facebook

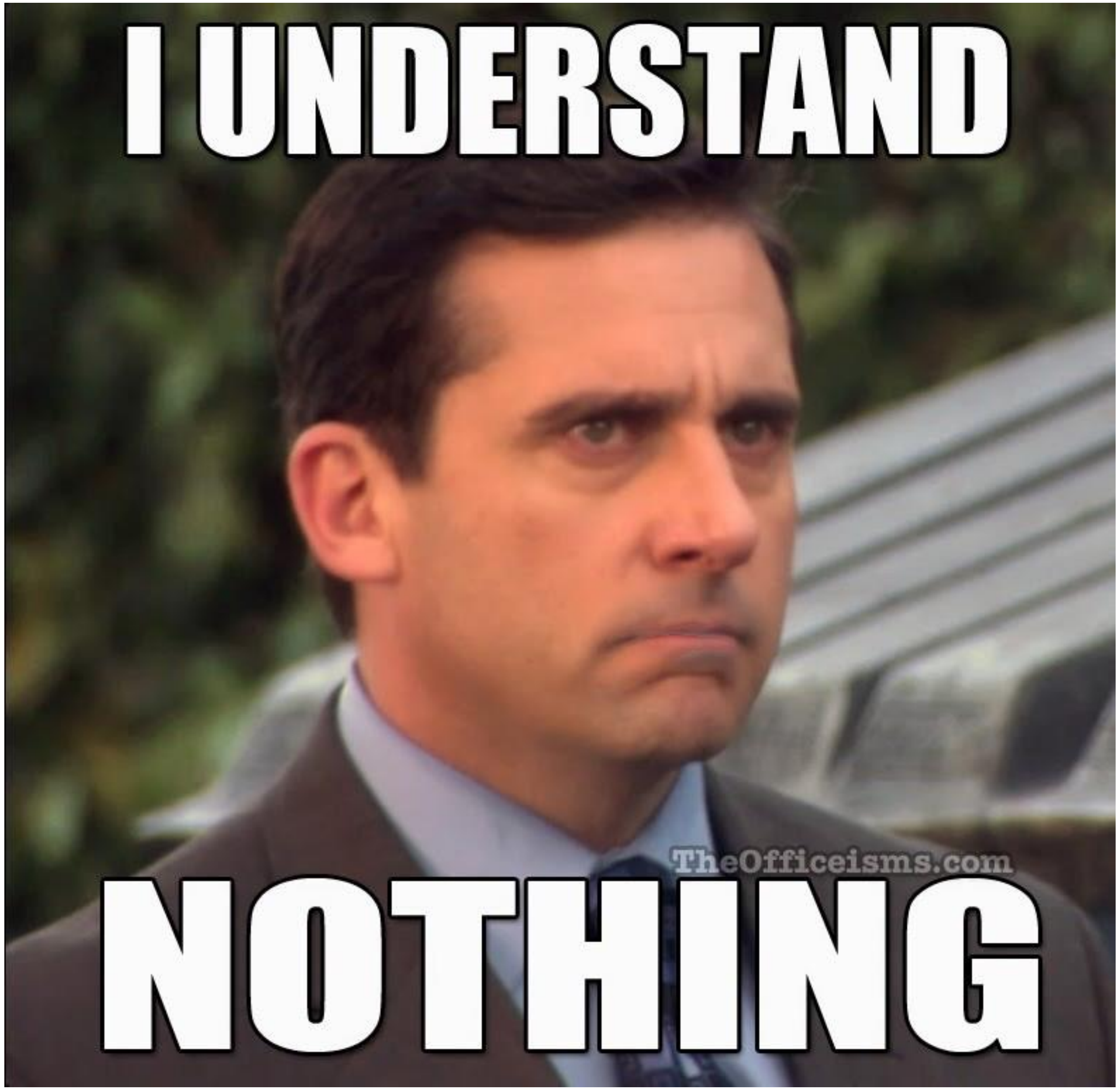
Using BPF for lightweight Android profiling

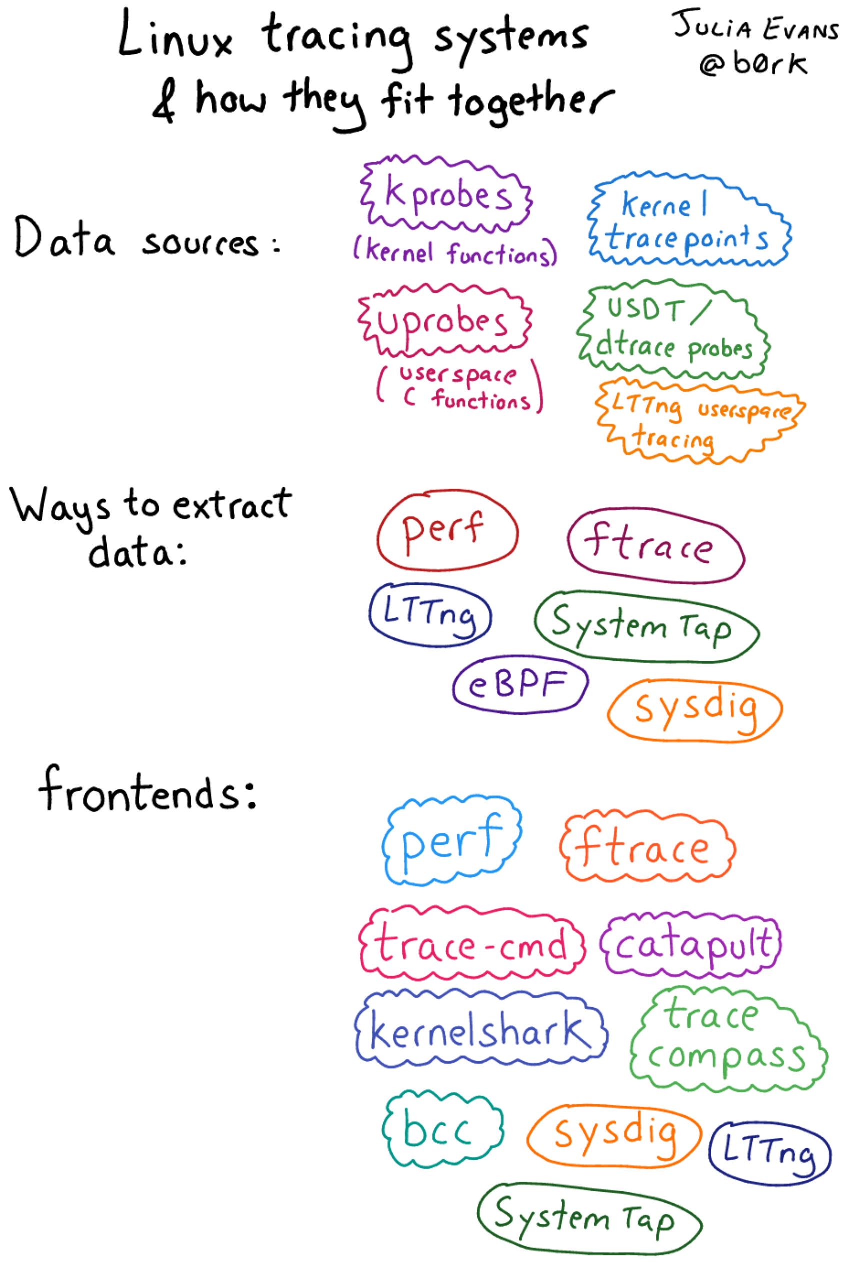
RIHAM SELIM

FACEBOOK

Linux Tracing: It’s confusing …

Kprobe?
Uprobe?
Tracepoint?
USDT?
BPF?
BCC?
BPFTrace?
Ftrace?
Perf
……..





