Linux Performance Analysis and Tools

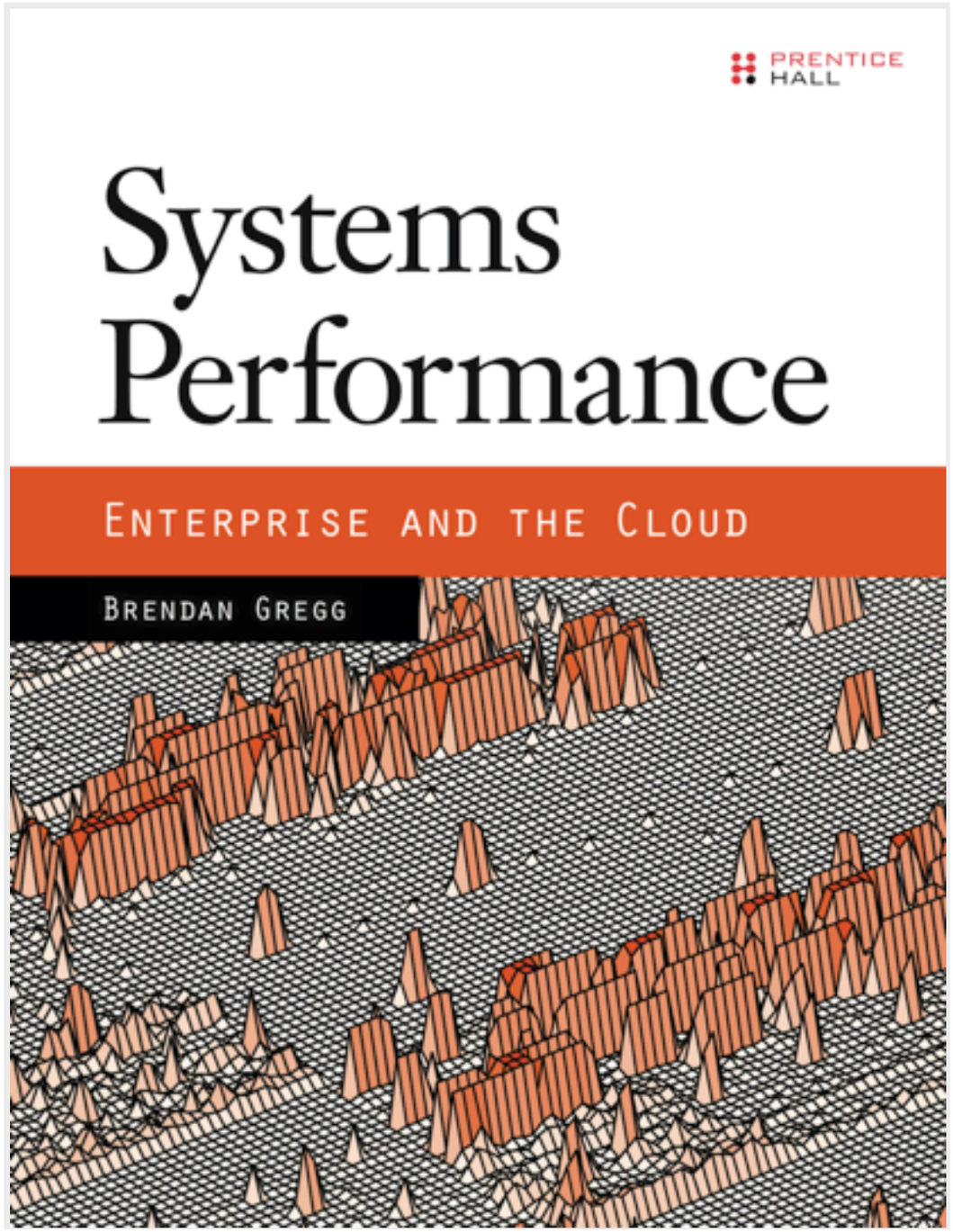
Brendan Gregg
Lead Performance Engineer

brendan@joyent.com
@brendangregg

Polyglot
Vancouver
October, 2013

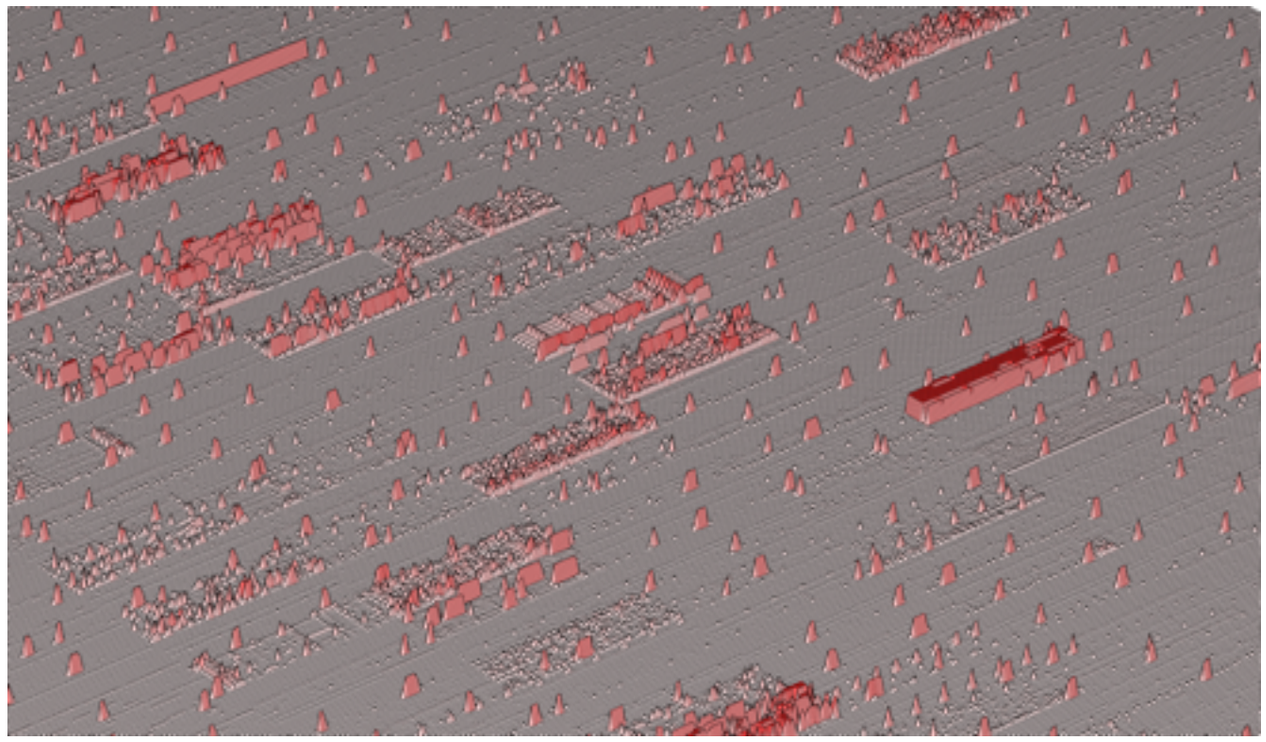
whoami

- G'Day, I'm Brendan
- Performance Engineering
- Work/Research: tools, visualizations, methodologies



Joyent

- High-Performance Cloud Infrastructure
- OS-Virtualization for bare metal performance (SmartOS), KVM for Linux and Windows guests, and all on ZFS
- Core developers of SmartOS and node.js
- Many customers, who collectively run everything imaginable (fruitful environment for performance research)
- CPU utilization on one of our datacenters:



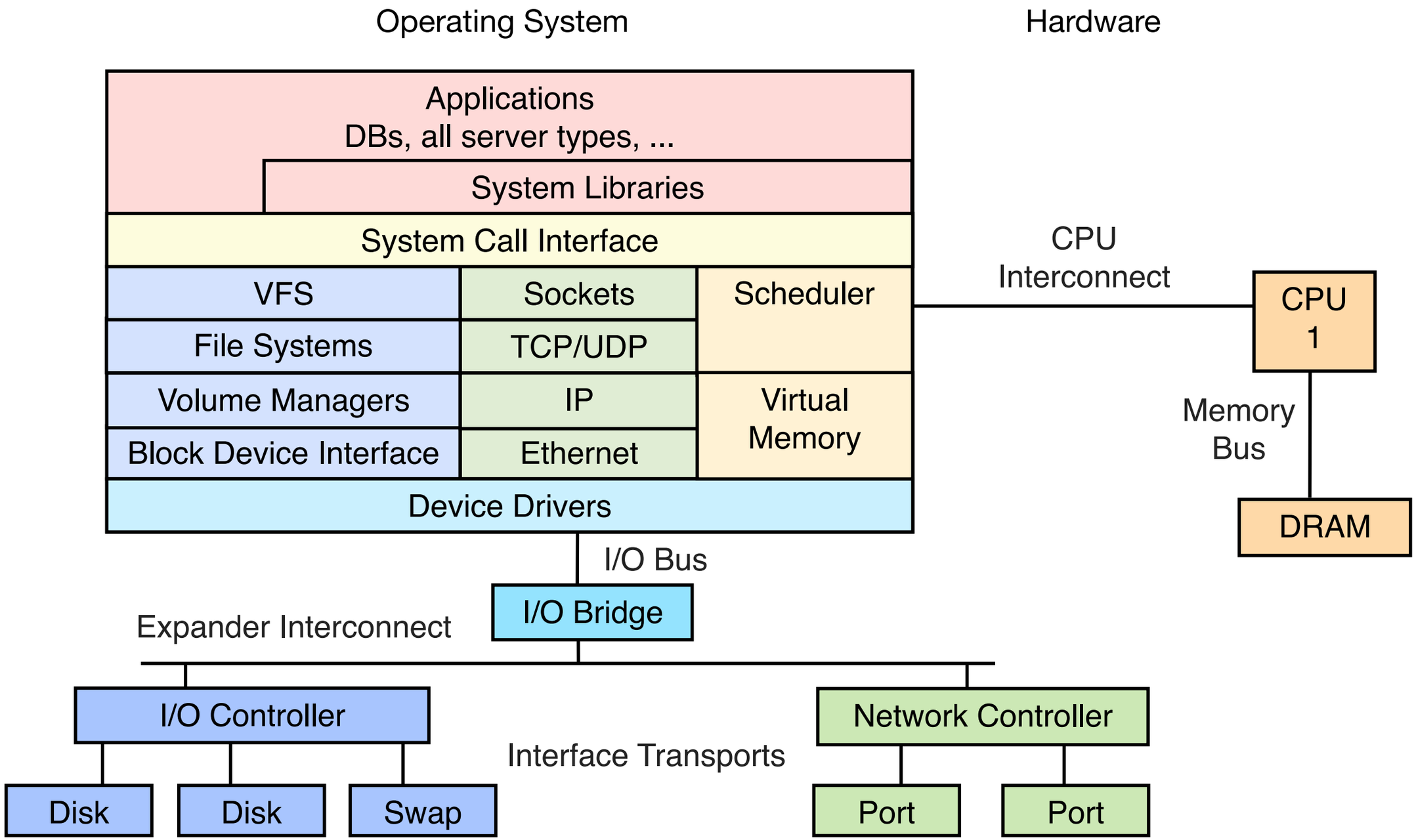
Agenda

- Aim: get the best performance from your systems and applications, and troubleshoot issues efficiently
- 1. Tool focus
- 2. Methodologies
- 3. Question focus

