Linux Storage and Virtualization

Christoph Hellwig

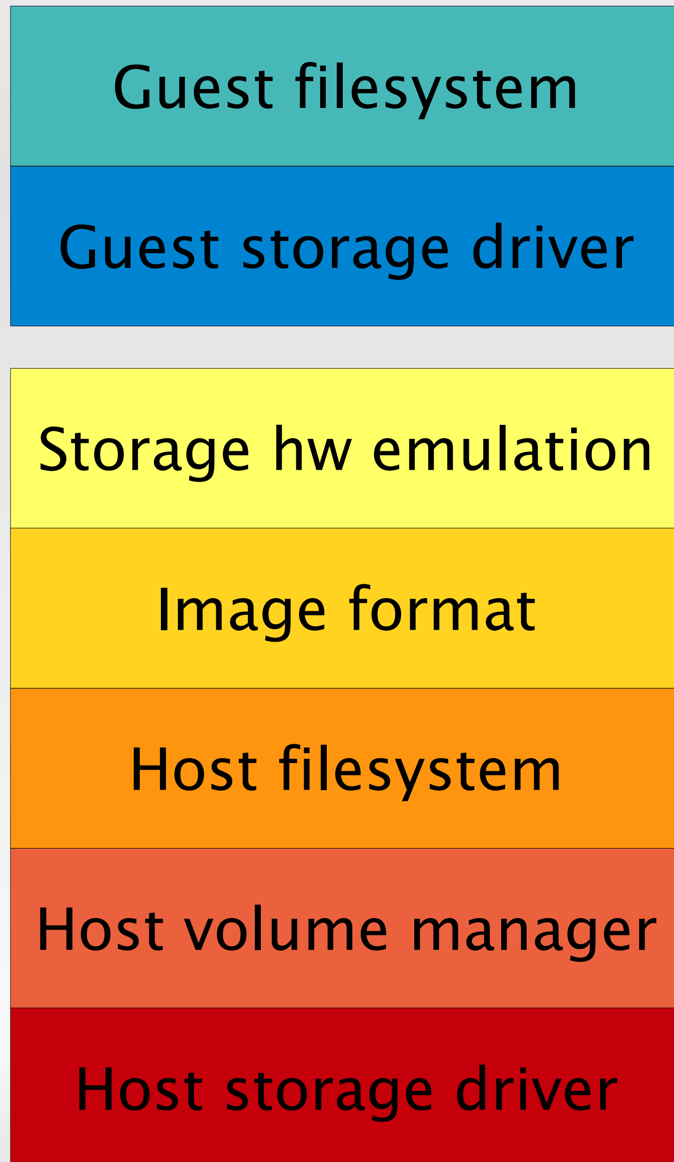
Storage in Virtual Machines – Why?

- A disk is an integral part of a normal computer
  - Most operating systems work best with local disks
  - Boot from NFS / iSCSI still has problems
- Simple integrated local storage is:
  - Easier to setup than network storage
  - More flexible to manage
- For the highend use PCI passthrough or Fibre channel NPIV instead

A view 10.000 feet

- The host system or virtual machine monitor (VMM):
  - exports virtual disks to the guest
  - The guest uses them like real disks
- The virtual disks are backed by real devices..
  - Whole disks / partitions / logical volumes
- .. or files
  - Either raw files on a filesystem or image formats

A virtual storage stack



- We have two full storage stacks in the host and in the guest
  - Potentially also two filesystem
  - Potentially also a image format (aka mini filesystem)

Requirements (high level)

- The traditional storage requirements apply:
  - Data integrity – data should actually be on disk when the user / application require it
  - Space efficiency – we want to store the user / application data as efficient as possible
  - Performance – do all of the above as fast as possible
- Additionally there is a strong focus on:
  - Manageability – we potentially have a lots of hosts to deal with

Requirements – guest

- None – Guests should work out of the box
- Migrating old operating system images to virtual machines is a typical use case
- Any guest changes should be purely optimizations for:
  - Storage efficiency or
  - Performance

Requirements – host

- The host is where all the intelligence sits
- Ensures data integrity
  - Aka: the data really is on disk when the guest thinks so
- Optimizantions of storage space usage

