

eBPF and IO Visor: The what and how!

Affan A. Syed

**Director Engineering,
PLUMgrid Inc.**

Talk outline

Outline

- Berkeley Packet Filter (BPF) and bytecode
- Extended BPF (eBPF): Motivation and features
- BCC and IO Visor
- USDT tracing of Node.js application (demo)

Packet Filters and cBPF

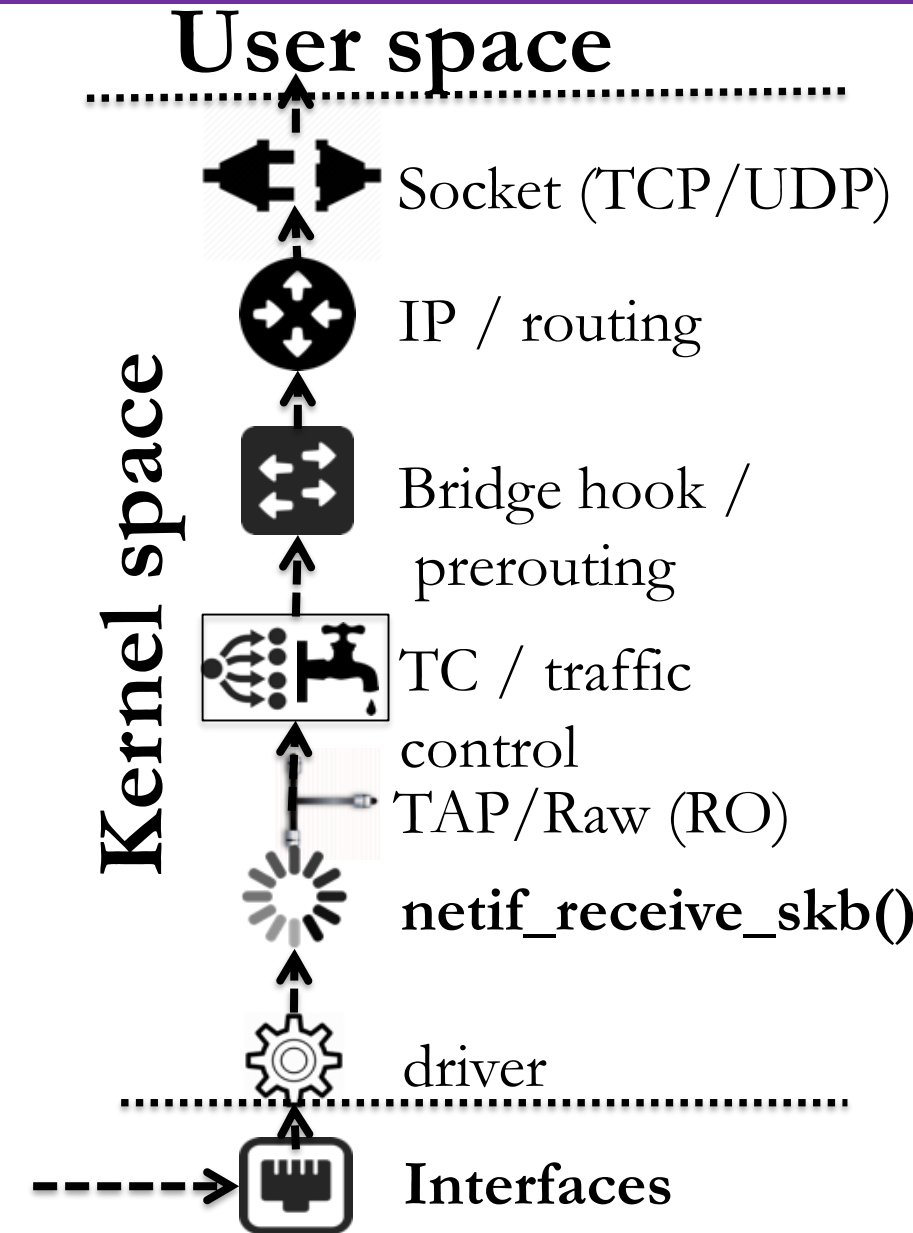
Packet Filters

Objective: To observe **all** traffic but capture only a subset

Problem: Packet traversal through normal stack is slow

Solution: Setup filters **in kernel** where packet dropped if not match

....but these filters need to be secure!



Tcpdump our friend

Lets do: `sudo tcpdump -p -ni eth0 "ip and udp"`

Now lets do: `$ sudo tcpdump -p -ni eth0 -d "ip and udp"`

```
(000) ldh    [12]
(001) jeq    #0x800      jt 2   jf 5
(002) ldb    [23]
(003) jeq    #0x11      jt 4   jf 5
(004) ret    #65535
(005) ret    #0
```

This code runs for every packet that arrives on eth0

The concept of pseudo-machine

Think Java --- but don't think VM !

