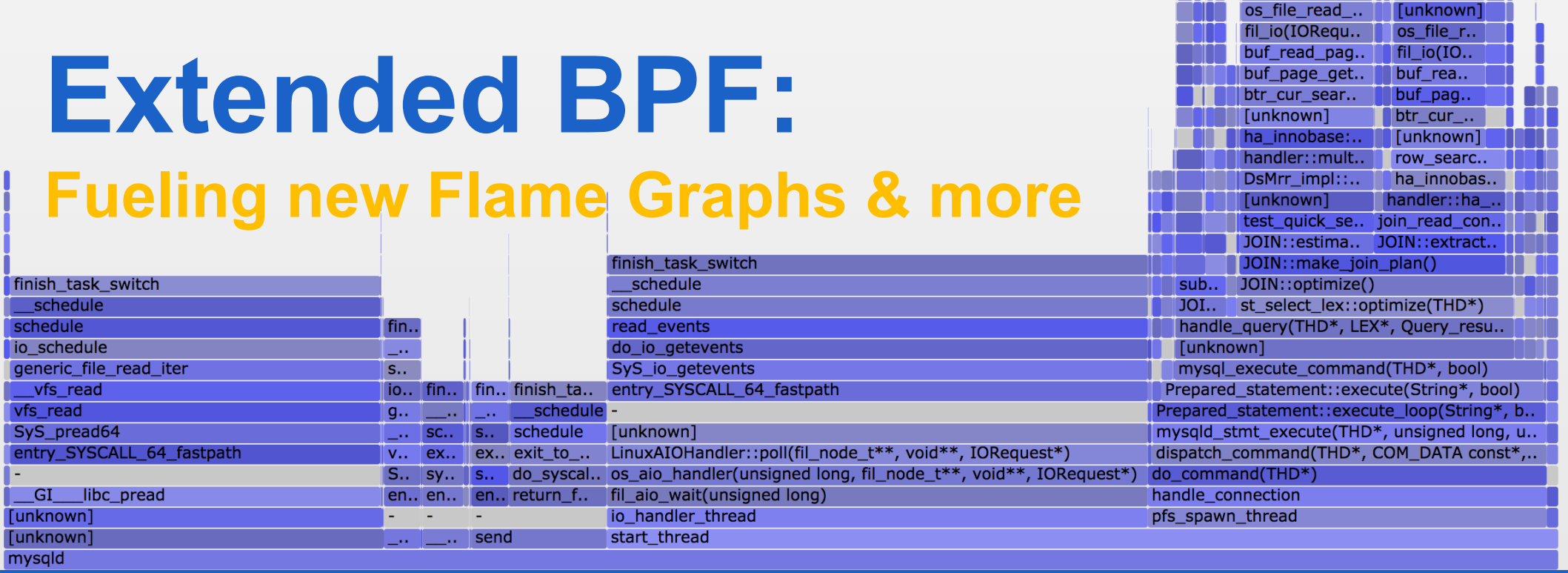
Extended BPF:

Fueling new Flame Graphs & more



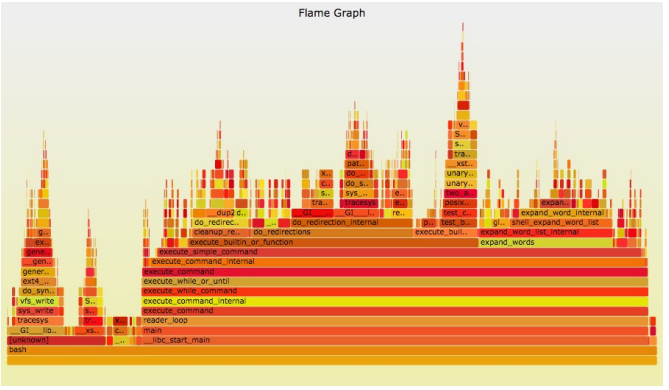
YOW! Perth

IN-PERSON CONFERENCE

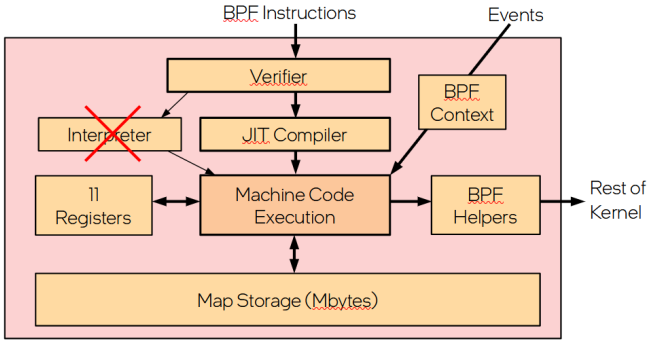
19 — 20 SEP 2022

Brendan Gregg

Agenda



1. CPU flame graphs



2. eBPF

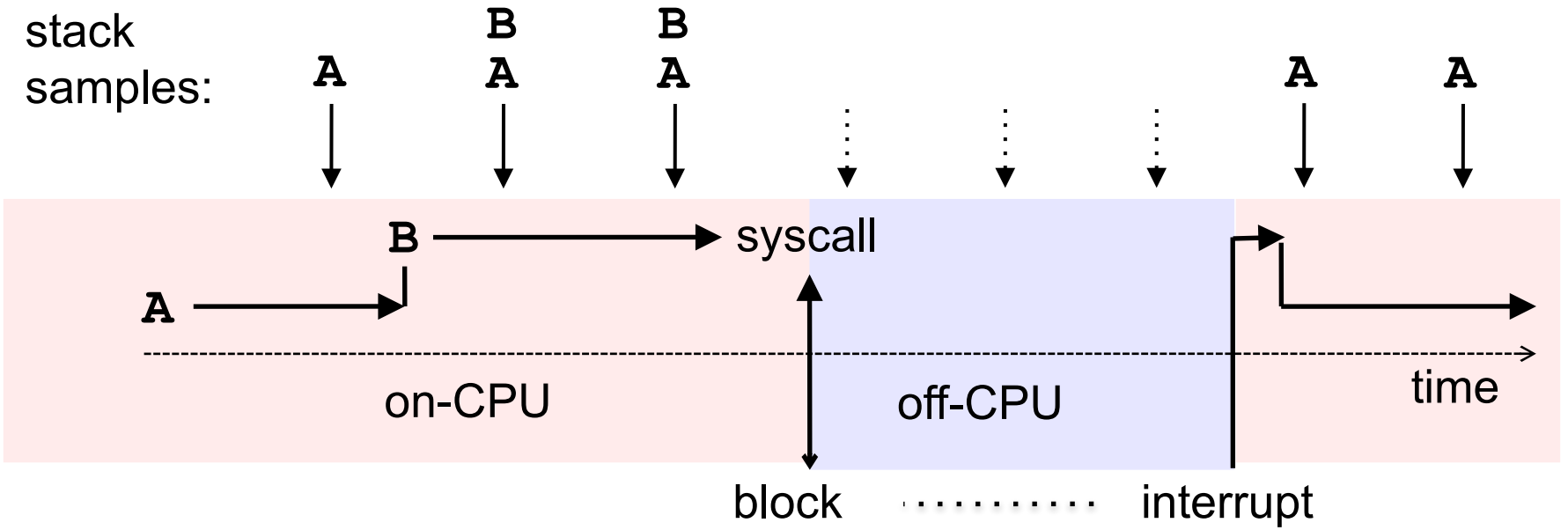
3. eBPF flame graphs

1. CPU Flame Graphs

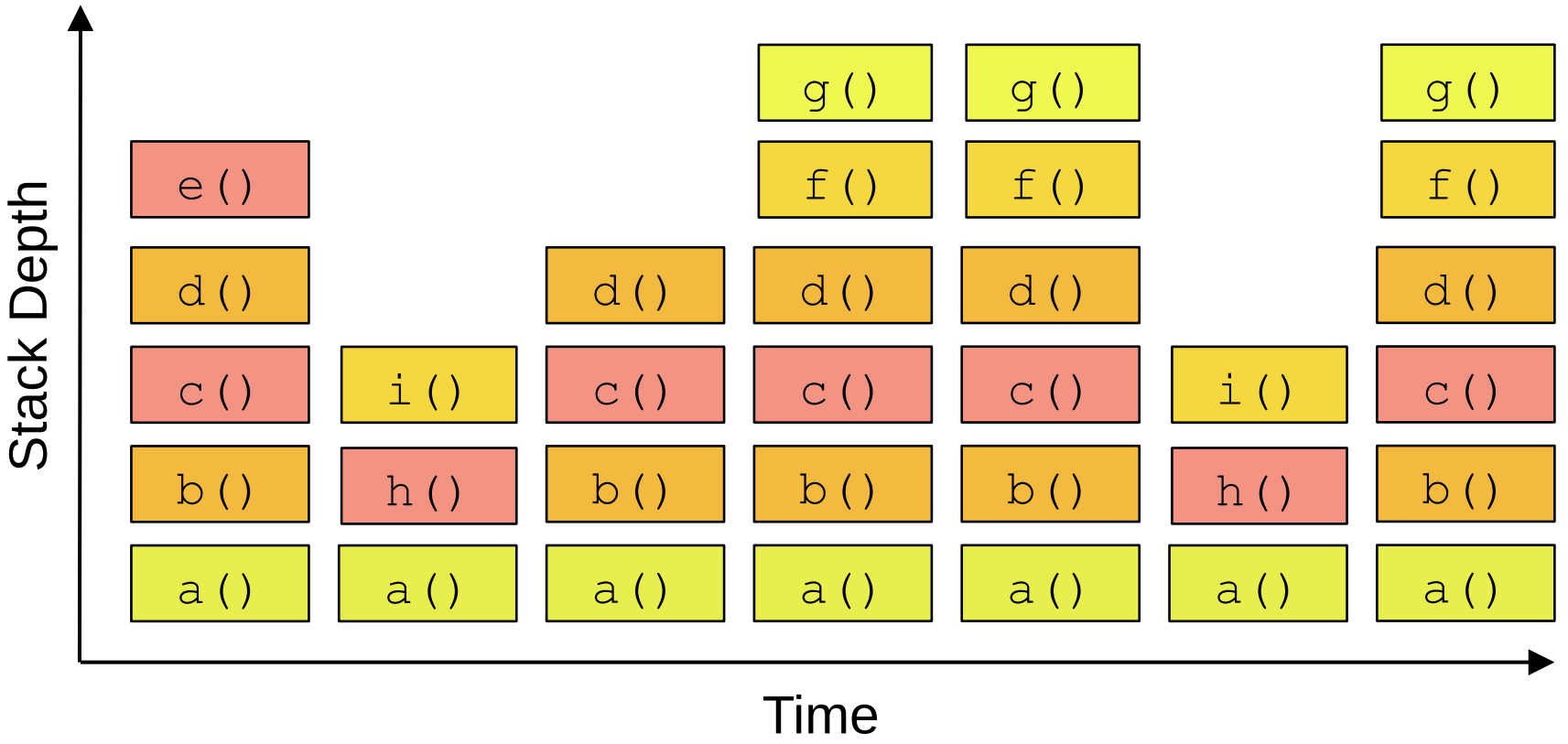
Stack Traces

```
$ jstack 1819
[…]
"main" prio=10 tid=0x00007ff304009000
nid=0x7361 runnable [0x00007ff30d4f9000]
   java.lang.Thread.State: RUNNABLE
   at Func_abc.func_c(Func_abc.java:6)
   at Func_abc.func_b(Func_abc.java:16)
   at Func_abc.func_a(Func_abc.java:23)
   at Func_abc.main(Func_abc.java:27)
```

