DELPHIX

Lies, Damn Lies and I/O Statistics

In collaboration with

Adam Leventhal

Matt Hayward

Kyle Hailey
kyle@delphix.com
http://kylehailey.com

Content

Part I - tests

- Expectations
- Anomalies
- Summary

Part II - reports

- Visualizing I/O Performance
- I/O report card for Oracle databases

Predictions

Given Gas Dynamics

Hard to predict Tornadoes

Modern storage systems are complex

Forecasting performance difficult

Emergent behavior

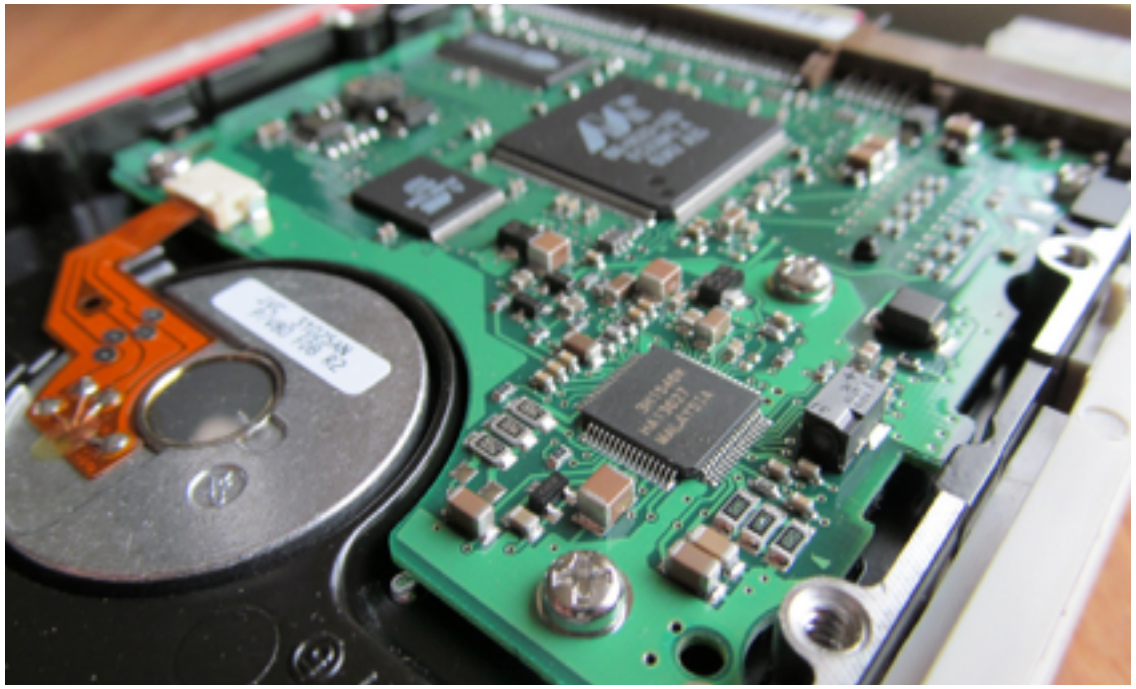
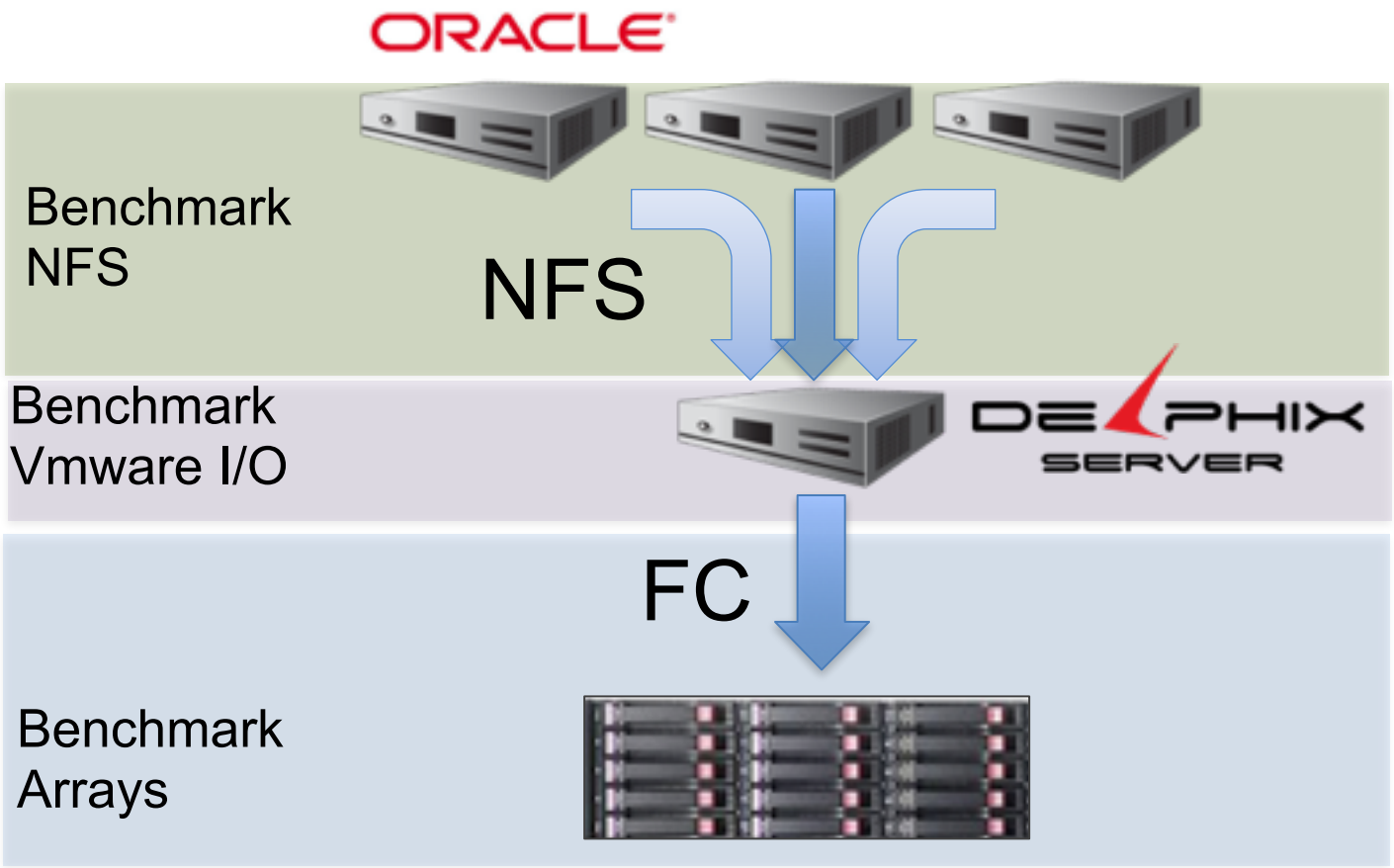


Photo by Clive Darra

At Delphix



moats.sh *
strace
dtrace

ioh.sh
dtrace

DELPHIX

Moats

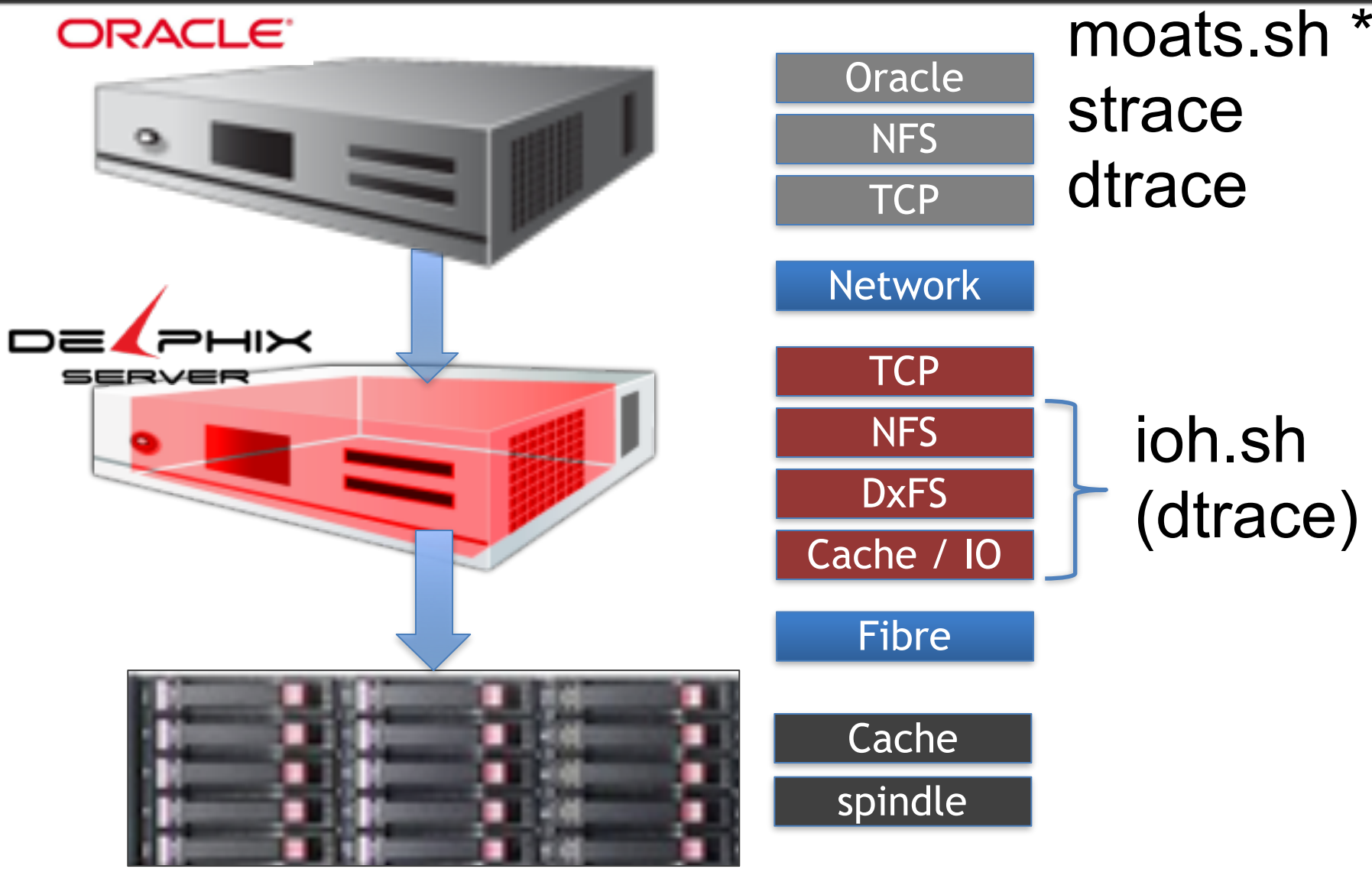
https://github.com/khailey/moats

```
MOATS: The Mother Of All Tuning Scripts v1.0 by Adrian Billington & Tanel Poder
       http://www.oracle-developer.net & http://www.e2sn.com

+ INSTANCE SUMMARY ------------------------------------------------------------------------------
| Instance: V1                | Execs/s:  3050.1 | sParse/s:    205.5 | LIOs/s:    28164.9 | Read MB/
| Cur Time: 18-Feb 12:08:22   | Calls/s:   633.1 | hParse/s:      9.1 | PhyRD/s:    5984.0 | Write MB
| History:  0h 0m 39s         | Commits/s: 446.7 | ccHits/s:   3284.6 | PhyWR/s:    1657.4 | Redo MB/
+------------------------------------------------------------------------------------------------

|            event name avg ms   1ms   2ms   4ms   8ms  16ms  32ms  64ms 128ms 256ms 512ms    1s
| db file scattered rea   .623     1
| db file sequential re  1.413 13046  8995  2822   916   215     7                 1
|      direct path read  1.331    25    13     3           1
| direct path read temp  1.673
|     direct path write  2.241    15    12    14     3
| direct path write tem  3.283
| log file parallel wri
|         log file sync

+ TOP SQL_ID (child#) -----+ TOP SESSIONS ---------+      + TOP WAITS ------------------------+ W
|  19% |  ()               |                       |      |  60% | db file sequential read     | U
|  19% | c13sma6rkr27c (0) | 245,147,374,386,267   |      |  17% | ON CPU                      | C
|  17% | bymb3ujkr3ubk (0) | 131,10,252,138,248    |      |  15% | log file sync               | C
|   9% | 8z3542ffmp562 (0) | 133,374,252,250       |      |   6% | log file parallel write     | S
|   9% | 0yas01u2p9ch4 (0) | 17,252,248,149        |      |   2% | read by other session       | U
+-----------------------------------------------+      +-------------------------------------
```

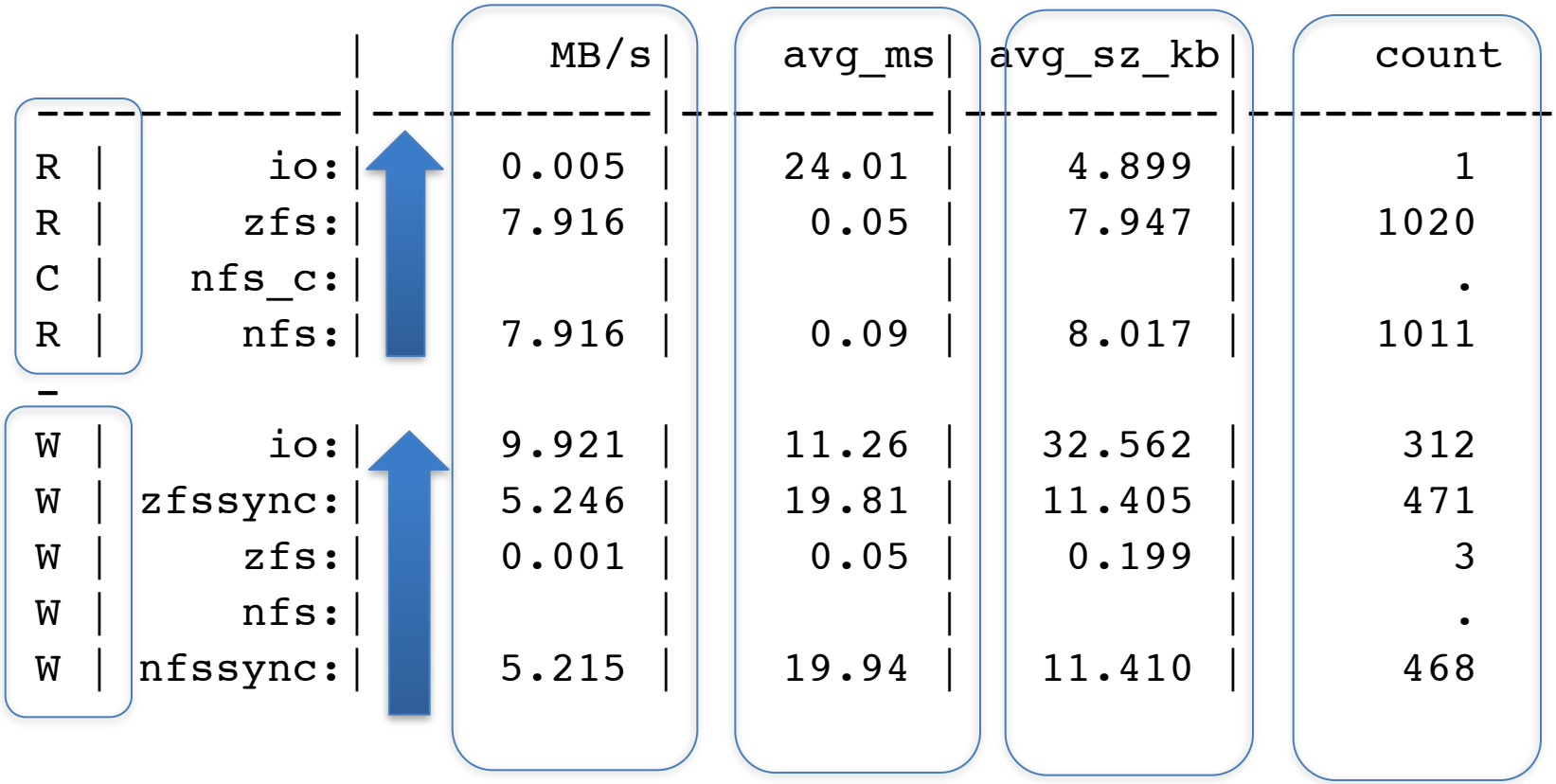


https://github.com/khailey/moats
https://github.com/khailey/ioh

ioh.sh

```
date: 1335282287 , 24/3/2012 15:44:47
TCP out:  8.107 MB/s, in:  5.239 MB/s, retrans:0 MB/s, ip discards:

                |      MB/s|    avg_ms| avg_sz_kb|     count
----------------|----------|----------|----------|--------------
R |        io:|     0.005|     24.01|     4.899|             1
R |       zfs:|     7.916|      0.05|     7.947|          1020
C |    nfs_c:|          |          |          |             .
R |       nfs:|     7.916|      0.09|     8.017|          1011
-
W |        io:|     9.921|     11.26|    32.562|           312
W |  zfssync:|     5.246|     19.81|    11.405|           471
W |       zfs:|     0.001|      0.05|     0.199|             3
W |       nfs:|          |          |          |             .
W |  nfssync:|     5.215|     19.94|    11.410|           468
```



https://github.com/khailey/ioh

ioh.sh

```
--- NFS latency by size ---------
    ms    sz_kb 32u    64u    .1m    .2m    .5m    1m     2m     4m     8m     16m    33m    65m
R   0.1     8     .   2909   2023     87     17     3      .      .      .      1      2      2
R   0.1    16     .      4      5      2
-
W   5.0     4     .      .      .      .      8     49     10      3      4     11      4      2
W  21.4     8     .      .      .      .      4     55    196     99    152    414    490    199
W  18.3    16     .      .      .      .      .     34     29     25     43     91     84     28
W  16.1    32     .      .      .      .      .     19     16      7     14     38     36     15
W  19.3    64     .      .      .      .      .      6     11      .      9     11     19      6
W  20.4   128     .      .      .      .      .      1      3      .      .      2      4      .
```

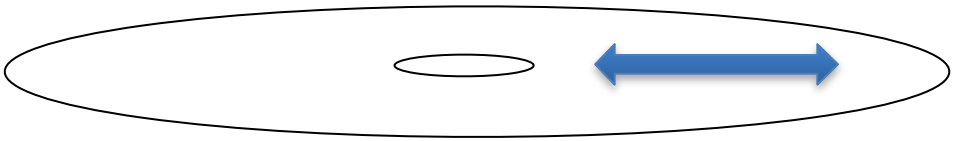
Along the road

- Rules of thumb
- Pitfalls
- Summarized Lessons

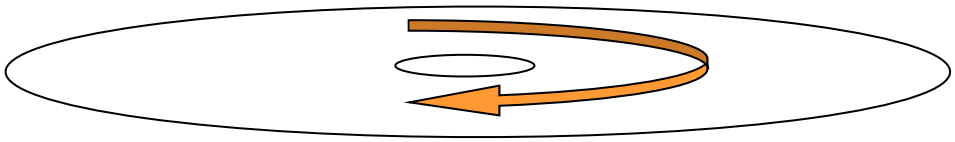
Photo by Helgi Halldórsson

Expectations

1. Seek Times
   - 2ms-8ms



2. Rotational latency, avg = ½ time for 1 rotation
   1. 7000 RPM = 4.2 ms
   2. 10000 RPM = 3 ms
   3. 15000 RPM = 2 ms



Total Times Avg

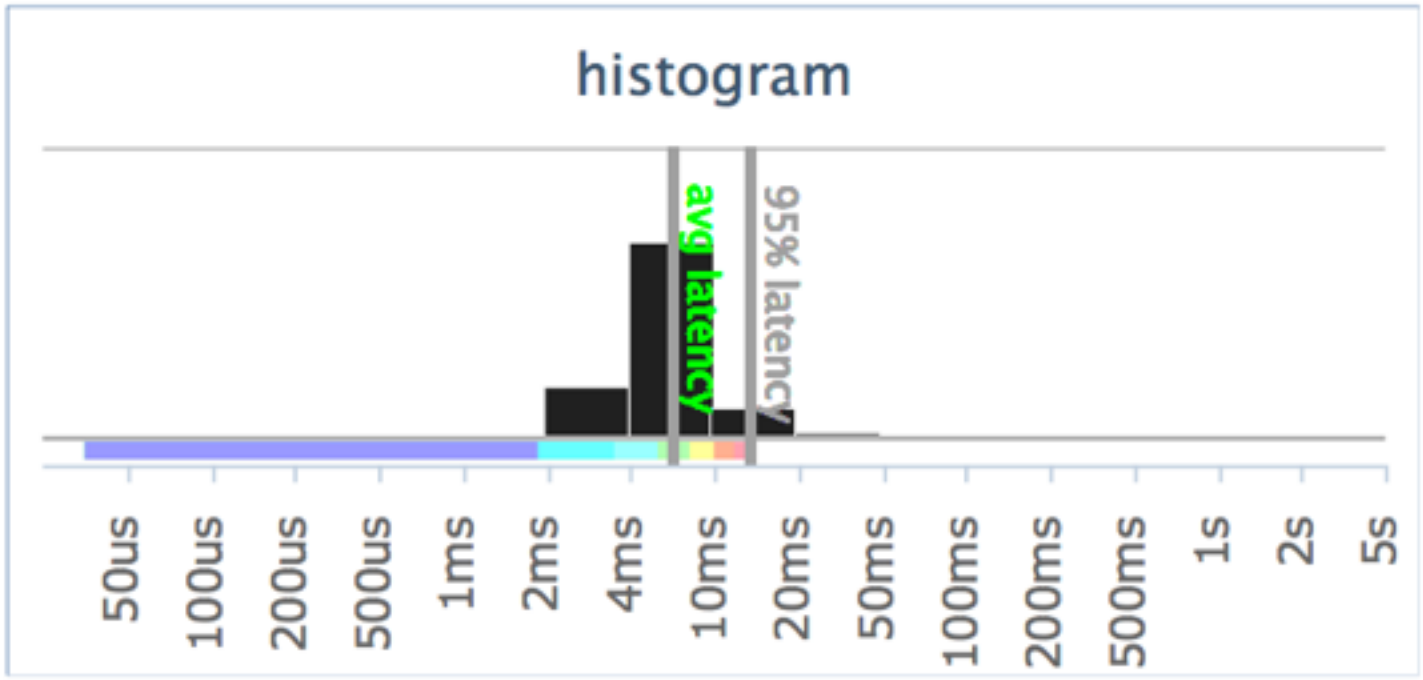
1. 7000RPM = 6 – 13 ms
2. 10000RPM = 5 – 11 ms
3. 15000RPM = 4 – 10 ms

speed	avg_rot_lat	seek	total
7.2K	4ms	4.8ms	= 8.8
10K	3ms	4.3ms	= 7.3
15K	2ms	3.8ms	= 5.8

6 ms = good

Expected distribution

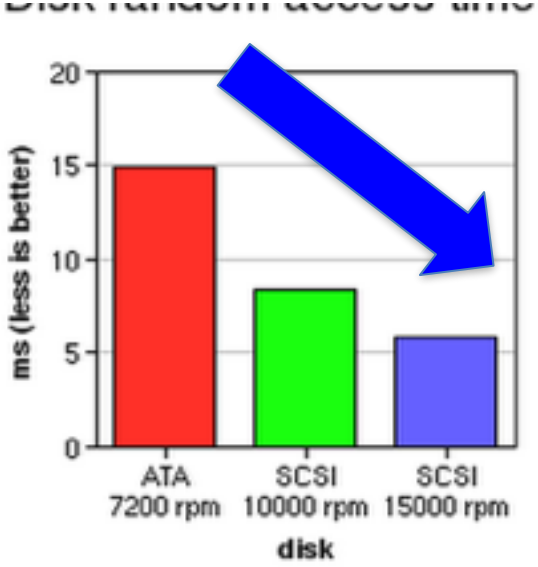
around 6ms on a fast system (real data)



