eBPF Summit 2021

Why is my eBPF code slow?

Simar Singh

Twitter: @simarpreet7

GitHub: @simar7

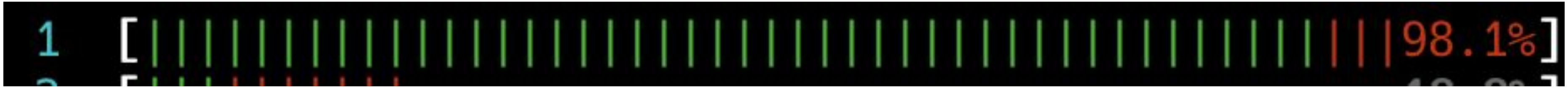
$ whoami

1. I work on Open Source stuff at Aqua Security 🔒

2. I like graphs & numbers 📊

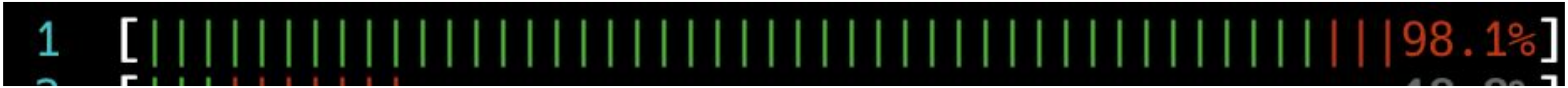
3. I also like to grow plants 🌱

We’ve all been here



```
1  [|||||||||||||||||||||||||||||||||||||||||||||||98.1%]
```

We’ve all been here



```
1  [|||||||||||||||||||||||||||||||||||||||||||||||98.1%]
```

- *“This code is slow”*

We’ve all been here

```
1  [|||||||||||||||||||||||||||||||||||||||||||||||98.1%]
```

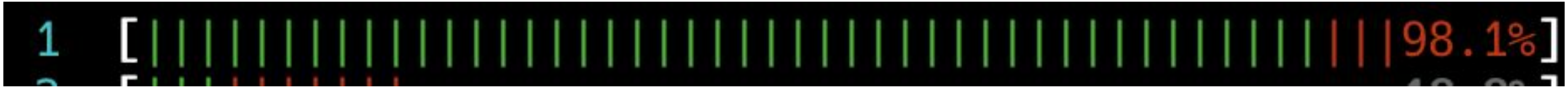
- *“This code is slow”*
- *“Need to improve performance”*

We’ve all been here

```
1  [|||||||||||||||||||||||||||||||||||||||||||||||98.1%]
```

- *“This code is slow”*
- *“Need to improve performance”*
- *“Can we optimize this?”*

pop quiz: What is taking up CPU%?

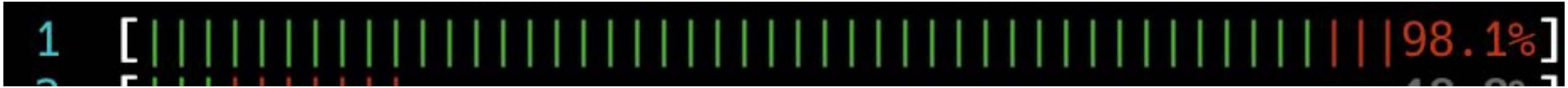


```
1  [|||||||||||||||||||||||||||||||||||||||||||||||98.1%]
```

Pick your best guess:

- Lot of computation?
- Waiting on something?

pop quiz: What is taking up CPU%?



```
1  [|||||||||||||||||||||||||||||||||||||||||||||||98.1%]
```

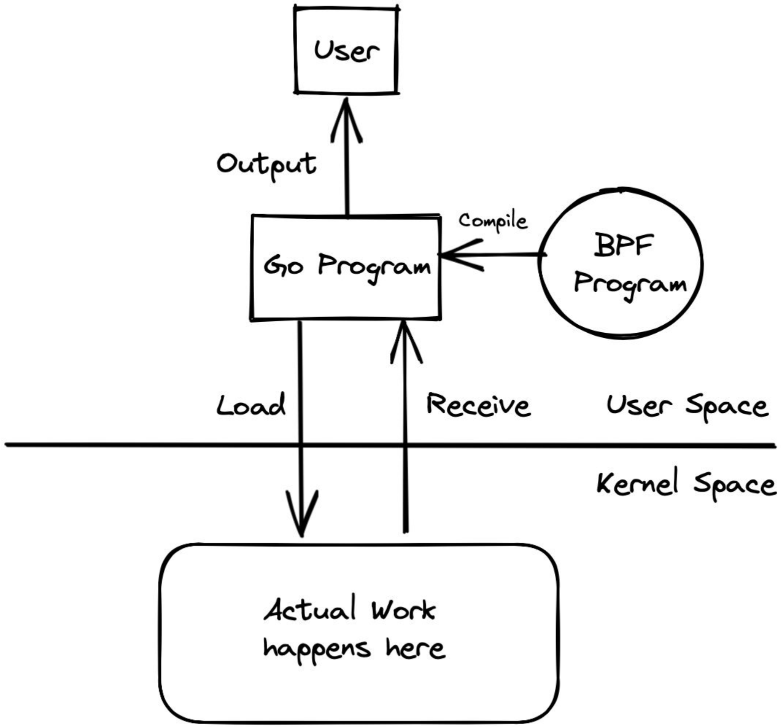
Pick your best guess:

- Lot of computation?
- Waiting on something?
- No idea.

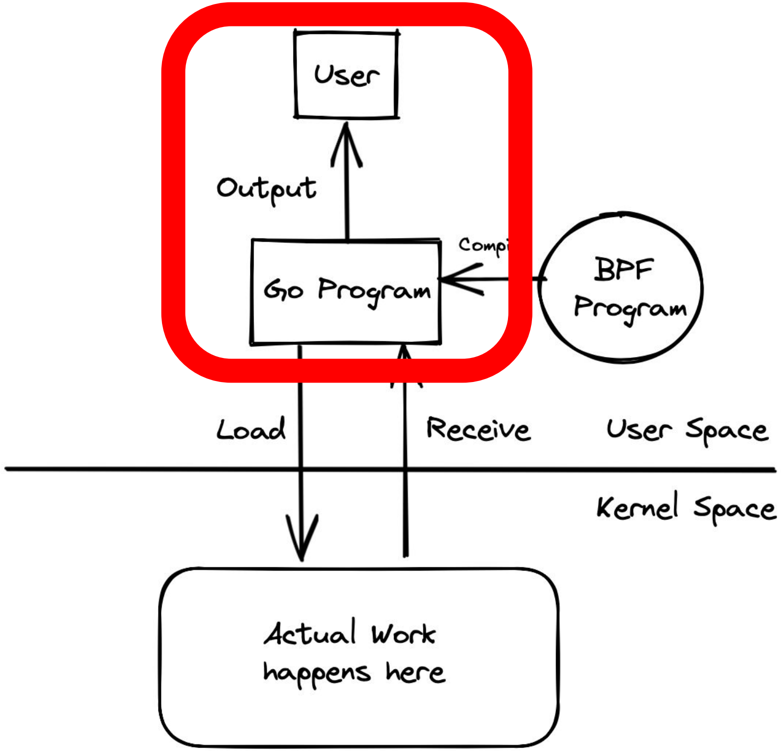
Usual benchmarking steps

- **Set a baseline**
  - What are the numbers today that matter?
- ***Kaizen* 改善: Continuous improvement**
  - Keep making small but incremental improvements
- **Know your limits**
  - No software is perfect, it’s always a tradeoff

