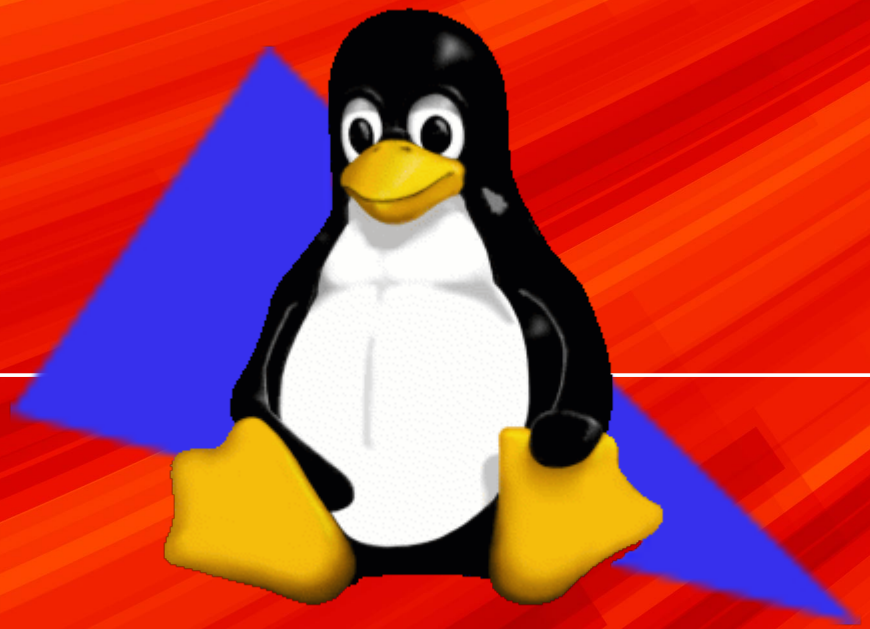
bpfTrace – Finally DTrace Replacement on Linux

Is not that great ?



Peter Zaitsev, CEO Percona

February 11th, 2021

TriLUG – Triangle Linux Users Group

PERCONA

About the Presentation

Dtrace and eBPF Refresher

eBPF Tools Landscape

Look at BPFTrace

Instrumentation Basics

Observability

Being Able to See Inside Running System

Critical for System Operation Monitoring, Troubleshooting, Performance Optimization

Achieved through Instrumentation

Instrumentation

Capturing Information from the Running system

There is Static instrumentation and Dynamic Instrumentation

The Instrumentation Approach

“Tracing”

- Emitting event when particular code point is reached

“Sampling”

- Checking the system state (ie program stack) at periodic interval

Static vs Dynamic Instrumentation

Static

- Counters, Logging points etc placed throughout the code
- Need to be mindful about overhead
- Limited Depth

Dynamic

- Dynamically chose what you want to instrument with running system
- Dynamically change level of instrumentation
- Can go very deep

DTrace

DTrace

Dynamic Tracing Framework

Developed Sun Microsystems starting 2001

Released in Solaris 10 in 2005

Define Specific Tracepoints in Kernel and User Land

Trace Function Calls and More

No Overhead than not enabled

D Language (Inspired by C and Awk)

DTrace Beyond Solaris

MacOS 10.5 (2007)

FreeBSD (2008)

NetBSD (2010)

Oracle Linux Supported dTrace since 2011

Code Re-Licensed GPLv2+ by Oracle in 2017

Dtrace is available on Windows in 2019

DTrace on Linux

Is not available in stock Linux Kernel

Is not available from major Linux Distributions

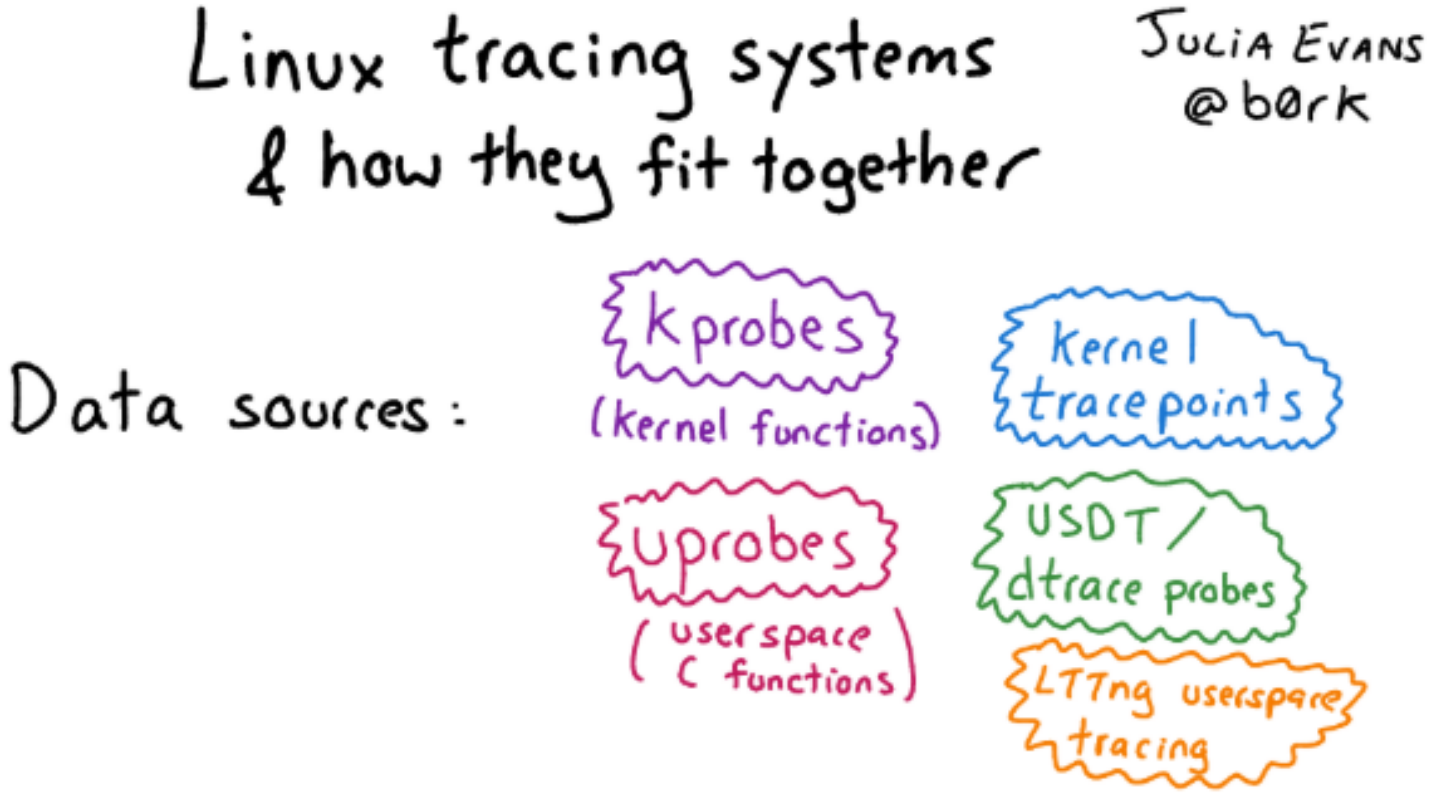
Recent GPL Code release is likely too little too late

