Performance Checklists for SREs

Brendan Gregg
Senior Performance Architect

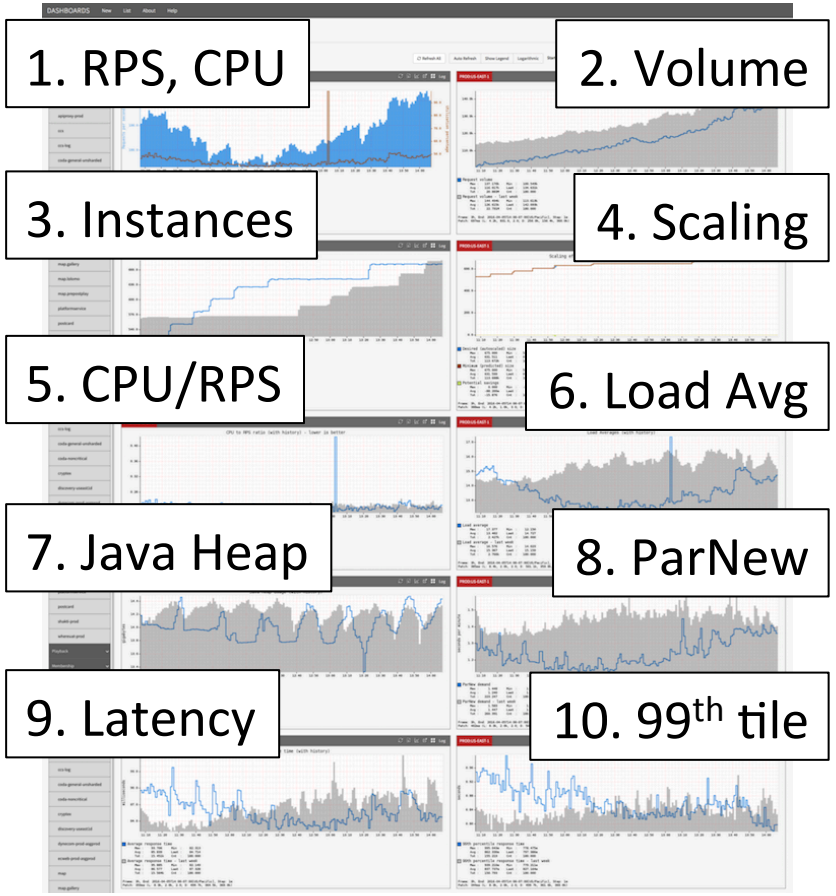
NETFLIX

Performance Checklists

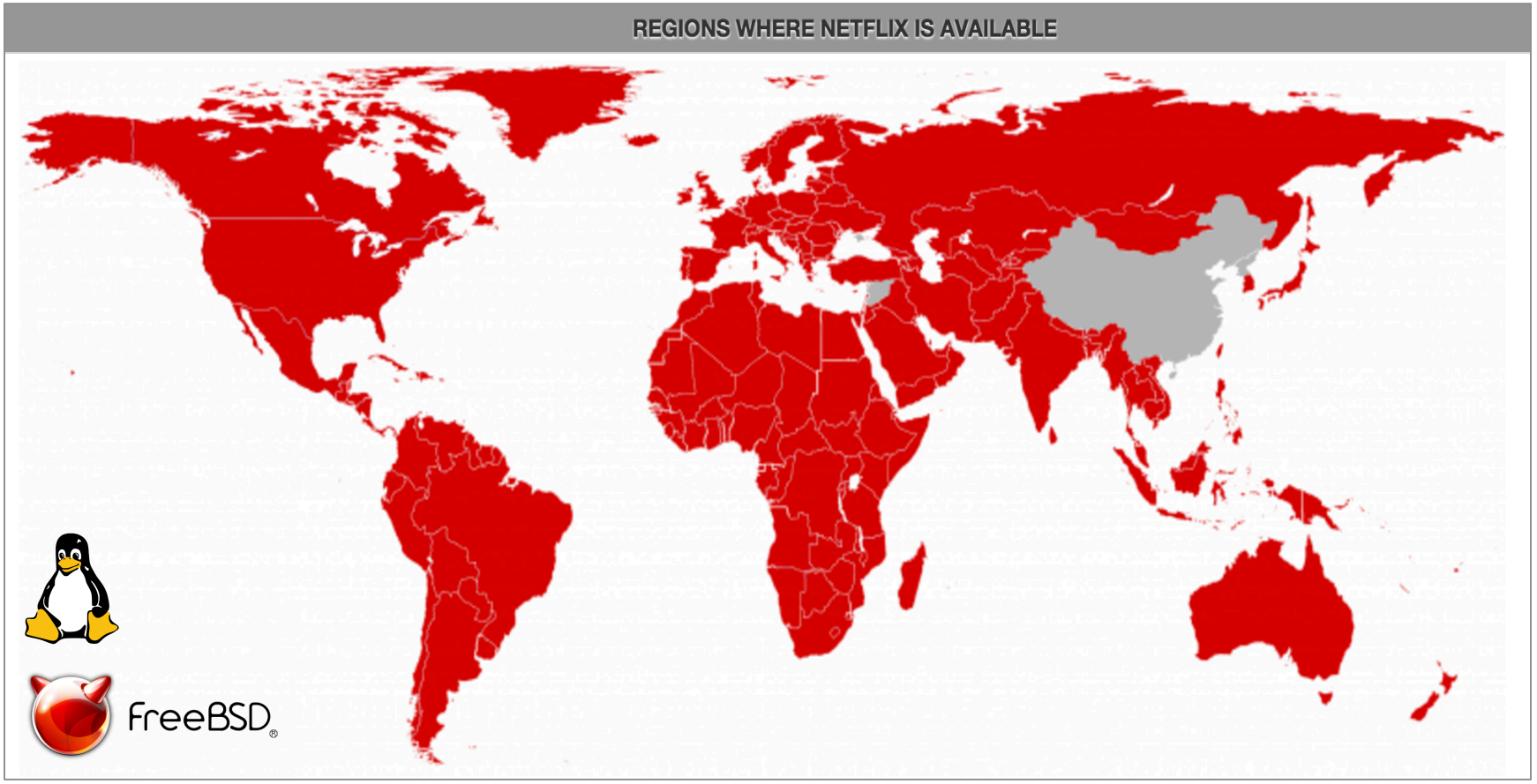
per instance:

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`

cloud wide:



NETFLIX



Brendan the SRE

- On the Perf Eng team & primary on-call rotation for Core: our central SRE team
  - we get paged on SPS dips (starts per second) & more

- In this talk I'll condense some perf engineering into SRE timescales (minutes) using checklists

Performance Engineering

!=

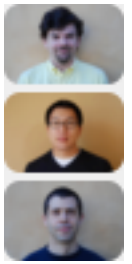
SRE Performance

Incident Response

Performance Engineering

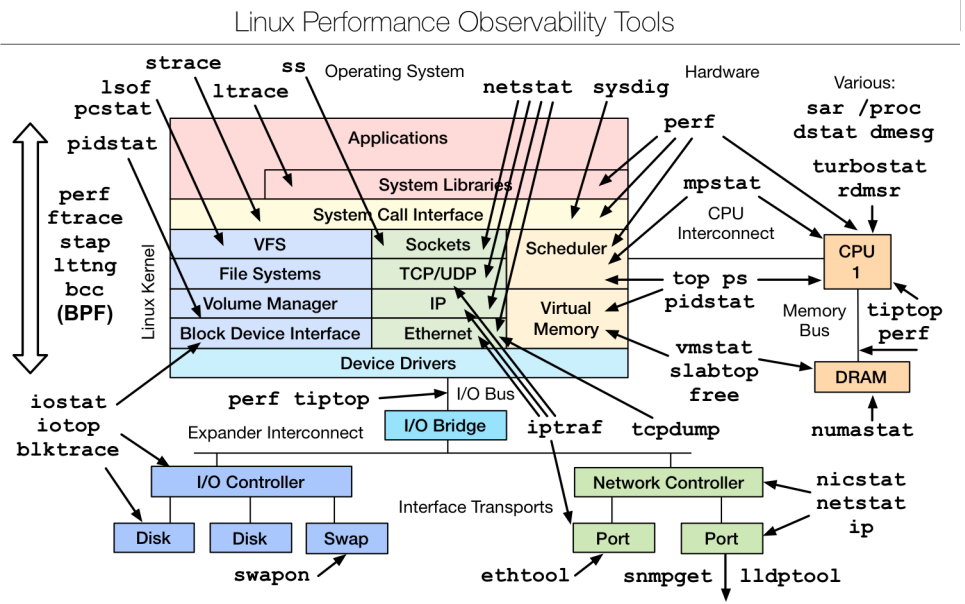
- Aim: best price/performance possible
  - Can be endless: continual improvement
- Fixes can take hours, days, weeks, months
  - Time to read docs & source code, experiment
  - Can take on large projects no single team would staff
- Usually no prior "good" state
  - No spot the difference. No starting point.
  - Is now "good" or "bad"? Experience/instinct helps
- Solo/team work

At Netflix: The Performance Engineering team, with help from developers

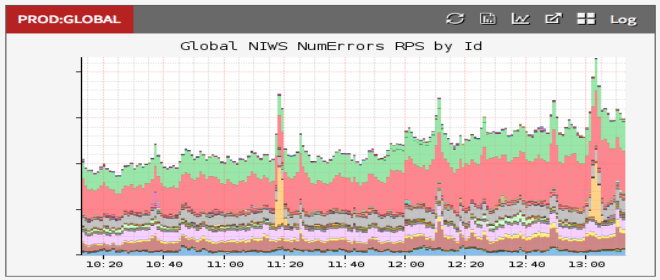


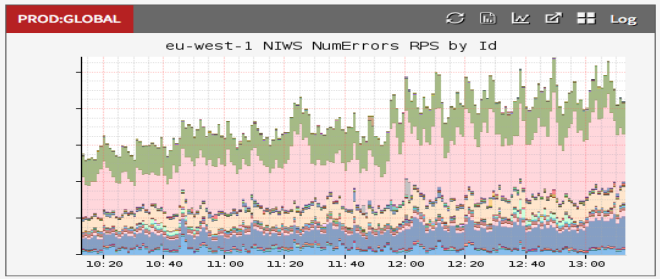
+3

Performance Engineering

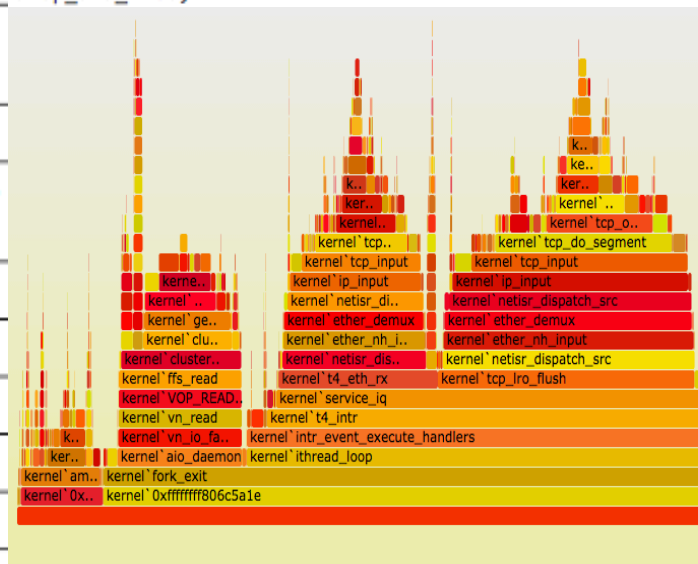



```
if (fastopen_sk) {
        af_ops->send_synack(fastopen_sk, dst, &fl, req,
                            &foc, false);
        /* Add the child socket directly into the accept queue */
        inet_csk_reqsk_queue_add(sk, req, fastopen_sk);
        sk->sk_data_ready(sk);
        bh_unlock_sock(fastopen_sk);
        sock_put(fastopen_sk);
} else {
        tcp_rsk(req)->tfo_listener = false;
        if (!want_cookie)
                inet_csk_reqsk_queue_hash_add(sk, req, TCP_TIMEOUT_INIT);
        af_ops->send_synack(sk, dst, &fl, req,
                            &foc, !want_cookie);
        if (want_cookie)
                goto drop_and_free;
```

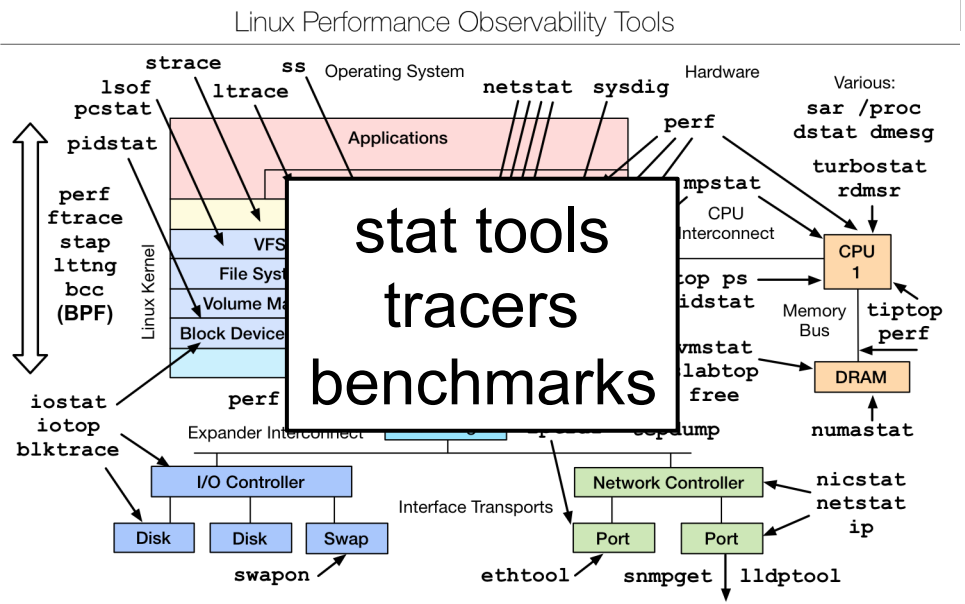




Event Num.	Event Mask Mnemonic
3CH	UnHalted Core Cycles
3CH	UnHalted Reference Cycles
C0H	Instruction Retired
2EH	LLC Reference
2EH	LLC Misses
C4H	Branch Instruction Retired
C5H	Branch Misses Retired



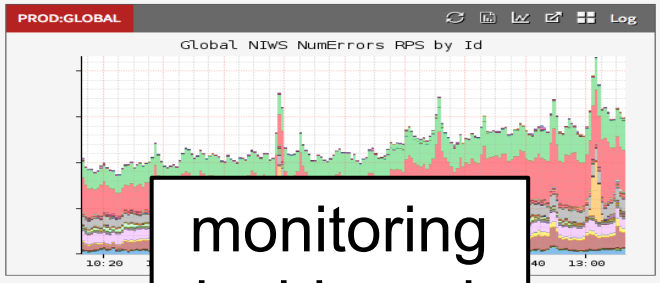
Performance Engineering

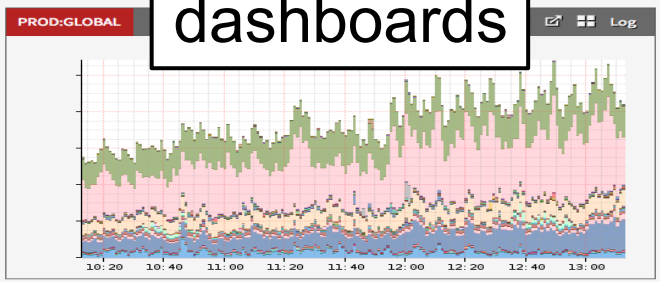


stat tools
tracers
benchmarks

```
if (fastopen_sk) {
        af_ops->send_synack(fastopen_sk, dst, &fl, req,
                            &foc, false);
        /* Add the child socket directly into the accept queue */
        inet_csk_reqs
        sk->sk_data_r
        bh_unlock_soc
        sock_put(fast
} else {
        tcp_rsk(req)-
        if (!want_coo
                inet_csk_reqsk_queue_hash_add(sk, req, TCP_TIMEOUT_INIT);
        af_ops->send_synack(sk, dst, &fl, req,
                            &foc, !want_cookie);
        if (want_cookie)
                goto drop_and_free;
```

documentation
source code
tuning

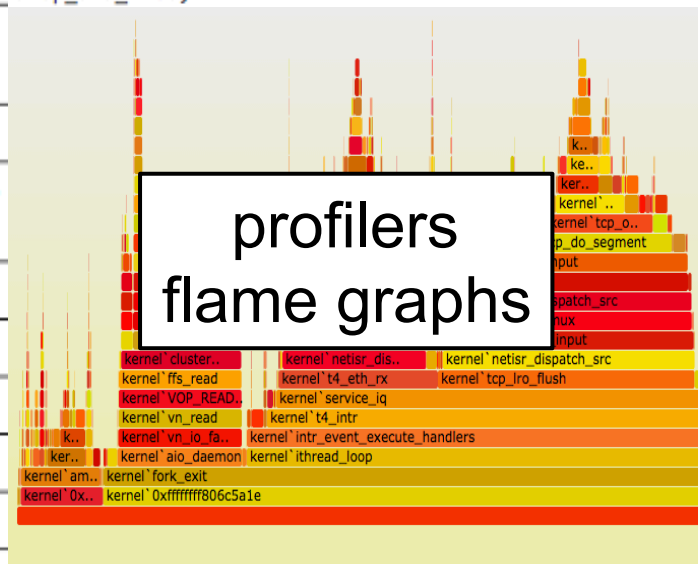




monitoring
dashboards

Event Num.	Event Mask Mnemonic
3CH	UnHalted Core Cycles
3CH	UnHalted Reference Cycles
C0H	...red
2EH	LLC Reference
2EH	LLC Misses
C4H	Branch Instruction Retired
C5H	Branch Misses Retired

