PyCon China 2020

Python For Good

Debug Python container with eBPF

张晋涛

网易有道资深运维开发

Who am I

- Container
- Docker
- eBPF
- Kubernetes
- Linux
- Python

Agenda

- How to debug Python code
- How to check system performance
- What is eBPF
- Debug Python container with eBPF

How to debug Python code

How to debug Python code

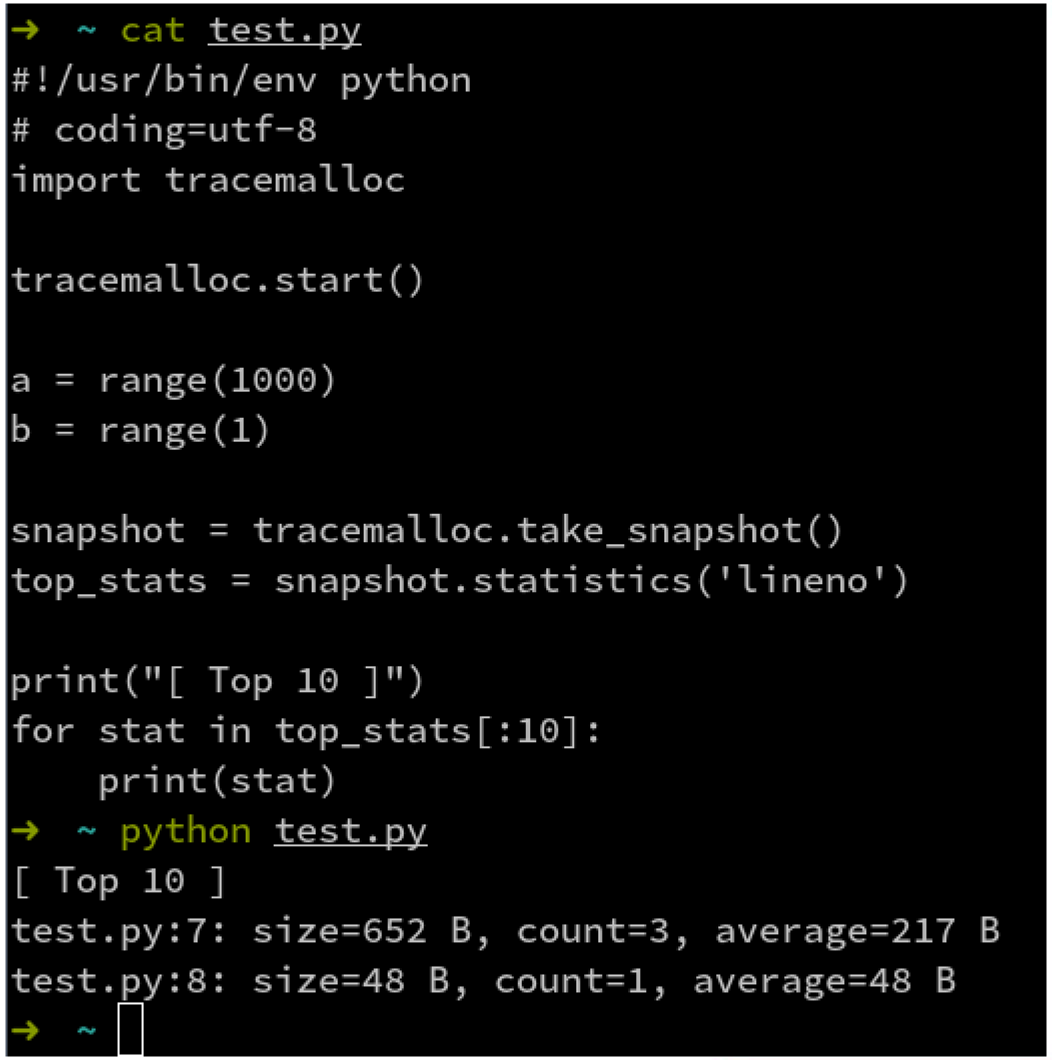
- print
- logging

How to debug Python code

- profile & timeit
- bdb & pdb
- trace
- ...

How to debug Python code

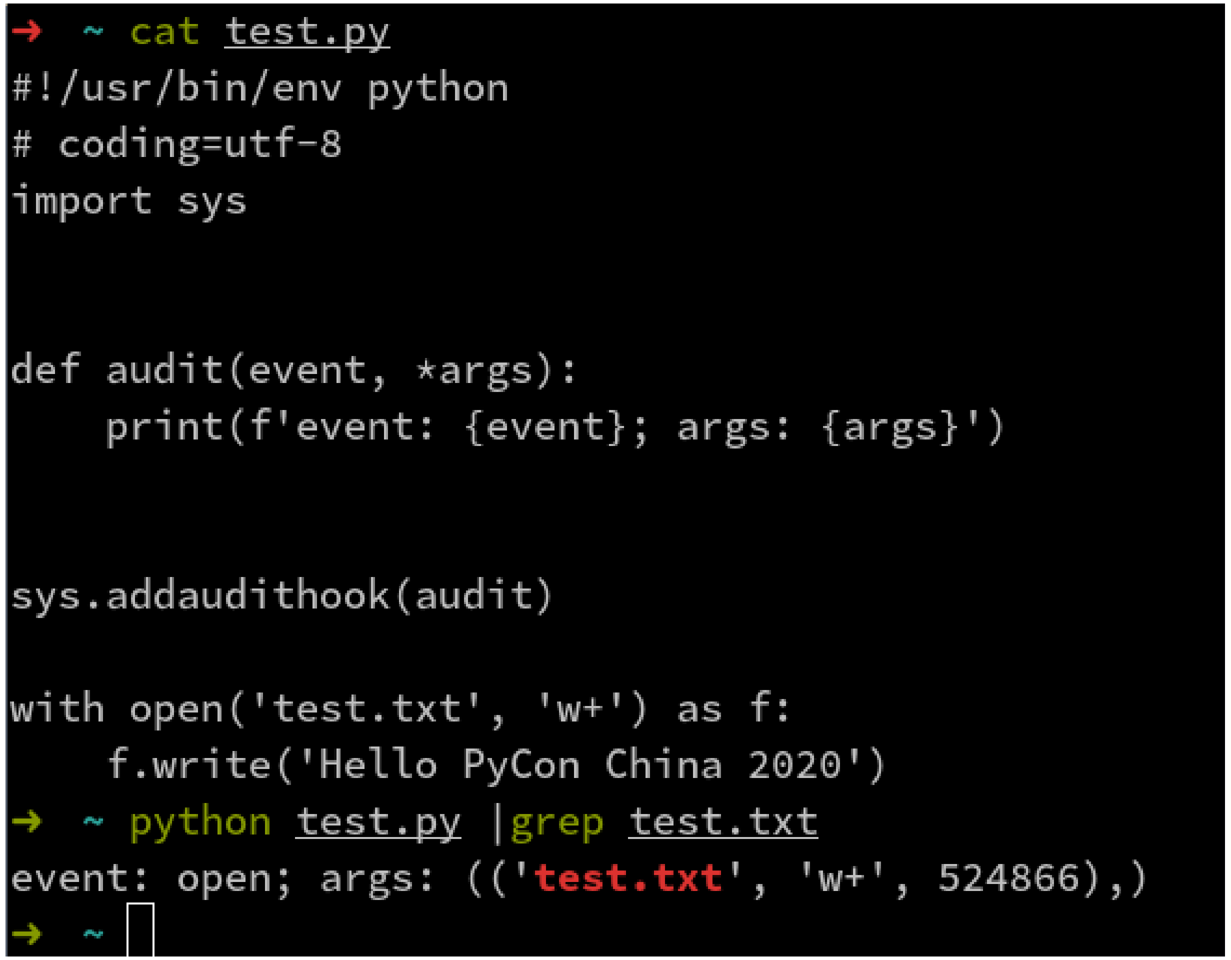
- tracemalloc
  - Since Python 3.4
  - PEP 454 – trace Python memory allocations



```
➜  ~ cat test.py
#!/usr/bin/env python
# coding=utf-8
import tracemalloc

tracemalloc.start()

a = range(1000)
b = range(1)

snapshot = tracemalloc.take_snapshot()
top_stats = snapshot.statistics('lineno')

print("[ Top 10 ]")
for stat in top_stats[:10]:
    print(stat)
➜  ~ python test.py
[ Top 10 ]
test.py:7: size=652 B, count=3, average=217 B
test.py:8: size=48 B, count=1, average=48 B
➜  ~
```

