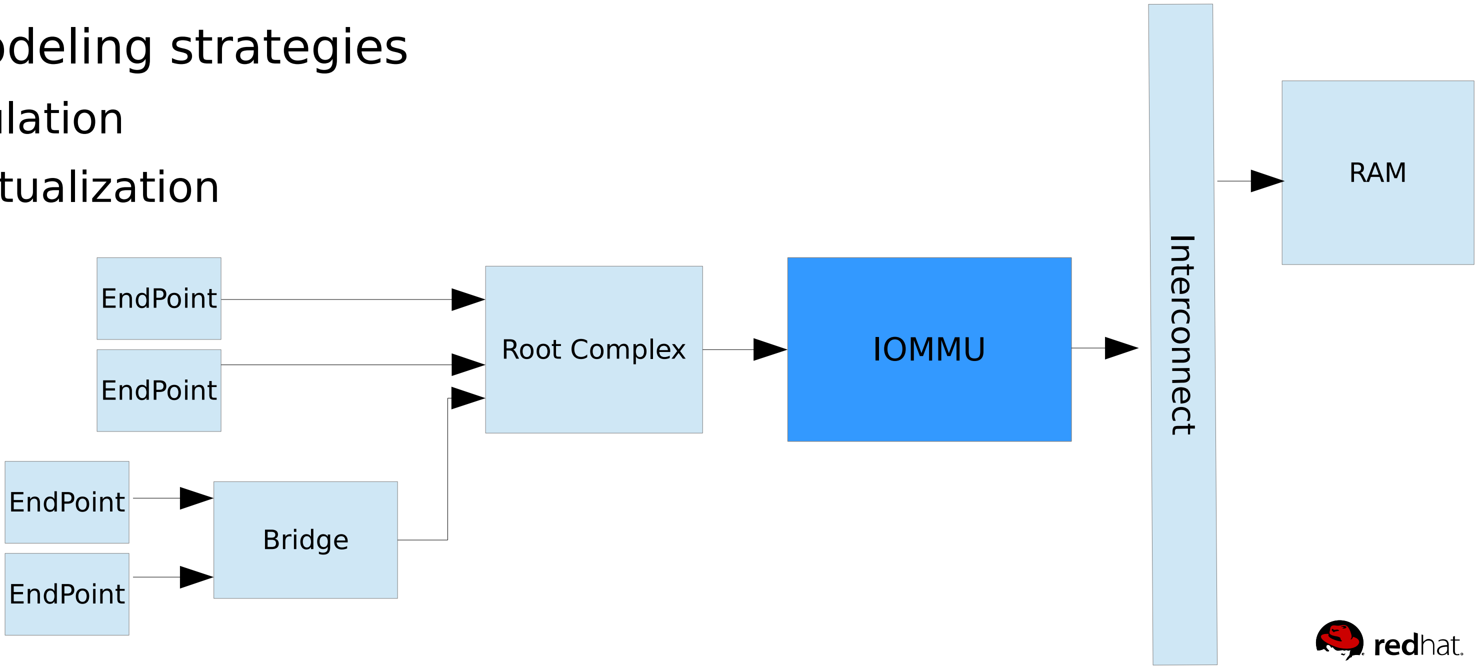
vIOMMU/ARM: full emulation and virtio-iommu approaches

