redhat.

Jobs & Unemployment

In The New QEMU Economy

John Snow
Software Engineer, Red Hat
Block Layer, Virtualization Team
2016-08-25

Acknowledgments

(Those that helped me when I was Job-less)

Thanks to:

- Jeff Cody — Block-jobs Czar
- Markus Armbruster — Resident QAPIbara
- Eric Blake — Full-time carelessness firewall
- Kevin Wolf — Block layer Bad Dude™
- Max Reitz – Assistant Bad Dude™
- Alberto Garcia – Preliminary Work

Overview

(99% accurate 30-minute jobs forecast)

QEMU 2.7 Jobs Report

- What are Jobs?
- Jobs in Today's economy

Job Lifetime & Management

- Workflow
- User Interface & Management
- Events
- Lifetime

Overview

(99% accurate 30-minute jobs forecast)

"Unemployment"

- Shortcomings
- Block specificity
- Lack of parallelism

Jobs Outlook

- Parallelism / Multijobs
- Expanded Jobs Layer
- Subsystems

QEMU 2.7 Jobs Report

(Unemployment is low, but so is worker participation)

What are jobs?

(A question also asked in future dystopian America)

Jobs are long-running QEMU tasks.

- User-visible, persistent objects
- User-manipulable
  - *Pause, resume, cancel, set-speed, etc.*
- *Created via QMP*
- Manipulated/Queried via QMP

What are jobs?

(A question also asked in future dystopian America)

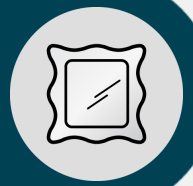
- Inherently Asynchronous
  - Async completion / failure
  - Async notification via QMP events
- (Usually) self-terminating
- Used for:
  - Tasks that will take a long time
  - Tasks of indeterminate or non-finite length
  - Ideal for storage tasks

Jobs in today's economy

(Jobs report: no new jobs added in 2.7 – eek!)

There are four block jobs today:

commit   stream

mirror   backup

(Though some have multiple interfaces and sub-types… we'll get to that.)

Job: block-commit

(I'm fully *commit*ted to these awful jokes.)

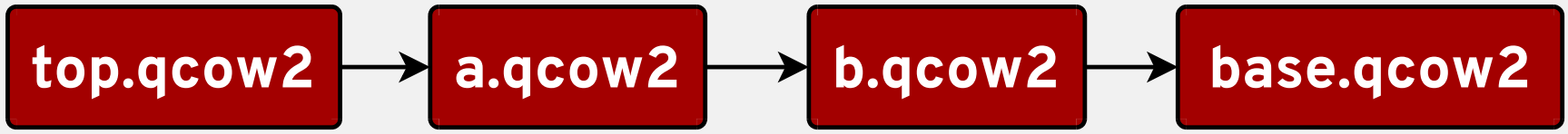
Block commit squashes layers of an image.

- Changes are written ***down*** to the base.
- Asynchronously commits changes into the base
- For more detailed information:
  - Eric Blake @ KVM Forum 2015
    "Backing Chain management in qemu and libvirt"
  - Kashyap Chamarthy @ LinuxCon NA 2016
    "A Practical Look at QEMU's Block Layer Primitives"

Job: block-commit

(I'm fully *commit*ted to these awful jokes.)

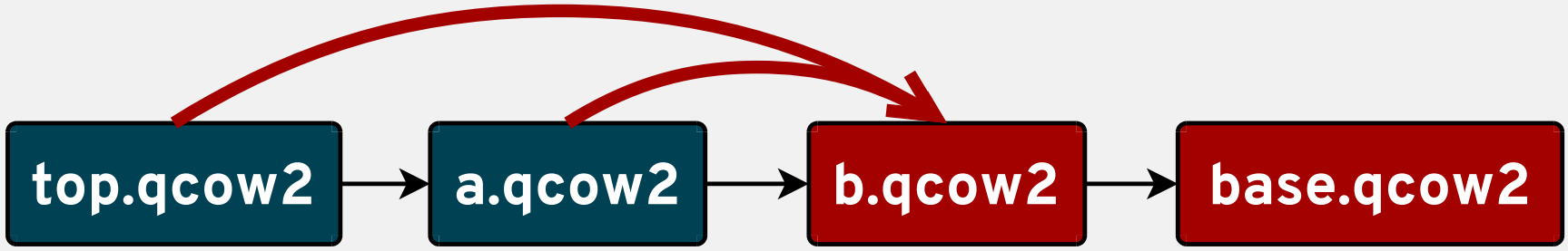
Let's take a sample qcow2 backing chain:



And let's say we want to squash the top three layers into a unified "b.qcow2."