How to debug Python code

- sys.audit
  - Since Python 3.8
  - PEP 578 – Python Runtime Audit Hooks



```
➜  ~ cat test.py
#!/usr/bin/env python
# coding=utf-8
import sys


def audit(event, *args):
    print(f'event: {event}; args: {args}')


sys.addaudithook(audit)

with open('test.txt', 'w+') as f:
    f.write('Hello PyCon China 2020')
➜  ~ python test.py |grep test.txt
event: open; args: (('test.txt', 'w+', 524866),)
➜  ~
```

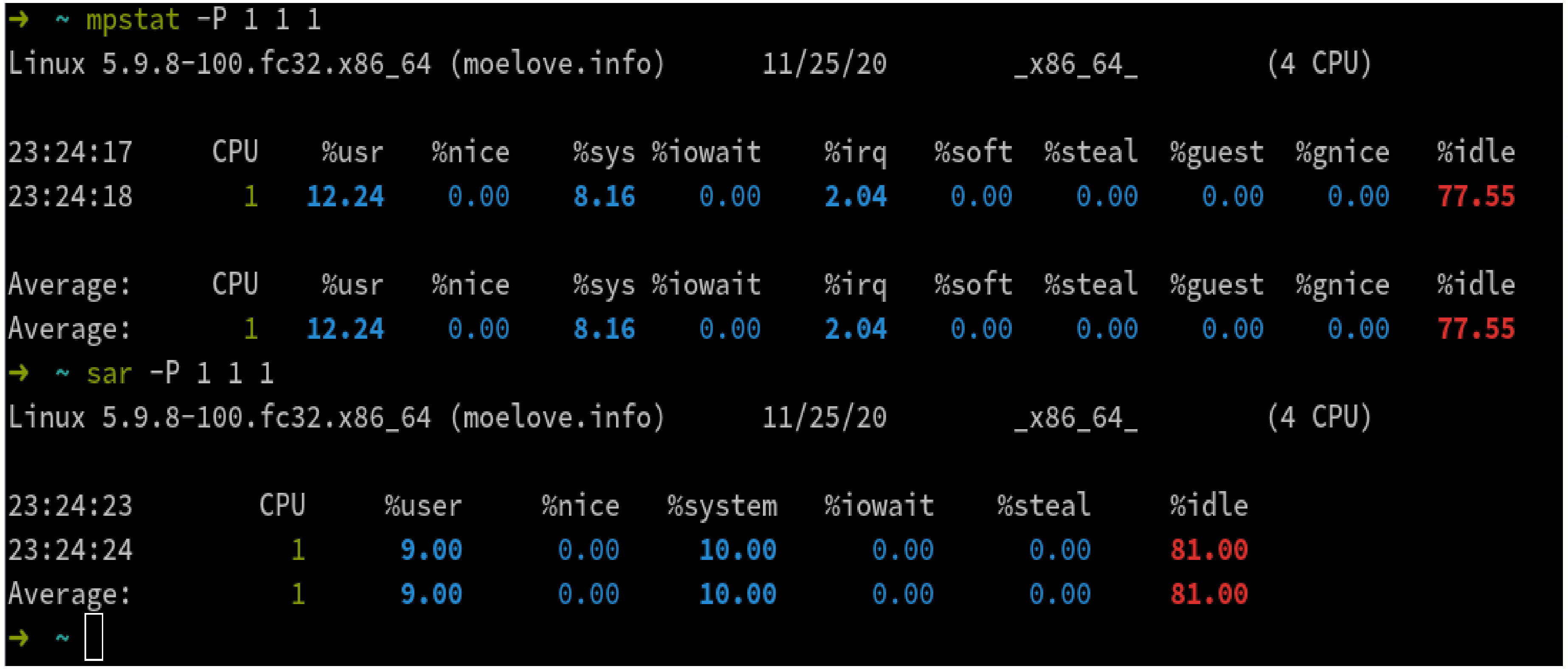
How to check system performance

How to check system performance

- Physical
  - CPU
  - Memory
  - Network
  - Storage I/O
- Software
  - File descriptors
  - mutex

How to check system performance

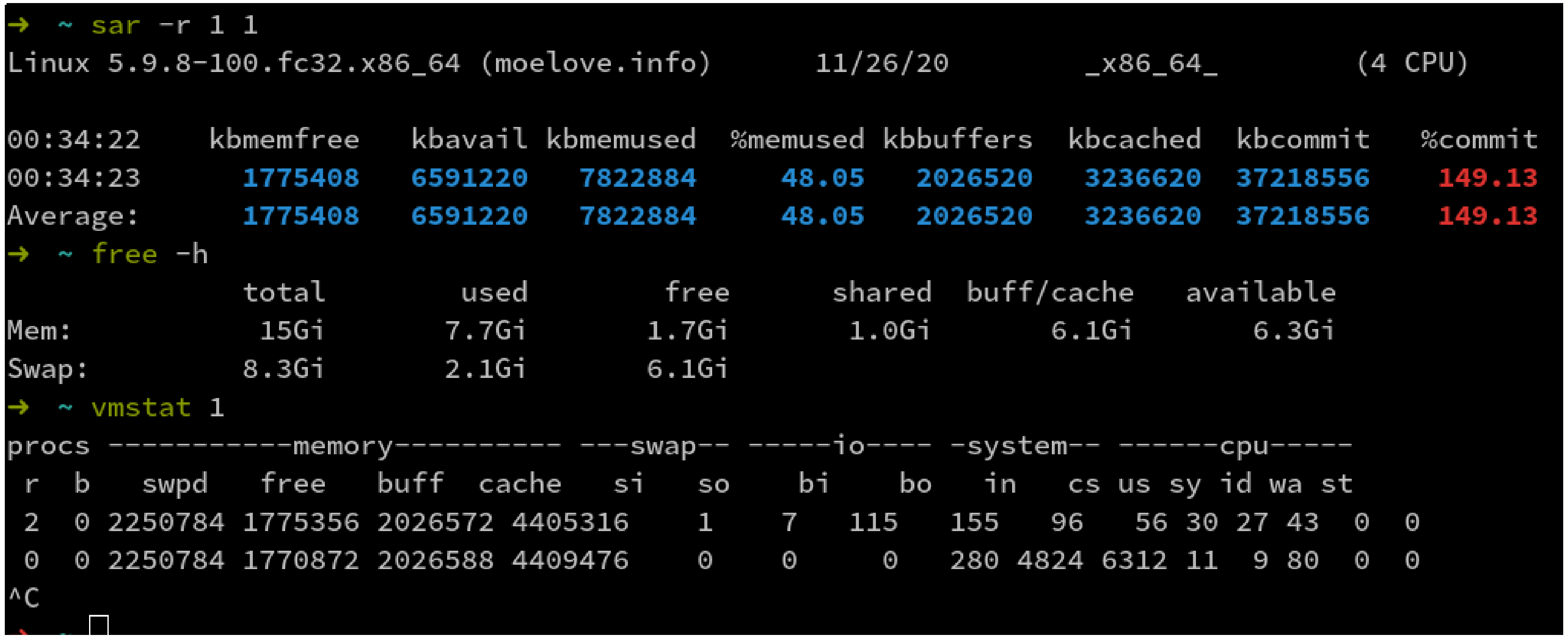
- CPU
  - sar
  - mpstat
  - top
  - ps



```
➜  ~ mpstat -P 1 1 1
Linux 5.9.8-100.fc32.x86_64 (moelove.info)     11/25/20        _x86_64_        (4 CPU)

23:24:17     CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
23:24:18       1   12.24    0.00    8.16    0.00    2.04    0.00    0.00    0.00    0.00   77.55

Average:     CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice   %idle
Average:       1   12.24    0.00    8.16    0.00    2.04    0.00    0.00    0.00    0.00   77.55
➜  ~ sar -P 1 1 1
Linux 5.9.8-100.fc32.x86_64 (moelove.info)     11/25/20        _x86_64_        (4 CPU)

23:24:23        CPU     %user     %nice   %system   %iowait    %steal     %idle
23:24:24          1      9.00      0.00     10.00      0.00      0.00     81.00
Average:          1      9.00      0.00     10.00      0.00      0.00     81.00
➜  ~
```

