redhat. Odin

Incremental Backups

(*Good things come in small packages*!)

John Snow (*yes, I know*)
Software Engineer, Red Hat

Vladimir Sementsov-Ogievskiy
Software Developer, Odin

2015-08-20

Acknowledgments

(Because computers are awful and I need help sometimes)

No feature is an island, so I'd like to acknowledge:

- Jagane Sundar
  - Initial feature proposal and prior work (2011)
- Fam Zheng
  - Initial drafts for current version (2014-2015)
- Stefan Hajnoczi & Max Reitz
  - Reviews and patience

Overview

(Things I hope not to stammer through)

Prologue

- Problem Statement
- Approach
- Design Goals

Act I: Building Blocks

- Block Dirty Bitmaps
- QMP interface and usage
- QMP transactions

Overview

(Things I hope not to stammer through)

Act II: Life-cycle

- Incremental backup life-cycle
- Examples

Act III: Advanced Features

- Migration
- Persistence

Act IV: Denouement

- Project Status, Questions and Answers

PROLOGUE

(In which our heroes come to know the enemy)

The Problem

(I just wandered into this talk, what's it about?)


Monday
128GiB


Tuesday
128GiB


Wednesday
128GiB


Thursday
128GiB

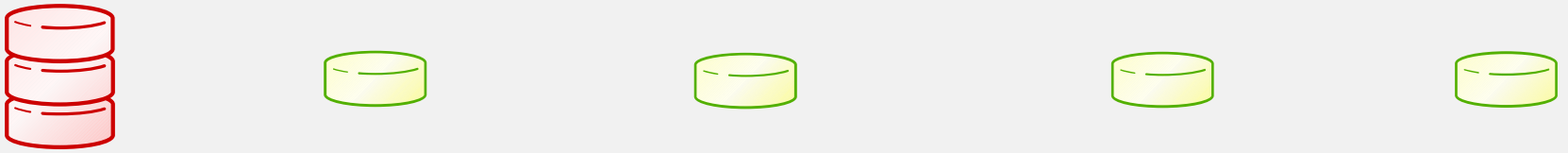

Friday
128GiB

Gross.

- Abysmal storage efficiency
- Clunky, slow
- But admittedly simple and convenient

The Problem

(I just wandered into this talk, what's it about?)



Monday	Tuesday	Wednesday	Thursday	Friday
128GiB	2GiB	1.5GiB	2.25GiB	1GiB

Much Better!

- Efficient: only copies modified data
- Fast!
- More complicated...?

Approach

(Where did we come from; where did we go)

Incremental Live Backups have a storied lineage.

- Jagane Sundar's LiveBackup (2011)
  - Separate CLI tools
  - Entirely new network protocol
  - Ran as an independent thread
  - Utilized temporary snapshots for atomicity
  - Implemented with in-memory dirty block bitmaps
  - Was ultimately not merged

Approach

(Where did we come from; where did we go)

Fam Zheng's Incremental Backup (2014)

- Also dirty sector bitmap based
  - Uses existing HBitmap/BdrvDirtyBitmap primitives
- No new external tooling or protocols
- Managed via QMP
- Implemented simply as a new backup mode
- Can be used with any image format
- Maximizes compatibility with existing backup tools

Design Goals

(What do we want?)

- Reuse existing primitives as much as possible
  - Key structure: 'block driver dirty bitmap'
    - Already tracks dirty sectors
    - Used for drive mirroring, block migration
    - Configurable granularity
    - Many bitmaps can be used per-drive

Design Goals

(What do we want? Efficient Backups!)

- Reuse existing primitives
  - Key interface: drive-backup
    - Implemented via well-known QMP protocol
    - Used to create e.g. full backups
    - Already capable of point-in-time live backups
    - Can already export data via NBD
    - We merely add a new sync=incremental mode
      - ...And a bitmap=<name> argument.

Design Goals

(When do we want it?)

- Coherency
  - Multi-drive point-in-time backup accuracy
  - Utilize existing QMP transaction feature
- Persistence
  - Bitmaps must survive shutdowns and reboots
  - Must not depend on drive data format
  - Nor on the backup target format

Design Goals

(When do we want it? By 2.5 hopefully!)

- Migration-safe
  - Migrating must not reset or lose bitmap data
- Error Handling
  - Bitmap data must not be lost on backup failure
  - Starting a new full backup is not sufficiently robust
- Integrity
  - We *must* be able to detect desync between persistence data and block data

Why not use snapshots?

