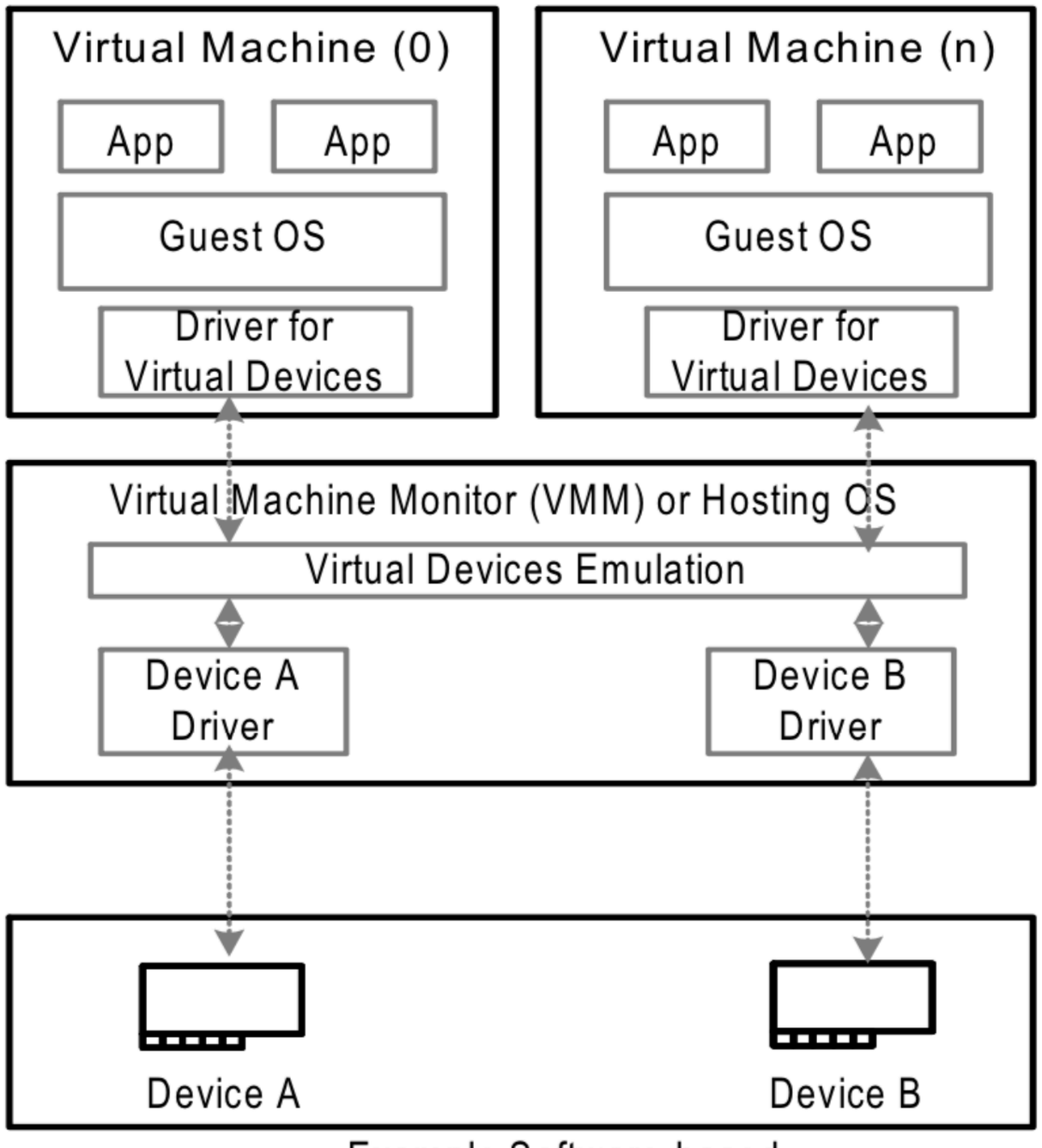
设备透传内幕

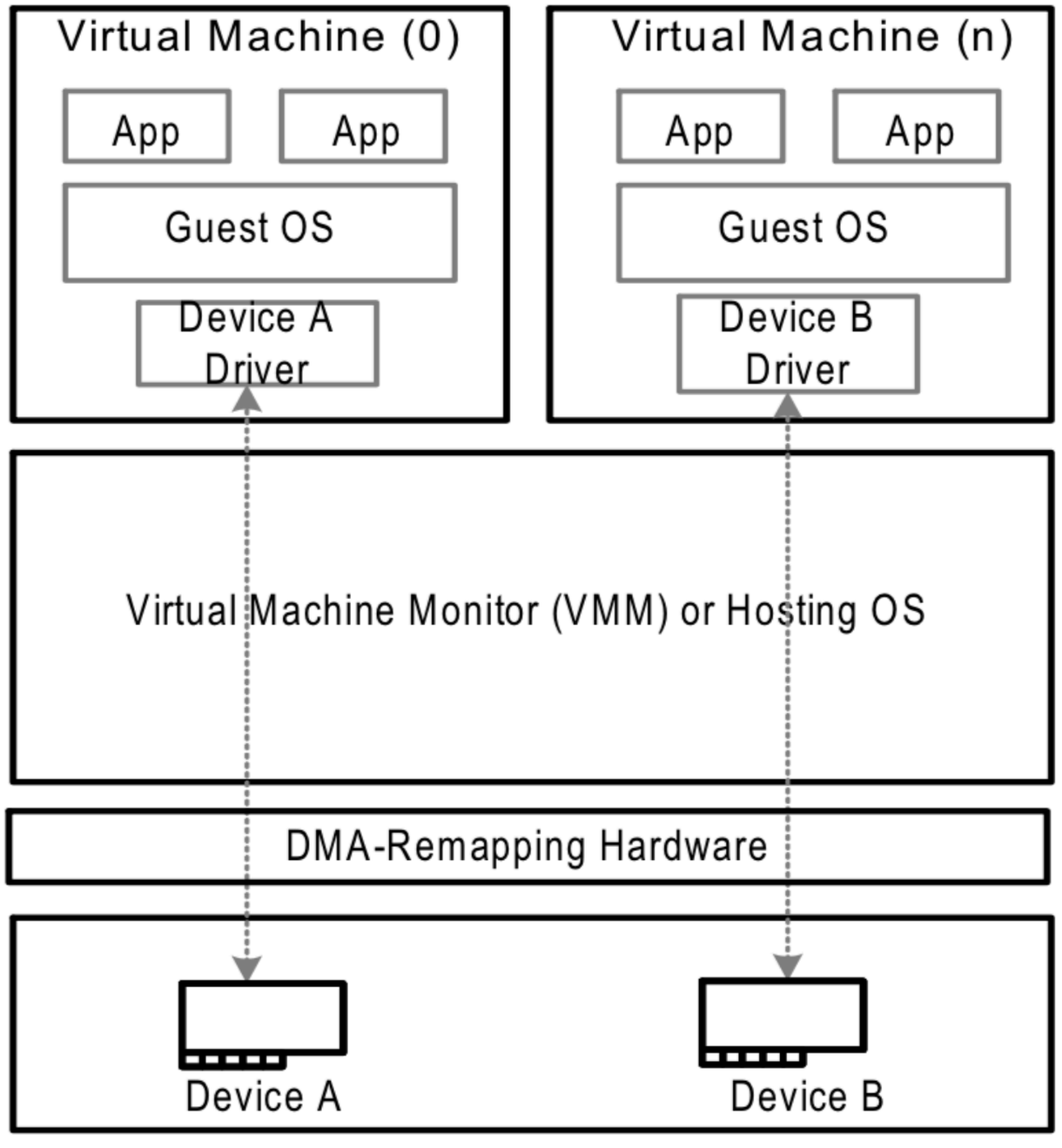
Agenda

1. VFIO
2. Memory Region(BAR)
3. DMA
4. Interrupt

what is passthrough



Example Software-based I/O Virtualization

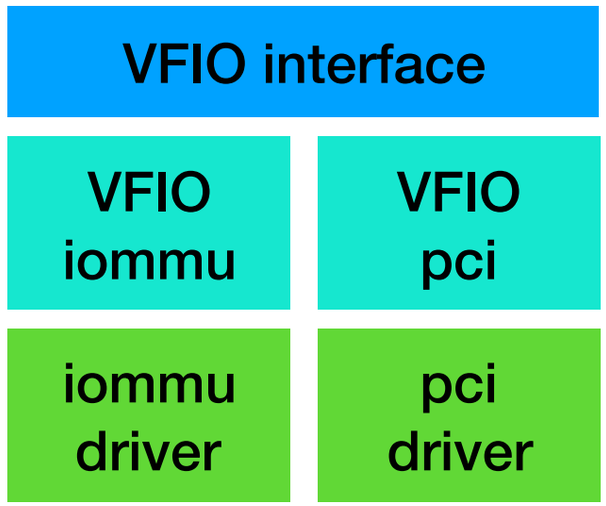


Direct Assignment of I/O Devices

VFIO

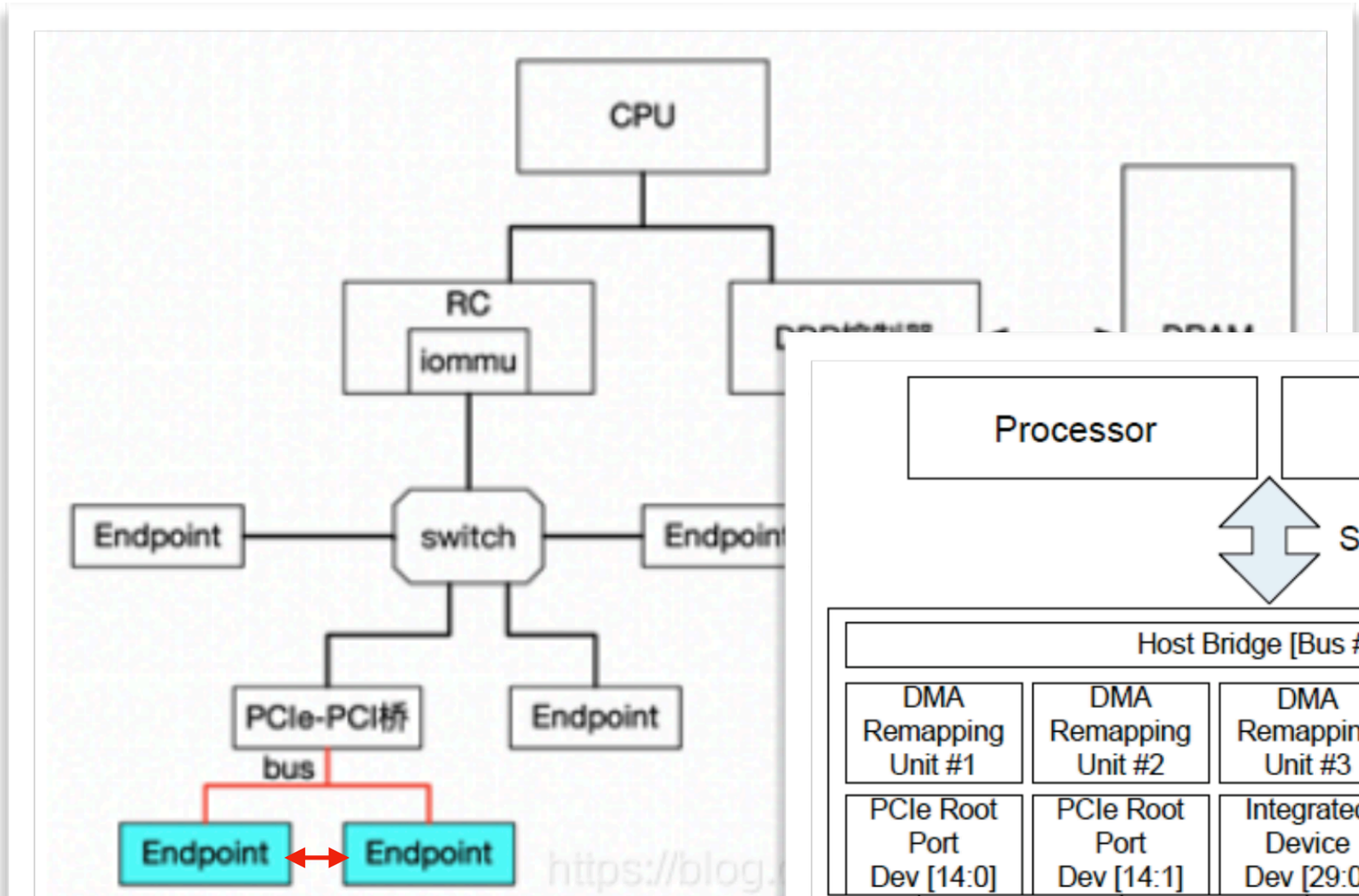
VFIO

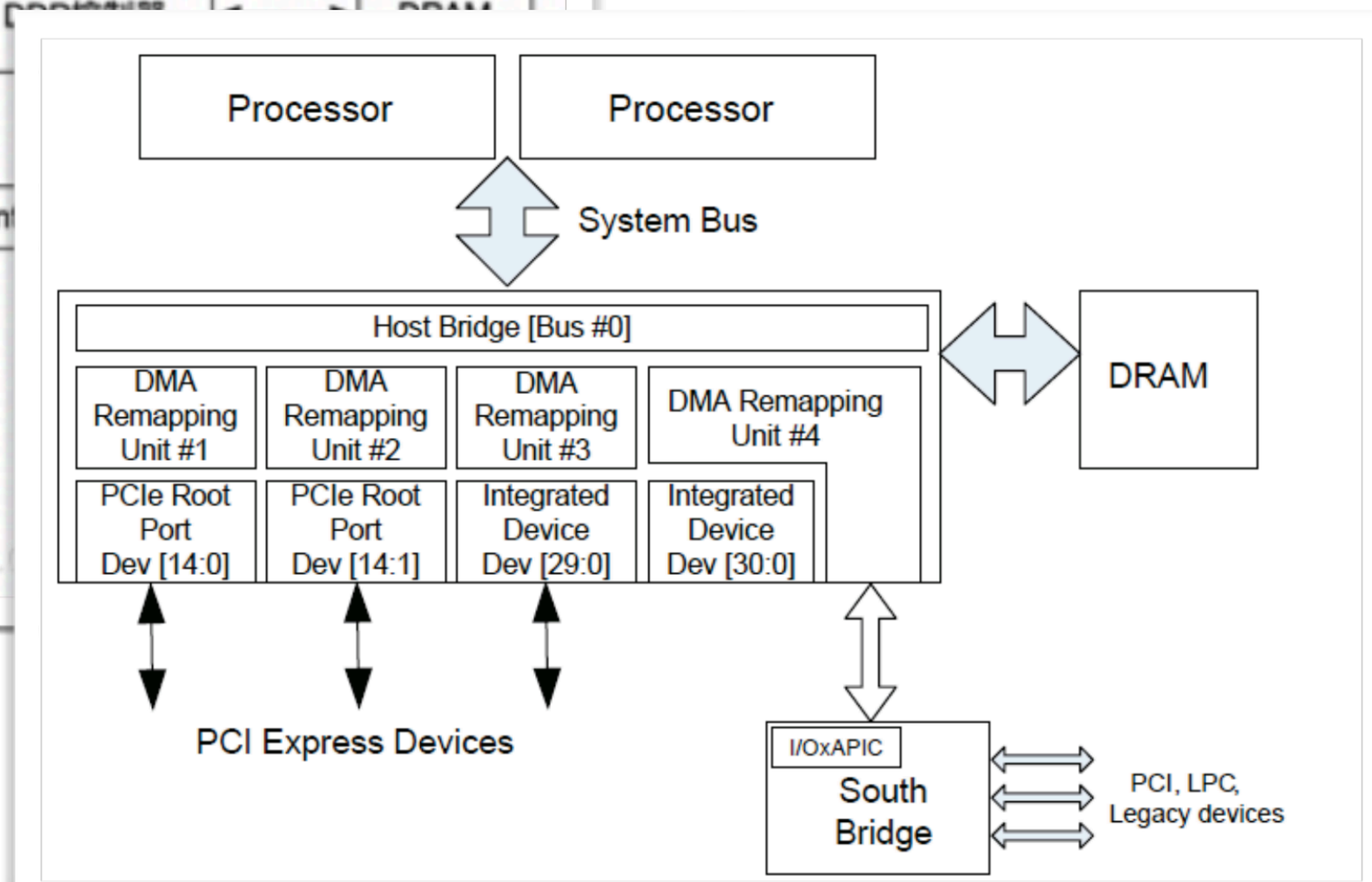
vfio is a secure, userspace, driver framework, it's a driver



VFIO

container: think it as a VM machine.
domain: abstraction of a DMR remapping unit
group: the minimal isolation target, configured by iommu driver.