Eric Auger
KVM Forum 2017

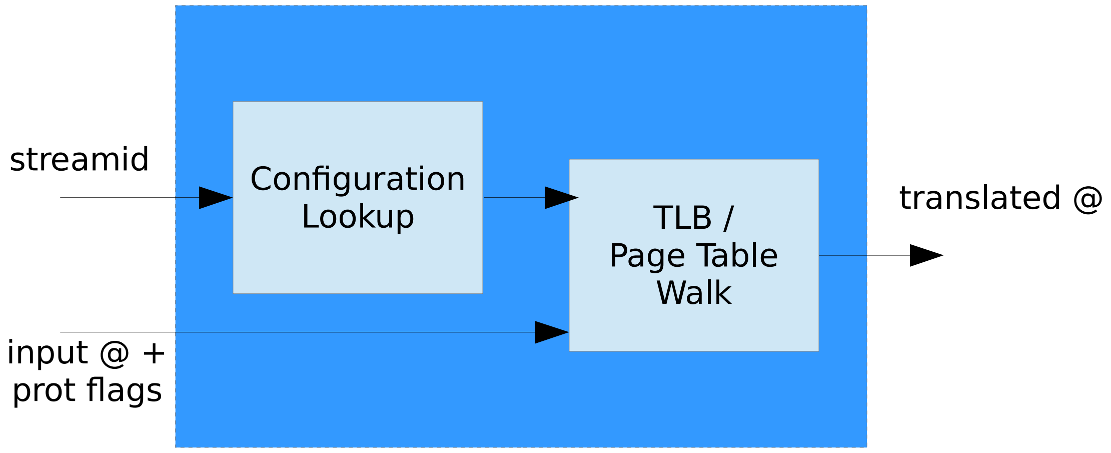
Overview

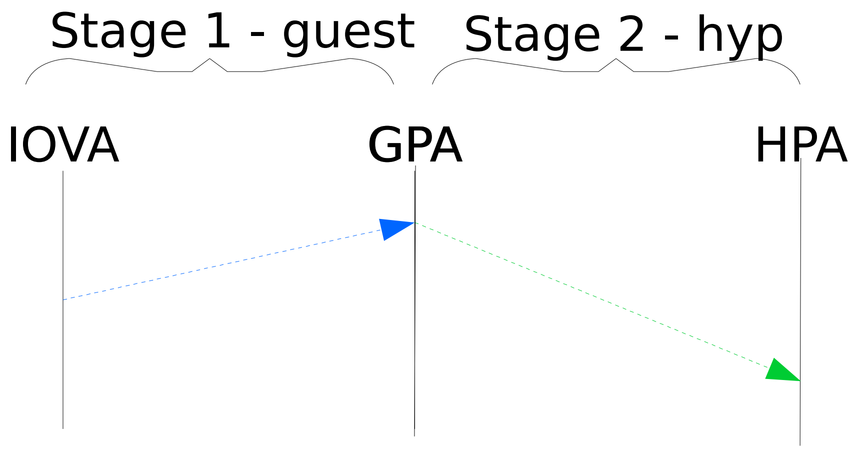
Main Goals

- Instantiate a virtual IOMMU in ARM virt machine
  - Isolate PCIe end-points
    1) VIRTIO devices
    2) VHOST devices
    3) VFIO-PCI assigned devices
  - DPDK on guest
  - Nested virtualization
- Explore Modeling strategies
  - full emulation
  - para-virtualization



Some Terminology





ARM IOMMU Emulation

ARM System MMU Family Tree

SMMU Spec	Highlights
v1	V7 VMSA* stage 2 (hyp), Register based configuration structures ARMv7 4kB, 2MB, 1GB granules
v2	+ V8 VMSA + dual stage capable + distributed design + enhanced TLBs
v3	+V8.1 VMSA + memory based configuration structures + In-memory command and event queues + PCIe ATS, PRI & PASID not backward-compatible with v2

