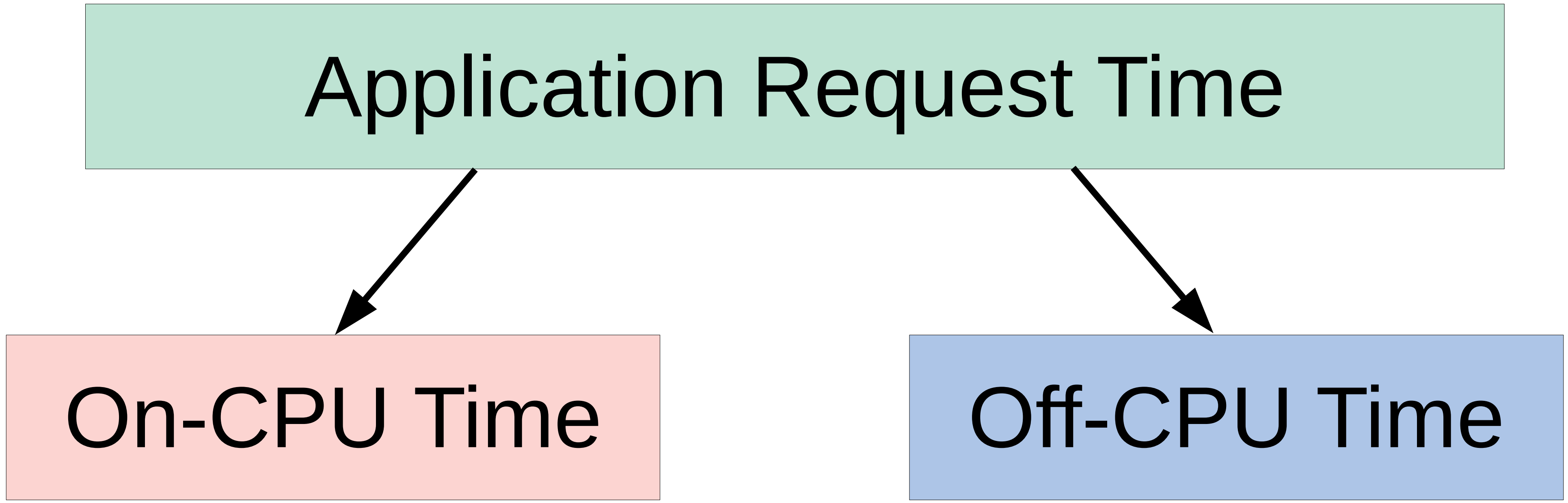
Sep, 2020

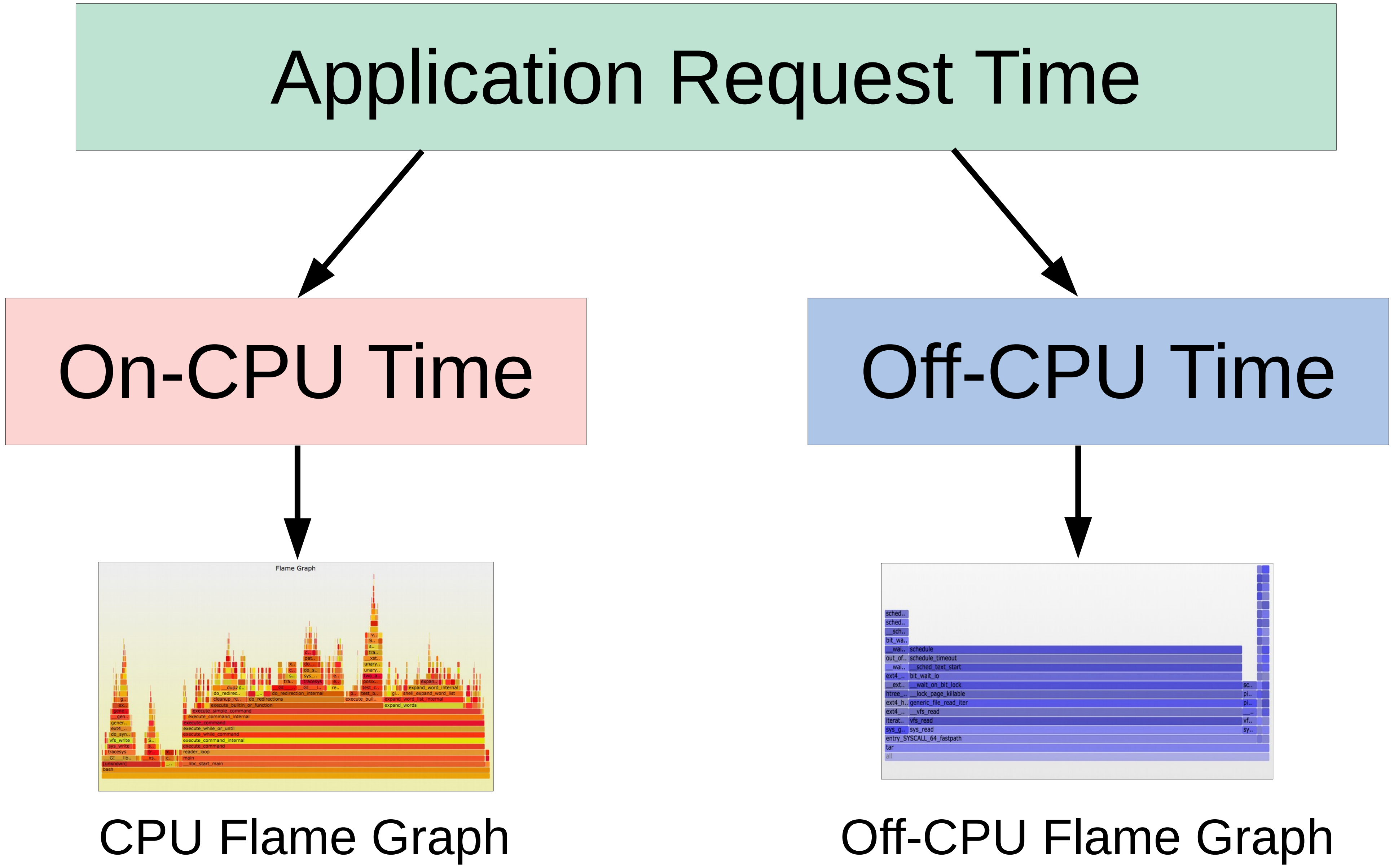
# Linux Systems Performance

Brendan Gregg
*Senior Performance Engineer*
NETFLIX

YOW! Auckland, Perth, Singapore, Hong Kong

# What I’m currently working on


Application Request Time
On-CPU Time
Off-CPU Time


Application Request Time
On-CPU Time
Off-CPU Time
CPU Flame Graph
Off-CPU Flame Graph

```
7fffc102ca03 nf_conntrack_in ([kernel.kallsyms])
7fffc10d341c ipv4_conntrack_local ([kernel.kallsyms])
7fff9deb09d8 nf_hook_slow ([kernel.kallsyms])
7fff9debe6c7 __ip_local_out ([kernel.kallsyms])
7fff9debe74c ip_local_out ([kernel.kallsyms])
7fff9debeab0 ip_queue_xmit ([kernel.kallsyms])
7fff9ded8e02 __tcp_transmit_skb ([kernel.kallsyms])
7fff9deda3f4 tcp_write_xmit ([kernel.kallsyms])
7fff9dedb215 __tcp_push_pending_frames ([kernel.kallsyms])
7fff9dec684b tcp_push ([kernel.kallsyms])
7fff9deca337 tcp_sendmsg_locked ([kernel.kallsyms])
7fff9decae5c tcp_sendmsg ([kernel.kallsyms])
7fff9defa8ee inet_sendmsg ([kernel.kallsyms])
7fff9de4a41e sock_sendmsg ([kernel.kallsyms])
7fff9de4a9af SYSC_sendto ([kernel.kallsyms])
7fff9de4b49e sys_sendto ([kernel.kallsyms])
7fff9d605bb3 do_syscall_64 ([kernel.kallsyms])
7fff9e002081 entry_SYSCALL_64_after_hwframe ([kernel.kallsyms])
       119ae __libc_send (/lib/x86_64-linux-gnu/libpthread-2.27.so)
```

X

Off-CPU stacks often end abruptly in libc

# More than one way to walk a stack

Frame pointers

Last branch record (LBR)

Branch trace store (BTS)

DWARF

ORC

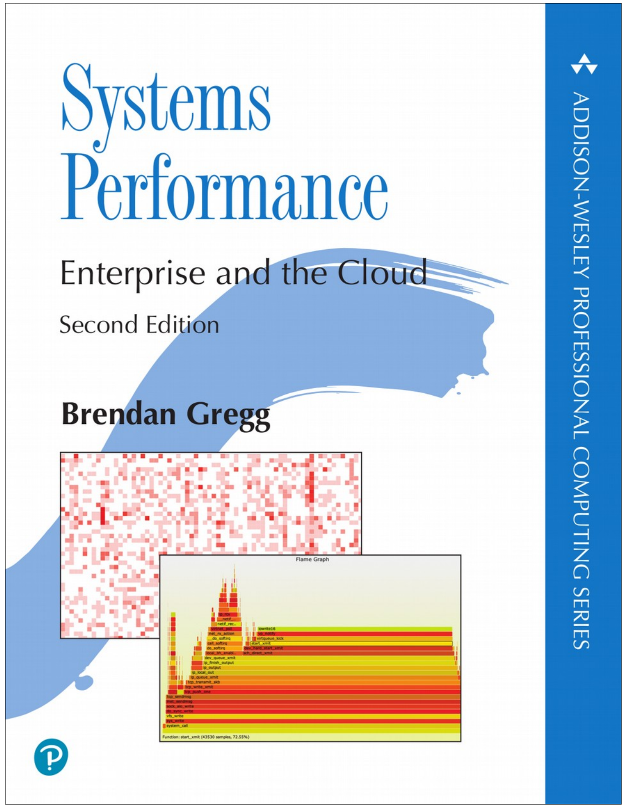
Application exception handler

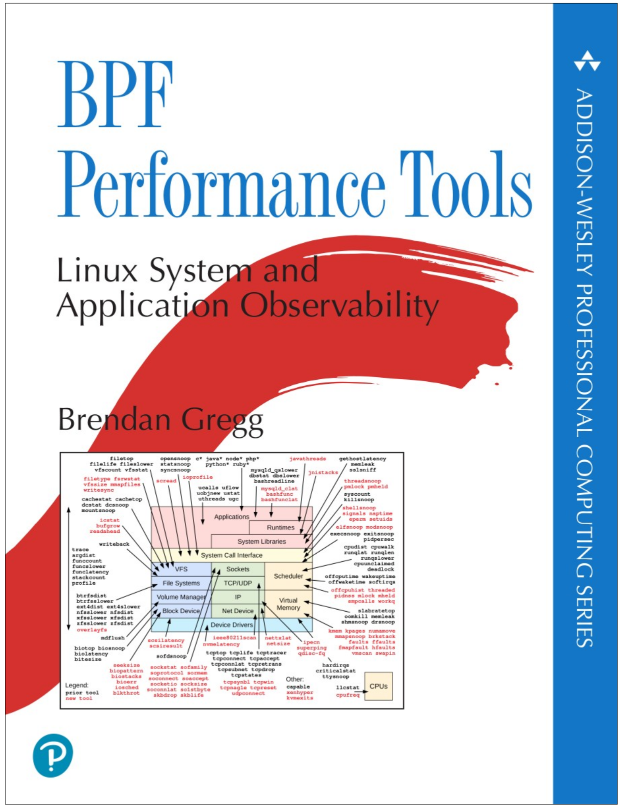
...

