xprobes

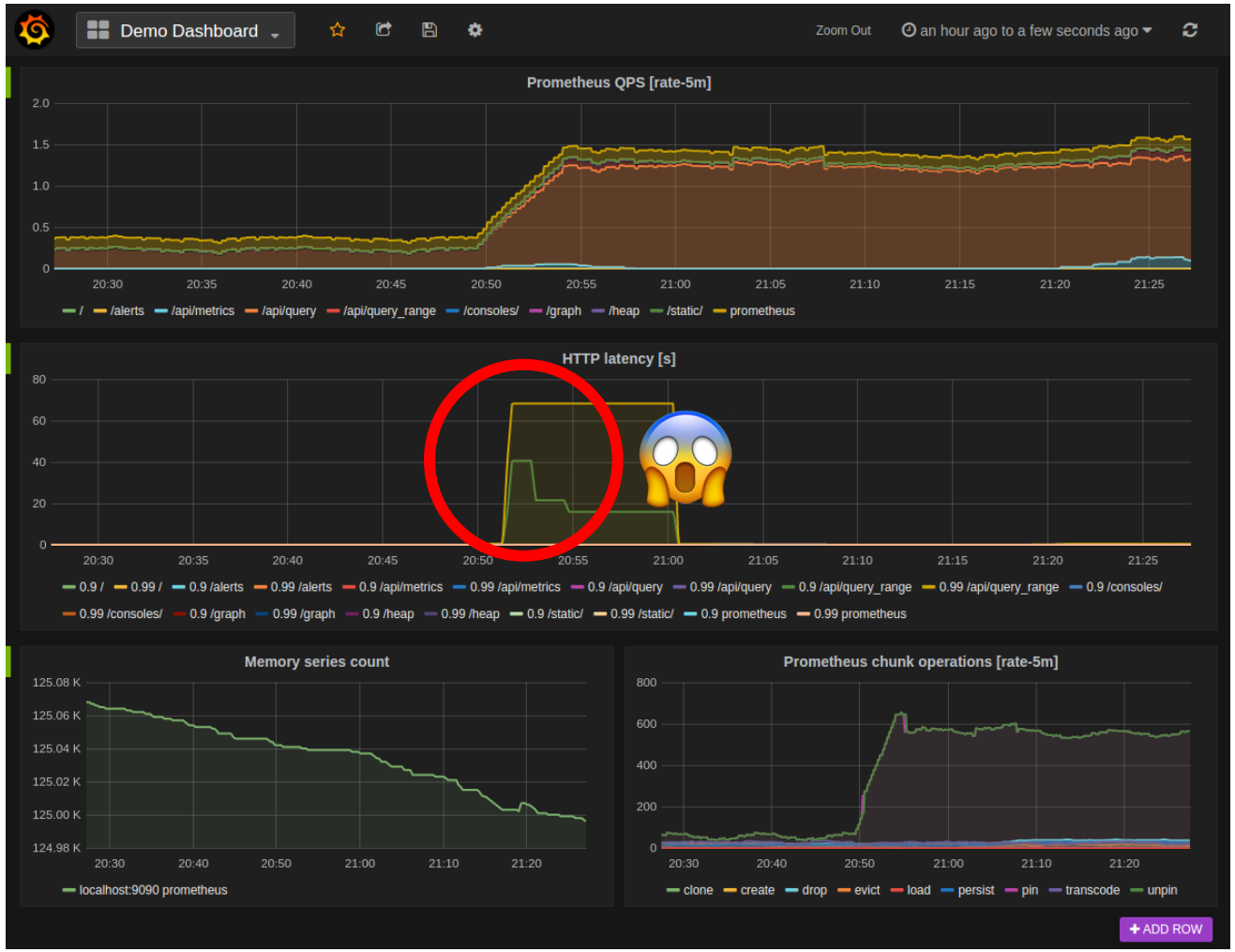
Hybrid User/Kernel eBPF Probes for Cross-Layer Observability

