

Linux 4.x Tracing Tools Using BPF Superpowers

Brendan Gregg, Netflix

bgregg@netflix.com

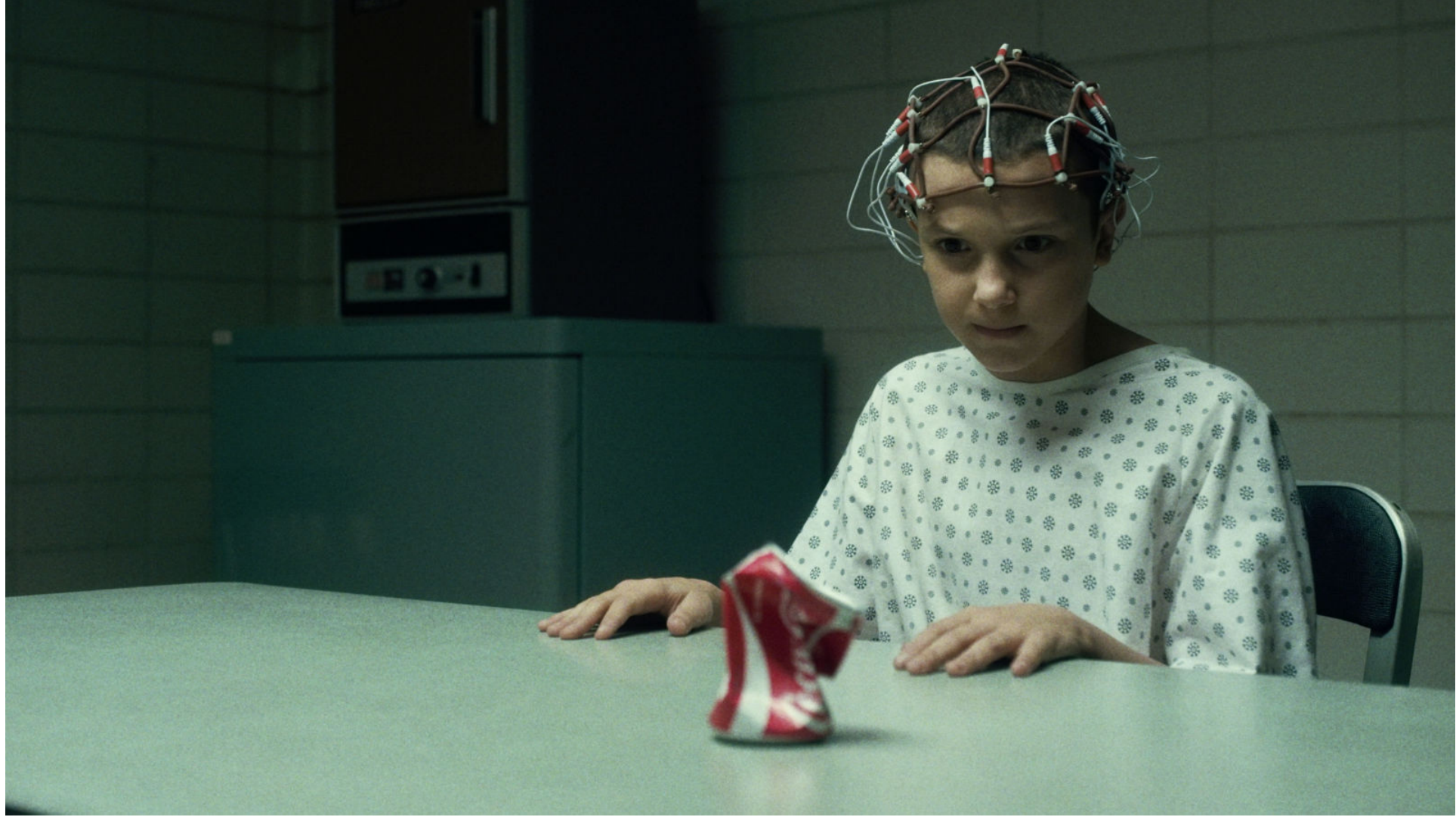
usenix

LISA16

December 4–9, 2016 | Boston, MA

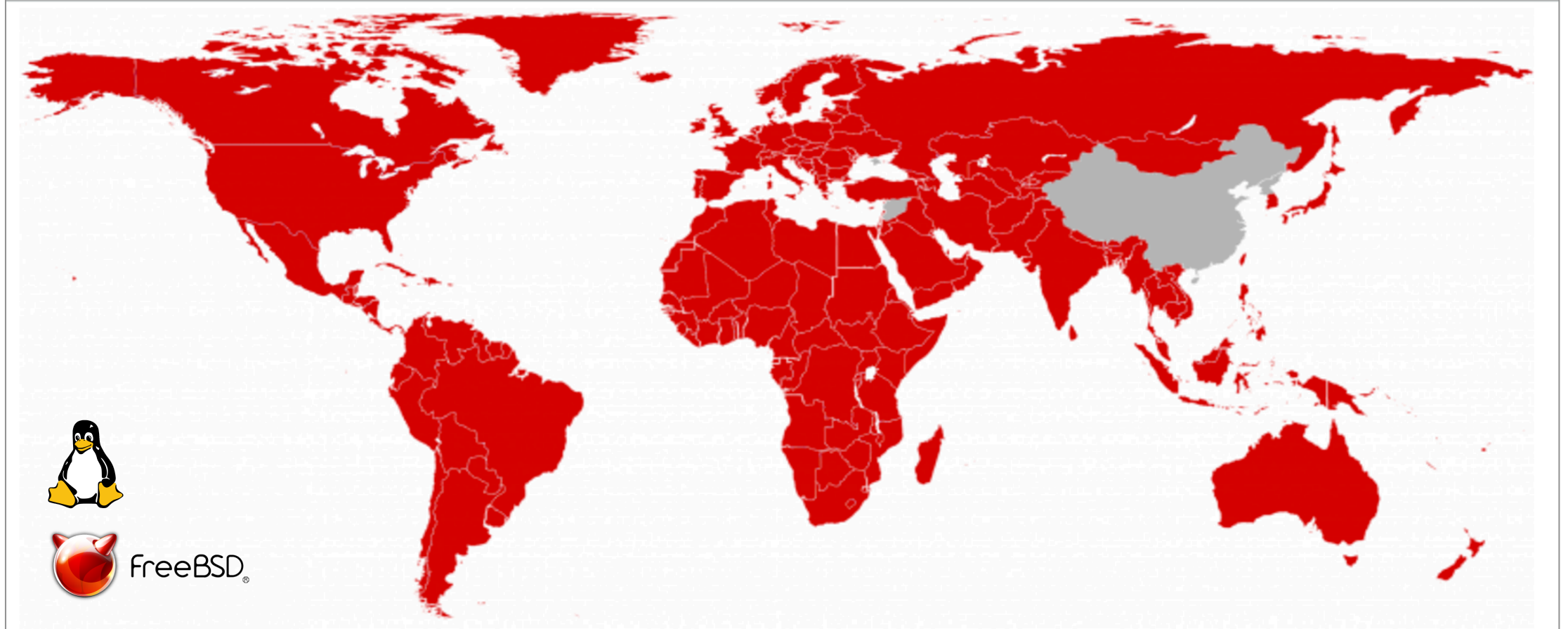
www.usenix.org/lisa16

#lisa16



NETFLIX

REGIONS WHERE NETFLIX IS AVAILABLE



FreeBSD®

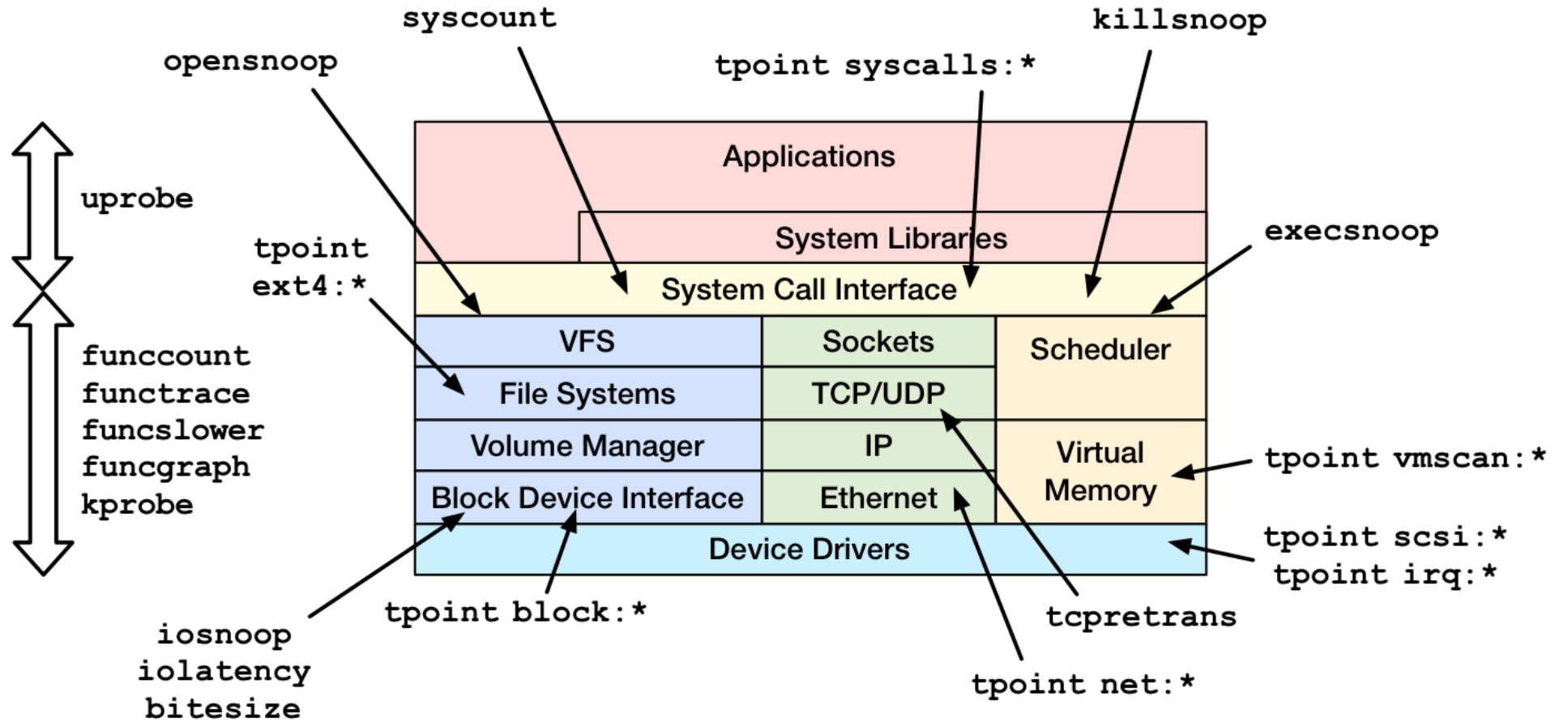
Demo time

**GIVE ME 15 MINUTES
AND I'LL CHANGE YOUR VIEW
OF LINUX TRACING**

Demo

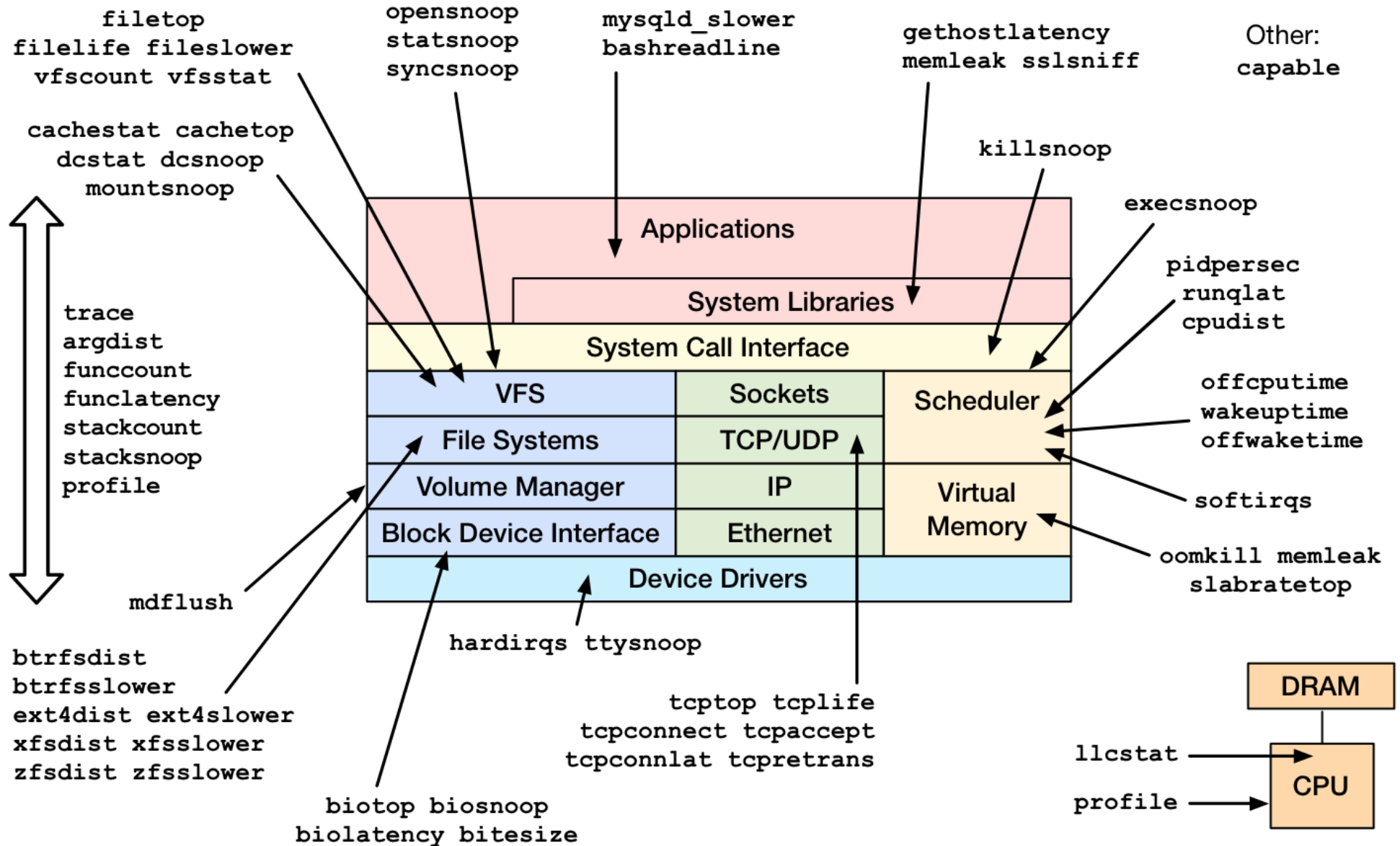
LISA 2014

perf-tools (ftrace)



