



Netflix Instance Performance Analysis Requirements

Brendan Gregg

Senior Performance Architect

Performance Engineering Team

bgregg@netflix.com [@brendangregg](https://twitter.com/brendangregg)





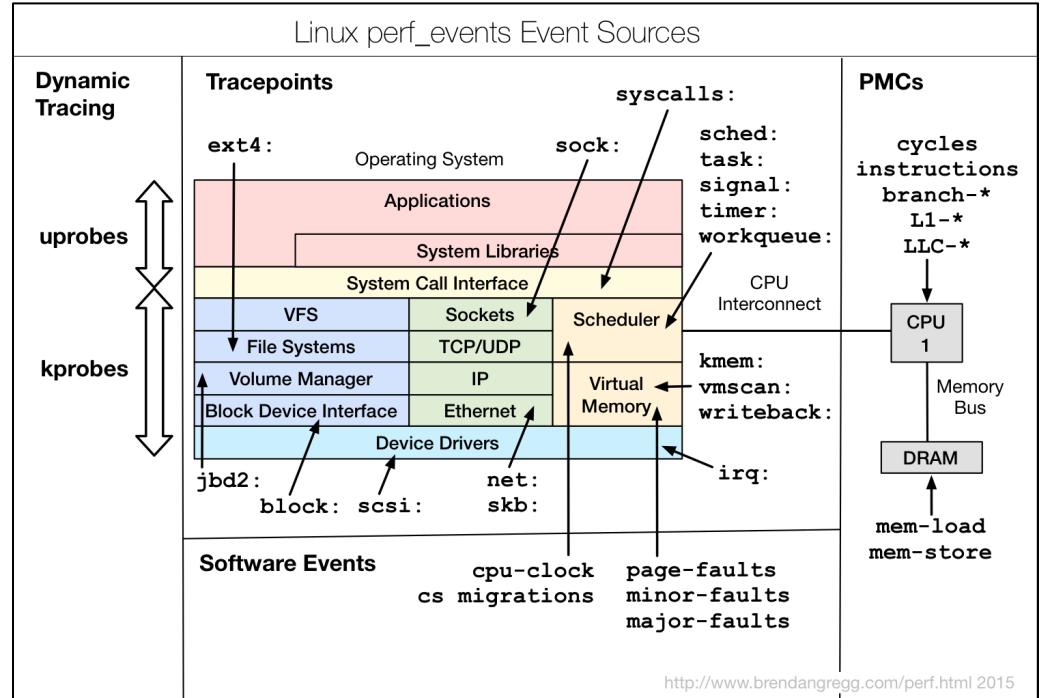
Monitoring companies are selling
faster horses

I want to buy a car

Server/Instance Analysis Potential

In the last 10 years...

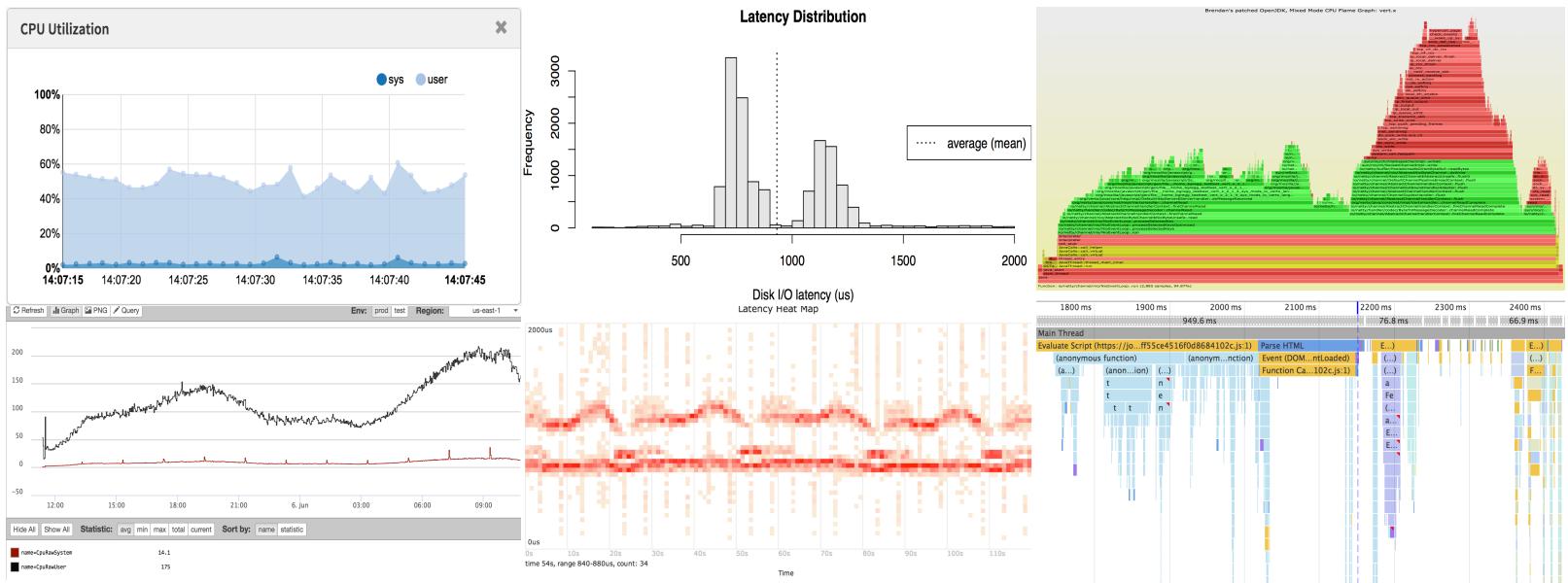
- More Linux
- More Linux metrics
- Better visualizations
- Containers



Conditions ripe for innovation: where is our Henry Ford?

This Talk

- **Instance analysis:** system resources, kernel, processes
 - For customers: what you can ask for
 - For vendors: our desirables & requirements
 - What we are building (and open sourcing) at Netflix to modernize instance performance analysis (Vector, ...)



NETFLIX

- Over 60M subscribers
- FreeBSD CDN for content delivery
- Massive AWS EC2 Linux cloud
- Many monitoring/analysis tools
- Awesome place to work

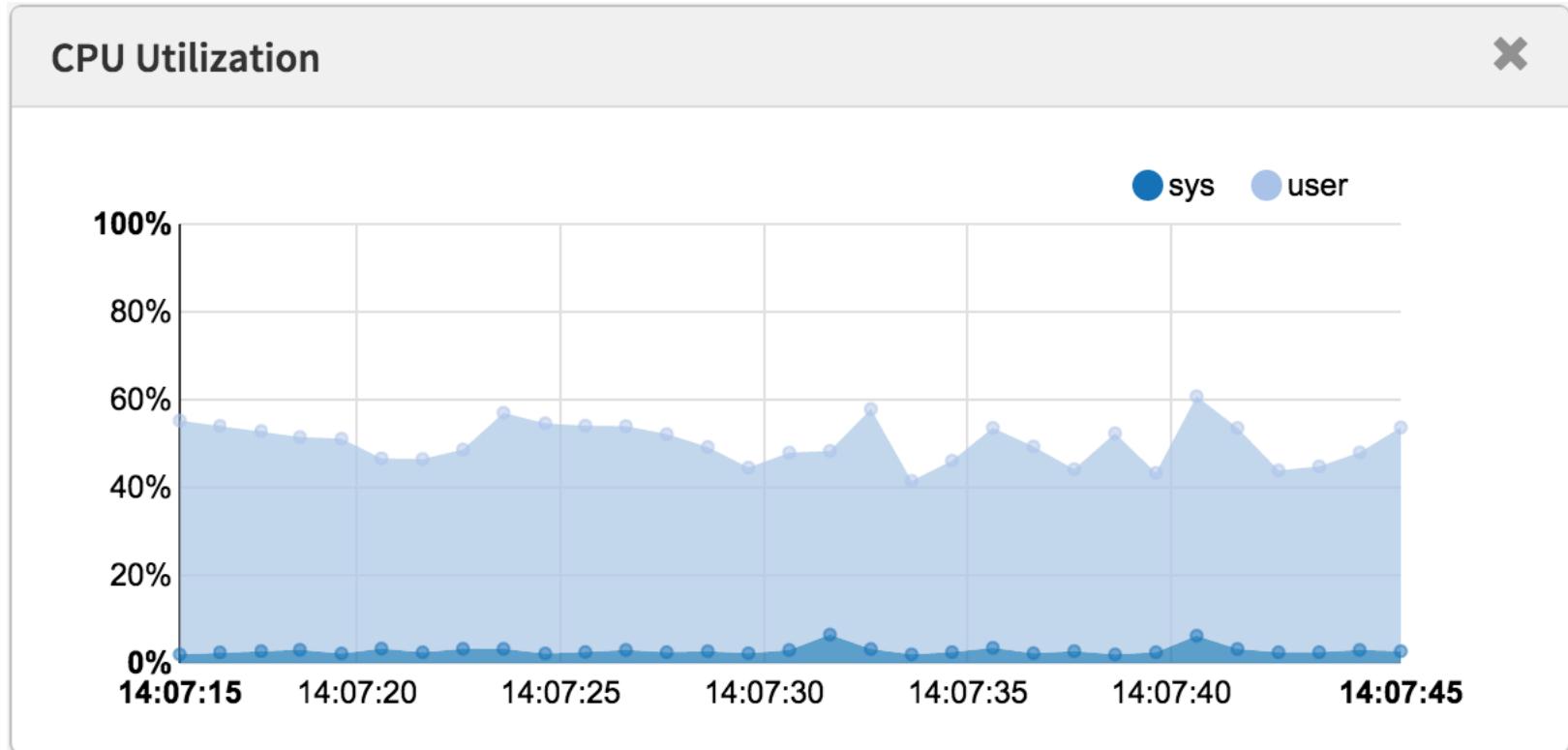


Agenda

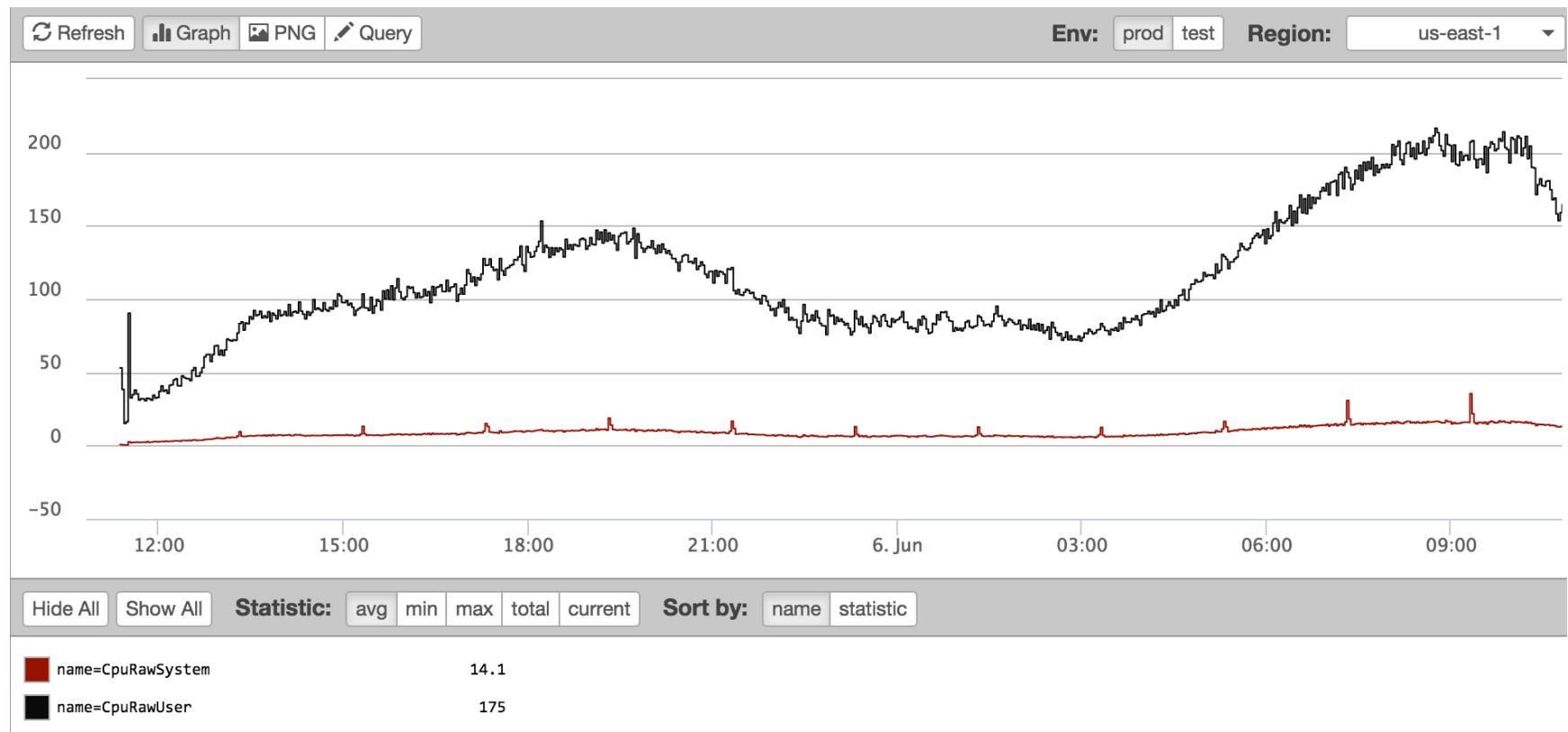
1. Desirables
2. Undesirables
3. Requirements
4. Methodologies
5. Our Tools