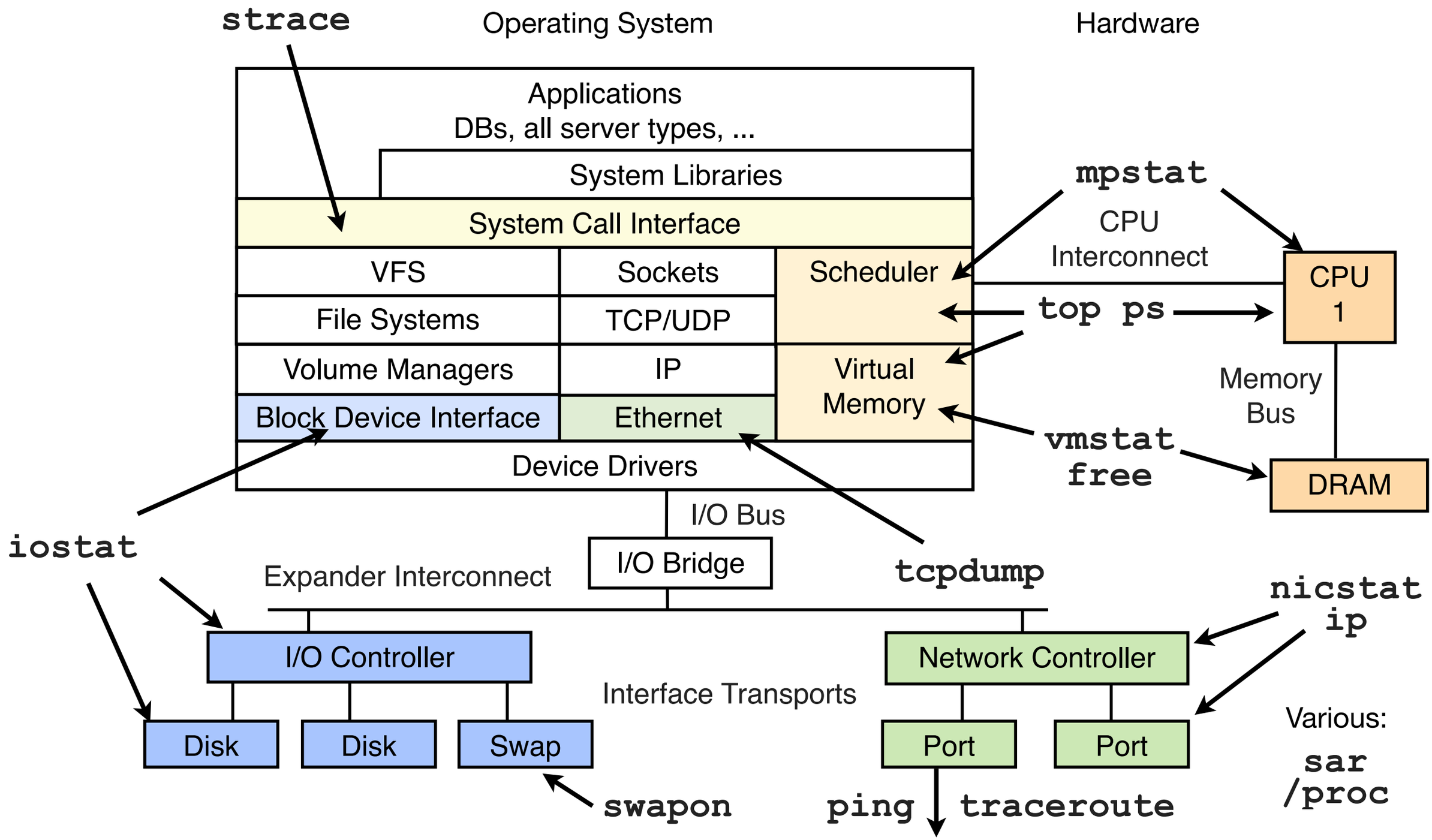
Tool Focus

- Run tools, look for problems

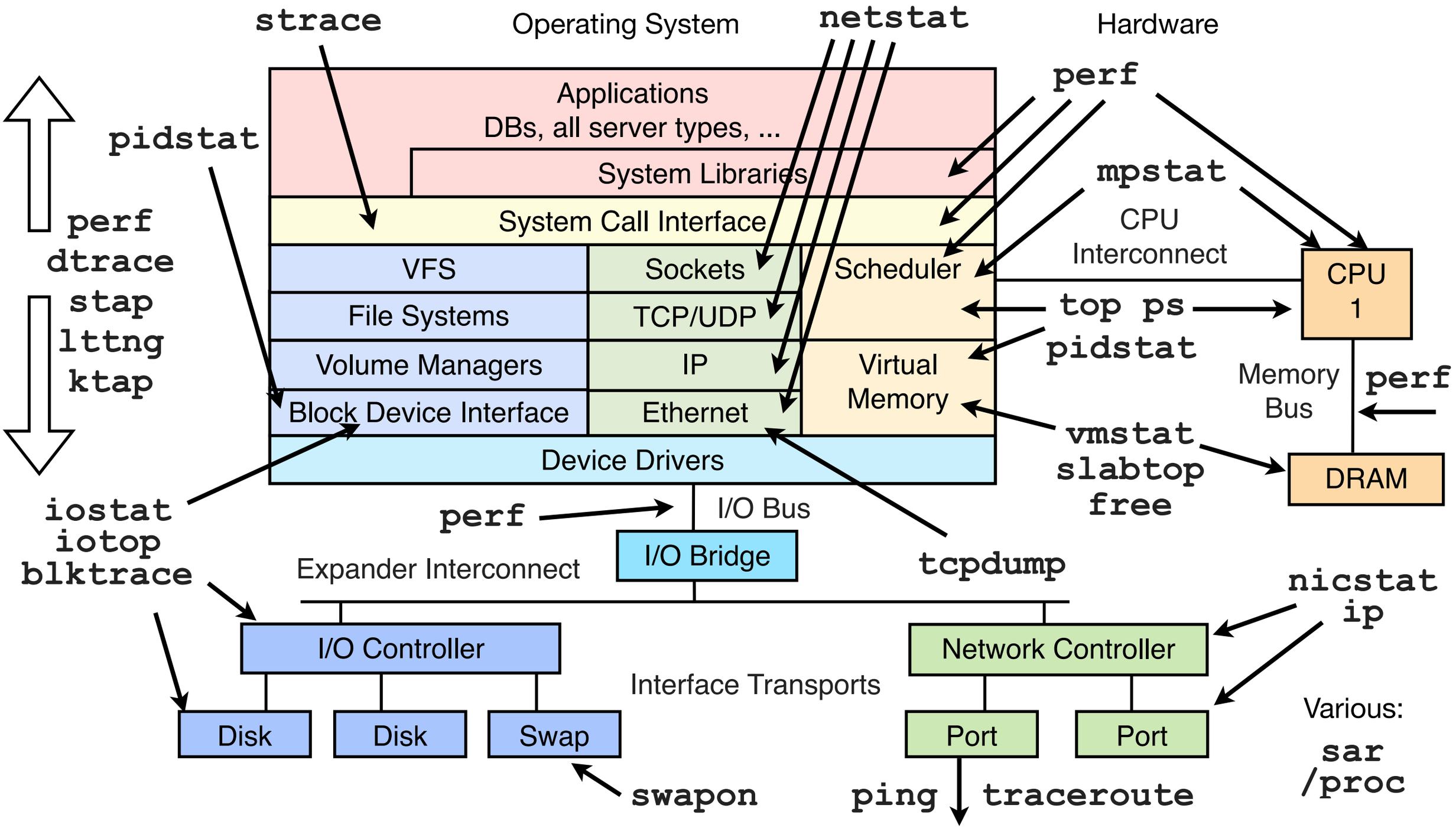
System Functional Diagram



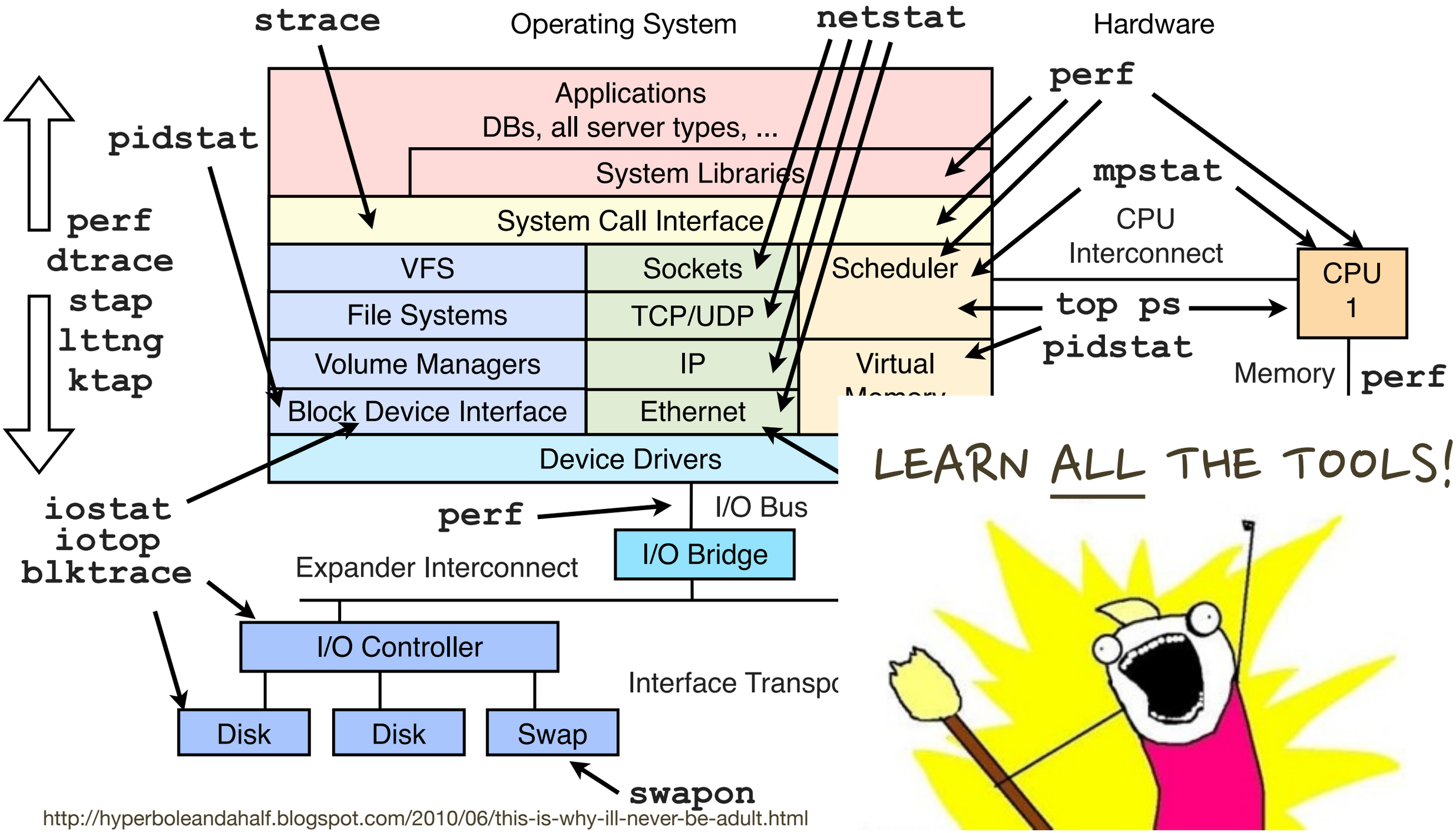
Basic Performance Analysis Tools: Linux



More Performance Analysis Tools: Linux



More Performance Analysis Tools: Linux



http://hyperboleandahalf.blogspot.com/2010/06/this-is-why-ill-never-be-adult.html

uptime

- Shows *load averages*, which are also shown by other tools:

```
$ uptime
 16:23:34 up 126 days,  1:03,  1 user,  load average: 5.09, 2.12, 1.82
```

