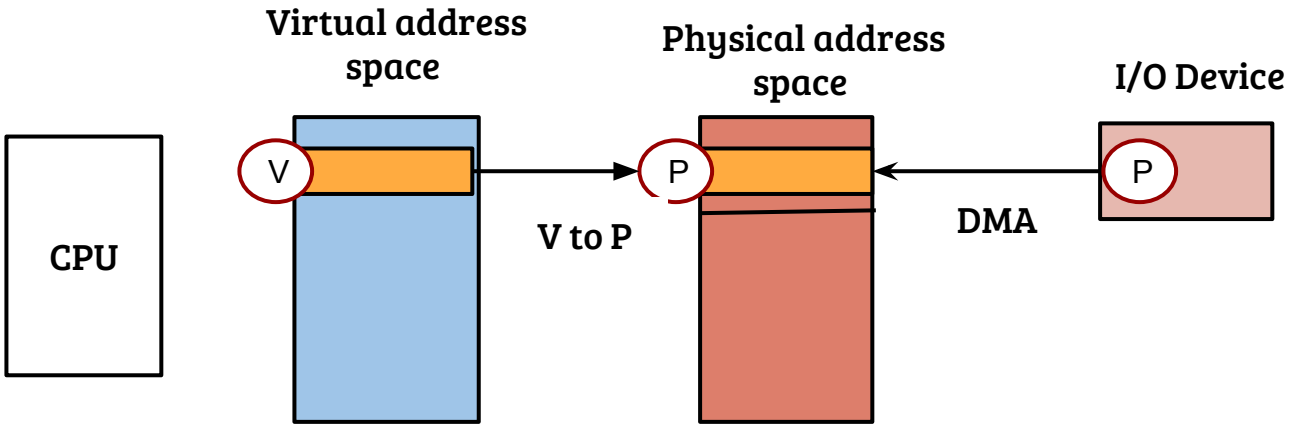
Topics in Operating Systems

DMA protection using IOMMU

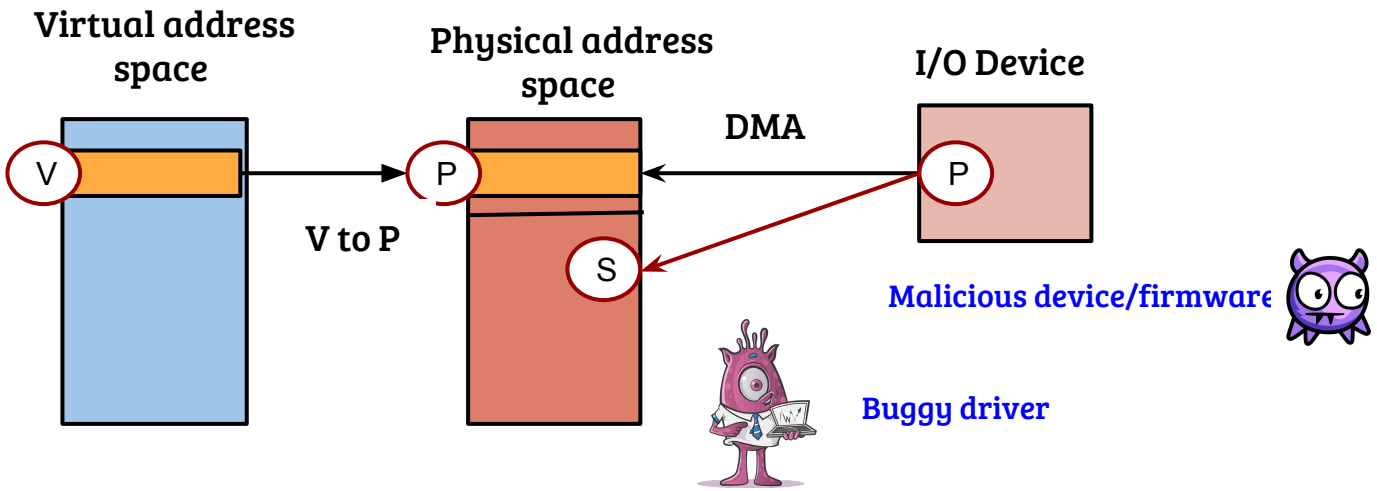
Debadatta Mishra, CSE, IITK

DMA without IOMMU



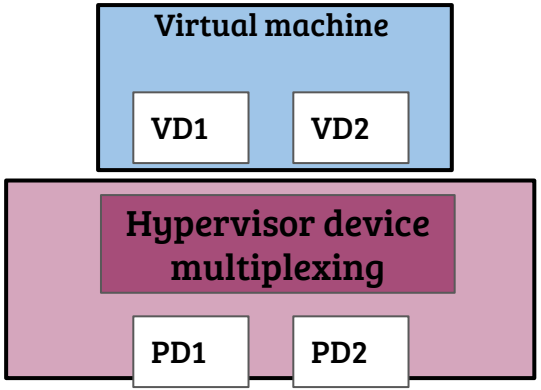
- Device driver allocates buffer (VA = V, PA = P) using OS APIs and configures the DMA address (P) in the device
- Device performs DMA to/from P, CPU access using V
- What are the possible issues?

Scenario-1: Device isolation



