Device Assignment with Nested Guests and DPDK

Peter Xu <peterx@redhat.com>
Red Hat Virtualization Team

Agenda

- Backgrounds
- Problems
  - Unsafe userspace device drivers
  - Nested device assignments
- Solution
- Status updates, known issues

Backgrounds

Backgrounds

- What is this talk about
  - DMA of assigned devices (nothing about configuration spaces, IRQs, MMIOs...), and...
  - vIOMMU!
- What is vIOMMU?
  - IOMMU is “MMU for I/O”
  - vIOMMU is the emulated IOMMU in the guests
- What is device assignment?
  - The fastest device (always) to do IOs in the guest! (at least when without a vIOMMU in the guest...)

QEMU, Device Assignment & vIOMMU

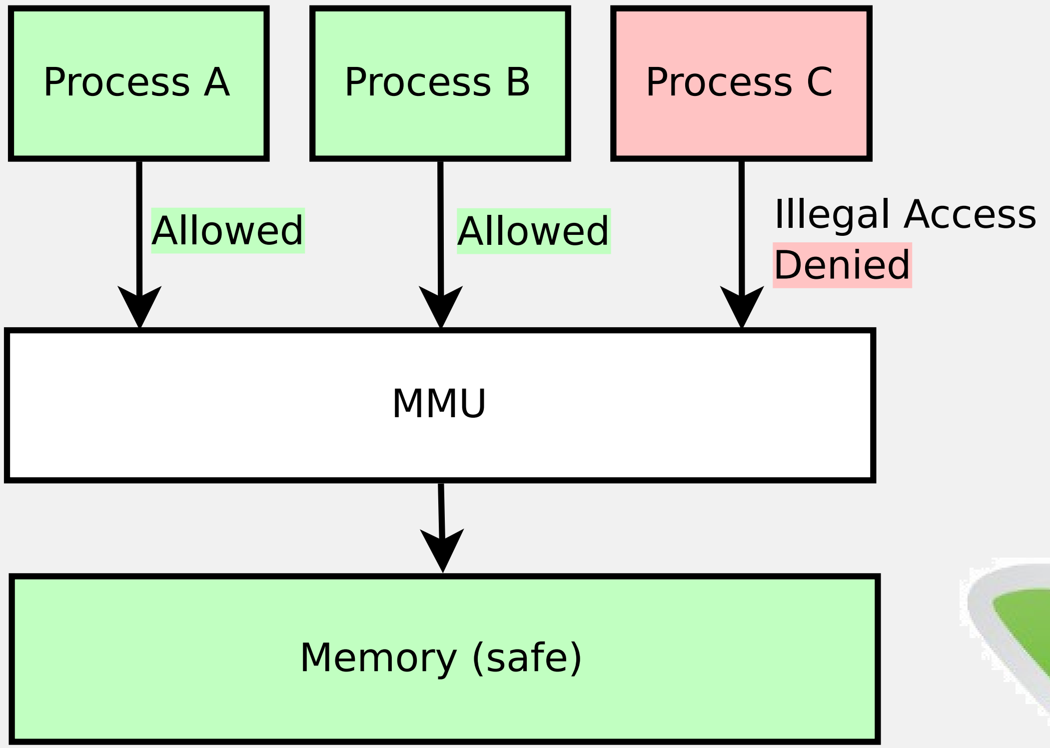
- QEMU had device assignment since 2012
- QEMU had vIOMMU emulation (VT-d) since 2014
- Emulated devices are supported by vIOMMU
  - *Using* QEMU's memory API when DMA
  - DMA happens *with* QEMU's awareness
  - Either full-emulated, or para-virtualized (vhost is special!)
- Assigned devices are not supported by vIOMMU
  - *Bypassing* QEMU's memory API when DMA
  - DMA happens *without* QEMU's awareness
  - Need to talk to host IOMMU for that
- Why bother?

The Problems (Why?)