7K, 10K, 15K

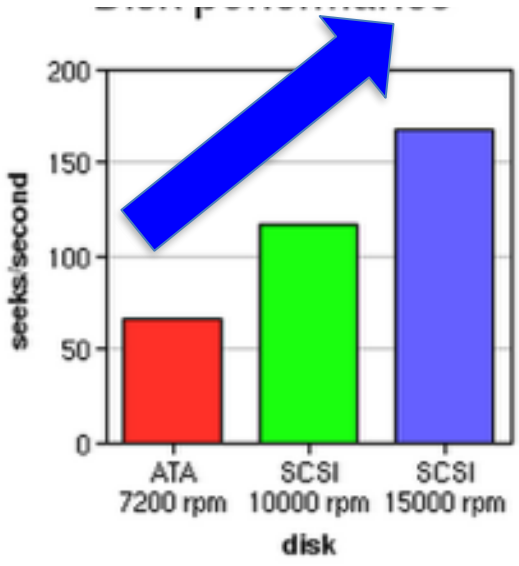
Random 512byte

Latency



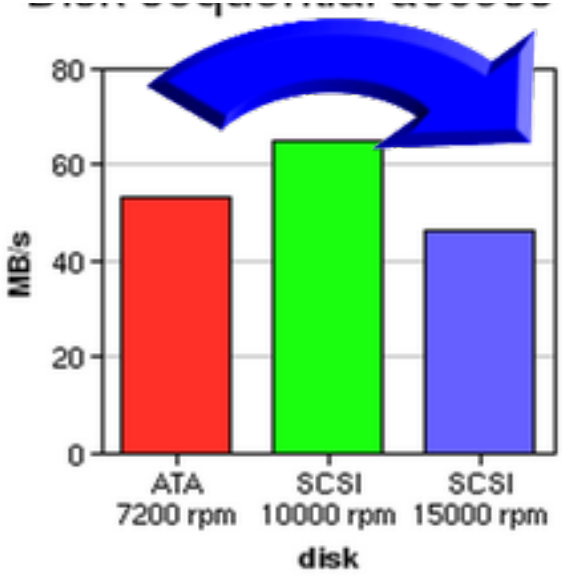
Random 512byte

IOPs



Sequential 1M

Throughput

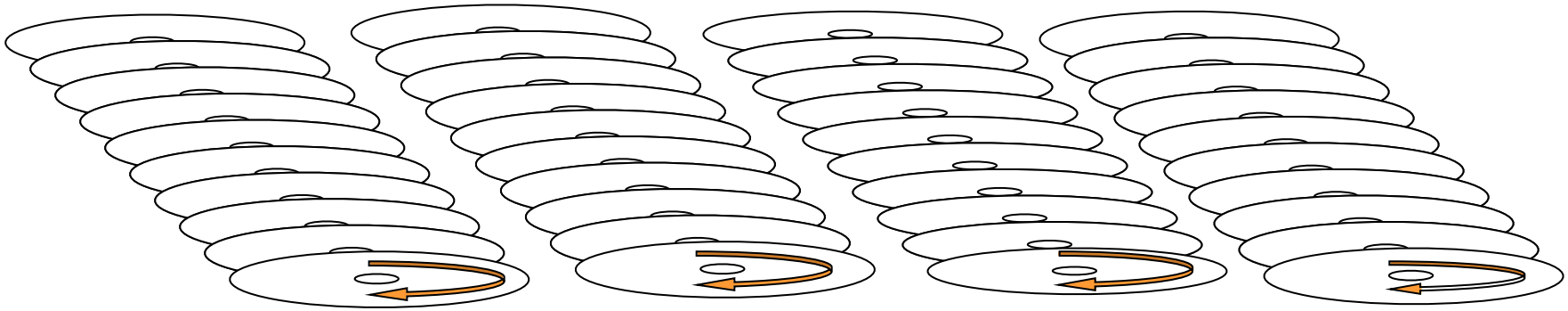


Disk 1: ATA 120GB, Seagate Barracuda 7200.7 Plus (ST3120026A), **8MB** cache, 7200 rpm
Disk 2: SCSI 36GB, Seagate Cheetah 10K.6 (ST336607LC), **8MB** cache, 10000 rpm
Disk 3: SCSI 18GB, IBM Ultrastar 36Z15 (IC35L018UCPR15-0), **4MB** cache, 15000 rpm

http://www.linuxinsight.com/how_fast_is_your_disk.html

IO Throughput

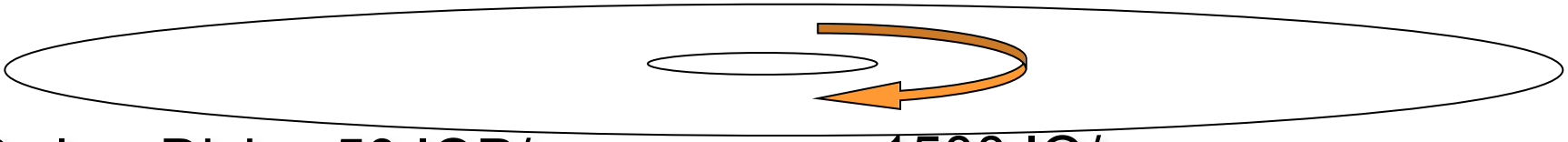
IOPs ~= (number of disks) * (IOP per disk)



IO Throughput

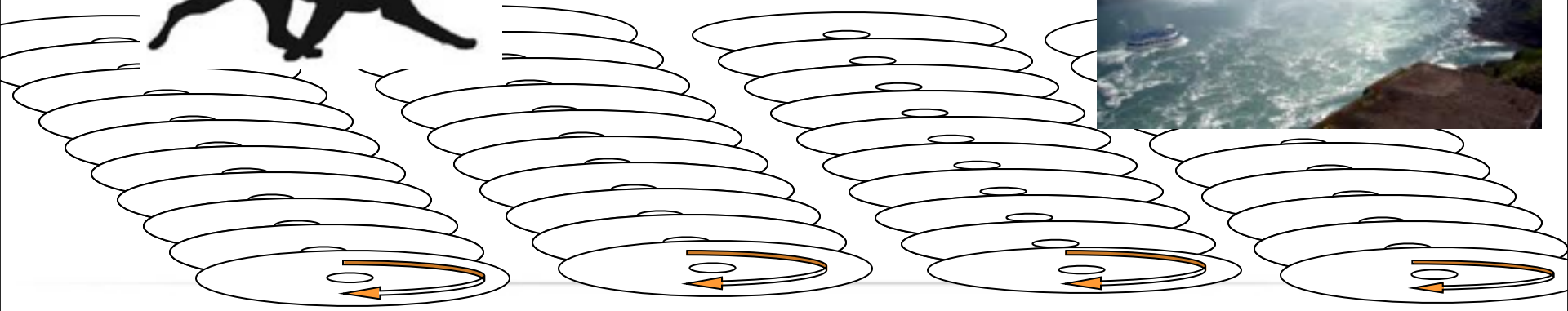
1 Fast Disk

150 IO/sec

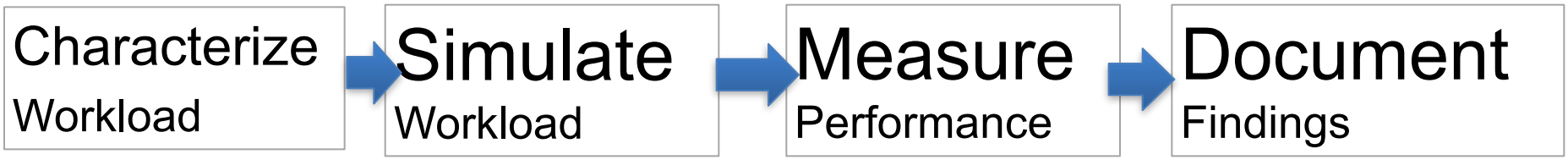


30 slow Disks 50 IOP/s

1500 IO/sec



Original Idea

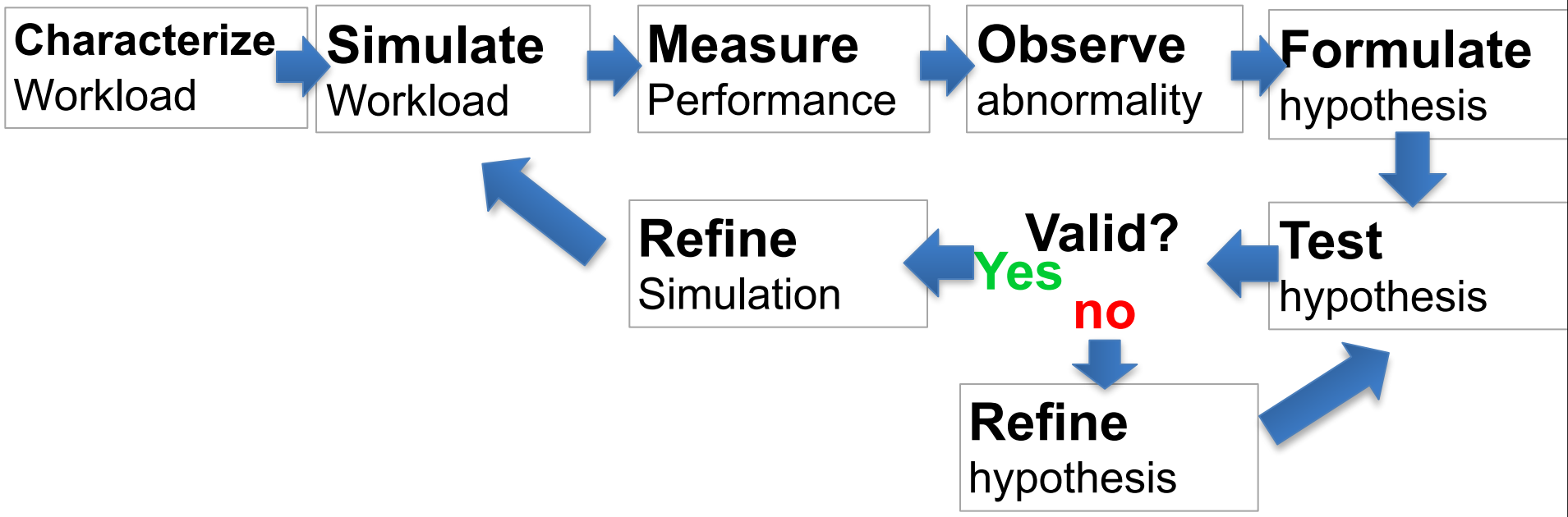


Benchmarking I/O should be easy

Can't I just use "dd"?

- dd writes to memory not disk (by default)
- dd reads from file system cache (by default)
- Difficult to manage multiple readers/writers
- dd is sequential, how do you test random read ?

Actual Process



Top 10 Anomalies

1. Caching
2. Shared drives
3. Shared connection
4. Consolidation of I/O Request
5. Fragmentation of I/O Request
6. Tiered Storage Migration
7. First Write Penalty
8. Elided Reads
9. Compressed I/O
10. Storage Maintenance

1. Caching

Symptom: Impossibly good performance

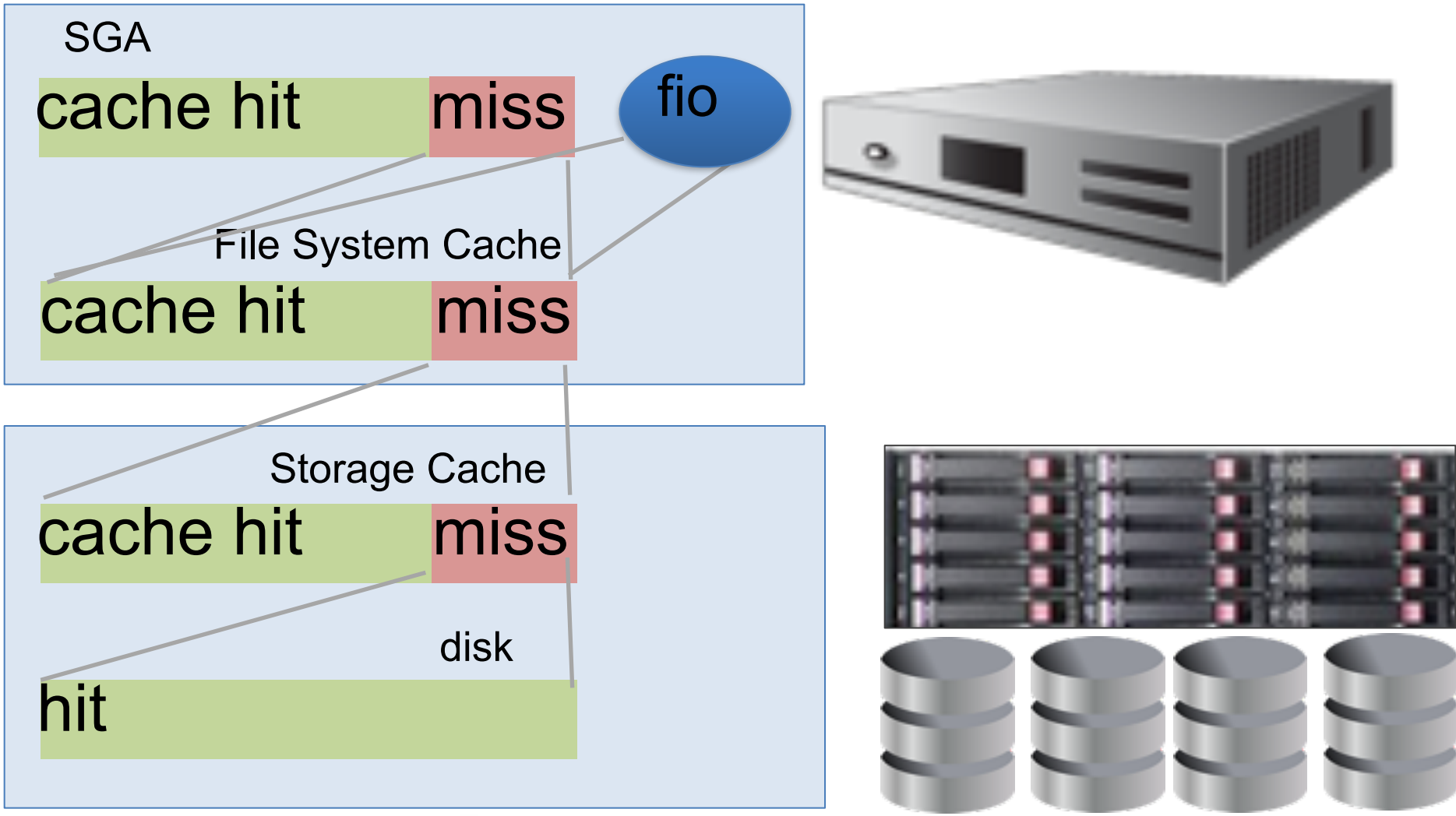
- Higher throughout than interconnect provides
- Faster latency than speed of light over the distance

Causes

- O/S has file system cache
- Storage Arrays have cache
- Drivers have small caches

Problem mainly reads

1. Caching


SGA
cache hit
miss
fio
File System Cache
cache hit
miss
Storage Cache
cache hit
miss
disk
hit

Two Databases: Dev slower than Prod

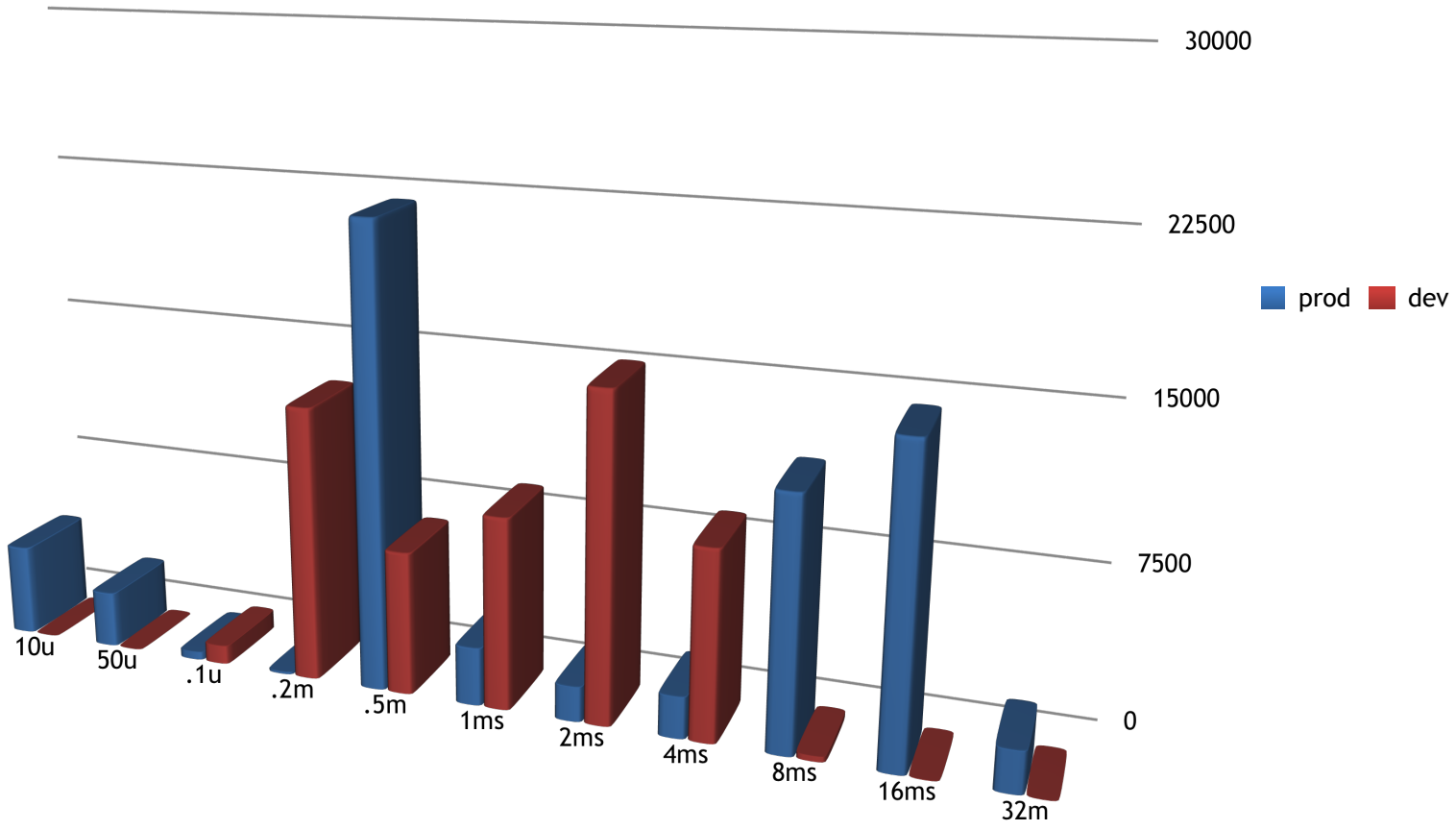
db file sequential read

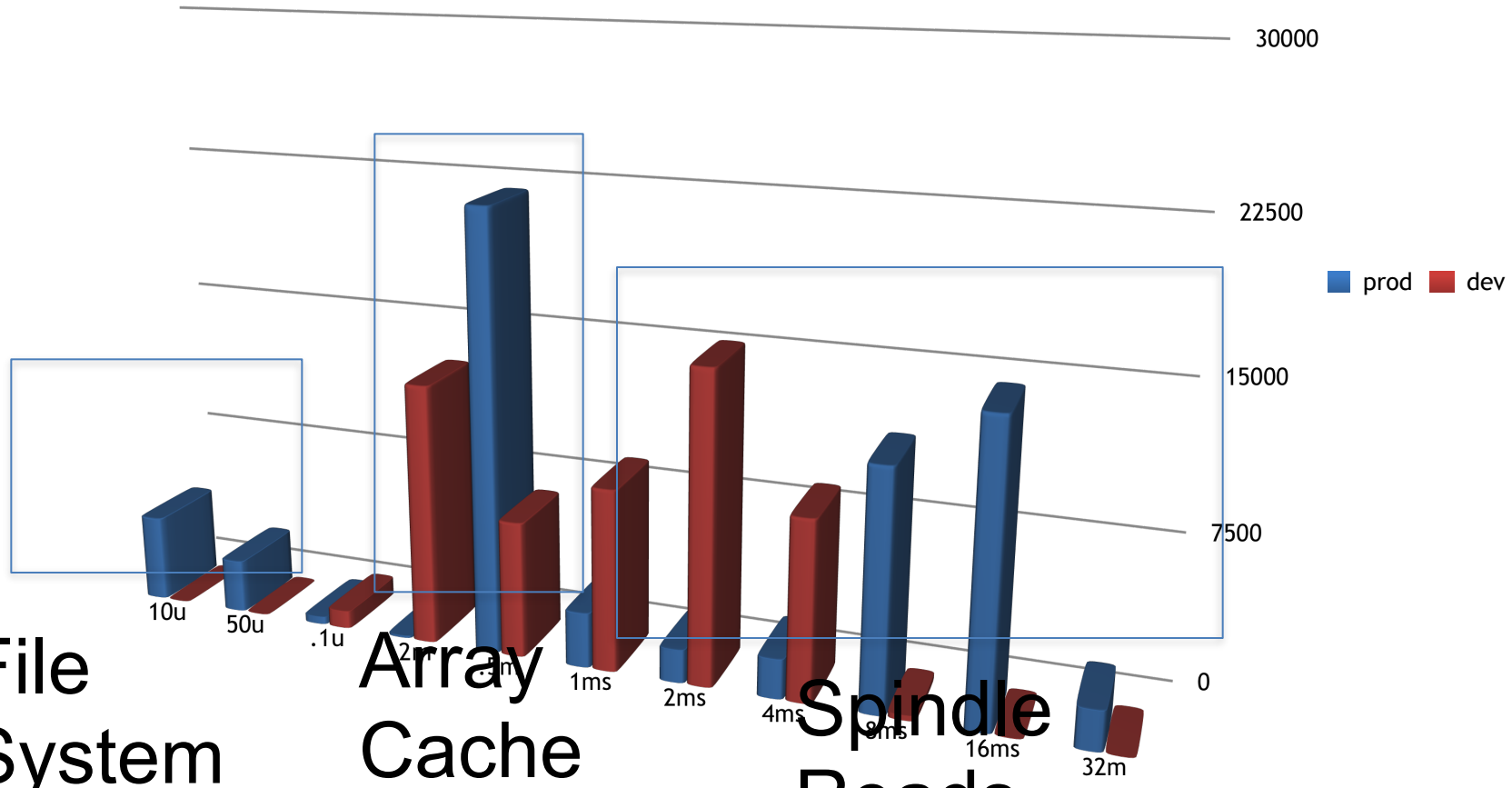
db	Times Waited	Elapsed (ms)	Avg Ela (ms)
dev	55528	479930	9
prod	65275	294785	5

10046 trace

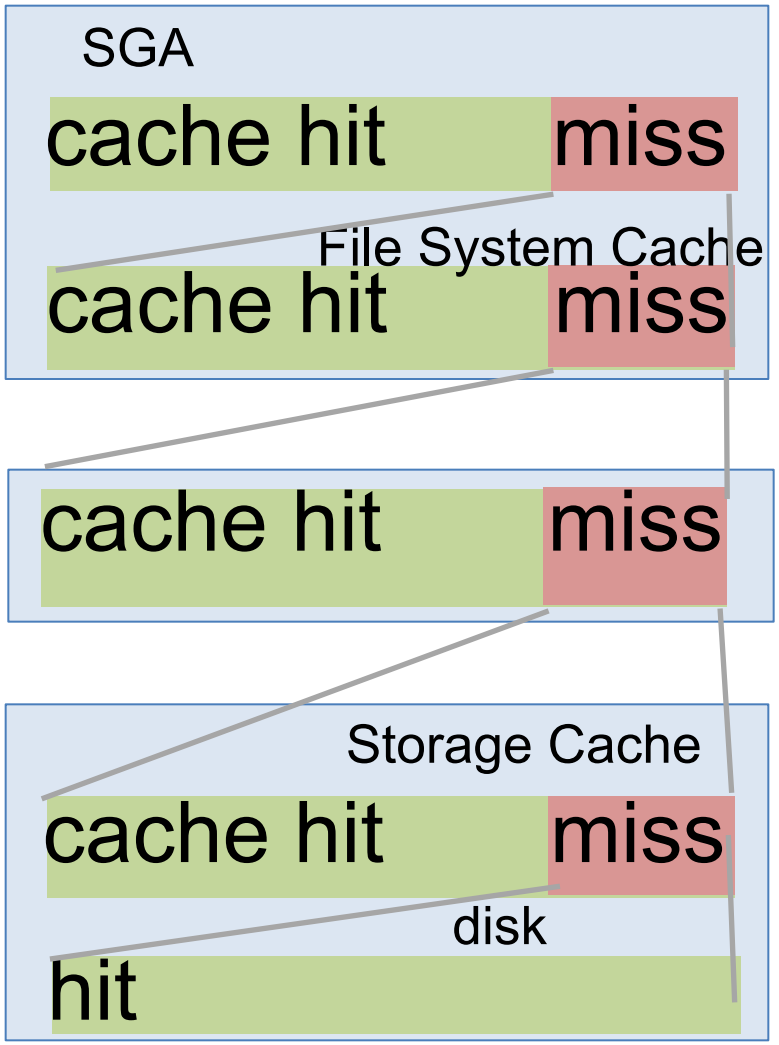
db	10u	50u	.1m	.2m	.5m	1ms	2ms	4ms	8ms	16ms	32m	64m	.1	.5	>.5
dev		1	14	908	13238	6900	9197	15603	9056	265	26	12	12		
prod	4419	2713	391	118	22189	2794	1688	2003	11969	14877	2003	105	3	3	

https://github.com/khailey/oracle_trace_parsing


30000
22500
15000
7500
0
prod
dev
10u
50u
.1u
.2m
.5m
1ms
2ms
4ms
8ms
16ms
32m


30000
22500
15000
7500
0
prod
dev
10u
50u
.1u
1ms
2ms
4ms
16ms
32m
File System Cache
Array Cache
Spindle Reads

1. Caching



Oracle Host

Delphix Host

SAN

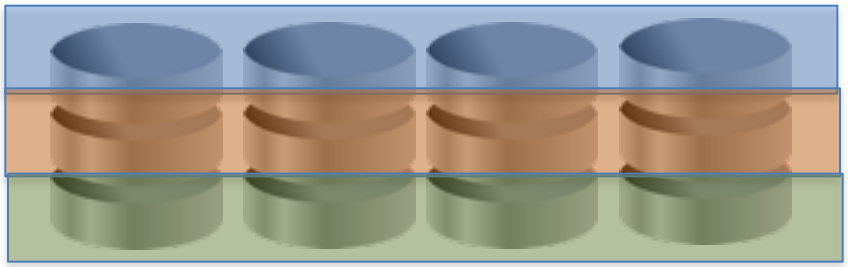
1. Caching Solutions

- Use Direct I/O
  - Writes
    - dd if=/dev/zero of=toto bs=8k count=100
    - dd if=/dev/zero of=toto bs=8k count=100 oflag=direct
  - Reads
    - dd of=/dev/null if=toto bs=8k count=100
    - dd of=/dev/null if=toto bs=8k count=100 iflag=direct
- ZFS
  - doesn't have Direct I/O
  - Set caching to metadata
    - zfs set primarycache=metadata $FILESYSTEM
- Array caching
  - use dataset that is significantly bigger than the cache
  - Monitor with I/O latency histograms

NOTE: dd is sub-optimal for I/O testing. Recommend fio

2. Shared drives

- Inconsistent performance



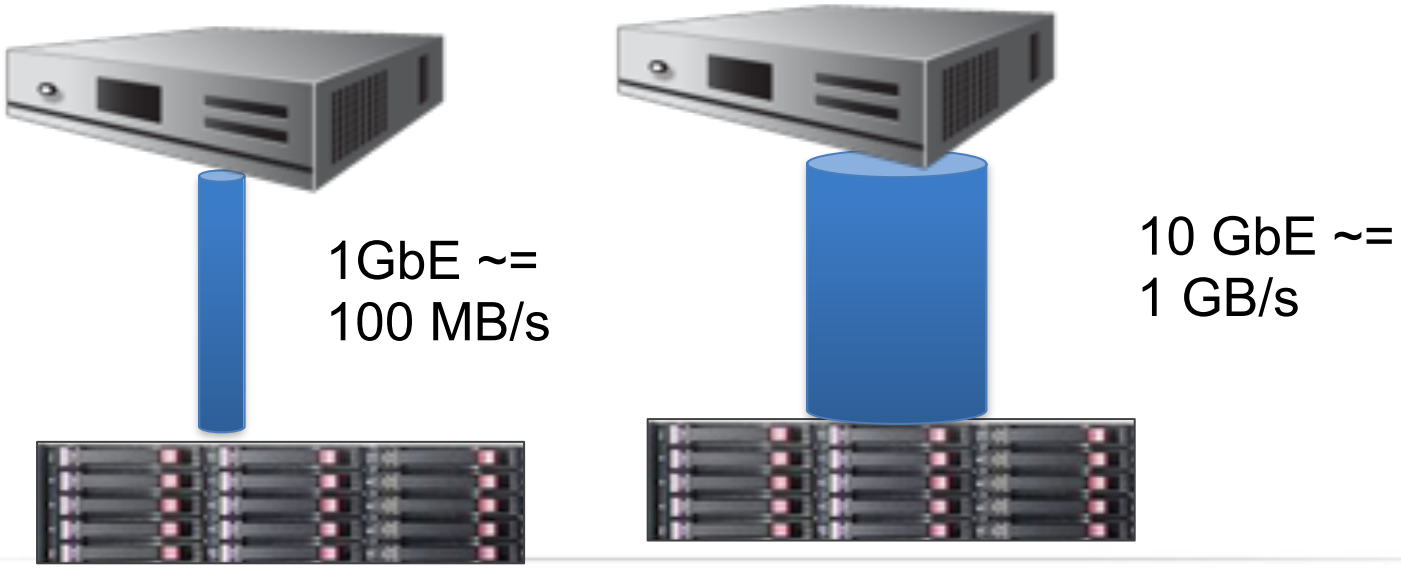
Common: LUNs carved from many spindles shared with other LUNs

Example:

If benchmarking LUN1 and load is changing on LUN2 then it results will vary

3. Connection limits & shared connections

- Inconsistent performance
  - High throughput tests
  - Pipe Size:
    - NAS NICs 1 GbE vs 10GbE vs 40GbE
    - SAN HBA 1G, 4G, 8G, 16G
  - Sharing the line
    - VMware: other VMs for access to the storage
    - Converged network (combines HBA and NIC functionality)



4. I/O Request Consolidation

I/O layers consolidate multiple I/Os

- Paradoxically higher than expected
  - Latency , slower
  - Throughput, more
- Writes
  - Small sequential
  - Over NFS
  - Without O_SYNC and without O_DSYNC

Client used Strace or Dtrace

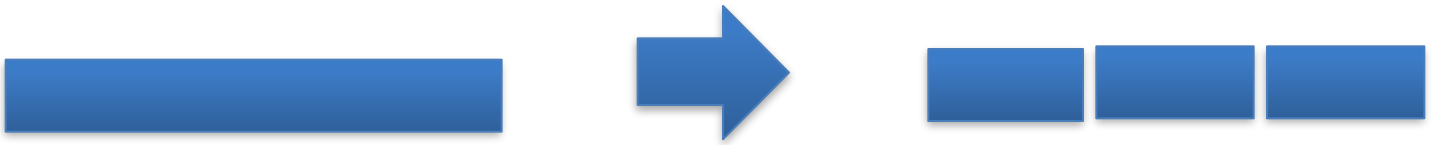
```
pwrite(int fd,
        const void *buf,
        size_t count,
        off_t offset);
 pwrite(259,
        "\1\"\0\0\20\"...,
        1024,
        2891776)
```

Server used DTrace

Many sequential writes can become one larger write request

5. I/O Request Fragmentation

- Higher latency
- lower throughput
- Seen with
  - large I/Os
  - NFS based NAS storage



Large application I/O requests can be broken down into multiple, smaller I/Os that are issued serially by intervening layers.

