redhat.

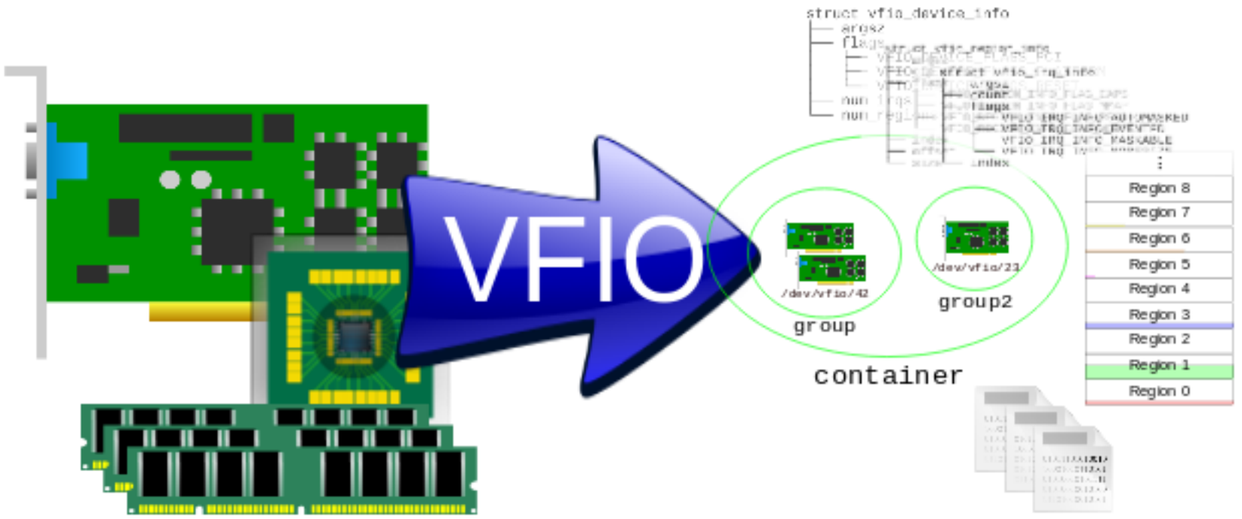
VFIO Device Assignment Quirks

...and how to avoid them in your hardware

Alex Williamson / alex.williamson@redhat.com


A quick VFIO refresher
redhat.

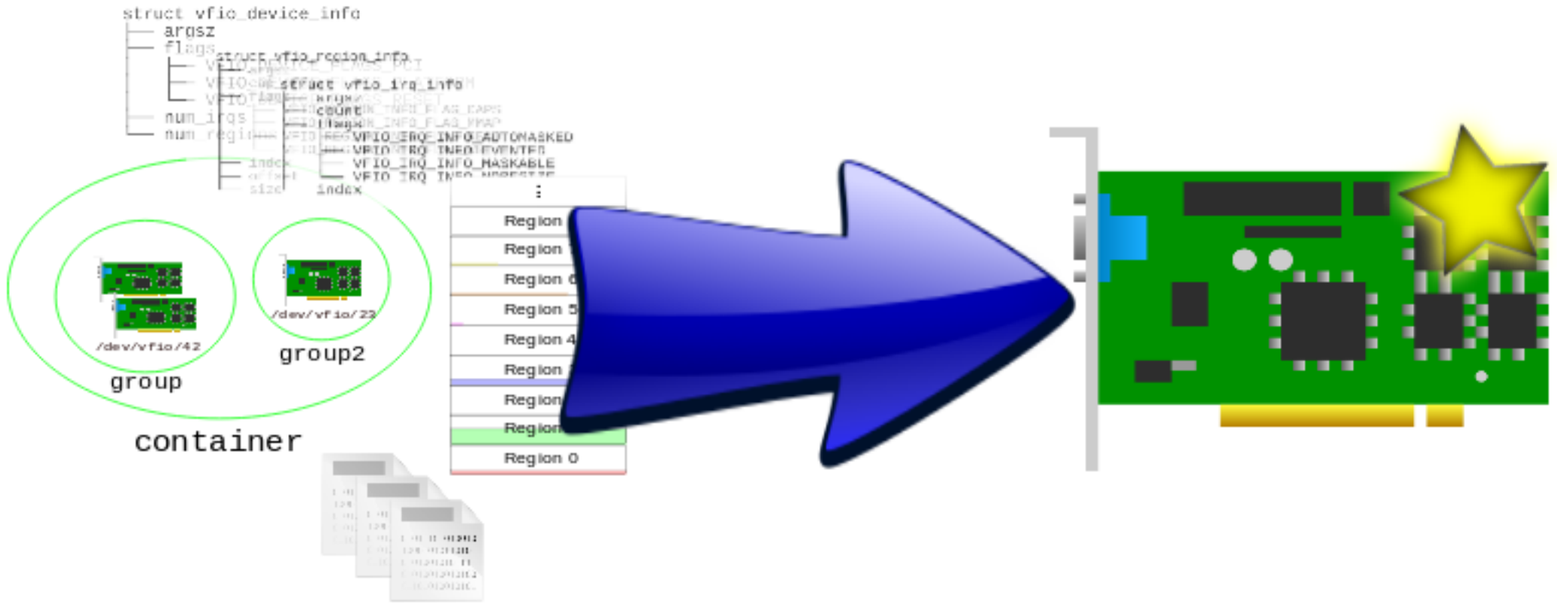
VFIO: A userspace driver interface



Devices are decomposed into a userspace API

redhat.

