Shared Virtual Addressing in KVM

Yi Liu (yi.l.liu@intel.com)
Jacob Pan (jacob.jun.pan@intel.com)
KVM Forum 2018

Legal Disclaimer

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

This document contains information on products, services and/or processes in development. All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest forecast, schedule, specifications and roadmaps.

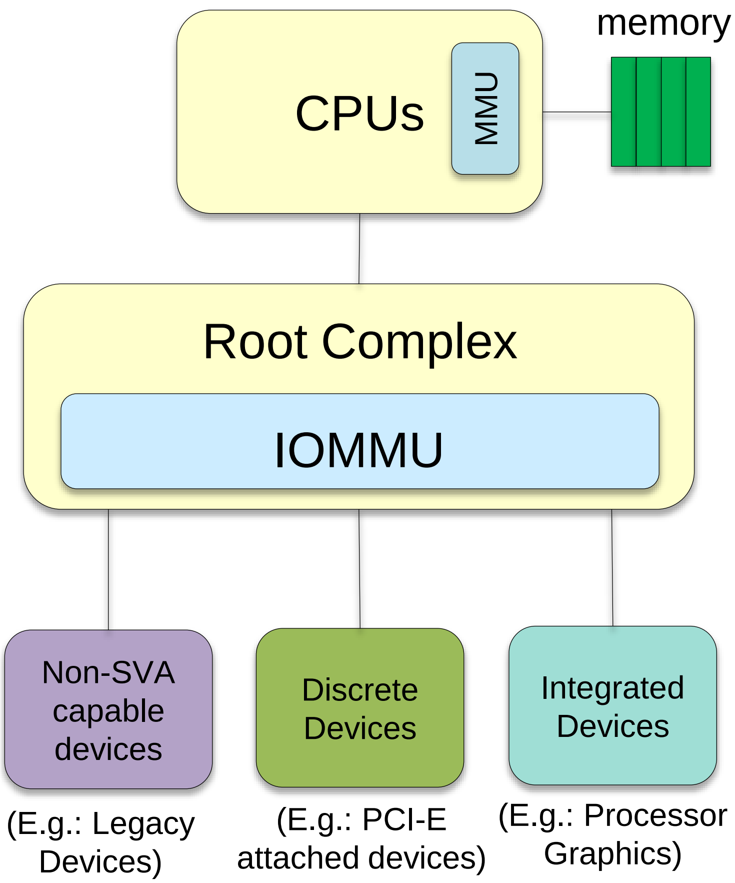
The products and services described may contain defects or errors known as errata which may cause deviations from published specifications. Current characterized errata are available on request.

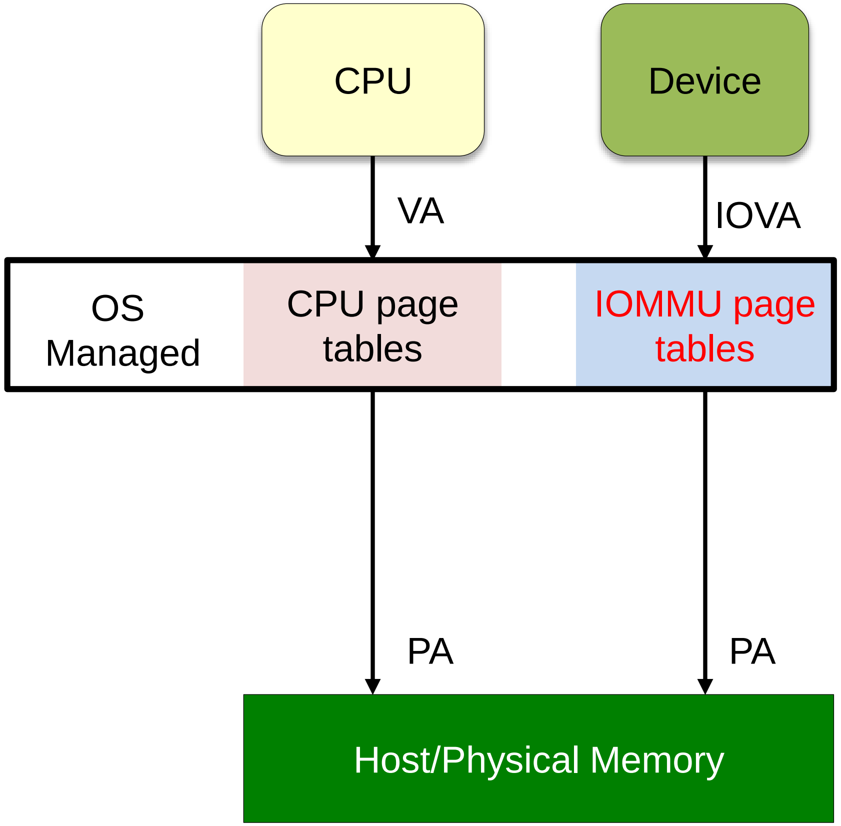
Copies of documents which have an order number and are referenced in this document may be obtained by calling 1-800-548-4725 or by visiting www.intel.com/design/literature.htm.

Intel and the Intel logo are trademarks of Intel Corporation or its subsidiaries in the U.S. and/or other countries.

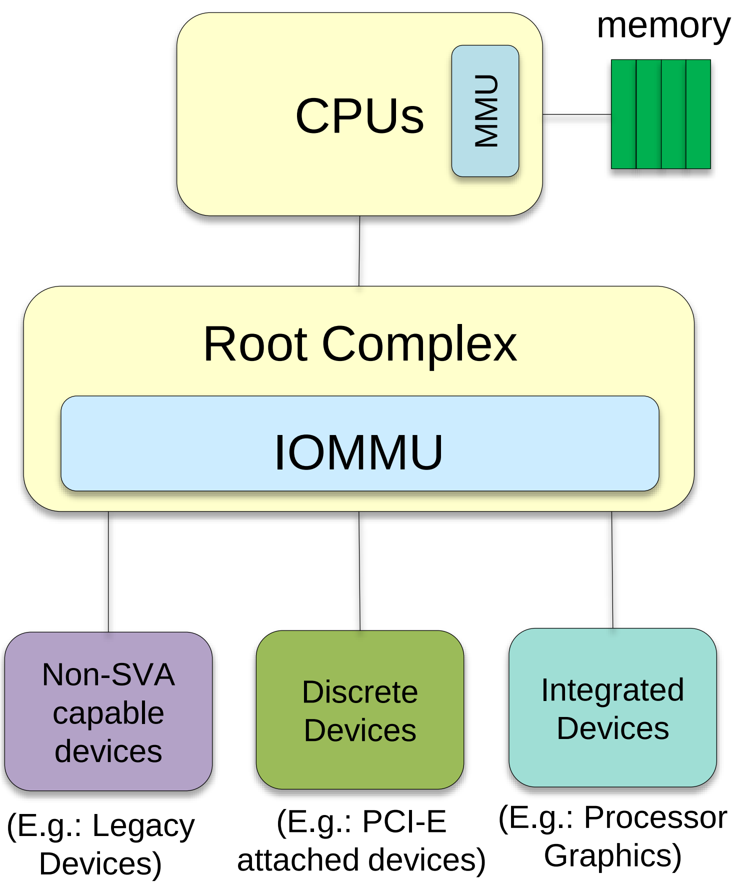
*Other names and brands may be claimed as the property of others

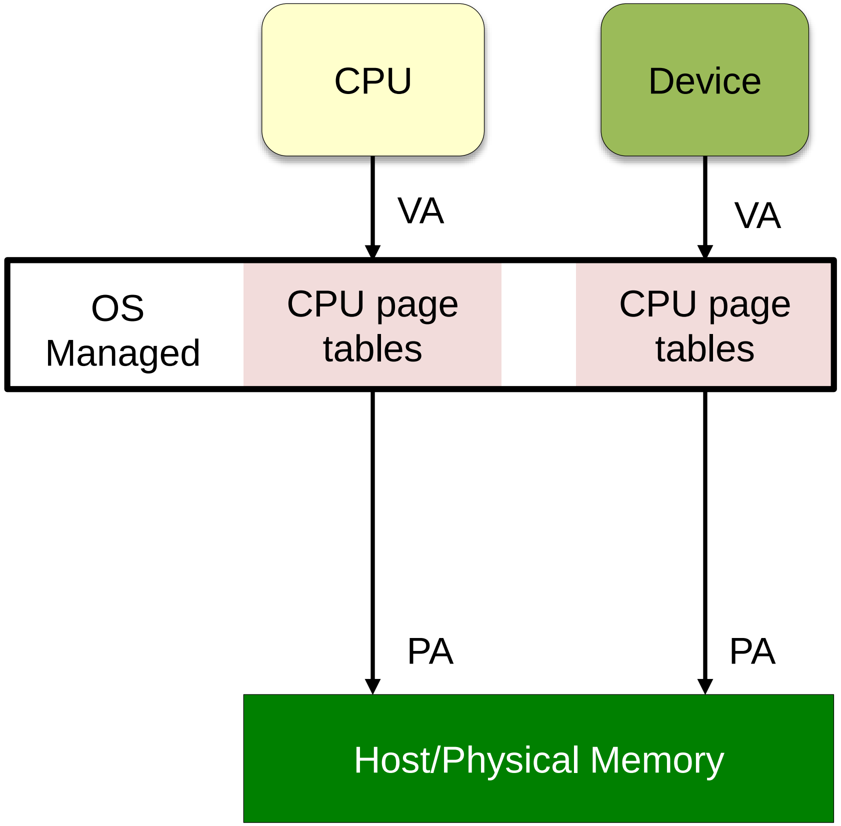
Shared Virtual Addressing(SVA)





