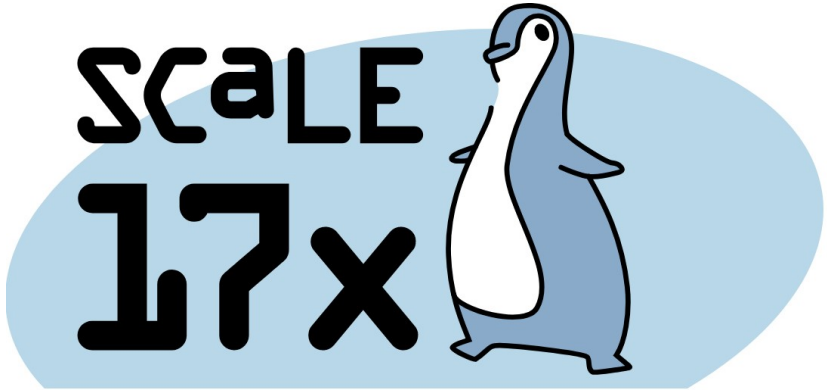
```
# biolatency.bt
Attaching 3 probes...
Tracing block device I/O... Hit Ctrl-C to end.
^C

@usecs:
[256, 512)             2 |                                                    |
[512, 1K)             10 |@                                                   |
[1K, 2K)             426 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[2K, 4K)             230 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                    |
[4K, 8K)               9 |@                                                   |
[8K, 16K)            128 |@@@@@@@@@@@@@@@@                                    |
[16K, 32K)            68 |@@@@@@@@                                            |
[32K, 64K)             0 |                                                    |
[64K, 128K)            0 |                                                    |
[128K, 256K)          10 |@                                                   |
```

