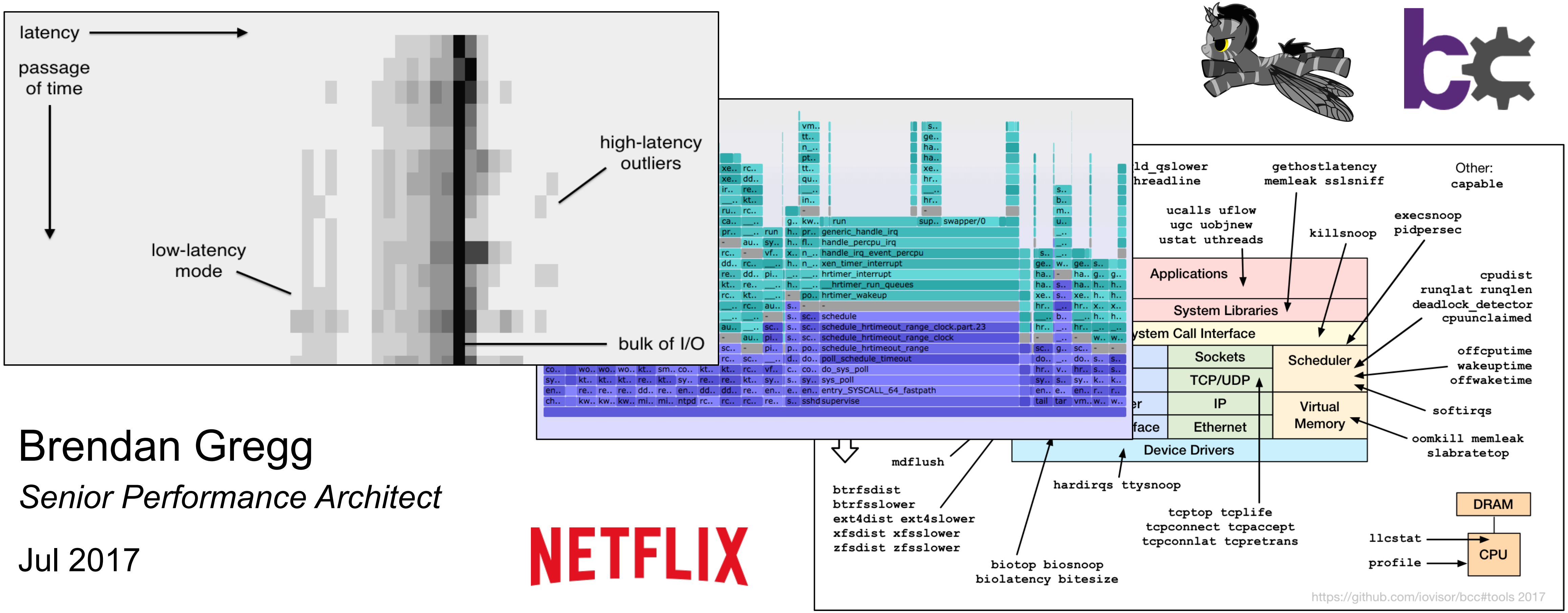
2017 USENIX Annual Technical Conference

Performance Analysis Superpowers with Linux eBPF

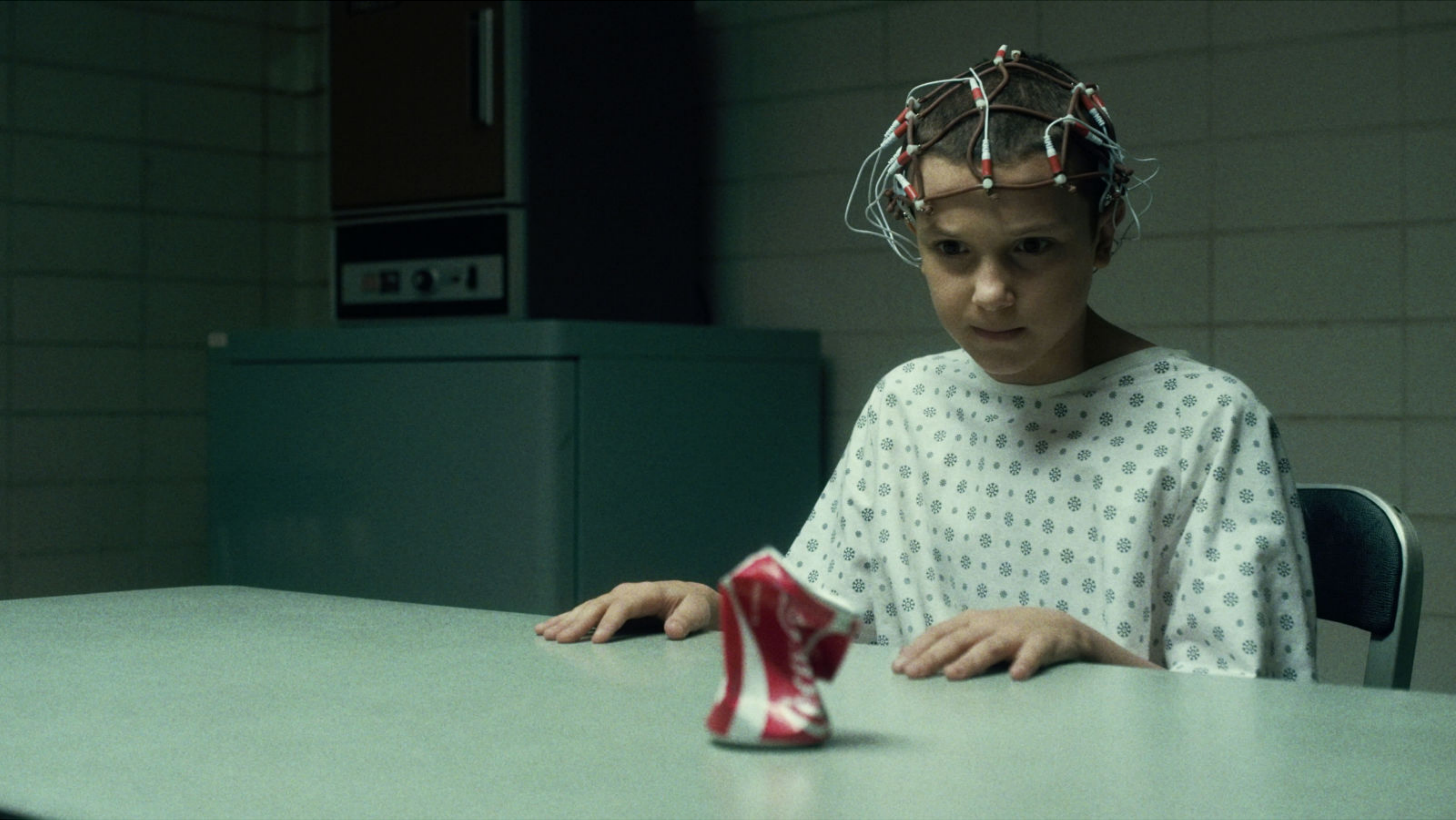


Brendan Gregg

Senior Performance Architect

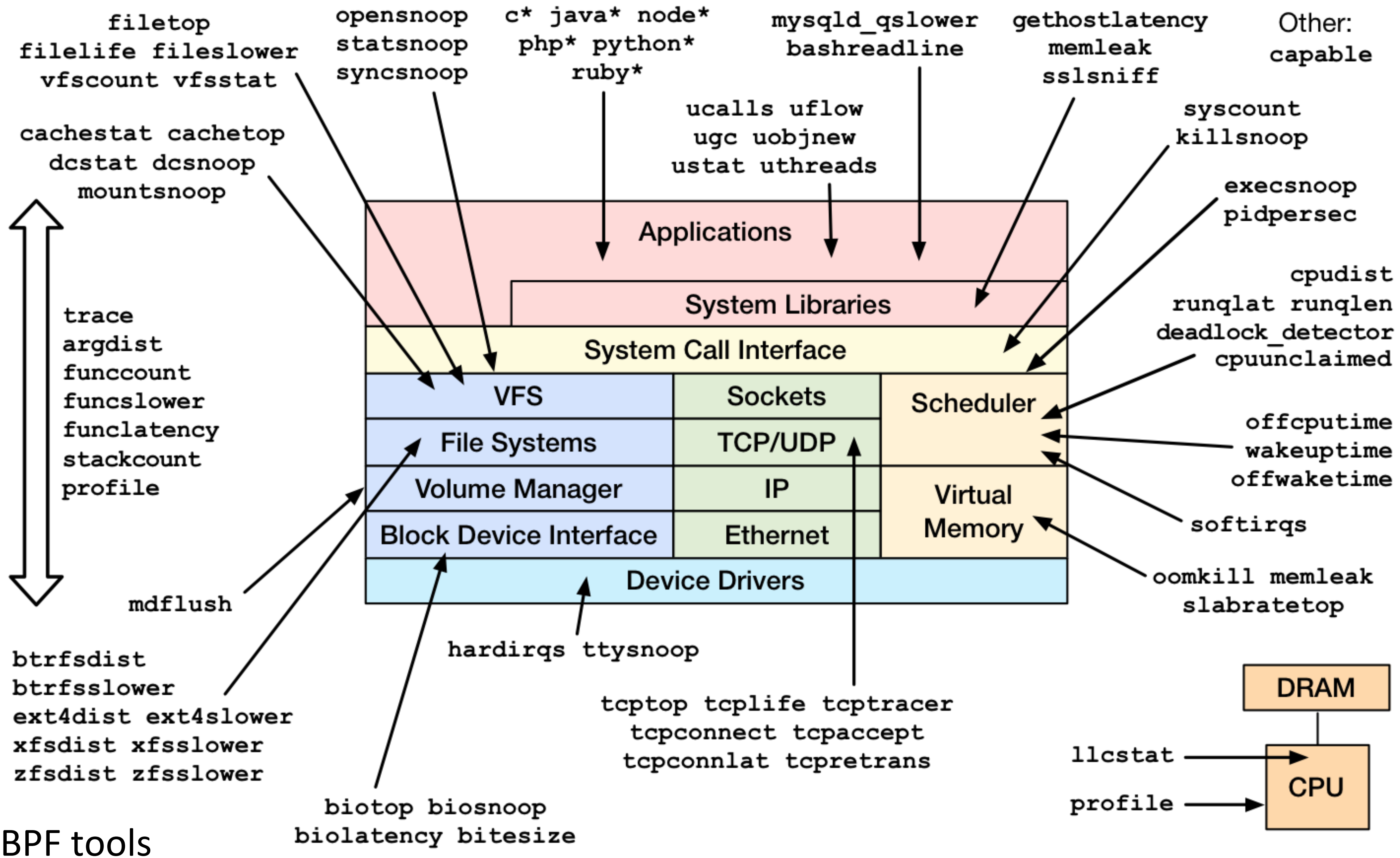
Jul 2017

NETFLIX



Efficiently trace TCP sessions with PID, bytes, and duration using `tcplife`

```
# /usr/share/bcc/tools/tcplife
PID   COMM       LADDR           LPORT RADDR           RPORT TX_KB RX_KB MS
2509  java       100.82.34.63    8078  100.82.130.159  12410     0     0 5.44
2509  java       100.82.34.63    8078  100.82.78.215   55564     0     0 135.32
2509  java       100.82.34.63    60778 100.82.207.252  7001      0    13 15126.87
2509  java       100.82.34.63    38884 100.82.208.178  7001      0     0 15568.25
2509  java       127.0.0.1       4243  127.0.0.1       42166     0     0 0.61
2509  java       127.0.0.1       42166 127.0.0.1       4243      0     0 0.67
12030 upload-mes 127.0.0.1       34020 127.0.0.1       8078     11     0 3.38
2509  java       127.0.0.1       8078  127.0.0.1       34020     0    11 3.41
12030 upload-mes 127.0.0.1       21196 127.0.0.1       7101      0     0 12.61
3964  mesos-slav 127.0.0.1       7101  127.0.0.1       21196     0     0 12.64
12021 upload-sys 127.0.0.1       34022 127.0.0.1       8078    372     0 15.28
2509  java       127.0.0.1       8078  127.0.0.1       34022     0   372 15.31
2235  dockerd    100.82.34.63    13730 100.82.136.233  7002      0     4 18.50
2235  dockerd    100.82.34.63    34314 100.82.64.53    7002      0     8 56.73
12068 titus-reap 127.0.0.1       46476 127.0.0.1       19609     0     0 1.25
[...]
```



bcc/BPF tools

https://github.com/iovisor/bcc#tools 2017

Agenda

1. eBPF & bcc

```
# /usr/share/bcc/tools/runqlat 10
Tracing run queue latency... Hit Ctrl-C to end.

     usecs               : count     distribution
         0 -> 1          : 2810     |*                                       |
         2 -> 3          : 5248     |**                                      |
         4 -> 7          : 12369    |******                                  |
         8 -> 15         : 71312    |****************************************|
        16 -> 31         : 55705    |*******************************         |
        32 -> 63         : 11775    |******                                  |
        64 -> 127        : 6230     |***                                     |
       128 -> 255        : 2758     |*                                       |
       256 -> 511        : 549      |                                        |
       512 -> 1023       : 46       |                                        |
      1024 -> 2047       : 11       |                                        |
      2048 -> 4095       : 4        |                                        |
      4096 -> 8191       : 5        |                                        |
[…]
```

