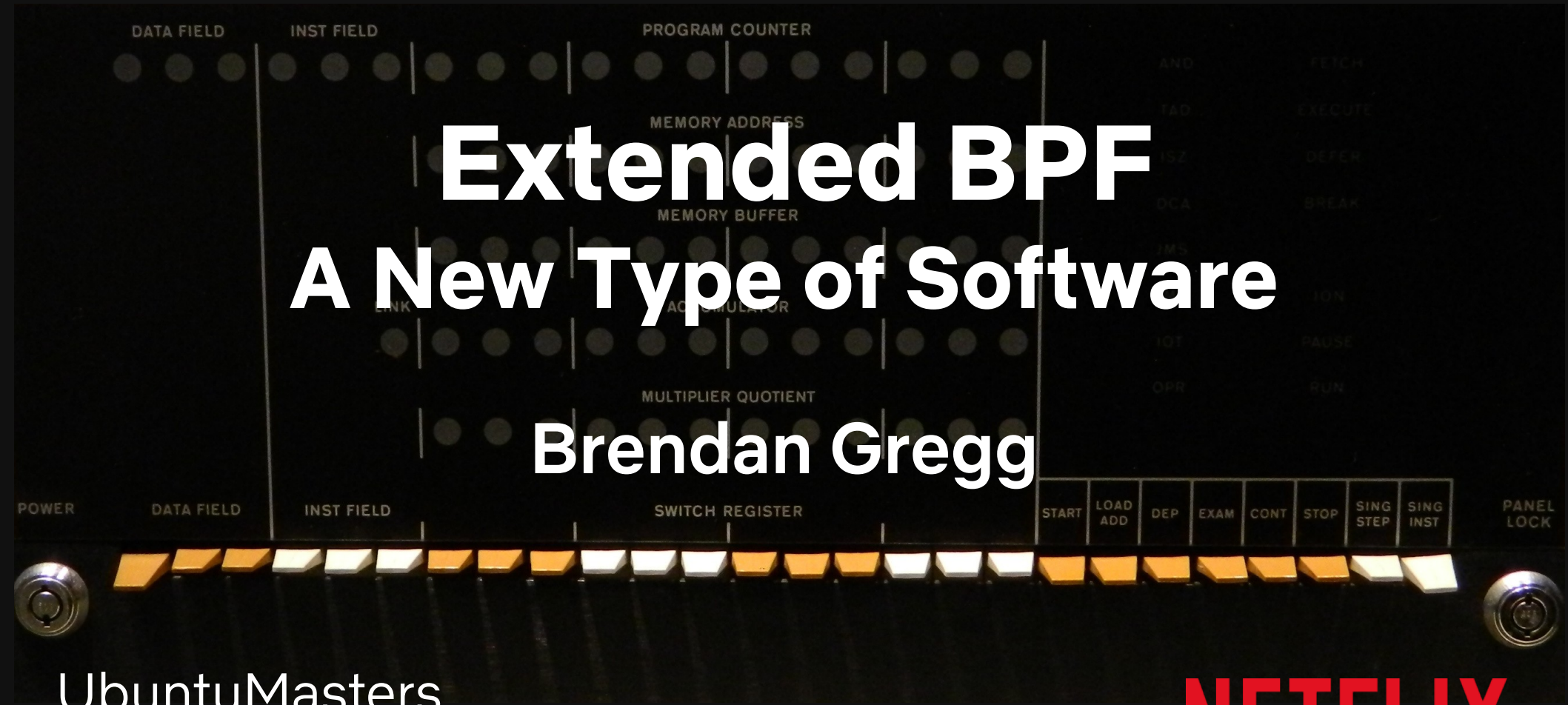
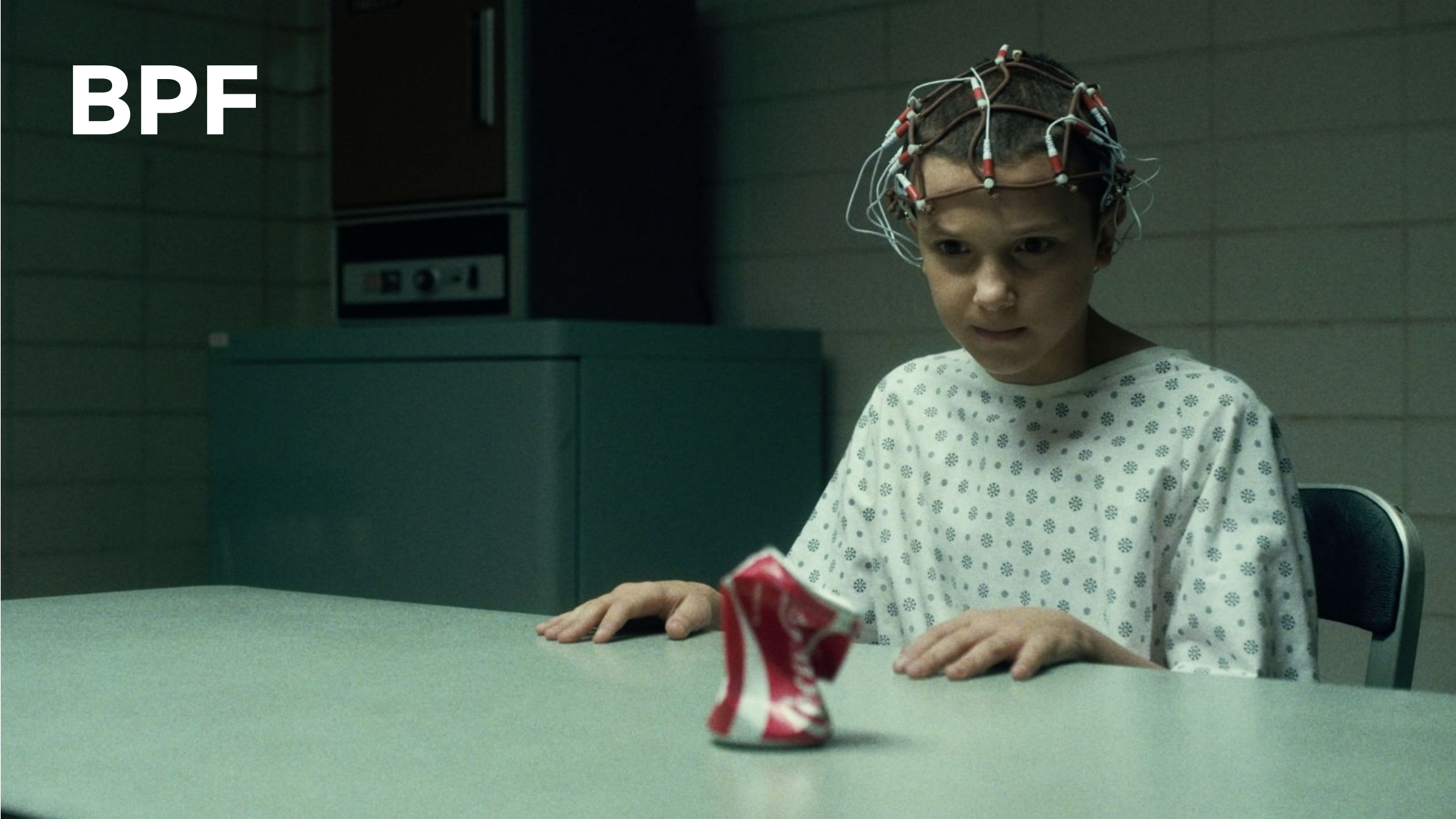


