The KVM/qemu storage stack

Christoph Hellwig

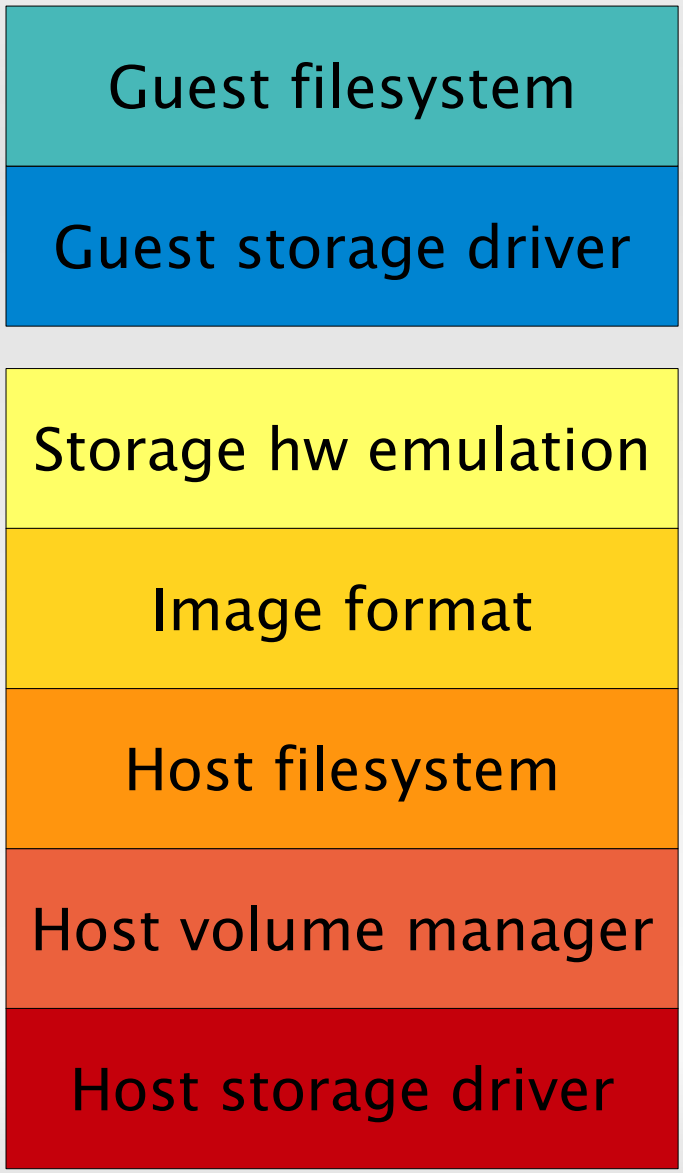
Storage in Virtual Machines – Why?

- 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 (VM):
  - 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