2. bcc/BPF CLI Tools

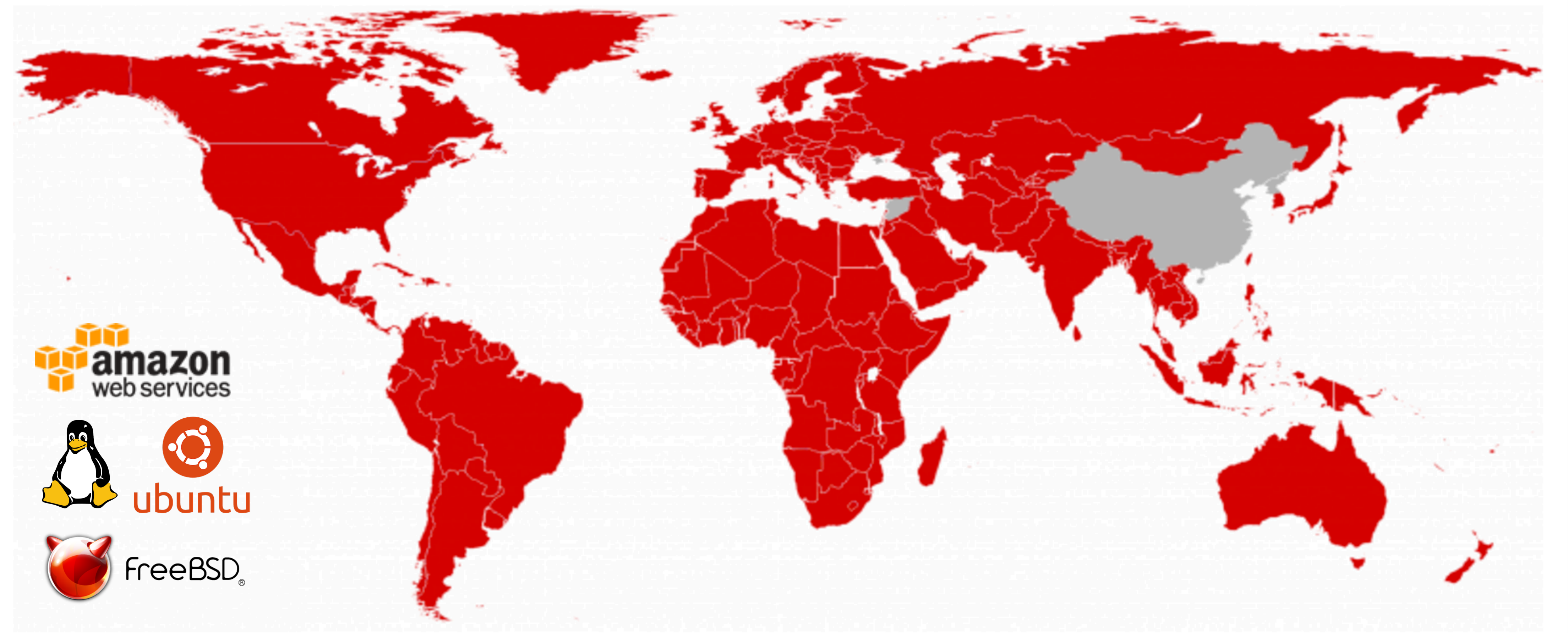
3. bcc/BPF Visualizations

Take aways

1. Understand Linux tracing components
2. Understand the role and state of enhanced BPF
3. Discover opportunities for future development

NETFLIX

REGIONS WHERE NETFLIX IS AVAILABLE



Who at Netflix will use BPF?

Introducing enhanced BPF for tracing: kernel-level software

BPF

Ye Olde BPF

Berkeley Packet Filter

```
# tcpdump host 127.0.0.1 and port 22 -d
(000) ldh      [12]
(001) jeq      #0x800           jt 2    jf 18
(002) ld       [26]
(003) jeq      #0x7f000001      jt 6    jf 4
(004) ld       [30]
(005) jeq      #0x7f000001      jt 6    jf 18
(006) ldb      [23]
(007) jeq      #0x84            jt 10   jf 8
(008) jeq      #0x6             jt 10   jf 9
(009) jeq      #0x11            jt 10   jf 18
(010) ldh      [20]
(011) jset     #0x1fff          jt 18   jf 12
(012) ldxb     4*([14]&0xf)
(013) ldh      [x + 14]
[...]
```

Optimizes packet filter performance

2 x 32-bit registers & scratch memory

User-defined bytecode executed by an in-kernel sandboxed virtual machine

Steven McCanne and Van Jacobson, 1993

Enhanced BPF

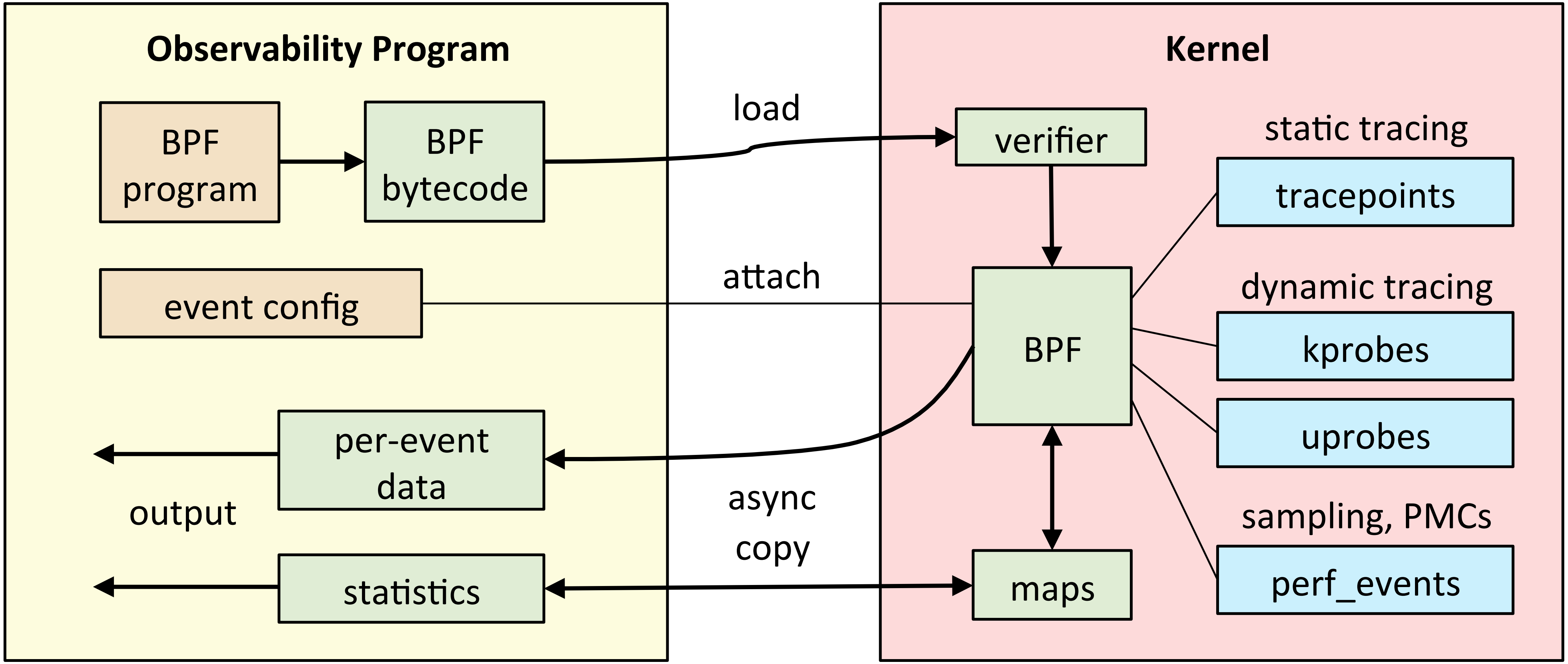
aka eBPF or just "BPF"

```
struct bpf_insn prog[] = {
        BPF_MOV64_REG(BPF_REG_6, BPF_REG_1),
        BPF_LD_ABS(BPF_B, ETH_HLEN + offsetof(struct iphdr, protocol) /* R0 = ip->proto */),
        BPF_STX_MEM(BPF_W, BPF_REG_10, BPF_REG_0, -4), /* *(u32 *)(fp - 4) = r0 */
        BPF_MOV64_REG(BPF_REG_2, BPF_REG_10),
        BPF_ALU64_IMM(BPF_ADD, BPF_REG_2, -4), /* r2 = fp - 4 */
        BPF_LD_MAP_FD(BPF_REG_1, map_fd),
        BPF_RAW_INSN(BPF_JMP | BPF_CALL, 0, 0, 0, BPF_FUNC_map_lookup_elem),
        BPF_JMP_IMM(BPF_JEQ, BPF_REG_0, 0, 2),
        BPF_MOV64_IMM(BPF_REG_1, 1), /* r1 = 1 */
        BPF_RAW_INSN(BPF_STX | BPF_XADD | BPF_DW, BPF_REG_0, BPF_REG_1, 0, 0), /* xadd r0 += r1 */
        BPF_MOV64_IMM(BPF_REG_0, 0), /* r0 = 0 */
        BPF_EXIT_INSN(),
};
```

10 x 64-bit registers
maps (hashes)
actions

Alexei Starovoitov, 2014+

BPF for Tracing, Internals



Enhanced BPF is also now used for SDNs, DDOS mitigation, intrusion detection, container security, …

Dynamic Tracing

To Appear in Proceedings of the
1994 Scalable High Performance Computing Conference, May 1994 (Knoxville, TN).