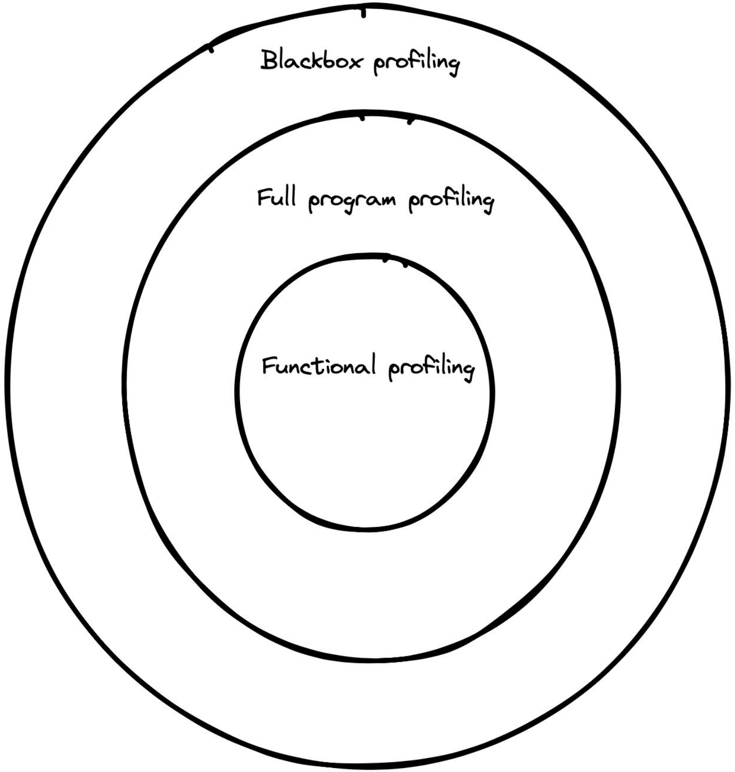
The stack


User
Output
Go Program
Compile
BPF Program
Load
Receive
User Space
Kernel Space
Actual Work happens here

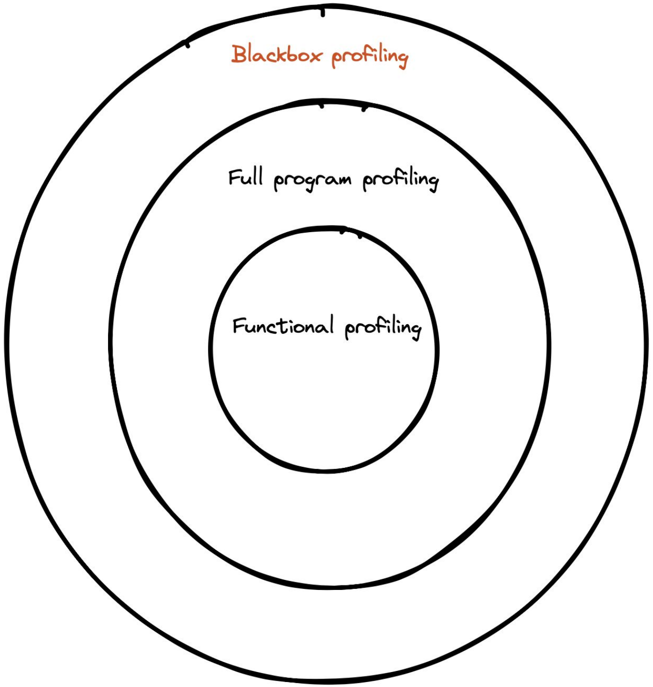
The stack


User
Output
Go Program
Compi
BPF Program
Load
Receive
User Space
Kernel Space
Actual Work happens here