Tracing in Linux

Linux Tracing

Many competing tracing frameworks and frontends rather than single one

Linux Tracing in Pictures



Source: https://jvns.ca/blog/2017/07/05/linux-tracing-systems/

PERCONA

Linux Tracing Infrastructure

The Type of Kernel Interface	• Kprobe, uprobe, Dtrace probe etc
The Type of "Program" Connected to it	• Built in Kernel Buffer, Kernel Module, eBPF
Front-end Tools to work with it from the user space	• Perf, SystemTap, SysDig, Bcc etc

eBPF – emerging Linux Standard

eBPF - Extended Berkeley Packet Filter

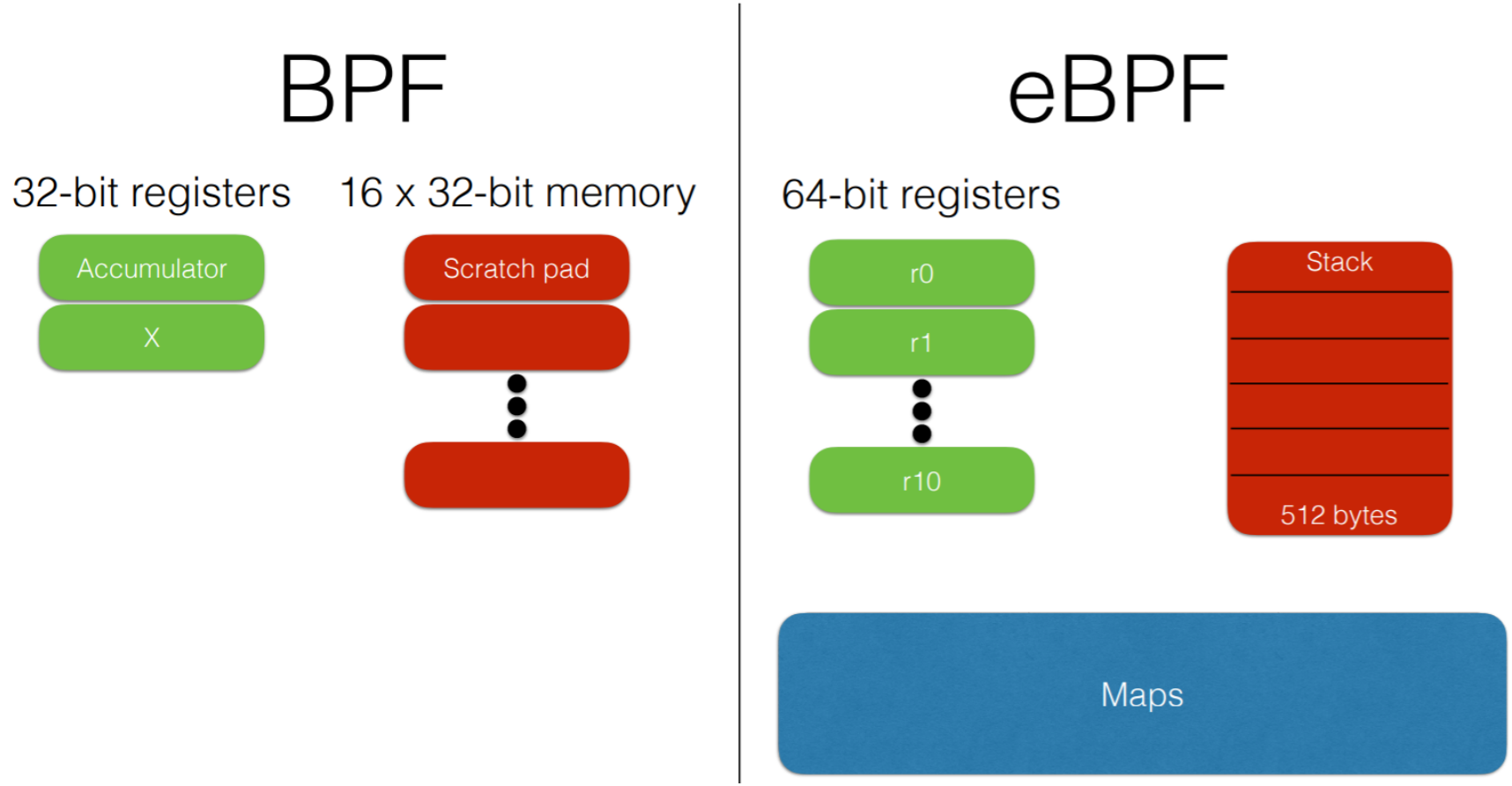
Berkeley Packet Filter - Originated in 1992 as efficient virtual machine for Packet Filtering

Extended Berkeley Packet Filter – Extended Version found in Linux

General Event Processing Framework

JIT Compiler for high efficiency

eBPF vs BPF



PERCONA

eBPF in Linux

Has been in Linux Kernel since 2014

Actively being improved

Integrated in “perf” tooling system

Improvements in recent Kernels

bpf2bpf function calls	4.16	`cc8b0b92a169`
BPF used for monitoring socket RX/TX data	4.17	`4f738adba30a`
BPF attached to raw tracepoints		
BPF attached to `bind()` system call		
BPF Type Format (BTF)		
AF_XDP		
bpfilter		
End.BPF action for seg6local LWT		
BPF attached to LIRC devices		

