Towards Multi-threaded Device Emulation in QEMU

Stefan Hajnoczi

Red Hat

KVM Forum 2014

Agenda

IOThread and AioContext

- Managing event loop threads
- How the block layer was made IOThread-friendly

Memory access in multi-threaded device emulation

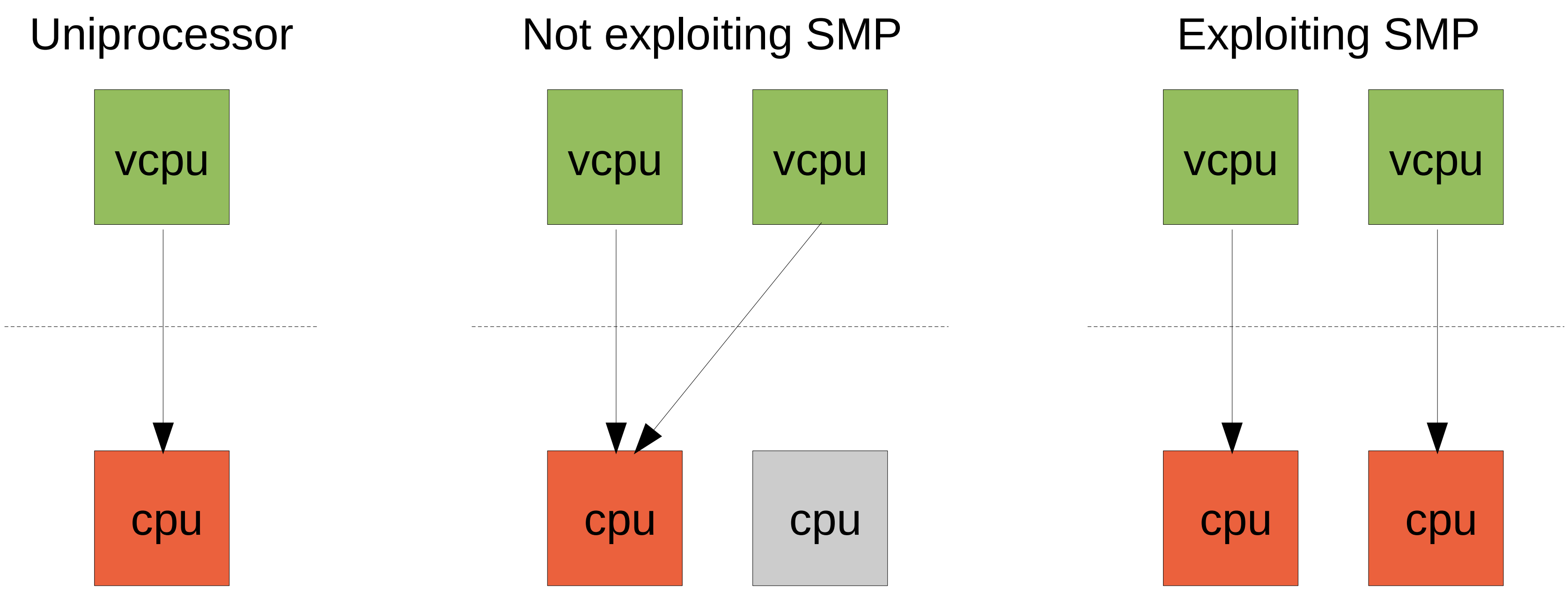
- Dirty memory bitmap
- Recursive memory dispatch

Multiqueue block layer

- Approaches

Why does multi-threading matter?

