redhat.

Vhost and VIOMMU

Jason Wang <jasowang@redhat.com>
(Wei Xu <wexu@redhat.com>)
Peter Xu <peterx@redhat.com>

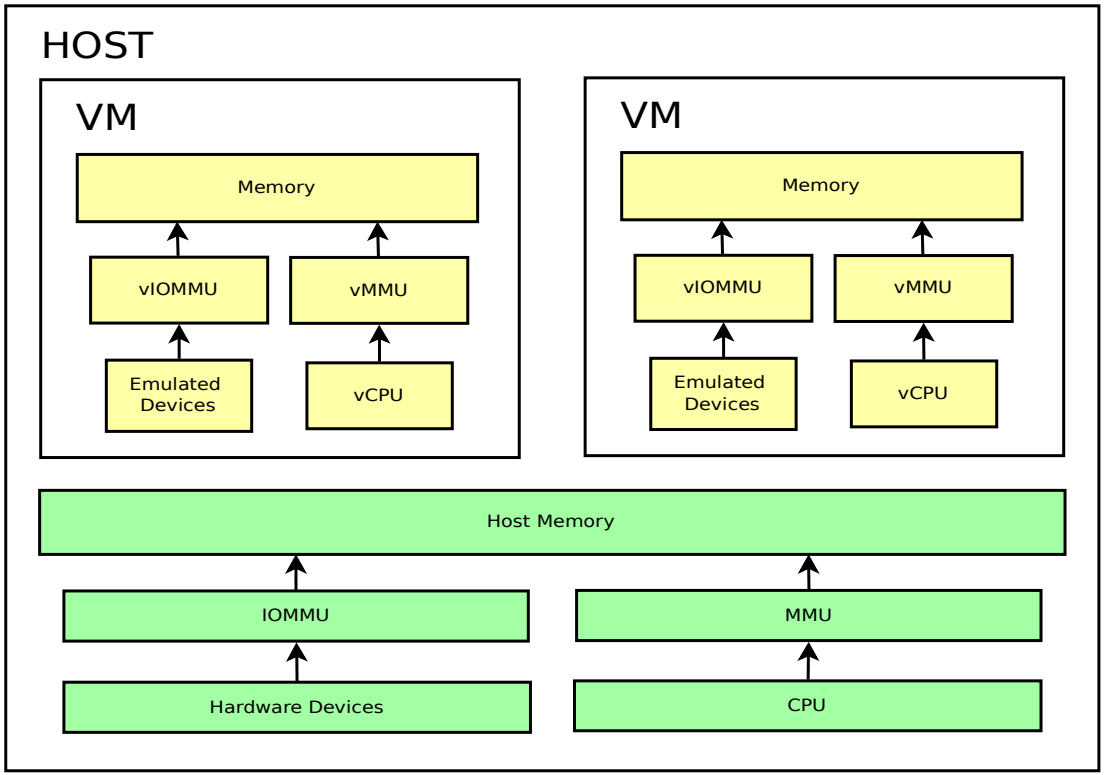
Agenda

- IOMMU & Qemu vIOMMU background
- Motivation of secure virtio
- DMAR (DMA Remapping)
  - Design Overview
  - Implementation illustration
  - Performance optimization – vhost device iotlb
- IR (Interrupt Remapping)
- Performance results & status

IOMMU & Qemu vIOMMU Revisit

- What is IOMMU?
  - A hardware component provides two main functions: IO Translation and Device Isolation.
- How IO Translation and Device Isolation are supported by IOMMU
  - DMA Remapping(DMAR), IO space address presented by devices are translated to physical address coupled with access permission on the fly, so the ability of devices are limited to access specific regions of memory.
  - Interrupt Remapping (IR), Some architectures also support interrupt remapping, in a manner similar to memory remapping.
- What's qemu vIOMMU?
  - An emulated IOMMU which behaves as a real one.
  - The functionality is always a subset of a physical unit depending on implementation.
  - Only Intel, ppc, sun4m iommus are support in qemu currently.

