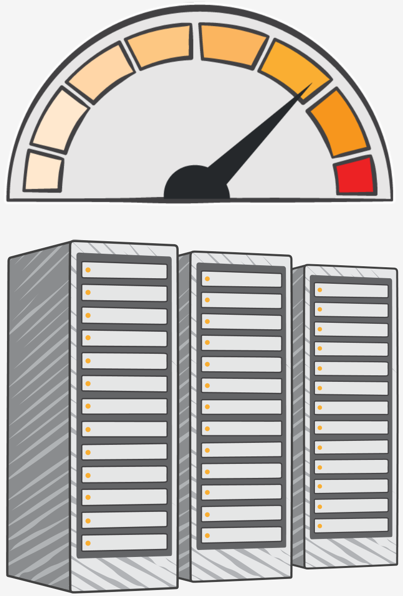
PFC306

# Performance Tuning EC2 Instances

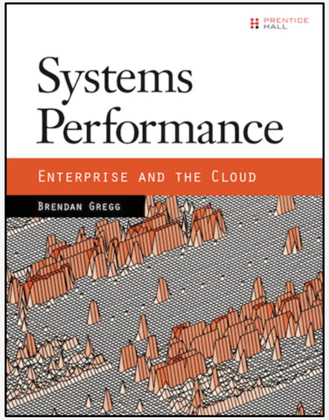
Brendan Gregg, Performance Engineering, Netflix

November 12, 2014 | Las Vegas, NV



# Brendan Gregg

- Senior Performance Architect, Netflix
  - Linux and FreeBSD performance
  - On the Performance Engineering Team, led by Coburn Watson (and we're hiring!)
- Recent work:
  - Linux perf-tools, using ftrace & perf_events
- Previous work includes:
  - USE Method, flame graphs, heat maps, DTrace tools
  - Sysadmin, training, kernel engineering, performance



NETFLIX

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

# Netflix Performance Engineering Team

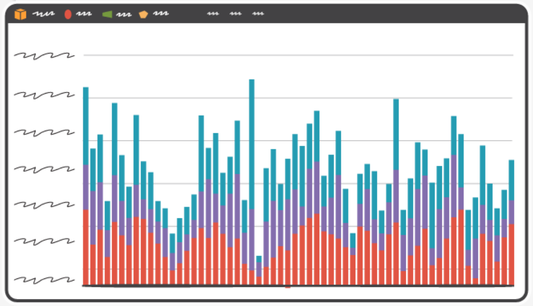
- Evaluate technology
  - Instance types, Amazon EC2 options
- Recommendations & best practices
  - Instance kernel tuning, assist app tuning
- Develop performance tools
  - Develop tools for observability and analysis
- Project support
  - New database, programming language, software change
- Incident response
  - Performance issues, scalability issues

## Agenda

1. Instance Selection
2. Amazon EC2 Features
3. Kernel Tuning
4. Observability

# Performance Tuning on Amazon EC2

- In the Netflix cloud, everything is a tunable
  - Including instance type
- Performance wins have immediate benefits
  - Great place to do performance engineering!



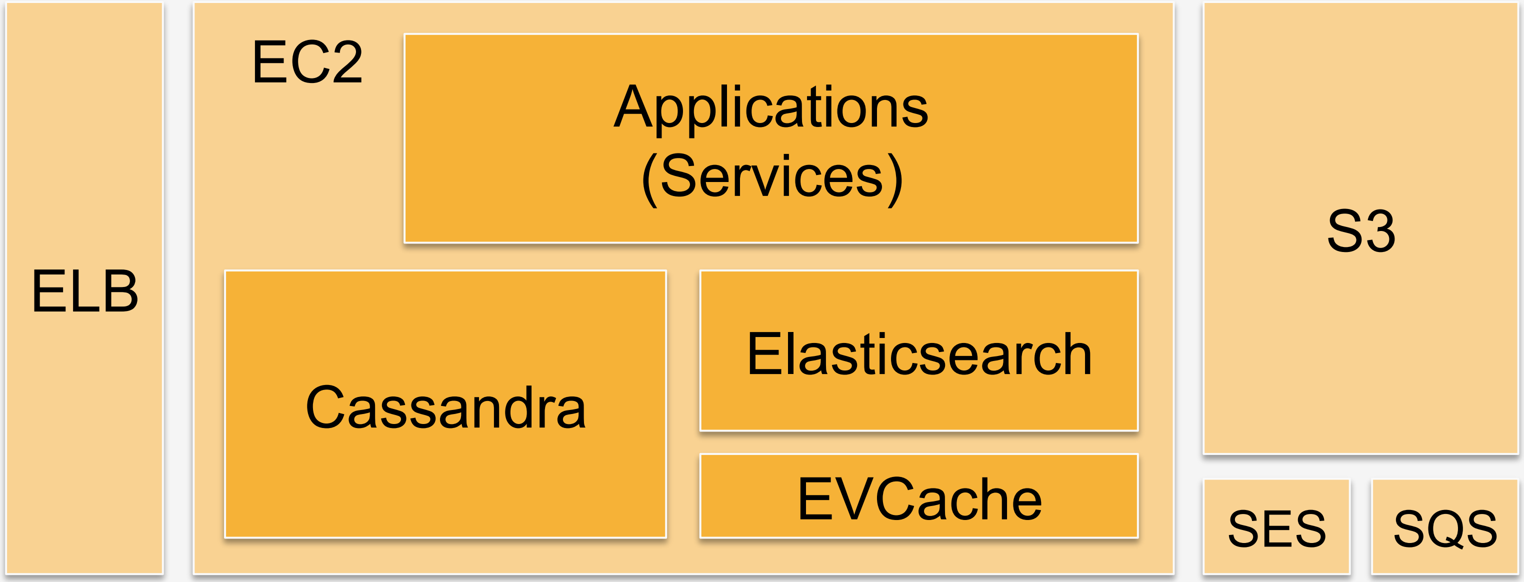
## WARNINGS

- This is what’s in our medicine cabinet
- Consider these “best before: 2015”
- Take only if prescribed by a performance engineer

## 1. Instance Selection

# The Netflix Cloud

- Many different application workloads: compute, storage, caching…



# Current Generation Instance Families

- i2: Storage-optimized
  - SSD large capacity storage
- r3: Memory optimized
  - Lowest cost/Gbyte
- c3: Compute-optimized
  - Latest CPUs, lowest price/compute perf
- m3: General purpose
  - Balanced
- Plus some others

*i2.8xlarge*

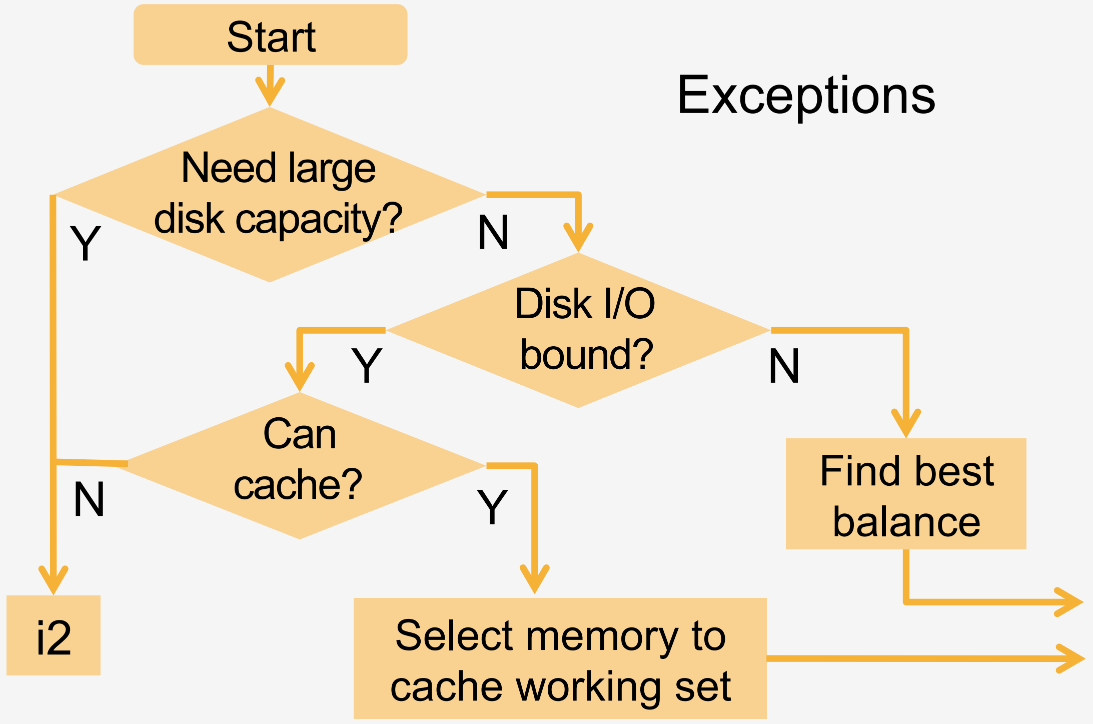
## Instance Sizes

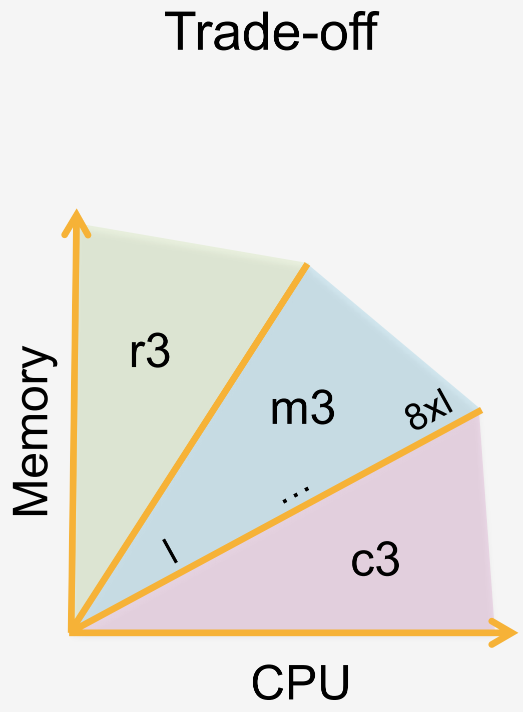
- Ranges from medium to 8xlarge, depending on type
- Netflix has over 30 instance types in use
- Traditional:
  - Tune the workload to match the server
- Cloud:
  - Find an ideal workload and instance type *combination*
    - Instead of: given A, optimize B; this is optimize A+B
  - Greater flexibility, best price/performance

