SDC
STORAGE DEVELOPER CONFERENCE
SNIA ■ SANTA CLARA, 2014

Best Practice on Distributed Intelligent Storage with NVMe-SSDs and Fast Interconnect

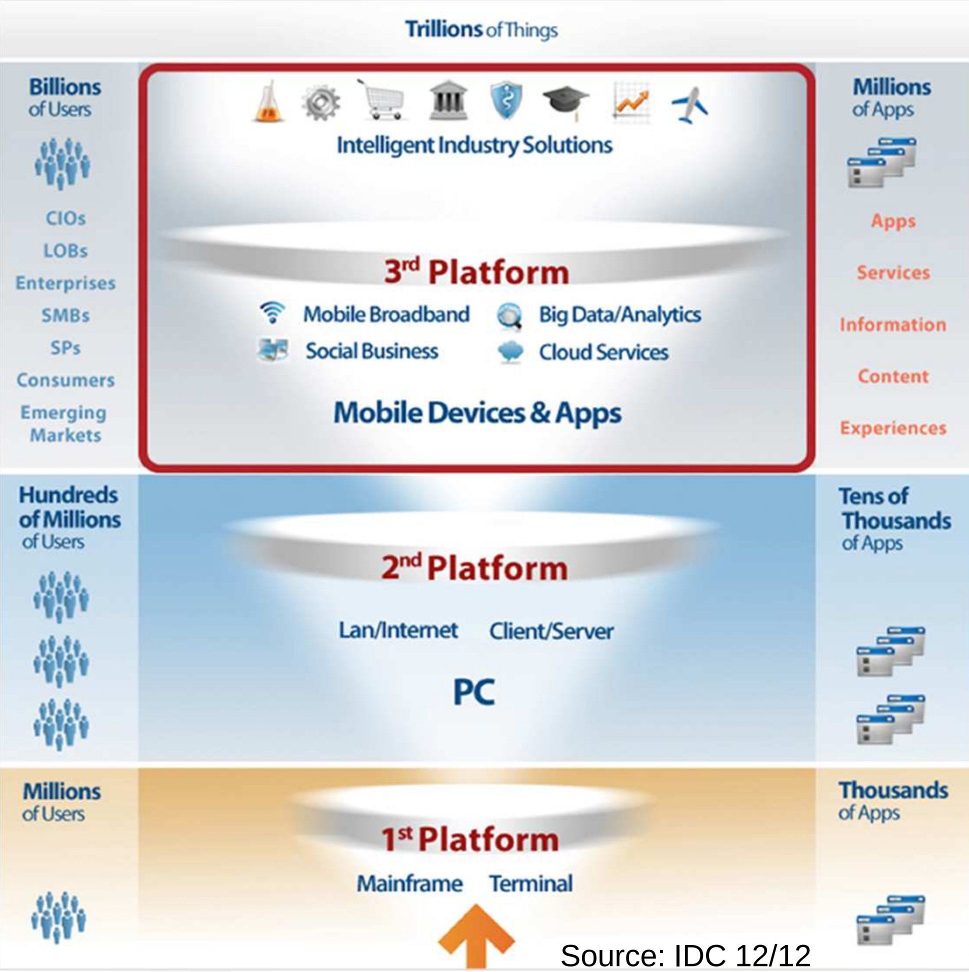
Dieter Kasper
Fujitsu

v8

Agenda

- **Introduction & Motivation**
- Hardware & System Software
- Software preparation & analysis
- Performance test cube
- Conclusion

IDC 3rd Platform: Opportunities at the intersection of Mobile, Cloud, Social and Big Data



Source: IDC 12/12

- From 2013 through 2020, **90% of IT industry growth will be driven by 3rd Platform technologies** that, today, represent just 22% of ICT spending
- Services will be build on **innovative mash-ups** of cloud, mobile devices/apps, social technologies, big data/analytics, and more
- Data Center Transforming
  - **Converged systems** will account for over 1/3 of enterprise cloud deployments by 2016
  - Software-defined networks will penetrate 35% of Ethernet switching in the data center
  - Growing importance of **mega DC, Service**

3rd Platform Implications for Storage

3rd Platform

- Big Data Analytics // Social Business // Mobile Broadband // Cloud Services

Scalability

- practically unlimited scalability in terms of performance & capacity
- no bottlenecks
- no hot spots

Reliability

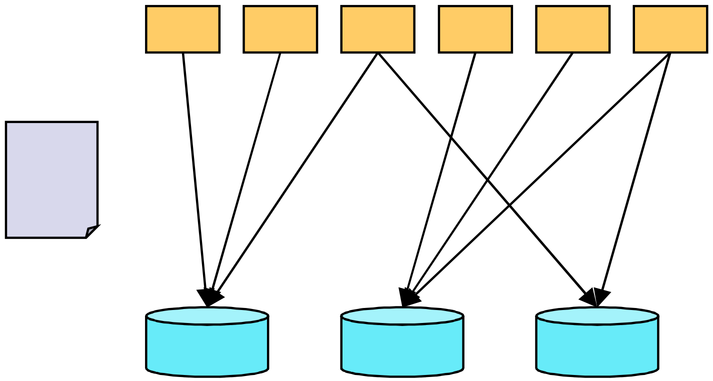
- full redundancy
- self healing
- geographical dispersion
- fast rebuild

Manageability

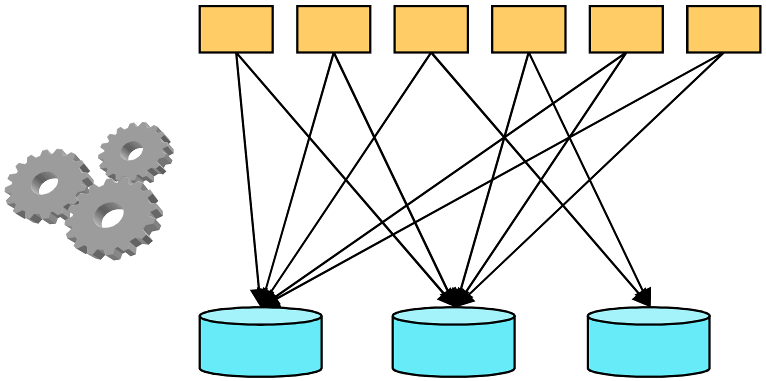
- central management of huge storage amounts
- unified multi protocol access (block, file, object)
- seamless introduction of new storage

Conventional data placement

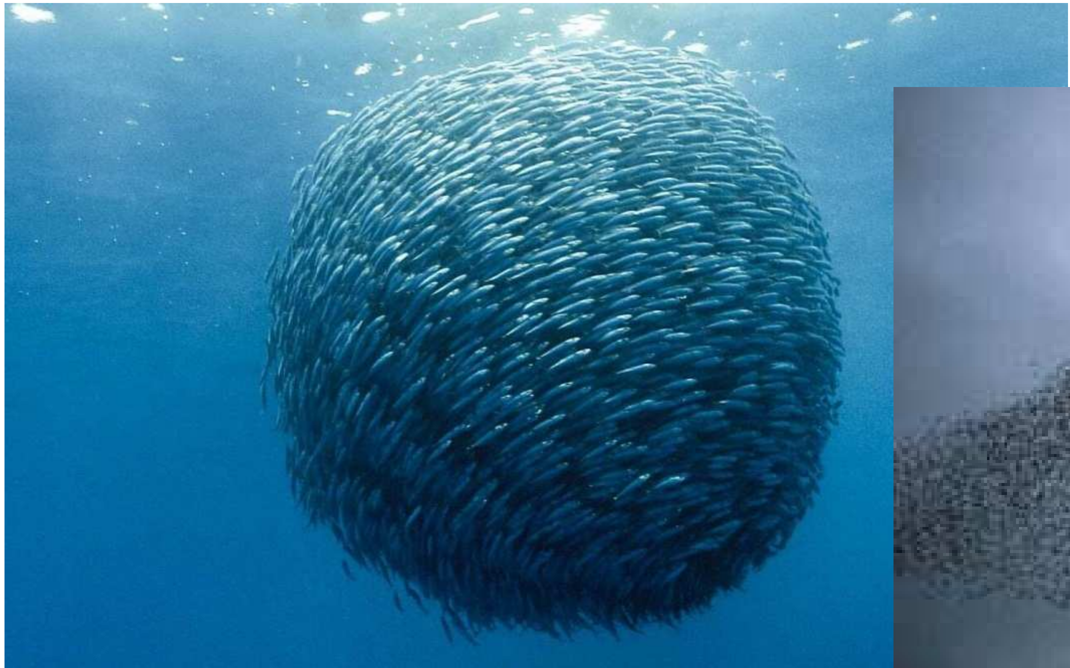
- Central allocation tables
  - File systems
- – Access requires lookup
- – Hard to scale table size
- + Stable mapping
- + Expansion trivial



- Hash functions
  - Web caching, Storage Virtualization
- + Calculate location
- + No tables
- – Unstable mapping
- – Expansion reshuffles



A model for dynamic “clouds” in nature



Swarm of birds or fishes
Source: wikipedia

Distributed intelligence

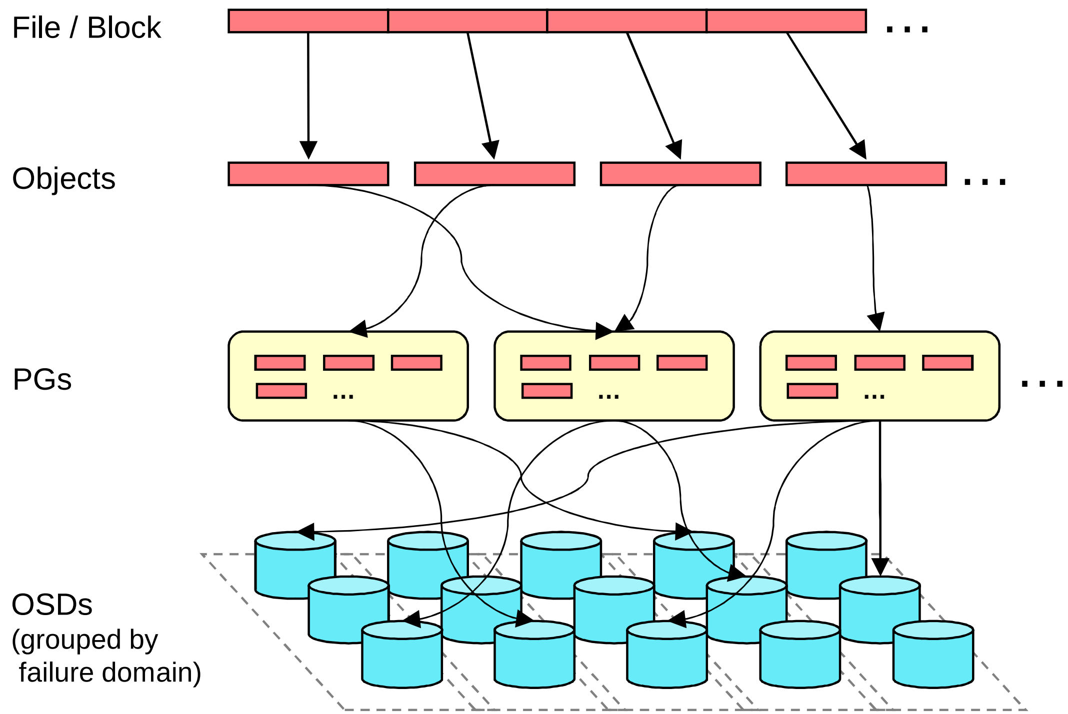
- **Swarm intelligence** [Wikipedia]
  - (SI) is the collective behavior of decentralized, self-organized systems, natural or artificial.
- **Swarm behavior** [Wikipedia]
  - Swarm behavior, or swarming, is a collective behavior exhibited by animals of similar size (…) moving en masse or migrating in some direction.
  - From a more abstract point of view, swarm behavior is the collective motion of a large number of self-propelled entities.
  - From the perspective of the mathematical modeler, it is an (…) **behavior arising from simple rules that are followed by individuals and does not involve any central coordination.**

Core Technology: CRUSH Data Placement

- Controlled Replication Under Scalable Hashing **(CRUSH)**
- **Metadata computed instead of stored**
  - almost no central lookups
- **No hot spots**
  - pseudo-random, uniform (weighted) distribution
- **Dynamic adaption to infrastructure changes**
  - adding devices has no significant impact on data mapping
- **Infrastructure aware algorithm**
  - Placement based on physical infrastructure
  - e.g., devices, servers, cabinets, rows, DCs, etc.
- **Easy and flexible placement rules**
  - ”three replicas, different cabinets, same row”
- **Quickly adjusts to failures**
  - Automatic and fast recovery from lost disks

Data placement with CRUSH

- Files/bdevs striped over objects
  - 4 MB objects by default
- Objects mapped to *placement groups (PGs)*
  - pgid = hash(object) & mask
- PGs mapped to sets of OSDs
  - crush(cluster, rule, pgid) = [osd2, osd3]
  - Pseudo-random, statistically uniform distribution
  - ~64 PGs per OSD



- **Fast:** O(log n) calculation, no lookups
- **Reliable:** replicas span failure domains
- **Stable:** adding/removing OSDs moves few PGs

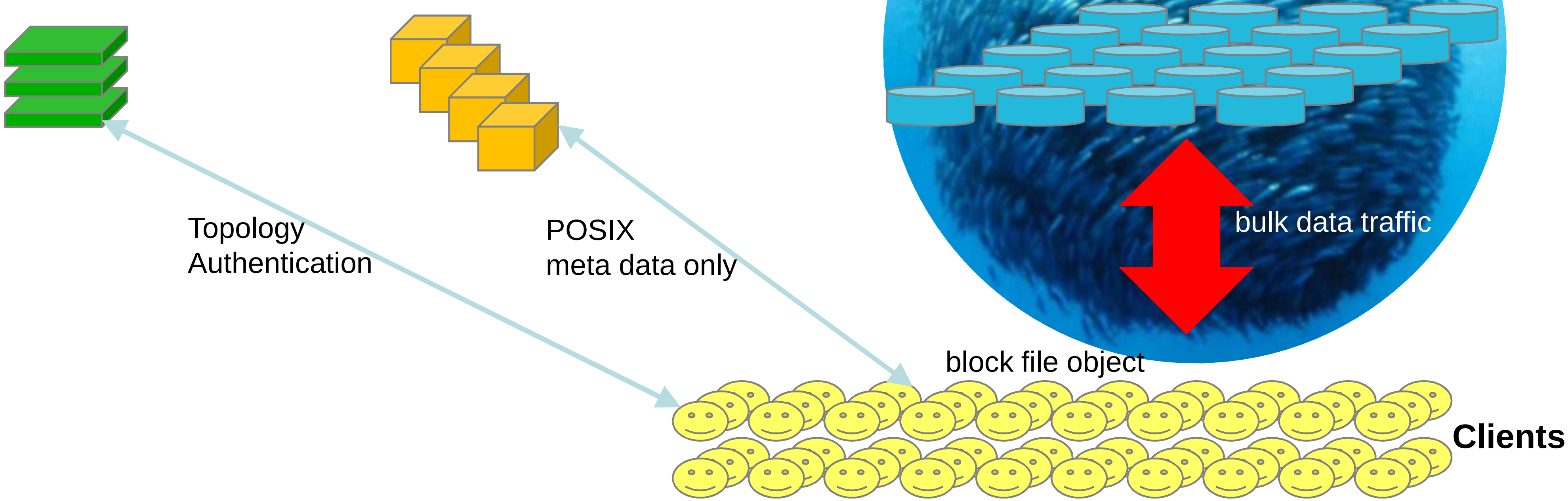
- A deterministic pseudo-random hash like function that distributes data uniformly among OSDs
- Relies on compact cluster description for new storage target w/o consulting a central allocator

Ceph Software Architecture

Cluster Monitors
<10
cluster membership
authentication
cluster state
cluster map

Meta Data Server (MDS)
10s
for POSIX only
Namespace mgmt.
Metadata ops (open, stat, rename, …)

Object Storage Daemons (OSD)
10,000s
stores all data / metadata
organises all data in flexibly sized containers



Overcome traditional challenges of rapidly growing and dynamically changing storage environments:

App

App
Object

Host / VM
Virtual Disk

Client
Files & Dirs

Librados
A library allowing apps to directly access RADOS, with support for C, C++, Java, Python, Ruby, and PHP

Ceph Object Gateway (RGW)
A bucket-based REST gateway, compatible with S3 and Swift

Ceph Block Device (RBD)
A reliable and fully-distributed block device, with a Linux kernel client and a QEMU/KVM driver

Ceph File System (CephFS)
A POSIX-compliant distributed file system, with a Linux kernel client and support for FUSE

Ceph Storage Cluster (RADOS)
A reliable, autonomous, distributed object store comprised of self-healing, self-managing, intelligent storage nodes

The Ceph difference

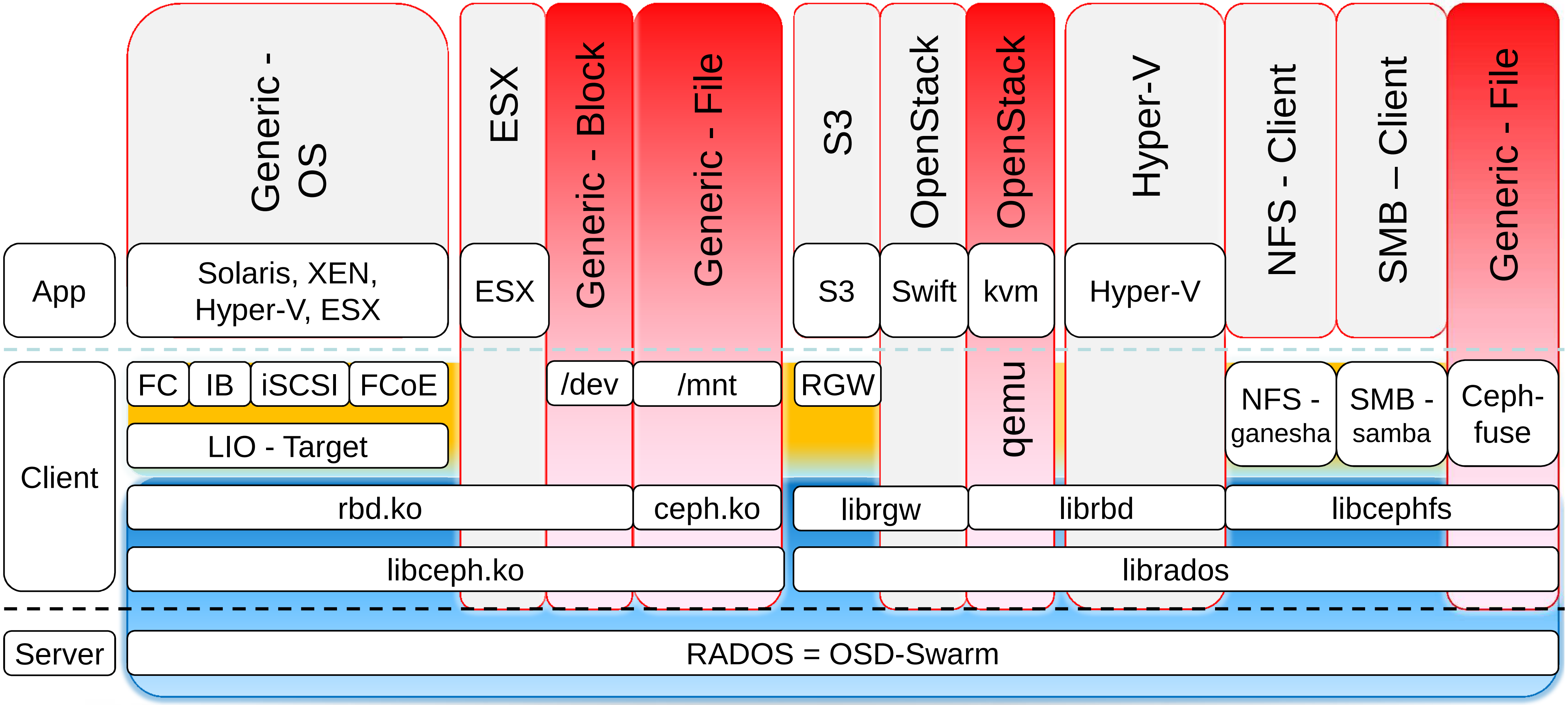
ceph

Ceph's CRUSH Algorithm liberates storage clusters from the scalability and performance limitations imposed by centralized data table mapping. It replicates and re-balance data within the cluster dynamically - eliminating this tedious task for administrators, while delivering high-performance and infinite scalability.

