PERFORMANCE @ SCALE

Feb 2016

Linux 4.x Performance Using BPF Superpowers

Brendan Gregg
Senior Performance Architect

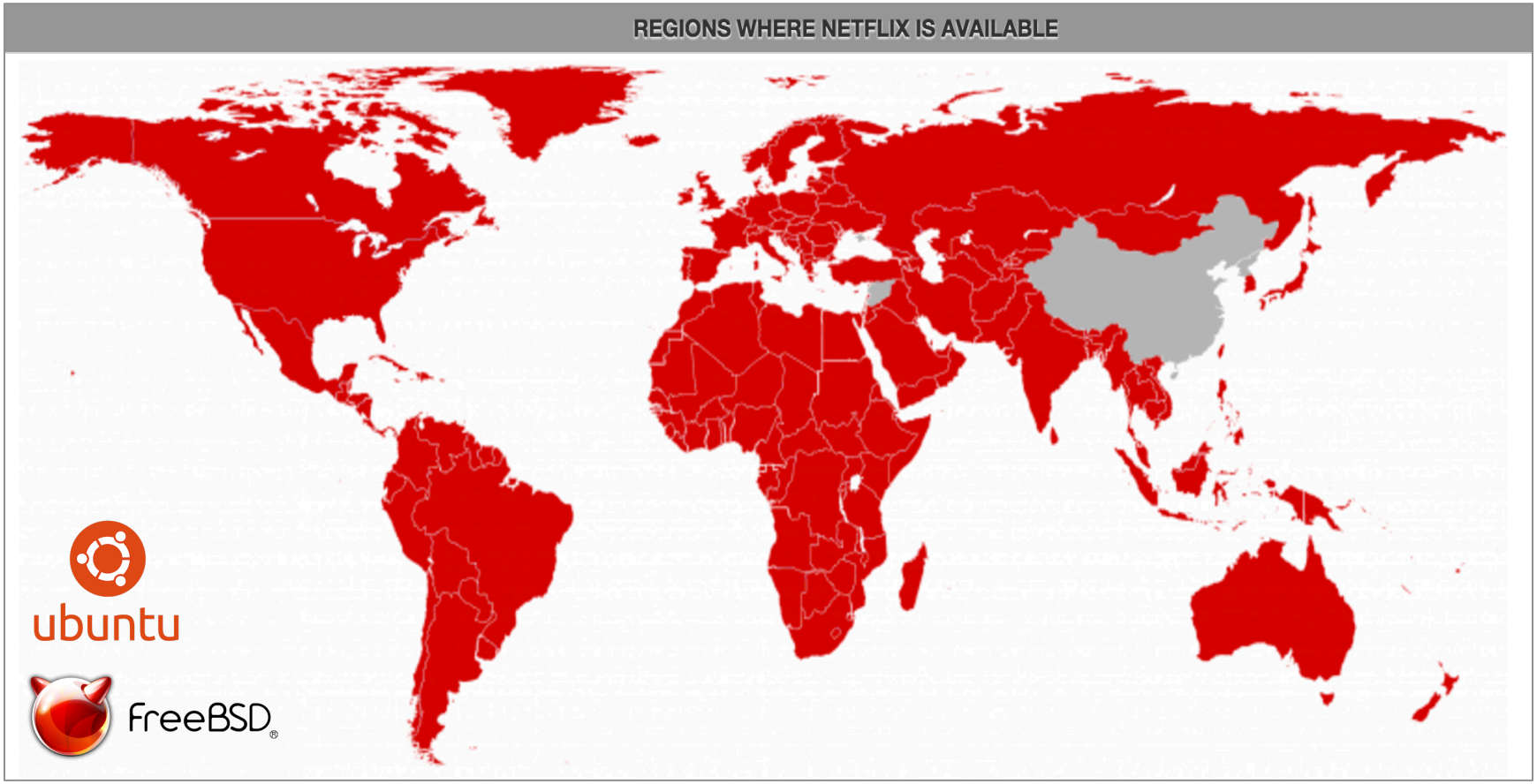
NETFLIX

MPK17 Third Floor

Flexible Offices

Ten years ago,
I gave a talk here
about DTrace tools…

NETFLIX

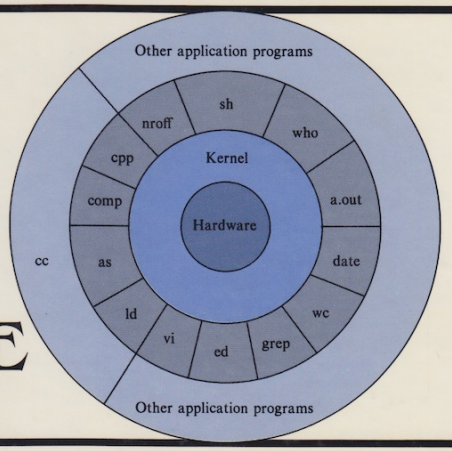


Superpowers are coming to Linux

Solve performance issues that were previously impossible

For example, full off-CPU analysis…

THE DESIGN OF THE UNIX® OPERATING SYSTEM



MAURICE J. BACH

PRENTICE-HALL SOFTWARE SERIES

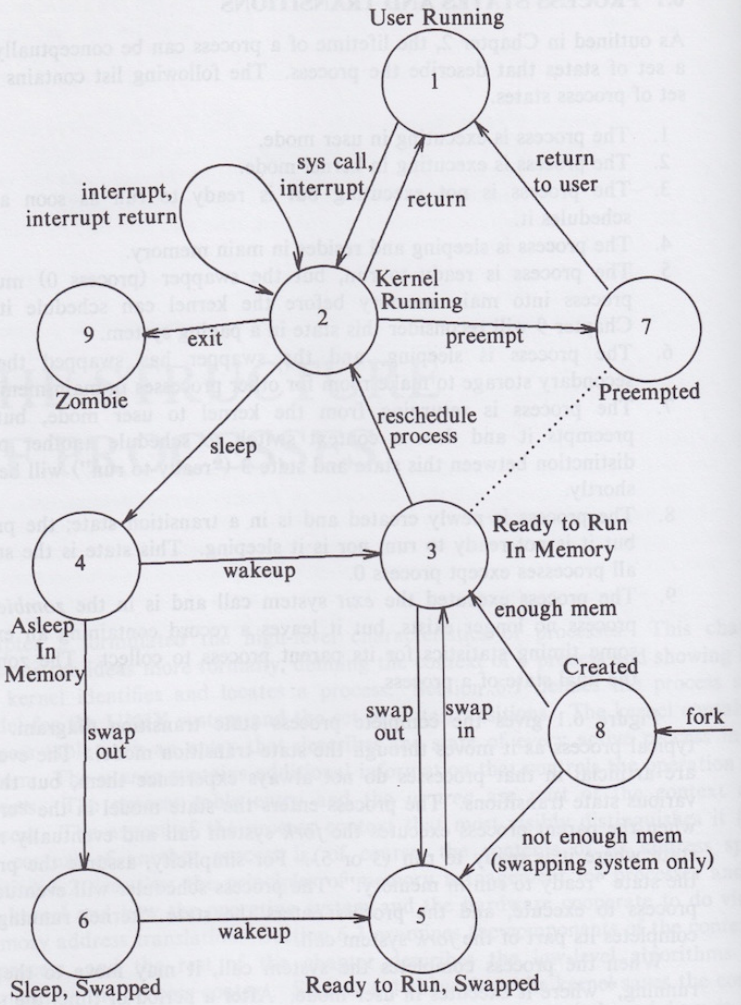
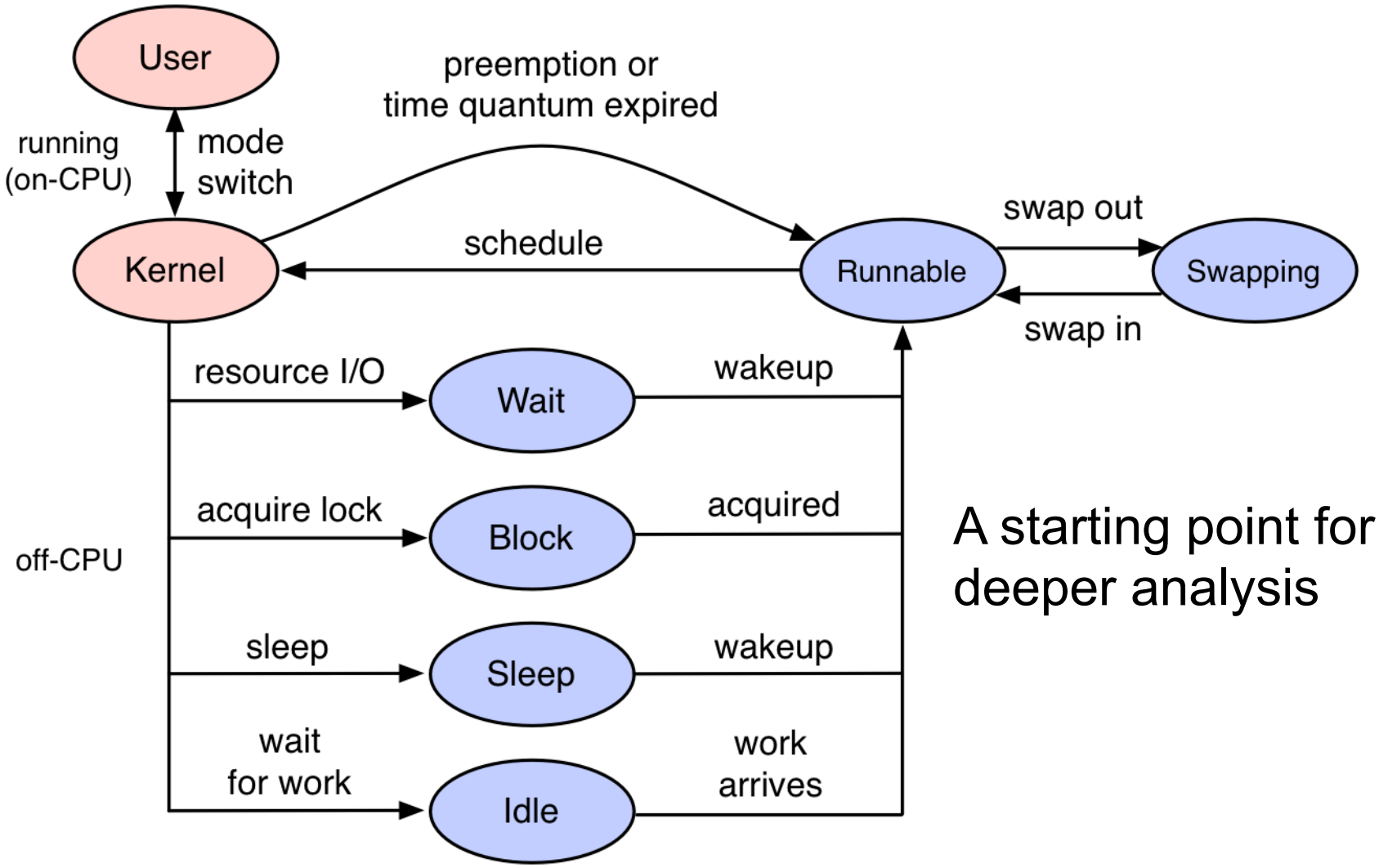


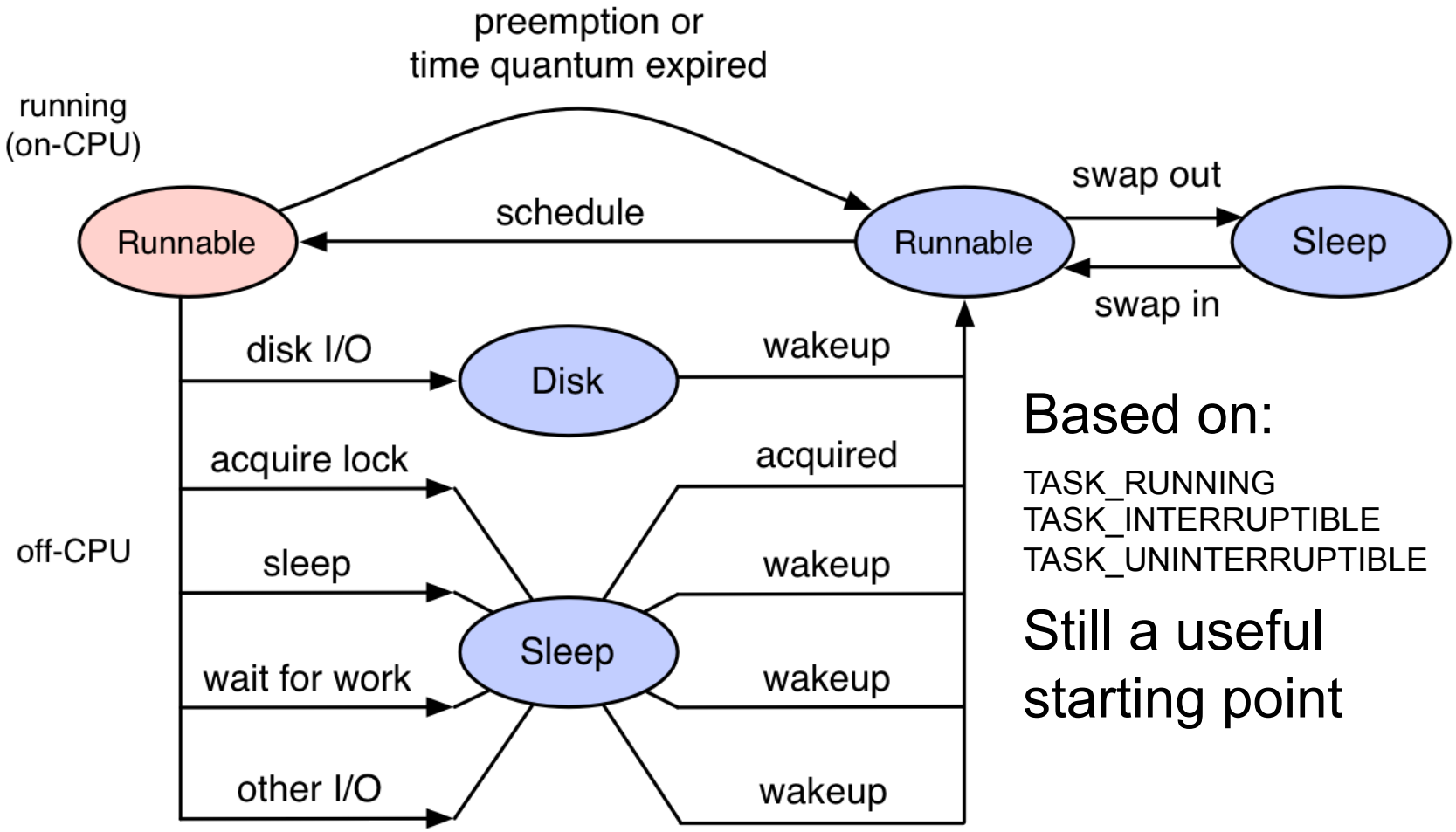
Figure 6.1. Process State Transition Diagram

Ideal Thread States



A starting point for deeper analysis

Linux Thread States



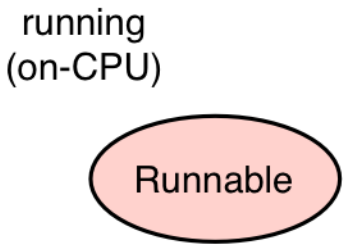
Based on:

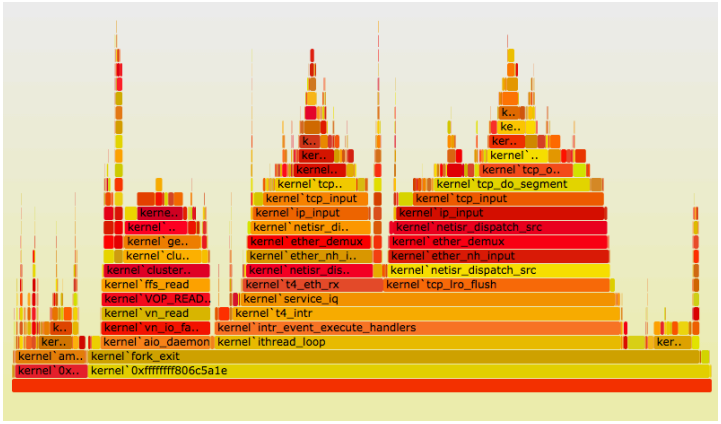
TASK_RUNNING
TASK_INTERRUPTIBLE
TASK_UNINTERRUPTIBLE

