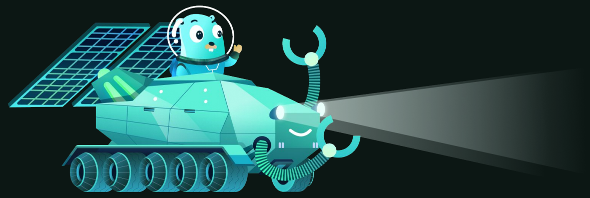


eBPF Summit
October 28, 2020

# Debugging Go in production using eBPF

# ABOUT ME

👋 i’m Zain

@zainasgar

Co-Founder/CEO Pixie (@pixie_run) &
Adjunct Professor of CS @ Stanford

Stanford University

Google AI

NVIDIA

PIXIE

# DEVELOPER PROBLEM

You’re an application developer, and your program is misbehaving.

- No problem. You have logs! Right?
- Uh-oh, not in the spot you need. 😞

We’ve all been there:

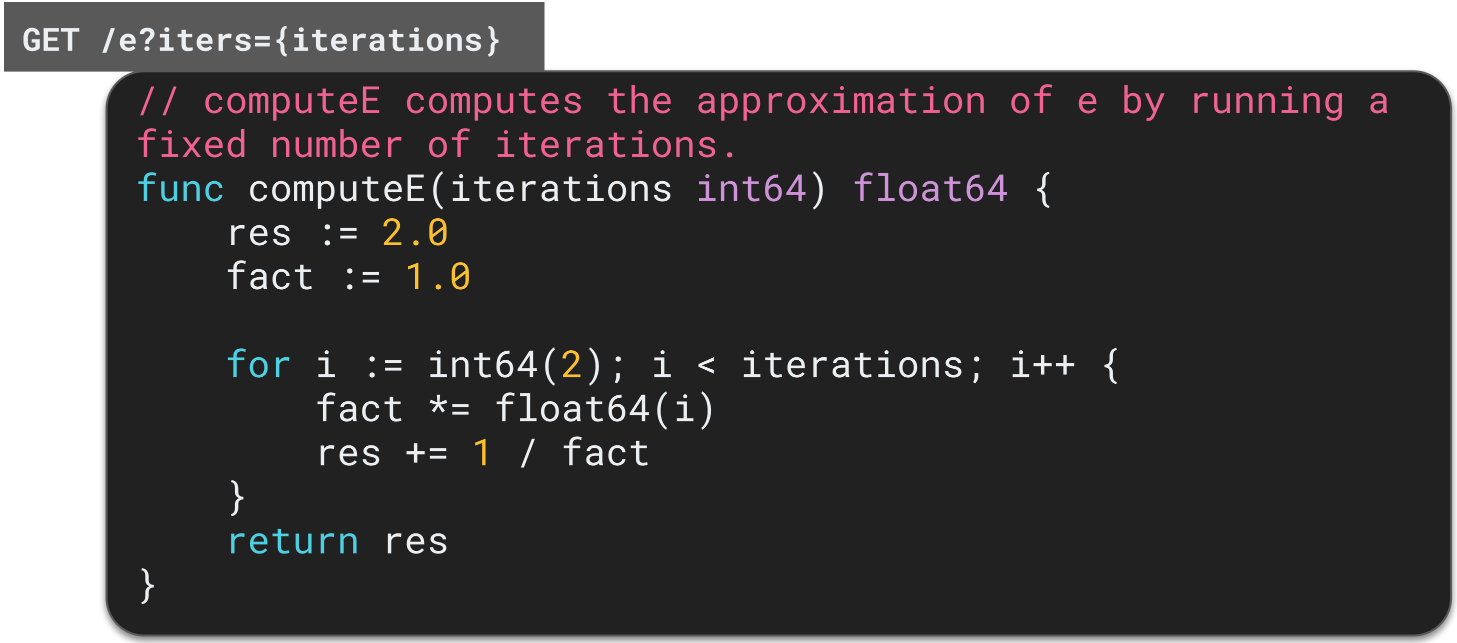
- *“I just wish I could see the variable* `x` *when* `Foo()` *is called”*