Symmetric multiprocessing (SMP) changes the game:



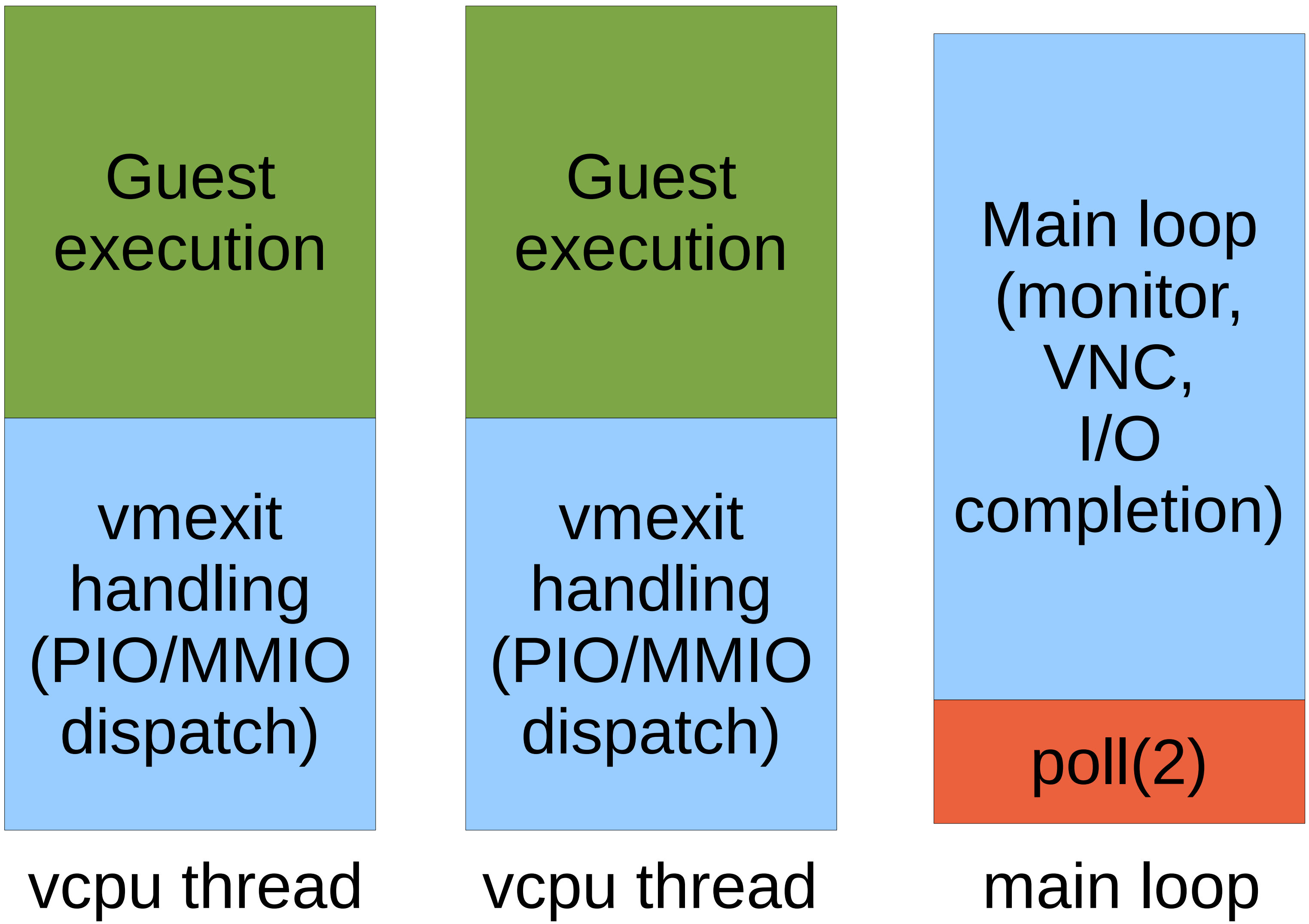
Must exploit SMP to fully utilize hardware

QEMU was not designed with SMP in mind

How can we exploit SMP in QEMU?

KVM architecture

QEMU code generally runs under global mutex



State of multi-threading in QEMU 2.1

Component	Status
TCG vcpu	No
KVM vcpu	Yes
virtio-blk dataplane	Yes but clean-up remaining
Migration (live phase)	Yes
virtio-scsi dataplane	In development
Other device emulation	No

Is incremental SMP support a good strategy?