5. I/O Request Fragmentation: NFS

- Max size set in NFS mount parameters
- SunOS by default sends maximum of 32K
  - Change with nfs3_bsize to 1M
- AIX sends maximum of 64K
  - Changeable with a patch to 512K
- Sun OS : 1M write O_SYNC -> 8k writes
- LINUX: Without Direct I/O, 1M write becomes
    - 1 M unstable NFS write
    - Followed by NFS comit
  - Other OS's issue 1 M NFS Sync write. LINUX with Direct I/O issues does as well.

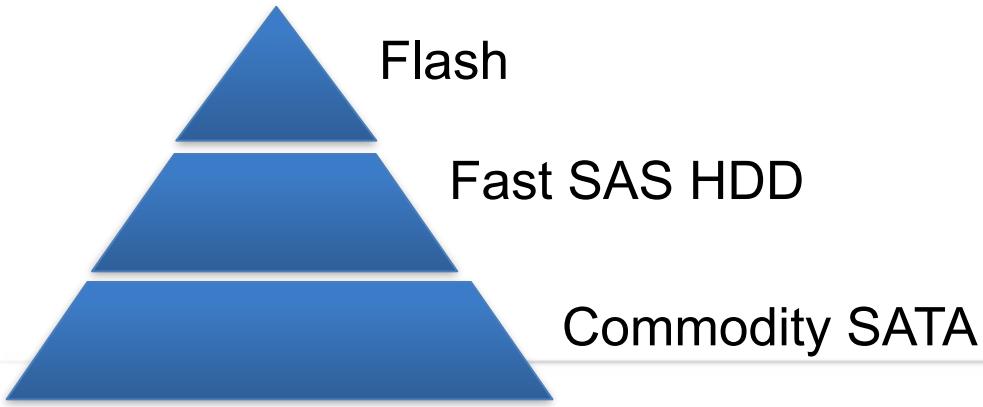
synchronous / asynchronous - blocking or non-blocking
synchronized / nonsynchronized - stable storage semantics O_[D]SYNC

6. Read Ahead

- Symptoms:
  - Improbably fast sequential read
  - Unexpectedly poor random I/O performance
  - Performance changes dramatically midway through a test
- Cause
  - O/S or Array optimistically fetch data adjacent to the requested data in case it is needed.
    - sequential read workload => improve performance.
    - random read workload => do unnecessary work
  - Systems may try to discern the random or sequential
    - example a sequential read test may start slowly and then speed up once read ahead kicks in.

7. Tiered Storage Migration

- Symptom
  - Bad performance early in tests
  - Fast performance later in tests or subsequent tests
- Example
  - high performance Enterprise Storage shows
  - Initial tests show 8 kB random read latencies
    - averaging around 20 ms
    - spikes to around 100 ms



8. First Write Penalty

- Symptom
  - Unexpectedly bad write performance
  - Especially early in testing and is not reproducible
- Cause
  - Storage system uses thin provisioning
  - First writes can require
    - Meta data adjustments
    - Region formatting
- Example
  - A thin provisioned VMDK on VMware must be zeroed on first write
  - A 1 kB application write can trigger a 1 MB write

9. Elided Reads

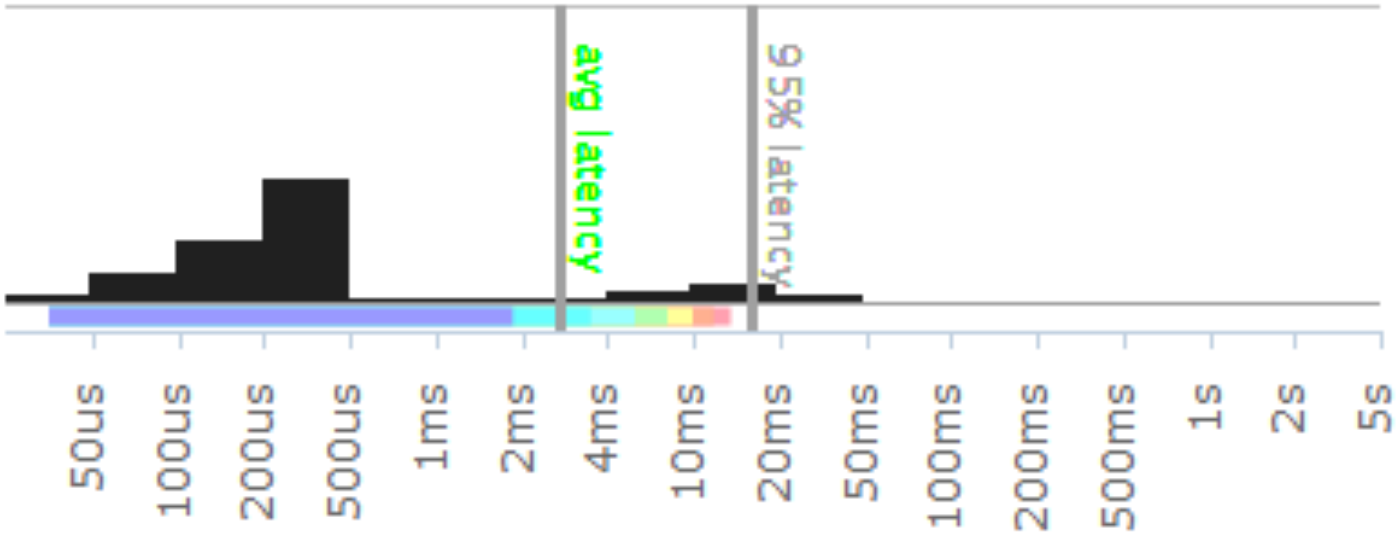
- Problem: Unexpectedly good read performance
  - raw devices
  - regions that have not been written
- Solution: initialize all regions being used

Reads respond immediately for uninitialized regions.

- VMFS
- ZFS

will do this, depending on configuration.

Elided Reads



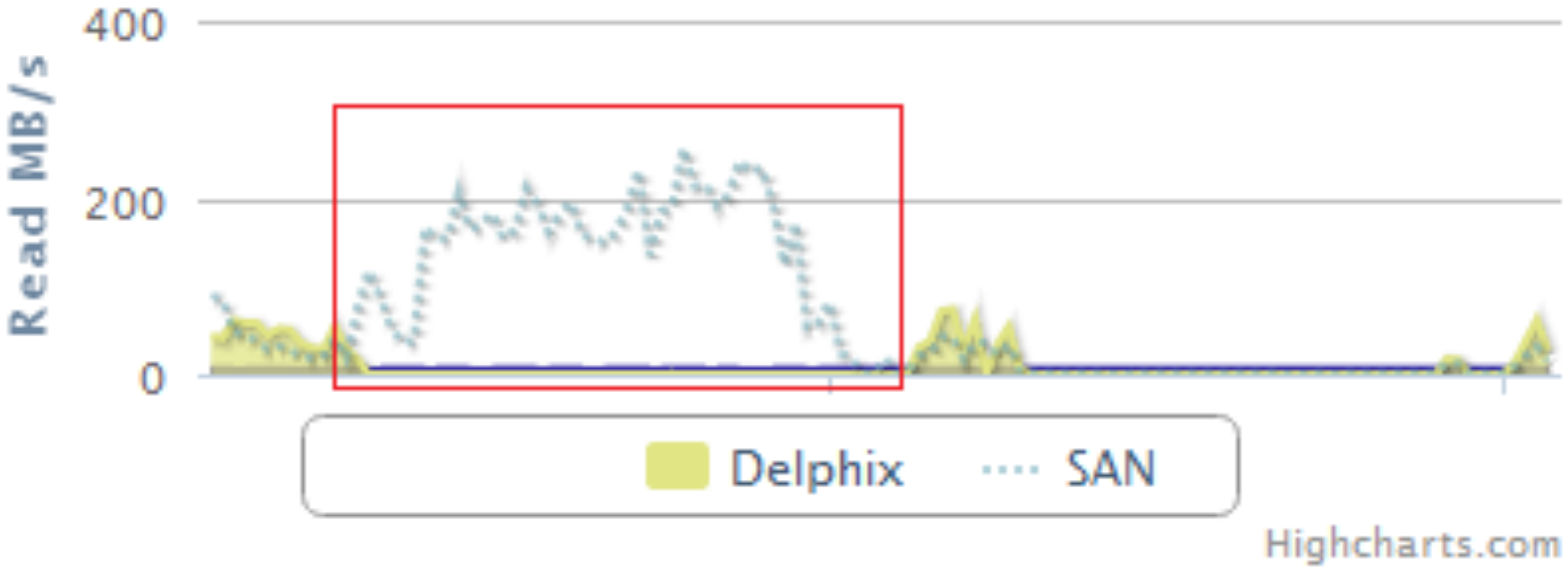
10. Compressed I/O

- Problem: Unexpected, or even impossibly good write or read performance
- Solution: use real or randomized data

Data that compresses well, like all zeros or all ones, compressed beyond realistic limits

11. Storage Maintenance

- Problem: Unexpectedly poor performance
- Solution: avoid storage maintenance operations
  - ZFS Scrubs
  - Data migrations
  - RAID config builds or rebuilds



Summary: To avoid anomalies while testing

1. Real workload
   - Or simulate actual application workload
2. Histograms for latencies
   - not averages
3. Reproducible results
4. Run sufficiently long
5. Use application data, or similar

1. Work Load

- Use real work load
- If simulating
  - Simulate actual I/O mix
  - Reads
  - Writes
  - I/O sizes
  - I/O rates
  - O_SYNC or O_DSYNC
  - is Direct I/O used

Tool “fio” is good for simulating I/O work loads

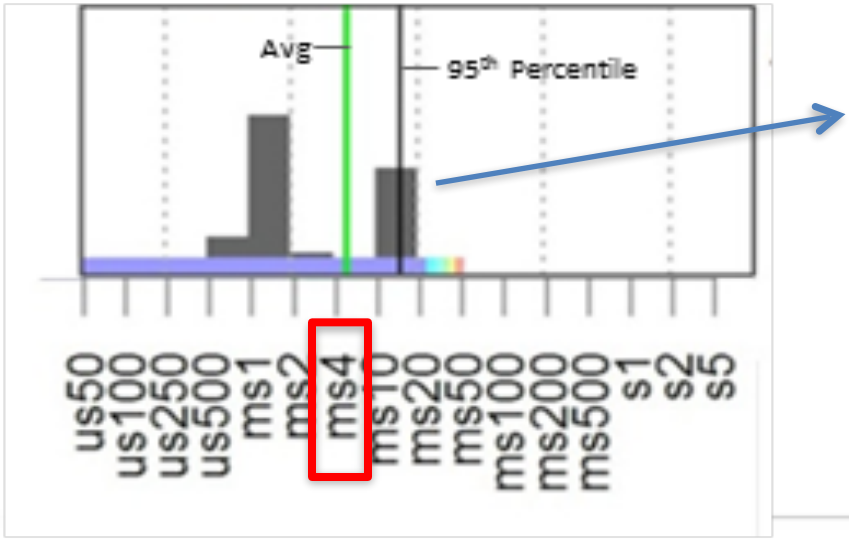
2. latency histograms

- Averages hide information
- Histograms can show caching effects

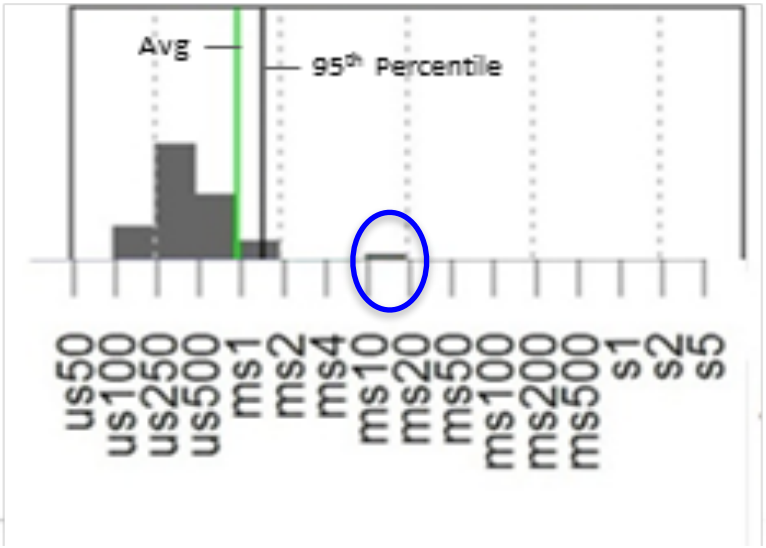
First, sequential read tests was run

Followed by random

128 kB Sequential Read Latency



8 kB Random Read Latency



3. Verify tests are reproducible

- Run tests multiple times
- If performance gets faster, then probably caching effect
- If performance goes up and down
  - then probably shared infrastructure
- If using shared infrastructure
  - Run tests when infrastructure load is actually representative

- NOTE: often told that infrastructure is not shared only to find out later it is

4. Running Time

- sufficient ramp up time
  - Hit steady state
  - Avoid or dynamic workload detection
- One strategy
  - Rep tests
  - Double running time each test
  - Until successive tests are results are similar

5. Ensure data sizes representative

- If eventual application sizes is too large to simulate
  - Then make sure data set sizes are much bigger than cache sizes at least
- Example:
  - 10 TB application
  - Create 10TB of data
  - If that's too big , then use at least 100GB based on
    - Find out array cache size
    - Find out host machine RAM
- Initialize data with real or similar data (not zeros)
  - Avoid first write penalty
  - Avoid instant reads of initialized data
  - Use actual data, not just zeros, to avoid unrealistic compression

Summary

- Important to forecast I/O requirements and performance
- Accurate forecast tricky
- Test workloads can trigger abnormalities
  - Invalidate results
  - Lucky when results are impossible, then results are clearly invalid
  - Unlucky, results will mislead
- Awareness of anomalies can help avoid them
  - Improve accuracy of tests

DELPHIX

Visualizing I/O performance

What ? Several dimensions

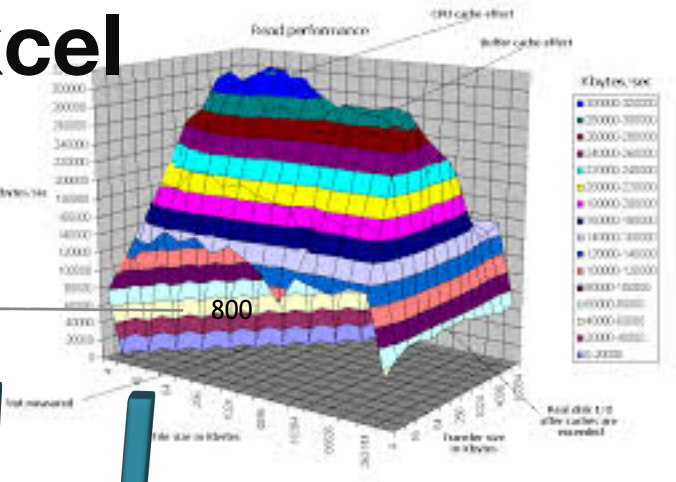
1. # Users
2. Latency
3. MB/s
4. IOP/s

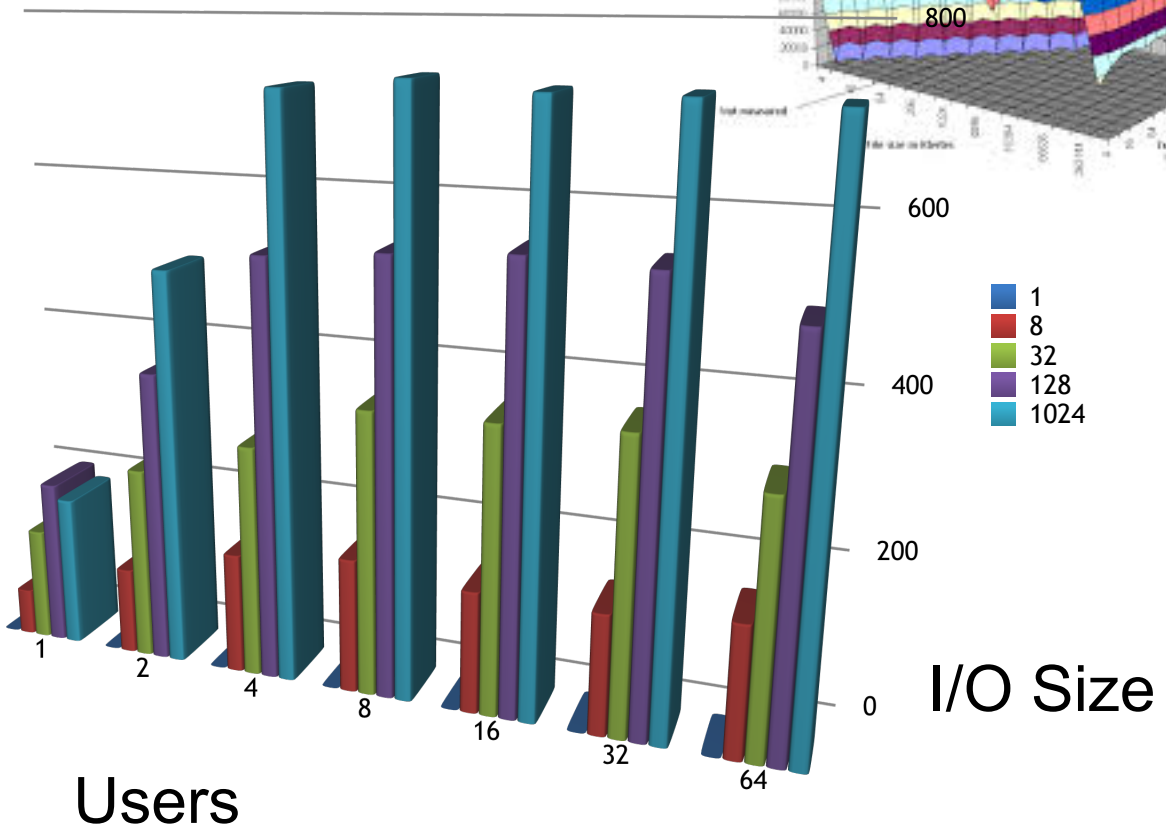
How?

- Line charts
- Histograms
- Multiple charts
- Aggregation Chart

MB/s vs IO Size vs Users : Excel

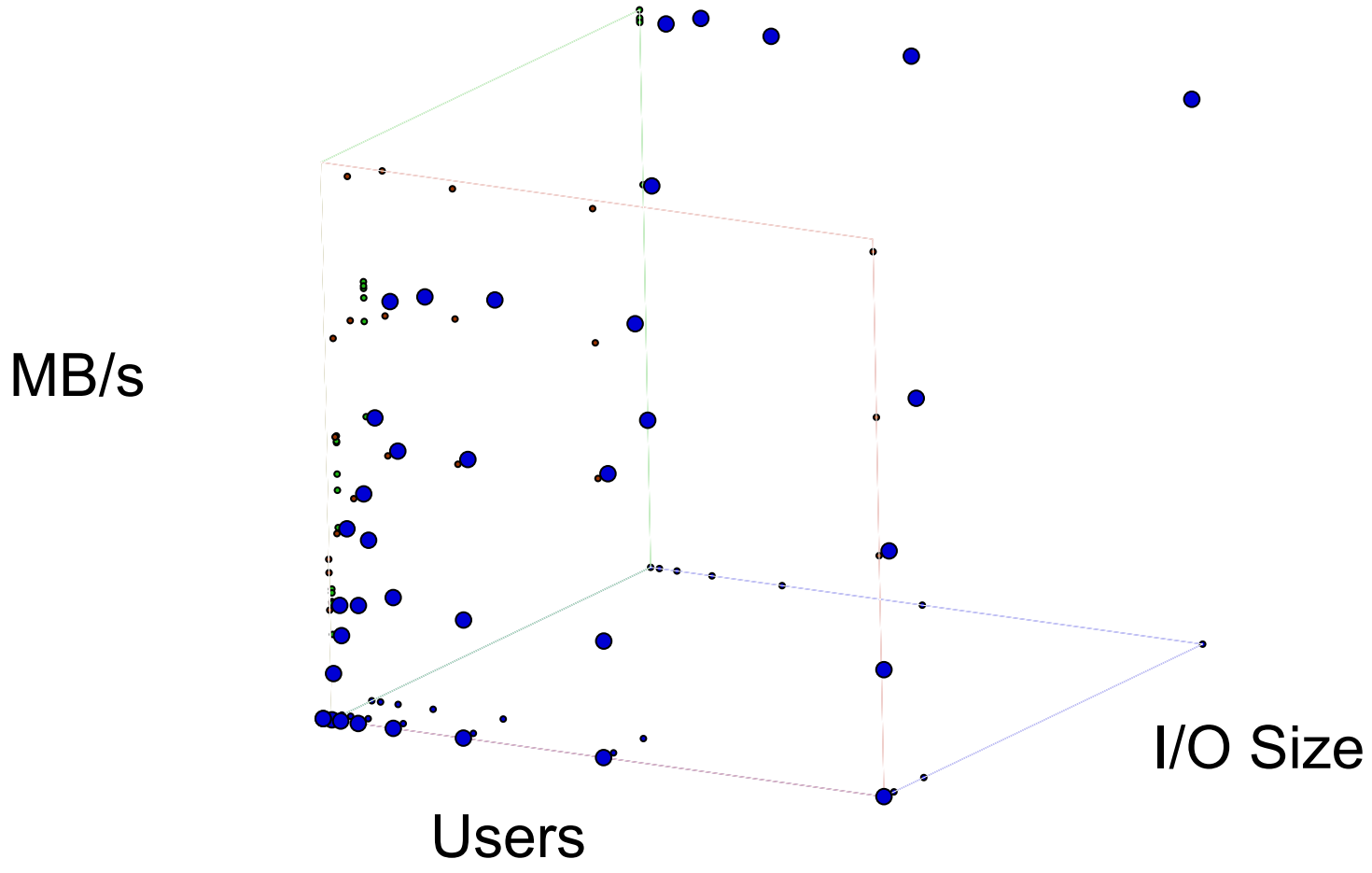
This is not a 3d plot:





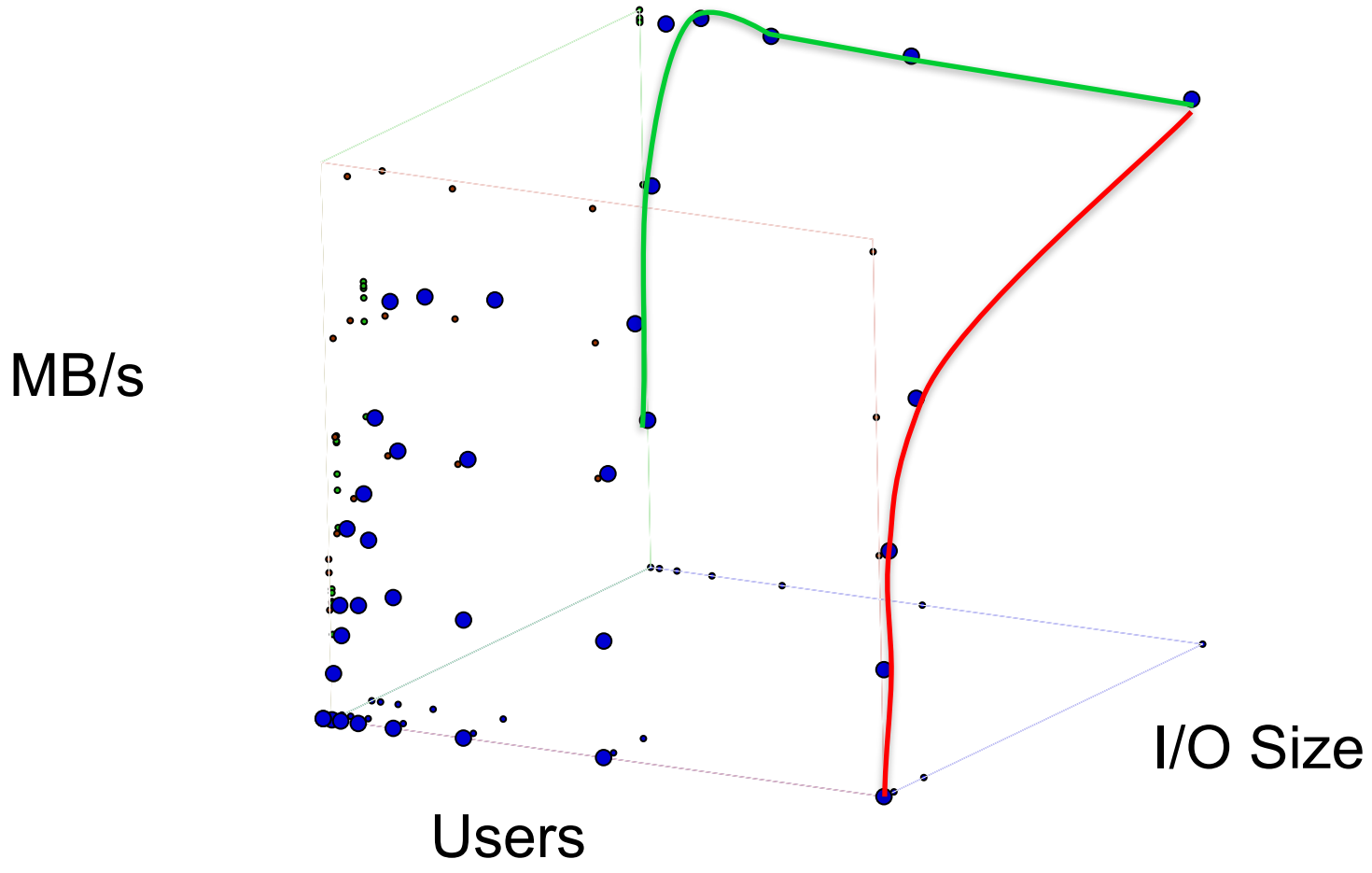
http://www.kylehailey.com/excel-doesnt-have-3d-charts-but-r-does/

MB/s vs IO_Size vs Users: Excel Plugin



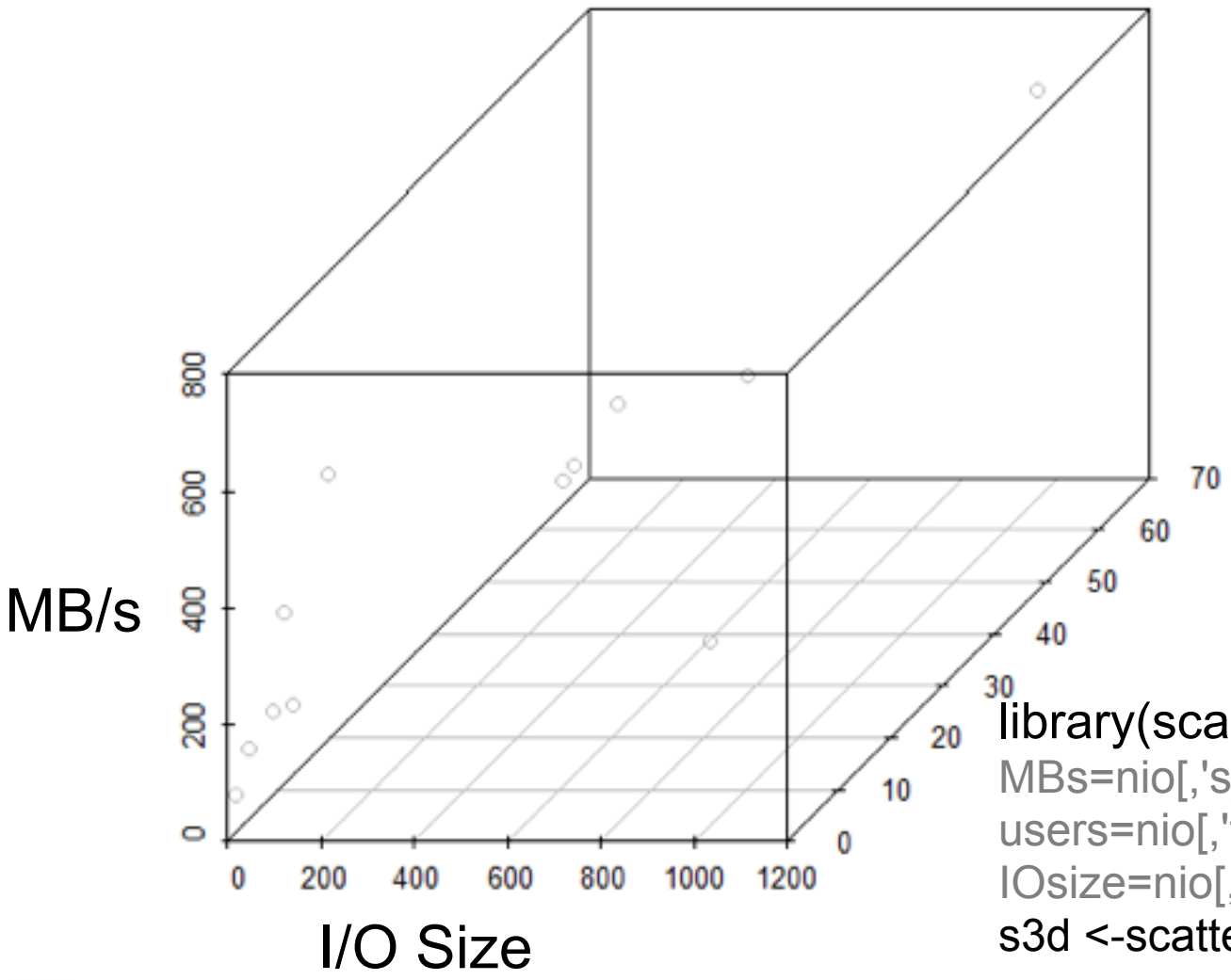
3d scatterplot : unreadable?

MB/s vs IO_Size vs Users: Excel Plugin



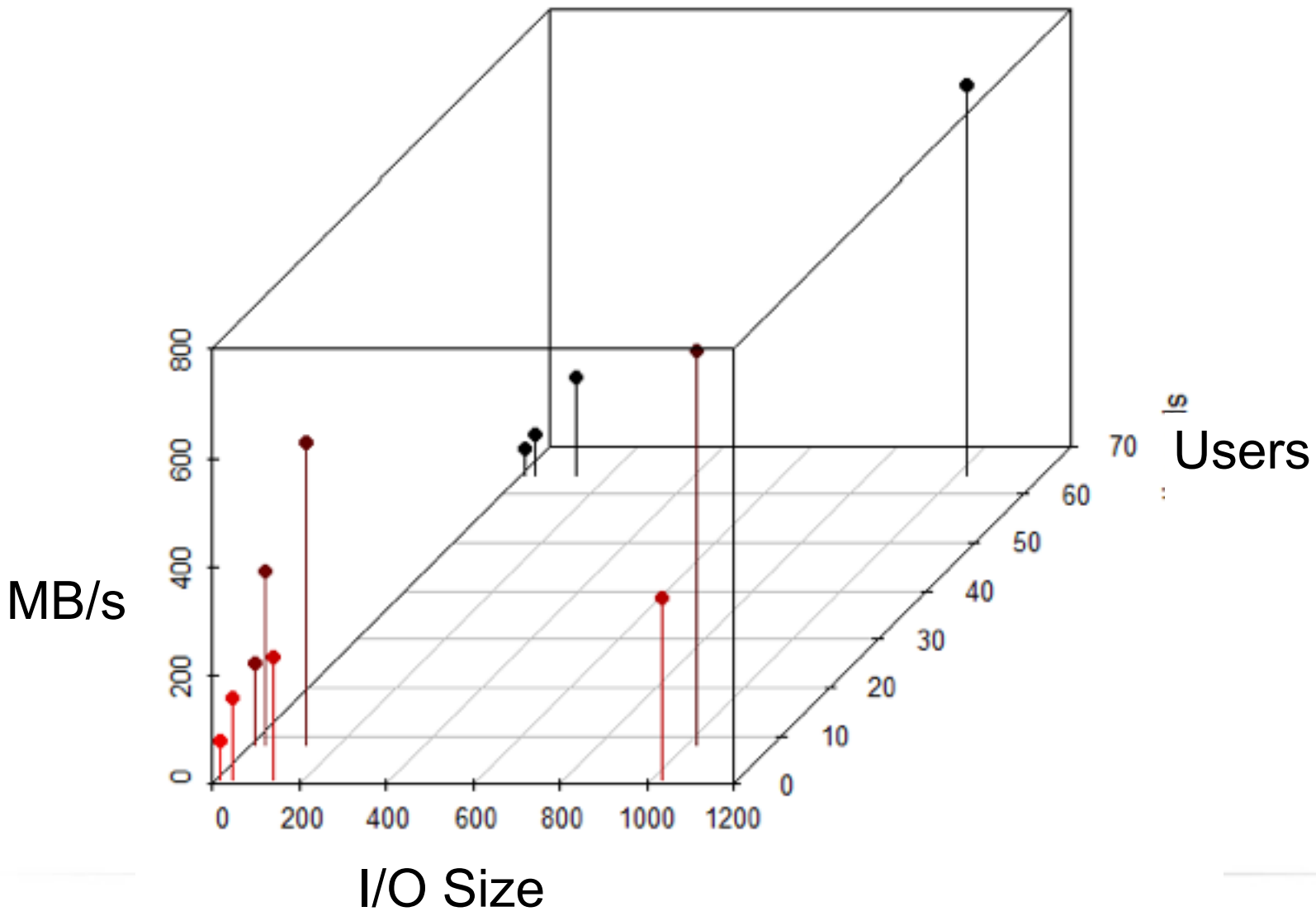
3d scatterplot : unreadable?

3d with R : MBs vs IO size vs users

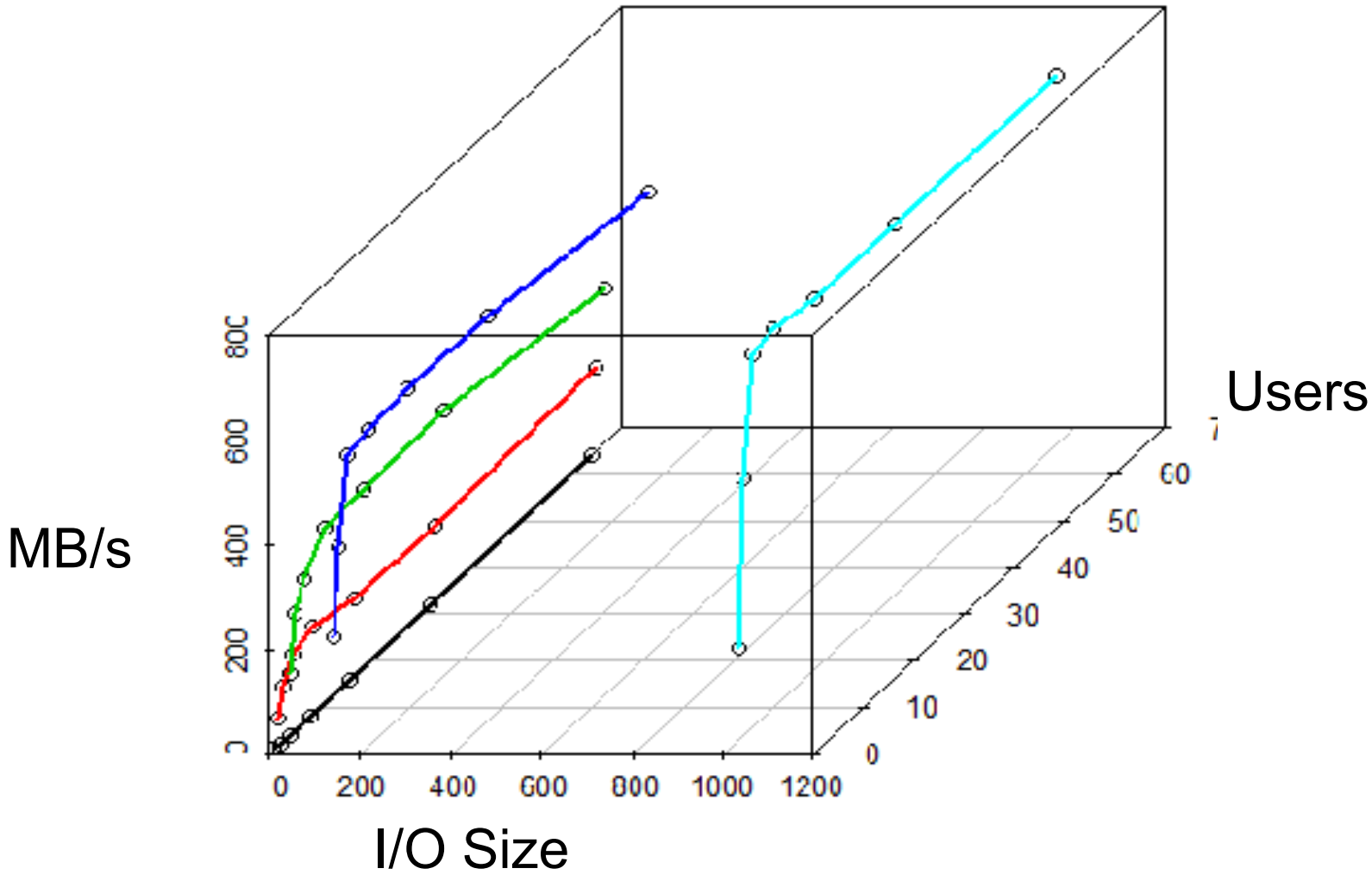



```
library(scatterplot3d)
MBs=nio[,'s_MB.s']
users=nio[,'thrds']
IOsize=nio[,'s_KB']
s3d <-scatterplot3d(IOsize,users,MBs)
```

