



Live block device operations in QEMU

Paolo Bonzini

Red Hat

Yokohama, June 2012

Outline

- What is QEMU?
- The QEMU block layer
- Live block operations
- Q&A



What is QEMU?

- “A FAST! processor emulator”
- Started by Fabrice Bellard as Linux-on-Linux userspace emulation
- Later extended to support system emulation



QEMU supports multiple accelerators

- Two emulate the whole machine
 - TCG (“Tiny Code Generator”) supports emulating any instruction set
 - KVM provides transparent acceleration of the native ISA
- Two emulate devices only
 - Xen (can be fully-virtualized or paravirtualized)
 - QTest (simple ASCII protocol for device unit testing)
- qemu-kvm could be merged completely in QEMU 1.2



Huge amount of shared code!

- Device backends
 - Block device
 - Character device
 - Filesystem device
 - Network interface
 - Audio device
 - UI backend
- Management interface
- Device models
 - PCI/PCIe
 - IDE/AHCI
 - SCSI
 - USB (1.1/2.0/3.0)
 - virtio
 - I²C
 - ISA
 - ...



The QEMU storage stack

- The formats:
 - raw, qcow2, QED
 - ... and many others
- The backends:
 - Flat file
 - Block device
 - usermode iSCSI & NBD
 - curl (HTTP etc.)
 - Sheepdog, ceph
- The clients:
 - Image manipulation tool (qemu-img)
 - NBD server (qemu-nbd)
 - Debugging/unit-testing tool (qemu-io)
 - Guest storage devices (floppy, CD-ROM, USB, IDE, SCSI)



Block device operations

- Create snapshot: `qemu-img create`
- Rebase: `qemu-img rebase`
- Copy: `cp`, `dd`, `qemu-img convert`
- Serve as NBD: `qemu-nbd`
- **Only possible when the VM is offline**



The QEMU storage stack

- The formats:
 - raw, qcow2, QED
 - ... and many others
- The backends:
 - Flat file
 - Block device
 - usermode iSCSI & NBD
 - curl (HTTP etc.)
 - Sheepdog, ceph
- The clients:
 - Image manipulation tool (qemu-img)
 - NBD server (qemu-nbd)
 - Debugging/unit-testing tool (qemu-io)
 - Guest storage devices (floppy, CD-ROM, USB, IDE, SCSI)
 - **QEMU live block operations**



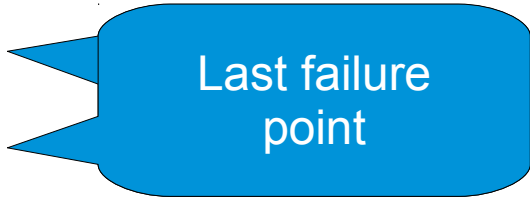
Live block operations

- Should not interfere with VM operation
 - Pauses must be minimized
 - Errors must be recoverable
- Long-running operations must be asynchronous
 - Asynchronous I/O runs outside the QEMU global lock
 - Monitor commands allow polling and cancellation



Starting simple: live snapshots

- Some image formats support stacking image files with copy-on-write
- Live snapshotting adds a new image to the stack
 1. Flush all pending I/O from the guest
 2. Create a new image file and open it
 3. Set the current image as the backing file
 4. Redirect guest I/O to new file
- Multiple snapshots can be part of a single transaction



Last failure point



Copy-on-read

- CoW backing files can be on remote storage
- Move data closer to the VM to avoid expensive access
- **All reads from a cluster wait for it to be copied**