BPF attached to `bind()` system call	4.17	`4fbac77d2d09`
BPF Type Format (BTF)	4.18	`69b693f0aefa`
AF_XDP	4.18	`fbfc504a24f5`
bpfilter	4.18	`d2ba09c17a06`
End.BPF action for seg6local LWT	4.18	`004d4b274e2a`
BPF attached to LIRC devices	4.18	`f4364dcfc86d`
BPF socket reuseport	4.19	`2dbb9b9e6df6`
BPF flow dissector	4.20	`d58e468b1112`
BPF cgroup sysctl	5.2	`7b146cebe30c`
BPF raw tracepoint writable	5.2	`9df1c28bb752`

https://github.com/iovisor/bcc/blob/master/docs/kernel-versions.md

PERCONA

eBPF Programs

Linux Kernel can load programs in custom byte code

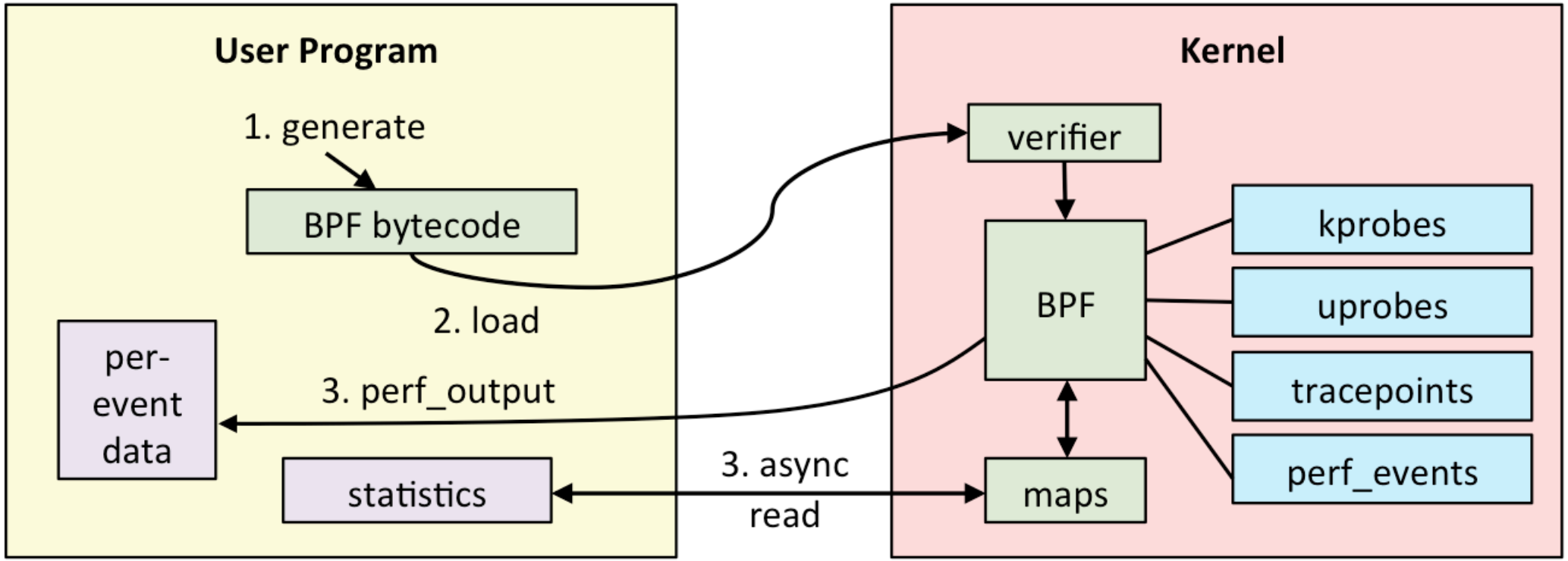
Programs verified before load to prevent misuse

