VFIO

PCI device assignment breaks free of KVM

Alex Williamson
<alex.williamson@redhat.com>

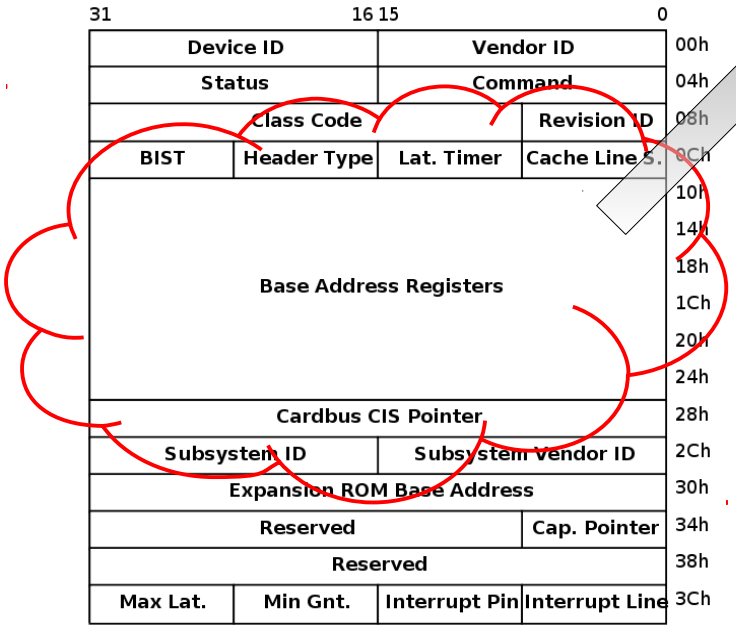
PCI 101 – Config space

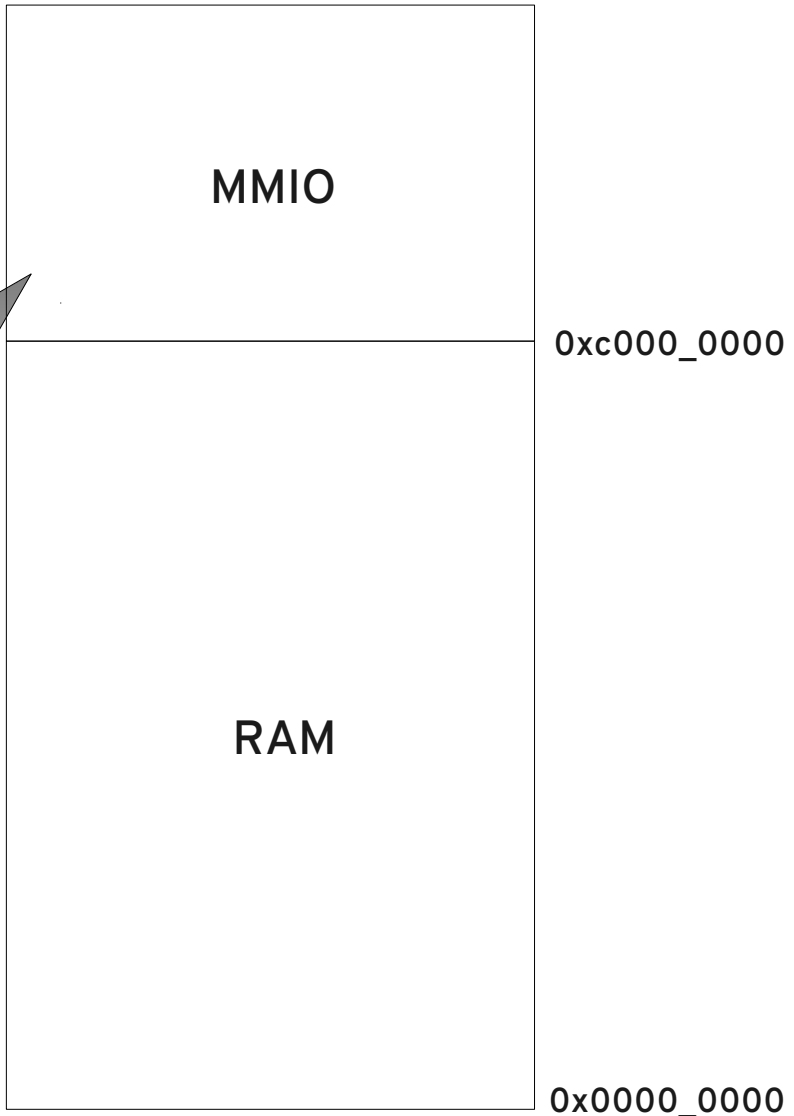
- Configuration space
  - Discovery
  - Pass-through
  - Emulated
  - Virtualized

