Backing Chain Management in libvirt and qemu

Eric Blake <eblake@redhat.com>
KVM Forum, August 2015

In this presentation

- How does the qcow2 format track point-in-time snapshots
- What are the qemu building blocks for managing backing chains
- How are these building blocks used together in libvirt

Part I

Understanding qcow2

qcow2 history

- qcow format (QEMU Copy On Write) documented in 2006
- qcow2 created in 2008, adding things like:
  - Internal snapshots with reference counting
- Hacky addition in 2009 to add header extensions
  - Backing file format, to avoid format probing CVEs
- qcow2v3 created in April 2012, adding things like:
  - Feature bits (extension is easier!)
  - Efficient zero cluster management

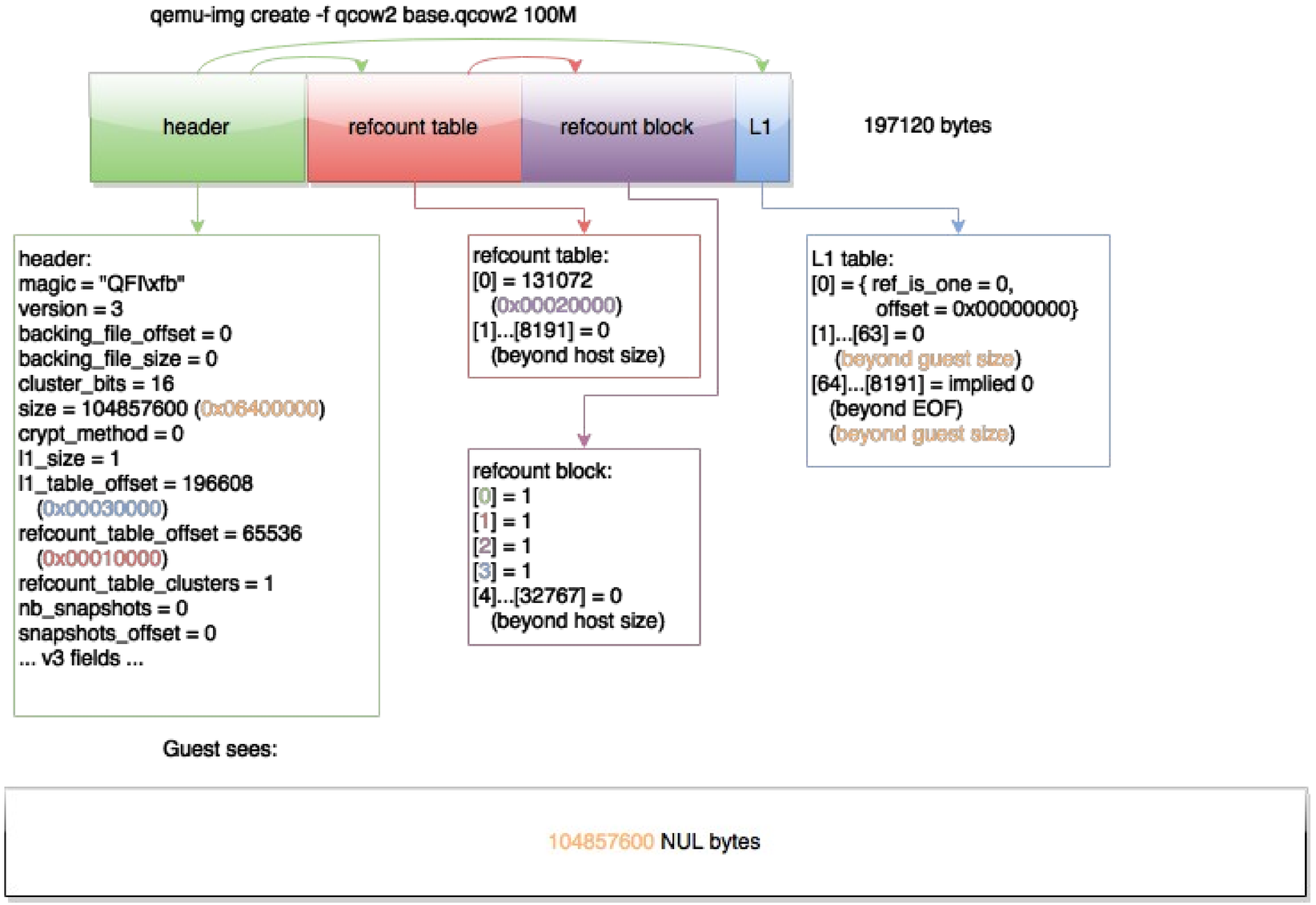
Let's look under the hood

- Create a new file
- Write some guest data
- Create an internal snapshot
- Write more guest data
- Create an external snapshot
- Write even more guest data

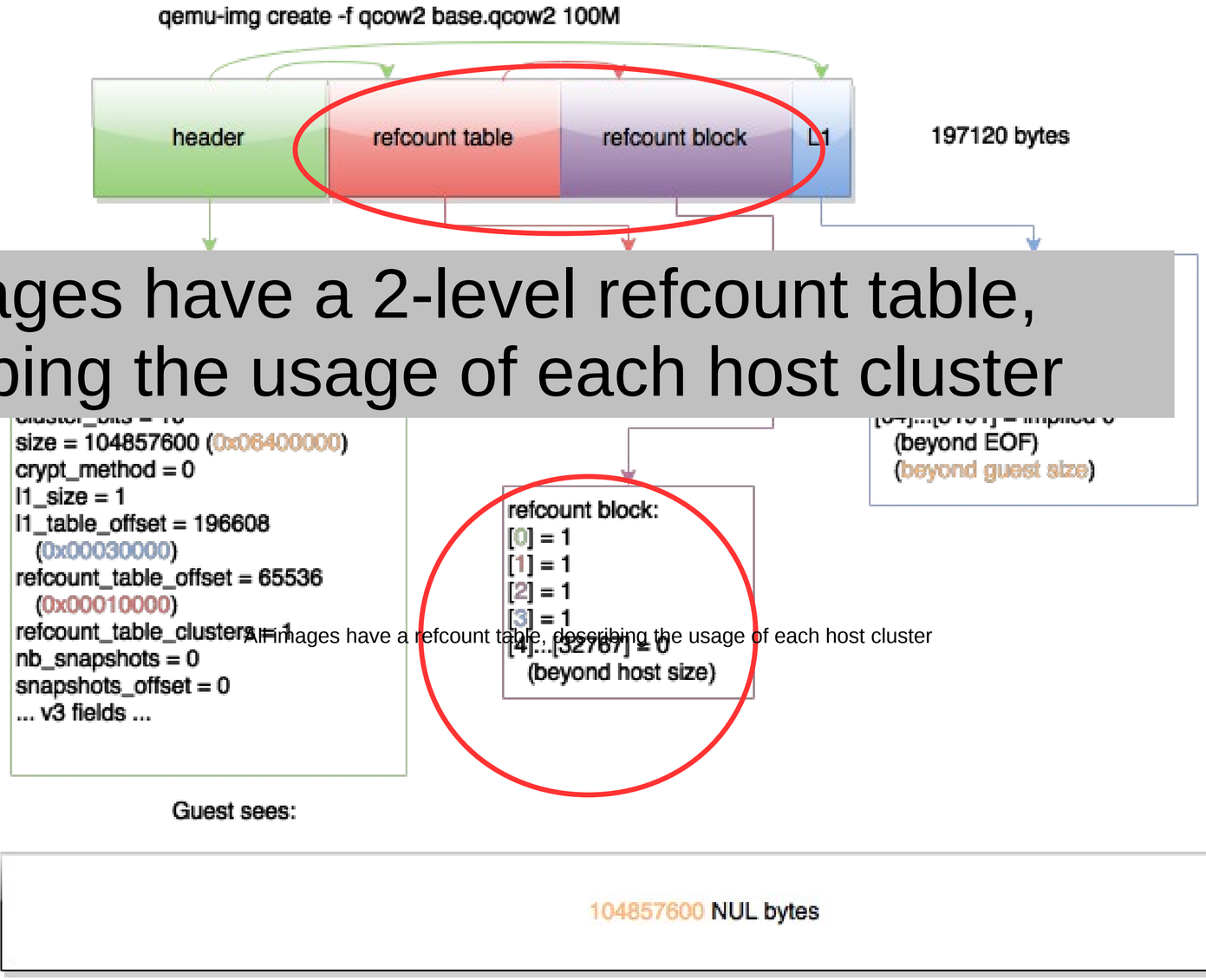
Create a new file

qemu-img create -f qcow2 base.qcow2 100M

Create a new file

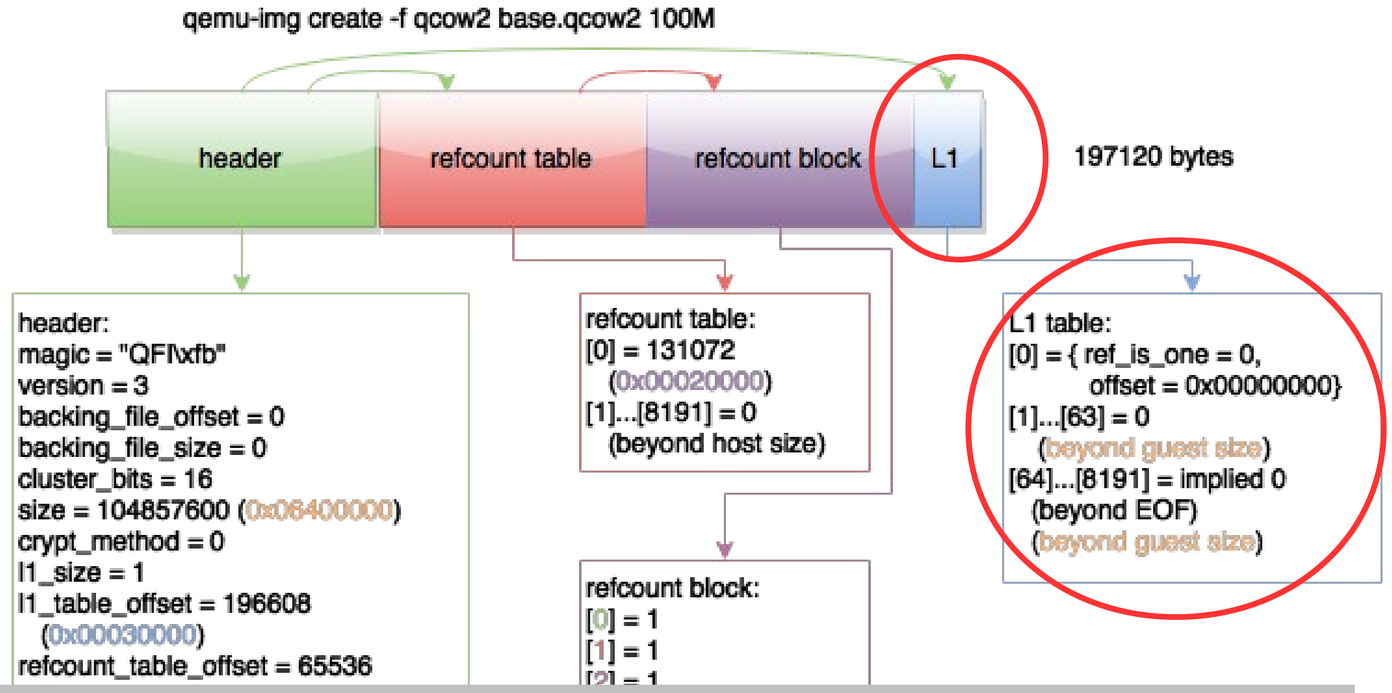


Create a new file



All images have a 2-level refcount table, describing the usage of each host cluster

Create a new file

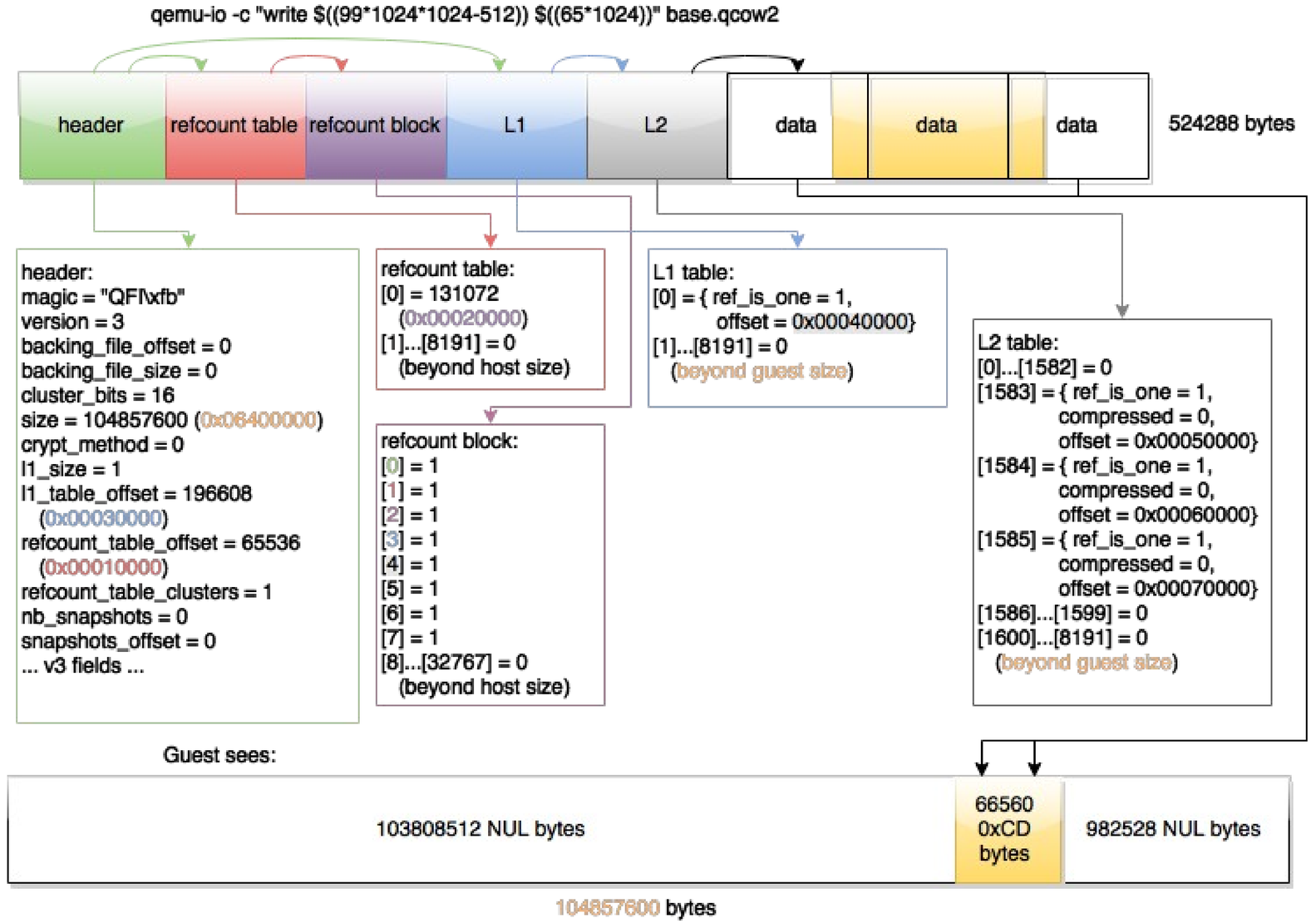


All images have an L1/L2 table,
describing the mapping of each guest cluster
(but with no data mapped, L2 is omitted)