- This counts runnable threads (tasks), on-CPU, or, runnable and waiting. Linux includes tasks blocked on disk I/O.
- These are exponentially-damped moving averages, with time constants of 1, 5 and 15 minutes. With three values you can see if load is increasing, steady, or decreasing.
- If the load is greater than the CPU count, it might mean the CPUs are saturated (100% utilized), and threads are suffering scheduler latency. Might. There's that disk I/O factor too.
- This is only useful as a clue. Use other tools to investigate!

top

- System-wide and per-process summaries:

```
$ top
top - 01:38:11 up 63 days,  1:17,  2 users,  load average: 1.57, 1.81, 1.77
Tasks: 256 total,   2 running, 254 sleeping,   0 stopped,   0 zombie
Cpu(s):  2.0%us,  3.6%sy,  0.0%ni, 94.2%id,  0.0%wa,  0.0%hi,  0.2%si,  0.0%st
Mem:  49548744k total, 16746572k used, 32802172k free,   182900k buffers
Swap: 100663292k total,        0k used, 100663292k free, 14925240k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
11721 web       20   0  623m  50m 4984 R   93  0.1   0:59.50 node
11715 web       20   0  619m  20m 4916 S   25  0.0   0:07.52 node
   10 root      20   0     0    0    0 S    1  0.0 248:52.56 ksoftirqd/2
   51 root      20   0     0    0    0 S    0  0.0   0:35.66 events/0
11724 admin     20   0 19412 1444  960 R    0  0.0   0:00.07 top
    1 root      20   0 23772 1948 1296 S    0  0.0   0:04.35 init
[...]
```

- %CPU = interval sum for all CPUs (varies on other OSes)
- top can consume CPU (syscalls to read /proc)
- Straight-forward. Or is it?

top, cont.

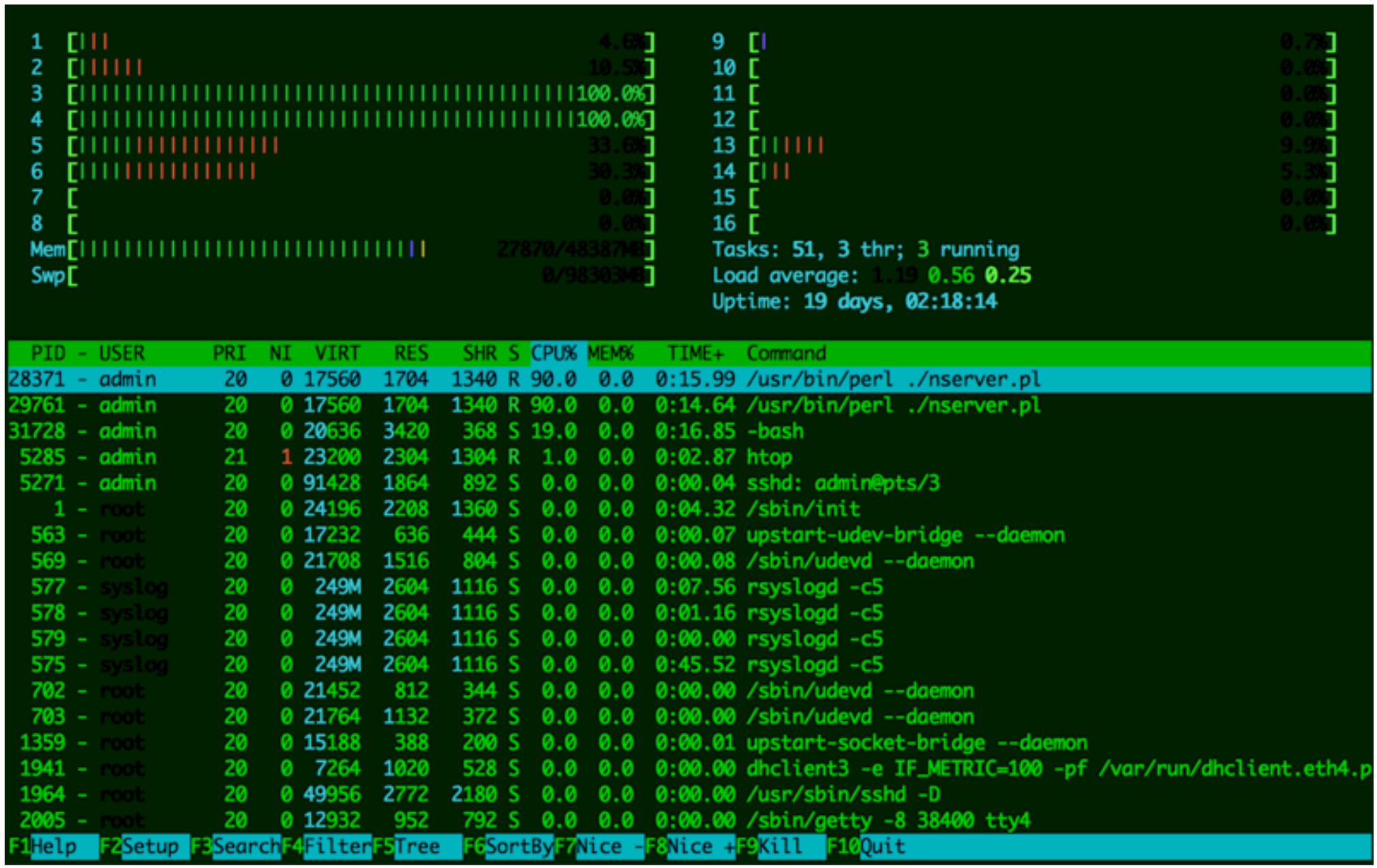
- Interview questions:
  - 1. Does it show all CPU consumers?
  - 2. A process has high %CPU – next steps for analysis?

top, cont.

- 1. top can miss:
  - short-lived processes
  - kernel threads (tasks), unless included (see top options)
- 2. analyzing high CPU processes:
  - identify why – profile code path
  - identify what – execution or stall cycles
- High %CPU time may be stall cycles on memory I/O – upgrading to faster CPUs doesn't help!

htop

- Super top. Super configurable. Eg, basic CPU visualization:



mpstat

- Check for hot threads, unbalanced workloads:

```
$ mpstat -P ALL 1
02:47:49   CPU    %usr   %nice    %sys %iowait    %irq  %soft %steal %guest  %idle
02:47:50   all   54.37    0.00   33.12    0.00    0.00   0.00   0.00   0.00  12.50
02:47:50     0   22.00    0.00   57.00    0.00    0.00   0.00   0.00   0.00  21.00
02:47:50     1   19.00    0.00   65.00    0.00    0.00   0.00   0.00   0.00  16.00
02:47:50     2   24.00    0.00   52.00    0.00    0.00   0.00   0.00   0.00  24.00
02:47:50     3  100.00    0.00    0.00    0.00    0.00   0.00   0.00   0.00   0.00
02:47:50     4  100.00    0.00    0.00    0.00    0.00   0.00   0.00   0.00   0.00
02:47:50     5  100.00    0.00    0.00    0.00    0.00   0.00   0.00   0.00   0.00
02:47:50     6  100.00    0.00    0.00    0.00    0.00   0.00   0.00   0.00   0.00
02:47:50     7   16.00    0.00   63.00    0.00    0.00   0.00   0.00   0.00  21.00
02:47:50     8  100.00    0.00    0.00    0.00    0.00   0.00   0.00   0.00   0.00
[...]
```

- Columns are summarized system-wide in top(1)’s header

iostat

- Disk I/O statistics. 1st output is summary since boot.

```
$ iostat -xkdz 1

Linux 2.6.35-32-server (prod21)         02/20/13      _x86_64_     (16 CPU)

Device:          rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s  \ ...
sda                0.00     0.00    0.00    0.00     0.00     0.00  / ...
sdb                0.00     0.35    0.00    0.05     0.10     1.58  \ ...
                                                                    / ...
Device:          rrqm/s   wrqm/s     r/s     w/s    rkB/s    wkB/s  \ ...
sdb                0.00     0.00  591.00    0.00  2364.00     0.00  / ...
```

workload input →

```
... \    avgqu-sz   await r_await w_await  svctm  %util
... /        0.00    0.84    0.84    0.00   0.84   0.00
... \        0.00    3.82    3.47    3.86   0.30   0.00
... /        0.00    2.31    2.31    0.00   2.31   0.00
... \
... /    avgqu-sz   await r_await w_await  svctm  %util
... \        0.95    1.61    1.61    0.00   1.61  95.00
```

resulting performance →

iostat, cont.

- %util: usefulness depends on target – virtual devices backed by multiple disks may accept more work a 100% utilization
- Also calculate I/O controller stats by summing their devices
- One nit: would like to see disk errors too. Add a “-e”?

vmstat

- Virtual-Memory statistics, and other high-level summaries:

```
$ vmstat 1
procs -----------memory---------- ---swap-- -----io---- -system-- ----cpu----
 r  b   swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa
15  0   2852 46686812 279456 1401196   0   0     0     0    0    0  0  0 100  0
16  0   2852 46685192 279456 1401196   0   0     0     0 2136 36607 56 33 11  0
15  0   2852 46685952 279456 1401196   0   0     0    56 2150 36905 54 35 11  0
15  0   2852 46685960 279456 1401196   0   0     0     0 2173 36645 54 33 13  0
[...]
```

- First line of output includes *some* summary-since-boot values
- “r” = total number of runnable threads, *including* those running
- Swapping (aka paging) allows over-subscription of main memory by swapping pages to disk, but costs performance

free

- Memory usage summary (Kbytes default):

```
$ free
             total       used       free     shared    buffers     cached
Mem:      49548744   32787912   16760832          0      61588     342696
-/+ buffers/cache:   32383628   17165116
Swap:    100663292          0  100663292
```

- buffers: block device I/O cache
- cached: virtual page cache

ping

- Simple network test (ICMP):

```
$ ping www.hilton.com
PING a831.b.akamai.net (63.234.226.9): 56 data bytes
64 bytes from 63.234.226.9: icmp_seq=0 ttl=56 time=737.737 ms
Request timeout for icmp_seq 1
64 bytes from 63.234.226.9: icmp_seq=2 ttl=56 time=819.457 ms
64 bytes from 63.234.226.9: icmp_seq=3 ttl=56 time=897.835 ms
64 bytes from 63.234.226.9: icmp_seq=4 ttl=56 time=669.052 ms
64 bytes from 63.234.226.9: icmp_seq=5 ttl=56 time=799.932 ms
^C
--- a831.b.akamai.net ping statistics ---
6 packets transmitted, 5 packets received, 16.7% packet loss
round-trip min/avg/max/stddev = 669.052/784.803/897.835/77.226 ms
```

- Used to measure network latency. Actually kernel <-> kernel IP stack latency, including how the network handles ICMP.
- Tells us some, but not a lot (above is an exception). Lots of other/better tools for this (eg, hping). Try using TCP.