IOMMU and vIOMMU

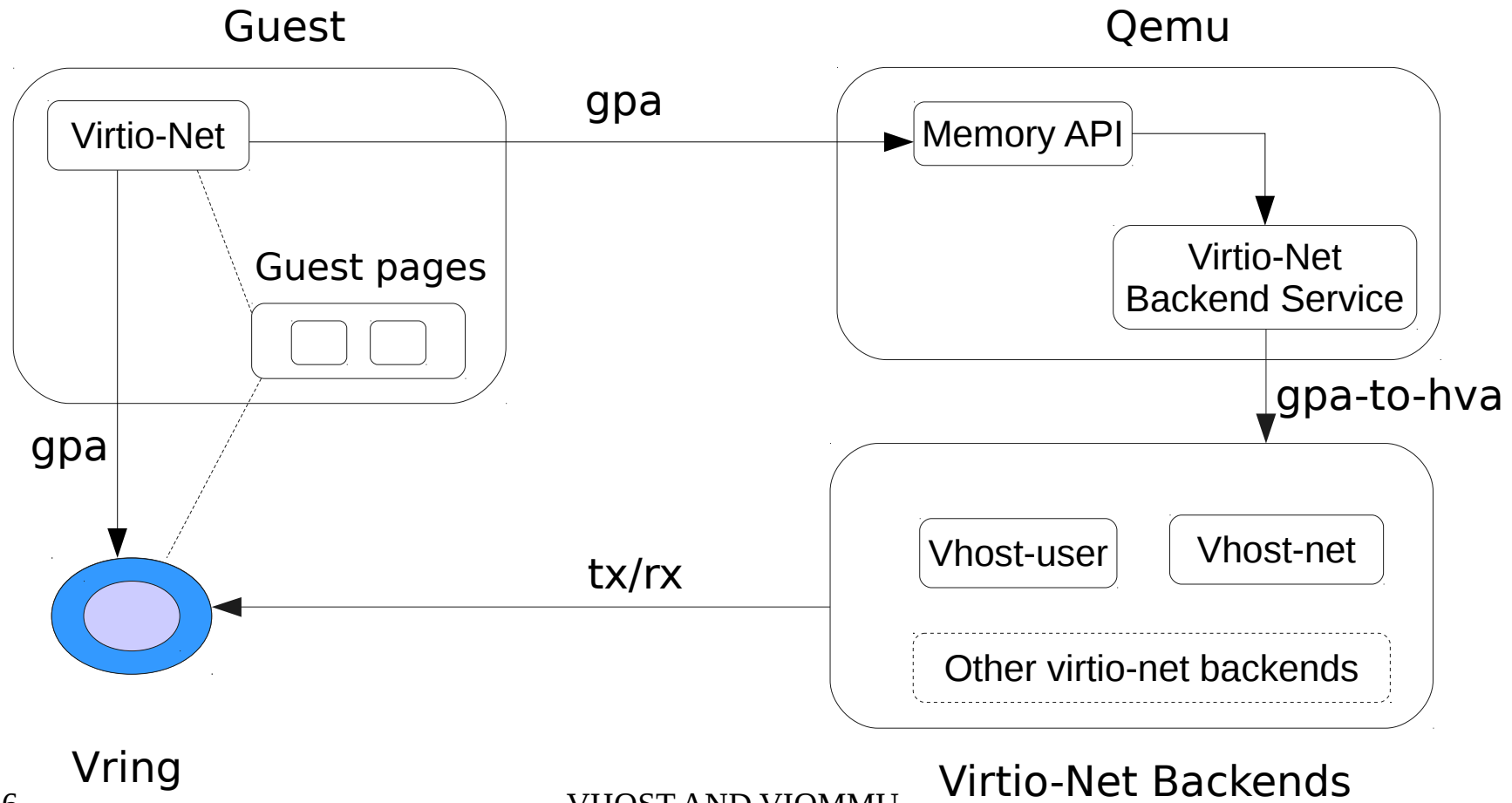


Motivation

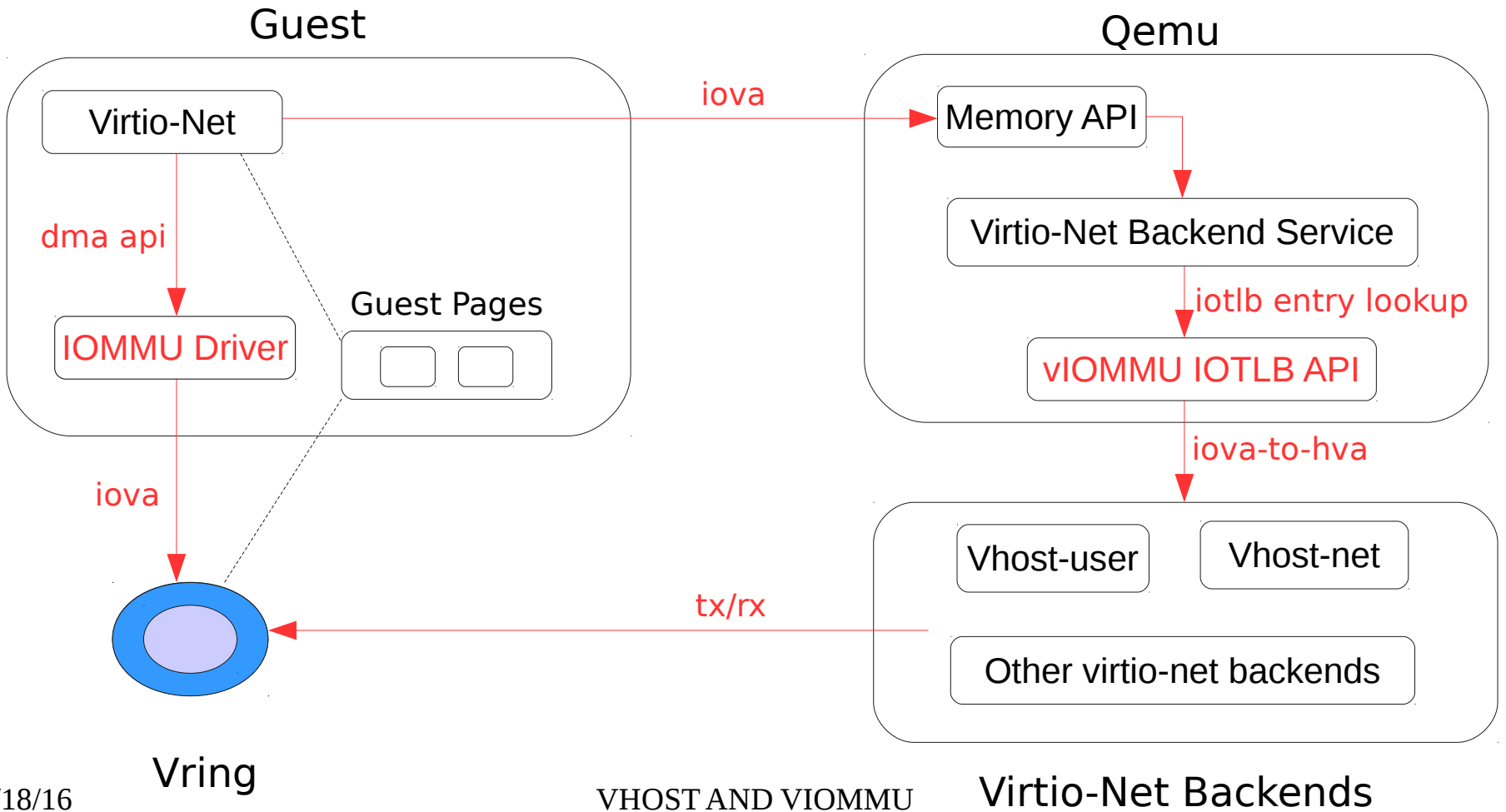
- Security, Securtiy and security.
- DPDK: The Userspace Polling-Mode drivers (DPDK) for virtio net devices are vastly used in NFV.
- Vhost is the popular backend for most of user cases.
- Vhost is still out of IOMMU scope.

DMA Remapping (DMAR)

Virtio-Net Device Address Space Overview



Design of Secure Virtio-Net Device Driver



Implementation: Guest

- Guest
  - Boot guest with a vIOMMU assigned.

  - VIRTIO_F_IOMMU_PLATFORM, if this feature bit is provided in the device, then the guest virtio driver is forced to use dma api to manage all corresponding dma memory access, otherwise the device will be disabled by system compulsorily.

Implementation: Qemu and Backends

- Qemu
  - DMA address translation for vIOMMU has been fully supported, unfortunately, virtio-pci devices is still using memory address space and never use iova at all, switch to use dma address(iova).

- Backends
  - All address access to vring must be translated from guest iova to hva, this is done via iotlb lookup with interfering of vIOMMU.