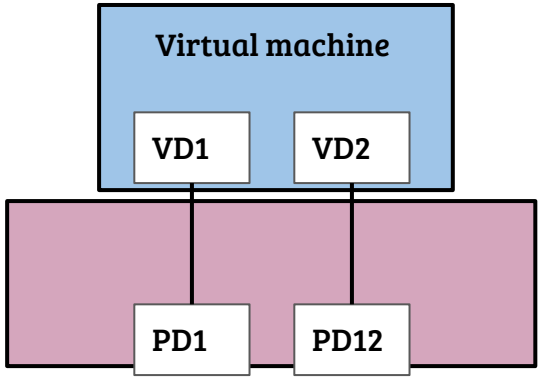
- I/O devices can access arbitrary memory locations
- Compromised security, information disclosure
- Any solutions?

I/O virtualization: background



Software multiplexing

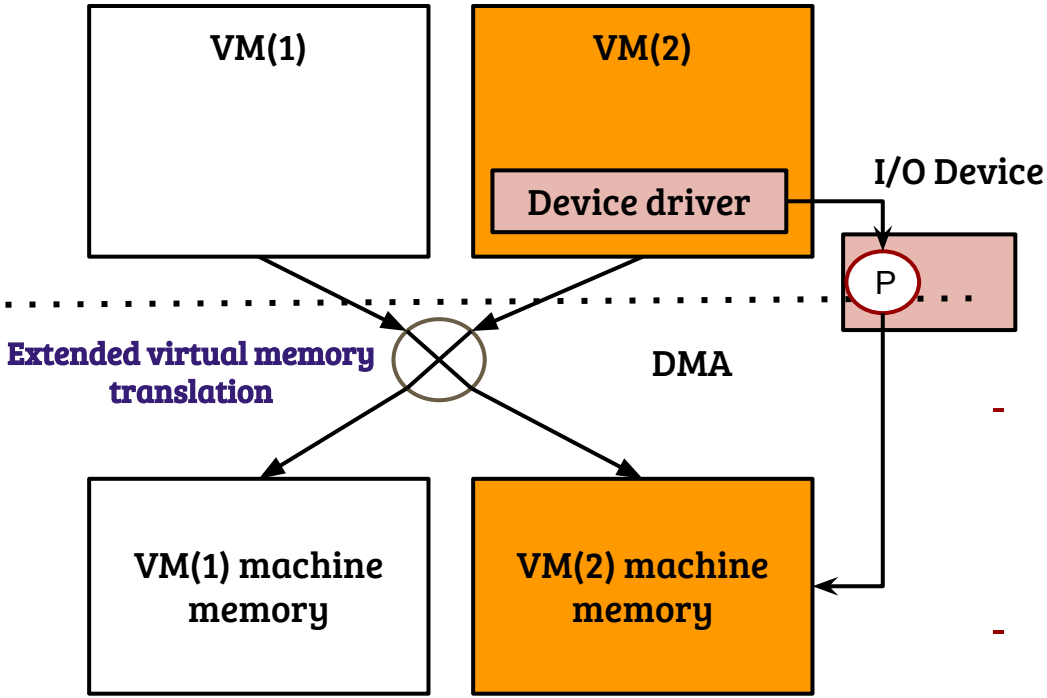
- Hypervisor implements software multiplexing, redirects events to appropriate virtual device
- Device driver is part of the hypervisor
- Example: I/O emulation, paravirtualized I/O



