QEMU Block Layer Concepts & Features

Stefan Hajnoczi
stefanha@redhat.com

Red Hat

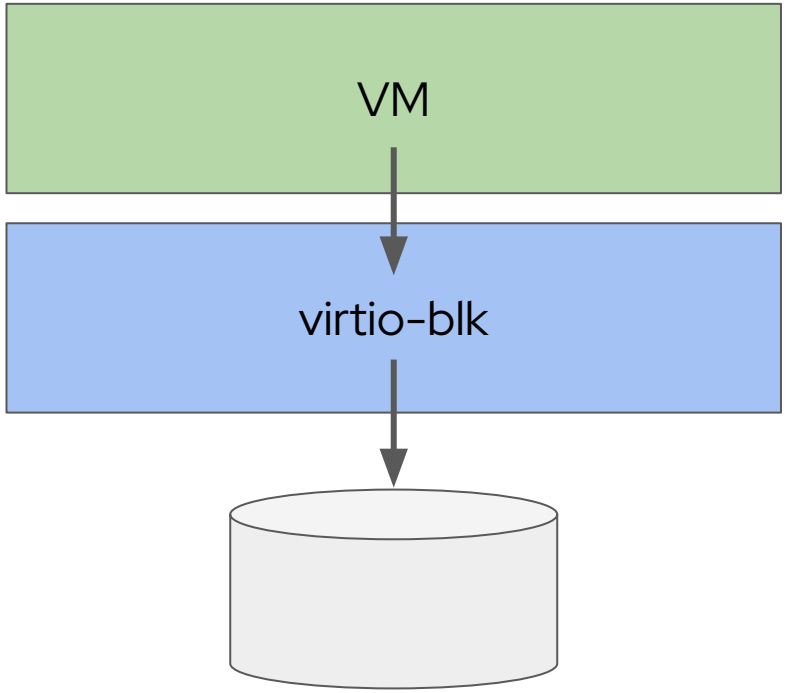
Block Devices in Virtual Machines

VMs are configured with emulated storage controllers

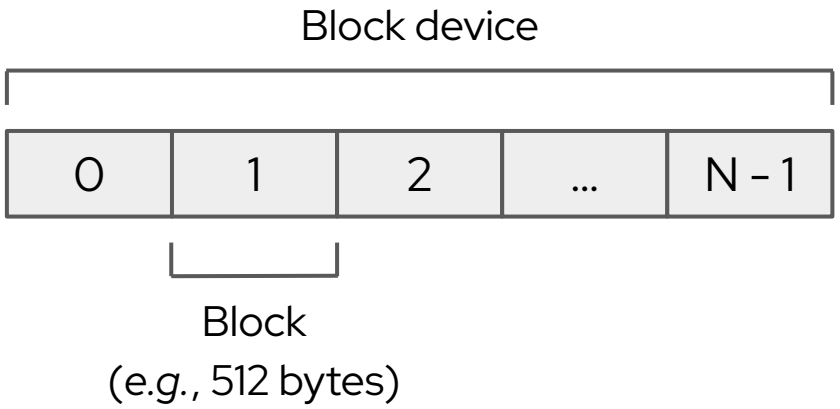
- virtio-blk, virtio-scsi, AHCI/SATA, etc

Storage controllers provide access to **block devices** for data storage.

Root file system and data volumes are usually block devices.