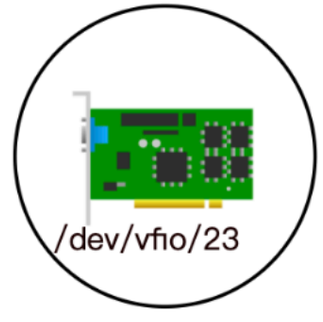


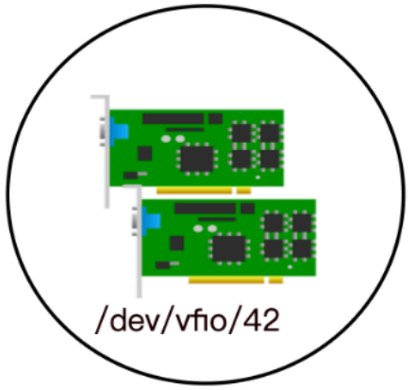
Enable IOMMU

GRUB_CMDLINE_LINUX_DEFAULT="… intel_iommu=on…"

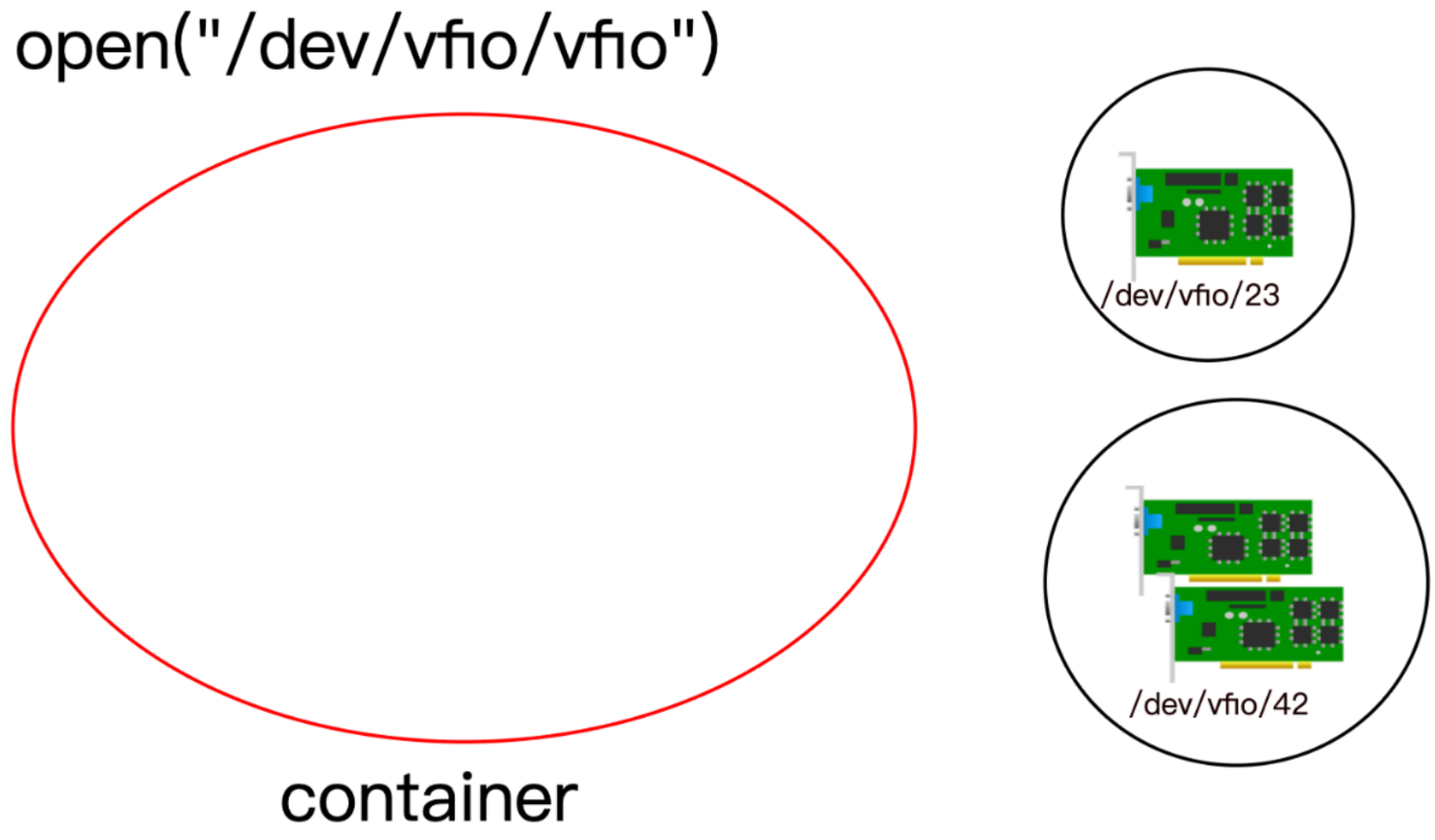
```
root@xulin:/home/xulin# dmesg | grep -i iommu
[    0.000000] Command line: BOOT_IMAGE=/boot/vmlinuz-5.0.0-38-generic r
andoff=1
[    0.098718] Kernel command line: BOOT_IMAGE=/boot/vmlinuz-5.0.0-38-ge
on vt.handoff=1
[    0.098788] DMAR: IOMMU enabled
[    0.149105] DMAR-IR: IOAPIC id 2 under DRHD base  0xfed91000 IOMMU 1
[    0.870998] iommu: Adding device 0000:00:00.0 to group 0
[    0.871008] iommu: Adding device 0000:00:01.0 to group 1
[    0.871015] iommu: Adding device 0000:00:02.0 to group 2
[    0.871021] iommu: Adding device 0000:00:08.0 to group 3
[    0.871030] iommu: Adding device 0000:00:12.0 to group 4
[    0.871041] iommu: Adding device 0000:00:14.0 to group 5
[    0.871047] iommu: Adding device 0000:00:14.2 to group 5
[    0.871056] iommu: Adding device 0000:00:16.0 to group 6
[    0.871062] iommu: Adding device 0000:00:17.0 to group 7
[    0.871077] iommu: Adding device 0000:00:1c.0 to group 8
[    0.871088] iommu: Adding device 0000:00:1c.5 to group 9
[    0.871097] iommu: Adding device 0000:00:1c.6 to group 10
[    0.871113] iommu: Adding device 0000:00:1f.0 to group 11
[    0.871120] iommu: Adding device 0000:00:1f.3 to group 11
[    0.871126] iommu: Adding device 0000:00:1f.4 to group 11
[    0.871133] iommu: Adding device 0000:00:1f.5 to group 11
[    0.871136] iommu: Adding device 0000:01:00.0 to group 1
[    0.871151] iommu: Adding device 0000:02:00.0 to group 12
[    0.871160] iommu: Adding device 0000:03:00.0 to group 13
[    0.871170] iommu: Adding device 0000:05:00.0 to group 14
```

VFIO

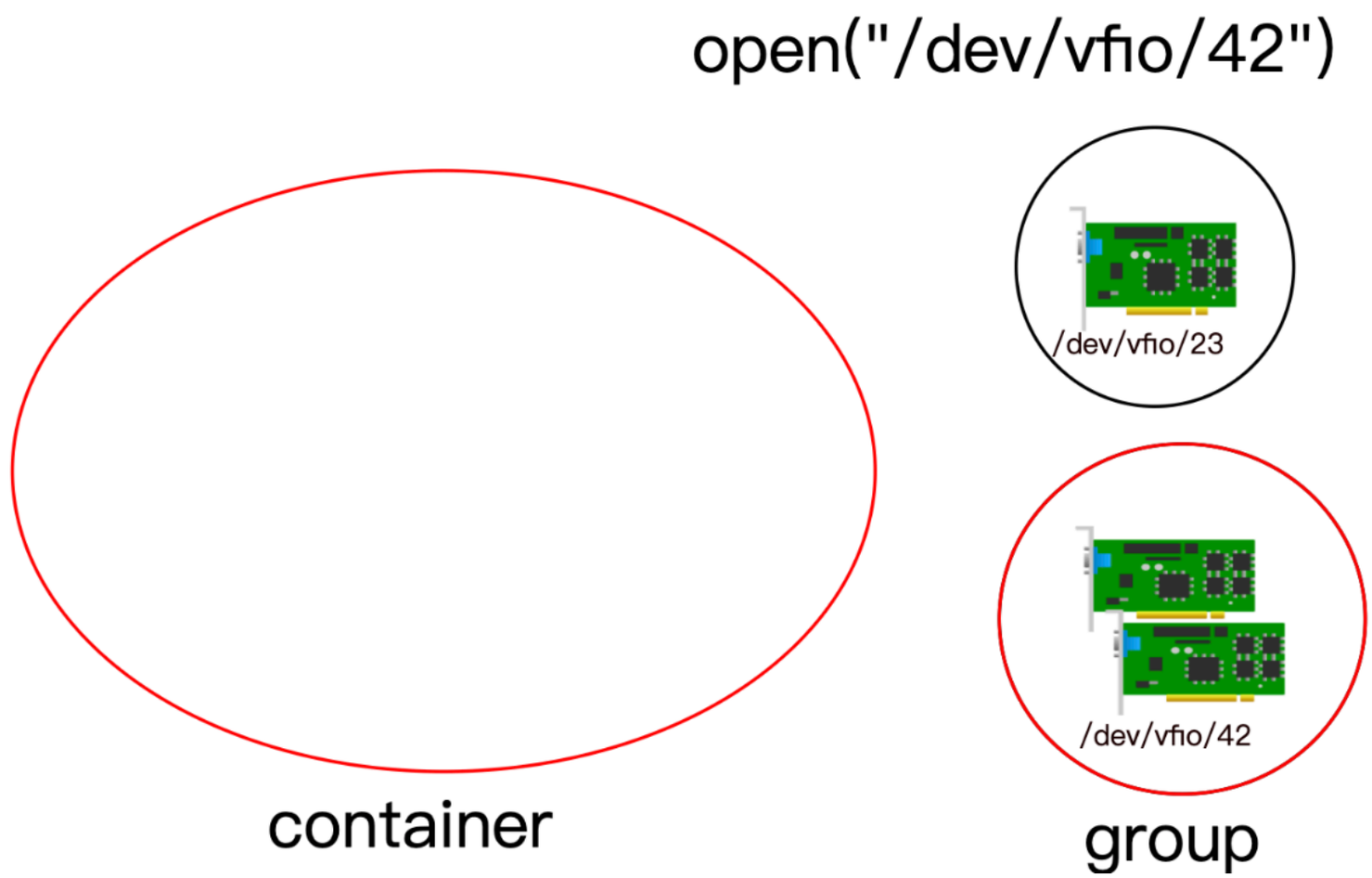




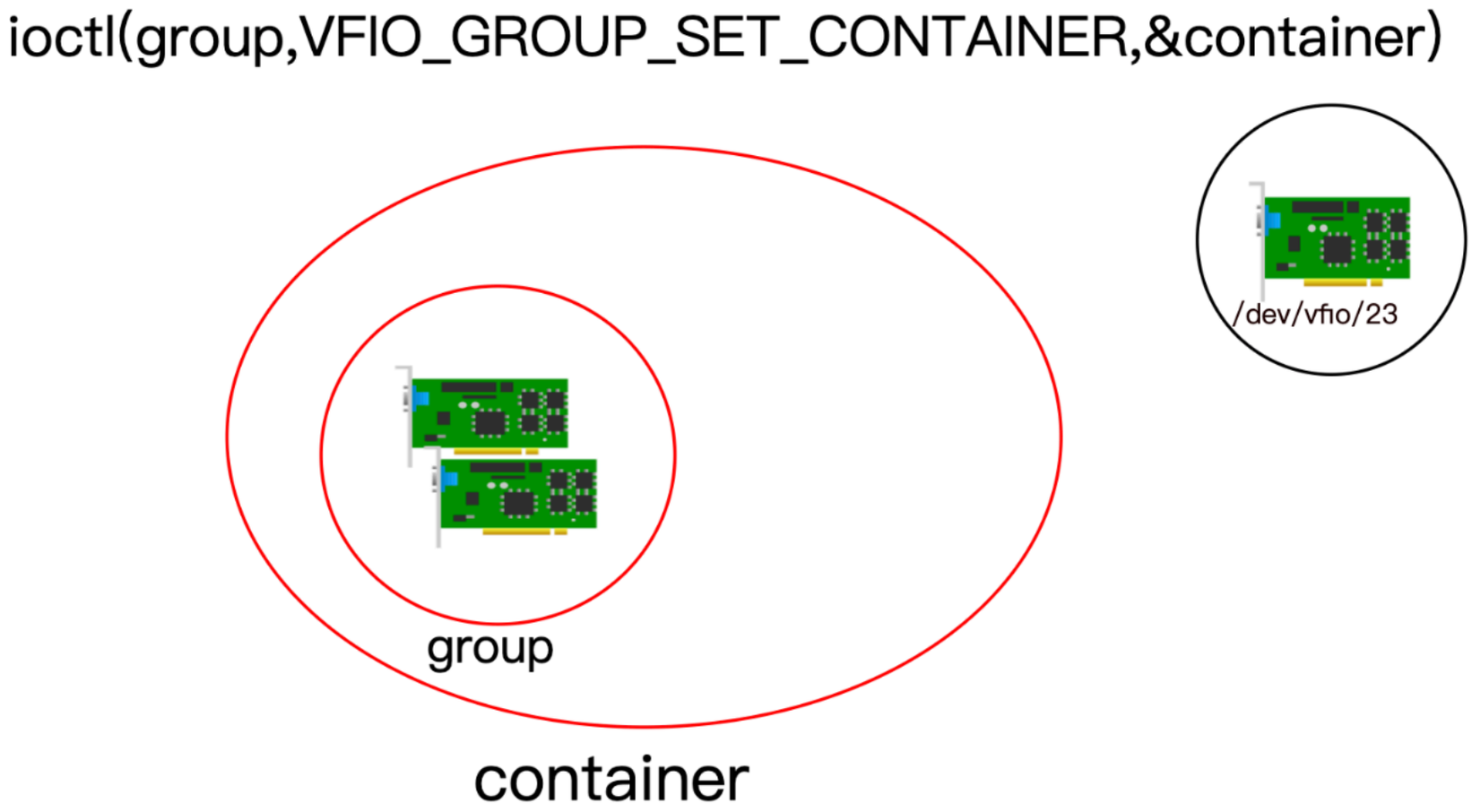
VFIO



VFIO

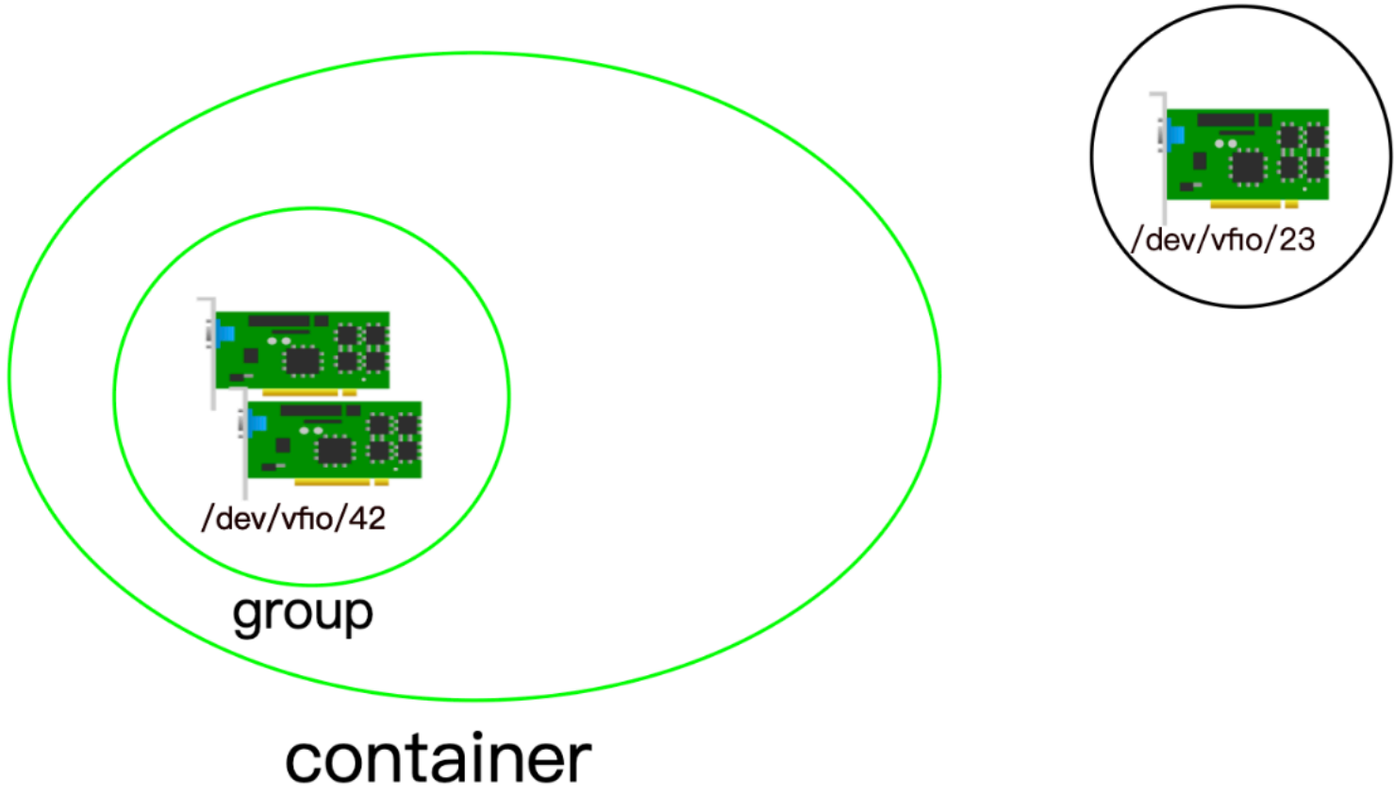


VFIO

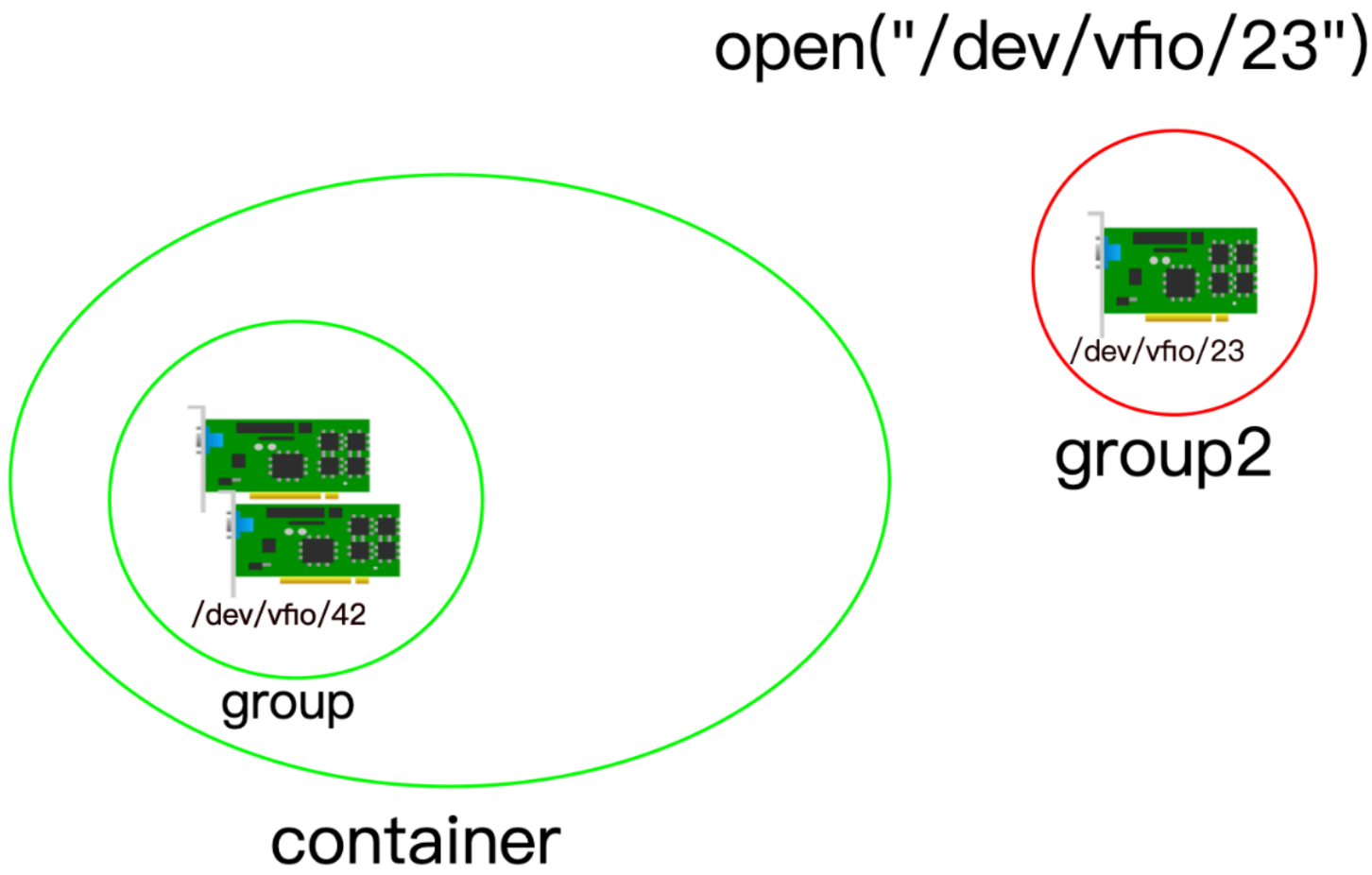

ioctl(group,VFIO_GROUP_SET_CONTAINER,&container)
/dev/vfio/23
group
container

VFIO

ioctl(container,VFIO_SET_IOMMU,VFIO_TYPE1_IOMMU)