eBPF Perf Tools 2019

Brendan Gregg

SCaLE
Mar 2019



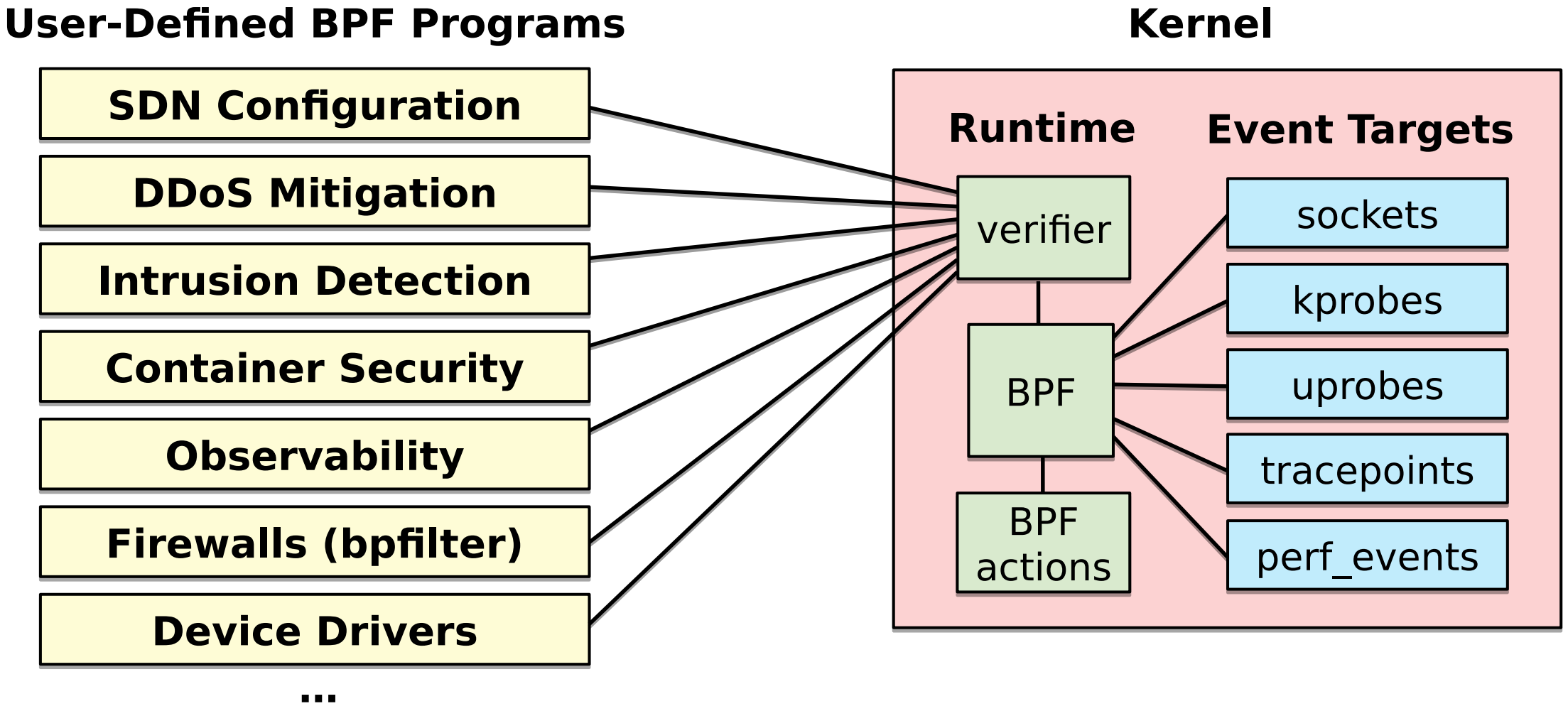
NETFLIX

LIVE DEMO
eBPF Minecraft Analysis

Enhanced BPF

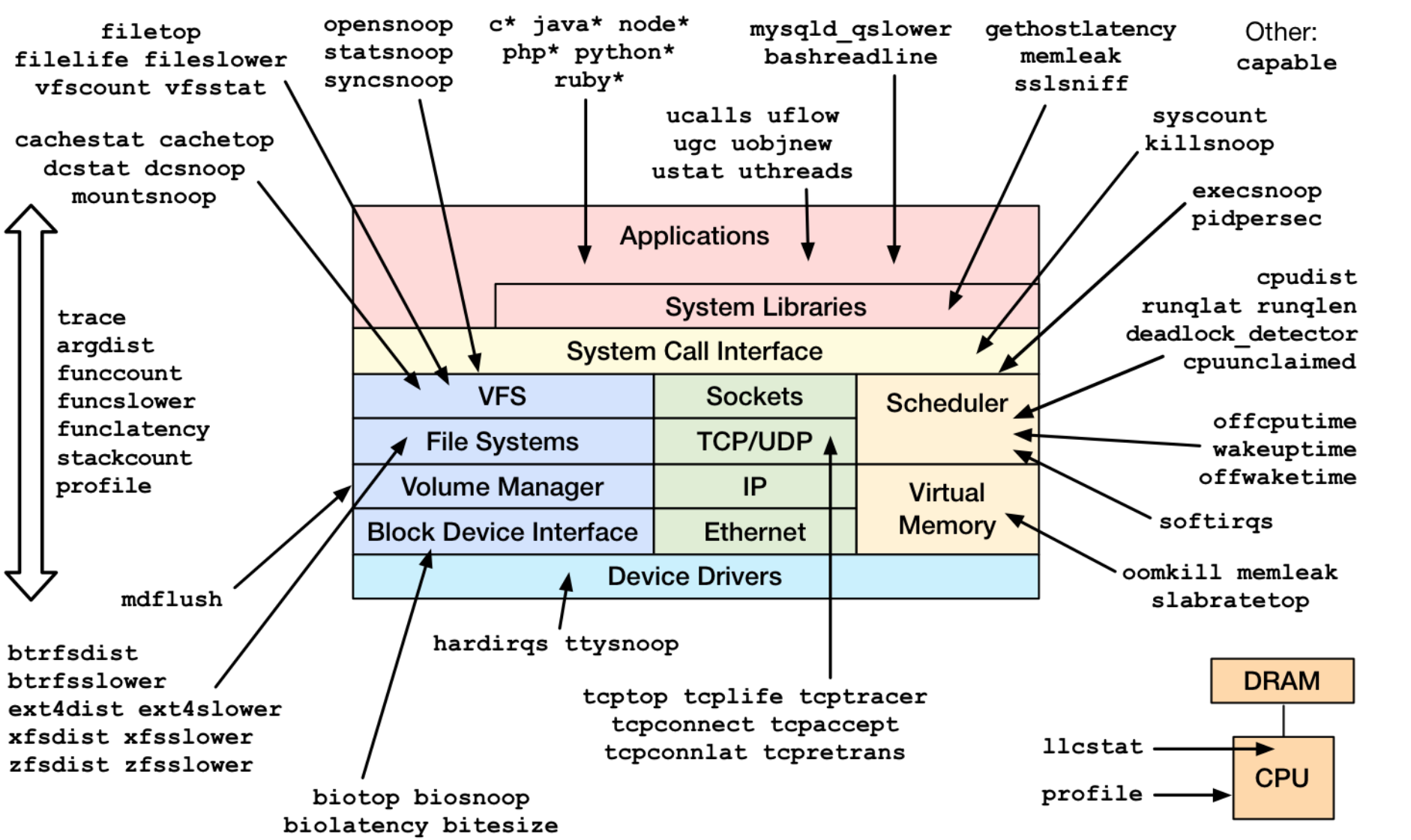
Linux 4.*

also known as just "BPF"



eBPF bcc

Linux 4.4+



https://github.com/iovisor/bcc

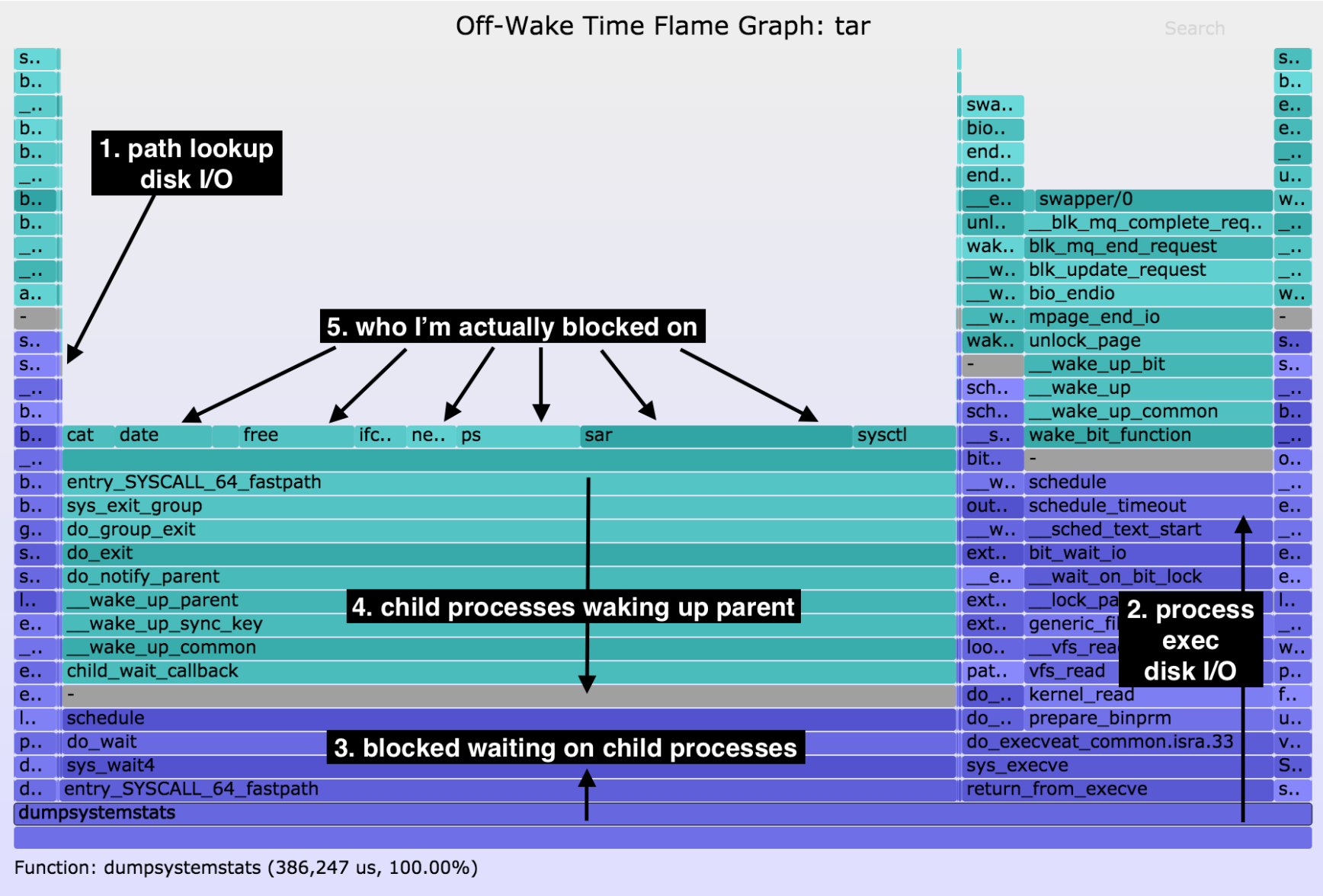
eBPF bpftrace (aka BPFtrace)