Shared Virtual Addressing(SVA)





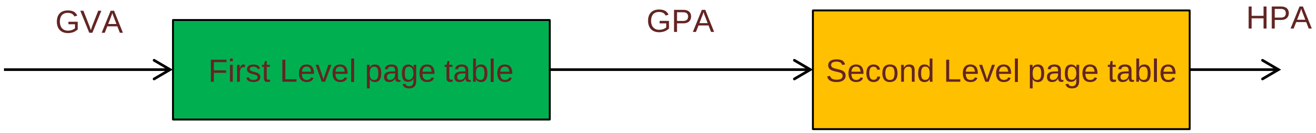
Previously called “Shared Virtual Memory”

SVA on Intel® VT-d

- Process Address Space ID (PASID)
  - Identify process address space
- First-level/Second-level translation
  - Supports different usages (IOVA/SVA) by different translation types
- Translation Types
  - First-Level translation
  - Second-Level translation
  - Nested translation
  - Pass-Through (address translation bypassed)
- Intel® VT-d 3.0 introduced Scalable Mode
  - SVA and Intel® Scalable I/O Virtualization technology are orthotropic

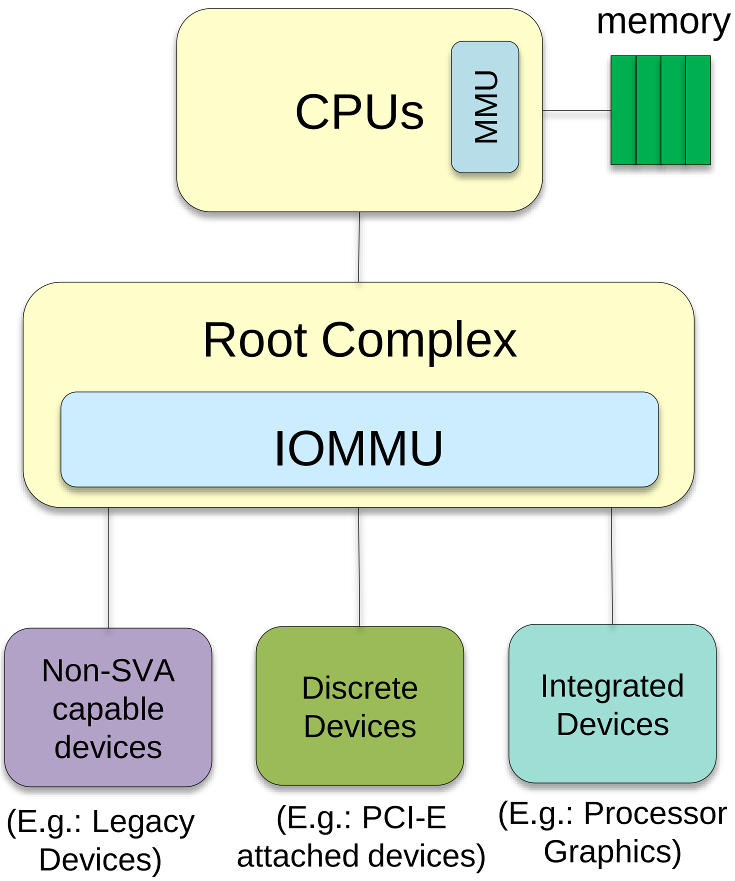
SVA on Intel® VT-d (Cont.)

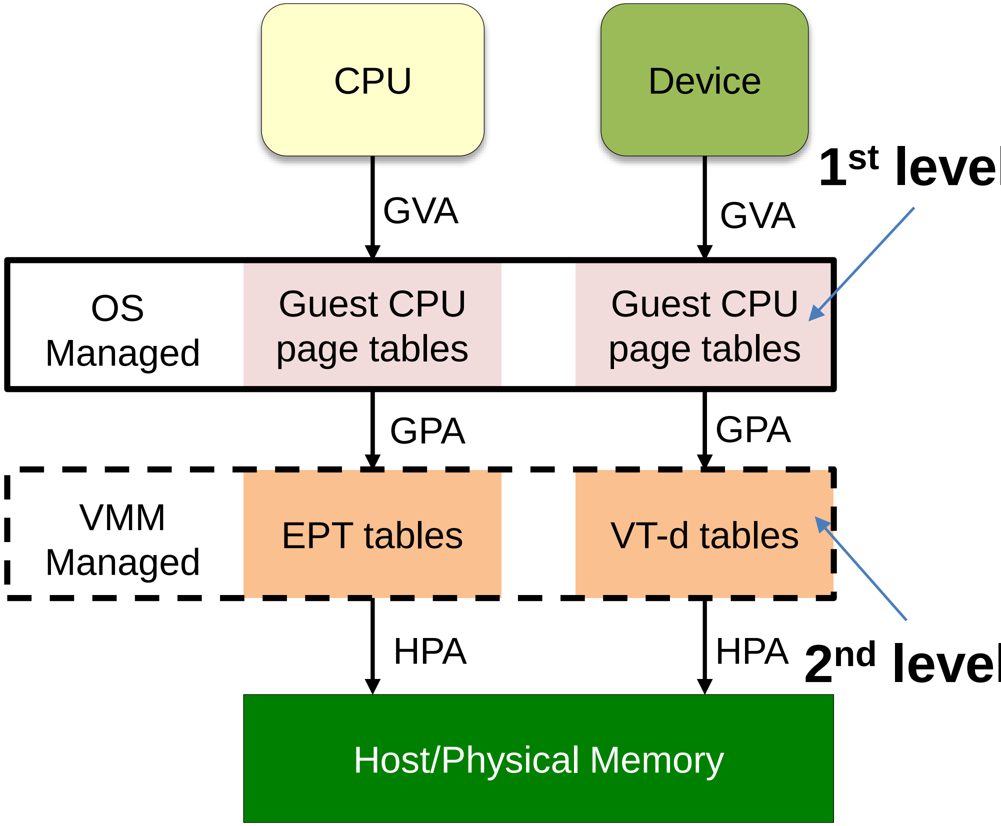
- Nested Translation
  - Use both first-level and second-level for address translation
  - Enable SVA in virtualization environment
    - First-level: GVA->GPA
    - Second-level: GPA->HPA



- Most vendor supports nested translation for SVA usage in Virtual Machine

vSVA on Intel® VT-d





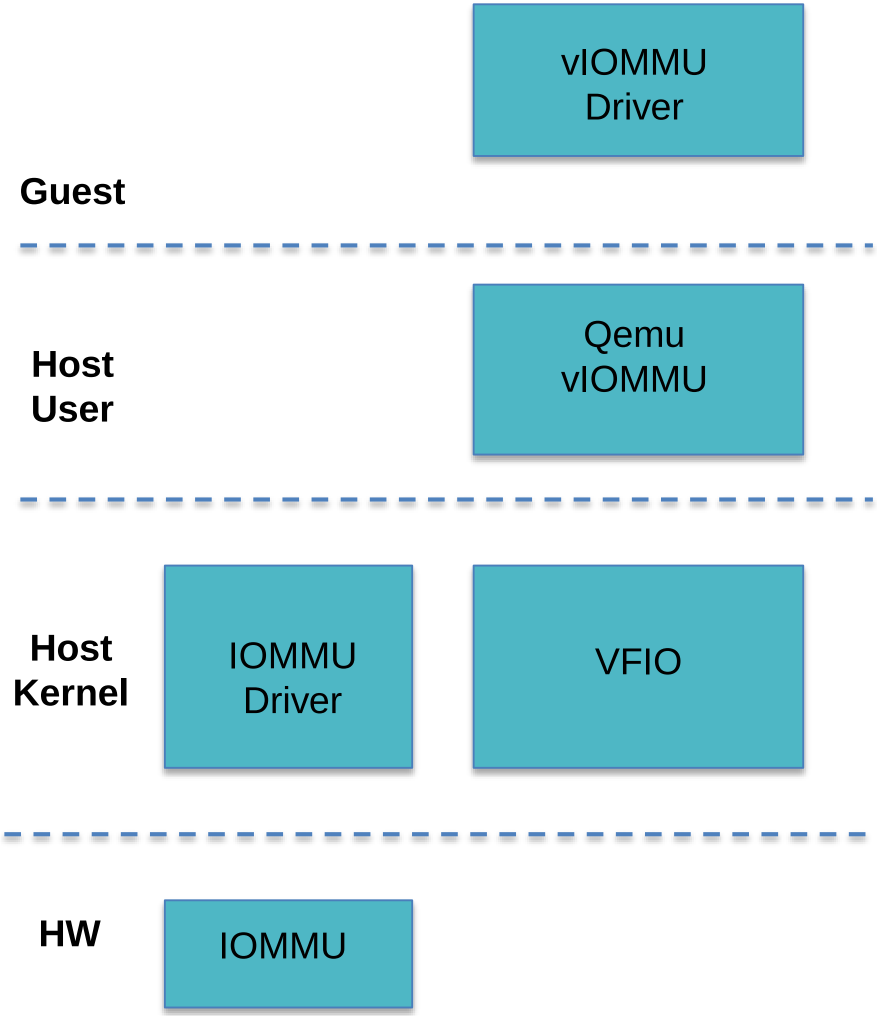
Enable SVA in VM

- Need a virtual IOMMU with SVA capability
  - Proper emulation according to IOMMU spec (e.g. Intel® VT-d specification)
    - either fully-emulated or virtio-based IOMMU

