VFIO: A user's perspective

Alex Williamson
alex.williamson@redhat.com
November 8th, 2012

What is VFIO?

What is VFIO?

- A new user level driver framework for Linux
- **V**irtual **F**unction **I**/**O**[*]
- Originally developed by Tom Lyon (Cisco)
- IOMMU-based DMA and interrupt isolation
- Full devices access (MMIO, I/O port, PCI config)
- Efficient interrupt mechanisms
- Modular IOMMU and device backends

* not limited to SR-IOV

What does this mean for  ?

What does this mean for Qemu?

- A new device assignment interface
  - Device assignment = userspace driver
  - Unbinds device assignment from KVM
  - Better security model
    - For both devices and users
  - Device isolation
  - Architecture portability

We already have KVM PCI device assignment

We already have KVM PCI device assignment

- pci-assign has problems
  - KVM is a hypervisor (not a device driver)
  - Resource access is incompatible with secure boot
  - IOMMU granularity is not assured
  - Poor device ownership model
  - x86 only
  - PCI only
  - KVM only

How does VFIO solve these problems?

KVM is not a device driver

- VFIO is a device driver
  - supports modular device driver backends
  - vfio-pci binds to non-bridge PCI devices
  - pci-stub available as “no access” driver
    - Allows admins to restrict access within a group
  - Users cannot attempt to use in-service host devices
  - Devices in use by users cannot be simultaneously claimed by other host drivers

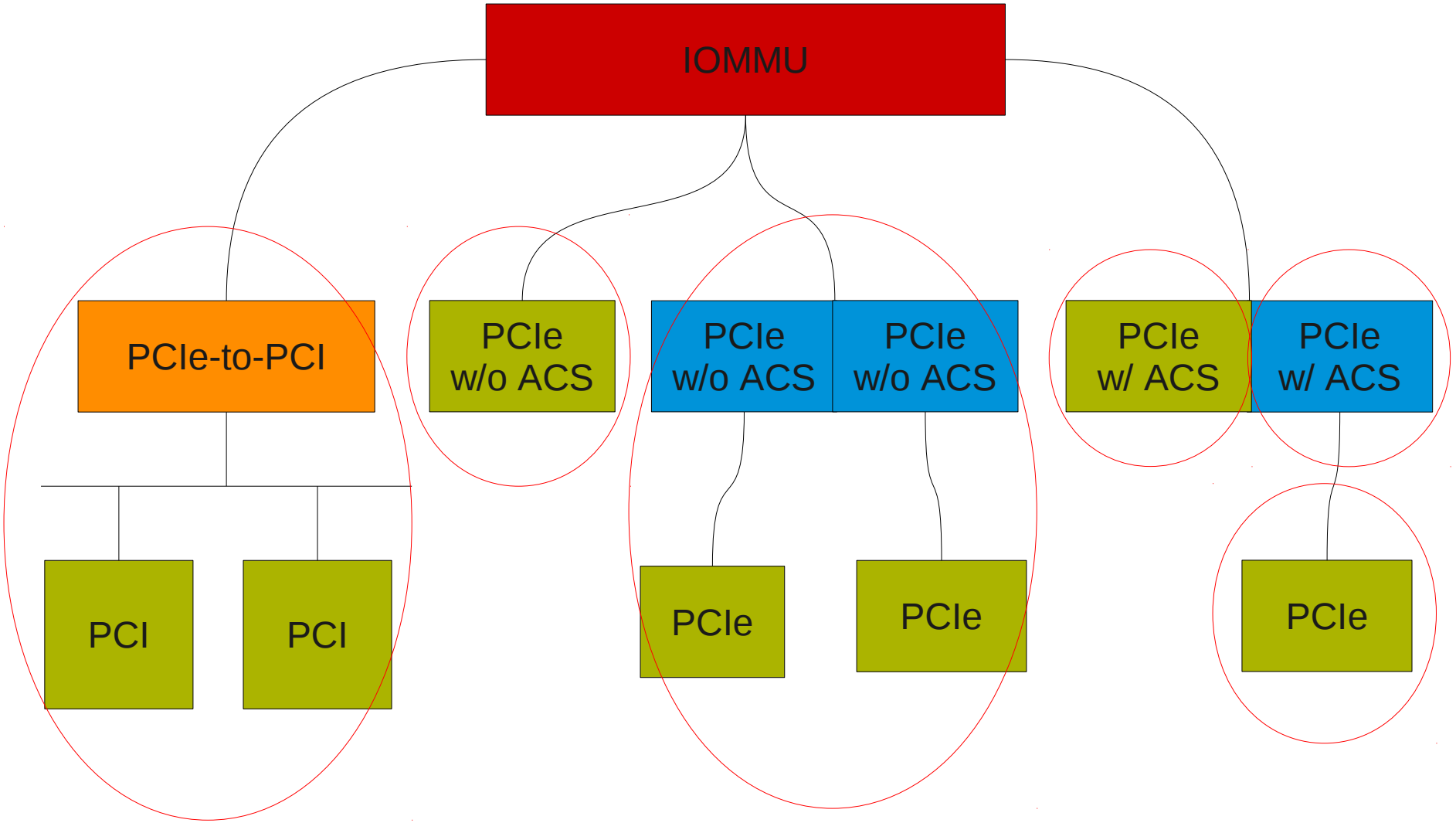
Resource access is incompatible with secure boot