Direct assignment

- Hypervisor assigns devices directly to the VM (exclusive access)
- Device driver is part of the guest OS
- Ex: PCI-passthrough, SRIOV devices

Scenario 2: Direct device access from VMs



- Memory isolation guaranteed by hardware/software techniques (will be discussed latter)
- DMA requires physical address
- What could go wrong?

Summary of DMA isolation

Types of access violation

V1

Bad-address fault (Buggy Device Driver)

V2

Bad-device fault (Device failure/corruption)

V3

Invalid-use fault (Repurpose used pages)

Usage scenarios

R1

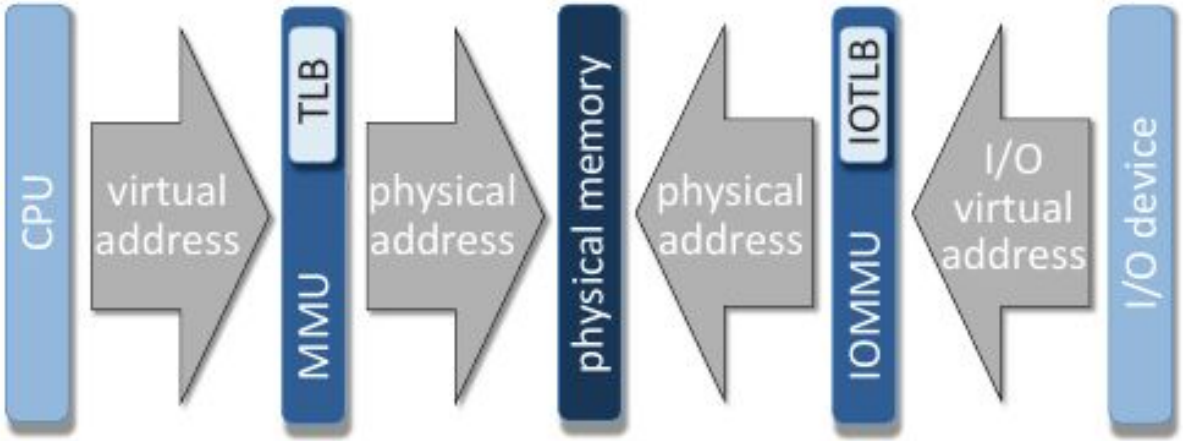
Intra-OS DMA protection (Secure and Robust OS)

R2

Inter-OS DMA protection (Secure and efficient I/O virtualization)

- Requirement: Hardware support which can meet the requirements of native and virtualized systems.

Introduction of I/O virtual address (IOVA) [1]