How to set a baseline

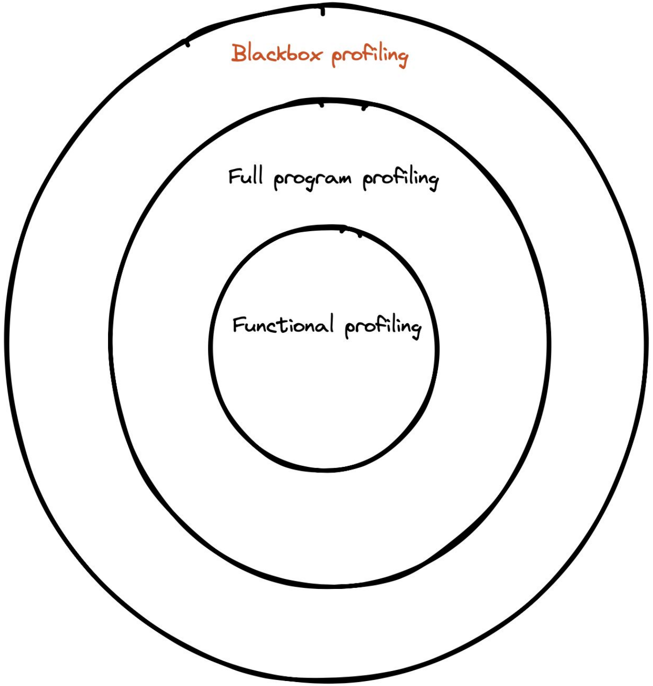


How to set a baseline



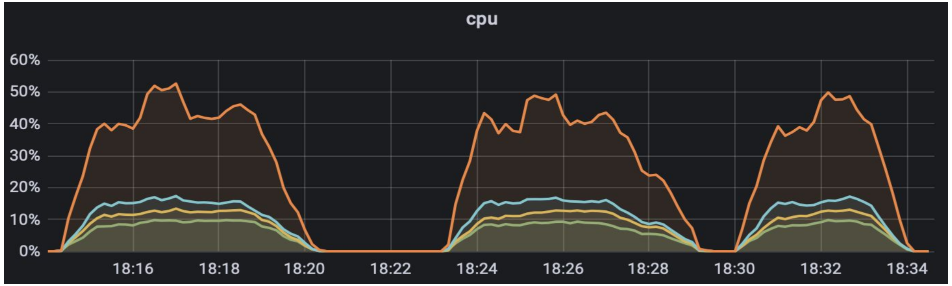
Measuring from the outside looking in

How to set a baseline

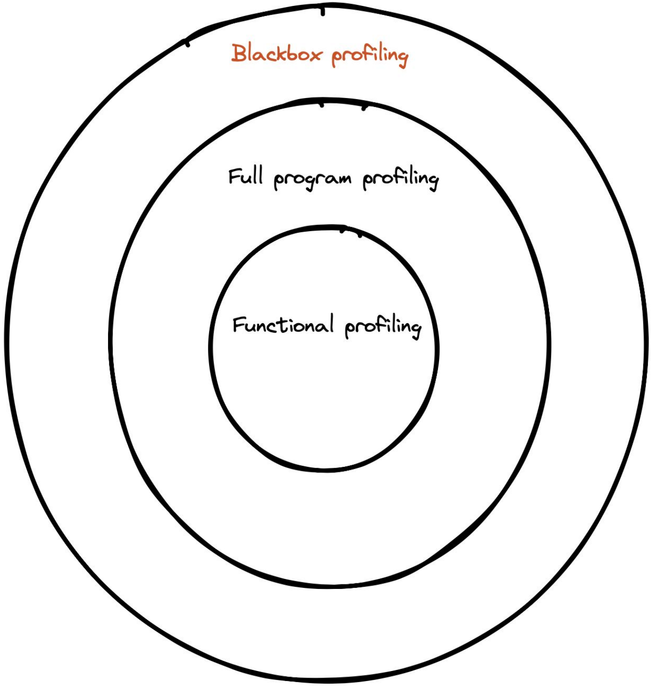


Measuring from the outside looking in

https://github.com/simar7/simple-linux-monitoring-stack

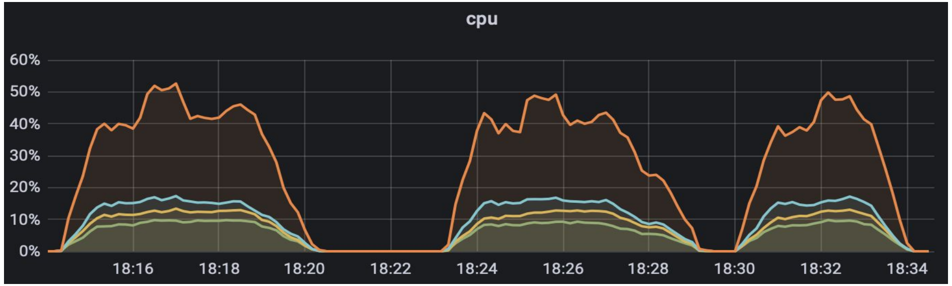


How to set a baseline



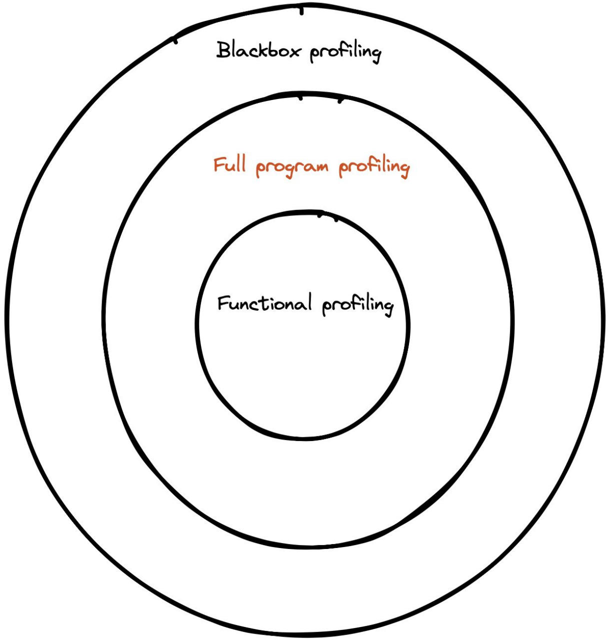
Measuring from the outside looking in

https://github.com/simar7/simple-linux-monitoring-stack



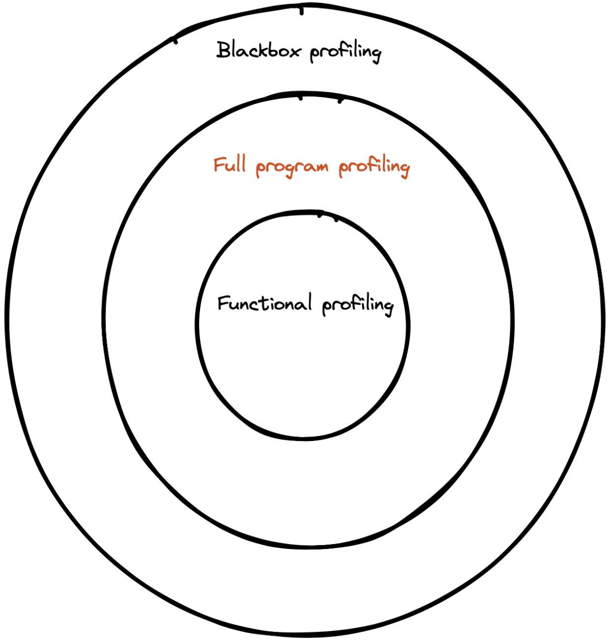
Pros	Cons
10,000ft overview	Hard to pinpoint
Easy to setup	Many components at play

How to set a baseline



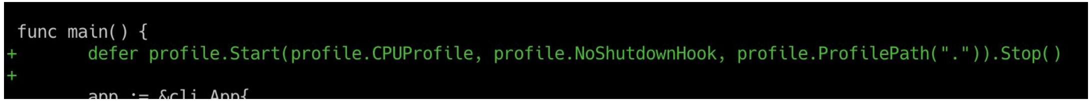
Measuring from within the system

How to set a baseline

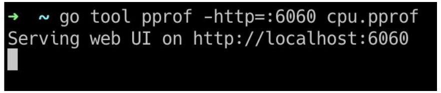


Measuring from within the system

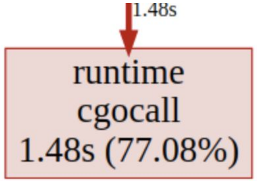
https://github.com/pkg/profile



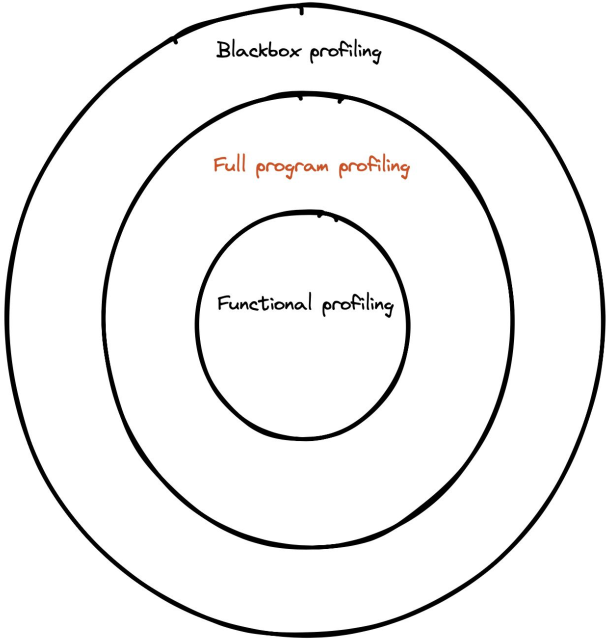
```
func main() {
+       defer profile.Start(profile.CPUProfile, profile.NoShutdownHook, profile.ProfilePath(".")).Stop()
+
        app := &cli.App{
```



```
➜  ~ go tool pprof -http=:6060 cpu.pprof
Serving web UI on http://localhost:6060
```

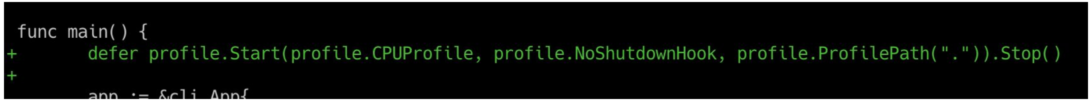


How to set a baseline

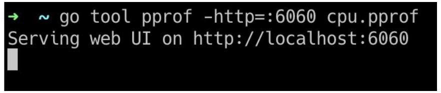


Measuring from within the system

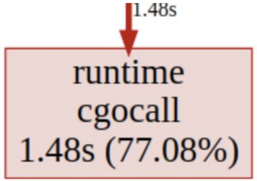
https://github.com/pkg/profile



```
 func main() {
+        defer profile.Start(profile.CPUProfile, profile.NoShutdownHook, profile.ProfilePath(".")).Stop()
+
         app := &cli.App{
```