Block Devices



NVMe, SCSI, ATA, virtio-blk follow the block device model

Read & **Write** access data in units of blocks

Flush persists previously written data to permanent storage

Discard (TRIM) and **Write Zeroes** manage block allocation

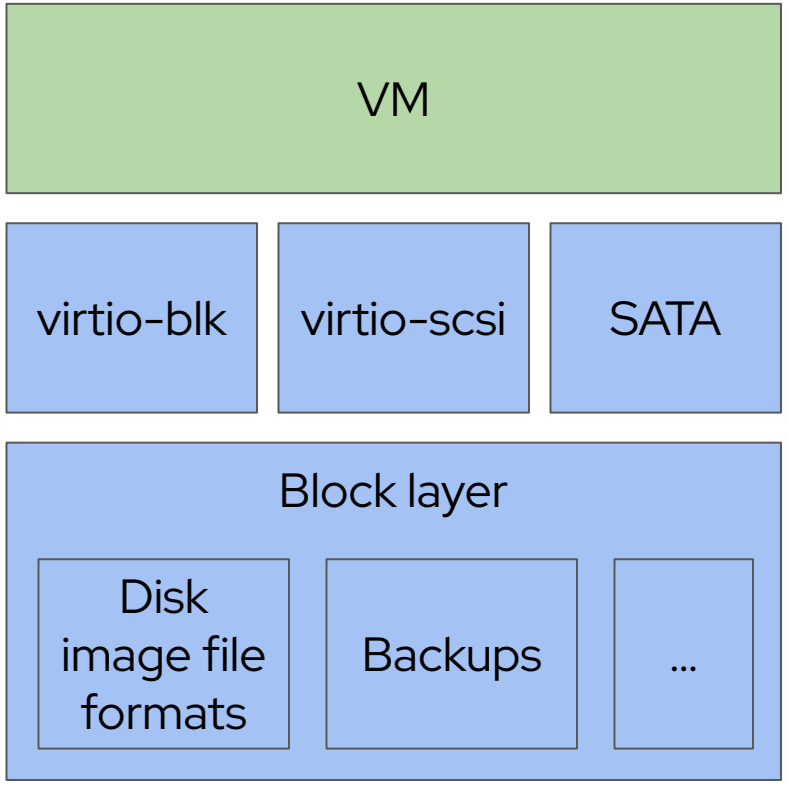
Purpose of the Block Layer

Reuse functionality in all emulated storage controllers

Avoids hardcoding the same code into each storage controller

The block layer is software-defined storage:

- Build block devices by composing primitives



Red Hat

Disk Images

Host Block Devices

Expose host block device `/dev/nvme0n1` to a guest

```
[host]$ lsblk
NAME    MAJ:MIN RM   SIZE RO TYPE
nvme0n1  259:0    0 238.5G  0 disk
```

```
[guest]$ lsblk
NAME MAJ:MIN RM   SIZE   RO TYPE
vda  254:0   0    238.5G  0 disk
```

Life is easy:

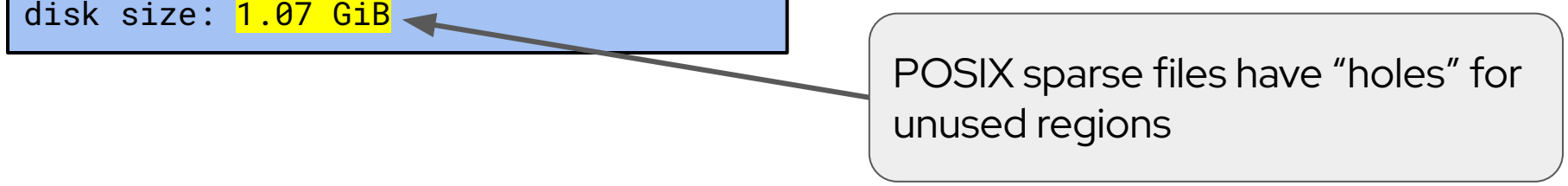
- I/O requests are passed through unmodified
- Virtual disk size matches host block device capacity

Raw Files

Expose a regular file from the host to the guest

```
[host]$ qemu-img info test.img
image: test.img
file format: raw
virtual size: 10 GiB (10737418240
bytes)
disk size: 1.07 GiB
```

```
[guest]$ lsblk
NAME MAJ:MIN RM   SIZE   RO TYPE
vda  254:0   0    10G  0 disk
```



Same format as dd(1) and disk imaging tools

Easy to copy, resize, download, backup, etc

Other Protocols

QEMU also supports non-file storage:

- rbd - Ceph RADOS block devices through librbd
- nbd - Network Block Device protocol
- iscsi - iSCSI targets through libiscsi
- https - libcurl HTTP client
- And more

Image Formats

Features that can be added by image formats:

- Compactness - avoid downloading zero-filled regions
- Compression - save disk space
- Encryption - confidentiality/integrity
- Snapshots - point-in-time save/restore
- Saving device/RAM state - save running VM state
- Dirty bitmaps - incremental backup metadata

