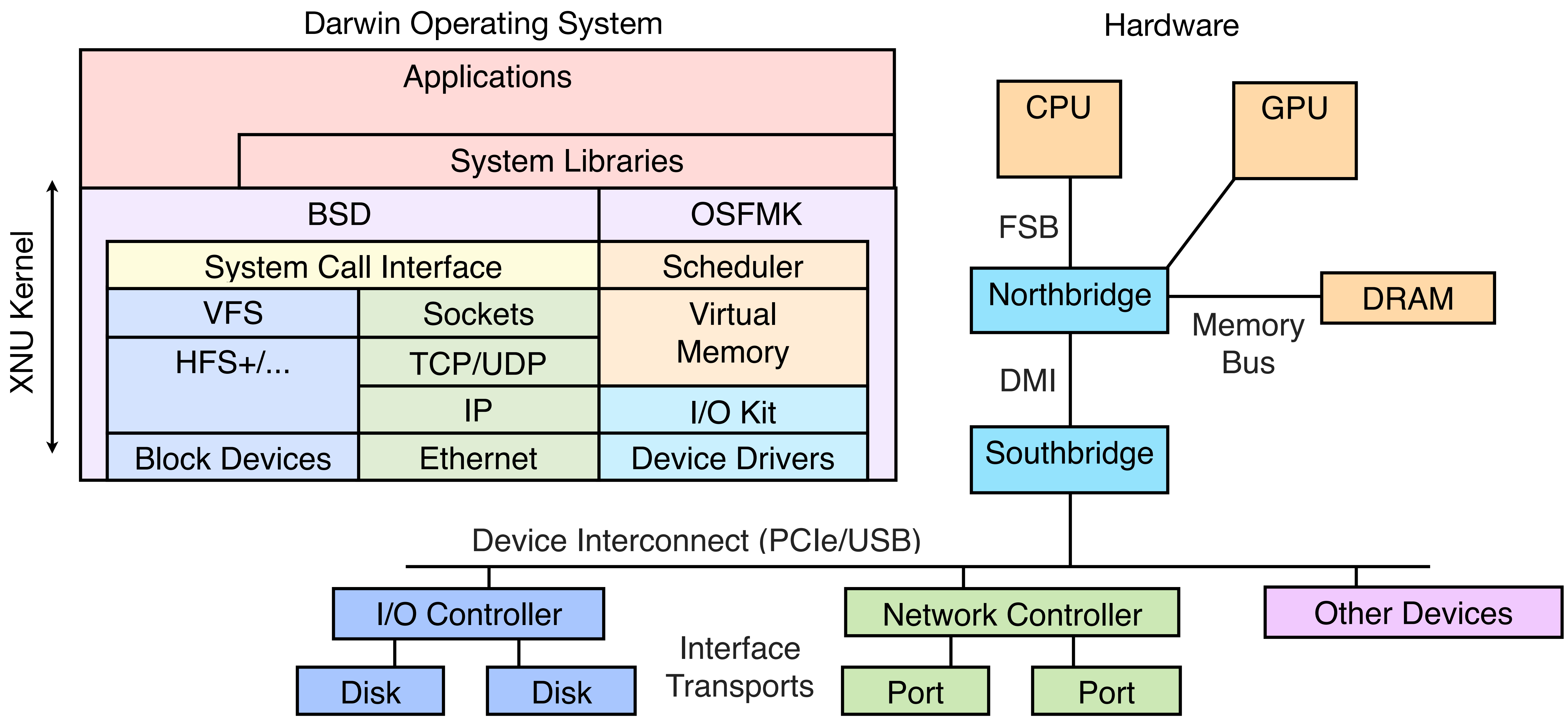
The World's Leading Conference for Deploying
iOS and OS X in the Enterprise

Analyzing OS X Systems Performance with the USE Method

Brendan Gregg, Senior Performance Architect, Netflix
March, 2014

Find the Bottleneck

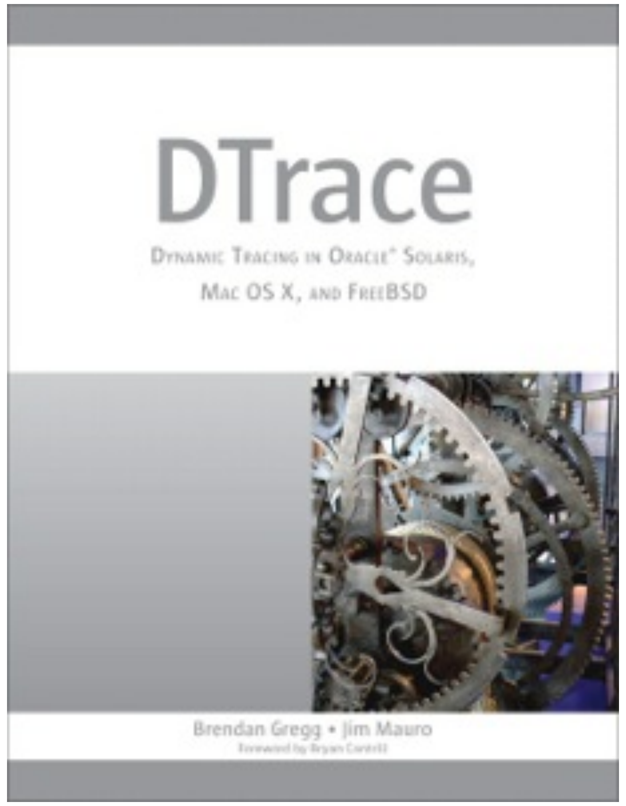


This Talk

- Summarizes casual to serious performance analysis of OS X
- From the *systems perspective*, not the application
  - Many application issues can be found easily this way
- Covering not just current tools, but suggestions for future work
- May change how you think about performance!

whoami

- Senior Performance Architect at Netflix
- Primary author of the DTrace book
- Wrote many DTrace scripts included with OS X.
  Eg: dtruss, iosnoop, iotop, opensnoop, execsnoop, procsystime, bitesize.d, seeksize.d, setuids.d, etc...
  - These were ported and enhanced by Apple engineering (thanks!)
- Created the USE method and USE method checklist for OS X



Agenda

- The Tools Method
- The USE Method
- Future work

The Tools Method

The Tools Method

- A tool-based performance analysis approach, commonly followed today. For reference, I've called it the "Tools Method".
  - 1. List available performance tools
  - 2. For each tool, list its useful metrics
  - 3. For each metric, list possible interpretation
- Simple, useful, but analysis is limited to what the tools provide easily

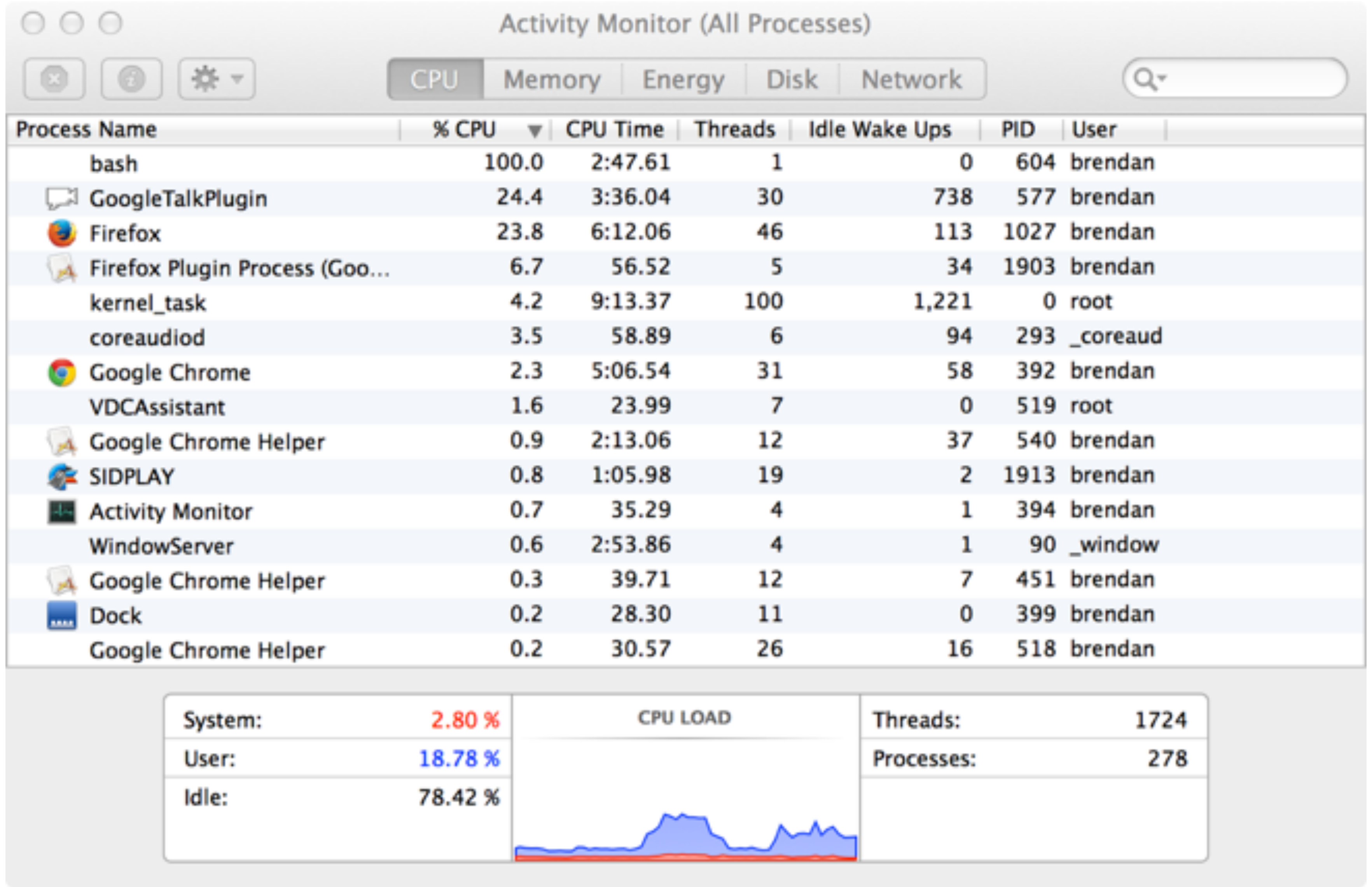
Tool Examples

- Activity Monitor
- atMonitor, Temperature Monitor Lite
- Command Line
- DTrace
- Instruments