- Notification for guest translation structure modifications
  - Notification mechanism is vendor specific
  - For Intel® VT-d
    - “caching-mode”: explicit cache invalidation is required for any translation structure change in software

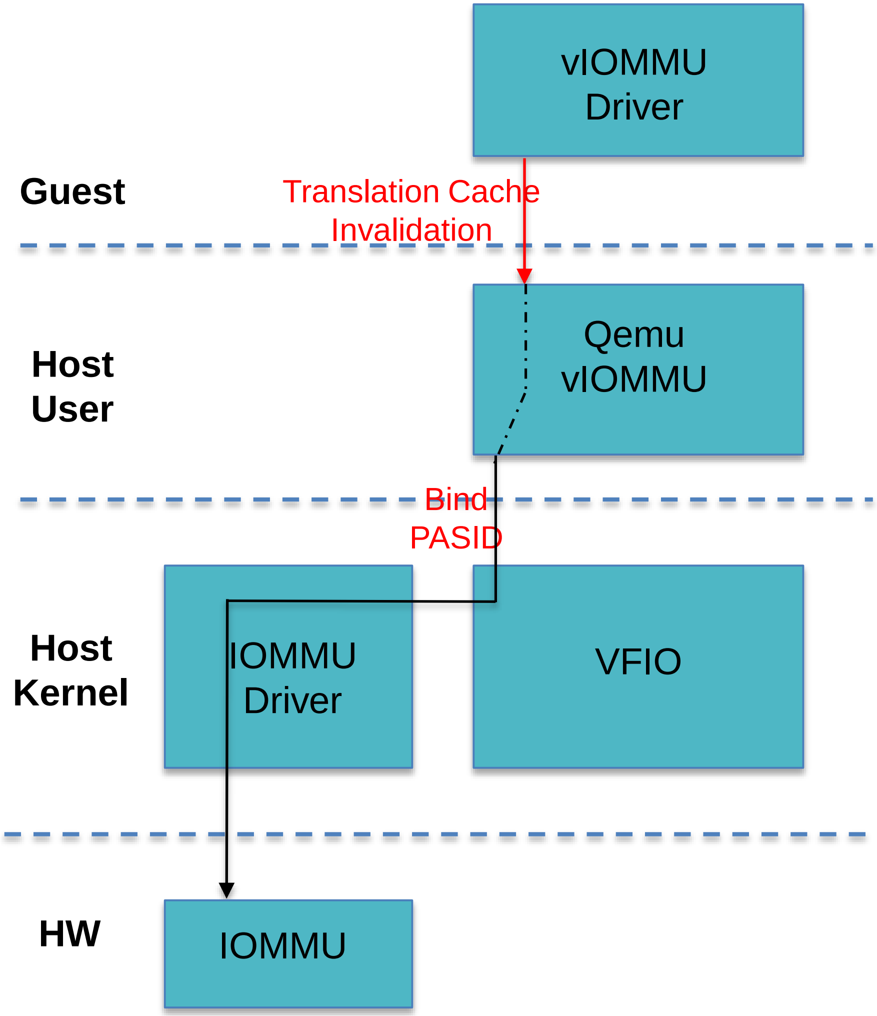
- Enable nested translation on physical IOMMU for given PASID

SVA Architecture in KVM



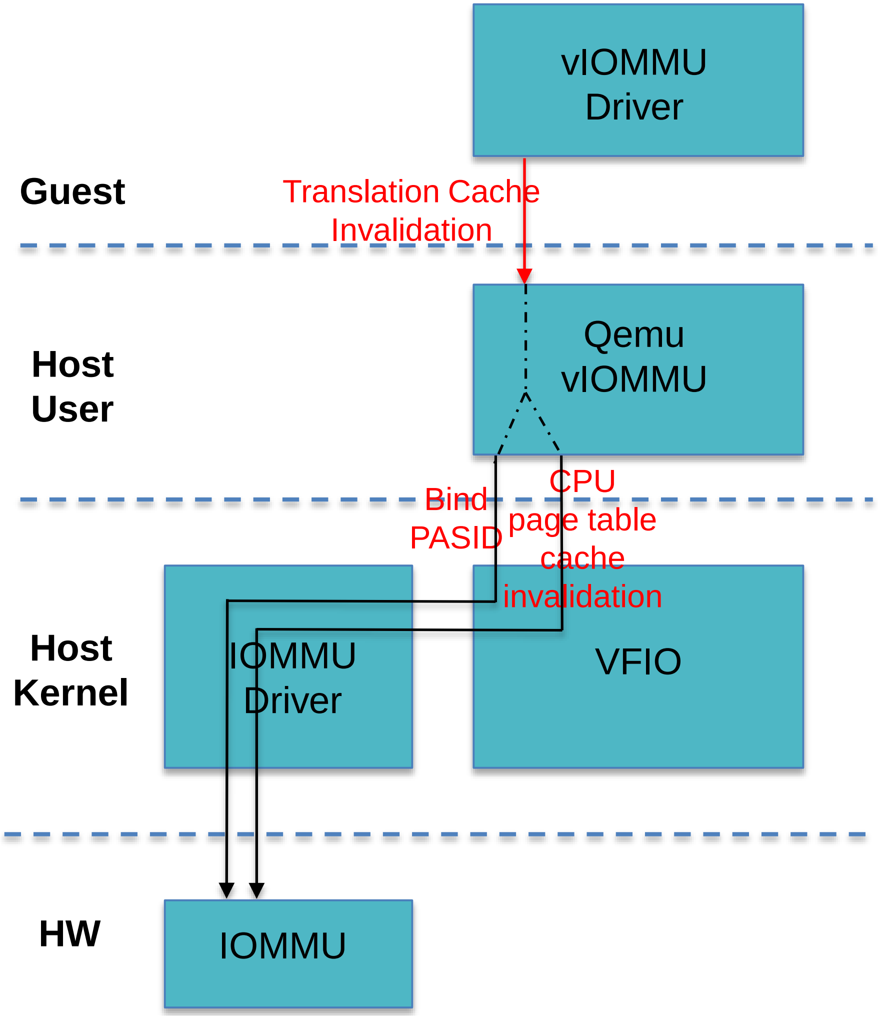
- Qemu
  - ➢ vIOMMU emulation is in Qemu
- VFIO: Virtual Function I/O
  - ➢ Program host IOMMU via VFIO
- IOMMU driver
  - ➢ New APIs exposed to VFIO for guest SVA

SVA Architecture in KVM (Cont.)