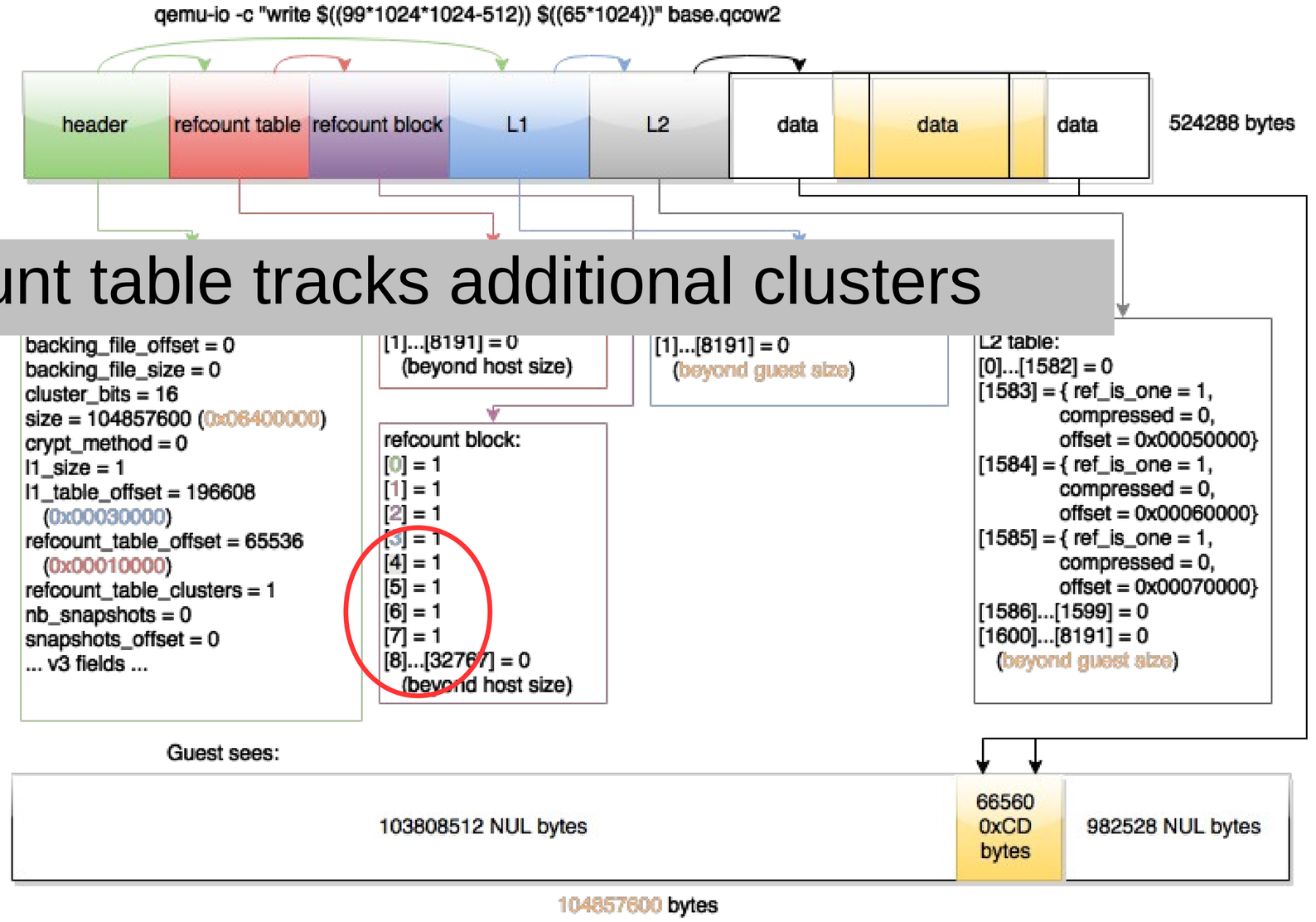
Write some guest data

qemu-io -c “write $((99*1024*1024-512)) $((65*1024))” base.qcow2

Write some guest data

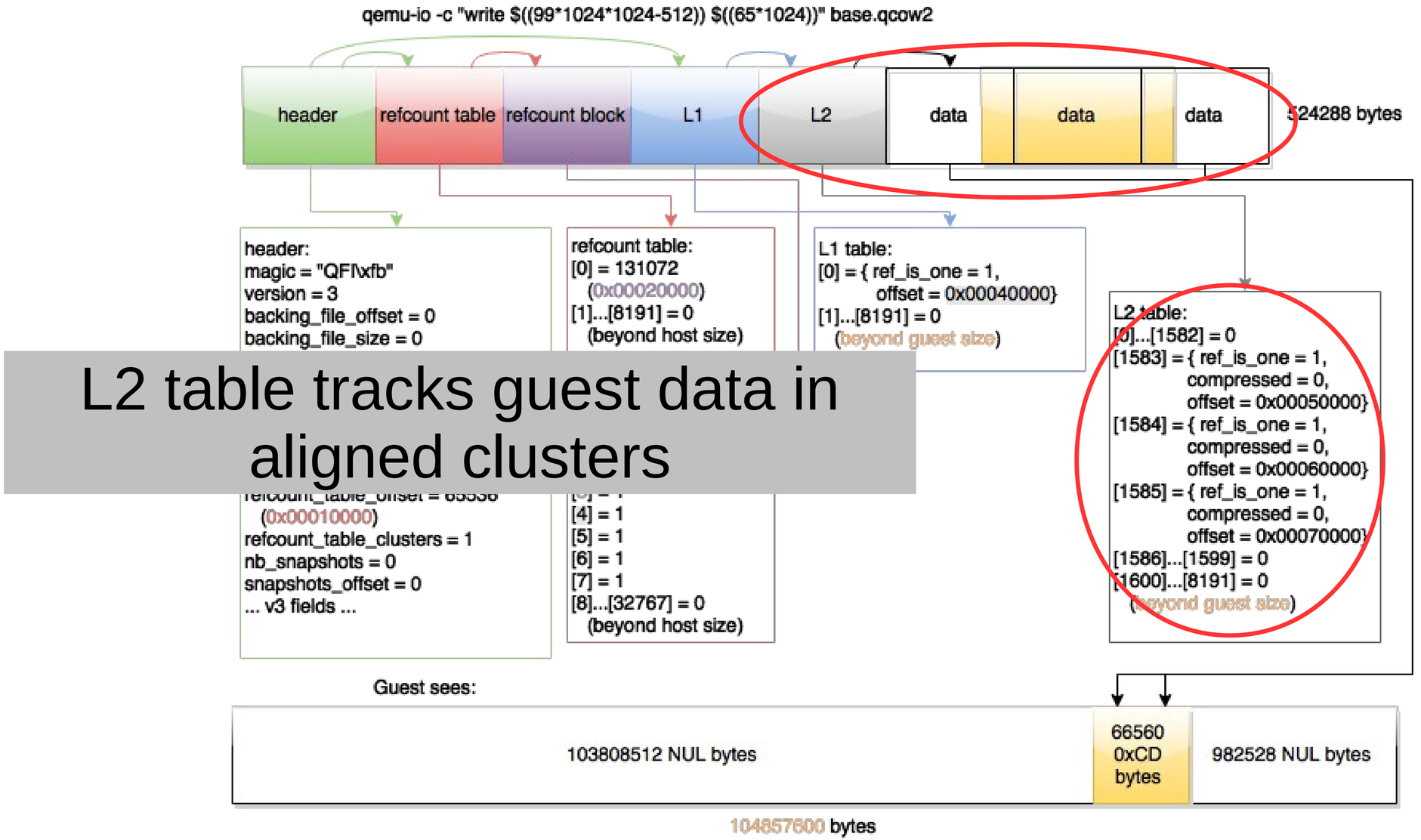


Write some guest data



Refcount table tracks additional clusters

Write some guest data

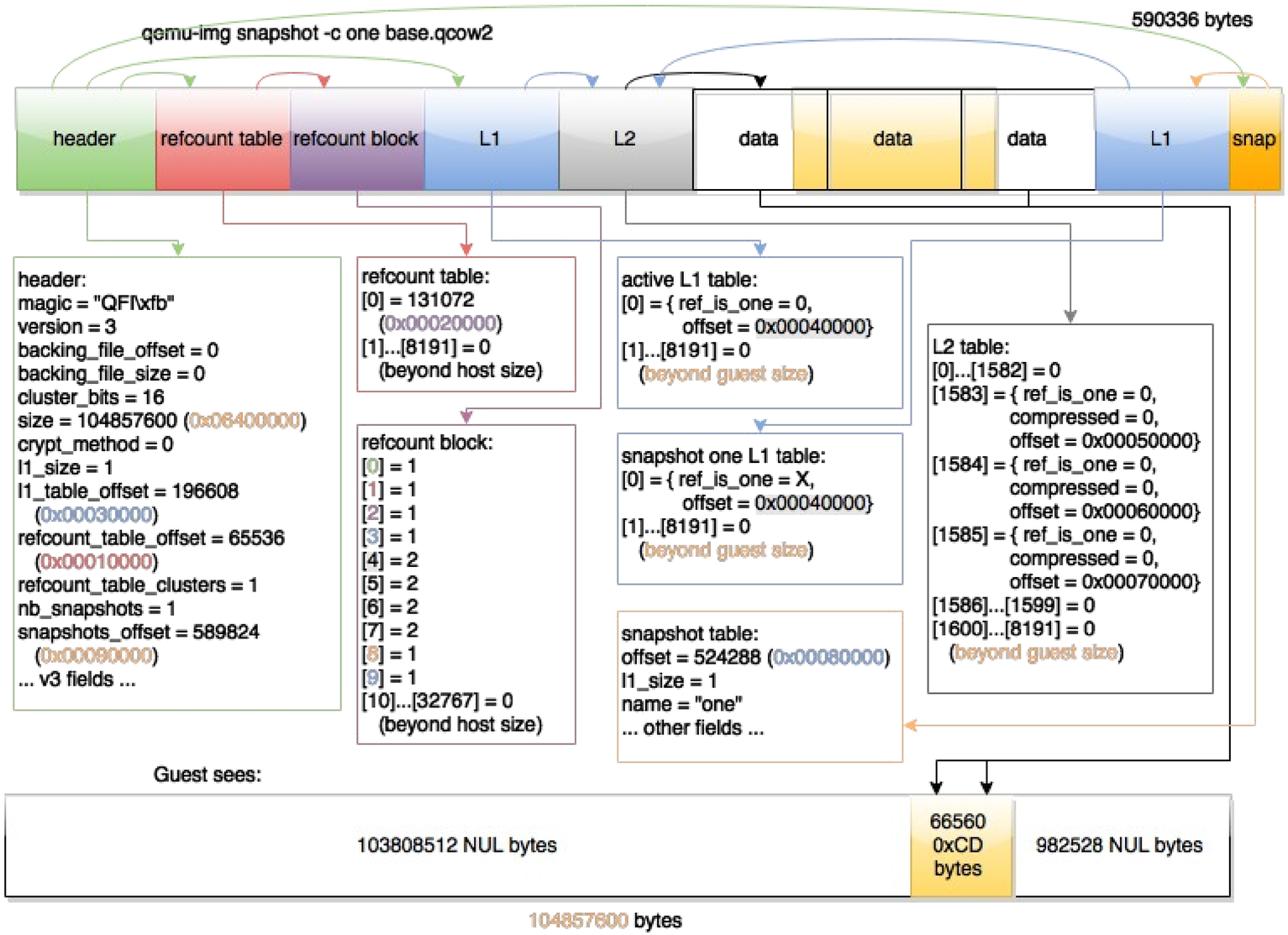


L2 table tracks guest data in aligned clusters

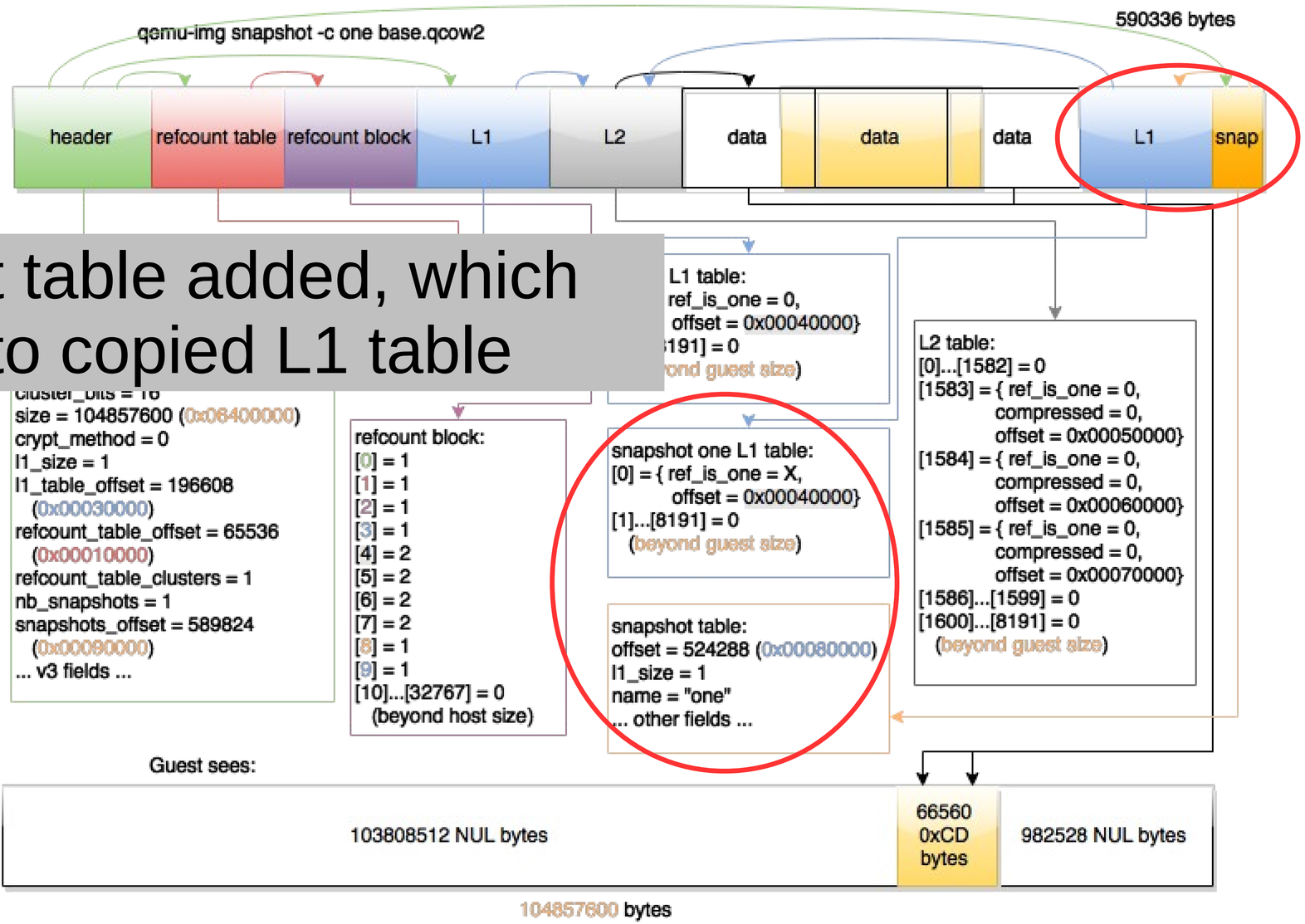
Create an internal snapshot

qemu-img snapshot -c one base.qcow2

Create an internal snapshot