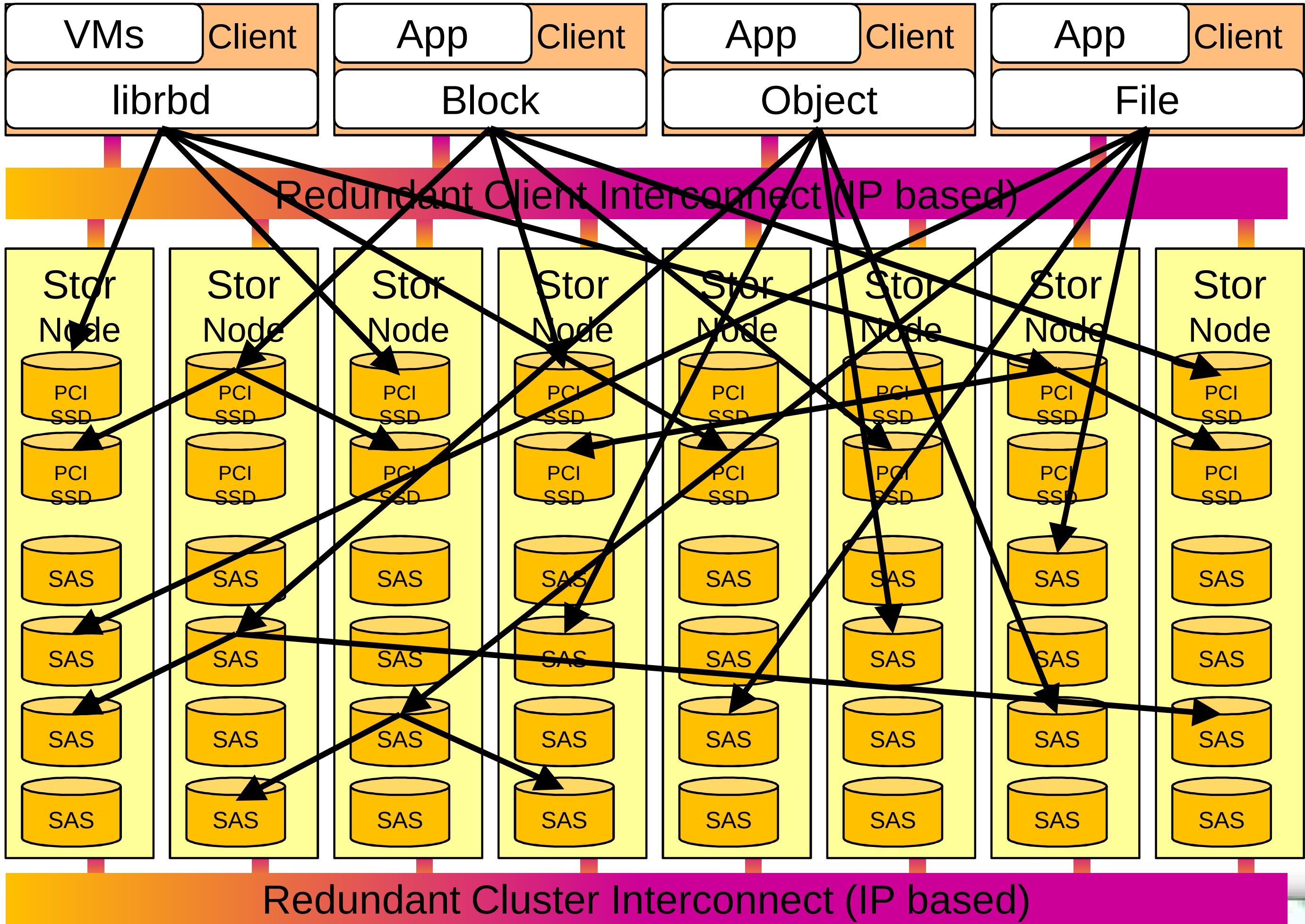
http://ceph.com/ceph-storage

Ceph is the most comprehensive implementation of Unified Storage

Ceph Front-End Interfaces



Ceph principles



Distributed Redundant Storage

- Intelligent data Distribution across all nodes and spindles = wide striping (64KB – 16MB)
- Redundancy with replica=2, 3 … 8
- Thin provisioning
- Fast distributed rebuild
- Availability, Fault tolerance
  - Disk, Node, Interconnect
  - Automatic rebuild
  - Distributed HotSpare Space
- Transparent Block, File access
- Reliability and Consistency
- Scalable Performance
- Pure PCIe-SSD for extreme Transaction processing

Agenda

- Introduction & Motivation
- **Hardware & System Software**
- Software preparation & analysis
- Performance test cube
- Conclusion

New Intel PCIe based NVMe SSD Device

- A Family of 2.5" SFF/AIC PCIe SSDs
  - NVMe compliant
  - 200GB – 2000GB
  - Up to 25W Active Power

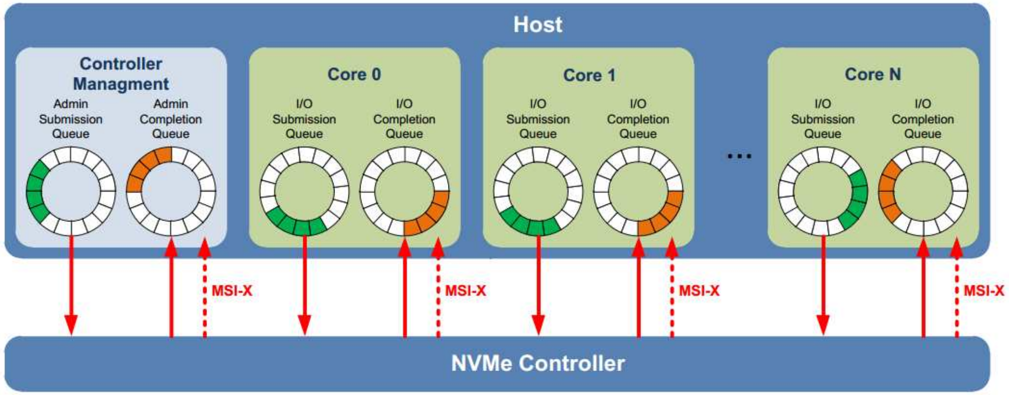
SKU Size	Random Read 4K IOPS	Random Write 4K IOPS	Seq. Read BW 64K Ops	Seq. Write BW 64K Ops	Max Power
200GB	300K	35K	1,400MB/s	400MB/s	15W
400GB	400K	75K	2,700MB/s	900MB/s	20W
800GB	450K	150K	2,800MB/s	1,700MB/s	25W
1,600GB	450k	150K	2,800MB/s	1,700MB/s	25W
2,000GB	450K	150K	2,800MB/s	1,700MB/s	25W

Perfect fit to host Journals

- 20nm HE MLC NAND
  - High Endurance
  - 10 Drive Writes/Day
- Fully Integrated *NVM Express* Controller
  - Power Fail Write Cache
  - End-to-End Datapath Protection
  - Endurance Management
  - Power and Thermal Throttling

```
INTEL SSDPEDMD800G4  CVFT40300057800CGN  8DV10036  /dev/nvme0
INTEL SSDPEDMD800G4  CVFT4030006F800CGN  8DV10036  /dev/nvme1
[root@rx38-0 ~]# ls -l /dev/nvm*
crw-rw---- 1 root root  10, 59 Mar 25 14:16 /dev/nvme0
brw-rw---- 1 root disk 252,  0 Mar 31 20:17 /dev/nvme0n1
brw-rw---- 1 root disk 252,  1 Mar 31 20:17 /dev/nvme0n1p1
crw-rw---- 1 root root  10, 58 Mar 25 14:16 /dev/nvme1
```

NVMe Driver Basics



Queue Allocation

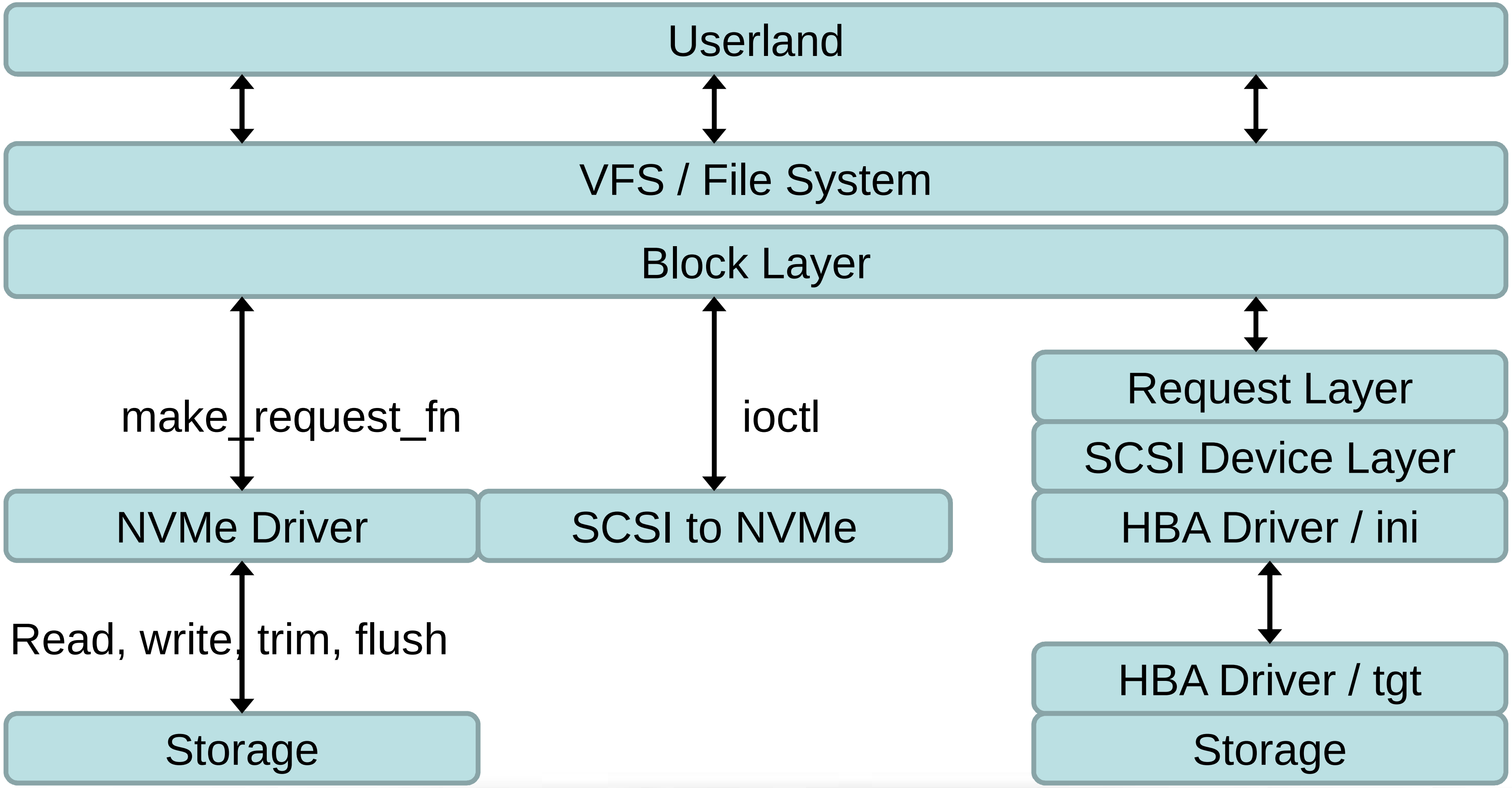
- Ideal case: one SQ/CQ pair per cpu core
- MSI-x IRQ affinity assigned to CPU associated Queue

All parameters for 4KB command in single 64B command

- Supports deep queues (64K commands per queue, up to 64K queues)
- Supports MSI-X and interrupt steering
- Streamlined & simple command set (13 required commands)
- Optional features to address target segment (Client, Enterprise, etc.)
    - Enterprise: End-to-end data protection, reservations, etc.
    - Client: Autonomous power state transitions, etc.
- Designed to scale for next generation NVM, agnostic to NVM type used

http://www.flashmemorysummit.com/English/Conference/Proceedings_Chrono.html, Keith Bush (Intel)

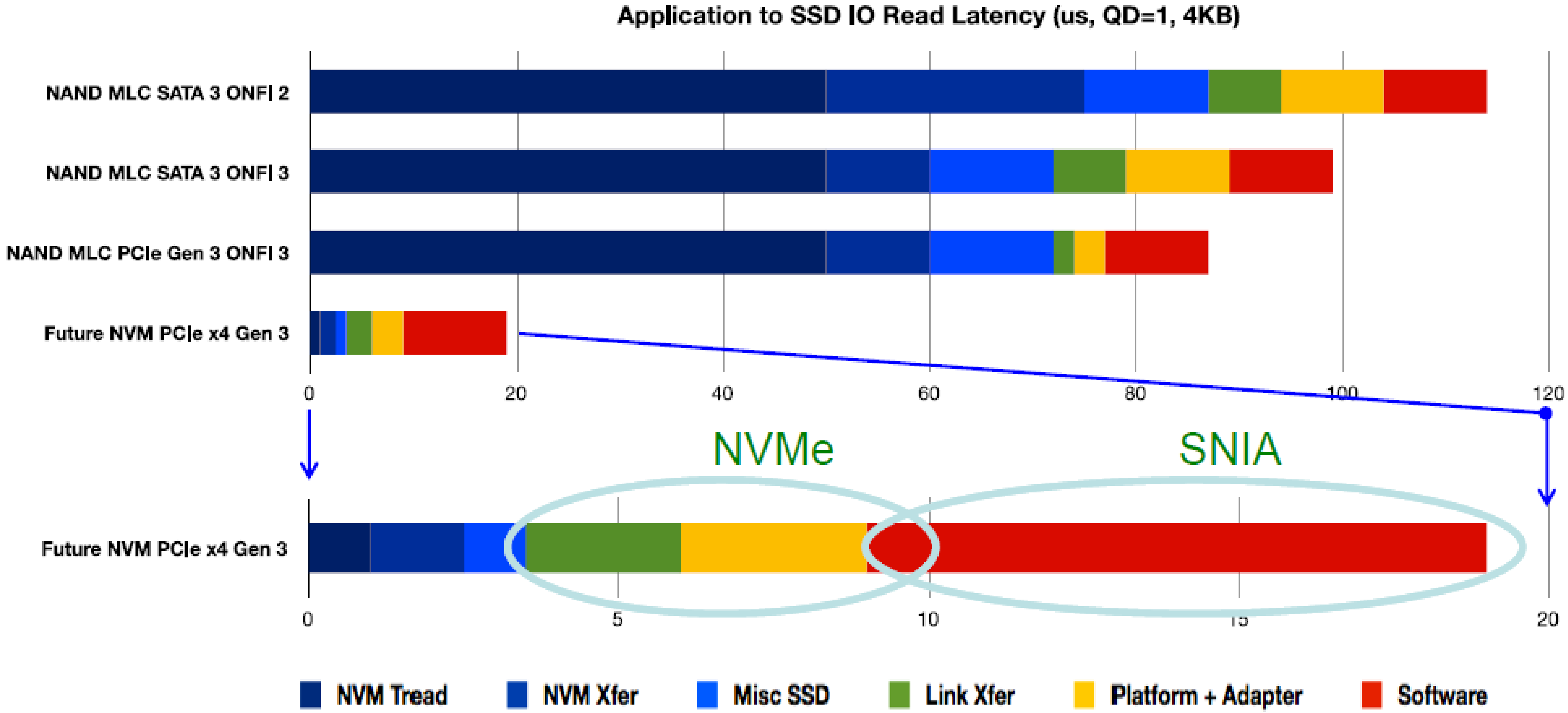
NVMe Driver Stack vs. SCSI



NVMe Driver: Feature history & roadmap

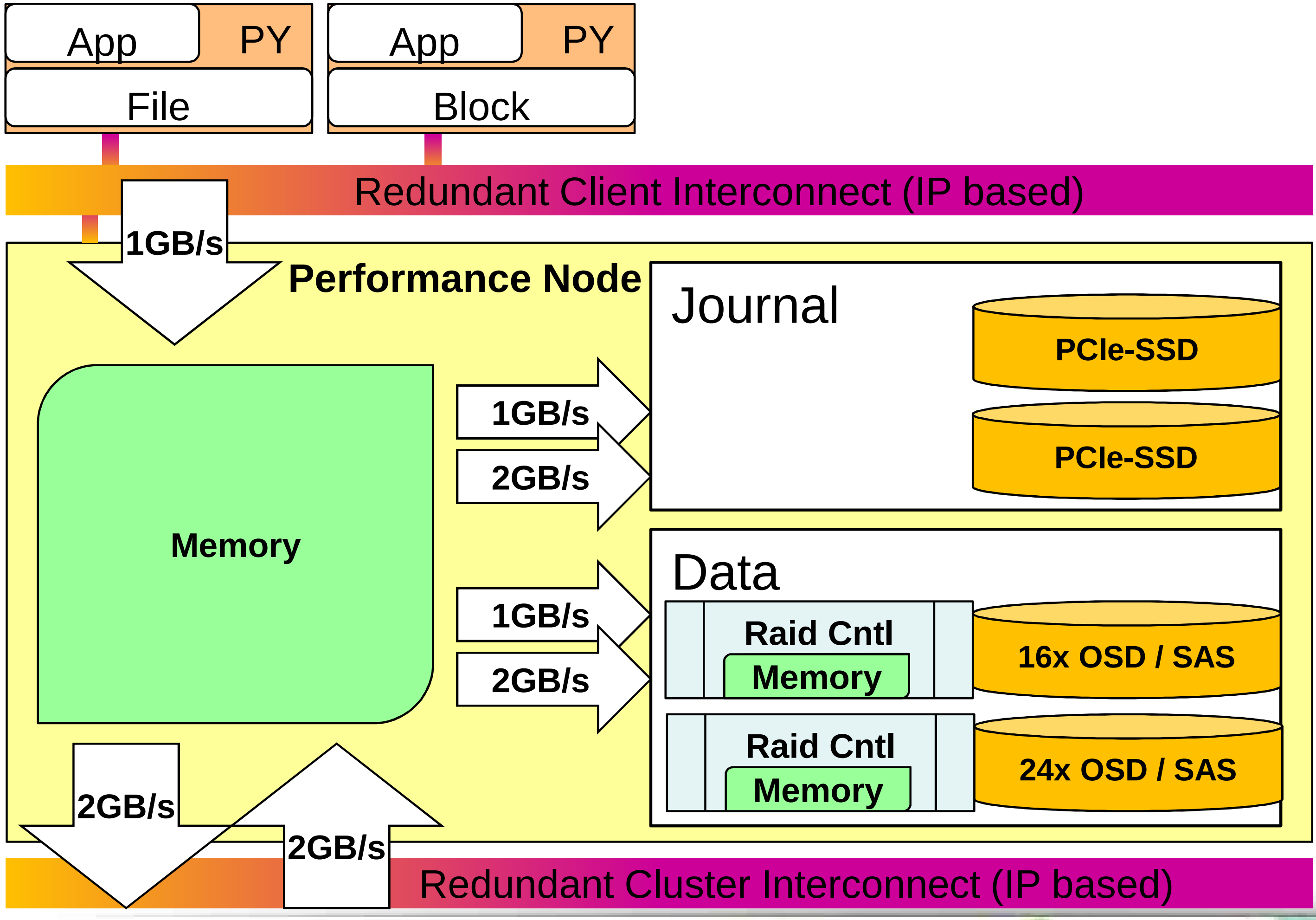
Kernel	Features (source: Keith Bush, Intel)
3.3	• Initial commit based on NVMe 1.0c
3.6	• Greater than 512 byte block support
3.9	• Discard/TRIM (NVMe Data-Set Mgmt) • SG_IO SCSI-to-NVMe translation
3.10	• Multiple Message MSI • Disk stats / iostat
3.12	• Power Management: Suspend/Resume
3.14	• Dynamic Partitions • Surprise Removal, no I/O
3.16	• Flush, Trace points, Function Level Reset Notify
3.17+	• Block Multi-Queue, Page IO, CRC T10 DIF/DIX

Future outlook for Post-NAND



http://www.flashmemorysummit.com/English/Conference/Proceedings_Chrono.html, Jim Pappas (Intel)