*VMSA = Virtual Memory System Architecture

Origin, Destination, Choice

SMMUv2
maintained
out-of-tree by Xilinx

SMMUv3
initiated by Broadcom
Interrupted Contribution

Scalability
Memory-based Cfg
Memory-based Queues
PRI & ATS

UPSTREAM

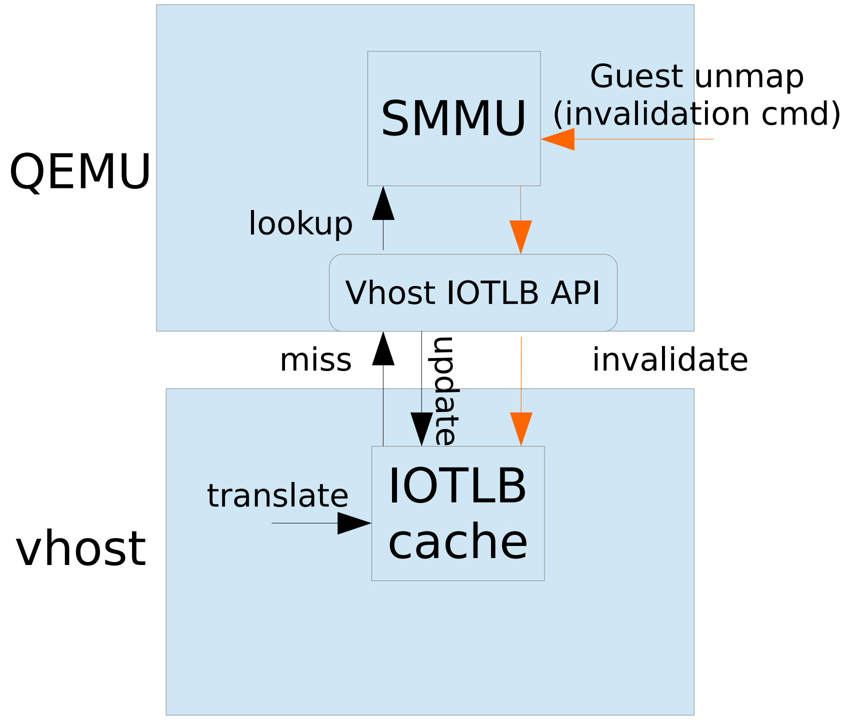
ENABLE VHOST and VFIO USE CASES

SMMUv3 Emulation Code

- Stage 1 or stage 2
- AArch64 State translation table format only
- DT & ACPI probing
- limited set of features (no PCIe ATS PASIDS PRI, no MSI, no TZ...)

	LOC	Content
common (model agnostic)	600	IOMMU memory region infra, page table walk
smmuv3 specific	1600	MMIO, config decoding (STE, CD), IRQ, cmd/event queue)
sysbus dynamic instantiation	200	sysbus-fdt, virt, virt-acpi-build
Total	2400	