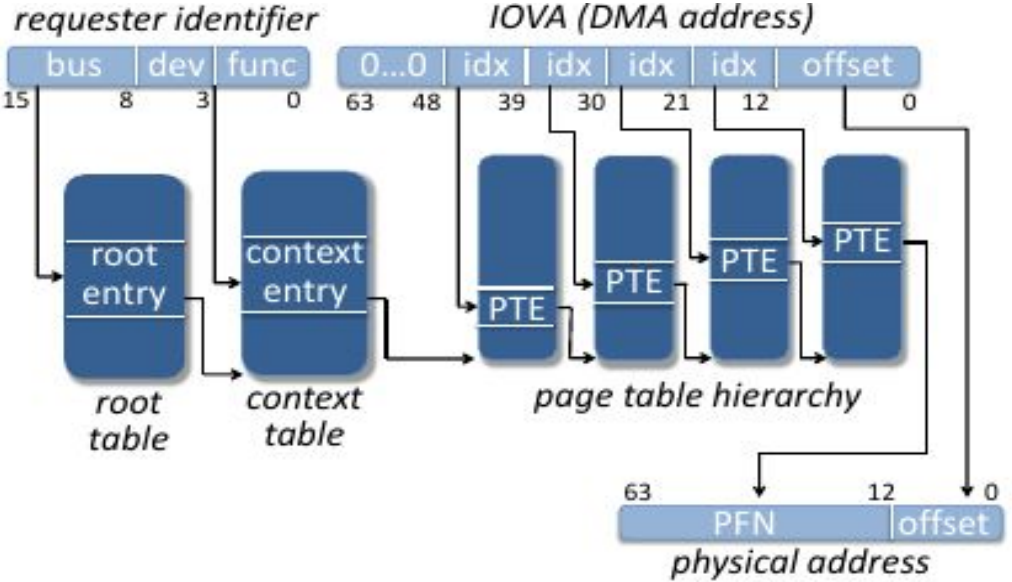


[1]

- In a nutshell, I/O devices are treated like a user process
- The OS associates the physical address with an IOVA and setup the IOVA-to-PA mapping in IOMMU tables
- IOMMU table is similar to page tables (with a TLB!)

1. Malka et al. rIOMMU:Efficient IOMMU for I/O Devices that Employ Ring Buffers https://dl.acm.org/citation.cfm?id=2694355

Overview of IOVA translation using Intel IOMMU



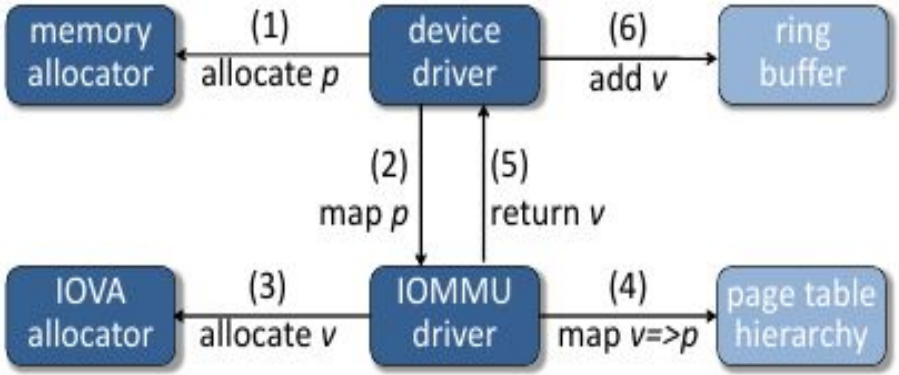
[1]

- PCI identifier (bus-device-function) is used to find the root of page table
- It supports features like huge page etc. (refer [2])
- TLB is similar to MMU TLB
- Page fault support is not there yet ⇒ no lazy allocations

1. Malka et al. rIOMMU:Efficient IOMMU for I/O Devices that Employ Ring Buffers https://dl.acm.org/citation.cfm?id=2694355
2. Intel® Virtualization Technology for Directed I/O https://software.intel.com/sites/default/files/managed/c5/15/vt-directed-io-spec.pdf

Intra-OS protection through IOMMU - DMA map

[1]

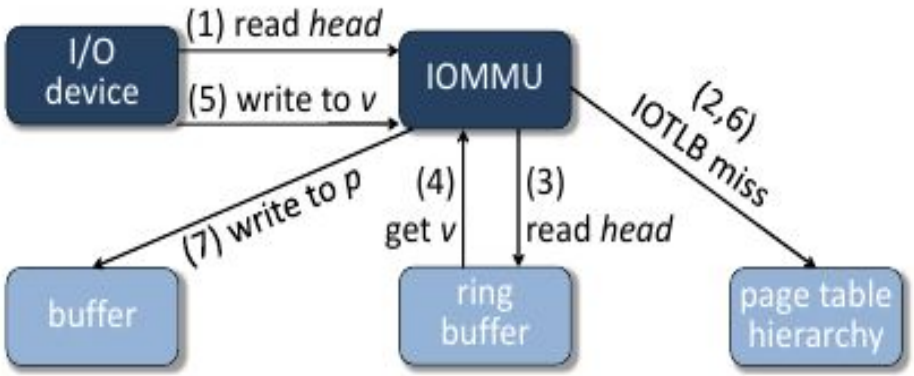


- Scenario: Fillup network receive/send descriptors
- IOVA allocator manages virtual addresses used by I/O devices
- Linux kernel API: dma_map( ), internally performs the above operations