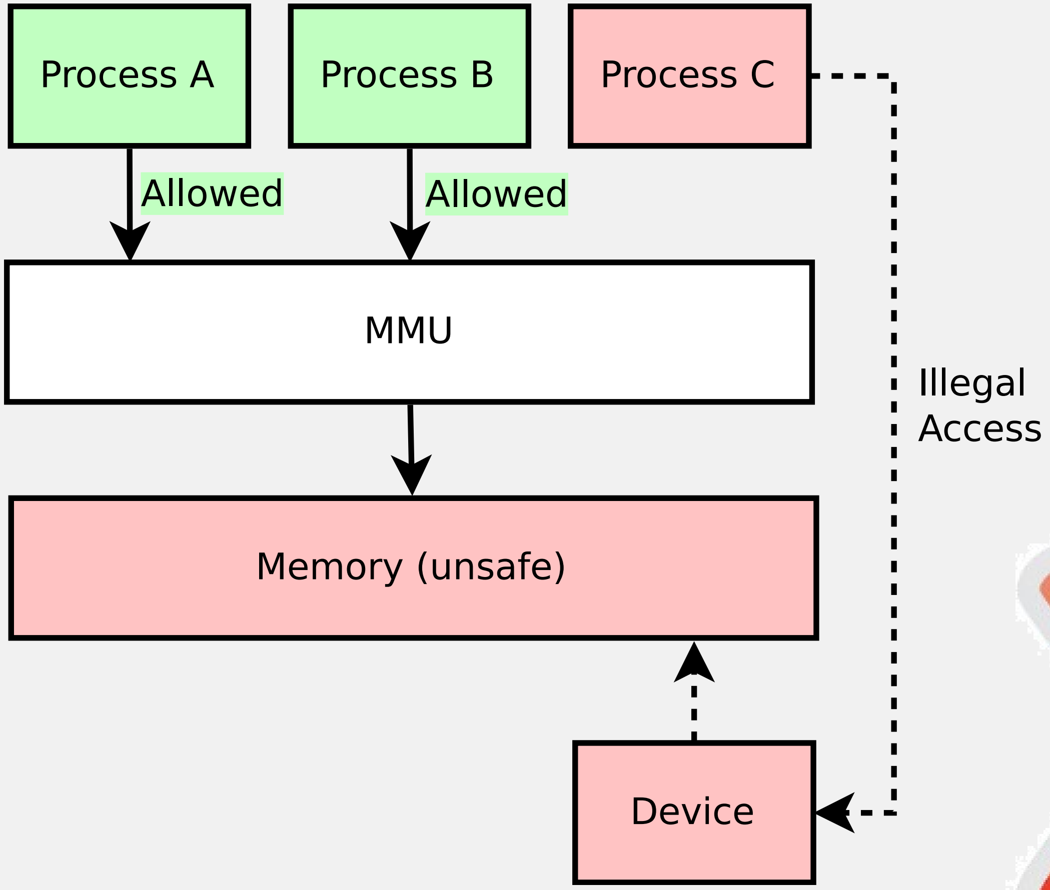
Problem 1: Userspace Drivers

- More userspace drivers!
  - DPDK/SPDK use PMDs to drive devices
- Userspace processes are not trusted
  - Processes can try to access any memory
  - Kernel protects against malicious memory access using MMU ~~(until we have Meltdown and Spectre…)~~
- Userspace device drivers are not trusted too!
  - Userspace drivers control devices, bypassing MMU
  - Need to protect the system on the device's side