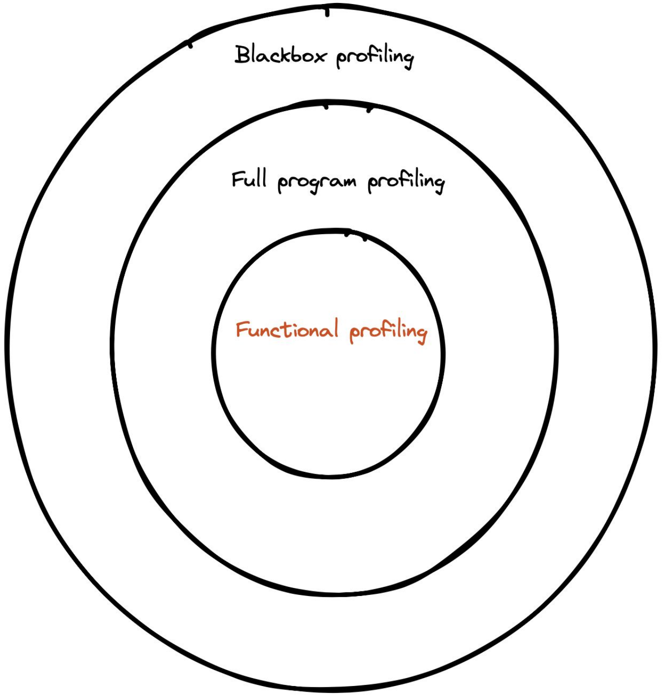


```
➜  ~ go tool pprof -http=:6060 cpu.pprof
Serving web UI on http://localhost:6060
```



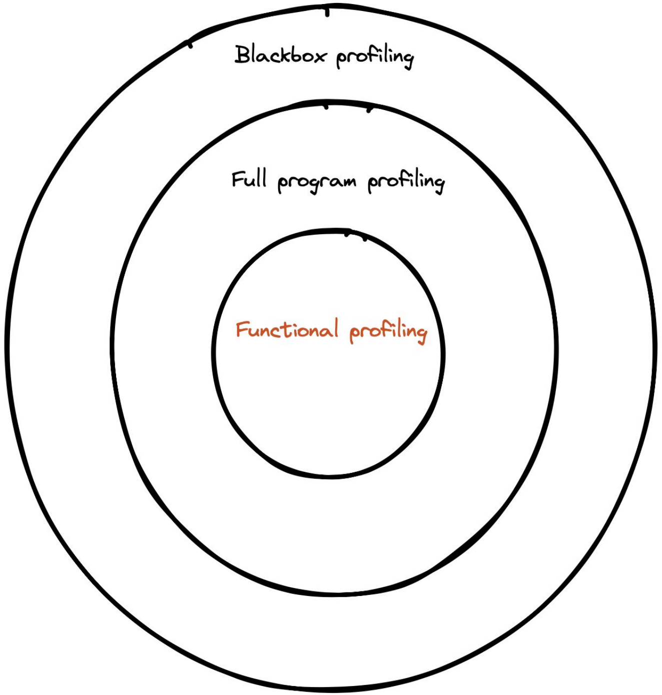
Pros	Cons
Easy to setup	No visibility outside of userspace
Code paths highlighted	Can have low Signal to Noise ratio

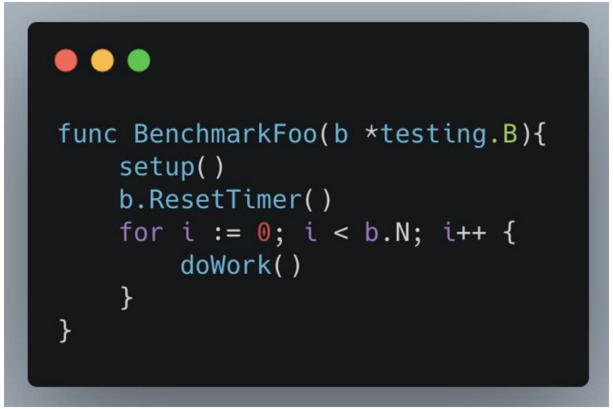
How to set a baseline



Measuring each unit on it’s own

How to set a baseline

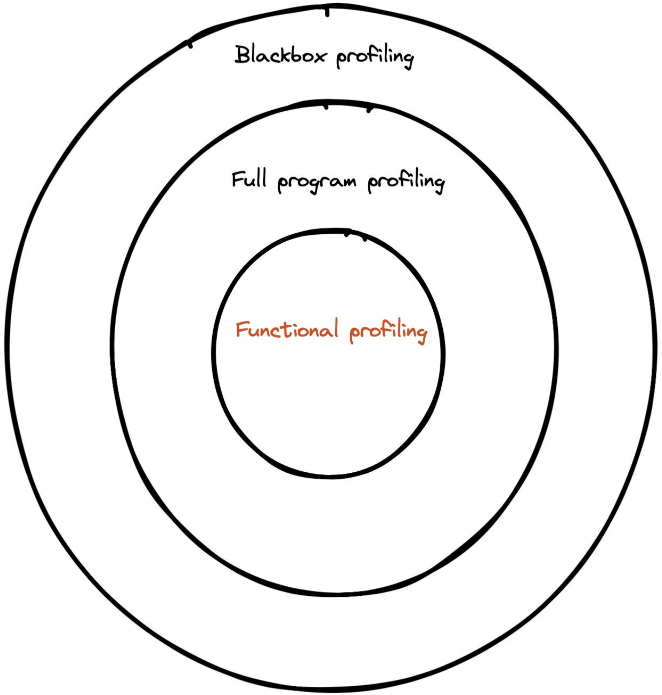




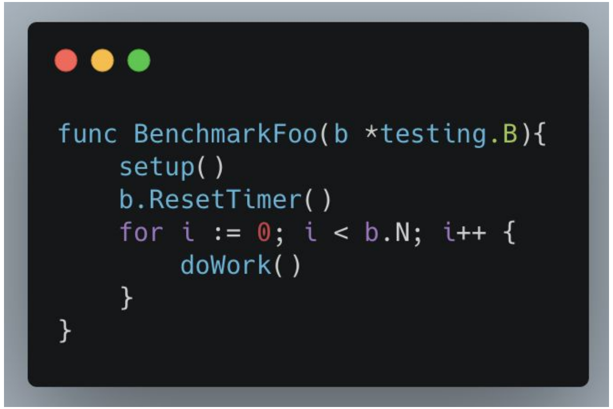
```
func BenchmarkFoo(b *testing.B){
    setup()
    b.ResetTimer()
    for i := 0; i < b.N; i++ {
        doWork()
    }
}
```