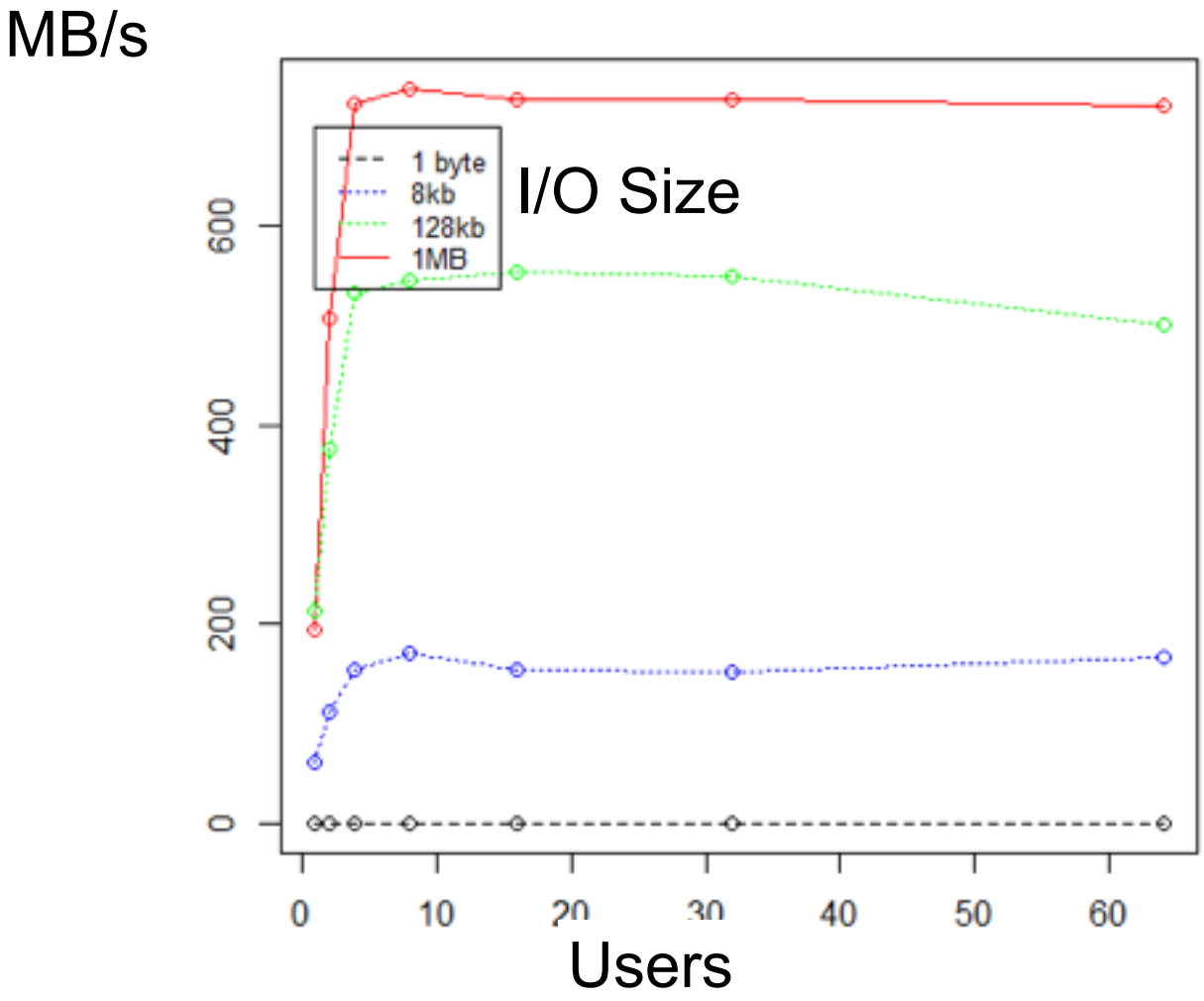
3d with R: drop pins



3d with R : add more data

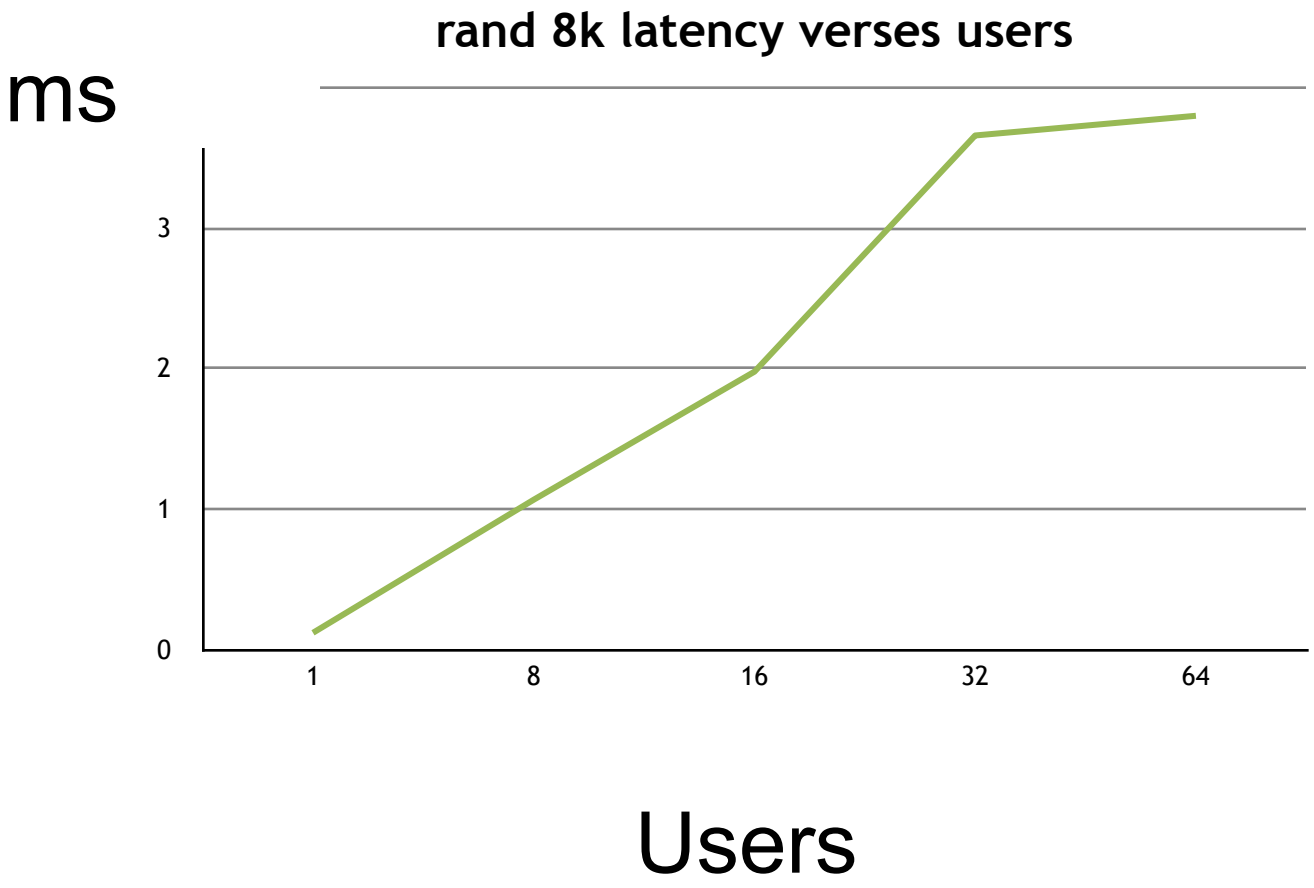


2d with R : use lines for 3rd dimension

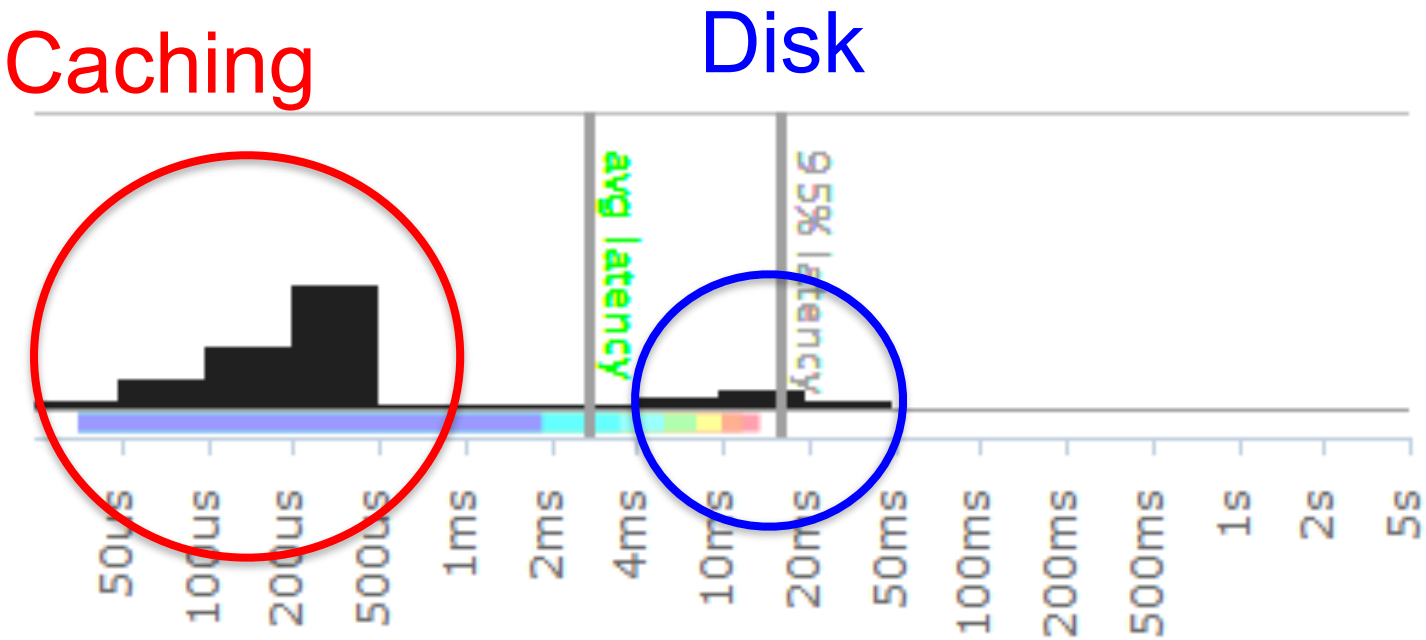


Summary:
3d graphs unnecessary

Latency vs Users

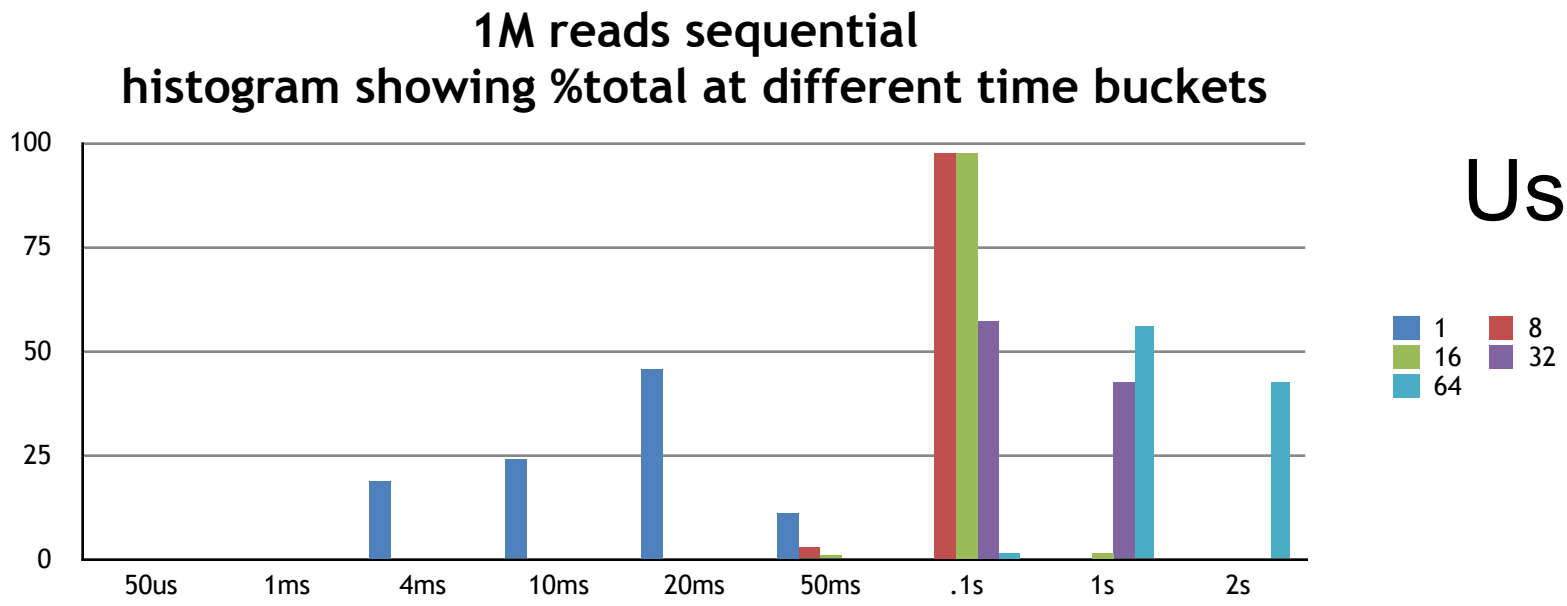


Average Latency hides information



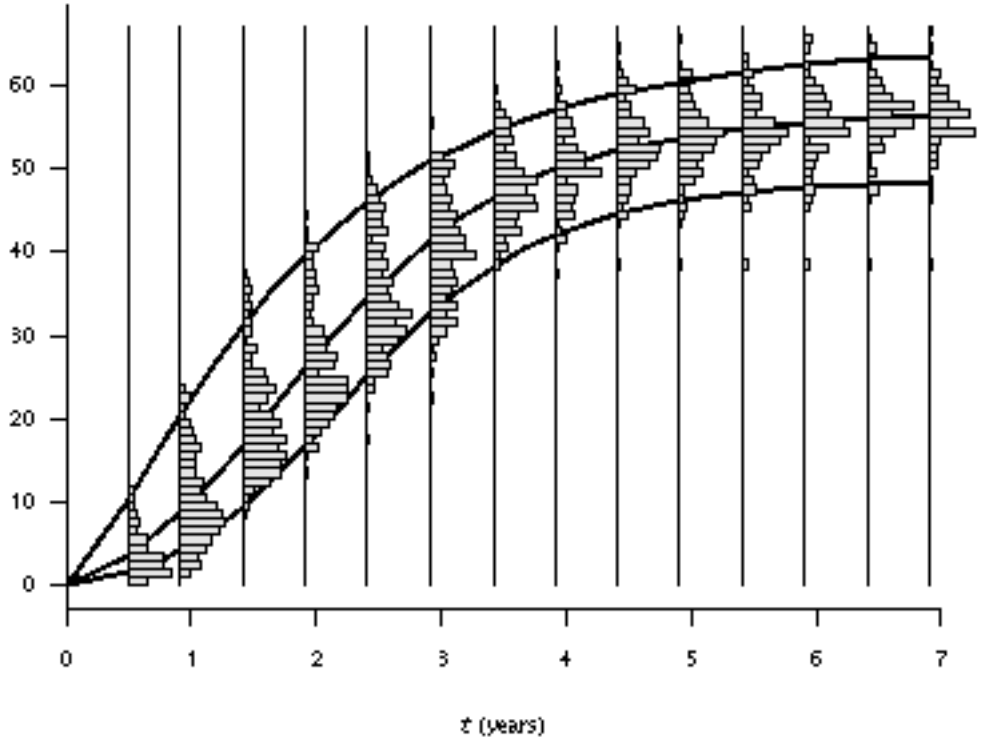
Shows hidden details
But only one user load

Showing Multiple loads

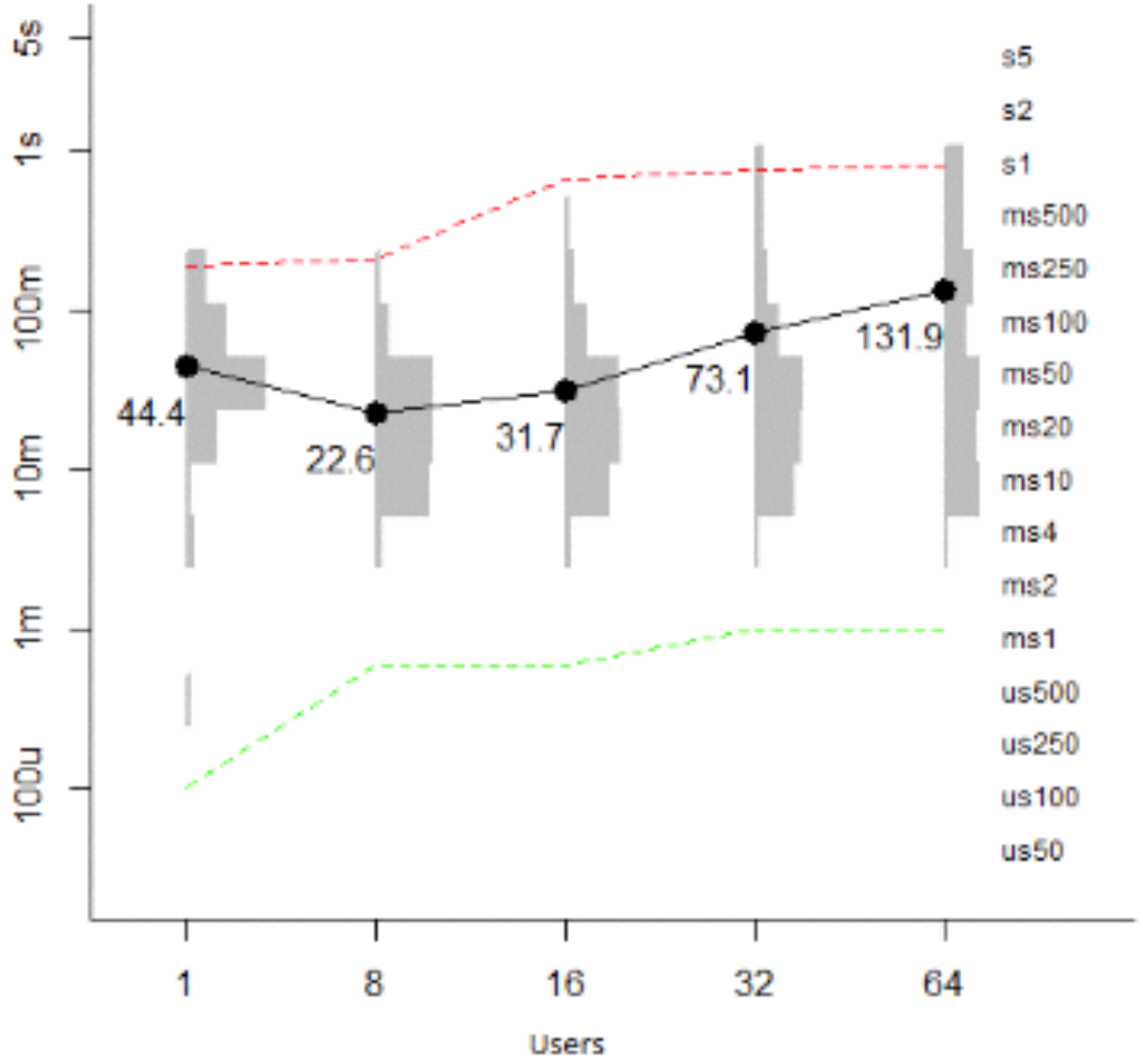


Difficult to read

Showing Histograms for multiple loads

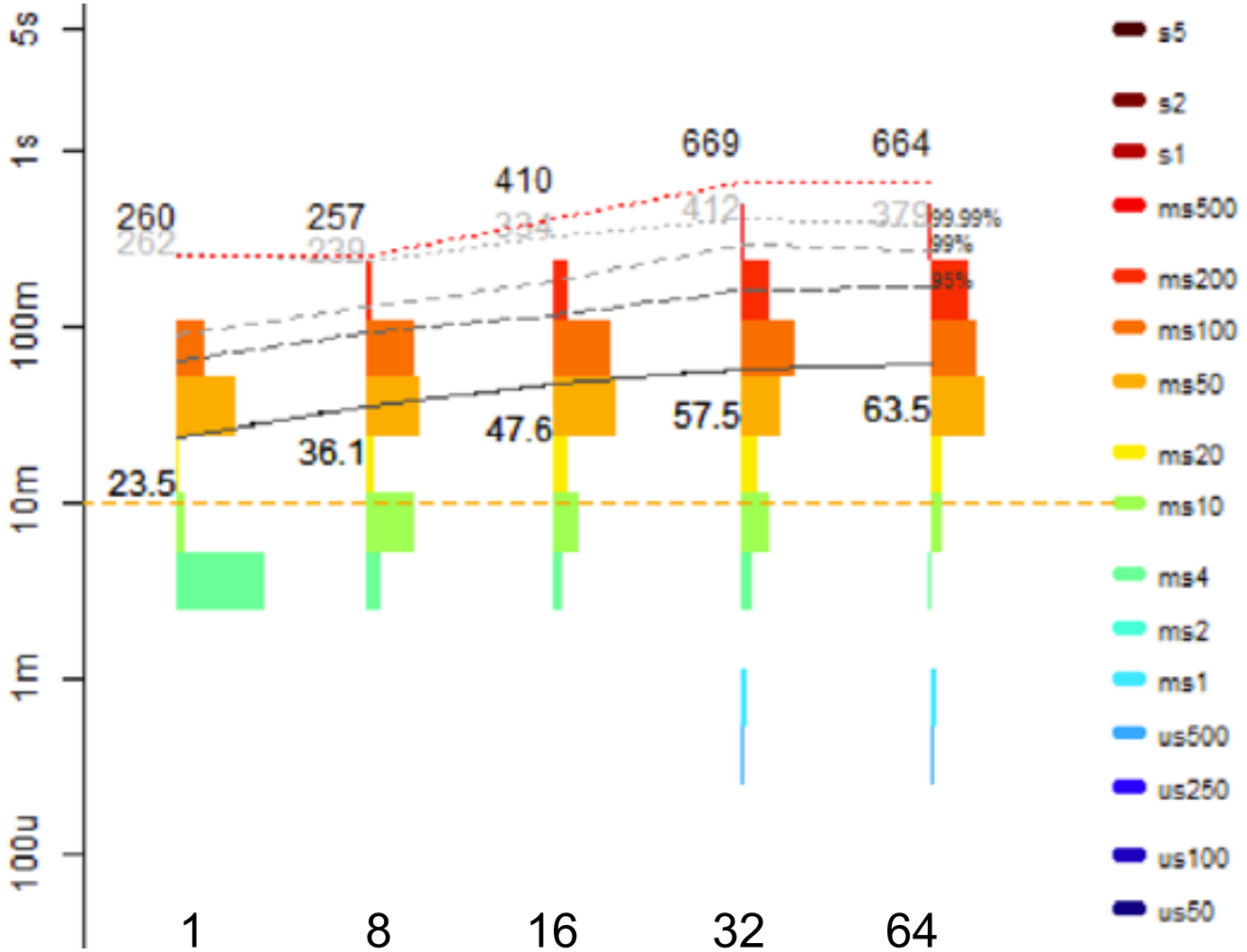


Histograms for multiple tests



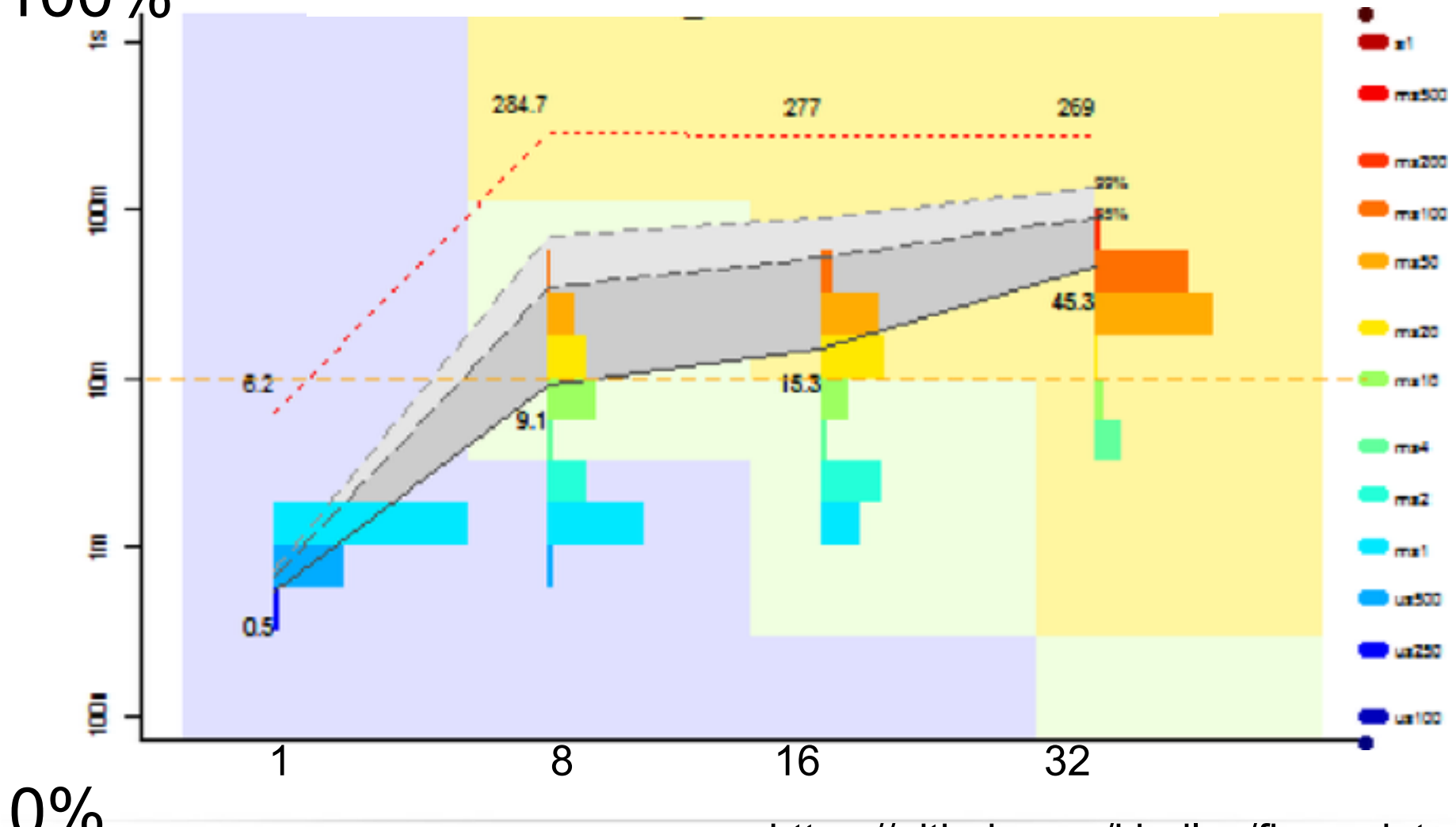
https://github.com/khailey/fio_scripts

Color Code Histograms



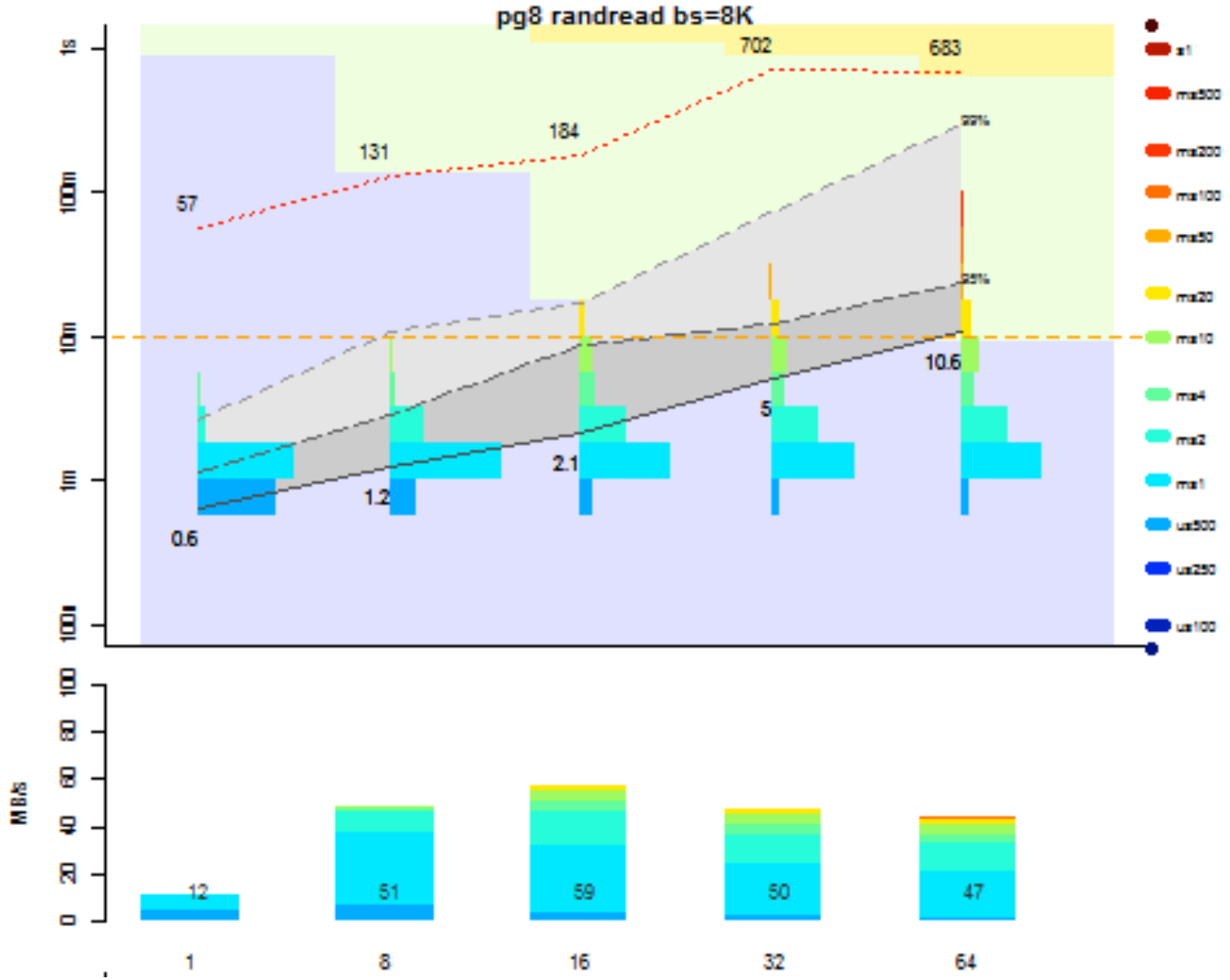
https://github.com/khailey/fio_scripts

Background Coloring



https://github.com/khailey/fio_scripts

Throughput



https://github.com/khailey/fio_scripts

I/O report card for Oracle databases

Define for tests

- Types of I/O
- Sizes of I/O
- Number of outstanding I/O (# of users or threads)

Pick Tool

- fio – flexible I/O benchmark
  - Active user community
  - Quite flexible

Note: Why don't I use Orion?

- Problems
- NFS support
- Installation

DELPHIX

What kind of I/O does a database do?

1. User Queries
   - Small random reads (of data blocks 8K usually)
     - db file sequential read
   - Large Sequential reads
     - Db file scattered reads
2. LGWR
   - Small sequential writes normally
   - Larger sequential writes possible
3. DBWR
   - Small random writes
   - Users don't wait on DBWR so who cares

Load

- Tests
  - Random reads (small)
  - Sequential reads (large)
  - Sequential writes (small and larger)
- What load?
  - 1 , 8 , 16, 32, 64 users ?
- Sizes
  - Random 8K , one typical database block (could be 2,4,8,16,32)
  - Sequential 1M (just take largest size)
  - Writes 1K , 8K, 128K

	users	sizes
Random read	***1,8,16,32,64***	***8k***
Sequential read	1,8,16,32,64	1M
Sequential read	***1***	***8k,32k,128K***
Sequential write	1,4,16	1k,8k,128k

Tool : fio

```
$ cat config_file

   filesize=12800m
   filename=/dev/rdsk/c1t1d0p0:/dev/rdsk/c1t2d0p0
   runtime=120
   rw=read
   bs=0008k
   numjobs=1

$ fio config_file
```

http://www.kylehailey.com/io-benchmarking-tools/

Fio output

```
(g=0): rw=read, bs=8K-8K/8K-8K, ioengine=libaio, iodepth=1
fio 1.50
Starting 1 process
Jobs: 1 (f=1): [R] [100.0% done] [8094K/0K /s] [988 /0  iops] [eta 00m:00s]
read_8k_200MB: (groupid=0, jobs=1): err= 0: pid=27041
  read : io=204800KB, bw=12397KB/s, iops=1549 , runt= 16520msec
    slat (usec): min=14 , max=2324 , avg=20.09, stdev=15.57
    clat (usec): min=62 , max=10202 , avg=620.90, stdev=246.24
     lat (usec): min=203 , max=10221 , avg=641.43, stdev=246.75
    bw (KB/s) : min= 7680, max=14000, per=100.08%, avg=12407.27, stdev=1770.39
  cpu          : usr=0.69%, sys=2.62%, ctx=26443, majf=0, minf=26
  IO depths    : 1=100.0%, 2=0.0%, 4=0.0%, 8=0.0%, 16=0.0%, 32=0.0%, >=64=0.0%
     submit    : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     complete  : 0=0.0%, 4=100.0%, 8=0.0%, 16=0.0%, 32=0.0%, 64=0.0%, >=64=0.0%
     issued r/w/d: total=25600/0/0, short=0/0/0
     lat (usec): 100=0.01%, 250=2.11%, 500=20.13%, 750=67.00%, 1000=3.29%
     lat (msec): 2=7.21%, 4=0.23%, 10=0.02%, 20=0.01%

Run status group 0 (all jobs):
   READ: io=204800KB, aggrb=12397KB/s, minb=12694KB/s, maxb=12694KB/s, mint=16520msec, maxt=16520ms
```

Fio Scripts Github

Making it easier

- fio.sh - run a set of benchmarks
- fioparse.sh - parse the output files
- fio.r - create a function called graphit() in R
- fiog.r - run graphit on output of fioparse.sh

	users	sizes
Random read	***1,8,16,3***	***8k***
Sequential read	1,8,16,32	1M
Sequential read	***1***	***8k,32k,128K***
Sequential write	1,4,16	1k,8k,128k

https://github.com/khailey/fio_scripts/blob/master/README.md

Fio.sh

-b binary name of fio binary, defaults to ./fio
-w directory work directory where fio creates a fio and reads and writes, default /domain0/fiotest
-o directory output directory, where to put output files, defaults to ./
-t tests tests to run, defaults to all, options are
readrand - IOPS test : 8k by 1,8,16,32 users
read - MB/s test : 1M by 1,8,16,32 users & 8k,32k,128k,1m by 1 user
write - redo test, ie sync seq writes : 1k, 4k, 8k, 128k, 1024k by 1 user
randrw - workload test: 8k read write by 1,8,16,32 users
-s seconds seconds to run each test for, default 60
-m megabytes megabytes for the test I/O file to be used, default 65536 (ie 64G)
-i individual file per process, default size 100m (otherwise uses the -m size)
-f force run, ie don't ask for confirmation on options
-c force creation of work file otherwise if it exists we use it as is
-u #users _test only use this many users
-l blocksize test only use this blocksize in KB, ie 1-1024
-e recordsize use this recordsize if/when creating the zfs file system, default 8K
-d Use DTrace on the run
-x remove work file after run
-y initialize raw devices to "-m megabytes" with writes
writes will be evenly written across multiple devices, default is 64GB
-z raw_sizes size of each raw device. If multiple, colon separate, list inorder of raw_device
-r raw_device use raw device instead of file, multi devices colon separated

Example

```
./fio.sh
        -b ./fio.opensolaris                                    # binary to use
        -s 120                                                  #  seconds to
run
        -t all                                                  # tests all
        -r /dev/rdsk/c1t1d0p0:/dev/rdsk/c1t2d0p0    # files
        -z 1046528:1046528                                      # sizes
        -m 256000                                               # MB to use
        -y                                                      # initialize
        -f                                                      # force,
don't prompt
        -o   .
```

	users	sizes
Random read	***1,8,16,3***	***8k***
Sequential read	1,8,16,32	1M
Sequential read	***1***	***8k,32k,128K***
Sequential write	1,4,16	1k,8k,128k

```
                                                        # output
```