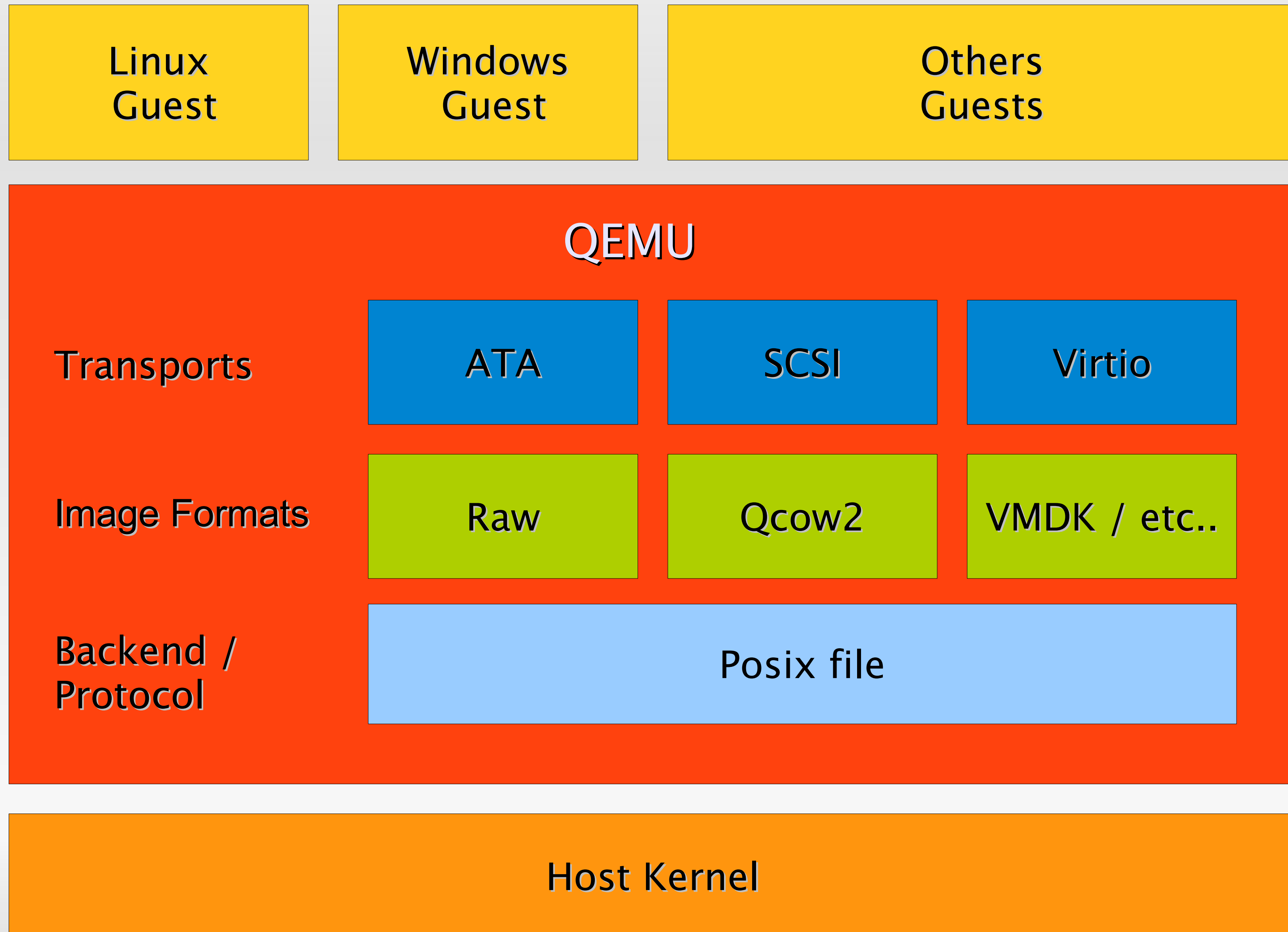
A practical implementation: QEMU/KVM

- KVM is the major virtualization solution for Linux
- Included in the mainline kernel, with lots of development from RedHat, Novell, Intel, IBM and various individual contributors

What is QEMU and what is KVM?

- QEMU primarily is a CPU emulator
- Grew a device model to emulate a whole computer
  - Actually not just one but a whole lot of them
- KVM is a kernel module to use expose hardware virtualization capabilities
  - e.g. Intel VT-x or AMD SVM
  - KVM uses QEMU for device emulation
- As far as storage is concerned they're the same

QEMU Storage stack overview



Storage transports

- QEMU provides a simple Intel ATA controller emulation by default
  - Works with about every operating systems because it is so common
- Alternatively QEMU can emulate a Symbios SCSI controller