MacIT

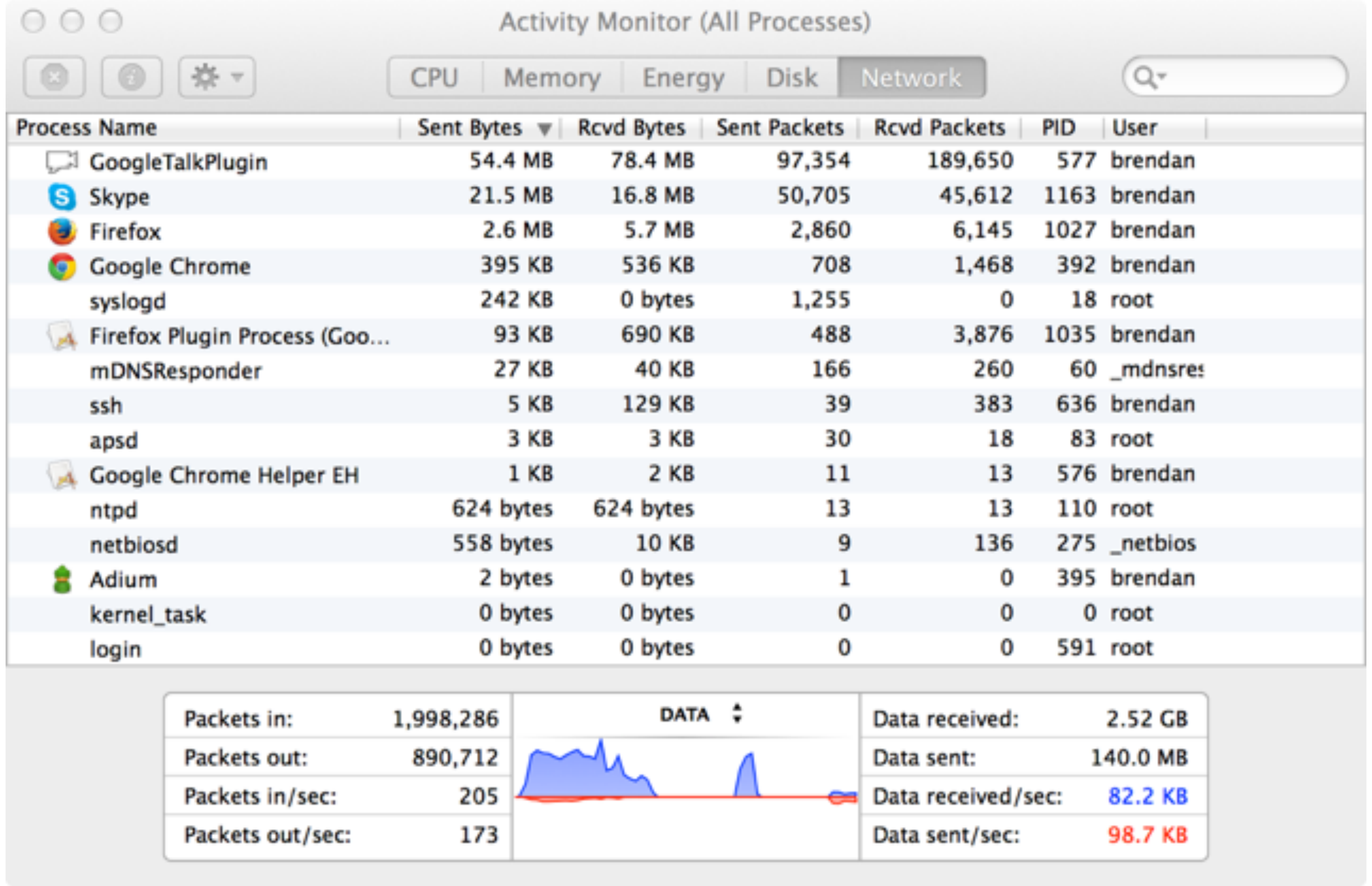
Activity Monitor

- High level process and system sumaries. A GUI version of top(1)
- Table shows processes by %CPU, memory
- CPU load over time
- Quit, info, and system diagnosis buttons



Activity Monitor Network

- Quick way to see current and recent network throughput
- Like the CPU summary, shows aggregate device stats, and not per-device


Activity Monitor (All Processes)

CPU | Memory | Energy | Disk | Network

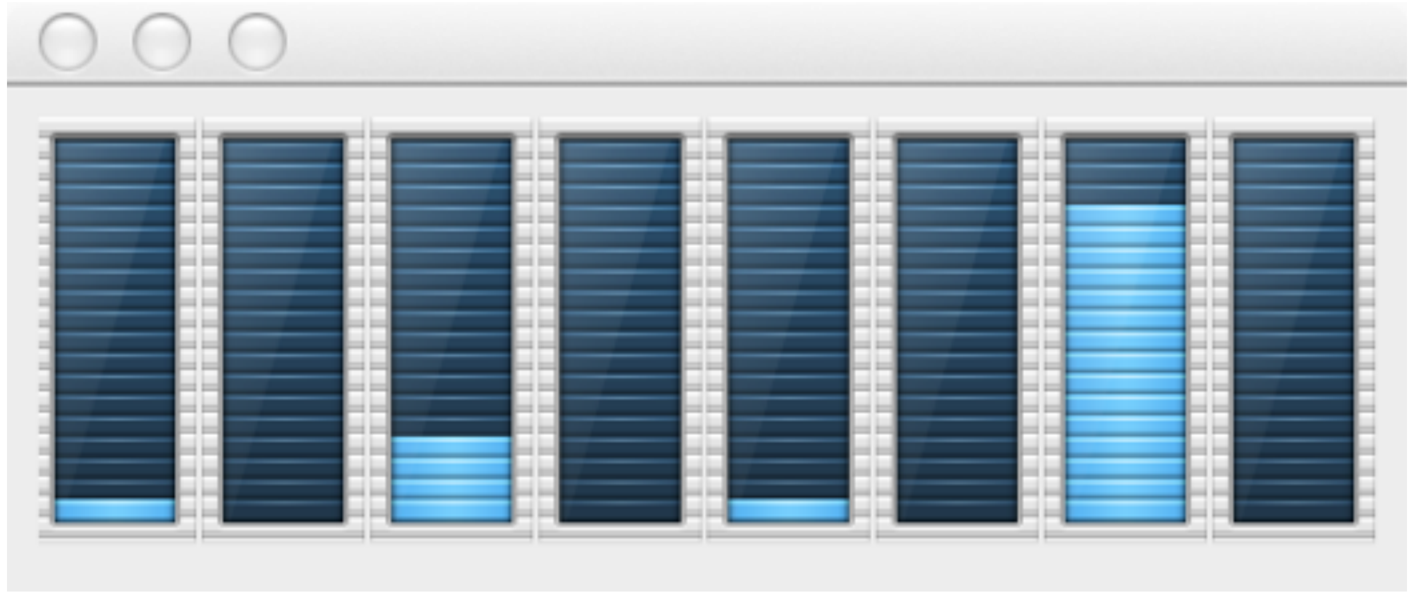
Process Name	Sent Bytes	Rcvd Bytes	Sent Packets	Rcvd Packets	PID	User
GoogleTalkPlugin	54.4 MB	78.4 MB	97,354	189,650	577	brendan
Skype	21.5 MB	16.8 MB	50,705	45,612	1163	brendan
Firefox	2.6 MB	5.7 MB	2,860	6,145	1027	brendan
Google Chrome	395 KB	536 KB	708	1,468	392	brendan
syslogd	242 KB	0 bytes	1,255	0	18	root
Firefox Plugin Process (Goo…	93 KB	690 KB	488	3,876	1035	brendan
mDNSResponder	27 KB	40 KB	166	260	60	_mdnsres
ssh	5 KB	129 KB	39	383	636	brendan
apsd	3 KB	3 KB	30	18	83	root
Google Chrome Helper EH	1 KB	2 KB	11	13	576	brendan
ntpd	624 bytes	624 bytes	13	13	110	root
netbiosd	558 bytes	10 KB	9	136	275	_netbios
Adium	2 bytes	0 bytes	1	0	395	brendan
kernel_task	0 bytes	0 bytes	0	0	0	root
login	0 bytes	0 bytes	0	0	591	root

Packets in:	1,998,286
Packets out:	890,712
Packets in/sec:	205
Packets out/sec:	173

DATA

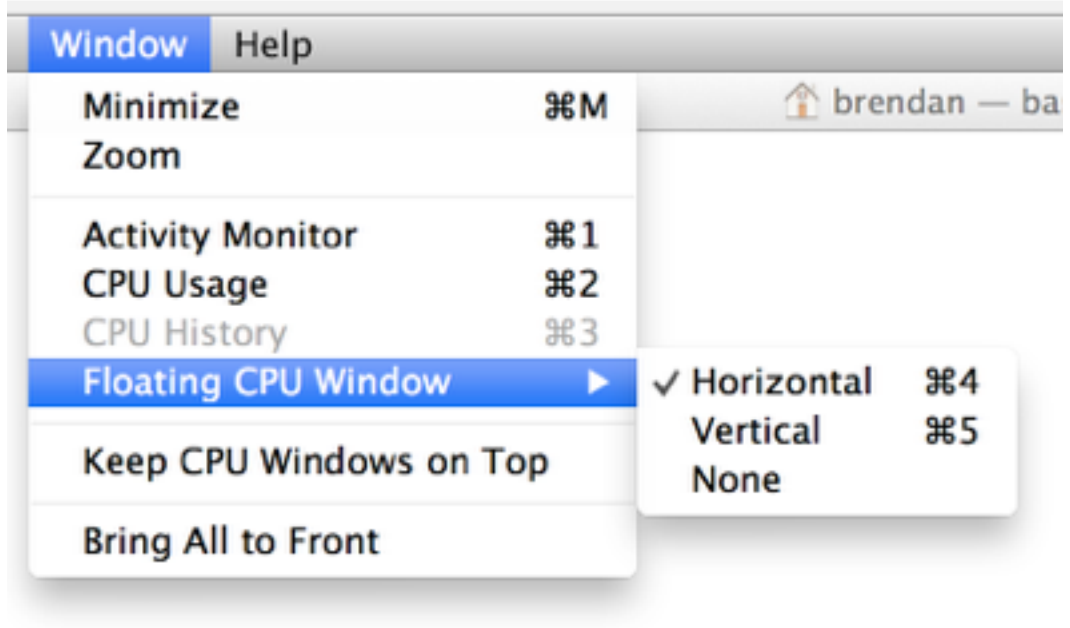
Data received:	2.52 GB
Data sent:	140.0 MB
Data received/sec:	82.2 KB
Data sent/sec:	98.7 KB

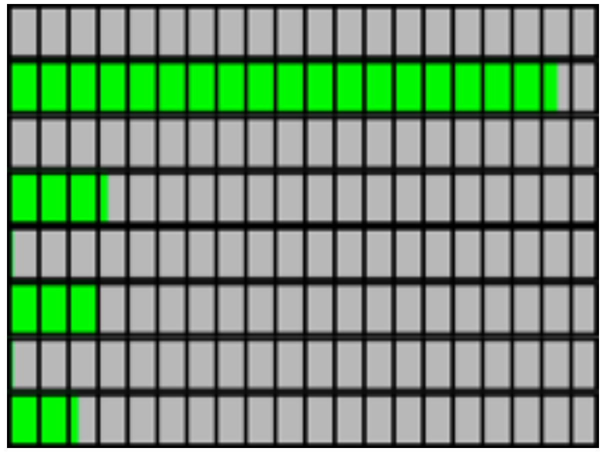
Activity Monitor CPU Usage



- Per-CPU utilization from previous 0.5 - 5 seconds (tunable)
- Handy to leave running. Look for single hot CPUs/threads