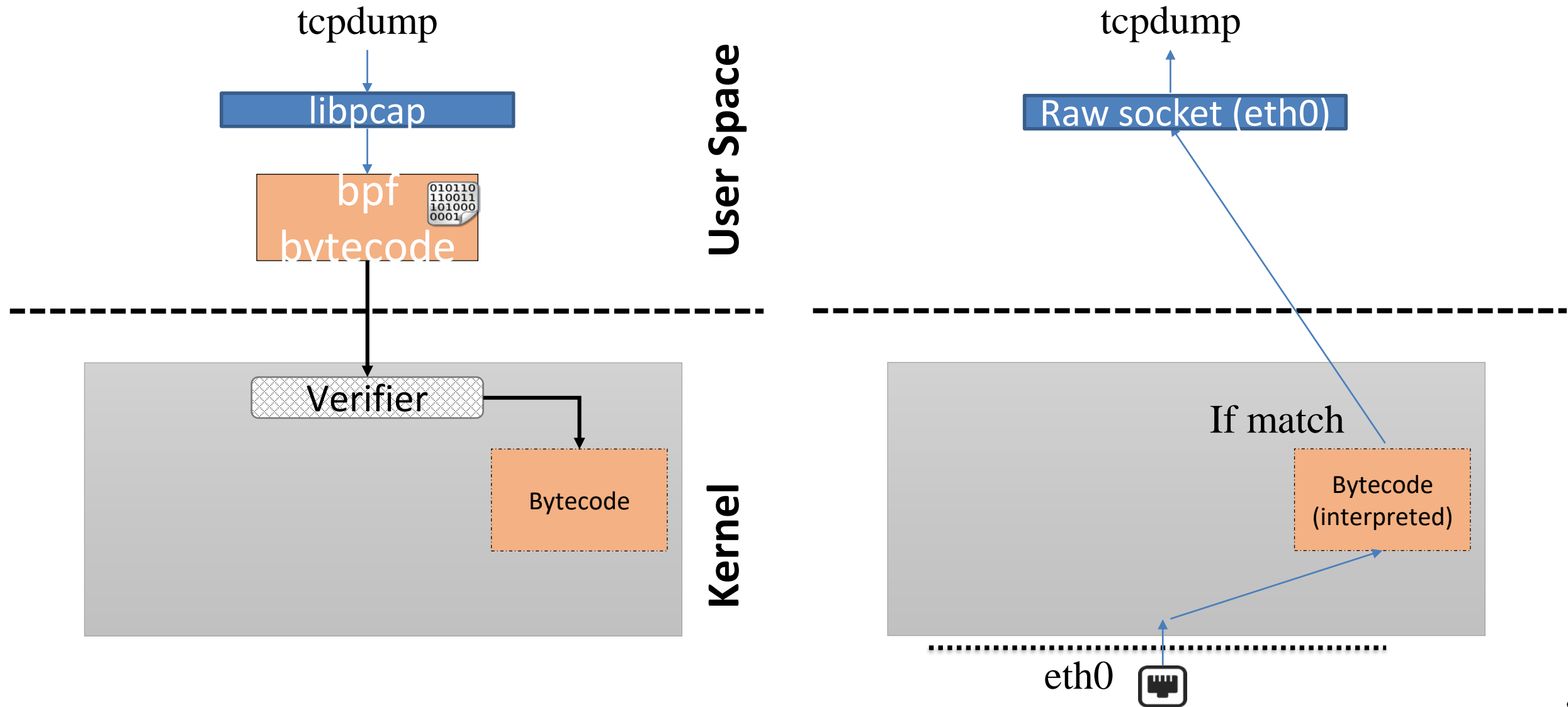
Virtualize a machine instruction set

Write byte code for this “fictional” machine

***verify** this code is loop free and optimized*

Interpret, in-kernel, for **any** real processor

BPF Overview (insertion and usage)



Other uses and extension

Slowly evolving cBPF

seccomp support for sandboxing

tc filter for traffic shaping

JIT compiler

Extending BPF

... while building a programmable Data Plane

Goals for a programmable Data Plane = eBPF

Enable packet parsing, lookup, modification, and updates

Guarantee safety of code run on a production system

Native performance

Enhanced architecture

classic BPF	extended BPF
2 registers + stack 32-bit registers 4-byte load/store to stack 1-4 byte load from packet Conditional jump forward +, -, *, ... instructions	10 registers + stack 64-bit registers with 32-bit sub-registers 1-8 byte load/store to stack, maps, context Same + store to packet Conditional jump forward and backward Same + signed_shift + endian Call instruction tail_call map lookup/update/delete helpers packet rewrite, csum, clone_redirect sk_buff read/write

- Can build more complicated program
- Faster interpretation and JIT
- Support for calls to *approved* helper functions

Maps

Maps = <key, value> storage

Save state across invocation of programs in kernel = state machines!