What to collect?

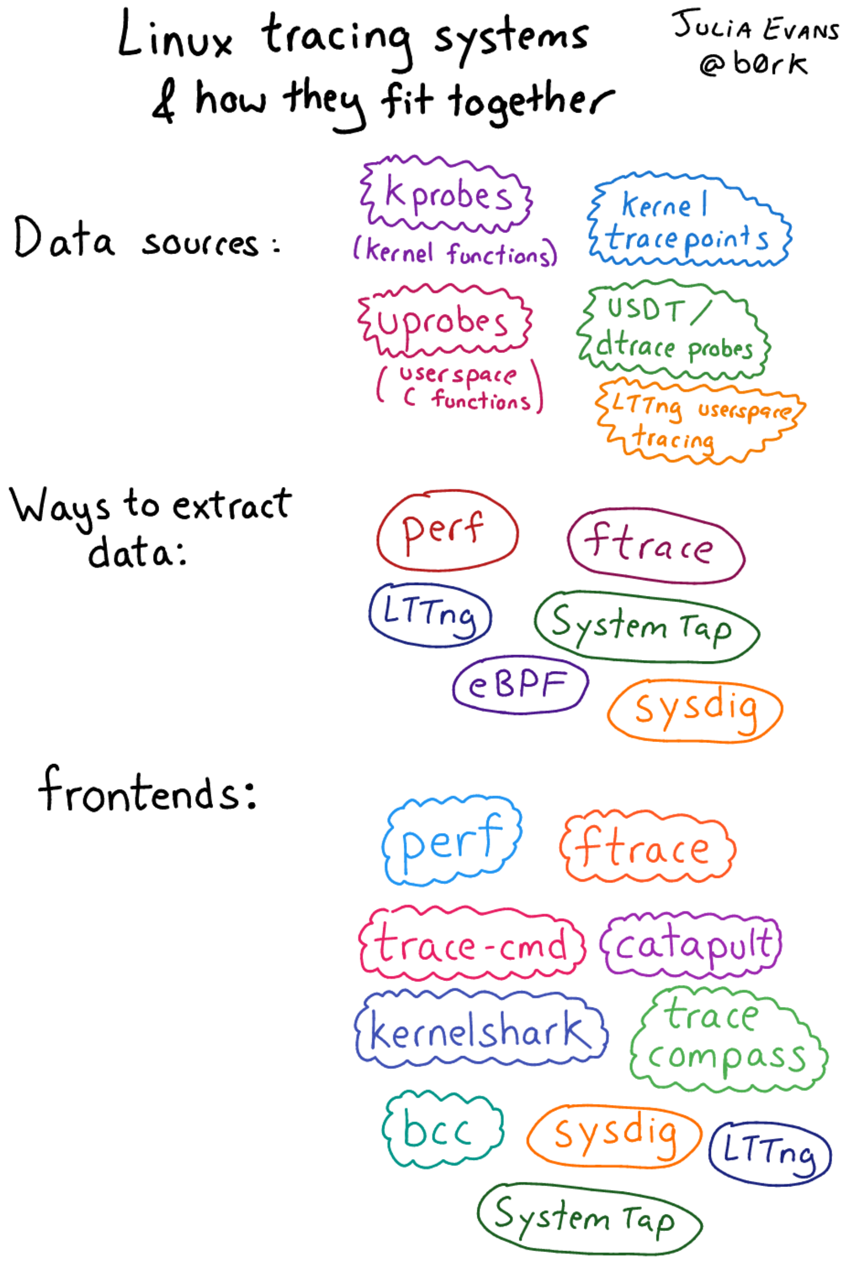
- Tracepoint
- USDT
- Kprobe
- Uprobe
- Software
- Hardware / PMCs

Way to extract data

- BPF
- Perf
- ftrace

Frontends: Configure and collect

- BCC
- BPFTrace
- Catapult
- Perf



What to collect?