Activity Monitor
Floating CPU Window

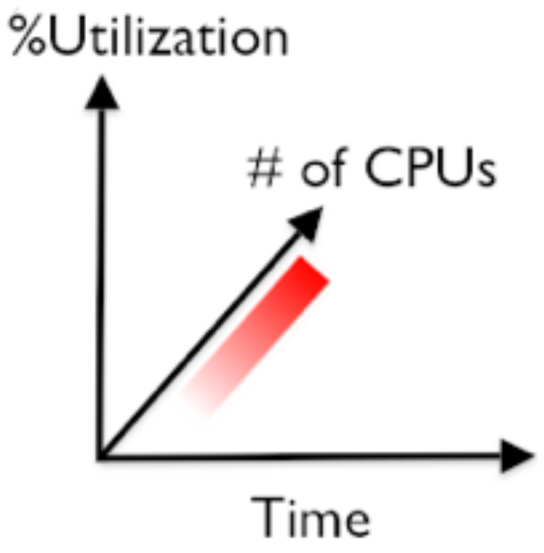




- Earlier OS X also had a compact version (gone in Mavericks)
- Was nice, but what I really want is a compact visualization for both per-CPU *and* historical data

Activity Monitor CPU/Disk Suggestion

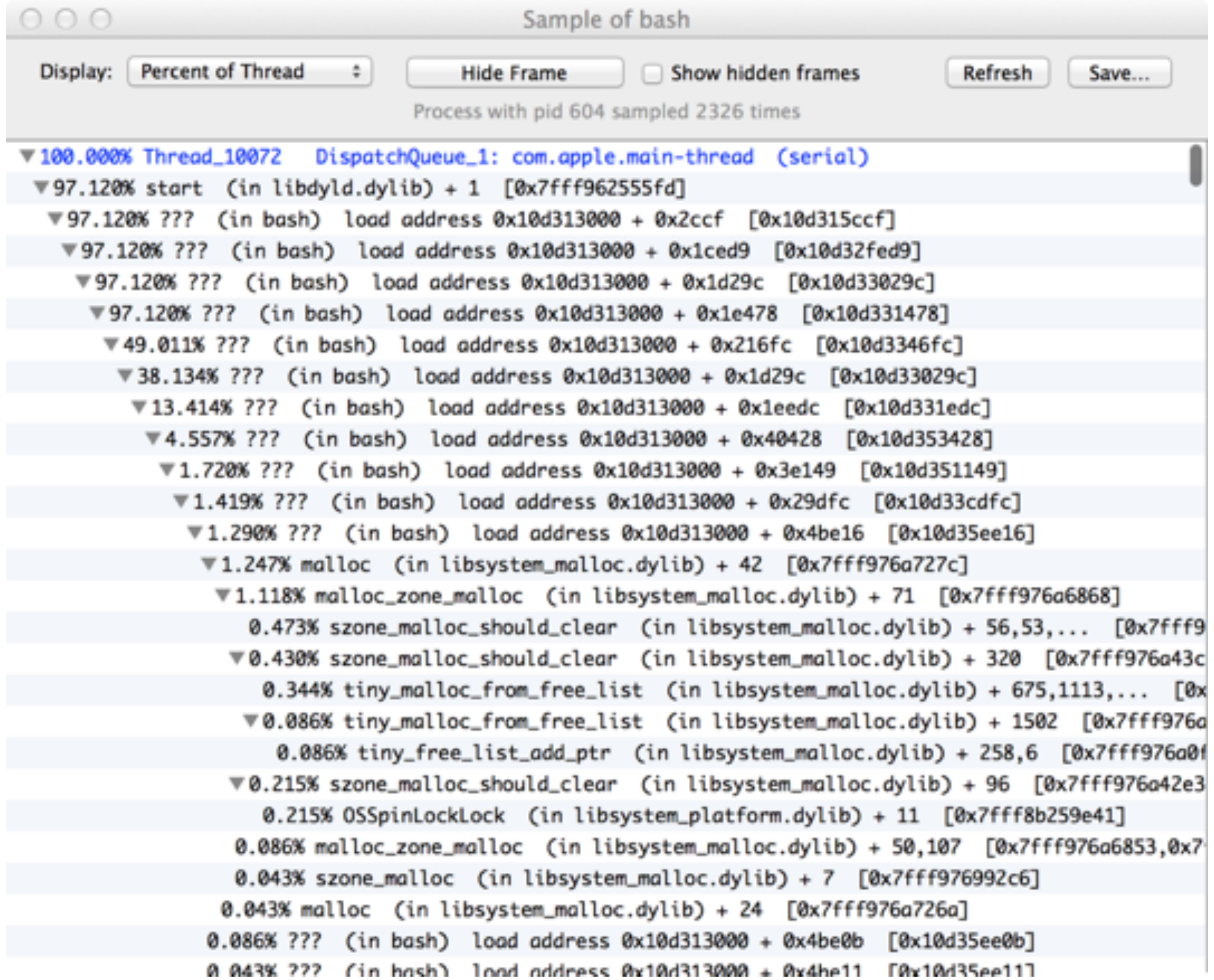
- Could show both per-device and history using a *utilization heat map*:



- http://dtrace.org/blogs/brendan/2011/12/18/visualizing-device-utilization/

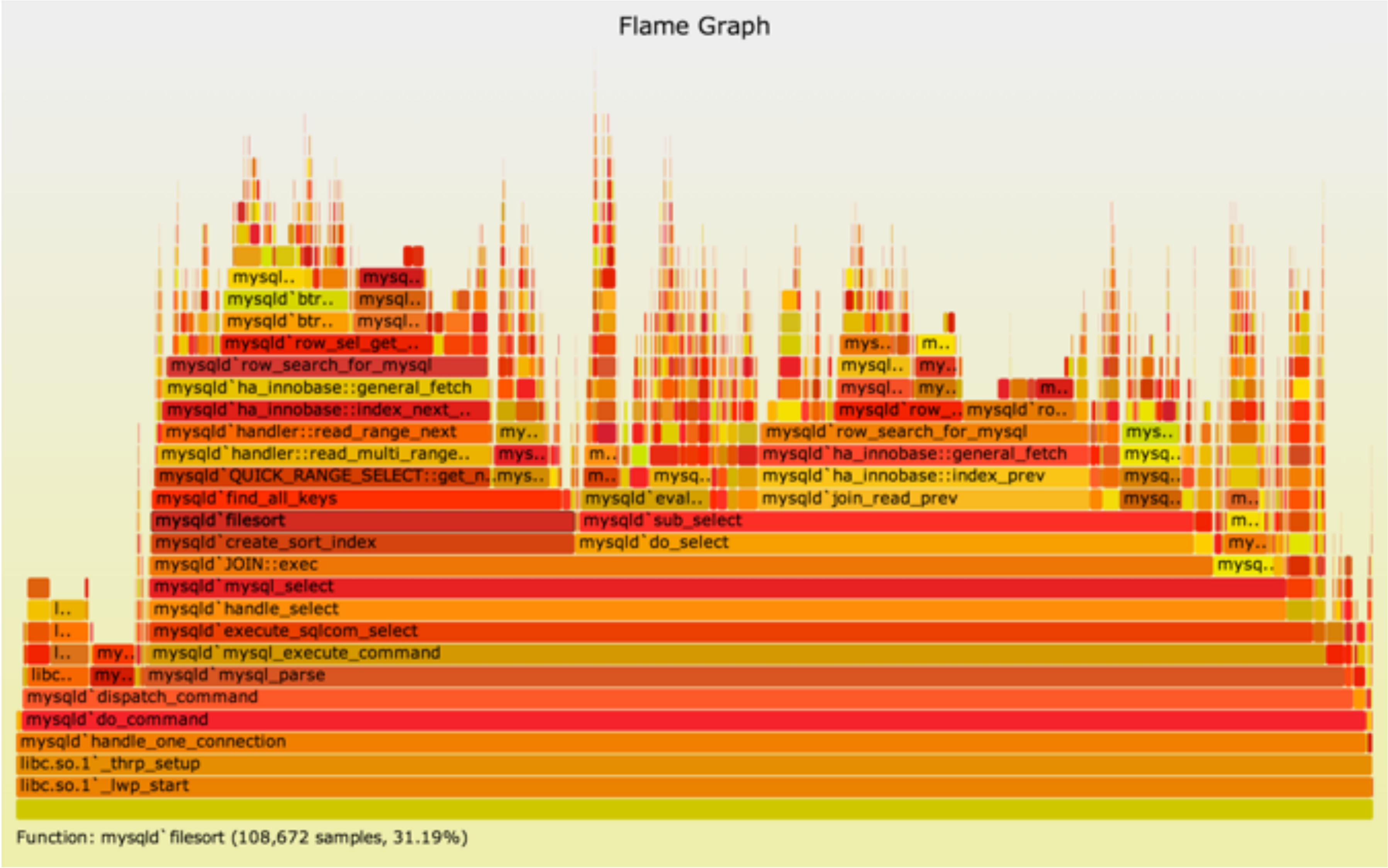
Activity Monitor
Sample Process

- The cog button ("System diagnostics options") has a "Sample process" option for profiling CPU code paths
- Explains %CPU usage
- Although output usually very long and time consuming to read (see scroll bar):

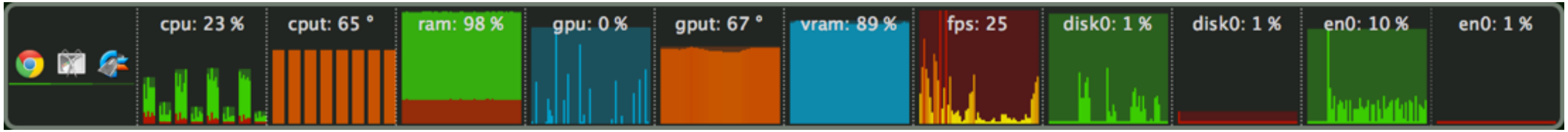


Activity Monitor Flame Graphs ?

- Suggestion: include a Flame Graph view
- Visualizes entire profile output in one screen
- http://github.com/brendangregg/FlameGraph



atMonitor



- 3rd party app. Version 2.7b crashes for me if "Top Window" is visible.
- Shows many useful metrics: per-CPU, RAM, GPU, per-disk, and per-network interface utilization perentages with histories.
- Currently the easiest way to see GPU, disk, and network utilization.
- Utilization is easy to interpret. I/O per second is not.

Temperature Monitor Lite

- Another 3rd party application
- Easy way to infer GPU utilization
- Normal: GPU1: 57℃
- Video: GPU1: 81℃

Command Line

- Accessed via the Terminal application
- Numerous performance tools available, from UNIX/BSD/OSX
- Eg, the uptime(1) command shows recent and historic CPU load:

```
$ uptime
14:36  up 43 days,  2:39, 30 users, load averages: 0.72 1.02 1.29
```

- There numbers are the 1, 5, and 15 minute load averages. Values are really constants in an exponential decay moving sum.
- Interpret: if average > number of CPUs, then CPUs are overloaded

Command Line: top

- top(1): high level process and system summary:

```
$ top -o cpu
Processes: 272 total, 4 running, 268 sleeping, 1546 threads              14:47:36
Load Avg: 1.14, 0.75, 0.95  CPU usage: 13.95% user, 2.78% sys, 83.26% idle
SharedLibs: 12M resident, 5112K data, 0B linkedit.
MemRegions: 339218 total, 6689M resident, 184M private, 2153M shared.
PhysMem: 3429M wired, 6502M active, 5910M inactive, 15G used, 537M free.
VM: 552G vsize, 1052M framework vsize, 111312590(1) pageins, 1437348(0) pageouts
Networks: packets: 120030109/127G in, 70582570/38G out.
Disks: 22089197/1050G read, 26756359/1163G written.

PID    COMMAND      %CPU  TIME     #TH   #WQ  #PORT #MREGS  RPRVT  RSHRD  RSIZE
602    bash         100.0 47:42.28 1/1   0    21    27      236K   816K   760K
94370  top          17.2  00:03.77 1/1   0    24    39      4368K  216K   5116K
52617  firefox      6.3   47:30:58 45/1  2    576-  177307+ 1984M+ 200M   2530M+
92489- Google Chrom 2.2   13:31.85 34    2    530   2454    273M   271M   734M
[...]
```

hey...

