Shared Virtual Addressing for high performance Arm Infrastructure platforms

Embedded Linux Conference 2021

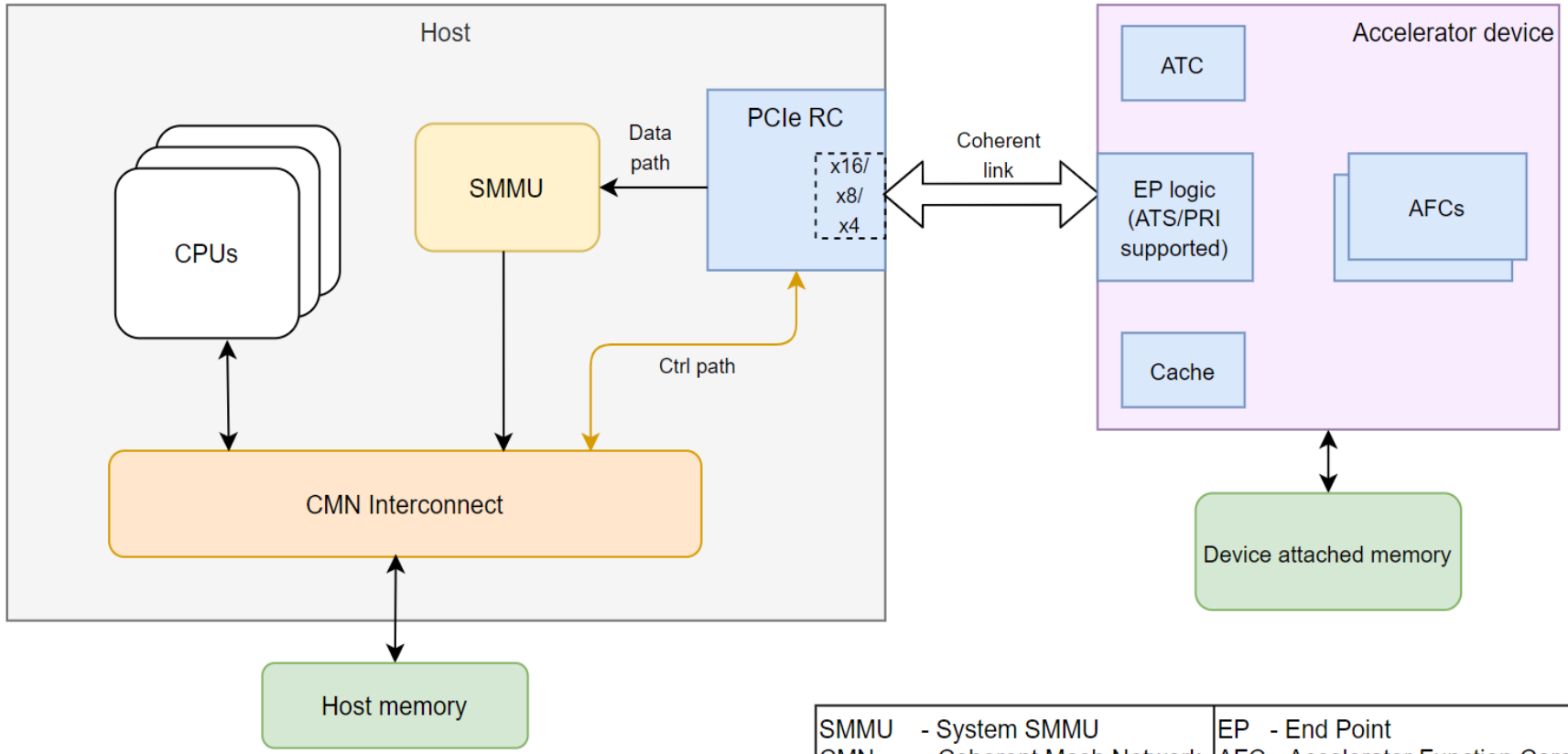
Vivek Kumar Gautam
Open Source Software Group, Arm Ltd