Job: block-commit

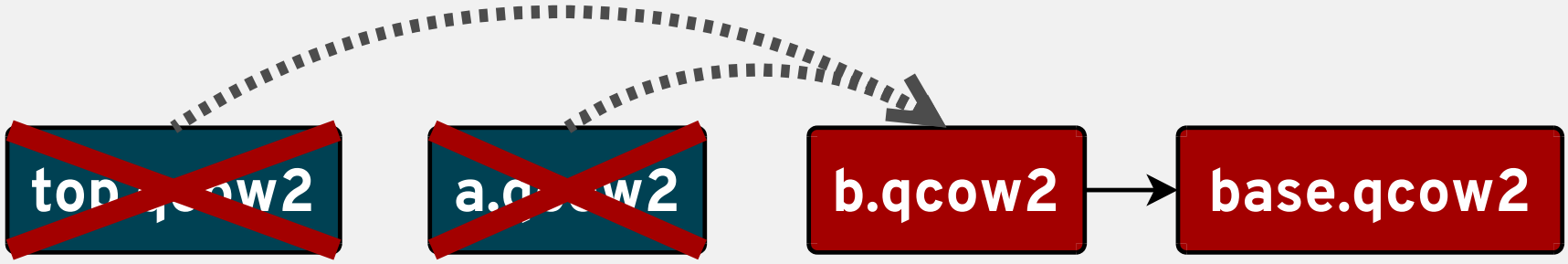
(I'm fully *commit*ted to these awful jokes.)



Via block-commit, we can asynchronously write everything down into b.qcow2.

Job: block-commit

(I'm fully *commit*ted to these awful jokes.)



After data has been merged into b.qcow2, the formerly top layer(s) can be safely removed.

Job: block-stream

(Not a waterfall of tiny cubes)

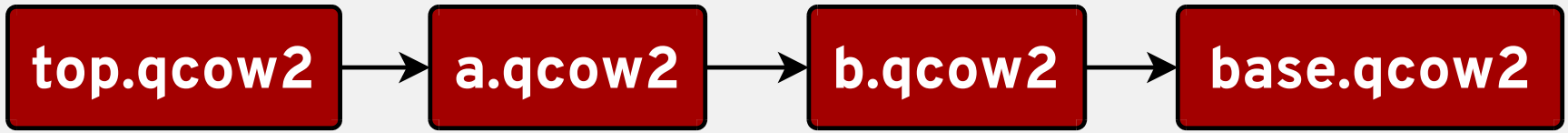
Block stream *also* squashes layers of an image.

- Changes are written ***up*** to the top/active layer.
- Asynchronously pulls changes up to the top.
- For more detailed information, again:
  - Eric Blake @ KVM Forum 2015
    “Backing Chain management in qemu and libvirt”
  - Kashyap Chamarthy @ LinuxCon NA 2016
    “A Practical Look at QEMU's Block Layer Primitives”

Job: block-stream

(Not a waterfall of tiny cubes)

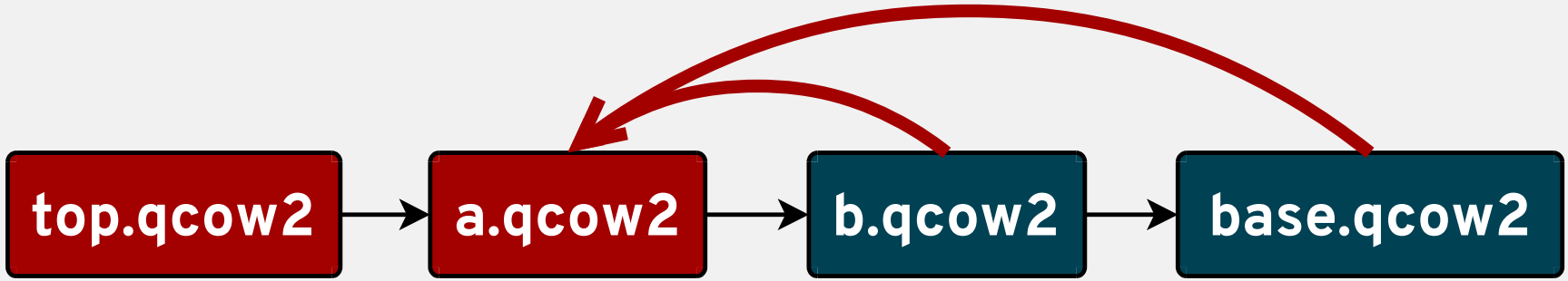
Let's take the same sample qcow2 backing chain:



But this time, let's squash the changes upwards into "a.qcow2."

Job: block-stream

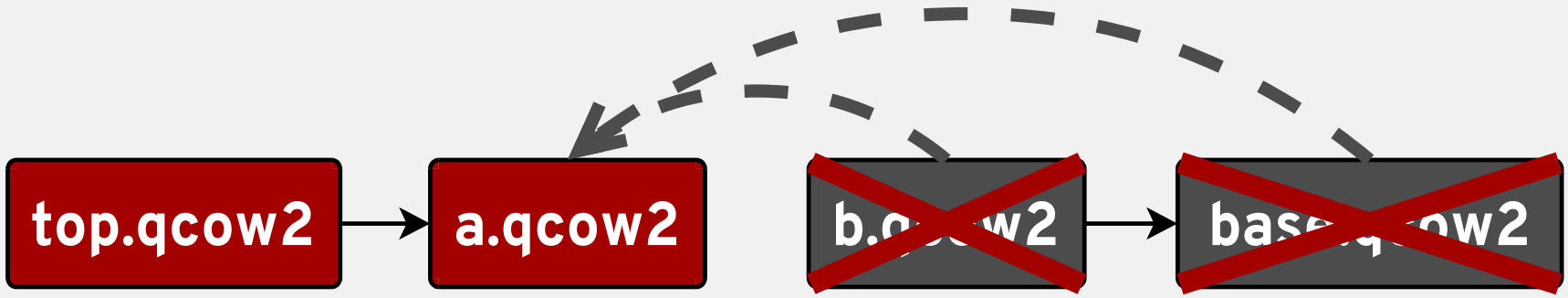
(Not a waterfall of tiny cubes)



Similarly, we asynchronously copy data up into a top layer.