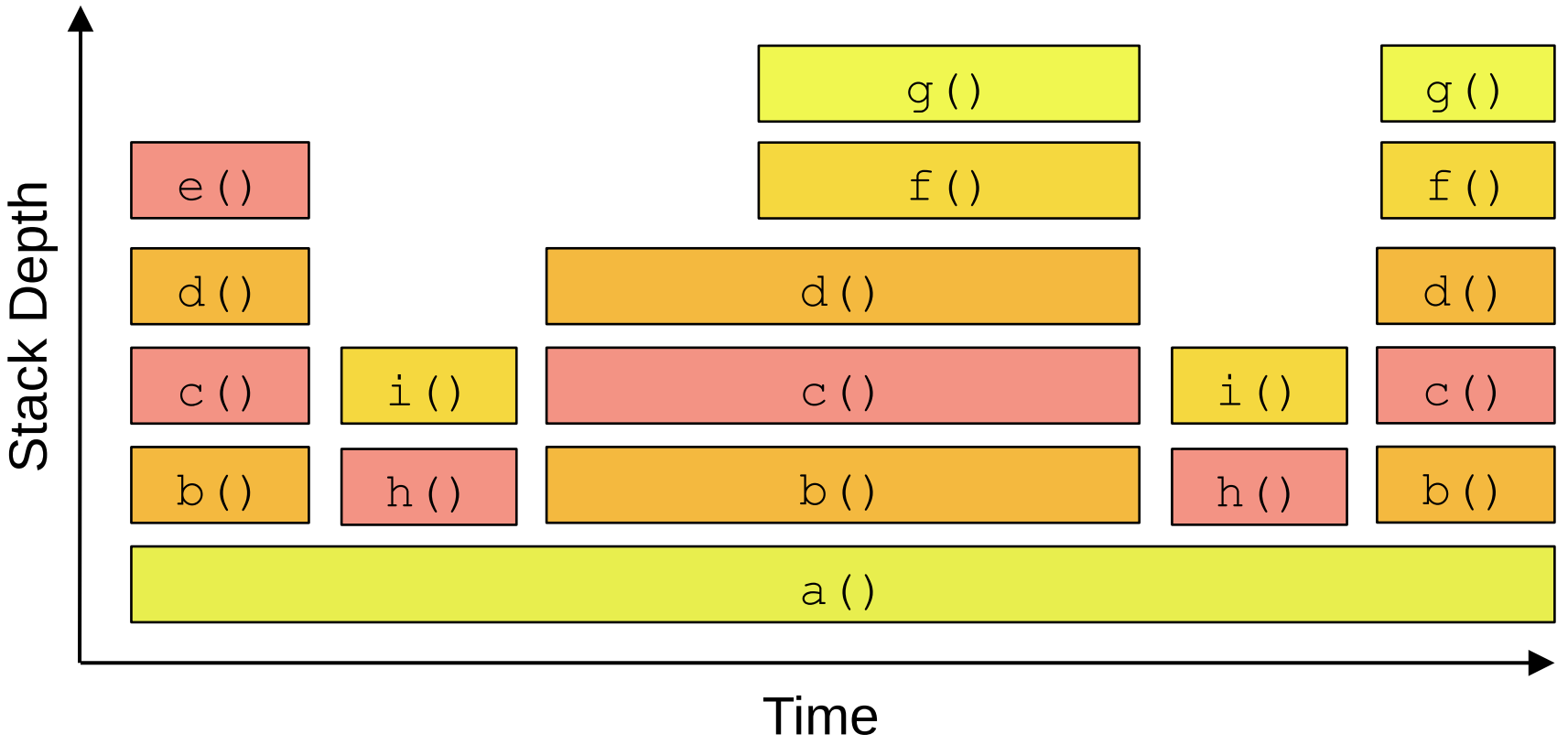
CPU Profiling



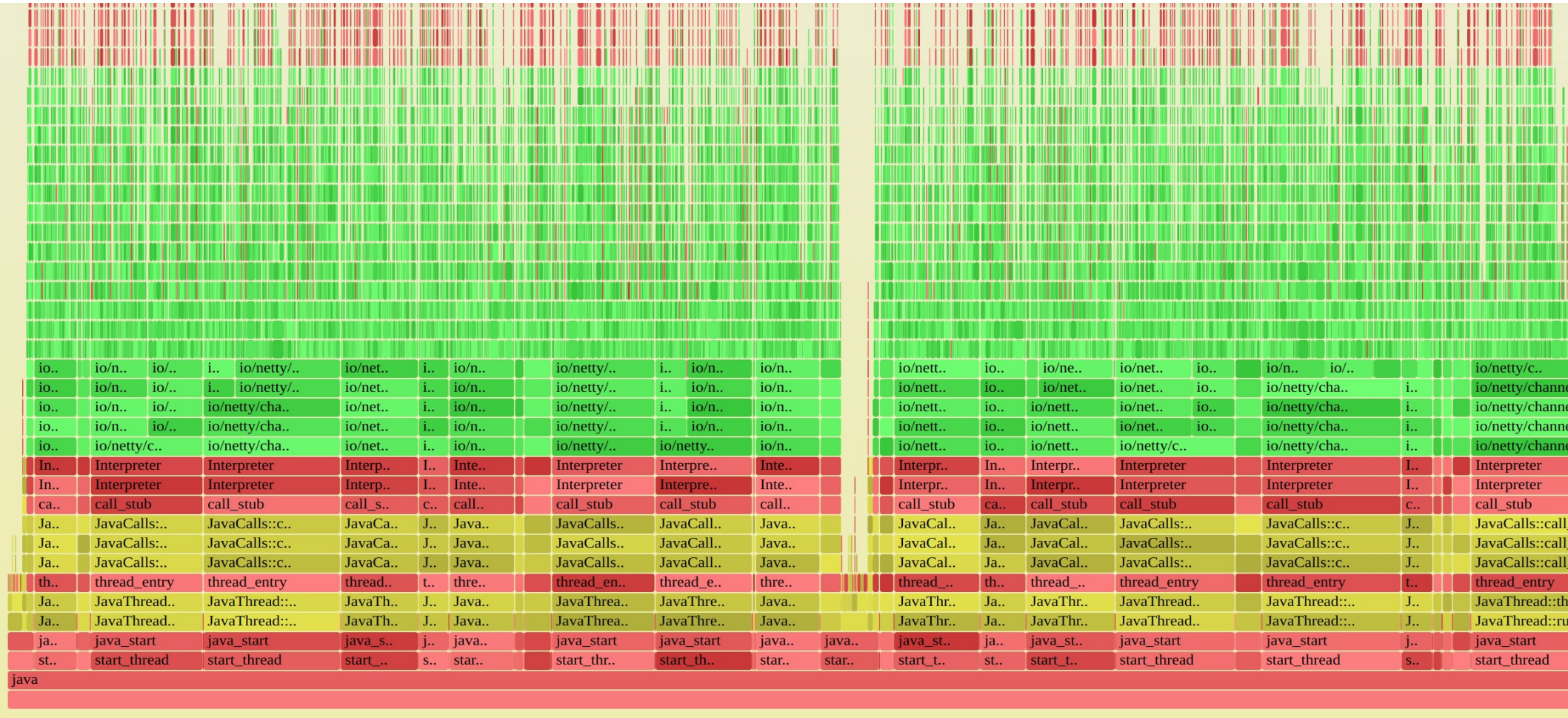
Stack Samples



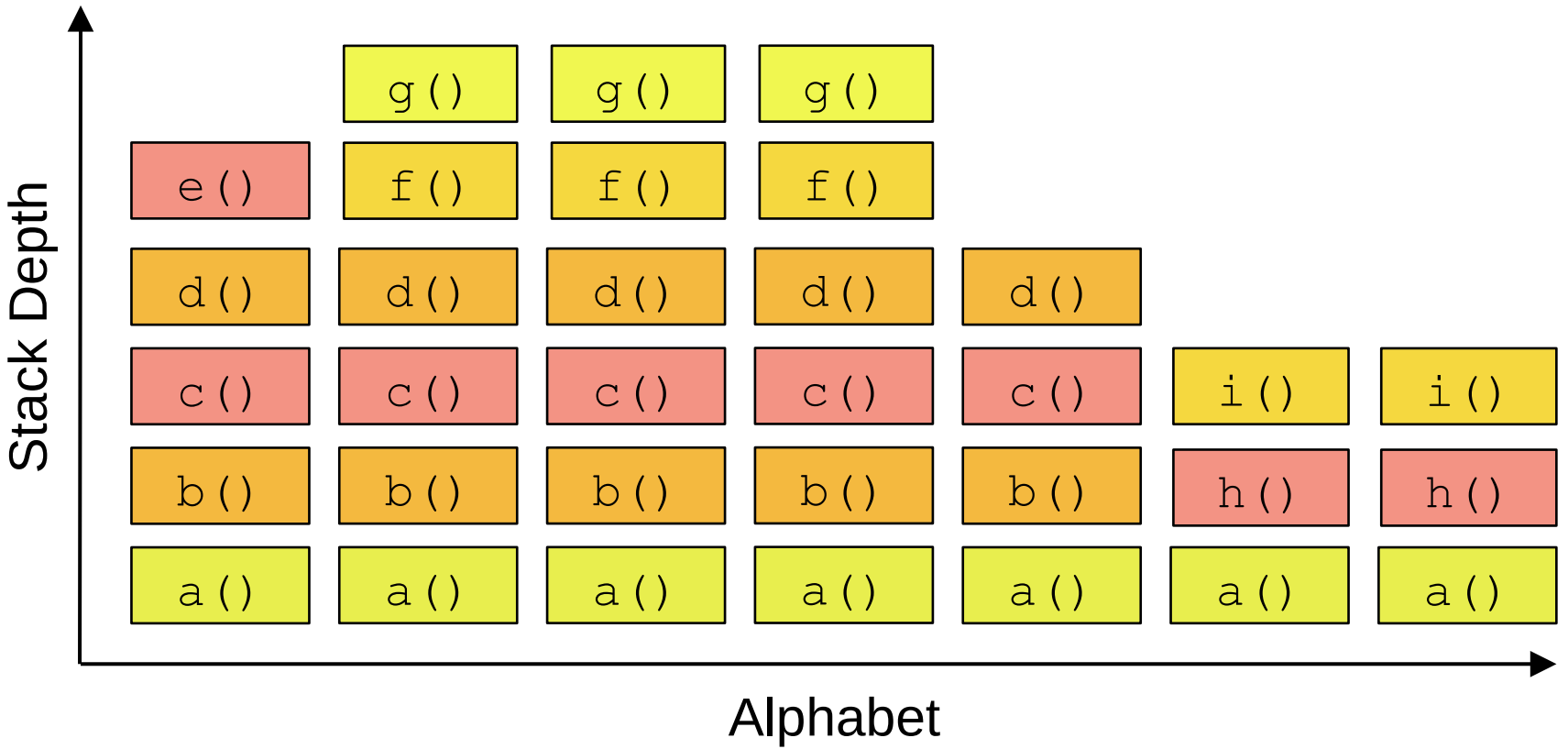
Stack Samples: Merged



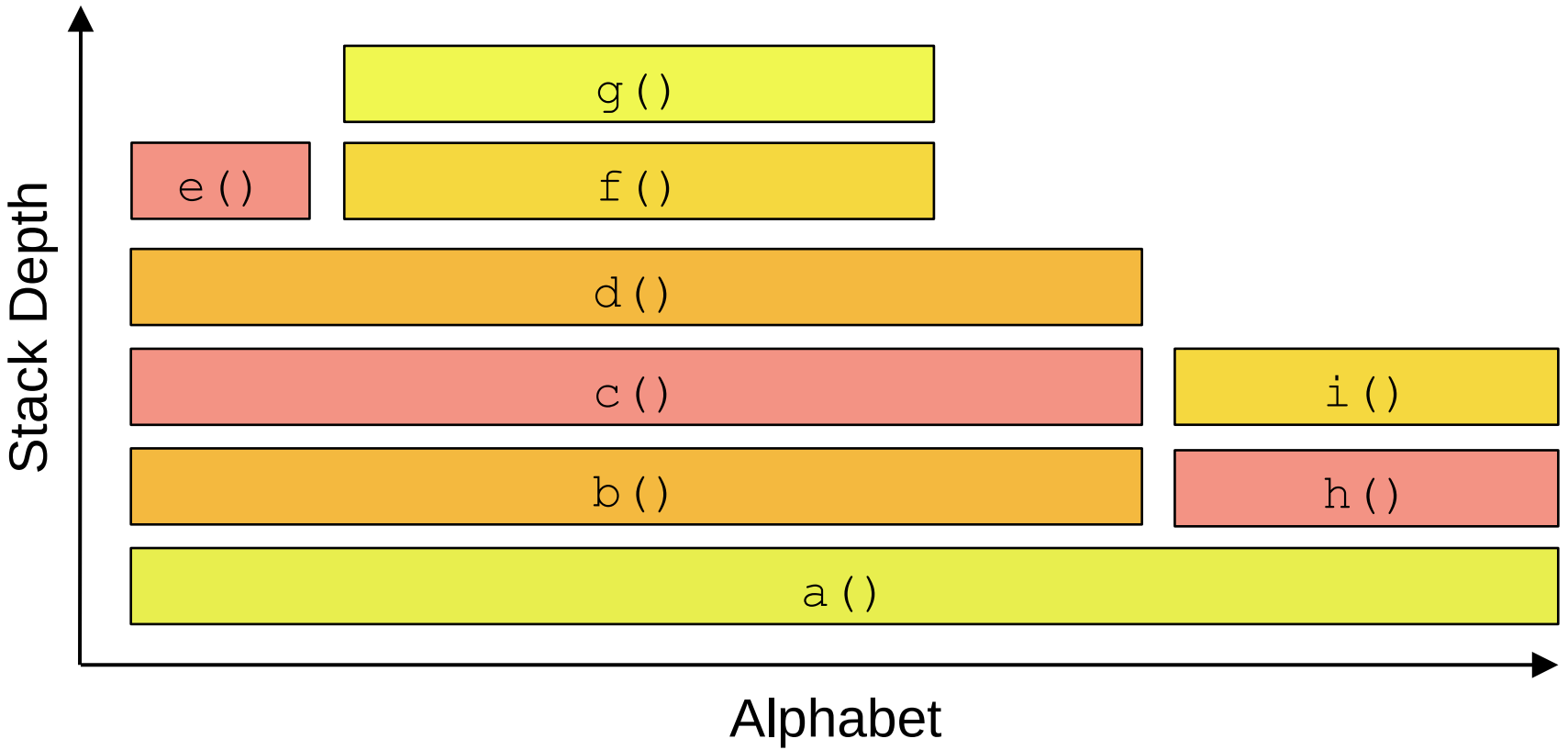
Example Profile (“hair graph”)



Alphabet Sort



Alphabet Merged (“Flame Graph”)



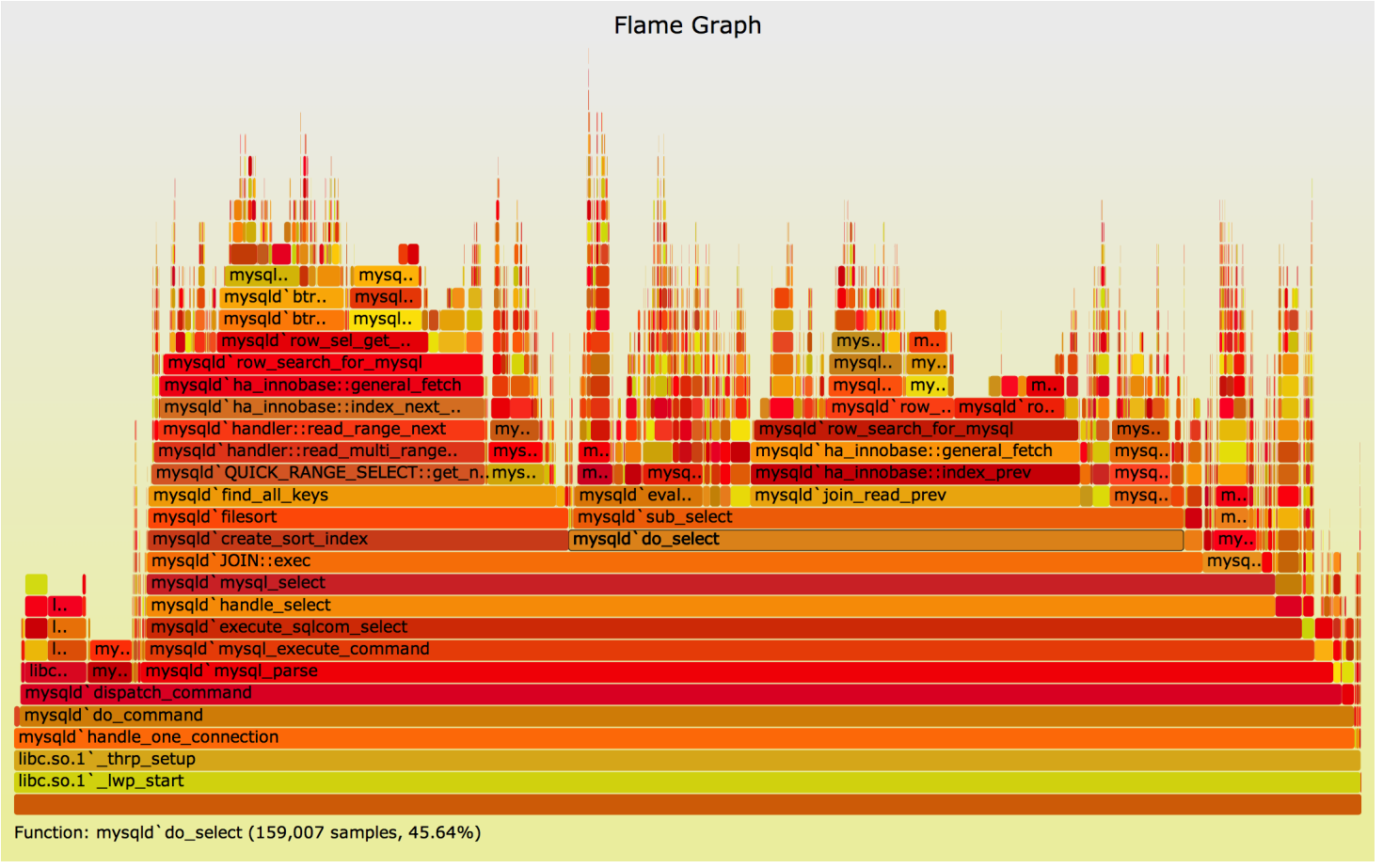
Example Profile: Flame Graph

Origin (2011): CPU Profiling

```
# dtrace -x ustackframes=100 -n 'profile-997 /execname == "mysqld"/ {
    @[ustack()] = count(); } tick-60s { exit(0); }'
[… over 500,000 lines truncated …]

              libc.so.1`__priocntlset+0xa
              libc.so.1`getparam+0x83
              libc.so.1`pthread_getschedparam+0x3c
              libc.so.1`pthread_setschedprio+0x1f
              mysqld`_Z16dispatch_command19enum_server_commandP3THDPcj+0x9ab
              mysqld`_Z10do_commandP3THD+0x198
              mysqld`handle_one_connection+0x1a6
              libc.so.1`_thrp_setup+0x8d
              libc.so.1`_lwp_start
             4884

              mysqld`_Z13add_to_statusP17system_status_varS0_+0x47
              mysqld`_Z22calc_sum_of_all_statusP17system_status_var+0x67
              mysqld`_Z16dispatch_command19enum_server_commandP3THDPcj+0x1222
              mysqld`_Z10do_commandP3THD+0x198
              mysqld`handle_one_connection+0x1a6
              libc.so.1`_thrp_setup+0x8d
              libc.so.1`_lwp_start
             5530
```

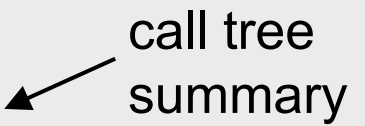
Full output

… as a Flame Graph

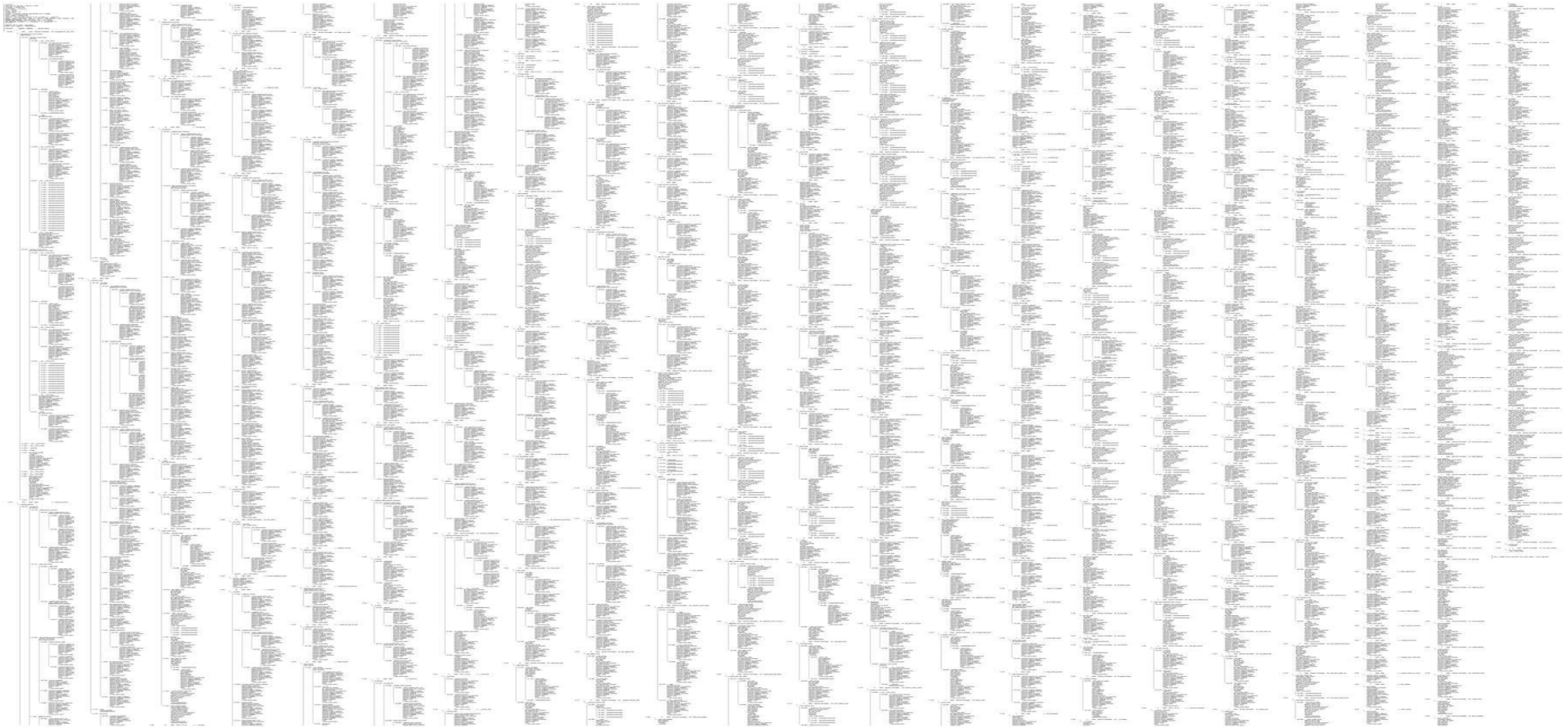


Linux example: perf Profiling

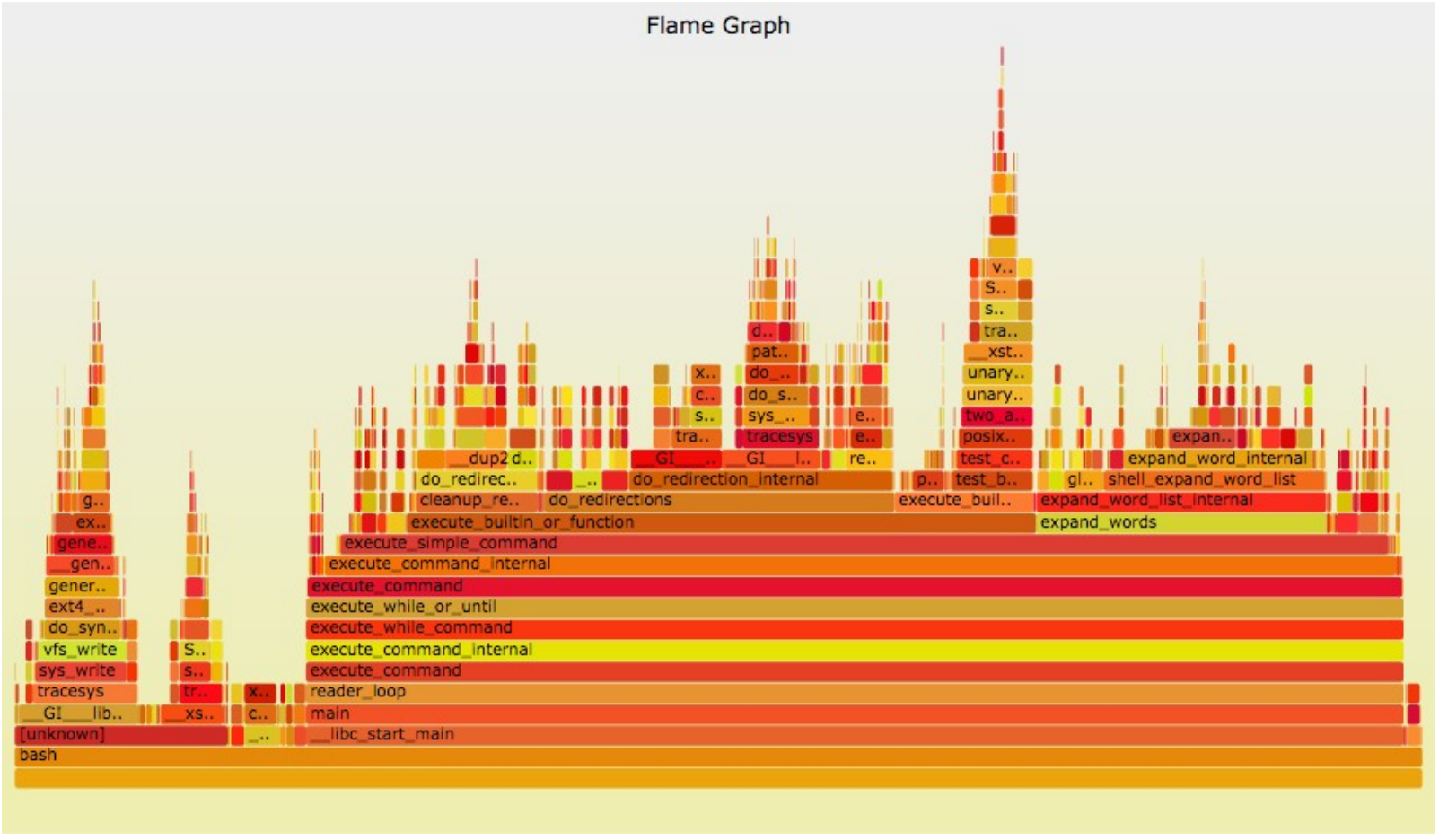
```
# perf record -F 99 -ag -- sleep 30
[ perf record: Woken up 9 times to write data ]
[ perf record: Captured and wrote 2.745 MB perf.data (~119930 samples) ]
# perf report -n -stdio
[…]
# Overhead       Samples  Command      Shared Object                          Symbol
# ........  ............  .......  .................  ..............................
#
    20.42%           605     bash  [kernel.kallsyms]  [k] xen_hypercall_xen_version
               |
               --- xen_hypercall_xen_version
                   check_events
                  |
                  |--44.13%-- syscall_trace_enter
                  |          tracesys
                  |          |
                  |          |--35.58%-- __GI___libc_fcntl
                  |          |          |
                  |          |          |--65.26%-- do_redirection_internal
                  |          |          |          do_redirections
                  |          |          |          execute_builtin_or_function
                  |          |          |          execute_simple_command
[… ~13,000 lines truncated …]
```



Full perf Output



… as a Flame Graph



Flame Charts (2013)

https://bugs.webkit.org/show_bug.cgi?id=111162

Ilya Tikhonovsky 2013-03-01 04:33:26 PST Description

Flame Chart may give to the developer a better clue what is going on with the performance without expanding the entire tree.

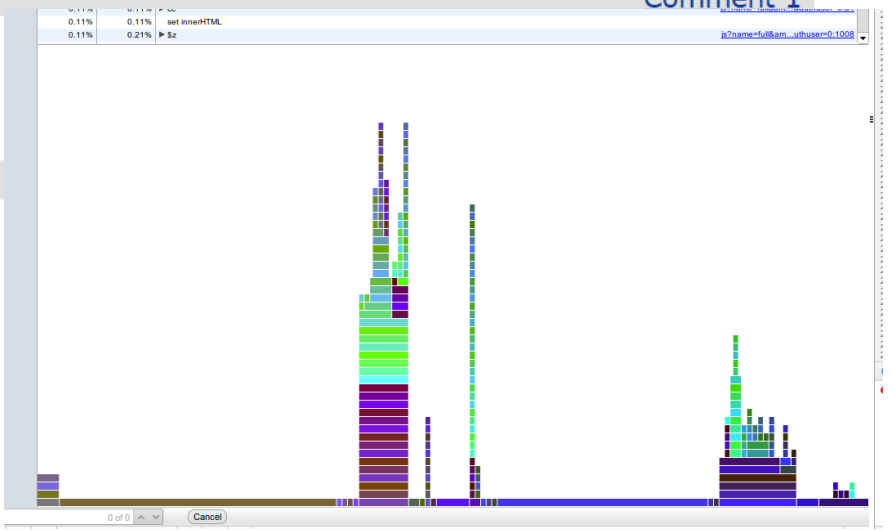
http://dtrace.org/blogs/brendan/2011/12/16/flame-graphs/

Ilya Tikhonovsky 2013-03-01 04:35:00 PST Comment 1

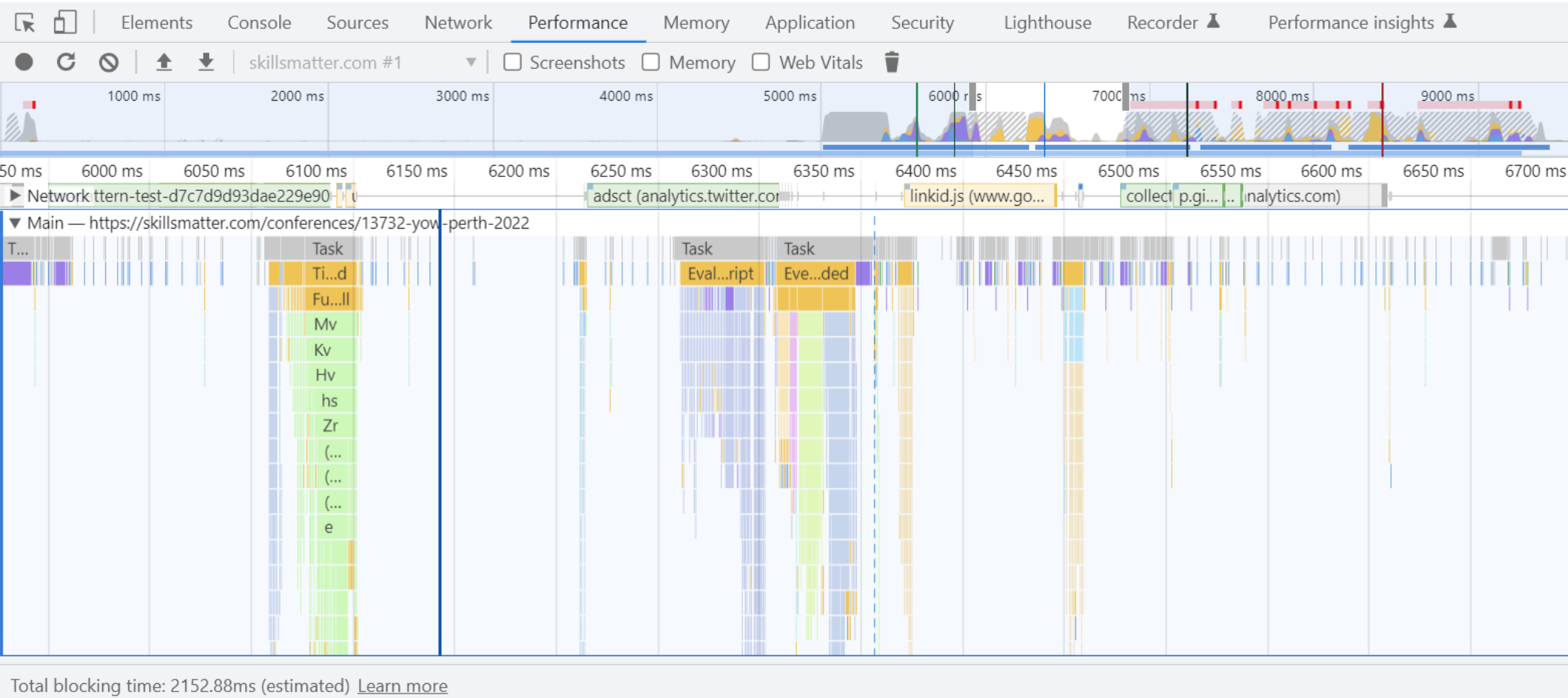
Created attachment 190933 [details]
screenshot