Agenda

- Introduction to Shared Virtual Addressing (SVA)
- Hardware and Software requirements
- Virtualization
- Current design for vSVA
- Upstream status

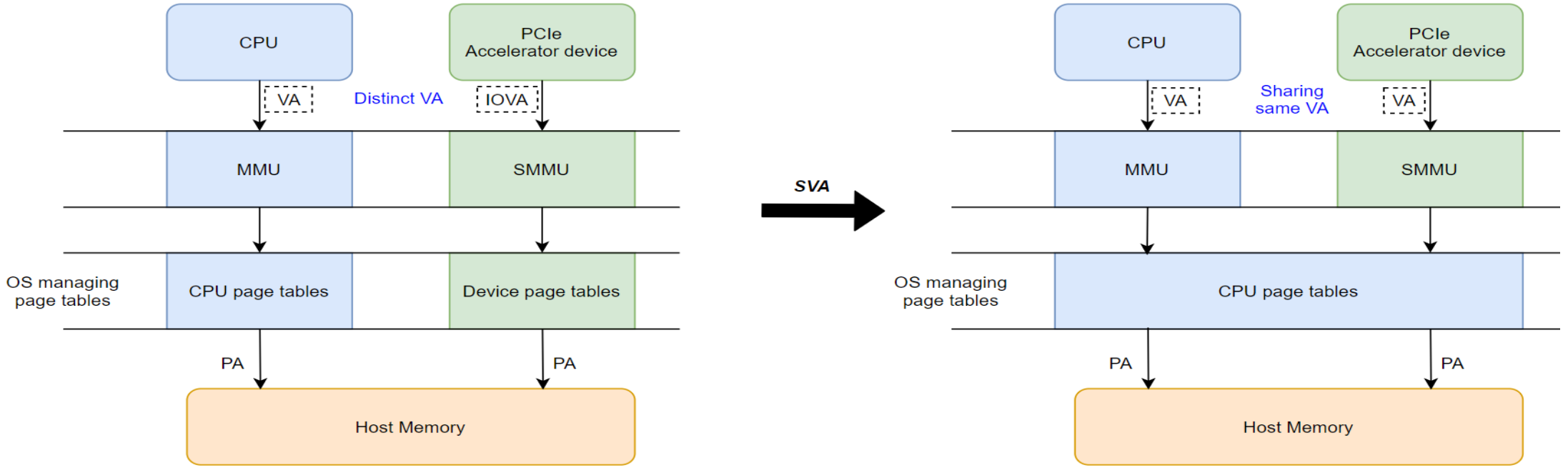
Why Shared virtual addressing

- Arm infrastructure platforms.
- High performance accelerators – such as GPGPUs, NPUs, SmartNICs etc. are used in infrastructure platforms.
- PCIe – the backbone bus of infrastructure.
- Hardware coherency requirement for accelerators.



SMMU - System SMMU	EP - End Point
CMN - Coherent Mesh Network	AFC - Accelerator Function Core
PCIe RC - PCIe root complex	ATC - Address Translation Cache

What is Shared virtual addressing (SVA)



Adjacent methodologies

- OpenCL 2.0 Shared virtual memory
- HSA Shared virtual memory
- CUDA Unified virtual memory

- SVA allows sharing same virtual address space between IO devices (accelerators) and application processors.
- The ability to perform DMA on a process address space rather than using a separate DMA address space.

arm

Advantages of SVA

- No duplicate or additional set of page tables for device.
- Reduced programming complexity:
  - The requirement to have specialized driver to take care of DMA buffer is gone with SVA.
  - Sharing data becomes as simple as passing a pointer between processors and devices (accelerators).
- Fewer cache and tlb maintenance operations.
- Cache coherency and Shared virtual addressing:
  - For fully coherent systems, the SVA solution allows CPU and accelerator (device) to share physical memory while working with same virtual address space.
  - Zero cache maintenance operations needed.
  - CPU and device can work on the buffer at the same time using atomic operations – e.g., PCIe also provides atomic operations.