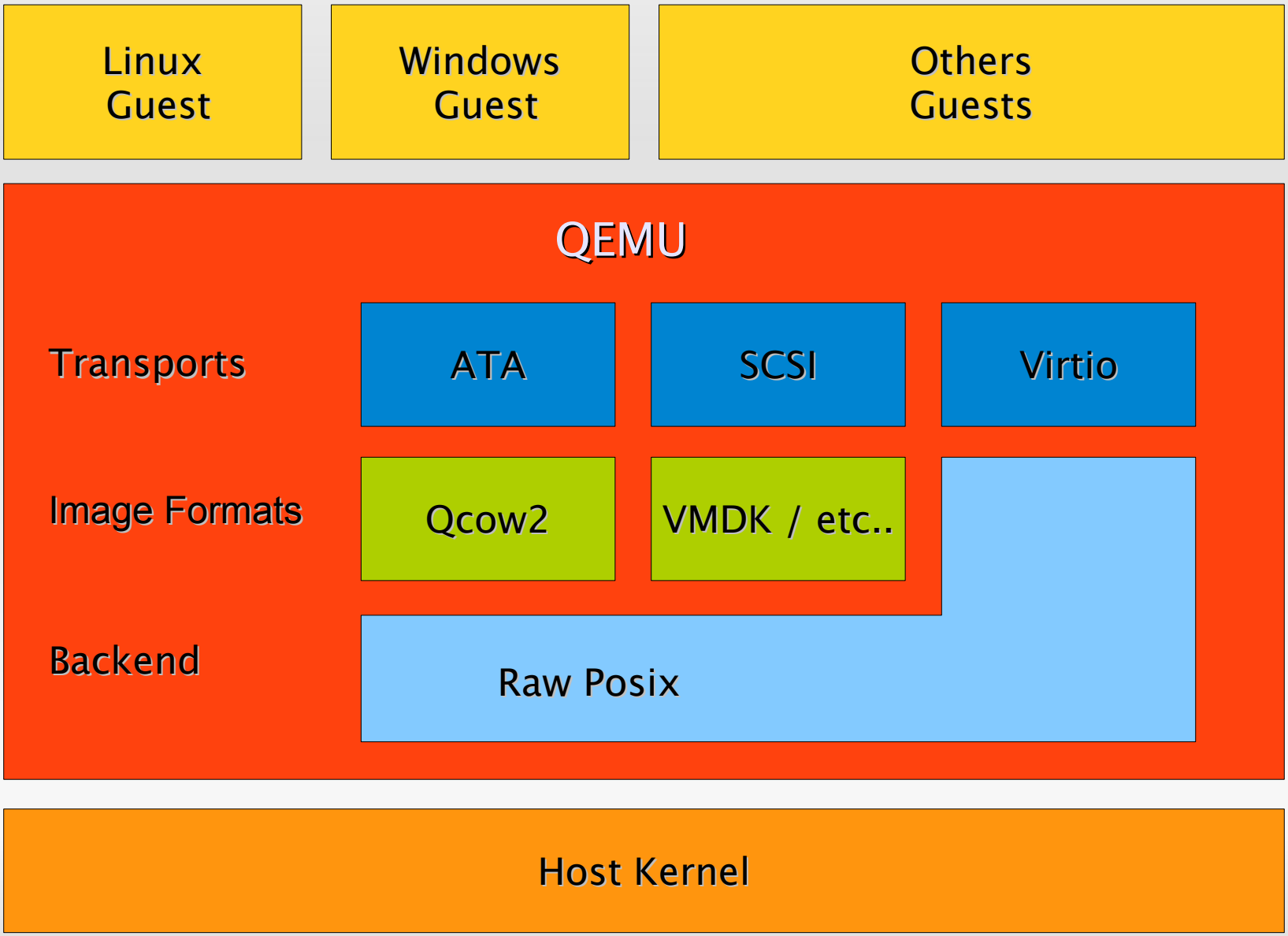
A practical implementation: QEMU/KVM

- 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 simpler block driver ontop of virtio
  - Just simple read/write requests
  - And SCSI requests through ioctls
  - And, and, and..