Command Line: vm_stat

- vm_stat(1): virtual memory statistics, including free memory, paging

```
$ vm_stat 1
Mach Virtual Memory Statistics: (page size of 4096 bytes, cache hits 0%)
  free active   spec inactive   wire   faults     copy    0fill reactive   pageins  pageout
101297  1662K  29920  1509998 888520   17650M  106072K   15926M  6833792   111312K  1437348
100919  1658K  29920  1509998 893230     2851        0     2043        0         0        0
101183  1658K  29918  1509998 893169      143        0       87        0         1        0
100517  1658K  29921  1509998 893354      396        3      136        0         2        0
 96590  1657K  29923  1514414 894426     5888       94     5146        0         2        0
 93184  1662K  28486  1514414 894484    14183      117    12521        0         0        0
 91224  1663K  28486  1514414 894886     5683        0     3454        0         0        0
 89195  1649K  29924  1514413 909225    11570      199    10050        0         4        0
 87550  1636K  29917  1514155 923179    24486     1432    12009        0      2134        0
 61596  1644K  28309  1515551 941688    49395     1446    46127        0      4941        0
 52932  1669K  28442  1515663 925755    70618     1731    53131        0      1221        0
 76395  1681K  28417  1515685 889983    30514        0    28072        0       428        0
 73520  1679K  28449  1515777 894905    20082       17    18077        0       107        0
 60335  1684K  29073  1515560 903152    39696       38    35535        0      1309        0
[...]
```

Command Line: iostat

- iostat(1): block device I/O statistics. Disks, USB drives.

```
$ iostat 1
              disk0               disk2          cpu      load average
    KB/t tps   MB/s     KB/t tps   MB/s   us  sy id    1m    5m   15m
   47.03  13   0.60    96.67   0   0.00    5   2 92  0.94 1.01 0.99
  972.42  19  18.02   128.00 141  17.60    2   3 95  0.94 1.01 0.99
  315.60  10   3.08   128.00  24   3.00    6   2 92  0.94 1.01 0.99
    4.00   1   0.00     0.00   0   0.00    6   2 92  0.94 1.01 0.99
 1024.00   8   7.99   128.00  69   8.61    6   2 92  0.94 1.01 0.99
 1024.00  18  17.97   128.00 143  17.85    2   2 95  0.86 0.99 0.99
 1024.00  17  16.98   128.00 142  17.72    2   2 96  0.86 0.99 0.99
  165.27 272  43.84   127.13 146  18.10    6   5 89  0.95 1.01 0.99
 1024.00  18  17.98   128.00 143  17.85    2   2 96  0.95 1.01 0.99
[...]
```

- No percent utilization/busy, like other OSes? Makes it hard to interpret.

Command Line: netstat

- netstat(1): various network statistics. -i for interface stats:

```
$ netstat -iI en0 1
            input          (en0)           output
   packets  errs      bytes    packets  errs      bytes colls
       237     0     296232        167     0      18555     0
        26     0      19374         16     0       4617     0
         5     0        661          1     0       2020     0
      1601     0    2231882        535     0      50072     0
      3519     0    5027348       1005     0      62086     0
      1362     0    1923223        627     0      39699     0
      1338     0    1866404        296     0      17166     0
       878     0    1203230        182     0      14803     0
         8     0       1302         11     0       2900     0
[...]
```

- No percent utilization, but can figure it out: throughput / known max

Command Line: tcpdump

- tcpdump(1): sniff and examine network packets:

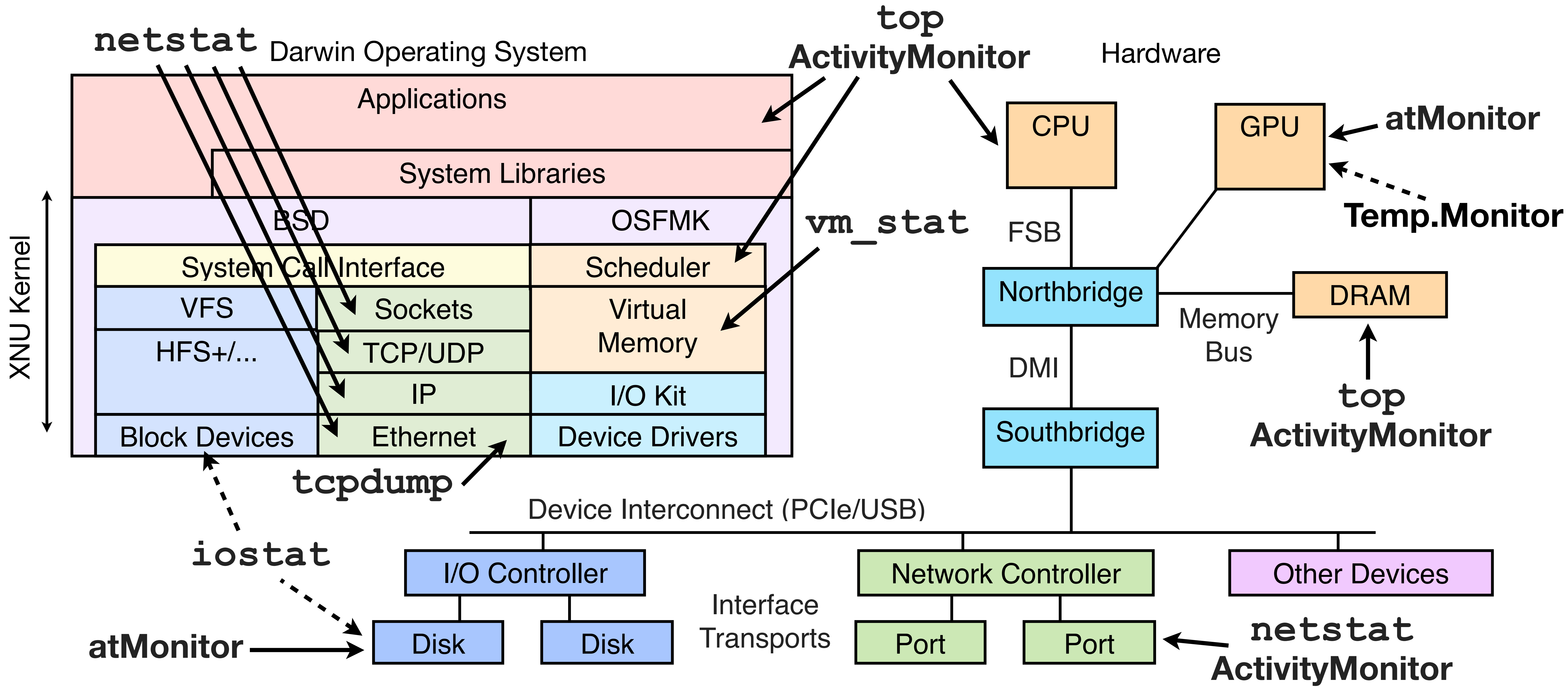
```
$ tcpdump -n
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on en0, link-type EN10MB (Ethernet), capture size 65535 bytes
18:00:55.228744 IP 10.0.1.92.53 > 10.0.1.148.49228: 26359 1/0/0 A 69.192.253.15 (81)
18:00:55.311056 ARP, Reply 10.0.1.162 is-at 2c:54:2d:a4:25:4c, length 28
18:00:55.342793 IP 74.125.28.189.443 > 10.0.1.148.62998: Flags [P.], seq
3544891232:3544891287, ack 3832081572, win 661, options [nop,nop,TS val 2936982235 ecr
2331923799], length 55
18:00:55.342933 IP 10.0.1.148.62998 > 74.125.28.189.443: Flags [.], ack 55, win 8188,
options [nop,nop,TS val 2331932237 ecr 2936982235], length 0
18:00:56.477029 IP 10.0.1.148.50359 > 67.195.141.201.443: Flags [P.], seq
696365506:696365533, ack 1903095540, win 16384, length 27
18:00:56.477158 IP 10.0.1.148.50359 > 67.195.141.201.443: Flags [F.], seq 27, ack 1, win
16384, length 0
[...]
```

- Also dump to a file and examine later. Does incur overhead.

Observability So Far...

- We can see all the things!
- Not really...

Observability So Far...



DTrace

- Programmable, real-time, dynamic and static tracing
- Write your own one-liners and scripts, or use other people's; including those in /usr/bin
- There is a great book about it...

DTrace: Scripts

- Over 40 DTrace scripts are shipped with OS X (which I mostly wrote originally). Listing them:

```
$ man -k dtrace
bitesize.d(1m)           - analyse disk I/O size by process. Uses DTrace
cpuwalk.d(1m)            - Measure which CPUs a process runs on. Uses DTrace
creatbyproc.d(1m)        - snoop creat()s by process name. Uses DTrace
dappprof(1m)             - profile user and lib function usage. Uses DTrace
dapptrace(1m)            - trace user and library function usage. Uses DTrace
diskhits(1m)             - disk access by file offset. Uses DTrace
dispqlen.d(1m)           - dispatcher queue length by CPU. Uses DTrace
dtrace(1)                - generic front-end to the DTrace facility
dtruss(1m)               - process syscall details. Uses DTrace
errinfo(1m)              - print errno for syscall fails. Uses DTrace
execsnoop(1m)            - snoop new process execution. Uses DTrace
[...]
```

DTrace: iosnoop

- iosnoop(1m): trace block device I/O

```
$ iosnoop
  UID   PID D     BLOCK   SIZE          COMM PATHNAME
  503   176 R 148471184   8192 SystemUIServer ??/vm/swapfile10
  503   176 R 835310312   4096 SystemUIServer ??/vm/swapfile4
  503 92489 W 746204600  61440 Google Chrome  ??/Chrome/.com.google.Chrome.hw1Inp
  503 92489 W 746204720  23472 Google Chrome  ??/Default/.com.google.Chrome.76k4tG
    0    19 W 425711304   4096       syslogd ??/DiagnosticMessages/2014.02.14.asl
    0    19 W 57246896    4096       syslogd ??/DiagnosticMessages/StoreData
    0    19 W 425710304   4096       syslogd ??/DiagnosticMessages/2014.02.14.asl
  503 52617 W 214894232   4096       firefox ??/iw4rbel9.default/_CACHE_CLEAN_
    0    19 W 57246896    4096       syslogd ??/DiagnosticMessages/StoreData
    0    19 W 425710304   4096       syslogd ??/DiagnosticMessages/2014.02.14.asl
[...]
```