Extended BPF
A New Type of Software

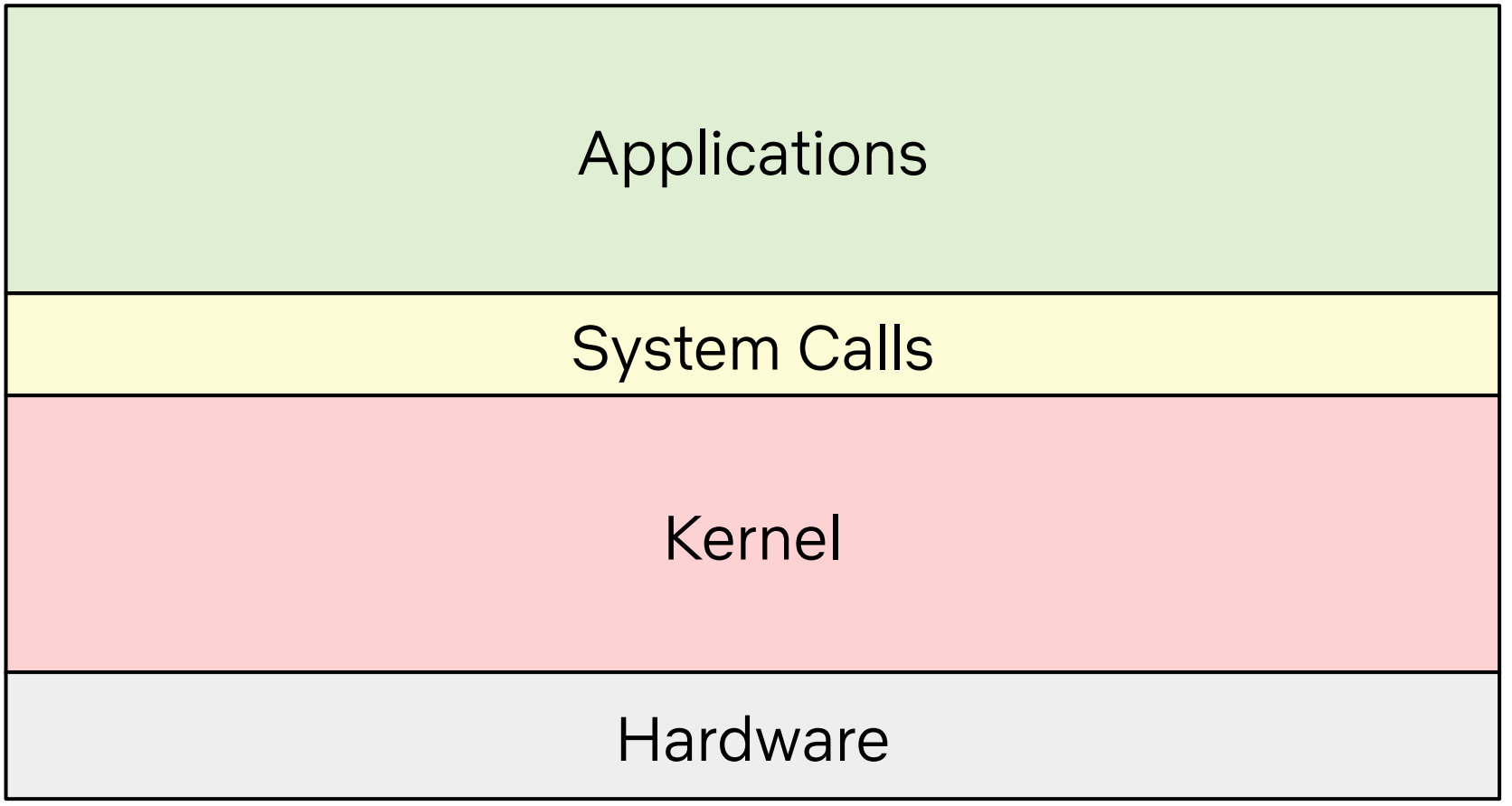
Brendan Gregg

UbuntuMasters
Oct 2019

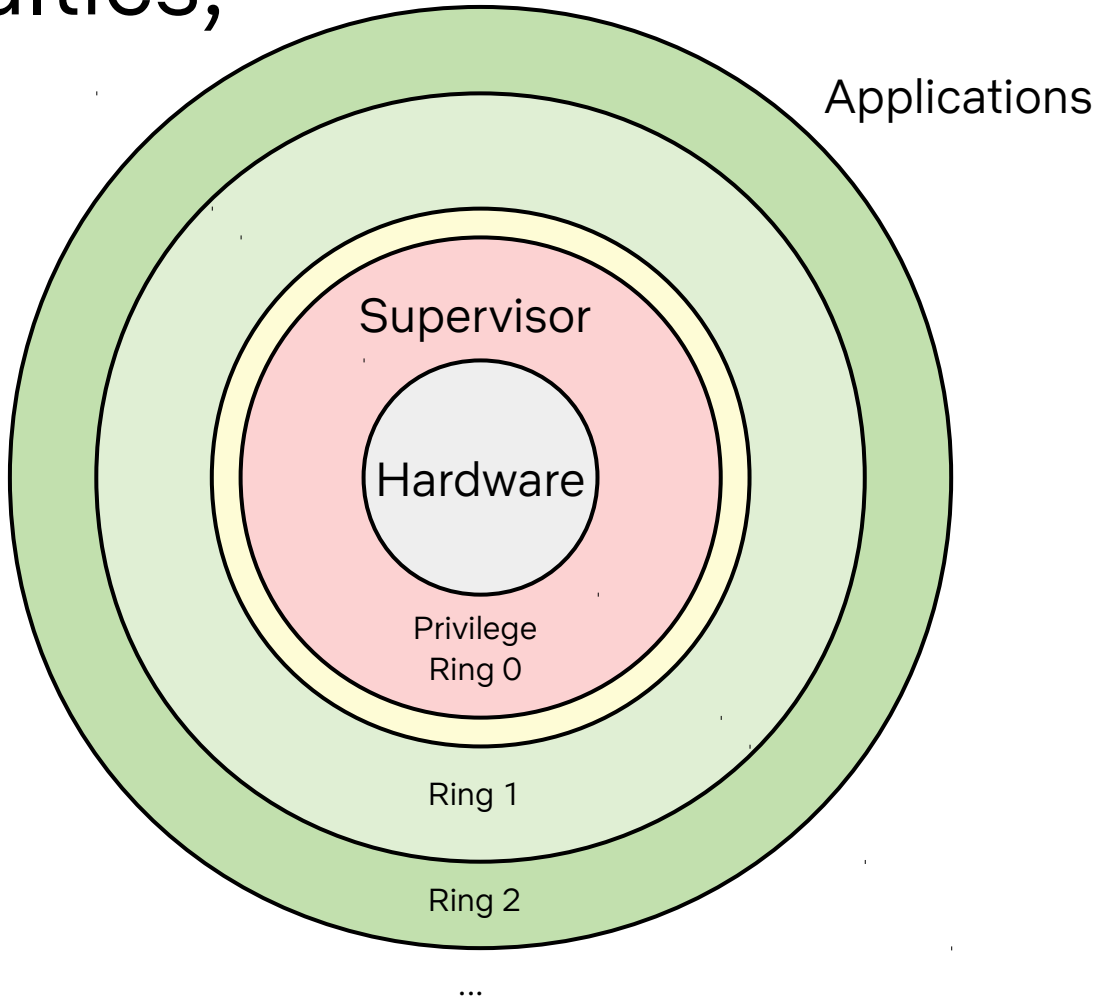
NETFLIX


BPF

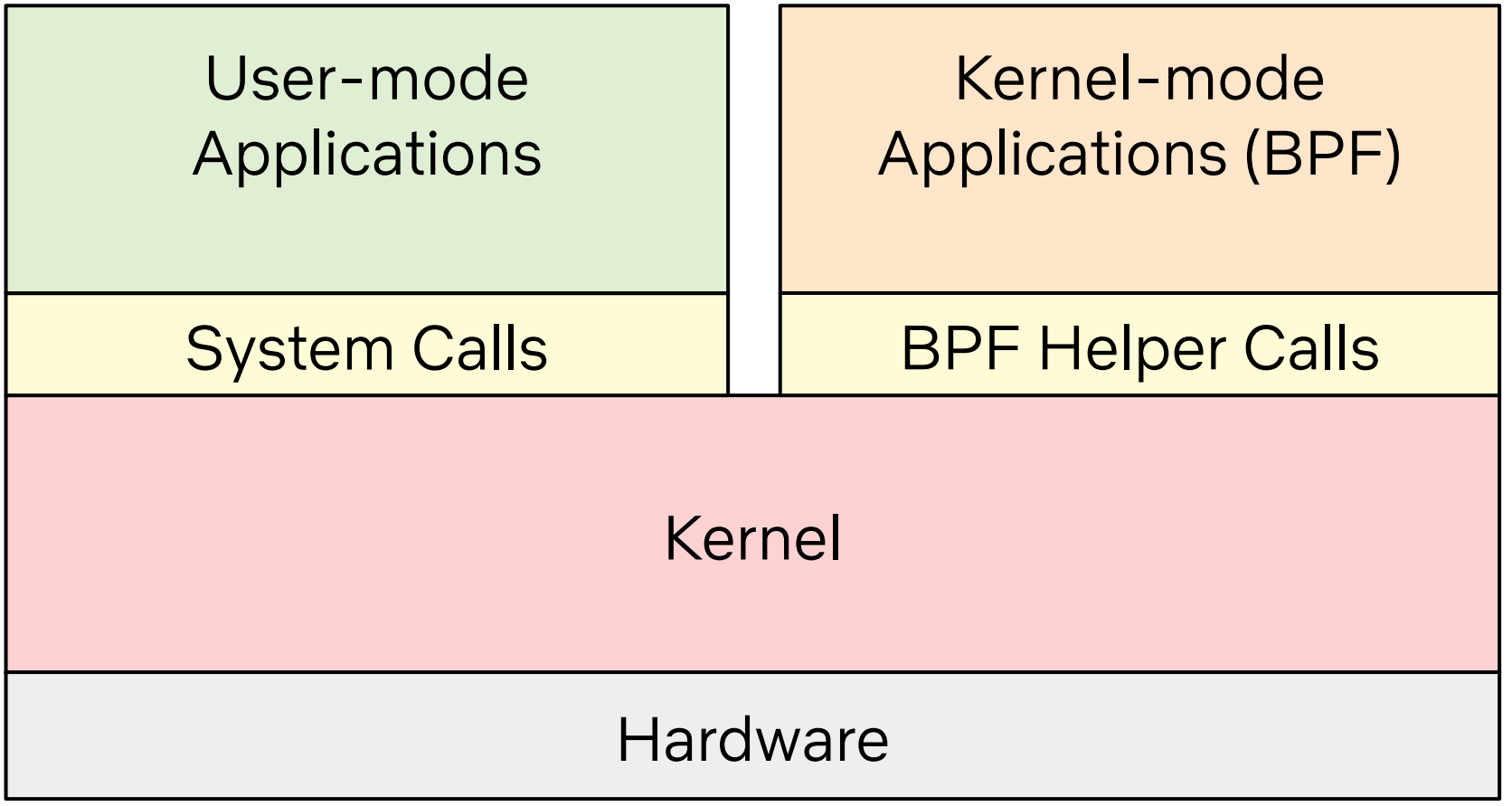
50 Years, one (dominant) OS model



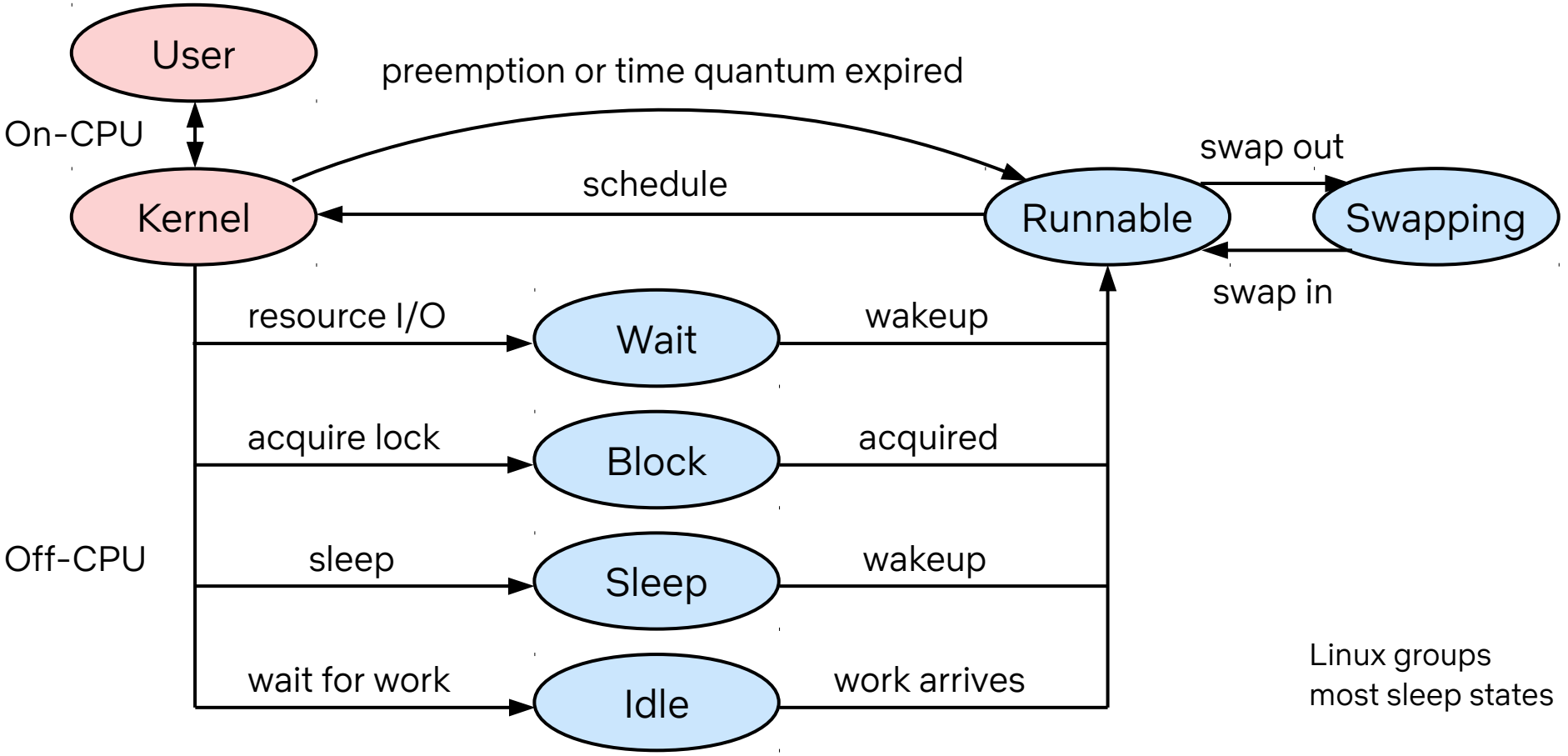
Origins: Multics, 1960s



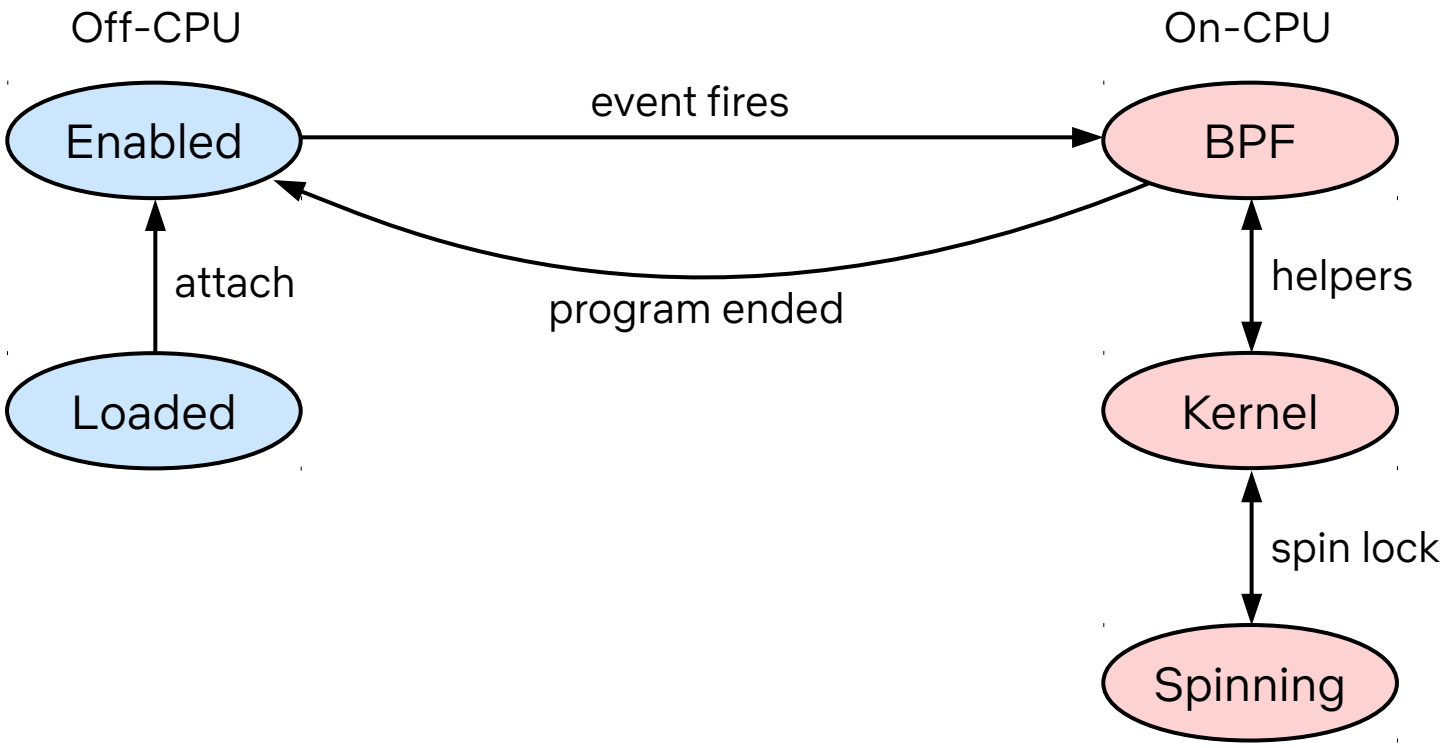
Modern Linux: A new OS model



50 Years, one process state model



BPF program state model



Netconf 2018
Alexei Starvoitov

BPF verifier in the future

- move away from existing brute force "walk all instructions" ap
  technology and static analysis
- remove #define BPF_COMPLEXITY_LIMIT 128k crutch
- remove #define BPF_MAXINSNS 4k
- support arbitrary large programs and libraries
  - 1 Million BPF instructions
- an algorithm to solve Rubik's cube will be expressible in BPF