nicstat

- Network statistics tool, ver 1.92 on Linux:

```
# nicstat -z 1
    Time      Int   rKB/s   wKB/s   rPk/s   wPk/s    rAvs    wAvs %Util     Sat
01:20:58     eth0    0.07    0.00    0.95    0.02   79.43   64.81  0.00    0.00
01:20:58     eth4    0.28    0.01    0.20    0.10  1451.3   80.11  0.00    0.00
01:20:58  vlan123    0.00    0.00    0.00    0.02   42.00   64.81  0.00    0.00
01:20:58      br0    0.00    0.00    0.00    0.00   42.00   42.07  0.00    0.00
    Time      Int   rKB/s   wKB/s   rPk/s   wPk/s    rAvs    wAvs %Util     Sat
01:20:59     eth4 42376.0   974.5 28589.4 14002.1  1517.8   71.27  35.5    0.00
    Time      Int   rKB/s   wKB/s   rPk/s   wPk/s    rAvs    wAvs %Util     Sat
01:21:00     eth0    0.05    0.00    1.00    0.00   56.00    0.00  0.00    0.00
01:21:00     eth4 41834.7   977.9 28221.5 14058.3  1517.9   71.23  35.1    0.00
    Time      Int   rKB/s   wKB/s   rPk/s   wPk/s    rAvs    wAvs %Util     Sat
01:21:01     eth4 42017.9   979.0 28345.0 14073.0  1517.9   71.24  35.2    0.00
[...]
```

- This was the tool I wanted, and finally wrote it out of frustration (Tim Cook ported and enhanced it on Linux)
- Calculate network controller stats by summing interfaces

dstat

- A better vmstat-like tool. Does coloring (FWIW).

```
# dstat 1
You did not select any stats, using -cdngy by default.
----total-cpu-usage---- -dsk/total- -net/total- ---paging-- ---system--
usr sys idl wai hiq sig| read  writ| recv  send|  in   out | int   csw
  0   0 100   0   0   0|  13k   10k|   0     0 |  15B   96B|   7    14
 25  27   0   0  11  37|   0     0 |  22M  122k|   0     0 |2333  1426
 22  23   9   0  13  32|   0    53M|  19M  143k|   0   508k|2037  1377
 22  26   1   0  12  38|   0   208k|  23M  174k|   0     0 |2425  1649
 14  12  40   1  13  19|   0    36k|  13M  127k|   0     0 |1164  1045
 18  16  16   0  24  25|4096B   16M|  18M  265k|   0     0 |1584  1822
 13  14  47   0   6  21|   0    39M|  13M  105k|   0     0 |1253   857
 23  27   0   0  12  37|   0     0 |  23M  113k|   0     0 |2248  1432
 23  30   0   0  10  37|   0    20k|  23M  113k|   0     0 |2305  1424
 12  11  48   0   9  19|   0    16M|  11M  128k|   0     0 |1133   959
 19  19  17   0  15  31|   0    56M|  18M  189k|   0     0 |1717  1388
  3   1  92   2   1   1| 428k    0 | 787k 5576B|  24k    0 | 136   216
  0   1  99   0   0   0|   0     0 | 108B   66B|   0     0 |   8     9
```

sar

- System Activity Reporter. Eg, paging statistics -B:

```
$ sar -B 1
Linux 3.2.6-3.fc16.x86_64 (node104)     02/20/2013 _x86_64_    (1 CPU)

05:24:34 PM  pgpgin/s pgpgout/s   fault/s  majflt/s  pgfree/s pgscank/s pgscand/s pgsteal/s  %vmeff
05:24:35 PM      0.00      0.00    267.68      0.00     29.29      0.00      0.00      0.00    0.00
05:24:36 PM     19.80      0.00    265.35      0.99     28.71      0.00      0.00      0.00    0.00
05:24:37 PM     12.12      0.00   1339.39      1.01   2763.64      0.00   1035.35   1035.35  100.00
05:24:38 PM      0.00      0.00    534.00      0.00     28.00      0.00      0.00      0.00    0.00
05:24:39 PM    220.00      0.00    644.00      3.00     74.00      0.00      0.00      0.00    0.00
05:24:40 PM   2206.06      0.00   6188.89     17.17   5222.22   2919.19      0.00   2919.19  100.00
[...]
```

- Configure to archive statistics from cron
- Many, many statistics available:
  - -d: block device statistics, -q: run queue statistics, ...
- Same statistics as shown by other tools (vmstat, iostat, ...)

netstat

- Various network protocol statistics using -s:

```
$ netstat -s
[...]
Tcp:
    127116 active connections openings
    165223 passive connection openings
    12904 failed connection attempts
    19873 connection resets received
    20 connections established
    662889209 segments received
    354923419 segments send out
    405146 segments retransmited
    6 bad segments received.
    26379 resets sent
[...]
TcpExt:
    2142 invalid SYN cookies received
    3350 resets received for embryonic SYN_RECV sockets
    7460 packets pruned from receive queue because of socket buffer overrun
    2932 ICMP packets dropped because they were out-of-window
    96670 TCP sockets finished time wait in fast timer
    86 time wait sockets recycled by time stamp
    1007 packets rejects in established connections because of timestamp
[...many...]
```

pidstat

- Very useful process breakdowns:

```
# pidstat 1
Linux 3.2.6-3.fc16.x86_64 (node107)     02/20/2013    _x86_64_ (1 CPU)

05:55:18 PM       PID    %usr %system  %guest    %CPU   CPU  Command
05:55:19 PM     12642    0.00    1.01    0.00    1.01     0  pidstat
05:55:19 PM     12643    5.05   11.11    0.00   16.16     0  cksum

05:55:19 PM       PID    %usr %system  %guest    %CPU   CPU  Command
05:55:20 PM     12643    6.93    6.93    0.00   13.86     0  cksum
[...]
```

```
# pidstat -d 1
Linux 3.2.6-3.fc16.x86_64 (node107)     02/20/2013    _x86_64_ (1 CPU)

05:55:22 PM       PID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
05:55:23 PM       279      0.00     61.90      0.00  jbd2/vda2-8
05:55:23 PM     12643 151985.71      0.00      0.00  cksum

05:55:23 PM       PID   kB_rd/s   kB_wr/s kB_ccwr/s  Command
05:55:24 PM     12643  96616.67      0.00      0.00  cksum
[...]
```

disk I/O (yay!)

strace

- System call tracer:

```
$ strace -tttT -p 12670
1361424797.229550 read(3, "REQUEST 1888 CID 2"..., 65536) = 959 <0.009214>
1361424797.239053 read(3, "", 61440)    = 0 <0.000017>
1361424797.239406 close(3)              = 0 <0.000016>
1361424797.239738 munmap(0x7f8b22684000, 4096) = 0 <0.000023>
1361424797.240145 fstat(1, {st_mode=S_IFCHR|0620, st_rdev=makedev(136, 0), ...}) = 0
<0.000017>
[...]
```

- -ttt: microsecond timestamp since epoch (left column)
- -T: time spent in syscall (<seconds>)
- -p: PID to trace (or provide a command)
- Useful – high application latency often caused by resource I/O, and *most* resource I/O is performed by syscalls

strace, cont.

- -c: print summary:

```
# strace -c dd if=/dev/zero of=/dev/null bs=512 count=1024k
[...]
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ----------------
 51.32    0.028376           0   1048581           read
 48.68    0.026911           0   1048579           write
  0.00    0.000000           0         7           open
[...]
```

- This is also a (worst case) demo of the strace *overhead*:

```
# time dd if=/dev/zero of=/dev/null bs=512 count=1024k
[...]
536870912 bytes (537 MB) copied, 0.35226 s, 1.5 GB/s
real 0m0.355s
user 0m0.021s
sys  0m0.022s
# time strace -c dd if=/dev/zero of=/dev/null bs=512 count=1024k
[...]
536870912 bytes (537 MB) copied, 71.9565 s, 7.5 MB/s
real 1m11.969s
user 0m3.179s
sys  1m6.346s
```

200x slower

tcpdump

- Sniff network packets, dump to output files for post analysis:

```
# tcpdump -i eth4 -w /tmp/out.tcpdump
tcpdump: listening on eth4, link-type EN10MB (Ethernet), capture size 65535
bytes
^C33651 packets captured
34160 packets received by filter
508 packets dropped by kernel

# tcpdump -nr /tmp/out.tcpdump
reading from file /tmp/out.tcpdump, link-type EN10MB (Ethernet)
06:24:43.908732 IP 10.2.0.2.55502 > 10.2.203.2.22: Flags [.], ack ...
06:24:43.908922 IP 10.2.0.2.55502 > 10.2.203.2.22: Flags [.], ack ...
06:24:43.908943 IP 10.2.203.2.22 > 10.2.0.2.55502: Flags [.], seq ...
06:24:43.909061 IP 10.2.0.2.55502 > 10.2.203.2.22: Flags [.], ack ...
```

- Output has timestamps with microsecond resolution
- Study odd network latency packet-by-packet
- Import file into other tools (wireshark)

tcpdump, cont.

- Does have overhead in terms of CPU and storage; previous example dropped packets
  - Should be using socket ring buffers to reduce overhead
  - Can use filter expressions to also reduce overhead
  - Could still be problematic for busy interfaces

blktrace

- Block device I/O event tracing. Launch using btrace, eg:

```
# btrace /dev/sdb
  8,16   3    1     0.429604145 20442  A   R 184773879 + 8 <- (8,17) 184773816
  8,16   3    2     0.429604569 20442  Q   R 184773879 + 8 [cksum]
  8,16   3    3     0.429606014 20442  G   R 184773879 + 8 [cksum]
  8,16   3    4     0.429607624 20442  P   N [cksum]
  8,16   3    5     0.429608804 20442  I   R 184773879 + 8 [cksum]
  8,16   3    6     0.429610501 20442  U   N [cksum] 1
  8,16   3    7     0.429611912 20442  D   R 184773879 + 8 [cksum]
  8,16   1    1     0.440227144     0  C   R 184773879 + 8 [0]
[...]
```

- Above output shows a single disk I/O event. Action time is highlighted (seconds).
- Use for investigating I/O latency outliers

iotop

- Disk I/O by process:

```
# iotop -bod5
Total DISK READ:       35.38 M/s | Total DISK WRITE:       39.50 K/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN      IO    COMMAND
12824 be/4 root       35.35 M/s    0.00 B/s  0.00 % 80.59 % cksum ...
  279 be/3 root        0.00 B/s   27.65 K/s  0.00 %  2.21 % [jbd2/vda2-8]
12716 be/4 root       28.44 K/s    0.00 B/s  2.35 %  0.00 % sshd: root@pts/0
12816 be/4 root        6.32 K/s    0.00 B/s  0.89 %  0.00 % python /usr/bin/
iotop -bod5
[...]
```

- IO: time thread was waiting on I/O (this is even more useful than pidstat's Kbytes)
- Needs CONFIG_TASK_IO_ACCOUNTING or something similar enabled to work.

slabtop

- Kernel slab allocator usage top:

```
# slabtop -sc
Active / Total Objects (% used)    : 900356 / 1072416 (84.0%)
 Active / Total Slabs (% used)      : 29085 / 29085 (100.0%)
 Active / Total Caches (% used)     : 68 / 91 (74.7%)
 Active / Total Size (% used)       : 237067.98K / 260697.24K (90.9%)
 Minimum / Average / Maximum Object : 0.01K / 0.24K / 10.09K

  OBJS ACTIVE  USE OBJ SIZE  SLABS OBJ/SLAB CACHE SIZE NAME
112035 110974  99%    0.91K  3201       35   102432K ext4_inode_cache
726660 579946  79%    0.11K 20185       36    80740K buffer_head
  4608   4463  96%    4.00K   576        8    18432K kmalloc-4096
 83496  76878  92%    0.19K  1988       42    15904K dentry
 23809  23693  99%    0.55K   821       29    13136K radix_tree_node
 11016   9559  86%    0.62K   216       51     6912K proc_inode_cache
  3488   2702  77%    1.00K   109       32     3488K kmalloc-1024
   510    431  84%    5.73K   102        5     3264K task_struct
 10948   9054  82%    0.17K   238       46     1904K vm_area_struct
  2585   1930  74%    0.58K    47       55     1504K inode_cache
[...]
```