31	16	15	0	
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Lat. Timer	Cache Line S.	0Ch
Base Address Registers				10h 14h 18h 1Ch 20h 24h
Cardbus CIS Pointer				28h
Subsystem ID		Subsystem Vendor ID		2Ch
Expansion ROM Base Address				30h
Reserved			Cap. Pointer	34h
Reserved				38h
Max Lat.	Min Gnt.	Interrupt Pin	Interrupt Line	3Ch

PCI 101 – Resources

- Resources
  - MMIO & IO port
  - Direct map
  - Trap & Emulate



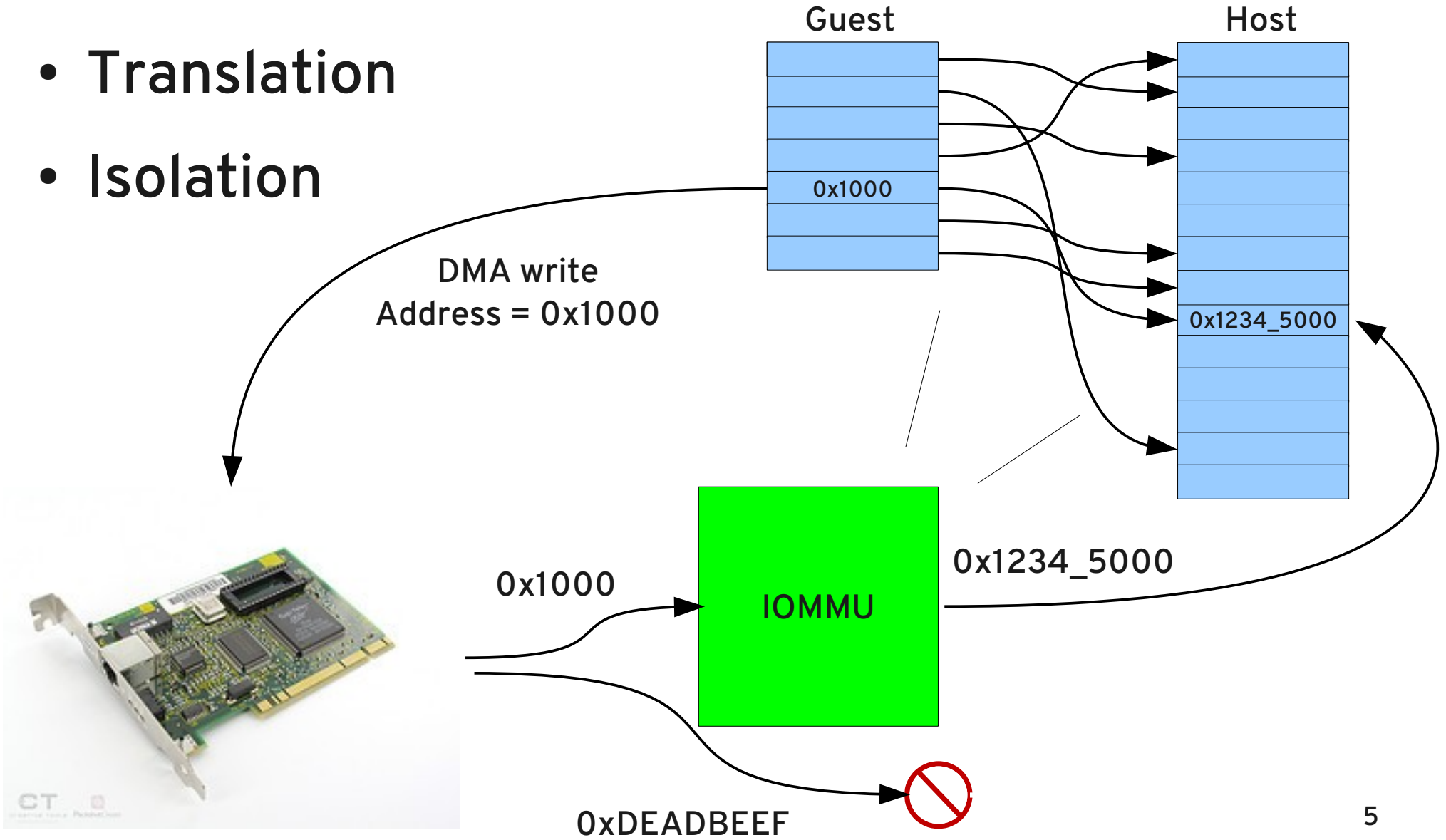


PCI 101 - Interrupts

- Interrupts
  - Legacy (INTx)
  - MSI/X

PCI 201 - IOMMU

- Translation
- Isolation



KVM Device Assignment

- Config space
  - Host pci-sysfs
  - Emulated in Qemu
- Resources
  - Host pci-sysfs
  - Direct map/trap & emulate in Qemu
- Interrupts
  - KVM irqchip

```
$ ls /sys/bus/pci/devices/0000:01:10.0/