Optimized IO balance for replica = 3



(1) Balanced I/O architecture

(2) 1 GB throughput including all redundant data copies

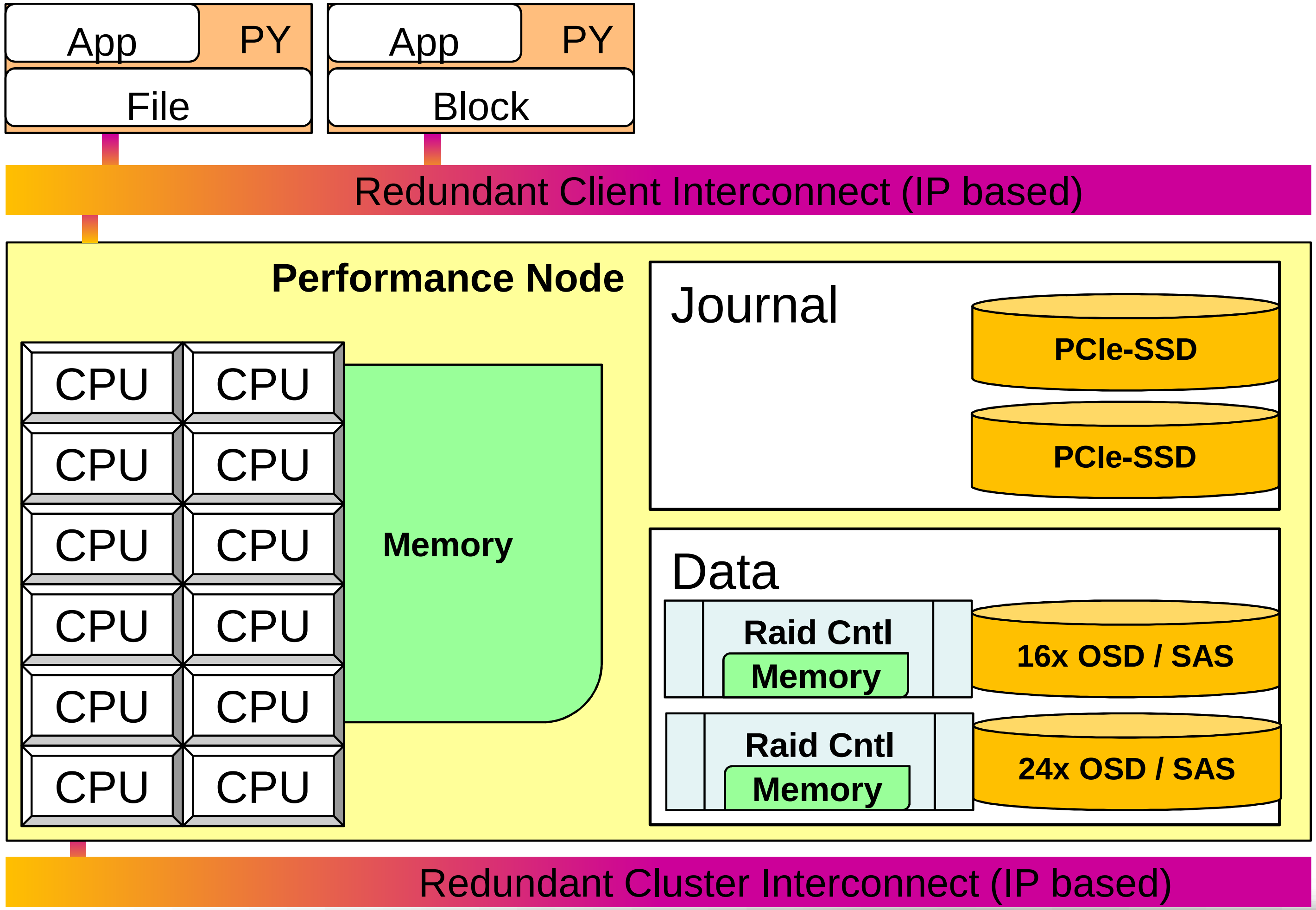
(3) Strong Infiniband backend (replicas, rebalancing, recovery)

(4) Journal on PCIe-SSDs (2x 1.7 GB/s bw)

(5) Data on SAS-6G HDDs thru LSI-2208 MegaRAID (2x 1.8 GB/s bw)

(6) Performance data to be confirmed

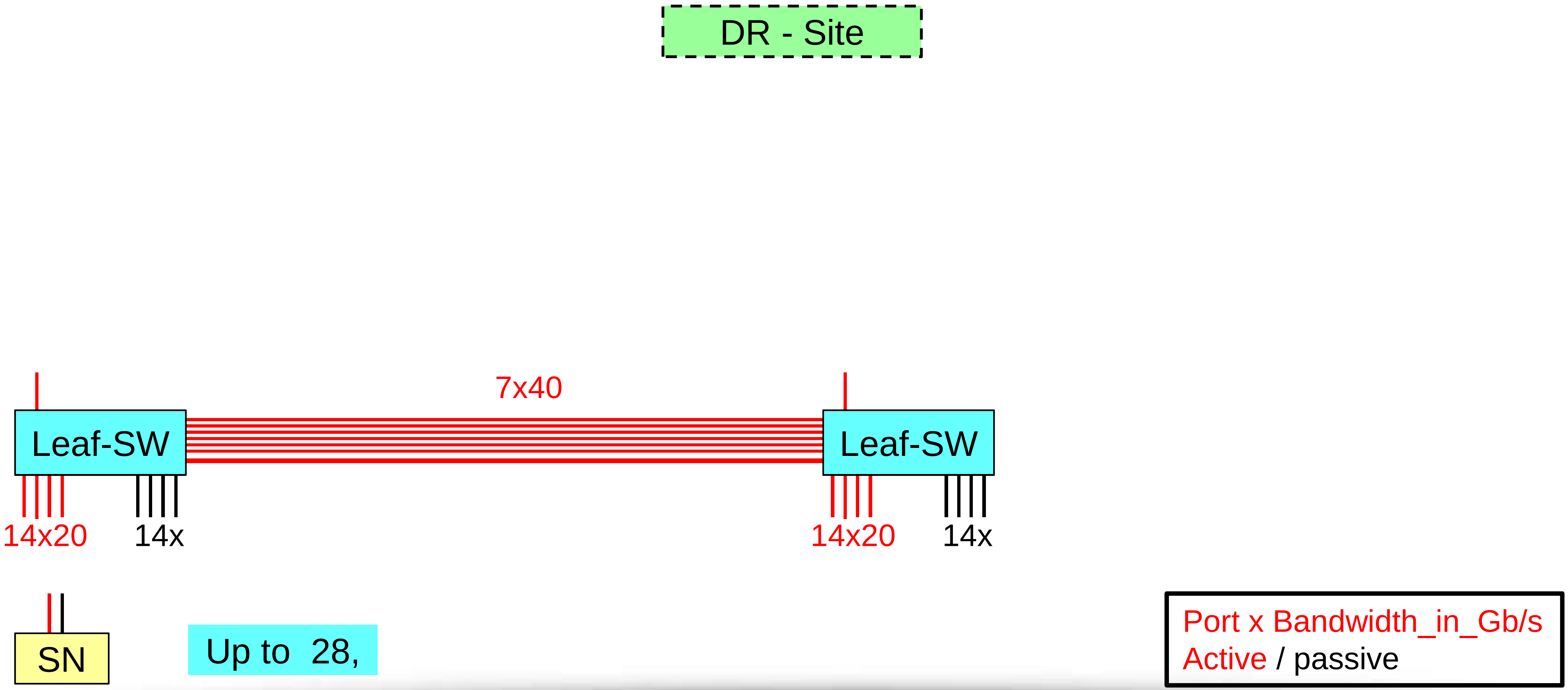
Consider balanced components CPU vs IO



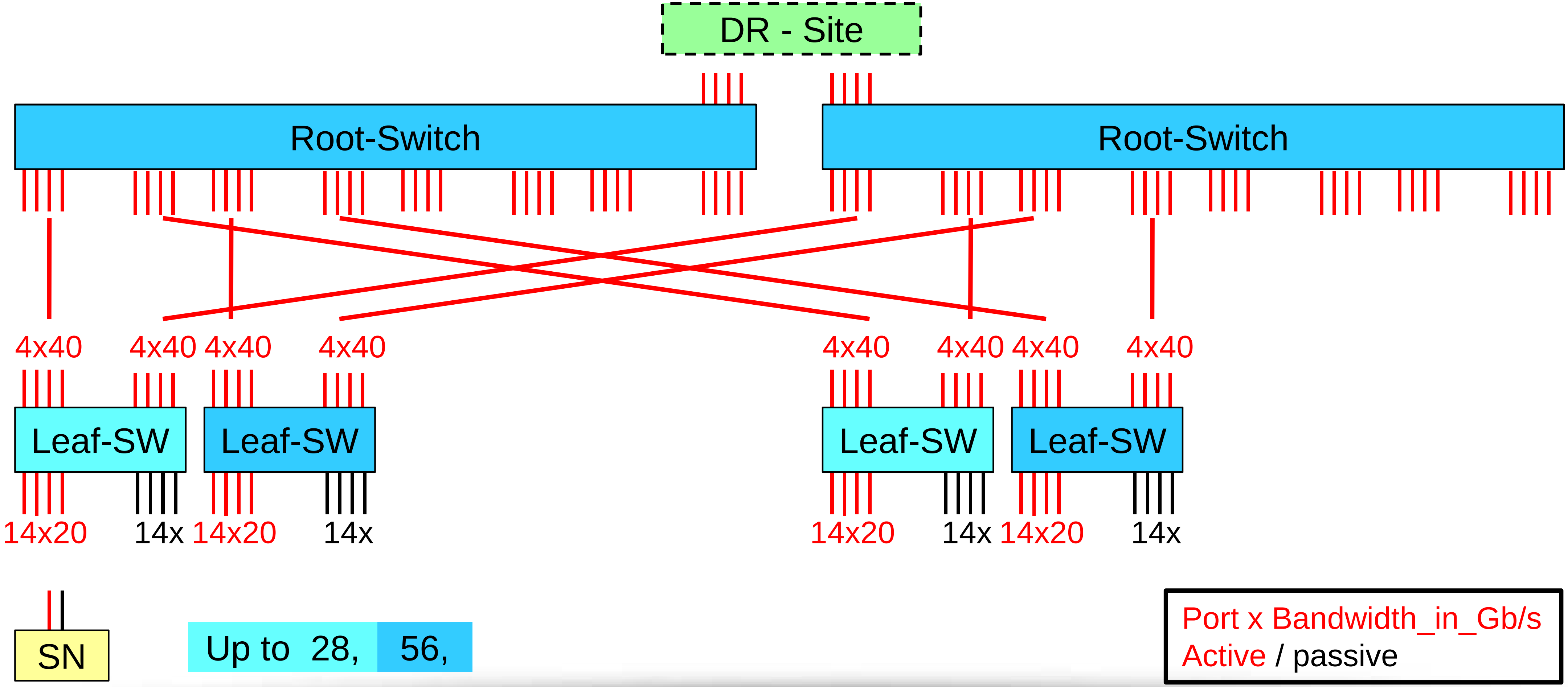
(1) IP-Stack Front-End
(2) IP-Stack Back-End
(3) Software CRC inside
(4) Software compression
(5) Software Erasure-Code
(6) Software Auto-Tiering
(7) Core Ceph functionality

Rule of thumb: 1GHz per OSD

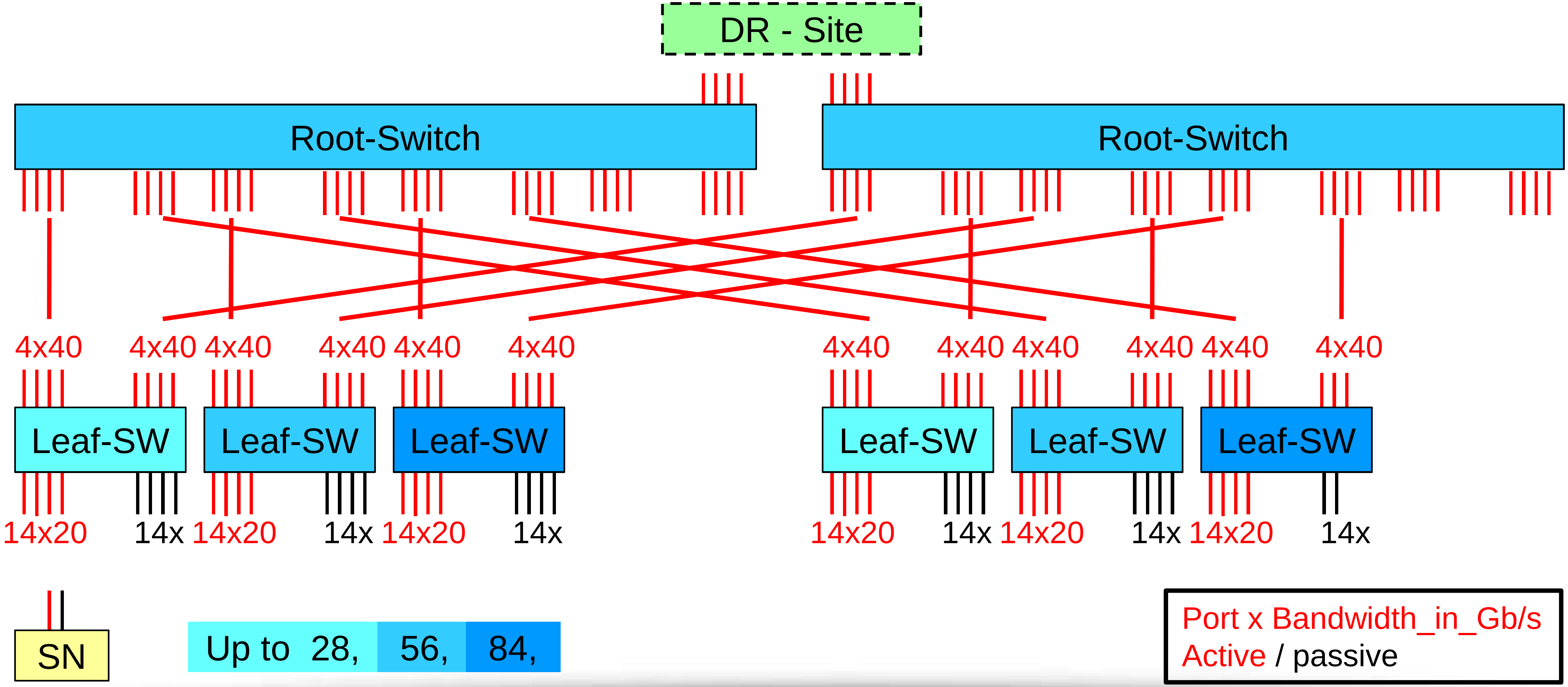
Up to 28 Storage Nodes



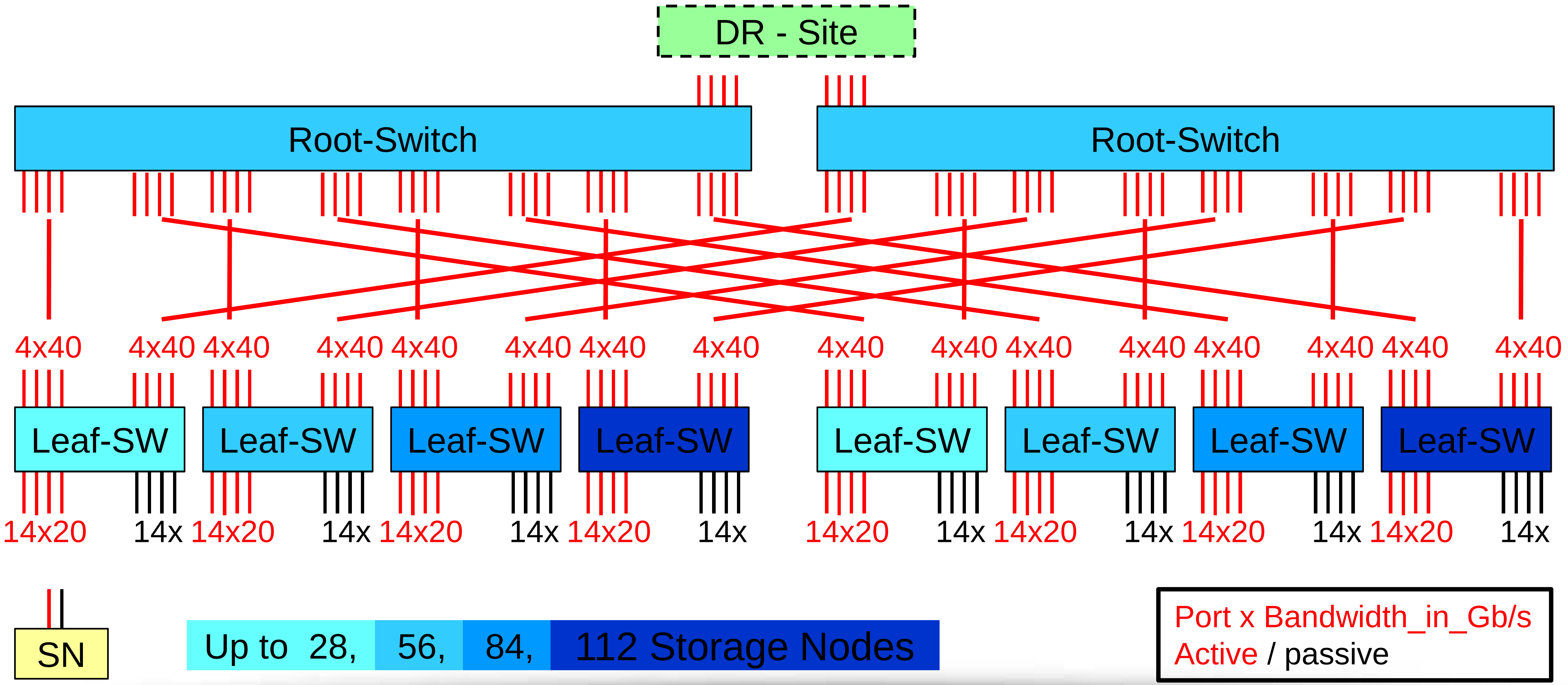
Up to 56 Storage Nodes



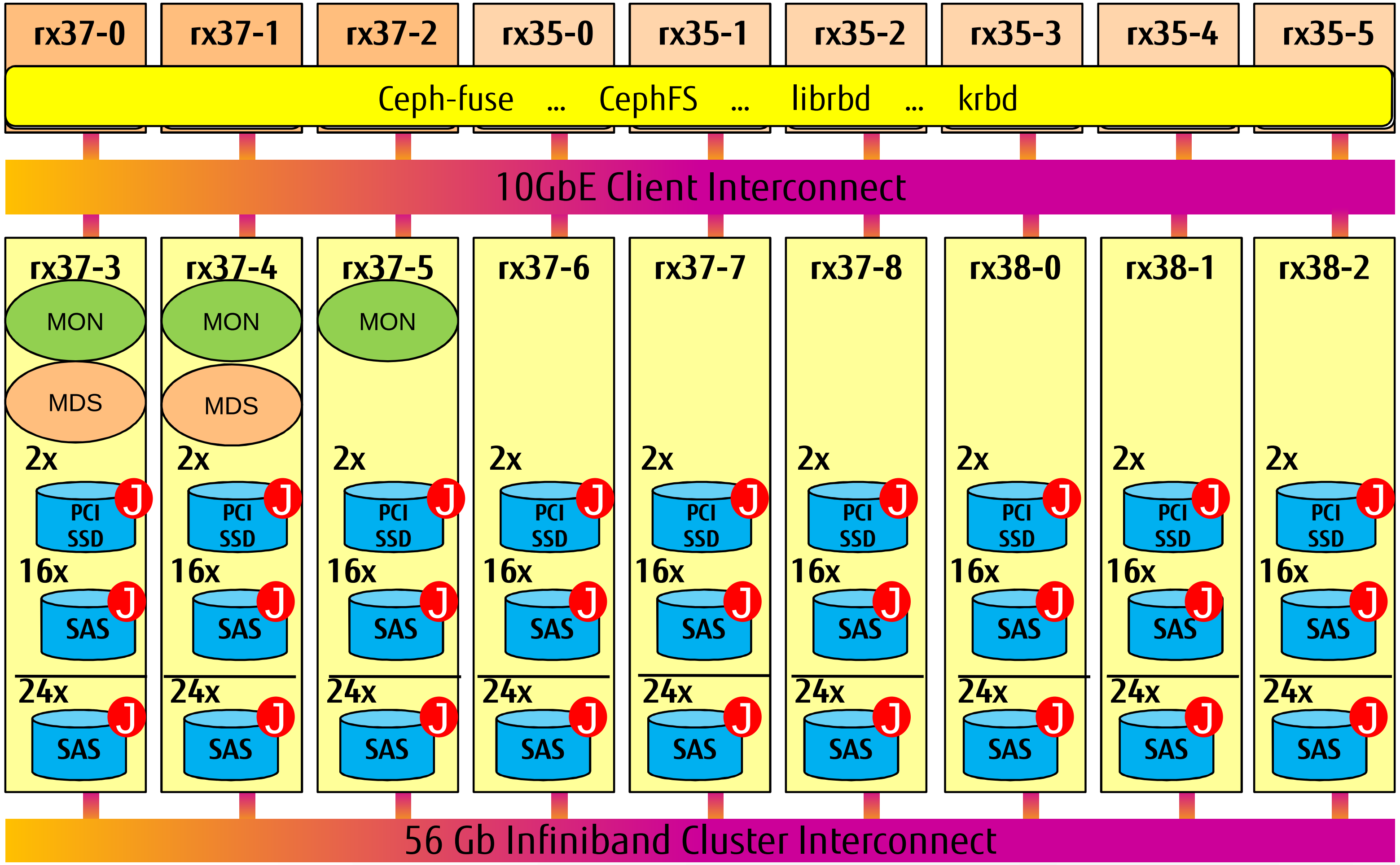
Up to 84 Storage Nodes



Up to 112 Storage Nodes



HW of the Performance test cluster



Client

- rx37: 1x E5-2630 / 6C, 2.30GHz, 64GB
- rx35: 2x E5540 / 4C, 2.53GHz, 48GB

Server

- rx37: 2x E5-2630 / 6C, 2.30GHz, 128GB MDS-Nodes with 192GB
- rx38: 2x E5-2640 v2 / 8C, 2.00GHz, 128GB
- 2x Intel P3700, 800GB
- 16x SAS-6G, 300GB inside
- 24x SAS-6G, 300GB in JX40
- 2x 10GbE Client connect
- 2x 56Gb IPoIB Cluster backbone

