openstack.

Low-Overhead System Tracing With eBPF

Akshay Kapoor

DevOps Engineer @ SAP Labs

May 2018

Low-Overhead System Tracing With eBPF

Low-Overhead System Tracing With eBPF

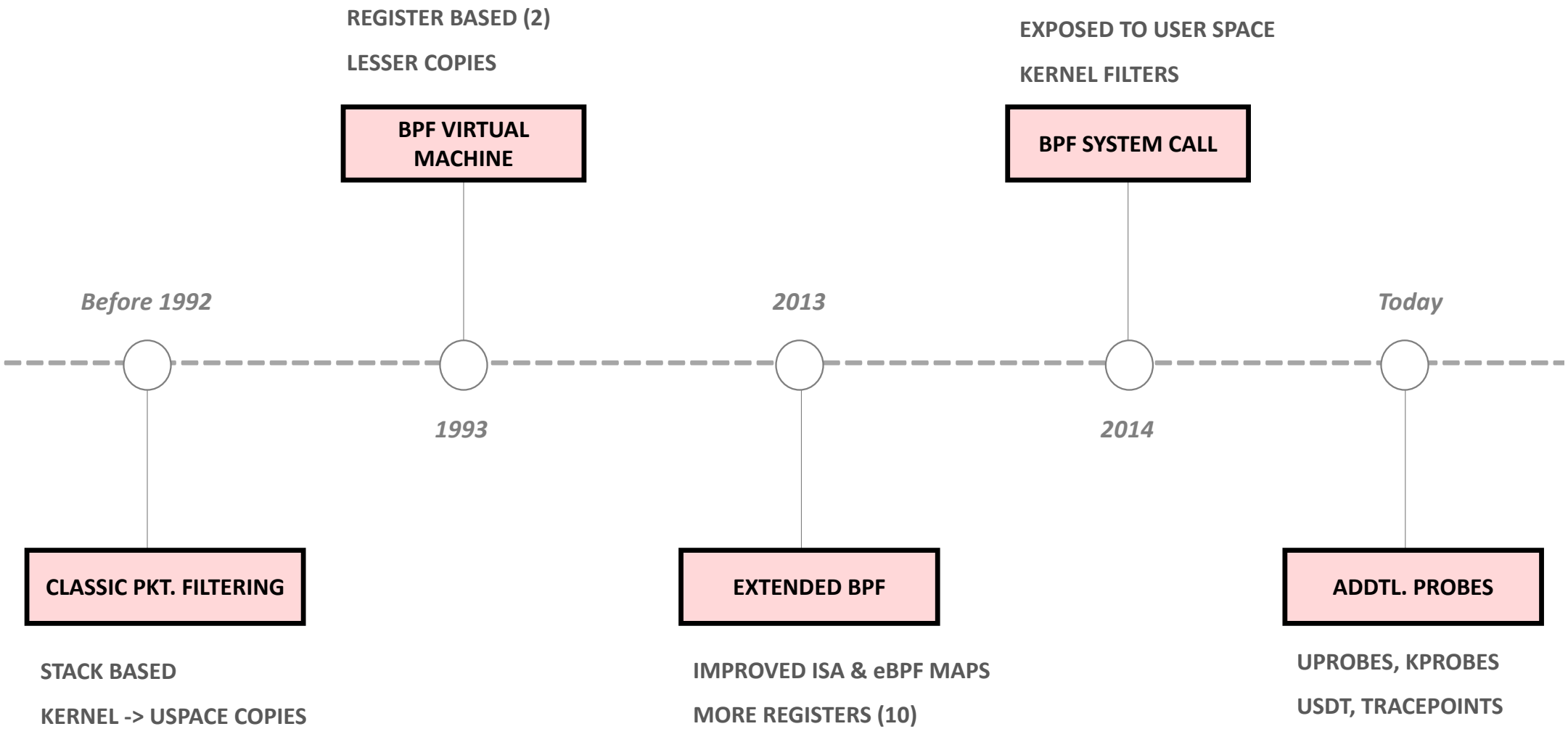
Low-Overhead System Tracing With eBPF

You don't need to know how to operate an X-ray machine,

but you do need to know that if you swallow a penny, an X-ray is an option!