- Identify processes and files causing disk I/O

DTrace: hfsslower.d

- hfsslower.d: trace HFS calls slower than a threshold. Eg, 10 ms:

```
$ ~/dtbook_scripts/Chap5/hfsslower.d 10
TIME                     PROCESS          D   KB     ms  FILE
2014 Feb 14 17:35:59 Terminal         R  5751     16  data.data
2014 Feb 14 17:35:59 Terminal         R  6166     17  data.data
2014 Feb 14 17:35:59 Terminal         W 11921     15  data.data
[...]
```

- Traces *all* application I/O to the file system, not just disk I/O
- Script is on http://www.dtracebook.com

DTrace: execsnoop

- execsnoop(1m): trace process execution

```
$ execsnoop -v
STRTIME                  UID    PID   PPID ARGS
2014 Feb 14 19:40:55     503  94835    551 man
2014 Feb 14 19:40:55     503  94835    551 man
2014 Feb 14 19:40:55     503  94841  94837 groff
2014 Feb 14 19:40:55     503  94839  94837 tbl
2014 Feb 14 19:40:55     503  94840  94838 cat
2014 Feb 14 19:40:56     503  94845  94841 grotty
2014 Feb 14 19:40:56     503  94844  94841 troff
2014 Feb 14 19:40:56     503  94843  94842 less
2014 Feb 14 19:40:58     503  94846  92489 Google Chrome He
2014 Feb 14 19:41:03     503  94847  92489 Google Chrome He
[...]
```

- Shows what programs are launched

DTrace: dtruss

- dtruss(1m): trace system calls, from one or many processes

```
$  dtruss -en bash
    PID/THRD    ELAPSD SYSCALL(args)             = return
  475/0x1199:    87917 read(0x0, "a\0", 0x1)             = 1 0
  475/0x1199:       12 write_nocancel(0x2, "a\0", 0x1)           = 1 0
  475/0x1199:        3 sigprocmask(0x1, 0x0, 0x7FFF55F898E0)            = 0x0 0
  475/0x1199:        2 sigaltstack(0x0, 0x7FFF55F898D0, 0x0)            = 0 0
  475/0x1199:    48163 read(0x0, "t\0", 0x1)             = 1 0
  475/0x1199:       10 write_nocancel(0x2, "t\0", 0x1)           = 1 0
  475/0x1199:        3 sigprocmask(0x1, 0x0, 0x7FFF55F898E0)            = 0x0 0
  475/0x1199:        2 sigaltstack(0x0, 0x7FFF55F898D0, 0x0)            = 0 0
  475/0x1199:       12 write_nocancel(0x2, "m\0", 0x1)           = 1 0
  475/0x1199:        2 sigprocmask(0x1, 0x0, 0x7FFF55F898E0)            = 0x0 0
[...]
```

- dtruss is a script - edit it to add/modify it as desired

DTrace: sotop

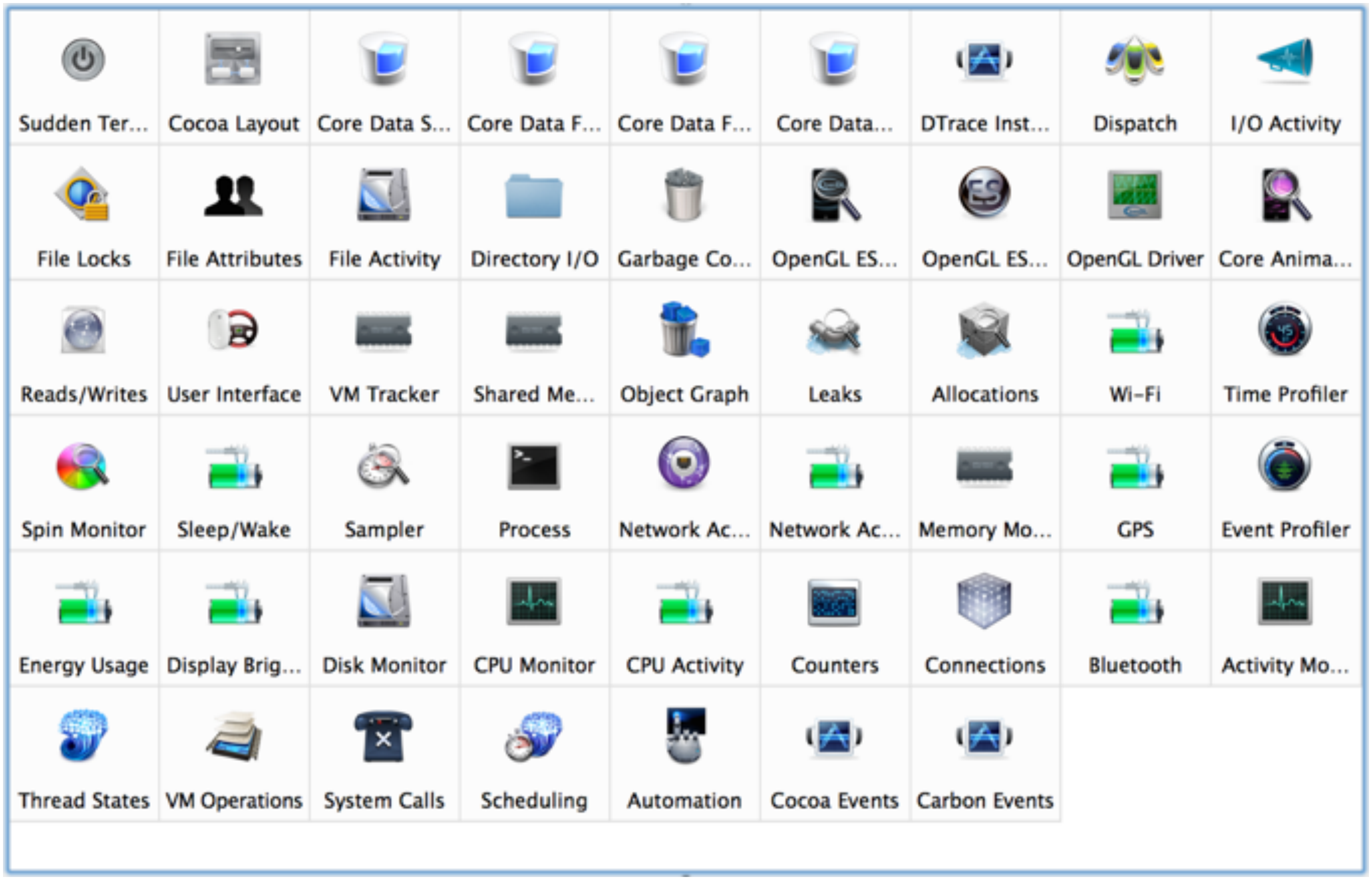
- sotop: summarize socket I/O by-process, top-style:

```
$ sotop
  PROCESS          PID         READS   WRITES    READ_KB   WRITE_KB       CPU
  kernel_task      0               0        0          0          0       475
  firefox          52617         205       14         84         22       118
  Terminal         165             0        0          0          0        35
  WindowServer     89              0        0          0          0        34
  SIDPLAY          51232           0        0          0          0        31
  Google Chrome H  92513           6       12          0          1        14
  Google Chrome H  94477           2        1          0          0        13
  clear            94909           0        0          0          0        13
  Google Chrome    92489          16        5          0          0        12
  sh               94909           0        0          0          0        12
[...]
```

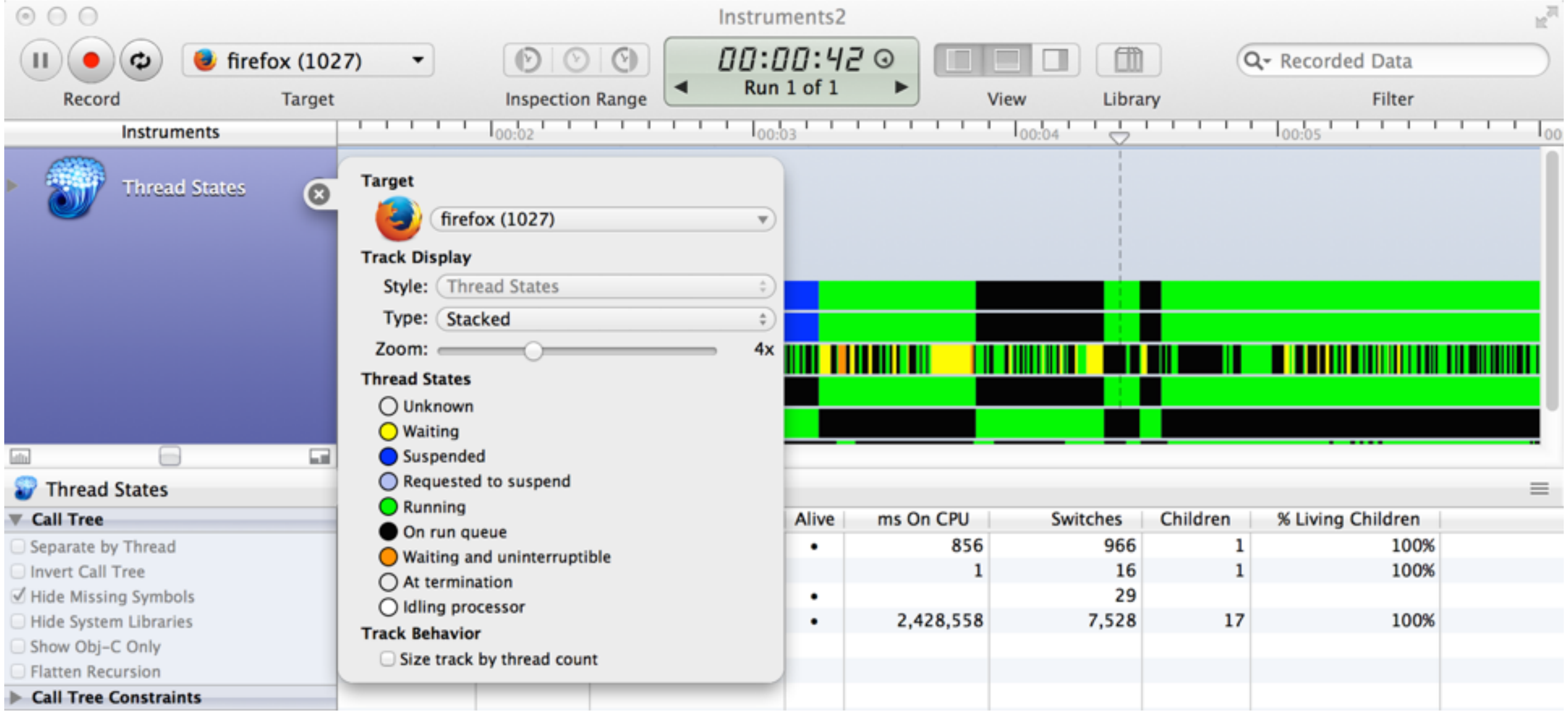
- Also from the DTrace book.