1. Desirables

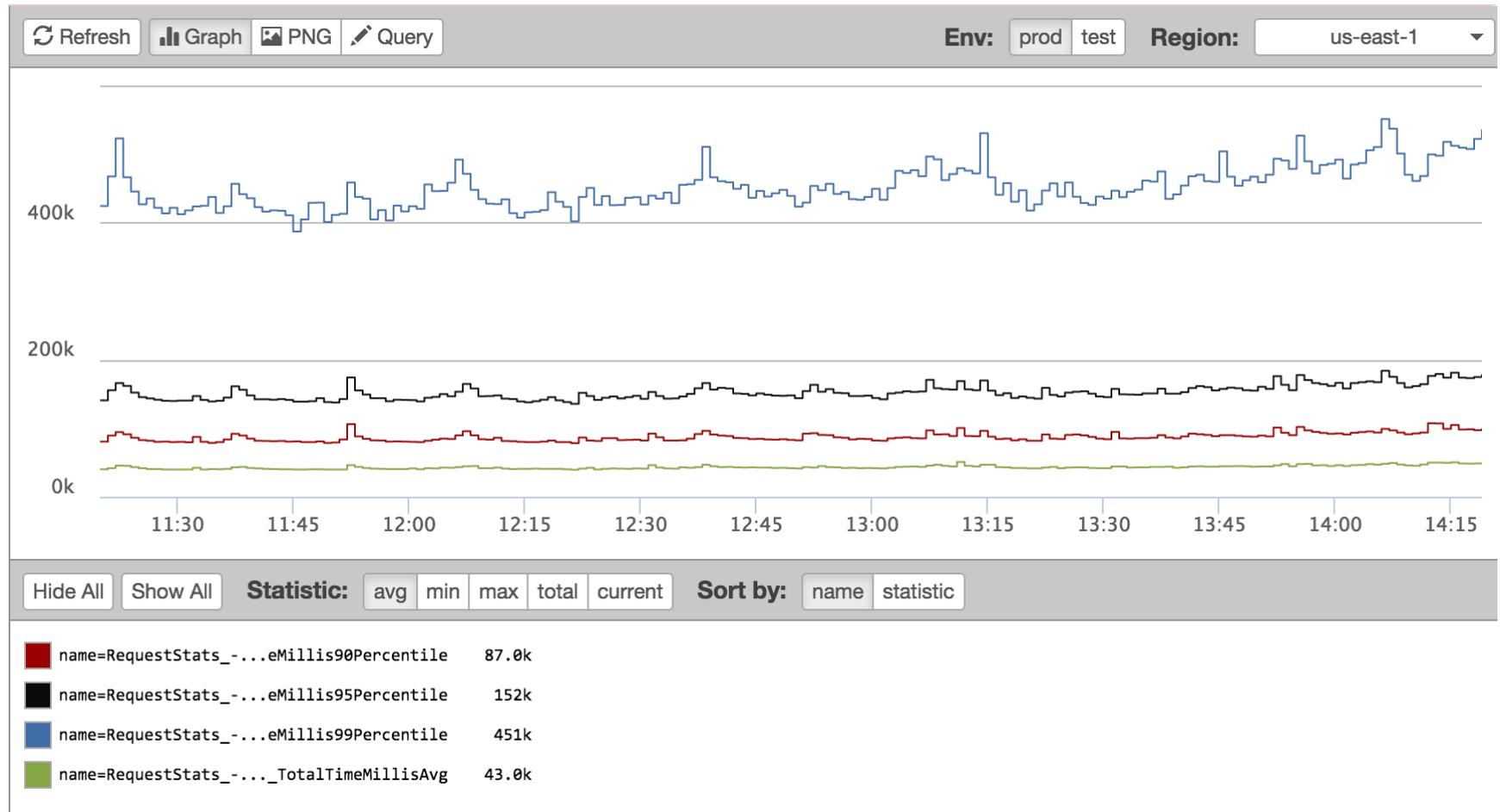
Line Graphs



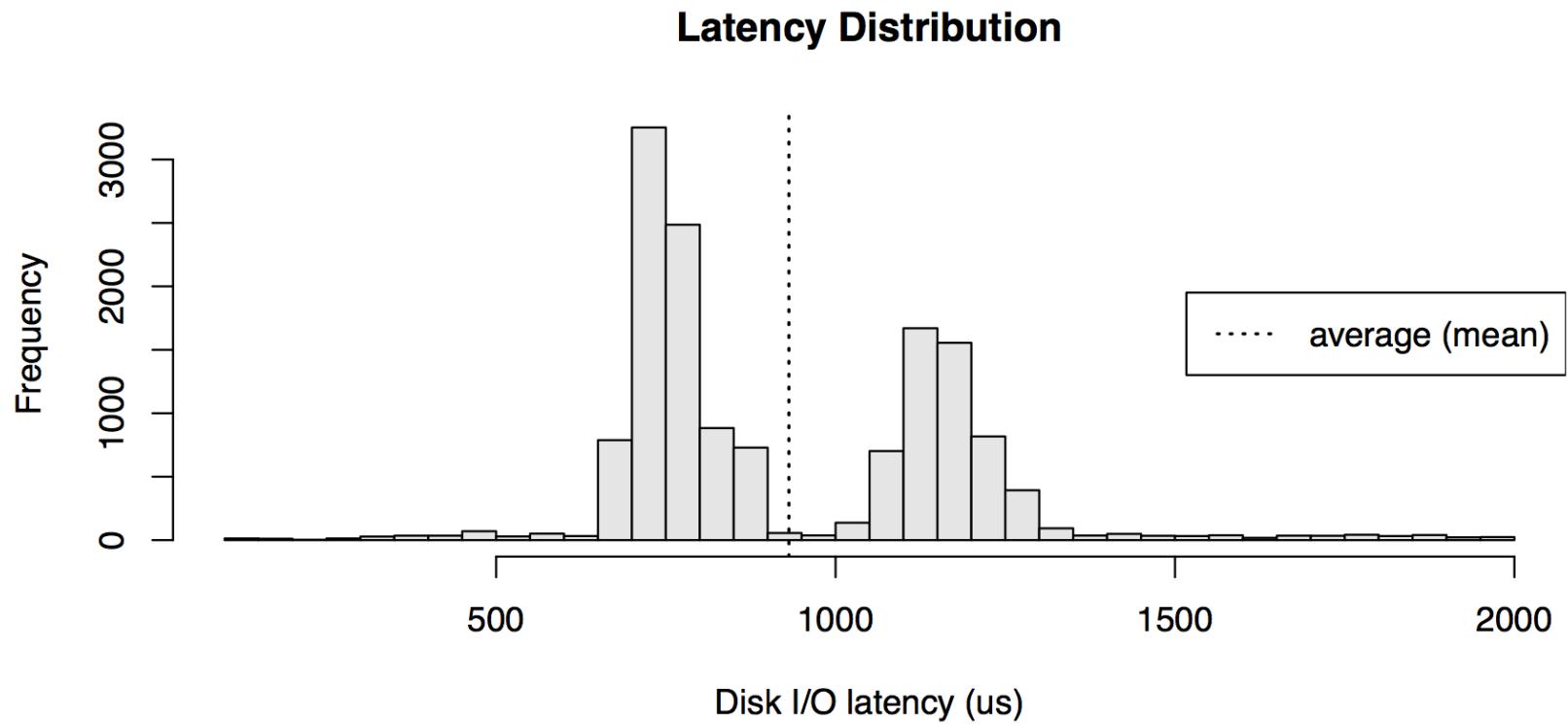
Historical Data



Summary Statistics

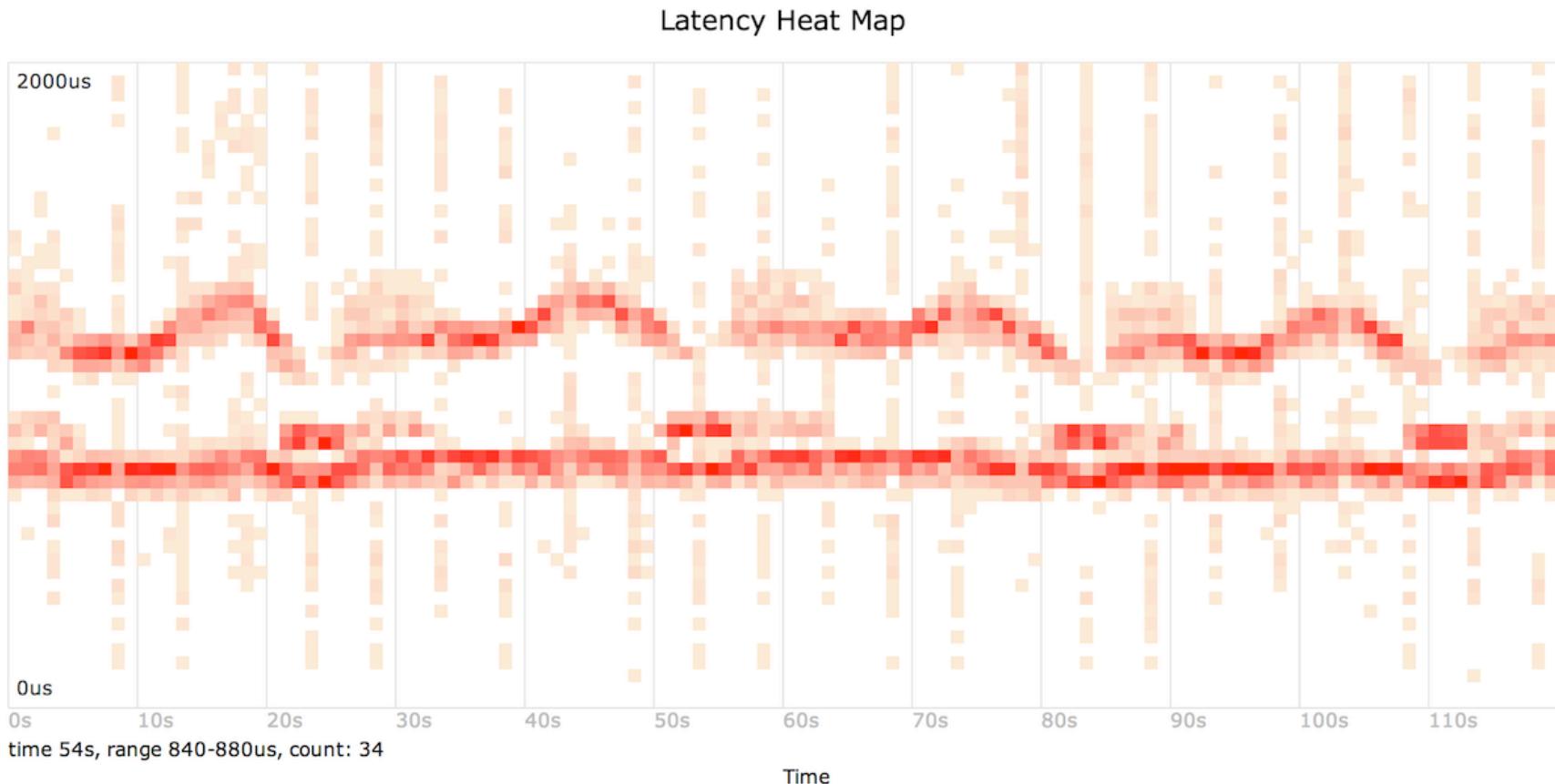


Histograms

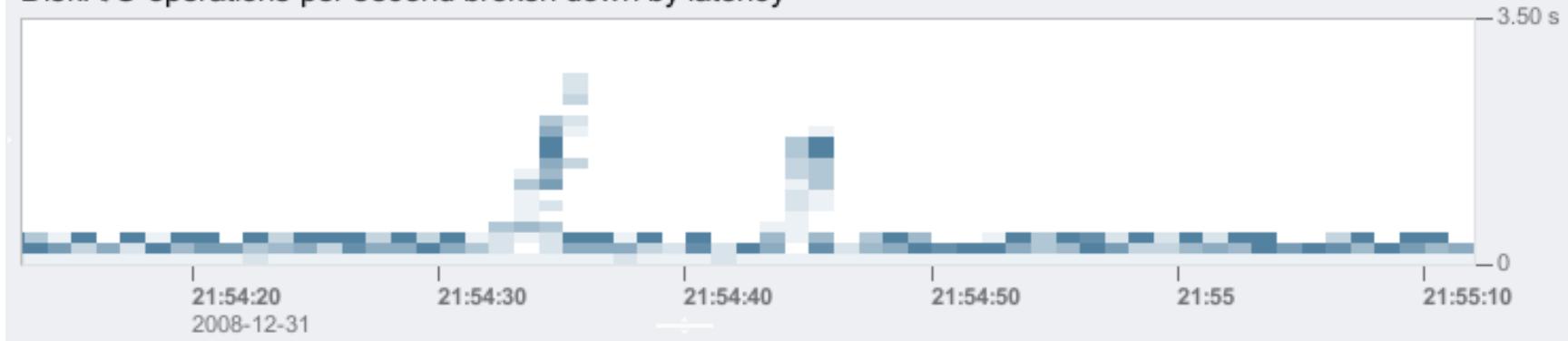


... or a density plot

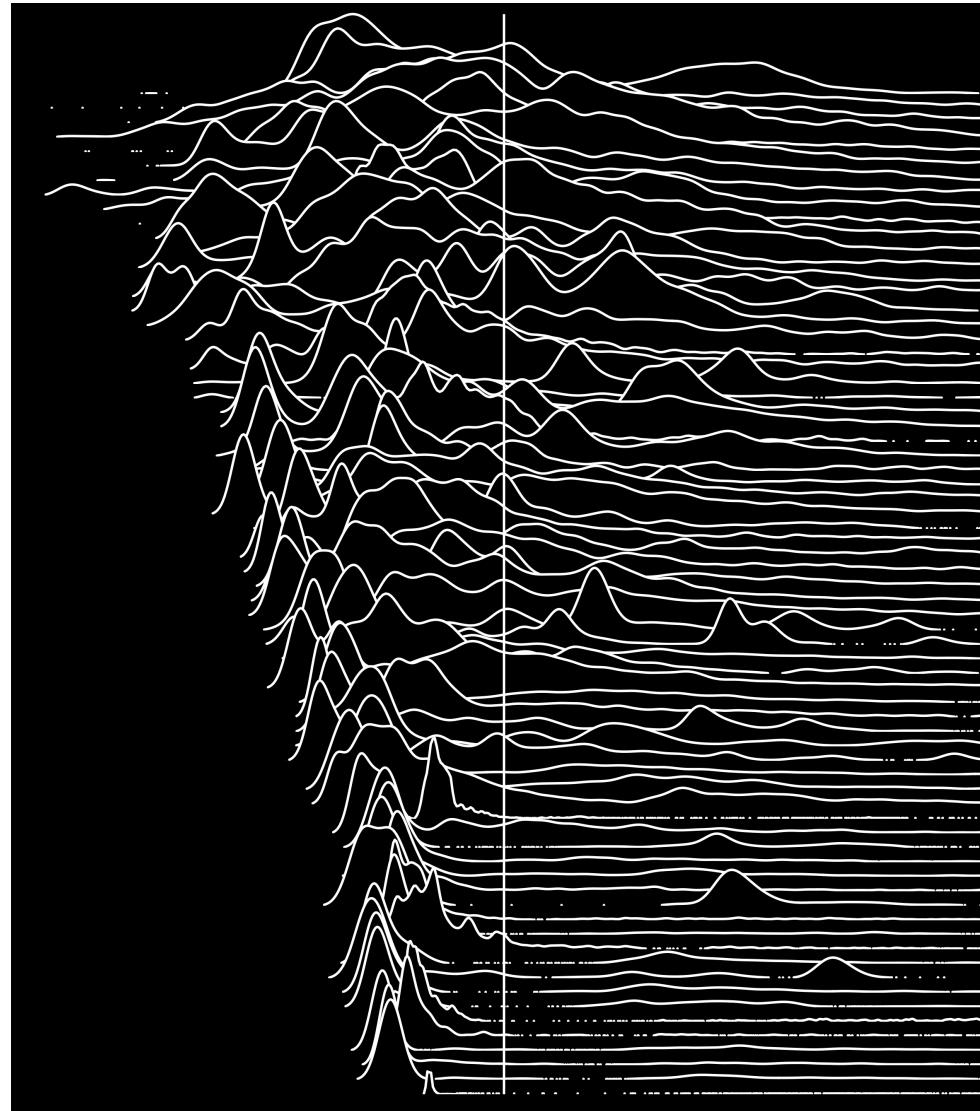
Heat Maps



Disk: I/O operations per second broken down by latency



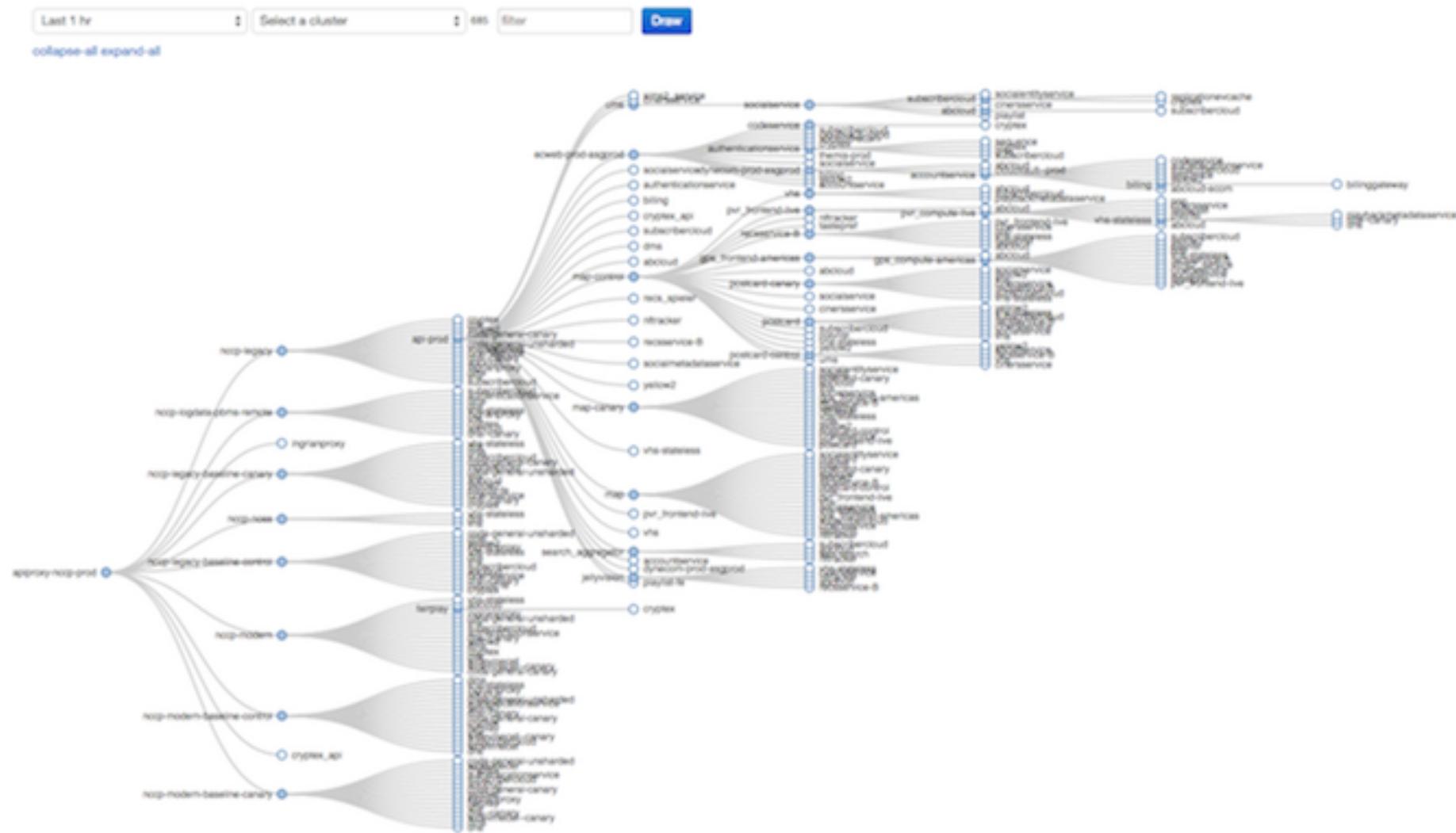
Frequency Trails



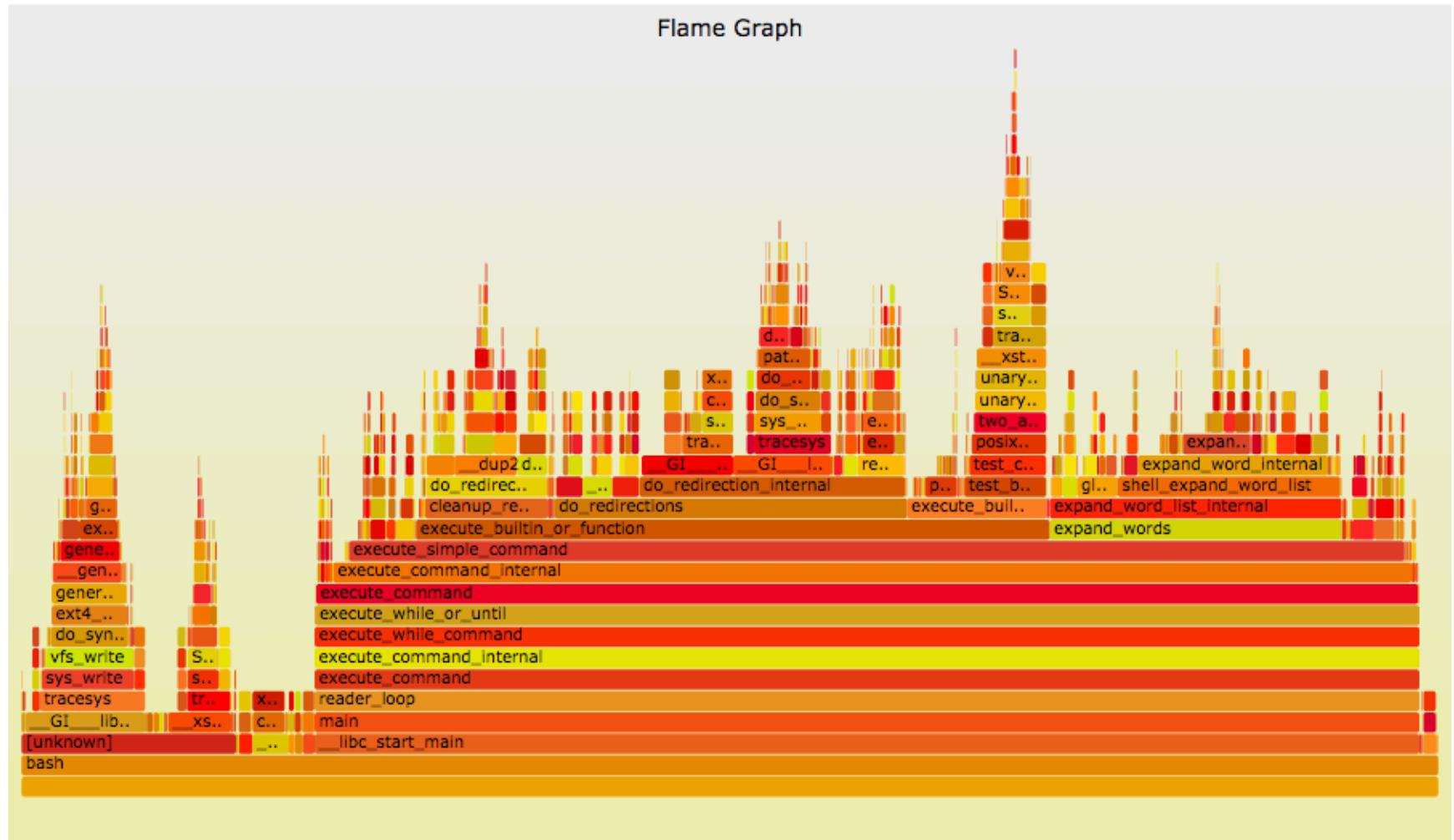
Waterfall Charts



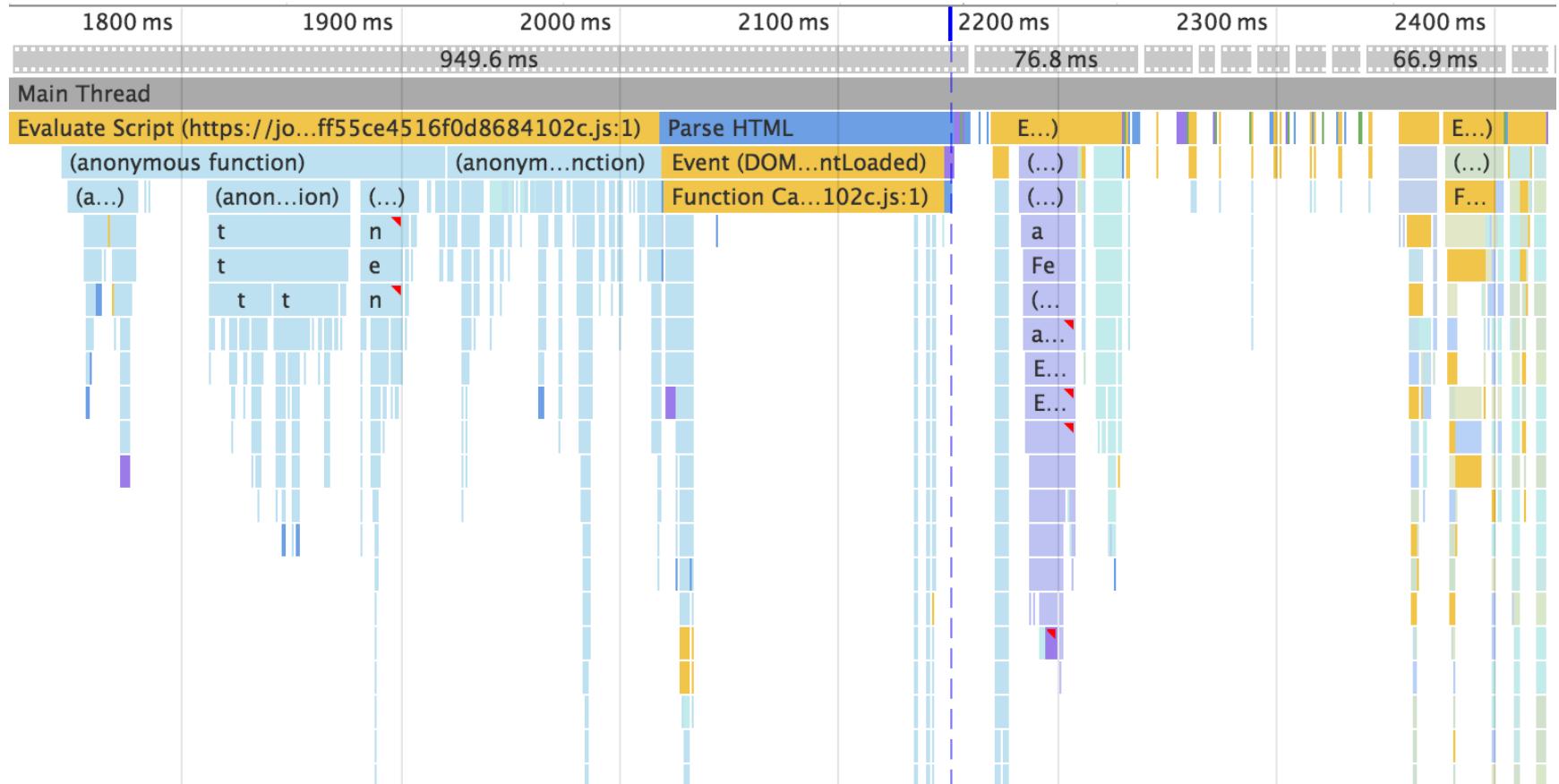
Directed Graphs



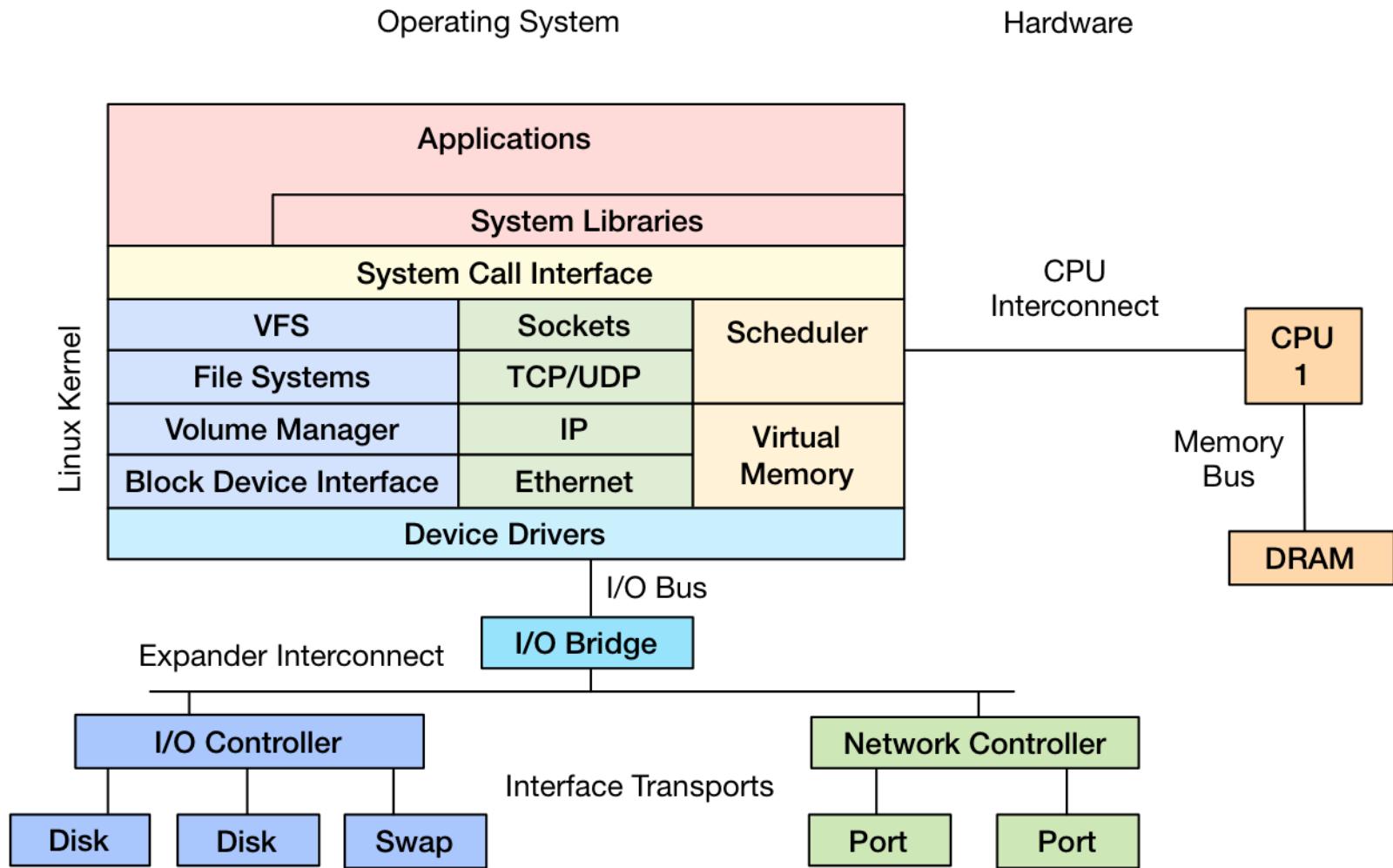
Flame Graphs



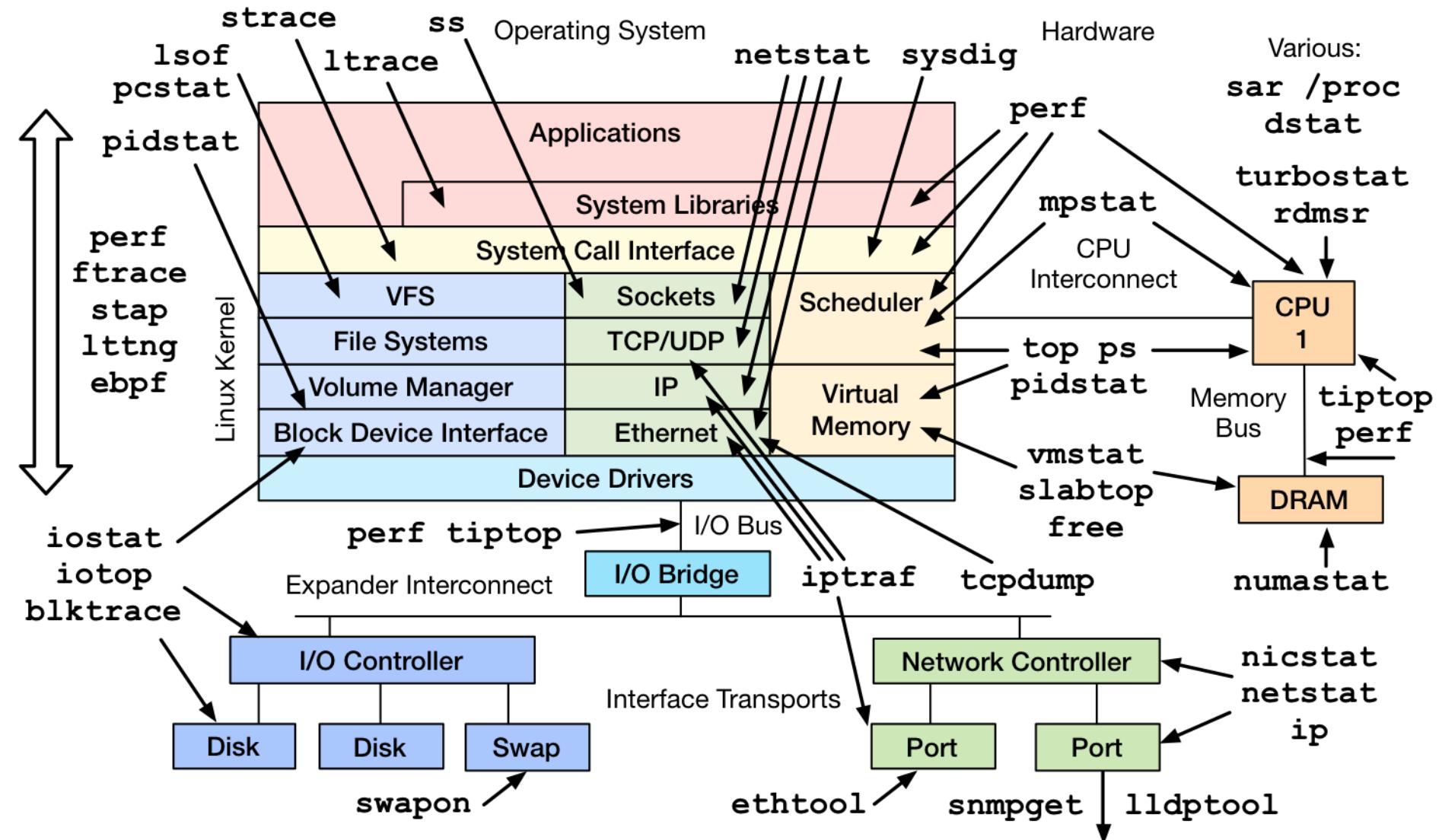
Flame Charts



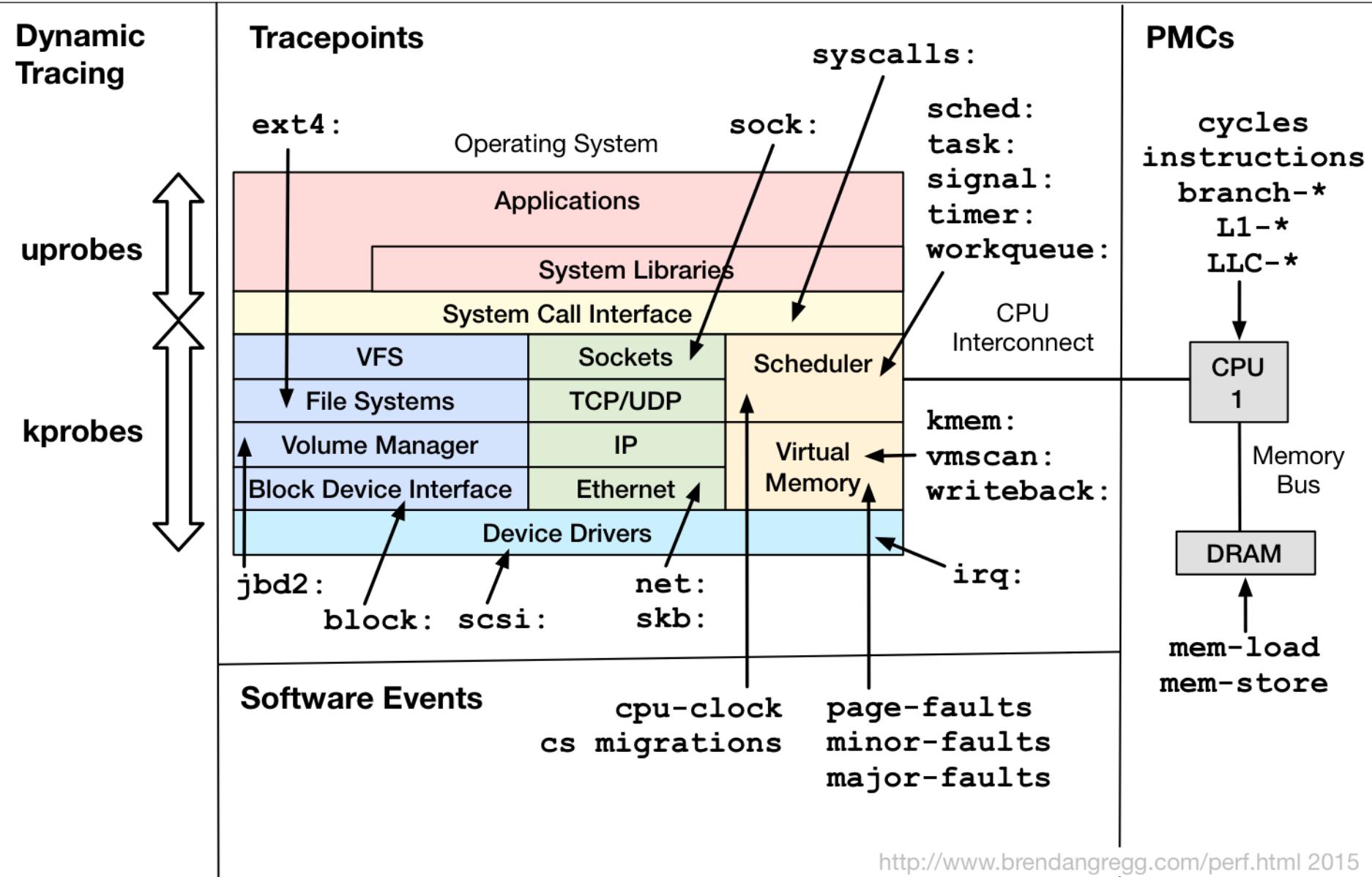
Full System Coverage



... Without Running All These



Deep System Coverage

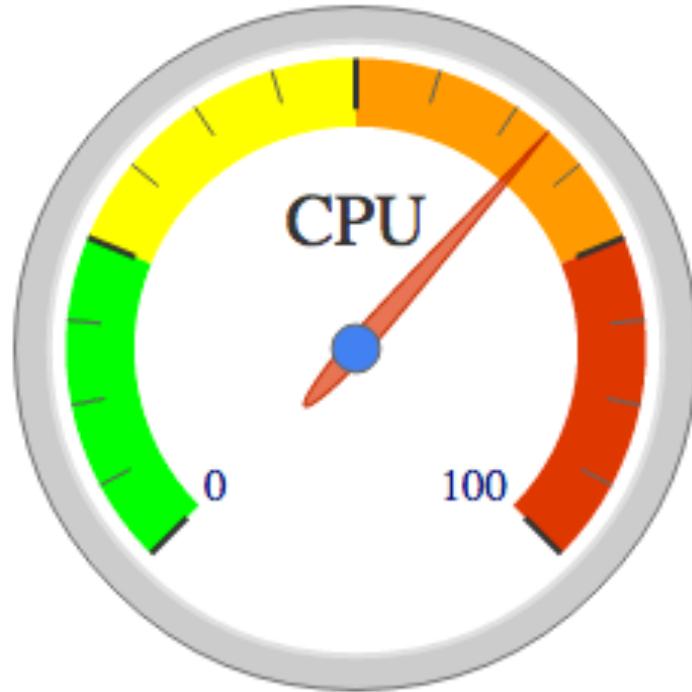


Other Desirables

- Safe for production use
- Easy to use: self service
- [Near] Real Time
- Ad hoc / custom instrumentation
- Complete documentation
- Graph labels and units
- Open source
- Community

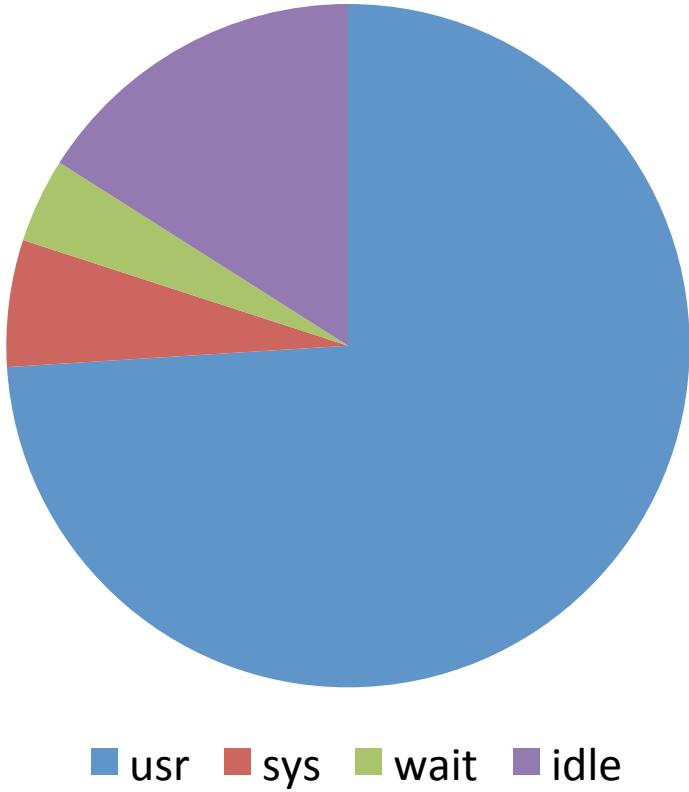
2. Undesirables

Tachometers



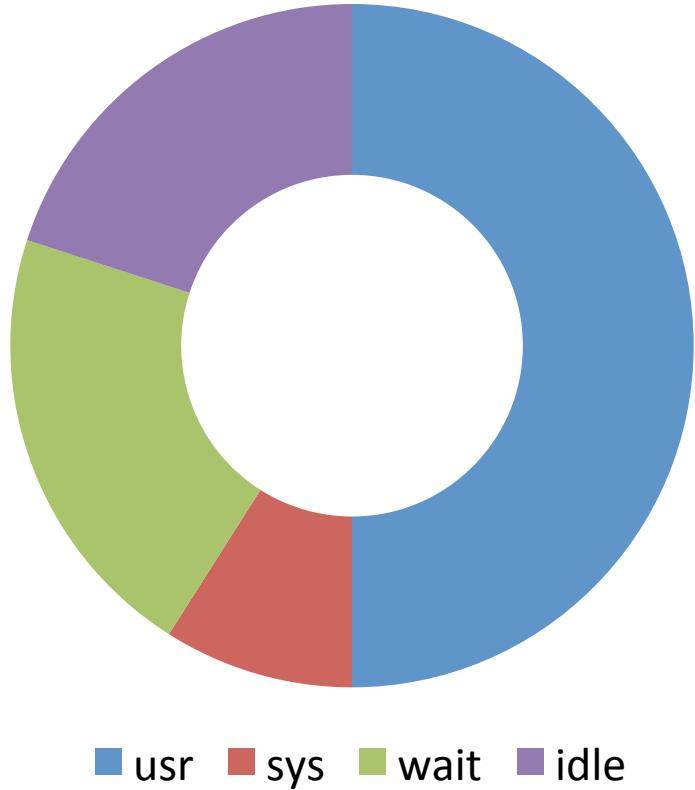
...especially with arbitrary color highlighting

Pie Charts



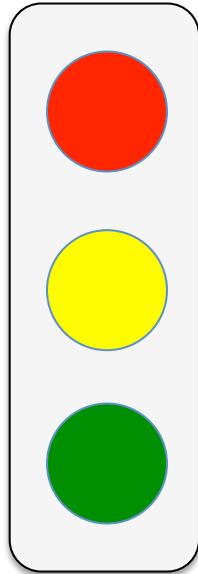
...for real-time metrics

Doughnuts



...like pie charts but worse

Traffic Lights



RED == BAD (usually)

GREEN == GOOD (hopefully)

...when used for *subjective* metrics

These can be used for *objective* metrics

For subjective metrics (eg, IOPS/latency) try weather icons instead

3. Requirements

Acceptable T&Cs

- Probably acceptable:

XXX, Inc. shall have a royalty-free, worldwide, transferable, and perpetual license to use or incorporate into the Service any suggestions, ideas, enhancement requests, feedback, or other information provided by you or any Authorized User relating to the Service.

- Probably **not** acceptable:

By submitting any Ideas, Customer and Authorized Users agree that: ... (iii) all right, title and interest in and to the Ideas, including all associated IP Rights, shall be, and hereby are, **assigned** to [us]

- Check with your legal team

Acceptable Technical Debt

- It must be worth the ...
 - Extra complexity when debugging
 - Time to explain to others
 - Production reliability risk
 - Security risk
- There is no such thing as a free trial

Known Overhead

- Overhead must be known to be managed
 - T&Cs should not prohibit its measurement or publication
- Sources of overhead:
 - CPU cycles
 - File system I/O
 - Network I/O
 - Installed software size
- We will measure it

Low Overhead

- Overhead should also be the lowest possible
 - 1% CPU overhead means 1% more instances, and \$\$\$
- Things we try to avoid
 - Tracing every function/method call
 - Needless kernel/user data transfers
 - strace (ptrace), tcpdump, libpcap, ...
- Event logging doesn't scale

Scalable

- Can the product scale to (say) 100,000 instances?
 - Atlas, our cloud-wide analysis tool, can
 - We tend to kill other monitoring tools that attempt this
- Real-time dashboards showing all instances:
 - How does that work? Can it scale to 1k? ... 100k?
 - Adrian Cockcroft's spigo can simulate protocols at scale
- High overhead might be worth it: on-demand only

Useful

An instance analysis solution must provide
actionable information
that helps us improve performance

4. Methodologies

Methodologies

Methodologies pose the questions
for metrics to answer

Good monitoring/analysis tools should support
performance analysis methodologies

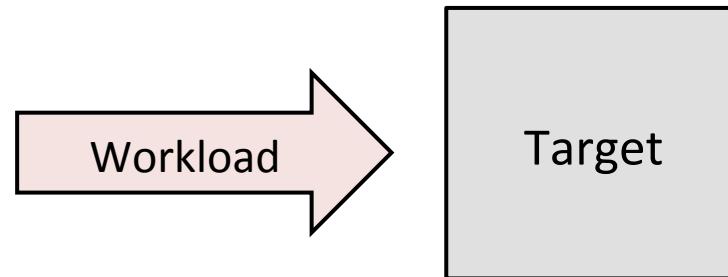
Drunk Man *Anti*-Method

- Tune things at random until the problem goes away

Workload Characterization

Study the workload applied:

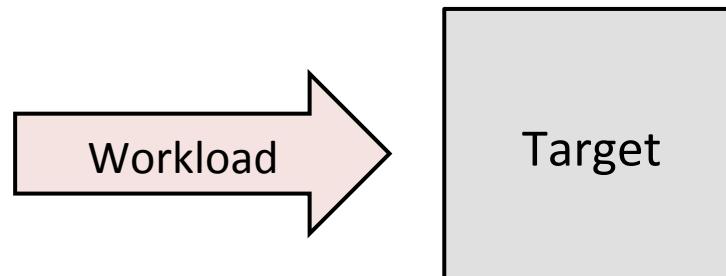
- 1. Who**
- 2. Why**
- 3. What**
- 4. How**



