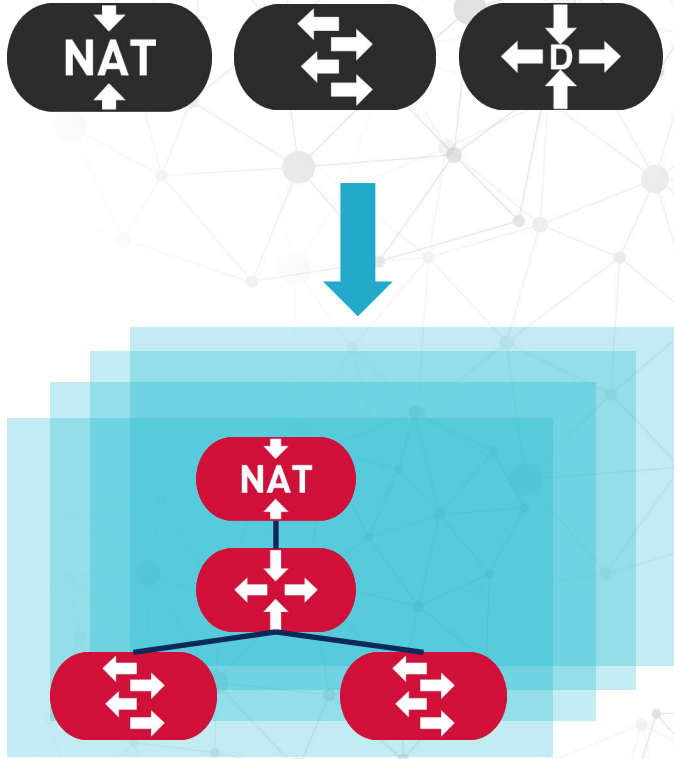
3

Repository of “IO Modules”

- A place to share / standardize new ideas in the form of “IO Modules”

IO Visor Project Use Cases Example: Networking

- IO Visor is used to build a fully distributed virtual network across multiple compute nodes
- All data plane components are inserted dynamically in the kernel
- No usage of virtual/physical appliances needed
- Example here https://github.com/iovisor/bcc/tree/master/examples/distributed_bridge



Virtual/Physical Appliances

Virtual Network Topology in Kernel Space