LLVM Clang can compile to eBPF byte code

This compilation is kernel-dependent

Few will need to write eBPF programs Directly

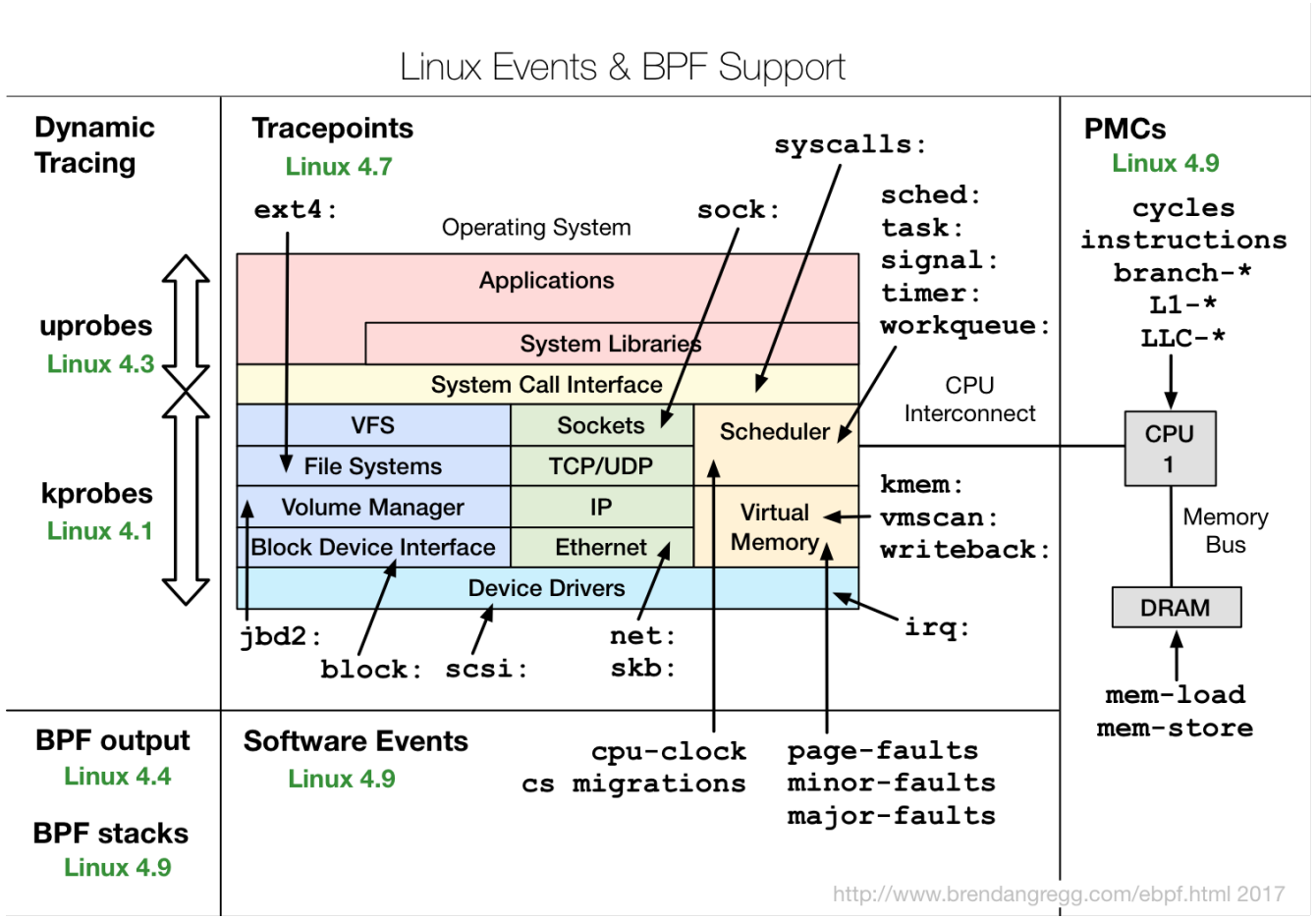
eBPF User Space vs Kernel



Source: http://www.brendangregg.com/ebpf.html

PERCONA

eBPF features in different kernel versions



PERCONA

Project to Know

https://github.com/iovisor

eBPF Overhead

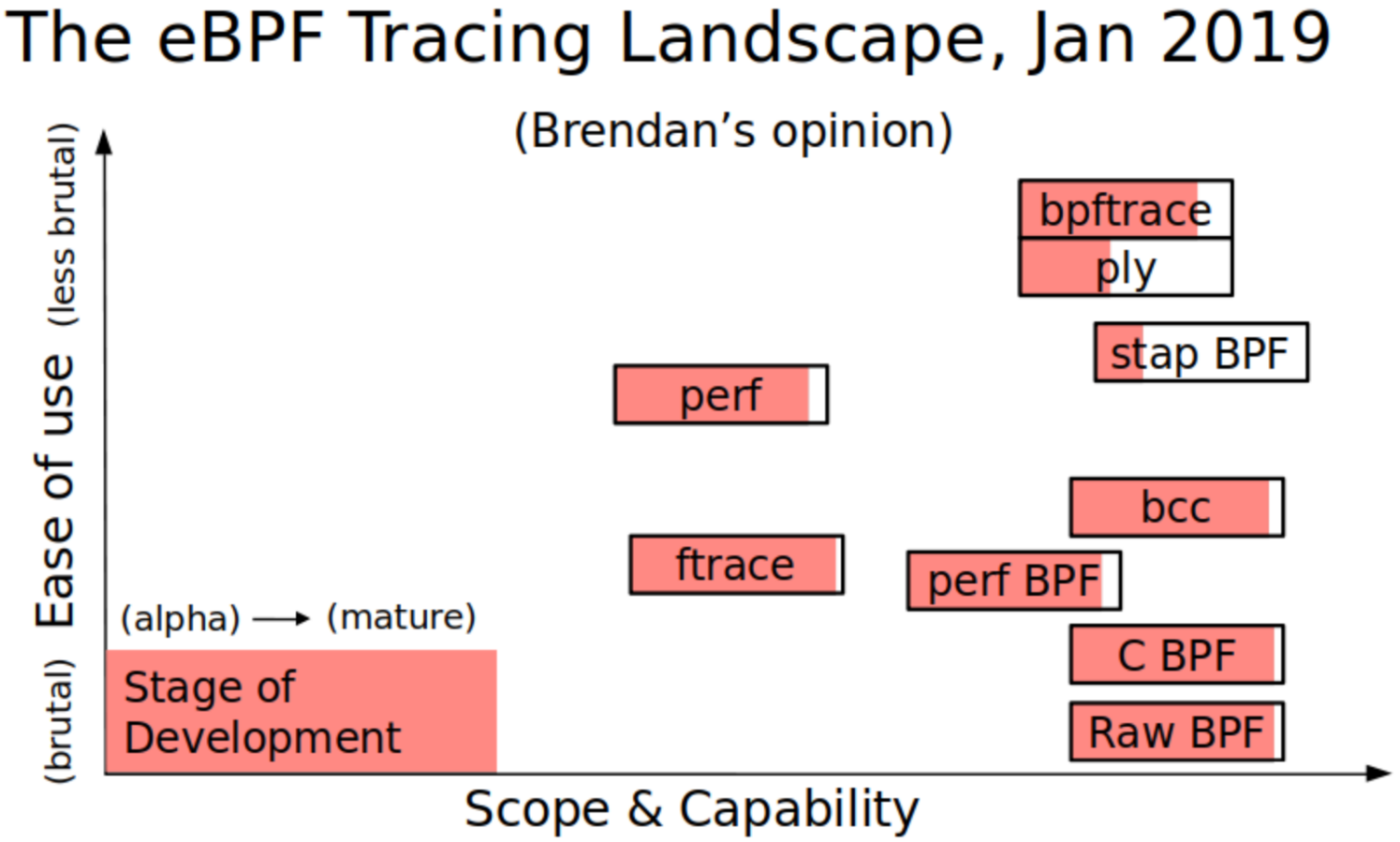
eBPF Programs can be run million+ times per second per core

Case	ns/op	overhead ns/op	ops/s	overhead percent
no probe	316	0	3,164,556	0%
simple	424	108	2,358,490	34%
complex	647	331	1,545,595	105%