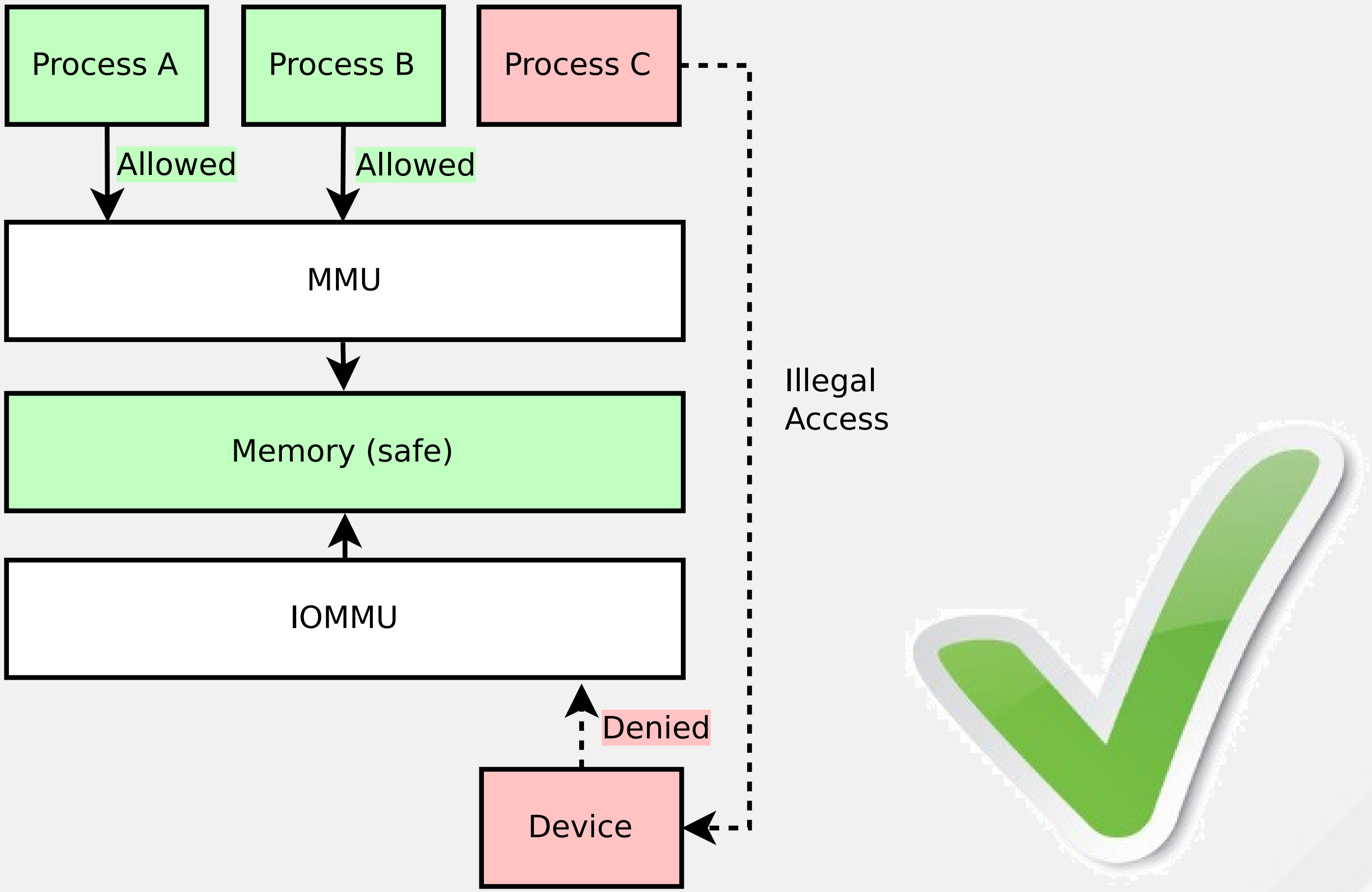
Userspace Drivers: MMU Protection



Userspace Drivers: Bypass MMU Protection



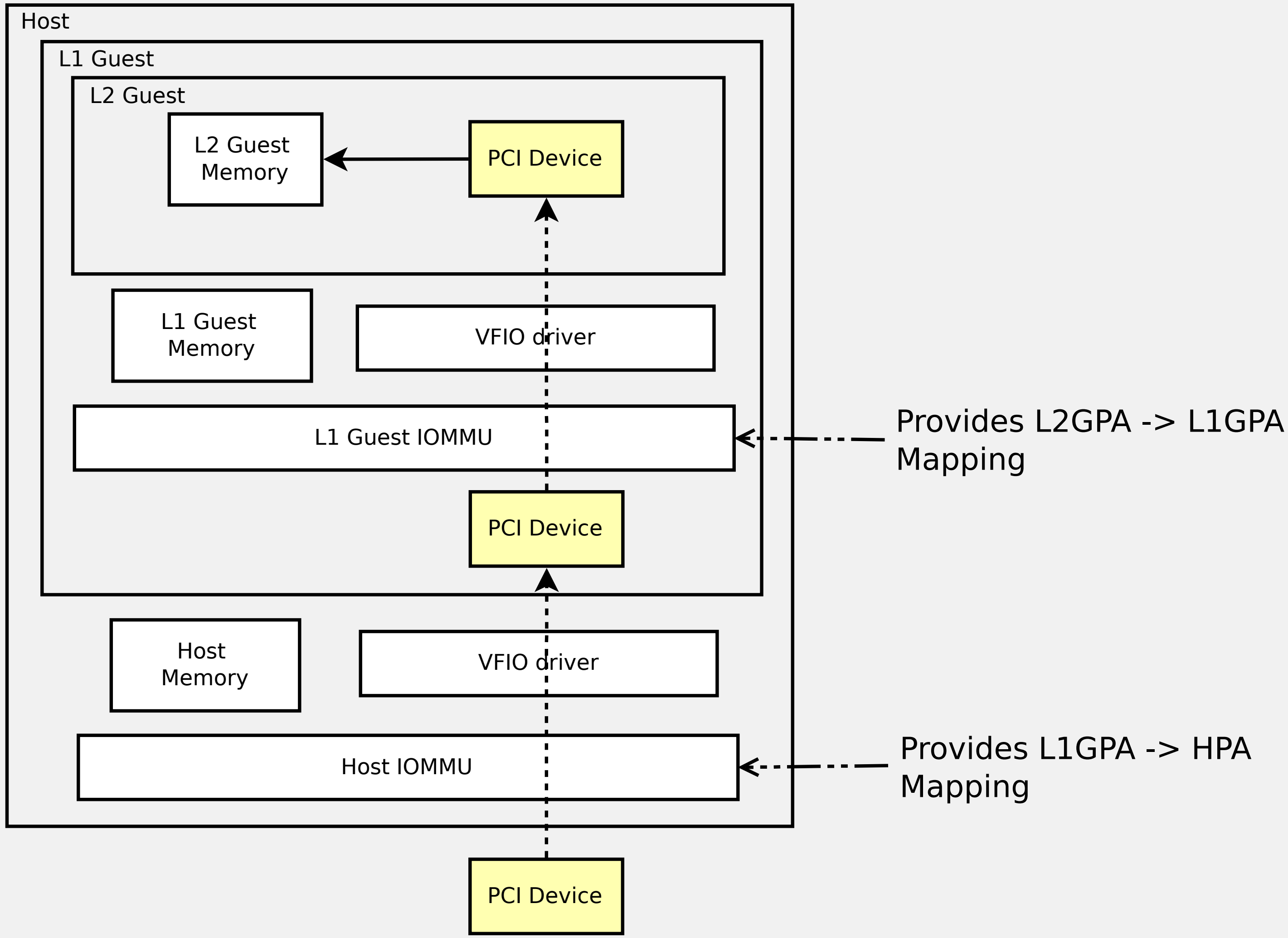
Userspace Drivers: MMU and IOMMU



Problem 2: Nested Device Assignments

- Terms:
  - **HPA**: Host Physical Address
  - $\mathbf{L_n}$**GPA**: nth-level Guest Physical Address
- How device assignment works
  - Maps $\mathbf{L_1}$**GPA → HPA** (L_1 guest is unaware of this)
- Can device assignment be nested?
  - What we want in the end: $\mathbf{L_2}$**GPA → HPA**