Workload Characterization

Eg, for CPUs:

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

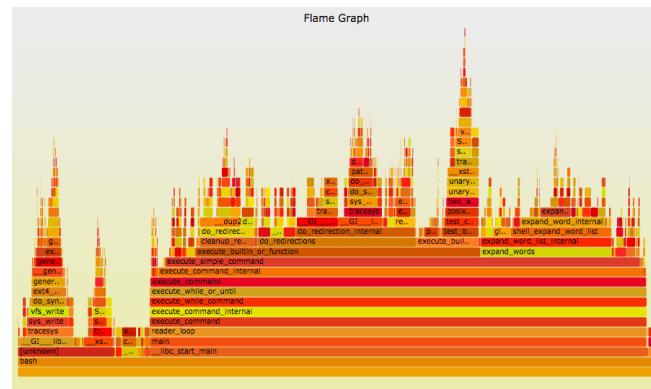


CPUs

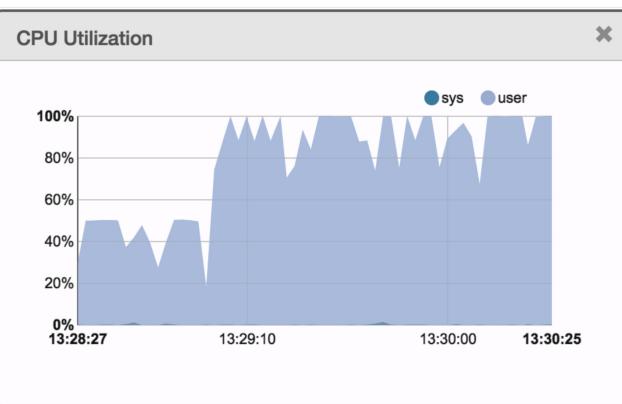
Who

PID	USER	VIRT	RES	CPU%	MEM%	TIME+	Command
27983	root	3233M	204M	147.	2.7	2:10.50	/usr/lib/jvm/java
28004	root	3233M	204M	144.	2.7	2:02.60	/usr/lib/jvm/java
28173	root	63488	4992	95.0	0.1	0:02.68	ab -k -c 100 -n 1
28170	root	24660	2176	3.0	0.0	0:00.62	htop
2730	bgregg-te	202M	58668	0.0	0.8	2h31:25	/apps/epic/perl/b
2752	bgregg-te	151M	10308	0.0	0.1	1h48:36	postgres: bgregg-
28000	root	3233M	204M	0.0	2.7	0:00.26	/usr/lib/jvm/java
1	root	24320	2256	0.0	0.0	0:01.29	/sbin/init
341	root	17236	632	0.0	0.0	0:00.04	upstart-udev-bridge
346	root	21600	1304	0.0	0.0	0:00.06	/sbin/udevd --dae
357	messagebus	23944	1164	0.0	0.0	0:00.21	dbus-daemon --sys
408	root	21464	792	0.0	0.0	0:00.00	/sbin/udevd --dae
549	root	15192	392	0.0	0.0	0:00.00	upstart-socket-br
612	root	7268	1028	0.0	0.0	0:00.24	dhclient3 -e IF_M
644	root	50036	2920	0.0	0.0	0:00.06	/usr/sbin/sshd -D
772	root	14508	956	0.0	0.0	0:00.00	/sbin/getty -8 38
777	root	14508	952	0.0	0.0	0:00.00	/sbin/getty -8 38
785	root	14508	952	0.0	0.0	0:00.00	/sbin/getty -8 38

Why



How



What

```
root@lgud-bgregg:~# perf stat -a -d sleep 10
Performance counter stats for 'system wide':
```

39996.388668 task-clock (msec)	# 3.999 CPUs ut
1,026,540 context-switches	# 0.026 M/sec
193,563 cpu-migrations	# 0.005 M/sec
4,835 page-faults	# 0.121 K/sec
83,859,543,001 cycles	# 2.097 GHz
61,028,919,136 stalled-cycles-frontend	# 72.78% frontend
50,812,852,642 stalled-cycles-backend	# 60.59% backend
52,969,864,055 instructions	# 0.63 insns per c
	# 1.15 stalled
10,223,584,755 branches	# 255.613 M/sec
376,529,869 branch-misses	# 3.68% of all br
0 L1-dcache-loads	# 0.000 K/sec
1,339,950,792 L1-dcache-load-misses	# 0.00% of all br
762,761,193 LLC-loads	# 19.071 M/sec

