Managing the New Block Layer

Kevin Wolf <kwolf@redhat.com>
Max Reitz <mreitz@redhat.com>

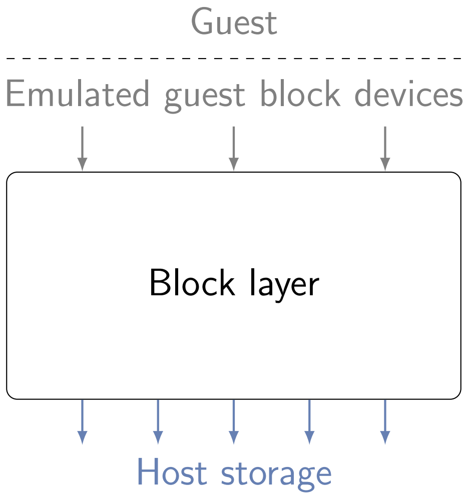
KVM Forum 2017

Part I
User management

Section 1

The New Block Layer

Block layer role



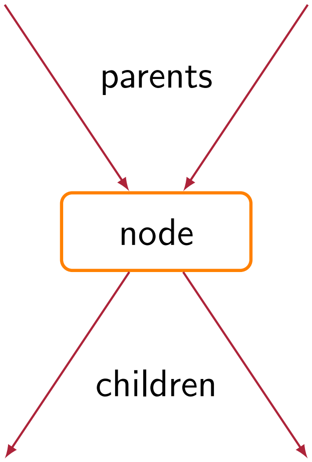
Block layer duties

- Read/write data from/to host storage (outside of QEMU)
- Interpret image formats
- Manipulate data on the way:
  - Encryption
  - Throttling
  - Duplication

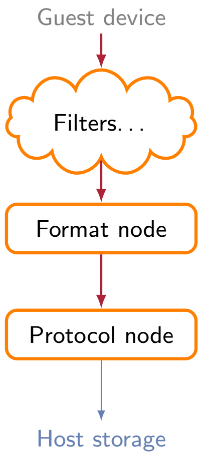
Block drivers

- Accessing host storage:
  Protocol drivers (e.g. `file`, `nbd`)
- Interpret image formats:
  Format drivers (e.g. `qcow2`)
- Data manipulation:
  Filter drivers (e.g. `throttle`, `quorum`)

Block driver “instantiation”