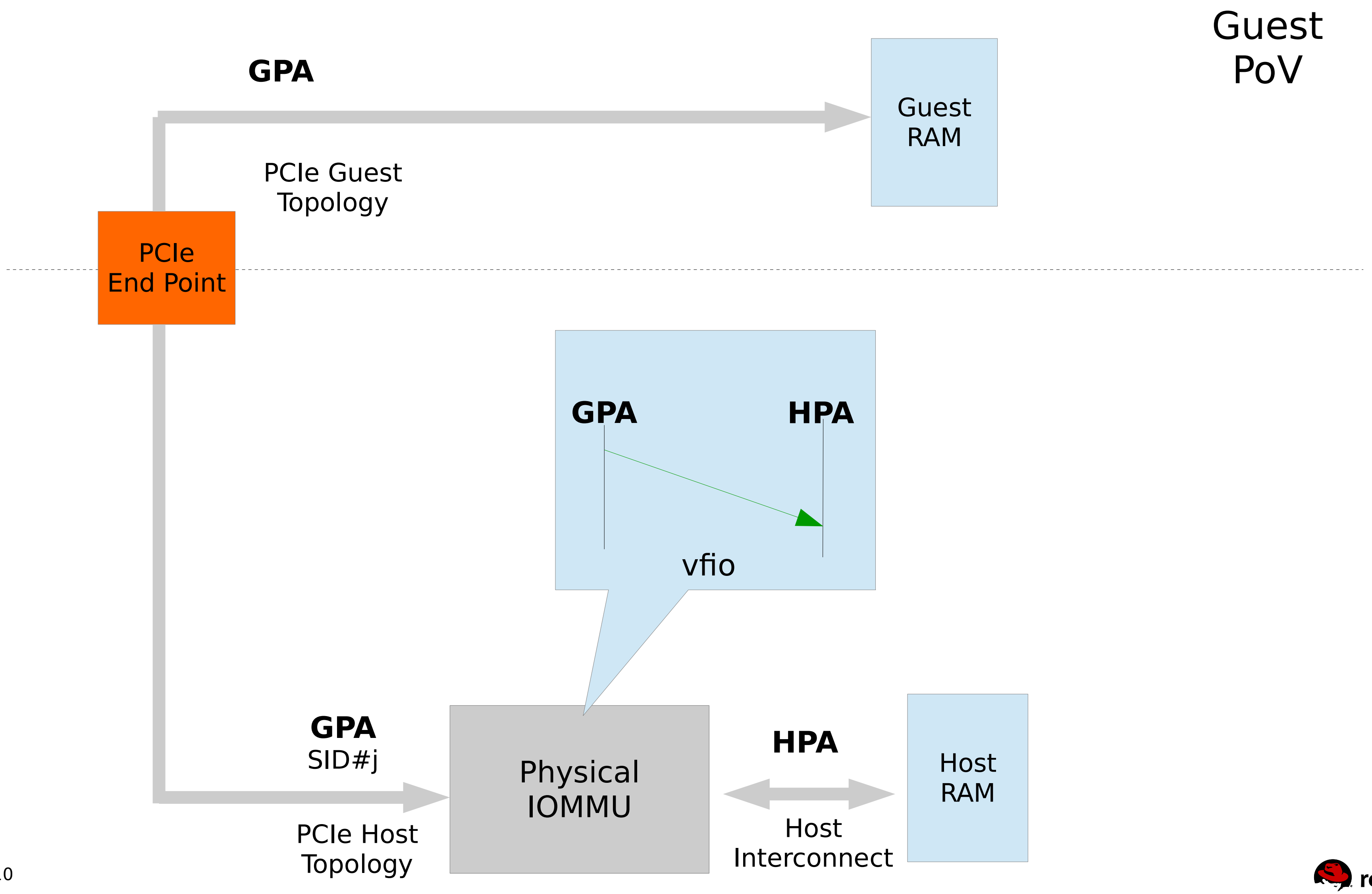
Vhost Enablement



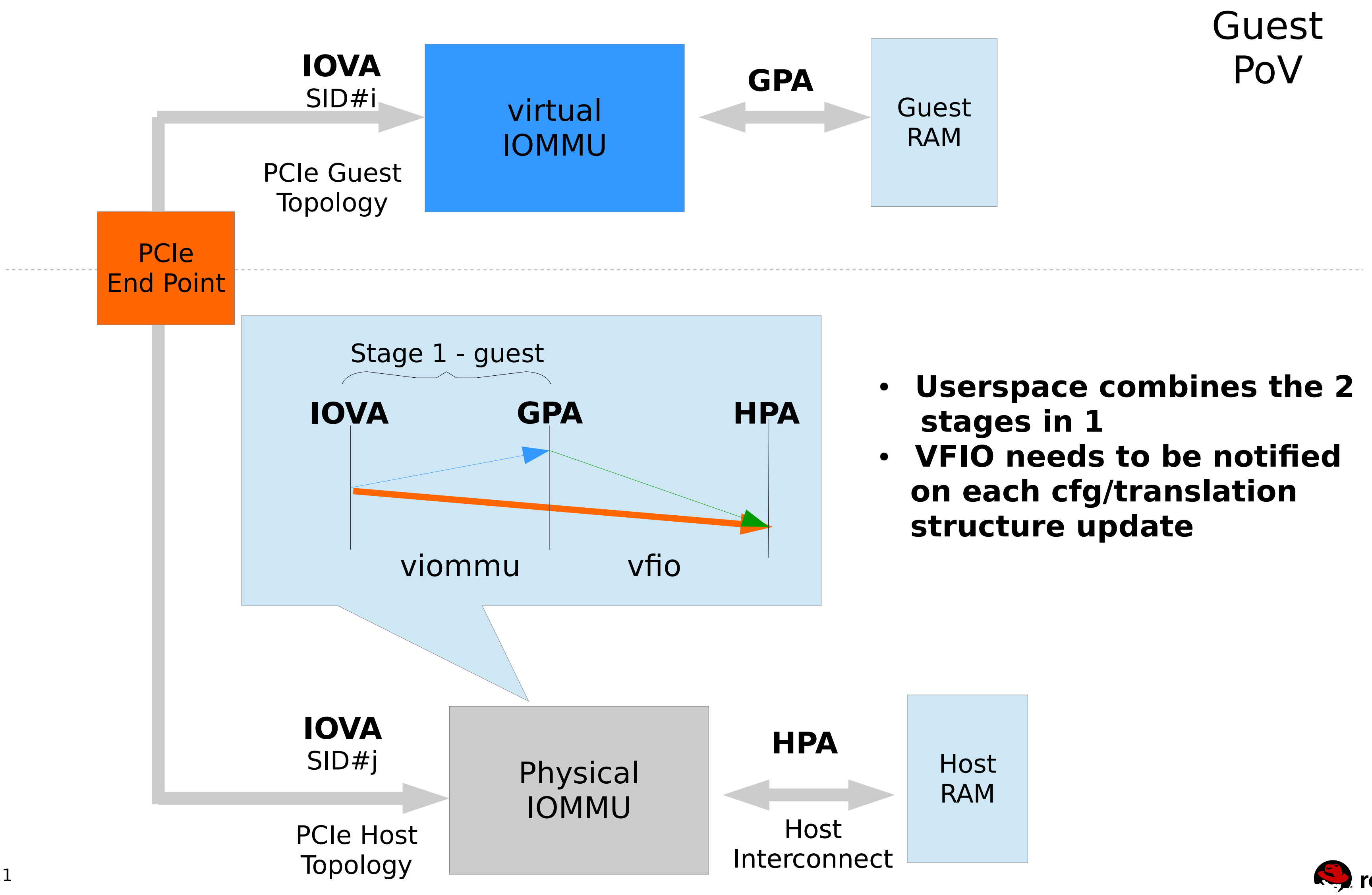
- Call IOMMU Notifiers on invalidation commands
- + 150 LOC