Job: block-stream

(Not a waterfall of tiny cubes)



And just like commit, we can safely remove the old layers.

Job: block-mirror

(When you gaze into the block layer, the block layer gazes back)

Block mirror is at its heart a copy operation.

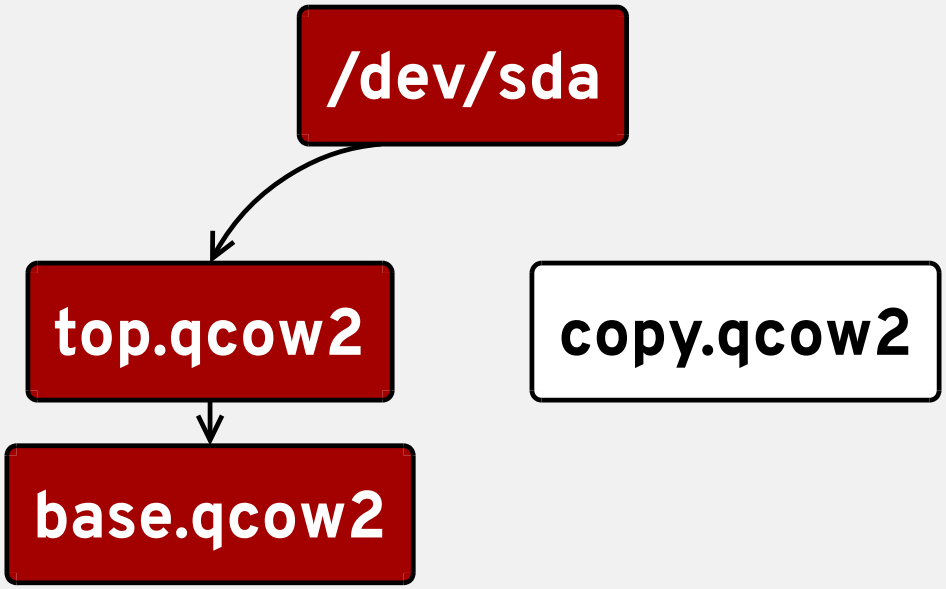
- Multiple sync modes:
  - Full, top, none
- Two-phase copy process:
  - Pre-synchronized, Post-synchronized
- Asynchronously handles backlog and new writes
- Can run indefinitely upon reaching parity
- More Info: Eric's talk (2015), Kashyap / Max (2016)

Job: block-mirror

(When you gaze into the block layer, the block layer gazes back)

Simple sync=full case:

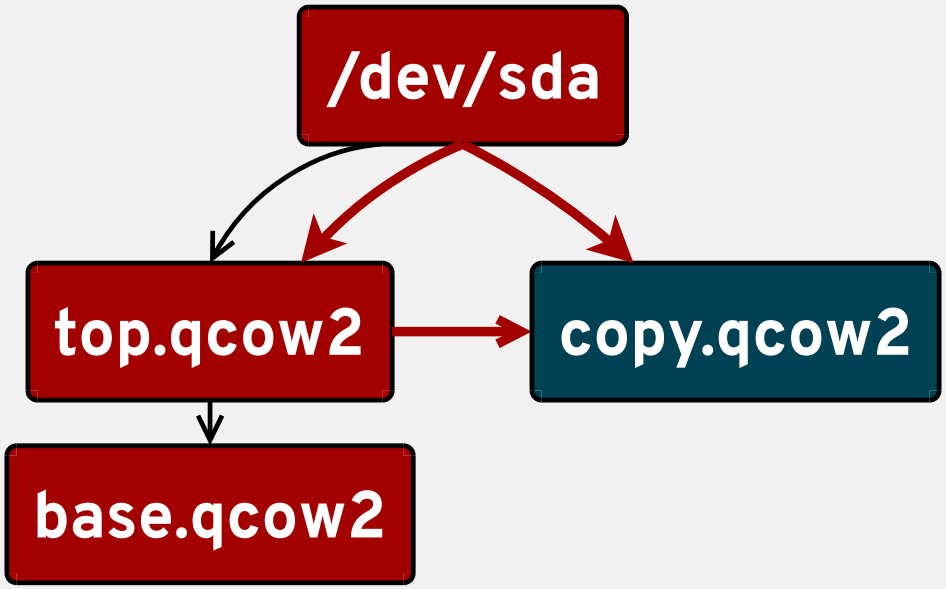
- Empty copy.qcow2 destination



Job: block-mirror

(When you gaze into the block layer, the block layer gazes back)

1) All new writes go to both top and copy.
2) All existing data from top and base get mirrored to copy.