We’re currently rolling out
our own build of libc
with frame pointers
(-fno-omit-frame-pointer)

# Systems Performance in 45 mins

- This is slides + discussion
- For more detail and stand-alone texts:

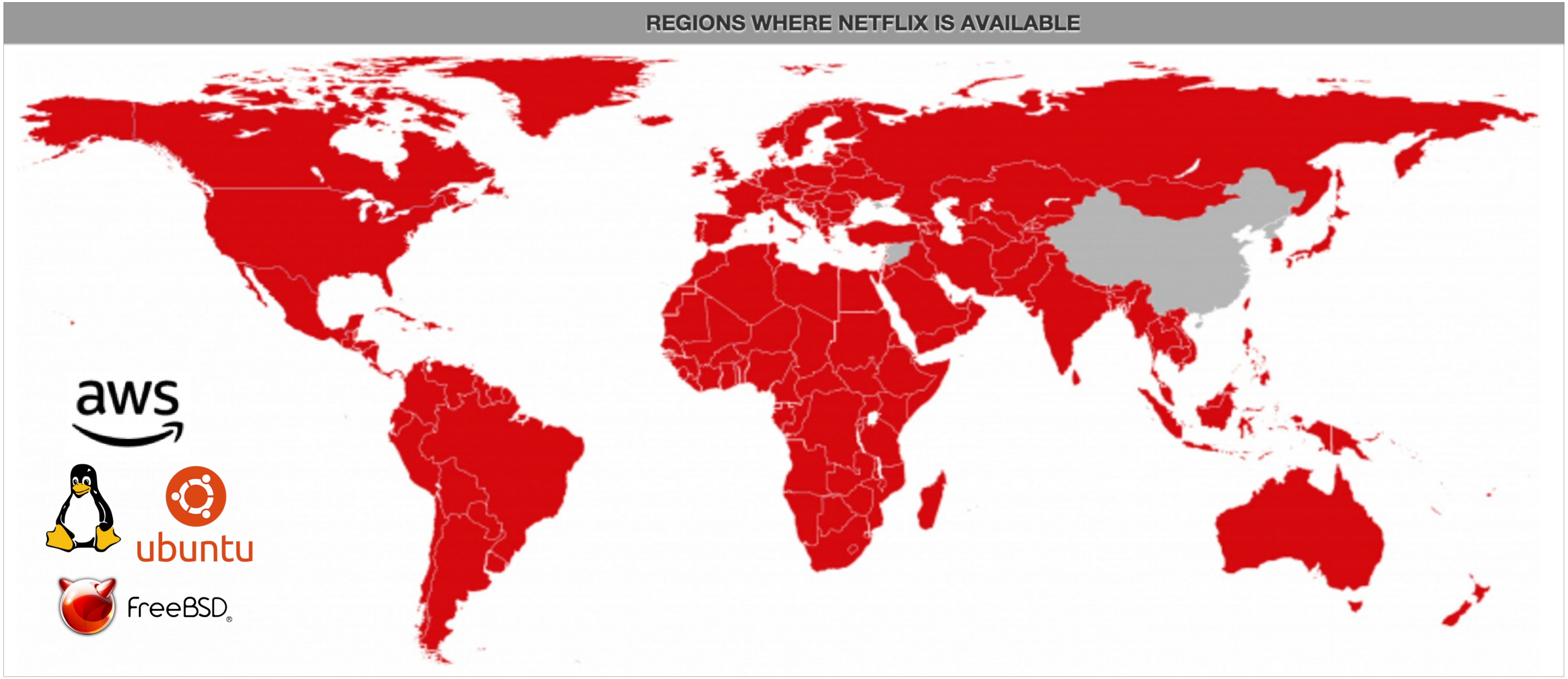


# Agenda

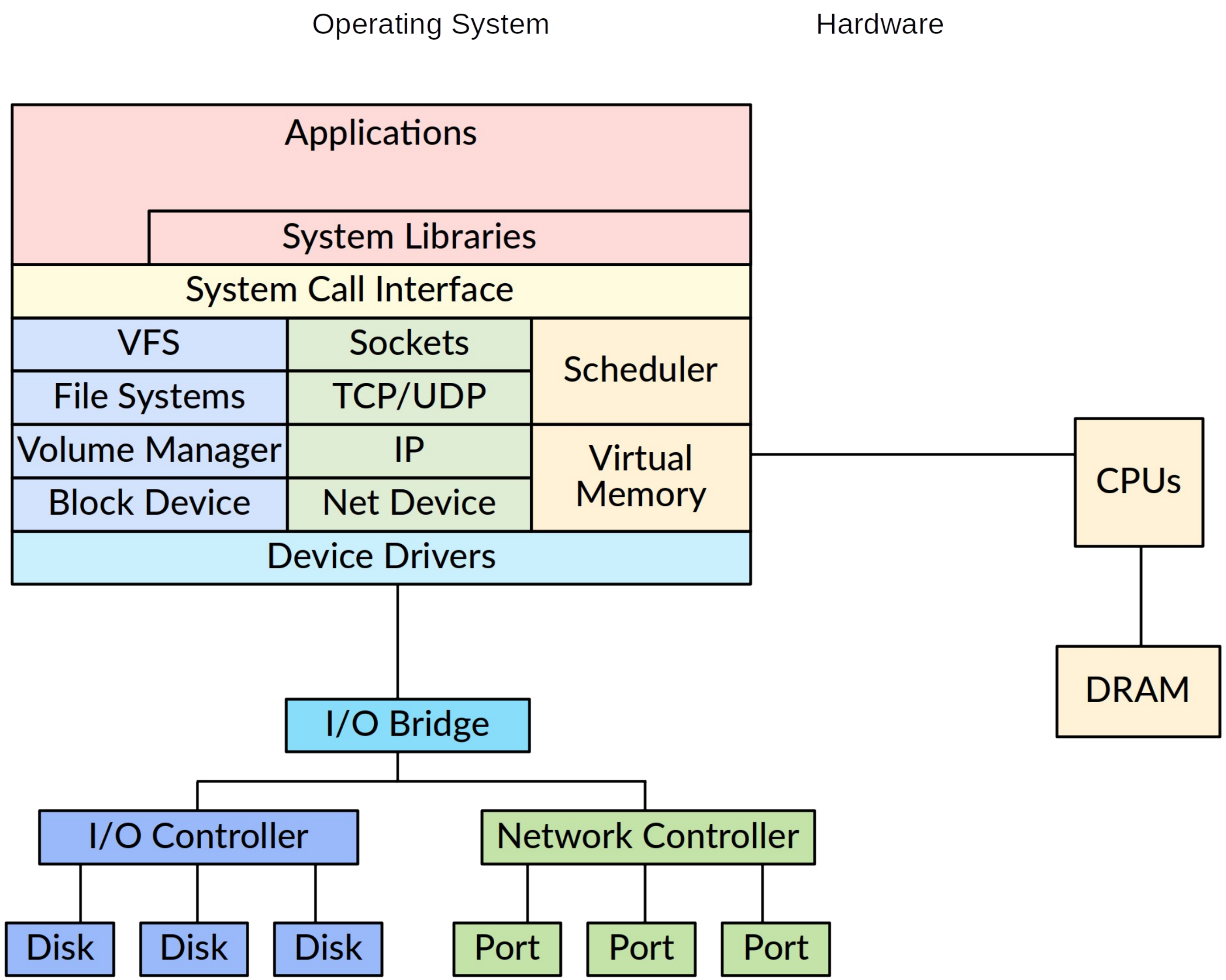
1. Observability
2. Methodologies
3. Benchmarking
4. Profiling
5. Tracing
6. Tuning

# NETFLIX

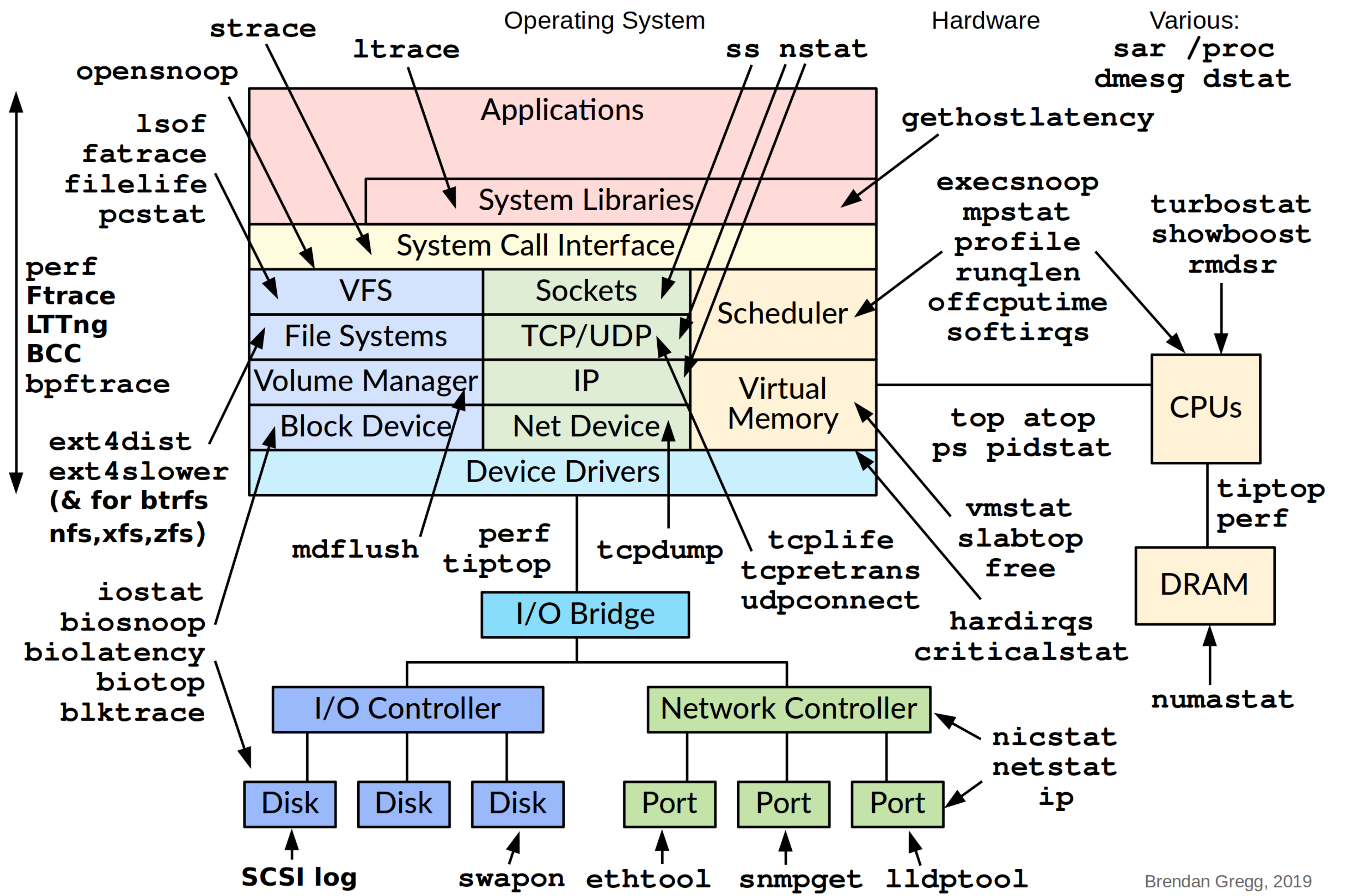


# 1. Observability

# How do you measure these?



# Linux Observability Tools



# Why Learn Tools?

- Most analysis at Netflix is via GUIs
- Benefits of command-line tools:
  - Helps you understand GUIs: they show the same metrics
  - Often documented, unlike GUI metrics
  - Often have useful options not exposed in GUIs
- Installing essential tools (something like):

```
$ sudo apt-get install sysstat bcc-tools bpftrace linux-tools-common \
    linux-tools-$(uname -r) iproute2 msr-tools
$ git clone https://github.com/brendangregg/msr-cloud-tools
$ git clone https://github.com/brendangregg/bpf-perf-tools-book
```

These are crisis tools and should be installed by default

In a performance meltdown you may be unable to install them

# uptime

- One way to print *load averages*:

```
$ uptime
 07:42:06 up  8:16,  1 user,  load average: 2.27, 2.84, 2.91
```

- A measure of resource demand: CPUs + disks
  - Includes TASK_UNINTERRUPTIBLE state to show all demand types
  - You can use BPF & off-CPU flame graphs to explain this state:
    http://www.brendangregg.com/blog/2017-08-08/linux-load-averages.html
  - PSI in Linux 4.20 shows CPU, I/O, and memory loads
- Exponentially-damped moving averages
  - With time constants of 1, 5, and 15 minutes. See historic trend.
- Load > # of CPUs, may mean CPU saturation

Don't spend more than 5 seconds studying these

# top

- System and per-process interval summary:

```
$ top - 18:50:26 up  7:43,  1 user,  load average: 4.11, 4.91, 5.22
Tasks: 209 total,   1 running, 206 sleeping,   0 stopped,   2 zombie
Cpu(s): 47.1%us, 4.0%sy, 0.0%ni, 48.4%id, 0.0%wa, 0.0%hi, 0.3%si, 0.2%st
Mem:  70197156k total, 44831072k used, 25366084k free,    36360k buffers
Swap:        0k total,        0k used,        0k free, 11873356k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND

 5738 apiprod   20   0 62.6g  29g 352m S  417 44.2   2144:15 java
 1386 apiprod   20   0 17452 1388  964 R    0  0.0   0:00.02 top
    1 root      20   0 24340 2272 1340 S    0  0.0   0:01.51 init
    2 root      20   0     0    0    0 S    0  0.0   0:00.00 kthreadd
[…]
```

