Linux Performance Analysis Using perf and BPF

Brendan Gregg
Senior Performance Architect
NETFLIX

Sasha Goldshtein
CTO
SELA GROUP

Agenda

- The modern Linux tracing landscape
- perf
- Flame graphs
- Lab: CPU profiling with perf
- BPF
- BCC – BPF Compiler Collection
- Lab: BCC one-liners
- BCC data types and program structure
- Lab: Authoring BCC tools

Learning Objectives

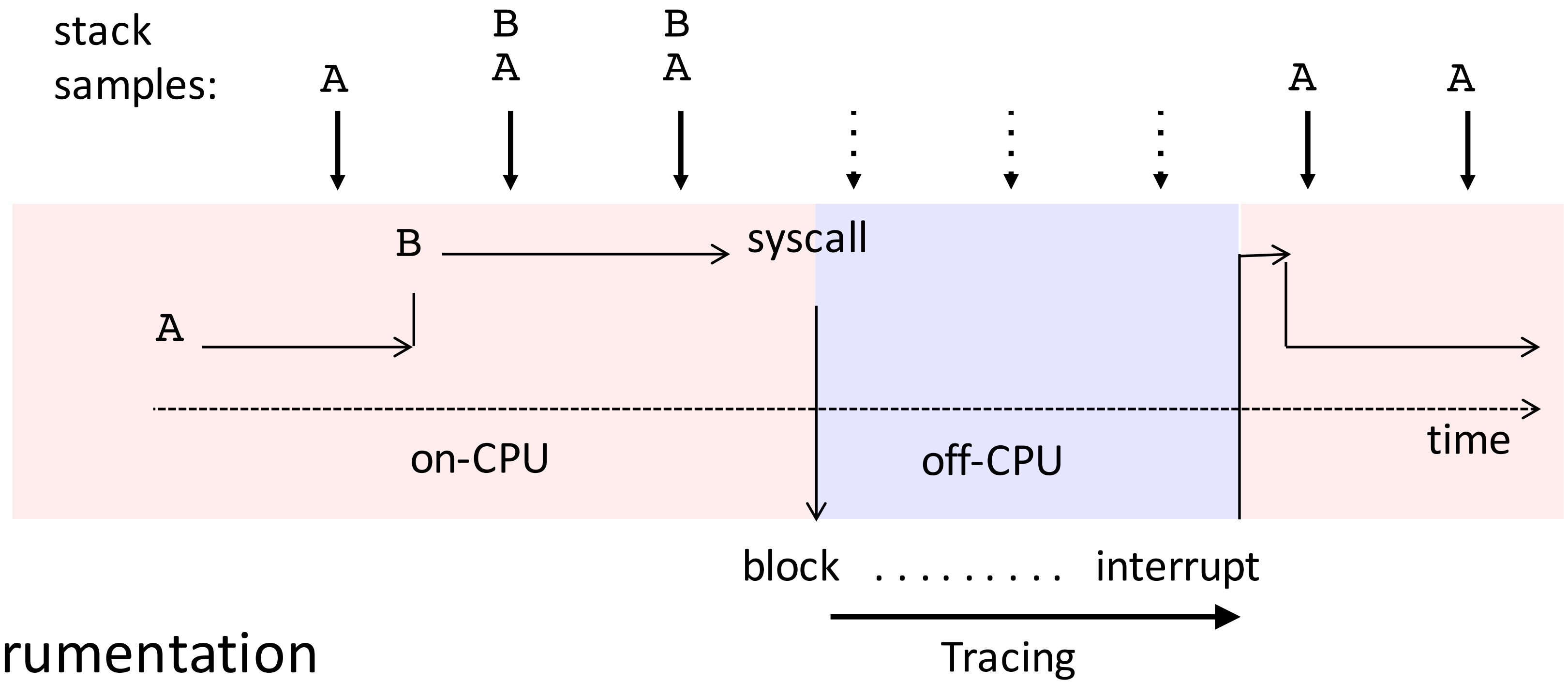
- Understand flame graphs and how to interpret them
- Perform commands using Linux perf to create a CPU flame graph
- Understand the role of BPF and Linux tracing
- Gain experience with installing and using bcc/BPF tools
- Apply methodology for analyzing system performance
- Identify bcc/BPF tool source code components
- Make simple customizations to a bcc/BPF tool
- Identify reference documentation for bcc development
- Optionally develop a from-scratch bcc/BPF tool

Prerequisites

- You should …
  - Have experience developing on or administering a Linux deployment
  - Be familiar with C/Python/Lua (a bonus)
- You can use the instructor-provided Strigo workspace (EC2)
  - Classroom link and token will be provided at the workshop
- To use your own machines for this workshop … (not recommended)
  - You will need Linux 4.6+
  - Clone or install some open source tools (perf, bcc)
- Instructions and labs:
  https://github.com/goldshtn/linux-tracing-workshop

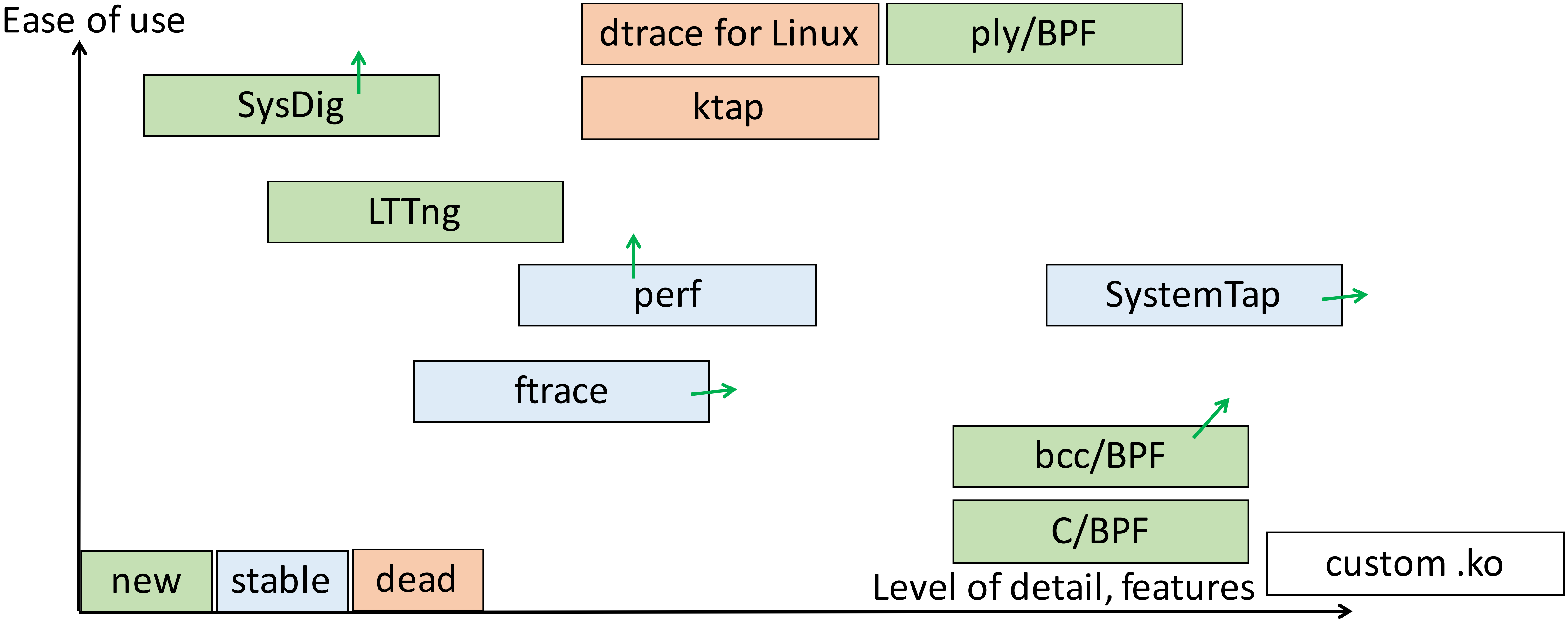
CPU Profiling and Tracing

- Profiling: sample stacks at a timed interval
  - Pros: Low (deterministic) overhead, simple
  - Cons: Coarse accuracy, but usually sufficient