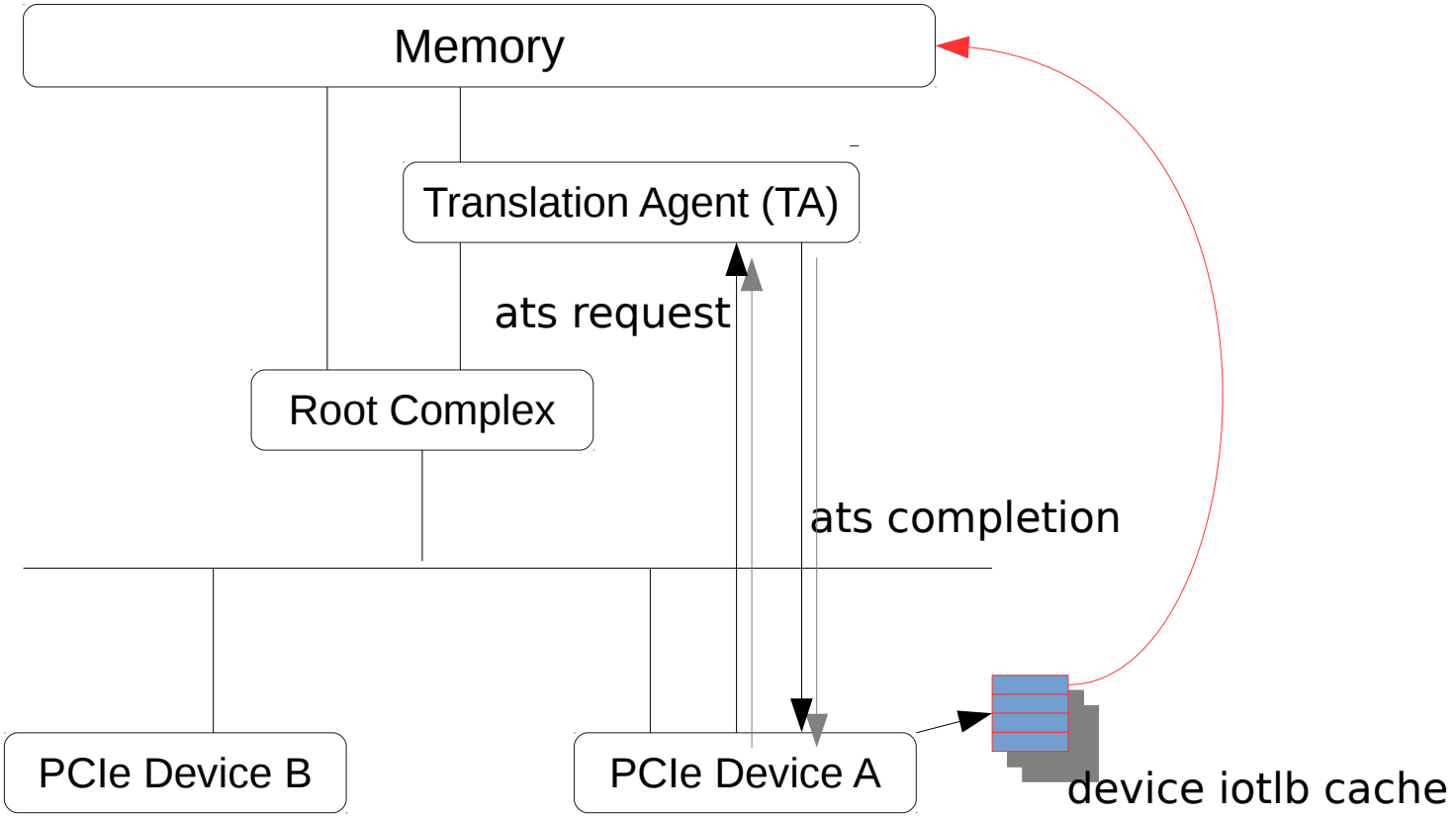
More optimization: Vhost Device IOTLB Cache

- Why it comes to vhost?
  - Vhost-net is the most powerful and reliable in-kernel network backend, and is widely used as a preferred backend.

- What problem does vhost encounter?
  - IOTLB api of vIOMMU is implemented in qemu, while vhost works in kernel, high frequency of iotlb translations which traverse between kernel and userspace will impact performance dramatically.

- How does vhost survive?
  - Kernel-Side device iotlb cache(ATS).