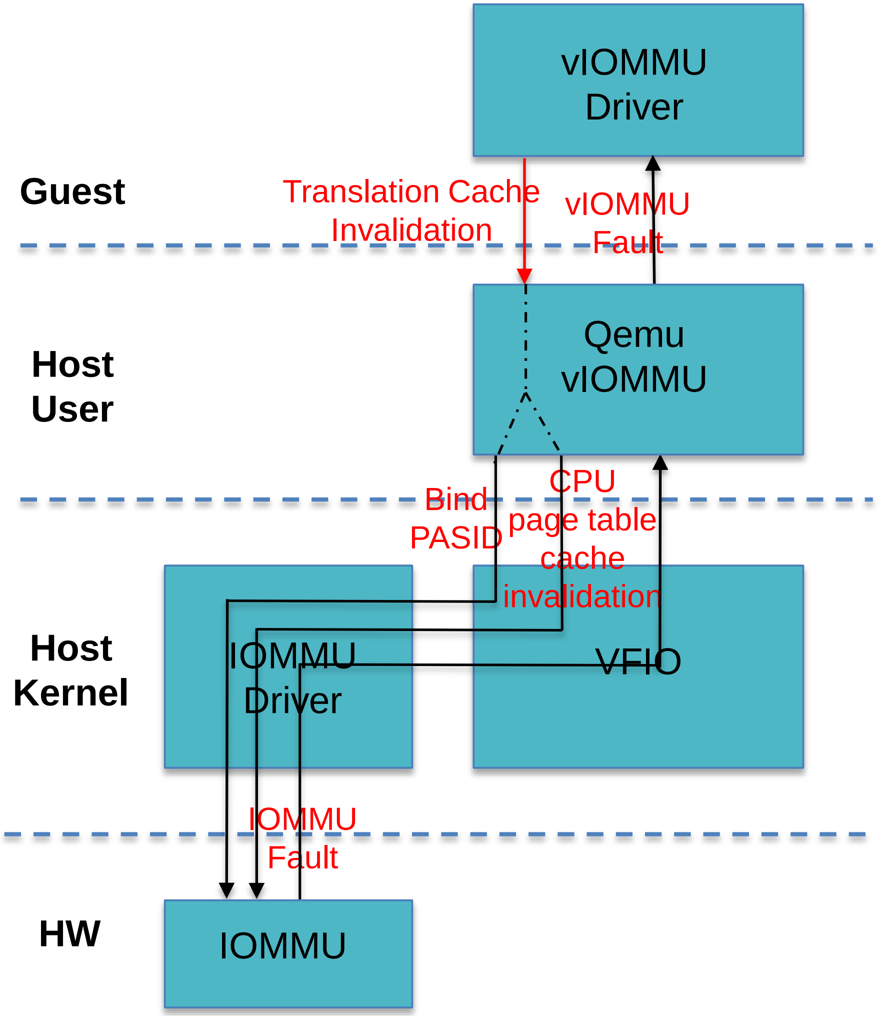
- Bind PASID
  - VT-d: guest CPU page table

SVA Architecture in KVM (Cont.)



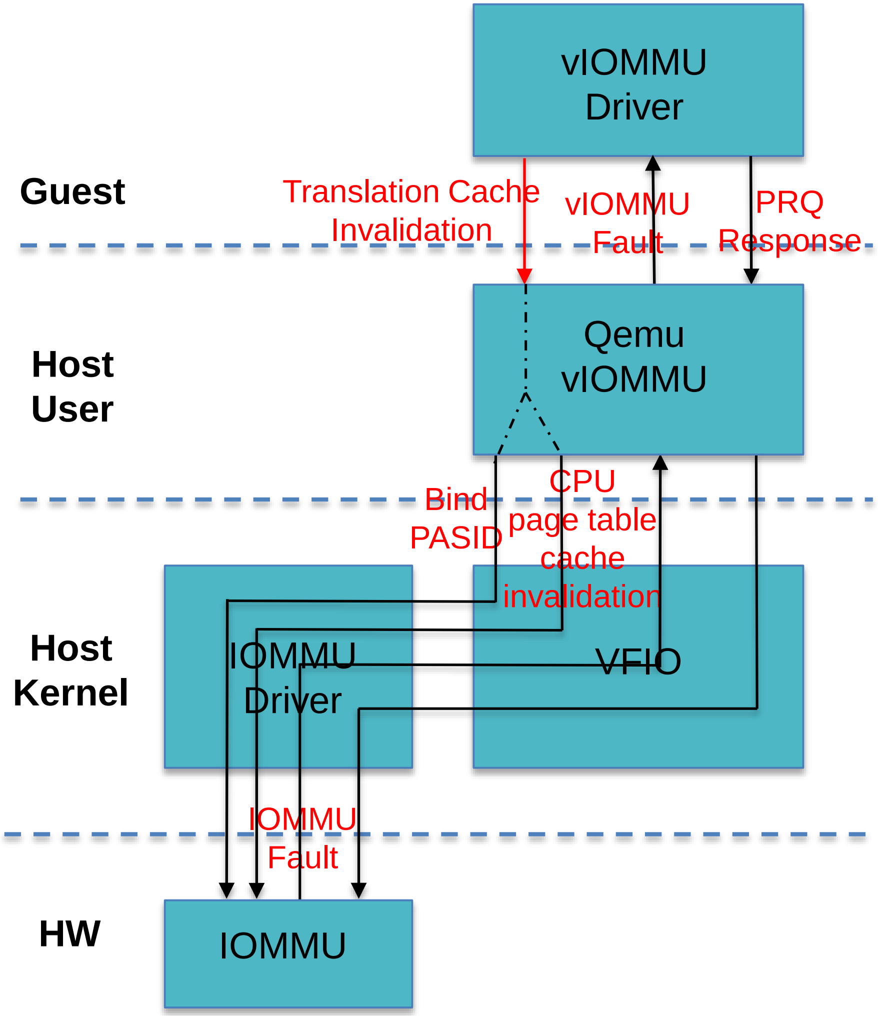
- Bind PASID
  - VT-d: guest CPU page table
- Forward guest CPU page table cache invalidation to host

SVA Architecture in KVM (Cont.)



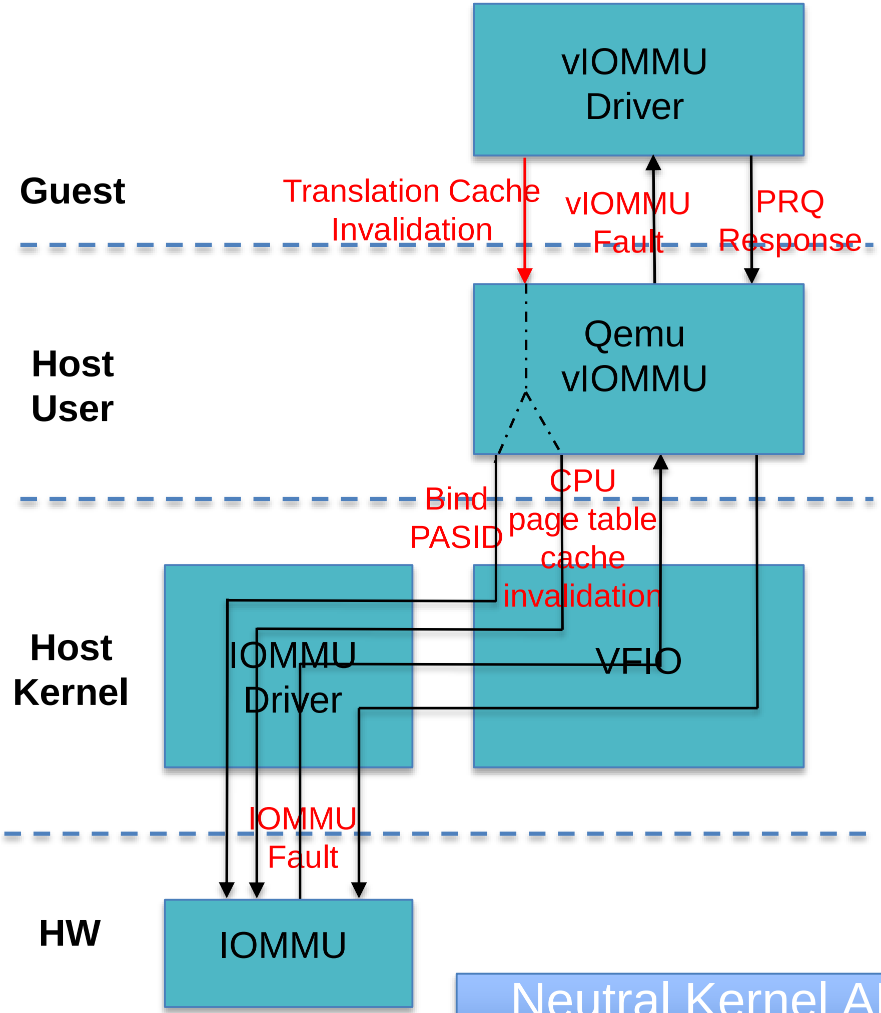
- Bind PASID
  - VT-d: guest CPU page table
- Forward guest CPU page table cache invalidation to host
- Page fault reporting and servicing

SVA Architecture in KVM (Cont.)



- Bind PASID
  - VT-d: guest CPU page table
- Forward guest CPU page table cache invalidation to host
- Page fault reporting and servicing

SVA Architecture in KVM (Cont.)



