Joyent

# Performance Visualizations

**Brendan Gregg**
*Software Engineer*

brendan.gregg@joyent.com

## G’Day, I’m Brendan

## ... also known as “shouting guy”

# 硬碟也會鬧情緒

Joyent

癮科技小劇場：硬碟也會鬧情緒

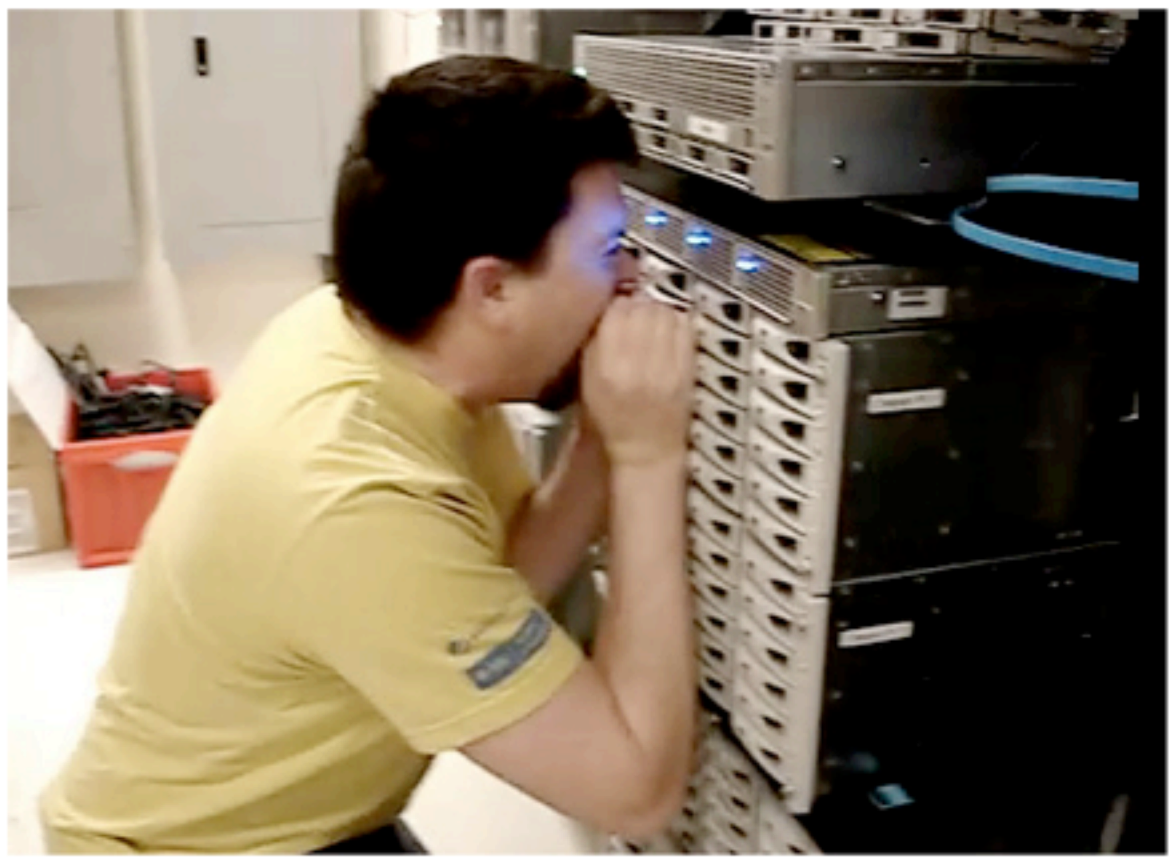
http://chinese.engadget.com/2009/01/05/video-shouting-at-disk-drive-causes-high-latency-low-r

別向硬碟亂叫，會影響它們的心情啊... | 癮科技小劇場：硬碟也會鬧情緒

## 癮科技小劇場：硬碟也會鬧情緒

由 Flow Yu 於 1 year 之前發表

**文章分類:** 儲存裝置

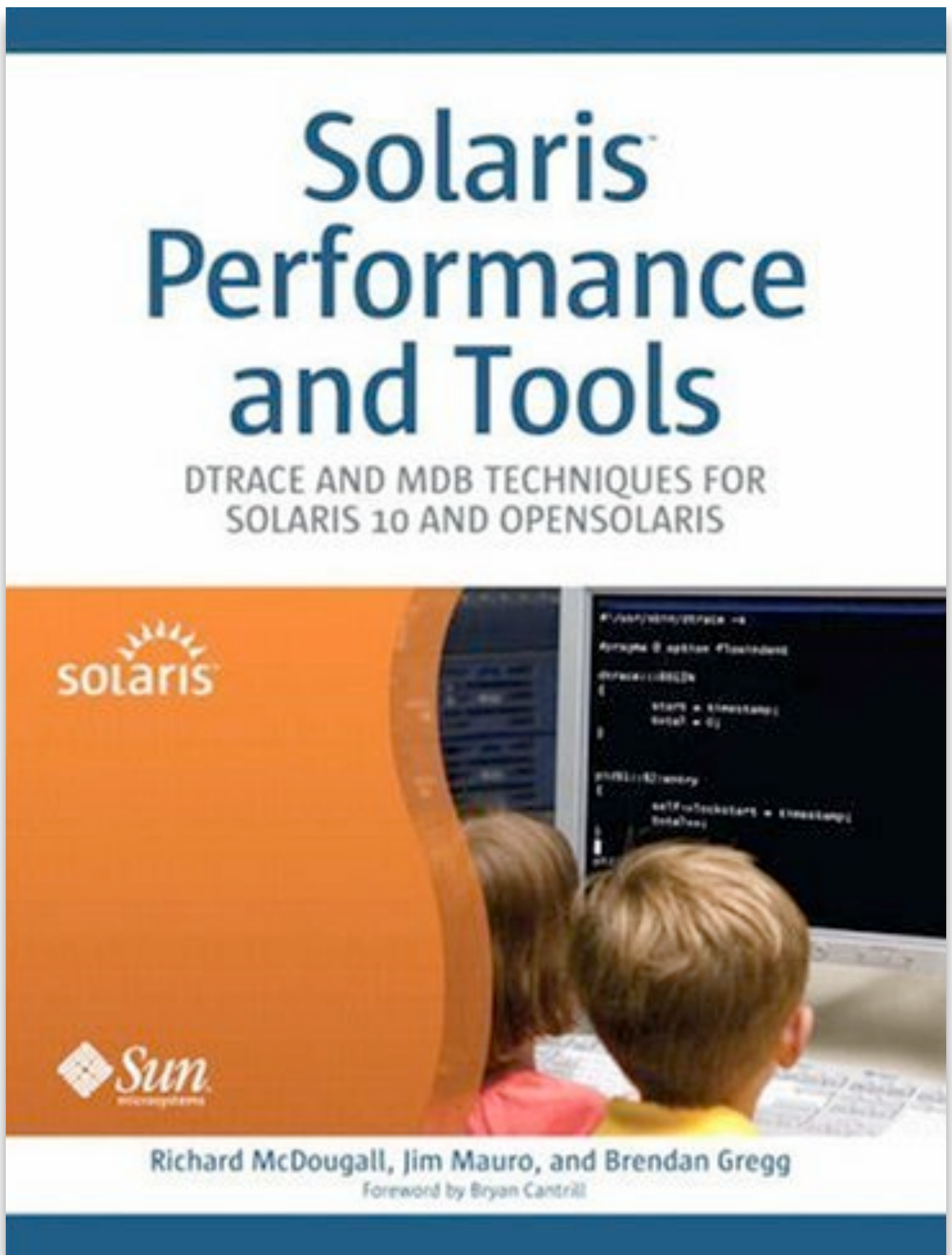


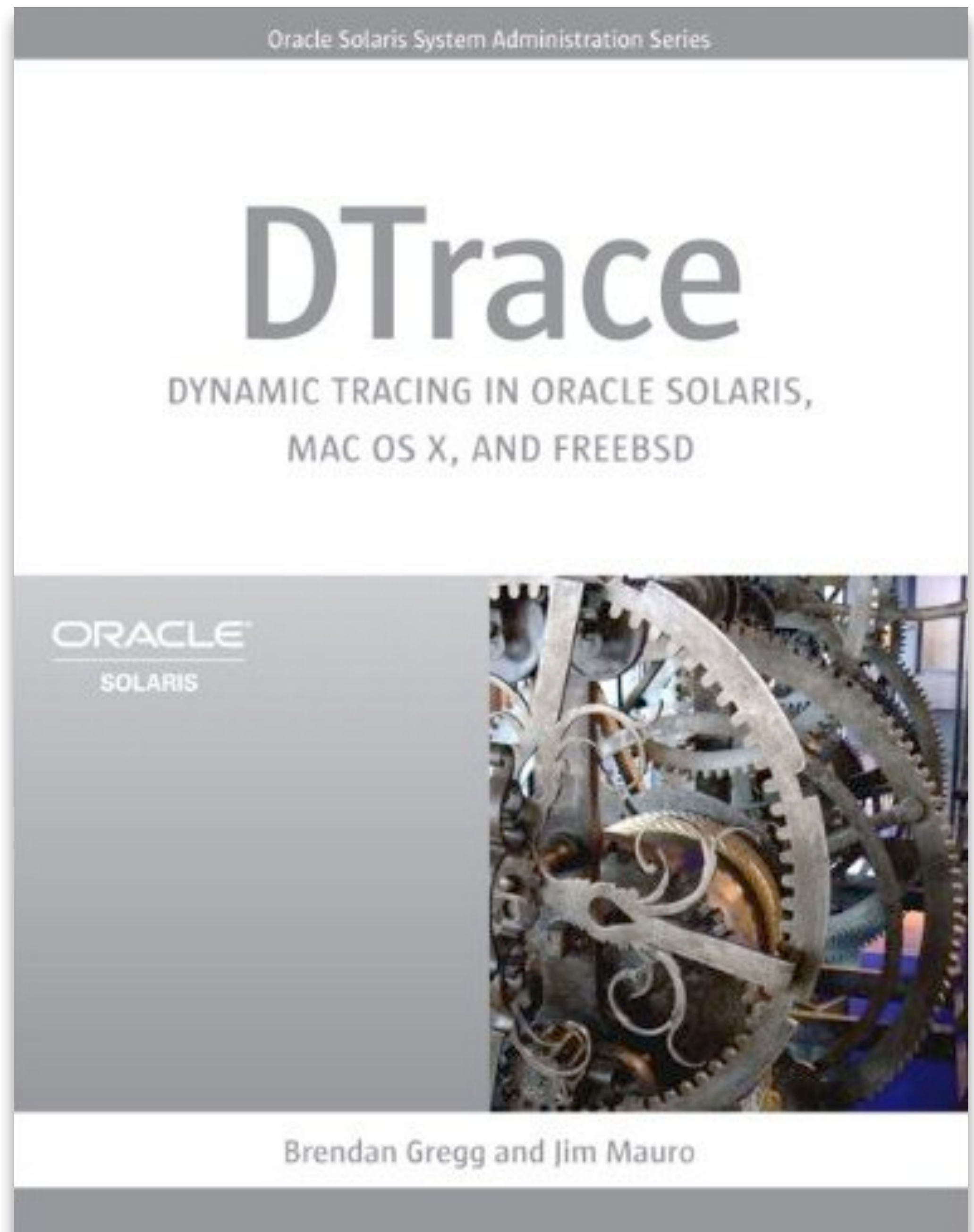
這可不是開玩笑來著，而是經過實驗証明的唷！來自 Sun Fishworks 團隊的 Brendan Gregg，或許是在機房裡待久了，於是突發異想跑去對機房中的磁碟陣列大吼大叫一陣，結果就這麼發現了一個真理，那就是：硬碟就跟員工一樣，吼叫並不會讓他們的效率變高，反而還會讓它們心生不爽而降低士氣，私底下就開始搞罷工。

**I do performance analysis**

**and I’m a DTrace addict**

Joyent


Solaris
Performance
and Tools
DTRACE AND MDB TECHNIQUES FOR
SOLARIS 10 AND OPENSOLARIS
solaris
Sun
Richard McDougall, Jim Mauro, and Brendan Gregg
Foreword by Bryan Cantrill


Oracle Solaris System Administration Series
DTrace
DYNAMIC TRACING IN ORACLE SOLARIS,
MAC OS X, AND FREEBSD
ORACLE
SOLARIS
Brendan Gregg and Jim Mauro

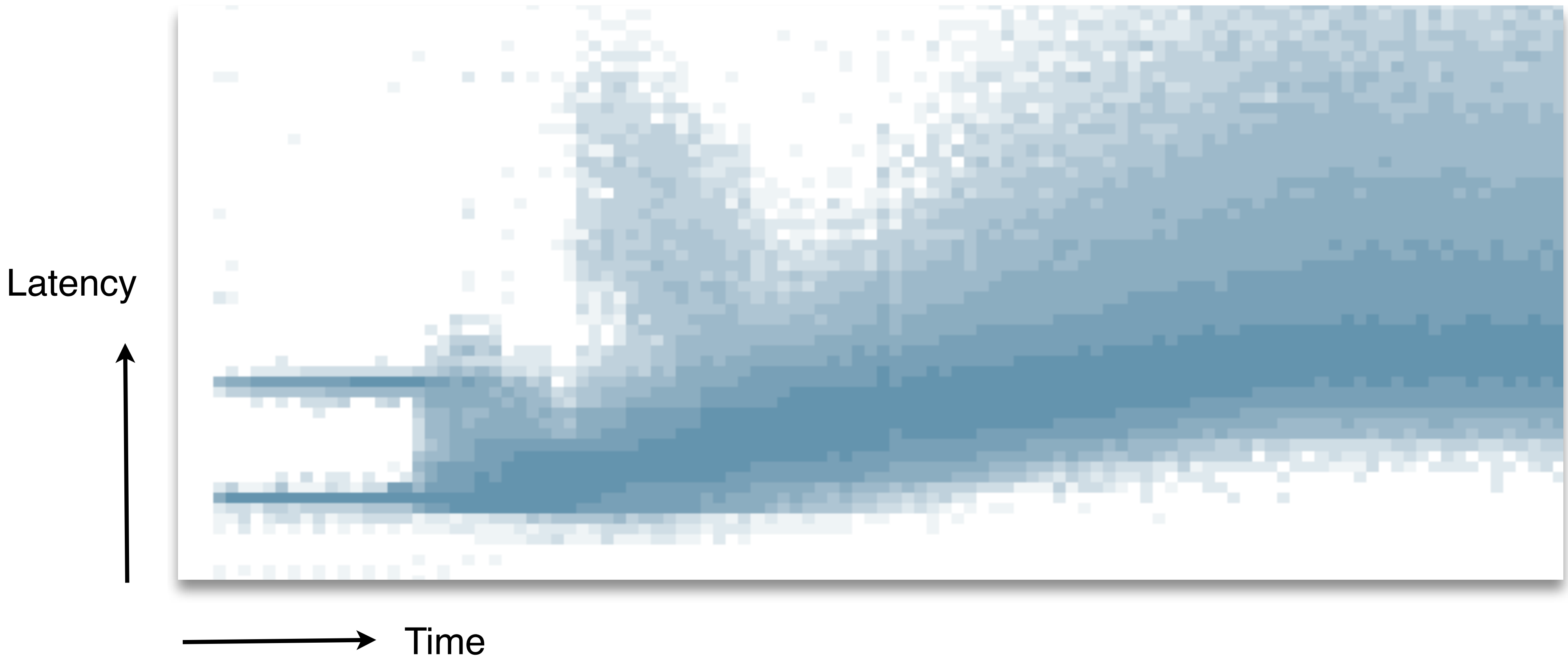
# Agenda

Joyent

- Performance
  - Workload Analysis and Resource Monitoring
  - Understanding available and ideal metrics before plotting
- Visualizations
  - Current examples
    - Latency
    - Utilization
  - Future opportunities
    - Cloud Computing

# Visualizations like these

- The “rainbow pterodactyl”




- ... which needs quite a bit of explanation

# Primary Objectives

- Consider performance metrics before plotting
- See the value of visualizations
- Remember key examples

# Secondary Objectives

- Consider performance metrics before plotting
  - Why studying latency is good
  - ... and studying IOPS can be bad
- See the value of visualizations
  - Why heat maps are needed
  - ... and line graphs can be bad
- Remember key examples
  - I/O latency, as a heat map
  - CPU utilization by CPU, as a heat map

# Content based on

- "Visualizing System Latency", Communications of the ACM July 2010, by Brendan Gregg
- and more

# Performance

Understanding the metrics before we visualize them

# Performance Activities

- Workload analysis
  - Is there an issue? Is an issue real?
  - Where is the issue?
  - Will the proposed fix work? Did it work?
- Resource monitoring
  - How utilized are the environment components?
  - Important activity for capacity planning

# Workload Analysis

- Applied during:
  - software and hardware development
  - proof of concept testing
  - regression testing
  - benchmarking
  - monitoring

# Workload Performance Issues

- Load
- Architecture

# Workload Performance Issues

- Load
  - Workload applied
  - Too much for the system?
  - Poorly constructed?
- Architecture
  - System configuration
  - Software and hardware bugs

# Workload Analysis Steps

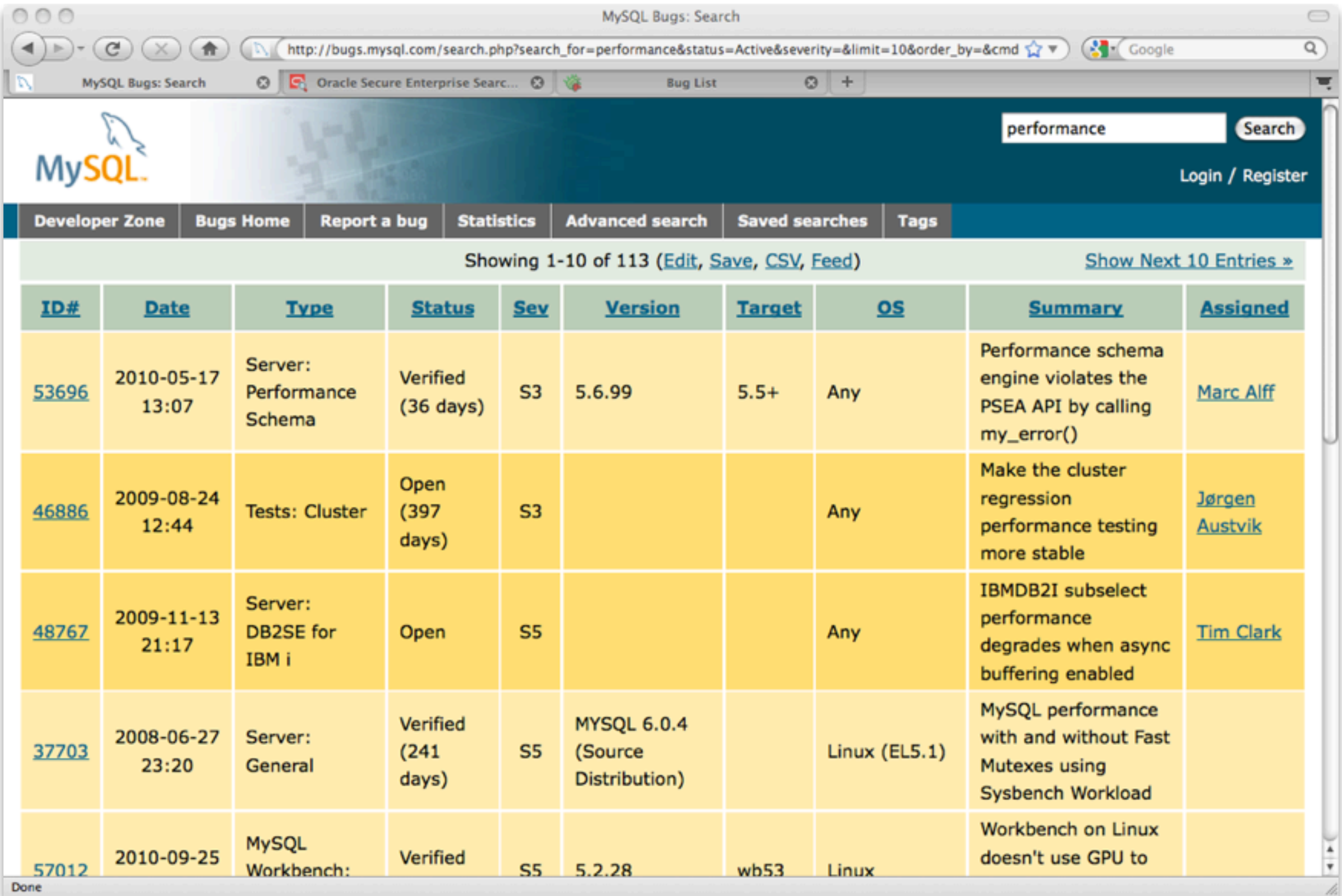
- Identify or confirm if a workload has a performance issue
  - Quantify
- Locate issue
  - Quantify
- Determine, apply and verify solution
  - Quantify