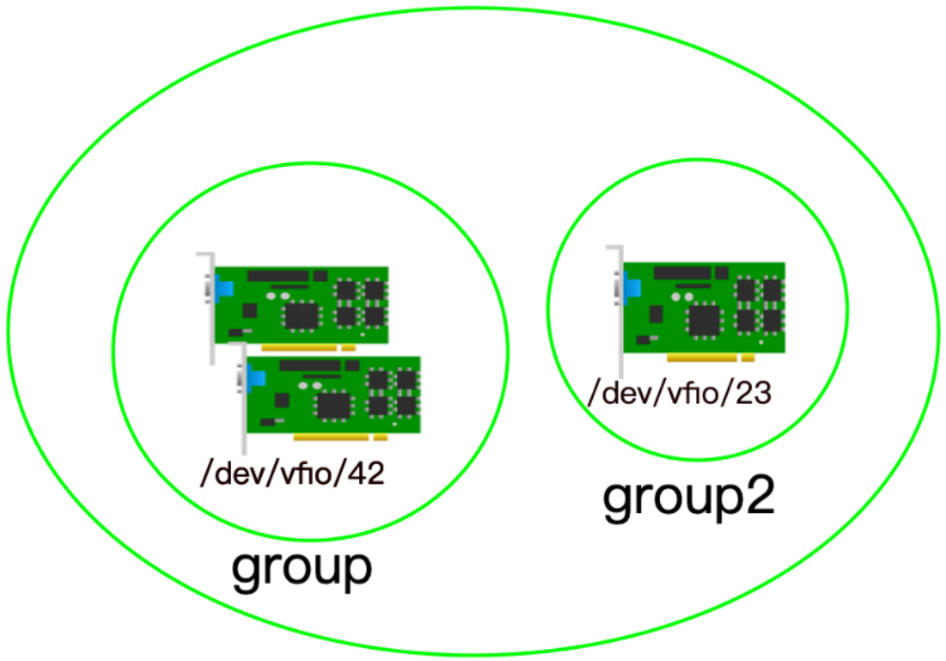


VFIO


open("/dev/vfio/23")
/dev/vfio/23
group2
/dev/vfio/42
group
container

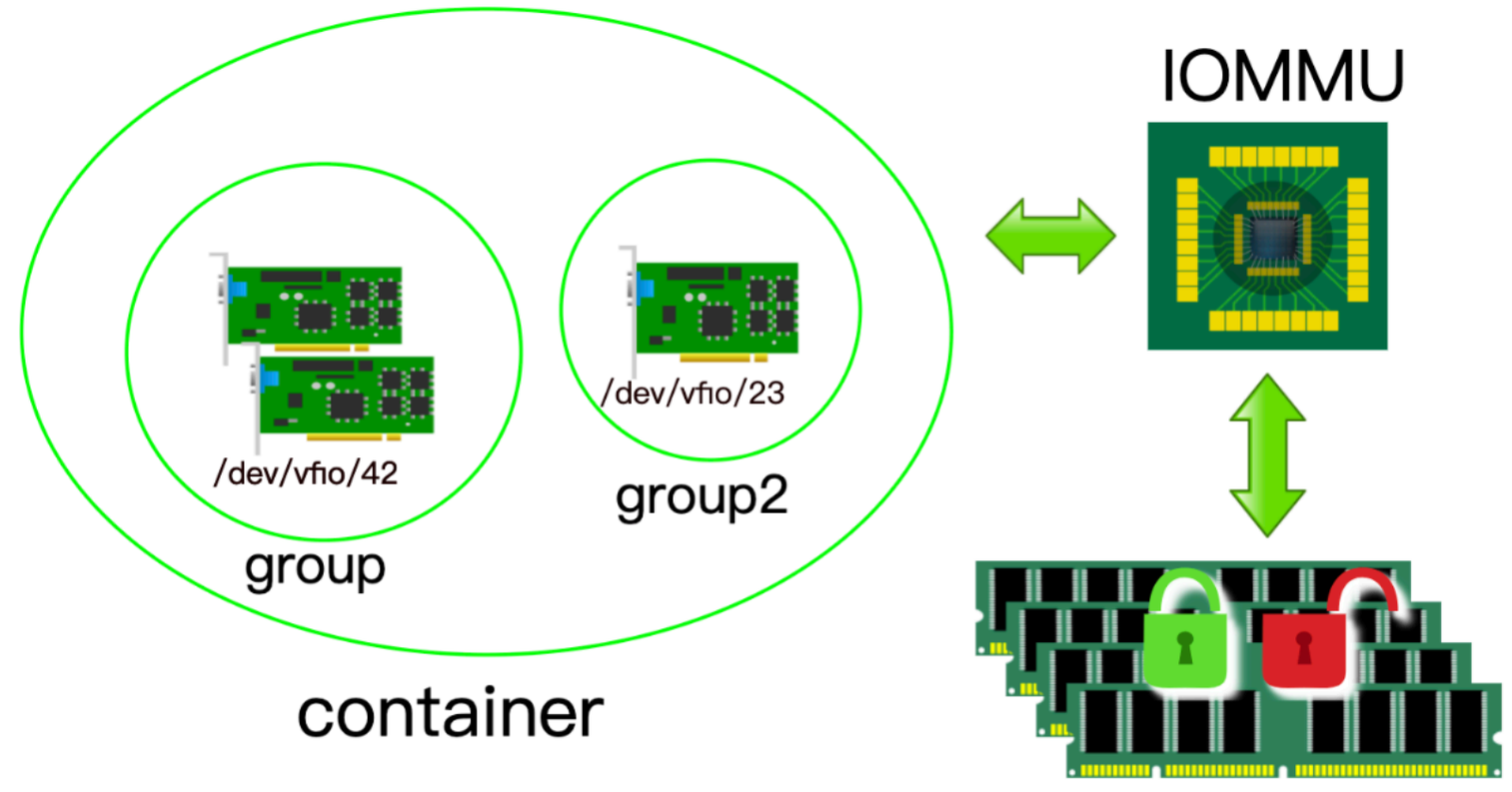
VFIO

ioctl(group2,VFIO_GROUP_SET_CONTAINER,&container)



VFIO

ioctl(container,VFIO_IOMMU_MAP_DMA,&map)
ioctl(container,VFIO_IOMMU_UNMAP_DMA,&unmap)

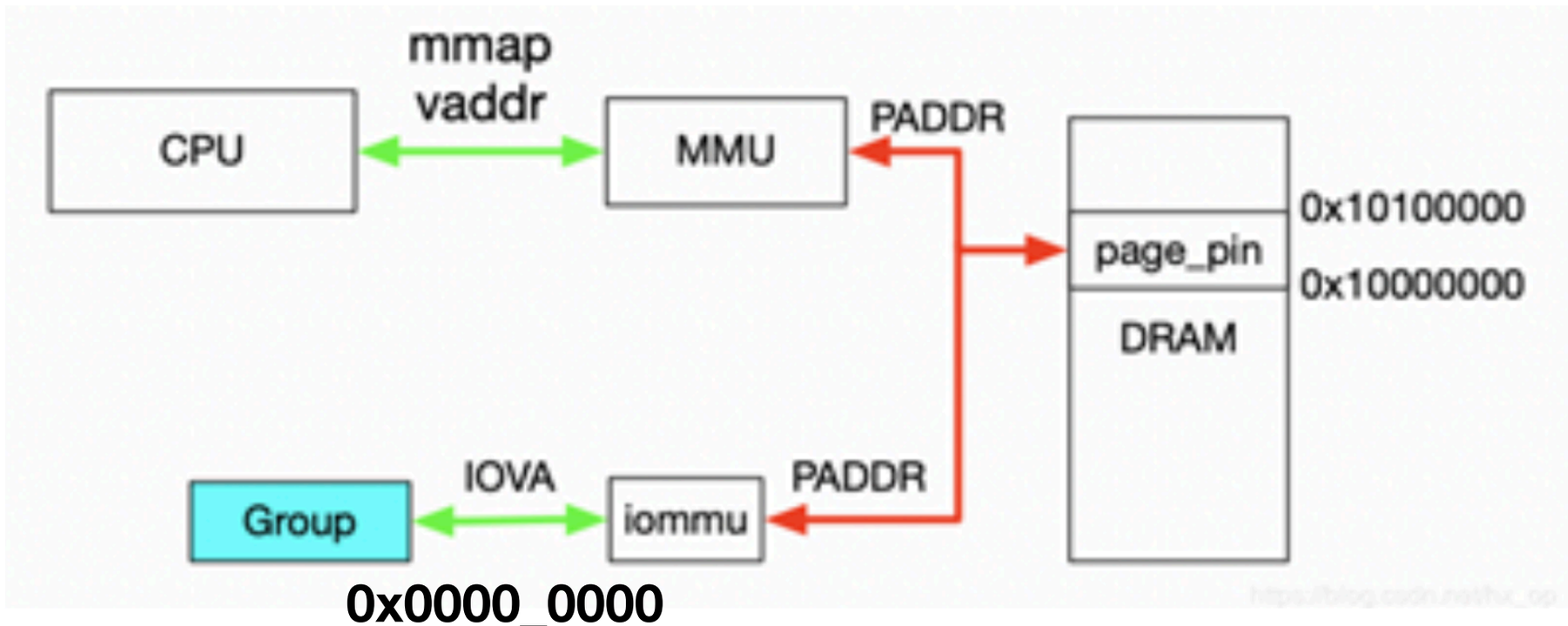


```
int container, group, device, i;
struct vfio_group_status group_status =
                 { .argsz = sizeof(group_status) };
struct vfio_iommu_type1_info iommu_info = { .argsz = sizeof(iommu_info) };
struct vfio_iommu_type1_dma_map dma_map = { .argsz = sizeof(dma_map) };
struct vfio_device_info device_info = { .argsz = sizeof(device_info) };

/* Create a new container */
container = open("/dev/vfio/vfio", O_RDWR);

if (ioctl(container, VFIO_GET_API_VERSION) != VFIO_API_VERSION)
    /* Unknown API version */

if (!ioctl(container, VFIO_CHECK_EXTENSION, VFIO_TYPE1_IOMMU))
    /* Doesn't support the IOMMU driver we want. */

/* Open the group */
group = open("/dev/vfio/26", O_RDWR);

/* Test the group is viable and available */
ioctl(group, VFIO_GROUP_GET_STATUS, &group_status);

if (!(group_status.flags & VFIO_GROUP_FLAGS_VIABLE))
    /* Group is not viable (ie, not all devices bound for vfio) */

/* Add the group to the container */
ioctl(group, VFIO_GROUP_SET_CONTAINER, &container);

/* Enable the IOMMU model we want */
ioctl(container, VFIO_SET_IOMMU, VFIO_TYPE1_IOMMU);

/* Get addition IOMMU info */
ioctl(container, VFIO_IOMMU_GET_INFO, &iommu_info);
```

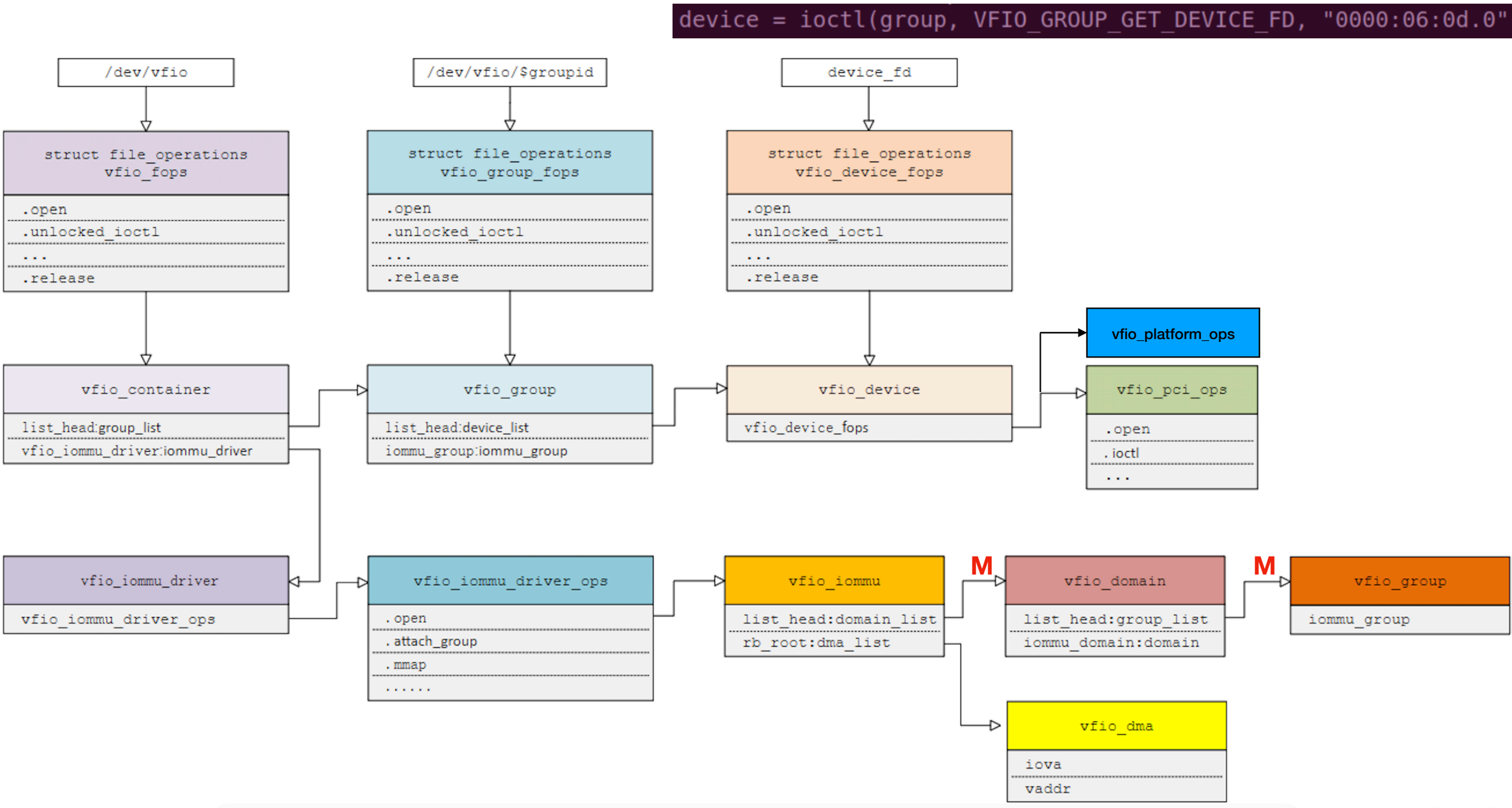
```
/* Allocate some space and setup a DMA mapping */
dma_map.vaddr = mmap(0, 1024 * 1024, PROT_READ | PROT_WRITE,
                     MAP_PRIVATE | MAP_ANONYMOUS, 0, 0);
dma_map.size = 1024 * 1024;
dma_map.iova = 0; /* 1MB starting at 0x0 from device view */
dma_map.flags = VFIO_DMA_MAP_FLAG_READ | VFIO_DMA_MAP_FLAG_WRITE;

ioctl(container, VFIO_IOMMU_MAP_DMA, &dma_map);

/* Get a file descriptor for the device */
device = ioctl(group, VFIO_GROUP_GET_DEVICE_FD, "0000:06:0d.0");

/* Test and setup the device */
ioctl(device, VFIO_DEVICE_GET_INFO, &device_info);

for (i = 0; i < device_info.num_regions; i++) {
    struct vfio_region_info reg = { .argsz = sizeof(reg) };

    reg.index = i;

    ioctl(device, VFIO_DEVICE_GET_REGION_INFO, &reg);

    /* Setup mappings... read/write offsets, mmaps
     * For PCI devices, config space is a region */
}

for (i = 0; i < device_info.num_irqs; i++) {
    struct vfio_irq_info irq = { .argsz = sizeof(irq) };

    irq.index = i;

    ioctl(device, VFIO_DEVICE_GET_IRQ_INFO, &irq);

    /* Setup IRQs... eventfds, VFIO_DEVICE_SET_IRQS */
}

/* Gratuitous device reset and go... */
ioctl(device, VFIO_DEVICE_RESET);
```

VFIO

IOVA page fault is hard to implement, so the memory is pinned



VFIO


device = ioctl(group, VFIO_GROUP_GET_DEVICE_FD, "0000:06:0d.0")
/dev/vfio
/dev/vfio/$groupid
device_fd
struct file_operations vfio_fops
.open
.unlocked_ioctl
...
.release
struct file_operations vfio_group_fops
.open
.unlocked_ioctl
...
.release
struct file_operations vfio_device_fops
.open
.unlocked_ioctl
...
.release
vfio_container
list_head:group_list
vfio_iommu_driver:iommu_driver
vfio_group
list_head:device_list
iommu_group:iommu_group
vfio_device
vfio_device_fops
vfio_platform_ops
vfio_pci_ops
.open
.ioctl
...
vfio_iommu_driver
vfio_iommu_driver_ops
vfio_iommu_driver_ops
.open
.attach_group
.mmap
.....
vfio_iommu
list_head:domain_list
rb_root:dma_list
M
vfio_domain
list_head:group_list
iommu_domain:domain
M
vfio_group
iommu_group
vfio_dma
iova
vaddr

let’s see a real example

VFIO example

load vfio driver

modprobe vfio
modprobe vfio-pci

prepare vfio device

```
xulin@xulin:~/smbshare$  lspci | grep -i xilinx
01:00.0 Serial controller: Xilinx Corporation Device 7022
```

```
xulin@xulin:~$ lspci -s 0000:01:00.0 -n
01:00.0 0700: 10ee:7022
```

```
root@xulin:/home/xulin# sudo echo "10ee 7022" > /sys/bus/pci/drivers/vfio-pci/new_id
```