Making everything thread-safe and SMP-friendly:

- Very invasive
- No performance improvement in many places
- Complicates code, single-threaded is simpler

Use multi-threading where there is a real benefit:

- Leave code that is not performance-sensitive

Current work

My focus has been virtio-blk dataplane since 2011

dataplane moves virtio-blk device emulation into dedicated thread

- I/O requests processed outside QEMU global mutex
- Benefits SMP high-iops workloads

dataplane is driving multi-threading work in QEMU:

- Guest memory access
- IOThreads and AioContext

Managing device emulation threads

IOThread is an event loop thread

- virtio-blk devices can be assigned to an IOThread

qemu -object iothread,id=iothread0

QMP command “query-iothreads”

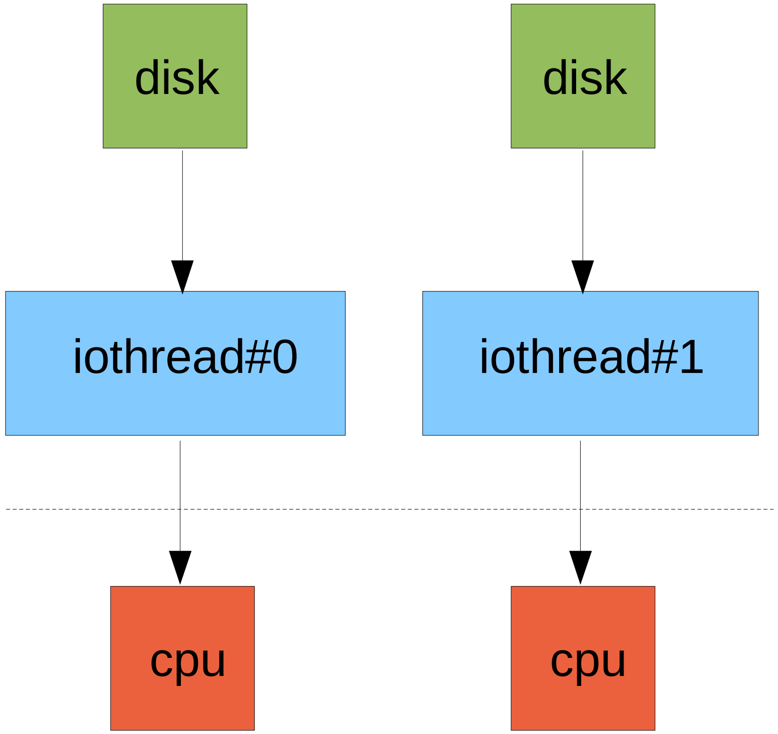
- Returns: [{ “id”: “iothread0”, “thread-id”: 3134 }]
- Use “thread-id” for host CPU affinity

Supported in libvirt 1.2.8+

IOThread CPU affinity x-data-plane=on 1:1 mode

Classic -device virtio-blk-pci,x-data-plane=on:

- 1 IOThread per device
- Makes sense with fewer devices than host CPUs



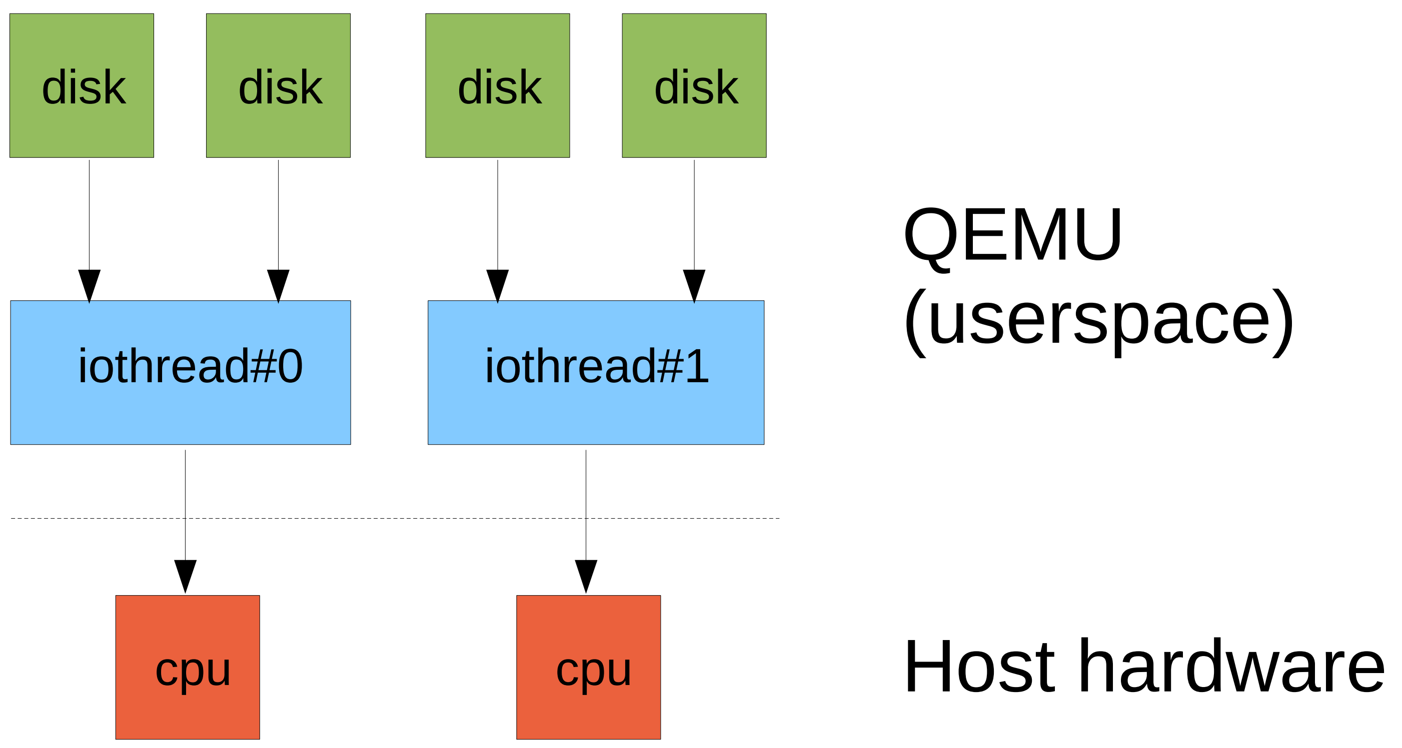
QEMU (userspace)