(Saving you time during the Q&A)

"Both offer point-in-time views of data, why not use the existing mechanism?"

- No need to parse format-specific snapshots on disk
- We can use *any format*
- Incremental backups are *inert* and do not grow
  - No IO required to delete incrementals
- We can utilize existing backup frameworks
- Access to QEMU's NBD server

ACT I: BUILDING BLOCKS

(In which our heroes prepare for battle)

Block Dirty Bitmaps

(Nothing to do with your image search settings)

Before showcasing incrementals, some background:

- BdrvDirtyBitmap is the existing block layer structure used to track writes
  - Already used for drive-mirror, live block migration
  - Implemented using Hierarchical bitmap
  - Any number can be attached to a drive
    - Allows for multiple independent backup regimes

Block Dirty Bitmaps - Naming

(A bitmap by any other name would smell as sweet...?)

- Block dirty bitmaps may have names:
  - Existing internal usages are anonymous
  - The name is unique to the drive
  - Bitmaps on different drives can have the same name
  - The (node, name) pair is the bitmap ID
    - Used to issue bitmap management commands

Block Dirty Bitmaps - Granularity

(Backups from *French Press* to *Turkish*)

- Block dirty bitmaps have granularities:
  - Small granularity – smaller backups*
    - Uses more memory
      - 1 TiB w/ g=32KiB → 4MiB
      - 1 TiB w/ g=128KiB → 1MiB
- Default: 64KiB**
  - Attempts to match cluster size
  - 64KiB clusters (default) for qcow2

Granularities – In Detail

(Tuned like the finest $4 ukulele)

- Bitmaps track writes ***per-sector***
  - Configure granularity in ***bytes***
  - 64K → 128 sectors (512 bytes/sector)
- The backup engine itself copies out per-cluster
  - Currently: non-configurable, 64K clusters
- The file format also has a cluster size
  - qcow2 defaults to 64K.
- Conclusion: 64K is probably best (for now)

Block Dirty Bitmaps - Management

(Bitmap wrangling 101)

We need to manage these bitmaps to make backups.

- Managed via QMP
  - Good news if you're a computer!
- Four commands:
  - `block-dirty-bitmap-add`
  - `block-dirty-bitmap-remove`
  - `block-dirty-bitmap-clear`
  - `query-block`

Block Dirty Bitmaps - Creation

(Let there be... bits!)

- Bitmaps can be created at any time, on any node
- Bitmaps begin recording writes immediately
- Granularity is optional

```
{ "execute": "block-dirty-bitmap-add",
  "arguments": {
    "node": "drive0",
    "name": "bitmap0",
    "granularity": 131072
  }
}
```

Block Dirty Bitmaps - Deletion

(For days when *less* is *more*)

- Can only be deleted when not in use
- Bitmaps are addressed by their (node, name) pair
- Has no effect on backups already made
- Has no effect on other bitmaps or nodes

```
{ "execute": "block-dirty-bitmap-remove",
  "arguments": {
    "node": "drive0",
    "name": "bitmap0"
  }
}
```

Block Dirty Bitmaps - Resetting

(Sometimes we just want a second chance)

- Bitmaps can be cleared of all data
- Primarily for convenience
- Begins recording new writes immediately, like add

```
{ "execute": "block-dirty-bitmap-clear",
  "arguments": {
    "node": "drive0",
    "name": "bitmap0"
  }
}
```

Block Dirty Bitmaps - Querying

(Who are you? Who who, who who?)

Bitmap data can be retrieved via block-query.

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

{"return": [{ …
  “device”: “drive0”,
  "dirty-bitmaps": [{
     "status": "active",
     "count": 296704,
     "name": "bitmap0",
     "granularity": 65536 }]
… }]}
```

Block Dirty Bitmaps - Querying

(Who are you? Who who, who who?)

Bitmap data can be queried via block-query.

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

{"return": [{ …
  “device”: “drive0”,
  "dirty-bitmaps": [{
     "status": "active",        (or “frozen”!)
     "count": 296704,
     "name": "bitmap0",
     "granularity": 65536 }]
… }]}
```

Block Dirty Bitmaps - Querying

(Who are you? Who who, who who?)

Bitmap data can be queried via block-query.