More Details: https://github.com/cloudflare/ebpf_exporter/tree/master/benchmark

eBPF Frontends

Tracing Landscape per Brendan Gregg



http://www.brendangregg.com/ebpf.html

PERCONA

Most Valuable

BCC

- Has Great set of Pre-Built tools
- Tricky to Develop your own Tools

BpfTrace

- Much Easier to use Language
- Collection of Tools is being Built out

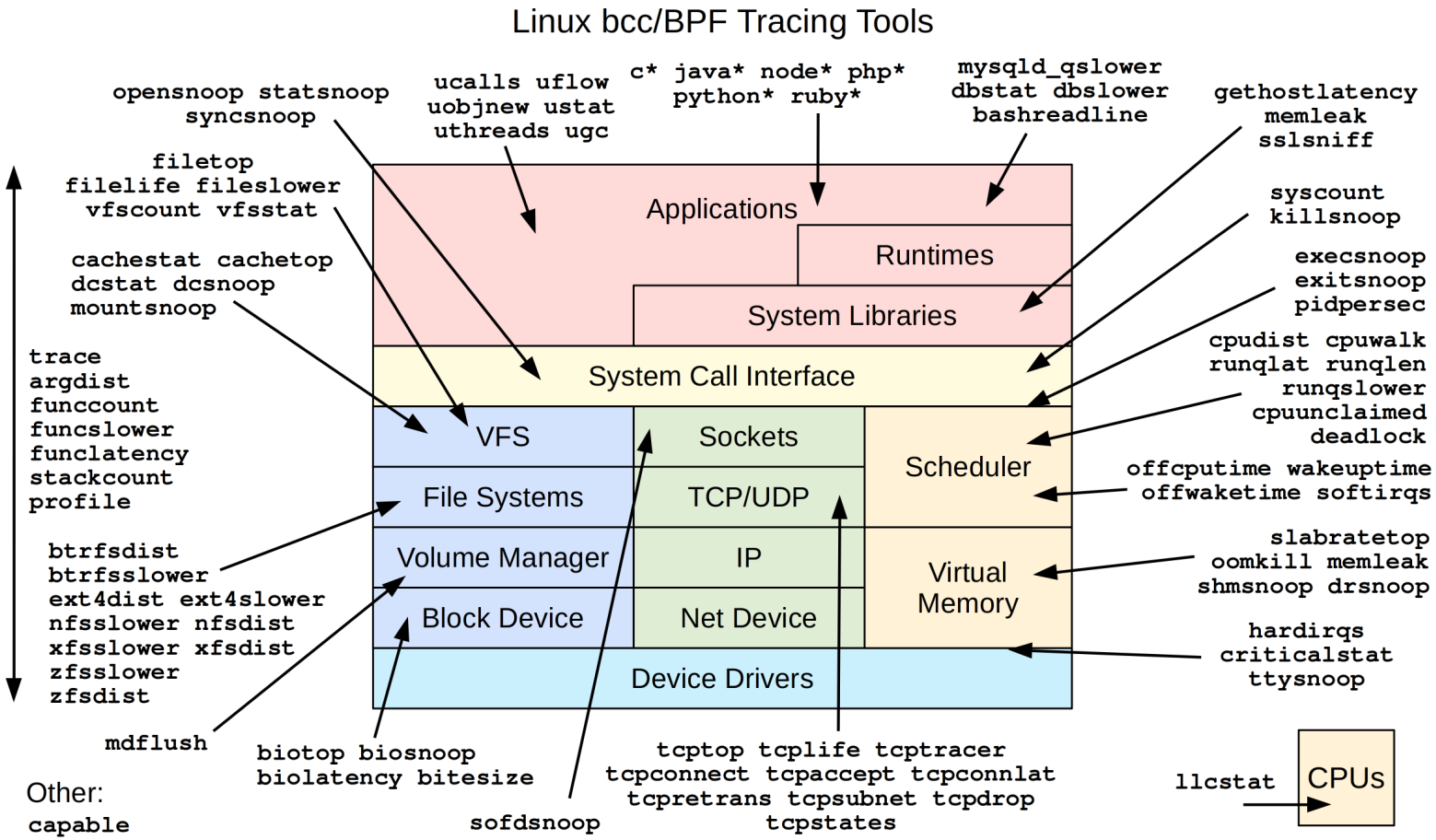
BCC

```
from __future__ import print_function
from bcc import BPF
from time import strftime
import ctypes as ct

# load BPF program
bpf_text = """
#include <uapi/linux/ptrace.h>
struct str_t {
    u64 pid;
    char str[80];
};
BPF_PERF_OUTPUT(events);
int printret(struct pt_regs *ctx) {
    struct str_t data  = {};
    u32 pid;
    if (!PT_REGS_RC(ctx))
        return 0;
    pid = bpf_get_current_pid_tgid();
    data.pid = pid;
    bpf_probe_read(&data.str, sizeof(data.str), (void *)PT_REGS_RC(ctx));
    events.perf_submit(ctx,&data,sizeof(data));
    return 0;
};
"""
STR_DATA = 80

class Data(ct.Structure):
    _fields_ = [
        ("pid", ct.c_ulonglong),
        ("str", ct.c_char * STR_DATA)
    ]

b = BPF(text=bpf_text)
b.attach_uretprobe(name="/bin/bash", sym="readline", fn_name="printret")

# header
print("%-9s %-6s %s" % ("TIME", "PID", "COMMAND"))

def print_event(cpu, data, size):
    event = ct.cast(data, ct.POINTER(Data)).contents
    print("%-9s %-6d %s" % (strftime("%H:%M:%S"), event.pid,
                            event.str.decode('utf-8', 'replace')))

b["events"].open_perf_buffer(print_event)
while 1:
b.perf_buffer_poll()
```

bpftrace

```
BEGIN
{
    printf("Tracing bash commands... Hit Ctrl-C to end.\n");
    printf("%-9s %-6s %s\n", "TIME", "PID", "COMMAND");
}

uretprobe:/bin/bash:readline
{
    time("%H:%M:%S  ");
    printf("%-6d %s\n", pid, str(retval));
}
```