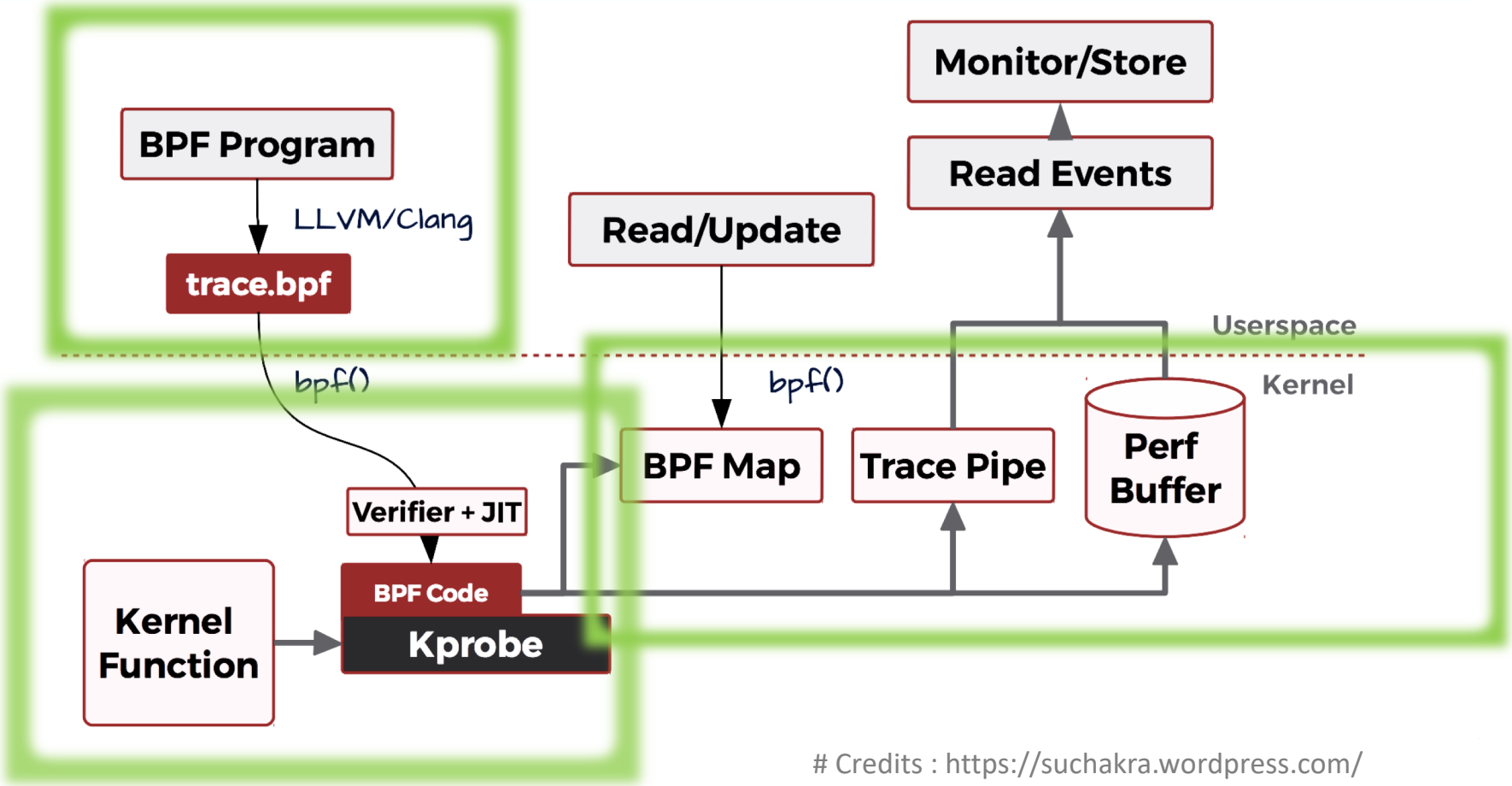
~ www.bredangregg.com

EVOLUTION OF BPF



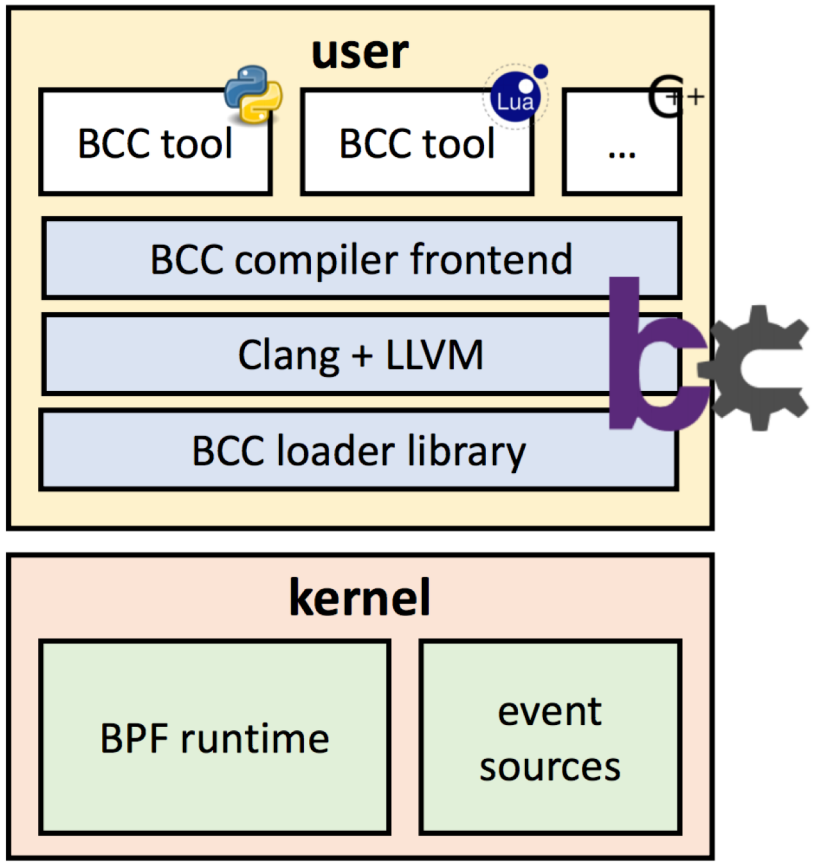
```
tcpdump -n "dst host 192.168.1.1 and dst port 23"
```

HOW BPF WORKS ?