Example: *fd bpf_table_lookup(table_id, key)*

Userspace can create/access/delete these maps (using bpf syscall)

loosely coupled communication between user and kernel space

Maps can be shared between eBPF programs

HASH, ARRAY ... and growing

Stateful programmability and async interaction with user space

Helper functions and tail calls

Invoke sanitized functions from within the eBPF program

like a library – but ... of course .. In-kernel

*e.g u64 bpf_ktime_get_ns(void), int bpf_trace_printk(const char *fmt, int fmt_size, ...), u32 prandom_u32(void)*

Tail call feature a combo of two components

bpf_tail_call(ctx, prog_array_map, index)

and PROG_ARRAY_MAP

increased capabilities, and sanitized access

Verifier and kernel safety

- eBPF new architecture more complex
*required a **brand** new verifier*
- Provably confirms inserted program **does not**:
*create loops, delays execution interminably, illegally
dereference pointers*
- Done statically, one-time, with an exhaustive search
some heuristics to improve verification time

WITH GREAT
POWER
COMES GREAT
RESPONSIBILITY

PETER PARKER is SPIDER MAN

LLVM compiler, Interpreter, and JIT

Restricted “C” code that compiles to bpf bytecode

LLVM backend for this purpose with clang frontend

Once inserted, the code is “hooked” to a kernel event

no sense hooking to userspace events!