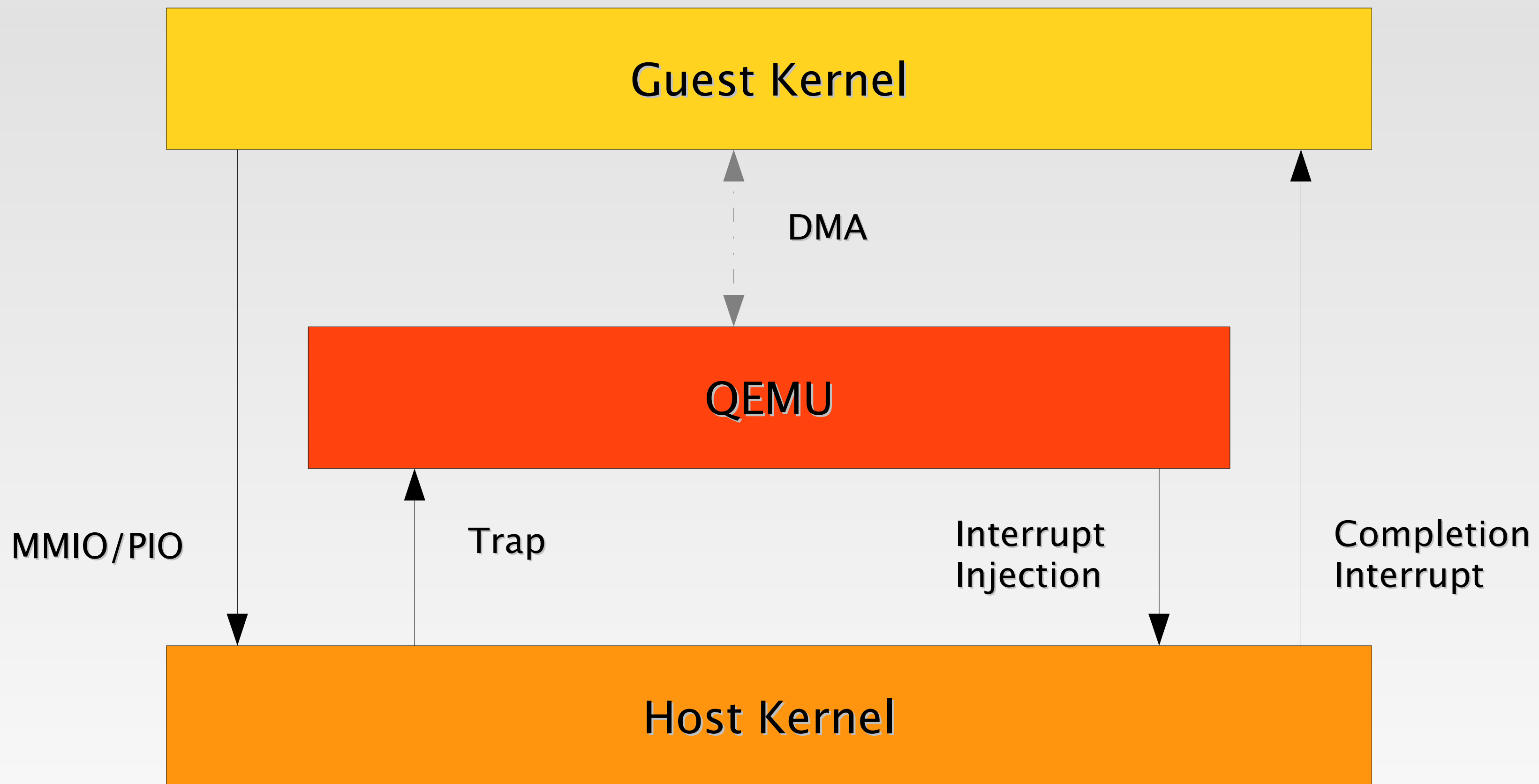
Paravirtualization

- Paravirtualization means providing interfaces more optimal than real hardware
  - Advantage: should be faster than full virtualization
  - Disadvantage: requires special drivers for each guest

Paravirtualized storage transport

- QEMU provides paravirtualized devices using the virtio framework
- Virtio-blk provides a simple block driver ontop of virtio
  - Just simple read/write requests
  - And SCSI requests through ioctls
  - ...

Life of an I/O request



Posix file storage backend

- The primary storage backend
  - Almost all I/O eventually ends up there
- Simply backs disk images using a regular file or device file

Posix storage backend – AIO

- The qemu main loop is effectively singe threaded:
  - Time spent there blocks execution of the guest
  - I/O needs to be offloaded as fast as possible
- I/O backends need to implement asynchronous semantics

Posix storage backend – AIO

- AIO support in hosts is severly lacking
  - Use a thread pool to hand off I/O by default
- Alternatively support for native Linux AIO:
  - Only works for uncached access (O_DIRECT)
  - Still can be synchronous for many use cases

Posix storage backend – vectored I/O

- Typical I/O requests from guests are split into non-contingous parts
  - scatter/gather lists
- In the optimal case a whole SG list is sent to the host kernel in one request
- preadv/pwritev system calls