broken_parity_status  modalias   resource
class                 msi_bus    resource0
config                numa_node  resource3
device                physfn     rom
enable                power      subsystem_device
irq                   remove     subsystem_vendor
local_cpulist         rescan     uevent
local_cpus            reset      vendor
```

Pitfalls of KVM device assignment

- PCI stub
- PCI sysfs
- Security
- Depends on KVM
- x86 only
- KVM is not a device driver (and shouldn't be)

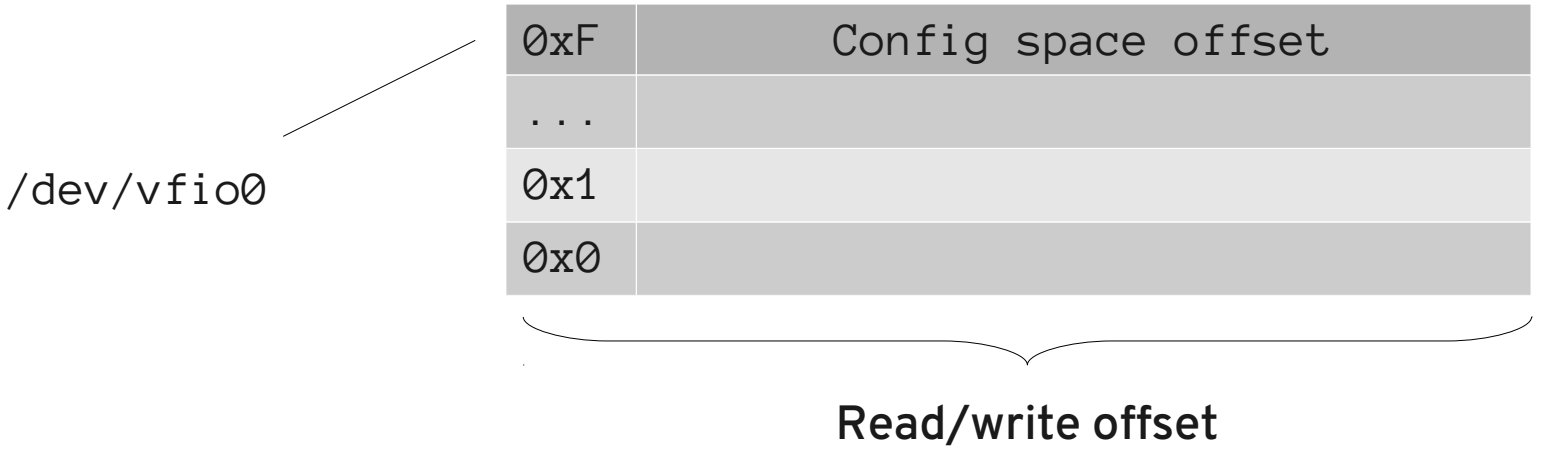
VFIO

- Virtual Function I/O*
- Derivative of UIO
  - Enhanced interrupt support
  - PCI config space virtualization
  - IOMMU support – via UIOMMU
- Supports virtualization and userspace drivers
- Developed by Tom Lyon
- VFIO is a device driver