# Quantify

- Finding a performance issue isn't the problem ... it's finding the issue that matters

# bugs.mysql.com “performance”

Joyent



MySQL Bugs: Search

http://bugs.mysql.com/search.php?search_for=performance&status=Active&severity=&limit=10&order_by=&cmd

MySQL Bugs: Search | Oracle Secure Enterprise Searc... | Bug List

MySQL

performance | Search

Login / Register

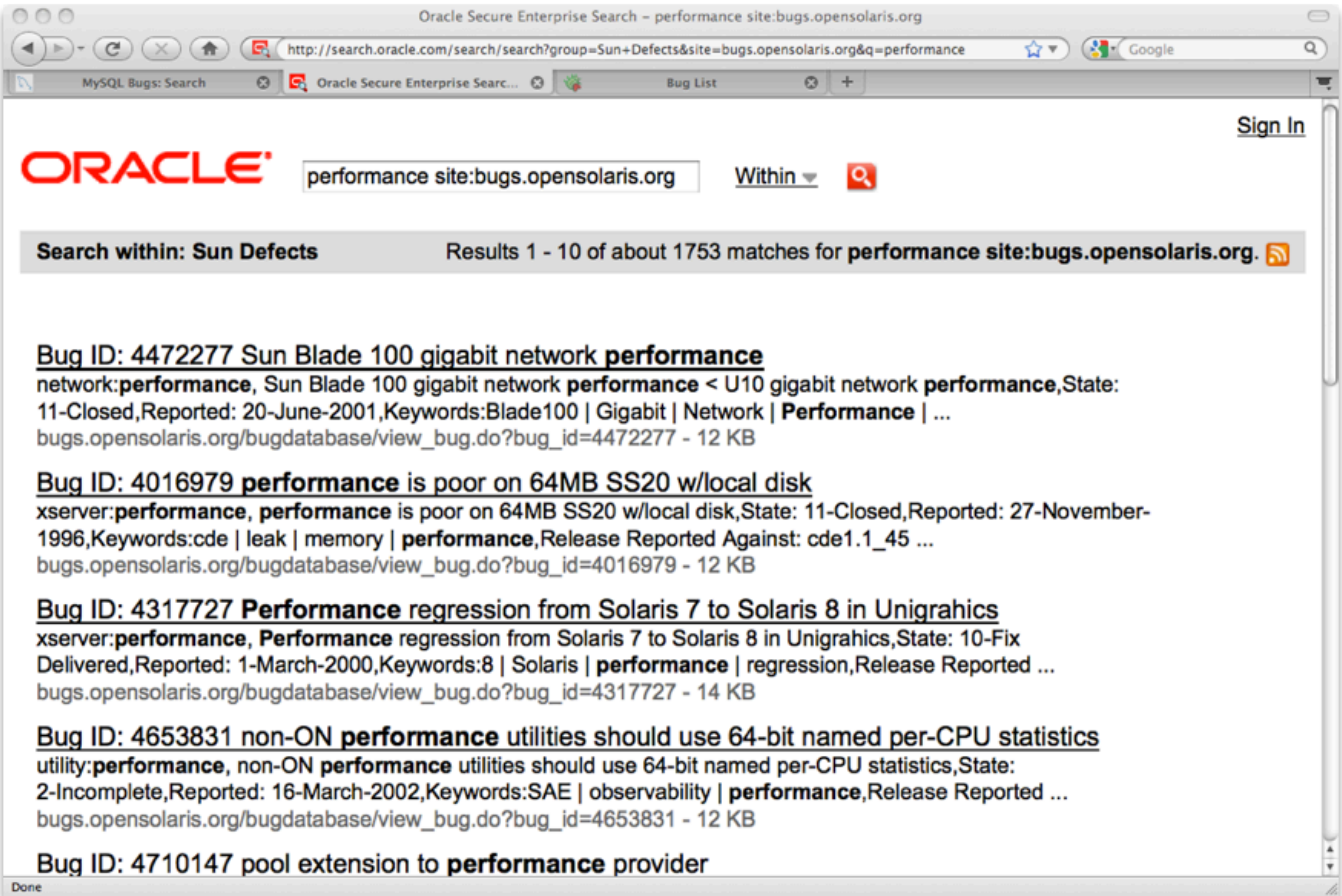
Developer Zone | Bugs Home | Report a bug | Statistics | Advanced search | Saved searches | Tags

Showing 1-10 of 113 (Edit, Save, CSV, Feed) Show Next 10 Entries »

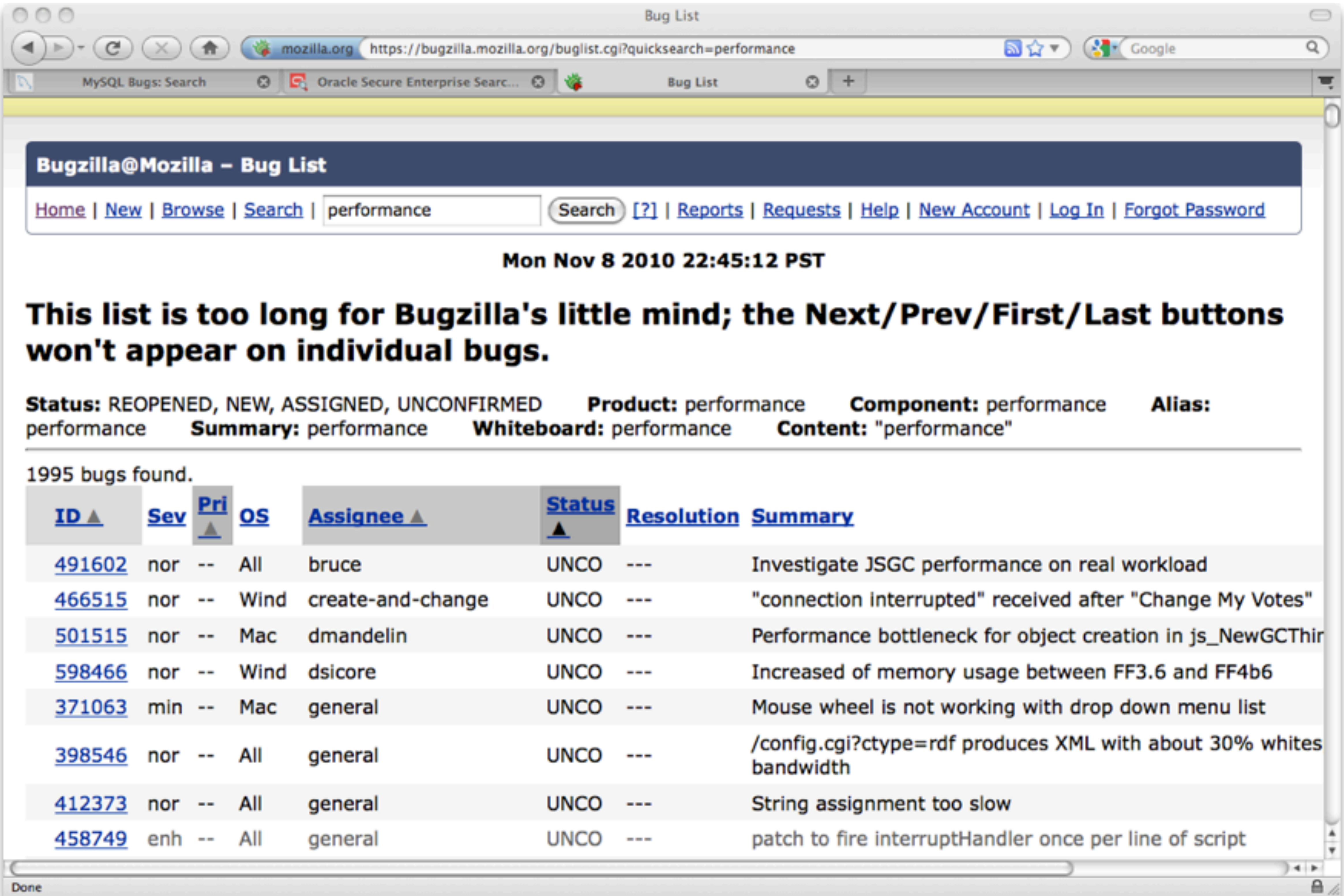
| ID# | Date | Type | Status | Sev | Version | Target | OS | Summary | Assigned |
|---|---|---|---|---|---|---|---|---|---|
| 53696 | 2010-05-17 13:07 | Server: Performance Schema | Verified (36 days) | S3 | 5.6.99 | 5.5+ | Any | Performance schema engine violates the PSEA API by calling my_error() | Marc Alff |
| 46886 | 2009-08-24 12:44 | Tests: Cluster | Open (397 days) | S3 | | | Any | Make the cluster regression performance testing more stable | Jørgen Austvik |
| 48767 | 2009-11-13 21:17 | Server: DB2SE for IBM i | Open | S5 | | | Any | IBMDB2I subselect performance degrades when async buffering enabled | Tim Clark |
| 37703 | 2008-06-27 23:20 | Server: General | Verified (241 days) | S5 | MYSQL 6.0.4 (Source Distribution) | | Linux (EL5.1) | MySQL performance with and without Fast Mutexes using Sysbench Workload | |
| 57012 | 2010-09-25 | MySQL Workbench: | Verified | S5 | 5.2.28 | wb53 | Linux | Workbench on Linux doesn't use GPU to | |

Done

# bugs.opensolaris.org “performance”



# bugs.mozilla.org: “performance”



Bug List

mozilla.org https://bugzilla.mozilla.org/buglist.cgi?quicksearch=performance

MySQL Bugs: Search | Oracle Secure Enterprise Searc... | Bug List

**Bugzilla@Mozilla – Bug List**

Home | New | Browse | Search | performance Search [?] | Reports | Requests | Help | New Account | Log In | Forgot Password

**Mon Nov 8 2010 22:45:12 PST**

## This list is too long for Bugzilla's little mind; the Next/Prev/First/Last buttons won't appear on individual bugs.

**Status:** REOPENED, NEW, ASSIGNED, UNCONFIRMED **Product:** performance **Component:** performance **Alias:** performance **Summary:** performance **Whiteboard:** performance **Content:** "performance"

1995 bugs found.

| ID ▲ | Sev | Pri ▲ | OS | Assignee ▲ | Status ▲ | Resolution | Summary |
|---|---|---|---|---|---|---|---|
| 491602 | nor | -- | All | bruce | UNCO | --- | Investigate JSGC performance on real workload |
| 466515 | nor | -- | Wind | create-and-change | UNCO | --- | "connection interrupted" received after "Change My Votes" |
| 501515 | nor | -- | Mac | dmandelin | UNCO | --- | Performance bottleneck for object creation in js_NewGCThir |
| 598466 | nor | -- | Wind | dsicore | UNCO | --- | Increased of memory usage between FF3.6 and FF4b6 |
| 371063 | min | -- | Mac | general | UNCO | --- | Mouse wheel is not working with drop down menu list |
| 398546 | nor | -- | All | general | UNCO | --- | /config.cgi?ctype=rdf produces XML with about 30% whites bandwidth |
| 412373 | nor | -- | All | general | UNCO | --- | String assignment too slow |
| 458749 | enh | -- | All | general | UNCO | --- | patch to fire interruptHandler once per line of script |

Done

# “performance” bugs

- ... and those are just the known performance bugs
- ... and usually only of a certain type (architecture)

# How to Quantify

- Observation based
  - Choose a reliable metric
  - Estimate performance gain from resolving issue
- Experimentation based
  - Apply fix
  - Quantify before vs. after using a reliable metric

# Observation based

- For example:
    - Observed: application I/O takes 10 ms
    - Observed: 9 ms of which is disk I/O
    - Suggestion: replace disks with flash-memory based SSDs, with an expected latency of ~100 us
    - Estimated gain: 10 ms -> 1.1 ms (10 ms - 9 ms + 0.1 ms) =~ 9x gain
- Very useful - but not possible without accurate quantification

# Experimentation based

- For example:
  - Observed: Application transaction latency average 10 ms
  - Experiment: Added more DRAM to increase cache hits and reduce average latency
  - Observed: Application transaction latency average 2 ms
  - Gain: 10 ms -> 2 ms = 5x

- Also very useful - but risky without accurate quantification

# Metrics to Quantify Performance

- Choose reliable metrics to quantify performance:
  - IOPS
  - transactions/second
  - throughput
  - utilization
  - latency
- Ideally
  - interpretation is straightforward
  - reliable

# Metrics to Quantify Performance

- Choose reliable metrics to quantify performance:
  - IOPS
  - transactions/second
  - throughput
  - utilization
  - latency

generally better suited for:

Capacity Planning

Workload Analysis

- Ideally
  - interpretation is straightforward
  - reliable

# Metrics Availability

Joyent

- Ideally (given the luxury of time):
  - design the desired metrics
  - then see if they exist, or,
  - implement them (eg, DTrace)
- Non-ideally
  - see what already exists
  - make-do (eg, vmstat -> gnuplot)

# Assumptions to avoid

- Available metrics are implemented correctly
  - all software has bugs
    - eg, CR: 6687884 nxge rbytes and obytes kstat are wrong
  - trust no metric without double checking from other sources
- Available metrics are designed by performance experts
  - sometimes added by the programmer to only debug their code
- Available metrics are complete
  - you won’t always find what you really need

# Getting technical

- This will be explained using two examples:
  - Workload Analysis
  - Capacity Planning

# Example: Workload Analysis

- Quantifying performance issues with IOPS vs latency
  - IOPS is commonly presented by performance analysis tools
  - eg: disk IOPS via kstat, SNMP, iostat, ...

# IOPS

Joyent

- Depends on where the I/O is measured
  - app -> library -> syscall -> VFS -> filesystem -> RAID -> device
- Depends on what the I/O is
  - synchronous or asynchronous
  - random or sequential
  - size
- Interpretation difficult
  - what value is good or bad?
  - is there a max?

# Some disk IOPS problems

- IOPS Inflation
  - Library or Filesystem prefetch/read-ahead
  - Filesystem metadata
  - RAID stripes
- IOPS Deflation
  - Read caching
  - Write cancellation
  - Filesystem I/O aggregation
- IOPS aren’t created equal

# IOPS example: iostat -xnz 1

Joyent

- Consider this disk: 86 IOPS == 99% busy

```
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
   86.6    0.0  655.5    0.0  0.0  1.0    0.0   11.5   0  99 c1d0
```

- Versus this disk: 21,284 IOPS == 99% busy

```
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
 21284.7    0.0 10642.4    0.0  0.0  1.8    0.0    0.1   2  99 c1d0
```

# IOPS example: iostat -xnz 1

- Consider this disk: 86 IOPS == 99% busy

```
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
   86.6    0.0  655.5    0.0  0.0  1.0    0.0   11.5   0  99 c1d0
```

- Versus this disk: 21,284 IOPS == 99% busy

```
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
 21284.7    0.0 10642.4    0.0  0.0  1.8    0.0    0.1   2  99 c1d0
```

- ... they are the same disk, different I/O types
  - 1) 8 Kbyte random
  - 2) 512 byte sequential (on-disk DRAM cache)

# Using IOPS to quantify issues

- to identify
  - is 100 IOPS an problem?  Per disk?
- to locate
  - 90% of IOPS are random. Is that the problem?
- to verify
  - A filesystem tunable caused IOPS to reduce.
    Has this fixed the issue?

# Using IOPS to quantify issues

- to identify
  - is 100 IOPS an problem?  Per disk?  (depends...)
- to locate
  - 90% of IOPS are random. Is that the problem?  (depends...)
- to verify
  - A filesystem tunable caused IOPS to reduce.
    Has this fixed the issue?  (probably, assuming...)
- We can introduce more metrics to understand these, but standalone IOPS is tricky to interpret

# Using latency to quantify issues

- to identify
  - is a 100ms I/O a problem?
- to locate
  - 90ms of the 100ms is lock contention. Is that the problem?
- to verify
  - A filesystem tunable caused the I/O latency to reduce to 1ms. Has this fixed the issue?

# Using latency to quantify issues

- to identify
  - is a 100ms I/O a problem? (probably - if synchronous to the app.)
- to locate
  - 90ms of the 100ms is lock contention. Is that the problem? (yes)
- to verify
  - A filesystem tunable caused the I/O latency to reduce to 1ms. Has this fixed the issue? (probably - if 1ms is acceptable)
- Latency is much more reliable, easier to interpret

# Latency

- Time from I/O or transaction request to completion
- Synchronous latency has a direct impact on performance
  - Application is waiting
  - higher latency == worse performance
- Not all latency is synchronous:
  - Asynchronous filesystem threads flushing dirty buffers to disk
    eg, zfs TXG synchronous thread
  - Filesystem prefetch
    no one is waiting at this point
  - TCP buffer and congestion window: individual packet latency may be high, but pipe is kept full for good throughput performance

# Turning other metrics into latency

- Currency converter (* -> ms):
  - random disk IOPS == I/O service latency
  - disk saturation == I/O wait queue latency
  - CPU utilization == code path execution latency
  - CPU saturation == dispatcher queue latency
  - ...
- Quantifying as latency allows different components to be compared, ratios examined, improvements estimated.

# Example: Resource Monitoring

- Different performance activity
  - Focus is environment components, not specific issues
  - incl. CPUs, disks, network interfaces, memory, I/O bus, memory bus, CPU interconnect, I/O cards, network switches, etc.
  - Information is used for capacity planning
  - Identifying future issues before they happen
- Quantifying resource monitoring with IOPS vs utilization

# IOPS vs Utilization

- Another look at this disk:

```
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
   86.6    0.0  655.5    0.0  0.0  1.0    0.0   11.5   0  99 c1d0
[...]
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
 21284.7    0.0 10642.4    0.0  0.0  1.8    0.0    0.1   2  99 c1d0
```

- Q. does this system need more spindles for IOPS capacity?

# IOPS vs Utilization

- Another look at this disk:

```
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
   86.6    0.0  655.5    0.0  0.0  1.0    0.0   11.5   0  99 c1d0
[...]
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
 21284.7    0.0 10642.4    0.0  0.0  1.8    0.0    0.1   2  99 c1d0
```

- Q. does this system need more spindles for IOPS capacity?
  - IOPS (r/s + w/s): ???
  - Utilization (%b): yes (even considering NCQ)

# IOPS vs Utilization

Joyent

- Another look at this disk:

```
                              extended device statistics
    r/s     w/s    kr/s     kw/s wait actv wsvc_t asvc_t  %w  %b device
   86.6     0.0   655.5      0.0  0.0  1.0    0.0   11.5   0  99 c1d0
[...]
                              extended device statistics
    r/s     w/s    kr/s     kw/s wait actv wsvc_t asvc_t  %w  %b device
 21284.7     0.0 10642.4      0.0  0.0  1.8    0.0    0.1   2  99 c1d0
```

- Q. does this system need more spindles for IOPS capacity?
  - IOPS (r/s + w/s): ???
  - Utilization (%b): yes (even considering NCQ)
  - Latency (wsvc_t): no
- Latency will identify the issue once it is an issue; utilization will forecast the issue - capacity planning

# Performance Summary

- Metrics matter - need to reliably quantify performance
  - to identify, locate, verify
  - try to think, design
- Workload Analysis
  - latency
- Resource Monitoring
  - utilization
- Other metrics are useful to further understand the nature of the workload and resource behavior

# Objectives

- Consider performance metrics before plotting
  - Why latency is good
  - ... and IOPS can be bad
- See the value of visualizations
  - Why heat maps are needed
  - ... and line graphs can be bad
- Remember key examples
  - I/O latency, as a heat map
  - CPU utilization by CPU, as a heat map

# Visualizations

Current Examples

Latency

# Visualizations

Joyent

- So far we’ve picked:
- Latency
    - for workload analysis
- Utilization
    - for resource monitoring

# Latency

- For example, disk I/O
- Raw data looks like this:

```
# iosnoop -o
DTIME        UID    PID D     BLOCK    SIZE       COMM PATHNAME
125          100    337 R     72608    8192       bash /usr/sbin/tar
138          100    337 R     72624    8192       bash /usr/sbin/tar
127          100    337 R     72640    8192       bash /usr/sbin/tar
135          100    337 R     72656    8192       bash /usr/sbin/tar
118          100    337 R     72672    8192       bash /usr/sbin/tar
108          100    337 R     72688    4096       bash /usr/sbin/tar
87           100    337 R     72696    3072       bash /usr/sbin/tar
9148         100    337 R    113408    8192        tar /etc/default/lu
8806         100    337 R    104738    7168        tar /etc/default/lu
2262         100    337 R     13600    1024        tar /etc/default/cron
76           100    337 R     13616    1024        tar /etc/default/devfsadm
```

[...many pages of output...]