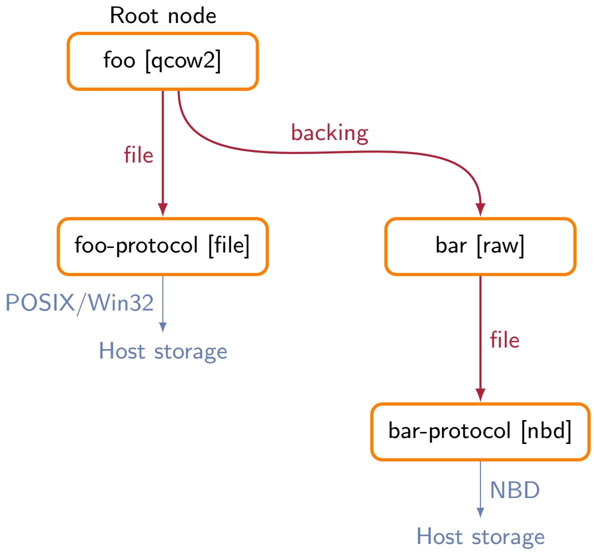
General block layer structure



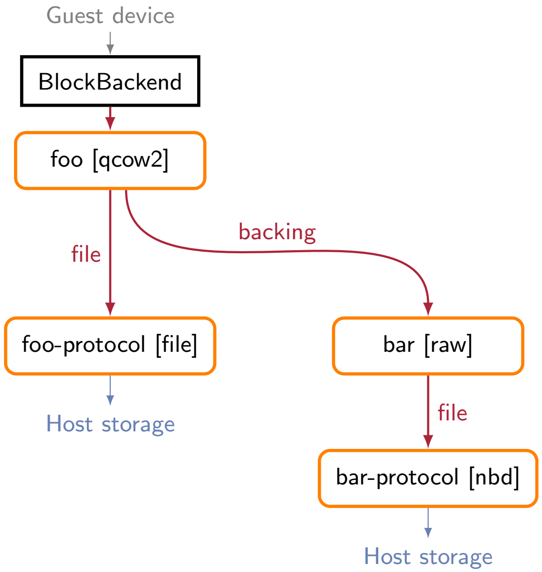
Block trees

From Minecraft

Growing a tree



Rooting the tree



Filters

- Format nodes have metadata, filters do not
  $\Rightarrow$ can put filters anywhere into the graph
- Throttling: Was basically at the device; can now be put anywhere
- Quorum: Data duplication; arbitrarily stackable (or you can throttle individual children)

Management – how and why

- Tree construction
- Runtime modifications
- Why?
  - Runtime block device configuration
  - Filter driver configuration
  - External snapshots
  - ...
- Op blockers to keep it safe

Section 2
Tree construction

Node configuration: Runtime options (1)

Generally:

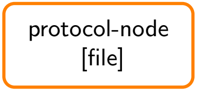
- `driver`: String (mandatory)
- `node-name`: String (mandatory for root nodes)

Specific options, e.g. for `file`:

- `filename`: String (mandatory)
- . . . (see QMP reference, `BlockdevOptionsFile` object)

Node configuration: Example (1)

```
{ "driver": "file",
  "node-name": "protocol-node",
  "filename": "foo.qcow2" }
```



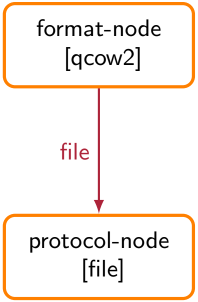
Node configuration: Runtime options (2)

Specific options for `qcow2`:

- `file`: Reference to a node (mandatory)
- ... (see QMP reference, `BlockdevOptionsQcow2` object)

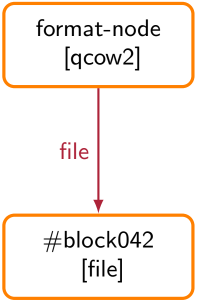
Node configuration: Example (2a)

```
{ "driver": "qcow2",
  "node-name": "format-node",
  "file": "protocol-node" }
```



Node configuration: Example (2b)

```
{ "driver": "qcow2",
  "node-name": "format-node",
  "file": {
      "driver": "file",
      "filename": "foo.qcow2"
  } }
```



Passing this JSON object into QEMU

QMP command: `blockdev-add`

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

Passing this JSON object into QEMU

Command line option: `-blockdev`

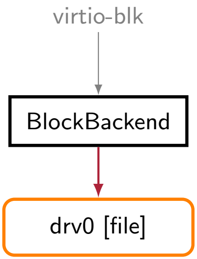
```
-blockdev '{
    "driver": "file",
    "node-name": "protocol-node",
    "filename": "foo.qcow2"
}'
```

Rooting block trees

Both `-device` and `device_add`:

Pass the root's `node-name` to the `drive` property

```
-blockdev '{ "driver": "file",
    "node-name": "drv0",
    "filename": "foo.raw" }' \
\
-device virtio-blk,drive=drv0
```



“Hey, what about `-drive`?”

Why you should no longer use `-drive`:

- Does not directly correspond to the QAPI schema
  - Has a different `file`
  - Has format probing
- All in all: Evolved into kind of a monstrosity
- With anything but `if=none`: Creates guest device
- With `if=none`: Creates `BlockBackend`

So what about `BlockBackend` now?

You should not worry about it.

