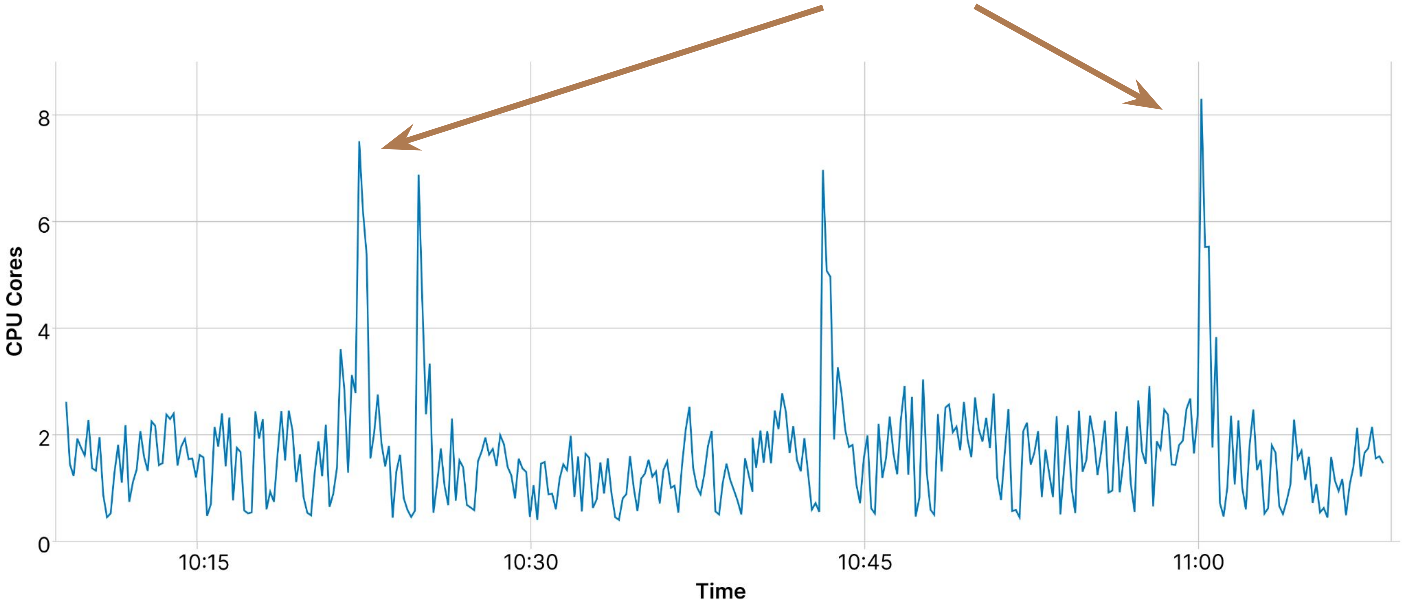
What's possible in observability when we have frame pointers