Still a useful starting point

Linux On-CPU Analysis

- I'll start with on-CPU analysis:

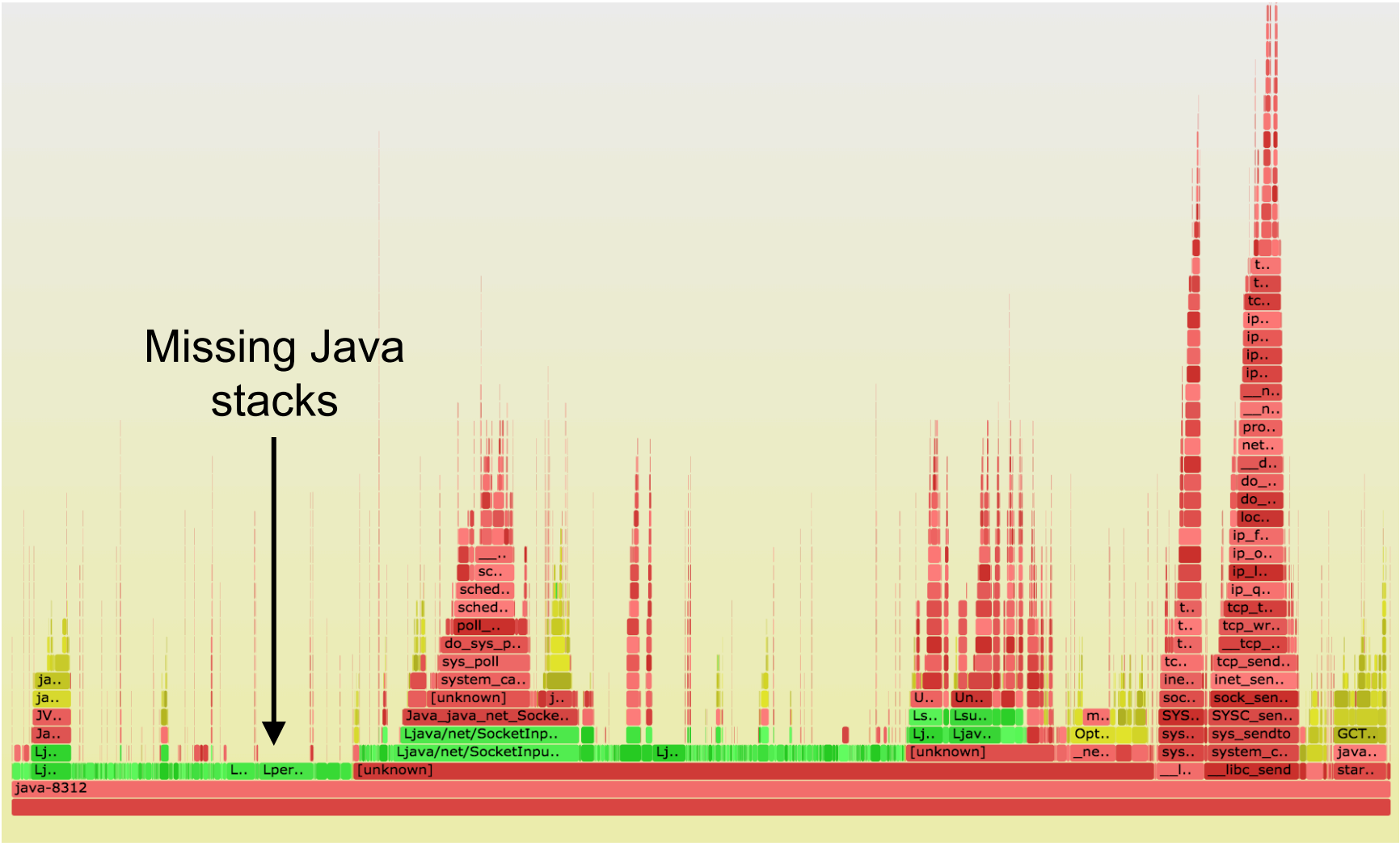




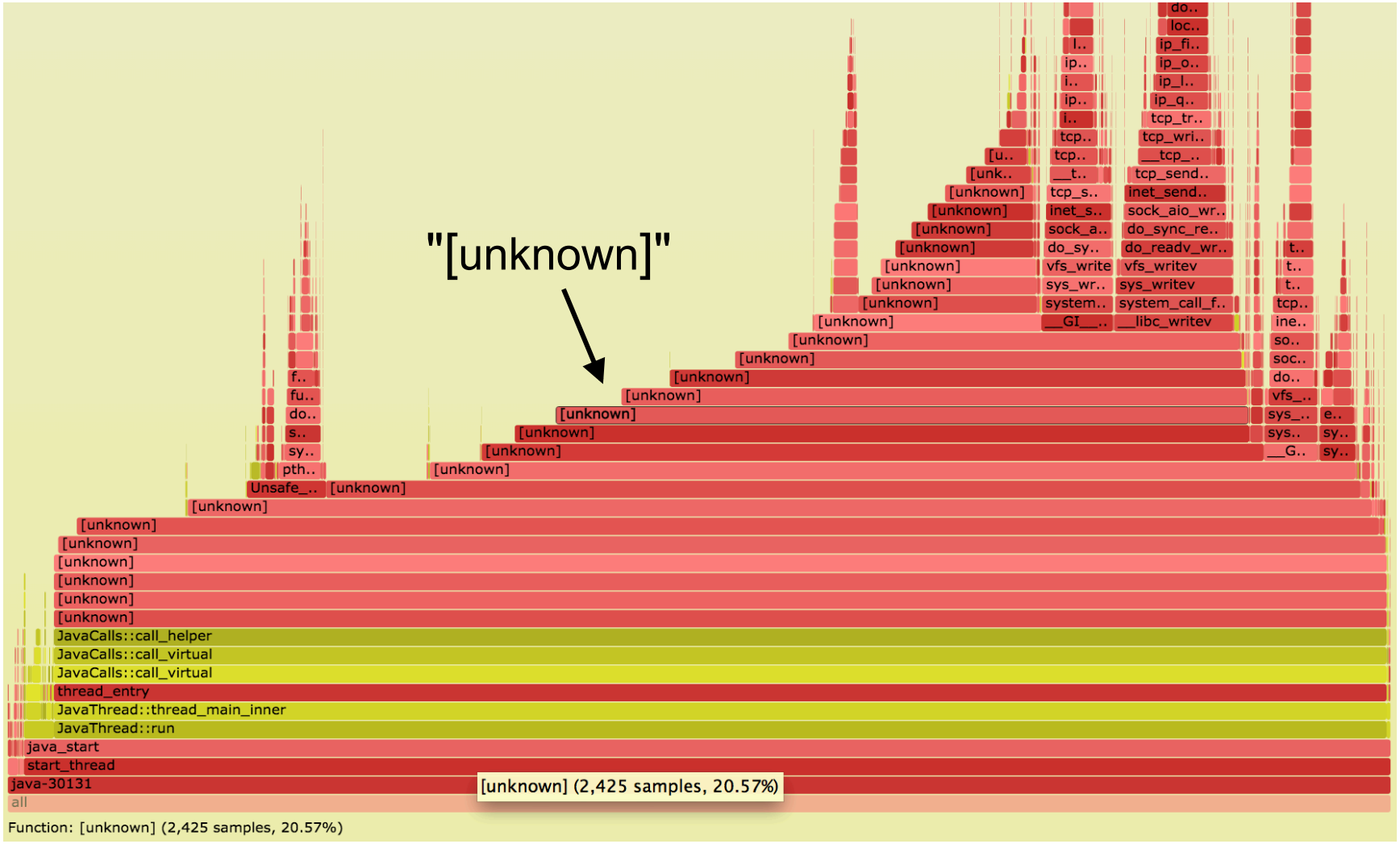
CPU Flame Graph

- Split into user/kernel states using /proc, mpstat(1), ...
- perf_events ("perf") to analyze further:
  - User & kernel stack sampling (as a CPU flame graph)
  - CPI
  - Should be easy, but…

Broken Stacks

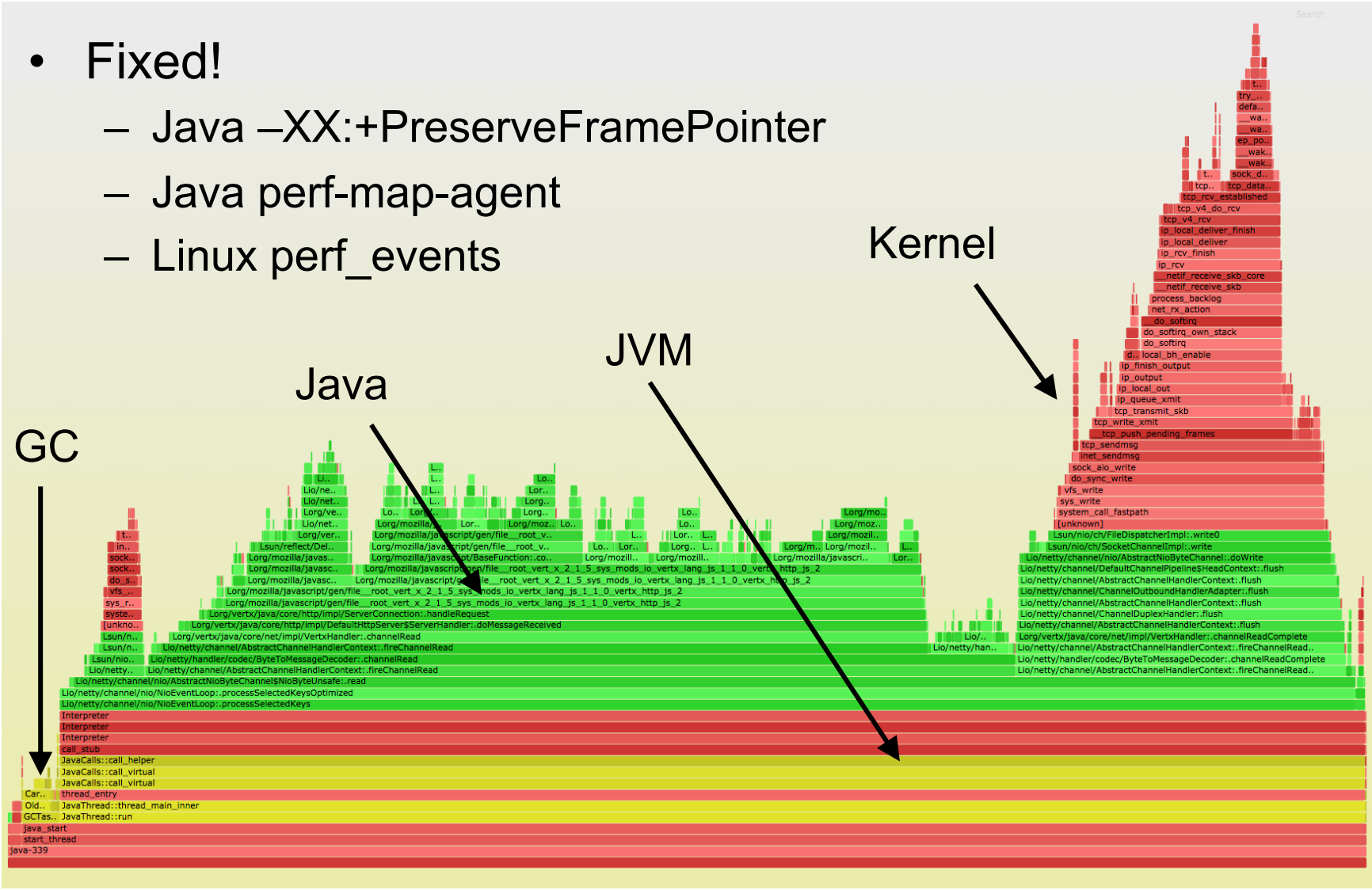


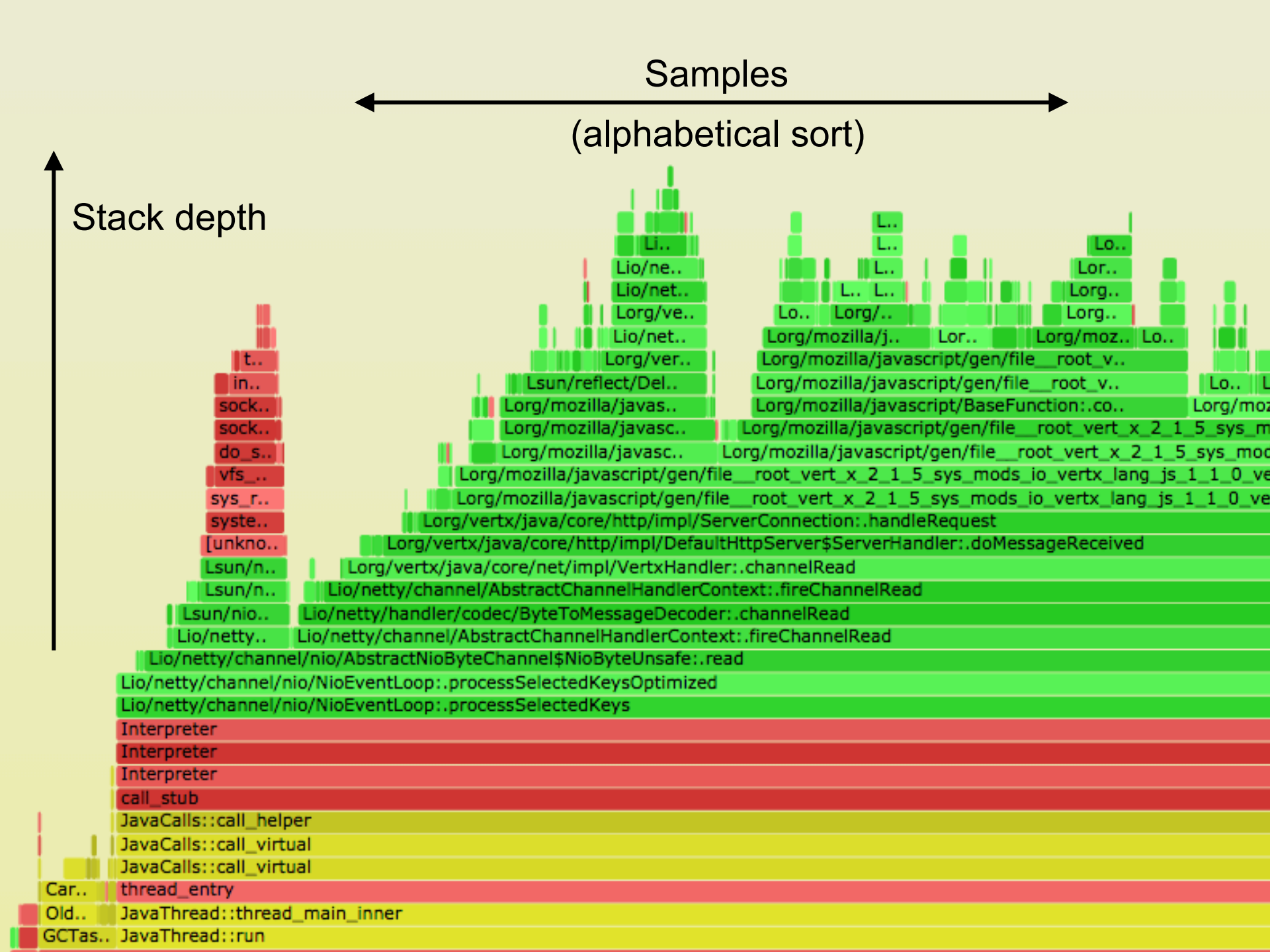
Missing Symbols



Java Mixed-Mode CPU Flame Graph

- Fixed!
  - Java –XX:+PreserveFramePointer
  - Java perf-map-agent
  - Linux perf_events