Measuring each unit on it’s own

How to set a baseline



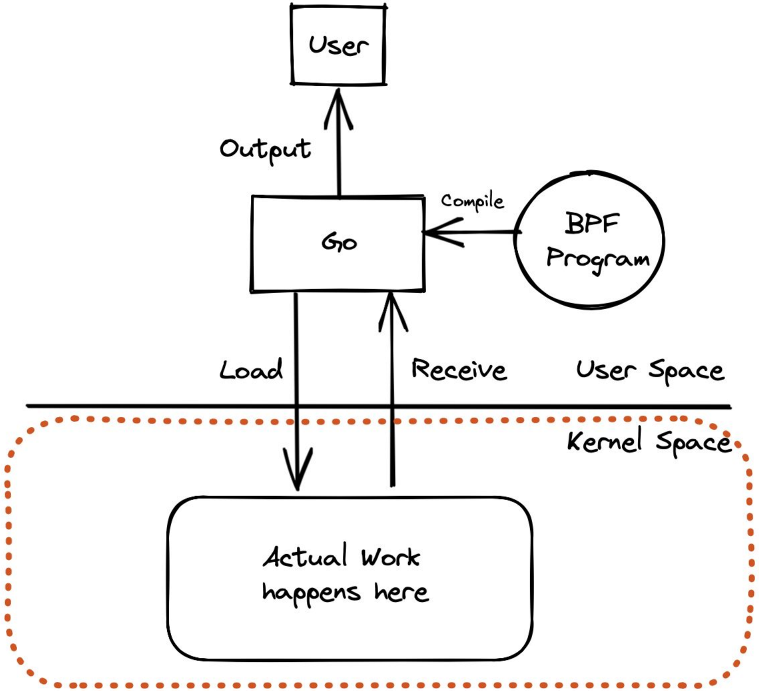
Measuring each unit on it’s own



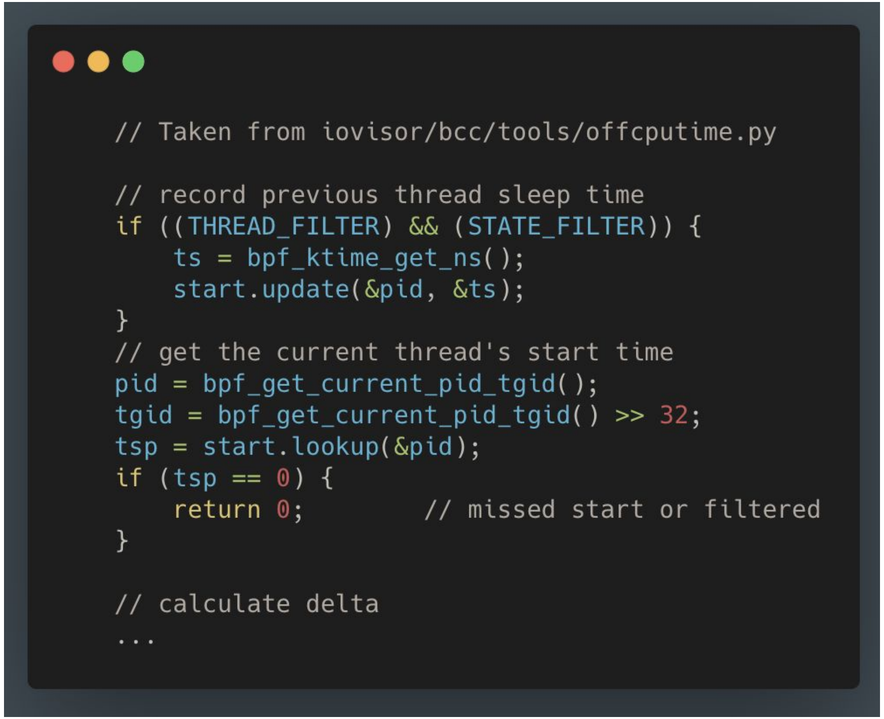
```
func BenchmarkFoo(b *testing.B){
    setup()
    b.ResetTimer()
    for i := 0; i < b.N; i++ {
        doWork()
    }
}
```

Pros	Cons
Small unit of work	Can lack external attributing factors
Well defined	Improvements might not make a big impact

What about the kernel space code?



⚡ eBPF to the rescue!



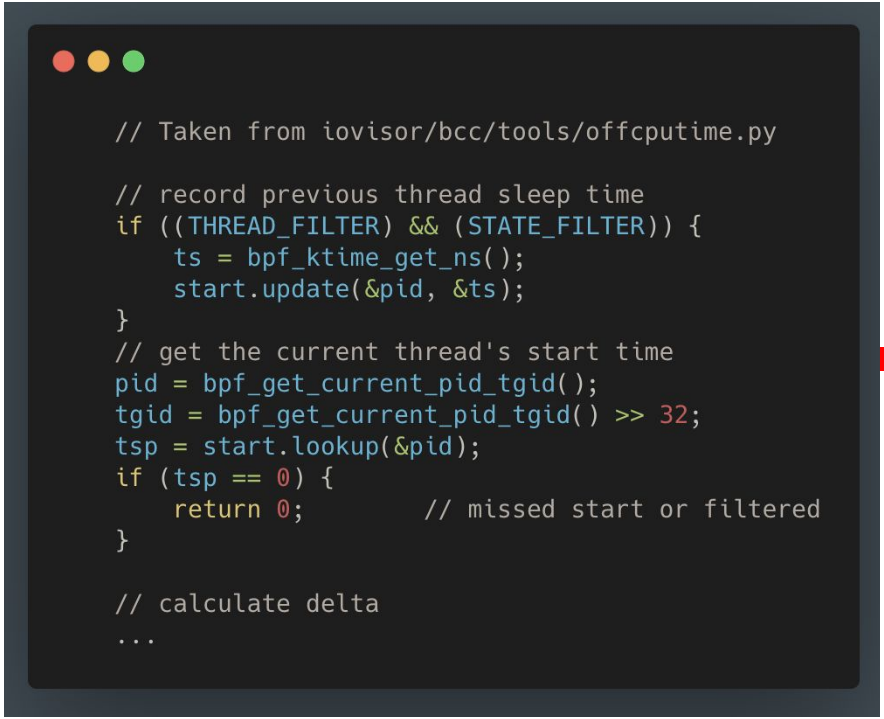
```
// Taken from iovisor/bcc/tools/offcputime.py

// record previous thread sleep time
if ((THREAD_FILTER) && (STATE_FILTER)) {
    ts = bpf_ktime_get_ns();
    start.update(&pid, &ts);
}
// get the current thread's start time
pid = bpf_get_current_pid_tgid();
tgid = bpf_get_current_pid_tgid() >> 32;
tsp = start.lookup(&pid);
if (tsp == 0) {
    return 0;        // missed start or filtered
}

// calculate delta
...
```

https://github.com/iovisor/bcc/blob/master/tools/offcputime.py

⚡ eBPF to the rescue!

