An Updated Overview of the QEMU Storage Stack

Stefan Hajnoczi – stefanha@linux.vnet.ibm.com
Open Virtualization
IBM Linux Technology Center

2011

The topic

- What is the QEMU storage stack?
- Configuring the storage stack
- Recent and future developments
  - “Cautionary statement regarding forward-looking statements”

QEMU and its uses

- “QEMU is a generic and open source machine emulator and virtualizer”
  - http://www.qemu.org/
- Emulation:
  - For cross-compilation, development environments
  - Android Emulator, shipping in an Android SDK near you
- Virtualization:
  - KVM and Xen use QEMU device emulation

Storage in QEMU

- Devices and media:
  - Floppy, CD-ROM, USB stick, SD card, harddisk
- Host storage:
  - Flat files (img, iso)
    - Also over NFS
  - CD-ROM host device (/dev/cdrom)
  - Block devices (/dev/sda3, LVM volumes, iSCSI LUNs)
  - Distributed storage (Sheepdog, Ceph)

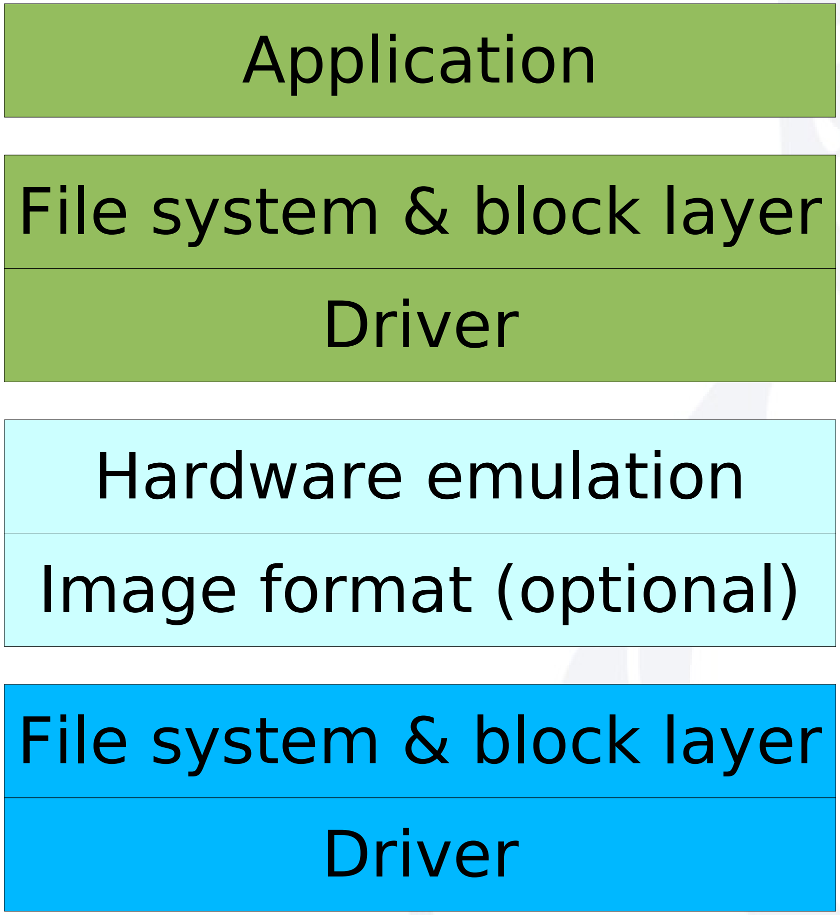
QEMU -drive option

```
qemu -drive
        if=ide|virtio|scsi,
        file=path/to/img,
        cache=writethrough|writeback|none|unsafe
```

- Storage interface is set with **if=**
- Path to image file or device is set with **path=**
- Caching mode is set with **cache=**

- More on what this means later, but first the picture of the overall storage stack...

The QEMU storage stack



Application

File system & block layer

Driver

Hardware emulation

Image format (optional)

File system & block layer

Driver

- Application and **guest** kernel work similar to bare metal.
- Guest talks to QEMU via emulated hardware.