- Only used internally now
- `-blockdev` + `-device` create it automatically
- Block trees are identified through the root's `node-name`

Section 3

Runtime configuration

`blockdev-del`

- Counterpart to `blockdev-add`

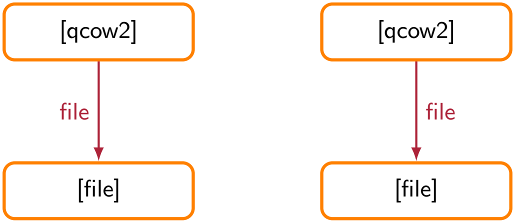
Details:

- Nodes are refcounted
- Automatic deletion when refcount reaches 0
- Nodes added with `blockdev-add` therefore must have a strong reference from the monitor – `blockdev-del` deletes this
  - Cannot `blockdev-del` in-use nodes

Graph manipulation (1)

Present: `blockdev-snapshot`
(and `blockdev-snapshot-sync`)

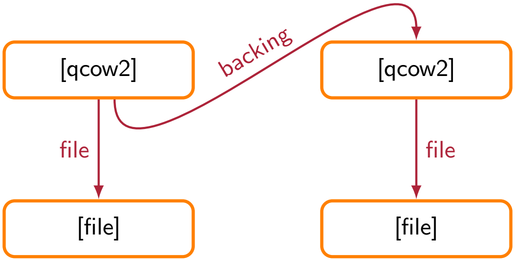
- Attach a node to another node as the latter's backing child



Graph manipulation (1)

Present: `blockdev-snapshot`
(and `blockdev-snapshot-sync`)

- Attach a node to another node as the latter's backing child



Graph manipulation (2)

Begun: `x-blockdev-change`

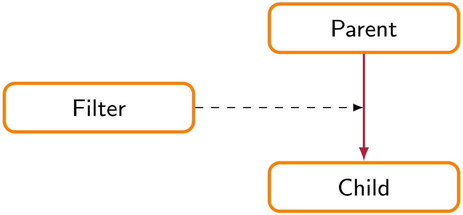
- Add/remove children to/from a block node
  - Currently only for quorum
  - For adding backing children: `blockdev-snapshot`
- Note: Most children are not optional
- Not yet implemented: Node replacement

Graph manipulation (3)

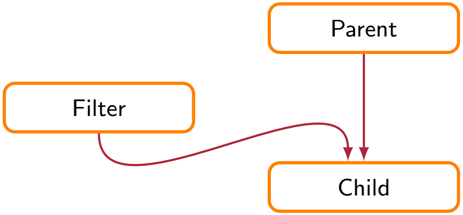
Proposal: `blockdev-insert-node` and `blockdev-remove-node`

- Effectively insert a new node between two existing nodes, or undo this operation
- Functionally a node replacement with various constraints

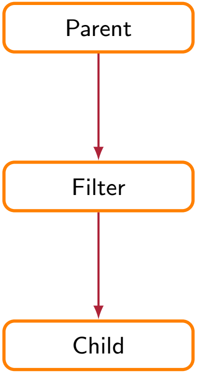
Graph manipulation (3)



Graph manipulation (3)



Graph manipulation (3)



Implicit graph manipulation

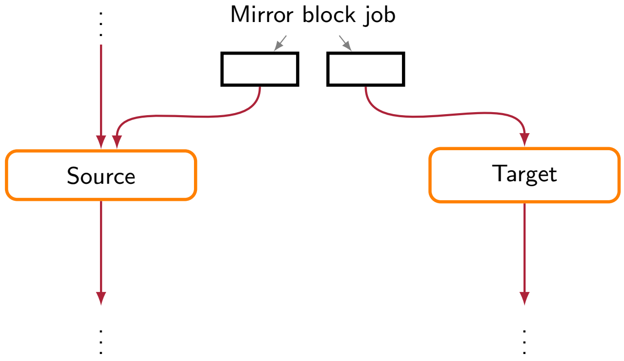
Block jobs on completion:

- e.g. mirror: Replaces source with target
- (commit, stream: Depends.)