Performance prerequisites

	Write 4k		Read 4k	
	Rand	Seq	Rand	Seq
Intel NVMe	150k	150k	350k	350k
LSI-RAID HDD	600	85k	350	9800
10 GbE tcp_bw	290k			
56 IPoIB tcp_bw	320k			

In total

- **NVM-SSD** **18x 150k = 2700k** **… repli=2 1300k, r=3 900k**
- **LSI SAS-HDD** **357x 600 = 214k** **… repli=2 107k, r=3 70k**
- **10 GbE** **9x 290k = 2600k**
- **56 IPoIB** **9x 320k = 2900k** **… repli=2 2900k, r=3 1400k**

1st Performance test case

- 9x Clients
- 9x Storage Nodes
- 9x fio jobs in total
- 357x SAS-OSDs
- 18x SSD-OSDs

```
fio --client $1 $1.fio \
    --client $2 $2.fio \
(…) --client $9 $9.fio \
    --output=fiowrite_4k_32
```

```
[global]
filename=/dev/rbd0
direct=1
name=file1
runtime=60
group_reporting

[file]
description=write-4k-32-0
size=32G
offset_increment=32G
rw=write
bs=4k
numjobs=32
```

1st Performance observations

10k write IOPS on 4k

Agenda

- Introduction & Motivation
- Hardware & System Software
- **Software preparation & analysis**
- Performance test cube
- Conclusion

Disabling Ceph trace

… will increase Performance on small I/Os by 30%

```
debug default          = 0/0        debug osd            = 0/0
debug lockdep          = 0/0        debug optracker      = 0/0
debug context          = 0/0        debug objclass       = 0/0
debug crush            = 0/0        debug filestore      = 0/0
debug mds              = 0/0        debug journal        = 0/0
debug mds balancer     = 0/0        debug ms             = 0/0
debug mds locker       = 0/0        debug mon            = 0/0
debug mds log          = 0/0        debug monc           = 0/0
debug mds log expire   = 0/0        debug paxos          = 0/0
debug mds migrator     = 0/0        debug tp             = 0/0
debug buffer           = 0/0        debug auth           = 0/0
debug timer            = 0/0        debug finisher       = 0/0
debug filer            = 0/0        debug heartbeatmap   = 0/0
debug objecter         = 0/0        debug perfcounter    = 0/0
debug rados            = 0/0        debug rgw            = 0/0
debug rbd              = 0/0        debug hadoop         = 0/0
debug journaler        = 0/0        debug javaclient     = 0/0
debug objectcacher     = 0/0        debug asok           = 0/0
debug client           = 0/0        debug throttle       = 0/0
```

Better monitor of time consumption

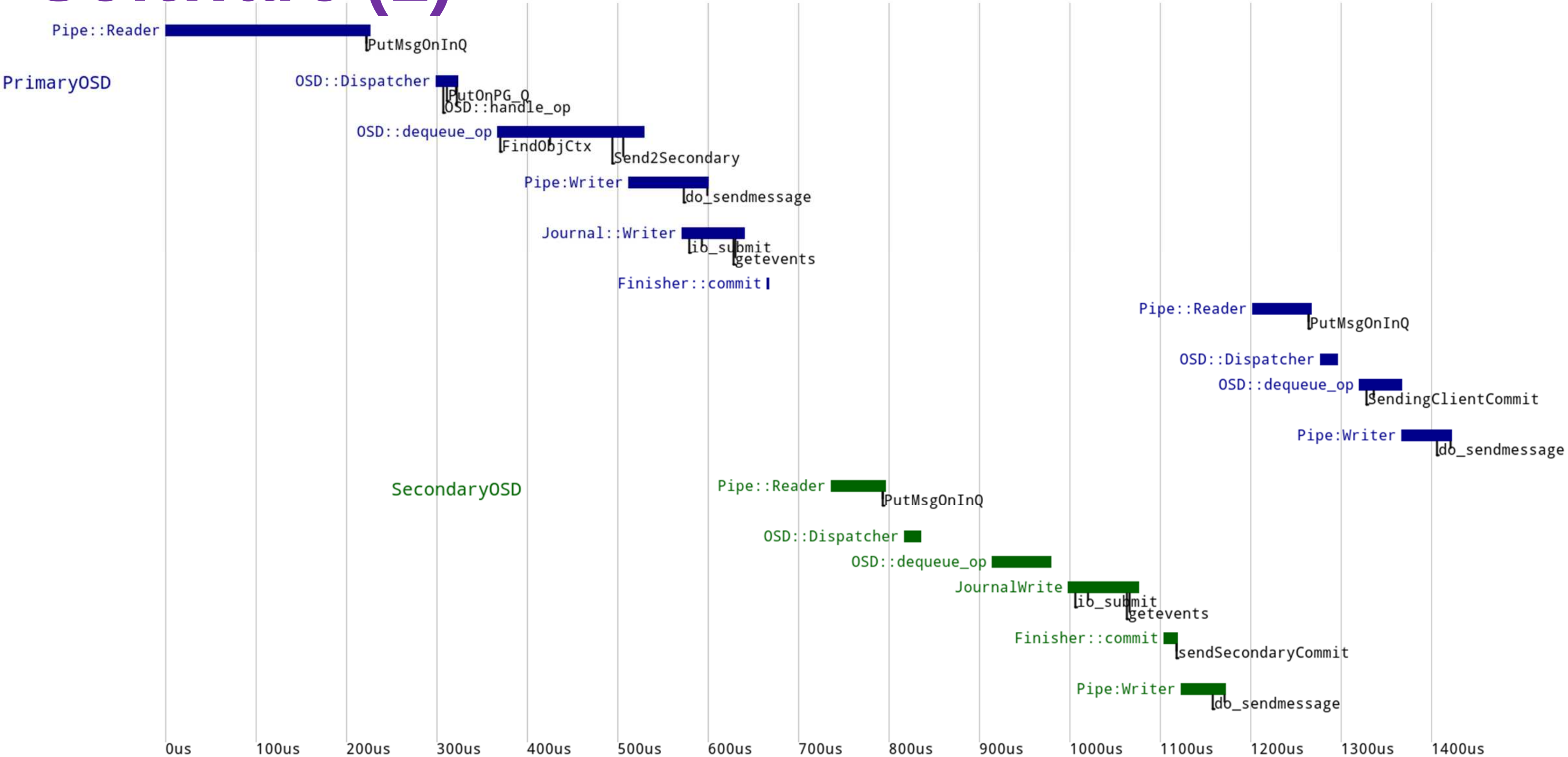
- Using the Ceph internal timestamps has an high impact on the performance

Concept

- Timestamps where introduced into the ceph-osd code specifically
- Primary target of the timestamp: to collect information about read and write
- … plus collect information about a complete operation
- collecting timing information should have minimal impact on performance (do it in memory only, use the osd-tell interface to export the data)
- of the affected process (ceph-osd daemon).

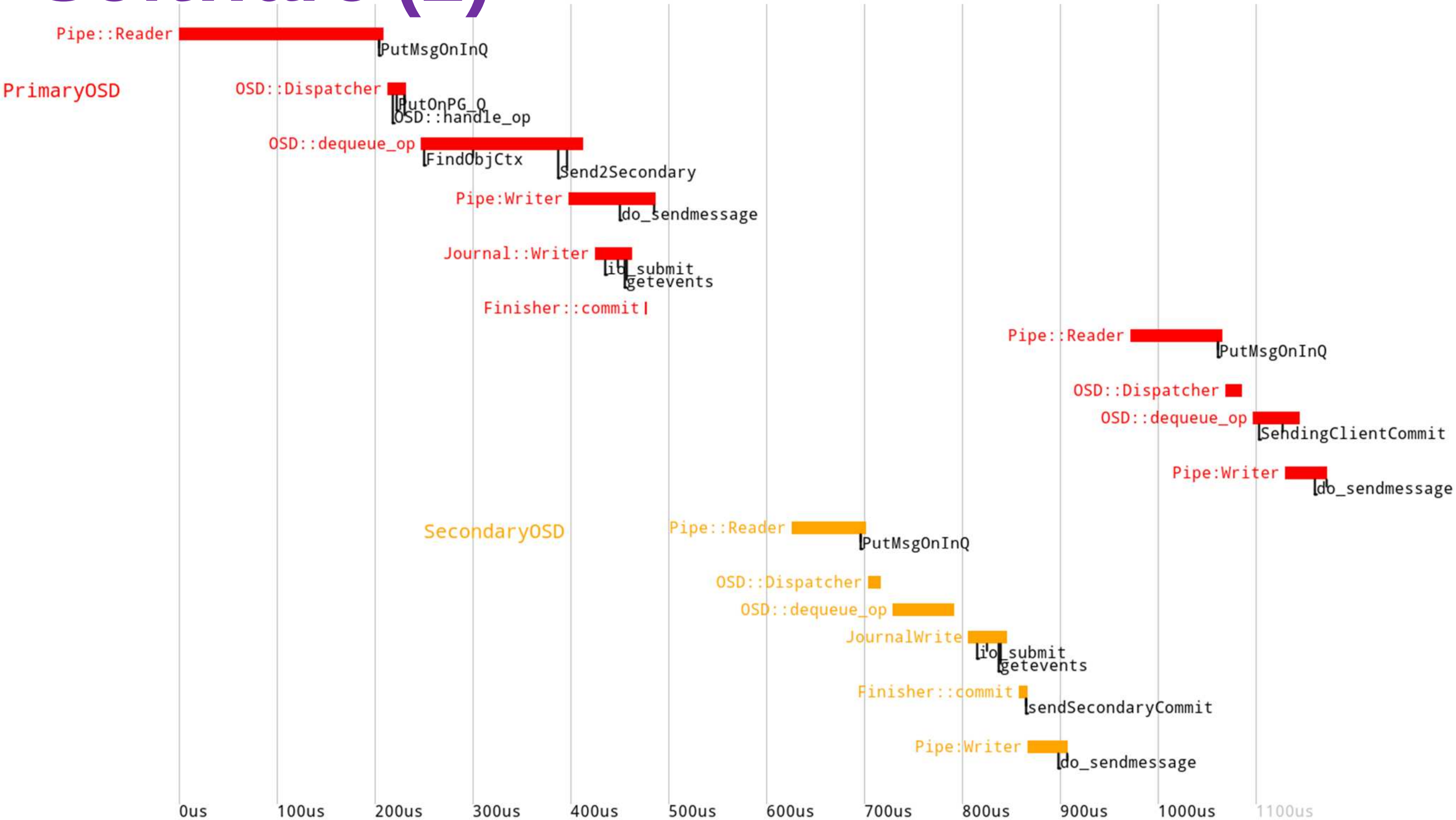
- Therefore: timestamps are collected in memory and are evaluated after the end of a test via an extension to the ceph-osd "tell" interface.
- ceph-osd code was instrumented with timestamp collection at selected places.
- Cons: in a new ceph version, a manual placement of timestamps is needed

Software (1)



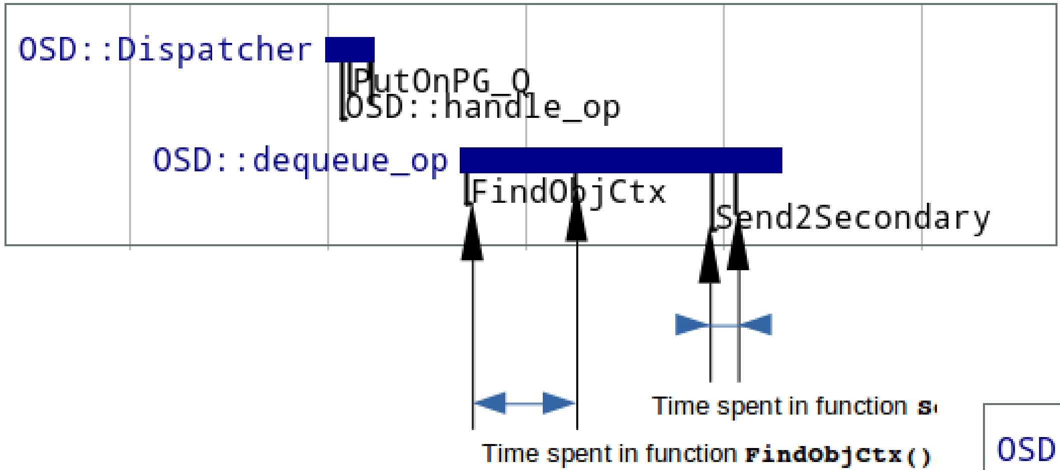
Pipe::Reader
PutMsgOnInQ
PrimaryOSD
OSD::Dispatcher
PutOnPG_Q
OSD::handle_op
OSD::dequeue_op
FindObjCtx
Send2Secondary
Pipe:Writer
do_sendmessage
Journal::Writer
io_submit
getevents
Finisher::commit
Pipe::Reader
PutMsgOnInQ
OSD::Dispatcher
OSD::dequeue_op
SendingClientCommit
Pipe:Writer
do_sendmessage
SecondaryOSD
Pipe::Reader
PutMsgOnInQ
OSD::Dispatcher
OSD::dequeue_op
JournalWrite
io_submit
getevents
Finisher::commit
sendSecondaryCommit
Pipe:Writer
do_sendmessage
0us
100us
200us
300us
400us
500us
600us
700us
800us
900us
1000us
1100us
1200us
1300us
1400us

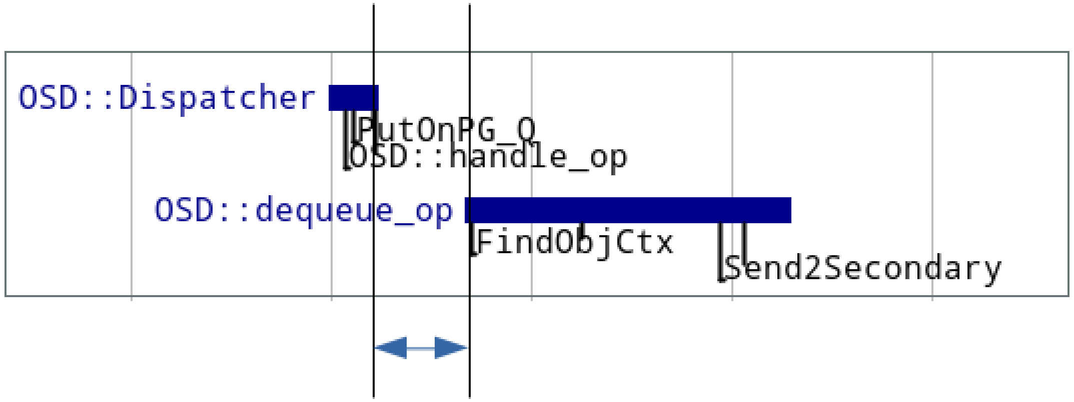
Software (2)



PrimaryOSD
Pipe::Reader
PutMsgOnInQ
OSD::Dispatcher
PutOnPG_Q
OSD::handle_op
OSD::dequeue_op
FindObjCtx
Send2Secondary
Pipe:Writer
do_sendmessage
Journal::Writer
io_submit
getevents
Finisher::commit
Pipe::Reader
PutMsgOnInQ
OSD::Dispatcher
OSD::dequeue_op
SendingClientCommit
Pipe:Writer
do_sendmessage
SecondaryOSD
Pipe::Reader
PutMsgOnInQ
OSD::Dispatcher
OSD::dequeue_op
JournalWrite
io_submit
getevents
Finisher::commit
sendSecondaryCommit
Pipe:Writer
do_sendmessage
0us
100us
200us
300us
400us
500us
600us
700us
800us
900us
1000us
1100us

Software (3)





The gap between the end of the OSD::Dispatcher and the start of OSD::dequeue_op is a result from thread switching and CPU-core migration.
The size gap was reduced significantly when booting the Linux kernel with **`idle=poll`** flag.

Software (4)

1. ceph-osd operation: request to write data

When receiving a request write data, the processing of such a request takes place in several steps:

- the request is received on the public network; the request type and contents is analyzed and checked.
- it then gets dispatched for internal processing; this step involves the location of the corresponding local data object, the replication partners.
- then the write of the journal entry is started, the write to the local data object and the transmission of the write request to the corresponding replicating osd instances is triggered
- acknowledges to these different actions are waited for and once all required acknowledges have received, then the ceph client is informed about the completion of it's original request.

** All these activities are handled by a number of processing threads, i.e. the thread model for ceph is based on the stages of processing a request.

** The communication between these threads is queue based, i.e. queues are used to transfer a request (or objects derived from it) between the processing threads.

** This model also defines a large group of places where timestamps are integrated into the ceph-osd code: the dequeue or enqueue of a request or derived message.

2. sample effect: diable energy saving modes of CPU

The two diagrams illustrate the effect of disabling the energy saving modes of Intel Xeon CPUs. These may lower the operating frequency when they encounter no processing activity. Once a task/thread gets scheduled on a core of the cpu, then processing resumes and CPU frequency is increased again. This special operation mode can be disabled on behalf of the Linux kernel when it is booted with "idle=poll" Option.

Analysis of the timestamps without and with this boot option show the time required for thread switching can be reduced signifcantly.

Adapt the NVMe Driver to our needs

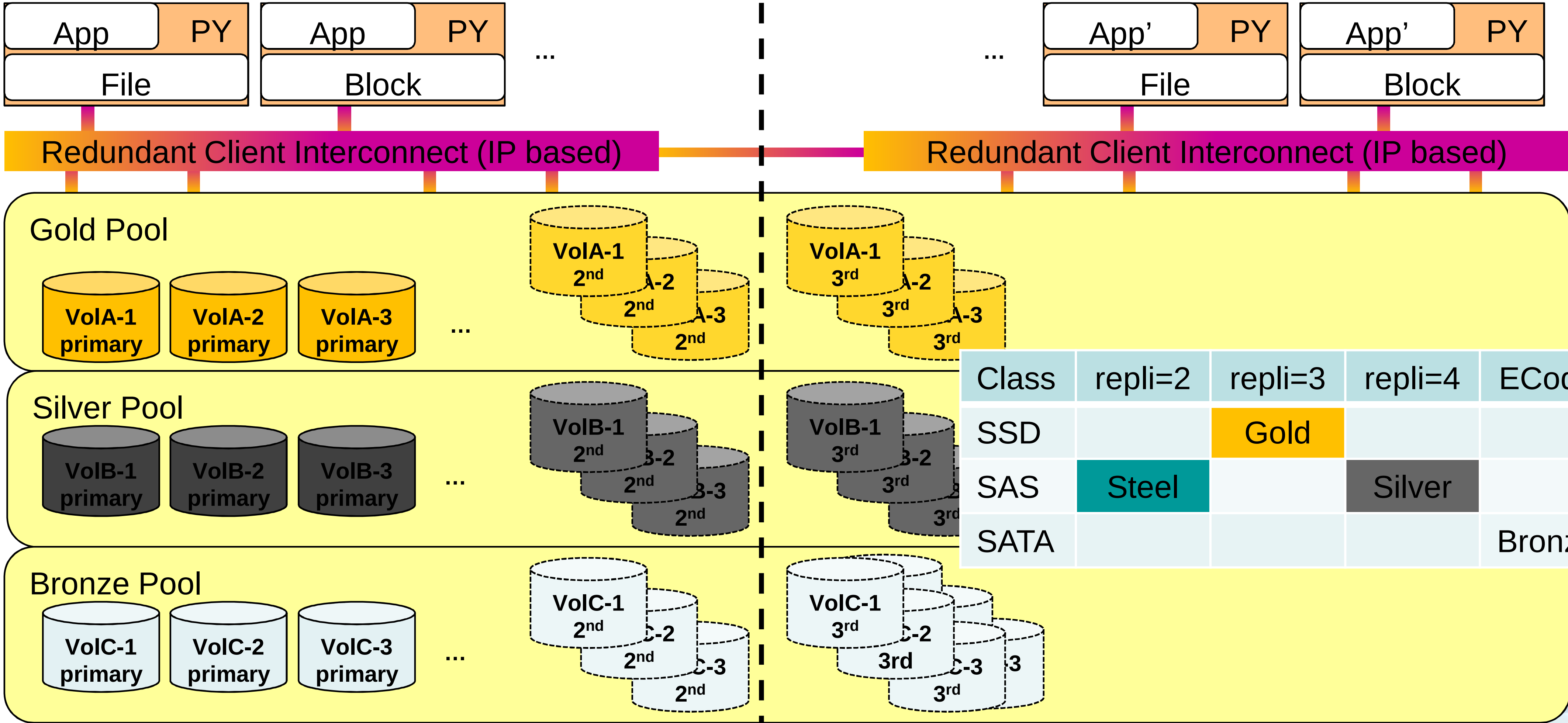
(1) IO statistics support (git://git.kernel.org/pub/scm/linux/kernel/git/stable/)

- Disk IO Statistics since kernel 3.12.1; not in 3.10.x (max: 3.10.53) 6198221fa0df0298513b35796f63f242ea97134e <keith.busch@intel.com>
- NVMe: Disk IO statistics
- Add io stats accounting for bio requests so nvme block devices show useful disk stats.
- Kernel 3.16.1: sysctl to *disable* IO statistics b4e75cbf1364c4bbce3599c3279892a55b6ede07 <sbradshaw@micron.com>
- NVMe: Adhere to request queue block accounting enable/disable
- Recently, a new sysfs control "iostats" was added to selectively enable or disable io statistics collection for request queues. (+50% on read perf in the million iops+)

(2) limitation to 64 partitions

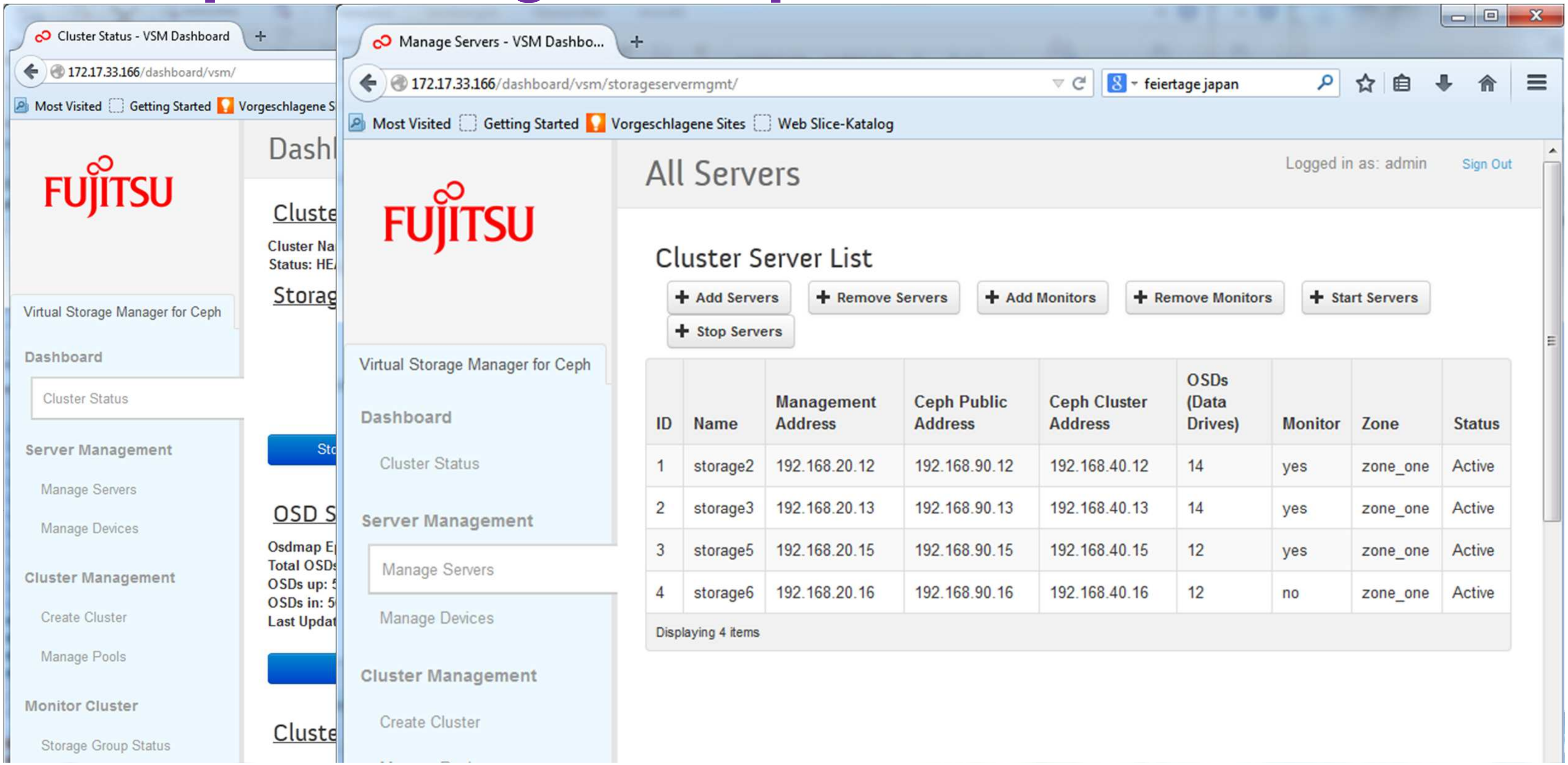
- The upper limit of 64 partitions on one NVMe Device has been overcome in linux 3.14 469071a37afc8a627b6b2ddf29db0a097d864845
- Using nvme source directly from 3.14.2 and trying to compile in a 3.10.32 environment fails because of changes in the generic block device layer of the linux kernel

HA/DR Design & different Storage pools



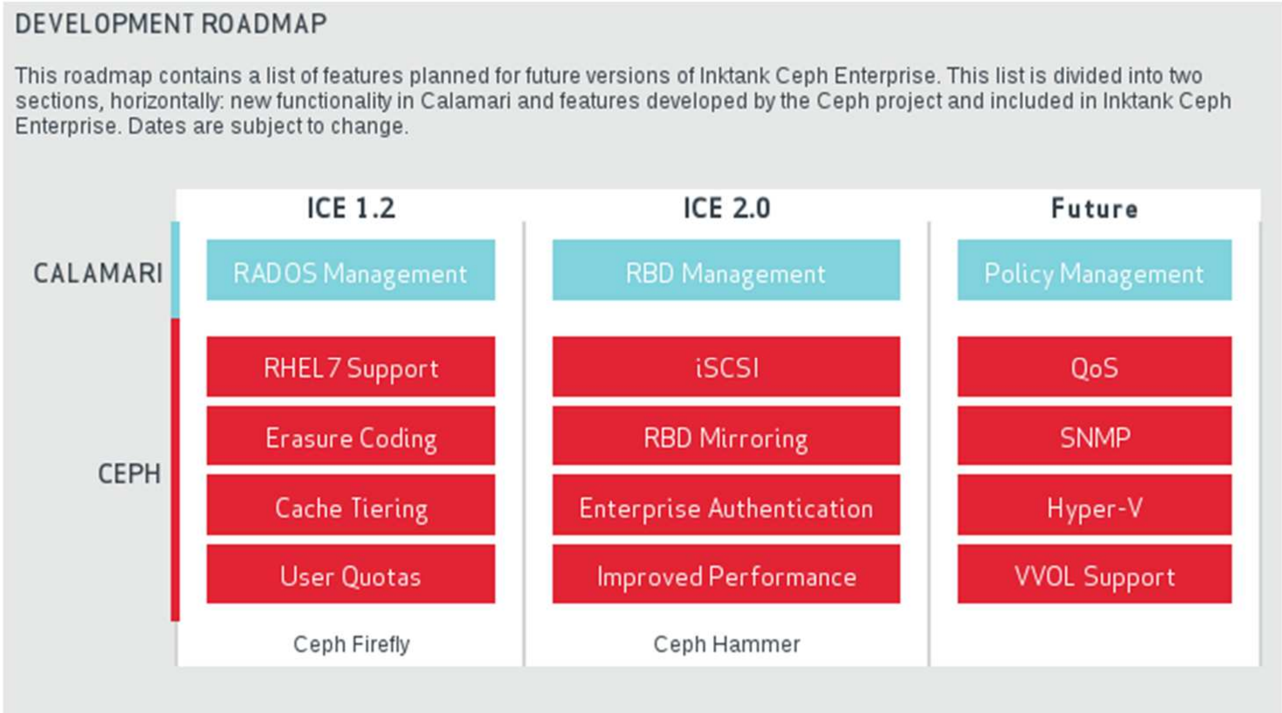
Class	repli=2	repli=3	repli=4	ECode
SSD		Gold		
SAS	Steel		Silver	
SATA				Bronze

Setup & Configure Ceph with VSM



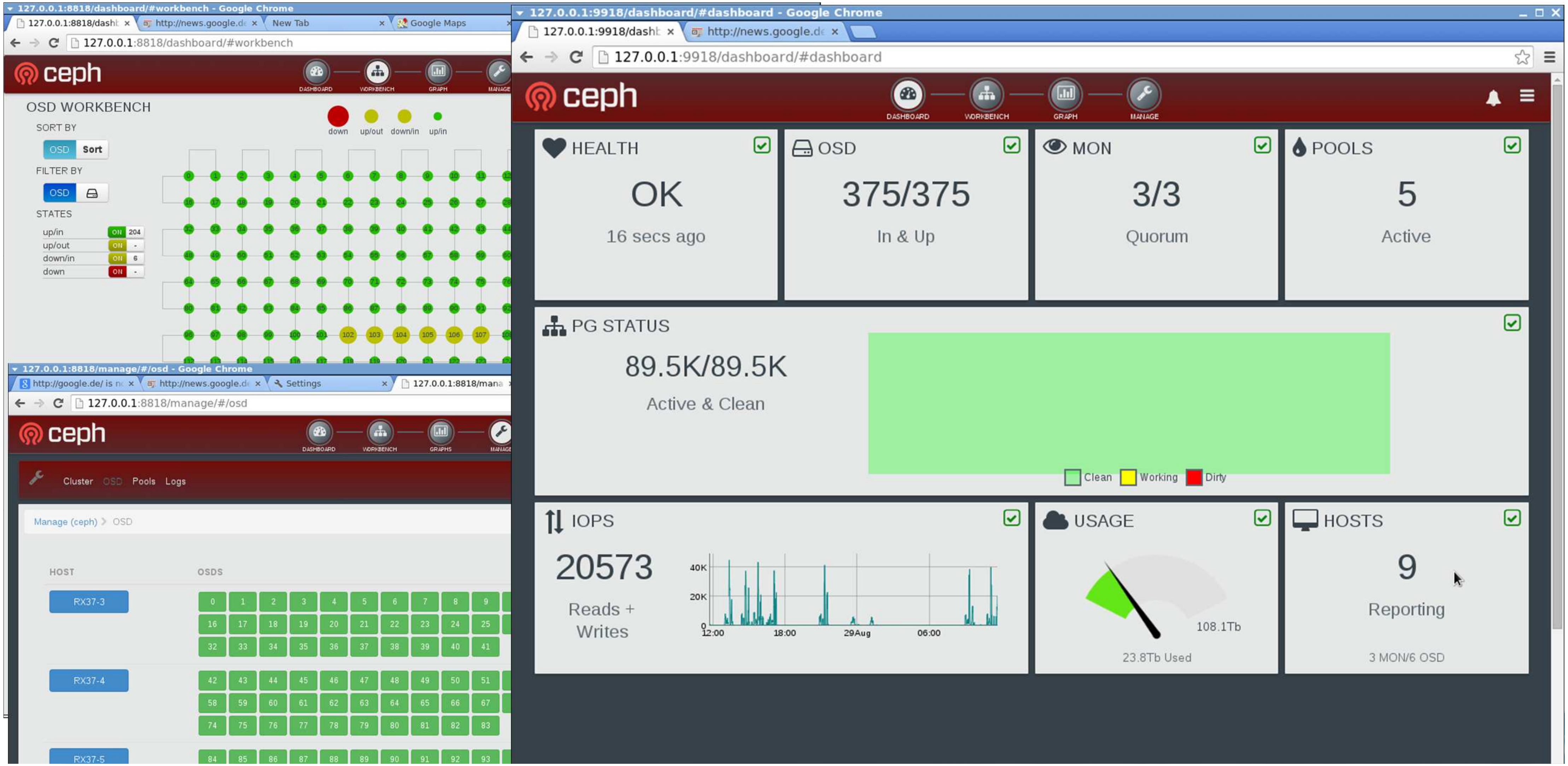
Monitoring Ceph with Calamari

- ICE = Inktank Ceph Enterprise
- Calamari is the Ceph management GUI, with v1.2 mainly for monitoring
- By default calamari depends on ceph-deploy to setup the cluster
- Our installation is with mkcephfs



- Install the Calamari modules as described in the ICE-1.2-Release-Notes.pdf
- Put the Ceph cluster ID as '`fsid = Cluster-ID`' under `[global]` in ceph.conf

Pictures from the Calamari GUI



Agenda

- Introduction & Motivation
- Hardware & System Software
- Software preparation & analysis
- **Performance test cube**
- Conclusion

Performance test matrix

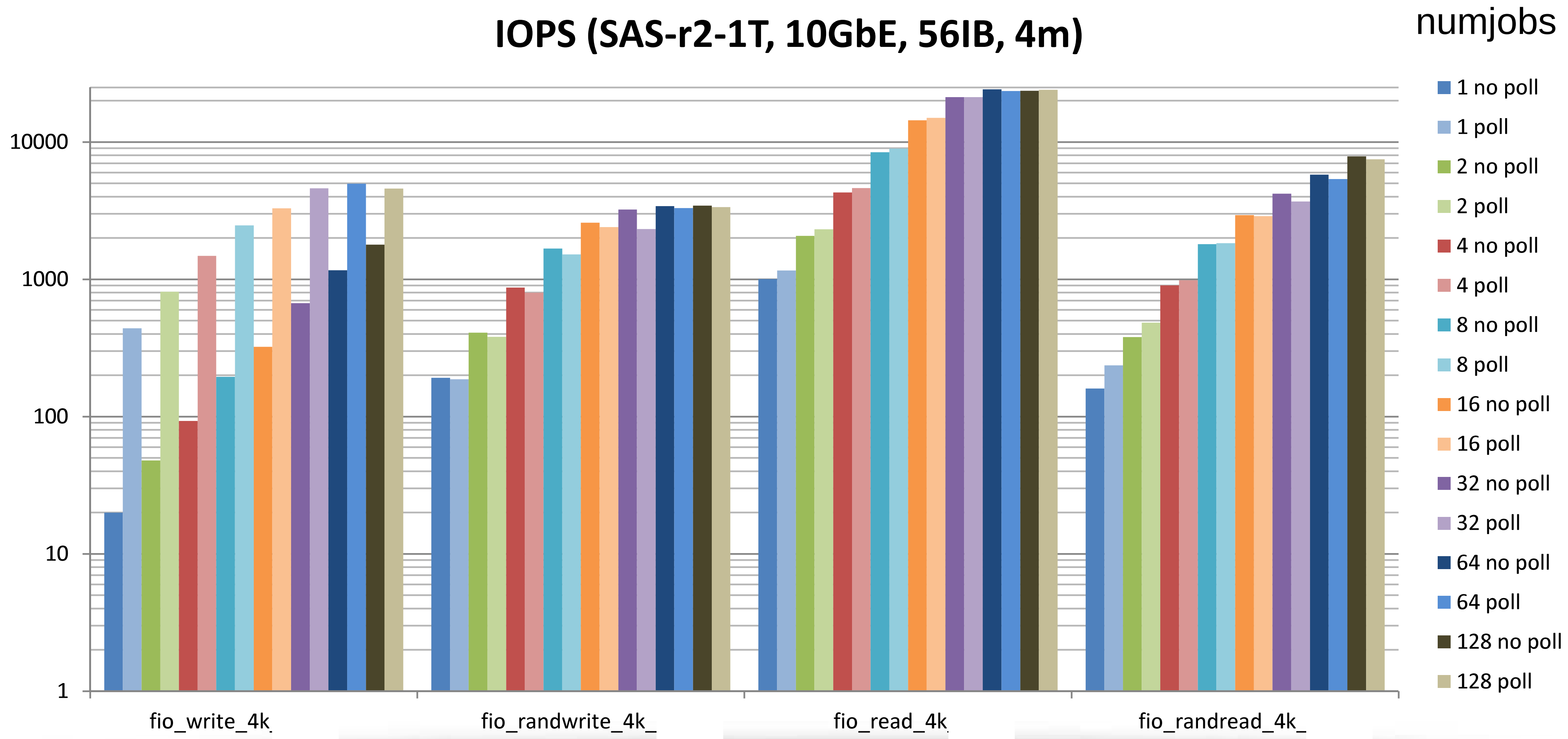
- # Clients
- # Storage Nodes
- Ceph 0.80.4, 0.81, 0.82, 0.83, 0.84
- CentOS-6.5
- SV: kernel 3.10.32-1.el6.FTS.x86_64 intel_idle.max_cstate=0 idle=poll
- CL: kernel-3.16.1 + libceph.ko patch
- fio-2.1.10

```
fio --client $1 $1.fio \
    --client $2 $2.fio \
(…) --client $9 $9.fio \
    --output=fiowrite_4k_32
```

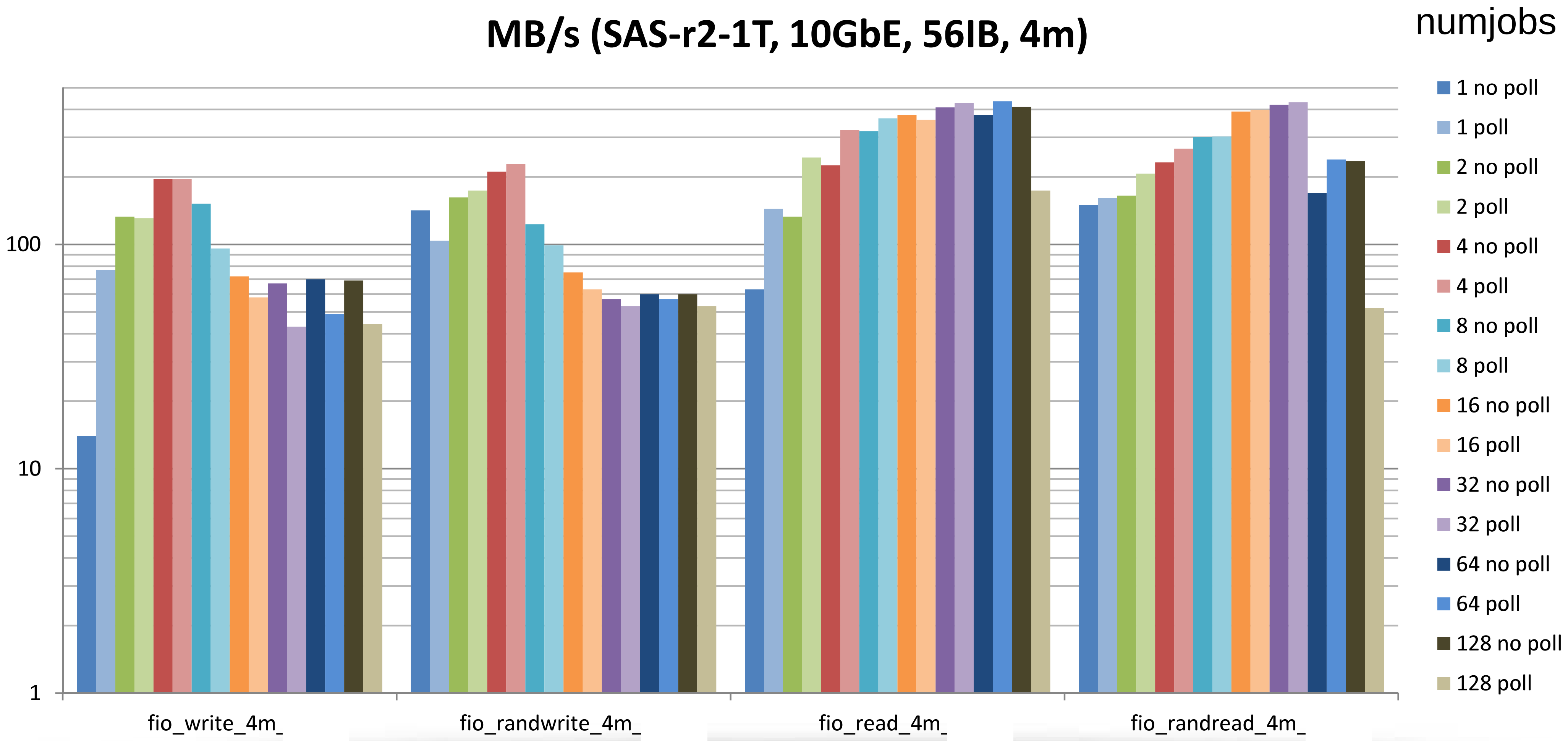
```
[global]
filename=/dev/rbd0 |CephFS|…
direct=1
name=file1
runtime=60
group_reporting

[file]
description=write-4k-32-0
size=32G
offset_increment=32G
rw=write   |read|randwr|randrd
bs=4k       |4m
numjobs=1   |2|4|8|16|32|64|128
```

Kernel poll	Ceph version	Server / OSD	Clients	Frontend	OSD FS
Yes / No	0.84	3 / 120	1	krbd	btrfs



Kernel poll Yes / No	Ceph version 0.84	Server / OSD 3 / 120	Clients 1	Frontend krbd	OSD FS btrfs



Kernel poll Yes / No	Ceph version 0.84	Server / OSD 3 / 120	Clients 1	Frontend krbd	OSD FS btrfs

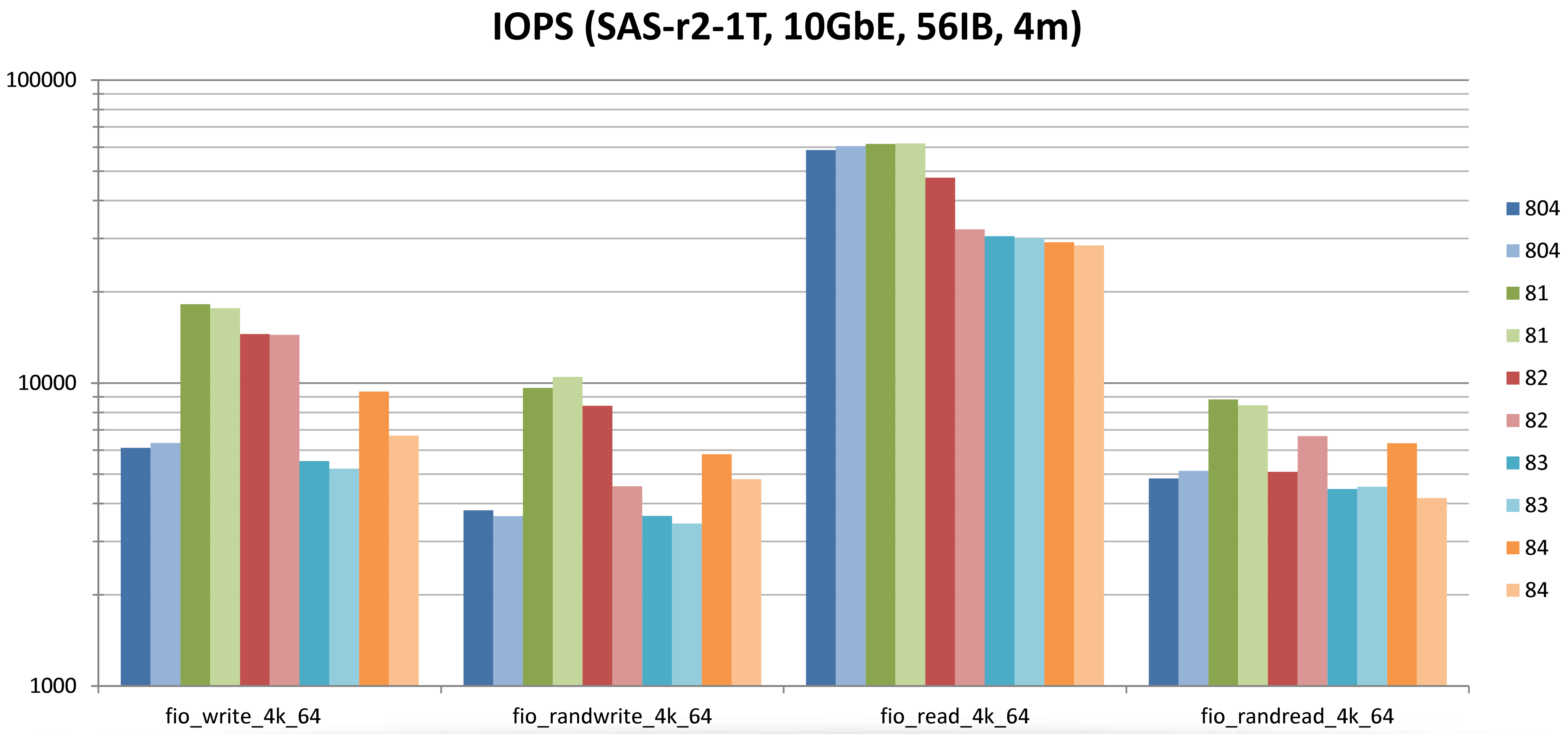
Findings …

- kernel idle=poll
  (Poll forces a polling idle loop that can slightly improve the waking up a idle CPU)

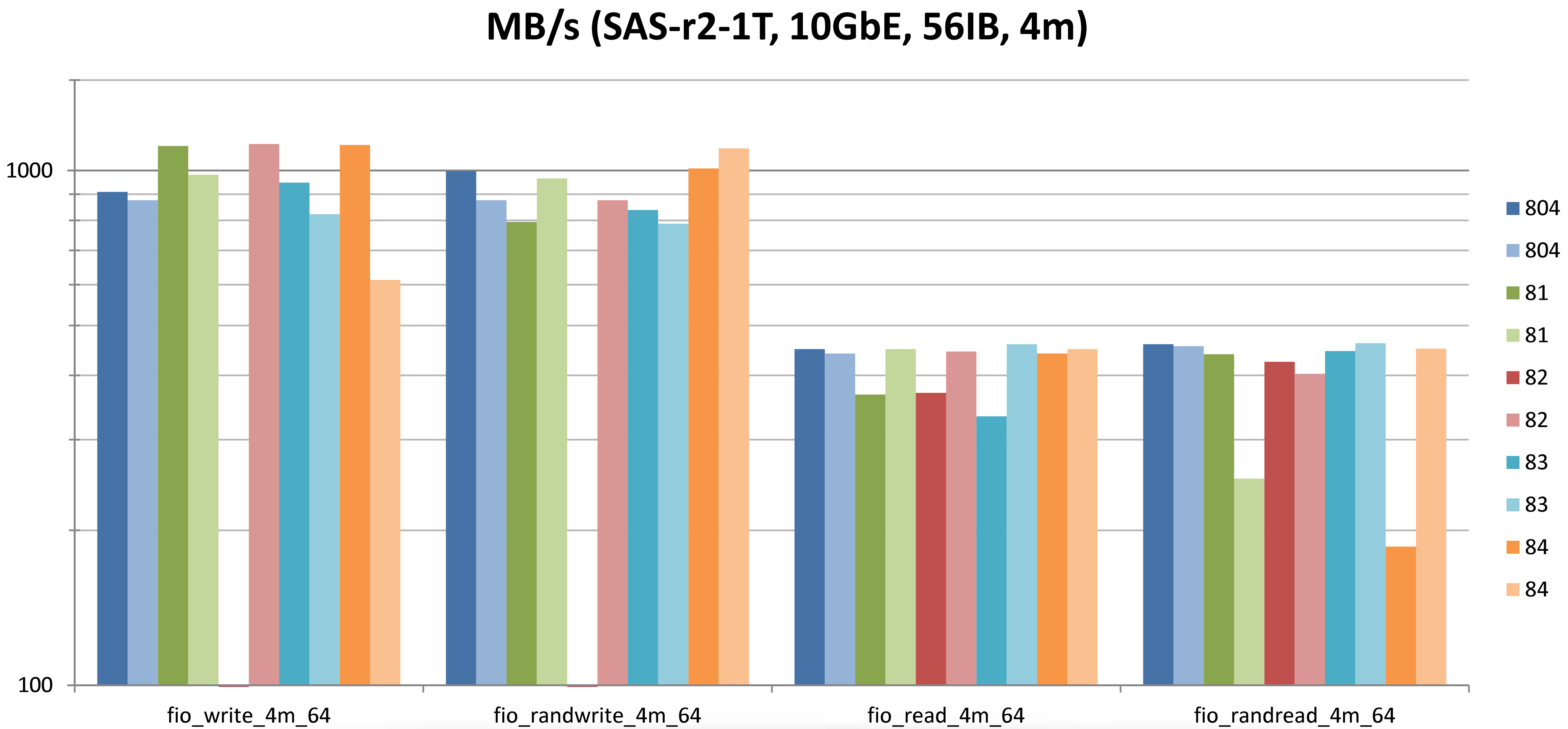
- Is a nice try, but only helps on small sequential writes
- It can only mitigate the symptoms,
  but cannot solve the root cause of expensive context switches

- Not recommended

Kernel poll No	Ceph version	Server / OSD 3x S8 / 117	Clients 1	Frontend krbd	OSD FS xfs



Kernel poll No	Ceph version	Server / OSD 3x S8 / 117	Clients 1	Frontend krbd	OSD FS xfs



Kernel poll No	Ceph version	Server / OSD 3x S8 / 117	Clients 1	Frontend krbd	OSD FS xfs

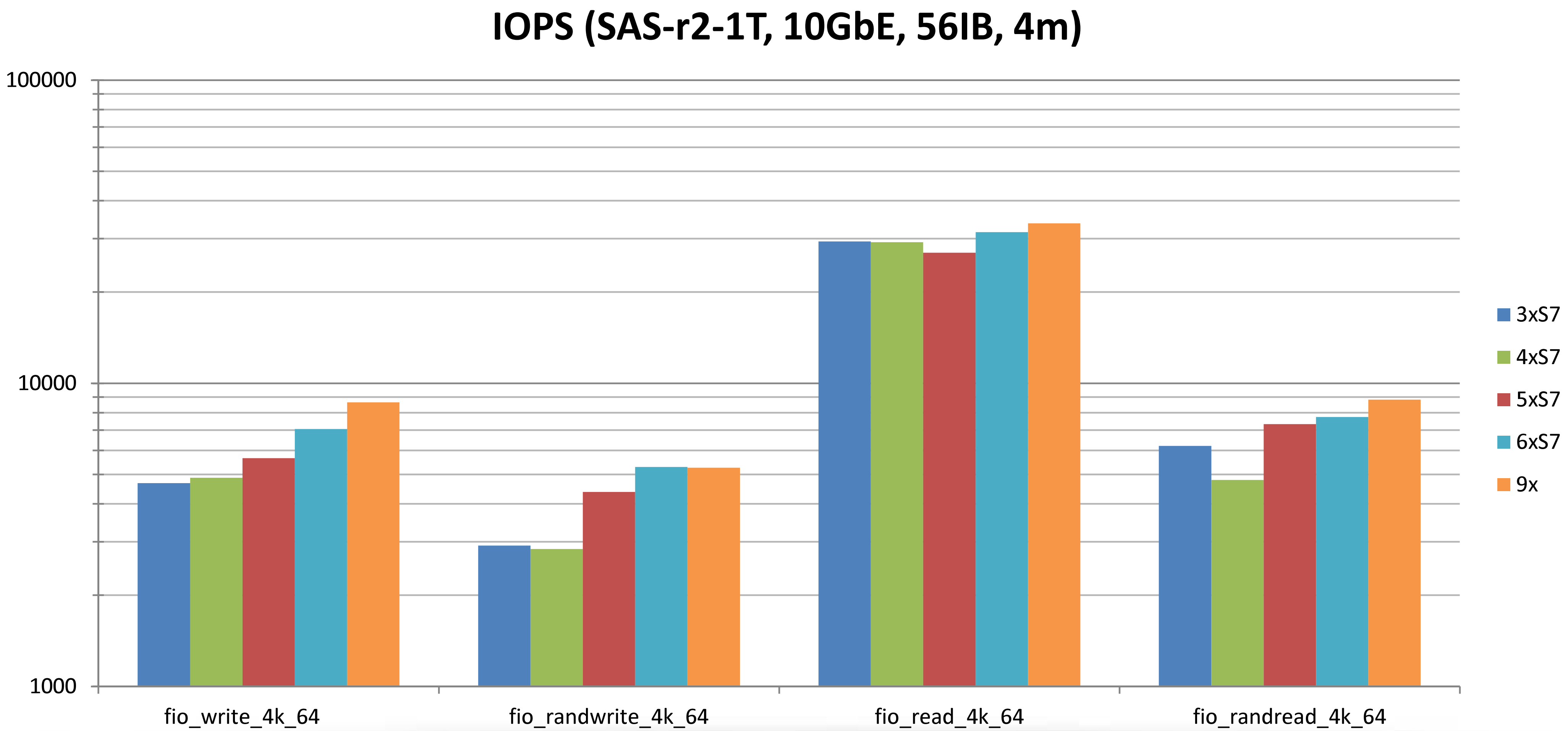
Findings …

- Ceph Version 0.80.4, 0.81, 0.82, 0.83, 0.84

- Measurable improvements have been made in v0.81

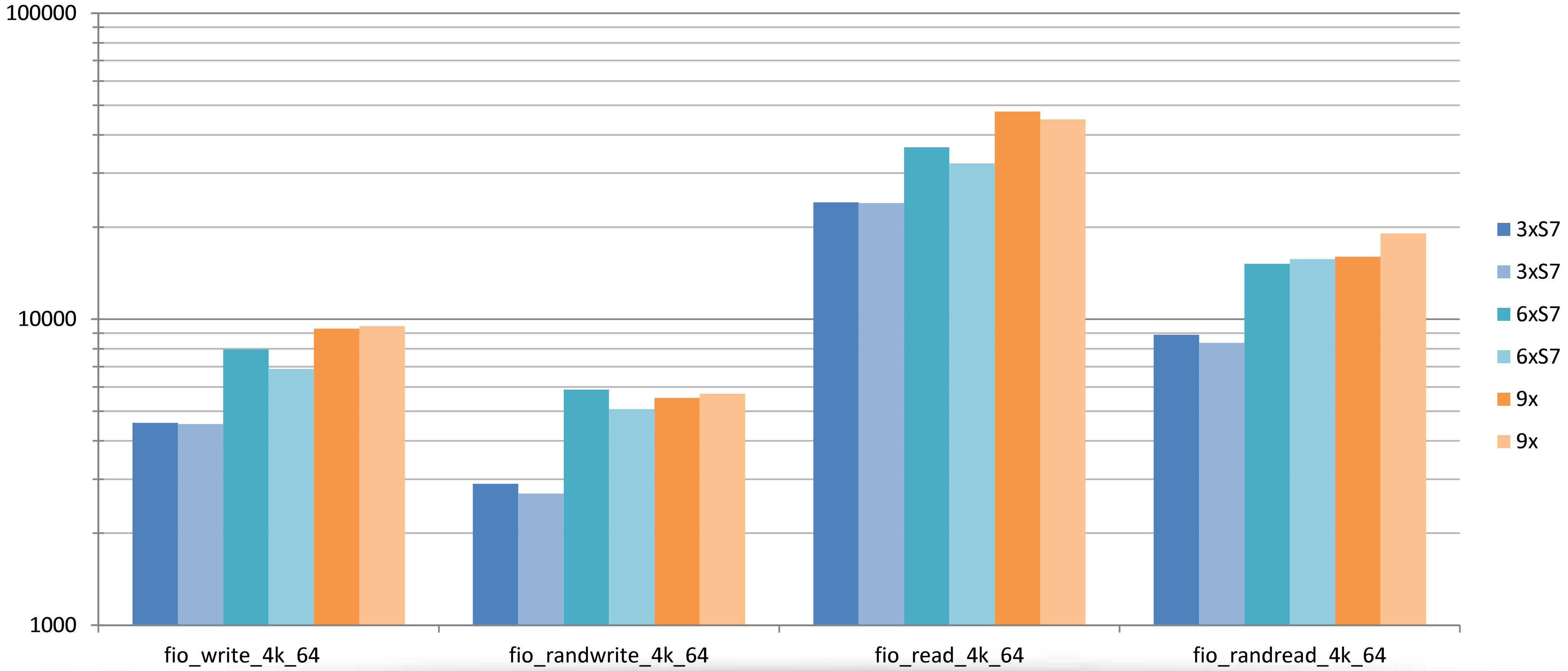
- No difference on large IO blocks
- v0.81 is the fastest one in the list above, especially for IOPS with small blocks
- Still lots of room for improvement

Kernel poll	Ceph version	Server / OSD	Clients	Frontend	OSD FS
Yes	0.84		1	krbd	xfs