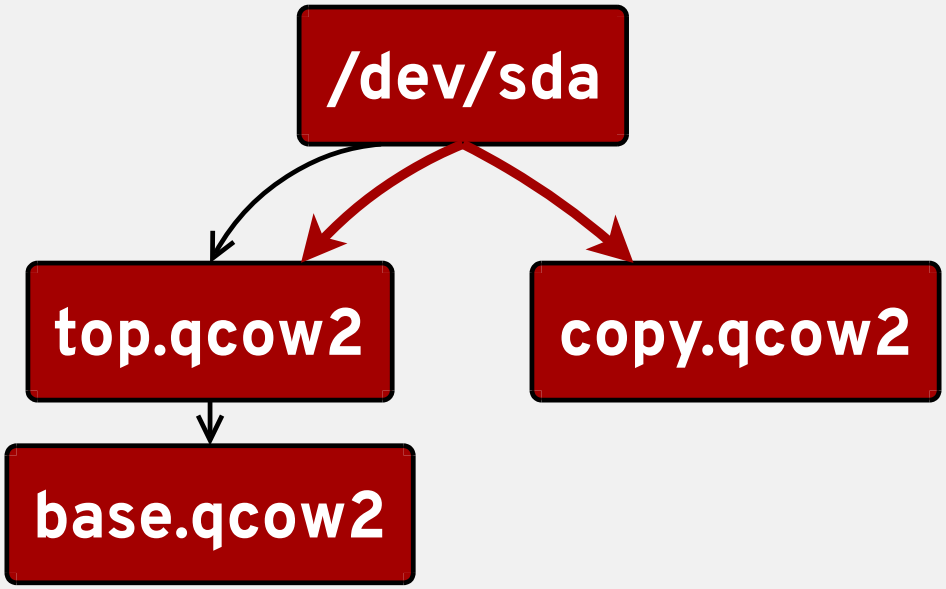


Job: block-mirror

(When you gaze into the block layer, the block layer gazes back)

Parity reached:

- New writes are mirrored indefinitely

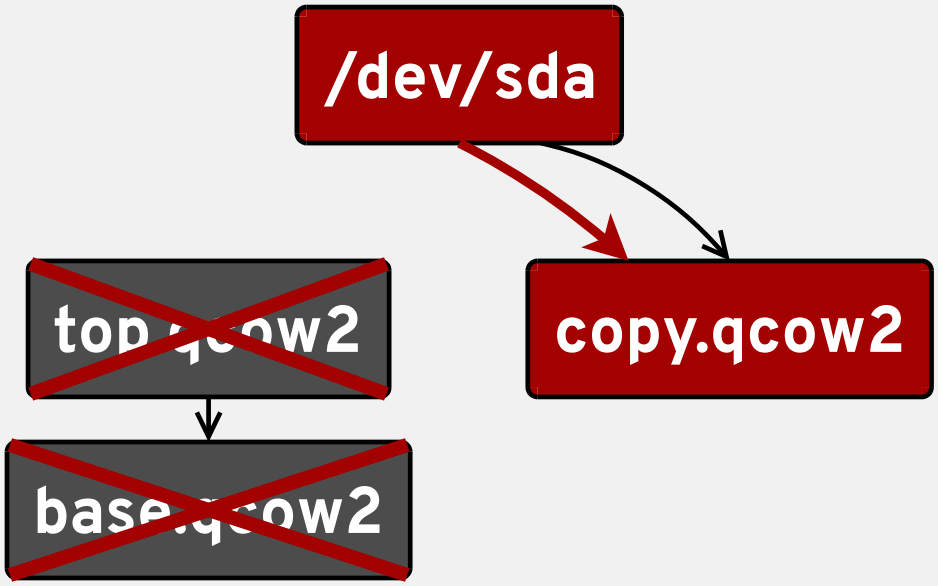


Job: block-mirror

(When you gaze into the block layer, the block layer gazes back)

Job told to finish:

- QEMU pivots to copy.qcow2 exclusively



Job: block-backup

(Backup plan: *rhombus-devel@nongnu.org* ?)

Backup is similar to mirror, it is a copy operation.