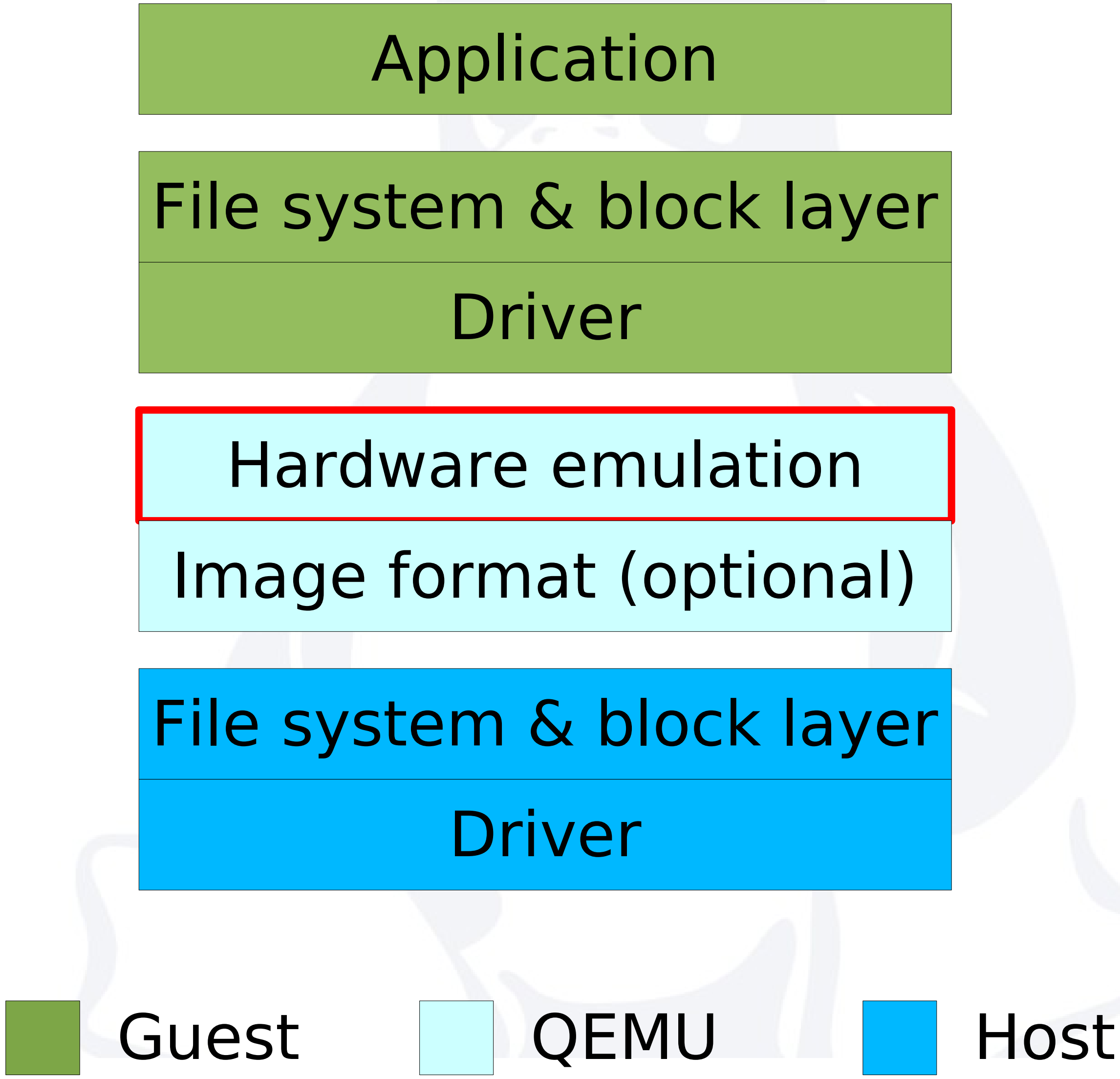
- **QEMU** performs I/O to an image file on behalf of the guest.
- **Host** kernel treats guest I/O like any userspace application.

🟩 Guest ⬜ QEMU 🟦 Host

Seeing double

- There may be two **file systems**. The guest file system and the host file system (which holds the image file).
- There may be two **volume managers**. The guest and host can both use LVM and md independently.
- There are two **page caches**. Both guest and host can buffer pages from a file.
- There are two **I/O schedulers**. The guest will reorder or delay I/O but the host will too.
- Configuring either the guest or the host to bypass these layers typically leads to best performance.