PMCs

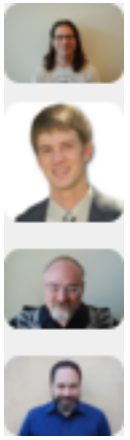


profilers
flame graphs

SRE Perf Incident Response

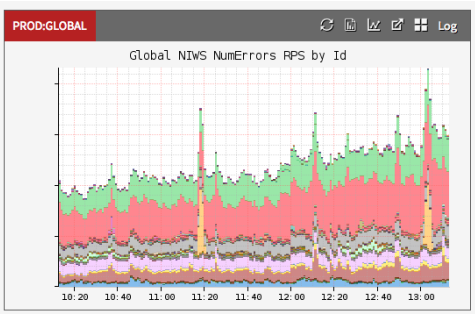
- Aim: resolve issue in minutes
  - Quick resolution is king. Can scale up, roll back, redirect traffic.
  - Must cope under pressure, and at 3am
- Previously was in a "good" state
  - Spot the difference with historical graphs
- Get immediate help from all staff
  - Must be social
- Reliability & perf issues often related

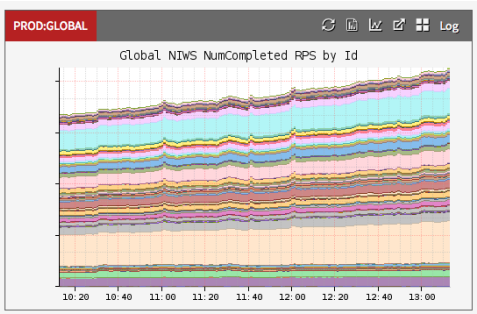
At Netflix, the Core team (5 SREs), with immediate help from developers and performance engineers

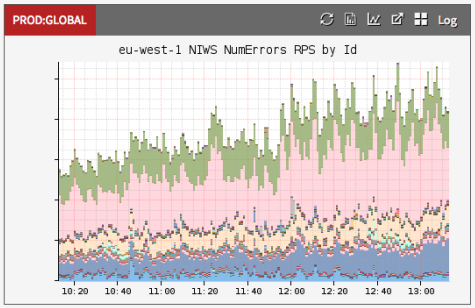


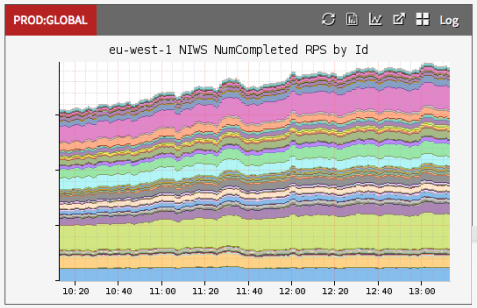
+1

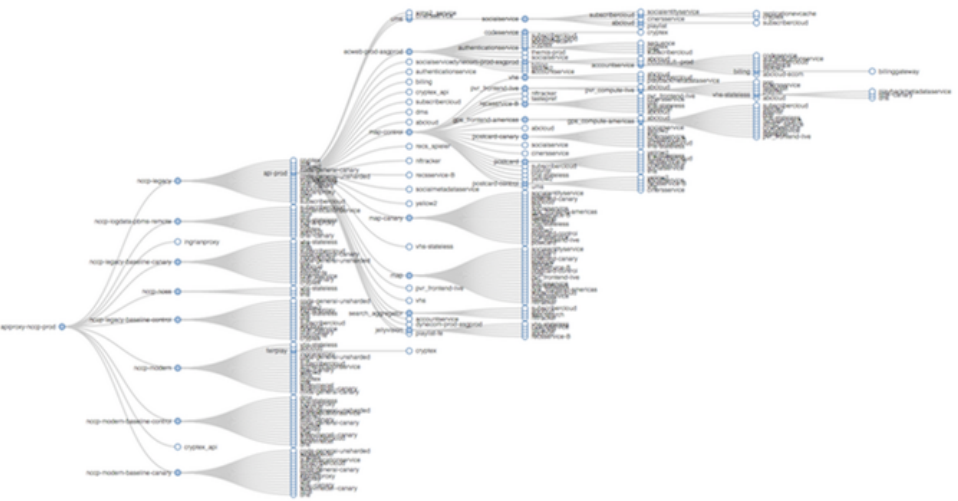
SRE Perf Incident Response

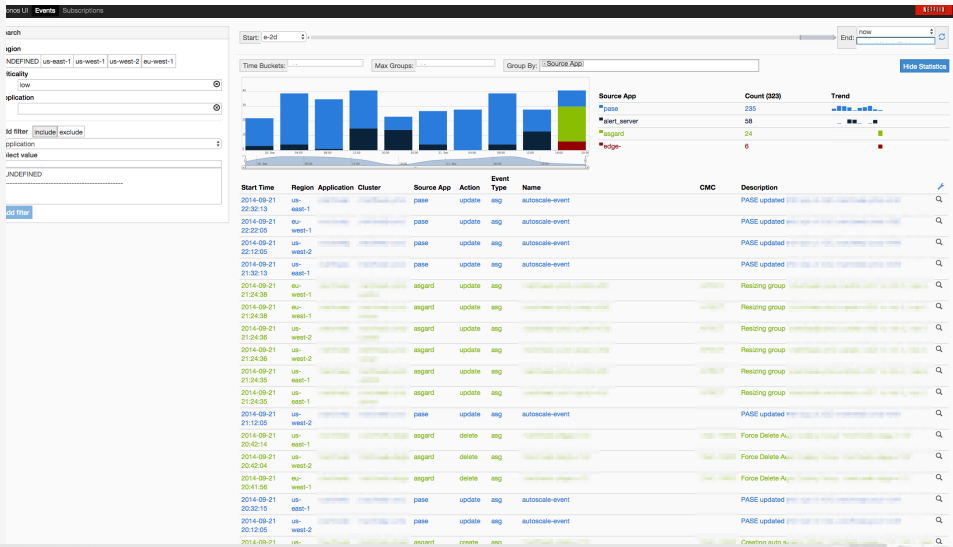


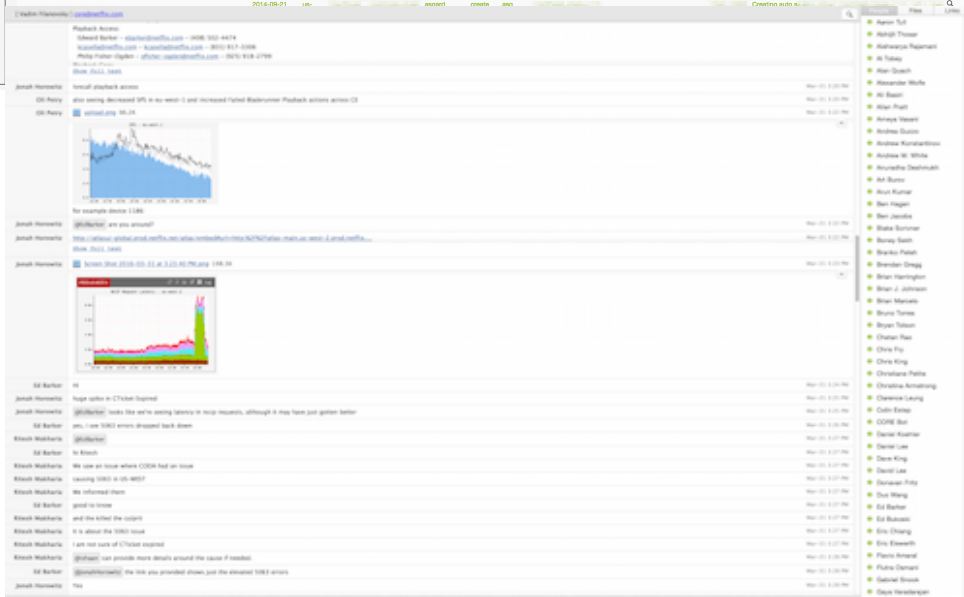




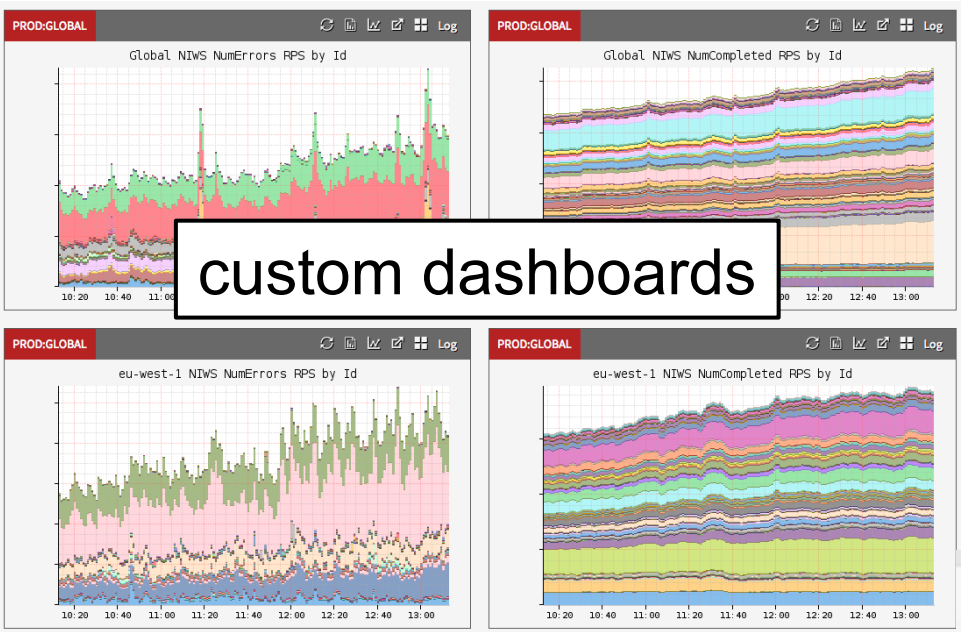




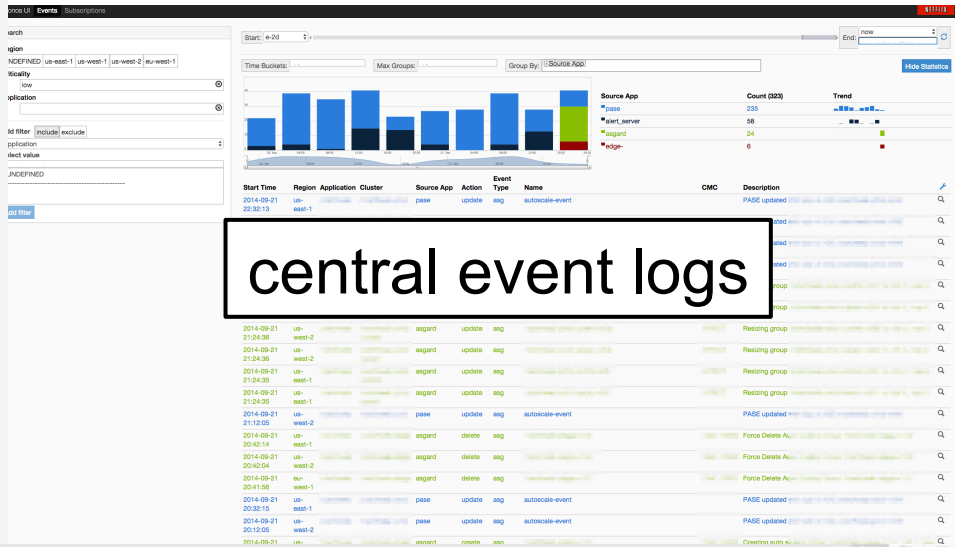




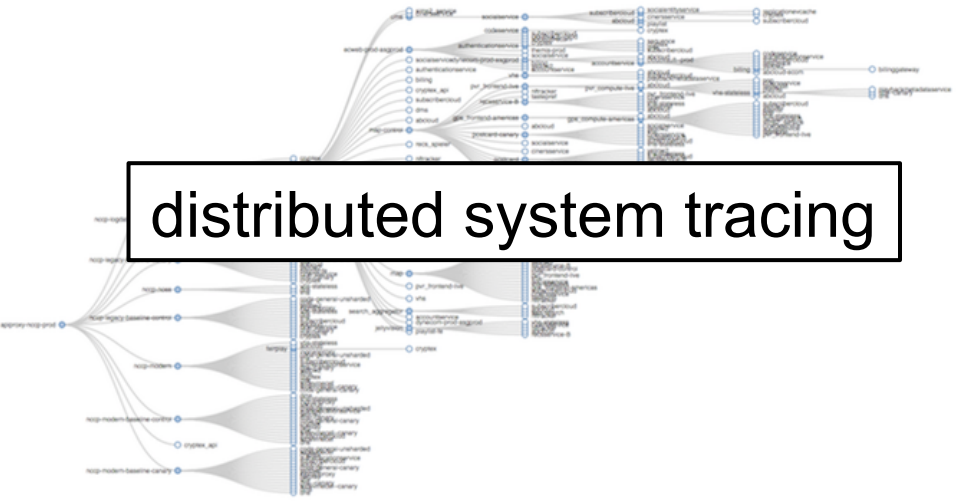
SRE Perf Incident Response



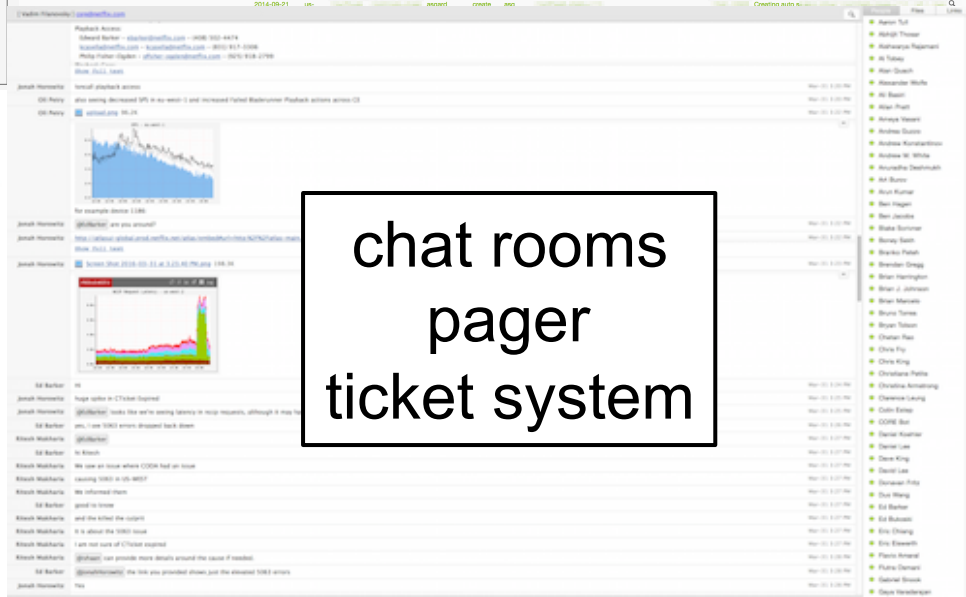
custom dashboards



central event logs



distributed system tracing

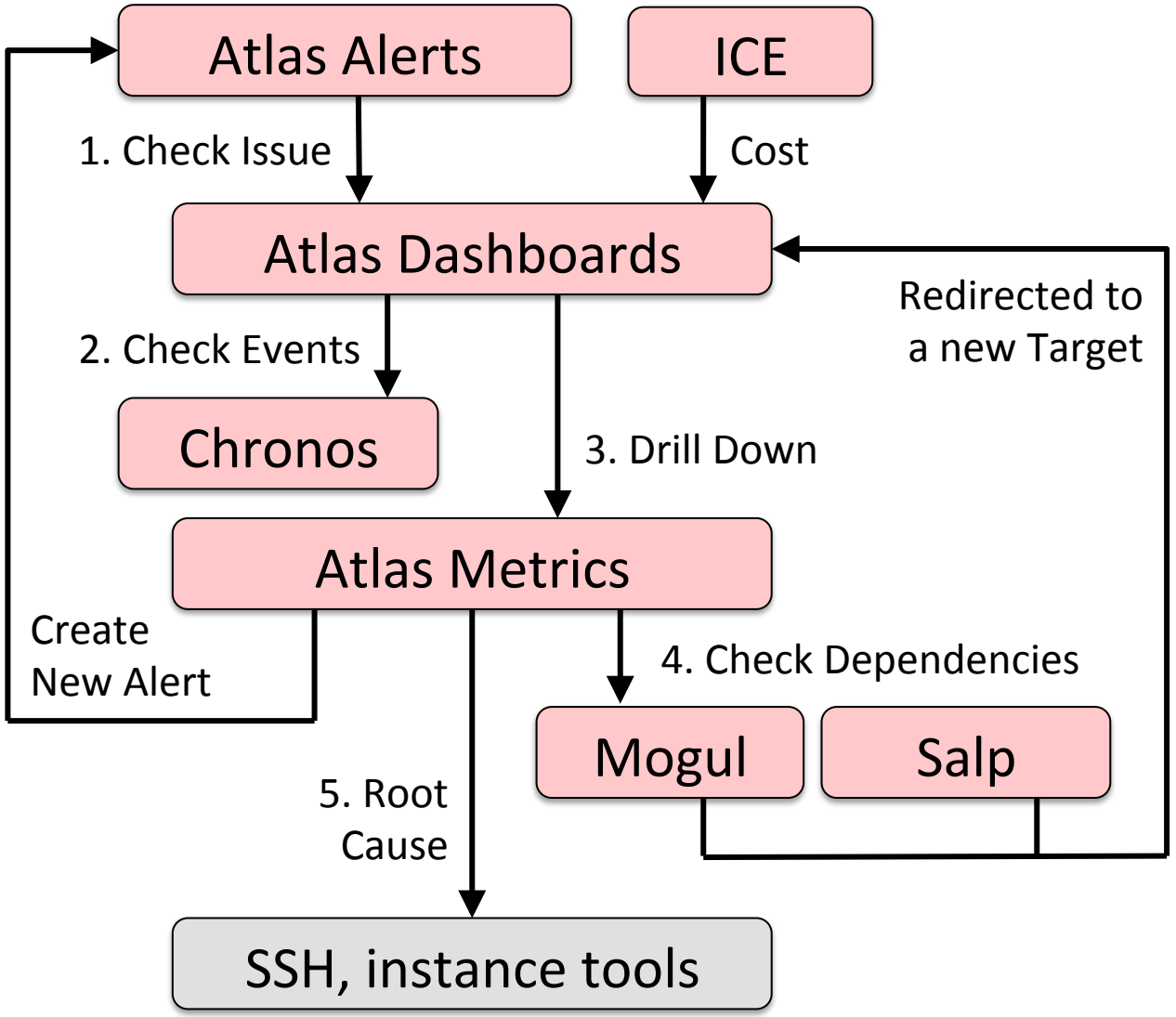


chat rooms
pager
ticket system

Netflix Cloud Analysis Process

In summary…

Example SRE response path enumerated



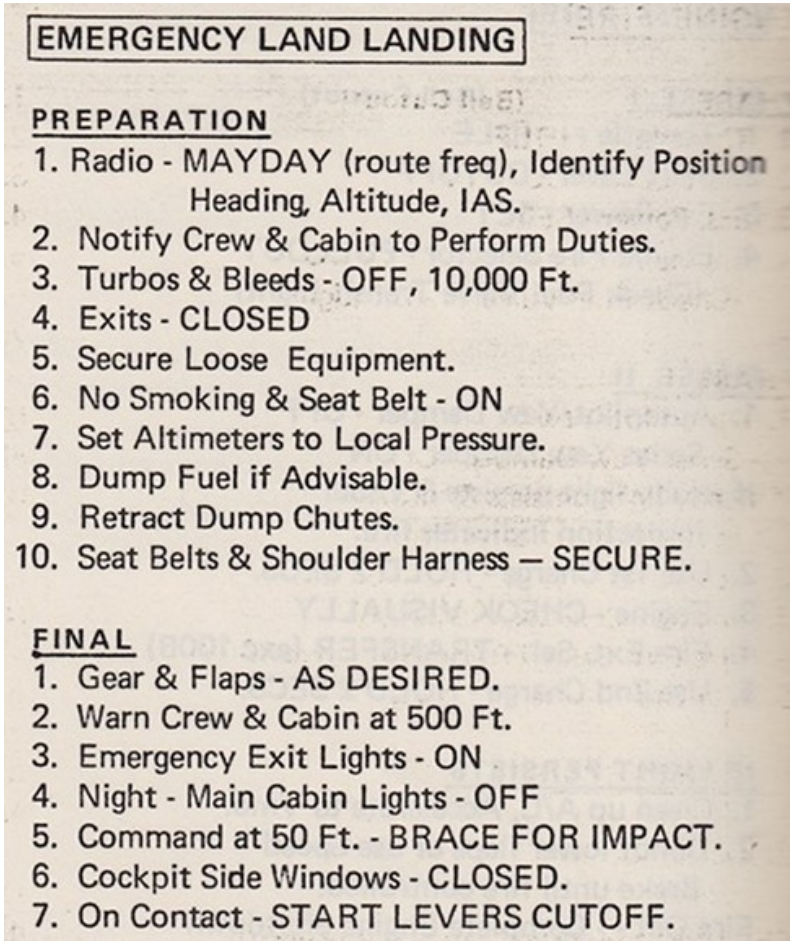
Plus some other tools not pictured

The Need for Checklists

- Speed
- Completeness
- A Starting Point
- An Ending Point
- Reliability
- Training

Perf checklists have historically been created for perf engineering (hours) not SRE response (minutes)

More on checklists: Gawande, A., *The Checklist Manifesto*. Metropolitan Books, 2008



EMERGENCY LAND LANDING

PREPARATION

1. Radio - MAYDAY (route freq), Identify Position Heading, Altitude, IAS.
2. Notify Crew & Cabin to Perform Duties.
3. Turbos & Bleeds - OFF, 10,000 Ft.
4. Exits - CLOSED
5. Secure Loose Equipment.
6. No Smoking & Seat Belt - ON
7. Set Altimeters to Local Pressure.
8. Dump Fuel if Advisable.
9. Retract Dump Chutes.
10. Seat Belts & Shoulder Harness — SECURE.

FINAL

1. Gear & Flaps - AS DESIRED.
2. Warn Crew & Cabin at 500 Ft.
3. Emergency Exit Lights - ON
4. Night - Main Cabin Lights - OFF
5. Command at 50 Ft. - BRACE FOR IMPACT.
6. Cockpit Side Windows - CLOSED.
7. On Contact - START LEVERS CUTOFF.

Boeing 707 Emergency Checklist (1969)

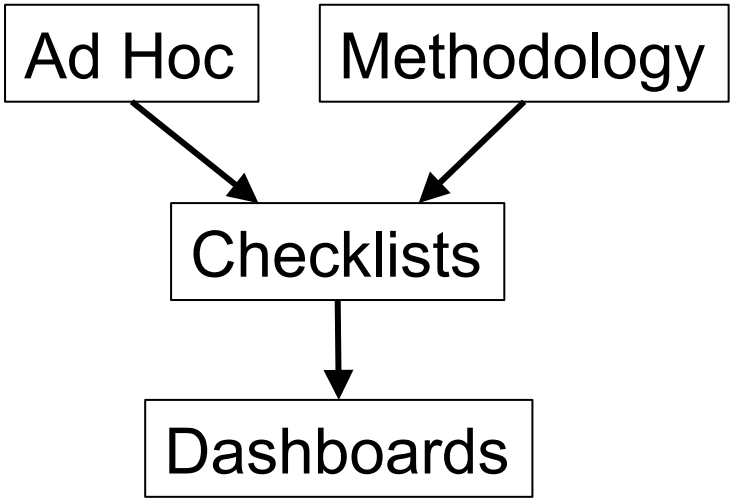
SRE Checklists at Netflix

- Some shared docs
  - PRE Triage Methodology
  - go/triage: a checklist of dashboards
- Most "checklists" are really custom dashboards
  - Selected metrics for both reliability and performance
- I maintain my own per-service and per-device checklists

SRE *Performance* Checklists

The following are:

- Cloud performance checklists/dashboards
- SSH/Linux checklists (lowest common denominator)
- Methodologies for deriving cloud/instance checklists



Including aspirational: what we want to do & build as dashboards

1. PRE Triage Checklist

Our initial checklist

Netflix specific

PRE Triage Checklist

- Performance and Reliability Engineering checklist
  - Shared doc with a hierarchal checklist with 66 steps total

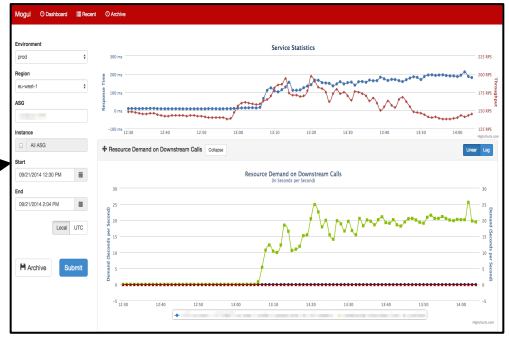
1. Initial Impact
   1. record timestamp
   2. quantify: SPS, signups, support calls
   3. check impact: regional or global?
   4. check devices: device specific?

Confirms, quantifies, & narrows problem. Helps you reason about the cause.

2. Time Correlations
   1. pretriage dashboard
      1. check for suspect NIWS client: error rates
      2. check for source of error/request rate change
      3. […dashboard specifics…]

PRE Triage Checklist. cont.

- 3. Evaluate Service Health
  - perfvitals dashboard
  - mogul dependency correlation
  - by cluster/asg/node:
    - latency: avg, 90 percentile
    - request rate
    - CPU: utilization, sys/user
    - Java heap: GC rate, leaks
    - memory
    - load average
    - thread contention (from Java)
    - JVM crashes
    - network: tput, sockets
    - […]



custom dashboards

2. predash

Initial dashboard

Netflix specific

predash

Performance and Reliability Engineering dashboard

A list of selected dashboards suited for incident response

predash

List of dashboards is its own checklist:

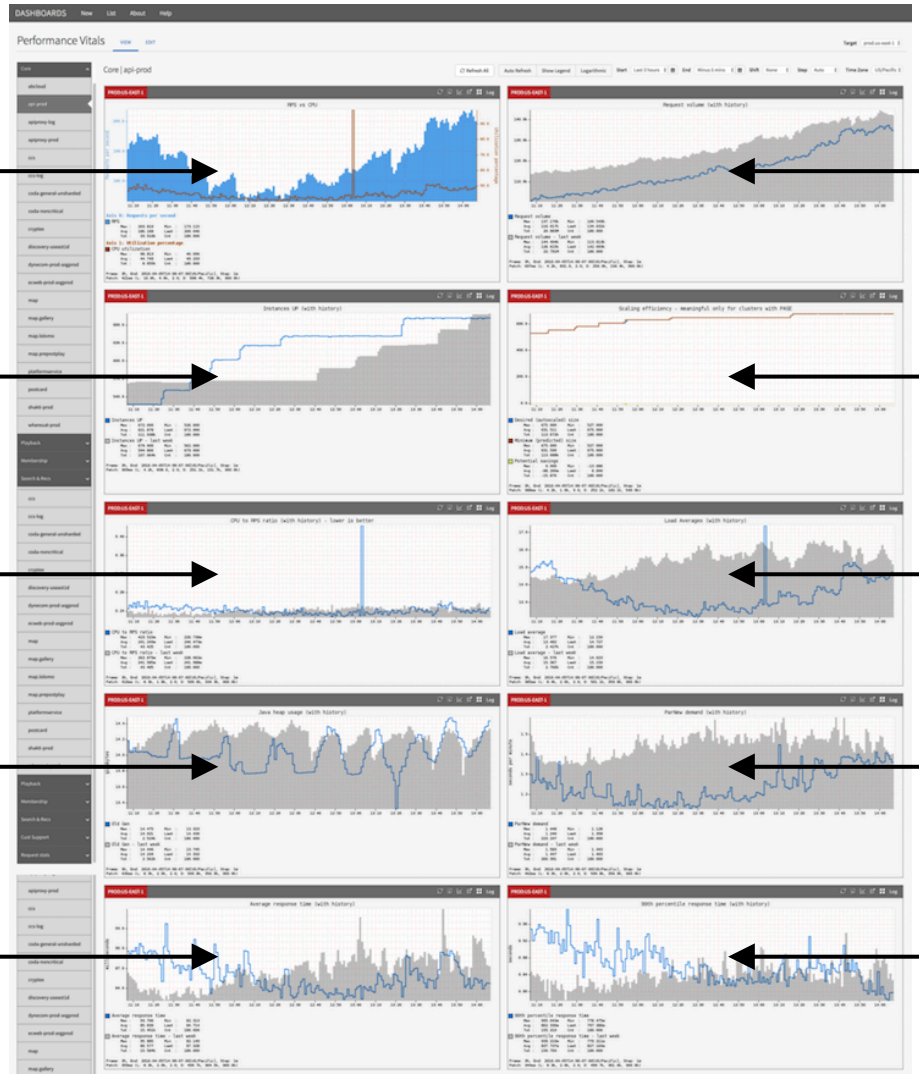
1. Overview
2. Client stats
3. Client errors & retries
4. NIWS HTTP errors
5. NIWS Errors by code
6. DRM request overview
7. DoS attack metrics
8. Push map
9. Cluster status

...

3. perfvitals

Service dashboard

perfvitals


1. RPS, CPU
2. Volume
3. Instances
4. Scaling
5. CPU/RPS
6. Load Avg
7. Java Heap
8. ParNew
9. Latency
10. 99th tile

4. Cloud Application Performance Dashboard

A generic example

Cloud App Perf Dashboard

1. Load
2. Errors
3. Latency
4. Saturation
5. Instances

Cloud App Perf Dashboard

1. Load ----------► problem of load applied? req/sec, by type
2. Errors ----------► errors, timeouts, retries
3. Latency --------► response time average, 99th -tile, distribution
4. Saturation -----► CPU load averages, queue length/time
5. Instances ------► scale up/down? count, state, version

All time series, for every application, and dependencies. Draw a functional diagram with the entire data path.

Same as Google's "Four Golden Signals" (Latency, Traffic, Errors, Saturation), with instances added due to cloud

- Beyer, B., Jones, C., Petoff, J., Murphy, N. *Site Reliability Engineering*. O'Reilly, Apr 2016

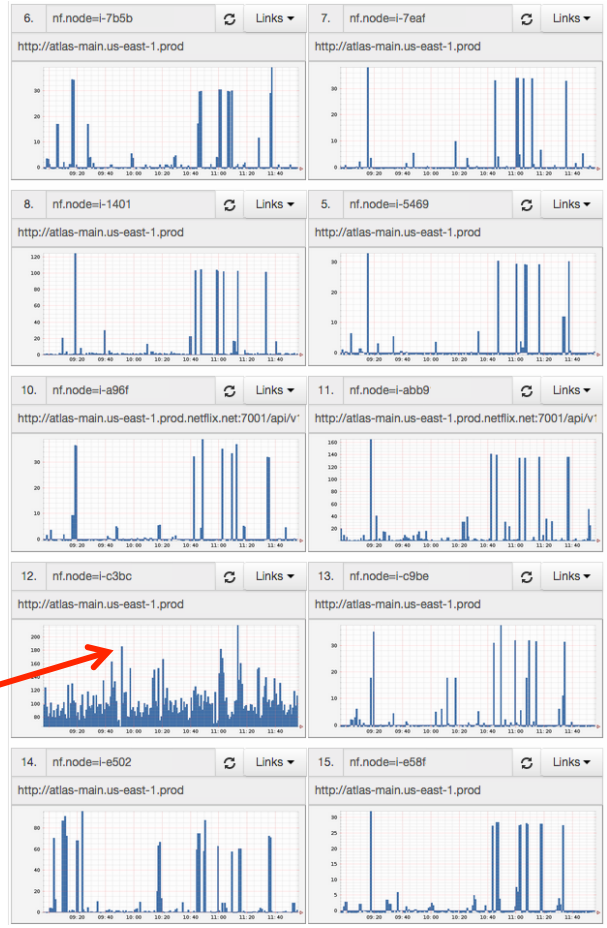
5. Bad Instance Dashboard

An *Anti*-Methodology

Bad Instance Dashboard

1. Plot request time per-instance
2. Find the bad instance
3. Terminate bad instance
4. Someone else's problem now!

In SRE incident response, if it works, do it.



Bad instance

Terminate!

95th percentile latency
(Atlas Exploder)

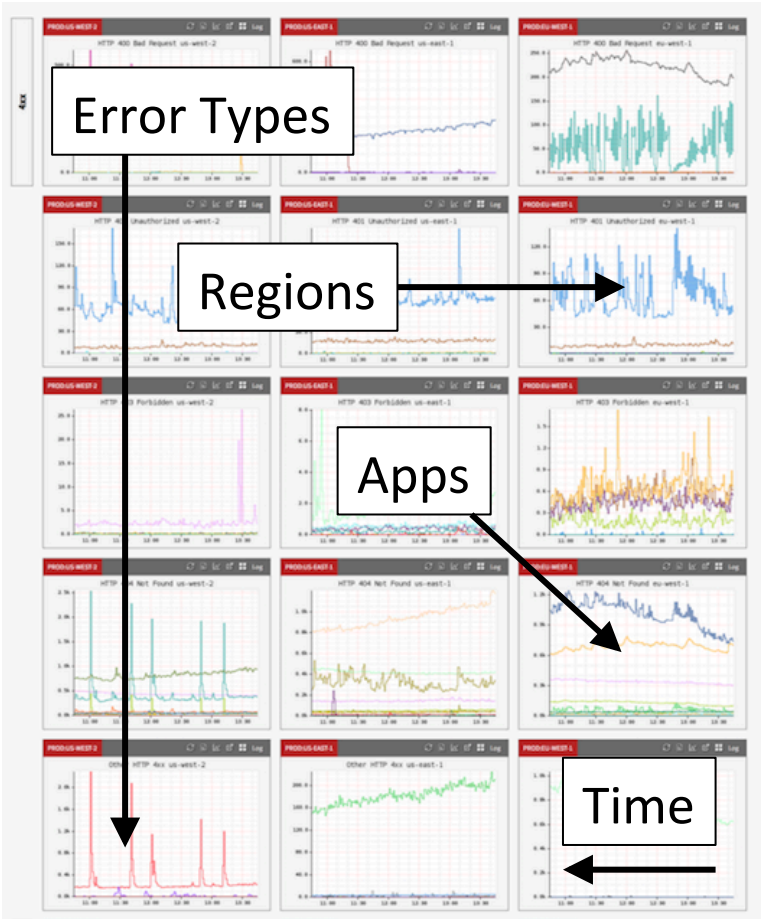
Lots More Dashboards

We have countless more, mostly app specific and reliability focused

- Most reliability incidents involve time correlation with a central log system

Sometimes, dashboards & monitoring aren't enough. Time for SSH.

NIWS HTTP errors:



6. Linux Performance Analysis in 60,000 milliseconds

Linux Perf Analysis in 60s

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`

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

60s: uptime, dmesg, vmstat

```
$ uptime
 23:51:26 up 21:31,  1 user,  load average: 30.02, 26.43, 19.02
```

```
$ dmesg | tail
[1880957.563150] perl invoked oom-killer: gfp_mask=0x280da, order=0, oom_score_adj=0
[...]
[1880957.563400] Out of memory: Kill process 18694 (perl) score 246 or sacrifice child
[1880957.563408] Killed process 18694 (perl) total-vm:1972392kB, anon-rss:1953348kB, file-rss:0kB
[2320864.954447] TCP: Possible SYN flooding on port 7001. Dropping request.  Check SNMP counters.
```

```
$ vmstat 1
procs ---------memory---------- ---swap-- -----io---- -system-- ------cpu-----
 r  b swpd   free   buff  cache   si   so    bi    bo   in   cs us sy id wa st
34  0    0 200889792  73708 591828    0    0     0     5    6   10 96  1  3  0  0
32  0    0 200889920  73708 591860    0    0     0   592 13284 4282 98  1  1  0  0
32  0    0 200890112  73708 591860    0    0     0     0 9501 2154 99  1  0  0  0
32  0    0 200889568  73712 591856    0    0     0    48 11900 2459 99  0  0  0  0
32  0    0 200890208  73712 591860    0    0     0     0 15898 4840 98  1  1  0  0
^C
```

60s: mpstat

```
$ mpstat -P ALL 1
Linux 3.13.0-49-generic (titanclusters-xxxxx)  07/14/2015  _x86_64_ (32 CPU)

07:38:49 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest  %gnice  %idle
07:38:50 PM  all   98.47    0.00    0.75    0.00    0.00    0.00    0.00    0.00    0.00   0.78
07:38:50 PM    0   96.04    0.00    2.97    0.00    0.00    0.00    0.00    0.00    0.00   0.99
07:38:50 PM    1   97.00    0.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00   2.00
07:38:50 PM    2   98.00    0.00    1.00    0.00    0.00    0.00    0.00    0.00    0.00   1.00
07:38:50 PM    3   96.97    0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00   3.03
[...]
```

60s: pidstat

```
$ pidstat 1
Linux 3.13.0-49-generic (titanclusters-xxxxx)  07/14/2015  _x86_64_  (32 CPU)

07:41:02 PM   UID       PID    %usr %system  %guest    %CPU   CPU  Command
07:41:03 PM     0         9    0.00    0.94    0.00    0.94     1  rcuos/0
07:41:03 PM     0      4214    5.66    5.66    0.00   11.32    15  mesos-slave
07:41:03 PM     0      4354    0.94    0.94    0.00    1.89     8  java
07:41:03 PM     0      6521 1596.23    1.89    0.00 1598.11    27  java
07:41:03 PM     0      6564 1571.70    7.55    0.00 1579.25    28  java
07:41:03 PM 60004     60154    0.94    4.72    0.00    5.66     9  pidstat

07:41:03 PM   UID       PID    %usr %system  %guest    %CPU   CPU  Command
07:41:04 PM     0      4214    6.00    2.00    0.00    8.00    15  mesos-slave
07:41:04 PM     0      6521 1590.00    1.00    0.00 1591.00    27  java
07:41:04 PM     0      6564 1573.00   10.00    0.00 1583.00    28  java
07:41:04 PM   108      6718    1.00    0.00    0.00    1.00     0  snmp-pass
07:41:04 PM 60004     60154    1.00    4.00    0.00    5.00     9  pidstat
^C
```

60s: iostat

```
$ iostat -xmdz 1

Linux 3.13.0-29 (db001-eb883efa)   08/18/2014   _x86_64_   (16 CPU)

Device:         rrqm/s   wrqm/s     r/s     w/s    rMB/s    wMB/s \ ...
xvda              0.00     0.00    0.00    0.00     0.00     0.00 / ...
xvdb            213.00     0.00 15299.00    0.00   338.17     0.00 \ ...
xvdc            129.00     0.00 15271.00    3.00   336.65     0.01 / ...
md0               0.00     0.00 31082.00    3.00   678.45     0.01 \ ...
```

Workload →

```
... \ avgqu-sz   await r_await w_await  svctm  %util
... /    0.00    0.00    0.00    0.00   0.00   0.00
... \  126.09    8.22    8.22    0.00   0.06  86.40
... /   99.31    6.47    6.47    0.00   0.06  86.00
... \    0.00    0.00    0.00    0.00   0.00   0.00
```

Resulting Performance →

60s: free, sar –n DEV

```
$ free -m
             total       used       free     shared    buffers     cached
Mem:        245998      24545     221453         83         59        541
-/+ buffers/cache:      23944     222053
Swap:            0          0          0
```

```
$ sar -n DEV 1
Linux 3.13.0-49-generic (titanclusters-xxxxx)  07/14/2015     _x86_64_    (32 CPU)

12:16:48 AM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s  %ifutil
12:16:49 AM      eth0  18763.00   5032.00  20686.42    478.30      0.00      0.00      0.00     0.00
12:16:49 AM        lo     14.00     14.00      1.36      1.36      0.00      0.00      0.00     0.00
12:16:49 AM   docker0      0.00      0.00      0.00      0.00      0.00      0.00      0.00     0.00

12:16:49 AM     IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s   rxcmp/s   txcmp/s  rxmcst/s  %ifutil
12:16:50 AM      eth0  19763.00   5101.00  21999.10    482.56      0.00      0.00      0.00     0.00
12:16:50 AM        lo     20.00     20.00      3.25      3.25      0.00      0.00      0.00     0.00
12:16:50 AM   docker0      0.00      0.00      0.00      0.00      0.00      0.00      0.00     0.00
^C
```

60s: sar –n TCP,ETCP

```
$ sar -n TCP,ETCP 1
Linux 3.13.0-49-generic (titanclusters-xxxxx)  07/14/2015    _x86_64_
(32 CPU)

12:17:19 AM  active/s passive/s    iseg/s    oseg/s
12:17:20 AM      1.00      0.00  10233.00  18846.00

12:17:19 AM  atmptf/s  estres/s retrans/s isegerr/s   orsts/s
12:17:20 AM      0.00      0.00      0.00      0.00      0.00

12:17:20 AM  active/s passive/s    iseg/s    oseg/s
12:17:21 AM      1.00      0.00   8359.00   6039.00

12:17:20 AM  atmptf/s  estres/s retrans/s isegerr/s   orsts/s
12:17:21 AM      0.00      0.00      0.00      0.00      0.00
^C
```

60s: top

```
$ top
top - 00:15:40 up 21:56,  1 user,  load average: 31.09, 29.87, 29.92
Tasks: 871 total,   1 running, 868 sleeping,   0 stopped,   2 zombie
%Cpu(s): 96.8 us,  0.4 sy,  0.0 ni,  2.7 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem:  25190241+total, 24921688 used, 22698073+free,    60448 buffers
KiB Swap:        0 total,        0 used,        0 free.   554208 cached Mem

   PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
 20248 root      20   0  0.227t 0.012t  18748 S  3090  5.2  29812:58 java
  4213 root      20   0 2722544  64640  44232 S  23.5  0.0 233:35.37 mesos-slave
 66128 titancl+  20   0   24344   2332   1172 R   1.0  0.0   0:00.07 top
  5235 root      20   0 38.227g 547004  49996 S   0.7  0.2   2:02.74 java
  4299 root      20   0 20.015g 2.682g  16836 S   0.3  1.1  33:14.42 java
     1 root      20   0   33620   2920   1496 S   0.0  0.0   0:03.82 init
     2 root      20   0       0      0      0 S   0.0  0.0   0:00.02 kthreadd
     3 root      20   0       0      0      0 S   0.0  0.0   0:05.35 ksoftirqd/0
     5 root       0 -20       0      0      0 S   0.0  0.0   0:00.00 kworker/0:0H
     6 root      20   0       0      0      0 S   0.0  0.0   0:06.94 kworker/u256:0
     8 root      20   0       0      0      0 S   0.0  0.0   2:38.05 rcu_sched
```

Other Analysis in 60s

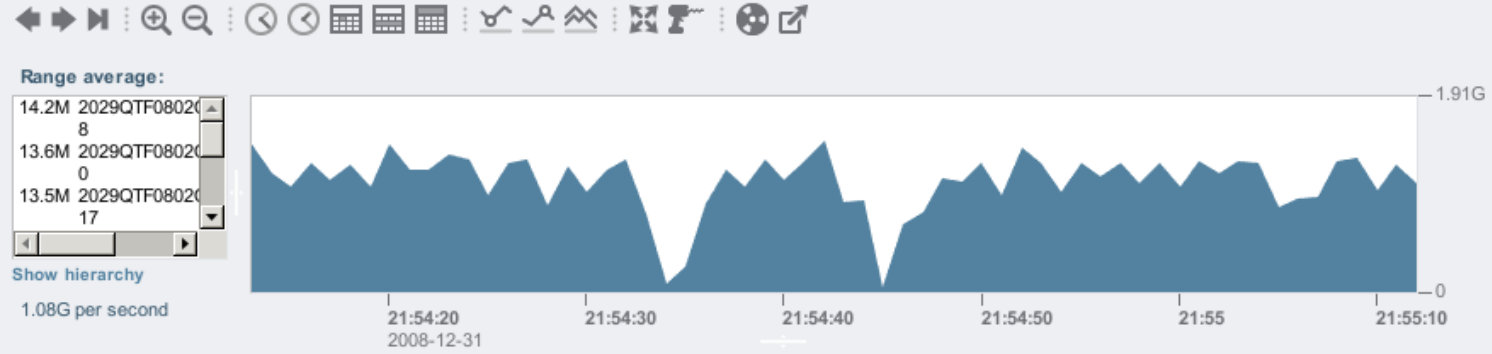
- We need such checklists for:
  - Java
  - Cassandra
  - MySQL
  - Nginx
  - etc…
- Can follow a methodology:
  - Process of elimination
  - Workload characterization
  - Differential diagnosis
  - Some summaries: http://www.brendangregg.com/methodology.html
- Turn checklists into dashboards (many do exist)

7. Linux Disk Checklist

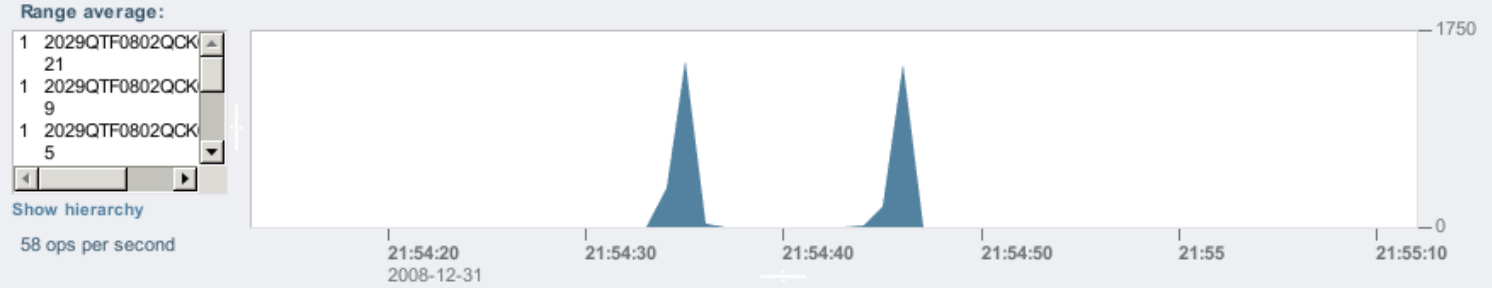
Worksheets ▸ Unusual disk latency New Save Clone Close

Add statistic...

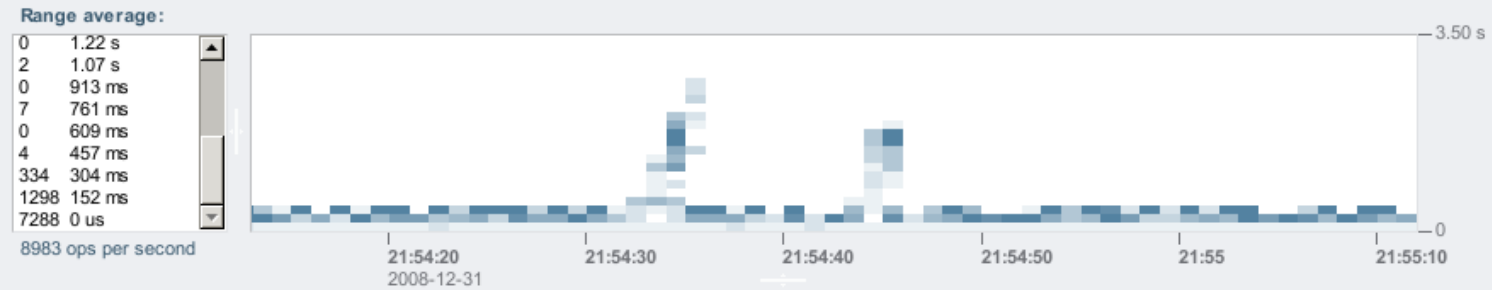
Disk: I/O bytes per second broken down by disk



Disk: I/O operations per second taking at least 521740 microseconds broken down by disk



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



Linux Disk Checklist

1. `iostat –xnz 1`
2. `vmstat 1`
3. `df -h`
4. `ext4slower 10`
5. `bioslower 10`
6. `ext4dist 1`
7. `biolatency 1`
8. `cat /sys/devices/…/ioerr_cnt`
9. `smartctl -l error /dev/sda1`

Linux Disk Checklist

1. `iostat –xnz 1` ----► any disk I/O? if not, stop looking
2. `vmstat 1` ----------► is this swapping? or, high sys time?
3. `df -h` --------------► are file systems nearly full?
4. `ext4slower 10` ----► (zfs*, xfs*, etc.) slow file system I/O?
5. `bioslower 10` -----► if so, check disks
6. `ext4dist 1` --------► check distribution and rate
7. `biolatency 1` -----► if interesting, check disks
8. `cat /sys/devices/…/ioerr_cnt` --► (if available) errors
9. `smartctl -l error /dev/sda1` ---► (if available) errors

Another short checklist. Won't solve everything. FS focused.
ext4slower/dist, bioslower, are from bcc/BPF tools.

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
06:49:17 cksum          3616   R 15008   0         19.31 acpi_listen
[…]
```

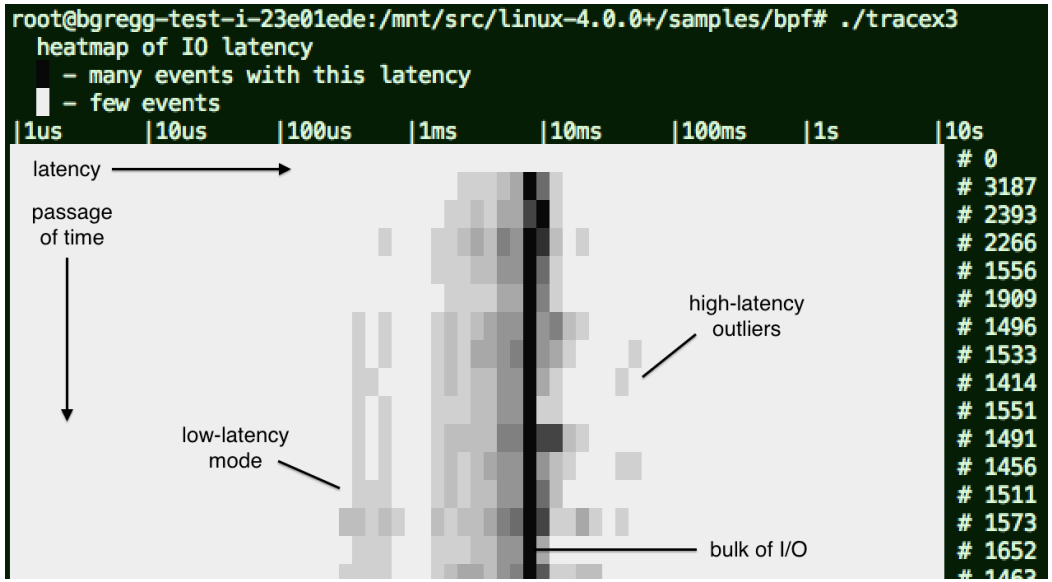
- Better indicator of application pain than disk I/O
- Measures & filters in-kernel for efficiency using BPF
  - From https://github.com/iovisor/bcc

BPF is coming…

Free your mind

BPF

- That file system checklist should be a dashboard:
  - FS & disk latency histograms, heatmaps, IOPS, outlier log
- Now possible with enhanced BPF (Berkeley Packet Filter)
  - Built into Linux 4.x: dynamic tracing, filters, histograms



System dashboards of 2017+ should look very different

8. Linux Network Checklist

Linux Network Checklist

1. `sar -n DEV,EDEV 1`
2. `sar -n TCP,ETCP 1`
3. `cat /etc/resolv.conf`
4. `mpstat -P ALL 1`
5. `tcpretrans`
6. `tcpconnect`
7. `tcpaccept`
8. `netstat -rnv`
9. check firewall config
10. `netstat -s`

Linux Network Checklist

1. `sar -n DEV,EDEV 1` --► at interface limits? or use nicstat
2. `sar -n TCP,ETCP 1` --► active/passive load, retransmit rate
3. `cat /etc/resolv.conf` -► it's always DNS
4. `mpstat -P ALL 1` --► high kernel time? single hot CPU?
5. `tcpretrans` ---------► what are the retransmits? state?
6. `tcpconnect` ----------► connecting to anything unexpected?
7. `tcpaccept` ----------► unexpected workload?
8. `netstat -rnv` ------► any inefficient routes?
9. check firewall config --► anything blocking/throttling?
10. `netstat -s` ---------► play 252 metric pickup

tcp*, are from bcc/BPF tools

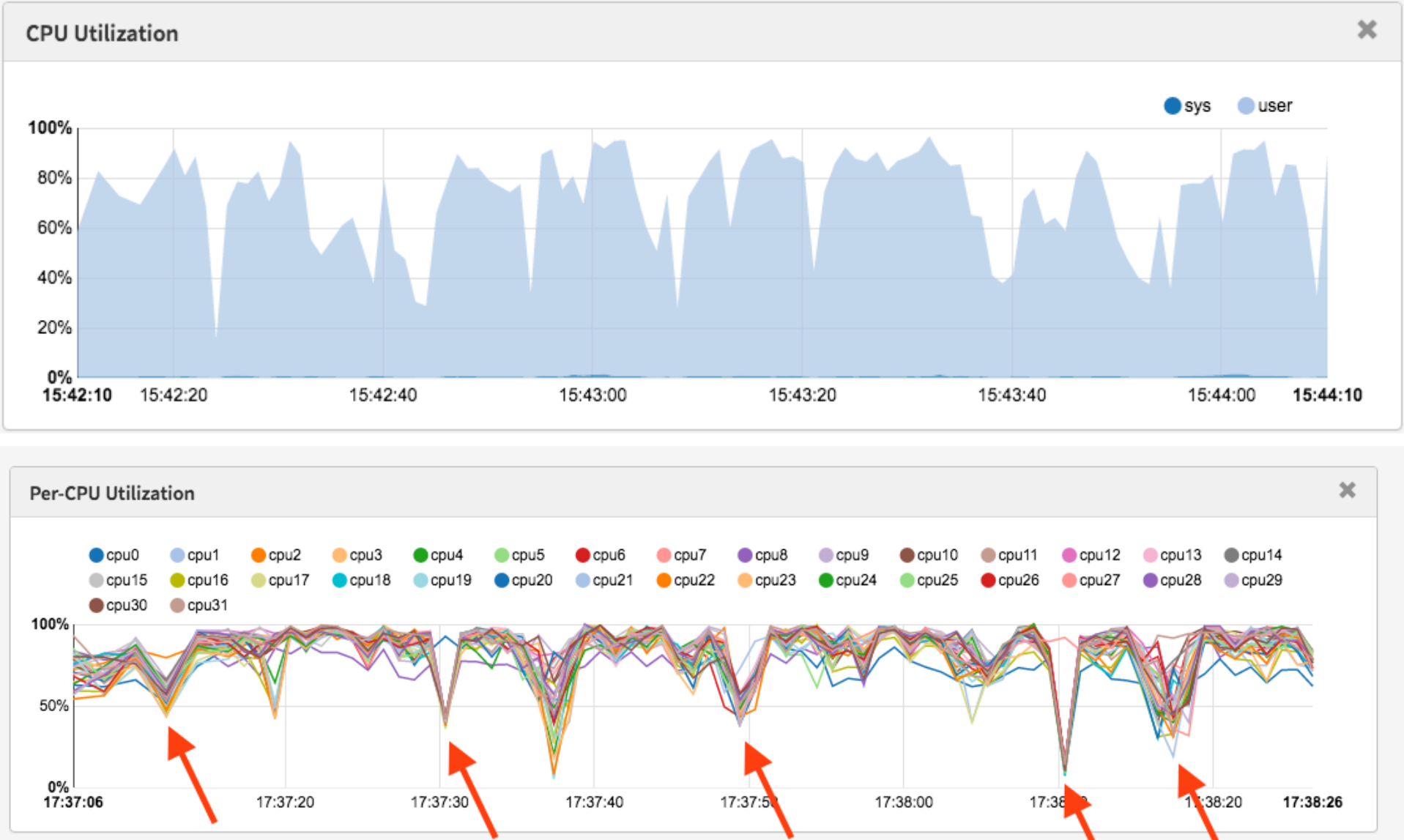
tcpretrans

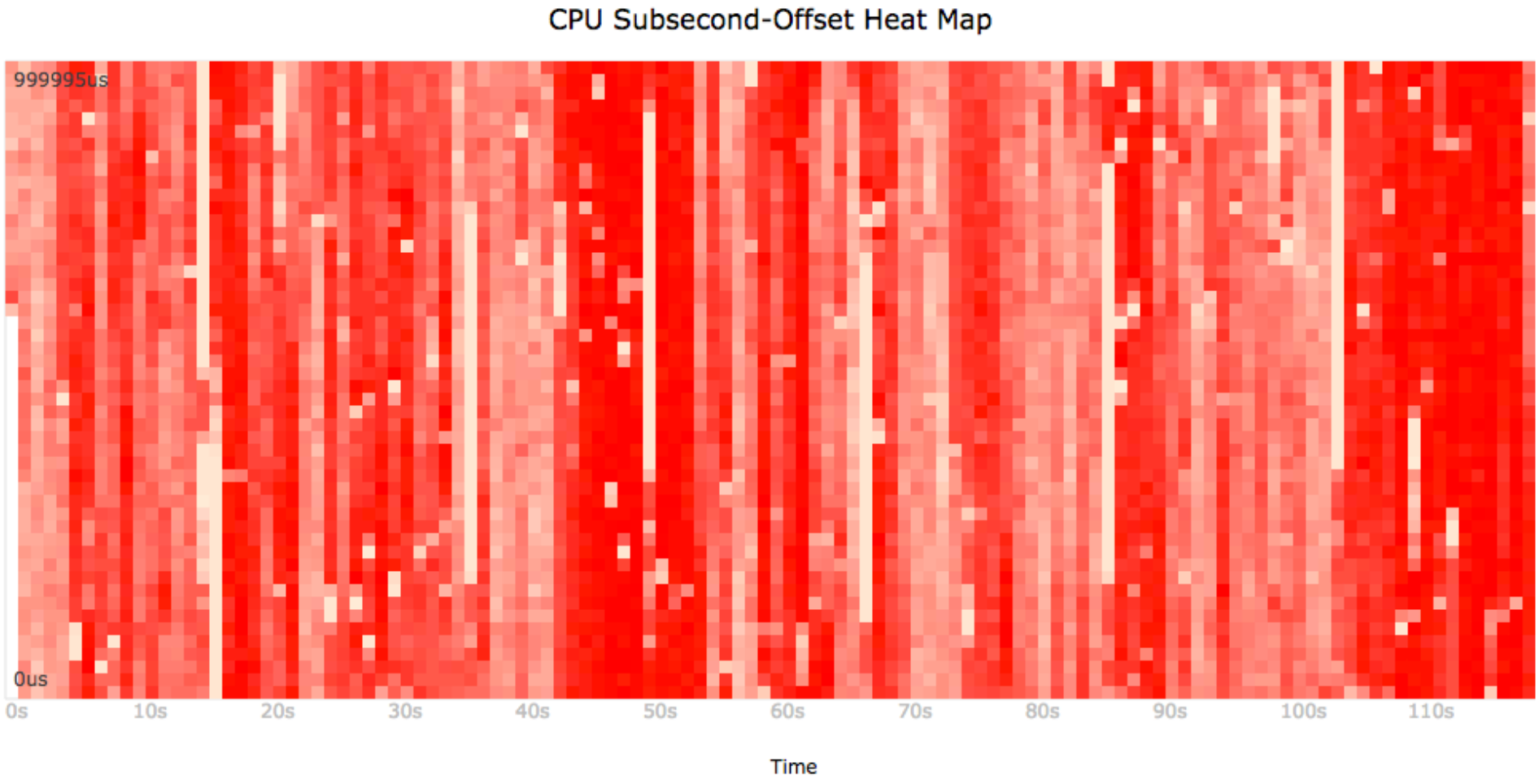
- Just trace kernel TCP retransmit functions for efficiency:

```
# ./tcpretrans
TIME     PID    IP LADDR:LPORT          T> RADDR:RPORT          STATE
01:55:05 0      4  10.153.223.157:22    R> 69.53.245.40:34619   ESTABLISHED
01:55:05 0      4  10.153.223.157:22    R> 69.53.245.40:34619   ESTABLISHED
01:55:17 0      4  10.153.223.157:22    R> 69.53.245.40:22957   ESTABLISHED
[…]
```

- From either bcc (BPF) or perf-tools (ftrace, older kernels)

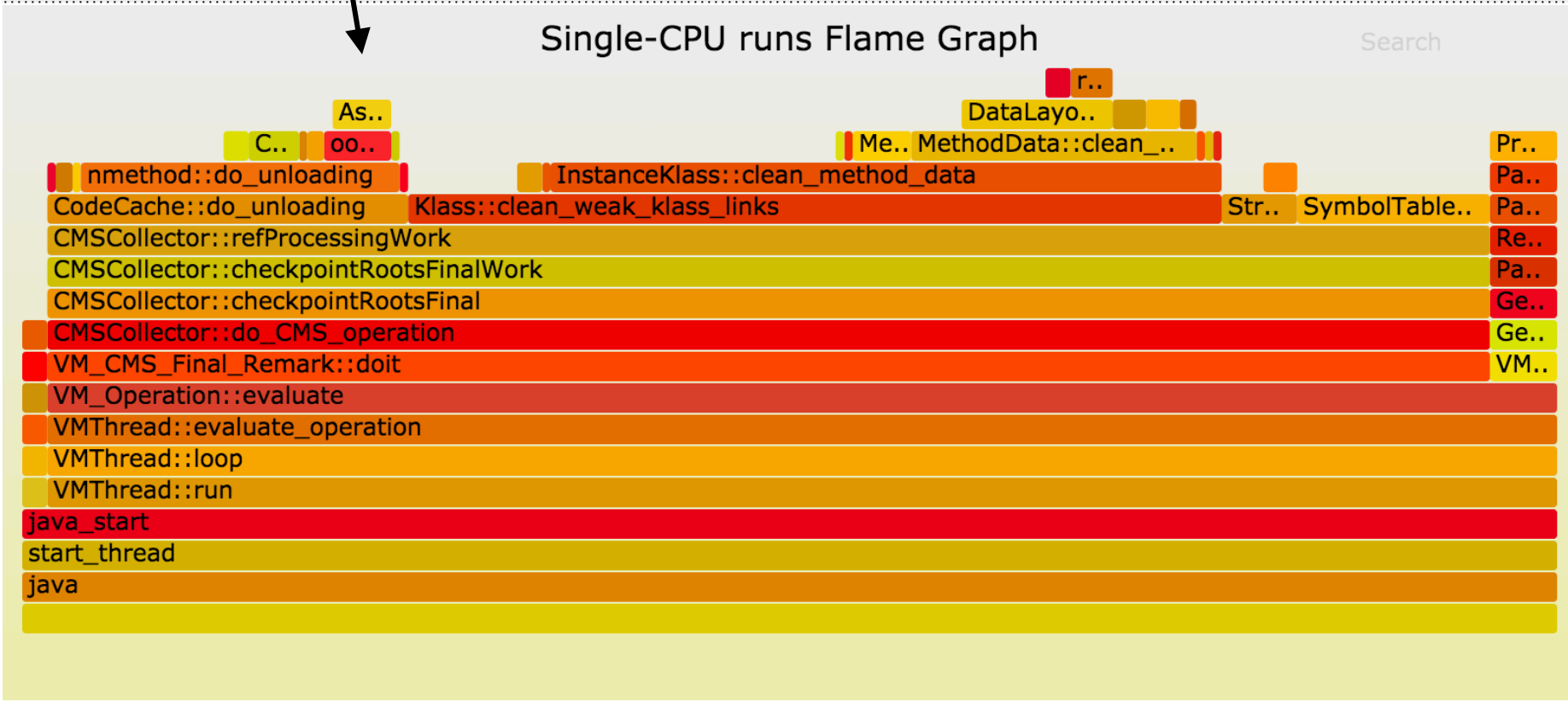
9. Linux CPU Checklist


CPU Utilization
sys
user
100%
80%
60%
40%
20%
0%
15:42:10
15:42:20
15:42:40
15:43:00
15:43:20
15:43:40
15:44:00
15:44:10
Per-CPU Utilization
cpu0 cpu1 cpu2 cpu3 cpu4 cpu5 cpu6 cpu7 cpu8 cpu9 cpu10 cpu11 cpu12 cpu13 cpu14
cpu15 cpu16 cpu17 cpu18 cpu19 cpu20 cpu21 cpu22 cpu23 cpu24 cpu25 cpu26 cpu27 cpu28 cpu29
cpu30 cpu31
100%
50%
0%
17:37:06
17:37:20
17:37:30
17:37:40
17:38:00
17:38:26
Version: v1.0.2-45-g0ec9197
(too many lines – should be a utilization heat map)



http://www.brendangregg.com/HeatMaps/subsecondoffset.html

```
$ perf script
[…]
   java 14327 [022] 252764.179741: cycles:      7f36570a4932 SpinPause (/usr/lib/jvm/java-8
   java 14315 [014] 252764.183517: cycles:      7f36570a4932 SpinPause (/usr/lib/jvm/java-8
   java 14310 [012] 252764.185317: cycles:      7f36570a4932 SpinPause (/usr/lib/jvm/java-8
   java 14332 [015] 252764.188720: cycles:      7f3658078350 pthread_cond_wait@@GLIBC_2.3.2
   java 14341 [019] 252764.191307: cycles:      7f3656d150c8 ClassLoaderDataGraph::do_unloa
   java 14341 [019] 252764.198825: cycles:      7f3656d140b8 ClassLoaderData::free_dealloca
   java 14341 [019] 252764.207057: cycles:      7f3657192400 nmethod::do_unloading(BoolObje
   java 14341 [019] 252764.215962: cycles:      7f3656ba807e Assembler::locate_operand(unsi
   java 14341 [019] 252764.225141: cycles:      7f36571922e8 nmethod::do_unloading(BoolObje
   java 14341 [019] 252764.234578: cycles:      7f3656ec4960 CodeHeap::block_start(void*) c
[…]
```



Linux CPU Checklist

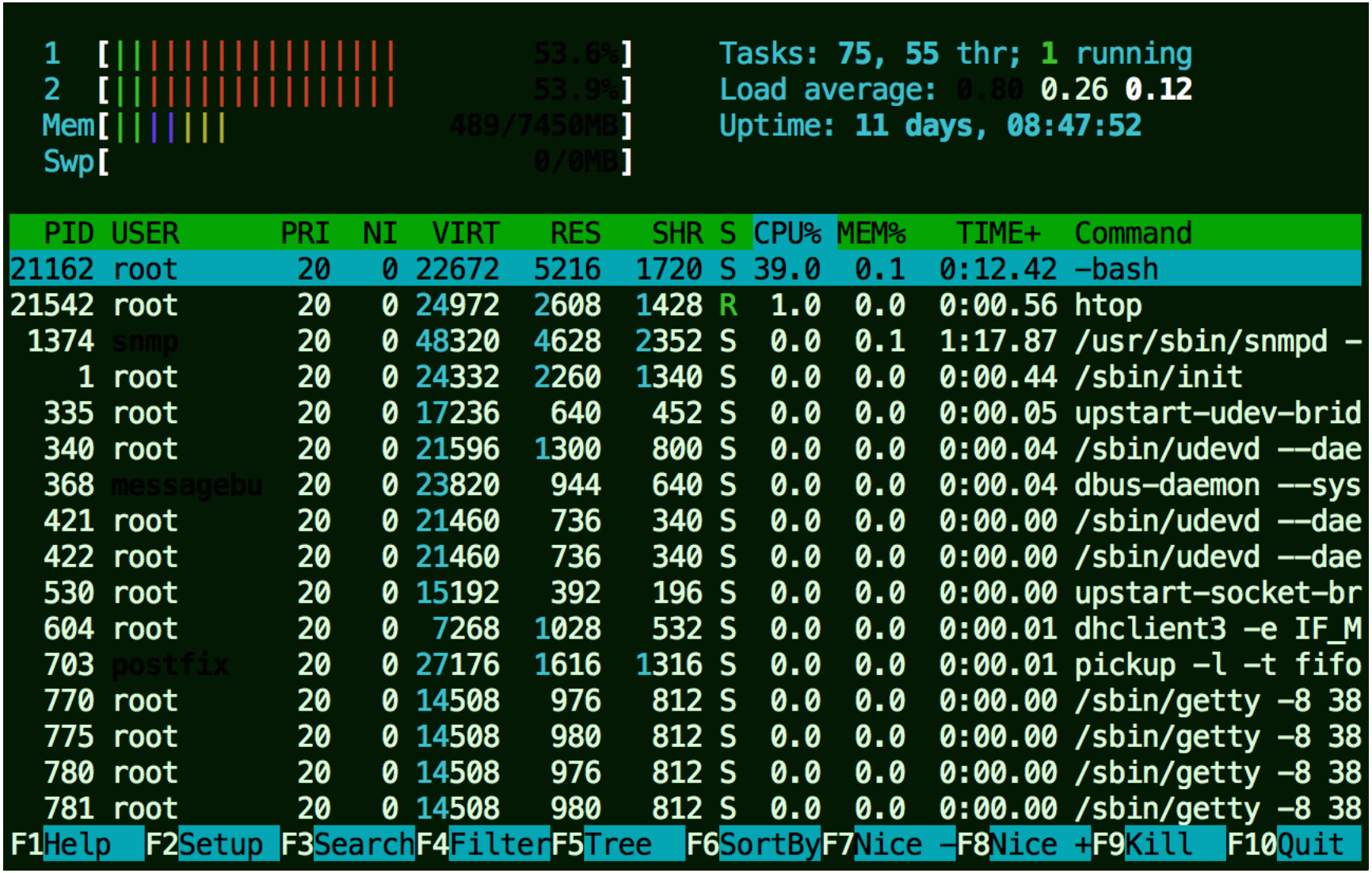
1. `uptime`
2. `vmstat 1`
3. `mpstat -P ALL 1`
4. `pidstat 1`
5. CPU flame graph
6. CPU subsecond offset heat map
7. `perf stat -a -- sleep 10`

Linux CPU Checklist

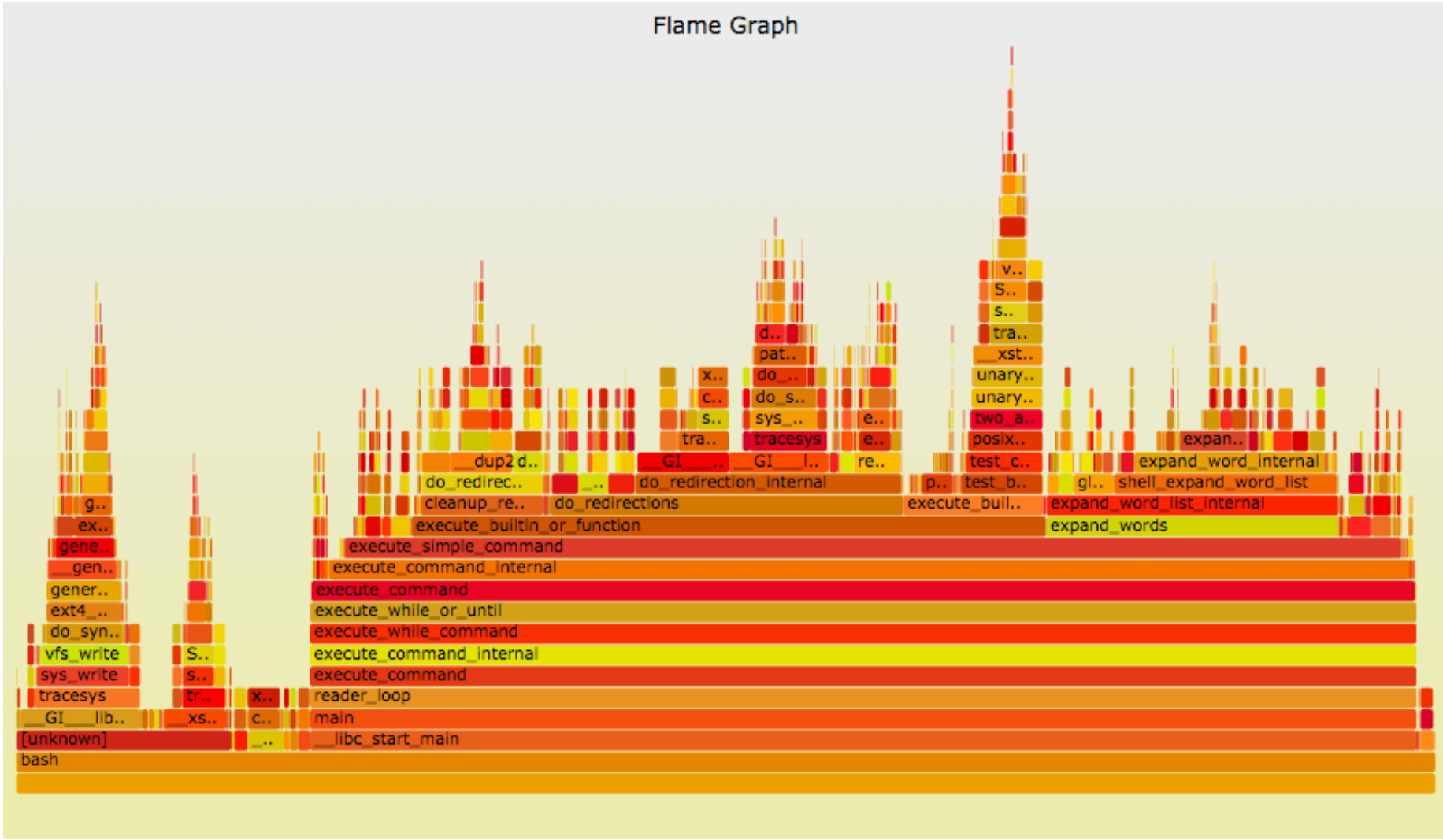
1. `uptime` ----------► load averages
2. `vmstat 1` ----------► system-wide utilization, run q length
3. `mpstat -P ALL 1` --► CPU balance
4. `pidstat 1` ----------► per-process CPU
5. CPU flame graph -----► CPU profiling
6. CPU subsecond offset heat map --► look for gaps
7. `perf stat -a -- sleep 10` -----► IPC, LLC hit ratio

htop can do 1-4

htop


1 [53.6%]
2 [53.9%]
Mem[489/7450MB]
Swp[0/0MB]
Tasks: 75, 55 thr; 1 running
Load average: 0.80 0.26 0.12
Uptime: 11 days, 08:47:52
PID USER PRI NI VIRT RES SHR S CPU% MEM% TIME+ Command
21162 root 20 0 22672 5216 1720 S 39.0 0.1 0:12.42 -bash
21542 root 20 0 24972 2608 1428 R 1.0 0.0 0:00.56 htop
1374 snmp 20 0 48320 4628 2352 S 0.0 0.1 1:17.87 /usr/sbin/snmpd -
1 root 20 0 24332 2260 1340 S 0.0 0.0 0:00.44 /sbin/init
335 root 20 0 17236 640 452 S 0.0 0.0 0:00.05 upstart-udev-brid
340 root 20 0 21596 1300 800 S 0.0 0.0 0:00.04 /sbin/udevd --dae
368 messagebu 20 0 23820 944 640 S 0.0 0.0 0:00.04 dbus-daemon --sys
421 root 20 0 21460 736 340 S 0.0 0.0 0:00.00 /sbin/udevd --dae
422 root 20 0 21460 736 340 S 0.0 0.0 0:00.00 /sbin/udevd --dae
530 root 20 0 15192 392 196 S 0.0 0.0 0:00.00 upstart-socket-br
604 root 20 0 7268 1028 532 S 0.0 0.0 0:00.01 dhclient3 -e IF_M
703 postfix 20 0 27176 1616 1316 S 0.0 0.0 0:00.01 pickup -l -t fifo
770 root 20 0 14508 976 812 S 0.0 0.0 0:00.00 /sbin/getty -8 38
775 root 20 0 14508 980 812 S 0.0 0.0 0:00.00 /sbin/getty -8 38
780 root 20 0 14508 976 812 S 0.0 0.0 0:00.00 /sbin/getty -8 38
781 root 20 0 14508 980 812 S 0.0 0.0 0:00.00 /sbin/getty -8 38
F1Help F2Setup F3SearchF4FilterF5Tree F6SortByF7Nice -F8Nice +F9Kill F10Quit

CPU Flame Graph



perf_events CPU Flame Graphs

- We have this automated in Netflix Vector:

```
git clone --depth 1 https://github.com/brendangregg/FlameGraph
cd FlameGraph
perf record -F 99 -a –g -- sleep 30
perf script | ./stackcollapse-perf.pl |./flamegraph.pl > perf.svg
```

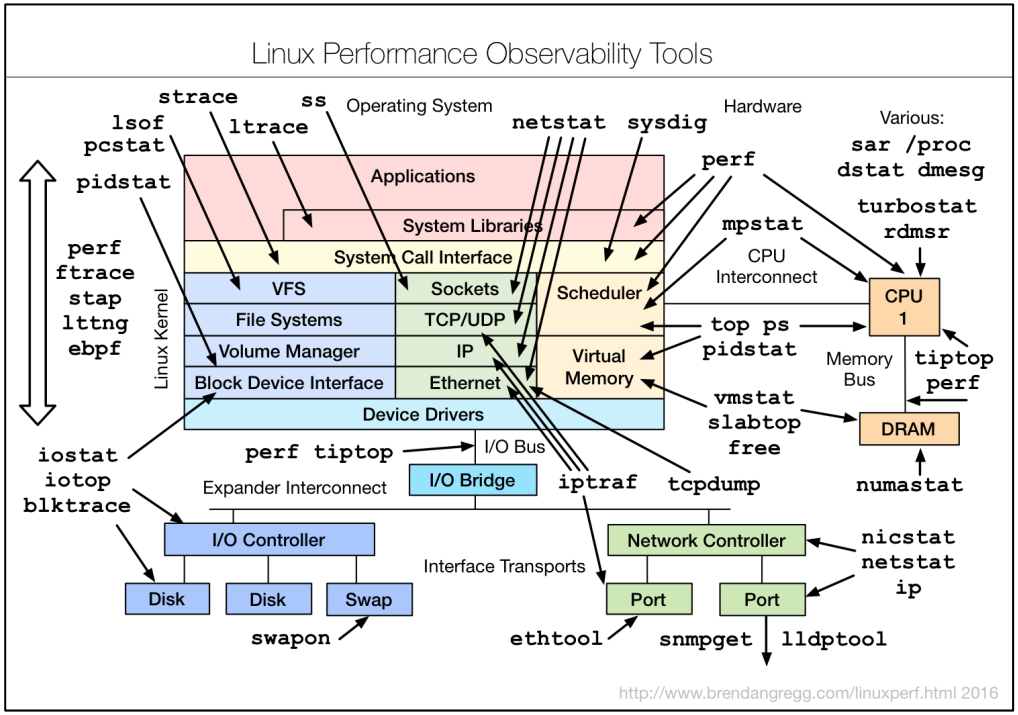
- Flame graph interpretation:
  - **x-axis**: alphabetical stack sort, to maximize merging
  - **y-axis**: stack depth
  - **color**: random, or hue can be a dimension (eg, diff)
  - Top edge is on-CPU, beneath it is ancestry
- Can also do Java & Node.js. Differentials.
- We're working on a d3 version for Vector

10. Tools Method

An *Anti*-Methodology

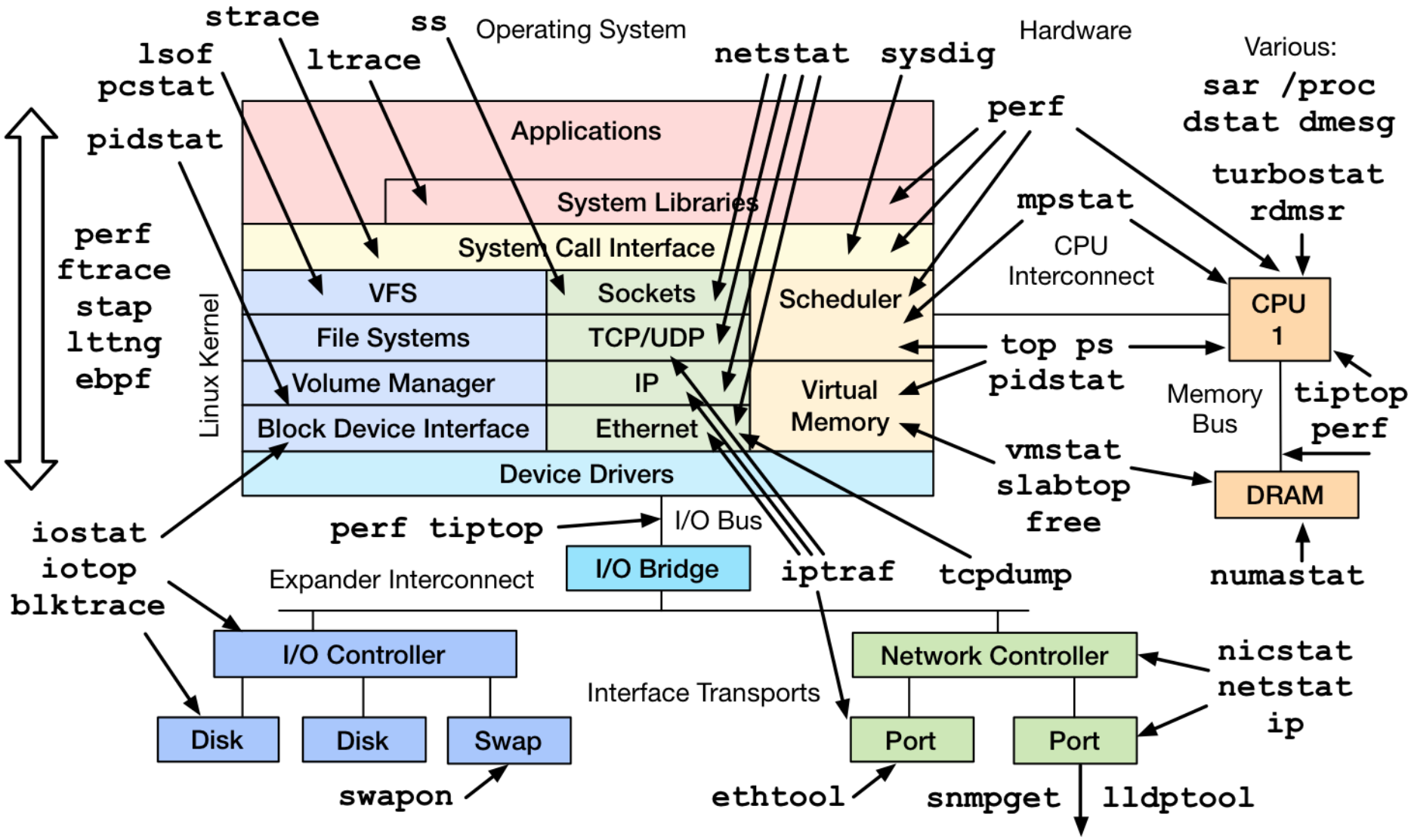
Tools Method

1. RUN EVERYTHING AND HOPE FOR THE BEST

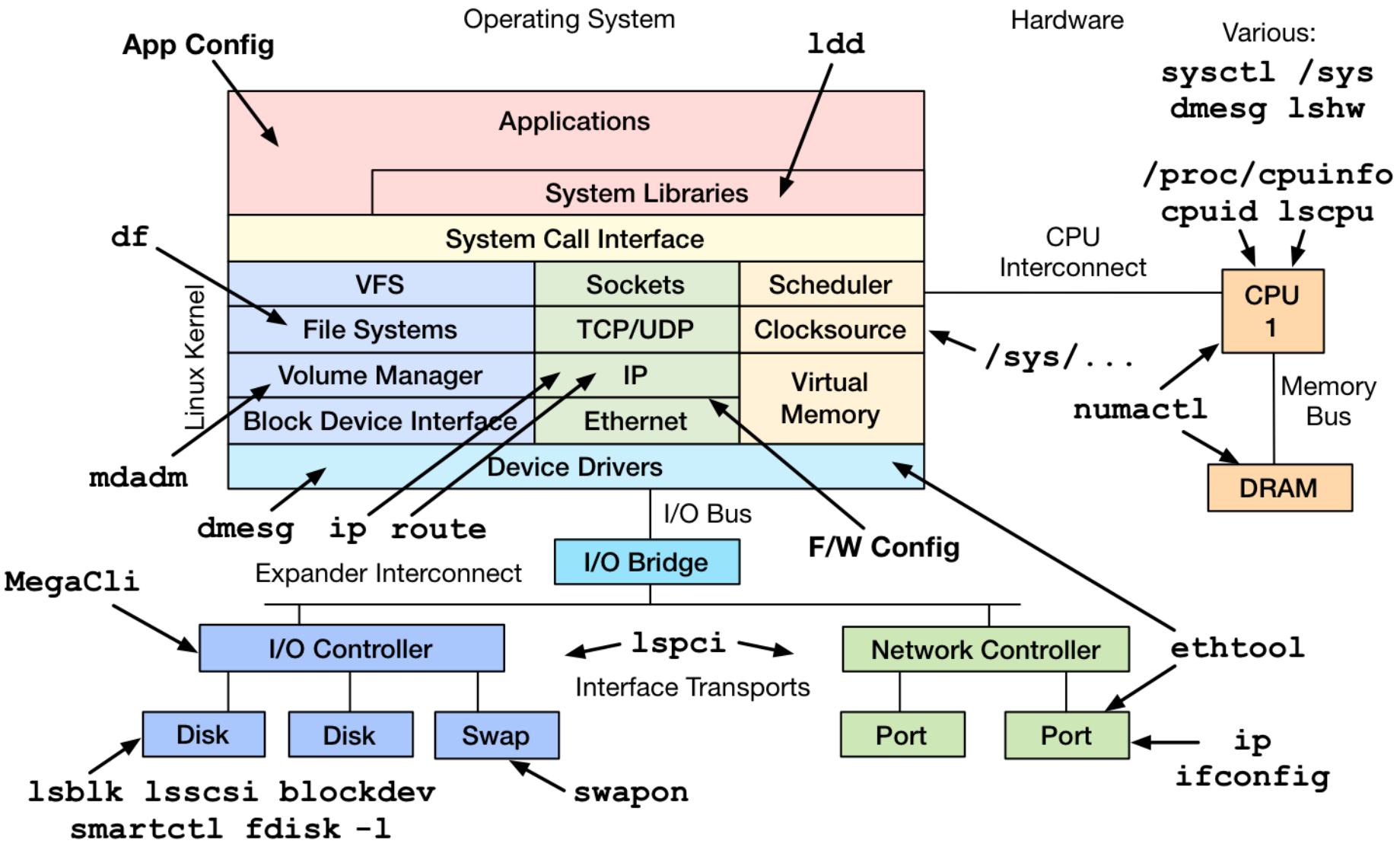


For SRE response: a mental checklist to see what might have been missed (no time to run them all)

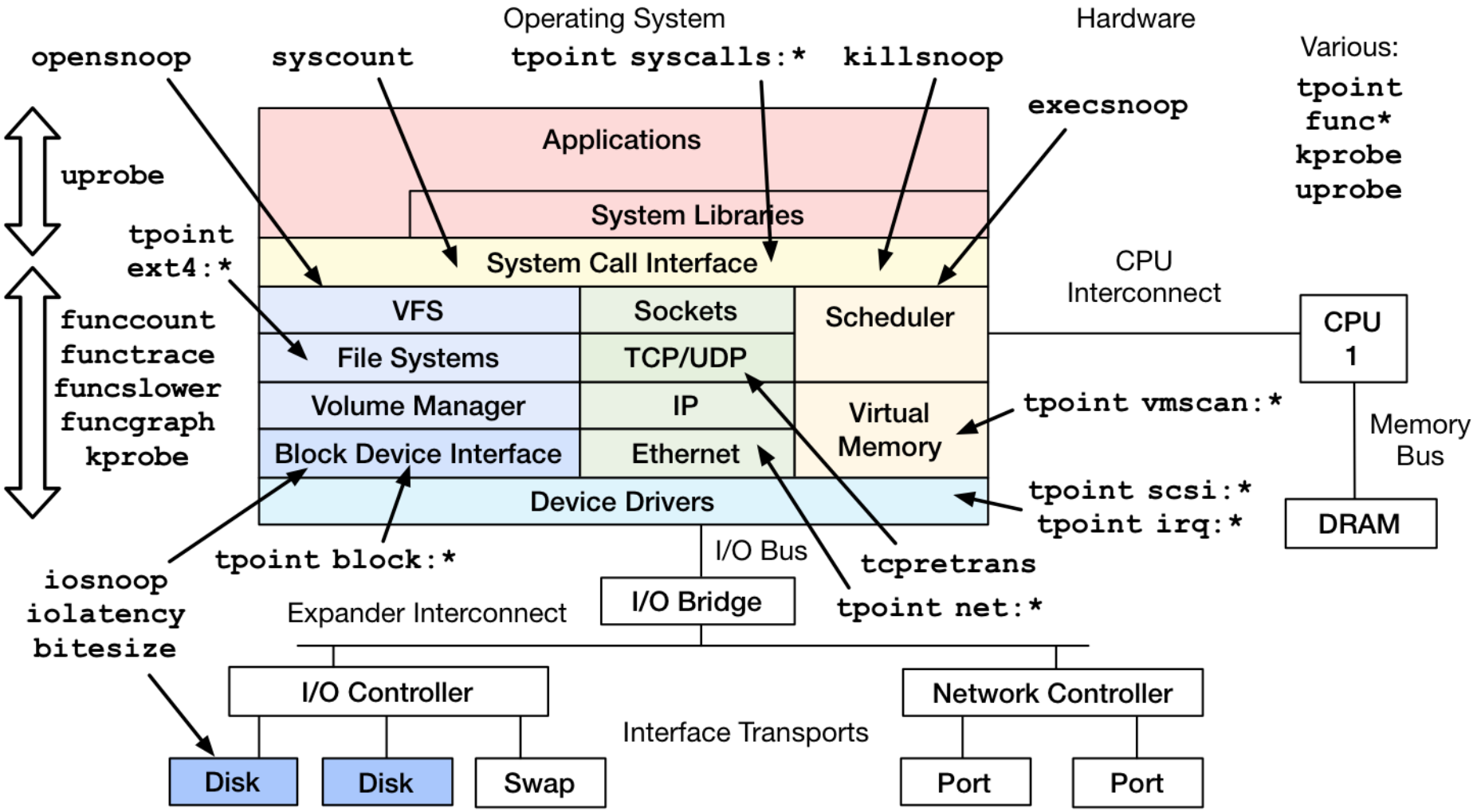
Linux Perf Observability Tools



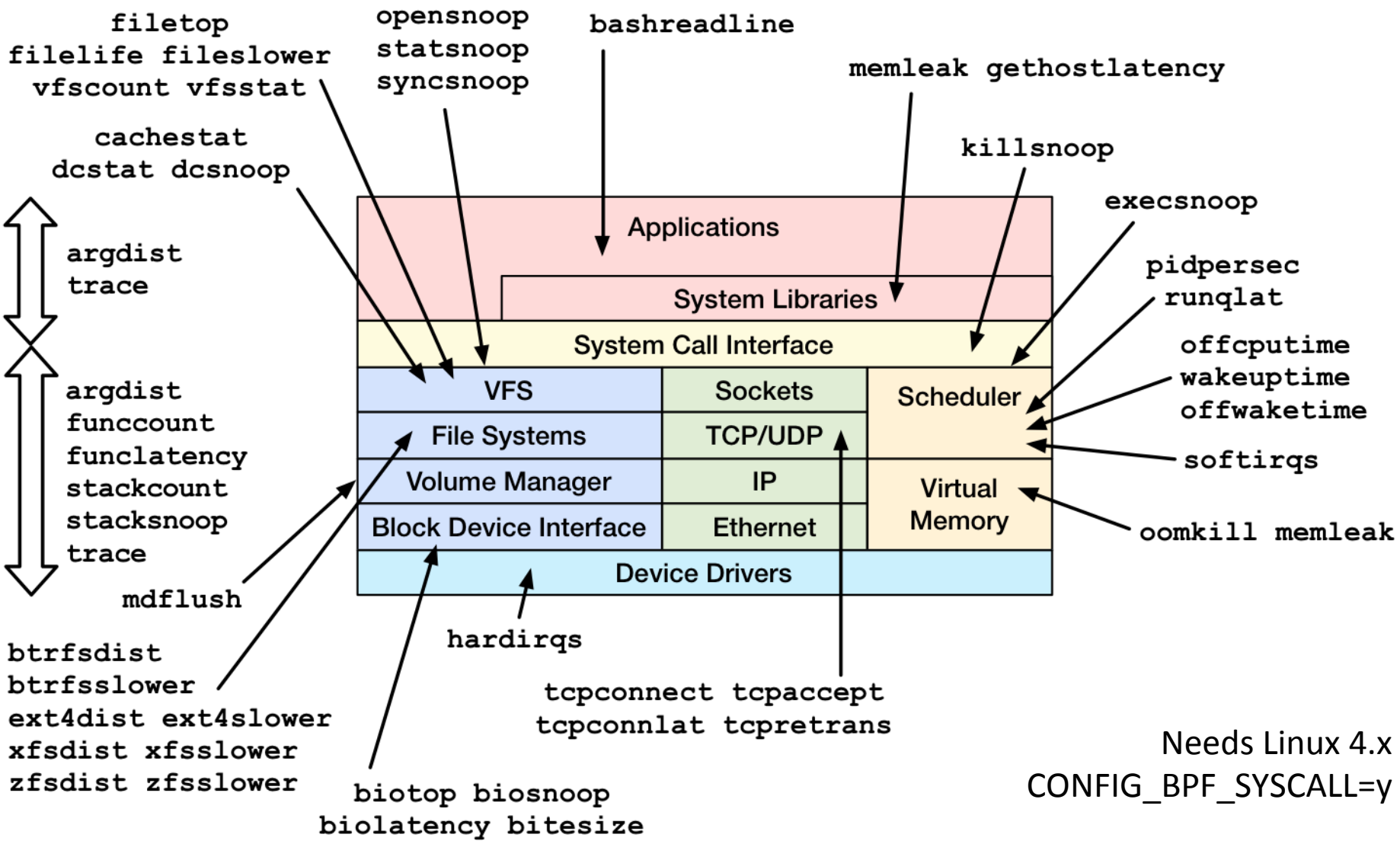
Linux Static Performance Tools



Linux perf-tools (ftrace, perf)



Linux bcc tools (BPF)



Needs Linux 4.x
CONFIG_BPF_SYSCALL=y

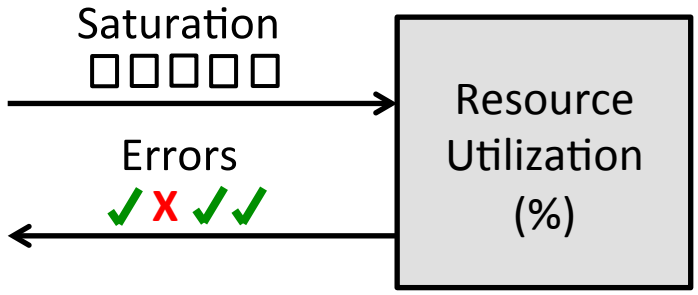
https://github.com/iovisor/bcc#tools 2016

11. USE Method

A Methodology

The USE Method

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

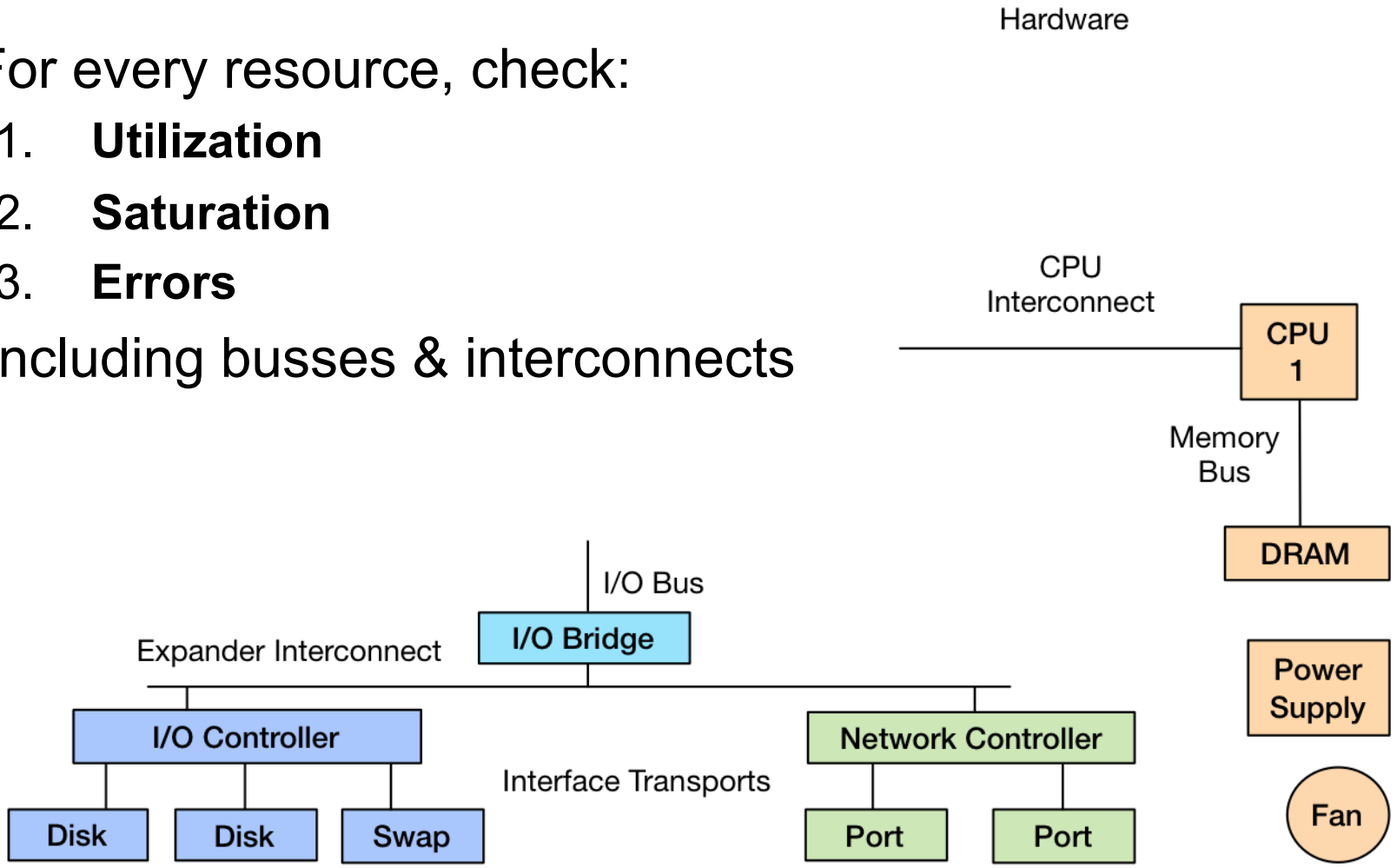


- Definitions:
  - Utilization: busy time
  - Saturation: queue length or queued time
  - Errors: easy to interpret (objective)

Used to generate checklists. Starts with the questions, then finds the tools.

USE Method for Hardware

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

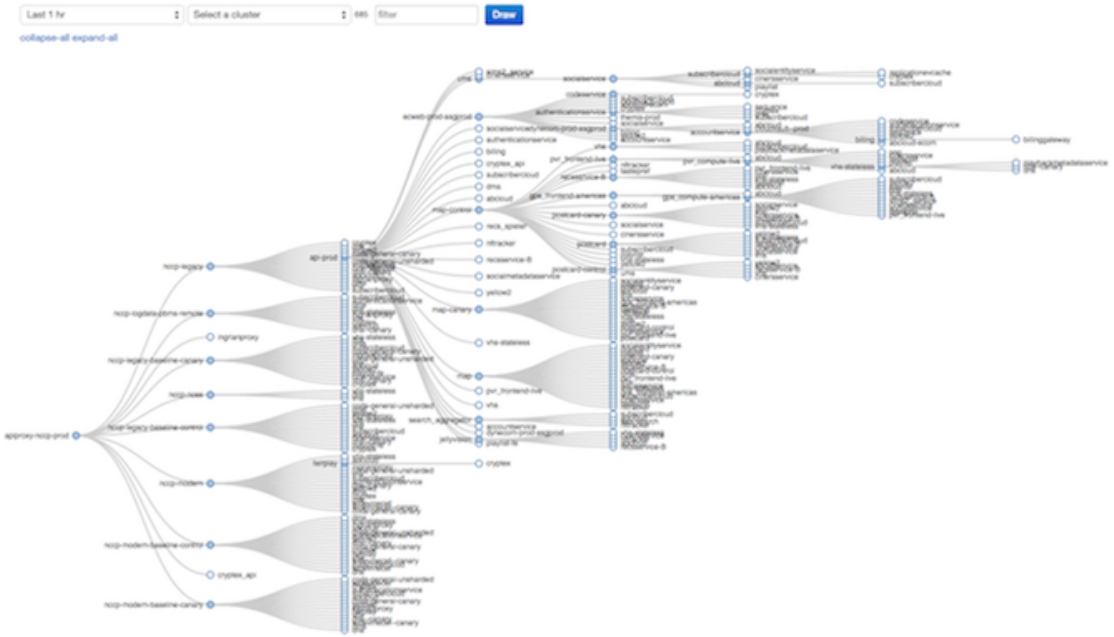
Physical Resources

(http://www.brendangregg.com/USEmethod/use-linux.html)

component	type	metric
CPU	utilization	system-wide: `vmstat 1`, "us" + "sy" + "st"; `sar -u`, sum fields except "%idle" and "%iowait"; `dstat -c`, sum fields except "idl" and "wai"; per-cpu: `mpstat -P ALL 1`, sum fields except "%idle" and "%iowait"; `sar -P ALL`, same as `mpstat`; per-process: `top`, "%CPU"; `htop`, "CPU%"; `ps -o pcpu`; `pidstat 1`, "%CPU"; per-kernel-thread: `top/htop` ("K" to toggle), where VIRT == 0 (heuristic). [1]
CPU	saturation	system-wide: `vmstat 1`, "r" > CPU count [2]; `sar -q`, "runq-sz" > CPU count; `dstat -p`, "run" > CPU count; per-process: /proc/PID/schedstat 2nd field (sched_info.run_delay); `perf sched latency` (shows "Average" and "Maximum" delay per-schedule); dynamic tracing, eg, SystemTap schedtimes.stp "queued(us)" [3]
CPU	errors	`perf` (LPE) 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: `free -m`, "Mem:" (main memory), "Swap:" (virtual memory); `vmstat 1`, "free" (main memory), "swap" (virtual memory); `sar -r`, "%memused"; `dstat -m`, "free"; `slabtop -s c` for kmem slab usage; per-process: `top/htop`, "RES" (resident main memory), "VIRT" (virtual memory), "Mem" for system-wide summary
Memory capacity	saturation	system-wide: `vmstat 1`, "si"/"so" (swapping); `sar -B`, "pgscank" + "pgscand" (scanning); `sar -W`; per-process: 10th field (min_flt) from /proc/PID/stat for minor-fault rate, or dynamic tracing [5]; OOM killer: `dmesg | grep killed`
Memory capacity	errors	`dmesg` for physical failures; dynamic tracing, eg, SystemTap uprobes for failed malloc()s
Network Interfaces	utilization	`sar -n DEV 1`, "rxKB/s"/max "txKB/s"/max; `ip -s link`, RX/TX tput / max bandwidth; /proc/net/dev, "bytes" RX/TX tput/max; nicstat "%Util" [6]
Network		`ifconfig`, "errors", "dropped"; `netstat -s`, "segments retransmitted"; `sar -n EDEV`, *drop and *fifo metrics

USE Method for Distributed Systems

- Draw a service diagram, and for every service:
  1. **Utilization**: resource usage (CPU, network)
  2. **Saturation**: request queueing, timeouts
  3. **Errors**
- Turn into a dashboard



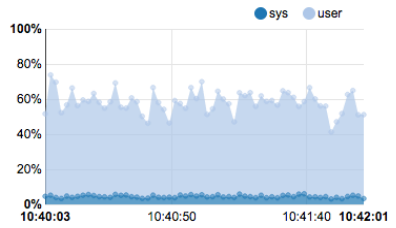
Netflix Vector

- Real time instance analysis tool
  - https://github.com/netflix/vector
  - http://techblog.netflix.com/2015/04/introducing-vector-netflixs-on-host.html
- USE method-inspired metrics
  - More in development, incl. flame graphs

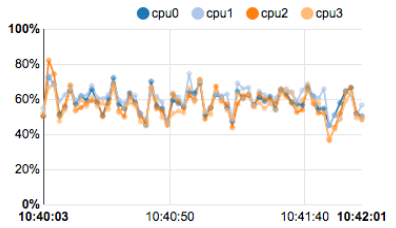
VECTOR

Hostname ec2-[redacted].compute-1.amazonaws.com ✔ | Widget ▾ | Window 2 min | Interval 2 sec

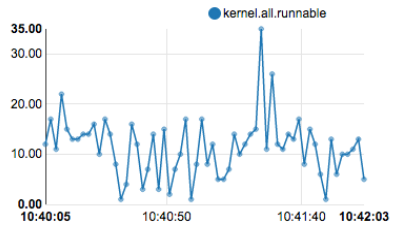
CPU Utilization



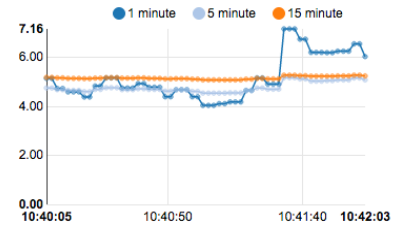
Per-CPU Utilization



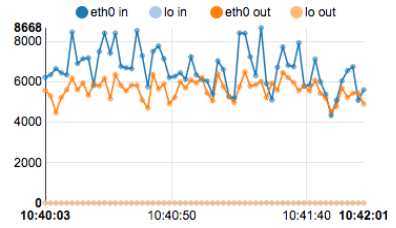
Runnable



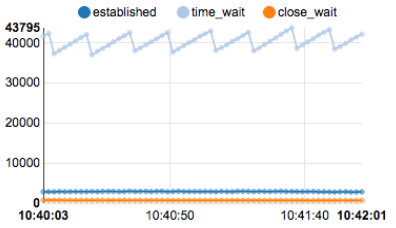
Load Average



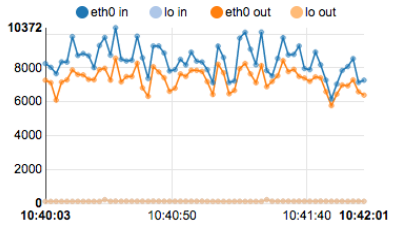
Network Throughput (kB)



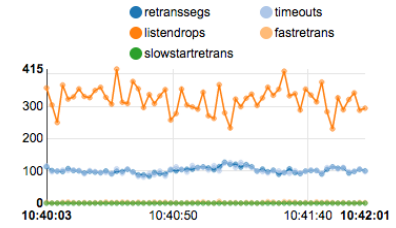
TCP Connections



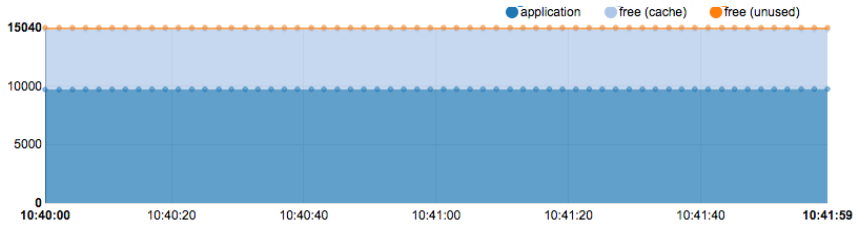
Network Packets



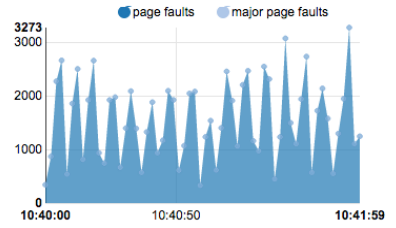
Network Retransmits



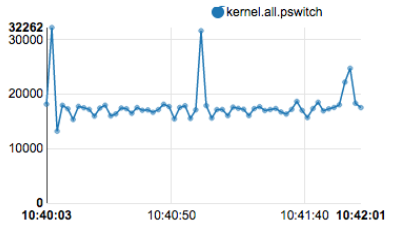
Memory Utilization



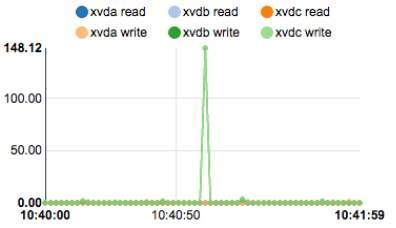
Page Faults



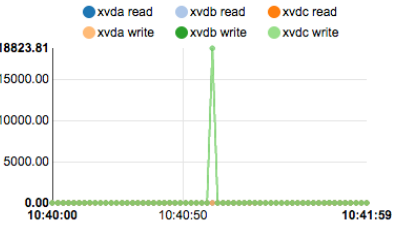
Context Switches



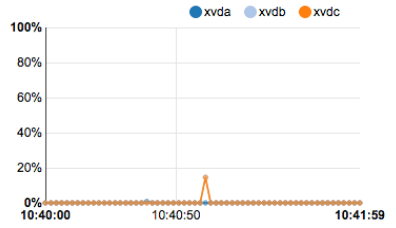
Disk IOPS



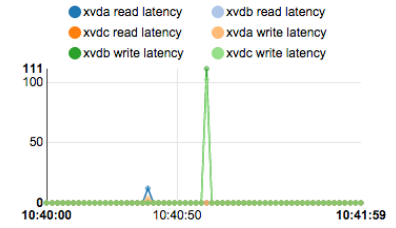
Disk Throughput (Bytes)

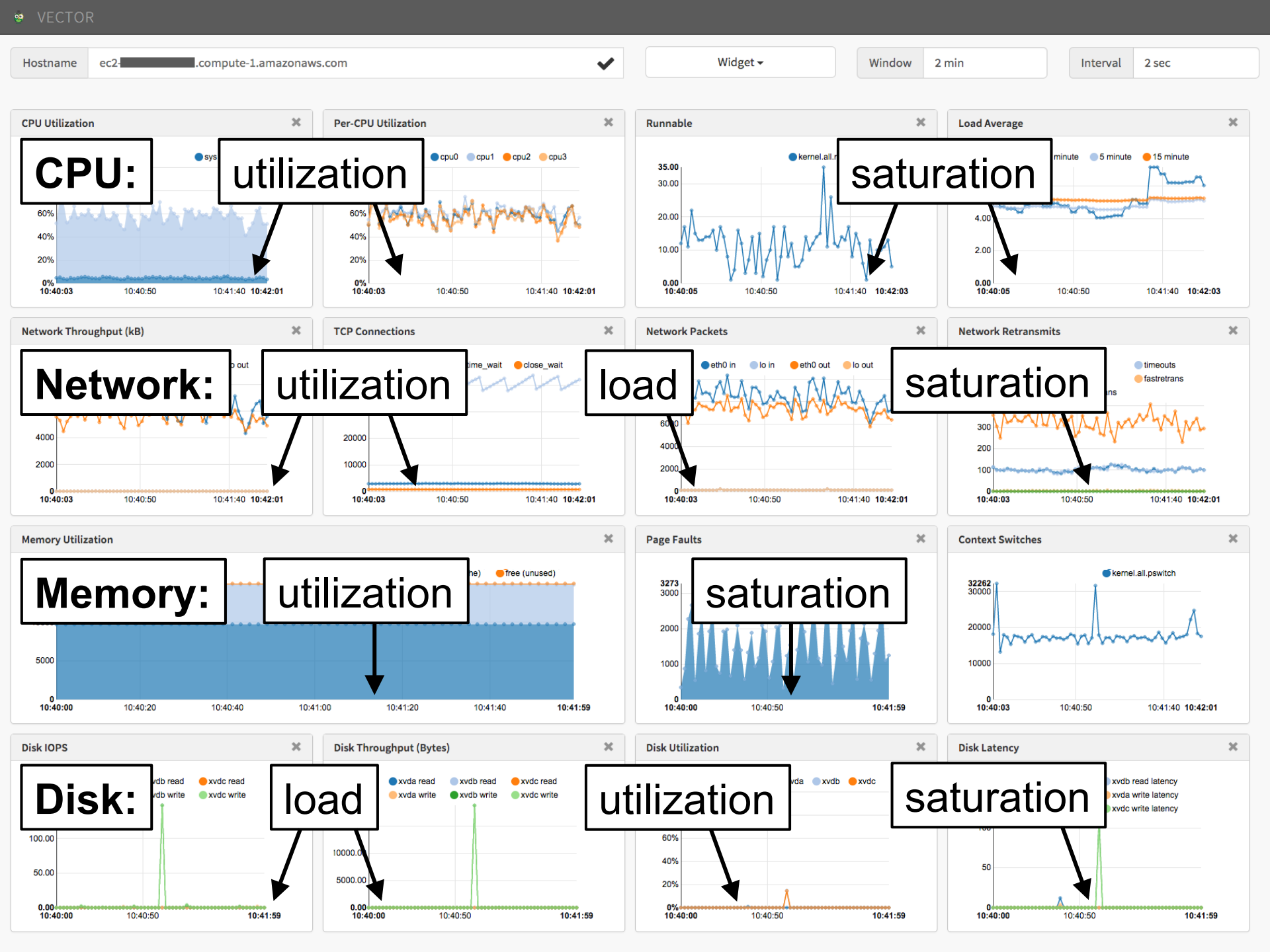


Disk Utilization



Disk Latency




VECTOR
Hostname
ec2-[redacted].compute-1.amazonaws.com
Widget
Window
2 min
Interval
2 sec
CPU Utilization
Per-CPU Utilization
Runnable
Load Average
CPU:
utilization
saturation
Network Throughput (kB)
TCP Connections
Network Packets
Network Retransmits
Network:
utilization
load
saturation
Memory Utilization
Page Faults
Context Switches
Memory:
utilization
saturation
Disk IOPS
Disk Throughput (Bytes)
Disk Utilization
Disk Latency
Disk:
load
utilization
saturation

12. Bonus: External Factor Checklist

External Factor Checklist

1. Sports ball?
2. Power outage?
3. Snow storm?
4. Internet/ISP down?
5. Vendor firmware update?
6. Public holiday/celebration?
7. Chaos Kong?

Social media searches (Twitter) often useful

- Can also be NSFW

Take Aways

- Checklists are great
  - Speed, Completeness, Starting/Ending Point, Training
  - Can be ad hoc, or from a methodology (USE method)
- Service dashboards
  - Serve as checklists
  - Metrics: Load, Errors, Latency, Saturation, Instances
- System dashboards with Linux BPF
  - Latency histograms & heatmaps, etc. Free your mind.

Please create and share more checklists

References

- Netflix Tech Blog:
  - http://techblog.netflix.com/2015/11/linux-performance-analysis-in-60s.html
  - http://techblog.netflix.com/2015/02/sps-pulse-of-netflix-streaming.html
  - http://techblog.netflix.com/2015/04/introducing-vector-netflixs-on-host.html
- Linux Performance & BPF tools:
  - http://www.brendangregg.com/linuxperf.html
  - https://github.com/iovisor/bcc#tools
- USE Method Linux:
  - http://www.brendangregg.com/USEmethod/use-linux.html
- Flame Graphs:
  - http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html
- Heat maps:
  - http://cacm.acm.org/magazines/2010/7/95062-visualizing-system-latency/fulltext
  - http://www.brendangregg.com/heatmaps.html
- Books:
  - Beyer, B., et al. *Site Reliability Engineering*. O'Reilly, Apr 2016
  - Gawande, A. *The Checklist Manifesto*. Metropolitan Books, 2008
  - Gregg, B. *Systems Performance*. Prentice Hall, 2013 (more checklists & methods!)
- Thanks: Netflix Perf & Core teams for predash, pretriage, Vector, etc

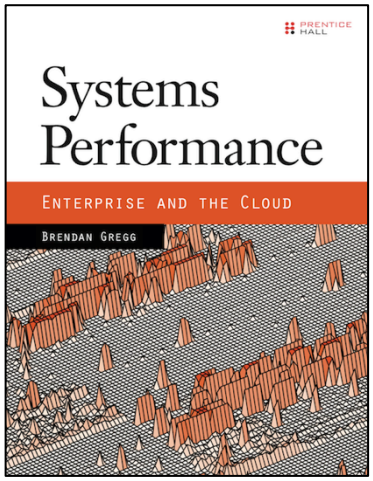
Thanks

http://slideshare.net/brendangregg

http://www.brendangregg.com

bgregg@netflix.com

@brendangregg



Netflix is hiring SREs!

NETFLIX