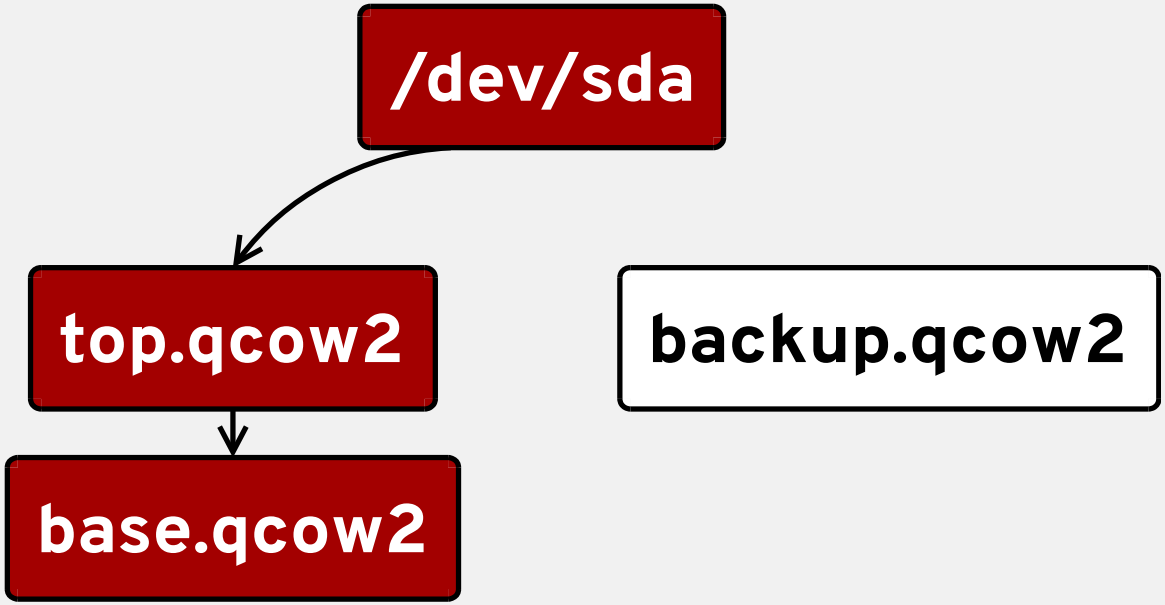
- Uses the same sync modes as mirror
  - (full, top, none)
- Does not include a sync phase
- Does not 'pivot' to the backup.
- Point-in-time: At job start
- Includes a bonus backup mode: Incremental
  - See ***my*** KVM Forum 2015 talk for more details!

Job: block-backup

(Backup plan: *rhombus-devel@nongnu.org*?)

Simple case:

- Back up drive ‘sda’ to backup.qcow2

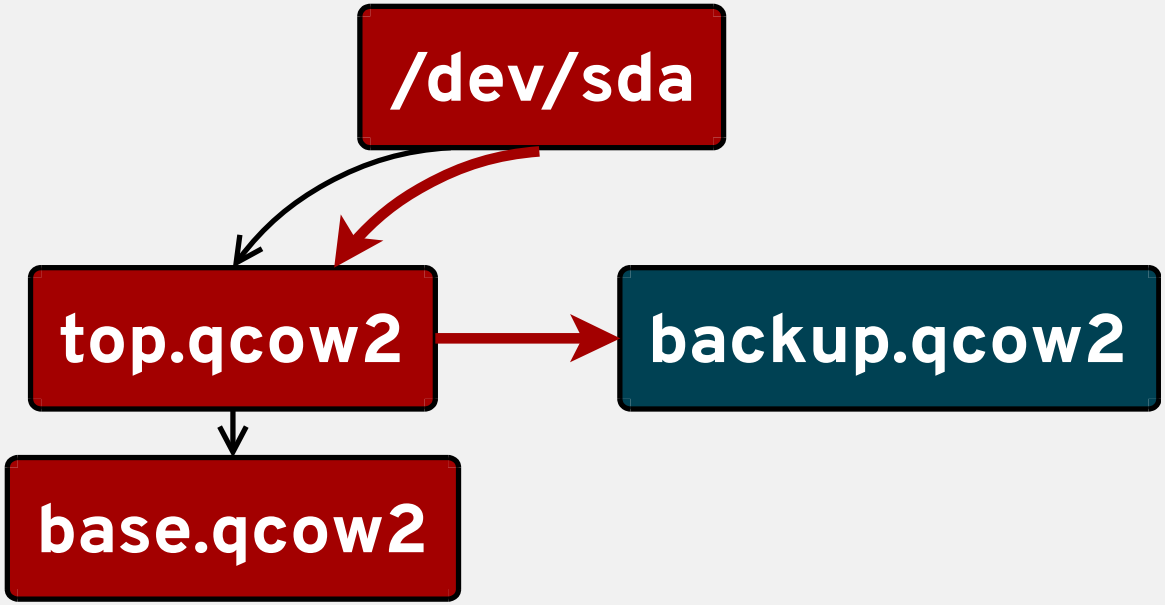


Job: block-backup

(Backup plan: *rhombus-devel@nongnu.org*?)

Unlike block-mirror…

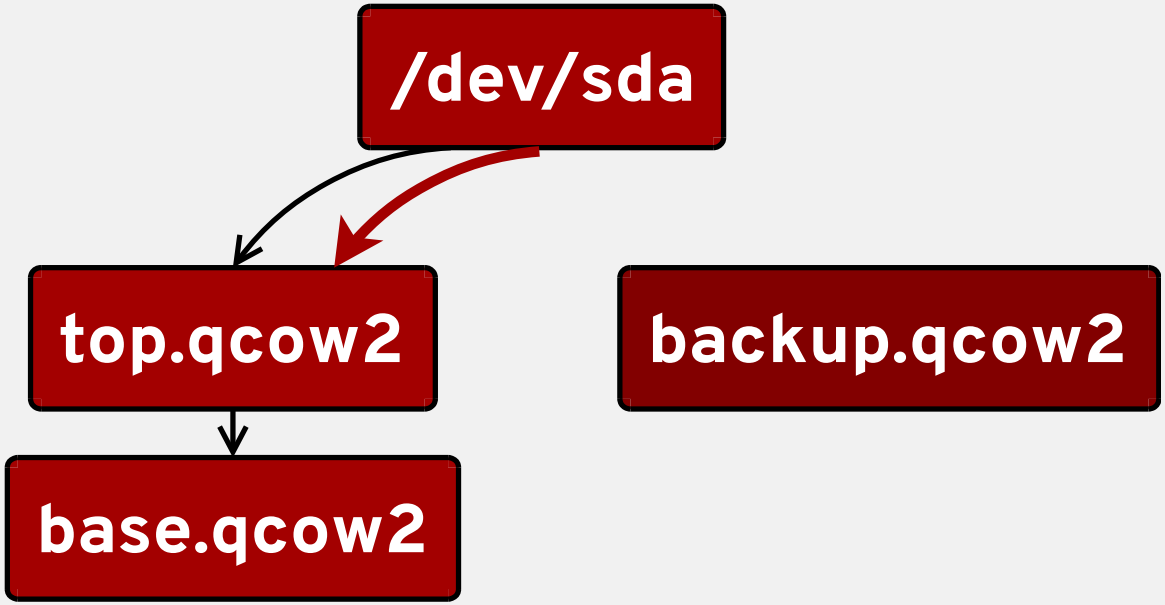
- Writes don't get mirrored to the backup.