Ilya Tikhonovsky 2013-03-01 04:37:34 PST

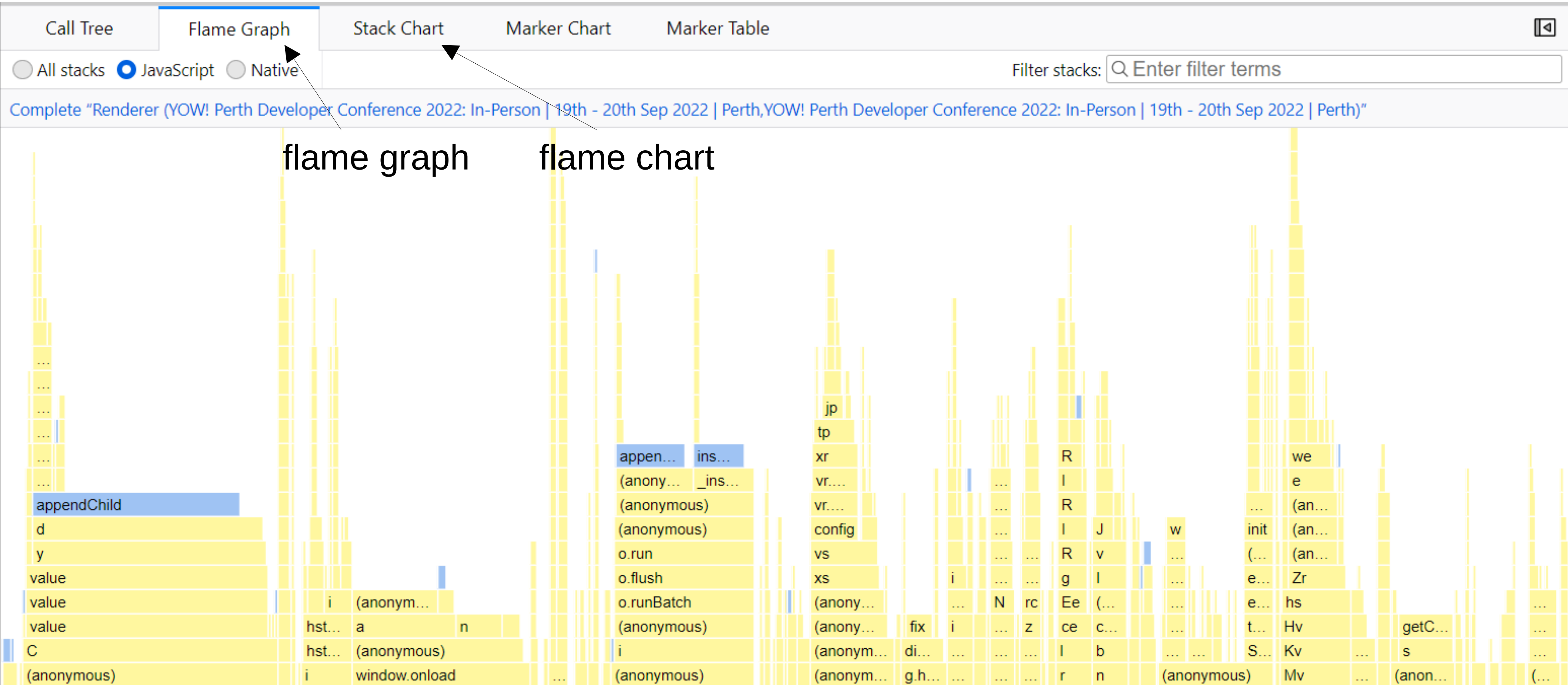
Created attachment 190934 [details]
Patch



Chrome DevTools Flame Charts (2022)



Firefox Profiler Flame Graph (2022)



Flame Graphs

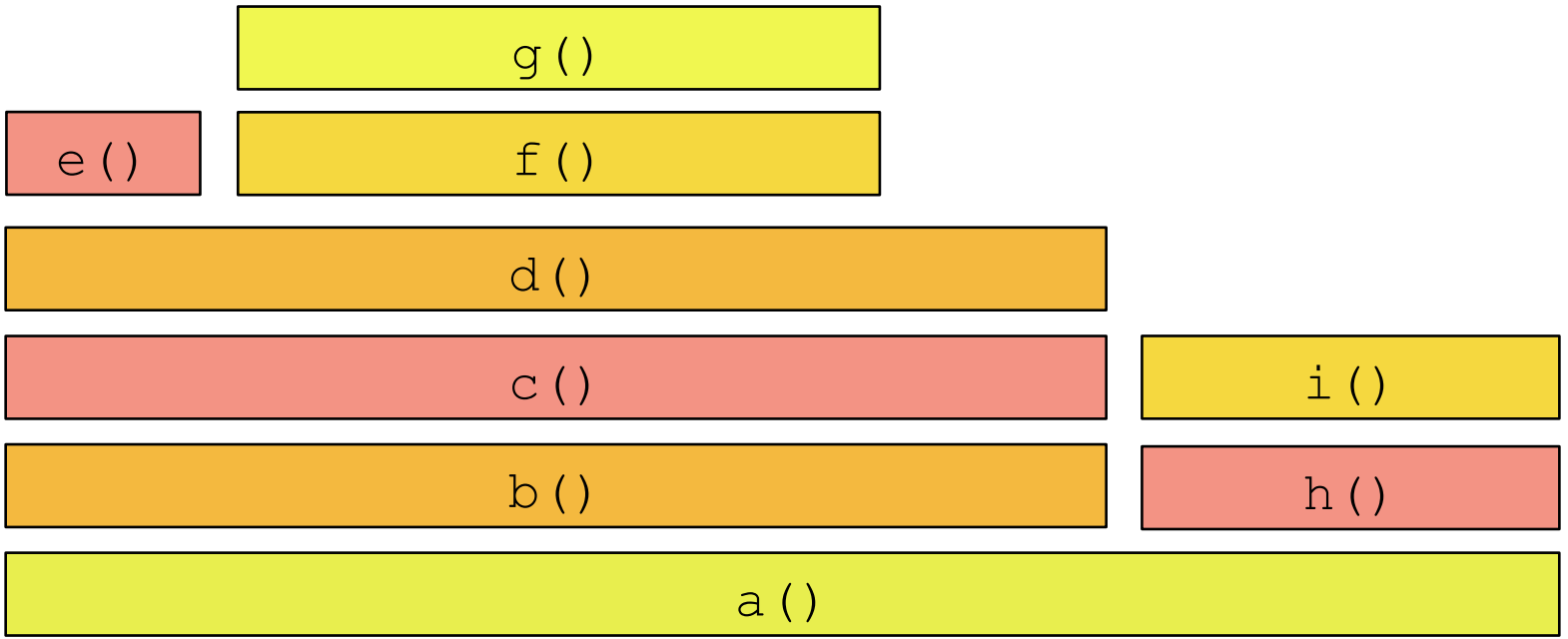
x-axis: population

alphabet sort or another frame merging algorithm

Flame Charts

x-axis: time

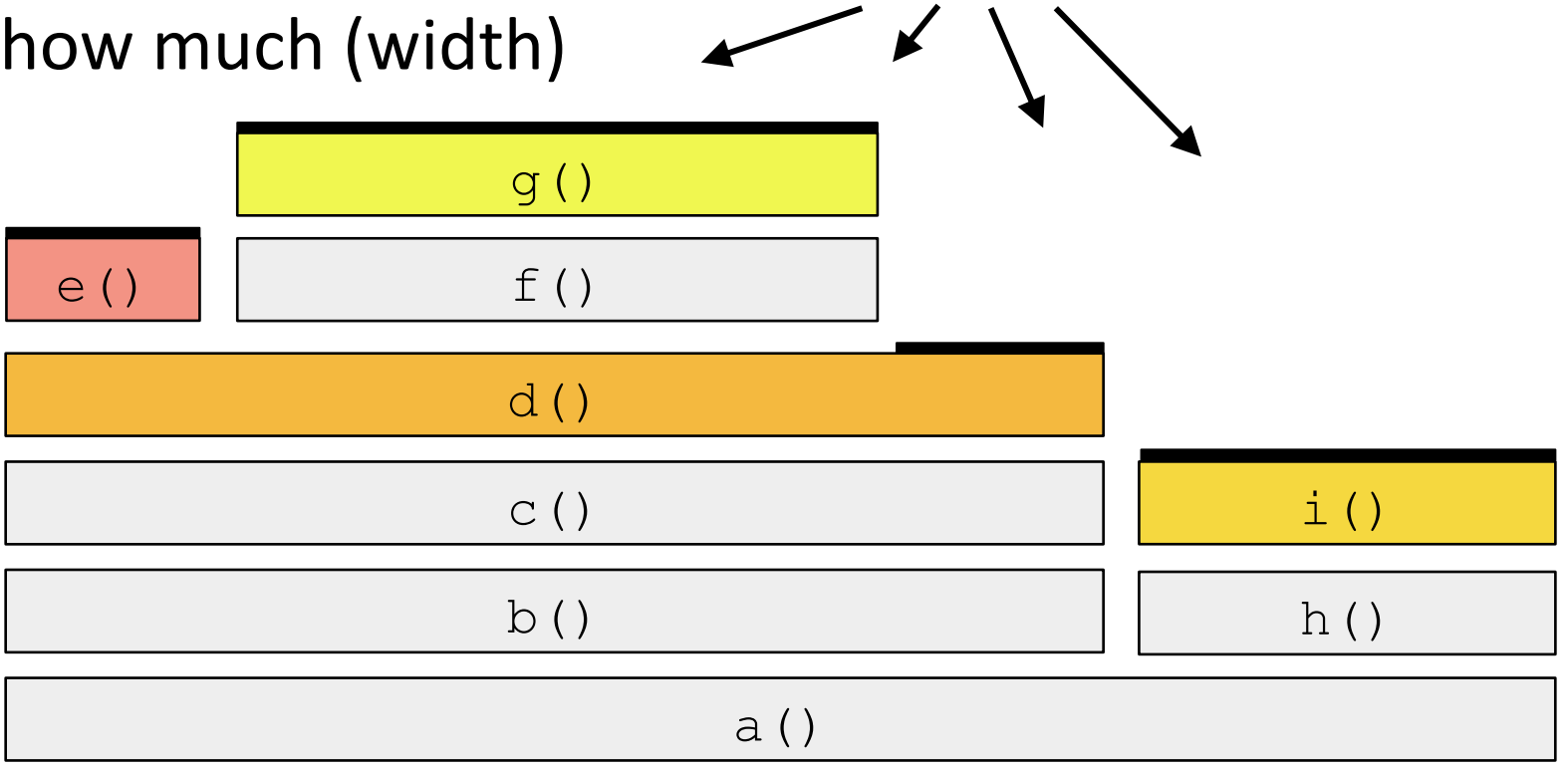
Flame Graph Interpretation



Flame Graph Interpretation (1/4)

Top edge shows who is running on-CPU,

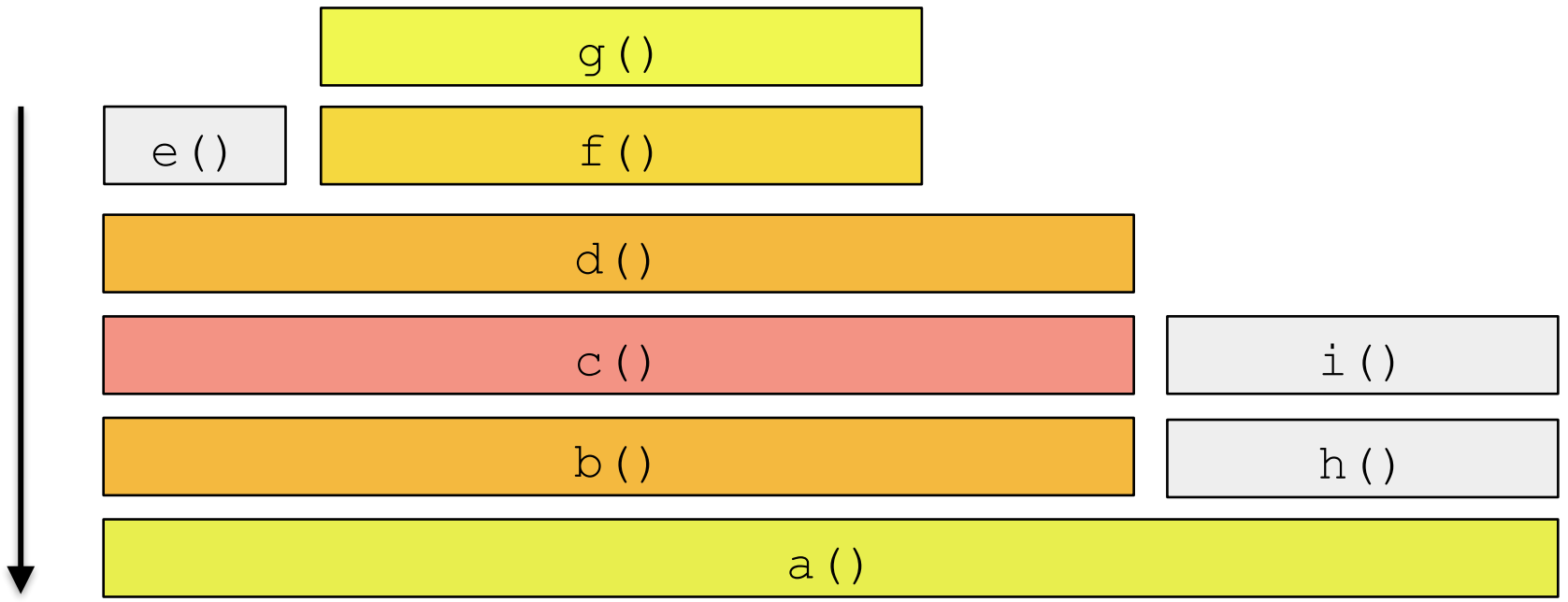
and how much (width)



Flame Graph Interpretation (2/4)

Top-down shows ancestry

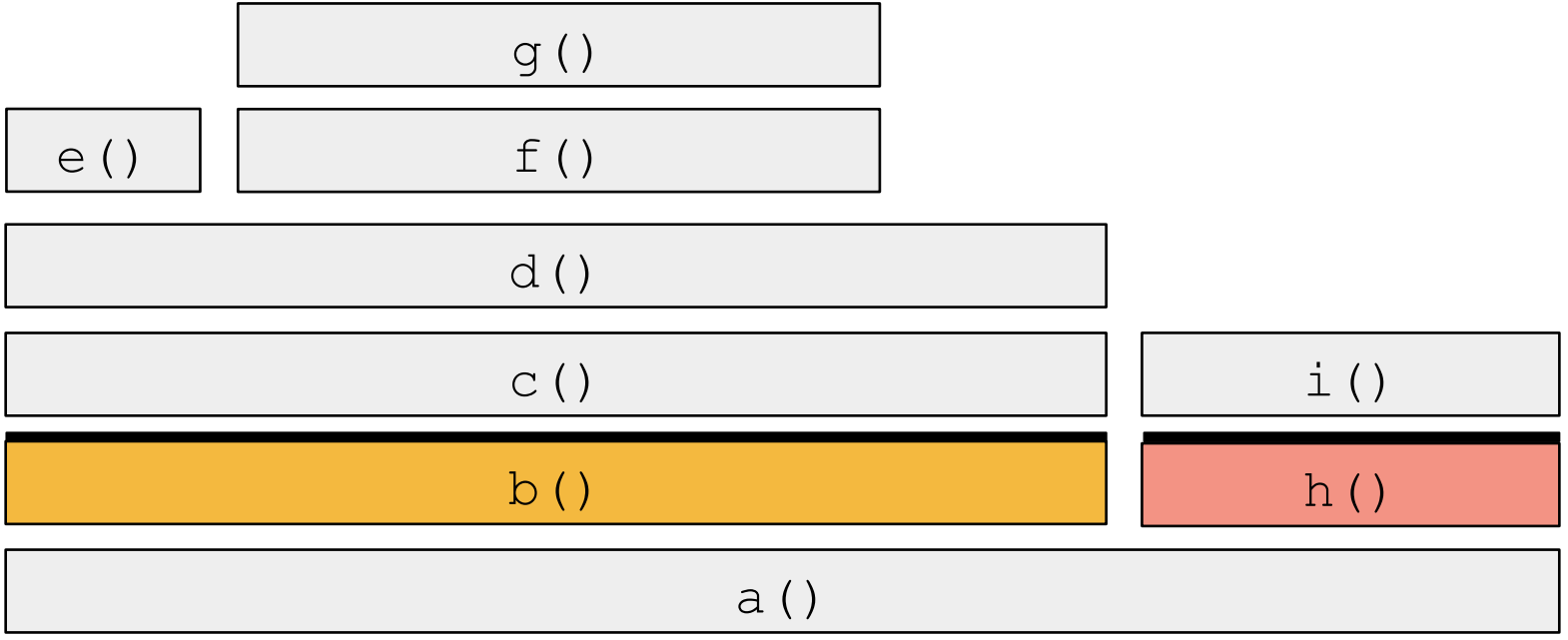
e.g., from g():



Flame Graph Interpretation (3/4)

Widths are proportional to presence in samples

e.g., comparing b() to h() (incl. children)

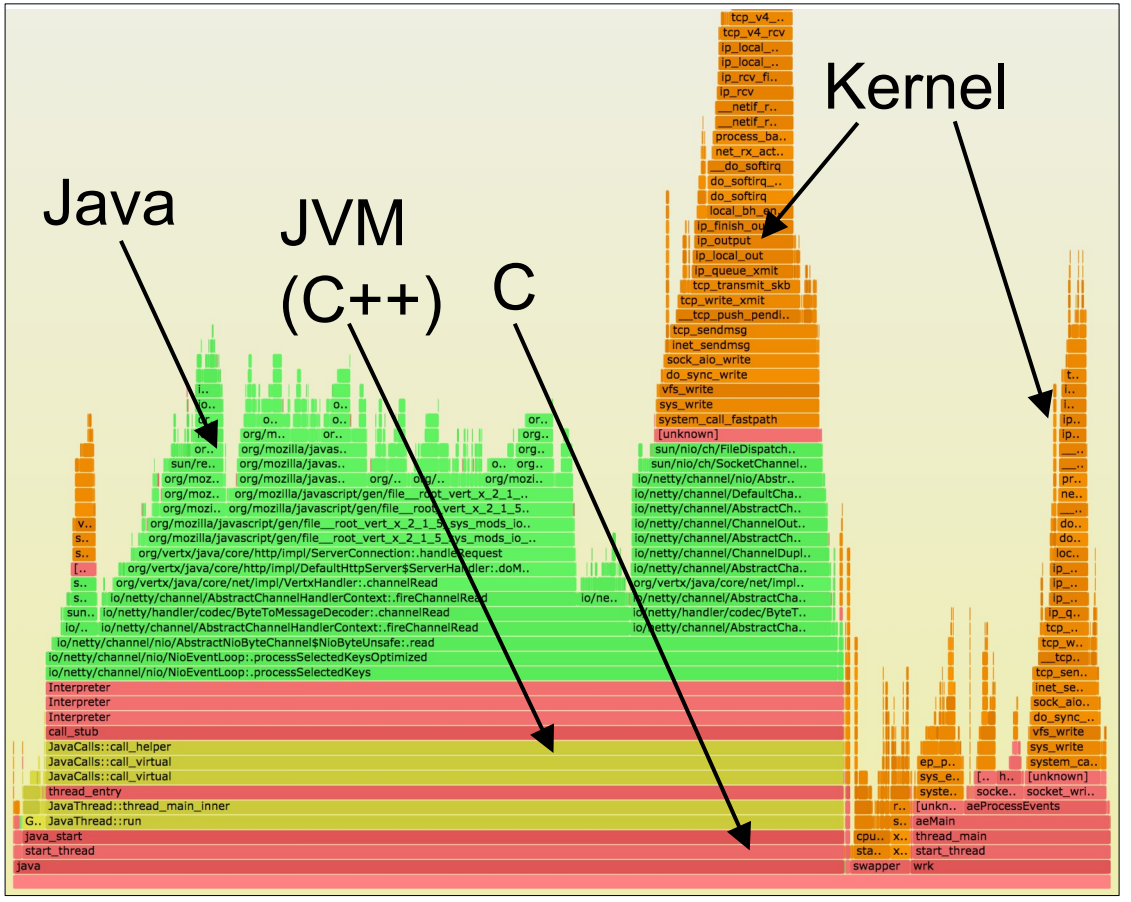


Flame Graph Interpretation (4/4)