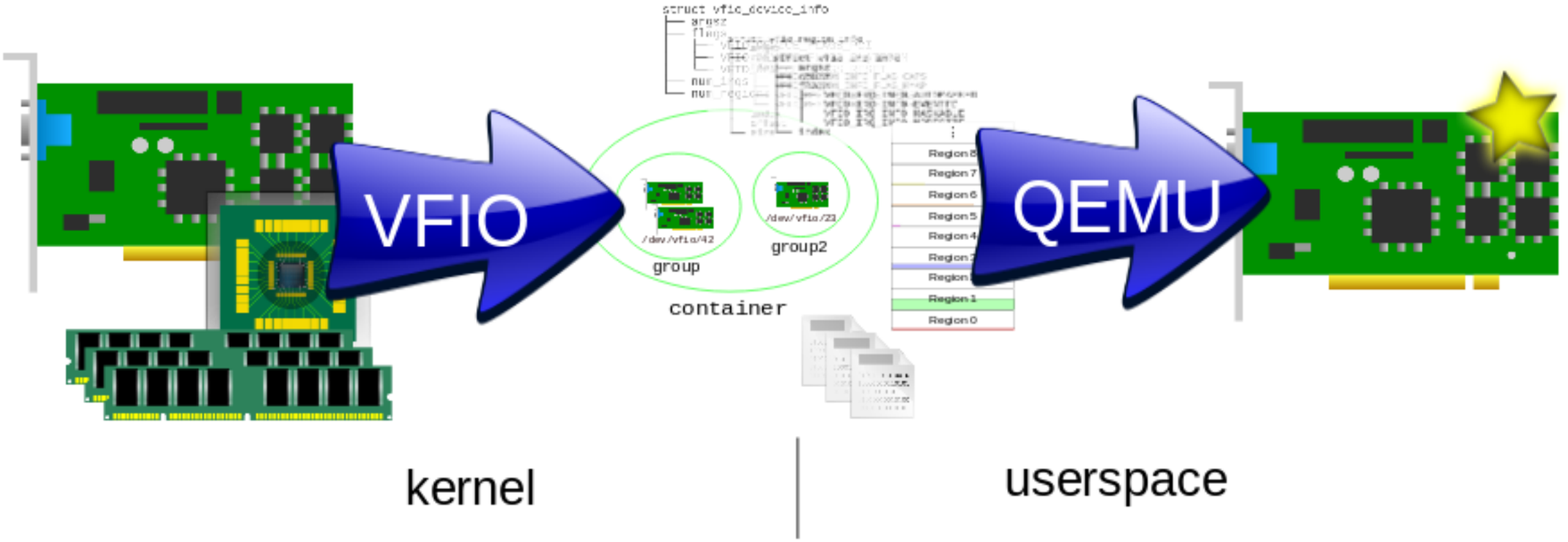
QEMU consumes the VFIO API



Recomposing the physical device to a virtual device

redhat.

For further details:



KVM Forum 2016: "An Introduction to PCI Device Assignment with VFIO"

redhat.


Device Assignment
Requirements
redhat.