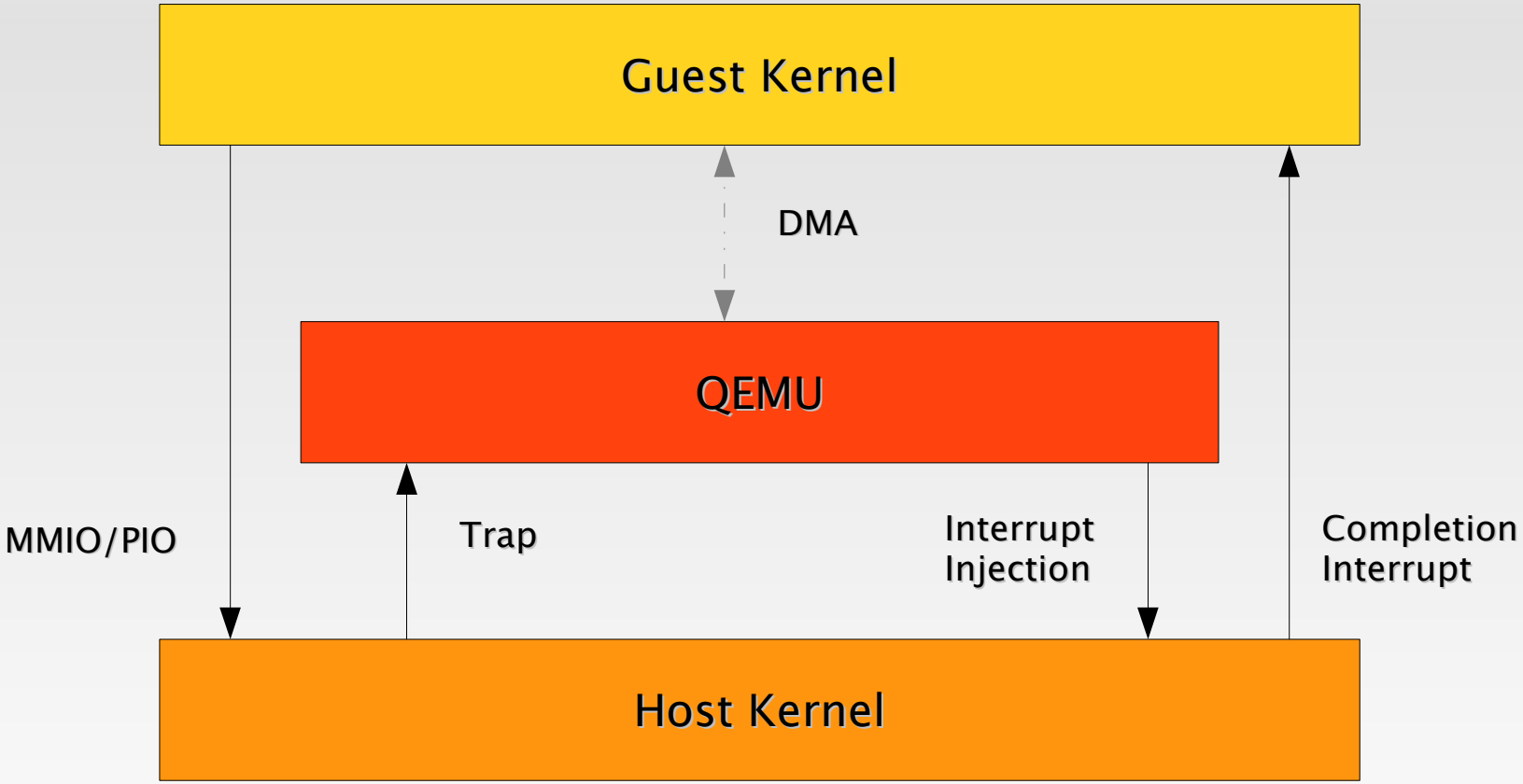
QEMU storage requirements - AIO

- 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

QEMU storage requirements – vectors

- Typical I/O requests from guest 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

Life of an I/O request



Posix 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
  - Also forwards some management commands (ioctls) in case of device files

Posix storage backend

- Except life isn't THAT simple..