  - What we have already: $\mathbf{L_1}$**GPA → HPA**
  - Can’t do this without an IOMMU in L_1 guest ($\mathbf{L_2}$**GPA → **$\mathbf{L_1}$**GPA**)!

Problem 2: Nested Device Assignments



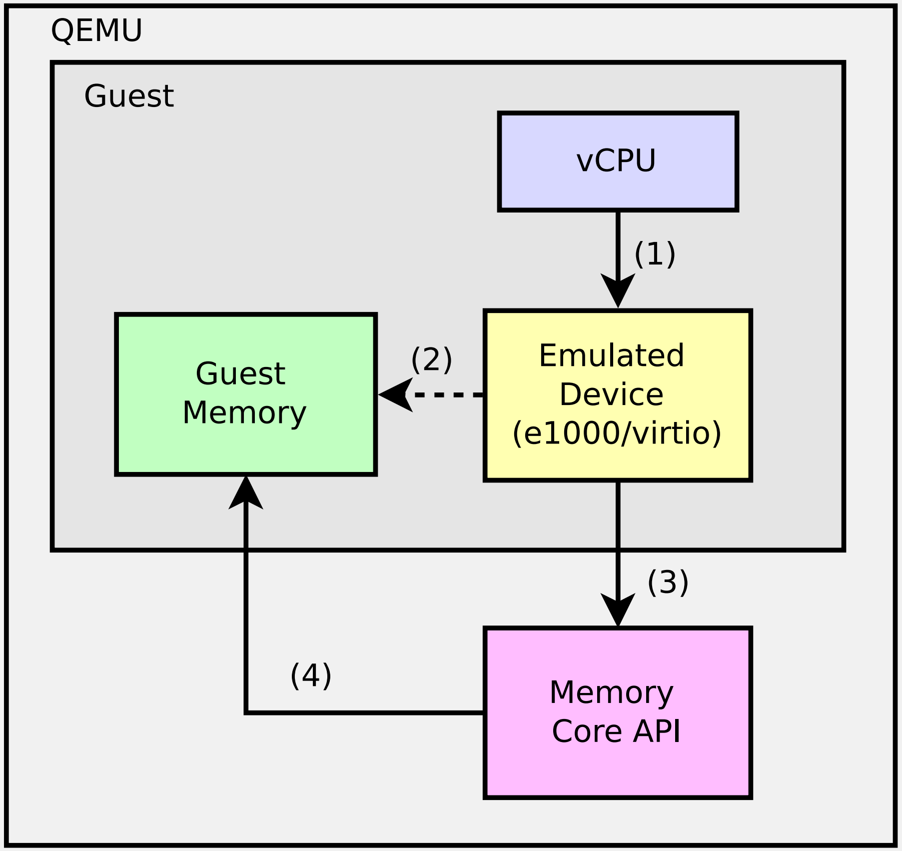
Summary of Problems

- Unsafe userspace device drivers:
  - needs IOMMU in L_1 guest to protect L_1 guest kernel from malicious/buggy userspace drivers
- Nested device assignments:
  - needs IOMMU in L_1 guest to provide the L_2GPA to L_1GPA mapping, finally to HPA
- We want device assignment to work under vIOMMU in the guests

redhat.