Emulated storage overview



Application

File system & block layer

Driver

Hardware emulation

Image format (optional)

File system & block layer

Driver

Guest

QEMU

Host

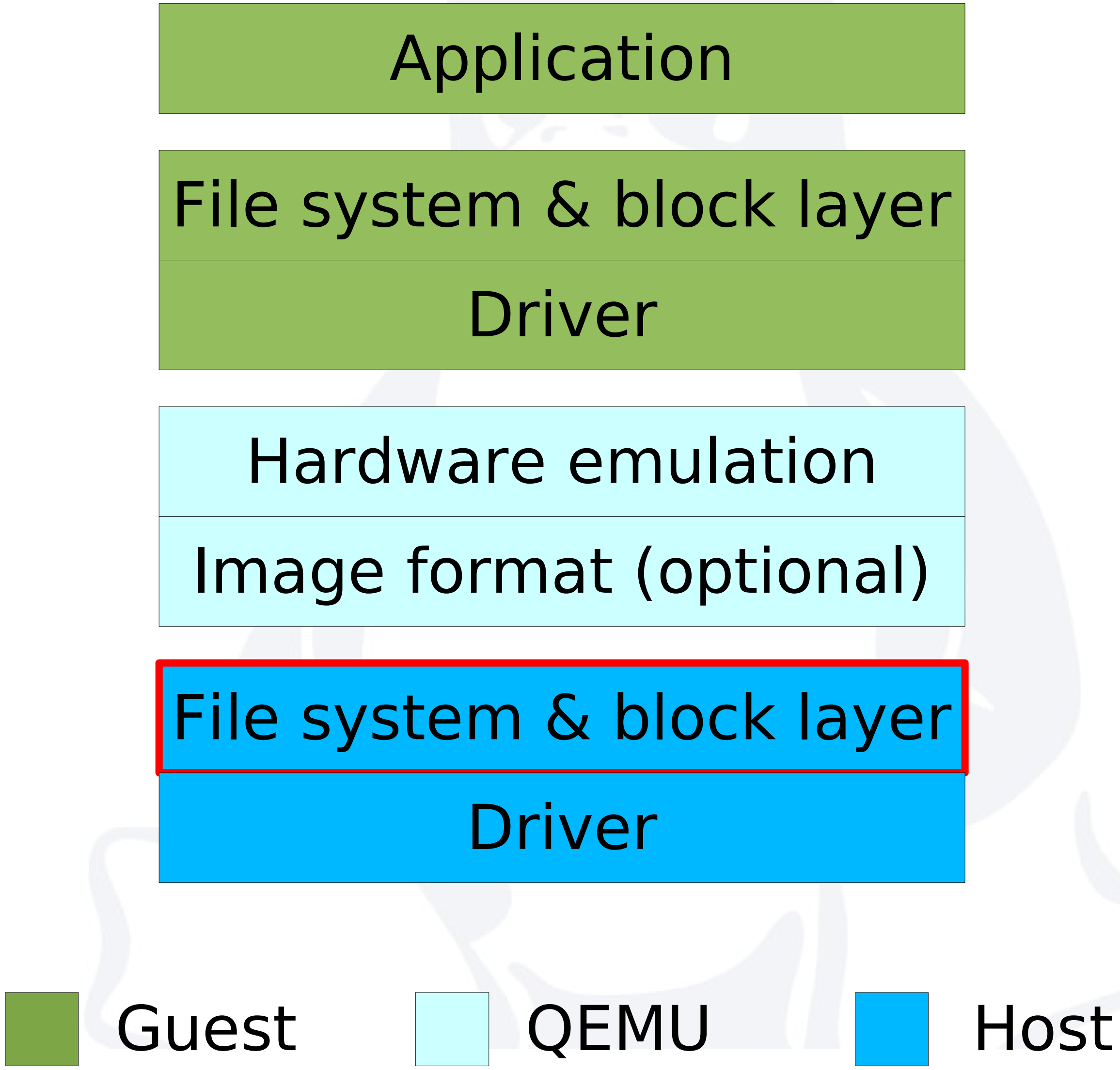
Emulated storage

- QEMU presents emulated storage interfaces to the guest
- **Virtio** is a paravirtualized storage interface, delivers the best performance, and is extensible for the future
    - One virtio-blk PCI adapter per block device
- **IDE** emulation is used for CD-ROMs and is also available for disks
    - Good guest compatibility but low performance
- **SCSI** emulation can be used for special applications but is still under development