LISA 2016

bcc tools (BPF)



Wielding Superpowers

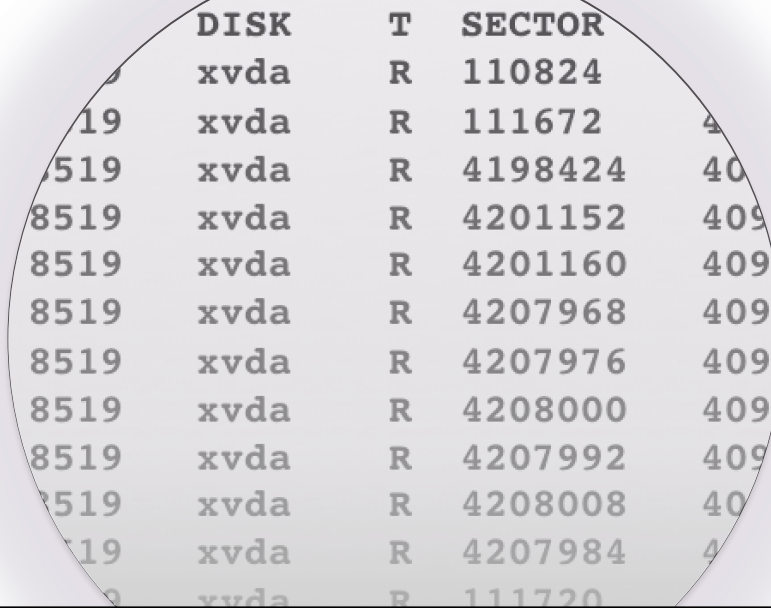
WHAT DYNAMIC TRACING CAN DO

Previously

- Metrics were vendor chosen, closed source, and incomplete
- The art of inference & making do

```
# ps alx
F S UID    PID    PPID  CPU  PRI  NICE  ADDR  SZ  WCHAN  TTY  TIME  CMD
3 S  0      0      0     0    0    20   2253  2   4412  ?   186:14  swapper
1 S  0      1      0     0    30   20   2423  8   46520 ?    0:00  /etc/init
1 S  0     16     1     0    30   20   2273  11  46554 co   0:00  -sh
[...]
```

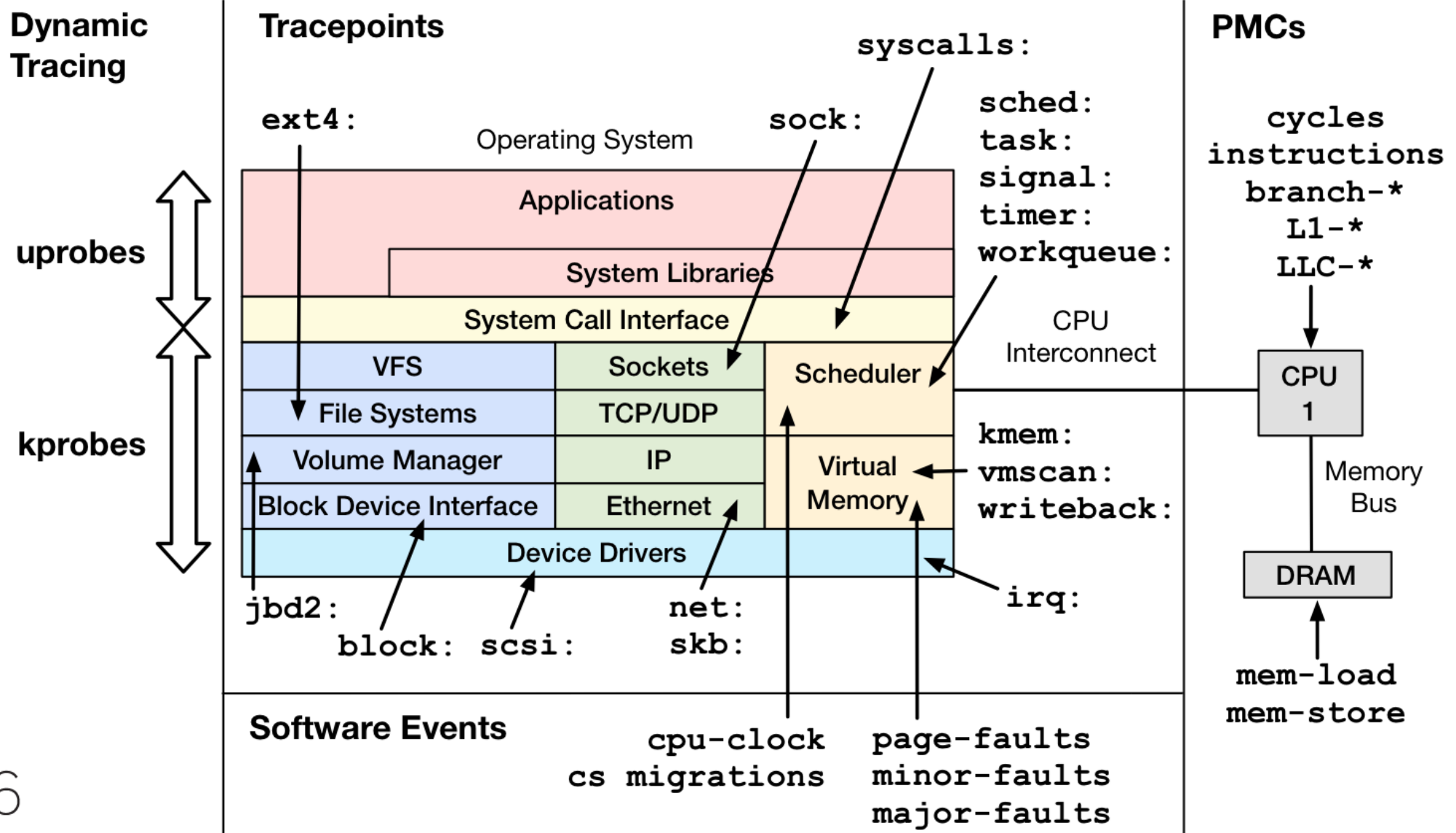
Crystal Ball Observability

A crystal ball containing a table of disk I/O data. The data is presented in a monospaced font, typical of system logs. The table has four columns: DISK, T, SECTOR, and an unlabeled column. The rows show various disk operations on the xvda device, including reads and writes at different sector addresses.

| | DISK | T | SECTOR | |
|------|------|---|---------|-----|
| | xvda | R | 110824 | |
| 19 | xvda | R | 111672 | 4 |
| 519 | xvda | R | 4198424 | 409 |
| 8519 | xvda | R | 4201152 | 409 |
| 8519 | xvda | R | 4201160 | 409 |
| 8519 | xvda | R | 4207968 | 409 |
| 8519 | xvda | R | 4207976 | 409 |
| 8519 | xvda | R | 4208000 | 409 |
| 8519 | xvda | R | 4207992 | 409 |
| 8519 | xvda | R | 4208008 | 409 |
| 8519 | xvda | R | 4207984 | 409 |
| 8519 | xvda | R | 111720 | 4 |

Dynamic Tracing

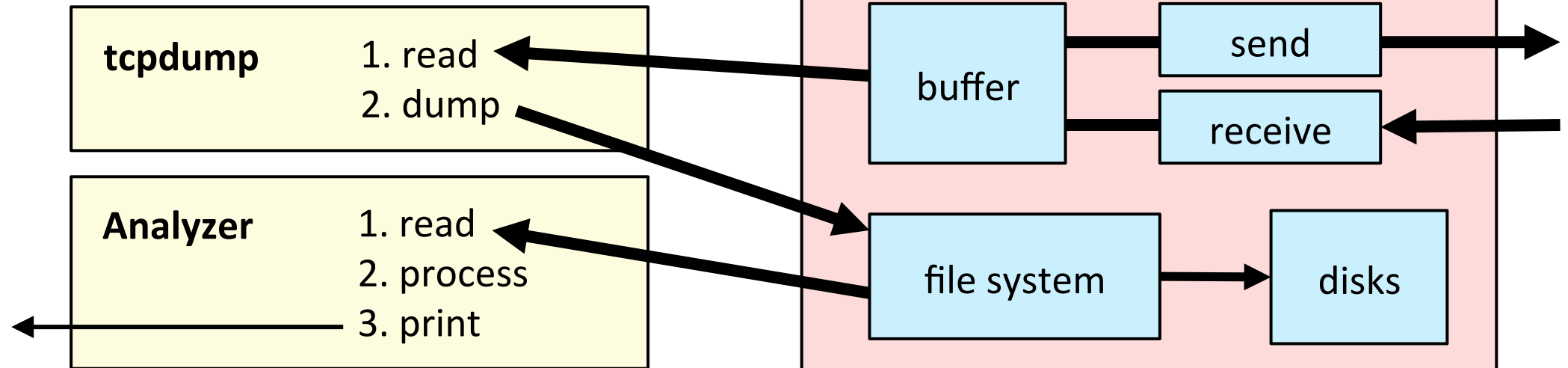
Linux Event Sources



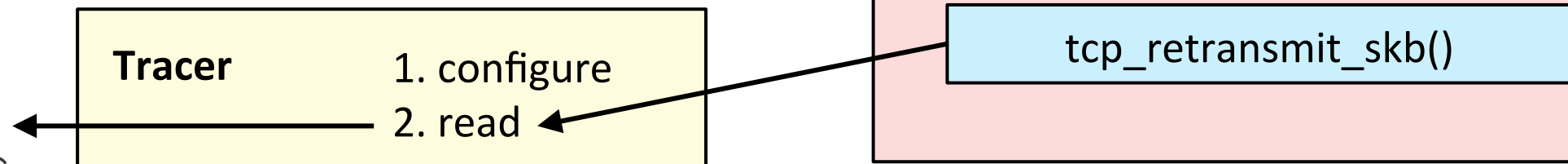
Event Tracing Efficiency

Eg, tracing TCP retransmits

Old way: packet capture



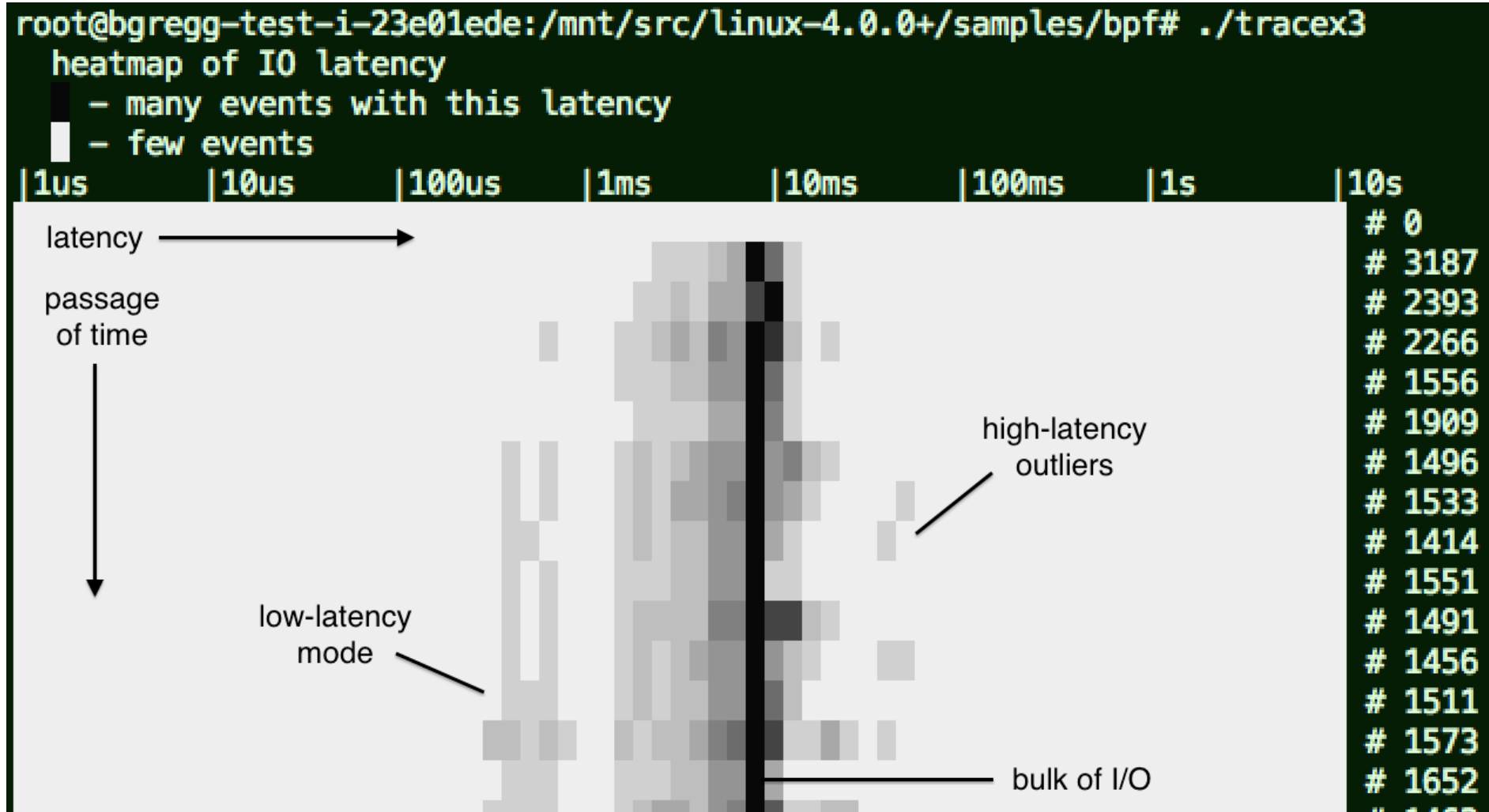
New way: dynamic tracing



New CLI Tools

```
# biolatency
Tracing block device I/O... Hit Ctrl-C to end.
^C
      usecs          : count      distribution
      4 -> 7         : 0
      8 -> 15        : 0
     16 -> 31        : 0
     32 -> 63        : 0
     64 -> 127       : 1
    128 -> 255       : 12
    256 -> 511       : 15
    512 -> 1023      : 43
   1024 -> 2047      : 52
   2048 -> 4095      : 47
   4096 -> 8191      : 52
   8192 -> 16383     : 36
  16384 -> 32767     : 15
  32768 -> 65535     : 2
  65536 -> 131071    : 2
```

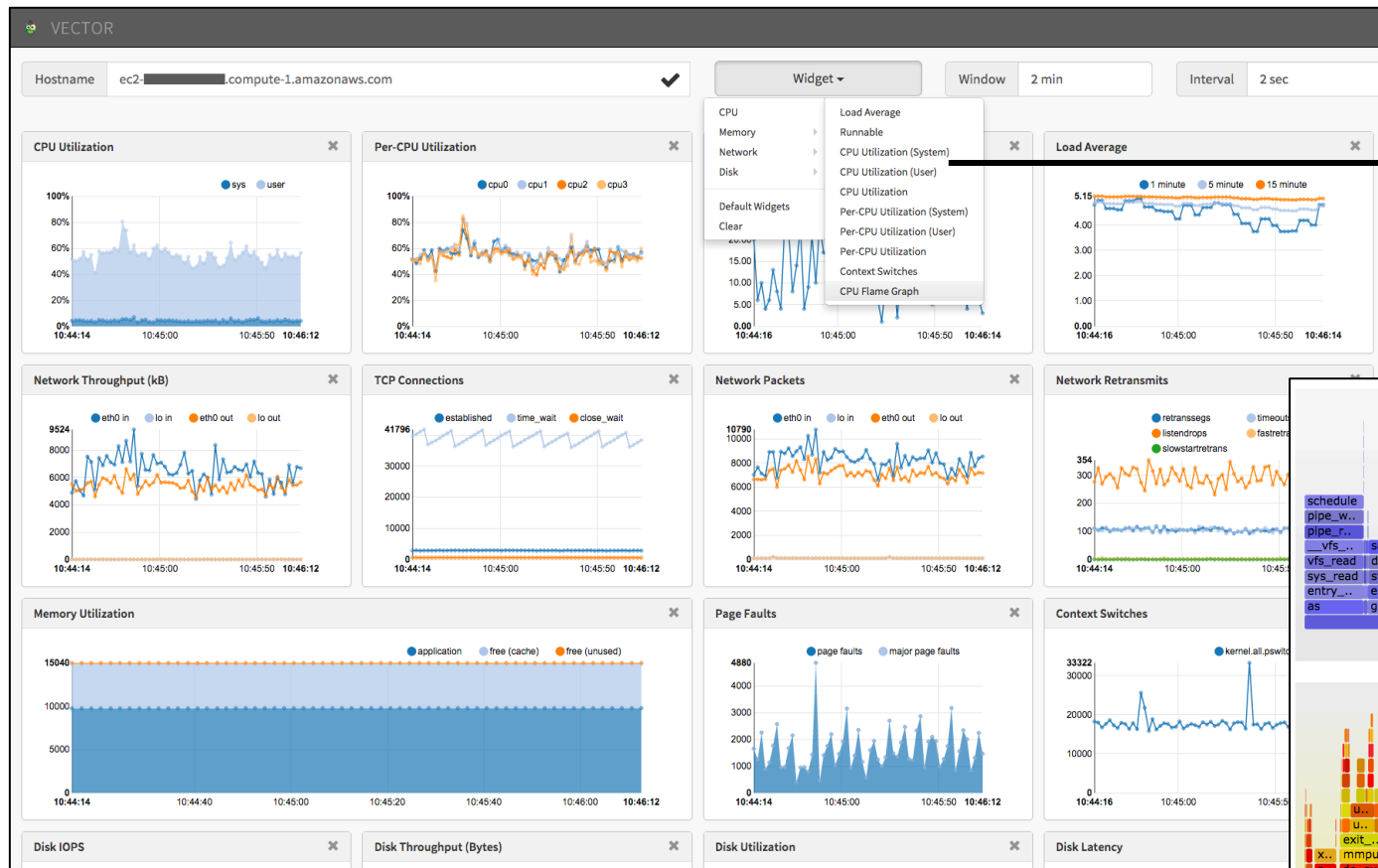
New Visualizations and GUIs



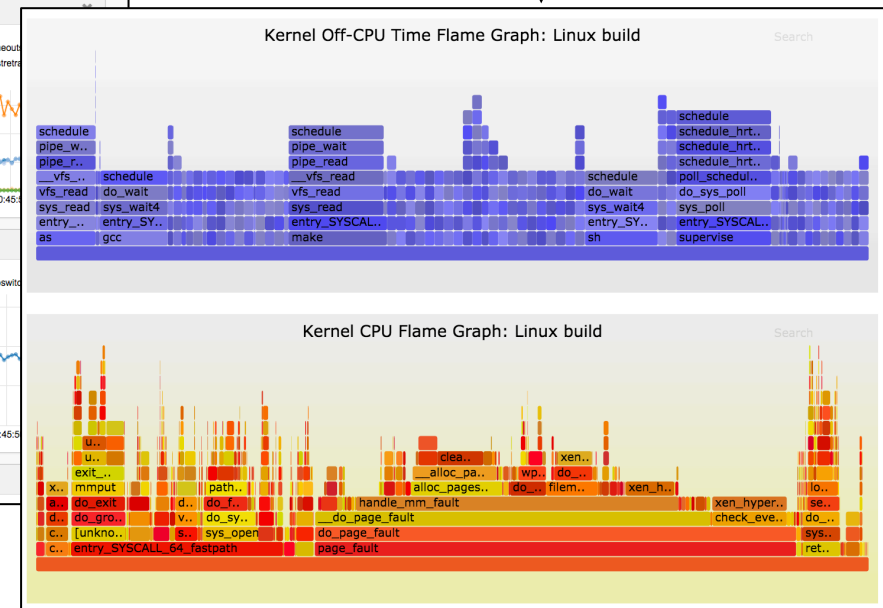
Netflix Intended Usage



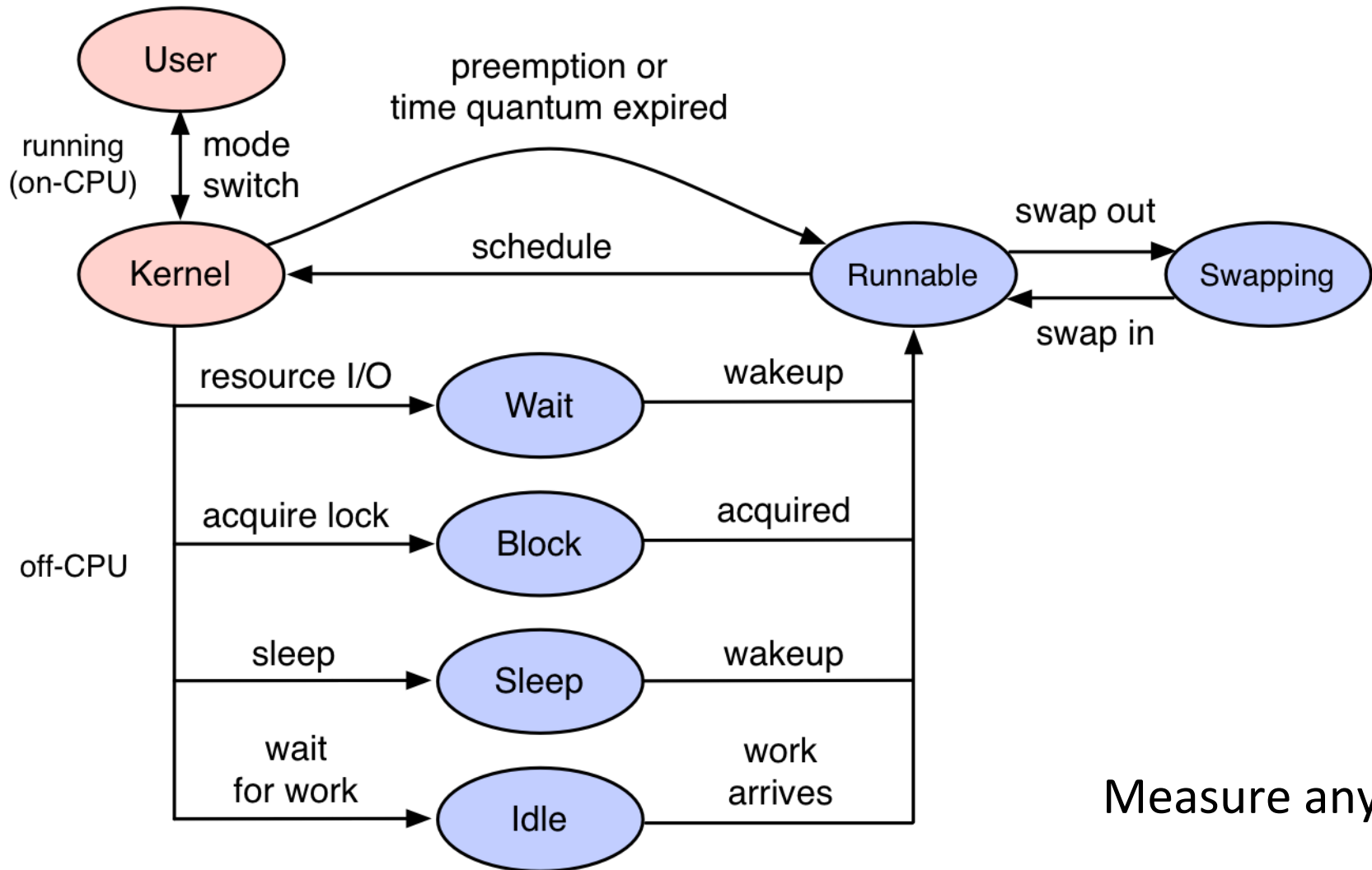
Self-service UI:



Flame Graphs
Tracing Reports
...



Conquer Performance



Measure anything

Introducing BPF

BPF TRACING

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 (2.6.31)
- 2009: tracepoints (2.6.32)
- 2010-2016: ftrace & perf_events enhancements
- 2014-2016: BPF patches

also: LTTng, ktap, sysdig, ...

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) ldx      4*([14]&0xf)
(013) ldh      [x + 14]
(014) jeq      #0x16          jt 17   jf 15
(015) ldh      [x + 16]
(016) jeq      #0x16          jt 17   jf 18
(017) ret      #65535
(018) ret      #0
```

BPF Enhancements by Linux Version

- 3.18: bpf syscall
- 3.19: sockets
- 4.1: kprobes
- 4.4: bpf_perf_event_output
- 4.6: stack traces
- 4.7: tracepoints
- 4.9: profiling

eg, Ubuntu:

16.04

16.10

Enhanced BPF is in Linux

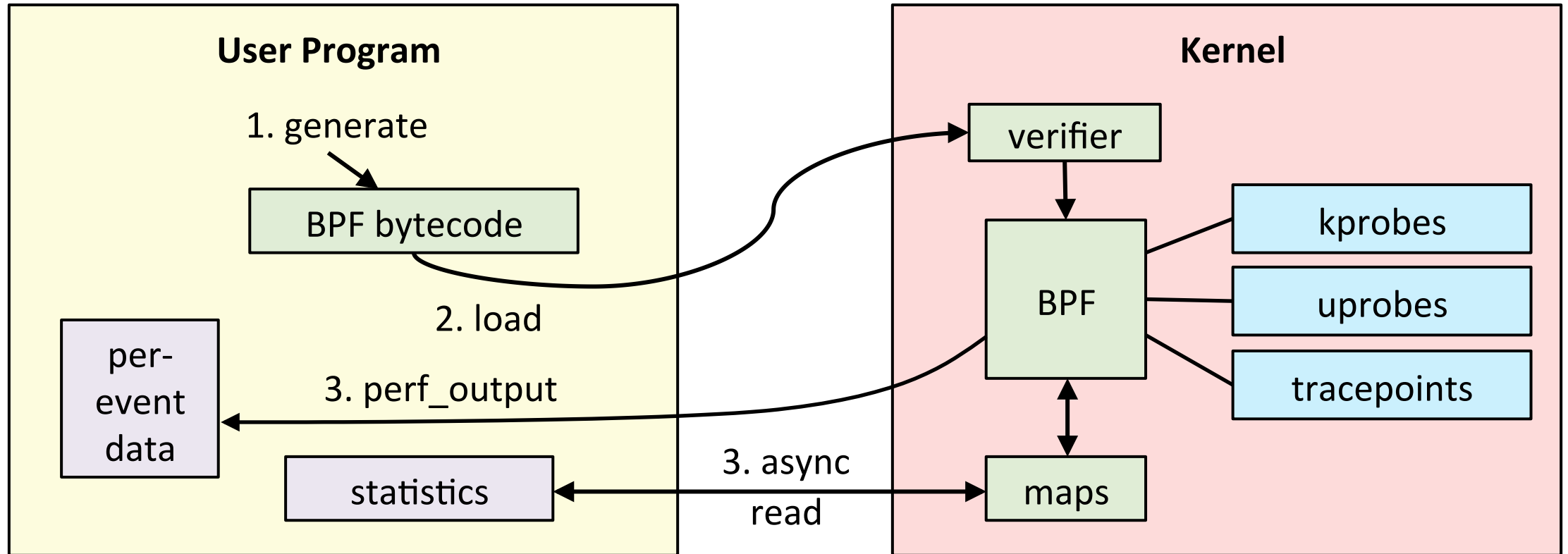
BPF

- aka eBPF == enhanced Berkeley Packet Filter
 - Lead developer: Alexei Starovoitov (Facebook)
- Many uses
 - Virtual networking
 - Security
 - Programmatic tracing
- Different front-ends
 - C, perf, bcc, ply, ...



BPF mascot

BPF for Tracing



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(),
};
```


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 loopback device and prints them
     */
    char devname[IFNAMSIZ];
    struct net_device *dev;
    struct sk_buff *skb;
    int len;

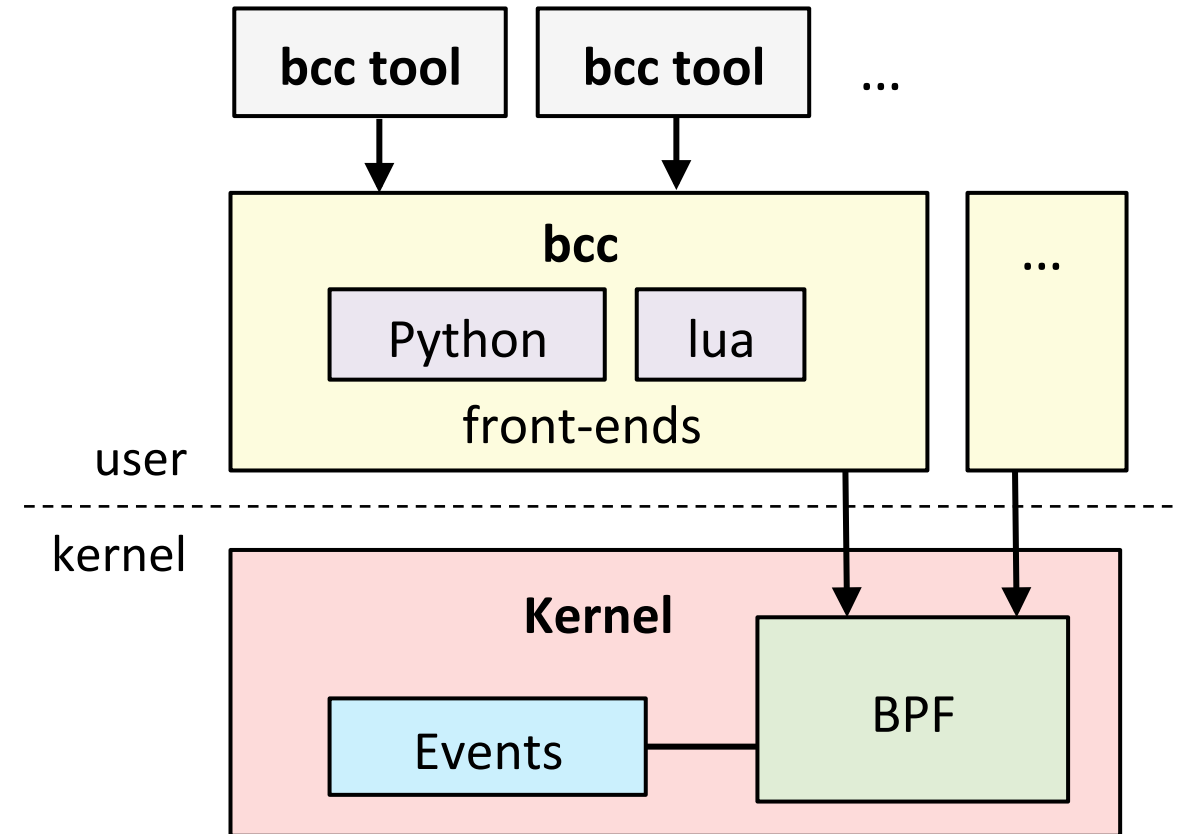
    /* non-portable! works for the given kernel only */
    skb = (struct sk_buff *) PT_REGS_PARM1(ctx);
    dev = _(skb->dev);
```

bcc



- BPF Compiler Collection
 - <https://github.com/iovisor/bcc>
 - Lead developer: Brenden Blanco (PlumGRID)
- Includes tracing tools
- Front-ends
 - Python
 - Lua
 - C helper libraries

Tracing layers:



bcc/BPF

```
# 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")
```

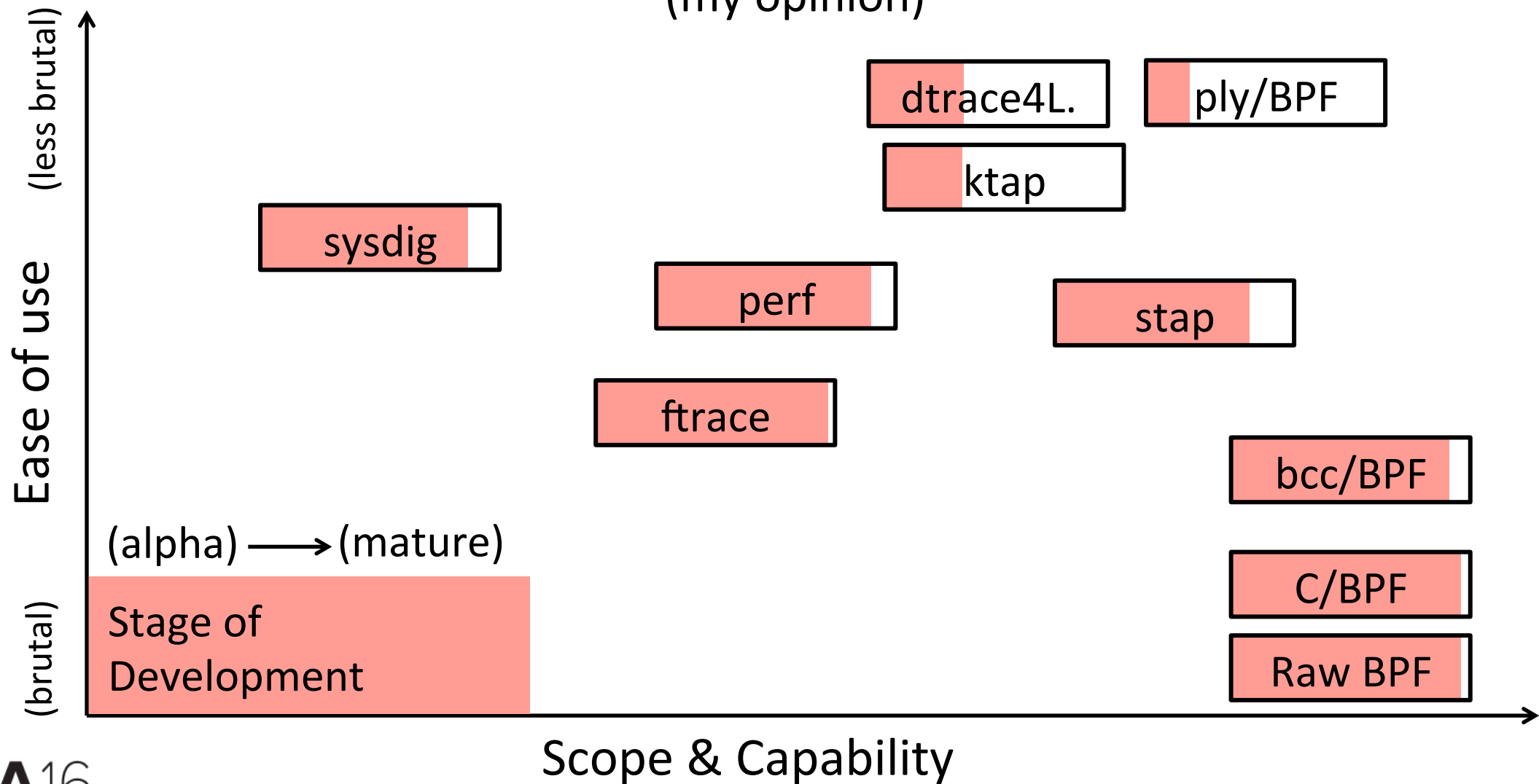
ply/BPF

```
#!/usr/bin/env ply

kprobe:Sys_read
{
    $sizes.quantize(arg(2))
}
```

The Tracing Landscape, Dec 2016

(my opinion)



State of BPF, Dec 2016

1. Dynamic tracing, kernel-level (BPF support for kprobes)
2. Dynamic tracing, user-level (BPF support for uprobes)
3. Static tracing, kernel-level (BPF support for tracepoints)
4. Timed sampling events (BPF with `perf_event_open`)
5. PMC events (BPF with `perf_event_open`)
6. Filtering (via BPF programs)
7. Debug output (`bpf_trace_printk()`)
8. Per-event output (`bpf_perf_event_output()`)
9. Basic variables (global & per-thread variables, via BPF maps)
10. Associative arrays (via BPF maps)
11. Frequency counting (via BPF maps)
12. Histograms (power-of-2, linear, and custom, via BPF maps)
13. Timestamps and time deltas (`bpf_ktime_get_()` and BPF)
14. Stack traces, kernel (BPF stackmap)
15. Stack traces, user (BPF stackmap)
16. Overwrite ring buffers
17. String factory (`stringmap`)
18. Optional: bounded loops, `<` and `<=`, ...

State of bcc, Dec 2016

1. Static tracing, user-level (USDT probes via uprobes)
2. Static tracing, dynamic USDT (needs library support)
3. Debug output (Python with `BPF.trace_pipe()` and `BPF.trace_fields()`)
4. Per-event output (`BPF_PERF_OUTPUT` macro and `BPF.open_perf_buffer()`)
5. Interval output (`BPF.get_table()` and `table.clear()`)
6. Histogram printing (`table.print_log2_hist()`)
7. C struct navigation, kernel-level (maps to `bpf_probe_read()`)
8. Symbol resolution, kernel-level (`ksym()`, `ksymaddr()`)
9. Symbol resolution, user-level (`usymaddr()`)
10. BPF tracepoint support (via `TRACEPOINT_PROBE`)
11. BPF stack trace support (incl. `walk` method for stack frames)
12. Examples (under `/examples`)
13. Many tools (`/tools`)
14. Tutorials (`/docs/tutorial*.md`)
15. Reference guide (`/docs/reference_guide.md`)
16. Open issues: (<https://github.com/iovisor/bcc/issues>)

For end-users

HOW TO USE BCC/BPF

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
```

- puts tools in /usr/share/bcc/tools, and tools/old for older kernels
- 16.04 is good, 16.10 better: more tools work
- bcc should also arrive as an official Ubuntu snap

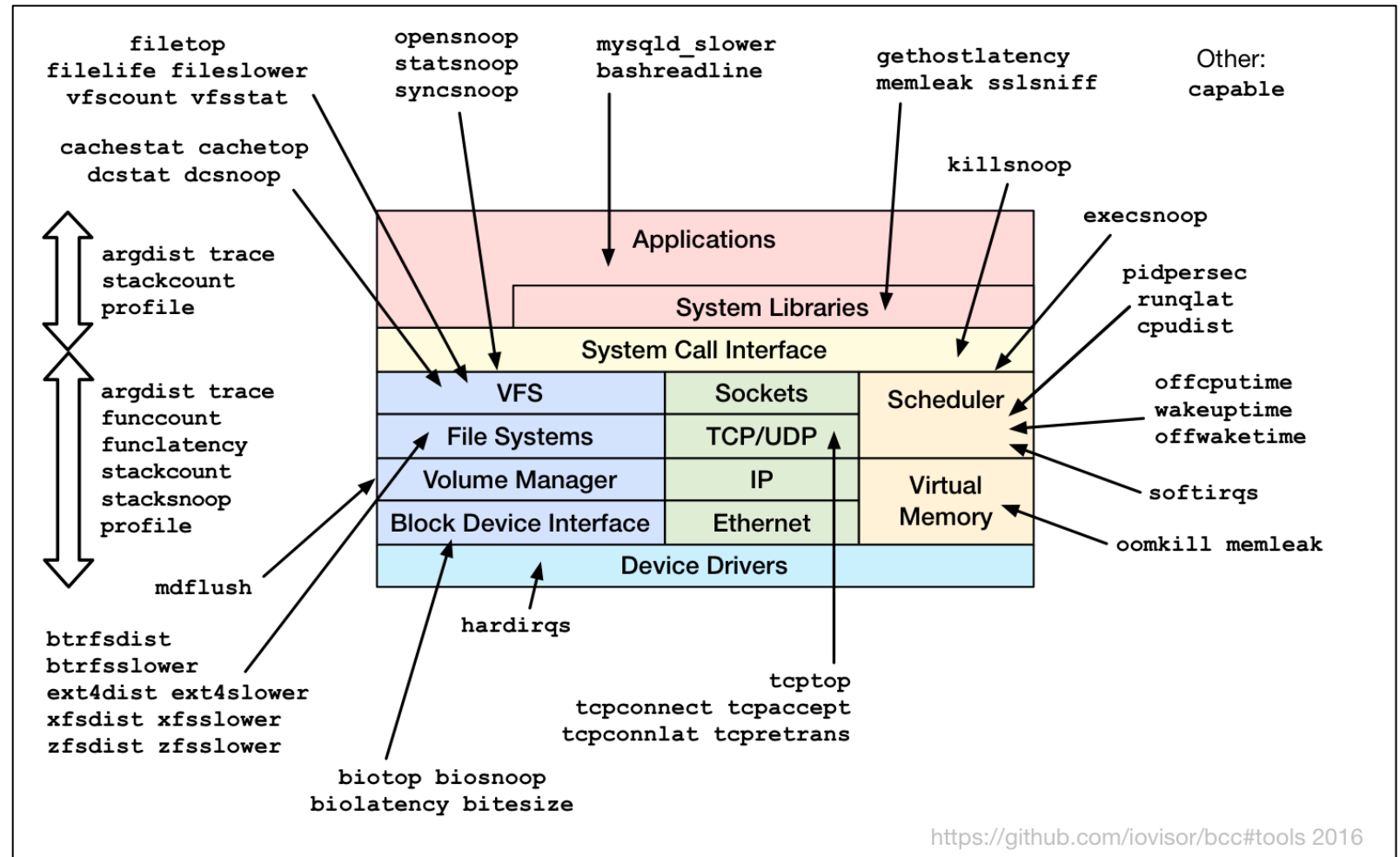
Pre-bcc Performance Checklist

1. `uptime`
2. `dmesg | 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 General Performance Checklist

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



<https://github.com/iovisor/bcc#tools> 2016

1. execsnoop

```
# execsnoop
PCOMM          PID    RET  ARGS
bash           15887    0  /usr/bin/man ls
preconv        15894    0  /usr/bin/preconv -e UTF-8
man            15896    0  /usr/bin/tbl
man            15897    0  /usr/bin/nroff -mandoc -rLL=169n -rLT=169n -Tutf8
man            15898    0  /usr/bin/pager -s
nroff          15900    0  /usr/bin/locale charmap
nroff          15901    0  /usr/bin/groff -mtty-char -Tutf8 -mandoc -rLL=169n -rLT=169n
groff          15902    0  /usr/bin/troff -mtty-char -mandoc -rLL=169n -rLT=169n -Tutf8
groff          15903    0  /usr/bin/grotty
[...]
```

2. opensnoop

```
# opensnoop
PID      COMM          FD  ERR  PATH
27159    catalina.sh   3   0    /apps/tomcat8/bin/setclasspath.sh
4057     redis-server  5   0    /proc/4057/stat
2360     redis-server  5   0    /proc/2360/stat
30668    sshd          4   0    /proc/sys/kernel/ngroups_max
30668    sshd          4   0    /etc/group
30668    sshd          4   0    /root/.ssh/authorized_keys
30668    sshd          4   0    /root/.ssh/authorized_keys
30668    sshd          -1  2    /var/run/nologin
30668    sshd          -1  2    /etc/nologin
30668    sshd          4   0    /etc/login.defs
30668    sshd          4   0    /etc/passwd
30668    sshd          4   0    /etc/shadow
30668    sshd          4   0    /etc/localtime
4510     snmp-pass     4   0    /proc/cpuinfo
[...]
```

3. ext4slower

```
# ext4slower 1
Tracing ext4 operations slower than 1 ms
TIME      COMM          PID    T BYTES  OFF_KB  LAT(ms)  FILENAME
06:49:17  bash          3616   R 128    0        7.75    cksum
06:49:17  cksum        3616   R 39552  0        1.34    [
06:49:17  cksum        3616   R 96     0        5.36    2to3-2.7
06:49:17  cksum        3616   R 96     0       14.94    2to3-3.4
06:49:17  cksum        3616   R 10320  0        6.82    411toppm
06:49:17  cksum        3616   R 65536  0        4.01    a2p
06:49:17  cksum        3616   R 55400  0        8.77    ab
06:49:17  cksum        3616   R 36792  0       16.34    aclocal-1.14
06:49:17  cksum        3616   R 15008  0       19.31    acpi_listen
06:49:17  cksum        3616   R 6123   0       17.23    add-apt-repository
06:49:17  cksum        3616   R 6280   0       18.40    addpart
06:49:17  cksum        3616   R 27696  0        2.16    addr2line
06:49:17  cksum        3616   R 58080  0       10.11    ag
[...]
```

4. biolateness

```
# biolateness -mT 1
Tracing block device I/O... Hit Ctrl-C to end.
```

```
06:20:16
```

| msecs | : | count | distribution |
|-----------|---|-------|--------------|
| 0 -> 1 | : | 36 | ***** |
| 2 -> 3 | : | 1 | * |
| 4 -> 7 | : | 3 | *** |
| 8 -> 15 | : | 17 | ***** |
| 16 -> 31 | : | 33 | ***** |
| 32 -> 63 | : | 7 | ***** |
| 64 -> 127 | : | 6 | ***** |

```
[...]
```

5. biosnoop

```
# biosnoop
TIME(s)      COMM          PID    DISK    T  SECTOR    BYTES    LAT(ms)
0.000004001  supervise    1950   xvda1  W  13092560  4096     0.74
0.000178002  supervise    1950   xvda1  W  13092432  4096     0.61
0.001469001  supervise    1956   xvda1  W  13092440  4096     1.24
0.001588002  supervise    1956   xvda1  W  13115128  4096     1.09
1.022346001  supervise    1950   xvda1  W  13115272  4096     0.98
1.022568002  supervise    1950   xvda1  W  13188496  4096     0.93
1.023534000  supervise    1956   xvda1  W  13188520  4096     0.79
1.023585003  supervise    1956   xvda1  W  13189512  4096     0.60
2.003920000  xfsaild/md0  456    xvdc   W  62901512  8192     0.23
2.003931001  xfsaild/md0  456    xvdb   W  62901513  512      0.25
2.004034001  xfsaild/md0  456    xvdb   W  62901520  8192     0.35
2.004042000  xfsaild/md0  456    xvdb   W  63542016  4096     0.36
2.004204001  kworker/0:3  26040  xvdb   W  41950344  65536    0.34
2.044352002  supervise    1950   xvda1  W  13192672  4096     0.65
[...]
```

6. cachestat

```
# cachestat
```

| HITS | MISSES | DIRTIES | READ_HIT% | WRITE_HIT% | BUFFERS_MB | CACHED_MB |
|--------|--------|---------|-----------|------------|------------|-----------|
| 170610 | 41607 | 33 | 80.4% | 19.6% | 11 | 288 |
| 157693 | 6149 | 33 | 96.2% | 3.7% | 11 | 311 |
| 174483 | 20166 | 26 | 89.6% | 10.4% | 12 | 389 |
| 434778 | 35 | 40 | 100.0% | 0.0% | 12 | 389 |
| 435723 | 28 | 36 | 100.0% | 0.0% | 12 | 389 |
| 846183 | 83800 | 332534 | 55.2% | 4.5% | 13 | 553 |
| 96387 | 21 | 24 | 100.0% | 0.0% | 13 | 553 |
| 120258 | 29 | 44 | 99.9% | 0.0% | 13 | 553 |
| 255861 | 24 | 33 | 100.0% | 0.0% | 13 | 553 |
| 191388 | 22 | 32 | 100.0% | 0.0% | 13 | 553 |

```
[...]
```


7. tcpconnect

```
# tcpconnect
PID      COMM          IP  SADDR          DADDR          DPORT
25333    recordProgra  4   127.0.0.1      127.0.0.1      28527
25338    curl          4   100.66.3.172   52.22.109.254  80
25340    curl          4   100.66.3.172   31.13.73.36    80
25342    curl          4   100.66.3.172   104.20.25.153  80
25344    curl          4   100.66.3.172   50.56.53.173   80
25365    recordProgra  4   127.0.0.1      127.0.0.1      28527
26119    ssh           6   ::1             ::1             22
25388    recordProgra  4   127.0.0.1      127.0.0.1      28527
25220    ssh           6   fe80::8a3:9dff:fed5:6b19 fe80::8a3:9dff:fed5:6b19 22
[...]
```

8. tcpaccept

```
# tcpaccept
PID      COMM          IP  RADDR          LADDR          LPORT
2287     sshd          4   11.16.213.254  100.66.3.172   22
4057     redis-server 4   127.0.0.1      127.0.0.1      28527
4057     redis-server 4   127.0.0.1      127.0.0.1      28527
4057     redis-server 4   127.0.0.1      127.0.0.1      28527
4057     redis-server 4   127.0.0.1      127.0.0.1      28527
2287     sshd          6   ::1            ::1            22
4057     redis-server 4   127.0.0.1      127.0.0.1      28527
4057     redis-server 4   127.0.0.1      127.0.0.1      28527
2287     sshd          6   fe80::8a3:9dff:fed5:6b19 fe80::8a3:9dff:fed5:6b19 22
4057     redis-server 4   127.0.0.1      127.0.0.1      28527
[...]
```

9. tcpretrans

```
# tcpretrans
TIME      PID      IP  LADDR:LPORT      T>  RADDR:RPORT      STATE
01:55:05  0        4   10.153.223.157:22  R>  69.53.245.40:34619 ESTABLISHED
01:55:05  0        4   10.153.223.157:22  R>  69.53.245.40:34619 ESTABLISHED
01:55:17  0        4   10.153.223.157:22  R>  69.53.245.40:22957 ESTABLISHED
[...]
```

10. gethostlatency

```
# gethostlatency
TIME          PID      COMM          LATms  HOST
06:10:24     28011   wget          90.00  www.iovisor.org
06:10:28     28127   wget           0.00  www.iovisor.org
06:10:41     28404   wget           9.00  www.netflix.com
06:10:48     28544   curl          35.00  www.netflix.com.au
06:11:10     29054   curl          31.00  www.plumgrid.com
06:11:16     29195   curl           3.00  www.facebook.com
06:11:24     25313   wget           3.00  www.usenix.org
06:11:25     29404   curl          72.00  foo
06:11:28     29475   curl           1.00  foo
[...]
```

11. runqlat

```
# runqlat -m 5
Tracing run queue latency... Hit Ctrl-C to end.

msecs           : count      distribution
  0 -> 1         : 3818      |*****|
  2 -> 3         : 39        |        |
  4 -> 7         : 39        |        |
  8 -> 15        : 62        |        |
 16 -> 31        : 2214     |*****|
 32 -> 63        : 226      |**      |
```

[...]

12. profile

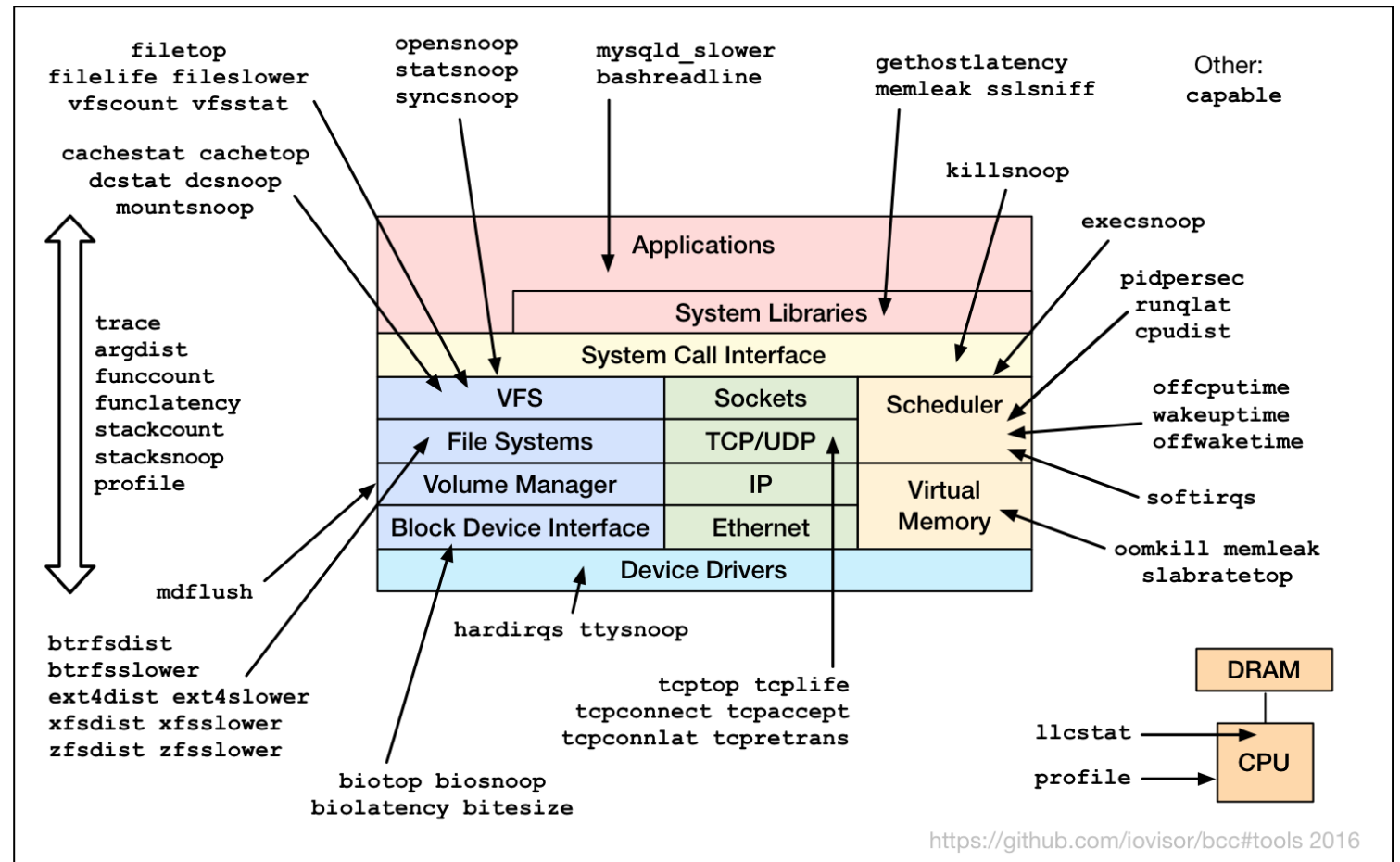
```
# profile
Sampling at 49 Hertz of all threads by user + kernel stack... Hit Ctrl-C to end.
^C
[...]

ffffff813d0af8 __clear_user
ffffff813d5277 iov_iter_zero
ffffff814ec5f2 read_iter_zero
ffffff8120be9d __vfs_read
ffffff8120c385 vfs_read
ffffff8120d786 sys_read
ffffff817cc076 entry_SYSCALL_64_fastpath
00007fc5652ad9b0 read
- dd (25036)
7
[...]

```

Other bcc Tracing Tools

- Single-purpose
 - bitesize
 - capable
 - memleak
 - ext4dist (btrfs, ...)
- Multi tools
 - funccount
 - argdist
 - trace
 - stackcount



trace

- Trace custom events. Ad hoc analysis:

```
# 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  
05:18:23 4490     dd        sys_read  read 1048576 bytes  
^C
```


trace One-Liners

```
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

trace 'p:c:write (arg1 == 1) "writing %d bytes to STDOUT", arg3'
    Trace the write() call from libc to monitor writes to STDOUT

trace 'r:c:malloc (retval) "allocated = %p", retval'
    Trace returns from malloc and print non-NULL allocated buffers

trace 't:block:block_rq_complete "sectors=%d", args->nr_sector'
    Trace the block_rq_complete kernel tracepoint and print # of tx sectors

trace 'u:pthread:pthread_create (arg4 != 0)'
    Trace the USDT probe pthread_create when its 4th argument is non-zero
```

argdist

```
# 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 | |
| 32768 -> 65535 | : 10086 | ***** |
| 65536 -> 131071 | : 60 | |
| 131072 -> 262143 | : 17285 | ***** |

```
[...]
```

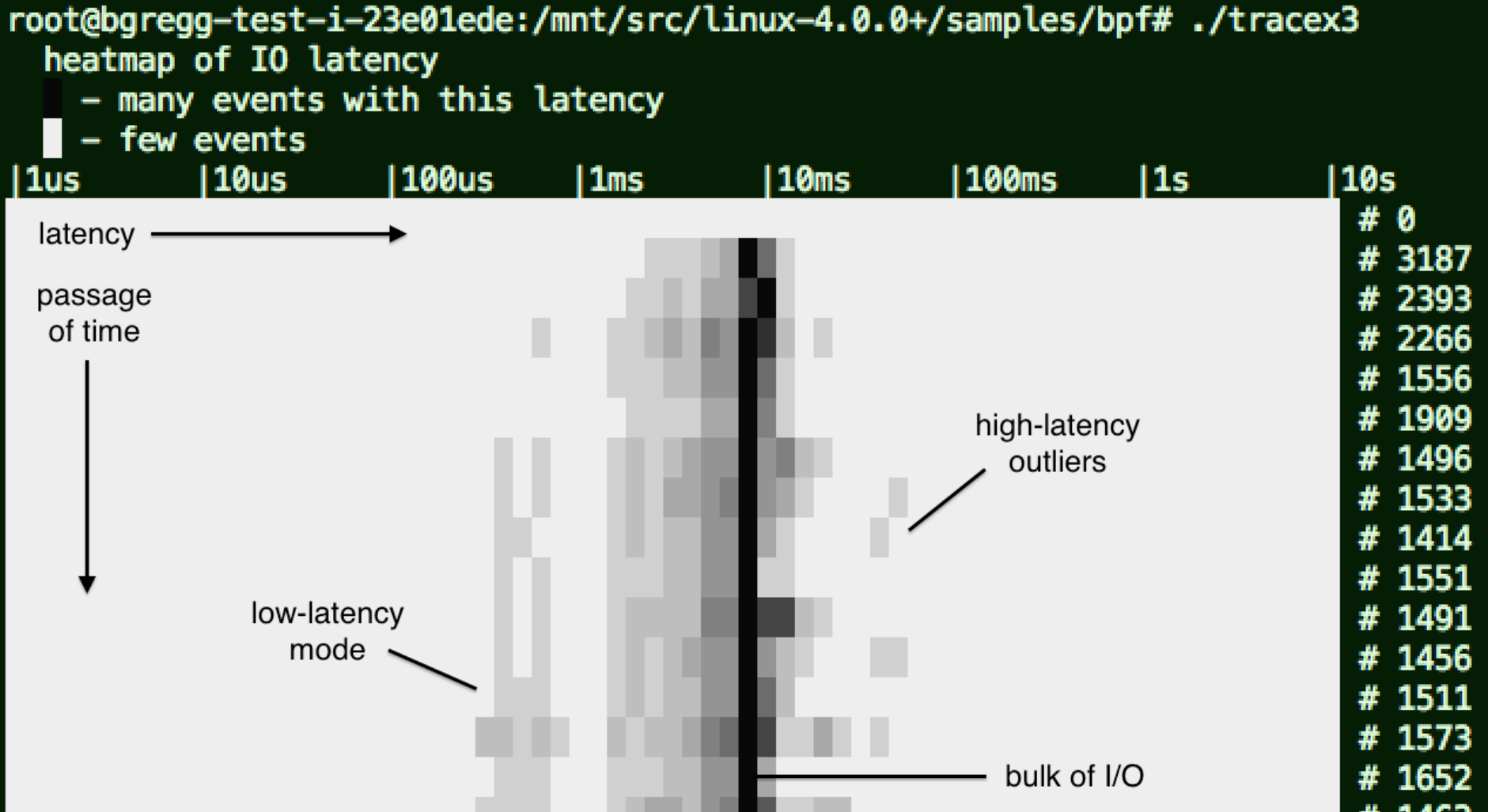
argdist One-Liners

```
argdist -H 'p::__kmalloc(u64 size):u64:size'  
Print a histogram of allocation sizes passed to kmalloc  
  
argdist -p 1005 -C 'p:c:malloc(size_t size):size_t:size:size==16'  
Print a frequency count of how many times process 1005 called malloc for 16 bytes  
  
argdist -C 'r:c:gets():char*:$retval#snooped strings'  
Snoop on all strings returned by gets()  
  
argdist -H 'r::__kmalloc(size_t size):u64:$latency/$entry(size)#ns per byte'  
Print a histogram of nanoseconds per byte from kmalloc allocations  
  
argdist -C 'p::__kmalloc(size_t size, gfp_t flags):size_t:size:flags&GFP_ATOMIC'  
Print frequency count of kmalloc allocation sizes that have GFP_ATOMIC  
  
argdist -p 1005 -C 'p:c:write(int fd):int:fd' -T 5  
Print frequency counts of how many times writes were issued to a particular file descriptor  
number, in process 1005, but only show the top 5 busiest fds  
  
argdist -p 1005 -H 'r:c:read()'  
Print a histogram of error codes returned by read() in process 1005  
  
argdist -C 'r::__vfs_read():u32:$PID:$latency > 100000'  
Print frequency of reads by process where the latency was >0.1ms
```

Coming to a GUI near you

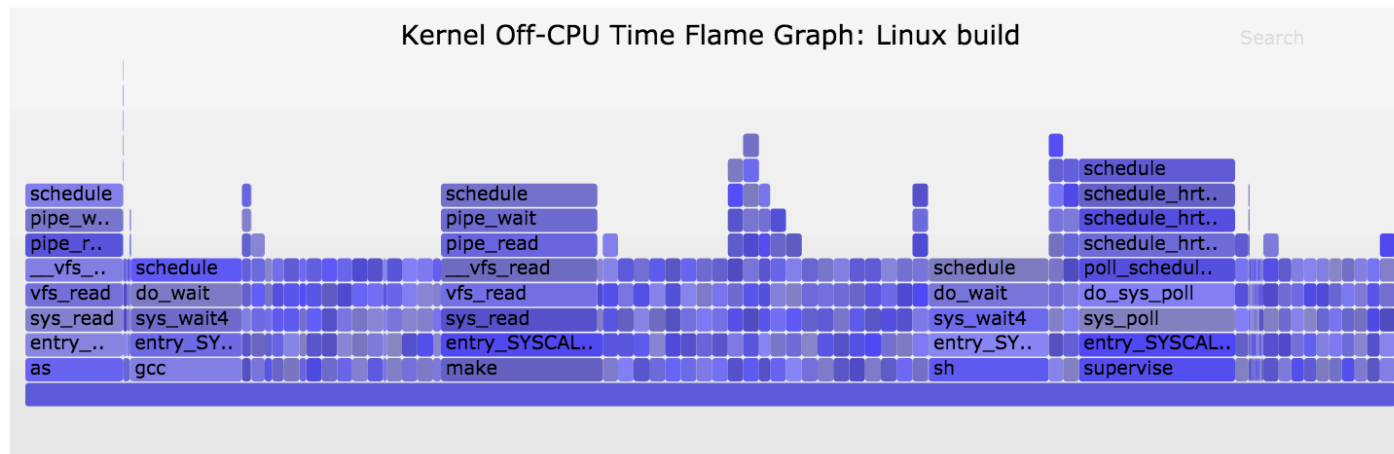
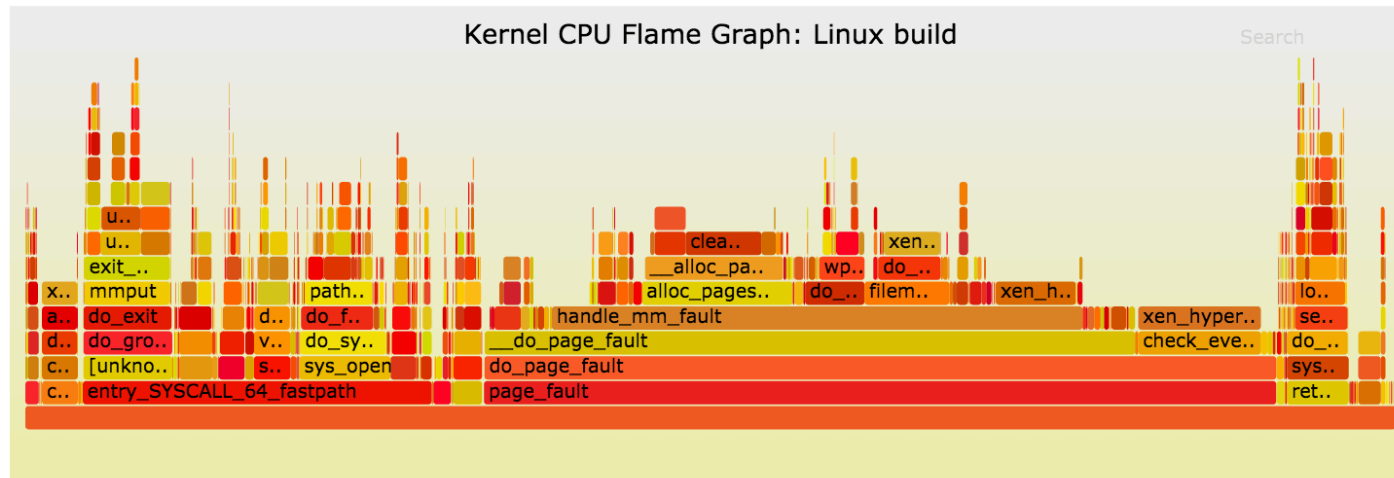
BCC/BPF VISUALIZATIONS

Latency Heatmaps



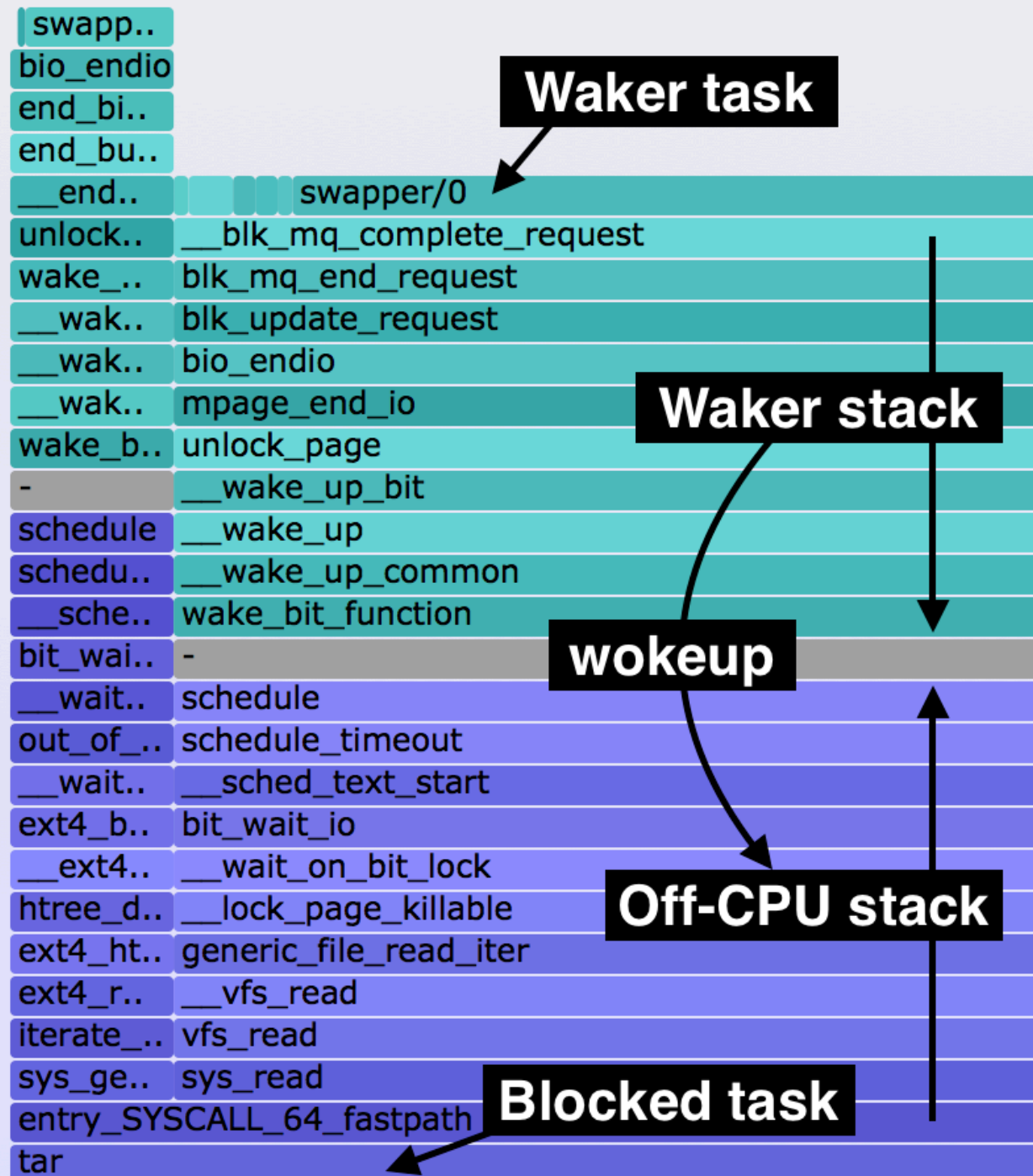
CPU + Off-CPU Flame Graphs

- Can now be BPF optimized



Off-Wake Flame Graphs

- Shows blocking stack with waker stack
 - Better understand why blocked
 - Merged in-kernel using BPF
 - Include multiple waker stacks == chain graphs
- We couldn't do this before



Overview for tool developers

HOW TO PROGRAM BCC/BPF

Linux Event Sources

BPF output
Linux 4.4

BPF stacks
Linux 4.6

uprobes
Linux 4.3

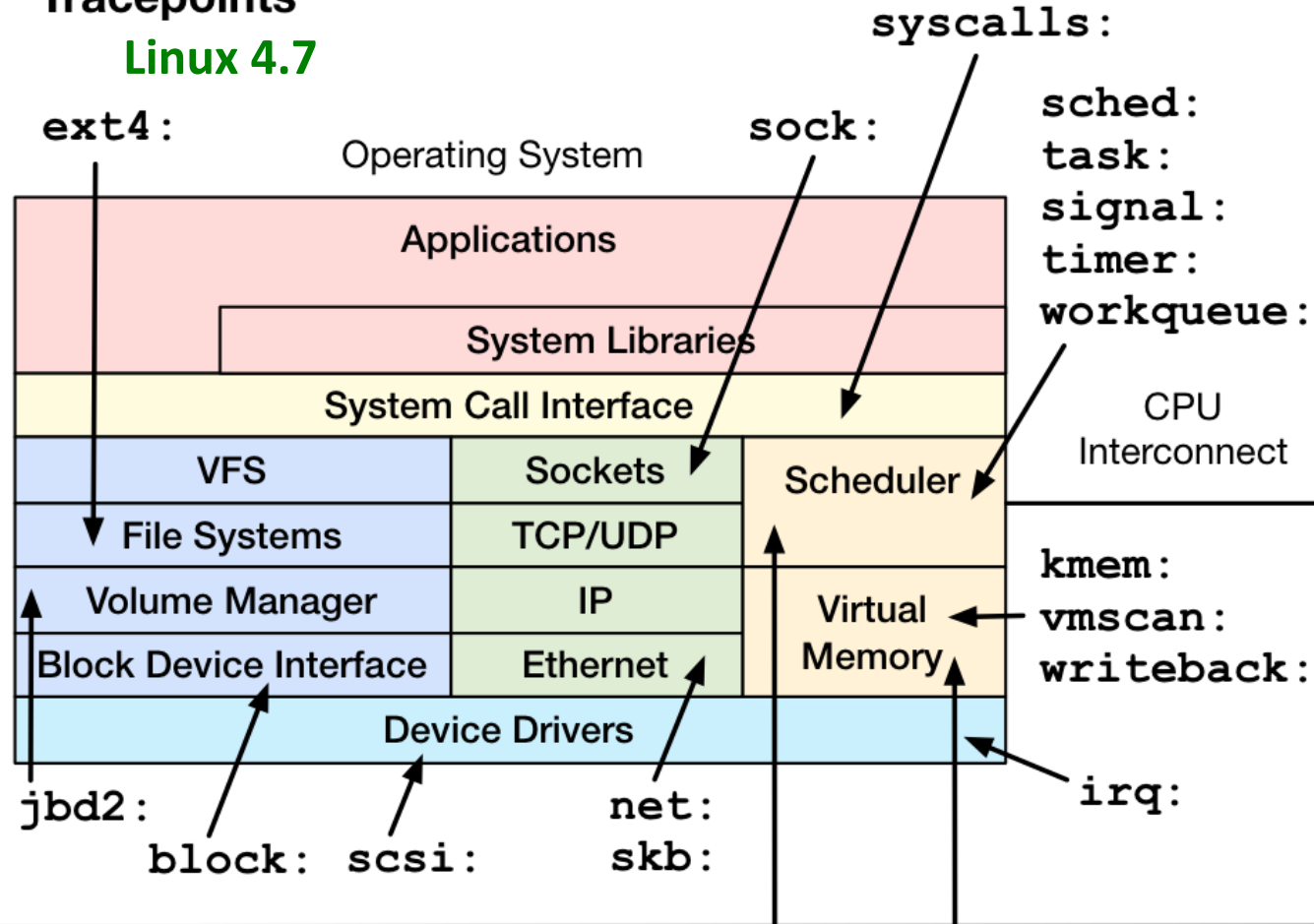
kprobes
Linux 4.1

(version
feature
arrived)

Dynamic
Tracing

Tracepoints
Linux 4.7

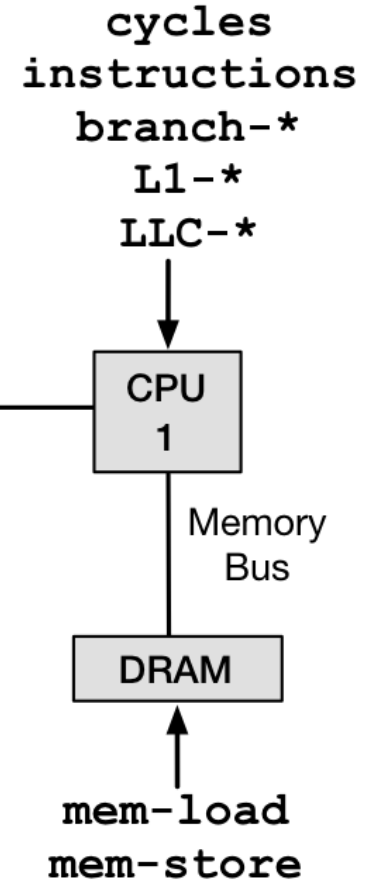
PMCs
Linux 4.9



Software Events
Linux 4.9

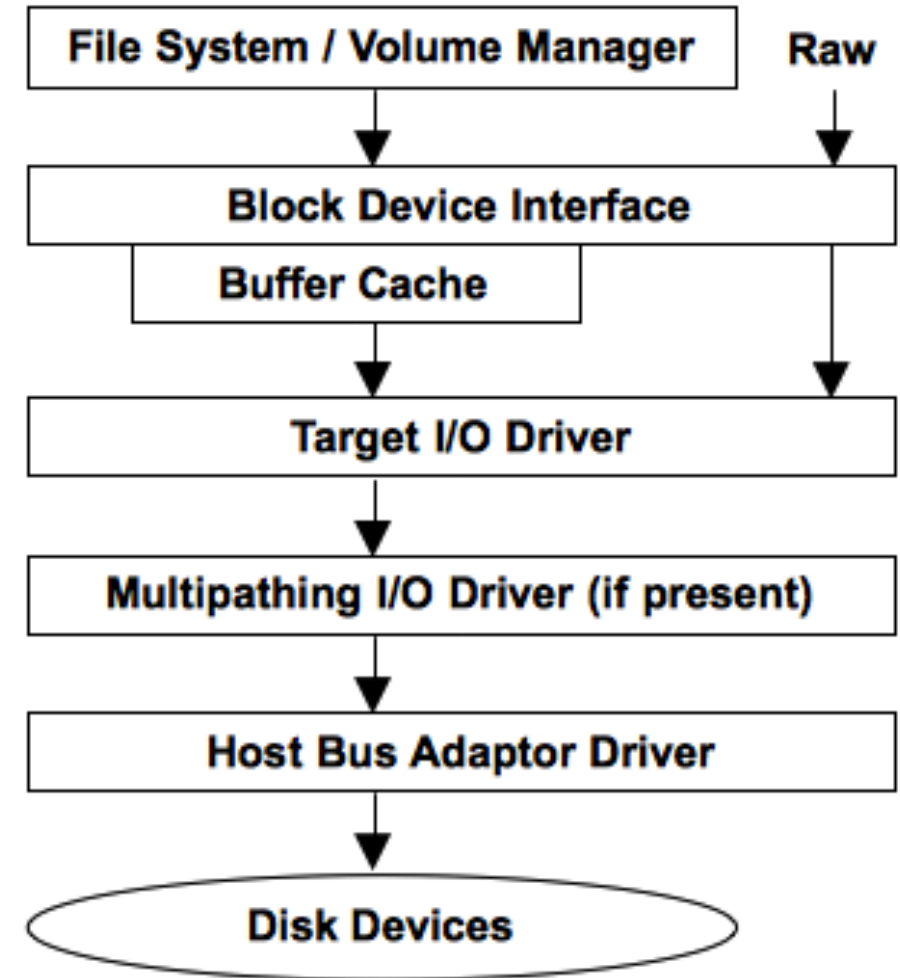
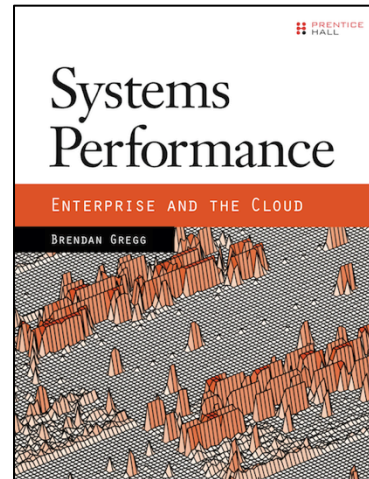
cpu-clock
cs migrations

page-faults
minor-faults
major-faults



Methodology

- Find/draw a functional diagram
 - Eg, storage I/O subsystem:
- 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



bitehist.py Output

```
# ./bitehist.py
Tracing... Hit Ctrl-C to end.
^C
```

| kbytes | : | count | distribution |
|------------|---|-------|--------------|
| 0 -> 1 | : | 3 | |
| 2 -> 3 | : | 0 | |
| 4 -> 7 | : | 211 | ***** |
| 8 -> 15 | : | 0 | |
| 16 -> 31 | : | 0 | |
| 32 -> 63 | : | 0 | |
| 64 -> 127 | : | 1 | |
| 128 -> 255 | : | 800 | ***** |

bitehist.py Code

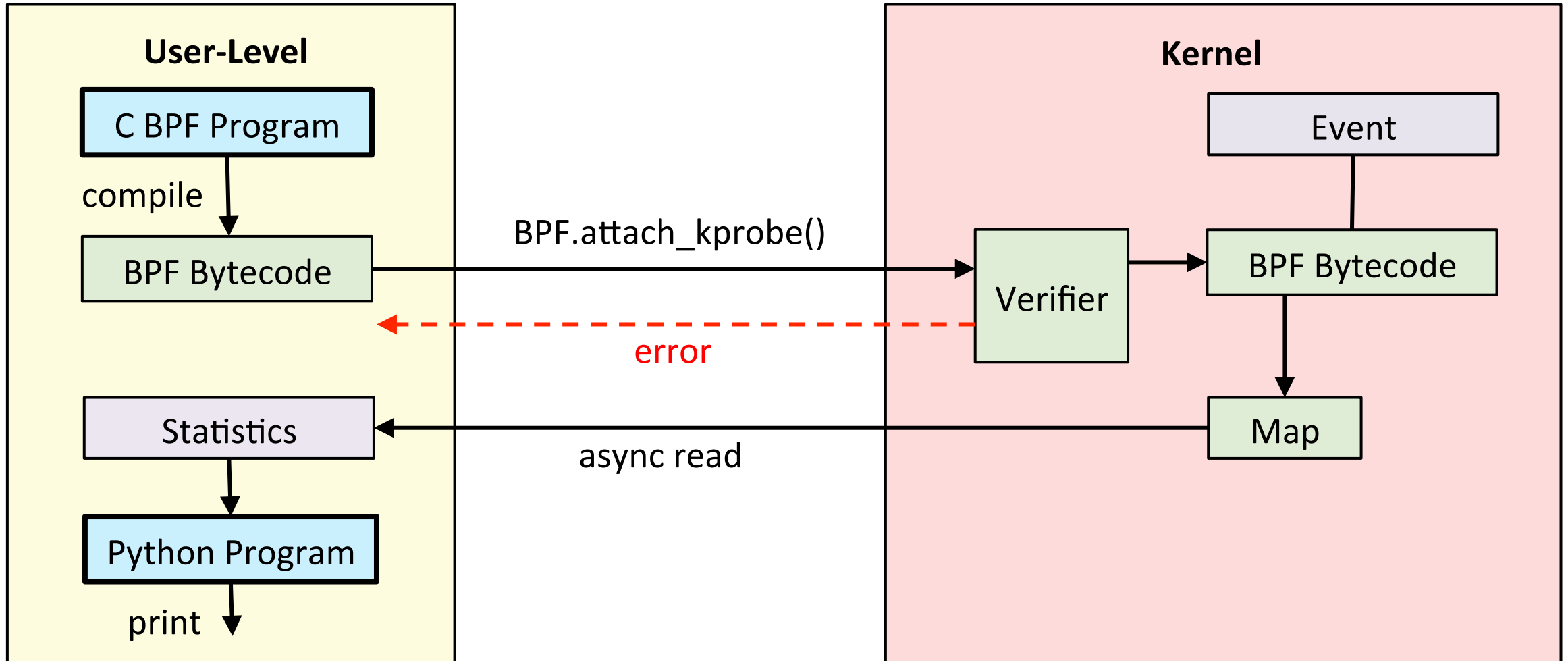
```
# load BPF program
b = BPF(text="""
#include <uapi/linux/ptrace.h>
#include <linux/blkdev.h>
BPF_HISTOGRAM(dist);
int kprobe__blk_account_io_completion(struct pt_regs *ctx,
    struct request *req)
{
    dist.increment(bpf_log2l(req->__data_len / 1024));
    return 0;
}
""")
```

```
# 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")
```

bytehist.py Internals



bytehist.py Annotated

```
# 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;
}
""")
```

Annotations:

- Map**: Points to `BPF_HISTOGRAM(dist);`
- C BPF Program**: Points to the `int kprobe__blk_account_io_completion` function.
- Event**: Points to the `struct request *req` parameter.
- Shortcut**: A box containing the text `"kprobe__" is a shortcut for BPF.attach_kprobe()` points to the `kprobe__` prefix in the function name.

```
# 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")
```

Annotations:

- Python Program**: Points to the `try:` block.
- Statistics**: Points to the `b["dist"].print_log2_hist("kbytes")` line.

Current Complications

- 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));
```

bcc Tutorials

1. <https://github.com/iovisor/bcc/blob/master/INSTALL.md>
2. <.../docs/tutorial.md>
3. .../docs/tutorial_bcc_python_developer.md
4. .../docs/reference_guide.md
5. <.../CONTRIBUTING-SCRIPTS.md>

bcc lua

```
local program = string.gsub([[
#include <uapi/linux/ptrace.h>
int printarg(struct pt_regs *ctx) {
    if (!PT_REGS_PARM1(ctx))
        return 0;
    u32 pid = bpf_get_current_pid_tgid();
    if (pid != PID)
        return 0;
    char str[128] = {};
    bpf_probe_read(&str, sizeof(str), (void *)PT_REGS_PARM1(ctx));
    bpf_trace_printk("strlen(\"%s\")\n", &str);
    return 0;
}
]], "PID", arg[1])
```

```
return function(BPF)
    local b = BPF:new{text=program, debug=0}
    b:attach_uprobe{name="c", sym="strlen", fn_name="printarg"}

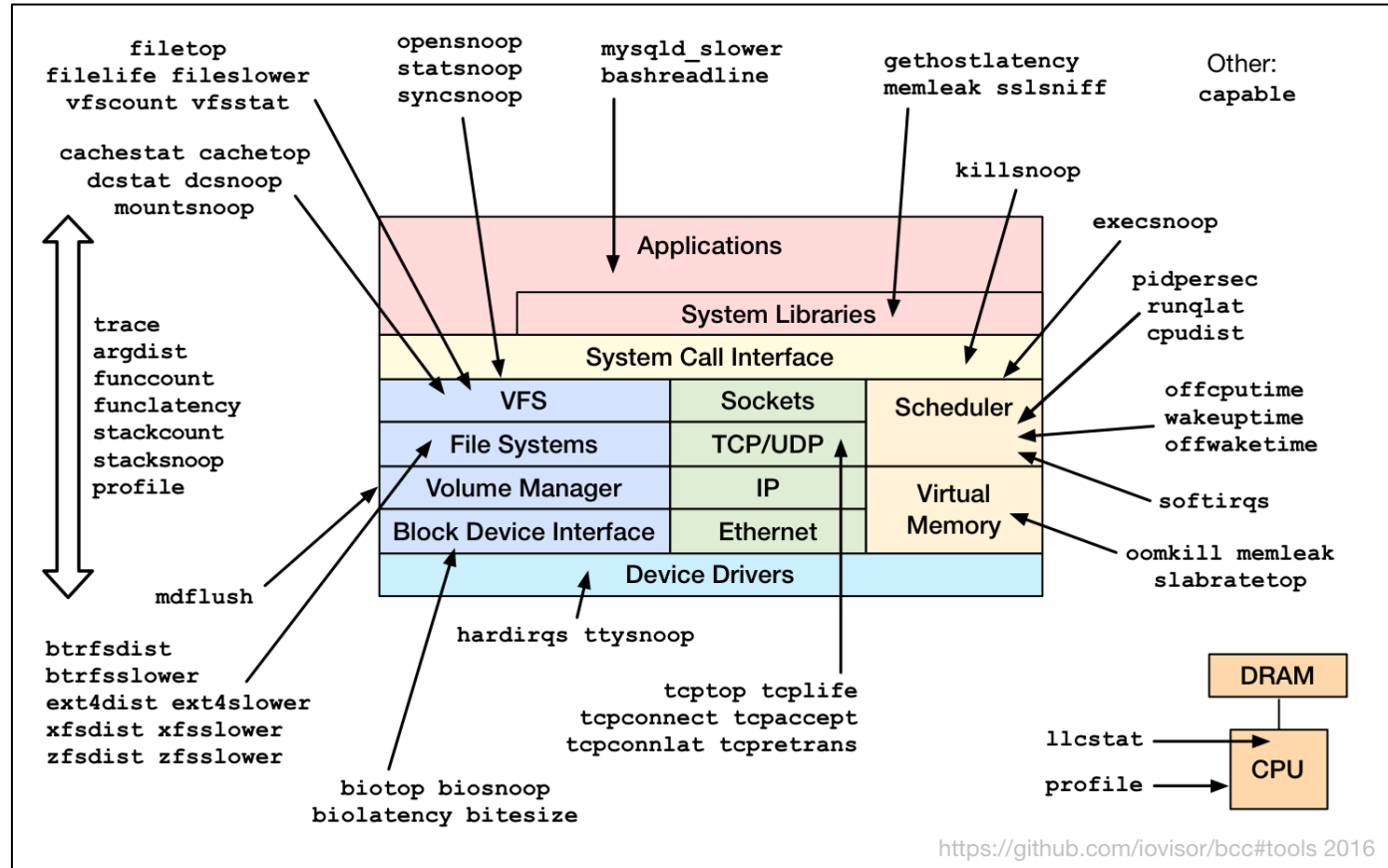
    local pipe = b:pipe()
    while true do
        local task, pid, cpu, flags, ts, msg = pipe:trace_fields()
        print("%-18.9f %-16s %-6d %s" % {ts, task, pid, msg})
    end
end
```

[bcc examples/lua/strlen_count.lua](#)

Summary

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



Future Work

- More tooling
- Bug fixes
- Better errors
- Visualizations
- GUIs
- High-level language

<https://github.com/iovisor/bcc#tools>

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://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>
- 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>
- Dynamic Instrumentation:
 - <http://ftp.cs.wisc.edu/par-distr-sys/papers/Hollingsworth94Dynamic.pdf>
 - <https://en.wikipedia.org/wiki/DTrace>
 - DTrace: Dynamic Tracing in Oracle Solaris, Mac OS X and FreeBSD, Brendan Gregg, Jim Mauro; Prentice Hall 2011
- Netflix Tech Blog on Vector:
 - <http://techblog.netflix.com/2015/04/introducing-vector-netflixs-on-host.html>
- Greg Law's GDB talk: <https://www.youtube.com/watch?v=PorfLSr3DDI>
- Linux Performance: <http://www.brendangregg.com/linuxperf.html>

Thanks

- 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), Sasha Goldshtein (Sela), Daniel Borkmann (Cisco), Wang Nan (Huawei), and other BPF and bcc contributors!