Requirements

- Virtualization
- Identification
- Isolation
- Reproducibility

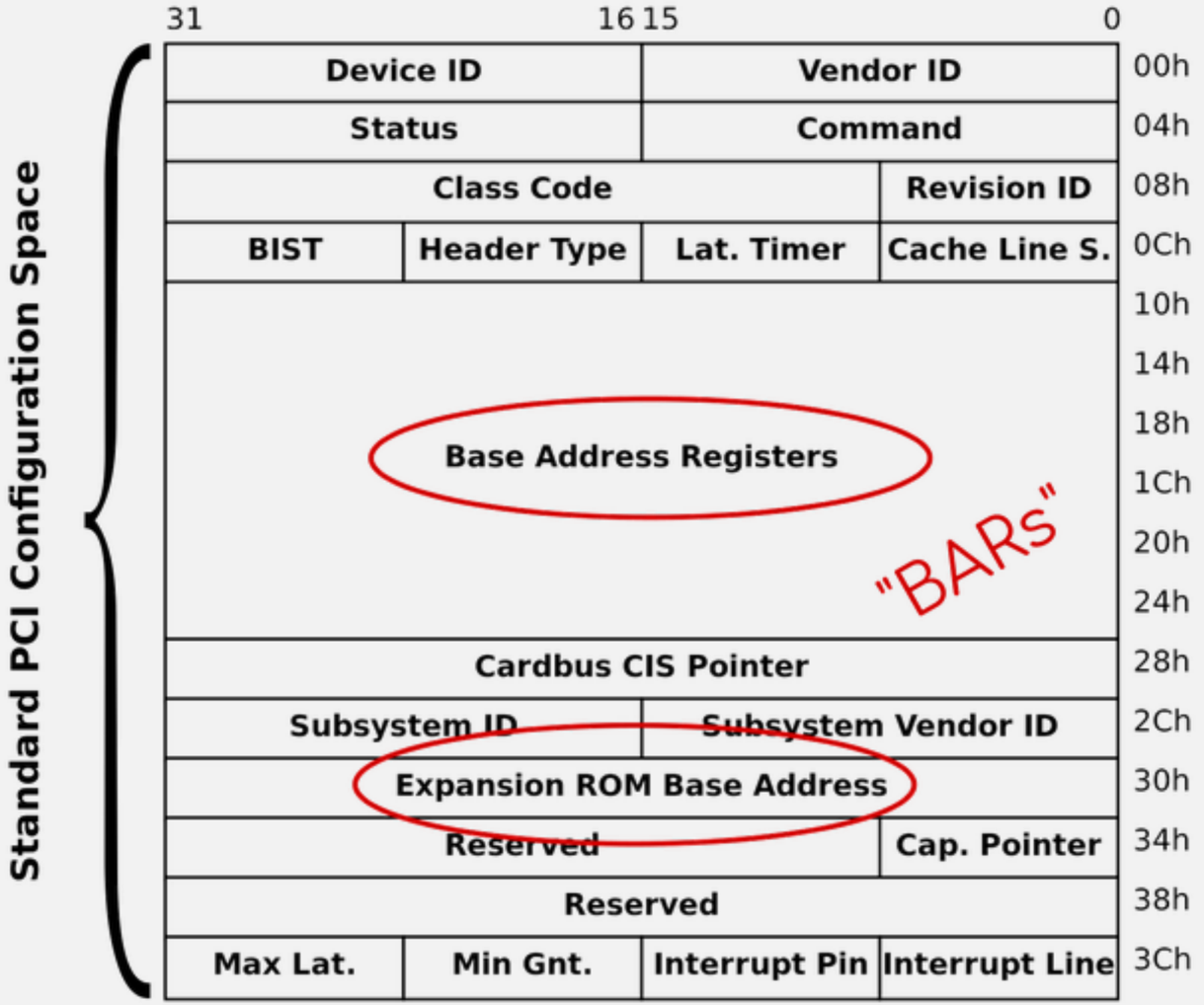
Virtualization

Requirement: Virtualization

- Hypervisor transposes device from host address space to VM address space
- IOMMU only accounts for DMA translation
- Virtualization required for:
  - Mapping device MMIO and I/O port resources
  - Translating interrupt address spaces
  - Topology translations
  - External dependencies

redhat.

PCI Resource Mapping



PCI BARs

- Describes resource type and features
  - MMIO vs I/O port, 32-bit, caching, etc
- Exposes resource size
- Locates resources within address space…

User cannot relocate physical resource,
BARs must be virtualized and mapped
into VM address space

PCI BAR Virtualization

- Pass-through type and feature bits
- Emulate BAR sizing interface
- Map resources into VM address space
- Host resource address remains static

Emulate the standard programming model defined by PCI

Except...

Is PCI configuration space the only way to access these registers?

Config Space Backdoors

- PCI defines the configuration address space and register definition
- Register implementation is not defined
- Implementations may expose configuration space registers via multiple address spaces

redhat.

Example Backdoors

- Config space mirrors in MMIO BAR spaces
- Window/data registers to config space through I/O port BARs
- VGA register hooks to BAR offsets

Result: PCI config space virtualization is not sufficient for all devices

Needed:

Infrastructure for virtualizing config space registers across all address spaces

Our first quirk…

VFIOQuirks

- QEMU-based quirks
- Infrastructure within VFIOPCIDevice for handling device specific quirks
- Helpers to facilitate common mirrors and windows to PCI config space
- Extensive use of MemoryRegions
- VFIOQuirks are sub-regions within the default region mapping

Problem solved?

- KVM maps MMIO at PAGE_SIZE granularity
- Any quirk within an otherwise directly mapped page traps the entire page through QEMU

What other registers reside within the page?