Lucas Castanheira

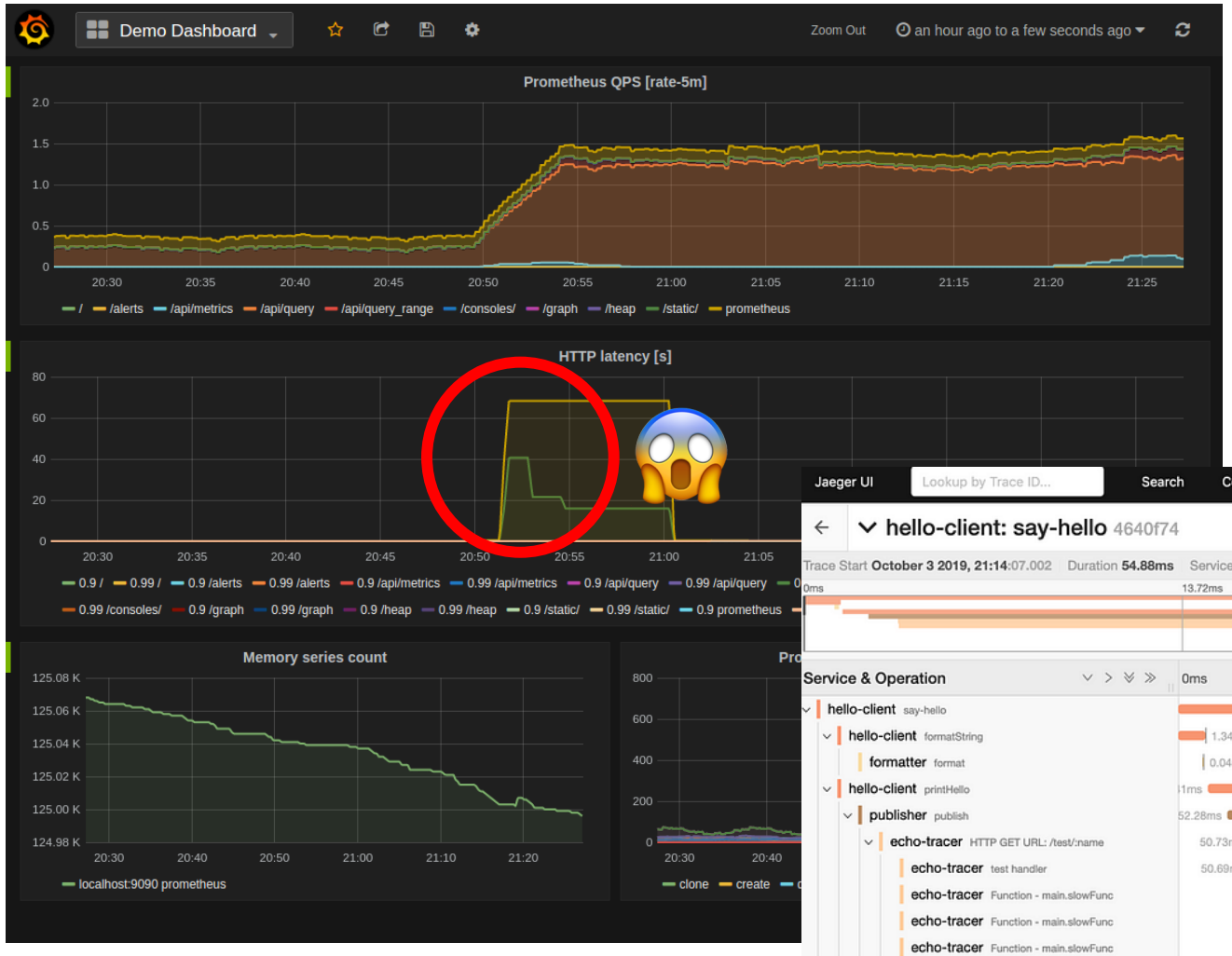
Theo Benson

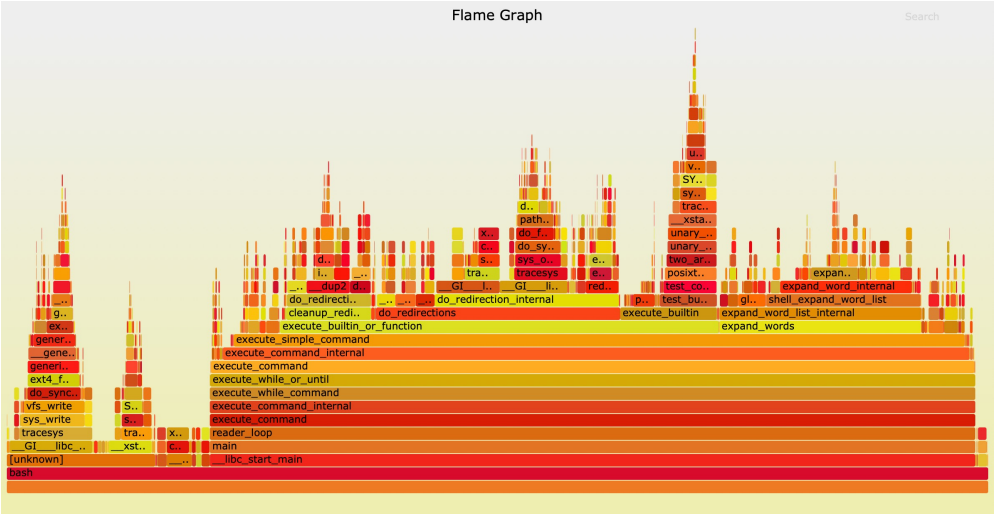
Carnegie Mellon University

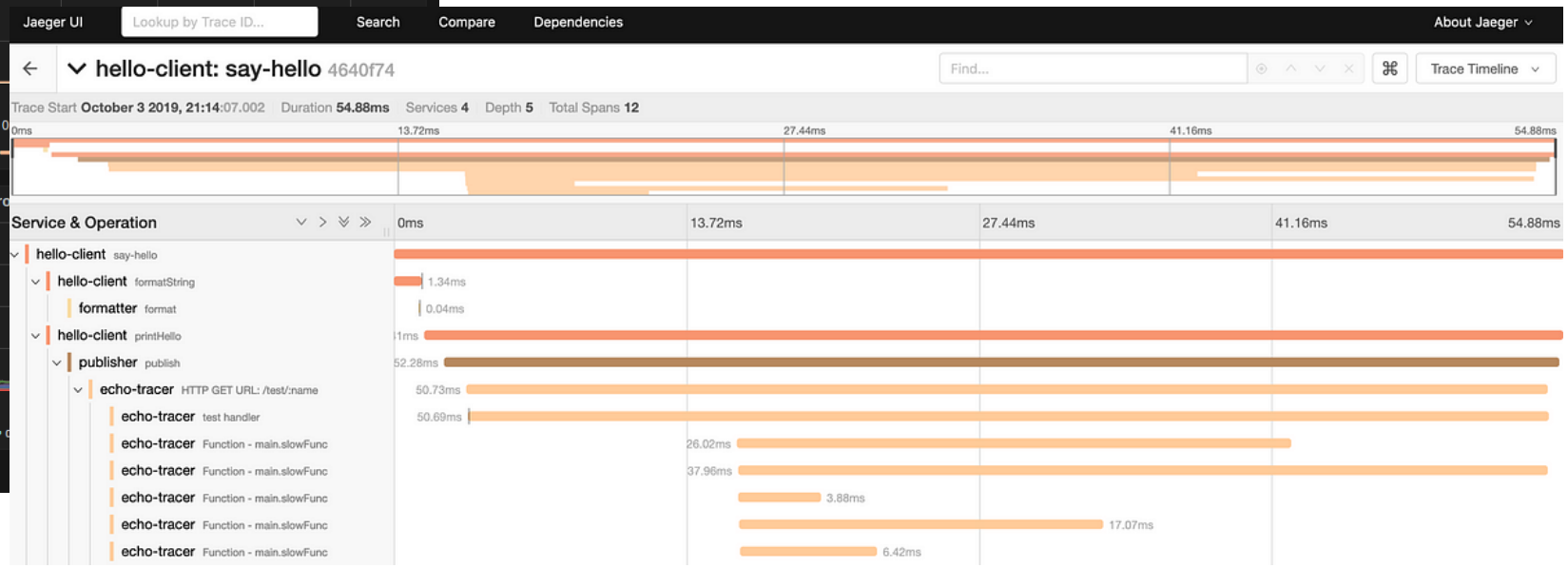
Understanding Why Your Endpoint is Slow



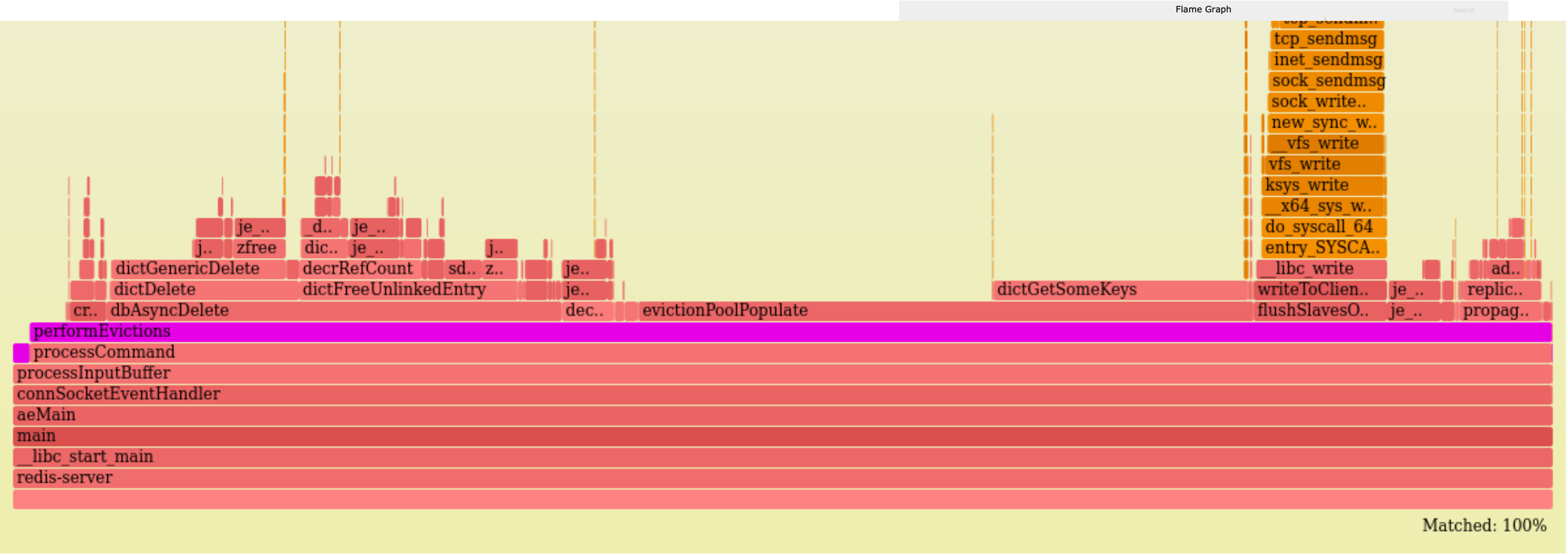
Understanding Why Your Endpoint is Slow

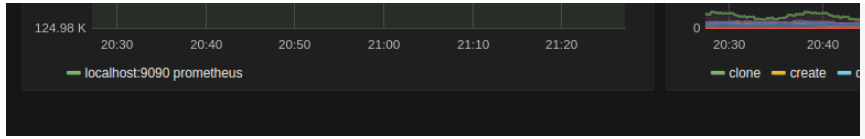


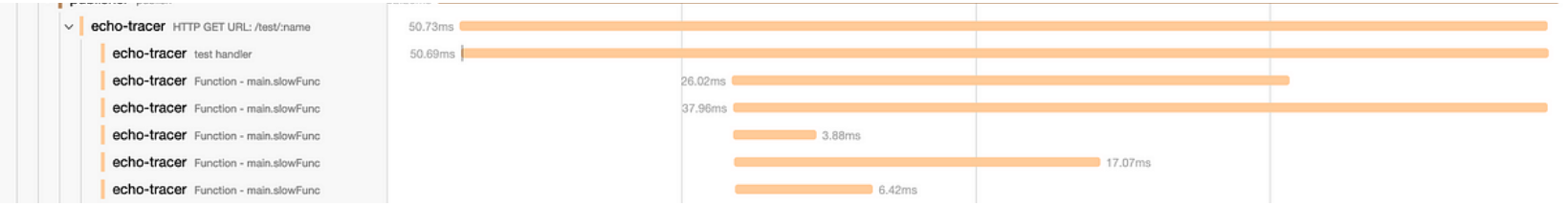




Great, now what?



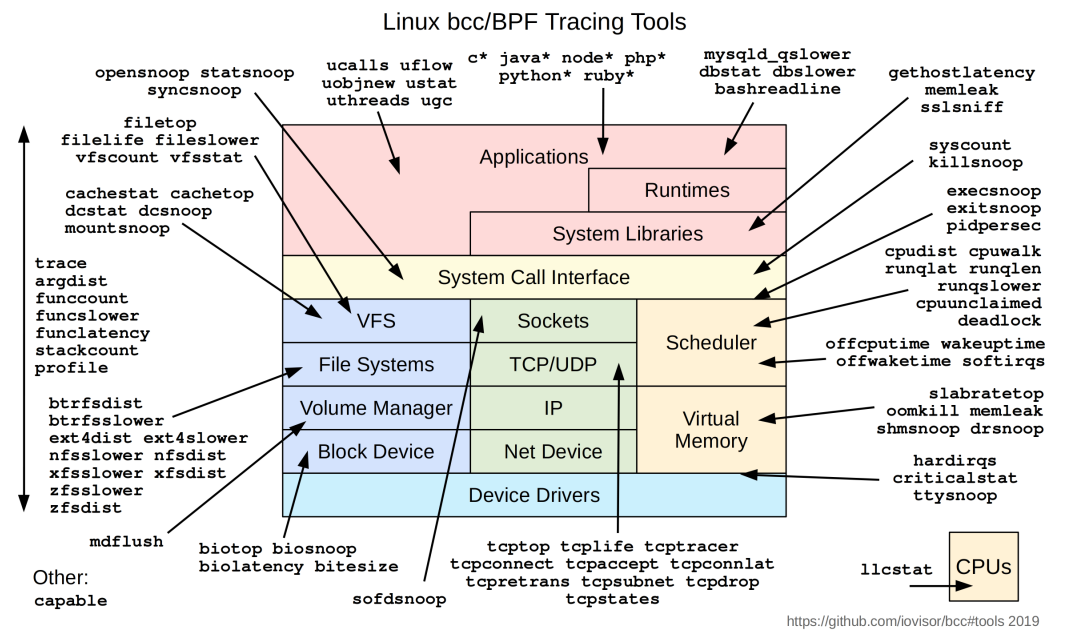




The Horsemen of High Latency


Gotta
Trace
'em all!