- %CPU is summed across all CPUs
- Can miss short-lived processes (atop won't)

# htop

```
$ htop
  1  [||||||||||70.0%]    13 [||||||||||70.6%]   25 [||||||||||69.7%]    37 [||||||||||66.6%]
  2  [||||||||||68.7%]    14 [||||||||||69.4%]   26 [||||||||||67.7%]    38 [||||||||||66.0%]
  3  [||||||||||68.2%]    15 [||||||||||68.5%]   27 [||||||||||68.8%]    39 [||||||||||73.3%]
  4  [||||||||||69.3%]    16 [||||||||||69.2%]   28 [||||||||||67.6%]    40 [||||||||||67.0%]
  5  [||||||||||68.0%]    17 [||||||||||67.6%]   29 [||||||||||70.1%]    41 [||||||||||66.5%]
[…]
  Mem[||||||||||||||||||||||||||||||176G/187G]   Tasks: 80, 3206 thr; 43 running
  Swp[                                0K/0K]   Load average: 36.95 37.19 38.29
                                               Uptime: 01:39:36

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 4067 www-data   20   0  202G  173G 55392 S 3359 93.0 48h51:30 /apps/java/bin/java -Dnop -Djdk.map
 6817 www-data   20   0  202G  173G 55392 R 56.9 93.0 48:37.89 /apps/java/bin/java -Dnop -Djdk.map
 6826 www-data   20   0  202G  173G 55392 R 25.7 93.0 22:26.90 /apps/java/bin/java -Dnop -Djdk.map
 6721 www-data   20   0  202G  173G 55392 S 25.0 93.0 22:05.51 /apps/java/bin/java -Dnop -Djdk.map
 6616 www-data   20   0  202G  173G 55392 S 13.6 93.0 11:15.51 /apps/java/bin/java -Dnop -Djdk.map
[…]
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

- Pros: configurable. Cons: misleading colors.
- dstat is similar, and now dead (May 2019); see pcp-dstat

# vmstat

- Virtual memory statistics and more:

```
$ vmstat –Sm 1
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
 8  0      0   1620    149    552    0    0     1   179   77   12 25 34  0  0
 7  0      0   1598    149    552    0    0     0     0  205  186 46 13  0  0
 8  0      0   1617    149    552    0    0     0     8  210  435 39 21  0  0
 8  0      0   1589    149    552    0    0     0     0  218  219 42 17  0  0
[…]
```

- USAGE: vmstat [interval [count]]
- First output line has *some* summary since boot values
- High level CPU summary
  - “r” is runnable tasks

# iostat

- Block I/O (disk) stats. 1st output is since boot.

```
$ iostat -xz 1
Linux 5.0.21 (c099.xxxx)     06/24/19    _x86_64_    (32 CPU)
[...]
Device              r/s      w/s     rkB/s      wkB/s   rrqm/s   wrqm/s  %rrqm  %wrqm  \...
sda                0.01     0.00      0.16       0.00     0.00     0.00   0.00   0.00  /...
nvme3n1        19528.04    20.39 293152.56   14758.05     0.00     4.72   0.00  18.81  \...
nvme1n1        18513.51    17.83 286402.15   13089.56     0.00     4.05   0.00  18.52  /...
nvme0n1        16560.88    19.70 258184.52   14218.55     0.00     4.78   0.00  19.51  \...
```

Workload

```
...\ r_await w_await aqu-sz rareq-sz wareq-sz  svctm  %util
.../    1.90    0.00   0.00    17.01     0.00   1.13   0.00
...\    0.13   53.56   1.05    15.01   723.80   0.02  47.29
.../    0.13   49.26   0.85    15.47   734.21   0.03  48.09
...\    0.13   50.46   0.96    15.59   721.65   0.03  46.64
```

Very useful set of stats

Resulting Performance

# free

- Main memory usage:

```
$ free -m
              total        used        free      shared  buff/cache   available
Mem:          23850       18248         592        3776        5008        1432
Swap:         31699        2021       29678
```

- Recently added “available” column
  - buff/cache: block device I/O cache + virtual page cache
  - available: memory likely available to apps
  - free: completely unused memory

# strace

- System call tracer:

```
$ strace –tttT –p 313
1408393285.779746 getgroups(0, NULL)    = 1 <0.000016>
1408393285.779873 getgroups(1, [0])     = 1 <0.000015>
1408393285.780797 close(3)              = 0 <0.000016>
1408393285.781338 write(1, "wow much syscall\n", 17wow much syscall
) = 17 <0.000048>
```

- Translates syscall arguments
- Not all kernel requests (e.g., page faults)
- Currently has massive overhead (ptrace based)
  - Can slow the target by > 100x. Skews measured time (-ttt, -T).
  - http://www.brendangregg.com/blog/2014-05-11/strace-wow-much-syscall.html
- `perf trace` will replace it: uses a ring buffer & BPF

# tcpdump

- Sniff network packets for post analysis:

```
$ tcpdump -i eth0 -w /tmp/out.tcpdump
tcpdump: listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes
^C7985 packets captured
8996 packets received by filter
1010 packets dropped by kernel
# tcpdump -nr /tmp/out.tcpdump | head
reading from file /tmp/out.tcpdump, link-type EN10MB (Ethernet)
20:41:05.038437 IP 10.44.107.151.22 > 10.53.237.72.46425: Flags [P.], seq 18...
20:41:05.038533 IP 10.44.107.151.22 > 10.53.237.72.46425: Flags [P.], seq 48...
20:41:05.038584 IP 10.44.107.151.22 > 10.53.237.72.46425: Flags [P.], seq 96...
[…]
```

- Study packet sequences with timestamps (us)
- CPU overhead optimized (socket ring buffers), but can still be significant. **Use BPF in-kernel summaries instead.**

# nstat

- Replacement for netstat from iproute2
- Various network protocol statistics:
  - -s won’t reset counters, otherwise intervals can be examined
  - -d for daemon mode
- Linux keeps adding more counters

```
$ nstat -s
#kernel
IpInReceives                    31109659            0.0
IpInDelivers                    31109371            0.0
IpOutRequests                   33209552            0.0
[...]
TcpActiveOpens                  508924              0.0
TcpPassiveOpens                 388584              0.0
TcpAttemptFails                 933                 0.0
TcpEstabResets                  1545                0.0
TcpInSegs                       31099176            0.0
TcpOutSegs                      56254112            0.0
TcpRetransSegs                  3762                0.0
TcpOutRsts                      3183                0.0
[...]
```

# slabtop

- Kernel slab allocator memory usage:

```
$ slabtop
 Active / Total Objects (% used)    : 4692768 / 4751161 (98.8%)
 Active / Total Slabs (% used)      : 129083 / 129083 (100.0%)
 Active / Total Caches (% used)     : 71 / 109 (65.1%)
 Active / Total Size (% used)       : 729966.22K / 738277.47K (98.9%)
 Minimum / Average / Maximum Object : 0.01K / 0.16K / 8.00K

  OBJS ACTIVE  USE OBJ SIZE  SLABS OBJ/SLAB CACHE SIZE NAME
3565575 3565575 100%    0.10K  91425       39    365700K buffer_head
314916 314066  99%    0.19K  14996       21     59984K dentry
184192 183751  99%    0.06K   2878       64     11512K kmalloc-64
138618 138618 100%    0.94K   4077       34    130464K xfs_inode
138602 138602 100%    0.21K   3746       37     29968K xfs_ili
102116  99012  96%    0.55K   3647       28     58352K radix_tree_node
 97482  49093  50%    0.09K   2321       42      9284K kmalloc-96
 22695  20777  91%    0.05K    267       85      1068K shared_policy_node
 21312  21312 100%    0.86K    576       37     18432K ext4_inode_cache
 16288  14601  89%    0.25K    509       32      4072K kmalloc-256
[…]
```

# pcstat

- Show page cache residency by file:

```
# ./pcstat data0*
|----------+----------------+------------+-----------+---------|
| Name     | Size           | Pages      | Cached    | Percent |
|----------+----------------+------------+-----------+---------|
| data00   | 104857600      | 25600      | 25600     | 100.000 |
| data01   | 104857600      | 25600      | 25600     | 100.000 |
| data02   | 104857600      | 25600      | 4080      | 015.938 |
| data03   | 104857600      | 25600      | 25600     | 100.000 |
| data04   | 104857600      | 25600      | 16010     | 062.539 |
| data05   | 104857600      | 25600      | 0         | 000.000 |
|----------+----------------+------------+-----------+---------|
```

- Uses mincore(2) syscall. Used for database perf analysis.

# docker stats

- Soft limits (cgroups) by container:

```
# docker stats
CONTAINER     CPU %     MEM USAGE / LIMIT     MEM %    NET I/O     BLOCK I/O          PIDS
353426a09db1  526.81%   4.061 GiB / 8.5 GiB   47.78%   0 B / 0 B   2.818 MB / 0 B     247
6bf166a66e08  303.82%   3.448 GiB / 8.5 GiB   40.57%   0 B / 0 B   2.032 MB / 0 B     267
58dcf8aed0a7  41.01%    1.322 GiB / 2.5 GiB   52.89%   0 B / 0 B   0 B / 0 B          229
61061566ffe5  85.92%    220.9 MiB / 3.023 GiB 7.14%    0 B / 0 B   43.4 MB / 0 B      61
bdc721460293  2.69%     1.204 GiB / 3.906 GiB 30.82%   0 B / 0 B   4.35 MB / 0 B      66
6c80ed61ae63  477.45%   557.7 MiB / 8 GiB     6.81%    0 B / 0 B   9.257 MB / 0 B     19
337292fb5b64  89.05%    766.2 MiB / 8 GiB     9.35%    0 B / 0 B   5.493 MB / 0 B     19
b652ede9a605  173.50%   689.2 MiB / 8 GiB     8.41%    0 B / 0 B   6.48 MB / 0 B      19
d7cd2599291f  504.28%   673.2 MiB / 8 GiB     8.22%    0 B / 0 B   12.58 MB / 0 B     19
05bf9f3e0d13  314.46%   711.6 MiB / 8 GiB     8.69%    0 B / 0 B   7.942 MB / 0 B     19
09082f005755  142.04%   693.9 MiB / 8 GiB     8.47%    0 B / 0 B   8.081 MB / 0 B     19
[...]
```

- Stats are in /sys/fs/cgroups
- CPU shares and bursting breaks monitoring assumptions

# showboost

- Determine current CPU clock rate

```
# showboost
Base CPU MHz : 2500
Set CPU MHz  : 2500
Turbo MHz(s) : 3100 3200 3300 3500
Turbo Ratios : 124% 128% 132% 140%
CPU 0 summary every 1 seconds...