Instruments

- Advanced analysis GUI
- Includes many "Instruments", which profile applications in different ways:
- Data sources include DTrace, CPU counters

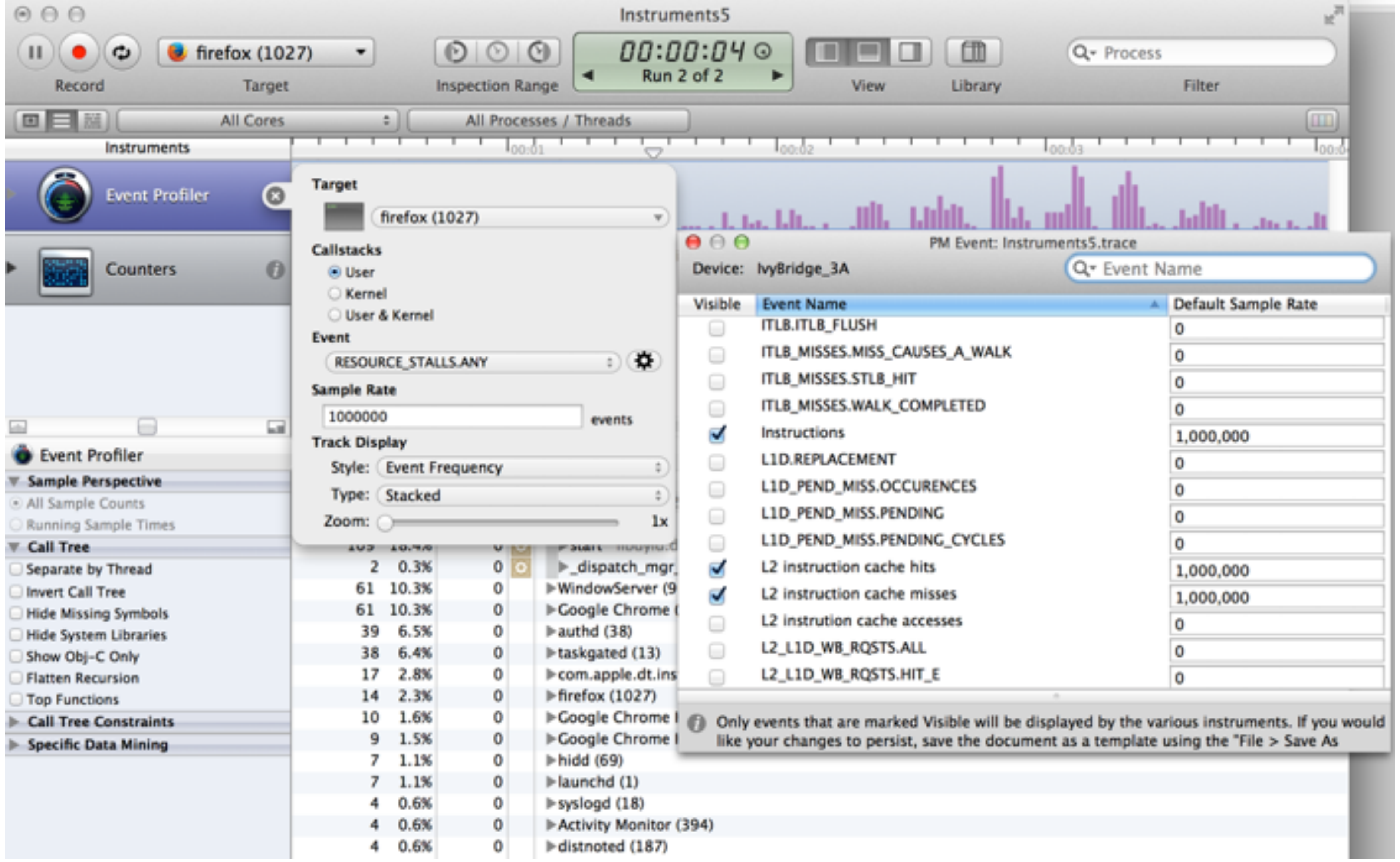


Instruments
Thread States



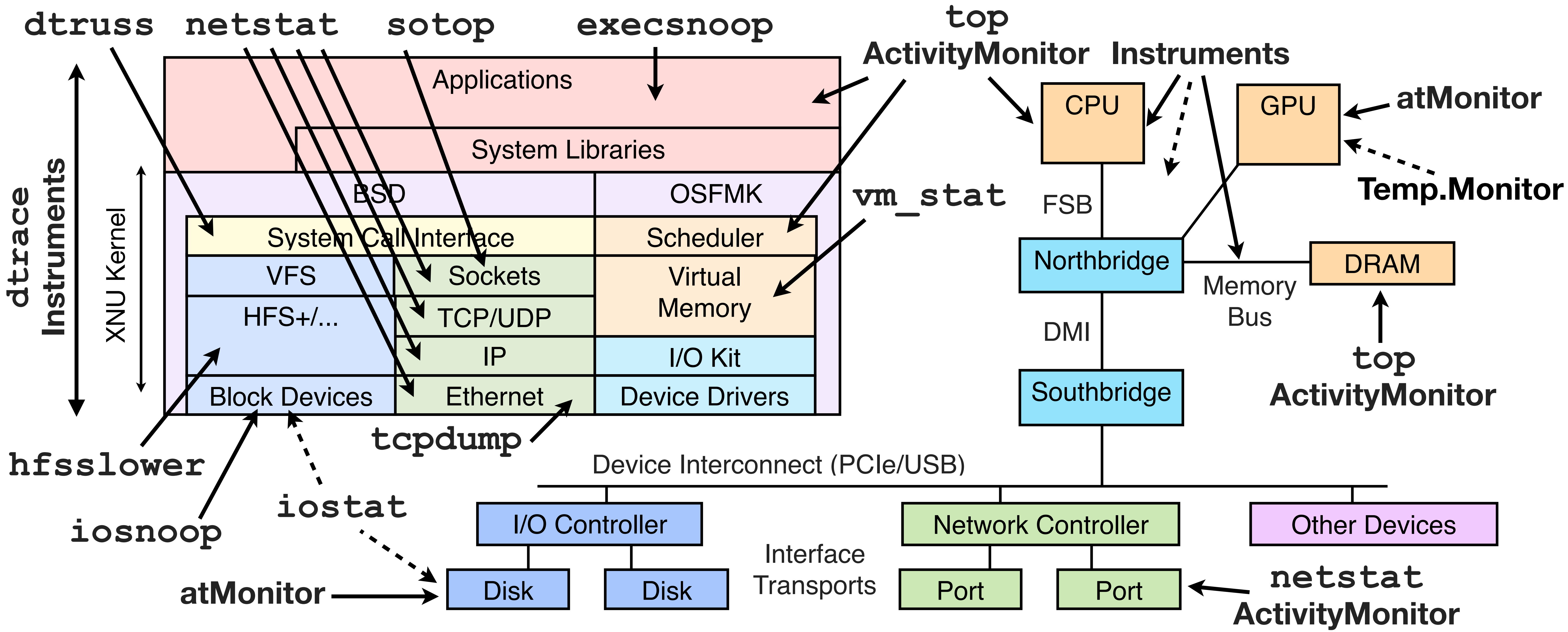
Instruments
Low Level CPU Counters

- Performance monitor counter (PMC) and performance monitor interrupts can be instrumented
- Hard work, but can be used to understand bus and interconnect activity



#macIT2014

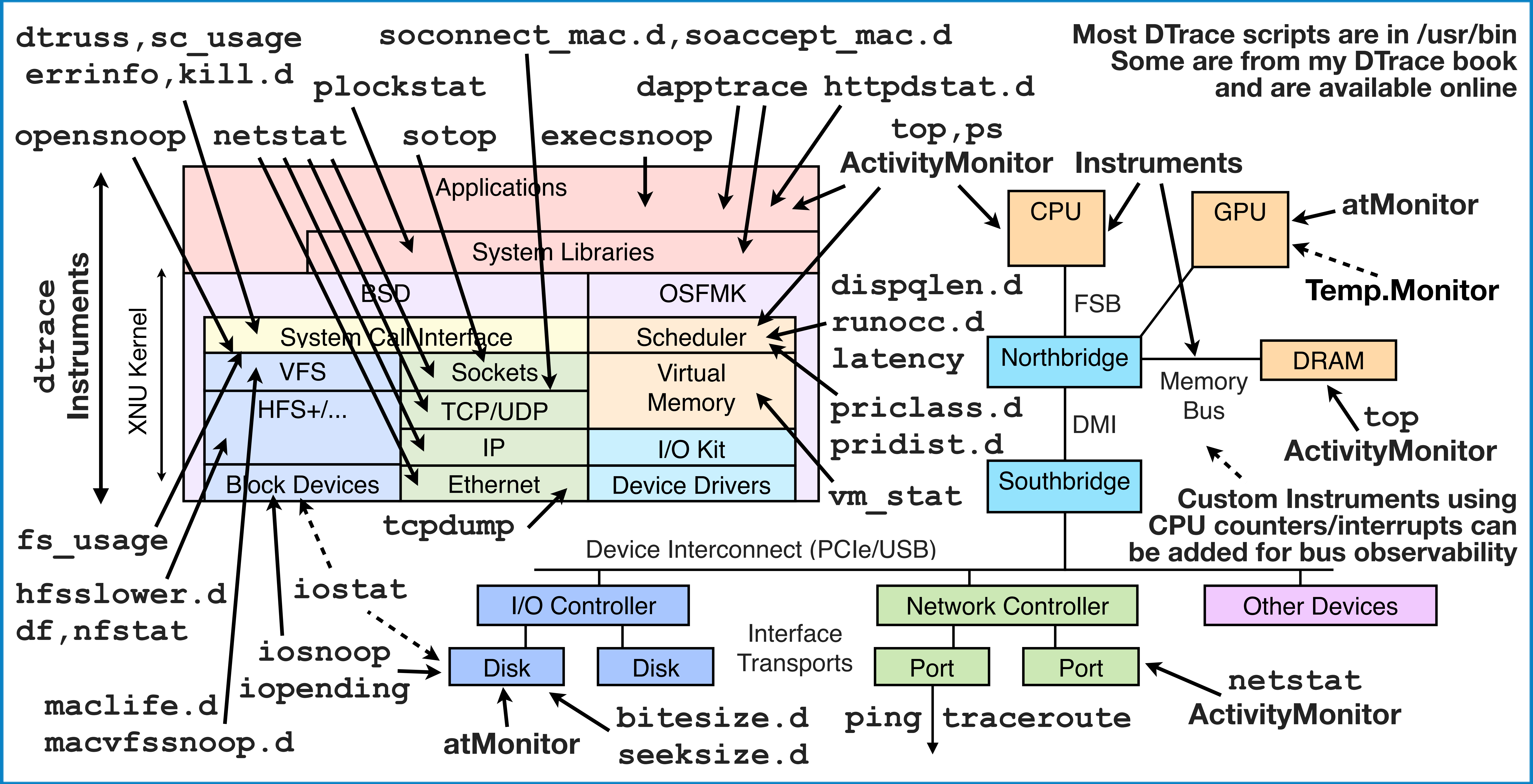
Observability So Far...