Use Case

# Let’s look at test application

Github Link: https://github.com/pixie-labs/pixie/tree/main/demos/simple-gotracing

Relevant Code:



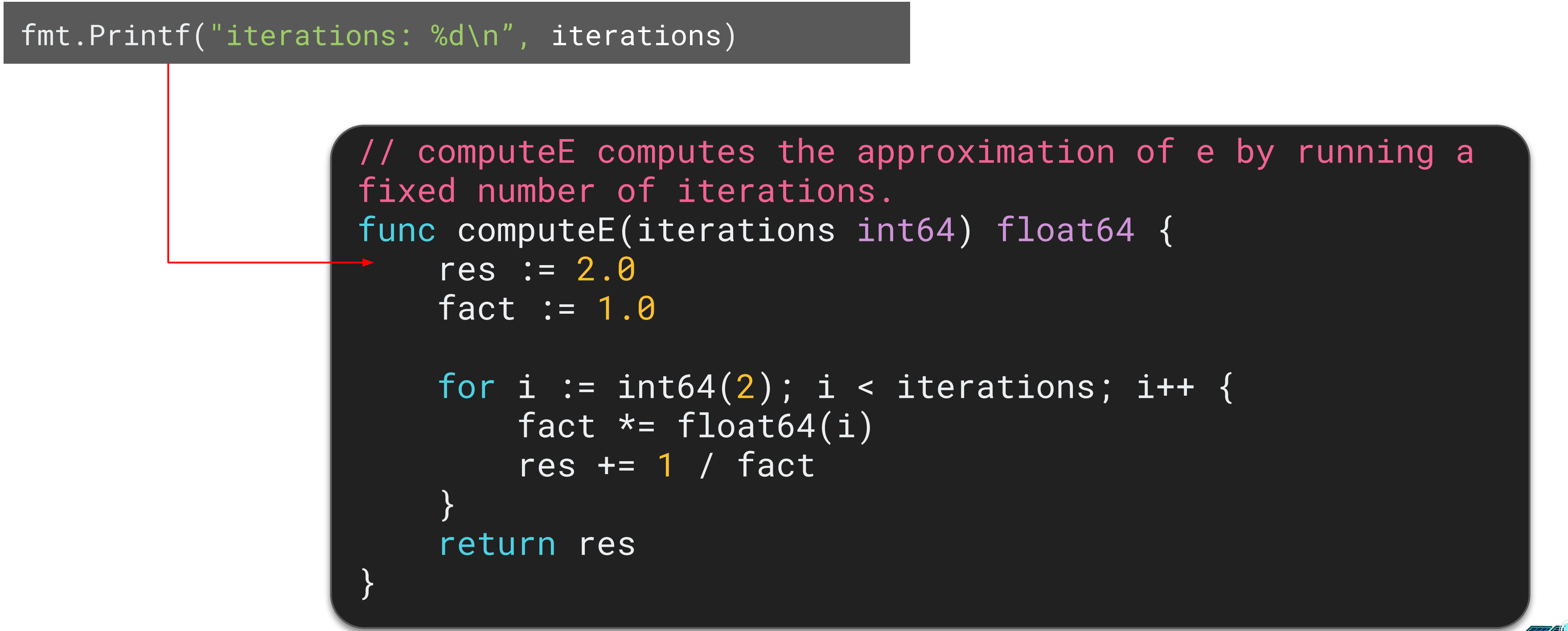
```
GET /e?iters={iterations}
```

```go
// computeE computes the approximation of e by running a
fixed number of iterations.
func computeE(iterations int64) float64 {
    res := 2.0
    fact := 1.0

    for i := int64(2); i < iterations; i++ {
        fact *= float64(i)
        res += 1 / fact
    }
    return res
}
```

Use Case

# What if we just want to log the iterations?



```
fmt.Printf("iterations: %d\n”, iterations)
```

```
// computeE computes the approximation of e by running a
fixed number of iterations.
func computeE(iterations int64) float64 {
    res := 2.0
    fact := 1.0

    for i := int64(2); i < iterations; i++ {
        fact *= float64(i)
        res += 1 / fact
    }
    return res
}
```

## YOUR OPTIONS

**Option 1: Add a log to your program, re-compile and re-deploy.**

- This can be simple log statements, or
- More comprehensive like Open tracing.