Colors randomized to differentiate frames

Or used for code type; e.g.:

- green == JIT (e.g., Java)
- aqua == inlined
- red == user-level
- orange == kernel
- yellow == C++
- magenta == search term



Flame Graph Summary

Visualizes a collection of stack traces

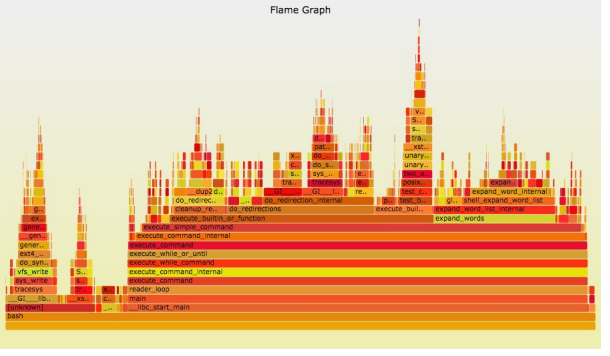
- **x-axis**: population: e.g., alphabetical sort to maximize merging
- **y-axis**: stack depth
- **color**: random (default) or a dimension

Original implementation: Perl + SVG + JavaScript

- **https://github.com/brendangregg/FlameGraph**
- Takes input from many different profilers

References:

- http://www.brendangregg.com/flamegraphs.html
- http://queue.acm.org/detail.cfm?id=2927301
- "The Flame Graph" CACM, June 2016

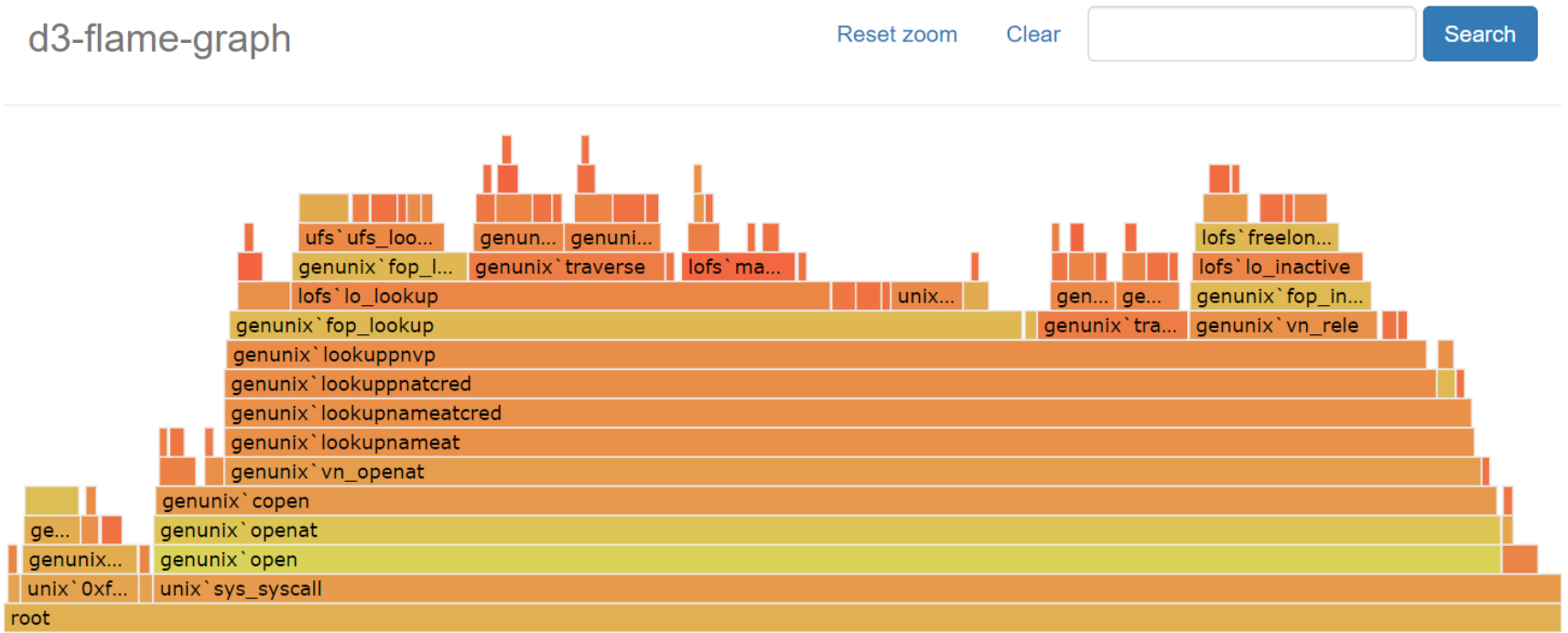


d3 Flame Graphs

Created by Martin Spier (my colleague at Netflix)

Used by Netflix's internal continuous profiler ("FlameCommander")

https://github.com/spiermar/d3-flame-graph



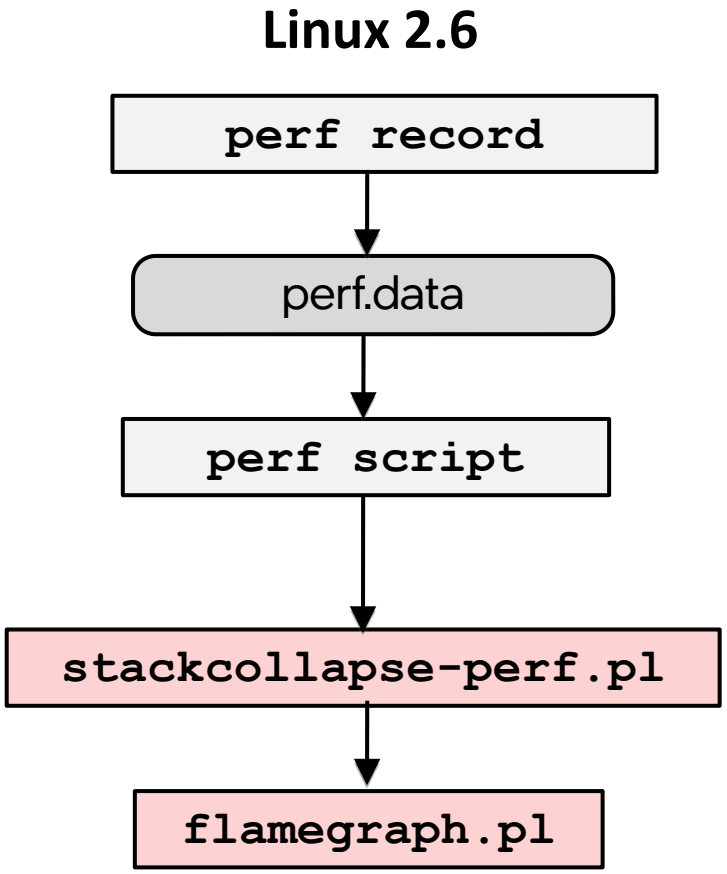
Linux CPU Flame Graphs in 3 Easy Steps

```
# git clone https://github.com/brendangregg/FlameGraph; cd FlameGraph
# perf record -F 49 -ag -- sleep 30
# perf script | ./stackcollapse-perf.pl | ./flamegraph.pl > out.svg
```

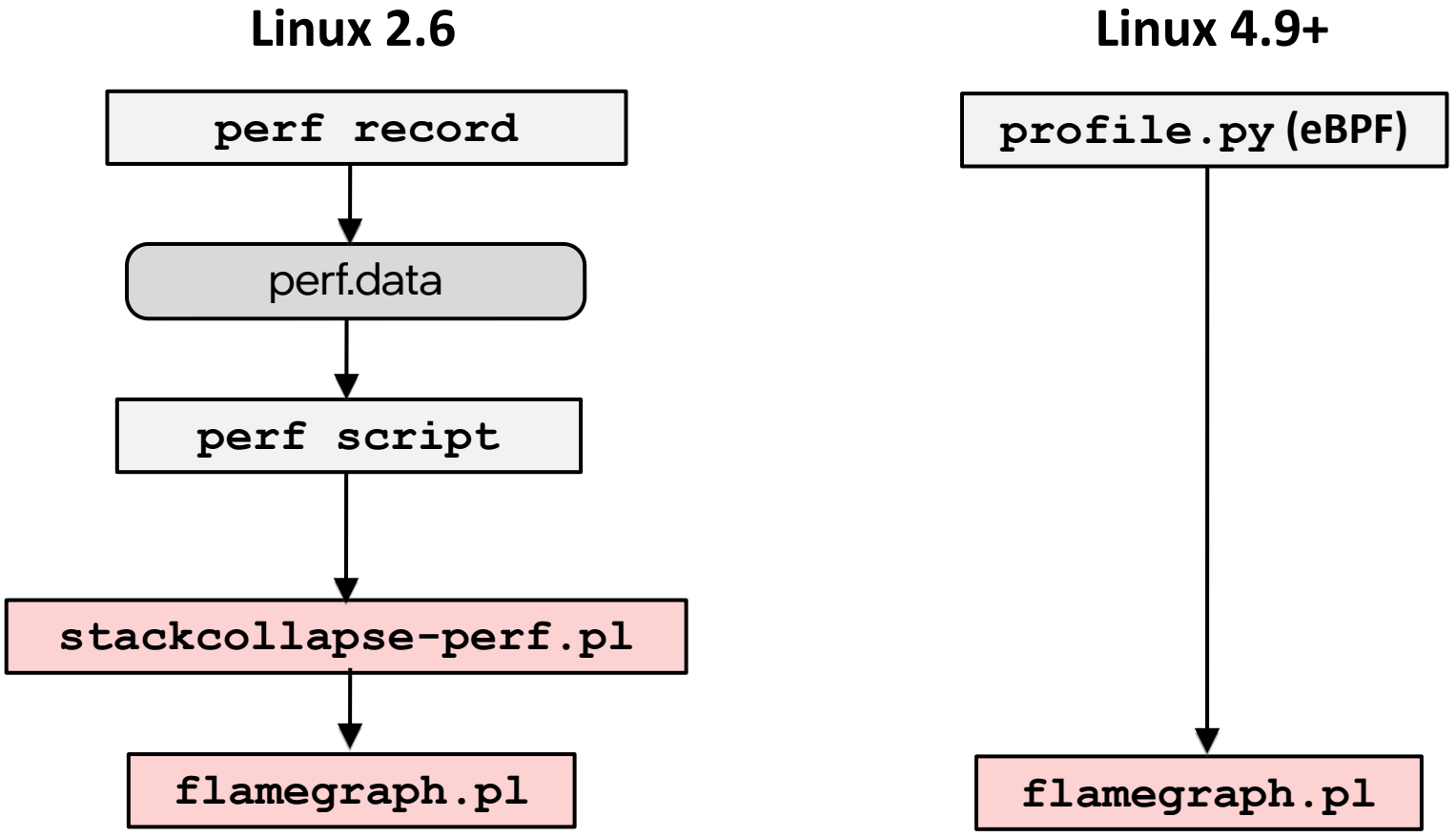
Some runtimes (e.g., JVM) require extra steps for stacks & symbols

see https://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html

CPU Flame Graphs



eBPF-based CPU Flame Graphs



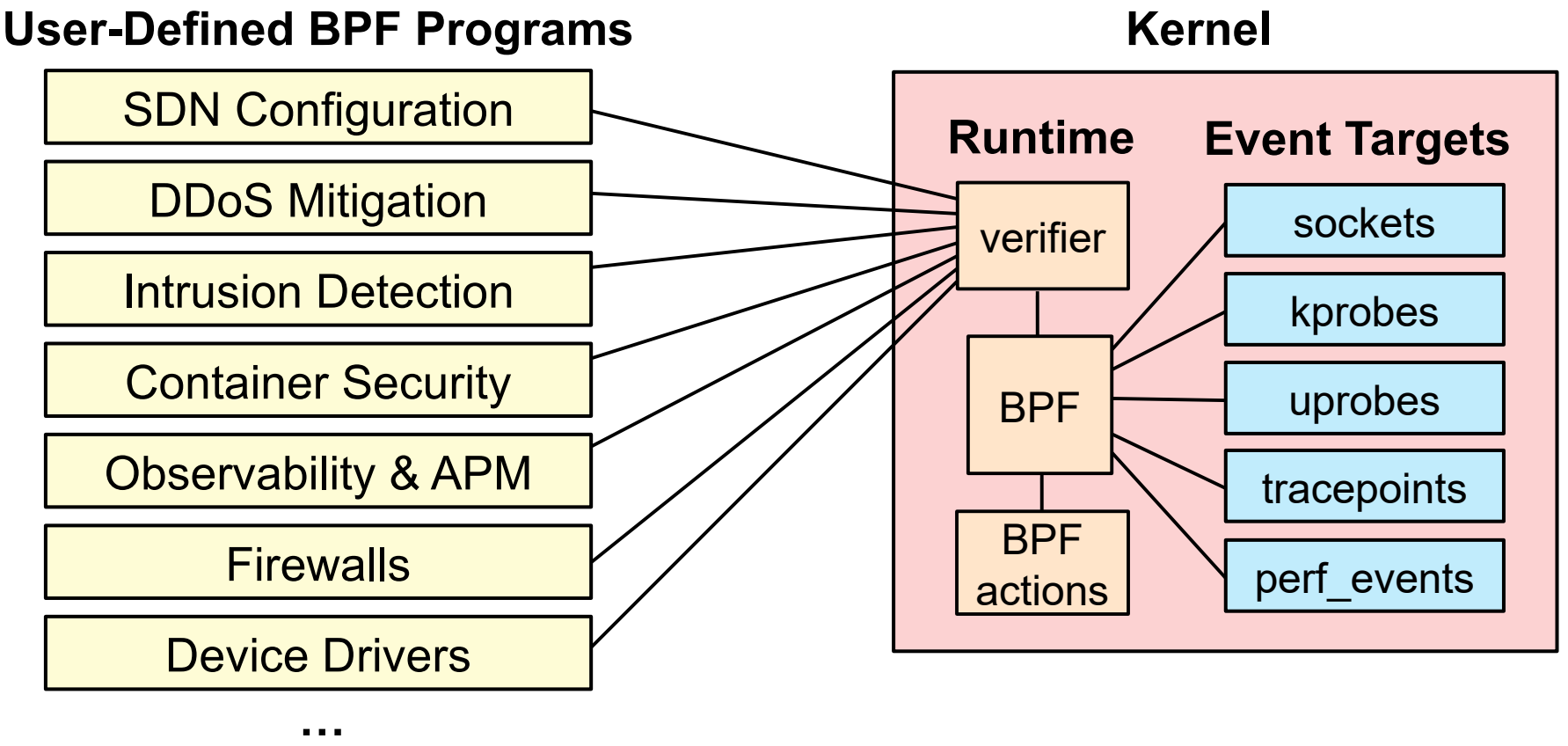
2. eBPF

BPF 1992: Berkeley Packet Filter

```
# tcpdump -d host 127.0.0.1 and port 80
(000) ldh      [12]
(001) jeq      #0x800           jt 2   jf 18
(002) ld       [26]
(003) jeq      #0x7f000001      jt 6   jf 4
(004) ld       [30]
(005) jeq      #0x7f000001      jt 6   jf 18
(006) ldb      [23]
(007) jeq      #0x84            jt 10  jf 8
(008) jeq      #0x6             jt 10  jf 9
(009) jeq      #0x11            jt 10  jf 18
(010) ldh      [20]
(011) jset     #0x1fff          jt 18  jf 12
(012) ldxb     4*([14]&0xf)
(013) ldh      [x + 14]
(014) jeq      #0x50            jt 17  jf 15
(015) ldh      [x + 16]
(016) jeq      #0x50            jt 17  jf 18
(017) ret      #262144
(018) ret      #0
```

A limited
virtual machine for
efficient packet filters

eBPF 2022


User-Defined BPF Programs
SDN Configuration
DDoS Mitigation
Intrusion Detection
Container Security
Observability & APM
Firewalls
Device Drivers
...
Kernel
Runtime
Event Targets
verifier
BPF
BPF actions
sockets
kprobes
uprobes
tracepoints
perf_events

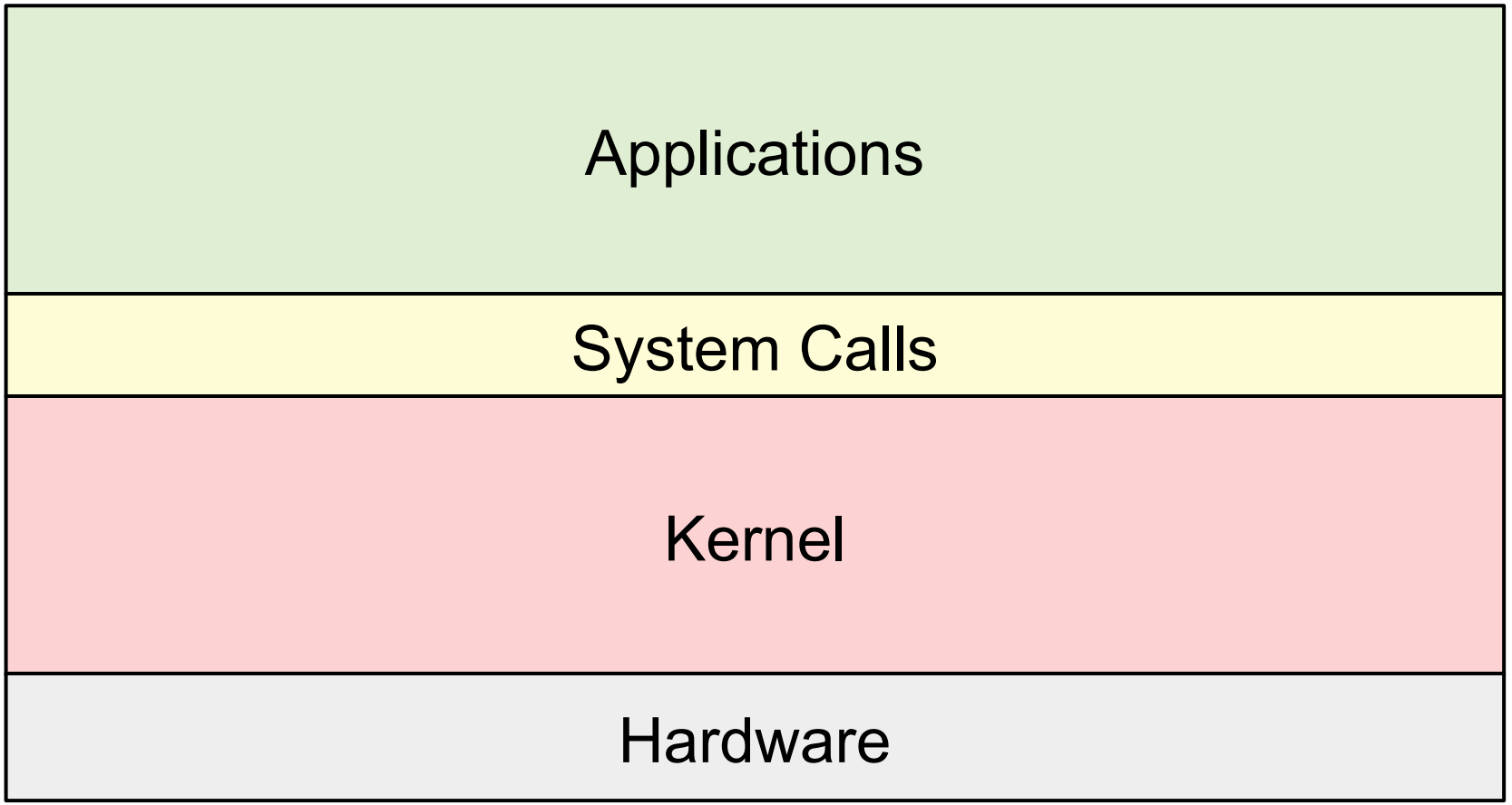
BPF (kernel engineers)

eBPF (marketing)