QEMU parameter

-device vfio-pci,host=0000:01:00.0

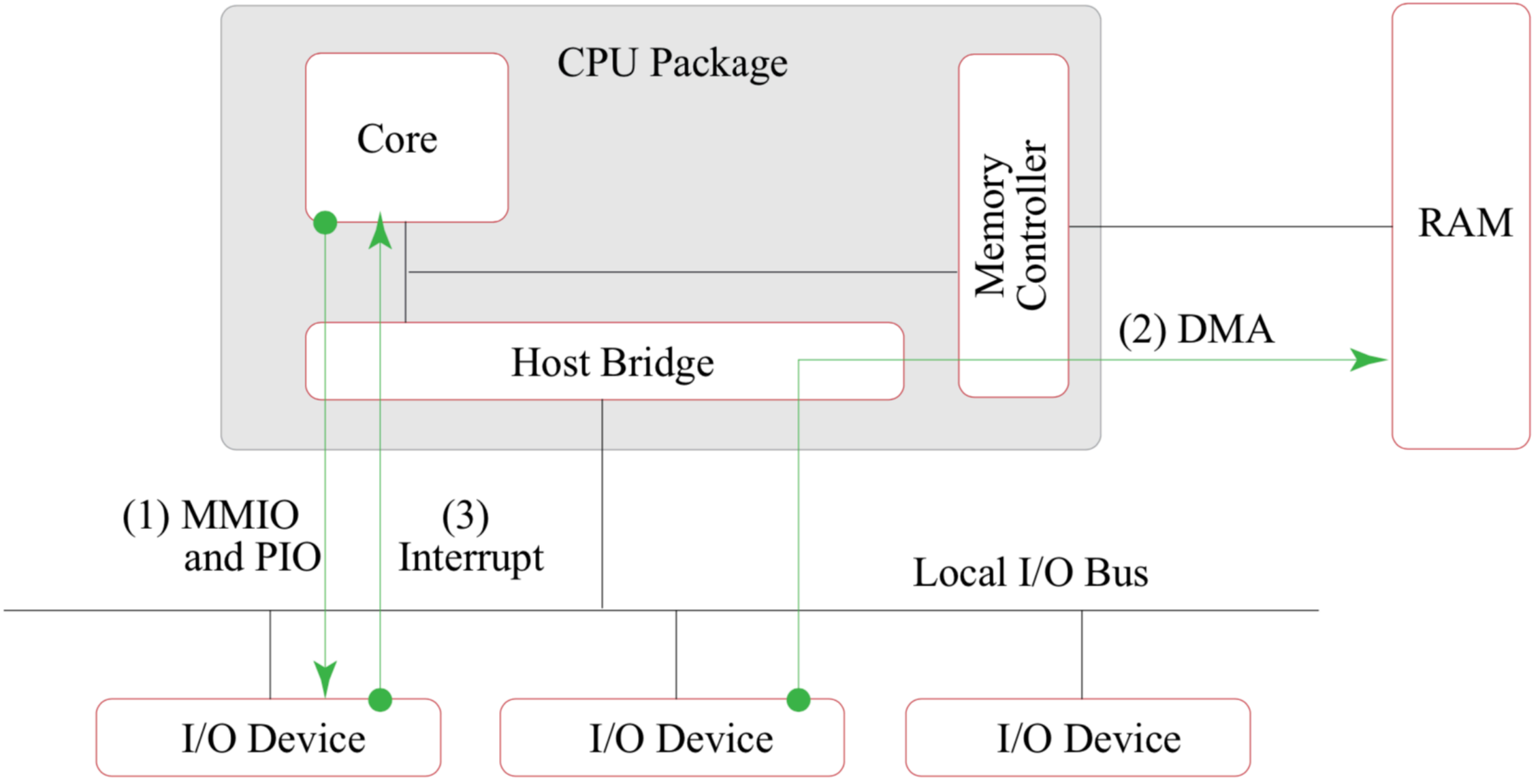
VFIO example

```
test@test-Standard-PC-i440FX-PIIX-1996:~/dma_ip_drivers-master/XDMA/linux-kern
el/tests$  lspci | grep -i xilinx
00:04.0 Serial controller: Xilinx Corporation Device 7022
```

test@test-Standard-PC-i440FX-PIIX-1996: ~/dma_ip_drivers-master/XDMA/linux-kernel/tests

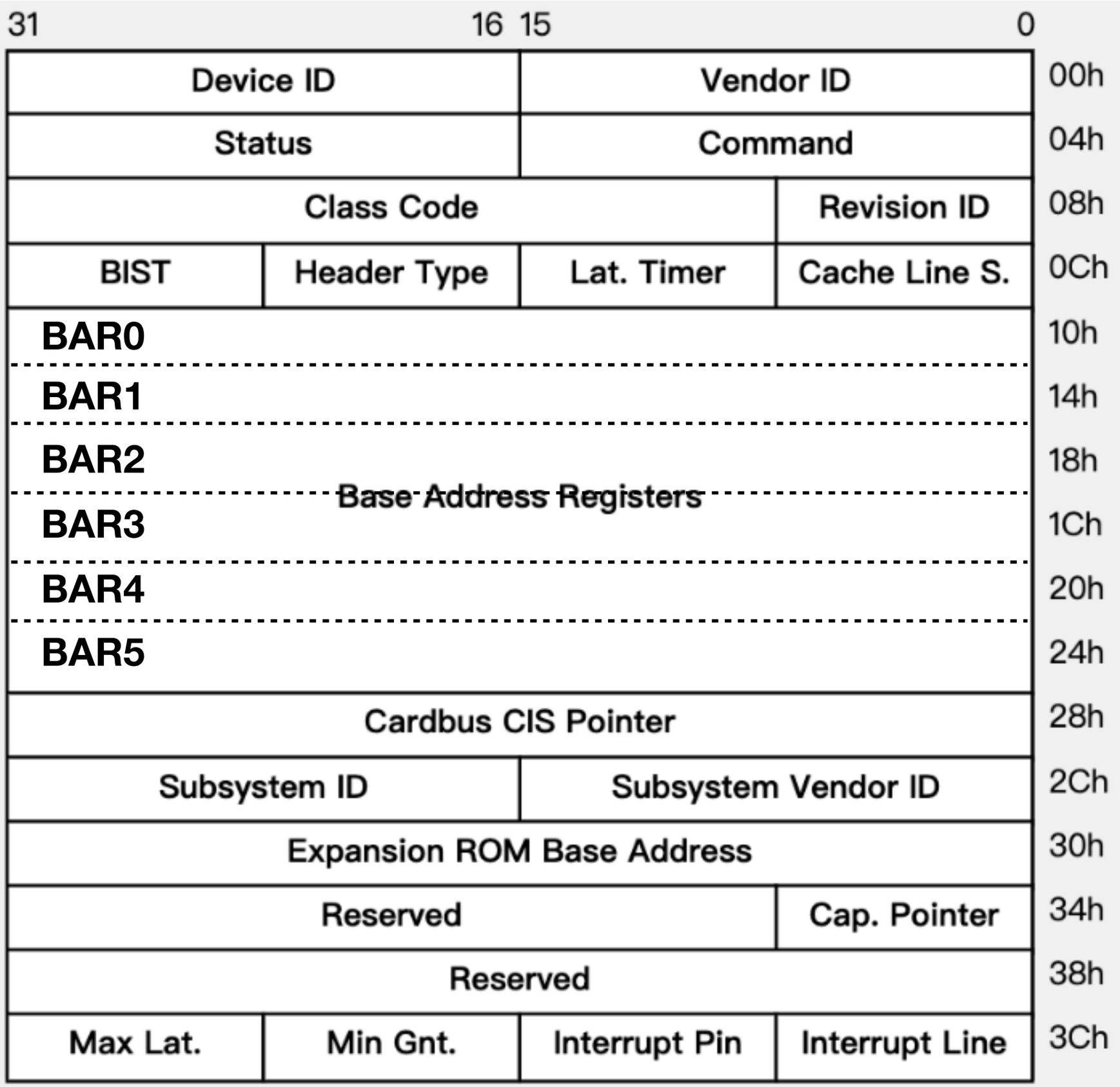
```
Error: No PCIe DMA channels were identified.
test@test-Standard-PC-i440FX-PIIX-1996:~/dma_ip_drivers-master/XDMA/linux-kern
el/tests$ sudo ./run_test.sh
Info: Number of enabled h2c channels = 1
Info: Number of enabled c2h channels = 1
Info: The PCIe DMA core is memory mapped.
Info: Running PCIe DMA memory mapped write read test
      transfer size:  1024
      transfer count: 1
Info: Writing to h2c channel 0 at address offset 0.
Info: Wait for current transactions to complete.
/dev/xdma0_h2c_0 ** Average BW = 1024, 4.024461
Info: Writing to h2c channel 0 at address offset 1024.
Info: Wait for current transactions to complete.
/dev/xdma0_h2c_0 ** Average BW = 1024, 1.229481
Info: Writing to h2c channel 0 at address offset 2048.
Info: Wait for current transactions to complete.
/dev/xdma0_h2c_0 ** Average BW = 1024, 0.827526
Info: Writing to h2c channel 0 at address offset 3072.
Info: Wait for current transactions to complete.
/dev/xdma0_h2c_0 ** Average BW = 1024, 1.867107
Info: Reading from c2h channel 0 at address offset 0.
Info: Wait for the current transactions to complete.
/dev/xdma0_c2h_0 ** Average BW = 1024, 1.575625
Info: Reading from c2h channel 0 at address offset 1024.
Info: Wait for the current transactions to complete.
/dev/xdma0_c2h_0 ** Average BW = 1024, 9.107161
```

Interact with PCI device

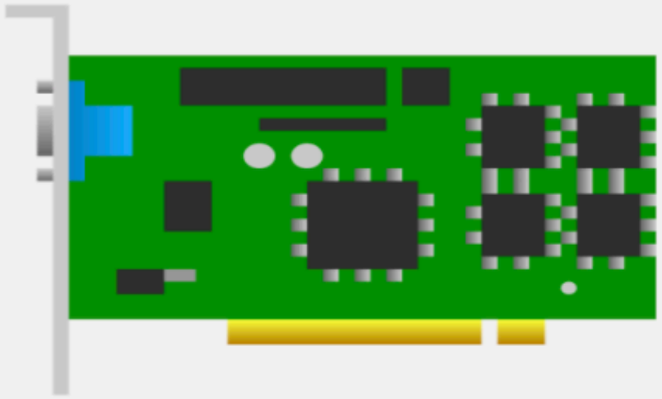


Memory Region

PCI Device Configuration Space



PCI Device Example

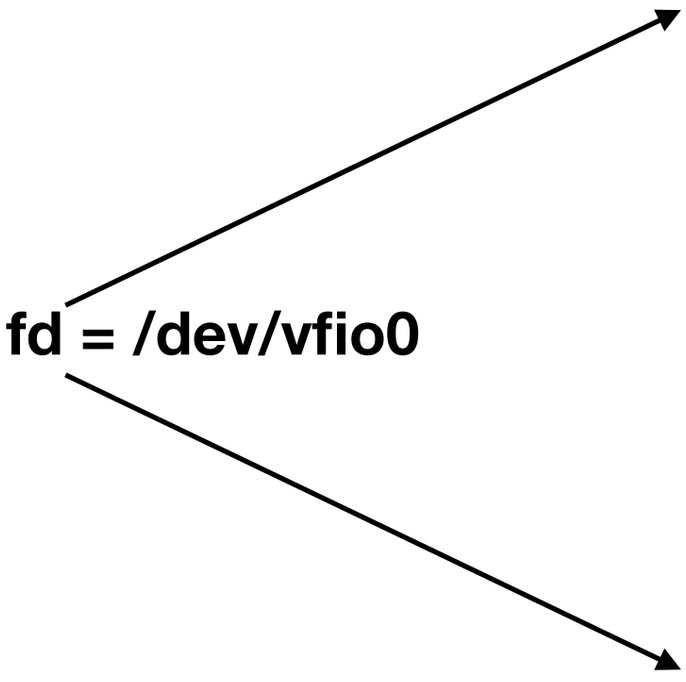


```
01:00.0 VGA compatible controller: NVIDIA Corporation GM107
 Region 0: Memory at f6000000 (32-bit, non-prefetchable) [size=16M]
 Region 1: Memory at e0000000 (64-bit, prefetchable) [size=256M]
 Region 3: Memory at f0000000 (64-bit, prefetchable) [size=32M]
 Region 5: I/O ports at e000 [size=128]
 Expansion ROM at f7000000 [disabled] [size=512K]
```

These are all regions

PCI Device Resource

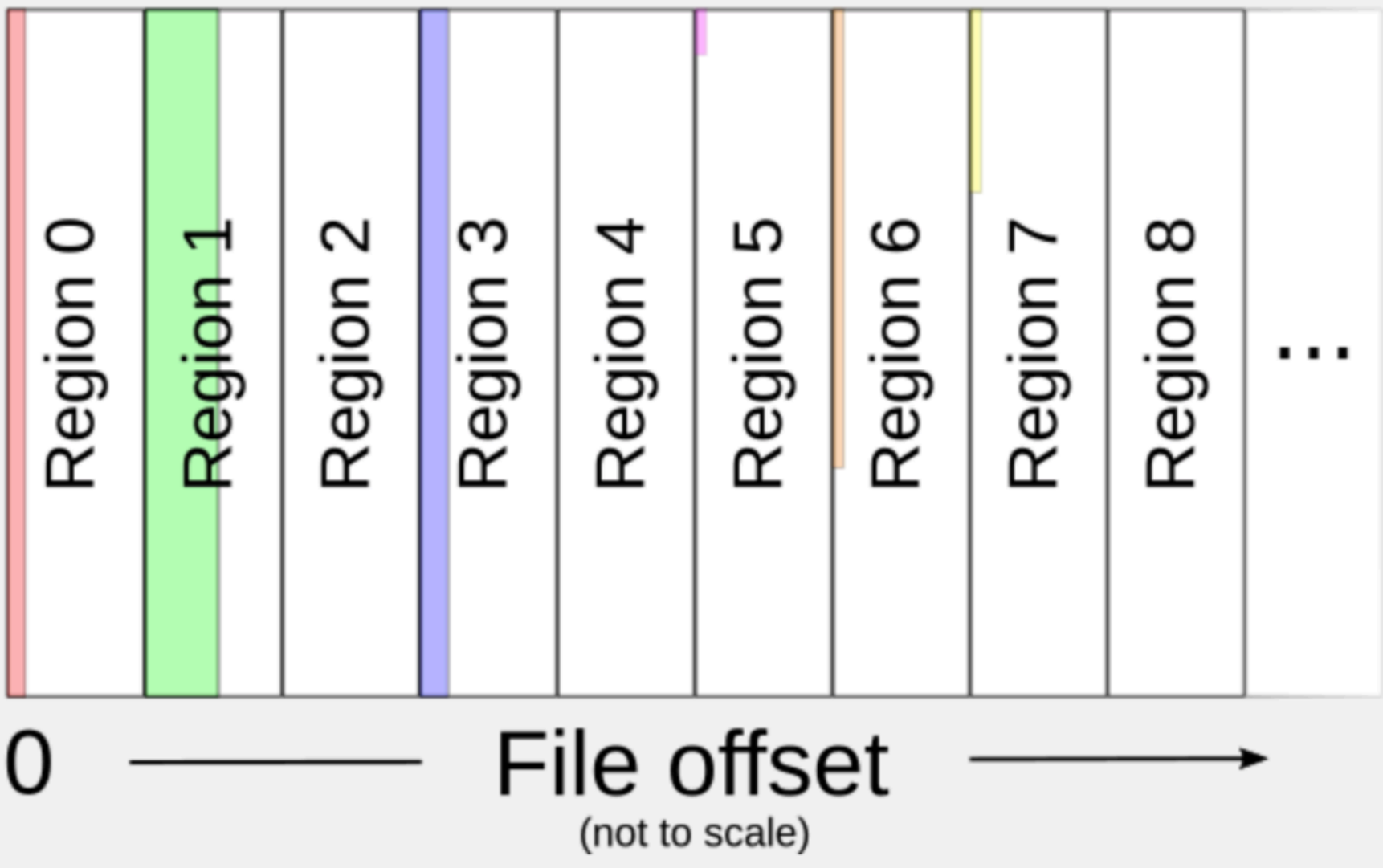
- vfio device file descriptor is divided into regions
- each region maps to device resource, MMIO/IO bar, PCI config space



Region 0	0<<40	BAR 0 offset
Region 1	1<<40	BAR 1 offset
Region 2	2<<40	BAR 2 offset
Region 3	3<<40	BAR 3 offset
Region 4	4<<40	BAR 4 offset
Region 5	5<<40	BAR 5 offset
Region 6	6<<40	ROM offset
Region 7	7<<40	PCI Config offset
Region 8	8<<40	PCI VGA offset

Regions map to device file offset

```
01:00.0 VGA compatible controller: NVIDIA Corporation GM107
 Region 0: Memory at f6000000 (32-bit, non-prefetchable) [size=16M]
 Region 1: Memory at e0000000 (64-bit, prefetchable) [size=256M]
 Region 3: Memory at f0000000 (64-bit, prefetchable) [size=32M]
 Region 5: I/O ports at e000 [size=128]
 Expansion ROM at f7000000 [disabled] [size=512K]
```