Linux 4.9+

```
# Files opened by process
bpftrace -e 't:syscalls:sys_enter_open { printf("%s %s\n", comm,
    str(args->filename)) }'

# Read size distribution by process
bpftrace -e 't:syscalls:sys_exit_read { @[comm] = hist(args->ret) }'

# Count VFS calls
bpftrace -e 'kprobe:vfs_* { @[func]++ }'

# Show vfs_read latency as a histogram
bpftrace -e 'k:vfs_read { @[tid] = nsecs }
    kr:vfs_read /@[tid]/ { @ns = hist(nsecs - @[tid]); delete(@tid) }'

# Trace user-level function
Bpftrace -e 'uretprobe:bash:readline { printf("%s\n", str(retval)) }'
…
```

https://github.com/iovisor/bpftrace

eBPF is solving new things: off-CPU + wakeup analysis



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(),
};
```

samples/bpf/sock_example.c

87 lines truncated

C/BPF

```c
SEC("kprobe/__netif_receive_skb_core")
int bpf_prog1(struct pt_regs *ctx)
{
        /* attaches to kprobe netif_receive_skb,
         * looks for packets on loobpack 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);
```

samples/bpf/tracex1_kern.c

58 lines truncated

bcc/BPF (C & Python)

```
# 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")
```

bcc examples/tracing/bitehist.py
entire program

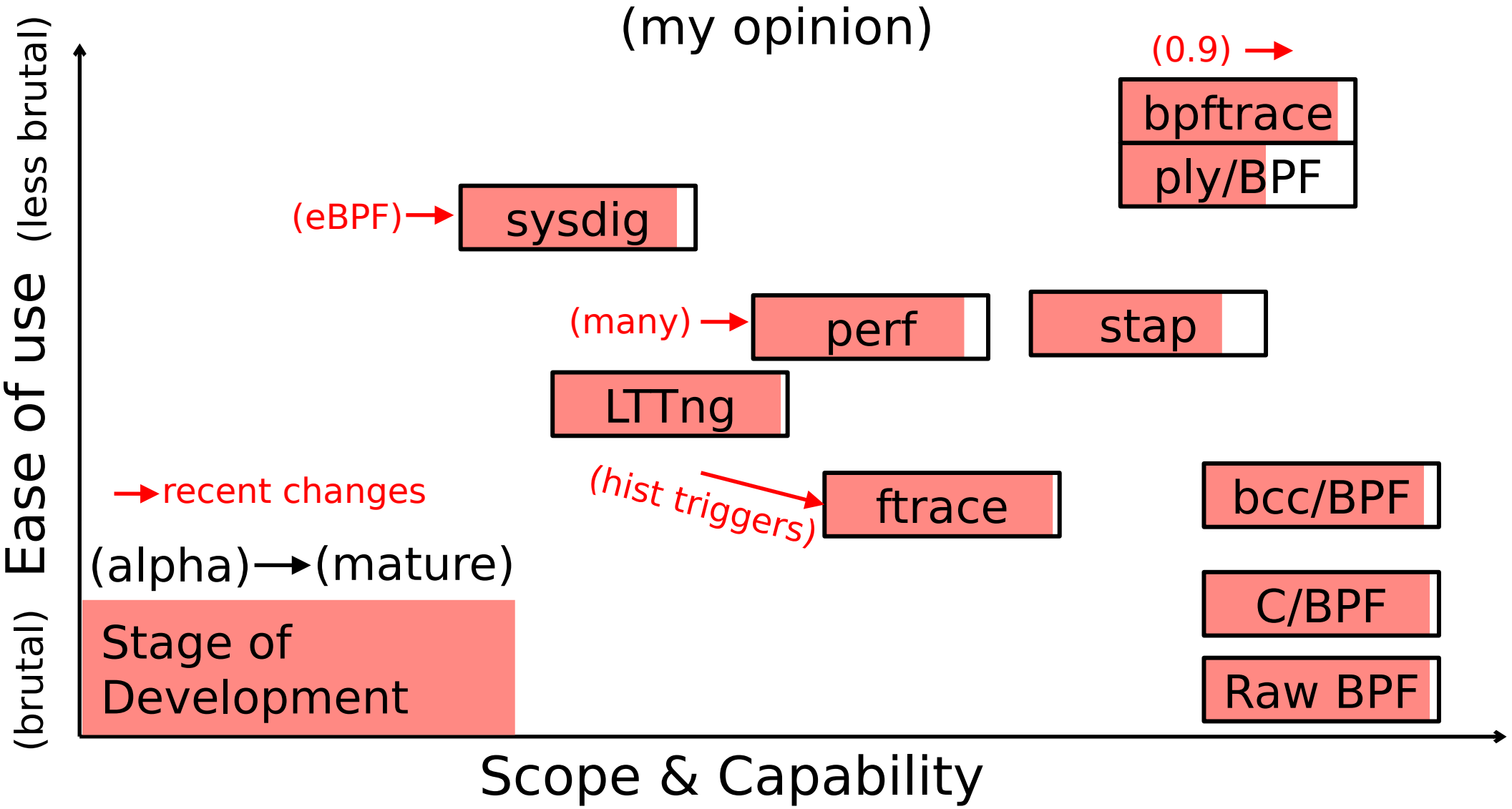
bpftrace

```
bpftrace -e 'kr:vfs_read { @ = hist(retval); }'
```

https://github.com/iovisor/bpftrace

entire program

The Tracing Landscape, Mar 2019



e.g., identify multimodal disk I/O latency and outliers with bcc/eBPF `biolatency`

```
# biolatency -mT 10
Tracing block device I/O... Hit Ctrl-C to end.

19:19:04
     msecs               : count     distribution
         0 -> 1          : 238      |*********                               |
         2 -> 3          : 424      |*****************                       |
         4 -> 7          : 834      |********************************        |
         8 -> 15         : 506      |*******************                     |
        16 -> 31         : 986      |**************************************|
        32 -> 63         : 97       |***                                     |
        64 -> 127        : 7        |                                        |
       128 -> 255        : 27       |*                                       |

19:19:14
     msecs               : count     distribution
         0 -> 1          : 427      |*****************                       |
         2 -> 3          : 424      |*****************                       |
[…]
```

bcc/eBPF programs can be laborious: `biolatency`

```
# define BPF program
bpf_text = """
#include <uapi/linux/ptrace.h>
#include <linux/blkdev.h>

typedef struct disk_key {
    char disk[DISK_NAME_LEN];
    u64 slot;
} disk_key_t;
BPF_HASH(start, struct request *);
STORAGE

// time block I/O
int trace_req_start(struct pt_regs *ctx, struct request *req)
{
    u64 ts = bpf_ktime_get_ns();
    start.update(&req, &ts);
    return 0;
}

// output
int trace_req_completion(struct pt_regs *ctx, struct request *req)
{
    u64 *tsp, delta;

    // fetch timestamp and calculate delta
    tsp = start.lookup(&req);
    if (tsp == 0) {
        return 0;   // missed issue
    }
    delta = bpf_ktime_get_ns() - *tsp;
    FACTOR