Full Details in 2016 “Vhost and VIOMMU” KVM Forum Presentation
Jason Wang (Wei Xu), Peter Xu

VFIO Integration : No viommu



VFIO Integration: viommu



- **Userspace combines the 2 stages in 1**
- **VFIO needs to be notified on each cfg/translation structure update**

SMMU VFIO Integration Challenges

	INTEL DMAR	ARM SMMU
1) Mean to force the driver to send invalidation commands for all cfg/translation structure update	✓	☐
2) Mean to invalidate more than 1 granule at a time	✓	☐

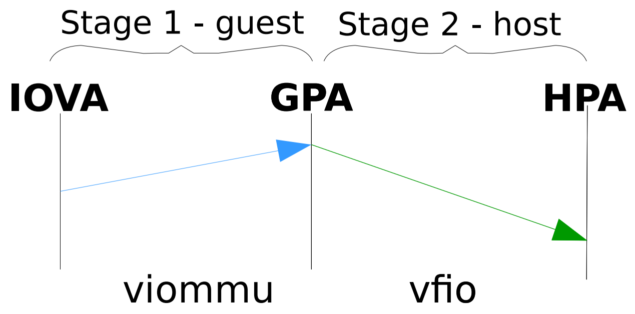
1) “Caching Mode” SMMUv3 driver option set by a FW quirk