1. Malka et al. rIOMMU:Efficient IOMMU for I/O Devices that Employ Ring Buffers https://dl.acm.org/citation.cfm?id=2694355

Intra-OS protection through IOMMU - DMA receive

[1]

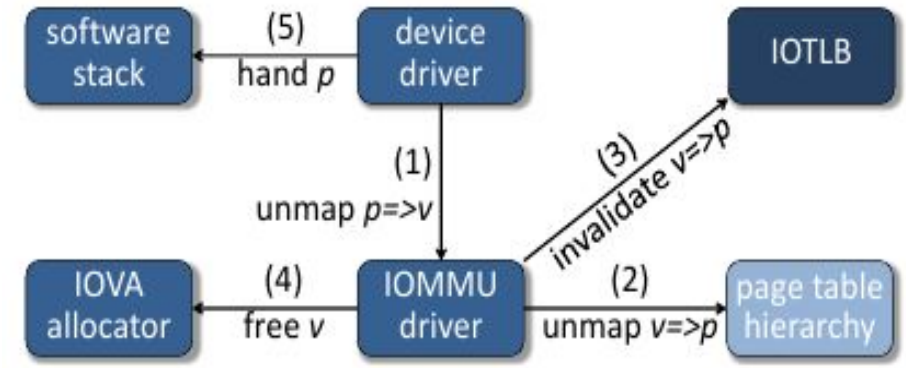


- Scenario: Device receive and perform DMA before raising interrupt
- Require two IOVA ⇒ PA translation: Read descriptor head, Write content to *V*
- IOTLB lookup miss results in IOMMU walk

1. Malka et al. rIOMMU:Efficient IOMMU for I/O Devices that Employ Ring Buffers https://dl.acm.org/citation.cfm?id=2694355

Intra-OS protection through IOMMU - DMA unmap

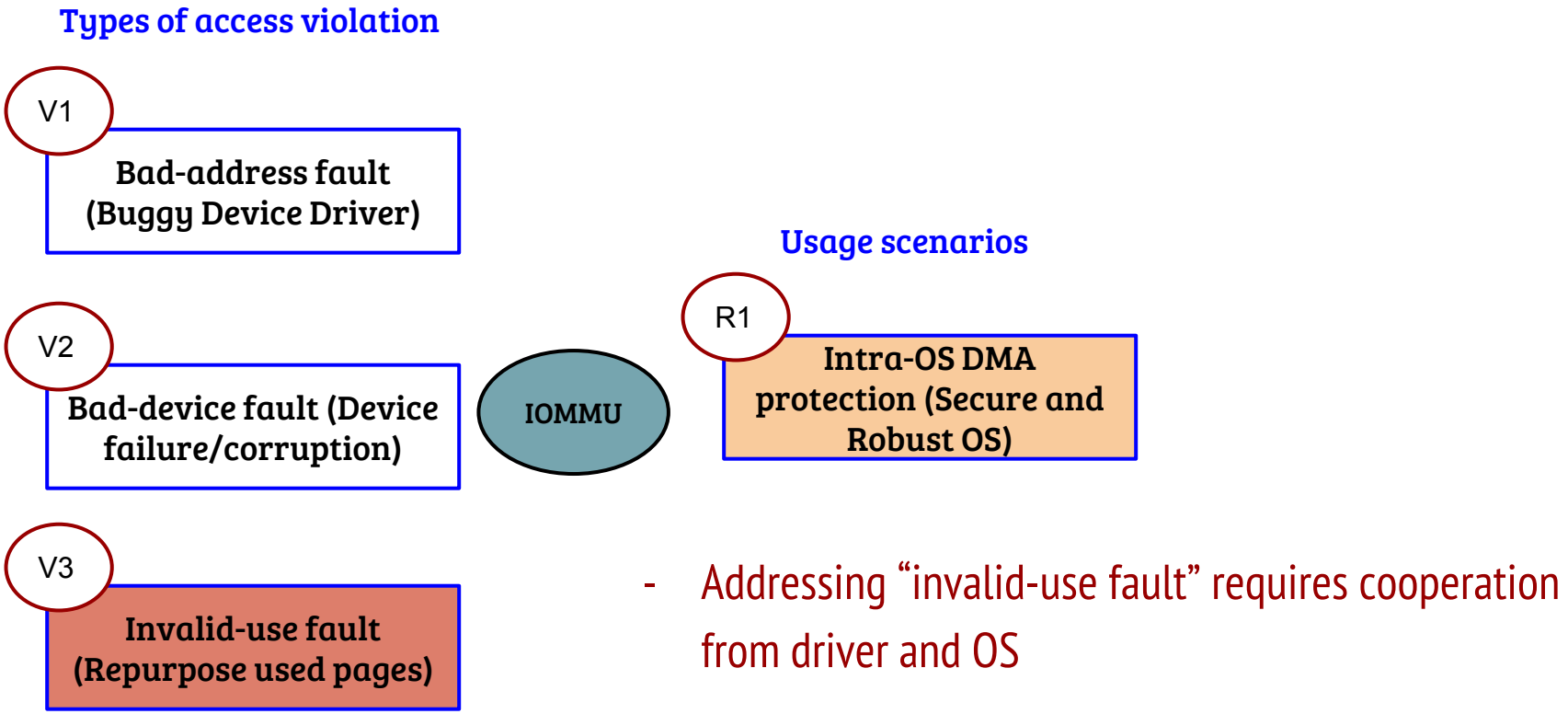
[1]