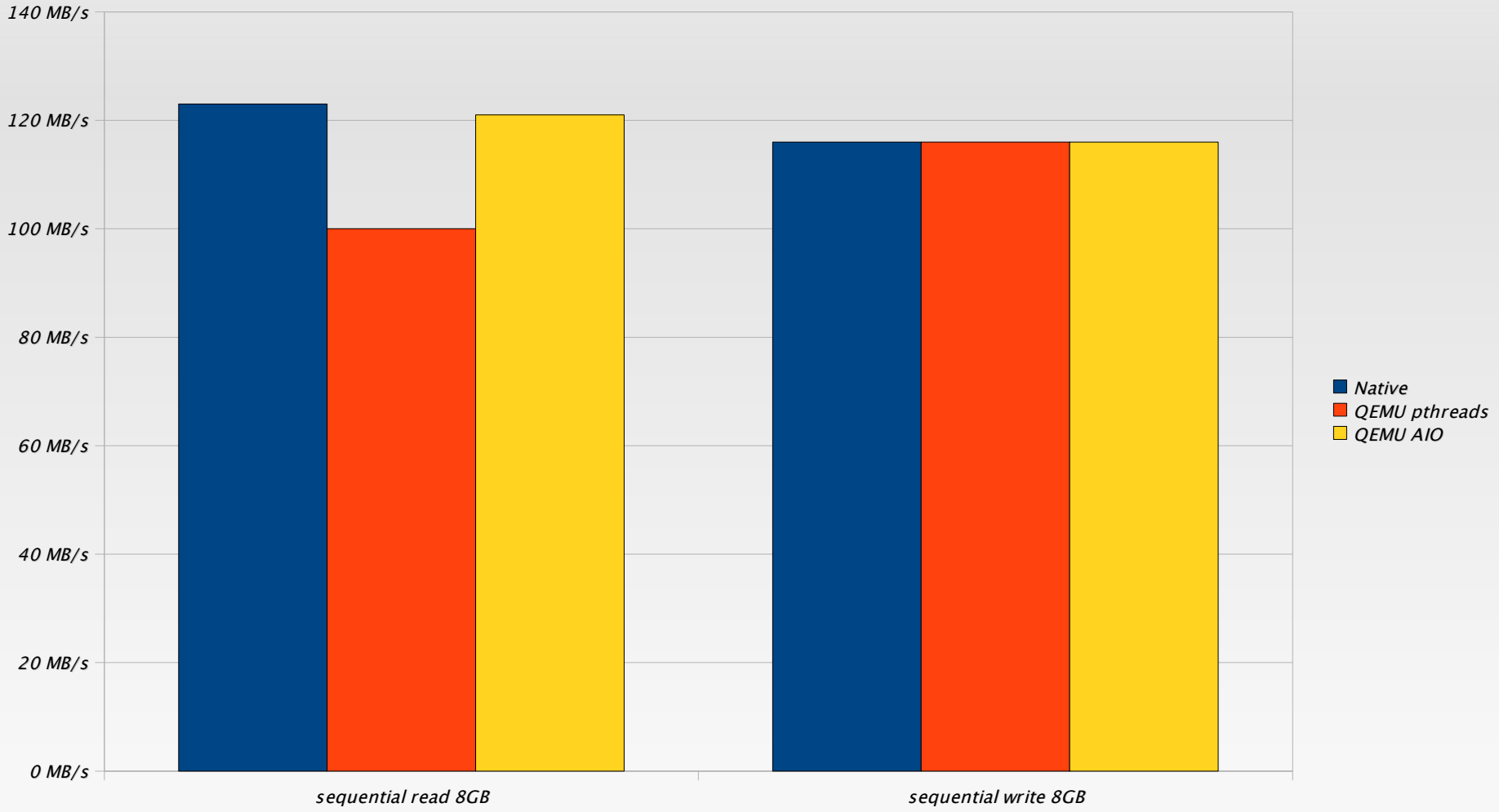
Posix storage backend – AIO

- Needs to implement asynchronous semantics
- 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