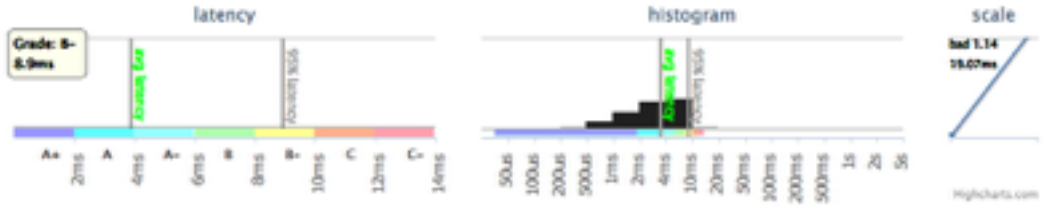
fio.sh

test	users	size		MB	ms	IOPS	50us	1ms	4ms	10ms	20ms	50ms	.1s	1s	2s	2s+
read	1	8K	r	28.299	0.271	3622		99	0	0	0					
read	1	32K	r	56.731	0.546	1815		97	1	1	0	0	0			
read	1	128K	r	78.634	1.585	629		26	68	3	1	0	0			
read	1	1M	r	91.763	10.890	91			14	61	14	8	0	0		
read	8	1M	r	50.784	156.160	50					3	25	31	38	2	
read	16	1M	r	52.895	296.290	52					2	24	23	38	11	
read	32	1M	r	55.120	551.610	55					0	13	20	34	30	
read	64	1M	r	58.072	1051.970	58						3	6	23	66	0
randread	1	8K	r	0.176	44.370	22	0	1	5	2	15	42	20	10		
randread	8	8K	r	2.763	22.558	353		0	2	27	30	30	6	1		
randread	16	8K	r	3.284	37.708	420		0	2	23	28	27	11	6		
randread	32	8K	r	3.393	73.070	434			1	20	24	25	12	15		
randread	64	8K	r	3.734	131.950	478			1	17	16	18	11	33		
write	1	1K	w	2.588	0.373	2650		98	1	0	0	0				
write	1	8K	w	26.713	0.289	3419		99	0	0	0	0				
write	1	128K	w	11.952	10.451	95		52	12	16	7	10	0	0		0
write	4	1K	w	6.684	0.581	6844		90	9	0	0	0	0			
write	4	8K	w	15.513	2.003	1985		68	18	10	1	0	0	0		
write	4	128K	w	34.005	14.647	272		0	34	13	25	22	3	0		
write	16	1K	w	7.939	1.711	8130		45	52	0	0	0	0	0	0	
write	16	8K	w	10.235	12.177	1310		5	42	27	15	5	2	0	0	
write	16	128K	w	13.212	150.080	105		0	0	3	10	55	26	0	2	

DELPHIX

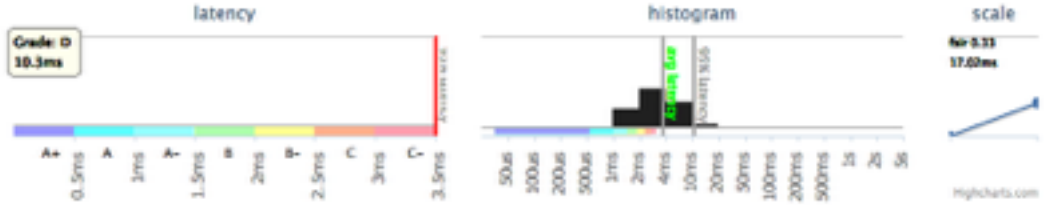
Test Configuration
(VERSION 1.20)
tests run: all
workarea type: file
remove workfile: yes
filesize: 2000 MB
test duration: 60 seconds each

Random Read 16 users , 8k I/O sizes



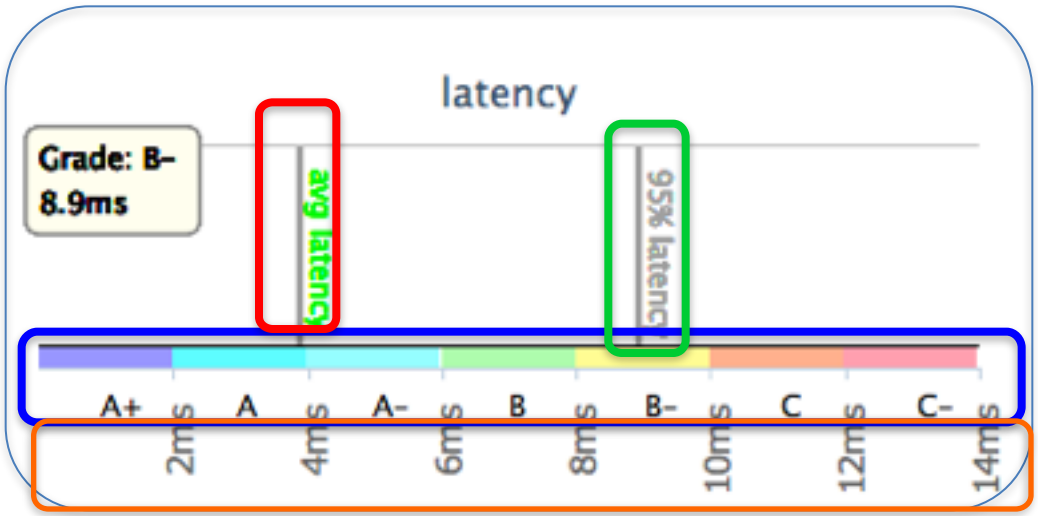
Sequential Read 1 users , 1M I/O sizes

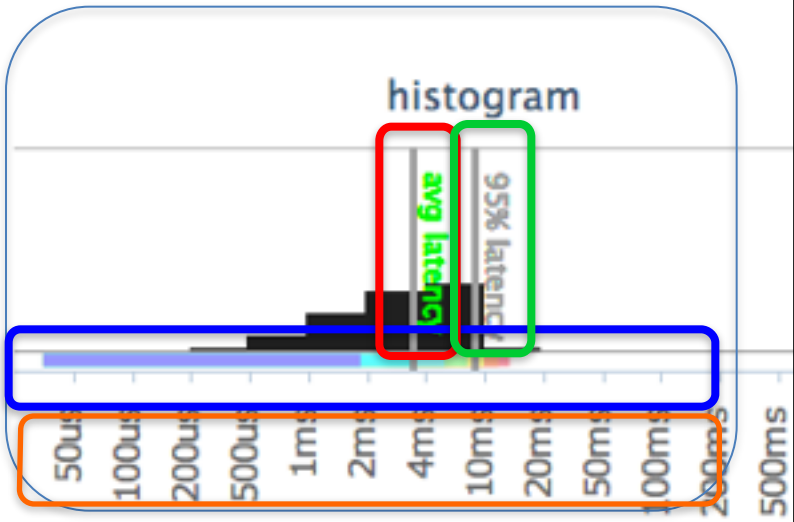
Sequential Write 4 users , 1K I/O sizes



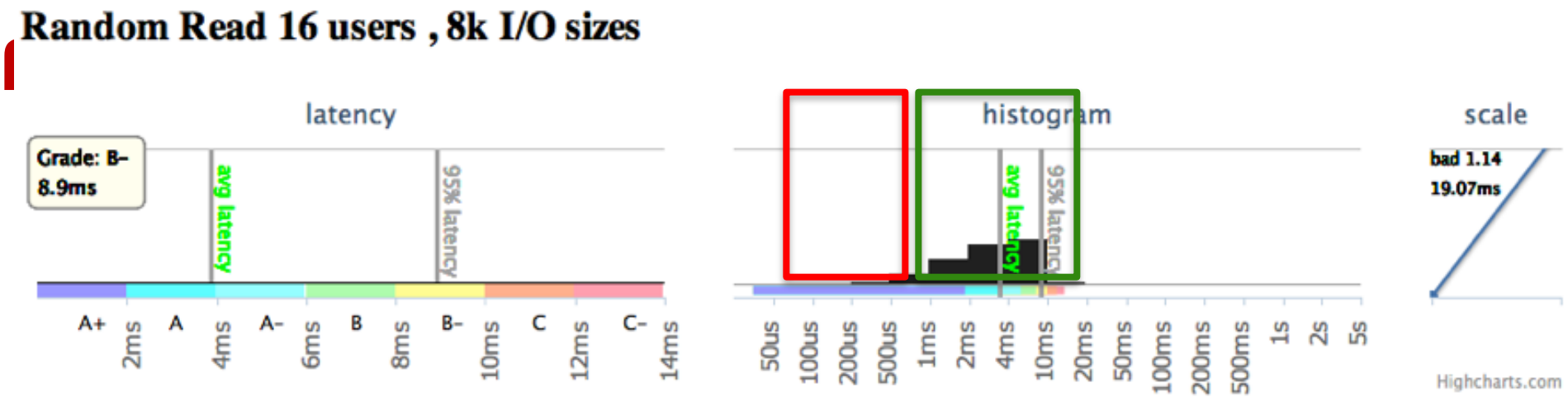
Sequential Write 4 users , 128K I/O sizes

Random Read 16 users , 8k I/O sizes

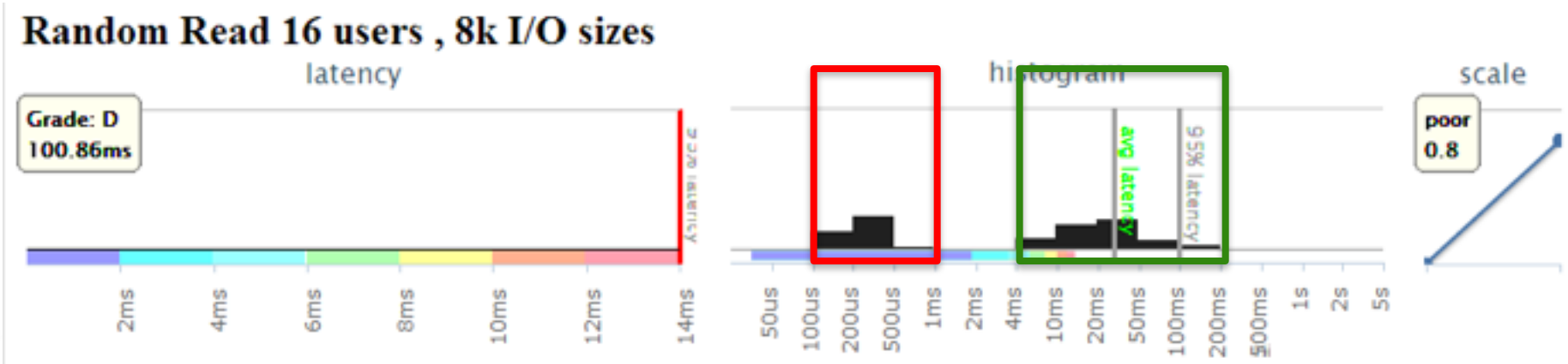




Reads from Disk

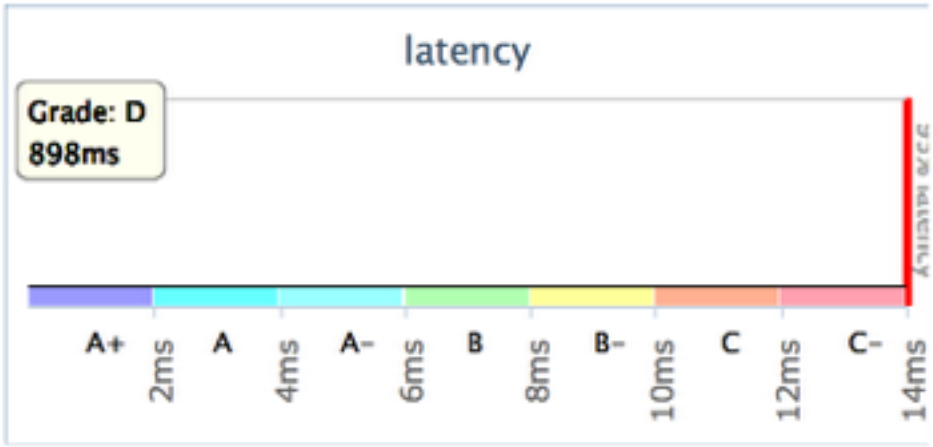


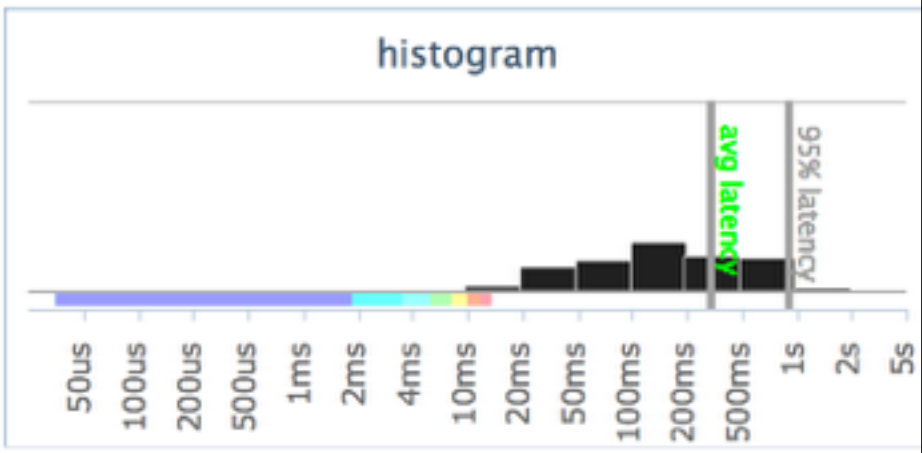
Reads hitting cache



Poor Storage

Random Read 16 users , 8k I/O sizes

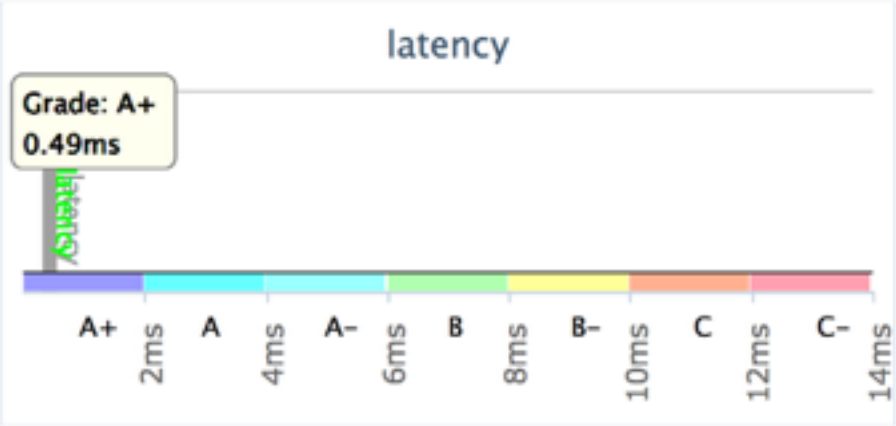


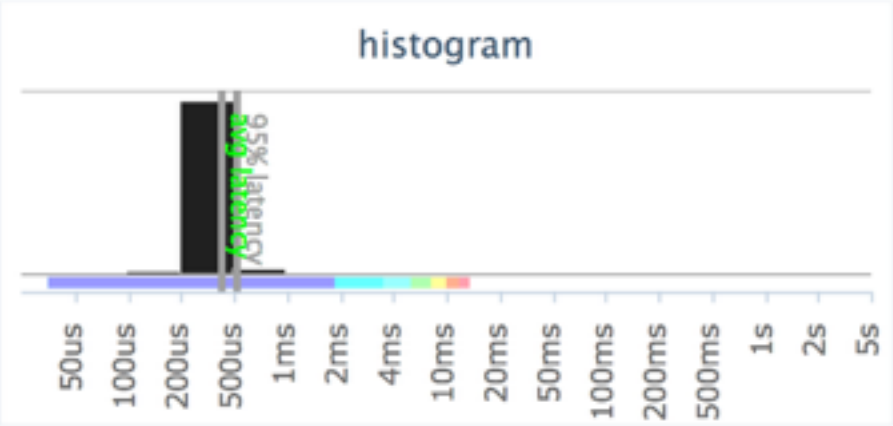


DELPHIX

Pure Storage

Random Read 16 users , 8k I/O sizes





DELPHIX

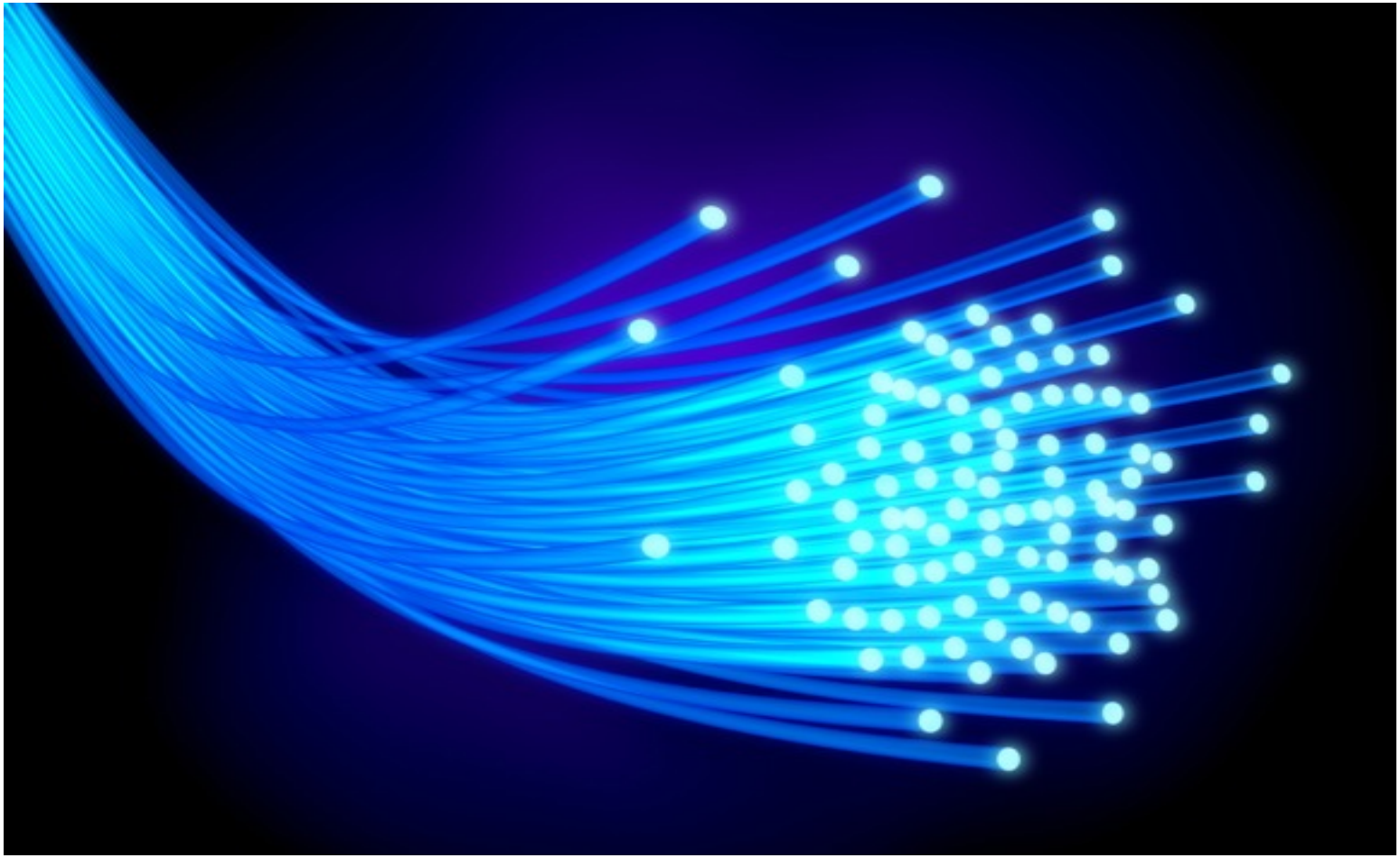
Scripts

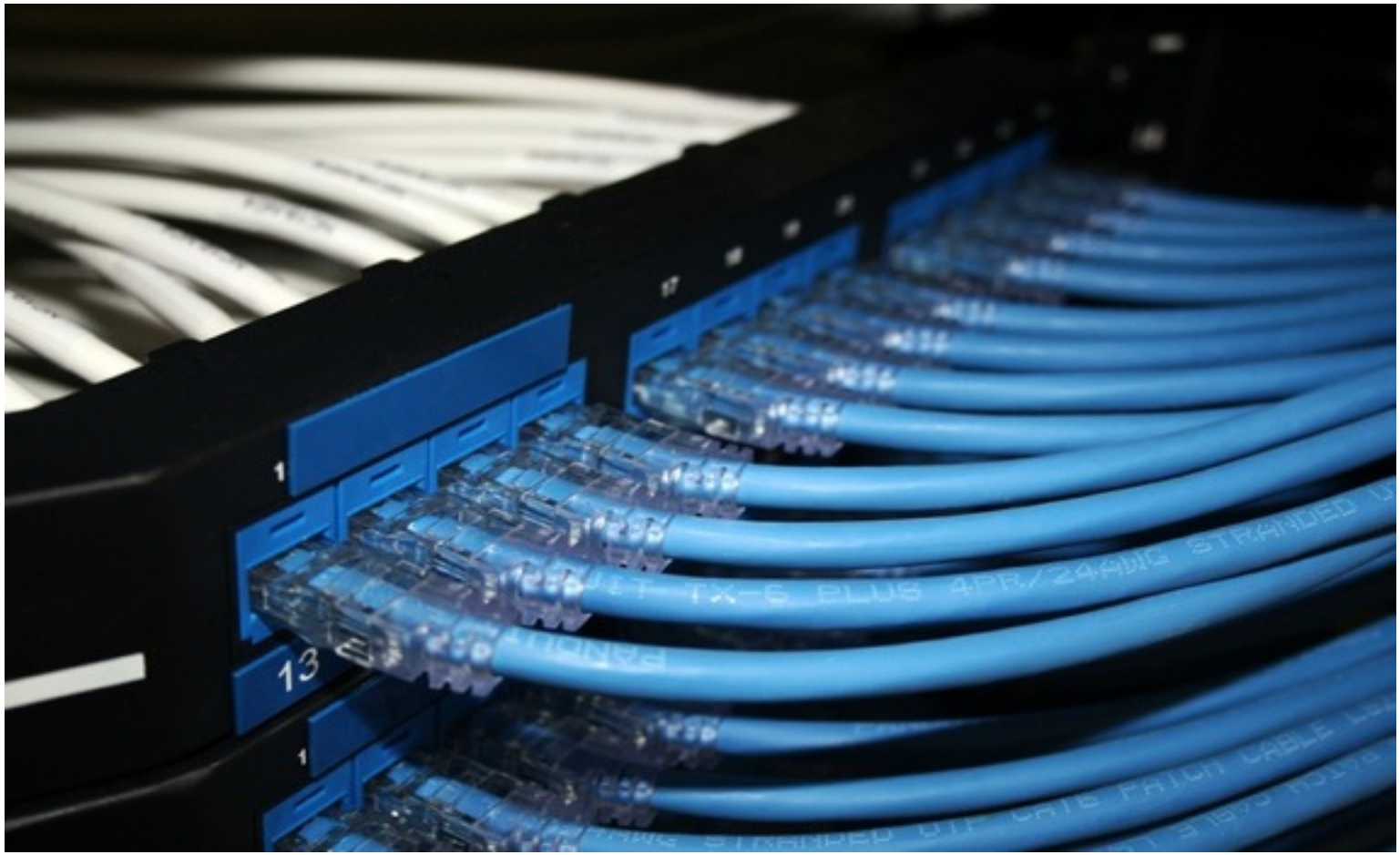
- https://github.com/khailey/fio_scripts
  - fio.sh
  - fioparse.sh
  - fio.r
  - fiog.r
- https://github.com/khailey/moats
- https://github.com/khailey/ioh