How to check system performance

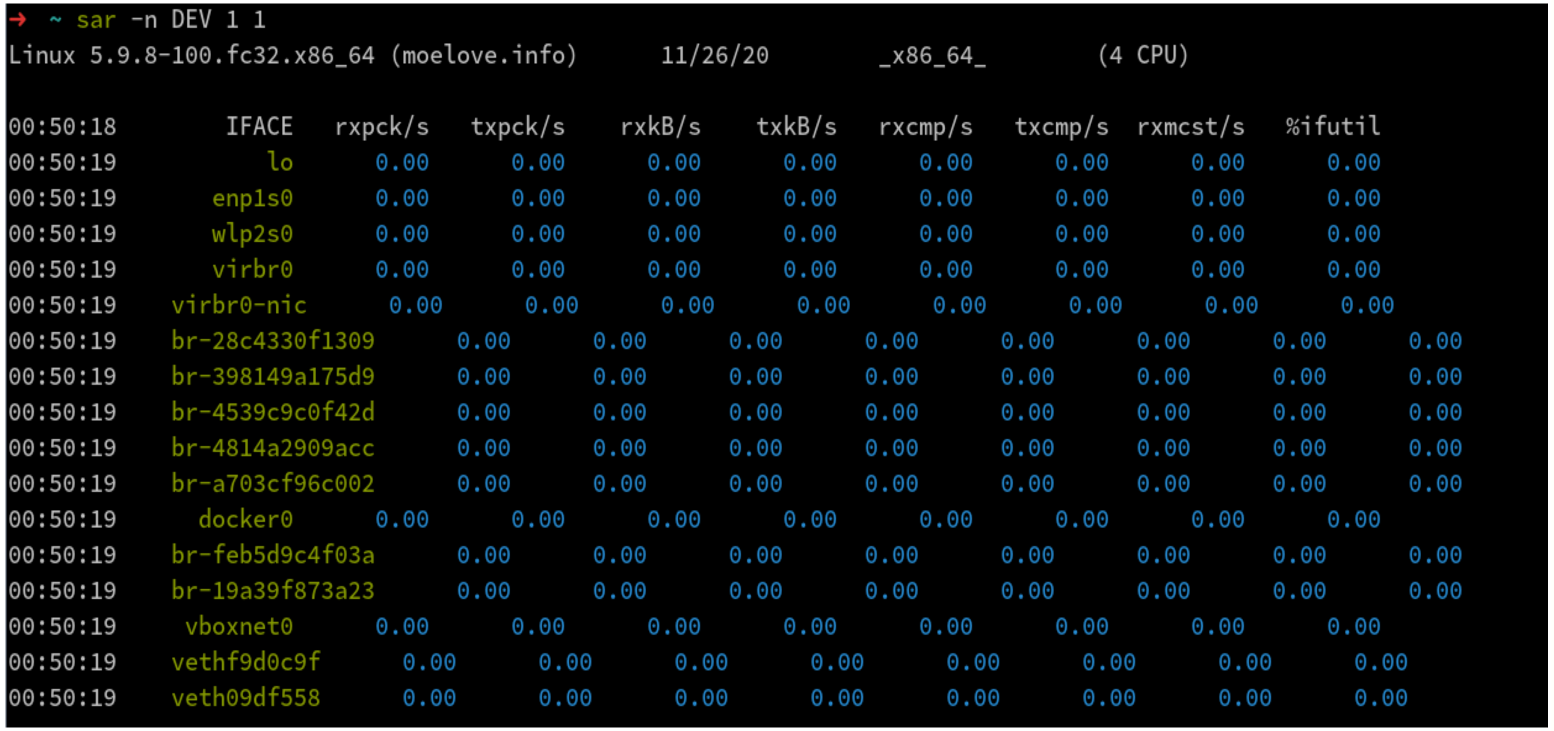
- Memory
  - sar
  - free
  - vmstat
  - top/htop



```
➜ ~ sar -r 1 1
Linux 5.9.8-100.fc32.x86_64 (moelove.info)     11/26/20        _x86_64_        (4 CPU)

00:34:22    kbmemfree   kbavail kbmemused  %memused kbbuffers  kbcached  kbcommit   %commit
00:34:23      1775408   6591220   7822884     48.05   2026520   3236620  37218556    149.13
Average:      1775408   6591220   7822884     48.05   2026520   3236620  37218556    149.13
➜ ~ free -h
              total        used        free      shared  buff/cache   available
Mem:           15Gi       7.7Gi       1.7Gi       1.0Gi       6.1Gi       6.3Gi
Swap:         8.3Gi       2.1Gi       6.1Gi
➜ ~ vmstat 1
procs -----------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
 2  0 2250784 1775356 2026572 4405316    1    7   115   155   96   56 30 27 43  0  0
 0  0 2250784 1770872 2026588 4409476    0    0     0   280 4824 6312 11  9 80  0  0
^C
```

How to check system performance

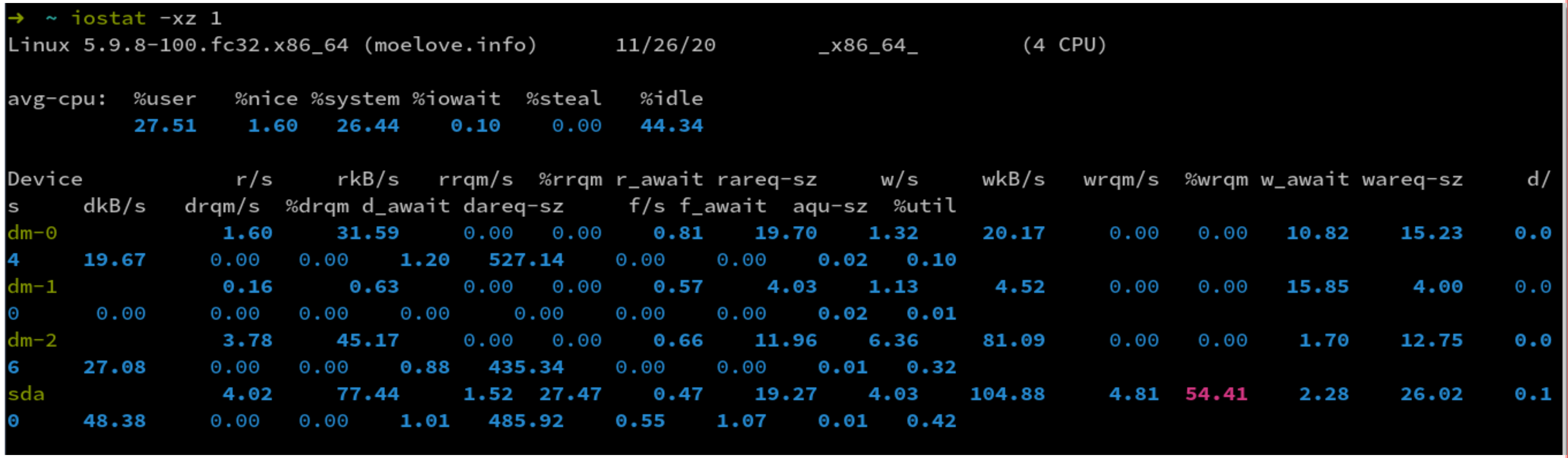
- Network
  - top/iftop
  - ifconfig
  - iproute2



```
➜ ~ sar -n DEV 1 1
Linux 5.9.8-100.fc32.x86_64 (moelove.info)      11/26/20        _x86_64_        (4 CPU)

00:50:18        IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s   %ifutil
00:50:19           lo      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:50:19       enp1s0      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:50:19       wlp2s0      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:50:19       virbr0      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:50:19    virbr0-nic      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:50:19    br-28c4330f1309      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:50:19    br-398149a175d9      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:50:19    br-4539c9c0f42d      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:50:19    br-4814a2909acc      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:50:19    br-a703cf96c002      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:50:19      docker0      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:50:19    br-feb5d9c4f03a      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:50:19    br-19a39f873a23      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:50:19     vboxnet0      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:50:19    vethf9d0c9f      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
00:50:19    veth09df558      0.00      0.00      0.00      0.00      0.00      0.00      0.00      0.00
```