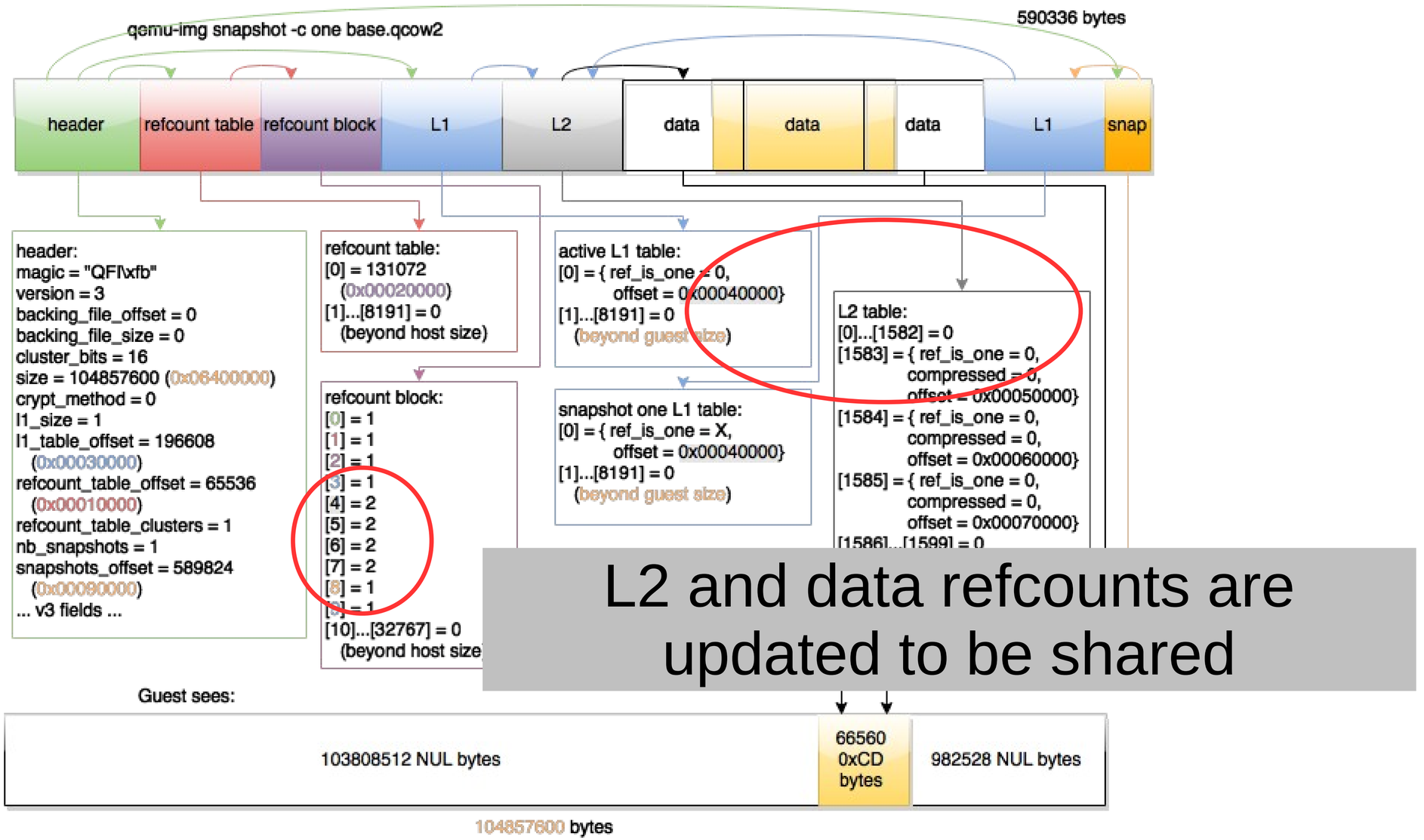


Create an internal snapshot



Snapshot table added, which points to copied L1 table

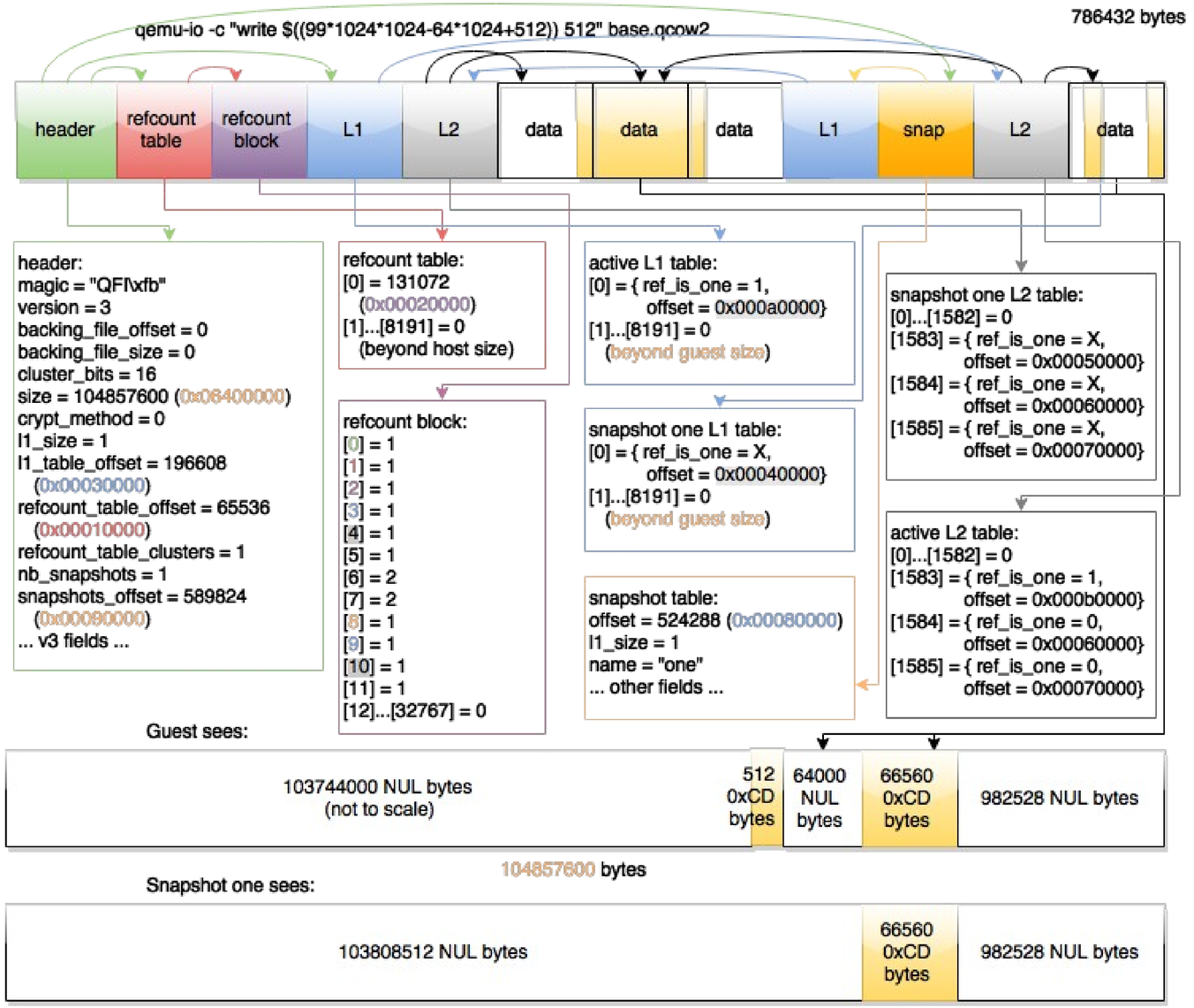
Create an internal snapshot



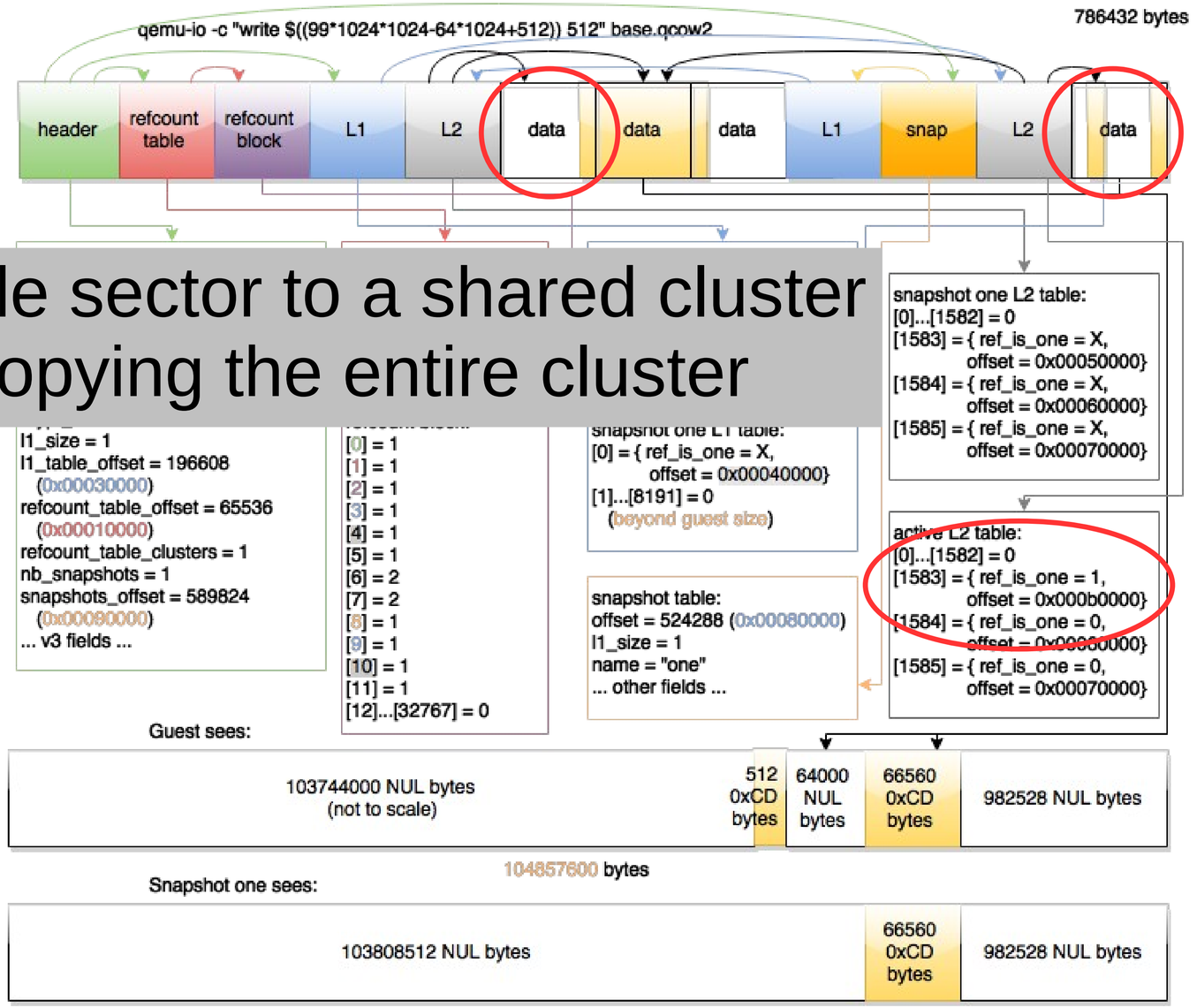
Write more guest data

qemu-io -c “write $((99*1024*1024-64*1024+512)) 512” base.qcow2

Write more guest data

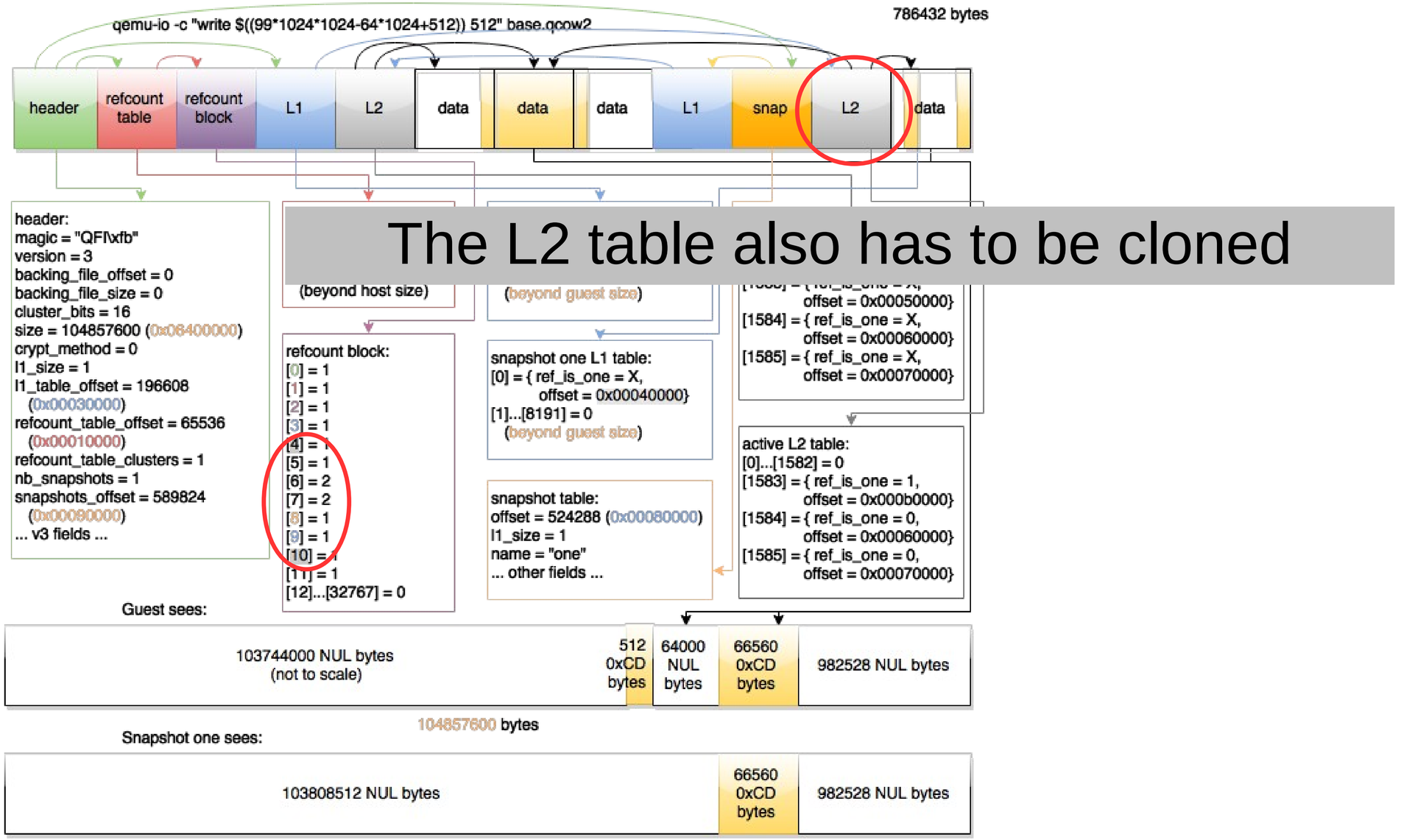


Write more guest data



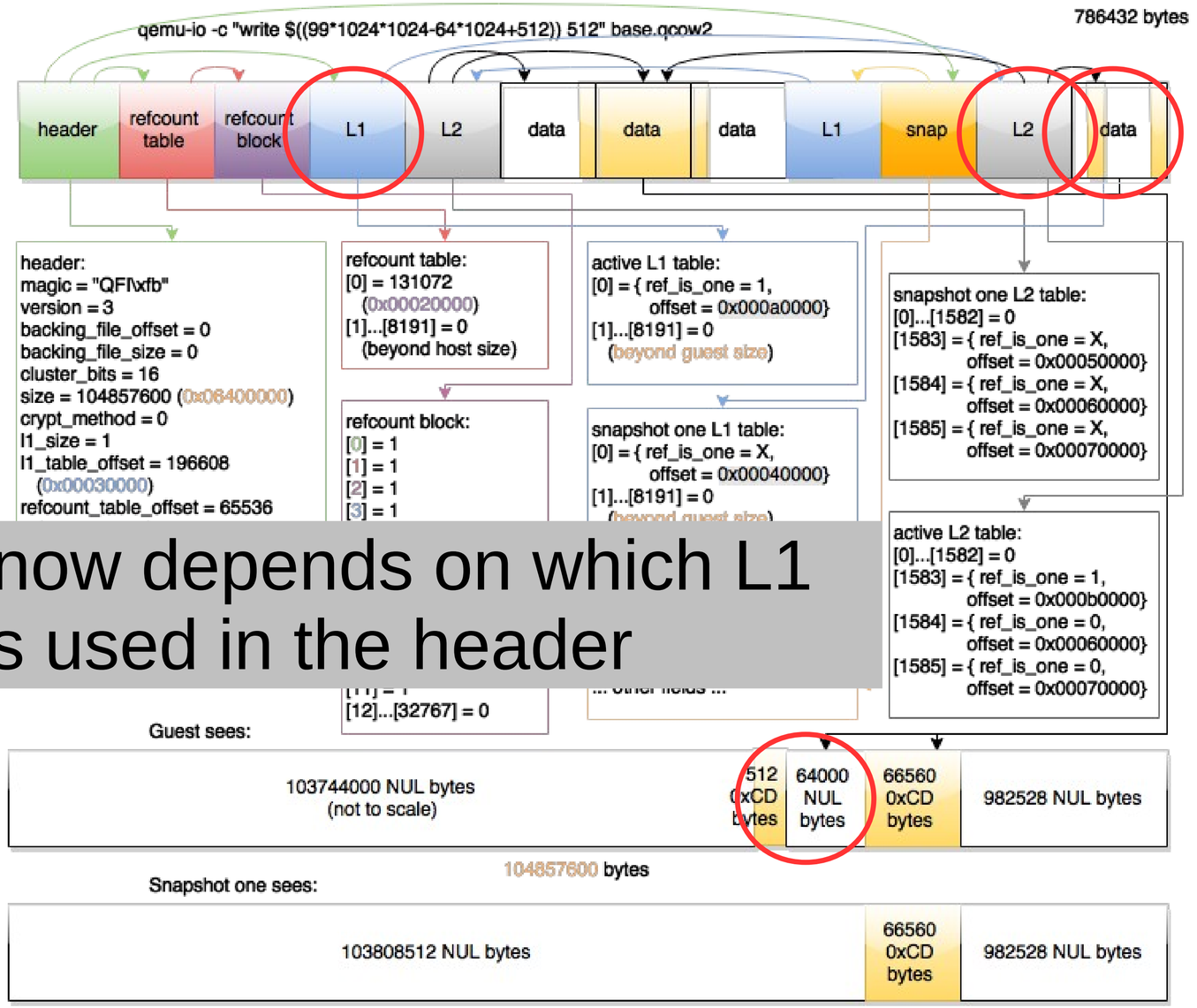
Writing a single sector to a shared cluster requires copying the entire cluster