Job: block-backup

(Backup plan: *rhombus-devel@nongnu.org*?)

When finished, there is no sync phase or pivot.



JOB LIFETIME & MANAGEMENT

(AKA: Retirement Planning?)

Jobs: Management

(MBA specializing in block-jobs)

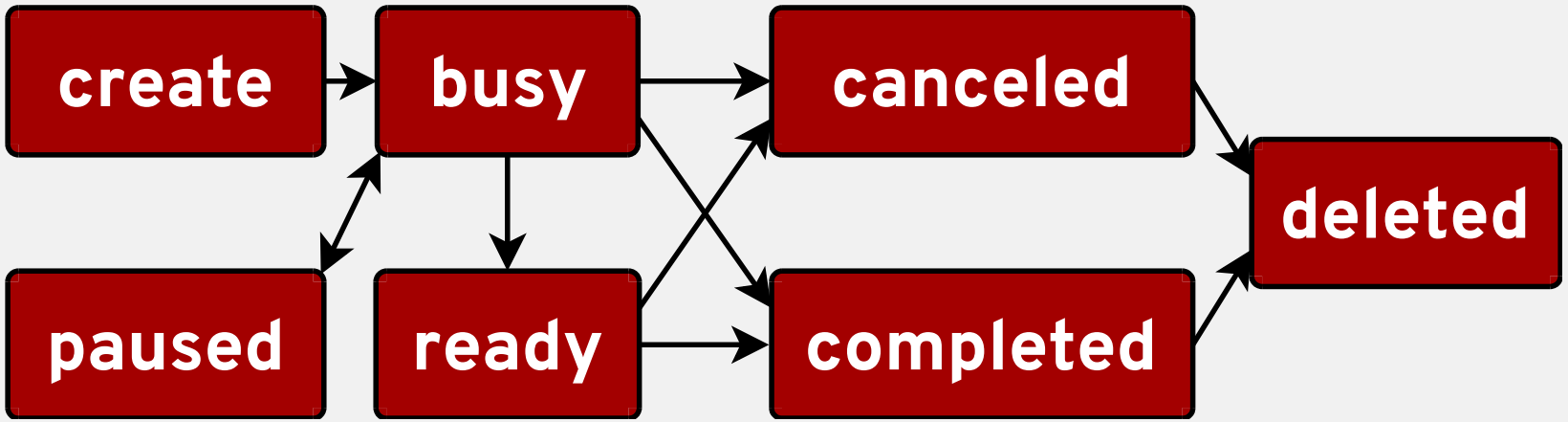
Jobs are *managed* entirely via QMP.

- Creation
- Verification and Query
- Pause, Cancel, or Resume
- Complete a ‘ready’ job
  - (i.e. block-mirror)
- Receive asynchronous event notifications
  - docs/qmp-events.txt

Jobs Workflow

(Charts to read while at the unemployment office)

Now that we know what they do, let's see how/when:



Jobs: Creation

(The least political talk of 'job creators' ever given)

- There is no central block-job-create command
  - Each job is created by its own 'front-end'
- Jobs are automatically started after create
- Historically, Job "ID" is that of the related device
  - i.e. no explicit ID given

Jobs: Creation (Example)

(The least political talk of ‘job creators’ ever given)

- Jobs are created via QMP.

```
{ "execute": "drive-backup",
  "arguments": {
    "device": "sda",
    "target": "sda.qcow2",
    "format": "qcow2",
    "sync": "full",
    "mode": "existing"
} }
```

```
{ “return”: { } }
```

Jobs: Events

(Nothing at all like a career fair)

Jobs report status via QMP events.