- Tracing: event instrumentation

Linux Tracing Tools



Perf and Flame Graphs

Linux perf_events

- Standard Linux profiler
  - Provides the `perf` command
  - Usually pkg added by linux-tools-common, etc.
- Many event sources:
  - Timer-based sampling
  - Hardware events (e.g. LLC misses)
  - Tracepoints (e.g. block:block_rq_complete)
  - Dynamic tracing (kprobes, uprobes)
- Can sample stacks of (almost) everything on CPU
  - Can miss hard interrupt ISRs, but these should be near-zero and can be measured separately if needed

perf

- Developed in-tree and actively maintained, new features landing often
  - Multi-tool for a variety of performance investigations
  - Records into **perf.data** for post-processing

```
# perf

 usage: perf [--version] [--help] [OPTIONS] COMMAND [ARGS]

 The most commonly used perf commands are:
   annotate        Read perf.data (created by perf record) and display annotated code
   archive         Create archive with object files with build-ids found in perf.data
   bench           General framework for benchmark suites
   buildid-cache   Manage build-id cache.
   buildid-list    List the buildids in a perf.data file
   config          Get and set variables in a configuration file.
   data            Data file related processing
   diff            Read perf.data files and display the differential profile
   evlist          List the event names in a perf.data file
   inject          Filter to augment the events stream with additional information
   kmem            Tool to trace/measure kernel memory properties
   kvm             Tool to trace/measure kvm guest os
   list            List all symbolic event types
   lock            Analyze lock events
   mem             Profile memory accesses
   record          Run a command and record its profile into perf.data
   report          Read perf.data (created by perf record) and display the profile
   sched           Tool to trace/measure scheduler properties (latencies)
   script          Read perf.data (created by perf record) and display trace output
   stat            Run a command and gather performance counter statistics
   test            Runs sanity tests.
   timechart       Tool to visualize total system behavior during a workload
   top             System profiling tool.
   probe           Define new dynamic tracepoints
   trace           strace inspired tool

 See 'perf help COMMAND' for more information on a specific command.
```

perf record Profiling

- Stack profiling on all CPUs at 99 Hertz, then dump:

```
# perf record -F 99 -ag -- sleep 30
[ perf record: Woken up 9 times to write data ]
[ perf record: Captured and wrote 2.745 MB perf.data (~119930 samples) ]
# perf script
[…]
bash 13204 cpu-clock:
                459c4c dequote_string (/root/bash-4.3/bash)
                465c80 glob_expand_word_list (/root/bash-4.3/bash)
                466569 expand_word_list_internal (/root/bash-4.3/bash)
                465a13 expand_words (/root/bash-4.3/bash)
                43bbf7 execute_simple_command (/root/bash-4.3/bash)
                435f16 execute_command_internal (/root/bash-4.3/bash)
                435580 execute_command (/root/bash-4.3/bash)
                43a771 execute_while_or_until (/root/bash-4.3/bash)
                43a636 execute_while_command (/root/bash-4.3/bash)
                436129 execute_command_internal (/root/bash-4.3/bash)
                435580 execute_command (/root/bash-4.3/bash)
                420cd5 reader_loop (/root/bash-4.3/bash)
                41ea58 main (/root/bash-4.3/bash)
          7ff2294edec5 __libc_start_main (/lib/x86_64-linux-gnu/libc-2.19.so)
[… ~47,000 lines truncated …]
```

one stack sample

perf report Summary

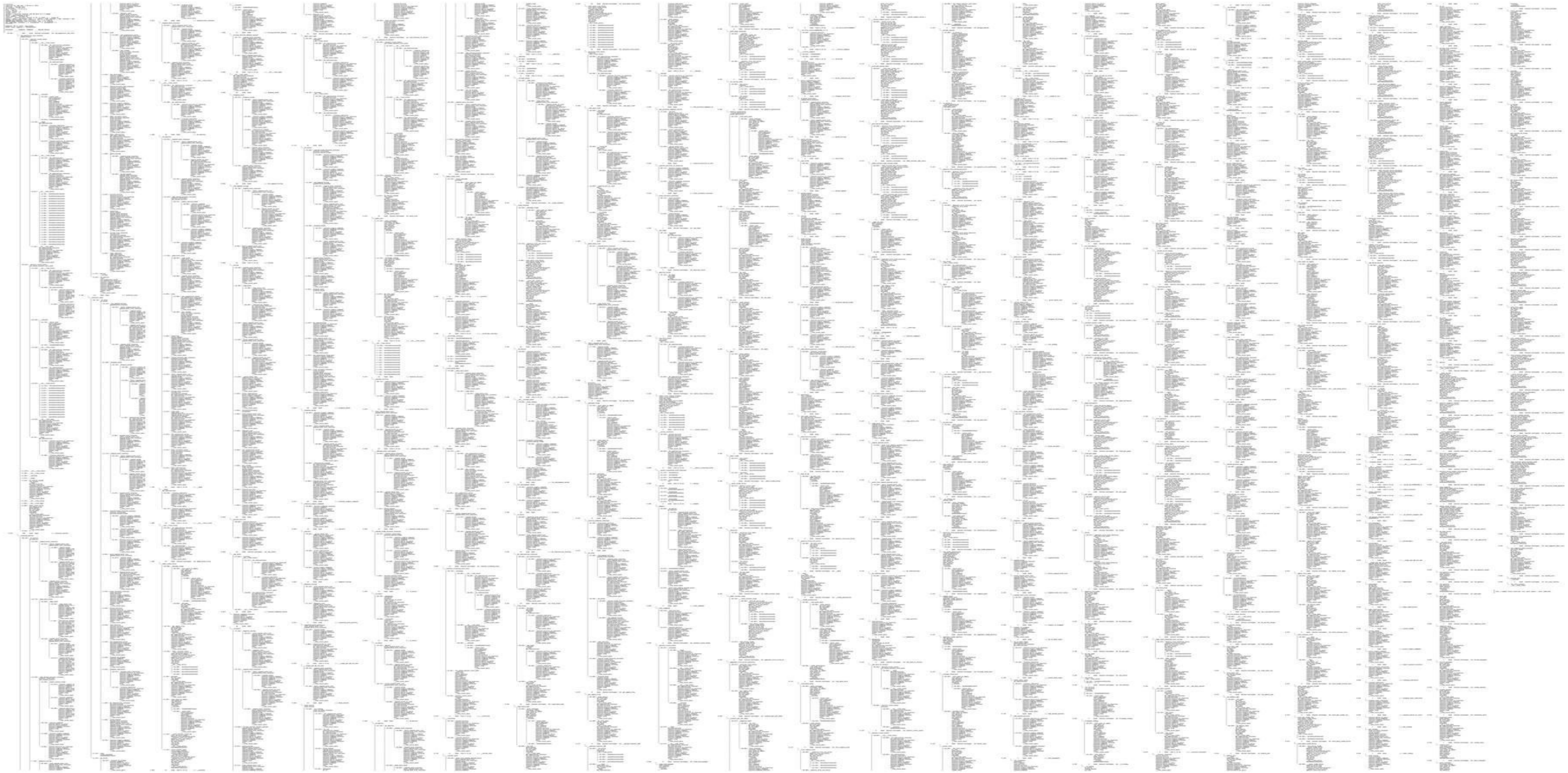
- Generates a call tree and combines samples:

```
# perf report -n –stdio
[…]
# Overhead       Samples  Command      Shared Object                            Symbol
# ........  ............  .......  .................  ..............................
#
    20.42%           605     bash  [kernel.kallsyms]  [k] xen_hypercall_xen_version
                    |
                    --- xen_hypercall_xen_version
                        check_events
                       |
                       |--44.13%-- syscall_trace_enter
                       |           tracesys
                       |           |
                       |           |--35.58%-- __GI___libc_fcntl
                       |           |           |
                       |           |           |--65.26%-- do_redirection_internal
                       |           |           |          do_redirections
                       |           |           |          execute_builtin_or_function
                       |           |           |          execute_simple_command
[… ~13,000 lines truncated …]
```