On event firing the appropriate code is run in either

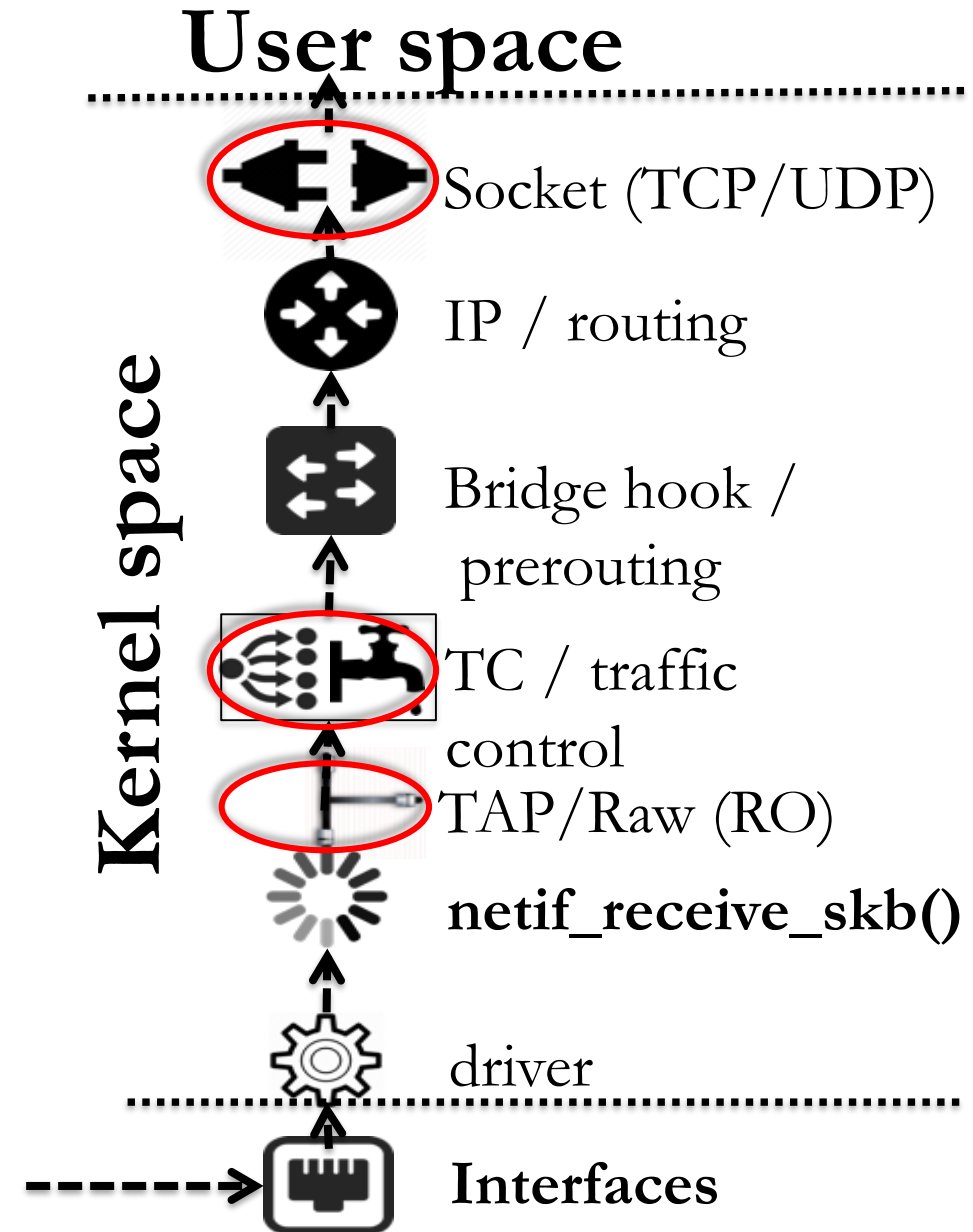
native or interpreted mode

Example Hooks: The networking stack