Requirements of SVA

PCIe Address Translation Service (ATS) – mem access type ATS TLP.

PCIe Page Request Interface (PRI) – message type TLP for PRI

PCIe Process Address Space ID (PASID) – PASID TLP prefix

PCIe Device features

SMMU-v3 support for ATS

SMMU-v3 support for PRI – PRI Queue

SMMU-v3 support for PASID – SubStream ID

SMMU-v3 support for Nested translation – StreamTableEntry and ContextDescriptor

Arm IP support

Paravirtualized IOMMU – virtio-iommu

Virtual Function IO framework (VFIO) – Device assignment

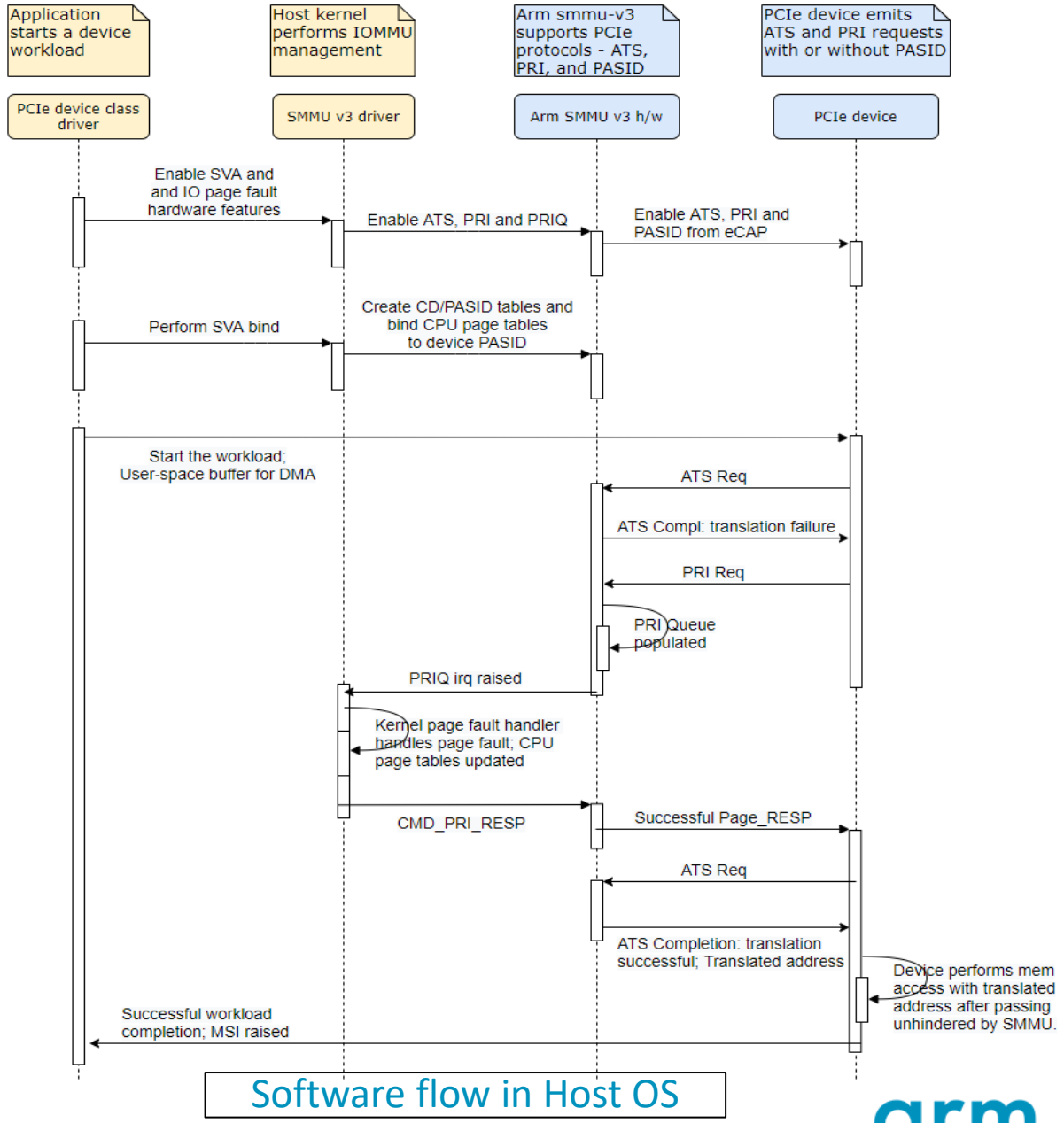
VMM (kvmtool) – vfio and virtio-iommu supported