Future `persistent` (?) option: Prevent block job from such automatic graph manipulation

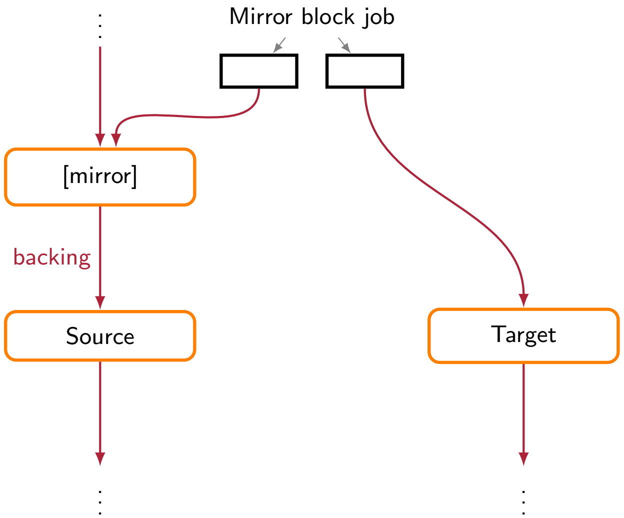
Speaking of block jobs...

...they are going to have filter nodes now:



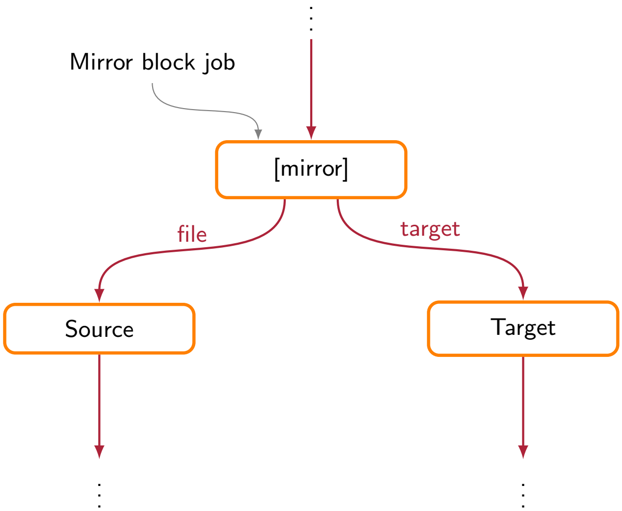
Speaking of block jobs...

(You *can* and *should* name this node)



Speaking of block jobs...

(You *can* and *should* name this node)



Part II

Op blockers

Users of block nodes

We have many different users of block nodes

- Other block nodes (parent nodes)
- Guest devices
- Block jobs
- Monitor commands (e.g. `block_resize`)
- Built-in NBD server
- Live block migration

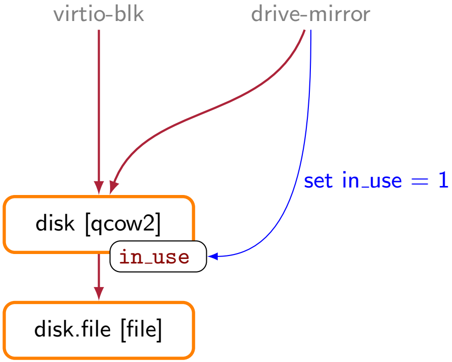
Conflicting users of block nodes

Some of them don't work well together

- Can't resize image during backup job
- Commit job invalidates intermediate nodes
- Guest doesn't expect a changing disk
- ...