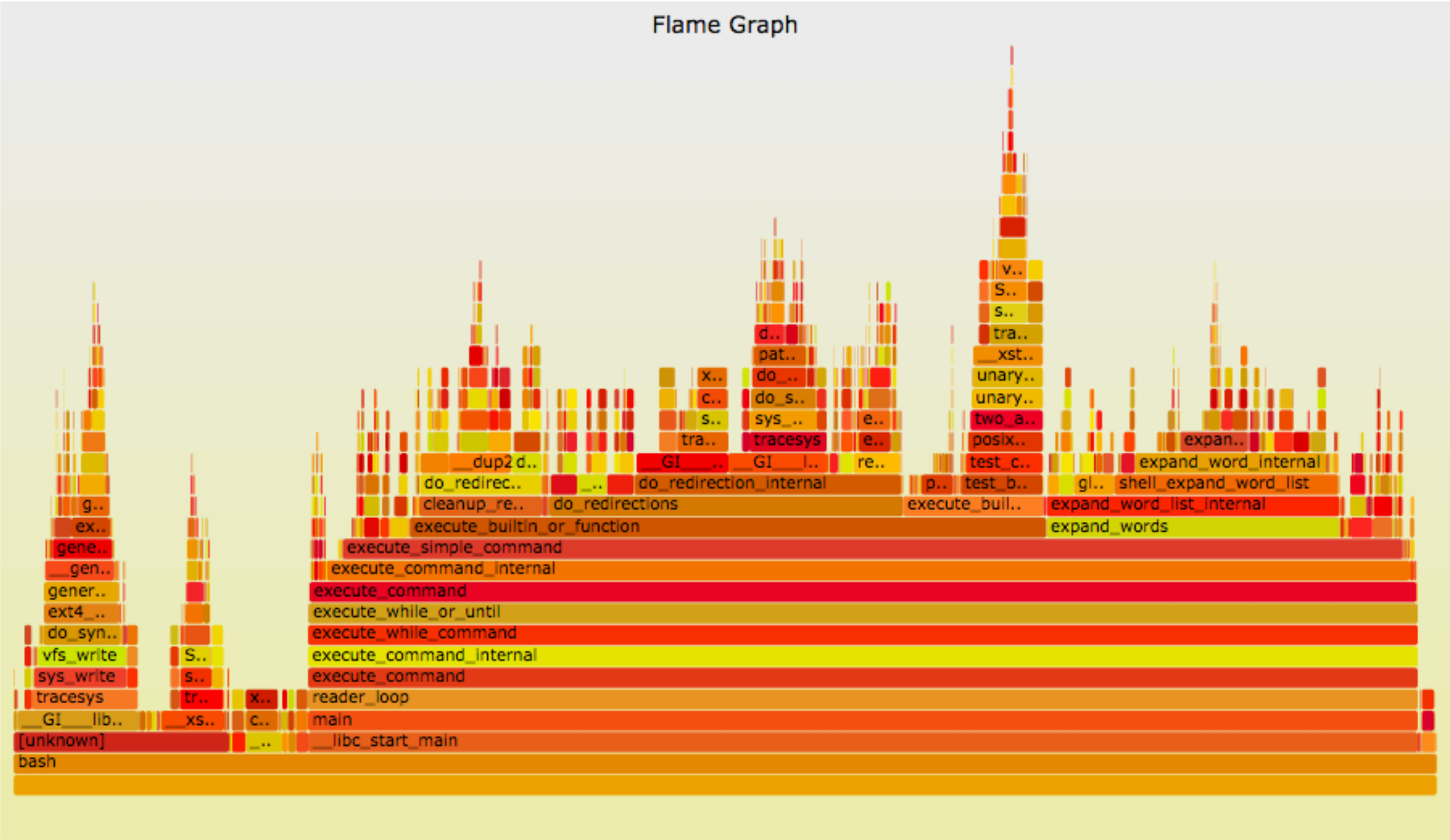
call tree summary

fewer lines, but still too many

Full perf report Output

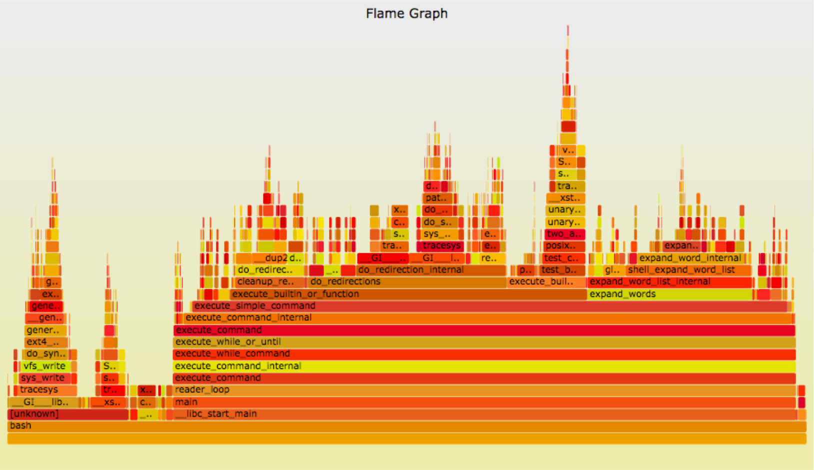


… as a Flame Graph



Flame Graphs

- A visual approach for summarizing stack traces
- Flame graphs:
  - **x-axis**: alphabetical stack sort, to maximize merging
  - **y-axis**: stack depth
  - **color**: random (default), or a dimension
- Currently made from Perl + SVG + JavaScript
  - https://github.com/brendangregg/FlameGraph
  - Multiple d3 versions are also being developed
- Easy to make
  - Converters for many profilers



Linux CPU Flame Graphs

- Linux 2.6+, via **perf.data** and `perf script`:

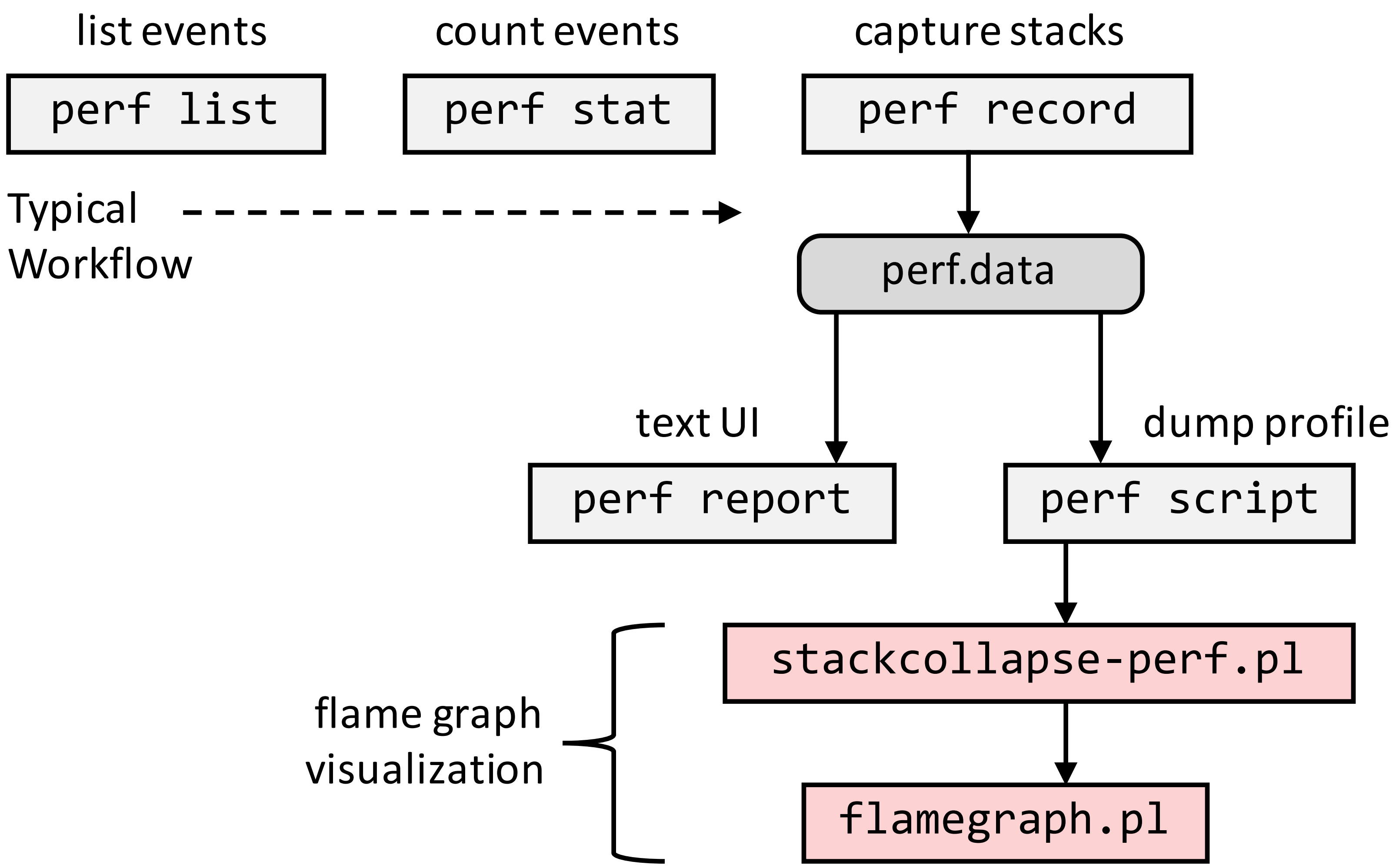
```
# git clone --depth 1 https://github.com/brendangregg/FlameGraph
# cd FlameGraph
# perf record -F 99 -a –g -- sleep 30
# perf script | ./stackcollapse-perf.pl |./flamegraph.pl > perf.svg
```