Emulated storage in the future

- SATA (**AHCI**) emulation
  - Currently experimental
  - Promises better performance than IDE
  - Relatively wide compatibility
- Renewed focus on **SCSI**
  - Patches to make SCSI emulation robust continue to come in, though slowly
  - Virtio-scsi is being prototyped
  - Industry standard, rich features

Host page cache overview



Application

File system & block layer

Driver

Hardware emulation

Image format (optional)

File system & block layer

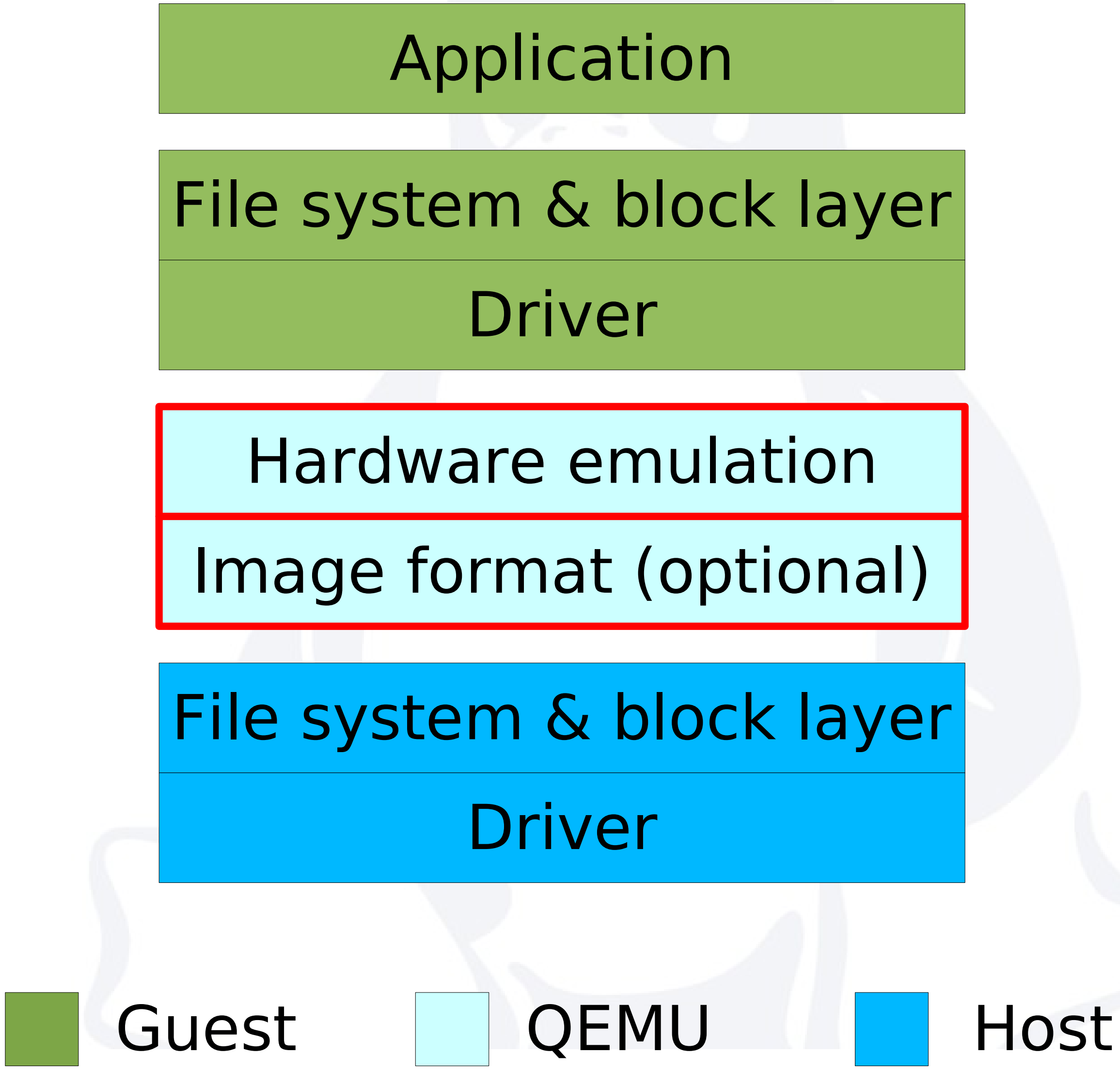
Driver

🟩 Guest 🟦 QEMU 🟦 Host

Host page cache

- Writes complete after copying data to page cache
- Cache is flushed on fsync(2)
- Reads may be satisfied from the cache
- Guest has its own page cache
  - Two copies of data in memory
- Disabling host page cache:
  - O_DIRECT I/O on the host
  - Bypasses host page cache when possible