- Bind PASID
  - VT-d: guest CPU page table
- Forward guest CPU page table cache invalidation to host
- Page fault reporting and servicing

Neutral Kernel APIs for both emulated and virtio-based vIOMMUs

Changes to Qemu/VFIO/IOMMU

- Qemu
  - Vendor specific vIOMMU emulation
  - Capture guest IOMMU translation modifications and program host IOMMU via VFIO IOCTL

- VFIO: Virtual Function I/O
  - New IOCTL will be introduced:
    - VFIO_IOMMU_BIND_PROCESS
      Binding to host CPU page table
    - VFIO_IOMMU_BIND_GUEST_PGTBL
      Binding to guest CPU page table
    - VFIO_IOMMU_BIND_GUEST_PASID_TBL
      Binding to guest PASID Table
    - VFIO_IOMMU_SVA_INVALIDATE
      Invalidate tlb for guest
    - VFIO_DEVICE_DMA_FAULT_FD_SET
      Set fault eventfd for notifying userspace (Qemu)
    - VFIO_DEVICE_GET_DMA_FAULT_INFO
      Get dma fault info to userspace (Qemu)

- IOMMU driver
  - Jacob will introduce detail on it

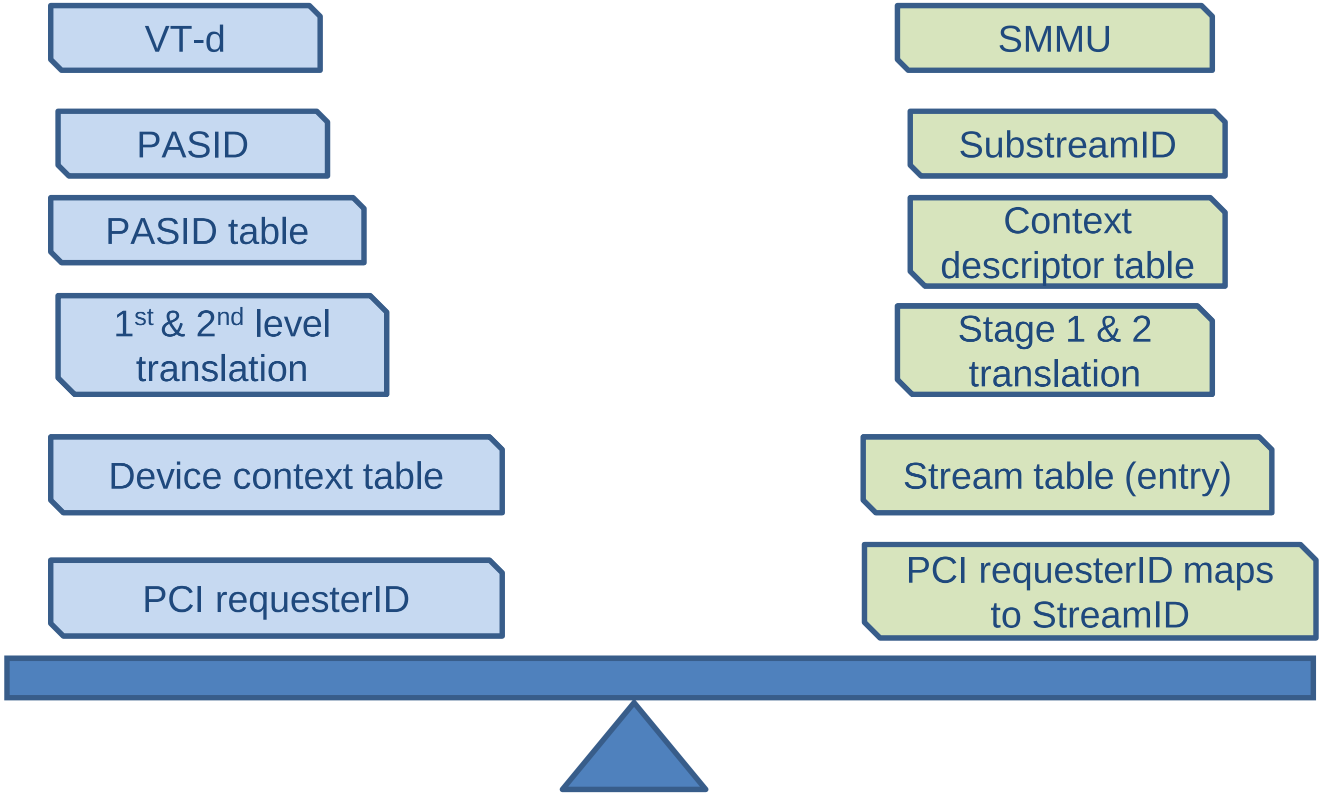
Upstream Status (Qemu)

- Qemu vSVA enabling has two parts
  - vIOMMU emulation
    - Earliest RFC patch for vSVA back to 2017-April
  - Notification framework between vIOMMU device-model and VFIO within Qemu
    - Yi Liu: Notifier framework in v3, proposed PCISVAOps for communication between vIOMMU emulator and VFIO
    - Eric Auger: vSMMUv3/pSMMUv3 2 stage VFIO integration v2
    - Shares the notification framework work
- TODO:
  - consolidate the common part between different tracks
  - Hardware IOMMU capability query interface
    - vIOMMU should not report capabilities with no host support if VM has assigned devices

Upstream Status (Kernel)

- IOMMU/VFIO extension for virtual SVA
  - Earliest RFC patch for vSVA support
  - IOMMU APIs & VT-d in v5 by Jacob Pan & Yi Liu (https://lkml.org/lkml/2018/5/11/605)
  - Reuse and extend the above IOMMU API with ARM SMMU support by Eric Auger (https://lkml.org/lkml/2018/9/18/1087)
- Native SVA support
  - Generic IOMMU/VFIO API and ARM support (Jean-Philippe Brucker, ARM) (https://patchwork.kernel.org/patch/10608303/, https://patchwork.kernel.org/patch/10394831/
- Shared requirements in the two tracks
  - binding PASID, fault reporting
- Dependent changes
  - VT-d v3 support by Lu, Baolu (https://lkml.org/lkml/2018/10/7/54)

Terminology puzzle



VT-d

PASID

PASID table

1st & 2nd level translation

Device context table

PCI requesterID

SMMU

SubstreamID

Context descriptor table

Stage 1 & 2 translation

Stream table (entry)