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

{"return": [{ …
  “device”: “drive0”,
  "dirty-bitmaps": [{
     "status": "active",
     "count": 296704,          (sectors!)
     "name": "bitmap0",
     "granularity": 65536 }]   (2318 clusters)
… }]}
```

Building Cognitive Dissonance

(Problem Statement 2: Electric Boogaloo)

- QMP commands are not particularly useful alone
  - They are not atomic
  - Only “safe” when VM is offline
  - No cross-drive coherence guarantee

Incremental Transactions

(Dissonance abated!)

- Bitmap management transactions allow us to—
  - Create full backups alongside a bitmap reset
  - Create a full backup alongside a new bitmap
  - Reset bitmaps across multiple drives
  - Issue a number of incremental backups across multiple drives

Incremental Transactions

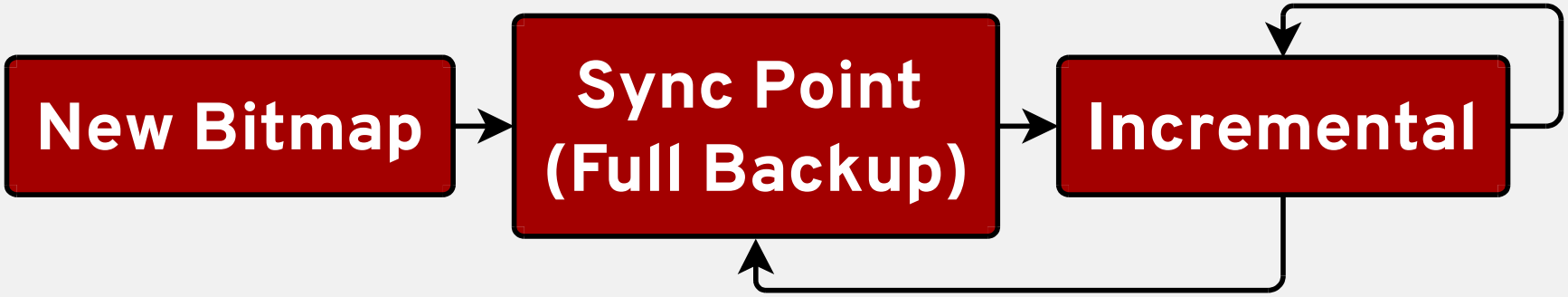
(Dissonance abated!)

- Supported transaction actions:
  - `type:block-dirty-bitmap-add`
  - `type:block-dirty-bitmap-clear`
- No transaction needed for remove
- Works in conjunction with `type:drive-backup`
  - For incrementals (multi-drive coherency)
  - For full backups
    - new incremental chains / sync points

ACT II: LIFE CYCLE

(In which our heroes save time and money)

Incrementals – Life Cycle



1) Create a new backup chain, or
2) Synchronize an existing backup chain
3) Create the first incremental backup
4) Create subsequent incremental backups

Life Cycle – New Chain

(There and backup again)

Example 1: Start a new backup chain atomically

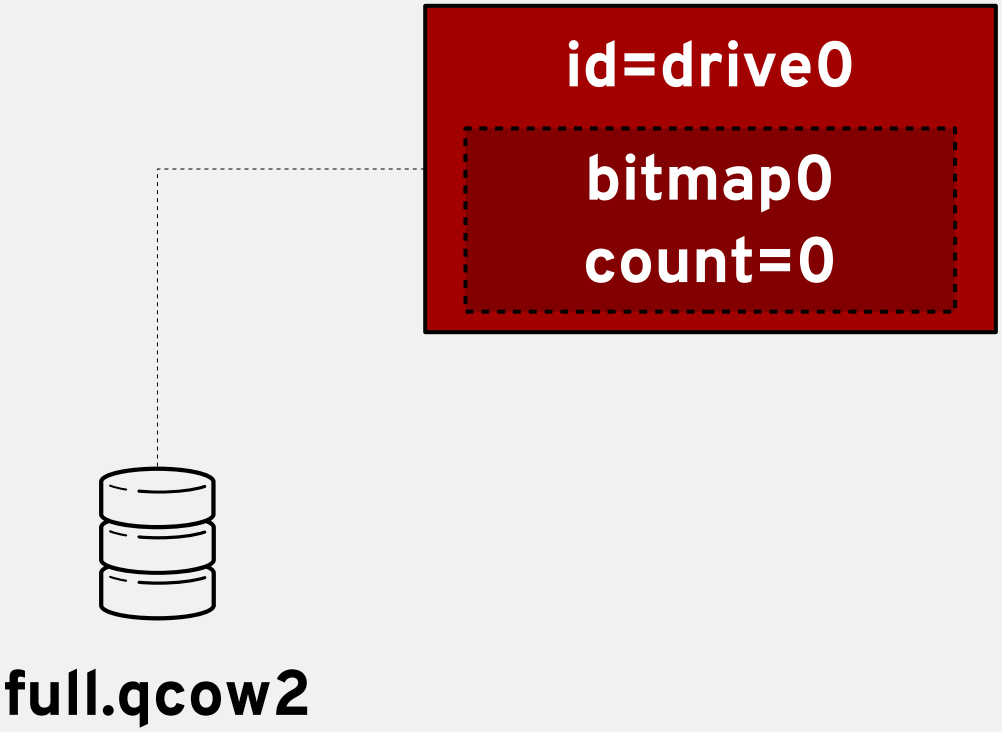
```
{ "execute": "transaction",
  "arguments": {
    "actions": [
      {"type": "block-dirty-bitmap-add",
       "data": {"node": "drive0", "name": "bitmap0"} },
      {"type": "drive-backup",
       "data": {"device": "drive0",
                "target": "/path/to/full.qcow2",
                "sync": "full", "format": "qcow2"} }
    ]
  }
}
```

Life Cycle – New Chain

(There and backup again)

Life Cycle – New Chain

(There and backup again)



Life Cycle – New Sync Point

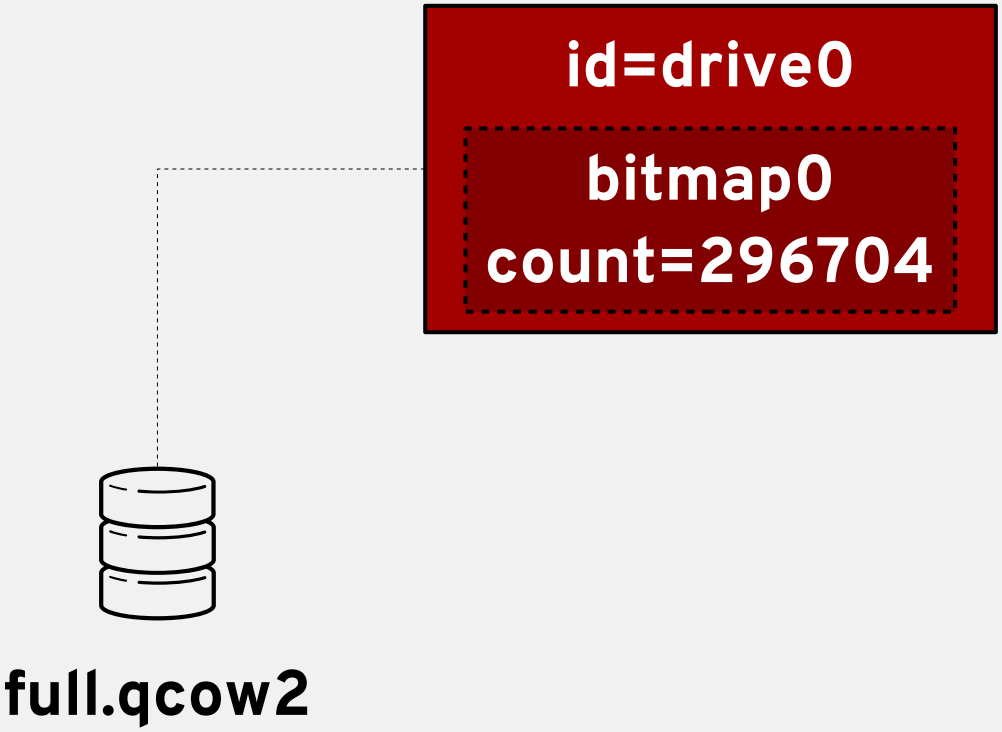
(Sunday night maintenance blues)

Example 2: Take an existing bitmap and create a new full backup as a synchronization point.

```
{ "execute": "transaction",
  "arguments": {
    "actions": [
      {"type": "block-dirty-bitmap-clear",
       "data": {"node": "drive0", "name": "bitmap0"} },
      {"type": "drive-backup",
       "data": {"device": "drive0",
                "target": "/path/to/new_full_backup.qcow2",
                "sync": "full", "format": "qcow2"} }
    ]
  }
}
```

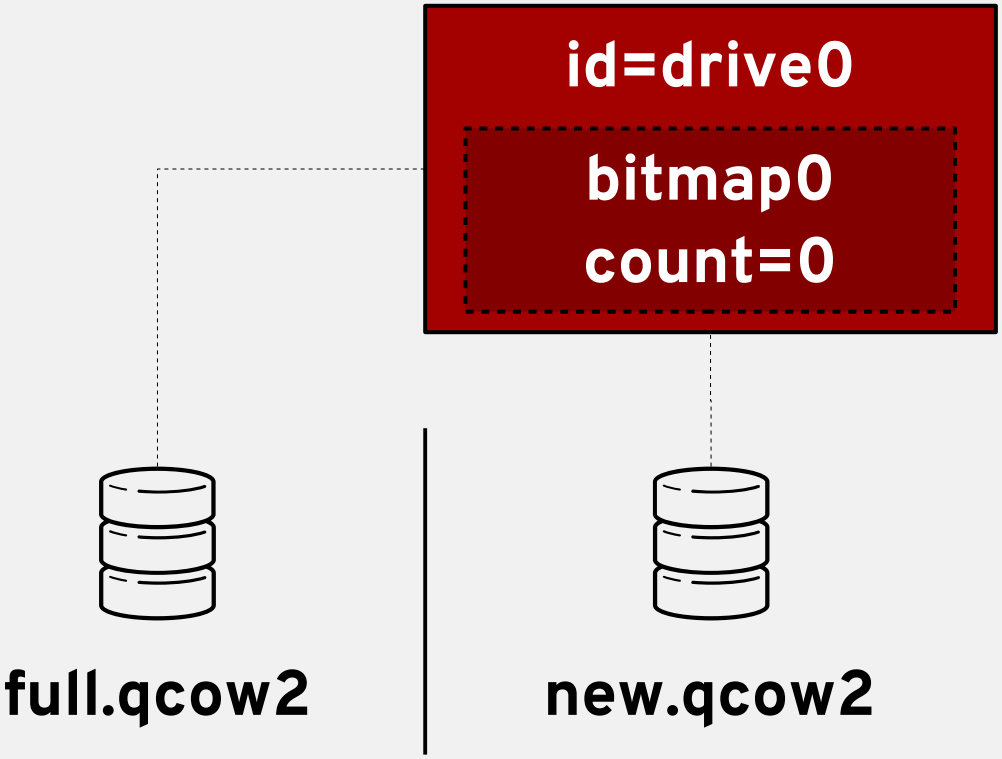
Life Cycle – New Sync Point

(Sunday night maintenance blues)



Life Cycle – New Sync Point

(Sunday night maintenance blues)



Life Cycle – First Incremental

(The first step of our journey)

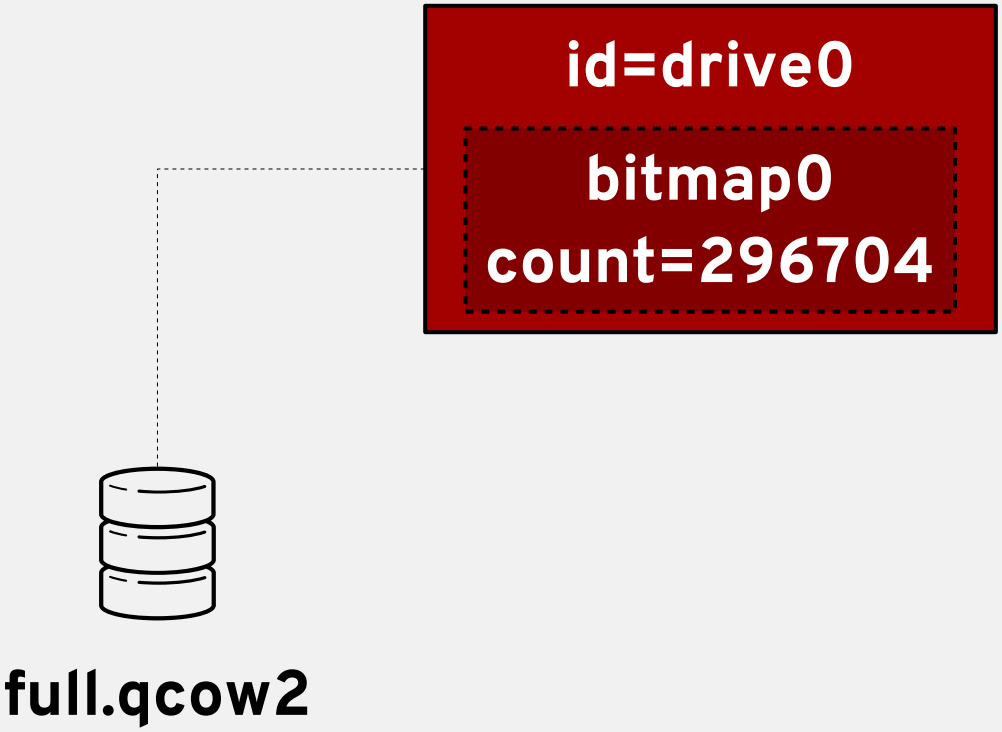
Example 3: Create an incremental backup. Can be done via transaction or single QMP command.

```
# qemu-img create -f qcow2 inc.0.qcow2 -b full.qcow2 -F qcow2
```

```
{ "execute": "drive-backup",
  "arguments": {
    "device": "drive0",
    "bitmap": "bitmap0",
    "target": "inc.0.qcow2",
    "format": "qcow2",
    "sync": "incremental",
    "mode": "existing"
  }
}
```

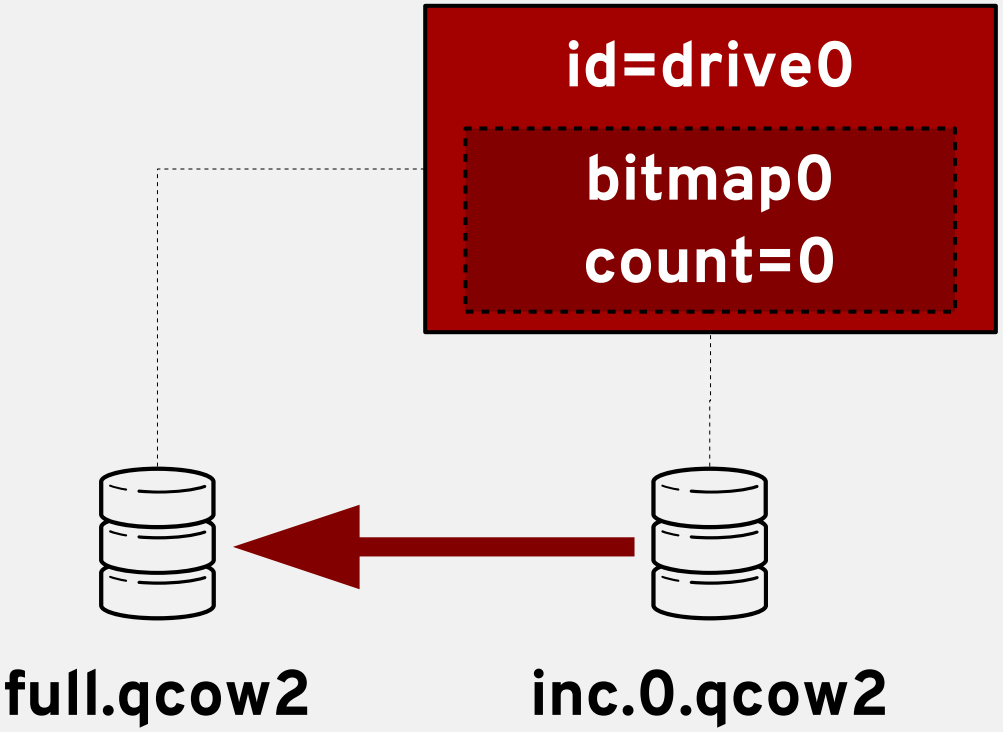
Life Cycle – First Incremental

(The first step of our journey)



Life Cycle – First Incremental

(The first step of our journey)



Life Cycle – Subsequent Backups

(To infinity, and beyond!)

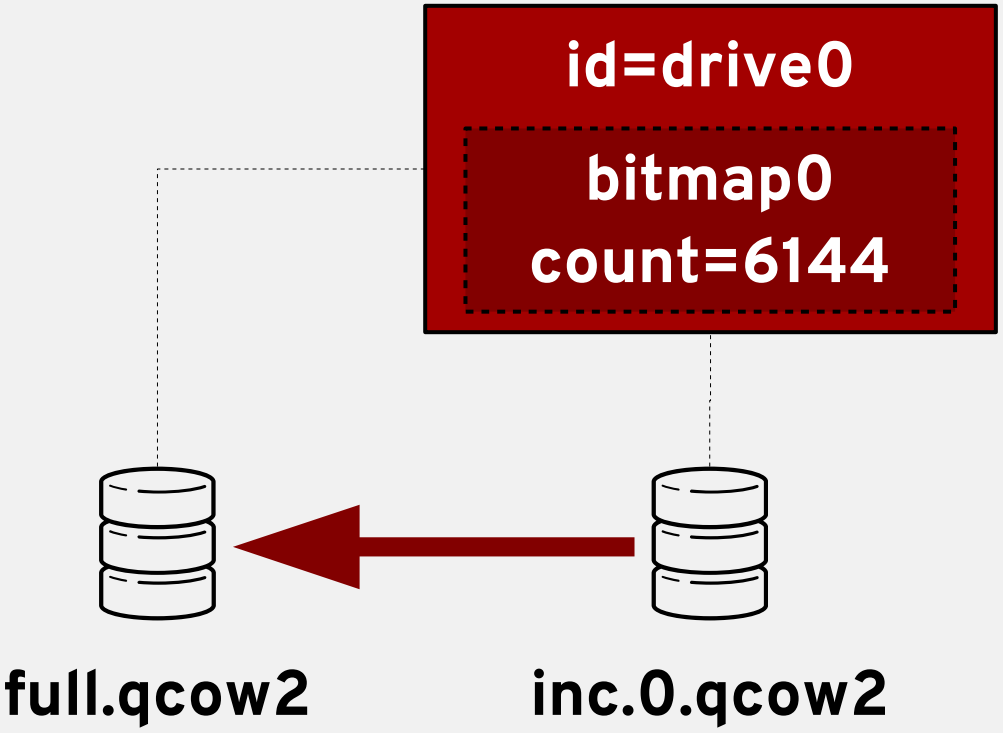
Examples [4,∞): Create subsequent incrementals.

```
# qemu-img create -f qcow2 inc.<n>.qcow2 -b inc.<n-1>.qcow2 -F qcow2
```

```
{ "execute": "drive-backup",
  "arguments": {
    "device": "drive0",
    "bitmap": "bitmap0",
    "target": "inc.<n>.qcow2",
    "format": "qcow2",
    "sync": "incremental",
    "mode": "existing"
  }
}
```

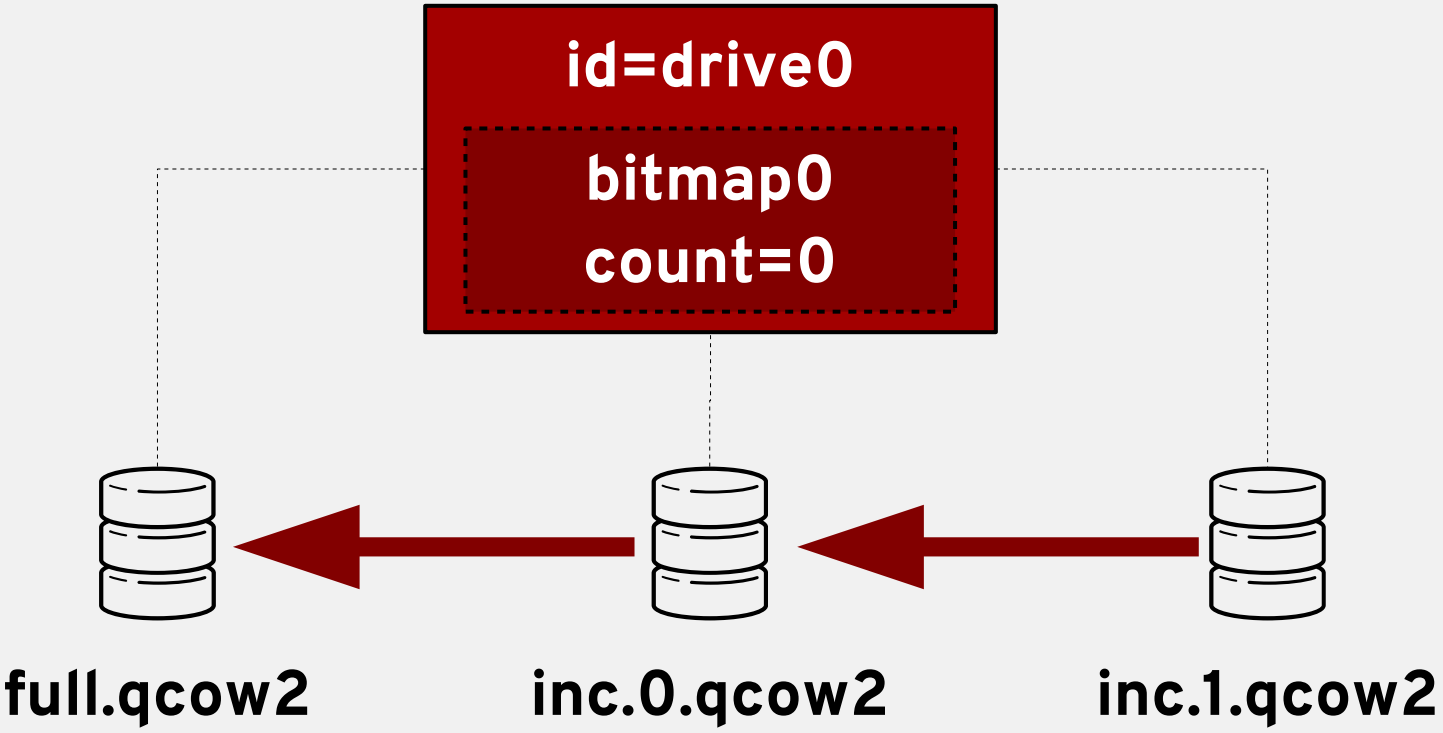
Life Cycle – Subsequent Backups

(To infinity, and beyond!)



Life Cycle – Subsequent Backups

(To infinity, and beyond!)



ACT III: ADVANCED FEATURES

(In which our heroes *rise above*)

Bitmap Migration

(Pack your data, we're moving to <target>)

- Mechanism is similar to disk migration
- Data is split into chunks (1KiB)
  - Bitmaps are serialized piece-by-piece
- For sets of bitmaps below 1MiB…
  - We skip the live phase and copy the data wholesale.
  - 64GiB disk bitmap is only 128KiB
    - (+node and bitmap names, and stream metadata)

Bitmap Migration

(Pack your data, we're moving to <target>)

- Bitmaps are not transferred alongside data
  - Transferred separately for flexibility
- “meta bitmaps” (*dirty “dirty bitmap” bitmaps!?*)
  - Captures any changes during live migration
  - Pieces can be resent if needed.
  - Uses *very little* memory: 64GiB → 16 bytes
- TODO: interoperability with persistence
- Patches on-list now (v4 2015-08-07)

Bitmap Persistence

(Object permanence: not just for toddlers)

- Persistence lets us to save bitmaps across shutdowns
- Having to start a new full backup after shutdown…
  - …Is really no good.
- Currently a work in progress (RFC v1, Vladimir)
- Targeting a qcow2 extension
  - Using qcow2 as a generic container
  - Modeled after snapshot storage
  - Does not require qcow2 for *data*.

Bitmap Persistence

(Object permanence: not just for toddlers)

- qcow2 as a container:
  - Bitmaps can be stored in an “empty” qcow2
  - Multiple bitmaps can be stored in a qcow2
  - Stored bitmaps can describe *other* files
    - They don't have to describe the *same* file
- For convenience:
  - Bitmaps *can* be stored alongside their data

TODOs

(<TODO: insert cheeky joke>)

- QMP interface for modifying persistence attributes
- CLI tools for verification, analysis
  - Deletion/cleaning tools
- “fsck support”
  - qemu-img check -r (?)
- Data integrity
  - Periodic/opportunistic flushing
- Migration: use post-copy?

ACT IV: Dénouement

(In which our heroes live incrementally ever after)

Project Status

(When do we get to use it!?)

- block-dirty-bitmap QMP interface
- sync=incremental mode
- Transactions
- Migration
- Persistence

- Merged! (2.4)
- Merged! (2.4)
- On-List, ETA 2.5
- On-List, ETA 2.5
- RFC, ETA 2.5+


Questions?


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