* not limited to SR-IOV

VFIO device assignment

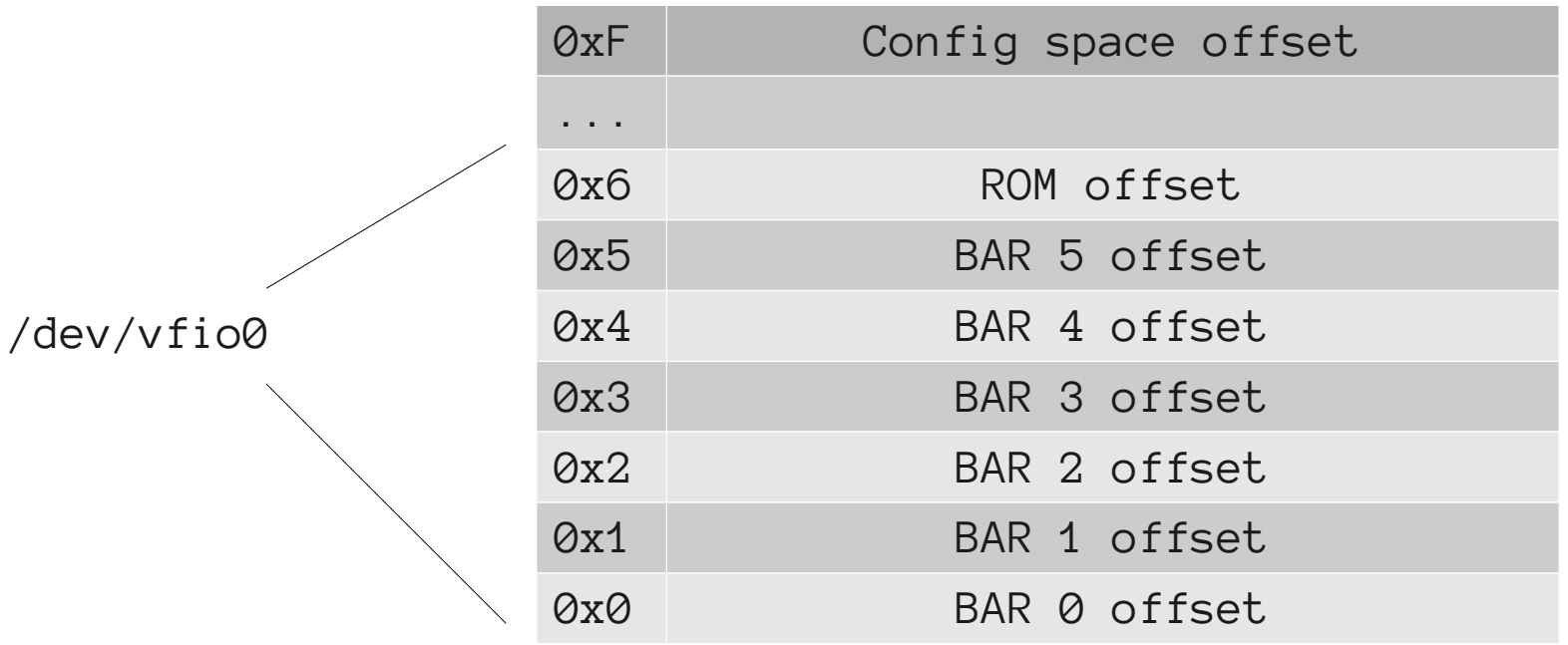
- Config space
  - Virtualized in the host
  - Read/write access via `/dev/vfio0`



Example:
pread(fd, buf, 4, 0xF00_0000_0010) = read dword at config space offset 0x10 (BAR0)