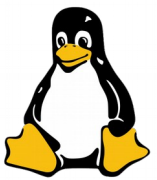
Kernel Recipes 2019, Alexei Starovoitov

~40 active BPF programs on every Facebook server

NETFLIX

>150k AWS EC2 Ubuntu server instances

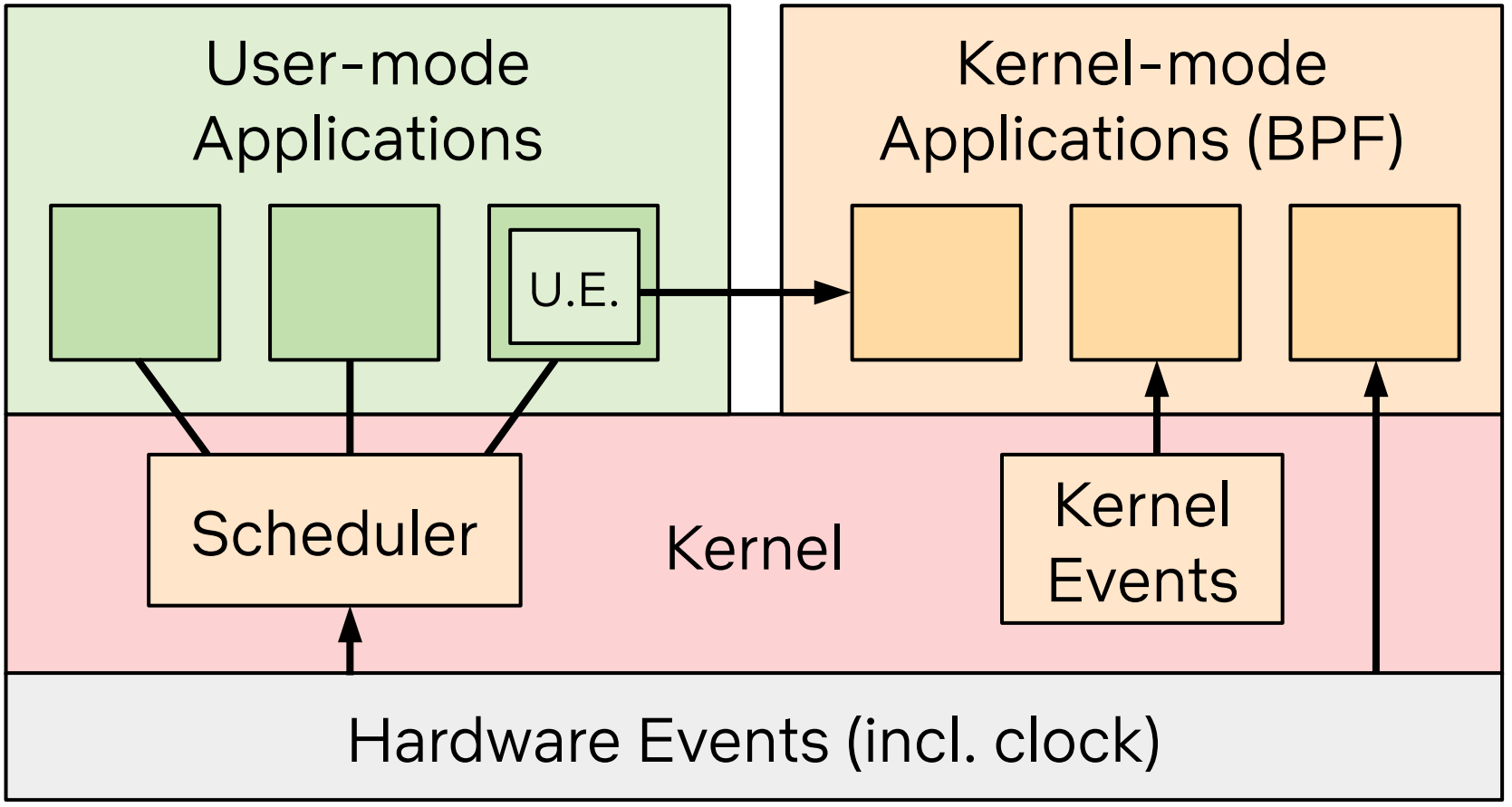
~34% US Internet traffic at night



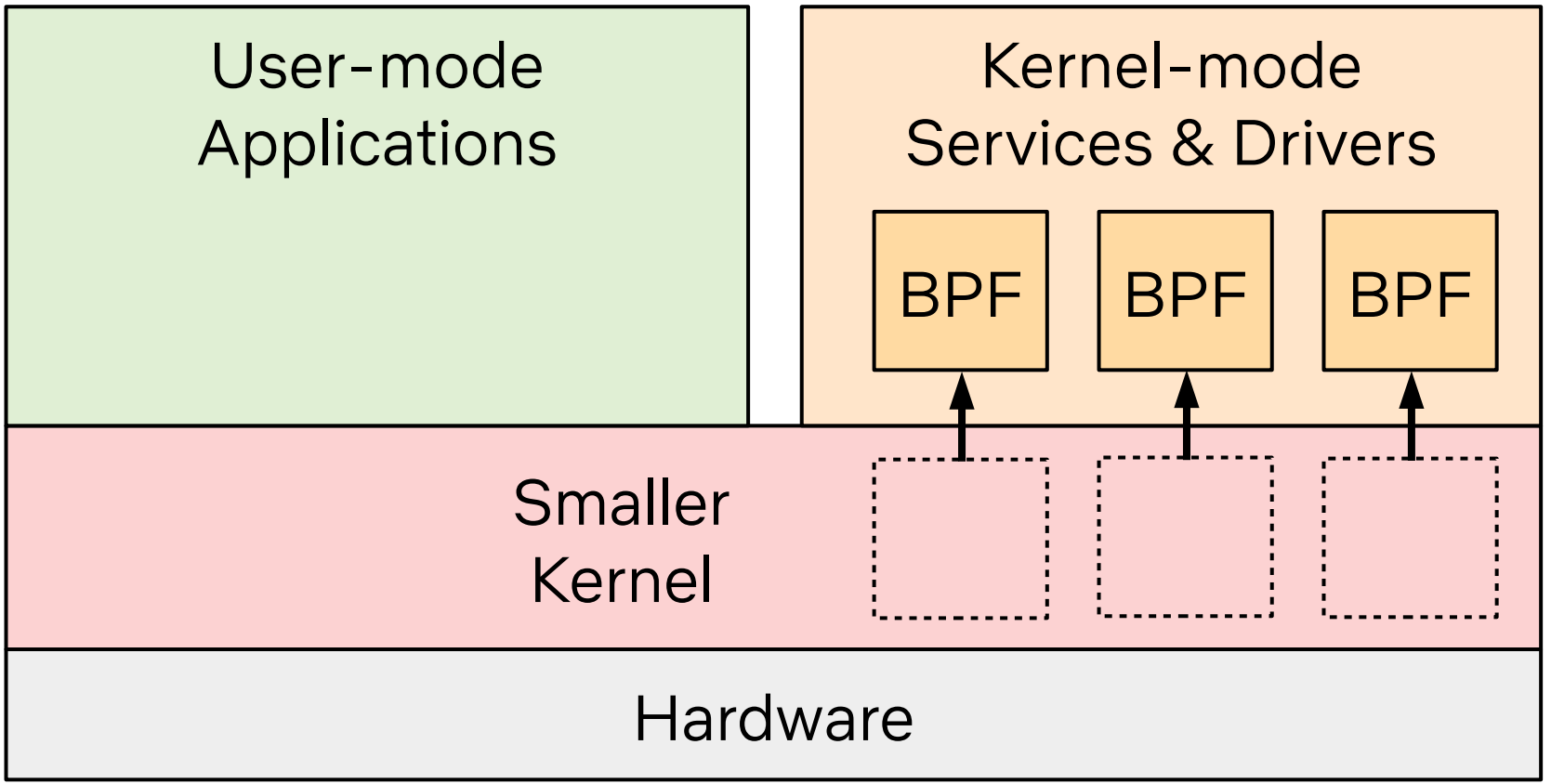
>130M subscribers

~14 active BPF programs on every instance (so far)

Modern Linux: Event-based Applications



Modern Linux is becoming Microkernel-ish



The word “microkernel” has already been invoked by Jonathan Corbet, Thomas Graf, Greg Kroah-Hartman, ...

Steven Rostedt
@srostedt

BPF will replace Linux #kr2019

2:06 AM · Sep 26, 2019 · Twitter for Android

18 Retweets **79** Likes

BPF

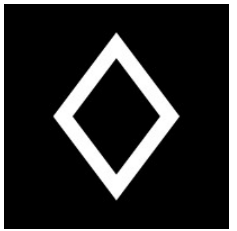
BPF 1992: Berkeley Packet Filter

```
# tcpdump -d host 127.0.0.1 and port 80
(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]
(014) jeq      #0x50            jt 17  jf 15
(015) ldh      [x + 16]
(016) jeq      #0x50            jt 17  jf 18
(017) ret      #262144
(018) ret      #0
```