**Option 2: Debugger**

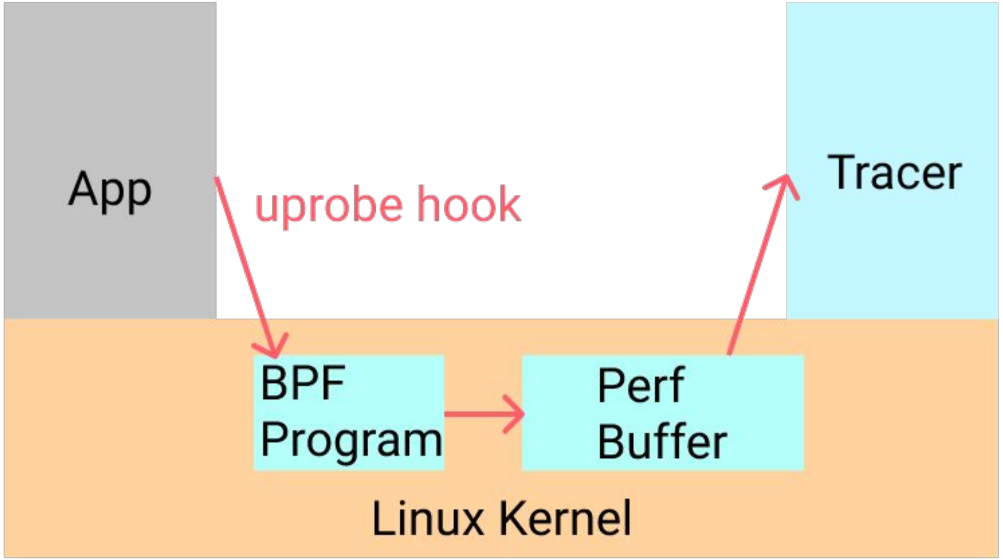
- GDB
- Delve

**Option 3: Linux tracing utility**

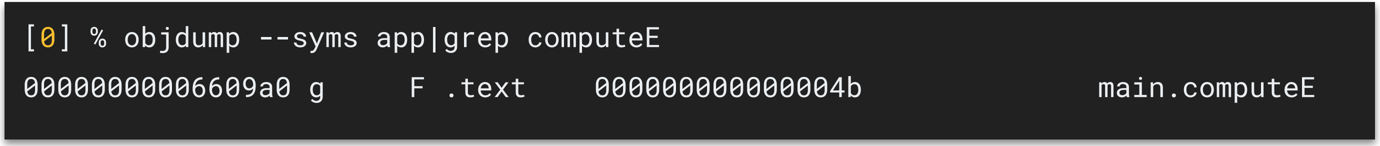
- strace/ftrace
- LTTng/USDT

**Option 4: eBPF** 🤔

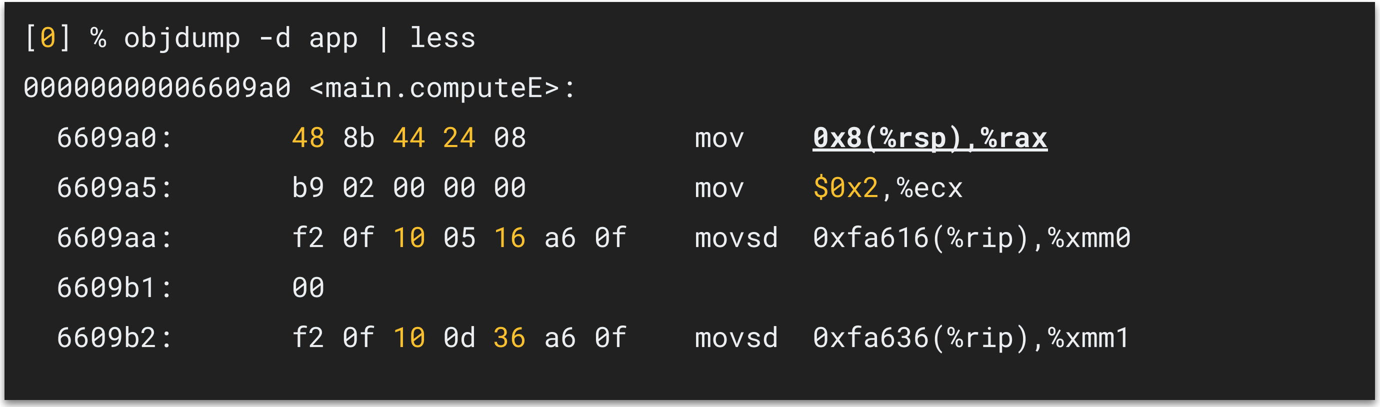
# What are we going to build?



# Diving into the details

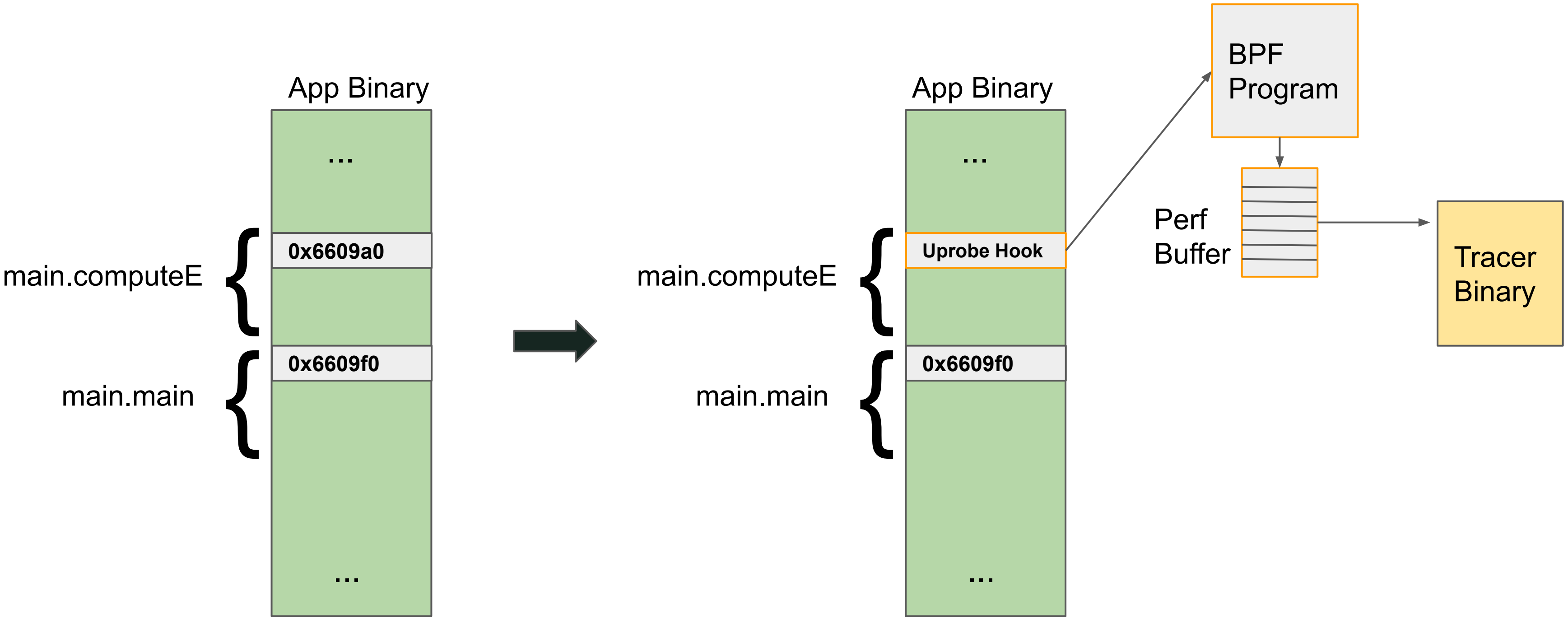


```
[0] % objdump --syms app|grep computeE
00000000006609a0 g     F .text    000000000000004b              main.computeE
```



```
[0] % objdump -d app | less
00000000006609a0 <main.computeE>:
  6609a0:       48 8b 44 24 08          mov    0x8(%rsp),%rax
  6609a5:       b9 02 00 00 00          mov    $0x2,%ecx
  6609aa:       f2 0f 10 05 16 a6 0f    movsd  0xfa616(%rip),%xmm0
  6609b1:       00
  6609b2:       f2 0f 10 0d 36 a6 0f    movsd  0xfa636(%rip),%xmm1
```