## Netflix Instance Type Selection

- Flow Chart
- By-Resource
- Brute Force

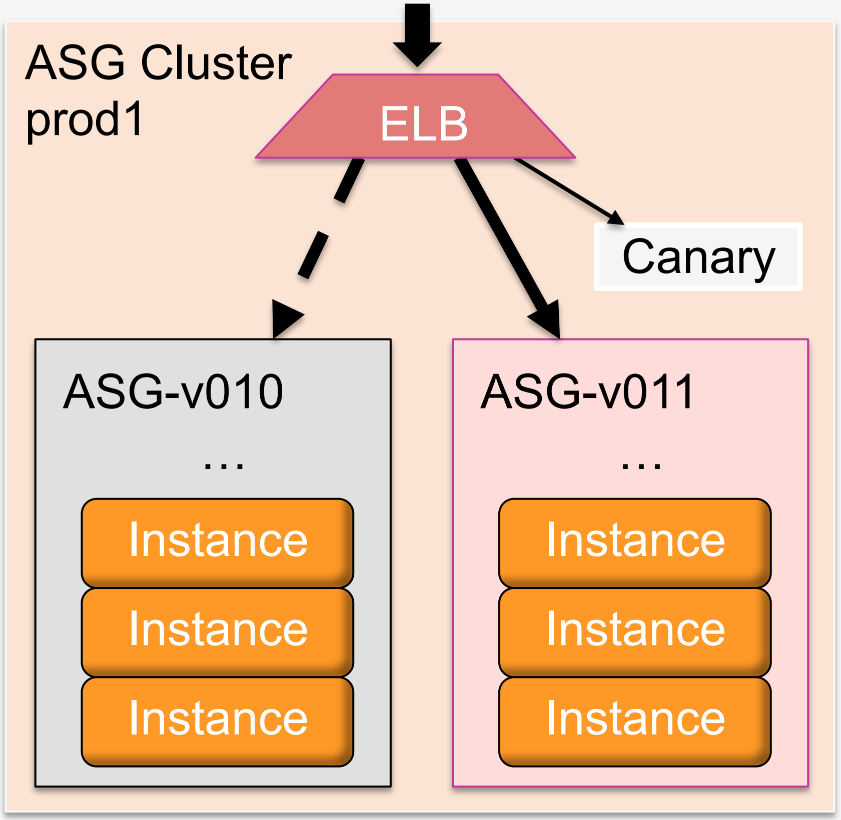
# Instance Selection Flow Chart





# Netflix AWS Environment

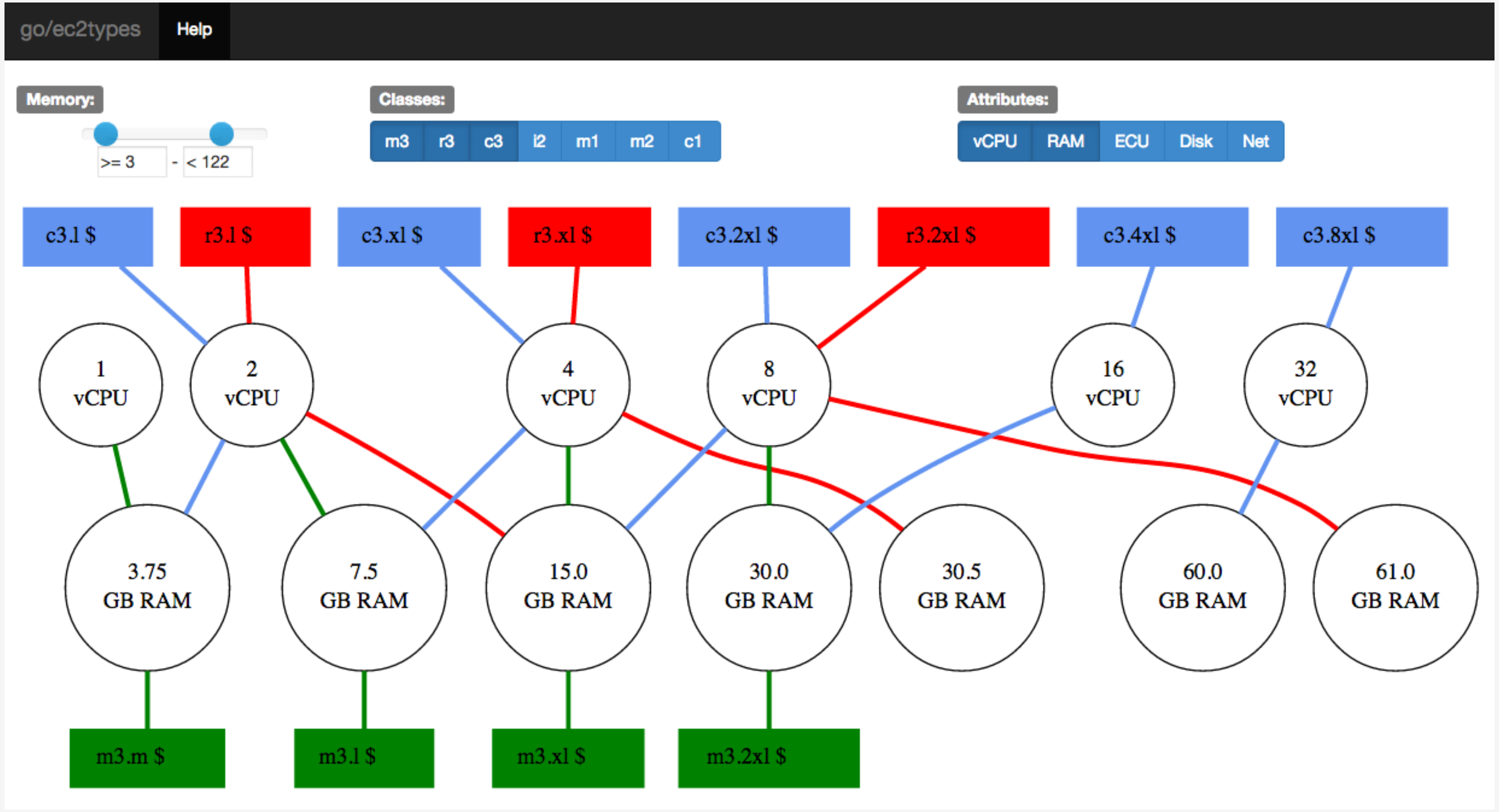
- Elastic Load Balancing allows instance types to be tested with real load
  1. Single instance canary, then,
  2. Auto Scaling Group
- Much better than micro-benchmarking alone, which is extremely error prone



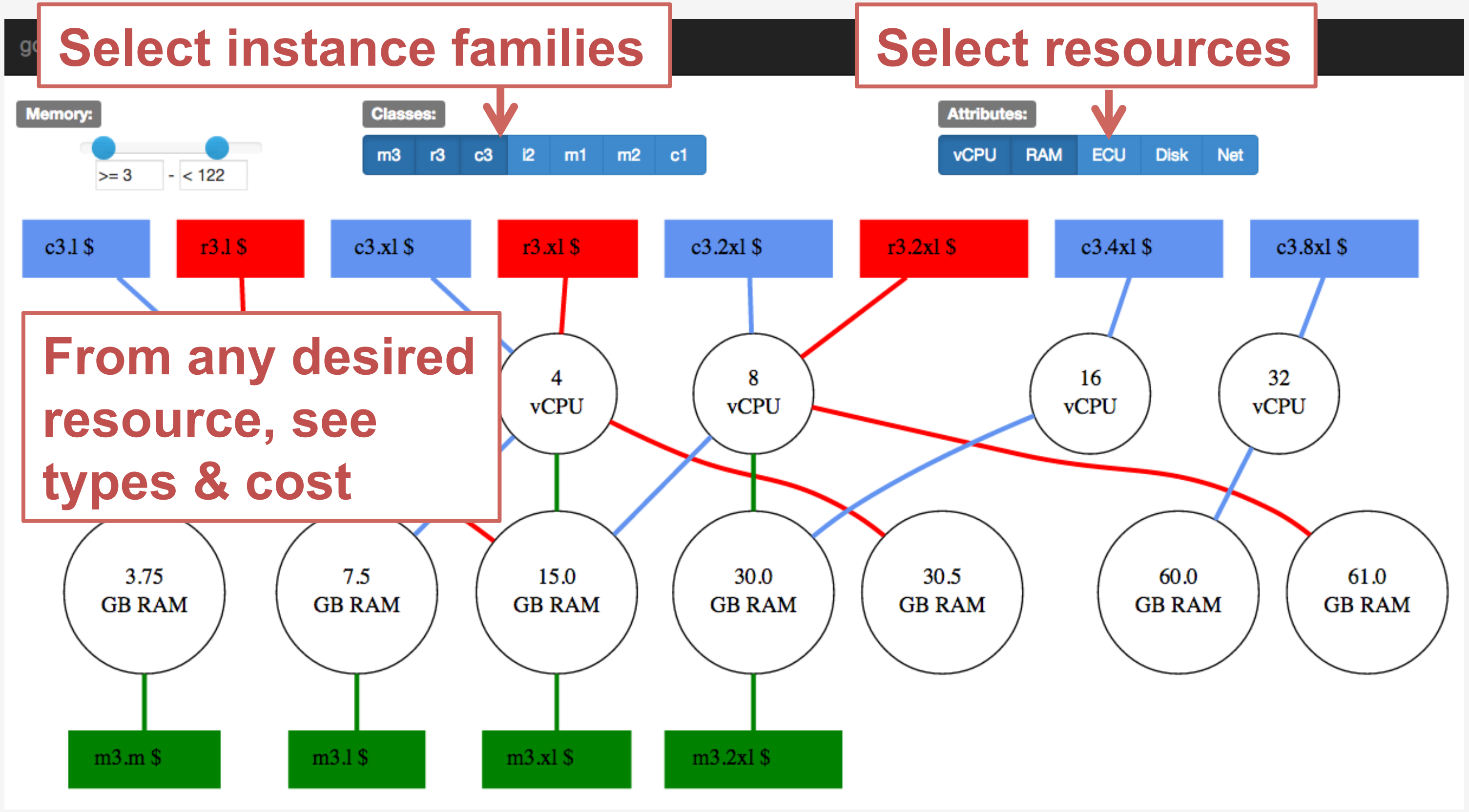
## By-Resource Approach

1. Determine bounding resource
   - Eg: CPU, disk I/O, or network I/O
   - Found using:
     - Estimation (expertise)
     - Resource observability with an existing real workload
     - Resource observability with a benchmark or load test (experimentation)
2. Choose instance type for the bounding resource
   - If disk I/O, consider caching, and a memory-optimized type
   - We have tools to aid this choice: Nomogram Visualization

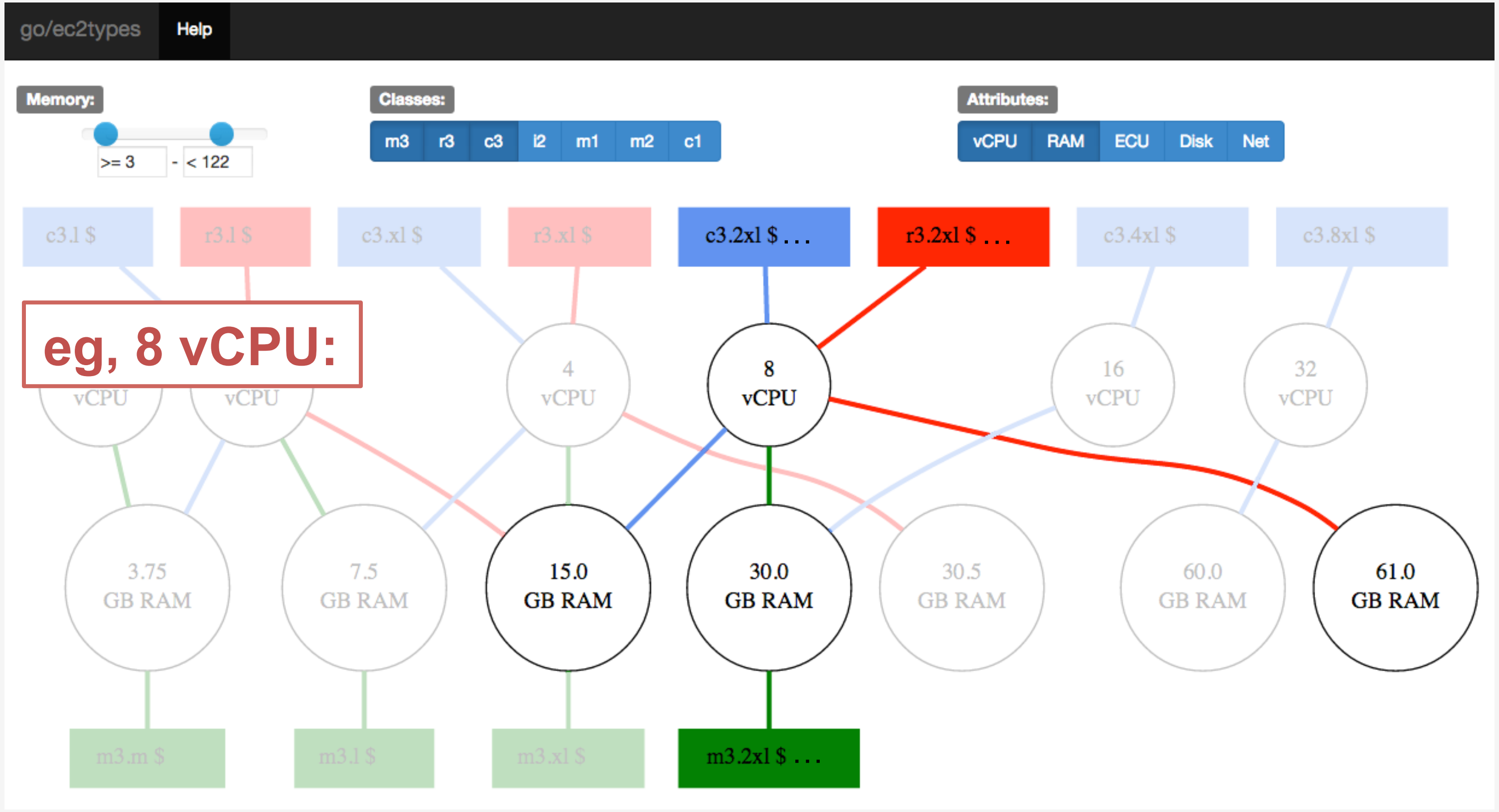
# Nomogram Visualization Tool



# Nomogram Visualization Tool



# Nomogram Visualization Tool



**eg, 8 vCPU:**

## By-Resource Approach, cont.

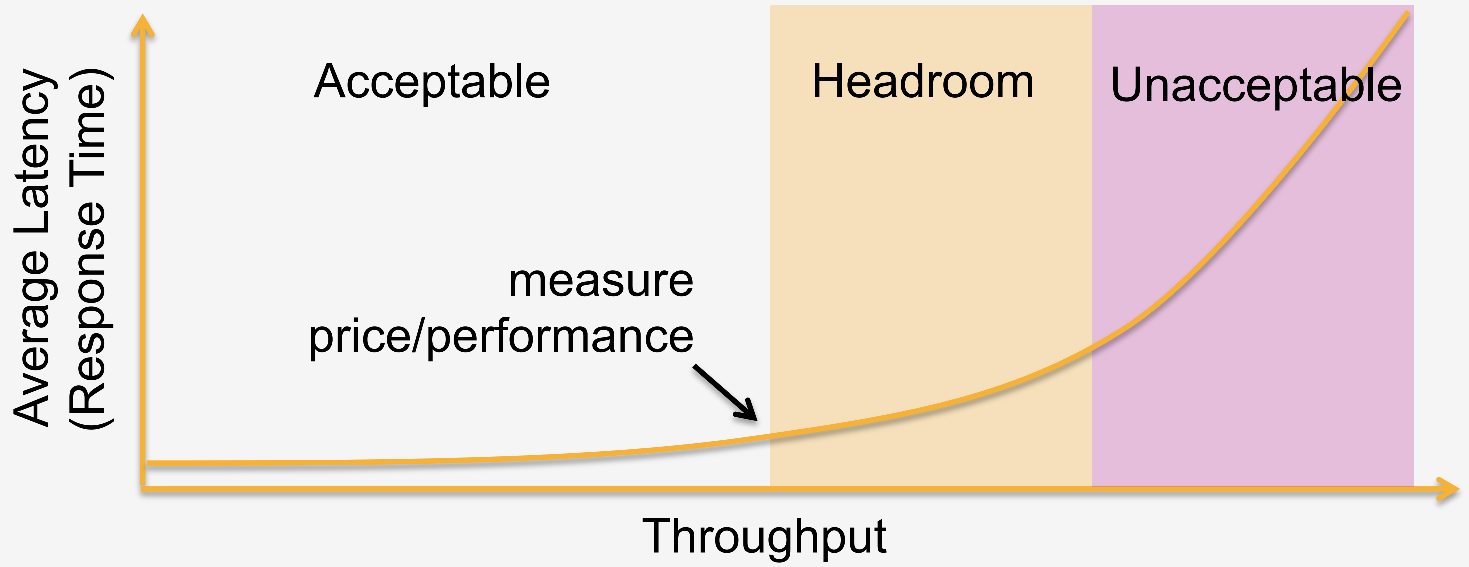
- This focuses on optimizing a given workload
- More efficiency can be found by adjusting the workload to suit different instance types