    // store as histogram
    STORE

    start.delete(&req);
    return 0;
}
"""

# code substitutions
if args.milliseconds:
    bpf_text = bpf_text.replace('FACTOR', 'delta /= 1000000;')
    label = "msecs"
else:
    bpf_text = bpf_text.replace('FACTOR', 'delta /= 1000;')
    label = "usecs"
```

```
if args.disks:
    bpf_text = bpf_text.replace('STORAGE',
        'BPF_HISTOGRAM(dist, disk_key_t);')
    bpf_text = bpf_text.replace('STORE',
        'disk_key_t key = {.slot = bpf_log2l(delta)}; ' +
        'void *__tmp = (void *)req->rq_disk->disk_name; ' +
        'bpf_probe_read(&key.disk, sizeof(key.disk), __tmp); ' +
        'dist.increment(key);')
else:
    bpf_text = bpf_text.replace('STORAGE', 'BPF_HISTOGRAM(dist);')
    bpf_text = bpf_text.replace('STORE',
        'dist.increment(bpf_log2l(delta));')
if debug or args.ebpf:
    print(bpf_text)
    if args.ebpf:
        exit()

# load BPF program
b = BPF(text=bpf_text)
if args.queued:
    b.attach_kprobe(event="blk_account_io_start", fn_name="trace_req_start")
else:
    b.attach_kprobe(event="blk_start_request", fn_name="trace_req_start")
    b.attach_kprobe(event="blk_mq_start_request", fn_name="trace_req_start")
b.attach_kprobe(event="blk_account_io_completion",
    fn_name="trace_req_completion")

print("Tracing block device I/O... Hit Ctrl-C to end.")

# output
exiting = 0 if args.interval else 1
dist = b.get_table("dist")
while (1):
    try:
        sleep(int(args.interval))
    except KeyboardInterrupt:
        exiting = 1

    print()
    if args.timestamp:
        print("%-8s\n" % strftime("%H:%M:%S"), end="")

    dist.print_log2_hist(label, "disk")
    dist.clear()

    countdown -= 1
    if exiting or countdown == 0:
        exit()
```

… rewritten in bpftrace (launched Oct 2018)!

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

BEGIN
{
    printf("Tracing block device I/O... Hit Ctrl-C to end.\n");
}

kprobe:blk_account_io_start
{
    @start[arg0] = nsecs;
}

kprobe:blk_account_io_completion
/@start[arg0]/

{
    @usecs = hist((nsecs - @start[arg0]) / 1000);
    delete(@start[arg0]);
}
```

… rewritten in bpftrace

```
# biolatency.bt
Attaching 3 probes...
Tracing block device I/O... Hit Ctrl-C to end.
^C

@usecs:
[256, 512)             2 |                                                    |
[512, 1K)             10 |@                                                   |
[1K, 2K)             426 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[2K, 4K)             230 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@                    |
[4K, 8K)               9 |@                                                   |
[8K, 16K)            128 |@@@@@@@@@@@@@@@@                                    |
[16K, 32K)            68 |@@@@@@@@                                            |
[32K, 64K)             0 |                                                    |
[64K, 128K)            0 |                                                    |
[128K, 256K)          10 |@                                                   |
```