MMIO/PIO

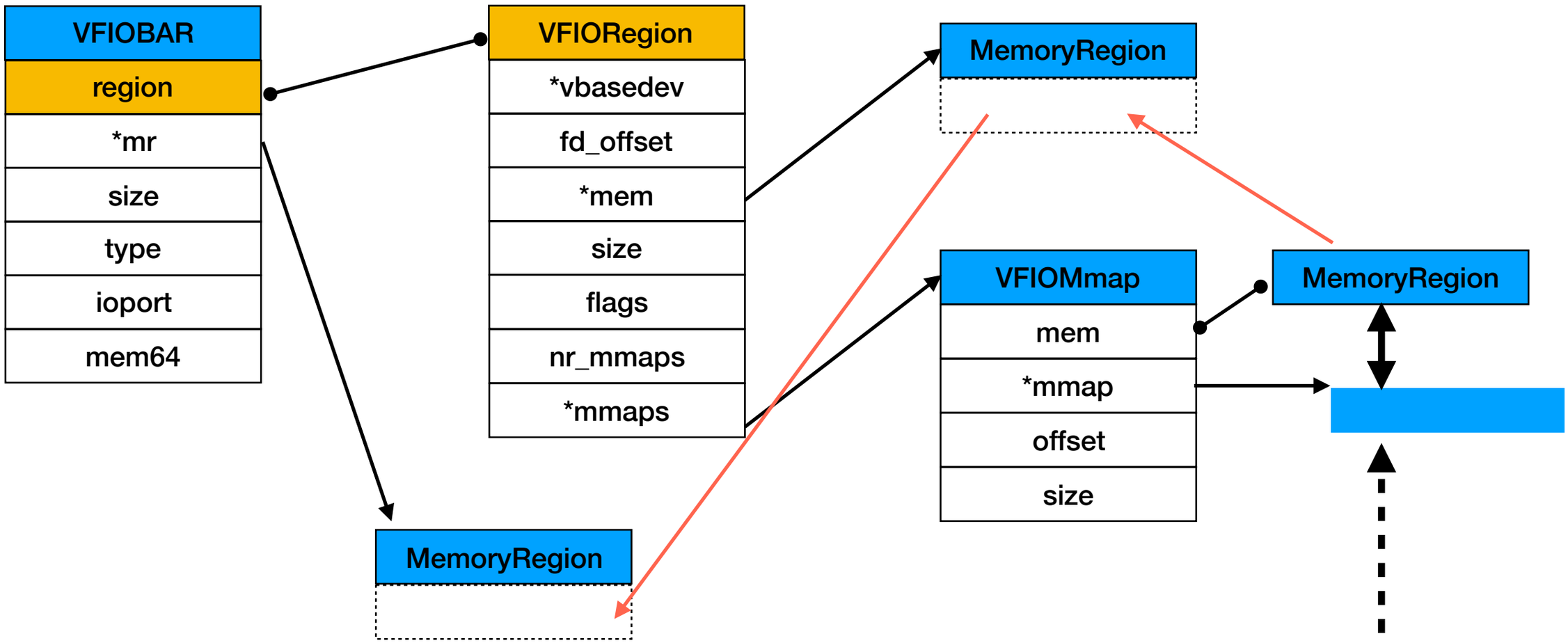
hw/vfio/pci.c

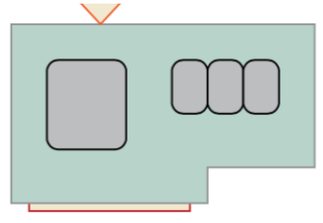
```
struct vfio_region_info {
    __u32   argsz;
    __u32   flags;
#define VFIO_REGION_INFO_FLAG_READ  (1 << 0) /* Region supports read */
#define VFIO_REGION_INFO_FLAG_WRITE (1 << 1) /* Region supports write */
#define VFIO_REGION_INFO_FLAG_MMAP  (1 << 2) /* Region supports mmap */
#define VFIO_REGION_INFO_FLAG_CAPS  (1 << 3) /* Info supports caps */
    __u32   index;      /* Region index */
    __u32   cap_offset; /* Offset within info struct of first cap */
    __u64   size;       /* Region size (bytes) */
    __u64   offset;     /* Region offset from start of device fd */
};
#define VFIO_DEVICE_GET_REGION_INFO _IO(VFIO_TYPE, VFIO_BASE + 8)
```

vfio_realize
->vfio_populate_device
->for every bar region
->vfio_region_setup
->vfio_get_region_info
->ioctl(vbasedev->fd, VFIO_DEVICE_GET_REGION_INFO, *info)
->init VFIORegion
->vfio_bars_prepare
->for every bar region
->vfio_bar_prepare
->read bar info from vbasedev->fd and init base info of VFIOBAR
->vfio_bars_register
->for every bar region
->memory_region_add_subregion(bar->mr, 0, bar->region.mem)
->vfio_region_mmap
->region->mmaps.mmap = mmap(..vbasedev->fd, region->fd_offset)
->memory_region_init_ram_device_ptr
(region->mmaps.mem,…region->mmaps.mmap)
->memory_region_add_subregion(region->mem,…, &region->mmaps.mem)
->pci_register_bar

MMIO/PIO

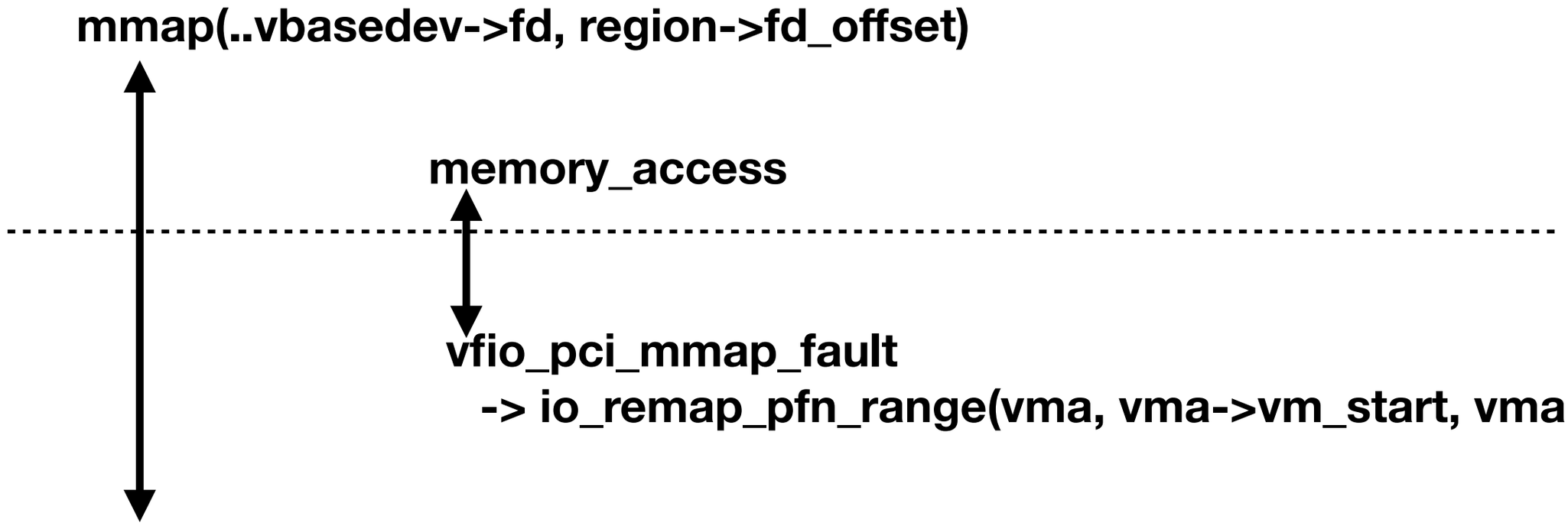
BAR0-5





vfio_populate_device
vfio_bar_prepare
vfio_bar_register
memory_region_add_subregion
mmap
memory_region_init_ram_device_ptr
memory_region_add_subregion

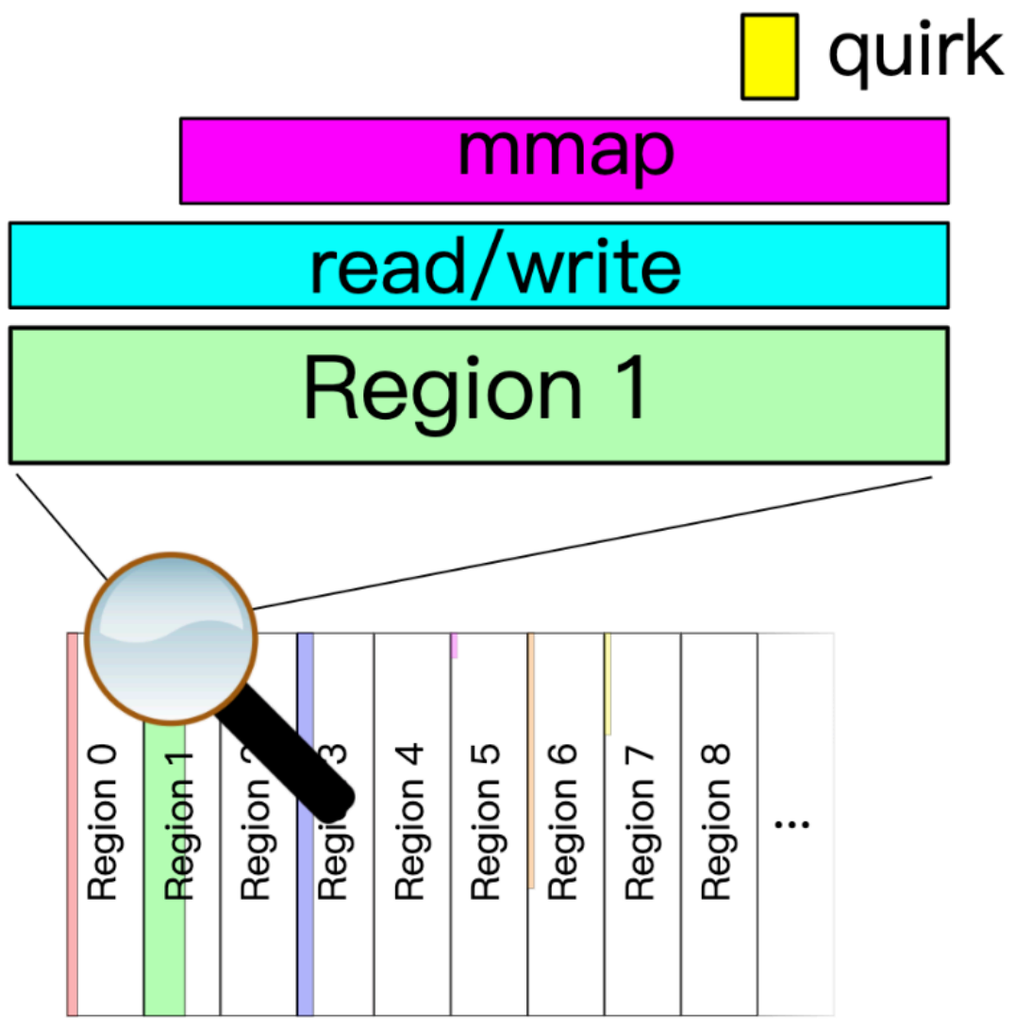
MMIO/PIO



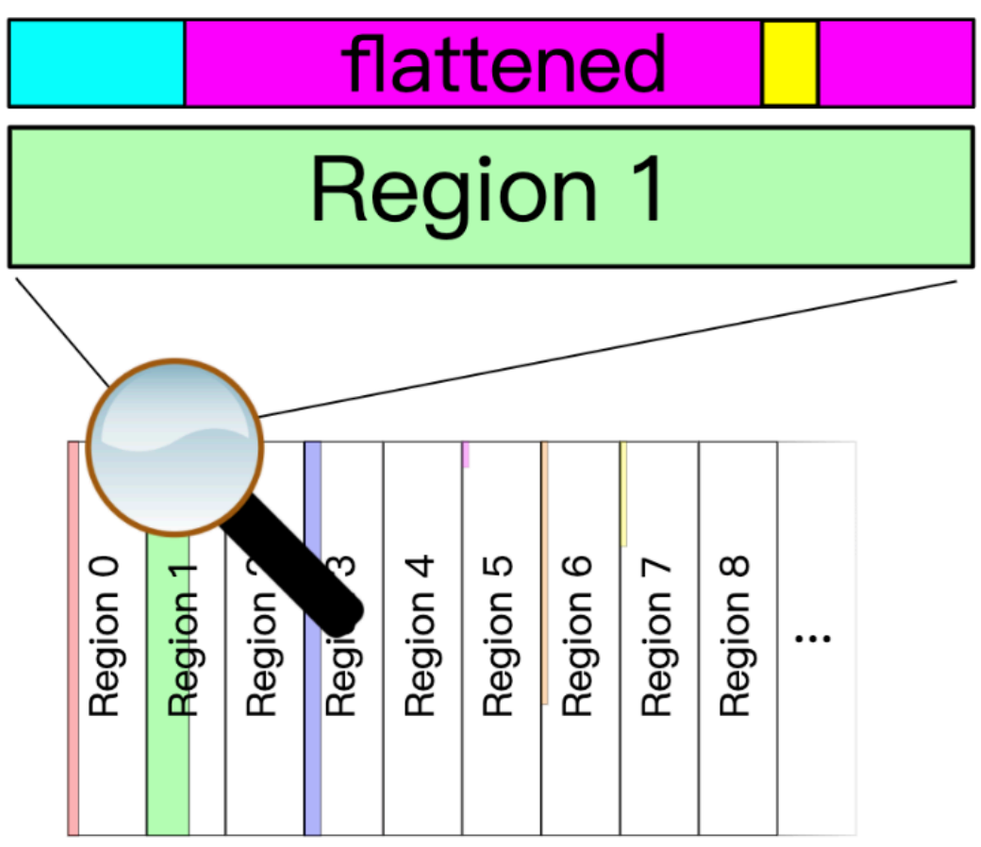
vfio_pci_mmap(*device_data, *vma)
->pci_request_selected_region(..”vfio-pci”)
->vma->vm_pgoff = (pci_resource_start(pdev, index) >> PAGE_SHIFT + pgff)

Region 0	0<<40
Region 1	1<<40
Region 2	2<<40
Region 3	3<<40
Region 4	4<<40
Region 5	5<<40
Region 6	6<<40
Region 7	7<<40
Region 8	8<<40

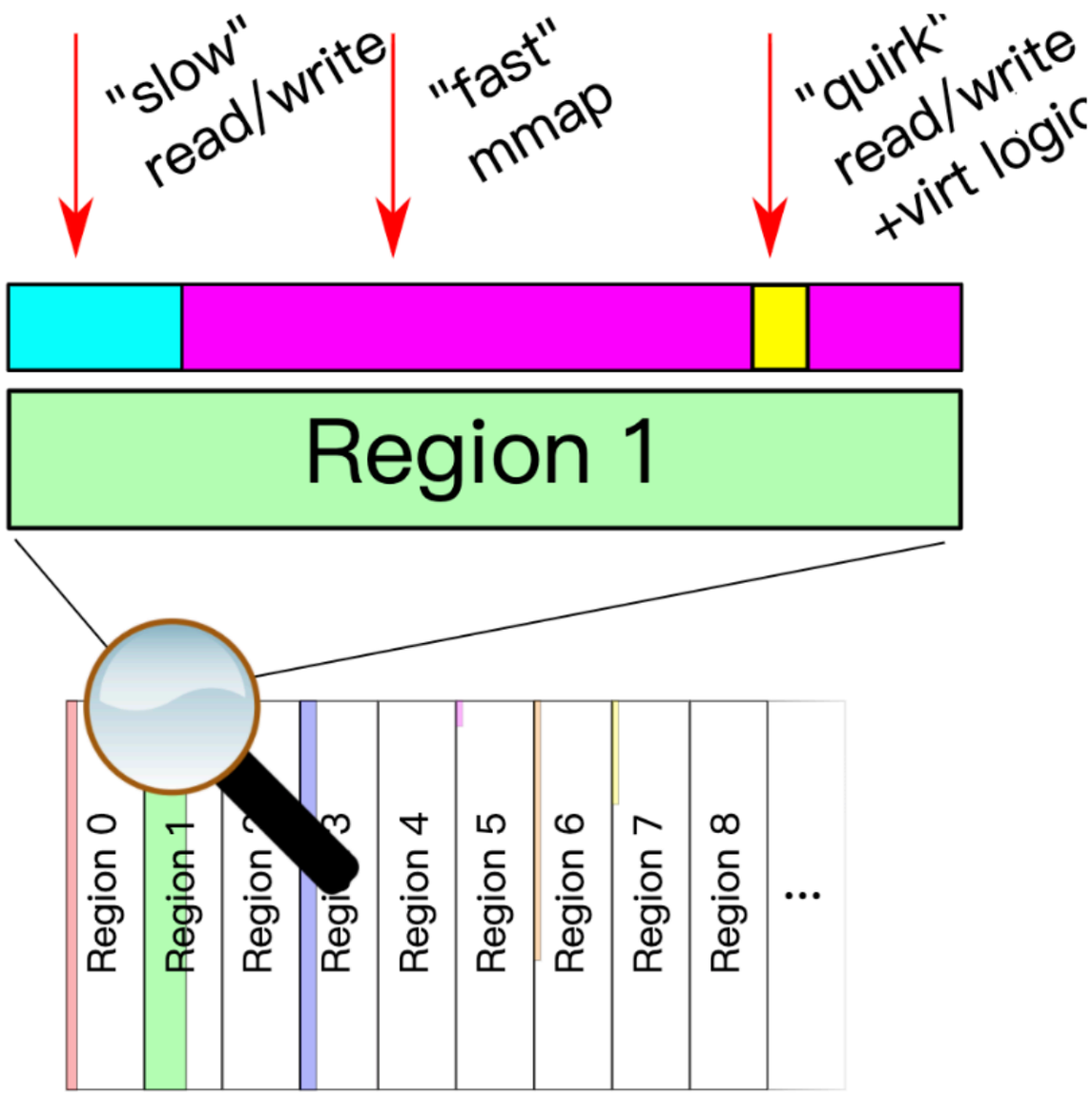
MMIO/PIO


quirk
mmap
read/write
Region 1
Region 0
Region 1
Region 4
Region 5
Region 6
Region 7
Region 8

MMIO/PIO


flattened
Region 1
Region 0
Region 1
Region 4
Region 5
Region 6
Region 7
Region 8

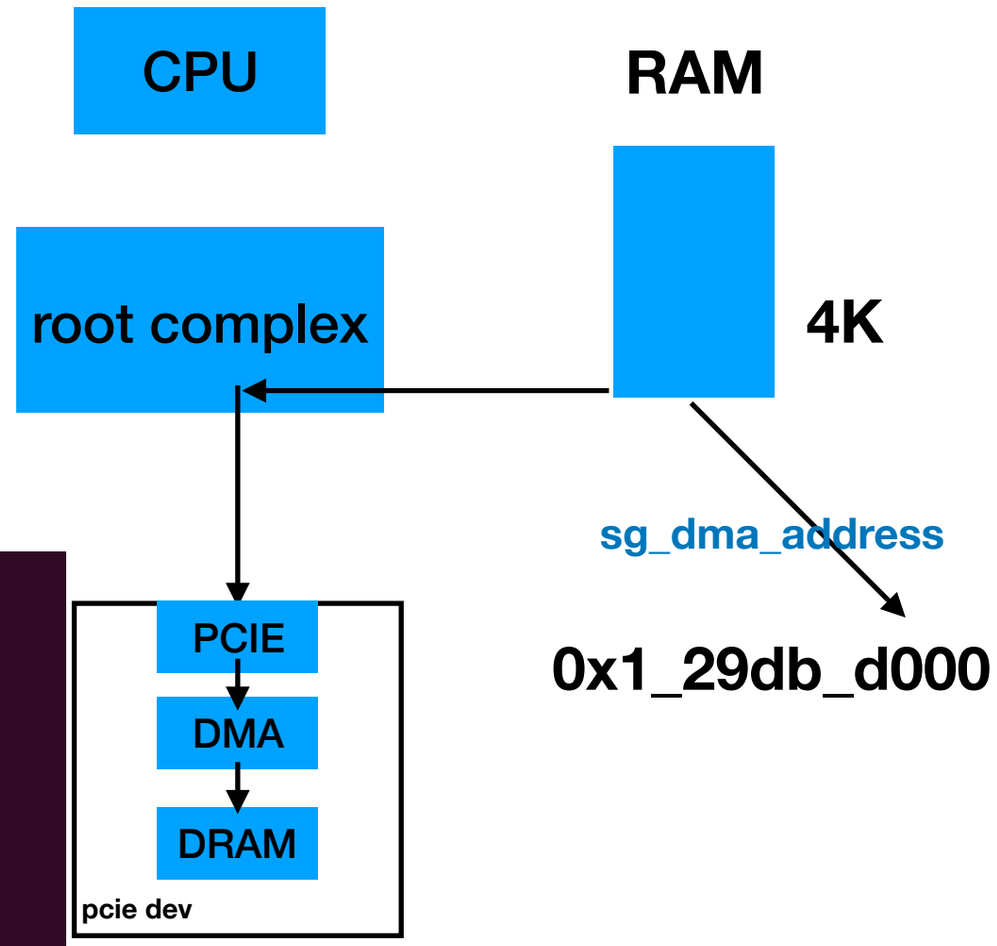
MMIO/PIO


"slow"
read/write
"fast"
mmap
"quirk"
read/write
+virt logic
Region 1
Region 0
Region 1
Region 2
Region 3
Region 4
Region 5
Region 6
Region 7
Region 8
...