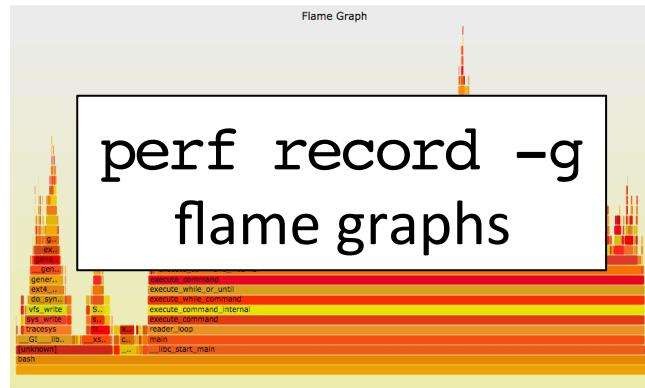
CPUs

Who

PID	USER	VIRT	RES	CPU%	MEM%	TIME+	Command
27983	root	3233M	204M	147.	2.7	2:10.50	/usr/lib/jvm/java
28004	root	3233M	204M	144.	2.7	2:02.60	/usr/lib/jvm/java
28173	root	63488	4992	95.0	0.1	0:02.68	ab -k -c 100 -n 1
28170	root	24660	2176	3.0	0.0	0:00.62	htop
2730	bgregg-te	202M	58668	0.0	0.8	2h31:25	/apps/epic/perl/b
2752	bgregg-te	151M	10308	0	0	0.1	1h48:36
28000	root						/sr/lib/jvm/java
	1	root					bin/init
341	root						start-udev-brid
346	root						bin/udevd --daem
357	messagebus	23944	1104	0.0	0.0	0:00.21	ubus-daemon --sys
408	root	21464	792	0.0	0.0	0:00.00	/sbin/udevd --daem
549	root	15192	392	0.0	0.0	0:00.00	upstart-socket-br
612	root	7268	1028	0.0	0.0	0:00.24	dhclient3 -e IF_M
644	root	50036	2920	0.0	0.0	0:00.06	/usr/sbin/sshd -D
772	root	14508	956	0.0	0.0	0:00.00	/sbin/getty -8 38
777	root	14508	952	0.0	0.0	0:00.00	/sbin/getty -8 38
785	root	14508	952	0.0	0.0	0:00.00	/sbin/getty -8 38

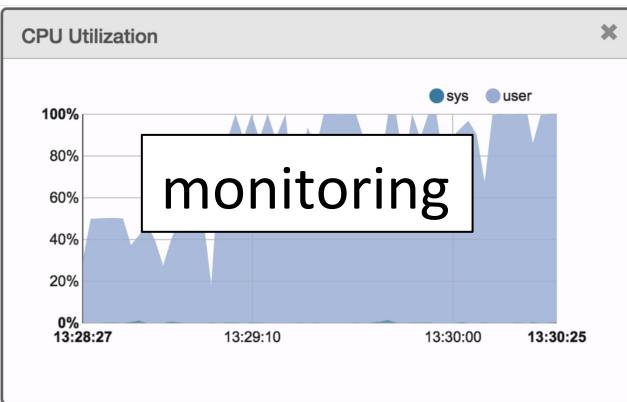
top, htop

Why



```
perf record -g  
flame graphs
```

How



monitoring

What

```
root@gud-bgregg:~# perf stat -a -d sleep 10
Performance counter stats for 'system wide':
      39996.388668 task-clock (msec)          #    3.999 CPUs uti
      1,026,540 context-switches             #    0.026 M/sec

perf stat -a -d
      52,969,864,055 instructions           #    0.63 insns per
                                             #    1.15 stalled p
      10,223,584,755 branches              # 255.613 M/sec
        376,529,869 branch-misses         #  3.68% of all b
          0 L1-dcache-loads               #    0.000 K/sec
      1,339,950,792 L1-dcache-load-misses   #    0.00% of all l
        762,761,193 LLC-loads            # 19.071 M/sec
```

```
perf stat -a -d
```

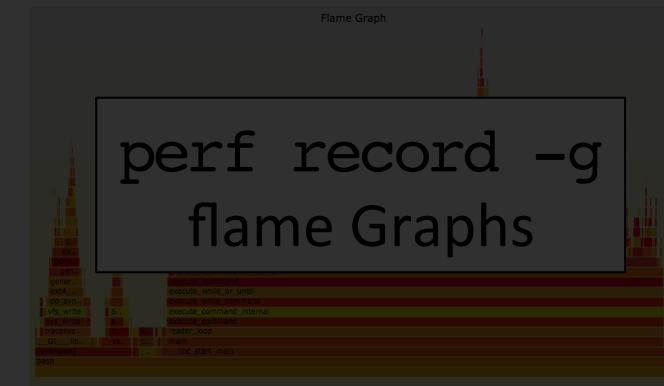
Most Monitoring Products Today

Who

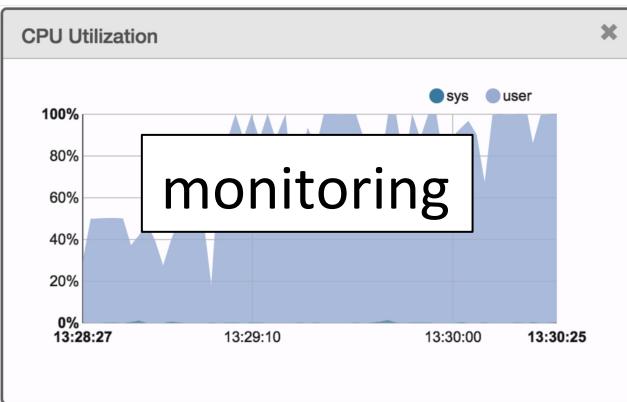
PID	USER	VIRT	RES	CPU%	MEM%	TIME+	Command
27983	root	3233M	204M	147.	2.7	2:10.50	/usr/lib/jvm/java
28004	root	3233M	204M	144.	2.7	2:02.60	/usr/lib/jvm/java
28173	root	63488	4992	95.0	0.1	0:02.68	ab -k -c 100 -n 1
28170	root	24660	2176	3.0	0.0	0:00.62	htop
2730	bgregg-te	202M	58668	0.0	0.8	2h31:25	/apps/epic/perl/b
2752	bgregg-te	151M	10308	0.0	0.1	1h48:36	postgres: bgregg-
28000	root						/sr/lib/jvm/java
1	root						bin/init
341	root						start-udev-bride
346	root						bin/udevd --dae
357	messagebus	23944	1164	0.0	0.0	0:00.21	ibus-daemon --sys
408	root	21464	792	0.0	0.0	0:00.00	/sbin/udevd --dae
549	root	15192	392	0.0	0.0	0:00.00	upstart-socket-br
612	root	7268	1028	0.0	0.0	0:00.24	dhclient3 -e IF_M
644	root	50036	2920	0.0	0.0	0:00.06	/usr/sbin/sshd -D
772	root	14508	956	0.0	0.0	0:00.00	/sbin/getty -8 38
777	root	14508	952	0.0	0.0	0:00.00	/sbin/getty -8 38
785	root	14508	952	0.0	0.0	0:00.00	/sbin/getty -8 38

top, htop

Why



How



What

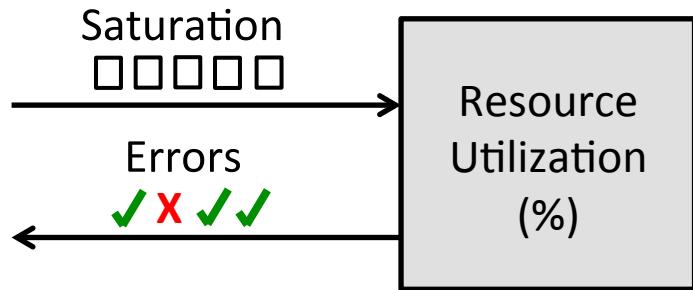
```
root@lgud-bgregg:~# perf stat -a -d sleep 10
Performance counter stats for 'system wide':
                                         39996.388668 task-clock (msec)      #    3.999 CPUs ut
                                         1,026,540 context-switches      #    0.026 M/sec
                                         52,969,864,055 instructions      #    0.63 insns p
                                         10,223,584,755 branches        #   1.15 stalled
                                         376,529,869 branch-misses     #   3.68% of all b
                                         0 L1-dcache-loads            #   0.000 K/sec
                                         1,339,950,792 L1-dcache-load-misses #   0.00% of all l
                                         762,761,193 LLC-Loads         # 19.071 M/sec
```

perf stat -a -d

The USE Method

- For every resource, check:

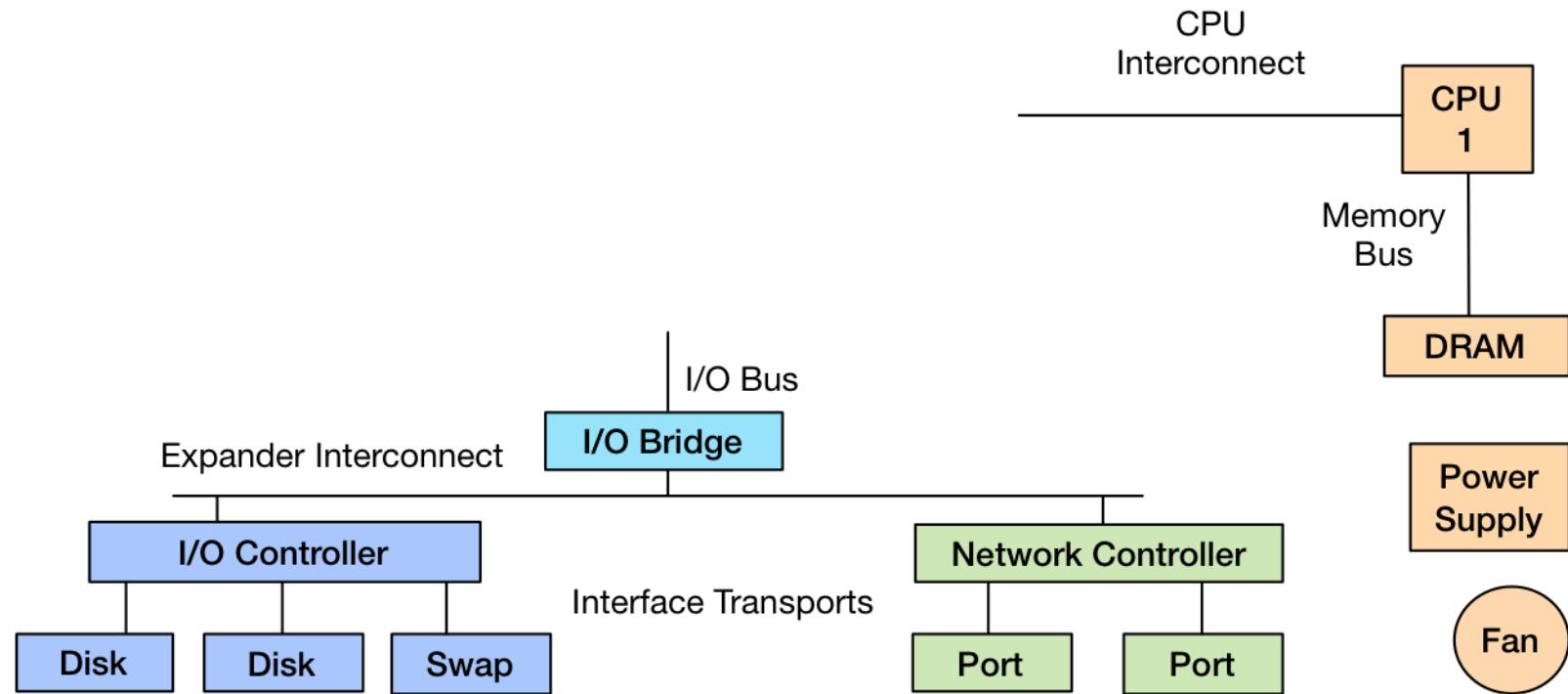
- Utilization**
- Saturation**
- Errors**



- Saturation is queue length or queued time
- Start by drawing a functional (block) diagram of your system / software / environment

USE Method for Hardware

Include busses & interconnects!



USE Method: Linux Performance Checklist

The [USE Method](#) provides a strategy for performing a complete check of system health, identifying common bottlenecks and errors. For each system resource, metrics for utilization, saturation and errors are identified and checked. Any issues discovered are then investigated using further strategies.

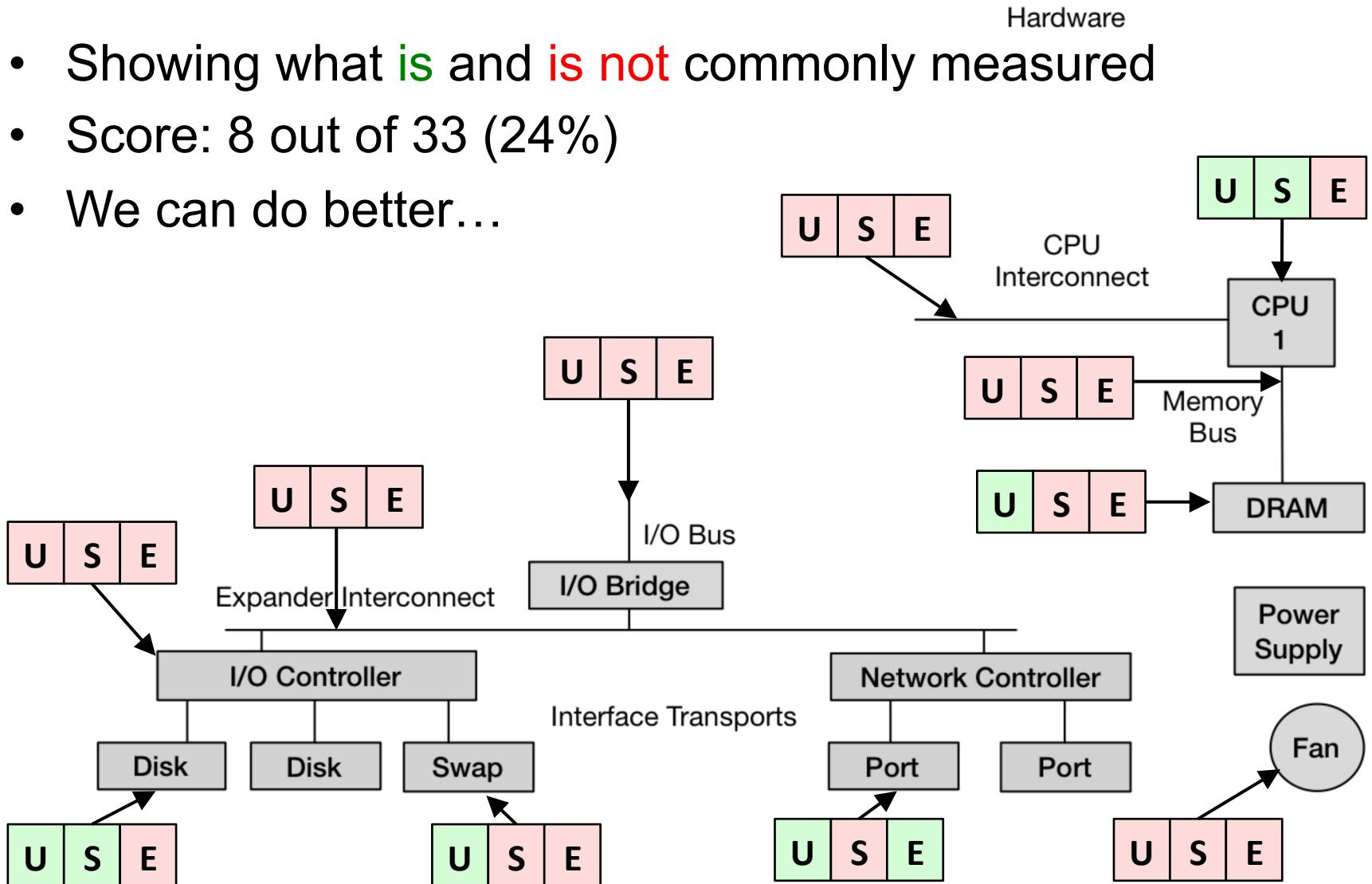
This is an example USE-based metric list for Linux operating systems (eg, Ubuntu, CentOS, Fedora). This is primarily intended for system administrators of the physical systems, who are using command line tools. Some of these metrics can be found in remote monitoring tools.