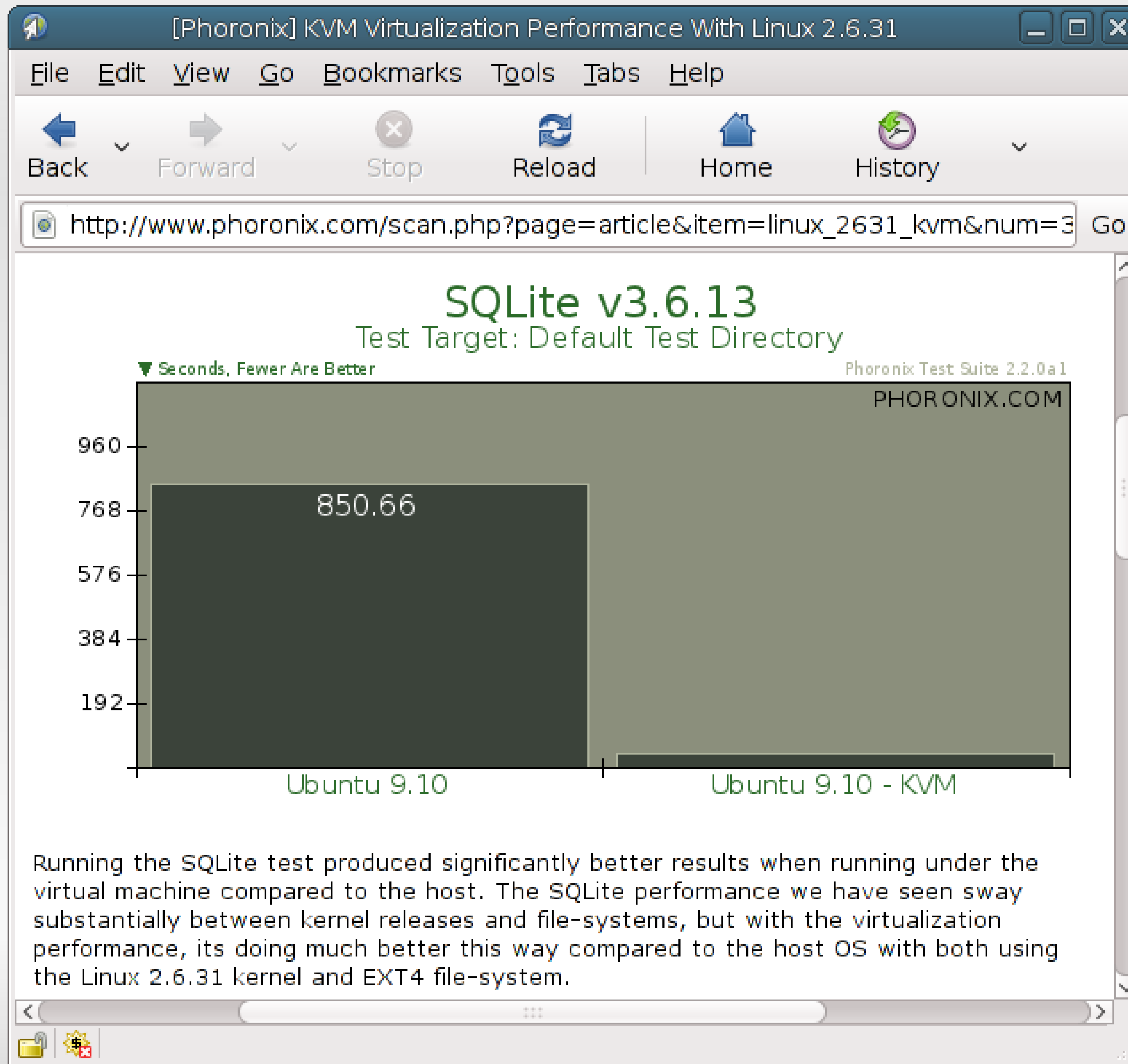
Posix storage backend – AIO

- AIO support in hosts is severly lacking
  - Use a thread pool to hand off I/O by default
- Alternatively support for native Linux AIO:
  - Only works for uncached access (O_DIRECT)
  - Still can be synchronous for many use cases

Posix storage backend – I/O restrictions

- O_DIRECT requires strict alignment and specific I/O sizes
  - Try to align memory allocations inside Qemu
  - The Posix backend needs to perform read/modify/write cycles in the worst case
- The alignment and size restrictions vary
  - There is no proper way to query the kernel for the restrictions
  - In Linux they generally depend on the sector size

Spot the error..



Data integrity in QEMU / caching modes

- cache=none
  - uses O_DIRECT I/O that bypasses the filesystem cache on the host
- cache=writethrough
  - uses O_SYNC I/O that is guaranteed to be commited to disk on return to userspace
- cache=writeback
  - uses normal buffered I/O that is written back later by the operating system

Data integrity – cache=writethrough

- This mode is the safest as far as qemu is concerned
  - There are no additional volatile write caches in the host
- The downside is that it's rather slow

Data integrity – cache=writeback

- When the guest writes data we simply put it in the filesystem cache
  - No guarantee that it actually goes to disk
  - Which is actually very similar to how modern disks work