bcc

canned complex tools, agents

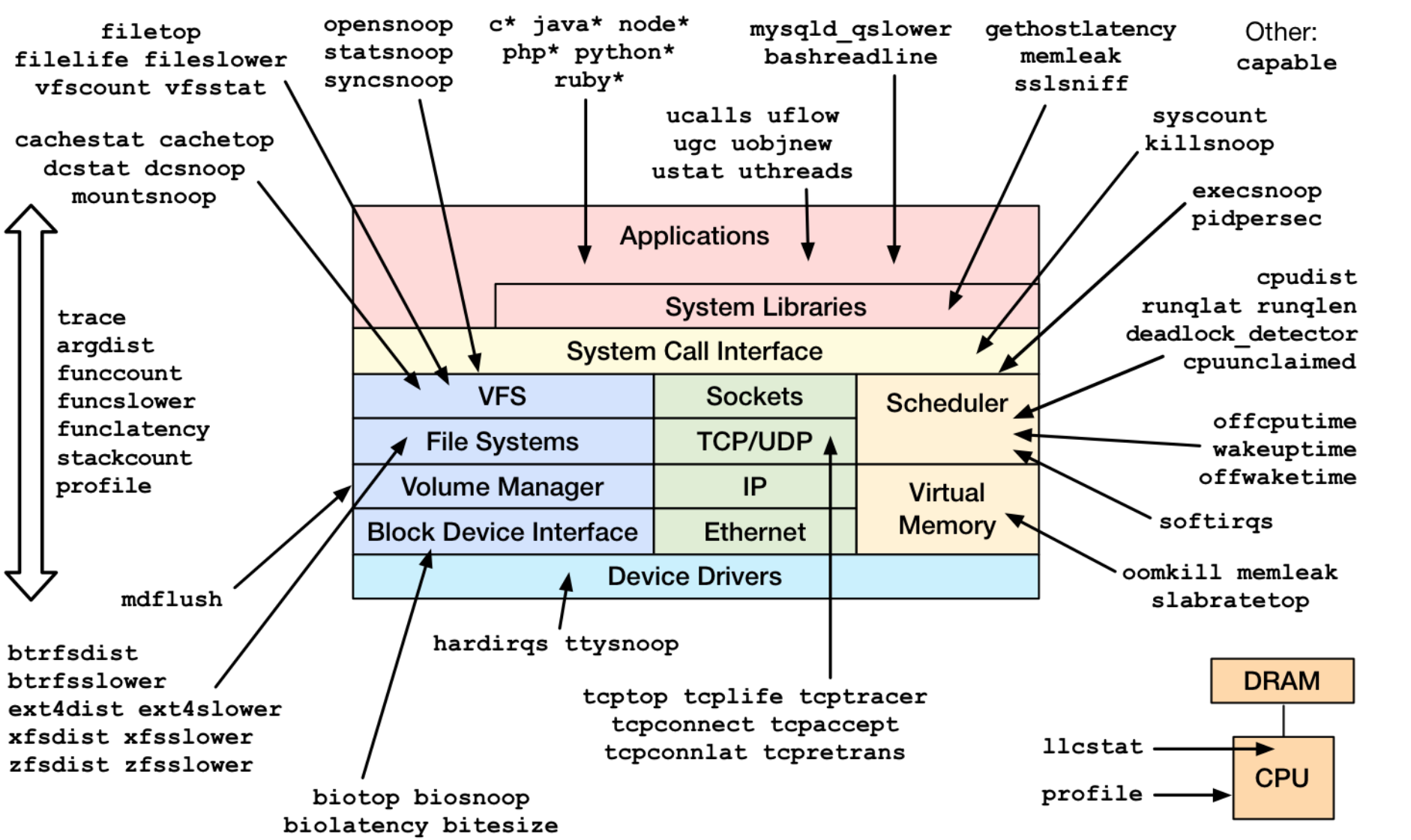
bpftrace

one-liners, custom scripts

bcc

eBPF bcc

Linux 4.4+

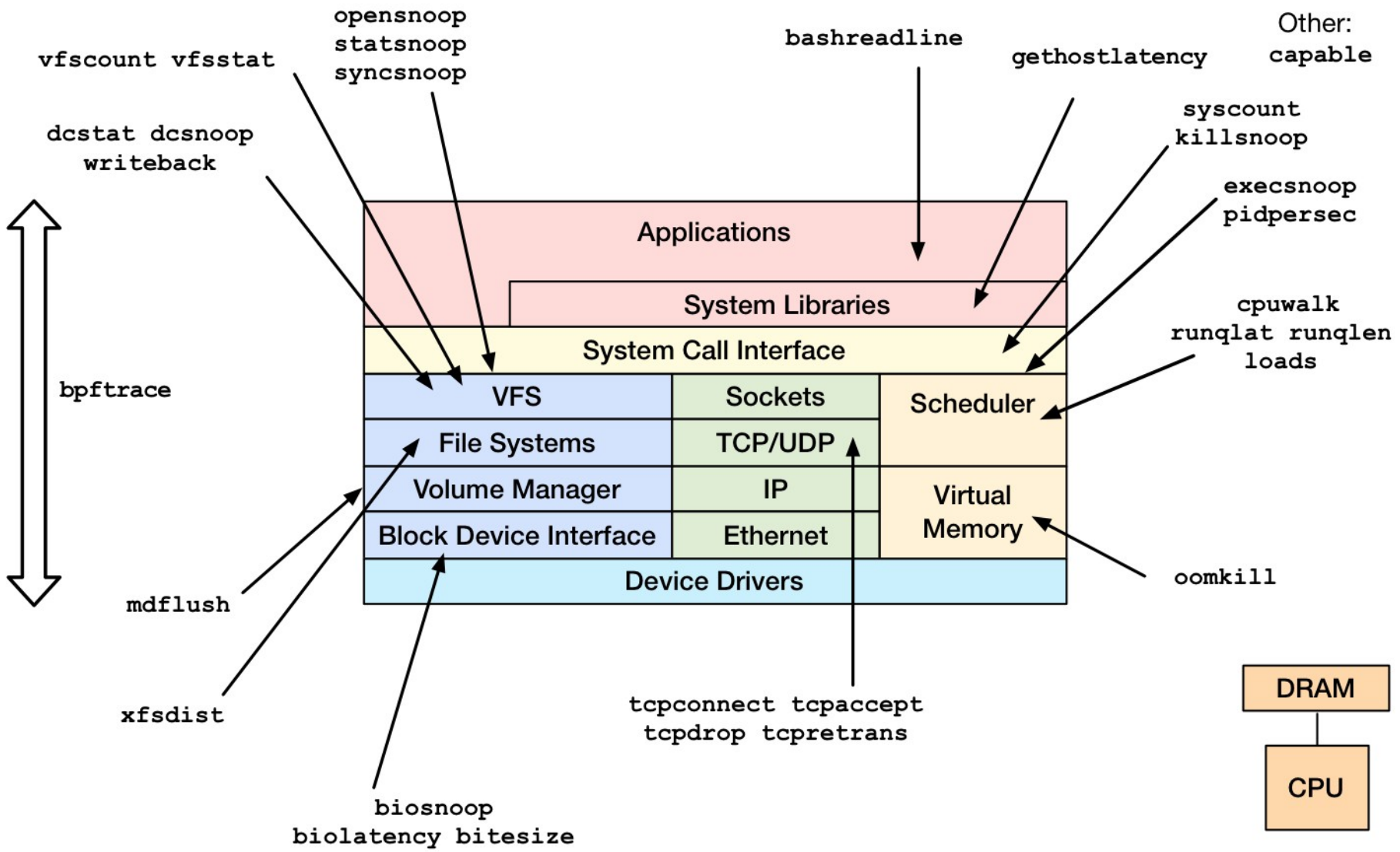


https://github.com/iovisor/bcc

bpftrace

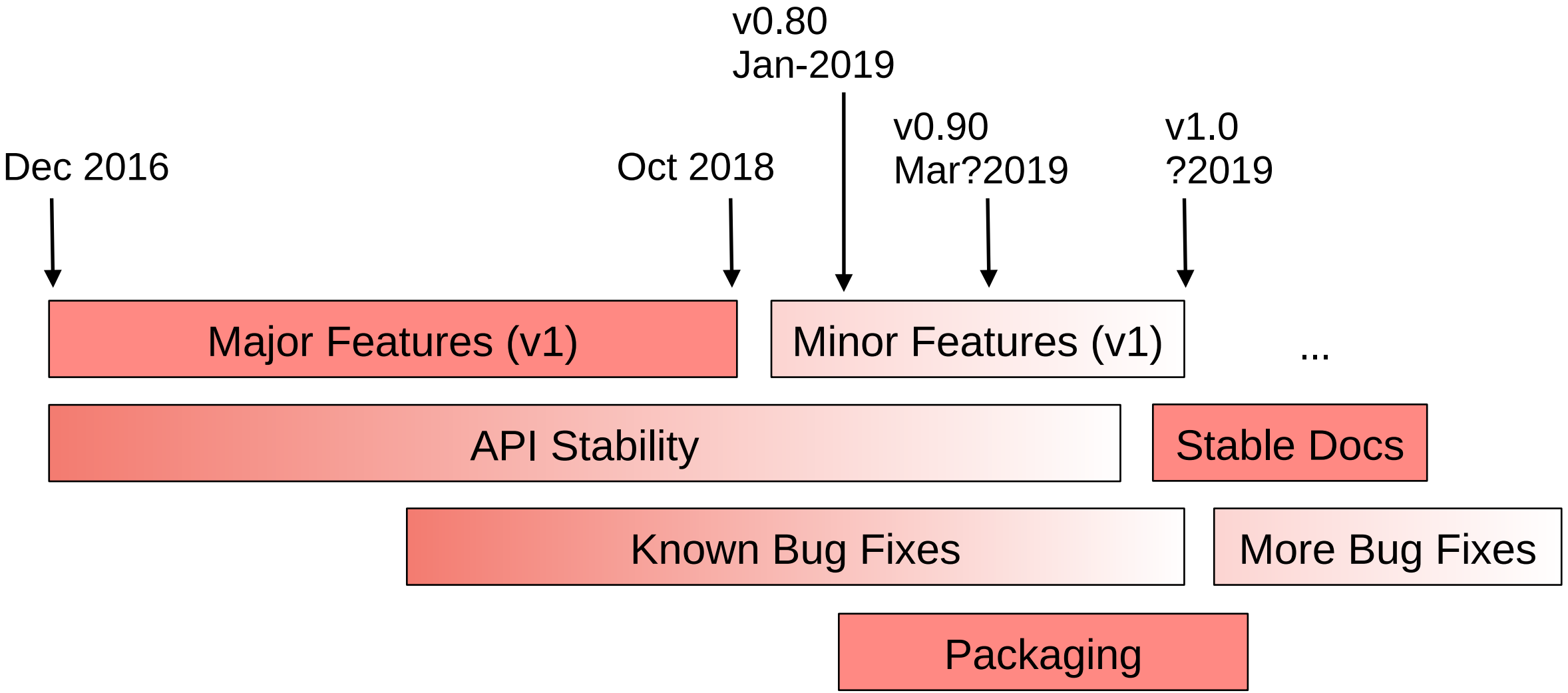
eBPF bpftrace

Linux 4.9+



https://github.com/iovisor/bcc

bpftrace Development



bpftrace Syntax

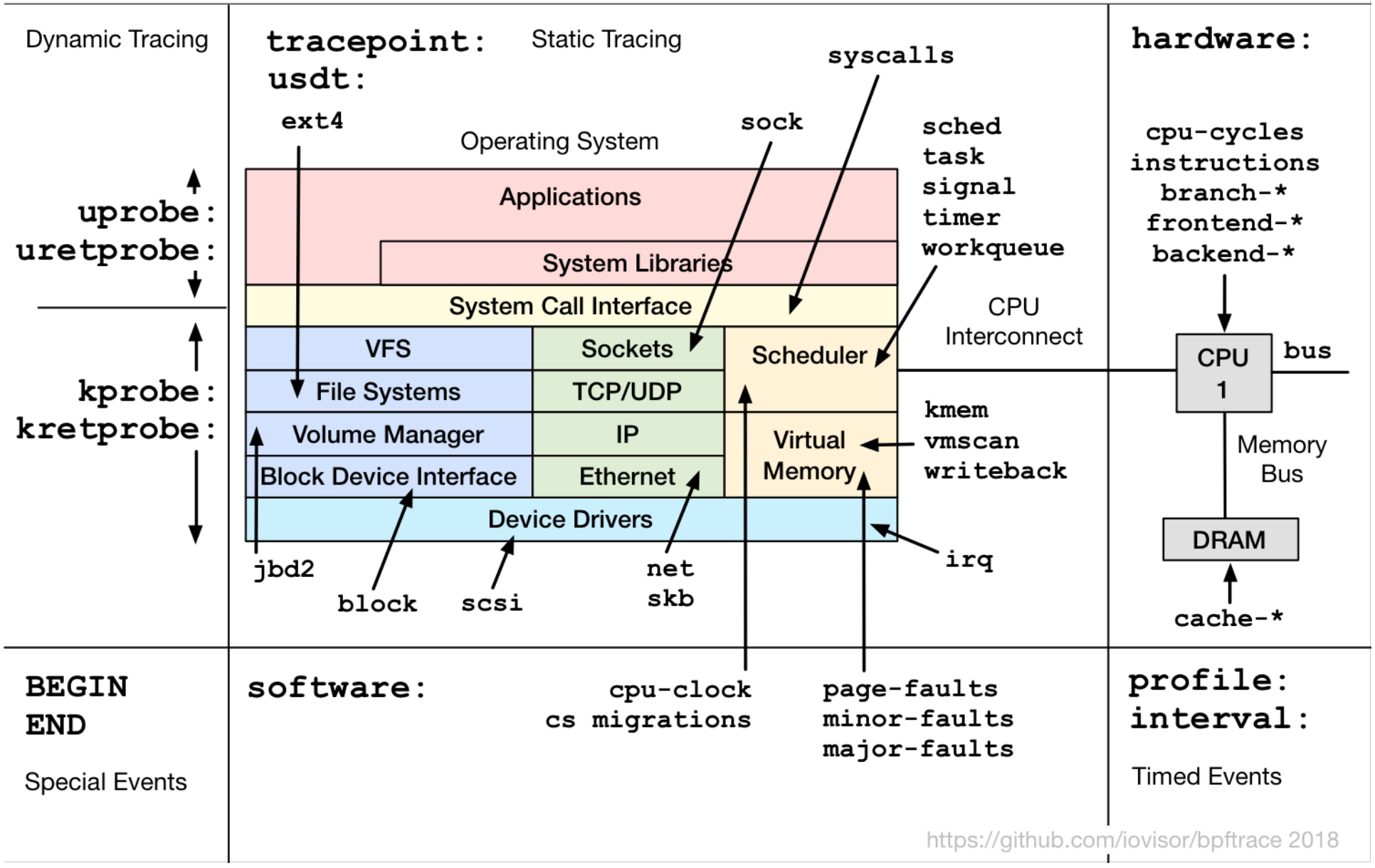
```
bpftrace -e 'k:do_nanosleep /pid > 100/ { @[comm]++ }'
```

Probe

Filter
(optional)

Action

Probes



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

Probe Type Shortcuts

`tracepoint`	`t`	Kernel static tracepoints
`usdt`	`U`	User-level statically defined tracing
`kprobe`	`k`	Kernel function tracing
`kretprobe`	`kr`	Kernel function returns
`uprobe`	`u`	User-level function tracing
`uretprobe`	`ur`	User-level function returns
`profile`	`p`	Timed sampling across all CPUs
`interval`	`i`	Interval output
`software`	`s`	Kernel software events
`hardware`	`h`	Processor hardware events

Filters

- `/pid == 181/`