Credits : https://suchakra.wordpress.com/

BCC (BPF COMPILER COLLECTION)

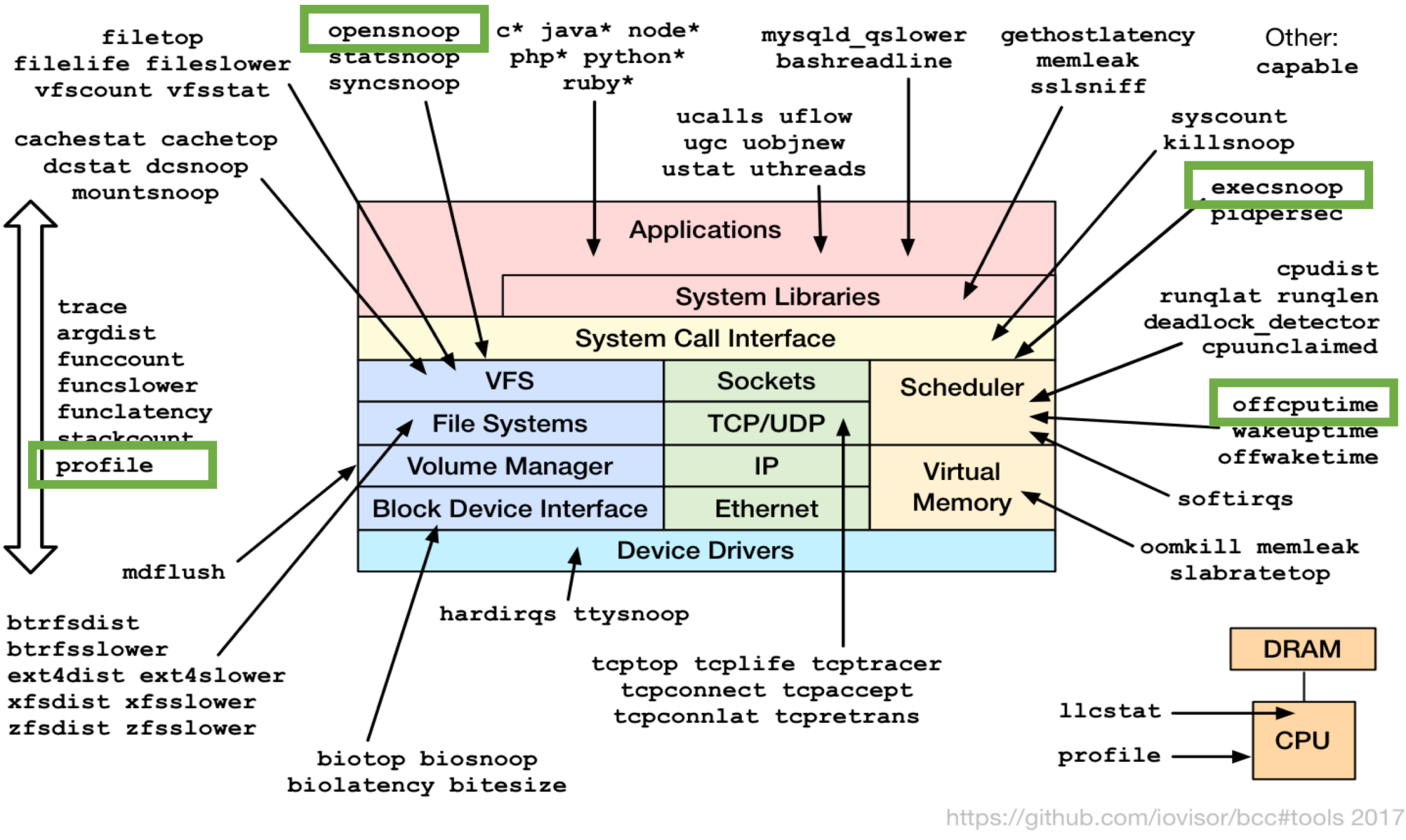



```
from bcc import BPF

BPF(
  text='
    int kprobe__sys_clone(void *ctx) {
        bpf_trace_printk("Hello, World!");
        return 0;
    }
  '
).trace_print()
```