- Tracepoint
- USDT
- Kprobe
- Uprobe
- Software
- Hardware / PMCs

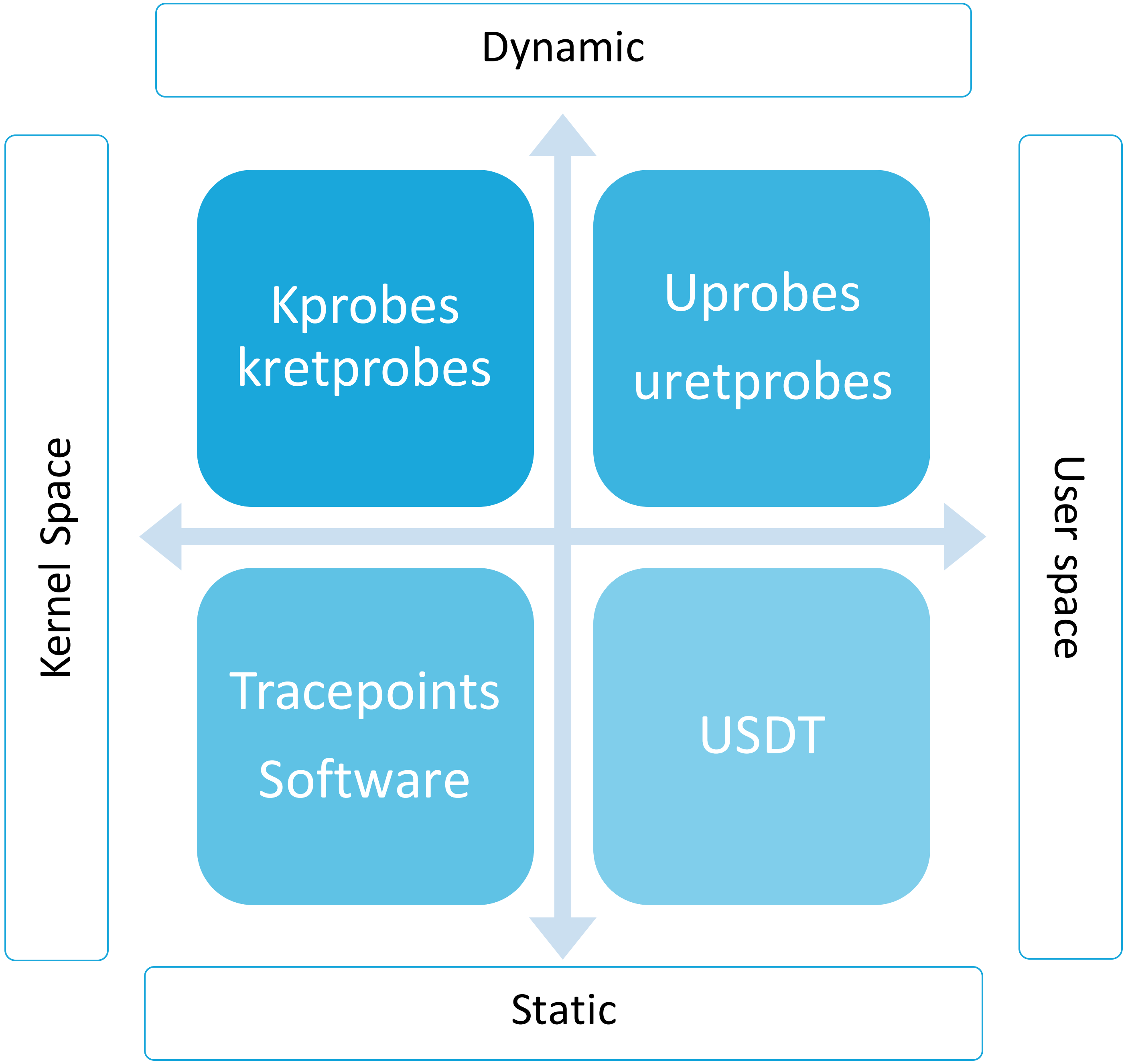
Way to extract data

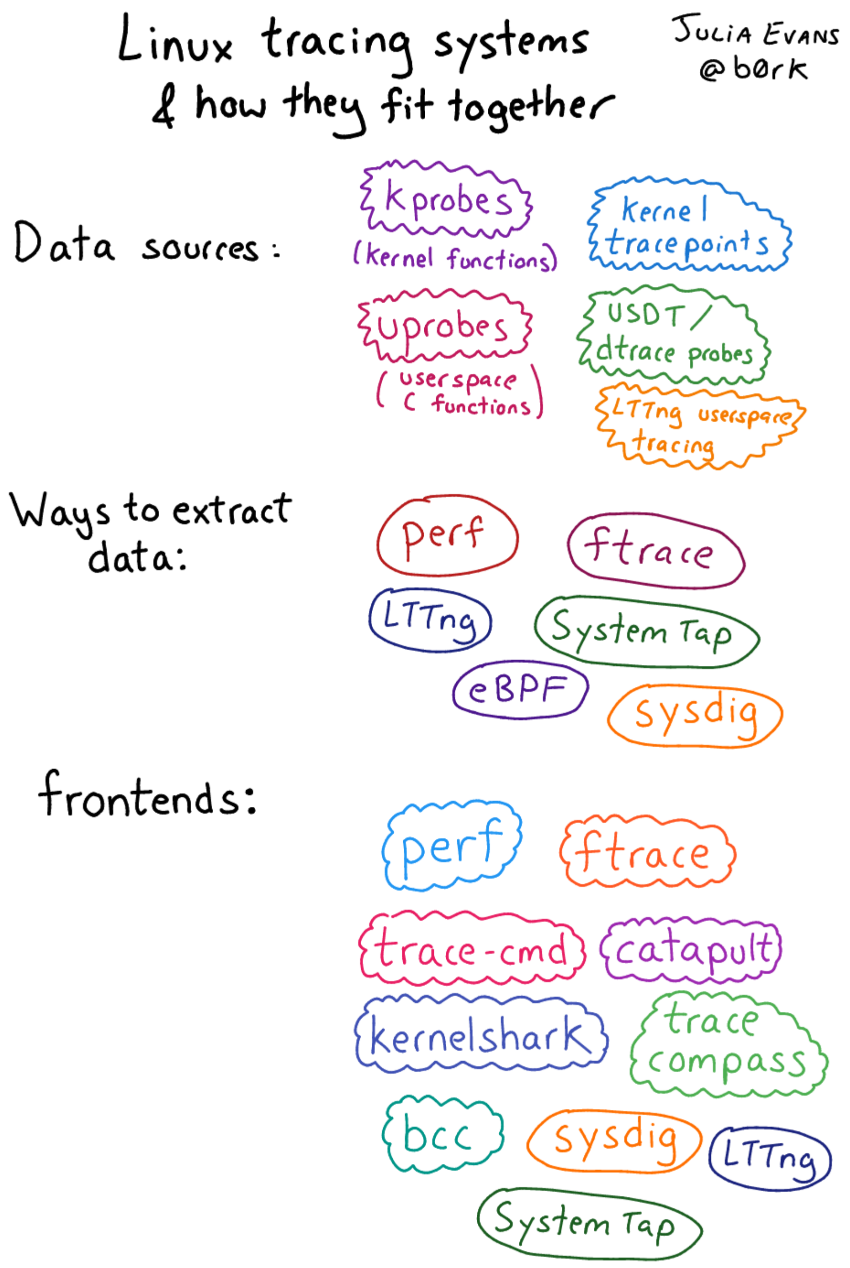
- BPF
- Perf
- ftrace

Frontends: Configure and collect

- BCC
- BPFTrace
- Catapult
- Perf

Probes


Dynamic
Kprobes
kretprobes
Uprobes
uretprobes
Kernel Space
User space
Tracepoints
Software
USDT
Static



What to collect?

- Tracepoint
- USDT
- Kprobe
- Uprobe
- Software
- Hardware / PMCs

Way to extract data

- BPF
- Perf
- ftrace

Frontends: Configure and collect

- BCC
- BPFTrace
- Catapult
- Perf

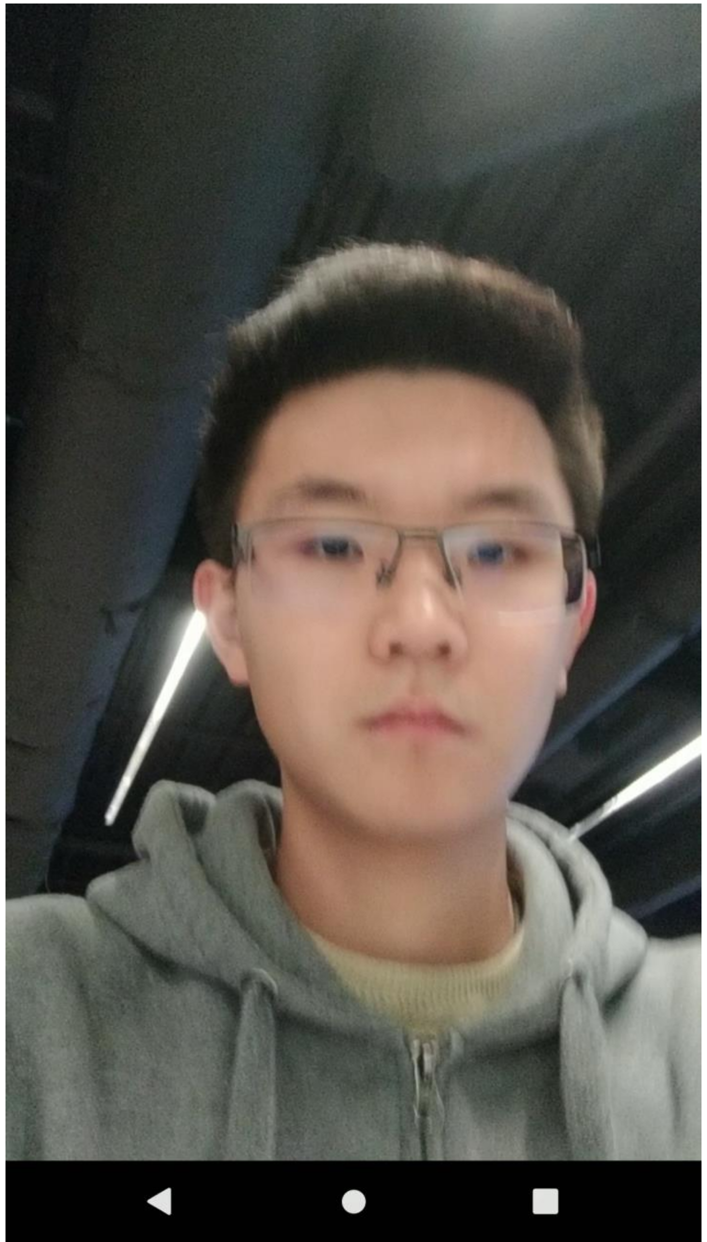
BPF

Linux kernel code execution engine.