VFIO device assignment

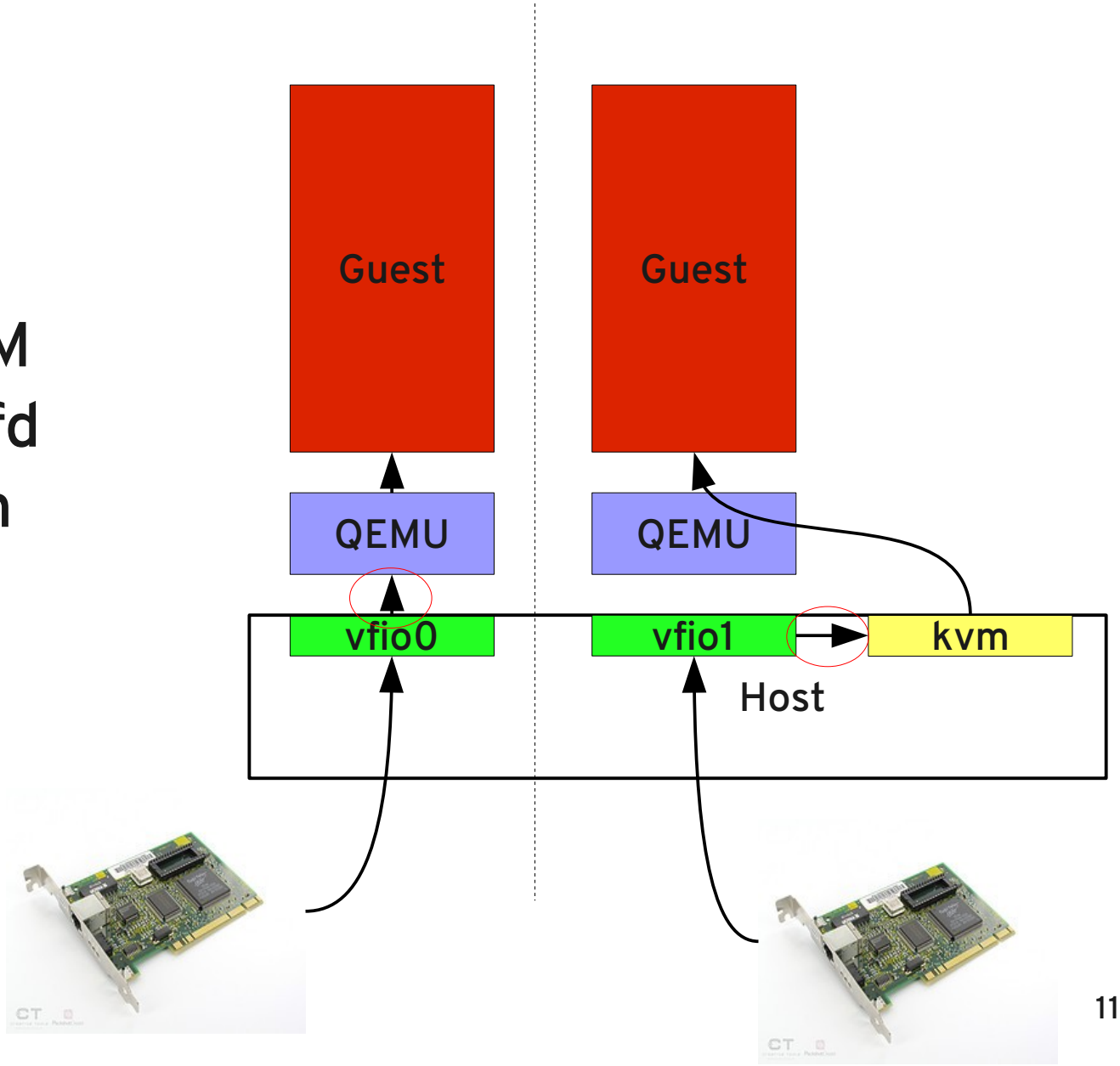
- Resources
  - Read/write & mmap via `/dev/vfio0`
  - Direct map or trap & emulate in Qemu