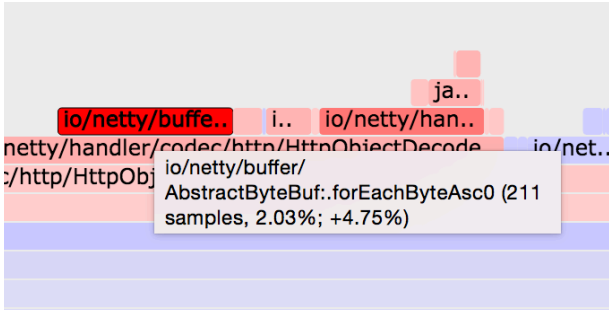

Samples
(alphabetical sort)
Stack depth
t..
in..
sock..
sock..
do_s..
vfs_..
sys_r..
syste..
[unkno..
Lsun/n..
Lsun/n..
Lsun/nio..
Lio/netty..
Li..
Lio/ne..
Lio/net..
Lorg/ve..
Lio/net..
Lorg/ver..
Lsun/reflect/Del..
Lorg/mozilla/javas..
Lorg/mozilla/javasc..
Lorg/mozilla/javasc..
Lorg/mozilla/javascript/gen/file__root_vert_x_2_1_5_sys_mods_io_vertx_lang_js_1_1_0_ve
Lorg/mozilla/javascript/gen/file__root_vert_x_2_1_5_sys_mods_io_vertx_lang_js_1_1_0_ve
Lorg/vertx/java/core/http/impl/ServerConnection:.handleRequest
Lorg/vertx/java/core/http/impl/DefaultHttpServer$ServerHandler:.doMessageReceived
Lorg/vertx/java/core/net/impl/VertxHandler:.channelRead
Lio/netty/channel/AbstractChannelHandlerContext:.fireChannelRead
Lio/netty/handler/codec/ByteToMessageDecoder:.channelRead
Lio/netty/channel/AbstractChannelHandlerContext:.fireChannelRead
Lio/netty/channel/nio/AbstractNioByteChannel$NioByteUnsafe:.read
Lio/netty/channel/nio/NioEventLoop:.processSelectedKeysOptimized
Lio/netty/channel/nio/NioEventLoop:.processSelectedKeys
Interpreter
Interpreter
Interpreter
call_stub
JavaCalls::call_helper
JavaCalls::call_virtual
JavaCalls::call_virtual
Car..
thread_entry
Old..
JavaThread::thread_main_inner
GCTas..
JavaThread::run
L..
L..
L..
L..
L..
Lo..
Lo..
Lor..
Lorg..
Lorg..
Lorg/mozilla/j..
Lor..
Lorg/moz..
Lo..
Lorg/mozilla/javascript/gen/file__root_v..
Lorg/mozilla/javascript/gen/file__root_v..
Lo..
Lorg/mozilla/javascript/BaseFunction:.co..
Lorg/moz
Lorg/mozilla/javascript/gen/file__root_vert_x_2_1_5_sys_m
Lorg/mozilla/javascript/gen/file__root_vert_x_2_1_5_sys_mod

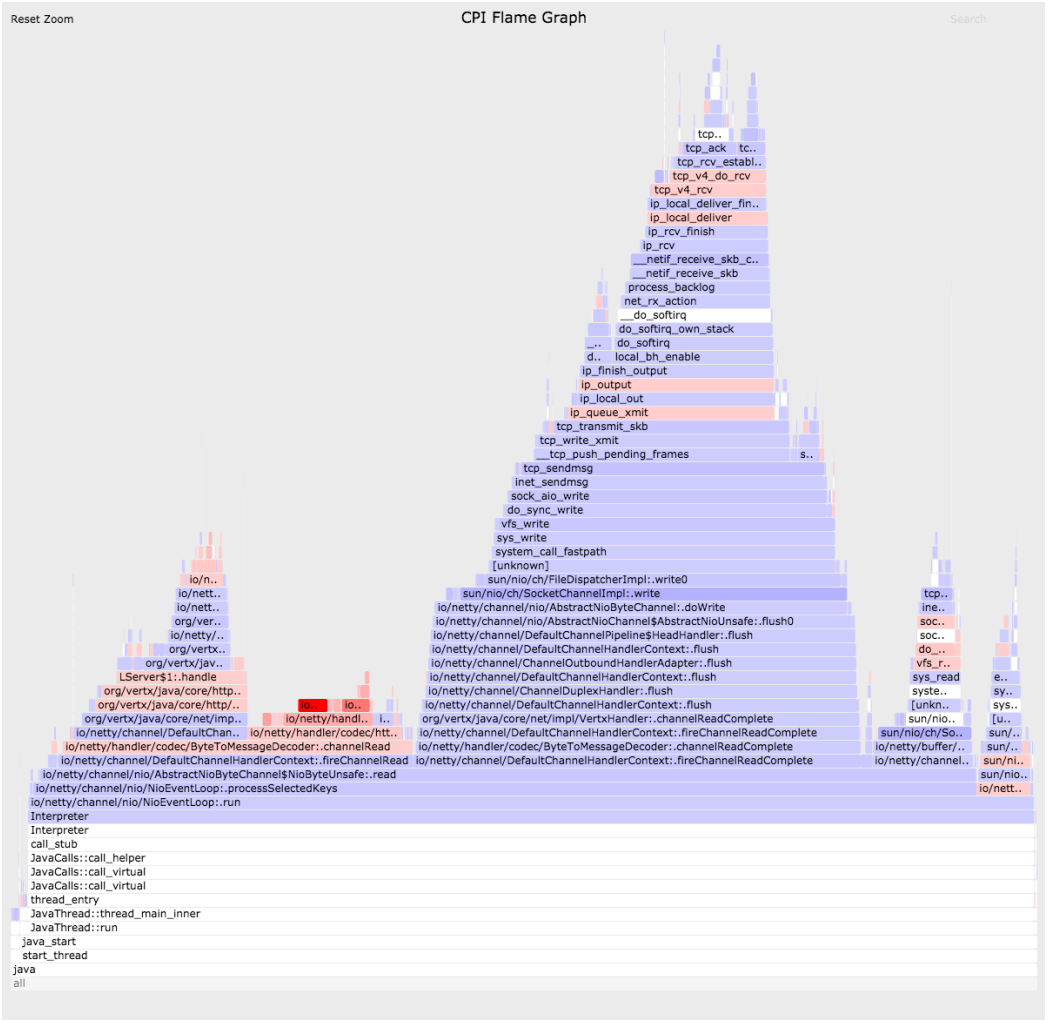
Also, CPI Flame Graph

Cycles Per Instruction

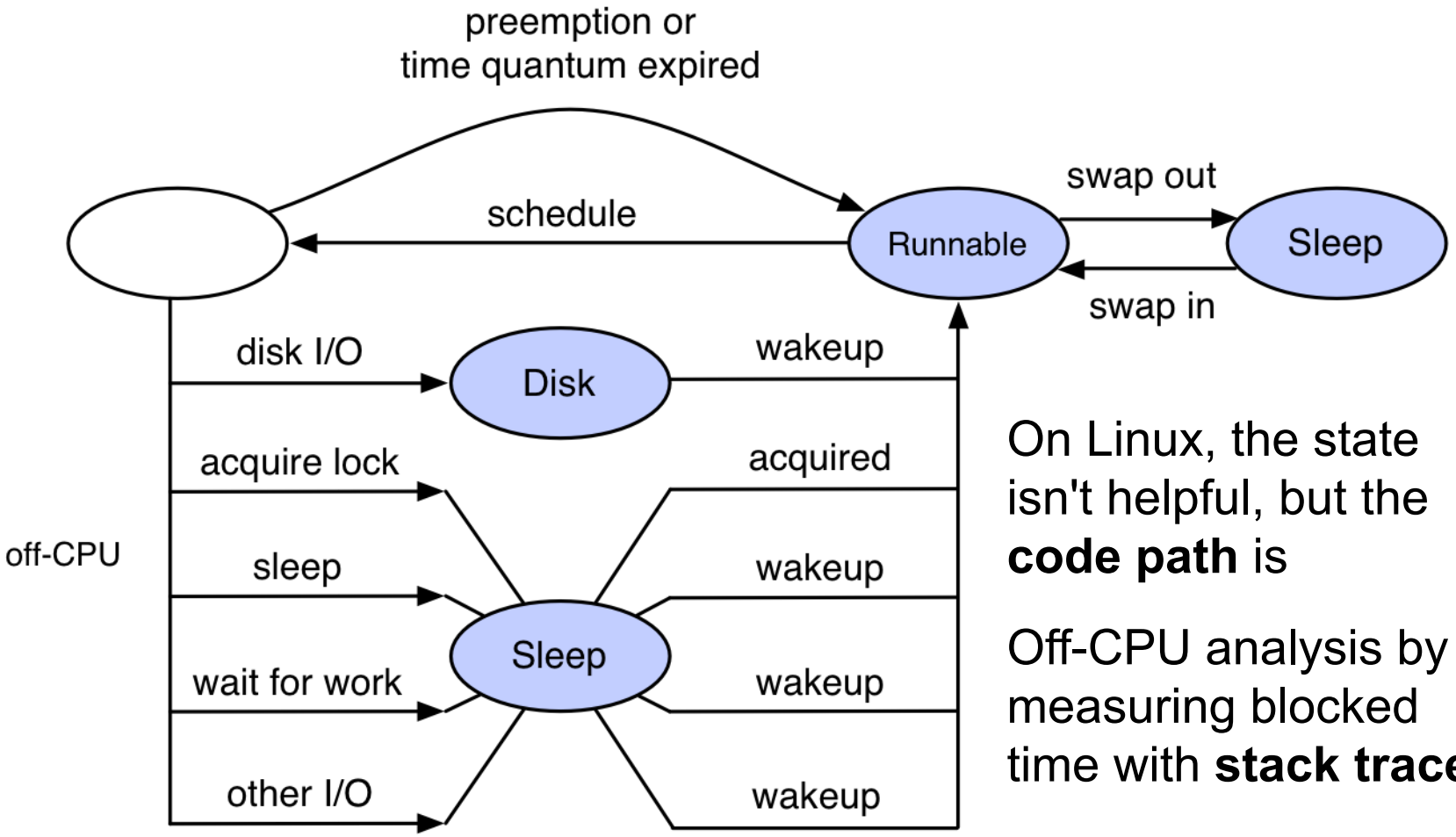
- red == instruction heavy
- blue == cycle heavy (likely mem stalls)

zoomed:





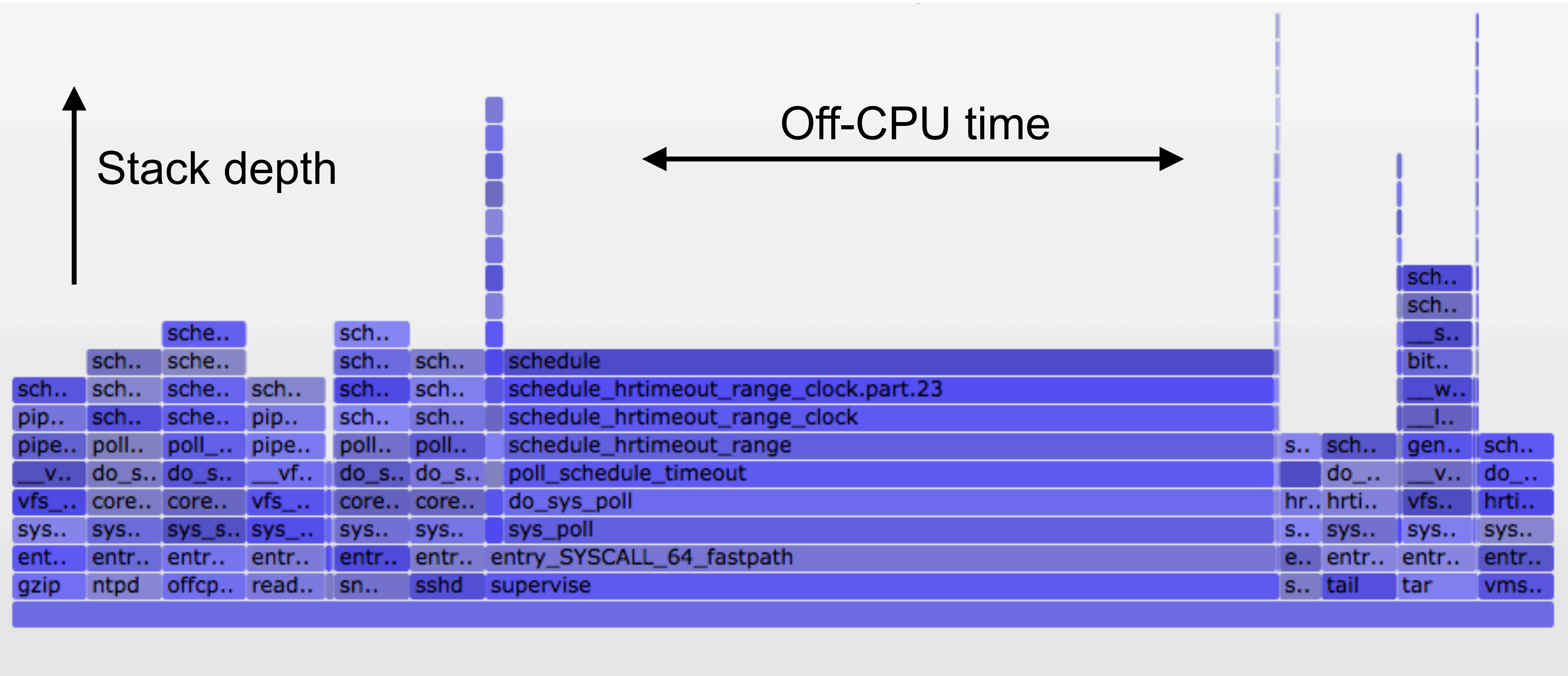
Linux Off-CPU Analysis



On Linux, the state isn't helpful, but the **code path** is

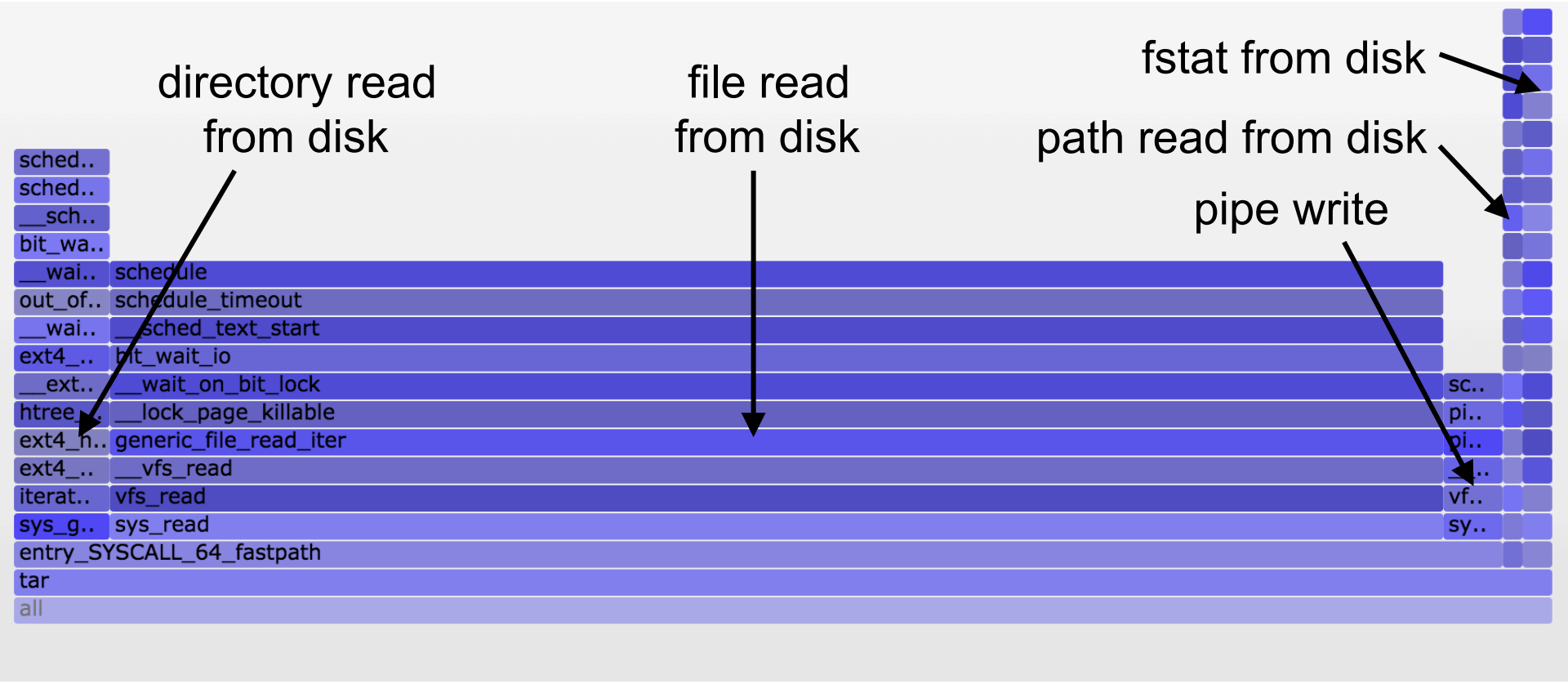
Off-CPU analysis by measuring blocked time with **stack traces**