Dynamic Program Instrumentation for Scalable Performance Tools

Jeffrey K. Hollingsworth
hollings@cs.wisc.edu

Barton P. Miller
bart@cs.wisc.edu

Jon Cargille
jon@cs.wisc.edu

Computer Sciences Department
University of Wisconsin-Madison

Abstract

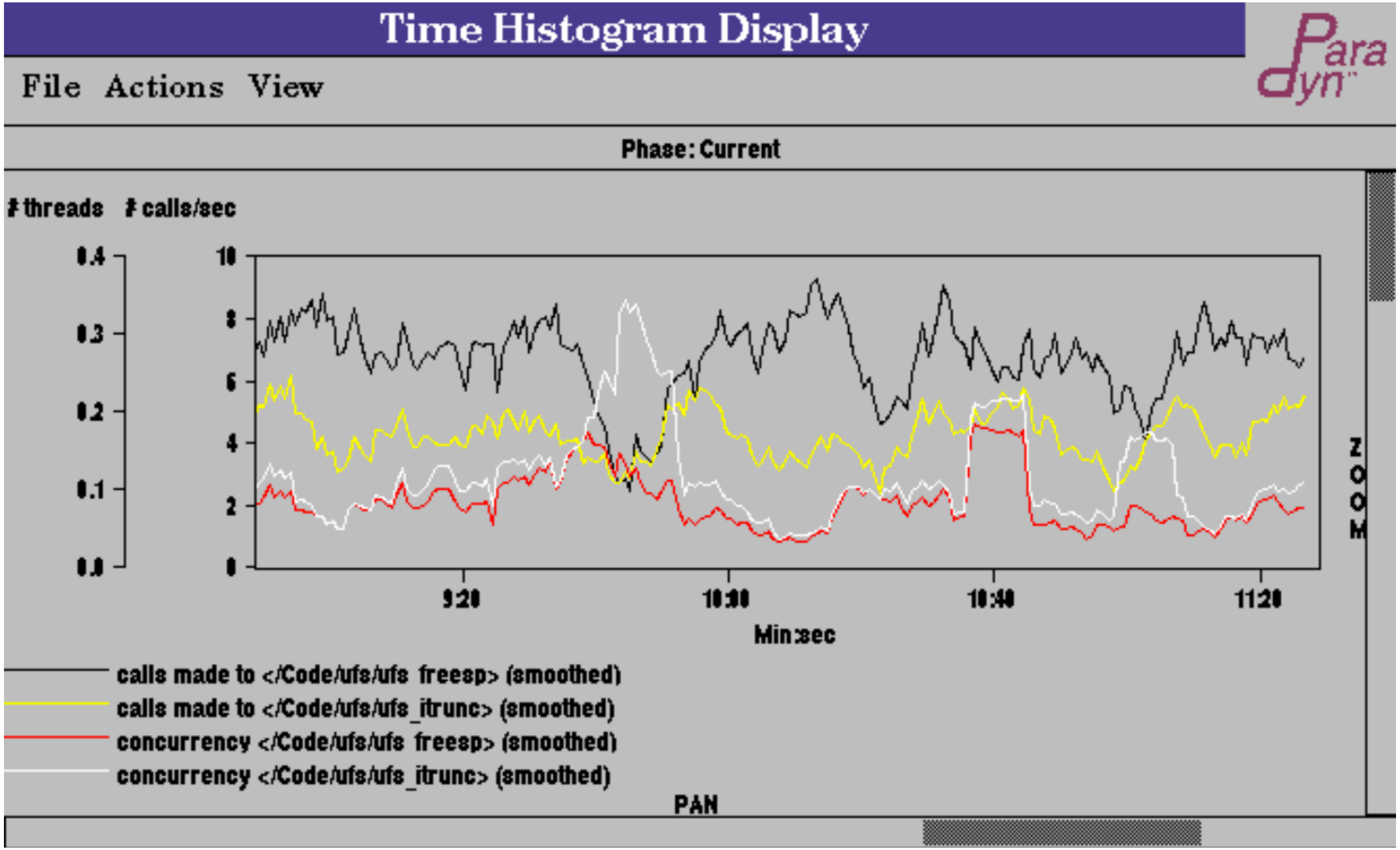
In this paper, we present a new technique called dynamic instrumentation that provides efficient, scalable, yet detailed data collection for large-scale parallel applications. Our approach is unique because it defers inserting any instrumentation until the application is in execution. We can insert or change instrumentation at any time during execution by modifying the application's binary image. Only the instrumentation required for the currently selected analysis or visualization is inserted. As a result, our technique collects several orders of magnitude less data than traditional data collection approaches. We have implemented a prototype of our dynamic instrumentation on the CM-5, and present results

understand the bottlenecks in their program. It must be frugal so that the instrumentation overhead does not obscure or distort the bottlenecks in the original program. The instrumentation system must also scale to large, production data set sizes and number of processors.

A detailed instrumentation system needs to be able to collect data about each component of a parallel machine. To correct bottlenecks, programmers need to know as precisely as possible how the utilization of these components is hindering the performance of their program.

There are two ways to provide frugal instrumentation: make data collection efficient, or collect less data. All tool builders strive to make their data collection more

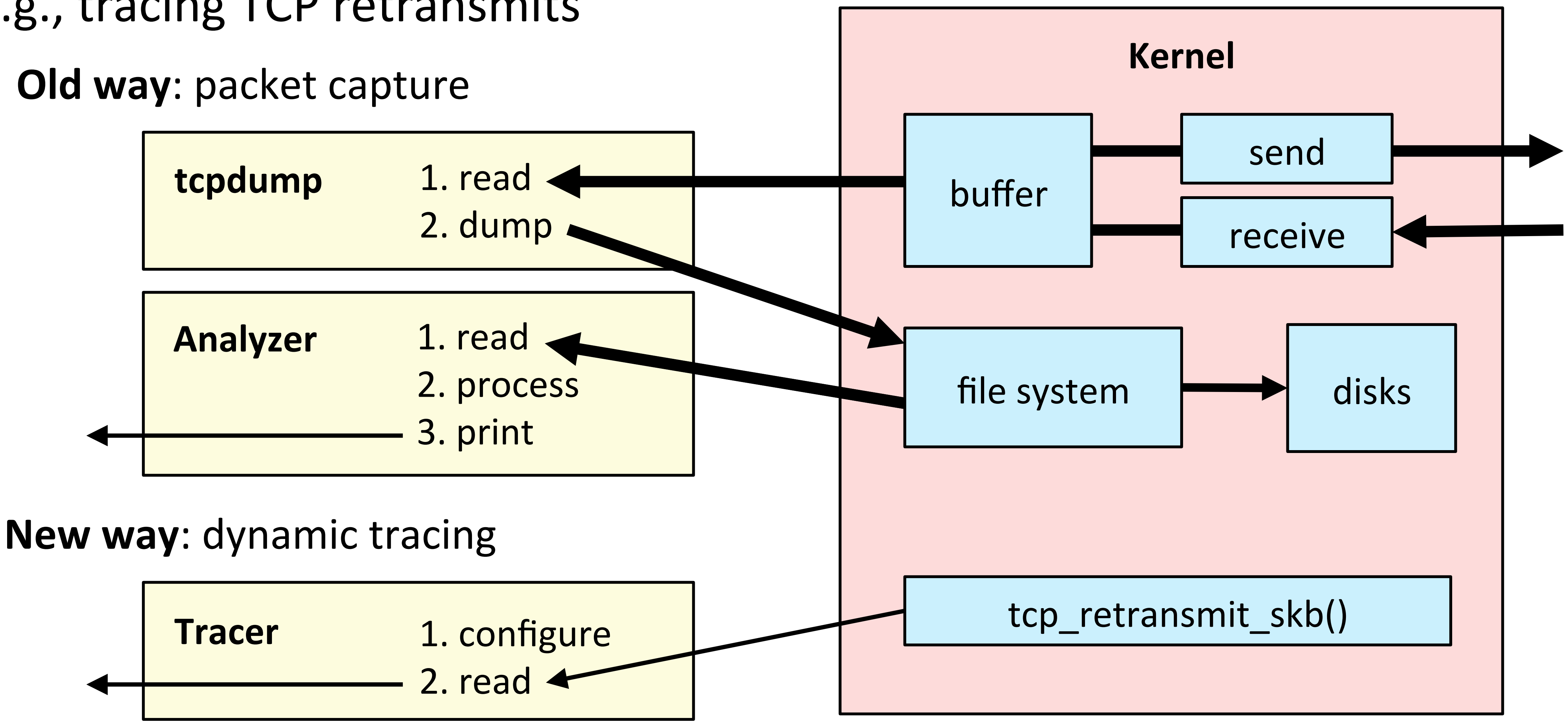
1999: Kerninst



http://www.paradyn.org/html/kerninst.html

Event Tracing Efficiency

E.g., tracing TCP retransmits

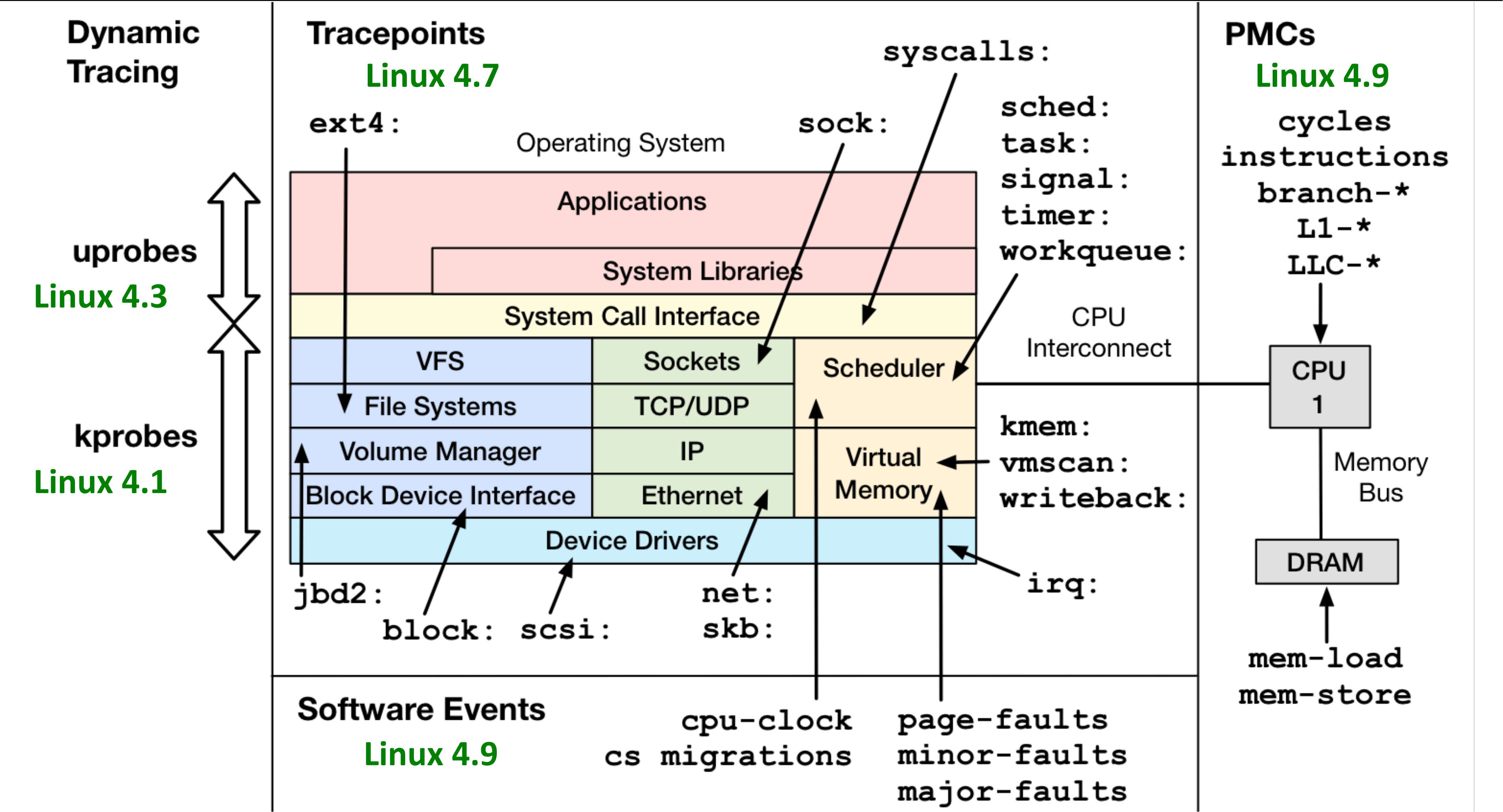


Linux Events & BPF Support

BPF output
Linux 4.4

BPF stacks
Linux 4.6

(version BPF support arrived)



A Linux Tracing Timeline

- 1990's: Static tracers, prototype dynamic tracers
- 2000: LTT + DProbes (dynamic tracing; not integrated)
- 2004: kprobes (2.6.9)
- 2005: DTrace (not Linux), SystemTap (out-of-tree)
- 2008: ftrace (2.6.27)
- 2009: perf_events (2.6.31)
- 2009: tracepoints (2.6.32)
- 2010-2016: ftrace & perf_events enhancements
- 2012: uprobes (3.5)
- **2014-2017: enhanced BPF patches: supporting tracing events**
- 2016-2017: ftrace hist triggers

also: LTTng, ktap, sysdig, ...