TIME       C0_MCYC      C0_ACYC        UTIL  RATIO    MHz
23:39:07   1618910294   89419923        64%     5%    138
23:39:08   1774059258   97132588        70%     5%    136
23:39:09   2476365498   130869241       99%     5%    132
^C
```

- Uses MSRs. Can also use PMCs for this.
- Also see turbostat.

https://github.com/brendangregg/msr-cloud-tools

# pmcarch

```
serverA# ./pmcarch -p 4093 10
K_CYCLES   K_INSTR      IPC BR_RETIRED   BR_MISPRED  BMR% LLCREF      LLCMISS     LLC%
982412660  575706336   0.59 126424862460 2416880487  1.91 15724006692 10872315070 30.86
999621309  555043627   0.56 120449284756 2317302514  1.92 15378257714 11121882510 27.68
991146940  558145849   0.56 126350181501 2530383860  2.00 15965082710 11464682655 28.19
996314688  562276830   0.56 122215605985 2348638980  1.92 15558286345 10835594199 30.35
979890037  560268707   0.57 125609807909 2386085660  1.90 15828820588 11038597030 30.26
^C

serverB# ./pmcarch -p 1928219 10
K_CYCLES   K_INSTR      IPC BR_RETIRED   BR_MISPRED  BMR% LLCREF      LLCMISS     LLC%
147523816  222396364   1.51 46053921119  641813770   1.39 8880477235  968809014  89.09
156634810  229801807   1.47 48236123575  653064504   1.35 9186609260  1183858023 87.11
152783226  237001219   1.55 49344315621  692819230   1.40 9314992450  879494418  90.56
140787179  213570329   1.52 44518363978  631588112   1.42 8675999448  712318917  91.79
136822760  219706637   1.61 45129020910  651436401   1.44 8689831639  617678747  92.89
```

- Measures instructions-per-cycle (IPC) and other metrics

https://github.com/brendangregg/pmc-cloud-tools

# cpuhot

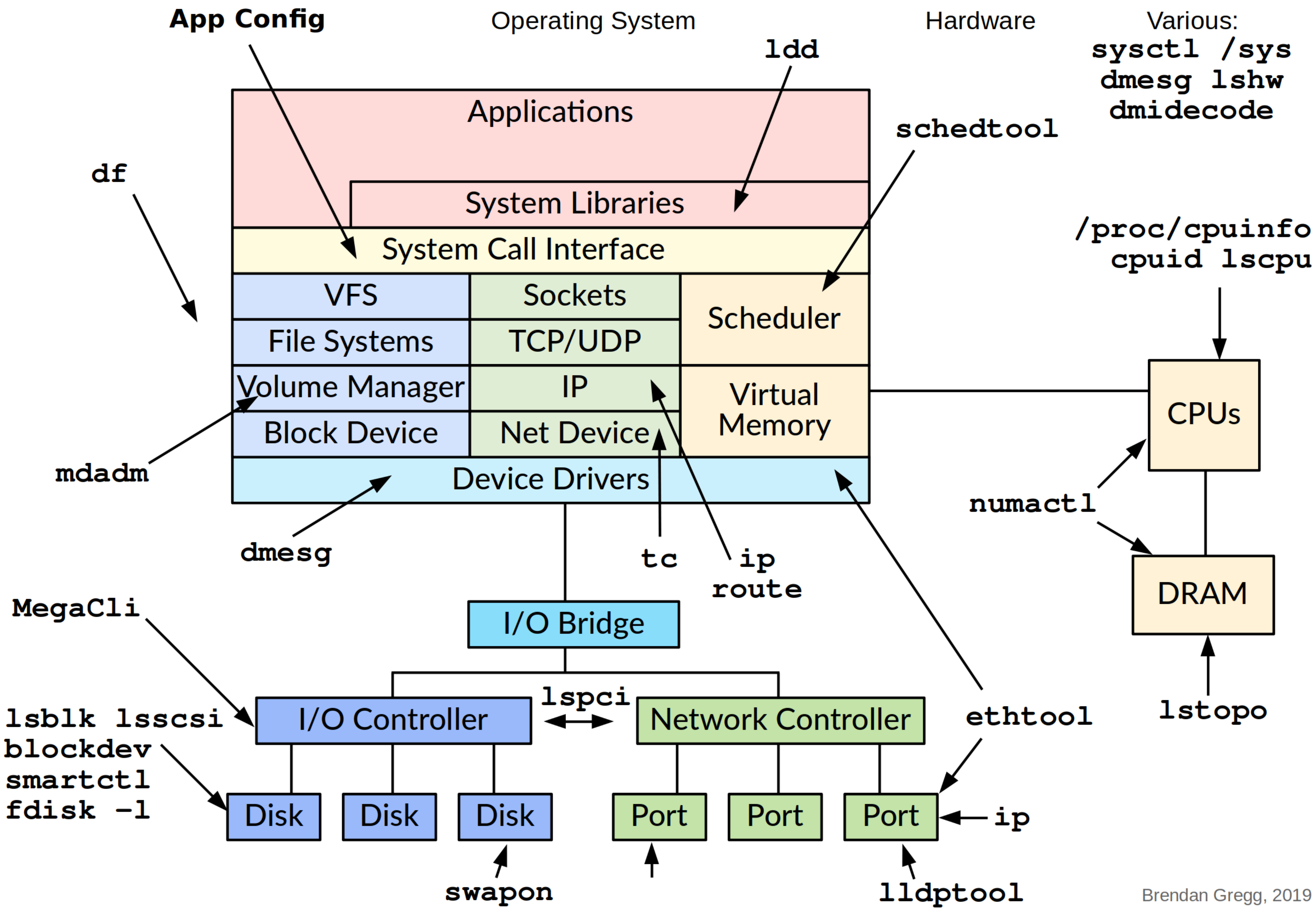
```
# dmesg
[…]
[1914678.201791] CPU6: Package temperature above threshold, cpu clock throttled ...
[1914678.206747] CPU5: Core temperature/speed normal
[1914678.206748] CPU6: Package temperature/speed normal
[…]
```

