From Clouds to Roots

Brendan Gregg

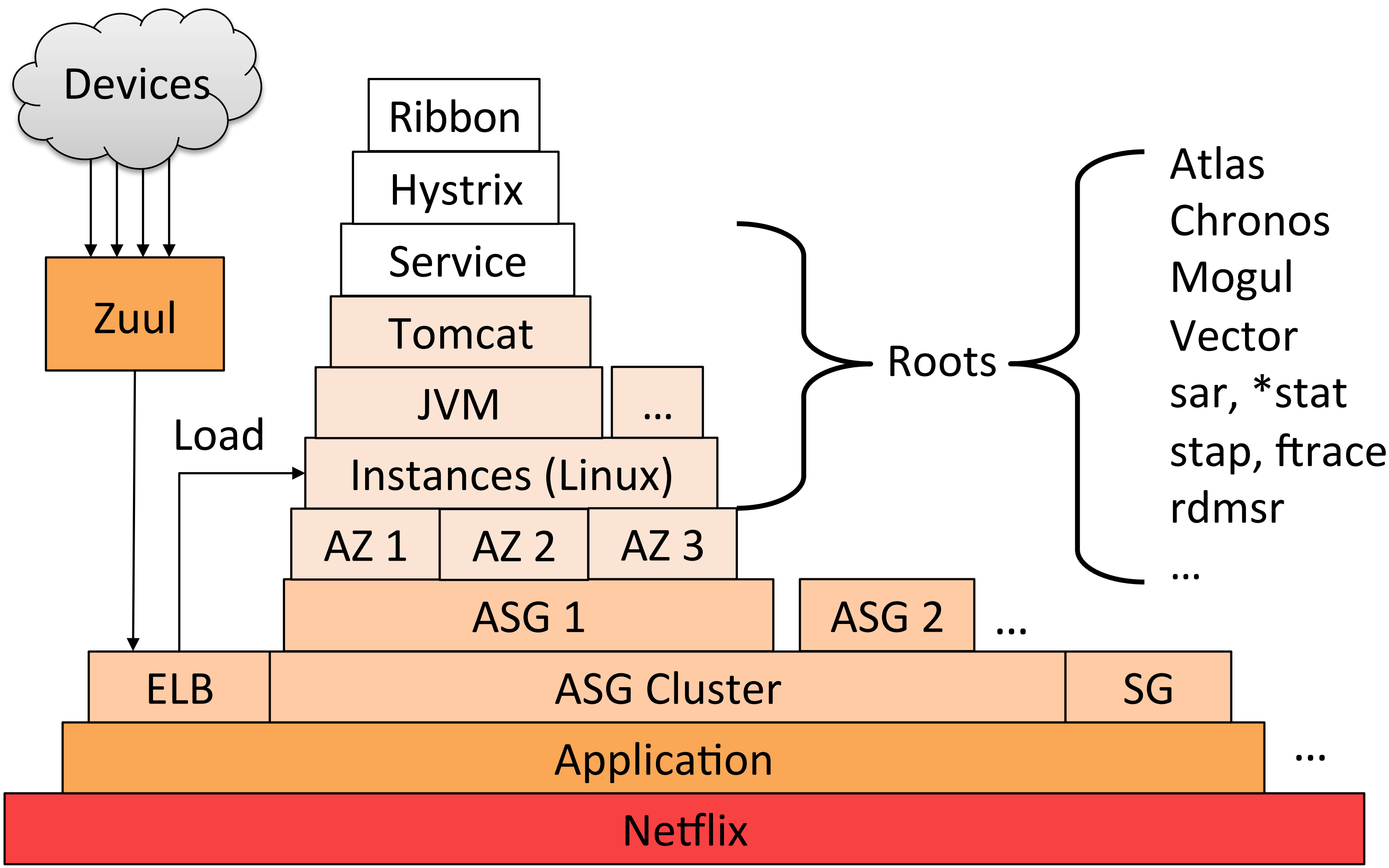
Senior Performance Architect
Performance Engineering Team

bgregg@netflix.com, @brendangregg

NETFLIX

September, 2014

Root Cause Analysis at Netflix

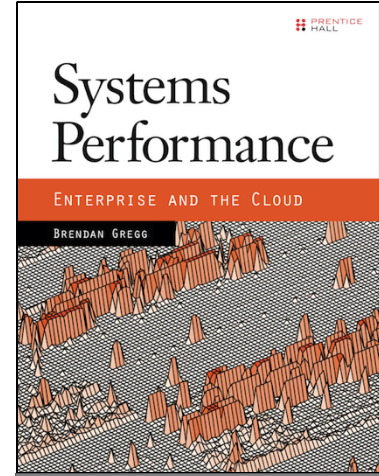


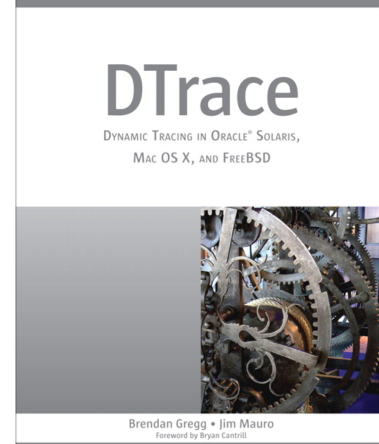
NETFLIX

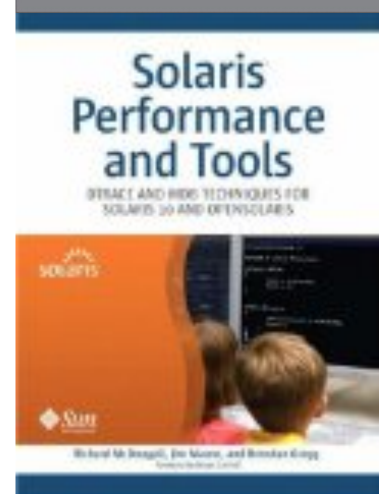
- Massive AWS EC2 Linux cloud
  - Tens of thousands of server instances
  - Autoscale by ~3k each day
  - CentOS and Ubuntu
- FreeBSD for content delivery
  - Approx 33% of US Internet traffic at night
- Performance is critical
  - Customer satisfaction: >50M subscribers
  - $$$ price/performance
  - Develop tools for cloud-wide analysis

Brendan Gregg

- Senior Performance Architect, Netflix
  - Linux and FreeBSD performance
  - Performance Engineering team (@coburnw)
- Recent work:
  - Linux perf-tools, using ftrace & perf_events
  - Systems Performance, Prentice Hall
- Previous work includes:
  - USE Method, flame graphs, latency & utilization heat maps, DTraceToolkit, iosnoop and others on OS X, ZFS L2ARC
- Twitter @brendangregg

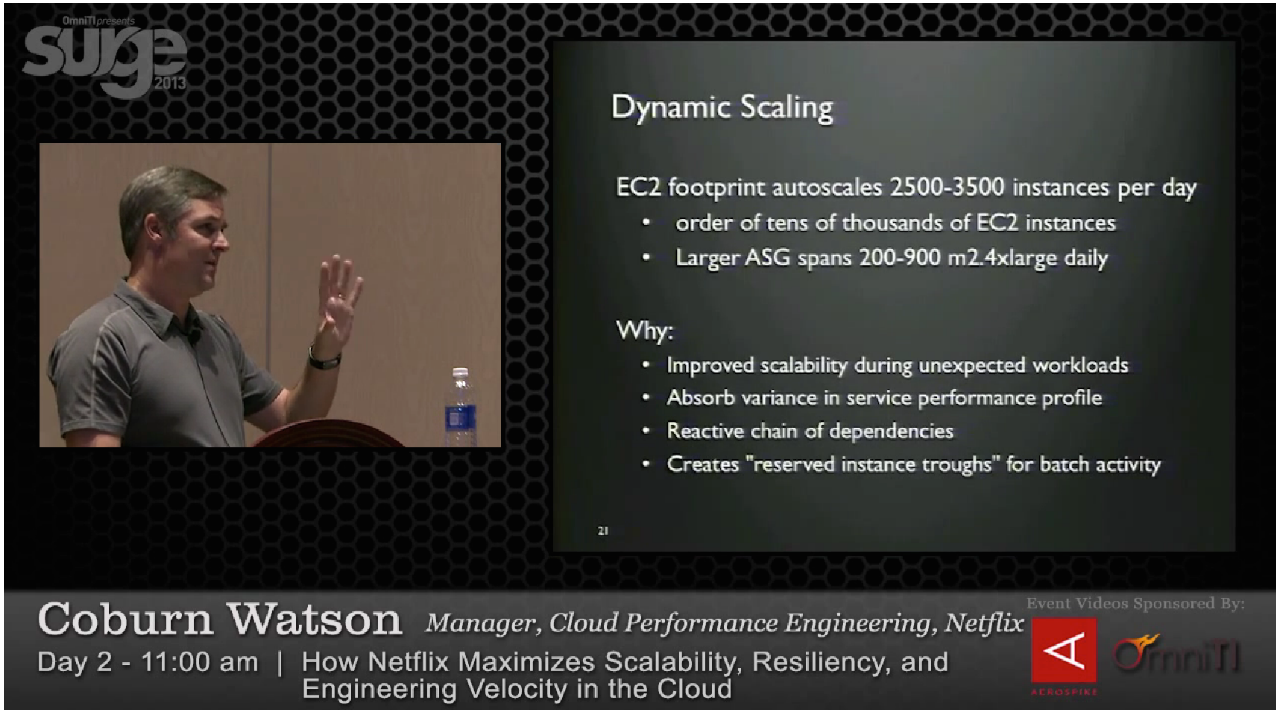






Last year at Surge…

- I saw a great Netflix talk by Coburn Watson:



- https://www.youtube.com/watch?v=7-13wV3WO8Q
- He’s now my manager (and also still hiring!)

Agenda

- The Netflix Cloud
  - How it works: ASG clusters, Hystrix, monkeys
  - And how it may fail
- Root Cause Performance Analysis
  - Why it's still needed
- Cloud analysis
- Instance analysis

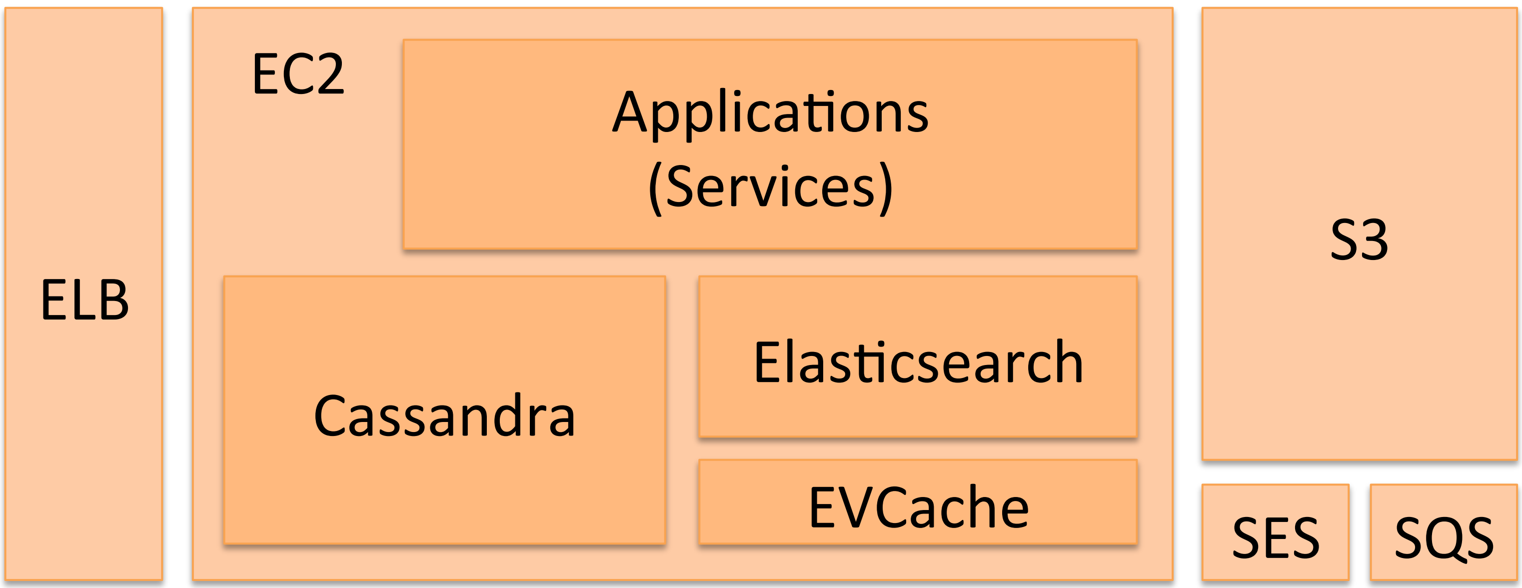
Terms

- AWS: Amazon Web Services
- EC2: AWS Elastic Compute 2 (cloud instances)
- S3: AWS Simple Storage Service (object store)
- ELB: AWS Elastic Load Balancers
- SQS: AWS Simple Queue Service
- SES: AWS Simple Email Service
- CDN: Content Delivery Network
- OCA: Netflix Open Connect Appliance (streaming CDN)
- QoS: Quality of Service
- AMI: Amazon Machine Image (instance image)
- ASG: Auto Scaling Group
- AZ: Availability Zone
- NIWS: Netflix Internal Web Service framework (Ribbon)
- MSR: Model Specific Register (CPU info register)
- PMC: Performance Monitoring Counter (CPU perf counter)

The Netflix Cloud

The Netflix Cloud

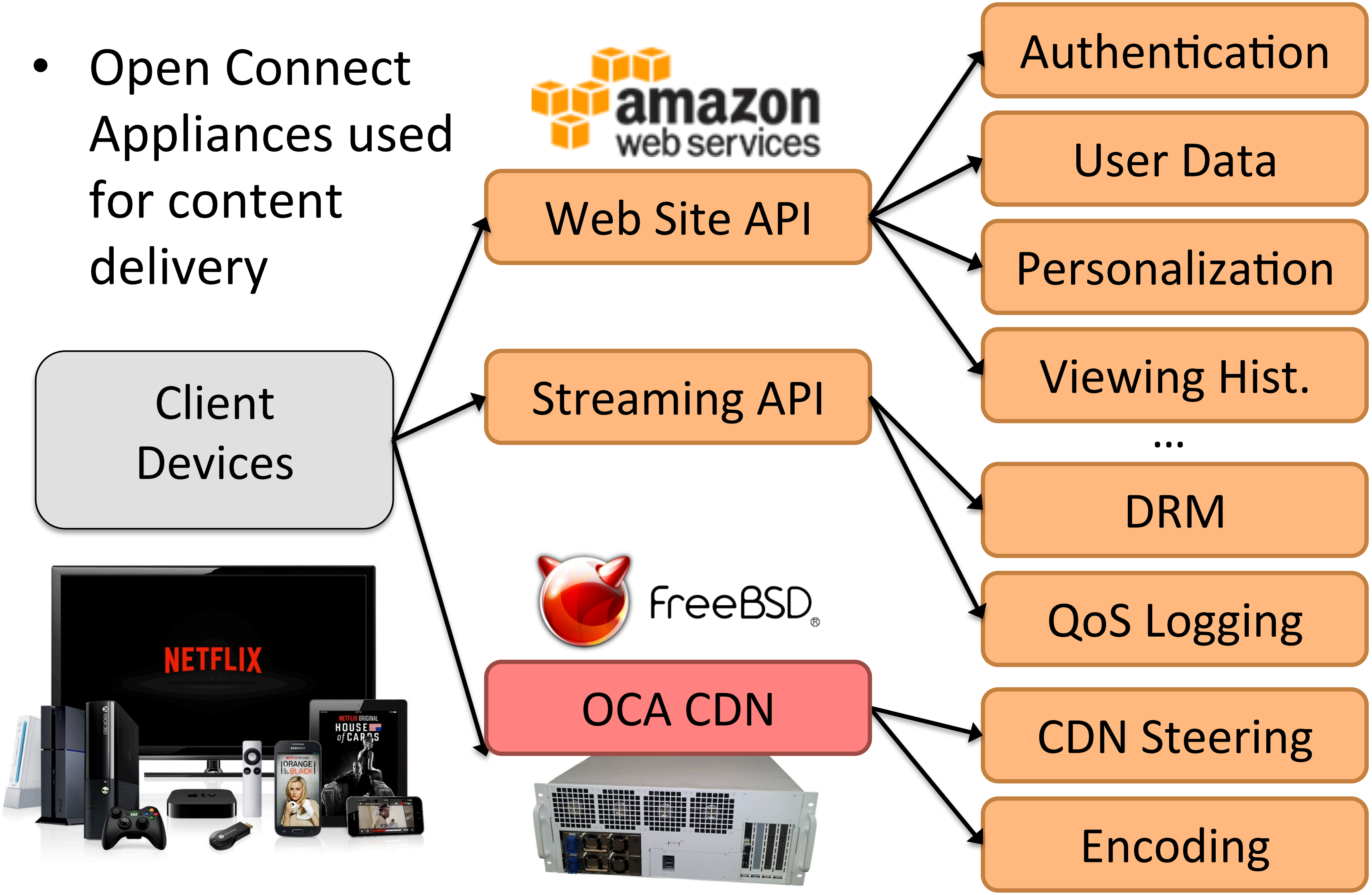
- Tens of thousands of cloud instances on AWS EC2, with S3 and Cassandra for storage



- Netflix is implemented by multiple logical services

Netflix Services

- Open Connect Appliances used for content delivery



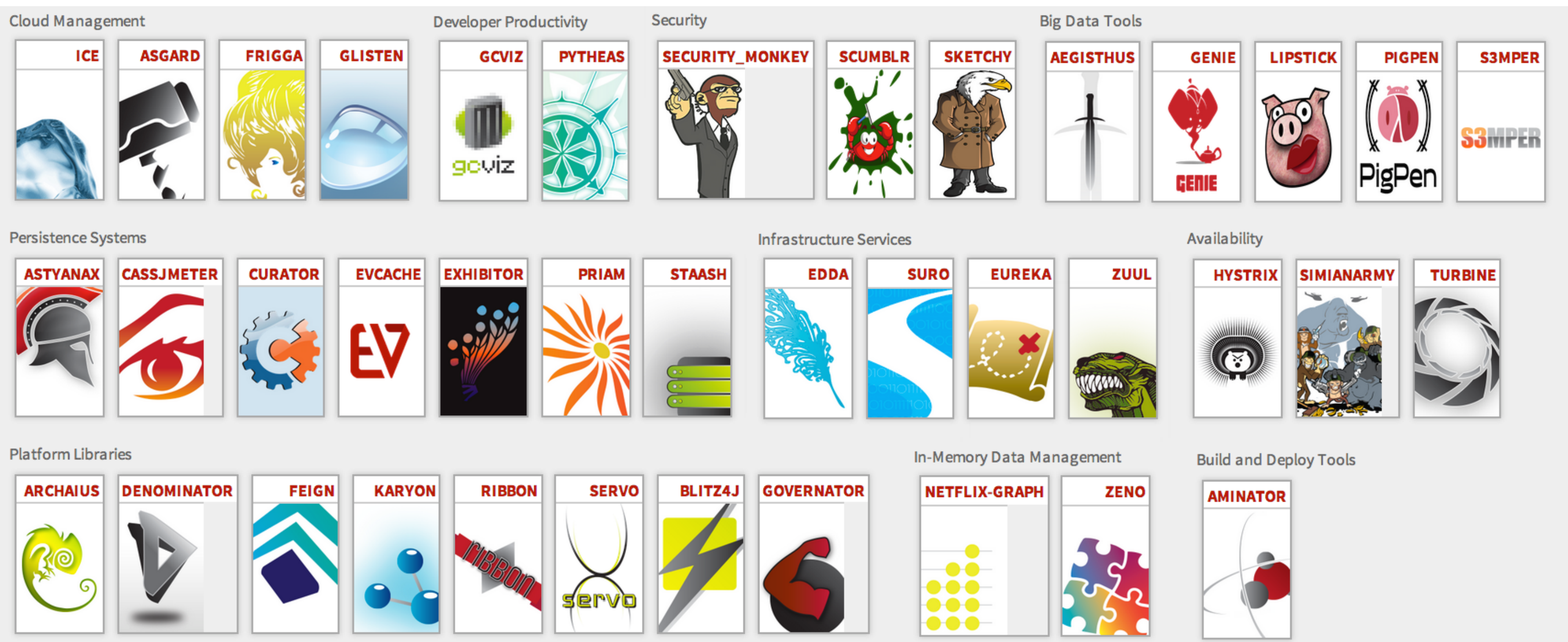
Freedom and Responsibility

- Culture deck is true
  - http://www.slideshare.net/reed2001/culture-1798664 (9M views!)
- Deployment freedom
  - Service teams choose their own tech & schedules
  - Purchase and use cloud instances without approvals
  - Netflix environment changes fast!

Cloud Technologies

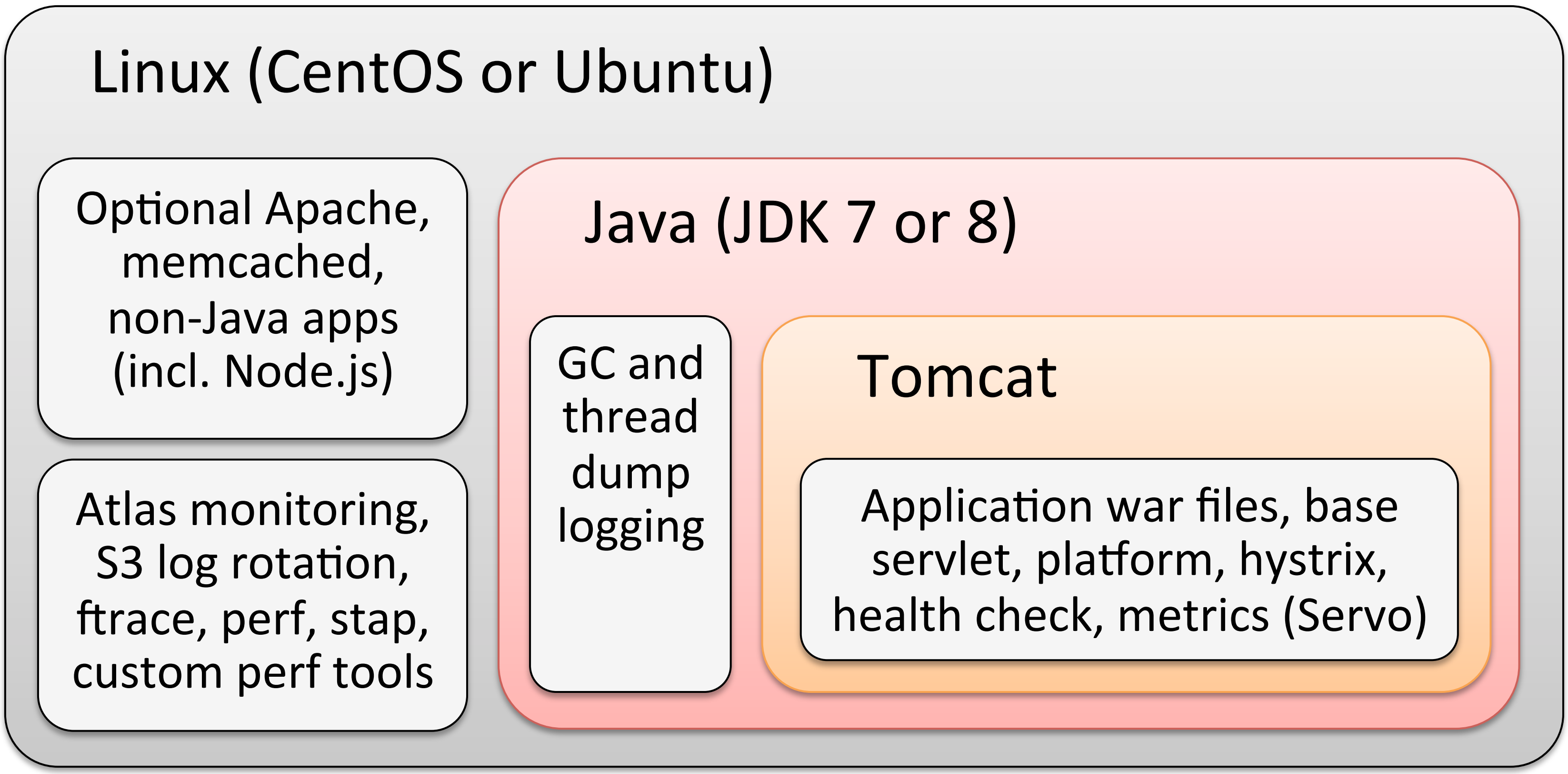
NETFLIX OSS

- Numerous open source technologies are in use:
  - Linux, Java, Cassandra, Node.js, ...
- Netflix also open sources: netflix.github.io



Cloud Instances

- Base server instance image + customizations by service teams (BaseAMI). Typically:

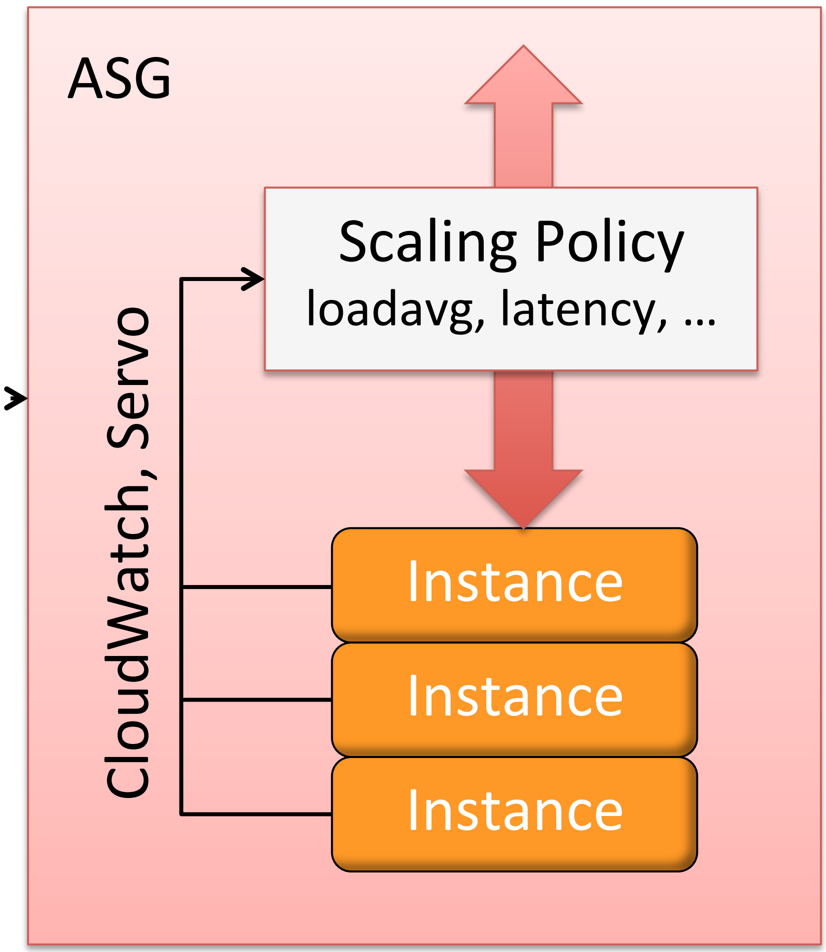


Scalability and Reliability

#	Problem	Solution
1	Load increases	Auto scale with ASGs
2	Poor performing code push	Rapid rollback with red/black ASG clusters
3	Instance failure	Hystrix timeouts and secondaries
4	Zone/Region failure	Zuul to reroute traffic
5	Overlooked and unhandled issues	Simian army
6	Poor performance	Atlas metrics, alerts, Chronos

1. Auto Scaling Groups

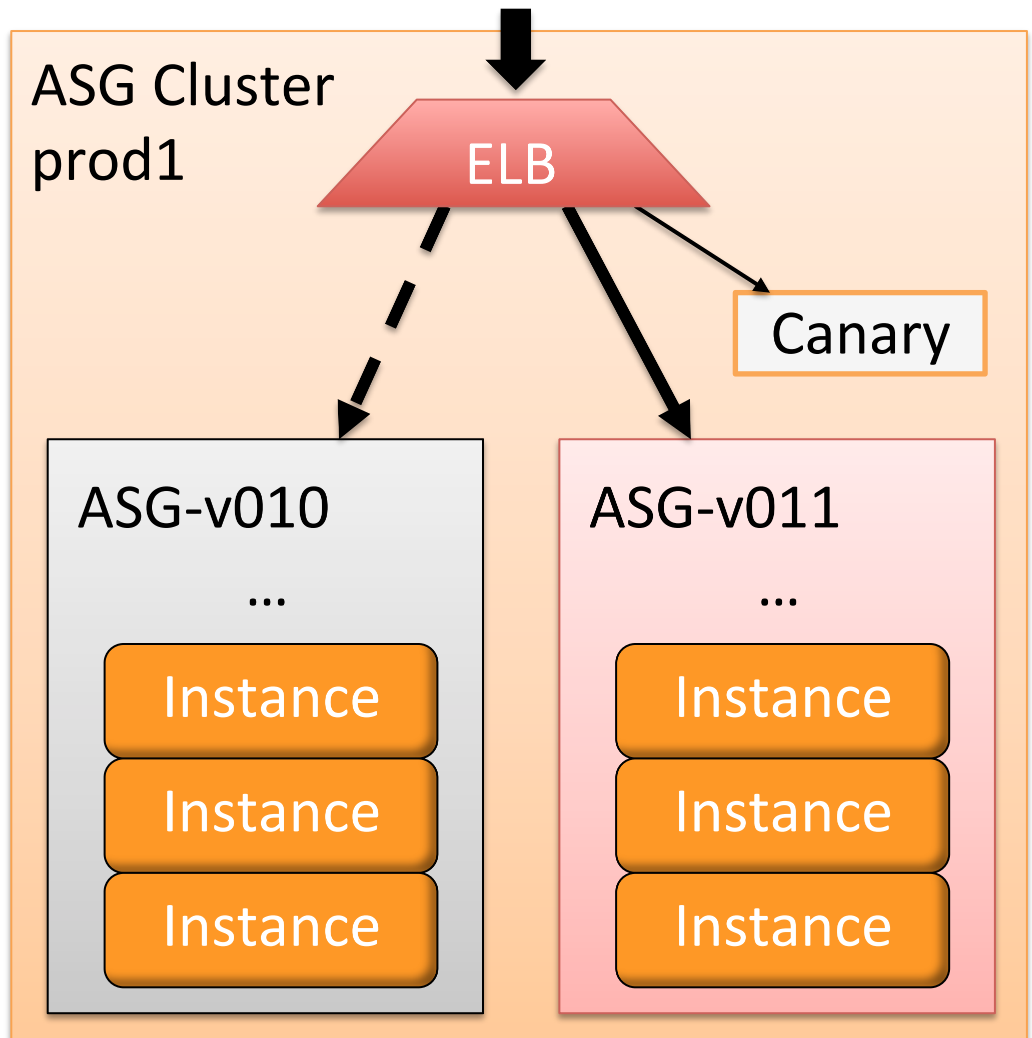
Cloud Configuration Management



- Instances automatically added or removed by a custom scaling policy
  - A broken policy could cause false scaling
- Alerts & audits used to check scaling is sane

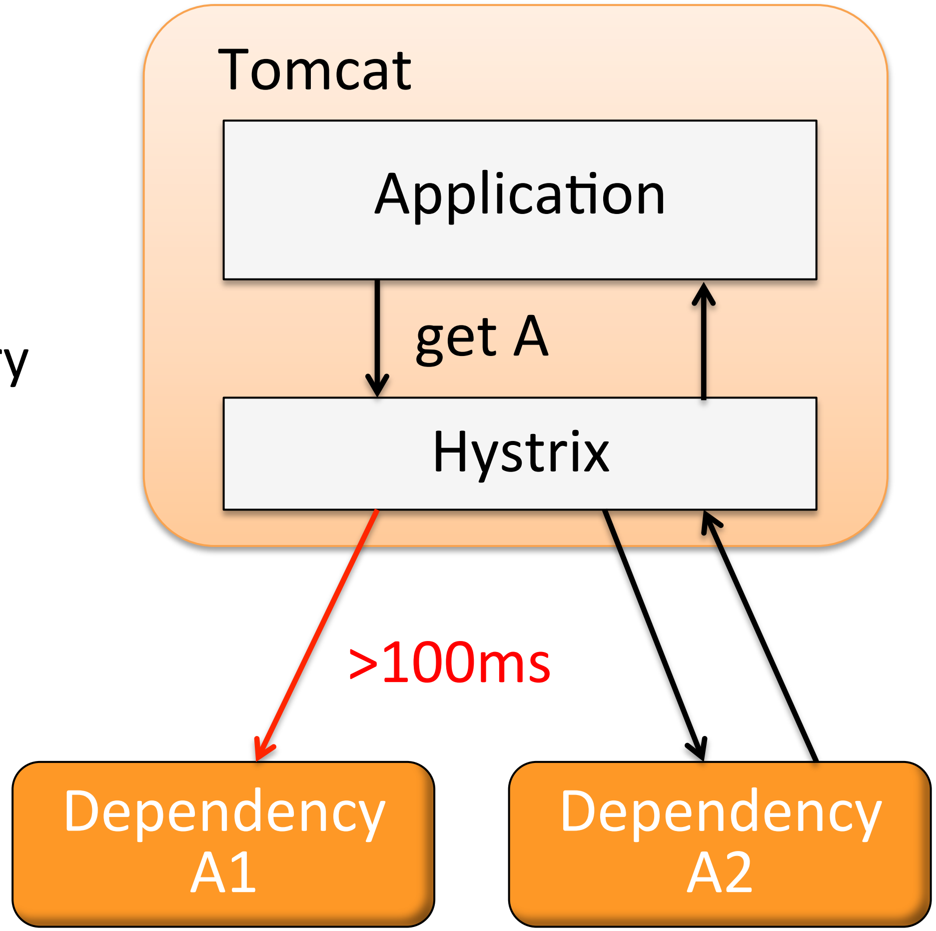
2. ASG Clusters

- How code versions are really deployed
- Traffic managed by Elastic Load Balancers (ELBs)
- Fast rollback if issues are found
  - Might rollback undiagnosed issues
- Canaries can also be used for testing (and automated)



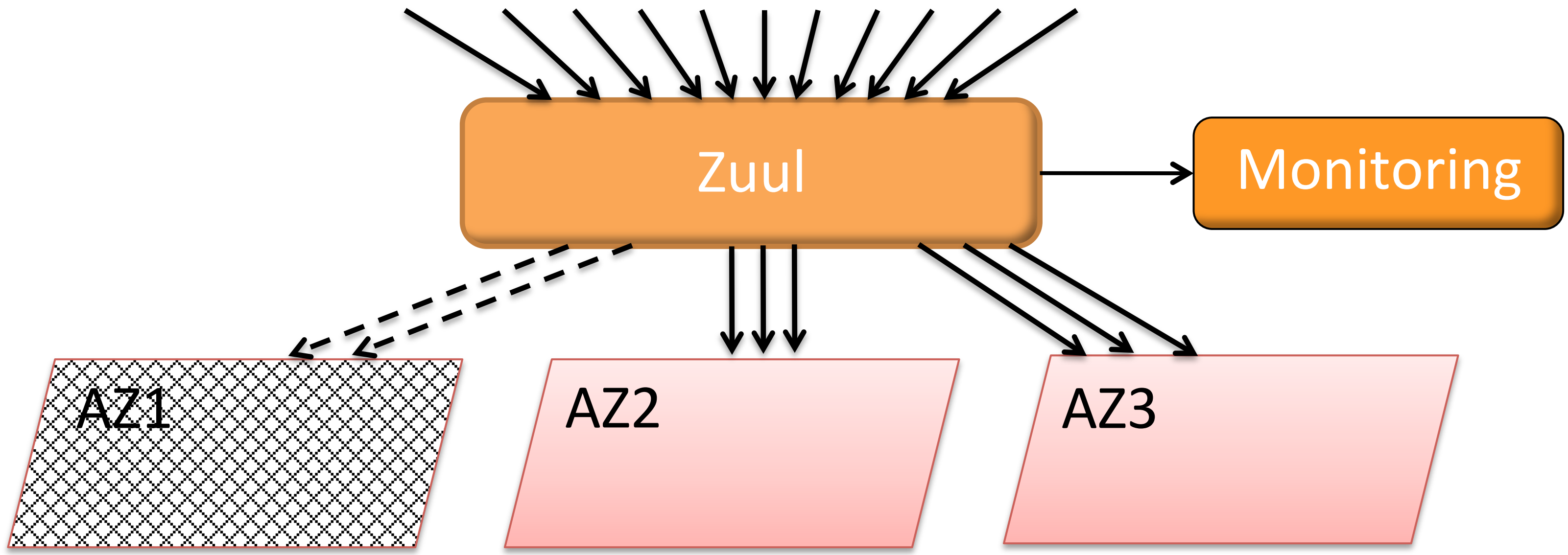
3. Hystrix

- A library for latency and fault tolerance for dependency services
  - Fallbacks, degradation, fast fail and rapid recovery
  - Supports timeouts, load shedding, circuit breaker
  - Uses thread pools for dependency services
  - Realtime monitoring
- Plus the Ribbon IPC library (NIWS), which adds even more fault tolerance



4. Redundancy

- All device traffic goes through the Zuul proxy:
  - dynamic routing, monitoring, resiliency, security
- Availability Zone failure: run from 2 of 3 zones
- Region failure: reroute traffic

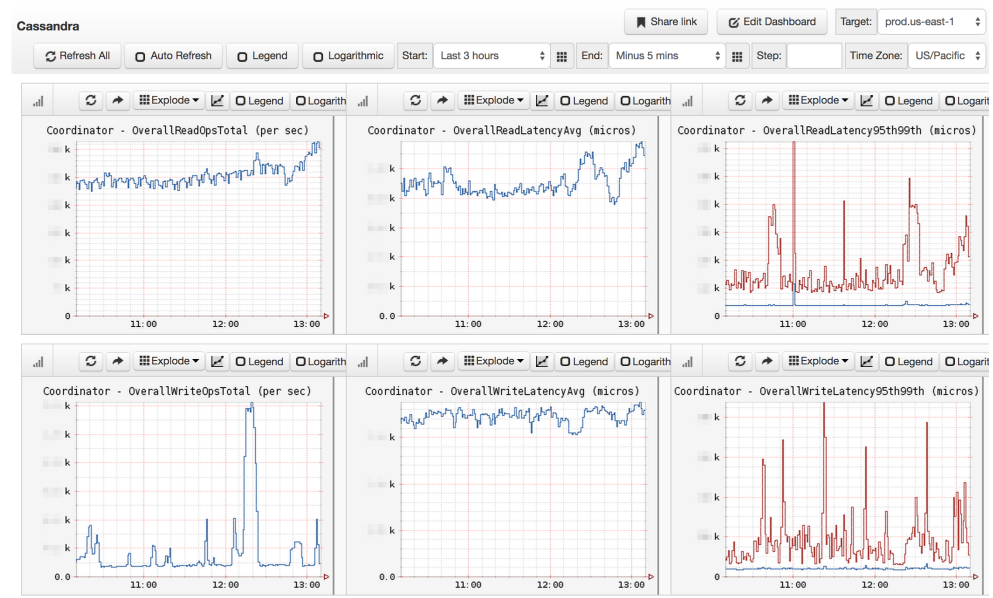


5. Simian Army

- Ensures cloud handles failures through regular testing
- Monkeys:
  - Latency: artificial delays
  - Conformity: kills non-best-practices instances
  - Doctor: health checks
  - Janitor: unused instances
  - Security: checks violations
  - 10-18: geographic issues
  - Chaos Gorilla: AZ failure
- We're hiring Chaos Engineers!

6. Atlas, alerts, Chronos

- Atlas: Cloud-wide monitoring tool
  - Millions of metrics, quick rollups, custom dashboards:



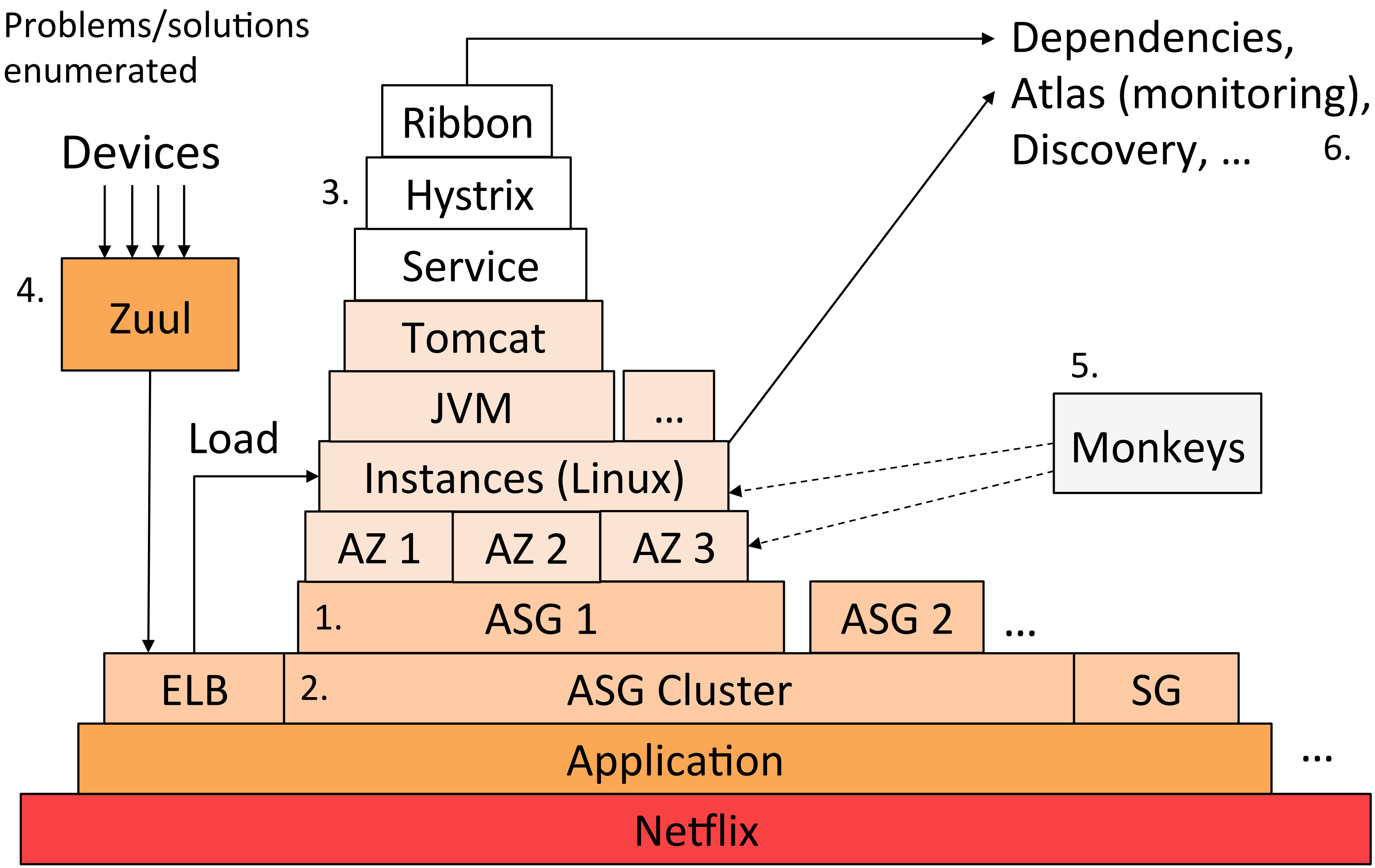
- Alerts: Custom, using Atlas metrics
  - In particular, error & timeout rates on client devices
- Chronos: Change tracking
  - Used during incident investigations

In Summary

- Netflix is very good at automatically handling failure
  - Issues often lead to rapid instance growth (ASGs)
- Good for customers
  - Fast workaround
- Good for engineers
  - Fix later, 9-5

#	Problem	Solution
1	Load increases	ASGs
2	Poor performing code push	ASG clusters
3	Instance issue	Hystrix
4	Zone/Region issue	Zuul
5	Overlooked and unhandled issues	Monkeys
6	Poor performance	Atlas, alerts, Chronos

Typical Netflix Stack



* Exceptions

- Apache Web Server
- Node.js
- …

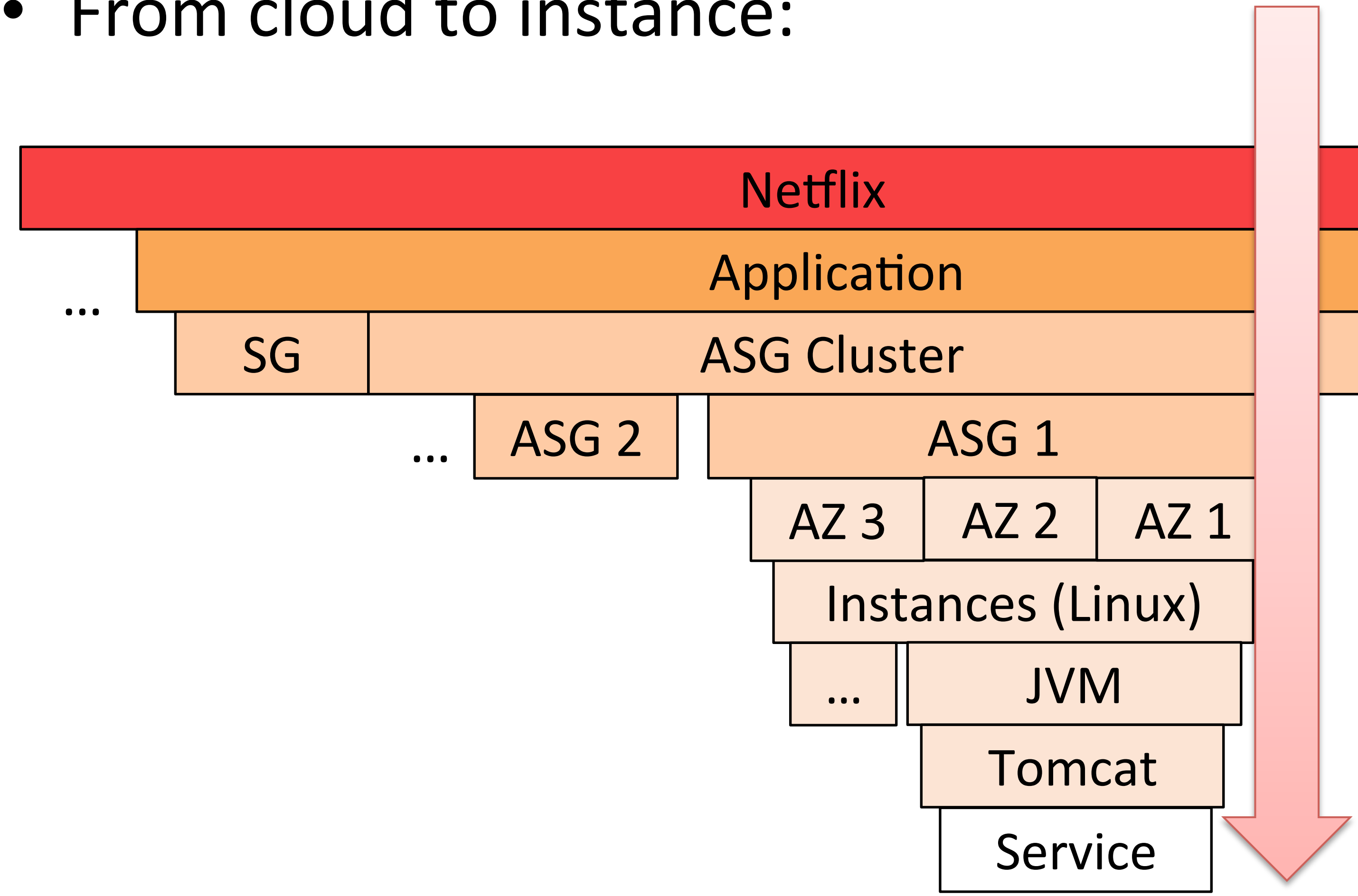
Root Cause Performance Analysis

Root Cause Performance Analysis

- Conducted when:
  - Growth becomes a cost problem
  - More instances or roll backs don't work
    - Eg: dependency issue, networking, …
  - A fix is needed for forward progress
    - "But it's faster on Linux 2.6.21 m2.xlarge!"
    - Staying on older versions for an undiagnosed (and fixable) reason prevents gains from later improvements
  - To understand scalability factors
- Identifies the origin of poor performance

Root Cause Analysis Process

- From cloud to instance:



Cloud Methodologies

- Resource Analysis
  - Any resources exhausted? CPU, disk, network
- Metric and event correlations
  - When things got bad, what else happened?
  - Correlate with distributed dependencies
- Latency Drilldowns
  - Trace origin of high latency from request down through dependencies
- USE Method
  - For every service, check: utilization, saturation, errors

Instance Methodologies

- Log Analysis
  - dmesg, GC, Apache, Tomcat, custom
- USE Method
  - For every resource, check: utilization, saturation, errors
- Micro-benchmarking
  - Test and measure components in isolation
- Drill-down analysis
  - Decompose request latency, repeat
- And other system performance methodologies

Bad Instances

- Not all issues root caused
  - “bad instance” != root cause
- Sometimes efficient to just kill “bad instances”
  - They could be a lone hardware issue, which could take days for you to analyze
- But they could also be an early warning of a global issue. If you kill them, you don’t know.

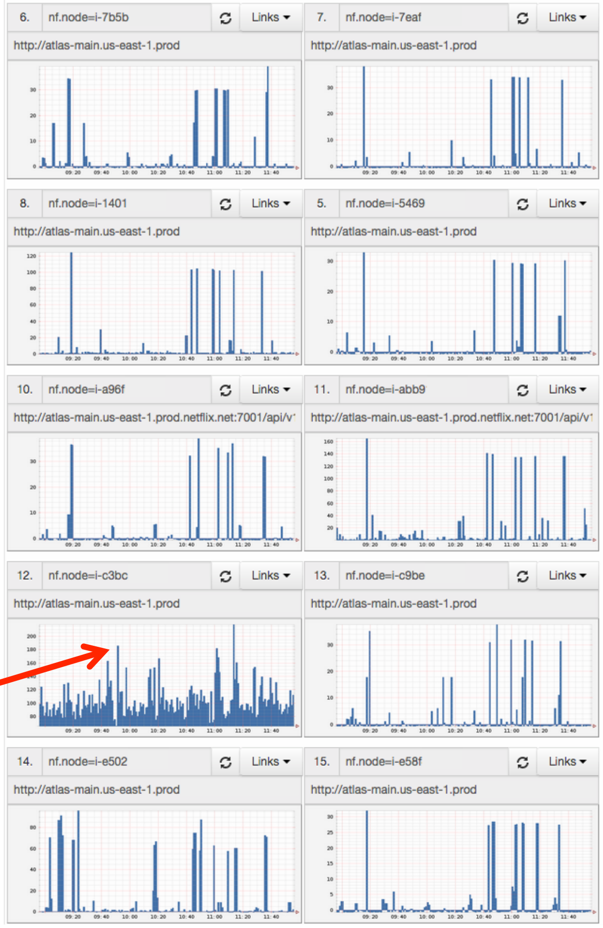
Instance   Bad Instance

Bad Instance Anti-Method

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

Bad instance

Terminate!



95th percentile latency (Atlas Exploder)

Cloud Analysis

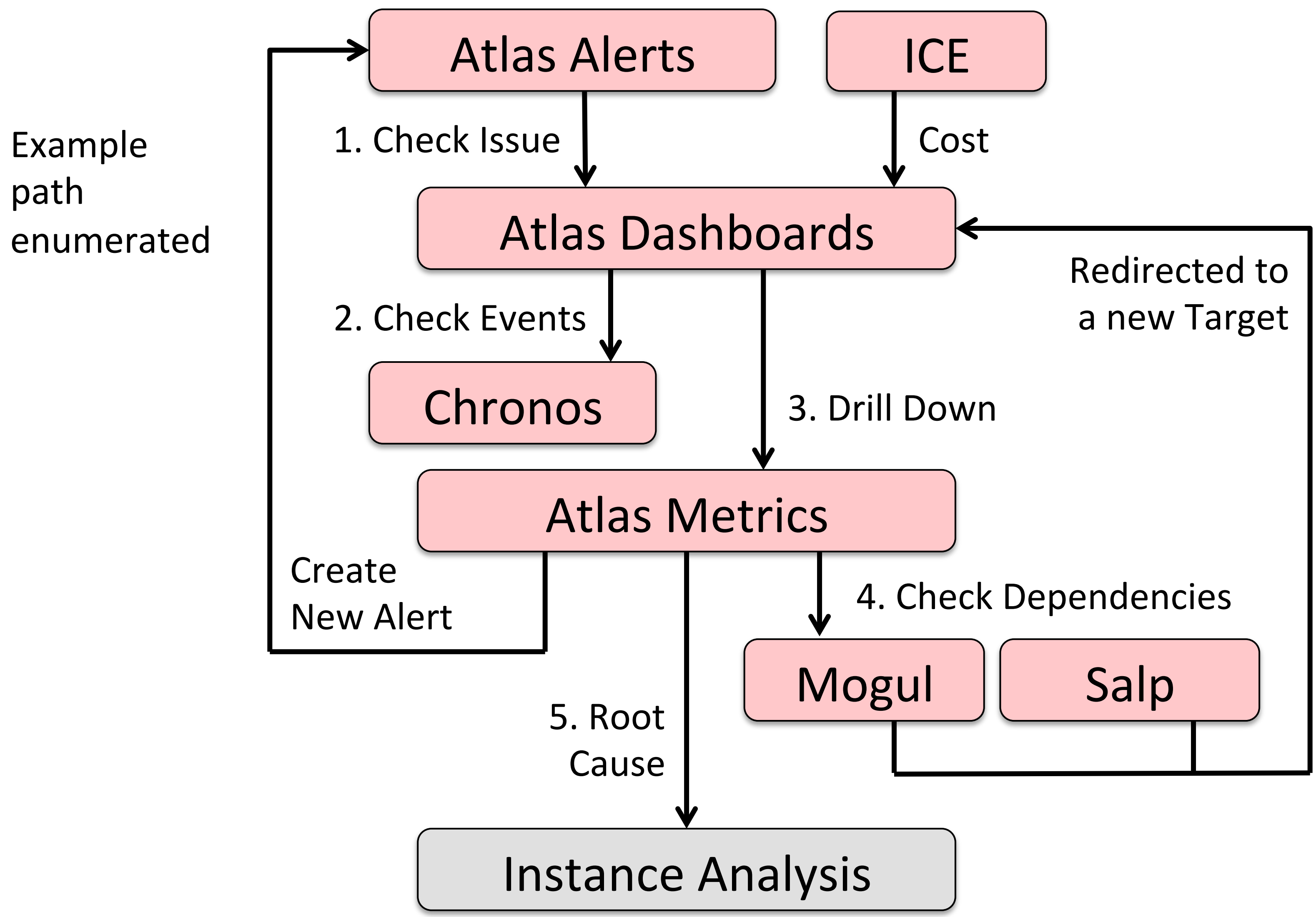
Cloud Analysis

- Cloud analysis tools made and used at Netflix include:

Tool	Purpose
Atlas	Metrics, dashboards, alerts
Chronos	Change tracking
Mogul	Metric correlation
Salp	Dependency graphing
ICE	Cloud usage dashboard

- Monitor everything: you can’t tune what you can’t see

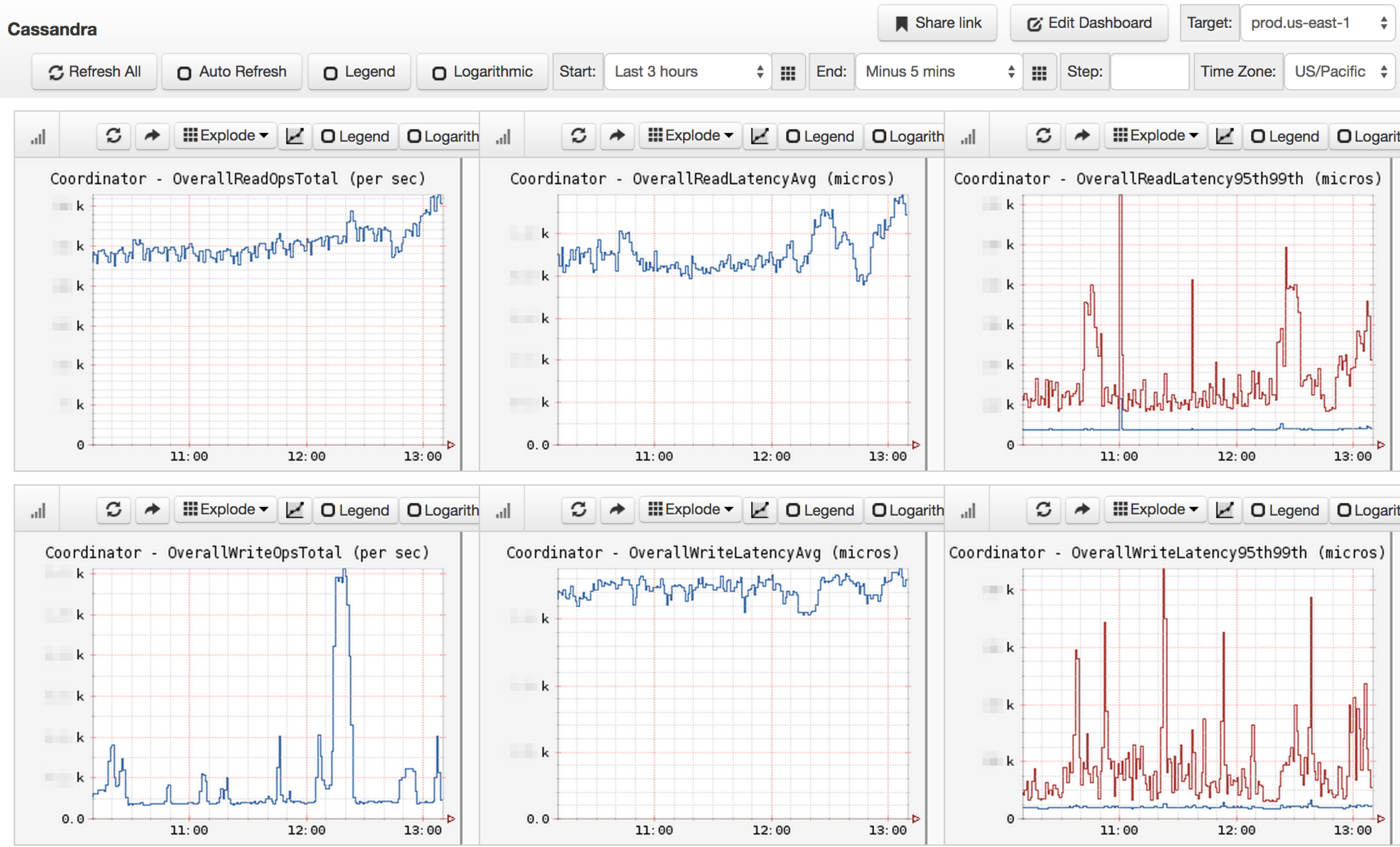
Netflix Cloud Analysis Process



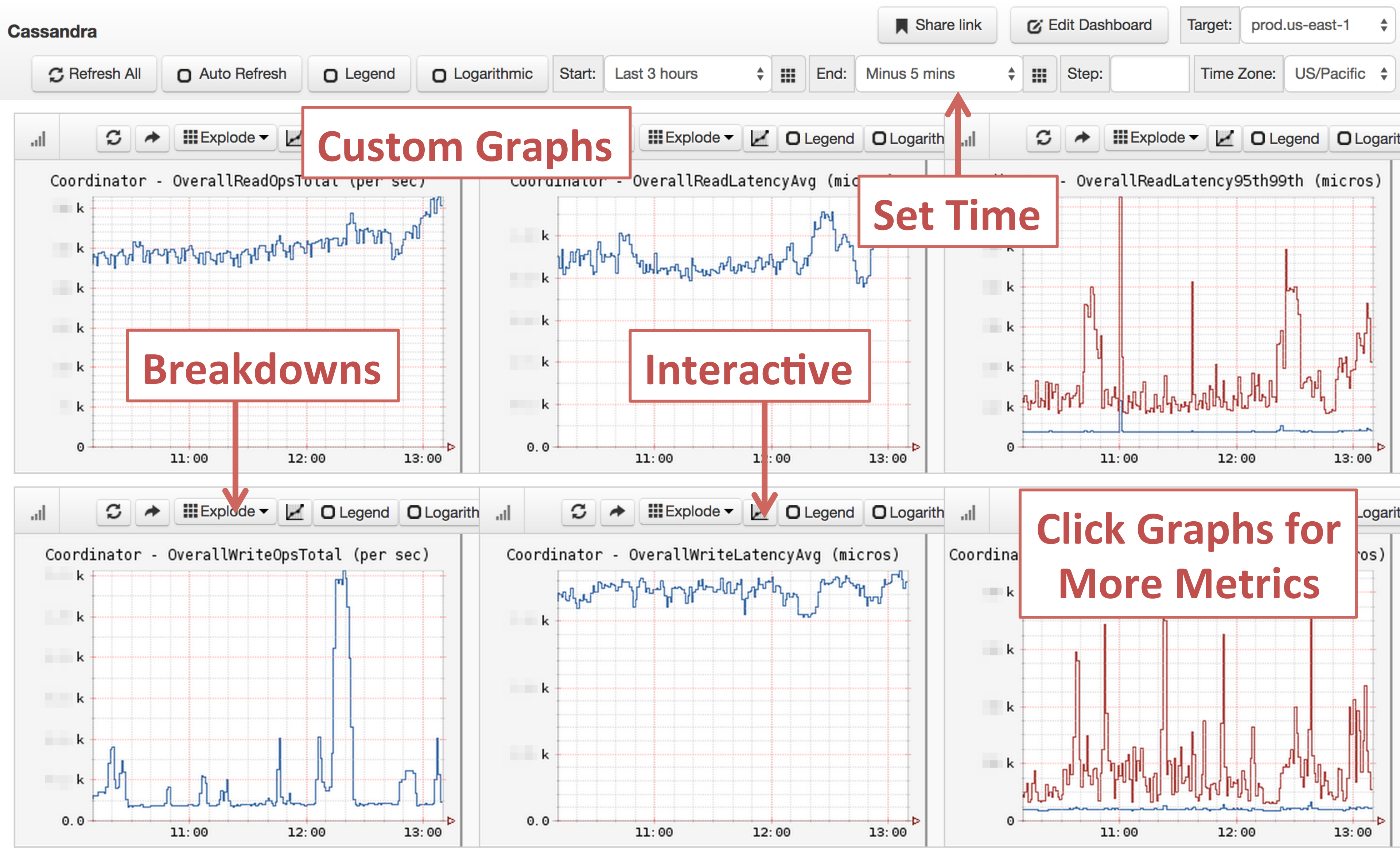
Atlas: Alerts

- Custom alerts based on the Atlas metrics
  - CPU usage, latency, instance count growth, …
- Usually email or pager
  - Can also deactivate instances, terminate, reboot
- Next step: check the dashboards

Atlas: Dashboards



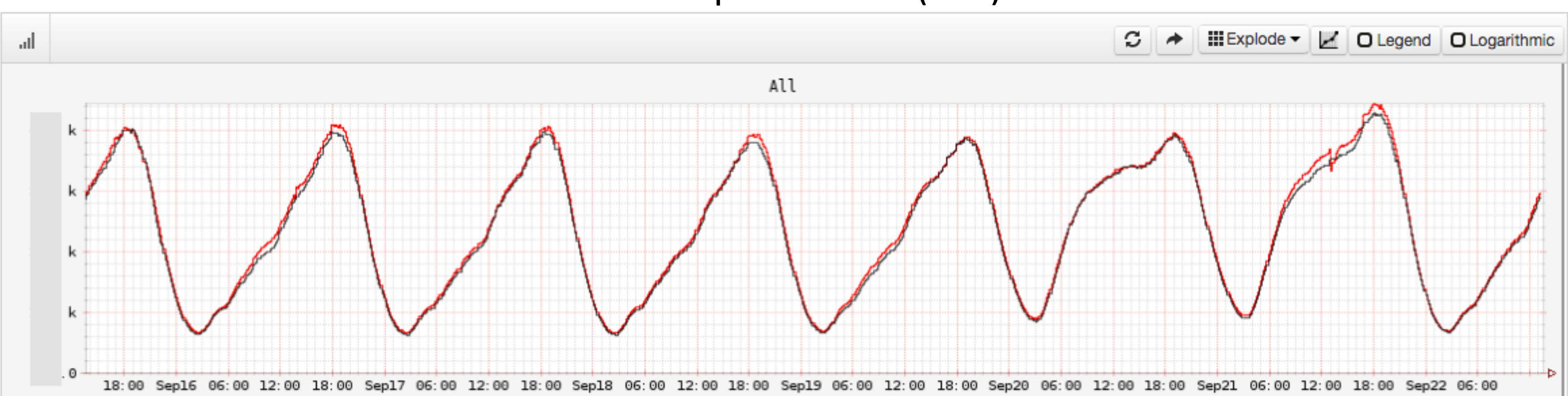
Atlas: Dashboards



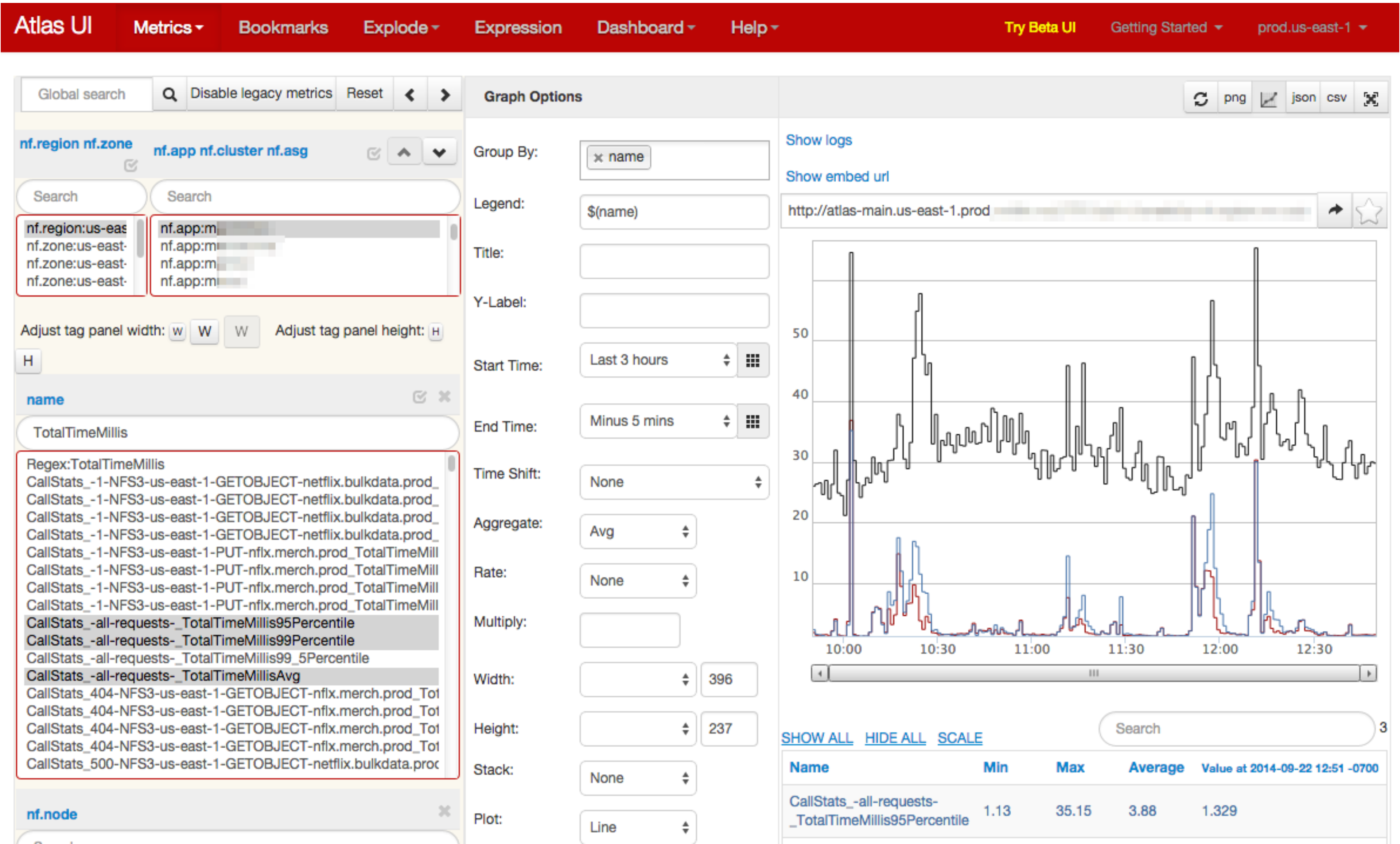
Atlas: Dashboards

- Cloud wide and per-service (all custom)
- Starting point for issue investigations
  1. Confirm and quantify issue
  2. Check historic trend
  3. Launch Atlas metrics view to drill down

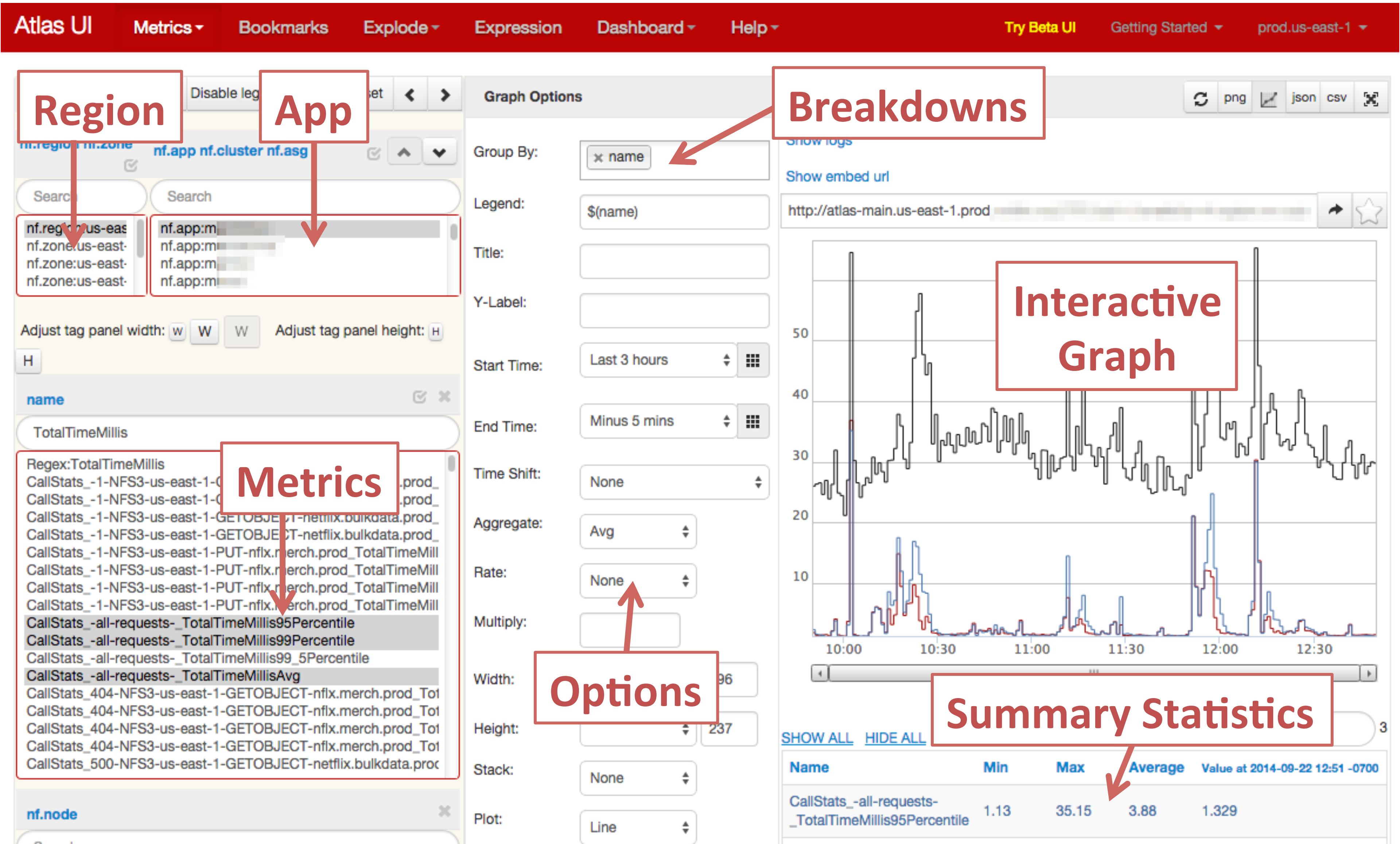
Cloud wide: streams per second (SPS) dashboard



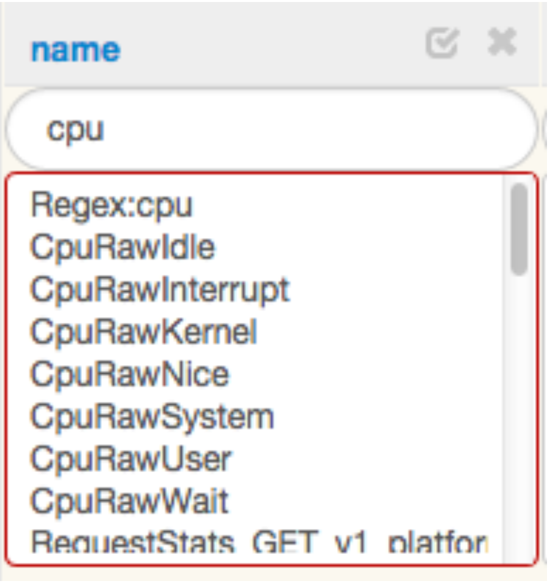
Atlas: Metrics



Atlas: Metrics

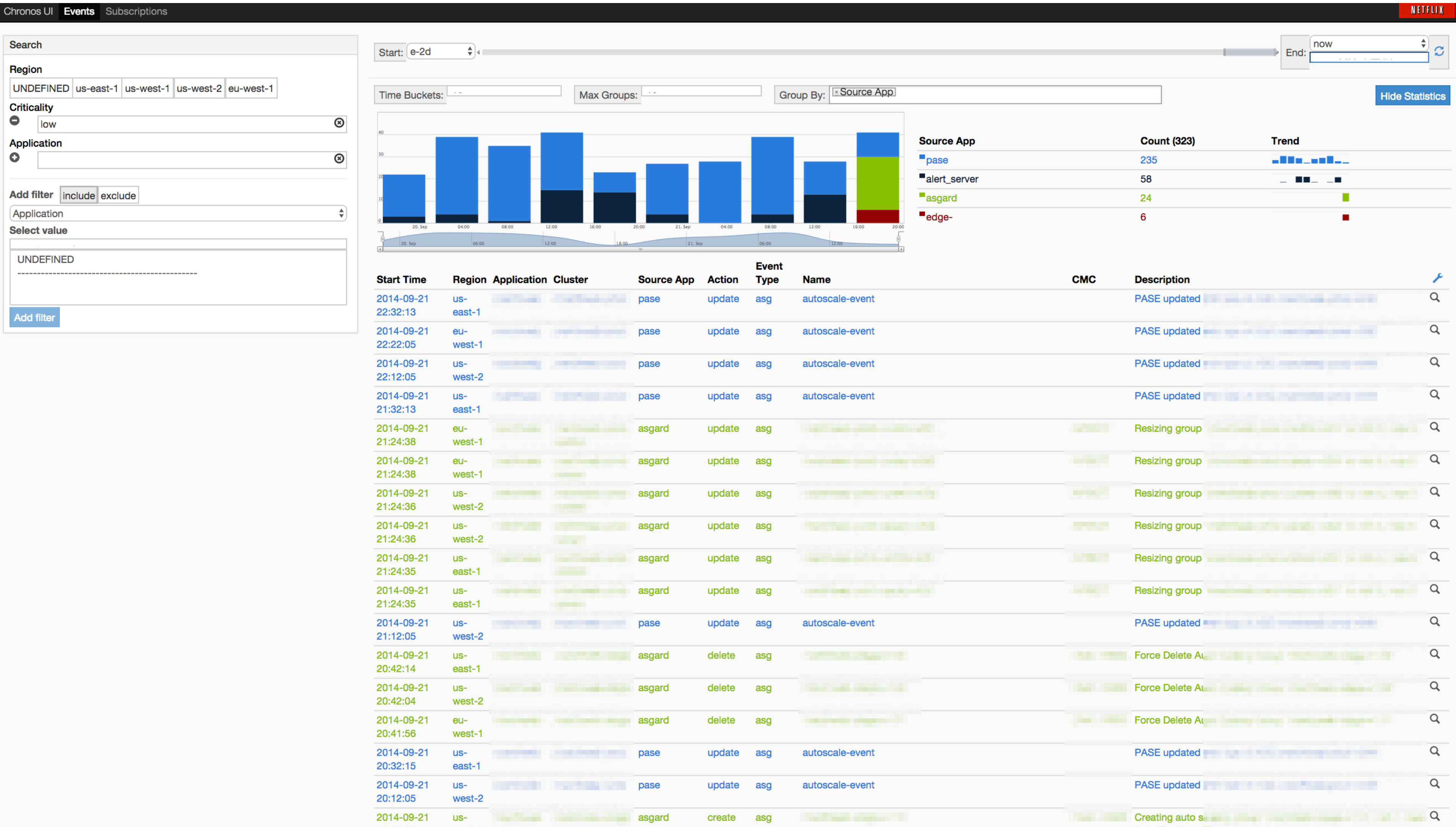


Atlas: Metrics

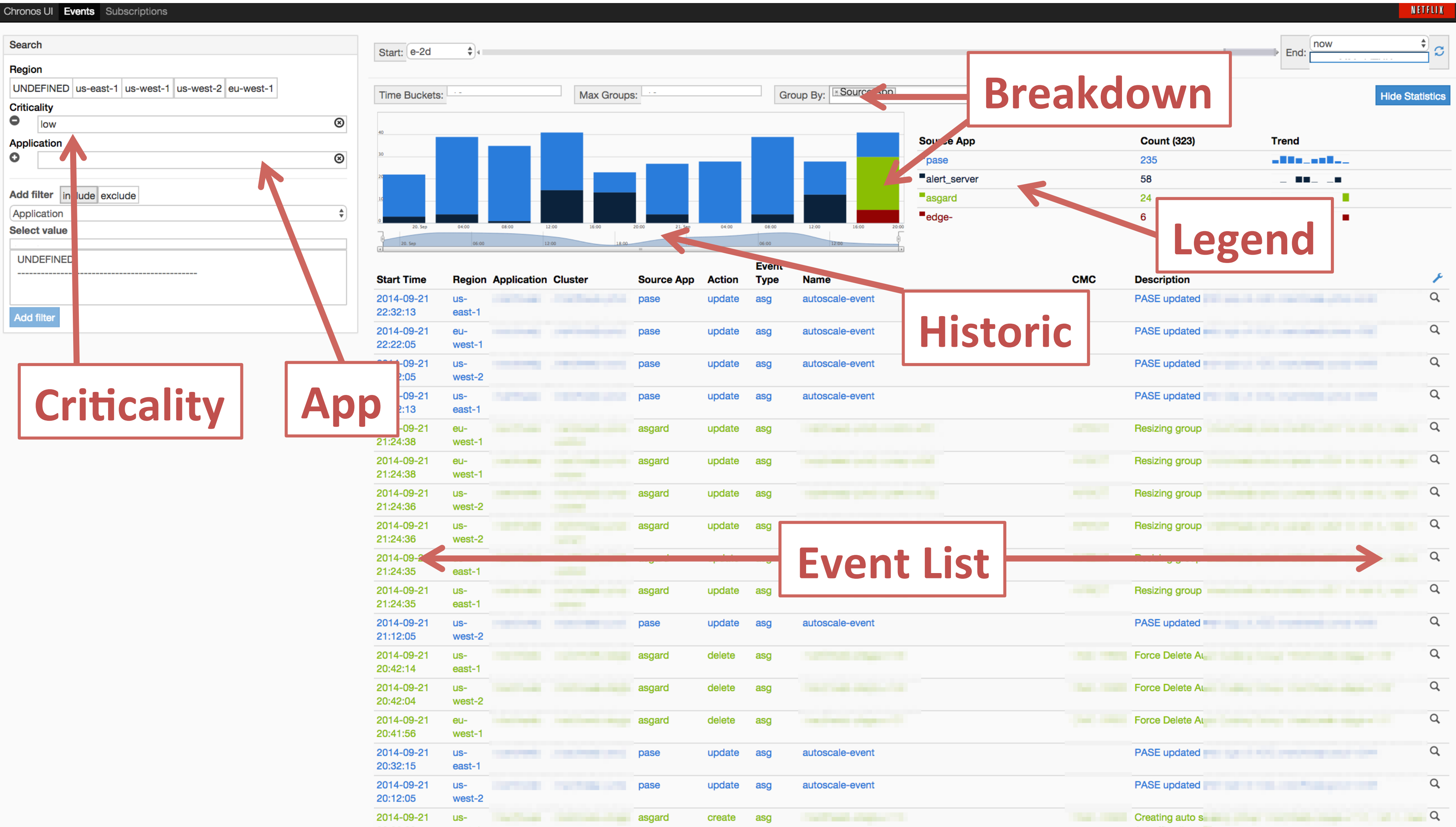


- All metrics in one system
- System metrics:
  - CPU usage, disk I/O, memory, …
- Application metrics:
  - latency percentiles, errors, …
- Filters or breakdowns by region, application, ASG, metric, instance, …
  - Quickly narrow an investigation
- URL contains session state: sharable

Chronos: Change Tracking



Chronos: Change Tracking



Chronos: Change Tracking

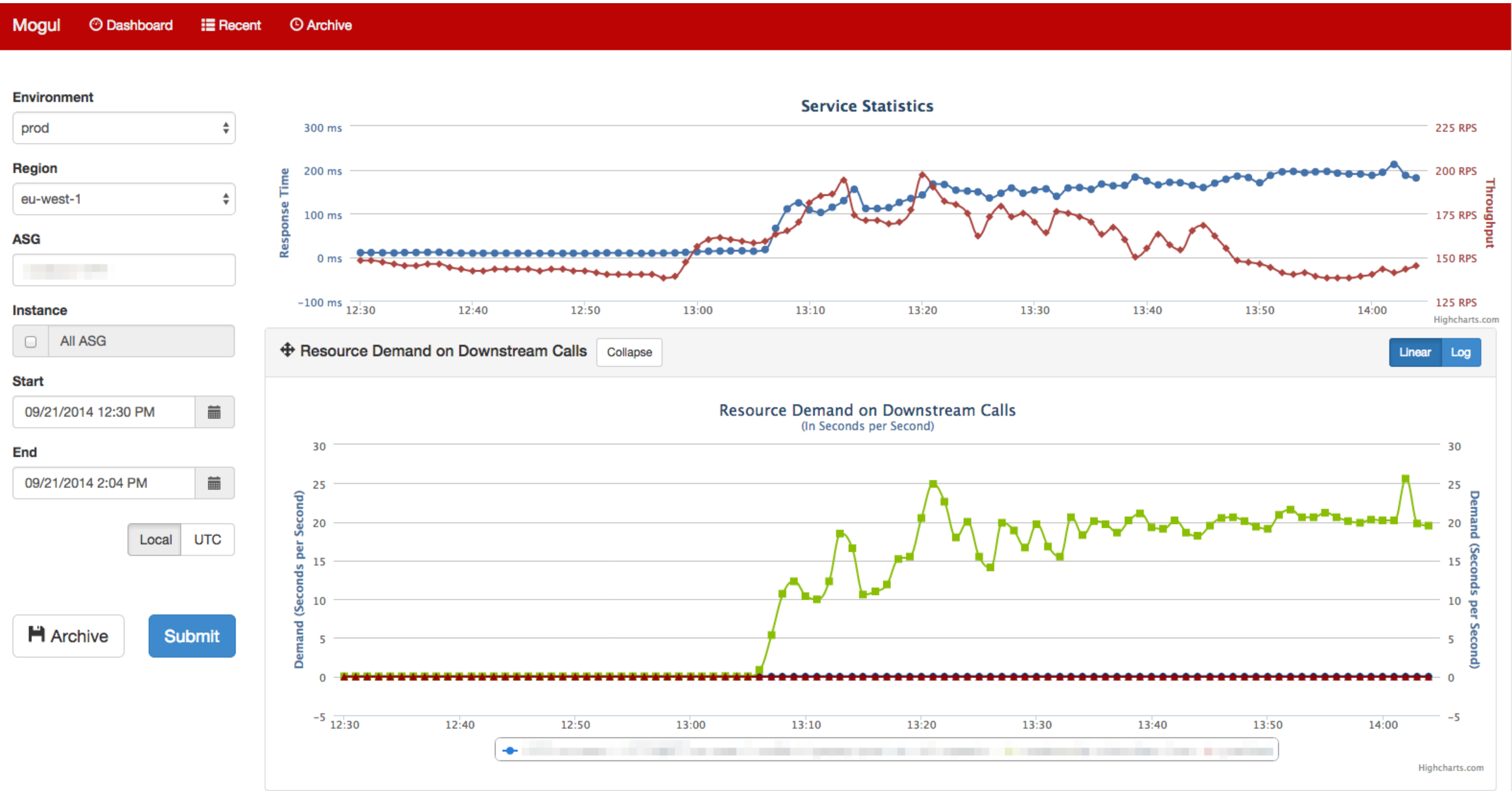
- Quickly filter uninteresting events

Start Time	Region	Application	Cluster	Source App	Action	Event Type	Name	CMC	Description
2014-09-21 21:24:38	eu-west-1	m[illegible]	[illegible]	asgard	update	asg	m[illegible]	[illegible]	Resizing group [illegible]
2014-09-21 21:24:38	eu-west-1	m[illegible]	[illegible]	asgard	Exclude update	asg	me[illegible]	[illegible]	Resizing group [illegible]

- Performance issues often coincide with changes
- The size and velocity of Netflix engineering makes Chronos crucial for communicating change

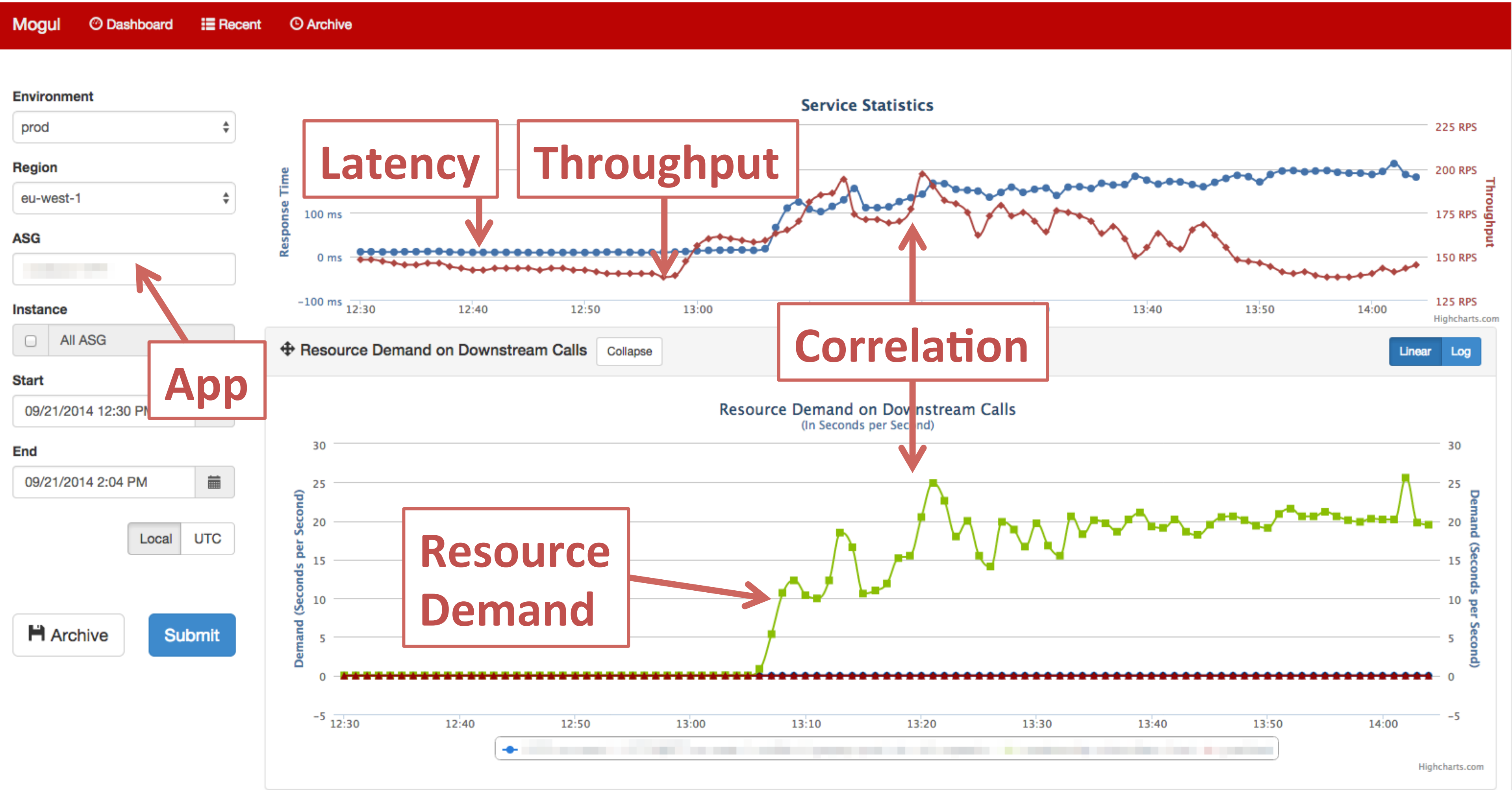
Mogul: Correlations

- Comparing performance with per-resource demand



Mogul: Correlations

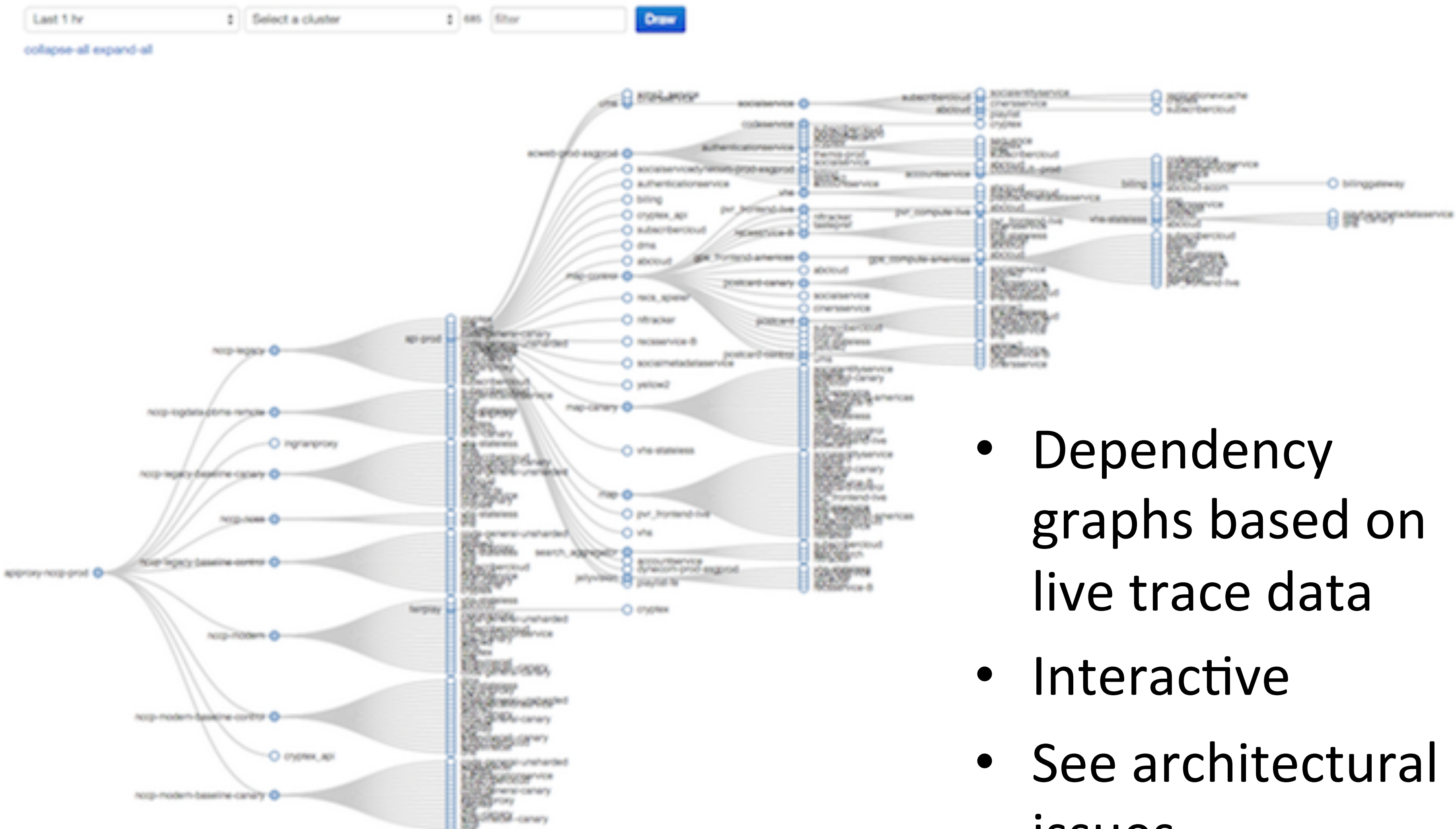
- Comparing performance with per-resource demand



Mogul: Correlations

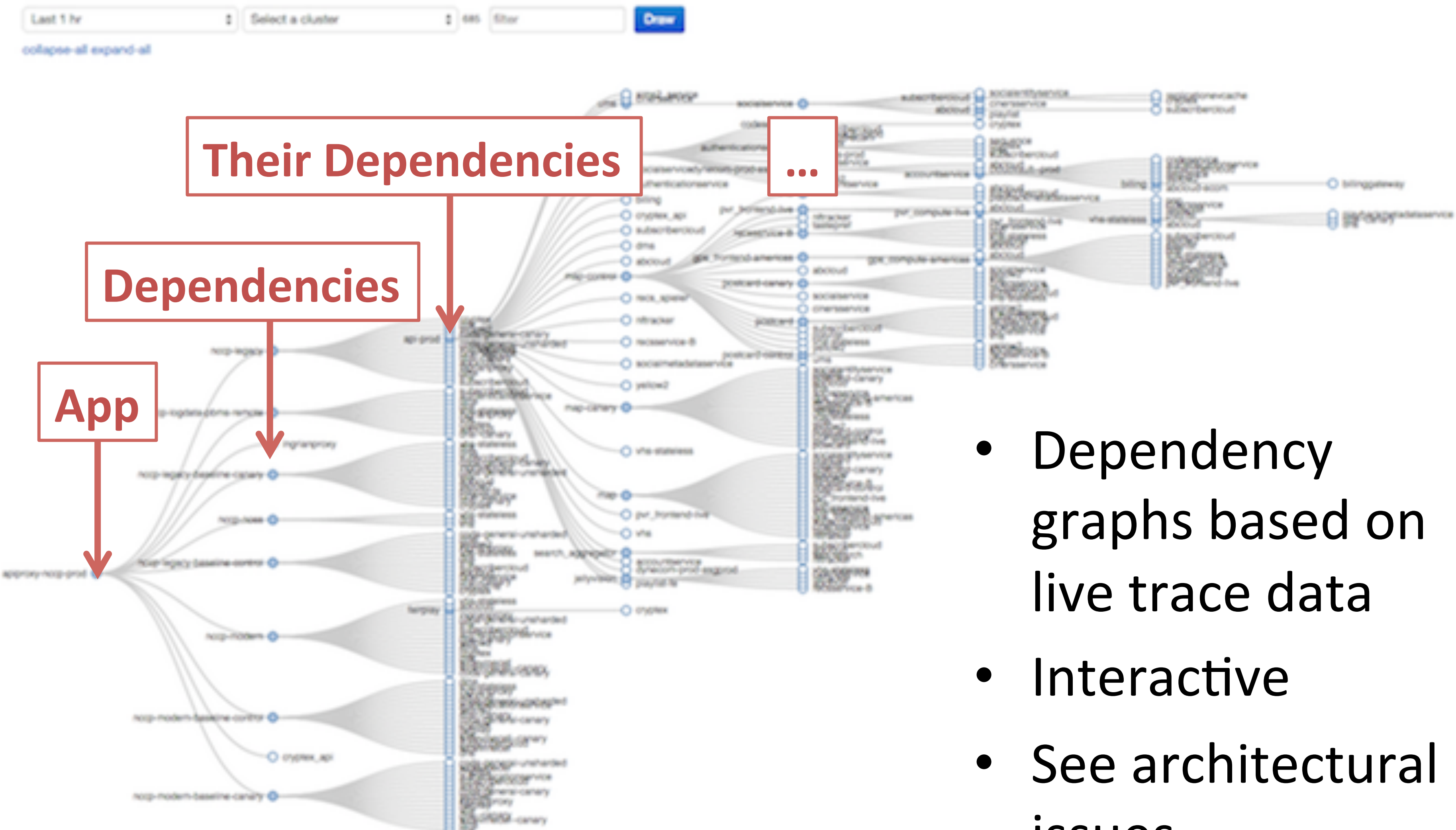
- Measures demand using Little's Law
  - D = R * X
    D = Demand (in seconds per second)
    R = Average Response Time
    X = Throughput
- Discover unexpected problem dependencies
  - That aren't on the service dashboards
- Mogul checks many other correlations
  - Weeds through thousands of application metrics, showing you the most related/interesting ones
  - (Scott/Martin should give a talk just on these)
- Bearing in mind correlation is not causation

Salp: Dependency Graphing



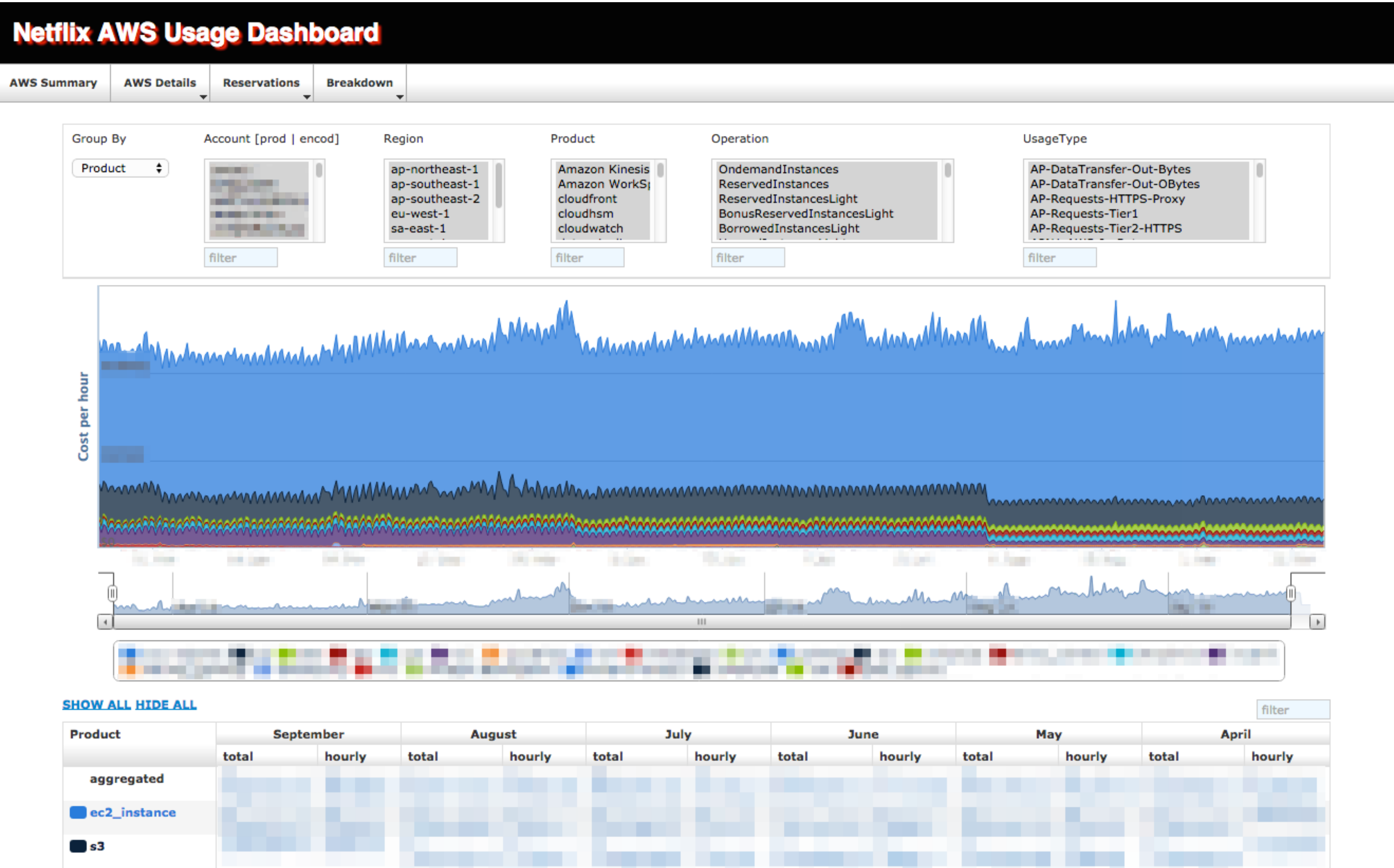
- Dependency graphs based on live trace data
- Interactive
- See architectural issues

Salp: Dependency Graphing

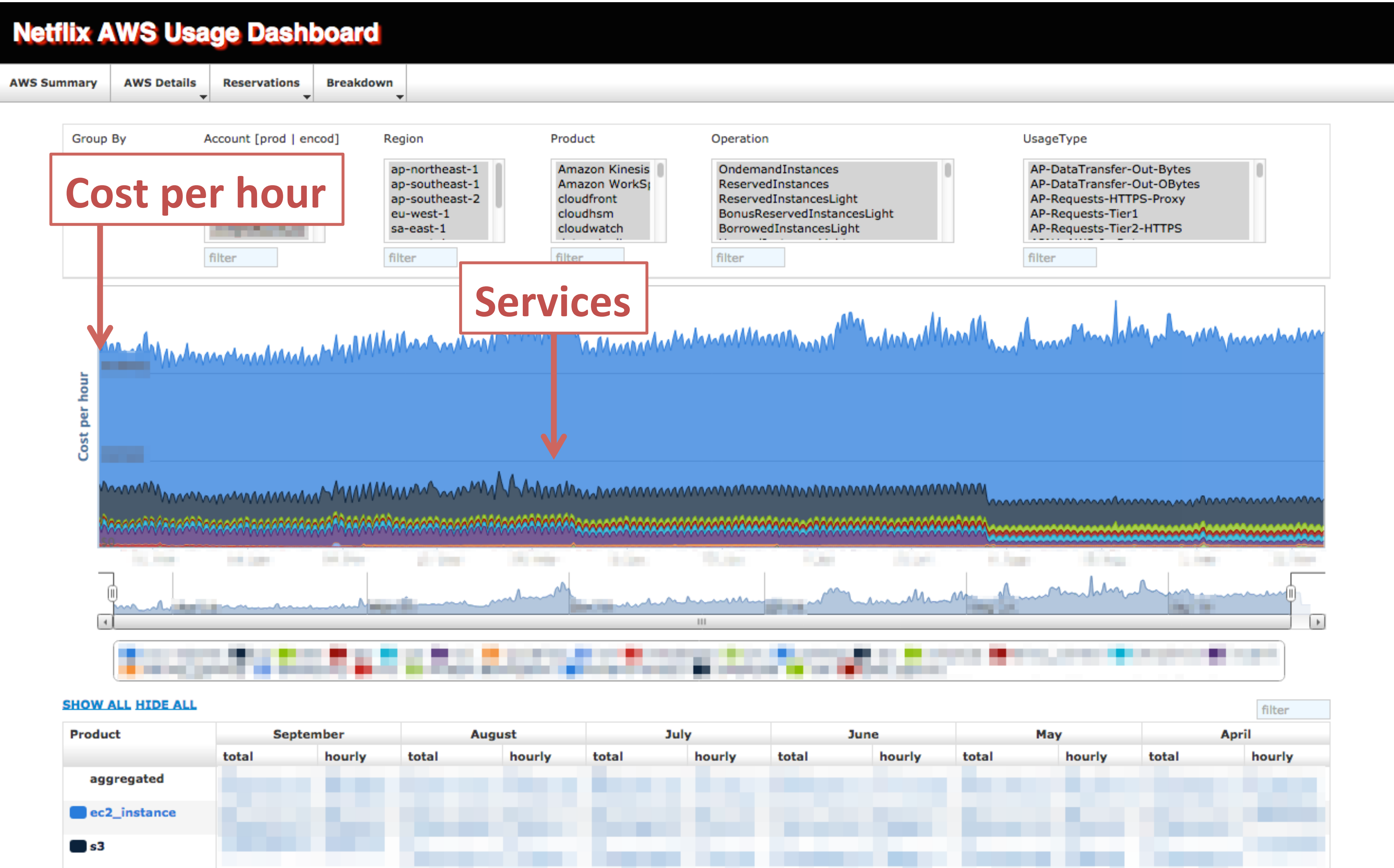


- Dependency graphs based on live trace data
- Interactive
- See architectural issues

ICE: AWS Usage



ICE: AWS Usage



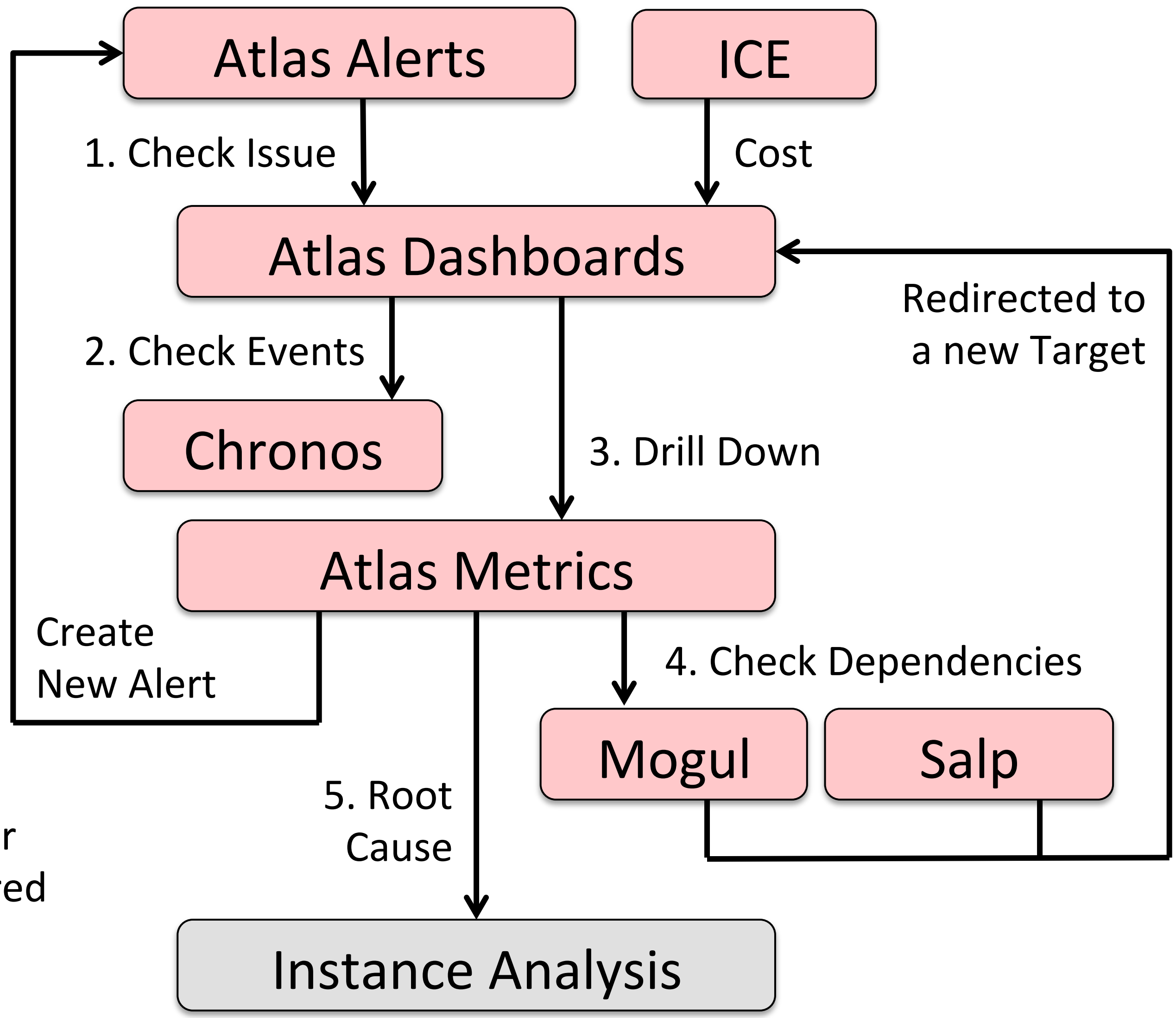
ICE: AWS Usage

- Cost per hour by AWS service, and Netflix application (service team)
  - Identify issues of slow growth
- Directs engineering effort to reduce cost

Netflix Cloud Analysis Process

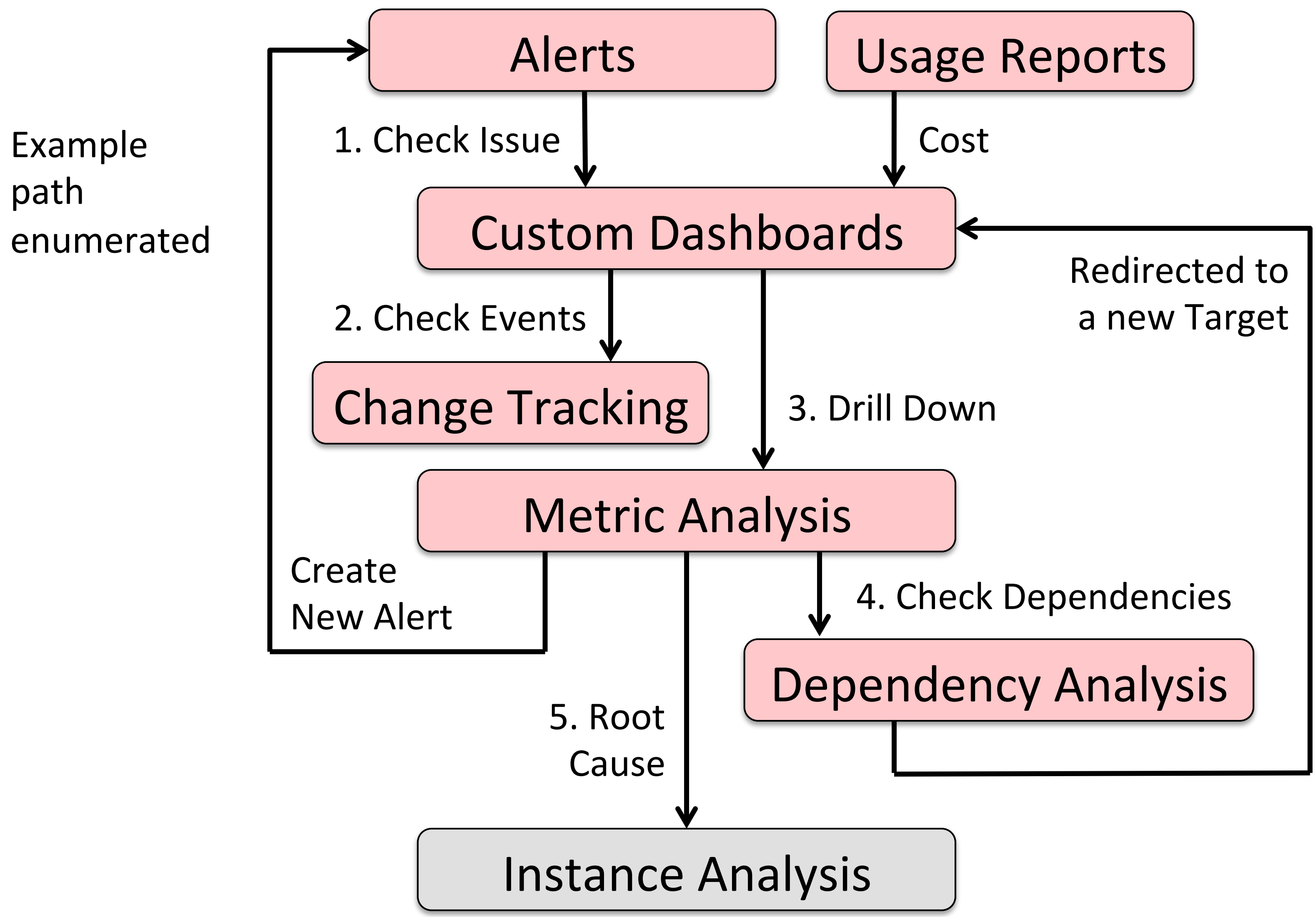
In summary…

Example path enumerated



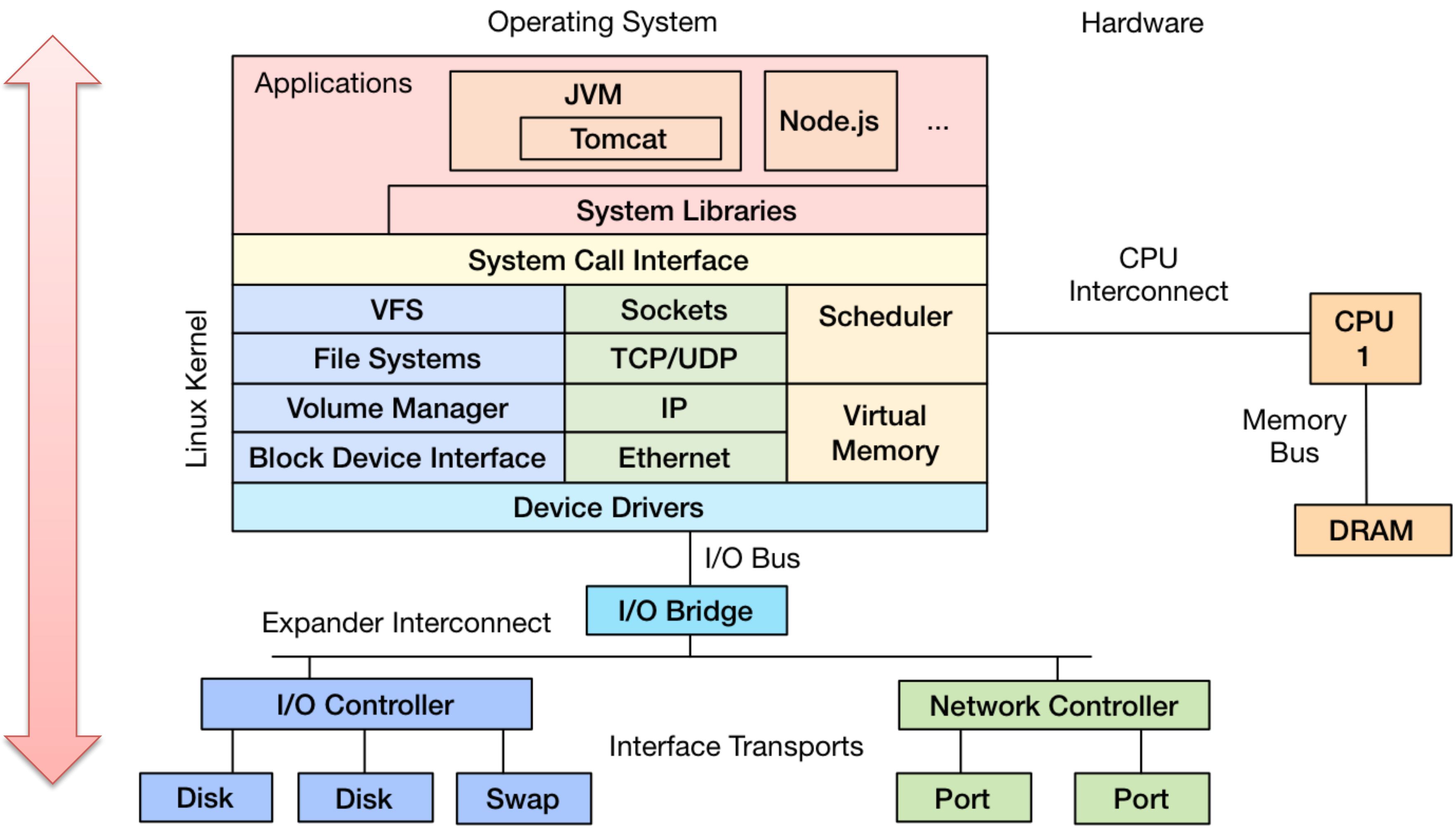
Plus some other tools not pictured

Generic Cloud Analysis Process



Instance Analysis

Instance Analysis

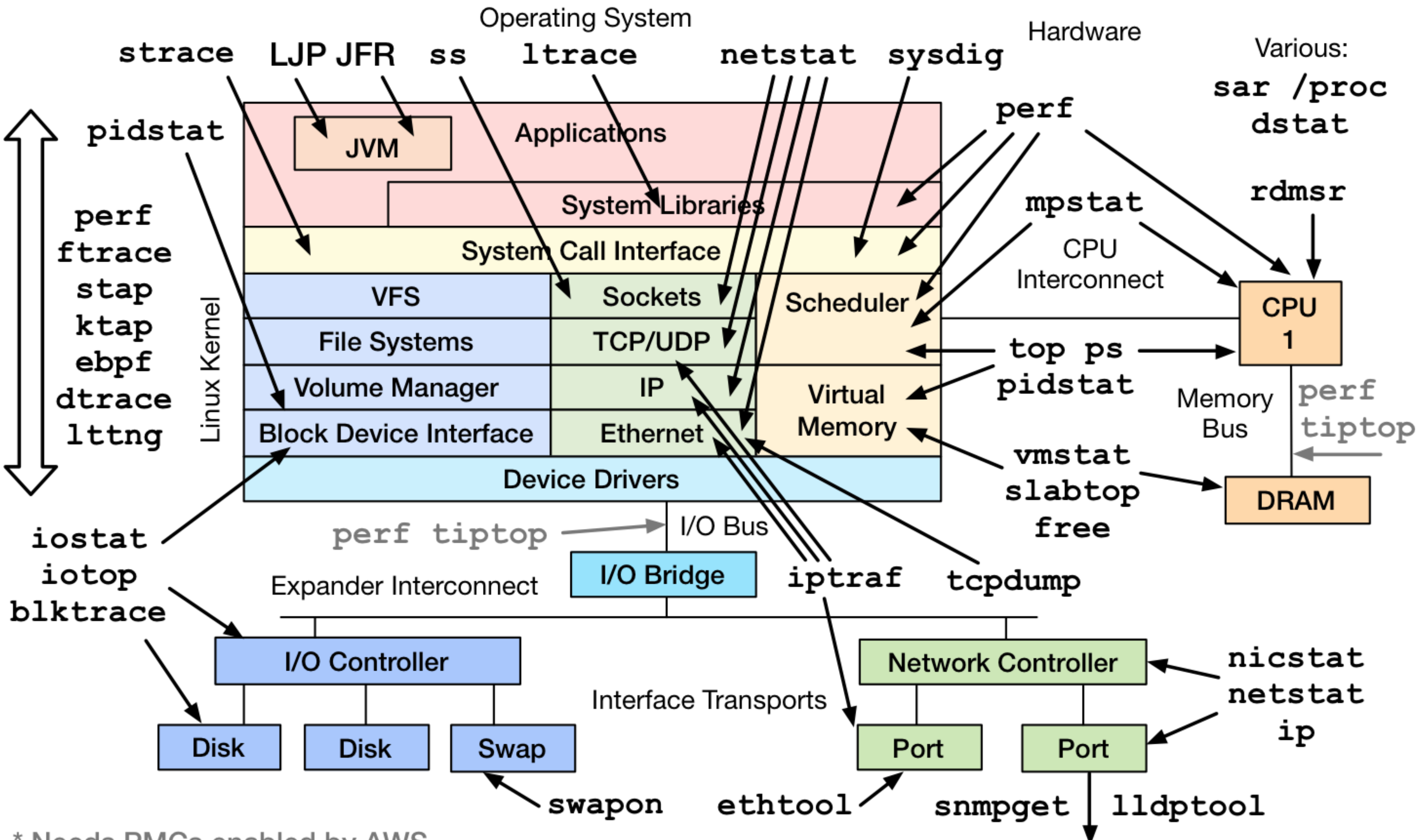


Locate, quantify, and fix performance issues anywhere in the system

Instance Tools

- Linux
  - top, ps, pidstat, vmstat, iostat, mpstat, netstat, nicstat, sar, strace, tcpdump, ss, …
- System Tracing
  - ftrace, perf_events, SystemTap
- CPU Performance Counters
  - perf_events, rdmsr
- Application Profiling
  - application logs, perf_events, Google Lightweight Java Profiler (LJP), Java Flight Recorder (JFR)

Tools in an AWS EC2 Linux Instance



* Needs PMCs enabled by AWS

Brendan Gregg 2014

Linux Performance Analysis

- vmstat, pidstat, sar, etc, used mostly normally

```
$ sar -n TCP,ETCP,DEV 1
Linux 3.2.55 (test-e4f1a80b)    08/18/2014    _x86_64_ (8 CPU)

09:10:43 PM   IFACE   rxpck/s   txpck/s    rxkB/s    txkB/s rxcmp/s txcmp/s  rxmcst/s
09:10:44 PM      lo     14.00     14.00      1.34      1.34    0.00    0.00      0.00
09:10:44 PM    eth0   4114.00   4186.00   4537.46  28513.24    0.00    0.00      0.00

09:10:43 PM  active/s passive/s    iseg/s    oseg/s
09:10:44 PM     21.00      4.00   4107.00  22511.00

09:10:43 PM  atmptf/s  estres/s retrans/s isegerr/s   orsts/s
09:10:44 PM      0.00      0.00     36.00      0.00      1.00
[…]
```

- Micro benchmarking can be used to investigate hypervisor behavior that can’t be observed directly

Instance Challenges

- Application Profiling
  – For Java, Node.js
- System Tracing
  – On Linux
- Accessing CPU Performance Counters
  – From cloud guests

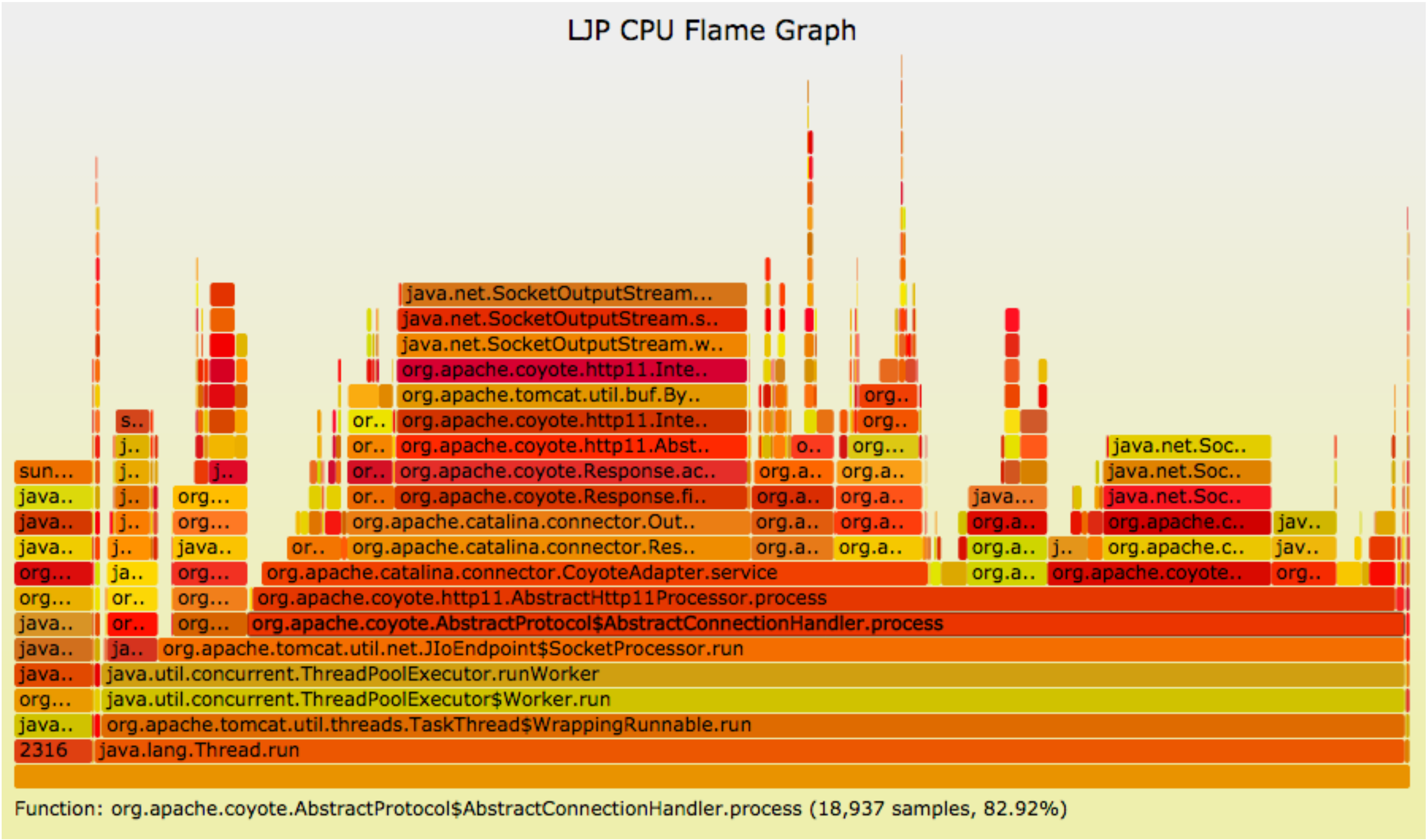
Application Profiling

- We've found many tools are inaccurate or broken
  - Eg, those based on java hprof
- Stack profiling can be problematic:
  - Linux perf_events: frame pointer for the JVM is often missing (by hotspot), breaking stacks. Also needs perf-map-agent loaded for symbol translation.
  - DTrace: jstack() also broken by missing FPs
    https://bugs.openjdk.java.net/browse/JDK-6276264, 2005
- Flame graphs are solving many performance issues. These need working stacks.

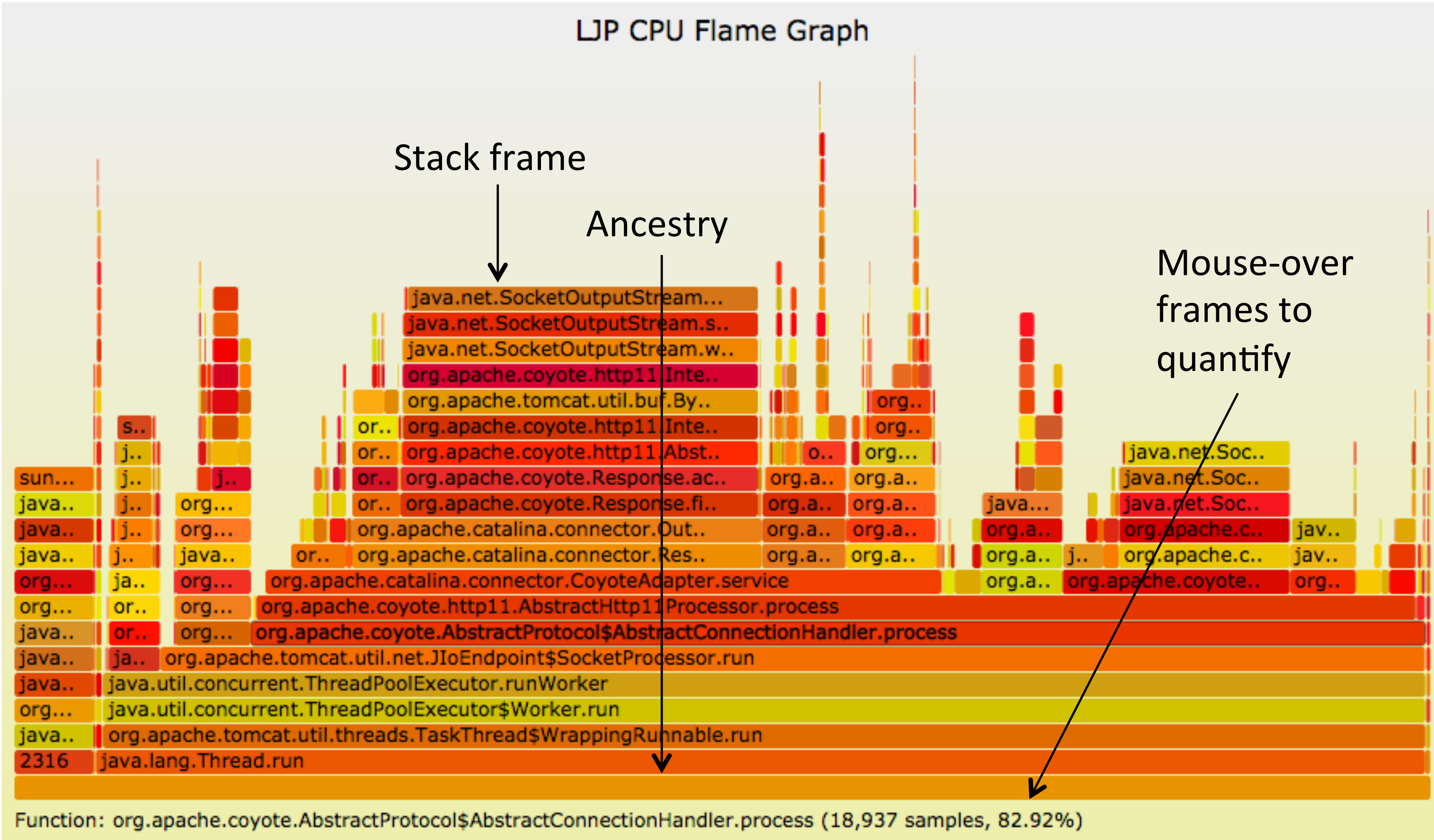
Application Profiling: Java

- Java Flight Recorder
  - CPU & memory profiling. Oracle. $$$
- Google Lightweight Java Profiler
  - Basic, open source, free, asynchronous CPU profiler
  - Uses an agent that dumps hprof-like output
    - https://code.google.com/p/lightweight-java-profiler/wiki/GettingStarted
    - http://www.brendangregg.com/blog/2014-06-12/java-flame-graphs.html
- Plus others at various times (YourKit, …)

LJP CPU Flame Graph (Java)



LJP CPU Flame Graph (Java)

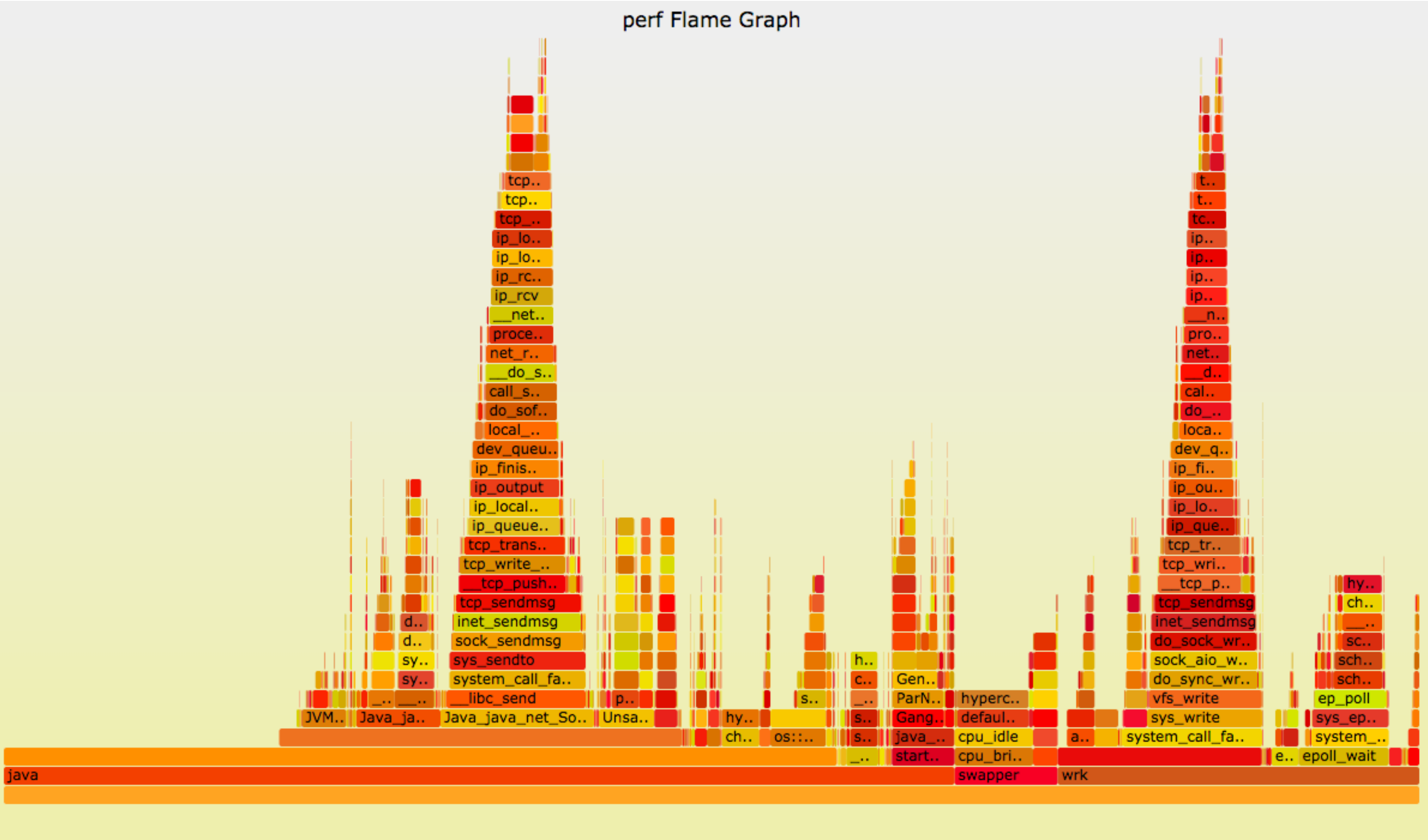


Linux System Profiling

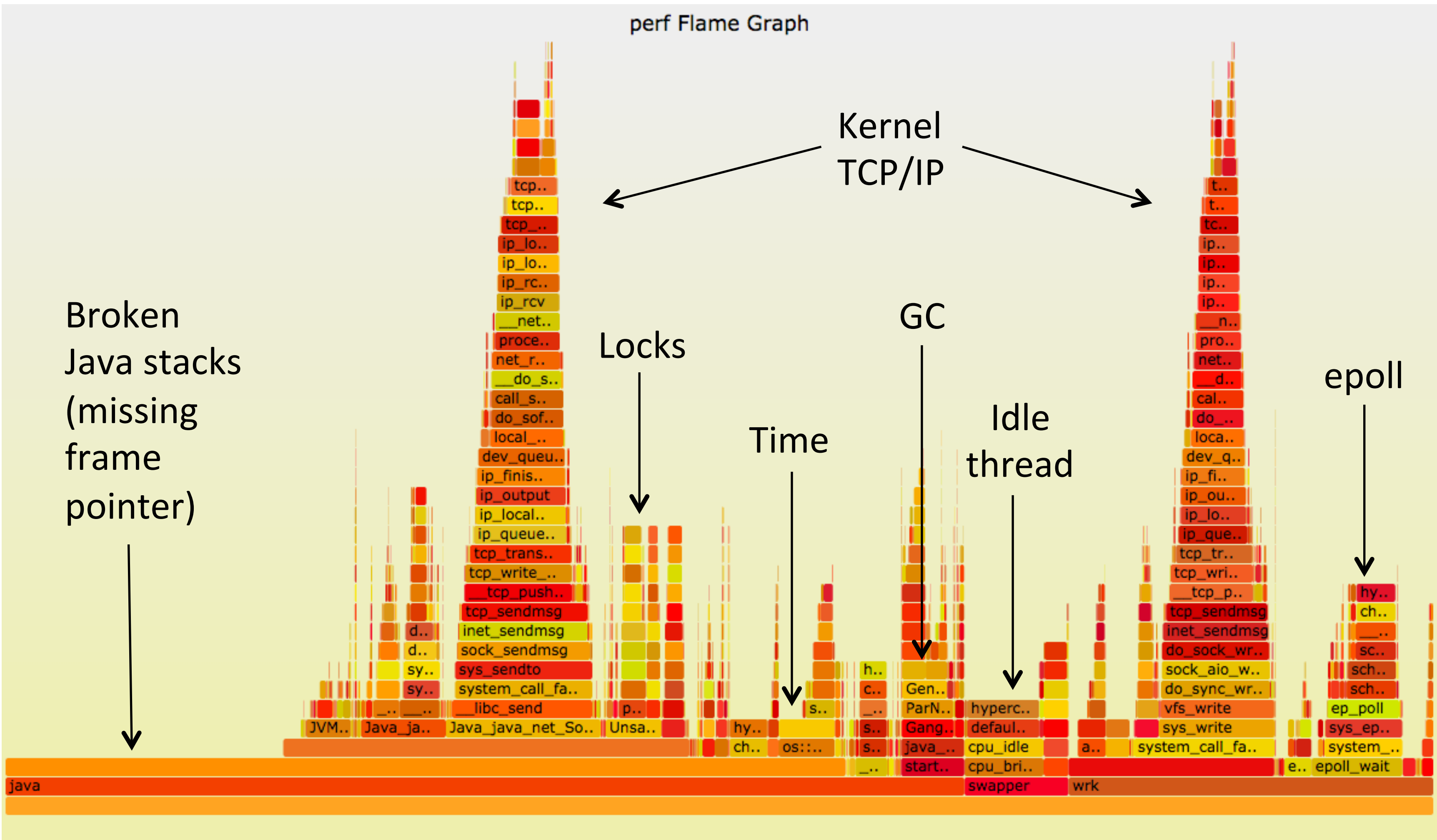
- Previous profilers only show Java CPU time
- We use perf_events (aka the "perf" command) to sample everything else:
  - JVM internals & libraries
  - The Linux kernel
  - Other apps, incl. Node.js
- perf CPU Flame graphs:

```
# git clone https://github.com/brendangregg/FlameGraph
# cd FlameGraph
# perf record -F 99 -ag -- sleep 60
# perf script | ./stackcollapse-perf.pl | ./flamegraph.pl > perf.svg
```

perf CPU Flame Graph

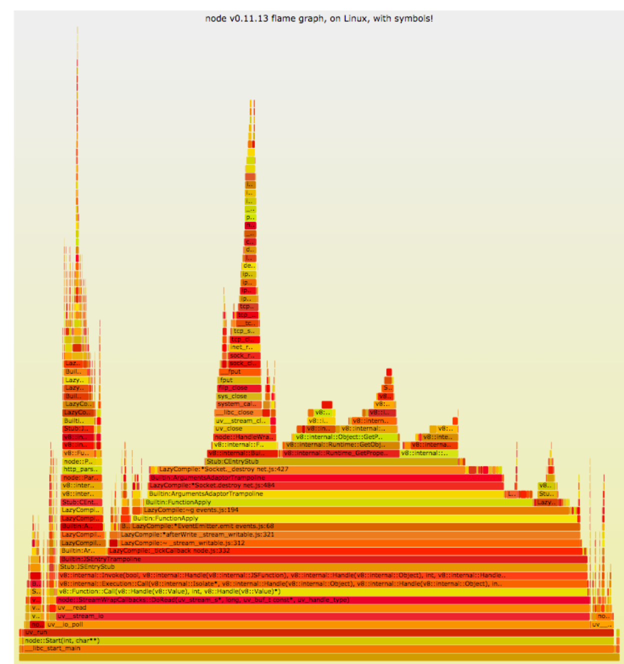


perf CPU Flame Graph



Application Profiling: Node.js

- Performance analysis on Linux a growing area
  - Eg, new postmortem tools from 2 weeks ago: https://github.com/tjfontaine/lldb-v8
- Flame graphs are possible using Linux perf_events (perf) and v8 --perf_basic_prof (node v0.11.13+)
  - Although there is currently a map growth bug; see: http://www.brendangregg.com/blog/2014-09-17/node-flame-graphs-on-linux.html
- Also do heap analysis
  - node-heapdump



Flame Graphs

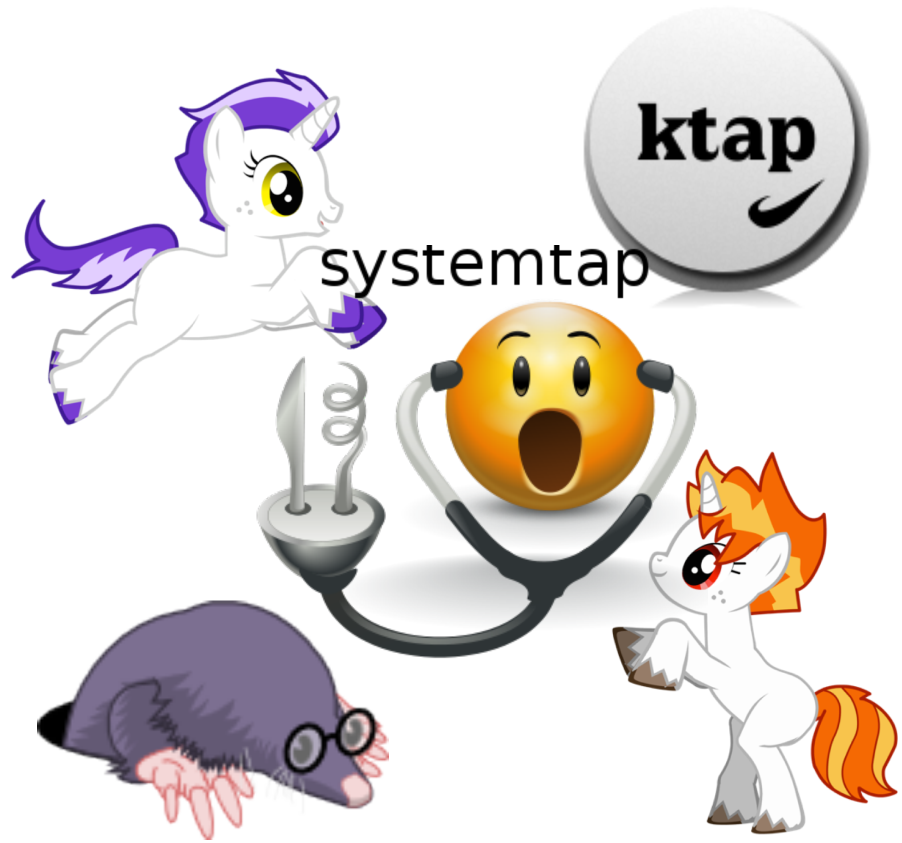
- CPU sample flame graphs solve many issues
  - We're automating their collection
  - If you aren't using them yet, you're missing out on low hanging fruit!
- Other flame graph types useful as well
  - Disk I/O, network I/O, memory events, etc
  - Any profile that includes more stacks than can be quickly read

Linux Tracing

- ... now for something more challenging

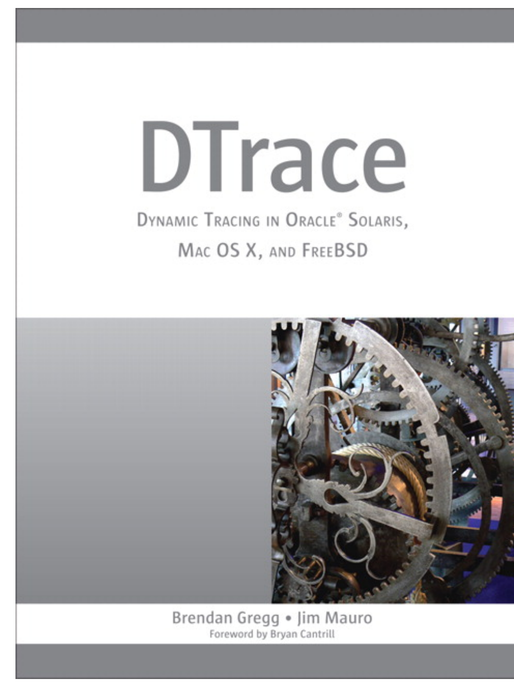
Linux Tracing

- Too many choices, and many are still in-development:
  - ftrace
  - perf_events
  - eBPF
  - SystemTap
  - ktap
  - LTTng
  - dtrace4linux
  - sysdig



Linux Tracing

- A system tracer is needed to root cause many issues: kernel, library, app
  - (There's a pretty good book covering use cases)
- DTrace is awesome, but the Linux ports are incomplete
- Linux does have has ftrace and perf_events in the kernel source, which – it turns out – can satisfy many needs already!



Linux Tracing: ftrace

- Added by Steven Rostedt and others since 2.6.27
- Already enabled on our servers (3.2+)
  - CONFIG_FTRACE, CONFIG_FUNCTION_PROFILER, …
  - Use directly via /sys/kernel/debug/tracing
- Front-end tools to aid usage: perf-tools
  - https://github.com/brendangregg/perf-tools
  - Unsupported hacks: see WARNINGs
  - Also see the trace-cmd front-end, as well as perf
- lwn.net: "Ftrace: The Hidden Light Switch"

perf-tools: iosnoop

- Block I/O (disk) events with latency:

```
# ./iosnoop –ts
Tracing block I/O. Ctrl-C to end.
STARTs         ENDs           COMM      PID  TYPE DEV    BLOCK     BYTES LATms
5982800.302061 5982800.302679 supervise 1809 W    202,1  17039600  4096   0.62
5982800.302423 5982800.302842 supervise 1809 W    202,1  17039608  4096   0.42
5982800.304962 5982800.305446 supervise 1801 W    202,1  17039616  4096   0.48
5982800.305250 5982800.305676 supervise 1801 W    202,1  17039624  4096   0.43
[…]
```

```
# ./iosnoop –h
USAGE: iosnoop [-hQst] [-d device] [-i iotype] [-p PID] [-n name] [duration]
                 -d device       # device string (eg, "202,1)
                 -i iotype       # match type (eg, '*R*' for all reads)
                 -n name         # process name to match on I/O issue
                 -p PID          # PID to match on I/O issue
                 -Q              # include queueing time in LATms
                 -s              # include start time of I/O (s)
                 -t              # include completion time of I/O (s)
                 -h              # this usage message
                 duration        # duration seconds, and use buffers
[…]
```

perf-tools: iolatency

- Block I/O (disk) latency distributions:

```
# ./iolatency
Tracing block I/O. Output every 1 seconds. Ctrl-C to end.

  >=(ms) .. <(ms)   : I/O      |Distribution                          |
       0 -> 1       : 2104     |######################################|
       1 -> 2       : 280      |######                                |
       2 -> 4       : 2        |#                                     |
       4 -> 8       : 0        |                                      |
       8 -> 16      : 202      |####                                  |

  >=(ms) .. <(ms)   : I/O      |Distribution                          |
       0 -> 1       : 1144     |######################################|
       1 -> 2       : 267      |#########                             |
       2 -> 4       : 10       |#                                     |
       4 -> 8       : 5        |#                                     |
       8 -> 16      : 248      |#########                             |
      16 -> 32      : 601      |####################                  |
      32 -> 64      : 117      |####                                  |
[…]
```

perf-tools: opensnoop

- Trace open() syscalls showing filenames:

```
# ./opensnoop -t
Tracing open()s. Ctrl-C to end.
TIMEs              COMM          PID     FD FILE
4345768.332626     postgres      23886   0x8 /proc/self/oom_adj
4345768.333923     postgres      23886   0x5 global/pg_filenode.map
4345768.333971     postgres      23886   0x5 global/pg_internal.init
4345768.334813     postgres      23886   0x5 base/16384/PG_VERSION
4345768.334877     postgres      23886   0x5 base/16384/pg_filenode.map
4345768.334891     postgres      23886   0x5 base/16384/pg_internal.init
4345768.335821     postgres      23886   0x5 base/16384/11725
4345768.347911     svstat        24649   0x4 supervise/ok
4345768.347921     svstat        24649   0x4 supervise/status
4345768.350340     stat          24651   0x3 /etc/ld.so.cache
4345768.350372     stat          24651   0x3 /lib/x86_64-linux-gnu/libselinux…
4345768.350460     stat          24651   0x3 /lib/x86_64-linux-gnu/libc.so.6
4345768.350526     stat          24651   0x3 /lib/x86_64-linux-gnu/libdl.so.2
4345768.350981     stat          24651   0x3 /proc/filesystems
4345768.351182     stat          24651   0x3 /etc/nsswitch.conf
[…]
```

perf-tools: funcgraph

- Trace a graph of kernel code flow:

```
# ./funcgraph -Htp 5363 vfs_read
Tracing "vfs_read" for PID 5363... Ctrl-C to end.
# tracer: function_graph
#
#     TIME        CPU  DURATION                  FUNCTION CALLS
#      |          |     |   |                     |   |   |   |
4346366.073832 |   0)               |  vfs_read() {
4346366.073834 |   0)               |    rw_verify_area() {
4346366.073834 |   0)               |      security_file_permission() {
4346366.073834 |   0)               |        apparmor_file_permission() {
4346366.073835 |   0)   0.153 us    |          common_file_perm();
4346366.073836 |   0)   0.947 us    |        }
4346366.073836 |   0)   0.066 us    |        __fsnotify_parent();
4346366.073836 |   0)   0.080 us    |        fsnotify();
4346366.073837 |   0)   2.174 us    |      }
4346366.073837 |   0)   2.656 us    |    }
4346366.073837 |   0)               |    tty_read() {
4346366.073837 |   0)   0.060 us    |      tty_paranoia_check();
[…]
```

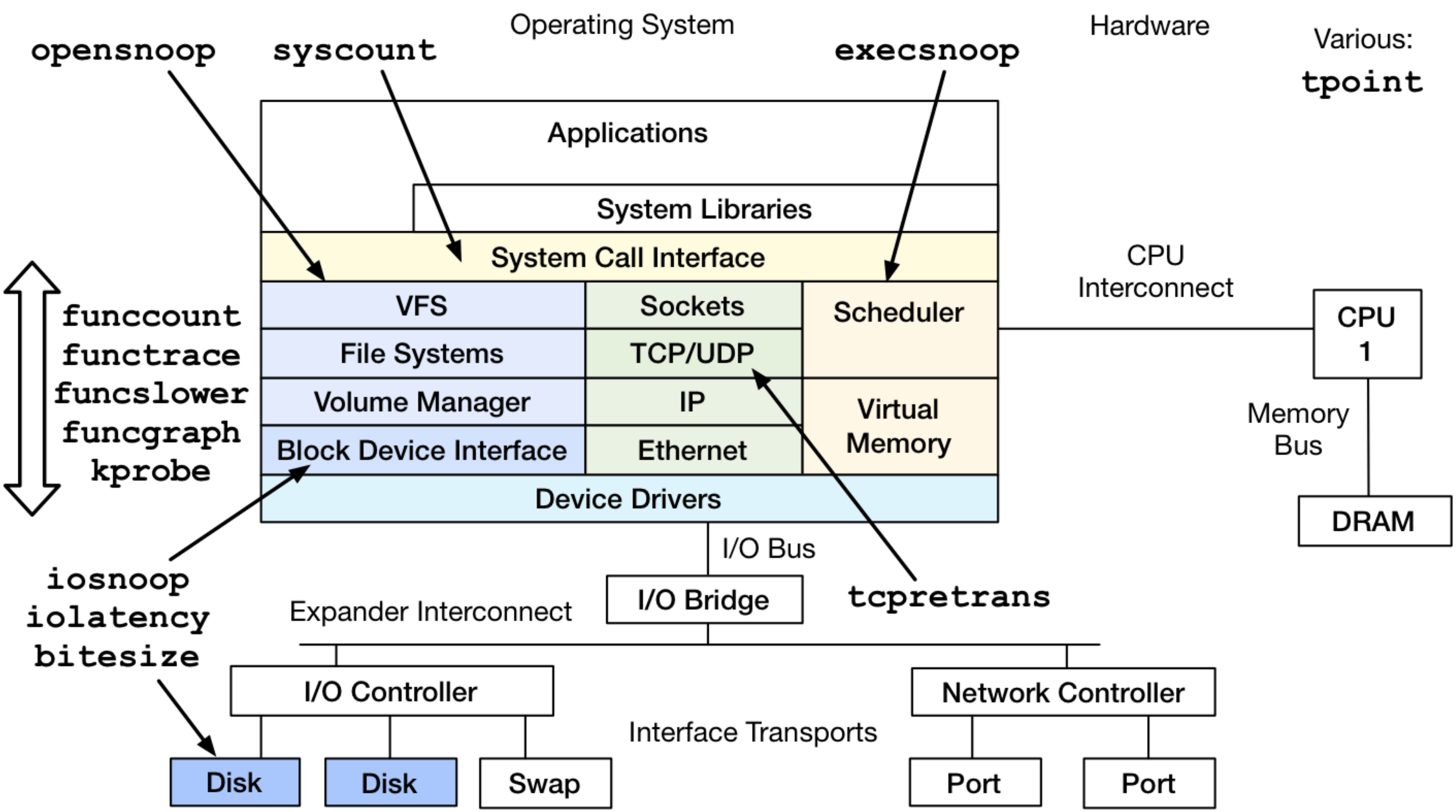
perf-tools: kprobe

- Dynamically trace a kernel function call or return, with variables, and in-kernel filtering:

```
# ./kprobe 'p:open do_sys_open filename=+0(%si):string' 'filename ~ "*stat"'
Tracing kprobe myopen. Ctrl-C to end.
        postgres-1172  [000] d... 6594028.787166: open: (do_sys_open
+0x0/0x220) filename="pg_stat_tmp/pgstat.stat"
        postgres-1172  [001] d... 6594028.797410: open: (do_sys_open
+0x0/0x220) filename="pg_stat_tmp/pgstat.stat"
        postgres-1172  [001] d... 6594028.797467: open: (do_sys_open
+0x0/0x220) filename="pg_stat_tmp/pgstat.stat"
^C
Ending tracing...
```

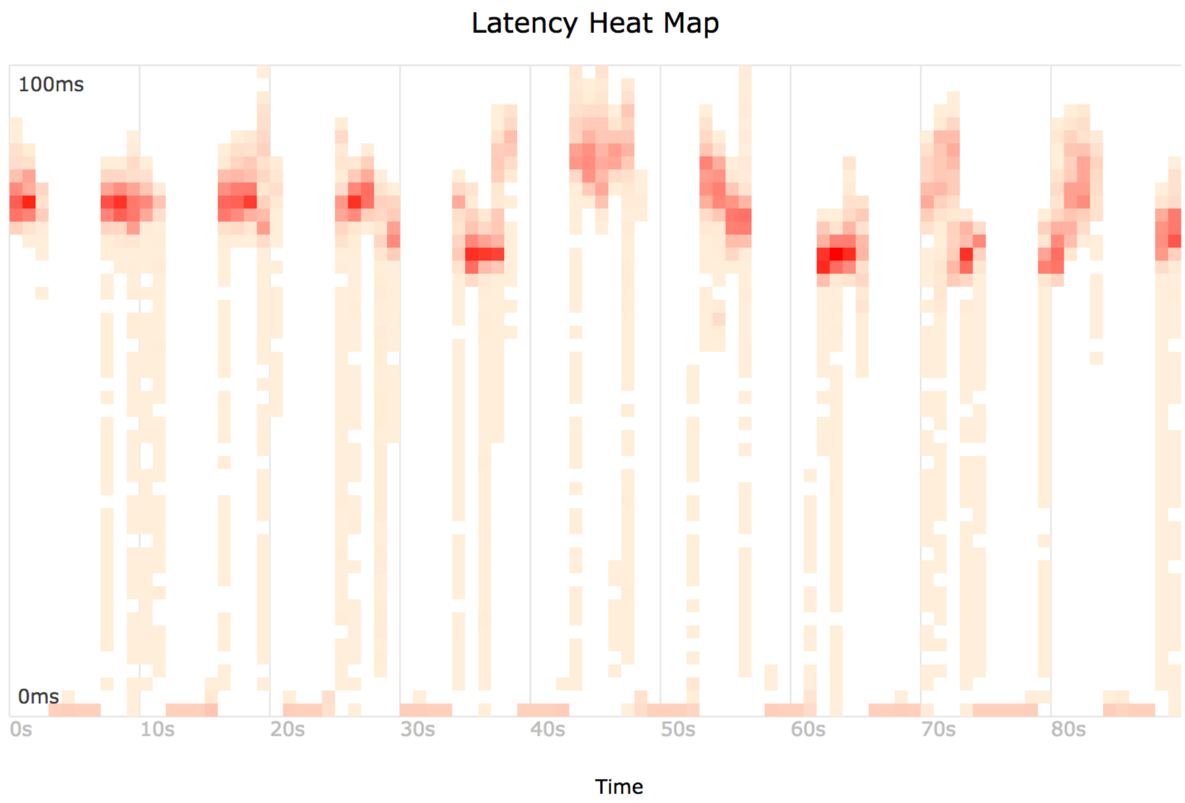
- Add -s for stack traces; -p for PID filter in-kernel.
- Quickly confirm kernel behavior; eg: did a tunable take effect?

perf-tools (so far…)



Heat Maps

- ftrace or perf_events for tracing disk I/O and other latencies as a heat map:



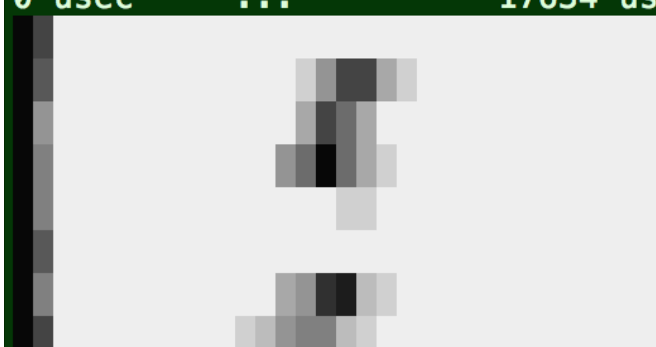
Other Tracing Options

- SystemTap
  - The most powerful of the system tracers
  - We'll use it as a last resort: deep custom tracing
  - I've historically had issues with panics and freezes
    - Still present in the latest version?
    - The Netflix fault tolerant architecture makes panics much less of a problem (that was the panic monkey)
- Instance canaries with DTrace are possible too
  - OmniOS
  - FreeBSD

Linux Tracing Future

- ftrace + perf_events cover much, but not custom in-kernel aggregations
- eBPF may provide this missing feature
  - eg, in-kernel latency heat map (showing bimodal):

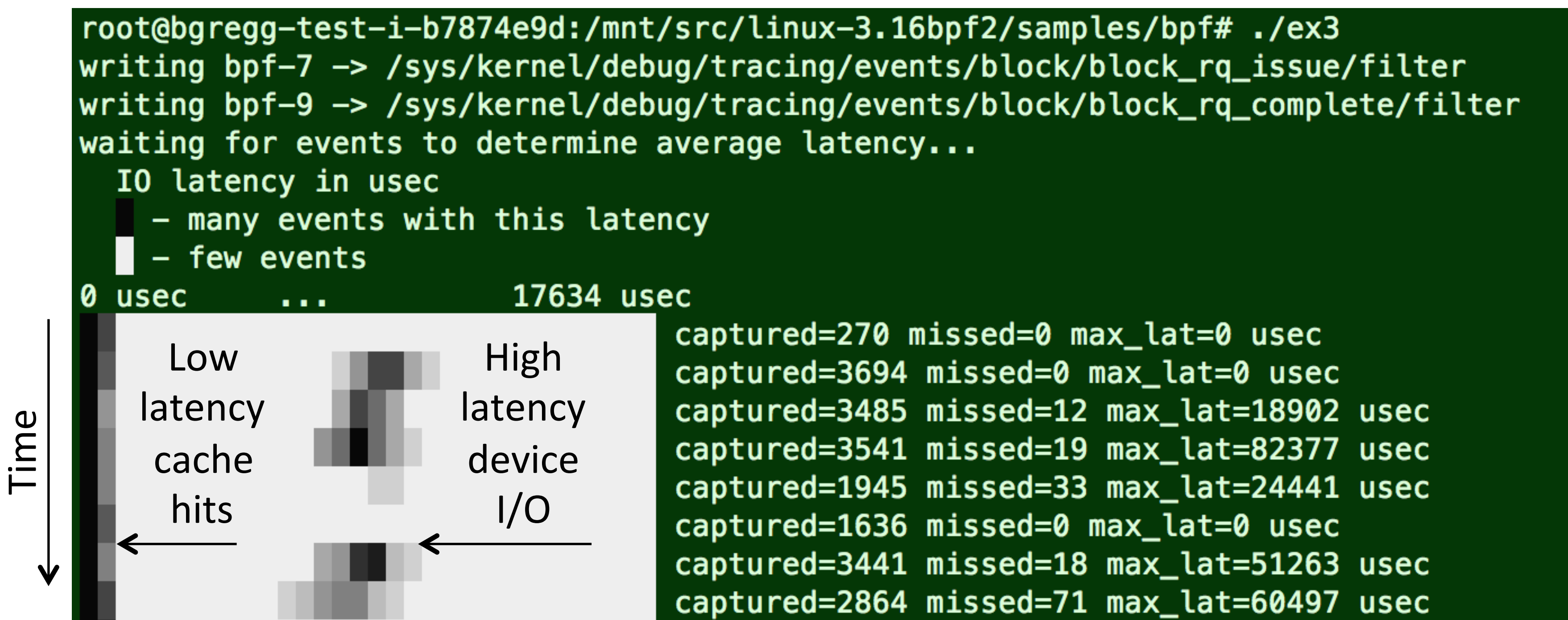
```
root@bgregg-test-i-b7874e9d:/mnt/src/linux-3.16bpf2/samples/bpf# ./ex3
writing bpf-7 -> /sys/kernel/debug/tracing/events/block/block_rq_issue/filter
writing bpf-9 -> /sys/kernel/debug/tracing/events/block/block_rq_complete/filter
waiting for events to determine average latency...
  IO latency in usec
    - many events with this latency
    - few events
0 usec     ...          17634 usec
```



```
captured=270 missed=0 max_lat=0 usec
captured=3694 missed=0 max_lat=0 usec
captured=3485 missed=12 max_lat=18902 usec
captured=3541 missed=19 max_lat=82377 usec
captured=1945 missed=33 max_lat=24441 usec
captured=1636 missed=0 max_lat=0 usec
captured=3441 missed=18 max_lat=51263 usec
captured=2864 missed=71 max_lat=60497 usec
```

Linux Tracing Future

- ftrace + perf_events cover much, but not custom in-kernel aggregations
- eBPF may provide this missing feature
  – eg, in-kernel latency heat map (showing bimodal):



CPU Performance Counters

- … is this even possible from a cloud guest?

CPU Performance Counters

- Model Specific Registers (MSRs)
  - Basic details: timestamp clock, temperature, power
  - Some are available in EC2
- Performance Monitoring Counters (PMCs)
  - Advanced details: cycles, stall cycles, cache misses, …
  - Not available in EC2 (by default)
- Root cause CPU usage at the cycle level
  - Eg, higher CPU usage due to more memory stall cycles

msr-cloud-tools

- Uses the msr-tools package and rdmsr(1)
  - https://github.com/brendangregg/msr-cloud-tools

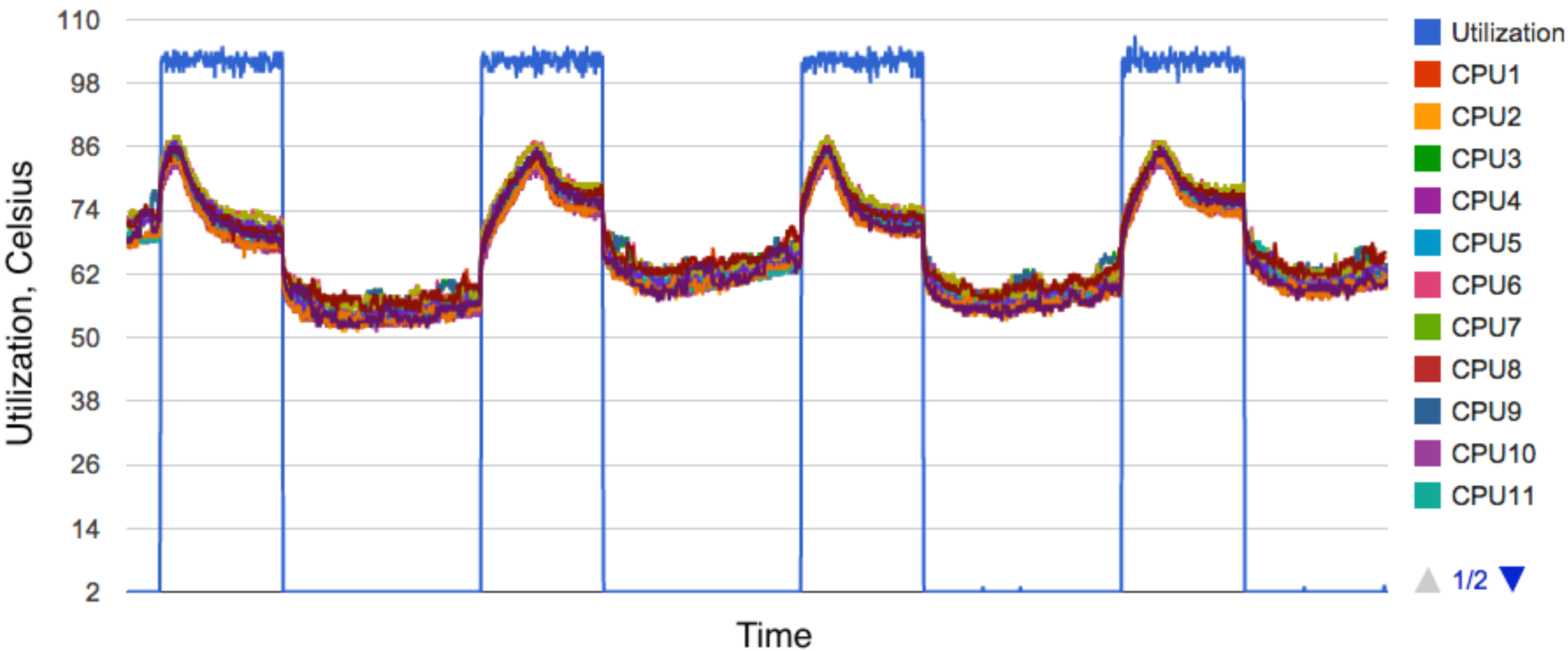
```
ec2-guest# ./cputemp 1
CPU1 CPU2 CPU3 CPU4
61 61 60 59
60 61 60 60
[...]
ec2-guest# ./showboost
CPU MHz      : 2500
Turbo MHz    : 2900 (10 active)
Turbo Ratio : 116% (10 active)
CPU 0 summary every 5 seconds...

TIME        C0_MCYC      C0_ACYC        UTIL  RATIO    MHz
06:11:35    6428553166   7457384521      51%   116%   2900
06:11:40    6349881107   7365764152      50%   115%   2899
06:11:45    6240610655   7239046277      49%   115%   2899
06:11:50    6225704733   7221962116      49%   116%   2900
[...]
```

CPU Temperature

Real CPU MHz

MSRs: CPU Temperature

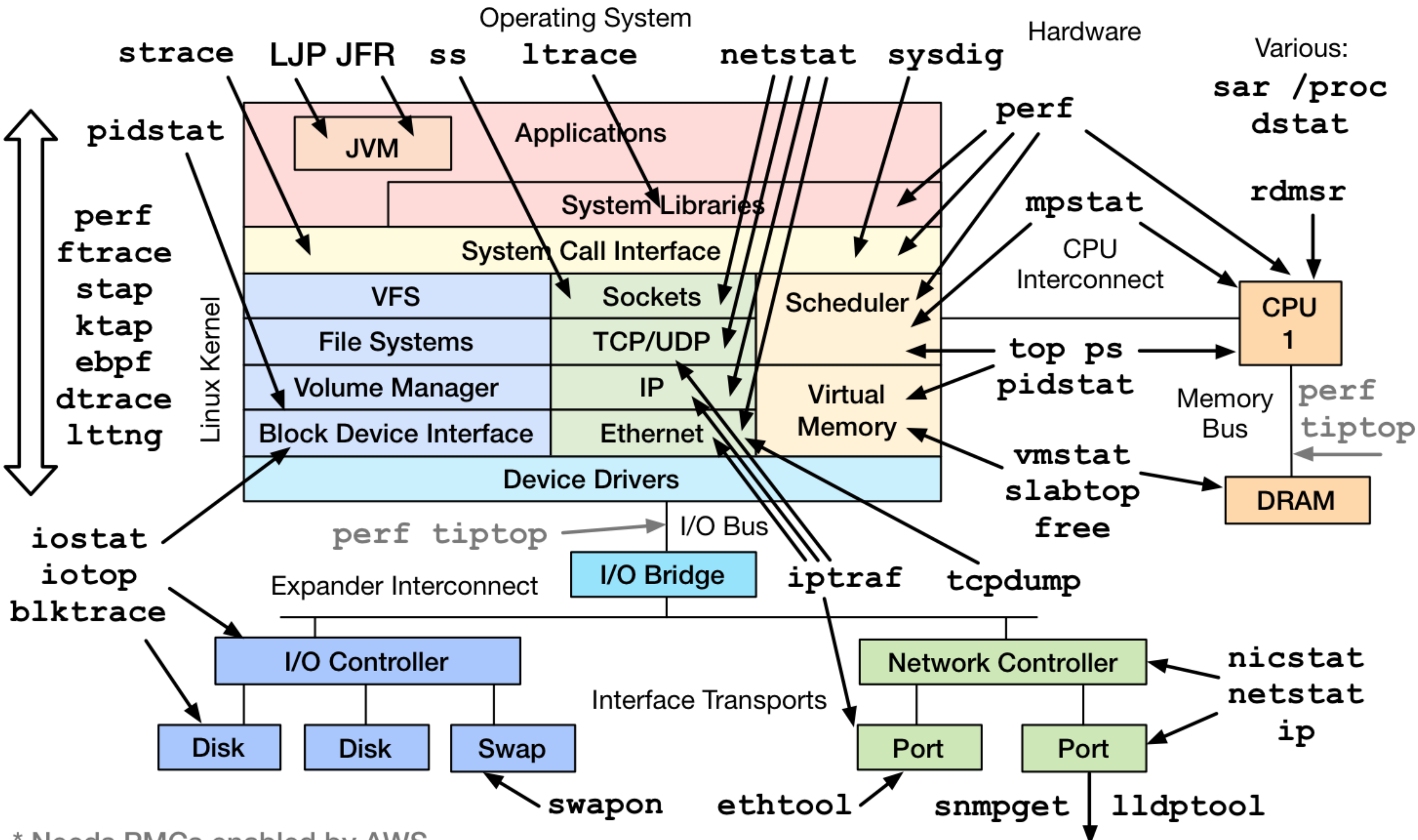
- Useful to explain variation in turbo boost (if seen)
- Temperature for a synthetic workload:



MSRs: Intel Turbo Boost

- Can dynamically increase CPU speed up to 30+%
- This can mess up all performance comparisons
- Clock speed can be observed from MSRs using
  - IA32_MPERF: Bits 63:0 is TSC Frequency Clock Counter C0_MCNT TSC relative
  - IA32_APERF: Bits 63:0 is TSC Frequency Clock Counter C0_ACNT actual clocks
- This is how msr-cloud-tools showturbo works

PMCs



* Needs PMCs enabled by AWS

PMCs

- Needed for remaining low-level CPU analysis:
  – CPU stall cycles, and stall cycle breakdowns
  – L1, L2, L3 cache hit/miss ratio
  – Memory, CPU Interconnect, and bus I/O
- Not enabled by default in EC2. Is possible, eg:

```
# perf stat -e cycles,instructions,r0480,r01A2 -p `pgrep -n java` sleep 10

 Performance counter stats for process id '17190':

    71,208,028,133 cycles                    #    0.000 GHz                    [100.00%]
    41,603,452,060 instructions              #    0.58  insns per cycle        [100.00%]
    23,489,032,742 r0480                                                       [100.00%]
    20,241,290,520 r01A2

      10.000894718 seconds time elapsed
```

(r0480 ← ICACHE.IFETCH_STALL; r01A2 ← RESOURCE_STALLS.ANY)

Using Advanced Perf Tools

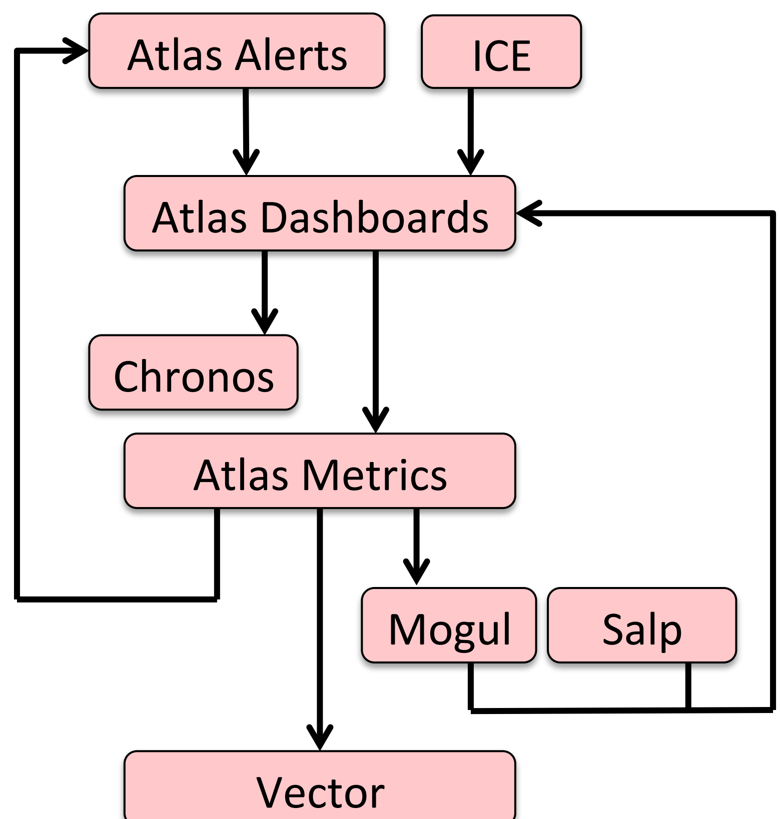
- Everyone doesn't need to learn these
- Reality:
  - A. Your company has one or more people for advanced perf analysis (perf team). Ask them.
  - B. You are that person
  - C. You buy a product that does it. Ask them.
- If you aren't the advanced perf engineer, you need to know what to ask for
  - Flame graphs, latency heat maps, ftrace, PMCs, etc...
- At Netflix, we're building the (C) option: Vector

Future Work: Vector

Future Work: Vector

Future Work: Vector

- Real-time, per-second, instance metrics
- On-demand CPU flame graphs, heat maps, ftrace metrics, and SystemTap metrics
- Analyze from clouds to roots quickly, and from a web interface
- Scalable: other teams can use it easily



In Summary

- 1. Netflix architecture
  - Fault tolerance: ASGs, ASG clusters, Hystrix (dependency API), Zuul (proxy), Simian army (testing)
  - Reduces the severity and urgency of issues
- 2. Cloud Analysis
  - Atlas (alerts/dashboards/metrics), Chronos (event tracking), Mogul & Salp (dependency analysis), ICE (AWS usage)
  - Quickly narrow focus from cloud to ASG to instance
- 3. Instance Analysis
  - Linux tools (*stat, sar, …), perf_events, ftrace, perf-tools, rdmsr, msr-cloud-tools, Vector
  - Read logs, profile & trace all software, read CPU counters

References & Links

https://netflix.github.io/#repo
http://techblog.netflix.com/2012/01/auto-scaling-in-amazon-cloud.html
http://techblog.netflix.com/2012/06/asgard-web-based-cloud-management-and.html
http://www.slideshare.net/benjchristensen/performance-and-fault-tolerance-for-the-netflix-api-qcon-sao-paulo
http://www.slideshare.net/adrianco/netflix-nosql-search
http://www.slideshare.net/ufried/resilience-with-hystrix
https://github.com/Netflix/Hystrix, https://github.com/Netflix/Zuul
http://techblog.netflix.com/2011/07/netflix-simian-army.html
http://techblog.netflix.com/2014/09/introducing-chaos-engineering.html
http://www.brendangregg.com/blog/2014-06-12/java-flame-graphs.html
http://www.brendangregg.com/blog/2014-09-17/node-flame-graphs-on-linux.html
Systems Performance: Enterprise and the Cloud, Prentice Hall, 2014
http://sourceforge.net/projects/nicstat/
perf-tools: https://github.com/brendangregg/perf-tools
Ftrace: The Hidden Light Switch: http://lwn.net/Articles/608497/
msr-cloud-tools: https://github.com/brendangregg/msr-cloud-tools

Thanks

- Coburn Watson, Adrian Cockcroft
- Atlas: Insight Engineering (Roy Rapoport, etc.)
- Mogul: Performance Engineering (Scott Emmons, Martin Spier)
- Vector: Performance Engineering (Martin Spier, Amer Ather)

Thanks

- Questions?
- http://techblog.netflix.com
- http://slideshare.net/brendangregg
- http://www.brendangregg.com
- bgregg@netflix.com
- @brendangregg