- Shows where kernel memory is consumed

sysctl

- System settings:

```
# sysctl -a
[...]
net.ipv4.tcp_fack = 1
net.ipv4.tcp_reordering = 3
net.ipv4.tcp_ecn = 2
net.ipv4.tcp_dsack = 1
net.ipv4.tcp_mem = 24180   32240    48360
net.ipv4.tcp_wmem = 4096   16384    1031680
net.ipv4.tcp_rmem = 4096   87380    1031680
[...]
```

- Static performance tuning: check the config of the sysetm

/proc

- Read statistic sources directly:

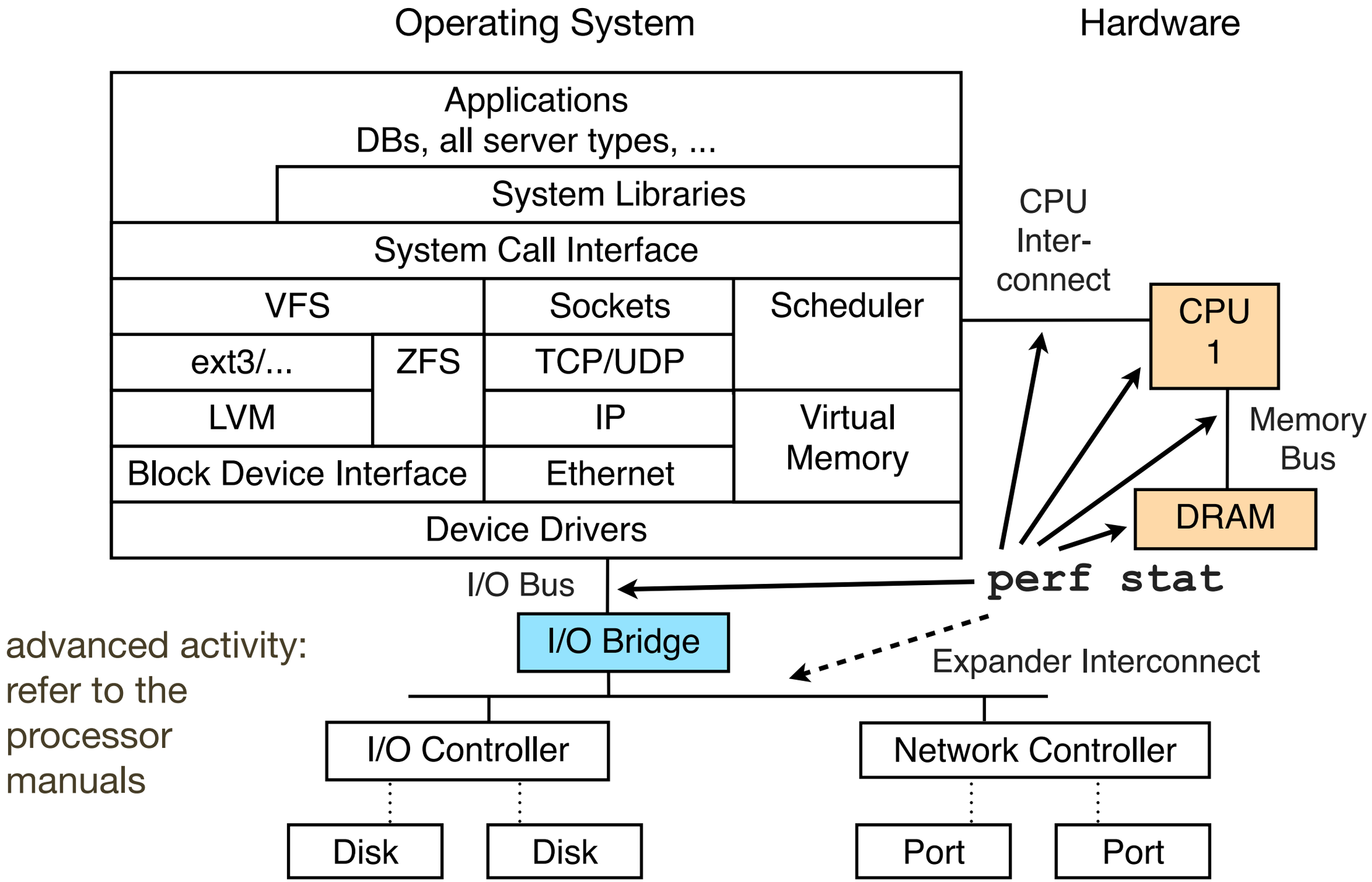
```
$ cat /proc/meminfo
MemTotal:        8181740 kB
MemFree:           71632 kB
Buffers:          163288 kB
Cached:          4518600 kB
SwapCached:         7036 kB
Active:          4765476 kB
Inactive:        2866016 kB
Active(anon):    2480336 kB
Inactive(anon):   478580 kB
Active(file):    2285140 kB
Inactive(file):  2387436 kB
Unevictable:           0 kB
Mlocked:               0 kB
SwapTotal:       2932728 kB
SwapFree:        2799568 kB
Dirty:                76 kB
Writeback:             0 kB
[...]
```

- Also see /proc/vmstat

Power Tools

- perf
- DTrace
- SystemTap
- LTTng
- ktap

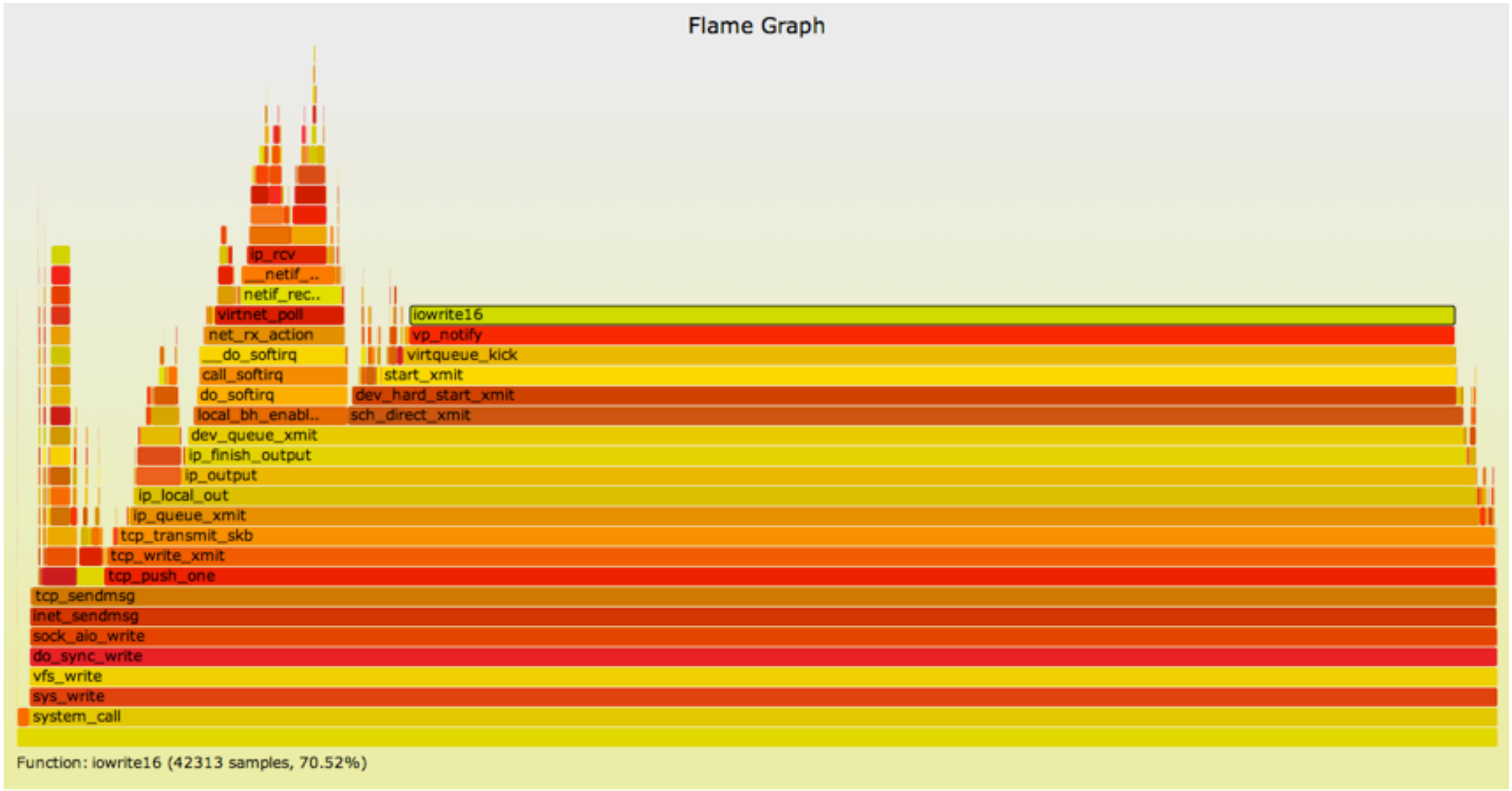
perf: CPU counter analysis



advanced activity: refer to the processor manuals

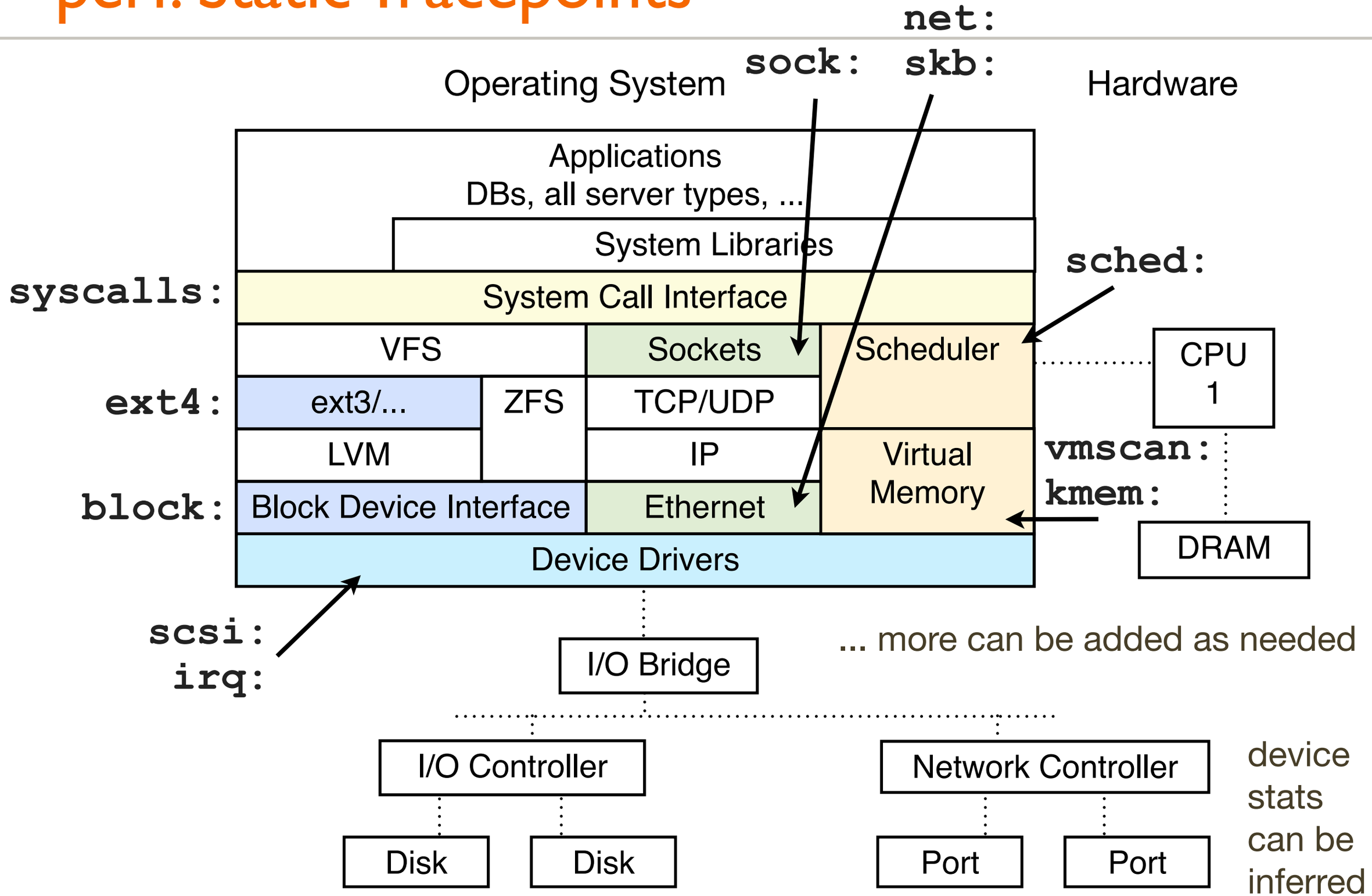
perf: Profiling, cont.