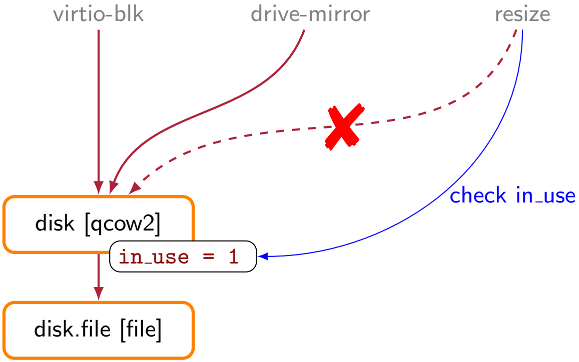
Avoiding conflicts: `bs->in_use`

Easy: Let's just flag devices for exclusive access



Avoiding conflicts: `bs->in_use`

Easy: Let's just flag devices for exclusive access



Avoiding conflicts: `bs->in_use`

Easy: Let's just flag devices for exclusive access

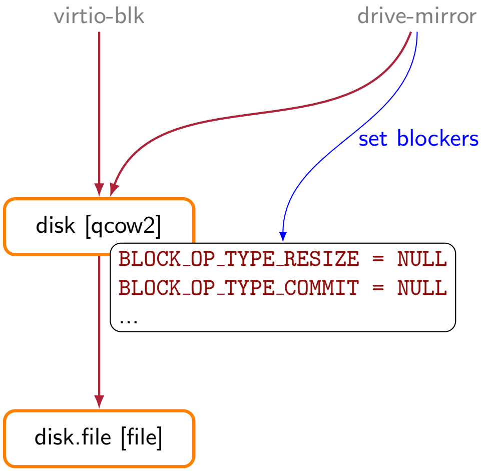
- Set `bs->in_use = true` for exclusive access
- All other users check the flag first
- Except guest devices, they are always allowed
- Very simple solution
- Way too restrictive
- And also a bit too lax

Avoiding conflicts: `BLOCK_OP_TYPE_*`

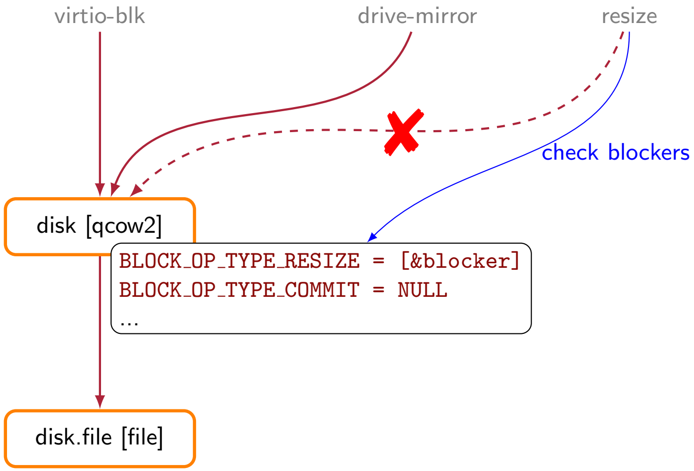
Okay... So we'll distinguish specific operations

- `bdrv_op_block()`
  prevents a specific operation from running
- `bdrv_op_is_blocked()`
  is checked first before the operation
- `BLOCK_OP_TYPE_RESIZE`
  `BLOCK_OP_TYPE_EXTERNAL_SNAPSHOT`
  `BLOCK_OP_TYPE_MIRROR_SOURCE`
  `...`

Avoiding conflicts: BLOCK_OP_TYPE_*



Avoiding conflicts: `BLOCK_OP_TYPE_*`



```
BLOCK_OP_TYPE_RESIZE = [&blocker]
BLOCK_OP_TYPE_COMMIT = NULL
...
```

Avoiding conflicts: `BLOCK_OP_TYPE_*`

Still not quite perfect

- Easy to forget calling the functions
- Need to know all conflicting operations
  - Ideally including future ones
- In practice: Just block everything else
  - That didn't quite achieve the goal...
- Usually only called for root node
  - Not how the block layer works in 2017

Avoiding conflicts: Permissions

Define requirements in terms of low-level operations

- Which operations do I need?
- Which ones may others use while I am active?

Avoiding conflicts: Permissions

Small set of low-level operations

- `CONSISTENT_READ` – read meaningful data
  - Not meaningful: intermediate nodes during commit
- `WRITE` – change data
- `WRITE_UNCHANGED` – invisible (re)writes
  - e.g. streaming, which pulls unchanged data from a backing file to an overlay
- `RESIZE` – resize the image
- `GRAPH_MOD` – something with the graph
  - To be figured out, but people expect we need it