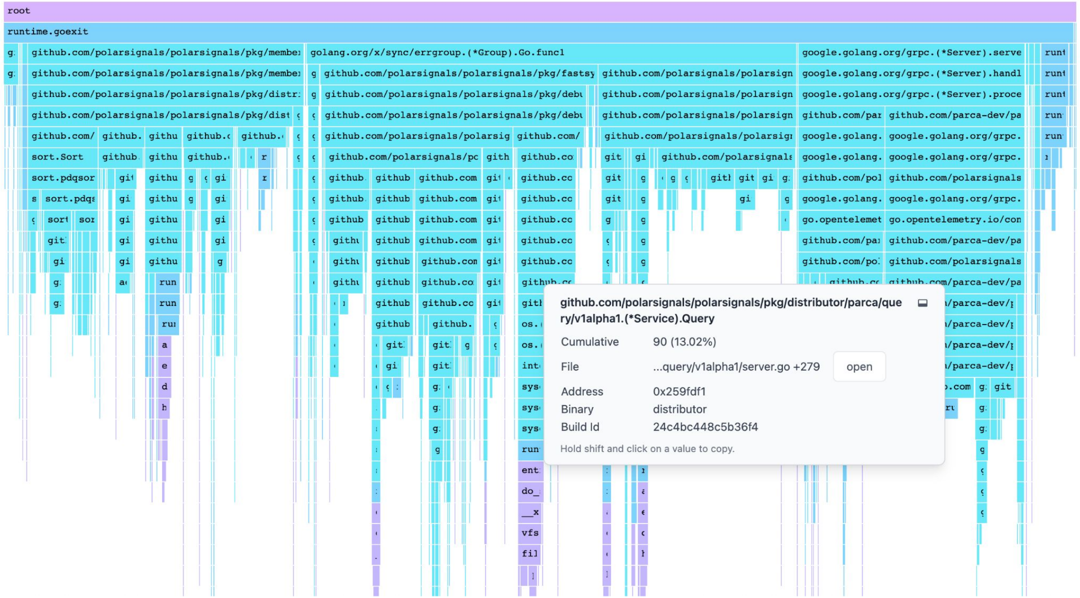
Matthias Loibl / Jon Seager

Continuous Profiling

What happened at the spikes?! 🤔



Continuous Profiling


root
runtime.goexit
github.com/polarsignals/polarsignals/pkg/member
golang.org/x/sync/errgroup.(*Group).Go.func1
google.golang.org/grpc.(*Server).serve
google.golang.org/grpc.(*Server).handl
google.golang.org/grpc.(*Server).proce
github.com/polarsignals/polarsignals/pkg/distr
github.com/polarsignals/polarsignals/pkg/fastsy
github.com/polarsignals/polarsignals/pkg/debu
github.com/parca-dev/pa
go.opentelemetry.io/con
sort.Sort
sort.pdqsor
open

github.com/polarsignals/polarsignals/pkg/distributor/parca/query/v1alpha1.(*Service).Query

Cumulative 90 (13.02%)