  - Zero-copy when possible

Guest disk write cache overview



Application

File system & block layer

Driver

Hardware emulation

Image format (optional)

File system & block layer

Driver

🟩 Guest

⬜ QEMU

🟦 Host

Guest disk write cache

- Disk completes writes after they reach cache
  - Data may not be on disk
- Volatile disk write cache loses contents on power failure
  - Correct applications fsync(2) to guarantee data is on disk
- When write cache is disabled:
  - Writes complete when they are on disk
  - Write performance is reduced
- Enabling write cache:
  - Improves write performance
  - Only ensures data integrity if applications and storage stack flush cache correctly

Caching modes in QEMU

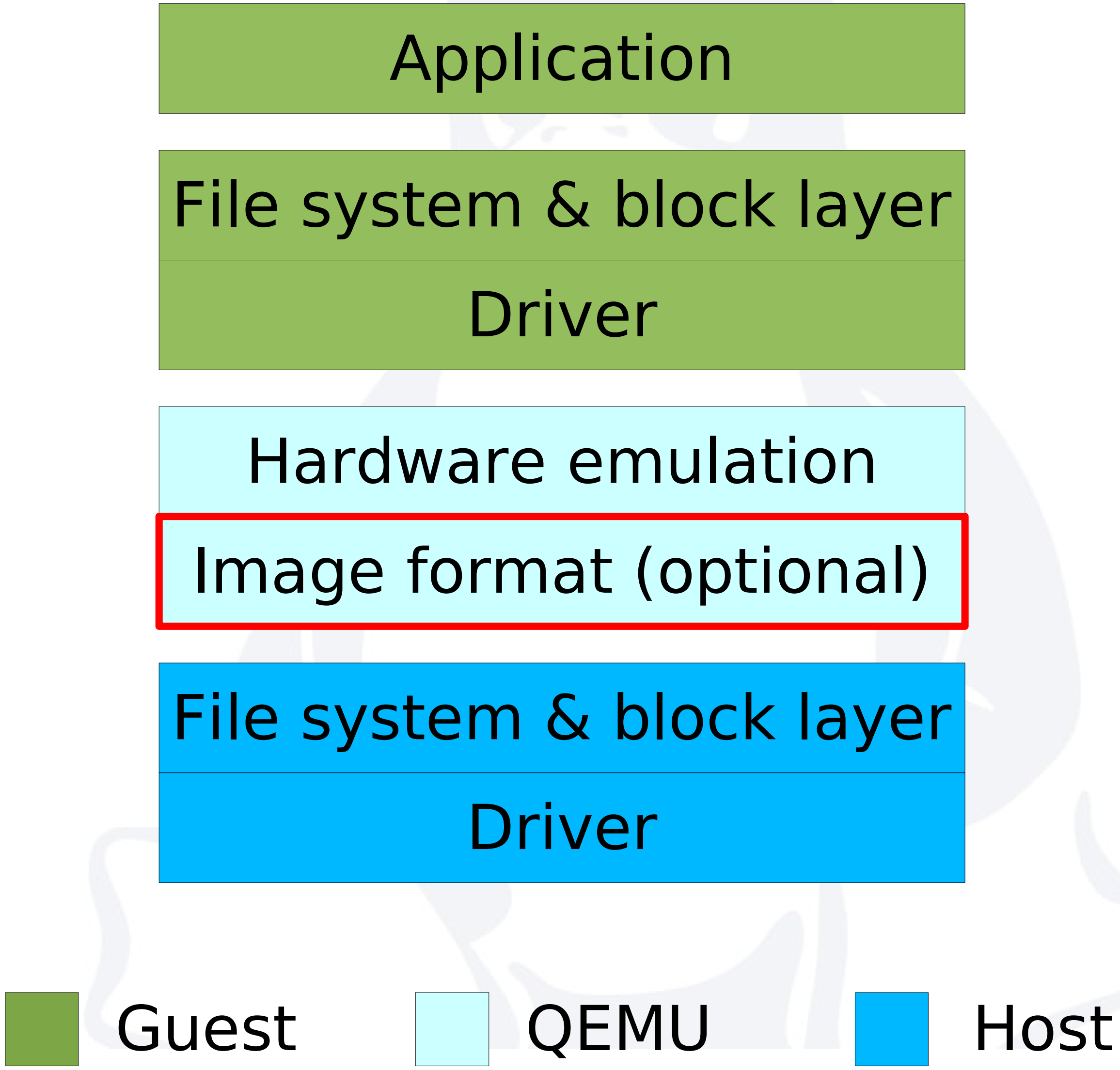
Mode	Host page cache	Guest disk write cache
none	off	on
writethrough	on	off
writeback	on	on
unsafe	on	ignored