Address Translation Services(ATS) Overview

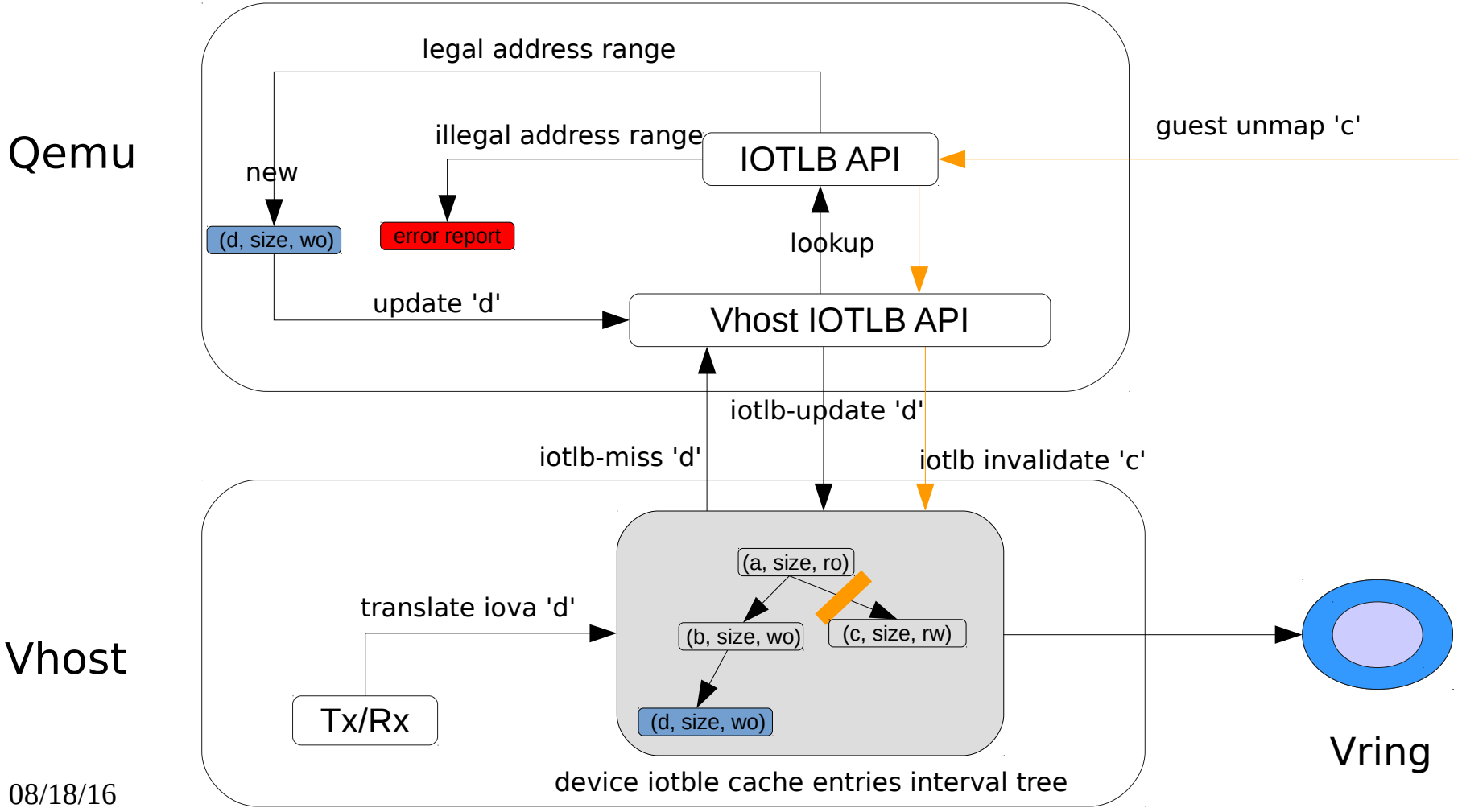


Why Address Translation Services(ATS)?

- Alternative
  - An individual VT-d in vhost, drawbacks:
    - Code duplication.
    - Vendor and architecture specific.
    - New api for error reporting.

- Benefits of ATS
  - PCIe spec
  - Platform independent.
  - Easily achieved based on current iommu infrastructure.

Vhost Device IOTLB Cache Workflow



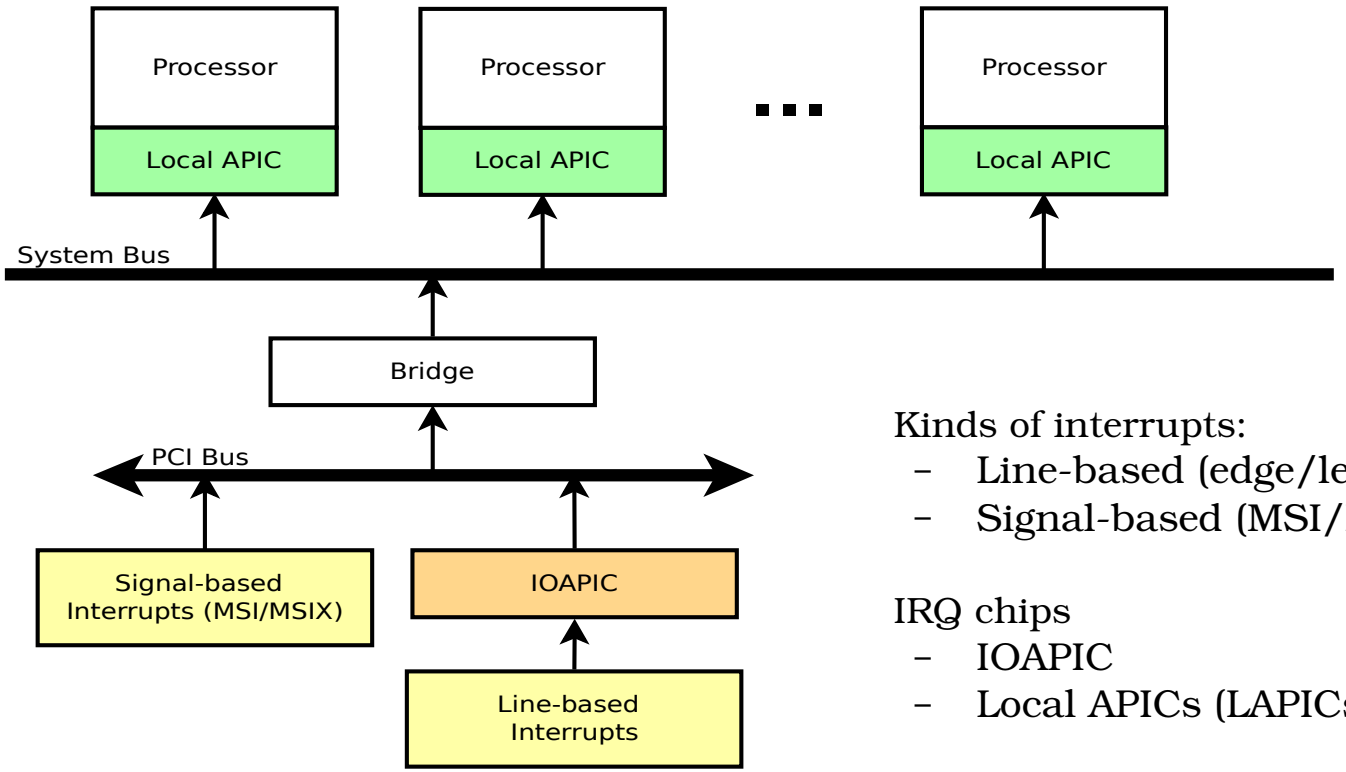
Vhost Device IOTLB Implementation Summary