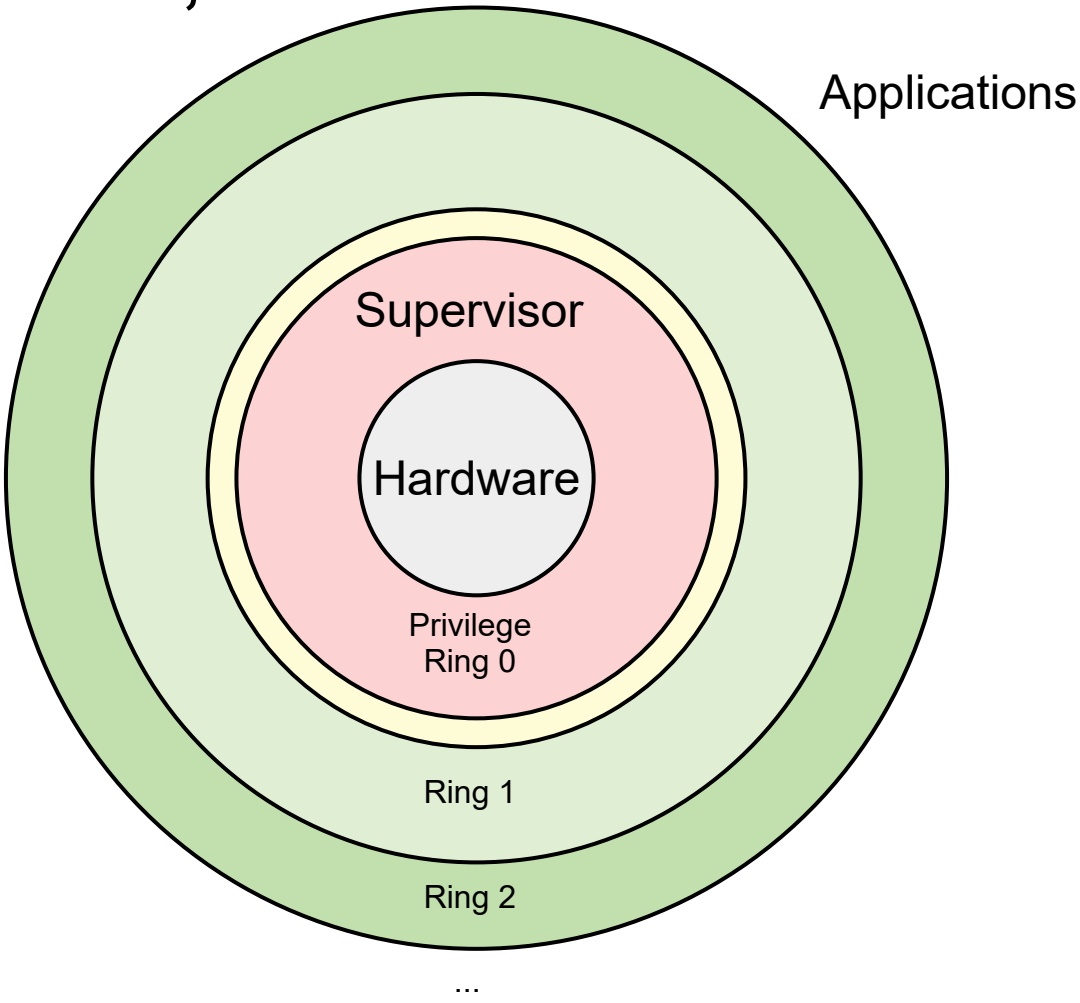
BPF is no longer an acronym

eBPF tracing pony

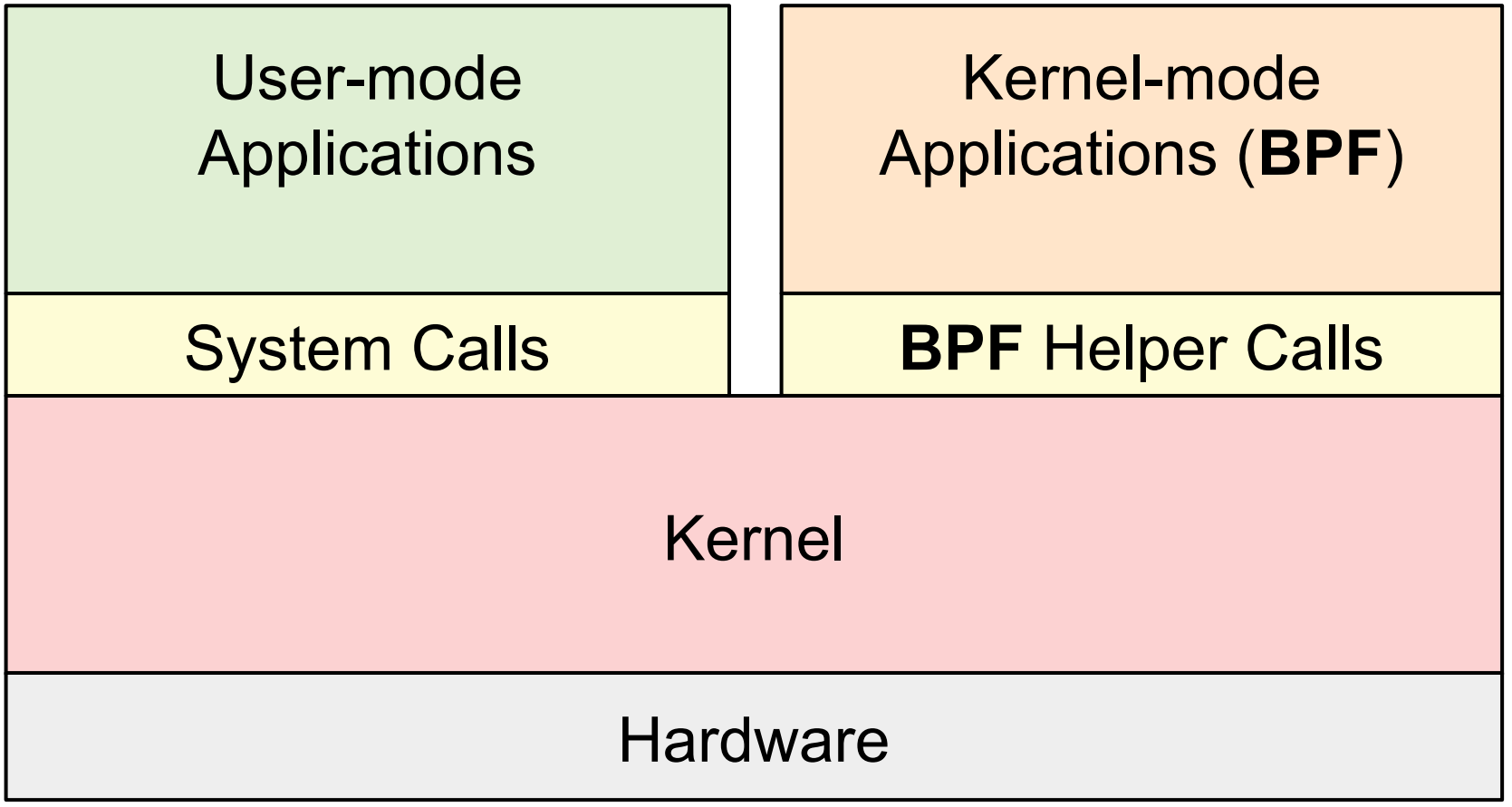
50 Years, one (dominant) OS model



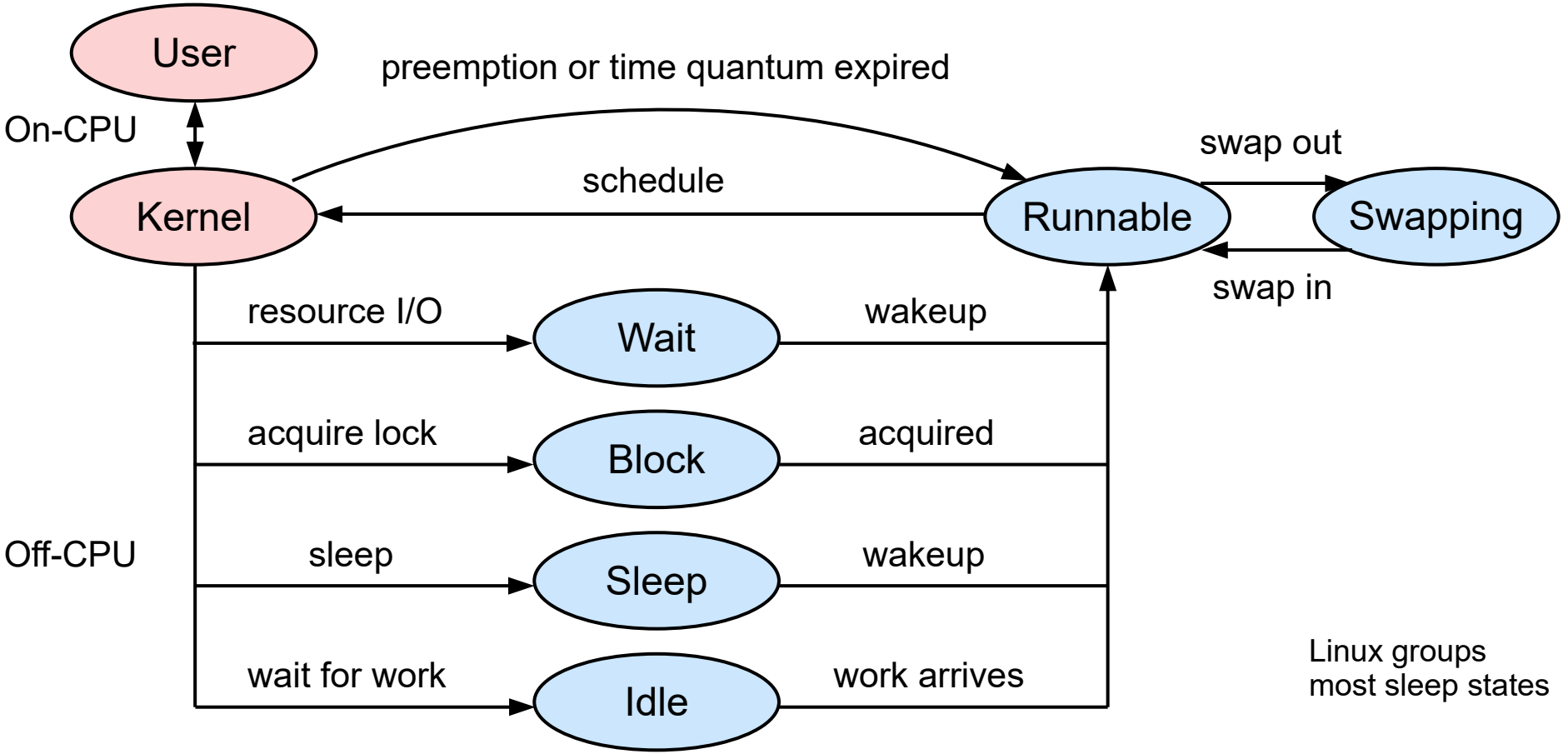
Origins: Multics, 1960s



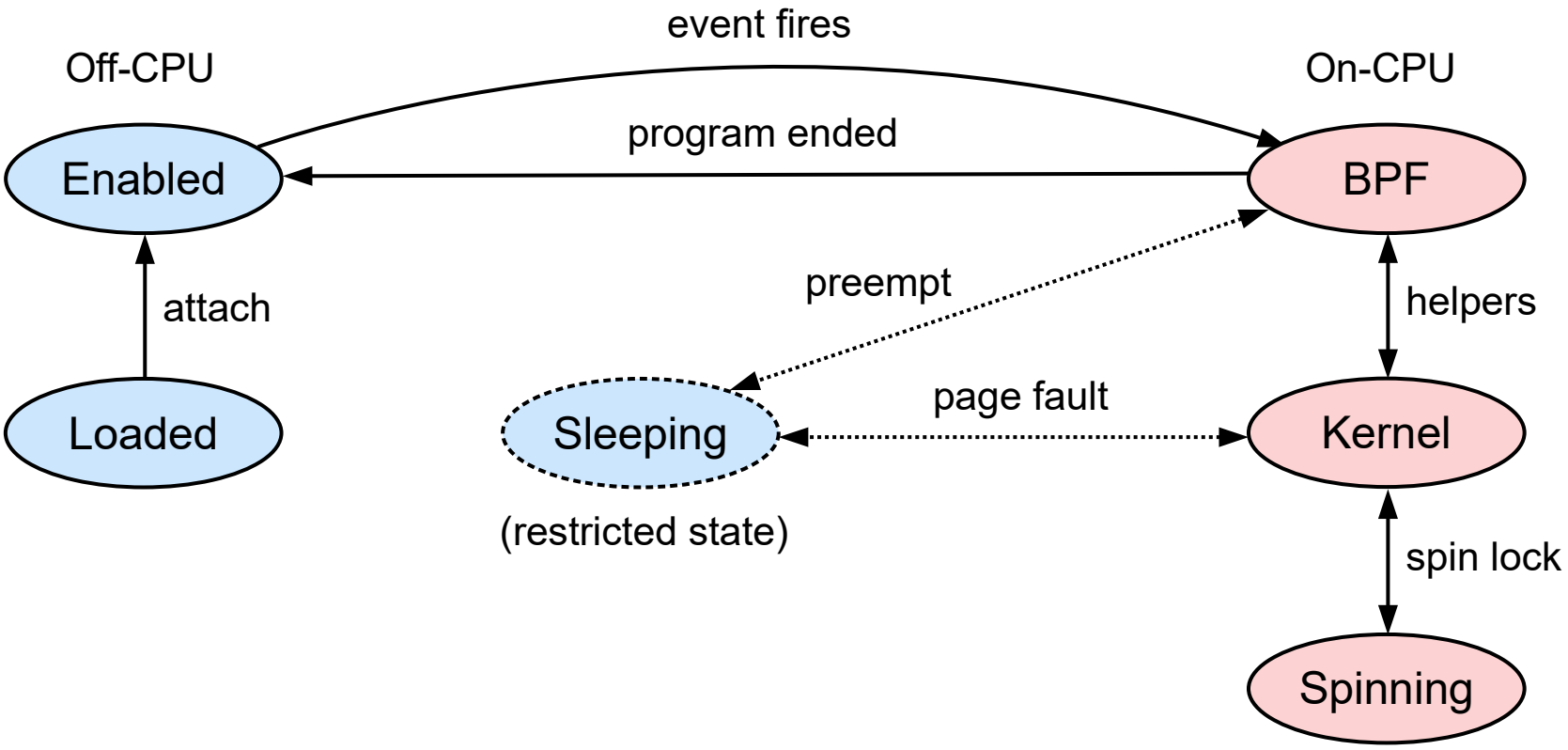
Modern Linux: A new OS model



50 Years, one process state model



BPF program state model



Netconf 2018
Alexei Starvoitov

BPF verifier in the future

- move away from existing brute force "walk all instructions" ap
  technology and static analysis
- remove #define BPF_COMPLEXITY_LIMIT 128k crutch
- remove #define BPF_MAXINSNS 4k
- support arbitrary large programs and libraries
  - 1 Million BPF instructions
- an algorithm to solve Rubik's cube will be expressible in BPF

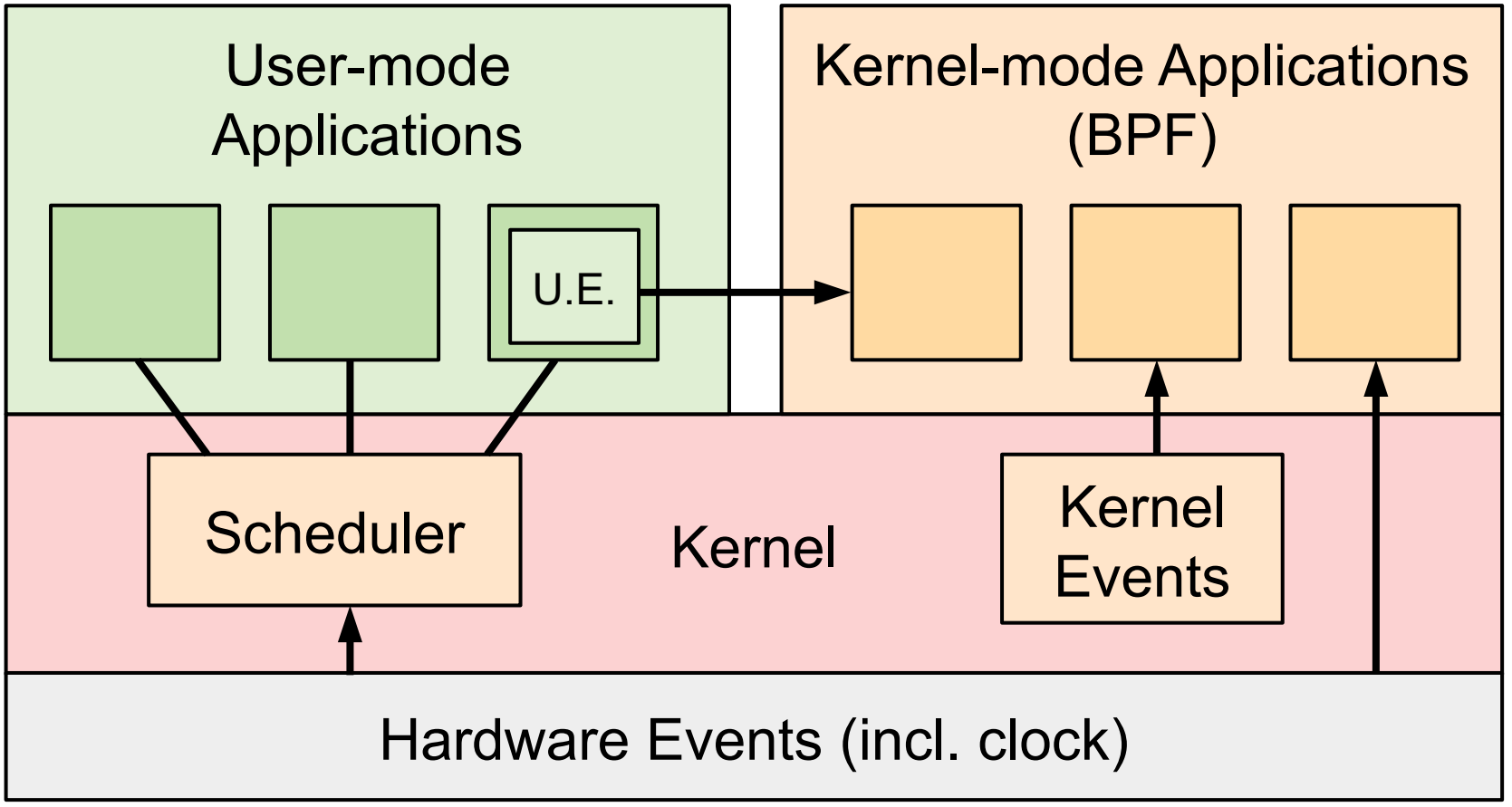
Kernel Recipes 2019, Alexei Starovoitov

~40 active BPF programs on every Facebook server

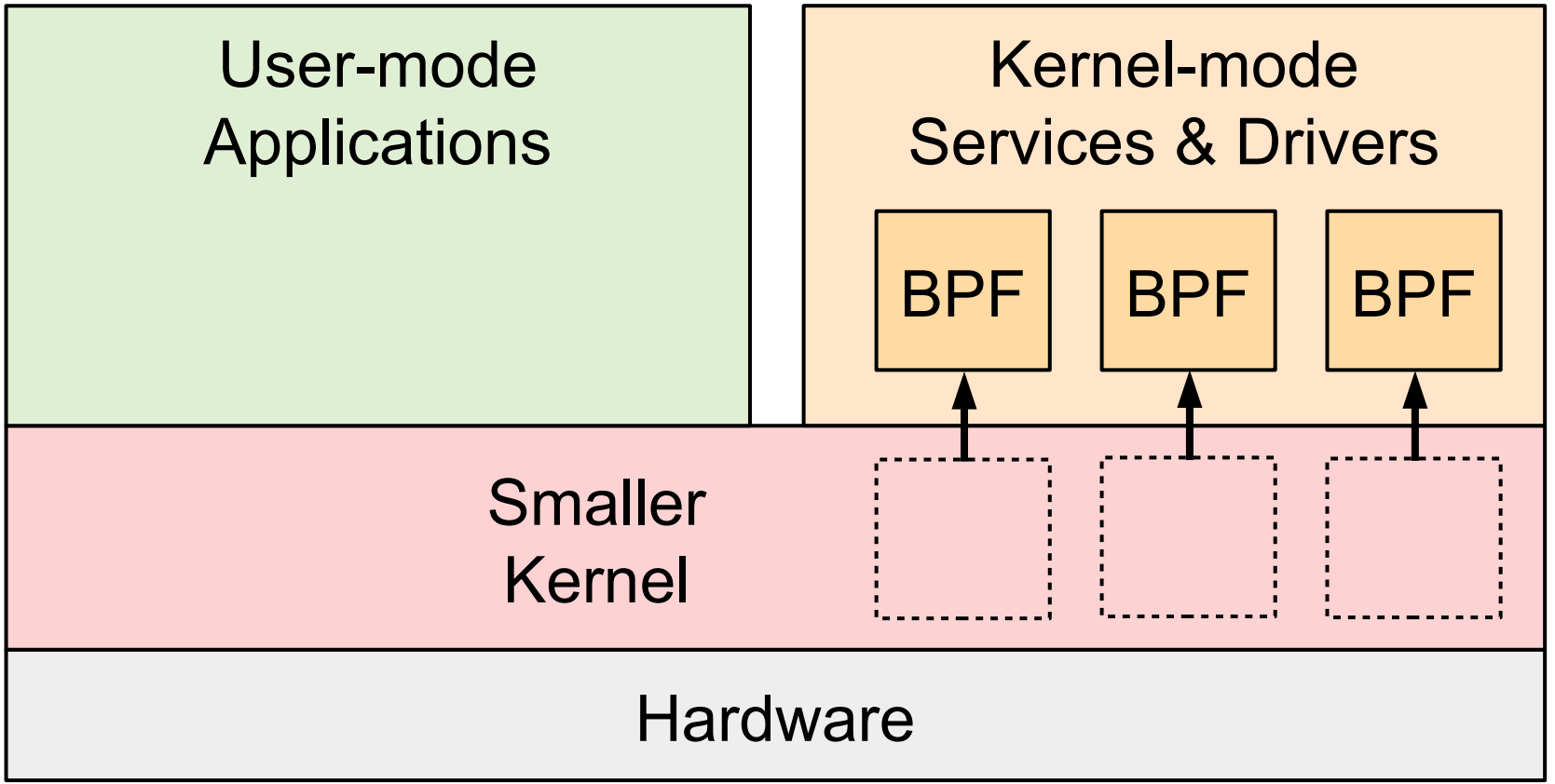
UbuntuMasters 2019, Brendan Gregg

~14 active BPF programs on every Netflix cloud instance

Modern Linux: Event-based Applications



Modern Linux is becoming Microkernel-ish



The word “microkernel” has already been invoked by Jonathan Corbet, Thomas Graf, Greg Kroah-Hartman, ...