Host hardware

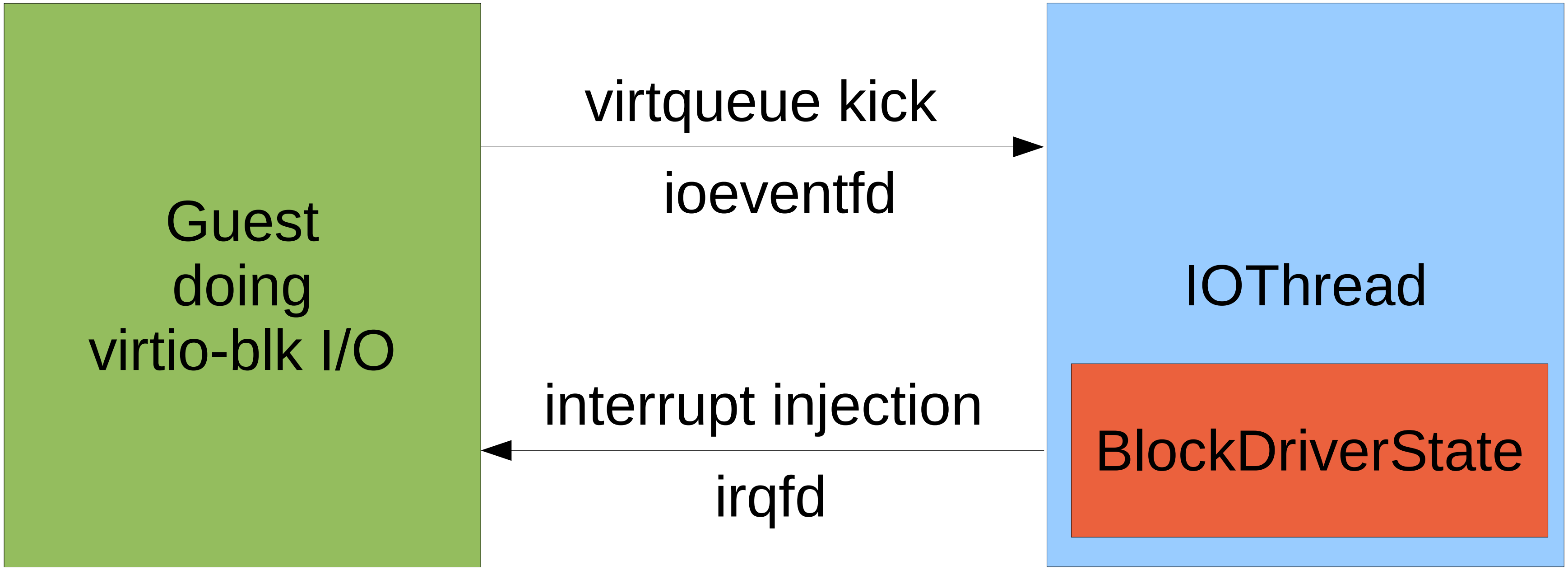
IOThread CPU affinity N:M mapping

Host CPU affinity N:M mapping:

- 1 IOThread per host CPU
- Distribute devices across IOThreads



virtio-blk dataplane with IOThread



QEMU main loop and global mutex are not involved

QEMU block layer used inside of IOThread

IOThread runs an AioContext event loop

AioContext event loop

AioContext is an event loop

- File descriptor monitoring
- Timers and BHs

IOThread runs an AioContext event loop:

```
while (running) {
    aio_context_acquire(context);
    aio_poll(context); /* can block waiting for events */
    aio_context_release(context);
}
```

AioContext acquire/release

Another thread may need to access shared resource

```
aio_context_acquire(context);
...access shared resource...
aio_context_release(context);
```

If AioContext is in use by IOThread loop, the loop is automatically “kicked” so we can acquire

BlockDriverState is protected by AioContext acquire

How block layer was made IOThread-compatible

Previously: Block layer ran under global mutex

Now: BlockDriverState is bound to AioContext

- bdrv_set_aio_context(bs, new_aio_context)

Rules:

- Acquire AioContext before accessing bs
- Creating/deleting BlockDriverState must be done from main loop – bdrv_states protected by global mutex
- Lock ordering – only main loop may acquire AioContext arbitrarily

Attaching and detaching from AioContext

BlockDriverState can be migrated to a new AioContext at run-time

- e.g. dataplane mode is enabled/disabled

Typical .bdrv_aio_context_attach/detach(...):

- Add/remove file descriptors from AioContext
- Add/remove timers from AioContext
- Ensure that BHs are not pending in old AioContext

Using the same approach for other subsystems

Do you need to put I/O into an IOThread, but allow main loop to access the resource safely?

Use AioContext!

Thread-safe guest memory access

Emulated devices use DMA or guest memory access

Hence guest memory access must be thread-safe

Thread-safe today:

- Memory regions can be acquired/released
- Guest RAM can be mapped

Missing today:

- Dirty bitmap (for live migration)
- Recursive memory dispatch (i.e. device-to-device)

Dirty bitmap for guest memory

Live migration tracks dirty guest memory pages

Devices must mark written pages dirty

Live migration will transfer them to the destination host

ram_list.dirty_memory[DIRTY_MEMORY_MIGRATION]

Currently access is protected by global mutex

Making dirty bitmap thread-safe

(Optimistic on this but have not written code yet)

Convert bitmap ops to atomics:

set_dirty() -> atomic_or(&bitmap[i], val)

test_and_clear_dirty() -> atomic_xchg(&bitmap[i], 0)

get_dirty()* -> atomic_mb_read(&bitmap[i])

* This may be used in non-atomic fashion by caller, may need to convert to fetch_and_set_dirty() or test_and_set_dirty()

Recursive memory dispatch

Problem: Address space can contain memory-mapped I/O registers, so device DMA can dispatch to a different device!