- Implementation
  - Save device iotlb cache entries in kernel.
  - Lookup entry from the cache when accessing virtio buffers.
  - Request qemu to translate for any tlb miss on demand.
  - Process update/invalidate message from qemu and manage the kernel cache correctly.

- Data Structure and Userspace/Kernel Interface
  - An interval tree is chosen to save the dynamica device iotlb caches.
  - A message mechanism via vhost 'fd' read/write is used to pass vATS request and reply.

Interrupt Remapping (IR)

X86 system interrupts



Kinds of interrupts:
- Line-based (edge/level)
- Signal-based (MSI/MSI-X)

IRQ chips
- IOAPIC
- Local APICs (LAPICs)

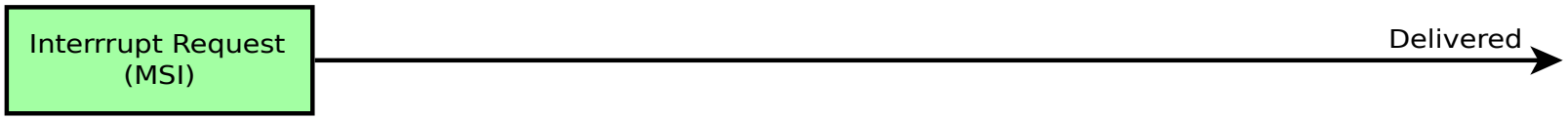
IR challenges for vhost

- Interrupt remapping (IR) still not supported for x86 vIOMMU
  - MSI and IOAPIC interrupts
- Kernel irqchip support:
  - How to define interface between user and kernel space?
  - How to enable vhost fast irq path (irqfd)?
- Performance impact?
- Interrupt caching