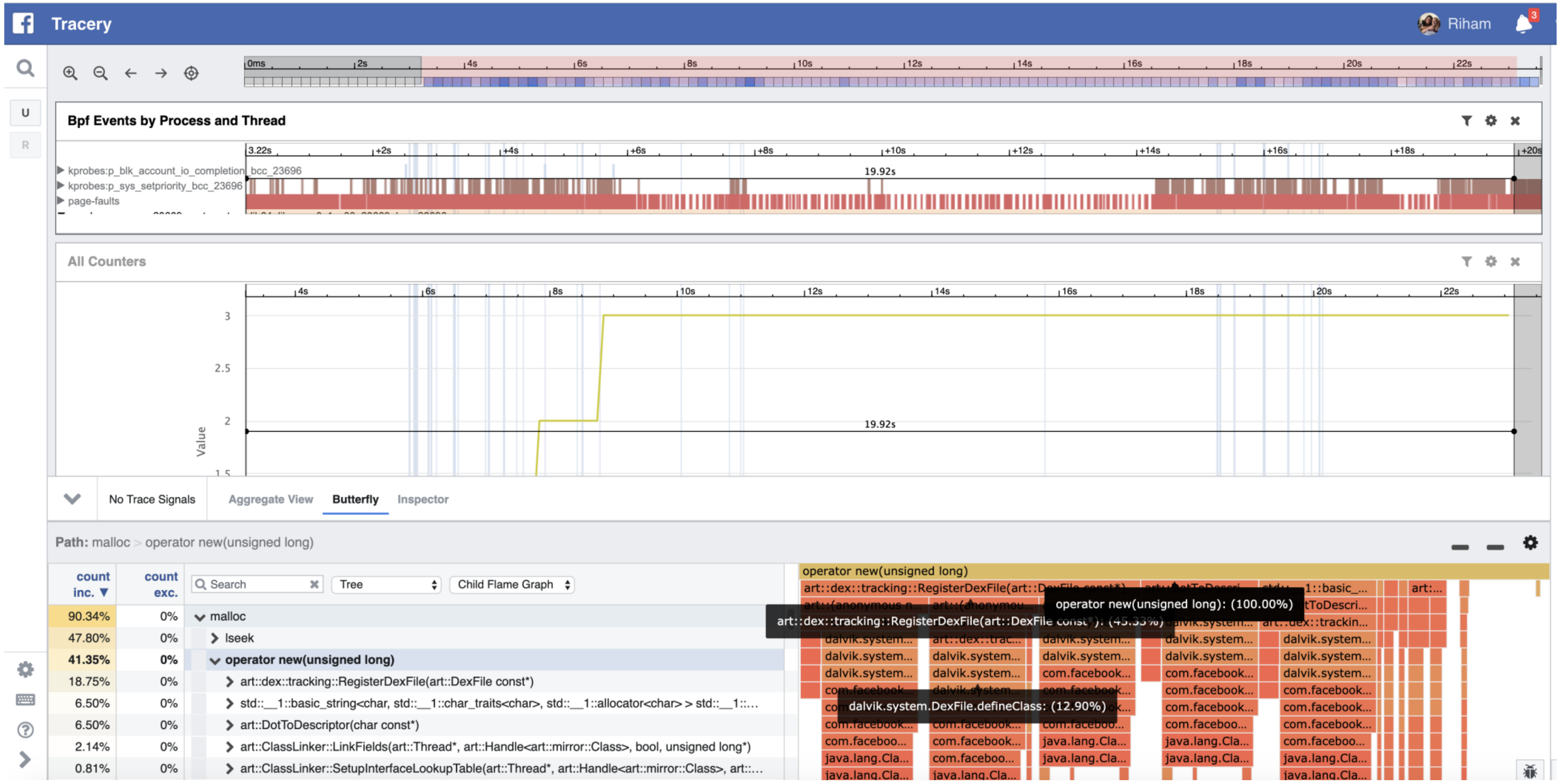
Allows running user provided code inside of the kernel.

Lightweight: minimally intrusive and virtually no cost when not enabled.

Tracking native memory allocations



Memory allocation Trace



Path: malloc > operator new(unsigned long)

count inc. ▼	count exc.	
90.34%	0%	malloc
47.80%	0%	lseek
41.35%	0%	operator new(unsigned long)
18.75%	0%	art::dex::tracking::RegisterDexFile(art::DexFile const*)
6.50%	0%	std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> > std::__1::...
6.50%	0%	art::DotToDescriptor(char const*)
2.14%	0%	art::ClassLinker::LinkFields(art::Thread*, art::Handle<art::mirror::Class>, bool, unsigned long*)
0.81%	0%	art::ClassLinker::SetupInterfaceLookupTable(art::Thread*, art::Handle<art::mirror::Class>, art::...

operator new(unsigned long): (100.00%)

art::dex::tracking::RegisterDexFile(art::DexFile const*): (45.35%)

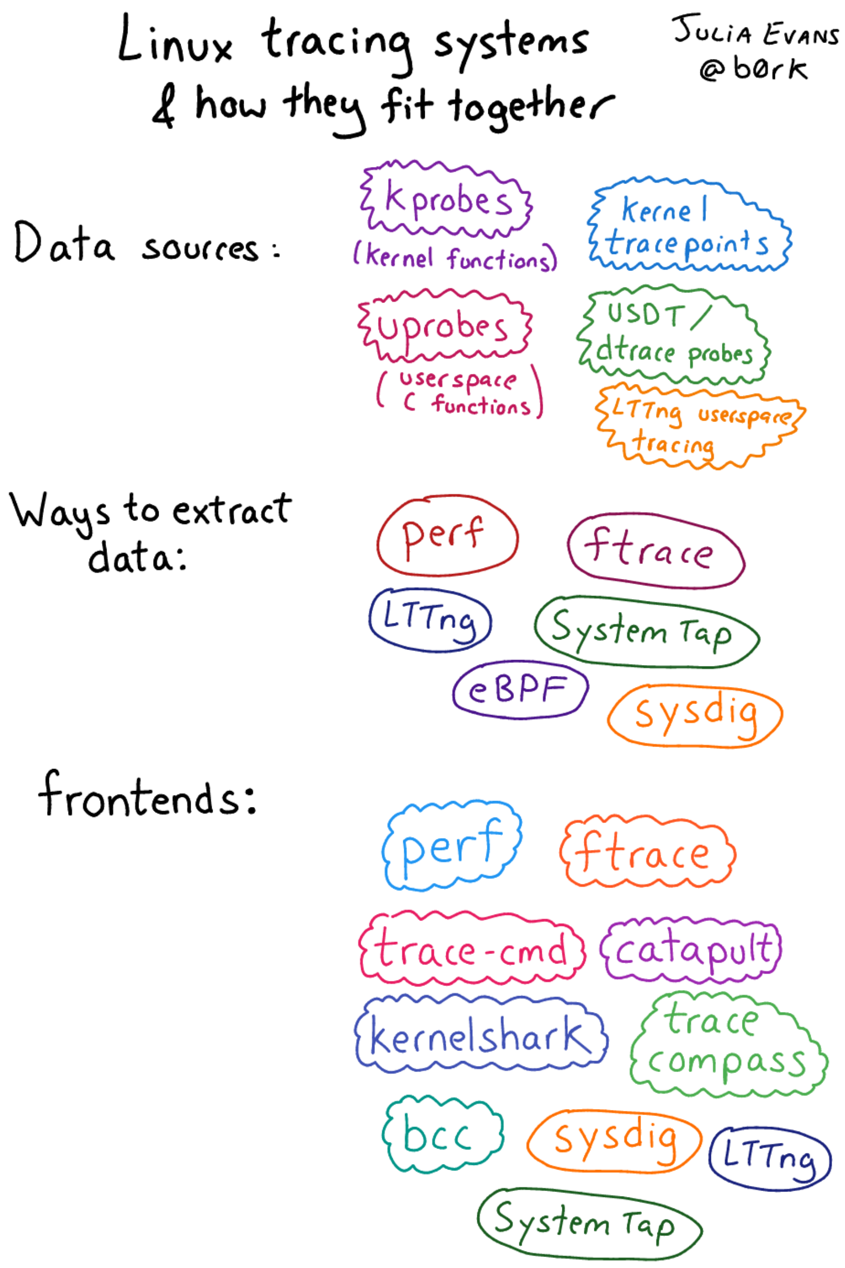
dalvik.system.DexFile.defineClass: (12.90%)

Facebook company

BPF Flavors

Counting

Sampling



What to collect?