VFIO device assignment

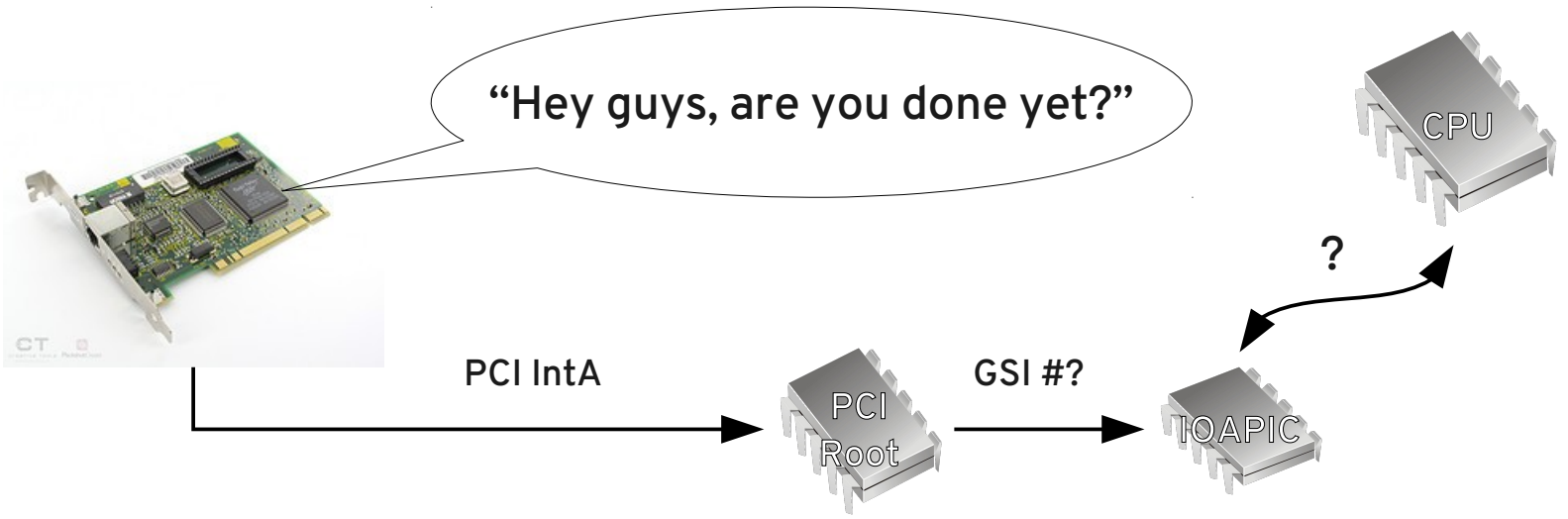
Interrupts

- Eventfds
- Enables KVM eventfd/irqfd acceleration



Legacy interrupts: not so fast!

- No QEMU EOI support
  - How do we know when to unmask?
- My PCI interrupt goes where?
  - PCI interrupt → “GSI” → IOAPIC pin madness



Legacy interrupt enhancements

- EOI notifiers
  - Register EOI notifier with GSI
  - Notified when guest CPU writes APIC EOI
  - What's my GSI?
- Add pci_get_irq()
  - Chipset specific implementation to translate PCI INT{A,B,C,D} → GSI
  - The guest reprogrammed the chipset, what now?
- Update IRQ notifier

Can we do better?