Case Study: NVIDIA

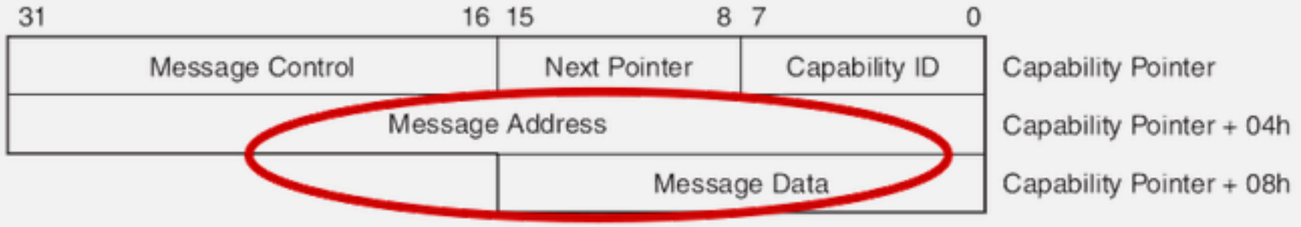
- MSI *"ACK"* required to retrigger interrupt
- MMIO write through BAR0 config mirror
- Increases MSI re-triggering latency

(partially addressed with ioeventfd handling)

But device resource mappings are not the only ranges needing virtualization...

Interrupt Address Space

Message Signaled Interrupts (MSI) trigger by writing pre-programmed data to a pre-programmed address



Both of these require address space virtualization

MSI is entirely configured through PCI configuration space, already virtualized…

MSI-X

Address/Data specified per vector within MMIO BAR resident vector table

BAR Index Register + Table Offset

DWORD 3	DWORD 2	DWORD 1	DWORD 0		
Vector Control	Msg Data	Msg Upper Addr	Msg Addr	entry 0	Base
Vector Control	Msg Data	Msg Upper Addr	Msg Addr	entry 1	Base + 1*16
Vector Control	Msg Data	Msg Upper Addr	Msg Addr	entry 2	Base + 2*16
...	...	...	...	...	...
Vector Control	Msg Data	Msg Upper Addr	Msg Addr	entry (N-1)	Base + (N-1)*16

MSI-X Vector Table

- Resides within MMIO BAR
- Requires address space translation
- Opportunities for backdoors
- Page size virtualization granularity
  - No performance critical registers within same page as MSI-X structures
  - Some architectures use **64KB** page size

redhat.

MSI-X Vector Table Recommendations

- Use an exclusive BAR for MSI-X data structures
- Or, allow ample alignment of structures (64KB)
- For existing devices, “x-msix-relocation”:
  - QEMU vfio-pci device option
  - Modifies MSI-X capability to relocate to new BAR or extend existing BAR for alignment

redhat.

Topology & Firmware Considerations

- Does the device driver depend on or assume PCI capability presence or offsets?
  - Virtualization may change, move, or hide
- Does the device depend on other devices, firmware tables, or reserved memory?
  - Intel IGD (GVT-d): All of the above!
  - *Integrated* vs *Discrete* mindset

redhat.

Lessons

- Avoid “backdoors” to PCI config space
- Use VFIOQuirks to fill virtualization gaps
- Be mindful of virtualization granularity
- Avoid external dependencies and hardwired assumptions

Corollary: Assigned device config space accesses may be slower than bare metal

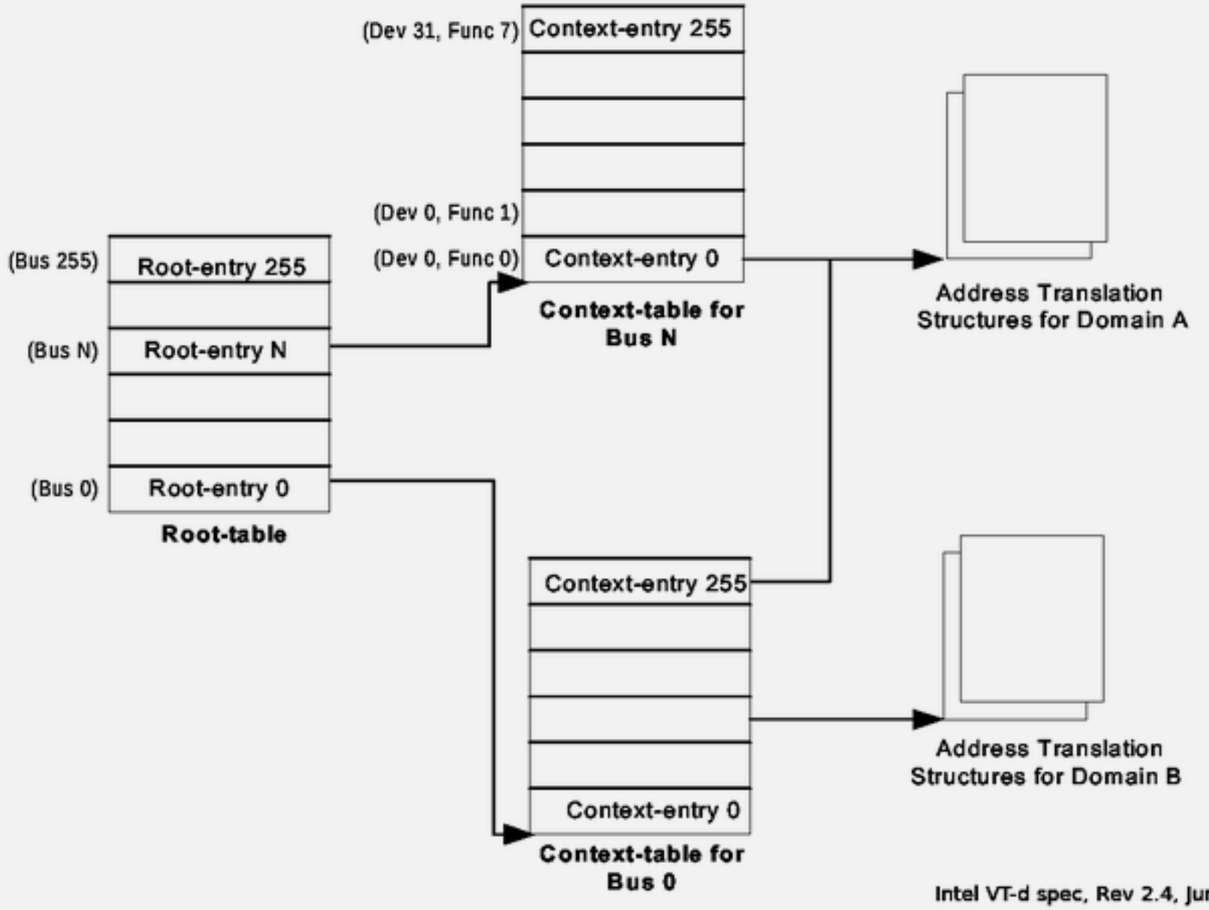
redhat.