- Scenario: Interrupt handling by device driver
- Device driver unmaps and frees the IOVA before allowing software processing
- IOTLB invalidation is required
- Linux API: dma_unmap( )

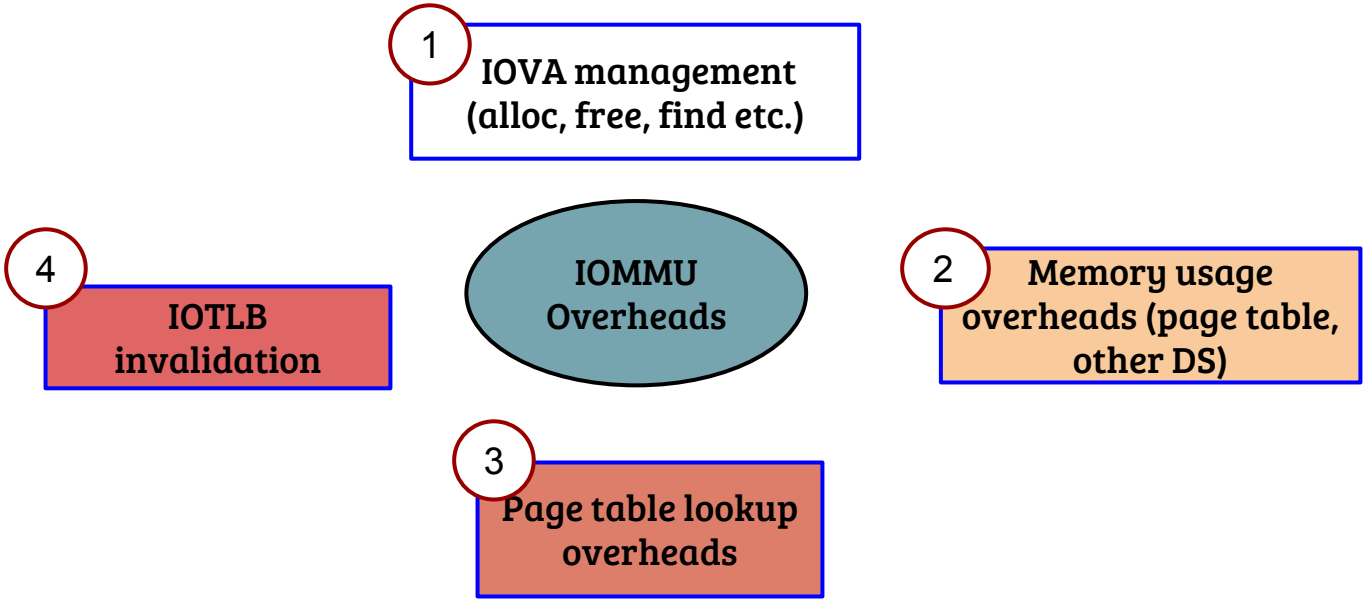
1. Malka et al. rIOMMU:Efficient IOMMU for I/O Devices that Employ Ring Buffers https://dl.acm.org/citation.cfm?id=2694355

