qcow2 – why (not)?

Max Reitz <mreitz@redhat.com>
Kevin Wolf <kwolf@redhat.com>

KVM Forum 2015

Choosing between raw and qcow2

- Traditional answer:
  - Performance? raw!
  - Features? qcow2!
- But what if you need both?

A car analogy

- Throwing out the seats gives you better acceleration
- Is it worth it?

A car analogy

- Throwing out the seats gives you better acceleration
- Is it worth it?

Our goal

- Keep the seats in!
- Never try to get away without qcow2's features

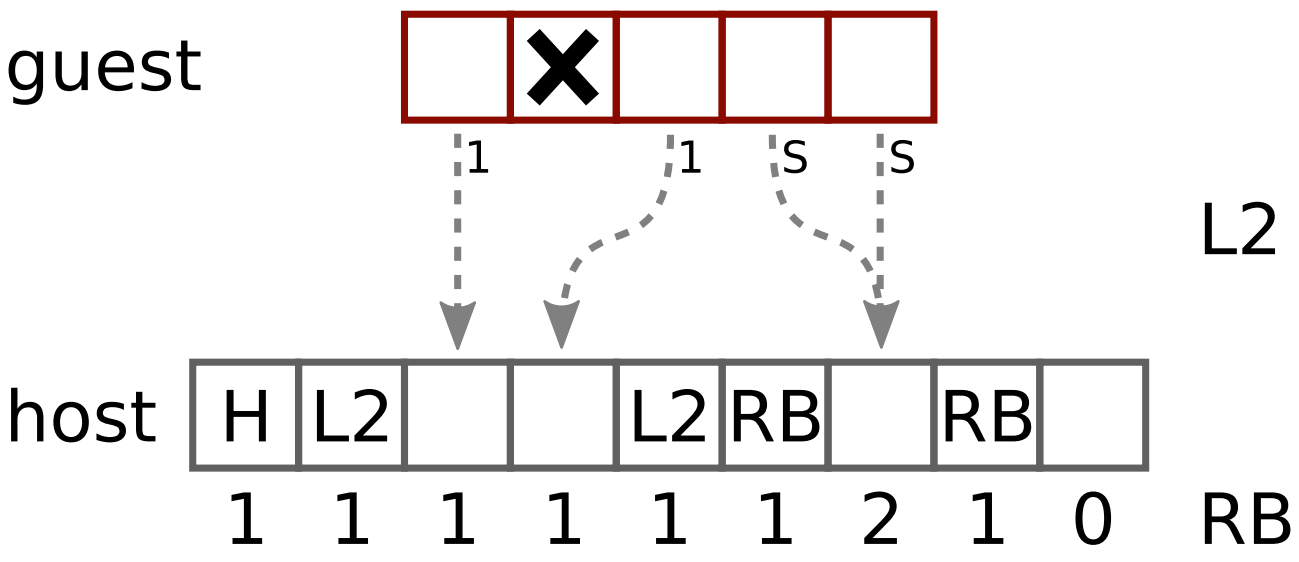
Part I
What are those features?

qcow2 features

- Backing files
- Internal snapshots
- Zero clusters and partial allocation (on all filesystems)
- Compression

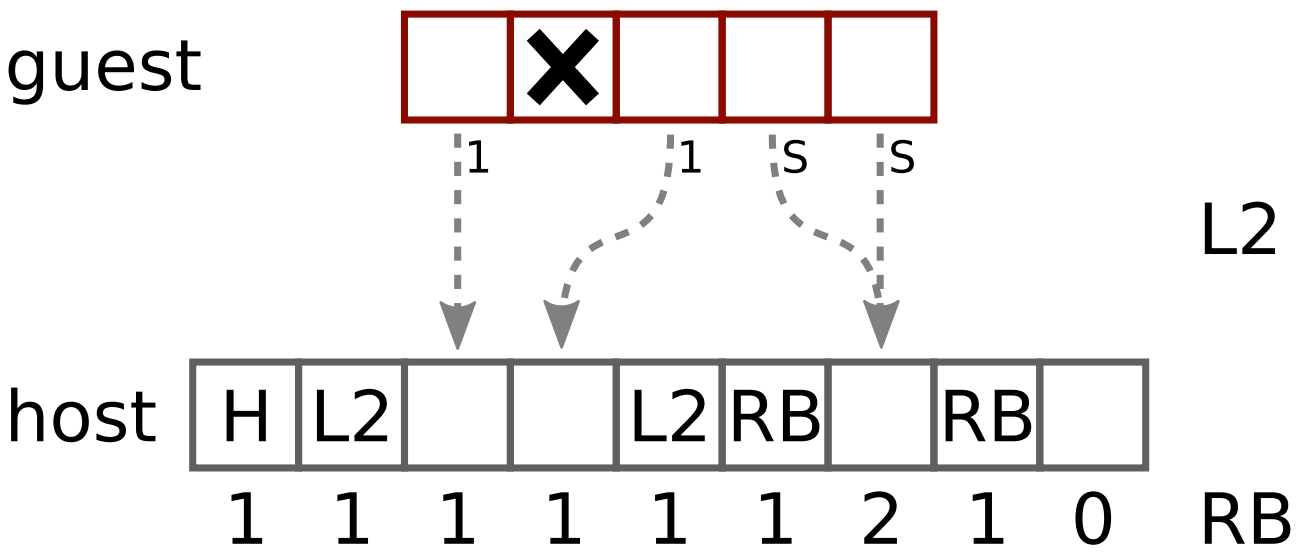
qcow2 metadata

- Image is split into clusters
  (default: 64 kB)
- L2 tables map guest offsets to host offsets
- Refcount blocks store allocation information



qcow2 metadata

- For non-allocating I/O:
  Only L2 tables needed



redhat.

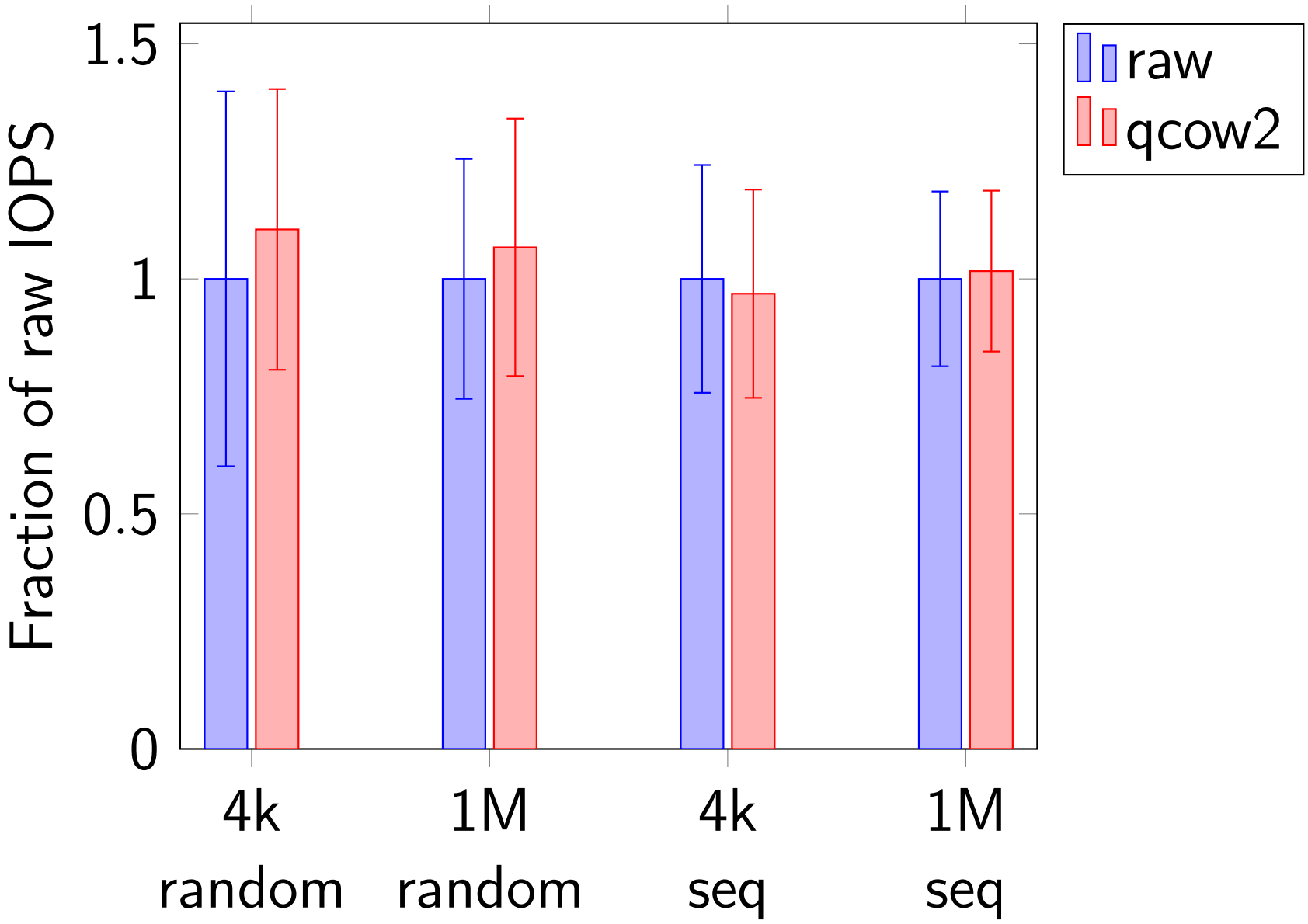
Part II

Preallocated images

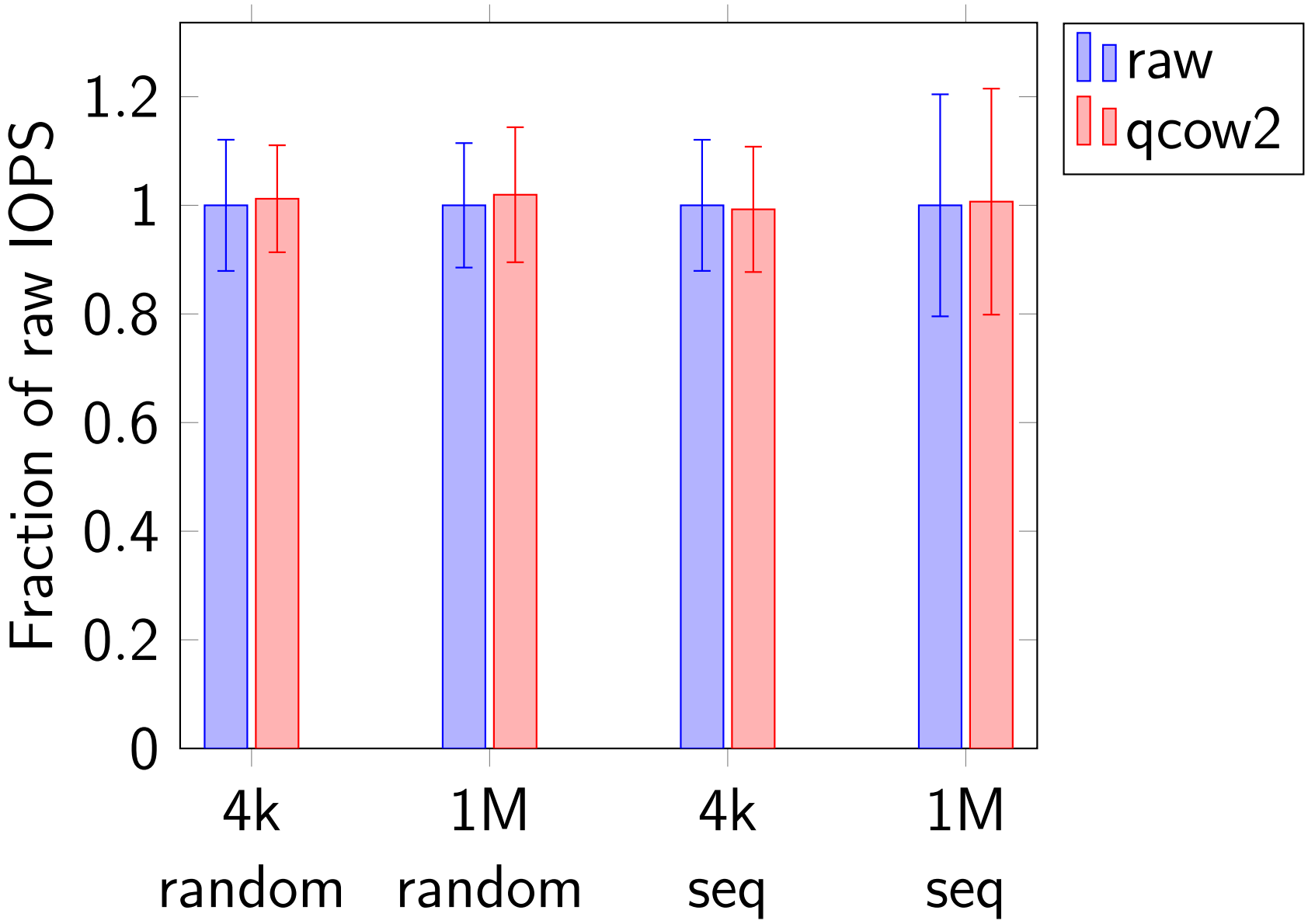
What is tested?

- Linux guest with fio
  (120 s runtime per test/pattern; `O_DIRECT` AIO)
- 6 GB images on SSD and HDD
- Random/sequential 4k/1M blocks
- qcow2: `preallocation=metadata`

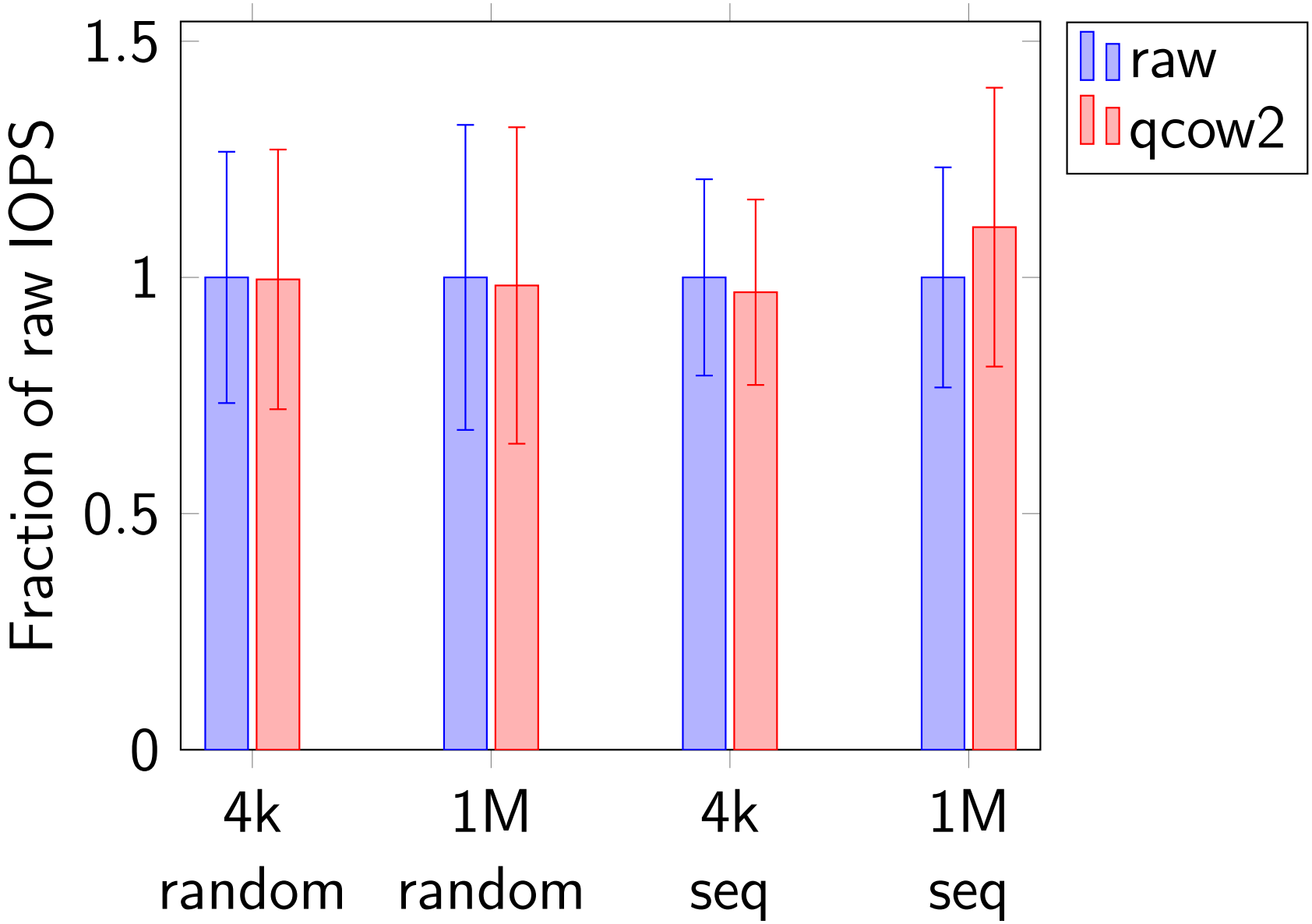
SSD write performance



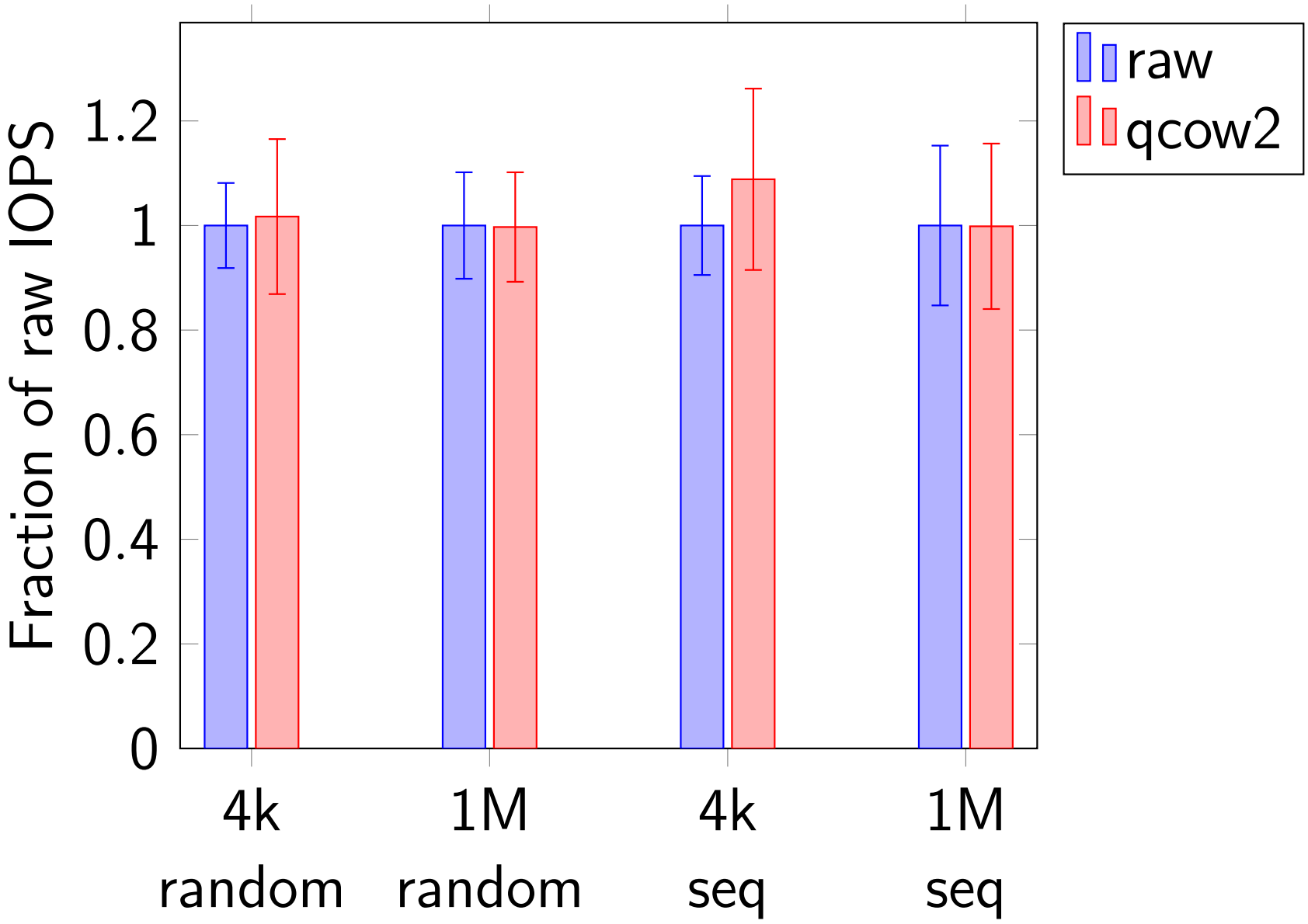
SSD read performance



HDD write performance



HDD read performance



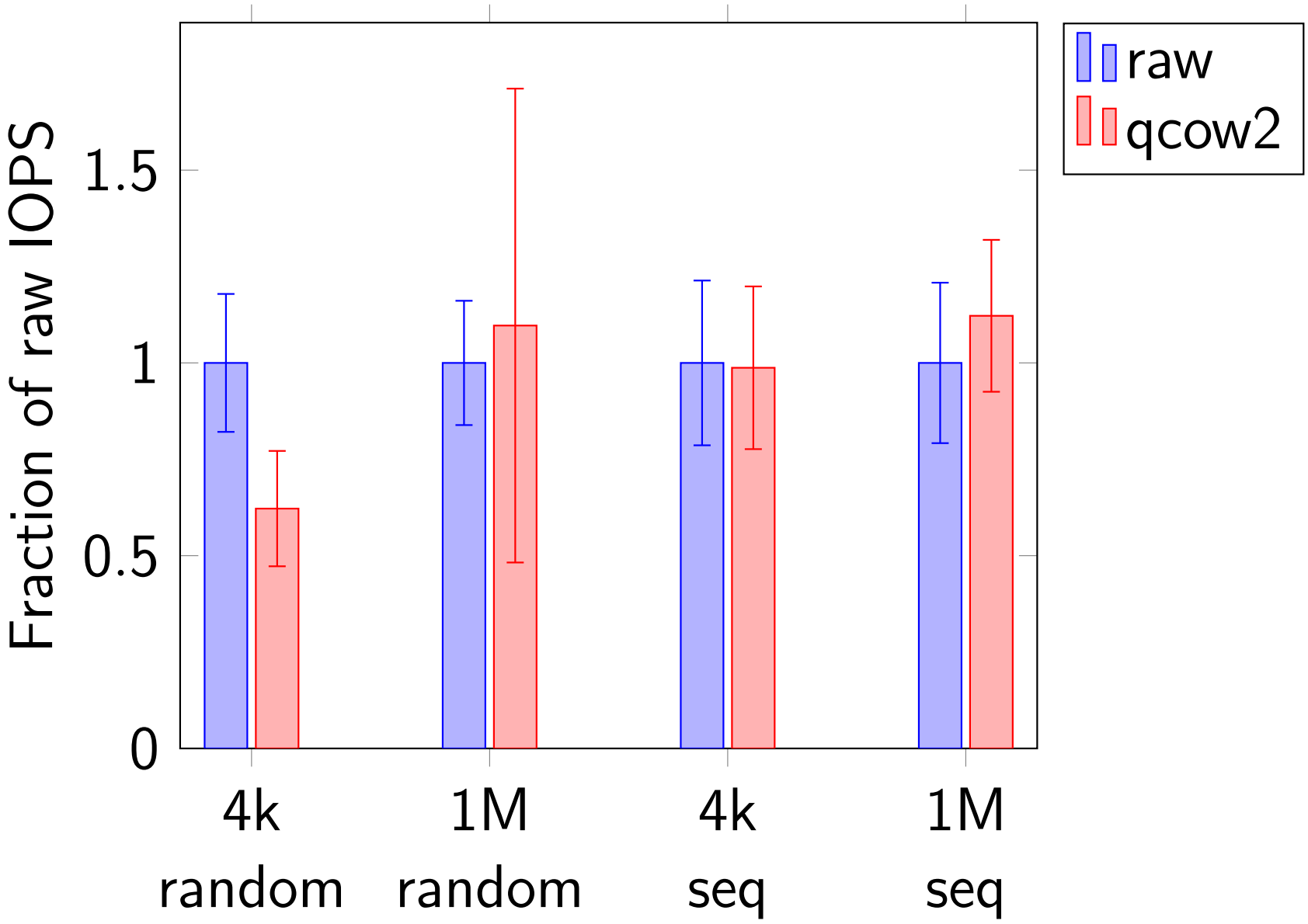
So?

Looks good, right?

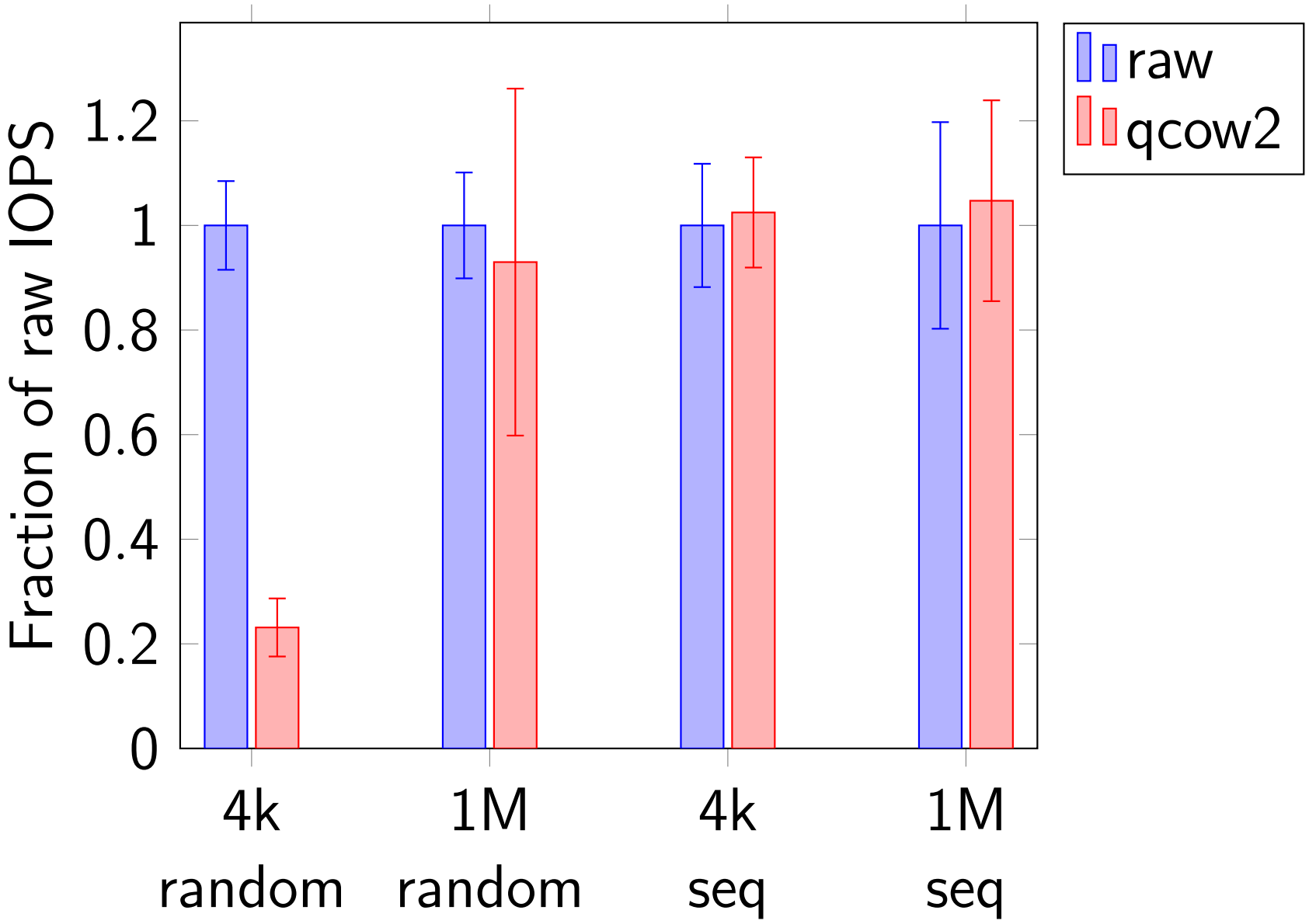
So?

Let's increase the image size!

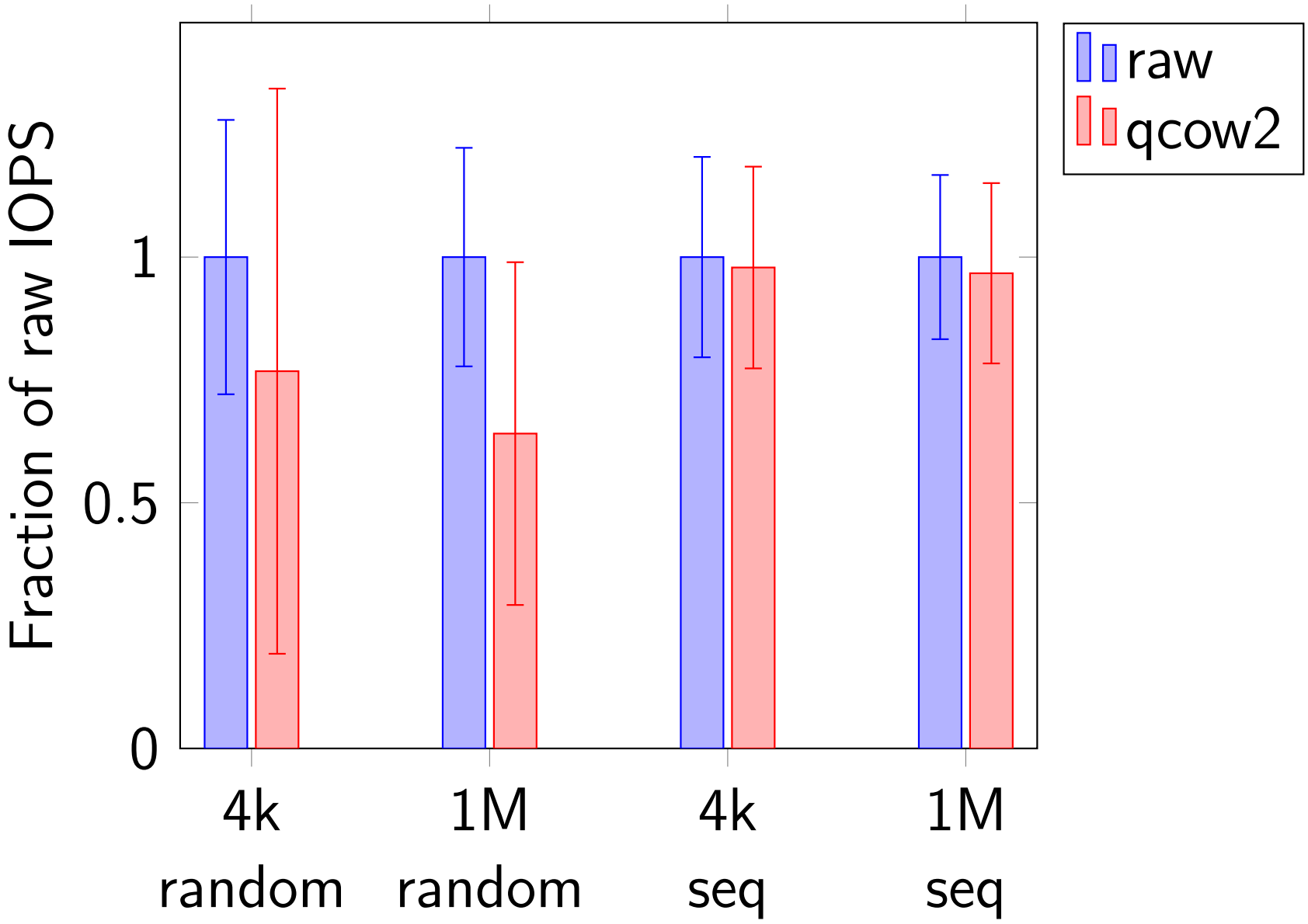
SSD 16 GB image write performance



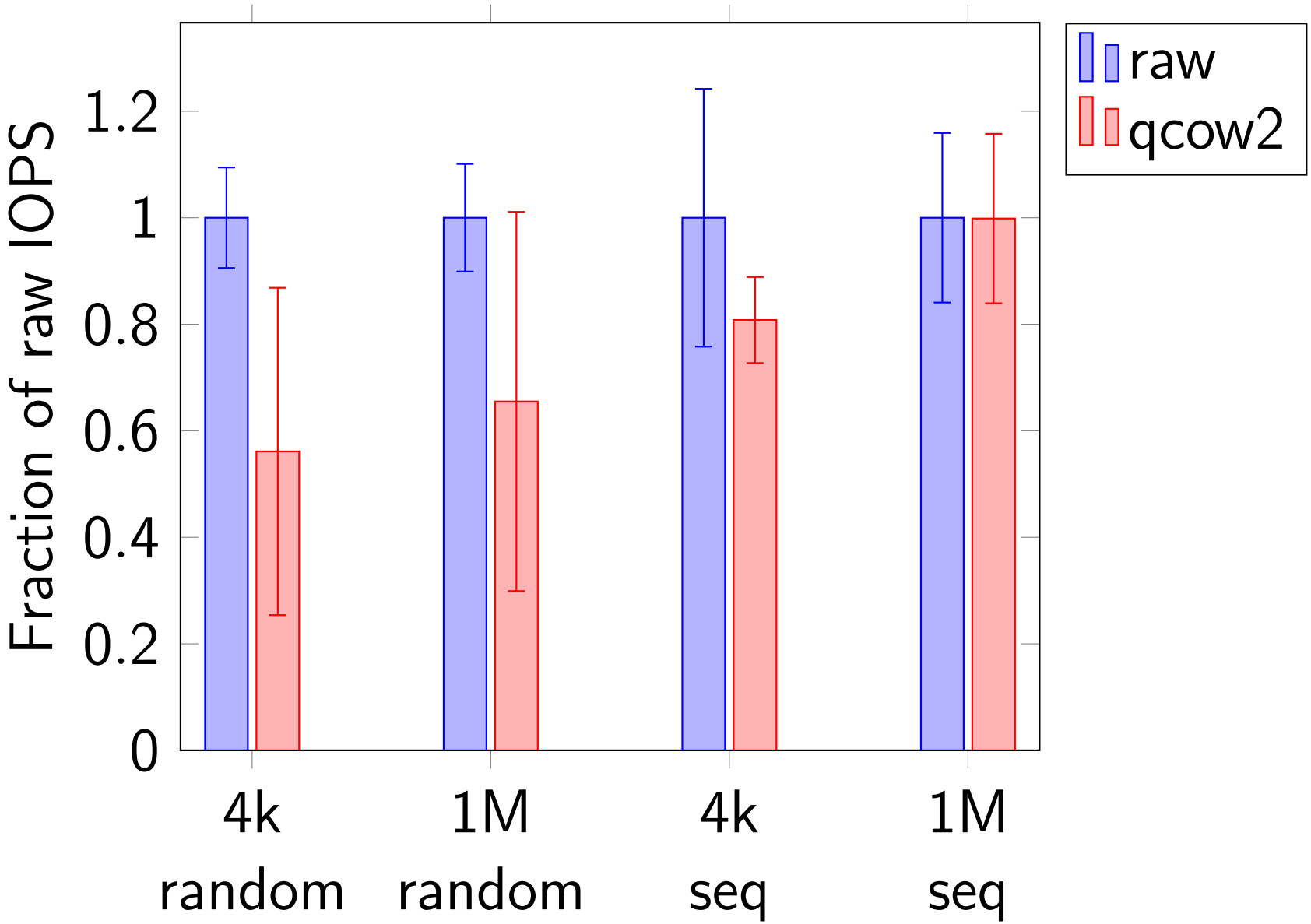
SSD 16 GB image read performance



HDD 32 GB image write performance



HDD 32 GB image read performance



What happened?

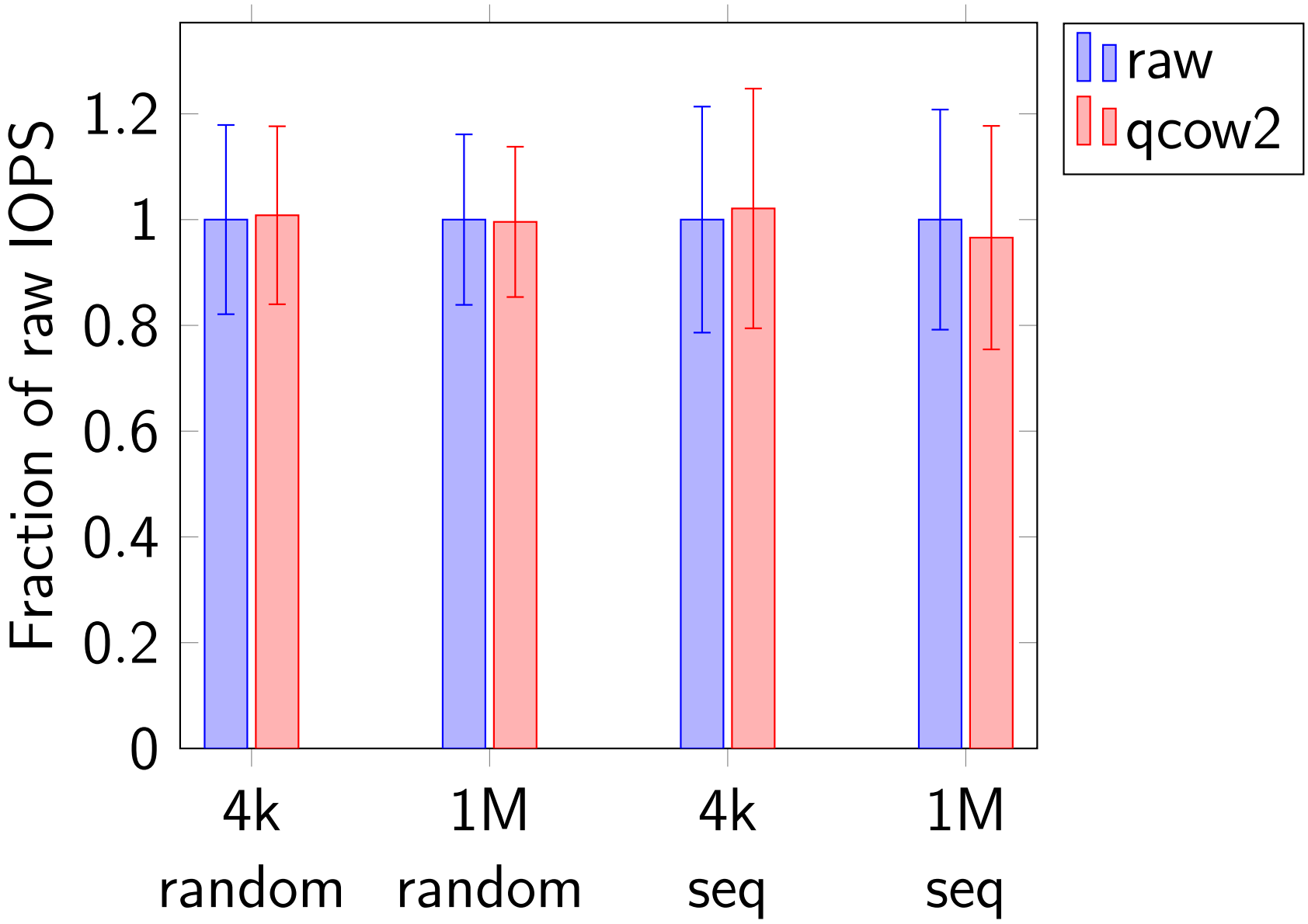
- Cache thrashing happened!
- qcow2 caches L2 tables;
  default cache size: 1 MB
- This covers 8 GB of an image!

How to fix it?

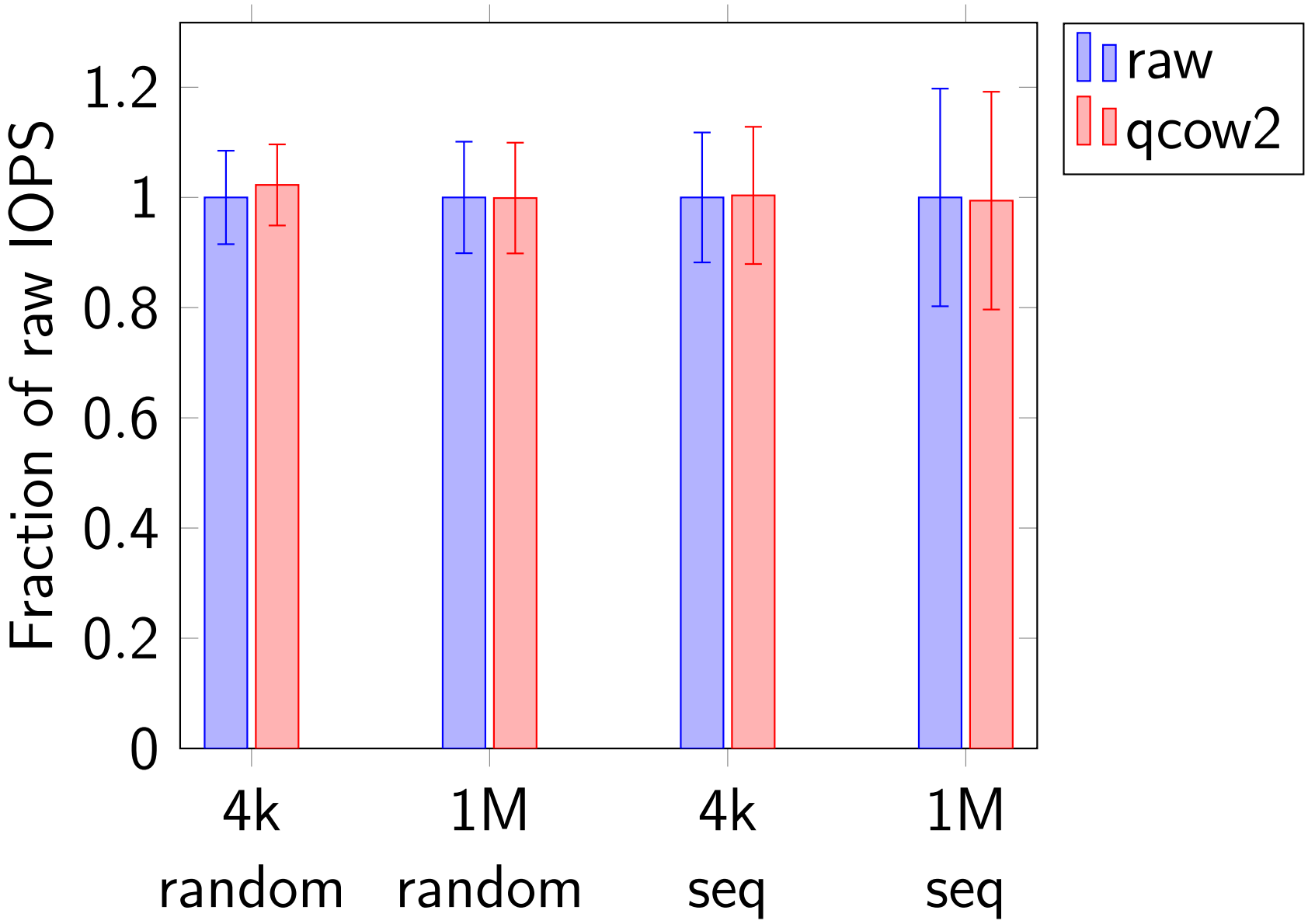
1. DON'T PANIC – Don't fix it.
   - Random accesses contained in an 8 GB area are fine, no matter the image size
2. Increase the cache size
   - `l2-cache-size` runtime option
   - e.g. `-drive format=qcow2,l2-cache-size=4M,...`

$$\frac{\texttt{area size}}{\texttt{cluster size} \div 8} = \frac{\texttt{area size}}{8192\ \texttt{B}}$$

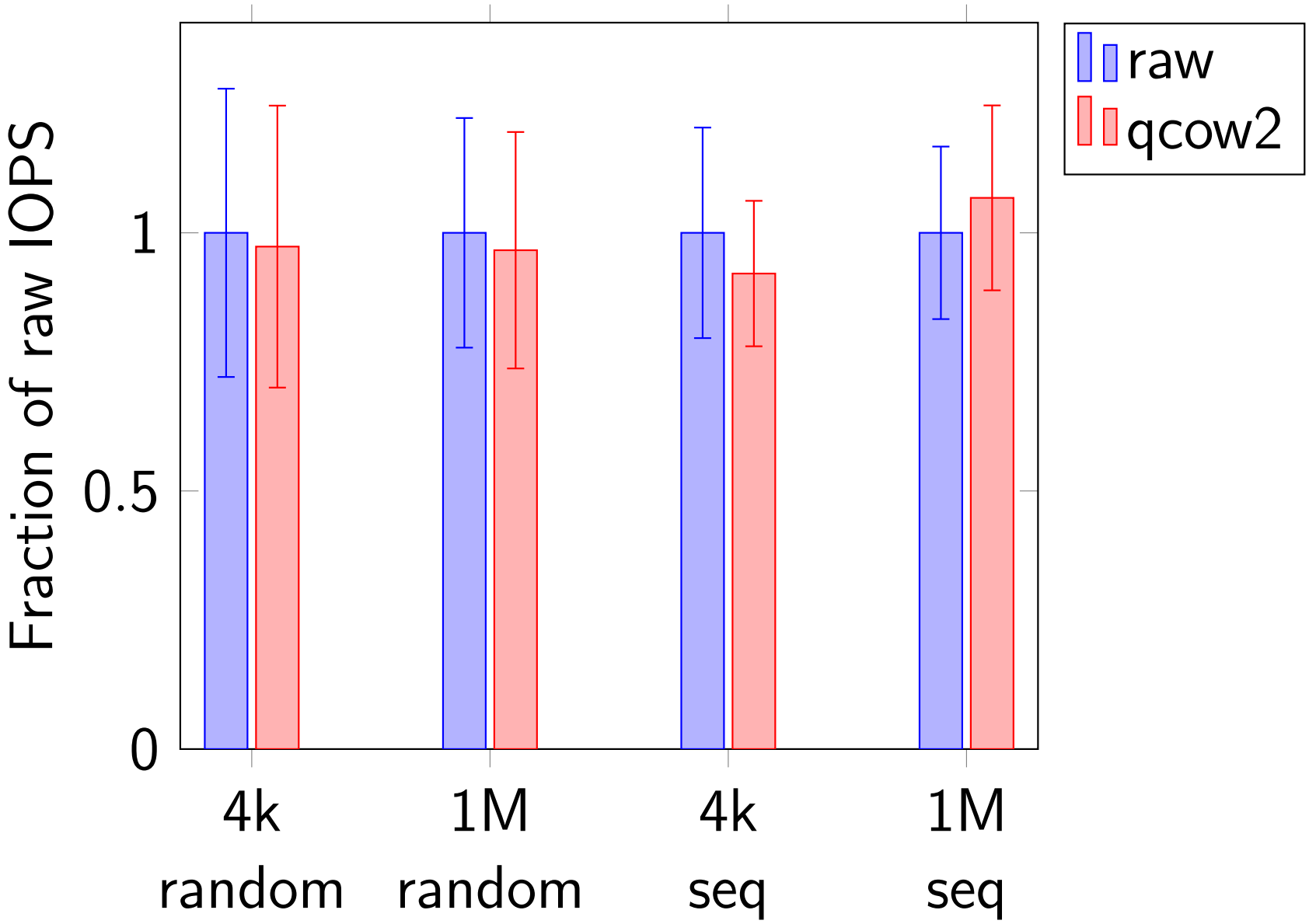
SSD 16 GB image, 2 MB L2 cache, writing



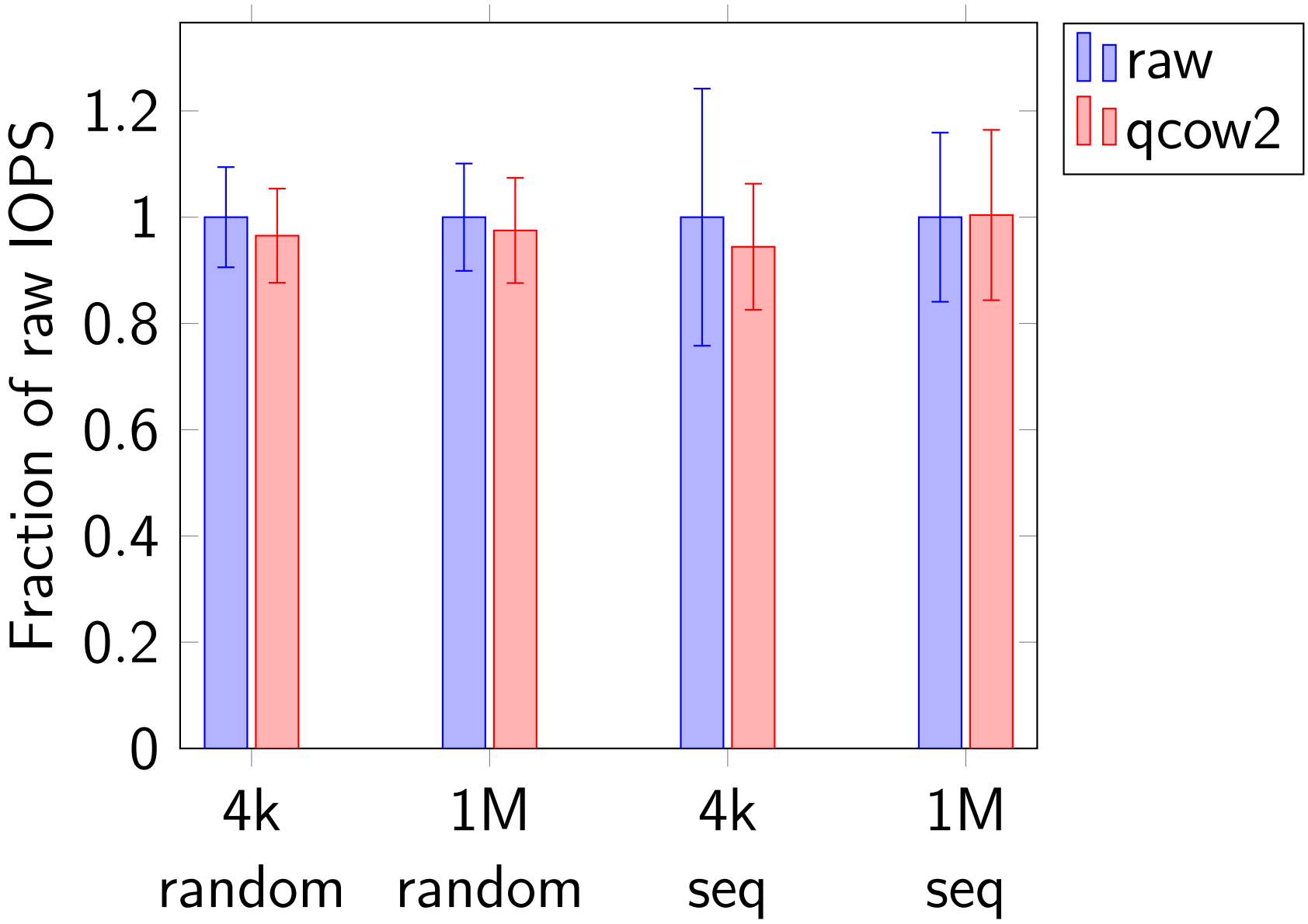
SSD 16 GB image, 2 MB L2 cache, reading



HDD 32 GB image, 4 MB L2 cache, writing



HDD 32 GB image, 4 MB L2 cache, reading



Results

- No significant difference between raw and qcow2 for preallocated images
- . . . As long as the L2 cache is large enough!

- Without COW, everything is good!
- But it *is* named **qcow2** for a reason. . .

redhat.

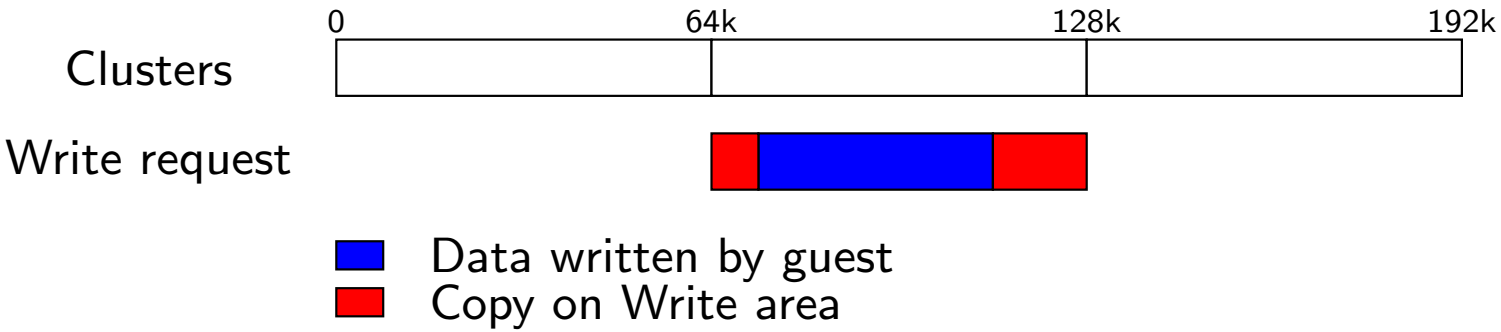
Part III
Cluster allocations

Cluster allocation

When is a new cluster allocated?

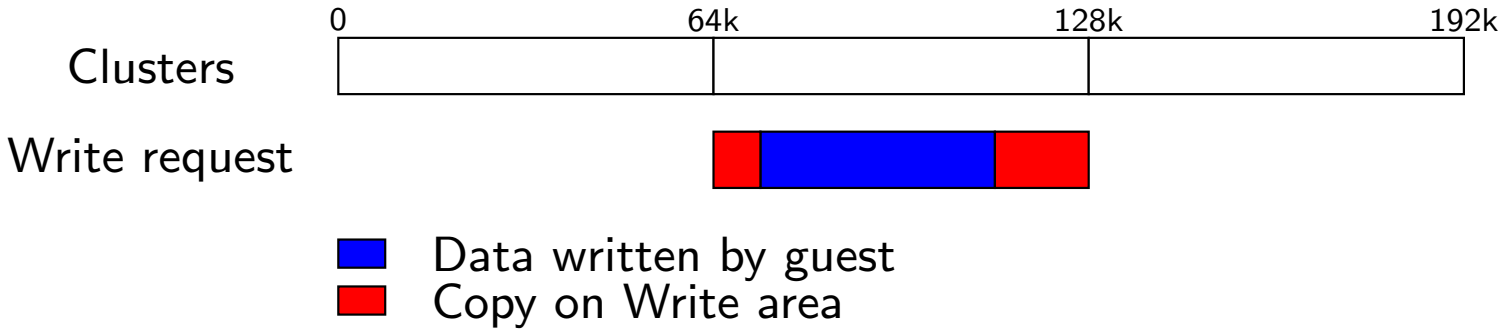
- When writing to unallocated clusters
  - Previous content in backing file
  - Without backing file: all zero
- For COW if existing cluster was shared
  - Internal snapshots
  - Compressed image

Copy on Write



- Cluster content must be completely valid (64k)
- Guest may write with sector granularity (512b)
- Partial write to newly allocated cluster
  → Rest must be filled with old data

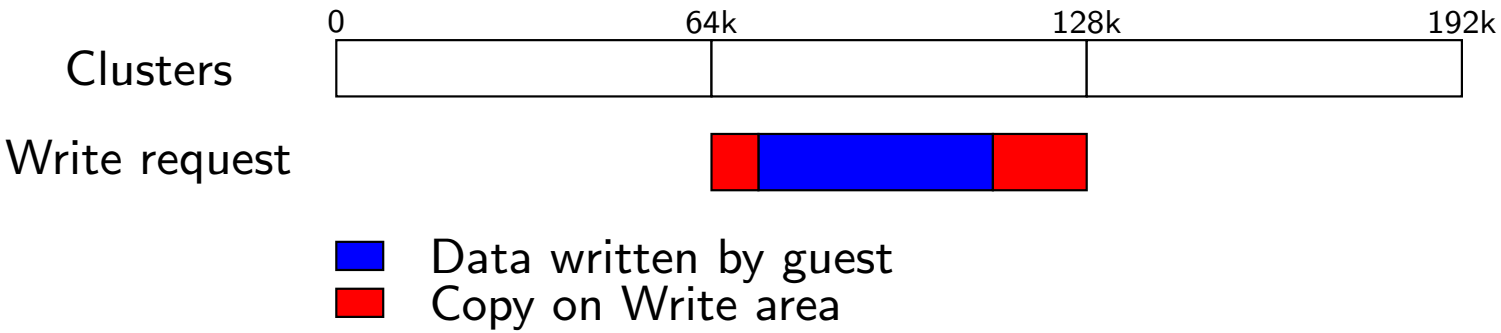
Copy on Write



COW cost is most expensive part of allocations

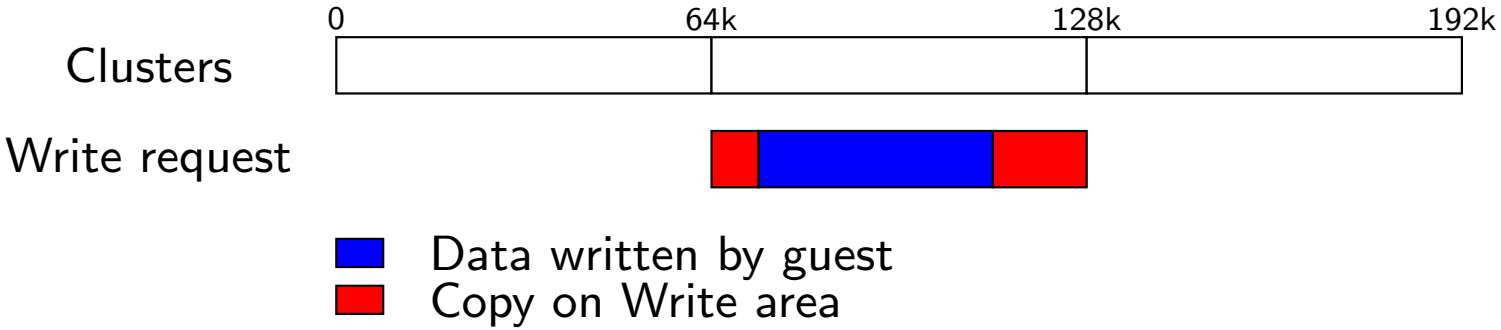
1. More I/O requests
2. More bytes transferred
3. More disk flushes (in some cases)

Copy on Write is slow (Problem 1)



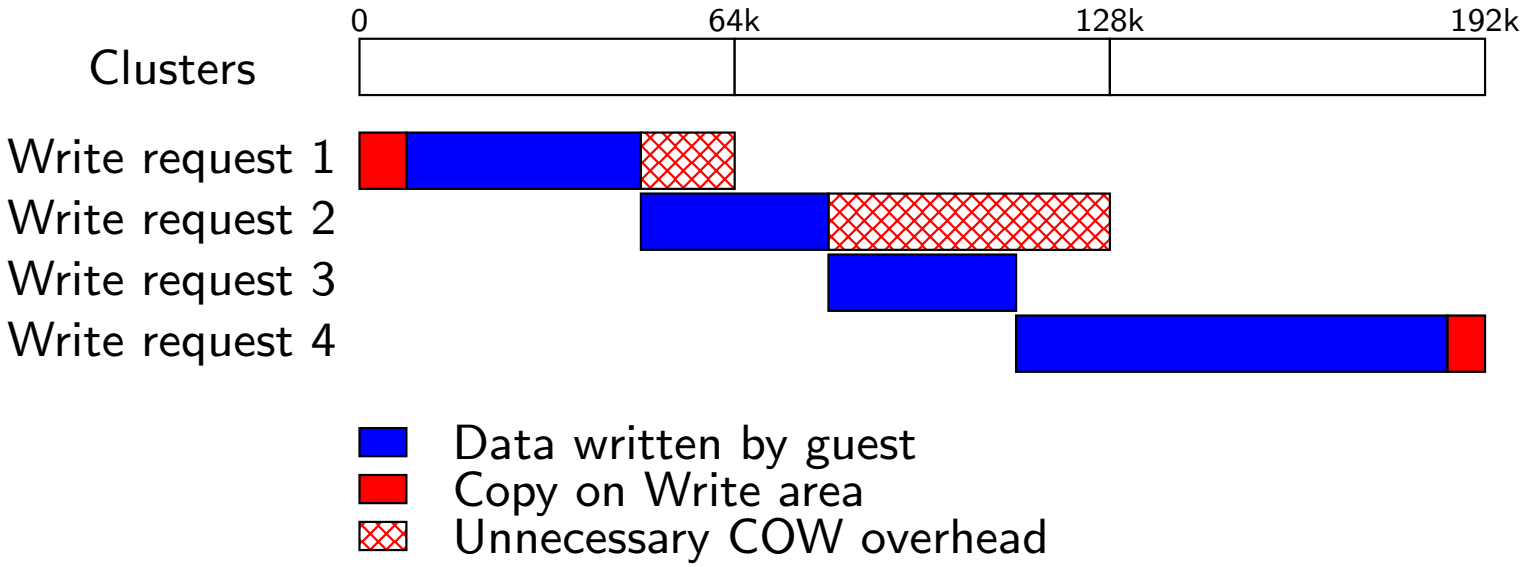
- Naive implementation: 2 reads and 3 writes
- About 30% performance hit vs. rewrite

Copy on Write is slow (Problem 1)



- Can combine writes into a single request
  - Fixes allocation performance without backing file
  - Doesn't fix other cases: read is expensive

Copy on Write is slow (Problem 2)



- Most COW is unnecessary for sequential writes
- If the COW area is overwritten anyway:
  Avoid the copy in the first place

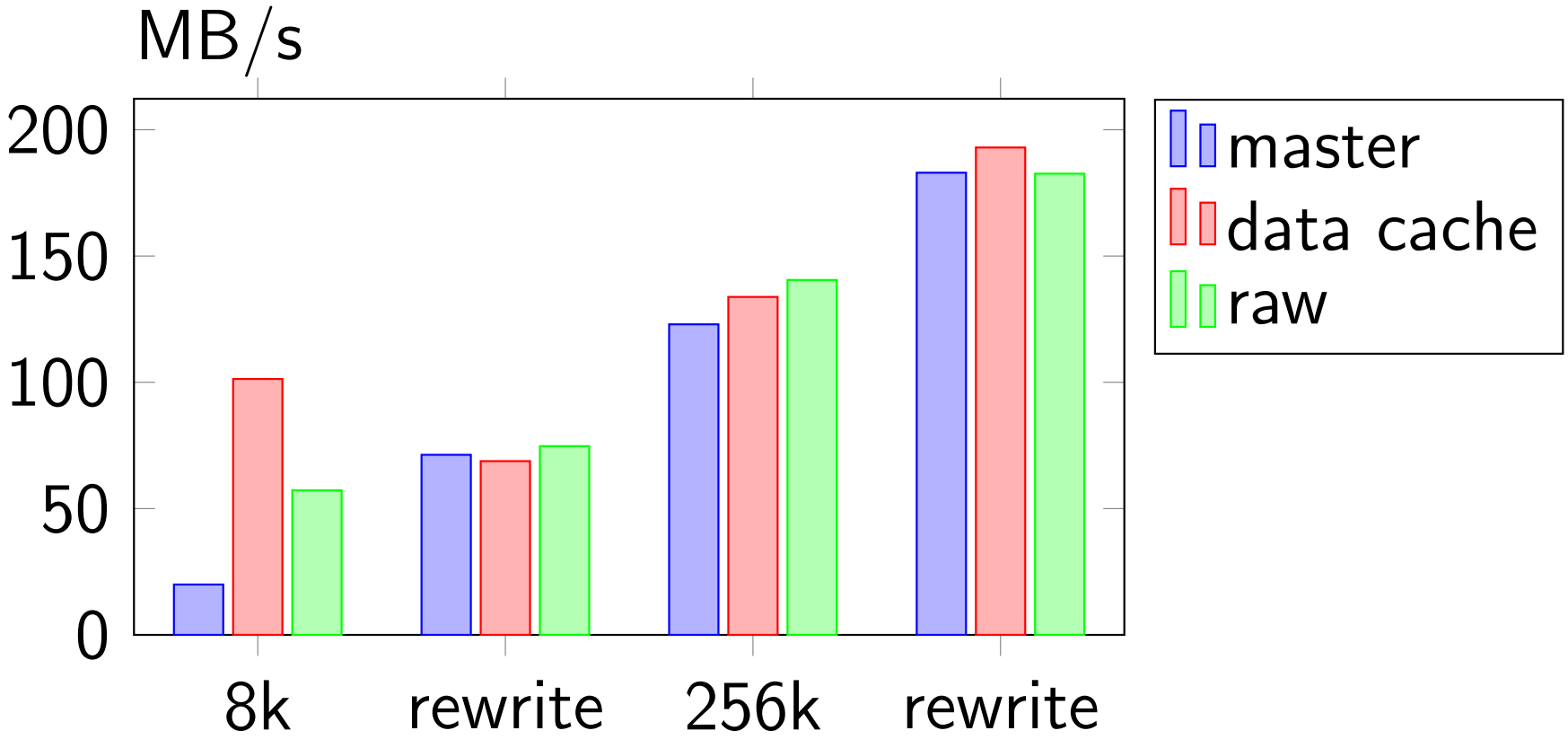
qcow2 data cache

Metadata already uses a cache for batching.
We can do the same for data!