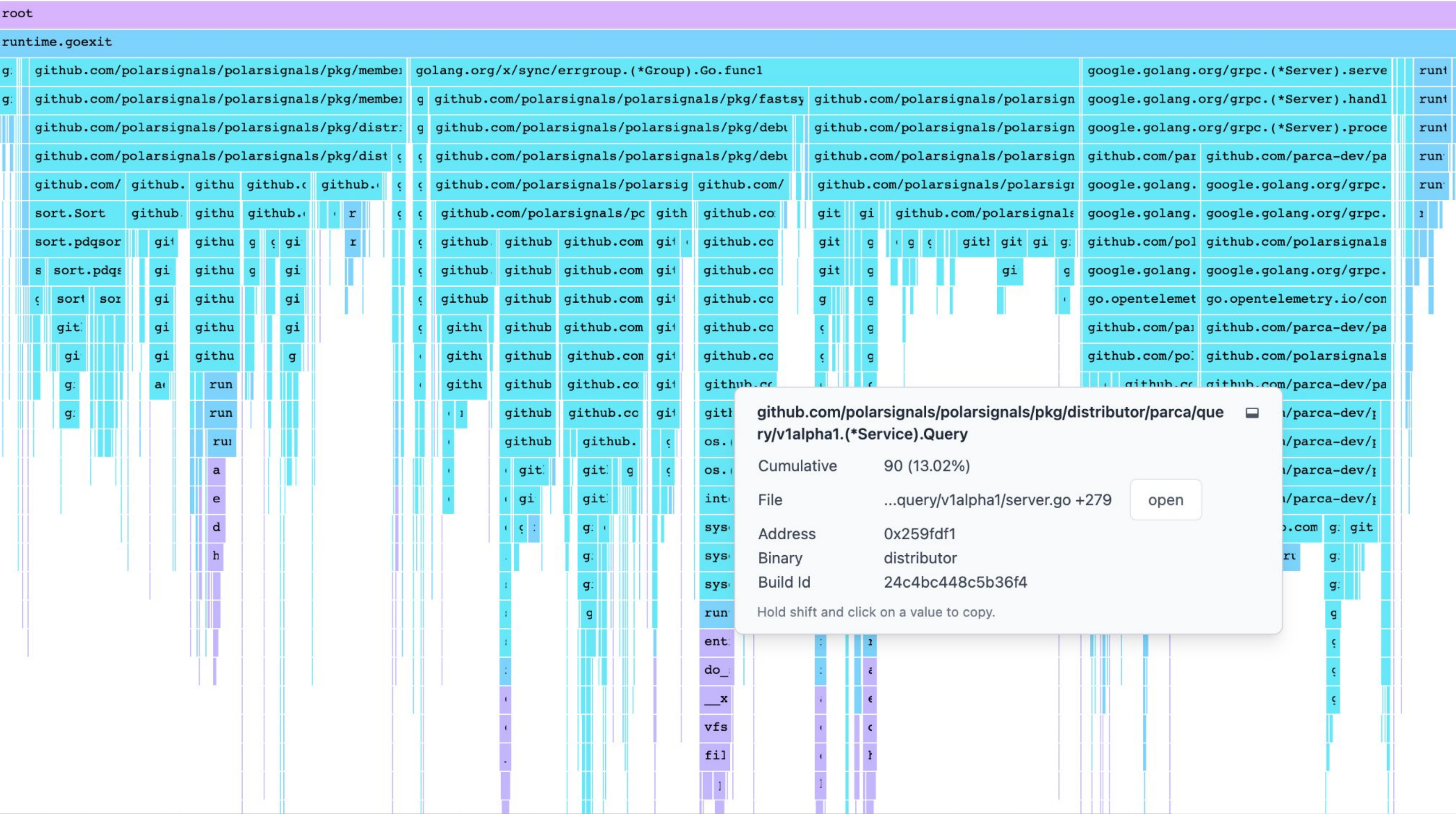
File ...query/v1alpha1/server.go +279

Address 0x259fdf1

Binary distributor

Build Id 24c4bc448c5b36f4

Hold shift and click on a value to copy.


root
runtime.goexit
github.com/polarsignals/polarsignals/pkg/membe
golang.org/x/sync/errgroup.(*Group).Go.func1
google.golang.org/grpc.(*Server).serve
github.com/polarsignals/polarsignals/pkg/fastsy
github.com/polarsignals/polarsign
google.golang.org/grpc.(*Server).handl
github.com/polarsignals/polarsignals/pkg/distr
github.com/polarsignals/polarsignals/pkg/debu
google.golang.org/grpc.(*Server).proce
github.com/parca-dev/pa
sort.Sort
sort.pdqsor
sort.pdqs
go.opentelemet
go.opentelemetry.io/con
github.com/polarsignals

github.com/polarsignals/polarsignals/pkg/distributor/parca/query/v1alpha1.(*Service).Query

Cumulative 90 (13.02%)

File ...query/v1alpha1/server.go +279 open

Address 0x259fdf1

Binary distributor

Build Id 24c4bc448c5b36f4

Hold shift and click on a value to copy.

About us

Matthias Loibl

@metalmatze@social.metalmatze.de

@metalmatze

Jon Seager

@jnsgruk@hachyderm.io

@jnsgruk

- Senior Software Engineer at Polar Signals
- Open-Source Maintainer
  - Parca
  - Thanos
  - Prometheus
  - Prometheus Operator
  - Pyrra

- VP Engineering at Canonical
- Leads development of Juju & charms
  - Observability
  - Identity
  - Data Platform
  - MLOps
  - Telco

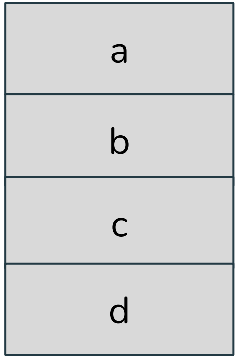
What is profiling data made up of?

t1	t2	t3
a	a	a
b	b	b
c	c	c
d	e	d

=

a;b;c;d 20ms
a;b;c;e 10ms

How do we get a stack?