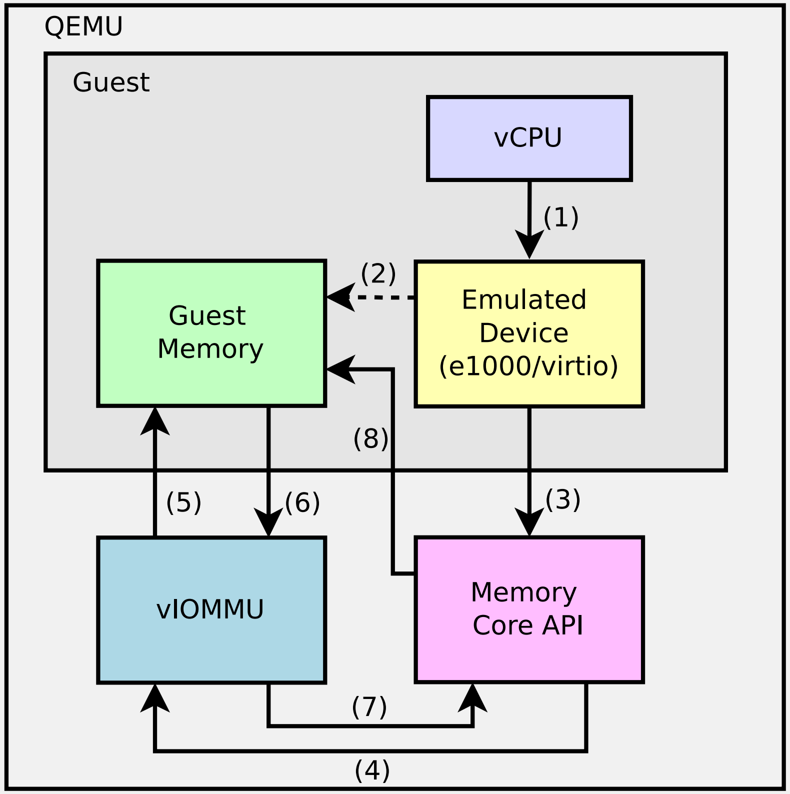
The Solution (How?)

Guest DMA for Emulated Devices, no vIOMMU



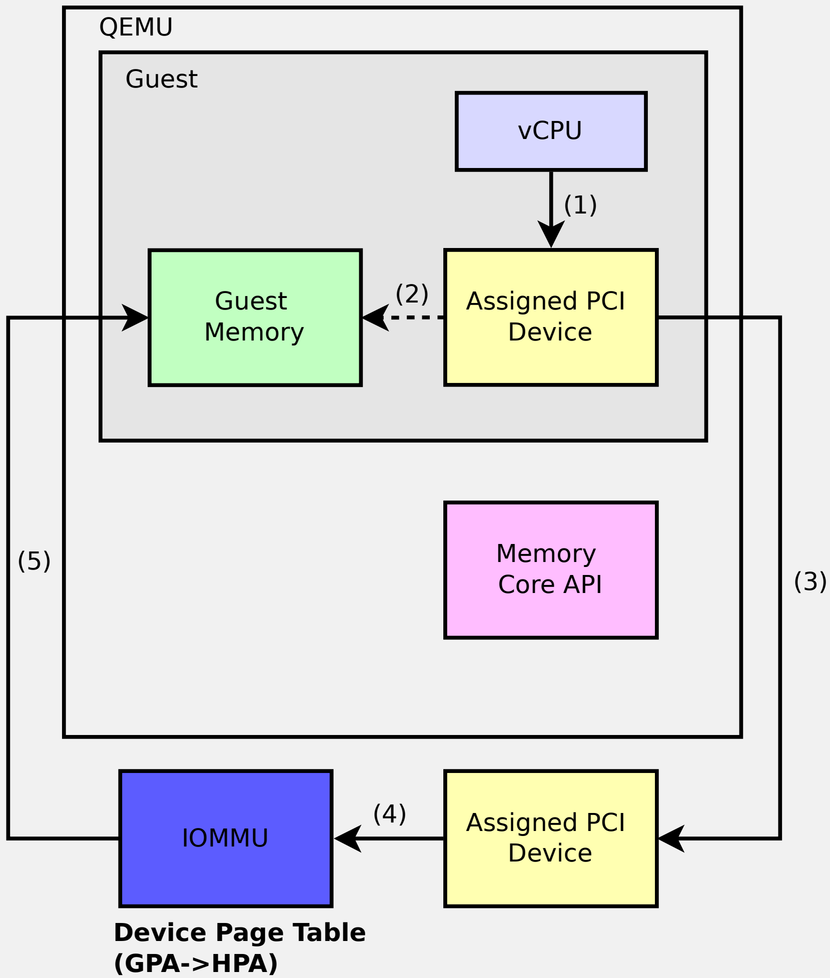
(1) IO Request
(2) Allocate DMA buffer
(3) DMA request (GPA)
(4) Memory access (GPA)