Trace



https://github.com/iovisor/bcc#tools 2019

Usual Tracing

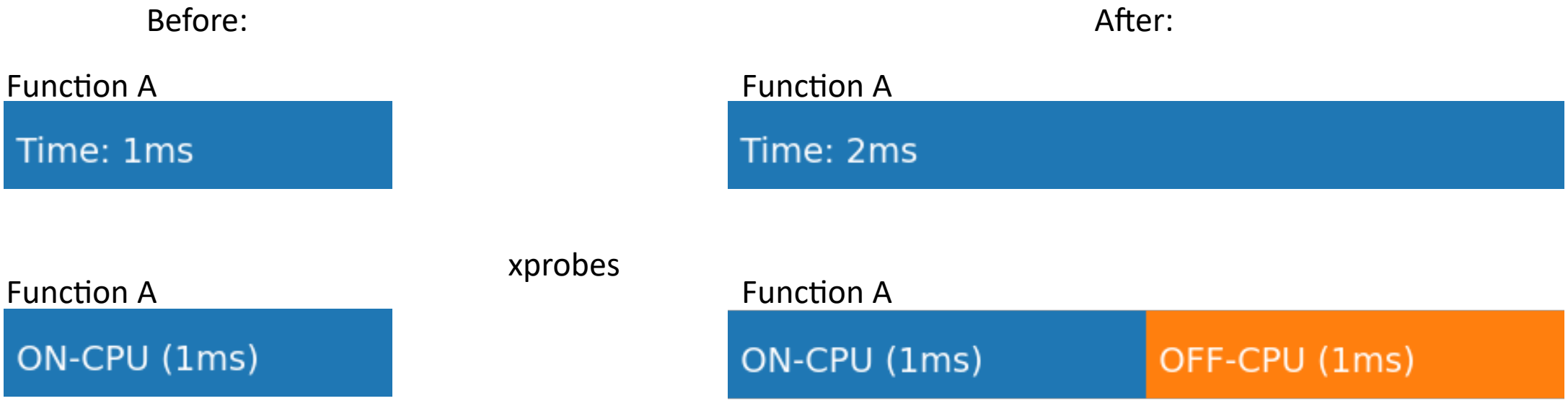
Before:

Function A

After:

Function A

Xprobe’s Kernel-informed Tracing



Xprobe’s Kernel-informed Tracing

Before:

Function A

Time: 1ms

Function A

ON-CPU (1ms)

xprobes

After:

Function A

Time: 2ms

Function A

ON-CPU (1ms) | OFF-CPU (1ms)

```
taskset -c 20 ./measure
```

```
taskset -c 20 ./noise &
taskset -c 20 ./measure
```

Generalized Xprobes

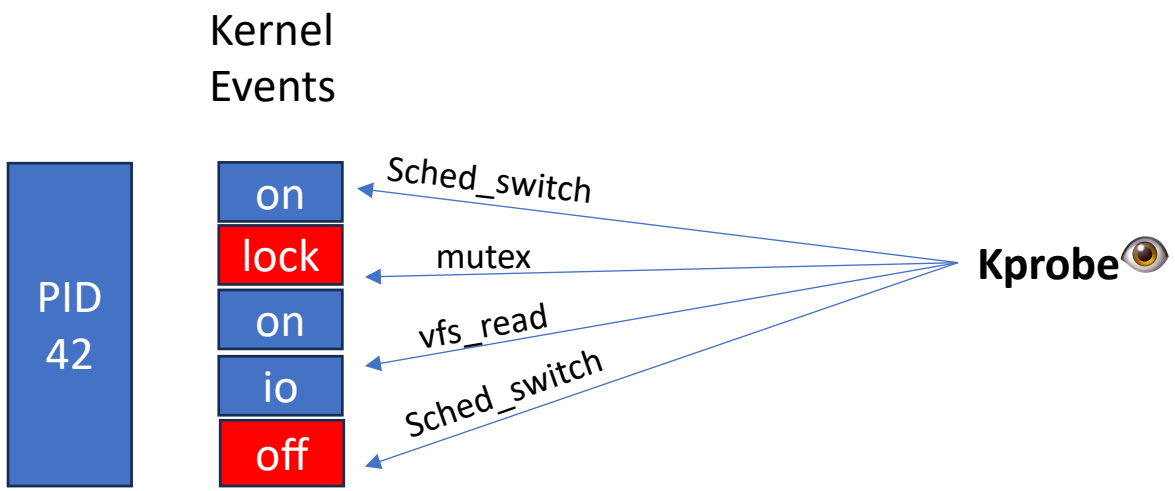
Scenario	Usual Tracing	Xprobes
Baseline	Function A	Function A: on, io, off
CPU contention	Function A	Function A: on, io, off
More Work	Function A	Function A: on, io, off
Lock contention	Function A	Function A: on, io, off, lock
More IO	Function A	Function A: on, io, off
Busy Disk	Function A	Function A: on, io, off, lock

Generalized Xprobes

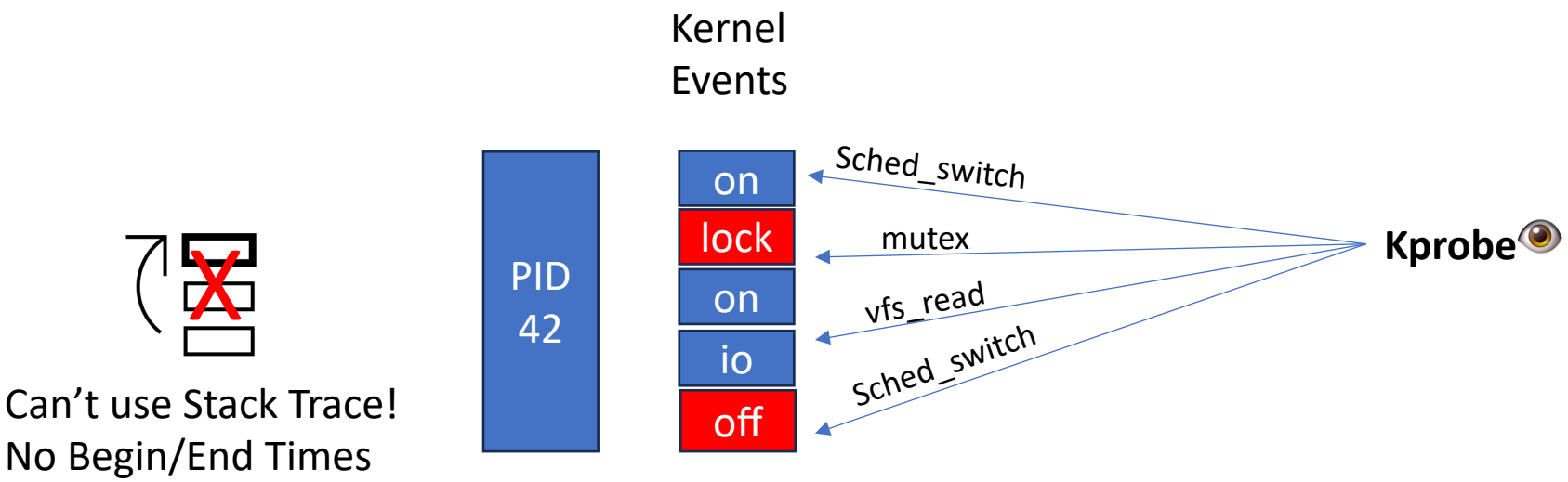
Scenario	Usual Tracing	Xprobes
Baseline	Function A	Function A: on, io, off
CPU con	Function A	Function A: off
More V		off
Lock contention	Function A	Function A: on, io, off, lock
More IO	Function A	Function A: on, io, off
Busy Disk	Function A	Function A: on, io, off, lock

How do we get all this information?

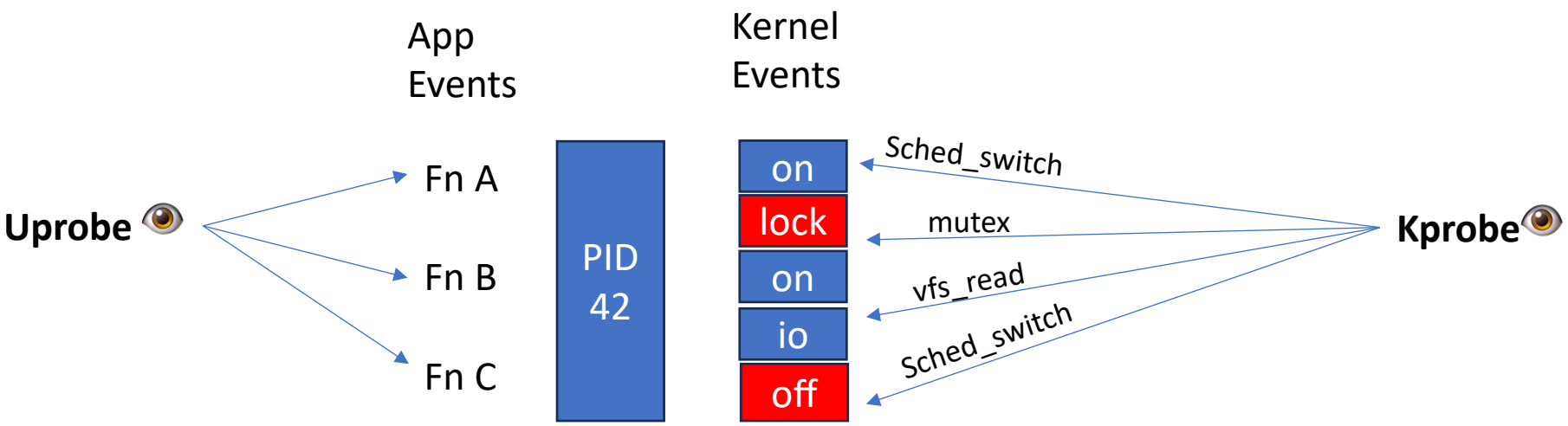
How to get all that information?