```
# ./cpuhot
- CPU1 CPU2 CPU3 CPU4 CPU5 CPU6 CPU7 CPU8 CPU9 CPU10 CPU11 CPU12 CPU13 CPU14 CPU15 CPU16
PROCHOT 95 95 95 95 95 95 95 95 95 95 95 95 95 95 95 95
Celsius 77 75 76 73 76 77 75 72 76 72 75 77 76 72 76 75
Flags 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0
```

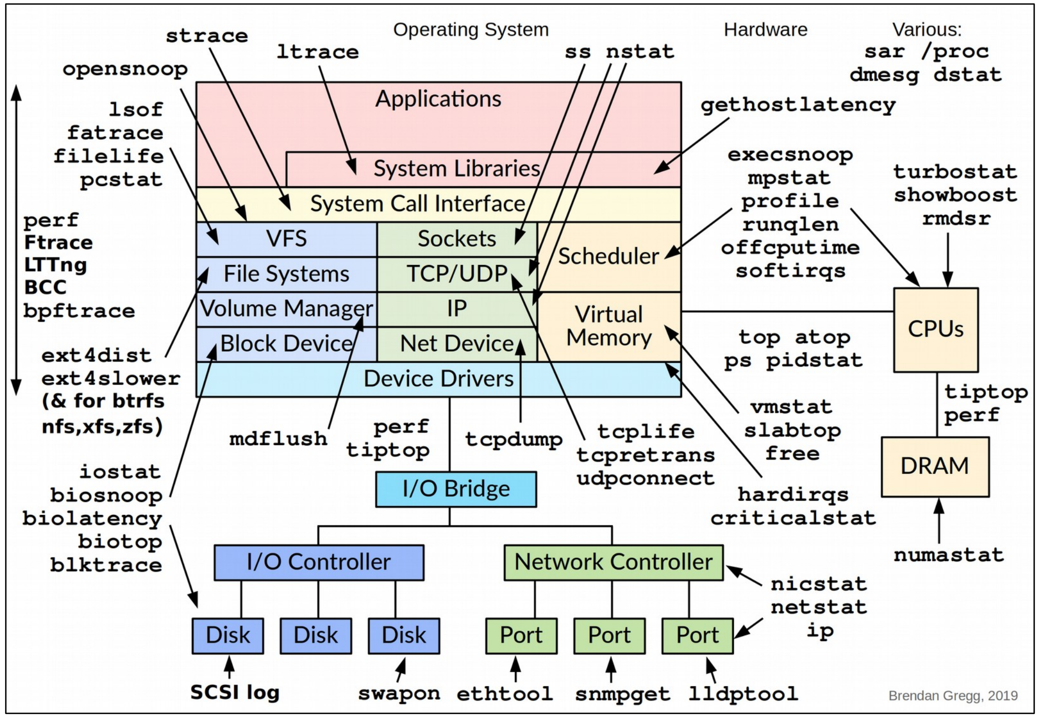
- Various thermal info is available in MSRs

https://github.com/brendangregg/msr-cloud-tools

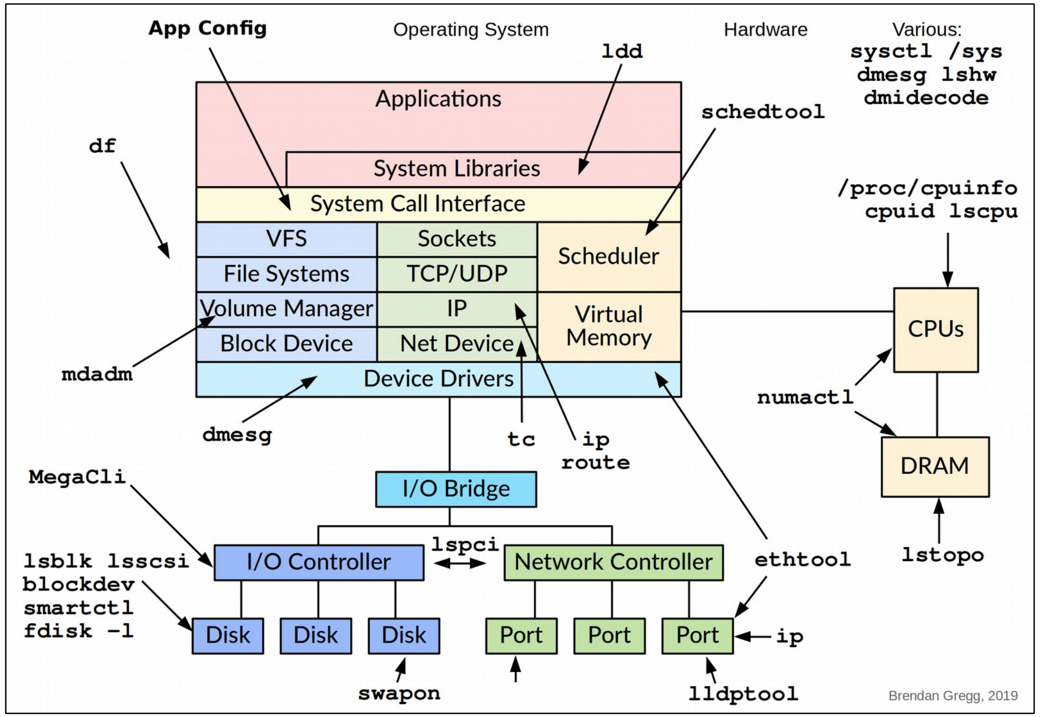
# Also: Static Performance Tuning Tools



# Where do you start...and stop?




Workload Observability




Static Configuration

# 2. Methodologies

# *Anti*-Methodologies

- The lack of a deliberate methodology…
- Street Light Anti-Method
- Drunk Man Anti-Method

# Linux Perf Analysis in 60s

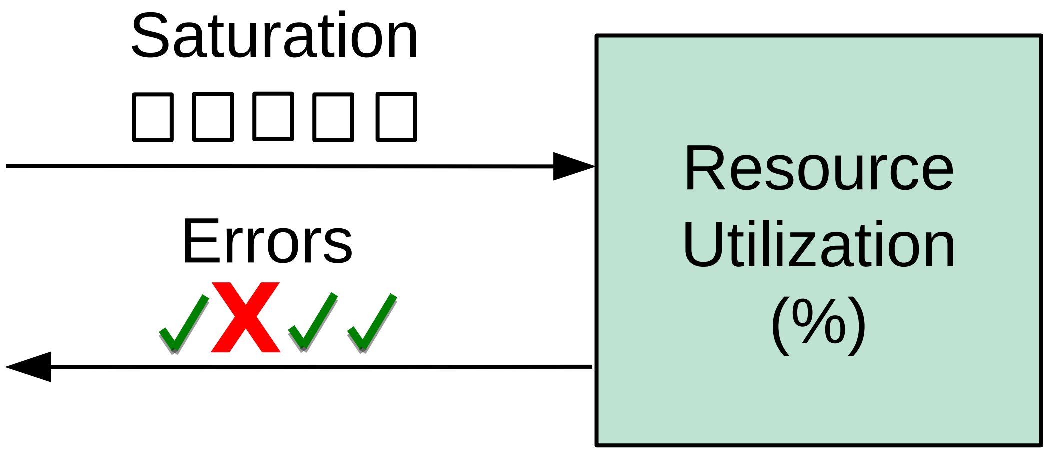
1. `uptime` → load averages
2. `dmesg -T | tail` → kernel errors
3. `vmstat 1` → overall stats by time
4. `mpstat -P ALL 1` → CPU balance
5. `pidstat 1` → process usage
6. `iostat -xz 1` → disk I/O
7. `free -m` → memory usage
8. `sar -n DEV 1` → network I/O
9. `sar -n TCP,ETCP 1` → TCP stats
10. `top` → check overview

http://techblog.netflix.com/2015/11/linux-performance-analysis-in-60s.html

# USE Method

For every resource, check:

1. **Utilization**
2. **Saturation**
3. **Errors**




For example, CPUs:

- Utilization: time busy
- Saturation: run queue length or latency
- Errors: ECC errors, etc.

Start with the questions, then find the tools

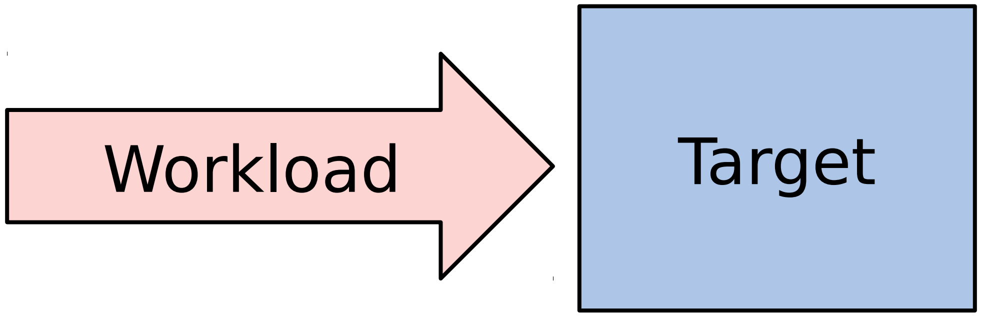
Can be applied to hardware and software (cgroups)

# Workload Characterization

Analyze workload characteristics, not resulting performance

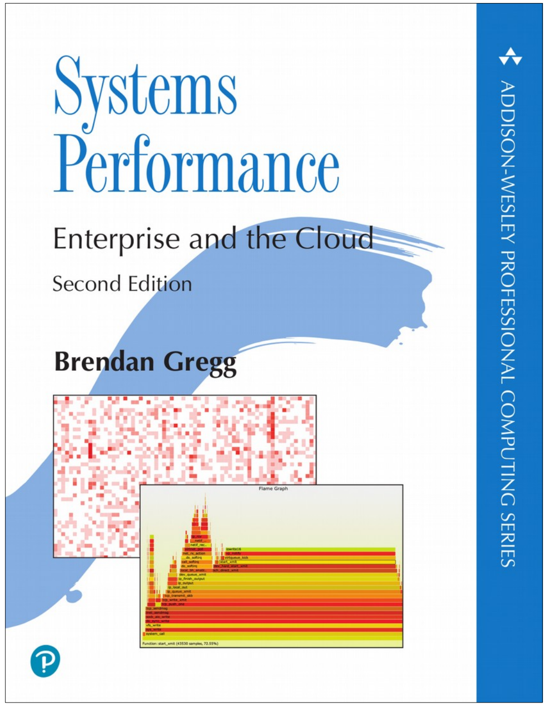
For example, CPUs:

1. **Who**: which PIDs, programs, users
2. **Why**: code paths, context
3. **What**: CPU instructions, cycles
4. **How**: changing over time



# Other Methodologies

- Resource analysis
- Workload analysis
- Drill-down analysis
- Off-CPU analysis
- Static performance tuning
- Performance mantras
- Scientific method
- 5 whys
- ...




All methodologies summarized:
http://www.brendangregg.com/methodology.html

# 3. Benchmarking

# Benchmarking

- An experimental analysis activity
  - Try observational analysis first; benchmarks can perturb
- My favorite tools:
  - fio, lmbench, sysperf, iperf, netperf
- Benchmarking is error prone
  - ~100% of benchmarks are wrong
  - You benchmark A, but actually measure B, and conclude you measured C

caution: benchmarking

# Solution: Active Benchmarking

- Root cause analysis while the benchmark runs
- For any given benchmark, ask: why not 10x?
- This takes time, but uncovers most mistakes

accurate benchmarking
takes serious effort

# 4. Profiling

# Profiling

## Can you do this?

> "As an experiment to investigate the performance of the resulting TCP/IP implementation ... the 11/750 is CPU saturated, but the 11/780 has about 30% idle time. The time spent in the system processing the data is spread out among handling for the Ethernet (20%), IP packet processing (10%), TCP processing (30%), checksumming (25%), and user system call handling (15%), with no single part of the handling dominating the time in the system."

– Bill Joy, **1981**, TCP-IP Digest, Vol 1 #6

https://www.rfc-editor.org/rfc/museum/tcp-ip-digest/tcp-ip-digest.v1n6.1

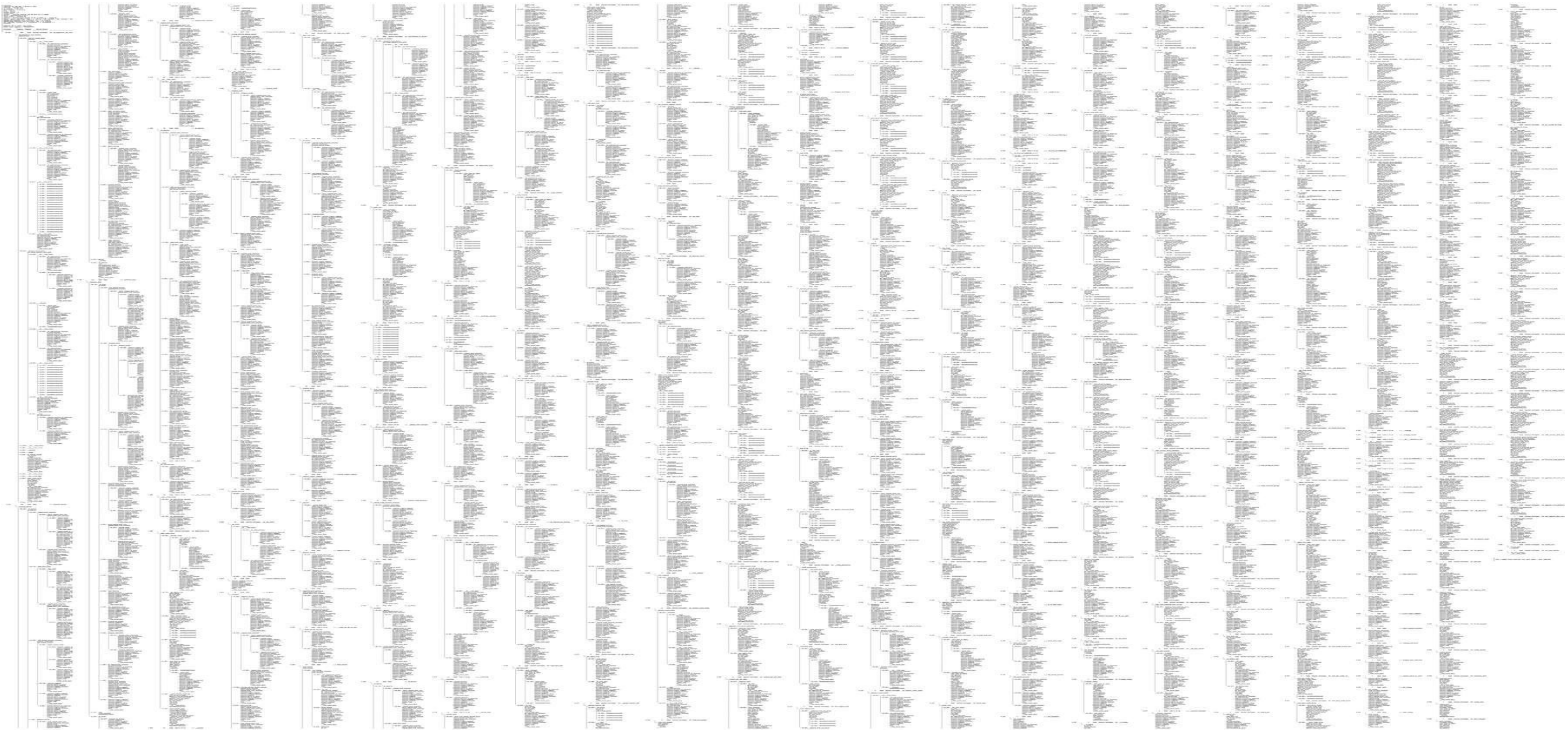
# perf: CPU profiling

- Sampling full stack traces at 99 Hertz, for 30 secs:

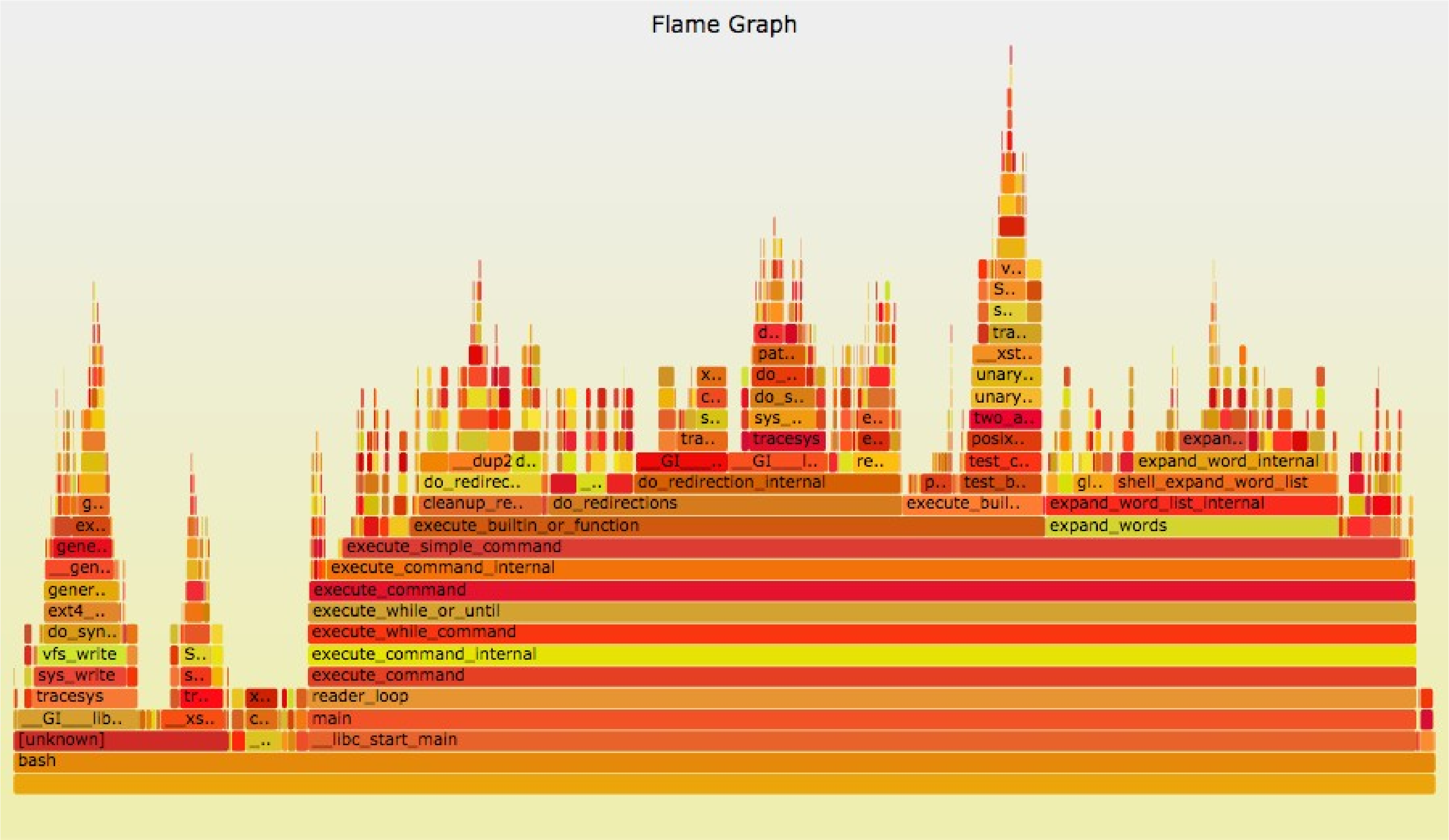
```
# perf record -F 99 -ag -- sleep 30
[ perf record: Woken up 9 times to write data ]
[ perf record: Captured and wrote 2.745 MB perf.data (~119930 samples) ]
# perf report -n --stdio
1.40%   162              java  [kernel.kallsyms]       [k] _raw_spin_lock

          |
          --- _raw_spin_lock
             |
             |--63.21%-- try_to_wake_up
             |          |
             |          |--63.91%-- default_wake_function
             |          |          |
             |          |          |--56.11%-- __wake_up_common
             |          |          |          __wake_up_locked
             |          |          |          ep_poll_callback
             |          |          |          __wake_up_common
             |          |          |          __wake_up_sync_key
             |          |          |          |
             |          |          |          |--59.19%-- sock_def_readable
[…78,000 lines truncated…]
```

# Full "perf report" Output

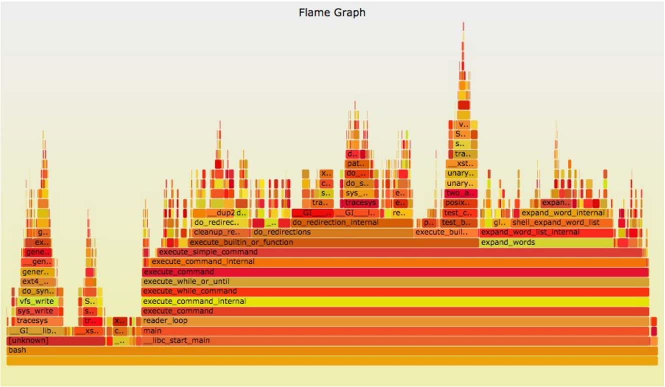


# ... as a Flame Graph



# Flame Graphs

- Visualizes a collection of stack traces
  - **x-axis**: alphabetical stack sort, to maximize merging
  - **y-axis**: stack depth
  - **color**: random (default), or a dimension
- Perl + SVG + JavaScript
  - https://github.com/brendangregg/FlameGraph
  - Takes input from many different profilers
  - Multiple d3 versions are being developed
- References:
  - **http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html**
  - http://queue.acm.org/detail.cfm?id=2927301
  - "The Flame Graph" CACM, June 2016




Instructions

# Linux CPU Flame Graphs

## Linux 2.6+, via perf:

```
git clone --depth 1 https://github.com/brendangregg/FlameGraph
cd FlameGraph
perf record -F 49 -a –g -- sleep 30
perf script --header > out.perf01
./stackcollapse-perf.pl < out.perf01 |./flamegraph.pl > perf.svg
```

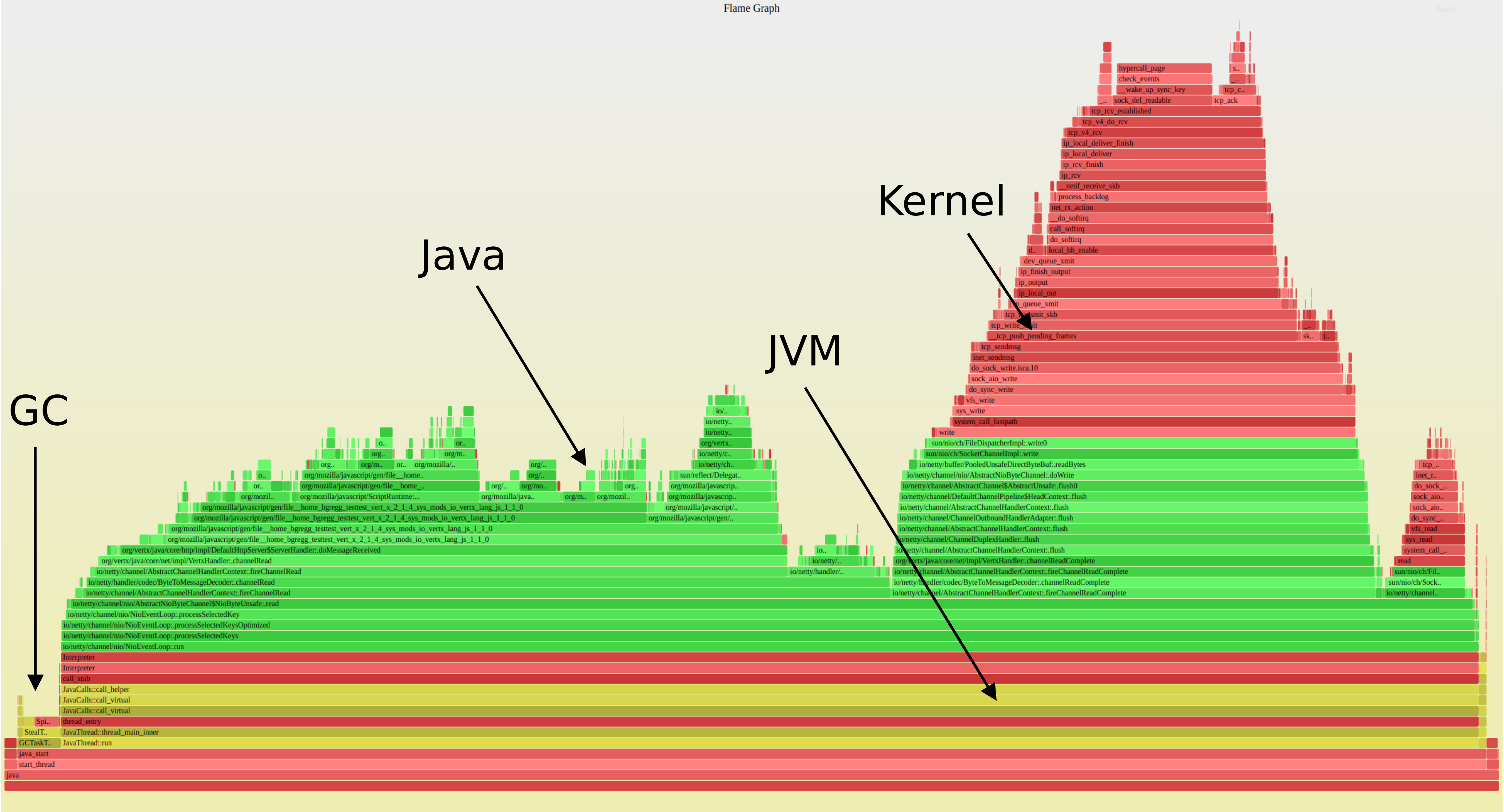
These files can be read using FlameScope

## Linux 4.9+, via BPF:

```
git clone --depth 1 https://github.com/brendangregg/FlameGraph
git clone --depth 1 https://github.com/iovisor/bcc
./bcc/tools/profile.py -dF 49 30 | ./FlameGraph/flamegraph.pl > perf.svg
```

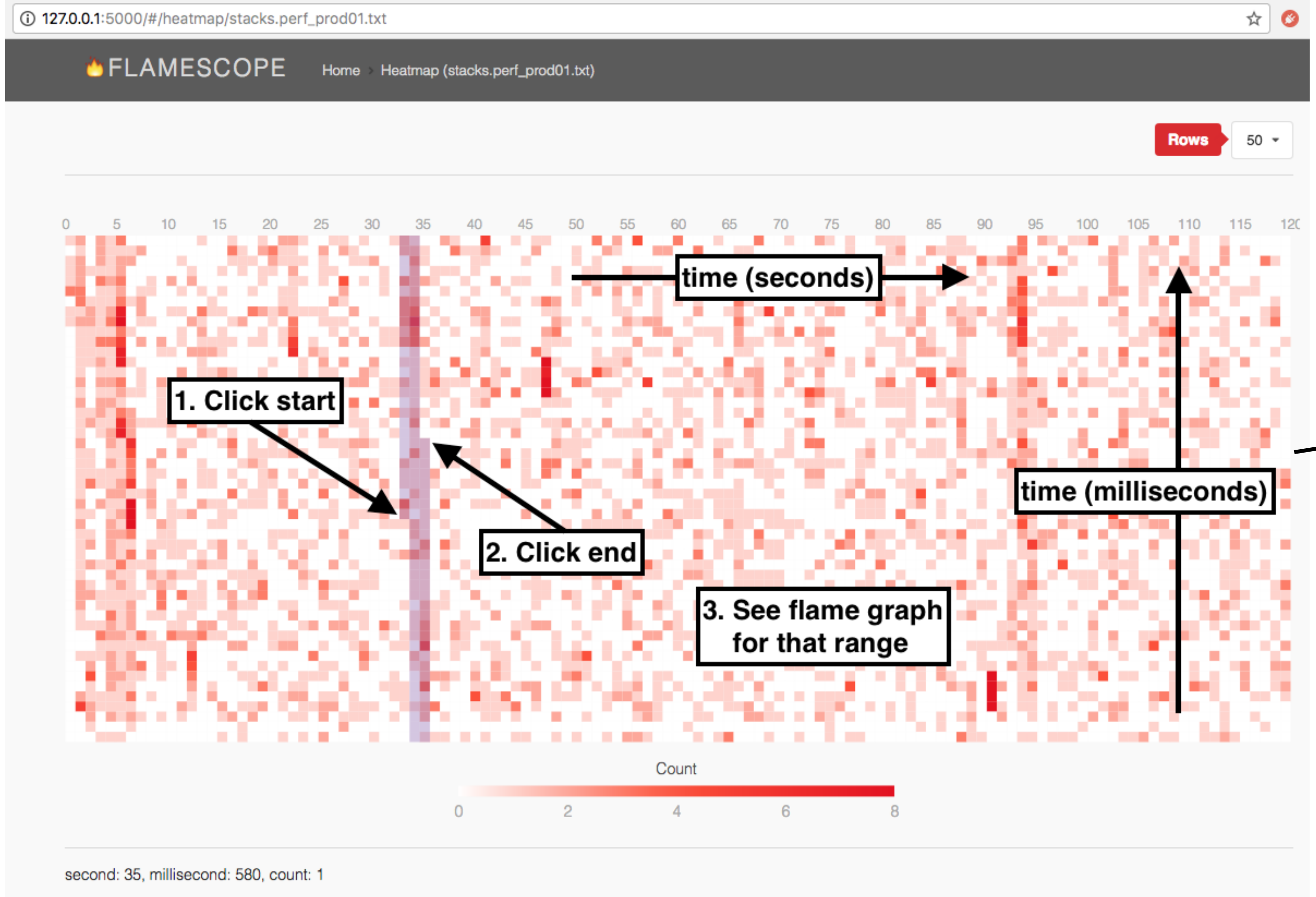
– Most efficient: no perf.data file, summarizes in-kernel

# Mixed-Mode Flame Graphs

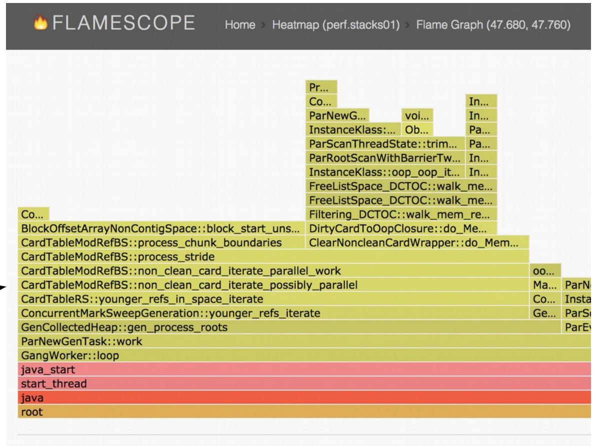


# FlameScope

- Analyze variance, perturbations




Subsecond-offset heat map




Flame graph

https://github.com/
Netflix/flamescope

# perf: Counters

- Performance Monitoring Counters (PMCs):

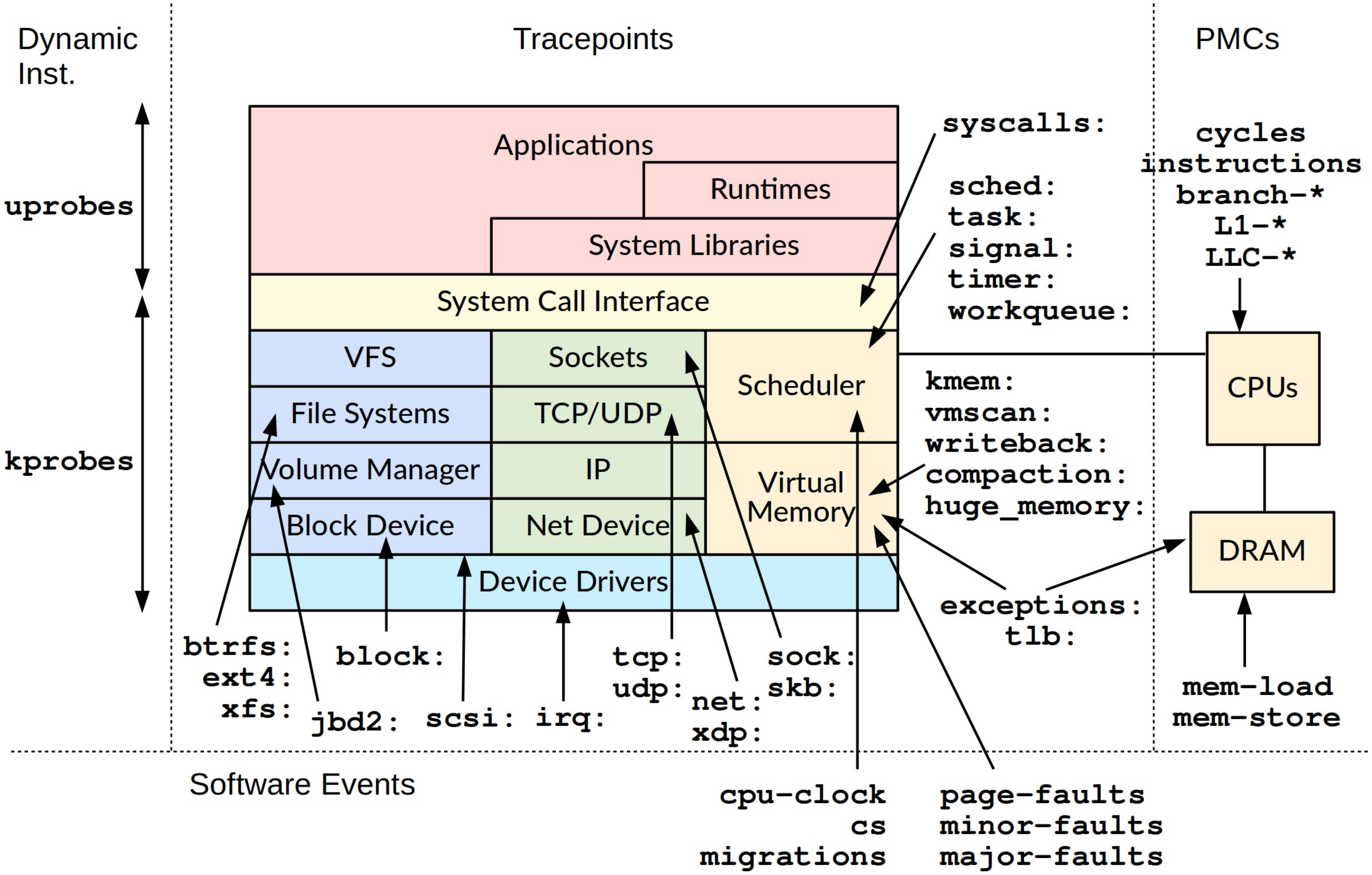
```
$ perf list | grep –i hardware
  cpu-cycles OR cycles                                [Hardware event]
  stalled-cycles-frontend OR idle-cycles-frontend     [Hardware event]
  stalled-cycles-backend OR idle-cycles-backend       [Hardware event]
  instructions                                        [Hardware event]
[…]
  L1-dcache-loads                                     [Hardware cache event]
  L1-dcache-load-misses                               [Hardware cache event]
[…]
  rNNN (see 'perf list --help' on how to encode it)  [Raw hardware event …
  mem:<addr>[:access]                                 [Hardware breakpoint]
```

- Measure CPU operations, cycles, including stall cycles
- PMCs only enabled for some cloud instance types

My front-ends, incl. pmcarch:
https://github.com/brendangregg/pmc-cloud-tools

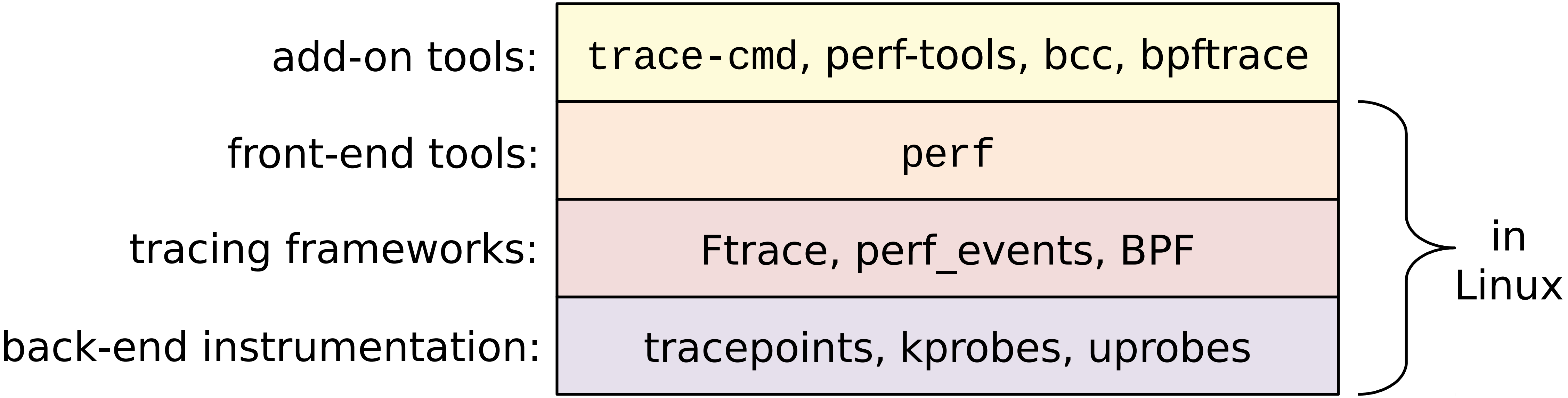
# 5. Tracing

# Linux Tracing Events




Source: http://www.brendangregg.com/bpf-performance-tools-book.html, Brendan Gregg, 2019

# Tracing Stack




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

# Ftrace: perf-tools funccount

- Built-in kernel tracing capabilities, added by Steven Rostedt and others since Linux 2.6.27