Intra-OS DMA isolation using IOMMU



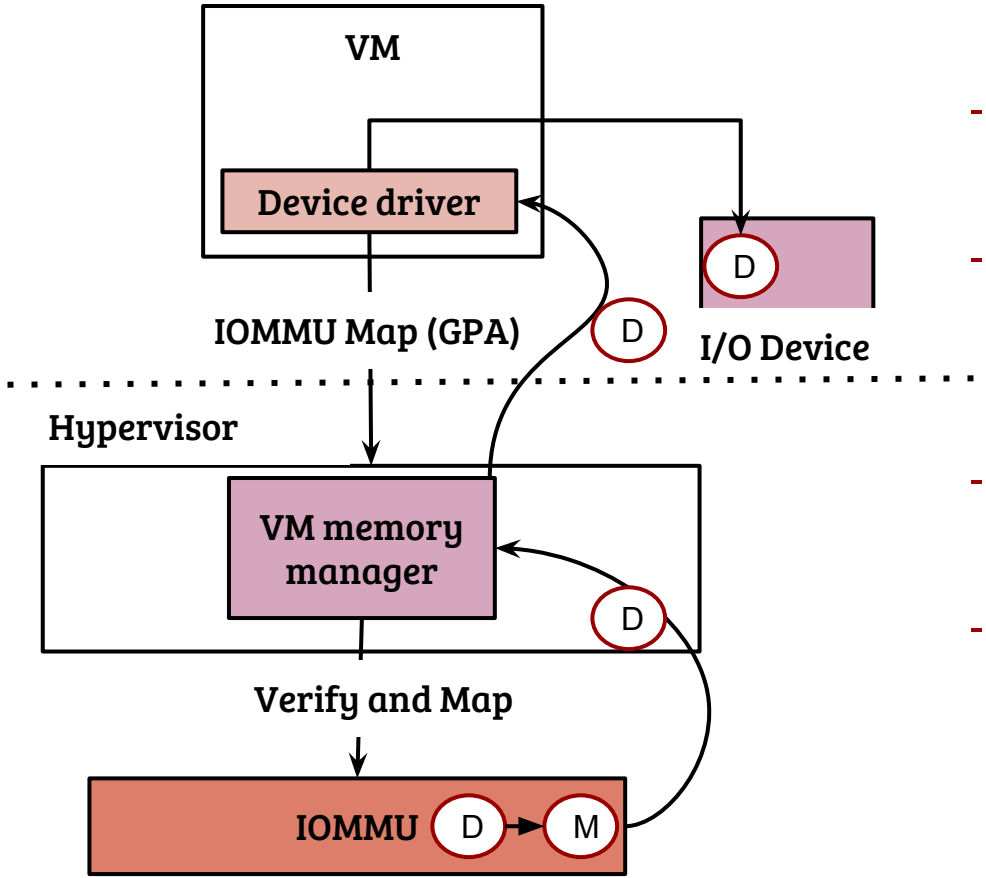
- Addressing "invalid-use fault" requires cooperation from driver and OS

Security comes at a price!