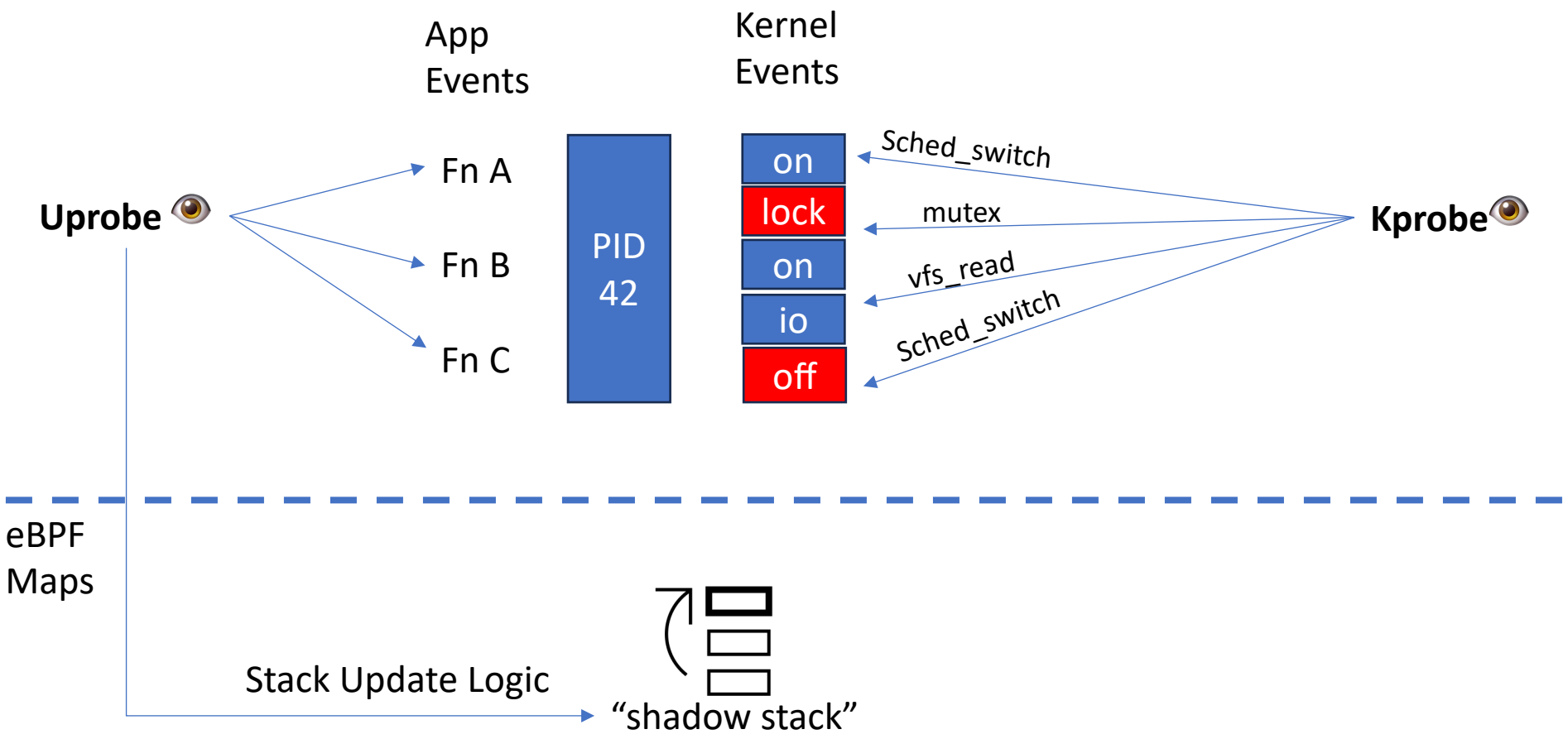
How to get all that information?



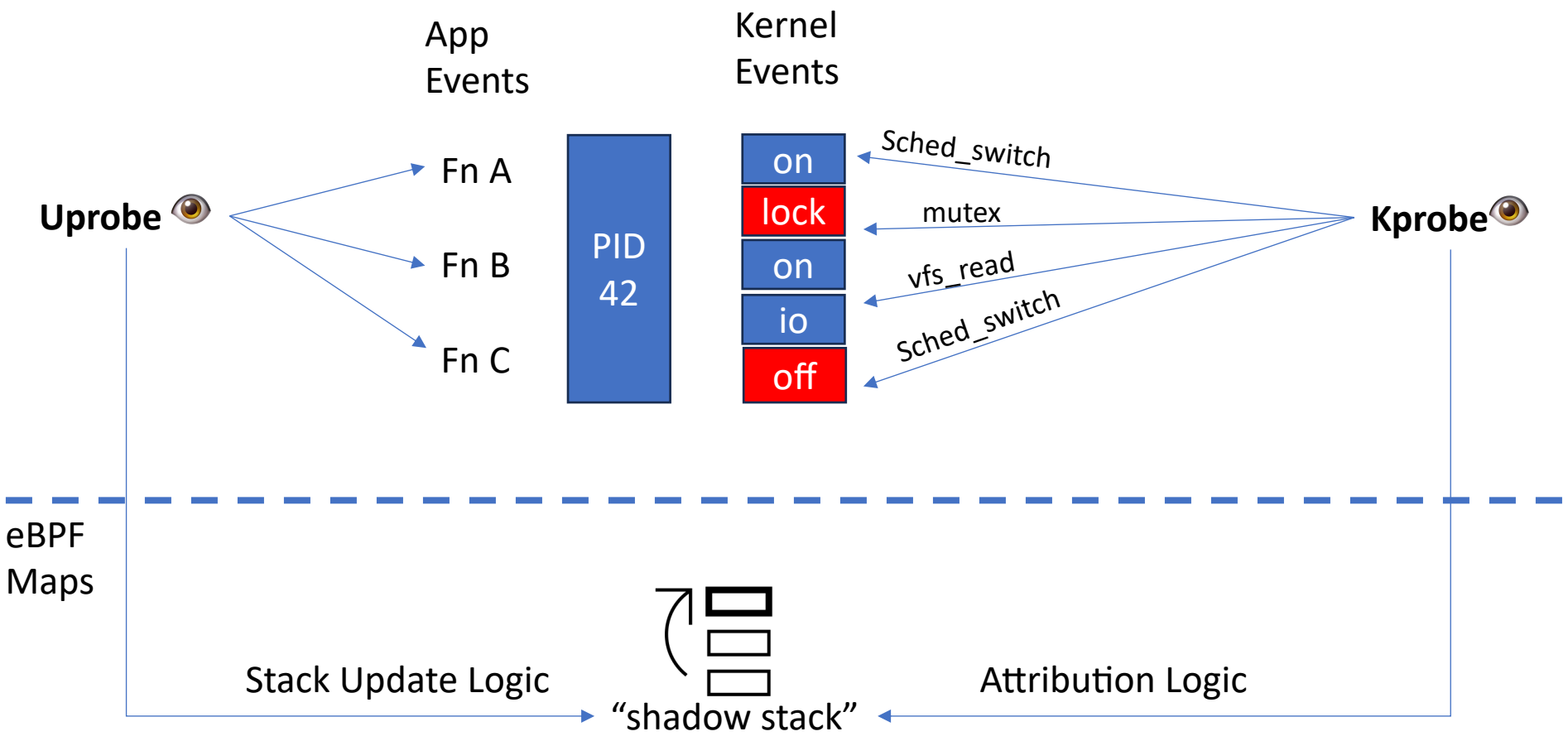
How to get all that information?



How to get all that information?

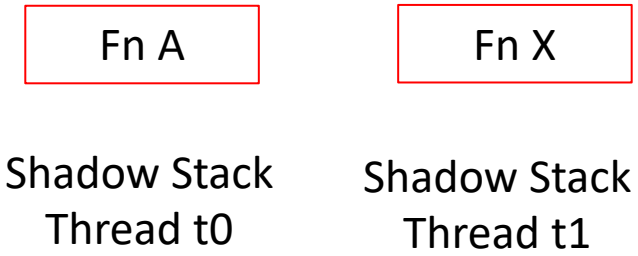


How to get all that information?