- iosnoop is DTrace based
    - examines latency for every disk (back end) I/O

# Latency Data

- tuples
  - I/O completion time
  - I/O latency
- can be 1,000s of these per second

# Summarizing Latency

- iostat(1M) can show per second average:

```
$ iostat -xnz 1
[...]
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
  471.0    7.0  786.1   12.0  0.1  1.2    0.2    2.5   4  90 c1d0
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
  631.0    0.0 1063.1    0.0  0.2  1.0    0.3    1.6   9  92 c1d0
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
  472.0    0.0  529.0    0.0  0.0  1.0    0.0    2.1   0  94 c1d0
[...]
```

# Per second

- Condenses I/O completion time
- Almost always a sufficient resolution
    - (So far I've only had one case where examining raw completion time data was crucial: an interrupt coalescing bug)

# Average/second

- Average loses latency outliers
- Average loses latency distribution
- ... but not disk distribution:

```
$ iostat -xnz 1
[...]
                    extended device statistics
    r/s    w/s   kr/s   kw/s wait actv wsvc_t asvc_t  %w  %b device
   43.9    0.0  351.5    0.0  0.0  0.4    0.0   10.0   0  34 c0t5000CCA215C46459d0
   47.6    0.0  381.1    0.0  0.0  0.5    0.0    9.8   0  36 c0t5000CCA215C4521Dd0
   42.7    0.0  349.9    0.0  0.0  0.4    0.0   10.1   0  35 c0t5000CCA215C45F89d0
   41.4    0.0  331.5    0.0  0.0  0.4    0.0    9.6   0  32 c0t5000CCA215C42A4Cd0
   45.6    0.0  365.1    0.0  0.0  0.4    0.0    9.2   0  34 c0t5000CCA215C45541d0
   45.0    0.0  360.3    0.0  0.0  0.4    0.0    9.4   0  34 c0t5000CCA215C458F1d0
   42.9    0.0  343.5    0.0  0.0  0.4    0.0    9.9   0  33 c0t5000CCA215C450E3d0
   44.9    0.0  359.5    0.0  0.0  0.4    0.0    9.3   0  35 c0t5000CCA215C45323d0
[...]
```

- only because iostat(1M) prints this per-disk
    - but that gets hard to read for 100s of disks, per second!

# Latency outliers

Joyent

- Occasional high-latency I/O
- Can be the sole reason for performance issues
- Can be lost in an average
  - 10,000 fast I/O @ 1ms
  - 1 slow I/O @ 500ms
  - average = 1.05 ms
- Can be seen using max instead of (or as well as) average

# Maximum/second

- iostat(1M) doesn't show this, however DTrace can
- can be visualized along with average/second
- does identify outliers
- doesn't identify latency distribution details

# Latency distribution

- Apart from outliers and average, it can be useful to examine the full profile of latency - all the data.
  - For such a crucial metric, keep as much details as possible
- For latency, distributions we'd expect to see include:
  - bi-modal: cache hit vs cache miss
  - tri-modal: multiple cache layers
  - flat: random disk I/O

# Latency Distribution Example

Joyent

- Using DTrace:

```
# ./disklatency.d
Tracing... Hit Ctrl-C to end.
^C
   sd4 (28,256), us:

           value  ------------- Distribution ------------- count
              16 |                                         0
              32 |                                         82
              64 |@@@                                      621
             128 |@@@@@                                    833
             256 |@@@@                                     641
             512 |@@@                                      615
            1024 |@@@@@@@                                  1239
            2048 |@@@@@@@@@                                1615
            4096 |@@@@@@@@                                 1483
            8192 |                                         76
           16384 |                                         1
           32768 |                                         0
           65536 |                                         2
          131072 |                                         0
```

# disklatency.d

- not why we are here, but before someone asks...

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN
{
        printf("Tracing... Hit Ctrl-C to end.\n");
}

io:::start
{
        start_time[arg0] = timestamp;
}

io:::done
/this->start = start_time[arg0]/
{
        this->delta = (timestamp - this->start) / 1000;
        @[args[1]->dev_statname, args[1]->dev_major, args[1]->dev_minor] =
            quantize(this->delta);
        start_time[arg0] = 0;
}

dtrace:::END
{
        printa("   %s (%d,%d), us:\n%@d\n", @);
}
```

# Latency Distribution Example

Joyent

```
# ./disklatency.d
Tracing... Hit Ctrl-C to end.
^C
   sd4 (28,256), us:

           value  ------------- Distribution ------------- count
              16 |                                         0
              32 |                                         82
              64 |@@@                                      621
             128 |@@@@@                                    833
             256 |@@@@                                     641
             512 |@@@                                      615
            1024 |@@@@@@@                                  1239
            2048 |@@@@@@@@@                                1615
            4096 |@@@@@@@@                                 1483
            8192 |                                         76
           16384 |                                         1
           32768 |                                         0
           65536 |                                         2
          131072 |                                         0
```

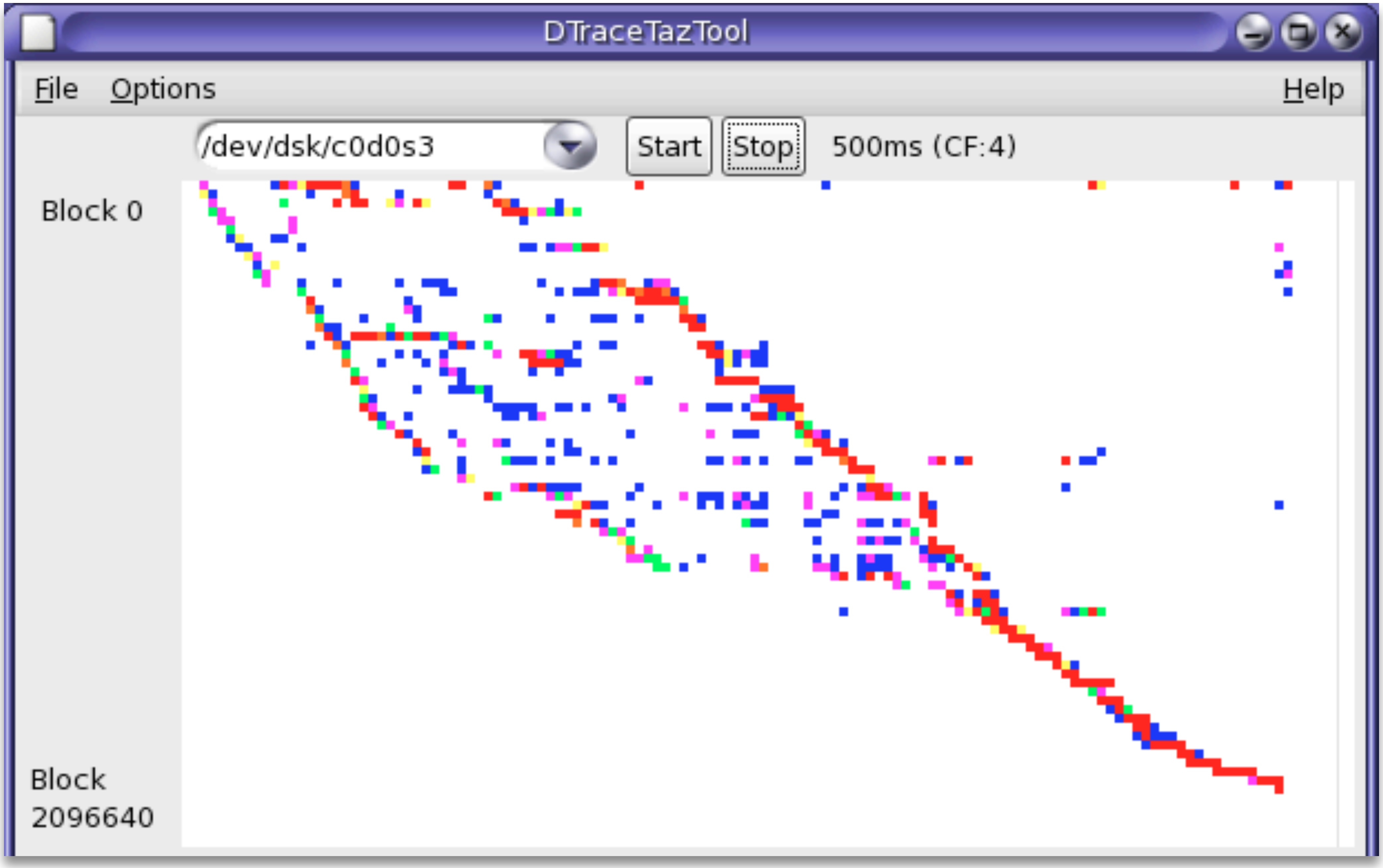
65 - 131 ms outliers

- ... but can we see this distribution per second?
- ... how do we visualize a 3rd dimension?

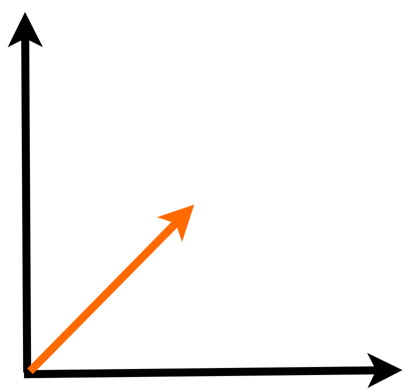
# Column Quantized Visualization
# aka “heat map”

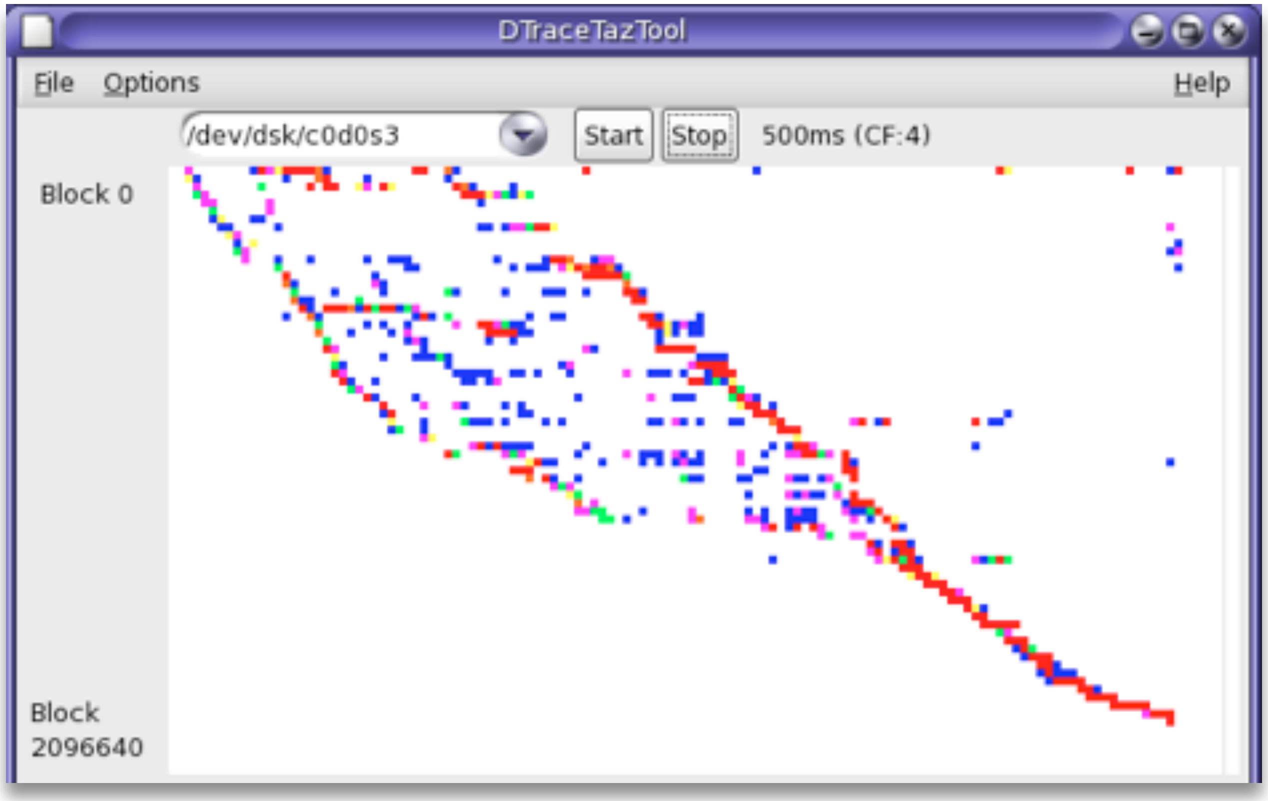
Joyent

- For example:



# Heat Map: Offset Distribution



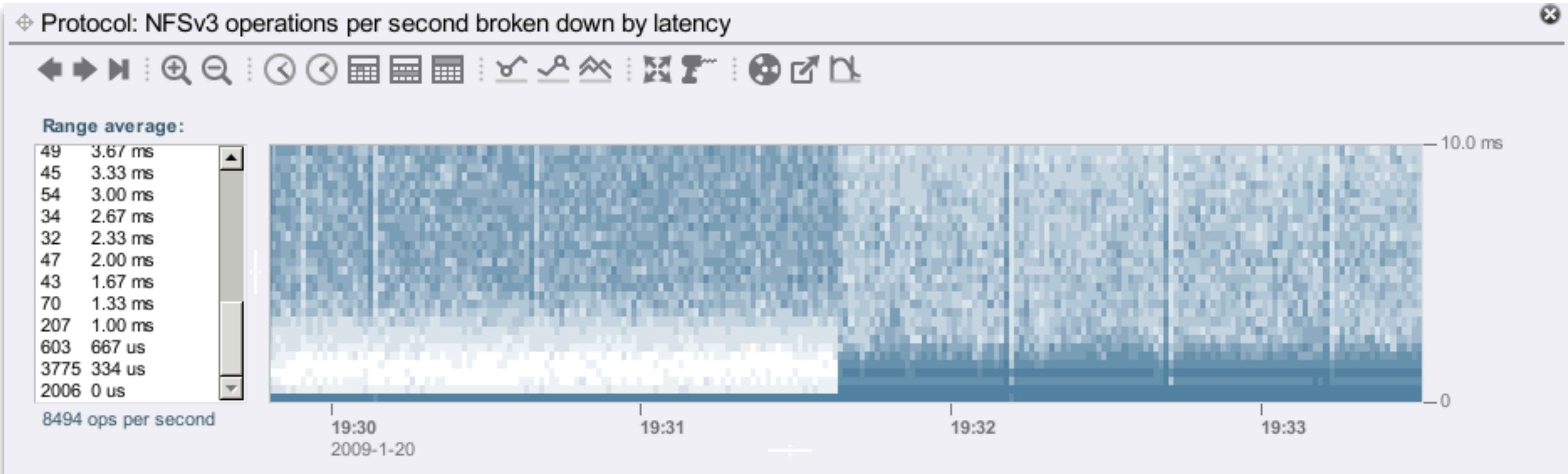



- x-axis: time
- y-axis: offset
- z-axis (color scale): I/O count for that time/offset range

- Identified random vs. sequential very well
- Similar heat maps have been used before by defrag tools

# Heat Map: Latency Distribution

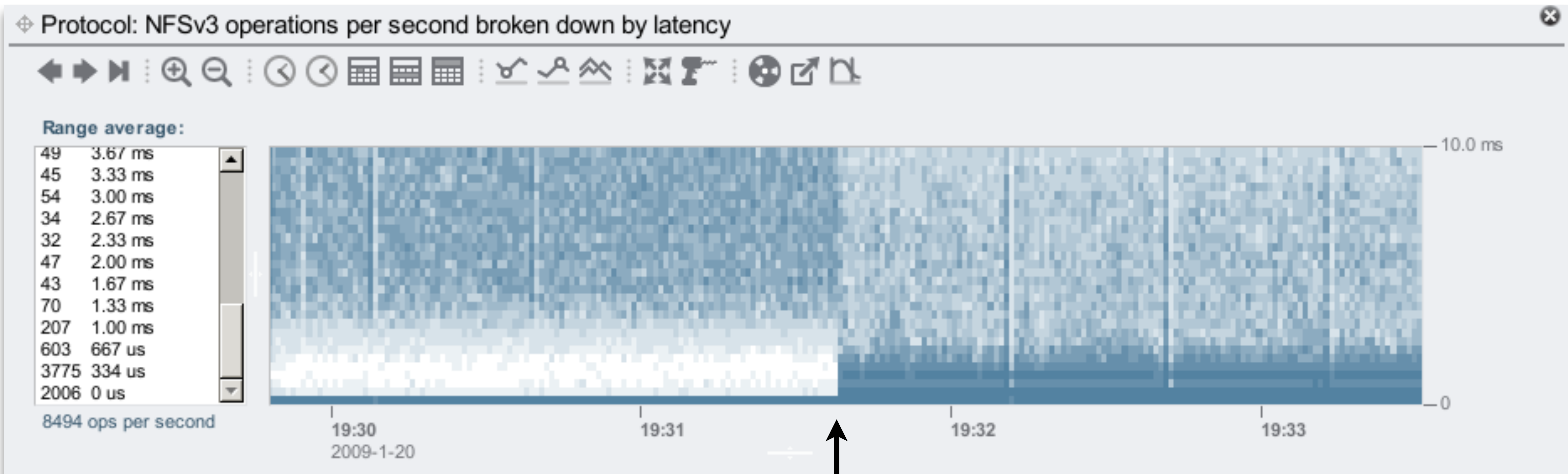
- For example:




- x-axis: time
- y-axis: latency
- z-axis (color saturation): I/O count for that time/latency range

# Heat Map: Latency Distribution

- ... in fact, this is a great example:




- reads served from:

DRAM
disk

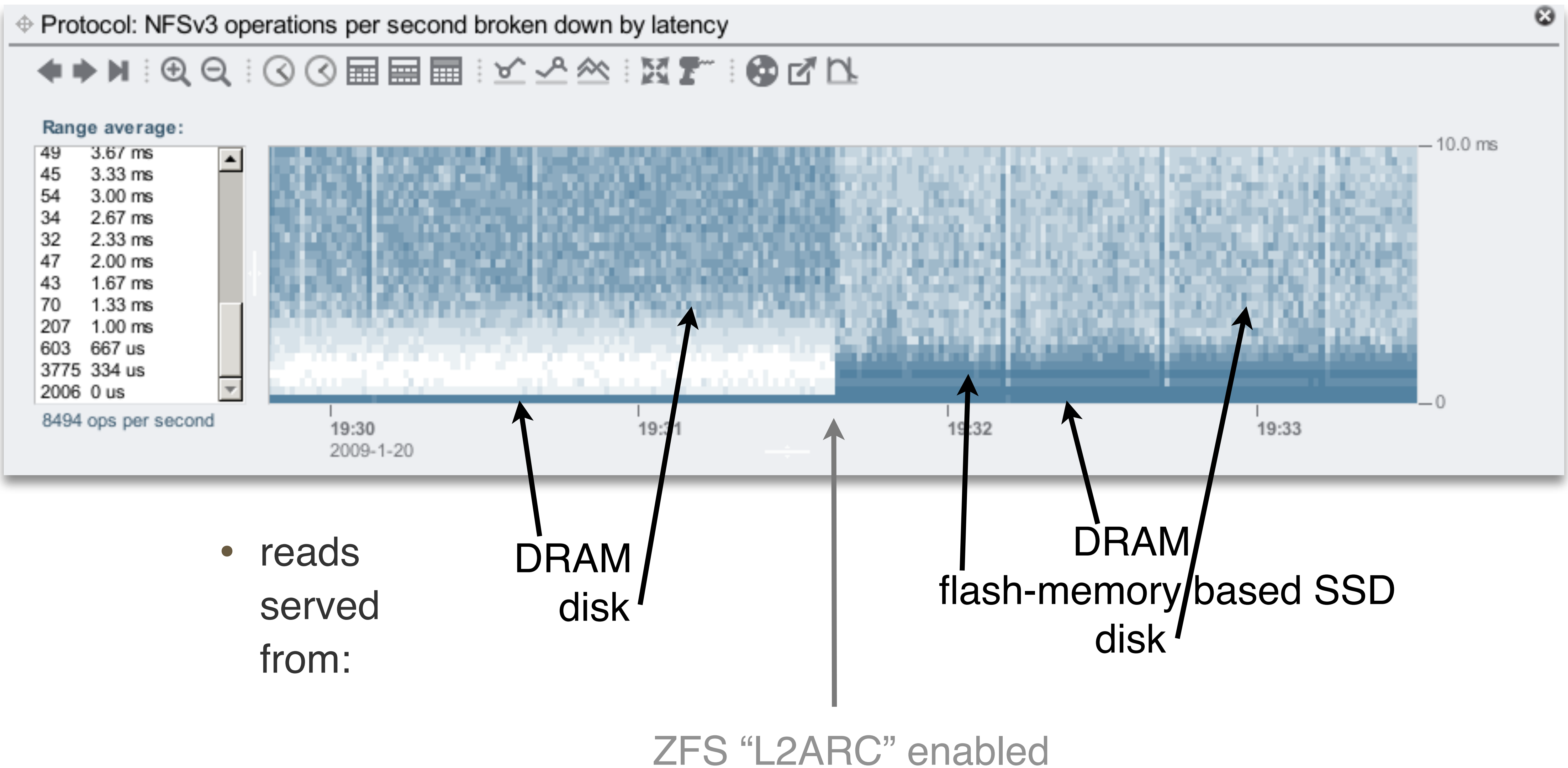
DRAM
flash-memory based SSD
disk

ZFS “L2ARC” enabled

# Heat Map: Latency Distribution

Joyent

- ... in fact, this is a great example:



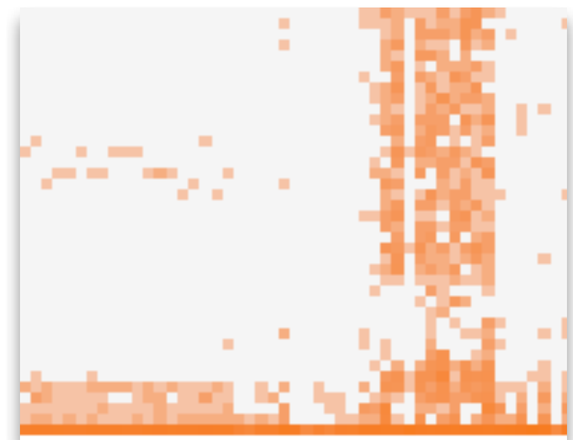

- reads served from:

# Latency Heat Map

- A color shaded matrix of pixels
- Each pixel is a time and latency range
- Color shade picked based on number of I/O in that range
- Adjusting saturation seems to work better than color hue. Eg:
    - darker == more I/O
    - lighter == less I/O

# Pixel Size

- Large pixels (and corresponding time/latency ranges)
  - increases likelyhood that adjacent pixels include I/O, have color, and combine to form patterns
  - allows color to be more easily seen



- Smaller pixels (and time/latency ranges)
  - can make heat map look like a scatter plot
  - of the same color - if ranges are so small only one I/O is typically included

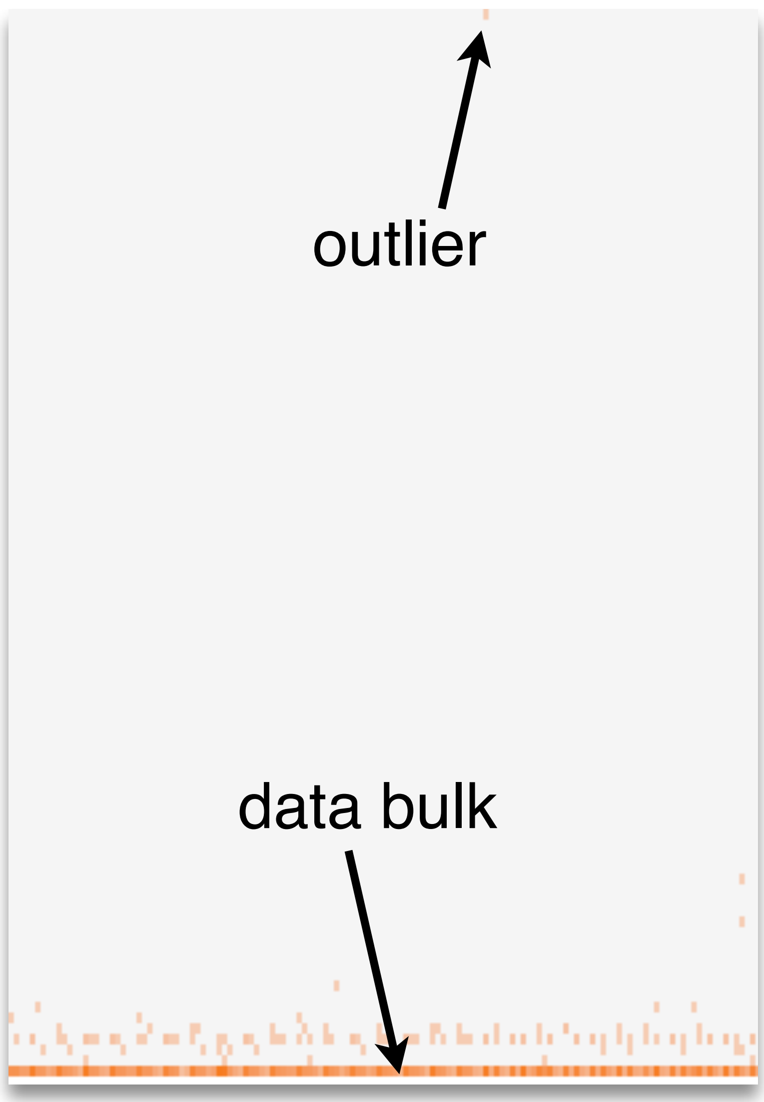
# Color Palette

- Linear scale can make subtle details (outliers) difficult to see
  - observing latency outliers is usually of high importance
  - outliers are usually < 1% of the I/O
  - assigning < 1% of the color scale to them will washout patterns
- False color palette can be used to emphasize these details
  - although color comparisons become more confusing - non-linear

# Outliers

Joyent

- Heat maps show these very well
- However, latency outliers can compress the bulk of the heat map data
  - eg, 1 second outlier while most I/O is < 10 ms
- Users should have some control to be able to zoom/truncate details
  - both x and y axis



# Data Storage

- Since heat-maps are three dimensions, storing this data can become costly (volume)
- Most of the data points are zero
  - and you can prevent storing zero's by only storing populated elements: associative array
- You can reduce to a sufficiently high resolution, and resample lower as needed
- You can also be aggressive at reducing resolution at higher latencies
  - 10 us granularity not as interesting for I/O > 1 second
  - non-linear resolution

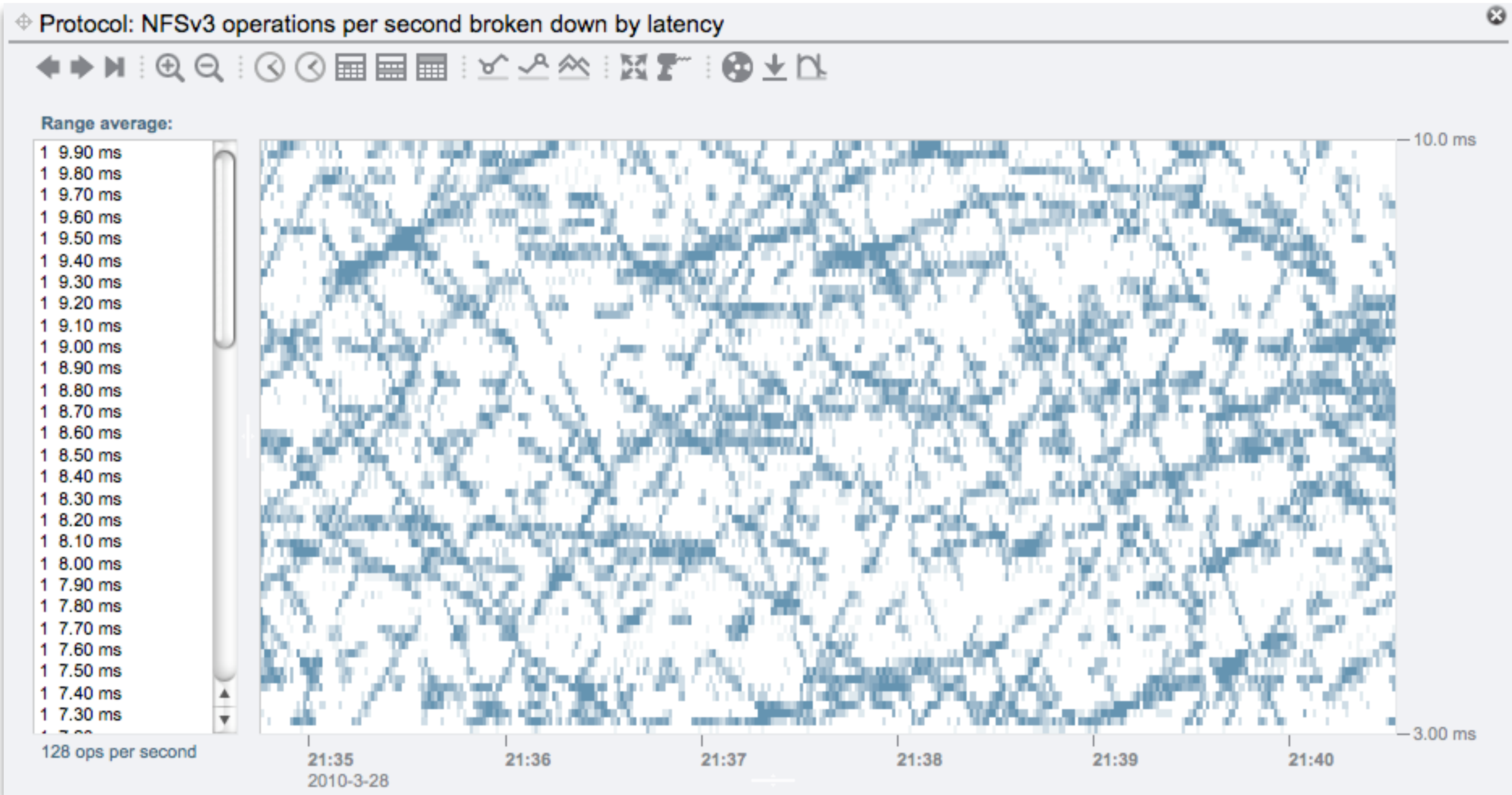
# Other Interesting Latency Heat Maps

- The “Icy Lake”
- The “Rainbow Pterodactyl”
- Latency Levels

# The “Icy Lake” Workload

- About as simple as it gets:
  - Single client, single thread, sequential synchronous 8 Kbyte writes to an NFS share
  - NFS server: 22 x 7,200 RPM disks, striped pool
- The resulting latency heat map was unexpected

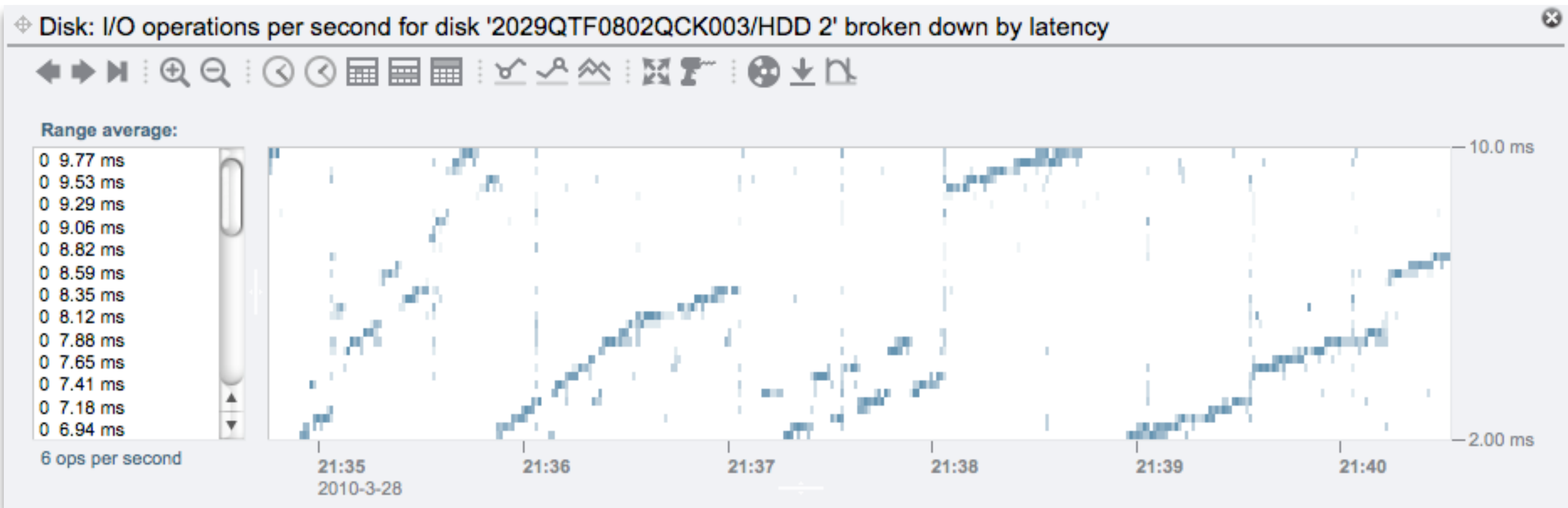
# The “Icy Lake”



Protocol: NFSv3 operations per second broken down by latency
Range average:
1 9.90 ms
1 9.80 ms
1 9.70 ms
1 9.60 ms
1 9.50 ms
1 9.40 ms
1 9.30 ms
1 9.20 ms
1 9.10 ms
1 9.00 ms
1 8.90 ms
1 8.80 ms
1 8.70 ms
1 8.60 ms
1 8.50 ms
1 8.40 ms
1 8.30 ms
1 8.20 ms
1 8.10 ms
1 8.00 ms
1 7.90 ms
1 7.80 ms
1 7.70 ms
1 7.60 ms
1 7.50 ms
1 7.40 ms
1 7.30 ms