- Default is writethrough
- Unsafe is a new mode that ignores cache flush operations
  - Only use for temporary data
  - Useful for speeding up guest installs
  - **Switch to another mode for production**

Caching modes in the future

- Guest control over disk write cache (WCE)
  - Real disks allow WCE toggling at runtime
  - Lets guest determine whether to enable
    - Useful for hosting or cloud environments
- Ability to change host page cache option at runtime
  - Today QEMU requires restart to change host page cache

Image formats overview



Application

File system & block layer

Driver

Hardware emulation

Image format (optional)

File system & block layer

Driver

🟩 Guest

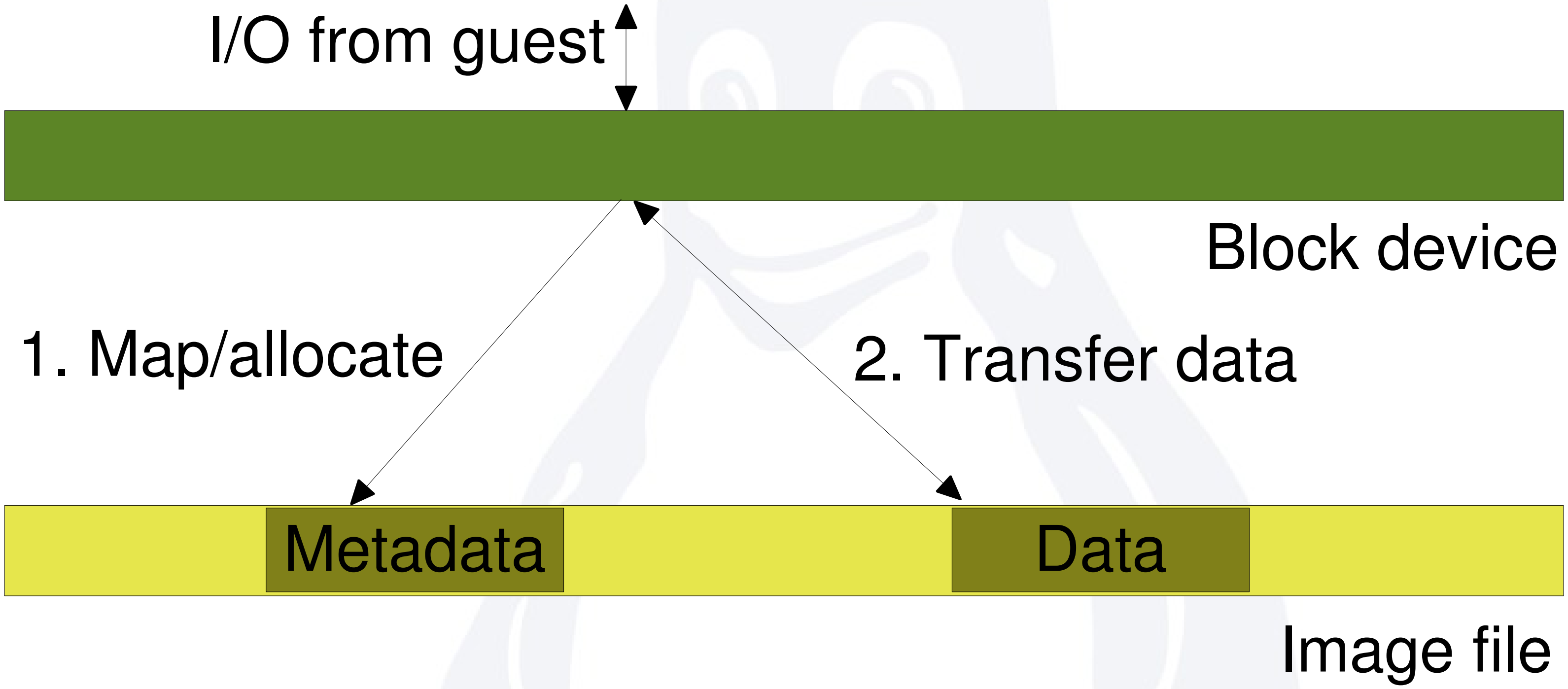
⬜ QEMU

🟦 Host

Image formats

- Supported image formats:
  - QCOW2, QED – QEMU
  - VMDK – VMware
  - VHD – Microsoft
  - VDI – VirtualBox
- Features that various image formats provide:
  - Sparse images
  - Backing files (delta images)
  - Encryption
  - Compression
  - Snapshots

How image formats work



- Map **logical block addresses** to **file offsets**
- Apply transformations on data (compression, encryption)

Manipulating image files

- Only raw image files can be loopback mounted
  - Use **qemu-nbd** to access image files on host
    - http://tinyurl.com/qemu-nbd
  - Or use the powerful **libguestfs**:
    - Http://libguestfs.org/
- Convert image formats with **qemu-img**
  - Qemu-img is the Rosetta Stone of image formats
  - Supports all image formats that QEMU does
  - Stand-alone program, can be used without installing QEMU

Image formats in the future

- Improving VMDK compatibility
  - Adding support for latest file format versions
  - Google Summer of Code 2011 project
- QCOW2<->QED in-place conversion
  - Convert formats without copying data
  - Google Summer of Code 2011 project
- QED image streaming
  - Start new guest immediately, populate data from backing file as it runs
- QCOW2v3
  - Currently in design phase
  - Enhance format with new ideas and address pain points

Recommendations

- Emulated storage interface:
  - **Virtio** for Linux and Windows guests
  - **IDE** when virtio is not possible
- Caching mode:
  - **cache=none** for local storage
- Host storage:
  - **LVM** if flexibility of image files not needed
  - **Raw** image files if features not needed
  - **QCOW2 or QED** if more features are required
  - **Vmdk and others** convert to native format