Qcow2 disk image format

QEMU's native disk image file format

Popular format for distributing disk images

```
[host]$ qemu-img info test.qcow2
image: test.qcow2
file format: qcow2
virtual size: 10 GiB (10737418240 bytes)
disk size: 1.07 GiB
cluster_size: 65536
Format specific information:
    compat: 1.1
    compression type: zlib
    …
[host]$ ls -lh test.qcow2
-rw-r--r--. 1 user user 1.1G Oct 24 07:21 test.qcow2
```

- ✓ Compactness
- ✓ Compression
- ✓ Snapshots
- ✓ Saving device/RAM state
- ✓ Dirty bitmaps
- ✓ Encryption

Other Disk Image Formats

QEMU supports other popular formats for import/export:

- VMware VMDK
- Microsoft VHDX
- VirtualBox VDI
- Parallels
- And more

Only use third-party formats for import/export

Use raw or qcow2 for running VMs with best performance

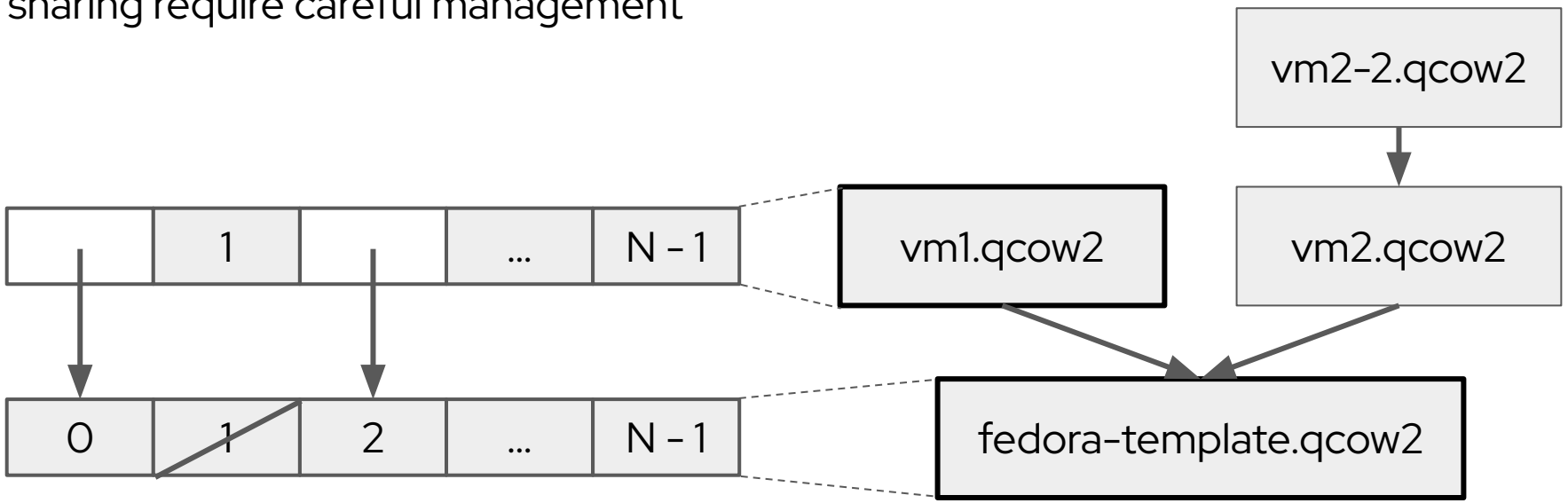
External Snapshots

If a block is not allocated, look in the backing file

Backing files can be chained together

Multiple files can share the same backing file

Chains & sharing require careful management



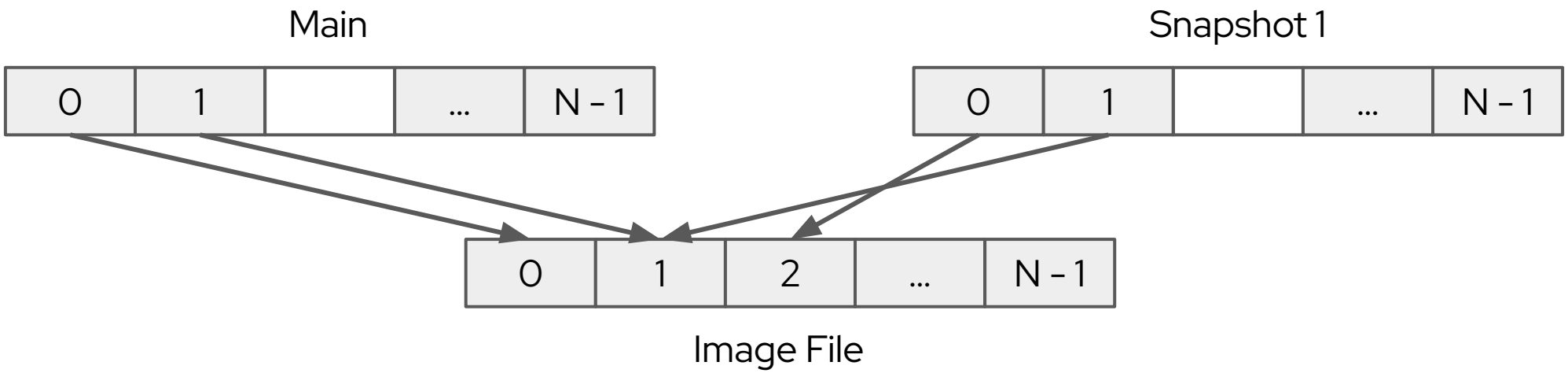
Red Hat

Internal Snapshots

Snapshots reside within a single disk image file

Blocks in the image file can be referenced by multiple snapshots