DMA Remapping

DMA

```
struct sg_table *sgt;
sg_alloc_table(sgt, 1, GFP_KERNEL);
pages = kcalloc(1, 4096);
sg_set_page(sgt->sgl, pages[0], 4096, 0);
pci_map_sg(dev, sgt->sgl, 1, DMA_TO_DEVICE);
```




```
struct xdma_desc {
    u32 control;
    u32 bytes;       /* transfer length in bytes */
    u32 src_addr_lo;    /* source address (low 32-bit) */
    u32 src_addr_hi;    /* source address (high 32-bit) */
    u32 dst_addr_lo;    /* destination address (low 32-bit) */
    u32 dst_addr_hi;    /* destination address (high 32-bit) */
    /*
     * next descriptor in the single-linked list of descriptors;
     * this is the PCIe (bus) address of the next descriptor in the
     * root complex memory
     */
    u32 next_lo;        /* next desc address (low 32-bit) */
    u32 next_hi;        /* next desc address (high 32-bit) */
} __packed;
```

```
xdma:dump_desc: 0xffff8be334790000/0x00: 0xad4b0013 0xad4b0013 magic|extra_adjacent|control
xdma:dump_desc: 0xffff8be334790004/0x04: 0x00000400 0x00000400 bytes
xdma:dump_desc: 0xffff8be334790008/0x08: 0x29dbd000 0x29dbd000 src_addr_lo
xdma:dump_desc: 0xffff8be33479000c/0x0c: 0x00000001 0x00000001 src_addr_hi
xdma:dump_desc: 0xffff8be334790010/0x00: 0x00000000 0x00000000 dst_addr_lo
xdma:dump_desc: 0xffff8be334790014/0x04: 0x00000000 0x00000000 dst_addr_hi
xdma:dump_desc: 0xffff8be334790018/0x08: 0x00000000 0x00000000 next_addr
xdma:dump_desc: 0xffff8be33479001c/0x0c: 0x00000000 0x00000000 next_addr_pad
```

DMAR

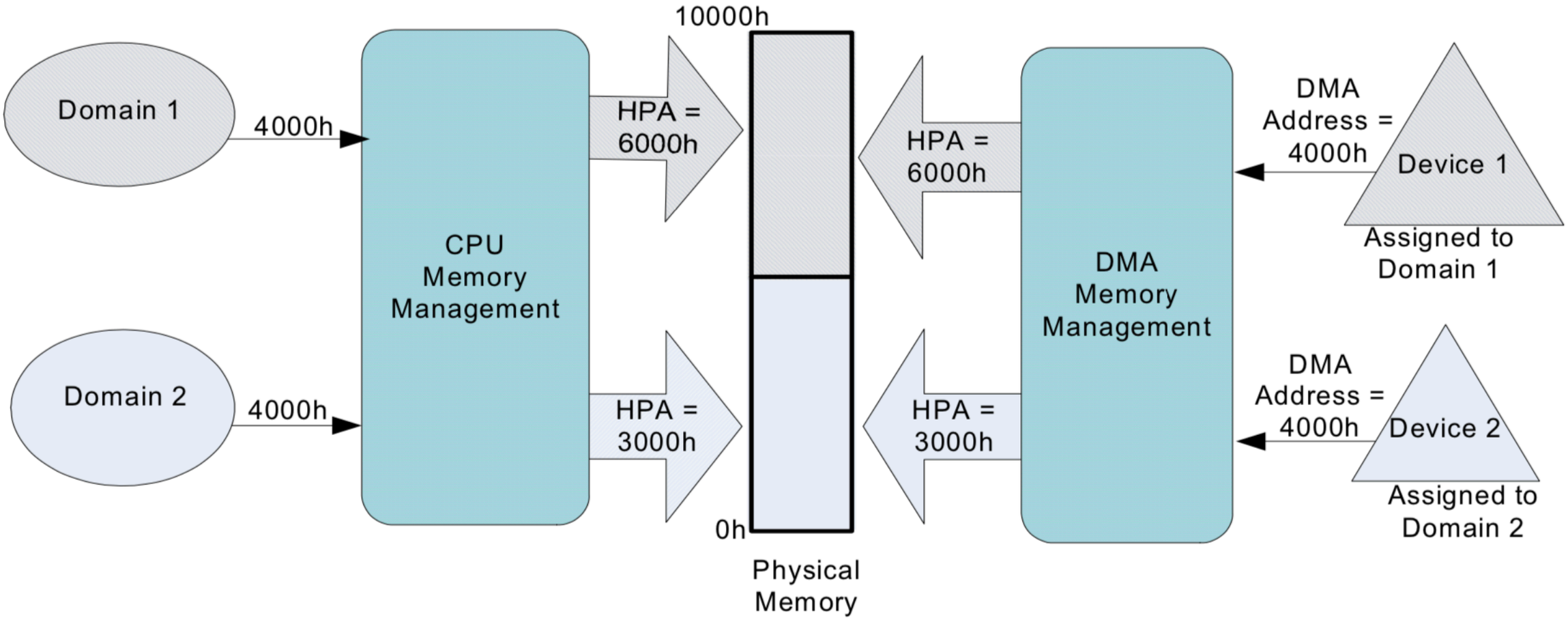
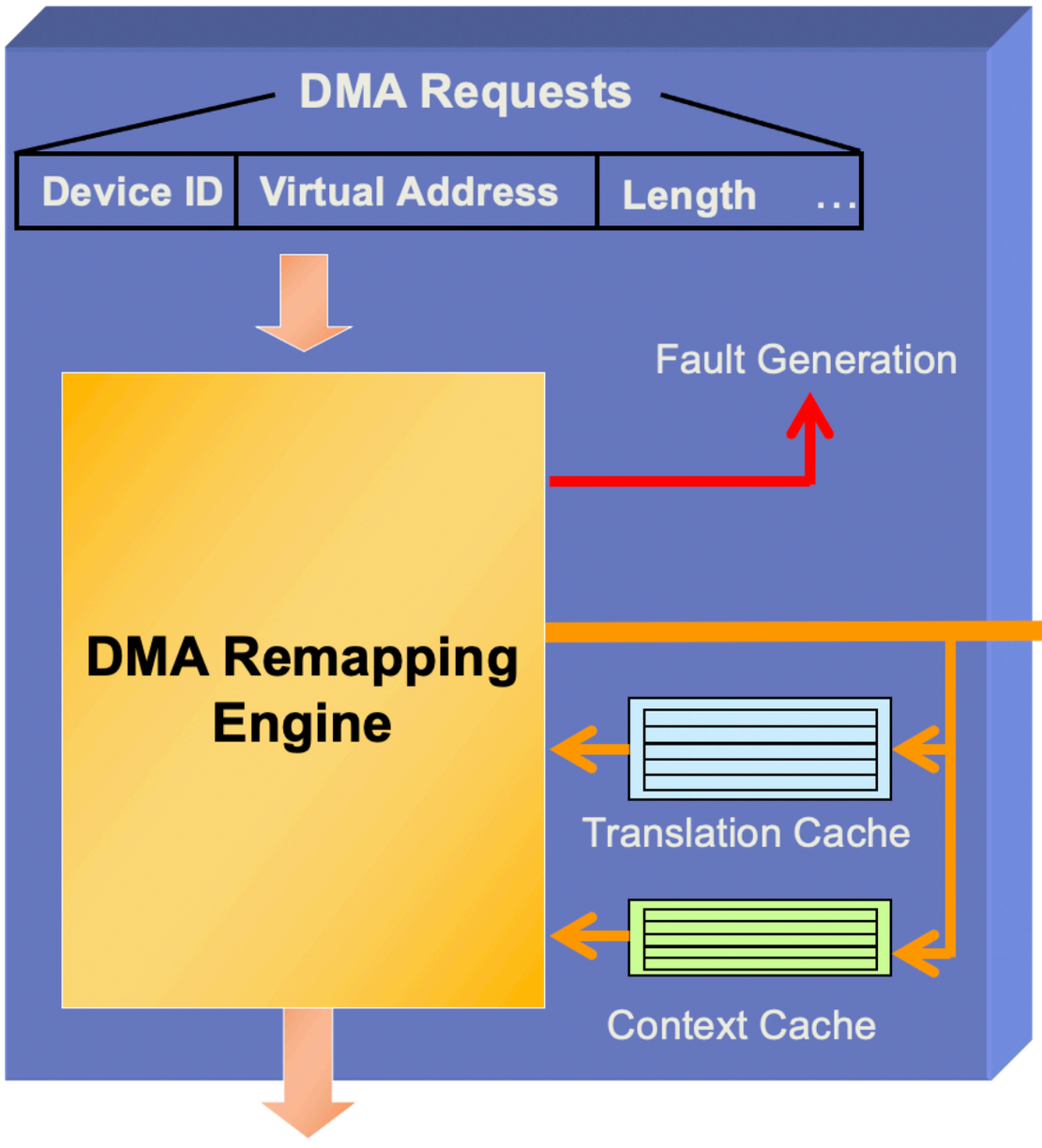
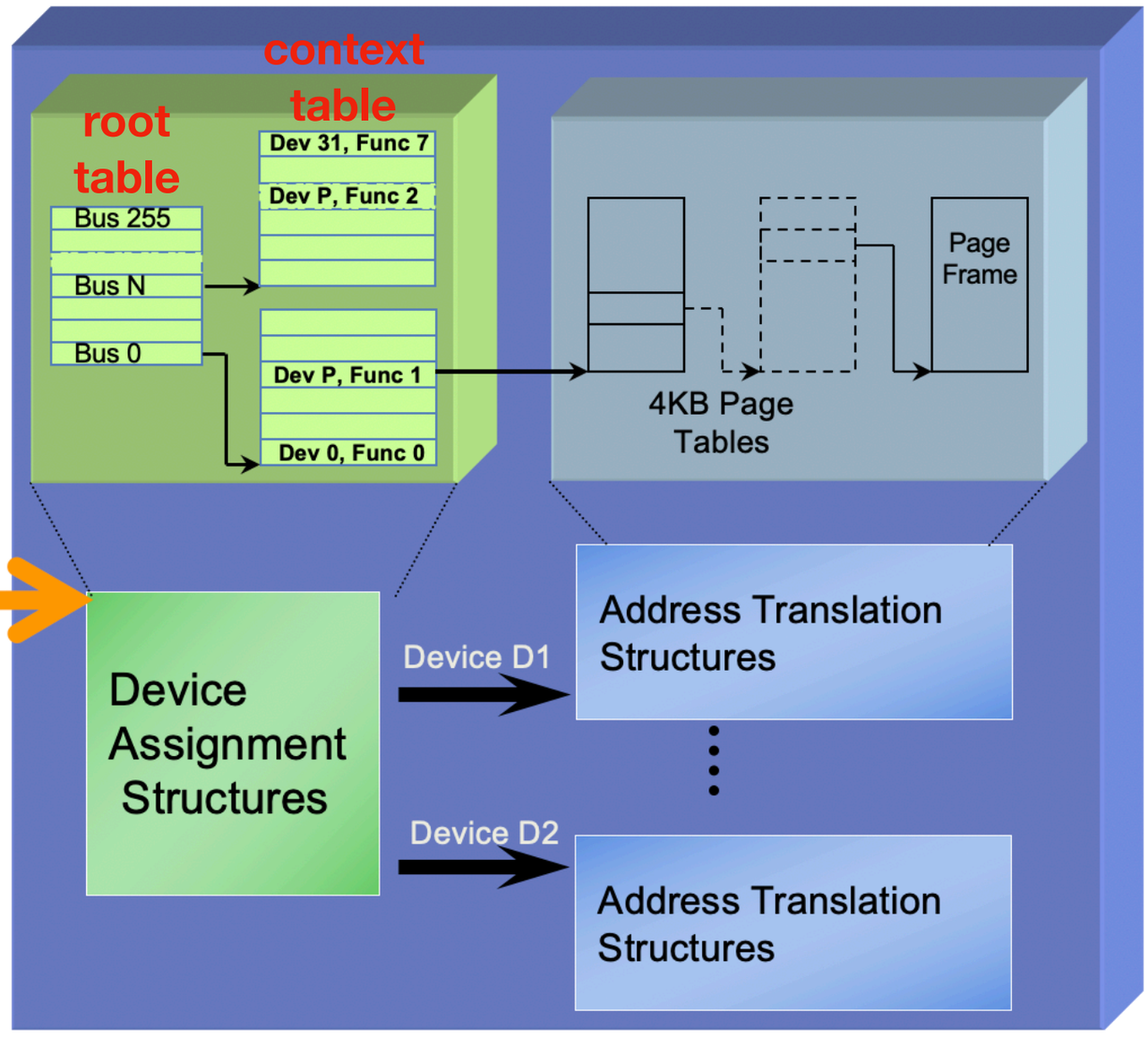


Figure 3-5. DMA Address Translation

DMAR



Memory Access with System Physical Address



Memory-resident Partitioning And Translation Structures

vfio_dma_map

vfio_realize

->vfio_get_group

->vfio_connect_container

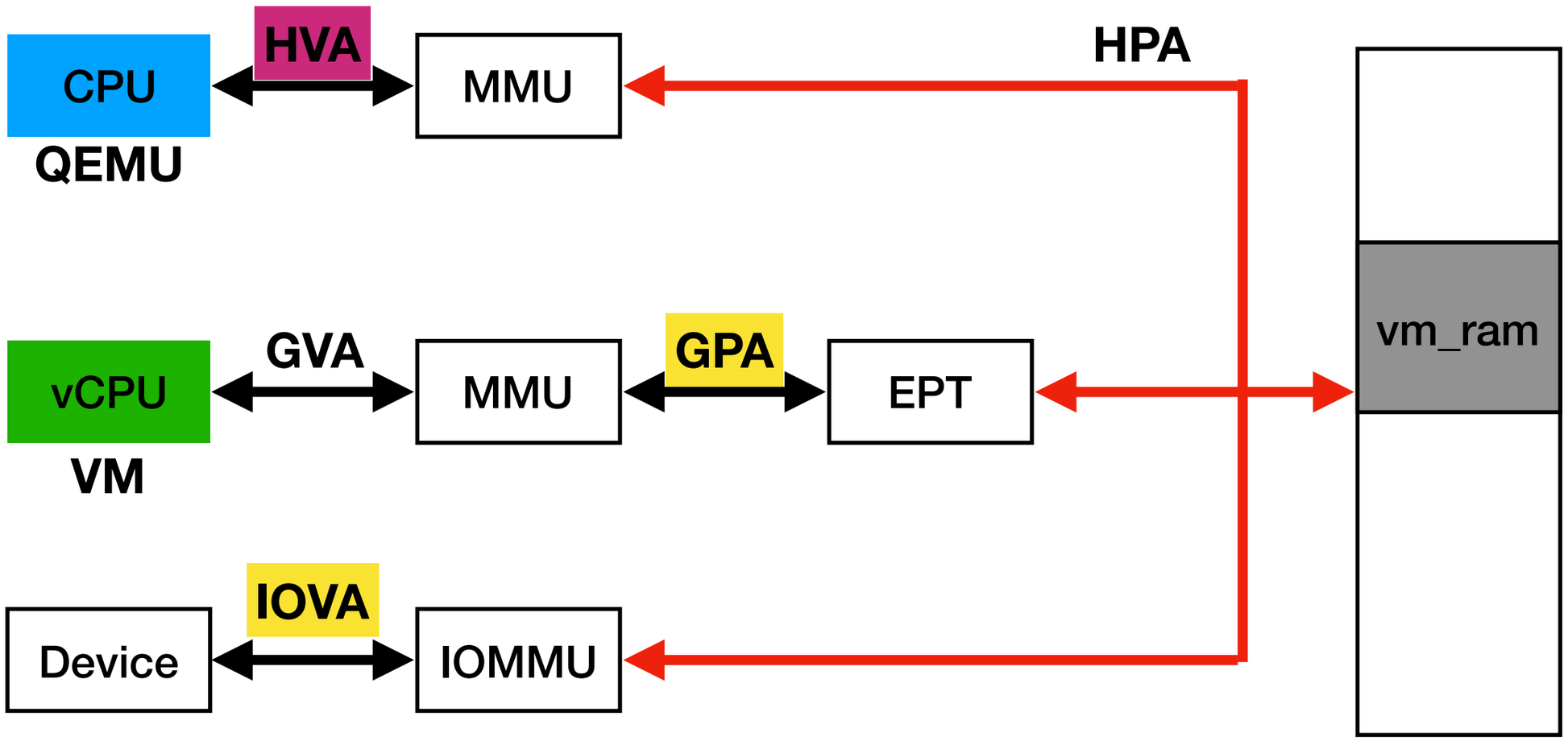
->memory_listener_register(vfio_memory_listener)

```
static const MemoryListener vfio_memory_listener = {
    .region_add = vfio_listener_region_add,
    .region_del = vfio_listener_region_del,
};
```

```
static int vfio_dma_map(VFIOContainer *container, hwaddr iova,
                        ram_addr_t size, void *vaddr, bool readonly)
{
    struct vfio_iommu_type1_dma_map map = {
        .argsz = sizeof(map),
        .flags = VFIO_DMA_MAP_FLAG_READ,
        .vaddr = (__u64)(uintptr_t)vaddr,
        .iova = iova,
        .size = size,
    };

    if (!readonly) {
        map.flags |= VFIO_DMA_MAP_FLAG_WRITE;
    }

    ioctl(container->fd, VFIO_IOMMU_MAP_DMA, &map);
    ...
}
```