- Mark COW area invalid at first
- Only read from backing file when accessed
- Overwriting makes it valid $\rightarrow$ read avoided

Data cache performance

Seq. allocating writes (qcow2 with backing file)



Copy on Write is slow (Problem 3)

Internal COW (internal snapshots, compression):

1. Allocate new cluster:
   Must increase refcount before mapping update
2. Drop reference for old cluster:
   Must update mapping before refcount decrease

$\rightarrow$ Need two (slow) disk flushes per allocation

Copy on Write is slow (Problem 3)

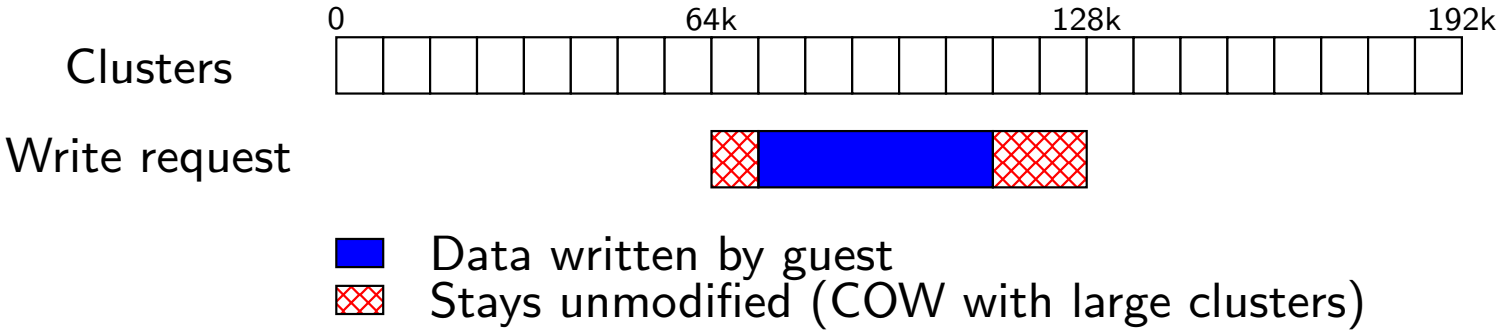
Possible solutions:

- `lazy_refcounts=on`
  allows inconsistent refcounts
- Implement journalling
  allows updating both at the same time

→ No flushes needed

→ Performance fixed

Another solution: Avoid COW

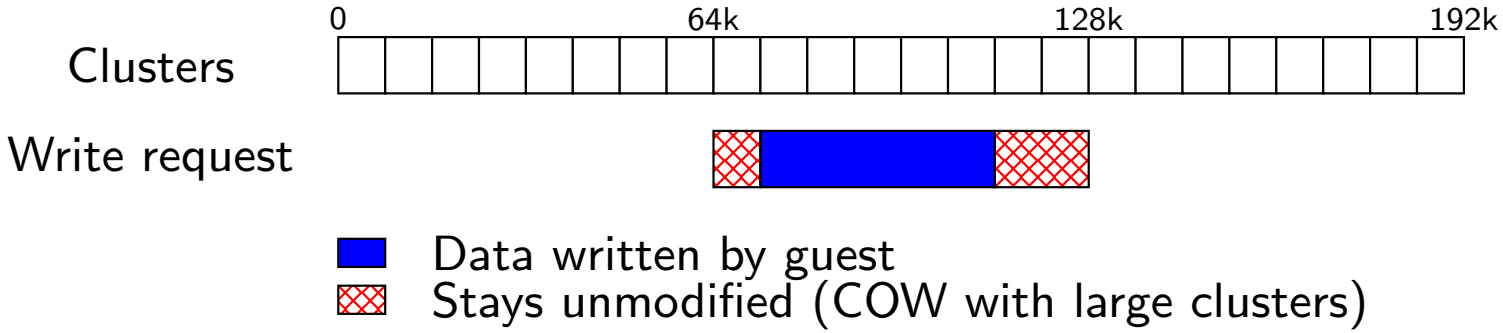


Don't optimize COW, avoid it

$\rightarrow$ Use a small cluster size (= sector size)

redhat.

Another solution: Avoid COW

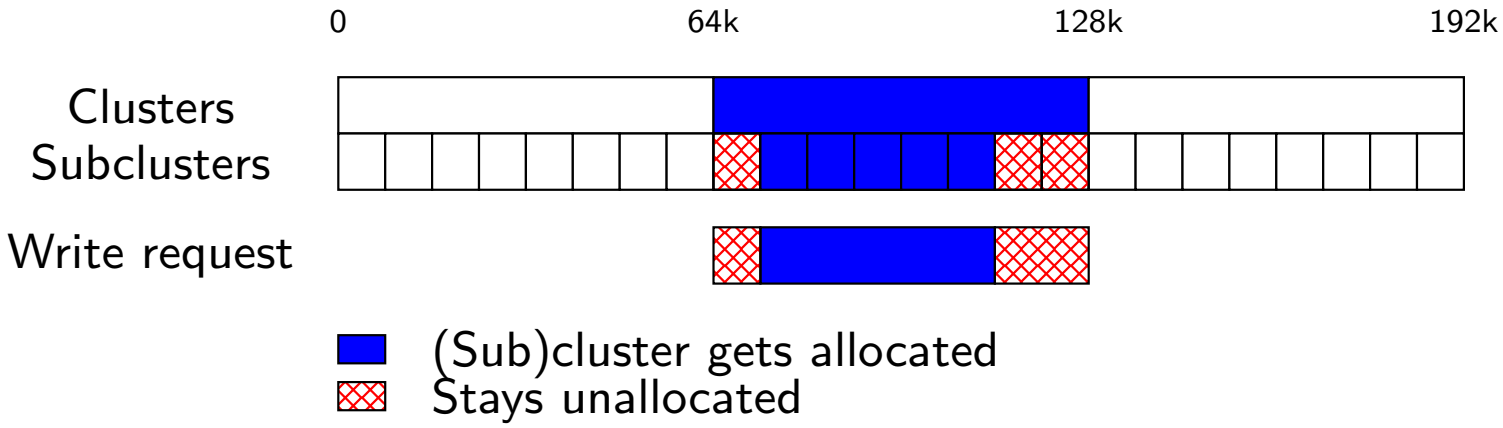


But small cluster size isn't practicable:

- Large metadata (but no larger caches)
- Potentially more fragmentation

→ No COW any more, but everything is slow

Subclusters

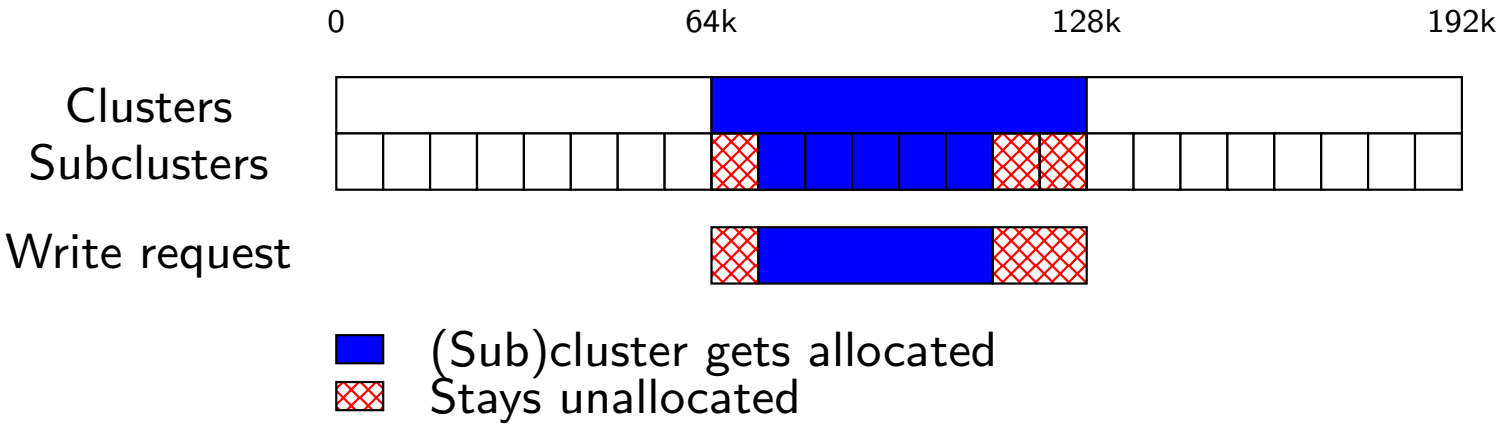


Split cluster size into two different sizes:

- Granularity for the mapping (clusters, large)
- Granularity of COW (subclusters, small)

Add subcluster bitmap to L2 table for COW status

Subclusters



- Requires incompatible image format change
- Can solve problems 1 and 2, but not 3

Status

Data cache:

- Prototype patches exist (ready for 2.5 or 2.6?)

Subclusters:

- Only theory, no code
- Still useful with cache merged

Journalling:

- Not anytime soon
- Use `lazy_refcounts` for internal COW

Questions?