How to check system performance

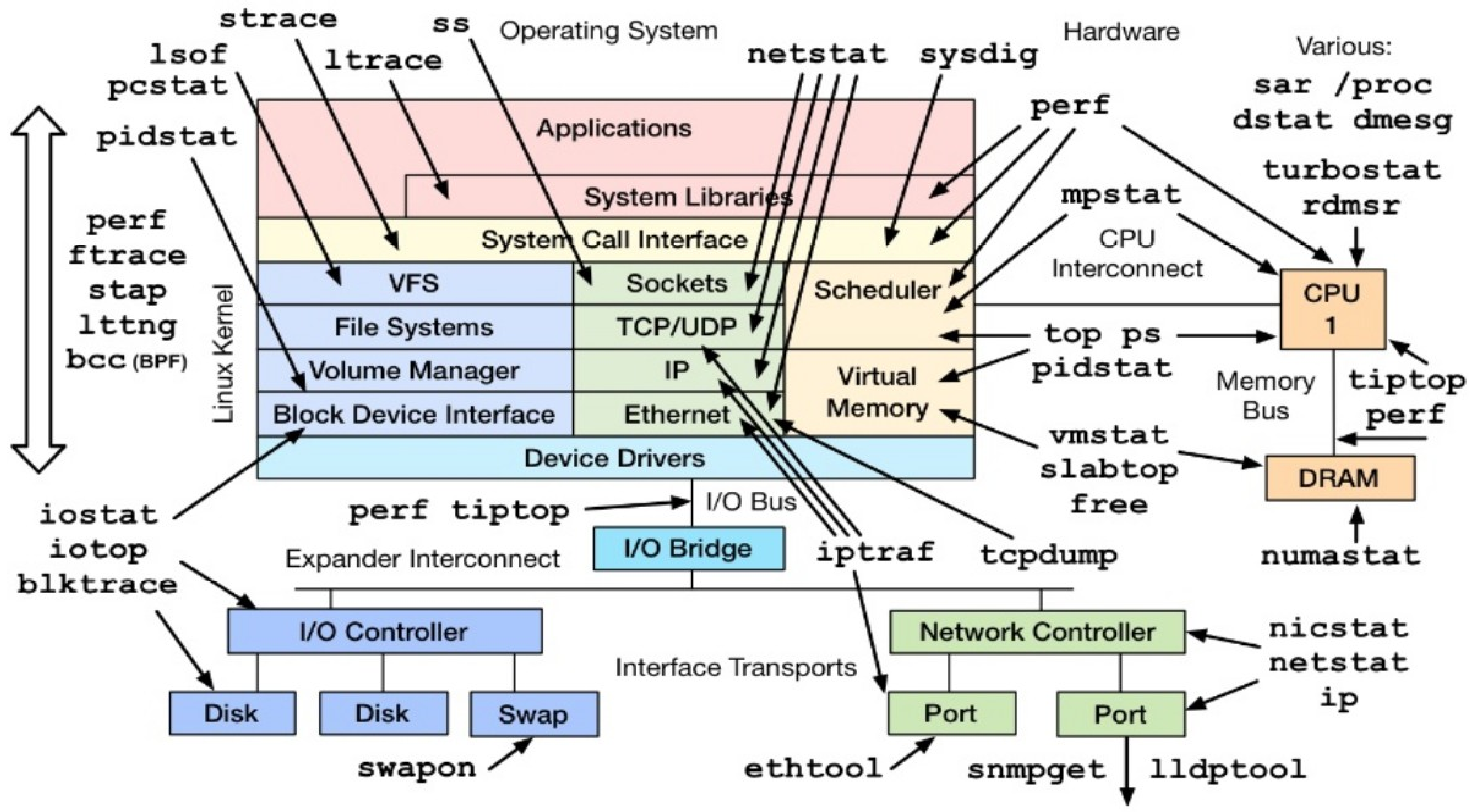
- Storage
  - sar
  - iostat
  - iotop



```
➜ ~ iostat -xz 1
Linux 5.9.8-100.fc32.x86_64 (moelove.info)      11/26/20        _x86_64_        (4 CPU)

avg-cpu:  %user   %nice %system %iowait  %steal   %idle
          27.51    1.60   26.44    0.10    0.00   44.34

Device            r/s     rkB/s   rrqm/s  %rrqm r_await rareq-sz     w/s     wkB/s   wrqm/s  %wrqm w_await wareq-sz     d/
s     dkB/s   drqm/s  %drqm d_await dareq-sz     f/s f_await  aqu-sz  %util
dm-0             1.60     31.59     0.00   0.00    0.81    19.70    1.32     20.17     0.00   0.00   10.82    15.23    0.0
4     19.67     0.00   0.00    1.20   527.14    0.00    0.00    0.02   0.10
dm-1             0.16      0.63     0.00   0.00    0.57     4.03    1.13      4.52     0.00   0.00   15.85     4.00    0.0
0      0.00     0.00   0.00    0.00     0.00    0.00    0.00    0.02   0.01
dm-2             3.78     45.17     0.00   0.00    0.66    11.96    6.36     81.09     0.00   0.00    1.70    12.75    0.0
6     27.08     0.00   0.00    0.88   435.34    0.00    0.00    0.01   0.32
sda              4.02     77.44     1.52  27.47    0.47    19.27    4.03    104.88     4.81  54.41    2.28    26.02    0.1
0     48.38     0.00   0.00    1.01   485.92    0.55    1.07    0.01   0.42
```