PCI requesterID maps to StreamID

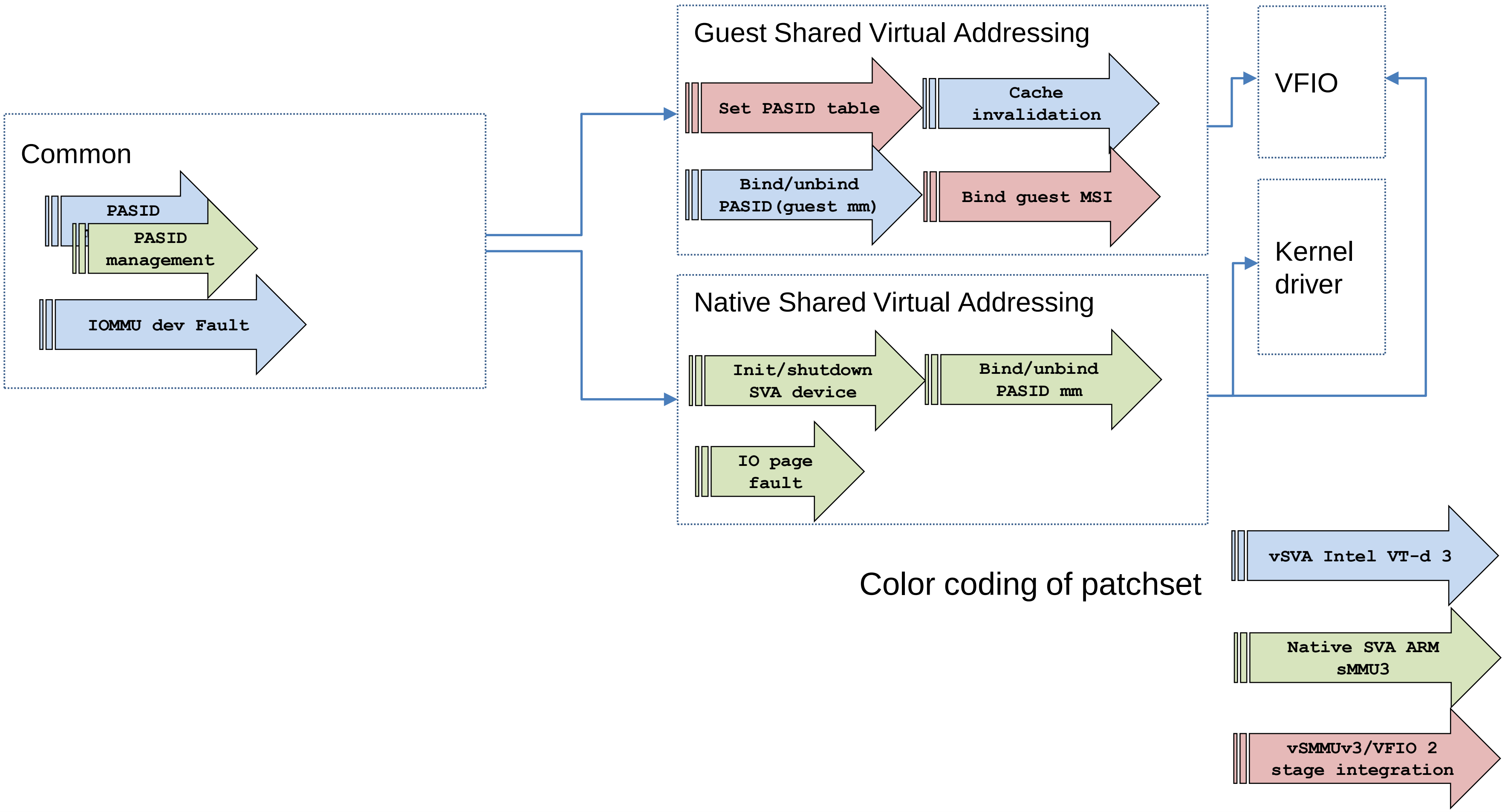
A tale of two SVAs

Key differentiation Features	Intel VT-d v3	ARM SMMUv3
Guest PASID allocation	Allocated by host system wide via virtual command interface	Allocated by guest in its own space
Device PASID table	Managed by host, shadowed	Managed solely by guest
GPA-HPA translation	2nd level per PASID	Stage 2, shared by all PASIDs per streamID
PASID 0	Available for allocation if RID2PASID is not enabled	Reserved for request w/o PASID
Page request/response	Has private data, needs page response w/o last page in group (LPIG)	Support non-PCI and PCI PRI-like stall model, no dependency on ATS**
IOMMU domains	Does not support default domain with DMA API*	Supports default DMA domain, can switch in/out default domain

* In progress to align with other IOMMU drivers

** All stall faults need response, faults contain more info such as which stage

IOMMU SVA API development



IOMMU API extensions proposed

API	Usage
iommu_set_pasid_table	Guest owns PASID table. PASID managed by guest.
Iommu_bind/unbind_pasid_for_guest*	Bind guest process to host allocated PASID. Host owns system wide PASID table
iommu_cache_invalidate	Translation cache invalidation passed down from guest
Iommu_report_device_fault	Report IOMMU detected device faults outside IOMMU subsystem, e.g. page request to be handled by guest.
Iommu_sva_suspend/resume_pasid()*	Device switch context while maintain PASID bond
Iommu_page_response()	Send page response after page request is handled
Iommu_sva_init/shutdown_device()	Prepare device for SVA, e.g. enable PRI, mm_exit notifier
Iommu_sva_bind/unbind_device()	Create bond between mm, PASID, and device
PASID management APIs	Management and helper function for lookup
Iommu_bind_guest_msi	Reuse gIOVA doorbell in host

* not yet published

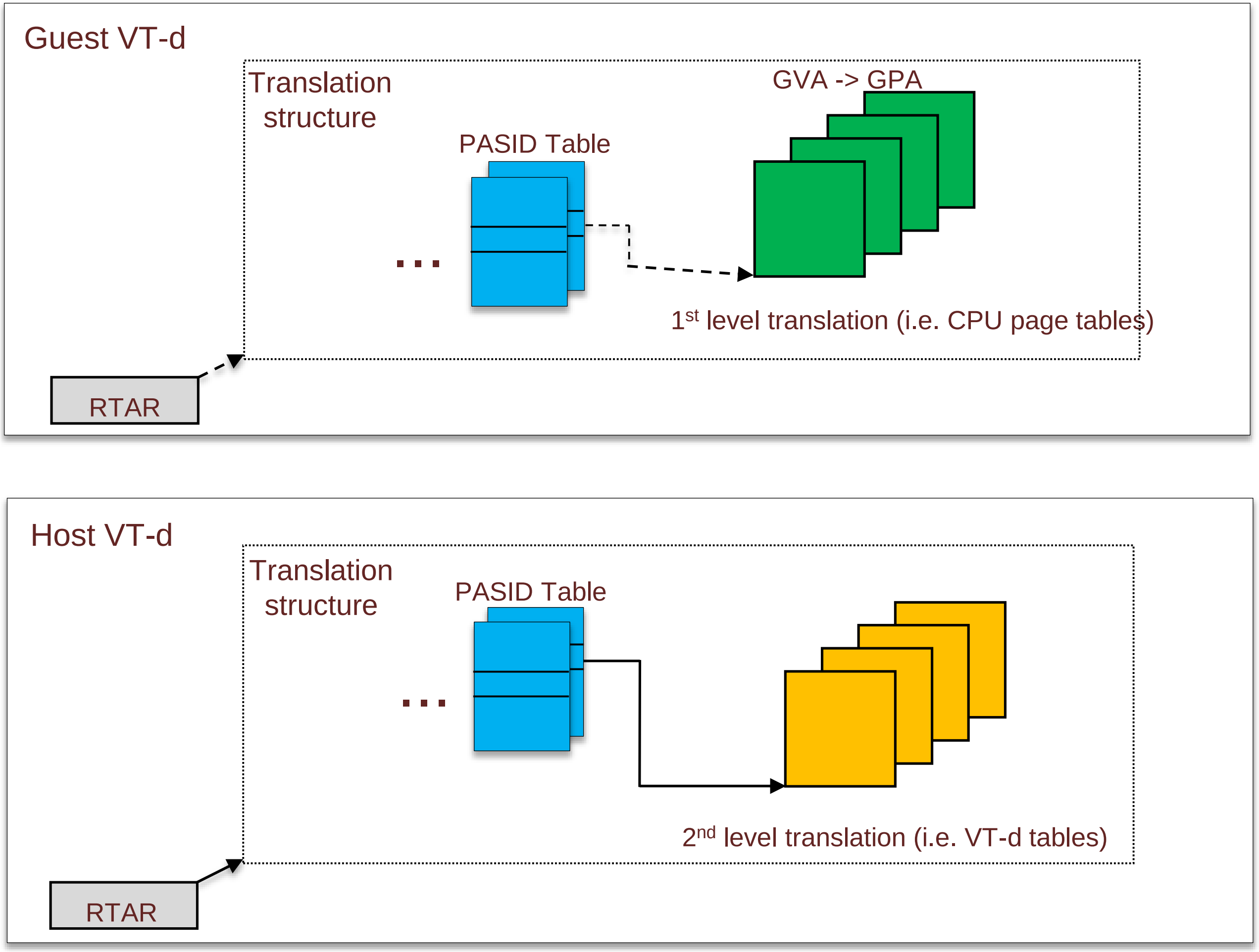
Summary

- Shared Virtual Addressing(SVA) enables efficient workload submission by directly programming CPU virtual addresses on the device
- Intel® VT-d 3.0 specification extends SVA usage together with Intel® Scalable I/O Virtualization
- Holistic enhancements are introduced cross multiple kernel/user space components, to enable SVA virtualization in KVM
- New kernel APIs are kept neutral to support all kinds of virtual IOMMUs (either emulated or para-virtualized)