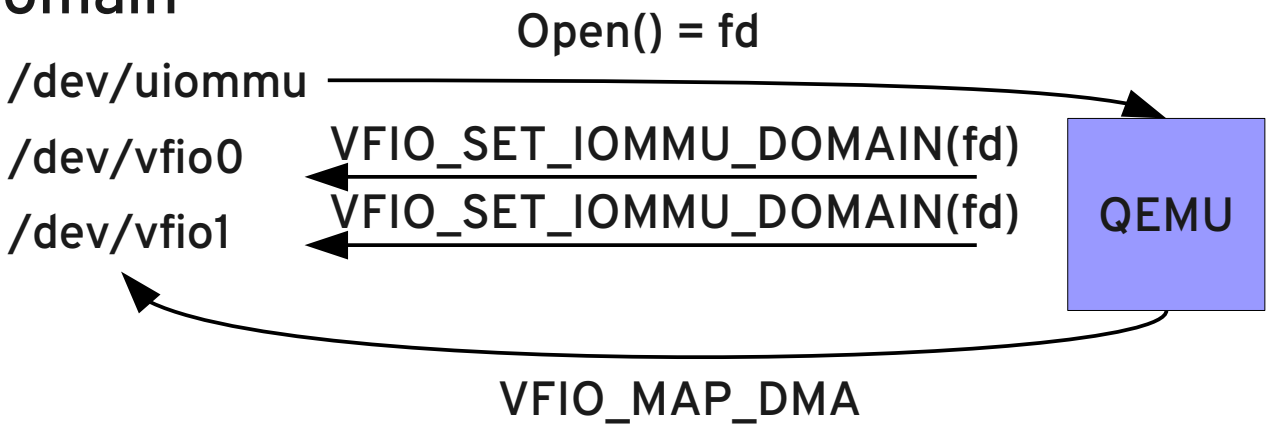
- High performance devices support MSI/X
  - No masking
  - No EOI
  - Easy to bypass userspace (same as vhost)
- Possible KVM accelerations
  - VFIO INTx eventfd → KVM irqfd
    - Need level triggered KVM irqfd support
  - KVM EOI eventfd → VFIO EOI irqfd

UIOMMU

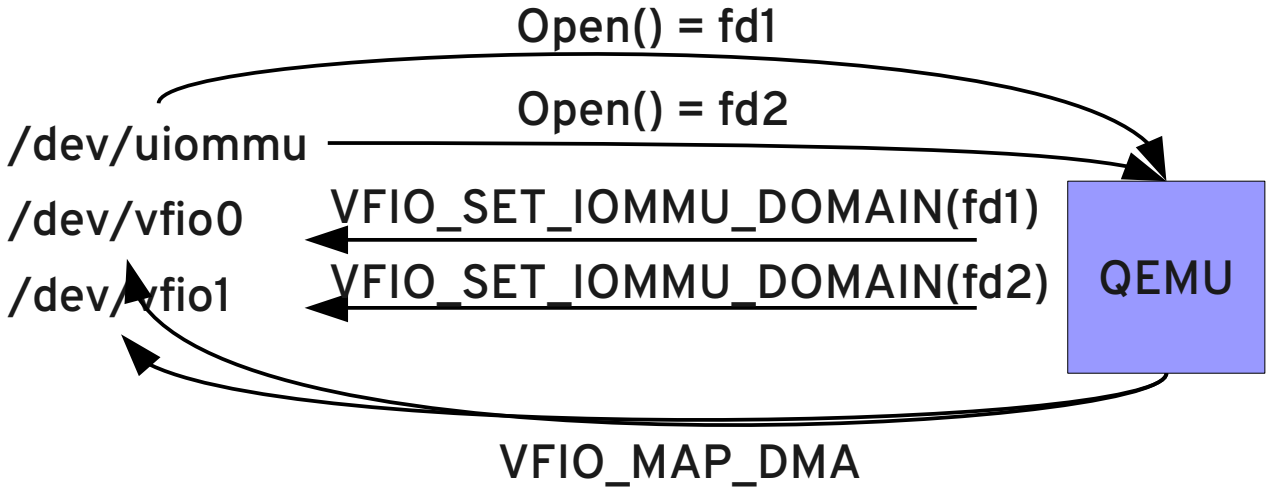
- User abstraction of IOMMU ops
  - Open() ~ iommu_domain_alloc()
  - Close() ~ iommu_domain_free()
  - VFIO_SET_UIOMMU_DOMAIN ~ iommu_attach/dettach_device()
  - VFIO_MAP/UNMAP_DMA ~ iommu_map/unmap()