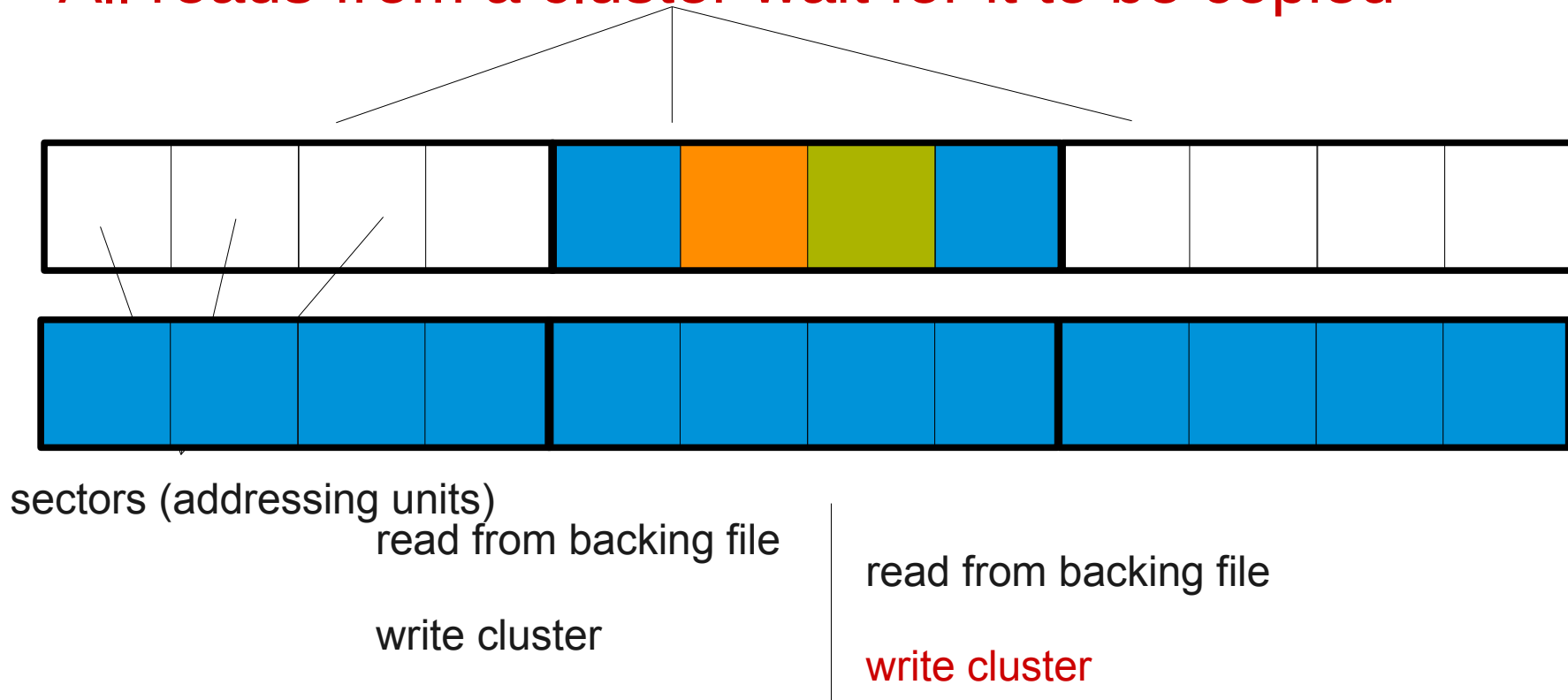


Image streaming

- Collapse all the stack of images into the topmost one
- Just perform copy-on-read from the beginning to the end of the file
 - Unallocated areas in the backing file are skipped
 - At the end, drop the backing file
- Lets you remove one or more images from the stack



Image migration

- Many use cases
 - Oh no, this disk says it's about to fail!
 - Consolidation
 - Defragmentation
- Two phases
 - Synchronization (copy everything)
 - **Steady state (copy what's changed)**



Image migration: how

Active

Perform each operation on both disks

“I need to write this!”

Passive

Track changes to the source, reproduce them

“Uh, someone wrote here...”

Synchronous

Operations are completed once they have reached both disks

Asynchronous

Operations are completed even before they reach the destination



Image migration: how

- **Active/synchronous**
Simplest handling of host crashes
Least robust in case the migration target has problems
- **Active/asynchronous**
Requires storing all I/O operations
- **Passive/asynchronous**
Most robust, good performance
Tricky to guarantee correctness of host crashes



Image migration: how

Active

Perform each operation on both disks

“I need to write this!”

Synchronous

Operations are completed once they have reached both disks

Passive

Track changes to the source, reproduce them

“Uh, someone wrote here...”