- Tracepoint
- USDT
- Kprobe
- Uprobe
- Software
- Hardware / PMCs

Way to extract data

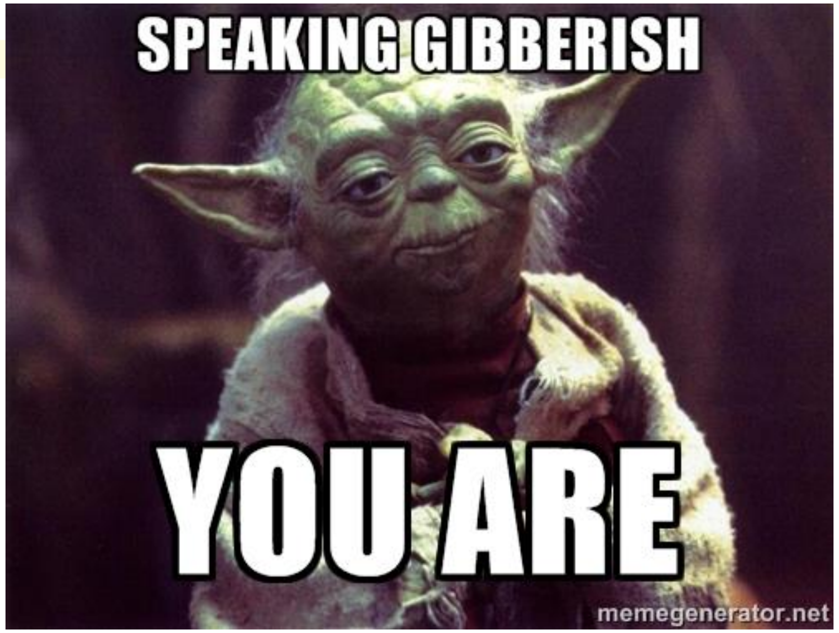
- BPF
- Perf
- ftrace

Frontends: Configure and collect

- BCC
- BPFTrace
- Catapult
- Perf

Writing BPF Programs

```
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(),
};
```



BPFTrace

```
# Files opened by process
bpftrace -e ‘tracepoint:syscalls:sys_enter_open { printf("%s %s\n", comm, str(args->filename)); }’


# Count page faults by process
bpftrace -e 'software:faults:1 { @[comm] = count(); }’

# Count LLC cache misses by process name and PID (uses PMCs):
bpftrace -e 'hardware:cache-misses:1000000 { @[comm, pid] = count(); }'
```

BCC

```
from bcc import BPF

# define BPF program
prog = """
int hello(void *ctx) {
    bpf_trace_printk("Hello, World!\\n");
    return 0;
}
"""

# load BPF program
b = BPF(text=prog)
b.attach_kprobe(event=b.get_syscall_fnname("clone"), fn_name="hello")

# header
print("%-18s %-16s %-6s %s" % ("TIME(s)", "COMM", "PID", "MESSAGE"))

# format output
while 1:
    try:
        (task, pid, cpu, flags, ts, msg) = b.trace_fields()
    except ValueError:
        continue
    print("%-18.9f %-16s %-6d %s" % (ts, task, pid, msg))
```

BPF ON ANDROID

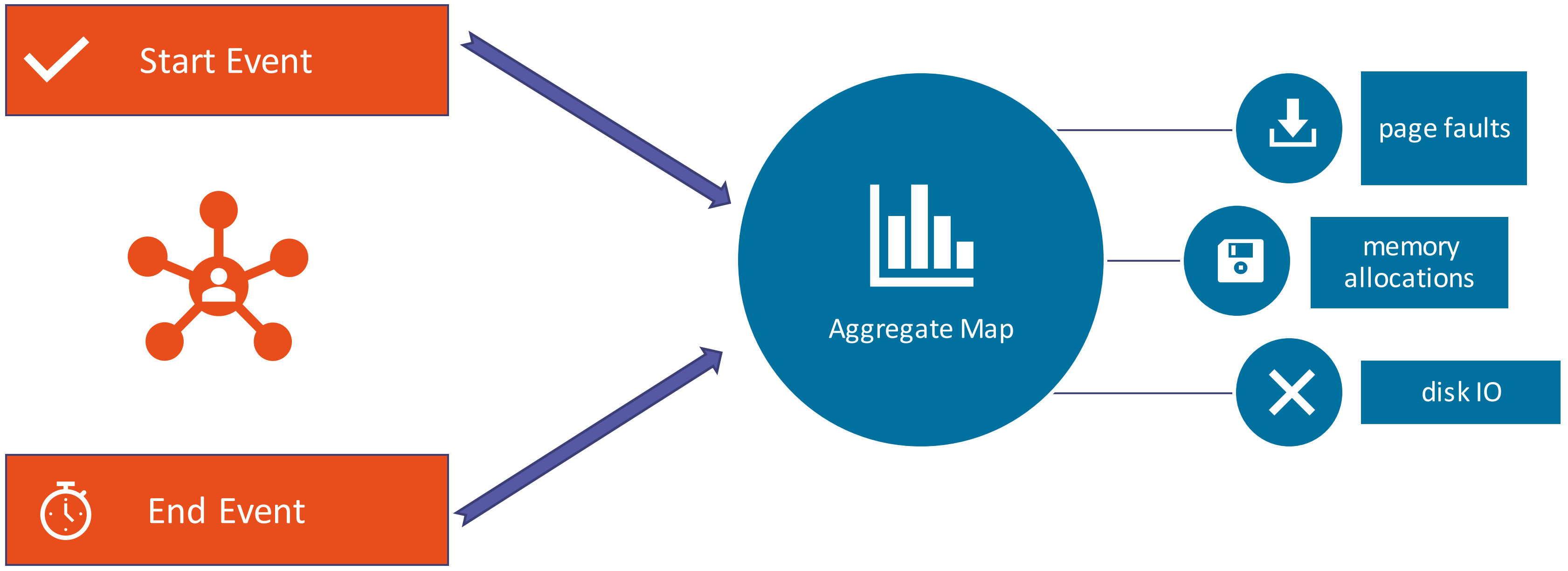
Tracking Consumption Metrics

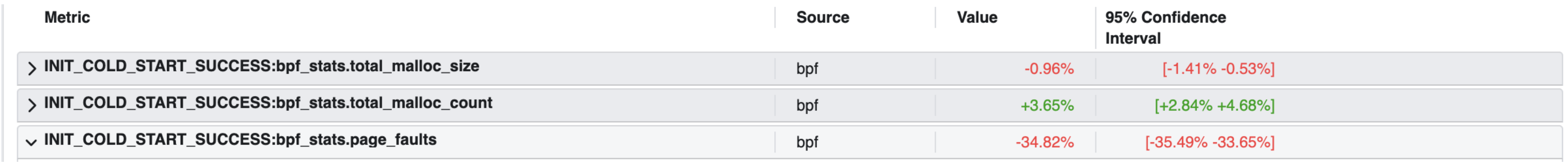
Dynamically tracking critical, limited resources with minimal overhead.