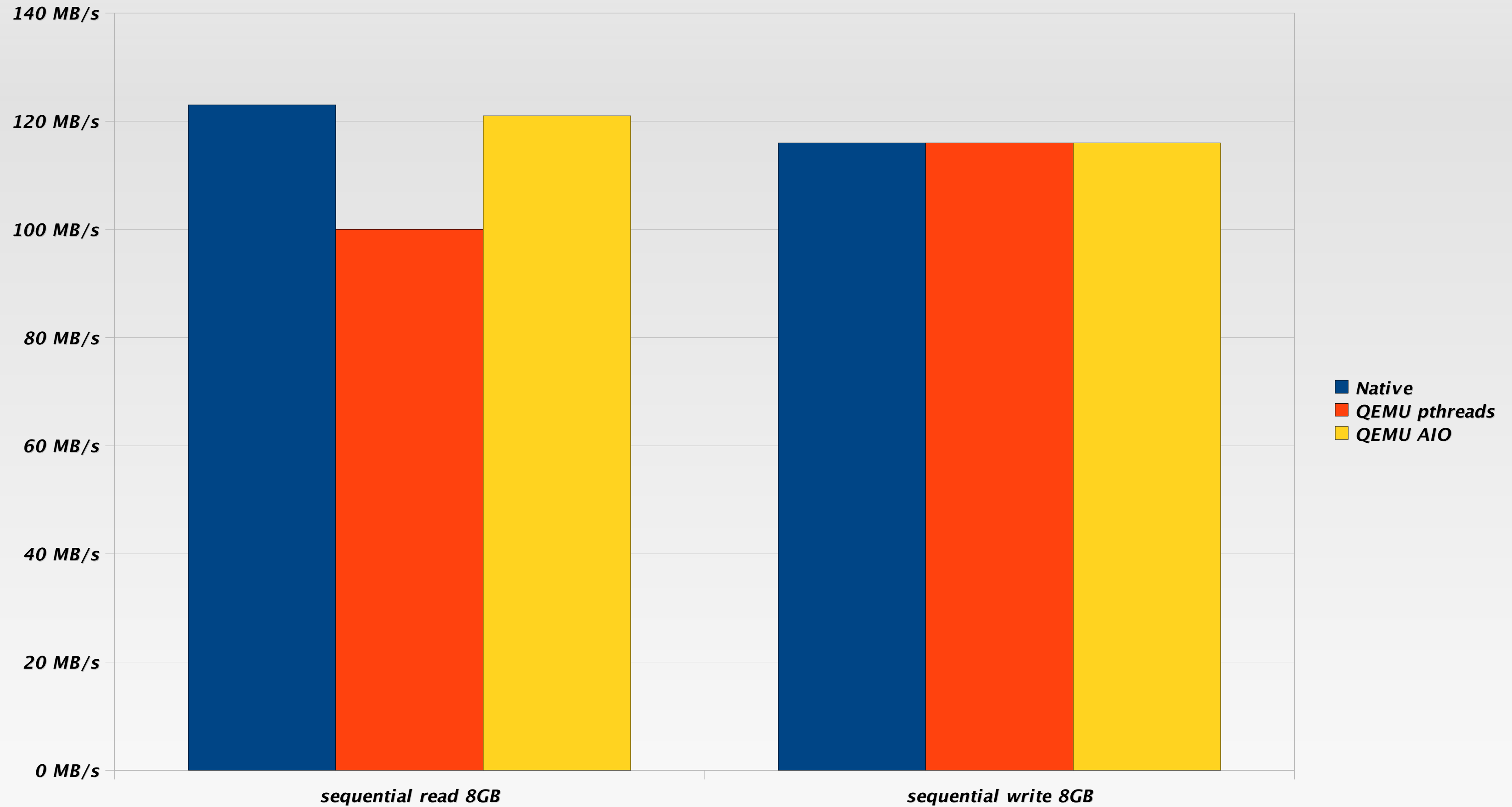
Data integrity – cache=writeback

- The guest needs to issue a cache flush command to make sure data goes to disk
  - Similar to real modern disks with writeback caches
  - Modern operating systems can deal with this
- And the host needs to actually implement the cache flush command and advertise it:
  - The QEMU SCSI emulation has always done this
  - IDE and virtio only started this very recently

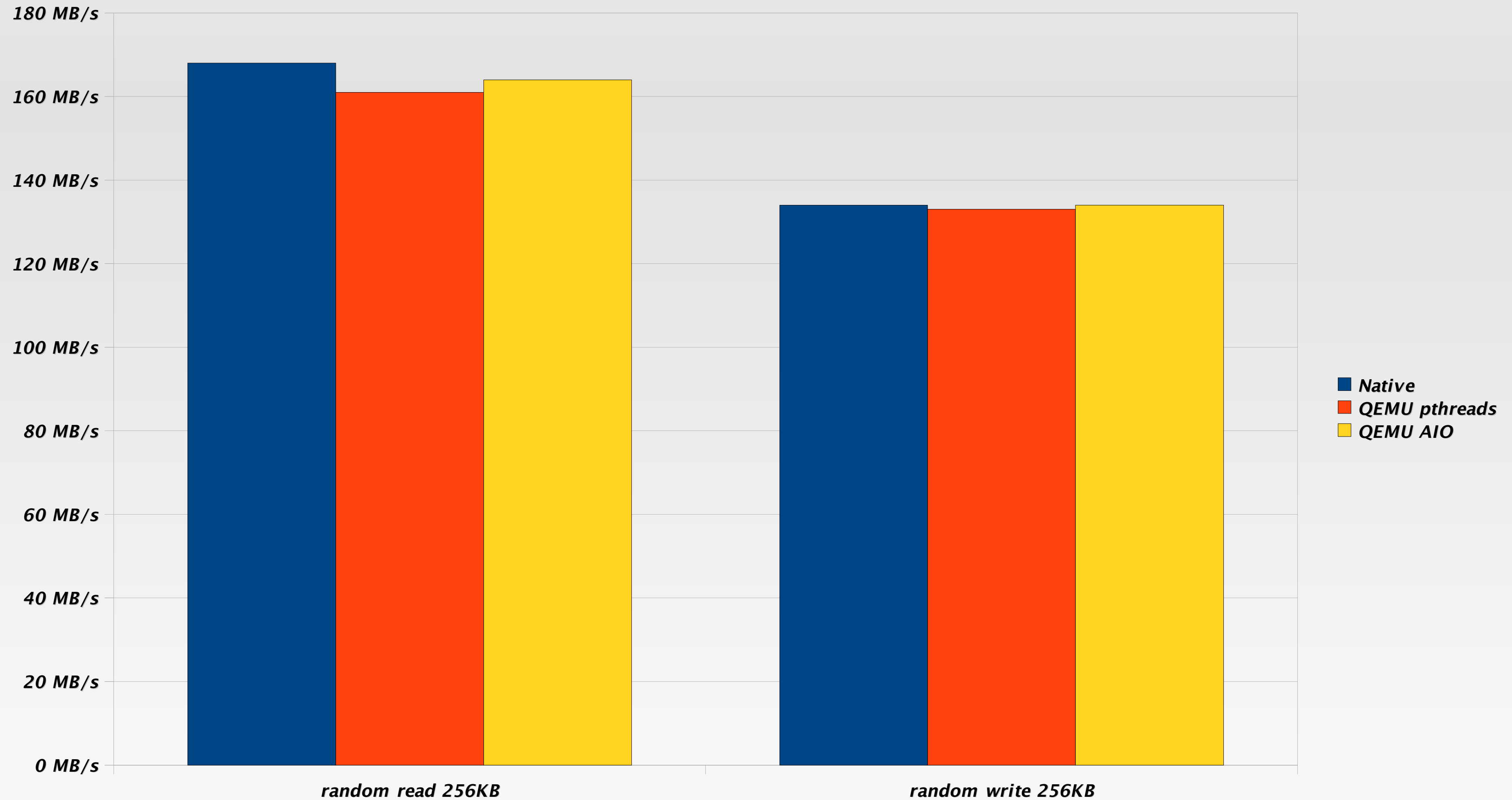
Data integrity – cache=none

- Direct transfer to disk should imply it's safe
- Except that it is not:
  - Does not guarantee disk caches are flushed
  - Does not give any guarantees about metadata
- Thus also needs an explicit cache flush