- Linux 4.5+ can generate folded output, skipping the costly folding step
- Linux 4.9+, via BPF:

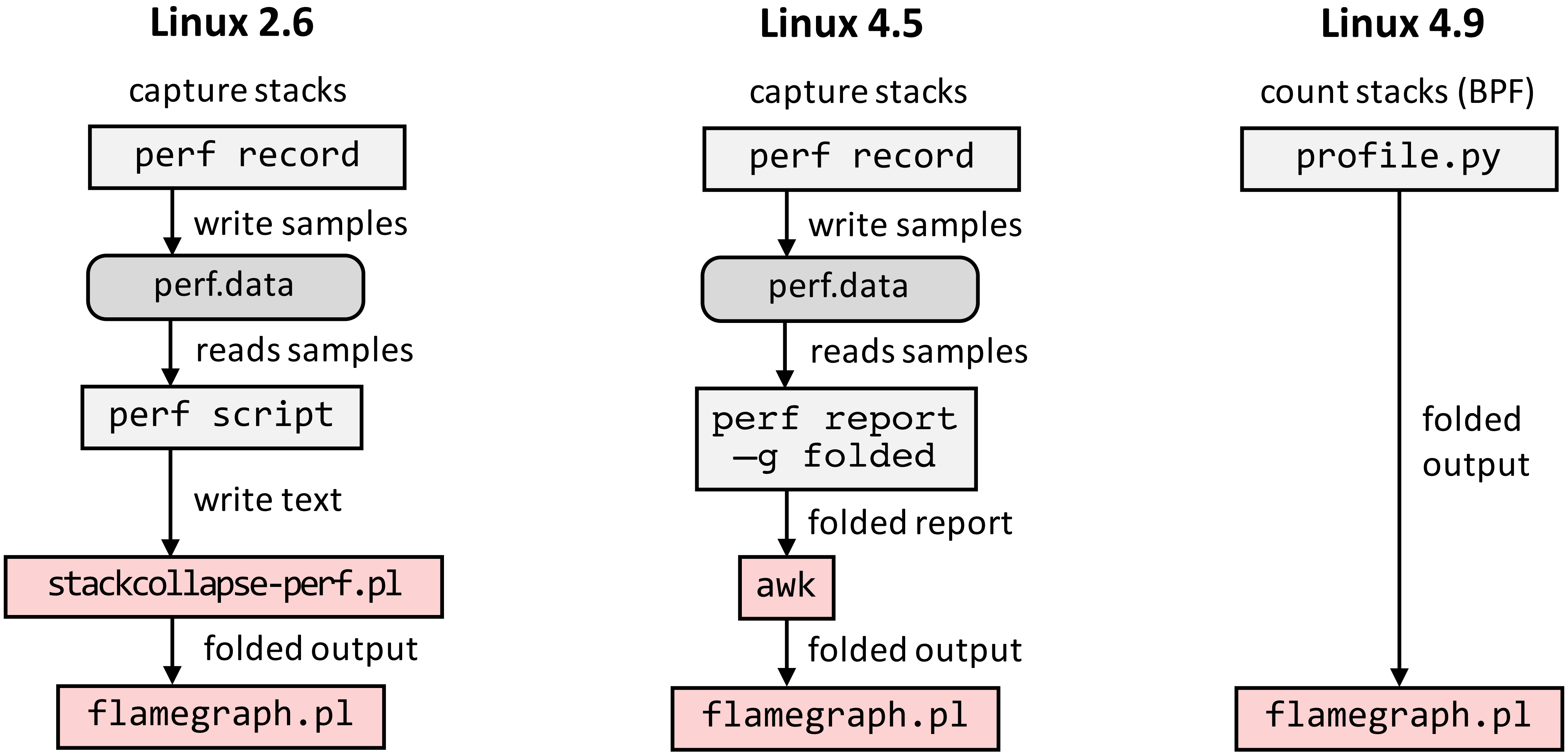
```
# git clone --depth 1 https://github.com/brendangregg/FlameGraph
# git clone --depth 1 https://github.com/iovisor/bcc
# ./bcc/tools/profile.py -dF 99 30 | ./FlameGraph/flamegraph.pl > perf.svg
```

  - Most efficient: no **perf.data** file, summarizes in-kernel

Linux 2.6: perf Workflow



Linux Profiling Optimizations



Broken Stacks

```
# perf script
[…]
java  4579 cpu-clock:
      7f417908c10b [unknown] (/tmp/…
java 8131 cpu-clock:
    7fff76f2dce1 [unknown] ([vdso])
    7fd3173f7a93 os::javaTimeMillis() (/usr/lib/jvm…
    7fd301861e46 [unknown] (/tmp/perf-8131.map)
    7fd30184def8 [unknown] (/tmp/perf-8131.map)
    7fd3010004e7 [unknown] (/tmp/perf-8131.map)
[…]
    7fd317a7e182 start_thread (/lib/x86_64-linux-gn…
```

- Fixing stacks:
  - `java -XX:+PreserveFramePointer`
  - `gcc -fno-omit-frame-pointer`
  - libunwind/DWARF and `perf -g dwarf`

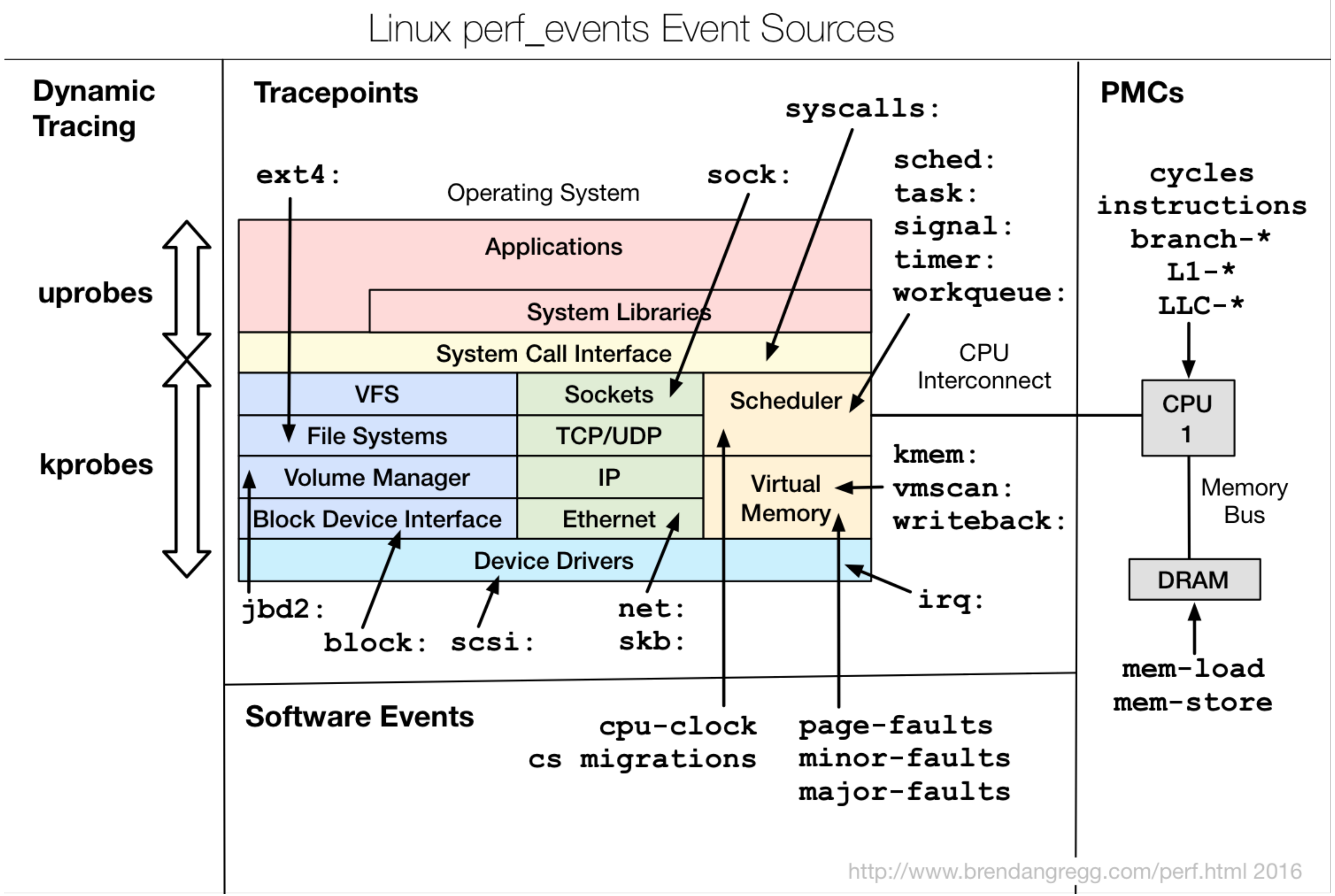
Missing Symbols

- perf can use external symbol files: /tmp/perf-PID.map

```
# perf script
Failed to open /tmp/perf-8131.map, continuing without symbols
[…]
java 8131 cpu-clock:
    7fff76f2dce1 [unknown] ([vdso])
    7fd3173f7a93 os::javaTimeMillis() (/usr/lib/jvm…
    7fd301861e46 [unknown] (/tmp/perf-8131.map)
[…]
```