Software support for Virtual machine

arm

Host SVA system

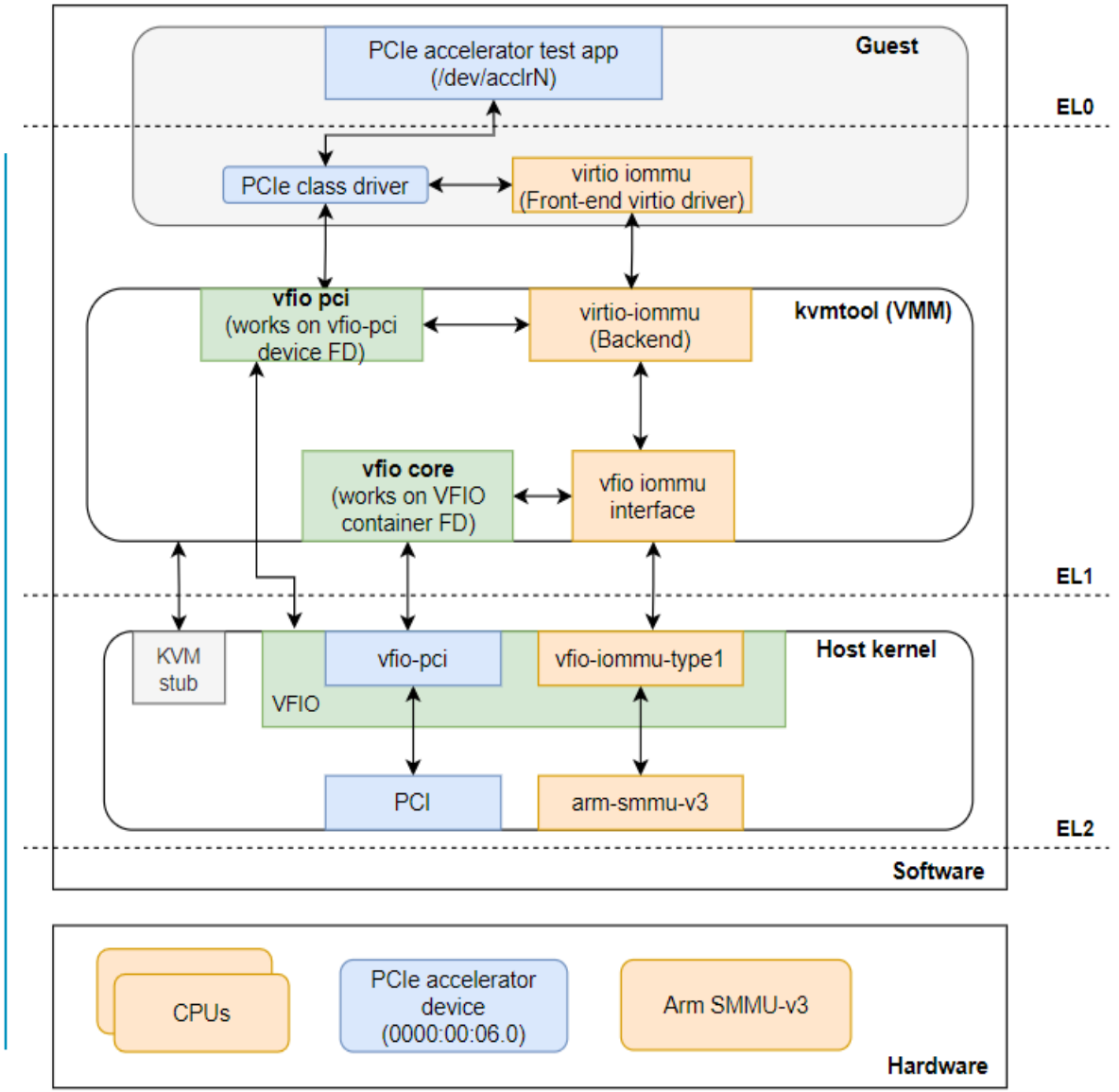
- Shared virtual addressing working in a host kernel environment involves:
  - SVA bind –
    - Any incoming PCIe transaction will contain PASID information.
    - SVA binding involves preparing bond between the process's 'struct mem' with the device PASID.
  - Page tables preparation –
    - Preparing SMMU-v3 Context descriptor (CD) tables with CPU page table info.
  - I/O page fault handler –
    - Interrupt handler for PRIQ of SMMUv3 that is populated from incoming PRI requests.
    - Calls kernel page fault handler to update CPU page tables to handler page fault.
    - Sends CMD_PRI_RESP once page fault is handled.