128 ops per second
10.0 ms
3.00 ms
21:35
2010-3-28
21:36
21:37
21:38
21:39
21:40

# "Icy Lake" Analysis: Observation

- Examining single disk latency:



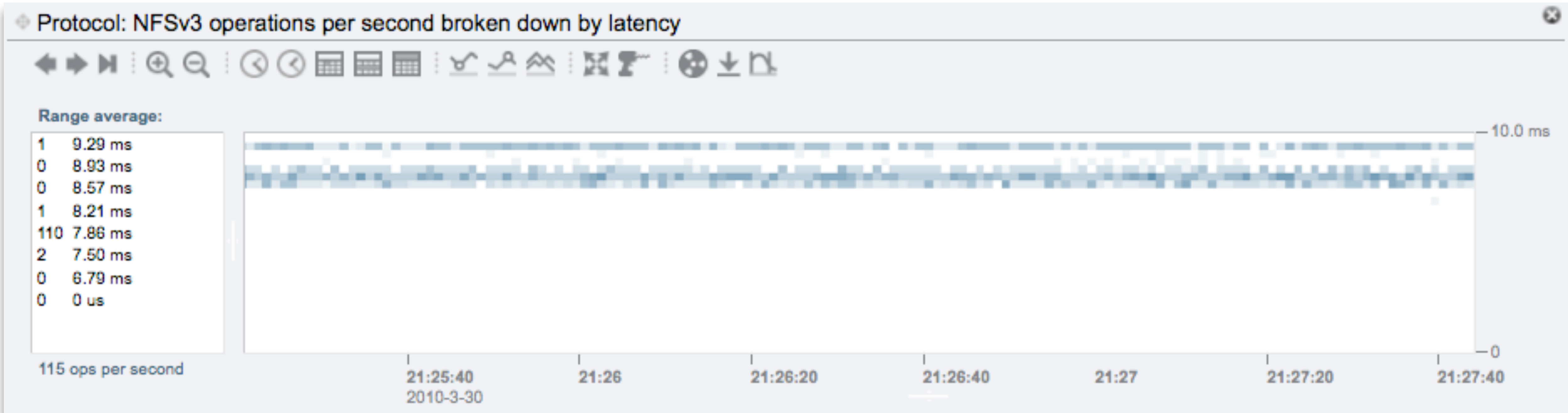

- Pattern match with NFS latency: similar lines
    - each disk contributing some lines to the overall pattern

# Pattern Match?

- We just associated NFS latency with disk device latency, using our eyeballs
  - see the titles on the previous heat maps
- You can programmatically do this (DTrace), but that can get difficult to associate context across software stack layers (but not impossible!)
- Heat Maps allow this part of the problem to be offloaded to your brain
  - and we are very good at pattern matching

# “Icy Lake” Analysis: Experimentation

- Same workload, single disk pool:

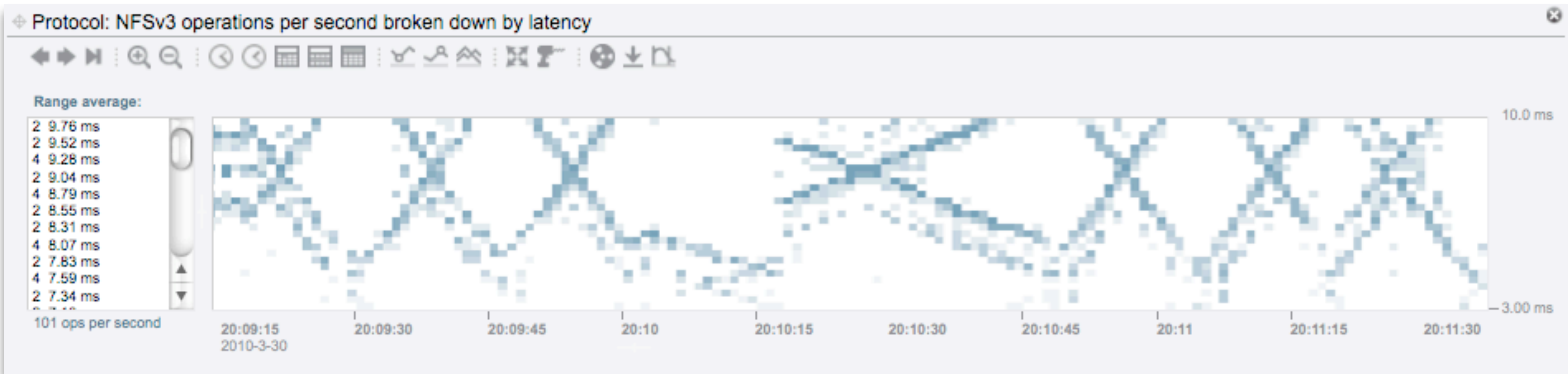



- No diagonal lines
  - but more questions - see the line (false color palette enhanced) at 9.29 ms? this is < 1% of the I/O. (I’m told, and I believe, that this is due to adjacent track seek latency.)

# “Icy Lake” Analysis: Experimentation

Joyent

- Same workload, two disk striped pool:

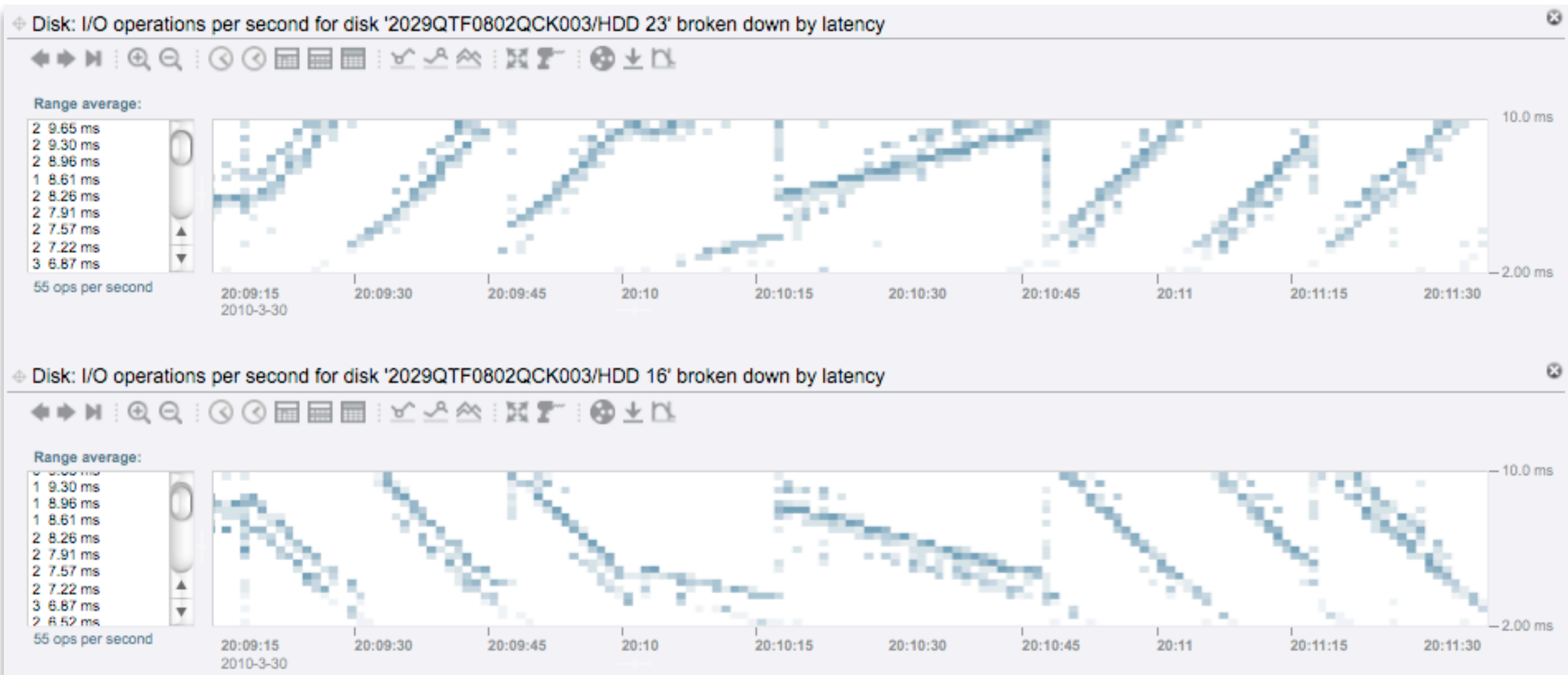



- Ah-hah!  Diagonal lines.
  - ... but still more questions: why does the angle sometimes change? why do some lines slope upwards and some down?

# “Icy Lake” Analysis: Experimentation

Joyent

- ... each disk from that pool:

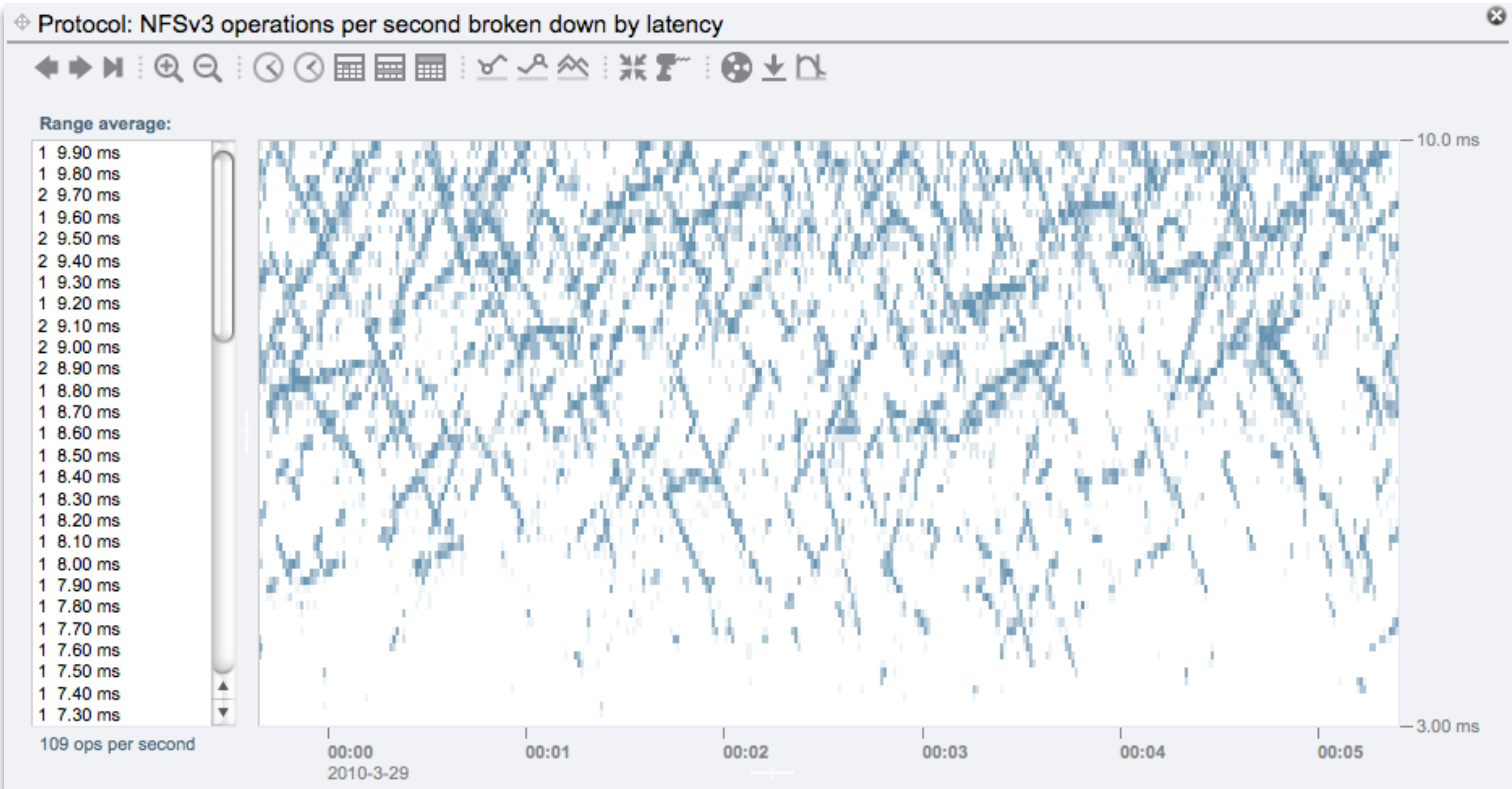


# “Icy Lake” Analysis: Questions

- Remaining Questions:
  - Why does the slope sometimes change?
  - What exactly seeds the slope in the first place?

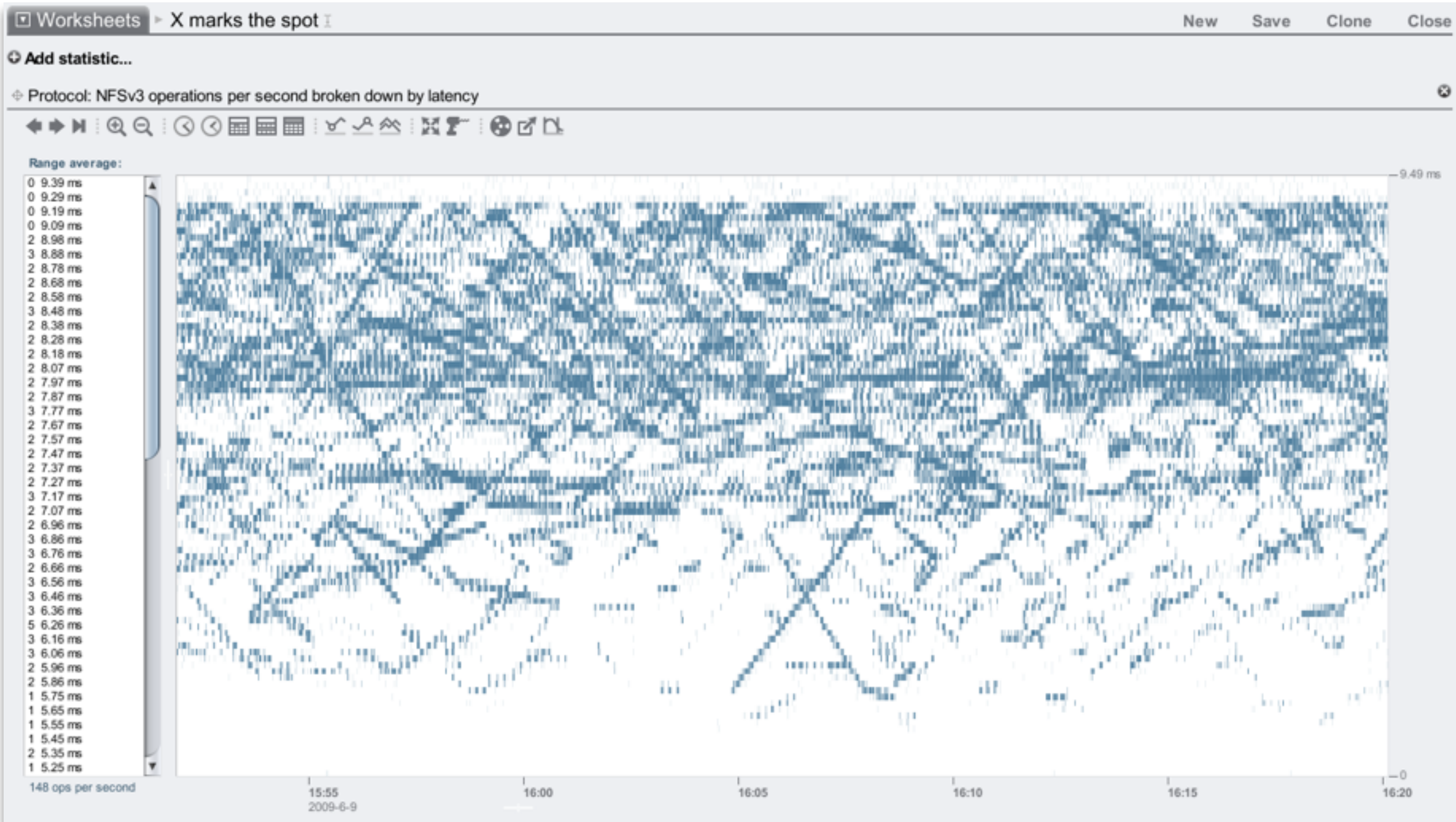
# "Icy Lake" Analysis: Mirroring

- Trying mirroring the pool disks instead of striping:



# Another Example: "X marks the spot"

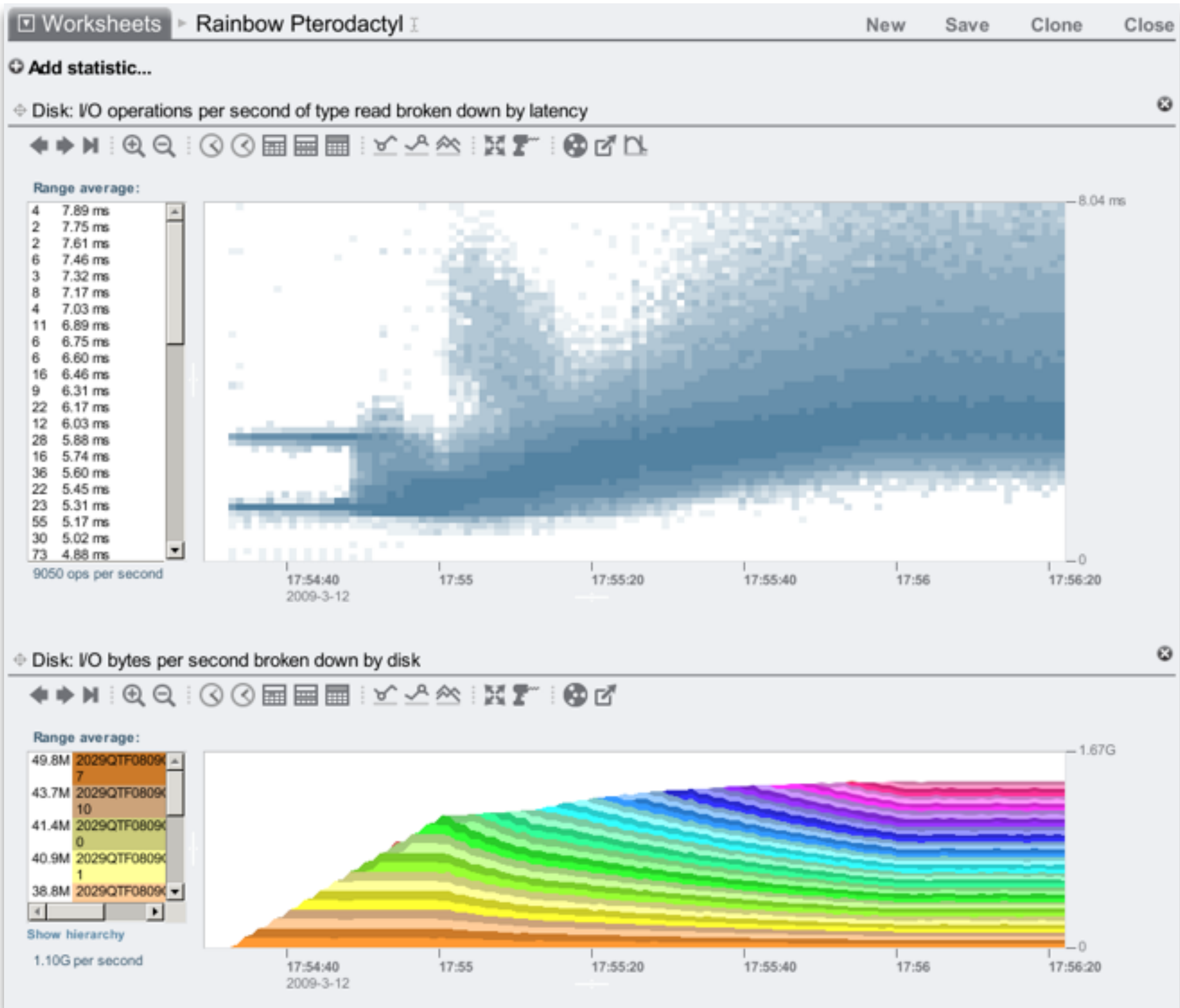
Joyent



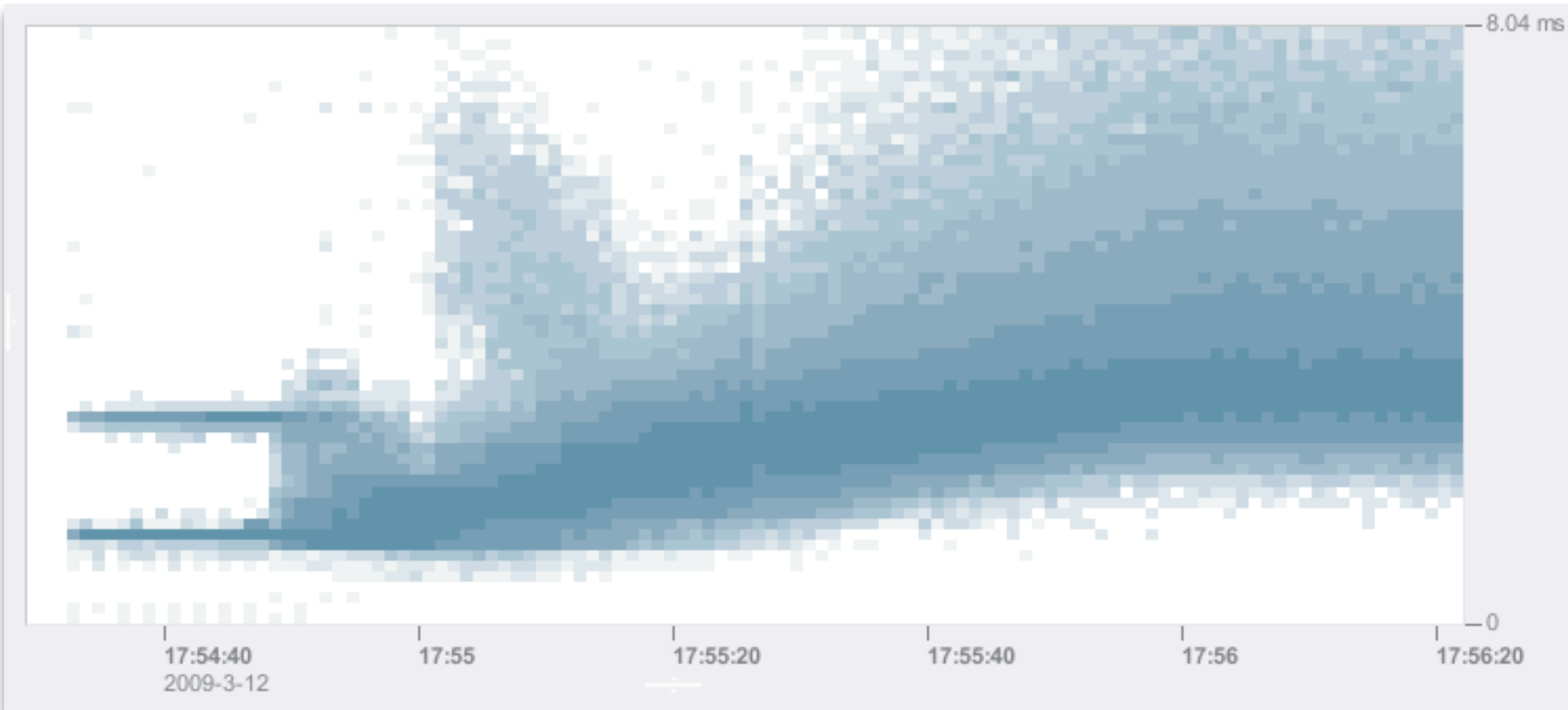
## The "Rainbow Pterodactyl" Workload

- 48 x 7,200 RPM disks, 2 disk enclosures
- Sequential 128 Kbyte reads to each disk (raw device), adding disks every 2 seconds
- Goal: Performance analysis of system architecture
    - identifying I/O throughput limits by driving I/O subsystem to saturation, one disk at a time (finds knee points)

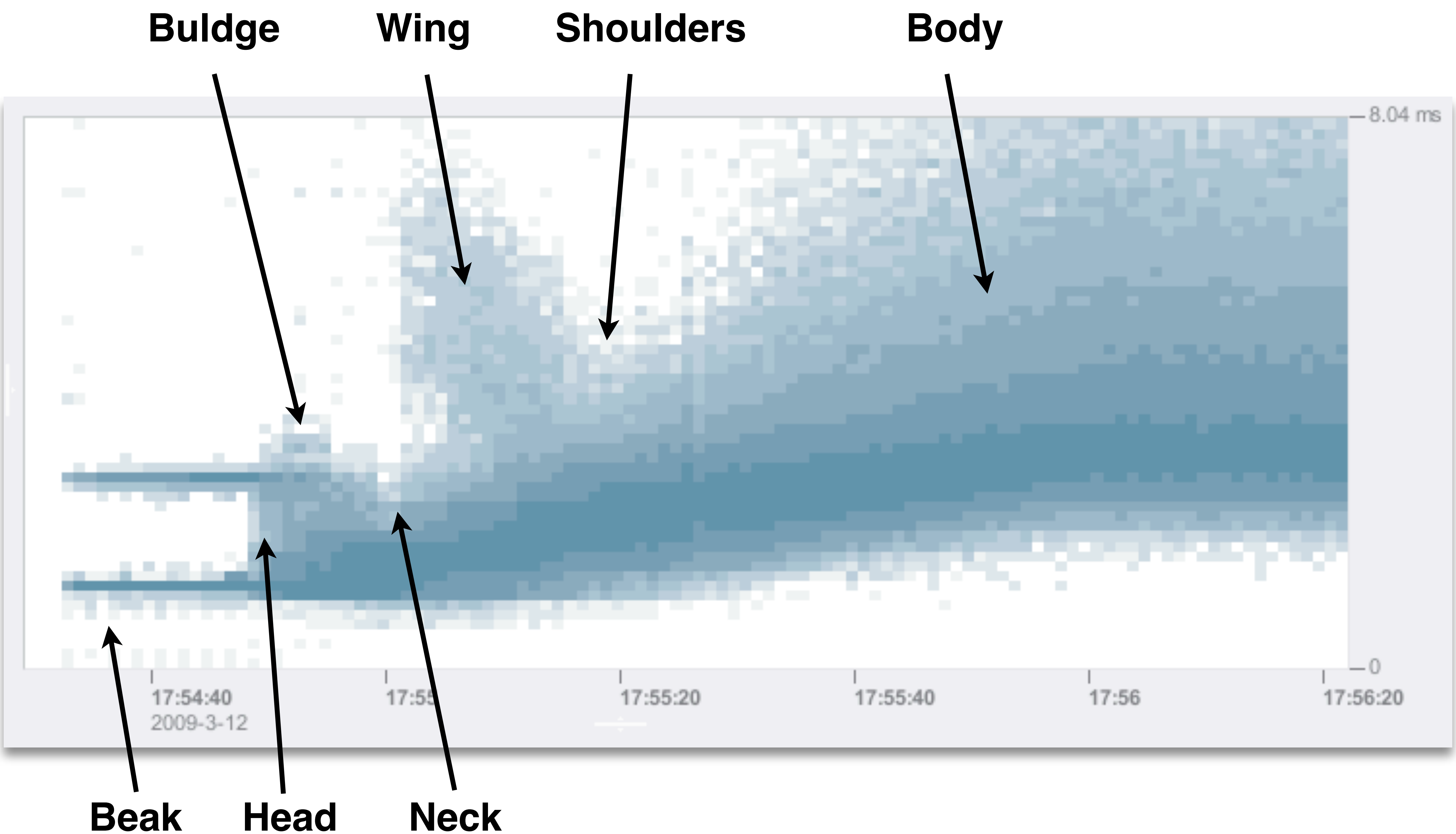
# The "Rainbow Pterodactyl"



# The "Rainbow Pterodactyl"



# The "Rainbow Pterodactyl"



# The “Rainbow Pterodactyl”: Analysis

Joyent

- Hasn’t been understood in detail
  - Would never be understood (or even known) without heat maps
- It is repeatable

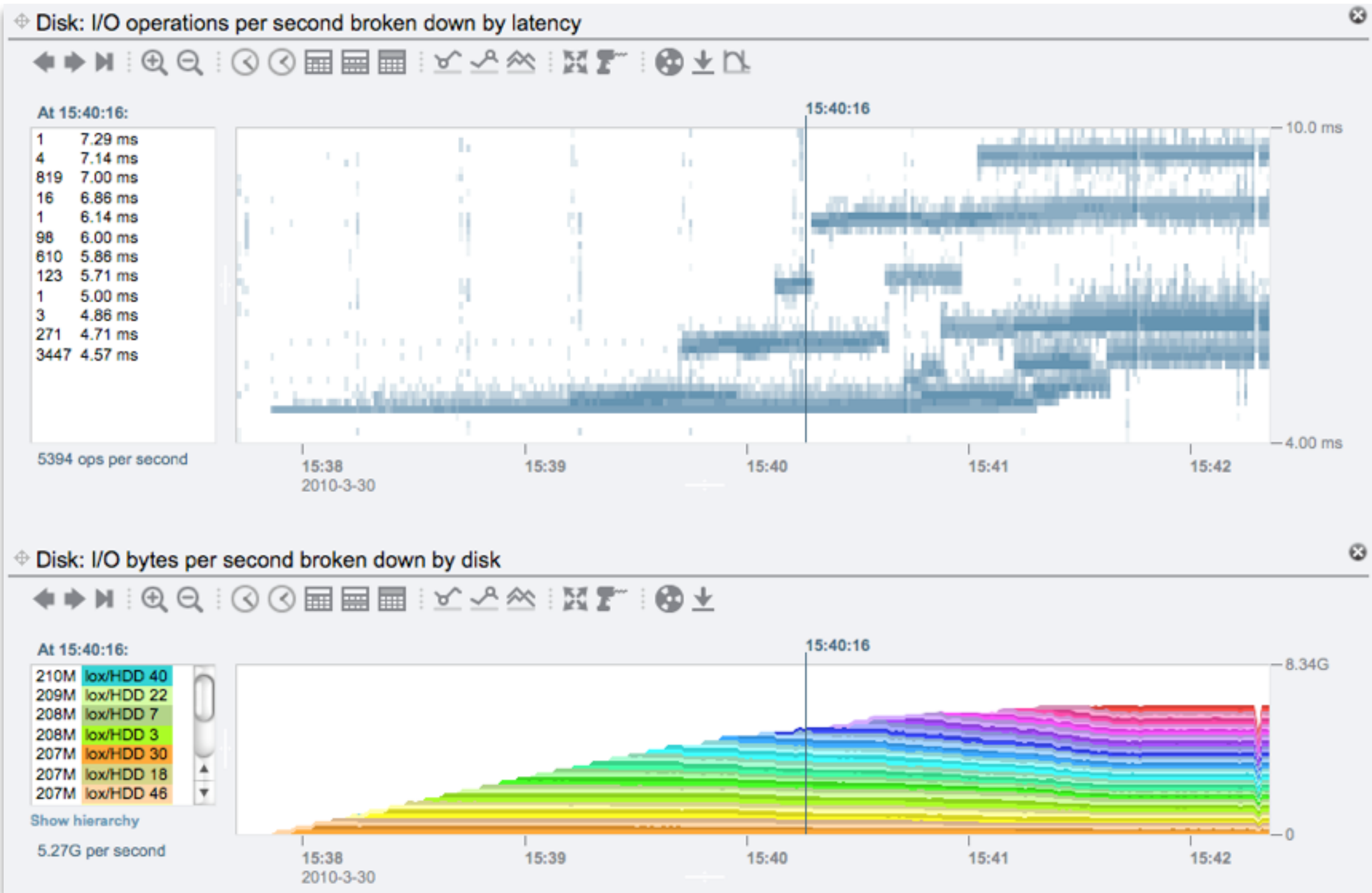
# The “Rainbow Pterodactyl”: Theories

- “Beak”: disk cache hit vs disk cache miss -> bimodal
- “Head”: 9th disk, contention on the 2 x4 SAS ports
- “Buldge”: ?
- “Neck”: ?
- “Wing”: contention?
- “Shoulders”: ?
- “Body”: PCI-gen1 bus contention

# Latency Levels Workload

- Same as “Rainbow Pterodactyl”, stepping disks
- Instead of sequential reads, this is repeated 128 Kbyte reads (read -> seek 0 -> read -> ...), to deliberately hit from the disk DRAM cache to improve test throughput

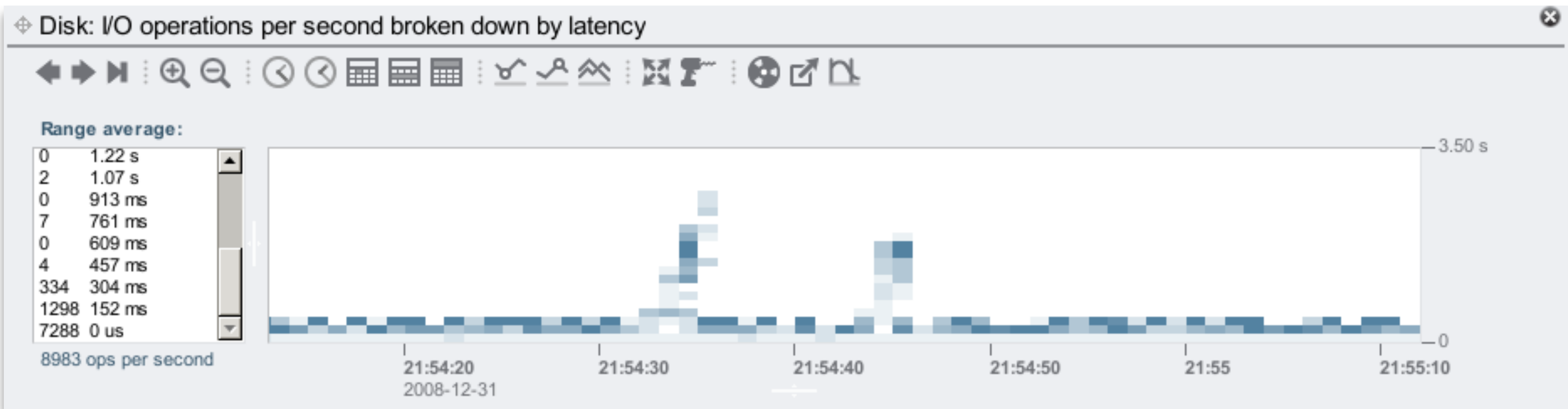
# Latency Levels



# Latency Levels Theories

- ???

# Bonus Latency Heat Map




- This time we do know the source of the latency...

# 硬碟也會鬧情緒

Joyent

癮科技小劇場：硬碟也會鬧情緒

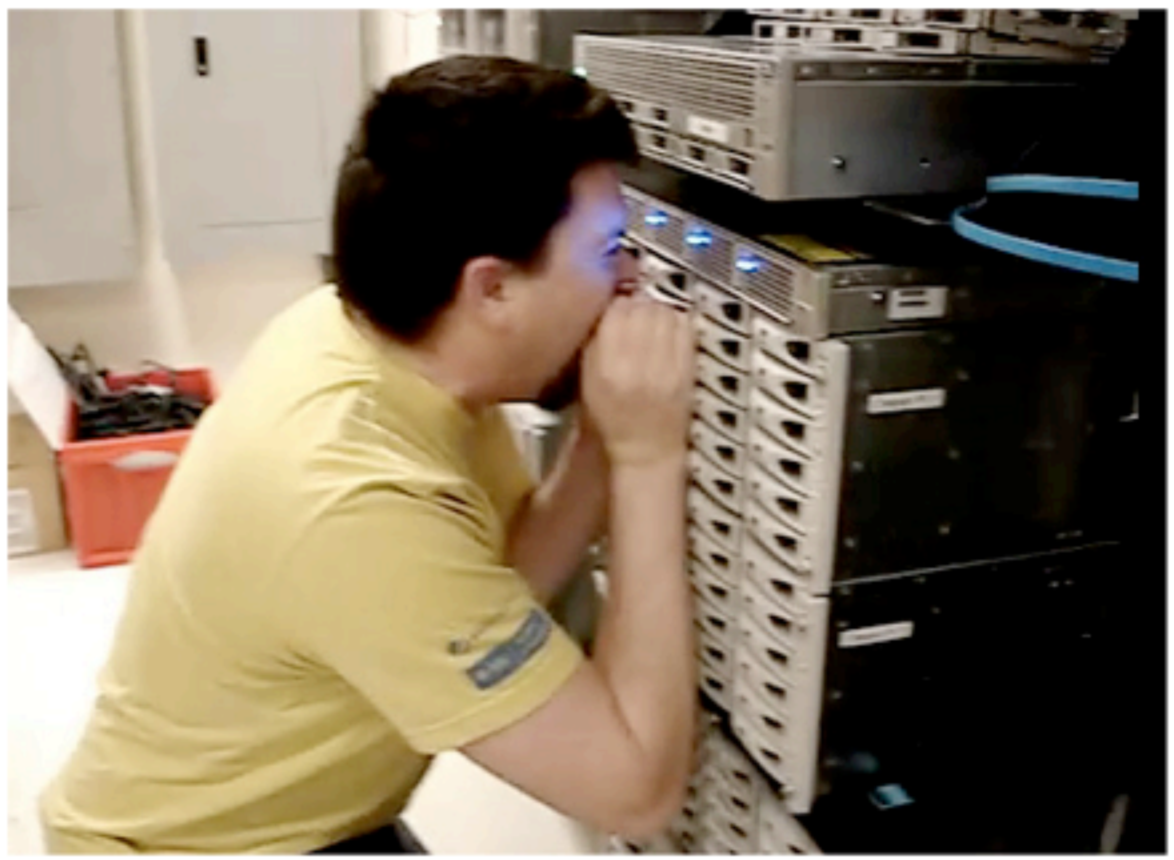
http://chinese.engadget.com/2009/01/05/video-shouting-at-disk-drive-causes-high-latency-low-r

別向硬碟亂叫，會影響它們的心情啊... | 癮科技小劇場：硬碟也會鬧情緒

## 癮科技小劇場：硬碟也會鬧情緒

由 Flow Yu 於 1 year 之前發表

**文章分類:** 儲存裝置



這可不是開玩笑來著，而是經過實驗証明的唷！來自 Sun Fishworks 團隊的 Brendan Gregg，或許是在機房裡待久了，於是突發異想跑去對機房中的磁碟陣列大吼大叫一陣，結果就這麼發現了一個真理，那就是：硬碟就跟員工一樣，吼叫並不會讓他們的效率變高，反而還會讓它們心生不爽而降低士氣，私底下就開始搞罷工。

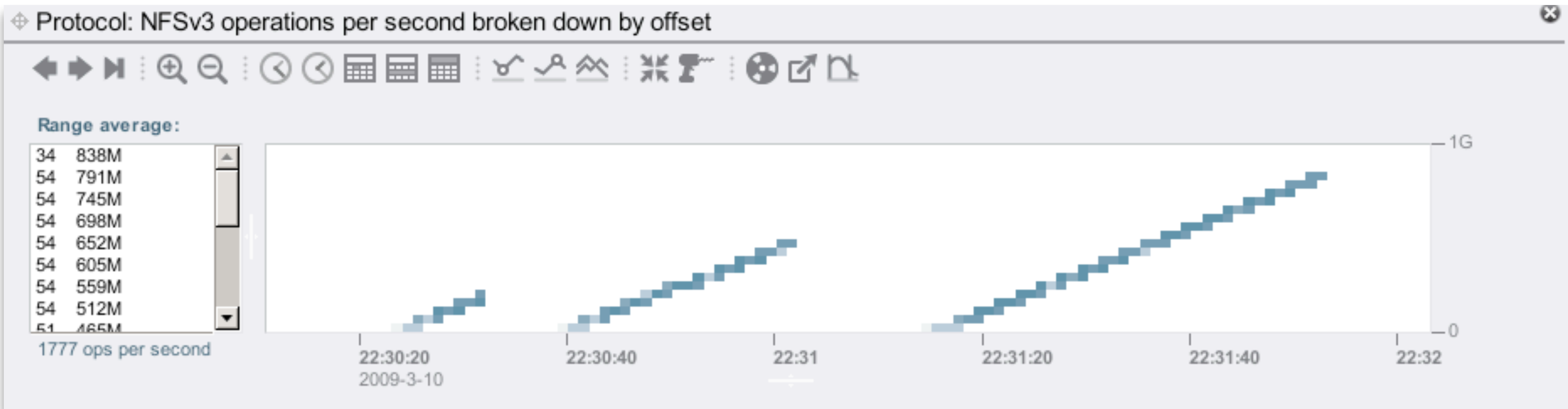
# Latency Heat Maps: Summary

- Shows latency distribution over time
- Shows outliers (maximums)
- Indirectly shows average
- Shows patterns
    - allows correlation with other software stack layers

# Similar Heat Map Uses

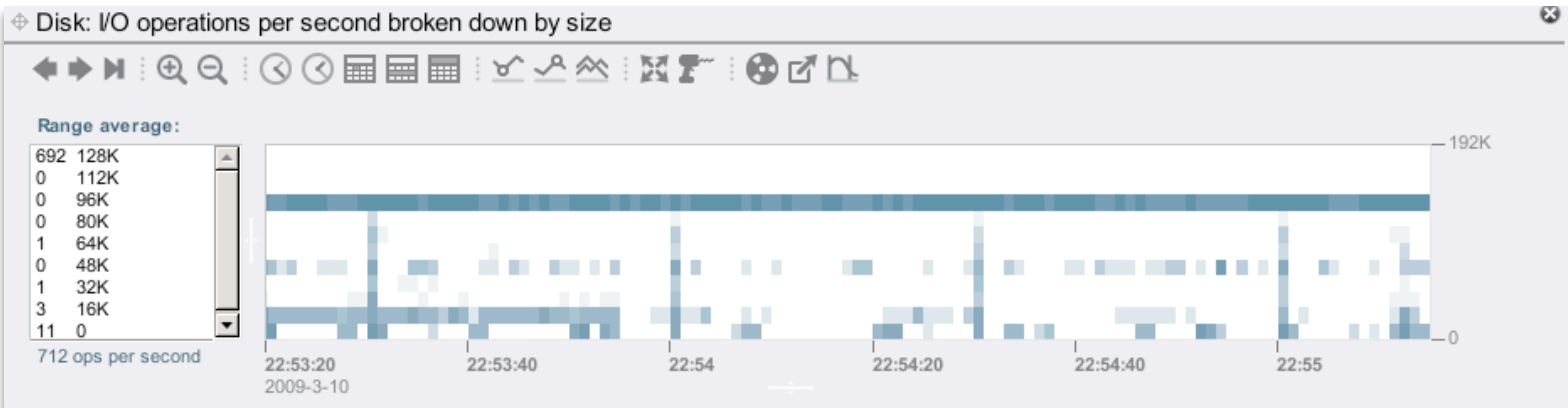
- These all have a dynamic y-axis scale:
  - I/O size
  - I/O offset
- These aren’t a primary measure of performance (like latency); they provide secondary information to understand the workload

# Heat Map: I/O Offset




- y-axis: I/O offset (in this case, NFSv3 file location)

# Heat Map: I/O Size

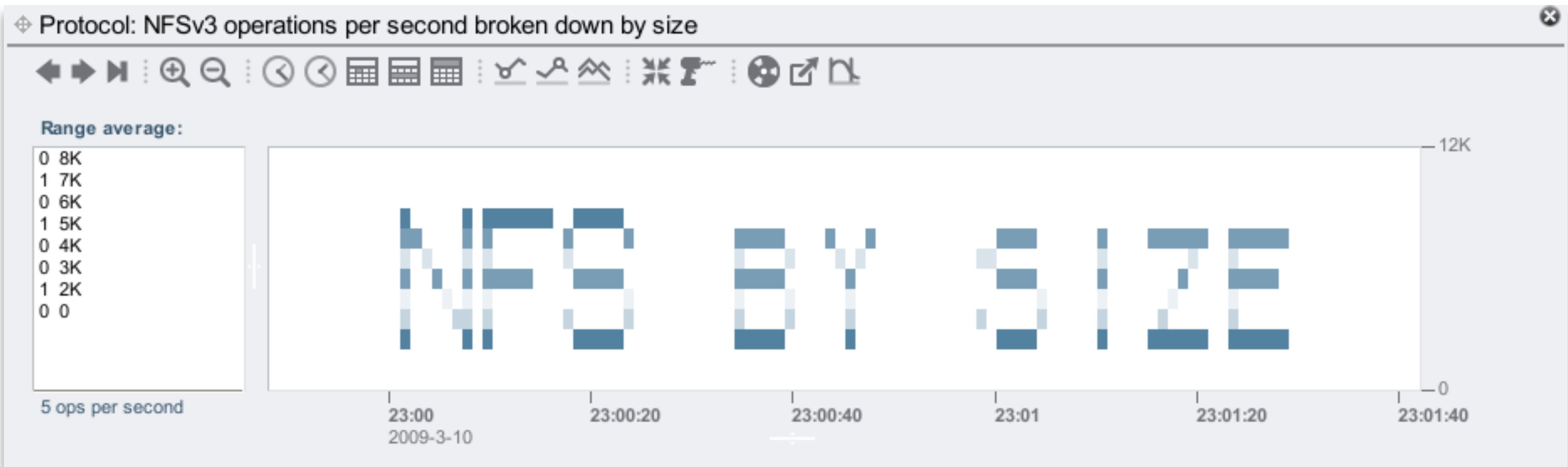



- y-axis: I/O size (bytes)

# Heat Map Abuse

- What can we ‘paint’ by adjusting the workload?

# I/O Size

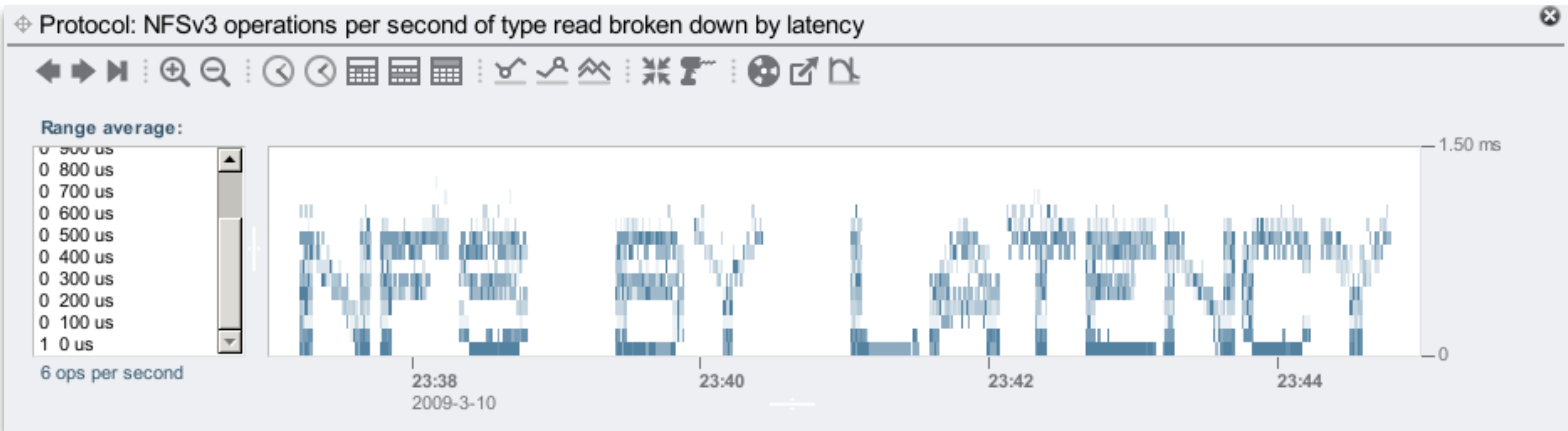



- How was this done?

# I/O Offset


- How was this done?

# I/O Latency




- How was this done?

# Visualizations

Current Examples

Utilization

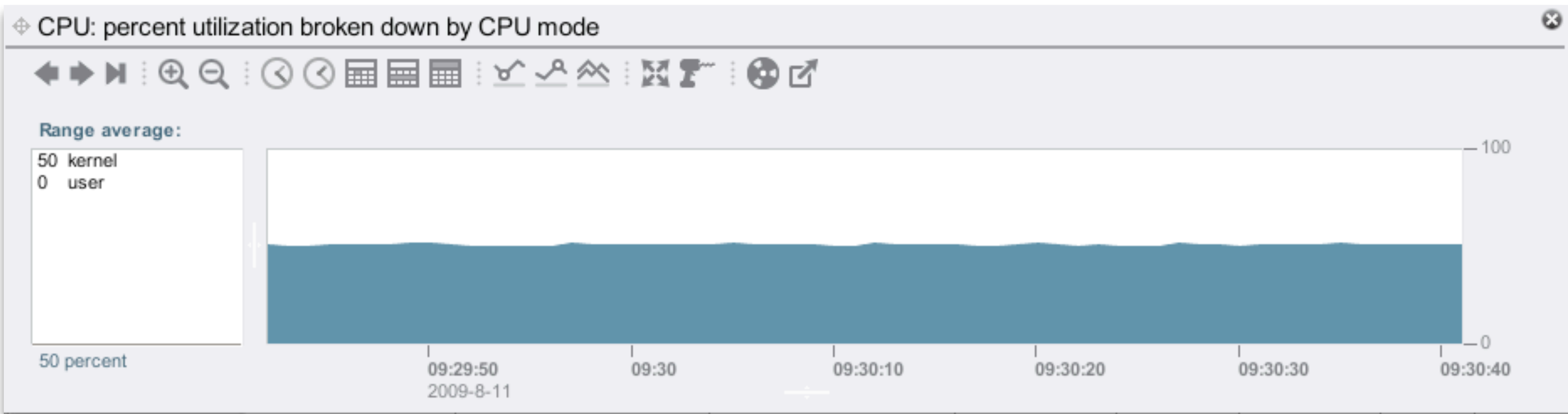
# CPU Utilization

- Commonly used indicator of CPU performance
- eg, vmstat(1M)