- https://github.com/iovisor/bcc
- Lead Developer – Brenden Blanco

#Credits : Sasha Goldshtein


filetop
filelife fileslower
vfscount vfsstat
opensnoop
statsnoop
syncsnoop
c* java* node*
php* python*
ruby*
mysqld_qslower
bashreadline
gethostlatency
memleak
sslsniff
Other:
capable
cachestat cachetop
dcstat dcsnoop
mountsnoop
ucalls uflow
ugc uobjnew
ustat uthreads
syscount
killsnoop
execsnoop
pidpersec
Applications
System Libraries
System Call Interface
VFS
Sockets
Scheduler
File Systems
TCP/UDP
Volume Manager
IP
Virtual Memory
Block Device Interface
Ethernet
Device Drivers
cpudist
runqlat runqlen
deadlock_detector
cpuunclaimed
offcputime
wakeuptime
offwaketime
softirqs
oomkill memleak
slabratetop
trace
argdist
funccount
funcslower
funclatency
stackcount
profile
mdflush
btrfsdist
btrfsslower
ext4dist ext4slower
xfsdist xfsslower
zfsdist zfsslower
hardirqs ttysnoop
tcptop tcplife tcptracer
tcpconnect tcpaccept
tcpconnlat tcpretrans
biotop biosnoop
biolatency bitesize
DRAM
CPU
llcstat
profile
https://github.com/iovisor/bcc#tools 2017

BCC Tools [examples...]

```
# ./biolatency
Tracing block device I/O... Hit Ctrl-C to end.
^C
     usecs               : count     distribution
       0 -> 1            : 0        |                                        |
       2 -> 3            : 0        |                                        |
       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        |*                                       |
```

BCC Tools [examples...]

```
# ./memleak
Attaching to kernel allocators, Ctrl+C to quit.
[00:47:46] Top 10 stacks with outstanding allocations:
        4608 bytes in 36 allocations from stack
                kmem_cache_alloc+0x14a [kernel]
                sys_epoll_ctl+0x68d [kernel]
                entry_SYSCALL_64_fastpath+0x24 [kernel]
        5952 bytes in 24 allocations from stack
                kmem_cache_alloc+0x14a [kernel]
                inet_twsk_alloc+0x44 [kernel]
                tcp_time_wait+0x30 [kernel]
                tcp_fin+0xb5 [kernel]
                tcp_data_queue+0x584 [kernel]
                tcp_rcv_state_process+0x3af [kernel]
                tcp_v4_do_rcv+0xc1 [kernel]
                tcp_v4_rcv+0x9b8 [kernel]
                ip_local_deliver_finish+0x9b [kernel]
                ip_local_deliver+0x6f [kernel]
                ip_rcv_finish+0x124 [kernel]
```

BCC Tools [examples...]

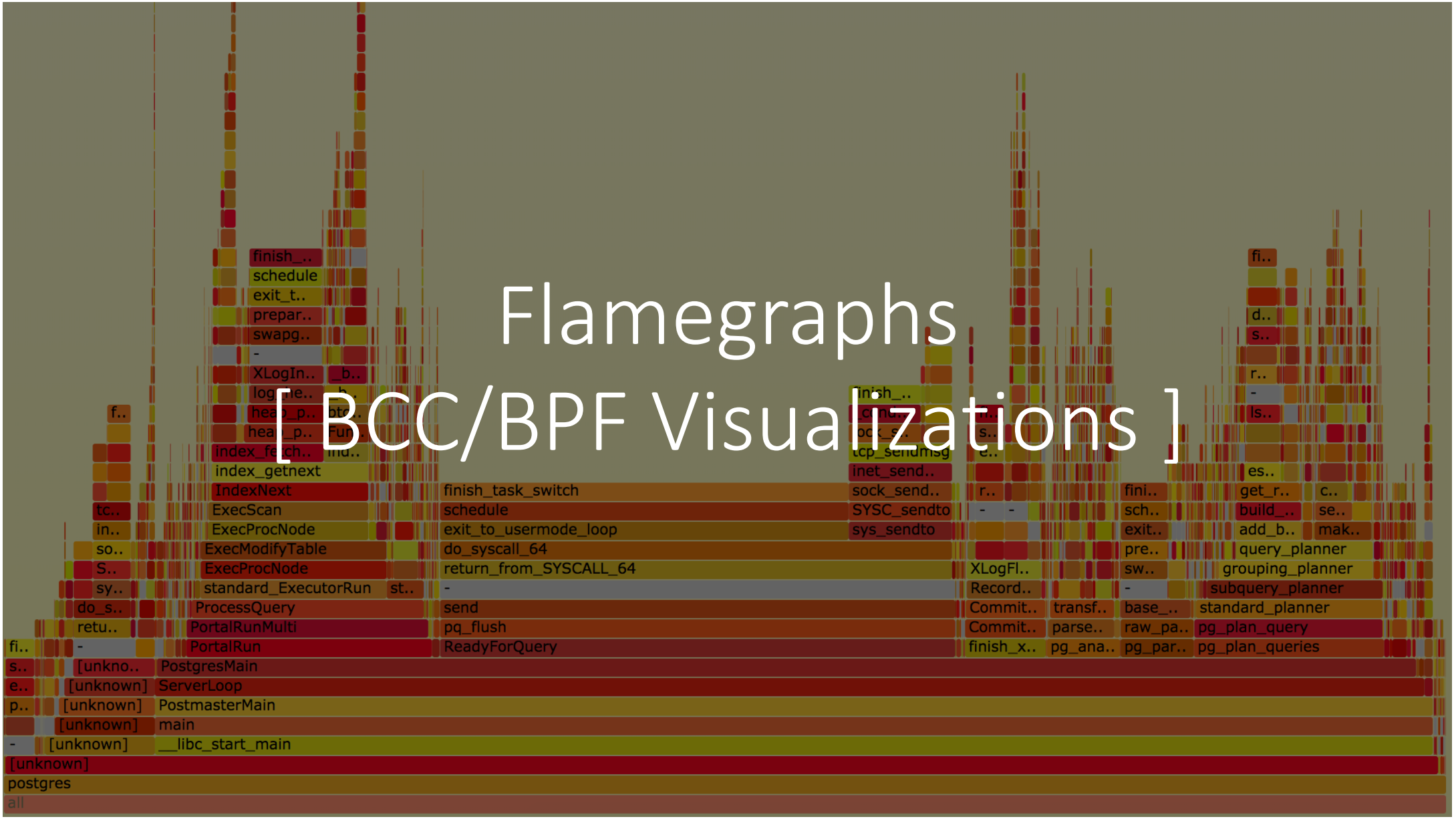
```
# ./runqlat
Tracing run queue latency... Hit Ctrl-C to end.
^C
     usecs               : count     distribution
         0 -> 1          : 233      |***********                             |
         2 -> 3          : 742      |************************************    |
         4 -> 7          : 203      |**********                              |
         8 -> 15         : 173      |********                                |
        16 -> 31         : 24       |*                                       |
        32 -> 63         : 0        |                                        |
        64 -> 127        : 30       |*                                       |
       128 -> 255        : 6        |                                        |
       256 -> 511        : 3        |                                        |
       512 -> 1023       : 5        |                                        |
      1024 -> 2047       : 27       |*                                       |
      2048 -> 4095       : 30       |*                                       |
      4096 -> 8191       : 20       |                                        |
      8192 -> 16383      : 29       |*                                       |
     16384 -> 32767      : 809      |****************************************|
     32768 -> 65535      : 64       |***                                     |
```