Software flow in Host OS

Virtualization and vSVA

- Key components:
  - PCIe passthrough – via vfio-pci.
  - DMA remapping in guest - Nested translation support – via emulated/para-virtualized iommu device.
  - DMA fault handling – via VFIO-IOMMU interface.
  - PCI Page request interface for I/O page fault – via PCIe and arm-smmu-v3 PRI support.

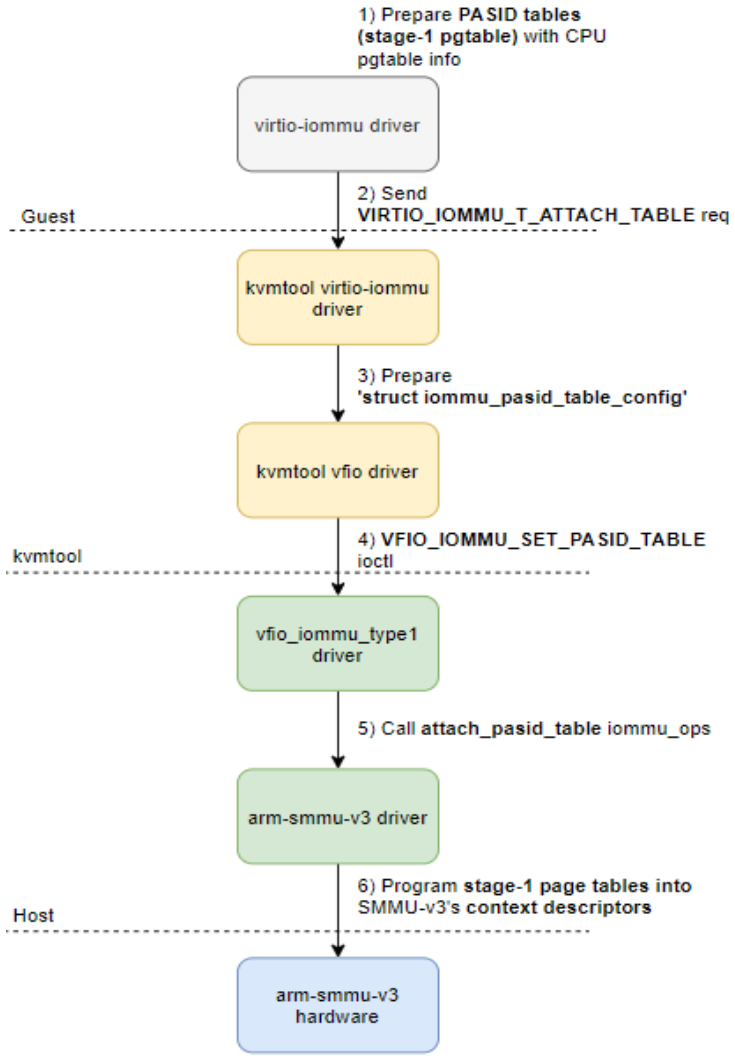