How to check system performance

Python For Good
PyCon China 2020 | PYTHON 中国开发者大会 2020



http://www.brendangregg.com/linuxperf.html 2017

PyCon China 2020

What is eBPF

What is eBPF

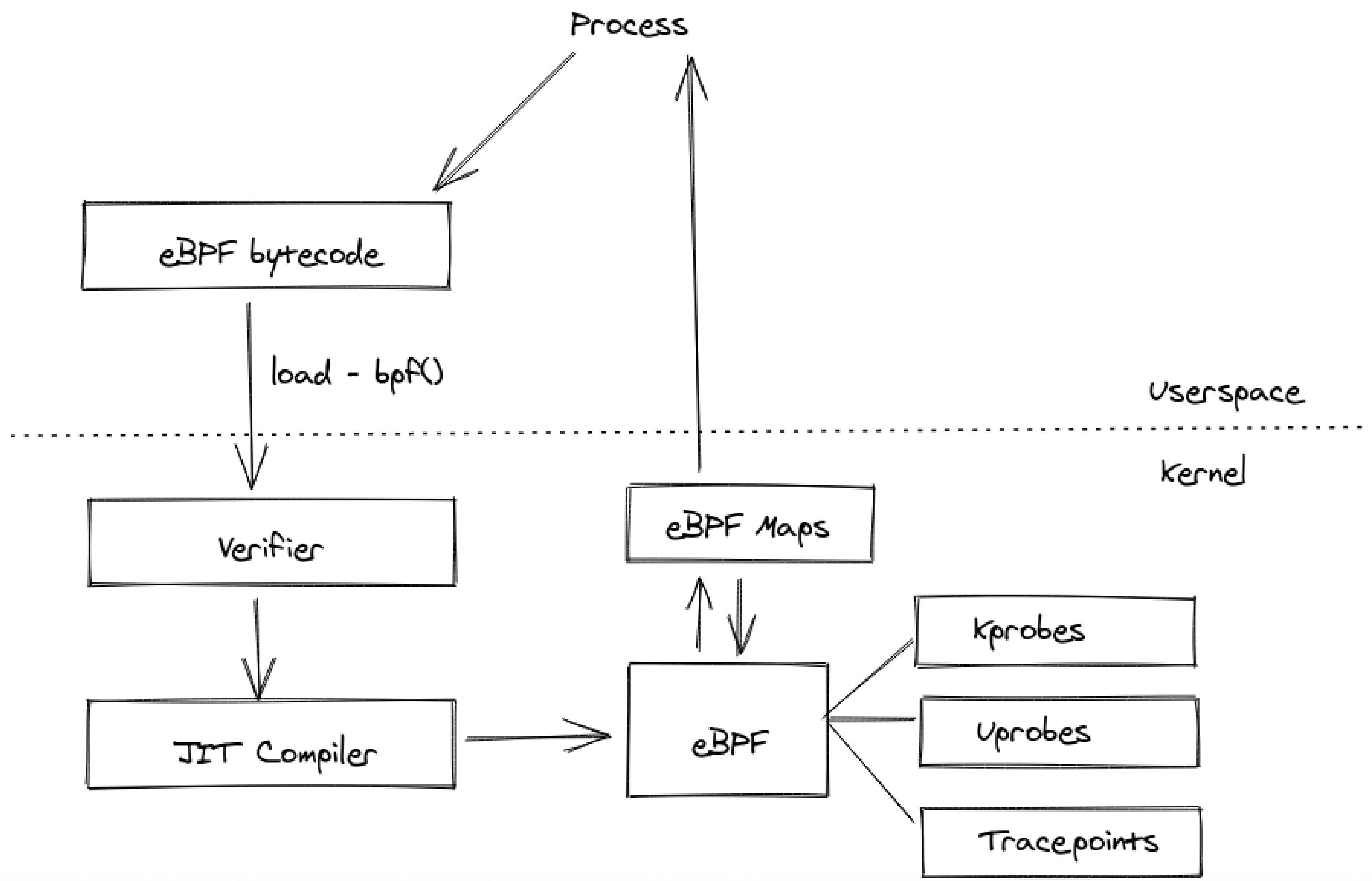
- eBPF is a revolutionary technology that can run sandboxed programs in the Linux kernel without changing kernel source code or loading kernel modules.

  - via:https://ebpf.io/what-is-ebpf/

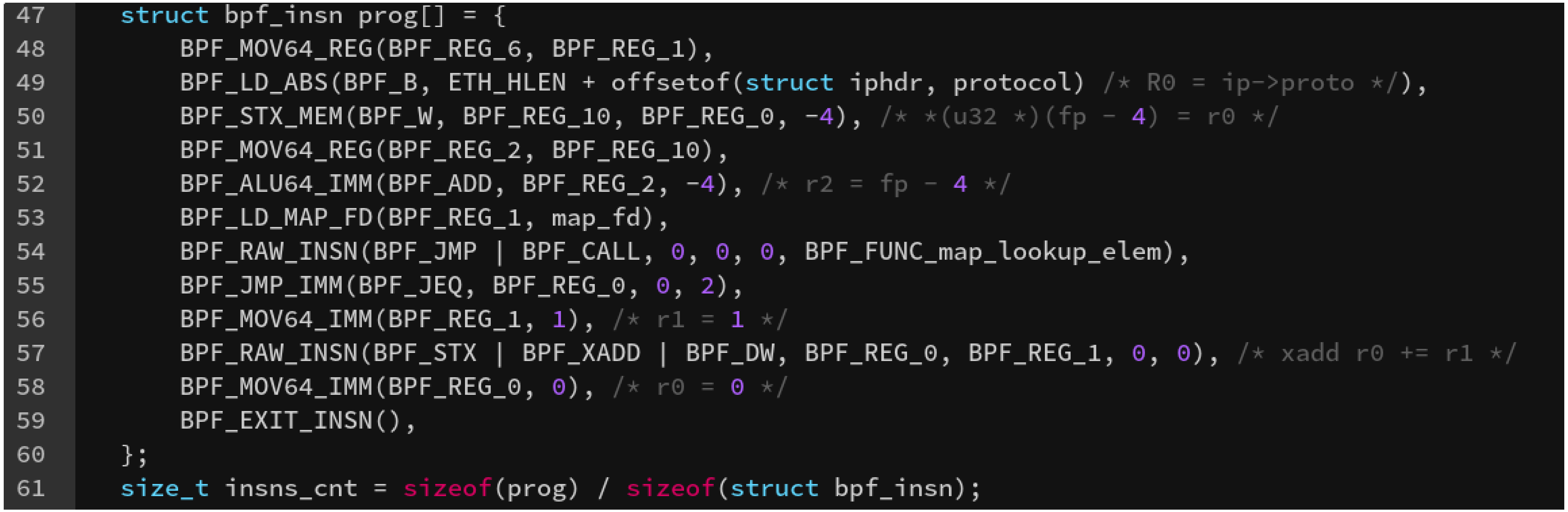
What is eBPF

- enhanced Berkeley Packet Filter
- Use cases:
  - Networking
  - Tracing
  - Security
  - ...

How does eBPF work



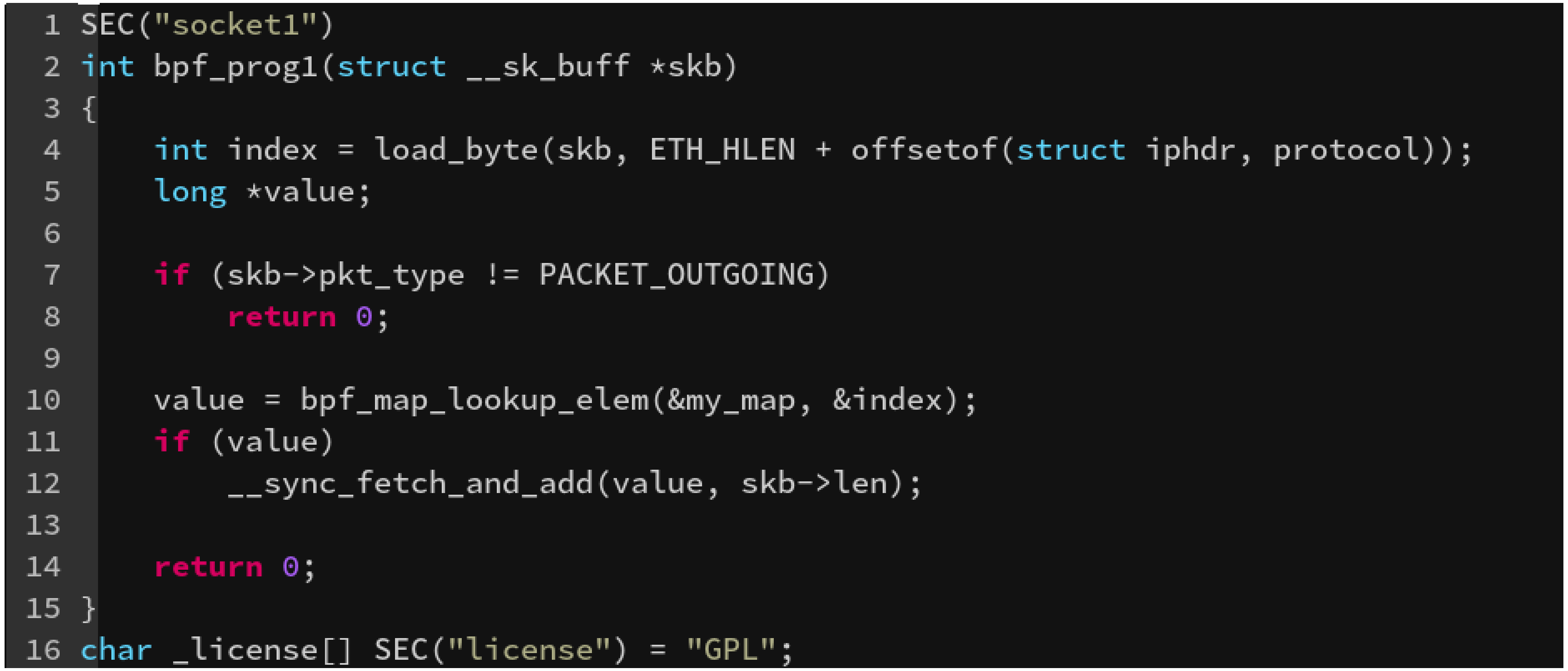
Raw BPF



```
    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(),
    };
    size_t insns_cnt = sizeof(prog) / sizeof(struct bpf_insn);
```