Write more guest data



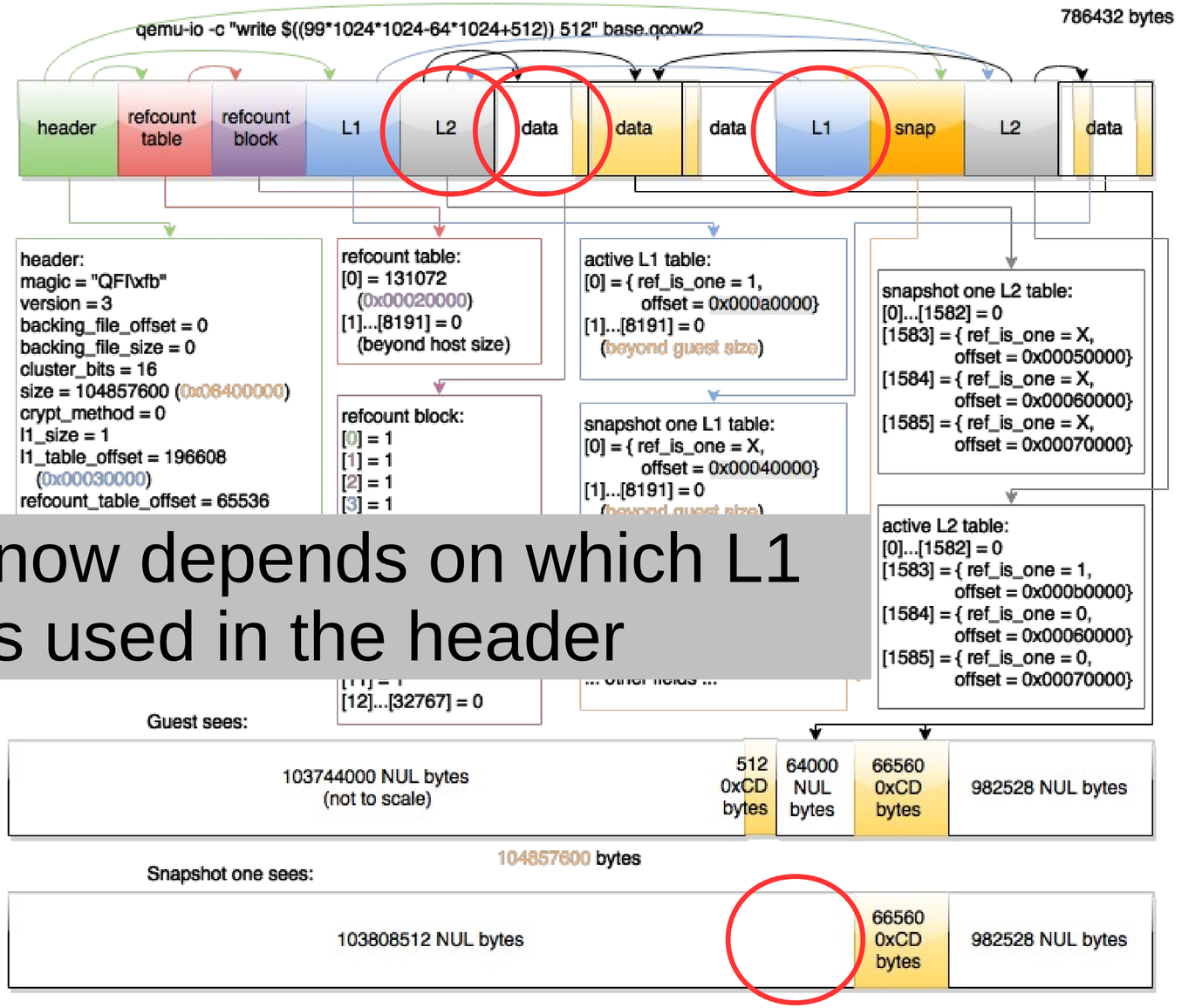
The L2 table also has to be cloned

Write more guest data



Guest view now depends on which L1 table is used in the header

Write more guest data

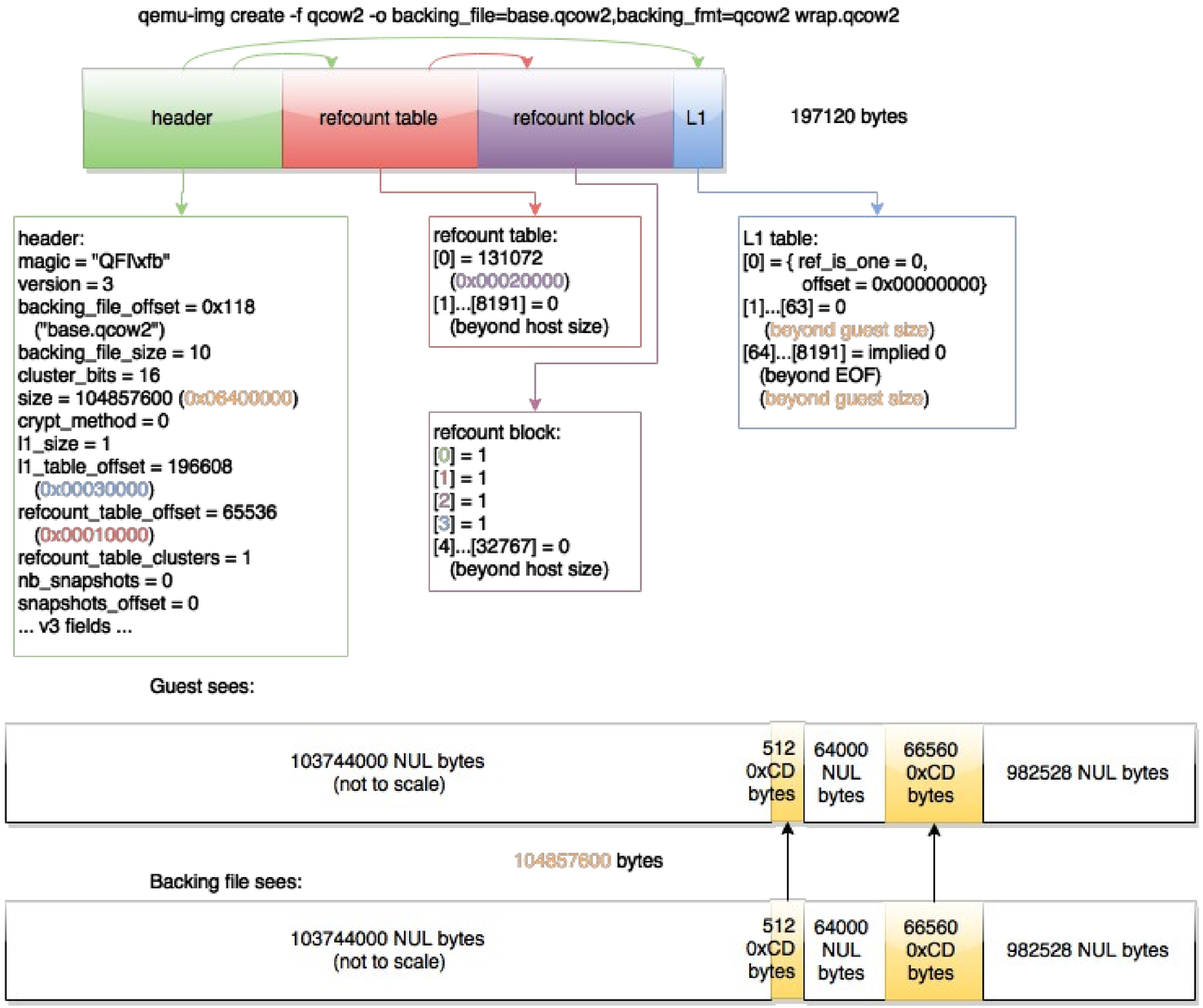


Guest view now depends on which L1 table is used in the header

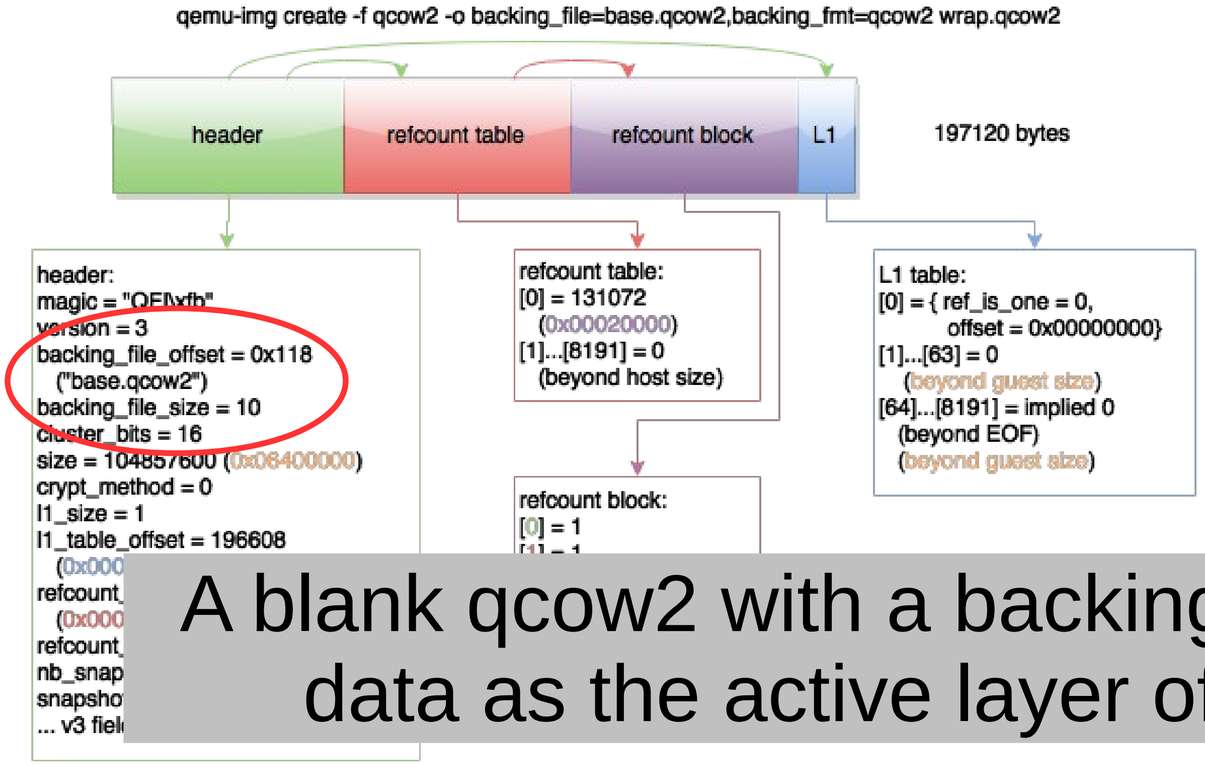
Create an external snapshot

qemu-img create -f qcow2 -o backing_file=base.qcow2,backing_fmt=qcow2 wrap.qcow2

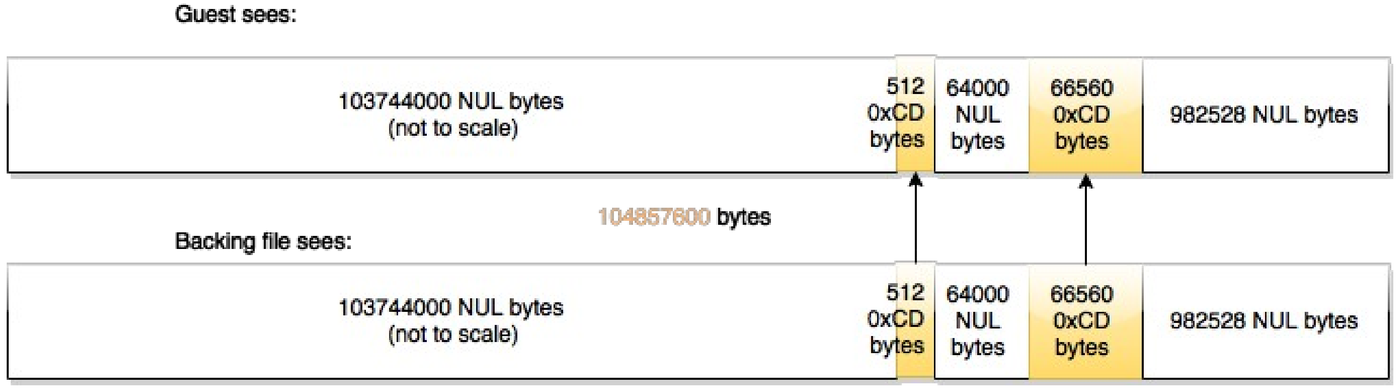
Create an external snapshot



Create an external snapshot



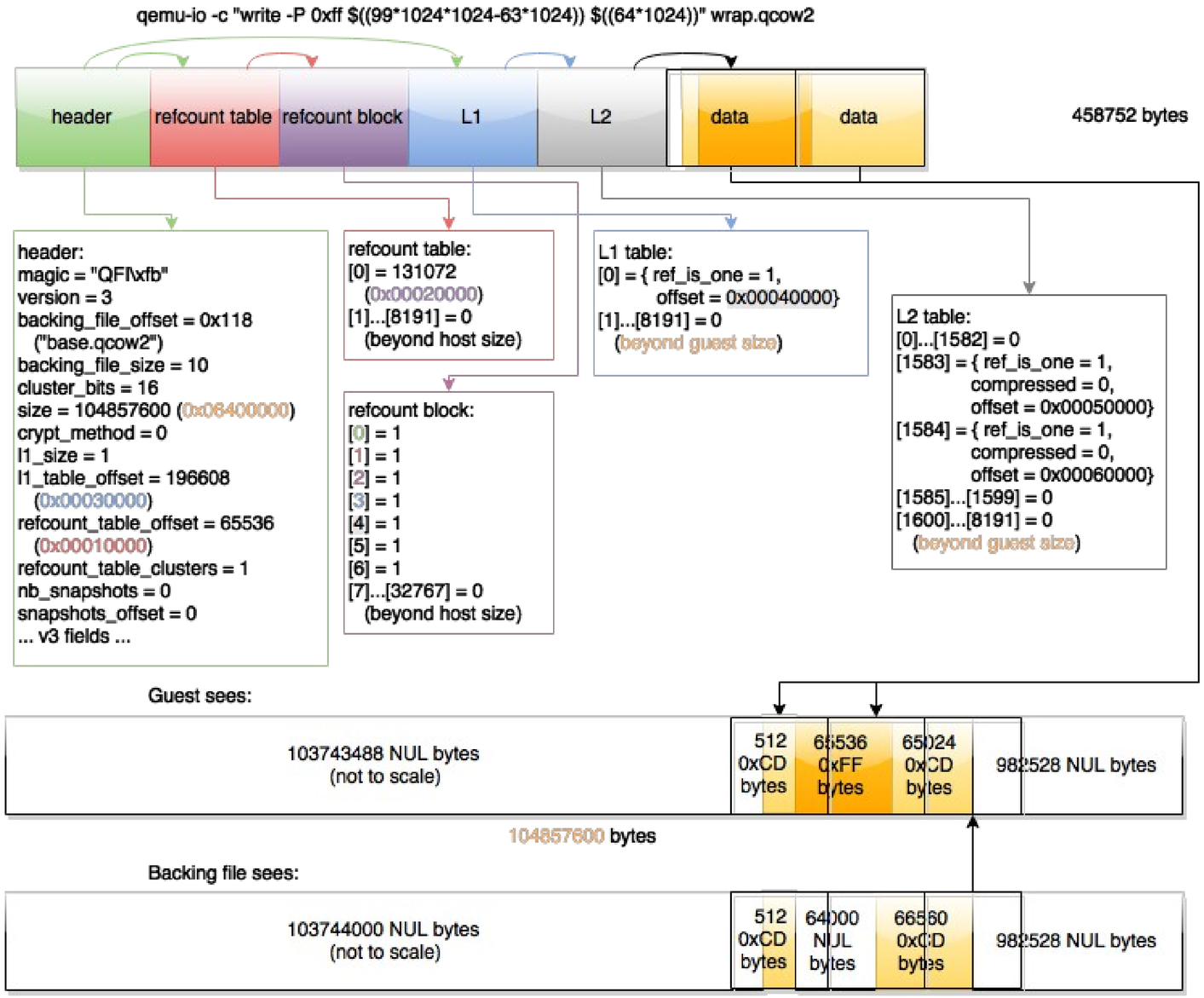
A blank qcow2 with a backing file sees the same data as the active layer of the backing file



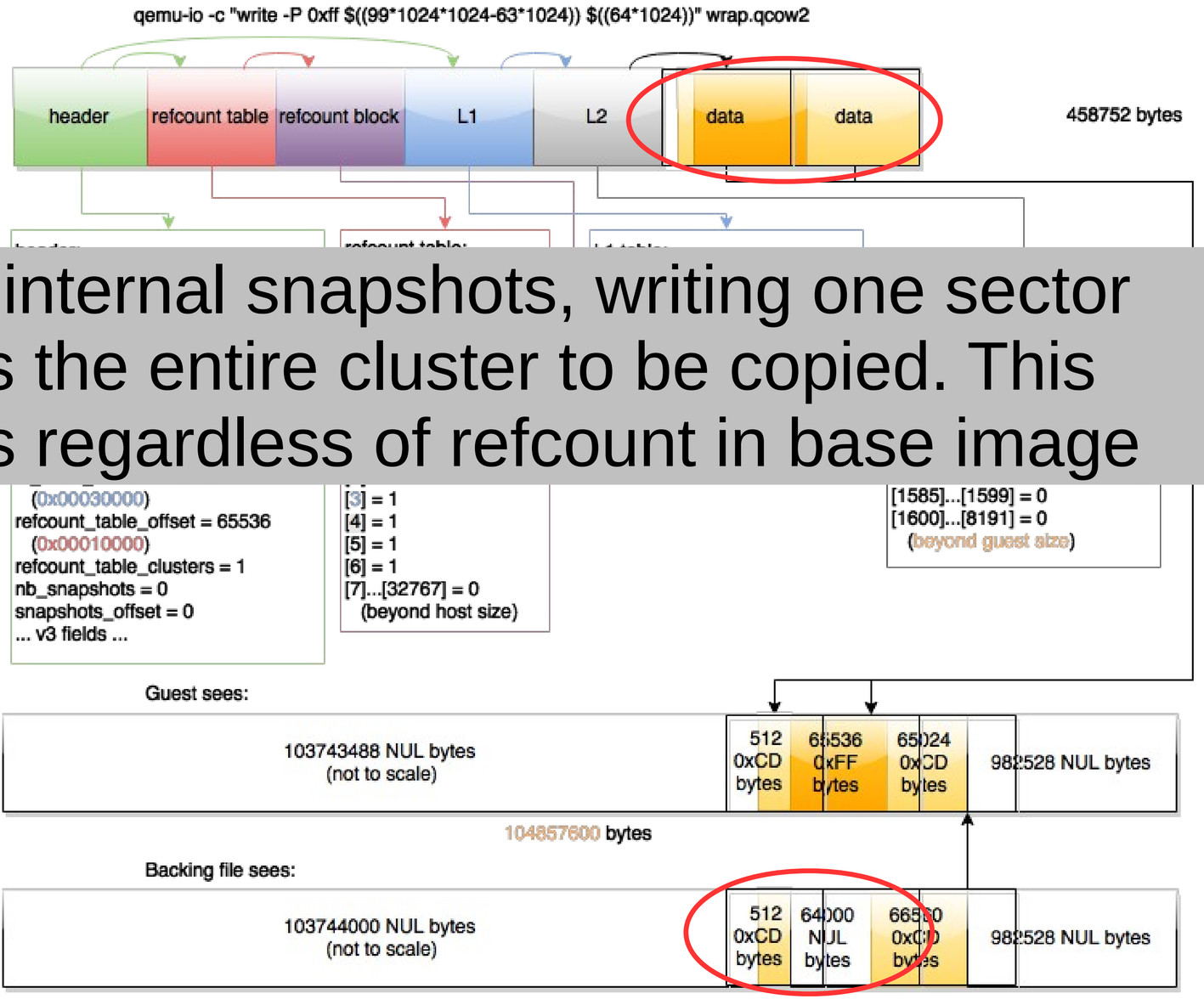
Write even more guest data

qemu-io -c “write -P 0xff $((99*1024*1024-63*1024)) $((64*1024))” wrap.qcow2

Write even more guest data

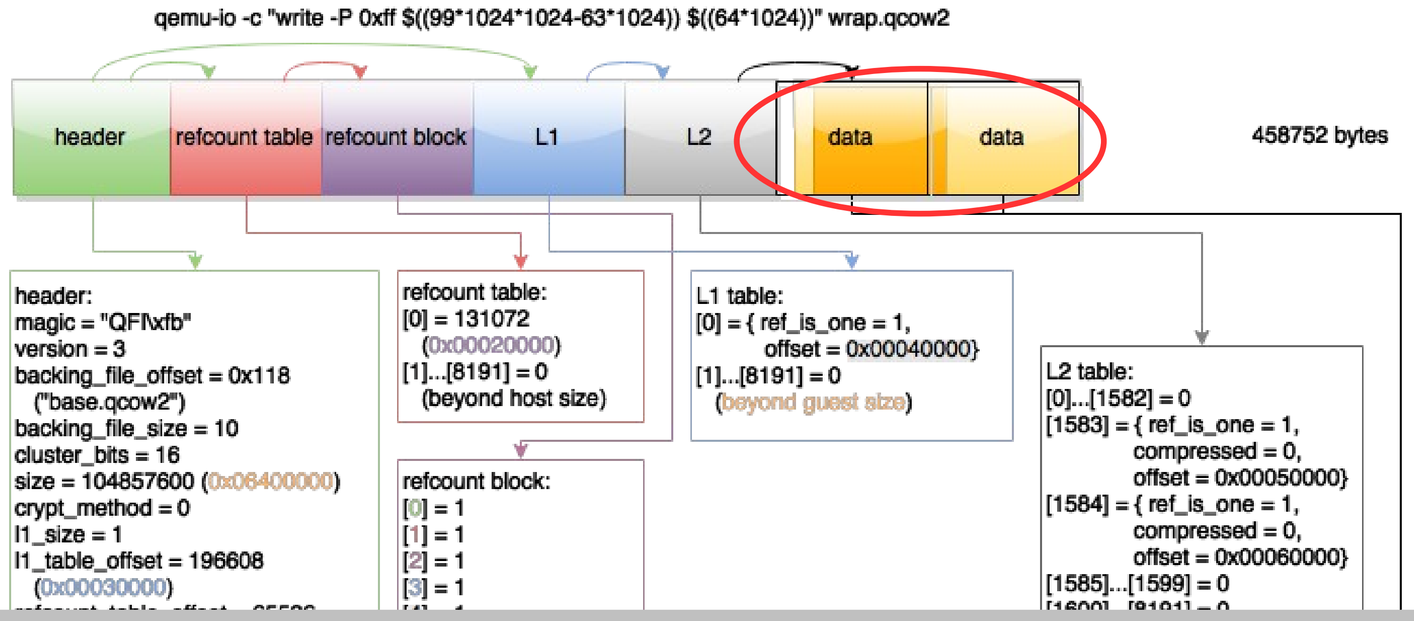


Write even more guest data

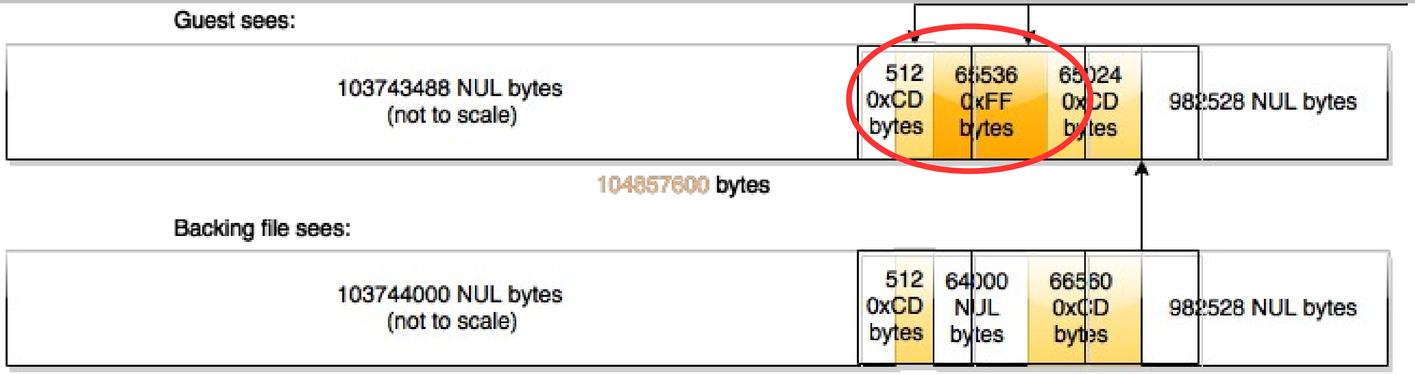


As with internal snapshots, writing one sector causes the entire cluster to be copied. This happens regardless of refcount in base image

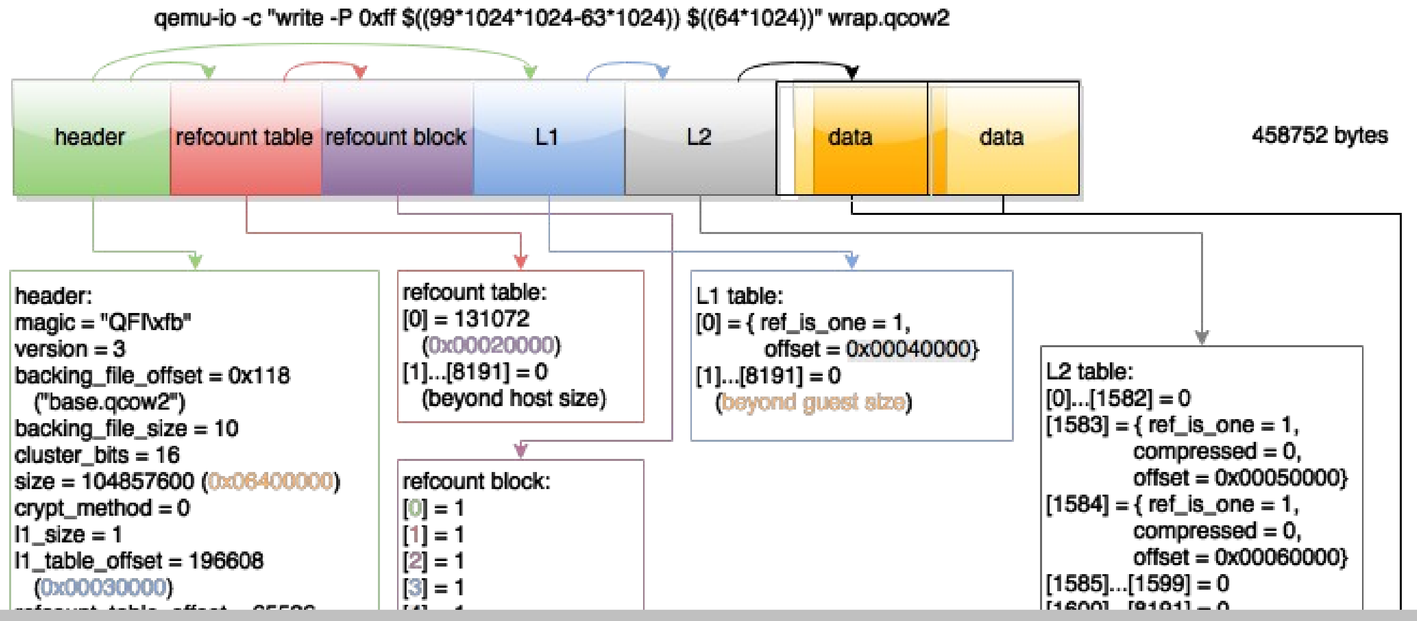
Write even more guest data



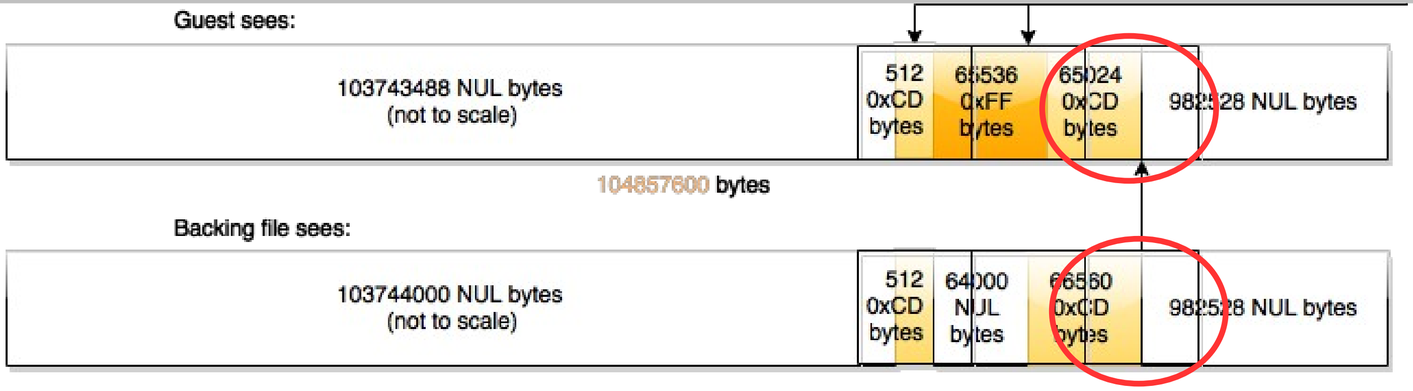
Reading a cluster finds the first file from the top of the chain that contains the cluster



Write even more guest data



Reading a cluster finds the first file from the top of the chain that contains the cluster



Part II

Backing Chains

Internal Snapshots

Pros

- Single file contains everything, optionally including live VM state
- Reverting is easy and supported by libvirt
- No I/O penalties to active state

Cons

- Cannot read snapshot while image is in use by guest; does not allow live backups
- QMP internal snapshot management is inefficient
- qcow2 file size can greatly exceed guest size
- No defragmentation

External Snapshots

Pros

- Live backups and storage migration are easy
- Optimized QMP performance
- Building blocks can be combined in a number of useful patterns
- Great for cluster provisioning from a common base install

Cons

- Deleting snapshots is trickier, libvirt currently delegates to manual qemu-img usage
- Multiple files to track
- I/O overhead in long chains

Backing Chain diagrams

- Notation “A ← B” for “image A backs image B”
- More recent wrappers listed on the right (also called top)
- The chain we created earlier is represented as:
  - base.qcow2 ← wrap.qcow2
- 'qemu-img map' can show where clusters live

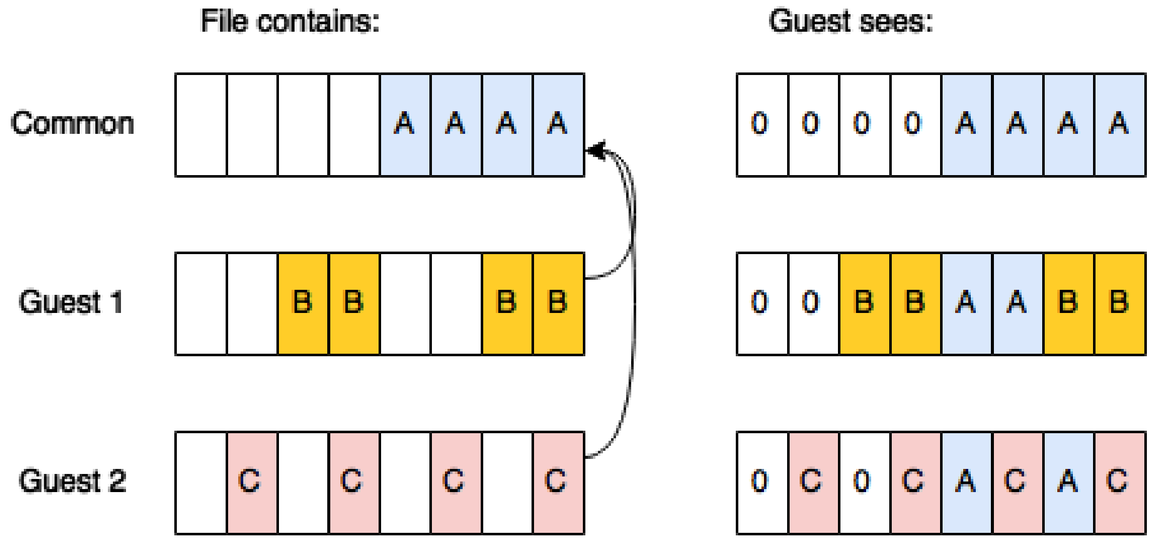
```
$ qemu-img map wrap.qcow2
Offset    Length  Mapped to File
0x62f0000 0x20000 0x50000   wrap.qcow2
0x6300000 0x10000 0x70000   base.qcow2
```

Points in time vs. file names

- Given the chain “A ← B ← C”, we have 2 points in time and an active layer
- Point 1: Guest state when B was created, contained in file A
- Point 2: Guest state when C was created, contained in A+B
- Active layer: Current guest state, contained in A+B+C
- Be careful with naming choices:
  - Naming a file after the time it is created is misleading – the guest data for that point in time is NOT contained in that file
  - Rather, think of files as a delta from the backing file

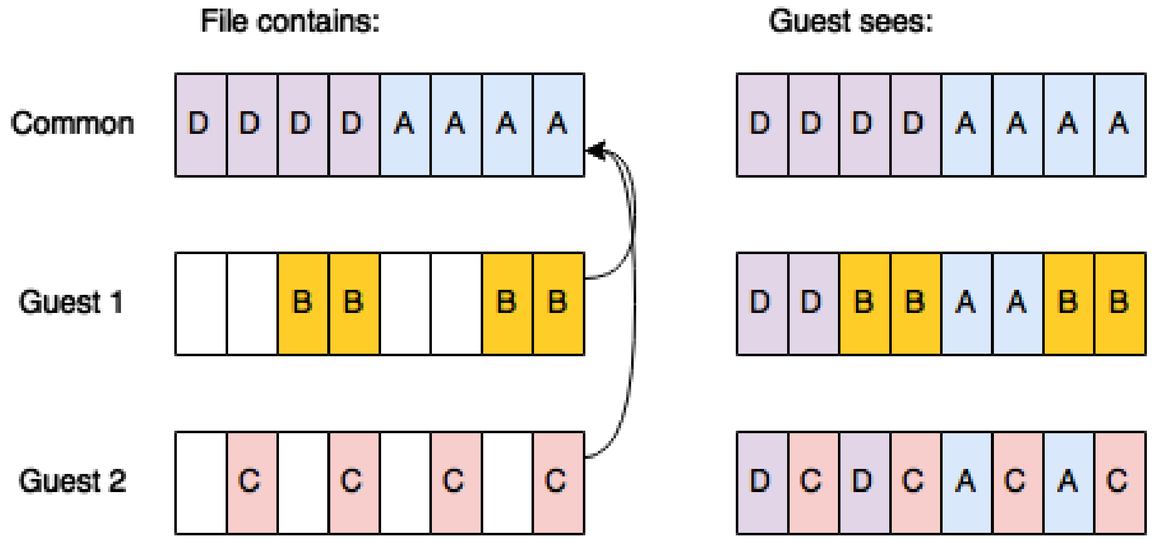
Backing files must not change

- Qcow2 block operations are NOT a substitute for overlayfs
- Observe what happens if a common backing file is modified
- Data seen by dependent images is now different from any state ever possibly observed by the guest, also different from base



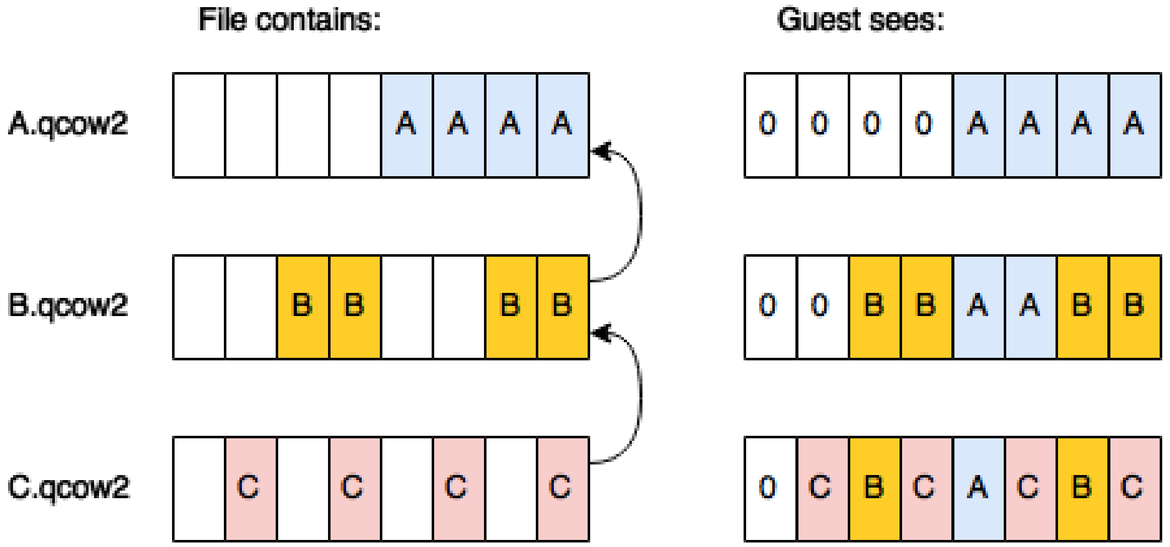
Backing files must not change

- Qcow2 block operations are NOT a substitute for overlayfs
- Observe what happens if a common backing file is modified
- Data seen by dependent images is now different from any state ever possibly observed by the guest, also different from base



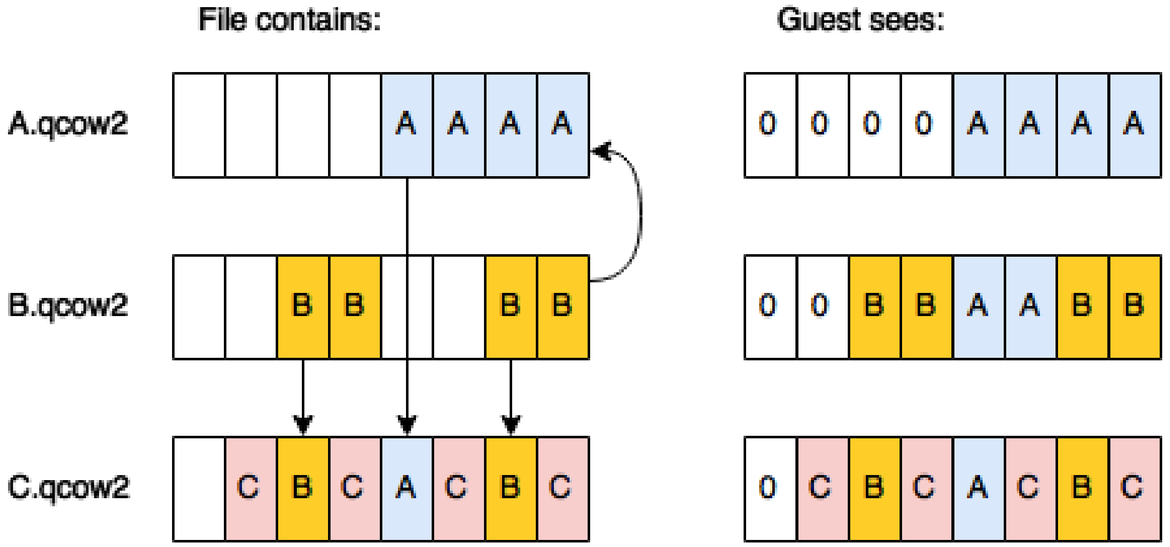
Block-stream primitive (“pull”)

- Starting with “A ← B ← C”, copy/move clusters towards the top
- Additionally, rewrite backing data to drop now-redundant files
- qemu 2.4 limited to top image (A+B into C, or B into C), but qemu 2.5 will add intermediate streaming (A into B)
- Always safe, restartable



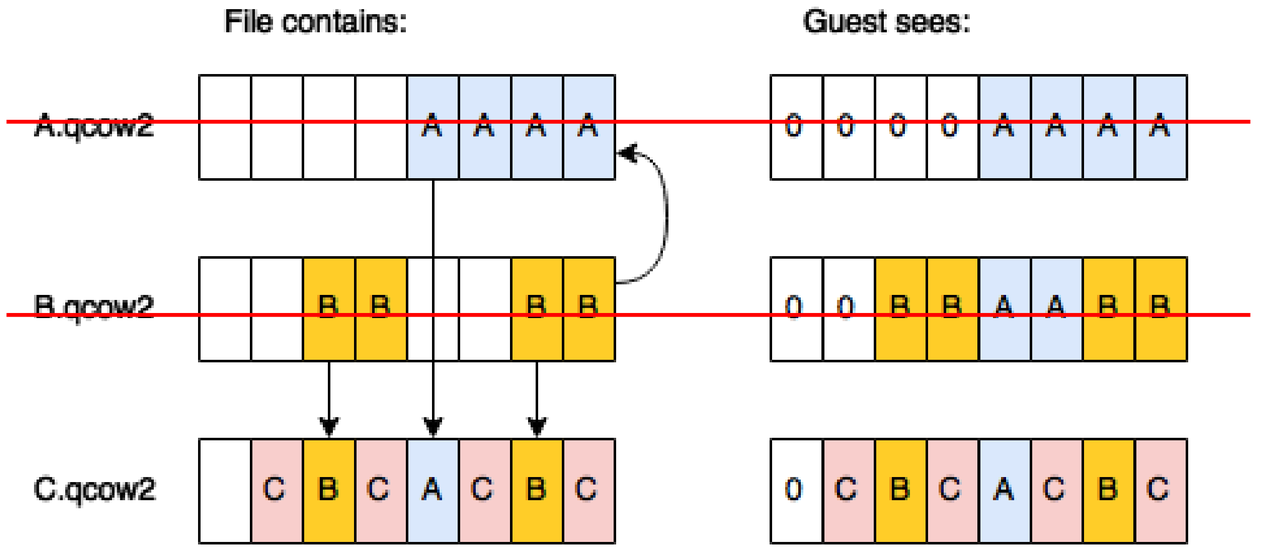
Block-stream primitive (“pull”)

- Starting with “A ← B ← C”, copy/move clusters towards the top
- Additionally, rewrite backing data to drop now-redundant files
- qemu 2.4 limited to top image (A+B into C, or B into C), but qemu 2.5 will add intermediate streaming (A into B)
- Always safe, restartable



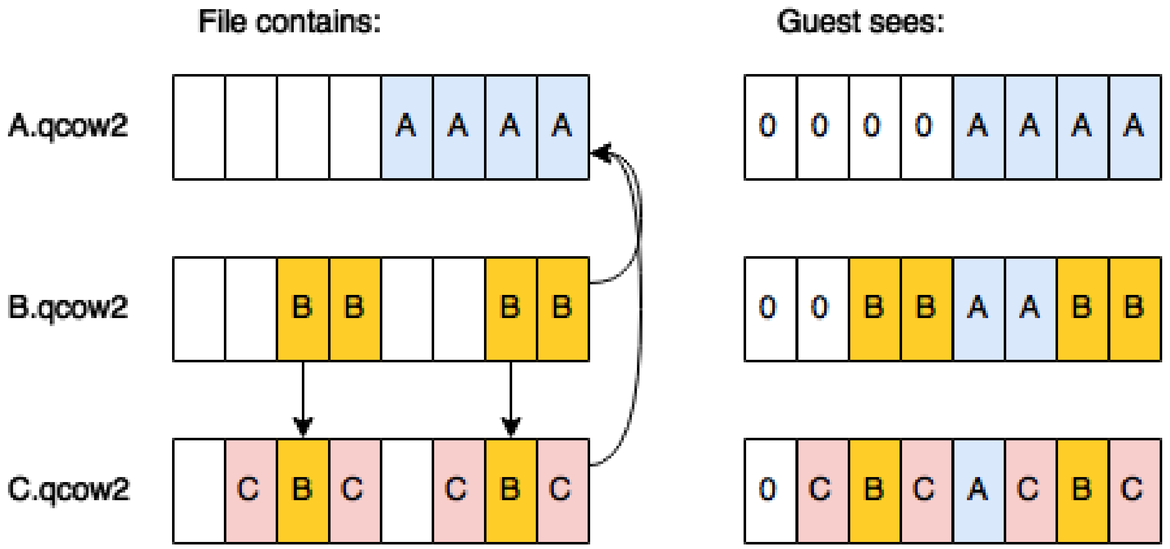
Block-stream primitive (“pull”)

- Starting with “A ← B ← C”, copy/move clusters towards the top
- Additionally, rewrite backing data to drop now-redundant files
- qemu 2.4 limited to top image (A+B into C, or B into C), but qemu 2.5 will add intermediate streaming (A into B)
- Always safe, restartable



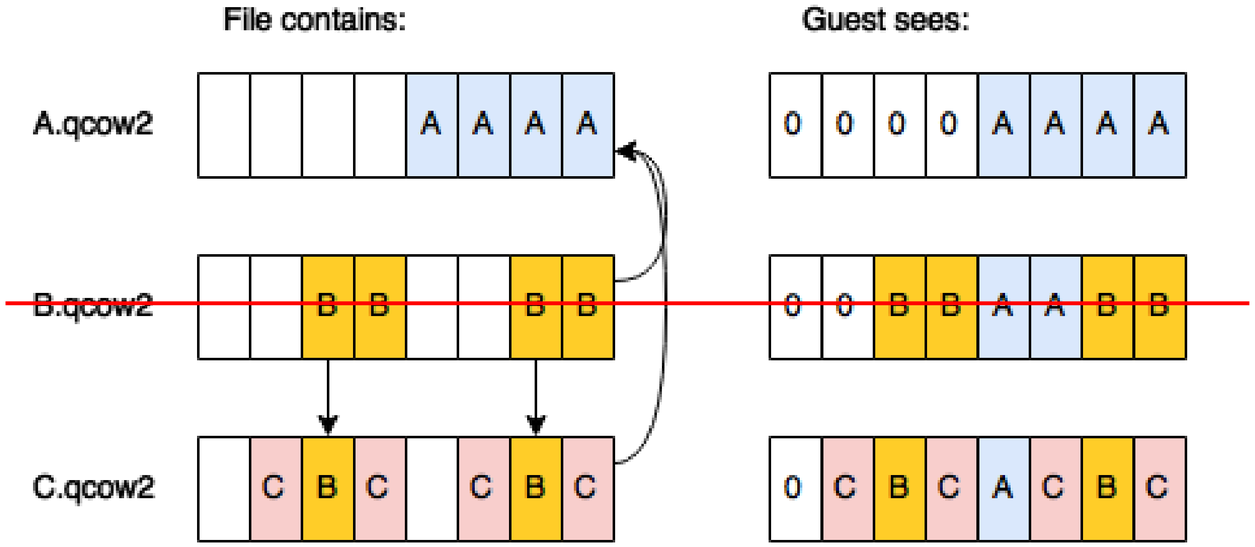
Block-stream primitive (“pull”)

- Starting with “A ← B ← C”, copy/move clusters towards the top
- Additionally, rewrite backing data to drop now-redundant files
- qemu 2.4 limited to top image (A+B into C, or B into C), but qemu 2.5 will add intermediate streaming (A into B)
- Always safe, restartable



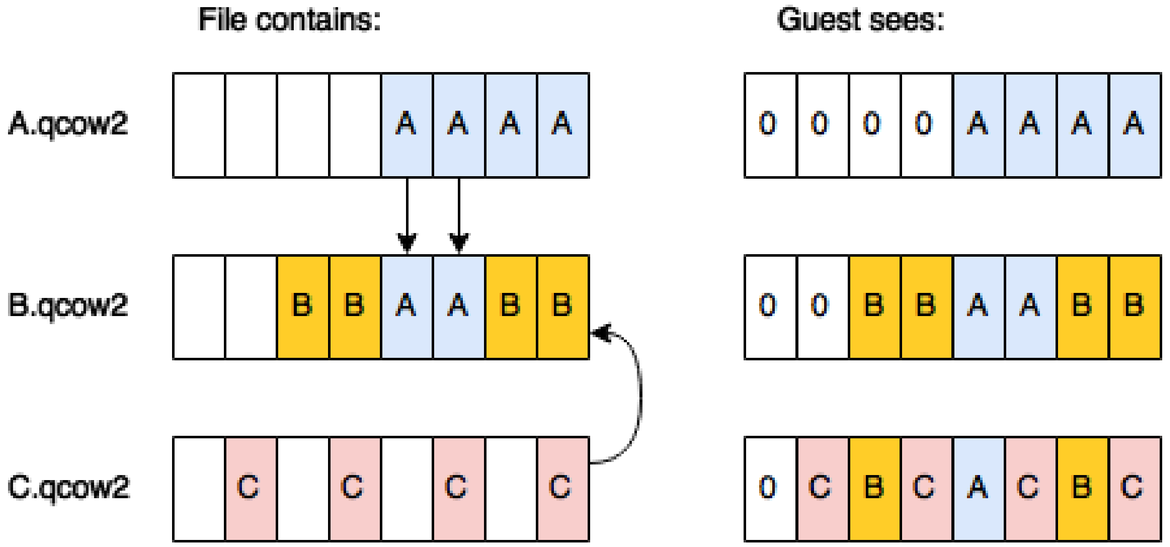
Block-stream primitive (“pull”)

- Starting with “A ← B ← C”, copy/move clusters towards the top
- Additionally, rewrite backing data to drop now-redundant files
- qemu 2.4 limited to top image (A+B into C, or B into C), but qemu 2.5 will add intermediate streaming (A into B)
- Always safe, restartable



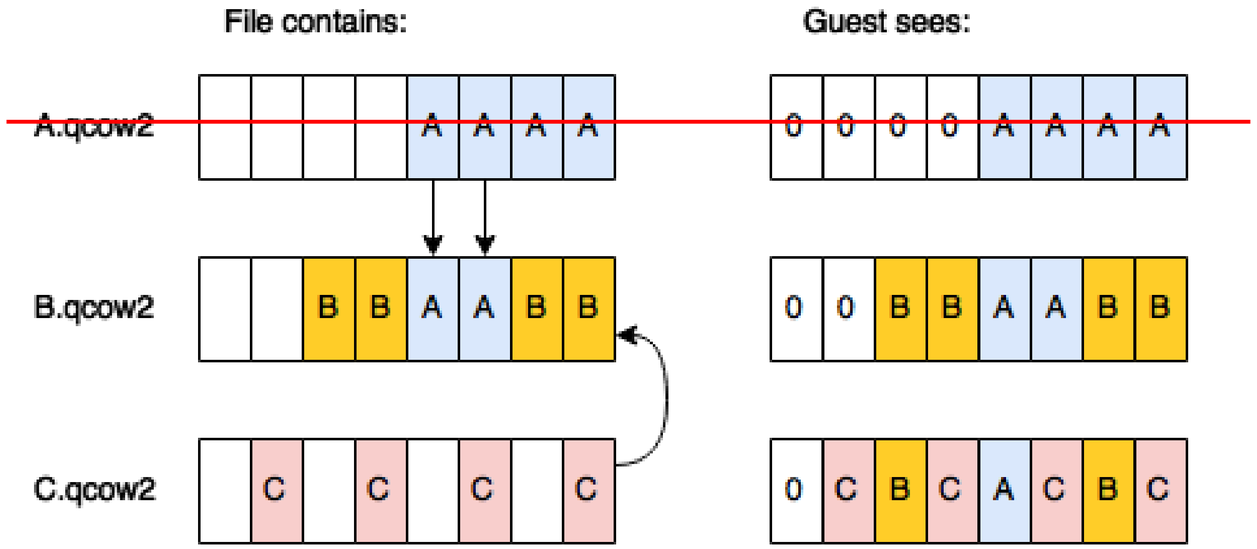
Block-stream primitive (“pull”)

- Starting with “A ← B ← C”, copy/move clusters towards the top
- Additionally, rewrite backing data to drop now-redundant files
- qemu 2.4 limited to top image (A+B into C, or B into C), but qemu 2.5 will add intermediate streaming (A into B)
- Always safe, restartable



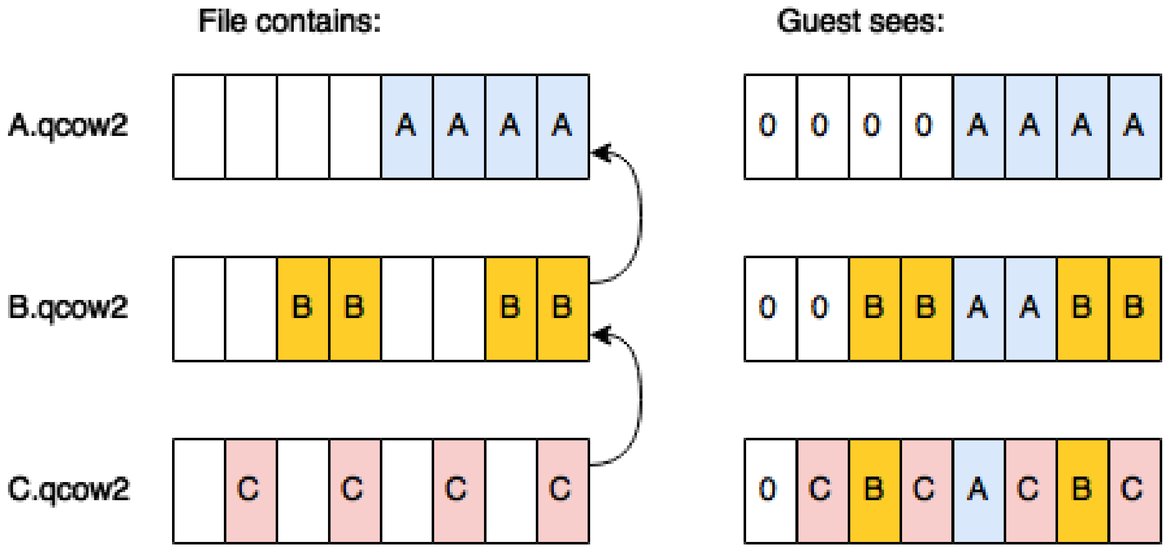
Block-stream primitive (“pull”)

- Starting with “A ← B ← C”, copy/move clusters towards the top
- Additionally, rewrite backing data to drop now-redundant files
- qemu 2.4 limited to top image (A+B into C, or B into C), but qemu 2.5 will add intermediate streaming (A into B)
- Always safe, restartable



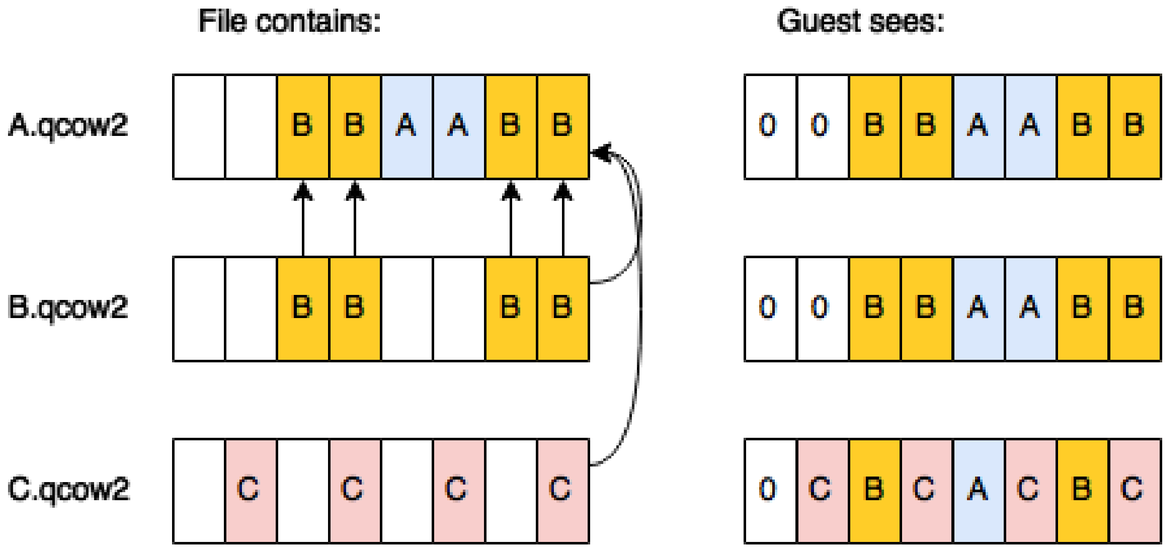
Block-commit primitive (“commit”)

- Starting with “A ← B ← C”, copy/move clusters away from top
- Additionally, rewrite backing data to drop now-redundant files
- qemu 1.3 supported intermediate commit (B into A), qemu 2.0 added active commit (C into B, C+B into A)
- Restartable, but remember caveat about editing a shared base file



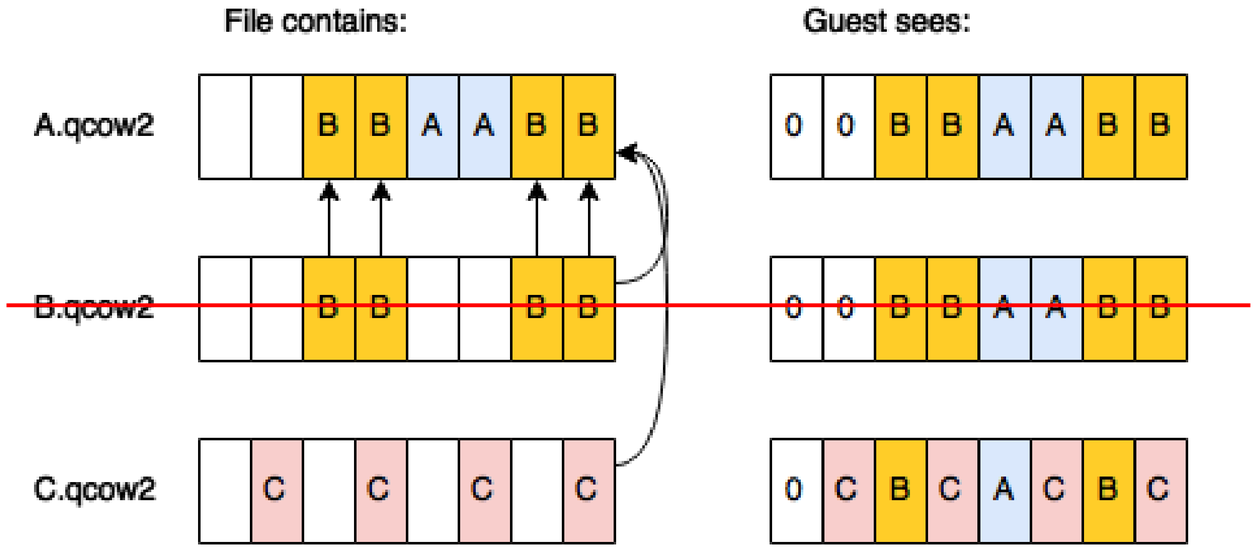
Block-commit primitive (“commit”)

- Starting with “A ← B ← C”, copy/move clusters away from top
- Additionally, rewrite backing data to drop now-redundant files
- qemu 1.3 supported intermediate commit (B into A), qemu 2.0 added active commit (C into B, C+B into A)
- Restartable, but remember caveat about editing a shared base file



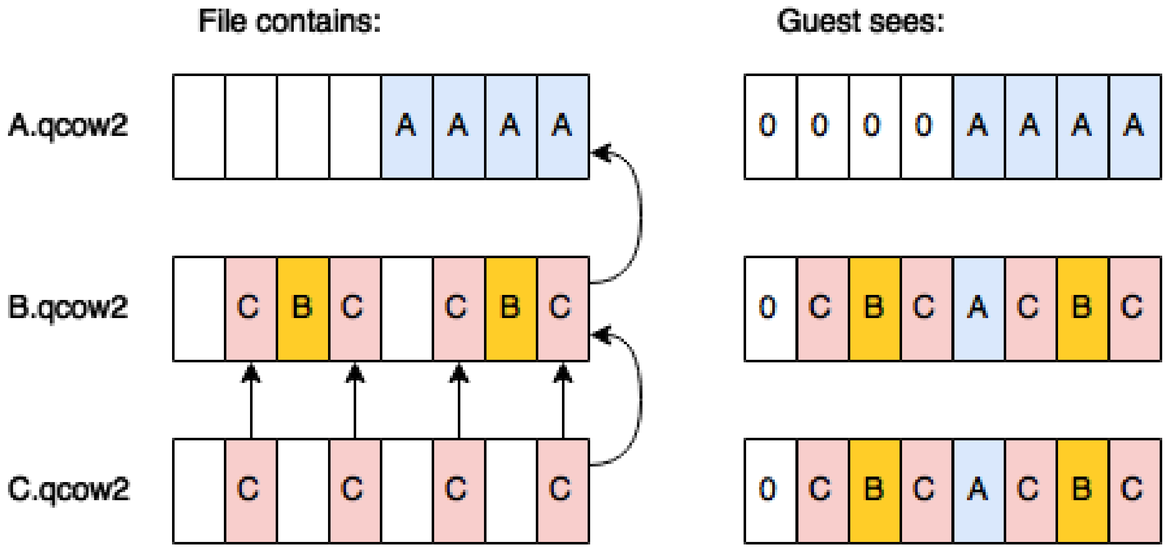
Block-commit primitive (“commit”)

- Starting with “A ← B ← C”, copy/move clusters away from top
- Additionally, rewrite backing data to drop now-redundant files
- qemu 1.3 supported intermediate commit (B into A), qemu 2.0 added active commit (C into B, C+B into A)
- Restartable, but remember caveat about editing a shared base file



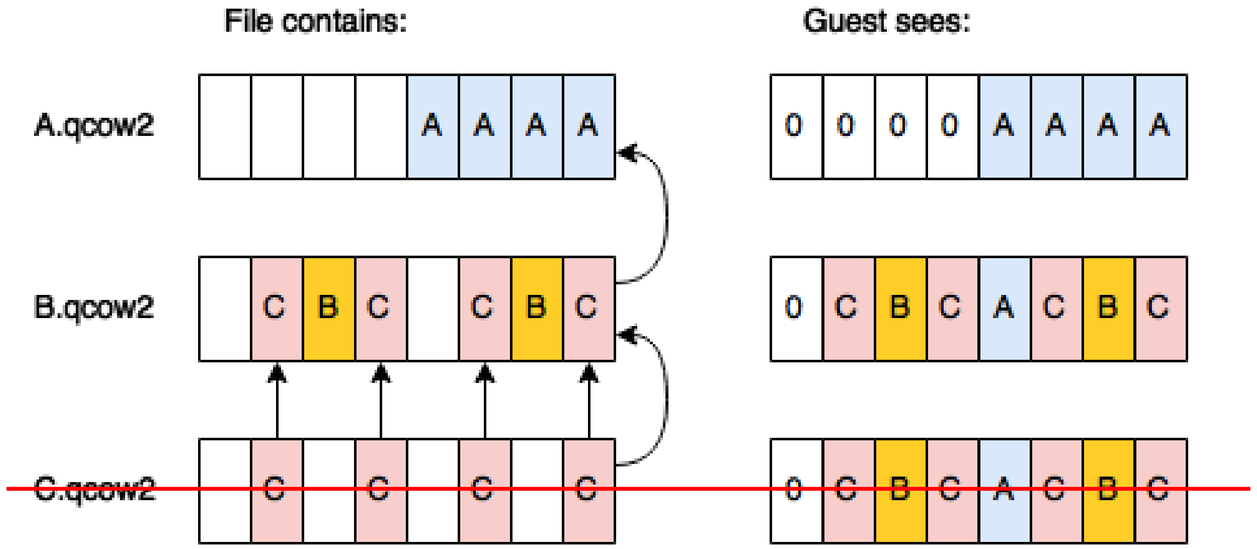
Block-commit primitive (“commit”)

- Starting with “A ← B ← C”, copy/move clusters away from top
- Additionally, rewrite backing data to drop now-redundant files
- qemu 1.3 supported intermediate commit (B into A), qemu 2.0 added active commit (C into B, C+B into A)
- Restartable, but remember caveat about editing a shared base file



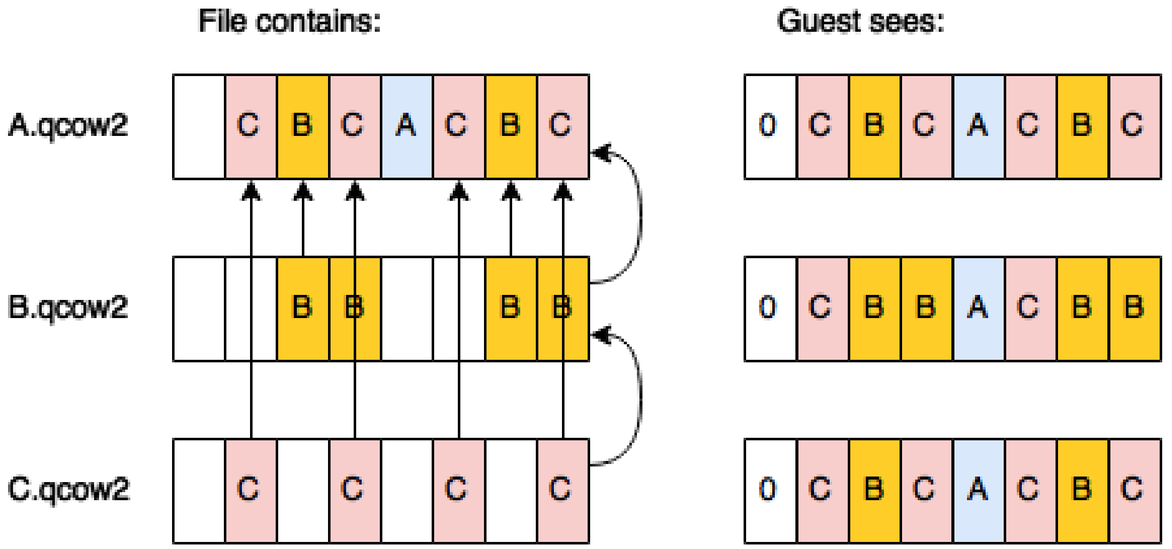
Block-commit primitive (“commit”)

- Starting with “A ← B ← C”, copy/move clusters away from top
- Additionally, rewrite backing data to drop now-redundant files
- qemu 1.3 supported intermediate commit (B into A), qemu 2.0 added active commit (C into B, C+B into A)
- Restartable, but remember caveat about editing a shared base file



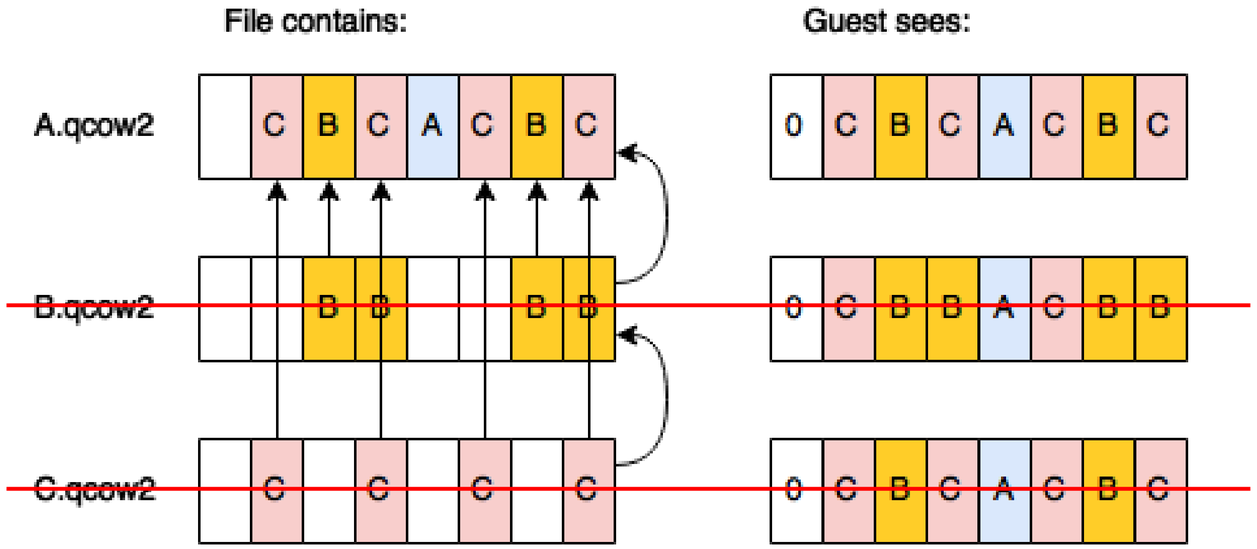
Block-commit primitive (“commit”)

- Starting with “A ← B ← C”, copy/move clusters away from top
- Additionally, rewrite backing data to drop now-redundant files
- qemu 1.3 supported intermediate commit (B into A), qemu 2.0 added active commit (C into B, C+B into A)
- Restartable, but remember caveat about editing a shared base file



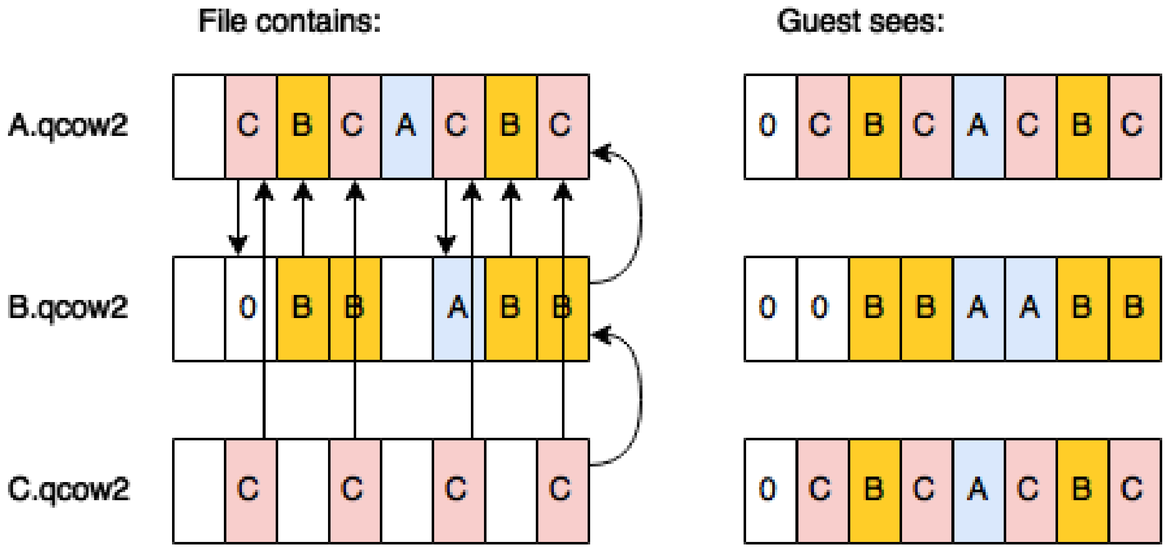
Block-commit primitive (“commit”)

- Starting with “A ← B ← C”, copy/move clusters away from top
- Additionally, rewrite backing data to drop now-redundant files
- qemu 1.3 supported intermediate commit (B into A), qemu 2.0 added active commit (C into B, C+B into A)
- Restartable, but remember caveat about editing a shared base file



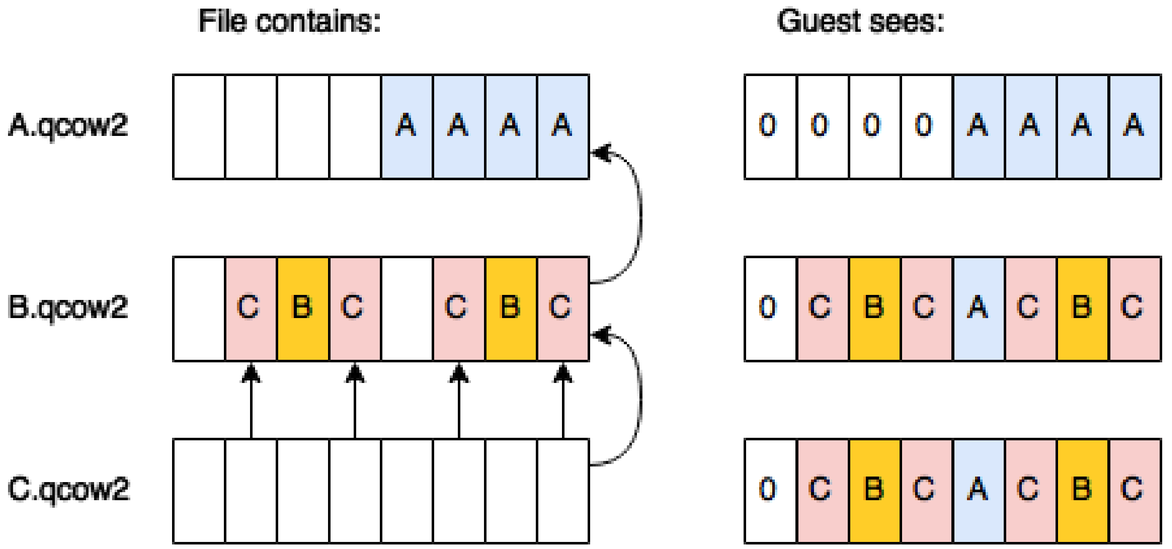
Block-commit primitive (“commit”)

- Future qemu may add additional commit mode that combines pull and commit, so that files removed from chain are still consistent
- Another future change under consideration would allow keeping the active image in chain, but clearing out clusters that are now redundant with backing file



Block-commit primitive (“commit”)

- Future qemu may add additional commit mode that combines pull and commit, so that files removed from chain are still consistent
- Another future change under consideration would allow keeping the active image in chain, but clearing out clusters that are now redundant with backing file

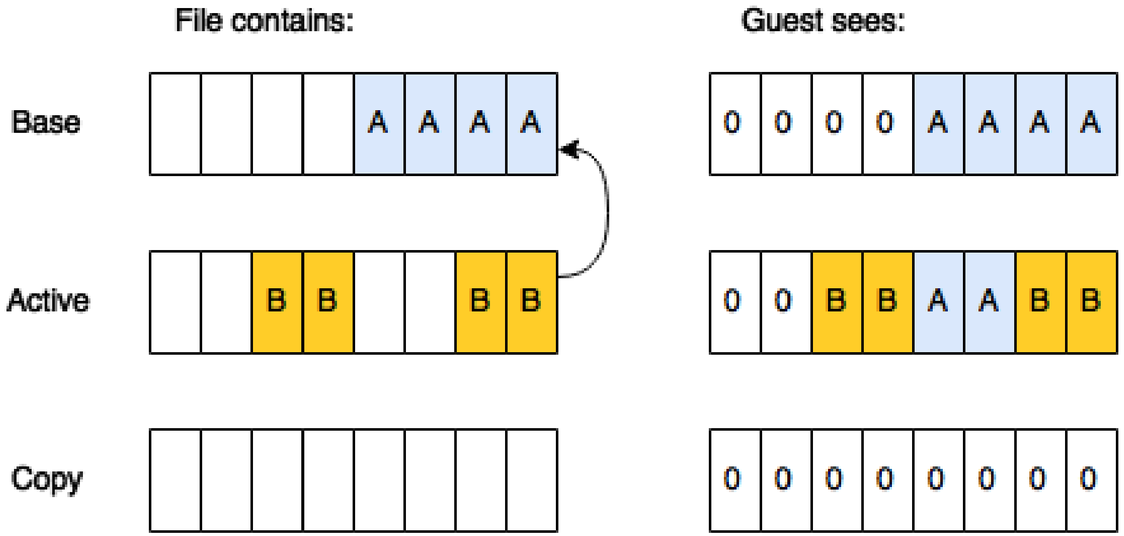


Which operation is more efficient?