Identification

Requirement: Identification

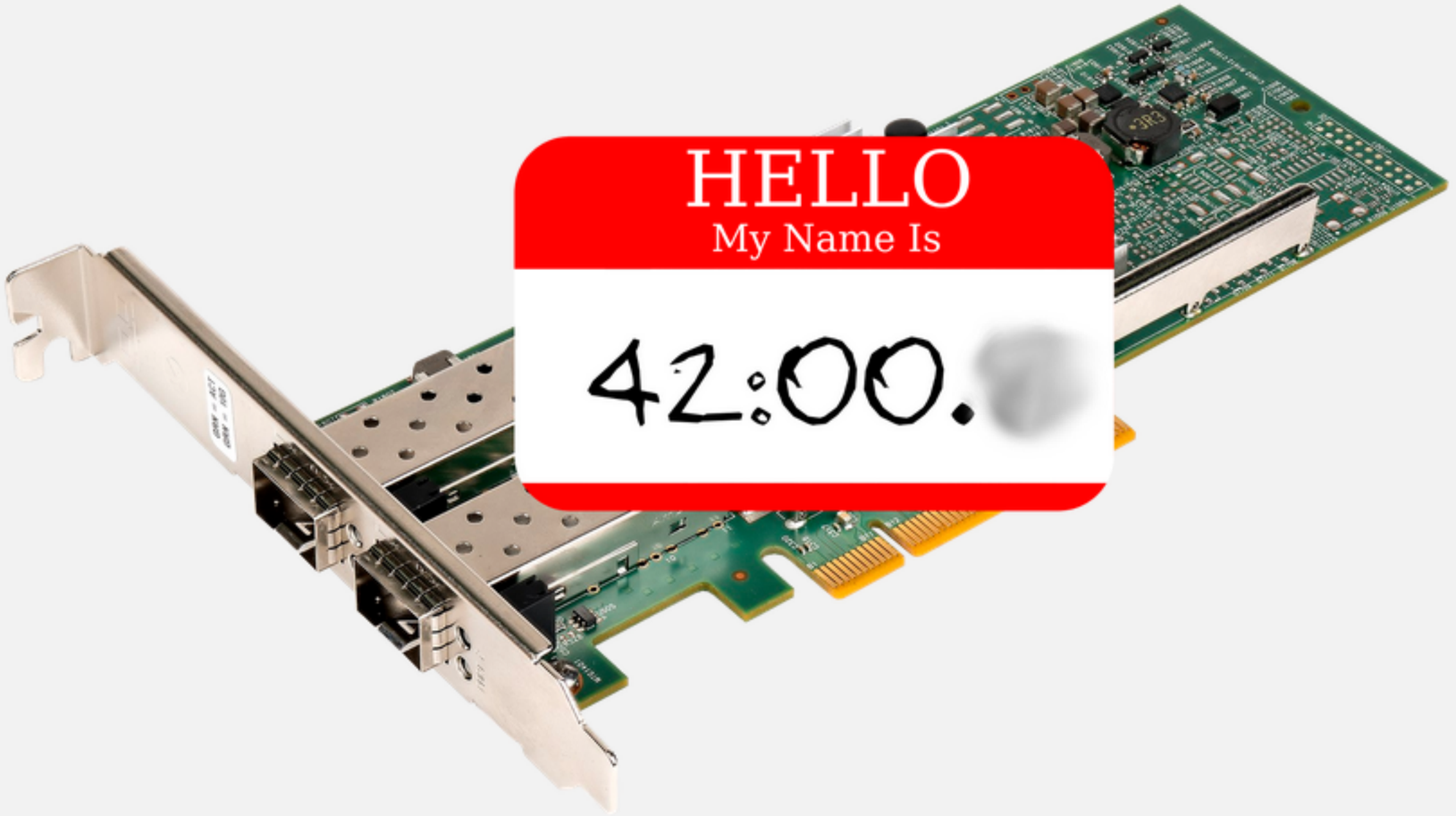
- IOMMU hardware must be able to associate each device to a translation context
- Native PCIe devices use unique Requester ID
- Platform devices use Stream IDs or ACPI IDs