A limited
virtual machine for
efficient packet filters

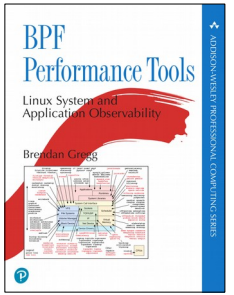
BPF 2019: aka extended BPF

bpftrace



XDP

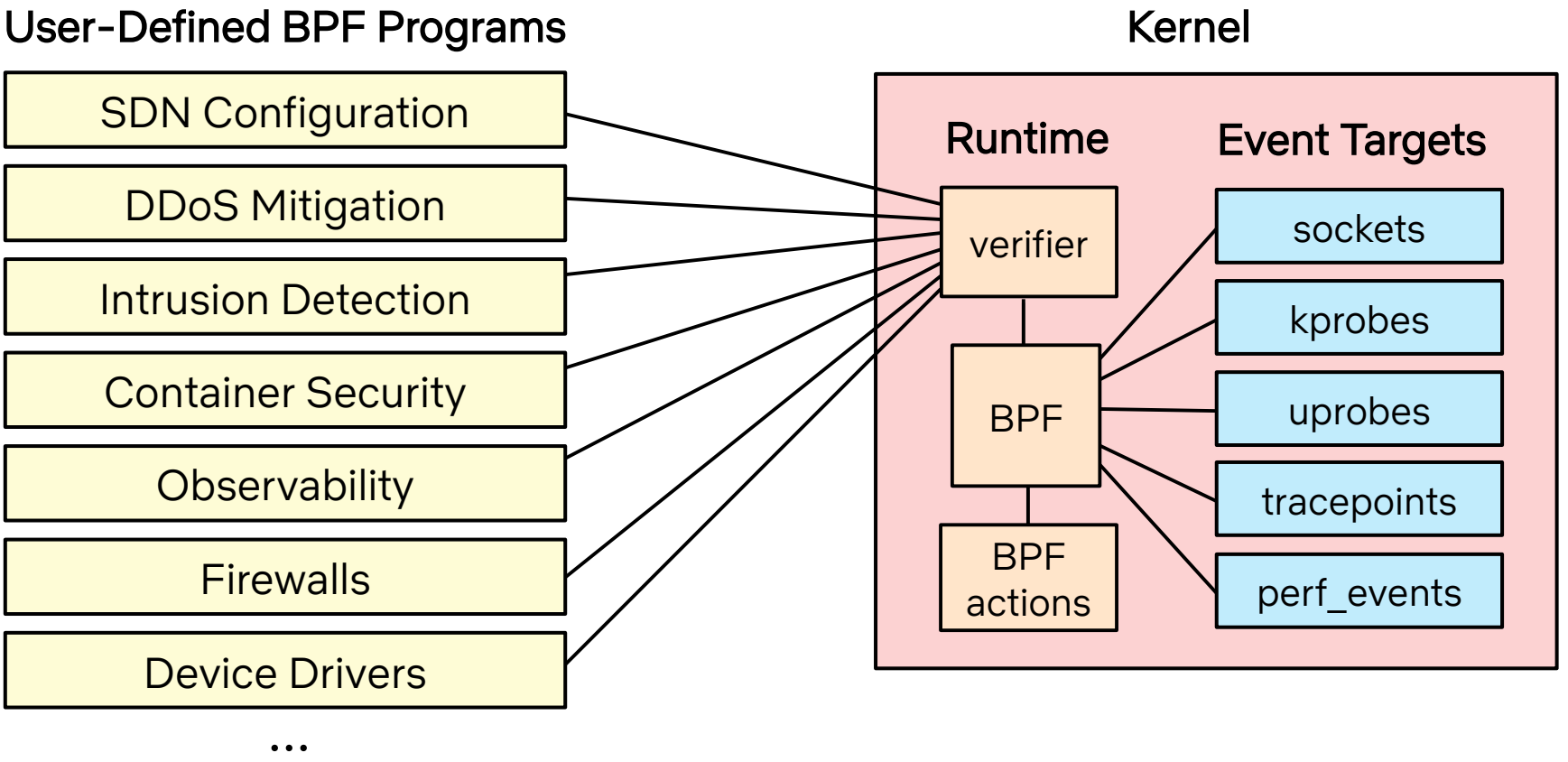
BPF microconference



bpfconf

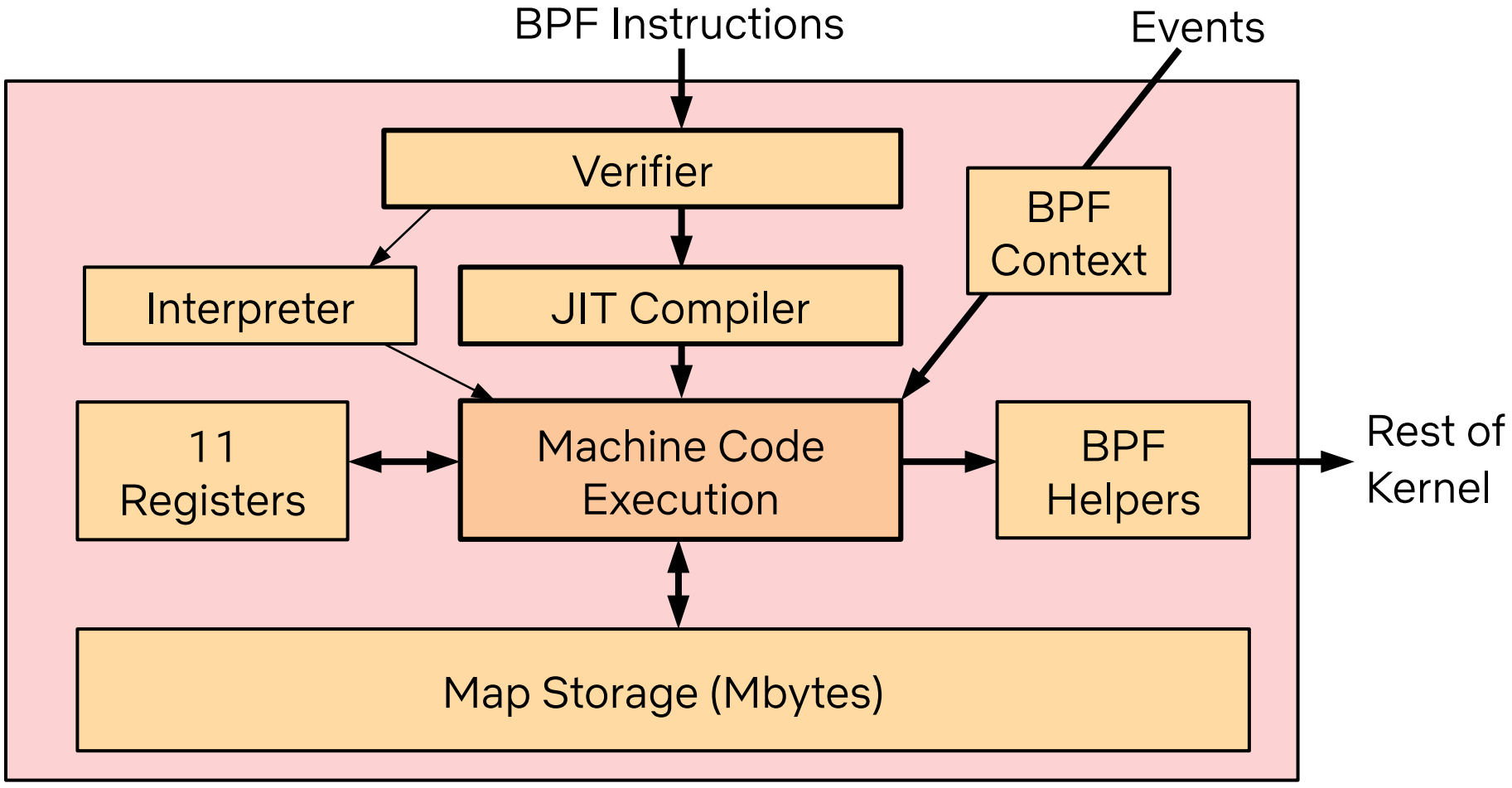
& Facebook Katran, Google KRSI, Netflix flowsrus,
and many more

BPF 2019


User-Defined BPF Programs
SDN Configuration
DDoS Mitigation
Intrusion Detection
Container Security
Observability
Firewalls
Device Drivers
...
Kernel
Runtime
Event Targets
verifier
BPF
BPF actions
sockets
kprobes
uprobes
tracepoints
perf_events

BPF is now a technology name,
and no longer an acronym

BPF Internals




Is BPF Turing complete?

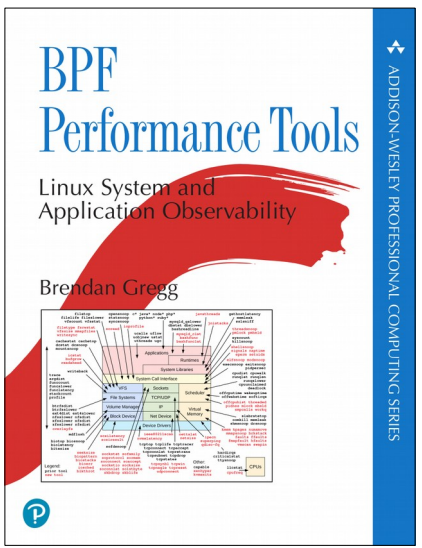
A New Type of Software

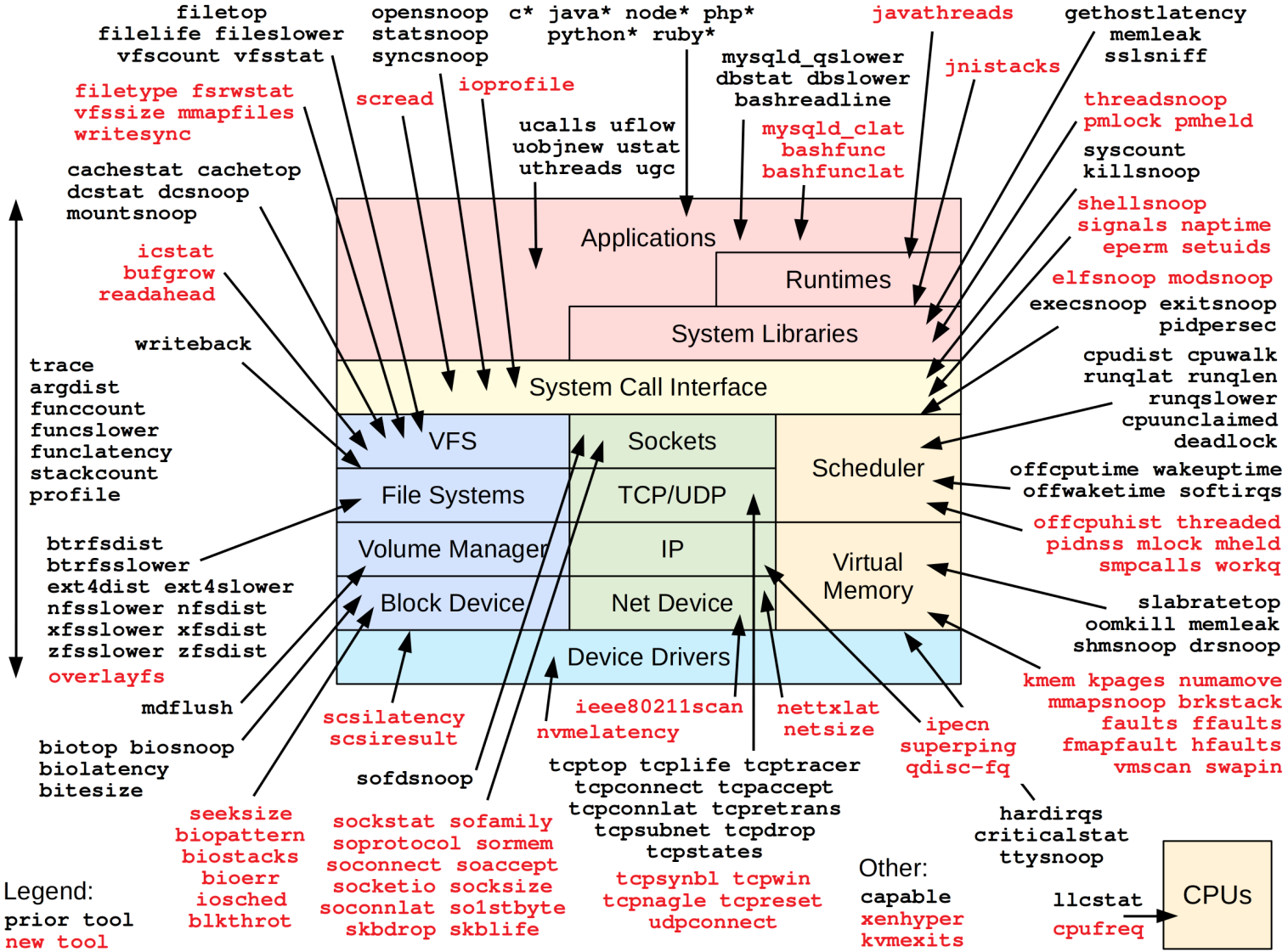
	Execution model	User defined	Compil-ation	Security	Failure mode	Resource access
User	task	yes	any	user based	abort	syscall, fault
Kernel	task	no	static	none	panic	direct
BPF	event	yes	JIT, CO-RE	verified, JIT	error message	restricted helpers