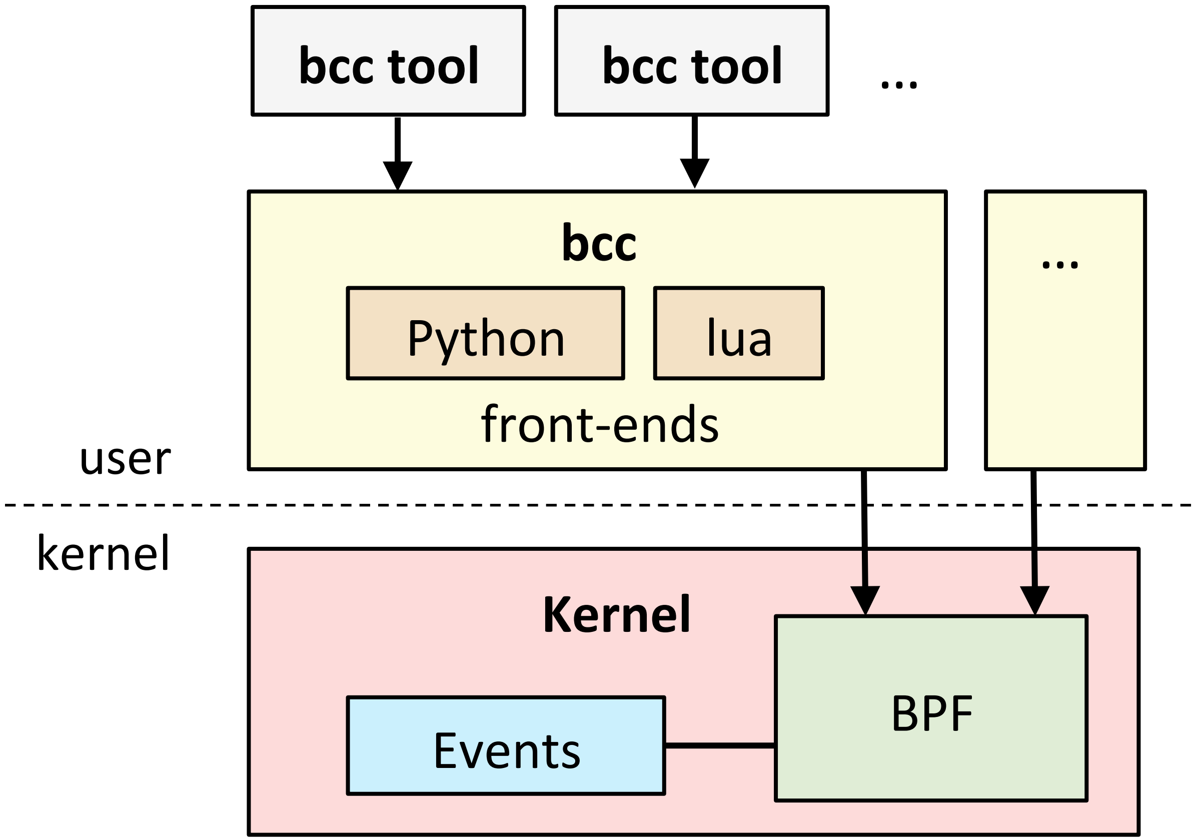
Introducing BPF Complier Collection: user-level front-end

BCC

bcc

- BPF Compiler Collection
  - https://github.com/iovisor/bcc
  - Lead developer: Brenden Blanco
- Includes tracing tools
- Provides BPF front-ends:
  - Python
  - Lua
  - C++
  - C helper libraries
  - golang (gobpf)

Tracing layers:



Raw BPF

```
struct bpf_insn prog[] = {
        BPF_MOV64_REG(BPF_REG_6, BPF_REG_1),
        BPF_LD_ABS(BPF_B, ETH_HLEN + offsetof(struct iphdr, protocol) /* R0 = ip->proto */),
        BPF_STX_MEM(BPF_W, BPF_REG_10, BPF_REG_0, -4), /* *(u32 *)(fp - 4) = r0 */
        BPF_MOV64_REG(BPF_REG_2, BPF_REG_10),
        BPF_ALU64_IMM(BPF_ADD, BPF_REG_2, -4), /* r2 = fp - 4 */
        BPF_LD_MAP_FD(BPF_REG_1, map_fd),
        BPF_RAW_INSN(BPF_JMP | BPF_CALL, 0, 0, 0, BPF_FUNC_map_lookup_elem),
        BPF_JMP_IMM(BPF_JEQ, BPF_REG_0, 0, 2),
        BPF_MOV64_IMM(BPF_REG_1, 1), /* r1 = 1 */
        BPF_RAW_INSN(BPF_STX | BPF_XADD | BPF_DW, BPF_REG_0, BPF_REG_1, 0, 0), /* xadd r0 += r1 */
        BPF_MOV64_IMM(BPF_REG_0, 0), /* r0 = 0 */
        BPF_EXIT_INSN(),
};
```

samples/bpf/sock_example.c
87 lines truncated

C/BPF

```
SEC("kprobe/__netif_receive_skb_core")
int bpf_prog1(struct pt_regs *ctx)
{
        /* attaches to kprobe netif_receive_skb,
         * looks for packets on loobpack device and prints them
         */
        char devname[IFNAMSIZ];
        struct net_device *dev;
        struct sk_buff *skb;
        int len;

        /* non-portable! works for the given kernel only */
        skb = (struct sk_buff *) PT_REGS_PARM1(ctx);
        dev = _(skb->dev);
```

samples/bpf/tracex1_kern.c

58 lines truncated

bcc/BPF (C & Python)

```
# load BPF program
b = BPF(text="""
#include <uapi/linux/ptrace.h>
#include <linux/blkdev.h>
BPF_HISTOGRAM(dist);
int kprobe__blk_account_io_completion(struct pt_regs *ctx,
    struct request *req)
{
    dist.increment(bpf_log2l(req->__data_len / 1024));
    return 0;
}
""")
```

```
# header
print("Tracing... Hit Ctrl-C to end.")

# trace until Ctrl-C
try:
    sleep(99999999)
except KeyboardInterrupt:
    print

# output
b["dist"].print_log2_hist("kbytes")
```

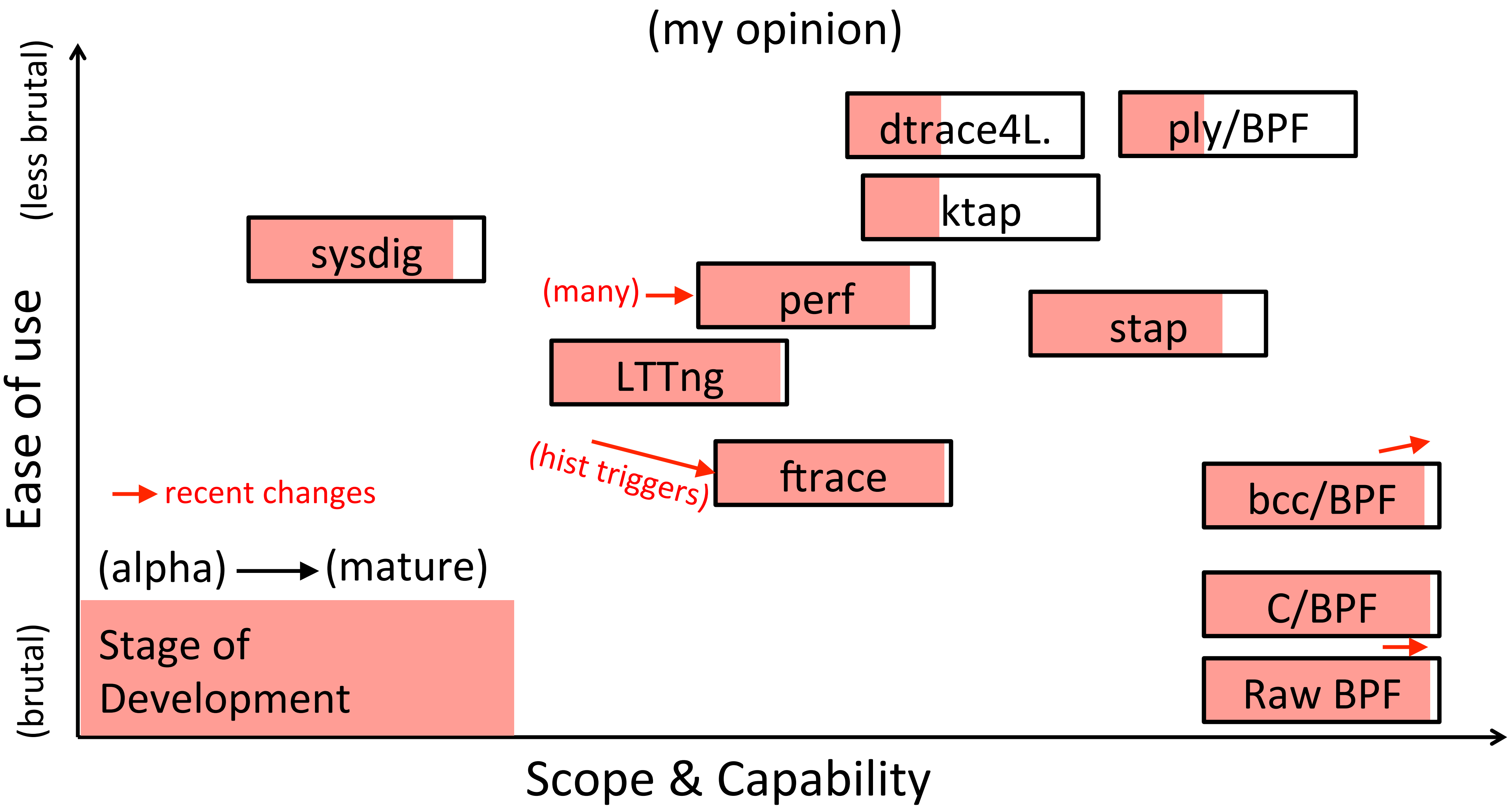
bcc examples/tracing/bitehist.py
entire program

ply/BPF

```
kretprobe:SyS_read
{
    @.quantize(retval());
}
```

https://github.com/iovisor/ply/blob/master/README.md
entire program

The Tracing Landscape, Jul 2017



Performance analysis

BCC/BPF CLI TOOLS

Pre-BPF: Linux Perf Analysis in 60s

1. `uptime`
2. `dmesg -T | tail`
3. `vmstat 1`
4. `mpstat -P ALL 1`
5. `pidstat 1`
6. `iostat -xz 1`
7. `free -m`
8. `sar -n DEV 1`
9. `sar -n TCP,ETCP 1`
10. `top`

http://techblog.netflix.com/2015/11/linux-performance-analysis-in-60s.html

bcc Installation

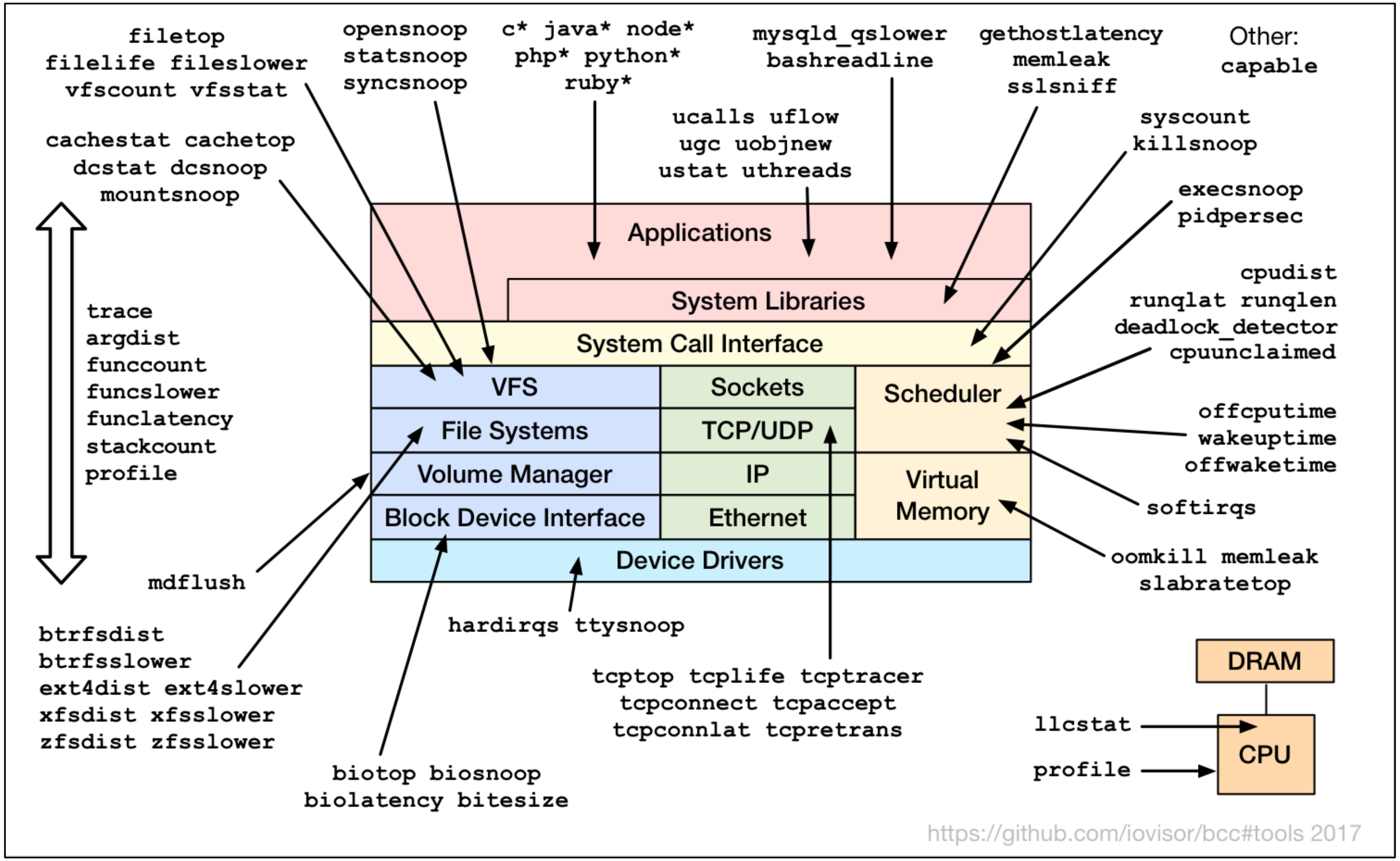
- https://github.com/iovisor/bcc/blob/master/INSTALL.md
- eg, Ubuntu Xenial:

```
# echo "deb [trusted=yes] https://repo.iovisor.org/apt/xenial xenial-nightly main" |\
    sudo tee /etc/apt/sources.list.d/iovisor.list
# sudo apt-get update
# sudo apt-get install bcc-tools
```

