Best Practices for RHS Performance

RED HAT SUMMIT

CONNECT TO THE INFORMATION REVOLUTION

Ben England

Neependra Khare

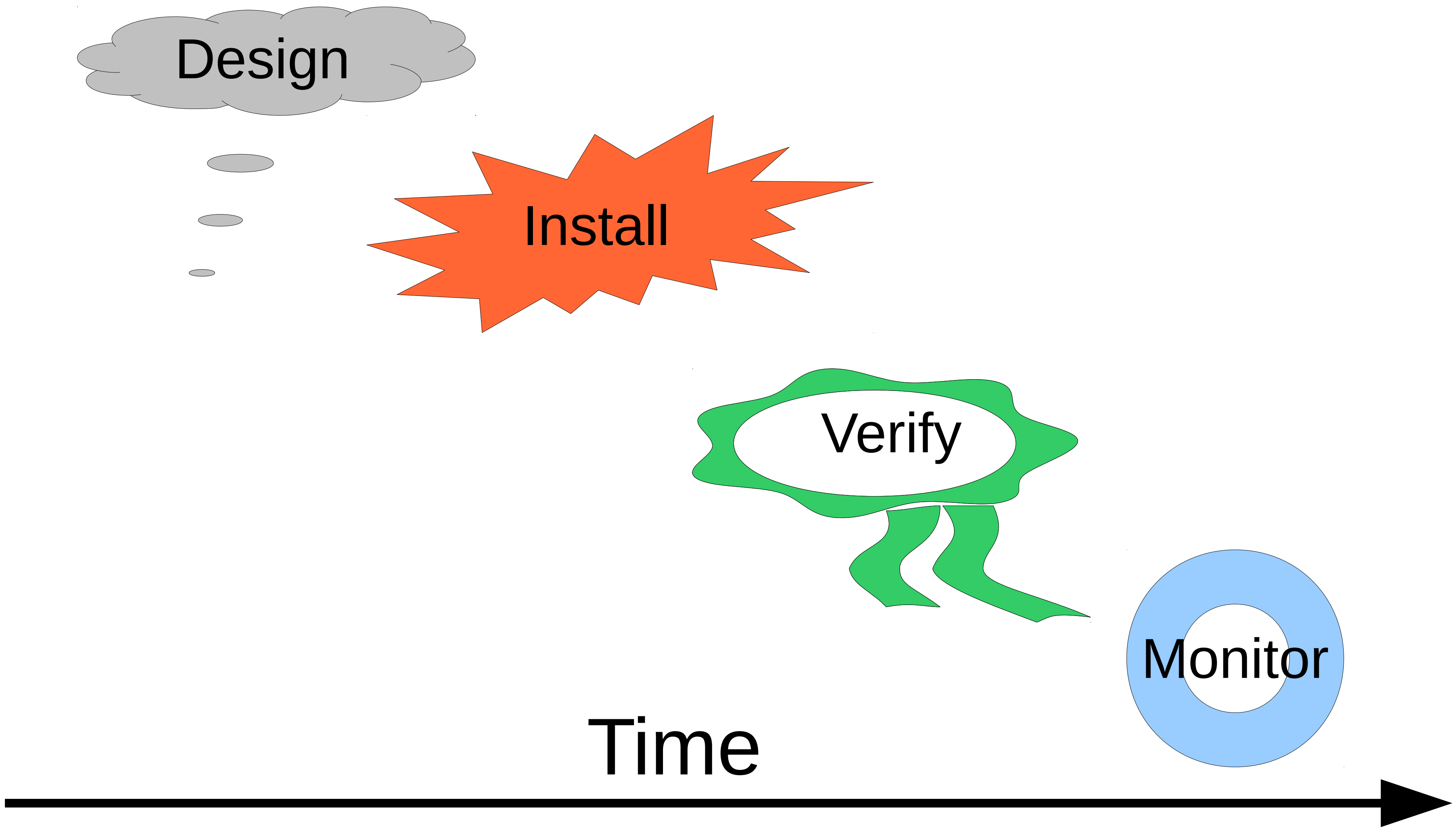
June 2013

redhat.

Scope of this session

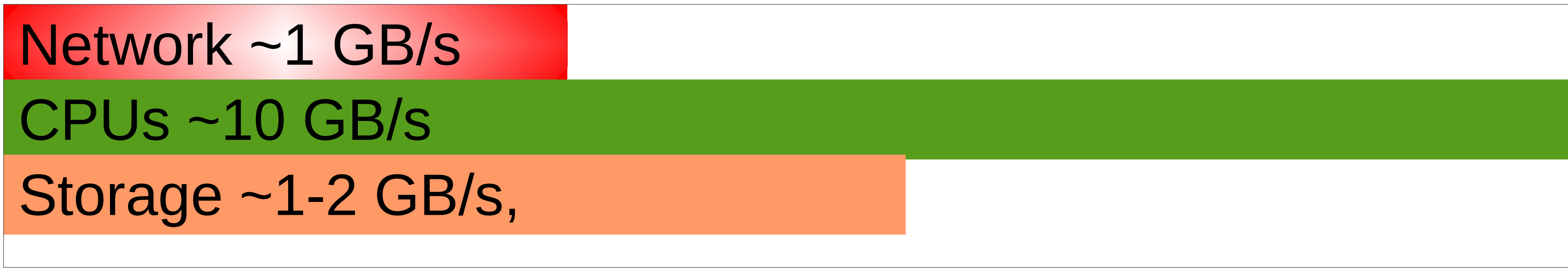
- Quick review of Gluster from perf. perspective
- Best practices to provision faster RHS storage
  - characterize use case (workload, config)
  - provision network + storage appropriately
- GUI and RHS performance

Workflow for provisioning RHS

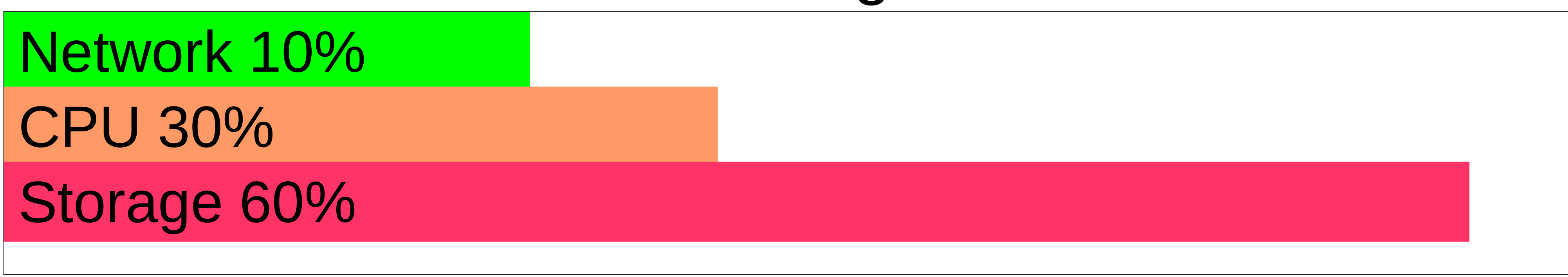


balancing RHS server configuration
don't underfund network

relative capacity – CPUs idle, network busy



relative cost – storage is cost driver

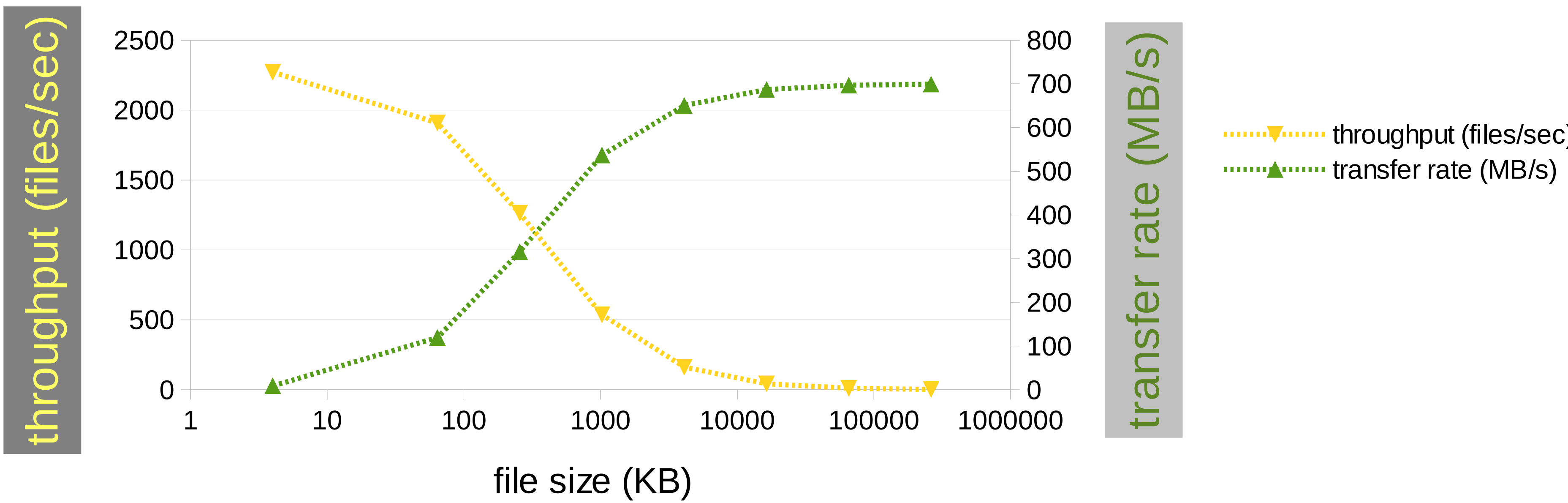


price-performance ratio

file size distribution and effect on sizing

one set of performance measurements with different file sizes
looking at measurements with 2 y-axes