Steven Rostedt
@srostedt

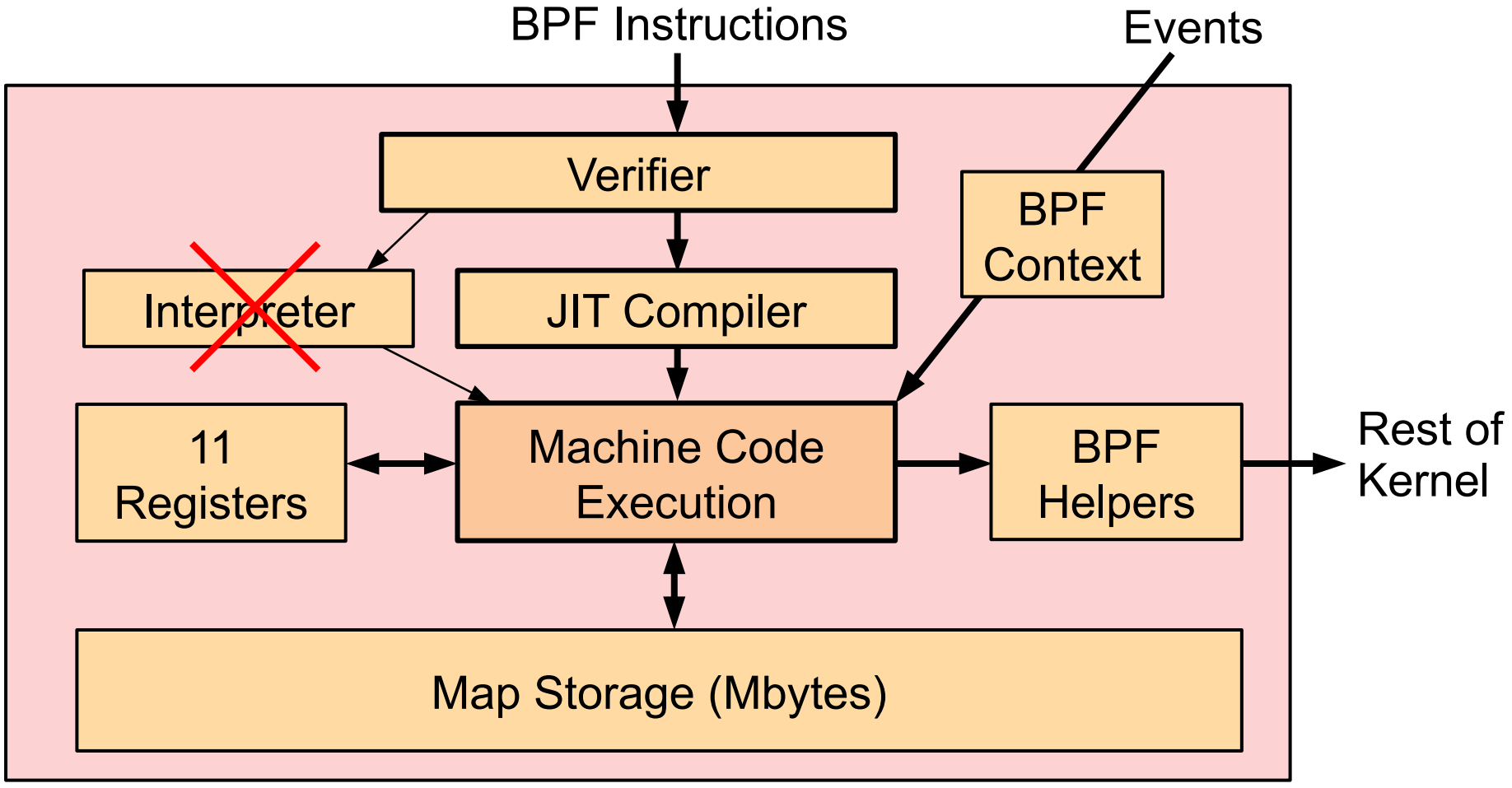
BPF will replace Linux #kr2019

2:06 AM · Sep 26, 2019 · Twitter for Android

18 Retweets **79** Likes

Microsoft are adding eBPF to Windows

BPF Internals




Is BPF Turing complete?

A New Type of Software

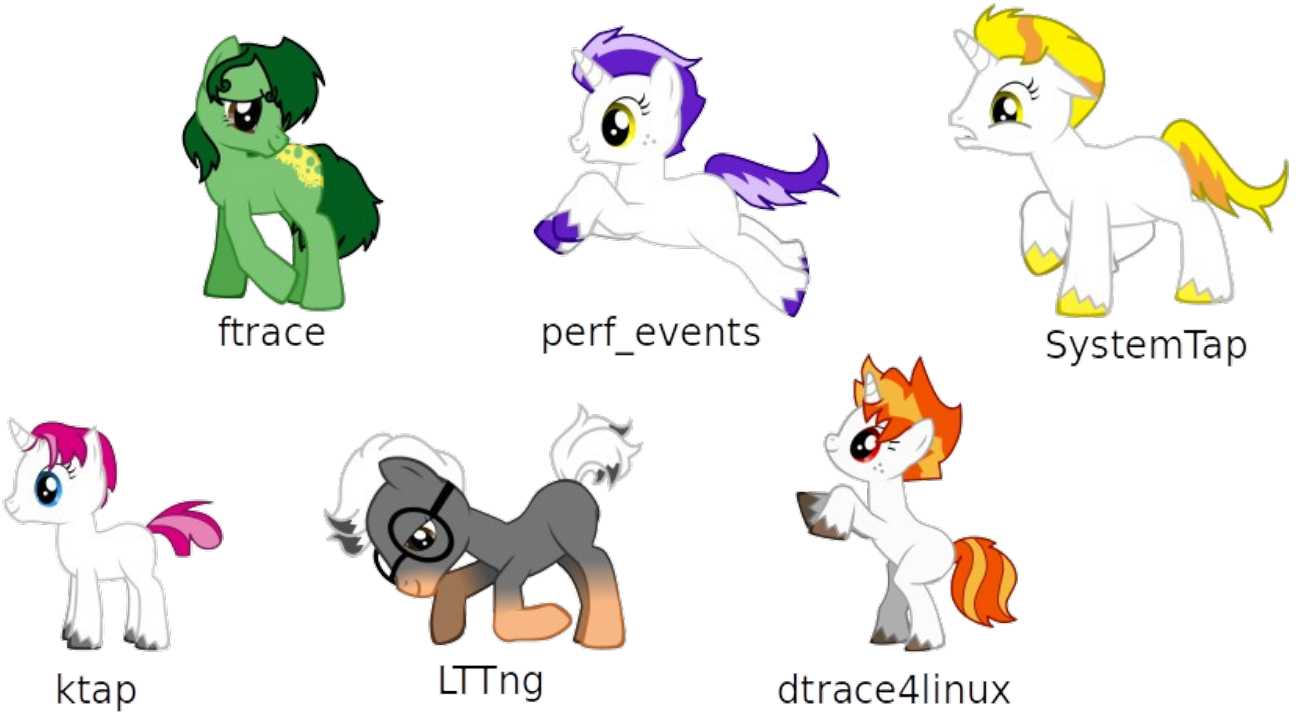
	Execution model	User defined	Compil-ation	Security	Failure mode	Resource access
User	task	yes	any	user based	abort	syscall, fault
Kernel	task	no	static	none	panic	direct
BPF	event	yes	JIT, CO-RE	verified, JIT	error message	restricted helpers

Example Use Case: BPF Observability (aka “Tracing”)

Example Use Case: BPF Observability (aka “Tracing”)

BPF enables a new class of
custom, **efficient**, and **production safe**
performance analysis tools

Linux tracing was a mess (with cute ponies)



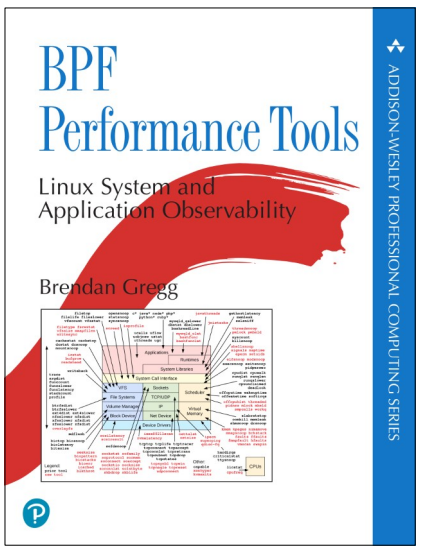
The tracing ponies were created by a tech marketing professional from Sun Microsystems, Deirdre Straughan (now my wife)

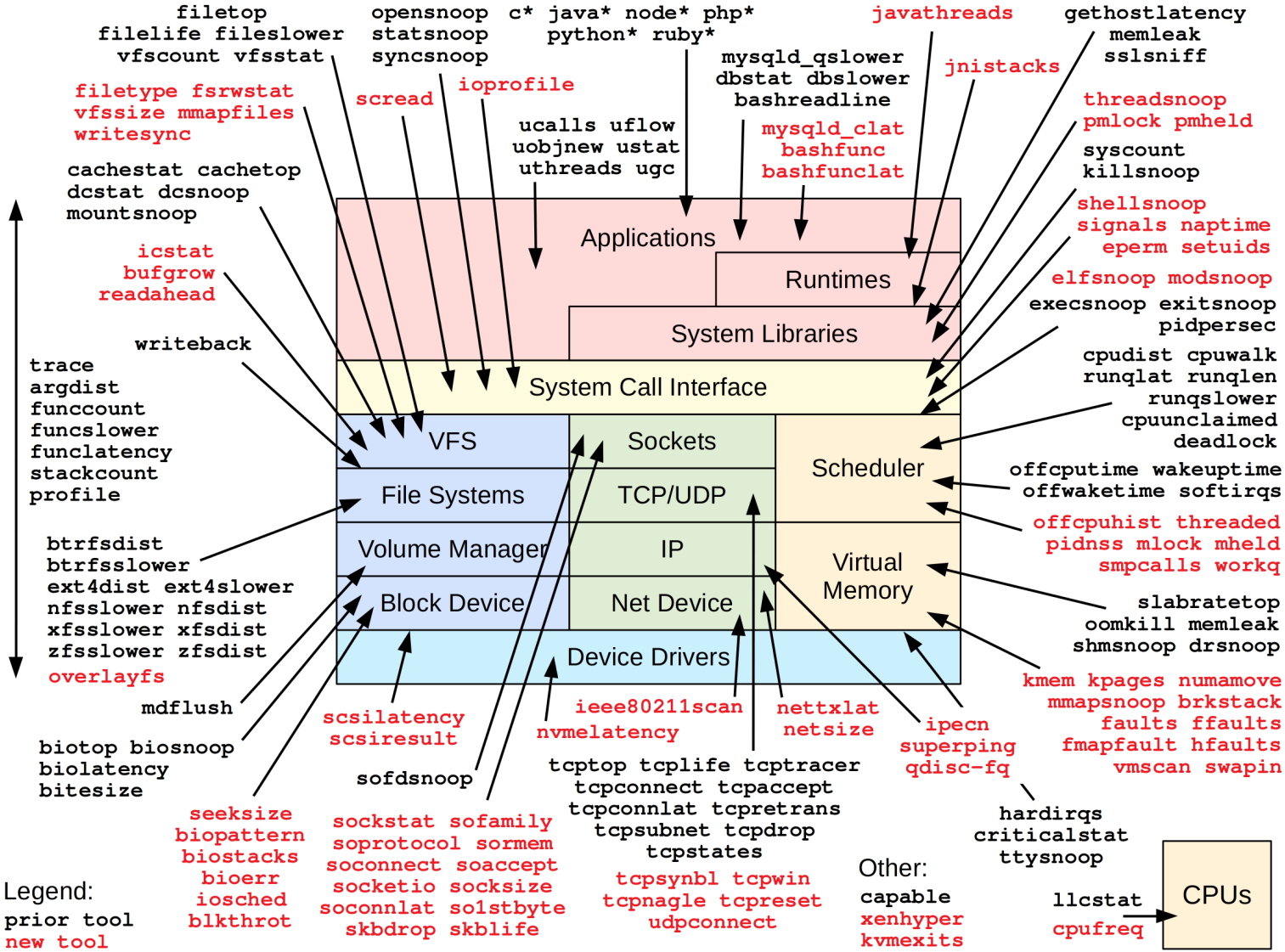
Linux Tracing Present and Future

bcc libbpf

bpftrace

eBPF Perf Tools





Example: BCC tcplife

Which processes are connecting to which port?

Example: BCC tcplife

Which processes are connecting to which port?

```
# ./tcplife
PID   COMM       LADDR           LPORT RADDR           RPORT TX_KB RX_KB MS
22597 recordProg 127.0.0.1       46644 127.0.0.1       28527     0     0 0.23
3277  redis-serv 127.0.0.1       28527 127.0.0.1       46644     0     0 0.28
22598 curl       100.66.3.172    61620 52.205.89.26    80        0     1 91.79
22604 curl       100.66.3.172    44400 52.204.43.121   80        0     1 121.38
22624 recordProg 127.0.0.1       46648 127.0.0.1       28527     0     0 0.22
3277  redis-serv 127.0.0.1       28527 127.0.0.1       46648     0     0 0.27
22647 recordProg 127.0.0.1       46650 127.0.0.1       28527     0     0 0.21
3277  redis-serv 127.0.0.1       28527 127.0.0.1       46650     0     0 0.26
[...]
```

Example: BCC tcplife

```
# tcplife -h
./usage: tcplife.py [-h] [-T] [-t] [-w] [-s] [-p PID] [-L LOCALPORT]
                    [-D REMOTEPORT]

Trace the lifespan of TCP sessions and summarize

optional arguments:
  -h, --help            show this help message and exit
  -T, --time            include time column on output (HH:MM:SS)
  -t, --timestamp       include timestamp on output (seconds)
  -w, --wide            wide column output (fits IPv6 addresses)
  -s, --csv             comma separated values output
  -p PID, --pid PID     trace this PID only
  -L LOCALPORT, --localport LOCALPORT
                        comma-separated list of local ports to trace.
  -D REMOTEPORT, --remoteport REMOTEPORT
                        comma-separated list of remote ports to trace.

examples:
    ./tcplife           # trace all TCP connect()s
    ./tcplife -t        # include time column (HH:MM:SS)
[...]
```

Example: BCC execsnoop

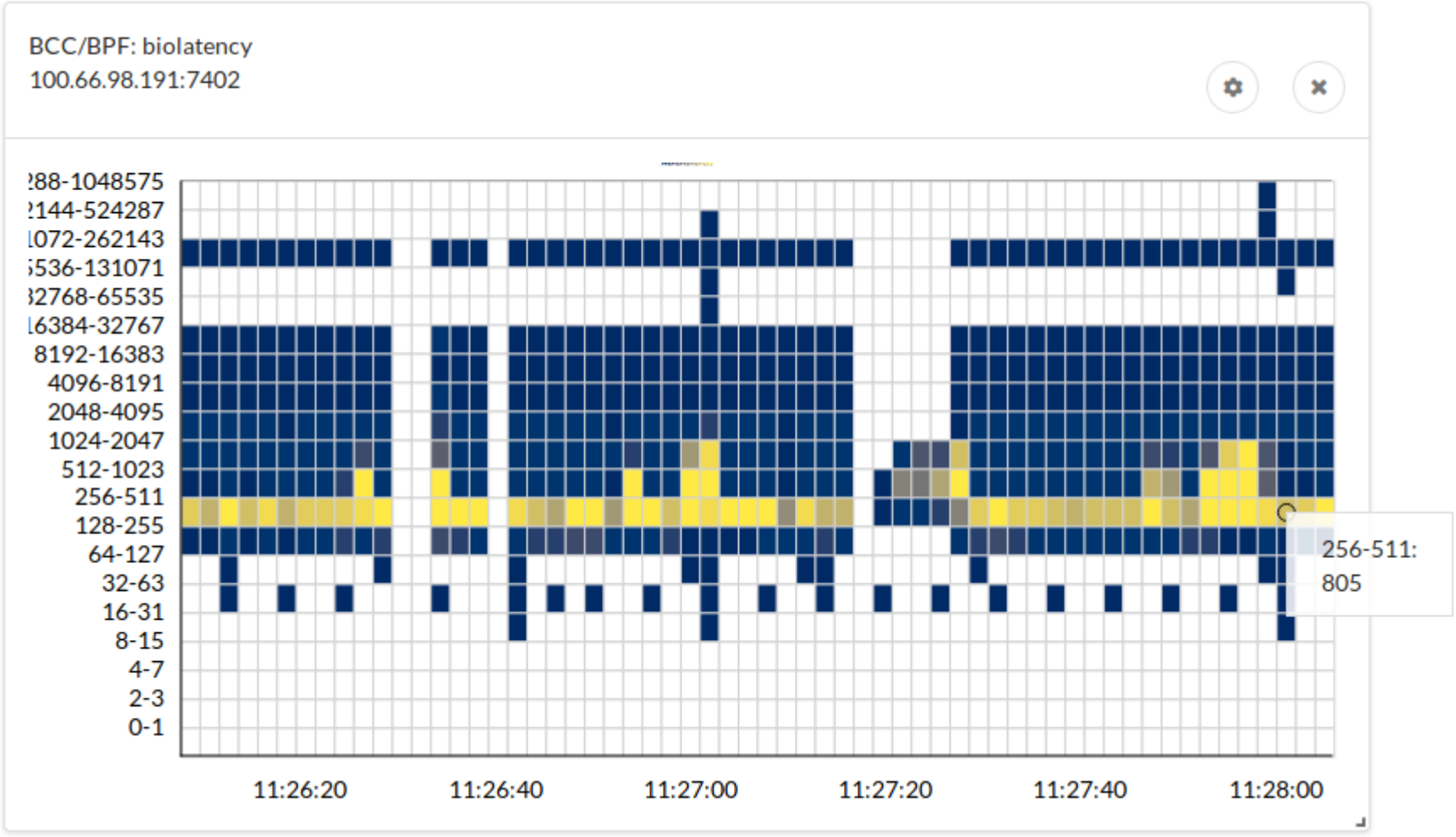
What processes are launching?

```
# execsnoop.py -T
TIME(s) PCOMM            PID    PPID   RET ARGS
0.506   run              8745   1828     0 ./run
0.507   bash             8745   1828     0 /bin/bash
0.511   svstat           8747   8746     0 /command/svstat /service/httpd
0.511   perl             8748   8746     0 /usr/bin/perl -e $l=<>;$l=~/(\d+) sec/;p...
0.514   ps               8750   8749     0 /bin/ps --ppid 1 -o pid,cmd,args
0.514   grep             8751   8749     0 /bin/grep org.apache.catalina
0.514   sed              8752   8749     0 /bin/sed s/^ *//;
0.515   xargs            8754   8749     0 /usr/bin/xargs
0.515   cut              8753   8749     0 /usr/bin/cut -d  -f 1
0.523   echo             8755   8754     0 /bin/echo
0.524   mkdir            8756   8745     0 /bin/mkdir -v -p /data/tomcat
[...]
1.533   svstat           8787   8786     0 /command/svstat /service/httpd
1.533   perl             8788   8786     0 /usr/bin/perl -e $l=<>;$l=~/(\d+) sec/;p...
[...]
```

Example: BCC biolatency

What is the distribution of disk I/O latency? Per second?

```
# ./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        |******                                  |

06:20:17
     msecs               : count     distribution
       0 -> 1            : 96       |****************************************|
       2 -> 3            : 25       |*********                               |
       4 -> 7            : 29       |***********                             |
[...]
```


BCC/BPF: biolatency
100.66.98.191:7402
288-1048575
2144-524287
1072-262143
536-131071
32768-65535
16384-32767
8192-16383
4096-8191
2048-4095
1024-2047
512-1023
256-511
128-255
64-127
32-63
16-31
8-15
4-7
2-3
0-1
11:26:20
11:26:40
11:27:00
11:27:20
11:27:40
11:28:00
256-511:
805

Example: bpftrace readahead

Is readahead polluting the cache?

```
# readahead.bt
Attaching 5 probes...
^C
Readahead unused pages: 128

Readahead used page age (ms):
@age_ms:
[1]                2455 |@@@@@@@@@@@@@@@@                                    |
[2, 4)             8424 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[4, 8)             4417 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@                        |
[8, 16)            7680 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@     |
[16, 32)           4352 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@                        |
[32, 64)              0 |                                                    |
[64, 128)             0 |                                                    |
[128, 256)          384 |@@                                                  |
```

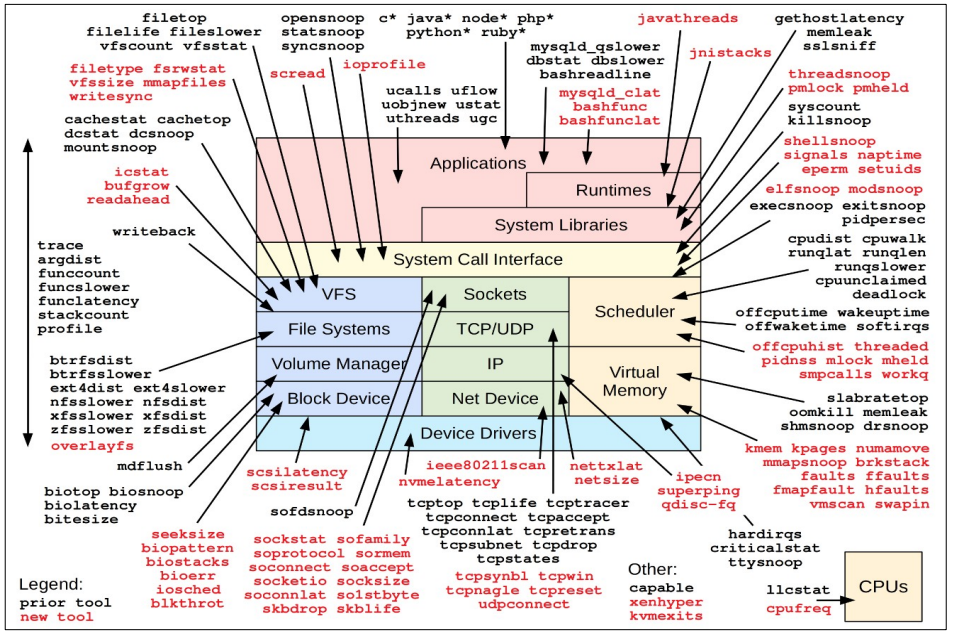
```
#!/usr/local/bin/bpftrace

kprobe:__do_page_cache_readahead    { @in_readahead[tid] = 1; }
kretprobe:__do_page_cache_readahead { @in_readahead[tid] = 0; }

kretprobe:__page_cache_alloc
/@in_readahead[tid]/
{
    @birth[retval] = nsecs;
    @rapages++;
}

kprobe:mark_page_accessed
/@birth[arg0]/
{
    @age_ms = hist((nsecs - @birth[arg0]) / 1000000);
    delete(@birth[arg0]);
    @rapages--;
}

END
{
    printf("\nReadahead unused pages: %d\n", @rapages);
    printf("\nReadahead used page age (ms):\n");
    print(@age_ms); clear(@age_ms);
    clear(@birth); clear(@in_readahead); clear(@rapages);
}
```

Entire readahead.bt bpftrace source

eBPF for Developers

Think like a
sysadmin

Not like a
programmer




```
# define BPF program
prog = """
int hello(void *ctx) {
    bpf_trace_printk("Hello, World!\\n");
    return 0;
}
"""

# load BPF program
b = BPF(text=prog)
b.attach_kprobe(event=b.get_syscall_fnname("clone"), f
```

Think like a **sysadmin**

Get it installed everywhere and use it.

```
# apt-get install bcc-tools bpftrace
# PATH=$PATH:/usr/share/bcc/tools
# execsnoop
# opensnoop
# tcplife
# ext4slower
# biosnoop
[...]
```

These are installed by default at Netflix, Meta, etc.