  - Also available as an Ubuntu snap
  - Ubuntu 16.04 is good, 16.10 better: more tools work
- Installs many tools
  - In /usr/share/bcc/tools, and …/tools/old for older kernels

bcc General Performance Checklist

1. execsnoop
2. opensnoop
3. ext4slower (…)
4. biolatency
5. biosnoop
6. cachestat
7. tcpconnect
8. tcpaccept
9. tcpretrans
10. gethostlatency
11. runqlat
12. profile



Discover short-lived process issues using execsnoop

```
# execsnoop -t
TIME(s) PCOMM            PID    PPID   RET ARGS
0.031   dirname          23832  23808    0 /usr/bin/dirname /apps/tomcat/bin/catalina.sh
0.888   run              23833  2344     0 ./run
0.889   run              23833  2344    -2 /command/bash
0.889   run              23833  2344    -2 /usr/local/bin/bash
0.889   run              23833  2344    -2 /usr/local/sbin/bash
0.889   bash             23833  2344     0 /bin/bash
0.894   svstat           23835  23834    0 /command/svstat /service/nflx-httpd
0.894   perl             23836  23834    0 /usr/bin/perl -e $l=<>;$l=~/(\d+) sec/;print $1||0;
0.899   ps               23838  23837    0 /bin/ps --ppid 1 -o pid,cmd,args
0.900   grep             23839  23837    0 /bin/grep org.apache.catalina
0.900   sed              23840  23837    0 /bin/sed s/^ *//;
0.900   cut              23841  23837    0 /usr/bin/cut -d  -f 1
0.901   xargs            23842  23837    0 /usr/bin/xargs
0.912   xargs            23843  23842   -2 /command/echo
0.912   xargs            23843  23842   -2 /usr/local/bin/echo
0.912   xargs            23843  23842   -2 /usr/local/sbin/echo
0.912   echo             23843  23842    0 /bin/echo
[...]
```

Efficient: only traces exec()

Discover short-lived process issues using **execsnoop**

```
# execsnoop -t
TIME(s) PCOMM            PID    PPID   RET ARGS
0.031   dirname          23832  23808    0 /usr/bin/dirname /apps/tomcat/bin/catalina.sh
0.888   run              23833  2344     0 ./run
0.889   run              23833  2344    -2 /command/bash
0.889   run              23833  2344    -2 /usr/local/bin/bash
0.889   run              23833  2344    -2 /usr/local/sbin/bash
0.889   bash             23833  2344     0 /bin/bash
0.894   svstat           23835  23834    0 /command/svstat /service/nflx-httpd
0.894   perl             23836  23834    0 /usr/bin/perl -e $l=<>;$l=~/(\d+) sec/;print $1||0;
0.899   ps               23838  23837    0 /bin/ps --ppid 1 -o pid,cmd,args
0.900   grep             23839  23837    0 /bin/grep org.apache.catalina
0.900   sed              23840  23837    0 /bin/sed s/^ *//;
0.900   cut              23841  23837    0 /usr/bin/cut -d  -f 1
0.901   xargs            23842  23837    0 /usr/bin/xargs
0.912   xargs            23843  23842   -2 /command/echo
0.912   xargs            23843  23842   -2 /usr/local/bin/echo
0.912   xargs            23843  23842   -2 /usr/local/sbin/echo
0.912   echo             23843  23842    0 /bin/echo
[...]
```

Efficient: only traces exec()

Exonerate or confirm storage latency issues and outliers with `ext4slower`

```
# /usr/share/bcc/tools/ext4slower 1
Tracing ext4 operations slower than 1 ms
TIME     COMM           PID    T BYTES   OFF_KB   LAT(ms) FILENAME
17:31:42 postdrop       15523  S 0       0           2.32 5630D406E4
17:31:42 cleanup        15524  S 0       0           1.89 57BB7406EC
17:32:09 titus-log-ship 19735  S 0       0           1.94 slurper_checkpoint.db
17:35:37 dhclient       1061   S 0       0           3.32 dhclient.eth0.leases
17:35:39 systemd-journa 504    S 0       0          26.62 system.journal
17:35:39 systemd-journa 504    S 0       0           1.56 system.journal
17:35:39 systemd-journa 504    S 0       0           1.73 system.journal
17:35:45 postdrop       16187  S 0       0           2.41 C0369406E4
17:35:45 cleanup        16188  S 0       0           6.52 C1B90406EC
[…]
```

Tracing at the file system is a more reliable and complete indicator than measuring disk I/O latency
Also: btrfsslower, xfsslower, zfsslower

Exonerate or confirm storage latency issues and outliers with `ext4slower`

```
# /usr/share/bcc/tools/ext4slower 1
Tracing ext4 operations slower than 1 ms
TIME     COMM           PID    T BYTES   OFF_KB   LAT(ms) FILENAME
17:31:42 postdrop       15523  S 0       0           2.32 5630D406E4
17:31:42 cleanup        15524  S 0       0           1.89 57BB7406EC
17:32:09 titus-log-ship 19735  S 0       0           1.94 slurper_checkpoint.db
17:35:37 dhclient       1061   S 0       0           3.32 dhclient.eth0.leases
17:35:39 systemd-journa 504    S 0       0          26.62 system.journal
17:35:39 systemd-journa 504    S 0       0           1.56 system.journal
17:35:39 systemd-journa 504    S 0       0           1.73 system.journal
17:35:45 postdrop       16187  S 0       0           2.41 C0369406E4
17:35:45 cleanup        16188  S 0       0           6.52 C1B90406EC
[…]
```

Tracing at the file system is a more reliable and complete indicator than measuring disk I/O latency
Also: btrfsslower, xfsslower, zfsslower

Identify multimodal disk I/O latency and outliers with `biolatency`

```
# biolatency -mT 10
Tracing block device I/O... Hit Ctrl-C to end.

19:19:04
     msecs               : count     distribution
         0 -> 1          : 238      |*********                               |
         2 -> 3          : 424      |*****************                       |
         4 -> 7          : 834      |*********************************       |
         8 -> 15         : 506      |********************                    |
        16 -> 31         : 986      |****************************************|
        32 -> 63         : 97       |***                                     |
        64 -> 127        : 7        |                                        |
       128 -> 255        : 27       |*                                       |

19:19:14
     msecs               : count     distribution
         0 -> 1          : 427      |*******************                     |
         2 -> 3          : 424      |******************                      |
[…]
```

The "count" column is summarized in-kernel

Average latency (iostat/sar) may not be representitive with multiple modes or outliers

Identify multimodal disk I/O latency and outliers with `biolatency`

```
# biolatency -mT 10
Tracing block device I/O... Hit Ctrl-C to end.

19:19:04
     msecs               : count     distribution
         0 -> 1          : 238      |*********                               |
         2 -> 3          : 424      |*****************                       |
         4 -> 7          : 834      |*********************************       |
         8 -> 15         : 506      |********************                    |
        16 -> 31         : 986      |****************************************|
        32 -> 63         : 97       |***                                     |
        64 -> 127        : 7        |                                        |
       128 -> 255        : 27       |*                                       |

19:19:14
     msecs               : count     distribution
         0 -> 1          : 427      |*******************                     |
         2 -> 3          : 424      |******************                      |
[…]
```

The "count" column is summarized in-kernel

Average latency (iostat/sar) may not be representitive with multiple modes or outliers

Efficiently trace TCP sessions with PID, bytes, and duration using `tcplife`

```
# /usr/share/bcc/tools/tcplife
PID   COMM       LADDR           LPORT RADDR           RPORT TX_KB RX_KB MS
2509  java       100.82.34.63    8078  100.82.130.159  12410     0     0 5.44
2509  java       100.82.34.63    8078  100.82.78.215   55564     0     0 135.32
2509  java       100.82.34.63    60778 100.82.207.252  7001      0    13 15126.87
2509  java       100.82.34.63    38884 100.82.208.178  7001      0     0 15568.25
2509  java       127.0.0.1       4243  127.0.0.1       42166     0     0 0.61
2509  java       127.0.0.1       42166 127.0.0.1       4243      0     0 0.67
12030 upload-mes 127.0.0.1       34020 127.0.0.1       8078     11     0 3.38
2509  java       127.0.0.1       8078  127.0.0.1       34020     0    11 3.41
12030 upload-mes 127.0.0.1       21196 127.0.0.1       7101      0     0 12.61
3964  mesos-slav 127.0.0.1       7101  127.0.0.1       21196     0     0 12.64
12021 upload-sys 127.0.0.1       34022 127.0.0.1       8078    372     0 15.28
2509  java       127.0.0.1       8078  127.0.0.1       34022     0   372 15.31
2235  dockerd    100.82.34.63    13730 100.82.136.233  7002      0     4 18.50
2235  dockerd    100.82.34.63    34314 100.82.64.53    7002      0     8 56.73
[...]
```

Dynamic tracing of TCP set state only; does *not* trace send/receive

Also see: tcpconnect, tcpaccept, tcpretrans

Efficiently trace TCP sessions with PID, bytes, and duration using `tcplife`

```
# /usr/share/bcc/tools/tcplife
PID   COMM       LADDR           LPORT RADDR           RPORT TX_KB RX_KB MS
2509  java       100.82.34.63    8078  100.82.130.159  12410     0     0 5.44
2509  java       100.82.34.63    8078  100.82.78.215   55564     0     0 135.32
2509  java       100.82.34.63    60778 100.82.207.252  7001      0    13 15126.87
2509  java       100.82.34.63    38884 100.82.208.178  7001      0     0 15568.25
2509  java       127.0.0.1       4243  127.0.0.1       42166     0     0 0.61
2509  java       127.0.0.1       42166 127.0.0.1       4243      0     0 0.67
12030 upload-mes 127.0.0.1       34020 127.0.0.1       8078     11     0 3.38
2509  java       127.0.0.1       8078  127.0.0.1       34020     0    11 3.41
12030 upload-mes 127.0.0.1       21196 127.0.0.1       7101      0     0 12.61
3964  mesos-slav 127.0.0.1       7101  127.0.0.1       21196     0     0 12.64
12021 upload-sys 127.0.0.1       34022 127.0.0.1       8078    372     0 15.28
2509  java       127.0.0.1       8078  127.0.0.1       34022     0   372 15.31
2235  dockerd    100.82.34.63    13730 100.82.136.233  7002      0     4 18.50
2235  dockerd    100.82.34.63    34314 100.82.64.53    7002      0     8 56.73
[...]
```

Dynamic tracing of TCP set state only; does *not* trace send/receive

Also see: tcpconnect, tcpaccept, tcpretrans

Identify DNS latency issues system wide with `gethostlatency`

```
# /usr/share/bcc/tools/gethostlatency
TIME      PID    COMM                  LATms HOST
18:56:36  5055   mesos-slave            0.01 100.82.166.217
18:56:40  5590   java                   3.53 ec2-…-79.compute-1.amazonaws.com
18:56:51  5055   mesos-slave            0.01 100.82.166.217
18:56:53  30166  ncat                   0.21 localhost
18:56:56  6661   java                   2.19 atlas-alert-….prod.netflix.net
18:56:59  5589   java                   1.50 ec2-…-207.compute-1.amazonaws.com
18:57:03  5370   java                   0.04 localhost
18:57:03  30259  sudo                   0.07 titusagent-mainvpc-m…3465
18:57:06  5055   mesos-slave            0.01 100.82.166.217
18:57:10  5590   java                   3.10 ec2-…-79.compute-1.amazonaws.com
18:57:21  5055   mesos-slave            0.01 100.82.166.217
18:57:29  5589   java                  52.36 ec2-…-207.compute-1.amazonaws.com
18:57:36  5055   mesos-slave            0.01 100.82.166.217
18:57:40  5590   java                   1.83 ec2-…-79.compute-1.amazonaws.com
18:57:51  5055   mesos-slave            0.01 100.82.166.217
[…]
```

Instruments using user-level dynamic tracing of getaddrinfo(), gethostbyname(), etc.