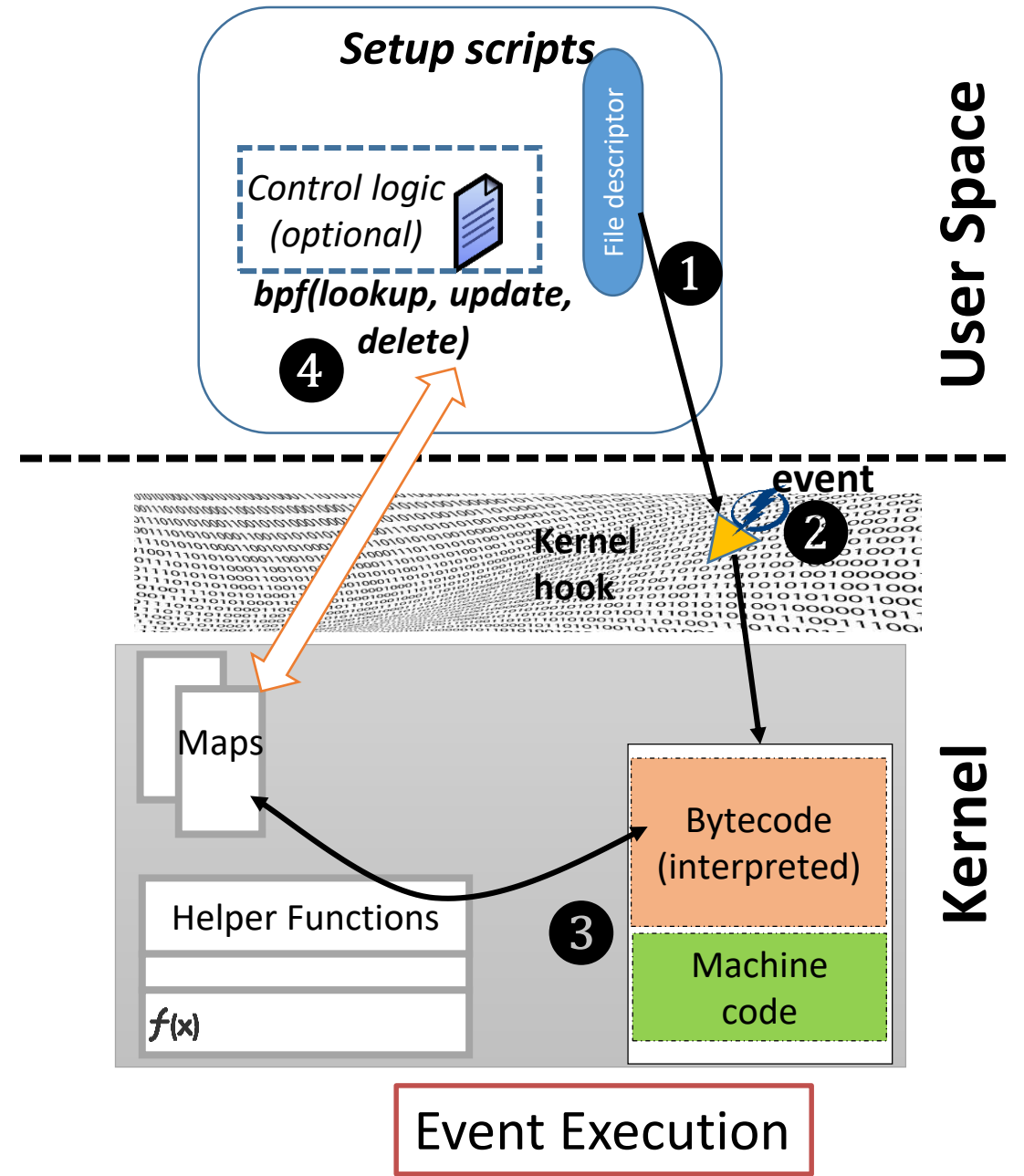
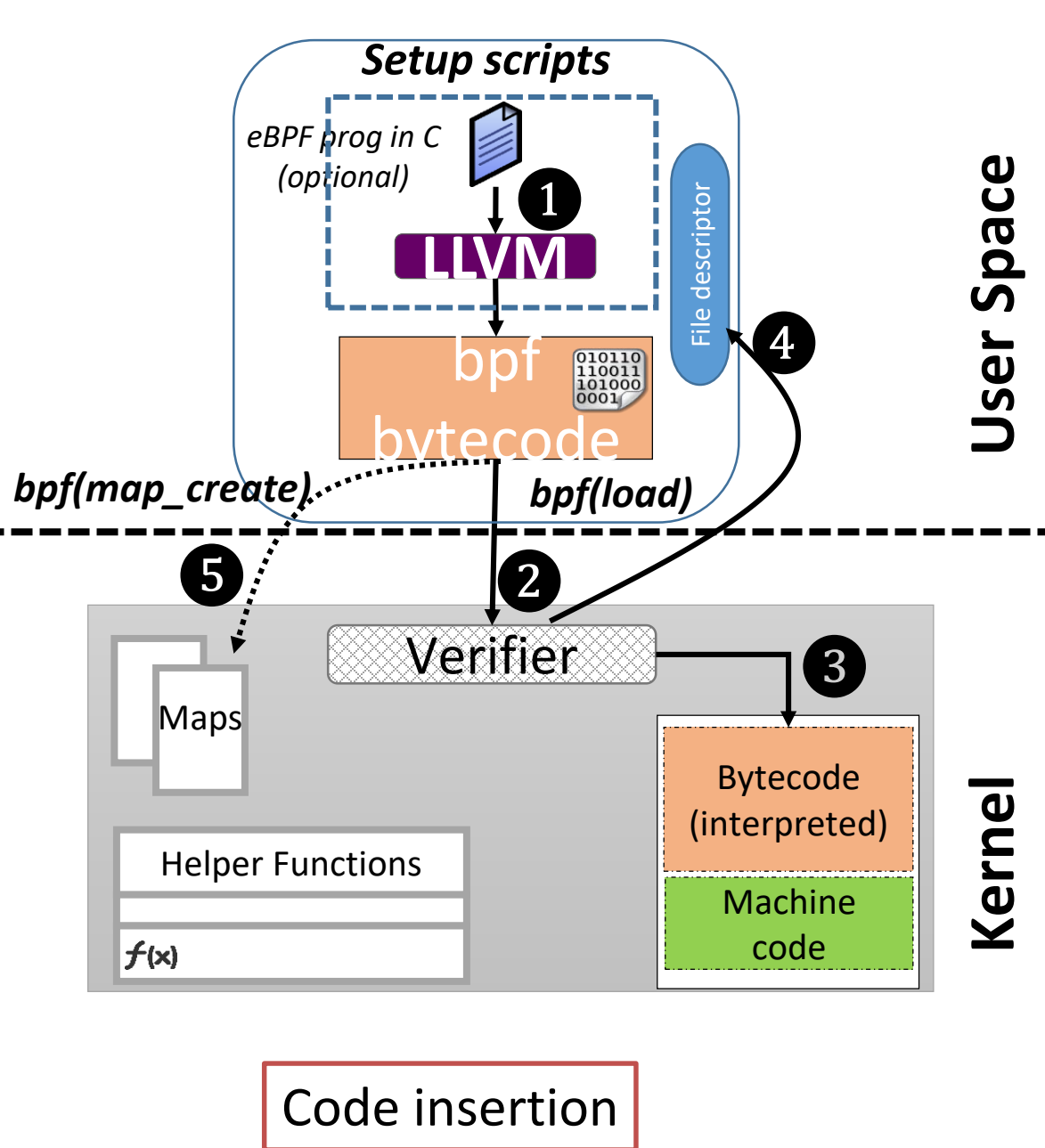
traffic control (TC):
queues (classification or
action time)