Benchmarks

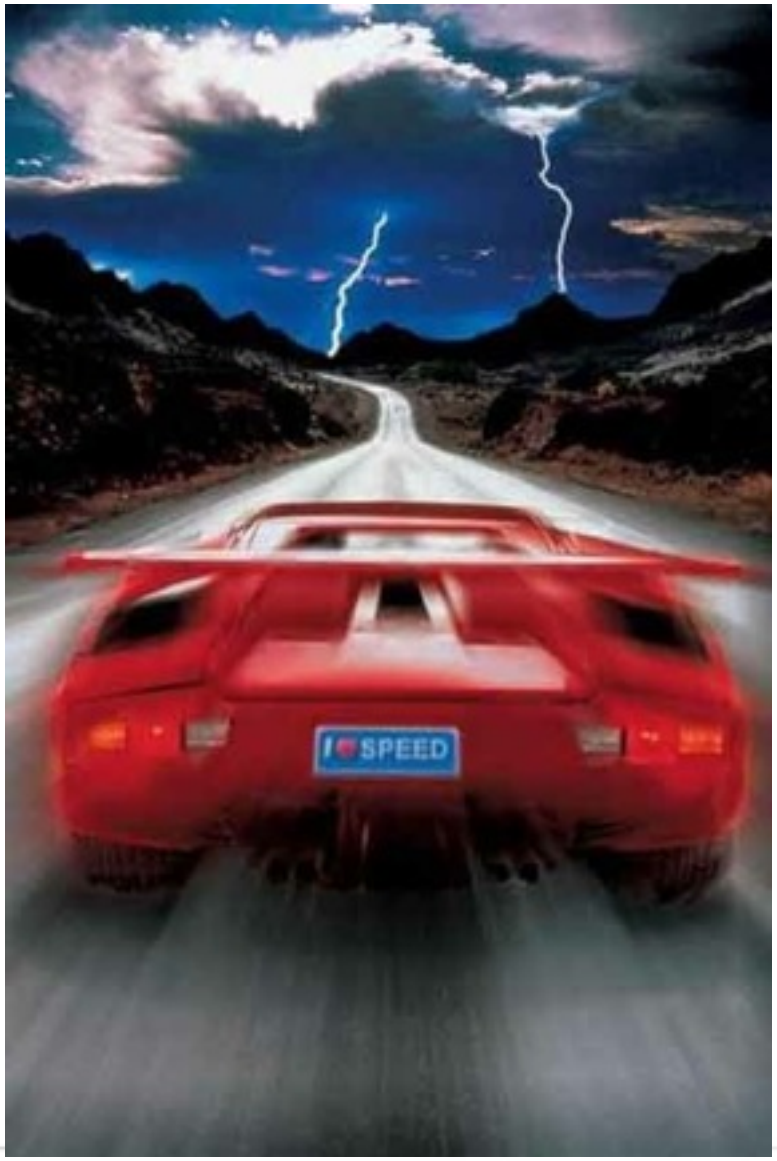
R graphs

•END




13


Manly men
only use
Fibre Channel


I SPEED

Wire 5ns/m

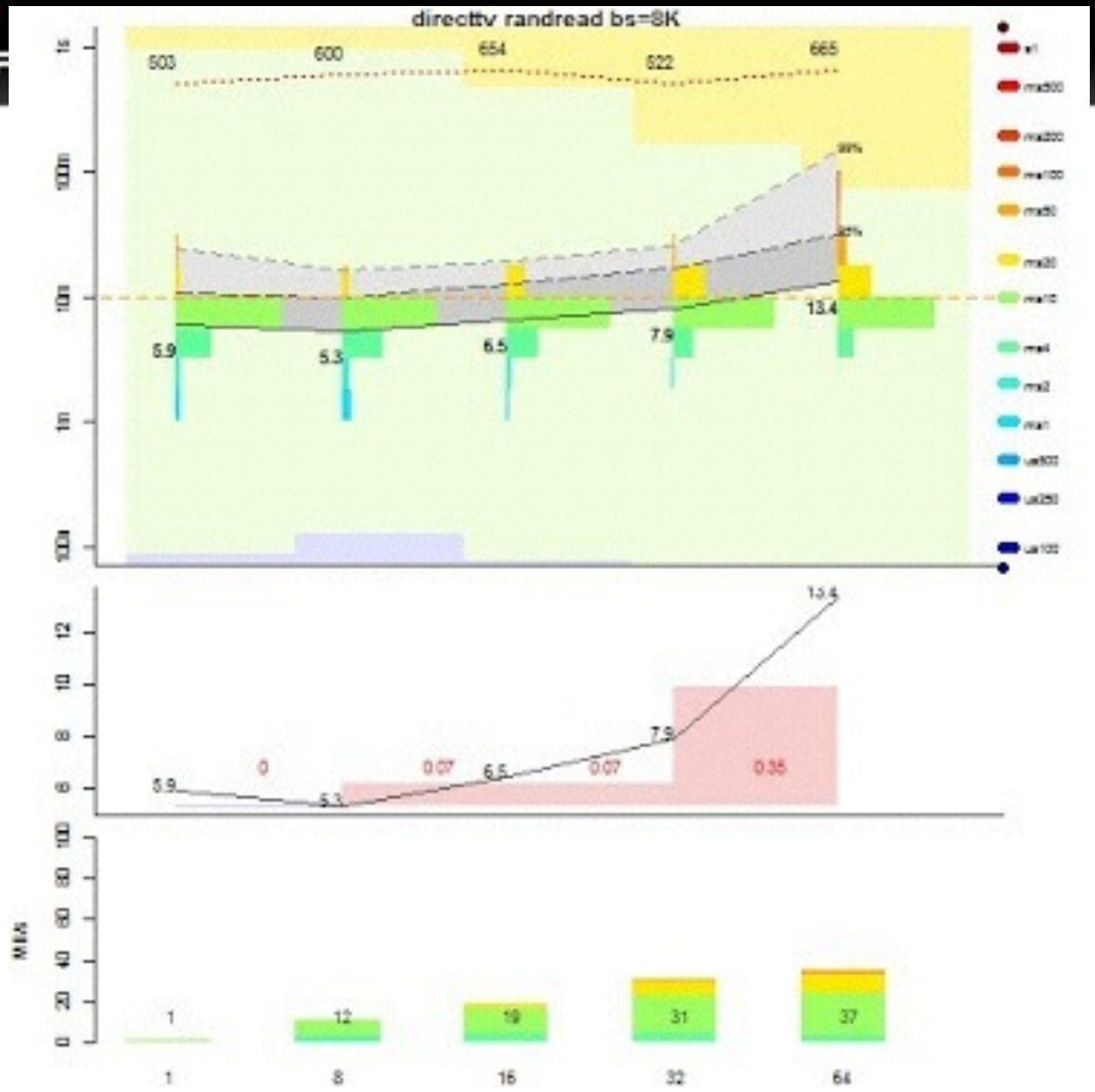
RAM

8K random disk read ~= 7 ms
small sequential write ~= 1 ms

Light = 0.3m/ns
Wire ~= 0.2m/ns
Wire ~= 5us/km

Data Center 10m = 50ns

LA to London = 30ms
LA to SF = 3ms


directtv randread bs=8K
503
600
654
622
665
5.9
5.3
6.5
7.9
13.4
0
0.07
0.07
0.35
MB/s
1
12
19
31
37
1
8
16
32
64

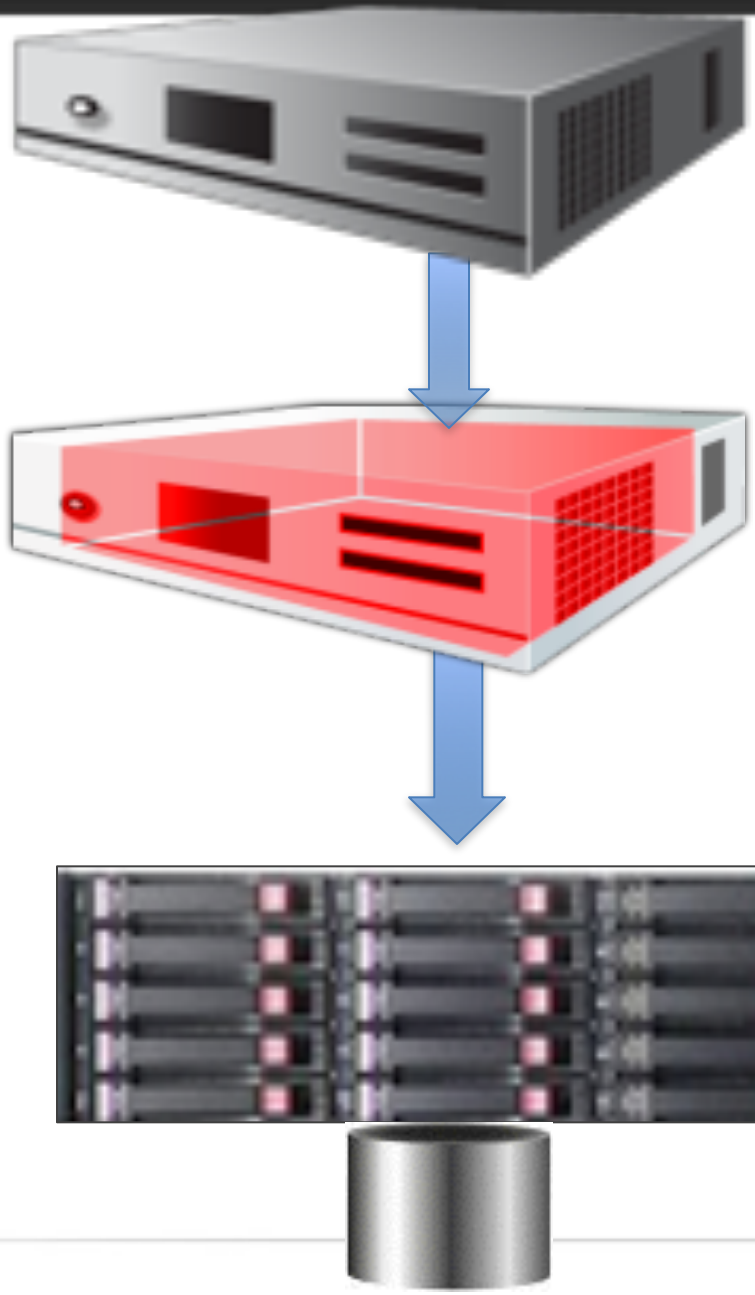
More stack more

Oracle
NFS
TCP
NIC

Network

NIC
TCP
NFS
File system
Cache/spindle

NFS
Not FC



Oracle
NFS
TCP

oram
on.sh

Network

TCP
NFS
ZFS
Cache

ioh.
sh

Fibre

Cache
spindle

VMware Performance

Summary

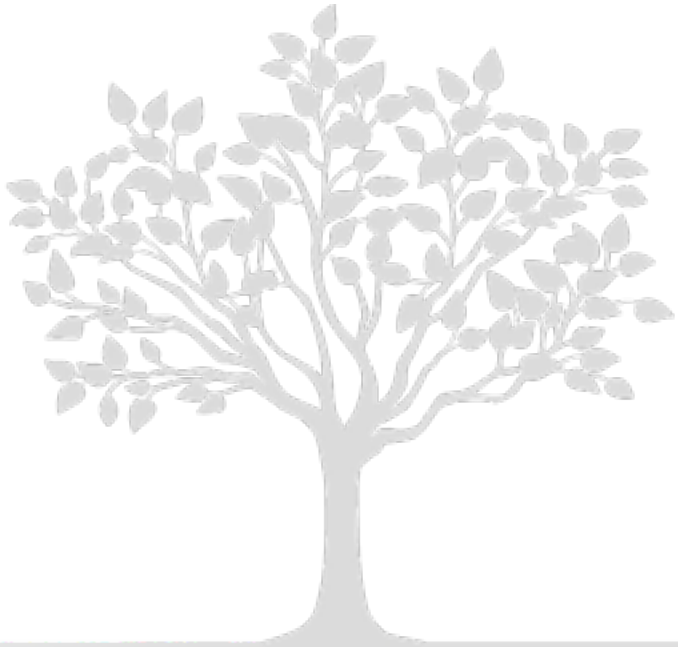
- The Good: Delphix performance within VMware is *generally* comparable to bare metal

- The Bad: Some important workloads were 33-50% slower within VMware (although there is hope…)

- The Ugly: Testing revealed substantial inadequacies in testing infrastructure

Approach and Testing Scope

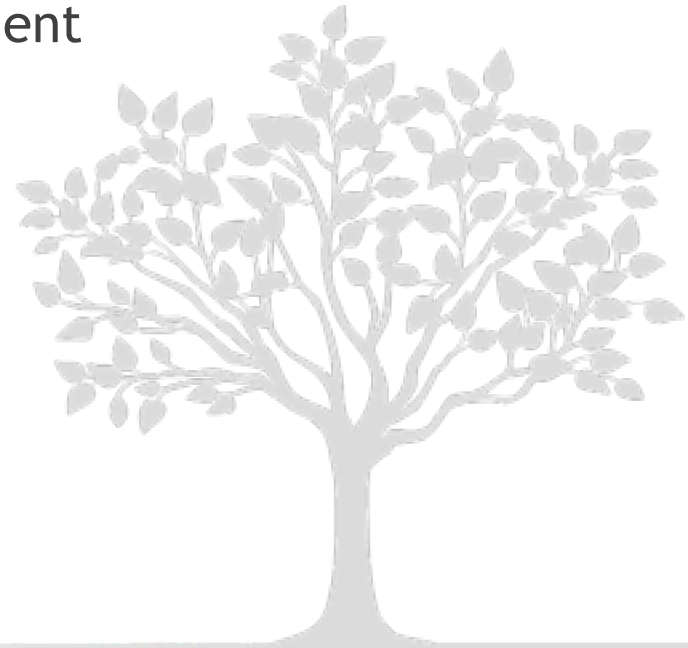
- Test NFS performance of several I/O operations across a spectrum of load levels on both bare metal and VMware
- Testing comprised :
  - 2 dedicated servers, 1 dedicated storage processor, 5 dedicated array drives
  - >10 test suite runs
  - >100 hours of manual effort
  - >200 hours of machine time
  - >3,000 tested configurations



Approach and Testing Scope

A large space was tested:

- Each of:
  - 8K random reads and sequential writes
  - 1M sequential reads and writes
- Were tested at each of: low, moderate, and high loads
- For both of synchronous and asynchronous I/O
- On each of bare metal and VMware
- On Delphix 2.5.2 and 2.6.in_development



Results: The Good

Operation	Heavy Load	Moderate Load	Light Load
Small cached		*	✗
Small disk reads			
Large cached			
Large writes			

Performance of most operations and workloads within VMware was within 10% of bare metal.

* For moderate loads within VMware small cached **asynchronous** reads were within 2% of bare metal performance, however small cached **synchronous** reads were 22% slower.

Results: The Bad

- Small cached reads at *light load levels* were:
  - Bare metal: 0.4 ms
  - VMware: 0.9 ms
    - **Over 2 times slower on VMware!**

- When VMware tools and the accelerated VMXNET3 driver was installed on the Delphix Server, the VMware times for small cached reads were in the green at all load levels
  - We need to productize the inclusion of VMware tools and the VMXNET3 driver for our VM deployments

Results: The Ugly

- Our small EMC array showed:
  - Consistent underperformance (>10 ms disk response)
  - Inconsistent testing results, for example:

Local Large Sequential Read Throughput	Heavy Load	Moderate Load	Light Load
Trial 1	*158 MB/s*	*193 MB/s*	*103 MB/s*
Trial 2	*146 MB/s*	*169 MB/s*	*195 MB/s*
Percent difference	*7.5%*	*12.4%*	*-89.3%*

?

??

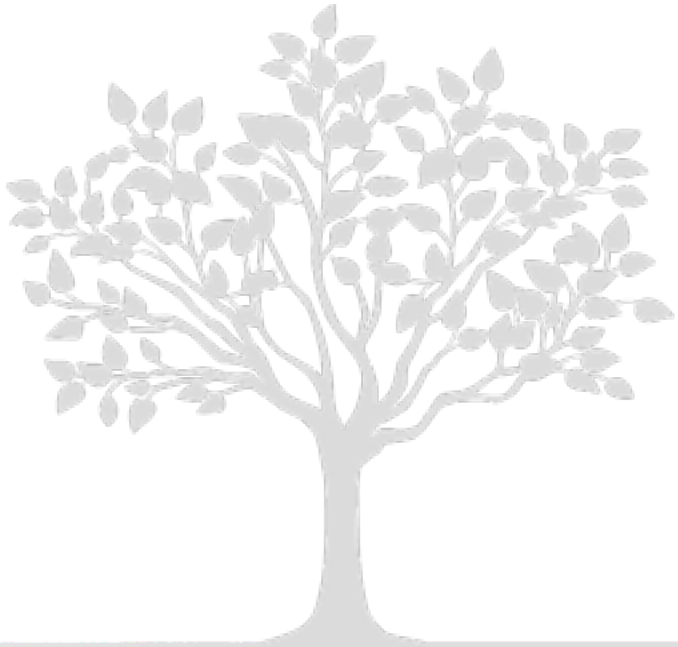
??!?

- These factors contribute to two things:
  - Data for small sequential write and large sequential disk read performance is inconclusive
  - Publishing these results would detract from our credibility

Proposed Next Steps

- Engineering:
  - Test and then enable use of VMXNET3 driver
- Performance Team:
  - Obtain access to better hardware to generate publishable results
  - Run multiple tests, determine impact of VMware on small sequential reads and large sequential writes, as well as other operations
  - Produce slides detailing VMware expectations and best practices
  - Produce customer facing whitepaper on benefits of VMware for manageability / availability, and the minor performance impacts

Thank you



Summary of Results – Delphix 2.5

Relative VM Performance to Physical Server - Asynchronous IO

Performance via NFS, Direct IO, Writes are O_SYNC	Workload		
	Heavy	Moderate	Light
Small Random Cache Read Latency in Small Random Disk Read Latency in Microseconds	97%	99%	43%
Small Sequential Write Latency in Microseconds	105%	106%	92%
Large Sequential Cache Read Throughput in kB/s	91%	94%	75%
Large Sequential Write Throughput in kB/s	100%	100%	97%
	92%	94%	95%
Heavy: 16 workers, 4 outstanding AIOs each Moderate: 4 workers, 4 outstanding AOIs Light: 1 worker, 1 outstanding AIO each			

Relative VM Performance to Physical Server - Synchronous IO

Performance via NFS, Direct IO, Writes are O_SYNC	Workload		
	Heavy	Moderate	Light
Small Random Cache Read Latency in Microseconds	99%	78%	42%
Small Random Disk Read Latency in Microseconds	92%	93%	89%
Small Sequential Write Latency in Microseconds	91%	106%	75%
Large Sequential Cache Read Throughput in kB/s	100%	100%	97%
Large Sequential Write Throughput in kB/s	92%	96%	90%
Heavy: 16 sync workers Moderate: 4 sync workers Light: 1 sync worker			

- There are two areas where performance suffers substantially on VMWare:
  - Light workloads with small random reads from the Delphix memory cache
  - Light workloads with small sequential writes, such as to a redo log file

Summary of Results – Delphix 2.6

Relative VM Performance to Physical Server - Asynchronous IO

Performance via NFS, Direct IO, Writes are O_SYNC	Relative VM Performance		
	Heavy	Moderate	Light
Small Random Cache Read Latency in Microseconds	98%	98%	44%
With Accelerated Driver	98%	99%	103%
Small Random Disk Read Latency in Microseconds	93%	92%	96%
Small Sequential Write Latency in Microseconds	70%	75%	60%
With Accelerated Driver	83%	86%	79%
Large Sequential Cache Read Throughput in kB/s	100%	100%	99%
Large Sequential Disk Read Throughput in kB/s	68%	75%	61%
Large Sequential Write Throughput in kB/s	92%	94%	95%
Heavy: 16 workers, 4 outstanding AIOs each Moderate: 4 workers, 4 outstanding AOIs each Light: 1 worker, 1 outstanding AIO each			

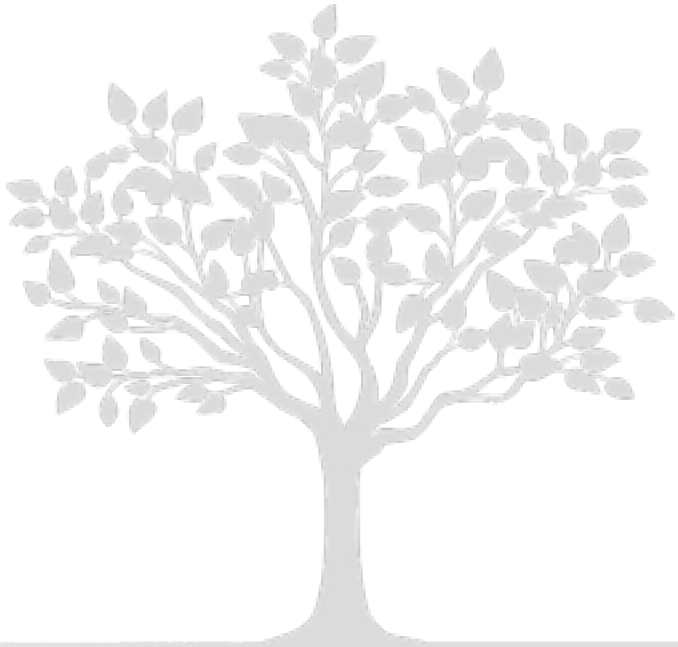
Relative VM Performance to Physical Server - Synchronous IO

Performance via NFS, Direct IO, Writes are O_SYNC	Relative VM Performance		
	Heavy	Moderate	Light
Small Random Cache Read Latency in Microseconds	99%	78%	48%
With Accelerated Driver	100%	105%	119%
Small Random Disk Read Latency in Microseconds	94%	91%	94%
Small Sequential Write Latency in	69%	74%	62%
With Accelerated Driver	86%	79%	80%
Large Sequential Cache Read Throughput in kB/s	100%	100%	99%
Large Sequential Disk Read Throughput in kB/s	63%	78%	69%
Large Sequential Write Throughput in kB/s	92%	96%	90%
Heavy: 16 sync workers Moderate: 4 sync workers Light: 1 sync worker			

- The data behind the small sequential write latency and large sequential disk read throughput is in question.

Discussion – Small Read Cache Latency

- The small random reads from cache case is important, because these reads reflect:
  - A potential performance drawback if they would have been serviced by the SAN cache in the customer's pre-Delphix configuration
  - Our opportunity to accelerate performance if they would have been serviced by SAN disk in the customer's pre-Delphix configuration



Discussion – Small Read Cache Latency

- The light workload, small random IO latency from cache exposes an unexpected interaction between DelphixOS 2.5 and VMWare – performance is poor at the lightest workload:

IO Engine	Parallelism 1	Parallelism 4
Synchronous	***980 us***	***517 us***
Asynchronous	***810 us***	***519 us***

- This asymmetry occurs only on VMWare, on physical hardware increasing parallelism leads to increasing latency
- Installing the VMware tools and the VMXNET3 accelerated driver brought the performance of parallelism 1 in line with that of bare metal under Delphix 2.6

Testing Procedure

For a 24 GB file, and then for a 512 GB file:

- Reboot the Delphix Server
- For each of 1, 4, and 16 processes, test synchronous and asynchronous IO with 1, 4, and 16 outstanding Ios
  - Warm the cache for the 24 GB file tests by reading the entire file twice
  - Do the following test sequence three times consecutively:
    - Small random reads – latency in microseconds
    - Large sequential reads – throughput in kB/s
    - Small sequential writes – latency in microseconds
    - Large sequential writes – throughput in kB/s
- The median value for each statistic is reported
- We report the figures for the 1 process, 4 asynchronous IOs – the lightest workload that does not see strange interference between VMWare and OpenSolaris at light workloads

Delphix 2.5 - Small Random Reads (8kB) – Key Metric is Latency

Configuration	Cache Hit 8kB Random Reads – us	Disk Access 8kB Random Reads – us
Bare Metal	***370***	***12,543***
VMWare – Physical RDM	520	12,912
VMWare - Independent Virtual RDM	***521***	***14,044***
VMWare – Virtual RDM	518	13,020
VMWare - Independent VMFS	***518***	***13,442***
VMWare – Snapshot VMFS	519	13,537

Delphix 2.5 - Large Sequential Reads (1MB) – Key Metric is Throughput

Configuration	Cache Hit 1MB Sequential Reads -
Bare Metal	***117,042***
VMWare – Physical RDM	116,721
VMWare - Independent Virtual RDM	***116,710***
VMWare – Virtual RDM	116,580
VMWare - Independent VMFS	***116,628***
VMWare – Snapshot VMFS	116,673

Delphix 2.5 - Small Sequential Writes (8kB) – Key Metric is Latency

Configuration	8kB Sequential Writes -
Bare Metal	***1,270***
VMWare – Physical RDM	1,361
VMWare - Independent	***1,320***
VMWare – Virtual RDM	1,265
VMWare - Independent	***1,276***
VMWare – Snapshot VMFS	1,277

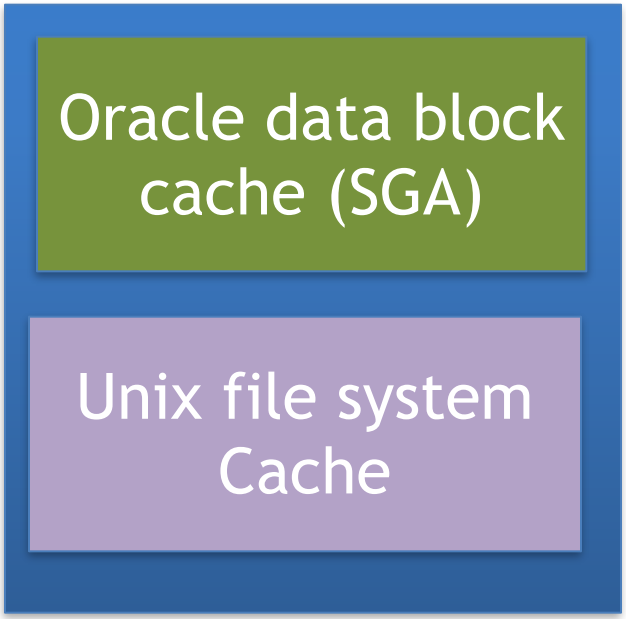
Delphix 2.5 - Large Sequential Writes (1MB) – Key Metric is Throughput

Configuration	1MB Sequential Writes - kB/s
Bare Metal	***75,959***
VMWare – Physical RDM	71,784
VMWare - Independent Virtual RDM	***62,708***
VMWare – Virtual RDM	70,380
VMWare - Independent VMFS	***67,208***
VMWare – Snapshot VMFS	72,206

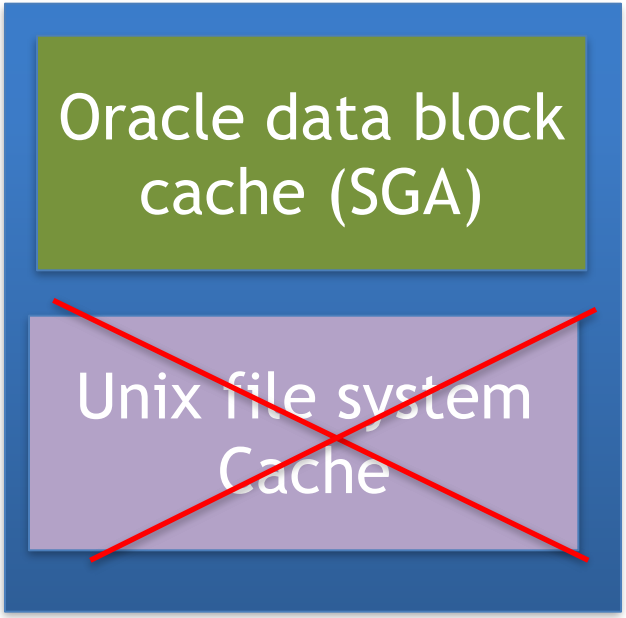
System Configuration

- Delphix Server:
  - Bare metal – HP ProLiant DL380 G7
    - Intel Xeon X5760 @ 2.93 GHz – 2 sockets, 6 cores per socket
    - 48 GB RAM
    - One 1GB Ethernet port active
  - VM Configuration Details – vSphere 4.1:
    - CPUs: 8
    - RAM: 40 GB
    - Network: One 1GB Ethernet Adapter
- SAN – EMC CX4-120
  - Dedicated storage processor
  - 5, 366.8GB Seagate C00A disks
  - 478 MB Write Cache
  - 120 MB Read Cache
- NFS Client – Linux
  - Intel Xeon X5760 @ 2.93 GHz – 2 sockets, 6 cores per socket
  - 48 GB RAM
  - One 1GB Ethernet port active