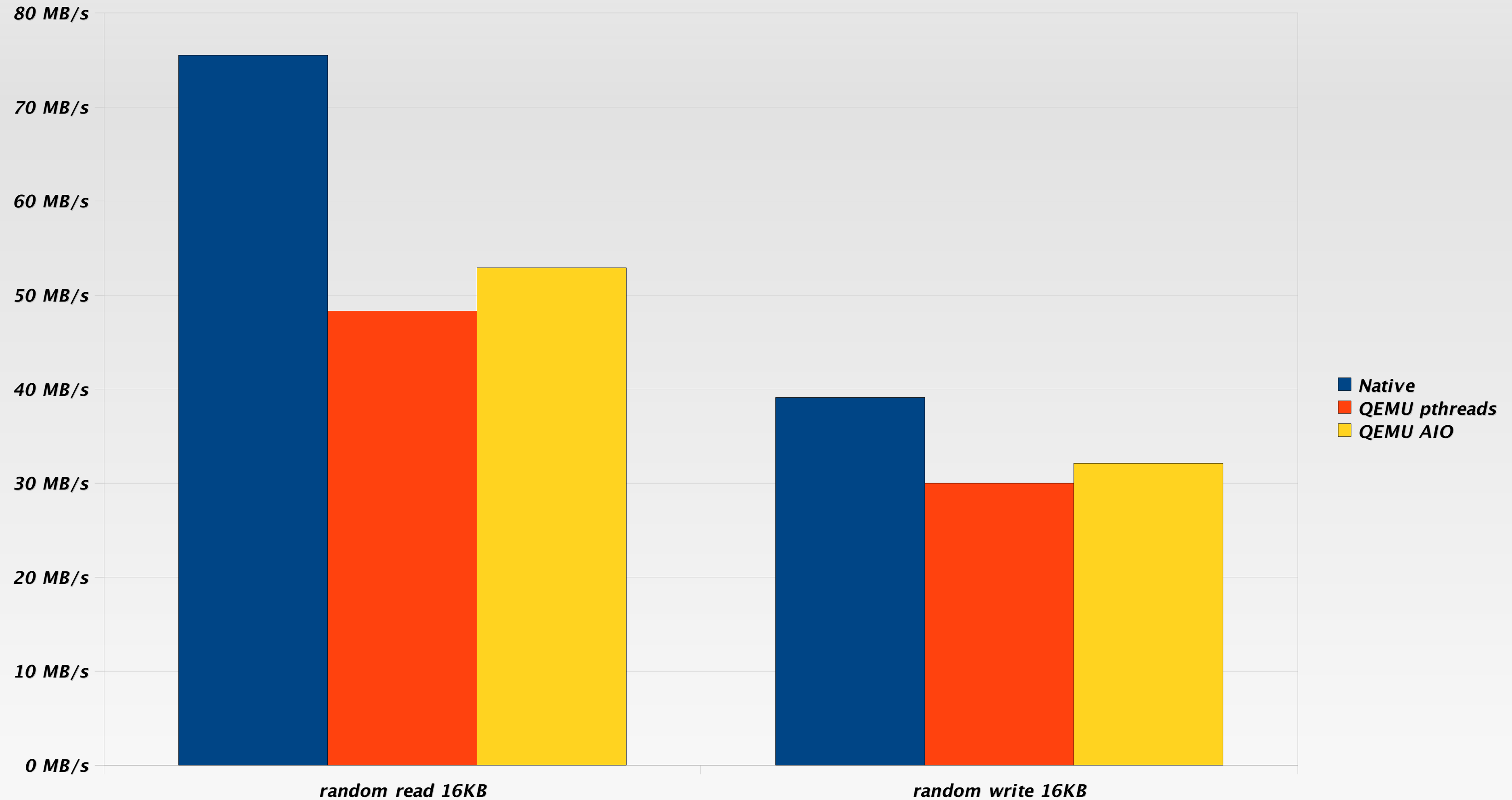
Performance – large sequential I/O



Performance – 256 kilobyte random I/O



Performance – 16 kilobyte random I/O



Disk image formats

- Users want volume-manager like features in image files
  - Copy-on write snapshots
  - Encryption
  - Compression
- Also VM snapshots need to store additional metadata

Disk Image formats – Qcow2

- Qcow was the initial QEMU image format to provide copy on write snapshots
- In Qemu 0.8.3 Qcow2 was added to add additional features and now is the standard image format for QEMU
  - Provides cluster based copy on write snaphots
  - Supports encryption and compression
  - Allows to store additional metadata for VM snaphots