Guest DMA for Emulated Devices, with vIOMMU



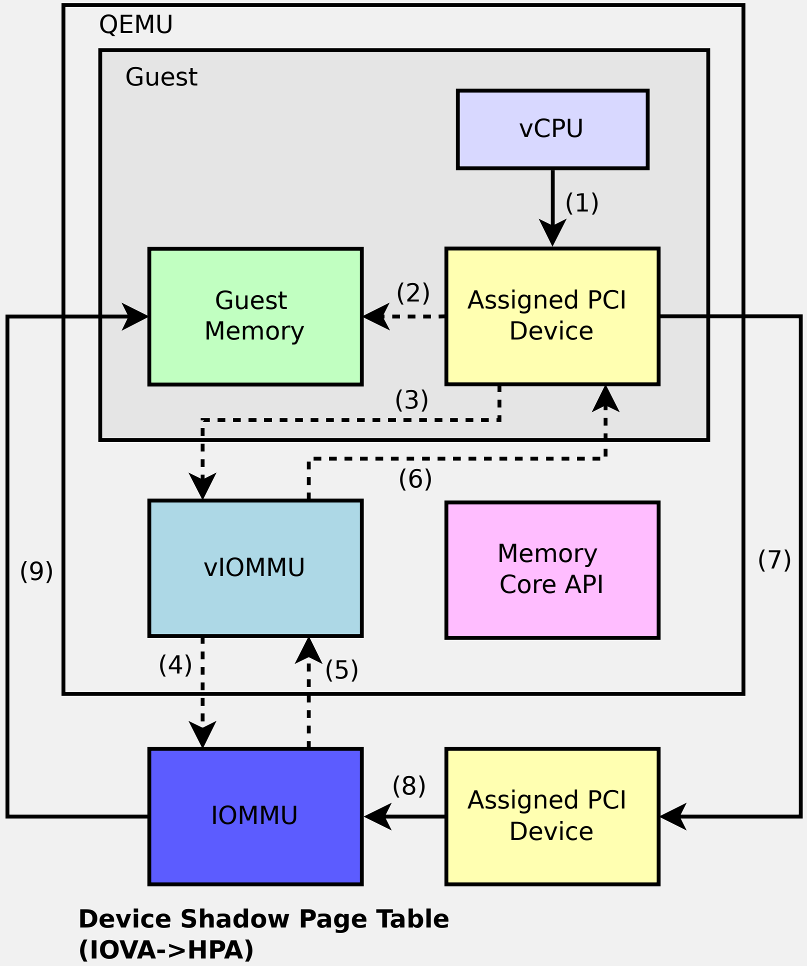
(1) IO request
(2) Allocate DMA buffer, setup device page table (IOVA->GPA)
(3) DMA request (IOVA)
(4) Page translation request (IOVA)
(5) Lookup device page table (IOVA->GPA)
(6) Get translation result (GPA)
(7) Complete translation request (GPA)
(8) Memory access (GPA)

Guest DMA for Assigned Devices, no vIOMMU



(1) IO request
(2) Allocate DMA buffer
(3) Virtual DMA request (using GPA)
(4) DMA request (using GPA)
(5) Memory access (using HPA)