No management of chains required but harder to share between VMs



Red Hat

LUKS Encryption

LUKS v1 format is supported

Compatible with Linux dm-crypt/cryptsetup

Integrated into qcow2 or available separately

Blockdev Graph

Blockdevs

Storage functionality is configured through the –blockdev option:

```
[host]$ qemu-system-x86_64 \
        -blockdev driver=file,node-name=file0,filename=vm.img \
        -device virtio-blk-pci,drive=file0
```

Lots of drivers are available: `$ man qemu-block-drivers`

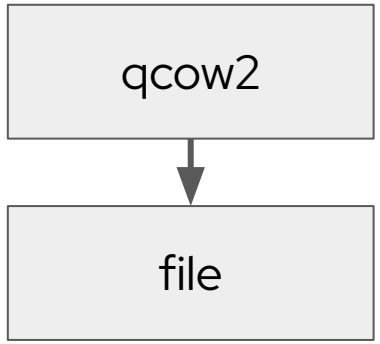
Each blockdev has a “node-name” so it can be referenced

The Graph

Blockdev graphs can be built to compose storage functionality

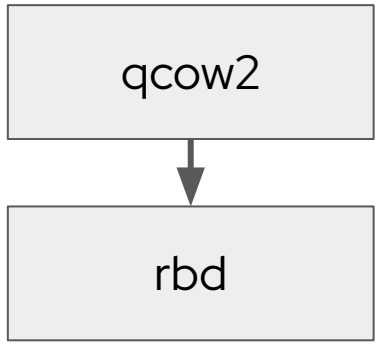
It's common to have a regular file with an image format on top:

```
[host]$ qemu-system-x86_64 \
        -blockdev driver=file,node-name=file0,filename=vm.qcow2 \
        -blockdev driver=qcow2,node-name=qcow2-0,file=file0 \
        -device virtio-blk-pci,drive=qcow2-0
```



But the image format could be on top of something that's not a file:

```
[host]$ qemu-system-x86_64 \
        -blockdev driver=rbd,node-name=rbd0,pool=my-pool,image=vm01 \
        -blockdev driver=qcow2,node-name=qcow2-0,file=rbd0 \
        -device virtio-blk-pci,drive=qcow2-0
```



Red Hat

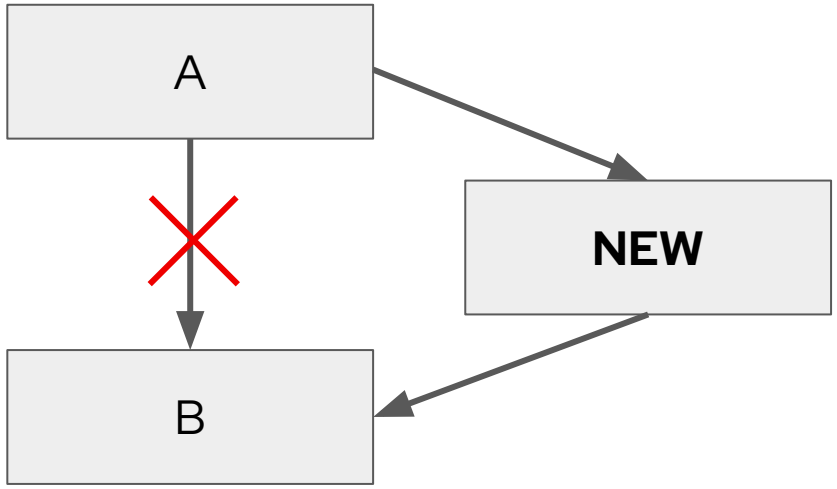
Changing the Graph at Run-time

The command-line is processed once during startup

Further changes can be made at run-time

Blockdevs can be added/removed

More on monitor commands later

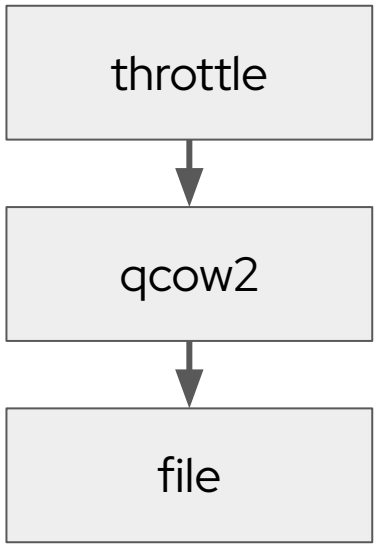


Filters

Filters are drivers that manipulate I/O requests

Example: I/O throttling

```
[host]$ qemu-system-x86_64 \
        -blockdev driver=file,node-name=file0,filename=vm.qcow2 \
        -blockdev driver=qcow2,node-name=qcow2-0,file=file0 \
        -blockdev driver=throttle,node-name=t0,throttle-group=tg0,file=qcow2-0 \
        -device virtio-blk-pci,drive=t0
```



I/O Throttling Filter

Linux cgroups blkio controller only supports block devices

QEMU's throttle filter supports all blockdevs

```
[host]$ qemu-system-x86_64 \
        -object throttle-group,id=tg0,x-iops-total=2000 \
        -blockdev driver=throttle,node-name=t0,throttle-group=tg0,file=qcow2-0 \
        …
```

Control over IOPS and throughput (read/write/total) with bursting

Can throttle a group of blockdevs for VMs with many disks

Documentation: https://gitlab.com/qemu-project/qemu/-/blob/master/docs/throttle.txt

Copy-on-Read Filter

On read requests:

- If the block is not yet allocated, populate it with data read from the backing file

Makes repeated accesses local and reduces accesses to backing file

Monitor Commands

Monitor Commands

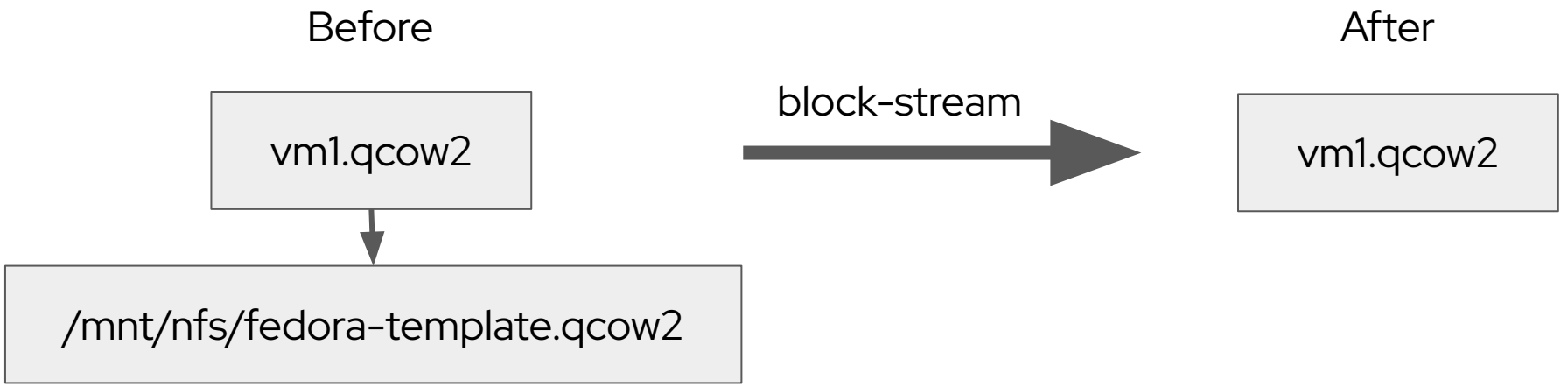
Commands are available to reconfigure running VMs:

- blockdev-snapshot-sync - create an external snapshot
- block-commit - merge backing files
- block-stream - populate from backing file
- blockdev-mirror - long-running storage cloning/migration
- blockdev-backup - point-in-time copy

Transparent to the running VM

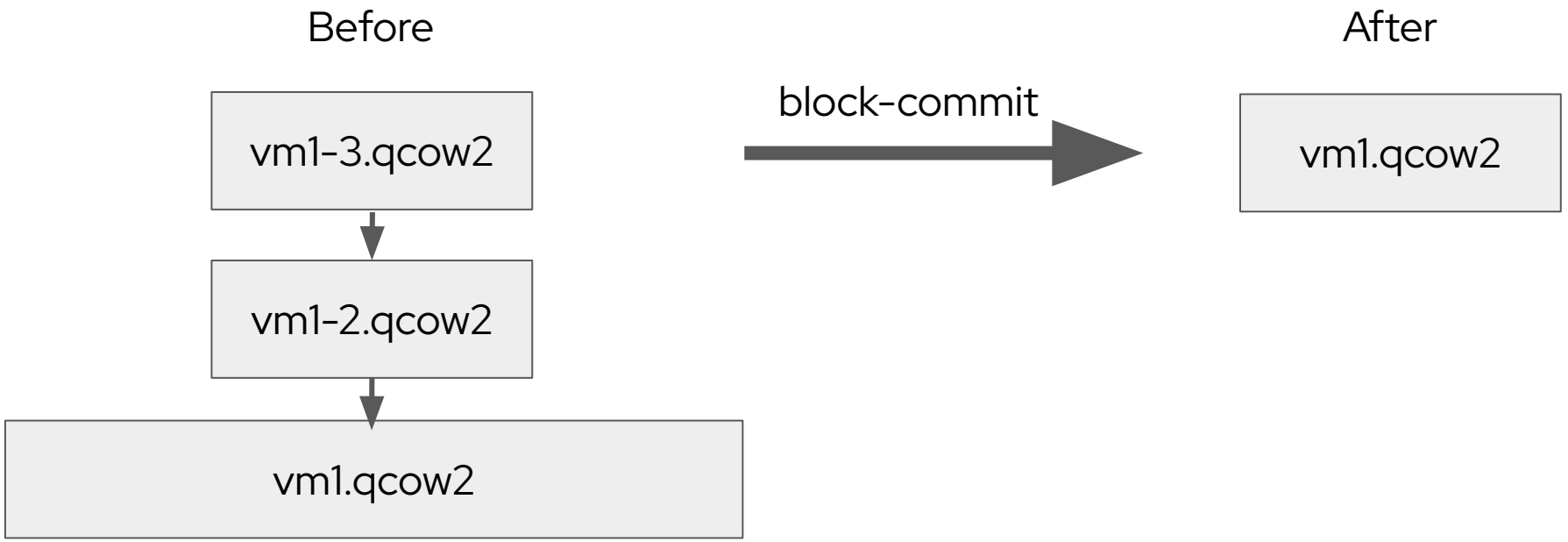
Documentation: https://www.qemu.org/docs/master/interop/live-block-operations.html