```
$ vmstat 1 5
 kthr      memory            page            disk          faults      cpu
 r b w   swap  free  re  mf pi po fr de sr s0 s1 s2 s3   in   sy   cs us sy id
 0 0 0 95125264 28022732 301 1742 1 17 17 0 0 -0 -0 -0 6 5008 21927 3886 4 1 94
 0 0 0 91512024 25075924 6 55 0 0  0  0  0  0  0  0  0 4665 18228 4299 10 1 89
 0 0 0 91511864 25075796 9 24 0 0  0  0  0  0  0  0  0 3504 12757 3158 8  0 92
 0 0 0 91511228 25075164 3 163 0 0 0  0  0  0  0  0  0 4104 15375 3611 9  5 86
 0 0 0 91510824 25074940 5 66 0 0  0  0  0  0  0  0  0 4607 19492 4394 10 1 89
```

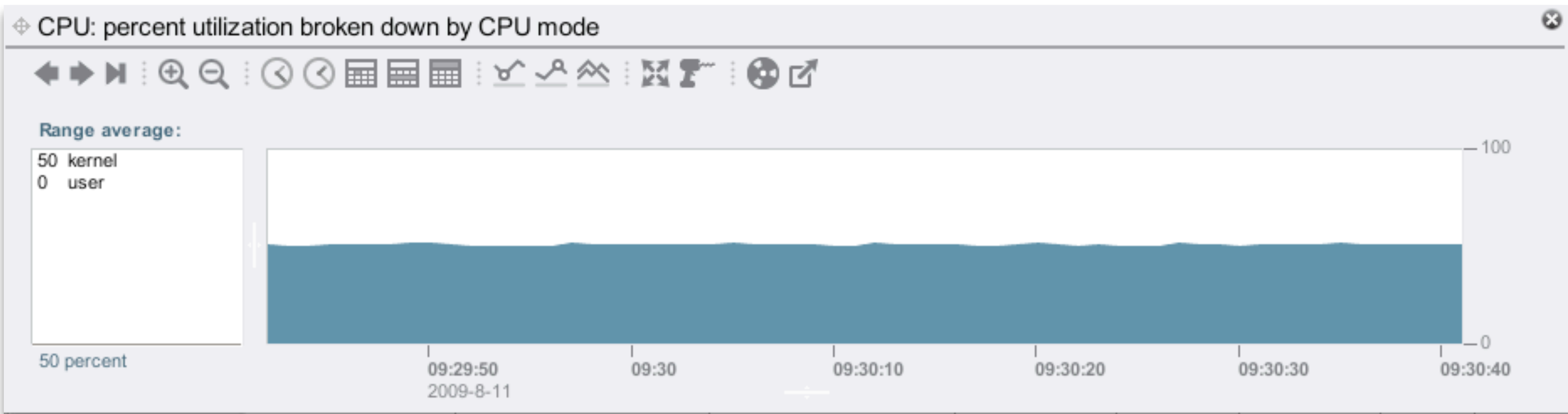
# CPU Utilization: Line Graph

- Easy to plot:



# CPU Utilization: Line Graph

- Easy to plot:




- Average across all CPUs:
  - Identifies how utilized all CPUs are, indicating remaining headroom - provided sufficient threads to use CPUs

# CPU Utilization by CPU

- mpstat(1M) can show utilization by-CPU:

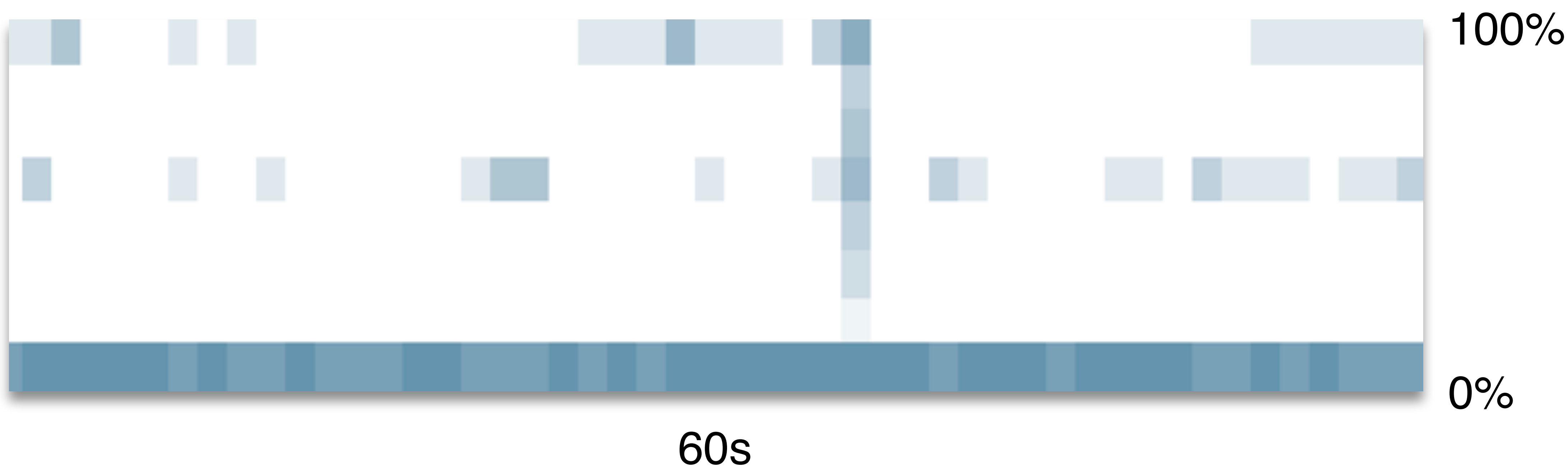
```
$ mpstat 1
[...]
CPU minf mjf xcal  intr ithr  csw icsw migr smtx  srw syscl  usr sys  wt idl
  0    0   0    2   313  105  315    0   24    4    0  1331    5   1   0  94
  1    0   0    0    65   28  190    0   12    4    0   576    1   1   0  98
  2    0   0    0    64   20  152    0   12    1    0   438    0   1   0  99
  3    0   0    0   127   74  274    1   21    3    0   537    1   1   0  98
  4    0   0    0    32    5  229    0    9    2    0   902    1   1   0  98
  5    0   0    0    46   19  138    0    7    3    0   521    1   0   0  99
  6    2   0    0   109   32  296    0    8    2    0  1266    4   0   0  96
  7    0   0    0    30    8    0    9    0    1    0     0  100   0   0   0
  8    0   0    0   169   68  311    0   22    2    0   847    2   1   0  97
  9    0   0   30   111   54  274    0   16    4    0   868    2   0   0  98
 10    0   0    0    69   29  445    0   13    7    0  2559    7   1   0  92
 11    0   0    0    78   36  303    0    7    8    0  1041    2   0   0  98
 12    0   0    0    74   34  312    0   10    1    0  1250    7   1   0  92
 13   38   0   15   456  285  336    2   10    1    0  1408    5   2   0  93
 14    0   0    0  2620 2497  209    0   10   38    0   259    1   3   0  96
 15    0   0    0    20    8   10    0    4    2    0     2    0   0   0 100
```

- can identify a single hot CPU (thread)
    - and un-balanced configurations

## CPU Resource Monitoring

- Monitor overall utilization for capacity planning
- Also valuable to monitor individual CPUs
  - can identify un-balanced configurations
  - such as a single hot CPU (thread)
- The virtual CPUs on a single host can now reach the 100s
  - its own dimension
  - how can we display this 3rd dimension?

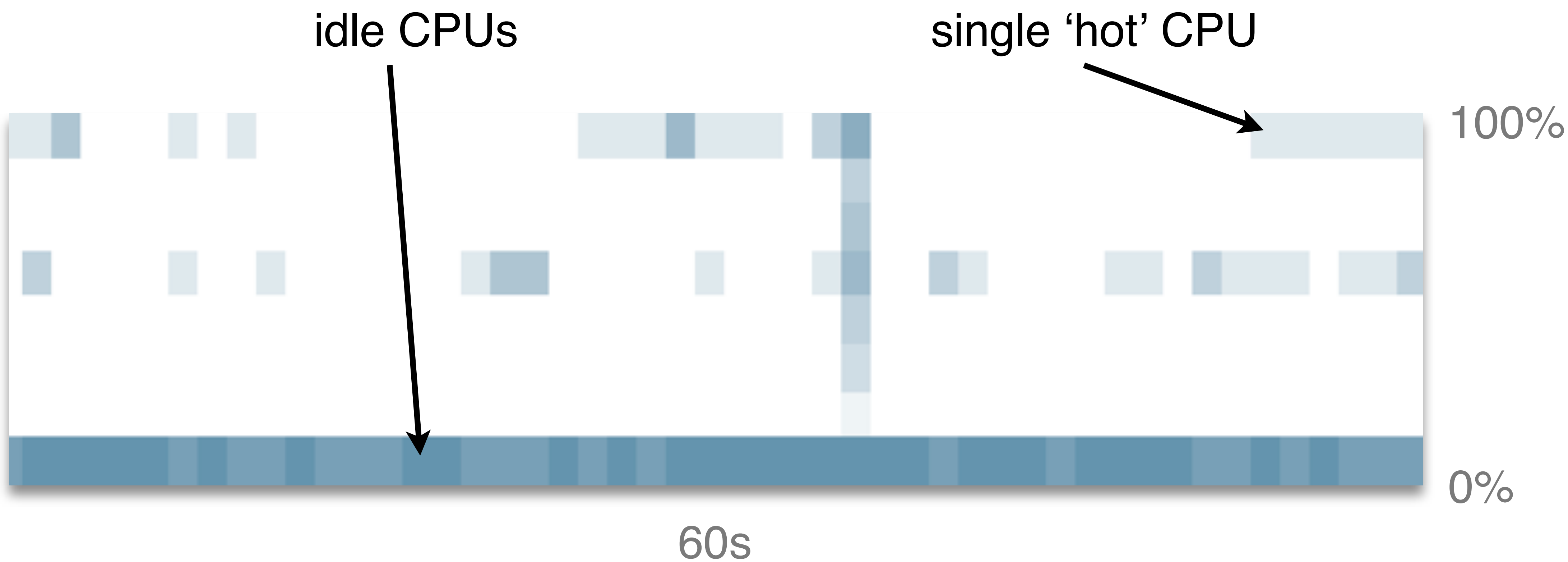
# Heat Map: CPU Utilization




- x-axis: time
- y-axis: percent utilization
- z-axis (color saturation): # of CPUs in that time/utilization range

# Heat Map: CPU Utilization

Joyent




- Single 'hot' CPUs are a common problem due to application scaleability issues (single threaded)
- This makes identification easy, without reading pages of mpstat (1M) output

# Heat Map: Disk Utilization

- Ditto for disks
- Disk Utilization heat map can identify:
  - overall utilization
  - unbalanced configurations
  - single hot disks (versus all disks busy)
- Ideally, the disk utilization heat map is tight (y-axis) and below 70%, indicating a well balanced config with headroom
  - which can’t be visualized with line graphs

# Back to Line Graphs...

- Are typically used to visualize performance, be it IOPS or utilization
- Show patterns over time more clearly than text (higher resolution)
- But graphical environments can do much more
    - As shown by the heat maps (to start with); which convey details line graphs cannot
- Ask: what “value add” does the GUI bring to the data?

# Resource Utilization Heat Map Summary

- Can exist for any resource with multiple components:
  - CPUs
  - Disks
  - Network interfaces
  - I/O busses
  - ...
- Quickly identifies single hot component versus all components
- Best suited for physical hardware resources
  - difficult to express ‘utilization’ for a software resource

# Visualizations

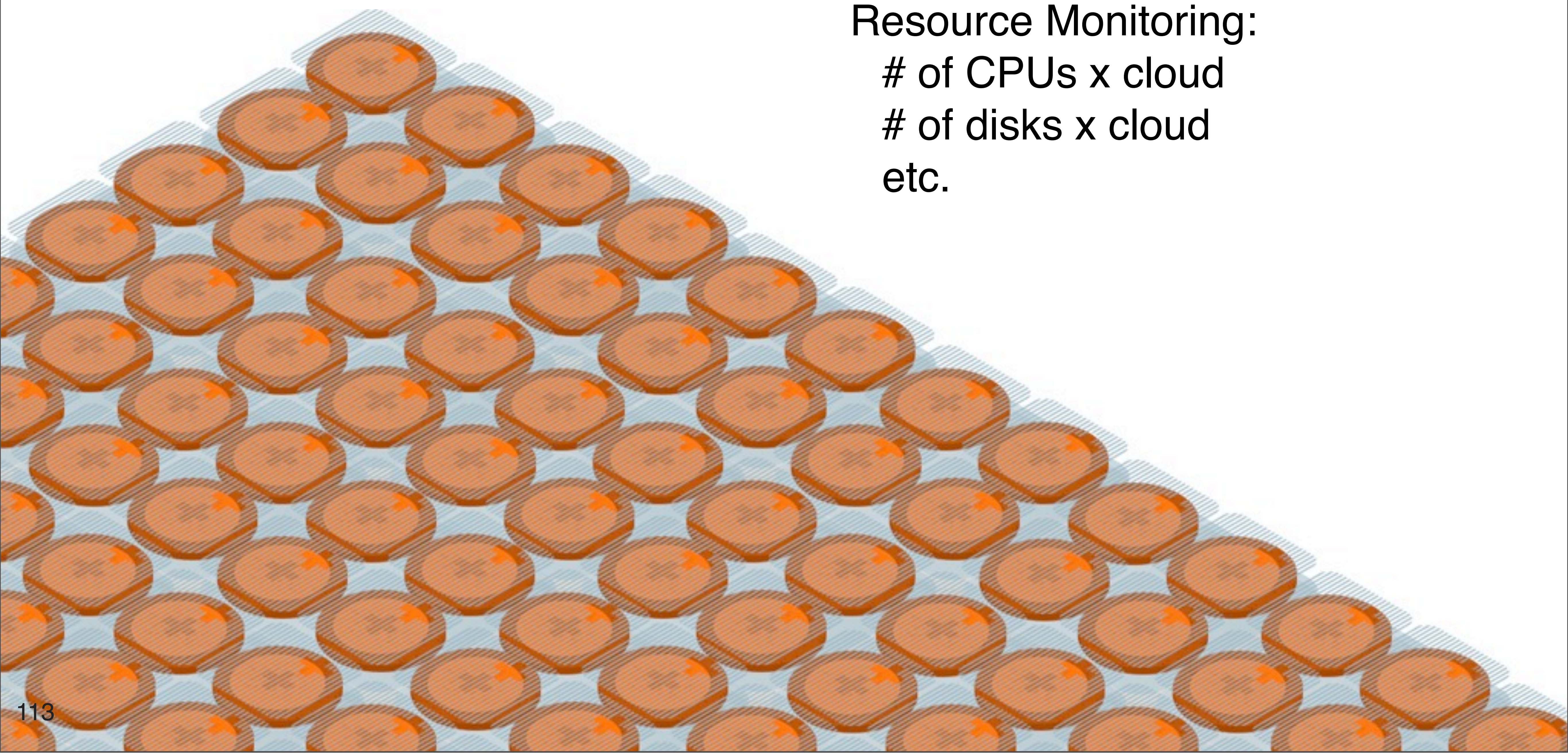
Future Opportunities

# Cloud Computing

So far analysis has been for a single server

What about the cloud?

# From one to thousands of servers



Workload Analysis:
  latency I/O x cloud

Resource Monitoring:
  # of CPUs x cloud
  # of disks x cloud
  etc.

# Heat Maps for the Cloud



- Heat Maps are promising for cloud computing observability:
  - additional dimension accommodates the scale of the cloud
- Find outliers regardless of node
  - cloud-wide latency heat map just has more I/O
- Examine how applications are load balanced across nodes
  - similar to CPU and disk utilization heat maps
- mpstat and iostat’s output are already getting too long
  - multiply by 1000x for the number of possible hosts in a large cloud application

# Proposed Visualizations

- Include:
  - Latency heat map across entire cloud
  - Latency heat maps for cloud application components
  - CPU utilization by cloud node
  - CPU utilization by CPU
  - Thread/process utilization across entire cloud
  - Network interface utilization by cloud node
  - Network interface utilization by port
  - lots, lots more

# Cloud Latency Heat Map

- Latency at different layers:
  - Apache
  - PHP/Ruby/...
  - MySQL
  - DNS
  - Disk I/O
  - CPU dispatcher queue latency
  - and pattern match to quickly identify and locate latency

# Latency Example: MySQL

Joyent

- Query latency (DTrace):