file create, 64 GB total data, 4 threads/client, 4 servers = 4 clients,



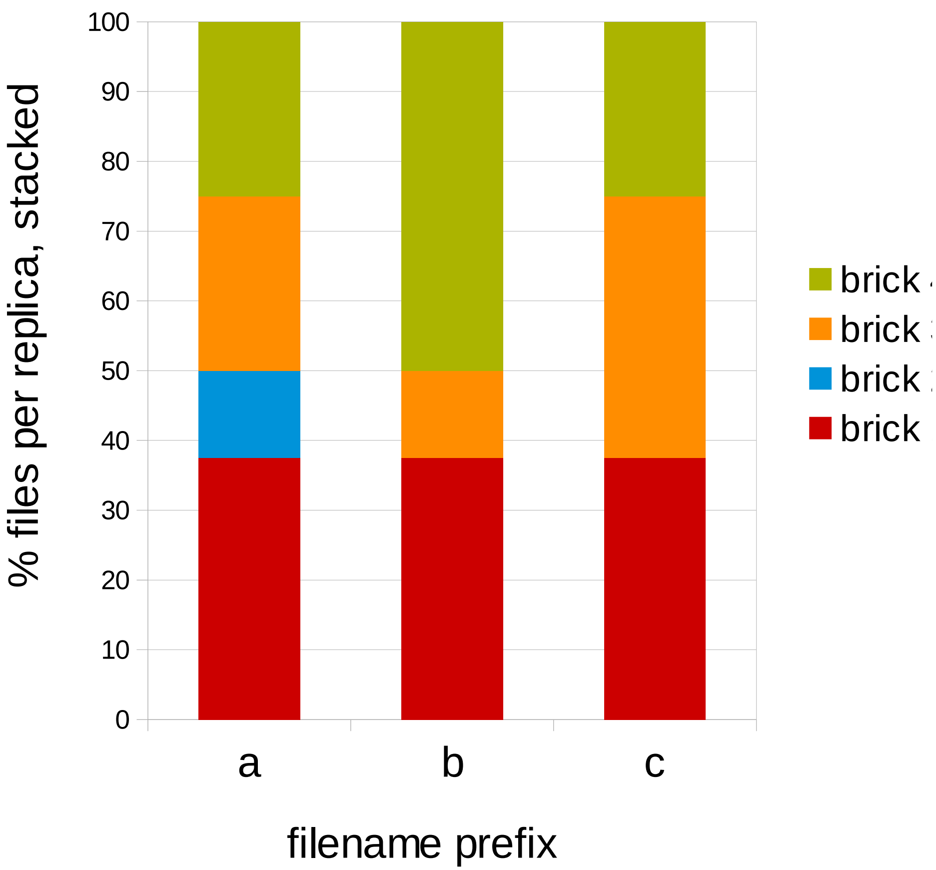
CPU,
Storage

bottleneck

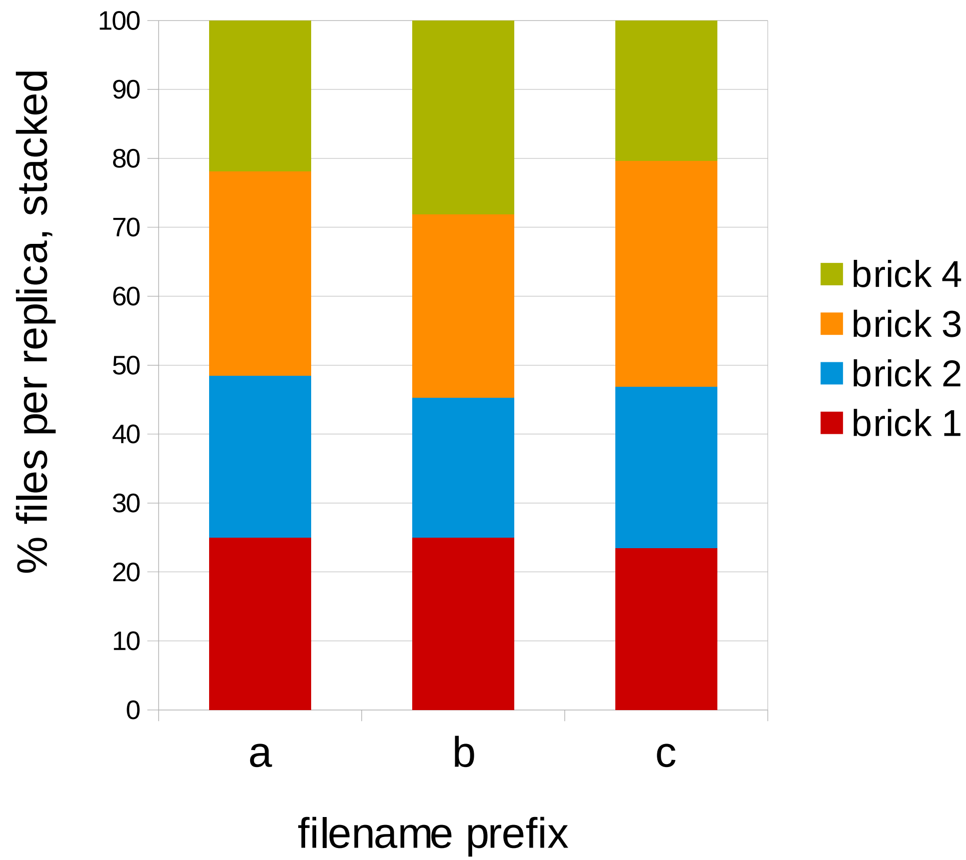
network

streams:server ratio must be >> 1 for even load
example: distribution of files across 4 bricks