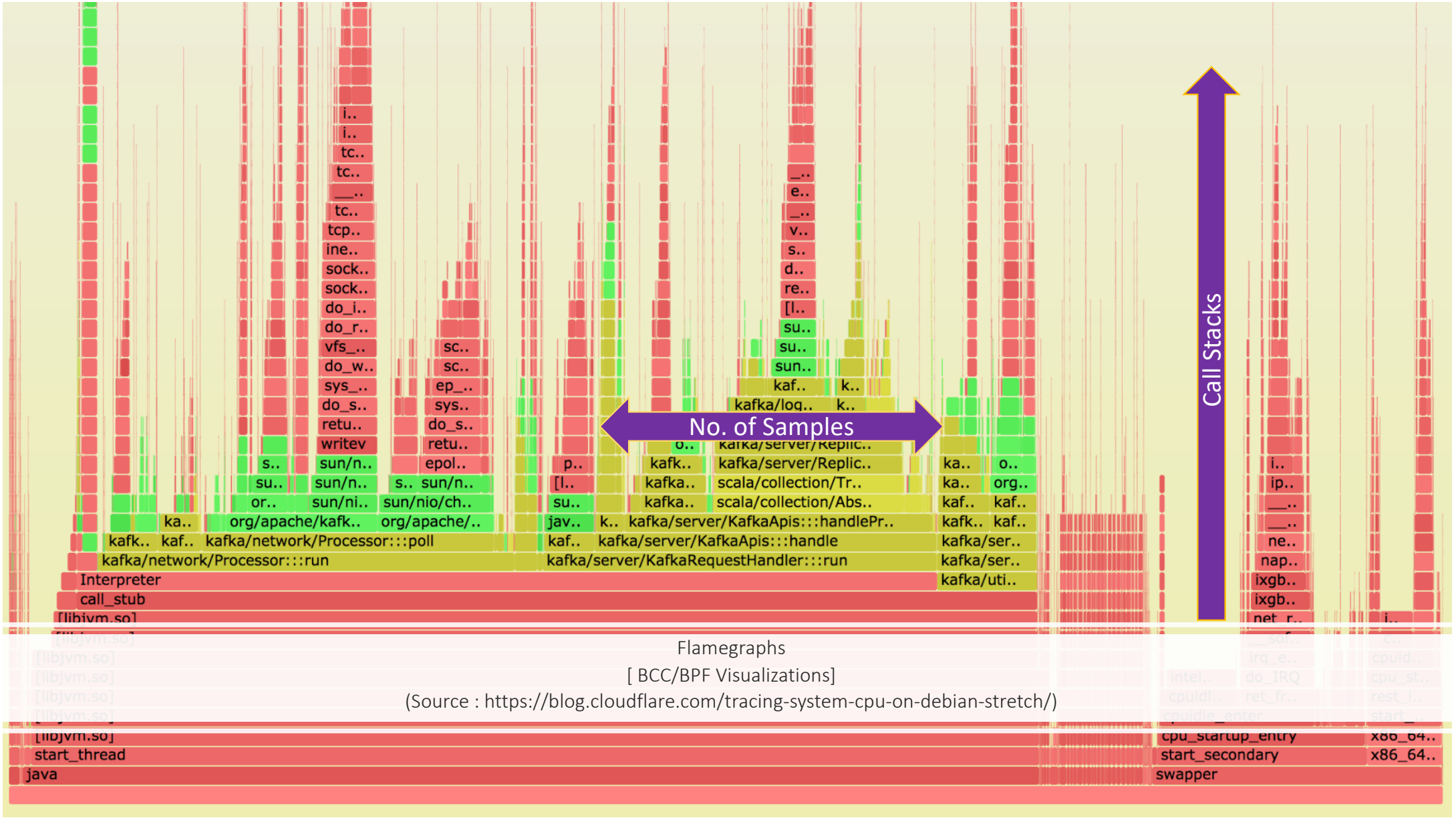
BCC Tools [examples...]

```
# ./profile 1 -d
Sampling at 49 Hertz of all threads by user + kernel stack for 1 secs.

    kmalloc_slab
    __kmalloc_reserve.isra.40
    __alloc_skb
    sk_stream_alloc_skb
    tcp_sendmsg
    inet_sendmsg
    sock_sendmsg
    SYSC_sendto
    sys_sendto
    do_syscall_64
    return_from_SYSCALL_64
    --
    send
    -                pgbench (25279)

    52
```