Linux samples/bpf/sock_example.c

LLVM eBPF compiler



```c
SEC("socket1")
int bpf_prog1(struct __sk_buff *skb)
{
    int index = load_byte(skb, ETH_HLEN + offsetof(struct iphdr, protocol));
    long *value;

    if (skb->pkt_type != PACKET_OUTGOING)
        return 0;

    value = bpf_map_lookup_elem(&my_map, &index);
    if (value)
        __sync_fetch_and_add(value, skb->len);

    return 0;
}
char _license[] SEC("license") = "GPL";
```

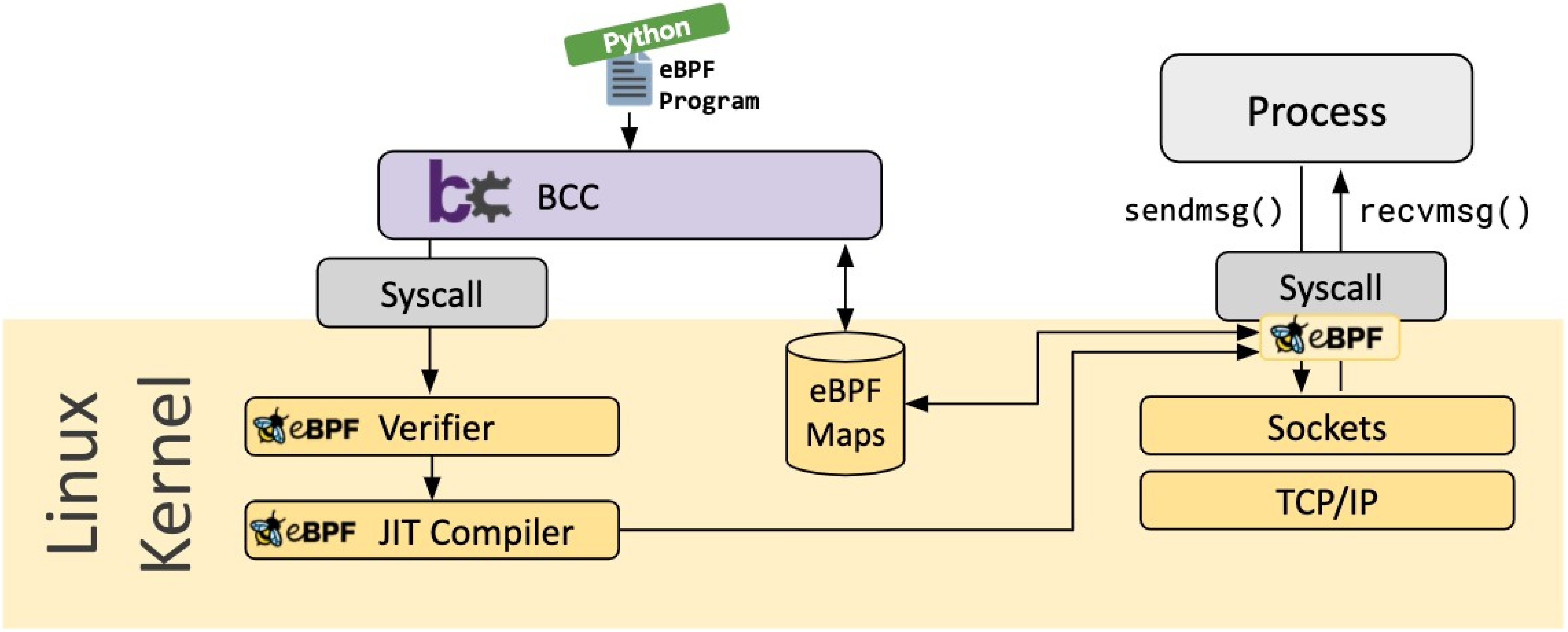
Linux samples/bpf/sockex1_kern.c