Attribution Logic – ONCPU Time

Maps



Attribution Logic – ONCPU Time

Maps

Fn A

Shadow Stack
Thread t0

Fn X

Shadow Stack
Thread t1

Events

Time	Thread	uprobe Events	kprobe Events
1	t0	Fn B() 1	

Attribution Logic – ONCPU Time

Maps

ONCPU

1

Fn B

Fn A

Shadow Stack
Thread t0

Fn X

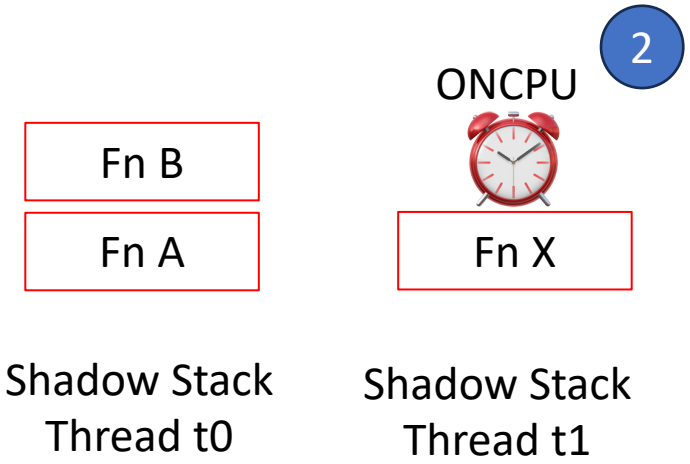
Shadow Stack
Thread t1

Events

Time	Thread	uprobe Events	kprobe Events
1	0	Fn B() 1	

Attribution Logic – ONCPU Time

Maps

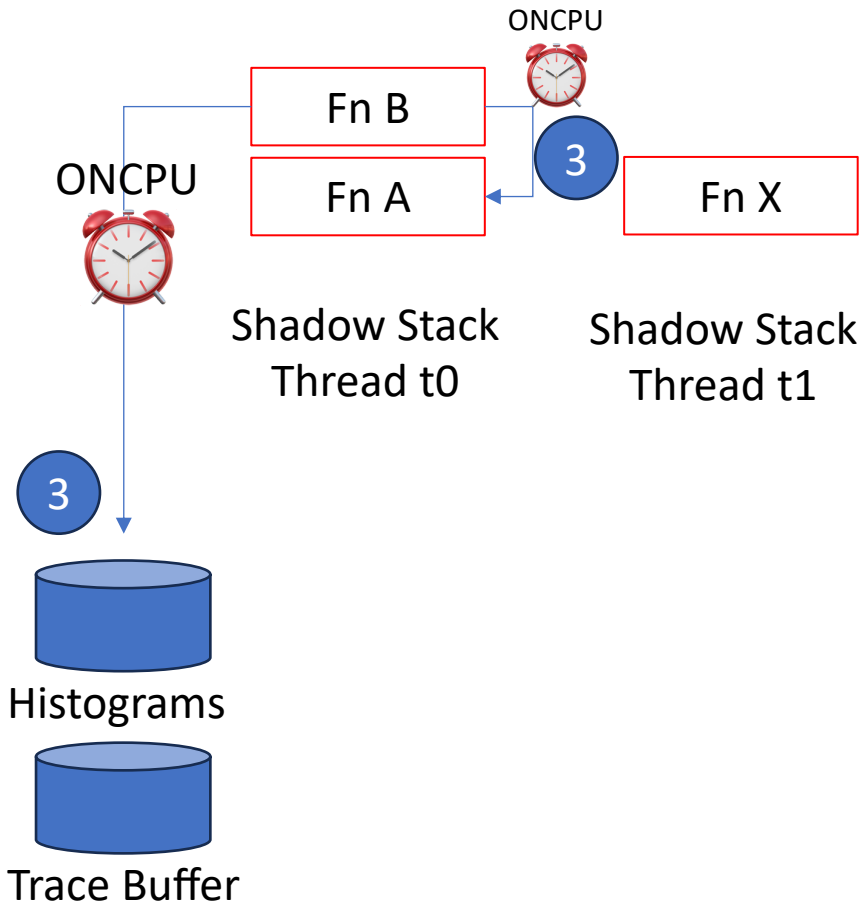


Events

Time	Thread	uprobe Events	kprobe Events
1	0	Fn B()	
2	-		2 Sched_switch(t0,t1)

Attribution Logic – ONCPU Time

Maps

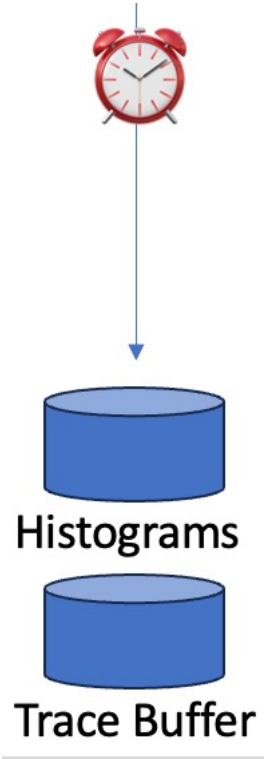


Events

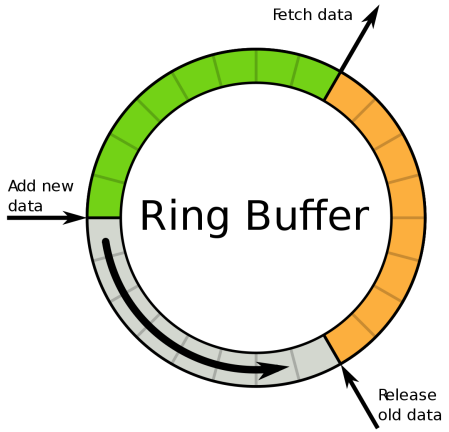
Time	Thread	uprobe Events	kprobe Events
1	0	Fn B()	
2	-		Sched_switch(t0,t1)
3	-		Sched_switch(t1,t0)
4	0	Exit Fn B()	

3

Storing Logic



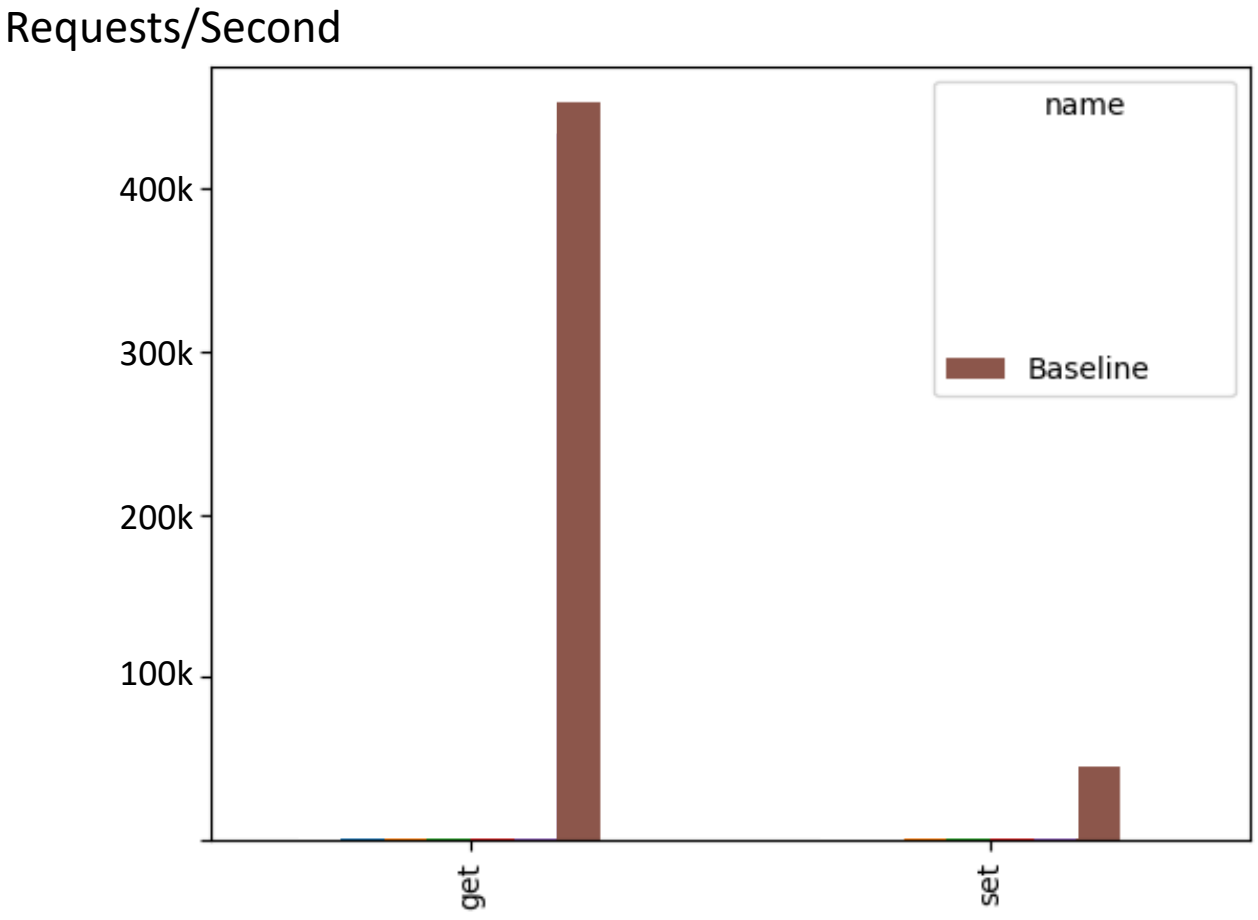

```
usecs               : count     distribution
    0 -> 1          : 2810     |*                                       |
    2 -> 3          : 5248     |**                                      |
    4 -> 7          : 12369    |******                                  |
    8 -> 15         : 71312    |****************************************|
   16 -> 31         : 55705    |******************************          |
   32 -> 63         : 11775    |******                                  |
   64 -> 127        : 6230     |***                                     |
  128 -> 255        : 2758     |*                                       |
  256 -> 511        : 549      |                                        |
  512 -> 1023       : 46       |                                        |
 1024 -> 2047       : 11       |                                        |
 2048 -> 4095       : 4        |                                        |
 4096 -> 8191       : 5        |                                        |
```




UPROBE
OVERHEAD
Ferdi Rizkiyanto

Redis Benchmark (Memtier)