## Brute Force Choice

1. Run load test on ALL instance types
   – Optionally different workload configurations as well
2. Measure throughput
   – And check for acceptable latency
3. Calculate price/performance for all types
4. Choose most efficient type

# Latency Requirements

- Check for an acceptable latency distribution when optimizing for price/performance

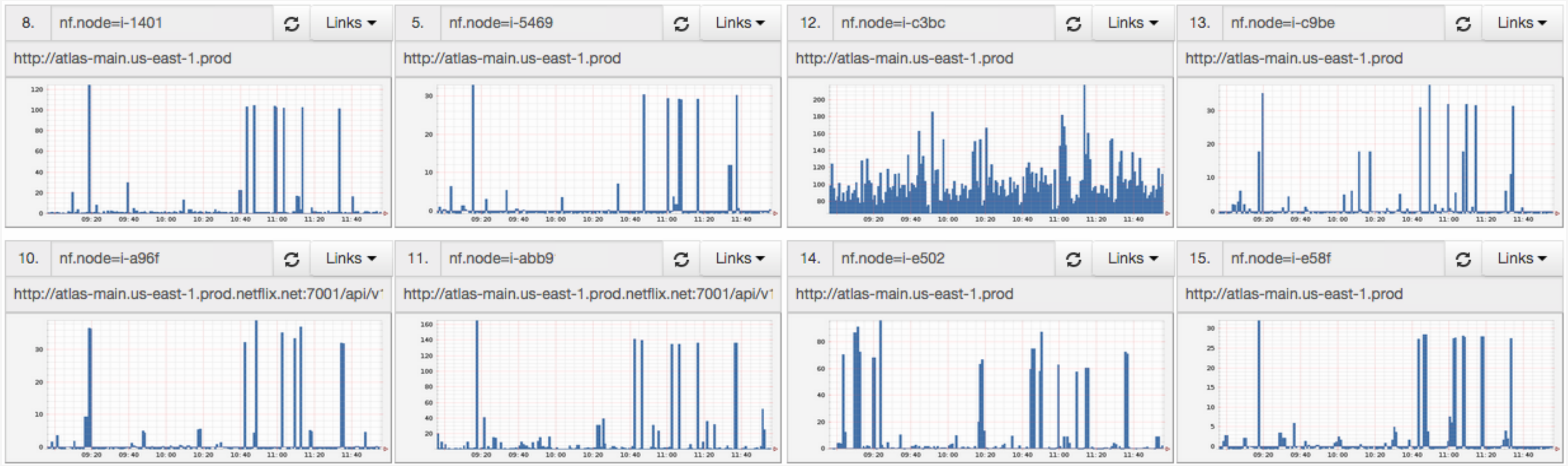


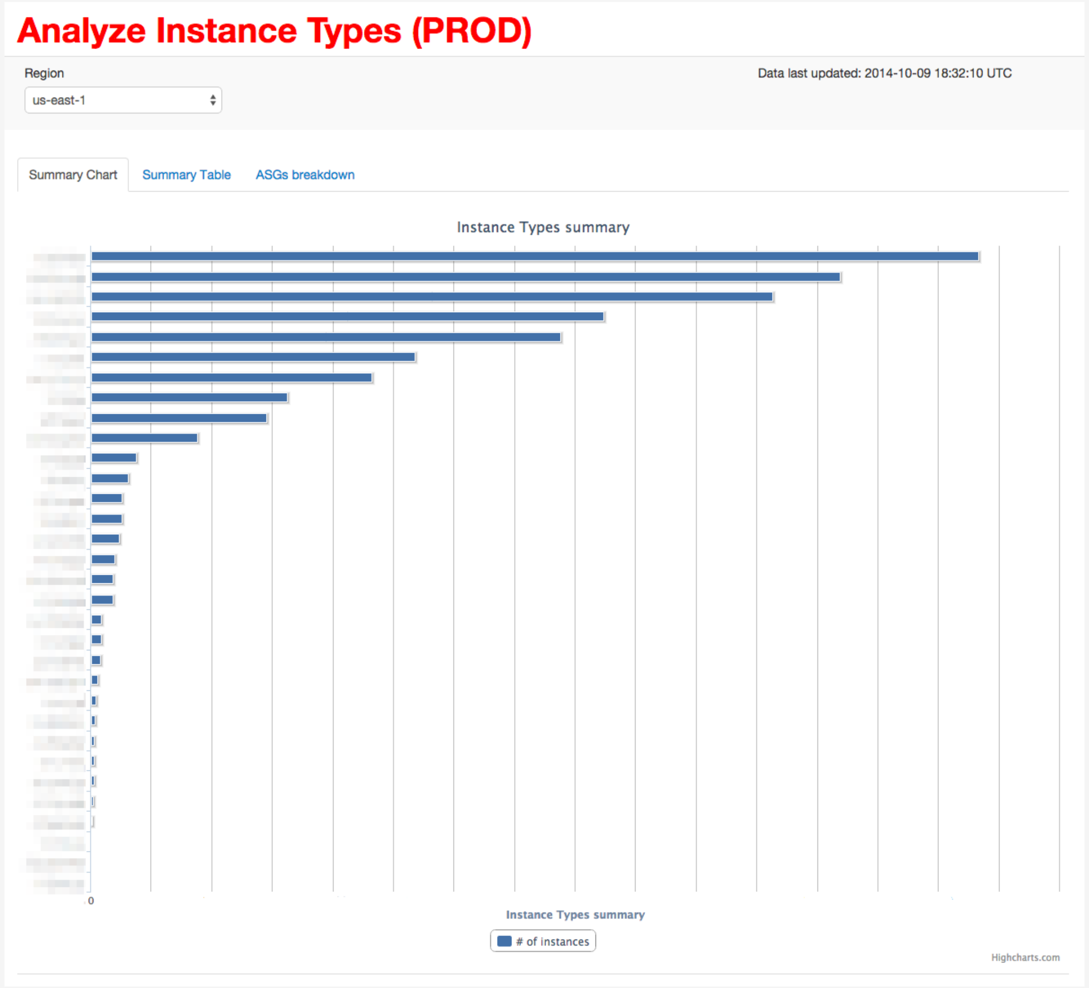
## Netflix Instance Type Re-selection

1. Variance
2. Usage
3. Cost

# 1. Instance Variance

- An instance type may be resource constrained only occasionally, or after warmup, or a code change
- Continually monitor performance, analyze variance/outliers


8. nf.node=i-1401
Links
http://atlas-main.us-east-1.prod
5. nf.node=i-5469
Links
http://atlas-main.us-east-1.prod
12. nf.node=i-c3bc
Links
http://atlas-main.us-east-1.prod
13. nf.node=i-c9be
Links
http://atlas-main.us-east-1.prod
10. nf.node=i-a96f
Links
http://atlas-main.us-east-1.prod.netflix.net:7001/api/v
11. nf.node=i-abb9
Links
http://atlas-main.us-east-1.prod.netflix.net:7001/api/v1
14. nf.node=i-e502
Links
http://atlas-main.us-east-1.prod
15. nf.node=i-e58f
Links
http://atlas-main.us-east-1.prod


Analyze Instance Types (PROD)

Region: us-east-1

Data last updated: 2014-10-09 18:32:10 UTC

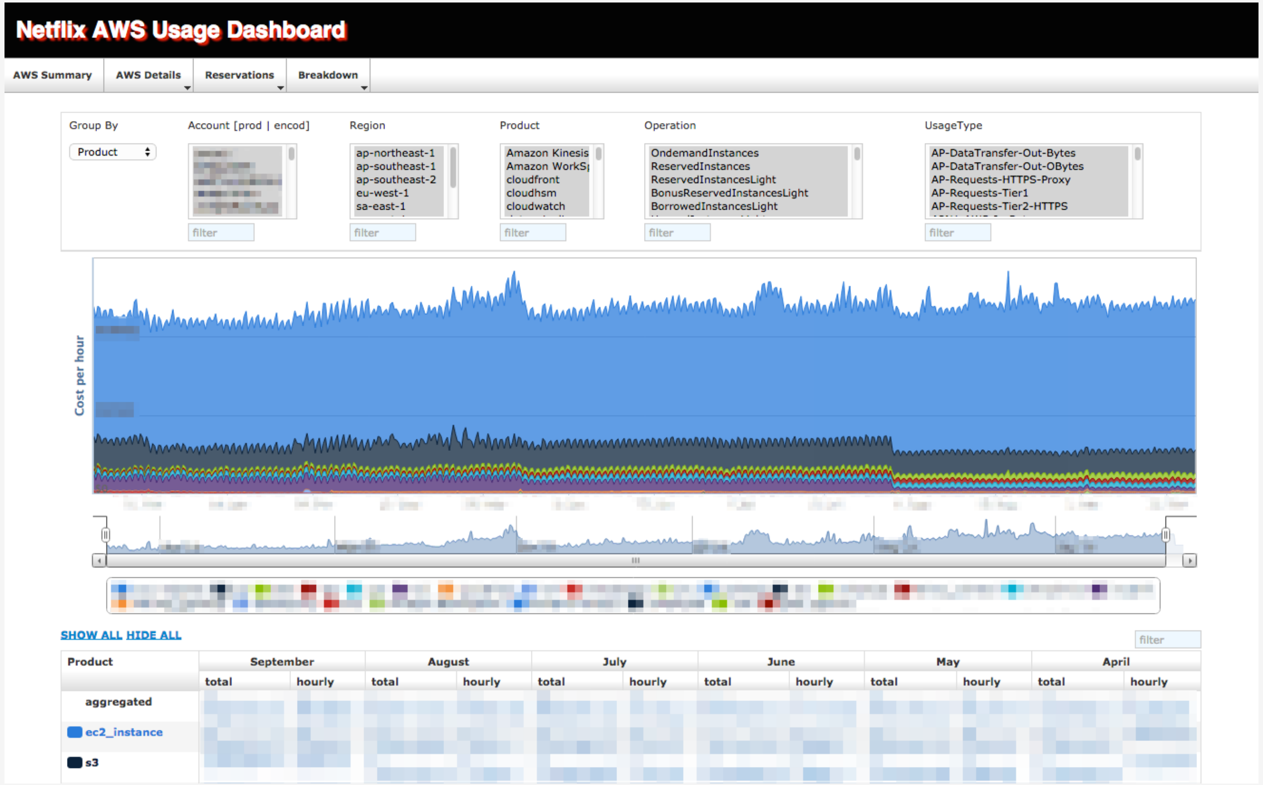
# 2. Instance Usage

Older instance type usage can be quickly identified using internal tools, re-evaluated, and upgraded to newer types

# 3. Instance Cost

Also checked regularly.

Tuning the price in price/perf.