Super Bugs

Issues that have been open (or closed & reopened) for dozens of versions and have been investigated by many different engineers without successful resolution

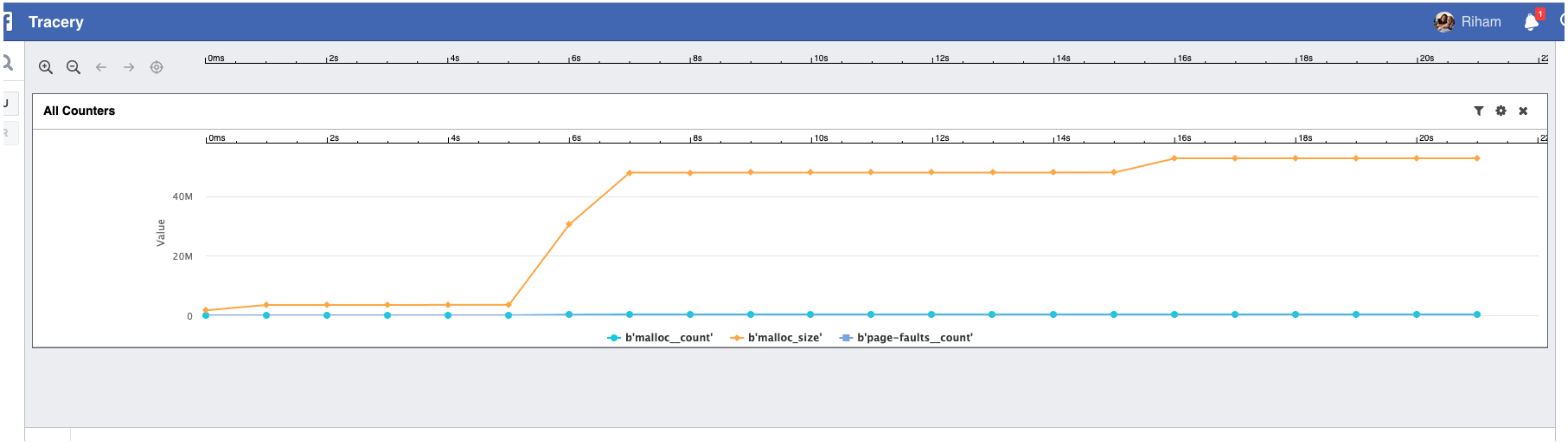
Tracking consumption Metrics





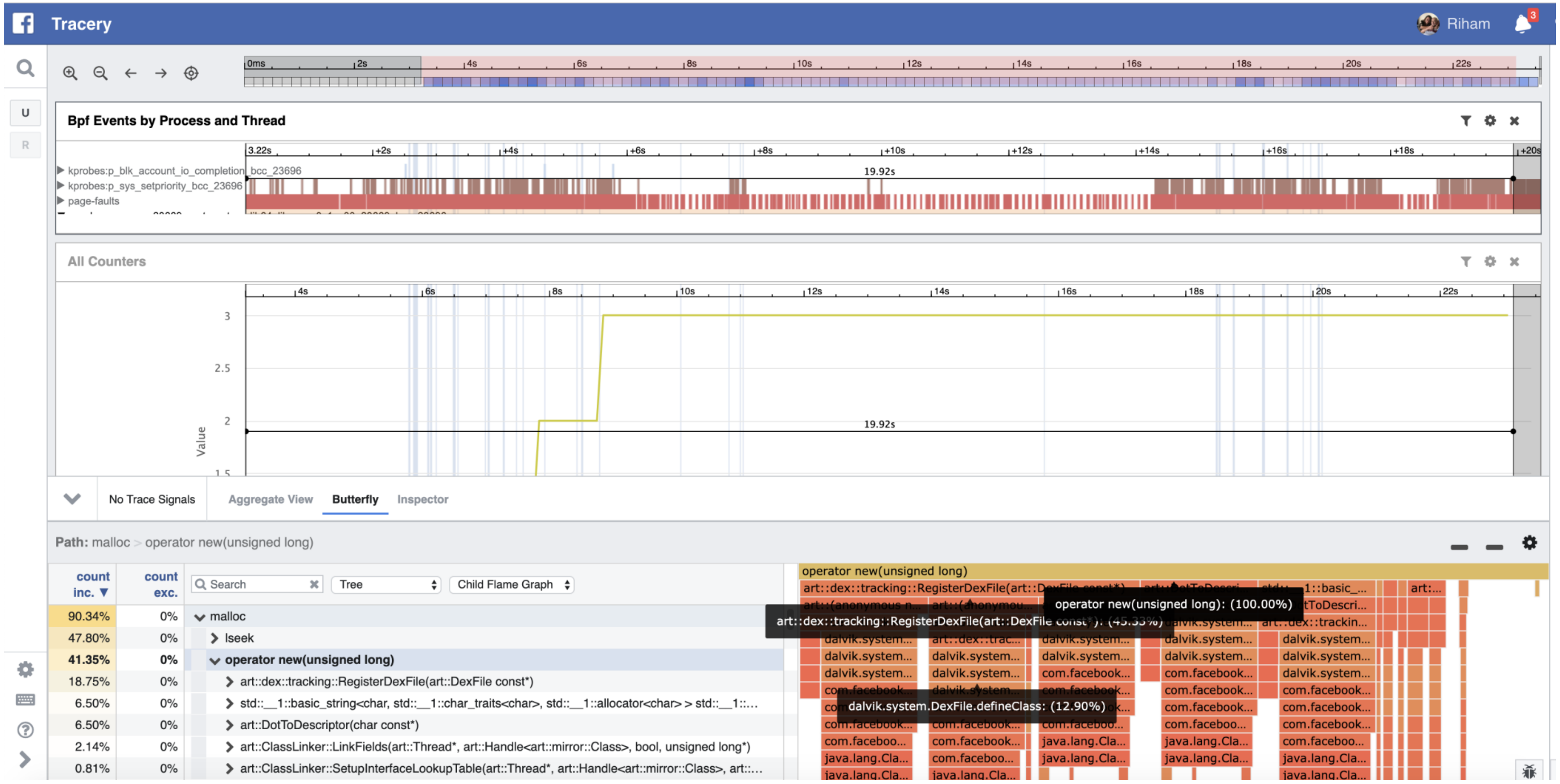
Metric	Source	Value	95% Confidence Interval
> INIT_COLD_START_SUCCESS:bpf_stats.total_malloc_size	bpf	-0.96%	[-1.41% -0.53%]
> INIT_COLD_START_SUCCESS:bpf_stats.total_malloc_count	bpf	+3.65%	[+2.84% +4.68%]
⌄ INIT_COLD_START_SUCCESS:bpf_stats.page_faults	bpf	-34.82%	[-35.49% -33.65%]

Tracking consumption Metrics



Correlate consumption with specific events.