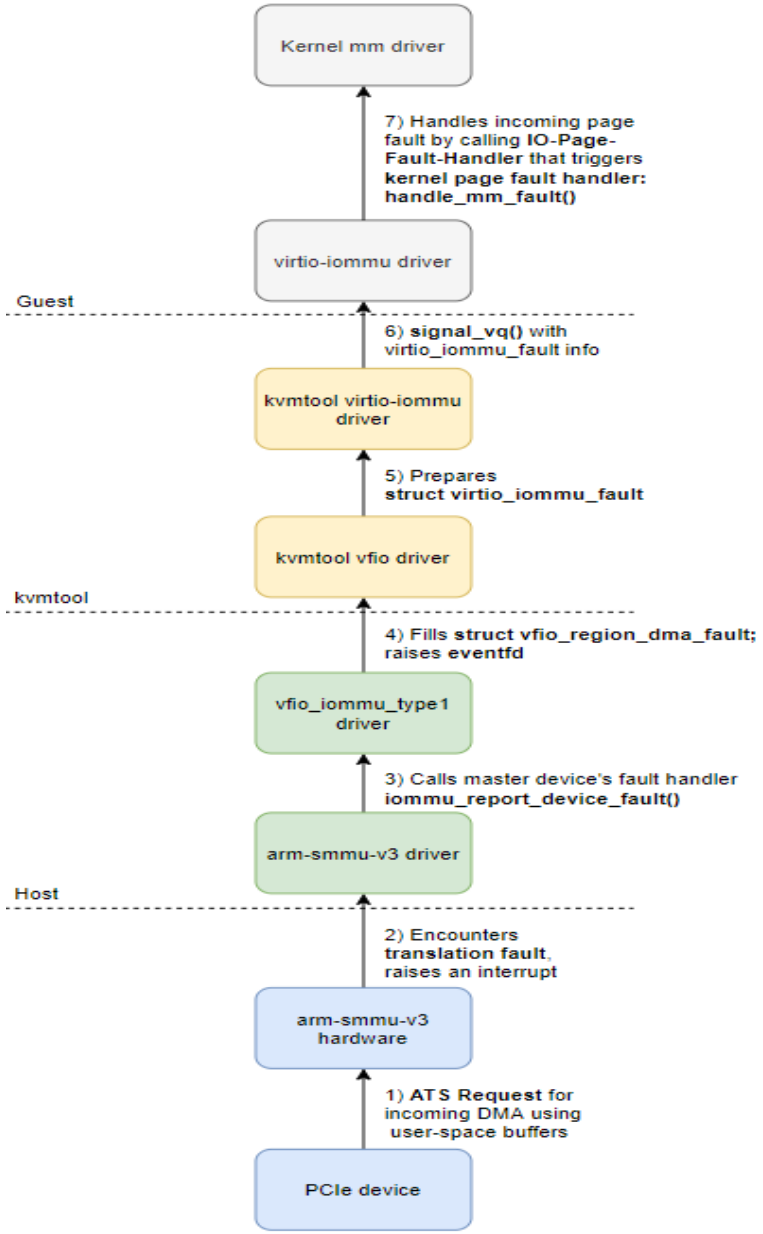
Current design of vSVA

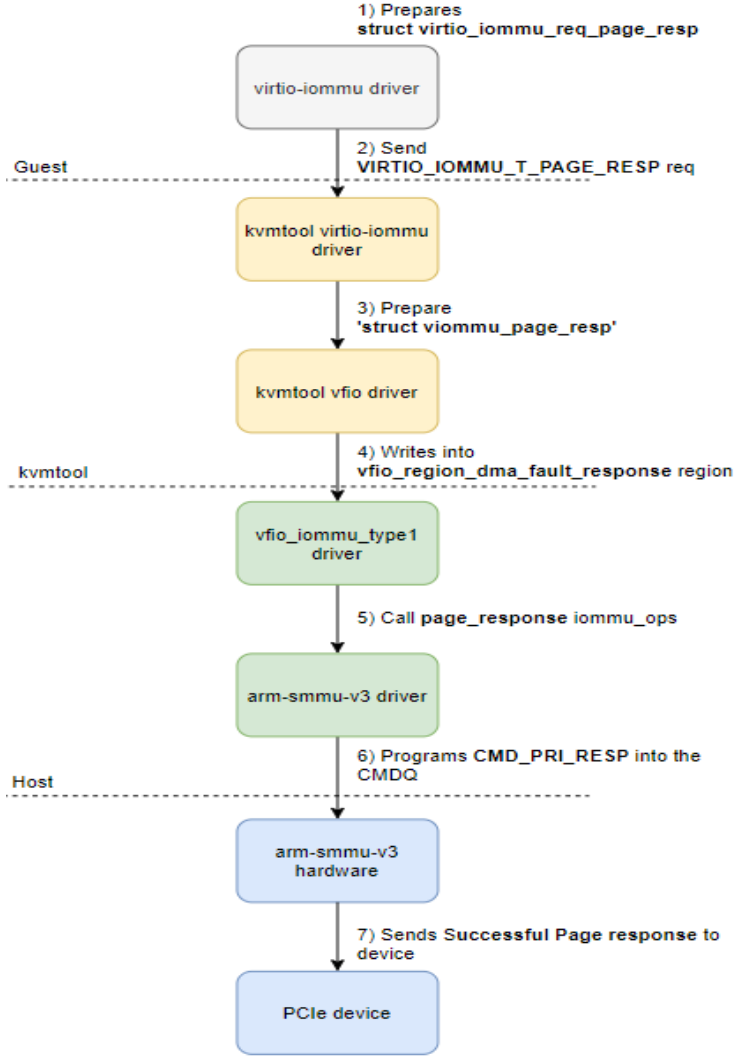
- Built on top of VFIO and IOMMU API changes by Eric Auger –
  - [PATCH v15 00/12] SMMUv3 Nested Stage Setup (IOMMU part)
  - [PATCH v13 00/13] SMMUv3 Nested Stage Setup (VFIO part)
- On going IOMMU user-API proposal –
  - [RFC v2] /dev/iommu uAPI proposal
- Design for virtio-iommu should be mostly independent of IOMMU/VFIO uAPI changes.
- Few additions proposed to virtio-iommu specification and driver:
  - virtio-iommu requests:
    - VIRTIO_IOMMU_T_ATTACH_TABLE
    - VIRTIO_IOMMU_T_INVALIDATE
    - VIRTIO_IOMMU_T_PAGE_RESP
  - virtio-iommu feature bits:
    - VIRTIO_IOMMU_F_ATTACH_TABLE
    - VIRTIO_IOMMU_F_SVA