Off-CPU Time Flame Graph



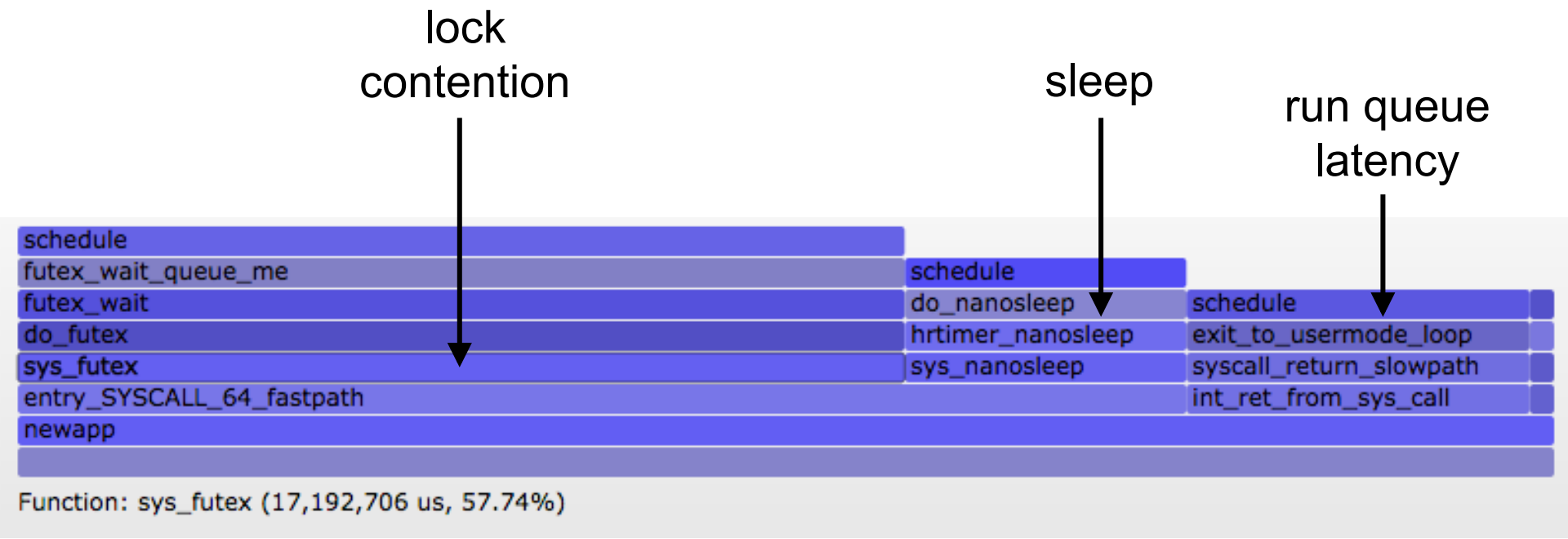
From http://www.brendangregg.com/blog/2016-02-01/linux-wakeup-offwake-profiling.html

Off-CPU Time (zoomed): tar(1)



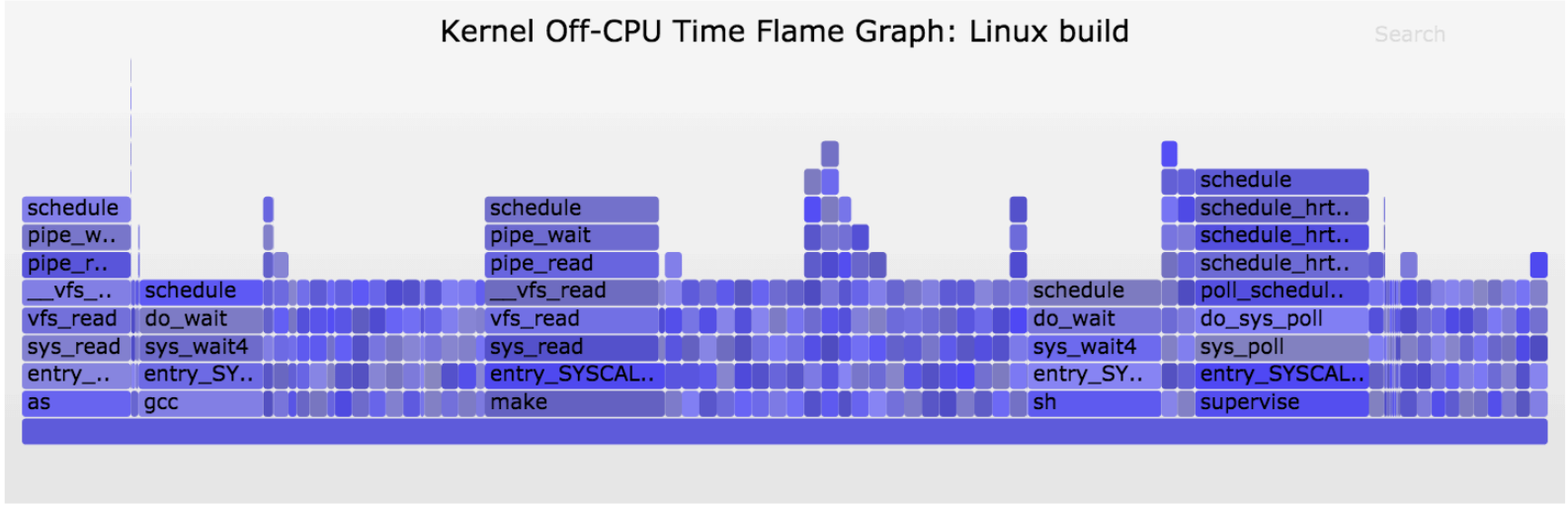
Currently kernel stacks only; user stacks will add more context

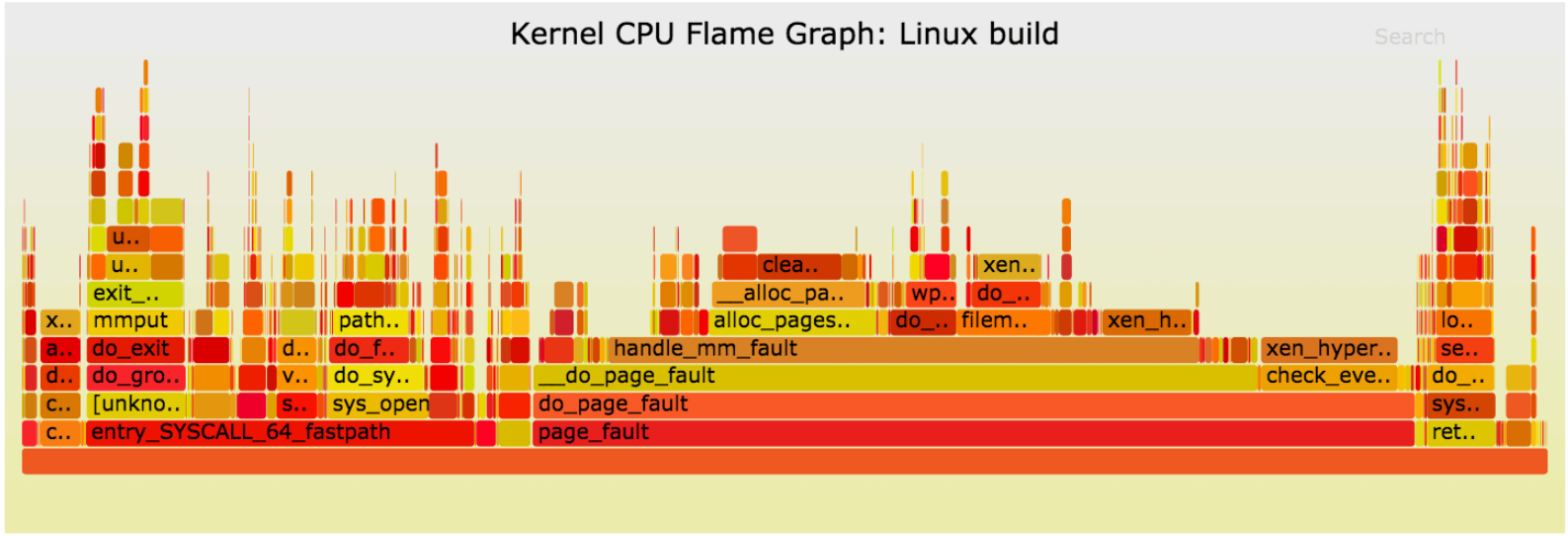
Off-CPU Time: more states



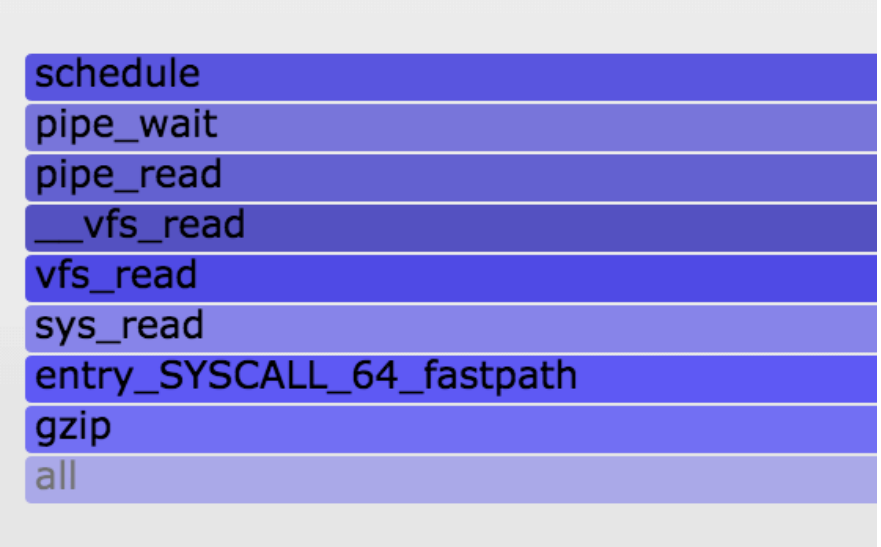
Flame graph quantifies total time spent in states

CPU + Off-CPU == See Everything?





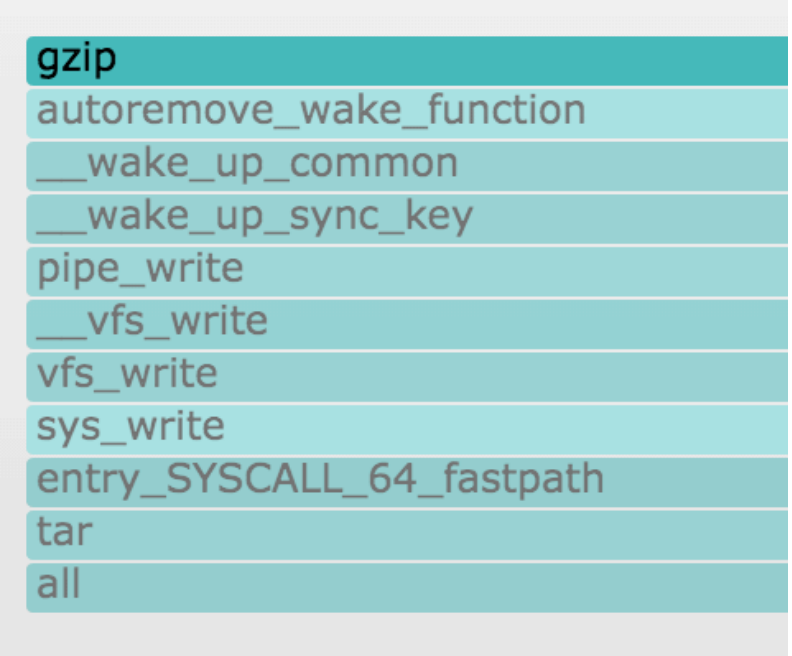
Off-CPU Time (zoomed): gzip(1)



Off-CPU doesn't always make sense:
what is gzip blocked on?

Wakeup Time Flame Graph

Wakeup Time (zoomed): gzip(1)