#macIT2014

Tools Method in Practice

- Tools Method provides *reasonable* coverage
  - Some observability gaps, some uneven coverage
  - Can improve coverage by adding more tools: ps, ping, traceroute, latency, df, sysctl, plockstat, opensnoop, dispqlen.d, runocc.d, nfsstat, iopending, soconnect_mac.d, httpdstat.d, sc_usage, fs_usage, ...
- I could keep covering tools for the rest of this talk...


dtruss,sc_usage
errinfo,kill.d
soconnect_mac.d,soaccept_mac.d
plockstat
dapptrace httpdstat.d
Most DTrace scripts are in /usr/bin
Some are from my DTrace book
and are available online
opensnoop
netstat
sotop
execsnoop
top,ps
ActivityMonitor
Instruments
Applications
System Libraries
BSD
OSFMK
System Call Interface
Scheduler
VFS
Sockets
HFS+/...
TCP/UDP
IP
Virtual
Memory
I/O Kit
Block Devices
Ethernet
Device Drivers
dtrace
Instruments
XNU Kernel
CPU
GPU
atMonitor
Temp.Monitor
FSB
dispqlen.d
runocc.d
latency
Northbridge
DRAM
Memory
Bus
DMI
priclass.d
pridist.d
top
ActivityMonitor
vm_stat
Southbridge
Custom Instruments using
CPU counters/interrupts can
be added for bus observability
fs_usage
tcpdump
Device Interconnect (PCIe/USB)
hfsslower.d
df,nfstat
iostat
I/O Controller
Network Controller
Other Devices
Interface
Transports
iosnoop
iopending
Disk
Disk
Port
Port
netstat
ActivityMonitor
maclife.d
macvfssnoop.d
atMonitor
bitesize.d
seeksize.d
ping
traceroute
#macIT2014

The Focus on Tools

- Useful, however, learning tools & metrics becomes laborious.
- *Still* limited by what the tools provide, or provide easily.
- You can try to approach this in a different way...

Instead of starting with the tools, start with the *questions*

The USE Method

The USE Method

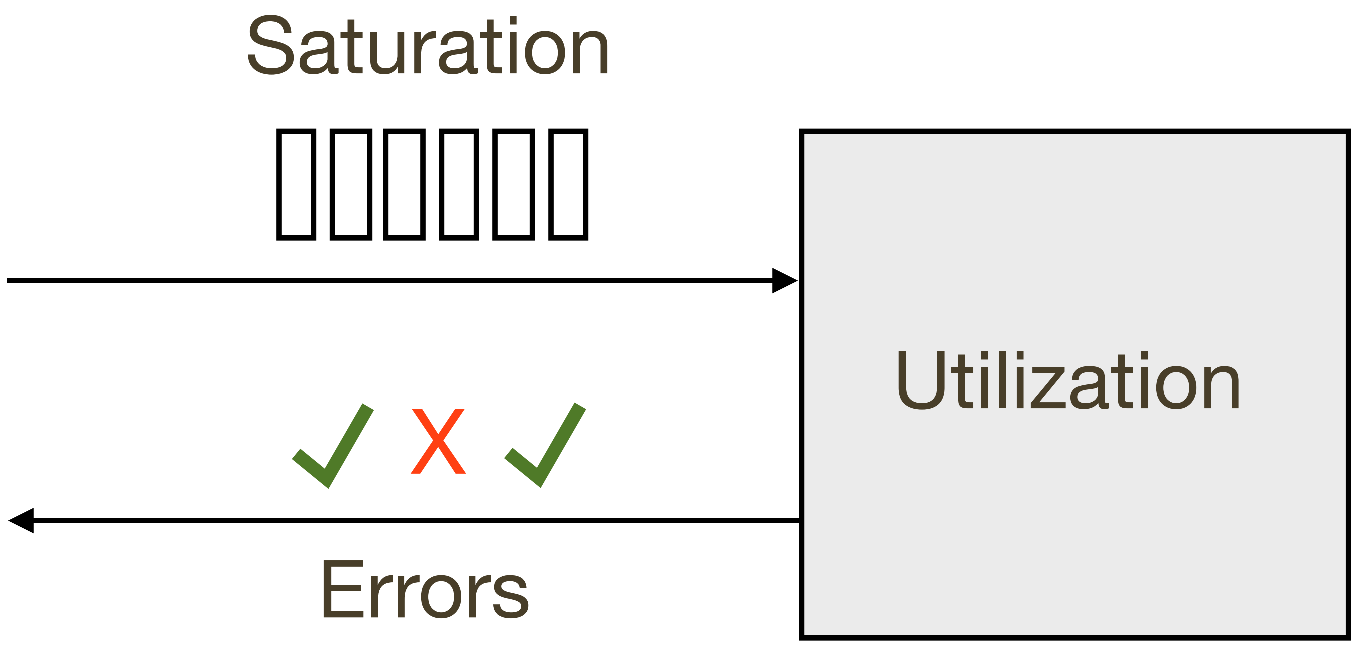
- For every resource, check:
- 1. Utilization
- 2. Saturation
- 3. Errors

The 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

Queueing System

- If it helps, consider all resources as a queueing system:
- Also check errors

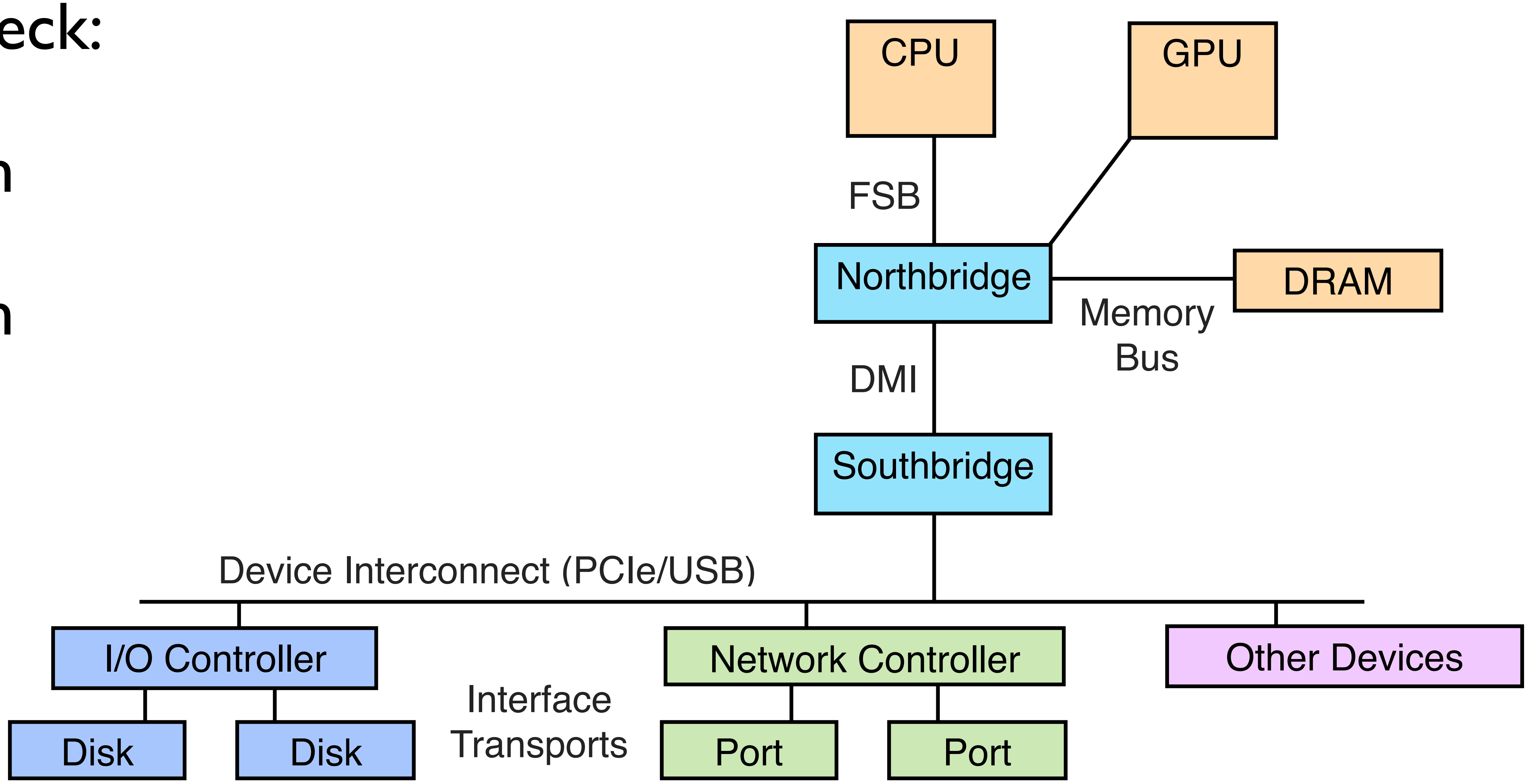


Hardware Resources

- CPUs
- Main Memory
- Network Interfaces
- Storage Devices
- Controllers, Interconnects
- Find the *functional diagram* and examine every item in the data path...

Hardware Functional Diagram

- For each check:
- 1. Utilization
- 2. Saturation
- 3. Errors



USE Method Checklists

- Build a checklist for all combinations, identifying tools/metrics to use

OS X Checklist

Resource	Type	Metric
CPU	Utilization	
CPU	Saturation	
CPU	Errors	

OS X Checklist

Resource	Type	Metric
CPU	Utilization	system-wide: iostat 1, "us" + "sy"; per-cpu: DTrace [1]; Activity Monitor → CPU Usage or Floating CPU Window; per-process: top -o cpu, "%CPU"; Activity Monitor → Activity Monitor, "%CPU"; ...
CPU	Saturation	system-wide: uptime, "load averages" > CPU count; latency, "SCHEDULER" and "INTERRUPTS"; per-cpu: dispqlen.d (DTT), non-zero "value"; runocc.d (DTT), non-zero "%runocc"; per-process: Instruments → Thread States, "On run queue"; DTrace [2]
CPU	Errors	dmesg; /var/log/system.log; Instruments → Counters, for PMC and whatever error counters are supported (eg, thermal throttling)

OS X Checklist

Resource	Type	Metric
CPU	Utilization	system-wide: iostat 1, "us" + "sy"; per-cpu: DTrace [1]; Activity Monitor → CPU Usage or Floating CPU Window; per-process: top -o cpu, "%CPU"; Activity Monitor → Activity Monitor, "%CPU"; ...
CPU	Saturation	system-wide: uptime, "load averages" > CPU count; latency, "SCHEDULER" and "INTERRUPTS"; per-cpu: dispqlen.d (DTT), non-zero "value"; runocc.d (DTT), non-zero "%runocc"; per-process: Instruments → Thread States, "On run queue"; DTrace [2]
CPU	Errors	dmesg; /var/log/system.log; Instruments → Counters, for PMC and whatever error counters are supported (eg, thermal throttling)

OS X Checklist, cont.

Resource	Type	Metric
Memory Capacity	Utilization	
Memory Capacity	Saturation	
"	Errors	

OS X Checklist, cont.

Resource	Type	Metric
Memory Capacity	Utilization	system-wide: vm_stat 1, main memory free = "free" + "inactive", in units of pages; Activity Monitor → Activity Monitor → System Memory, "Free" for main memory; per-process: top -o rsize, "RSIZE" is resident main memory size, "VSIZE" is virtual memory size; ps -alx, "RSS" is resident set size, "SZ" is virtual memory size;
Memory Capacity	Saturation	system-wide: vm_stat 1, "pageout"; per-process: anonpgpid.d (DTT), DTrace vminfo:::anonpgin [3] (frequent anonpgin == pain); Instruments → Memory Monitor, high rate of "Page Ins" and "Page Outs"; sysctl vm.memory_pressure [4]
"	Errors	System Information → Hardware → Memory, "Status" for physical failures; DTrace failed malloc()s

OS X Checklist, cont.