sockets: STREAM
(L4/UDP), DATAGRAM
(L4/TCP) or RAW

others: kprobes, syscalls,
tracepoints ...



Visual Flow of code insertion and use



eBPF, IO Visor and BCC

Complexity of making eBPF code

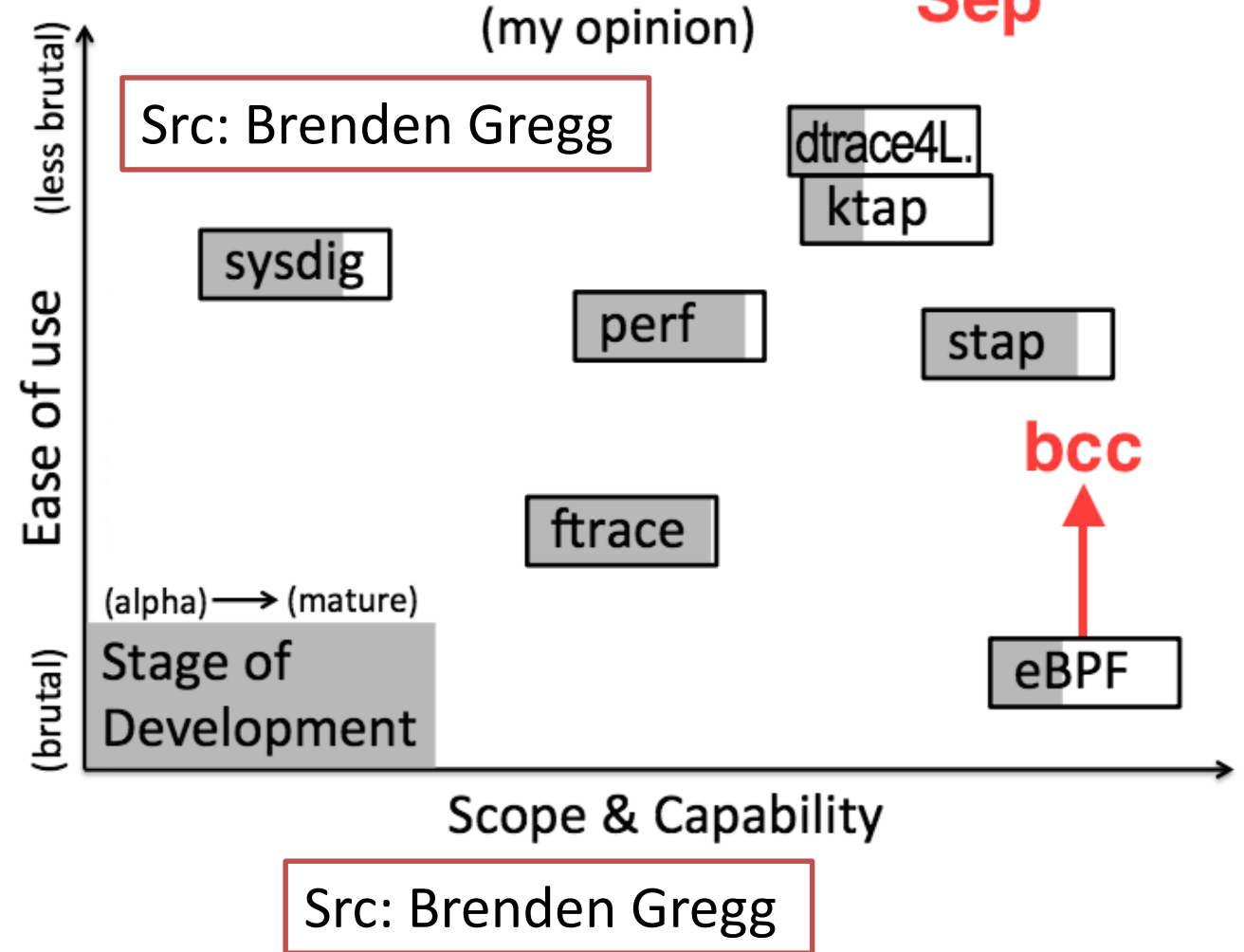
Writing eBPF programs was “brutal”