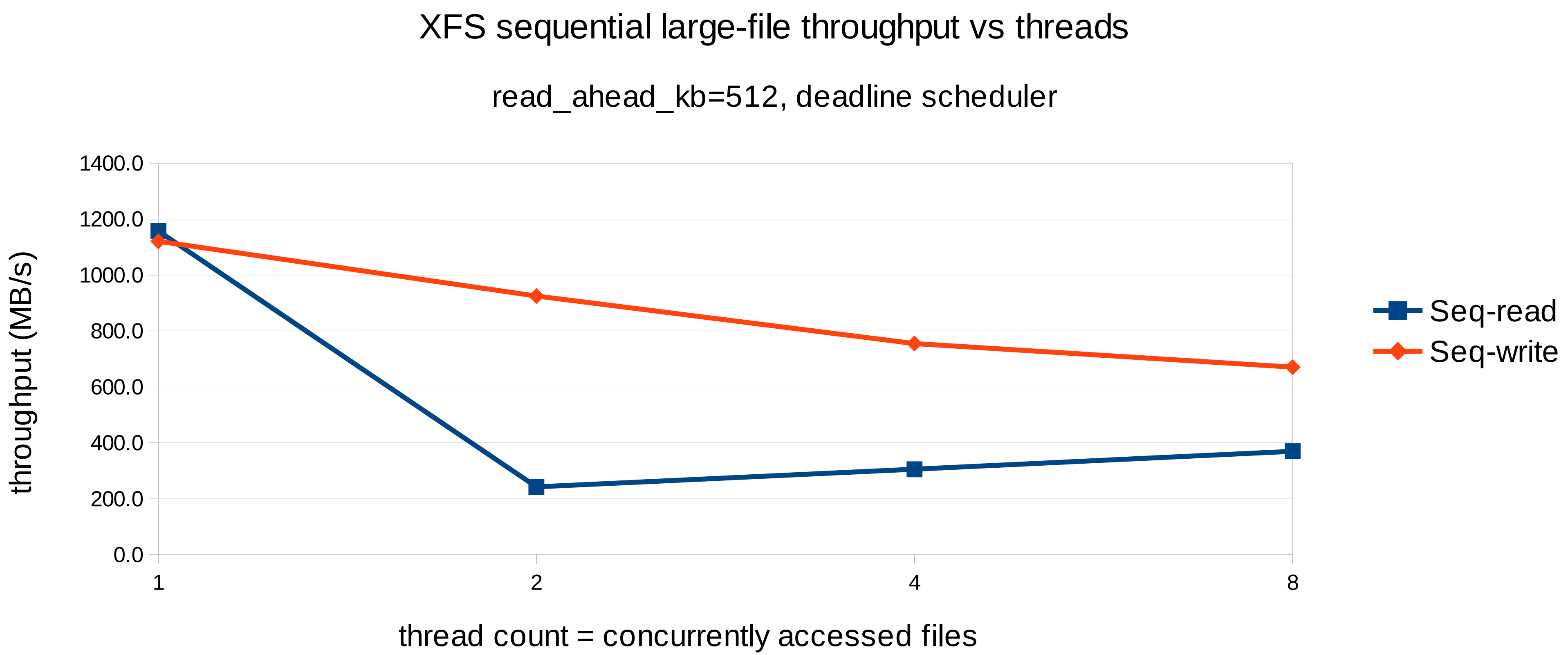
2 files/brick



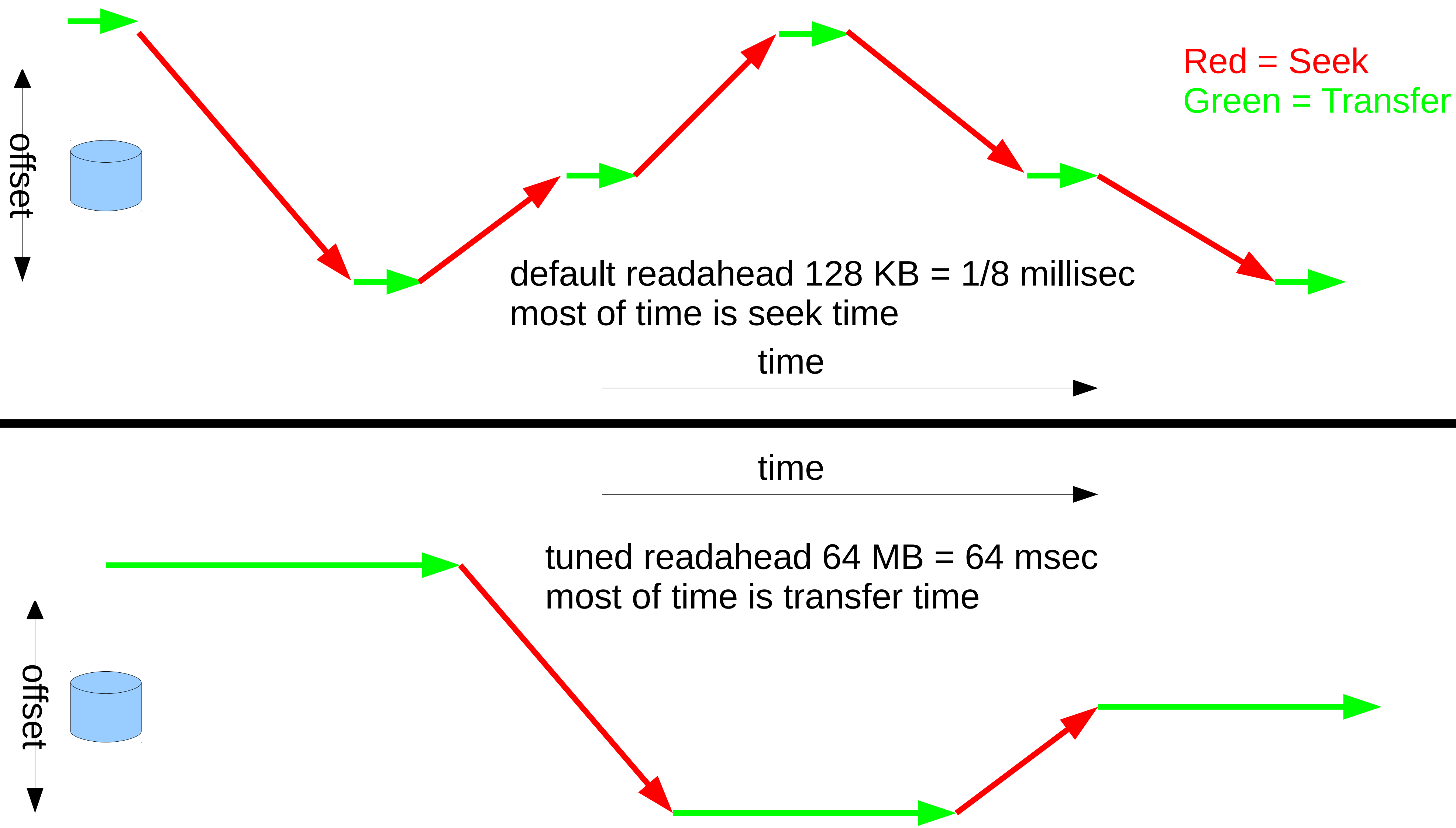
16 files/brick



with >> 1 stream:brick,
large-file sequential reads see I/O contention



Eliminating seek time in multi-stream read

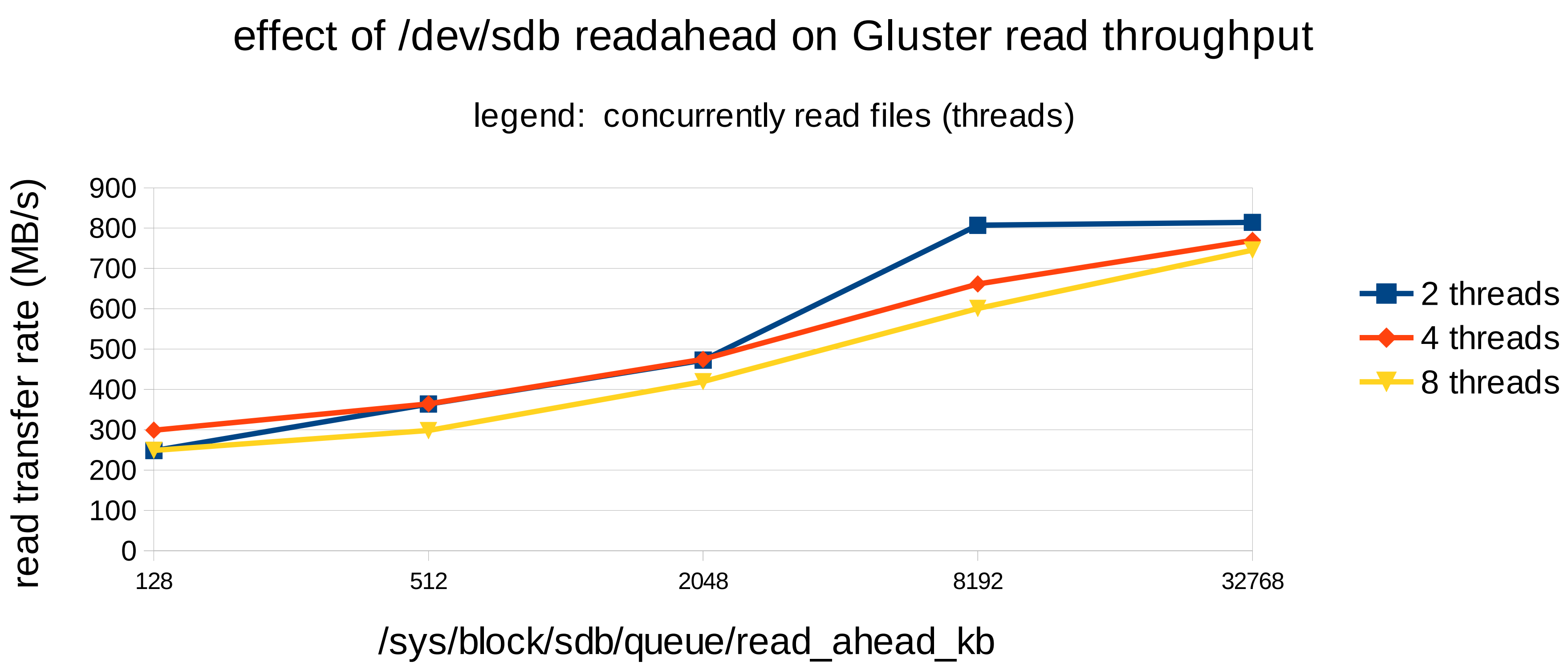


Solution: tuned profile “rhs-high-throughput”