- `/comm != “sshd”/`
- `/@ts[tid]/`

Actions

- Per-event output
  - `printf()`
  - `system()`
  - `join()`
  - `time()`
- Map Summaries
  - `@ = count()` or `@++`
  - `@ = hist()`
  - …

The following is in the https://github.com/iovisor/bpftrace/blob/master/docs/reference_guide.md

Functions

- **`hist(n)`** Log2 histogram
- **`lhist(n, min, max, step)`** Linear hist.
- **`count()`** Count events
- **`sum(n)`** Sum value
- **`min(n)`** Minimum value
- **`max(n)`** Maximum value
- **`avg(n)`** Average value
- **`stats(n)`** Statistics
- **`str(s)`** String
- **`sym(p)`** Resolve kernel addr
- **`usym(p)`** Resolve user addr
- **`kaddr(n)`** Resolve kernel symbol
- **`uaddr(n)`** Resolve user symbol

- **`printf(fmt, ...)`** Print formatted
- **`print(@x[, top[, div]])`** Print map
- **`delete(@x)`** Delete map element
- **`clear(@x)`** Delete all keys/values
- **`reg(n)`** Register lookup
- **`join(a)`** Join string array
- **`time(fmt)`** Print formatted time
- **`system(fmt)`** Run shell command
- **`exit()`** Quit bpftrace