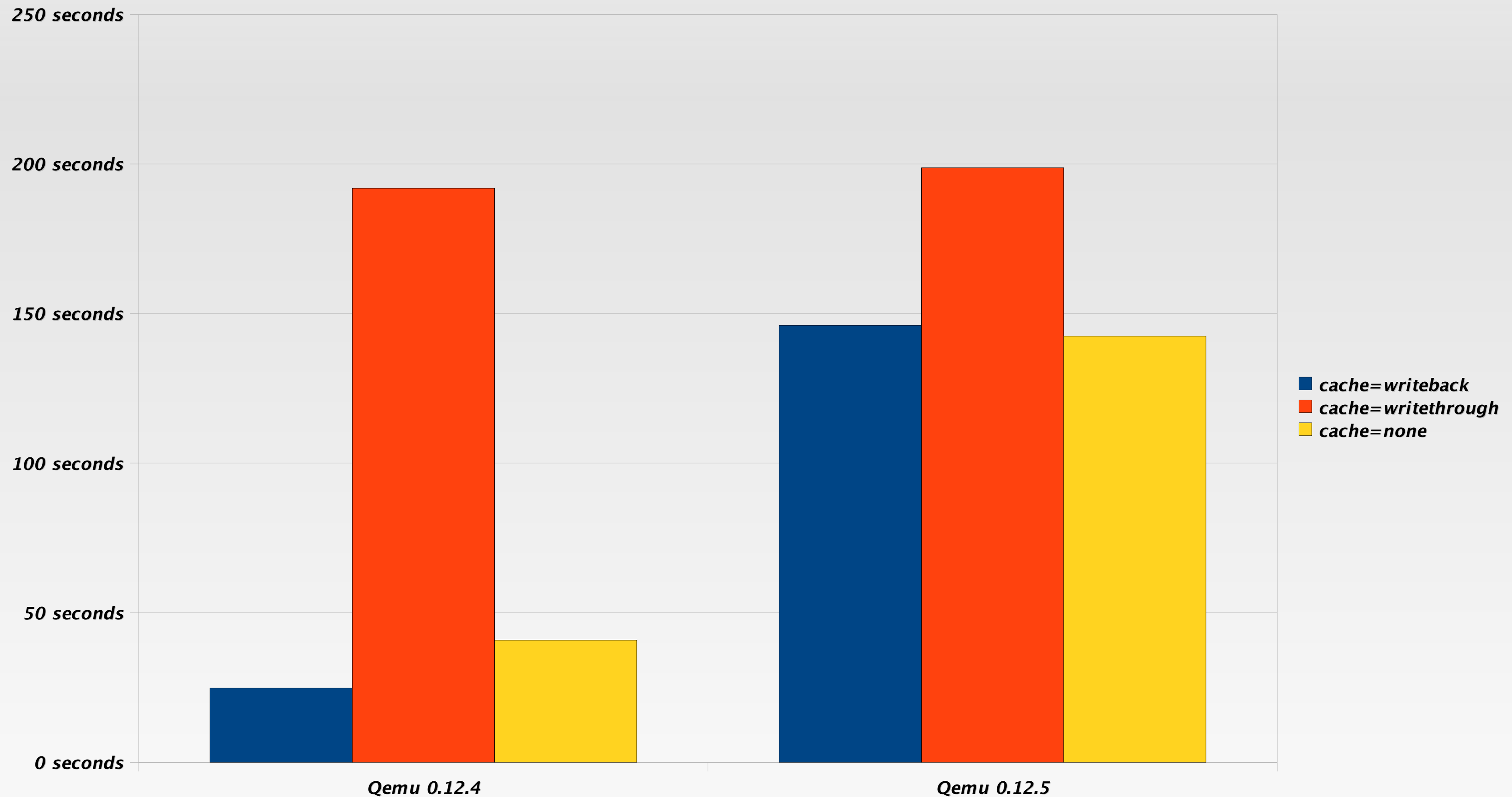
Disk image format data integrity issues

- A disk image is a minimal filesystem
  - Metadata for block allocation tables
  - Reference counts for snapshots
- So the same integrity issues apply:
  - A guest cache flush needs to guarantee all metadata updates are on disk
  - Multiple metadata updates need to be ordered
  - Multiple metadata updates should be transactional or a image check is required on an unclean shutdown

Qcow2 data integrity issues

- Until recently Qcow2 did not care about metadata integrity.
- Recently Qcow2 was changed to write metadata synchronously.
  - Performance for some workloads decreased to a large extent.
- Work is under way to implement metadata integrity more efficiently.