```
query time (ns)
           value  ------------- Distribution ------------- count
            1024 |                                         0
            2048 |                                         2
            4096 |@                                        99
            8192 |                                         20
           16384 |@                                        114
           32768 |@                                        105
           65536 |@                                        123
          131072 |@@@@@@@@@@@@@                            1726
          262144 |@@@@@@@@@@@                              1515
          524288 |@@@@                                     601
         1048576 |@@                                       282
         2097152 |@                                        114
         4194304 |                                         61
         8388608 |@@@@@                                    660
        16777216 |                                         67
        33554432 |                                         12
        67108864 |                                         7
       134217728 |                                         4
       268435456 |                                         5
       536870912 |                                         0
```

# Latency Example: MySQL

Joyent

- Query latency (DTrace):

```
query time (ns)
           value  ------------- Distribution ------------- count
            1024 |                                         0
            2048 |                                         2
            4096 |@                                        99
            8192 |                                         20
           16384 |@                                        114
           32768 |@                                        105
           65536 |@                                        123
          131072 |@@@@@@@@@@@@@                            1726
          262144 |@@@@@@@@@@@                              1515
          524288 |@@@@                                     601
         1048576 |@@                                       282
         2097152 |@                                        114
         4194304 |                                         61
         8388608 |@@@@@                                    660
        16777216 |                                         67
        33554432 |                                         12
        67108864 |                                         7
       134217728 |                                         4
       268435456 |                                         5
       536870912 |                                         0
```

**What is this?**
**(8-16 ms latency)**

# Latency Example: MySQL

```
query time (ns)
           value  ------------- Distribution ------------- count
            1024 |                                         0
            2048 |                                         2
            4096 |@                                        99
            8192 |                                         20
           16384 |@                                        114
           32768 |@                                        105
           65536 |@                                        123
          131072 |@@@@@@@@@@@@@                            1726
          262144 |@@@@@@@@@@@                              1515
          524288 |@@@@                                     601
         1048576 |@@                                       282
         2097152 |@                                        114
         4194304 |                                         61
         8388608 |@@@@@                                    660
        16777216 |                                         67
        33554432 |                                         12
        67108864 |                                         7
       134217728 |                                         4
       268435456 |                                         5
       536870912 |                                         0
```

oh...