Variable Types

- Basic Variables
  - **`@global`**
  - **`@thread_local[tid]`**
  - **`$scratch`**
- Associative Arrays
  - **`@array[key] = value`**
- Buitins
  - **`pid`**
  - `...`

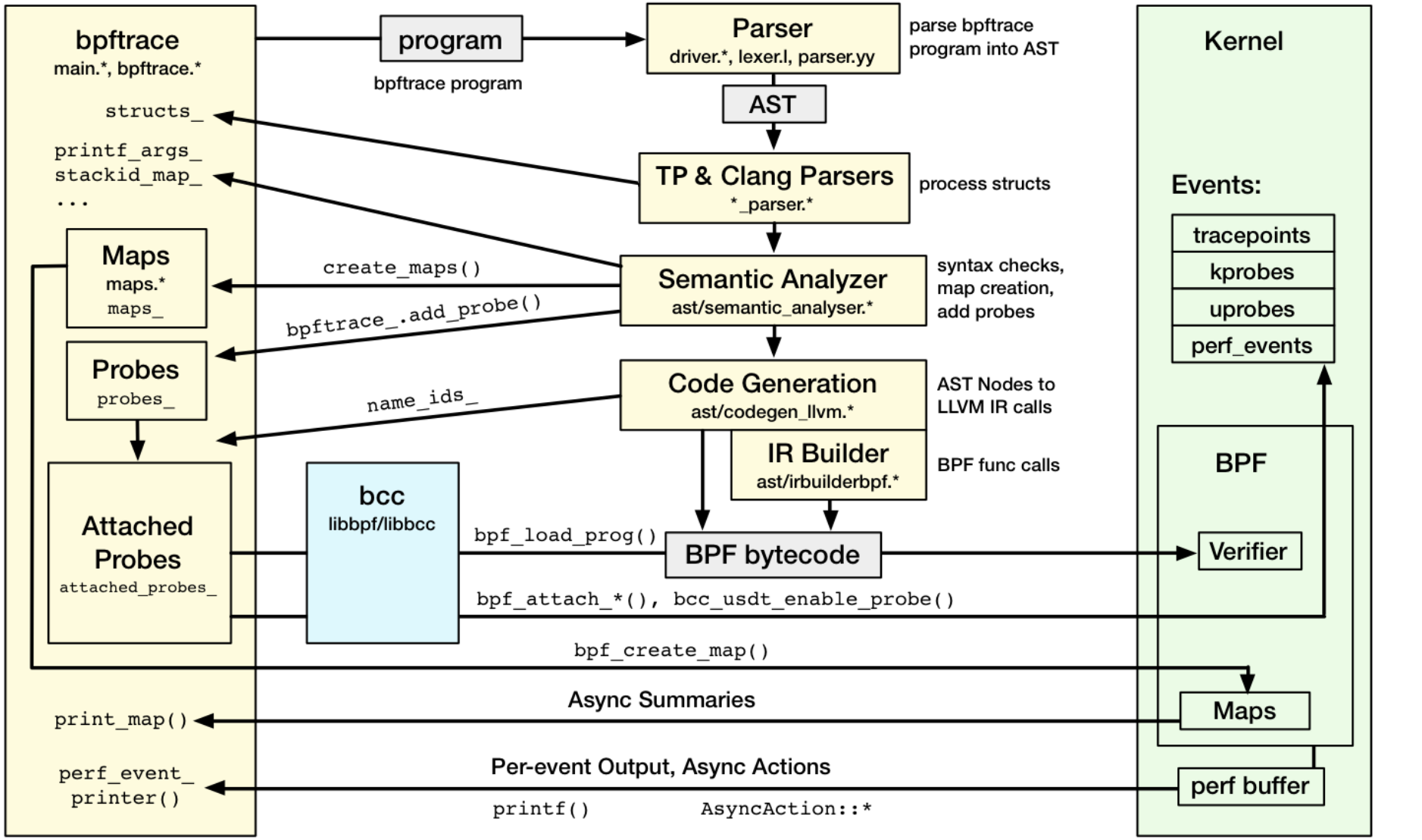
Builtin Variables

- **`pid`** Process ID (kernel tgid)
- **`tid`** Thread ID (kernel pid)
- **`cgroup`** Current Cgroup ID
- **`uid`** User ID
- **`gid`** Group ID
- **`nsecs`** Nanosecond timestamp
- **`cpu`** Processor ID
- **`comm`** Process name
- **`stack`** Kernel stack trace
- **`ustack`** User stack trace
- **`arg0`**, **`arg1`**, ... Function arguments
- **`retval`** Return value
- **`func`** Function name
- **`probe`** Full name of the probe
- **`curtask`** Current task_struct (u64)
- **`rand`** Random number (u32)

biolatency (again)