BCC Tools Available

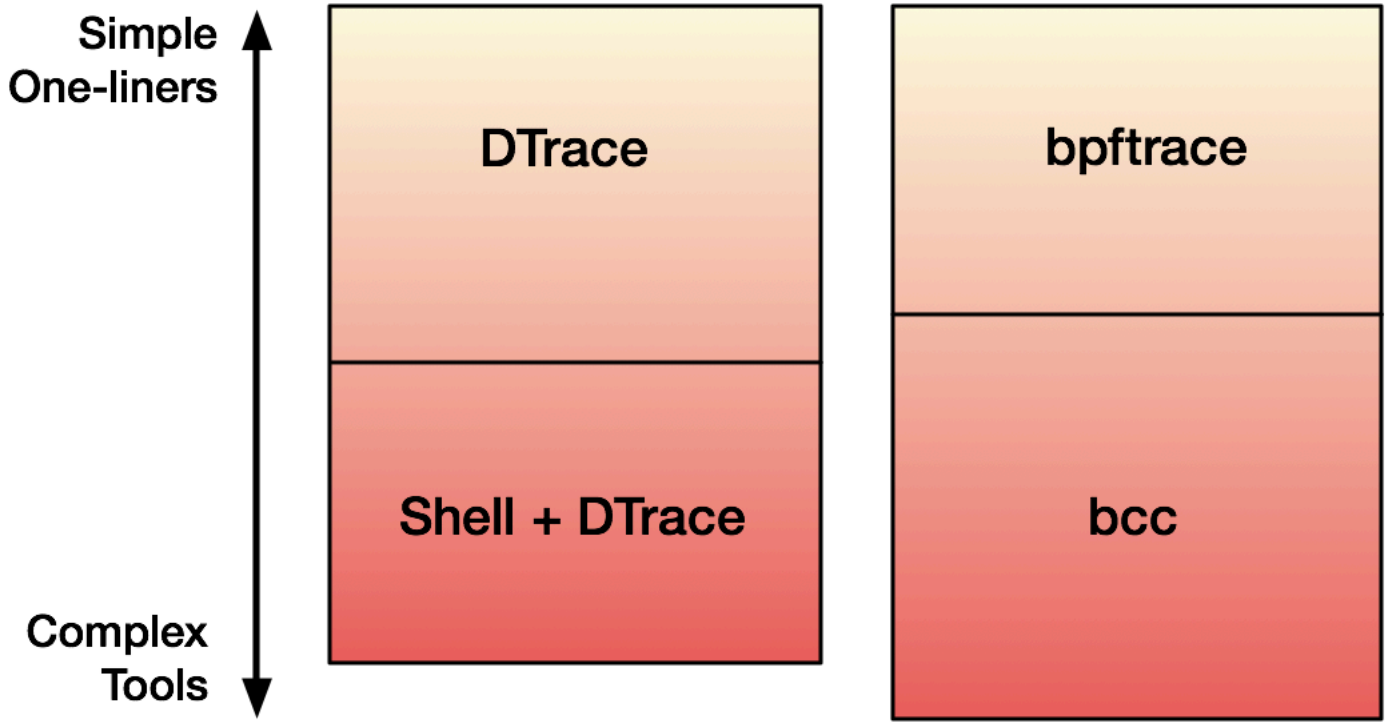


https://github.com/iovisor/bcc#tools 2019

PERCONA

DTrace vs bpfTrace

General Landscape Comparison



http://www.brendangregg.com/blog/2018-10-08/dtrace-for-linux-2018.html

PERCONA

bpfTrace and DTrace

There is no Direct compatibility

Similar in Spirit

bpfTrace is more powerful

Function Comparison Checklist

Type	DTrace	bpftrace
function	@ = quantize(value)	@ = hist(value)
function	@ = lquantize(value, min, max, step)	@ = lhist(value, min, max, step)
variable	this->name	$name
variable	self->name	@name[tid]
variable	name[key]	@name[key]
variable	global_name	@global_name
variable	self->name = 0	delete(@name[tid])
variable	curthread	curtask
variable	probeprov probemod probename	probe
provider	fbt::func:entry	kprobe:func
provider	fbt::func:return	kretprobe:func
provider	pid$target::func:entry	uprobe:func
provider	pid$target::func:return	uretprobe:func
provider	profile:::99	profile:hz:99
provider	profile:::tick-1sec	interval:s:1

Script Example

DTrace

```
#pragma D option quiet

/*
 * Print header
 */
dtrace:::BEGIN
{
    printf("Tracing... Hit Ctrl-C to end.\n"
}

self int last[dev_t];

/*
 * Process io start
 */
io:genunix::start
/self->last[args[0]->b_edev] != 0/
{
    /* calculate seek distance */
    this->last = self->last[args[0]->b_edev]
    this->dist = (int)(args[0]->b_blkno - th
        args[0]->b_blkno - this->last : this

    /* store details */
    @Size[pid, curpsinfo->pr_psargs] = quant
}

io:genunix::start
{
    /* save last position of disk head */
    self->last[args[0]->b_edev] = args[0]->b
        args[0]->b_bcount / 512;
}

/*
 * Print final report
 */
dtrace:::END
{
    printf("\n%8s  %s\n", "PID", "CMD");
    printa("%8d  %S\n%@d\n", @Size);
}
```

bpftrace

```
/*
 * Print header
 */
BEGIN
{
    printf("Tracing... Hit Ctrl-C to end.\n"
}

/*
 * Process io start
 */
tracepoint:block:block_rq_insert
/@last[args->dev]/
{
    // calculate seek distance
    $last = @last[args->dev];
    $dist = (args->sector - $last) > 0 ?
        args->sector - $last : $last - args-

    // store details
    @size[pid, comm] = hist($dist);
}

tracepoint:block:block_rq_insert
{
    // save last position of disk head
    @last[args->dev] = args->sector + args->
}

/*
 * Print final report
 */
END
{
    printf("\n@[PID, COMM]:\n");
    print(@size);
    clear(@size);
    clear(@last);
}
```


PERCONA

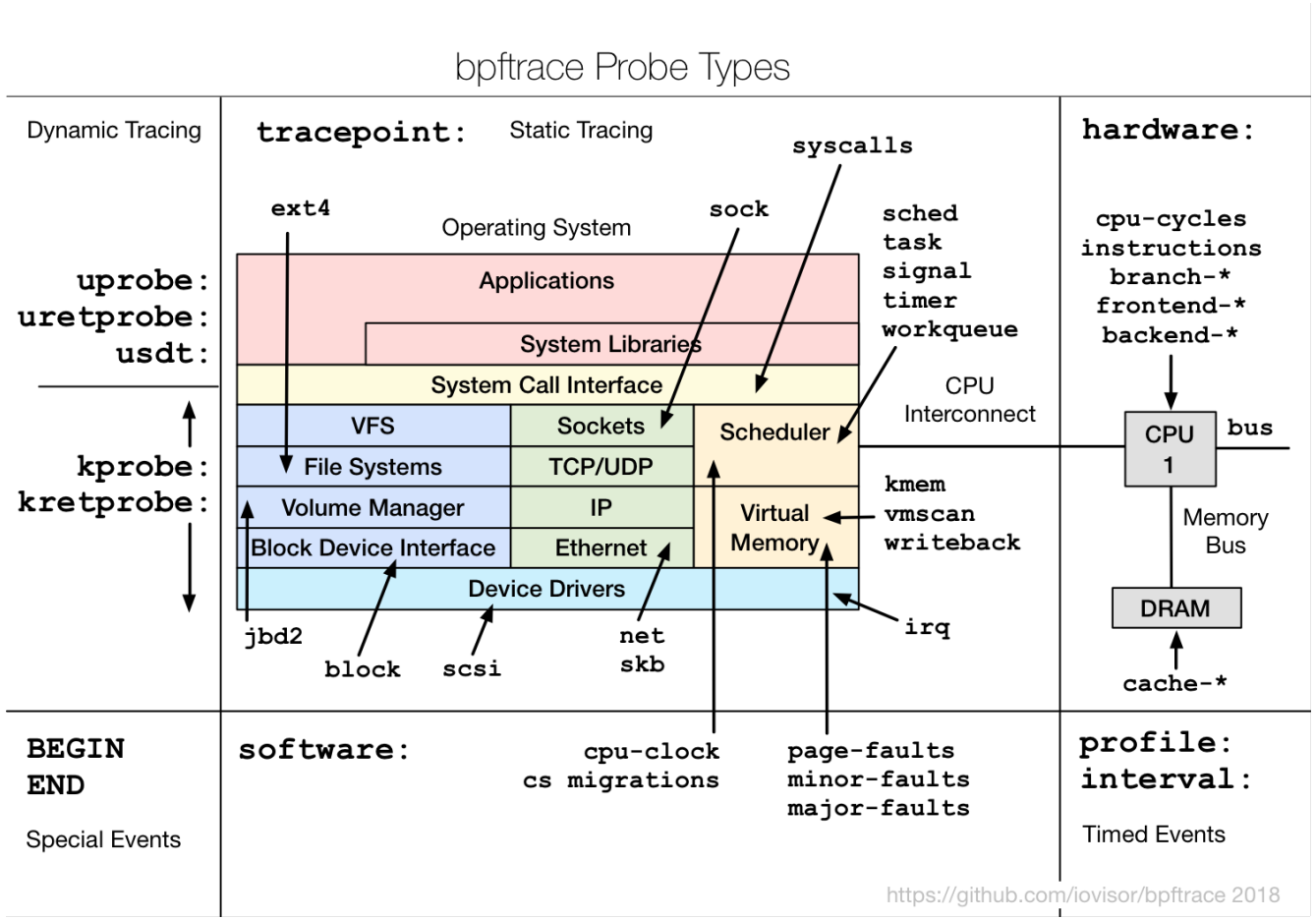
bpfTrace

Linux Requirements

Kprobes (4.1)	`kprobe:vfs_read { ... }`
Uprobes (4.3)	`uprobe:/bin/bash:readline { ... }`
USDT (4.3)	
Stack traces, per-cpu maps (4.6)	
Tracepoints (4.7)	`tracepoint:sched:sched_switch { ... }`
Timers (4.9)	`profile:hz:99 { ... }`
Software events (4.9)	`software:faults: { ... }`
Hardware events (4.9)	`hardware:cache-references: { ... }`