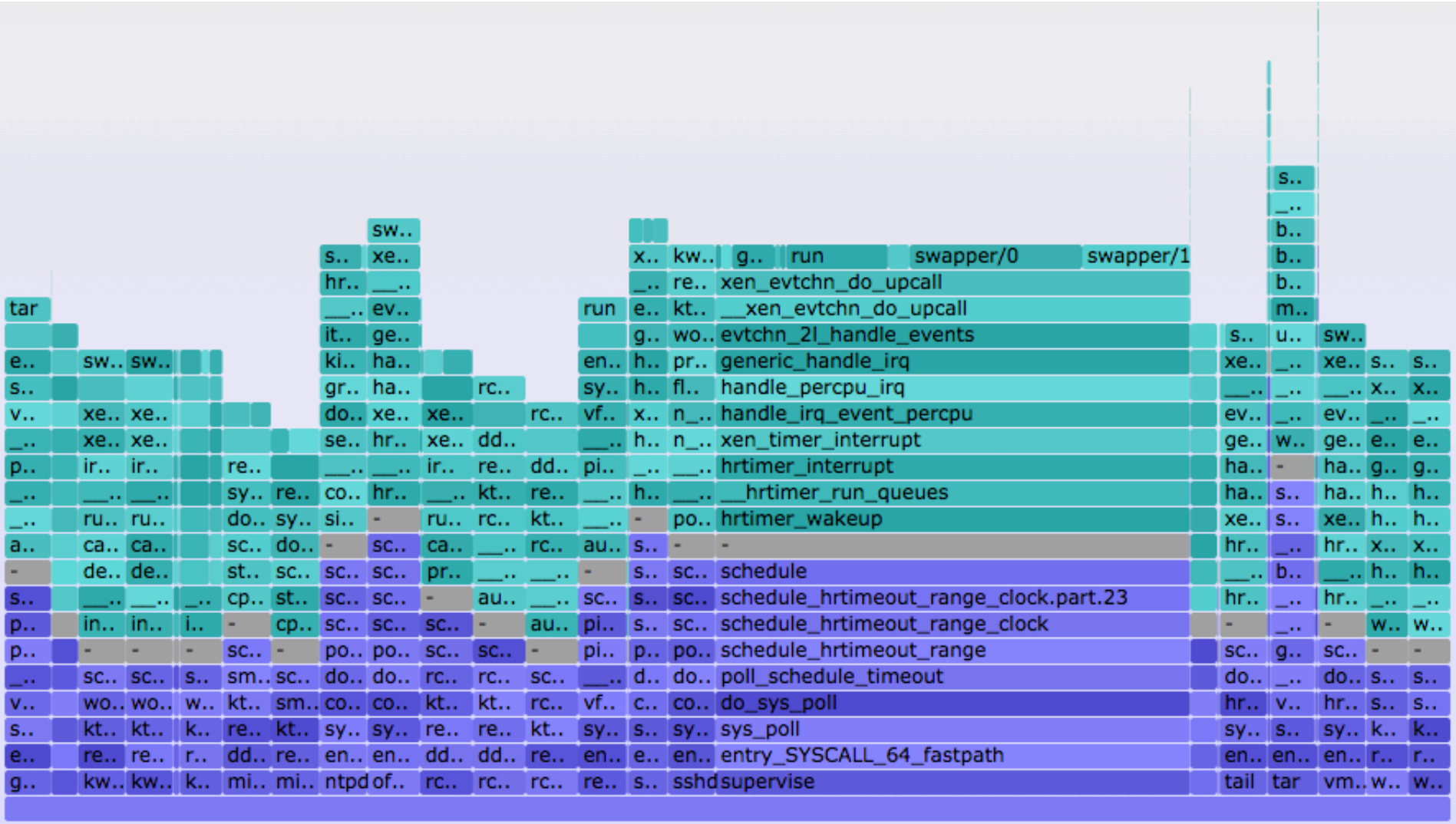


gzip(1) is blocked on tar(1)!

```
tar cf - * | gzip > out.tar.gz
```

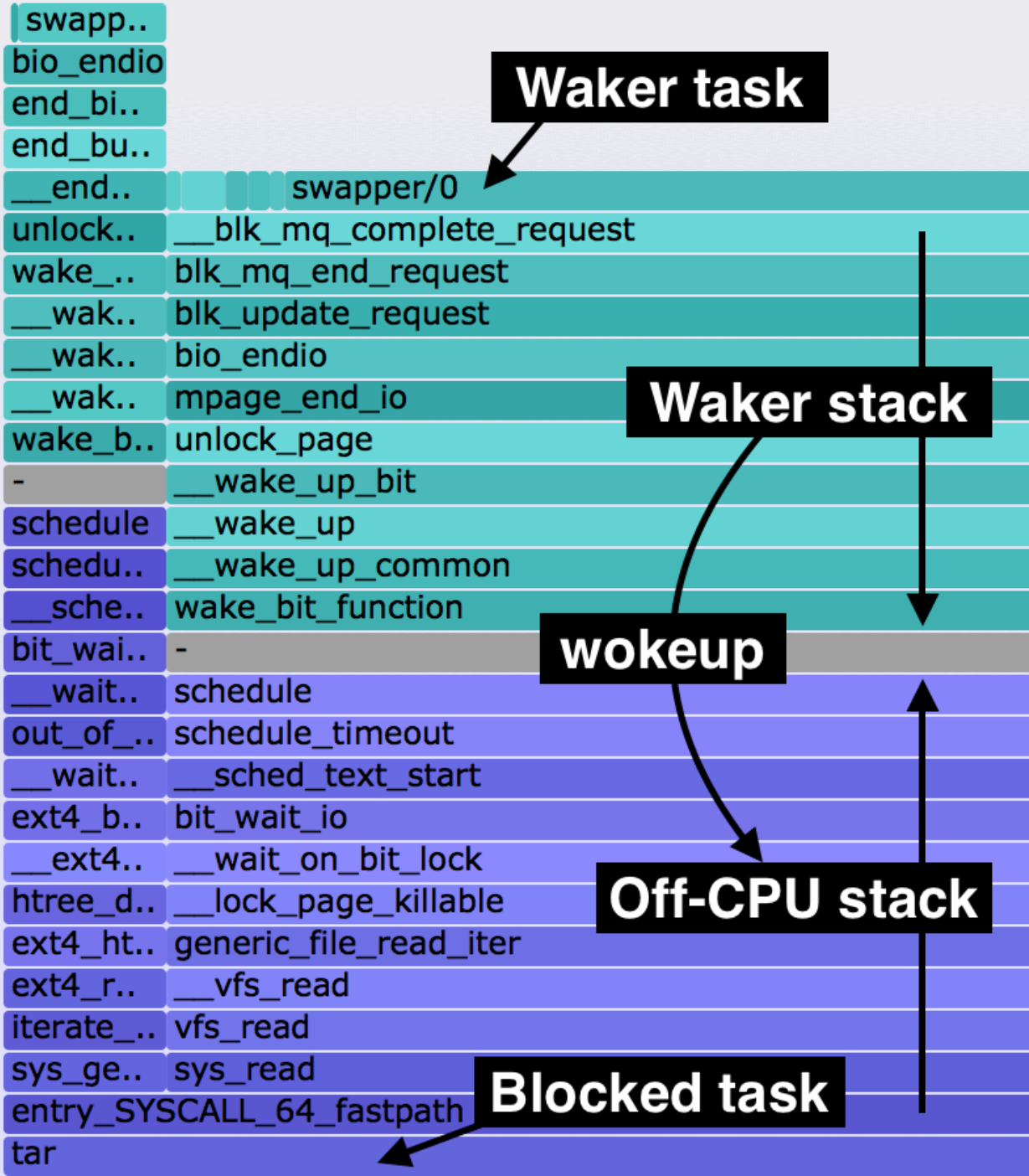
Can't we associate off-CPU with wakeup stacks?

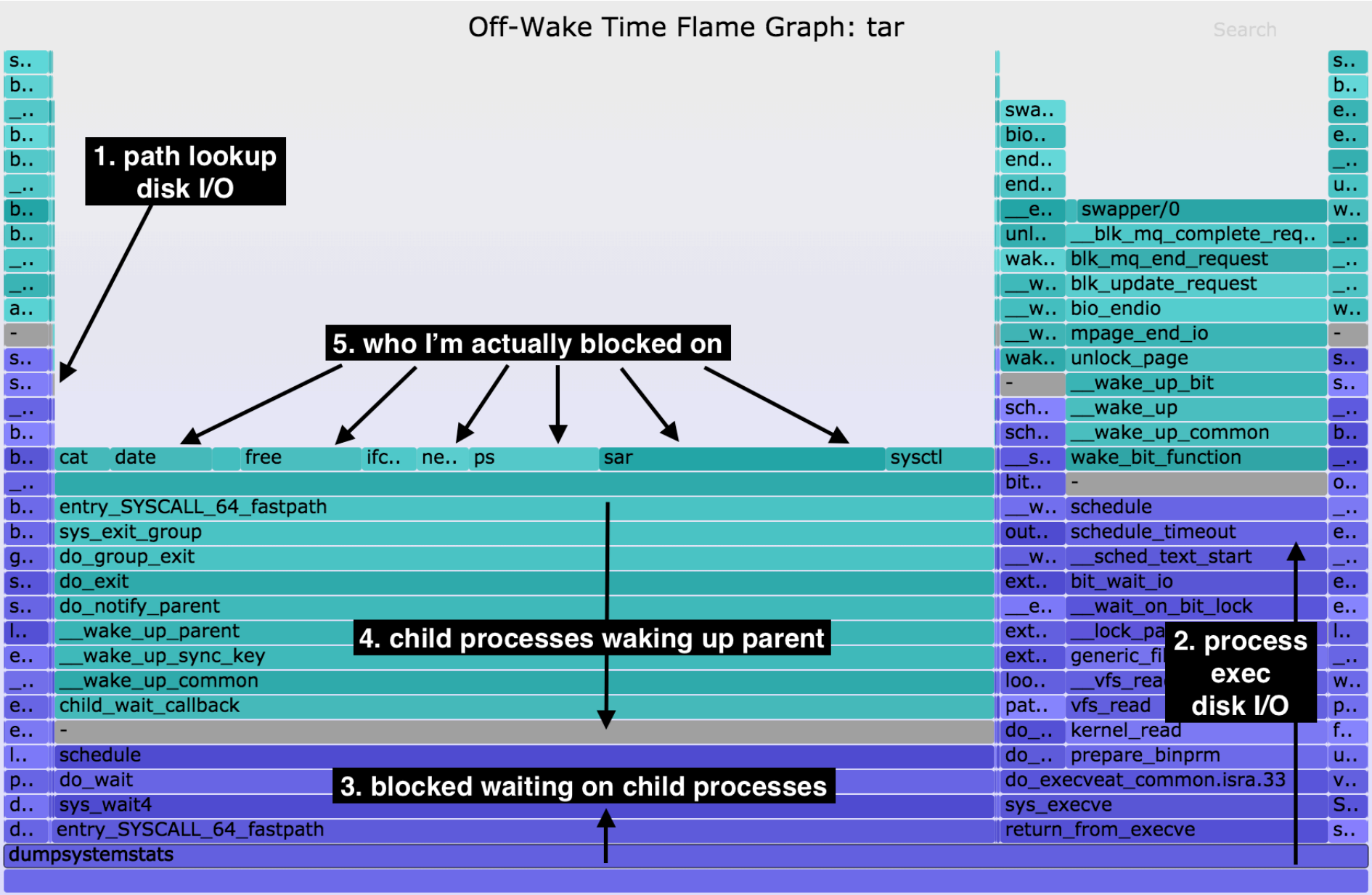
Off-Wake Time Flame Graph



Wakeup stacks are associated and merged in-kernel using BPF

We couldn't do this before

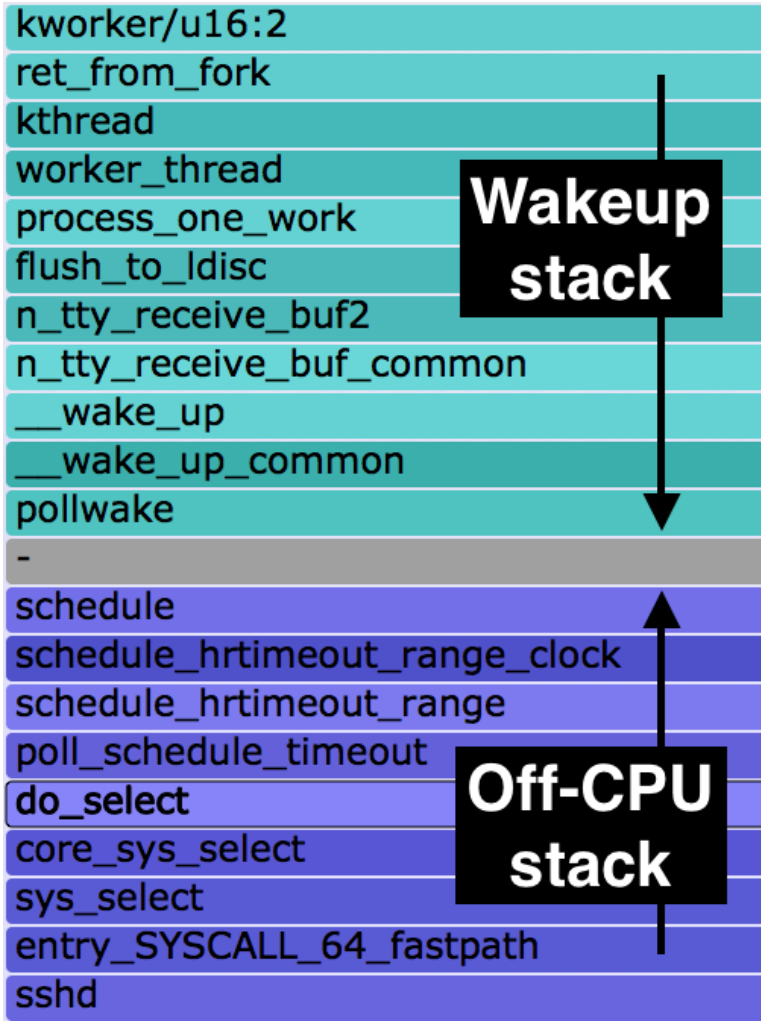




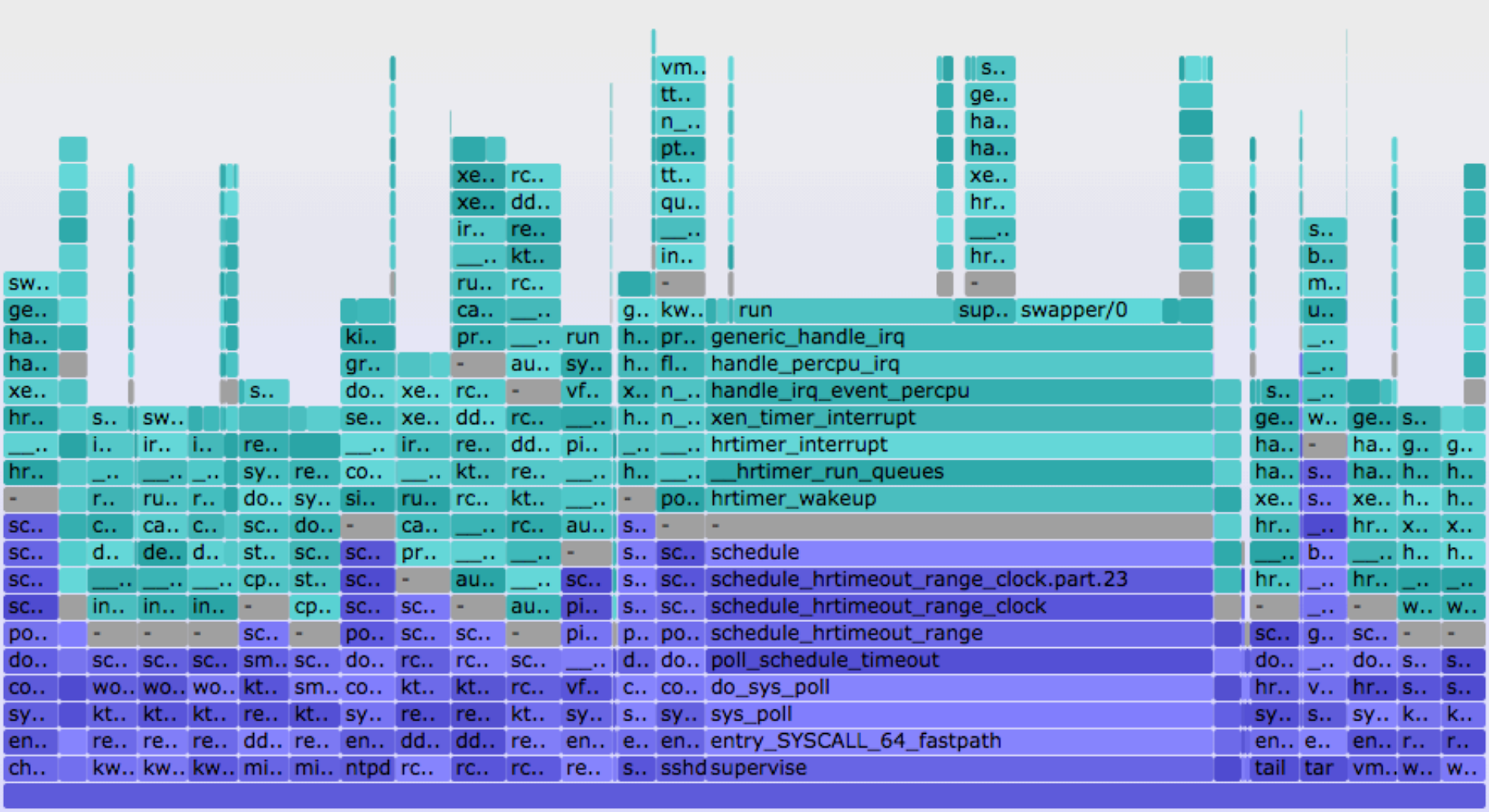
Function: dumpsystemstats (386,247 us, 100.00%)

Haven't Solved Everything Yet…

- One wakeup stack is often not enough…
- Who woke the waker?