IOMMU context lookup by PCIe requester ID (bus:dev.fn)



Intel VT-d spec, Rev 2.4, June 2016

What requester ID does the IOMMU see?



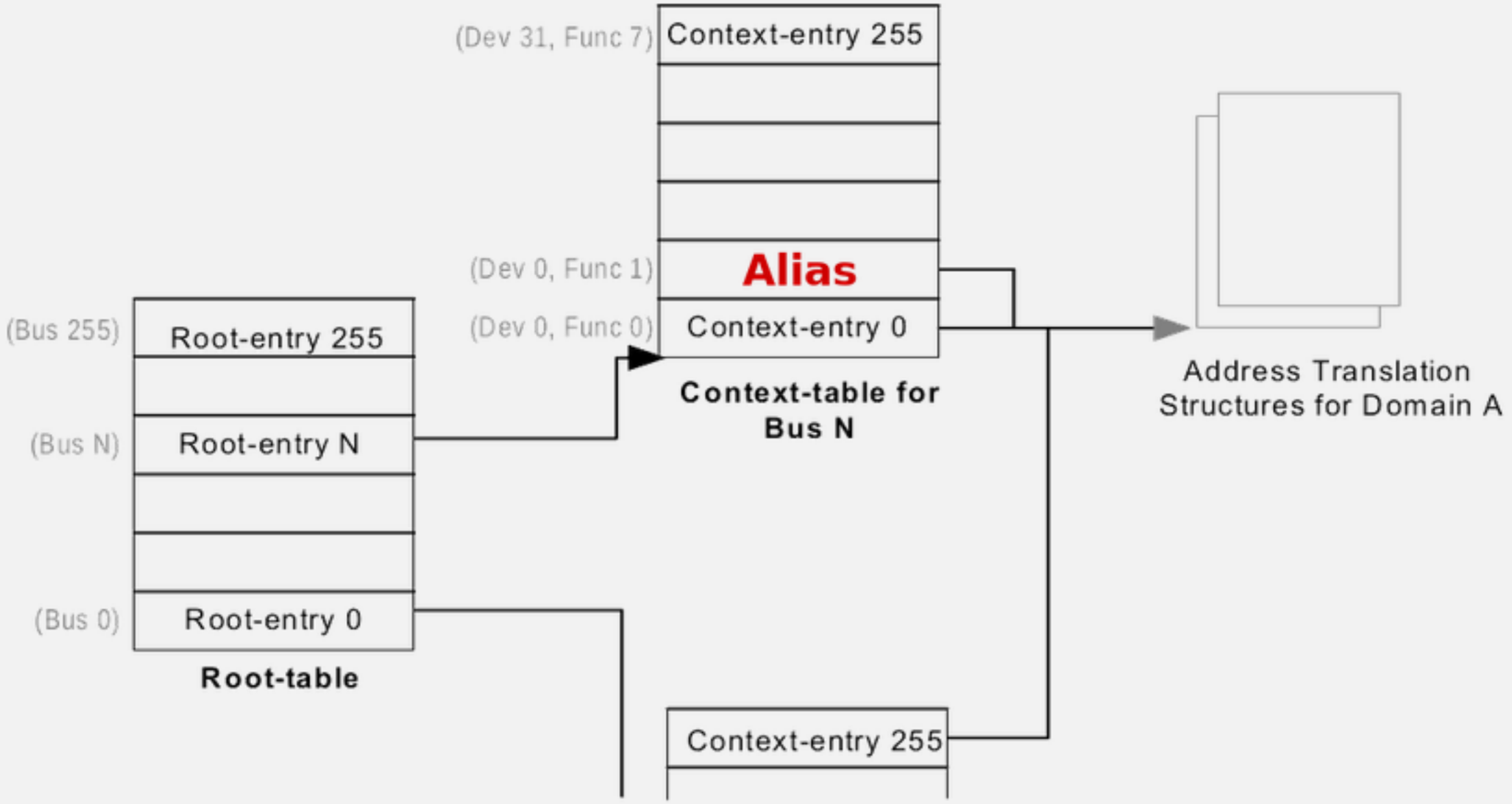
Requester ID Issues

- Bus topology can create DMA aliases
  - No requester ID on conventional PCI
- Requester ID failures:
  - *Shared* requester IDs
  - *Ghost* functions

pci_add_dma_alias()