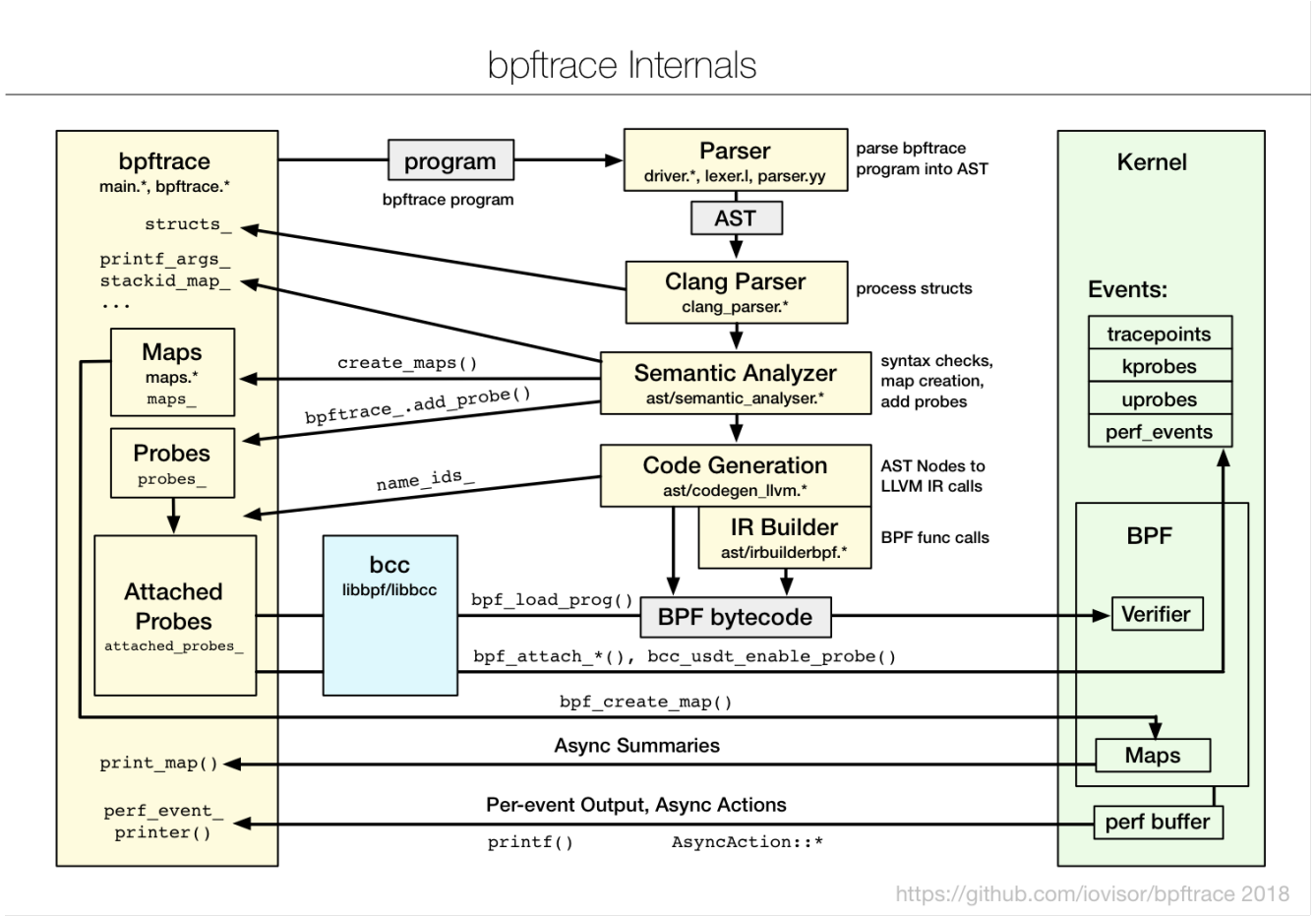
https://tracingsummit.org/w/images/8/82/Tracingsummit2018-bpftrace-robertson.pdf

bpfTrace Probe Types



PERCONA

How BPFTrace Works



PERCONA

Support In Linux Distributions

Not all Distributions have packages

Development is Quick Paced – Many have outdated packages

If you use eBPF consider getting new packages

Install bpftrace

Ubuntu snap package

Warning: snap packages have limited functionality

On Ubuntu 16.04 and later, bpftrace is available as a snap package (https://snapcraft.io/bpftrace) and can be installed with snap. The current snap provides extremely limited file permissions so the --devmode option should be specified on installation in order avoid file access issues.

```
sudo snap install --devmode bpftrace
sudo snap connect bpftrace:system-trace
```

More Details: https://github.com/iovisor/bpftrace/blob/master/INSTALL.md

Timing Reads by Process

```
bpftrace -e 'kprobe:vfs_read { @start[tid] = nsecs; }
kretprobe:vfs_read /@start[tid]/ { @ns[comm] = hist(nsecs -
@start[tid]); delete(@start[tid]); }'
```

```
@ns[mysqld]:
[512, 1K)              8 |                                                    |
[1K, 2K)            1755 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@                        |
[2K, 4K)            3237 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[4K, 8K)             504 |@@@@@@@@                                            |
[8K, 16K)            142 |@@                                                  |
[16K, 32K)            42 |                                                    |
[32K, 64K)            12 |                                                    |
[64K, 128K)           17 |                                                    |
[128K, 256K)          22 |                                                    |
[256K, 512K)         505 |@@@@@@@@                                            |
[512K, 1M)           315 |@@@@@                                               |
[1M, 2M)              61 |                                                    |
[2M, 4M)              41 |                                                    |
[4M, 8M)              37 |                                                    |
[8M, 16M)             11 |                                                    |
[16M, 32M)             2 |                                                    |
[32M, 64M)             1 |                                                    |
```

Saving it as Script File

```
// read.bt file
tracepoint:syscalls:sys_enter_read
{
  @start[tid] = nsecs;
}

tracepoint:syscalls:sys_exit_read / @start[tid] /
{
  @times = hist(nsecs - @start[tid]);
  delete(@start[tid]);
}
```

```
# bpftrace read.bt
Attaching 2 probes...
^C
```

Concept Overview

General Syntax	• probe[,probe,...] /filter/ { action }
Filter	• Filtering output of the Probe (ie by Pid)
Action	• Mini-Program to be ran

https://github.com/iovisor/bpftrace/blob/master/docs/reference_guide.md

BpfTrace Tools

Tools

bpftrace contains various tools, which also serve as examples of programming in the bpftrace language.