(Not all architectures may support this but some do)

Example: SCSI disk READ command DMAs to graphics card PCI BAR

...and

- SCSI disk is attached to iothread#0
- Graphics card is attached to iothread#1

Lock ordering problem for recursive dispatch

Currently in SCSI disk's iothread#0 context,

need to access graphics card's iothread#1 context.

If multiple devices do this at the same time there are lock ordering problems -> deadlock!

Solution: Release iothread#0 before acquiring iothread#1, and re-acquire iothread#0 when finished.

...easier said than done!

Multiqueue block layer

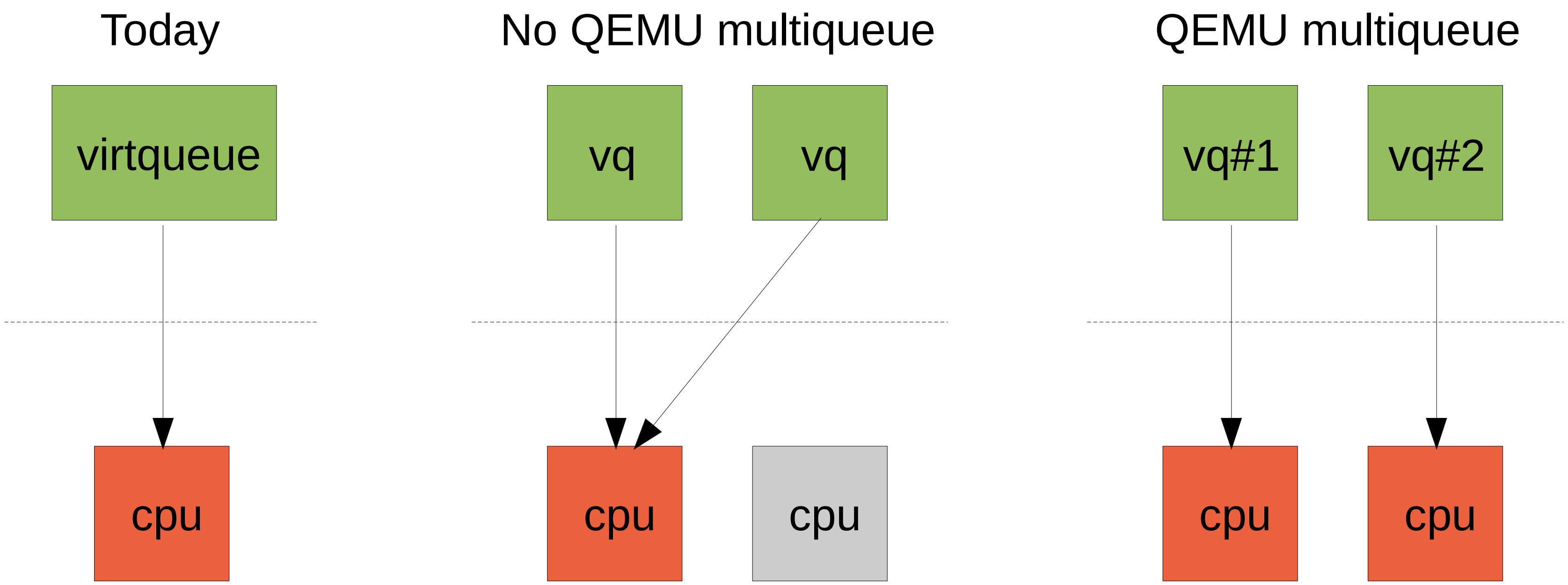
Host kernel now supports block devices with multiple request queues

- Improves SMP scalability

virtio-blk will support multiple virtqueues

How can QEMU avoid becoming the bottleneck?