UIOMMU

Shared Domain

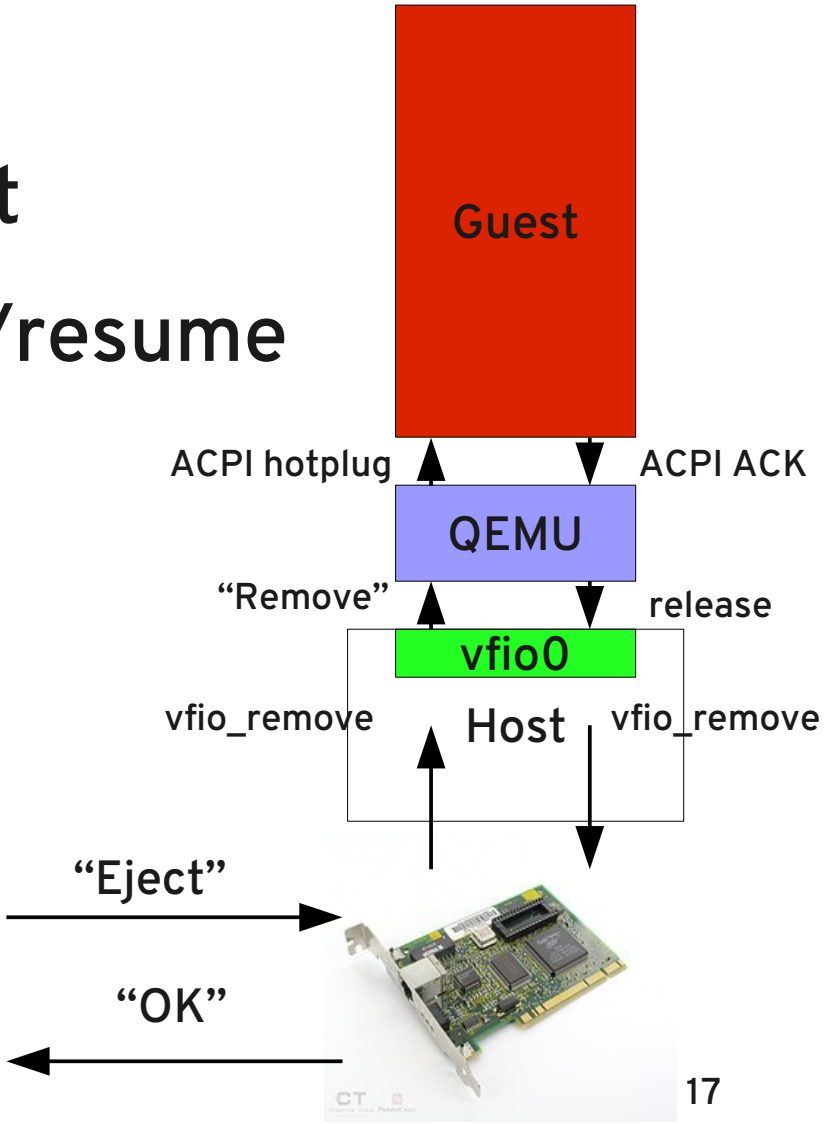


Independent Domains



Netlink

- Side-band channel
- Signal host PCI errors to guest
- Notify QEMU of host suspend/resume
- Host initiated device removal



Where to find it

- git://github.com/awilliam/linux-vfio.git vfio
- git://github.com/awilliam/qemu-vfio.git vfio
- Similar usage to pci-assign
  - `-device vfio,host=1:10.0,id=hostdev0,<vfiofd=>, \`
    `<uiommufd=>,<shared_uiommu_domain=on/off>`

What's next?

- Merge upstream
- PCI-e host enhancements
  - Need Q35 chipset support
- Non-PCI support?!
- Non-x86 support...

IOMMU and Isolation differences

- VT-d/AMD-Vi
  - Page table driven (can map entire guest)
  - Share page tables across separate IOMMUs
  - Requester ID mapping granularity
- POWER
  - Partitionable Endpoints ~ “groups”
  - IOVA windows
  - MMIO segments
  - pvDMA

Can we support both?

- Currently:
  - Only support fully mapped guests
  - Require device granularity
- Needed:
  - Proper group handling
  - Logical DMA to IOMMU mappings
  - pvDMA mode
  - Efficient DMA mapping API

VFIO (Device Assignment) BoF

- sysfs “groups” or “iommus” or $YOUR_IDEA?
- Implementation plan
- Upstream plan
- Ponies

Thank you!