- VFIO device backends provide secure resource access
  - No device access without IOMMU isolation
  - Integral to the interface
    - Not outsourced to pci-sysfs
  - Virtualized access to PCI config space

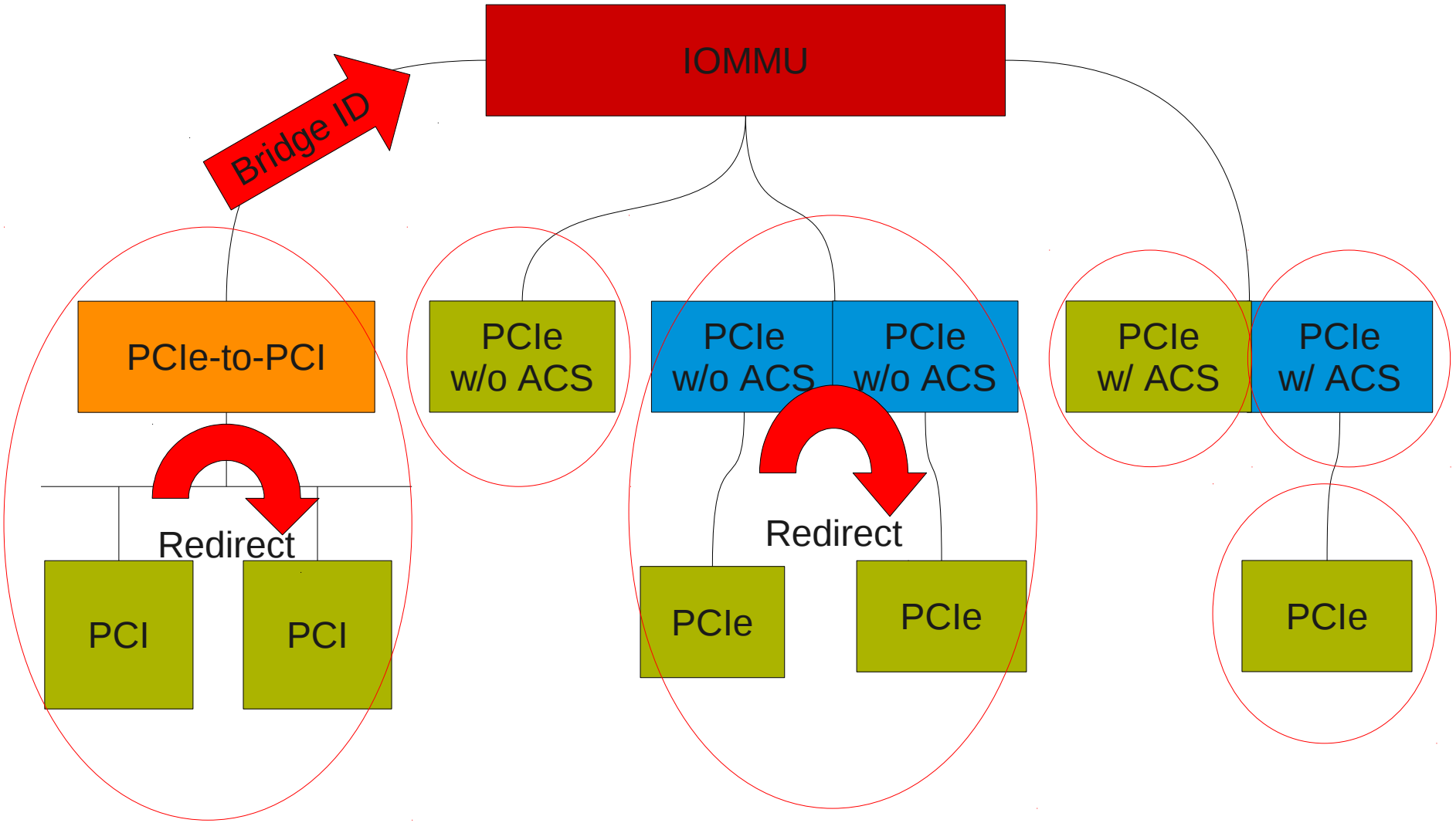
IOMMU granularity is not assured

- VFIO uses IOMMU groups
  - Allows the IOMMU driver to define both visibility and containment
  - Solves devices hidden by bridges
    - IOMMU cannot differentiate devices behind PCI bridge
  - Solves peer-to-peer back channels
    - All transactions required to reach IOMMU for translation
    - For PCIe, ACS (Access Control Services) indicates support
  - Result is better **security**

IOMMU Group examples



IOMMU Group examples



Poor device ownership model

- VFIO moves ownership to the group level
  - Access to device file grants ownership
  - Ownership extends to all devices within the group
  - All accesses through VFIO

x86 only, PCI only, KVM only

- VFIO supports a modular IOMMU interface
  - IOMMU API (type1) implemented
  - POWER (SPAPR) under development
- VFIO supports a modular device interface
  - PCI (vfio-pci) implemented
- VFIO has no KVM dependencies
  - Used only for acceleration
  - Non-x86 guests on x86 host work today
    - ppc g3beige – Big Endian driver test platform!
    - Any guest platform with PCI support

Great, how do we use it?

Requirements

- AMD-Vi or Intel VT-d capable hardware
- Linux 3.6+ host
  - CONFIG_VFIO_IOMMU_TYPE1=m
  - CONFIG_VFIO=m
  - CONFIG_VFIO_PCI=m
  - modprobe vfio-pci
- Qemu 92e1fb5e+ (1.3 development tree)

Understanding IOMMU groups (easy example)

- Device to assign:

```
01:10.0 Ethernet controller: Intel Corporation 82576
Virtual Function (rev 01)
```

- Find the group:

```