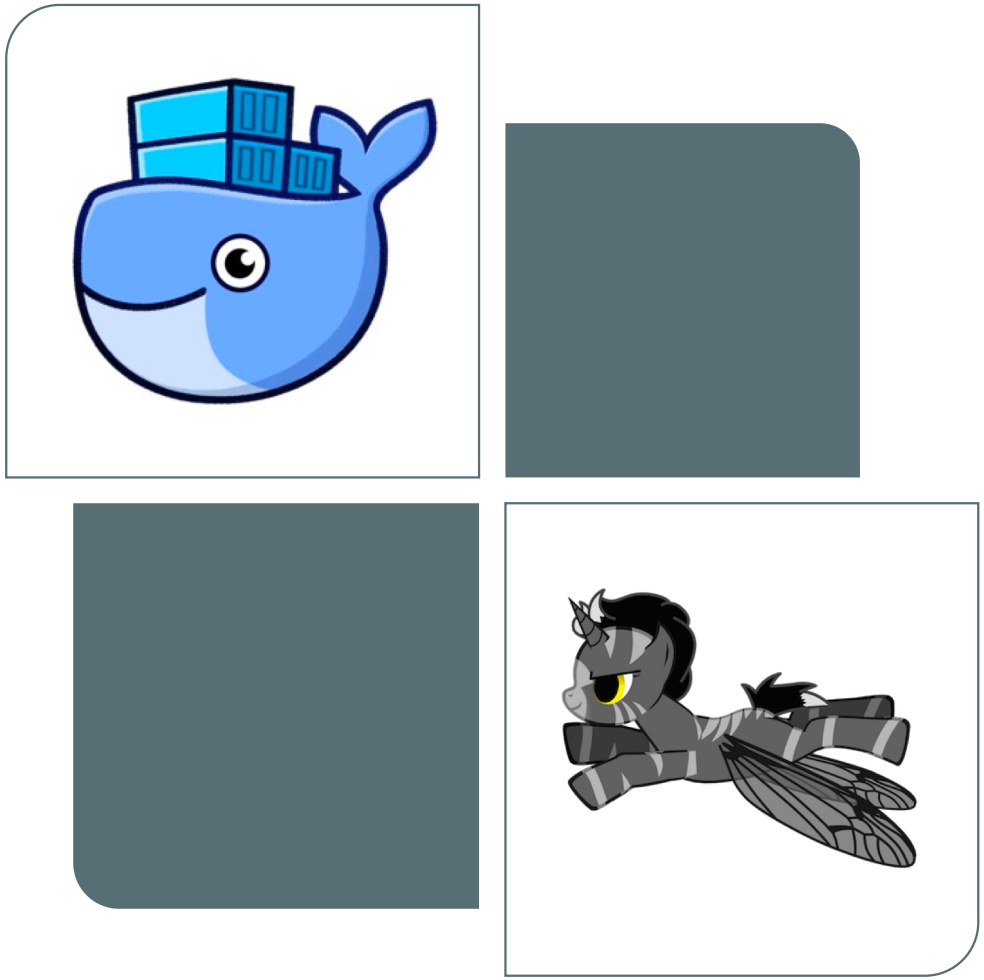

Flamegraphs
[BCC/BPF Visualizations]
finish_..
schedule
exit_t..
prepar..
swapg..
XLogIn..
index_getnext
IndexNext
ExecScan
ExecProcNode
ExecModifyTable
ExecProcNode
standard_ExecutorRun
ProcessQuery
PortalRunMulti
PortalRun
finish_task_switch
schedule
exit_to_usermode_loop
do_syscall_64
return_from_SYSCALL_64
send
pq_flush
ReadyForQuery
inet_send..
sock_send..
SYSC_sendto
sys_sendto
XLogFl..
Record..
Commit..
Commit..
finish_x..
transf..
parse..
pg_ana..
fini..
sch..
exit..
pre..
sw..
base_..
raw_pa..
pg_par..
get_r..
build_..
add_b..
query_planner
grouping_planner
subquery_planner
standard_planner
pg_plan_query
pg_plan_queries
PostgresMain
ServerLoop
PostmasterMain
main
__libc_start_main
[unknown]
postgres
all