```
# ./funccount -i 1 'bio_*'
Tracing "bio_*"... Ctrl-C to end.

FUNC                              COUNT
[...]
bio_alloc_bioset                    536
bio_endio                           536
bio_free                            536
bio_fs_destructor                   536
bio_init                            536
bio_integrity_enabled               536
bio_put                             729
bio_add_page                       1004
```

- Also see trace-cmd

# perf: Tracing Tracepoints

- perf was introduced earlier; it is also a powerful tracer

In-kernel counts (efficient)

```
# perf stat -e block:block_rq_complete -a sleep 10
 Performance counter stats for 'system wide':

                91      block:block_rq_complete
```

Dump & post-process

```
# perf record -e block:block_rq_complete -a sleep 10
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.428 MB perf.data (~18687 samples) ]
# perf script
       run 30339 [000] 2083345.722857: block:block_rq_complete: 202,1 W () 12986336 + 8 [0]
       run 30339 [000] 2083345.723180: block:block_rq_complete: 202,1 W () 12986528 + 8 [0]
   swapper     0 [000] 2083345.723489: block:block_rq_complete: 202,1 W () 12986496 + 8 [0]
   swapper     0 [000] 2083346.745840: block:block_rq_complete: 202,1 WS () 1052984 + 144 [0]
 supervise 30342 [000] 2083346.746571: block:block_rq_complete: 202,1 WS () 1053128 + 8 [0]
[...]
```

http://www.brendangregg.com/perf.html
https://perf.wiki.kernel.org/index.php/Main_Page

# BCC/BPF: ext4slower

- ext4 operations slower than the 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
[…]
```

- Better indicator of application pain than disk I/O
- Measures & filters in-kernel for efficiency using BPF

https://github.com/iovisor/bcc

# bpftrace: one-liners

- Block I/O (disk) events by type; by size & comm:

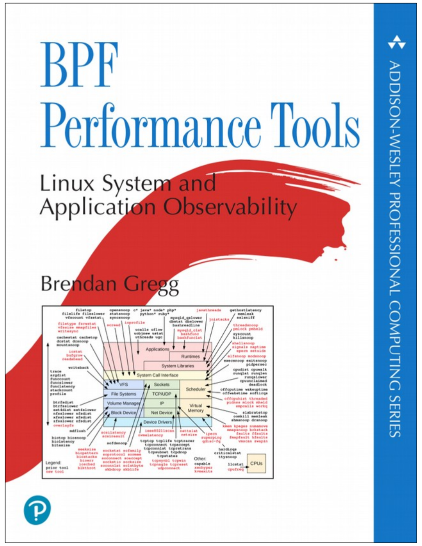
```
# bpftrace -e 't:block:block_rq_issue { @[args->rwbs] = count(); }'
Attaching 1 probe...
^C
@[WS]: 2
@[RM]: 12
@[RA]: 1609
@[R]: 86421
```

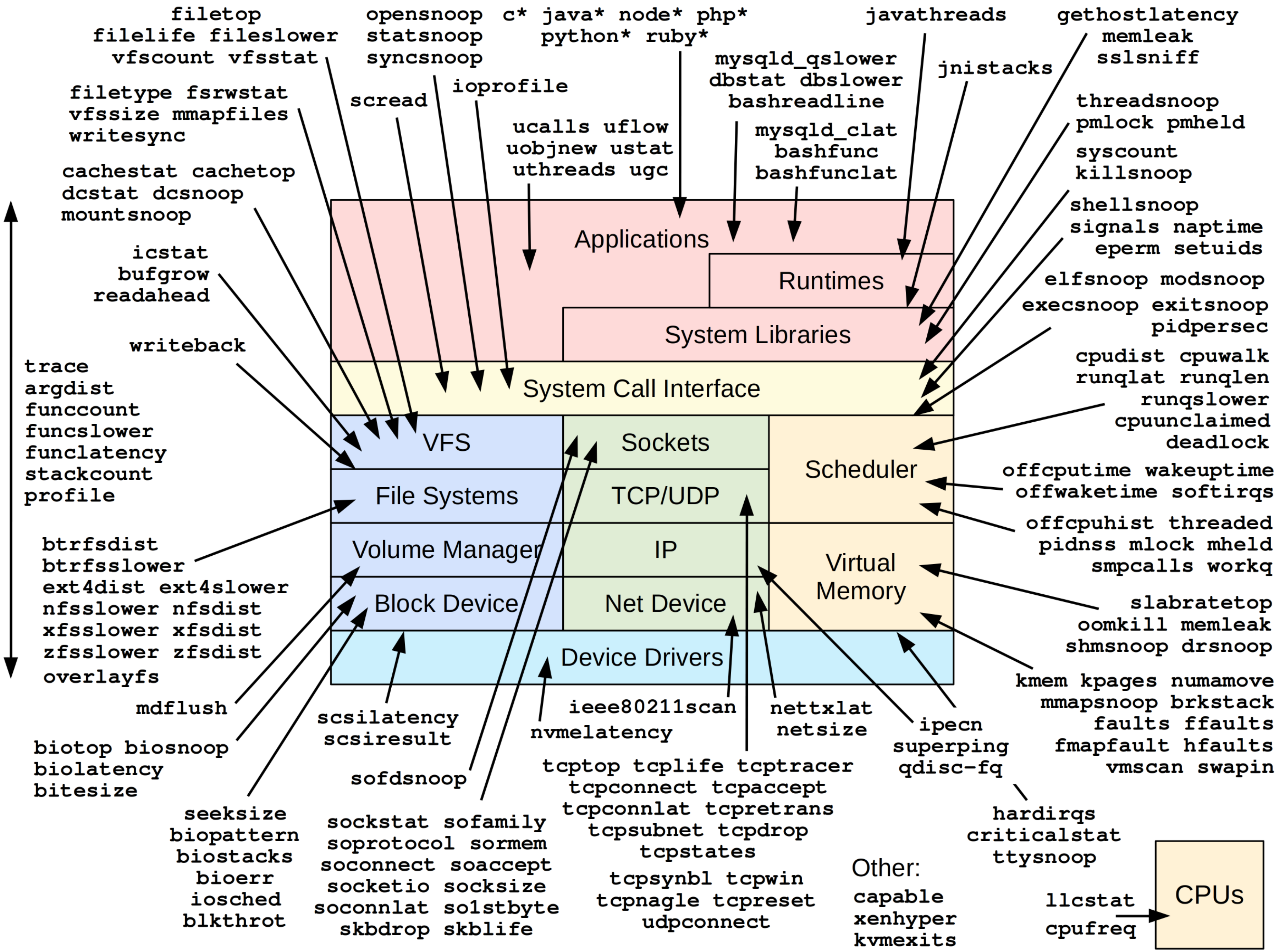
```
# bpftrace -e 't:block:block_rq_issue { @bytes[comm] = hist(args->bytes); }'
Attaching 1 probe...
^C
@bytes[dmcrypt_write]:
[4K, 8K)              68 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@|
[8K, 16K)             35 |@@@@@@@@@@@@@@@@@@@@@@@@@@                          |
[16K, 32K)             4 |@@@                                                 |
[32K, 64K)             1 |                                                    |
[64K, 128K)            2 |@                                                   |
[...]
```