Multiqueue block layer in QEMU



BlockDriverState requires AioContext acquire

For raw images it should be cheap to dispatch I/O

Image formats (qcow2) and storage features may restrict multiqueue

Multiqueue hack for raw image files

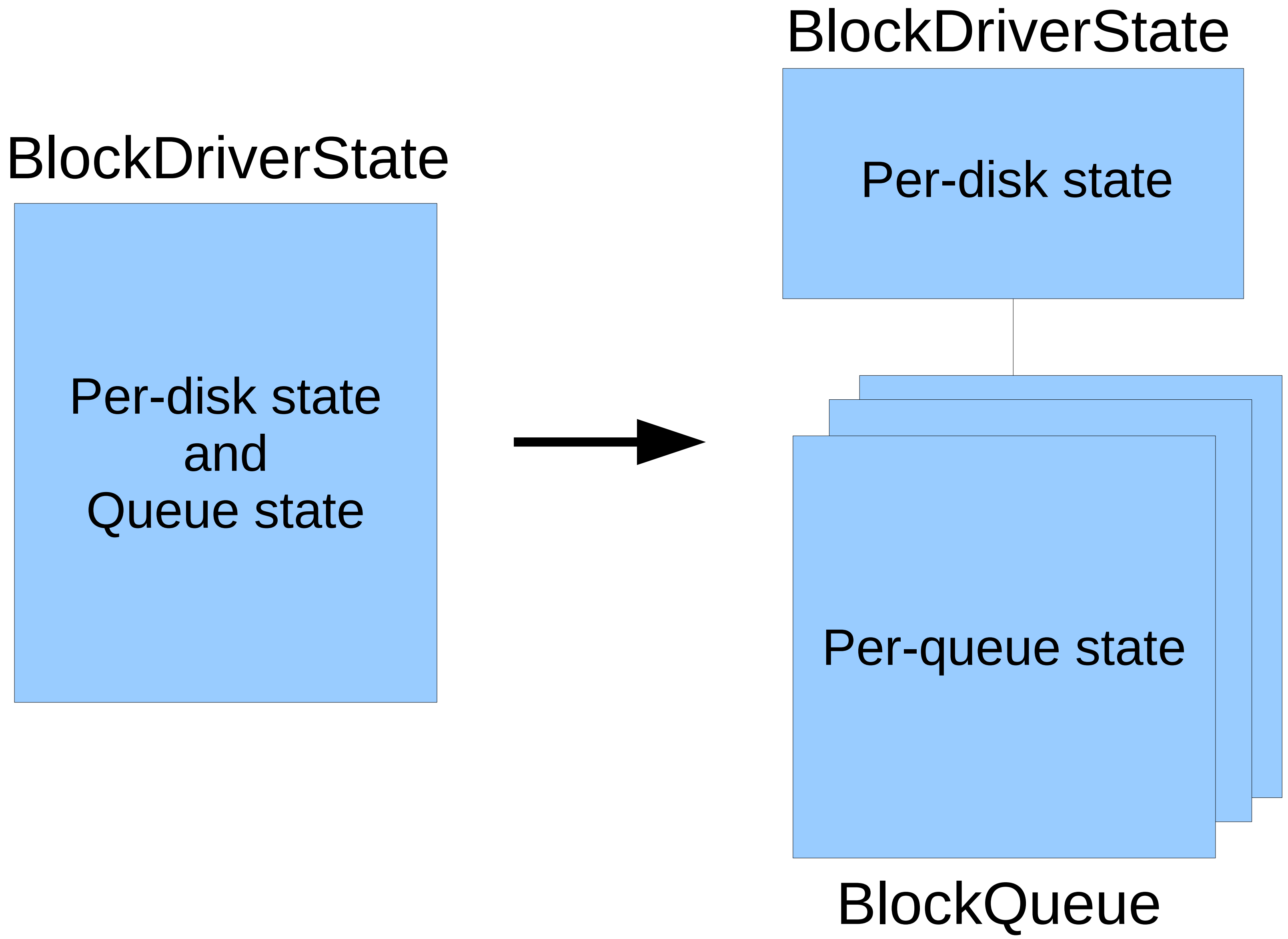
Start with dataplane code, lock per-device virtio state

For raw image files **only**:

- Open multiple BlockDriverStates for the same file
- Bind virtio-blk-pci device to the BDSes
- Run each device in a separate IOThread

Problem: Does not support image formats and breaks if snapshots/resize/etc operations are performed.

Breaking up BlockDriverState for multiqueue?



Questions?

Email: stefanha@redhat.com

Blog: http://blog.vmsplice.net/

IRC: stefanha on #qemu irc.oftc.net