BPF Compiler Collection

- BCC proper
- BCC-tools
- Front-ends
  - Python
  - Lua
  - C helper functions

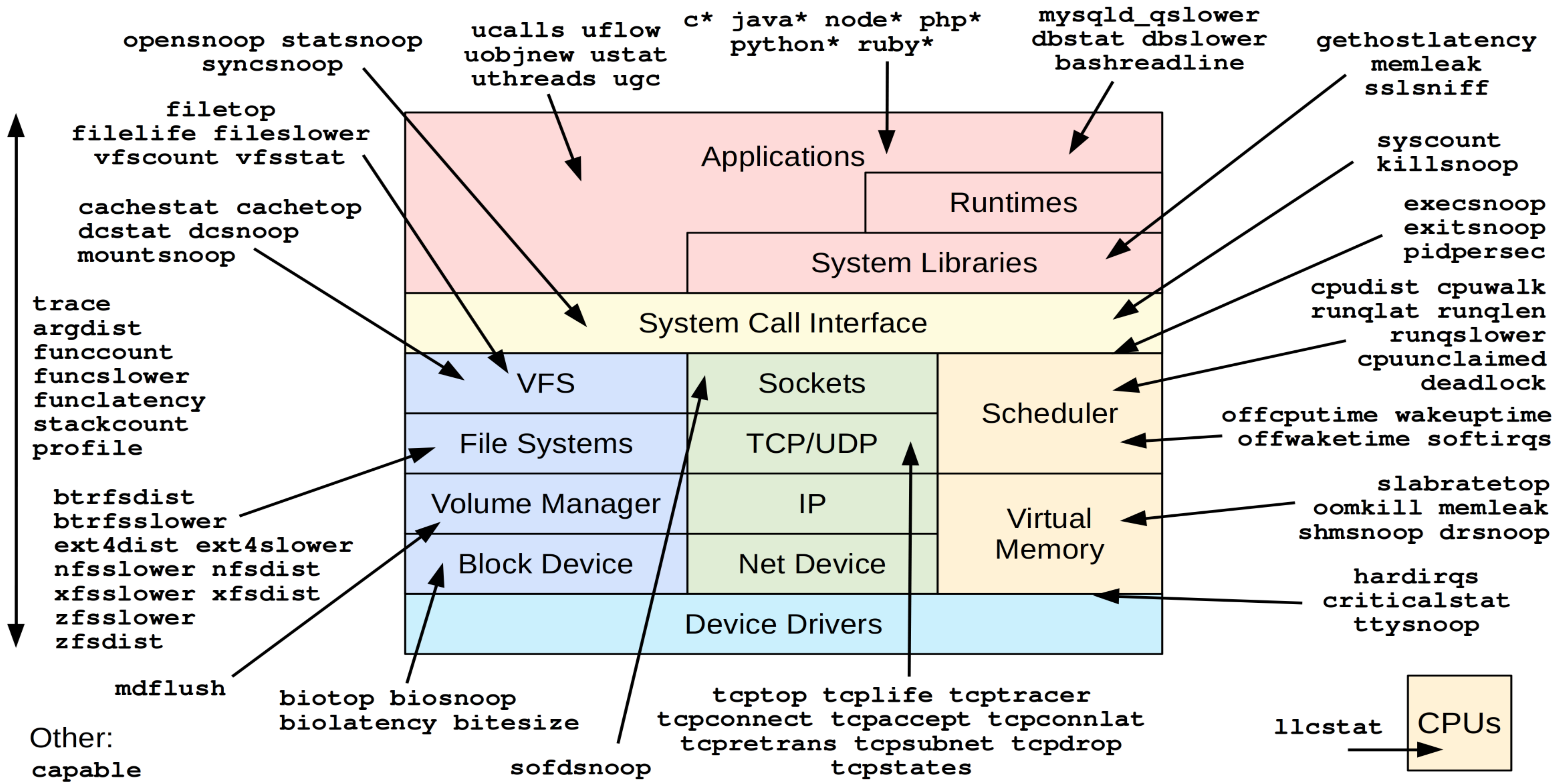
https://github.com/iovisor/bcc

BPF Compiler Collection



https://ebpf.io/what-is-ebpf/#development-toolchains

Linux bcc/BPF Tracing Tools

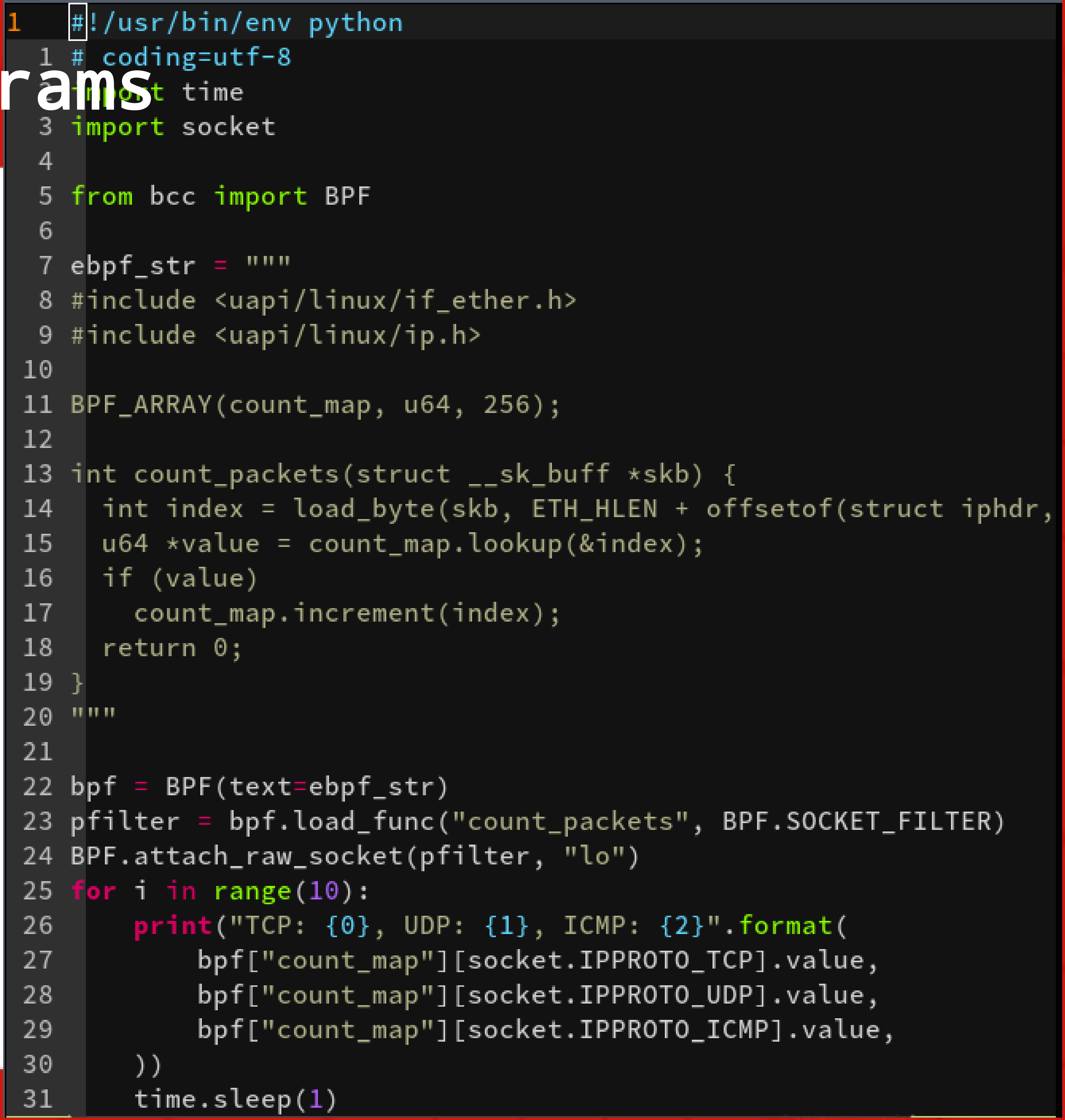


https://github.com/iovisor/bcc#tools 2019

How to write eBPF programs

bcc eBPF compiler

- Backends & data structures
  - “restricted C”
- Frontends & loaders
  - Python/Lua