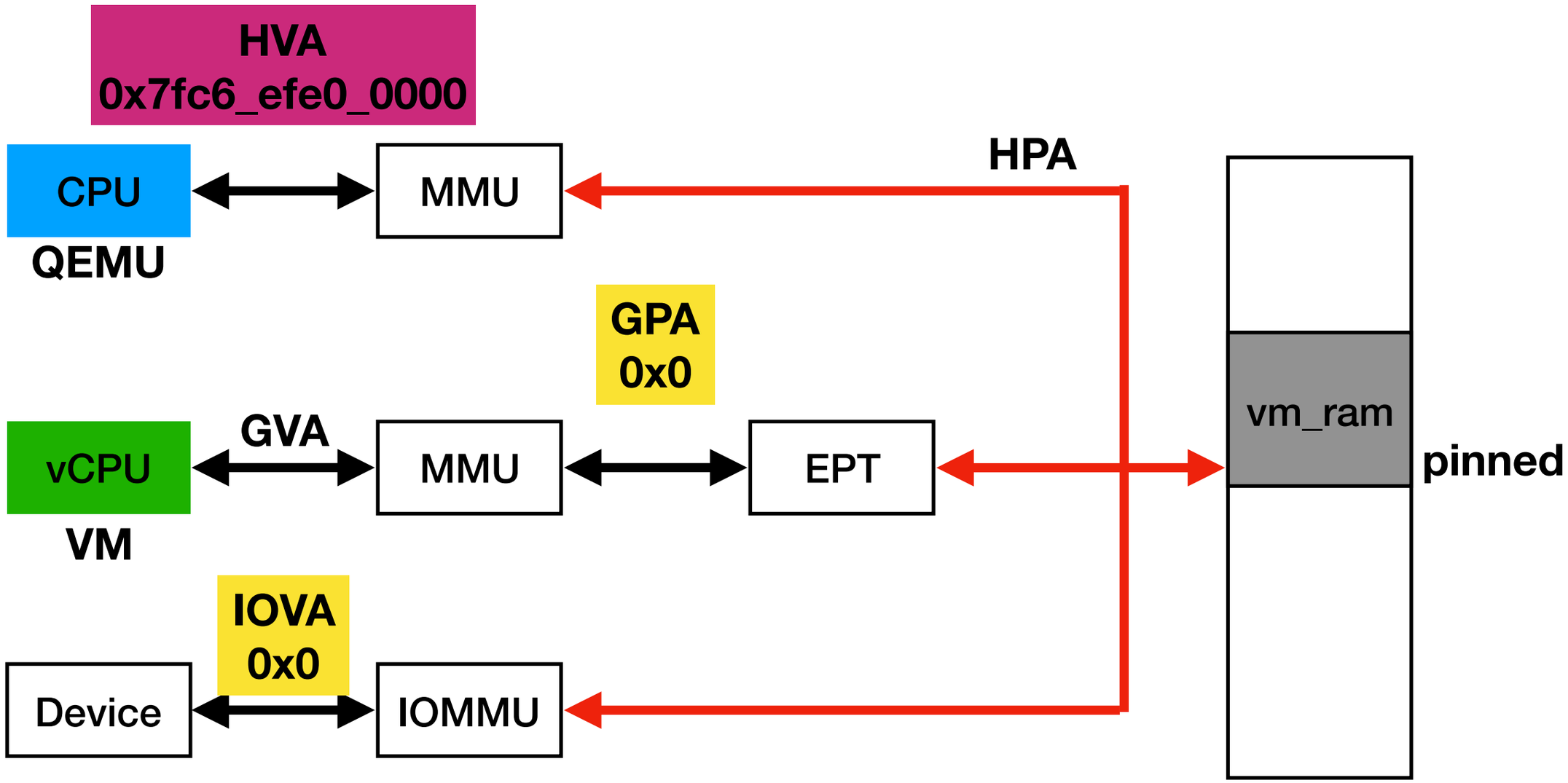
vfio_dma_map

HVA: host virtual address
HPA: host physical address
GVA: guest virtual address
GPA: guest physical address
IOVA: I/O virtual address




```
iova = TARGET_PAGE_ALIGN(section->offset_within_address_space);
vaddr = memory_region_get_ram_ptr(section->mr) +
        section->offset_within_region +
        (iova - section->offset_within_address_space);
```

vfio_dma_map



kvm_set_phys_mem

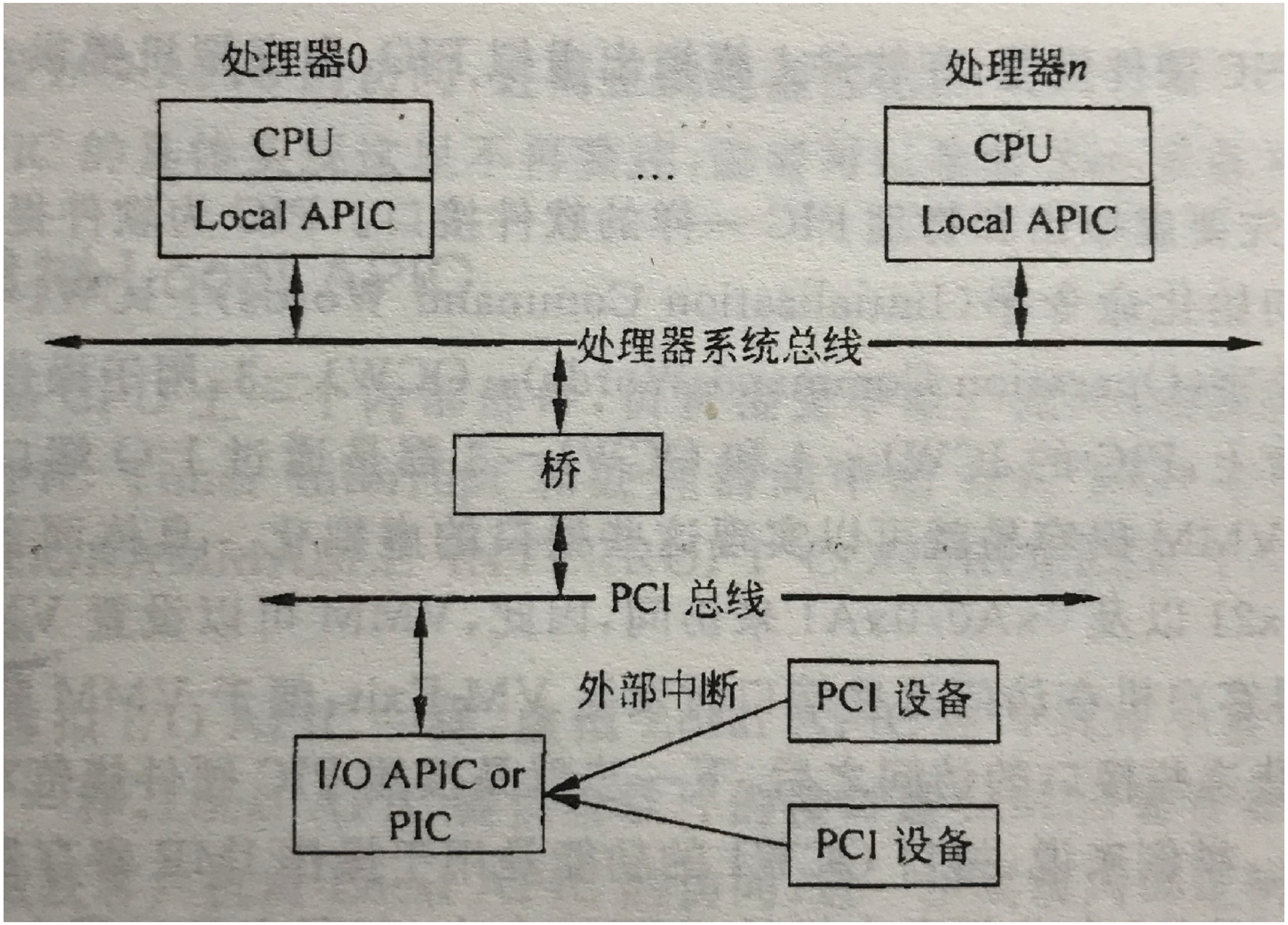
vfio_dma_map

vfio_dma_map

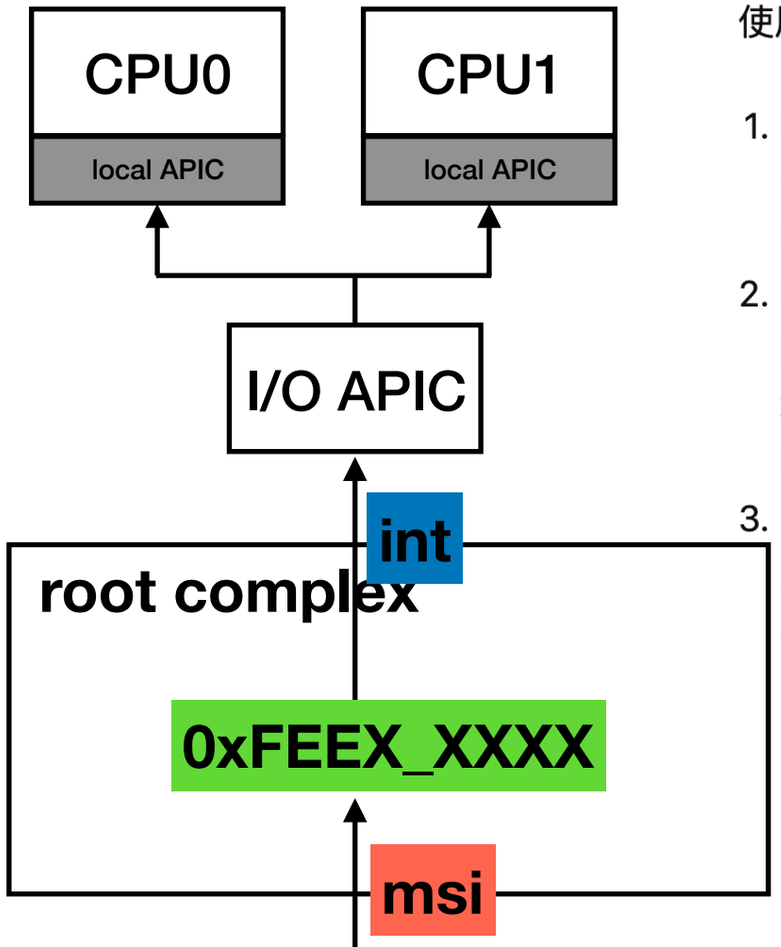

```
vfio_dma_map(0x0, 0xa0000, 0x7fc6efe00000) -- pc.ram
vfio_dma_map(0xc0000, 0x20000, 0x7fc804800000) -- pc.rom
vfio_dma_map(0xe0000, 0x20000, 0x7fc804a20000) -- pc.bios
vfio_dma_map(0x100000, 0xbff00000, 0x7fc6eff00000) -- pc.ram
vfio_dma_map(0xfffc0000, 0x40000, 0x7fc804a00000) -- pc.bios
vfio_dma_map(0x100000000, 0x40000000, 0x7fc7afe00000) -- pc.ram
vfio_dma_map(0xc0000, 0x10000, 0x7fc6efec0000) -- pc.ram
vfio_dma_map(0xd0000, 0x10000, 0x7fc804810000) -- pc.rom
vfio_dma_map(0xe0000, 0x10000, 0x7fc804a20000) -- pc.bios
vfio_dma_map(0xf0000, 0xbff10000, 0x7fc6efef0000) -- pc.ram
vfio_dma_map(0xc0000, 0xbff40000, 0x7fc6efec0000) -- pc.ram
vfio_dma_map(0xf4000000, 0x4000000, 0x7fc6ebc00000) -- vga.vram
vfio_dma_map(0xf8000000, 0x4000000, 0x7fc6e7a00000) -- qxl.vram
vfio_dma_map(0xfc180000, 0x2000, 0x7fc804600000) -- qxl.vrom
vfio_dma_map(0xfc000000, 0x100000, 0x7fc804500000) -- 0000:01:00.0 BAR 0 mmaps[0]
vfio_dma_map(0xfc170000, 0x10000, 0x7fc805fc8000) -- 0000:01:00.0 BAR 1 mmaps[0]
vfio_dma_map(0x0, 0xc0000000, 0x7fc6efe00000) -- pc.ram
vfio_dma_map(0x0, 0xa0000, 0x7fc6efe00000) -- pc.ram
vfio_dma_map(0xc0000, 0xbff40000, 0x7fc6efec0000) -- pc.ram
vfio_dma_map(0xfc160000, 0x10000, 0x7fc804400000) -- qxl.rom
vfio_dma_map(0xfc100000, 0x40000, 0x7fc804200000) -- e1000.rom
vfio_dma_map(0xc0000, 0xa000, 0x7fc6efec0000) -- pc.ram
vfio_dma_map(0xca000, 0x3000, 0x7fc6efeca000) -- pc.ram
vfio_dma_map(0xcd000, 0x3000, 0x7fc6efecd000) -- pc.ram
vfio_dma_map(0xd0000, 0x20000, 0x7fc6efed0000) -- pc.ram
vfio_dma_map(0xf0000, 0x10000, 0x7fc6efef0000) -- pc.ram
vfio_dma_map(0x100000, 0xbff00000, 0x7fc6eff00000) -- pc.ram
vfio_dma_map(0xcd000, 0x1b000, 0x7fc6efecd000) -- pc.ram
vfio_dma_map(0xe8000, 0x8000, 0x7fc6efee8000) -- pc.ram
```

Interrupt

Interrupt

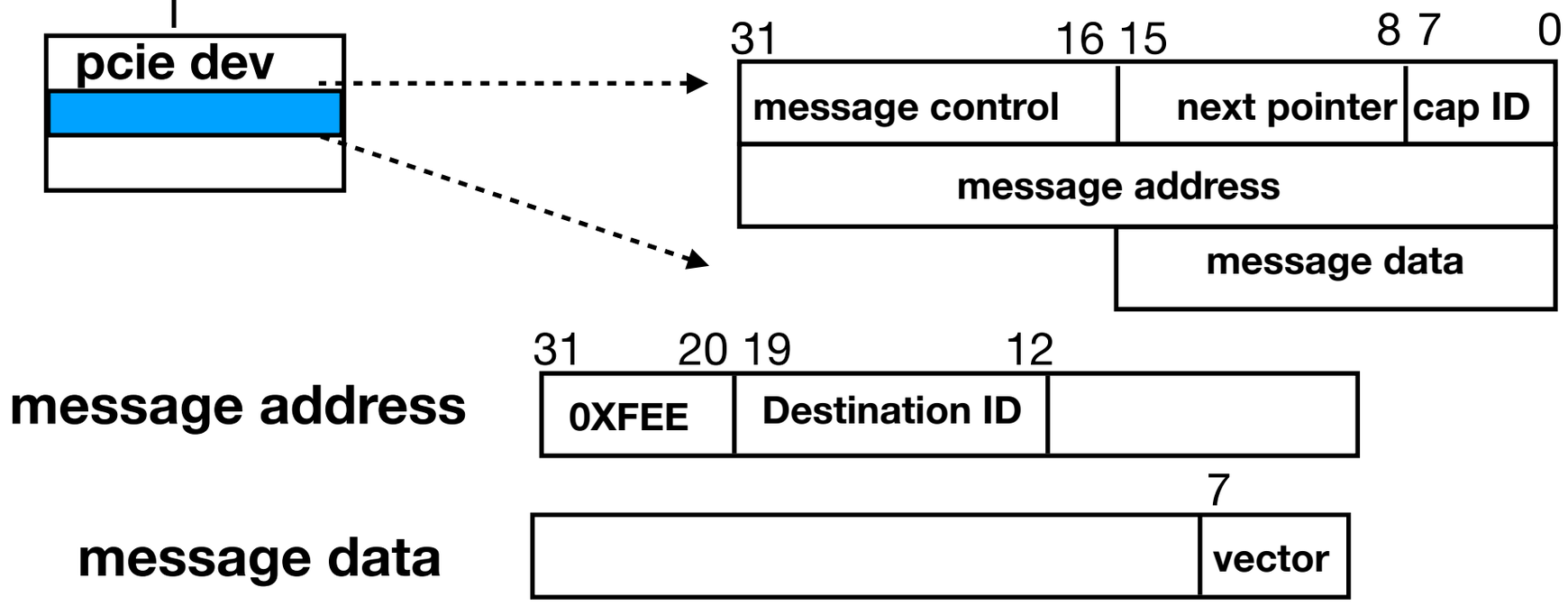

处理器0
CPU
Local APIC
...
处理器n
CPU
Local APIC
处理器系统总线
桥
PCI 总线
外部中断
I/O APIC or
PIC
PCI 设备
PCI 设备