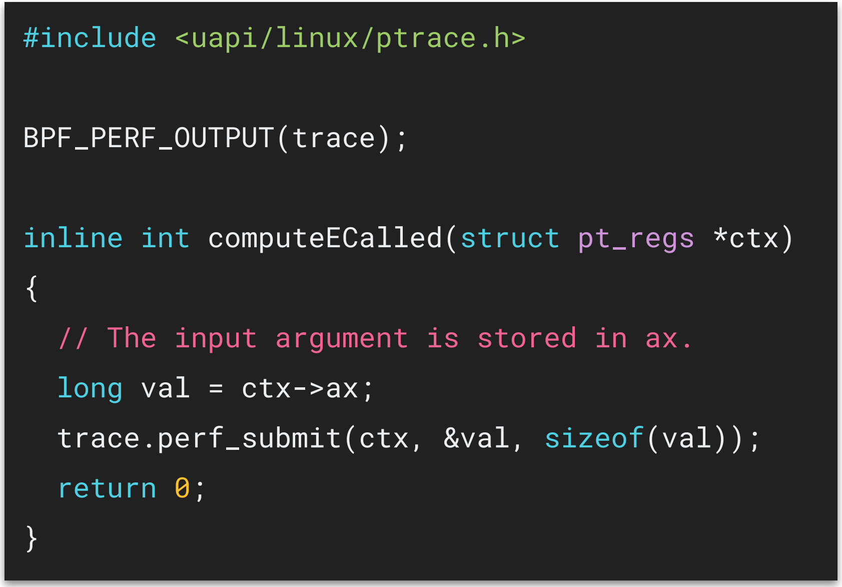
# Using uprobes



eBPF

# What does the BPF program look like?

- The function is simply invoked whenever main.computeE is called.
- The registration is done via UProbes
- It attaches to every running version of the binary



```
#include <uapi/linux/ptrace.h>

BPF_PERF_OUTPUT(trace);

inline int computeECalled(struct pt_regs *ctx)
{
  // The input argument is stored in ax.
  long val = ctx->ax;
  trace.perf_submit(ctx, &val, sizeof(val));
  return 0;
}
```

Github Link: https://github.com/pixie-labs/pixie/tree/main/demos/simple-gotracing/trace_example

# DEMO: Go Argument Tracer

# What’s next?

- Utilizing tracepoints for dynamic logging allows for easy instrumentation of production binaries
- The complexities of the Go ABI make it difficult to do. Especially when you consider: interfaces, channels, etc.
- Still possible to do complex things, like capture HTTP messages.

# DEMO: HTTP Tracer

# Checkout out repo/blog for open source examples

https://github.com/pixie-labs/pixie

https://blog.pixielabs.ai/ebpf

# Some related projects

https://github.com/kinvolk/inspektor-gadget

https://github.com/draios/sysdig/wiki/eBPF