Resource	Type	Metric
Memory Capacity	Utilization	system-wide: vm_stat 1, main memory free = "free" + "inactive", in units of pages; Activity Monitor → Activity Monitor → System Memory, "Free" for main memory; per-process: top -o rsize, "RSIZE" is resident main memory size, "VSIZE" is virtual memory size; ps -alx, "RSS" is resident set size, "SZ" is virtual memory size;
Memory Capacity	Saturation	system-wide: vm_stat 1, "pageout"; per-process: anonpgpid.d (DTT), DTrace vminfo:::anonpgin [3] (frequent anonpgin == pain); Instruments → Memory Monitor, high rate of "Page Ins" and "Page Outs"; sysctl vm.memory_pressure [4]
"	Errors	System Information → Hardware → Memory, "Status" for physical failures; DTrace failed malloc()s

OS X Checklist, cont.

- Full list: http://www.brendangregg.com/USEmethod/use-macosx.html
- Includes references from earlier tables

USE Method: Mac OS X Performance Checklist

This is my example USE Method-based performance checklist for the Apple Mac OS X operating system, for identifying common bottlenecks and errors. This draws upon both command line and graphical tools for coverage, focusing where possible on those that are provided with the OS by default, or by Apple (eg, Instruments). Further notes about tools are provided after this table.

Some of the metrics are easy to find in various GUIs or from the command line (eg, using Terminal; if you've never used Terminal before, follow my instructions at the top of this post). Many metrics require some math, inference, or quite a bit of digging. This will hopefully get easier in the future, as tools include a USE method wizard or the metrics required to follow this easily.

Physical Resources, Standard

component	type	metric
CPU	utilization	system-wide: `iostat 1`, "us" + "sy"; per-cpu: DTrace [1]; Activity Monitor → CPU Usage or Floating CPU Window; per-process: `top -o cpu`, "%CPU"; Activity Monitor → Activity Monitor, "%CPU"; per-kernel-thread: DTrace profile stack()
CPU	saturation	system-wide: `uptime`, "load averages" > CPU count; `latency`, "SCHEDULER" and "INTERRUPTS"; per-cpu: `dispqlen.d` (DTT), non-zero "value"; `runocc.d` (DTT), non-zero "%runocc"; per-process: Instruments → Thread States, "On run queue"; DTrace [2]
CPU	errors	`dmesg`; /var/log/system.log; Instruments → Counters, for PMC and whatever error counters are supported (eg, thermal throttling)
Memory capacity	utilization	system-wide: `vm_stat 1`, main memory free = "free" + "inactive", in units of pages; Activity Monitor → Activity Monitor → System Memory, "Free" for main memory; per-process: `top -o rsize`, "RSIZE" is resident main memory size, "VSIZE" is virtual memory size; `ps -alx`, "RSS" is resident set size, "SZ" is virtual memory size; `ps aux` similar (legacy format)
Memory		system-wide: `vm_stat 1`, "pageout"; per-process: anonpgpid.d (DTT), DTrace vminfo:::anonpgin [3] (frequent anonpgin == pain);

#macIT2014

Software Resources

- Can be studied using USE metrics as well, if possible
- OS X Checklist includes some example software resources:
  - Processes, file descriptors, kernel mutexes, user-level mutexes

Mutex Lock

- Can you think of what these could mean for a mutex lock?:
  - Utilization
  - Saturation
  - Errors

Mutex Lock

- Can you think of what these could mean for a mutex lock?:
  - Utilization: held time per second
  - Saturation: measure of contention time or waiters
  - Errors: EDEADLK, EINVAL

Future Work

Future Work

- Tools/Metrics for USE Method
- More methodologies, and then tools

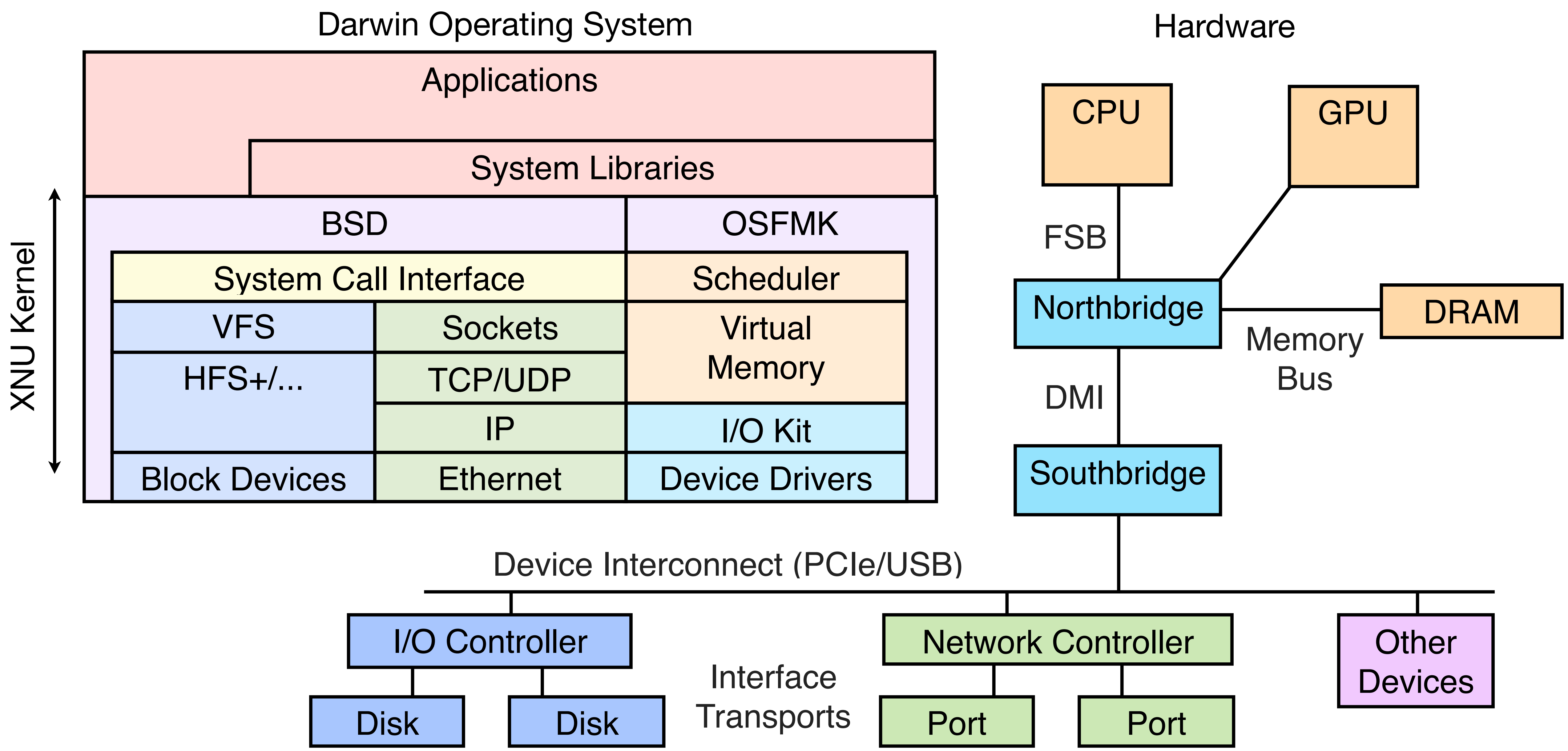
USE Method Tools

- Tools can be developed to fetch USE metrics more easily
  - Especially for busses and interconnects
- Would love to see USE metrics in Activity Monitor

USE Method New Uses

- Can be applied new areas, developing new metrics
- May not always work, but worth trying
- Find a functional diagram of your system, application, or environment, and look for U.S.E. metrics for each component

USE Metrics for all of:



Stranger Example: TCP

- "netstat -s" output has over 50 metrics for TCP
- Do you understand them all?
- Could USE metrics provide a high level summary, treating TCP as a software resource? (might be a stretch)

```
$ netstat -s
tcp:
        80444499 packets sent
                28706719 data packets (3613656050 bytes)
                76599 data packets (65712152 bytes) retransmitted
                68 resends initiated by MTU discovery
                41687640 ack-only packets (248964 delayed)
                0 URG only packets
                0 window probe packets
                9286129 window update packets
                707685 control packets
                0 data packets sent after flow control
        177149270 packets received
                16296459 acks (for 3602941580 bytes)
                556237 duplicate acks
                0 acks for unsent data
                154775303 packets (1214952475 bytes) received in-sequence
                200501 completely duplicate packets (151553377 bytes)
                1884 old duplicate packets
                79 packets with some dup. data (17270 bytes duped)
                6102493 out-of-order packets (4236017281 bytes)
                67 packets (0 bytes) of data after window
                0 window probes
                14180 window update packets
                72825 packets received after close
                85 bad resets
                0 discarded for bad checksums
                0 discarded for bad header offset fields
                0 discarded because packet too short
        378961 connection requests
        613 connection accepts
        37 bad connection attempts
        0 listen queue overflows
        332688 connections established (including accepts)
        381180 connections closed (including 13038 drops)
                14527 connections updated cached RTT on close
                14527 connections updated cached RTT variance on close
                5495 connections updated cached ssthresh on close
        1721 embryonic connections dropped
        16204052 segments updated rtt (of 8674926 attempts)
        374184 retransmit timeouts
                4465 connections dropped by rexmit timeout
                0 connections dropped after retransmitting FIN
        91 persist timeouts
                0 connections dropped by persist timeout
        12784 keepalive timeouts
                262 keepalive probes sent
                1214 connections dropped by keepalive
        1312411 correct ACK header predictions
        152849516 correct data packet header predictions
        17244 SACK recovery episodes
        21329 segment rexmits in SACK recovery episodes
        25852298 byte rexmits in SACK recovery episodes
        180630 SACK options (SACK blocks) received
        5682514 SACK options (SACK blocks) sent
        0 SACK scoreboard overflow
[...]
```

USE Method: TCP

- TCP as a software resource metrics:
  - Utilization
  - Saturation
  - Errors

USE Method: TCP

- TCP as a software resource metrics:
  - Utilization: time data was buffered per second
  - Saturation: listen queue overflows
  - Errors: bad connection attempts, bad resets, bad checksums, ...
- I think I'd classify retransmits and duplicates as errors.

Other Methodologies

- Other methodologies include:
  - Drill Down Analysis Method
  - Workload Characterization
  - Thread State Analysis (TSA) Method
- These too can pose questions that tools then answer

References

- http://www.brendangregg.com/USEmethod/use-macosx.html
- http://www.brendangregg.com/usemethod.html
- http://dtracebook.com - has DTrace book scripts online
- http://dtrace.org/blogs/brendan/2011/10/10/top-10-dtrace-scripts-for-mac-os-x/
- http://dtrace.org/blogs/brendan/2011/12/18/visualizing-device-utilization/ - utilization heat maps
- http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html - flame graphs

Thanks

- http://www.brendangregg.com
- bgregg@netflix.com
- @brendangregg