- Linux kernel PCI quirk
- DMA aliases incorporated into IOMMU grouping
- Aliases to any device:fn on the same bus
- Context entries mapped for all aliases of device

DMA Alias


(Dev 31, Func 7)
Context-entry 255
(Dev 0, Func 1)
Alias
(Dev 0, Func 0)
Context-entry 0
Context-table for Bus N
(Bus 255)
Root-entry 255
(Bus N)
Root-entry N
(Bus 0)
Root-entry 0
Root-table
Context-entry 255
Address Translation Structures for Domain A

When to use this quirk

DMA requester ID does not strictly match the PCIe endpoint address

redhat.

How to avoid this quirk

DMA requester ID must strictly match PCIe endpoint address initiating DMA transaction

IOMMU context entries evident from topology

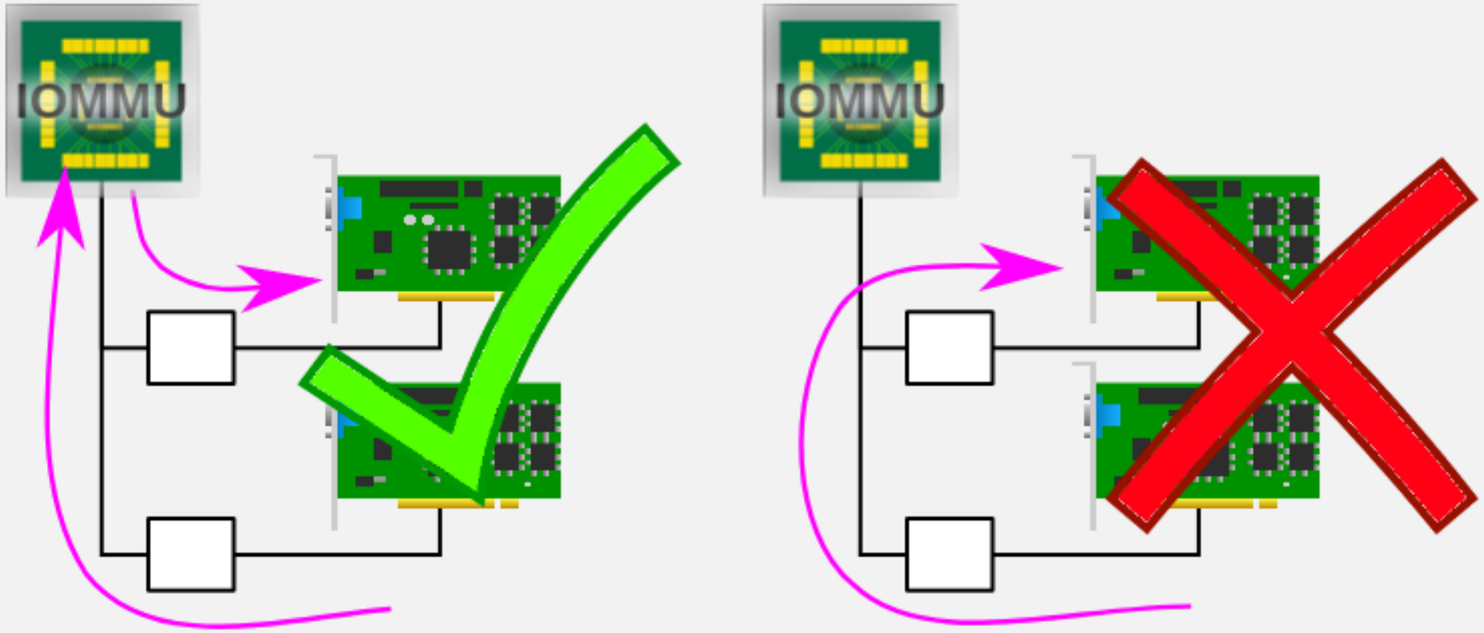

Isolation
redhat.

Requirement: Isolation

- Assigned devices must not interfere with or have access to memory except as explicitly allowed
- DMA isolation is provided through an IOMMU
- IOMMU granularity defines the minimum isolation group
- Device and bus topology features may expand this IOMMU group

redhat.

IOMMU Isolation



redhat.