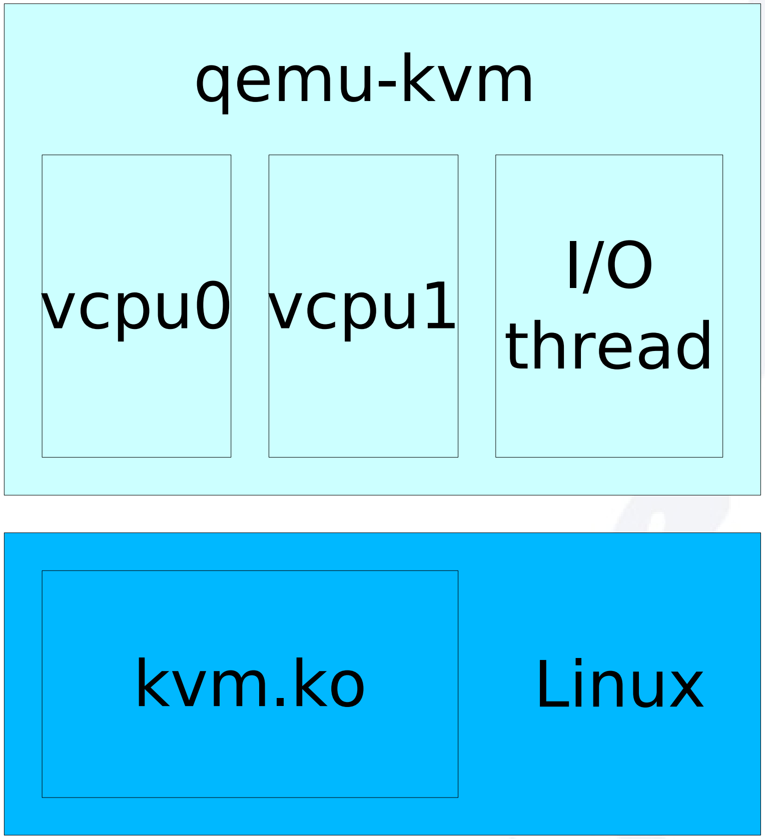
Summary

- There are many layers to the storage stack
  - Some layers are optional
  - Choose what you need
- Defaults: IDE storage interface and writethrough cache mode
  - Conservative and compatible
  - Consider virtio-blk and none cache mode
- Image formats can be tamed with qemu-img, qemu-nbd, and libguestfs

Questions?

Blog: http://blog.vmsplice.net/

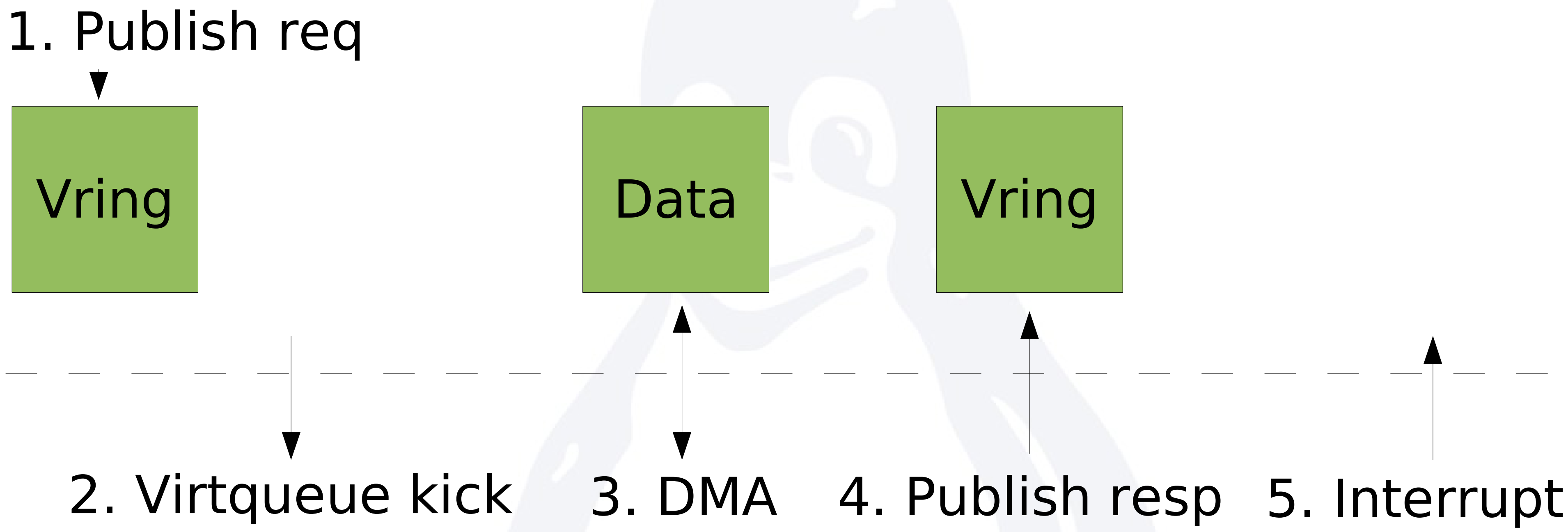
QEMU Architecture



- Each guest CPU has a dedicated **vcpu thread** that uses the kvm.ko module to execute guest code.
- There is an **I/O thread** that runs a select(2) loop to handle events.

- Only one thread may be executing QEMU code at any given time. This excludes guest code and blocking in select(2).

Virtio-blk request lifecycle



1. Publish req

Vring

Data

Vring

2. Virtqueue kick

3. DMA

4. Publish resp

5. Interrupt

- Request/response data and metadata live in guest memory.
- Virtqueue kick is a pio write to a virtio PCI hardware register.
- Completion is signaled by virtio PCI interrupt.