Flamegraphs

[BCC/BPF Visualizations]

(Source : https://blog.cloudflare.com/tracing-system-cpu-on-debian-stretch/)



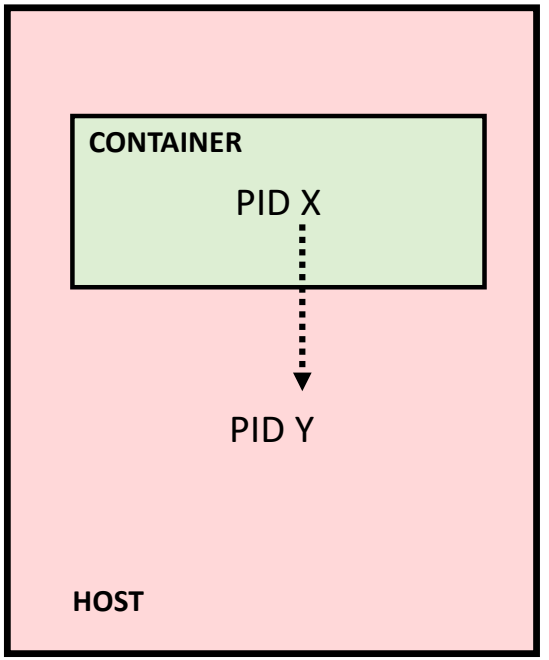
BPF and Containers

BPF and containers…

- Namespaces

Restricts Visibility

- mnt
- pid
- net
- . . .

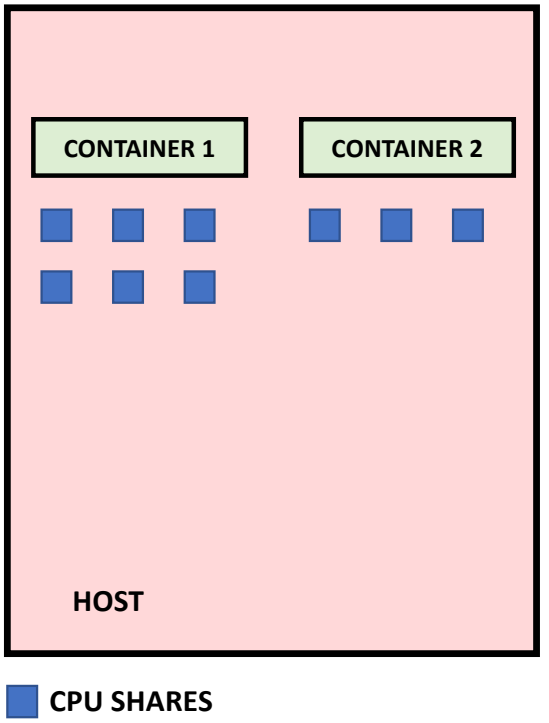


BPF and containers...

- Namespaces
- CGroups

Restricts Quota/Usage

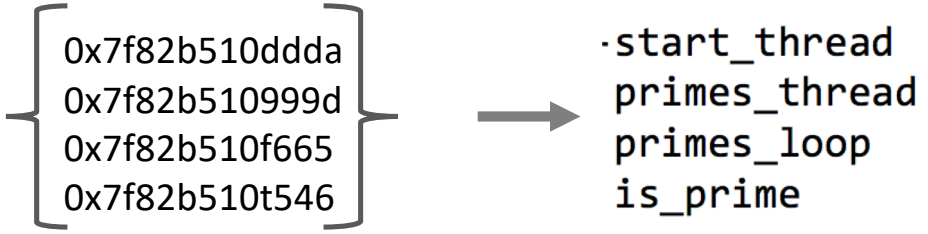
- cpu
- mem
- blkio
- . . .