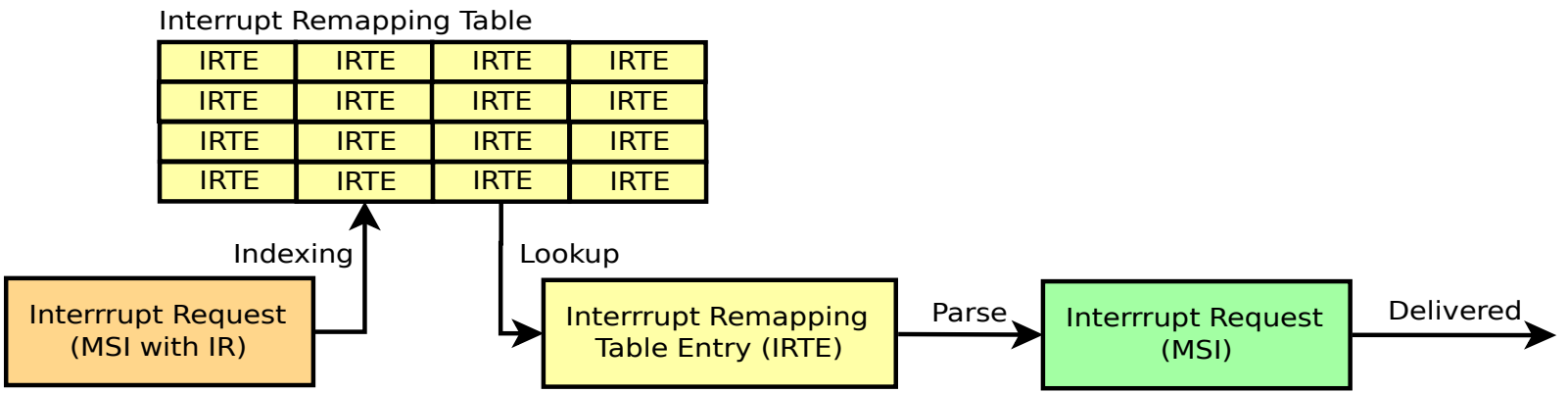
IOAPIC interrupt delivery

- Workflow before IR:
  - Fill in **IOAPIC entry** with interrupt information (trigger mode, destination ID, destination mode, etc.).
  - When line triggered, interrupt sent to CPU with information stored in **IOAPIC entry**.
- Workflow after IR *(IRTE: Interrupt Remapping Table Entry)*:
  - Fill in **IRTE** with interrupt information (in system memory).
  - Fill in **IOAPIC entry** with IRTE index.
  - When line triggered, fetch IRTE index from **IOAPIC entry**, send the interrupt with information stored in specific **IRTE**.

MSI/MSI-X delivery



MSI Delivery without IR



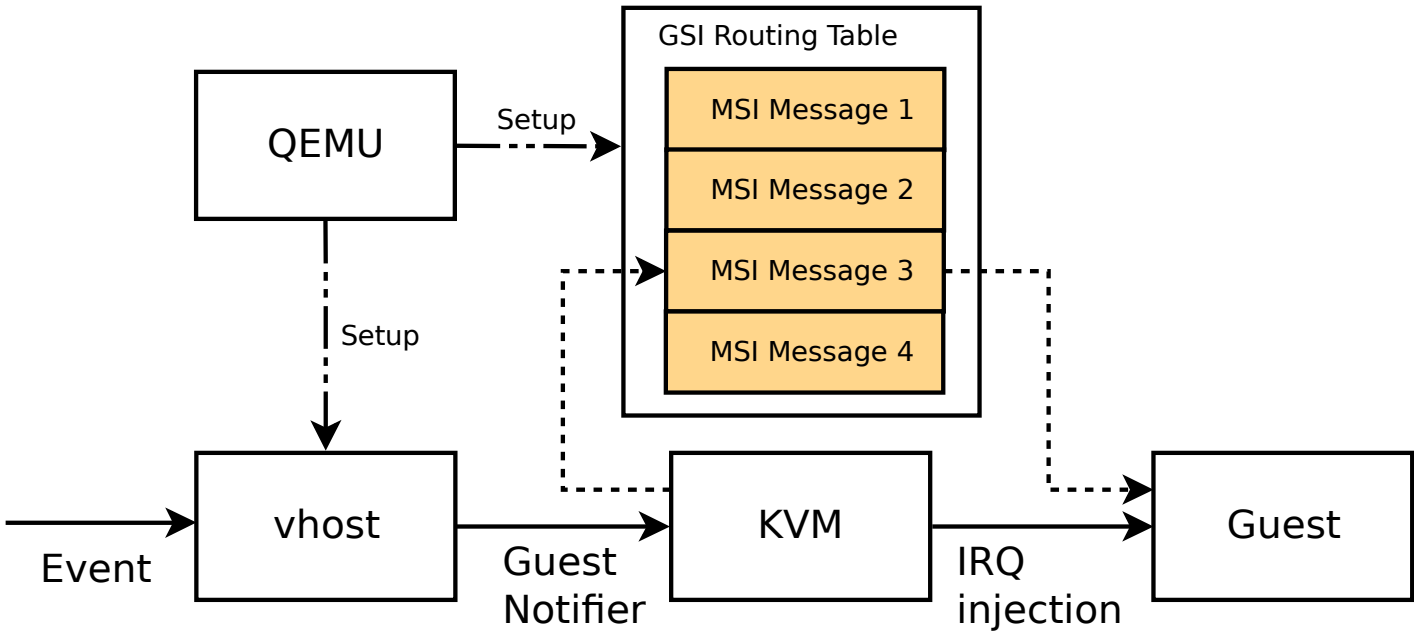
MSI Delivery with IR

IR with kernel-irqchip

- We want interrupts “as fast as before”.
- Current implementation:
  - Leverage existing GSI routing table in KVM
  - Instead of translate “on the fly”, translate during setup
  - Easy to implement (no KVM change required)
  - Little performance impact (slow setup, fast delivery)
  - Only support “split|off” kernel irqchip, not “on”