Think like a **sysadmin**

Get it installed everywhere and use it.

```
# apt-get install bcc-tools bpftrace
# PATH=$PATH:/usr/share/bcc/tools
# execsnoop
# opensnoop
# tcplife
# ext4slower
# biosnoop
[...]
```

Anything periodic running? crontab?

Any misconfigurations? File not found?

Any unexpected TCP sessions?

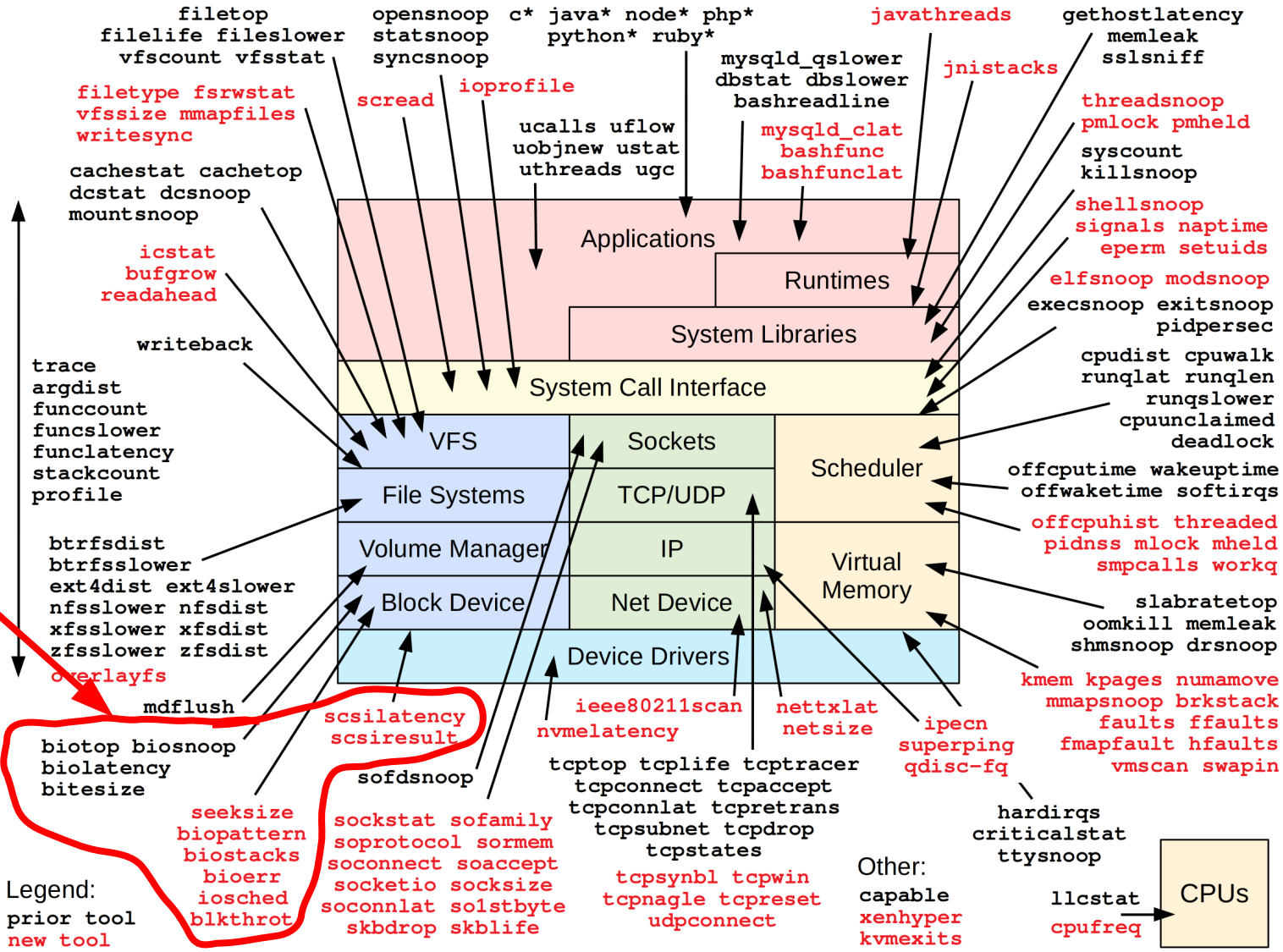
Any file system I/O slower than 10ms?

Any unusual disk access patterns? Outliers?

Print for your office wall, use as a checklist

Suspected disk issue?
Try these first:

https://
www.brendangregg.com/
Perf/bpf_book_tools.png



Recommended tracing front-ends

I want to run some tools

- bcc, bpftrace

I want to hack up some *new* tools

- bpftrace

I want to spend weeks developing a BPF product

- bcc libbpf C, bcc Python (maybe), gobpf, libbpf-rs

New, lightweight,
CO-RE & BTF based

Requires LLVM; becoming
obsolete / special-use only

Recommended tracing front-ends

I want to run some tools — Unix analogies

- bcc, bpftrace — /usr/bin/*

I want to hack up some *new* tools

- bpftrace — bash, awk

I want to spend weeks developing a BPF product

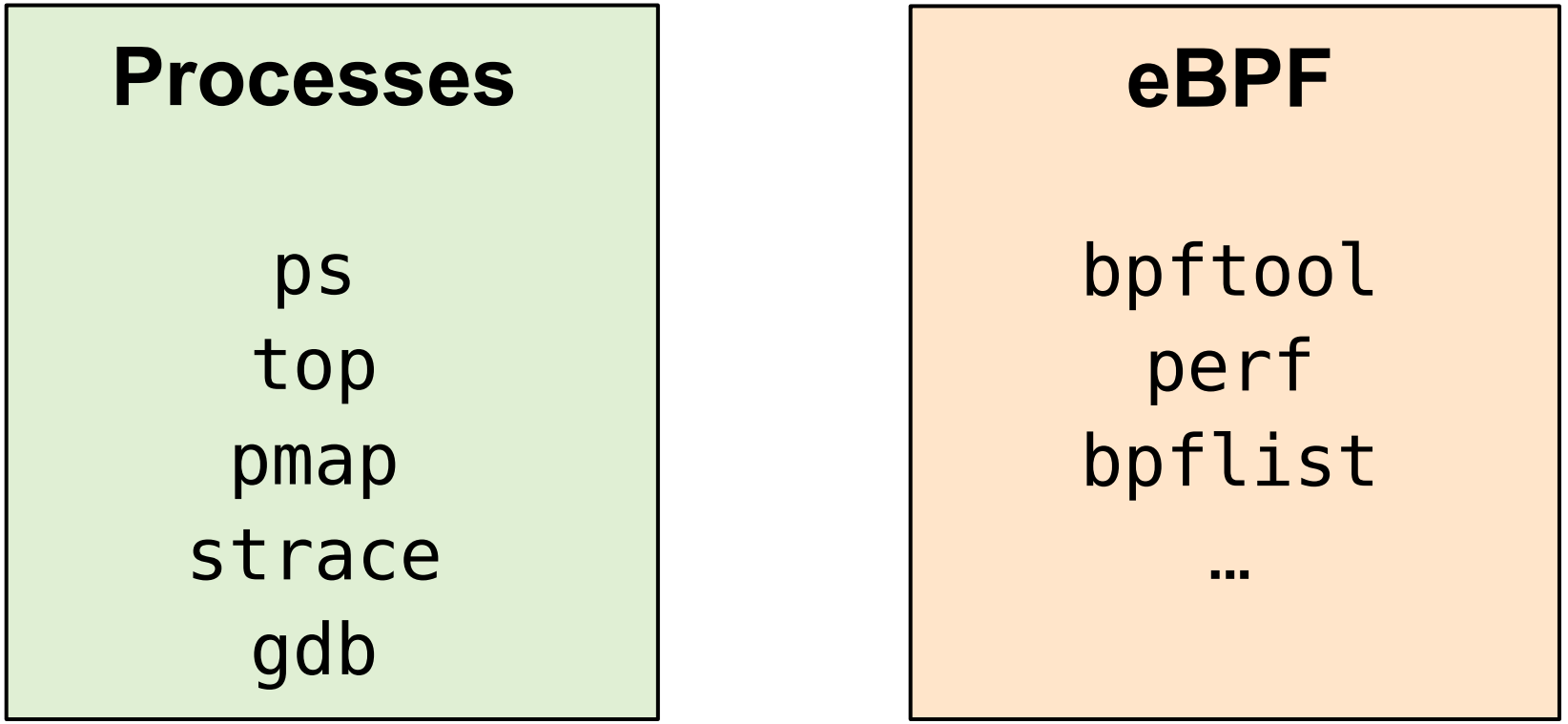
- bcc libbpf C, bcc Python (maybe), gobpf, libbpf-rs — C, C++, Rust

New, lightweight,
CO-RE & BTF based

Requires LLVM; becoming
obsolete / special-use only

Observability *of* eBPF

Observability *of* eBPF (WIP)



bpftool

PID

BPF ID

Event

```
# bpftool perf
pid 1765  fd 6: prog_id 26  kprobe  func blk_account_io_start  offset 0
pid 1765  fd 8: prog_id 27  kprobe  func blk_account_io_done  offset 0
pid 1765  fd 11: prog_id 28  kprobe  func sched_fork  offset 0
pid 1765  fd 15: prog_id 29  kprobe  func ttwu_do_wakeup  offset 0
pid 1765  fd 17: prog_id 30  kprobe  func wake_up_new_task  offset 0
pid 1765  fd 19: prog_id 31  kprobe  func finish_task_switch  offset 0
pid 1765  fd 26: prog_id 33  tracepoint  inet_sock_set_state
pid 21993  fd 6: prog_id 232  uprobe  filename /proc/self/exe  offset 1781927
pid 21993  fd 8: prog_id 233  uprobe  filename /proc/self/exe  offset 1781920
pid 21993  fd 15: prog_id 234  kprobe  func blk_account_io_done  offset 0
pid 21993  fd 17: prog_id 235  kprobe  func blk_account_io_start  offset 0
pid 25440  fd 8: prog_id 262  kprobe  func blk_mq_start_request  offset 0
pid 25440  fd 10: prog_id 263  kprobe  func blk_account_io_done  offset 0
```

```
# bpftool prog dump jited id 263
int trace_req_done(struct pt_regs * ctx):
0xffffffffc082dc6f:
; struct request *req = ctx->di;
   0:   push   %rbp
   1:   mov    %rsp,%rbp
   4:   sub    $0x38,%rsp
   b:   sub    $0x28,%rbp
   f:   mov    %rbx,0x0(%rbp)
  13:   mov    %r13,0x8(%rbp)
  17:   mov    %r14,0x10(%rbp)
  1b:   mov    %r15,0x18(%rbp)
  1f:   xor    %eax,%eax
  21:   mov    %rax,0x20(%rbp)
  25:   mov    0x70(%rdi),%rdi
; struct request *req = ctx->di;
  29:   mov    %rdi,-0x8(%rbp)
; tsp = bpf_map_lookup_elem((void *)bpf_pseudo_fd(1, -1), &req);
  2d:   movabs $0xffff96e680ab0000,%rdi
  37:   mov    %rbp,%rsi
  3a:   add    $0xfffffffffffffff8,%rsi
; tsp = bpf_map_lookup_elem((void *)bpf_pseudo_fd(1, -1), &req);
  3e:   callq  0xffffffffc39a49c1
```

LPC 2019, Arnaldo Carvalho de Melo
CPU profiling of BPF programs

“We should be able to single-step execution...
We should be able to take a core dump of all state.”

– David S. Miller, LSFMM 2019

UNIVAC 1
1951

eBPF Future Predictions

eBPF Future Predictions

More device drivers, incl. USB on BPF

Monitoring agents

Intrusion detection systems

TCP congestion controls

CPU & container schedulers

FS readahead policies

CDN accelerator

Reality Check

More perf wins come from CPU flame graphs
not CLI tracing

3. eBPF Flame Graphs

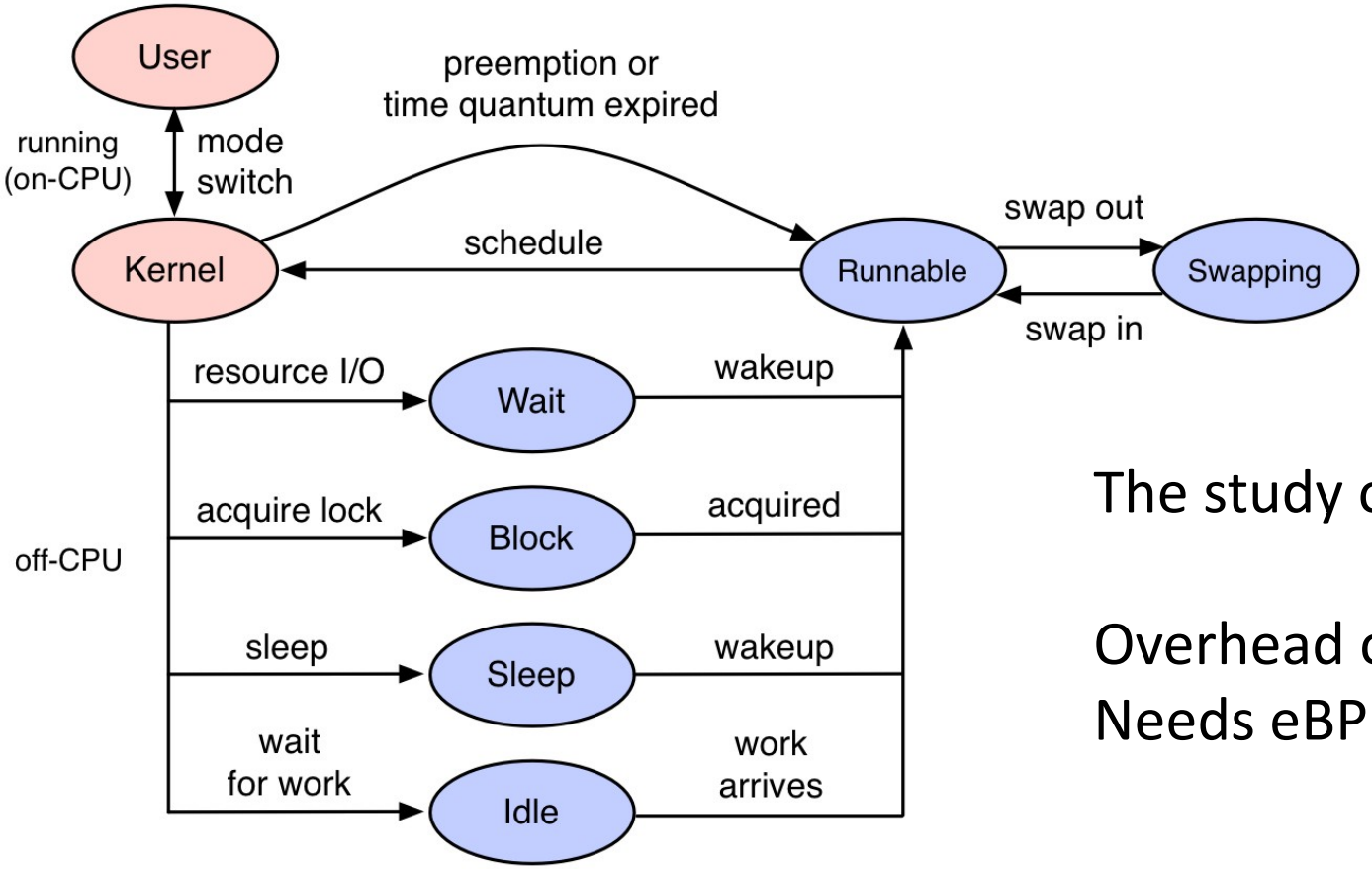
Linux eBPF CPU Flame Graphs

```
# profile -af -F 49 30 | ./flamegraph.pl > out.svg
```

Lower overhead profiling

Make continuous profiling more practical

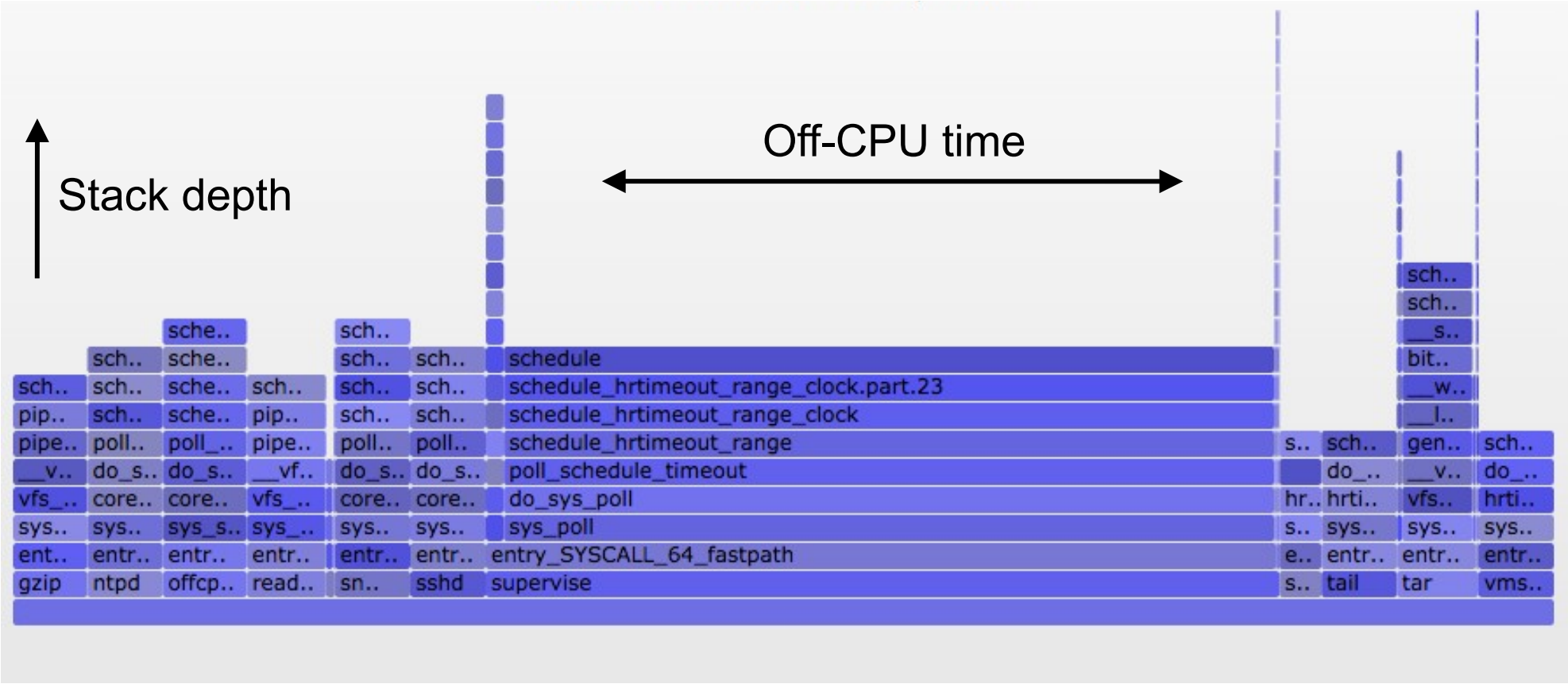
Off-CPU Analysis



The study of blocking states

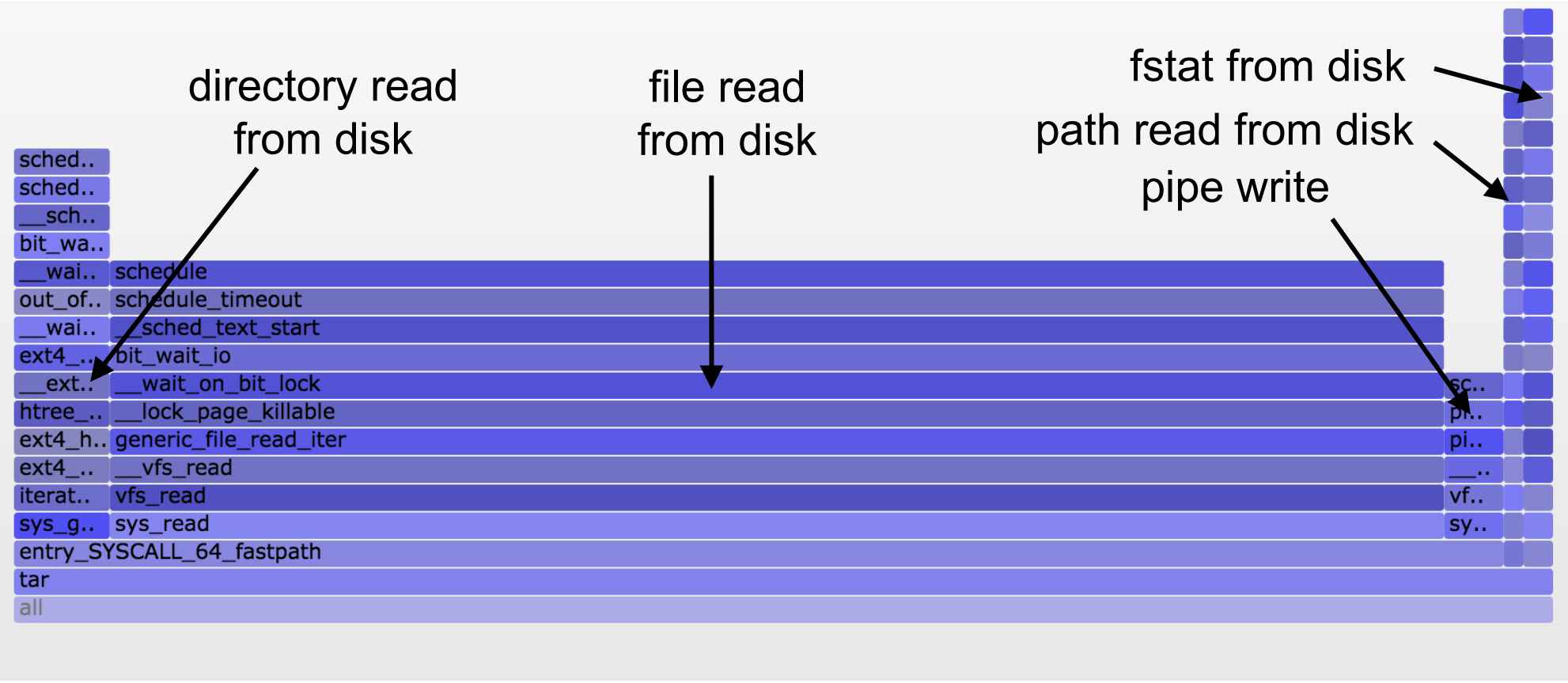
Overhead often prohibitive
Needs eBPF!