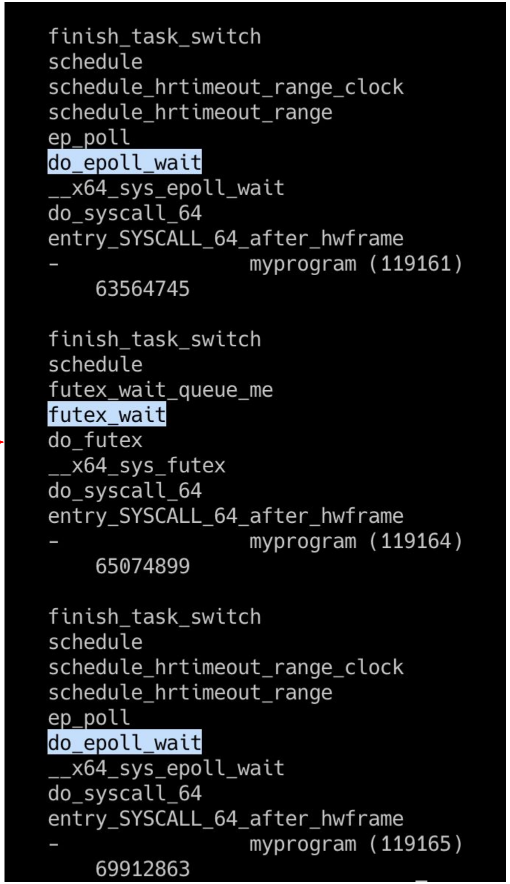


```
// Taken from iovisor/bcc/tools/offcputime.py

// record previous thread sleep time
if ((THREAD_FILTER) && (STATE_FILTER)) {
    ts = bpf_ktime_get_ns();
    start.update(&pid, &ts);
}
// get the current thread's start time
pid = bpf_get_current_pid_tgid();
tgid = bpf_get_current_pid_tgid() >> 32;
tsp = start.lookup(&pid);
if (tsp == 0) {
    return 0;        // missed start or filtered
}

// calculate delta
...
```

https://github.com/iovisor/bcc/blob/master/tools/offcputime.py

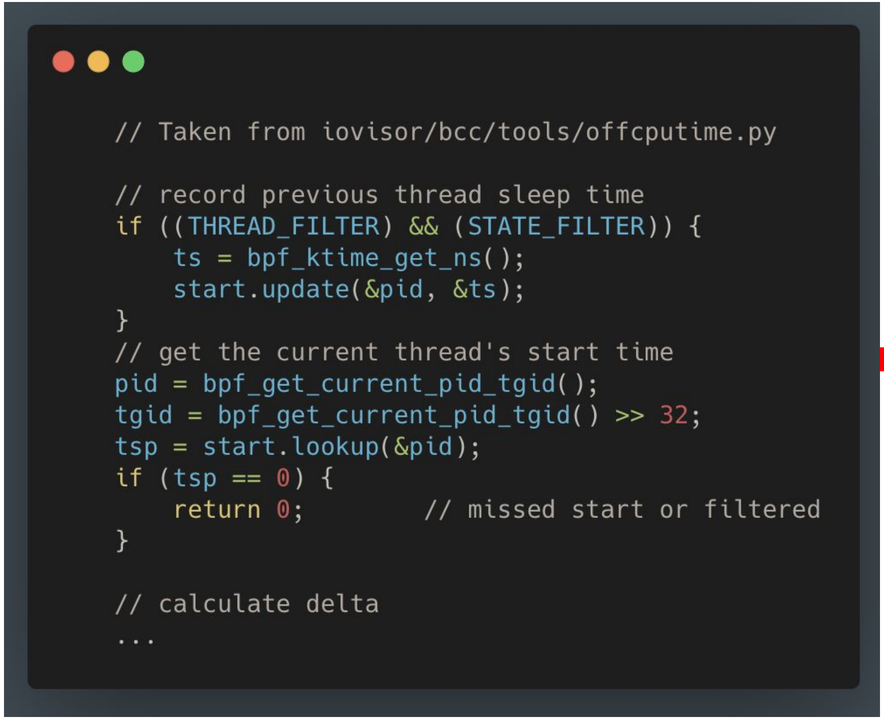


```
finish_task_switch
schedule
schedule_hrtimeout_range_clock
schedule_hrtimeout_range
ep_poll
do_epoll_wait
__x64_sys_epoll_wait
do_syscall_64
entry_SYSCALL_64_after_hwframe
-                myprogram (119161)
    63564745

finish_task_switch
schedule
futex_wait_queue_me
futex_wait
do_futex
__x64_sys_futex
do_syscall_64
entry_SYSCALL_64_after_hwframe
-                myprogram (119164)
    65074899

finish_task_switch
schedule
schedule_hrtimeout_range_clock
schedule_hrtimeout_range
ep_poll
do_epoll_wait
__x64_sys_epoll_wait
do_syscall_64
entry_SYSCALL_64_after_hwframe
-                myprogram (119165)
    69912863
```

⚡ eBPF to the rescue!

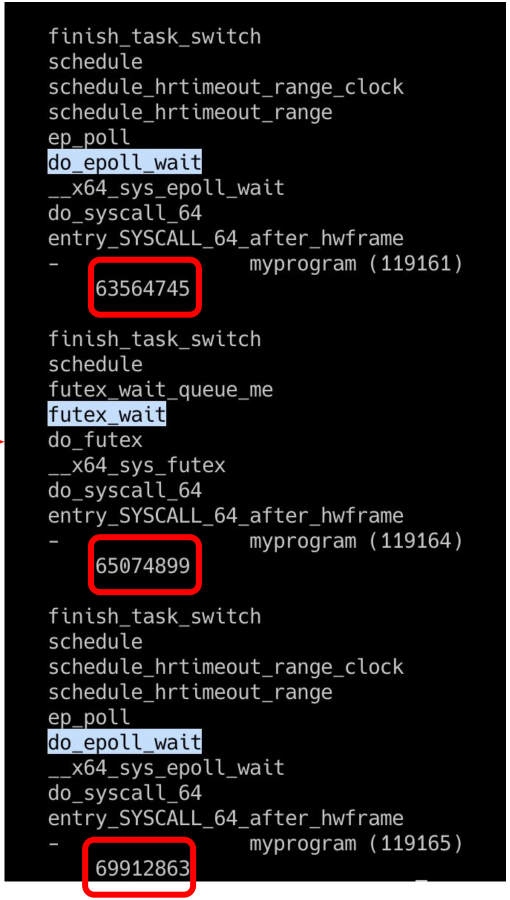


```
// Taken from iovisor/bcc/tools/offcputime.py

// record previous thread sleep time
if ((THREAD_FILTER) && (STATE_FILTER)) {
    ts = bpf_ktime_get_ns();
    start.update(&pid, &ts);
}
// get the current thread's start time
pid = bpf_get_current_pid_tgid();
tgid = bpf_get_current_pid_tgid() >> 32;
tsp = start.lookup(&pid);
if (tsp == 0) {
    return 0;           // missed start or filtered
}

// calculate delta
...
```

https://github.com/iovisor/bcc/blob/master/tools/offcputime.py

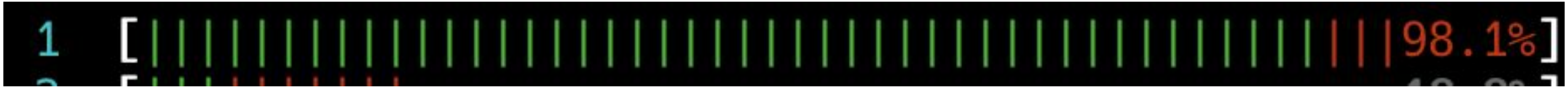


```
finish_task_switch
schedule
schedule_hrtimeout_range_clock
schedule_hrtimeout_range
ep_poll
do_epoll_wait
__x64_sys_epoll_wait
do_syscall_64
entry_SYSCALL_64_after_hwframe
-                myprogram (119161)
    63564745

finish_task_switch
schedule
futex_wait_queue_me
futex_wait
do_futex
__x64_sys_futex
do_syscall_64
entry_SYSCALL_64_after_hwframe
-                myprogram (119164)
    65074899

finish_task_switch
schedule
schedule_hrtimeout_range_clock
schedule_hrtimeout_range
ep_poll
do_epoll_wait
__x64_sys_epoll_wait
do_syscall_64
entry_SYSCALL_64_after_hwframe
-                myprogram (119165)
    69912863
```