- Flame Graphs support perf profiling data:



- Interactive SVG. Navigate to quantify and compare code paths

perf: Static Tracepoints



perf: Dynamic Tracing

- Define custom probes from kernel code; eg, tcp_sendmsg():

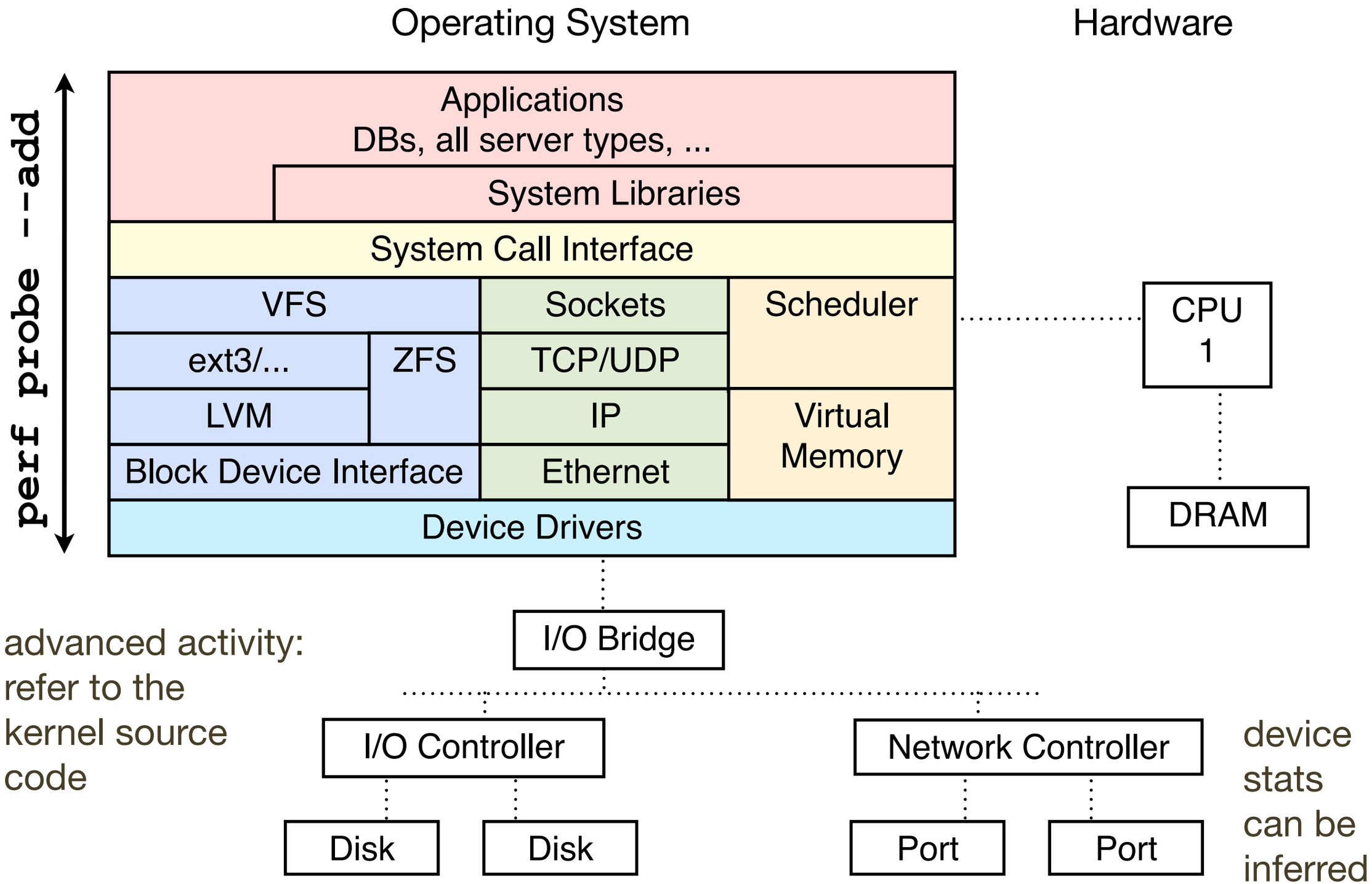
```
# perf probe --add='tcp_sendmsg'
Add new event:
  probe:tcp_sendmsg    (on tcp_sendmsg)
[...]

# perf record -e probe:tcp_sendmsg -aR -g sleep 5
[ perf record: Woken up 1 times to write data ]
[ perf record: Captured and wrote 0.091 MB perf.data (~3972 samples) ]

# perf report --stdio
[...]
# Overhead  Command      Shared Object        Symbol
# ........  .......  ..................  ............
#
   100.00%      sshd  [kernel.kallsyms]  [k] tcp_sendmsg
                |
                --- tcp_sendmsg
                    sock_aio_write
                    do_sync_write
                    vfs_write
                    sys_write
                    system_call
                    __GI___libc_write
```

active traced call stacks from arbitrary kernel locations!