Example Use Case: BPF Observability

BPF enables a new class of
custom, **efficient**, and **production safe**
performance analysis tools

BPF Perf Tools





Ubuntu Install

BCC (BPF Compiler Collection): complex tools

```
# apt install bcc
```

bpftrace: custom tools (Ubuntu 19.04+)

```
# apt install bpftrace
```

These are default installs at Netflix, Facebook, etc.

Example: BCC tcplife

Which processes are connecting to which port?

Example: BCC tcplife

Which processes are connecting to which port?

```
# ./tcplife
PID   COMM       LADDR           LPORT RADDR           RPORT TX_KB RX_KB MS
22597 recordProg 127.0.0.1       46644 127.0.0.1       28527     0     0 0.23
3277  redis-serv 127.0.0.1       28527 127.0.0.1       46644     0     0 0.28
22598 curl       100.66.3.172    61620 52.205.89.26    80        0     1 91.79
22604 curl       100.66.3.172    44400 52.204.43.121   80        0     1 121.38
22624 recordProg 127.0.0.1       46648 127.0.0.1       28527     0     0 0.22
3277  redis-serv 127.0.0.1       28527 127.0.0.1       46648     0     0 0.27
22647 recordProg 127.0.0.1       46650 127.0.0.1       28527     0     0 0.21
3277  redis-serv 127.0.0.1       28527 127.0.0.1       46650     0     0 0.26
[...]
```

Example: BCC tcplife

```
# tcplife -h
./usage: tcplife.py [-h] [-T] [-t] [-w] [-s] [-p PID] [-L LOCALPORT]
                    [-D REMOTEPORT]

Trace the lifespan of TCP sessions and summarize

optional arguments:
  -h, --help            show this help message and exit
  -T, --time            include time column on output (HH:MM:SS)
  -t, --timestamp       include timestamp on output (seconds)
  -w, --wide            wide column output (fits IPv6 addresses)
  -s, --csv             comma separated values output
  -p PID, --pid PID     trace this PID only
  -L LOCALPORT, --localport LOCALPORT
                        comma-separated list of local ports to trace.
  -D REMOTEPORT, --remoteport REMOTEPORT
                        comma-separated list of remote ports to trace.

examples:
    ./tcplife           # trace all TCP connect()s
    ./tcplife -t        # include time column (HH:MM:SS)
[...]
```

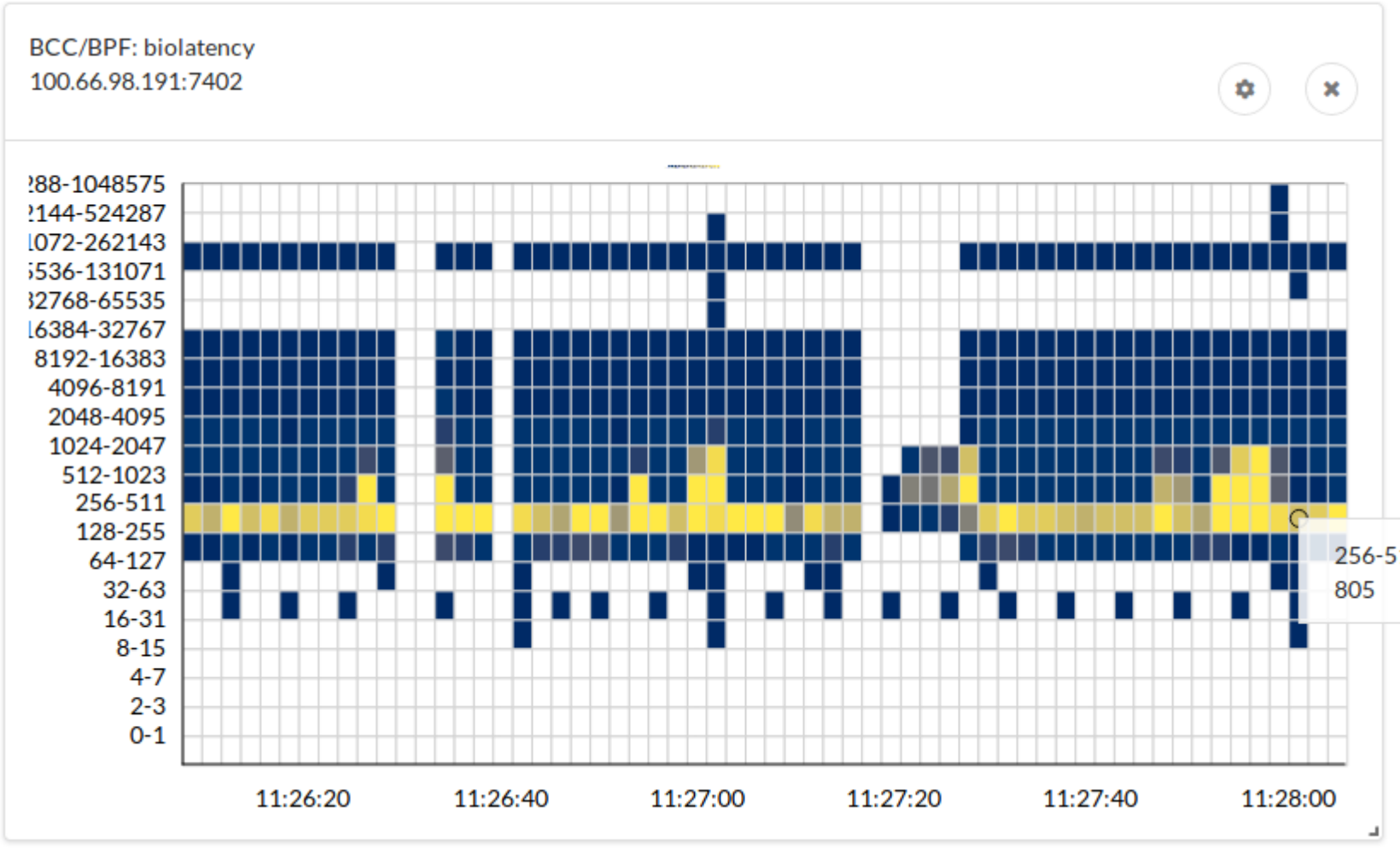
Example: BCC biolatency

What is the distribution of disk I/O latency? Per second?

Example: BCC biolatency

What is the distribution of disk I/O latency? Per second?

```
# ./biolatency -mT 1 5
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        |******                                  |

06:20:17
     msecs               : count     distribution
       0 -> 1            : 96       |****************************************|
       2 -> 3            : 25       |*********                               |
       4 -> 7            : 29       |***********                             |
[...]
```


BCC/BPF: biolatency
100.66.98.191:7402
?88-1048575
?144-524287
l072-262143
5536-131071
32768-65535
l6384-32767
8192-16383
4096-8191
2048-4095
1024-2047
512-1023
256-511
128-255
64-127
32-63
16-31
8-15
4-7
2-3
0-1
11:26:20
11:26:40
11:27:00
11:27:20
11:27:40
11:28:00
256-511:
805

Example: bpftrace readahead

Is readahead polluting the cache?