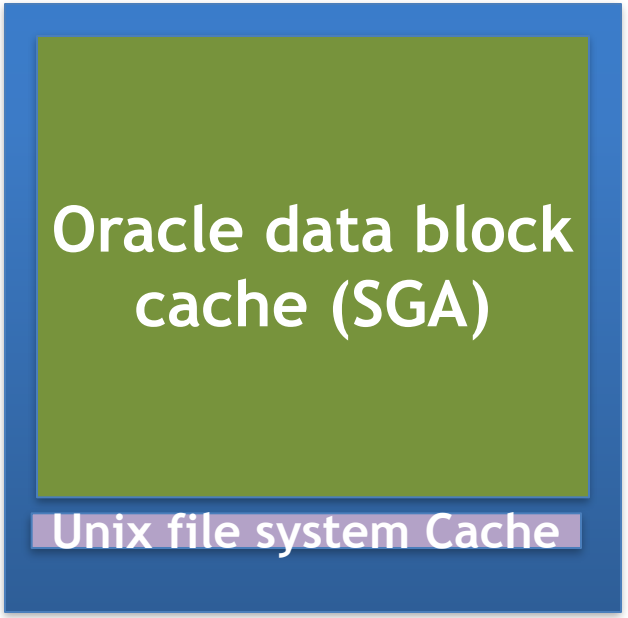
Full Table Scan Query



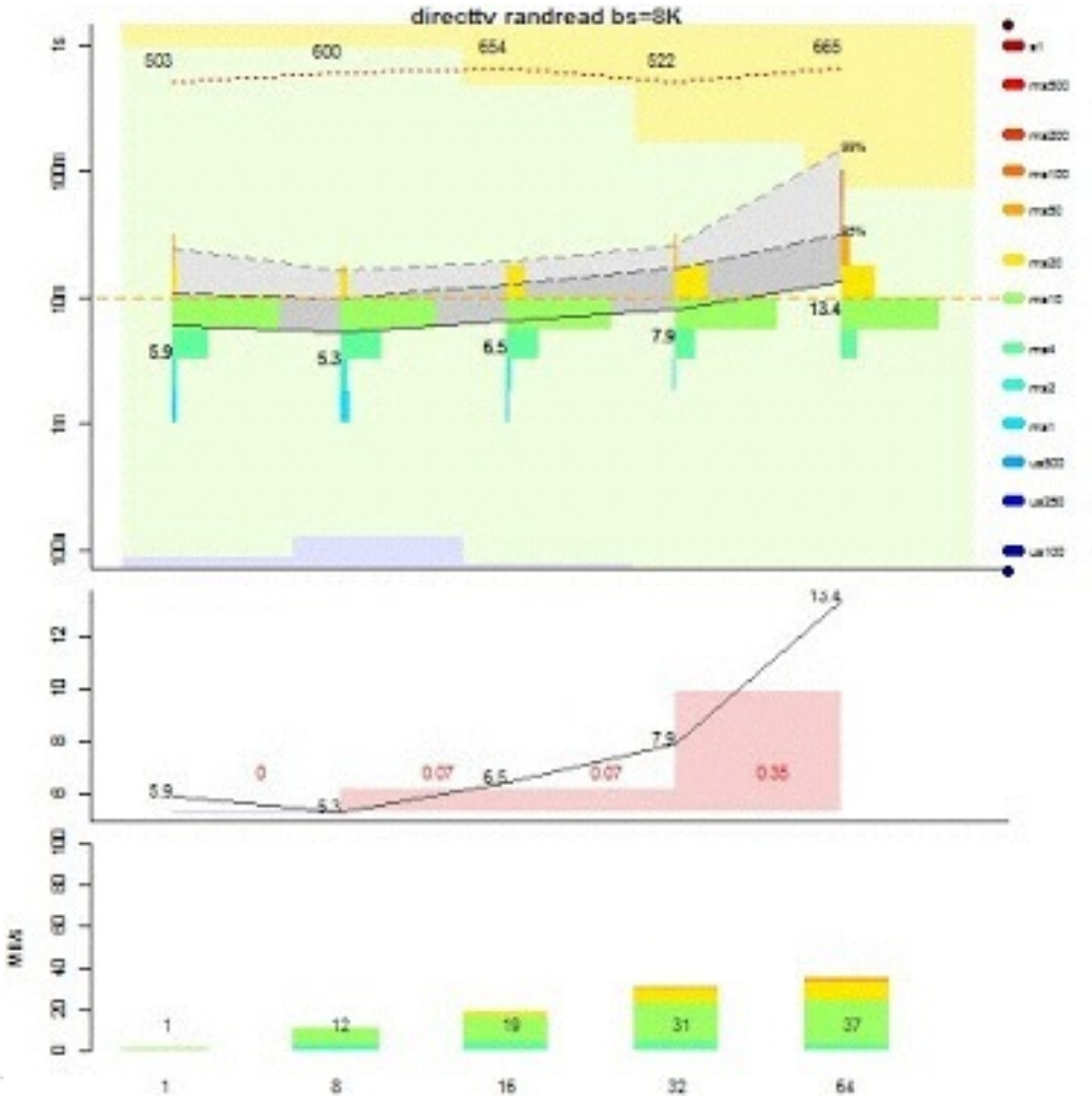
5 seconds

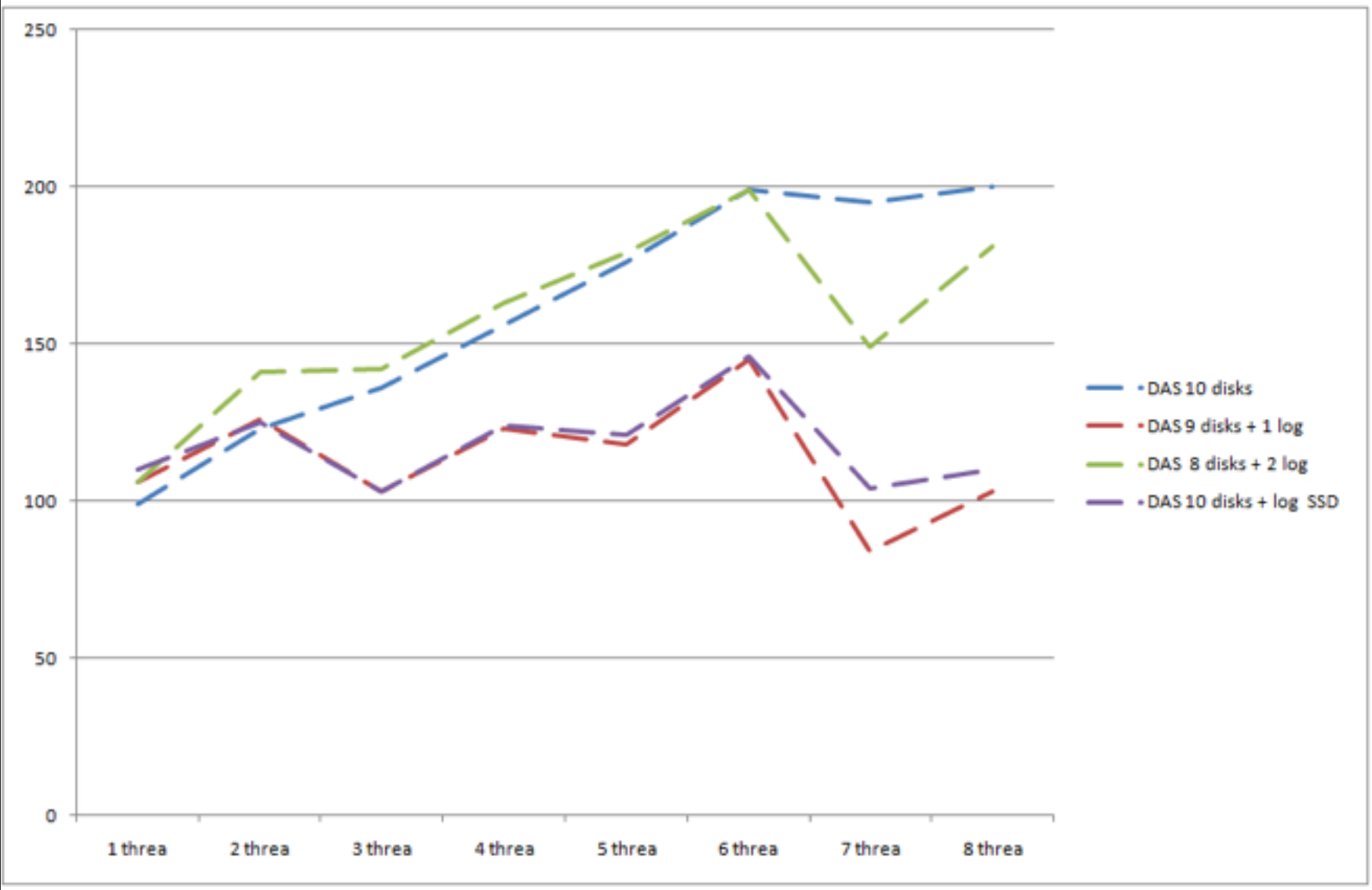


60 seconds



2 seconds


directtv randread bs=8K
503
600
654
622
665
99%
82%
13.4
7.9
6.5
5.3
5.9
0
0.07
0.07
0.35
MB/s
1
12
19
31
37
1
8
16
32
64


250
200
150
100
50
0
1 threa
2 threa
3 threa
4 threa
5 threa
6 threa
7 threa
8 threa
DAS 10 disks
DAS 9 disks + 1 log
DAS 8 disks + 2 log
DAS 10 disks + log SSD

Considerations

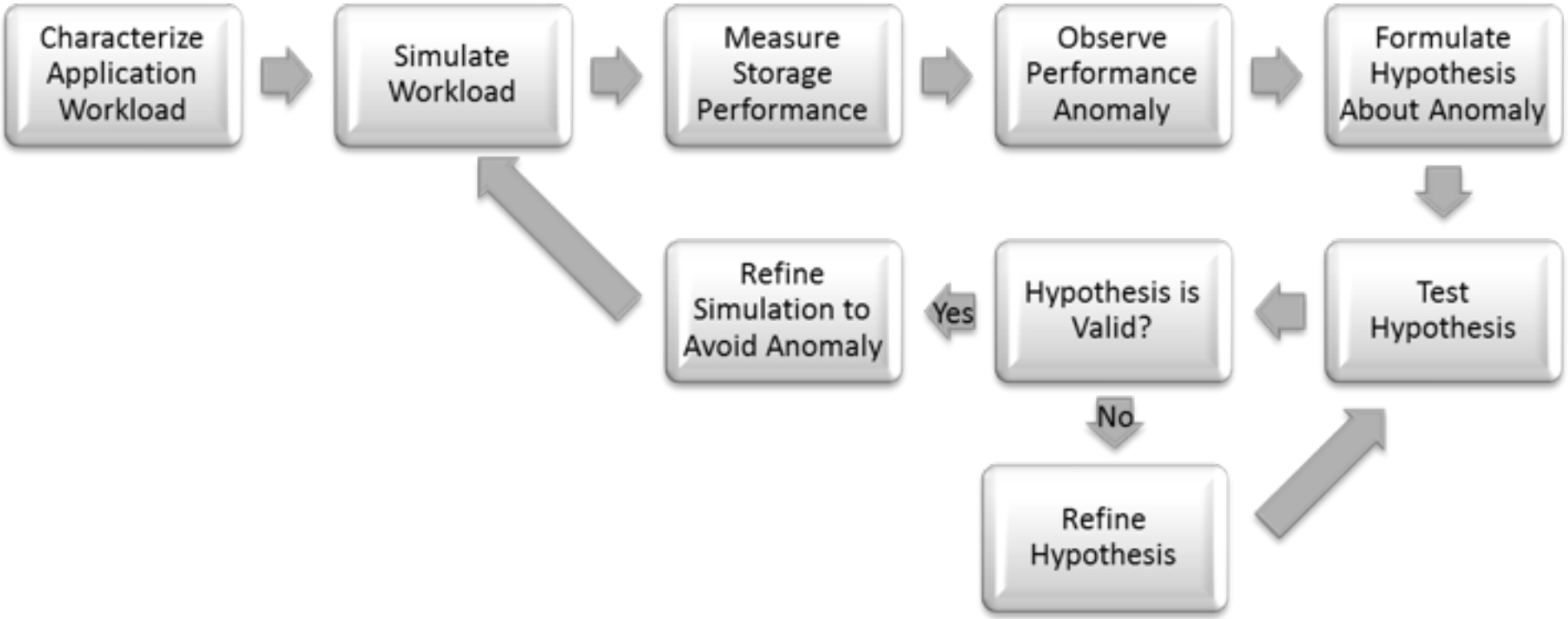
SAN

- Read Cache
- Write Cache
- Spindles
- Speeds
- Shared
- RAID levels
- Controllers

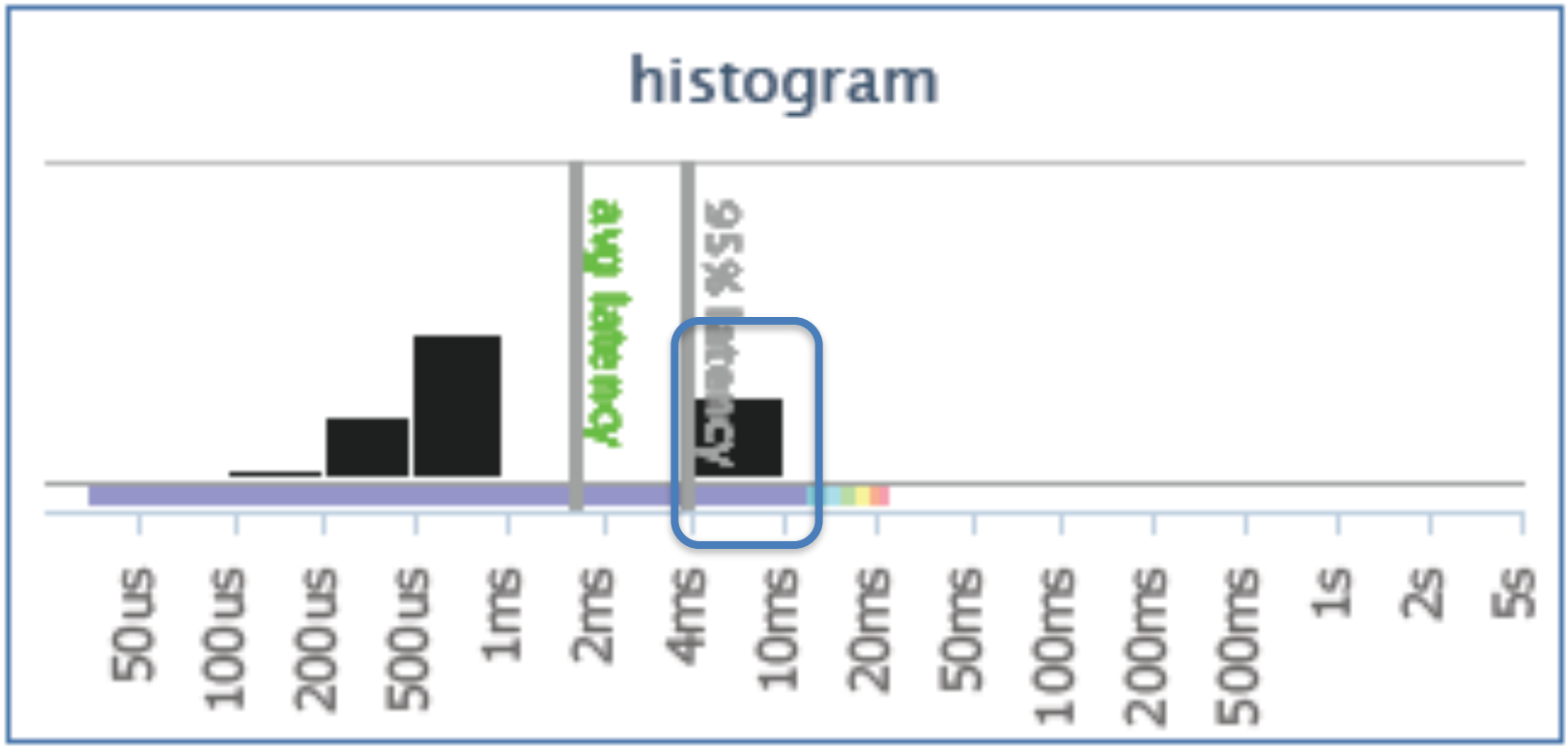
NFS particular

- NFS
- NICs
- Jumbo Frames
- TCP settings

Actual Process

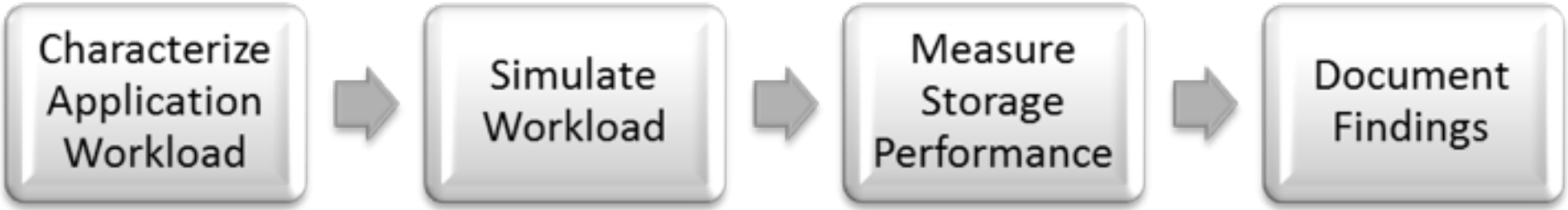


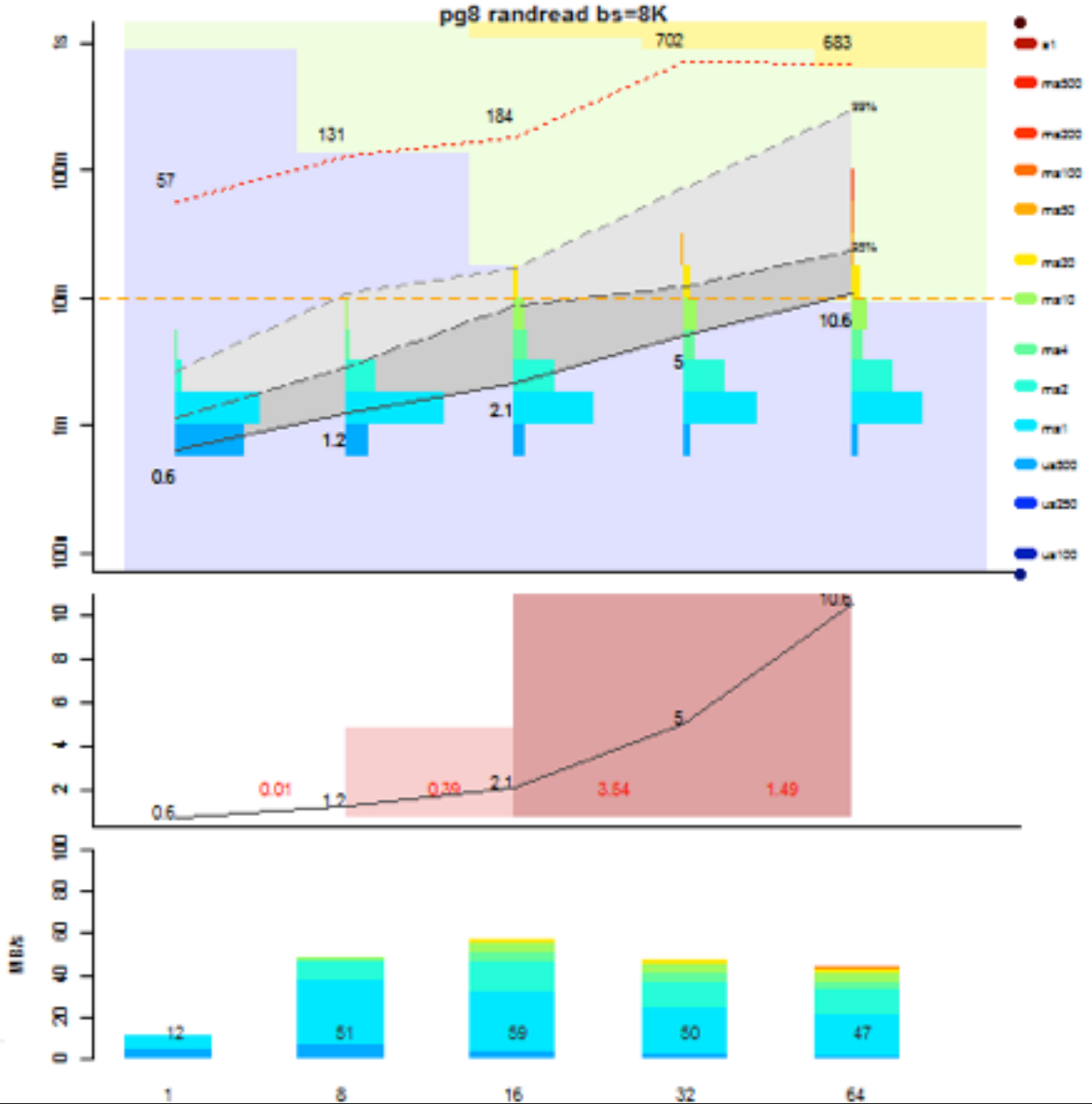
Multiple Readers starting at the same location



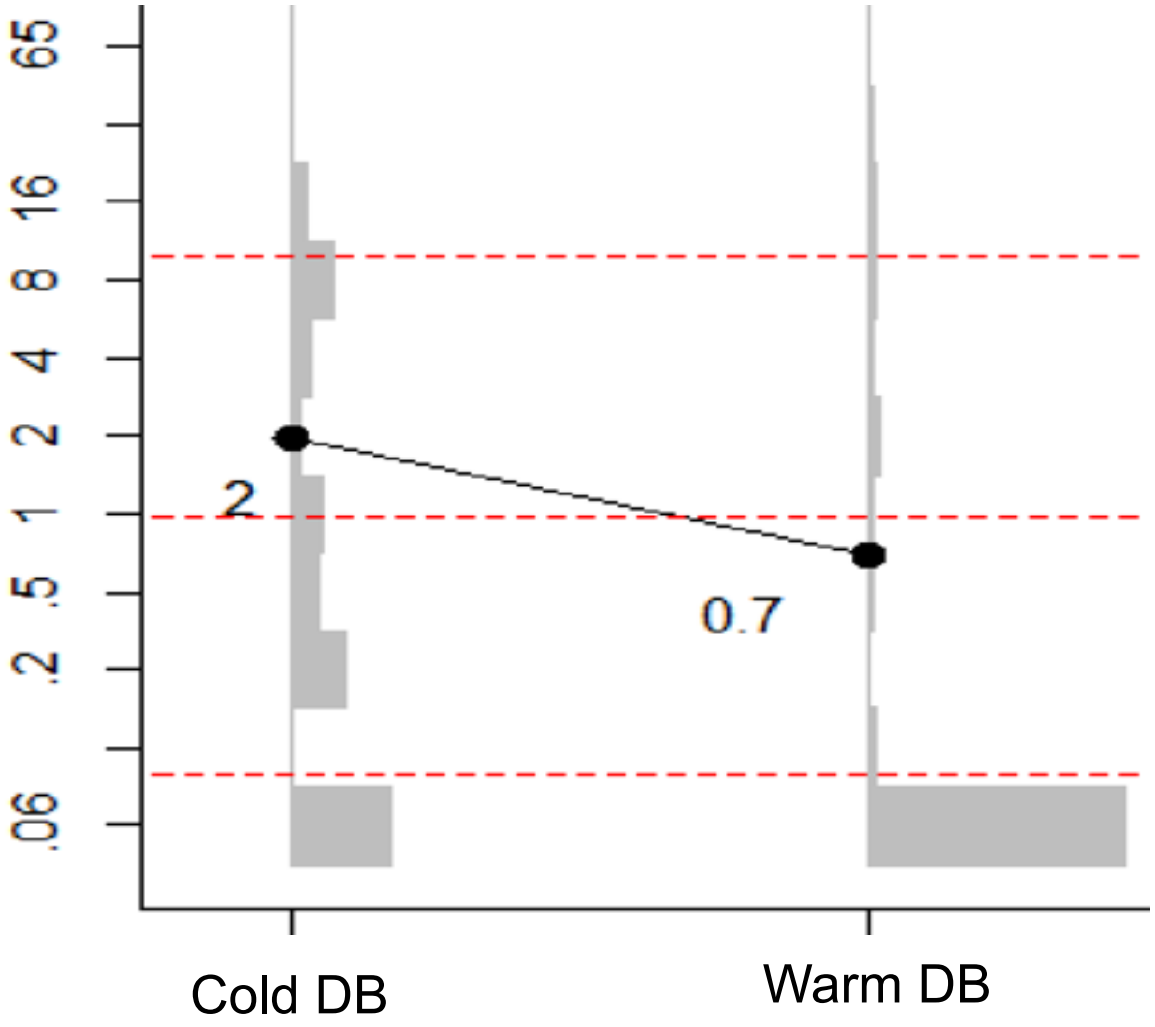
Original Idea

Theory	Simulate	Measure	Document
Characterize	Workload	Performance	Findings




pg8 randread bs=8K
702
683
184
131
57
10.6
5
2.1
1.2
0.6
0.01
0.39
3.54
1.49
MB/s
12
51
59
50
47
1
8
16
32
64

Comparing two databases



5. I/O Request Fragmentation: NFS

- Sun OS : 1M write O_SYNC -> 8k writes
- LINUX: Without Direct I/O, 1M write becomes
  - 1 M unstable NFS write
  - Followed by NFS comit
  - Other OS's issue 1 M NFS Sync write. LINUX with Direct I/O issues does as well.
- SunOS by default sends maximum of 32K
  - Change with nfs3_bsize to 1M
- AIX sends maximum of 64K
  - Changeable with a patch to 512K
- Linux , AIX : Direct I/O cause O_SYNC writes even when not requested
- HP-UX: all writes were O_SYNC requested or not

synchronous / asynchronous - blocking or non-blocking
synchronized / nonsynchronized - stable storage semantics O_[D]SYNC

3d with R : Color same I/O sizes