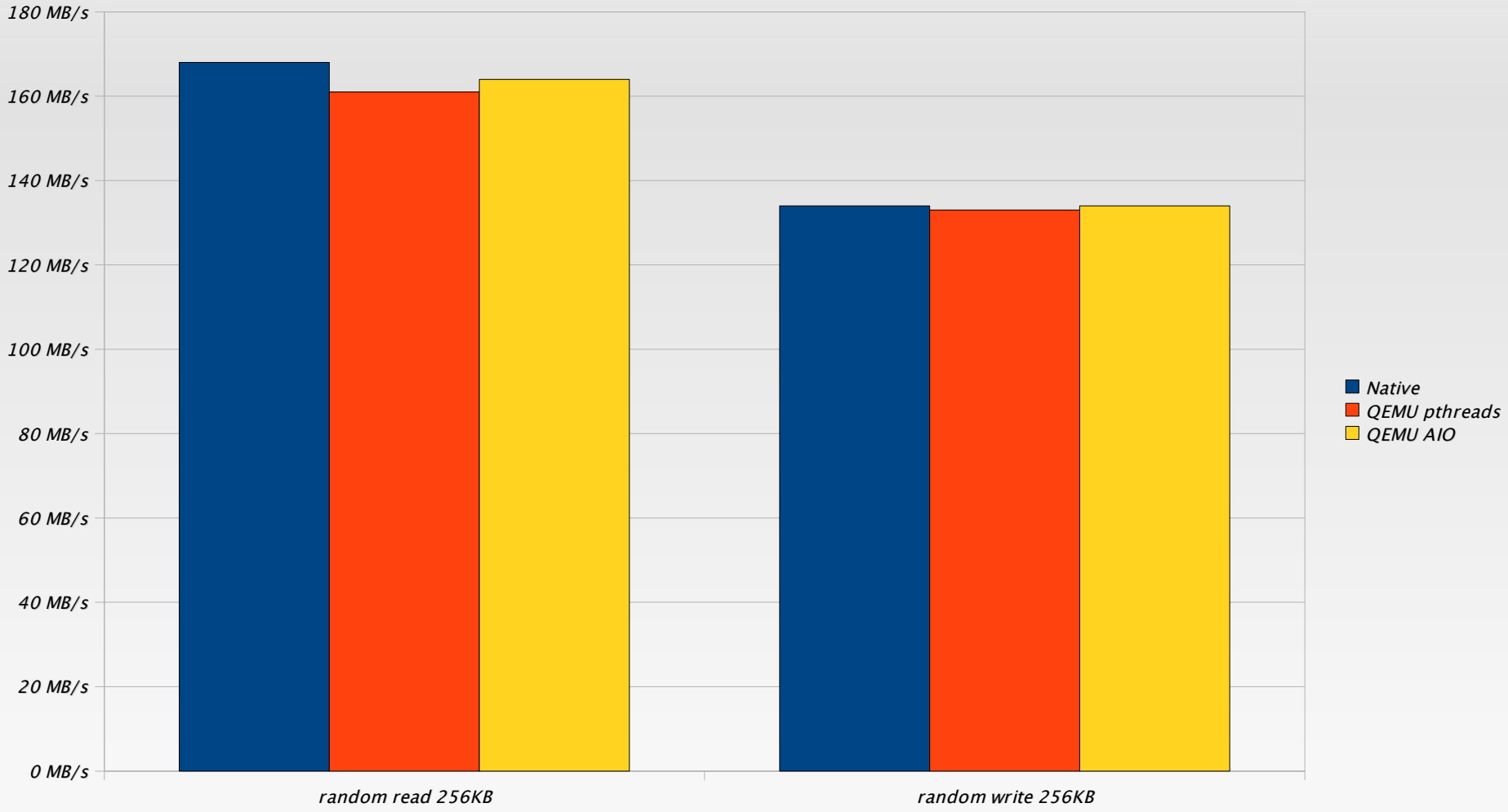
Posix storage backend – more fun

- Hosts often have I/O restrictions
  - Uncached I/O requires strict alignment and specific I/O sizes
  - Posix backed needs to perform read/modify/write cycles
- Want to pass-through commands (ioctls) to host devices
  - Different for every OS or even driver