Off-CPU Time Flame Graph



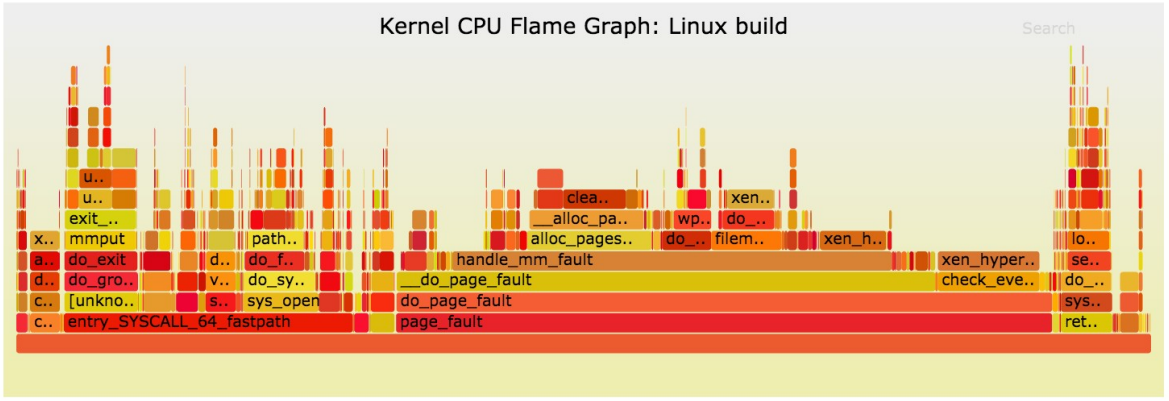
http://www.brendangregg.com/blog/2016-02-01/linux-wakeup-offwake-profiling.html

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

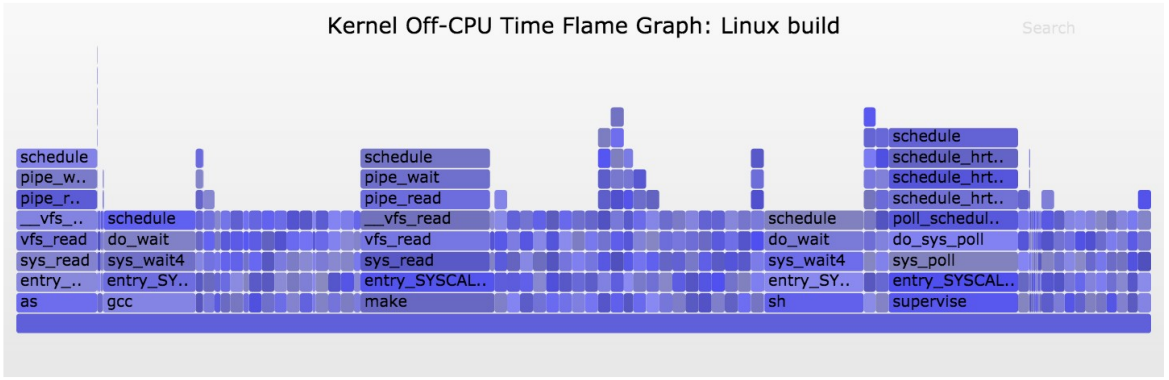


Only showing kernel stacks in this example

CPU + Off-CPU Flame Graphs: See Everything

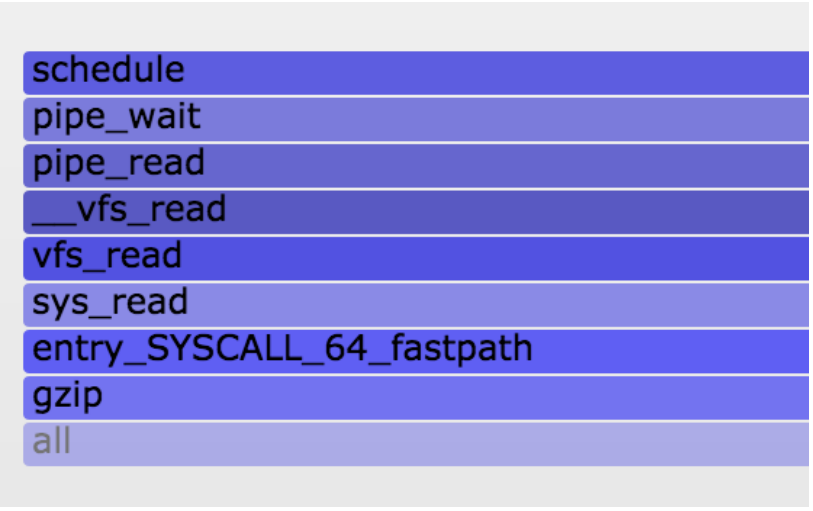


CPU



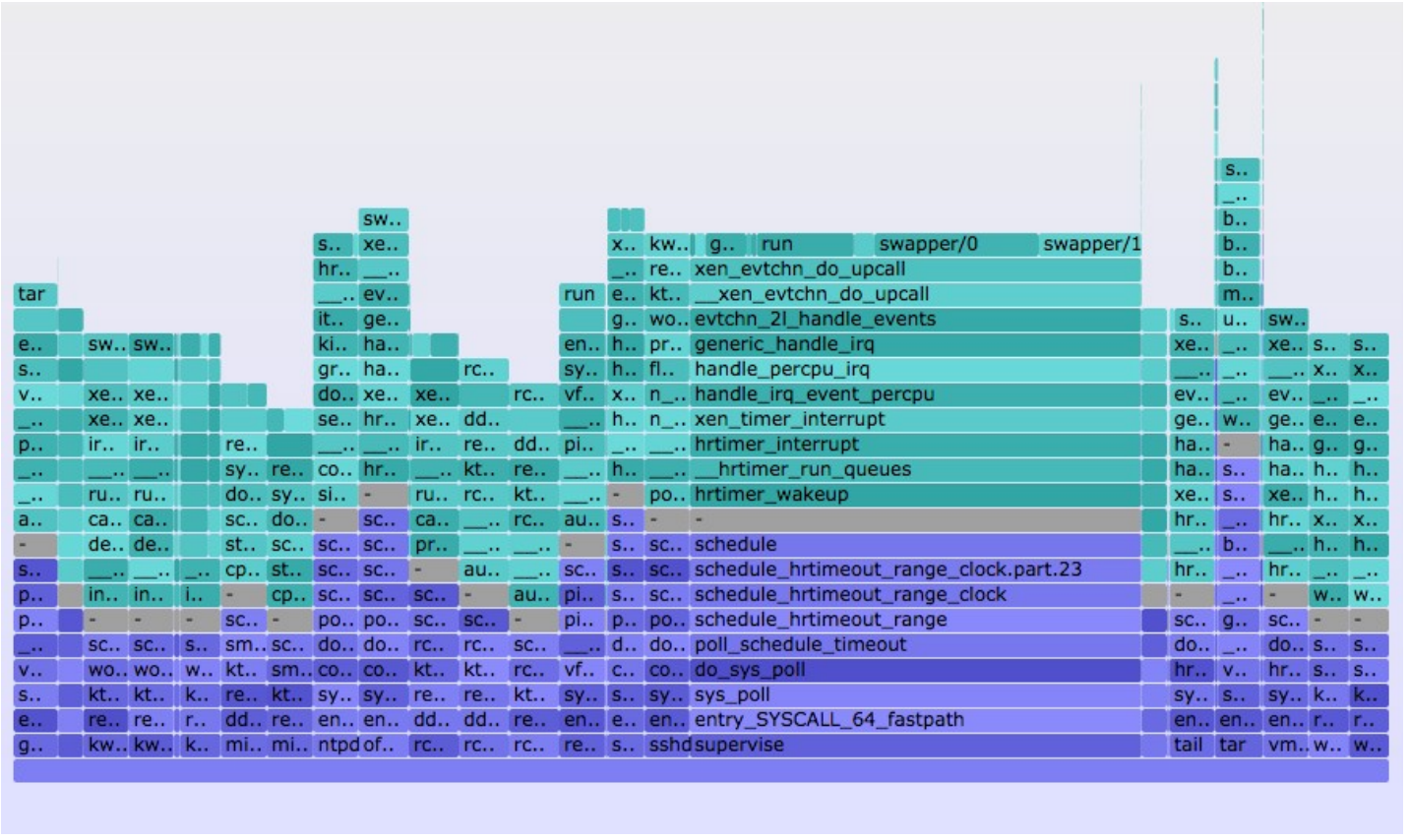
Off-CPU

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



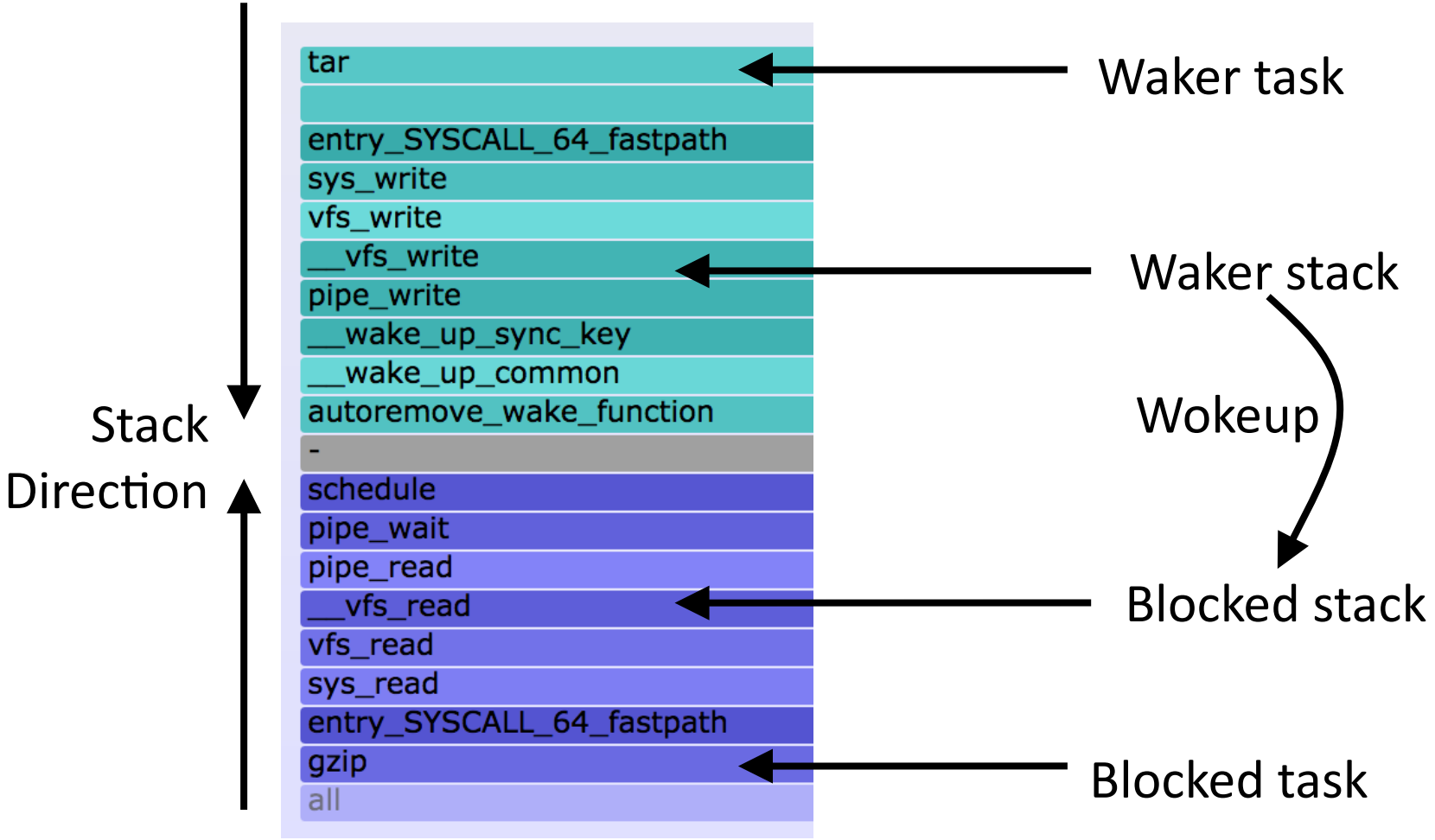
The off-CPU stack trace often doesn't show the root cause of latency.
What is gzip blocked on?

Off-Wake Time Flame Graph



Uses eBPF to merge off-CPU and waker stack in kernel context

Off-Wake Time Flame Graph (zoomed)



eBPF Flame Graph Futures

Practical off-CPU flame graphs

Other types: disk, network, malloc, etc.

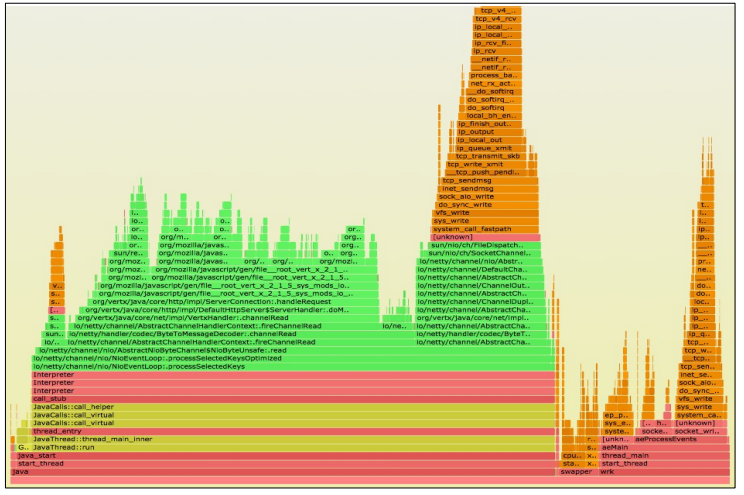
Custom stack walking

- Frame pointers not needed
- Include other app context

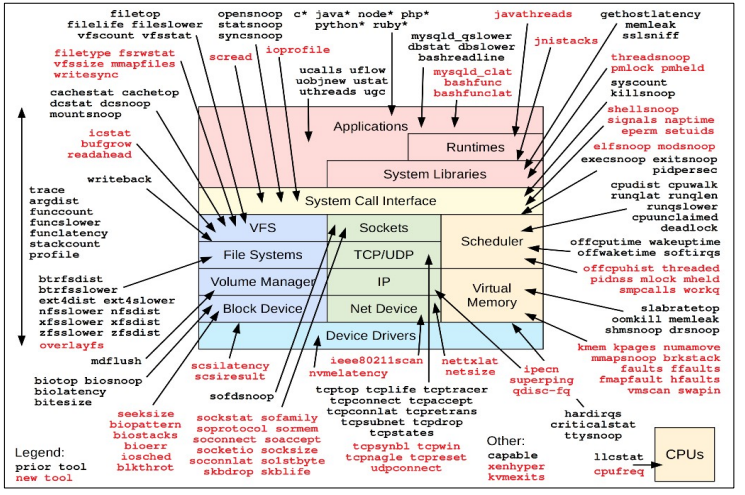
And/or frame pointers everywhere

Take Aways

Print these for your office wall:

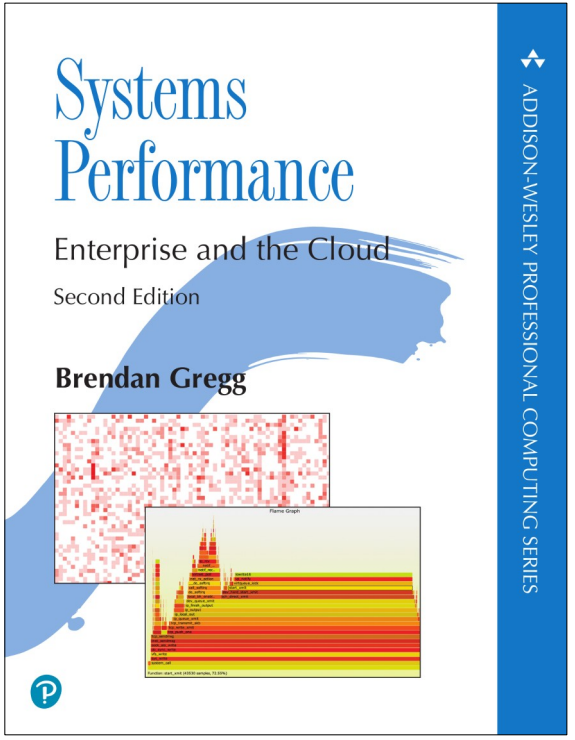


A CPU flame graph of your software



Open source eBPF tools you can run

Books



Volume 1

Volume 2

References/URLs

"The Flame Graph" Communications of the ACM, Vol. 56, No. 6 (June 2016)
http://queue.acm.org/detail.cfm?id=2927301
http://www.brendangregg.com/flamegraphs.html -> http://www.brendangregg.com/flamegraphs.html#Updates
http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html
http://www.brendangregg.com/FlameGraphs/memoryflamegraphs.html
http://www.brendangregg.com/FlameGraphs/offcpuflamegraphs.html
https://github.com/spiermar/d3-flame-graph
http://techblog.netflix.com/2015/07/java-in-flames.html
http://techblog.netflix.com/2016/04/saving-13-million-computational-minutes.html
http://techblog.netflix.com/2014/11/nodejs-in-flames.html
http://www.brendangregg.com/blog/2014-11-09/differential-flame-graphs.html
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
http://corpaul.github.io/flamegraphdiff/
https://perf.wiki.kernel.org/index.php/Main_Page
http://www.brendangregg.com/perf.html
https://github.com/iovisor/bcc
https://github.com/iovisor/bpftrace

Thanks

YOW! Organizers, especially Dave and Aino

BPF: Alexei Starovoitov (Facebook), Daniel Borkmann (Isovalent), David S. Miller (Red Hat), Jakub Kicinski (Facebook), Yonghong Song (Facebook), Martin KaFai Lau (Facebook), John Fastabend (Isovalent), Quentin Monnet (Isovalent), Jesper Dangaard Brouer (Red Hat), Andrey Ignatov (Facebook), Stanislav Fomichev (Google), Linus Torvalds, and many more in the BPF community

BCC: Brenden Blanco (VMware), Yonghong Song, Sasha Goldsthein (Google), Teng Qin (Facebook), Paul Chaignon (Isovalent), Vicent Martí (PlanetScale), Dave Marchevsky (Facebook), Hengqi Chen (Tencent), and many more in the BCC community

bpftrace: Alastair Robertson (Facebook), Dan Xu (Facebook), Bas Smit, Mary Marchini (Netflix), Masanori Misono, Jiri Olsa, Viktor Malík, Dale Hamel, Willian Gaspar, Augusto Mecking Caringi, and many more in the bpftrace community

Flame graphs/charts: Adrien Mahieux, Ilya Tikhonovsky, Martin Spier, Perl/d3 contributors

Canonical: BPF support, and libc-prof

brendan@intel.com

All photos credit myself; except slide 42 (KernelRecipes) and 43 (UbuntuMasters)