2) Implementation defined invalidation command with addr_mask

- Shadow page tables
- Use 2 physical stages
- Use VIRTIO-IOMMU

Use 2 physical stages

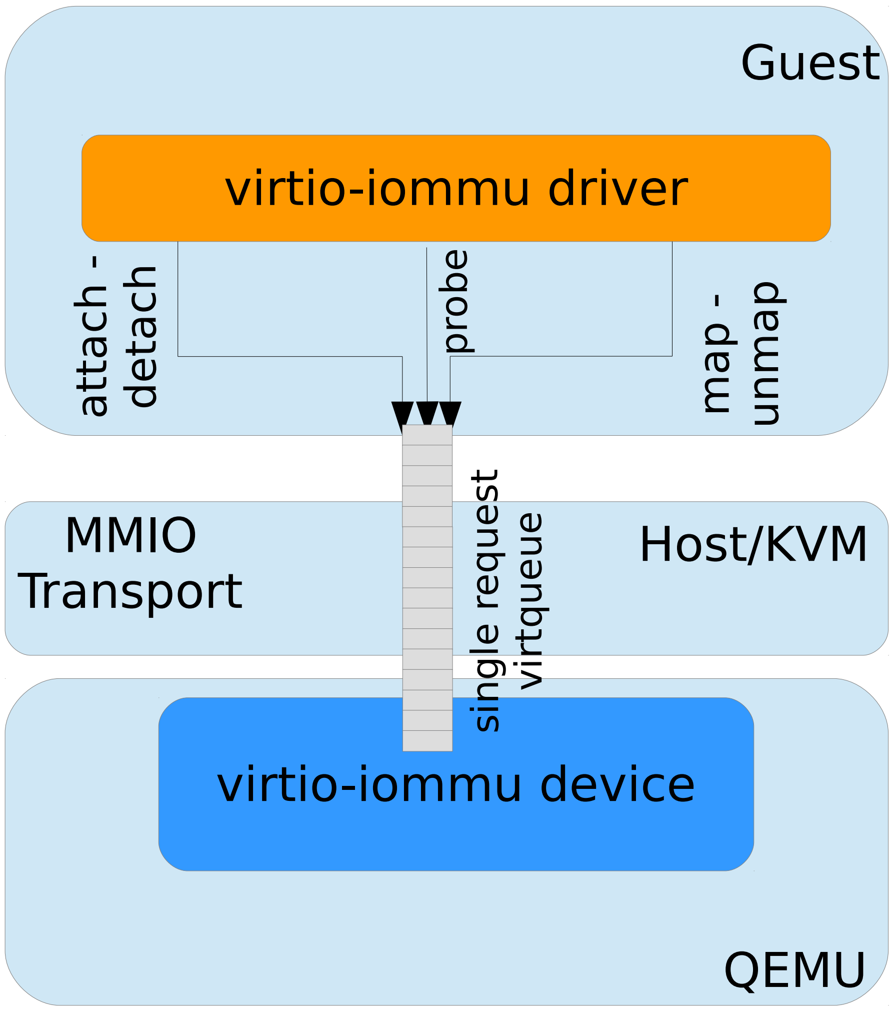
- Guest owns stage 1 tables and context descriptors
  - Host does not need to be notified on each change anymore
  - Removes the need for the FW quirk
- Need to teach VFIO to use stage 2
- Still a lot to SW virtualize: Stream tables, registers, queues
- Miss an API to pass STE info
- Miss an Error Reporting API
- Related to SVM discussions …