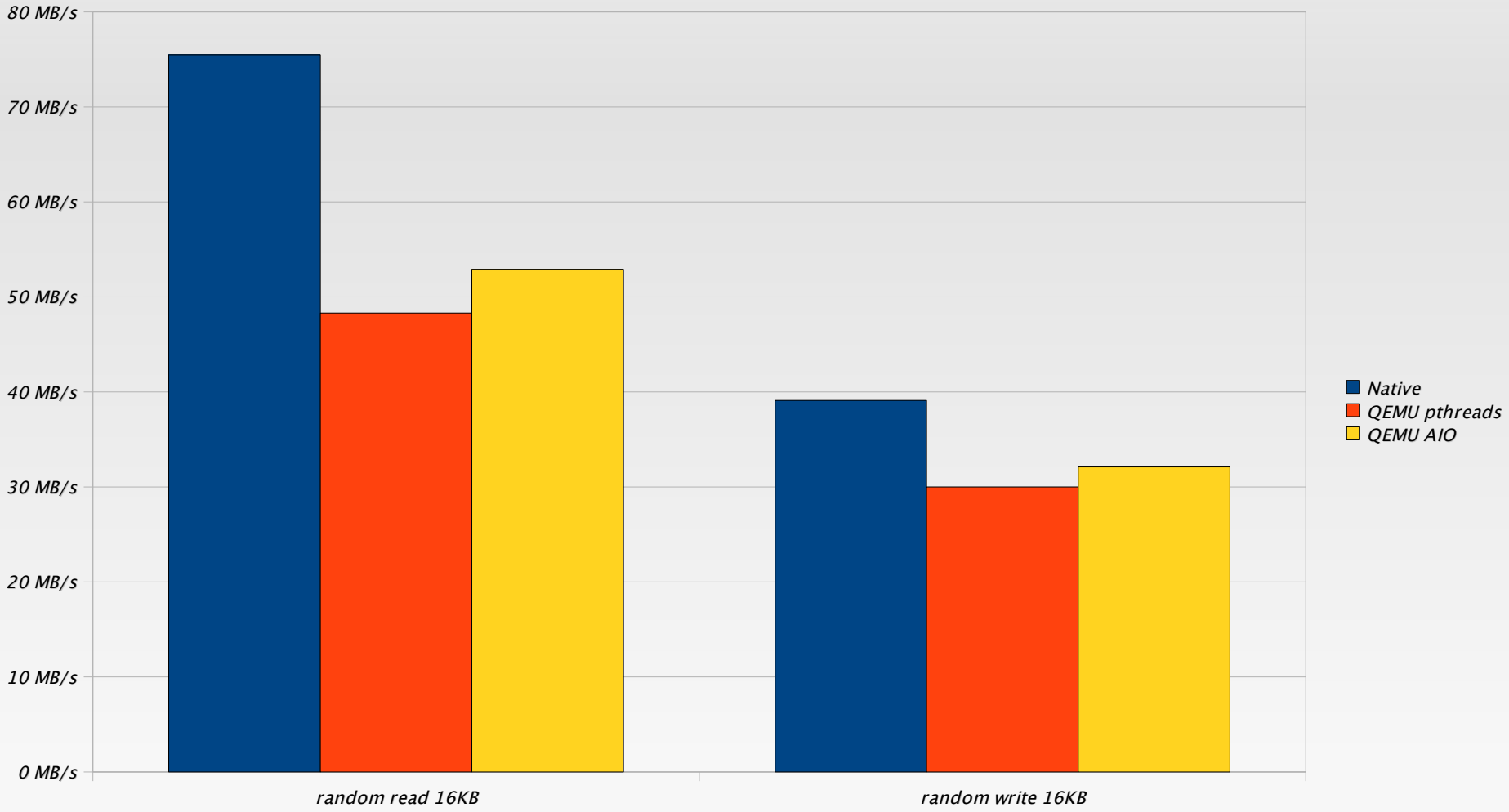
Performance – large sequential I/O



Performance – 256 kilobyte random I/O



Performance – 16 kilobyte random I/O



The quest for 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

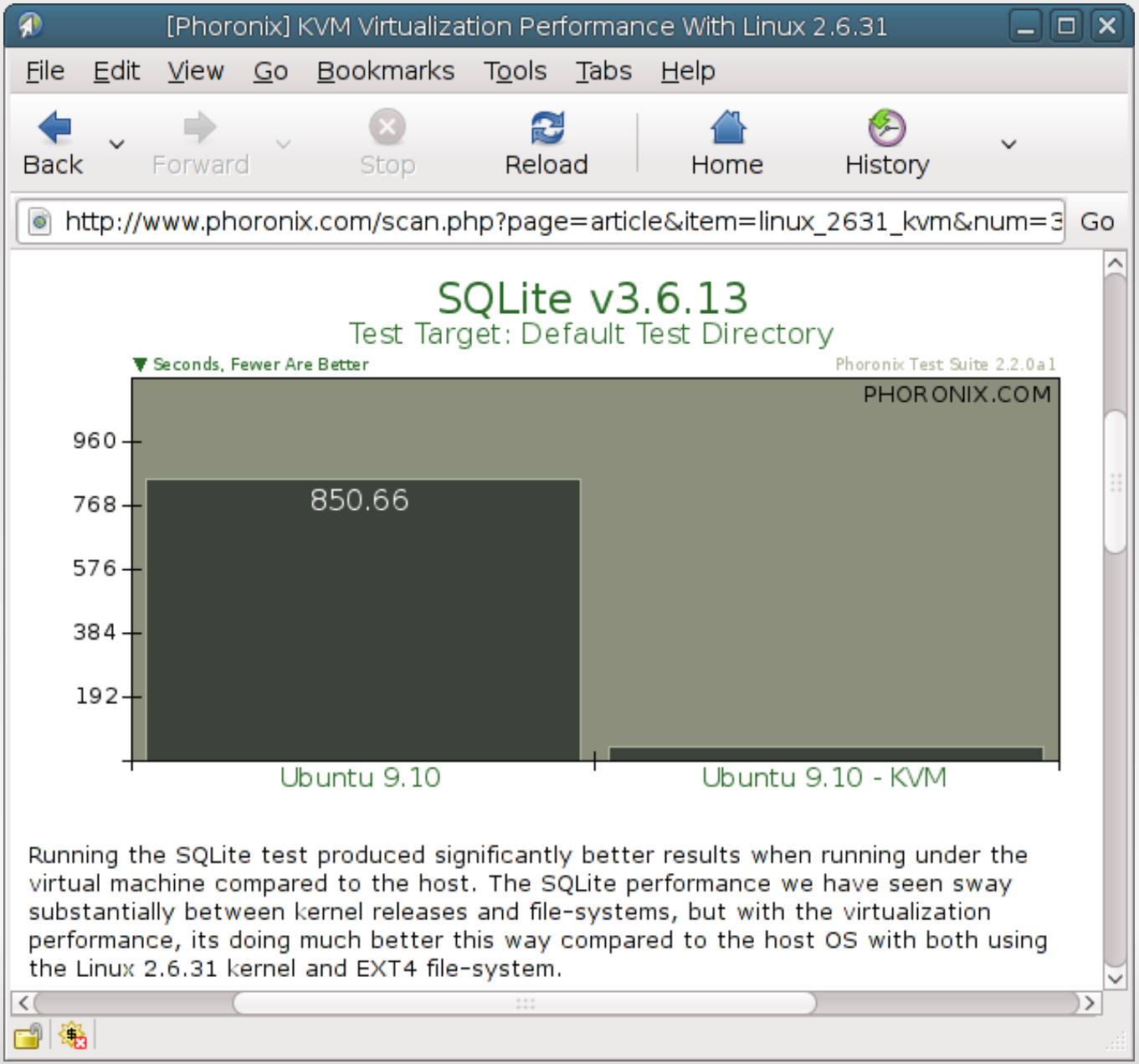
Disk Image formats

- QEMU also supports various foreign image formats:
  - Cow – User Mode Linux
  - Vpc – Microsoft Virtual PC
  - Vmdk – Vmware
  - Bochs
  - Parallels
  - Cloop – popular Linux compressed loop patch
  - Dmg – MacOS native filesystem images

Non-image backends

- 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 vvfat backend allows exporting host directories as image with a far format filesystem

Benchmarks..


[Phoronix] KVM Virtualization Performance With Linux 2.6.31
File Edit View Go Bookmarks Tools Tabs Help
Back Forward Stop Reload Home History
http://www.phoronix.com/scan.php?page=article&item=linux_2631_kvm&num=3
Go
SQLite v3.6.13
Test Target: Default Test Directory
Seconds, Fewer Are Better
Phoronix Test Suite 2.2.0a1
PHORONIX.COM
960
768
576
384
192
850.66
Ubuntu 9.10
Ubuntu 9.10 - KVM
Running the SQLite test produced significantly better results when running under the virtual machine compared to the host. The SQLite performance we have seen sway substantially between kernel releases and file-systems, but with the virtualization performance, its doing much better this way compared to the host OS with both using the Linux 2.6.31 kernel and EXT4 file-system.

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 gurantees about metadata
- Thus also needs an explicit cache flush.

Thin provisioning