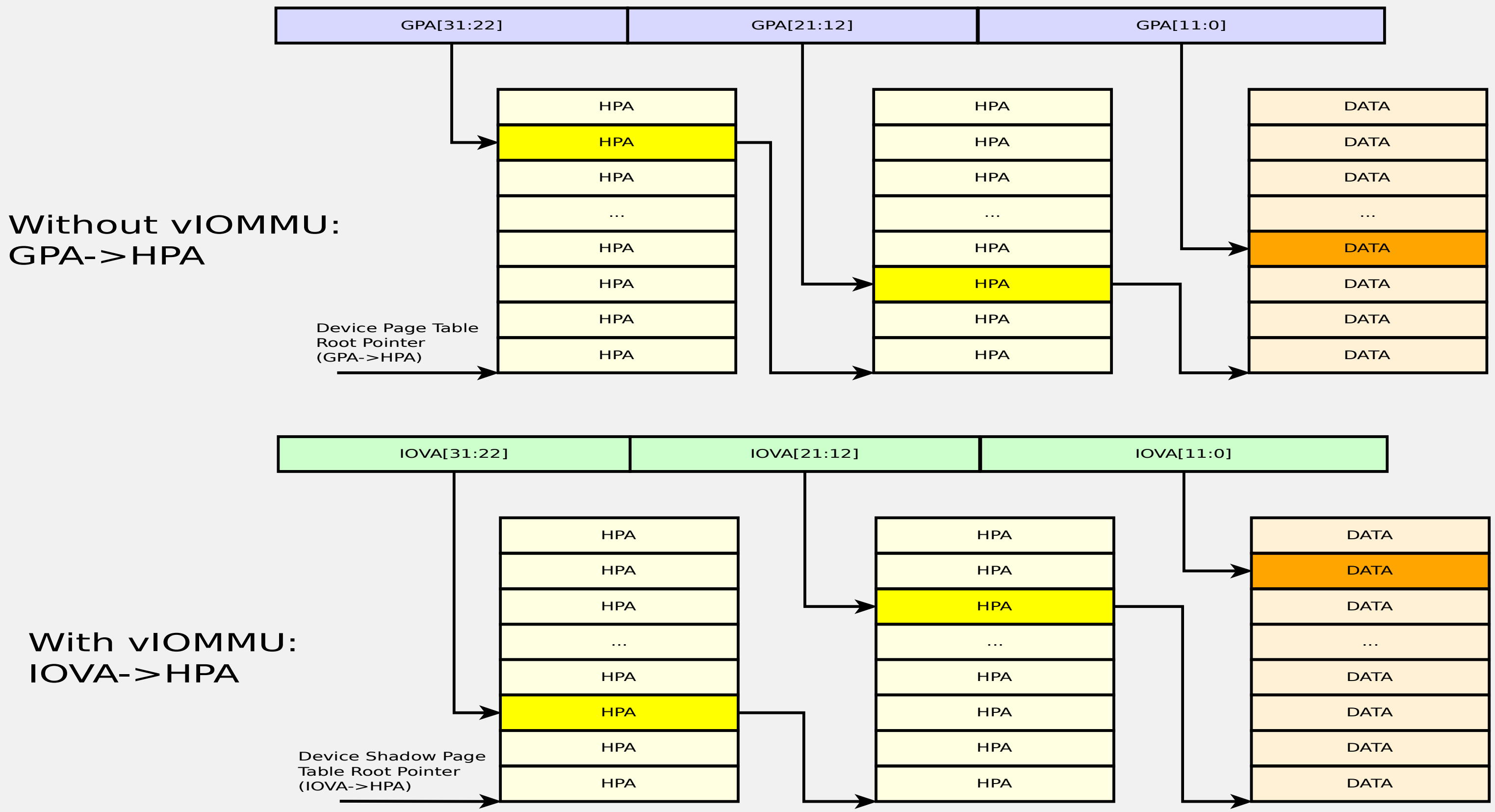
Guest DMA for Assigned Devices, with vIOMMU



(1) IO request
(2) Allocate DMA buffer, setup device page table (IOVA->GPA)
(3) Send MAP notification
(4) Sync shadow page table (IOVA->HPA)
(5) Sync Complete
(6) MAP notification Complete
(7) Virtual DMA request (using IOVA)
(8) DMA request (using IOVA)
(9) Memory access (using HPA)

IOMMU Shadow Page Table

Hardware IOMMU page tables without/with a vIOMMU in the guests
(GPA→HPA is the original page table; IOVA→HPA is the shadow page table)



Shadow Page Synchronization

- General solution:
  - Write-protect the whole device page table?
- Actual solution:
  - VT-d caching-mode: Any page entry update will require explicit invalidation of caches (VT-d spec chapter 6.1)
  - Intel only solution; PV-like, but also applies to hardware (Is there real hardware that declares caching-mode?)
  - *Maybe* it could be nicer if…?
    - Each invalidation can be marked as MAP or UNMAP
    - Invalidation range can be strict for MAPs

Shadow Page Table: MMU vs. IOMMU

Type	MMU	IOMMU
Target	Processor memory accesses	Device memory accesses (DMA)
Trigger mode (shadow sync)	#PF (Page Faults)	Caching mode (PV?)
Code path (shadow sync)	Short (KVM only)	Long (We'll see...)
Page table formats	32-bits, 64-bits, PAE...	64-bits only
Need previous state?	No	Yes (cares more about page changes[1])
Page faults?	Yes	No (not yet?)