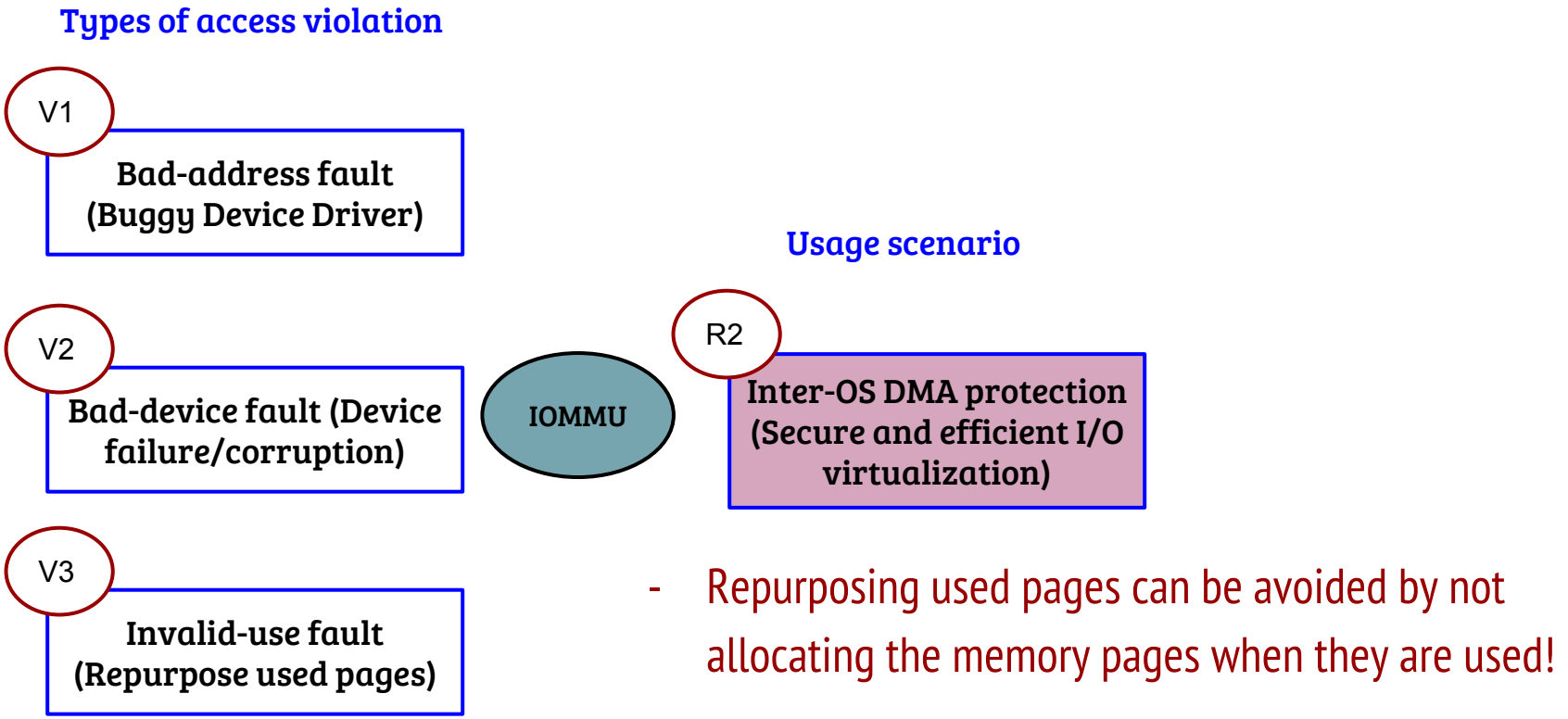
- IOVA mapping must exist only when device is supposed to use (Single-use mapping)
- Cost of single use mapping is non-trivial
- Deferred protection models with compromised security: delayed IOTLB flushing

Direct device access from VMs using IOMMU



- Guest OS requests IOMMU mapping with guest physical address (GPA)
- Hypervisor validates the ownership (finds GPA ⇒ M) and performs the map and returns the DMA address (D)
- Device driver in guest OS configures the device with DMA address
- Device uses the DMA descriptor like a native system

Inter-OS DMA isolation using IOMMU



- Repurposing used pages can be avoided by not allocating the memory pages when they are used!