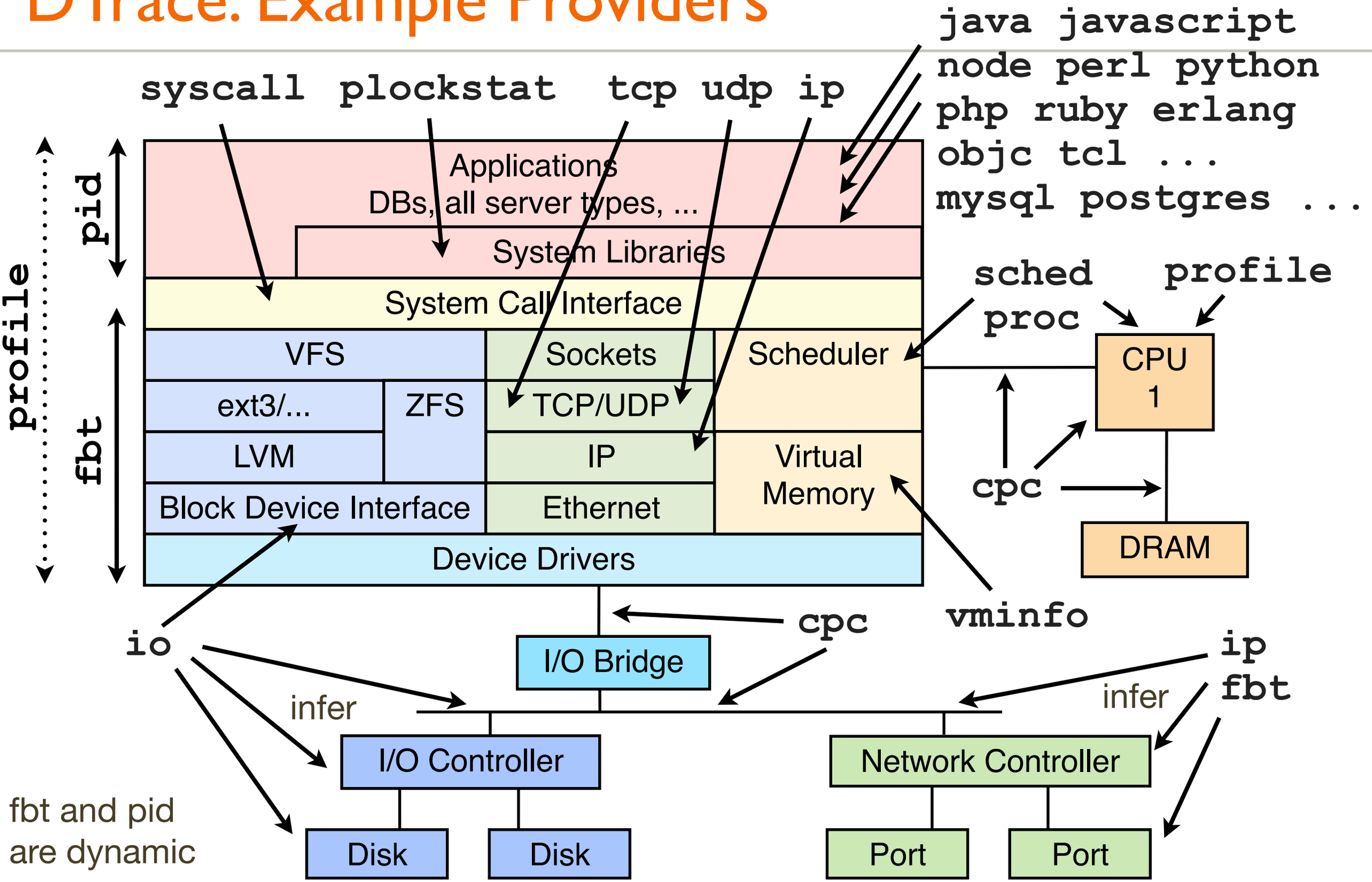
perf: Dynamic Tracing



advanced activity: refer to the kernel source code

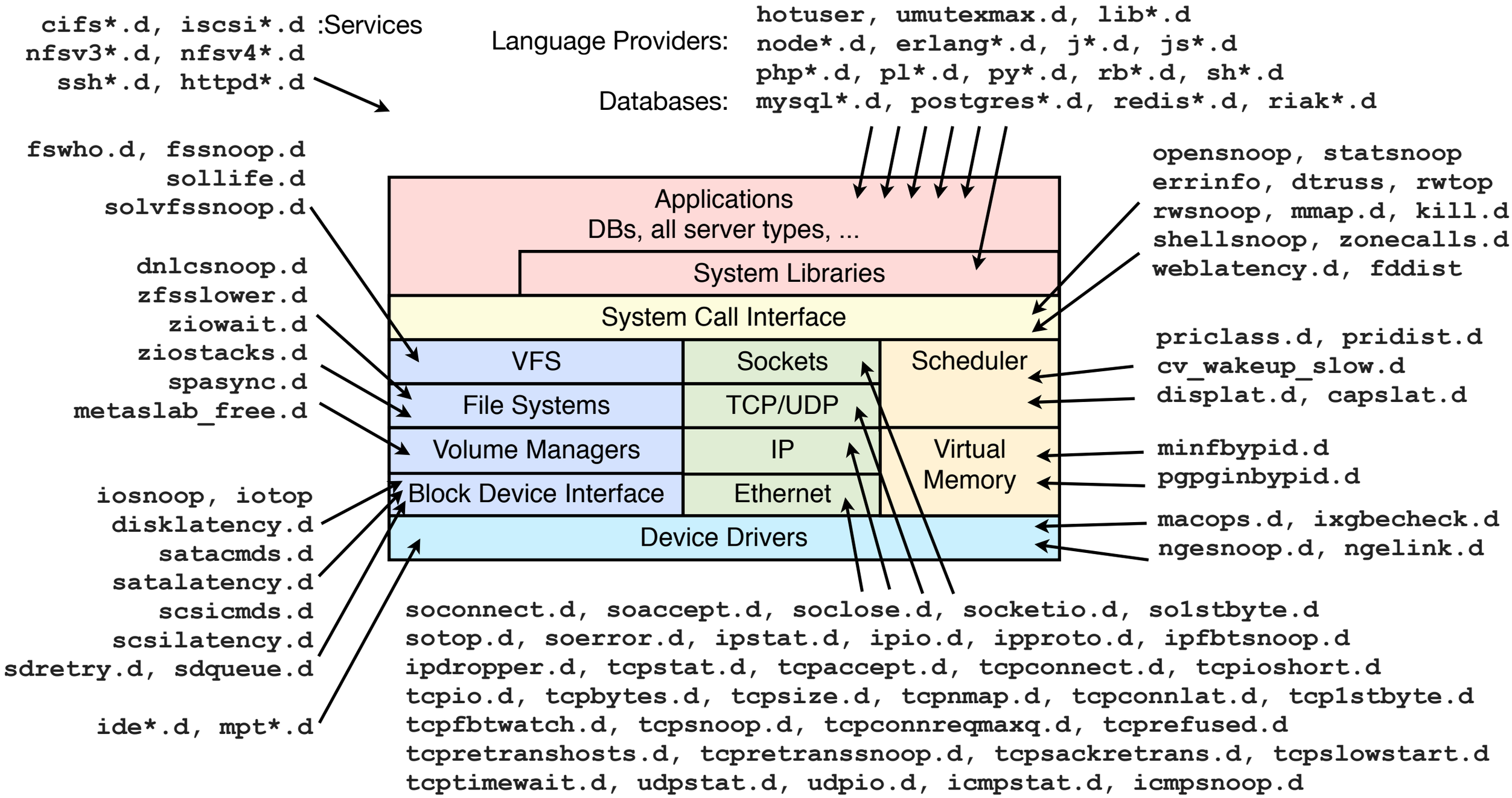
device stats can be inferred

DTrace: Example Providers



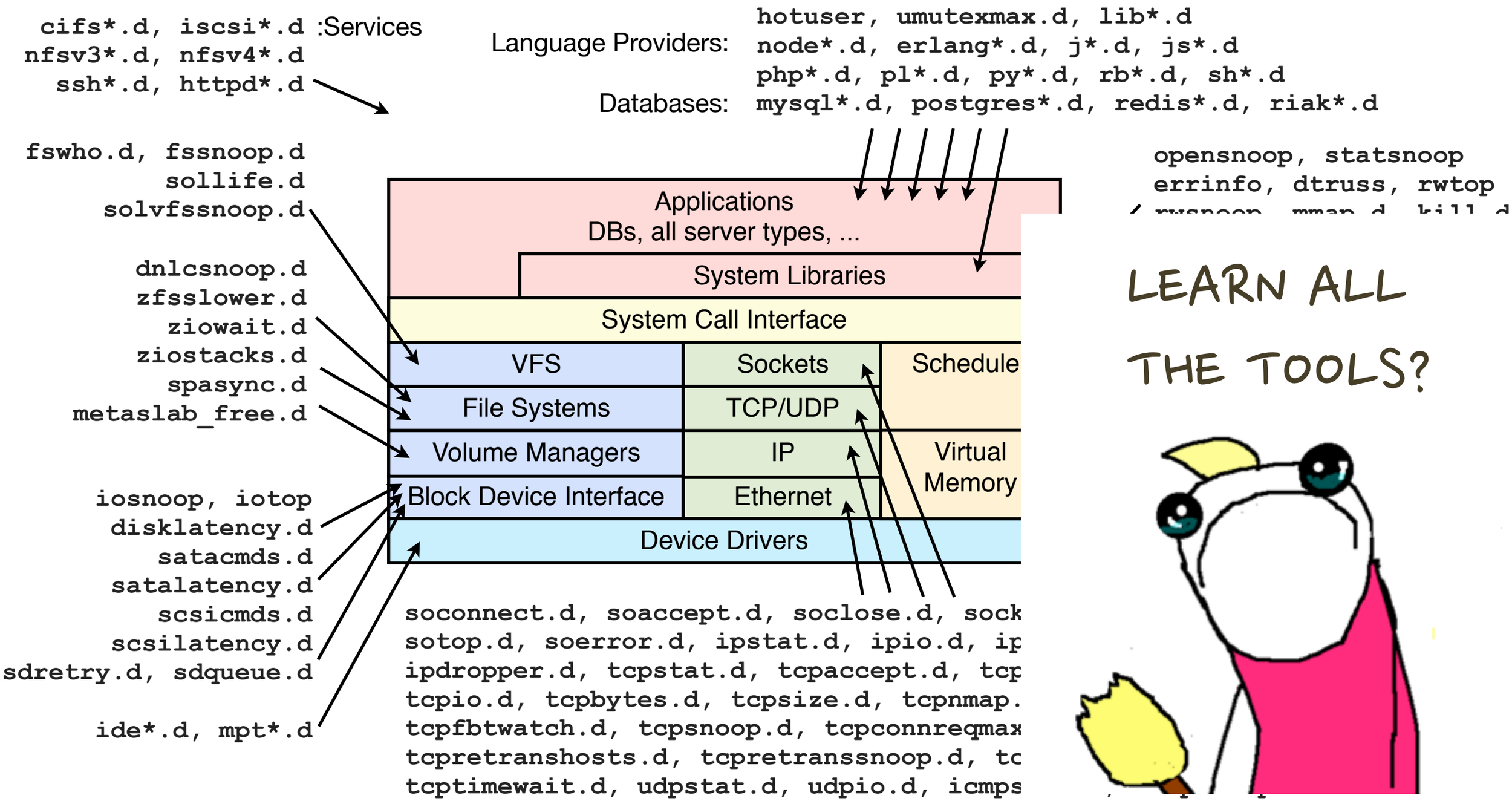
Dynamic Tracing Examples: SmartOS

- Example DTrace scripts from the DTraceToolkit, DTrace book, ...



Even More... Dynamic Tracing: SmartOS

- Example DTrace scripts from the DTraceToolkit, DTrace book, ...



Tool Focus: Problems

- Too many tools
  - Overlapping functionality, duplication
  - Time consuming to learn and wade through
  - Confusing for beginners: why do these similar tools exist?
- Not enough unique metrics
  - Observablity gaps, usually due prohibitive difficulty (especially CPU performance counters)

Methodologies

- Streetlight Anti-Method
- Tools Method
- USE Method
- TSA Method

Streetlight Anti-Method

- 1. Pick observability tools that are
  - familiar
  - found on the Internet
  - found at random
- 2. Run tools
- 3. Look for obvious issues

- Included for comparison (don’t use this methodology)

Tools Method

- 1. List available performance tools
- 2. For each tool, list useful metrics it provides
- 3. For each metric, list possible rules for interpretation

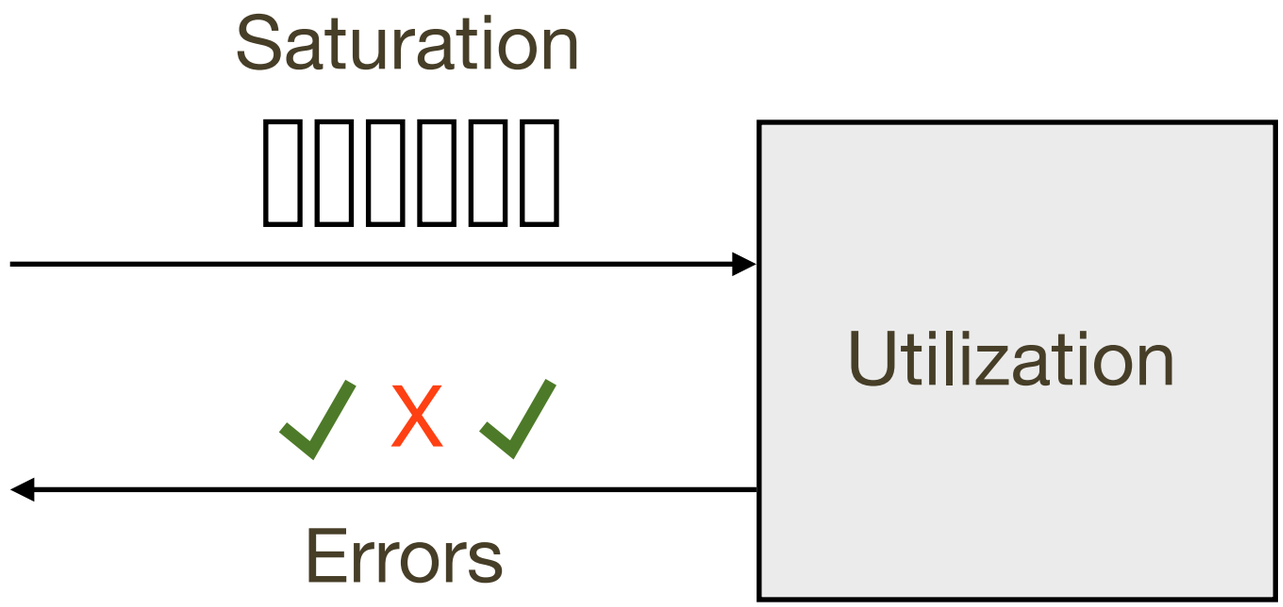
- Pros:
  - Prescriptive
- Cons:
  - Time consuming
  - Can miss issues