Avoiding conflicts: Permissions

Make it a mandatory core concept

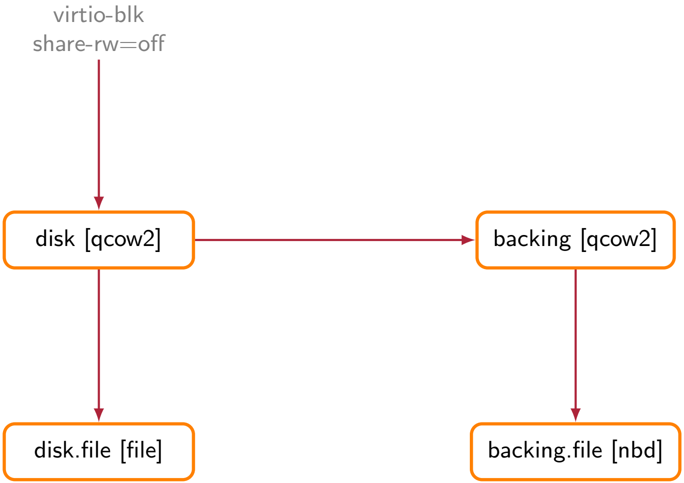
- When attaching to a node...
  - ...required permissions must be specified
  - ...shared permissions must be specified
- If permissions conflict, attaching fails
- Permissions are checked with assert()
  - If you write without write permission, you crash

Avoiding conflicts: Permissions

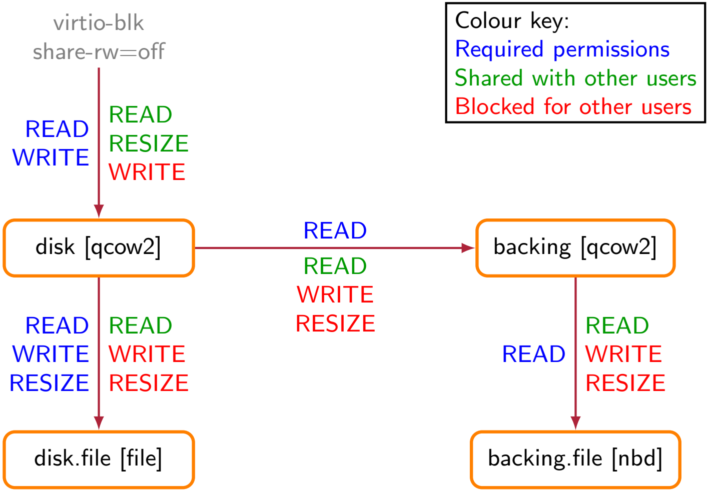
Almost no user configuration needed

- QEMU generally knows the requirements
  - Block drivers need write access if opened read-write
  - Sparse image formats need resize for the file, too
  - Non-raw drivers can't tolerate concurrent writes to the image file
- Exception: Guest devices
  - Whether writes are okay depends on the guest
  - New `share-rw=on|off` property for `-device`