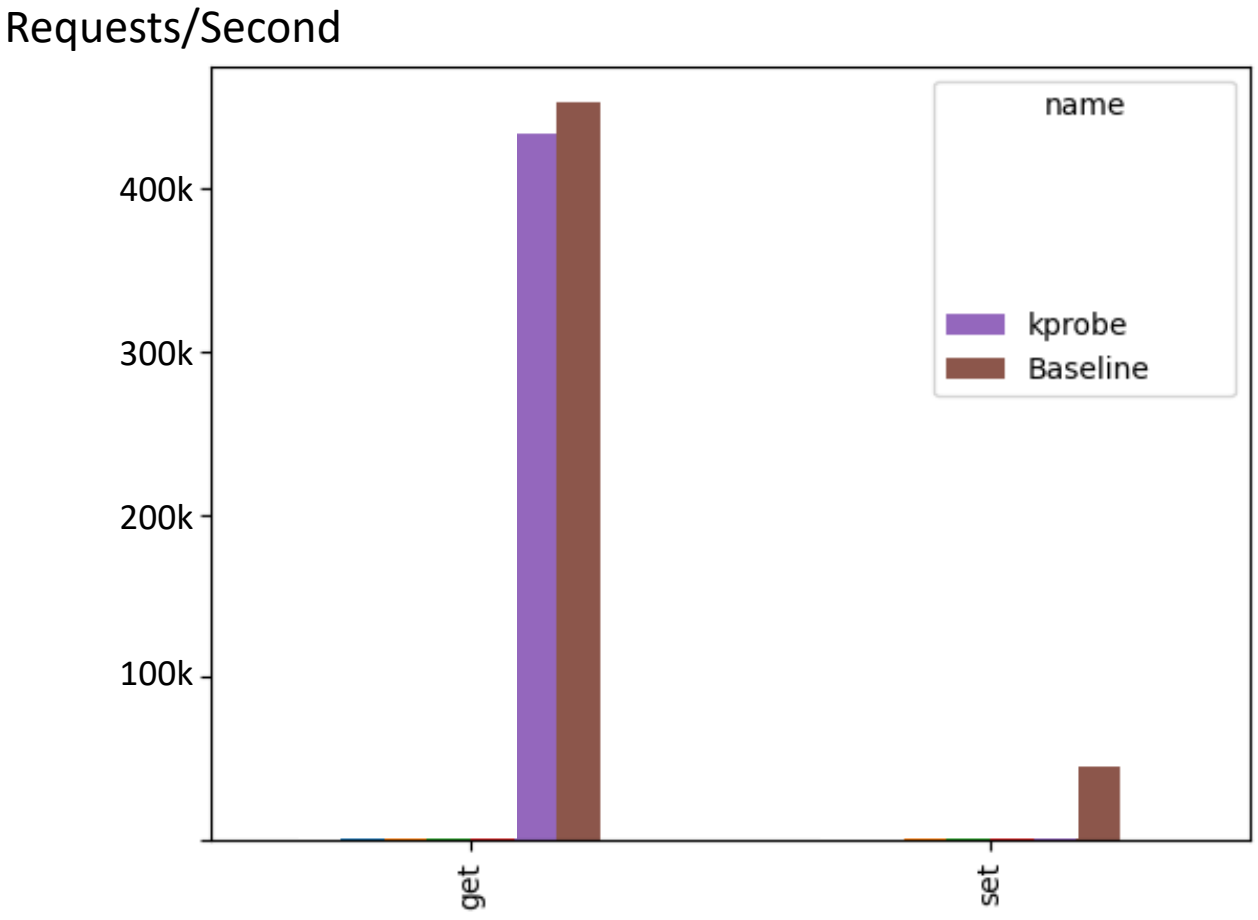
Takeaways



https://github.com/RedisLabs/memtier_benchmark

Redis Benchmark (Memtier)

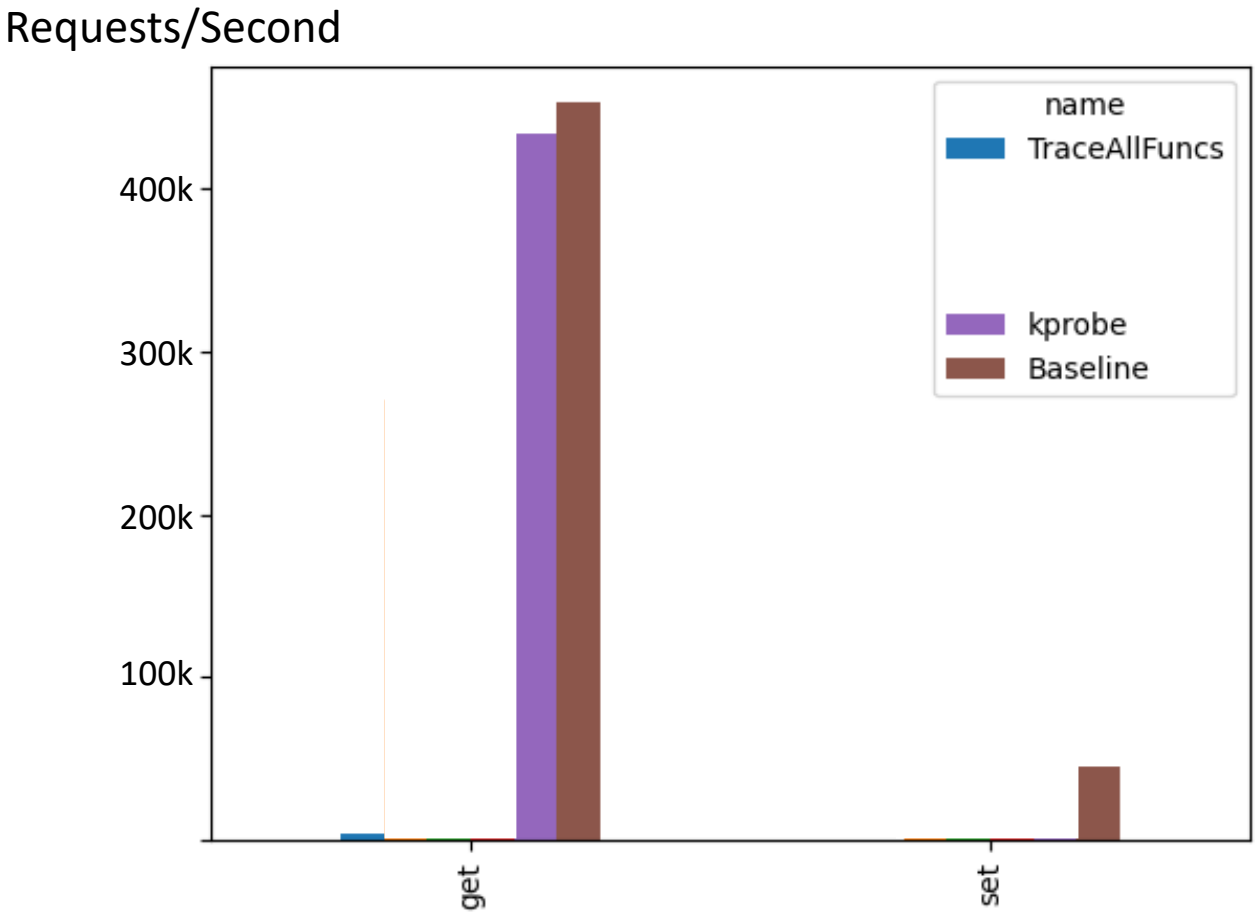
Takeaways



Redis Benchmark

Takeaways

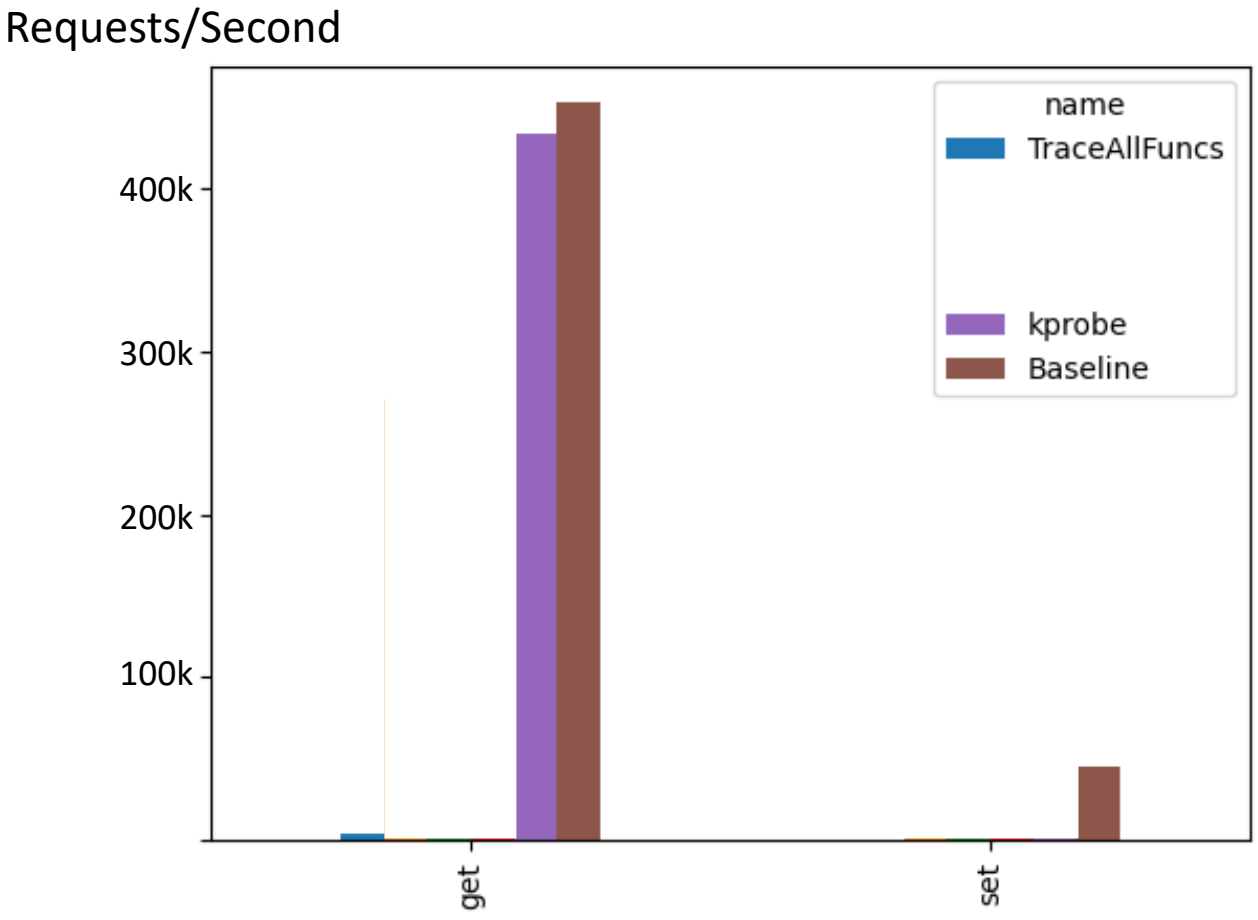
- TraceAllFuncs (kprobes+uprobes)
  - Tracing all ~5700 Redis functions is undoable