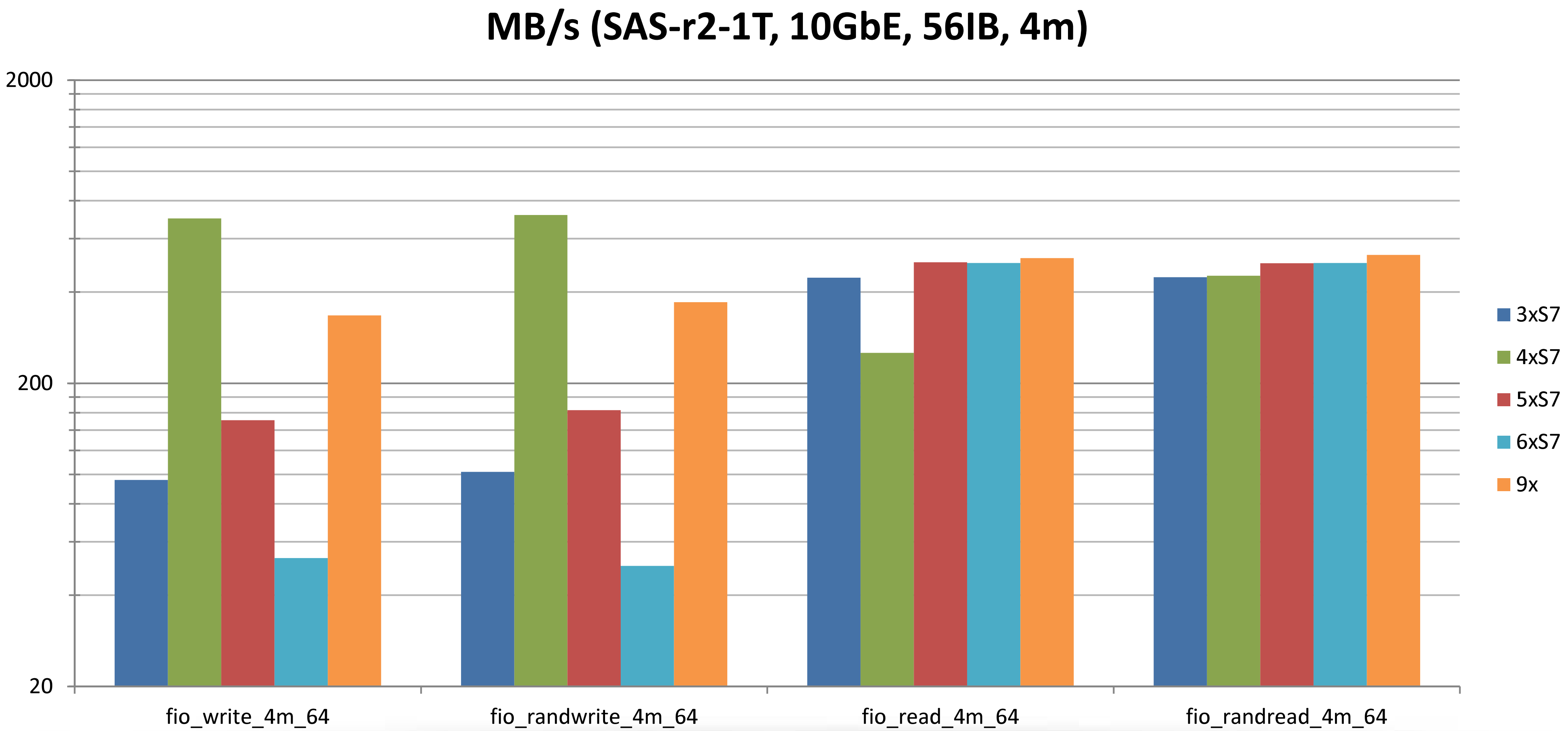


Kernel poll	Ceph version	Server / OSD	Clients	Frontend	OSD FS
Yes	0.84		3	krbd	xfs

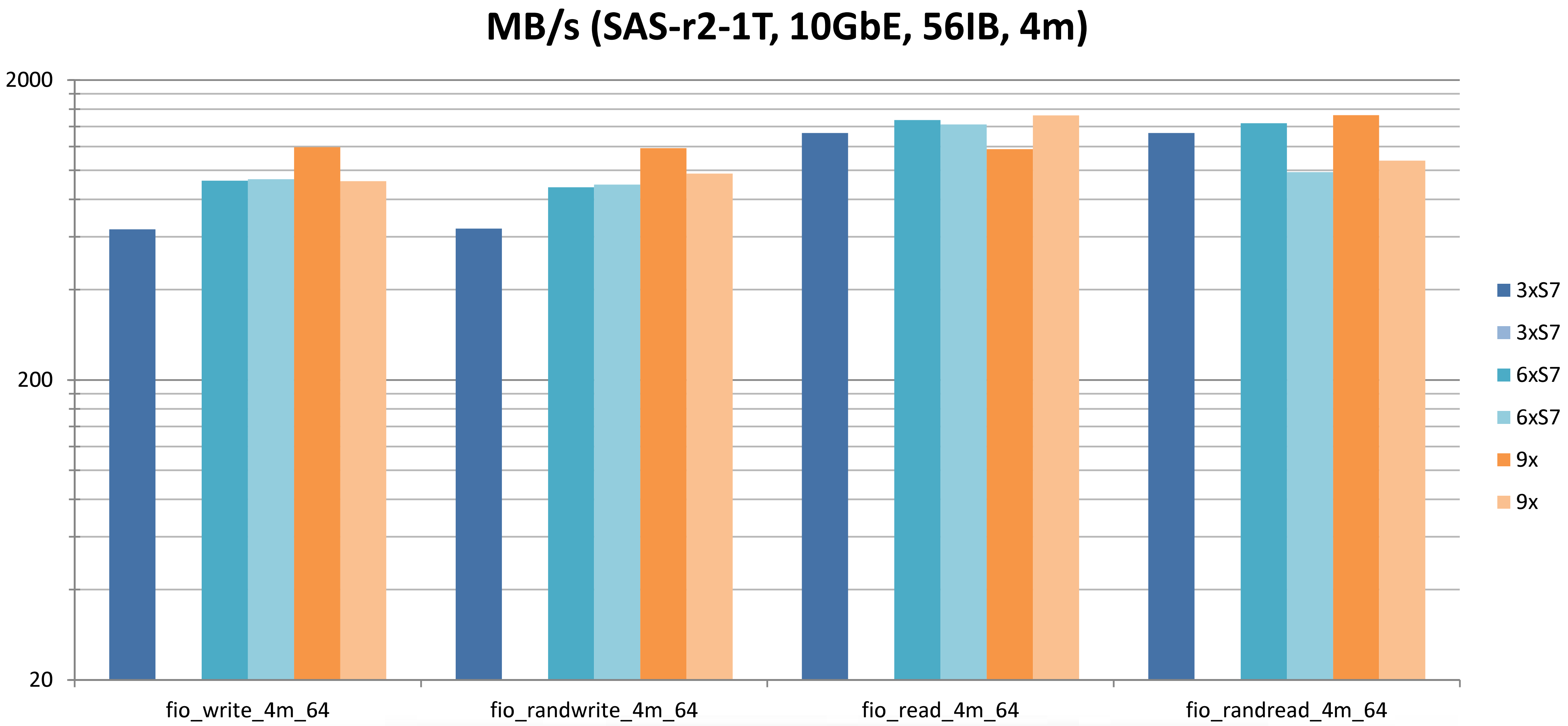
IOPS (SAS-r2-1T, 10GbE, 56IB, 4m)



Kernel poll	Ceph version	Server / OSD	Clients	Frontend	OSD FS
Yes	0.84		1	krbd	xfs



Kernel poll	Ceph version	Server / OSD	Clients	Frontend	OSD FS
Yes	0.84		3	krbd	xfs



Kernel poll Yes	Ceph version 0.84	Server / OSD	Clients 1 | 3	Frontend krbd	OSD FS xfs

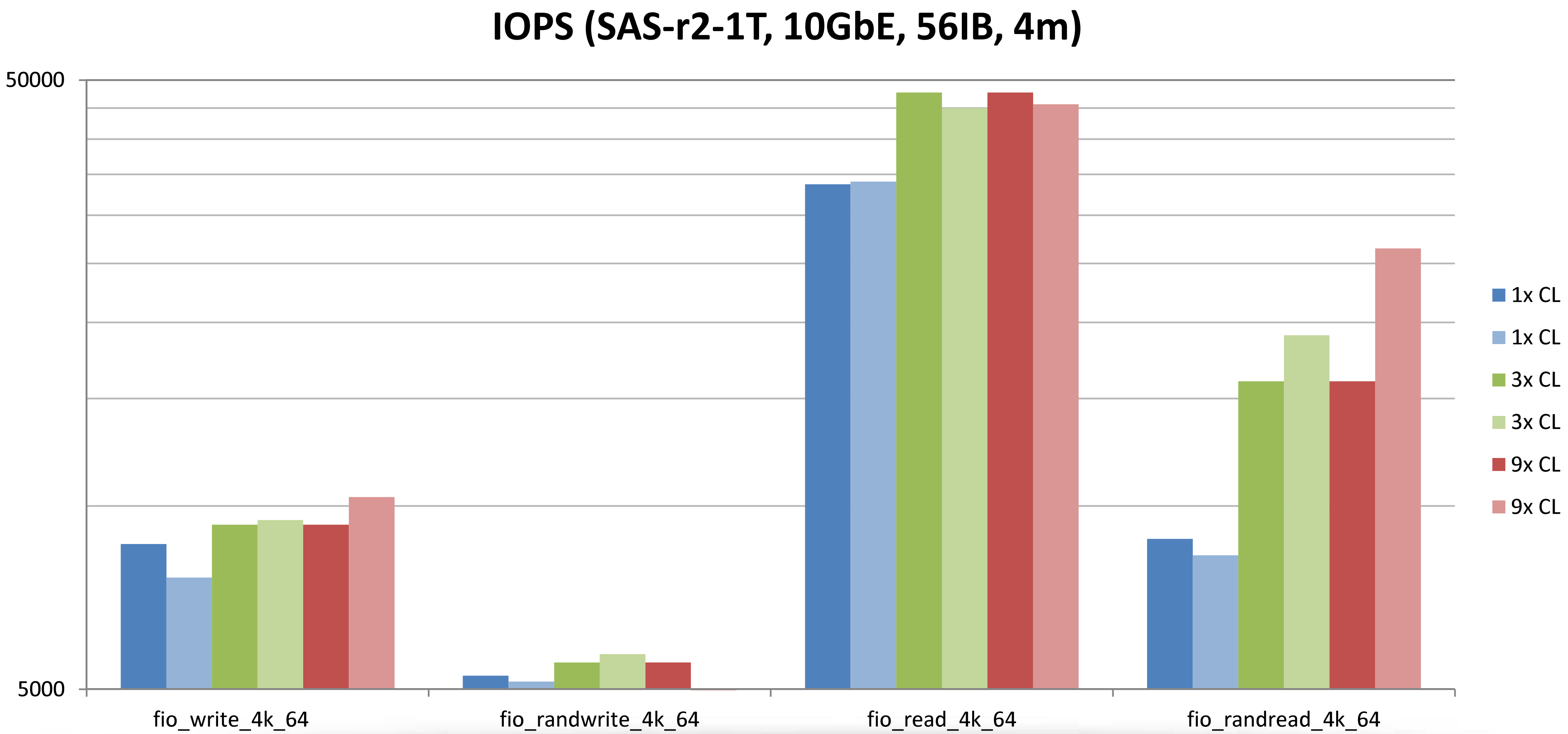
Findings …

- Storage nodes 3x, 4x, 5x, 6x, 9x

- 1 CL does see ~66% scale factor for 4k writes (sequential & random)
- 1 CL gets max. IOPS of 30k already with 3 nodes, no scale
- 3 CL make no change to 1 CL on 4k writes
- 3 CL do benefit on read IOPS by 25-50% scale factor
- On large IO more OSD will increase writes by ~50%, but reads only by ~10%

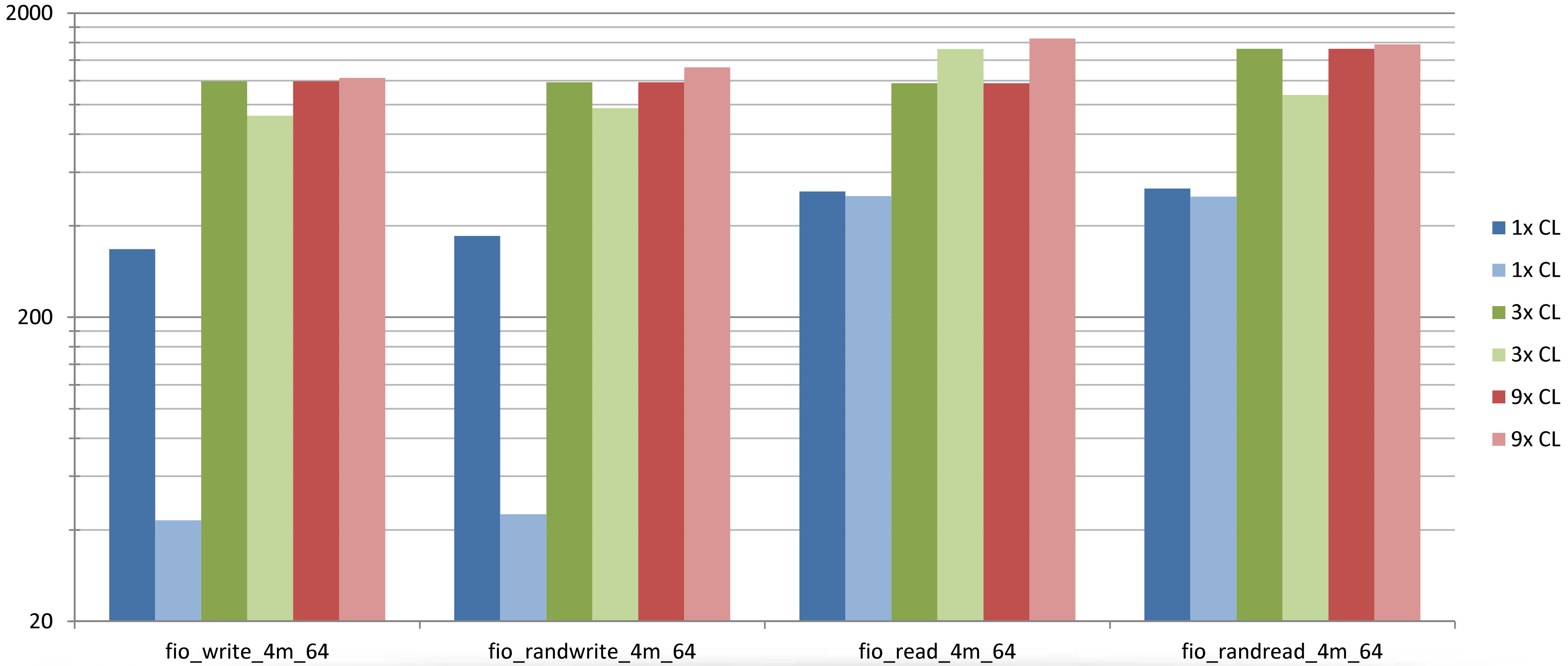
- krbd client seems to have limitation to scale over 30k IOPS
- The Ceph OSD tread implementation seems to inhibit a scale on IOPS when adding more Storage nodes / OSDs
- Still lots of room for improvement

Kernel poll	Ceph version	Server / OSD	Clients	Frontend	OSD FS
Yes	0.84	9x / 357		krbd	xfs



Kernel poll	Ceph version	Server / OSD	Clients	Frontend	OSD FS
Yes	0.84	9x / 357		krbd	xfs

MB/s (SAS-r2-1T, 10GbE, 56IB, 4m)



Kernel poll Yes	Ceph version 0.84	Server / OSD 9x / 357	Clients	Frontend krbd	OSD FS xfs

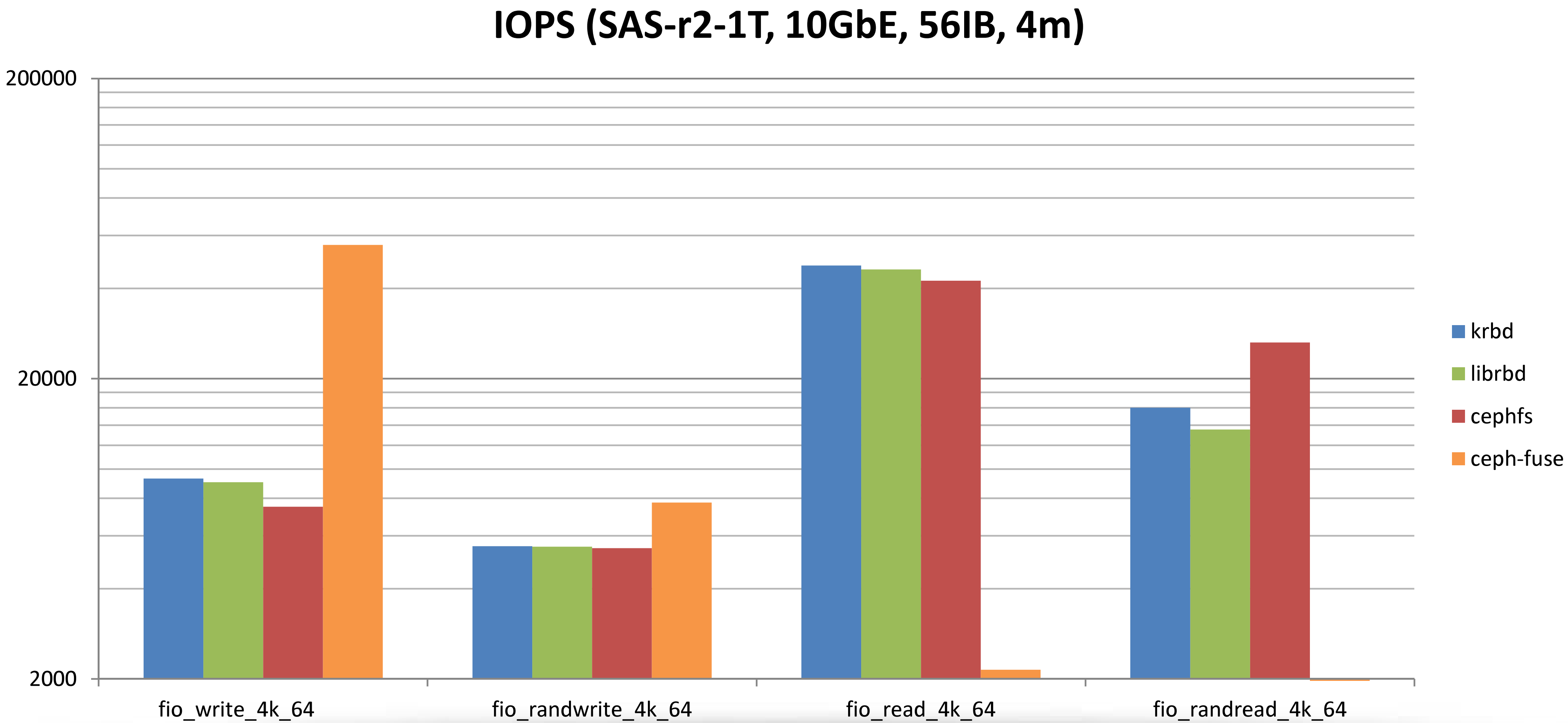
Findings …

- Client nodes 1x, 3x, 9x

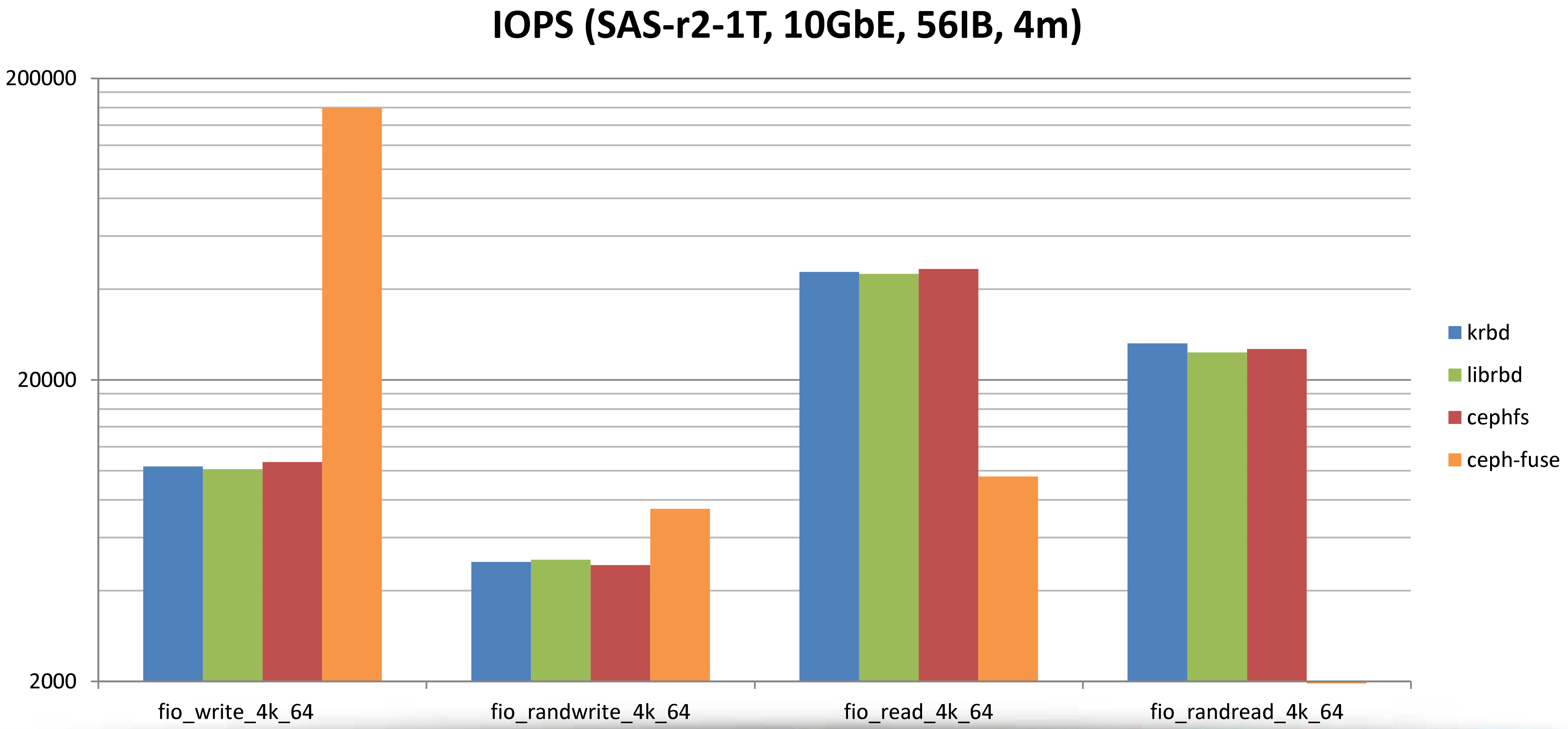
- ~10% scale factor for 4k IOPS (sequential & random)
- 50% scale on sequential read IOPS from 1x to 3x Clients, but no more improvement with 9 CLs
- Continuous 66% scale on small random writes
- Large IOs do scale 100% for read and write between 1x – 3x CL, but then get saturated when switching to 9x CLs