pop quiz: revisited

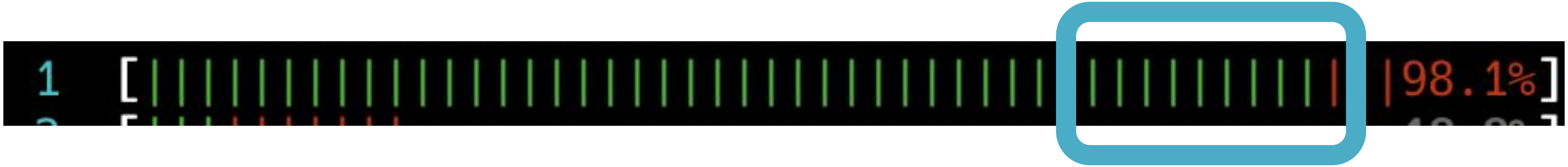


```
1  [|||||||||||||||||||||||||||||||||||||||||||||||98.1%]
```

Pick your best guess:

- Lot of computation?
- Waiting on something?

pop quiz: revisited

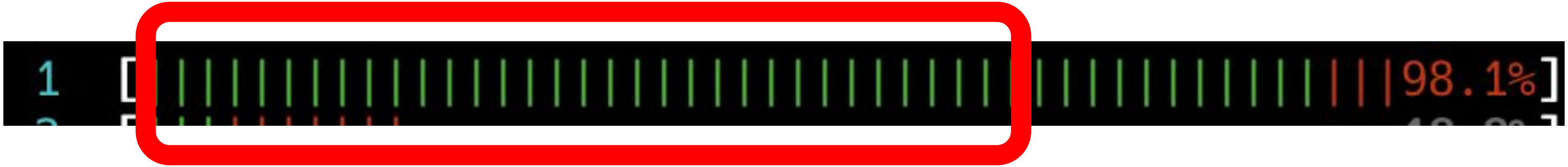


```
1  [||||||||||||||||||||||||||||||||||||||||||||||98.1%]
```

Pick your best guess:

- Lot of computation?
- Waiting on something?

pop quiz: revisited

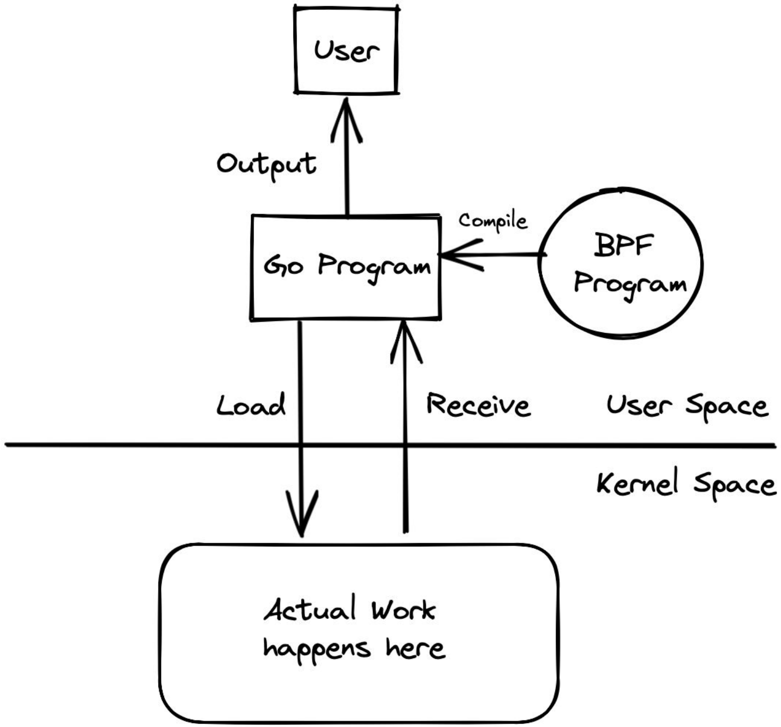


```
1  [|||||||||||||||||||||||||||||||||||||||||||||||98.1%]
```

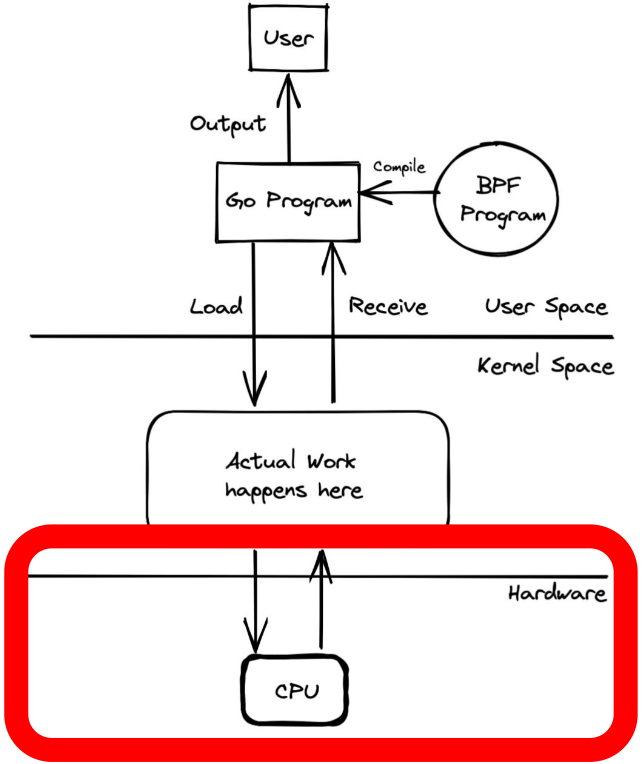
Pick your best guess:

- Lot of computation?
- Waiting on something?