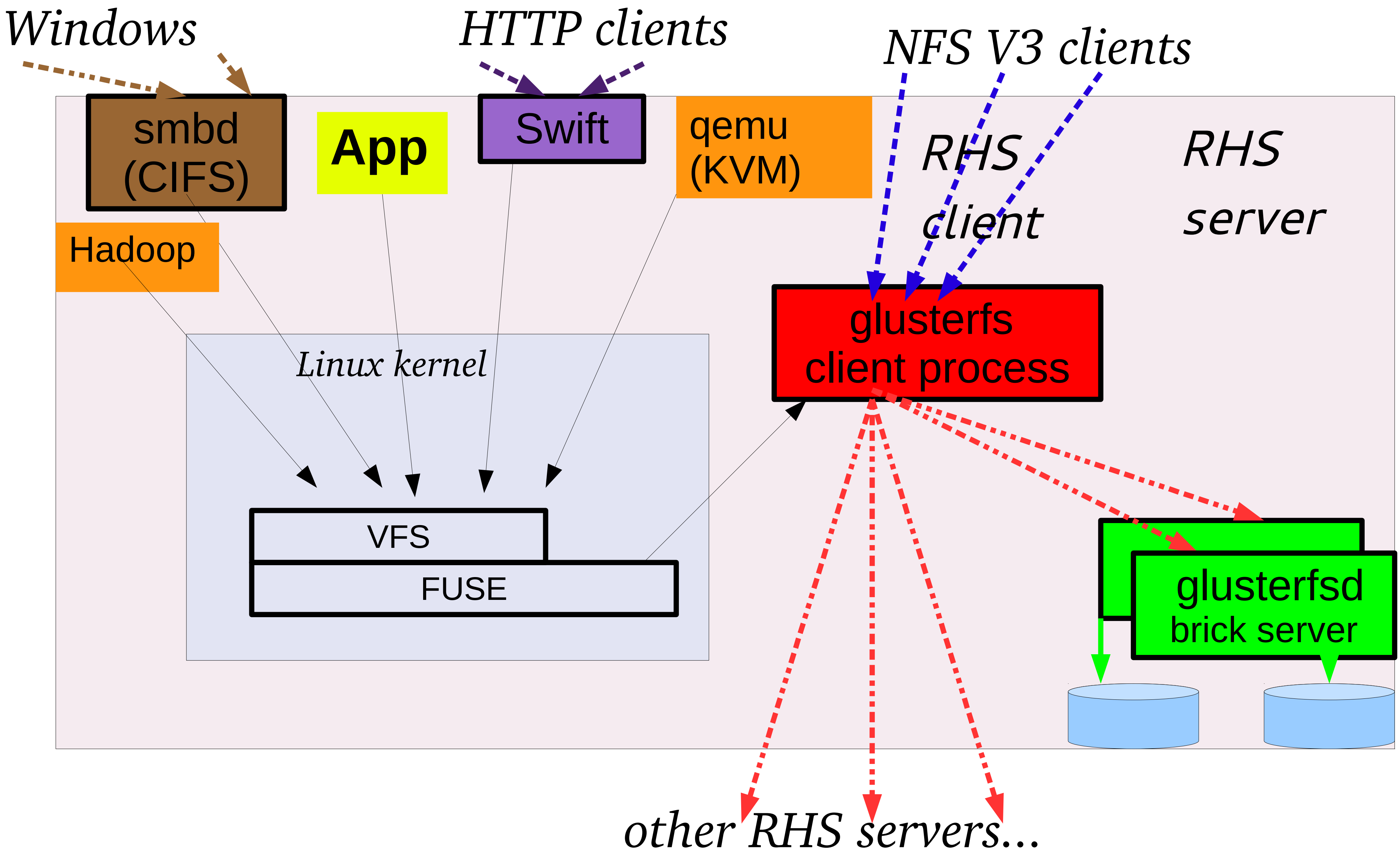
uses 64-MB readahead in XFS bricks

trades off latency for higher throughput

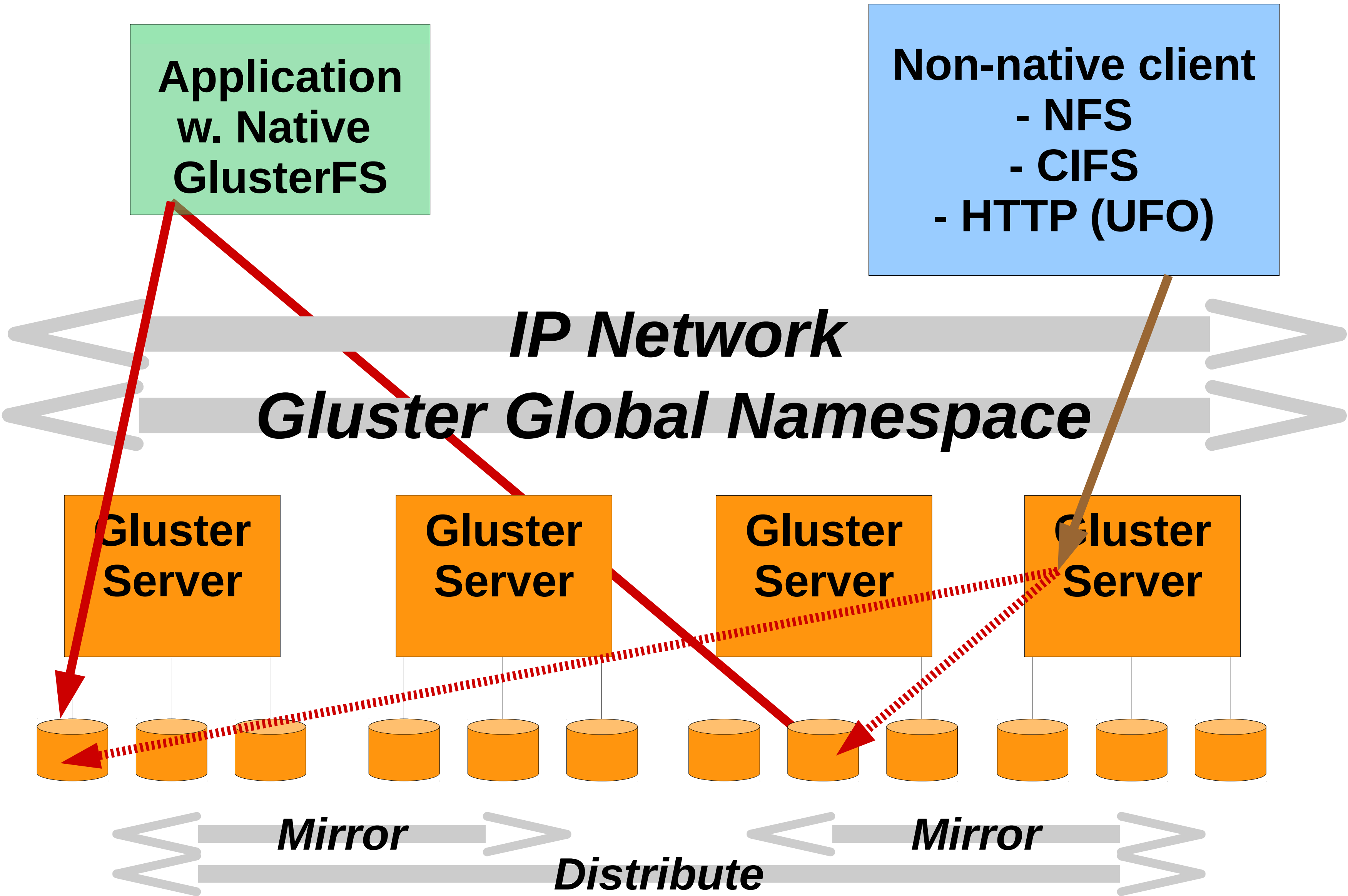
on servers: **tuned-adm profile rhs-high-throughput**



Anatomy of Red Hat Storage (RHS)



non-native protocol adds a network hop



RED HAT STORAGE

before deployment
Is hardware sufficient?

capture network requirements

- non-native reads – 2x application transfer rate
- native writes – 2x app. transfer rate
- non-native writes – 3x app. transfer rate
- clients on same VLAN as Gluster volume?

capture storage requirements:

- random reads – each disk = 100 IOPS
- random writes – each disk = 15 IOPS
- SSDs, flash can temporarily accelerate

recommended storage brick configuration

12 drives/RAID6 LUN, 1 LUN / brick

- limited bricks/volume -> make bricks big

hardware RAID stripe size 256 KB (default 64 KB)

pvcreate –dataalignment 2560k

mkfs.xfs -i size=512 -n size=8192 \
    -d su=256k,sw=10 /dev/vg_bricks/lv*N*

mount options: inode64

Deploying network

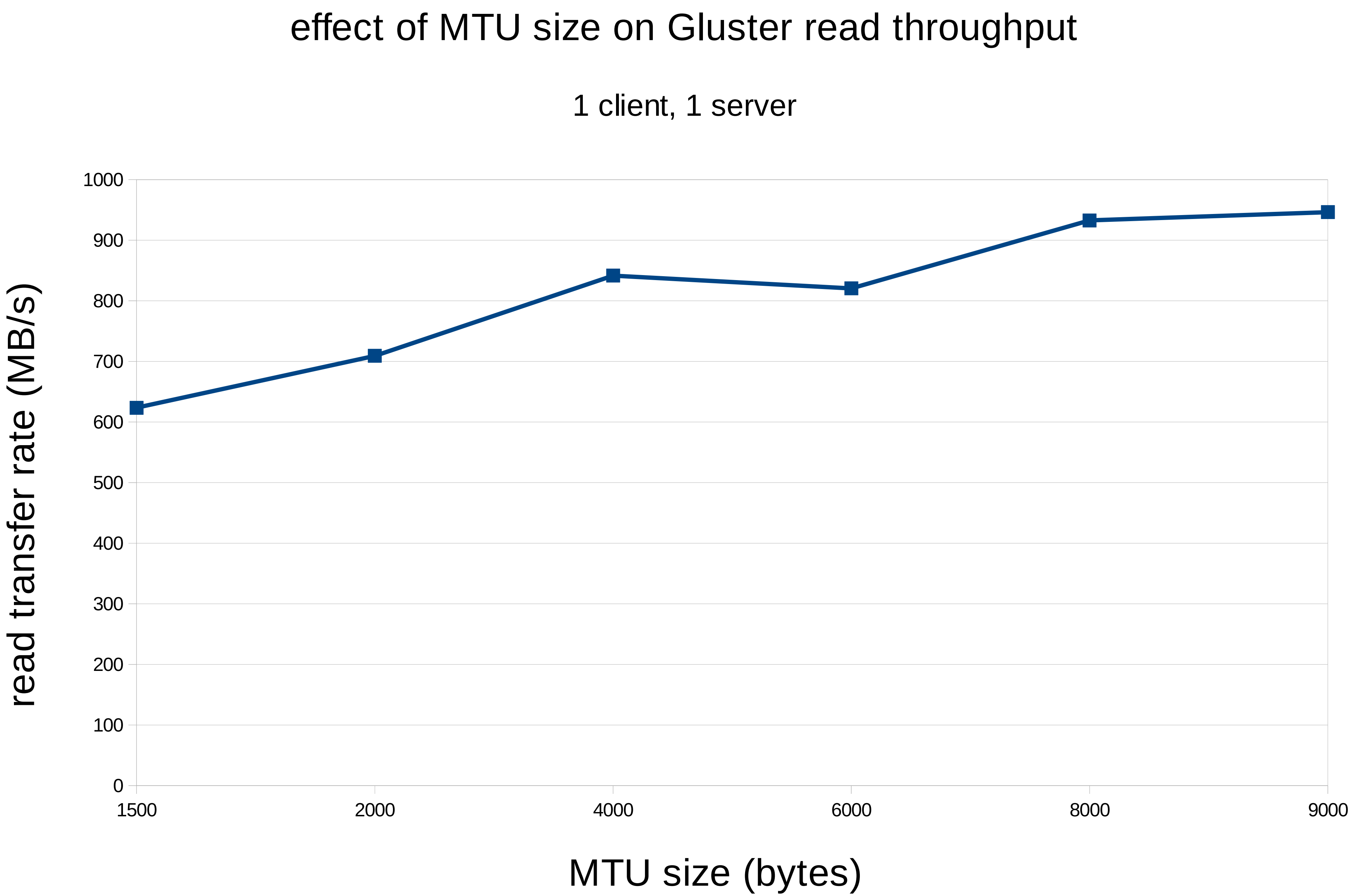
if non-native protocol only, separate Gluster and non-Gluster traffic onto separate VLANs

– separates replication traffic from user traffic

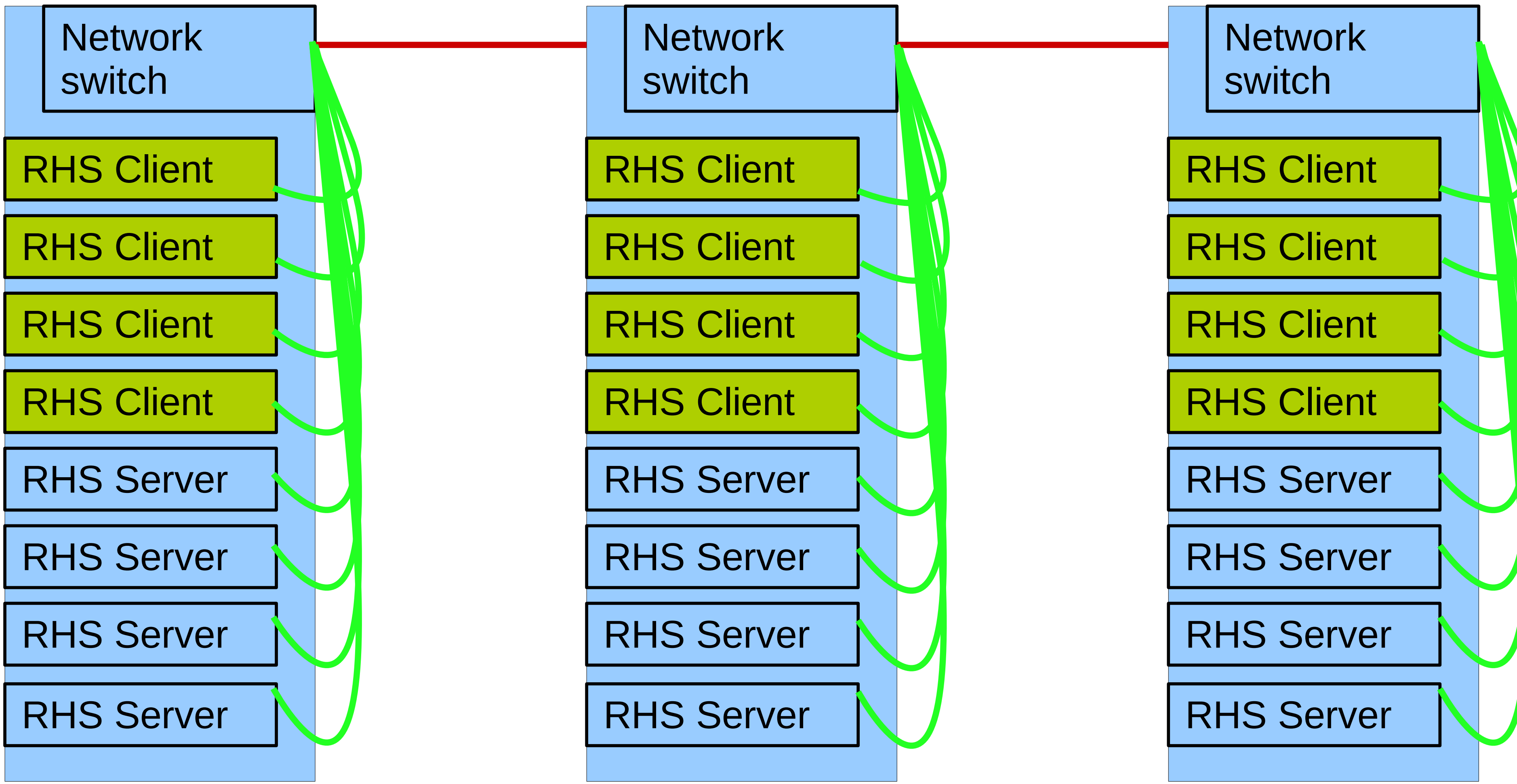
jumbo frames – improve throughput, but requires switch configuration