Identify DNS latency issues system wide with `gethostlatency`

```
# /usr/share/bcc/tools/gethostlatency
TIME      PID    COMM                  LATms HOST
18:56:36  5055   mesos-slave            0.01 100.82.166.217
18:56:40  5590   java                   3.53 ec2-…-79.compute-1.amazonaws.com
18:56:51  5055   mesos-slave            0.01 100.82.166.217
18:56:53  30166  ncat                   0.21 localhost
18:56:56  6661   java                   2.19 atlas-alert-….prod.netflix.net
18:56:59  5589   java                   1.50 ec2-…-207.compute-1.amazonaws.com
18:57:03  5370   java                   0.04 localhost
18:57:03  30259  sudo                   0.07 titusagent-mainvpc-m…3465
18:57:06  5055   mesos-slave            0.01 100.82.166.217
18:57:10  5590   java                   3.10 ec2-…-79.compute-1.amazonaws.com
18:57:21  5055   mesos-slave            0.01 100.82.166.217
18:57:29  5589   java                  52.36 ec2-…-207.compute-1.amazonaws.com
18:57:36  5055   mesos-slave            0.01 100.82.166.217
18:57:40  5590   java                   1.83 ec2-…-79.compute-1.amazonaws.com
18:57:51  5055   mesos-slave            0.01 100.82.166.217
[…]
```

Instruments using user-level dynamic tracing of getaddrinfo(), gethostbyname(), etc.

Examine CPU scheduler run queue latency as a histogram with `runqlat`

```
# /usr/share/bcc/tools/runqlat 10
Tracing run queue latency... Hit Ctrl-C to end.

     usecs               : count     distribution
         0 -> 1          : 2810     |*                                       |
         2 -> 3          : 5248     |**                                      |
         4 -> 7          : 12369    |******                                  |
         8 -> 15         : 71312    |****************************************|
        16 -> 31         : 55705    |*******************************         |
        32 -> 63         : 11775    |******                                  |
        64 -> 127        : 6230     |***                                     |
       128 -> 255        : 2758     |*                                       |
       256 -> 511        : 549      |                                        |
       512 -> 1023       : 46       |                                        |
      1024 -> 2047       : 11       |                                        |
      2048 -> 4095       : 4        |                                        |
      4096 -> 8191       : 5        |                                        |
[…]
```

As efficient as possible: scheduler calls can become frequent

Examine CPU scheduler run queue latency as a histogram with `runqlat`

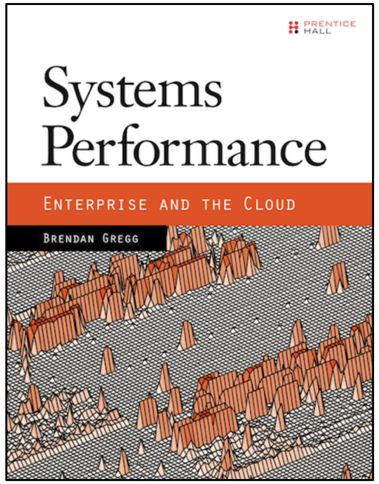
```
# /usr/share/bcc/tools/runqlat 10
Tracing run queue latency... Hit Ctrl-C to end.

     usecs               : count     distribution
         0 -> 1          : 2810     |*                                       |
         2 -> 3          : 5248     |**                                      |
         4 -> 7          : 12369    |******                                  |
         8 -> 15         : 71312    |****************************************|
        16 -> 31         : 55705    |*******************************         |
        32 -> 63         : 11775    |******                                  |
        64 -> 127        : 6230     |***                                     |
       128 -> 255        : 2758     |*                                       |
       256 -> 511        : 549      |                                        |
       512 -> 1023       : 46       |                                        |
      1024 -> 2047       : 11       |                                        |
      2048 -> 4095       : 4        |                                        |
      4096 -> 8191       : 5        |                                        |
[…]
```

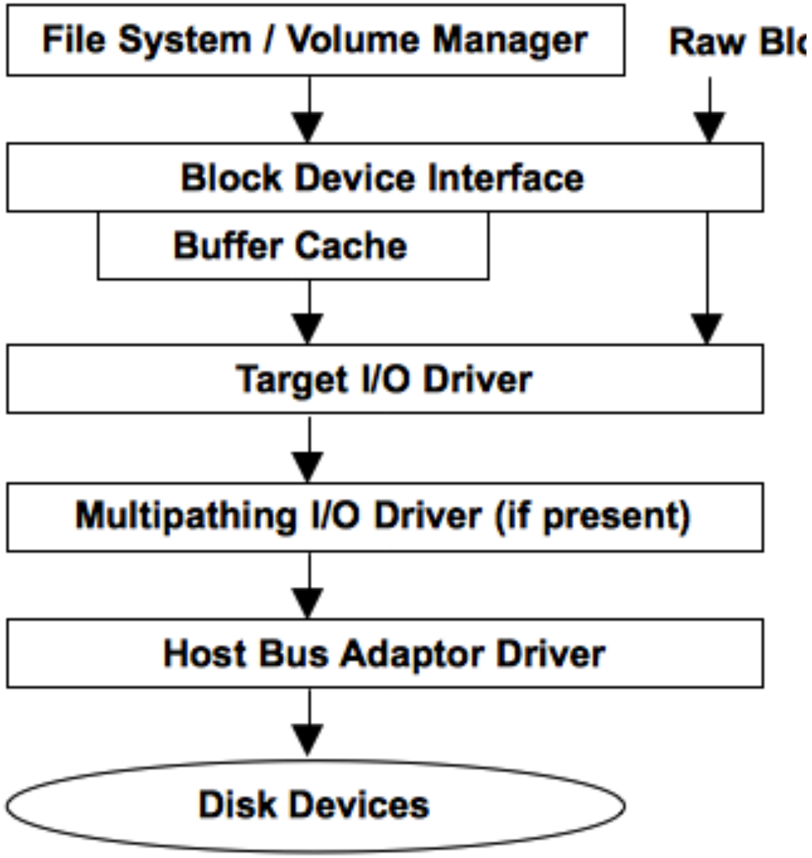
As efficient as possible: scheduler calls can become frequent

Advanced Analysis

- Find/draw a functional diagram
- Apply performance methods
  http://www.brendangregg.com/methodology.html
  1. Workload Characterization
  2. Latency Analysis
  3. USE Method
- Start with the Q's,
  then find the A's
- Use multi-tools:
  – funccount, trace, argdist, stackcount



e.g., storage I/O subsystem:



Construct programmatic one-liners with `trace`

e.g. reads over 20000 bytes:

```
# trace 'sys_read (arg3 > 20000) "read %d bytes", arg3'
TIME     PID    COMM         FUNC             -
05:18:23 4490   dd           sys_read         read 1048576 bytes
05:18:23 4490   dd           sys_read         read 1048576 bytes
05:18:23 4490   dd           sys_read         read 1048576 bytes
^C
```

```
# trace -h
[...]
trace –K blk_account_io_start
        Trace this kernel function, and print info with a kernel stack trace

trace 'do_sys_open "%s", arg2'
        Trace the open syscall and print the filename being opened

trace 'sys_read (arg3 > 20000) "read %d bytes", arg3'
        Trace the read syscall and print a message for reads >20000 bytes

trace r::do_sys_return
        Trace the return from the open syscall

trace 'c:open (arg2 == 42) "%s %d", arg1, arg2'
        Trace the open() call from libc only if the flags (arg2) argument is 42
[...]
```

argdist by Sasha Goldshtein

Create in-kernel summaries with `argdist`

e.g. histogram of tcp_cleanup_rbuf() copied:

```
# argdist -H 'p::tcp_cleanup_rbuf(struct sock *sk, int copied):int:copied'
[15:34:45]
     copied              : count     distribution
         0 -> 1          : 15088    |**********************************      |
         2 -> 3          : 0        |                                        |
         4 -> 7          : 0        |                                        |
         8 -> 15         : 0        |                                        |
        16 -> 31         : 0        |                                        |
        32 -> 63         : 0        |                                        |
        64 -> 127        : 4786     |***********                             |
       128 -> 255        : 1        |                                        |
       256 -> 511        : 1        |                                        |
       512 -> 1023       : 4        |                                        |
      1024 -> 2047       : 11       |                                        |
      2048 -> 4095       : 5        |                                        |
      4096 -> 8191       : 27       |                                        |
      8192 -> 16383      : 105      |                                        |
     16384 -> 32767      : 0        |                                        |
```

argdist by Sasha Goldshtein

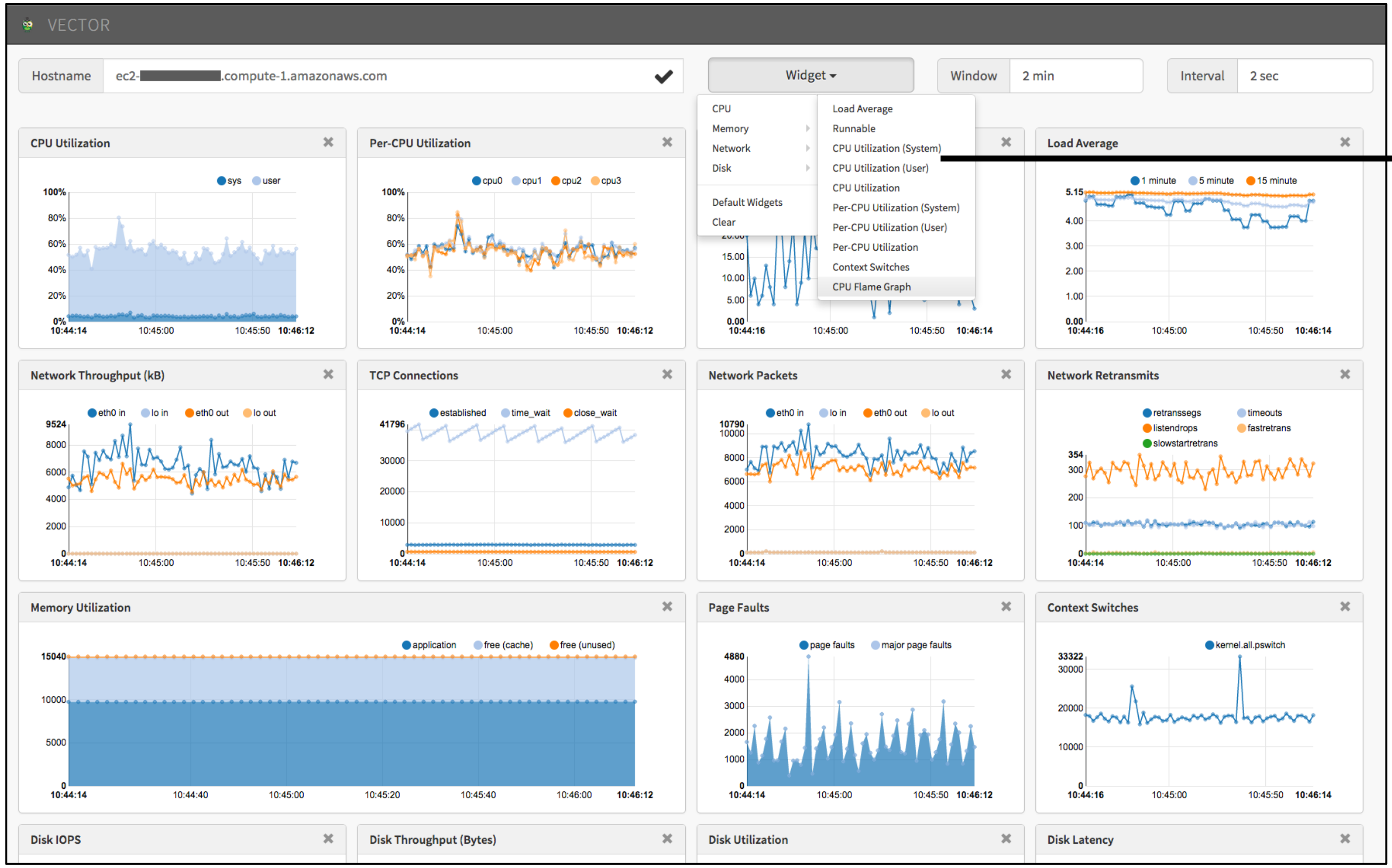
Coming to a GUI near you

BCC/BPF VISUALIZATIONS

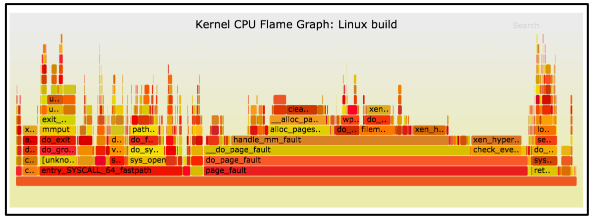
BPF metrics and analysis can be automated in GUIs