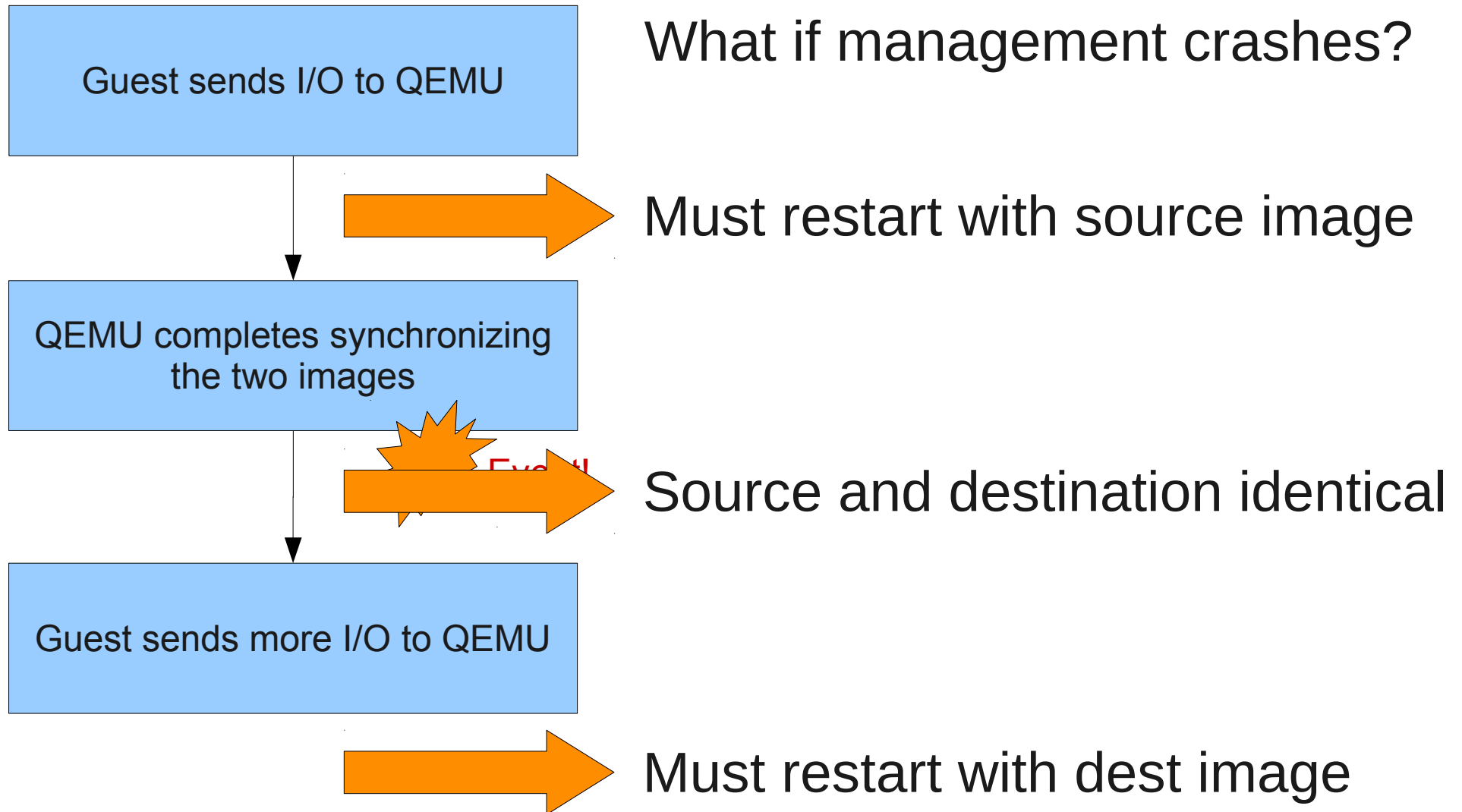
Asynchronous

Operations are completed even before they reach the destination

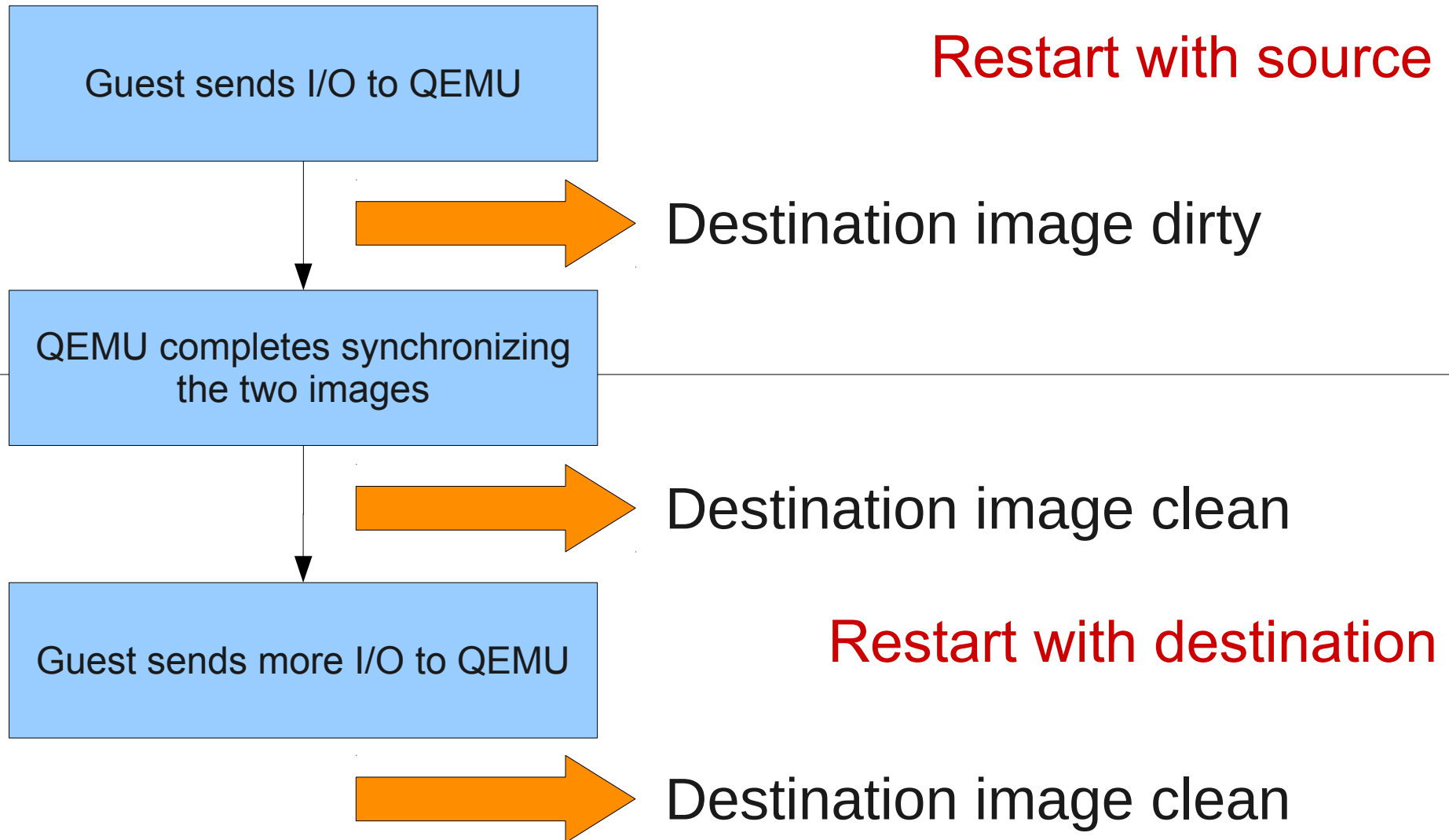
Other implementations possible with the same API



Image migration: switching



“Dirtiness” of destination



Solution

- Add a persistent dirty bitmap
 - Initialize dirty bitmap with all allocated sectors
 - After a crash, if “all zeros” start QEMU with the destination image
 - Otherwise, start with the source image
- Need to be careful!
 - Write new “1” bits synchronously upon guest flushes
 - Write “0” bits asynchronously
 - msync improvements sent to LKML :)



Other dirty bitmap uses

- What if the guest exits?
 - Management can pick up the job and complete it offline
- What if the copy is paused?
 - Dirty sectors will accumulate and will be cleared later