?

Best Case: Frame Pointers

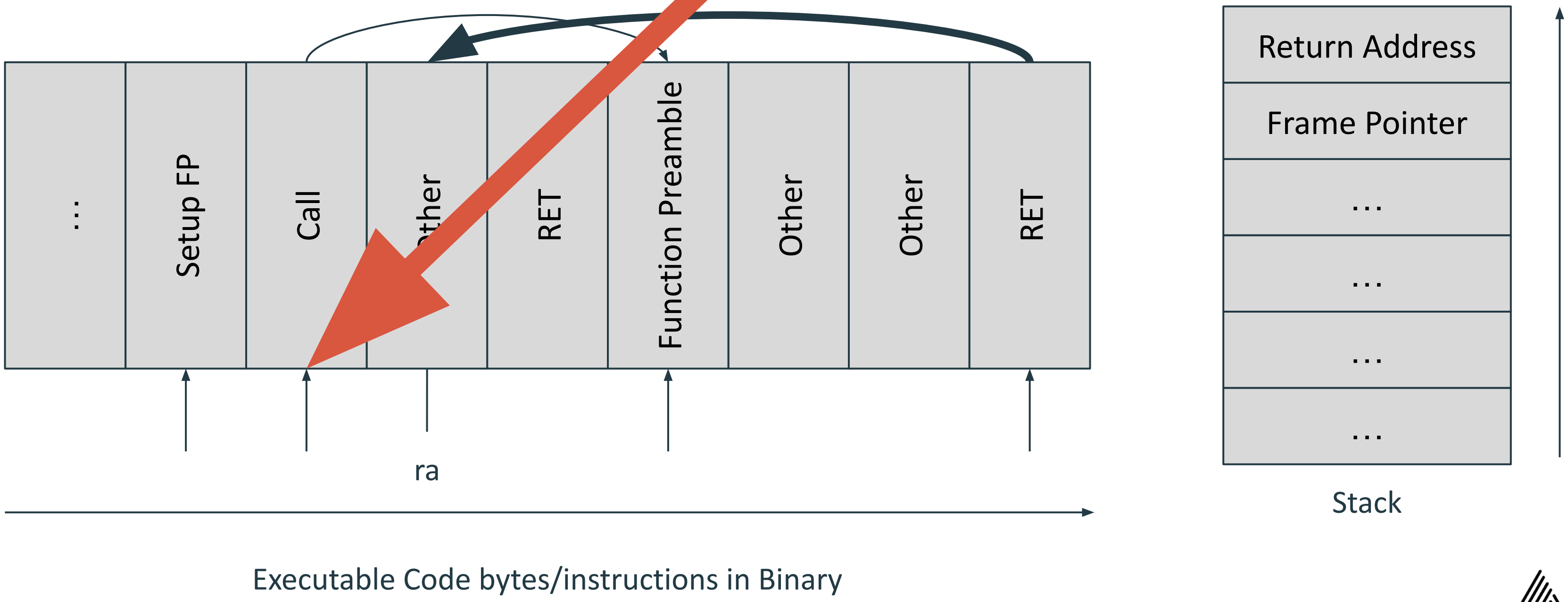
What are frame pointers?

```c
int top(void) {
    for(;;) { }
}
int c1(void) {
    top();
}
int b1(void) {
    c1();
}
int a1(void) {
    b1();
}
int main(void) {
  a1();
}
```

```
# compiled with `gcc sample.c -o sample_with_frame_pointers -fno-omit-frame-pointer`
$ objdump -d ./sample_with_frame_pointers


0000000000401106 <top>:
  401106:       55                      push   %rbp
  401107:       48 89 e5                mov    %rsp,%rbp
  40110a:       eb fe                   jmp    40110a <top+0x4>

000000000040110c <c1>:
  40110c:       55                      push   %rbp
  40110d:       48 89 e5                mov    %rsp,%rbp
  401110:       e8 f1 ff ff ff          call   401106 <top>
  401115:       90                      nop
  401116:       5d                      pop    %rbp
  401117:       c3                      ret

0000000000401118 <b1>:
  401118:       55                      push   %rbp
  401119:       48 89 e5                mov    %rsp,%rbp
  40111c:       e8 eb ff ff ff          call   40110c <c1>
  401121:       90                      nop
  401122:       5d                      pop    %rbp
  401123:       c3                      ret

...
```