- Technical term for overcommiting storage resources
  - A simple example is a sparse file that doesn't actually have blocks allocated to it before use
  - Full Thin Provisioning also means reclaiming space again when data is delete
- A big topic both for high-end storage arrays and virtualization

Thin provisioning – standards

- The T10 SPC standard for SCSI disks / 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
- Needs extensive guest support
- The ATA spec has a similar TRIM command for Solid State Drives (SSDs)

Thin provisioning – implementation

- On the guest side leverage the support for SSDs / Arrays
- On the host side the command decoding in qemu is easy
- But the standard filesystem API does not allow punching holes into files
  - Some filesystem (e.g. XFS) offer extensions for it

Thin provisioning – demo

Avoiding duplicate data

- Often many similar virtual machines are running on one host
  - Aim for stroing duplicate data only once
- 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 backing devices in the QCOW2 format.
  - Very easy to use
- LVM supports copy on writevolumes
  - Similarly easy to use
  - But requires a full block devices, not files
- Filesystems like btrfs and ocfs allows file level snapshots

Data deduplication

- All these have one common disadvantage:
  - The sharing needs to be planned from the beginning.
- Data deduplication is the process of finding these duplicates later
  - It's an expensive and slow process without additional metadata
  - Not currently implemented in a way usable by QEMU currently

Questions?

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