- Good scale factor for reads
- On write IOPS the Ceph OSDs get in the way of themselves and inhibit scale
- Room for improvement

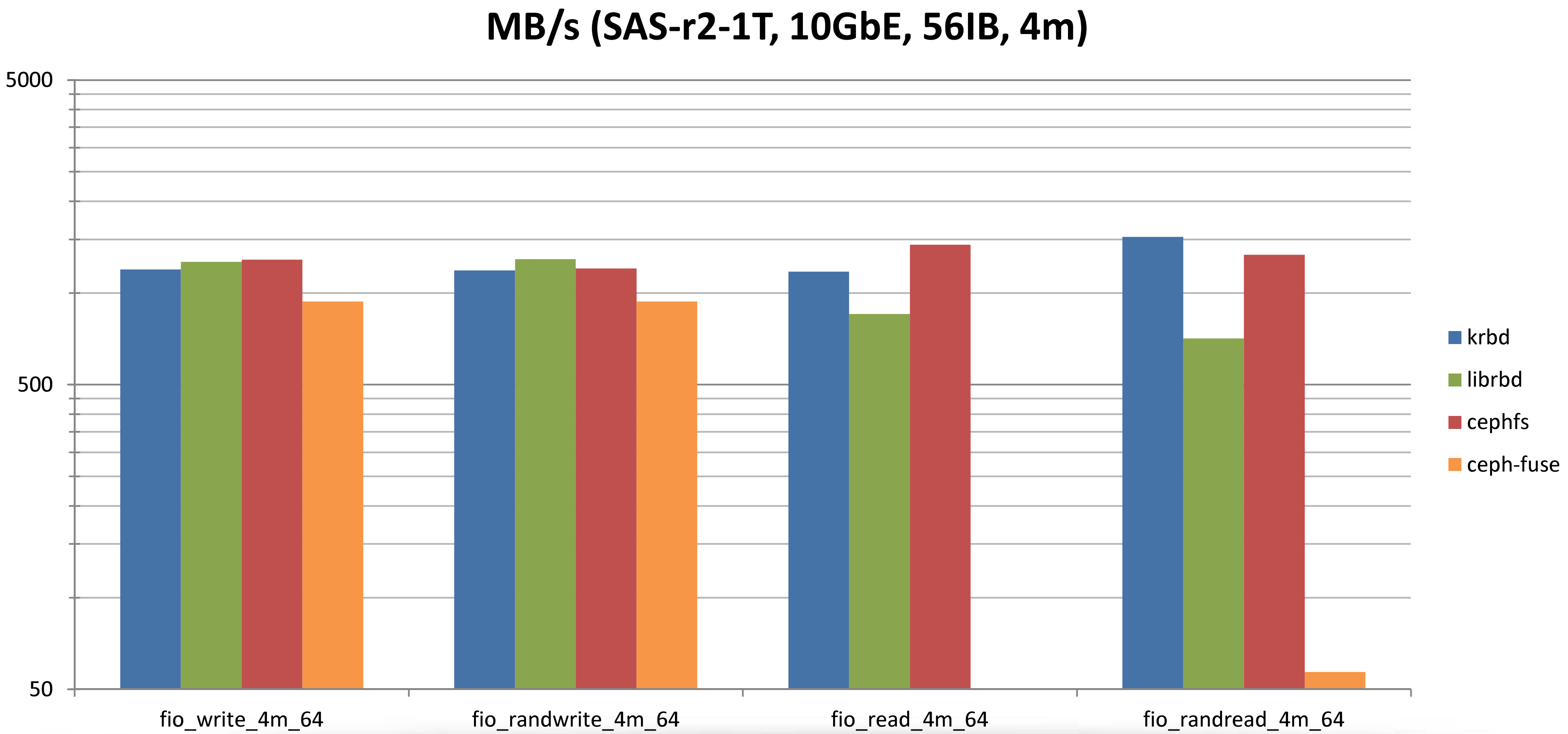
Kernel poll	Ceph version	Server / OSD	Clients	Frontend	OSD FS
Yes	0.84	9x / 357	3		xfs



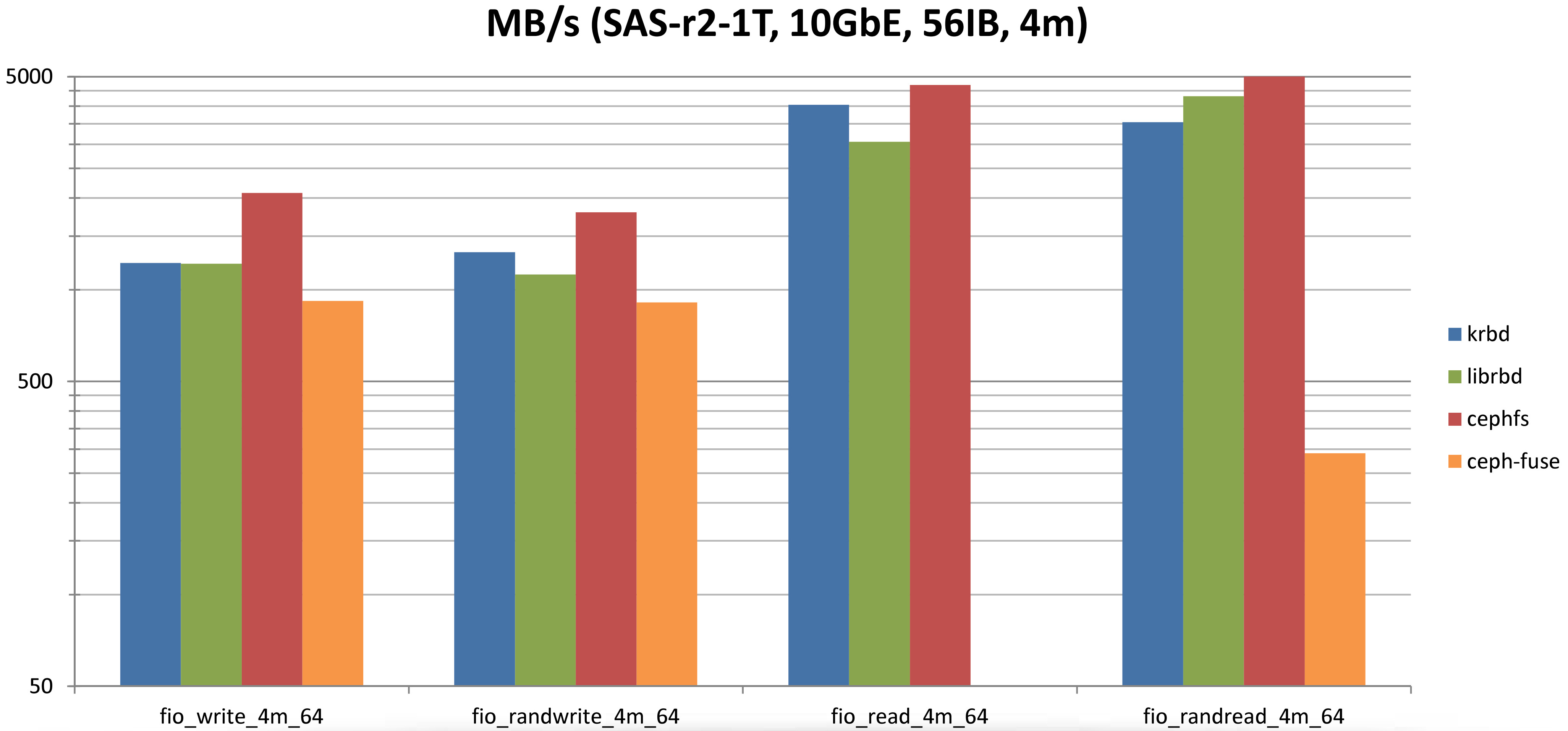
Kernel poll	Ceph version	Server / OSD	Clients	Frontend	OSD FS
Yes	0.84	9x / 357	9		xfs



Kernel poll	Ceph version	Server / OSD	Clients	Frontend	OSD FS
Yes	0.84	9x / 357	3		xfs



Kernel poll	Ceph version	Server / OSD	Clients	Frontend	OSD FS
Yes	0.84	9x / 357	9		xfs



Kernel poll Yes	Ceph version 0.84	Server / OSD 9x / 357	Clients 3	Frontend	OSD FS xfs

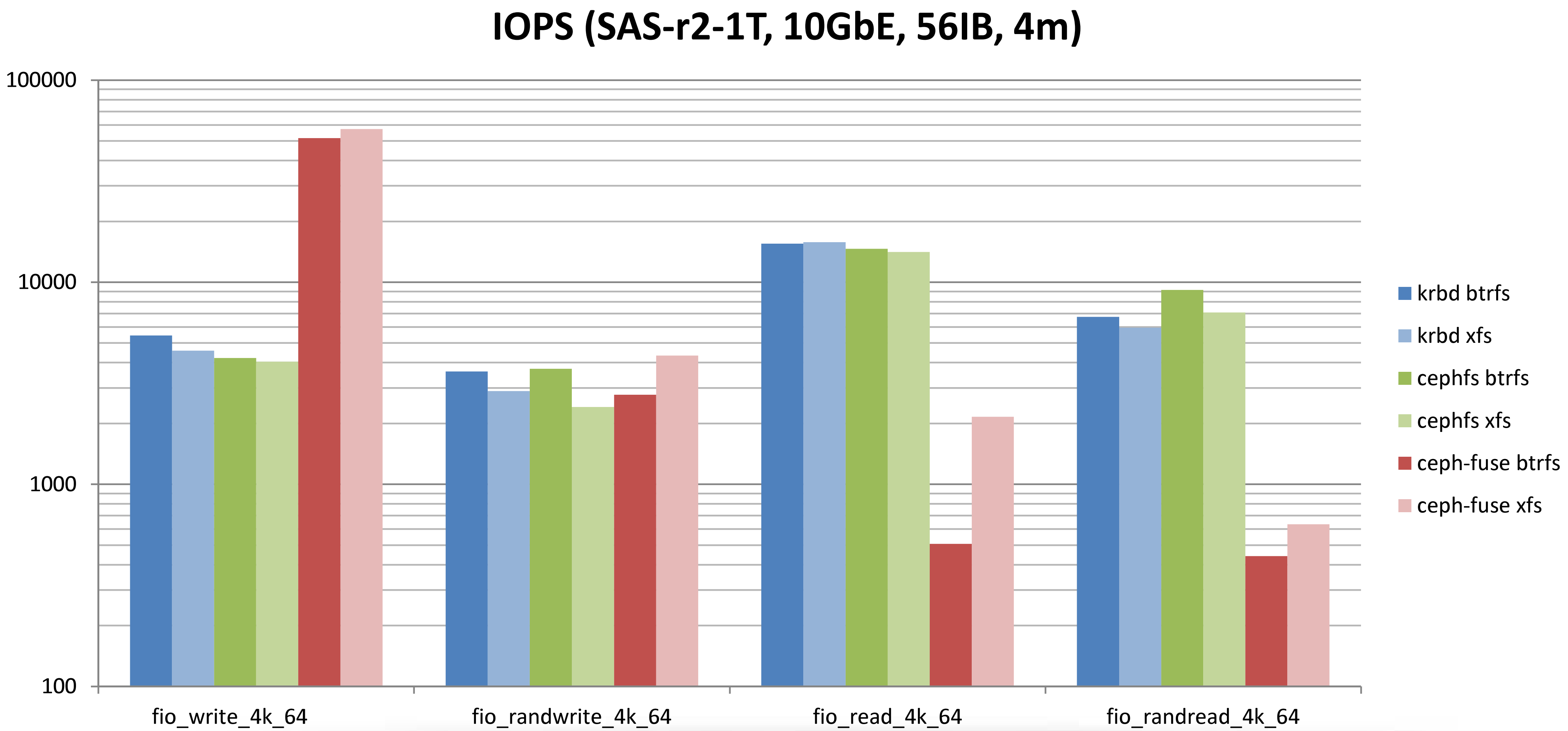
Findings …

- Frontends: krbd, librbd, CephFS, ceph-fuse

- Only small differences between krbd, librbd and CephFS
- Ceph-fuse is fantastic fast on write IOPS, especially in the sequential case, but unacceptably slow on reads

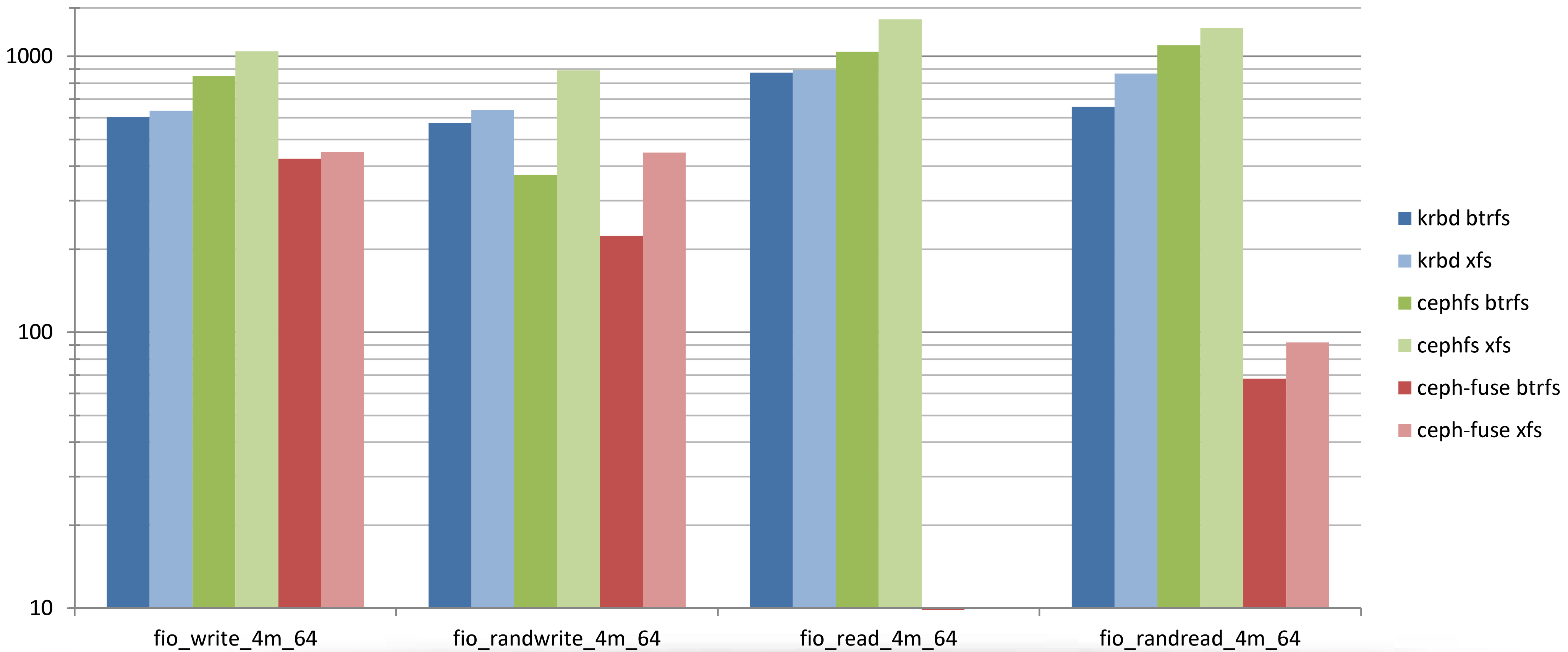
- CephFS is doing extremely well
- A check is needed if the complete stack below ceph-fuse is respecting the ‘direct=1’ flag (try to minimize cache effects of the I/O to and from this file).
- In the read case the ceph-fuse seems to have big limitations doing parallel IOs and avoiding readahead / cacheing for small IOs
- Room for improvement for Ceph-Fuse

Kernel poll Yes	Ceph version 0.84	Server / OSD 3x / 120	Clients 3	Frontend	OSD FS



Kernel poll Yes	Ceph version 0.84	Server / OSD 3x / 120	Clients 3	Frontend	OSD FS

MB/s (SAS-r2-1T, 10GbE, 56IB, 4m)



Kernel poll Yes	Ceph version 0.84	Server / OSD 3x / 120	Clients 3	Frontend	OSD FS

Findings …

- OSD File Systems: btrfs, xfs

- Btrfs has small advantages for IOPS with the kernel front-ends krbd & CephFS
- Xfs is doing better with ceph-fuse
- For 4m IOs xfs is slightly better than btrfs for writes and reads and for all front- end interfaces

➢ If compression and de-dupe is not needed xfs is the better choice

Summary and conclusion

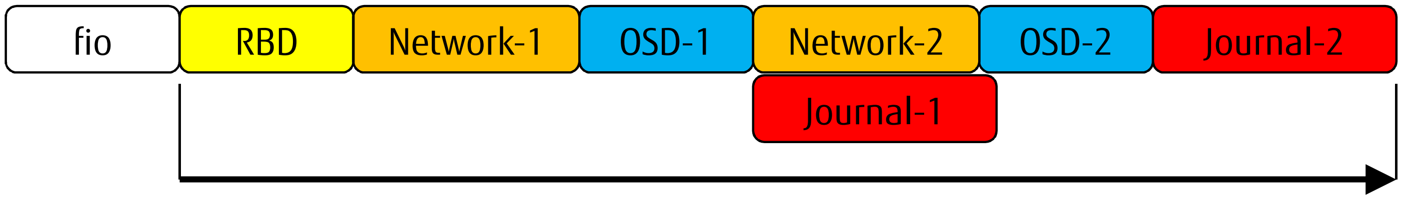
- Ceph is the most comprehensive implementation of Unified Storage. Ceph simulates “distributed swarm intelligence” which arise from simple rules that are followed by individual processes and does not involve any central coordination.
- The Crush algorithm acts as an enabler for a controlled, scalable, decentralized placement of replica data.
- NVMe with High-Endurance SSD is recommended to host the Journal
- The Client / Cluster bandwidth should have a factor of ~ 2.5
- With todays implementation of the OSD the CPU is the critical resource and dominating factor for high performance, especially for IOPS
- Almost equal performance between the different frontend interfaces
- Inktank/Redhat has continue with the code optimization
  to increase the overall performance and scalability

FUJITSU

shaping tomorrow with you


Fujitsu
Technology
Solutions
FUJITSU
CTO Data Center Infrastructure,
Global Emerging Technologies
Dieter.Kasper@ts.fujitsu.com
v6

Analysis of the TAT of a single 4k IO (v0.61 in 2013)



Time = avg latency of one IO (queue-depth=1) with 5x ACK

	rbd		Network		Intel 910		ACK	Ceph code	
µsec	fio write		qperf lat		fio_randwrite		msg		
	4k	8k	4k	8k	4k	8k	128	4k	8k
1 GbE	2565	2709	182	227	54	64	26	2017	2061
10 GbE	2555	2584	109	122	54	64	21	2178	2171
40 GbE	2191	2142	19	22	54	64	15	2024	1959
40 Gb IPoIB	2392	2357	29	24	54	64	18	2190	2155
56 Gb IPoIB	1848	1821	19	37	54	64	14	1686	1613

- approximately 1600 µs on a single 4k/8k IO is spend in the Ceph code
  - ➢ The Ceph code has a lot of room for improvement