vSVA flow in VM

DMA map

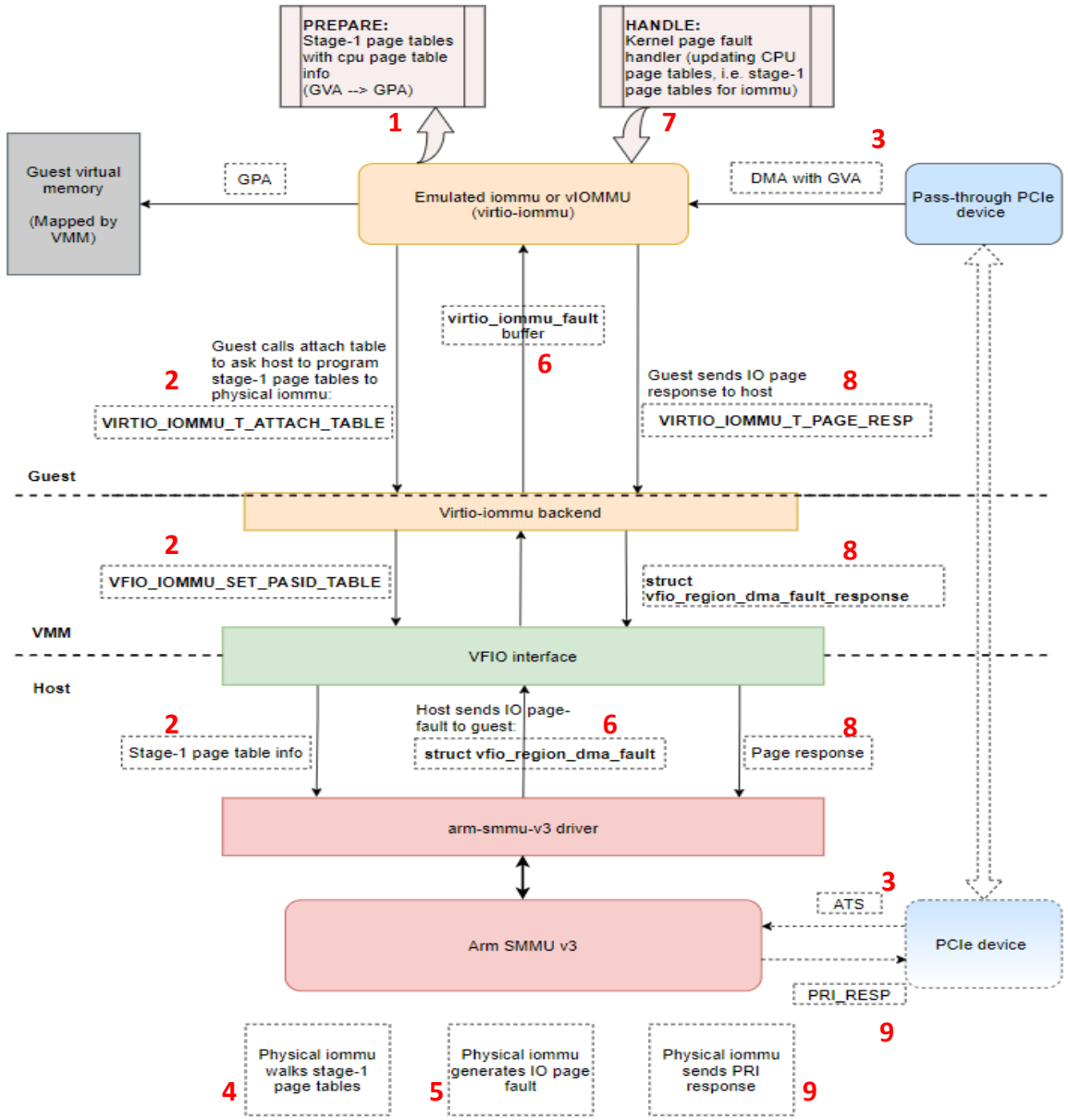






vSVA flow (contd.)

- DMA unmap: Invalidation request from guest virtio-iommu to host arm-smmu-v3 via vfio to invalidate TLBs (tagged with PASID) or Context Descriptor cache.



Upstreaming status

Linux kernel

- Major work has been done towards enabling SVA support in guest using virtio-iommu.
- The changes include support for nested page table, support to handle dma fault from host kernel, and to send page fault response from guest kernel.
- Patches:
  - Nested page table support with virtio-iommu:
    [PATCH RFC v1 00/15] iommu/virtio: Nested stage support with Arm
  - Patches to add SVA support to virtio-iommu:
    [PATCH RFC v1 00/11] iommu/virtio: vSVA support with Arm
  - Publishing next version soon.
- Future virtio-iommu changes will incorporate dependencies arising from IOMMU uAPI proposal.

kvmtool

- SVA changes on kvmtool based on virtio-iommu driver that was added by Jean Philippe (not yet upstreamed):
  - https://jpbrucker.net/git/kvmtool/log/?h=virtio-iommu/devel

Thank you!

Questions?

arm

Thank You
Danke
Gracias
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה