Live Block Device Operations in QEMU

Kashyap Chamarthy <kashyap@redhat.com>

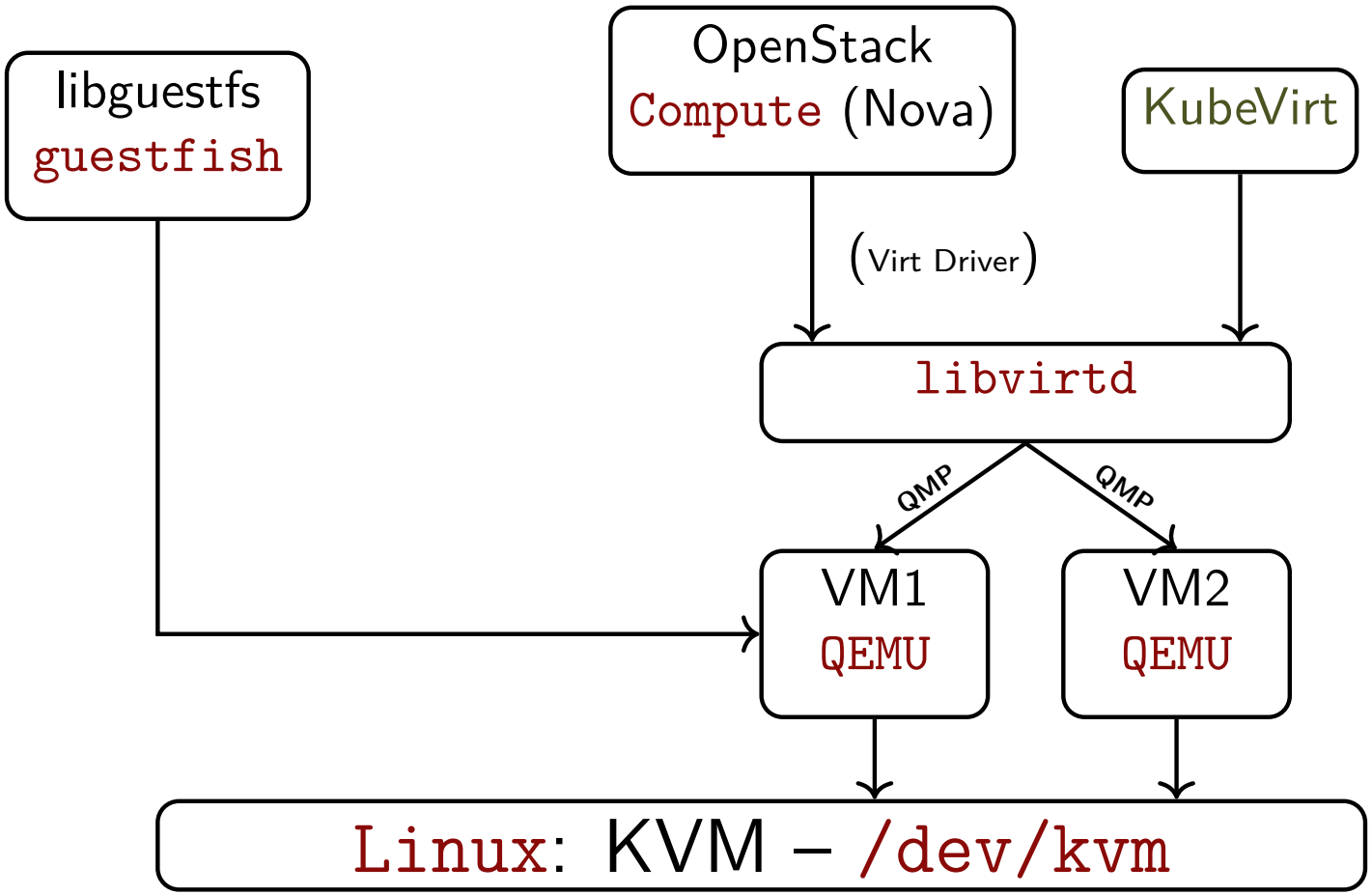
FOSDEM 2018

Brussels

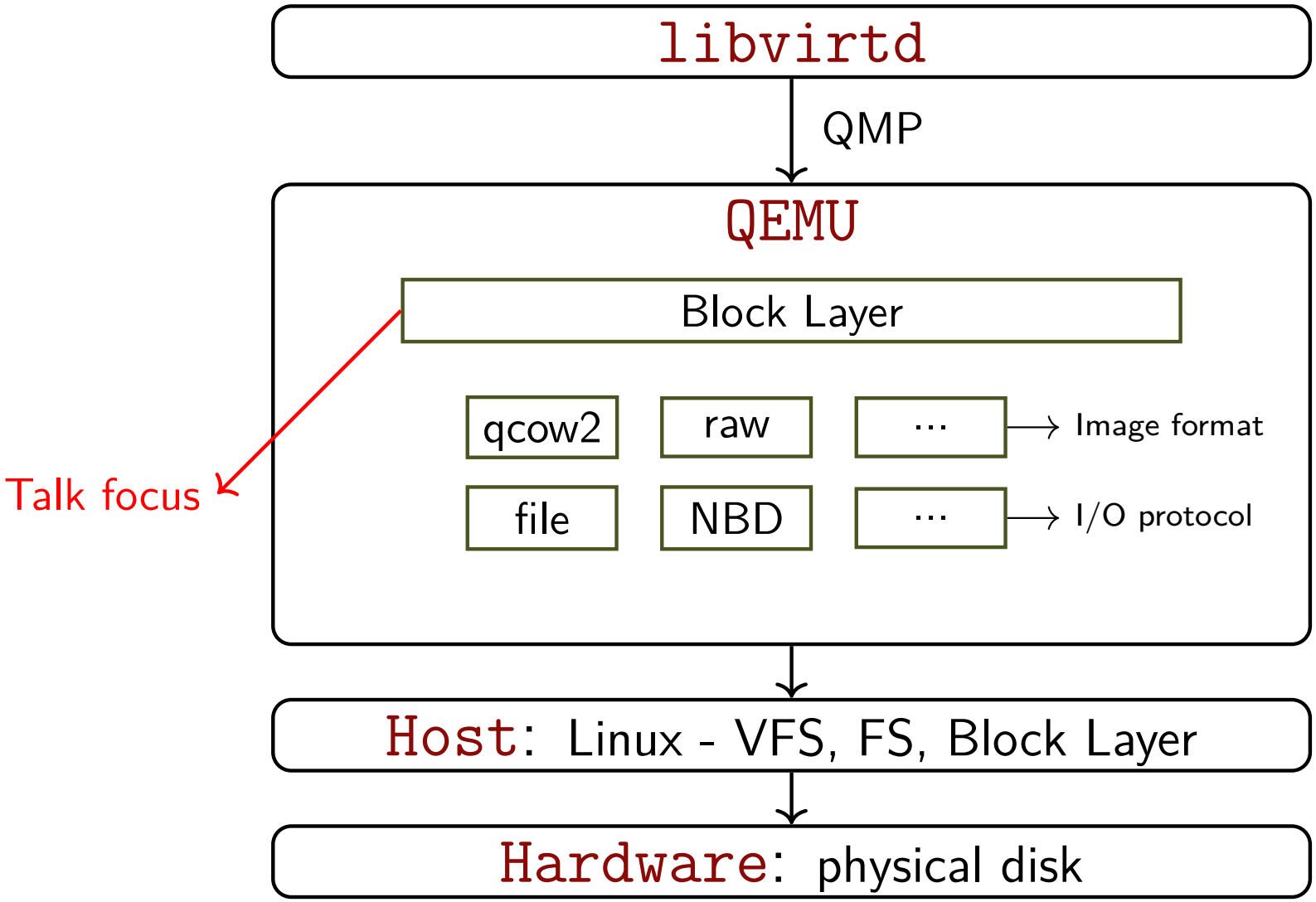
Part I

Background

KVM / QEMU virtualization components



In this talk


libvirtd
QMP
QEMU
Block Layer
qcow2
raw
···
Image format
file
NBD
···
I/O protocol
Talk focus
Host: Linux - VFS, FS, Block Layer
Hardware: physical disk

QEMU's block subsystem

Emulated storage devices: SCSI, IDE, virtio-blk, ...

```
$ qemu-system-x86_64 -device help
```

Block driver types:

- Format: qcow2, raw, vmdk
- I/O Protocol: NBD, file, RBD/Ceph

Block device operations:

- Offline image manipulation: `qemu-img`, `qemu-nbd`
- Live: snapshots, image streaming, storage migration, ...

QEMU's block subsystem

Emulated storage devices: SCSI, IDE, virtio-blk, ...

```
$ qemu-system-x86_64 -device help
```

Look for "Storage devices:"

Block driver types:

- Format: qcow2, raw, vmdk
- I/O Protocol: NBD, file, RBD/Ceph

Block device operations:

- Offline image manipulation: `qemu-img`, `qemu-nbd`
- Live: snapshots, image streaming, storage migration, ...

QEMU Copy-On-Write overlays

– Read from the overlay if allocated, otherwise from base
– Write to overlay only

Use cases: Thin provisioning, snapshots, backups, ...

QEMU Copy-On-Write overlays

– Read from the overlay if allocated, otherwise from base
– Write to overlay only

Use cases: Thin provisioning, snapshots, backups, ...

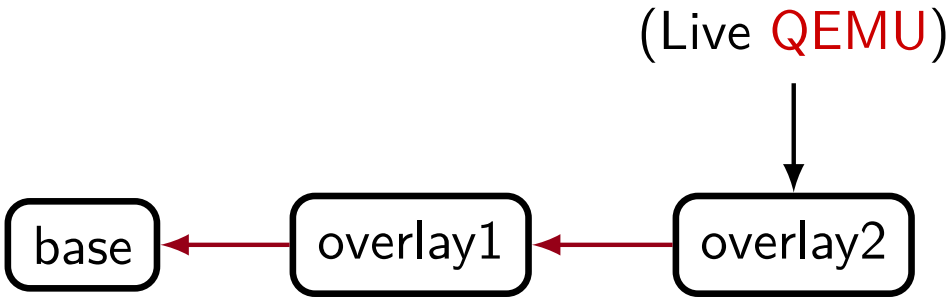
Create a minimal backing chain:

```
$ qemu-img create -f raw base.raw 5G
```

```
$ qemu-img create -f qcow2 overlay.qcow2 2G \
      -b base.raw -F raw
```

(Backing file) (Backing file format)

Accessing disk images opened by QEMU



Disk images that are opened by QEMU should not be accessed by external tools

– QEMU offers equivalent monitor commands

For safe, read-only access, use libguestfs:

```
$ guestfish -ro -i -a disk.img
```

Disk image locking

Prevents two concurrent writers to a disk image

– Using Linux Open File Description (OFD) Locks

To query an in use disk image:

```
$ qemu-img info foo.qcow2 --force-share
```

Disk image locking

Prevents two concurrent writers to a disk image

– Using Linux Open File Description (OFD) Locks

To query an in use disk image:

```
$ qemu-img info foo.qcow2 --force-share
```

Allows read-only access to an active disk; may return stale data

Disk image locking

Prevents two concurrent writers to a disk image