- BLOCK_JOB_CANCELLED
  - (Yes, with two Ls, says the American)
- BLOCK_JOB_COMPLETED
  - Not indicative of actual success
- BLOCK_JOB_ERROR
- BLOCK_JOB_READY

Jobs: Querying

(Help Wanted: Seek Within)

```
{ "execute": "query-block-jobs",
  "arguments": {} }
```

```
{ "return": [ {
    "busy": true,
    "type": "backup",
    "len": 68719476736,
    "paused": false,
    "ready": false,
    "io-status": "ok",
    "offset": 26104299520,
    "device": "sda",
    "speed": 0
} ] }
```

Jobs: Pausing

(I guess this would be a leave of absence?)

```
{ "execute": "block-job-pause",
  "arguments": { "device": "sda" } }
```

```
{ "return": {} }
```

```
{ "return": [ {
    "busy": false,
    "type": "backup",
    "len": 68719476736,
    "paused": true,
    "ready": false,
    "io-status": "ok",
    "offset": 26104299520,
    "device": "sda",
    "speed": 0
} ] }
```

Jobs: Resuming

(select-all → mark as read)

```
{ "execute": "block-job-resume",
  "arguments": { "device": "sda" } }
```

```
{ "return": {} }
```

```
{ "return": [ {
    "busy": true,
    "type": "backup",
    "len": 68719476736,
    "paused": false,
    "ready": false,
    "io-status": "ok",
    "offset": 26104299520,
    "device": "sda",
    "speed": 0
} ] }
```

Jobs: Completion

(Nothing like a hard day's work completed)

Jobs can either **complete successfully**, error out, or get canceled.

```
{ 'timestamp': {
    'seconds': 1471637374,
    'microseconds': 508344},
  'data': {
    'device': 'sda',
    'type': 'backup',
    'speed': 0,
    'len': 68719476736,
    'offset': 68719476736},
  'event': 'BLOCK_JOB_COMPLETED'
}
```

Jobs: Completion

(Nothing like a hard day's work completed)

Jobs can either complete successfully, **error out**, or get canceled.

```
{ "timestamp": {
    "seconds": 1471637374,
    "microseconds": 683015 },
  "data": {
    "device": "sda",
    "action": "report",
    "operation": "read" },
  "event": "BLOCK_JOB_ERROR"
}
```

```
{ "timestamp": {
    "seconds": 1471637374,
    "microseconds": 683315 },
  "data": {
    "speed": 0,
    "offset": 0,
    "len": 68719476736,
    "error": "Input/output error",
    "device": "sda",
    "type": "backup" },
  "event": "BLOCK_JOB_COMPLETED"
}
```

Jobs: Completion

(Nothing like a hard day's work completed)

Jobs can either complete successfully, error out, or get **canceled**.

```
{ "timestamp": {
    "seconds": 1447193702,
    "microseconds": 640163 },
  "data": {
    "device": "sda",
    "type": "backup",
    "speed": 0,
    "len": 67108864,
    "offset": 16777216 },
  "event": "BLOCK_JOB_CANCELLED"
}
```

Jobs: Manual Completion

(Not an advert for a technical copywriter)

Jobs that run indefinitely need to be told to complete (or cancel) when **ready**.

```
{ "timestamp": {
    "seconds": 1471647044,
    "microseconds": 444237 },
  "data": {
    "device": "sda",
    "type": "mirror",
    "speed": 0,
    "len": 67108864,
    "offset": 67108864 },
  "event": "BLOCK_JOB_READY"
}
```

```
{ "execute": "block-job-
complete",
  "arguments": {
    "device": "sda"
  }
}
```

```
{ "return": {} }
```

Unemployment

(Shortcomings with Block Jobs)

Shortcomings

(Okay, that was all fine and dandy, but… so?)

Jobs were implemented as block-specific primitives.

- All QMP commands are block-related:
  - Query-block-jobs, block-job-pause, etc.
- Implemented in a block-centric way
  - Code tied fairly closely to block layer
- Historically do not have unique IDs
  - Tied to the ‘device’ instead
  - Increasingly outdated paradigm

Identification

(Papers, Please)

Block Jobs are currently* managed via device ID.

- Some jobs interact with more than one node/device
- Some jobs ‘pivot’ on their focal device
- Jobs currently only open blockers on one node
  - May interact with/affect more
- Only one job allowed per device
  - This is unsatisfactory for multiple read operations
  - We’d like true multiple (block) job support

Multijobs

(A very atypical pinball machine bonus)

- Jobs take more locks than they need.
- We want increased parallelism
  - Nothing prohibits us from multiple RO jobs
- New Op Blockers will help in part
  - More fine-grained
- But we need a re-factoring of the QAPI, too

The Power of Co-routines

(Hey, what are you folks doing tomorrow?)

Jobs are a powerful user interface to coroutines.

- Useful interface and user abstraction for tasks
- This interface currently limited to block layer…
  - But it could be separated and used more broadly!
- If Jobs need reworking for multi-jobs anyway…
  - …Let's bring this power to all of QEMU.
- Jeff Cody's talk tomorrow @ 11:15AM EDT
  - QEMU Coroutines, Exposed"

Jobs Forecast

(I’m sorry, this pun is getting really strained)

Jobs for everyone?

(I'm not running for office, I promise)

Since we need to make a new API for multijobs…

- Let's bring co-routines and jobs to all of QEMU.
- Better abstraction for tasks
  - More generic
  - Simpler to manage, query
- Brings a powerful interface to QEMU
- Already well understood

Block Jobs → Jobs

(I think I have writer's block)

NOW		2.0
block-job-cancel		job-cancel
block-job-pause		job-pause
block-job-resume	→	job-resume
block-job-complete		job-complete
query-block-jobs		query-jobs
block-job-set-speed		job-set-option

Block Jobs → Jobs

(I think I have writer's block)

NOW		2.0
BLOCK_JOB_CANCELLED		JOB_CANCELLED
BLOCK_JOB_COMPLETED		JOB_COMPLETED
BLOCK_JOB_ERROR	→	JOB_ERROR
BLOCK_JOB_READY		JOB_READY
-		JOB_STARTED ?