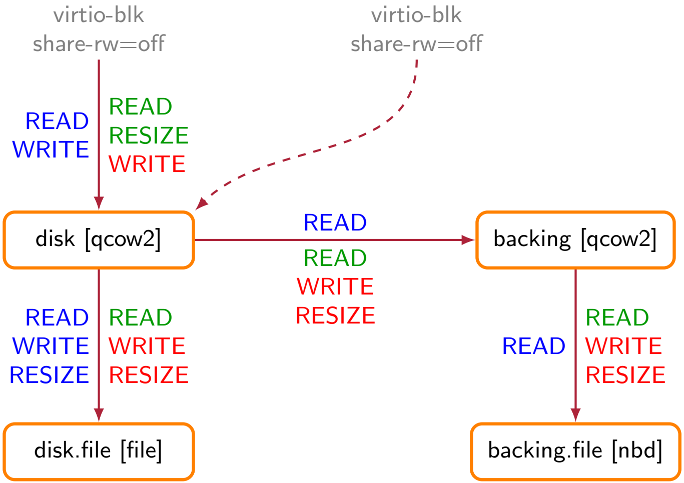
Example: Permission system in practice



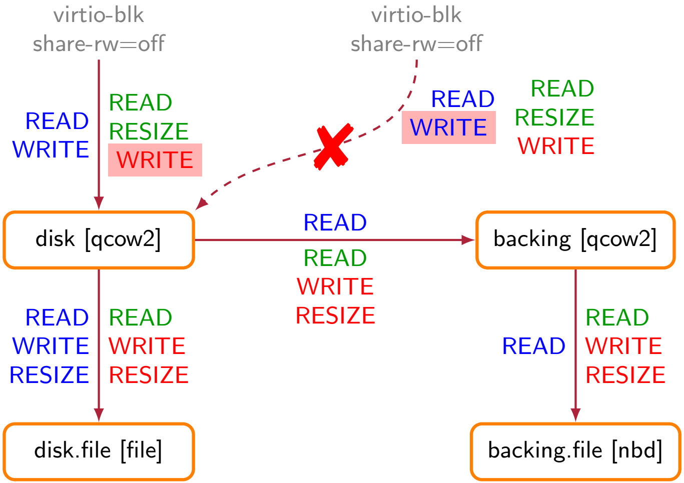
Example: Permission system in practice



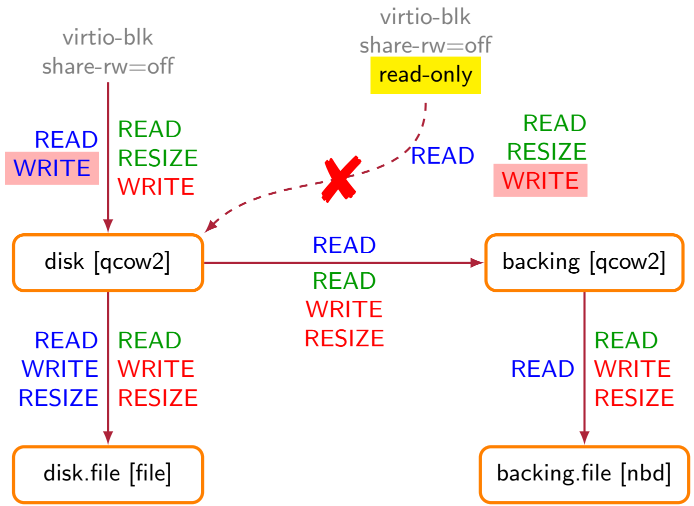
Example: Permission system in practice



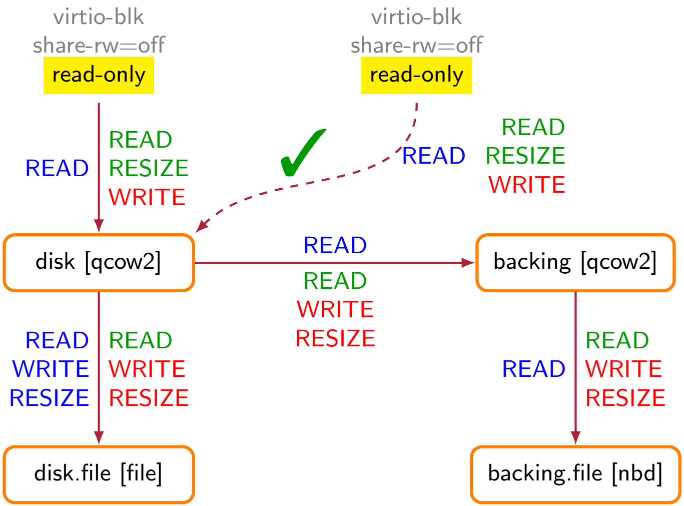
Example: Permission system in practice



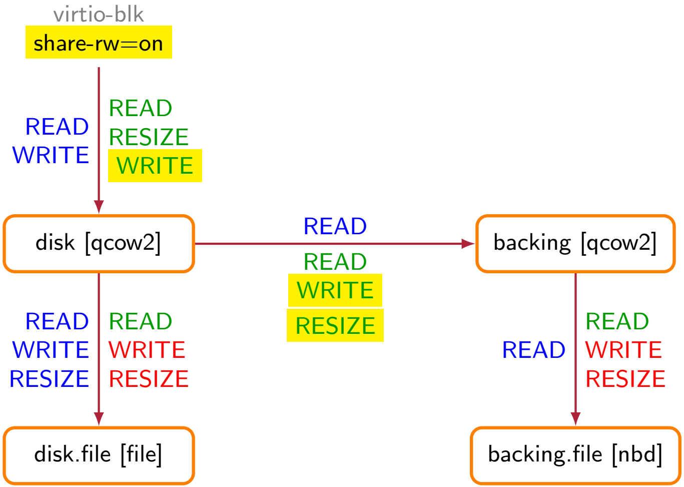
Example: Permission system in practice



Example: Permission system in practice



Example: Permission system in practice



Example: Permission system in practice

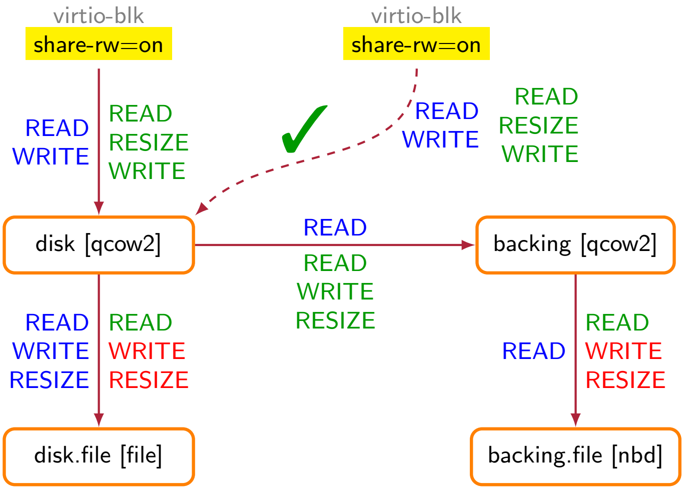


Image locking

Goal: Extend permission system across processes

- Use Open File Description (OFD) locks
- Locks can be taken on byte ranges
- Each permission = pair of shared locks
  - Byte 100-163: Permission used
  - Byte 200-263: Permission can't be shared
- For check: Could exclusive lock be set?

Getting image locking out of the way

What to do if you get locking errors?

- Check that `share-rw` is set correctly
- If so, you're doing something unsafe
- Unsafe because of active writers:
  - Can ignore if read-only and unreliable results are okay
  - QEMU: Override with `force-share=on` in `-drive`/`-blockdev` (applies to whole tree)
  - qemu-img: Override with `-U` or `--force-share`
- Want to do something evil and all else fails?
  - `locking=off` (node-level option for `file`)

Part III

Action items for management tools

Avoid BlockBackend names

- Node and device names are enough for everyone
- Explicitly managing a third type of objects is cumbersome. For you and for QEMU.
- When creating devices, use node names instead
- Replace existing use of BB names in QMP
  - All device commands accept qdev IDs/QOM paths
  - All backend commands accept node names
- Goal: No `id=...` in `-drive` needed
  - And don't use the default IDs, obviously

-blockdev and blockdev-add

- `-drive` and `drive_add` compatibility impedes development. We want to get rid of it sooner rather than later.
- Start using `-blockdev`/`blockdev-add` **now**
  - Preferably even yesterday
- If you got rid of BB names, not too hard

Filter nodes

Legacy config may create filter nodes internally

- Manage filter nodes manually instead
- If you let QEMU create filters automatically...
  - the internal node is unnamed
  - internal nodes may not appear in the right order
  - it makes managing the graph harder for you
- New in 2.11: I/O throttling filter (`throttle`)

Block jobs

- Expect that jobs insert filter nodes in the graph
- Assign names to these filter nodes
  - Option of the QMP command to start a job
- Make use of explicit job deletion
  - ...as soon as QEMU implements it
  - This avoids race conditions

Permission system

- Ideally, just don't use dangerous setups
  Only dangerous setups result in new errors
- Make sure to set `share-rw` correctly
- Avoid `force-share` and `locking=off`
  - Use the monitor of the running VM instead
  - If you must, prefer `force-share` where possible
  - If you think you must, think twice.
    Many people said they need to disable locking.
    Most of them were wrong.

redhat.

Questions?