network design – Gluster doesn't know about switch boundaries

effect of jumbo frames



Network interconnection problem example top-of-rack switches



tuning tips

gluster volume set *your-vol* eager-lock on

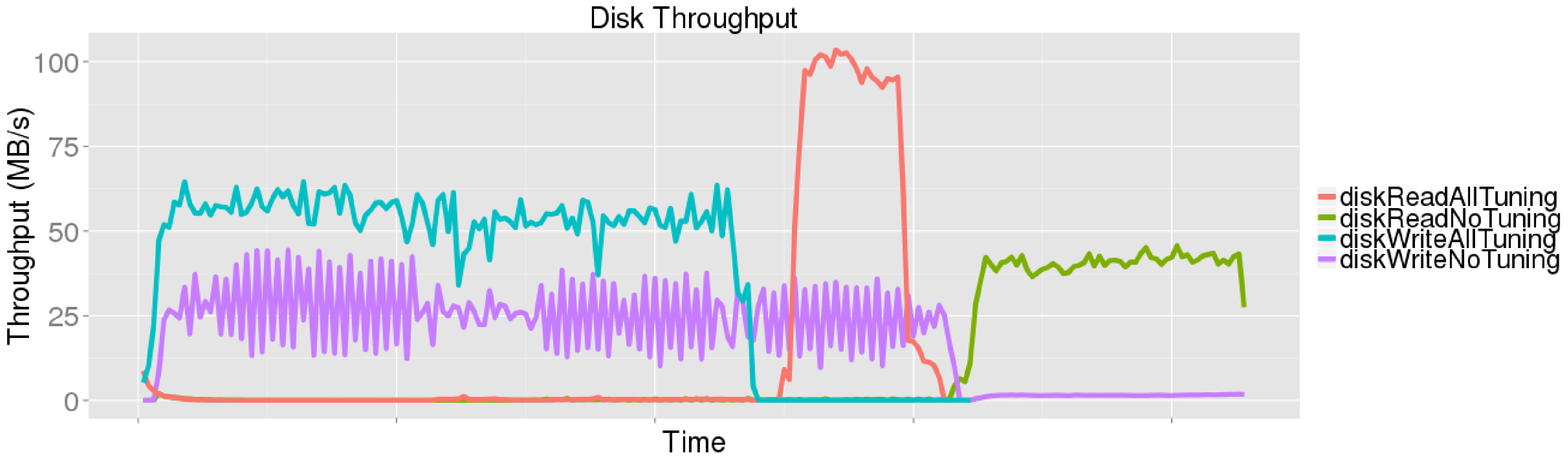
per-server:

- **tuned-adm profile rhs-high-throughput**

native (glusterfs) client:

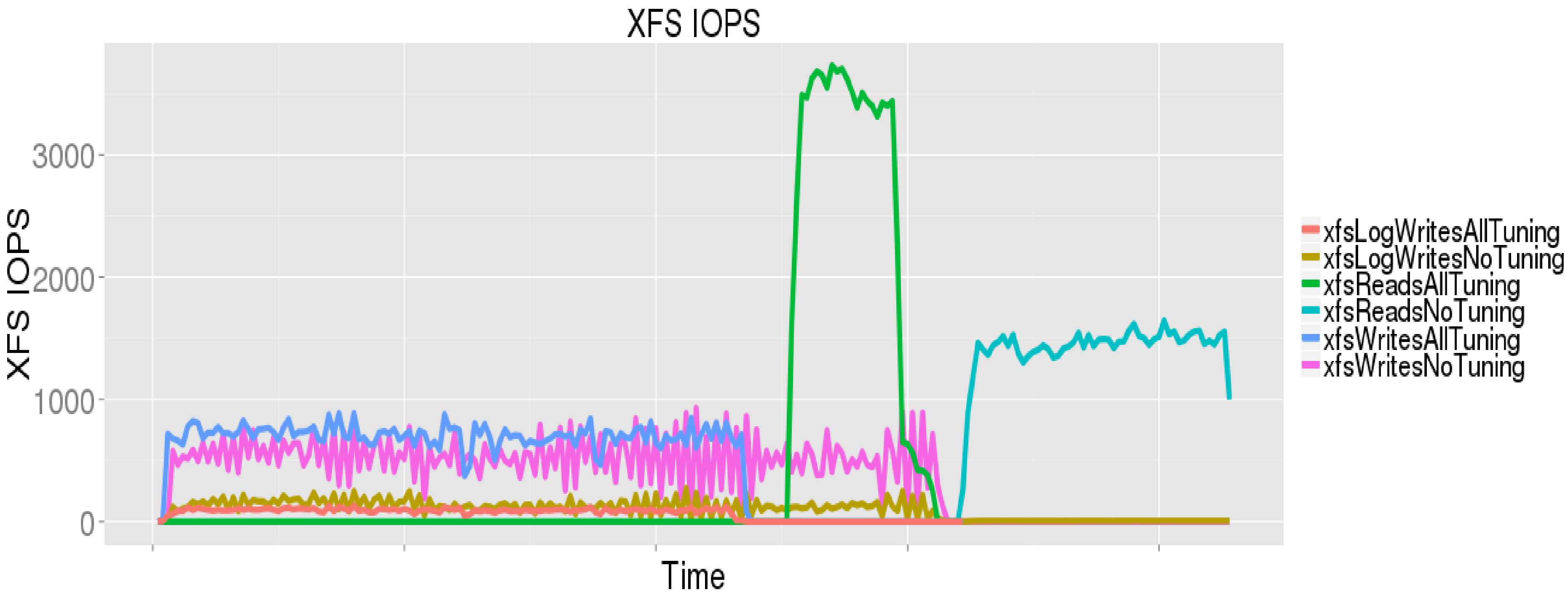
- write transfer size > 32 KB
- avoid single-threaded apps
- more bricks, mountpoints help wi random I/O, small files

Performance with and without brick recommendations

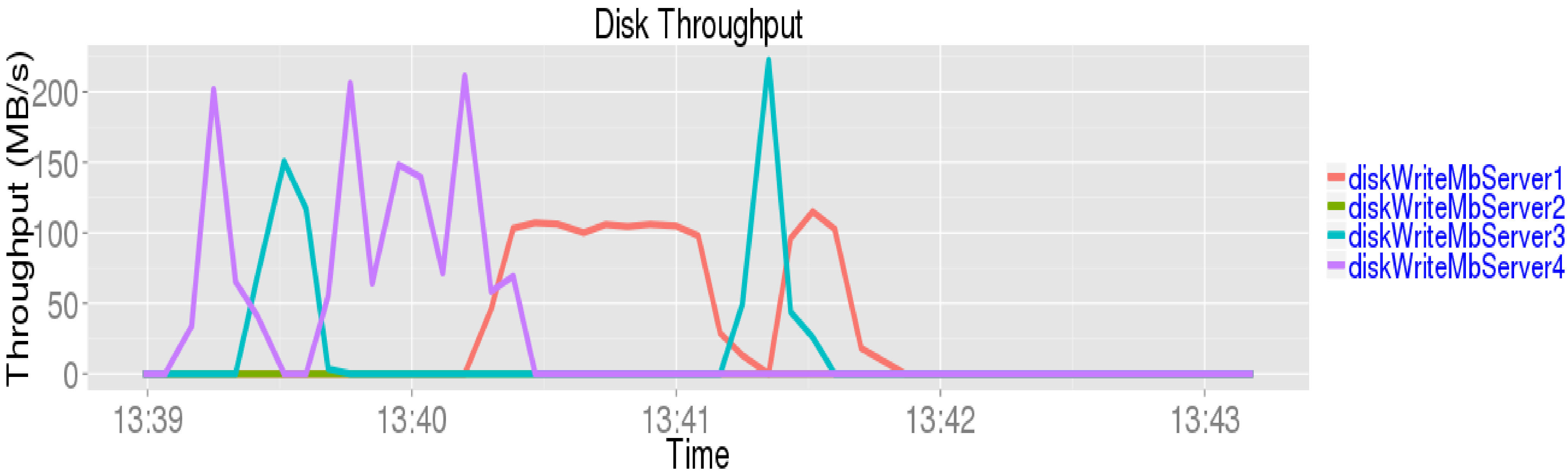


RED HAT STORAGE

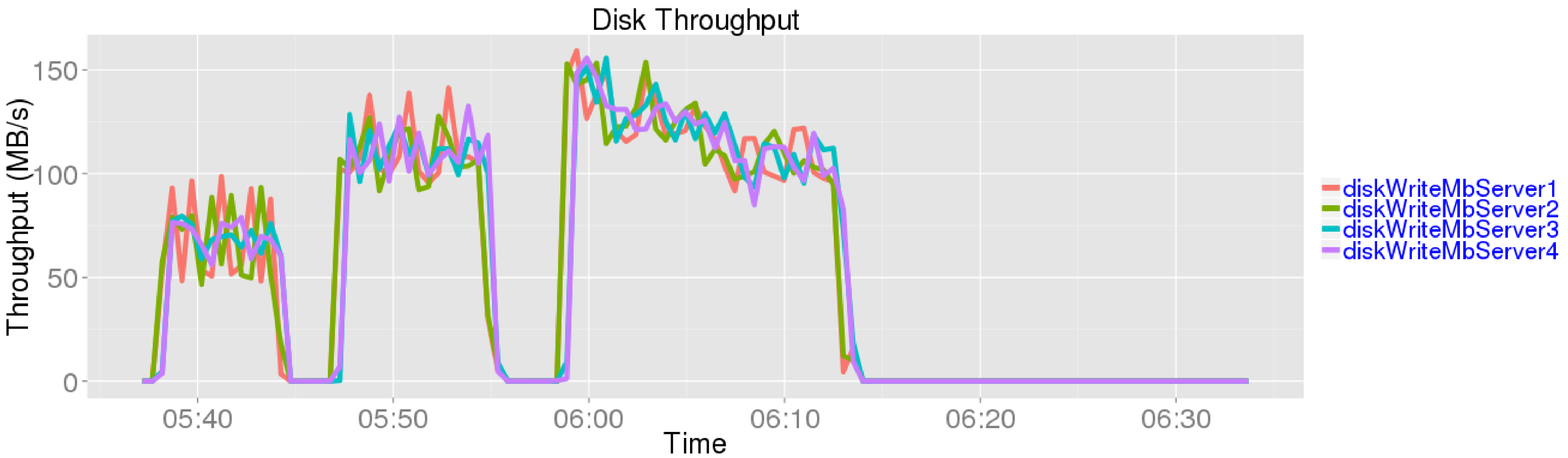
Performance with and without brick recommendations