Redis Benchmark

Takeaways

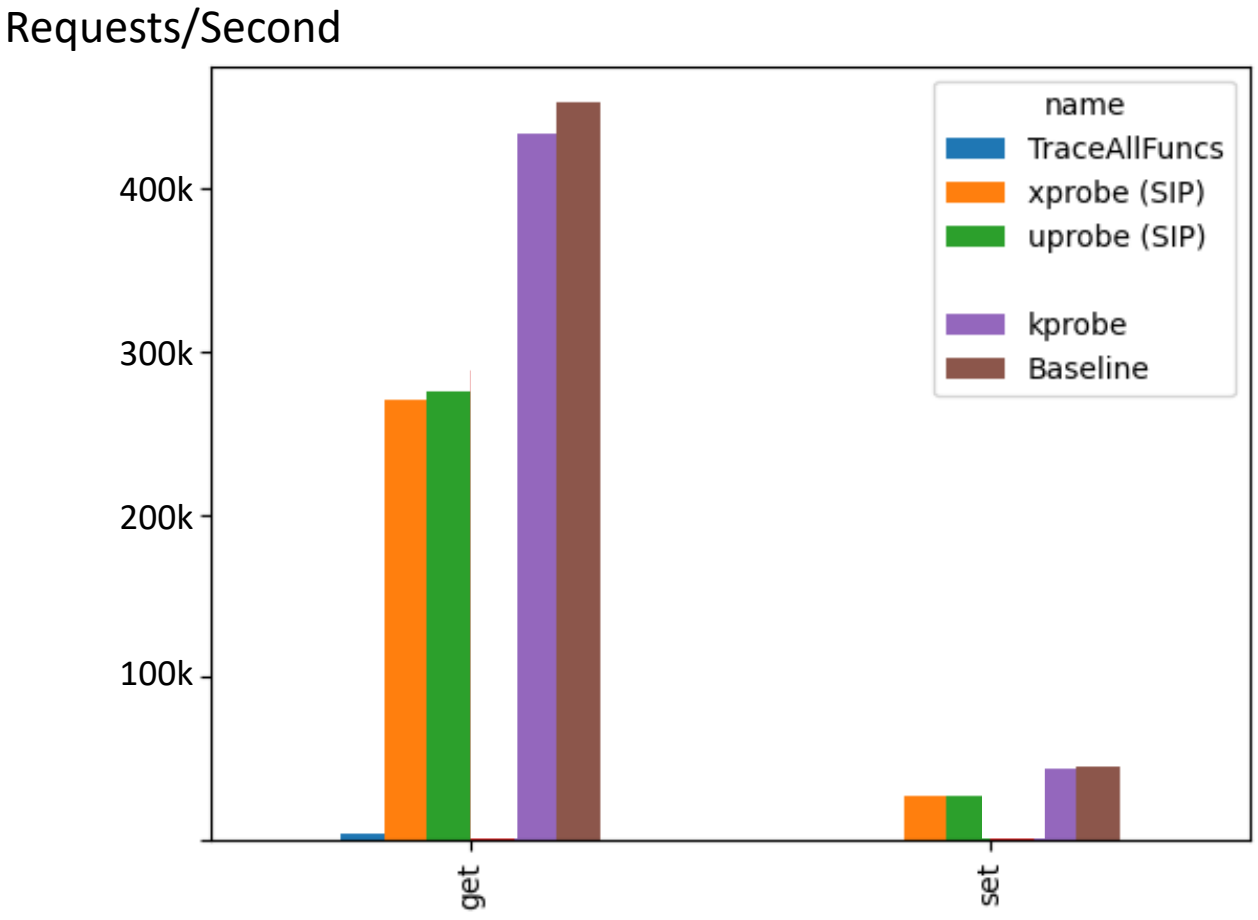
- TraceAllFuncs
  - Tracing all ~5700 Redis functions is undoable
- Trace1Func (siphash)



Redis Benchmark

Takeaways

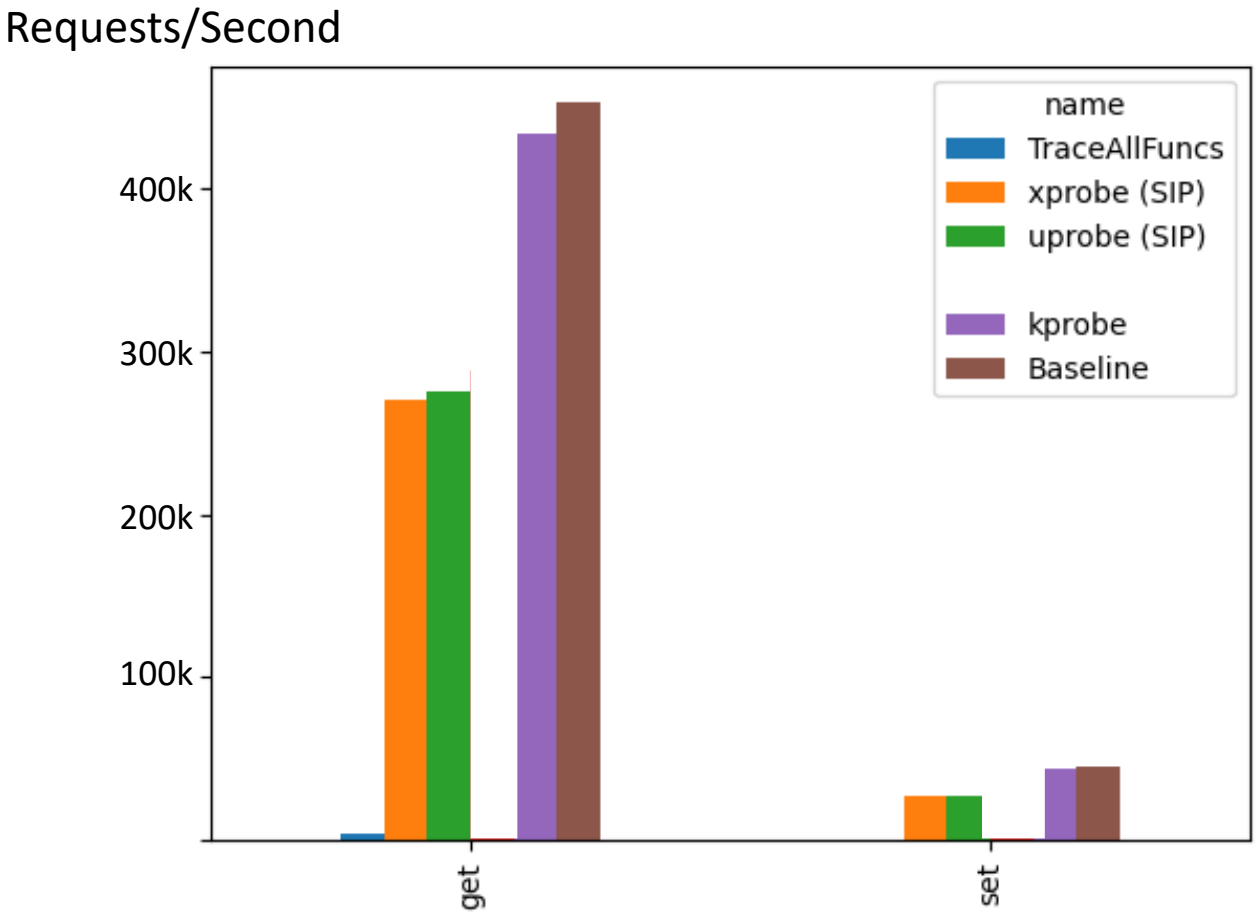
- TraceAllFuncs
  - Tracing all ~5700 Redis functions is undoable
- Trace1Func (siphash)
  - Almost all performance hit from uprobe