Populating Images with block-stream



1. Quickly create VM from template on NFS
2. Populate local disk image so backing file is no longer needed

Merging Backing Files with block-commit

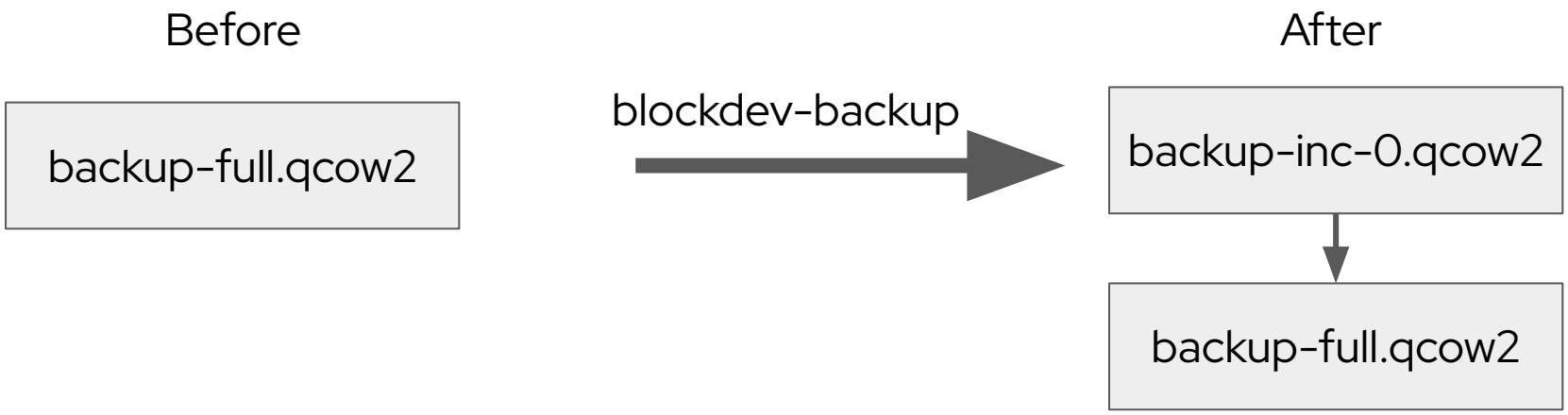


1. Create external snapshots
2. When snapshots are no longer needed, flatten the backing file chain by copying data down

Storage Migration with blockdev-mirror

Copy VM disks to a new location without interrupting VM

Incremental Backup with blockdev-backup



Given an existing full backup and dirty bitmap tracking,
blockdev-backup can copy out changed blocks into a new qcow2 file.

The existing full backup serves as the backing file.

Documentation: https://www.qemu.org/docs/master/interop/bitmaps.html

qemu-storage-daemon

Exporting Blockdevs to External Clients

Blockdevs can be **exported** to the outside world

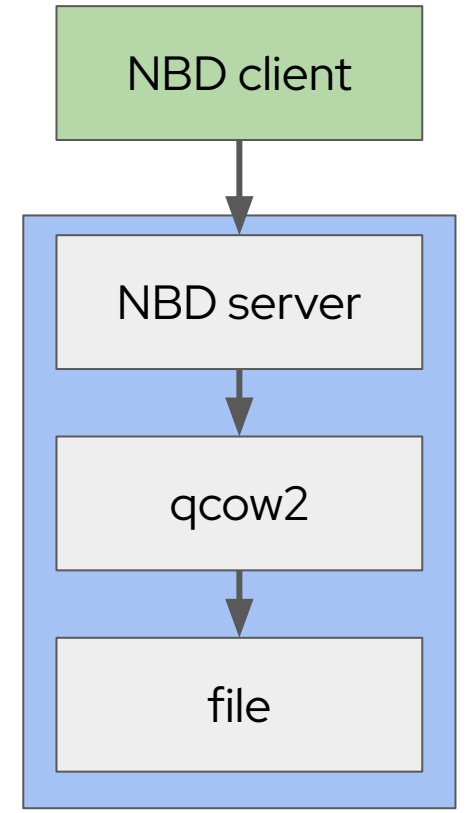
Stand-alone daemon process

Does not require running a VM

Makes QEMU's block layer available to other programs

Kevin Wolf & Stefano Garzarella's presentation:
https://www.youtube.com/watch?v=rAL_tH-gr8k

```
$ man qemu-storage-daemon
```



qemu-storage-daemon

Export Types

- NBD - Network Block Device for libnbd or kernel NBD clients over TCP
- FUSE - appears as raw file, works with many programs
- vhost-user-blk - for high-performance local applications over AF_UNIX like QEMU & libblkio applications
- VDUSE - attaches as host block device or vhost kernel device
- More expected in the future

qemu-storage-daemon Use Cases

- Separating storage from running VMs (e.g. Kubernetes CSI & KubeVirt)
- Manipulate and access disk images while VM is offline
- Get access to QEMU's many supported disk image file formats
- Safely manipulate backing files used by multiple VMs
- Centralize disk I/O emulation into one process and enable aggressive polling
- Apply tighter security policies on separate QEMU and qemu-storage-daemon processes
- Providing remote access to disk images over NBD