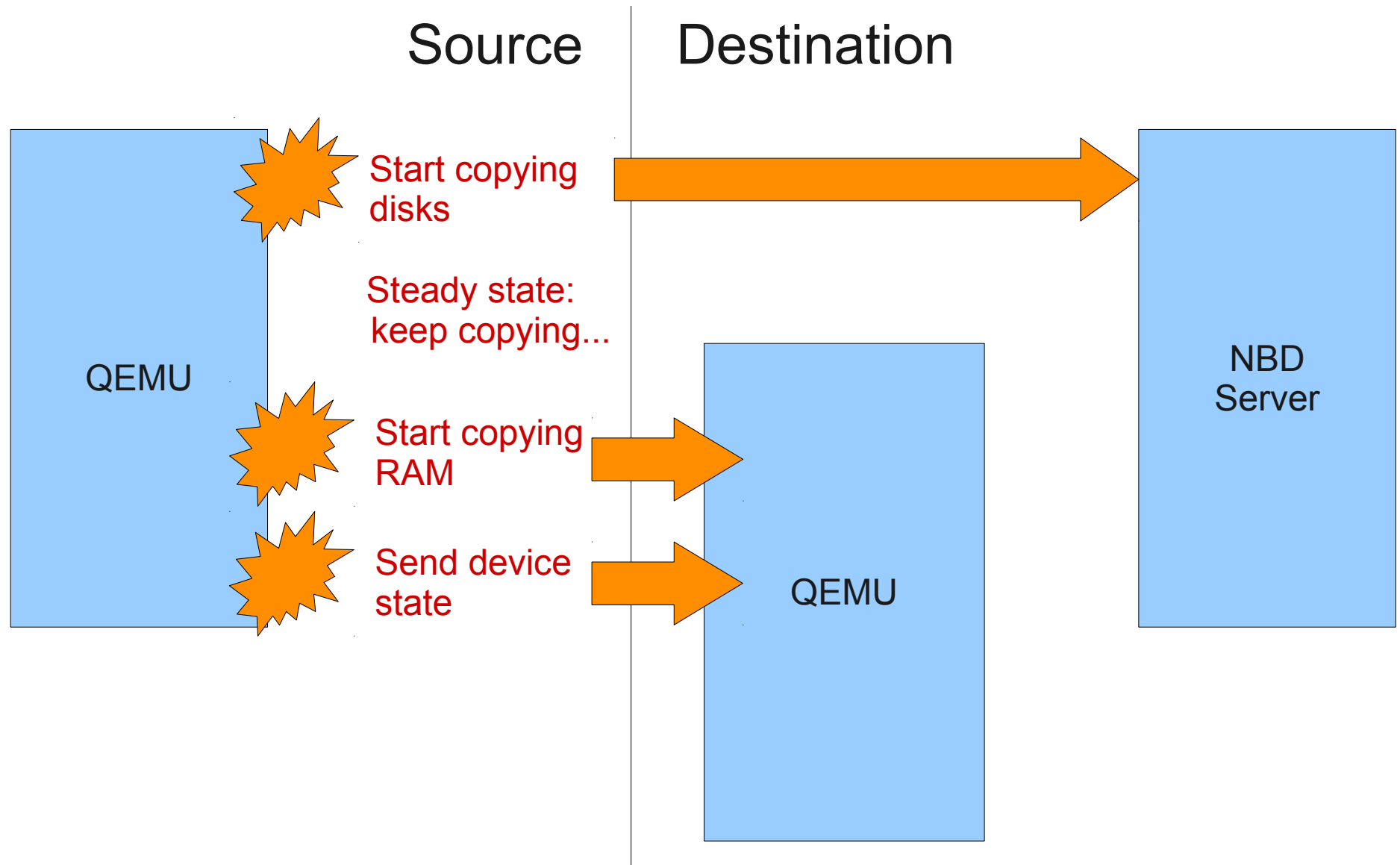


Migration without shared storage

- What if the target of the copy is an NBD server?
- I can migrate entire VMs to other hosts, including disks
 - That will take a while...
- I can replicate disks elsewhere
 - High availability
 - Continuous snapshotting



Migration without shared storage



Status

- Snapshotting
 - Added in 0.14
 - Atomic snapshot of multiple disks since 1.1
- Copy-on-read, streaming
 - New in 1.1
- Image migration
 - Will be in 1.2
 - Dirty bitmap too, hopefully



Thanks!

- Questions?