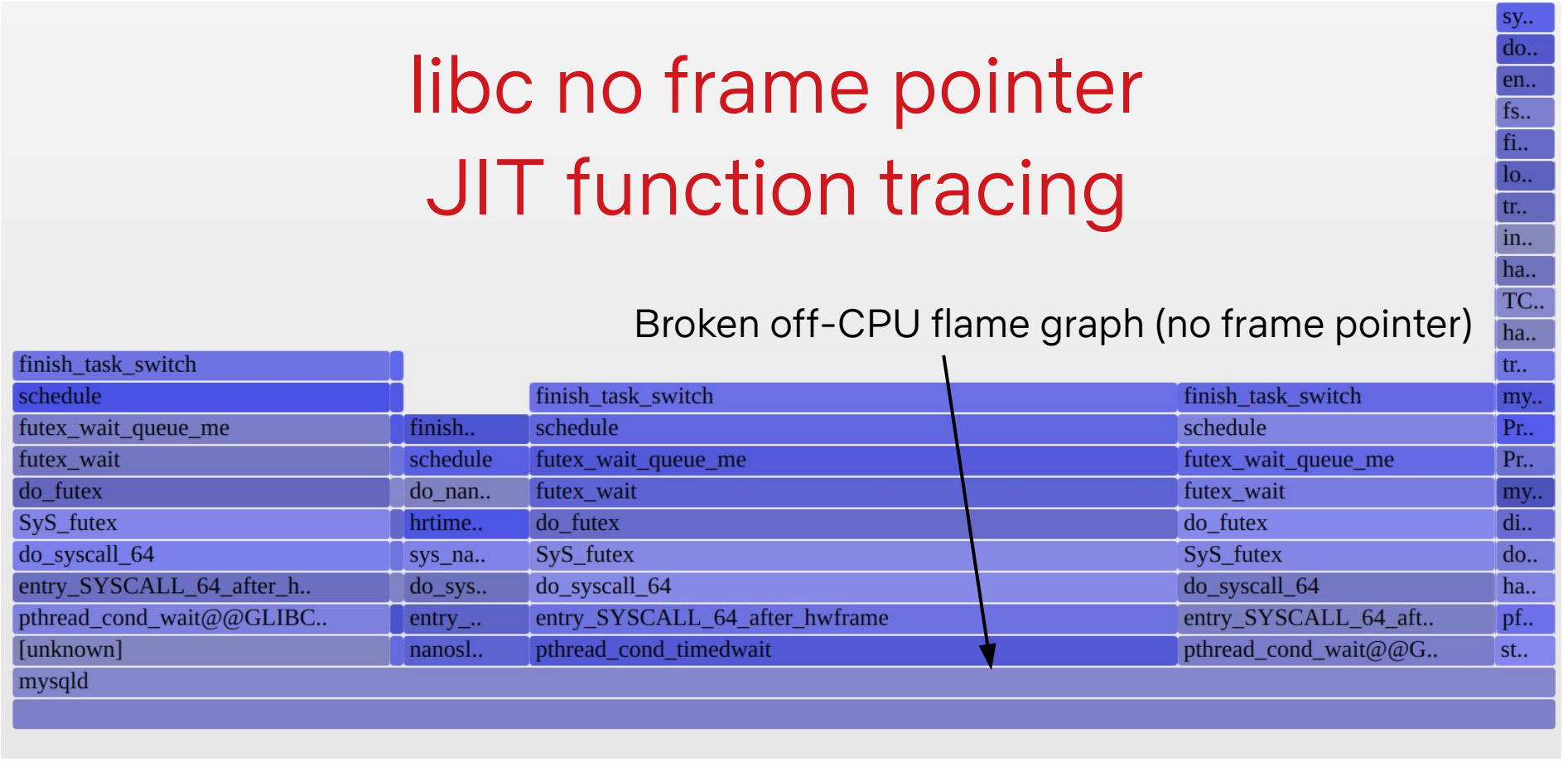
Example: bpftrace readahead

Is readahead polluting the cache?

```
# readahead.bt
Attaching 5 probes...
^C
Readahead unused pages: 128

Readahead used page age (ms):
@age_ms:
[1]                 2455 |@@@@@@@@@@@@@@@@                                    |
[2, 4)              8424 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[4, 8)              4417 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@                        |
[8, 16)             7680 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@    |
[16, 32)            4352 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@                        |
[32, 64)               0 |                                                    |
[64, 128)              0 |                                                    |
[128, 256)           384 |@@                                                  |
```

```
#!/usr/local/bin/bpftrace

kprobe:__do_page_cache_readahead    { @in_readahead[tid] = 1; }
kretprobe:__do_page_cache_readahead { @in_readahead[tid] = 0; }

kretprobe:__page_cache_alloc
/@in_readahead[tid]/
{
    @birth[retval] = nsecs;
    @rapages++;
}

kprobe:mark_page_accessed
/@birth[arg0]/
{
    @age_ms = hist((nsecs - @birth[arg0]) / 1000000);
    delete(@birth[arg0]);
    @rapages--;
}

END
{
    printf("\nReadahead unused pages: %d\n", @rapages);
    printf("\nReadahead used page age (ms):\n");
    print(@age_ms); clear(@age_ms);
    clear(@birth); clear(@in_readahead); clear(@rapages);
}
```