Userspace NVMe driver

Userspace NVMe driver

Fast local NVMe drives have <10 microsecond latency

Software overhead, including kernel and hypervisor, is significant

Userspace VFIO PCI driver for NVMe devices in QEMU bypasses layers

Supports polling (busy waiting) to avoid interrupt latency

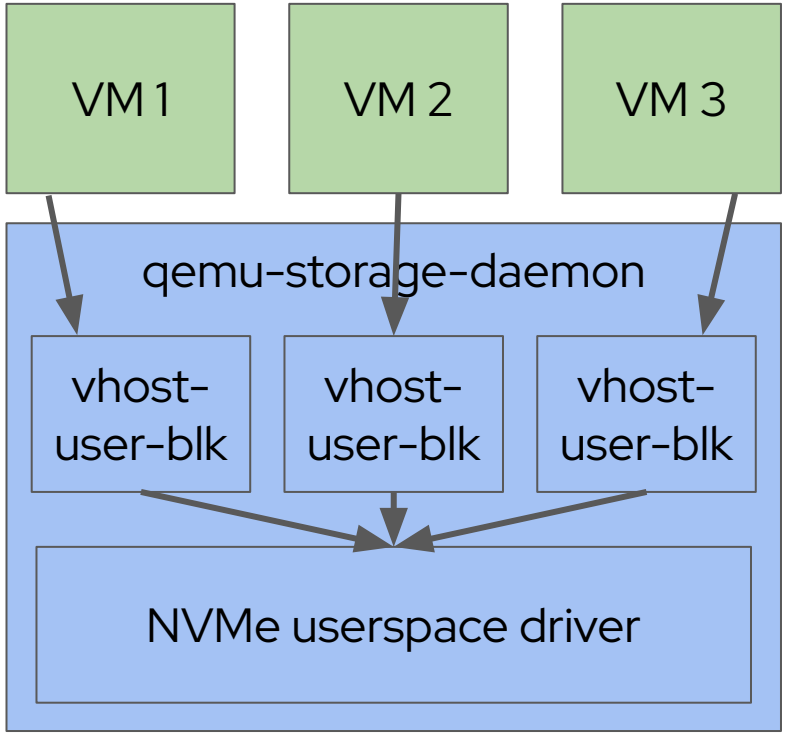
Experimental QEMU feature for high-performance use cases

NVMe userspace driver with qemu-storage-daemon

qemu-storage-daemon handles I/O from all VMs

One NVMe drive can be shared by multiple VMs

QEMU connects to the vhost-user-blk export



Red Hat

qemu-img

qemu-img disk image tool

convert - import/export between formats

create - initialize empty disk image file

snapshot - manipulate internal snapshots

commit & rebase - manipulate backing files

resize - grow/shrink

check - consistency check (like fsck)

info - query details like disk usage, etc

And lots more

```
$ man qemu-img
```

Summary

- Features for running QEMU VMs:
  - Disk images, monitor commands, filters like I/O throttling
  - Userspace NVMe driver for high-performance use cases
- Features independent of VMs (qemu-storage-daemon):
  - All the above features
  - Exporting blockdevs over NBD and other protocols
- Tools:
  - qemu-img to create, inspect, and manipulate disk images

#qemu on irc.oftc.net

Thank you

Disk images: https://www.qemu.org/docs/master/system/images.html

qemu-img: https://www.qemu.org/docs/master/tools/qemu-img.html

qemu-storage-daemon:
https://www.qemu.org/docs/master/tools/qemu-storage-daemon.html

Monitor commands:
https://www.qemu.org/docs/master/interop/live-block-operations.html

Incremental backups: https://www.qemu.org/docs/master/interop/bitmaps.html