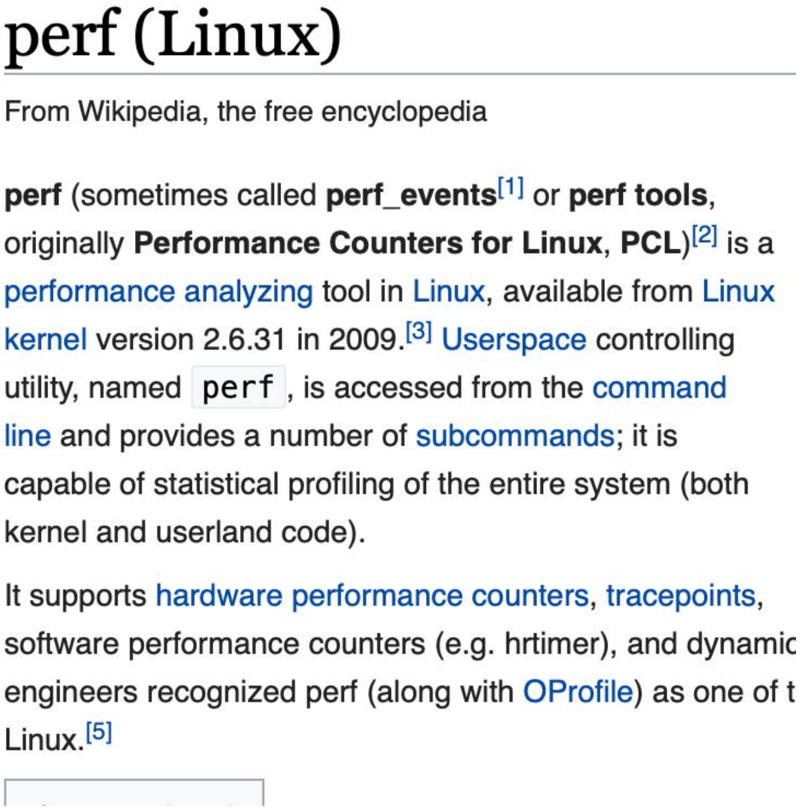
Can we go even lower?



Can we go even lower? *yes.*



Visiting an old friend



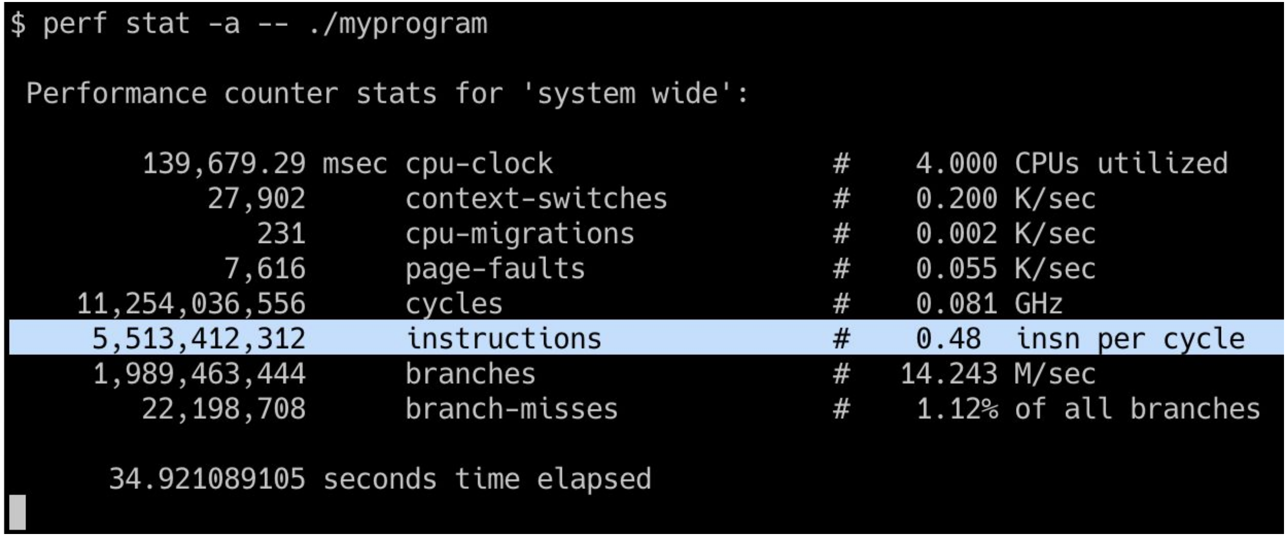
perf (Linux)

From Wikipedia, the free encyclopedia

perf (sometimes called **perf_events**[1] or **perf tools**, originally **Performance Counters for Linux**, **PCL**)[2] is a performance analyzing tool in Linux, available from Linux kernel version 2.6.31 in 2009.[3] Userspace controlling utility, named `perf`, is accessed from the command line and provides a number of subcommands; it is capable of statistical profiling of the entire system (both kernel and userland code).

It supports hardware performance counters, tracepoints, software performance counters (e.g. hrtimer), and dynamic engineers recognized perf (along with OProfile) as one of t Linux.[5]

Visiting an old friend



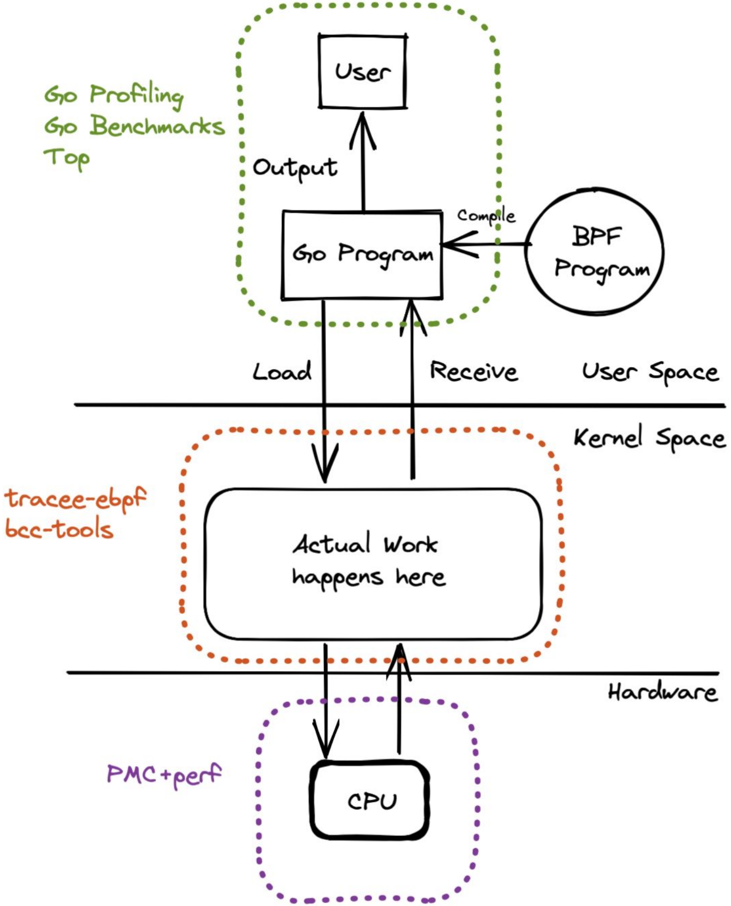
```
$ perf stat -a -- ./myprogram

 Performance counter stats for 'system wide':

        139,679.29 msec cpu-clock                 #    4.000 CPUs utilized
            27,902      context-switches          #    0.200 K/sec
               231      cpu-migrations            #    0.002 K/sec
             7,616      page-faults               #    0.055 K/sec
    11,254,036,556      cycles                    #    0.081 GHz
     5,513,412,312      instructions              #    0.48  insn per cycle
     1,989,463,444      branches                  #   14.243 M/sec
        22,198,708      branch-misses             #    1.12% of all branches

      34.921089105 seconds time elapsed
```

Wrap up



Go Profiling
Go Benchmarks
Top
User
Output
Compile
Go Program
BPF Program
Load
Receive
User Space
Kernel Space
tracee-ebpf
bcc-tools
Actual Work happens here
Hardware
PMC+perf
CPU

Thanks!

Simar Singh

Twitter: @simarpreet7

GitHub: @simar7