Physical Resources <http://www.brendangregg.com/USEmethod/use-linux.html>

component	type	metric
CPU	utilization	system-wide: <code>vmstat 1, "us" + "sy" + "st"; sar -u</code> , sum fields except "%idle" and "%iowait"; <code>dstat -c</code> , sum fields except "idl" and "wai"; per-cpu: <code>mpstat -P ALL 1</code> , sum fields except "%idle" and "%iowait"; <code>sar -P ALL</code> , same as <code>mpstat</code> ; per-process: <code>top, "%CPU"; htop, "CPU%"</code> ; <code>ps -o pcpu</code> ; <code>pidstat 1, "%CPU"</code> ; per-kernel-thread: <code>top/htop ("K" to toggle)</code> , where VIRT == 0 (heuristic). [1]
CPU	saturation	system-wide: <code>vmstat 1, "r" > CPU count</code> [2]; <code>sar -q, "runq-sz" > CPU count</code> ; <code>dstat -p, "run" > CPU count</code> ; per-process: <code>/proc/PID/schedstat</code> 2nd field (sched_info.run_delay); <code>perf sched latency</code> (shows "Average" and "Maximum" delay per-schedule); dynamic tracing, eg, SystemTap schedtimes.stp "queued(us)" [3]
CPU	errors	<code>perf (LPE)</code> if processor specific error events (CPC) are available; eg, AMD64's "04Ah Single-bit ECC Errors Recorded by Scrubber" [4]
Memory capacity	utilization	system-wide: <code>free -m, "Mem:"</code> (main memory), <code>"Swap:"</code> (virtual memory); <code>vmstat 1, "free"</code> (main memory), <code>"swap"</code> (virtual memory); <code>sar -r, "%memused"</code> ; <code>dstat -m, "free"</code> ; <code>slabtop -s c</code> for kmem slab usage; per-process: <code>top/htop, "RES"</code> (resident main memory), <code>"VIRT"</code> (virtual memory), <code>"Mem"</code> for system-wide summary
Memory capacity	saturation	system-wide: <code>vmstat 1, "si"/"so"</code> (swapping); <code>sar -B, "pgscank" + "pgscand"</code> (scanning); <code>sar -W</code> ; per-process: 10th field (min_flt) from <code>/proc/PID/stat</code> for minor-fault rate, or dynamic tracing [5]; OOM killer: <code>dmesg grep killed</code>
Memory capacity	errors	<code>dmesg</code> for physical failures; dynamic tracing, eg, SystemTap uprobes for failed malloc()
Network Interfaces	utilization	<code>sar -n DEV 1, "rxKB/s"/max "txKB/s"/max</code> ; <code>ip -s link, RX/TX tput / max bandwidth</code> ; <code>/proc/net/dev, "bytes"</code> RX/TX tput/max; <code>nicstat "%Util"</code> [6]
Network Interfaces	saturation	<code>ifconfig, "overruns", "dropped"</code> ; <code>netstat -s, "segments retransmited"</code> ; <code>sar -n EDEV, *drop and *fifo metrics</code> ; <code>/proc/net/dev, RX/TX "drop"</code> ; <code>nicstat "Sat"</code> [6]; dynamic tracing for other TCP/IP stack queueing [7]
Network Interfaces	errors	<code>ifconfig, "errors", "dropped"</code> ; <code>netstat -i, "RX-ERR"/"TX-ERR"</code> ; <code>ip -s link, "errors"</code> ; <code>sar -n EDEV, "rxerr/s"</code> "txerr/s"; <code>/proc/net/dev, "errs", "drop"</code> ; extra counters may be under <code>/sys/class/net/...</code> ; dynamic tracing of driver function returns 76]

Most Monitoring Products Today

- Showing what **is** and **is not** commonly measured
- Score: 8 out of 33 (24%)
- We can do better...



Other Methodologies

- There are many more:
 - Drill-Down Analysis Method
 - Time Division Method
 - Stack Profile Method
 - Off-CPU Analysis
 - ...
 - I've covered these in previous talks & books

5. Our Tools



Atlas



Netflix Open Source Software

BaseAMI

- Many sources for instance metrics & analysis
 - Atlas, Vector, sar, perf-tools (ftrace, perf_events), ...
- Currently not using 3rd party monitoring vendor tools

Linux (usually Ubuntu)

Optional Apache,
memcached, Node.js,
...

Atlas, S3 log rotation,
sar, ftrace, perf, stap,
perf-tools

Vector, pcp

Java (JDK 7 or 8)