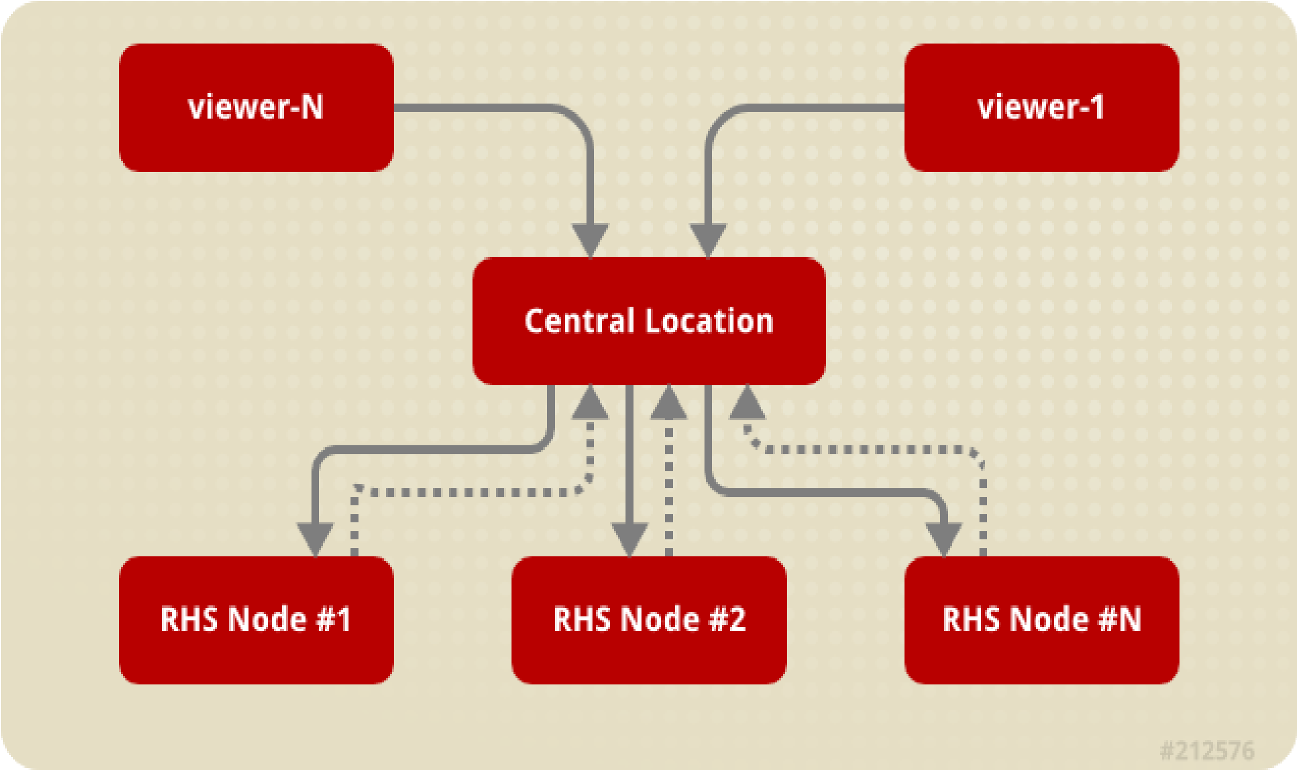
Hashing Behaviour for small number of files



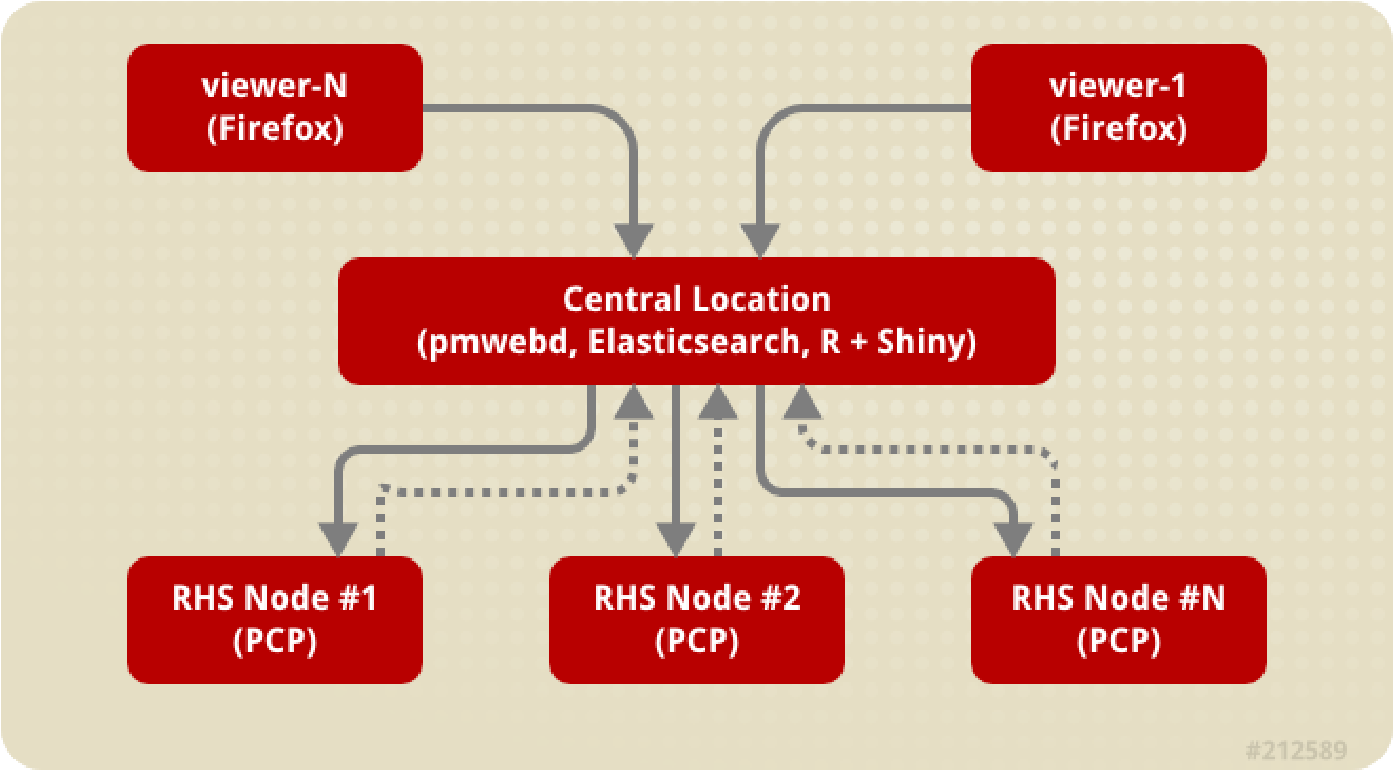
Hashing Behaviour for large number of files



Real time data collection and analysis tool



Real time data collection and analysis tool



Real time data collection and analysis tool

- On RHS nodes
  - System tool to collect the stats
    - PCP (Performance Co-Pilot)
      - http://oss.sgi.com/projects/pcp/
- On Central Location
  - Way to query specific performance stats from RHS nodes.
    - pmwebd
      - A web daemon of PCP to request the stats from PCP service running on RHS nodes.
  - Way to store the results from above query
    - Elasticsearch (http://www.elasticsearch.org/)
      - distributed RESTful search and analytics
  - Way to serve requested stats from the viewers
    - R
      - R is a free software environment for statistical computing and graphics.
    - Shiny
      - Turn analyses of R into interactive web applications that anyone can use.
- On Viewers
  - Web browser

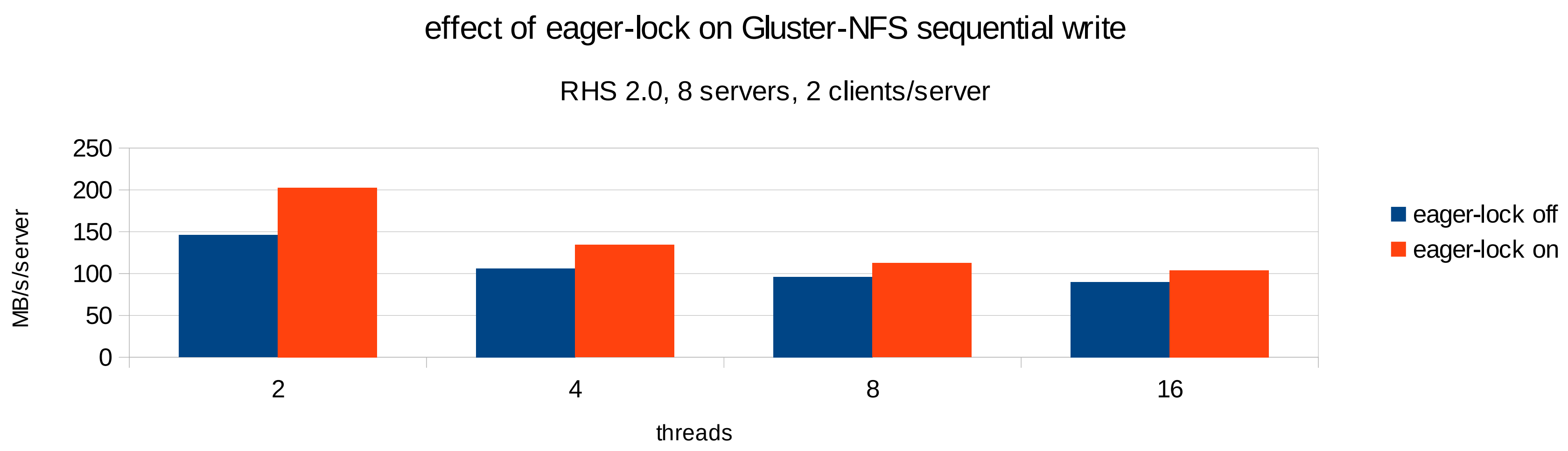
Real time data collection and analysis tool