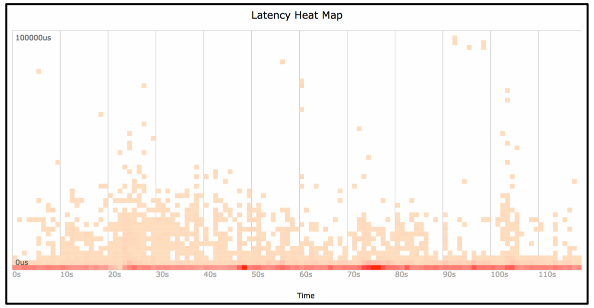
Eg, Netflix Vector (self-service UI):

Vector



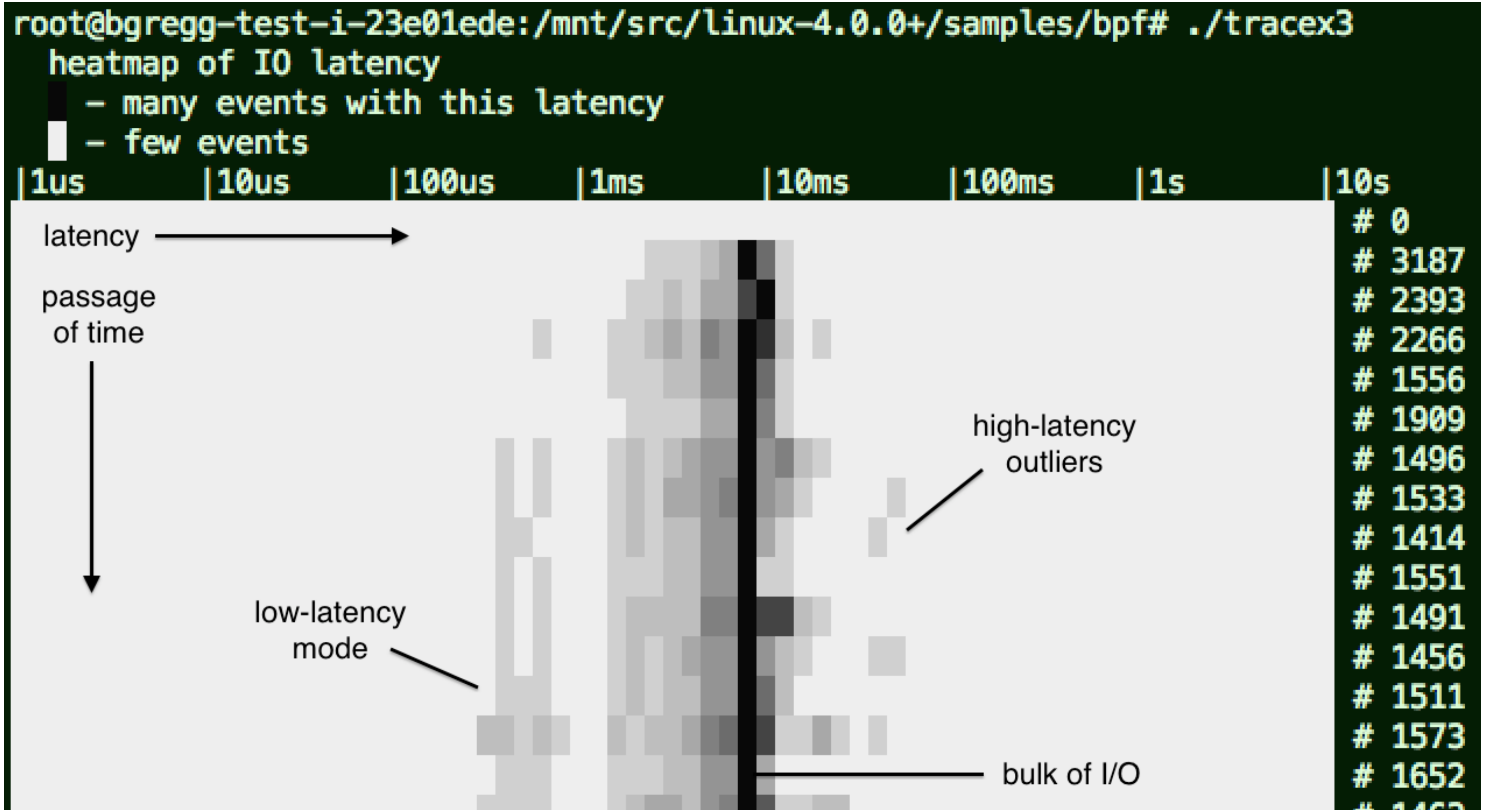
Flame Graphs
Heat Maps
Tracing Reports
…



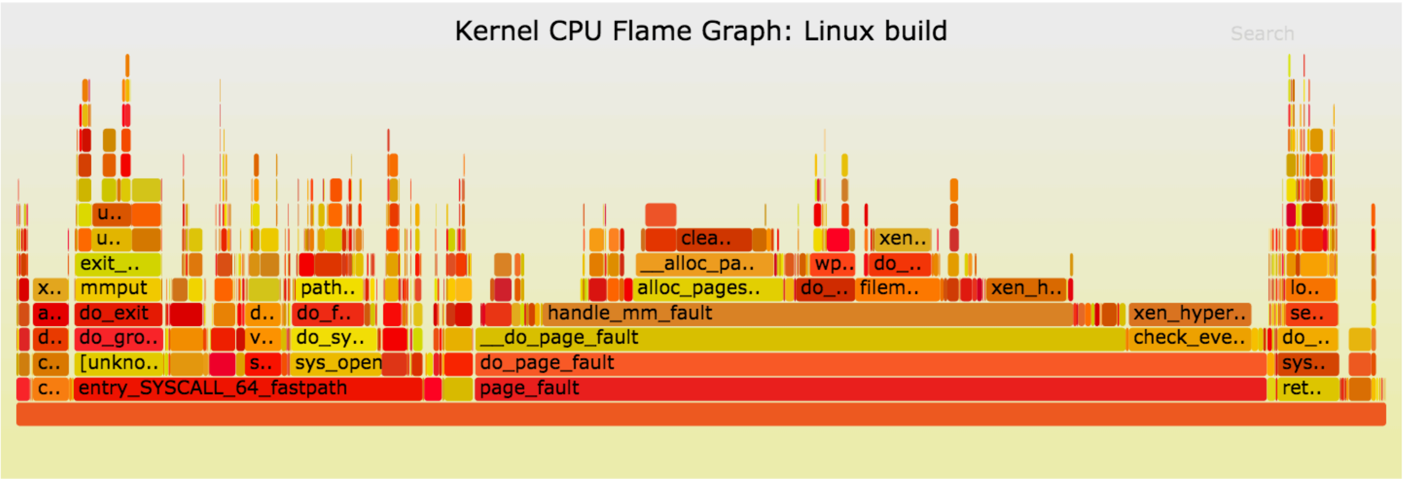


Should be open sourced; you may also build/buy your own

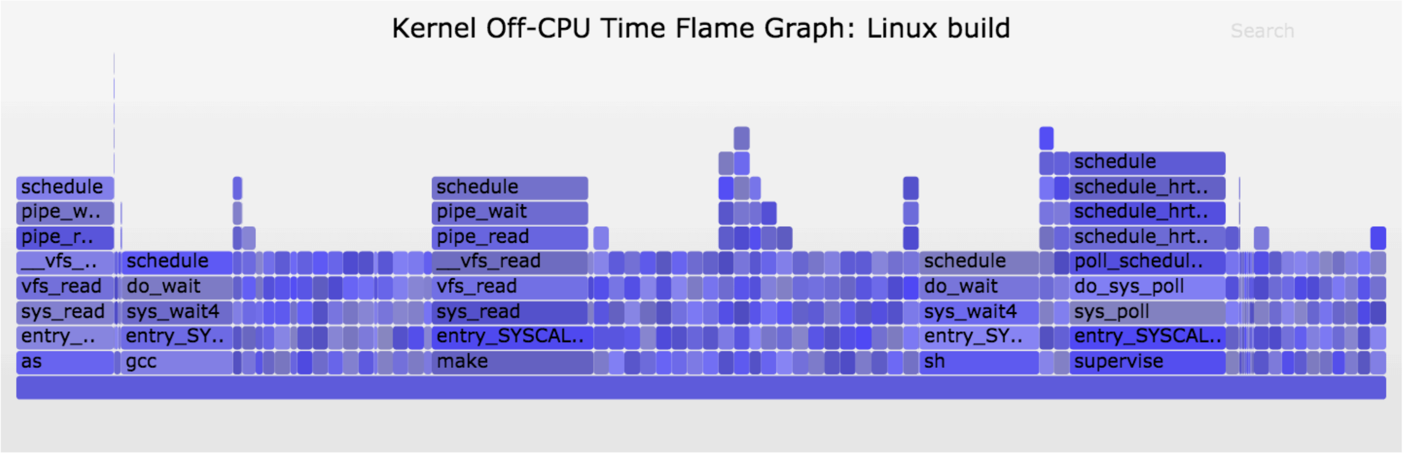
Latency heatmaps show histograms over time



Efficient CPU and off-CPU flame graphs by counting stacks in kernel context

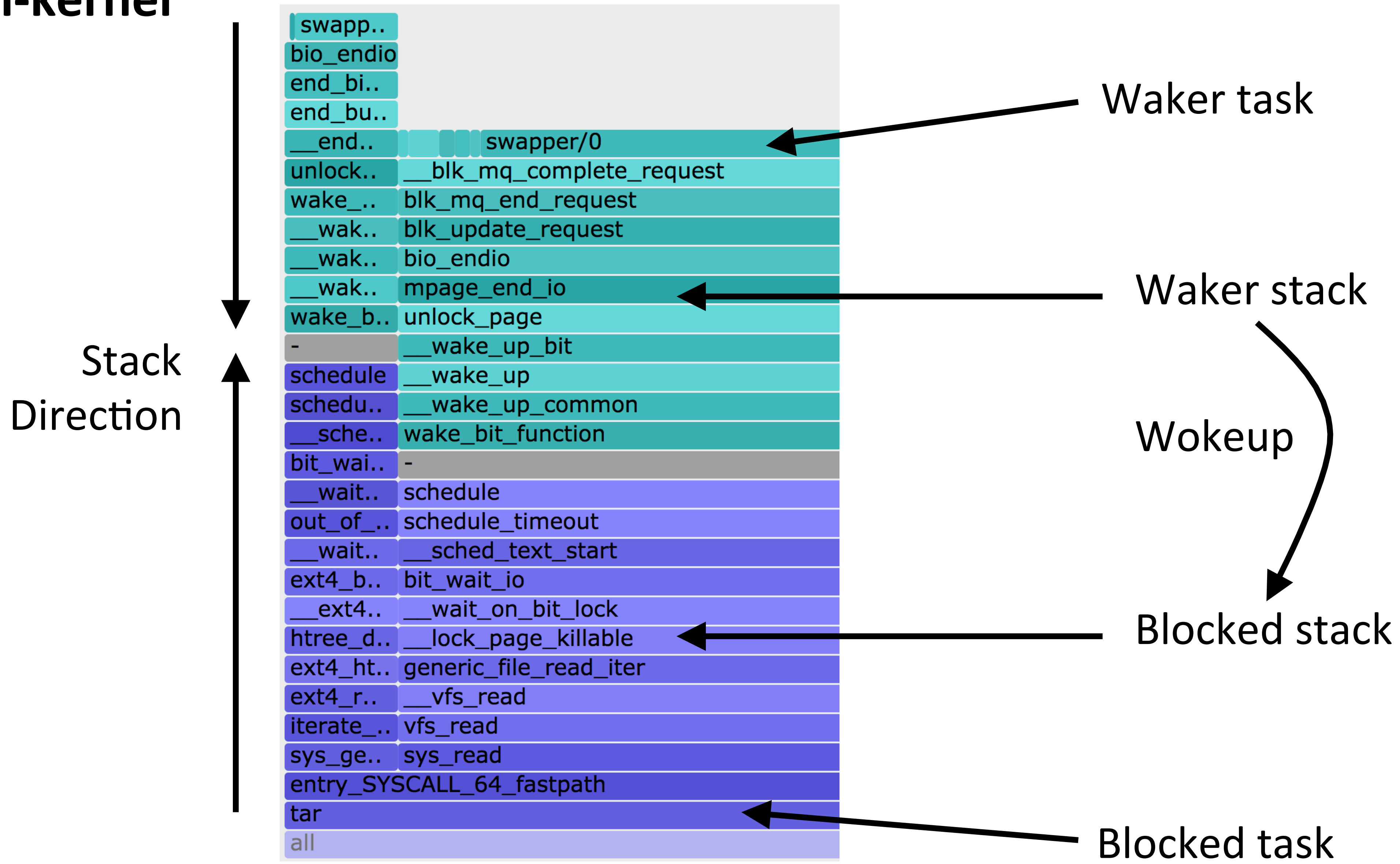


CPU



Off-CPU

Advanced off-CPU analysis: BPF can merge the blocking stack with the waker stack in-kernel