Qcow2 performance – streaming write



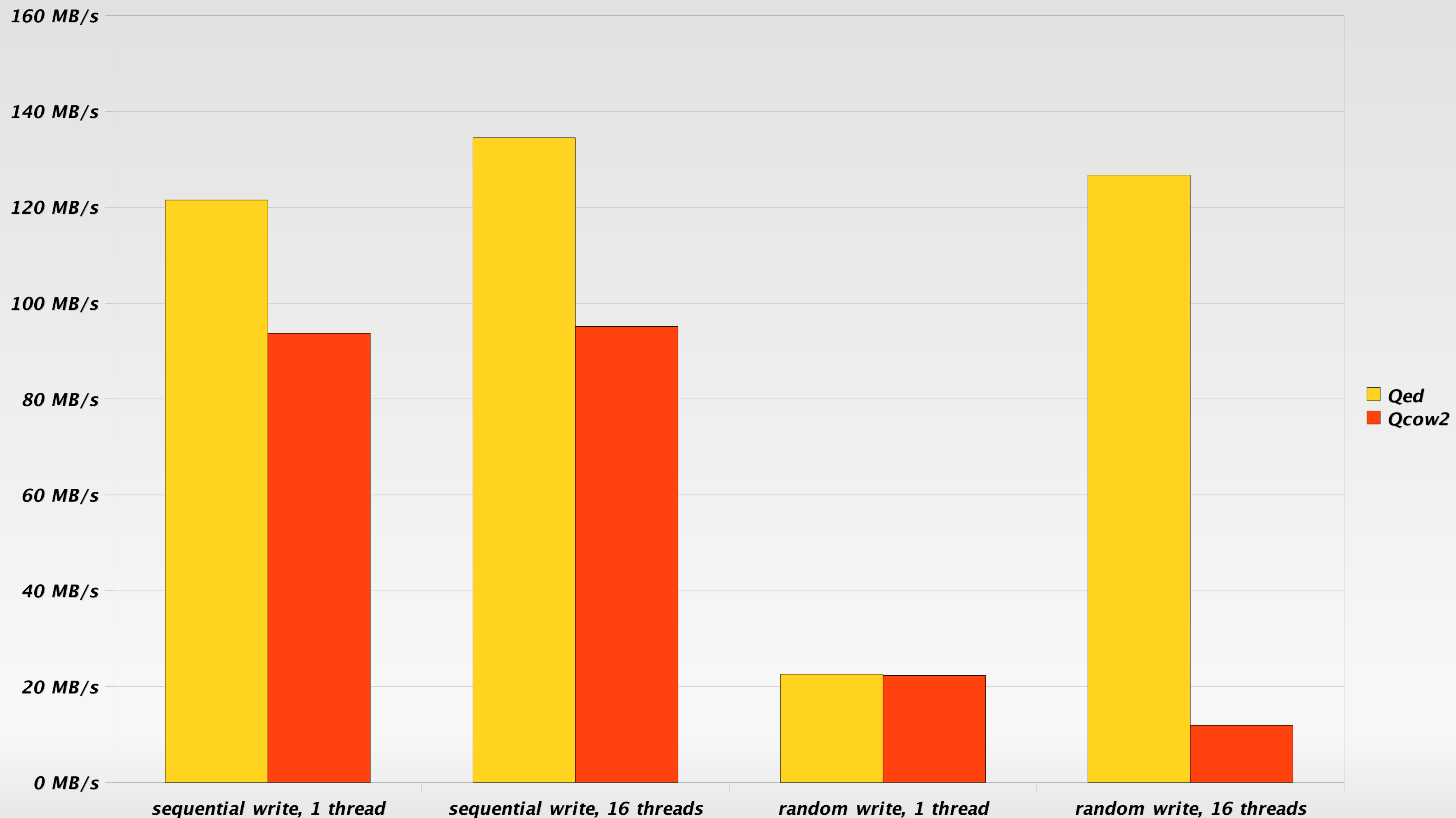
1GB streaming write using dd bs=1024k count=1024

Numbers from Alexander Loob <psionyx@gmx.de>

Qed image format

- New simplified image format proposed by IBM in September 2010:
  - No support for internal snapshots, encyption, compression.
  - Requires sparse file support.
- Initial implementation supports very efficient metadata operations
  - But requires a image check on unclean shutdown.

Qcow2 vs Qed performance



FFSB on ext4 in the guest

Numbers from Khoa Huynh <khoa@us.ibm.com>

Storage protocol drivers

- Qemu allows other images to reside outside the local filesystem.
- Protocol drivers implement the storage access:
  - The **curl** backend allows using VM images from the internet over http and ftp connections.
  - The **nbd** backend allows direct access to nbd servers.
  - The **sheepdog** backs images by a distributed storage protocol.

Thin provisioning

- Simple example: a sparse image file
  - Initially does not have blocks allocated to it
  - Block get allocate on the first write
- To make it fully useful also needs to support reclaiming space after deletions
- An important topic both for high-end storage arrays and virtualization

Thin provisioning – standards

- The T10 SBC standard for SCSI disks / storage arrays contains TP support in it's newest revisions
  - The UNMAP and WRITE SAME commands allow telling the storage device to free data
  - Perfect use case for qemu to know that the guest has freed the storage
- The ATA spec has a similar TRIM command for Solid State Drives (SSDs)

Thin provisioning – implementation

- The guest needs extensive enablement for thin provisioning:
  - Support in the device drivers to actually send the commands
  - Code in the filesystem to track deleted space
  - Guest enablement is shared with support for thinkly provisionen RAID arrays and SSDs

Thin provisioning – implementation

- Decoding the **WRITE SAME** / **UNMAP** / **TRIM** commands is easy
- Actually freeing space is harder:
  - The standard Posix APIs don't allow punching holes into files
  - Need filesystem specific extensions for that (e.g. in **XFS**)

Thin provisioning – implementation

- Fine grained allocation and freeing of blocks is problematic:
  - Causes fragmentation of the backing file
  - Allocation overhead can be high
- Need good thresholds for freeing blocks
  - Similar problem faced by storage arrays
  - SBC allows to communicate these thresholds
- Block allocation also needs the same thresholds

Storage efficiency

- A big virtualization specific problem is to avoid duplicate storage of data data:
  - Often many similar guests run on the same host
- Two approaches:
  - Image clones – start with a common image and track changes with a copy on write scheme
  - Data deduplication – find duplicate blocks and merge them after the fact

Backing images

- QEMU allows for copy on write images in the QCOW2 format.
  - Simple to set up and use
- LVM supports copy on write volumes (snapshots)
  - Requires the usage of full block devices,
- Some modern filesystems (**`btrfs`**, **`ocfs2`**) allow file level snapshots
- None of the above two integrated with QEMU yet

Data deduplication

- All the above options have one disadvantage:
  - The data sharing needs to be planned ahead
- The term data deduplication is used for the process of finding these duplicates later and merging them
  - A relatively expensive and slow process without additional metadata
  - Hot topic in the storage industry
  - Not yet implemented in QEMU or lower layers

Questions?

- Thanks for your attention!
- Feel free to contact me at: **hch@lst.de**