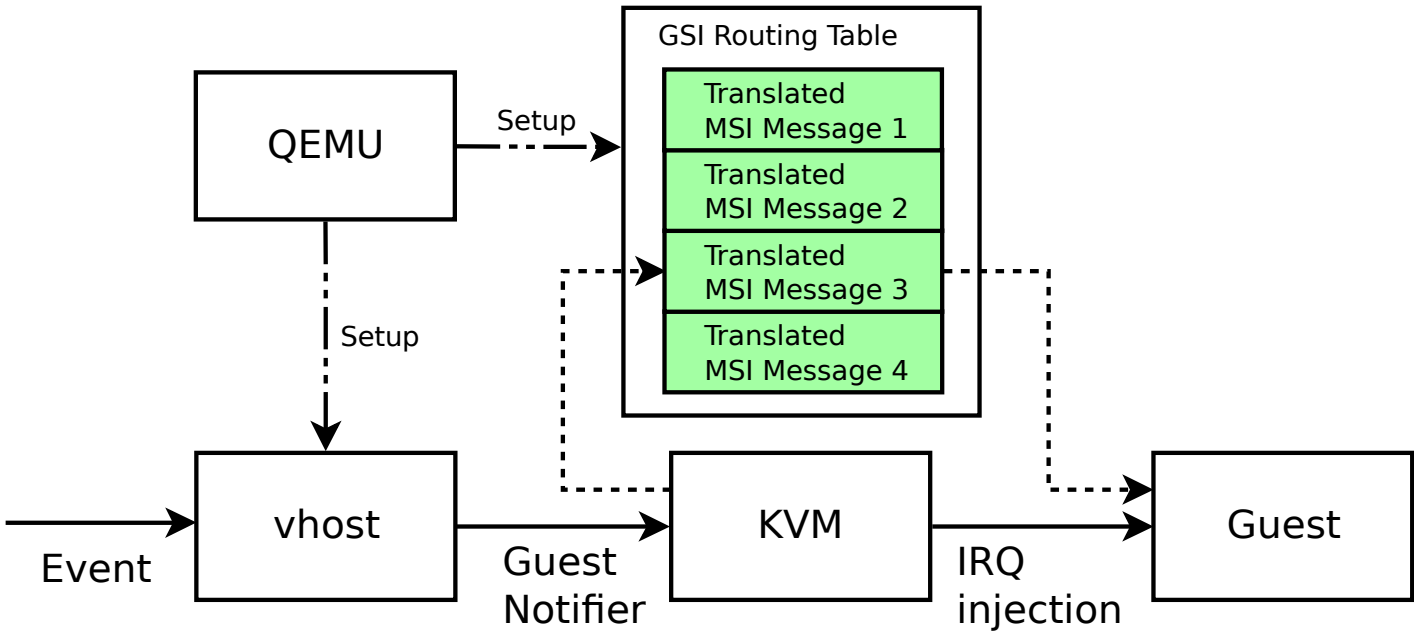
Remap irqfd interrupts

- Fast IRQ path for vhost devices: *without remapping*



Remap irqfd interrupts (cont.)

- Fast IRQ path for vhost devices: *with remapping*



All in all...

- To boot guest with DMAR and IR enabled:

  (Possibly one extra flag to enable DMAR for guest virtio driver)

```
qemu-system-x86_64 -M q35,accel=kvm,kernel-irqchip=split \
    -device intel-iommu,intremap=on \
    -netdev tap,id=tap1,script=no,downscript=no,vhost=on \
    -device virtio-net-pci,netdev=tap1,disable-modern=off,ats=on
```

Vhost + VIOMMU Performance

- For dynamic DMA mapping (e.g., using generic Linux kernel drivers):
  - Performance dropped drastically
  - TCP_STREAM: 24500 Mbps → 600 Mbps
  - TCP_RR: 25000 trans/s → 11600 trans/s
- For static DMA mapping (e.g., DPDK based application like l2fwd)
  - Around 5% performance drop for throughput (pktgen)
  - Still more work TBD…

Current status & TBDs

- DMAR/IR upstream status:
  - QEMU: IR merged (Peter Xu), DMAR still RFC (Jason Wang will post formal patch soon)
  - Vhost & Virtio driver: merged (Michael S. Tsirkin/Jason Wang)
  - DPDK: vhost-user IOTLB is being developed (Victor Kaplansky)
- TBDs
  - Performance tuning for DMAR
  - Quite a few enhancements for IR: explicit cache invalidations, better error handling, etc.

Thanks!

Appendix

Kernel-irqchip: a review

- Command line interface:

```
qemu-system-x86_64 -M q35,kernel-irqchip={on|off|split}
```

- Supported modes

Mode	IOAPIC	APIC
“ON”	In kernel	
“SPLIT”	In userspace	In kernel
	~~In kernel~~	~~In userspace~~
“OFF”	In userspace	