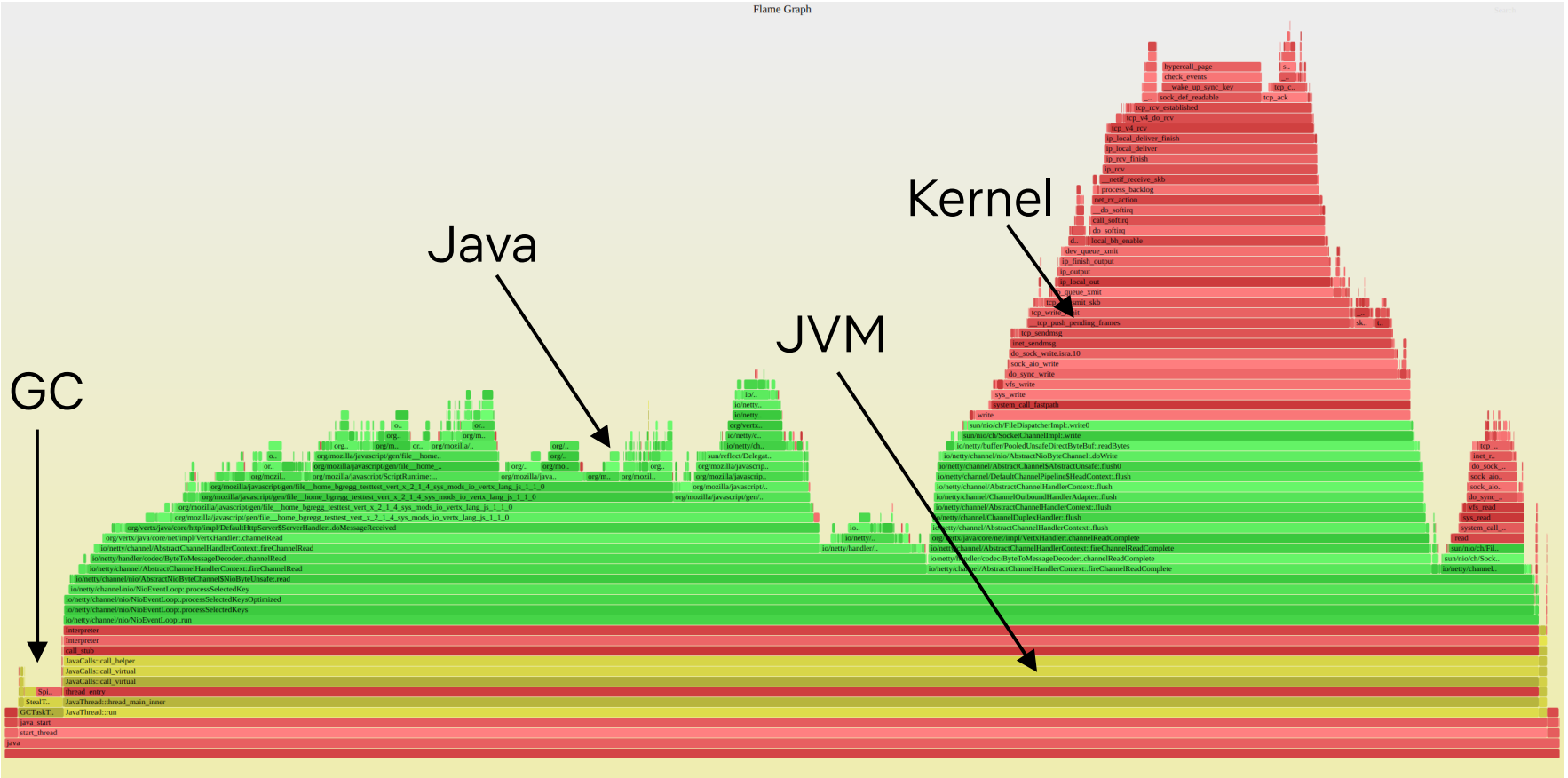
Observability Challenges



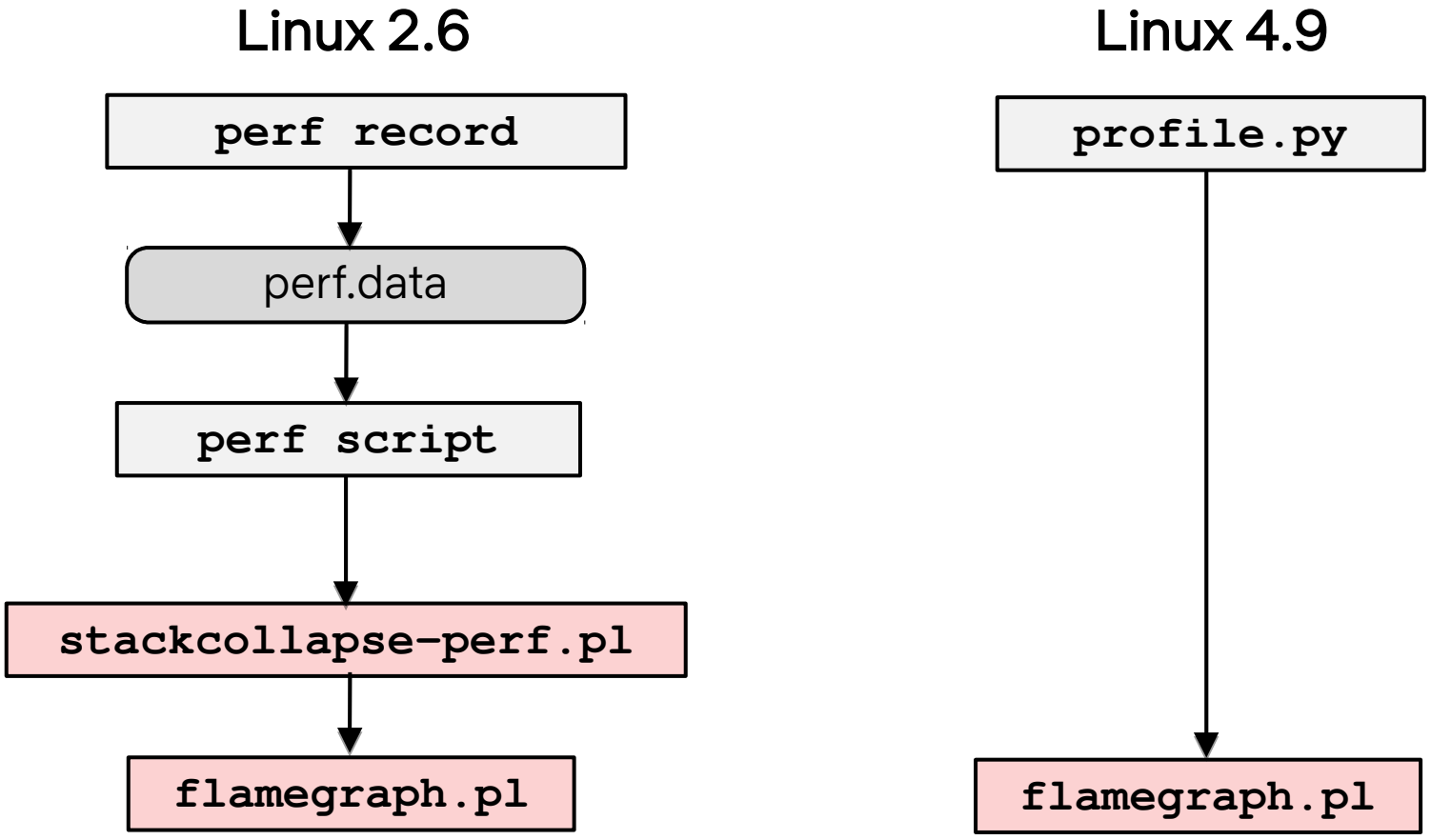
Reality Check

Many of our perf wins are from CPU flame graphs not CLI tracing

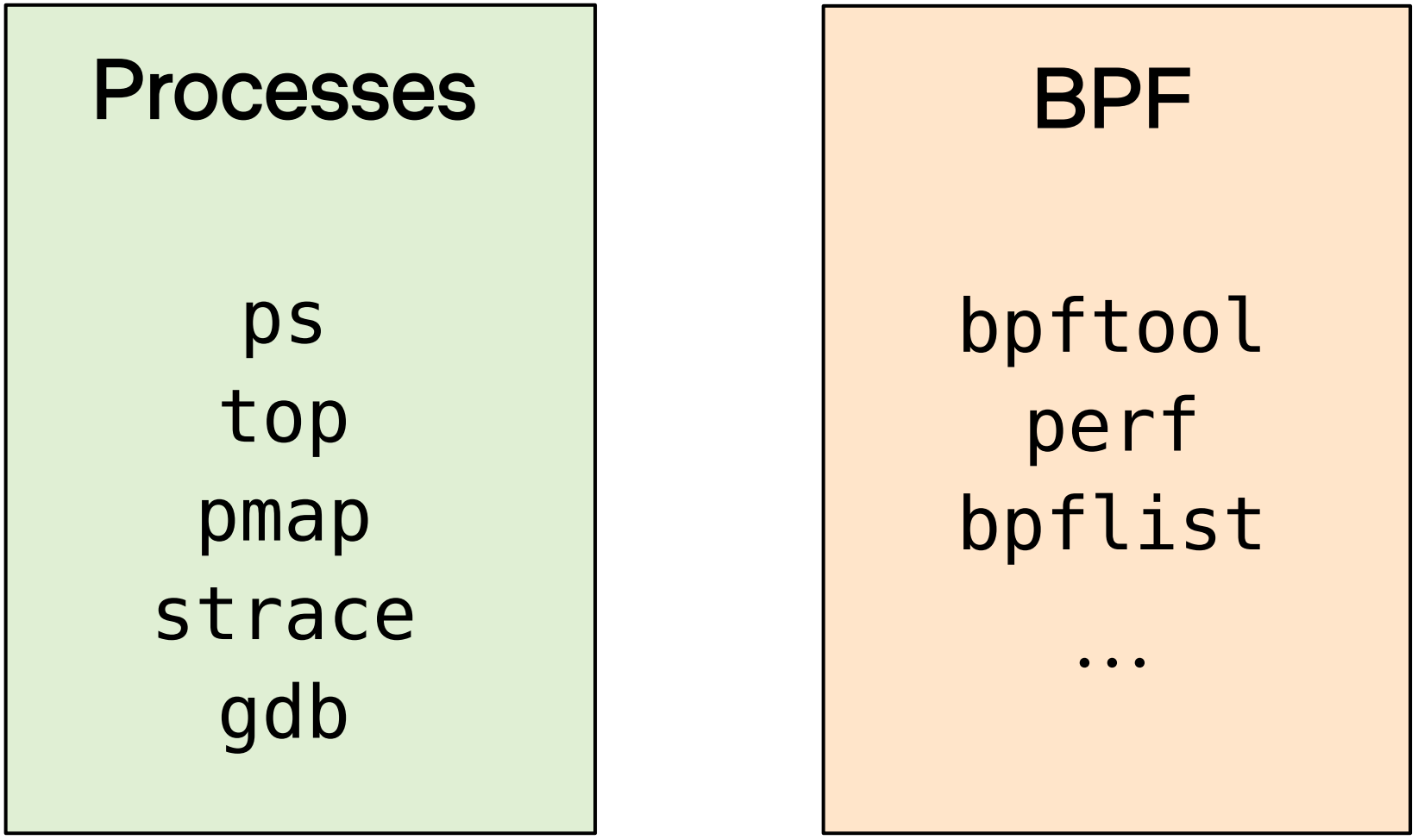
CPU Flame Graphs



BPF-based CPU Flame Graphs



Observability of BPF


Processes
ps
top
pmap
strace
gdb
BPF
bpftool
perf
bpflist
…

bpftool

PID

BPF ID

Event

```
# bpftool perf
pid 1765  fd 6: prog_id 26  kprobe  func blk_account_io_start  offset 0
pid 1765  fd 8: prog_id 27  kprobe  func blk_account_io_done  offset 0
pid 1765  fd 11: prog_id 28  kprobe  func sched_fork  offset 0
pid 1765  fd 15: prog_id 29  kprobe  func ttwu_do_wakeup  offset 0
pid 1765  fd 17: prog_id 30  kprobe  func wake_up_new_task  offset 0
pid 1765  fd 19: prog_id 31  kprobe  func finish_task_switch  offset 0
pid 1765  fd 26: prog_id 33  tracepoint  inet_sock_set_state
pid 21993  fd 6: prog_id 232  uprobe  filename /proc/self/exe  offset 1781927
pid 21993  fd 8: prog_id 233  uprobe  filename /proc/self/exe  offset 1781920
pid 21993  fd 15: prog_id 234  kprobe  func blk_account_io_done  offset 0
pid 21993  fd 17: prog_id 235  kprobe  func blk_account_io_start  offset 0
pid 25440  fd 8: prog_id 262  kprobe  func blk_mq_start_request  offset 0
pid 25440  fd 10: prog_id 263  kprobe  func blk_account_io_done  offset 0
```

```
# bpftool prog dump jited id 263
int trace_req_done(struct pt_regs * ctx):
0xffffffffc082dc6f:
; struct request *req = ctx->di;
   0:   push   %rbp
   1:   mov    %rsp,%rbp
   4:   sub    $0x38,%rsp
   b:   sub    $0x28,%rbp
   f:   mov    %rbx,0x0(%rbp)
  13:   mov    %r13,0x8(%rbp)
  17:   mov    %r14,0x10(%rbp)
  1b:   mov    %r15,0x18(%rbp)
  1f:   xor    %eax,%eax
  21:   mov    %rax,0x20(%rbp)
  25:   mov    0x70(%rdi),%rdi
; struct request *req = ctx->di;
  29:   mov    %rdi,-0x8(%rbp)
; tsp = bpf_map_lookup_elem((void *)bpf_pseudo_fd(1, -1), &req);
  2d:   movabs $0xffff96e680ab0000,%rdi
  37:   mov    %rbp,%rsi
  3a:   add    $0xfffffffffffffff8,%rsi
; tsp = bpf_map_lookup_elem((void *)bpf_pseudo_fd(1, -1), &req);
  3e:   callq  0xffffffffc39a49c1
```

LPC 2019, Arnaldo Carvalho de Melo
CPU profiling of BPF programs

“We should be able to single-step execution...
We should be able to take a core dump of all state.”
– David S. Miller, LSFMM 2019

UNIVAC 1
1951

Future

Future Predictions

More device drivers, incl. USB on BPF (ghk)

Monitoring agents

Intrusion detection systems

TCP congestion controls

CPU & container schedulers

FS readahead policies

CDN accelerator

Take Aways

BPF is a new software type

Start using BPF perf tools on Ubuntu:
bcc, bpftrace

Thanks

BPF: Alexei Starovoitov, Daniel Borkmann, David S. Miller, Linus Torvalds, BPF community

BCC: Brenden Blanco, Yonghong Song, Sasha Goldsthein, BCC community

bpftrace: Alastair Robertson, Matheus Marchini, Dan Xu, bpftrace community

Canonical: BPF support, and libc-fp (thanks in advance)

All photos credit myself; except slide 2 (Netflix) and 9 (KernelRecipes)