USE Method

- For every resource, check:
- 1. Utilization
- 2. Saturation
- 3. Errors

USE Method

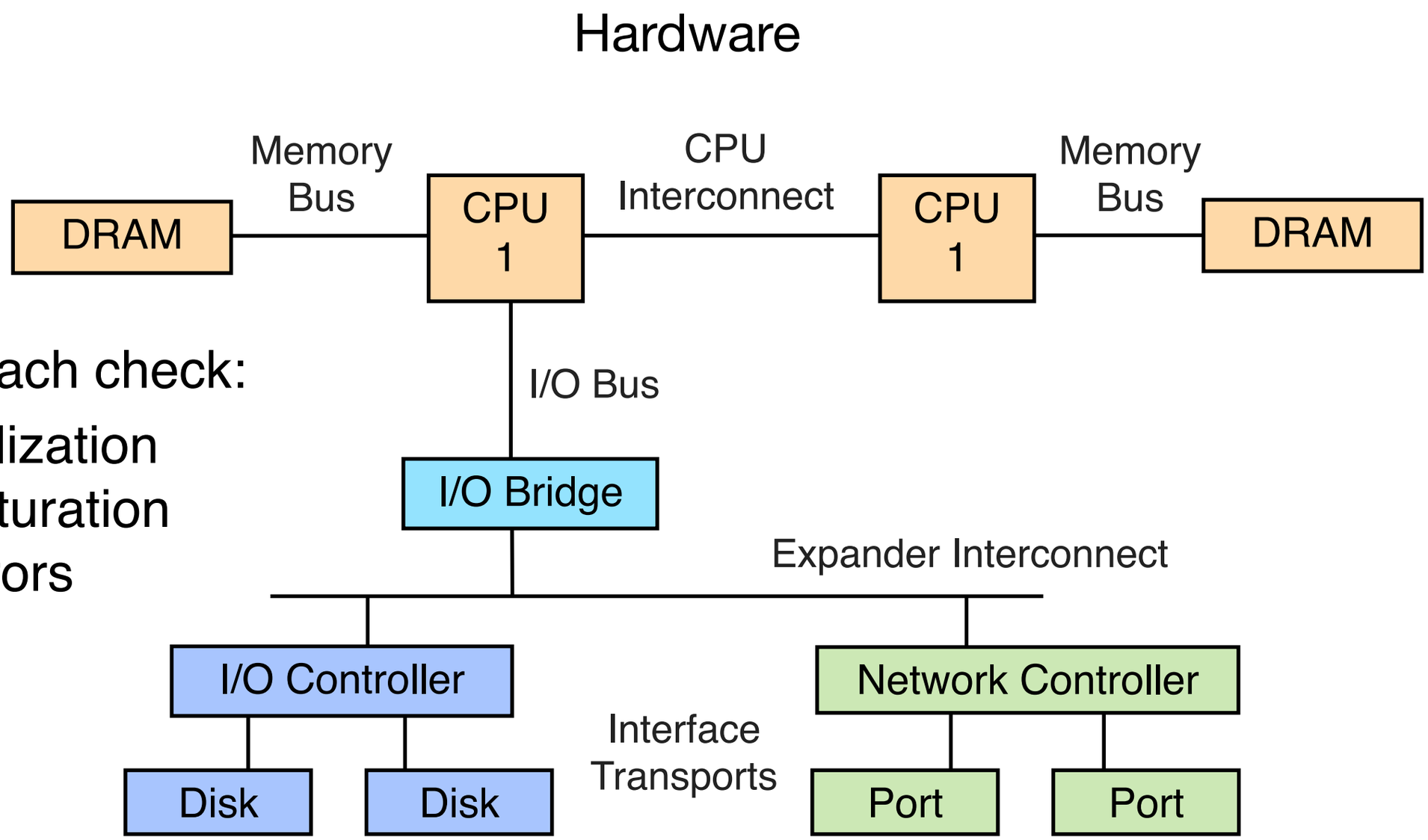
- For every resource, check:
- 1. Utilization: time resource was busy, or degree used
- 2. Saturation: degree of queued extra work
- 3. Errors: any errors



USE Method

For each check:

1. Utilization
2. Saturation
3. Errors



USE Method

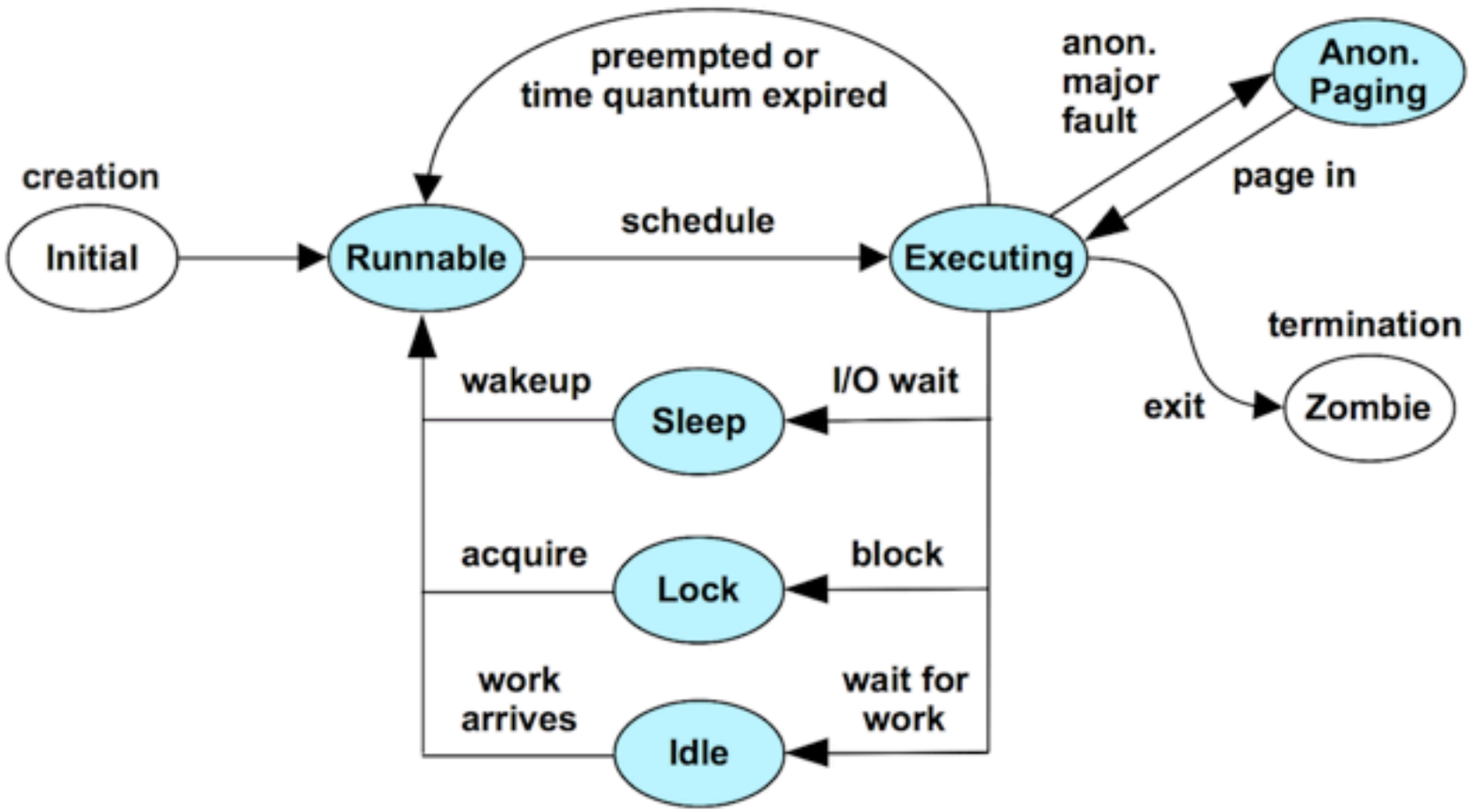
- This process may reveal *missing metrics* – those not provided by your current toolset
  - They are your *known unknowns*
  - Much better than *unknown unknowns*
- More tools can be installed and developed to help
  - So many top(1)s, but where is the *interconnect*-top?
- Full USE Method checklist may, practically, only be used for critical issues

USE Method

- Resource-based approach
- Quick system health check, early in an investigation
- Pros:
    - Complete: all resource bottlenecks and errors
    - Not limited in scope by your current toolset
    - No unknown unknowns – at least known unknowns
    - Efficient: picks three metrics for each resource – from what may be dozens available
- Cons:
    - Limited to a class of issues

TSA Method

- Thread State Analysis (TSA) Method:
- 1. For each thread of interest, measure total time in different thread states
- 2. Investigate states from the most to the least frequent, using appropriate tools



TSA Method

- Thread states:
  - **Executing**: on-CPU
  - **Runnable**: and waiting for a turn on-CPU
  - **Anonymous Paging**: (aka swapping) runnable, but blocked waiting for residency
  - **Sleeping**: waiting for I/O, including network, block, and data/text page-ins
  - **Lock**: waiting to acquire a synchronization lock (waiting on someone else)
  - **Idle**: waiting for work

TSA Method

- Pros:
  - Directs analysis quickly to problem areas
- Cons:
  - States can be difficult to measure on Linux

More Performance Methodologies

- Workload characterization
- USE Method
- TSA Method
- Drill-down Analysis
- Latency Analysis
- Event Tracing
- Static performance tuning
- ... in Chapter 2 of SystemsPerformance:

Table 2.4 Generic System Performance Methodologies (*Continued*)

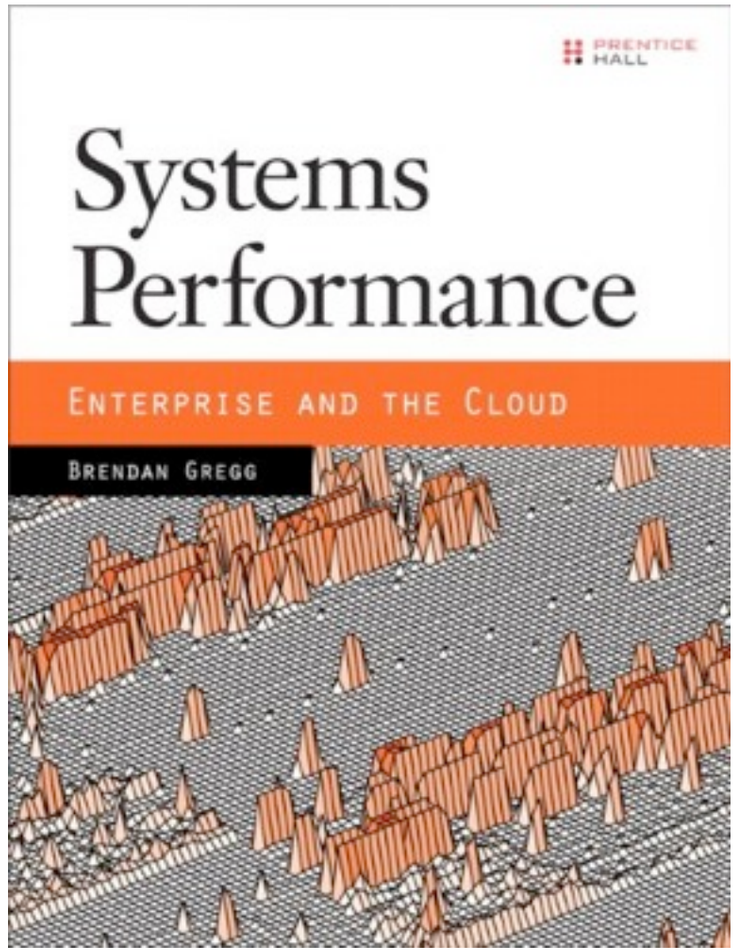
Methodology	Type
Scientific method	observational analysis
Diagnosis cycle	analysis life cycle
Tools method	observational analysis
USE method	observational analysis
Workload characterization	observational analysis, capacity planning
Drill-down analysis	observational analysis
Latency analysis	observational analysis
Method R	observational analysis
Event tracing	observational analysis
Baseline statistics	observational analysis
Performance monitoring	observational analysis, capacity planning
Queueing theory	statistical analysis, capacity planning
Static performance tuning	observational analysis, capacity planning
Cache tuning	observational analysis, tuning
Micro-benchmarking	experimental analysis
Capacity planning	capacity planning, tuning

Question Focus

- Example implementation of methodologies:
- The USE Method, Linux:
  - http://dtrace.org/blogs/brendan/2012/03/07/the-use-method-linux-performance-checklist/
- The TSA Method:
  - http://dtrace.org/blogs/brendan/2013/10/11/the-tsa-method/
- DEMO

Thank You!

- Methodologies and tools are covered in my new book:
  - Systems Performance: Enterprise and the Cloud
- Many checklists and articles on performance are on my blog and homepage:
  - http://dtrace.org/blogs/brendan
  - http://www.brendangregg.com
- twitter: @brendangregg



Sample Chapter:

http://dtrace.org/blogs/brendan/2013/06/21/systems-performance-enterprise-and-the-cloud/