- tools/bashreadline.bt: Print entered bash commands system wide. Examples.
- tools/biolatency.bt: Block I/O latency as a histogram. Examples.
- tools/biosnoop.bt: Block I/O tracing tool, showing per I/O latency. Examples.
- tools/biostacks.bt: Show disk I/O latency with initialization stacks. Examples.
- tools/bitesize.bt: Show disk I/O size as a histogram. Examples.
- tools/capable.bt: Trace security capability checks. Examples.
- tools/cpuwalk.bt: Sample which CPUs are executing processes. Examples.
- tools/dcsnoop.bt: Trace directory entry cache (dcache) lookups. Examples.
- tools/execsnoop.bt: Trace new processes via exec() syscalls. Examples.
- tools/gethostlatency.bt: Show latency for getaddrinfo/gethostbyname[2] calls. Examples.
- tools/killsnoop.bt: Trace signals issued by the kill() syscall. Examples.

https://github.com/iovisor/bpftrace

Curios to see BPF Code ?

```
# bpftrace -v -e 'tracepoint:syscalls:sys_enter_nanosleep { printf("%s is sleeping.\n", comm); }'
Attaching 1 probe...

Bytecode:
0: (bf) r6 = r1
1: (b7) r1 = 0
2: (7b) *(u64 *)(r10 -24) = r1
3: (7b) *(u64 *)(r10 -32) = r1
4: (7b) *(u64 *)(r10 -40) = r1
5: (7b) *(u64 *)(r10 -8) = r1
6: (7b) *(u64 *)(r10 -16) = r1
7: (bf) r1 = r10
8: (07) r1 += -16
9: (b7) r2 = 16
10: (85) call bpf_get_current_comm#16
11: (79) r1 = *(u64 *)(r10 -16)
12: (7b) *(u64 *)(r10 -32) = r1
```

Tracing MySQL

```
bpftrace -e 'uprobe:/usr/sbin/mysqld:dispatch_command {
printf("%s\n", str(arg2)); }'
failed to stat uprobe target file /usr/sbin/mysqld: No such
file or directory

root@mysql1:/# ls -la /usr/sbin/mysqld
-rwxr-xr-x 1 root root 60718384 Oct 25 09:19 /usr/sbin/mysqld
```

Tracing MySQL

Using apt installed bpftrace rather than snap package

```
root@mysql1:~# bpftrace -e
'uprobe:/usr/sbin/mysqld:dispatch_command { printf("%s\n",
str(arg2)); }'
Attaching 1 probe...
Could not resolve symbol: /usr/sbin/mysqld:dispatch_command
```

Tracing MySQL(MariaDB)

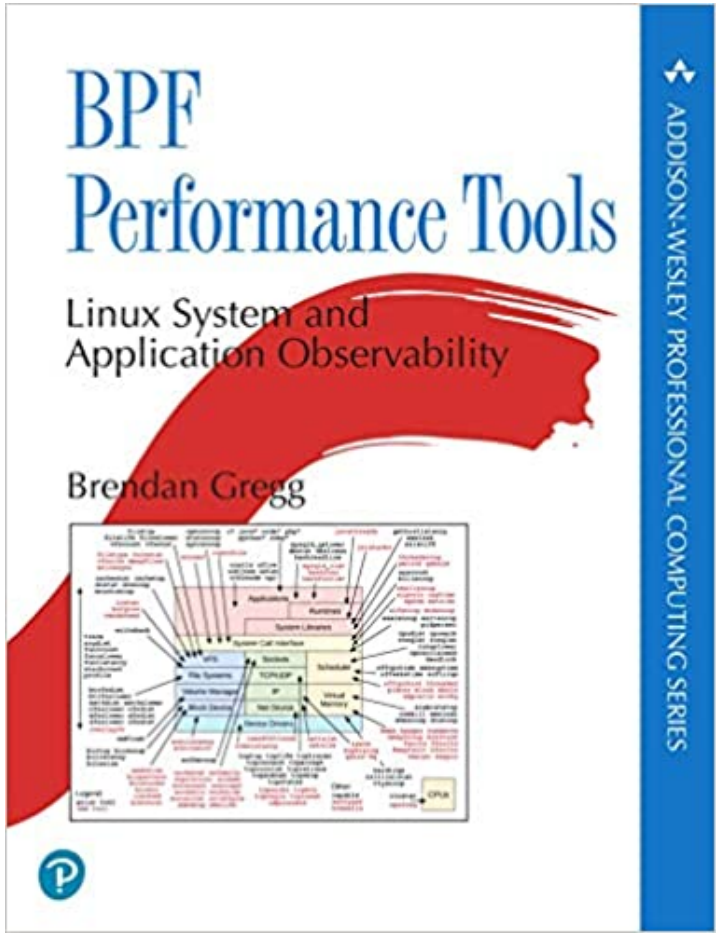
```
root@mysql1:~# nm -D /usr/sbin/mysqld | grep dispatch_command
0000000005af770 T
_Z16dispatch_command19enum_server_commandP3THDPcjbb
root@localhost:~# bpftrace -e
'uprobe:/usr/sbin/mysqld:_Z16dispatch_command19enum_server_co
mmandP3THDPcjbb { printf("%s\n", str(arg2)); }'
Attaching 1 probe...
select @@version_comment limit 1
select 1
```


PERCONA

Check out eBPF Bible

http://www.brendangregg.com/ebpf.html

Books



PERCONA

Further Reading List

https://github.com/zoidbergwill/awesome-ebpf

https://slideplayer.com/slide/12710510/

http://www.brendangregg.com/ebpf.html

http://vger.kernel.org/netconf2018_files/BrendanGregg_netconf2018.pdf

http://www.brendangregg.com/Slides/Velocity2017_BPF_superpowers.pdf

https://lwn.net/Articles/740157/

https://tracingsummit.org/w/images/8/82/Tracingsummit2018-bpftrace-robertson.pdf


SAVE THE DATE
ONLINE, AROUND-THE-WORLD OPEN SOURCE DATABASE CONFERENCE
PERCONA
LIVEONLINE
MAY 12 - 13th
2021
CALL FOR PAPERS
NOW OPEN!
http://www.perconalive.com

Thank You!