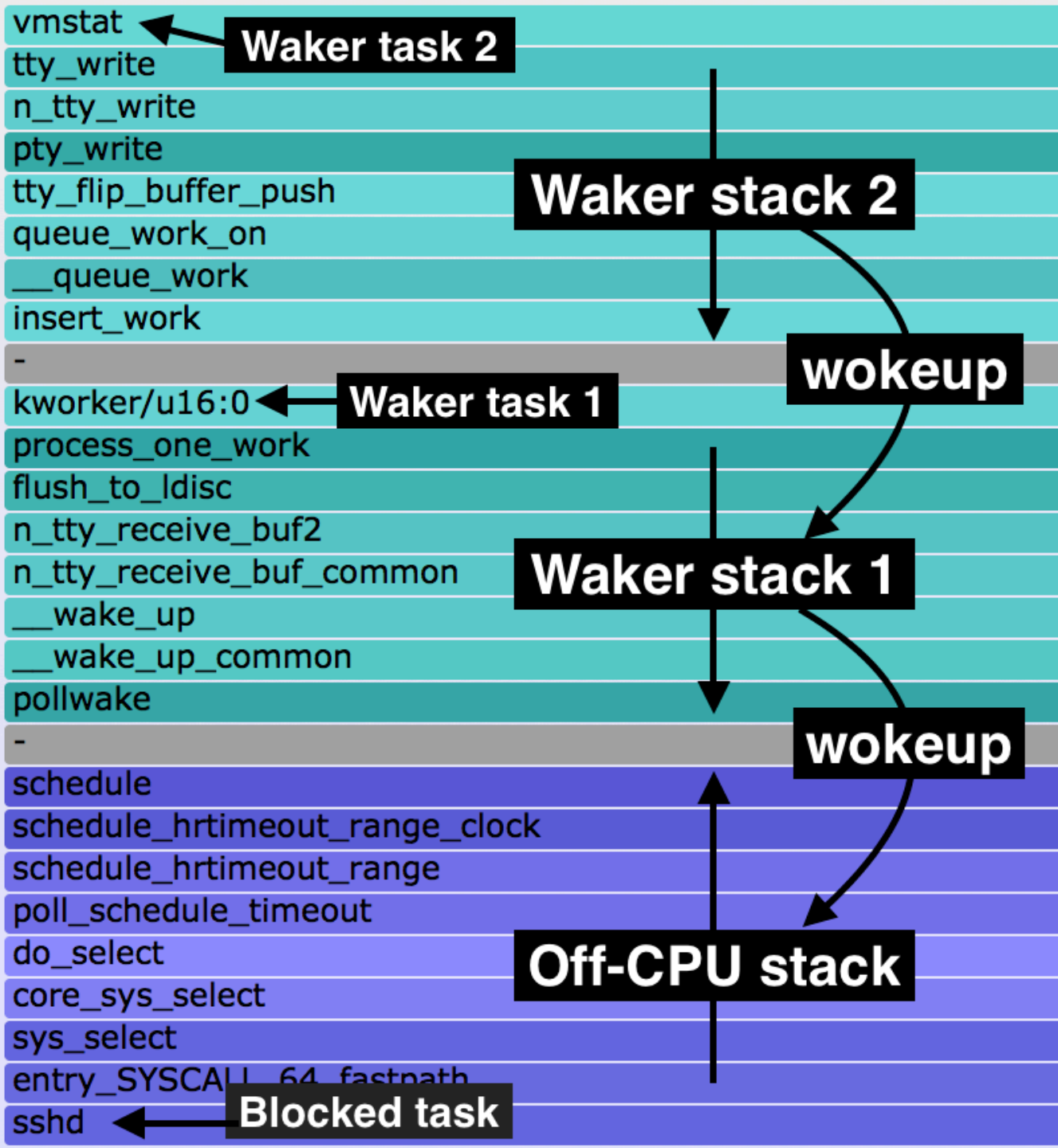


Chain Graphs



Merging multiple wakeup stacks in kernel using BPF

With enough stacks, all paths lead to metal

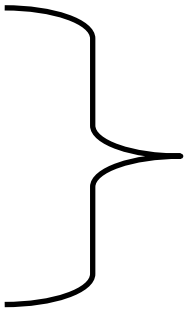


Solve Everything

CPU + off-CPU analysis can solve most issues

Flame graph (profiling) types:

1. CPU
2. CPI
3. Off-CPU time
4. Wakeup time
5. Off-wake time
6. Chain



different off-CPU analysis views, with more context and increasing measurement cost

BPF makes this all more practical

2. BPF

"One of the more interesting features in this cycle is the ability to attach eBPF programs (**user-defined, sandboxed bytecode executed by the kernel**) to kprobes. This allows user-defined instrumentation on a live kernel image that can never crash, hang or interfere with the kernel negatively."

– Ingo Molnár (Linux developer)

Source: https://lkml.org/lkml/2015/4/14/232

2. BPF

"crazy stuff"

– Alexei Starovoitov (eBPF lead)

Source: http://www.slideshare.net/AlexeiStarovoitov/bpf-inkernel-virtual-machine

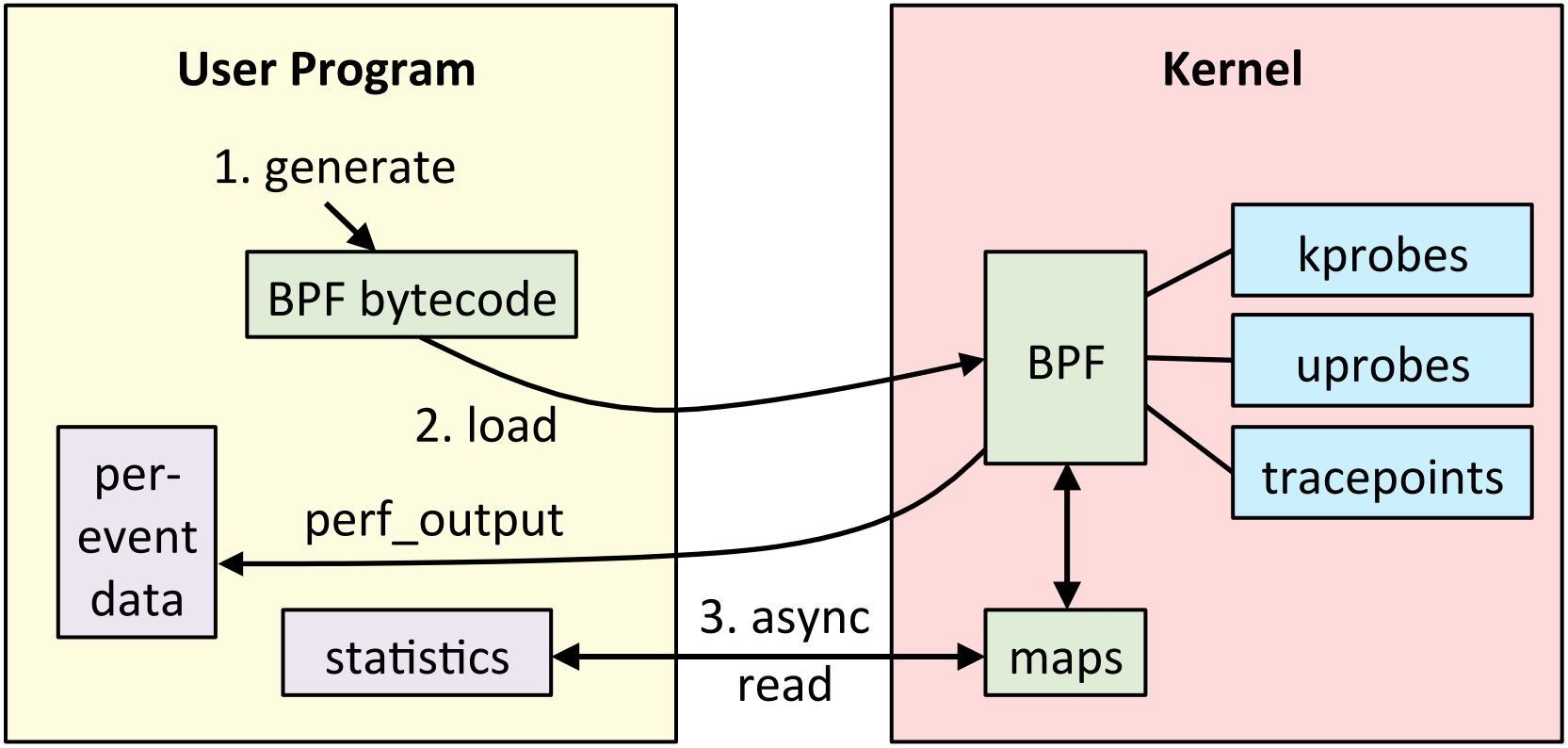
BPF

- eBPF == enhanced Berkeley Packet Filter; now just BPF
- Integrated into Linux (in stages: 3.15, 3.19, 4.1, 4.5, …)
- Uses
  - virtual networking
  - tracing
  - "crazy stuff"
- Front-ends
  - samples/bpf (raw)
  - bcc: Python, C
  - Linux perf_events

BPF mascot

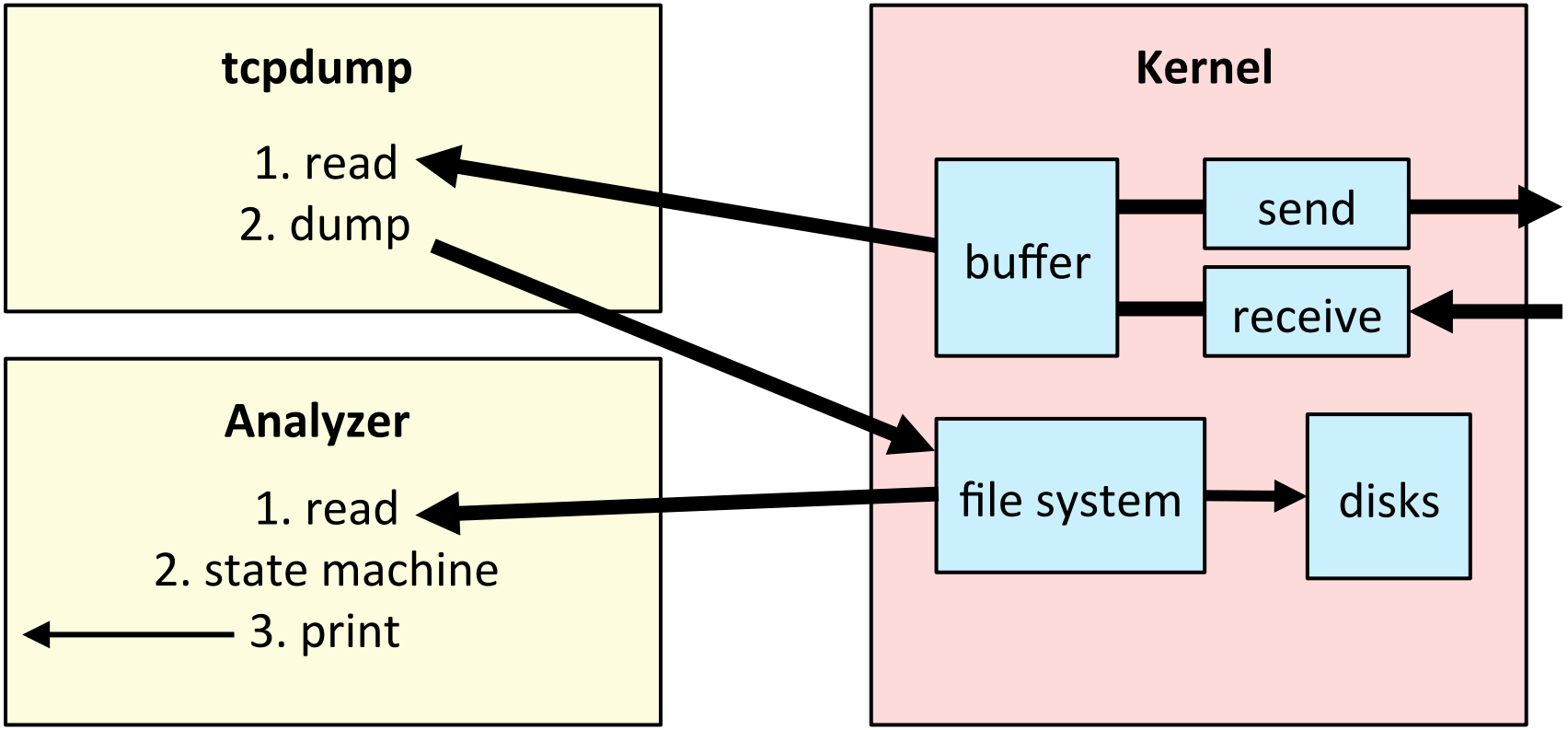
BPF for Tracing

- Can do per-event output and in-kernel summary statistics (histograms, etc).



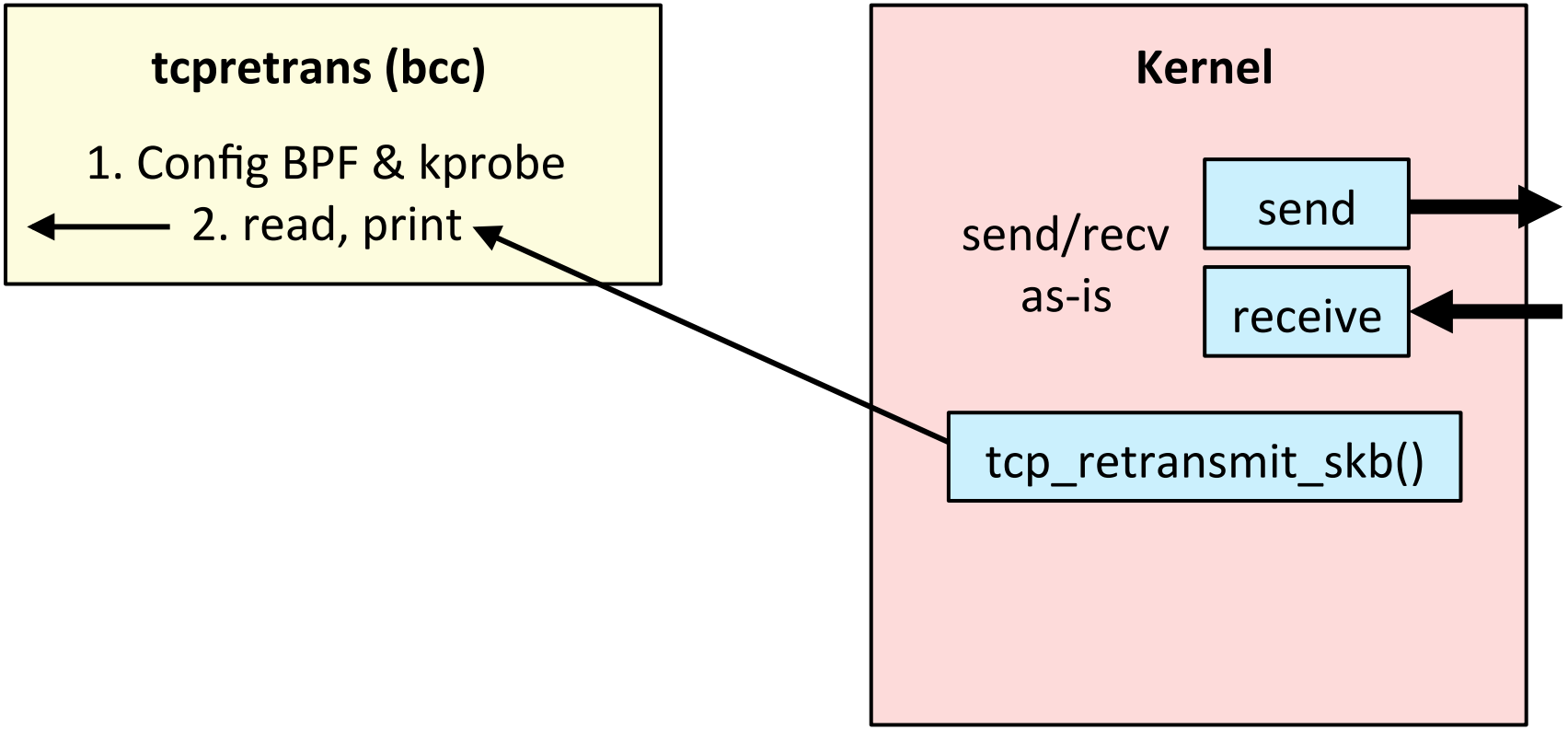
Old way: TCP Retransmits

- tcpdump of all send & receive, dump to FS, post-process
- Overheads adds up on 10GbE+