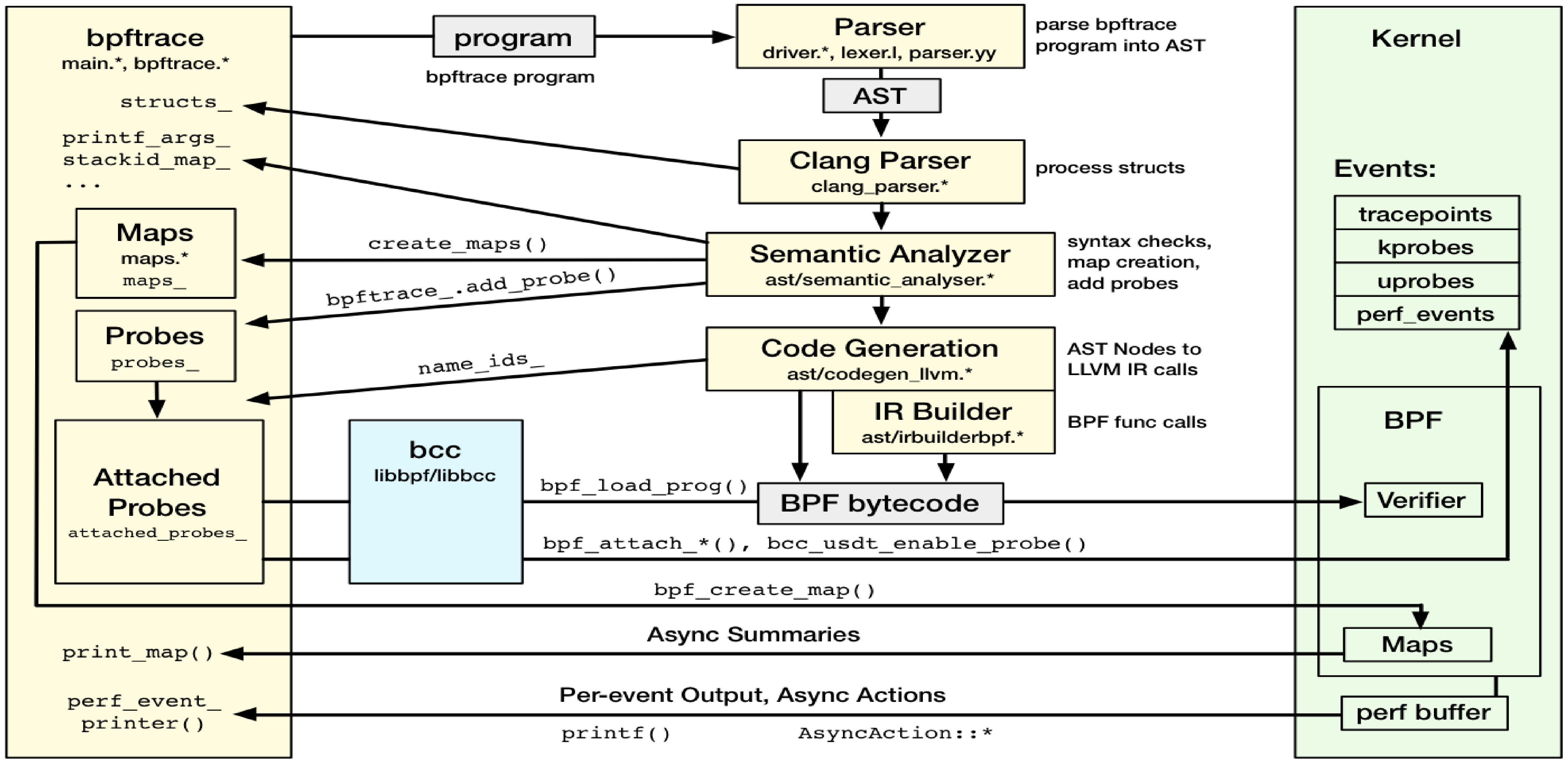


```python
#!/usr/bin/env python
# coding=utf-8
import time
import socket

from bcc import BPF

ebpf_str = """
#include <uapi/linux/if_ether.h>
#include <uapi/linux/ip.h>

BPF_ARRAY(count_map, u64, 256);

int count_packets(struct __sk_buff *skb) {
  int index = load_byte(skb, ETH_HLEN + offsetof(struct iphdr,
  u64 *value = count_map.lookup(&index);
  if (value)
    count_map.increment(index);
  return 0;
}
"""

bpf = BPF(text=ebpf_str)
pfilter = bpf.load_func("count_packets", BPF.SOCKET_FILTER)
BPF.attach_raw_socket(pfilter, "lo")
for i in range(10):
    print("TCP: {0}, UDP: {1}, ICMP: {2}".format(
        bpf["count_map"][socket.IPPROTO_TCP].value,
        bpf["count_map"][socket.IPPROTO_UDP].value,
        bpf["count_map"][socket.IPPROTO_ICMP].value,
    ))
    time.sleep(1)
```

bpftrace (DTrace 2.0)

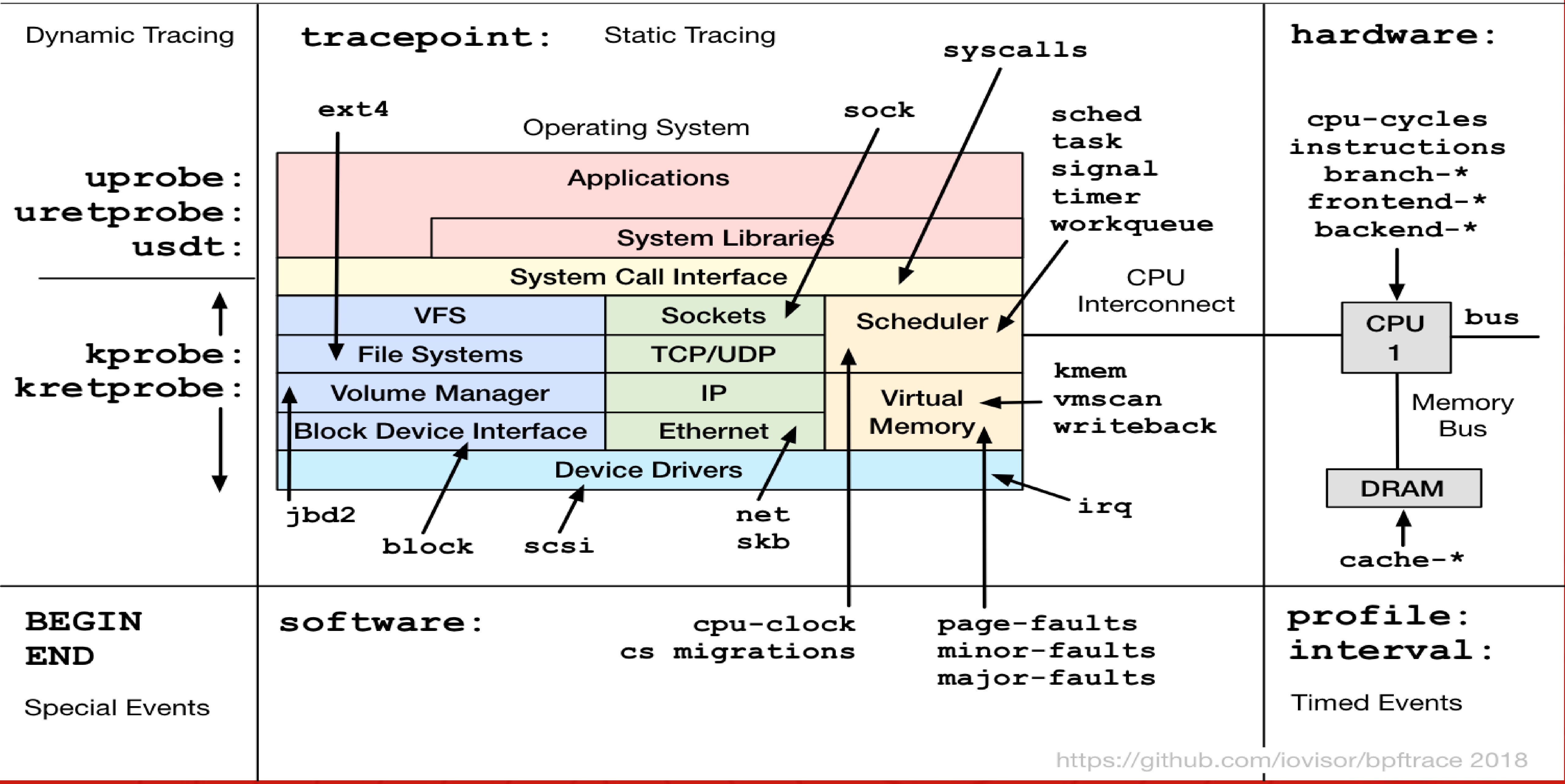
- High-level tracing language
- Using LLVM as a backend to compile scripts to BPF-bytecode
- The bpftrace language is inspired by awk and C

bpftrace Internals



https://github.com/iovisor/bpftrace 2018

bpftrace Probe Types



https://github.com/iovisor/bpftrace 2018

PyCon China 2020 | PYTHON 中国开发者大会 2020

Debug Python container with eBPF

PyCon China 2020

Debug Python container with eBPF

Python For Good
PyCon China 2020 | PYTHON 中国开发者大会 2020

- Container's Namespace
  - docker run --pid container:xxxx
  - nsenter --target $PID --pid

PyCon China 2020

Debug Python container with eBPF

- USDT (--with-pydebug --with-dtrace)
  - pythoncalls.sh
  - pythonflow.sh
  - pythongc.sh
  - pythonstat.sh

THANK YOU