GC and
thread
dump
logging

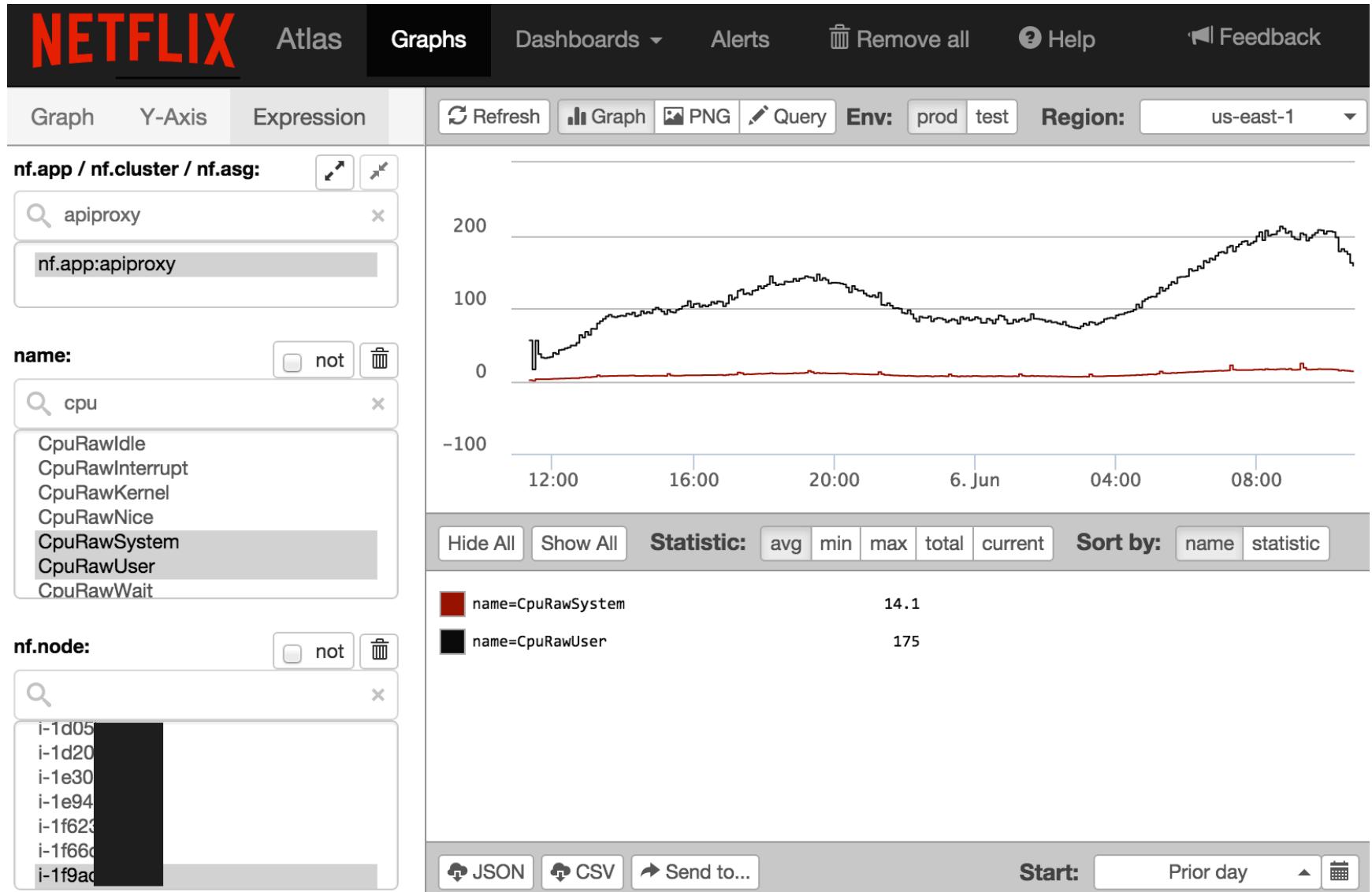
Tomcat

Application war files,
platform, base servlet

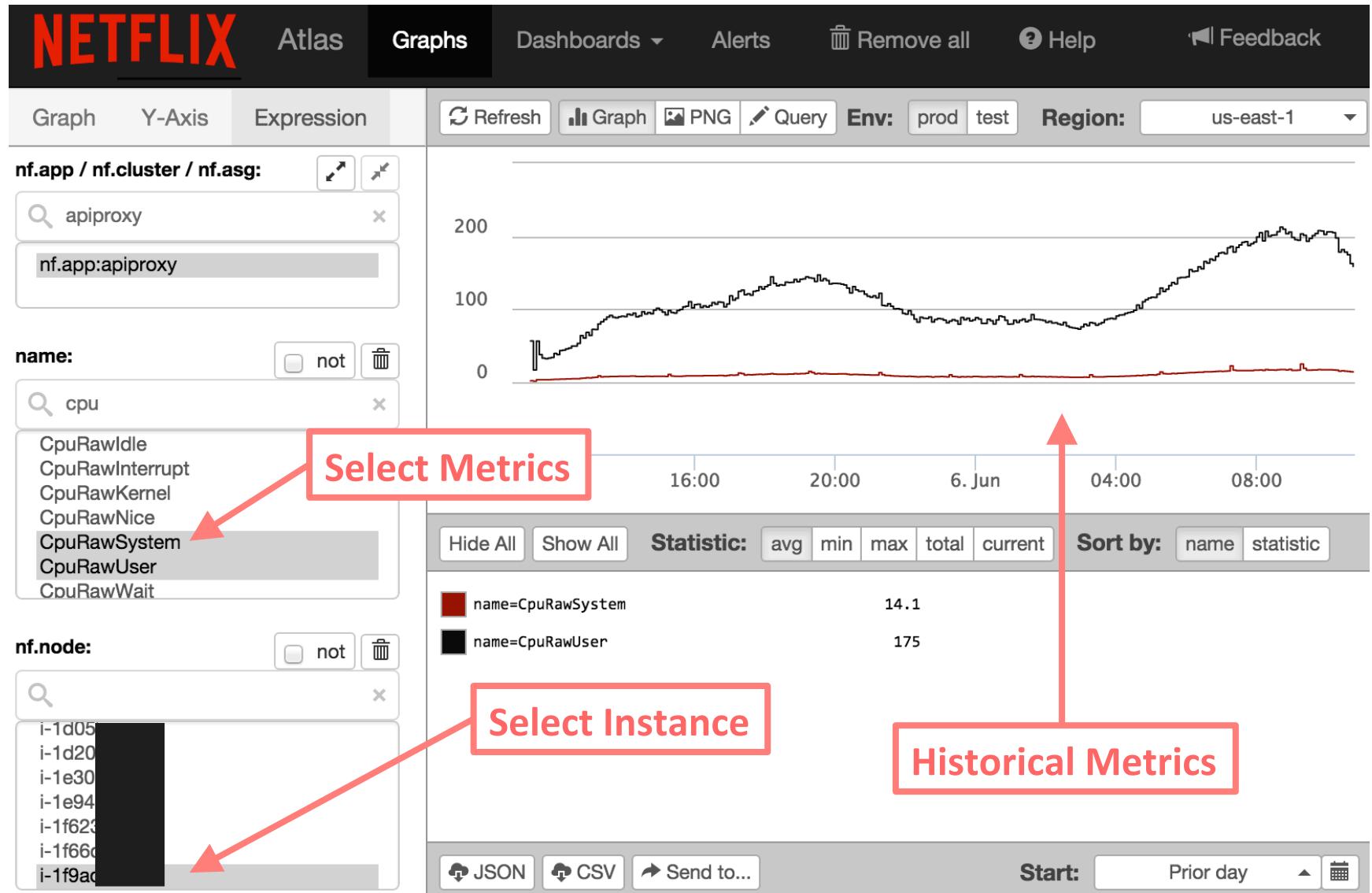
hystrix, metrics (Servo),
health check



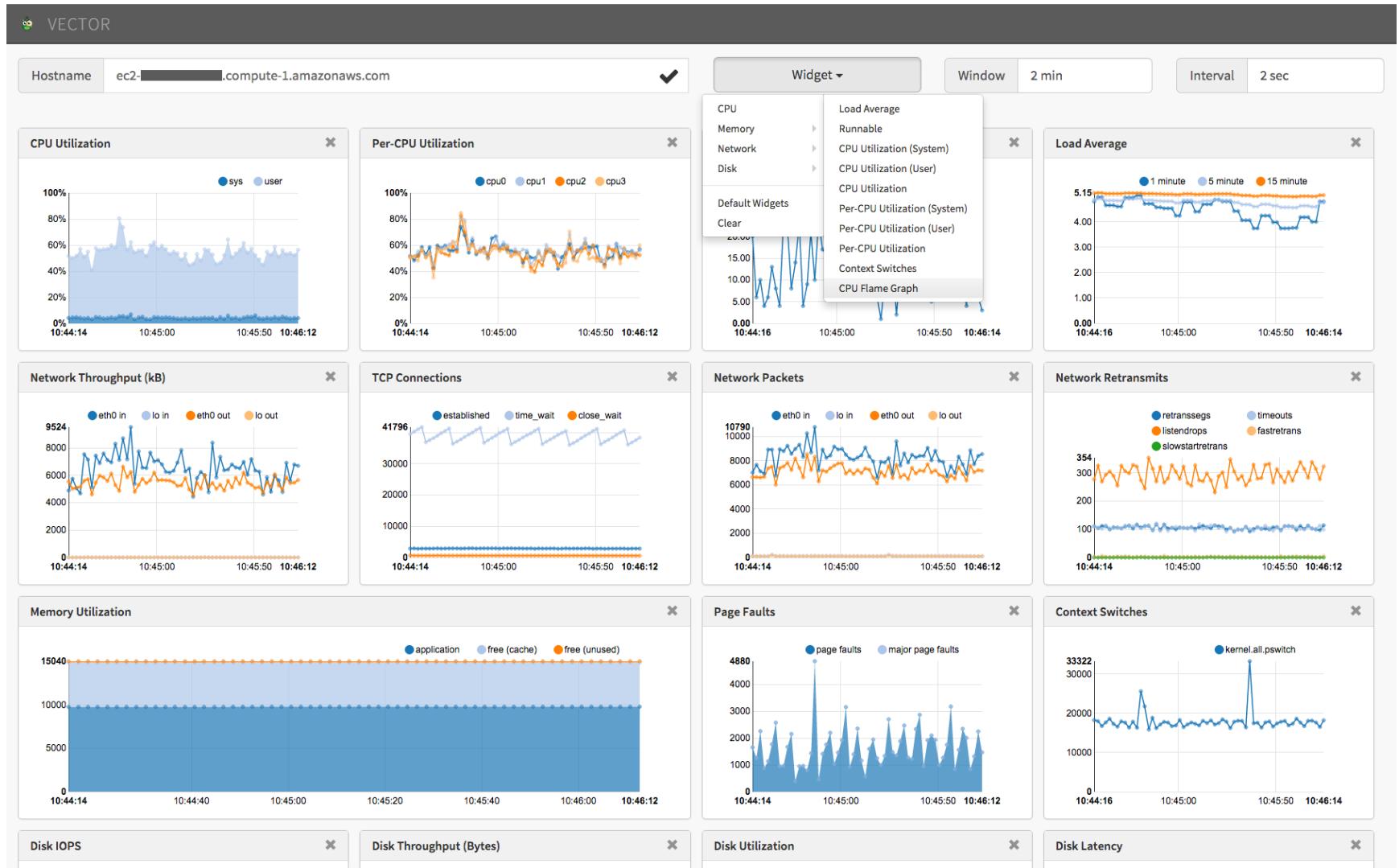
Netflix Atlas



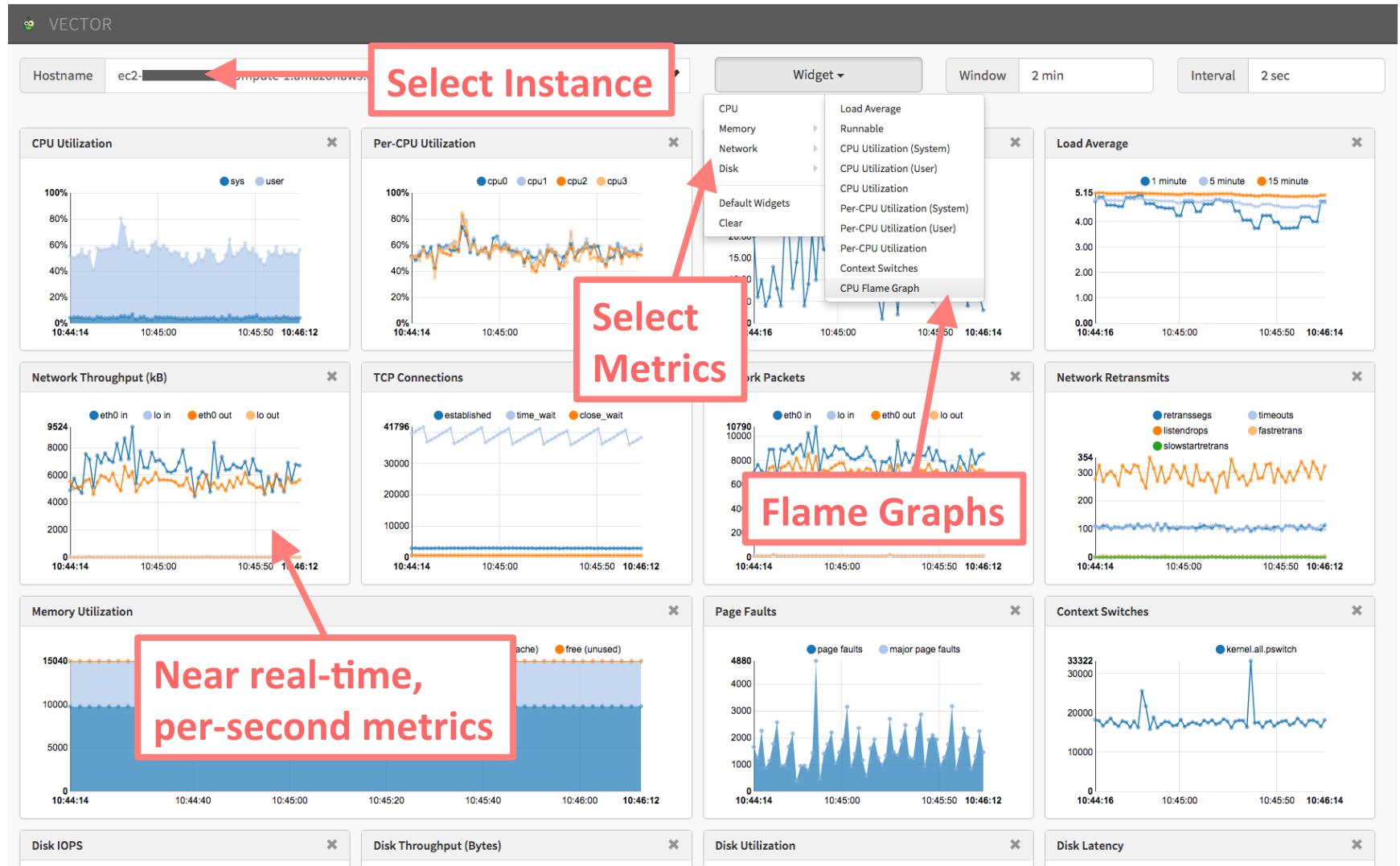
Netflix Atlas



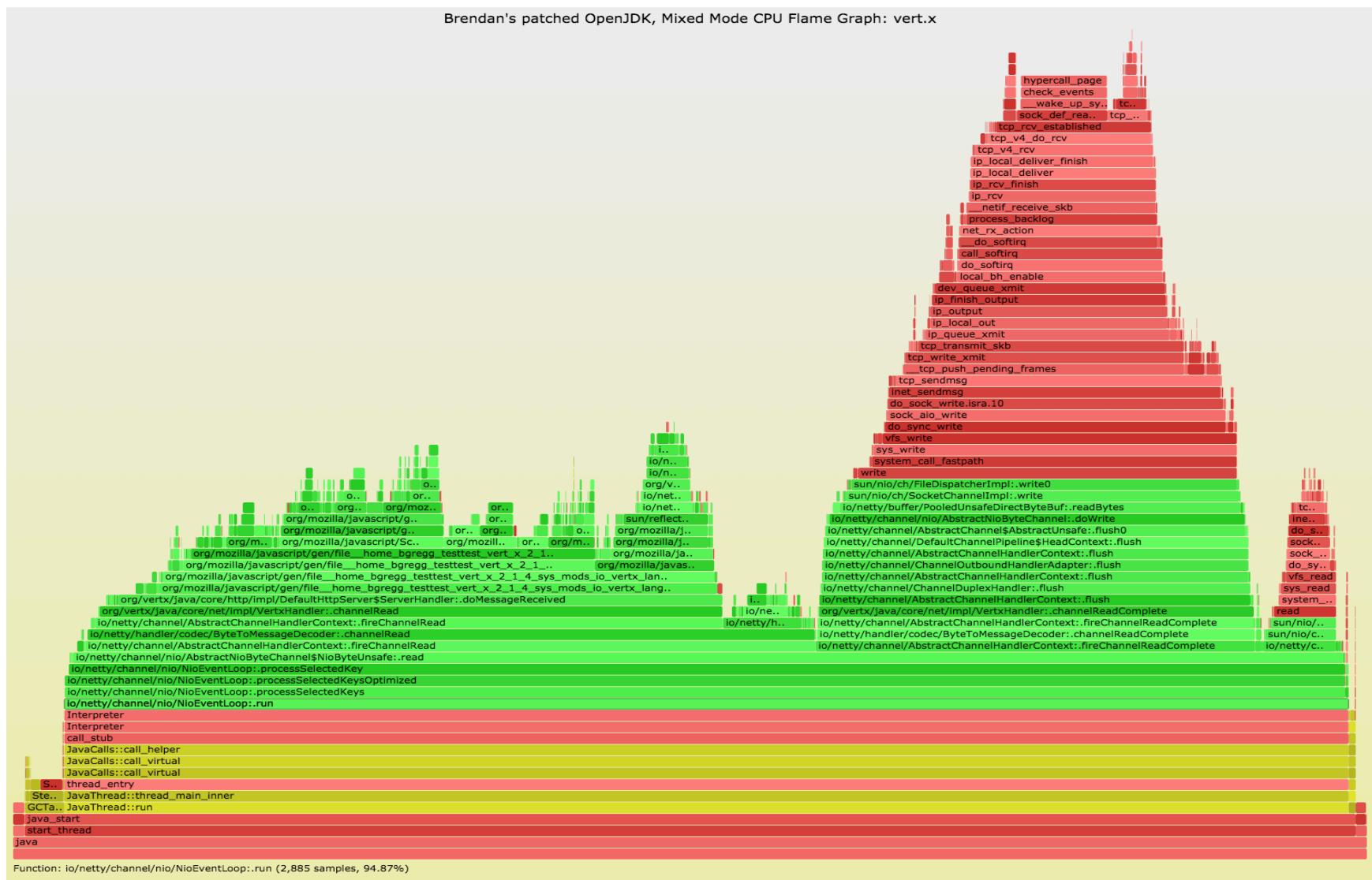
Netflix Vector



Netflix Vector



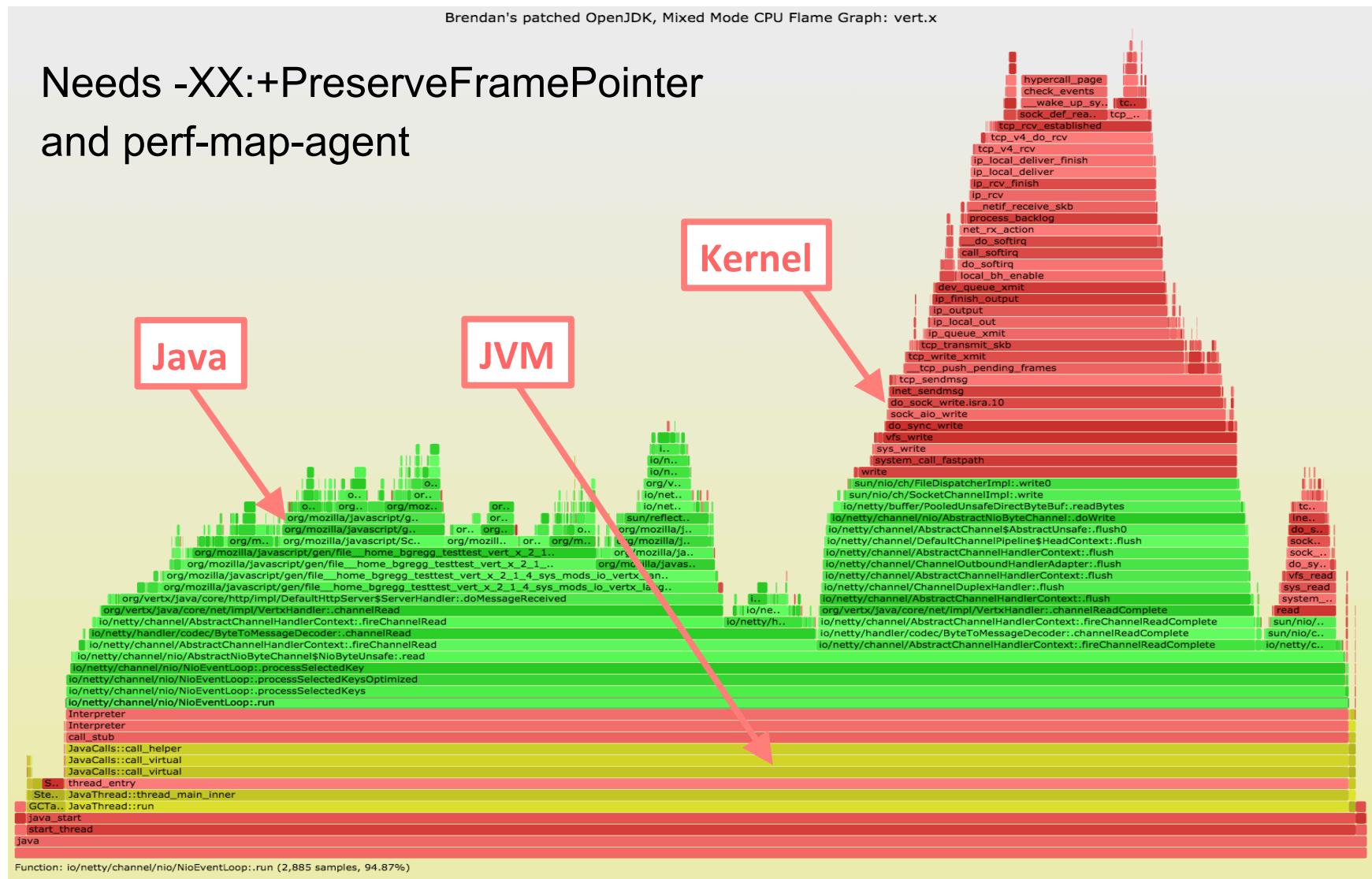
Java CPU Flame Graphs



Java CPU Flame Graphs

Brendan's patched OpenJDK, Mixed Mode CPU Flame Graph: vert.x

Needs -XX:+PreserveFramePointer
and perf-map-agent



sar

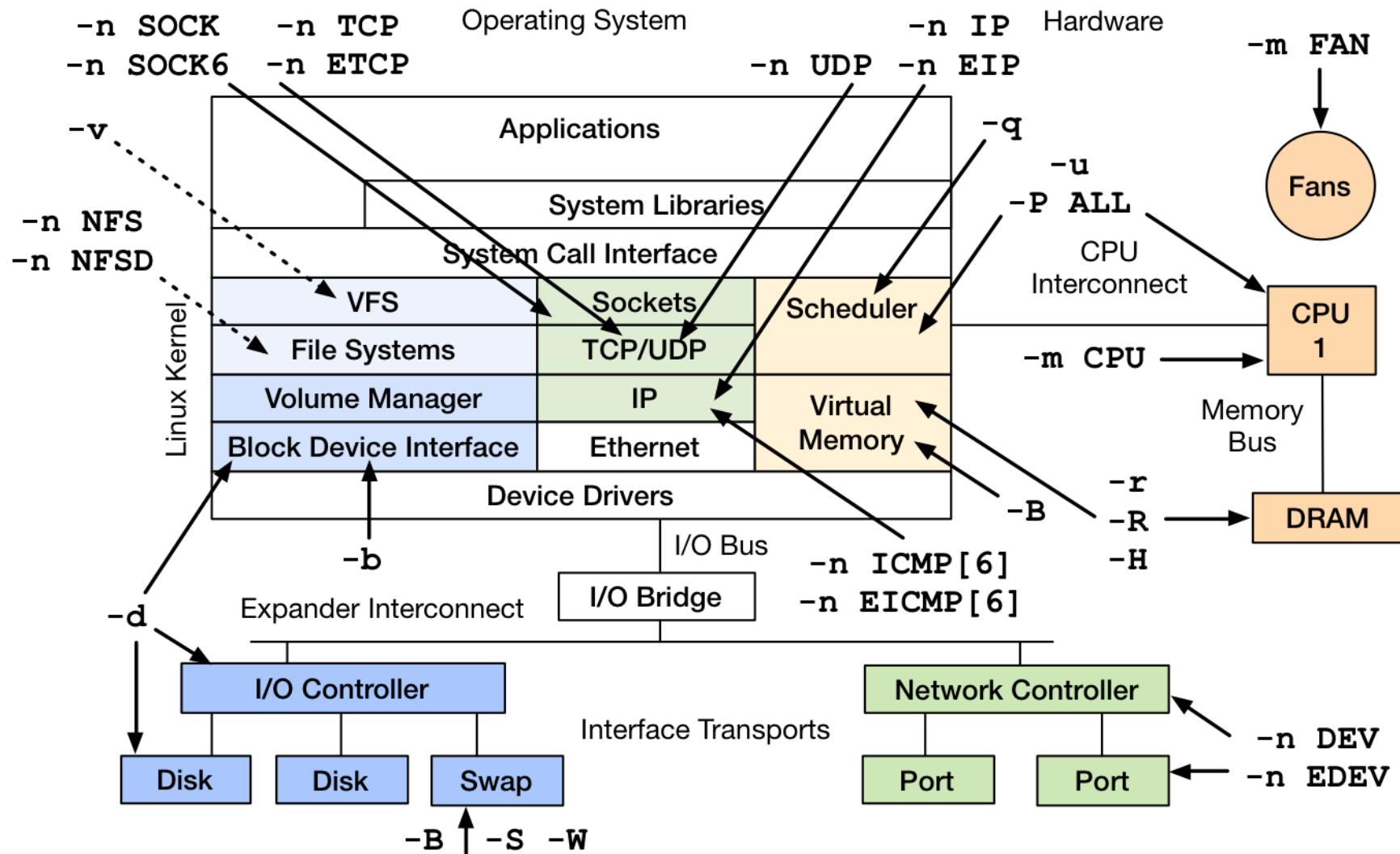
- System Activity Reporter. Archive of metrics, eg:

```
$ sar -n DEV
Linux 3.13.0-49-generic (prod0141)        06/06/2015 _x86_64_    (16 CPU)

12:00:01 AM      IFACE   rxpck/s   txpck/s   rxB/s   kB/s   rxcmp/s   txcmp/s   rxmcst/s %ifutil
12:05:01 AM      eth0     4824.26   3941.37   919.57  15706.14   0.00     0.00     0.00     0.00
12:05:01 AM      lo       23913.29  23913.29  17677.23 17677.23   0.00     0.00     0.00     0.00
12:15:01 AM      eth0     4507.22   3749.46   909.03  12481.74   0.00     0.00     0.00     0.00
12:15:01 AM      lo       23456.94  23456.94  14424.28 14424.28   0.00     0.00     0.00     0.00
12:25:01 AM      eth0     10372.37  9990.59   1219.22  27788.19   0.00     0.00     0.00     0.00
12:25:01 AM      lo       25725.15  25725.15  29372.20 29372.20   0.00     0.00     0.00     0.00
12:35:01 AM      eth0     4729.53   3899.14   914.74  12773.97   0.00     0.00     0.00     0.00
12:35:01 AM      lo       23943.61  23943.61  14740.62 14740.62   0.00     0.00     0.00     0.00
[...]
```

- Metrics are also in Atlas and Vector
- Linux sar is well designed: units, groups

sar Observability



perf-tools

- Some front-ends to Linux ftrace & perf_events
 - Advanced, custom kernel observability when needed (rare)
 - <https://github.com/brendangregg/perf-tools>
 - Unsupported hacks: see WARNINGS
- ftrace
 - First added to Linux 2.6.27
 - A collection of capabilities, used via /sys/kernel/debug/tracing/
- perf_events
 - First added to Linux 2.6.31
 - Tracer/profiler multi-tool, used via "perf" command

perf-tools: funccount

- Eg, count a kernel function call rate:

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

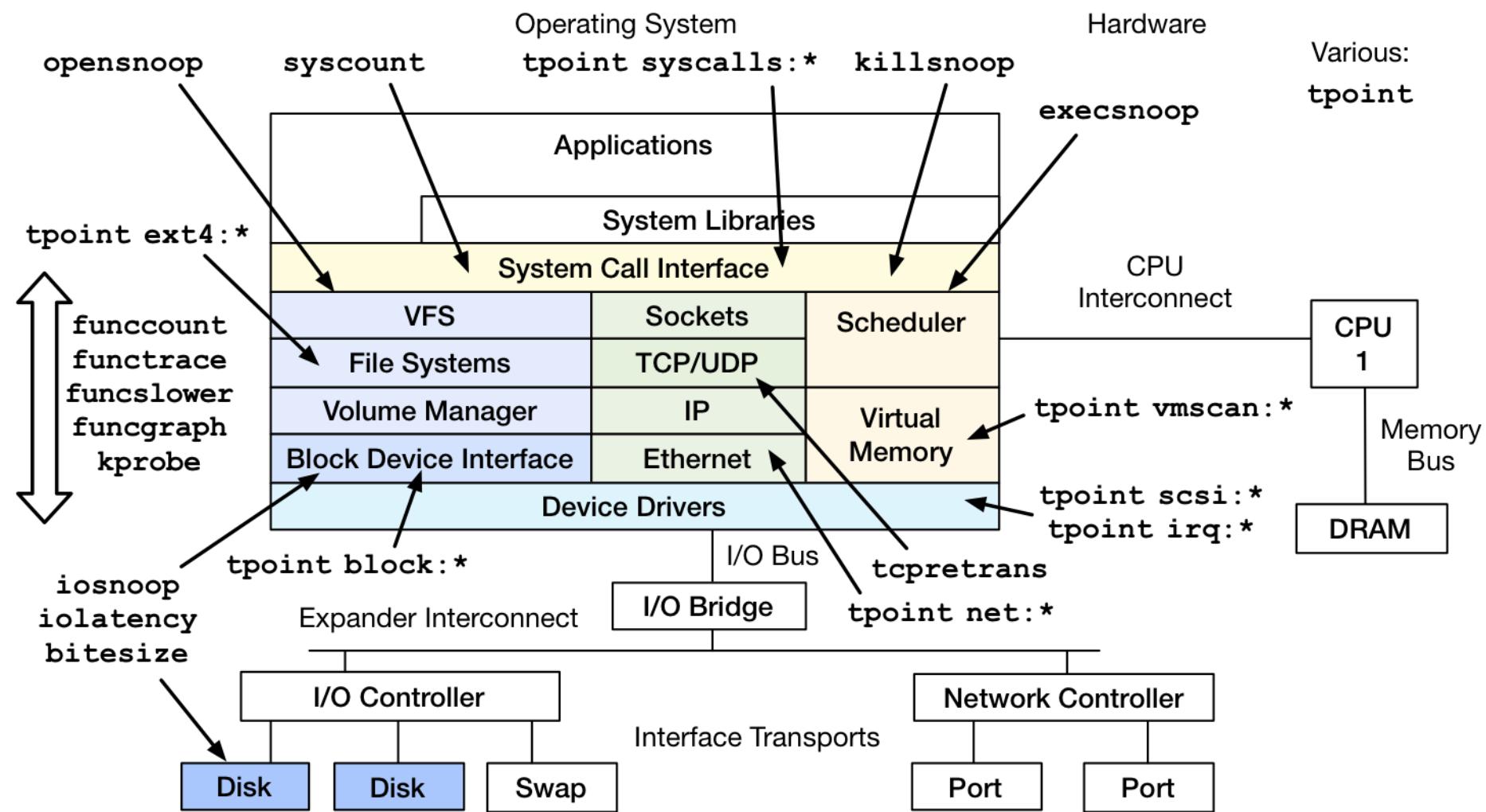
FUNC	COUNT
bio_attempt_back_merge	26
bio_get_nr_vecs	361
bio_alloc	536
bio_alloc_bioset	536
bio_endio	536
bio_free	536
bio_fs_destructor	536
bio_init	536
bio_integrity_enabled	536
bio_put	729
bio_add_page	1004

Counts are in-kernel,
for low overhead

[...]

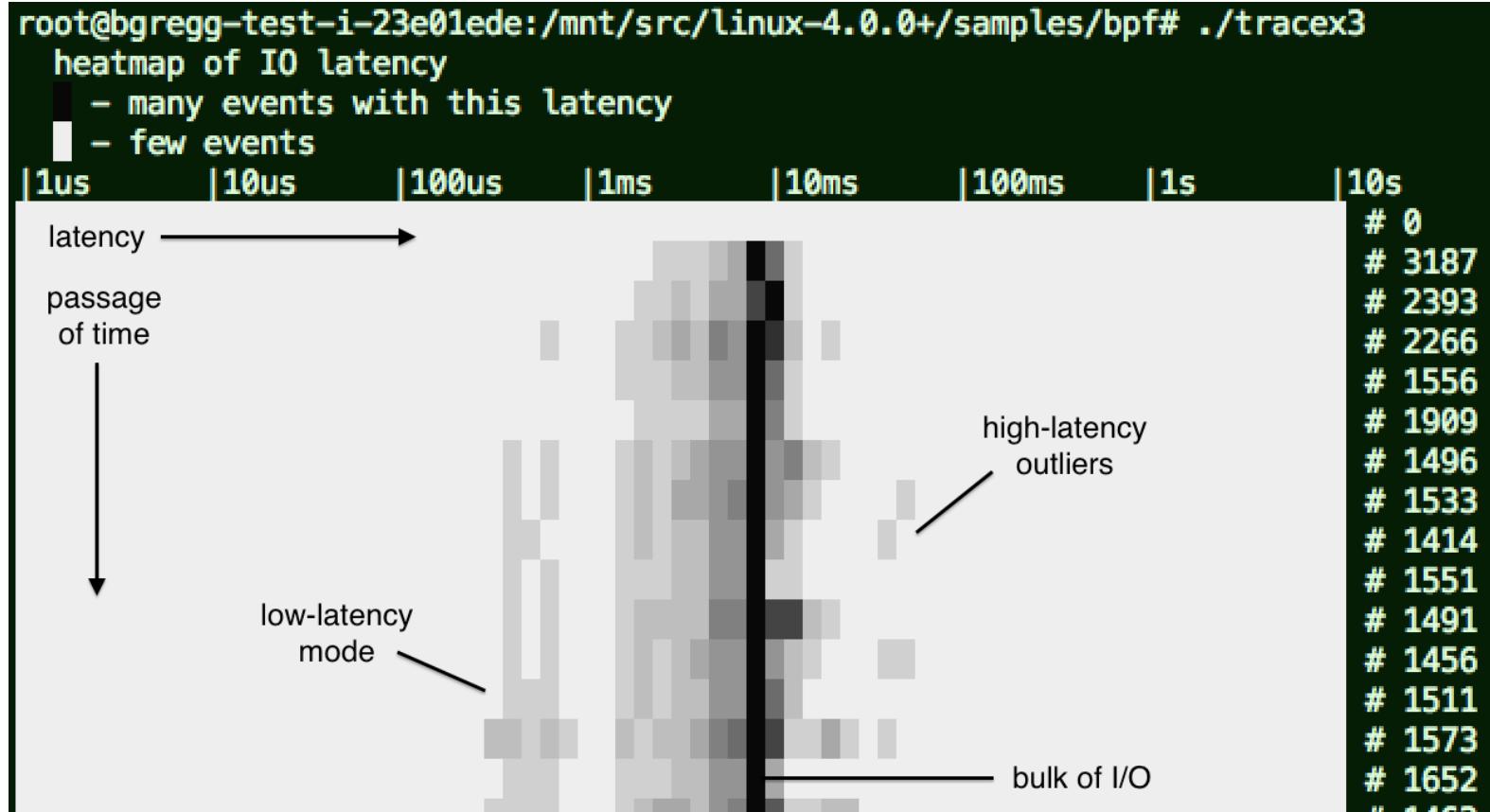
- Other perf-tools can then instrument these in more detail

perf-tools (so far...)



eBPF

- Currently being integrated. Efficient (JIT) in-kernel maps.
- Measure latency, heat maps, ...



eBPF



eBPF will make a profound difference to monitoring on Linux systems

There will be an arms race to support it, post Linux 4.1+
If it's not on your roadmap, it should be

Summary

Requirements

- Acceptable T&Cs
- Acceptable technical debt
- Known overhead
- Low overhead
- Scalable
- Useful

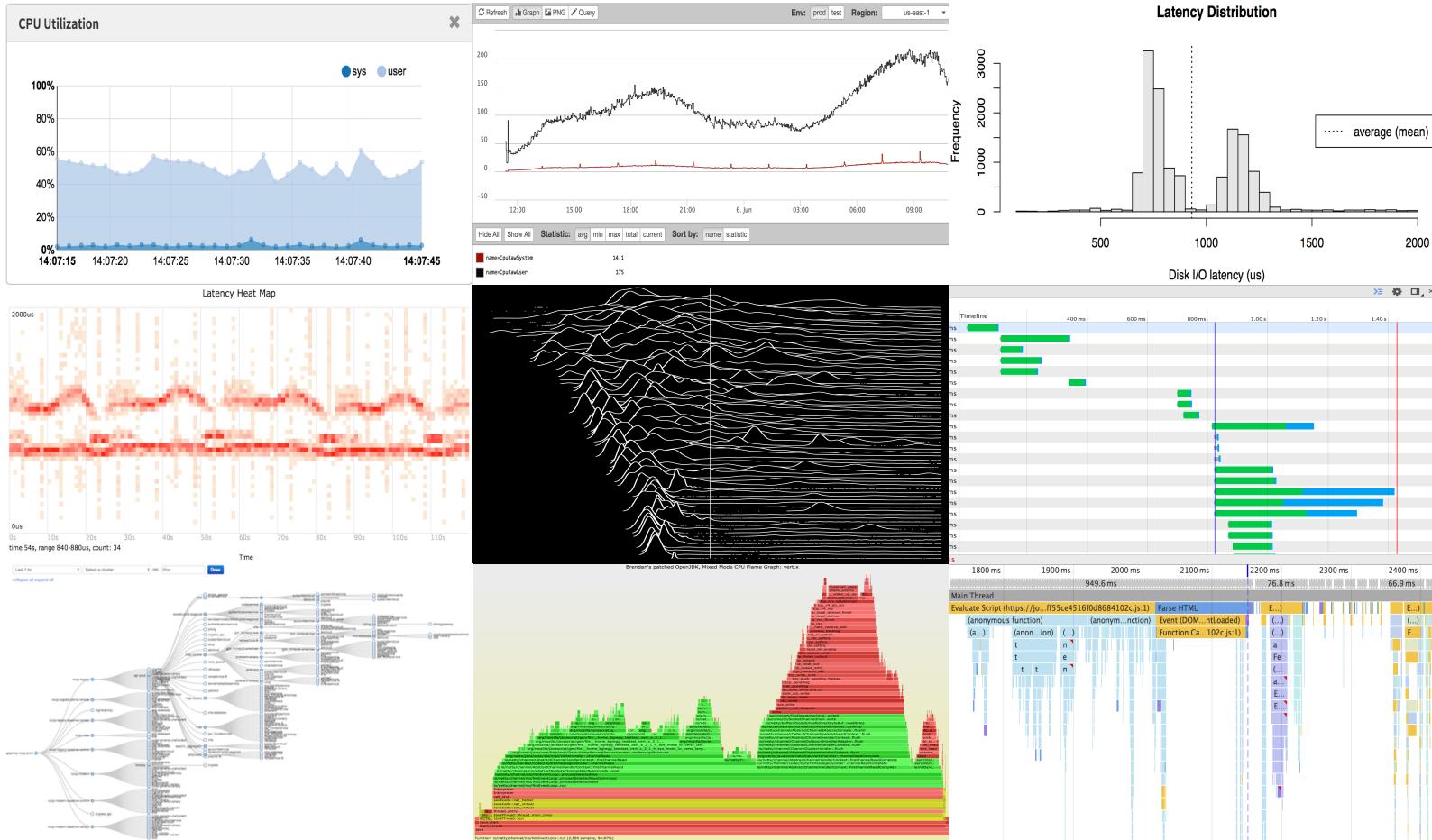
Methodologies

Support for:

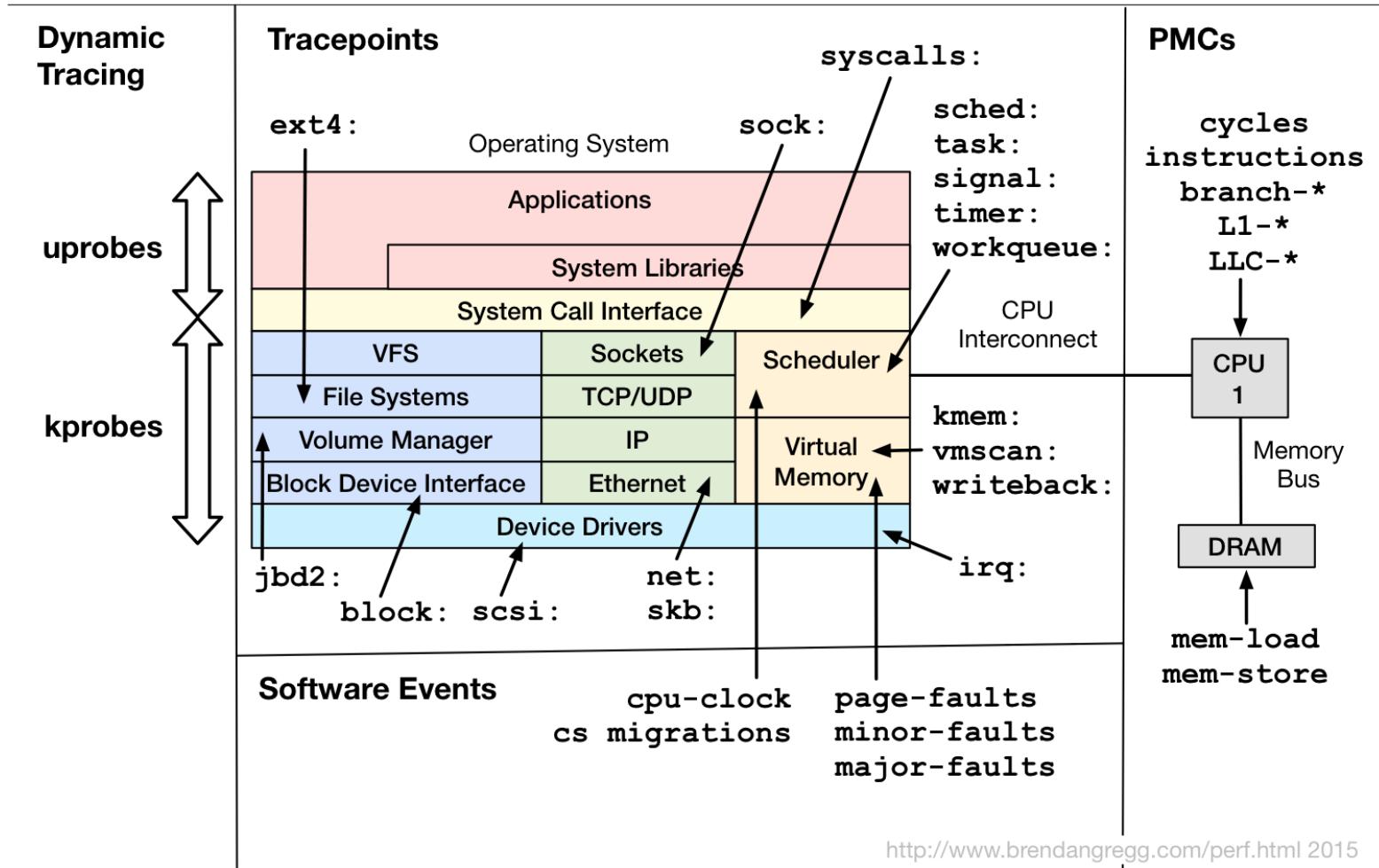
- Workload Characterization
- The USE Method
- ...

Not starting with metrics in search of uses

Desirables



Instrument These



With full eBPF support
Linux has awesome instrumentation: use it!

Links & References

- Netflix Vector
 - <https://github.com/netflix/vector>
 - <http://techblog.netflix.com/2015/04/introducing-vector-netflixs-on-host.html>
- Netflix Atlas
 - <http://techblog.netflix.com/2014/12/introducing-atlas-netflixs-primary.html>
- Heat Maps
 - <http://www.brendangregg.com/heatmaps.html>
 - <http://www.brendangregg.com/HeatMaps/latency.html>
- Flame Graphs
 - <http://www.brendangregg.com/flamegraphs.html>
 - <http://techblog.netflix.com/2014/11/nodejs-in-flames.html>
- Frequency Trails: <http://www.brendangregg.com/frequencytrails.html>
- Methodology
 - <http://www.brendangregg.com/methodology.html>
 - <http://www.brendangregg.com/USEmethod/use-linux.html>
- perf-tools: <https://github.com/brendangregg/perf-tools>
- eBPF: <http://www.brendangregg.com/blog/2015-05-15/ebpf-one-small-step.html>
- Images:
 - horse: Microsoft Powerpoint clip art
 - gauge: <https://github.com/thlorenz/d3-gauge>
 - eBPF ponycorn: Deirdré Straughan & General Zoi's Pony Creator



Thanks

- Questions?
- <http://techblog.netflix.com>
- <http://slideshare.net/brendangregg>
- <http://www.brendangregg.com>
- bgregg@netflix.com
- [@brendangregg](https://twitter.com/brendangregg)

NETFLIX