BPF

FUTURE WORK

BCC Improvements

Challenges:

- Initialize all variables
- Extra bpf_probe_read()s
- BPF_PERF_OUTPUT()
- Verifier errors

```
struct sock *skp = NULL;
bpf_probe_read(&skp, sizeof(skp), &sk);

// pull in details
u16 family = 0, lport = 0, dport = 0;
char state = 0;
bpf_probe_read(&family, sizeof(family), &skp->__sk_common
bpf_probe_read(&lport, sizeof(lport), &skp->__sk_common.s
bpf_probe_read(&dport, sizeof(dport), &skp->__sk_common.s
bpf_probe_read(&state, sizeof(state), (void *)&skp->__sk_

if (family == AF_INET) {
    struct ipv4_data_t data4 = {.pid = pid, .ip = 4, .typ
    bpf_probe_read(&data4.saddr, sizeof(u32),
        &skp->__sk_common.skc_rcv_saddr);
    bpf_probe_read(&data4.daddr, sizeof(u32),
        &skp->__sk_common.skc_daddr);
    // lport is host order
    data4.lport = lport;
    data4.dport = ntohs(dport);
    data4.state = state;
    ipv4_events.perf_submit(ctx, &data4, sizeof(data4));
```

Higher-level Language

- bcc's Python/C interface is ok, but verbose
- Alternate higher-level language front end?
  - New front-ends can use existing libbcc, and can be added as part of bcc itself
  - Whave a problem in search of a new language (instead of the other way around)

ply

- A new BPF-based language and tracer for Linux
  - Created by Tobias Waldekranz
  - https://github.com/iovisor/ply https://wkz.github.io/ply/
- High-level language
  - Simple one-liners
  - Short scripts
- In development (?)
  - kprobes and tracepoints only, uprobes/perf_events not yet
  - Successful so far as a proof of concept
  - Not production tested yet (bcc is)

ply

/* a dynamic tracer for Linux */

File opens can be traced using a short ply one-liner

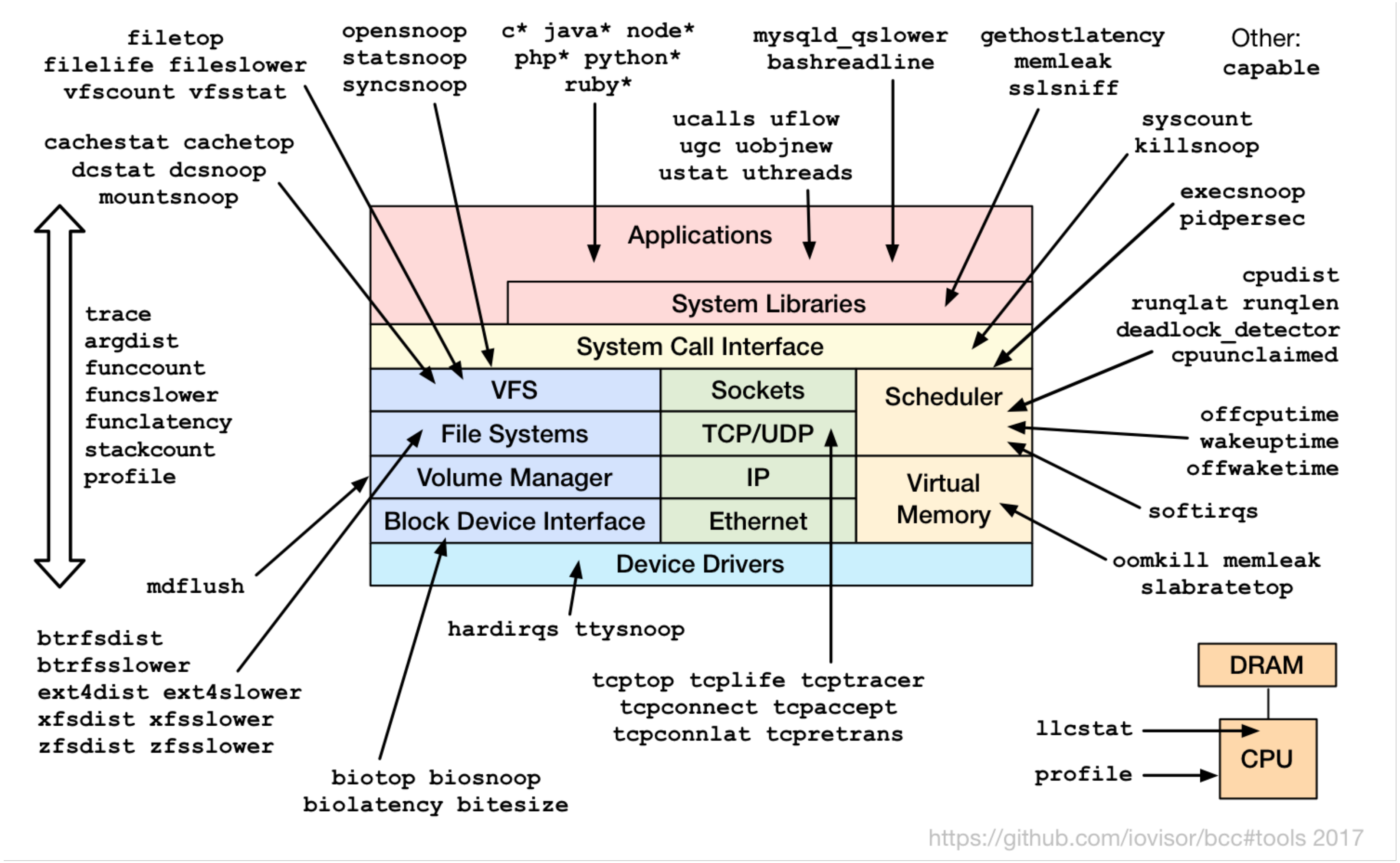
```
# ply -c 'kprobe:do_sys_open { printf("opened: %s\n", mem(arg(1), "128s")); }'
1 probe active
opened: /sys/kernel/debug/tracing/events/enable
opened: /etc/ld.so.cache
opened: /lib/x86_64-linux-gnu/libselinux.so.1
opened: /lib/x86_64-linux-gnu/libc.so.6
opened: /lib/x86_64-linux-gnu/libpcre.so.3
opened: /lib/x86_64-linux-gnu/libdl.so.2
opened: /lib/x86_64-linux-gnu/libpthread.so.0
opened: /proc/filesystems
opened: /usr/lib/locale/locale-archive
opened: .
[...]
```

ply programs are concise, such as measuring read latency

```
# ply -A -c 'kprobe:SyS_read { @start[tid()] = nsecs(); }
    kretprobe:SyS_read /@start[tid()]/ { @ns.quantize(nsecs() - @start[tid()]);
        @start[tid()] = nil; }'
2 probes active
^Cde-activating probes
[...]
@ns:

     [ 512,   1k)          3 |########                        |
     [  1k,   2k)          7 |###################             |
     [  2k,   4k)         12 |################################|
     [  4k,   8k)          3 |########                        |
     [  8k,  16k)          2 |#####                           |
     [ 16k,  32k)          0 |                                |
     [ 32k,  64k)          0 |                                |
     [ 64k, 128k)          3 |########                        |
     [128k, 256k)          1 |###                             |
     [256k, 512k)          1 |###                             |
     [512k,   1M)          2 |#####                           |
[...]
```

New Tooling/Metrics



New Visualizations

Case Studies

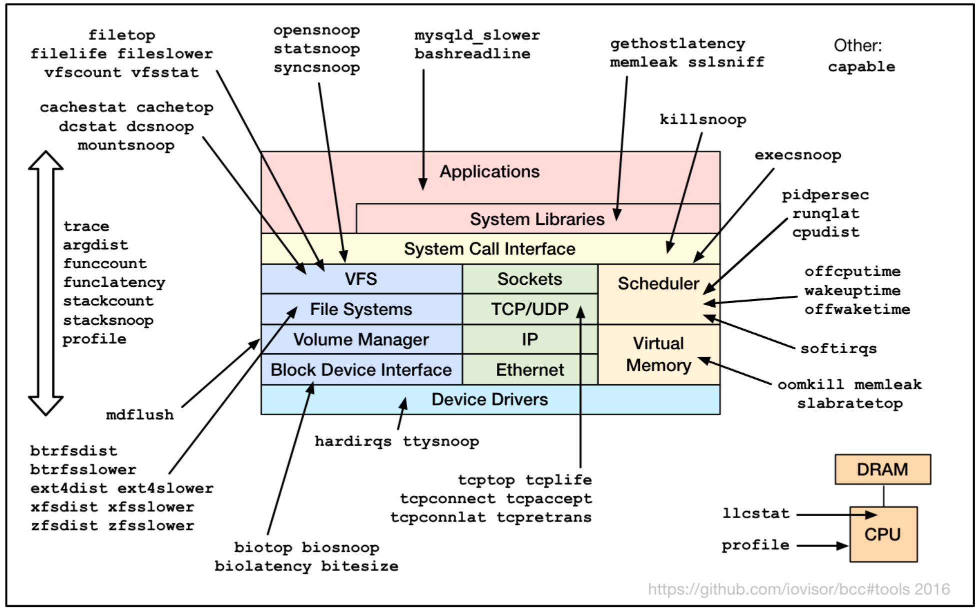
- Use it
- Solve something
- Write about it
- Talk about it

Take aways

1. Understand Linux tracing components
2. Understand the role and state of enhanced BPF
3. Discover opportunities for future development

Please contribute:
- https://github.com/iovisor/bcc
- https://github.com/iovisor/ply

BPF Tracing in Linux
- 3.19: sockets
- 3.19: maps
- 4.1: kprobes
- 4.3: uprobes
- 4.4: BPF output
- 4.6: stacks
- 4.7: tracepoints
- 4.9: profiling
- 4.9: PMCs



Links & References

iovisor bcc:

- https://github.com/iovisor/bcc https://github.com/iovisor/bcc/tree/master/docs
- http://www.brendangregg.com/blog/ (search for "bcc")
- http://www.brendangregg.com/ebpf.html#bcc
- http://blogs.microsoft.co.il/sasha/2016/02/14/two-new-ebpf-tools-memleak-and-argdist/
- On designing tracing tools: https://www.youtube.com/watch?v=uibLwoVKjec

bcc tutorial:

- https://github.com/iovisor/bcc/blob/master/INSTALL.md
- …/docs/tutorial.md
- …/docs/tutorial_bcc_python_developer.md
- …/docs/reference_guide.md
- …/CONTRIBUTING-SCRIPTS.md

ply: https://github.com/iovisor/ply

BPF:

- https://www.kernel.org/doc/Documentation/networking/filter.txt
- https://github.com/iovisor/bpf-docs
- https://suchakra.wordpress.com/tag/bpf/

Flame Graphs:

- http://www.brendangregg.com/flamegraphs.html
- http://www.brendangregg.com/blog/2016-01-20/ebpf-offcpu-flame-graph.html
- http://www.brendangregg.com/blog/2016-02-01/linux-wakeup-offwake-profiling.html

Netflix Tech Blog on Vector:

- http://techblog.netflix.com/2015/04/introducing-vector-netflixs-on-host.html

Linux Performance: http://www.brendangregg.com/linuxperf.html

Thank You

- Questions?
- iovisor bcc: https://github.com/iovisor/bcc
- http://www.brendangregg.com
- http://slideshare.net/brendangregg
- bgregg@netflix.com
- @brendangregg

Thanks to Alexei Starovoitov (Facebook), Brenden Blanco (PLUMgrid/VMware), Sasha Goldshtein (Sela), Daniel Borkmann (Cisco), Wang Nan (Huawei), and other BPF and bcc contributors!