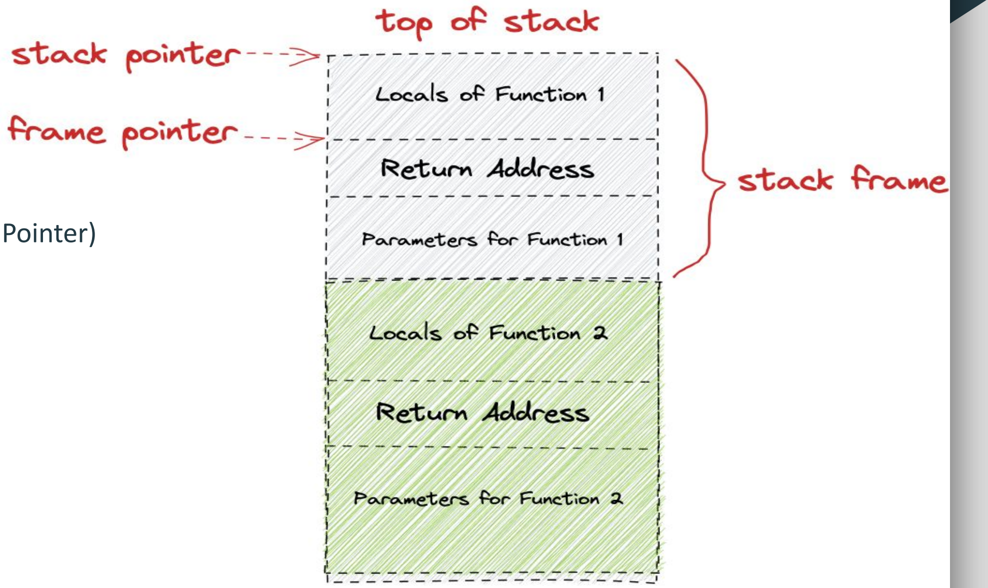
Binary

ra - 1 is our caller



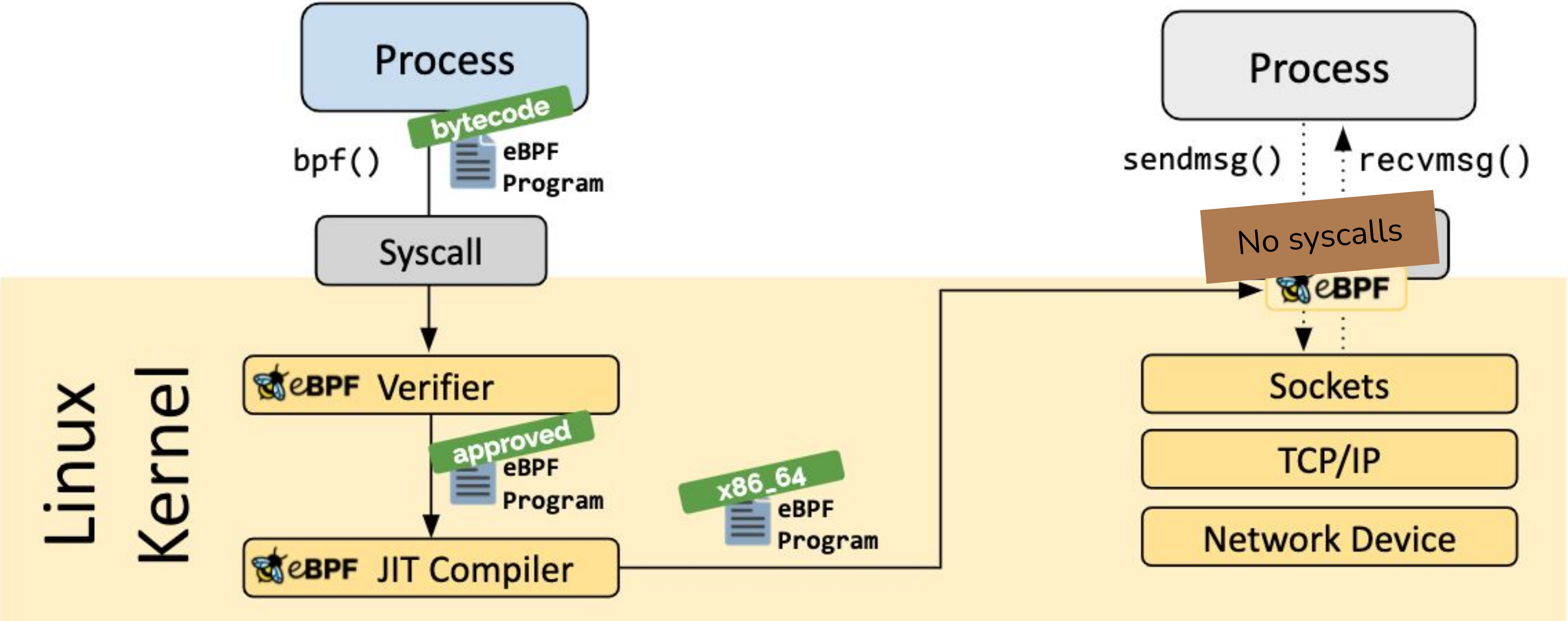
Walking the Stacks

- Core registers
  - Keep track of process state
    - **PC** (program Counter)/ IP (Instruction Pointer)
    - **rbp/fp**: base pointer/frame pointer
    - **rsp/sp**: stack pointer
    - **ra:** return address



Using frame pointers in eBPF

What’s eBPF?


Process
bytecode
bpf()
eBPF
Program
Syscall
Linux
Kernel
eBPF Verifier
approved
eBPF
Program
eBPF JIT Compiler
x86_64
eBPF
Program
Process
sendmsg()
recvmsg()
No syscalls
eBPF
Sockets
TCP/IP
Network Device

Get a stack in BPF

- **`bpf_get_stack`**
  - BPF helper to unwind the stack using frame pointers

```
bpf_user_pt_regs_t *regs = &ctx->regs;
u64 ip = PT_REGS_IP(regs); // read leaf instruction pointer
u64 bp = PT_REGS_FP(regs); // read leaf base pointer
u64 ra = 0; // return address

// *save leaf frame*

for (int i = 0; i < MAX_STACK_DEPTH; i++) {
  // return address is the next register from rbp, so 8 bytes away
  err = bpf_probe_read_user(&ra, 8, (void *)bp + 8);
  if (err < 0) {
    // error
  }

  // Rewinding the program counter to get the instruction pointer for the
  // previous function would be ideal but is unreliable in `x86` due to
  // variable width encoding. We can ensure correctness only by disassembling
  // the `.text` section which would be unfeasible. Since return addresses
  // always point to the next instruction to be executed after returning from
  // the function (and stack grows downwards), subtracting 1 from the current
  // `ra` gives us the current instruction pointer location, if not the exact
  // instruction boundary
  ip = ra - 1;

  // *save frame*

  // read content of base pointer into bp variable
  err = bpf_probe_read_user(&bp, 8, (void *)bp);
  if (err < 0) {
    // error
  }

  // if bp == 0 we’ve reached the bottom of the stack
}
```

Having frame pointers in BPF makes regular profiling **easy**

Why do frame pointers matter for observability?