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

BEGIN
{
    printf("Tracing block device I/O... Hit Ctrl-C to end.\n");
}

kprobe:blk_account_io_start
{
    @start[arg0] = nsecs;
}

kprobe:blk_account_io_completion
/@start[arg0]/

{
    @usecs = hist((nsecs - @start[arg0]) / 1000);
    delete(@start[arg0]);
}
```

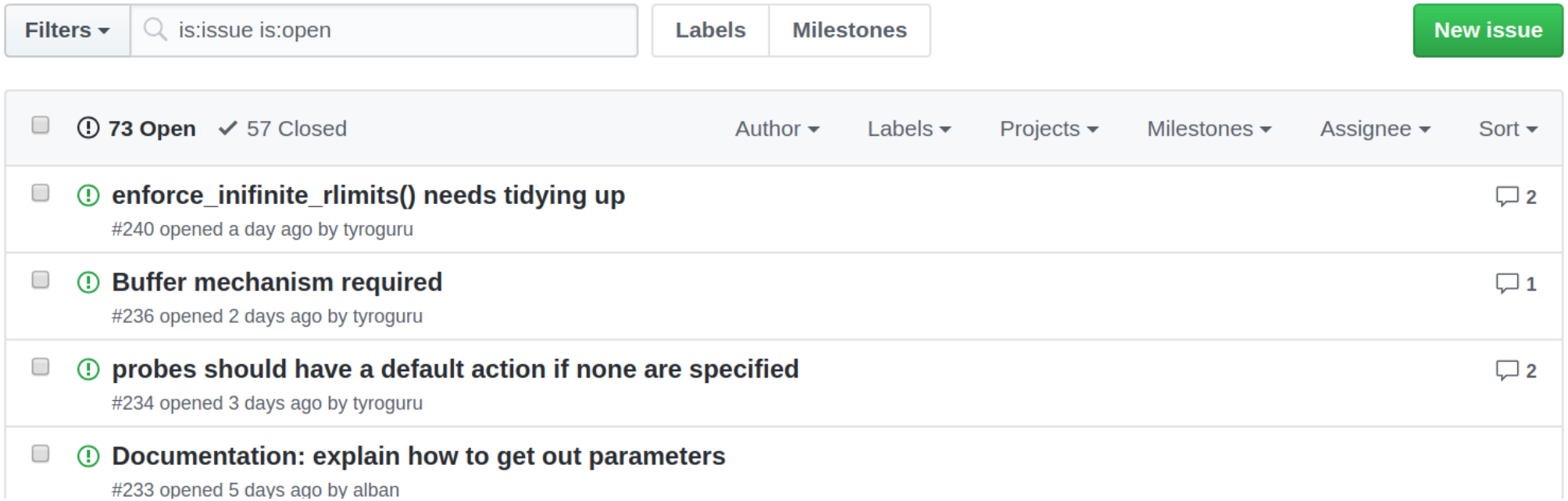
bpftrace Internals



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

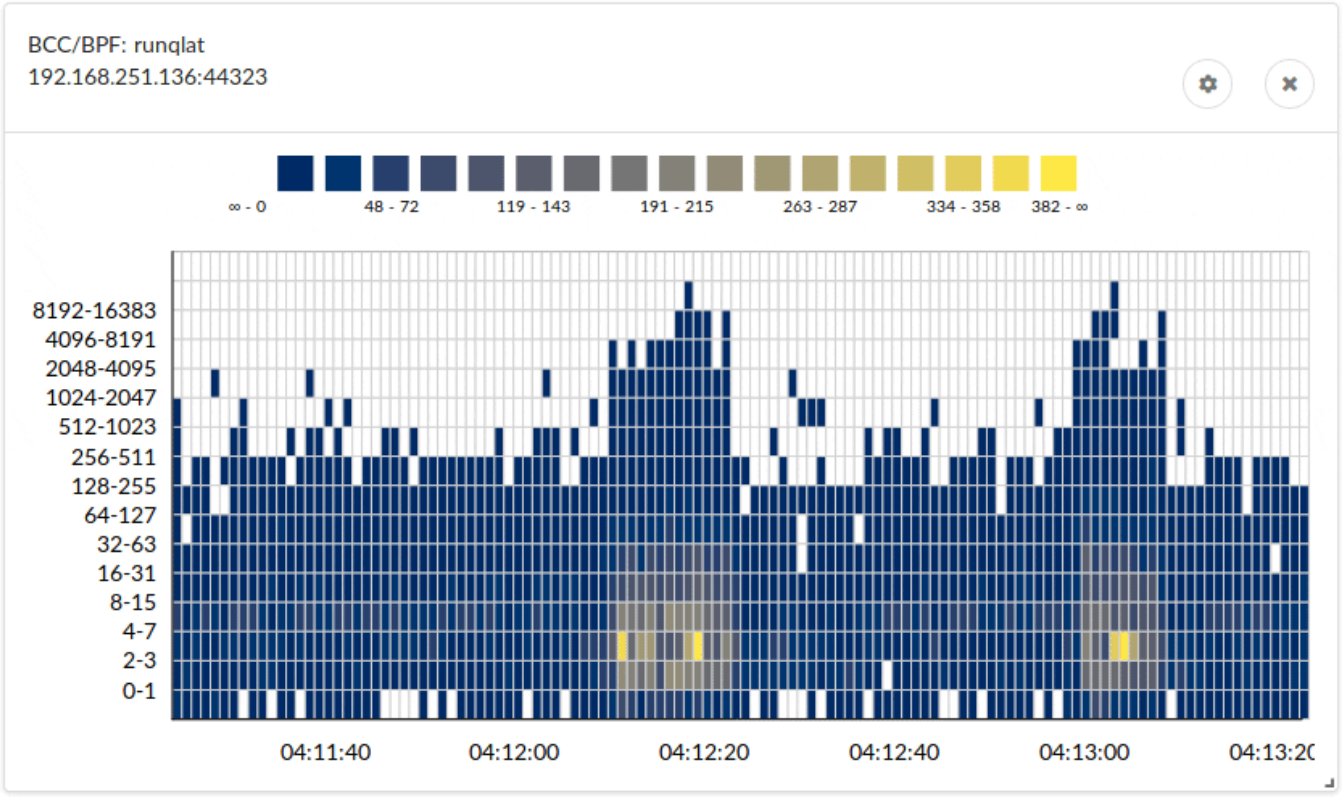
Issues

- All major capabilities exist
- Many minor things
- https://github.com/iovisor/bpftrace/issues



Other Tools

Netflix Vector: BPF heat maps



BCC/BPF: tcptop
192.168.251.136:44323

PID	COMM	LADDR	LPORT	DADDR	DPORT	RX_KB	TX_KB
1453	pmwebd	127.0.0.1	33546	127.0.0.1	44321	2	0
1453	pmwebd	192.168.251.136	44323	192.168.251.1	50674	1	1
1225	pmcd	127.0.0.1	44321	127.0.0.1	33546	0	2
1225	pmcd	127.0.0.1	44321	127.0.0.1	33546	1	2

https://medium.com/netflix-techblog/extending-vector-with-ebpf-to-inspect-host-and-container-performance-5da3af4c584b

Anticipated Worldwide Audience

- BPF Tool Developers:
  - Raw BPF: <20
  - C (or C++) BPF: ~20
  - bcc: >200
  - bpftrace: >5,000
- BPF Tool Users:
  - CLI tools (of any type): >20,000
  - GUIs (fronting any type): >200,000

Other Tools

- cloudflare/ebpf_exporter
- kubectl-trace
- sysdig eBPF support

Take Aways

Easily explore systems with bcc/bpftrace

Contribute: see bcc/bpftrace issue list

Share: posts, talks

URLs

- https://github.com/iovisor/bcc
  - https://github.com/iovisor/bcc/blob/master/docs/tutorial.md
  - https://github.com/iovisor/bcc/blob/master/docs/reference_guide.md
- https://github.com/iovisor/bpftrace
  - https://github.com/iovisor/bpftrace/blob/master/docs/tutorial_one_liners.md
  - https://github.com/iovisor/bpftrace/blob/master/docs/reference_guide.md

Thanks

- bpftrace
  - Alastair Robertson (creator)
  - Netflix: myself so for
  - Sthima: Mary Marchini, Willian Gaspar
  - Facebook: Jon Haslam, Dan Xu
  - Augusto Mecking Caringi, Dale Hamel, ...
- eBPF/bcc
  - Facebook: Alexei Starovoitov, Teng Qin, Yonghong Song, Martin Lau, Mark Drayton, …
  - Netflix: myself
  - VMware: Brenden Blanco
  - Sasha Goldsthein, Paul Chaignon, ...

NETFLIX