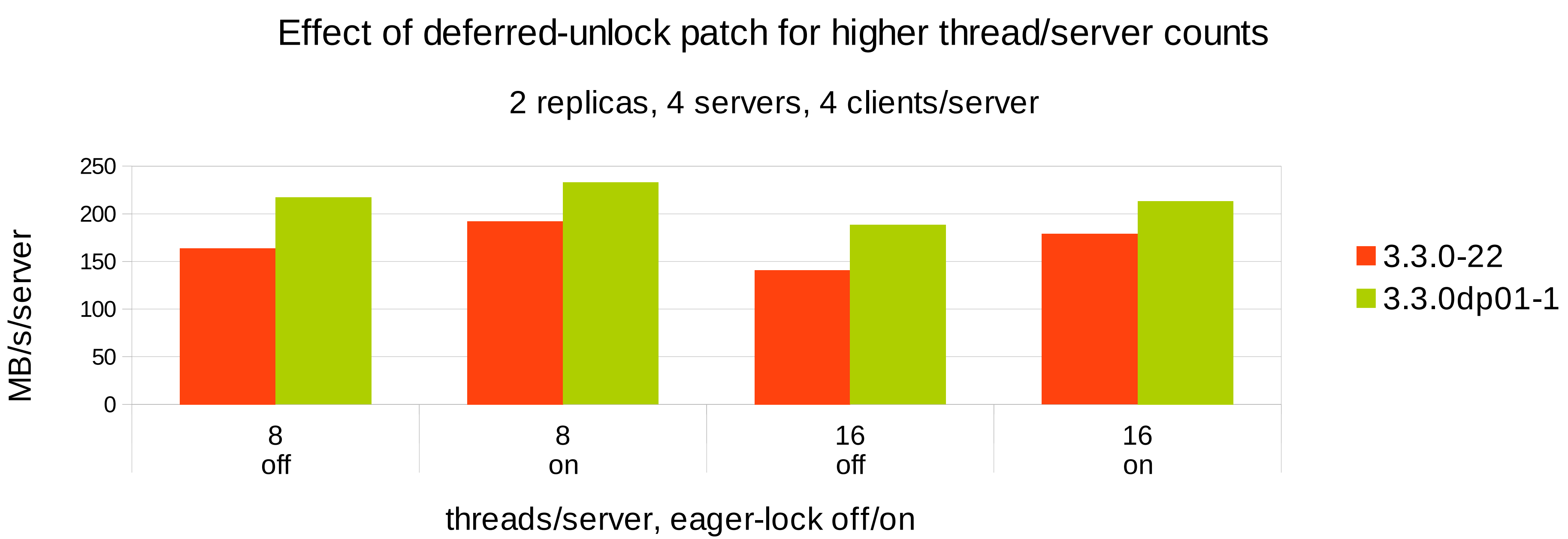
https://forge.gluster.org/gluster-performance-stats-collection-and-analysis-tool

End of presentation

Additional material in slides that follow

NFS large-file performance gains





IOPS – requests/sec bottleneck

In Object Server PUT workload

- glusterfs thread is 85% of 1 core -- bottleneck
- average system CPU core utilization 20%

gprfc017 : root <2>

File Edit View Bookmarks Settings Help

```
top - 11:43:59 up 14 days,  1:45,  1 user,  load average: 0.74, 0.30, 0.23
Tasks: 719 total,   2 running, 717 sleeping,   0 stopped,   0 zombie
Cpu(s): 13.3%us,  4.8%sy,  0.0%ni, 80.3%id,  0.0%wa,  0.0%hi,  1.6%si,  0.0%st
Mem:  49409944k total, 18935072k used, 30474872k free,   163312k buffers
Swap: 51642360k total,        0k used, 51642360k free,   530520k cached

  PID USER      PR  NI  VIRT  RES  SHR S %CPU %MEM    TIME+  COMMAND
31258 root      20   0  263m  36m 2084 S 85.7  0.1   6:40.82 glusterfs
31246 root      20   0  263m  36m 2084 R 24.1  0.1   3:39.99 glusterfs
 9992 root      20   0  231m  28m 1800 S 23.5  0.1   0:21.26 swift-proxy-ser
 9987 root      20   0 1702m  21m 1840 D 22.2  0.0   0:20.46 swift-object-se
 9973 root      20   0 1702m  21m 1840 S 21.8  0.0   0:20.53 swift-object-se
 9977 root      20   0 1702m  21m 1840 S 21.8  0.0   0:20.73 swift-object-se
 9986 root      20   0 1702m  21m 1840 D 21.8  0.0   0:20.37 swift-object-se
 9981 root      20   0 1702m  21m 1840 D 21.5  0.0   0:20.55 swift-object-se
 9994 root      20   0  231m  28m 1800 S 21.5  0.1   0:21.27 swift-proxy-ser
 9995 root      20   0  229m  26m 1800 S 21.5  0.1   0:21.06 swift-proxy-ser
 9996 root      20   0  232m  29m 1800 S 21.5  0.1   0:21.66 swift-proxy-ser
 9975 root      20   0 1702m  21m 1840 S 21.2  0.0   0:20.65 swift-object-se
 9979 root      20   0 1702m  21m 1840 S 20.8  0.0   0:20.43 swift-object-se
 9984 root      20   0 1702m  21m 1840 D 20.5  0.0   0:20.38 swift-object-se
 9990 root      20   0  229m  26m 1800 S 20.2  0.1   0:21.57 swift-proxy-ser
 9991 root      20   0  233m  29m 1800 S 20.2  0.1   0:21.23 swift-proxy-ser
 9993 root      20   0  231m  28m 1800 S 20.2  0.1   0:21.28 swift-proxy-ser
 9989 root      20   0  231m  27m 1800 S 19.5  0.1   0:21.11 swift-proxy-ser
 9983 root      20   0  226m  13m 1488 S 10.9  0.0   0:12.14 swift-container
 9985 root      20   0  226m  13m 1500 S 10.9  0.0   0:12.32 swift-container
 9988 root      20   0  226m  13m 1500 S 10.9  0.0   0:12.12 swift-container
 9980 root      20   0  226m  13m 1488 S 10.3  0.0   0:12.15 swift-container
15876 root      20   0  242m 1552  984 S  2.3  0.0   0:11.76 rsyslogd
12938 root      20   0  242m 1552  984 S  2.3  0.0   0:02.15 rs:main Q:Reg
 2970 memcache  20   0  325m 3692  608 S  0.7  0.0   0:02.50 memcached
10165 root      20   0 1702m  21m 1840 S  0.7  0.0   0:00.03 swift-object-se
```

gprfc017 : root

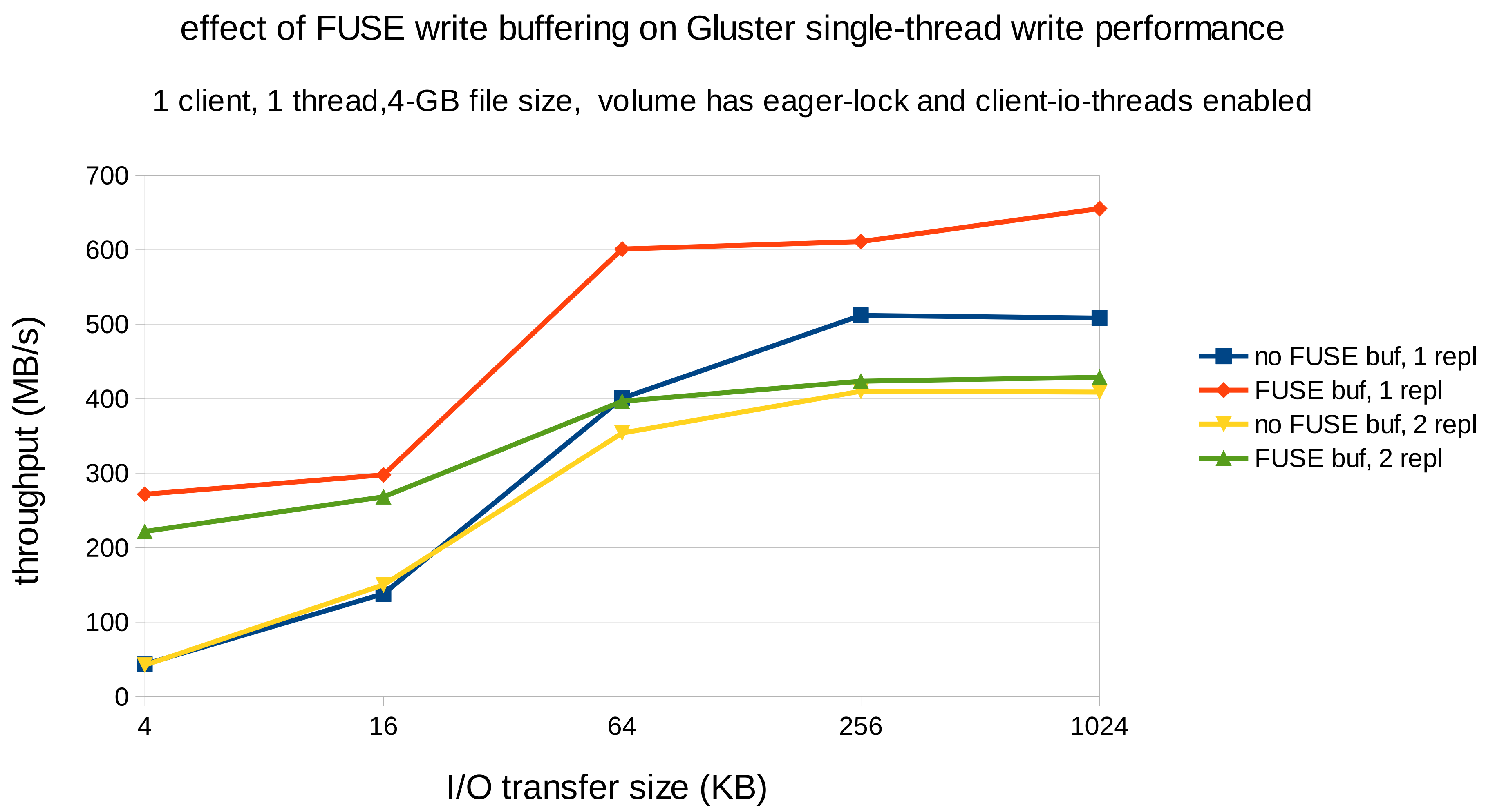
RPC read optimization in RHS 2.1

```
17:39:16.467980 epoll_wait(3, {{EPOLLIN, {u32=9, u64=4294967305}}}, 257,
4294967295) = 1 <0.000013>
17:39:16.468035 readv(9, [{"\200\0\20\214", 4}], 1) = 4 <0.000026>
17:39:16.468097 readv(9, [{"\0 \366$\0\0\0\1", 8}], 1) = 8 <0.000034>
17:39:16.468168 readv(9, [{"\0\0\0\0", 4}], 1) = 4 <0.000010>
17:39:16.468209 readv(9, [{"\0\0\0\0\0\0\0\0", 8}], 1) = 8 <0.000007>
17:39:16.468242 readv(9, [{"\0\0\0\0", 4}], 1) = 4 <0.000008>
17:39:16.468283 readv(9, [...], 1) = 116 <0.000015>
17:39:16.468359 readv(9, [..., 4096}], 1) = 4096 <0.000023>
```