Block Jobs → Block Jobs

(Obligatory Compatibility Slide)

- Legacy interface will remain
  - Can be used by e.g. older libvirt
  - Returns errors after any new API usage
  - As strict over 2.7-era usage (no multijobs)
- Block Jobs implemented as ‘subclass’ of Jobs
- Provides example for future subsystems...
- Existing job-specific creation interfaces remain

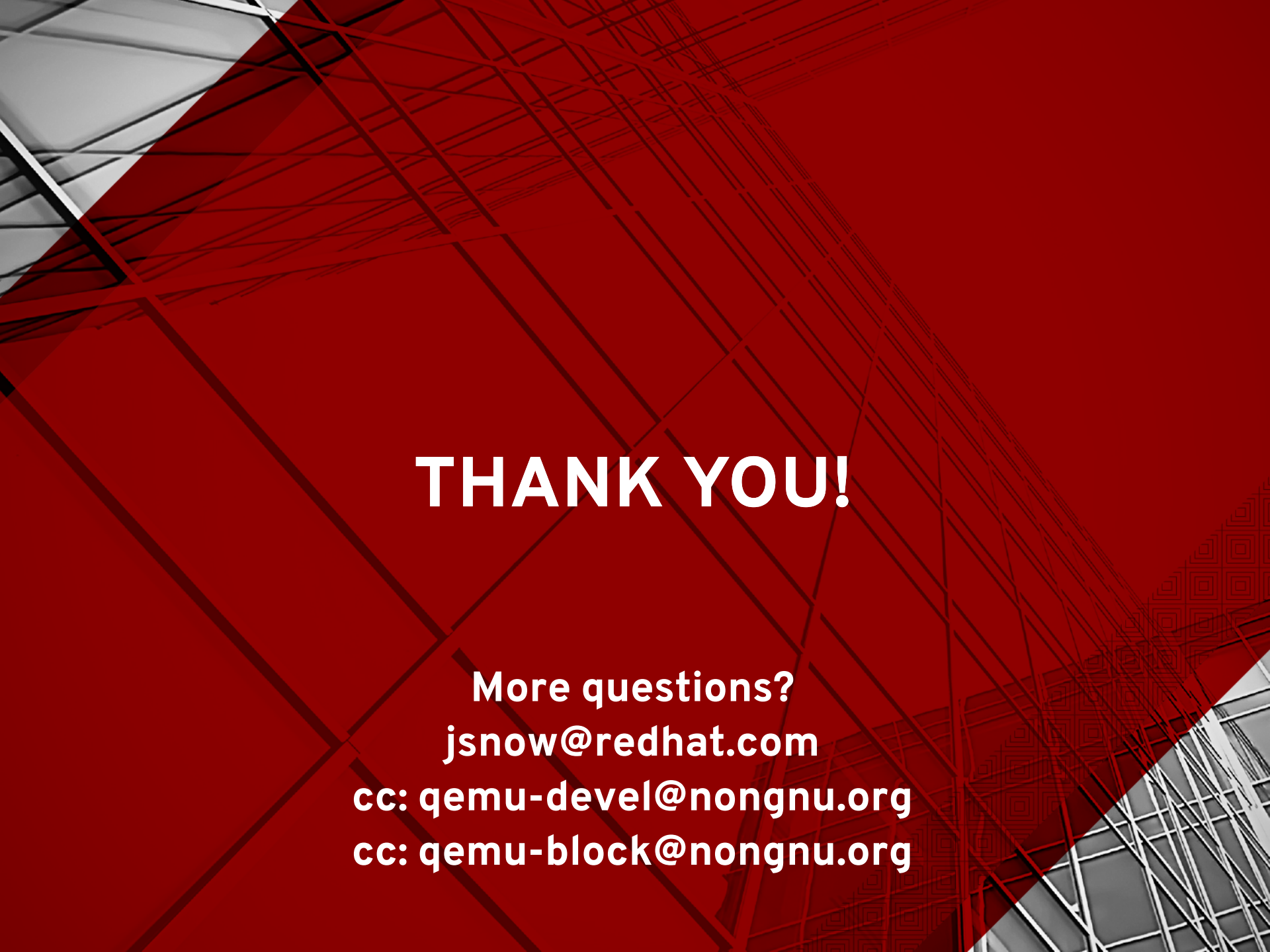
Jobs: Subsystems

(Not transit authority jobs)

- Block-jobs are now a ‘subsystem’
  - Capacity to expand query/set-options
    - e.g. set-speed if not applicable to general case
- Other subsystems may wish to utilize coroutines
  - At the risk of getting volunteered, Migration?
  - Colo? Debugging? Statistics? Fault Tolerance?


Requests?


Questions?


THANK YOU!
More questions?
jsnow@redhat.com
cc: qemu-devel@nongnu.org
cc: qemu-block@nongnu.org