$ readlink /sys/bus/pci/devices/0000:01:10.0/iommu_group
../../../../kernel/iommu_groups/15
```

- IOMMU Group = 15
- Check the devices in the group:

```
$ ls /sys/bus/pci/devices/0000:01:10.0/iommu_group/devices/
0000:01:10.0
```


Binding to vfio-pci

- Unbind from device driver

```
$ echo 0000:01:10.0 | sudo tee \
/sys/bus/pci/devices/0000:01:10.0/driver/unbind
```

- Find vendor & device ID

```
$ lspci -n -s 01:10.0
01:10.0 0200: 8086:10ca (rev 01)
```

- Bind to vfio-pci

```
$ echo 8086 10ca | sudo tee \
/sys/bus/pci/drivers/vfio-pci/new_id
```

- Check

```
$ ls /dev/vfio
15  vfio
```


Start a guest

```
sudo qemu-system-x86_64 -m 2048 -hda rhel6vm \
-vga std -vnc :0 -net none -enable-kvm \
-device vfio-pci,host=01:10.0,id=net0
```


Start a guest

```
sudo qemu-system-x86_64 -m 2048 -hda rhel6vm \
-vga std -vnc :0 -net none -enable-kvm \
-device vfio-pci,host=01:10.0,id=net0
```

- Why the sudo?

Start a guest

```
sudo qemu-system-x86_64 -m 2048 -hda rhel6vm \
-vga std -vnc :0 -net none -enable-kvm \
-device vfio-pci,host=01:10.0,id=net0
```

megabytes

- Why the sudo?

```
$ ulimit -l
64
```

kilobytes

- VFIO enforces user limits!
  - VFIO security++

Why is memory locked?

- For x86, all of guest memory is pinned on the host
  - No guest visible IOMMU
  - Devices can DMA to any guest memory address
  - Guest memory can't be swapped if it's a DMA target
  - We don't know what memory is a DMA target
  - Pin it all!

It's just a ulimit, increase it!

```
$ sudo -s
# chown $USER:$GROUP /dev/vfio/15
# chmod 660 /dev/vfio/vfio
# ulimit -l 2117632
# su - $USER
$ qemu-system-x86_64...
```


Maths

```
$ sudo -s
# chown $USER:$GROUP /dev/vfio/15
# chmod 660 /dev/vfio/vfio
# ulimit -l 2117632
# su - $USER
$ qemu-system-x86_64...
```

?