New way: BPF TCP Retransmits

- Just trace the retransmit functions
- Negligible overhead



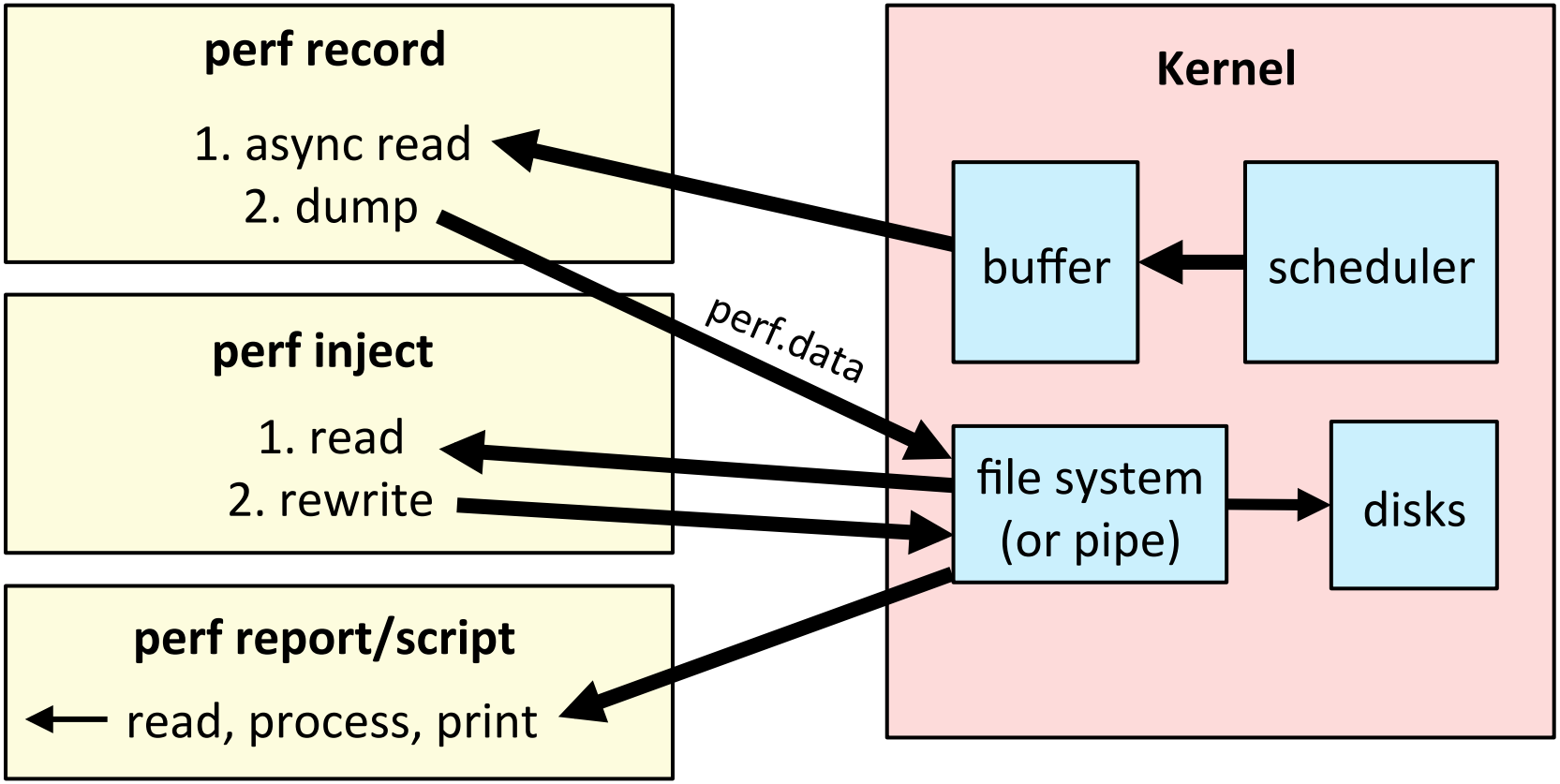
BPF: TCP Retransmits

```
# ./tcpretrans
TIME     PID    IP LADDR:LPORT          T> RADDR:RPORT          STATE
01:55:05 0      4  10.153.223.157:22    R> 69.53.245.40:34619   ESTABLISHED
01:55:05 0      4  10.153.223.157:22    R> 69.53.245.40:34619   ESTABLISHED
01:55:17 0      4  10.153.223.157:22    R> 69.53.245.40:22957   ESTABLISHED
[…]
```

includes kernel state

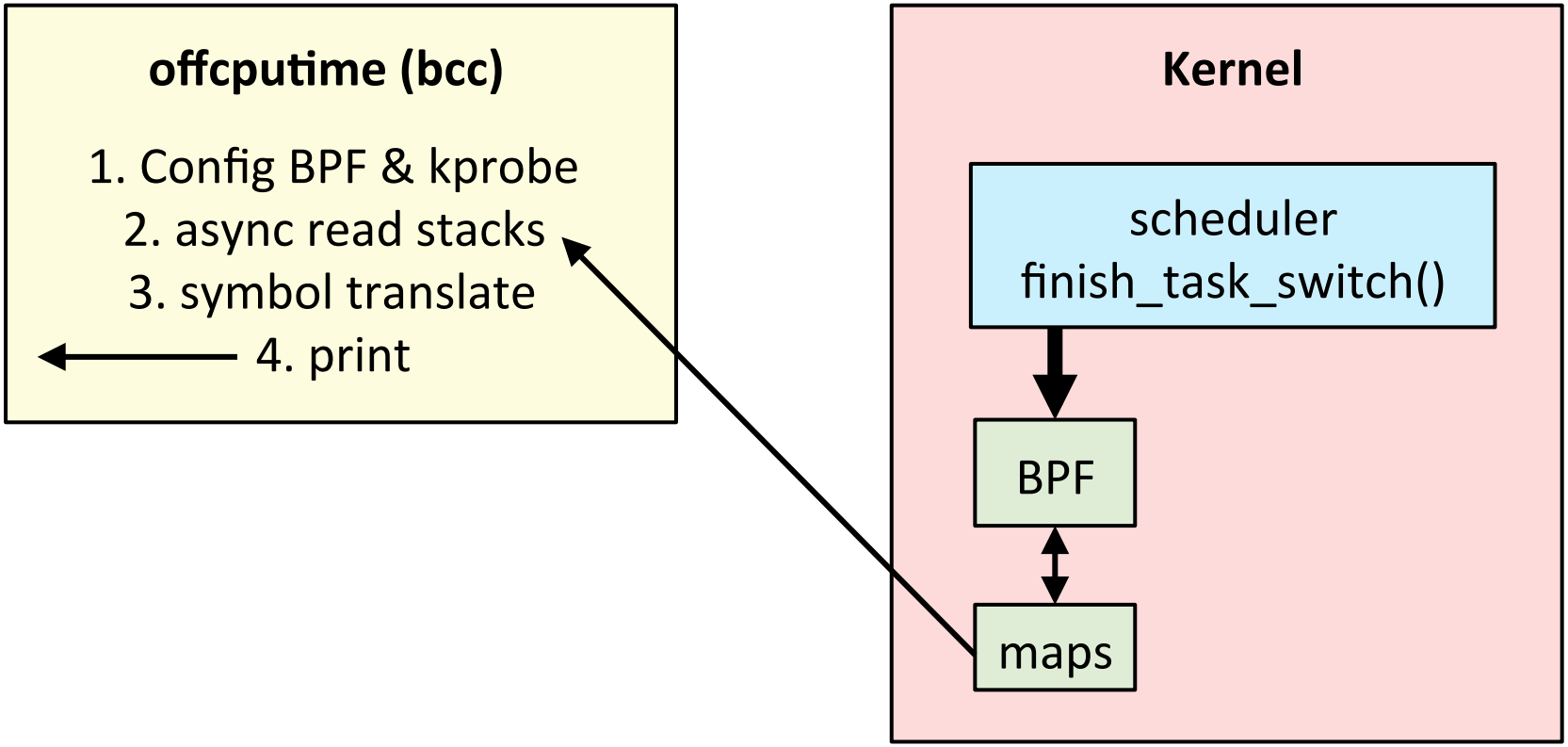
Old: Off-CPU Time Stack Profiling

- perf_events tracing of sched events, post-process
- Despite buffering, usually high cost (>1M events/sec)



New: BPF Off-CPU Time Stacks

- Measure off-CPU time, add to map with key = stack, value = total time. Async read map.



Stack Trace Hack

- For my offcputime tool, I wrote a BPF stack walker:

```
 8  static u64 get_frame(u64 *bp) {
 9      if (*bp) {
10          // The following stack walker is x86_64 specific
11          u64 ret = 0;
12          if (bpf_probe_read(&ret, sizeof(ret), (void *)(*bp+8)))
13              return 0;
14          if (bpf_probe_read(bp, sizeof(*bp), (void *)*bp))
15              *bp = 0;
16          if (ret < __START_KERNEL_map)
17              return 0;
18          return ret;
19      }
20      return 0;
21  }
```

"Crazy Stuff"

- … using unrolled loops & goto:

```
28      bp = ctx->bp;
29      // unrolled loop, 10 (MAXDEPTH) frames deep:
30      if (!(key.ret[depth++] = get_frame(&bp))) goto out;
31      if (!(key.ret[depth++] = get_frame(&bp))) goto out;
32      if (!(key.ret[depth++] = get_frame(&bp))) goto out;
33      if (!(key.ret[depth++] = get_frame(&bp))) goto out;
34      if (!(key.ret[depth++] = get_frame(&bp))) goto out;
35      if (!(key.ret[depth++] = get_frame(&bp))) goto out;
36      if (!(key.ret[depth++] = get_frame(&bp))) goto out;
37      if (!(key.ret[depth++] = get_frame(&bp))) goto out;
38      if (!(key.ret[depth++] = get_frame(&bp))) goto out;
39      if (!(key.ret[depth++] = get_frame(&bp))) goto out;
40  out:
```

BPF Stack Traces

- Proper BPF stack support just landed in net-next:

```
Date    Sat, 20 Feb 2016 00:25:05 -0500 (EST)
Subject  Re: [PATCH net-next 0/3] bpf_get_stackid() and stack_trace map
From    David Miller <>

From: Alexei Starovoitov <ast@fb.com>
Date: Wed, 17 Feb 2016 19:58:56 -0800

> This patch set introduces new map type to store stack traces and
> corresponding bpf_get_stackid() helper.
 ...

Series applied, thanks Alexei.
```

- Allows more than just chain graphs

memleak

- Real-time memory growth and leak analysis:

```
# ./memleak.py -o 10 60 1
Attaching to kmalloc and kfree, Ctrl+C to quit.
[01:27:34] Top 10 stacks with outstanding allocations:
    72 bytes in 1 allocations from stack
        alloc_fdtable [kernel] (ffffffff8121960f)
        expand_files [kernel] (ffffffff8121986b)
        sys_dup2 [kernel] (ffffffff8121a68d)
[…]
    2048 bytes in 1 allocations from stack
        alloc_fdtable [kernel] (ffffffff812195da)
        expand_files [kernel] (ffffffff8121986b)
        sys_dup2 [kernel] (ffffffff8121a68d) ]
```

Trace for 60s
Show kernel allocations older than 10s that were not freed

- Uses my stack hack, but will switch to BPF stacks soon
- By Sasha Goldshtein. Another bcc tool.

3. bcc

- BPF Compiler Collection
  - https://github.com/iovisor/bcc
- Python front-end, C instrumentation
- Currently beta – in development!
- Some example tracing tools…

execsnoop

- Trace new processes:

```
# ./execsnoop
PCOMM       PID    RET ARGS
bash        15887    0 /usr/bin/man ls
preconv     15894    0 /usr/bin/preconv -e UTF-8
man         15896    0 /usr/bin/tbl
man         15897    0 /usr/bin/nroff -mandoc -rLL=169n -rLT=169n -Tutf8
man         15898    0 /usr/bin/pager -s
nroff       15900    0 /usr/bin/locale charmap
nroff       15901    0 /usr/bin/groff -mtty-char -Tutf8 -mandoc -rLL=169n …
groff       15902    0 /usr/bin/troff -mtty-char -mandoc -rLL=169n -rLT=169 …
groff       15903    0 /usr/bin/grotty
```

biolatency

- Block device (disk) I/O latency distribution:

```
# ./biolatency -mT 1 5
Tracing block device I/O... Hit Ctrl-C to end.

06:20:16
     msecs               : count     distribution
       0 -> 1            : 36       |****************************************|
       2 -> 3            : 1        |*                                       |
       4 -> 7            : 3        |***                                     |
       8 -> 15           : 17       |*****************                       |
      16 -> 31           : 33       |********************************        |
      32 -> 63           : 7        |*******                                 |
      64 -> 127          : 6        |******                                  |
[…]
```

ext4slower

- ext4 file system I/O, slower than a threshold:

```
# ./ext4slower 1
Tracing ext4 operations slower than 1 ms
TIME     COMM           PID    T BYTES   OFF_KB   LAT(ms) FILENAME
06:49:17 bash           3616   R 128     0           7.75 cksum
06:49:17 cksum          3616   R 39552   0           1.34 [
06:49:17 cksum          3616   R 96      0           5.36 2to3-2.7
06:49:17 cksum          3616   R 96      0          14.94 2to3-3.4
06:49:17 cksum          3616   R 10320   0           6.82 411toppm
06:49:17 cksum          3616   R 65536   0           4.01 a2p
06:49:17 cksum          3616   R 55400   0           8.77 ab
06:49:17 cksum          3616   R 36792   0          16.34 aclocal-1.14
06:49:17 cksum          3616   R 15008   0          19.31 acpi_listen
06:49:17 cksum          3616   R 6123    0          17.23 add-apt-
repository
06:49:17 cksum          3616   R 6280    0          18.40 addpart
06:49:17 cksum          3616   R 27696   0           2.16 addr2line
06:49:17 cksum          3616   R 58080   0          10.11 ag
06:49:17 cksum          3616   R 906     0           6.30 ec2-meta-data
[…]
```

bashreadline

- Trace bash interactive commands system-wide:

```
# ./bashreadline
TIME      PID    COMMAND
05:28:25  21176  ls -l
05:28:28  21176  date
05:28:35  21176  echo hello world
05:28:43  21176  foo this command failed
05:28:45  21176  df -h
05:29:04  3059   echo another shell
05:29:13  21176  echo first shell again
```

gethostlatency

- Show latency for getaddrinfo/gethostbyname[2] calls:

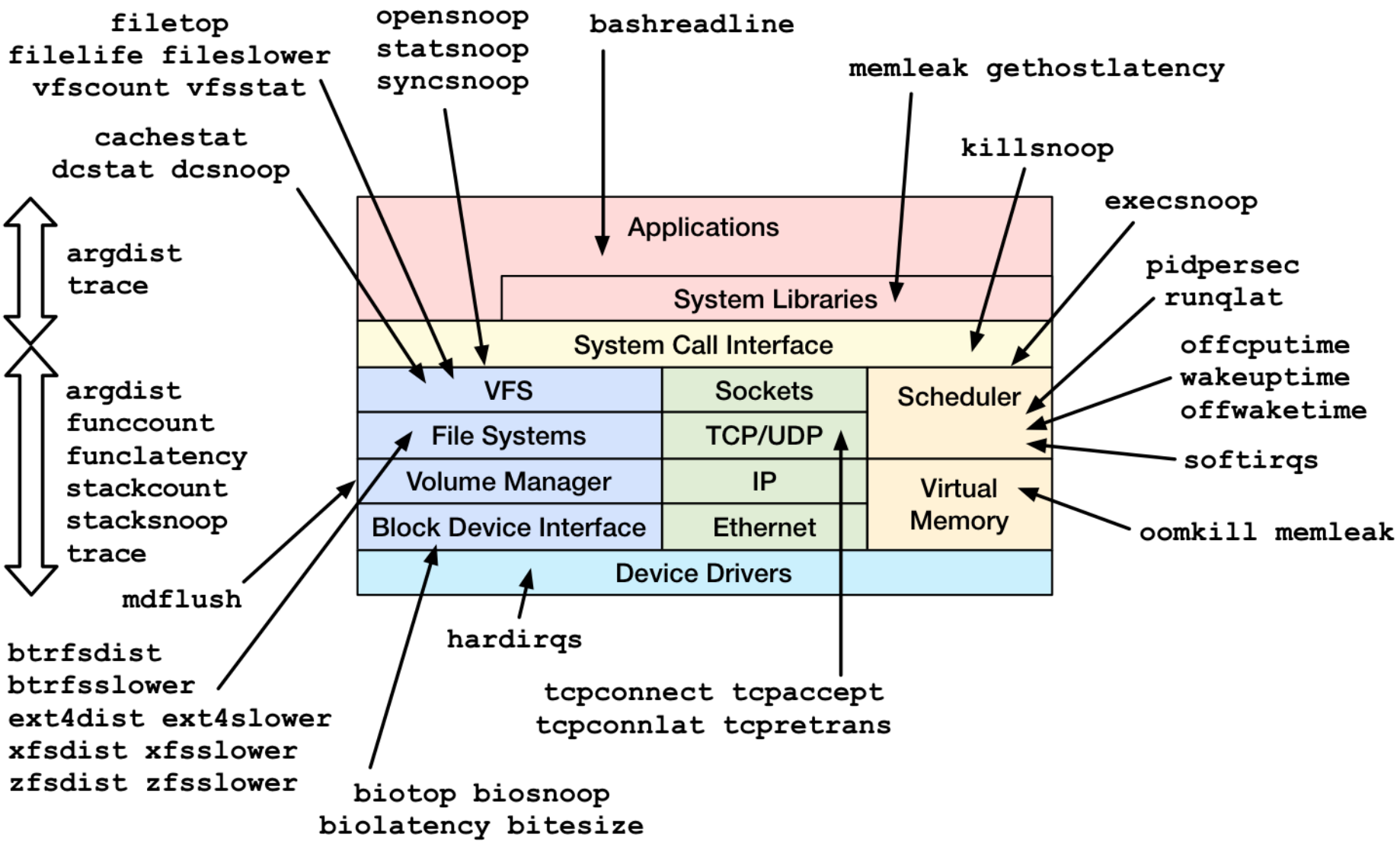
```
# ./gethostlatency
TIME      PID    COMM            LATms HOST
06:10:24  28011  wget            90.00 www.iovisor.org
06:10:28  28127  wget             0.00 www.iovisor.org
06:10:41  28404  wget             9.00 www.netflix.com
06:10:48  28544  curl            35.00 www.netflix.com.au
06:11:10  29054  curl            31.00 www.plumgrid.com
06:11:16  29195  curl             3.00 www.facebook.com
06:11:25  29404  curl            72.00 foo
06:11:28  29475  curl             1.00 foo
```

trace

- Trace custom events. Ad hoc analysis multitool:

```
# trace 'sys_read (arg3 > 20000) "read %d bytes", arg3'
TIME     PID    COMM         FUNC               -
05:18:23 4490   dd           sys_read           read 1048576 bytes
05:18:23 4490   dd           sys_read           read 1048576 bytes
05:18:23 4490   dd           sys_read           read 1048576 bytes
05:18:23 4490   dd           sys_read           read 1048576 bytes
^C
```