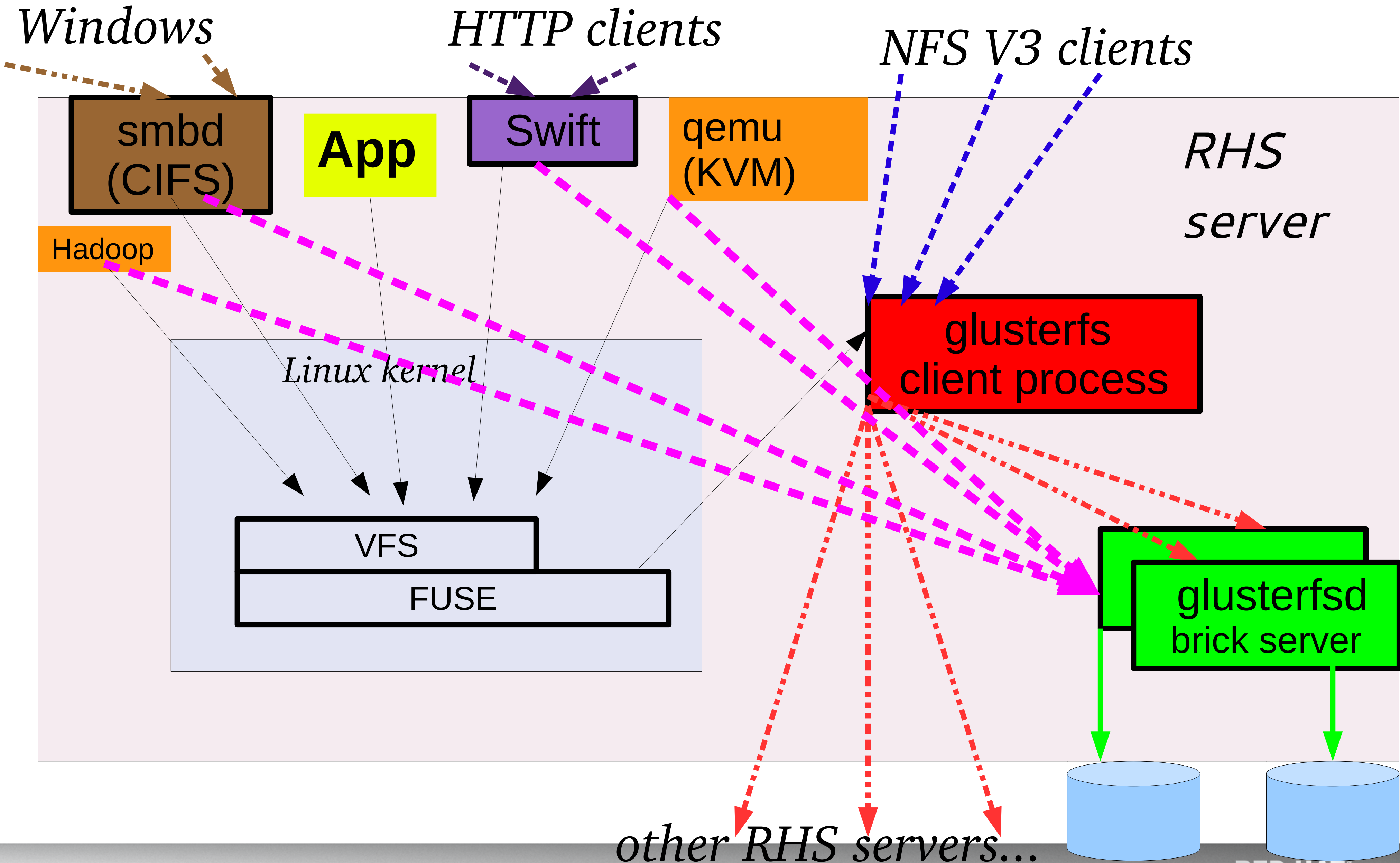
readdirplus - efficient metadata reads

- classic small-file app scans directory and performs operations on files in directory
  - example: **rsync -ravu /home/ben /mnt/glusterfs/ben**
  - problem: each file's metadata requires *network round-trip*, so big directory listings take a long time
  - solution from NFS: **readdirplus** – return directory contents together with important metadata about each file in directory
  - upstream now, release in RHS 2.1

Upstream Linux 3.7 FUSE enhancement for small-transfer writes



Anatomy of Red Hat Storage (RHS) – libgfapi



The network IS the system

test between clients and servers if possible

test between servers 2nd best

network – scripts available

large-file – **iozone -+m (-+h)**

small-file – **smallfile_cli.py --host-set**

Swift – **ssbench** will exercise

potential perf. problems in storage device

Check no failed disks – RAID6 will continue but performance will degrade.

Check writeback caching is enabled. If dependent on battery, check that battery is installed and working.

disconnected brick can cause perf. degradation (example: brick wasn't mounted)

capturing perf. problems onsite

top utility – press **H** to show per-thread CPU utilization, will detect “hot-thread” problems where thread is using up its core

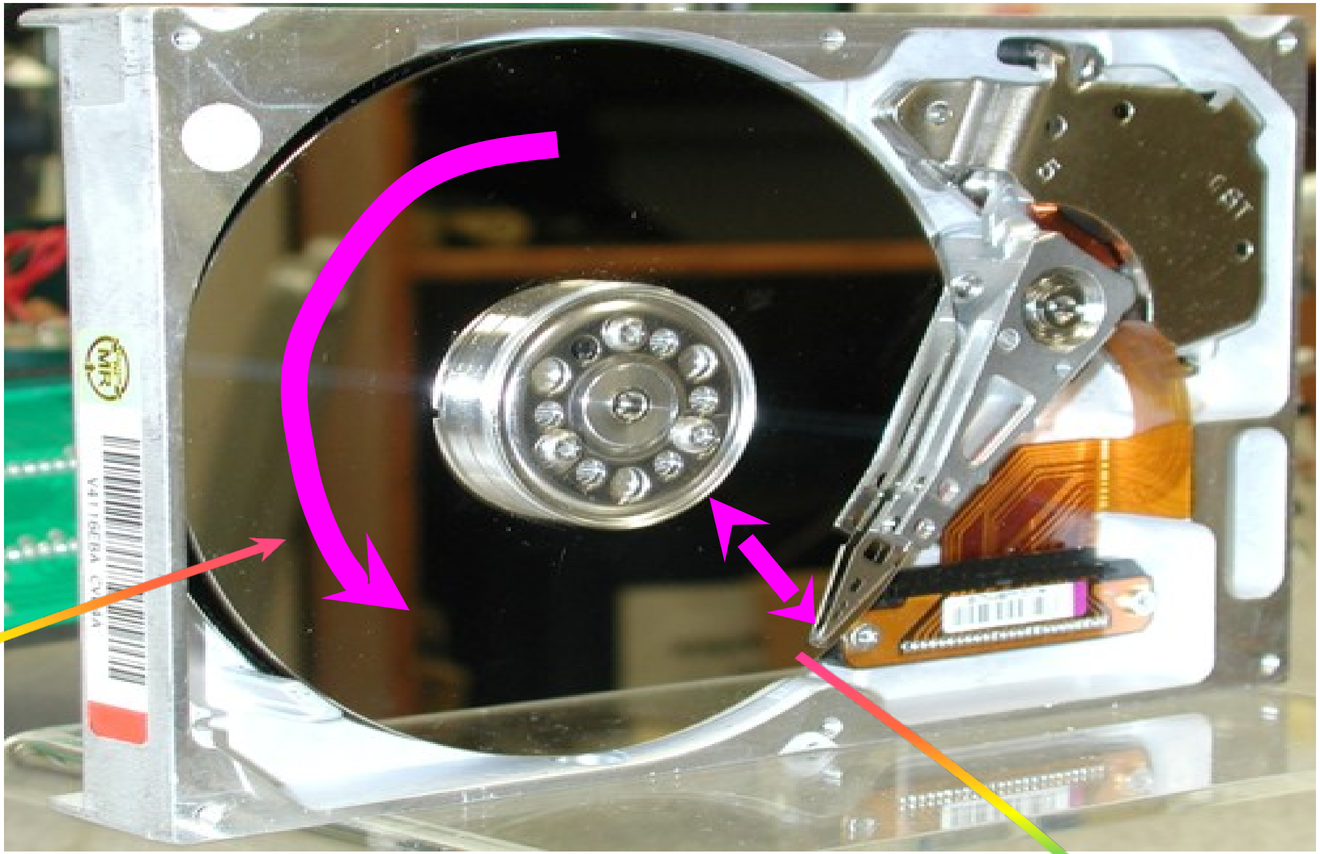
NFS: **nfsiostat** and **nfsstat** utilities

gluster volume profile – shows latency, throughput for Gluster RPC operations

gluster volume top – shows which files, servers are hot

Wireshark Gluster plug-in – isolates problems

effect of non-sequential I/O



7200 RPM → 4.15 ms average rotation time

Assume 5 ms average cylinder seek time

At 50 MB/s, 32 KB request size takes 0.625 ms per transfer

Total = about 10 ms / rq ➔ 100 random IOPS/drive

divide by 6 for random writes to RAID6 LUN

- divide by 2 for random writes to RAID10 LUN