Maths

- ulimit is padded: 2048 x 1024 = 2097152
- Both guest memory and devices are mapped
  - Frame buffer, PCI MMIO BARs, etc.
- +20MB covers additional mappings for this config
  - (2048 + 20) x 1024 = 2117632
- Deterministic?
  - +512MB covers 32bit MMIO space (Q35?)
  - What about 64bit MMIO or memory hotplug?

Other options

- `/etc/security/limits.conf`
  - Set the ulimit for a user
- libvirt will need to set limits when using vfio-pci
- Other?

Understanding IOMMU groups (harder example)

- Device to assign:

```
05:00.0 Ethernet controller: Broadcom Corporation NetXtreme
BCM5755 Gigabit Ethernet PCI Express (rev 02)
```

Find the group:

```
$ readlink /sys/bus/pci/devices/0000:05:00.0/iommu_group
../../../../kernel/iommu_groups/8
```

- IOMMU Group = 8
- Check the devices in the group:

```
$ ls /sys/bus/pci/devices/0000:05:00.0/iommu_group/devices/
0000:00:1c.0  0000:00:1c.4  0000:04:00.0  0000:05:00.0
```

Whoa

Why?

```
$ lspci -t -s 1c.
-[0000:00]-+-1c.0-[04]--
           \-1c.4-[05]--

$ lspci -s 1c.
00:1c.0 PCI bridge: Intel Corporation 82801JI
          (ICH10 Family) PCI Express Root Port 1
00:1c.4 PCI bridge: Intel Corporation 82801JI
          (ICH10 Family) PCI Express Root Port 5
```

Device 1c is a multifunction device that does not support PCI ACS control

- Devices 04:00.0 & 05:00.0 can potentially do peer-to-peer DMA bypassing the IOMMU
- IOMMU Groups recognize they are not isolated

Can we still use it?

```
for i in $(ls /sys/kernel/iommu_groups/8/devices/); do
  echo $i | sudo tee \
    /sys/kernel/iommu_groups/8/devices/$i/driver/unbind
  VEN=$(cat /sys/kernel/iommu_groups/8/devices/$i/vendor)
  DEV=$(cat /sys/kernel/iommu_groups/8/devices/$i/device)
  echo $VEN $DEV | sudo tee \
    /sys/bus/pci/drivers/vfio-pci/new_id
Done
```

- Attach all the devices to vfio-pci
- Ownership is based on group
- Unused devices are held by vfio-pci for isolation
- Advanced users: VFIO also allows group members to be assigned to pci-stub or no driver to prevent user access. pci-stub strongly preferred.

What about performance?

- PCI config space
  - Not performance critical
  - vfio-pci & pci-assign are equivalent
- I/O port access
  - Not used by high performance devices
  - vfio-pci & pci-assign are equivalent
- MMIO region access
  - Both vfio-pci & pci-assign directly map to VM
  - vfio-pci & pci-assign are equivalent

What about performance? (cont)

- Interrupts
  - pci-assign: KVM interrupt handler, posted to guest
  - vfio-pci: VFIO interrupt handler connected to KVM irqfd
  - Very low overhead VFIO → KVM signaling
  - Testing shows vfio-pci has an advantage*
    - Likely from non-threaded vs threaded interrupt handler
  - Preliminary data from HP on 10G NIC is promising

*netperf TCP_RR (igbvf, e1000e, tg3)

Device support

- Most commercial use of device assignment?
  - NICs
  - HBAs
- Most requested hobbyist/enthusiast device?
  - VGA
  - Video encoders/capture

Why is VGA so hard?

- Legacy I/O ranges
  - MMIO: `0xa0000 – 0xbffff`
  - I/O port: `0x3c0 – 0x3df`
  - Routing controlled through host chipset
    - For every R/W to regions, switch host routing, access, restore
    - Host use of VGA arbiter still evolving
- ROM dependencies
  - ROM initializes the device (primary head or Linux)
  - Can bypass virtualized access paths (1:1 mapping)
  - Accessibility problems

Why is VGA so hard? (cont)

- Driver
  - Companion device & chipset dependencies
  - Black box
- Qemu
  - Emulated VGA is not easy to remove
    - `-nographics` is not sufficient (getting better?)
  - Bus topology for multiple graphics cards
- BIOS/Qemu
  - Greatly improved to support large framebuffers
    - But not multiple

Call to action

- Please test & use VFIO
  - Host Linux kernel 3.6+
  - Qemu 1.3+ & current development tree
- Needed
  - libvirt & virt-manager support
  - Test infrastructure
  - Error handling (AER)
  - VGA support
  - Power management
  - New host platform support
  - Hardware vendors: Support PCI ACS!

Questions?

Thanks!