Q/A

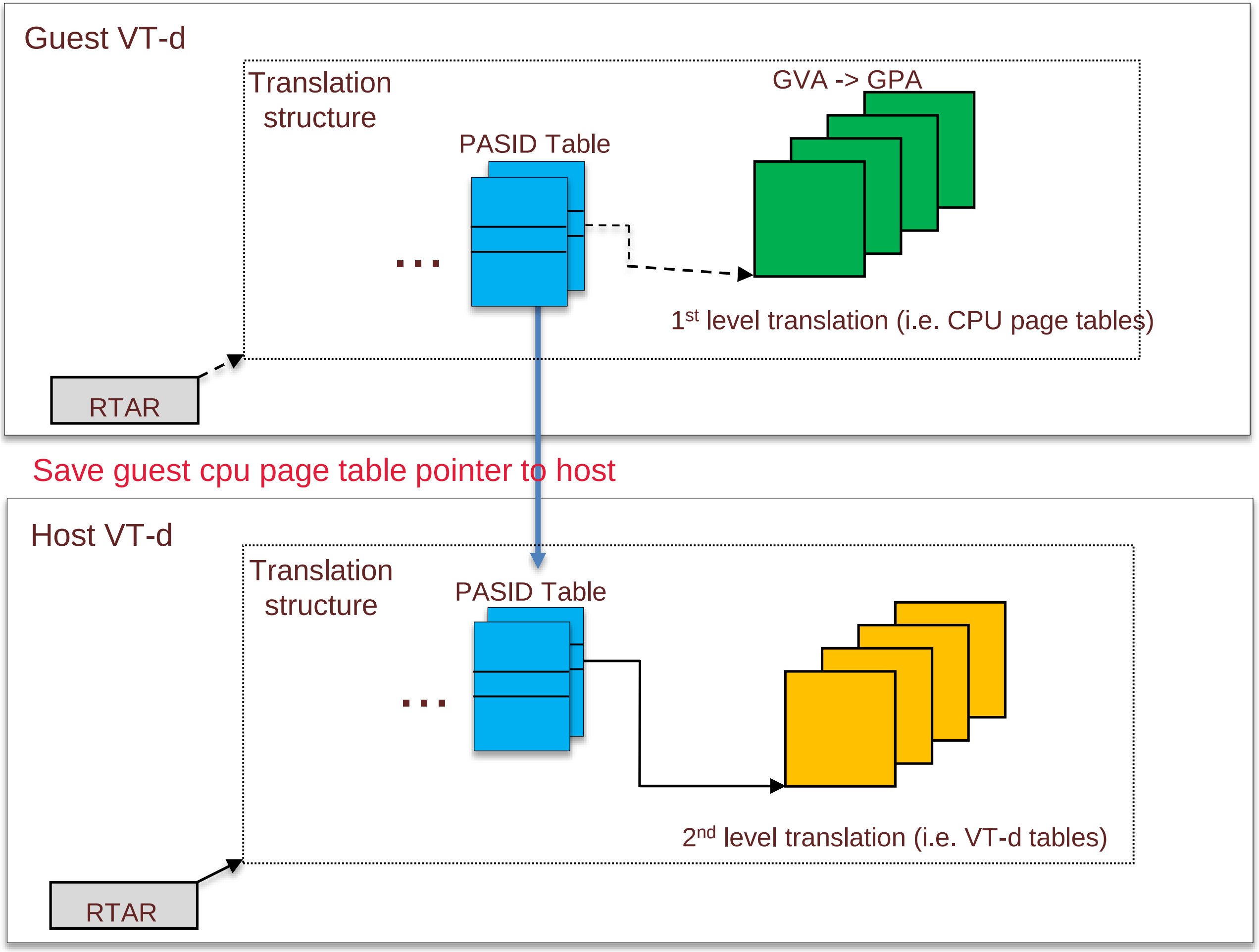
Backup

Enable SVA in VM (Cont.)



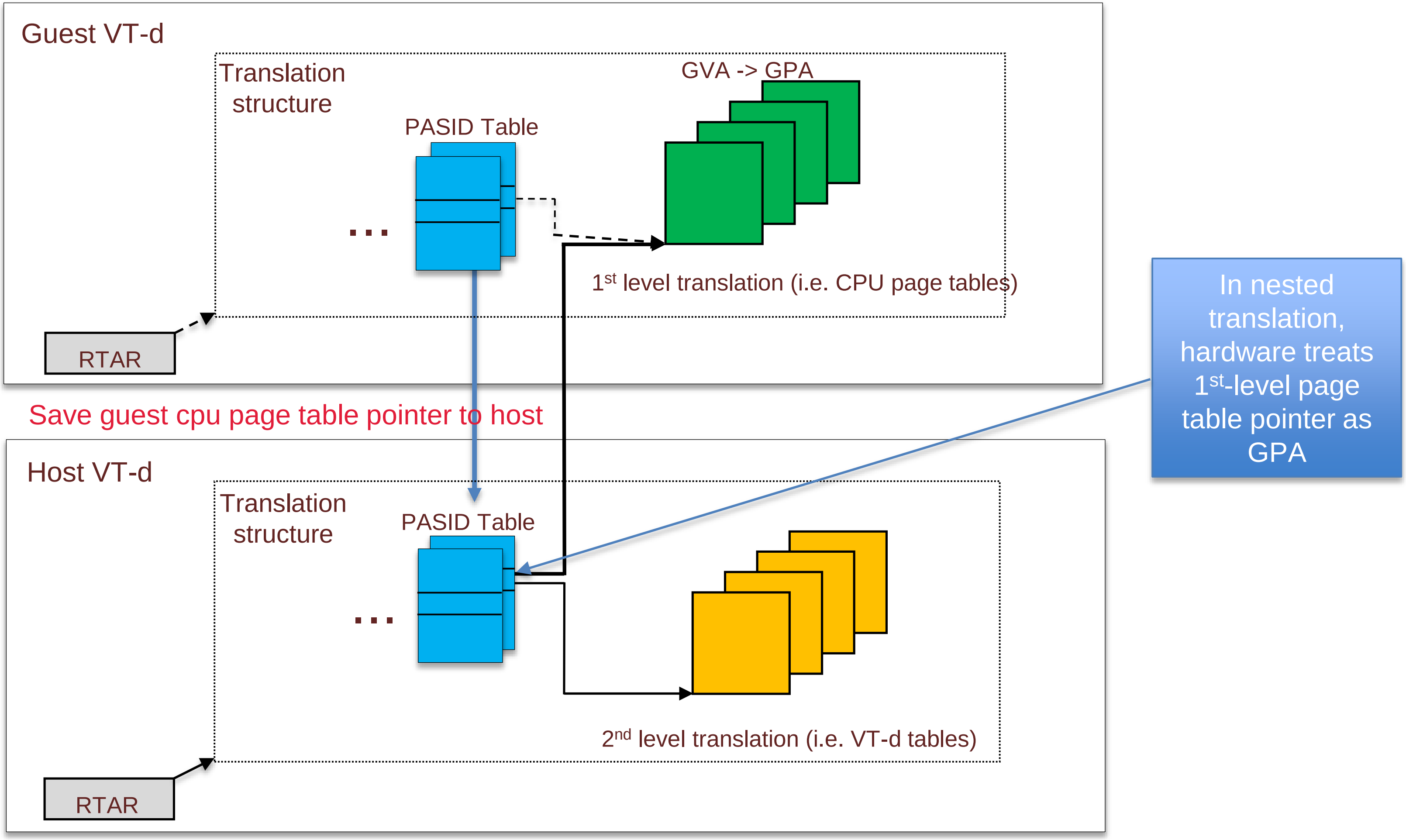
Guest SVA Support on Intel® VT-d

Enable SVA in VM (Cont.)


Guest VT-d
Translation structure
GVA -> GPA
PASID Table
…
1st level translation (i.e. CPU page tables)
RTAR
Save guest cpu page table pointer to host
Host VT-d
Translation structure
PASID Table
…
2nd level translation (i.e. VT-d tables)
RTAR

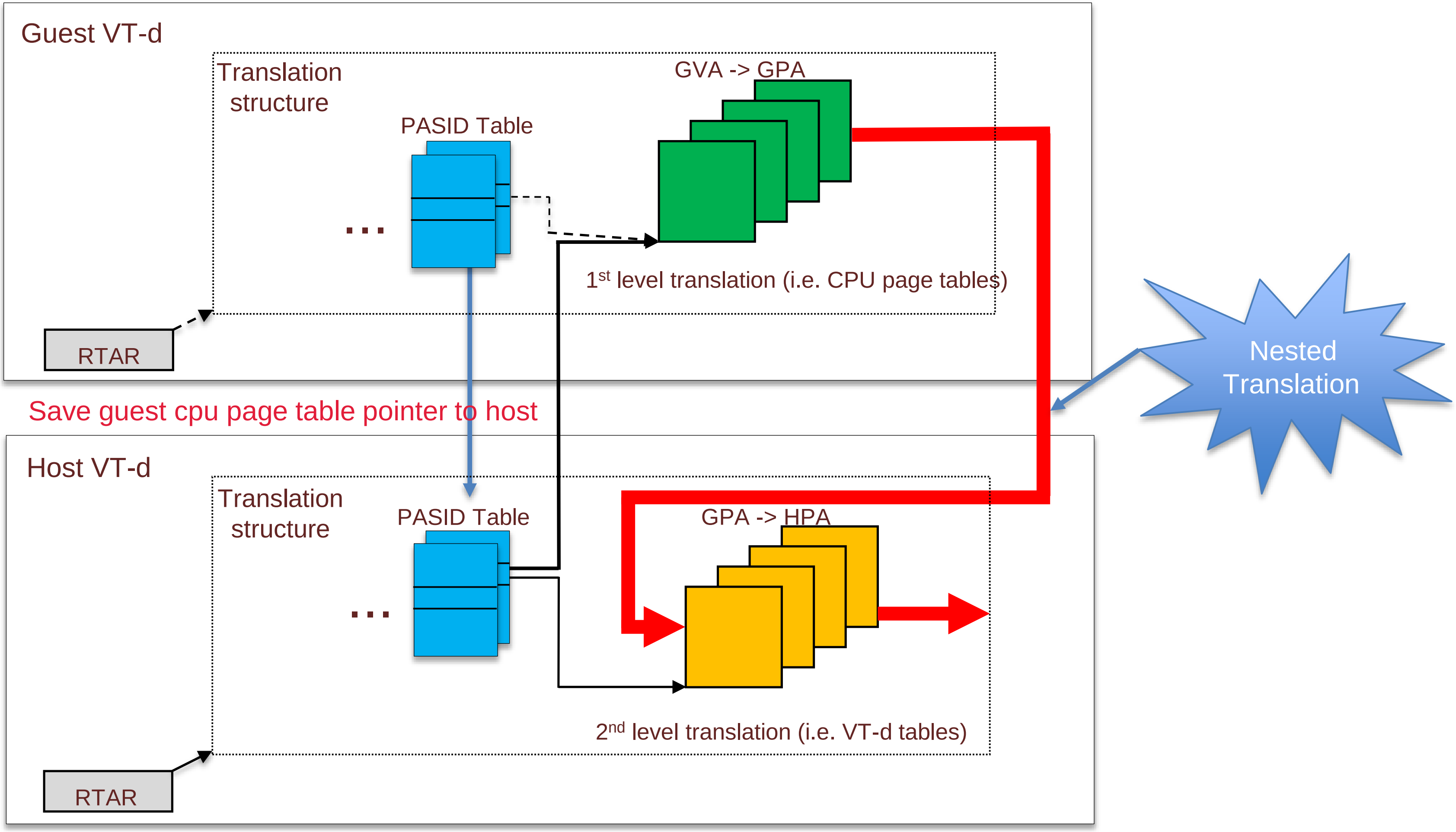
Guest SVA Support on Intel® VT-d

Enable SVA in VM (Cont.)