BPF and containers…

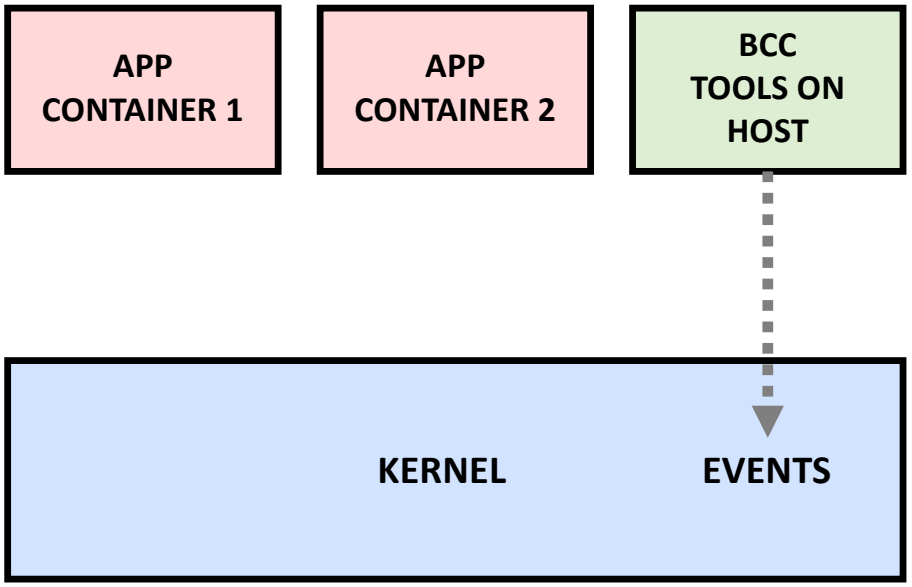
- Namespaces
- CGroups
- Analysis from the host
  - PID Mappings (/sys/fs/cgroup/docker/*)
  - Symbol file locations

```
7f2cd1108880 1e8 Ljava/lang/System;::arraycopy
7f2cd1108c00 200 Ljava/lang/String;::hashCode
7f2cd1109120 2e0 Ljava/lang/String;::indexOf
7f2cd1109740 1c0 Ljava/lang/String;::charAt
```



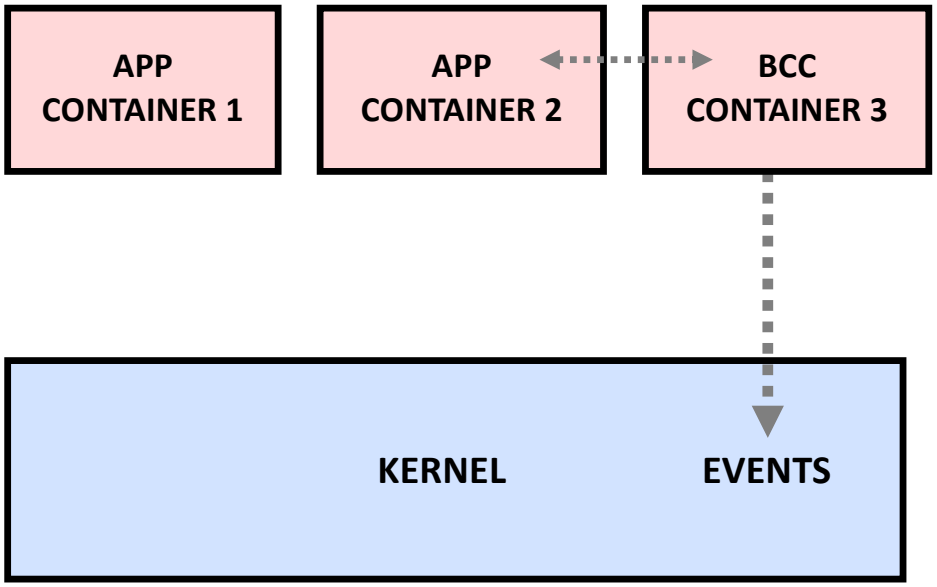
BPF and containers…

- Namespaces
- CGroups
- Analysis from the host
  - PID Mappings (/sys/fs/cgroup*)
  - Symbol file locations
- Deployment Methodologies



BPF and containers…

- Namespaces
- CGroups
- Analysis from the host
  - PID Mappings (/sys/fs/cgroup*)
  - Symbol file locations
- Deployment Methodologies



DEMO

NETWORKING | **OBSERVABILITY** | **SECURITY**

BPF Implementations…

- Seccomp
  - Control system calls made by a process
- Cilium
  - Controls Networking, Security and Load Balancing for containers
- Weavescope
  - Observability into containerized application stacks like Docker and Kubernetes
- Iptables
  - Bpfilter implementations to optimize ingress/outgress security rules
- Systemtap
  - BPF backend for optimizations

References

https://github.com/iovisor/bcc
http://man7.org/linux/man-pages/man2/bpf.2.html
http://brendangregg.com/ebpf.html
https://github.com/goldshtn/linux-tracing-workshop
https://suchakra.wordpress.com/ - eBPF
https://blog.yadutaf.fr/ - Networking & eBPF
https://jvns.ca/blog/2017/07/05/linux-tracing-systems/
https://www.youtube.com/watch?v=aaTQM7wcmfk – Kernel Meetup | eBPF
https://cilium.io/blog/2018/04/17/why-is-the-kernel-community-replacing-iptables/
https://blog.yadutaf.fr/2016/03/30/turn-any-syscall-into-event-introducing-ebpf-kernel-probes/
https://lwn.net/Articles/740157/ - Thorough eBPF intro
https://developers.redhat.com/blog/2017/12/13/introducing-stapbpf-systemtaps-new-bpf-backend/
https://lwn.net/Articles/747551/ - BPF comes to firewalls

@Follow

Sasha Goldshtein (goldshtn)

Brendan Gregg (brendangregg)

Suchakra (tuxology)

Julia Evans (b0rk)

Thank You !


akshay.kapoor@sap.com


akskap


akskap


akskap