Linux bcc/BPF Tracing Tools

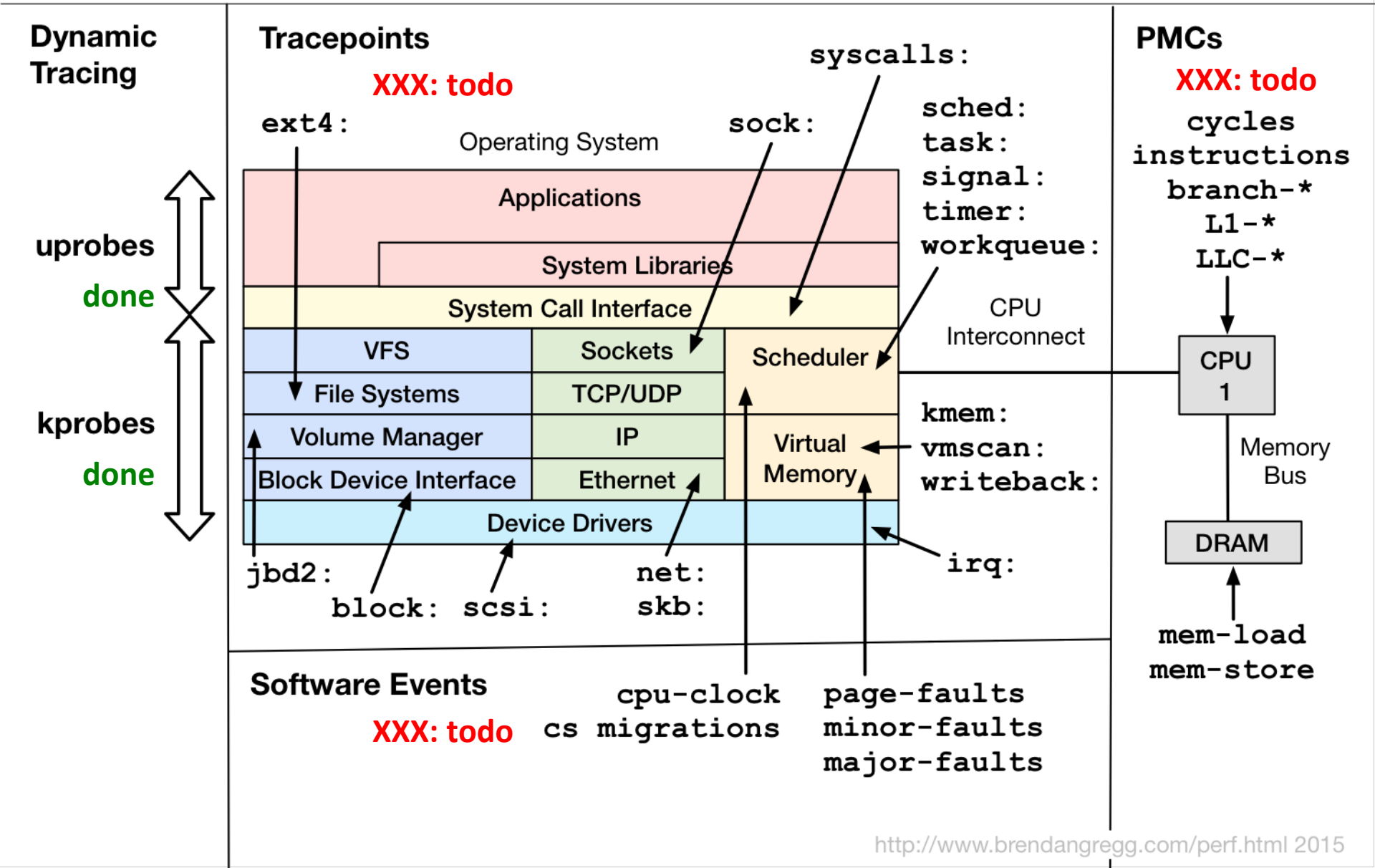


https://github.com/iovisor/bcc#tracing 2016

4. Future Work

- All event sources
- Language improvements
- More tools: eg, TCP
- GUI support

Linux Event Sources



BPF/bcc Language Improvements

```
static int trace_event(struct pt_regs *ctx, struct sock *sk, int type)
{
    if (sk == NULL)
        return 0;
    u32 pid = bpf_get_current_pid_tgid();
    struct sock *skp = NULL;
    bpf_probe_read(&skp, sizeof(skp), &sk);
    // pull in details
    u16 family = 0, lport = 0, dport = 0;
    char state = 0;
    bpf_probe_read(&family, sizeof(family), &skp->__sk_common.skc_family);
    bpf_probe_read(&lport, sizeof(lport), &skp->__sk_common.skc_num);
    bpf_probe_read(&dport, sizeof(dport), &skp->__sk_common.skc_dport);
    bpf_probe_read(&state, sizeof(state), (void *)&skp->__sk_common.skc_state);
    if (family == AF_INET) {
        struct ipv4_data_t data4 = {.pid = pid, .ip = 4, .type = type};
        bpf_probe_read(&data4.saddr, sizeof(u32),
            &skp->__sk_common.skc_rcv_saddr);
        bpf_probe_read(&data4.daddr, sizeof(u32),
            &skp->__sk_common.skc_daddr);
        data4.lport = lport;
        data4.dport = dport;
        data4.state = state;
        ipv4_events.perf_submit(ctx, &data4, sizeof(data4));
```

More Tools

- eg, netstat(8)…

```
$ netstat -s
Ip:
    7962754 total packets received
    8 with invalid addresses
    0 forwarded
    0 incoming packets discarded
    7962746 incoming packets delivered
    8019427 requests sent out
Icmp:
    382 ICMP messages received
    0 input ICMP message failed.
    ICMP input histogram:
        destination unreachable: 125
        timeout in transit: 257
    3410 ICMP messages sent
    0 ICMP messages failed
    ICMP output histogram:
        destination unreachable: 3410
IcmpMsg:
        InType3: 125
        InType11: 257
        OutType3: 3410
Tcp:
    17337 active connections openings
    395515 passive connection openings
    8953 failed connection attempts
    240214 connection resets received
    3 connections established
    7198375 segments received
    7504939 segments send out
    62696 segments retransmited
    10 bad segments received.
 1072 resets sent
    InCsumErrors: 5
Udp:
    759925 packets received
    3412 packets to unknown port received.
    0 packet receive errors
    784370 packets sent
UdpLite:
TcpExt:
    858 invalid SYN cookies received
    8951 resets received for embryonic SYN_RECV sockets
    14 packets pruned from receive queue because of socket buffer overrun
    6177 TCP sockets finished time wait in fast timer
    293 packets rejects in established connections because of timestamp
    733028 delayed acks sent
    89 delayed acks further delayed because of locked socket
    Quick ack mode was activated 13214 times
    336520 packets directly queued to recvmsg prequeue.
    43964 packets directly received from backlog
    11406012 packets directly received from prequeue
    1039165 packets header predicted
    7066 packets header predicted and directly queued to user
    1428960 acknowledgments not containing data received
    1004791 predicted acknowledgments
    1 times recovered from packet loss due to fast retransmit
    5044 times recovered from packet loss due to SACK data
    2 bad SACKs received
    Detected reordering 4 times using SACK
    Detected reordering 11 times using time stamp
    13 congestion windows fully recovered
    11 congestion windows partially recovered using Hoe heuristic
    TCPDSACKUndo: 39
    2384 congestion windows recovered after partial ack
    228 timeouts after SACK recovery
    100 timeouts in loss state
    5018 fast retransmits
    39 forward retransmits
    783 retransmits in slow start
    32455 other TCP timeouts
    TCPLossProbes: 30233
    TCPLossProbeRecovery: 19070
    992 sack retransmits failed
    18 times receiver scheduled too late for direct processing
    705 packets collapsed in receive queue due to low socket buffer
    13658 DSACKs sent for old packets
    8 DSACKs sent for out of order packets
    13595 DSACKs received
    33 DSACKs for out of order packets received
    32 connections reset due to unexpected data
    108 connections reset due to early user close
    1608 connections aborted due to timeout
    TCPSACKDiscard: 4
    TCPDSACKIgnoredOld: 1
    TCPDSACKIgnoredNoUndo: 8649
    TCPSpuriousRTOs: 445
    TCPSackShiftFallback: 8588
    TCPRcvCoalesce: 95854
    TCPOFOQueue: 24741
    TCPOFOMerge: 8
    TCPChallengeACK: 1441
    TCPSYNChallenge: 5
    TCPSpuriousRtxHostQueues: 1
    TCPAutoCorking: 4823
IpExt:
    InOctets: 1561561375
    OutOctets: 1509416943
    InNoECTPkts: 8201572
    InECT1Pkts: 2
    InECT0Pkts: 3844
    InCEPkts: 306
```

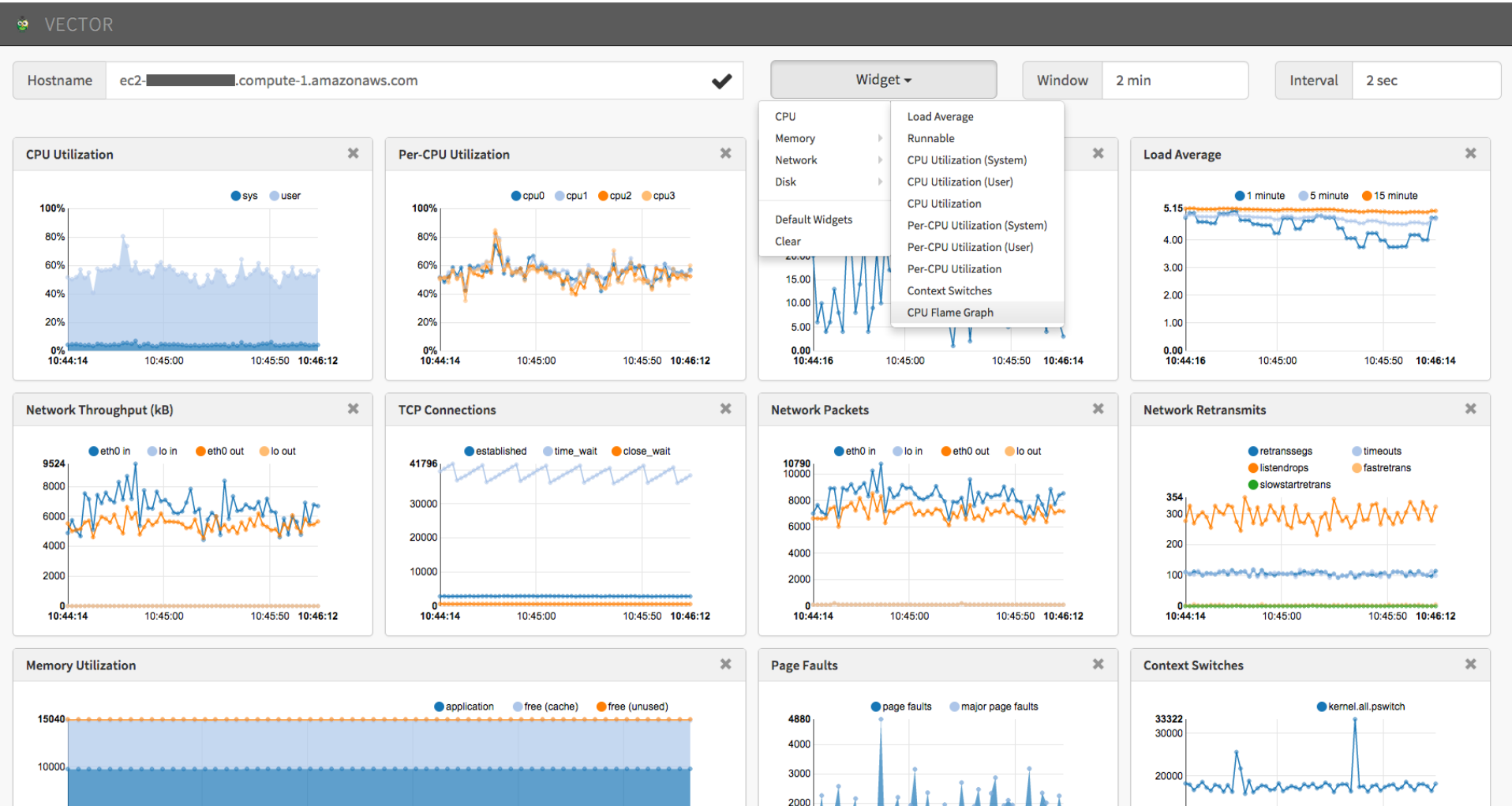

packets
received
ICMP
connections
sent
DSACKs
retransmits
directly
times
recovered
segments
connection
messages
failed
acknowledgments
destination
sack
TCP
windows
reordering
histogram
predicted
timeout
receive
input
invalid
order
due
data
congestion
unreachable
packet
resets
established
Detected
delayed
queued
socket
sockets
buffer
user
queue
timeouts
openings
header
fast
time
loss
prequeue
bad
acks
using
reset
ack
3410
257
125
39
11
8
5
4
2
1
0

Better TCP Tools

- TCP retransmit by type and time
- Congestion algorithm metrics
- etc.

GUI Support

- eg, Netflix Vector: open source instance analyzer:



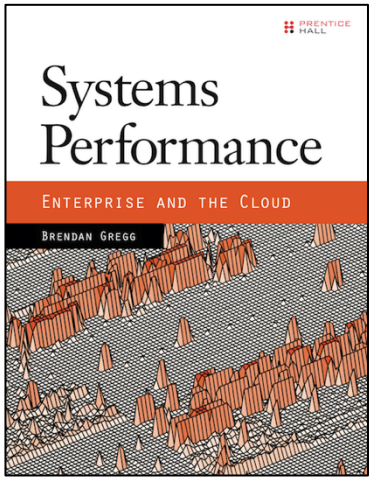
Summary

- BPF in Linux 4.x makes many new things possible
  - Stack-based thread state analysis (solve all issues!)
  - Real-time memory growth/leak detection
  - Better TCP metrics
  - etc...
- Get involved: see iovisor/bcc
- So far just a preview of things to come

Links

- iovisor bcc:
  - https://github.com/iovisor/bcc
  - http://www.brendangregg.com/blog/2015-09-22/bcc-linux-4.3-tracing.html
  - http://blogs.microsoft.co.il/sasha/2016/02/14/two-new-ebpf-tools-memleak-and-argdist/
- BPF Off-CPU, Wakeup, Off-Wake & Chain Graphs:
  - http://www.brendangregg.com/blog/2016-01-20/ebpf-offcpu-flame-graph.html
  - http://www.brendangregg.com/blog/2016-02-01/linux-wakeup-offwake-profiling.html
  - http://www.brendangregg.com/blog/2016-02-05/ebpf-chaingraph-prototype.html
- Linux Performance:
  - http://www.brendangregg.com/linuxperf.html
- Linux perf_events:
  - https://perf.wiki.kernel.org/index.php/Main_Page
  - http://www.brendangregg.com/perf.html
- Flame Graphs:
  - http://techblog.netflix.com/2015/07/java-in-flames.html
  - http://www.brendangregg.com/flamegraphs.html
- Netflix Tech Blog on Vector:
  - http://techblog.netflix.com/2015/04/introducing-vector-netflixs-on-host.html
- Wordcloud: https://www.jasondavies.com/wordcloud/

- Questions?
- http://slideshare.net/brendangregg
- http://www.brendangregg.com
- bgregg@netflix.com
- @brendangregg



Thanks to Alexei Starovoitov (Facebook), Brenden Blanco (PLUMgrid), Daniel Borkmann (Cisco), Wang Nan (Huawei), Sasha Goldshtein (Sela), and other BPF and bcc contributors!