IOMMU: How It Helps to Provide IO Security

Vasant Hegde <vasant.hegde@amd.com>

AMD
together we advance_

Agenda

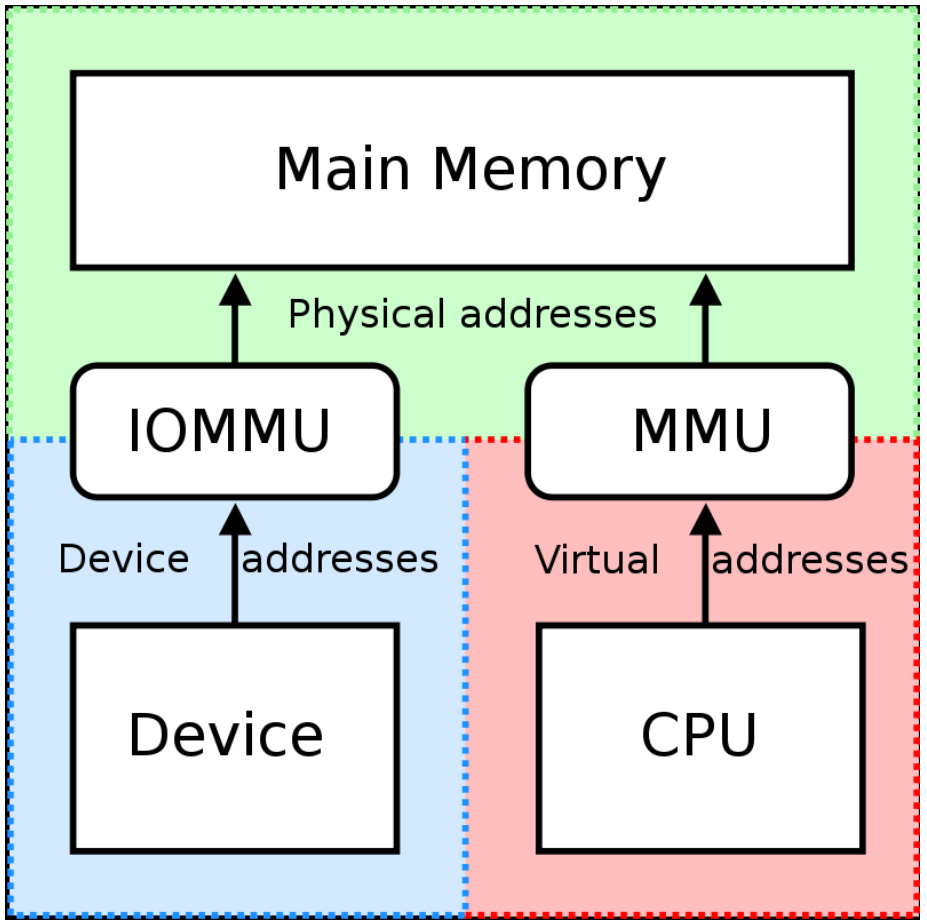
- Introduction
- DMA remapping
- Interrupt remapping
- Shared Virtual Addressing (SVA)
- vIOMMU

Motivation for IOMMU

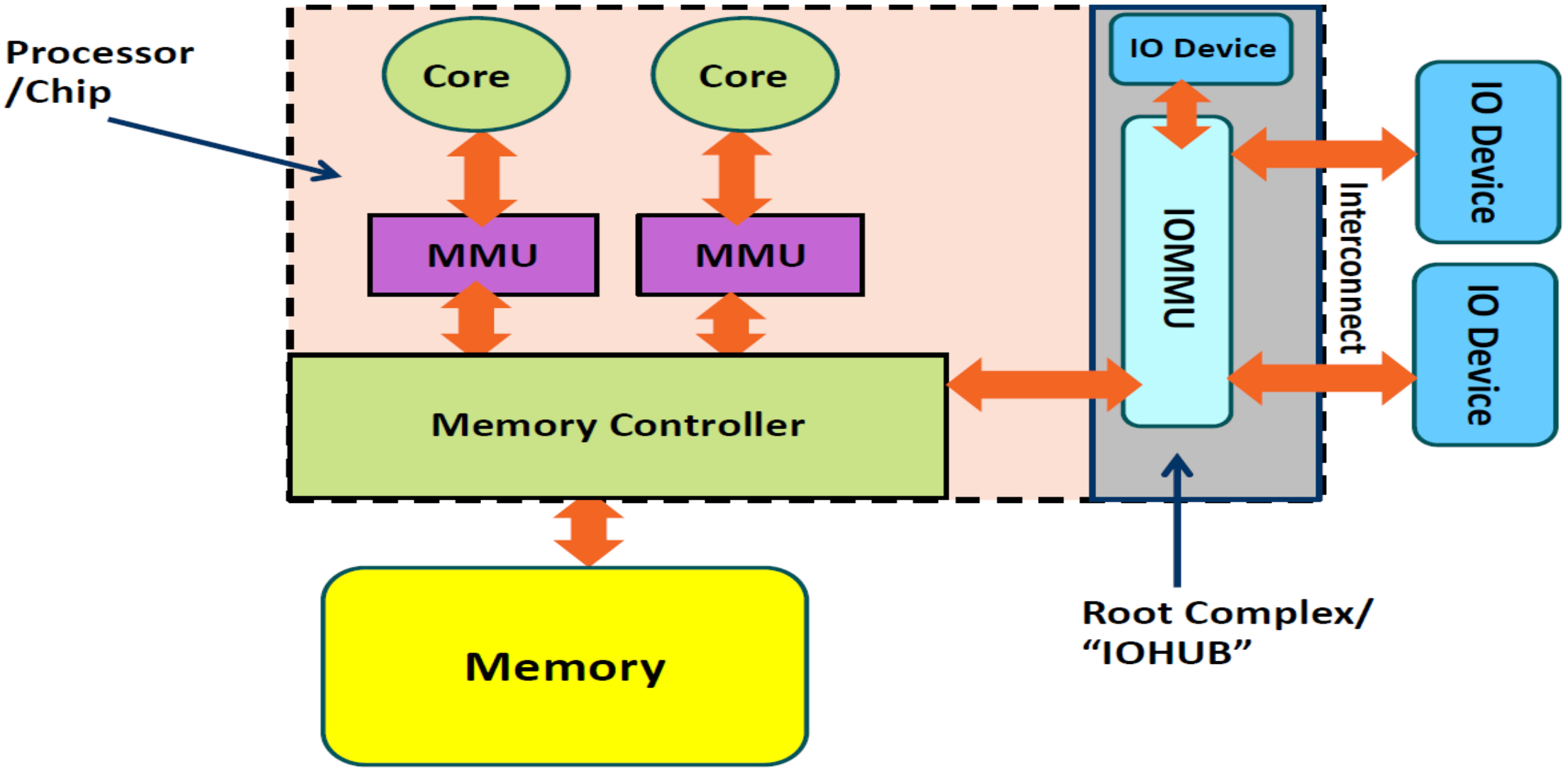
- IOMMUs was originally promoted along with 64-bit x86 processors
  - Support DMA from peripheral devices that support 32-bit addresses only
  - Currently vast majority of peripherals support 64-bit DMA
    - Those devices that do not support 64-bit can get buffers in the 32-bit range.
- Today IOMMU is center of IO security
    - Protects system from malicious IO devices
    - Provides trusted IO access to confidential guest