```
innodb srv sleep (ns)
           value  ------------- Distribution ------------- count
         4194304 |                                         0
         8388608 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 841
        16777216 |                                         0
```

# Latency Example: MySQL

- Spike of MySQL query latency: 8 - 16 ms
- innodb thread concurrency back-off sleep latency: 8 - 16 ms
- Both have a similar magnitude (see “count” column)
- Add the dimension of time as a heat map, for more characteristics to compare
- ... quickly compare heat maps from different components of the cloud to pattern match and locate latency

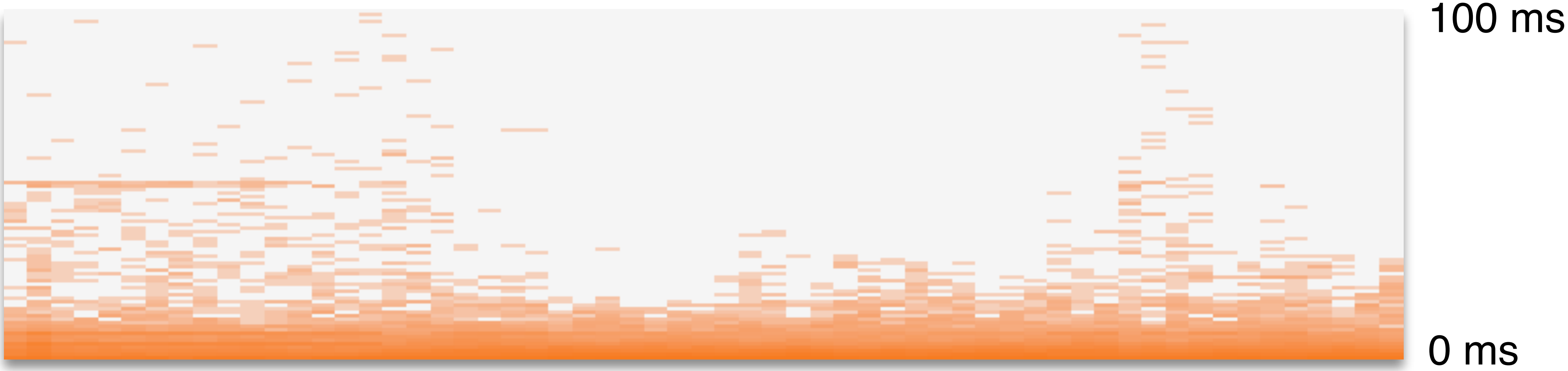
# Cloud Latency Heat Map

- Identify latency outliers, distributions, patterns
- Can add more functionality to identify these by:
  - cloud node
  - application, cloud-wide
  - I/O type (eg, query type)
- Targeted observability (DTrace) can be used to fetch this
- Or, we could collect it for everything
  - ... do we need a 4th dimension?

# 4th Dimension!

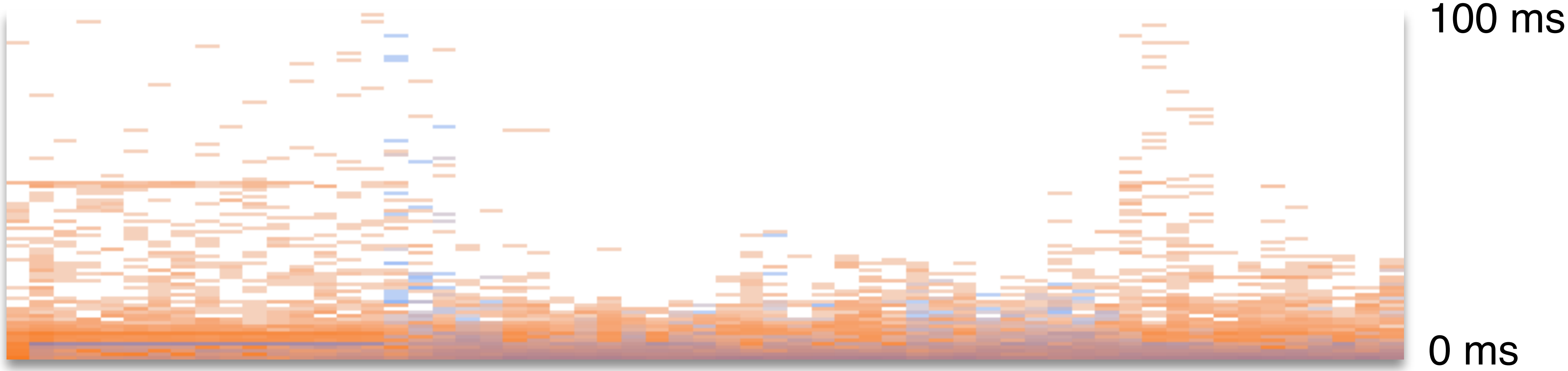
- Bryan Cantrill @Joyent coded this 11 hours ago
  - assuming it's now about 10:30am during this talk
  - ... and I added these slides about 7 hours ago

# 4th Dimension Example: Thread Runtime




- x-axis: time
- y-axis: thread runtime
- z-axis (color saturation): count at that time/runtime range

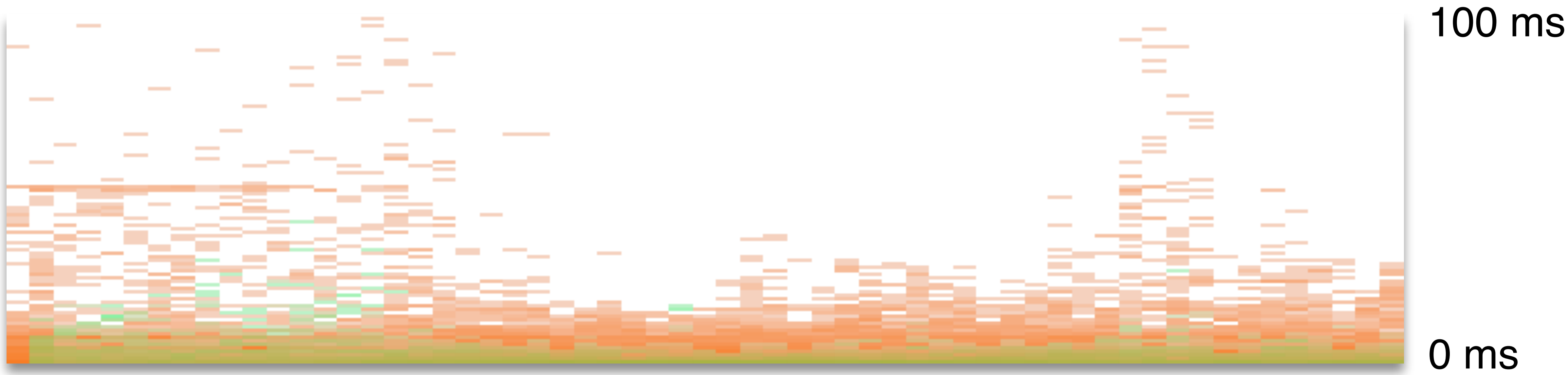
# 4th Dimension Example: Thread Runtime




- x-axis: time
- y-axis: thread runtime
- z-axis (color saturation): count at that time/runtime range
- omega-axis (color hue): application

- blue == “coreaudiod”

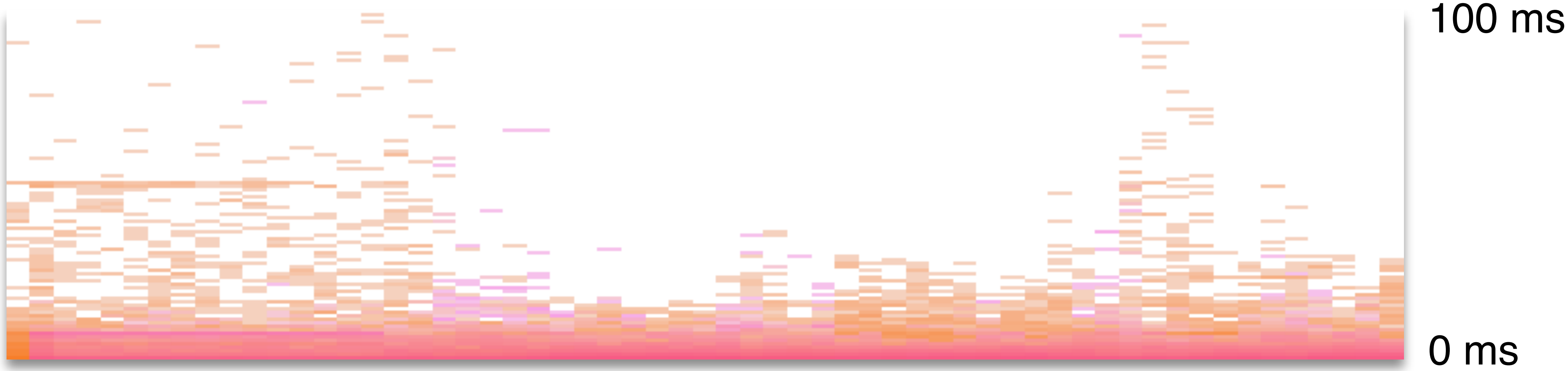
# 4th Dimension Example: Thread Runtime




- x-axis: time
- y-axis: thread runtime
- z-axis (color saturation): count at that time/runtime range
- omega-axis (color hue): application

- green == “iChat”

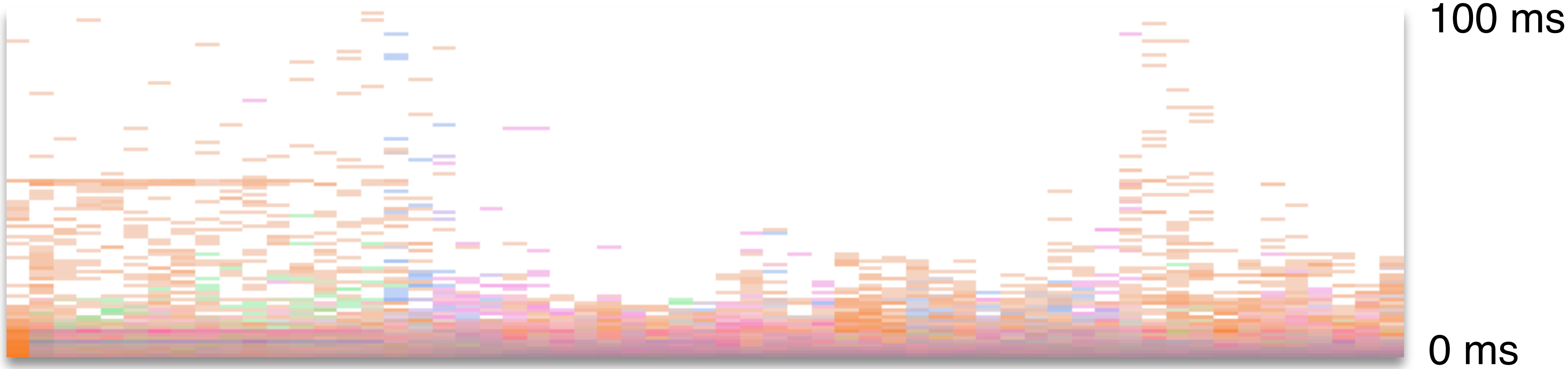
# 4th Dimension Example: Thread Runtime




- x-axis: time
- y-axis: thread runtime
- z-axis (color saturation): count at that time/runtime range
- omega-axis (color hue): application

- violet == “Chrome”

# 4th Dimension Example: Thread Runtime




- x-axis: time
- y-axis: thread runtime
- z-axis (color saturation): count at that time/runtime range
- omega-axis (color hue): application

- All colors

# “Dimensionality”

- While the data supports the 4th dimension, visualizing this properly may become difficult (we are eager to find out)
  - The image itself is still only 2 dimensional
- May be best used to view a limited set, to limit the number of different hues; uses can include:
  - Highlighting different cloud application types: DB, web server, etc.
  - Highlighting one from many components: single node, CPU, disk, etc.
- Limiting the set also helps storage of data

# More Visualizations

- We plan much more new stuff
  - We are building a team of engineers to work on it; including Bryan Cantrill, Dave Pacheo, and mysqlf
  - Dave and I have only been at Joyent for 2 1/2 weeks

# Beyond Performance Analysis

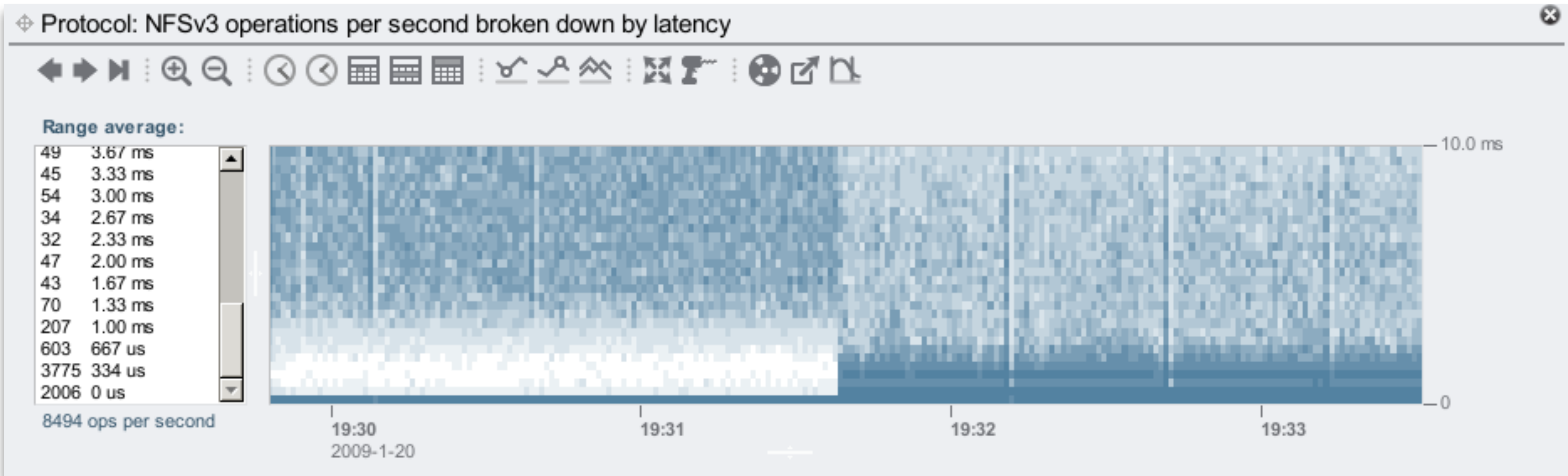
- Visualizations such as heat maps could also be applied to:
- Security, with pattern matching for:
  - robot identification based on think-time latency analysis
  - inter-keystroke-latency analysis
  - brute force username latency attacks?
- System Administration
  - monitoring quota usage by user, filesystem, disk
- Other multi-dimensional datasets

# Objectives

Joyent

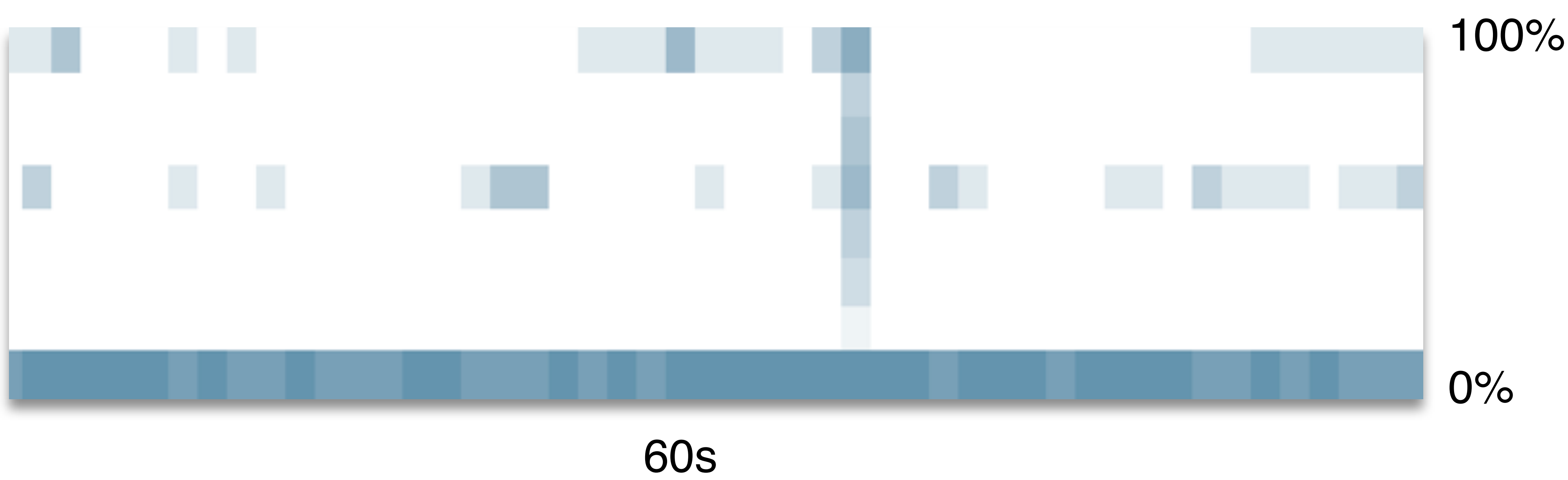
- Consider performance metrics before plotting
  - Why latency is good
  - ... and IOPS can be bad
- See the value of visualizations
  - Why heat maps are needed
  - ... and line graphs can be bad
- Remember key examples
  - I/O latency, as a heat map
  - CPU utilization by CPU, as a heat map

# Heat Map: I/O Latency




- Latency matters
  - synchronous latency has a direct impact on performance
- Heat map shows
  - outliers, balance, cache layers, patterns

# Heat Map: CPU Utilization




- Identify single threaded issues
    - single CPU hitting 100%
- Heat map shows
    - fully utilized components, balance, overall headroom, patterns

# Tools Demonstrated

- For Reference:
- DTraceTazTool
  - 2006; based on TazTool by Richard McDougall 1995.  Open source, unsupported, and probably no longer works (sorry).
- Analytics
  - 2008; Oracle Sun ZFS Storage Appliance
- “new stuff” (not named yet)
  - 2010; Joyent; Bryan Cantrill, Dave Pacheco, Brendan Gregg

# Question Time

- Thank you!

- How to find me on the web:
  - http://dtrace.org/blogs/brendan
  - http://blogs.sun.com/brendan <-- is my old blog
  - twitter @brendangregg