VIRTIO-IOMMU

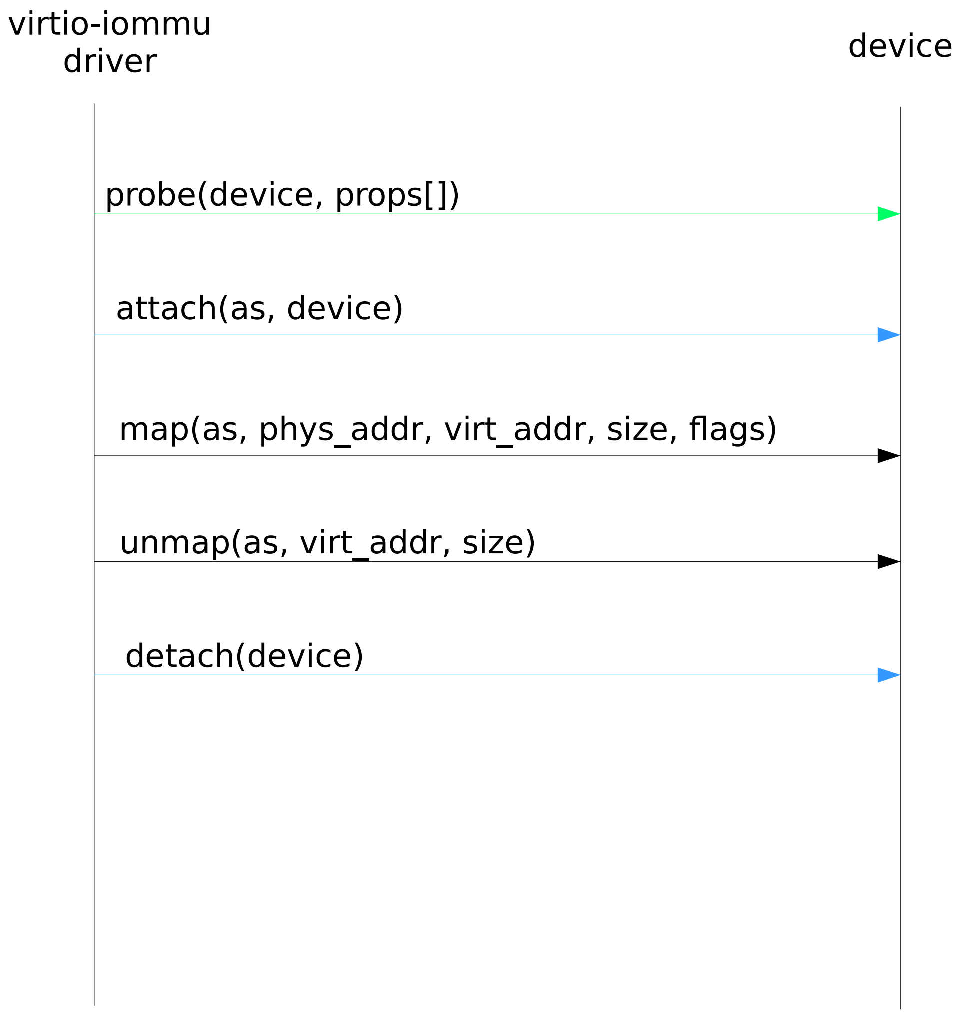
Overview

- rev 0.1 draft, April 2017, ARM
  + FW notes
  + kvm-tool example device
  + longer term vision
- rev 0.4 draft, Aug 2017
- QEMU virtio-iommu device



Device Operations

- Device is an identifier unique to the IOMMU
- An address space is a collection of mappings
- Devices attached to the same address space share mappings
- if the device exposes the feature, the driver sends probe requests on all devices attached to the IOMMU



QEMU VIRTIO-IOMMU Device

- Dynamic instantiation in ARM virt (dt mode)
- VIRTIO, VHOST, VFIO, DPDK use cases

	LOC	
virtio-iommu device	980	infra + request decoding + mapping data structures
vhost/vfio integration	220	IOMMU notifiers
machvirt dynamic instantiation	100	dt only
Total	1300	

virtio-iommu driver: 1350 LOC

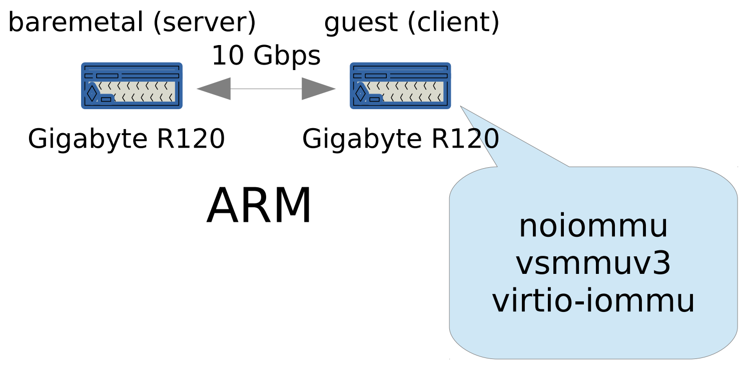
x86 Prototype

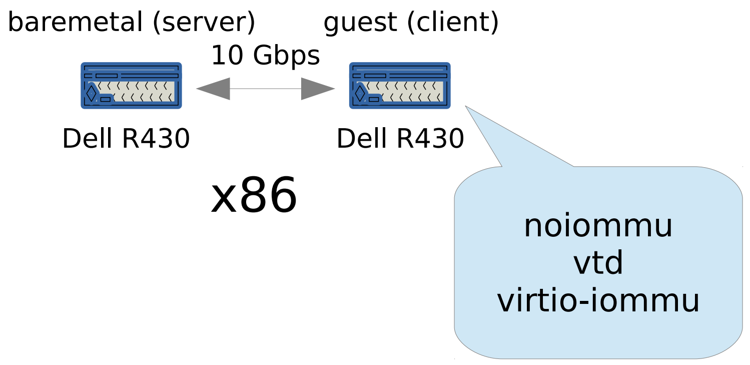
- Hacky Integration (Red Hat Virt Team, Peter Xu)
  - QEMU
    - Instantiate 1 virtio MMIO bus
    - Bypass MSI region in virtio-iommu device
  - Guest Kernel
    - Pass device mmio window via boot param (no FW handling)
    - Limited to a single virtio-iommu
    - Implement dma_map_ops in virtio-iommu driver
    - Use PCI BDF as device id
    - Remove virtio-iommu platform bus related code