Topology Isolation: PCIe ACS

- PCIe Access Control Services (ACS) capabilities define upstream packet routing
- Without ACS packet redirection possible at:
  - PCIe Downstream Ports
  - PCIe Multifunction Endpoints

Hardware PCIe ACS Support

- **Many** PCIe downstream ports lack ACS
- **Many** PCIe multifunction endpoints lack ACS
- Some ACS implementations don't match the specification standard

  IOMMU groups assume worst case routing

What quirks do we have in our toolbox for ACS isolation issues?

pci_dev_specific_acs_enabled()

- Linux kernel PCI quirk
- Allows ACS ***equivalent*** isolation to be exposed
- Hardware vendor ***guarantees*** hardware behavior
- Used by downstream ports and multifunction endpoints (many vendors, many devices)

redhat.

pci_dev_specific_enable_acs()

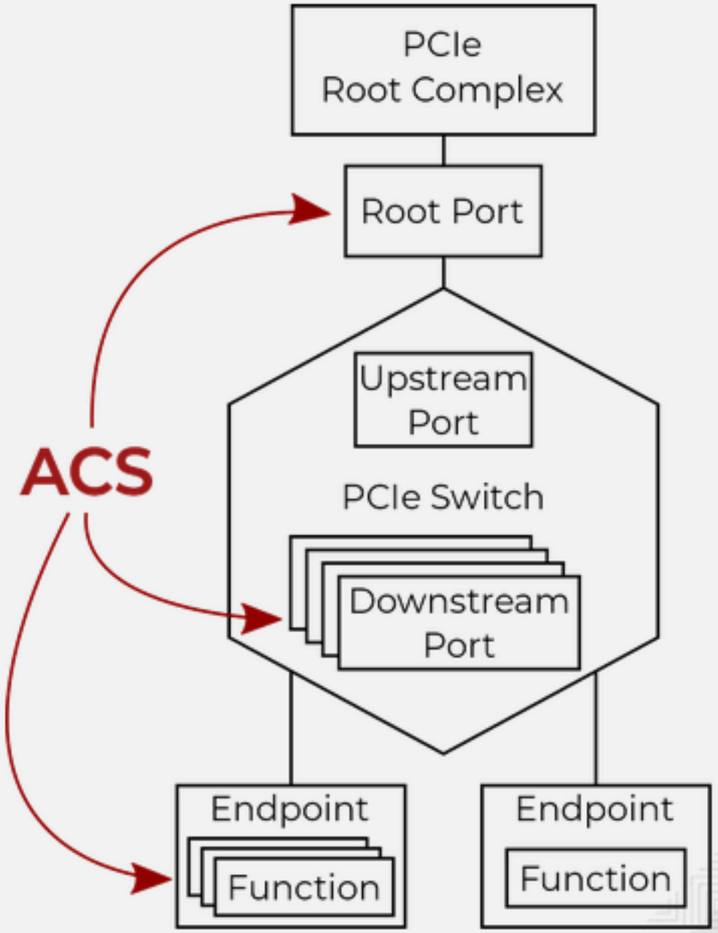
- Linux kernel PCI quirk
- Device specific setup of ACS equivalent isolation
- Often used in conjunction with pci_dev_specific_acs_enabled()

When to use these quirks

- PCIe downstream port or multifunction endpoint with broken or missing ACS
- Hardware vendor ***guarantees ACS equivalent*** routing features

How to avoid these quirks

Implement *standard* PCIe ACS capabilies in all downstream ports and multifunction endpoints



redhat.

Reproducibility

Requirement: Reproducibility

- Method to put a device into a consistent state
- Prevents data leakage between users

redhat.

How do we put a device into a consistent state for reproducible device assignment results?

redhat.

PCI Resets

- Function Level Reset (FLR)
  - PCIe & AF capabilities
- Hotplug Slot & Secondary Bus Reset
- Power Management (PM) capability reset

Reset Problems

- Bus & Slot reset scope
- Scarce FLR support outside of SR-IOV VFs
- Poorly defined PM reset
- Reset incompatibility...
  - Endpoints fail to recover from bus reset
  - Downstream ports fail during bus reset
  - Resets may be ineffective
  - Lack of any reset mechanism

redhat.

Quirks

redhat.

pci_dev_specific_reset()

- Linux kernel PCI quirk
- Override or implement PCI endpoint reset
- Few users
  - Requires device specific knowledge

PCI_DEV_FLAGS_NO_BUS_RESET

- Linux kernel PCI quirk
- Flag in *struct pci_dev* set via FIXUP quirk
- Available for endpoints and downstream ports
  - No secondary bus reset from downstream port
  - No bus resets inclusive of endpoints with flag
- Tool of last resort
  - Often eliminates only reset option

VFIOPCIDevice.resetfn

- QEMU vfio-pci device specific reset quirk
- “Second class” reset
  - Not sufficiently reliable/stable for kernel implementation

How to Avoid Reset Quirks

Implement FLR

Minimally: Test both endpoints and downstream ports with secondary bus resets

Summary

- Be mindful of device assignment requirements in hardware
- Avoid config space backdoors
- Note alignment recommendations
- Avoid topology and firmware dependencies & requirements
- Test hardware with IOMMU enabled
- Provide isolation and reset mechanisms

redhat.

Questions?

redhat®

Alex Williamson / alex.williamson@redhat.com