Tracking consumption Metrics


Tracery
Riham
Bpf Events by Process and Thread
kprobes:p_blk_account_io_completion_bcc_23696
kprobes:p_sys_setpriority_bcc_23696
page-faults
19.92s
All Counters
Value
No Trace Signals
Aggregate View
Butterfly
Inspector
operator new(unsigned long): (100.00%)
art::dex::tracking::RegisterDexFile(art::DexFile const*): (45.35%)
dalvik.system.DexFile.defineClass: (12.90%)

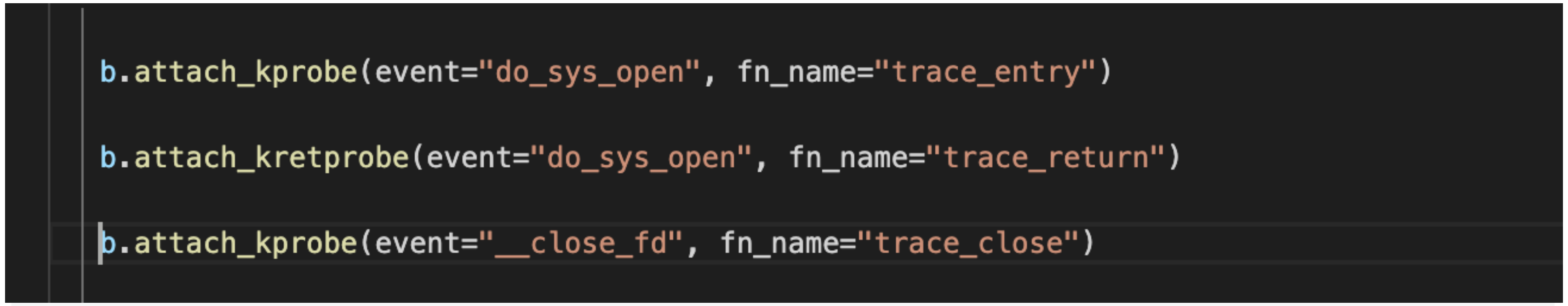
Path: malloc > operator new(unsigned long)

count inc. ▼	count exc.	
90.34%	0%	malloc
47.80%	0%	lseek
41.35%	**0%**	**operator new(unsigned long)**
18.75%	0%	art::dex::tracking::RegisterDexFile(art::DexFile const*)
6.50%	0%	std::__1::basic_string<char, std::__1::char_traits<char>, std::__1::allocator<char> > std::__1::...
6.50%	0%	art::DotToDescriptor(char const*)
2.14%	0%	art::ClassLinker::LinkFields(art::Thread*, art::Handle<art::mirror::Class>, bool, unsigned long*)
0.81%	0%	art::ClassLinker::SetupInterfaceLookupTable(art::Thread*, art::Handle<art::mirror::Class>, art::...

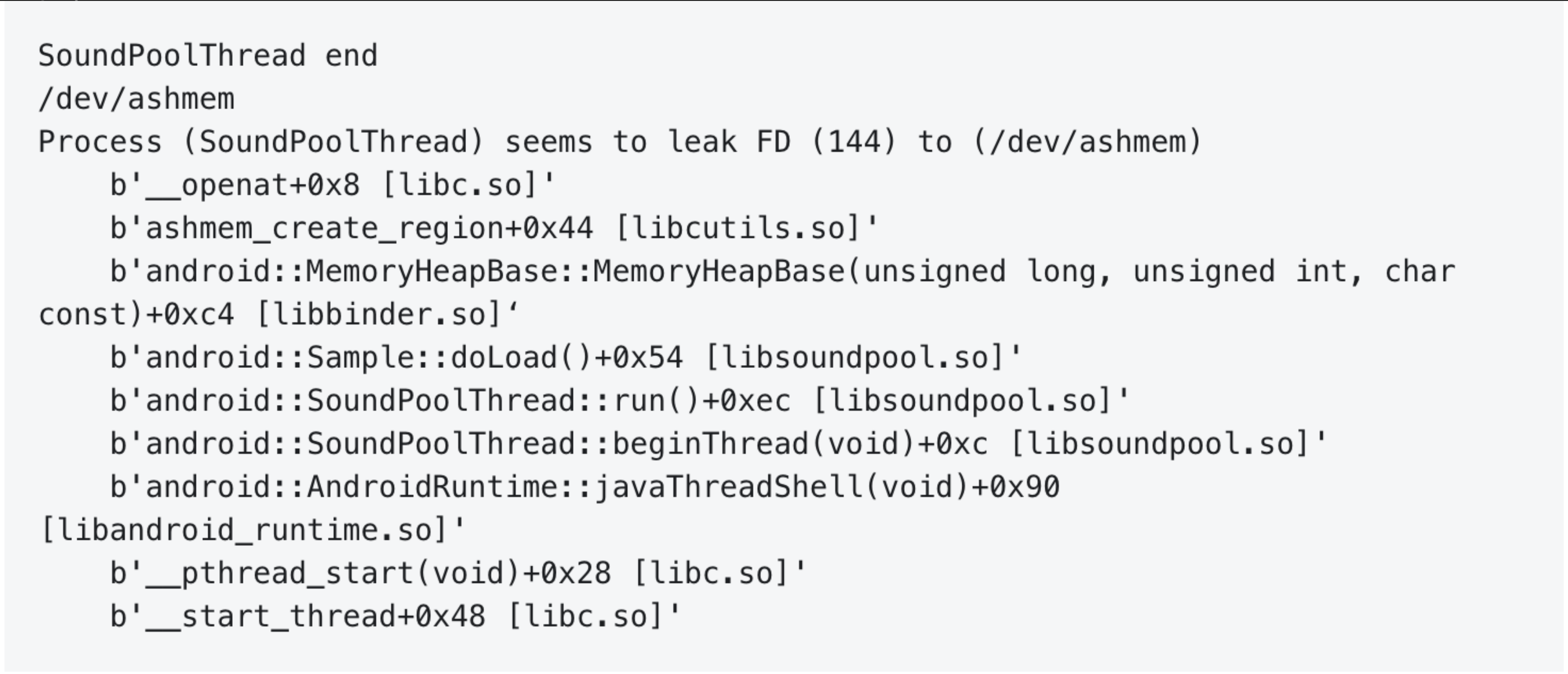
Facebook company

Super Bugs

Resource Leaks

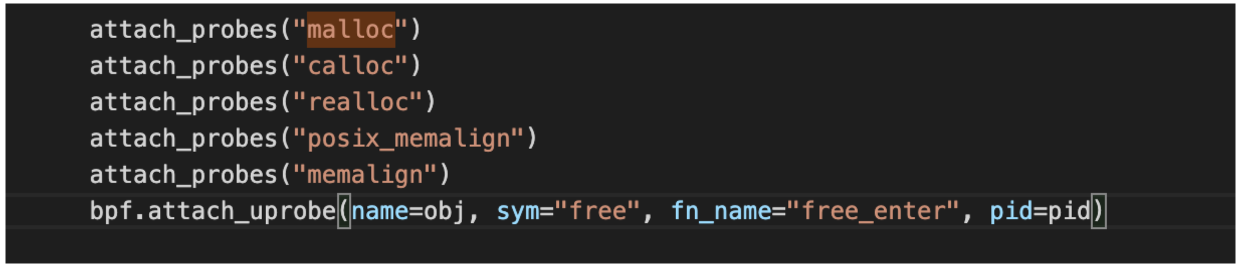


```
b.attach_kprobe(event="do_sys_open", fn_name="trace_entry")

b.attach_kretprobe(event="do_sys_open", fn_name="trace_return")

b.attach_kprobe(event="__close_fd", fn_name="trace_close")
```



```
SoundPoolThread end
/dev/ashmem
Process (SoundPoolThread) seems to leak FD (144) to (/dev/ashmem)
    b'__openat+0x8 [libc.so]'
    b'ashmem_create_region+0x44 [libcutils.so]'
    b'android::MemoryHeapBase::MemoryHeapBase(unsigned long, unsigned int, char
const)+0xc4 [libbinder.so]‘
    b'android::Sample::doLoad()+0x54 [libsoundpool.so]'
    b'android::SoundPoolThread::run()+0xec [libsoundpool.so]'
    b'android::SoundPoolThread::beginThread(void)+0xc [libsoundpool.so]'
    b'android::AndroidRuntime::javaThreadShell(void)+0x90
[libandroid_runtime.so]'
    b'__pthread_start(void)+0x28 [libc.so]'
    b'__start_thread+0x48 [libc.so]'
```

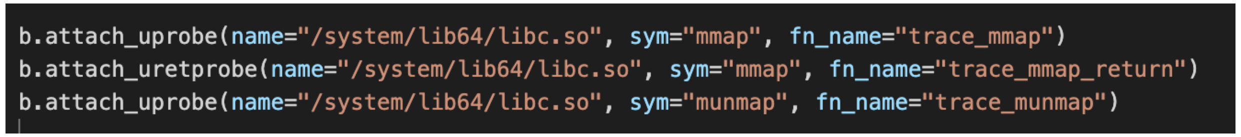
Memory Leaks



```
    attach_probes("malloc")
    attach_probes("calloc")
    attach_probes("realloc")
    attach_probes("posix_memalign")
    attach_probes("memalign")
    bpf.attach_uprobe(name=obj, sym="free", fn_name="free_enter", pid=pid)
```

```
2893664 bytes in 1172 allocations from stack
    EsxMergedRectList::Create(EsxSettings const*, int)+0x24
[libGLESv2_adreno.so]
    EsxFramebufferObject::Init(EsxFramebufferObjectCreateData*)+0x224
[libGLESv2_adreno.so]
    EsxContext::GlBindFramebuffer(unsigned int, unsigned int)+0xf0
[libGLESv2_adreno.so]
    [unknown] [libhwui.so]
    GrRenderTargetContext::getRTOpList()+0x64 [libhwui.so]
    GrRenderTargetContext::GrRenderTargetContext(GrContext*, GrDrawingManager*,
sk_sp<GrRenderTargetProxy>, sk_sp<SkColorSpace>, SkSurfaceProps const*,
GrAuditTrail*, GrSingleOwner*, bool)+0xe0 [libhwui.so]
    [unknown] [libhwui.so]
    GrContext::makeDeferredRenderTargetContext(SkBackingFit, int, int,
GrPixelConfig, sk_sp<SkColorSpace>, int, GrMipMapped, GrSurfaceOrigin,
SkSurfaceProps const*, SkBudgeted)+0xb8 [libhwui.so]
    [unknown] [libhwui.so]
    [unknown] [libhwui.so]
    SkSurface::MakeRenderTarget(GrContext*, SkBudgeted, SkImageInfo const&, int,
GrSurfaceOrigin, SkSurfaceProps const*, bool)+0x94 [libhwui.so]
```

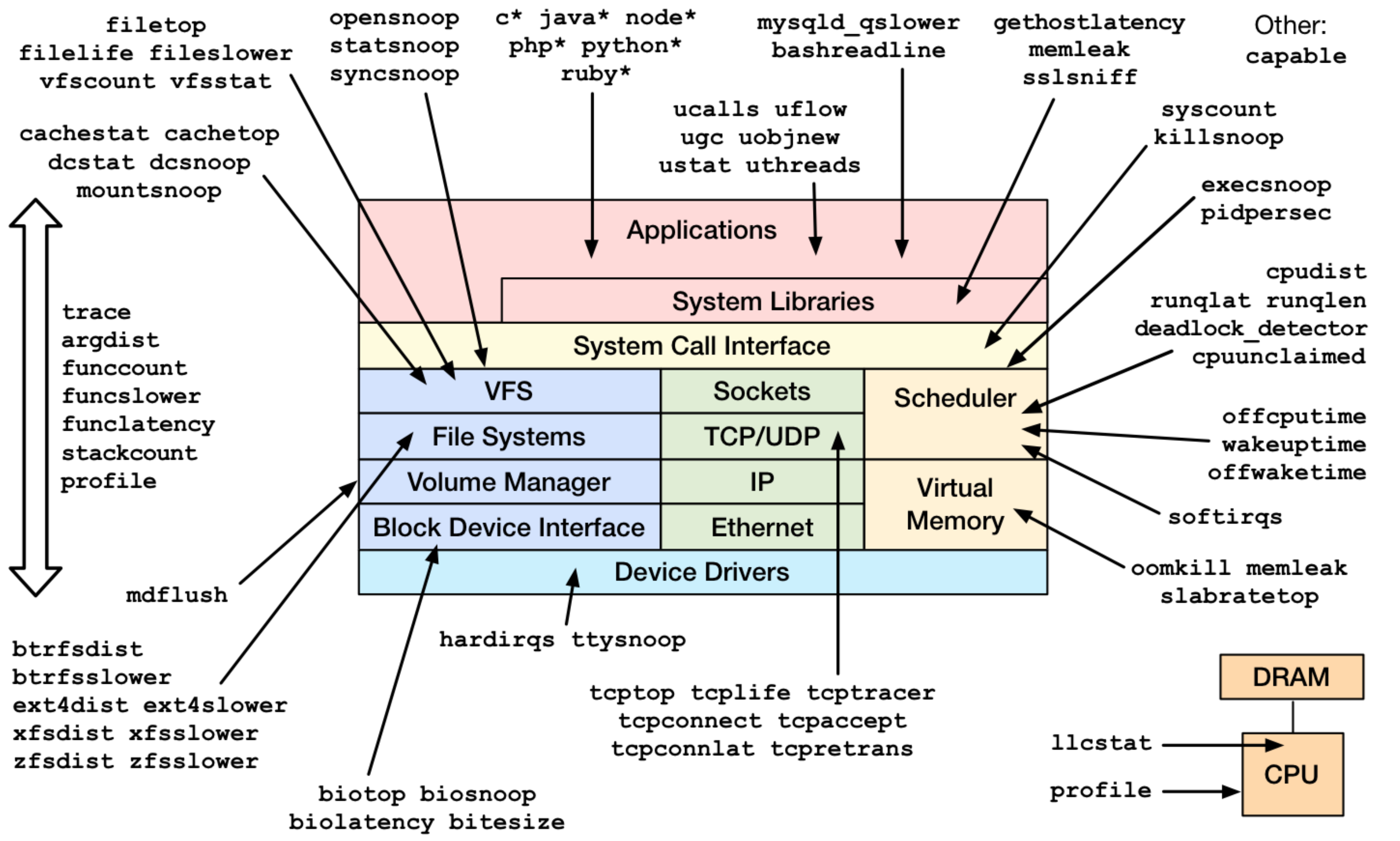
Memory Corruption



```
b.attach_uprobe(name="/system/lib64/libc.so", sym="mmap", fn_name="trace_mmap")
b.attach_uretprobe(name="/system/lib64/libc.so", sym="mmap", fn_name="trace_mmap_return")
b.attach_uprobe(name="/system/lib64/libc.so", sym="munmap", fn_name="trace_munmap")
```

```
MMAP ERRORS
Liger-EventBase Mismatched mmap unmap at 494576381952, mmap size (287760),
munmap size (287776)
MMAP Stack:
   b'mobileconfig::FBMobileConfigFileUtils::mmapFile(std::string const&,
unsigned char const, unsigned long, mobileconfig::FBMobileConfigLogger)+0x4fc
[libxplat_mobileconfig_FBMobileConfigCore_FBMobileConfigCoreAndroid.so]'
   b'mobileconfig::FBMobileConfigValueStore::mmapFile(std::string const&)+0xa0
[libxplat_mobileconfig_FBMobileConfigCore_FBMobileConfigCoreAndroid.so]'
```

Linux bcc/BPF Tracing Tools



https://github.com/iovisor/bcc#tools 2018

The Future

- Mobile Lab
- Sapienz
- Production

Thank you

FACEBOOK

Questions?

FACEBOOK