Redis Benchmark

Takeaways

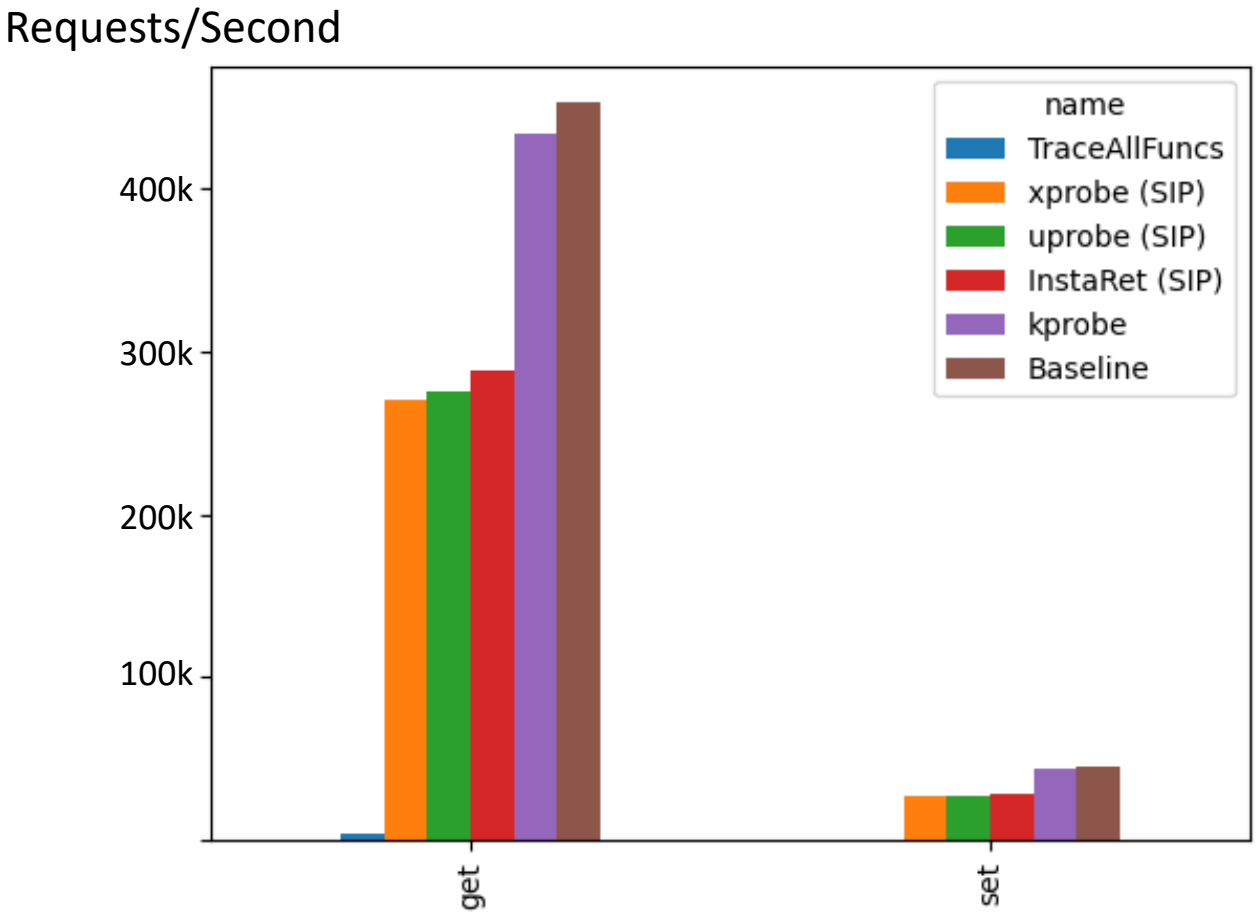
- TraceAllFuncs
  - Tracing all ~5700 Redis functions is undoable
- Trace1Func (siphash)
  - Almost all performance hit from uprobe
- InstaRet: Uprobe which retuns immediately
  -
  -



Redis Benchmark

Takeaways

- TraceAllFuncs
  - Tracing all ~5700 Redis functions is undoable
- Trace1Func (siphash)
  - Almost all performance hit from uprobe
- InstaRet: Uprobe which retuns immediately
  - As expected, almost all performance hit from uprobe trap
    - Sampling cannot help us here



The State of Xprobes

Functionality-wise:

We **WANT** the benefits of a **fully eBPF** solution
(i.e., using both kprobes for kernel events
and uprobes for userspace functions)

✅ Dynamic (Re)instrumentation
✅ “Safe” Code
✅ Wide Observability Adoption

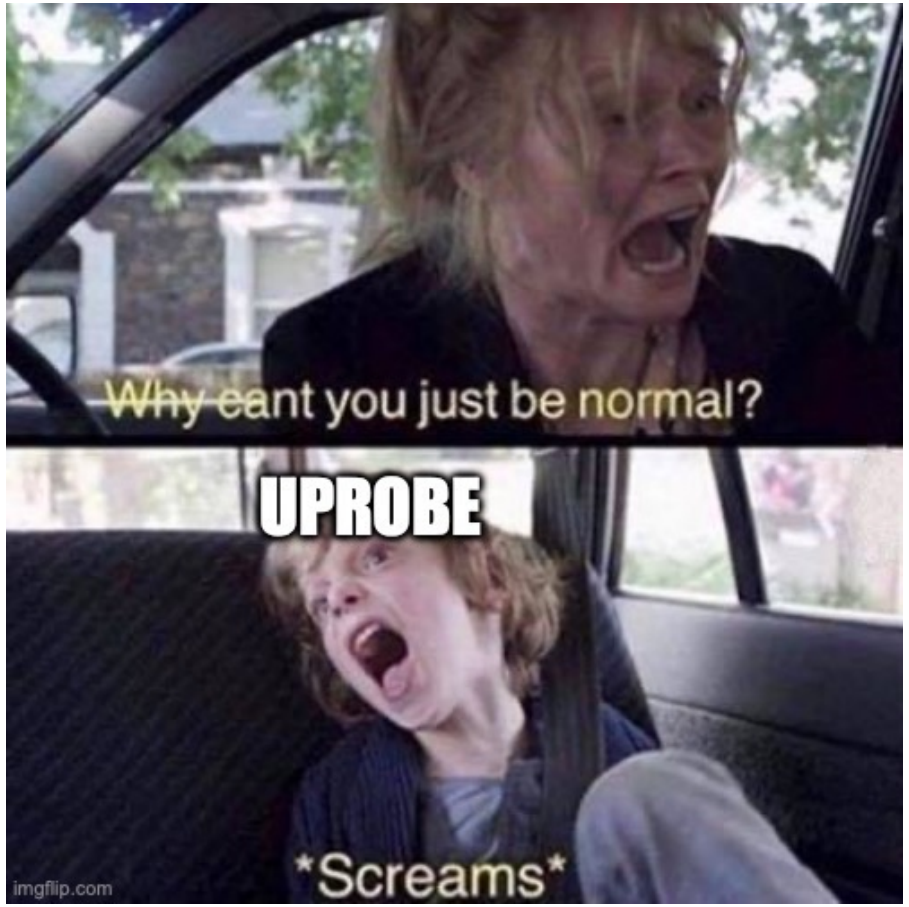
The State of Xprobes

Functionality-wise:

We **WANT** the benefits of a **fully eBPF** solution (i.e., using both kprobes for kernel events and uprobes for userspace functions)

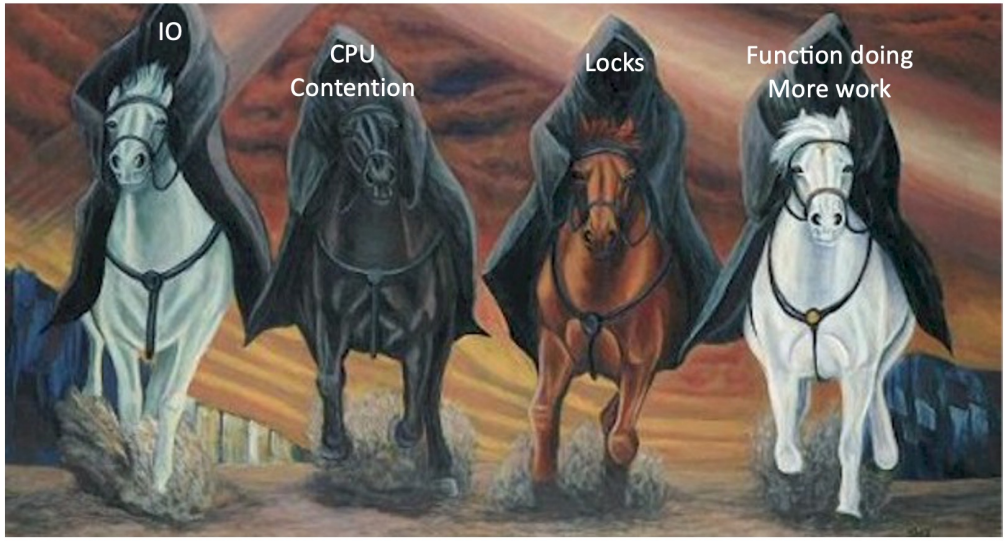
✅ Dynamic (Re)instrumentation
✅ “Safe” Code
✅ Wide Observability Adoption

Overhead-wise…



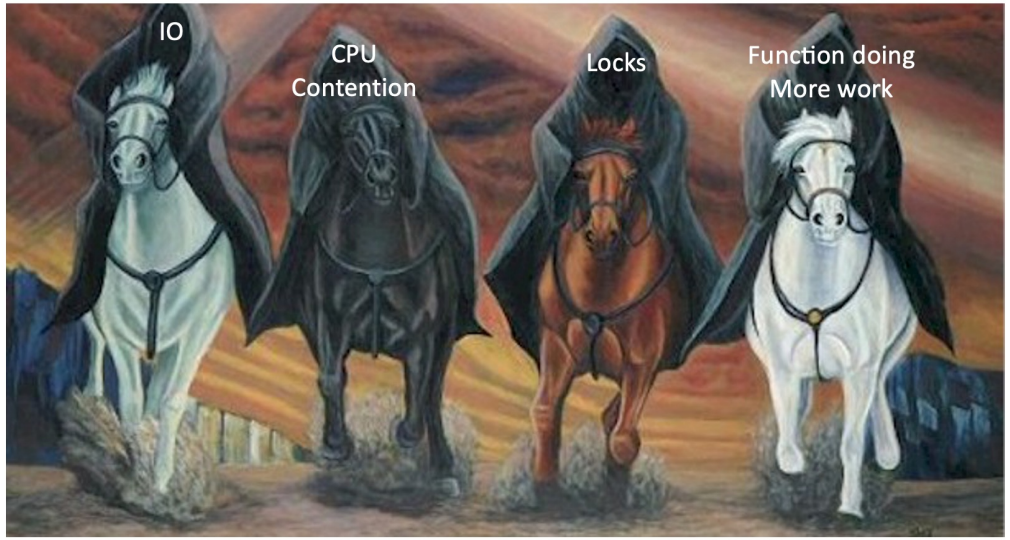
What next?

Improve uprobes or just not use them?



What other horsemen did we miss?

What next?



- Thank you! And please don't hesitate to reach out!
  - lcas@cmu.edu
  - Theophilus@cmu.edu