https://github.com/iovisor/bpftrace

# BPF Perf Tools (2019)

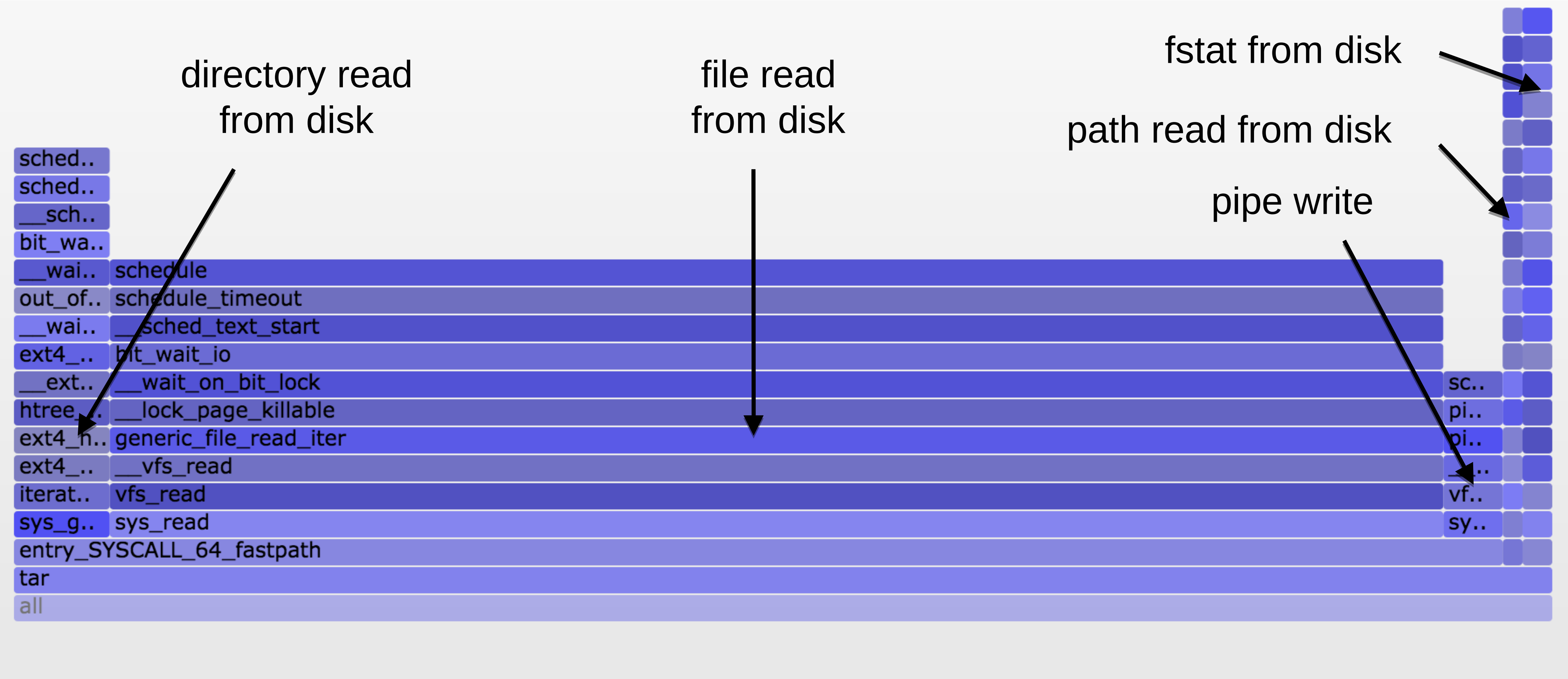
BCC & bpftrace repos contain many of these. The book has them all.



# Off-CPU Analysis

- Explain all blocking events. High-overhead: needs BPF.



# 6. Tuning

# Ubuntu Bionic Tuning: Sep 2020 (1/2)

- CPU
  **schedtool –B PID**
  disable Ubuntu apport (crash reporter)
  upgrade to Bionic (scheduling improvements)
- Virtual Memory
  **vm.swappiness = 0 # from 60**
- Memory
  **echo madvise > /sys/kernel/mm/transparent_hugepage/enabled**
  **kernel.numa_balancing = 0**
- File System
  **vm.dirty_ratio = 80 # from 40**
  **vm.dirty_background_ratio = 5 # from 10**
  **vm.dirty_expire_centisecs = 12000 # from 3000**
  **mount -o defaults,noatime,discard,nobarrier …**
- Storage I/O
  **/sys/block/*/queue/rq_affinity 1 # or 2**
  **/sys/block/*/queue/scheduler kyber**
  **/sys/block/*/queue/nr_requests 256**
  **/sys/block/*/queue/read_ahead_kb 128**
  **mdadm –chunk=64 …**

# Ubuntu Bionic Tuning: Sep 2020 (2/2)

- Networking

```
net.core.default_qdisc = fq
net.core.netdev_max_backlog = 5000           # may update to 1000
net.core.rmem_max = 16777216
net.core.somaxconn = 1024                    # may update to 4096
net.core.wmem_max = 16777216
net.ipv4.ip_local_port_range = 10240 65535
net.ipv4.tcp_abort_on_overflow = 1   # maybe
net.ipv4.tcp_congestion_control = bbr
net.ipv4.tcp_max_syn_backlog = 8192
net.ipv4.tcp_rmem = 4096 12582912 16777216   # or 8388608 ...
net.ipv4.tcp_slow_start_after_idle = 0
net.ipv4.tcp_syn_retries = 2
net.ipv4.tcp_tw_reuse = 1
net.ipv4.tcp_wmem = 4096 12582912 16777216   # or 8388608 ...
```

- Hypervisor

```
echo tsc > /sys/devices/…/current_clocksource
```

  Plus use AWS Nitro

- Other

```
net.core.bpf_jit_enable = 1
sysctl -w kernel.perf_event_max_stack=1000
```

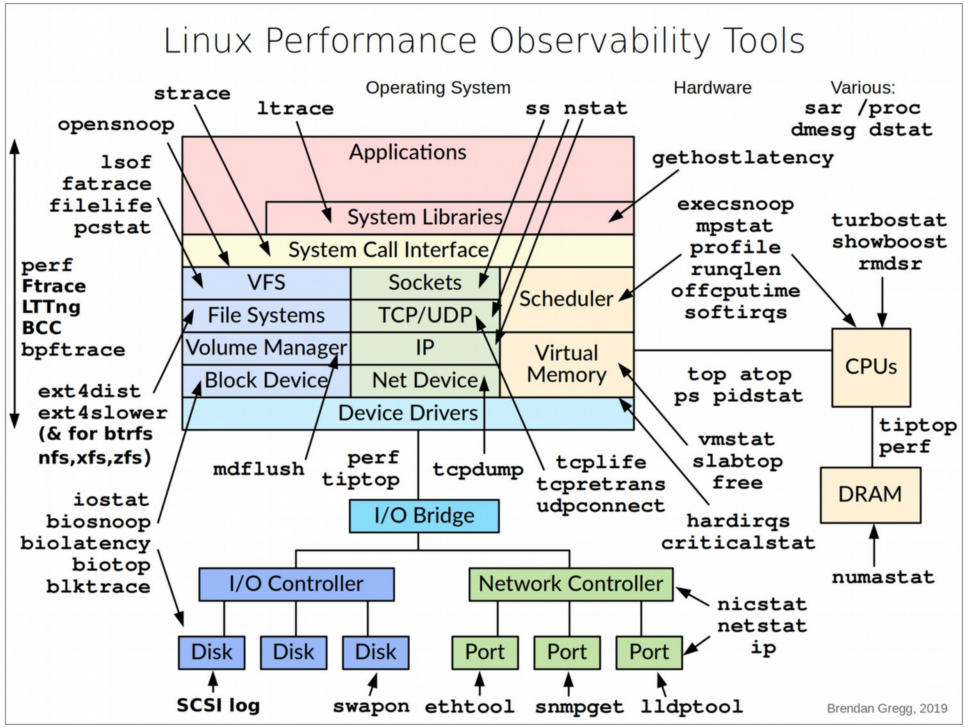
# Takeaways

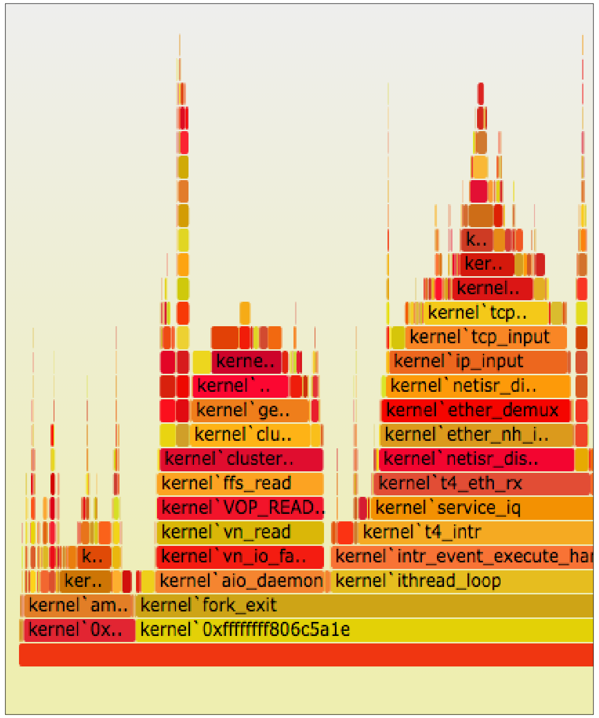
## Systems Performance is:

Observability, Methodologies, Benchmarking, Profiling, Tracing, Tuning

## Print out for your office wall:

1. uptime
2. dmesg -T | tail
3. vmstat 1
4. mpstat -P ALL 1
5. pidstat 1
6. iostat -xz 1
7. free -m
8. sar -n DEV 1
9. sar -n TCP,ETCP 1
10. top



# Links

Netflix Tech Blog on Linux:
- http://techblog.netflix.com/2015/11/linux-performance-analysis-in-60s.html
- http://techblog.netflix.com/2015/08/netflix-at-velocity-2015-linux.html

Linux Performance:
- http://www.brendangregg.com/linuxperf.html

Linux perf:
- https://perf.wiki.kernel.org/index.php/Main_Page
- http://www.brendangregg.com/perf.html

Linux ftrace:
- https://www.kernel.org/doc/Documentation/trace/ftrace.txt
- https://github.com/brendangregg/perf-tools

Linux BPF:
- http://www.brendangregg.com/ebpf.html
- http://www.brendangregg.com/bpf-performance-tools-book.html
- https://github.com/iovisor/bcc
- https://github.com/iovisor/bpftrace

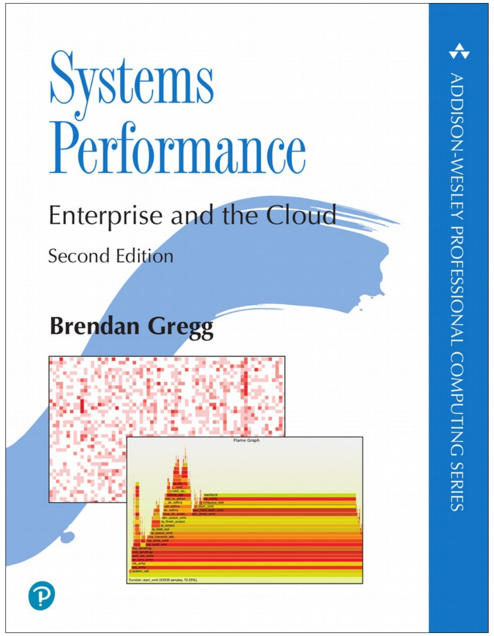
Methodologies:
- http://www.brendangregg.com/USEmethod/use-linux.html
- http://www.brendangregg.com/activebenchmarking.html

Flame Graphs & FlameScope:
- http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html
- http://queue.acm.org/detail.cfm?id=2927301
- https://github.com/Netflix/flamescope

MSRs and PMCs
- https://github.com/brendangregg/msr-cloud-tools
- https://github.com/brendangregg/pmc-cloud-tools

# Thanks

- Q&A in #qa-brendan
- http://slideshare.net/brendangregg
- http://www.brendangregg.com
- bgregg@netflix.com
- @brendangregg




Nov/Dec 2020