Epilogue

First Performance Figures

- Netperf/iperf TCP throughput measurements between 2 machines
- Dynamic mappings only (guest feat. a single virtio-net-pci device)
- No tuning





Gigabyte R120, T34 (1U Server),
Cavium CN88xx, 1.8 Ghz,
32 procs, 32 cores
64 GB RAM

Dell R430,
Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz,
32 proc, 16 cores
32 GB RAM

Performance: ARM benchmarks

	netperf		iperf3	
Guest Config	Rx (Mbps) vhost off / on	Tx (Mbps) vhost off / on	Rx (Mbps) vhost off / on	Tx (Mbps) vhost off / on
noiommu	4126 / 3924	5070 / 5011	4290 / 3950	5120 / 5160
smmuv3	1000 / 1410	238 / 232	955 / 1390	706 / 692
smmuv3,cm	560 / 734	85 / 86	631 / 740	352 / 353
virtio-iommu	970 / 738	102 / 97	993 / 693	420 / 464

- Low performance overall with virtual iommu, especially in Tx
  - smmuv3 performs better than virtio-iommu
    - when vhost=on
    - in Tx
  - Both perform similarly in Rx when vhost=off
- Better performance observed on next generation ARM64 server
  - Max Rx/Tx with smmuv3: 2800 Mbps/887 Mbps (42%/11% of noiommu cfg)
  - Same perf ratio between smmuv3 and virtio-iommu

Performance: x86 benchmarks

	netperf		iperf3	
Guest Config (vhost=off)	Rx (Mbps)	Tx (Mbps)	Rx (Mbps)	Tx (Mbps)
noiommu	9245 (100%)	9404 (100%)	9301 (100%)	9400 (100%)
vt-d (deferred invalidation)	7473 (81%)	9360 (100%)	7300 (78%)	9370 (100%)
vt-d (strict)	3058 (33%)	2100 (22%)	3140 (34%)	6320 (67%)
vt-d (strict + caching mode)	2180 (24%)	1179 (13%)	2200 (24%)	3770 (40%)
virtio-iommu	924 (10%)	464 (5%)	1600 (17%)	924 (10%)

- Indicative but not fair
  - virtio-iommu driver does not implement any optimization yet
  - Behaves like vtd strict + caching mode
- Looming Optimizations:
  - Deferred IOTLB invalidation
  - Page Sharing avoids explicit mappings
  - QEMU device IOTLB emulation
  - vhost-iommu

Some Pros & Cons

vSMMUv3	virtio-iommu
++ unmodified guest ++ smmuv3 driver reuse (good maturity) ++ better perf in virtio/vhost + plug & play FW probing - QEMU device is more complex and incomplete -- ARM SMMU Model specific -- Some key enablers are missing in the HW spec for VFIO integration: only for virtio/vhost	++ generic/ reusable on different archs ++ extensible API to support high end features & query host properties ++ vhost allows in-kernel emulation + simpler QEMU device, simpler driver - virtio-mmio based - virtio-iommu device will include some arch specific hooks - mapping structures duplicated in host & guest --para-virt (issues with non Linux OSes) -- OASIS and ACPI specification efforts (IORT vs. AMD IVRS, DMAR and sub-tables) -- Driver upstream effort (low maturity) -- explicit map brings overhead in virtio/vhost use case

Next

- vSMMUv3 & virtio-iommu now support standard use cases
  - Please test & report bug/performance issues
- virtio-iommu spec/ACPI proposal review
  - Discuss new extensions
  - Follow-up SVM and guest fault injection related work
- Code Review
- Implement various optimization strategies

THANK YOU