- Java: perf-map-agent (perf symbol logging), jmaps
- Node.js: `--perf_basic_prof_only_functions`
- .NET Core: `COR_PerfMapEnabled=1`

```
# perf script
 java 14025 [017]  8048.157085: cpu-clock:
     7fd781253265 Ljava/util/HashMap;::get (/tmp/perf-12149.map)
[…]
```


Linux perf_events Event Sources
Dynamic Tracing
uprobes
kprobes
Tracepoints
ext4:
Operating System
sock:
syscalls:
sched:
task:
signal:
timer:
workqueue:
Applications
System Libraries
System Call Interface
VFS
Sockets
Scheduler
File Systems
TCP/UDP
Volume Manager
IP
Virtual Memory
Block Device Interface
Ethernet
Device Drivers
CPU Interconnect
kmem:
vmscan:
writeback:
irq:
jbd2:
block:
scsi:
net:
skb:
PMCs
cycles
instructions
branch-*
L1-*
LLC-*
CPU 1
Memory Bus
DRAM
mem-load
mem-store
Software Events
cpu-clock
cs migrations
page-faults
minor-faults
major-faults
http://www.brendangregg.com/perf.html 2016

perf_events: Counters

- Performance Monitoring Counters (PMCs):

```
$ perf list | grep –i hardware
  cpu-cycles OR cycles                               [Hardware event]
  stalled-cycles-frontend OR idle-cycles-frontend    [Hardware event]
  stalled-cycles-backend OR idle-cycles-backend      [Hardware event]
  instructions                                       [Hardware event]
[…]
  branch-misses                                      [Hardware event]
  bus-cycles                                         [Hardware event]
  L1-dcache-loads                                    [Hardware cache event]
  L1-dcache-load-misses                              [Hardware cache event]
[…]
  rNNN (see 'perf list --help' on how to encode it)  [Raw hardware event …
  mem:<addr>[:access]                                [Hardware breakpoint]
```

- Identify CPU cycle breakdowns, esp. stall types
- PMCs not enabled by-default in clouds (yet)
- Can be time-consuming to use (CPU manuals)

perf_events: Tracepoints

```
# perf record -e skb:consume_skb -ag
^C[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.065 MB perf.data (~2851 samples) ]

# perf report
[...]
    74.42%  swapper  [kernel.kallsyms]  [k] consume_skb
            |
            --- consume_skb
                arp_process
                arp_rcv
                __netif_receive_skb_core
                __netif_receive_skb
                netif_receive_skb
                virtnet_poll
                net_rx_action
                __do_softirq
                irq_exit
                do_IRQ
                ret_from_intr
 […]
```

Summarizing stack traces for a tracepoint

One-Liners: Static Tracing

```
# Trace new processes, until Ctrl-C:
perf record -e sched:sched_process_exec –a

# Trace all context-switches with stack traces, for 1 second:
perf record -e context-switches –ag -- sleep 1

# Trace CPU migrations, for 10 seconds:
perf record -e migrations -a -- sleep 10

# Trace all connect()s with stack traces (outbound connections), until Ctrl-C:
perf record -e syscalls:sys_enter_connect –ag

# Trace all block device issues and completions (has timestamps), until Ctrl-C:
perf record -e block:block_rq_issue -e block:block_rq_complete -a

# Trace all block completions, of size at least 100 Kbytes, until Ctrl-C:
perf record -e block:block_rq_complete --filter 'nr_sector > 200'

# Trace all block completions, synchronous writes only, until Ctrl-C:
perf record -e block:block_rq_complete --filter 'rwbs == "WS"'

# Trace all ext4 calls, and write to a non-ext4 location, until Ctrl-C:
perf record -e 'ext4:*' -o /tmp/perf.data -a
```

Tracing Stack

add-on tools:	trace-cmd, perf-tools, bcc, …
front-end tools:	perf
tracing frameworks:	ftrace, perf_events, BPF
back-end instrumentation:	tracepoints, kprobes, uprobes

in Linux

Lab

CPU profiling with perf and flame graphs

BPF and BCC

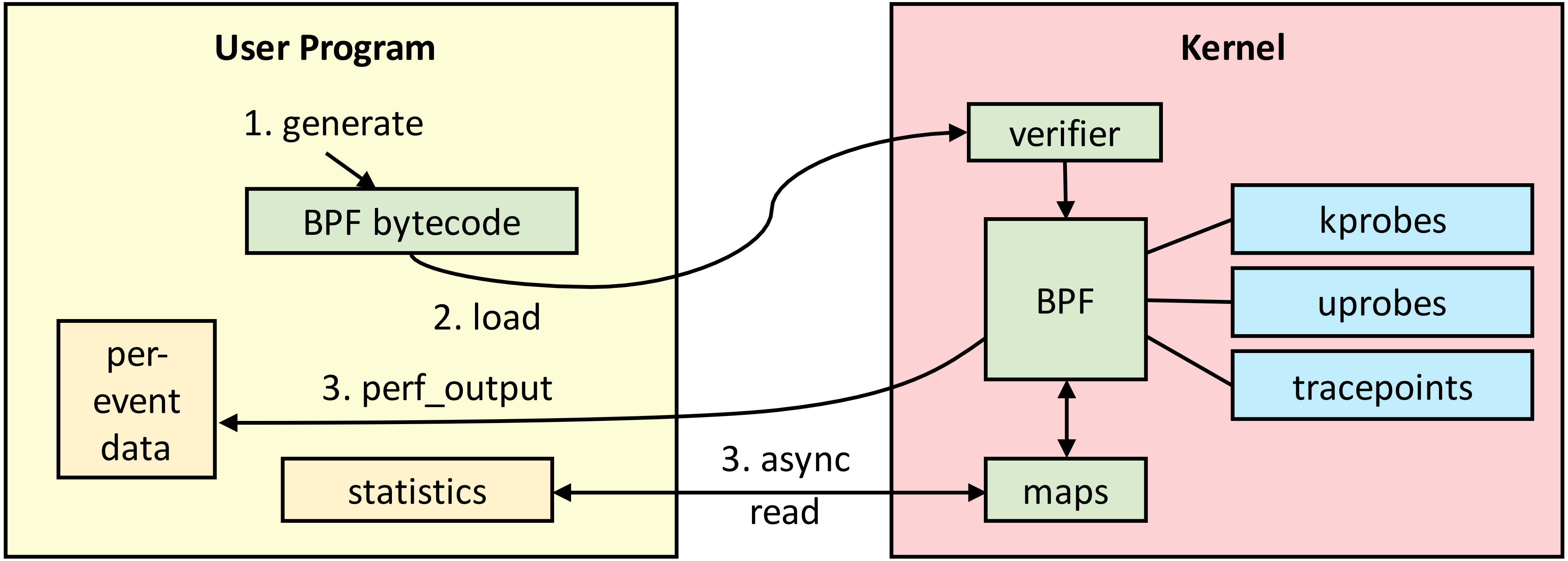
Berkeley Packet Filters (BPF)

- Originally designed for, well, packet filtering:
  `dst port 80 and len >= 100`
- Custom instruction set, interpreted/JIT compiled

```
0: (bf) r6 = r1
1: (85) call 14
2: (67) r0 <<= 32
3: (77) r0 >>= 32
4: (15) if r0 == 0x49f goto pc+40
```

Extended BPF

- Used for virtual network, security, tracing
- Multiple front-ends: C, perf, SystemTap, bcc, ply, …