– Using Linux Open File Description (OFD) Locks

To query an in use disk image:

```
$ qemu-img info foo.qcow2 --force-share
```

When launching QEMU (2.10+):

```
$ qemu-system-x86_64\
   -blockdev driver=qcow2,file.driver=file\
   file.filename=./foo.qcow2,file.locking=auto,\
   [...]
```

Disk image locking

Prevents two concurrent writers to a disk image

– Using Linux Open File Description (OFD) Locks

To query an in use disk image:

```
$ qemu-img info foo.qcow2 --force-share
```

When launching QEMU (2.10+):

```
$ qemu-system-x86_64\
   -blockdev driver=qcow2,file.driver=file\
   file.filename=./foo.qcow2,file.locking=auto,\
   [...]
```

Defaults to OFD locking on Linux 3.15+

redhat.

Part II

Primer on operating QEMU

QEMU's QMP monitor

Provides a JSON RPC interface

- Send commands to query / modify VM state
- QMP (asynchronous) events on certain state changes

If you zoom into libvirt-generated QEMU command-line:

```
/usr/bin/qemu-system-x86_64 [...]\
 -chardev socket,id=charmonitor,\
 path=[...]/monitor.sock,server,nowait\
 -mon chardev=charmonitor,id=monitor,mode=control
```

QEMU's QMP monitor

Provides a JSON RPC interface

- Send commands to query / modify VM state
- QMP (asynchronous) events on certain state changes

If you zoom into libvirt-generated QEMU command-line:

```
/usr/bin/qemu-system-x86_64 [...]\
 -chardev socket,id=charmonitor,\
 path=[...]/monitor.sock,server,nowait\
 -mon chardev=charmonitor,id=monitor,mode=control
```

UNIX stream socket setup for libvirt $\longleftrightarrow$ QEMU communication

QEMU's QMP monitor

Provides a JSON RPC interface

- Send commands to query / modify VM state
- QMP (asynchronous) events on certain state changes

If you zoom into libvirt-generated QEMU command-line:

```
/usr/bin/qemu-system-x86_64 [...]\
 -chardev socket,id=charmonitor,\
 path=[...]/monitor.sock,server,nowait\
 -mon chardev=charmonitor,id=monitor,mode=control
```

Shorthand for the above:

```
    -qmp unix:./qmp-sock,server,nowait
```

Interacting with the QMP monitor

```
$ socat UNIX:./qmp-sock \
  READLINE,history=$HOME/.qmp_history \
{"QMP": {"version":
        {"qemu": {"micro": 50, "minor": 11, "major": 2},
        "package": "(v2.11.0-355-g281f327487)"},
        "capabilities": []}}
{"execute": "qmp_capabilities"}
{"return": {}}

{"execute": "query-status"}
{"return": {"status": "running", "singlestep": false,
            "running": true} }
```

Send arbitrary commands: `query-kvm`, `blockdev-backup`, ...

Interacting with the QMP monitor

```
$ socat UNIX:./qmp-sock \
  READLINE,history=$HOME/.qmp_history \
{"QMP": {"version":
        {"qemu": {"micro": 50, "minor": 11, "major": 2},
        "package": "(v2.11.0-355-g281f327487)"},
        "capabilities": []}}
{"execute": "qmp_capabilities
{"return": {}}

{"execute": "query-status"}
{"return": {"status": "running", "singlestep": false,
            "running": true} }
```

Indicates successful connection

Send arbitrary commands: `query-kvm`, `blockdev-backup`, ...

Interacting with the QMP monitor

```
$ socat UNIX:./qmp-sock \
  READLINE,history=$HOME/.qmp_history \
{"QMP": {"version":
        {"qemu": {"micro": 50, "minor": 11, "major": 2},
        "package": "(v2.11.0-355-g281f327487)"},
        "capabilities": []}}
{"execute": "qmp_capabilities"}
{"return": {}}

{"execute": "query-status"}
{"return": {"status": "running", "singlestep": false,
            "running": true} }
```

Prerequisite

Send arbitrary commands: `query-kvm`, `blockdev-backup`, ...

Interacting with the QMP monitor

```
$ socat UNIX:./qmp-sock \
  READLINE,history=$HOME/.qmp_history \
{"QMP": {"version":
        {"qemu": {"micro": 50, "minor": 11, "major": 2},
        "package": "(v2.11.0-355-g281f327487)"},
        "capabilities": []}}
{"execute": "qmp_capabilities"}
{"return": {}}

{"execute": "query-status"}
{"return": {"st                                    lse,
            "ru
```

Issue regular QMP commands

Send arbitrary commands: `query-kvm`, `blockdev-backup`, ...

Interacting with the QMP monitor

```
$ socat UNIX:./qmp-sock \
  READLINE,history=$HOME/.qmp_history \
{"QMP": {"version":
        {"qemu": {"micro": 50, "minor": 11, "major": 2},
        "package": "(v2.11.0-355-g281f327487)"}
```

Invoking JSON manually is no fun — thankfully, libvirt automates it all

```
{"execute": "query-status"}
{"return": {"status": "running", "singlestep": false,
            "running": true} }
```

Send arbitrary commands: `query-kvm`, `blockdev-backup`, ...

Other ways to interact with QMP monitor

qmp-shell: A low-level tool located in QEMU source.
Takes key-value pairs (& JSON dicts):

```
$ qmp-shell -v -p ./qmp-sock
(QEMU) block-job-complete device=virtio1
```

virsh: libvirt's shell interface

```
$ virsh qemu-monitor-command \
   vm1 --pretty '{"execute":"query-kvm"}'
```

NB: Modifying VM state behind libvirt's back voids support warranty!

$\rightsquigarrow$ Useful for test / development

redhat.

Part III

Configuring block devices

Aspects of a QEMU block device

QEMU block devices have a notion of a:

Frontend — guest-visible devices (IDE, SCSI, virtio-blk, ...)

- `-device`: command-line
- `device_add`: run-time; like any other type of guest device

Backend — block drivers (NBD, qcow2, raw, ...)

- `-drive` (legacy) / `-blockdev`: command-line
- `blockdev-add`: run-time

Configure on command-line: `-blockdev`

Provides fine-grained control over configuring block devices

Since QEMU 2.9; successor to `-drive` option

E.g. Attach a `qcow2` disk to a `virtio-blk` guest device:

```
$ qemu-system-x86_64 [...]\
   -blockdev node-name=node1,driver=qcow2,\
   file.driver=file,file.filename=./base.qcow2\
   -device virtio-blk,drive=node-Base
```

Configure on command-line: `-blockdev`

Provides fine-grained control over configuring block devices

Since QEMU 2.9; successor to `-drive` option

E.g. Attach a `qcow2` disk to a `virtio-blk` guest device:

```
$ qemu-system-x86_64 [...]\
    -blockdev node-name=node1,driver=qcow2,\
    file.driver=file,file.filename=./base.qcow2\
    -device virtio-blk,dri
```

Configures the 'backend'

Configure on command-line: `-blockdev`

Provides fine-grained control over configuring block devices

Since QEMU 2.9; successor to `-drive` option

E.g. Attach a `qcow2` disk to a `virtio-blk` guest device:

```
$ qemu-system-x86_64 [...]\
   -blockdev node-name=node1,driver=qcow2,\
   file.driver=file,file.filename=./base.qcow2\
   -device virtio-blk,drive=node-Base
```

Configures the 'frontend' (guest-visible device)

Configure on command-line: `-blockdev`

Provides fine-grained control over configuring block devices

Since QEMU 2.9; successor to `-drive` option

E.g. Attach a `qcow2` disk to a `virtio-blk` guest device:

```
$ qemu-system-x86_64 [...]\
   -blockdev node-name=node1,driver=qcow2,\
   file.driver=file,file.filename=./base.qcow2\
   -device virtio-blk,drive=node-Base
```

→ More details: Talks from previous KVM Forums

Configure at run-time: QMP `blockdev-add`

Add a `qcow2` block device at run-time:

```
{"execute": "blockdev-add", "arguments": {
    "driver": "qcow2", "node-name": "node-A",
    "file": {"driver": "file",
             "filename": "disk-A.qcow2"}}}
```

Command-line is 1:1 mapping of JSON (from above):

```
-blockdev driver=qcow2,node-name=node-A,\
file.driver=file,file.filename=./disk-A.qcow2
```

→ Here too, refer to previous KVM Forum talks

redhat.

Part IV

Live block operations

`blockdev-snapshot-sync`: External snapshots

When invoked while the guest is running:

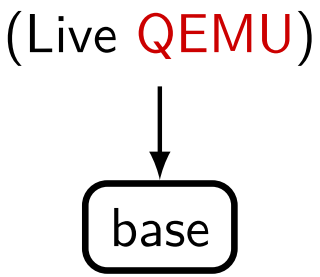
1. the **existing disk becomes the backing file**; and
2. a **QCOW2 overlay** is created **to track new writes**

Base image can be of any format; overlays must be QCOW2

Allows atomic live snapshot of multiple disks

No guest downtime — snapshot creation is instantaneous

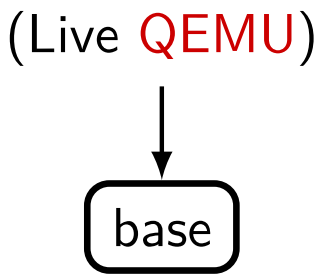
`blockdev-snapshot-sync`: Example



Create an external snapshot (`qmp-shell` invocation):

```
blockdev-snapshot-sync node-name=node-Base \
 snapshot-file=overlay1.qcow2 \
 snapshot-node-name=node-Overlay1
```

`blockdev-snapshot-sync`: Example



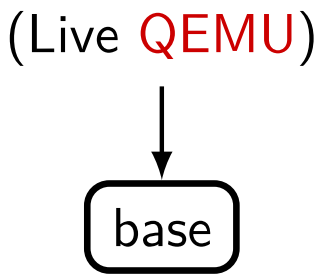
Create an external snapshot (`qmp-shell` invocation):

```
blockdev-snapshot-sync node-name=node-Base \
 snapshot-file=overlay1.qcow2 \
 snapshot-node-name=node-Overlay1
```

libvirt equivalent:

```
$ virsh snapshot-create-as vm1 --disk-only
```

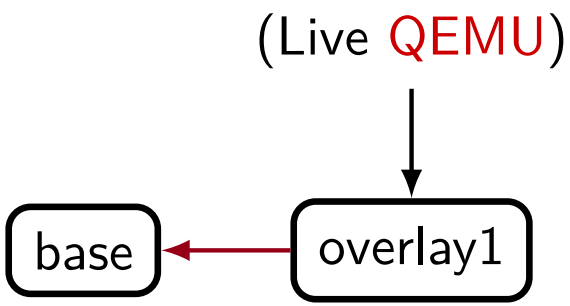
`blockdev-snapshot-sync`: Example



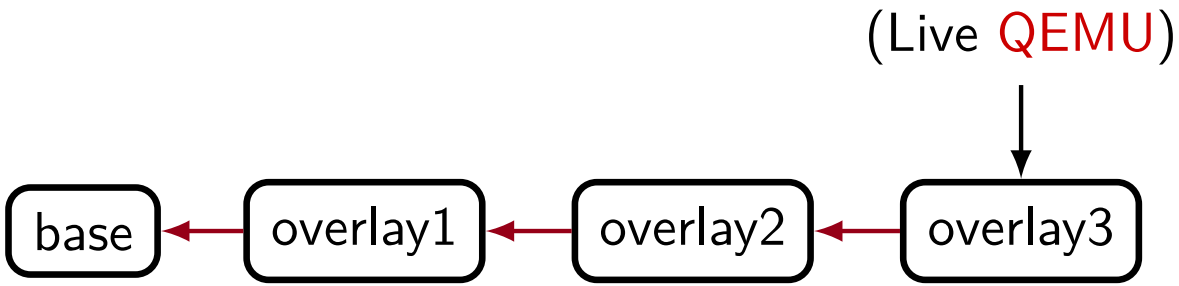
Create an external snapshot (`qmp-shell` invocation):

```
blockdev-snapshot-sync node-name=node-Base \
 snapshot-file=overlay1.qcow2 \
 snapshot-node-name=node-Overlay1
```

Result:



But...long image chains can get cumbersome

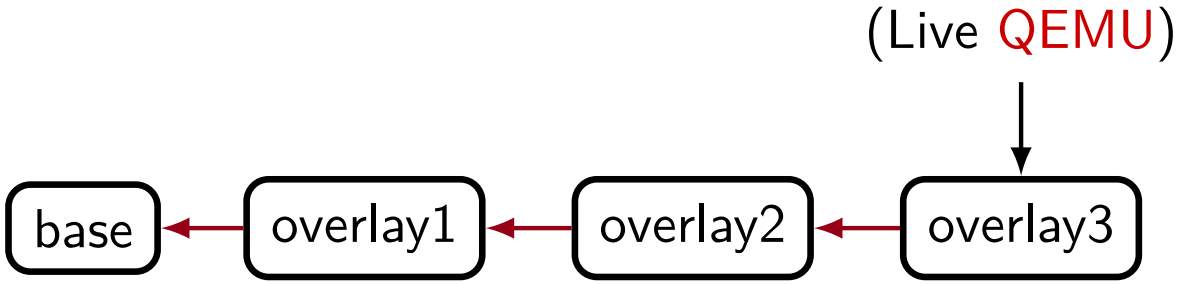


Problems:

- Revert to external snapshot is non-trivial
- Multiple files to track
- I/O penalty with a long disk image chain

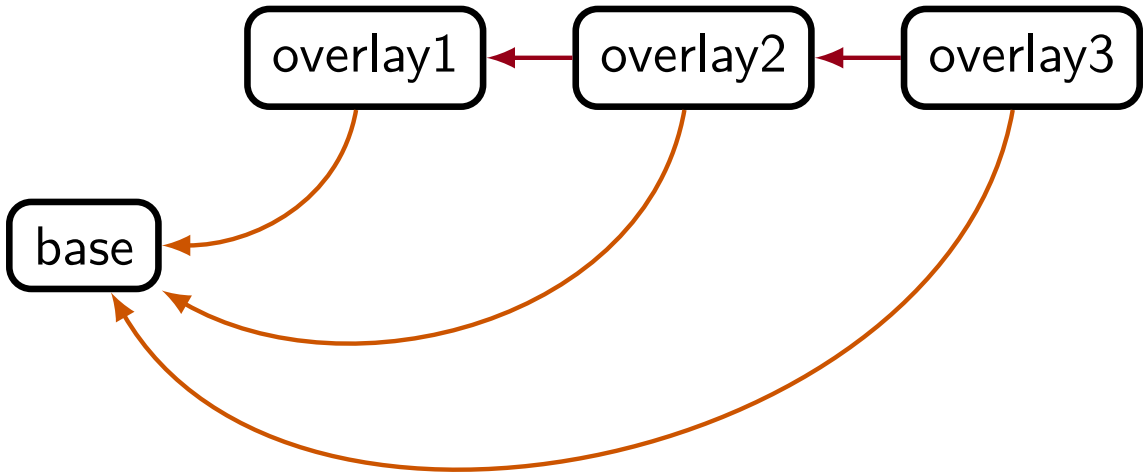
There are some solutions. . .

`commit`: Live merge a disk image chain (1)

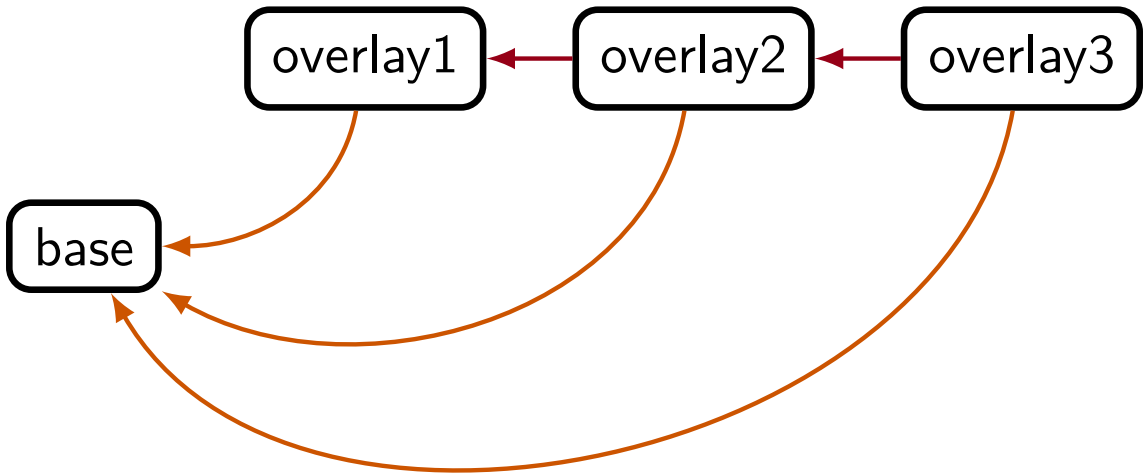


Problem: Shorten the chain of overlays

Simplest case: Merge all of them into 'base'



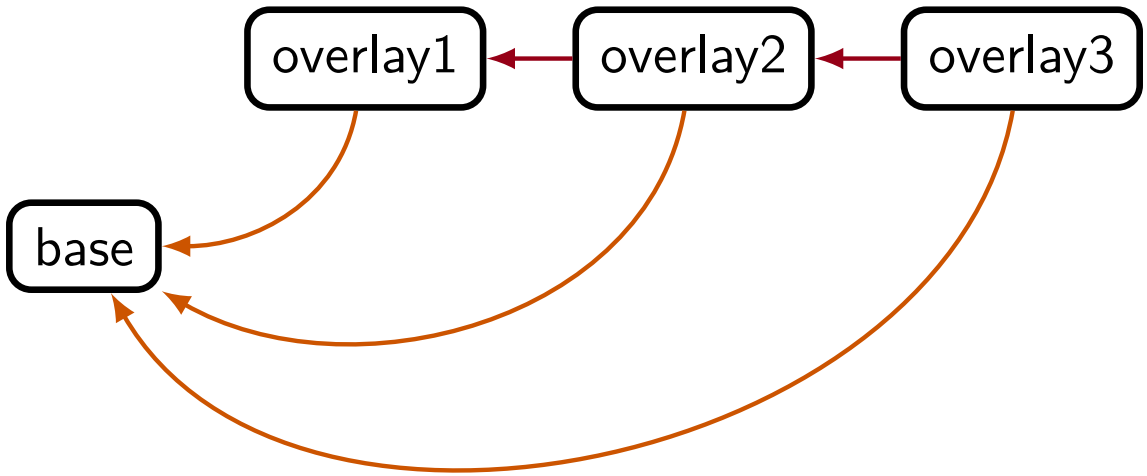
`commit`: Live merge a disk image chain (2)



Run-time invocation (using `qmp-shell`):

```
blockdev-snapshot-sync [...]
block-commit device=node-Overlay3 job-id=jobA
block-job-complete device=jobA
```

`commit`: Live merge a disk image chain (2)

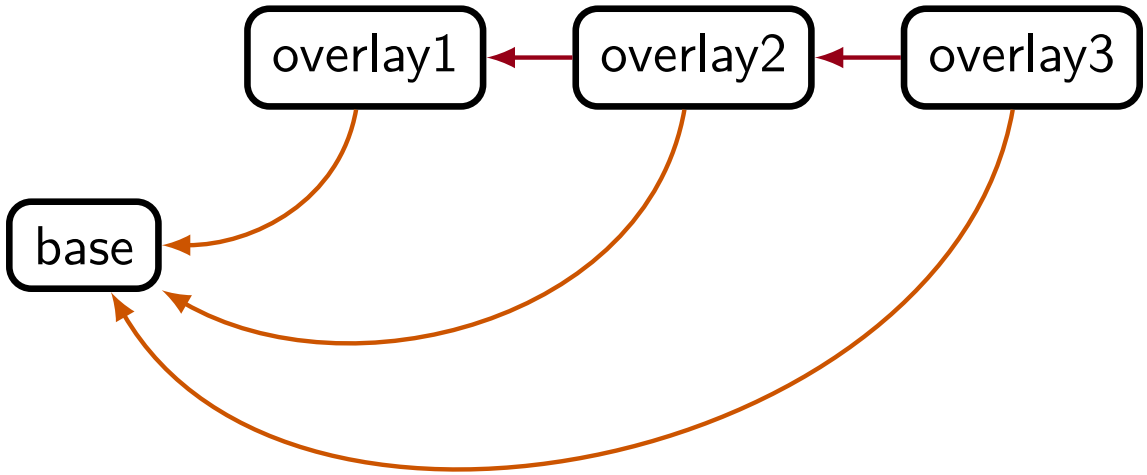


Run-time invocation (using `qmp-shell`):

```
blockdev-snapshot-sync [...]
block-commit
block-job-com
```

Invoke it thrice – to create 3 overlays

`commit`: Live merge a disk image chain (2)

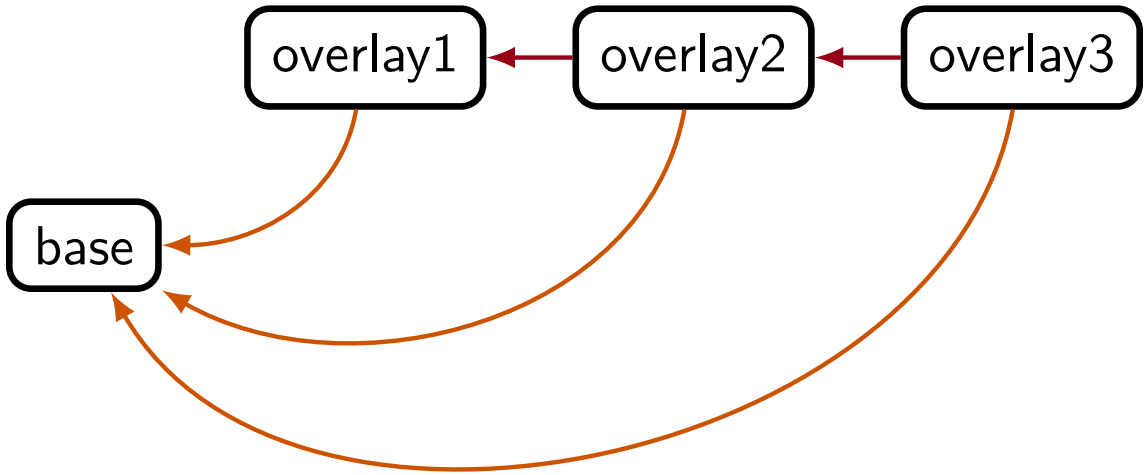


Run-time invocation (using `qmp-shell`):

```
blockdev-snapshot-sync [...]
block-commit device=node-Overlay3 job-id=jobA
block-job-com
```

Copy content from all the overlays into base

`commit`: Live merge a disk image chain (2)

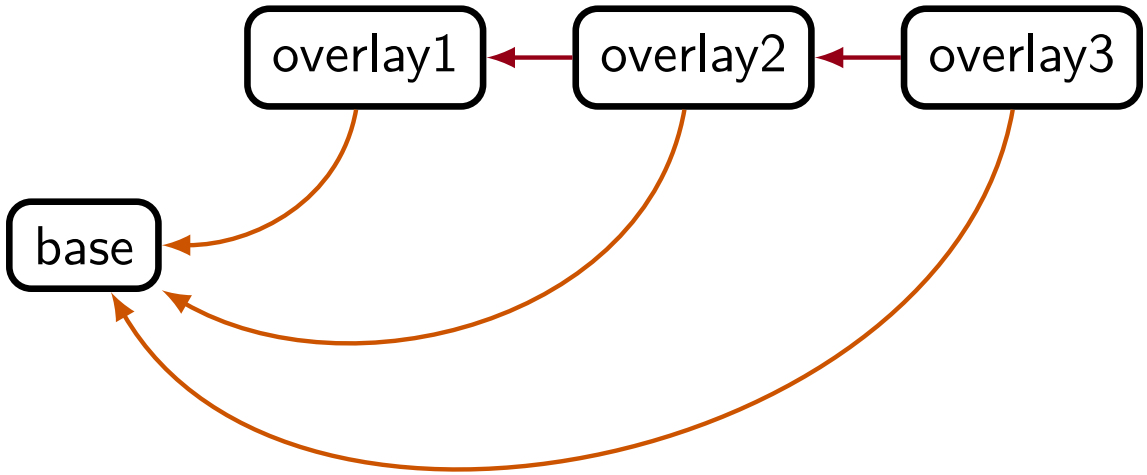


Run-time invocation (using `qmp-shell`):

```
blockdev-snapshot-sync [...]
block-commit device=node-Overlay3 job-id=jobA
block-job-complete device=jobA
```

Gracefully end the 'commit' job, and pivot QEMU

`commit`: Live merge a disk image chain (2)



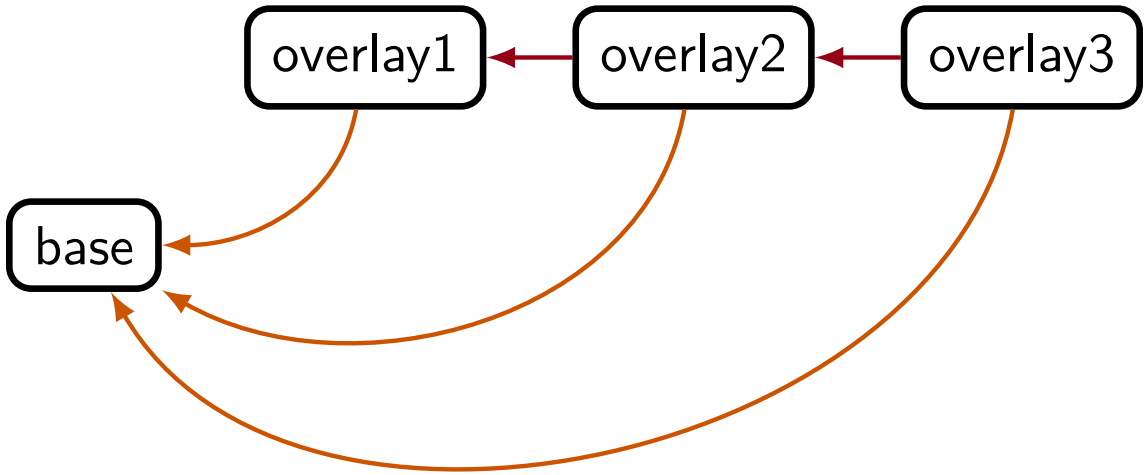
Run-time invocation (using `qmp-shell`):

```
blockdev-snapshot-sync [...]
block-commit device=node-Overlay3 job-id=jobA
block-job-complete device=jobA
```

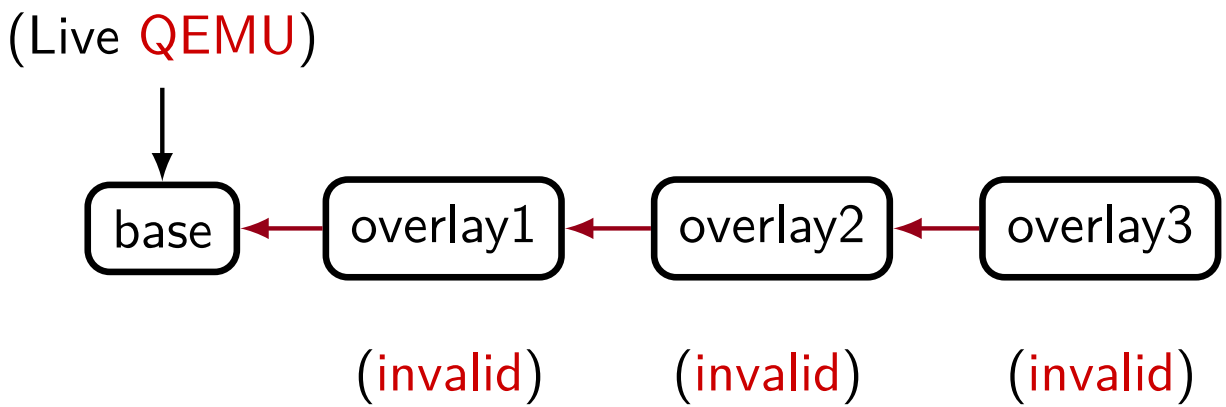
libvirt equivalent:

```
$ virsh blockcommit vm1 vda --pivot
```

`commit`: Live merge a disk image chain (3)



Two phase (**sync** + **pivot**) operation = a consolidated disk:



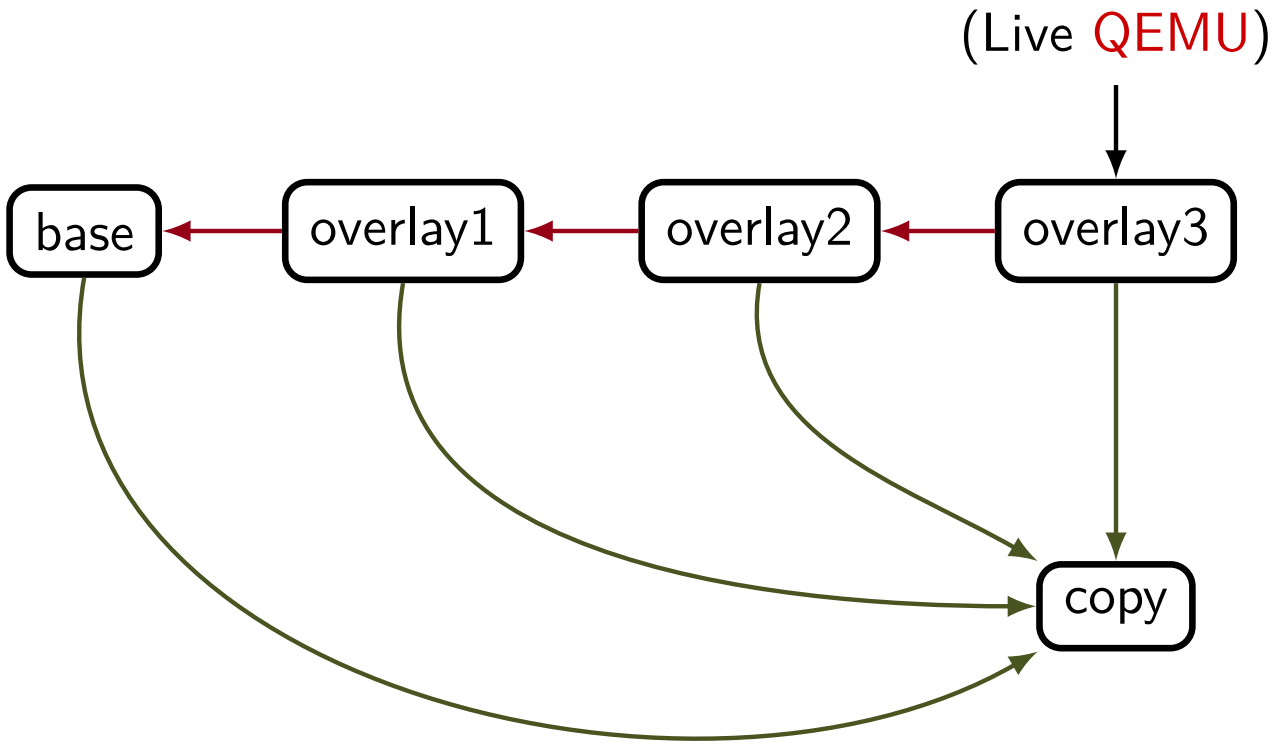
`stream`: Copy from backing files to overlays

A bit similar to `'commit'`, but in the *opposite* direction

Specifics:

- `'stream'` operation is safe – data is being pulled forward
- Intermediate overlays remain valid – *unlike* `'commit'`
- Intermediate image streaming (from QEMU 2.8+)

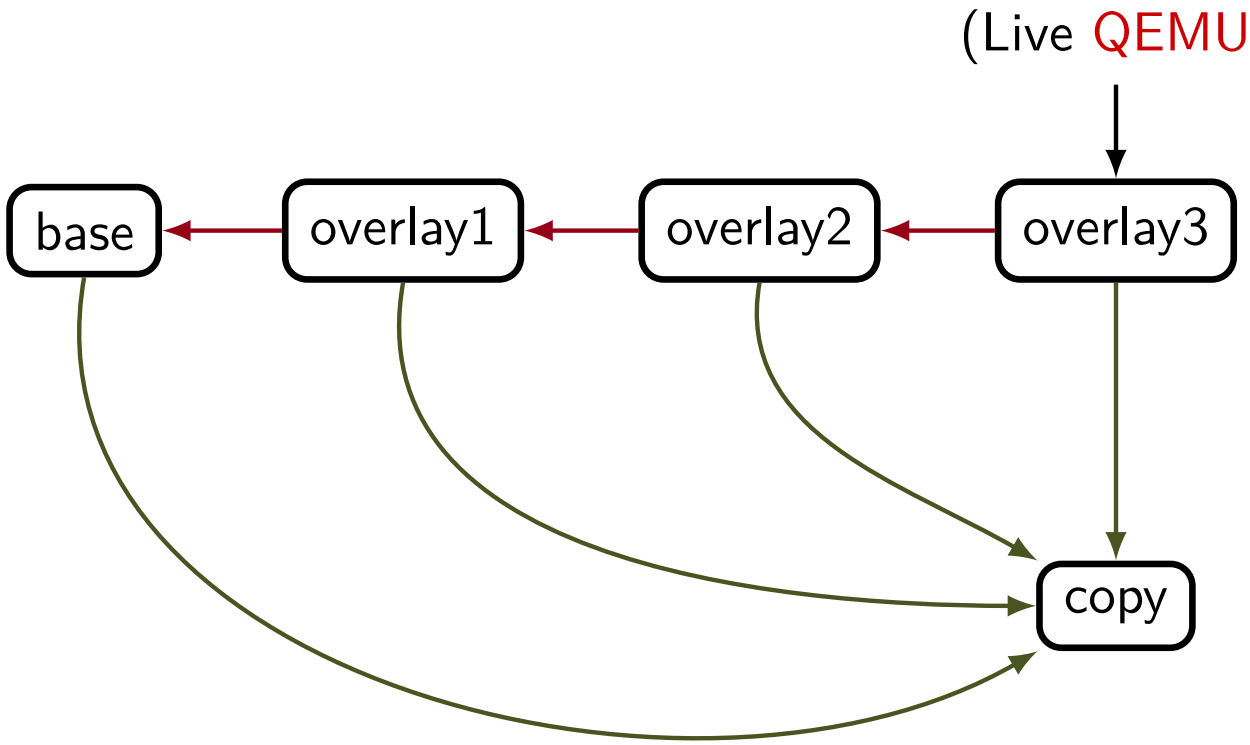
`mirror`: Synchronize active disk to another image



Synchronization modes:

- `'full'` – copy the **entire chain**
- `'top'` – only from the **topmost (active) image**
- `'none'` – copy only **new writes** from now on

`mirror`: Operation




```
drive-mirror [...] target=copy1.qcow2 sync=full
query-block-jobs
block-job-complete device=virtio0
```

QEMU NBD server

Network Block Device server — built into QEMU

– Lets you export images *while in use*

QMP commands:

```
nbd-server-start addr={"type":"unix",
                       "data":{"path":"./nbd-sock"}}}
```

```
nbd-server-add device=TargetDisk writable=true
```

```
nbd-server-stop
```

→ External program for offline use: `qemu-nbd`

Combining 'mirror' and NBD

Use case: Live VM migration with non-shared storage

1. Destination QEMU sets up the NBD server
2. Source QEMU issues `drive-mirror` to synchronize disk(s):

```
{"execute": "drive-mirror",\
 "arguments":{
    "device": "disk0",\
    "target": "nbd:dest:49153:exportname=disk0",\
    "sync": "top", "mode": "existing"
}
```

⇝ Details: qemu/docs/interop/live-block-operations.rst

`mirror` + NBD: libvirt automation

NBD-based live storage migration as done by libvirt:

```
$ virsh migrate \
  --live \
  --verbose \
  --p2p \
  --copy-storage-all \
  vm1 \
  qemu+ssh://root@desthost/system
```

Higher layers, such as OpenStack, use the equivalent libvirt Python APIs: `migrateToURI[2,3]()`

`backup`: Point-in-time copy of a block device

Point-in-time:

- For `backup`: when you ***start*** **the operation**
- For `mirror`: when you ***end*** **the sync** (via `block-job-complete`)

Synchronization modes for `backup`:

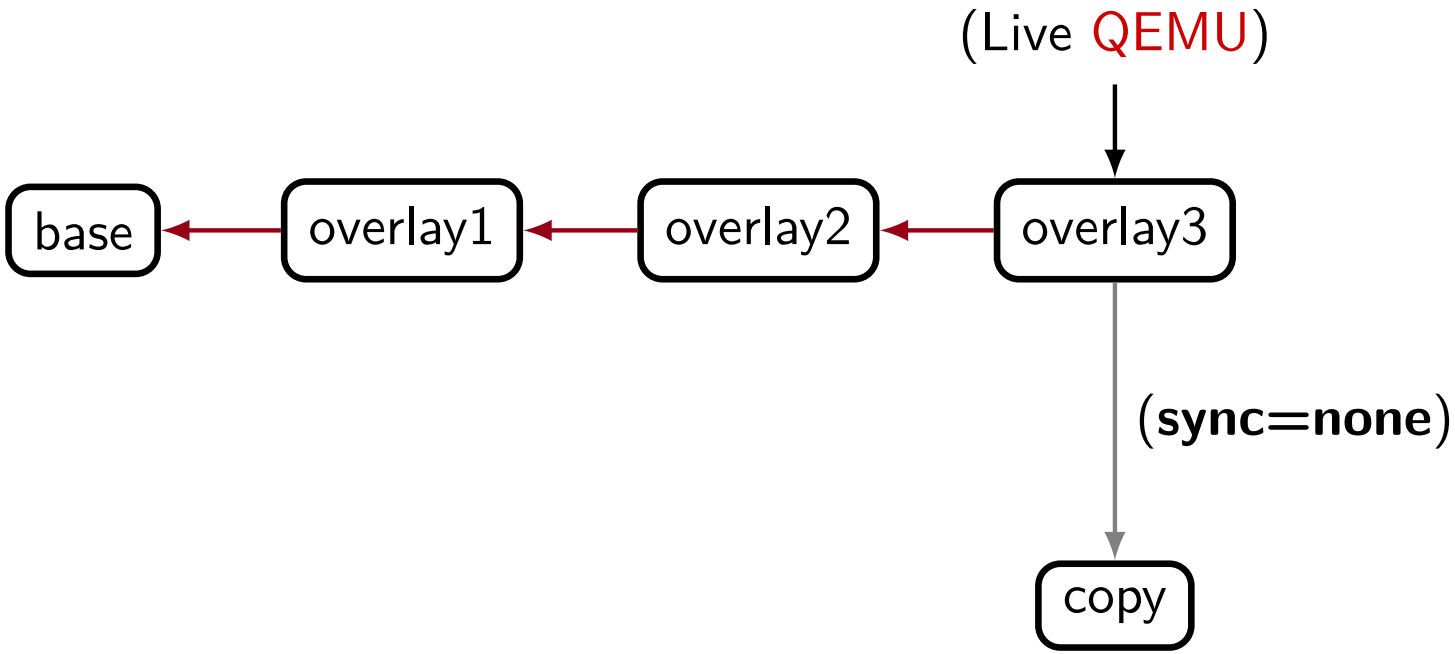
- `'top'`, `'full'`, `'none'`
- `'incremental'`

(For incremental backups; WIP as of 2.12)

⇝ `qemu/docs/interop/bitmaps.rst`

Combining ‘backup’ and NBD

Use case: Examine guest I/O patterns




```
{"execute":"blockdev-backup",
   "arguments":{"device": "node-Overlay3", "target": "copy",
                "job-id": "job0", "sync": "none"}}
[...] # Start NBD server and export the ‘copy’
```

Summary

`commit`: Move data from overlays into backing files

`stream`: Move data from backing files into overlays

`mirror`: Live VM migration with non-shared storage

`backup`: Point-in-time copy

⇝ `qemu/docs/interop/live-block-operations.rst`
⇝ `qemu/docs/interop/bitmaps.rst`

References

Detailed docs with examples: Live Block Device Operations
https://kashyapc.fedorapeople.org/QEMU-Docs/_build/html/docs/live-block-operations.html

"Incremental Backups - Good things come in small packages!" by John Snow
https://fosdem.org/2017/schedule/event/backup_dr_incr_backups/

"Managing the New Block Layer" by Kevin Wolf & Max Reitz
https://events.static.linuxfound.org/sites/events/files/slides/talk_11.pdf

"Backing Chain Management in libvirt and qemu" by Eric Blake
http://events.linuxfoundation.org/sites/events/files/slides/2015-qcow2-expanded.pdf

"Towards a more expressive and introspectable QEMU command line"
https://events.static.linuxfound.org/sites/events/files/slides/armbru-qapi-cmdline_1.pdf

Questions?