What is an IOMMU?

- Like Memory Management Unit (MMU) on a CPU, but for IO devices
- Creates per domain virtual address spaces for DMA
  - Domain per IOMMU group (devices are grouped for isolation in IOMMU groups)
- Provides remapping services for interrupts
- Useful on bare metal for device isolation and addressing limitations
- Useful for virtualization that includes pass-through of devices (VFIO) into a guest environment



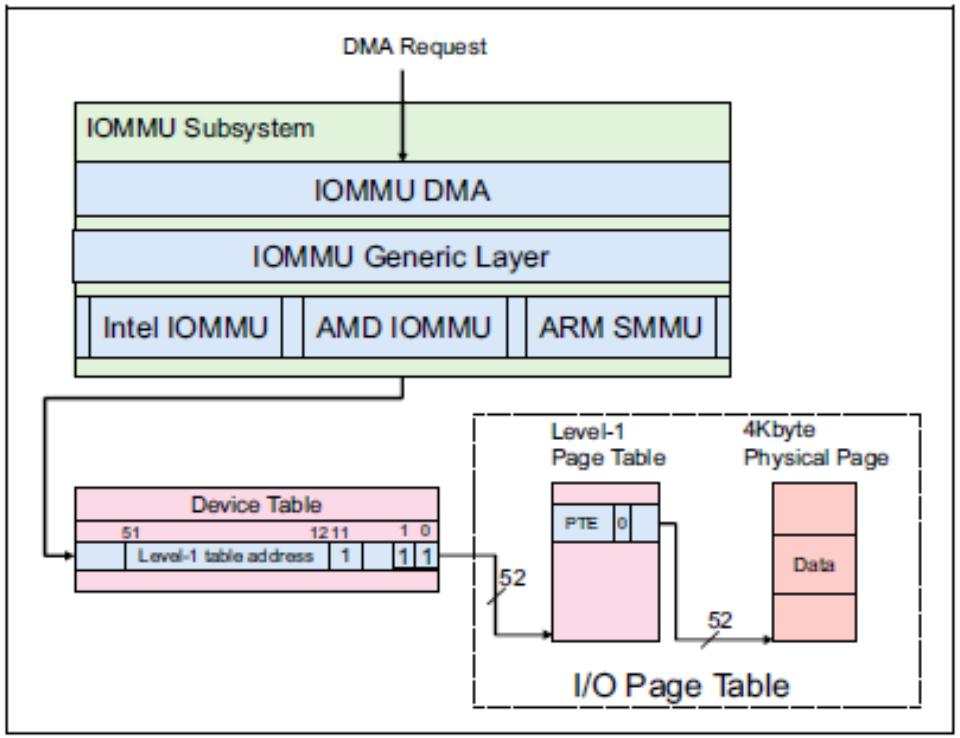
Physical View



Reference : https://pages.cs.wisc.edu/~basu/isca_iommu_tutorial/IOMMU_TUTORIAL_ASPLOS_2016.pdf