- **Simplified Profiling**:
  - Don't worrying about compiler configurations
- **Lower Overhead**:
  - Cheaper than using DWARF or DWARF-derived information to unwind
- **Debugging Accessibility**:
  - bcc-tools, bpftrace, perf and other such tooling to work out of the box

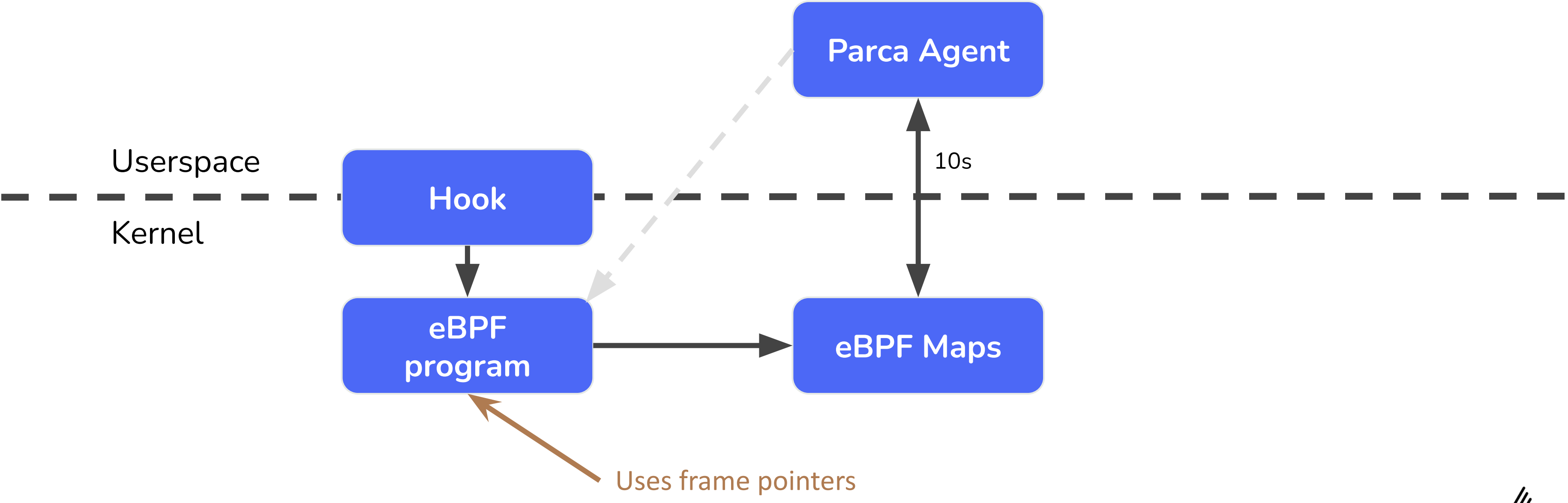
Check out last year's FOSDEM talk! "Stack walking/unwinding without frame pointers"

Possibilities are endless:

Unwinding takes 2 memory reads

- bpftrace
  - ustack builtin, the bpf_get_stack in bpftrace’s language
  - `bpftrace -e 'profile:hz:99 { @[ustack] = count(); }' // one-liner profiler`
- Go execution tracer
- Profile Guided Optimizations (PGO) research
  - Context-sensitive sampling-based PGO (CSSPGO)

Communicating with Userspace



Bringing frame pointers to the masses.

Ubuntu 24.04 LTS will have frame pointers enabled by default on 64-bit platforms.

Performance implications and future plans for optimisation.

Frame pointers are just the start.

Canonical is building a company-wide Performance Engineering machine.

Get it from the
Snap Store

```
snap install parca
```

```
snap install parca-agent --classic
```

Get it from the
Charmhub

```
juju deploy parca[-k8s]
```

Get in touch!

Matthias Loibl

@metalmatze@social.metalmatze.de

@metalmatze

Jon Seager

@jnsgruk@hachyderm.io

@jnsgruk