Netflix AWS Usage Dashboard

AWS Summary | AWS Details | Reservations | Breakdown

| Group By | Account [prod \| encod] | Region | Product | Operation | UsageType |
|---|---|---|---|---|---|
| Product | [redacted] | ap-northeast-1 | Amazon Kinesis | OndemandInstances | AP-DataTransfer-Out-Bytes |
| | | ap-southeast-1 | Amazon WorkS | ReservedInstances | AP-DataTransfer-Out-OBytes |
| | | ap-southeast-2 | cloudfront | ReservedInstancesLight | AP-Requests-HTTPS-Proxy |
| | | eu-west-1 | cloudhsm | BonusReservedInstancesLight | AP-Requests-Tier1 |
| | | sa-east-1 | cloudwatch | BorrowedInstancesLight | AP-Requests-Tier2-HTTPS |
| | filter | filter | filter | filter | filter |

SHOW ALL HIDE ALL

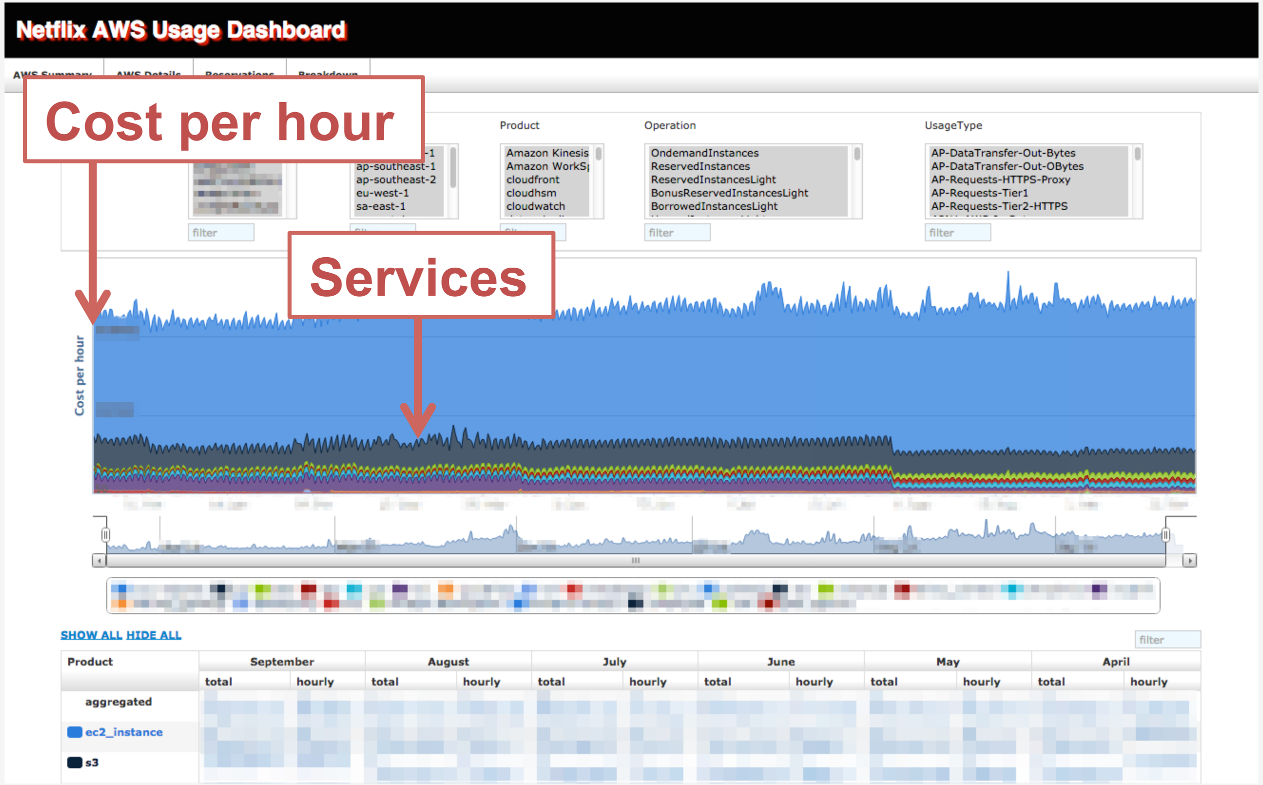
filter

| Product | September | | August | | July | | June | | May | | April | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | total | hourly | total | hourly | total | hourly | total | hourly | total | hourly | total | hourly |
| aggregated | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] |
| 🟦 ec2_instance | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] |
| ⬛ s3 | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] | [redacted] |

# 3. Instance Cost

Also checked regularly.

Tuning the price in price/perf.



| Product | September | | August | | July | | June | | May | | April | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | total | hourly | total | hourly | total | hourly | total | hourly | total | hourly | total | hourly |
| aggregated | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| ec2_instance | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |
| s3 | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] | [illegible] |

## 2. EC2 Features

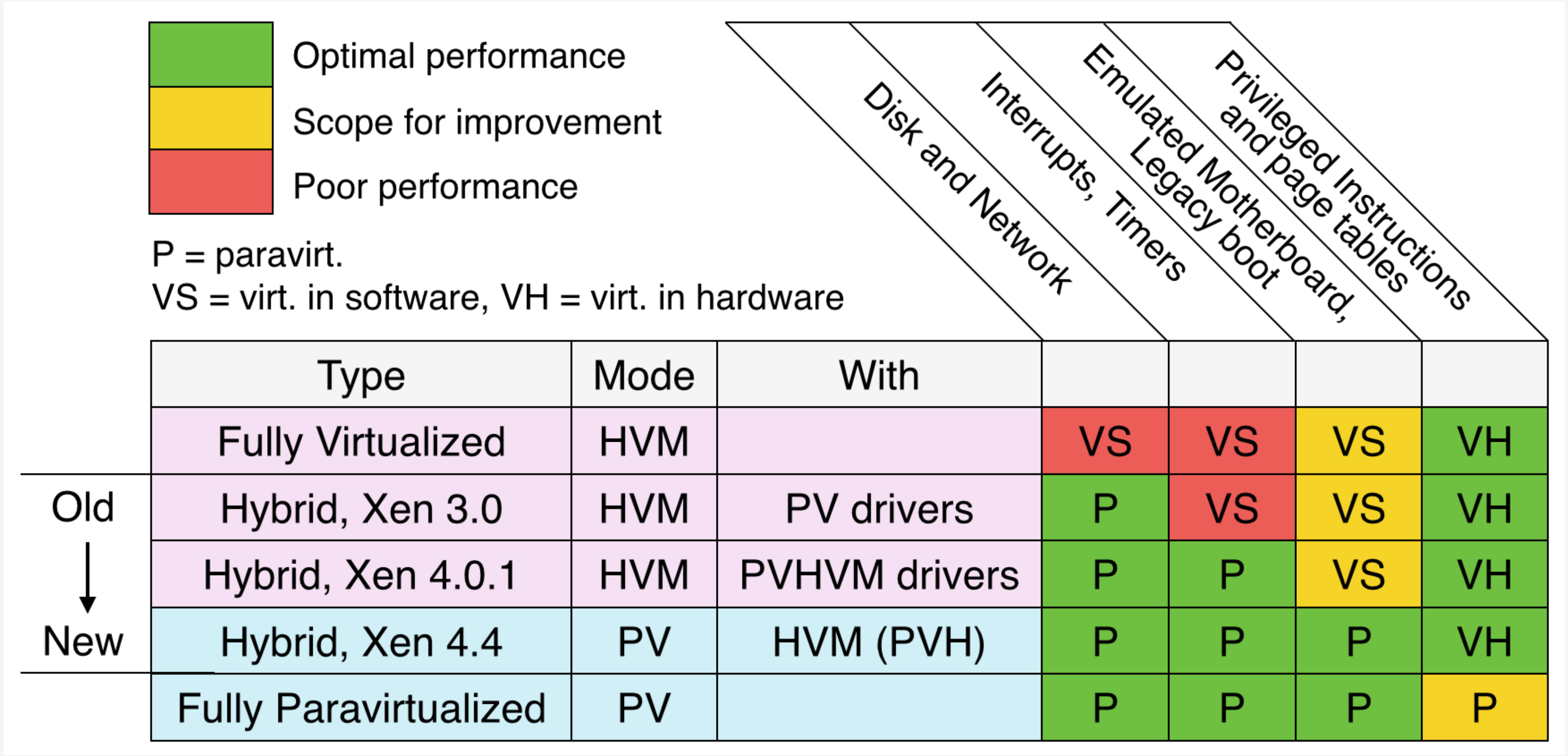
# EC2 Features

- A) Xen Modes: HVM, PV…
- B) SR-IOV

## Xen Modes

- The best performance is the latest PV/HVM hybrid mode that Xen supports
- On Amazon EC2:
  - “HVM” == “PVHVM” if your Linux kernel version supports it
  - “PV” == “PV”
- Current fastest on Amazon EC2, in general: “HVM” (PVHVM)
- Future fastest, in general: should be PVH
- WARNINGS: Your mileage may vary, and beware of outdated info.

# Xen Modes



🟩 Optimal performance

🟨 Scope for improvement

🟥 Poor performance

P = paravirt.
VS = virt. in software, VH = virt. in hardware

| | Type | Mode | With | Disk and Network | Interrupts, Timers | Emulated Motherboard, Legacy boot | Privileged Instructions and page tables |
|---|---|---|---|---|---|---|---|
| | Fully Virtualized | HVM | | 🟥 VS | 🟥 VS | 🟨 VS | 🟩 VH |
| Old | Hybrid, Xen 3.0 | HVM | PV drivers | 🟩 P | 🟥 VS | 🟨 VS | 🟩 VH |
| ↓ | Hybrid, Xen 4.0.1 | HVM | PVHVM drivers | 🟩 P | 🟩 P | 🟨 VS | 🟩 VH |
| New | Hybrid, Xen 4.4 | PV | HVM (PVH) | 🟩 P | 🟩 P | 🟩 P | 🟩 VH |
| | Fully Paravirtualized | PV | | 🟩 P | 🟩 P | 🟩 P | 🟨 P |

## SR-IOV

- Single Root I/O Virtualization (SR-IOV)
  - PCIe device provides virtualized instances
- AWS “Enhanced Networking”
  - Available for some instance types: C3, R3, I2
  - VPC only
- Example improvements:
  - Improved network throughput
  - Reduced average network RTT, and reduced jitter
  - Improvement depends on instance type and network

## 3. Kernel Tuning

## Kernel Tuning

- Typically 5-25% wins, for average performance
  - Adds up to significant savings for the Netflix cloud
- Bigger wins when reducing latency outliers
- Deploying tuning:
  - Generic performance tuning is baked into our base AMI
  - Experimental tuning is a package add-on (nflx-kernel-tunables)
  - Workload specific tuning is configured in application AMIs
  - Remember to tune the workload with the tunables

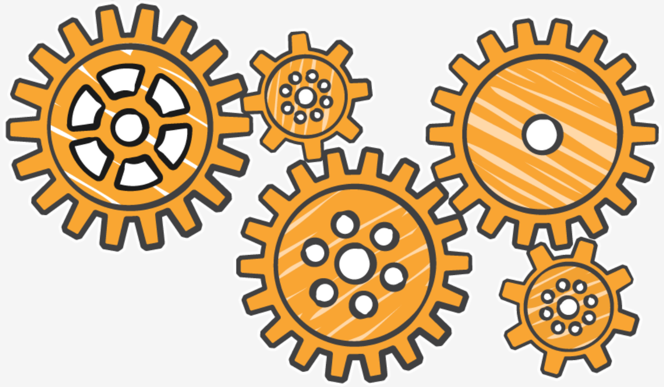
## OS Distributions

- Netflix uses both CentOS and Ubuntu
  - Operating system distributions can override tuning defaults
  - Different kernel versions can provide different defaults, and additional tunables
- Tunable parameters include sysctl and /sys

*The following tunables provide a look into our current medicine cabinet. Your mileage may vary. Only take if prescribed by a performance engineer.*

## Tuning Targets

1. CPU Scheduler
2. Virtual Memory
3. Huge Pages
4. File System
5. Storage I/O
6. Networking
7. Hypervisor (Xen)



# 1. CPU Scheduler

- Tunables:
  - Scheduler class, priorities, migration latency, tasksets…
- Usage:
  - Some apps benefit from reducing migrations using taskset(1), numactl(8), cgroups, and tuning sched_migration_cost_ns
  - Some Java apps have benefited from SCHED_BATCH, to reduce context switching. E.g.:

```
# schedtool –B PID
```

# 2. Virtual Memory

- Tunables:
  - swappiness, overcommit, OOM behavior…
- Usage:
  - Swappiness is set to zero to disable swapping and favor ditching the file system page cache first to free memory. (This tunable doesn't make much difference, as swap devices are usually absent.)

```
vm.swappiness = 0                                      # from 60
```

## 3. Huge Pages

Page size of 2 or 4 Mbytes, instead of 4k, should reduce various CPU overheads and improve MMU page translation cache reach.

- Tunables:
  - Explicit huge page usage, transparent huge pages (THPs)
- Usage:
  - THPs (enabled in later kernels), depending on the workload & CPUs, sometimes improve perf (~5% lower CPU), but sometimes hurt perf (~25% higher CPU during %usr, and more during %sys refrag). To disable:

```
# echo never > /sys/kernel/mm/transparent_hugepage/enabled   # from madvise
```

# 4. File System

- Tunables:
  - page cache flushing behavior, file system type and its own tunables (e.g., ZFS on Linux)
- Usage:
  - Page cache flushing is tuned to provide a more even behavior: background flush earlier, aggressive flush later
  - Access timestamps disabled, and other options depending on the FS

```
vm.dirty_ratio = 80                      # from 40
vm.dirty_background_ratio = 5            # from 10
vm.dirty_expire_centisecs = 12000        # from 3000
mount -o defaults,noatime,discard,nobarrier …
```

## 5. Storage I/O

- Tunables:
  - Read ahead size, number of in-flight requests, I/O scheduler, volume stripe width…
- Usage:
  - Some workloads, e.g., Cassandra, can be sensitive to read ahead size
  - SSDs can perform better with the "noop" scheduler (if not default already)
  - Tuning md chunk size and stripe width to match workload

```
/sys/block/*/queue/rq_affinity    2
/sys/block/*/queue/scheduler      noop
/sys/block/*/queue/nr_requests    256
/sys/block/*/queue/read_ahead_kb  256
mdadm –chunk=64 ...
```

# 6. Networking

- Tunables:
  - TCP buffer sizes, TCP backlog, device backlog, TCP reuse, …
- Usage:

```
net.core.somaxconn = 1000
net.core.netdev_max_backlog = 5000
net.core.rmem_max = 16777216
net.core.wmem_max = 16777216
net.ipv4.tcp_wmem = 4096 12582912 16777216
net.ipv4.tcp_rmem = 4096 12582912 16777216
net.ipv4.tcp_max_syn_backlog = 8096
net.ipv4.tcp_slow_start_after_idle = 0
net.ipv4.tcp_tw_reuse = 1
net.ipv4.ip_local_port_range = 10240 65535
net.ipv4.tcp_abort_on_overflow = 1    # maybe
```

# 7. Hypervisor (Xen)

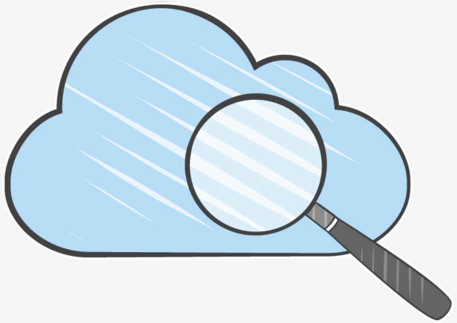
- Tunables:
  - PV/HVM (baked into AMI)
  - Kernel clocksource. From slow to fast: hpet, xen, tsc
- Usage:
  - We’ve encountered a Xen clocksource regression moving to Ubuntu Trusty; fixed by tuning clocksource to TSC (although beware of clock drift). Best case example (so far): CPU usage reduced by 30%, and average app latency reduced by 43%.

```
echo tsc > /sys/devices/system/clocksource/clocksource0/current_clocksource
```

## 4. Observability

# Observability

- Finding, quantifying, and confirming tunables
- Discovering system wins (5-25%’s) and application wins (2-10x’s)



# Inefficient Tuning

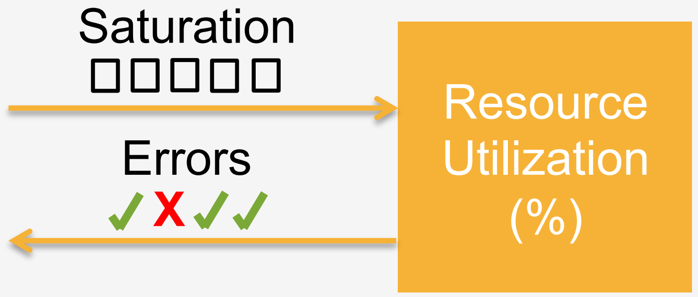
- Whack-A-Mole Anti-Method:
  - Tune things at random until the problem goes away
- Street Light Anti-Method:
  - 1. Pick observability tools that are
    - Familiar
    - Found on the Internet
    - At random
  - 2. Run tools
  - 3. Look for obvious issues

## Efficient Tuning

- Based on observability (data-driven):
  1. Observe workload and resource usage
  2. Analyze results:
     - Identify software and hardware components in use
     - Study in-use components for tunables
     - Quantify expected speed-up
  3. Tune, including experimental, the highest value targets
- Example methodology: the USE Method

# USE Method

- For every hardware and software resource, check:
  1. Utilization
  2. Saturation
  3. Errors
- Resource constraints show as saturation or high utilization
  - Resize or change instance type
  - Investigate tunables for the resource
- The USE Method poses questions you then use tools to answer



# More Inefficient Tuning

- Blame-Someone-Else Anti-Method
  1. Find a system or environment component you are not responsible for, or cannot observe
  2. Hypothesize that the issue is with that component
  3. Redirect the issue to the responsible team
  4. When proven wrong, go to 1
- Eg, blaming the instance…

## Exonerating the Instance

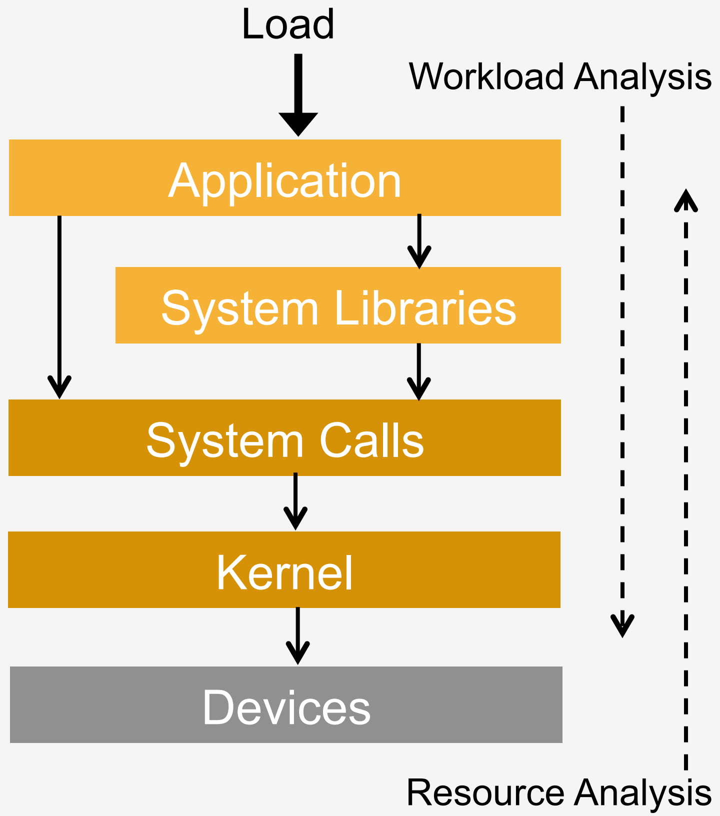
- Observability often exonerates the instance
  - Complex perf issue with no obvious cause: an instance issue?
  - After analysis, sometimes: yes; usually, it's the app (80/20 rule)
- The 80/20 Rule:
  - 80%: Improvement gained by application refactoring and tuning
  - 20%: OS tuning, instance or infrastructure improvement, etc.
- There is also the 20/80 Latency Outlier Rule:
  - 20%: Latency outliers are caused by application code
  - 80%: Caused by instance, crontab, network, JVM GC, etc.

# Or Finding the Monkey

- The Netflix Latency Monkey
  - Injects latency in odd places
  - Part of the Simian Army: a test suite for our fault-tolerant architecture

# Analysis Perspectives

- Workload Analysis:
  - A.k.a. “Top Down Analysis”
  - Begin with application workload, then break down request time.
- Resource Analysis:
  - A.k.a. “Bottom Up Analysis”
  - Begin with resource performance and then their workloads.
  - E.g., the USE Method, followed by workload characterization



## Instance Observability Tools

A. Linux performance tools
B. Netflix custom tools

## A. Linux Performance Tools

1. Statistical tools
2. Profiling tools
3. Tracing tools
4. Hardware counters

## 1. Statistical Tools

- 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
[…]
```

## 2. Profiling Tools

- Profiling: characterizing usage. E.g.:
  - Sampling on-CPU stack traces to explain CPU usage
  - Frequency counting object types to explain memory usage
- Profiling leads to tuning
  - Hot code path -> any related tunables/config?
  - Frequent object allocation -> any way to avoid this?

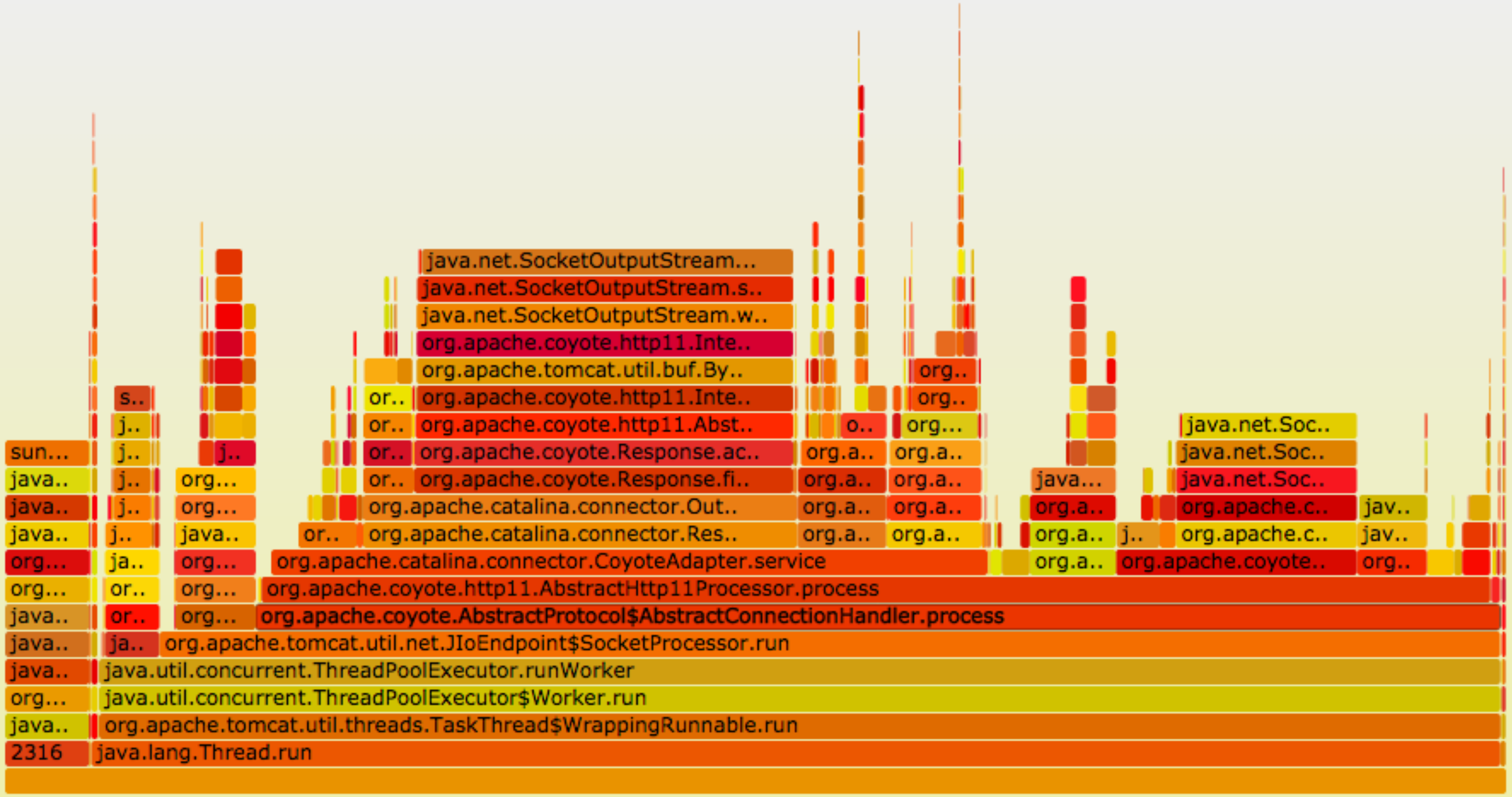
# Profiling Types

- Application profiling
  - Depends on application and language
  - E.g., Java Flight Recorder, Yourkit, Lightweight Java Profiler
  - Many tools are inaccurate or broken; test, verify, cross-check
- System profiling
  - Linux perf_events (the “perf” command)
  - ftrace can do kernel function counts
  - SystemTap has various profiling capabilities

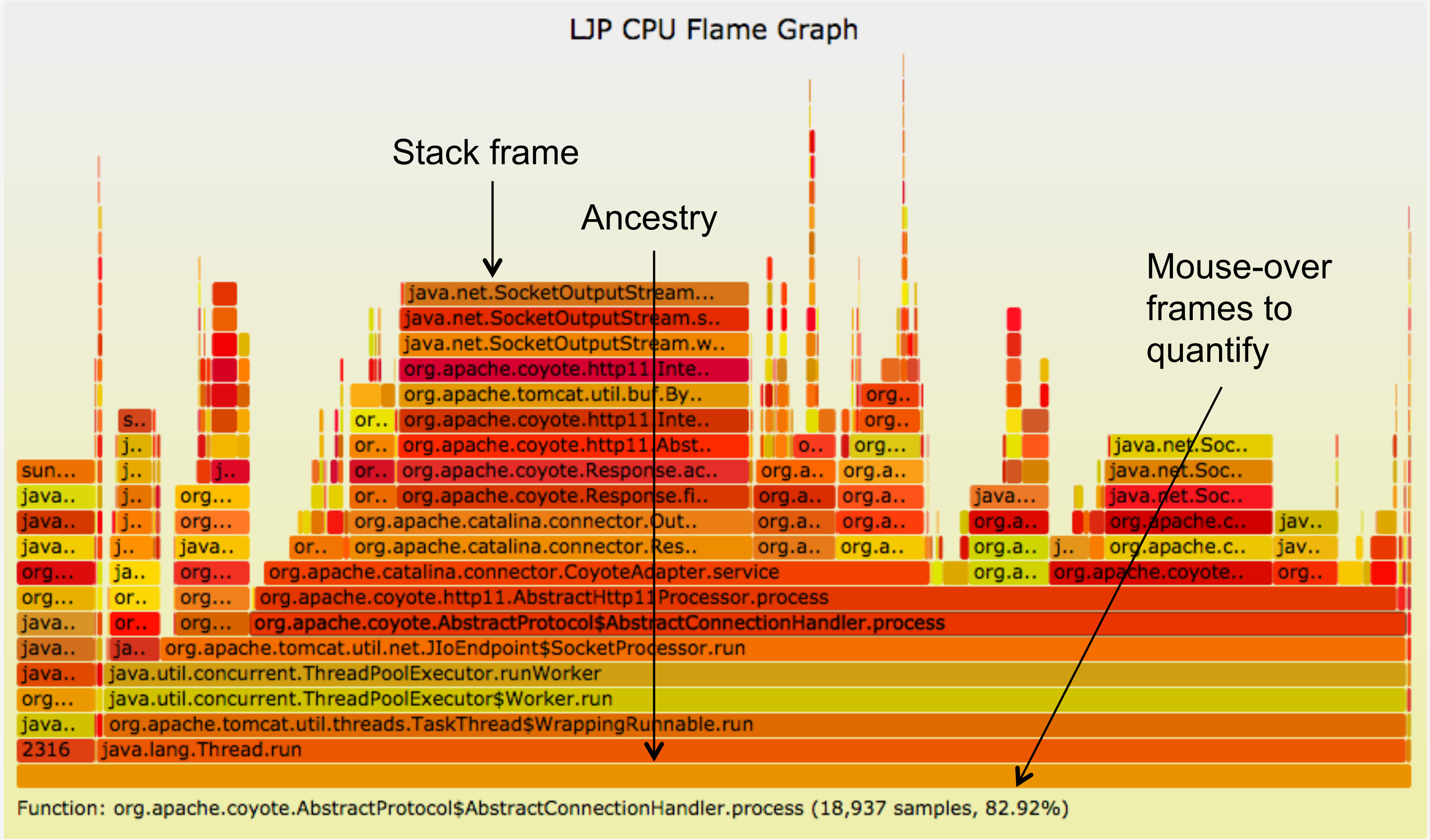
## Application Profiling: LJP

- The Lightweight Java Profiler (LJP)
  - 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
- The profiling output can be visualized as flame graphs

## LJP CPU Flame Graph



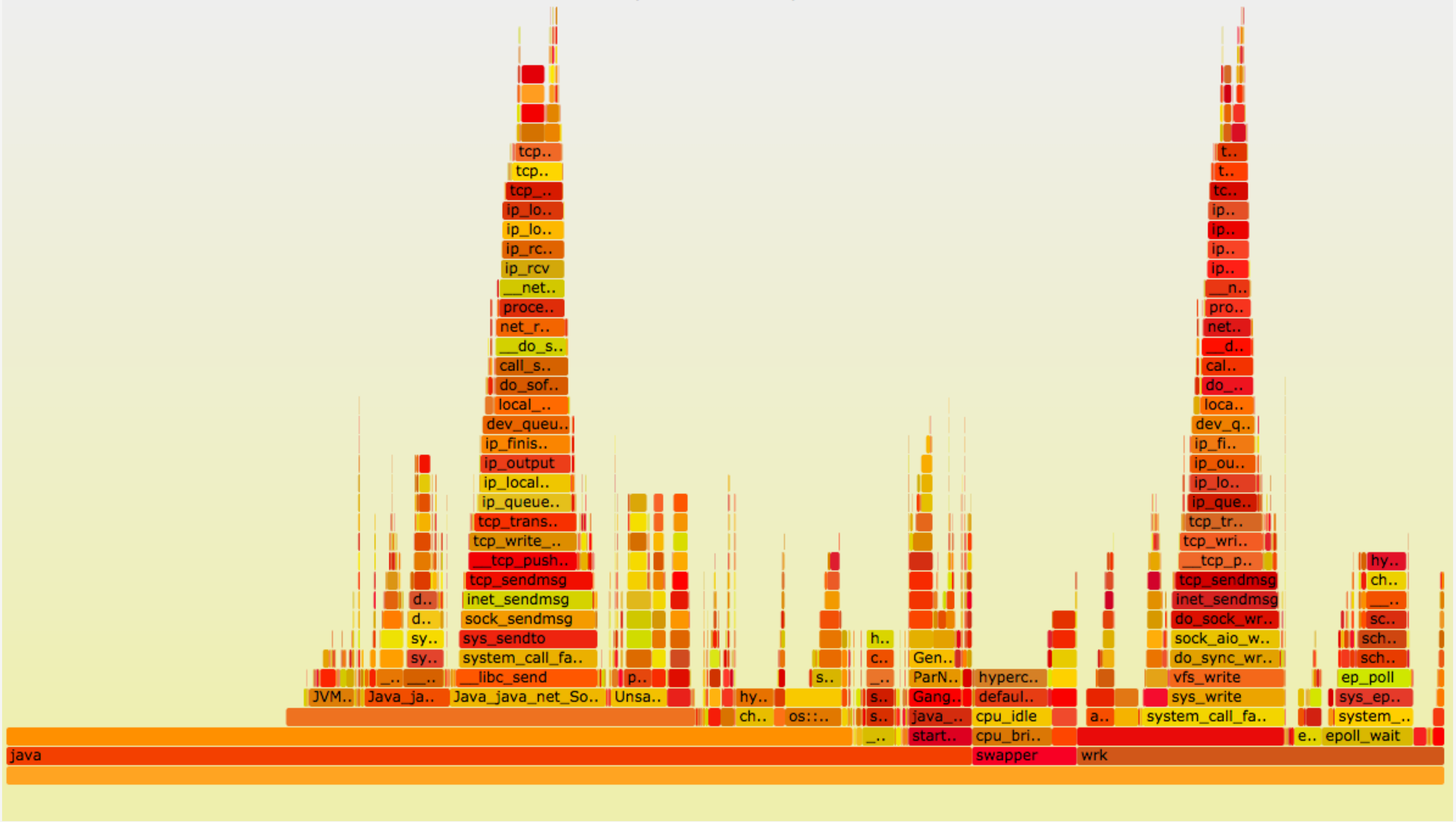
Function: org.apache.coyote.AbstractProtocol$AbstractConnectionHandler.process (18,937 samples, 82.92%)

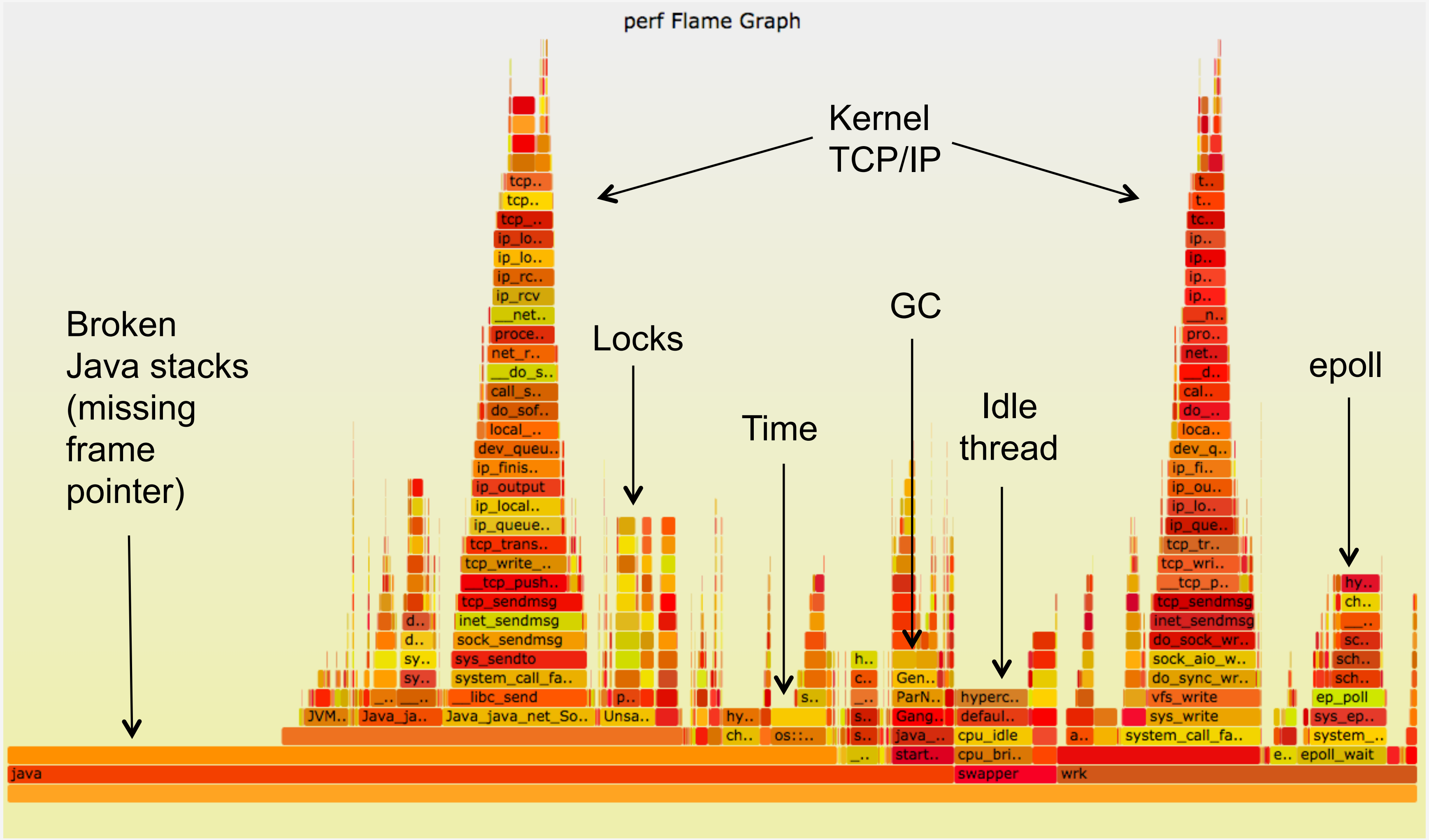

LJP CPU Flame Graph
Stack frame
Ancestry
Mouse-over frames to quantify
java.net.SocketOutputStream...
java.net.SocketOutputStream.s..
java.net.SocketOutputStream.w..
org.apache.coyote.http11.Inte..
org.apache.tomcat.util.buf.By..
org.apache.coyote.http11.Inte..
org.apache.coyote.http11.Abst..
org.apache.coyote.Response.ac..
org.apache.coyote.Response.fi..
org.apache.catalina.connector.Out..
org.apache.catalina.connector.Res..
org.apache.catalina.connector.CoyoteAdapter.service
org.apache.coyote.http11.AbstractHttp11Processor.process
org.apache.coyote.AbstractProtocol$AbstractConnectionHandler.process
org.apache.tomcat.util.net.JIoEndpoint$SocketProcessor.run
java.util.concurrent.ThreadPoolExecutor.runWorker
java.util.concurrent.ThreadPoolExecutor$Worker.run
org.apache.tomcat.util.threads.TaskThread$WrappingRunnable.run
java.lang.Thread.run
2316
java.net.Soc..
org.apache.c..
org.apache.coyote..
Function: org.apache.coyote.AbstractProtocol$AbstractConnectionHandler.process (18,937 samples, 82.92%)

# System Profiling: perf_events

- perf_events sampling CPU stack traces can show:
  - All application logic
    - Whether high level logic can be seen depends on the app and VM
  - JVM internals & libraries
  - The Linux kernel
- 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 Flame Graph
java
swapper
wrk
tcp_sendmsg
inet_sendmsg
sock_sendmsg
sys_sendto
system_call_fa..
__libc_send
Java_java_net_So..
tcp_write_..
tcp_trans..
ip_queue..
ip_local..
ip_output
ip_finis..
dev_queu..
local_..
do_sof..
call_s..
__do_s..
net_r..
proce..
__net..
ip_rcv
ip_rc..
ip_lo..
tcp_..
tcp..
cpu_idle
cpu_bri..
defaul..
hyperc..
start..
java_..
Gang..
ParN..
Gen..
os::..
Unsa..
JVM..
Java_ja..
system_call_fa..
sys_write
vfs_write
do_sync_wr..
sock_aio_w..
do_sock_wr..
inet_sendmsg
tcp_sendmsg
__tcp_p..
tcp_wri..
tcp_tr..
ip_que..
ip_lo..
ip_ou..
ip_fi..
dev_q..
loca..
epoll_wait
system_..
sys_ep..
ep_poll
sch..
sc..
ch..
hy..


perf Flame Graph
Kernel TCP/IP
Broken Java stacks (missing frame pointer)
Locks
Time
GC
Idle thread
epoll
java
swapper
wrk
Java_java_net_So..
sys_sendto
sock_sendmsg
inet_sendmsg
tcp_sendmsg
tcp_push..
tcp_write_..
tcp_trans..
ip_queue..
ip_local..
ip_output
ip_finis..
dev_queu..
local_..
do_sof..
call_s..
__do_s..
net_r..
proce..
__net..
ip_rcv
ip_rc..
ip_lo..
tcp_..
tcp..
__libc_send
system_call_fa..
Unsa..
JVM..
Java_ja..
hyperc..
defaul..
cpu_idle
cpu_bri..
Gen..
ParN..
Gang..
java_..
start..
system_call_fa..
sys_write
vfs_write
do_sync_wr..
sock_aio_w..
do_sock_wr..
ep_poll
sys_ep..
system_..
epoll_wait

## 3. Tracing Tools

- Many system tracers exist for Linux
  1. ftrace
  2. perf_events
  3. eBPF
  4. SystemTap
  5. ktap
  6. LTTng
  7. dtrace4linux
  8. Oracle Linux DTrace
  9. sysdig
- I’ll summarize ftrace & perf_events

## ftrace

- Part of the Linux kernel
  - first added in 2.6.27 (2008), and enhanced in later releases
  - Use directly via /sys/kernel/debug/tracing
  - Already available in all Netflix Linux instances
- Front-end tools aid usage: perf-tools collection
  - https://github.com/brendangregg/perf-tools
  - Unsupported hacks: see WARNINGs
  - Also see the trace-cmd front-end, as well as perf

# ftrace tool: iosnoop

```
# ./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)
[…]
```

# perf_events

- Part of the Linux source
  - Added from linux-tools-common, etc.
- Powerful multi-tool and profiler
  - interval sampling, CPU performance counter events
  - user and kernel dynamic tracing
  - kernel line tracing and local variables (debuginfo)
  - kernel filtering, and in-kernel counts (perf stat)
- Needs kernel debuginfo for advanced uses
  - A difficulty for our instances, since it’s > 100 Mbytes

# perf_events Example

```
# perf record -e skb:consume_skb -ag -- sleep 10
# perf report
[...]
    74.42%  swapper  [kernel.kallsyms]  [k] consume_skb
            |
            --- consume_skb
                arp_process
                arp_rcv
                __netif_receive_skb_core
                __netif_receive_skb
                netif_receive_skb
                virtnet_poll
                net_rx_action
                __do_softirq
                irq_exit
                do_IRQ
                ret_from_intr
[…]
```

← Summarizing stack traces for a tracepoint

perf_events can do many things, it is hard to pick just one example

# 4. Hardware Counters

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

# MSRs

- Can be used to verify real CPU clock rate
  - Can vary with turboboost. Important to know for perf comparisons.
  - Tool from https://github.com/brendangregg/msr-cloud-tools:

```
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
[...]
```

Real CPU MHz

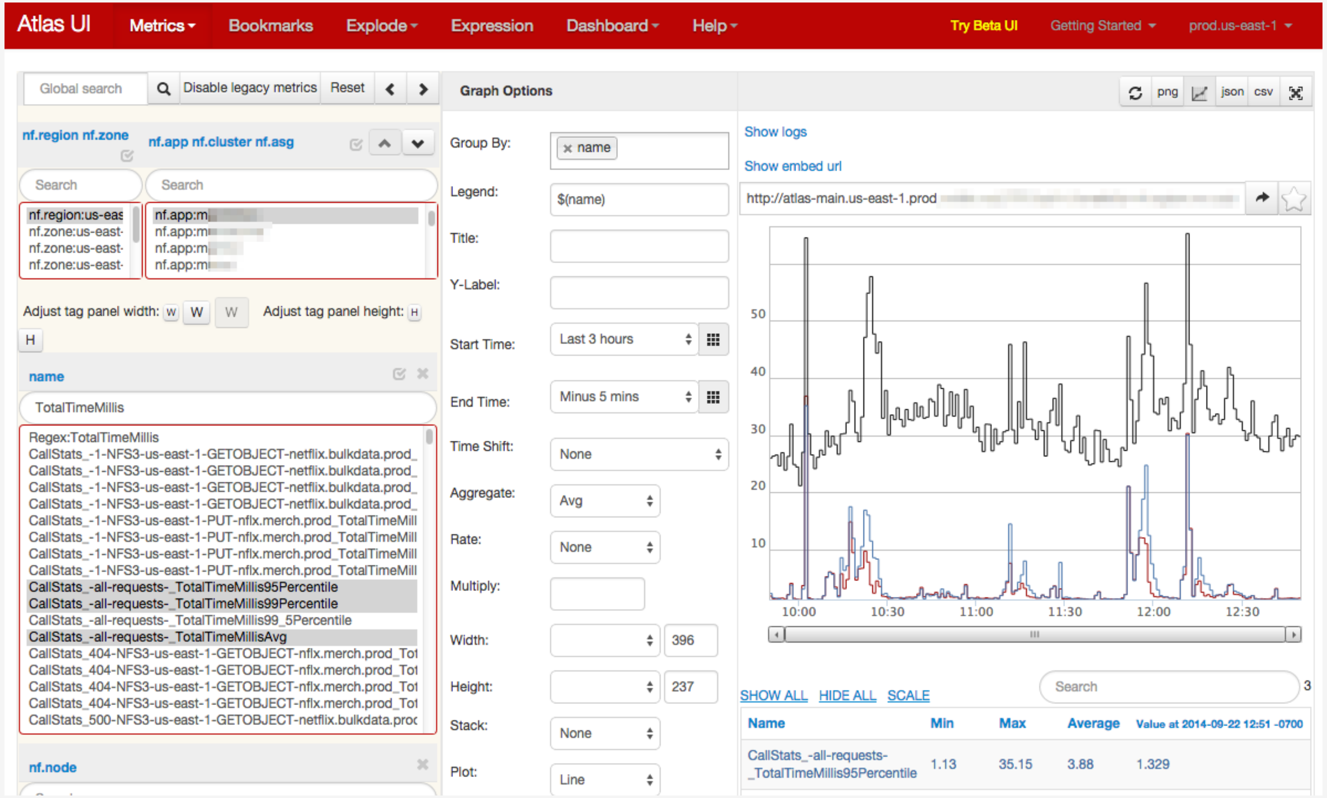
## B. Netflix Online Tools

- Netflix develops many online web-based tools for cloud-wide performance observability
- Per-instance capable tools include:
  - Atlas
  - Vector

# Atlas

Cloud-wide and instance monitoring.

Resource and app metrics, trends.



nf.region nf.zone

nf.app nf.cluster nf.asg

nf.region:us-eas
nf.zone:us-east-
nf.zone:us-east-
nf.zone:us-east-

nf.app:m
nf.app:m
nf.app:m
nf.app:m

name

TotalTimeMillis

Regex:TotalTimeMillis
CallStats_-1-NFS3-us-east-1-GETOBJECT-netflix.bulkdata.prod_
CallStats_-1-NFS3-us-east-1-GETOBJECT-netflix.bulkdata.prod_
CallStats_-1-NFS3-us-east-1-GETOBJECT-netflix.bulkdata.prod_
CallStats_-1-NFS3-us-east-1-GETOBJECT-netflix.bulkdata.prod_
CallStats_-1-NFS3-us-east-1-PUT-nflx.merch.prod_TotalTimeMill
CallStats_-1-NFS3-us-east-1-PUT-nflx.merch.prod_TotalTimeMill
CallStats_-1-NFS3-us-east-1-PUT-nflx.merch.prod_TotalTimeMill
CallStats_-1-NFS3-us-east-1-PUT-nflx.merch.prod_TotalTimeMill
CallStats_-all-requests-_TotalTimeMillis95Percentile
CallStats_-all-requests-_TotalTimeMillis99Percentile
CallStats_-all-requests-_TotalTimeMillis99_5Percentile
CallStats_-all-requests-_TotalTimeMillisAvg
CallStats_404-NFS3-us-east-1-GETOBJECT-nflx.merch.prod_Tot
CallStats_404-NFS3-us-east-1-GETOBJECT-nflx.merch.prod_Tot
CallStats_404-NFS3-us-east-1-GETOBJECT-nflx.merch.prod_Tot
CallStats_404-NFS3-us-east-1-GETOBJECT-nflx.merch.prod_Tot
CallStats_500-NFS3-us-east-1-GETOBJECT-netflix.bulkdata.proc

nf.node

Group By: × name

Legend: $(name)

Title:

Y-Label:

Start Time: Last 3 hours

End Time: Minus 5 mins

Time Shift: None

Aggregate: Avg

Rate: None

Multiply:

Width: 396

Height: 237

Stack: None

Plot: Line

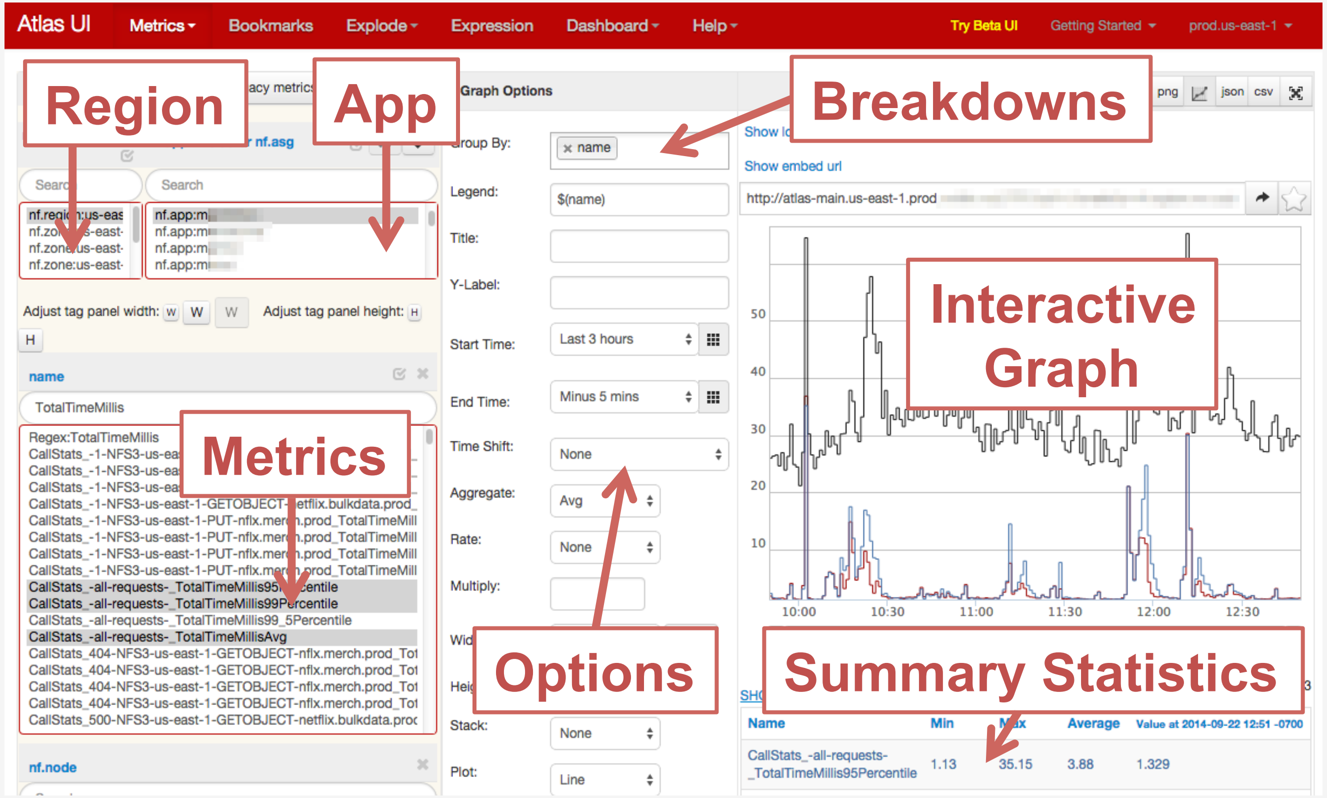
http://atlas-main.us-east-1.prod

| Name | Min | Max | Average | Value at 2014-09-22 12:51 -0700 |
|---|---|---|---|---|
| CallStats_-all-requests-_TotalTimeMillis95Percentile | 1.13 | 35.15 | 3.88 | 1.329 |

# Atlas

Cloud-wide and instance monitoring.

Resource and app metrics, trends.



nf.region:us-eas
nf.zone:us-east-
nf.zone:us-east-
nf.zone:us-east-

nf.asg

Adjust tag panel width: W W W Adjust tag panel height: H H

name

TotalTimeMillis

Regex:TotalTimeMillis
CallStats_-1-NFS3-us-ea
CallStats_-1-NFS3-us-ea
CallStats_-1-NFS3-us-ea
CallStats_-1-NFS3-us-east-1-GETOBJECT-netflix.bulkdata.prod_
CallStats_-1-NFS3-us-east-1-PUT-nflx.merch.prod_TotalTimeMill
CallStats_-1-NFS3-us-east-1-PUT-nflx.merch.prod_TotalTimeMill
CallStats_-1-NFS3-us-east-1-PUT-nflx.merch.prod_TotalTimeMill
CallStats_-1-NFS3-us-east-1-PUT-nflx.merch.prod_TotalTimeMill
CallStats_-all-requests-_TotalTimeMillis95Percentile
CallStats_-all-requests-_TotalTimeMillis99Percentile
CallStats_-all-requests-_TotalTimeMillis99_5Percentile
CallStats_-all-requests-_TotalTimeMillisAvg
CallStats_404-NFS3-us-east-1-GETOBJECT-nflx.merch.prod_Tot
CallStats_404-NFS3-us-east-1-GETOBJECT-nflx.merch.prod_Tot
CallStats_404-NFS3-us-east-1-GETOBJECT-nflx.merch.prod_Tot
CallStats_404-NFS3-us-east-1-GETOBJECT-nflx.merch.prod_Tot
CallStats_500-NFS3-us-east-1-GETOBJECT-netflix.bulkdata.proc

nf.node

Graph Options

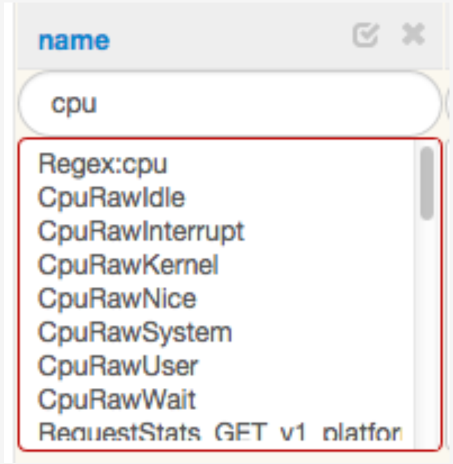
Group By: × name
Legend: $(name)
Title:
Y-Label:
Start Time: Last 3 hours
End Time: Minus 5 mins
Time Shift: None
Aggregate: Avg
Rate: None
Multiply:
Stack: None
Plot: Line

http://atlas-main.us-east-1.prod

| Name | Min | Max | Average | Value at 2014-09-22 12:51 -0700 |
|---|---|---|---|---|
| CallStats_-all-requests-_TotalTimeMillis95Percentile | 1.13 | 35.15 | 3.88 | 1.329 |

## Atlas

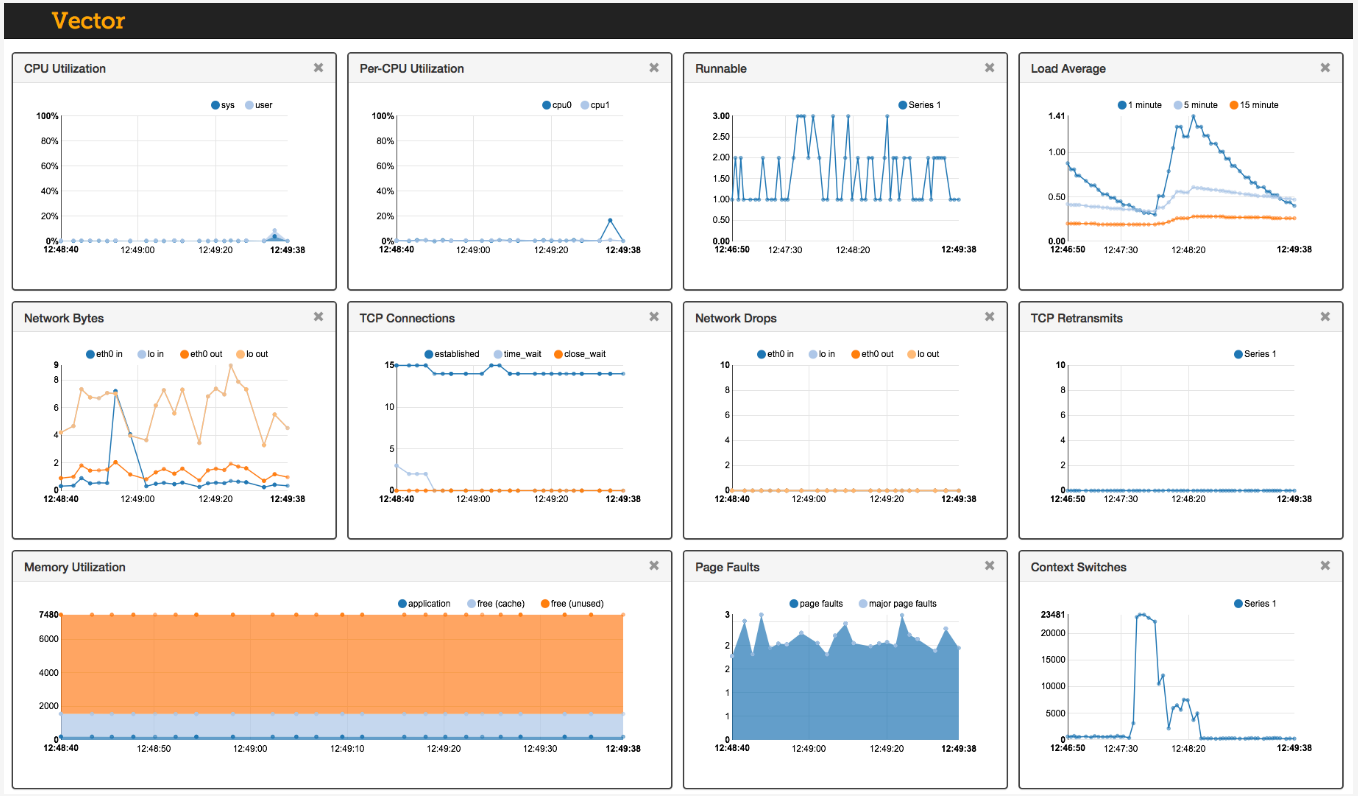
- All metrics in one system
- System metrics:
  - CPU usage, disk I/O, memory…
- Application metrics:
  - latency percentiles, errors…
- Filters or breakdowns by region, application, ASG, metric, instance…
  - Quickly narrow an investigation to an instance or resource type
  - Can identify potential instance type changes from Atlas data



Regex:cpu
CpuRawIdle
CpuRawInterrupt
CpuRawKernel
CpuRawNice
CpuRawSystem
CpuRawUser
CpuRawWait
RequestStats_GET_v1_platfor

# Vector

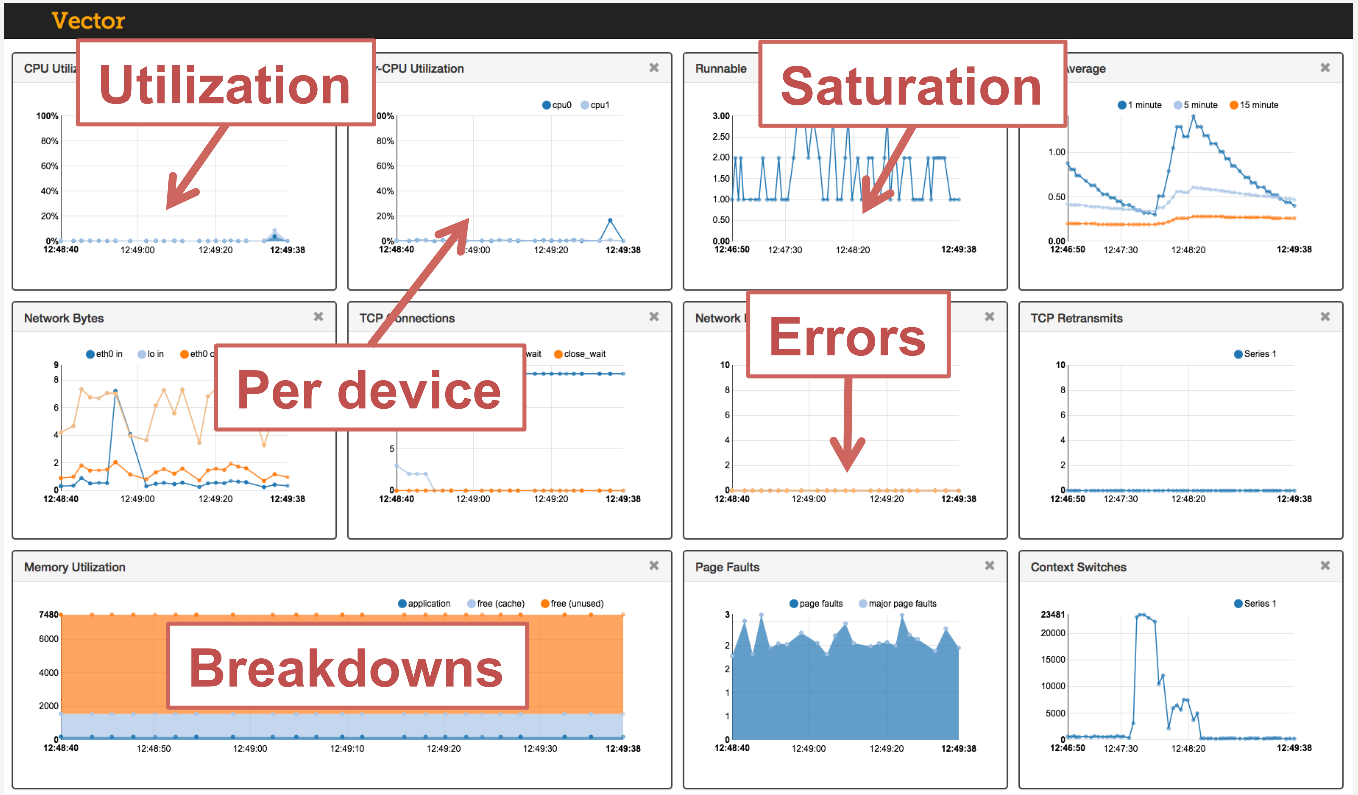
Real-time per-second instance metrics.
On-demand profiling.



# Vector

Real-time per-second instance metrics.

On-demand profiling.



## Vector

- Given an instance, analyze low-level performance
- On-demand: CPU flame graphs, heat maps, ftrace metrics, and SystemTap metrics
- Quick: GUI-driven root cause analysis
- Scalable: other teams can use it easily
- Currently in development (beta-use)

## Summary

1. Instance Selection
2. Amazon EC2 Features
3. Kernel Tuning
4. Observability

## References & Links

- Amazon EC2:
  - http://aws.amazon.com/ec2/instance-types/
  - http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-types.html
  - http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/enhanced-networking.html
- Netflix on EC2:
  - http://www.slideshare.net/cpwatson/cpn302-yourlinuxamioptimizationandperformance
  - http://www.brendangregg.com/blog/2014-09-27/from-clouds-to-roots.html
- Xen Modes:
  - http://www.brendangregg.com/blog/2014-05-07/what-color-is-your-xen.html
- More Performance Analysis:
  - http://www.brendangregg.com/linuxperf.html
  - http://www.slideshare.net/brendangregg/linux-performance-tools-2014
  - http://www.brendangregg.com/USEmethod/use-linux.html
  - http://www.brendangregg.com/blog/2014-06-12/java-flame-graphs.html
  - https://github.com/brendangregg/FlameGraph https://github.com/brendangregg/perf-tools

## Thanks

- Netflix Performance Engineering Team
  - Coburn Watson
  - Scott Emmons: nomogram visualization, insttypes
  - Martin Spier: Vector
  - Amer Ather: tracing, Vector
  - Vadim Filanovsky: AWS cost reporting
- Netflix Insight Engineering Team
  - Roy Rapoport, etc: Atlas
- Amazon Web Services
  - Bryan Nairn, Callum Hughes

# Netflix talks at re:Invent

| Talk | Time | Title |
|---|---|---|
| PFC-305 | Wednesday, 1:15pm | Embracing Failure: Fault Injection and Service Reliability |
| BDT-403 | Wednesday, 2:15pm | Next Generation Big Data Platform at Netflix |
| PFC-306 | Wednesday, 3:30pm | Performance Tuning EC2 |
| DEV-309 | Wednesday, 3:30pm | From Asgard to Zuul, How Netflix's proven Open Source Tools can accelerate and scale your services |
| ARC-317 | Wednesday, 4:30pm | Maintaining a Resilient Front-Door at Massive Scale |
| PFC-304 | Wednesday, 4:30pm | Effective Inter-process Communications in the Cloud: The Pros and Cons of Micro Services Architectures |
| ENT-209 | Wednesday, 4:30pm | Cloud Migration, Dev-Ops and Distributed Systems |
| APP-310 | Friday, 9:00am | Scheduling using Apache Mesos in the Cloud |

AWS re:Invent

Please give us your feedback on this presentation

PFC306

amazon web services

Join the conversation on Twitter with #reinvent