[1]: Converts new/deleted pages into MAP/UNMAP notifies downwards. A funny fact is that we can't really "modify" an IOMMU page table entry since we don't normally have a modify API along the way (Please refer to VFIO_IOMMU_[UN]MAP_DMA in VFIO API, or iommu_[un]map() in kernel API)

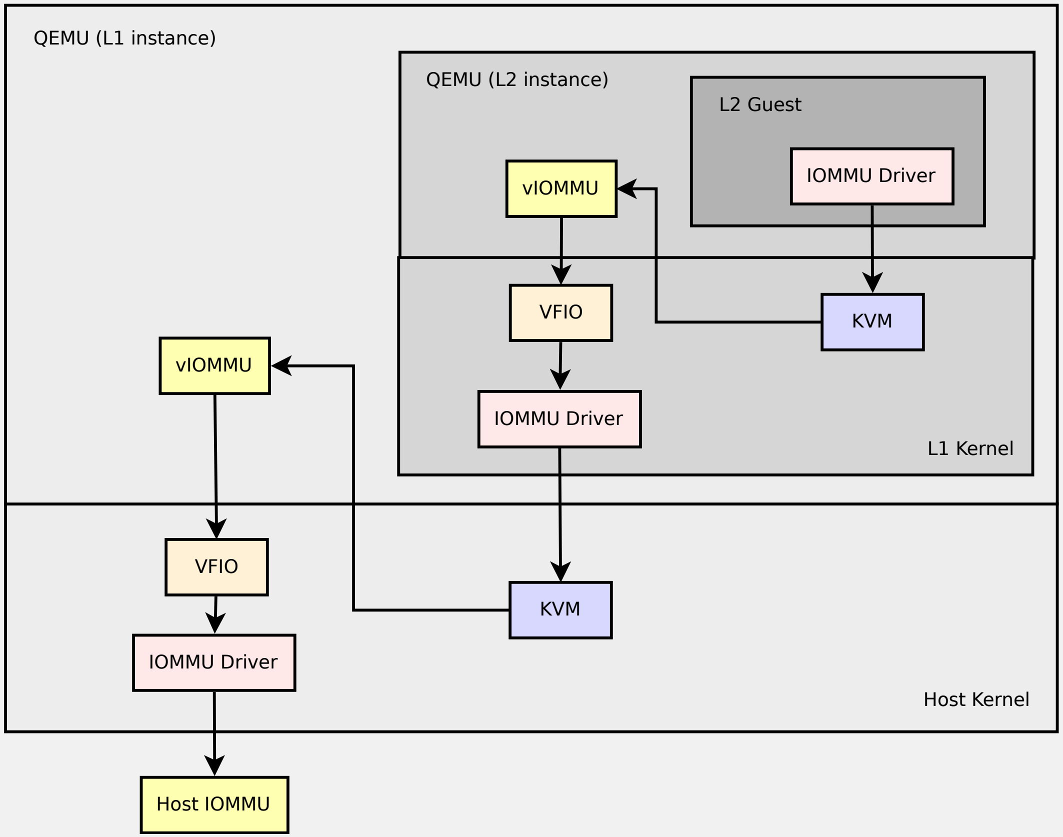
Some Facts...

- Emulated devices vs. Assigned devices
  - Emulated: quick mappings, slow IOs
  - Assigned: slow mappings, quick IOs
- Performance (assigned devices + vIOMMU, 10gbps NIC)
  - Kernel drivers are slow (>80% degradation)
  - DPDK drivers are as fast as when without vIOMMU
    - Both L_1/L_2 guests performances close to line speed
  - What matters: whether the mapping is static
- Long code path on shadow page synchronization
  - Reduce context switches? “Yet-Another vhost(-iommu)”?
  - “How long?” Please see the next slide...

redhat.

“How Long?”

(Example: when L_2 guest maps one page)



Status Update

Status and Update

- QEMU
  - QEMU 2.12 provided initial support for device assignment with vIOMMU, QEMU 2.13 (3.0) contains some important bug fixes
  - Please use QEMU 2.13 (3.0) or newer
- Linux
  - Linux v4.18 contains a very critical bug fix: 87684fd997a6
  - Please use v4.18-rc1 or newer
- For more information about VT-d emulation on QEMU, please refer to:
  - https://wiki.qemu.org/Features/VT-d

redhat.

Known Issues

- Extremely bad performance for dynamical mapping DMA
  - >80% performance drop for kernel drivers
  - DPDK applications are not affected
- Limitation on assigning multiple functions that share a single IOMMU group in the host (when vIOMMU exists)
  - Currently only allow to assign a single function if multiple functions are sharing the same IOMMU group on the host

THANK YOU

plus.google.com/+RedHat

linkedin.com/company/red-hat

youtube.com/user/RedHatVideos

facebook.com/redhatinc

twitter.com/RedHat