- Consider removing 2nd point in time from chain “A ← B ← C ← D”
- Can be done by pulling B into C
  - Creates “A ← C' ← D”
- Can be done by committing C into B
  - Creates “A ← B' ← D”
- But one direction may have to copy more clusters than the other
- Efficiency also impacted when doing multi-step operations (deleting 2+ points in time, to shorten chain by multiple files)

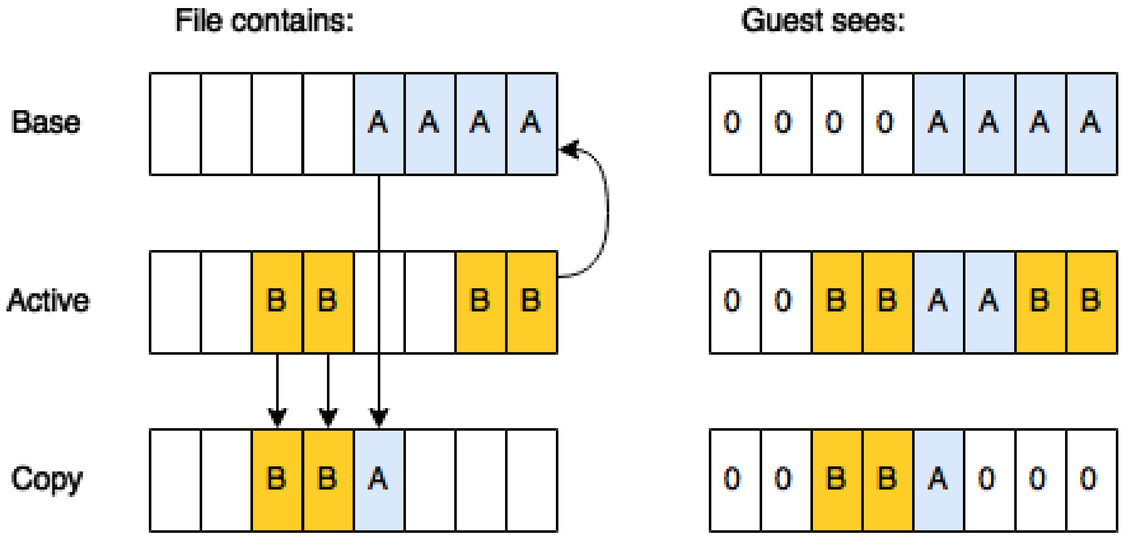
Drive-mirror primitive (“copy”)

- Copy all or part of one chain to another destination
- Destination can be pre-created, as long as the data seen by the guest is identical between source and destination when starting
  - Empty qcow2 file backed by different file but same contents
- Point in time is consistent when copy is manually ended
- Aborting early requires full restart (until persistent bitmaps)



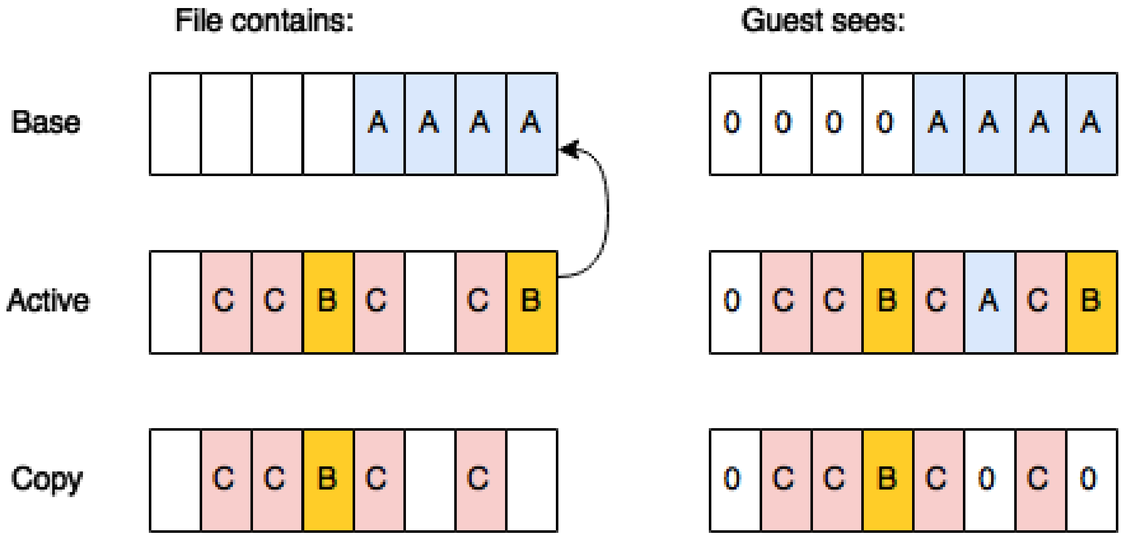
Drive-mirror primitive (“copy”)

- Copy all or part of one chain to another destination
- Destination can be pre-created, as long as the data seen by the guest is identical between source and destination when starting
  - Empty qcow2 file backed by different file but same contents
- Point in time is consistent when copy is manually ended
- Aborting early requires full restart (until persistent bitmaps)



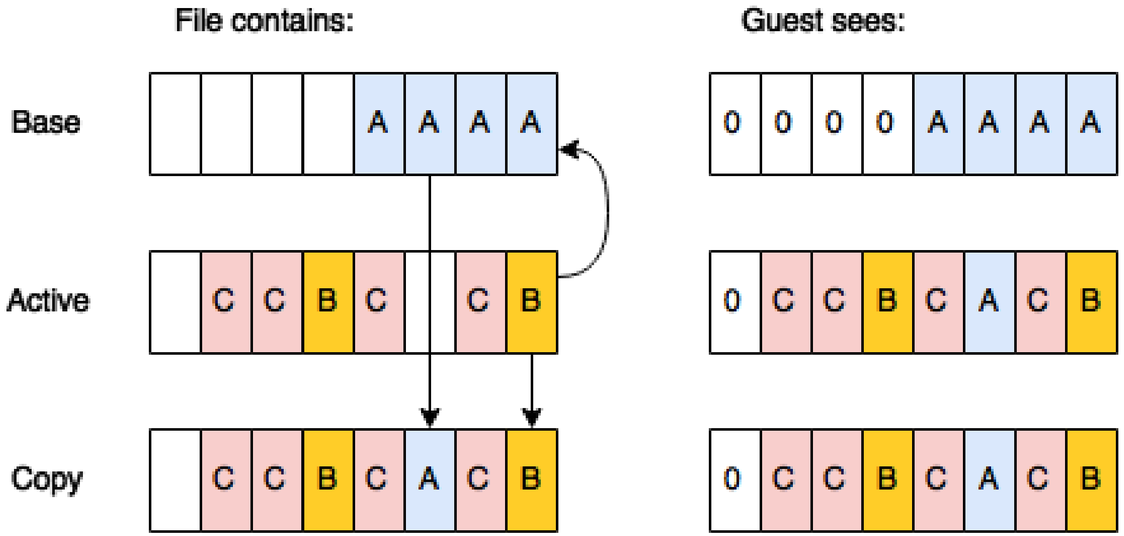
Drive-mirror primitive (“copy”)

- Copy all or part of one chain to another destination
- Destination can be pre-created, as long as the data seen by the guest is identical between source and destination when starting
  - Empty qcow2 file backed by different file but same contents
- Point in time is consistent when copy is manually ended
- Aborting early requires full restart (until persistent bitmaps)



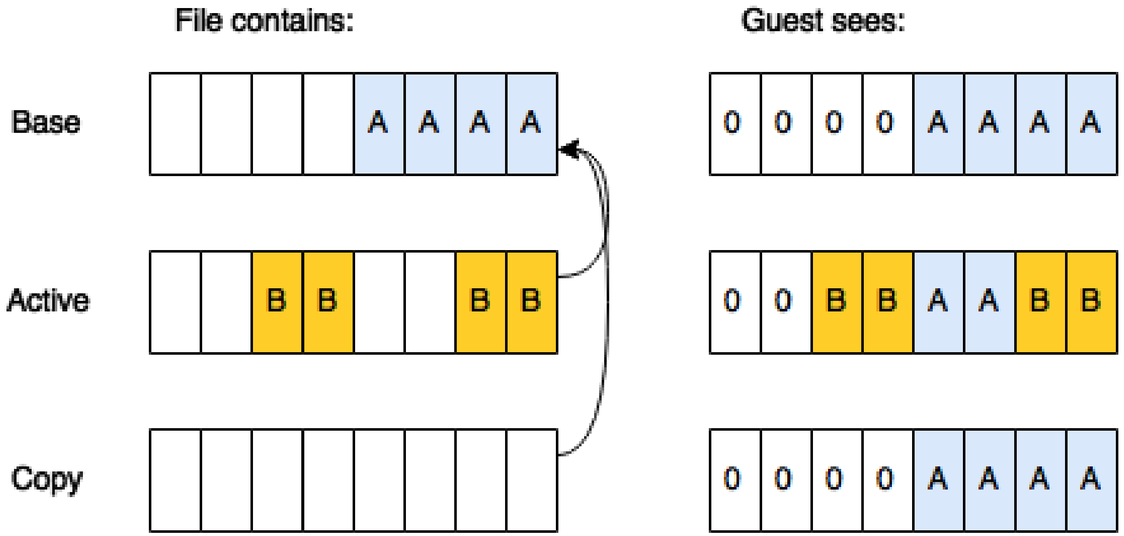
Drive-mirror primitive (“copy”)

- Copy all or part of one chain to another destination
- Destination can be pre-created, as long as the data seen by the guest is identical between source and destination when starting
  - Empty qcow2 file backed by different file but same contents
- Point in time is consistent when copy is manually ended
- Aborting early requires full restart (until persistent bitmaps)



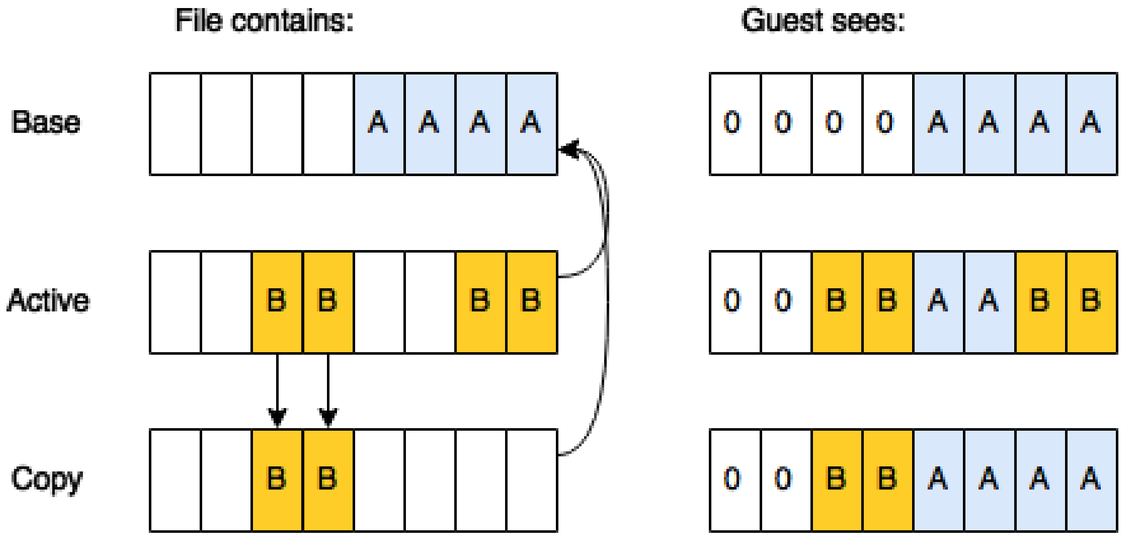
Drive-mirror primitive (“copy”)

- Copy all or part of one chain to another destination
- Destination can be pre-created, as long as the data seen by the guest is identical between source and destination when starting
  - Empty qcow2 file backed by different file but same contents
- Point in time is consistent when copy is manually ended
- Aborting early requires full restart (until persistent bitmaps)



Drive-mirror primitive (“copy”)

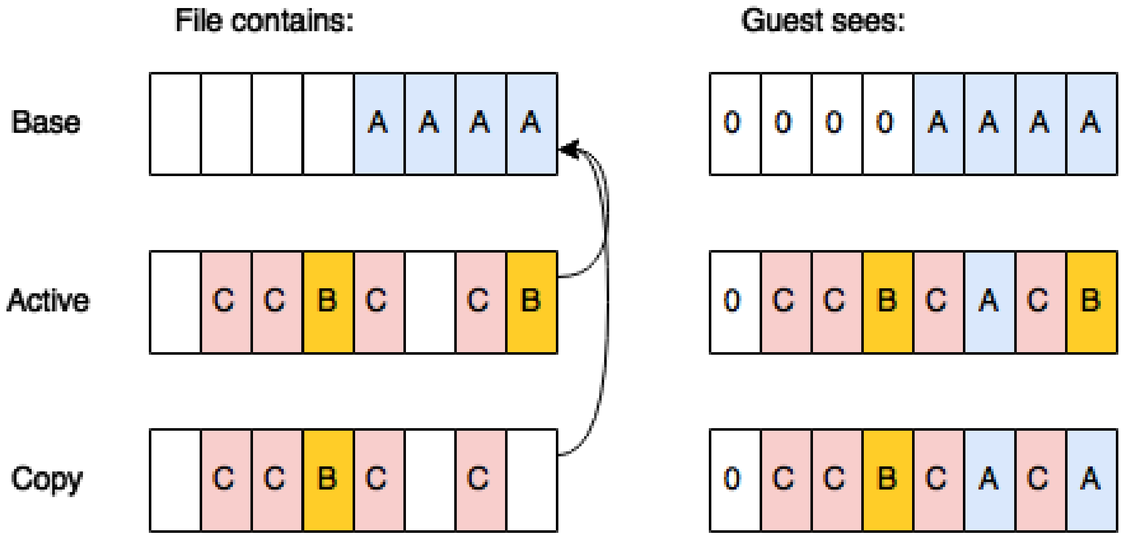
- Copy all or part of one chain to another destination
- Destination can be pre-created, as long as the data seen by the guest is identical between source and destination when starting
  - Empty qcow2 file backed by different file but same contents
- Point in time is consistent when copy is manually ended
- Aborting early requires full restart (until persistent bitmaps)



redhat.

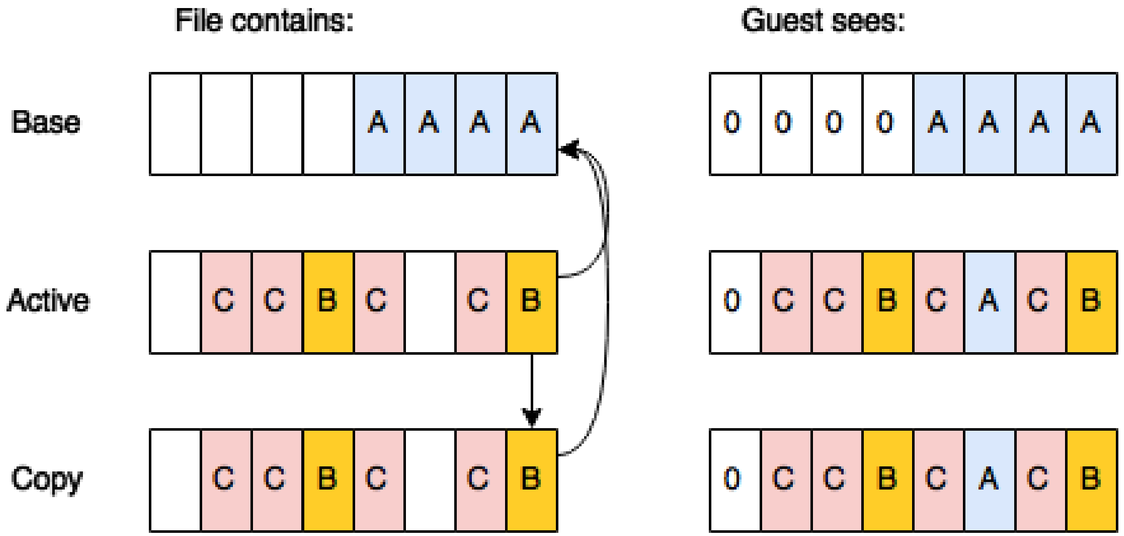
Drive-mirror primitive (“copy”)

- Copy all or part of one chain to another destination
- Destination can be pre-created, as long as the data seen by the guest is identical between source and destination when starting
  - Empty qcow2 file backed by different file but same contents
- Point in time is consistent when copy is manually ended
- Aborting early requires full restart (until persistent bitmaps)