Even with compiler, the map/code sharing

Enter BPF compiler collection (BCC)

The Tracing Landscape, ~~May~~ 2015

Sep



Make using features of eBPF easier to use

Python front-end and scripts to

create/access/delete maps

load programs from a restricted “C” format

attach to different locations with a simple API

BCC and a few screen-shots!



BPF Compi

BCC is a toolkit for creating examples. It makes use of € of what BCC uses requires

Kernel requirements

Requirements

In general, to use these features, a Linux kernel version 4.1 or newer is required. In addition, the following flags should be set:

```
CONFIG_BPF=y
CONFIG_BPF_SYSCALL=y
# [optional, for tc filters]
CONFIG_NET_CLS_BPF=m
# [optional, for tc actions]
CONFIG_NET_ACT_BPF=m
CONFIG_BPF_JIT=y
CONFIG_HAVE_BPF_JIT=y
# [optional, for kprobes]
CONFIG_BPF_EVENTS=y
```

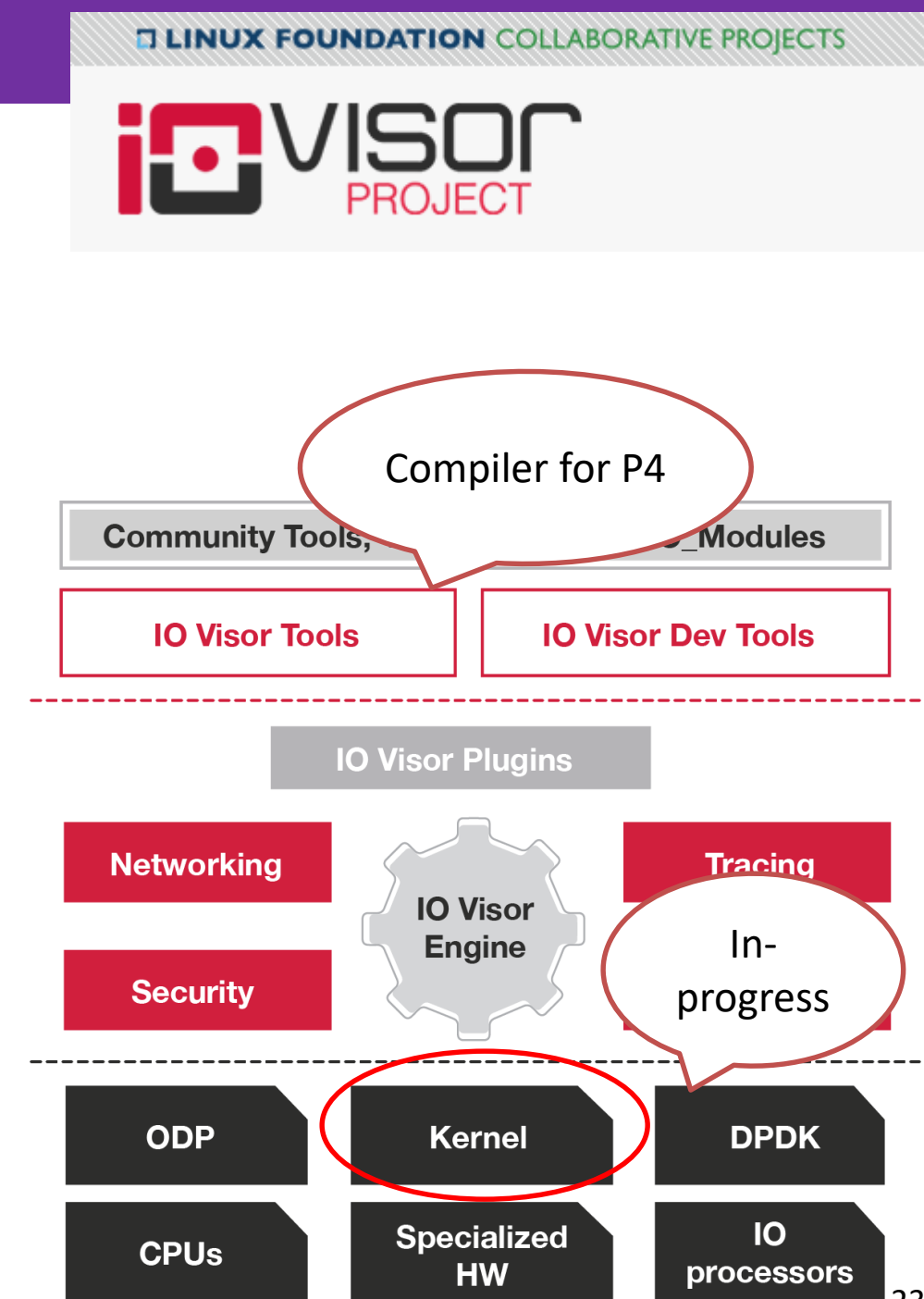
IO Visor Project

Linux Foundation Collaborative Project

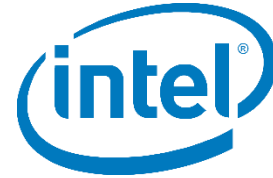
IO Visor Engine is an abstraction of an IO execution engine

A set of development tools, **IO Visor Dev and Management**

A set of **use cases & applications** like Networking, Security, Tracing & others

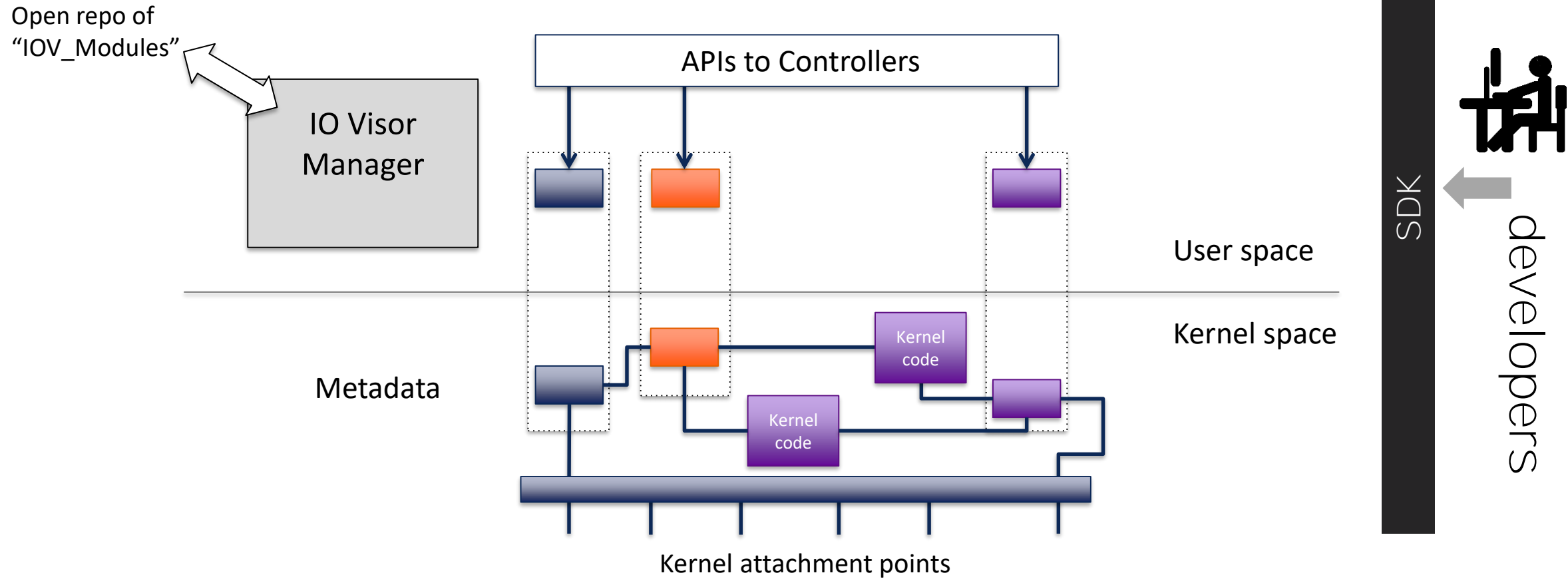


Founding Members



www.iovisor.org

IO Visor Modules



Lots of interesting projects there right now

bpf-based file system (FUSE)

eXtensible Data Plane (XDP)

Join and contribute!

Useful links

- <https://github.com/iovisor/bcc>
- <https://github.com/iovisor/bpf-docs>
- <http://lwn.net/Articles/603984/>
- <http://lwn.net/Articles/603983/>
- <https://lwn.net/Articles/625224/>
- <https://www.kernel.org/doc/Documentation/networking/filter.txt>
- <http://man7.org/linux/man-pages/man2/bpf.2.html>
- https://linuxplumbersconf.org/2015/ocw//system/presentations/3249/original/bpf_llvm_2015aug19.pdf
- https://videos.cdn.redhat.com/summit2015/presentations/13737_an-overview-of-linux-networking-subsystem-extended-bpf.pdf
- <https://github.com/torvalds/linux/tree/master/samples/bpf>
- <http://events.linuxfoundation.org/sites/events/files/slides/tracing-linux-ezannoni->
- https://www.kernel.org/doc/Documentation/prctl/seccomp_filter.txt
- http://lxr.free-electrons.com/source/net/sched/cls_bpf.c

Live Demo

... and prayers!