Guest VT-d
Translation structure
GVA -> GPA
PASID Table
...
1st level translation (i.e. CPU page tables)
RTAR
Save guest cpu page table pointer to host
Host VT-d
Translation structure
PASID Table
...
2nd level translation (i.e. VT-d tables)
RTAR
In nested translation, hardware treats 1st-level page table pointer as GPA

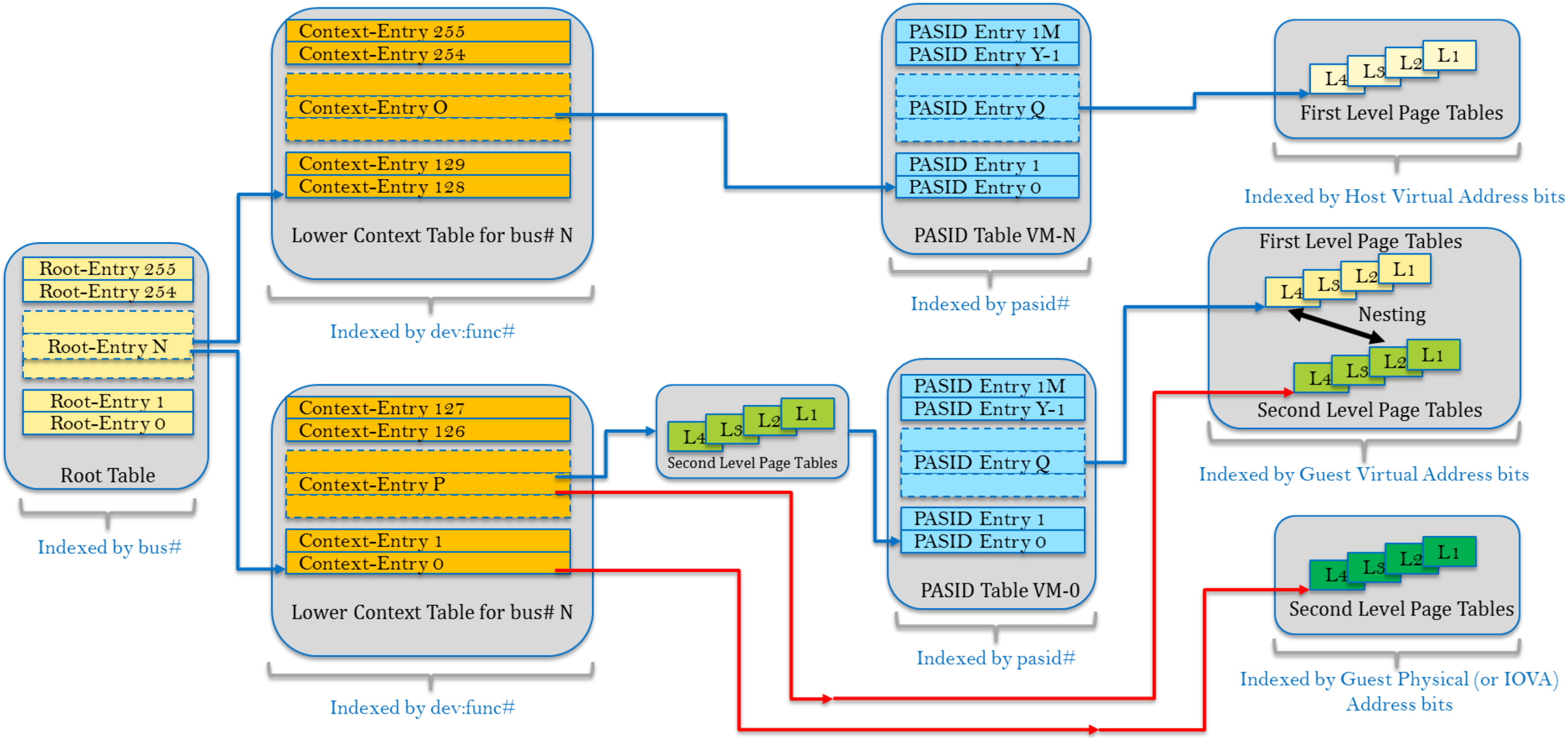
Guest SVA Support on Intel® VT-d

Enable SVA in VM (Cont.)

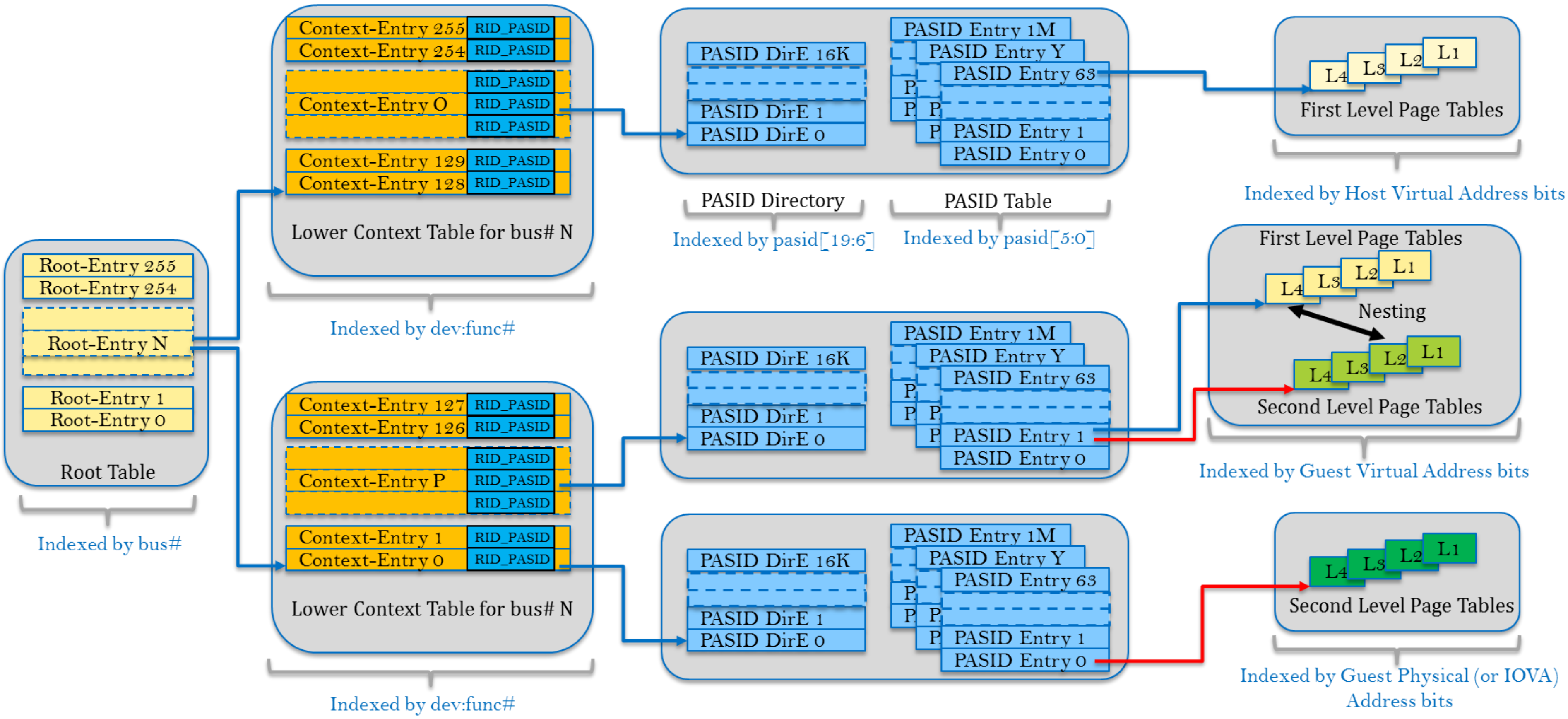


Guest SVA Support on Intel® VT-d

VT-d Extended Context Mode (Deprecated)



VT-d Scalable Mode (New)



Key Difference: PASID is a global ID space shared by all VMs.

ALL page-table pointers moved to PASID Granular table