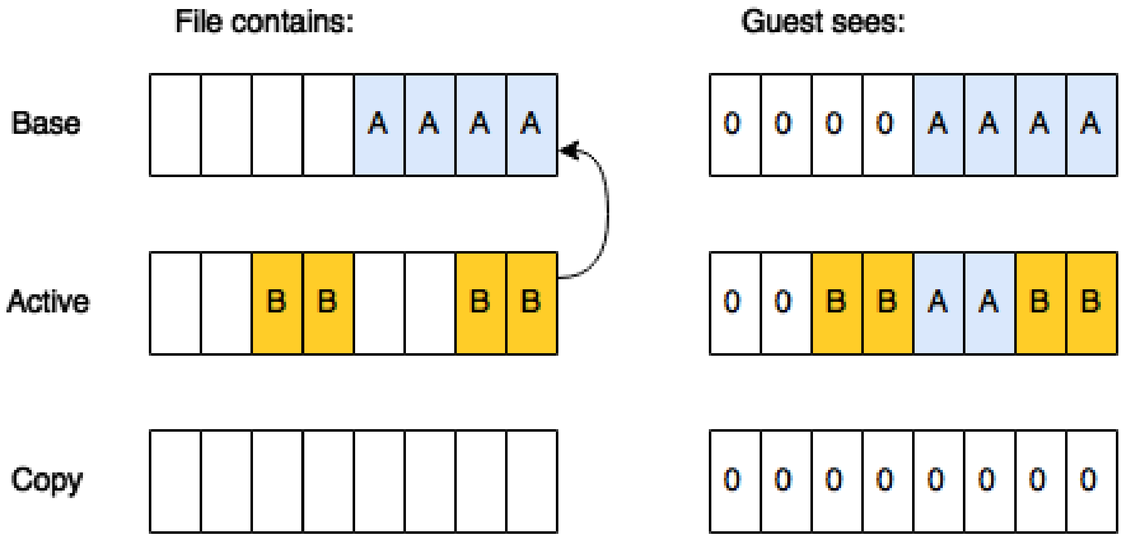
Drive-mirror primitive (“copy”)

- Copy all or part of one chain to another destination
- Destination can be pre-created, as long as the data seen by the guest is identical between source and destination when starting
  - Empty qcow2 file backed by different file but same contents
- Point in time is consistent when copy is manually ended
- Aborting early requires full restart (until persistent bitmaps)



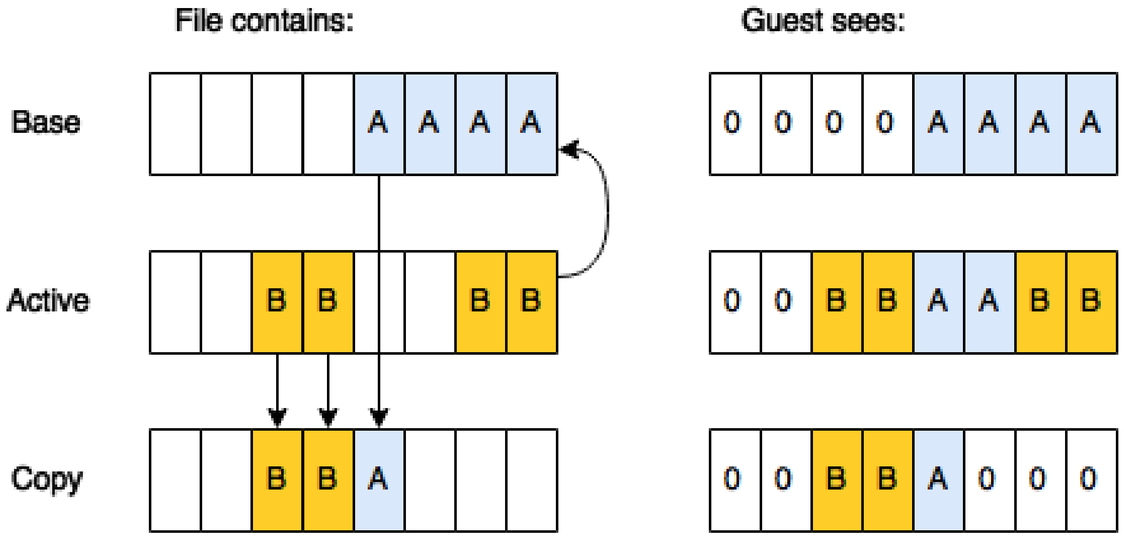
Drive-backup primitive

- Copy guest state from point in time into destination
- Any guest writes will first flush the old cluster to the destination before writing the new cluster to the source
- Meanwhile, bitmap tracks what additional clusters still need to be copied in background
- Similar to drive-mirror, but with different point in time



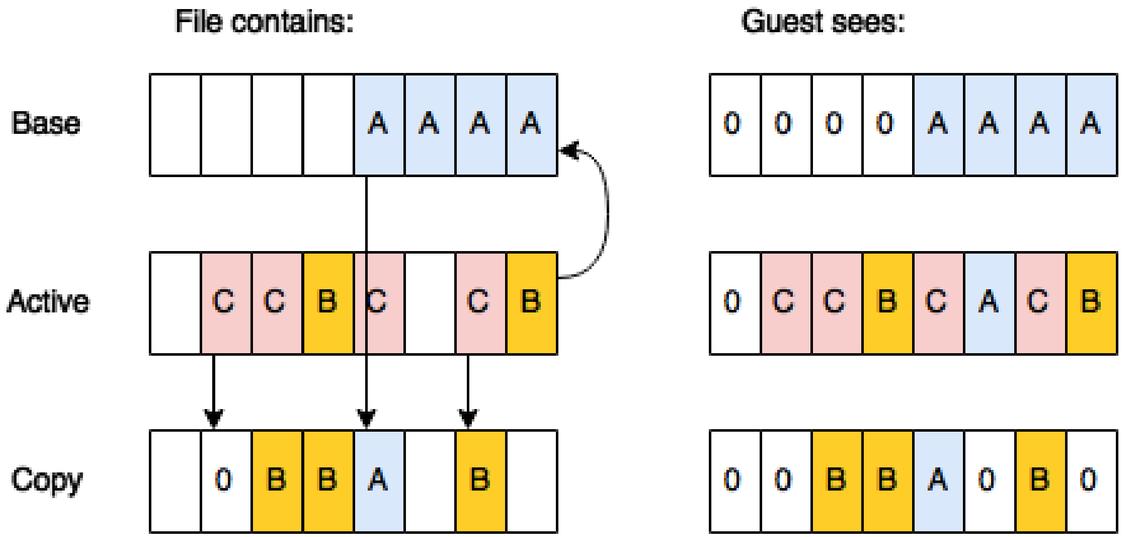
Drive-backup primitive

- Copy guest state from point in time into destination
- Any guest writes will first flush the old cluster to the destination before writing the new cluster to the source
- Meanwhile, bitmap tracks what additional clusters still need to be copied in background
- Similar to drive-mirror, but with different point in time



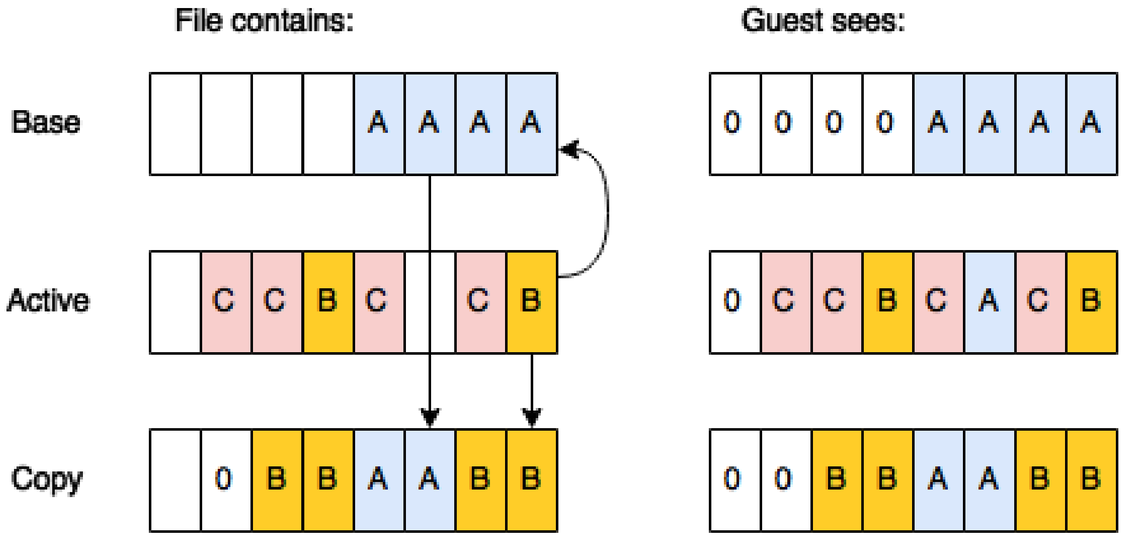
Drive-backup primitive

- Copy guest state from point in time into destination
- Any guest writes will first flush the old cluster to the destination before writing the new cluster to the source
- Meanwhile, bitmap tracks what additional clusters still need to be copied in background
- Similar to drive-mirror, but with different point in time



Drive-backup primitive

- Copy guest state from point in time into destination
- Any guest writes will first flush the old cluster to the destination before writing the new cluster to the source
- Meanwhile, bitmap tracks what additional clusters still need to be copied in background
- Similar to drive-mirror, but with different point in time



redhat.

Incremental backup

- qemu 2.5 will add ability for incremental backup via bitmaps
- User can create bitmaps at any point in guest time; each bitmap tracks guest cluster changes after that point
- While drive-mirror can only copy at backing chain boundaries, a bitmap allows extracting all clusters changed since point in time, capturing incremental state without a source backing chain
- Incremental backups can then be combined in backing chains of their own to reform full image

Part III

Libvirt control

Libvirt representation of backing chain

- virDomainGetXMLDesc() API
- `virsh dumpxml guest`
- Backing chain represented by nested children of <disk>
  - Currently only for live guests, but planned for offline guests
- Name a specific chain member by index (“vda[1]”) or filename (“/tmp/wrap.qcow2”)

```
<disk type='file' device='disk'>
 <driver name='qemu' type='qcow2'/>
 <source file='/tmp/wrap2.qcow2'/>
 <backingStore type='file' index='1'>
  <format type='qcow2'/>
  <source file='/tmp/wrap.qcow2'/>
  <backingStore type='file' index='2'>
   <format type='qcow2'/>
   <source file='/tmp/base.qcow2'/>
   <backingStore/>
  </backingStore>
 </backingStore>
 <target dev='vda' bus='virtio'/>
 ...
```

Libvirt representation of backing chain

- virDomainGetXMLDesc() API
- `virsh dumpxml guest`
- Backing chain represented by nested children of <disk>
  - Currently only for live guests, but planned for offline guests
- Name a specific chain member by index (“vda[1]”) or filename (“/tmp/wrap.qcow2”)

```
<disk type='file' device='disk'>
 <driver name='qemu' type='qcow2'/>
 <source file='/tmp/wrap2.qcow2'/>
 <backingStore type='file' index='1'>
  <format type='qcow2'/>
  <source file='/tmp/wrap.qcow2'/>
  <backingStore type='file' index='2'>
   <format type='qcow2'/>
   <source file='/tmp/base.qcow2'/>
   <backingStore/>
  </backingStore>
 </backingStore>
 <target dev='vda' bus='virtio'/>
 ...
```

Libvirt representation of backing chain

- virDomainGetXMLDesc() API
- `virsh dumpxml guest`
- Backing chain represented by nested children of <disk>
  - Currently only for live guests, but planned for offline guests
- Name a specific chain member by index (“vda[1]”) or filename (“/tmp/wrap.qcow2”)

```
<disk type='file' device='disk'>
 <driver name='qemu' type='qcow2'/>
 <source file='/tmp/wrap2.qcow2'/>
 <backingStore type='file' index='1'>
  <format type='qcow2'/>
  <source file='/tmp/wrap.qcow2'/>
  <backingStore type='file' index='2'>
   <format type='qcow2'/>
   <source file='/tmp/base.qcow2'/>
   <backingStore/>
  </backingStore>
 </backingStore>
 <target dev='vda' bus='virtio'/>
 ...
```

Creating an external snapshot

- virDomainSnapshotCreateXML() API
- `virsh snapshot-create domain description.xml`
- `virsh snapshot-create-as domain --disk-only \`
  `--diskspec vda,file=/path/to/wrapper.qcow2`
- Maps to qemu blockdev-snapshot-sync, also manages offline chain creation through qemu-img
- Often used with additional flags:
  - --no-metadata: cause only side effect of backing chain growth
  - --quiesce: freeze guest I/O, but requires guest agent

Performing block pull

- virDomainBlockRebase() API
- `virsh blockpull domain vda --wait --verbose`
- Mapped to qemu block-stream, with current limitation of only pulling into active layer
- When qemu 2.5 adds intermediate streaming, syntax will be:
  - `virsh blockpull domain "vda[1]" --base "vda[3]"`

Performing block commit

- virDomainBlockCommit() API, plus virDomainBlockJobAbort() for active jobs
- `virsh blockcommit domain vda --top "vda[1]"`
- `virsh blockjob domain vda`
- `virsh blockcommit domain vda --shallow \`
  `--pivot --verbose --timeout 60`
- May gain additional flags if qemu block-commit adds features

Performing block copy

- virDomainBlockCopy()/virDomainBlockJobAbort() APIs
- `virsh blockcopy domain vda /path/to/dest --pivot`
- Currently requires transient domain
  - Plan to relax that with qemu 2.5 persistent bitmap support
- Currently captures point in time at end of job (drive-mirror)
  - May later add flag for start of job semantics (drive-backup)
- Plan to add --quiesce flag to job abort, like in snapshot creation, instead of having to manually use domfsfreeze/domfsthaw

Piecing it all together: efficient live backup

- Goal: create (potentially bootable) backup of live guest disk state

/my/base ← /my/image

Piecing it all together: efficient live backup

- Goal: create (potentially bootable) backup of live guest disk state

```
$ virsh snapshot-create-as domain tmp \
   --no-metadata --disk-only
```

/my/base ← /my/image ← /my/image.tmp

Piecing it all together: efficient live backup

- Goal: create (potentially bootable) backup of live guest disk state

```
$ virsh snapshot-create-as domain tmp \
   --no-metadata --disk-only --quiesce
```

/my/base ← /my/image ← /my/image.tmp

Piecing it all together: efficient live backup

- Goal: create (potentially bootable) backup of live guest disk state

```
$ virsh snapshot-create-as domain tmp \
   --no-metadata --disk-only --quiesce
$ cp --reflink=always /my/image /backup/image
```

/my/base ← /my/image ← /my/image.tmp

/my/base ← /backup/image

Piecing it all together: efficient live backup

- Goal: create (potentially bootable) backup of live guest disk state

```
$ virsh snapshot-create-as domain tmp \
   --no-metadata --disk-only --quiesce
$ cp --reflink=always /my/image /backup/image
$ virsh blockcommit domain vda --shallow \
   --pivot --verbose
```

/my/base ← /my/image

/my/base ← /backup/image

Piecing it all together: efficient live backup

- Goal: create (potentially bootable) backup of live guest disk state

```
$ virsh snapshot-create-as domain tmp \
   --no-metadata --disk-only --quiesce
$ cp --reflink=always /my/image /backup/image
$ virsh blockcommit domain vda --shallow \
   --pivot --verbose
$ rm /my/image.tmp
```

/my/base ← /my/image

/my/base ← /backup/image

Piecing it all together: efficient live backup

- Goal: create (potentially bootable) backup of live guest disk state

```
$ virsh snapshot-create-as domain tmp \
   --no-metadata --disk-only --quiesce
$ cp --reflink=always /my/image /backup/image
$ virsh blockcommit domain vda --shallow \
   --pivot --verbose
$ rm /my/image.tmp
```

- No guest downtime, and with a fast storage array command, the delta contained in temporary chain wrapper is small enough for entire operation to take less than a second

Piecing it all together: revert to snapshot

- Goal: roll back to disk state in an external snapshot

/my/base ← /my/experiment

```
<disk type='file' device='disk'>
 <driver name='qemu' type='qcow2'/>
 <source file='/my/experiment'/>
...
```

Piecing it all together: revert to snapshot

- Goal: roll back to disk state in an external snapshot

```
$ virsh destroy domain
```

/my/base ← /my/experiment

```
<disk type='file' device='disk'>
 <driver name='qemu' type='qcow2'/>
 <source file='/my/experiment'/>
...
```

Piecing it all together: revert to snapshot

- Goal: roll back to disk state in an external snapshot

```
$ virsh destroy domain
$ virsh edit domain # update <disk> details
```

/my/base ← /my/experiment

```
<disk type='file' device='disk'>
 <driver name='qemu' type='qcow2'/>
 <source file='/my/base'/>
...
```

Piecing it all together: revert to snapshot

- Goal: roll back to disk state in an external snapshot

```
$ virsh destroy domain
$ virsh edit domain # update <disk> details
$ rm /my/experiment
```

/my/base

```
<disk type='file' device='disk'>
 <driver name='qemu' type='qcow2'/>
 <source file='/my/base'/>
...
```

Piecing it all together: revert to snapshot

- Goal: roll back to disk state in an external snapshot

```
$ virsh destroy domain
$ virsh edit domain # update <disk> details
$ rm /my/experiment
$ virsh start domain
```

/my/base

```
<disk type='file' device='disk'>
 <driver name='qemu' type='qcow2'/>
 <source file='/my/base'/>
...
```

Piecing it all together: revert to snapshot

- Goal: roll back to disk state in an external snapshot

```
$ virsh destroy domain
$ virsh edit domain # update <disk> details
$ rm /my/experiment
$ virsh start domain
```

- If rest of chain must be kept consistent, use copies or create additional wrappers with qemu-img to avoid corrupting base
- If rest of chain is not needed, be sure to delete files that are invalidated after reverting

Piecing it all together: live storage migration

- Goal: rebase storage chain from network to local storage

/nfs/image

Piecing it all together: live storage migration

- Goal: rebase storage chain from network to local storage

```
$ virsh snapshot-create-as domain tmp \
   --no-metadata --disk-only
```

/nfs/image ← /nfs/image.tmp

Piecing it all together: live storage migration

- Goal: rebase storage chain from network to local storage

```
$ virsh snapshot-create-as domain tmp \
   --no-metadata --disk-only
$ cp /nfs/image /local/image
```

/nfs/image ← /nfs/image.tmp

/local/image

Piecing it all together: live storage migration

- Goal: rebase storage chain from network to local storage

```
$ virsh snapshot-create-as domain tmp \
   --no-metadata --disk-only
$ cp /nfs/image /local/image
$ qemu-img create -f qcow2 -b /local/image \
   -F qcow2 /local/wrap
```

/nfs/image ← /nfs/image.tmp

/local/image ← /local/wrap

Piecing it all together: live storage migration

- Goal: rebase storage chain from network to local storage

```
$ virsh snapshot-create-as domain tmp \
   --no-metadata --disk-only
$ cp /nfs/image /local/image
$ qemu-img create -f qcow2 -b /local/image \
   -F qcow2 /local/wrap
$ virsh undefine domain
...
```

/nfs/image ← /nfs/image.tmp

/local/image ← /local/wrap

Piecing it all together: live storage migration

- Goal: rebase storage chain from network to local storage

```
$ virsh blockcopy domain vda /local/wrap \
   --shallow --pivot --verbose --reuse-external
```

/nfs/image ← /nfs/image.tmp

/local/image ← /local/wrap

Piecing it all together: live storage migration

- Goal: rebase storage chain from network to local storage

```
$ virsh blockcopy domain vda /local/wrap \
   --shallow --pivot --verbose --reuse-external
$ virsh dumpxml domain > file.xml
$ virsh define file.xml
```

/nfs/image ← /nfs/image.tmp

/local/image ← /local/wrap

Piecing it all together: live storage migration

- Goal: rebase storage chain from network to local storage

```
$ virsh blockcopy domain vda /local/wrap \
   --shallow --pivot --verbose --reuse-external
$ virsh dumpxml domain > file.xml
$ virsh define file.xml
$ virsh blockcommit domain vda --shallow \
   --pivot --verbose
```

/nfs/image ← /nfs/image.tmp

/local/image

Piecing it all together: live storage migration

- Goal: rebase storage chain from network to local storage

```
$ virsh blockcopy domain vda /local/wrap \
   --shallow --pivot --verbose --reuse-external
$ virsh dumpxml domain > file.xml
$ virsh define file.xml
$ virsh blockcommit domain vda --shallow \
   --pivot --verbose
$ rm file.xml /local/wrap /nfs/image.tmp
```

/nfs/image

/local/image

Piecing it all together: live storage migration

- Goal: rebase storage chain from network to local storage

```
$ virsh blockcopy domain vda /local/wrap \
   --shallow --pivot --verbose --reuse-external
$ virsh dumpxml domain > file.xml
$ virsh define file.xml
$ virsh blockcommit domain vda --shallow \
   --pivot --verbose
$ rm file.xml /local/wrap /nfs/image.tmp
```

- The undefine/dumpxml/define steps will drop once libvirt can use persistent bitmaps to allow copy with non-transient domains

Future work

- Libvirt support of offline chain management
- Libvirt support of revert to external snapshot
- Qemu 2.5 additions, and adding libvirt support:
  - Intermediate streaming
  - Incremental backup
  - Use persistent bitmap
- Libvirt support to expose mapping information, or at a minimum whether pull or commit would move less data
- Patches welcome!

Questions?