<https://github.com/goldshn/linux-tracing-workshop/blob/master/bpf-usdt.md>

Research Threads

... for the adventurous amongst you!

Networking and packet manipulations

Increasing application response time

frequent /cached responses in kernel

Container Networking

custom encapsulation protocols and metadata

State-full QoS

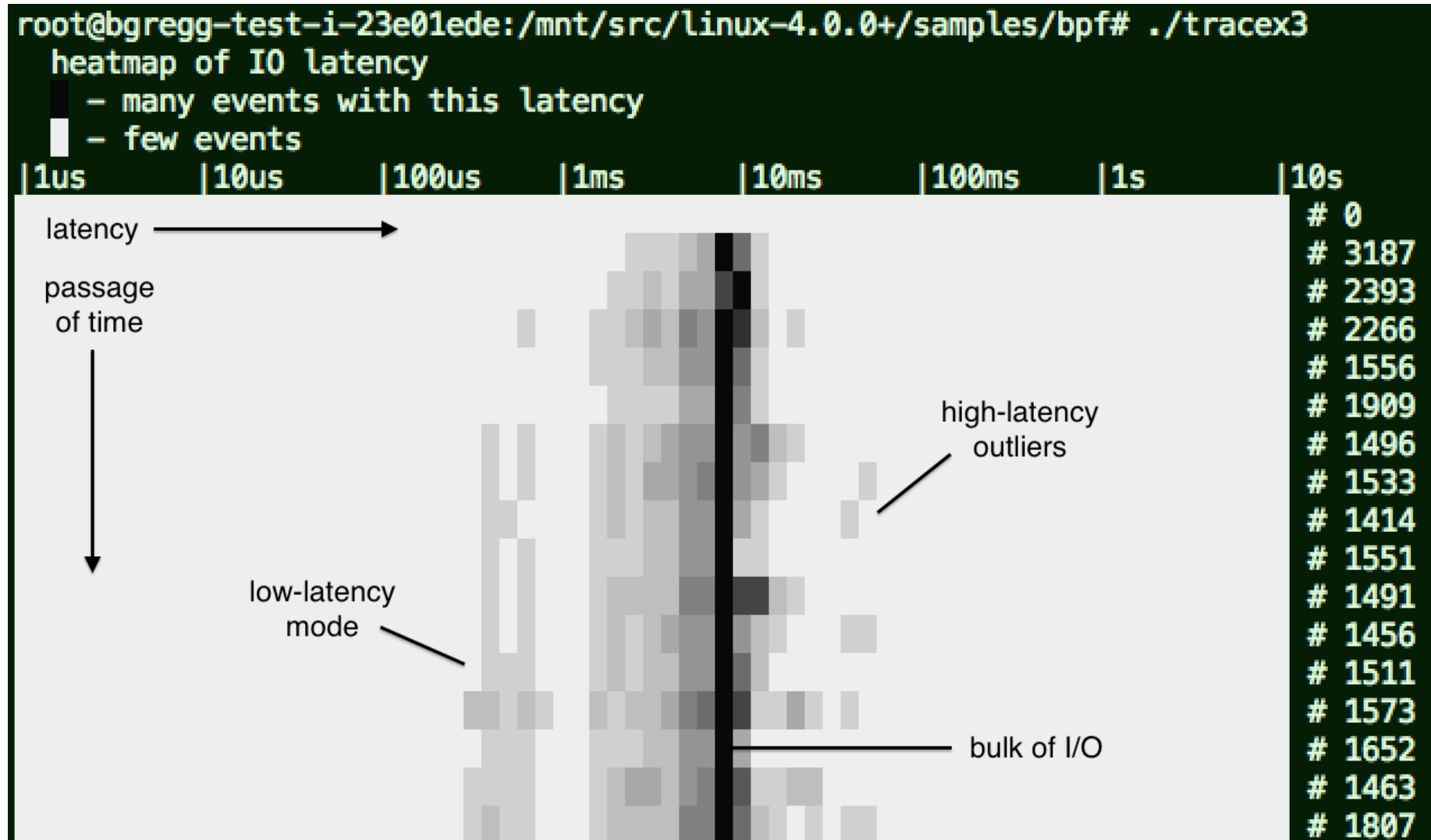
Fast, flexible and state-full Firewalls

System call trapping and *dynamic* taint analysis

Faster and less resource hungry analysis

Disk latency heat-maps

<https://github.com/torvalds/linux/tree/master/samples/bpf/>



One map for time-stamp, other for latency

Event-based, packet-based micro-kernel abstraction

*Saving energy with operation in-kernel,
user-space tools for config/mgmt/debug
think tinyOS-inside-Linux*

Software defining an API for heterogenous and
opportunistic low-power coms

wifi, zigbee, Z-wave

Thank you!
Questions?