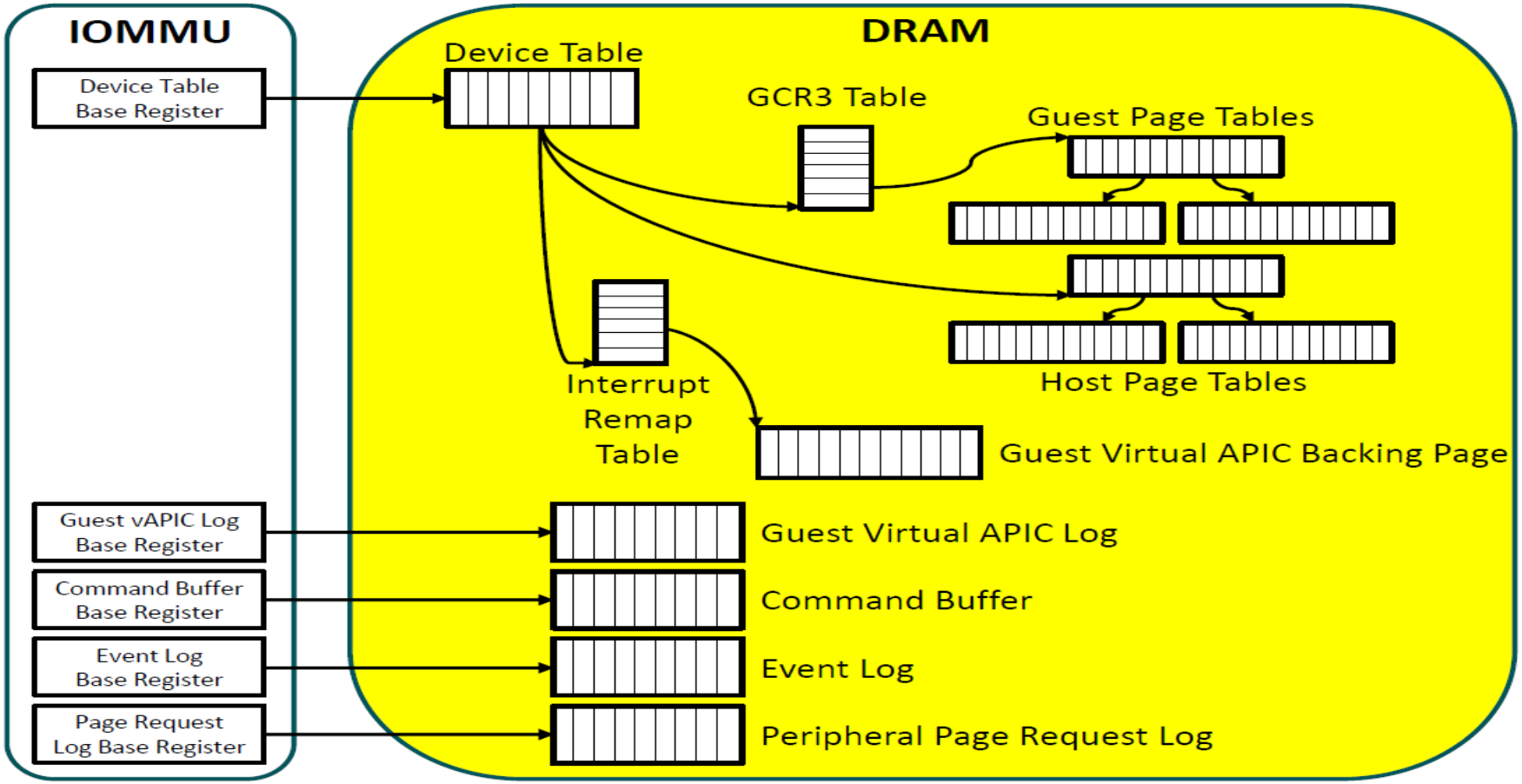
IOMMU Subsystem in Linux Kernel – High-Level Overview

- On x86 system IOMMU enabled by default if the IOMMU is present in the system
  - Even on bare metal!
- Every IOMMU domain gets its own IO virtual address space
  - IOMMU domain is often a single device, but could be multiple devices (“device group”)
  - Cannot access memory that is not mapped into its address space
  - Can utilize features like memory encryption
  - Can access memory beyond the devices IO address limitations (e.g., 32-bit device on a CPU with a 52-bit address space)
- Hypervisors use IOMMUs to create virtual address space for devices passed through to the guest
- IOMMU command line parameters are a mix of generic and vendor specific options
  - iommu=pt – Use a 1:1 mapping from IOVA to SPA
  - amd_iommu=<options> - AMD driver specific options
  - intremap=on/off – Interrupt remapping
  - https://www.kernel.org/doc/html/latest/admin-guide/kernel-parameters.html



Reference : https://lenovopress.lenovo.com/lp1467-an-introduction-to-iommu-infrastructure-in-the-linux-kernel

AMD IOMMUs Key Data Structure

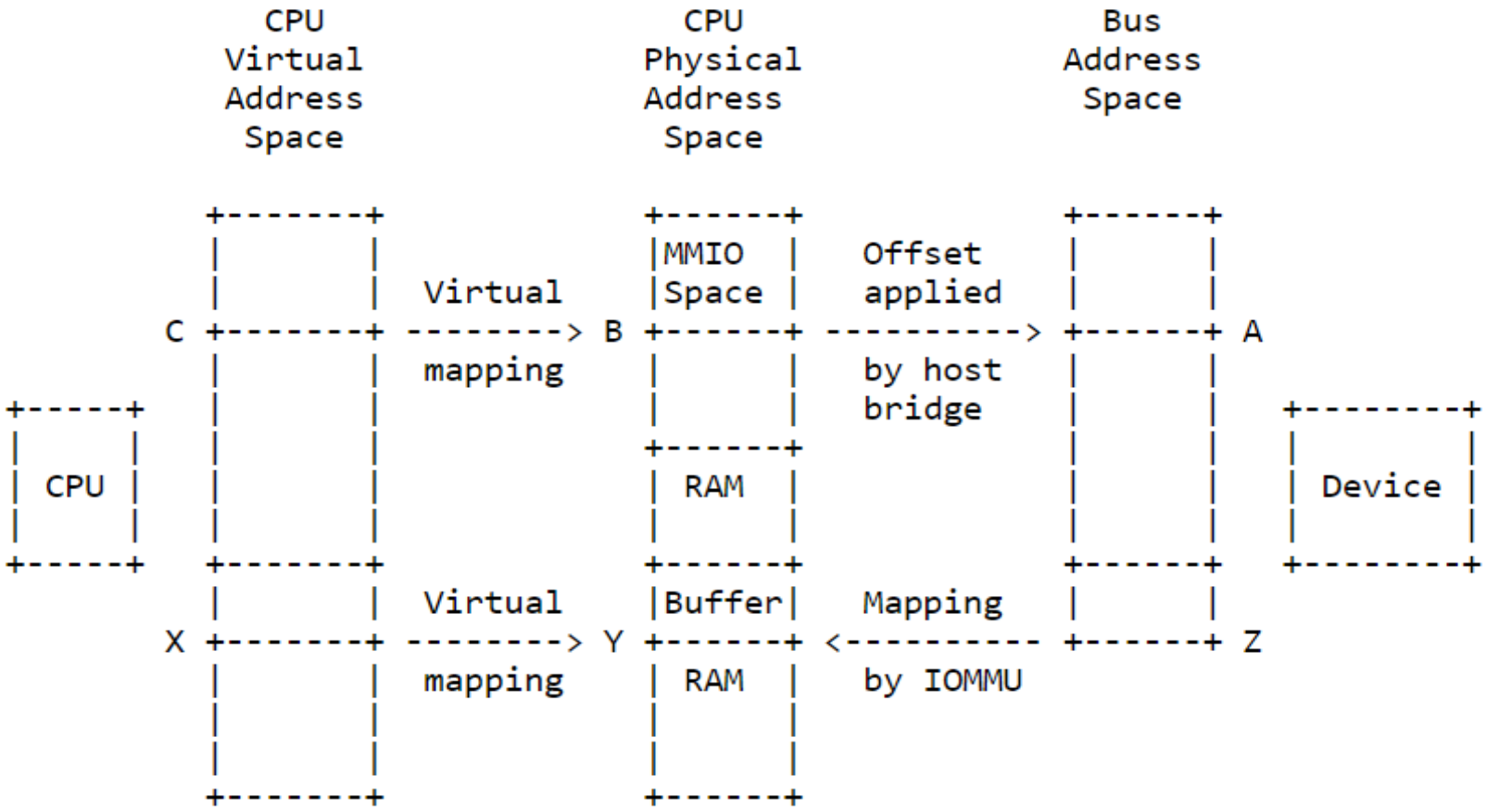


Reference : https://pages.cs.wisc.edu/~basu/isca_iommu_tutorial/IOMMU_TUTORIAL_ASPLOS_2016.pdf

DMA Translation

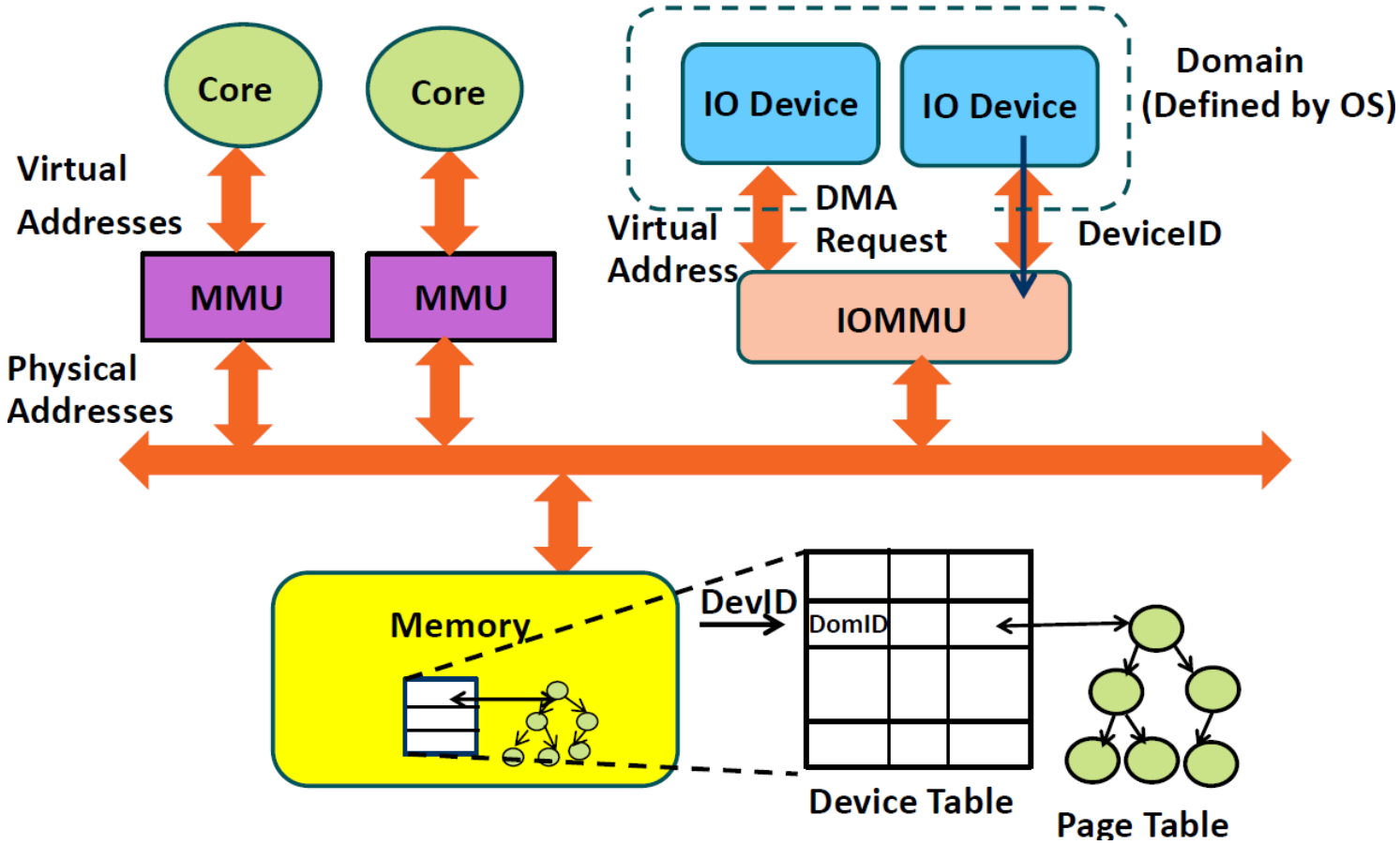
- Address translation and memory protection in host system
- Making address translation faster through IOMMU cache
- Enabling shared address space in heterogeneous system
- Enabling pre-translation through IOMMU
- Enabling demand paging from devices (dynamic page fault)
- Nested address translation in virtualized system
- Invalidating IOMMU mappings

DMA Address View



Reference : https://www.kernel.org/doc/Documentation/DMA-API-HOWTO.txt

DMA Translation and Memory Protection



Reference : https://pages.cs.wisc.edu/~basu/isca_iommu_tutorial/IOMMU_TUTORIAL_ASPLOS_2016.pdf

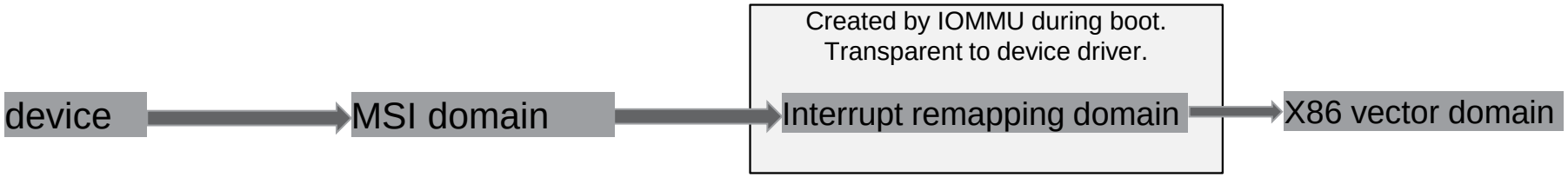
IOMMU DMA APIs

- *Implementation is completely transparent to device drivers*
  - IOMMU driver will setup *dma_ops* for the devices
  - Device driver will use *dma_** APIs

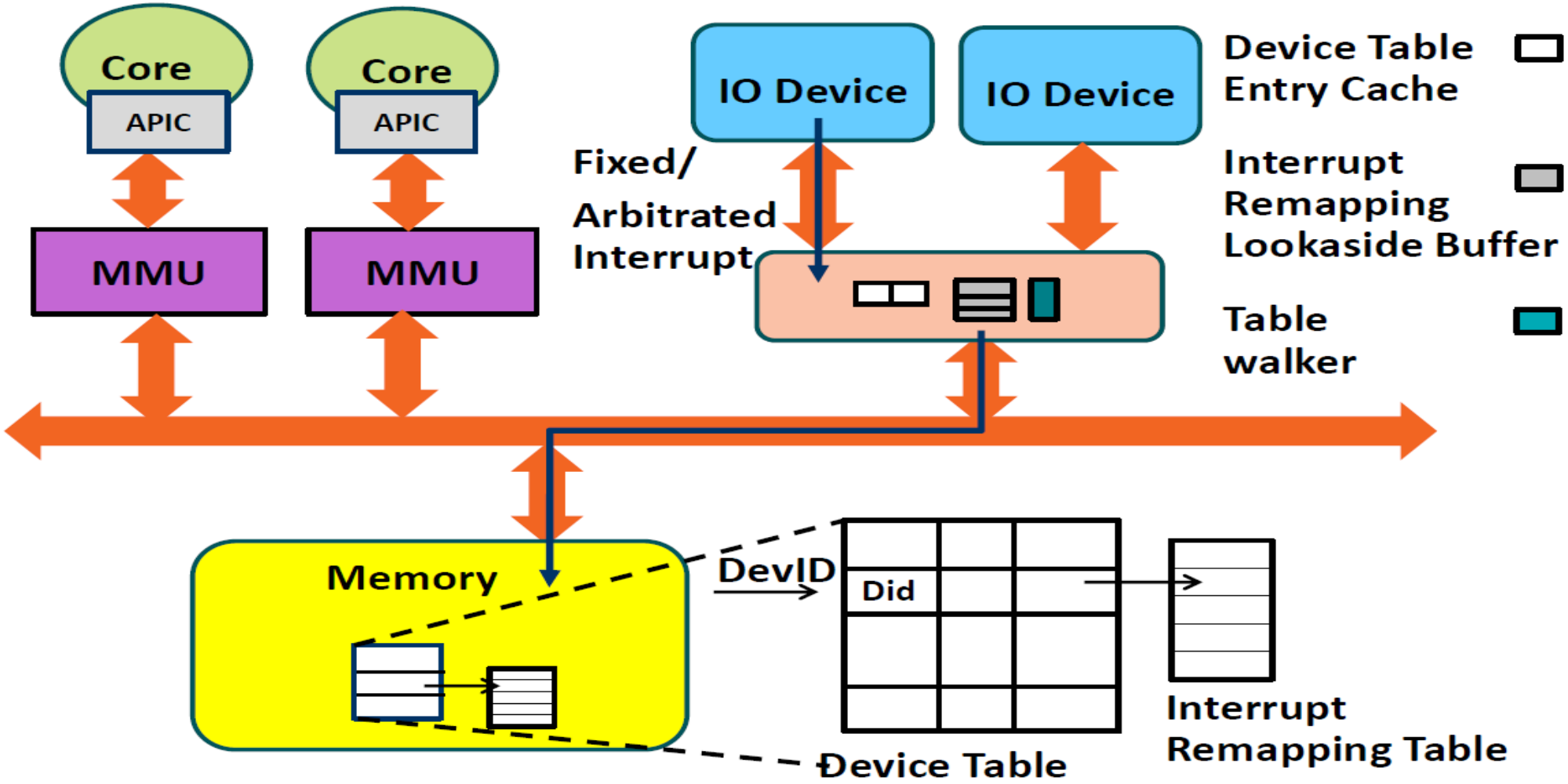
- /* include/linux/iommu.h */
  - extern int **iommu_map**(struct iommu_domain *domain, unsigned long iova,
    phys_addr_t paddr, size_t size, int prot, gfp_t gfp);
  - extern size_t **iommu_unmap**(struct iommu_domain *domain, unsigned long iova,
    size_t size);
  - extern size_**t iommu_unmap_fast**(struct iommu_domain *domain,
    unsigned long iova, size_t size,
    struct iommu_iotlb_gather *iotlb_gather);
  - extern ssize_t **iommu_map_sg**(struct iommu_domain *domain, unsigned long iova,
    struct scatterlist *sg, unsigned int nents,
    int prot, gfp_t gfp);

Interrupt Remapping implementation

- Interrupt remapping is transparent to device
- During boot, IOMMU driver creates interrupt remapping domain and sets *ir domain* in device structure
- During irq allocation, remapping domain allocates vector, stores it in remapping table and gives table index to device
- Device will use index provided by IOMMU to deliver the interrupt
- IOMMU will intercept the interrupt and uses remapping table to deliver interrupt to host



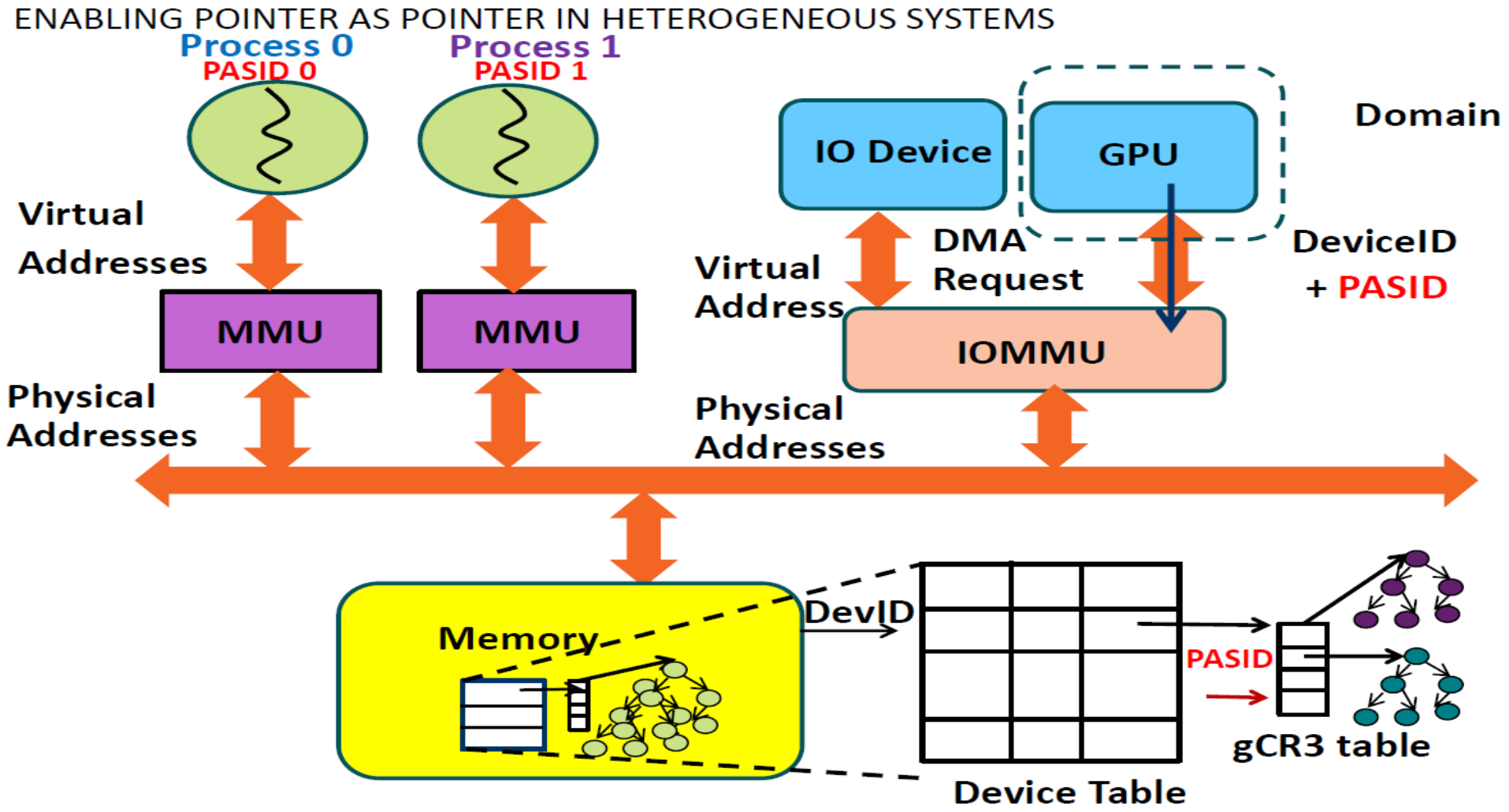
Interrupt remapping internal



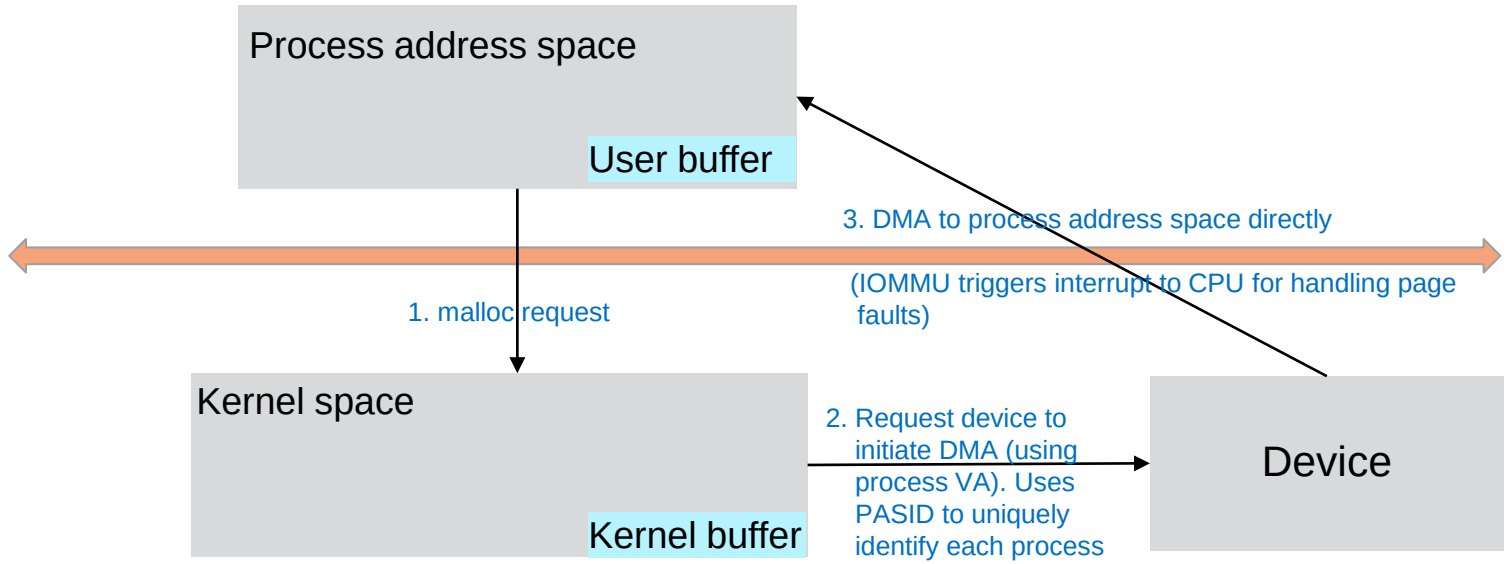
Shared Virtual Addressing (SVA)

- Devices use same virtual address with CPU
- Device can DMA to process address space
- Uses MMU for handling IO page faults
- Uses mmu notifier for TLB invalidations
- Use case
  - Pointer as pointer in heterogeneous system
  - DMA to user space address
  - No need to pin the memory
    - Device can handle page fault

SVA Data Structures



SVA Flow

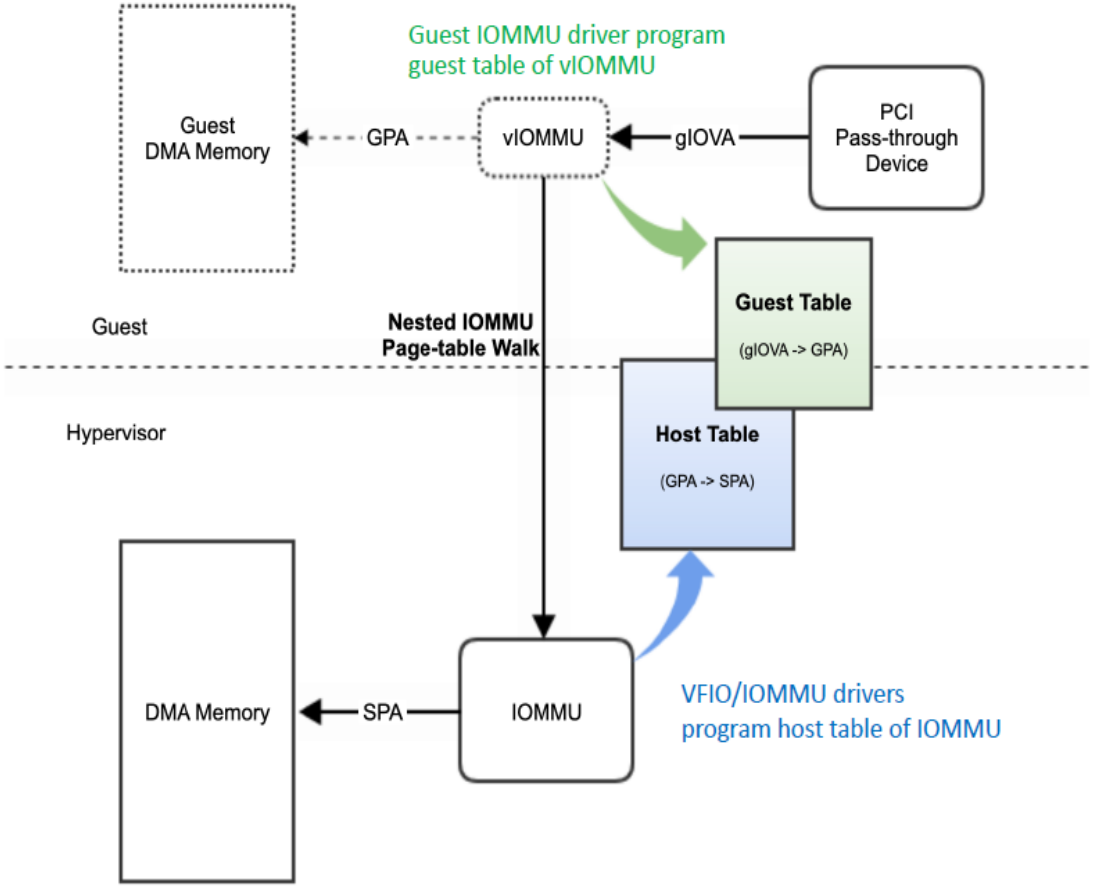


SVA Upstream Status

- AMD had special module in kernel for supporting SVA
- Later IOMMU core layer is enhanced to provide generic set of API for enabling SVA
- Intel and Arm SMMU v3 supports generic SVA APIs
- Working on enabling SVA support for AMD driver
  - https://lore.kernel.org/linux-iommu/20231016104351.5749-1-vasant.hegde@amd.com/T/#t

HW-vIOMMU

- Only support PCI pass-through devices
- No longer use **Shadow Host Table**
- Use **nested IO page-table**
  - Guest (x86-compatible) table for gIOVA -> GPA
    - Managed by guest IOMMU driver
  - Host (AMD-specific) table for GPA -> SPA
- Managed by VFIO/iommufd driver
- Use interrupt acceleration (AMD AVIC) for delivering interrupt to vCPU
- **Use case**
  - Guest IO protection
  - Guest shared virtual addressing
  - Nested pass-through devices



Reference : KVM forum 2020

vIOMMU Upstream Status

- Under heavy development in upstream
- It will be on top of *iommufd (/dev/iommu) interface*
- AMD vIOMMU implement
  - https://lore.kernel.org/linux-iommu/20230621235508.113949-1-suravee.suthikulpanit@amd.com/

Reference

- https://lenovopress.lenovo.com/lp1467-an-introduction-to-iommu-infrastructure-in-the-linux-kernel
- https://www.amd.com/en/support/tech-docs/amd-io-virtualization-technology-iommu-specification
- https://pages.cs.wisc.edu/~basu/isca_iommu_tutorial/IOMMU_TUTORIAL_ASPLOS_2016.pdf
- https://kvmforum2020.sched.com/event/eE3H/amd-viommu-a-hardware-assisted-virtual-iommu-technology-suravee-suthikulpanit-wei-huang-amd

COPYRIGHT AND DISCLAIMER

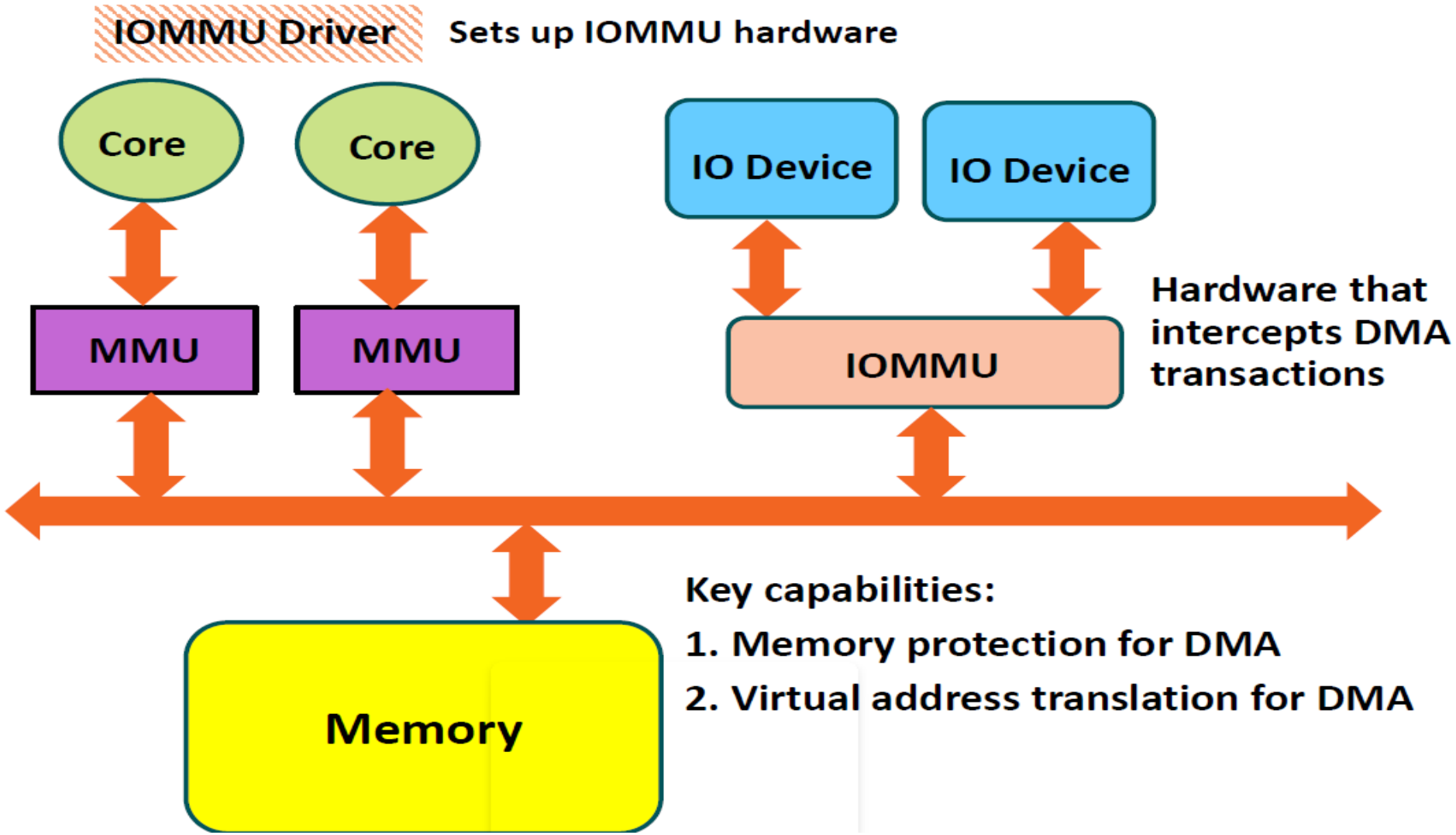
The information presented in this document is for informational purposes only and may contain technical inaccuracies, omissions, and typographical errors. The information contained herein is subject to change and may be rendered inaccurate releases, for many reasons, including but not limited to product and roadmap changes, component and motherboard version changes, new model and/or product differences between differing manufacturers, software changes, BIOS flashes, firmware upgrades, or the like. Any computer system has risks of security vulnerabilities that cannot be completely prevented or mitigated. AMD assumes no obligation to update or otherwise correct or revise this information. However, AMD reserves the right to revise this information and to make changes from time to time to the content hereof without obligation of AMD to notify any person of such revisions or changes.

THIS INFORMATION IS PROVIDED 'AS IS." AMD MAKES NO REPRESENTATIONS OR WARRANTIES WITH RESPECT TO THE CONTENTS HEREOF AND ASSUMES NO RESPONSIBILITY FOR ANY INACCURACIES, ERRORS, OR OMISSIONS THAT MAY APPEAR IN THIS INFORMATION. AMD SPECIFICALLY DISCLAIMS ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. IN NO EVENT WILL AMD BE LIABLE TO ANY PERSON FOR ANY RELIANCE, DIRECT, INDIRECT, SPECIAL, OR OTHER CONSEQUENTIAL DAMAGES ARISING FROM THE USE OF ANY INFORMATION CONTAINED HEREIN, EVEN IF AMD IS EXPRESSLY ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.


AMD

Backup

INTRODUCTION OF IOMMU: THE LOGICAL VIEW



Reference : https://pages.cs.wisc.edu/~basu/isca_iommu_tutorial/IOMMU_TUTORIAL_ASPLOS_2016.pdf