Extended BPF

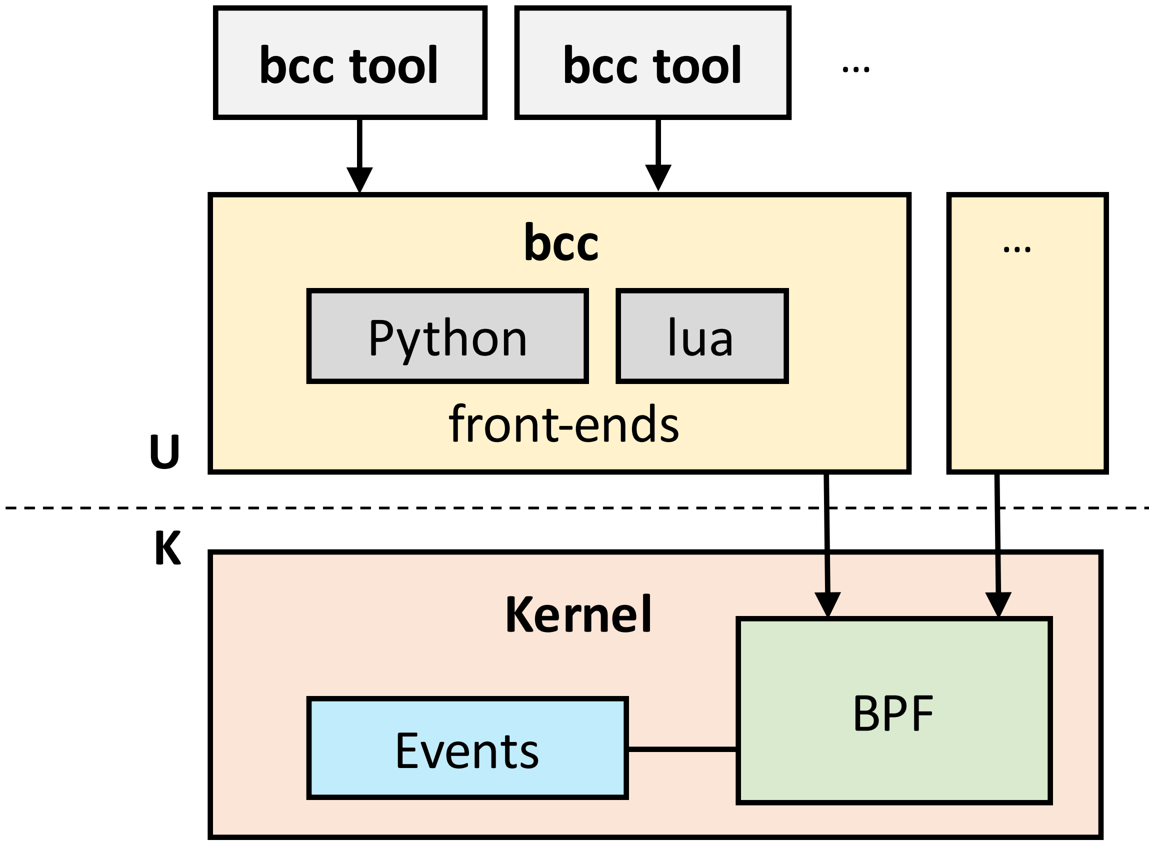
- 3.19: attach to sockets, map data structures
- 4.1: attach to kprobes
- 4.3: attach to uprobes
- 4.4: BPF output
- 4.6: stack traces
- 4.7: attach to tracepoints
- 4.9: profiling
- 4.9: attach to PMCs and software events

BPF mascot

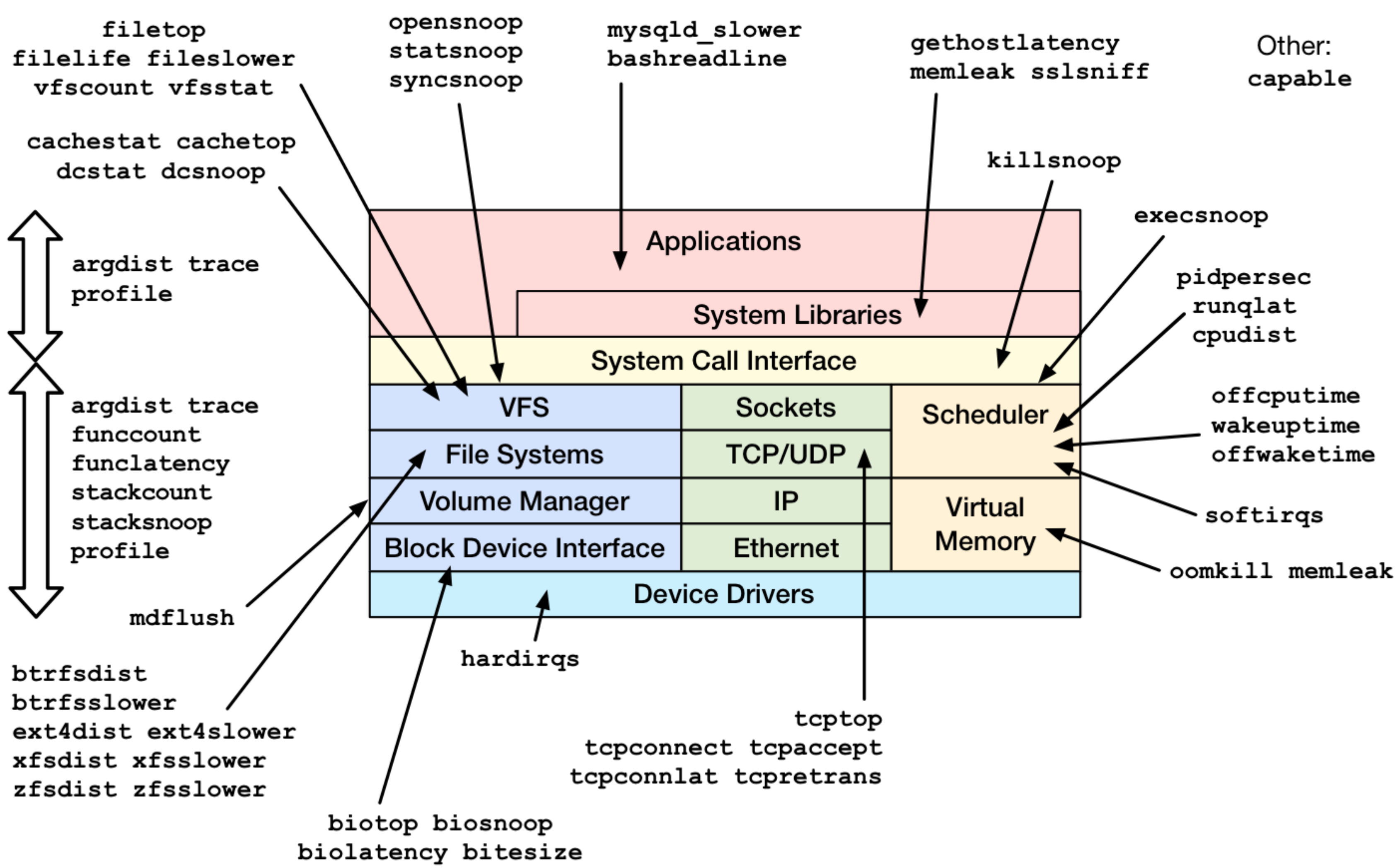
BCC: BPF Compiler Collection

- Library and Python/Lua module for compiling, loading, and executing BPF programs
  - https://github.com/iovisor/bcc
  - C + Python/Lua front-end for BPF
- Includes many tracing tools

Tracing layers:



Linux bcc/BPF Tracing Tools



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

BCC Tools

```
$ ls *.py          dcstat.py          memleak.py       tcpaccept.py
argdist.py         execsnoop.py       offcputime.py    tcpconnect.py
bashreadline.py    ext4dist.py        offwaketime.py   tcpconnlat.py
biolatency.py      ext4slower.py      oomkill.py       tcpretrans.py
biosnoop.py        filelife.py        opensnoop.py     tplist.py
biotop.py          fileslower.py      pidpersec.py     trace.py
bitesize.py        filetop.py         profile.py       vfscount.py
btrfsdist.py       funccount.py       runqlat.py       vfsstat.py
btrfsslower.py     funclatency.py     softirqs.py      wakeuptime.py
cachestat.py       gethostlatency.    solisten.py      xfsdist.py
cachetop.py        py                 stackcount.py    xfsslower.py
capable.py         hardirqs.py        stacksnoop.py    zfsdist.py
cpudist.py         killsnoop.py       statsnoop.py     zfsslower.py
dcsnoop.py         mdflush.py         syncsnoop.py
```

Installation

https://github.com/iovisor/bcc/blob/master/INSTALL.md

- E.g. on 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
```

- Or, build from source

BCC General Performance Checklist

1. execsnoop
2. opensnoop
3. ext4slower
   (or btrfs*, xfs*, zfs*)
4. biolatency
5. biosnoop
6. cachestat
7. tcpconnect
8. tcpaccept
9. tcpretrans
10. gethostlatency
11. runqlat
12. profile

Specialized Tools

```
# hardirqs
Tracing hard irq event time... Hit Ctrl-C to end.
^C
HARDIRQ                           TOTAL_usecs
virtio0-input.0                           959
ahci[0000:00:1f.2]                       1290

# biolatency
Tracing block device I/O... Hit Ctrl-C to end.
^C
     usecs               : count     distribution
        64 -> 127        : 7        |*********                               |
       128 -> 255        : 14       |******************                      |
       256 -> 511        : 5        |******                                  |
       512 -> 1023       : 30       |****************************************|
      1024 -> 2047       : 1        |*                                       |
```

Specialized Tools

```
# 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
^C
```

```
# 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
^C
```

Specialized Tools

```
# 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
2287   sshd         6  ::1              ::1              22
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
^C
```

```
# opensnoop
PID    COMM               FD ERR PATH
27159  catalina.sh         3   0 /apps/tomcat8/bin/setclasspath.sh
4057   redis-server        5   0 /proc/4057/stat
30668  sshd                4   0 /proc/sys/kernel/ngroups_max
30668  sshd                4   0 /etc/group
30668  sshd                4   0 /root/.ssh/authorized_keys
^C
```

BCC/BPF Tracing Targets (December 2016)

Target	Support	Overhead
kprobes	Native	Low
uprobes	Native	Medium *handler runs in KM*
Kernel tracepoints (4.7+)	Native	Low
USDT tracepoints	Temporary *through uprobes*	Medium *handler runs in KM*
Perf events (4.9+)	Native	Low

Multi-Tools: stackcount

```
# stackcount __kmalloc
Tracing 1 functions for "__kmalloc"... Hit Ctrl-C to end.
^C
 __kmalloc
  alloc_fdtable
  dup_fd
  copy_process.part.31
  _do_fork
  sys_clone
  do_syscall_64
  return_from_SYSCALL_64
    4

  __kmalloc
  create_pipe_files
  __do_pipe_flags
  sys_pipe
  entry_SYSCALL_64_fastpath
    6

  __kmalloc
  htree_dirblock_to_tree
  ext4_htree_fill_tree
  ext4_readdir
  iterate_dir
  SyS_getdents
  entry_SYSCALL_64_fastpath
    14
```

Multi-Tools: argdist

```
# argdist -C 'p:c:write(int fd, const void *buf,
  size_t count):size_t:count:fd==1'
[01:49:00]
p:c:write(int fd, const void *buf, size_t count):size_t:count:fd==1
        COUNT      EVENT
        1          count = 3134
        1          count = 170
        1          count = 181
        2          count = 18
        3          count = 30
```

Multi-Tools: argdist

```
# argdist -i 5 -H 'r::__vfs_read(void *file, void *buf,
  size_t count):size_t:$entry(count):$latency > 1000000'
[01:51:40]
     count               : count     distribution
         0 -> 1          : 20       |****************************************|
         2 -> 3          : 0        |                                        |
         4 -> 7          : 0        |                                        |
         8 -> 15         : 0        |                                        |
        16 -> 31         : 0        |                                        |
        32 -> 63         : 0        |                                        |
        64 -> 127        : 0        |                                        |
       128 -> 255        : 6        |************                            |
       256 -> 511        : 0        |                                        |
       512 -> 1023       : 0        |                                        |
      1024 -> 2047       : 1        |**                                      |
```

Multi-Tools: 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    |****************************************|
^C
```

Multi-Tools: trace

```
# trace 'r:/usr/bin/bash:readline "%s", retval'
TIME     PID    COMM         FUNC             -
02:02:26 3711   bash         readline         ls –la
02:02:36 3711   bash         readline         wc -l src.c

# tplist -v block:block_rq_complete
block:block_rq_complete
    dev_t dev;
    sector_t sector;
    unsigned int nr_sector;
    int errors;
    char rwbs[8];

# trace 't:block:block_rq_complete "sectors=%d", args->nr_sector'
TIME     PID    COMM         FUNC             -
02:03:56 0      swapper/0    block_rq_complete sectors=16
02:03:56 0      swapper/0    block_rq_complete sectors=8
02:03:58 0      swapper/0    block_rq_complete sectors=24
02:04:00 0      swapper/0    block_rq_complete sectors=0
```

Multi-Tools: trace

```
# tplist -l pthread -vv libpthread:pthread_create
/lib64/libpthread.so.0 libpthread:pthread_create [sema 0x0]
  location #0 0x7d73
    argument #0 8 unsigned bytes @ ax
    argument #1 8 unsigned bytes @ *(bp - 192)
    argument #2 8 unsigned bytes @ *(bp - 168)
    argument #3 8 unsigned bytes @ *(bp - 176)

# trace 'u:pthread:pthread_create "%U", arg3'
TIME     PID    COMM         FUNC              -
02:07:29 4051   contentions  pthread_create    primes_thread+0x0
02:07:29 4051   contentions  pthread_create    primes_thread+0x0
02:07:29 4051   contentions  pthread_create    primes_thread+0x0
02:07:29 4051   contentions  pthread_create    primes_thread+0x0
^C
```

Multi-Tools: trace

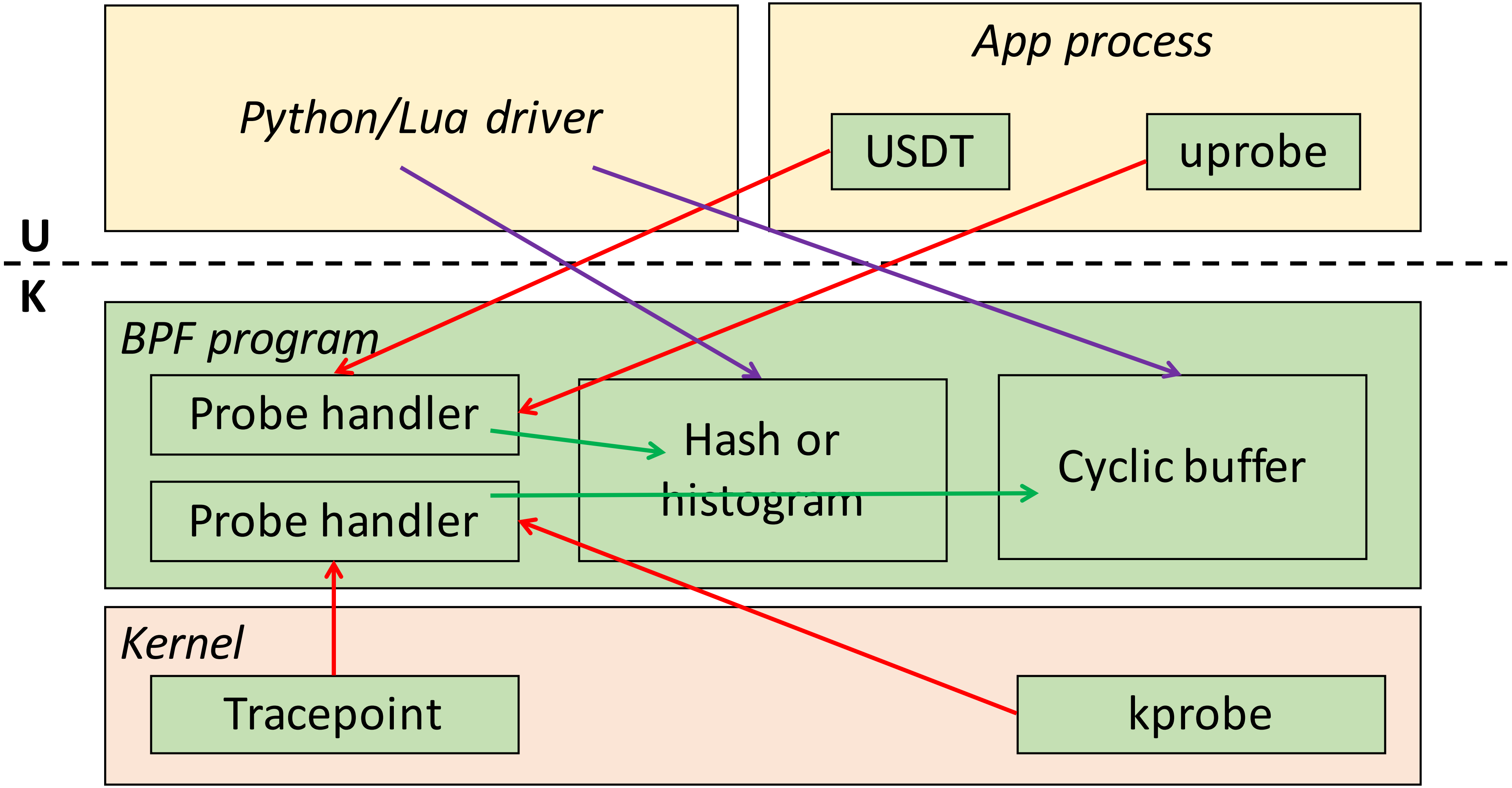
```
# trace -p $(pidof node) 'u:node:http__server__request
                          "%s %s (from %s:%d)" arg5, arg6, arg3, arg4'
TIME     PID   COMM FUNC                -
04:50:44 22185 node http__server__request GET /foofoo (from ::1:51056)
04:50:46 22185 node http__server__request GET / (from ::1:51056)
^C

# trace 'u:/tmp/libjvm.so:thread__start "%s [%d]", arg1, arg4' \
        'u:/tmp/libjvm.so:thread__stop "%s [%d]", arg1, arg4'
TIME     PID    COMM         FUNC             -
06:55:24 32157  java         thread__start    Reference Handler [32157]
06:55:24 32158  java         thread__start    Finalizer [32158]
06:55:24 32159  java         thread__start    Signal Dispatcher [32159]
06:55:24 32160  java         thread__start    C2 CompilerThread0 [32160]
06:55:24 32161  java         thread__start    C2 CompilerThread1 [32161]
06:55:24 32162  java         thread__start    C1 CompilerThread2 [32162]
06:55:24 32163  java         thread__start    Service Thread [32163]
06:55:28 32159  java         thread__stop     Signal Dispatcher [32159]
^C
```

Lab

BCC one-liners

Authoring BCC Tools

Custom Tool Design



BPF Program: Counting Allocations

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

struct alloc_info_t {
        u64 count;
        u64 size;
};

BPF_HASH(allocs, u32, struct alloc_info_t);

int handler(struct pt_regs *ctx, size_t size) {
        u32 pid = bpf_get_current_pid_tgid();
        struct alloc_info_t init = { 0 }, *info;

        info = allocs.lookup_or_init(&pid, &init);
        info->count += 1;
        info->size += size;

        return 0;
}
```

BPF Driver

```
#!/usr/bin/env python
from bcc import BPF
from time import sleep

program = BPF(src_file="allocs.c")
program.attach_kprobe(event="__kmalloc", fn_name="handler")
allocs = program["allocs"]

while True:
        sleep(5)
        print("\n%-8s %-8s %-10s" % ("PID", "COUNT", "SIZE"))
        for key, value in sorted(allocs.items(), key=lambda (k, v): k.value):
                print("%-8d %-8d %-8d" % (key.value, value.count, value.size))
```

BPF Execution

```
# ./allocs.py
PID       COUNT     SIZE
28064     3         456
28157     10        76
28158     5         1116

PID       COUNT     SIZE
28001     113       1828
28064     8         1216
28110     38        683
28157     46        328
28158     5         1116
28159     41        12894
^C
```

Inline BPF Program

```
#!/usr/bin/env python
from bcc import BPF
from time import sleep

program = BPF(text="""BPF_HASH(counts, u32, u32);
TRACEPOINT_PROBE(irq, irq_handler_entry) {
  u32 zero = 0, *existing, irq = args->irq;
  existing = counts.lookup_or_init(&irq, &zero);
  ++(*existing);
  return 0;
}""")


counts = program["counts"]
sleep(9999999)
print("\n%-8s %-8s" % ("IRQ", "COUNT"))
for key, value in counts.items():
       print("%-8d %-8d" % (key.value, value.value))
```

BPF Attach Targets

```
bpf = BPF(text=...)
bpf.attach_kprobe(event="vfs_read", fn_name="trace_read")
bpf.attach_kretprobe(event="schedule", fn_name="trace_schedule")
bpf.attach_uprobe(name="pthread", sym="pthread_mutex_lock", fn_name="trace_lock")
bpf.attach_uretprobe(name="pthread", sym="pthread_create", fn_name="trace_create")
bpf.attach_tracepoint(tp="net:net_dev_start_xmit", fn_name="trace_xmit")

usdt = USDT(pid=123, path="/lib64/libpthread.so.0")
usdt.enable_probe("mutex_acquired")
bpf = BPF(text=..., usdt_contexts=[usdt])
```

Data Types

- Array
- Hash
- Histogram
- Perf buffer (4.4+)
- Stack map (4.6+)

Example: Histogram

```
struct dist_key_t {
  char op[OP_NAME_LEN];
  u64 slot;
};
BPF_HISTOGRAM(dist, struct dist_key_t);

...
struct dist_key_t key = { .slot=bpf_log2l(elapsed_time) };
__builtin_memcpy(&key.op, op, sizeof(key.op));
dist.increment(key);

...
bpf.get_table("dist").print_log2_hist("operation")
```

Example: Perf Buffer

```
bpf = BPF(text="""#include <linux/ptrace.h>
struct data_t { u64 pid; char str[80]; };
BPF_PERF_OUTPUT(events);
int print(struct pt_regs *ctx) {
  struct data_t data = {0};
  ...
  events.perf_submit(ctx, &data, sizeof(data));
  return 0;
}""")

class Data(ct.Structure):
  _fields_ = [ ("pid", ct.c_ulonglong), ("str", ct.c_char*80) ]

bpf.attach_uretprobe(name="/bin/bash", sym="readline", fn_name="print")

b["events"].open_perf_buffer(lambda cpu, data, size:
  event = ct.cast(data, ct.POINTER(Data)).contents
  print(event)
)
while True: bpf.kprobe_poll()
```

Example: Stack Map

```
BPF_HASH(counts, int);
BPF_STACK_TRACE(stacks, 1024);

...
int key = stacks.get_stackid(ctx, BPF_F_REUSE_STACKID);
u64 zero = 0;
u64 *val = counts.lookup_or_init(&key, &zero);
++(*val);

...
counts, stacks = bpf["counts"], bpf["stacks"]
for k, v in counts:
  for addr in stacks.walk(k.value):
    print(BPF.ksym(addr))
  print(v.value)
```

Custom Tool Design Tips

- Try to perform all aggregations in the BPF program and keep copying to user space to a minimum
- Limit hash/histogram/stackmap sizes, prune, keep only top entries
- Clear cyclic buffer often and quickly

Deployment

- For Python tools, deploy Python + libbcc.so
- For Lua tools, deploy only **bcc-lua**
  - Statically links libbcc.a but allows plugging libbcc.so
- Kernel build flags:
  - `CONFIG_BPF=y`
  - `CONFIG_BPF_SYSCALL=y`
  - `CONFIG_BPF_JIT=y`
  - `CONFIG_HAVE_BPF_JIT=y`
  - `CONFIG_BPF_EVENTS=y`

Lab

Authoring BCC tools

Summary

- Tracing can identify bugs and performance issues that no debugger or profiler can catch
- Tools make low-overhead, dynamic, production tracing possible
- Flame graphs help visualize complex stack trace information and other hierarchical data
- BPF is the next-generation backend for Linux tracing tools

Learning Objectives

- Understand flame graphs and how to interpret them
- Perform commands using Linux perf to create a CPU flame graph
- Understand the role of BPF and Linux tracing
- Gain experience with installing and using bcc/BPF tools
- Apply methodology for analyzing system performance
- Identify bcc/BPF tool source code components
- Make simple customizations to a bcc/BPF tool
- Identify reference documentation for bcc development
- Optionally develop a from-scratch bcc/BPF tool

References

- Perf and flame graphs
  - https://perf.wiki.kernel.org/index.php/Main_Page
  - http://www.brendangregg.com/flamegraphs.html
- BCC tutorials (by Brendan Gregg)
  - https://github.com/iovisor/bcc/blob/master/docs/tutorial.md
  - https://github.com/iovisor/bcc/blob/master/docs/tutorial_bcc_python_developer.md
  - https://github.com/iovisor/bcc/blob/master/docs/reference_guide.md
- BPF
  - https://github.com/torvalds/linux/tree/master/samples/bpf
  - https://www.kernel.org/doc/Documentation/networking/filter.txt
  - https://github.com/iovisor/bpf-docs

Thank You!

Brendan Gregg
@brendangregg
brendangregg.com
bregg@netflix.com

Sasha Goldshtein
@goldshtn
sashag.net
sashag@sela.co.il