Interrupt



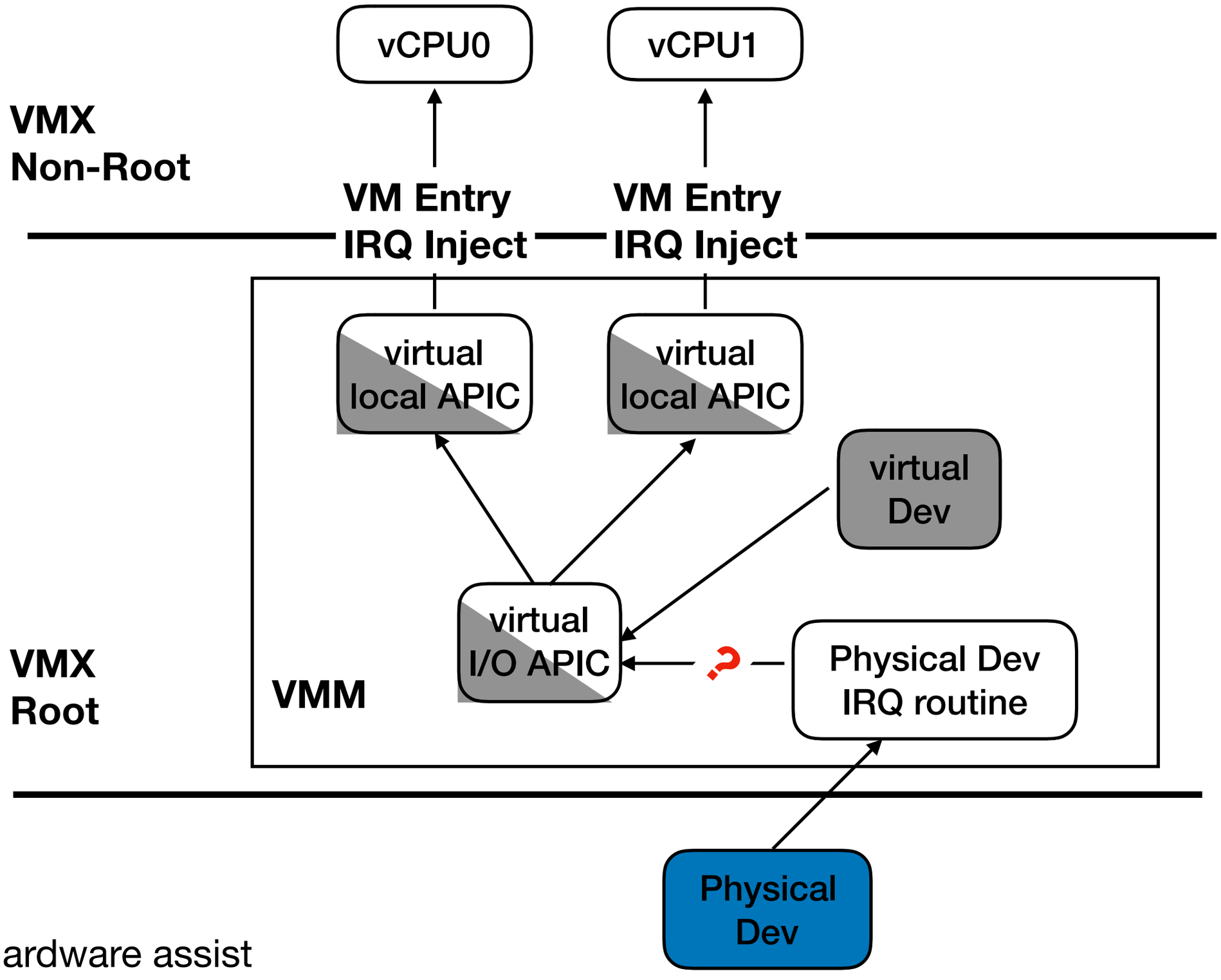
使用MSI传递中断请求的过程：

1. PCIe 设备在发送 MSI/MSI-X中断请求之前， **BIOS** 需要合理设置PCIe设备MSI/MSI-X Capability 寄存器，使 Message Address寄存器的值为0xFEExx00y，同时合理地设置 Message Data寄存器Vector字段。
2. PCIe设备提交MSI/MSI-X中断请求时，需要向0xFEExx00y地址写Message Data寄存器中包含的数据，并以存储器写TLP的形式发送到RC。当桥片收到这个TLP后，发现这个TLP的目 的地址在系统总线Interrupts存储器空间中，则将PCIe总线的存储器写请求转换为系统总线Interrupt Message总线事务，并在系统总线上广播。
3. 系统总线上的CPU，根据APIC ID信息，选择是否接收这个Interrupt Message总线事务，并进入中断状态，之后该CPU将直接从这个总线事务中获得中断向量号，执行相应的中断服 务例程，而不需要从APIC中断控制器获得中断向量。



https://zhuanlan.zhihu.com/p/326412992

Interrupt

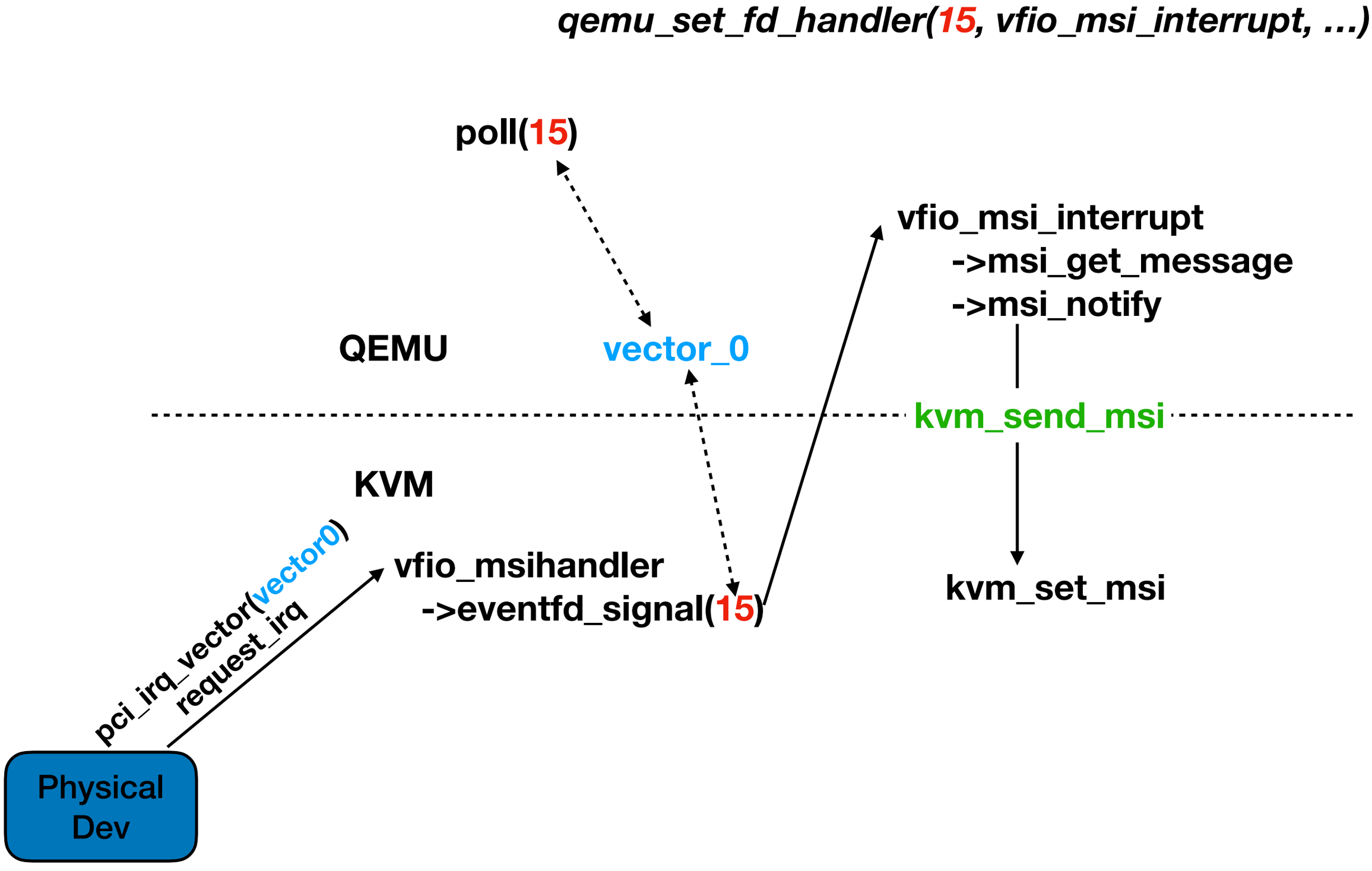

vCPU0
vCPU1
VMX
Non-Root
VM Entry
IRQ Inject
VM Entry
IRQ Inject
virtual
local APIC
virtual
local APIC
virtual
Dev
virtual
I/O APIC
?
Physical Dev
IRQ routine
VMM
VMX
Root
Physical
Dev
hardware assist
virtualization

Interrupt

How to interrupt user-space?

```
EVENTFD(2)              Linux Programmer's Manual              EVENTFD(2)

NAME
   eventfd - create a file descriptor for event notification

SYNOPSIS
   #include <sys/eventfd.h>

   int eventfd(unsigned int initval, int flags);

DESCRIPTION
   eventfd() creates an "eventfd object" that can be used as an event
   wait/notify mechanism by user-space applications, and by the kernel
   to notify user-space applications of events...
```

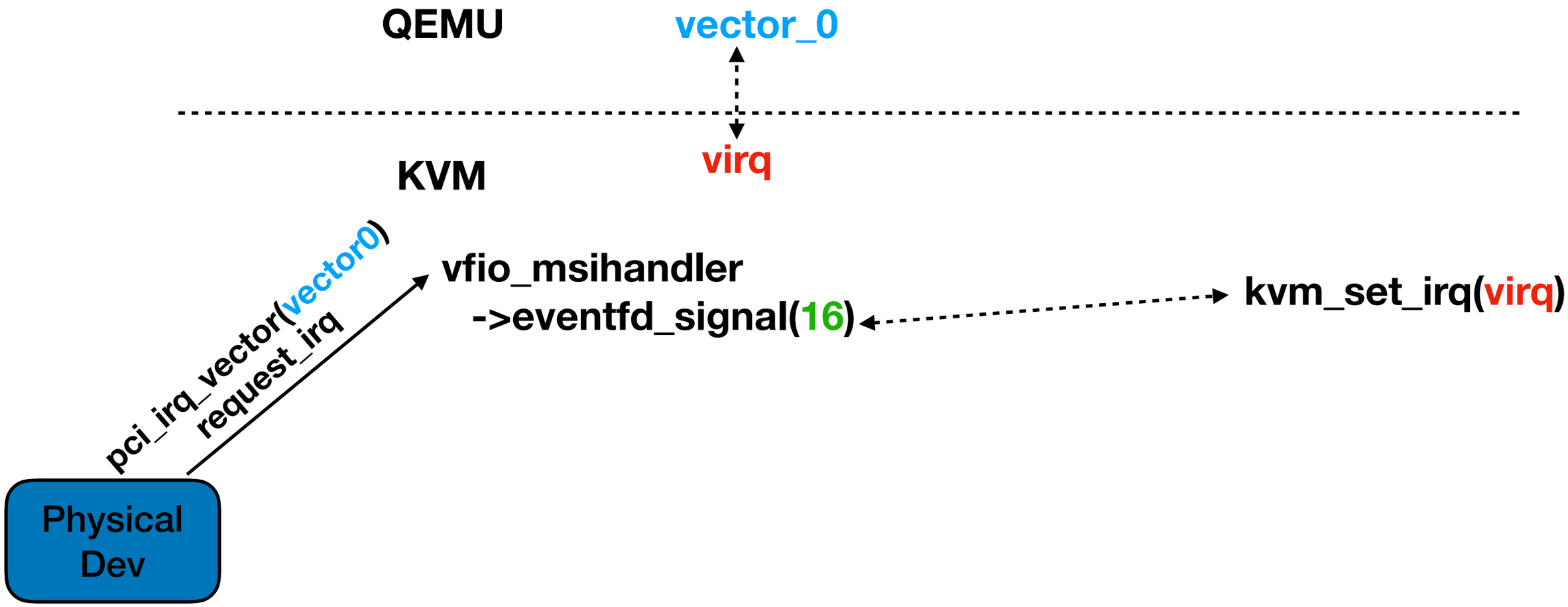
Interrupt-eventfd



Interrupt-irqfd

irqfd: Allows an fd to be used to inject an interrupt to the guest

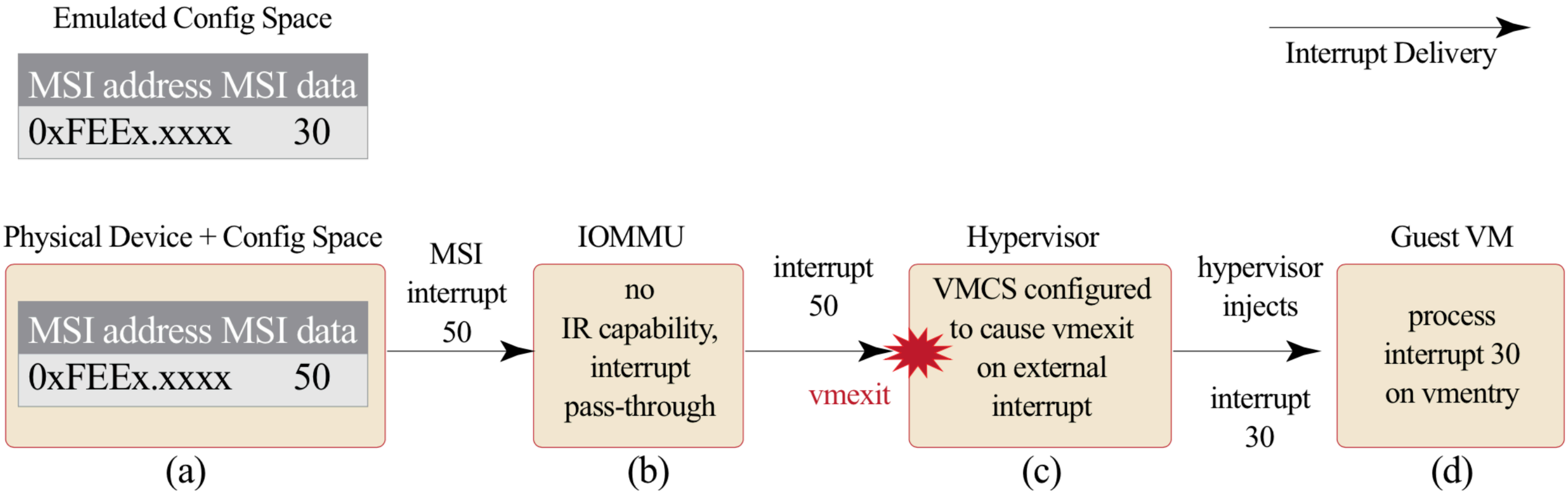
virq = kvm_irqchip_add_msi_route(vector_0)
kvm_irqchip_add_irqfd_notifier_gsi(16, virq)



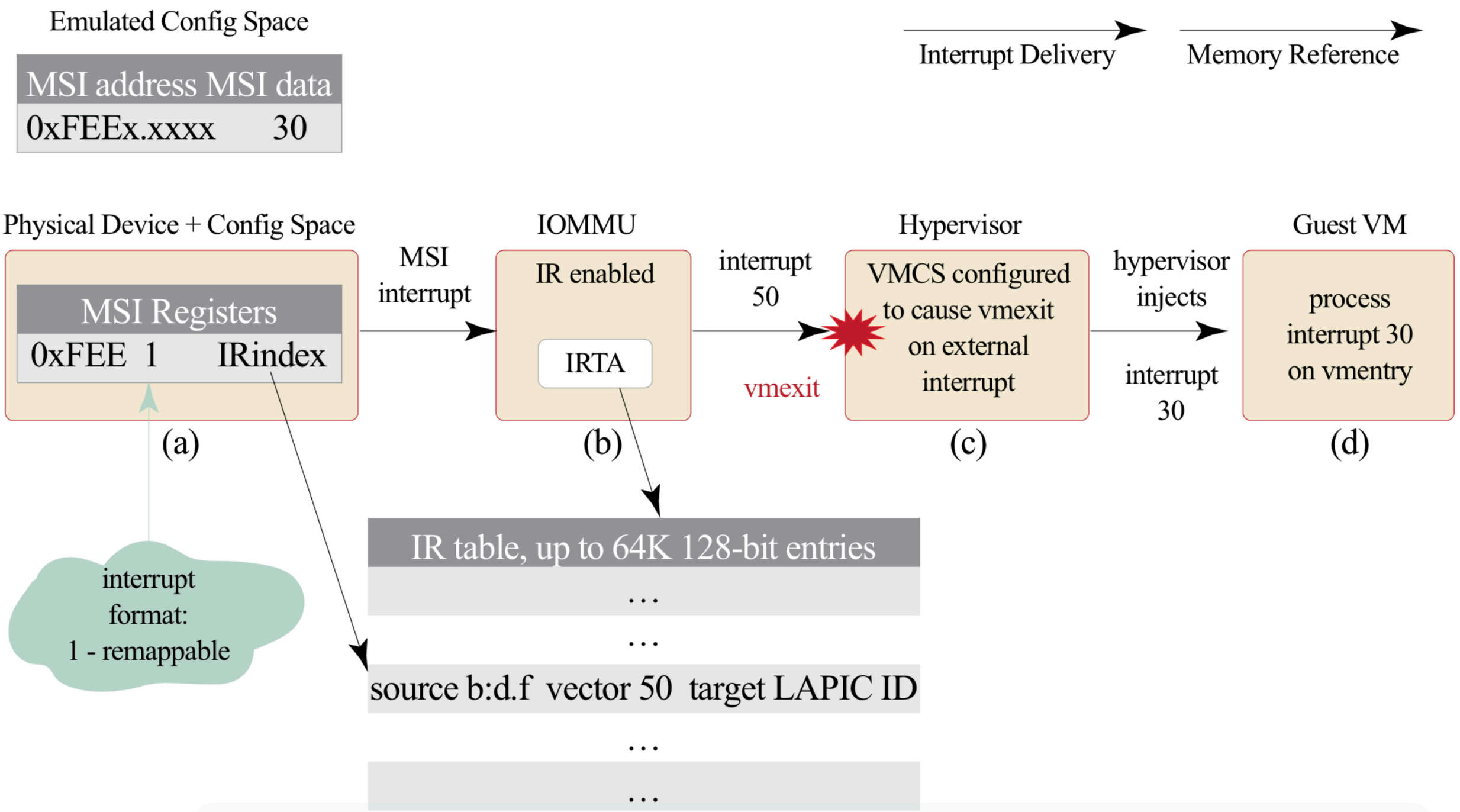
https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=721eecbf4fe995ca94a9edec0c9843b1cc0eaaf3

Interrupt Remapping

Interrupt-remapping

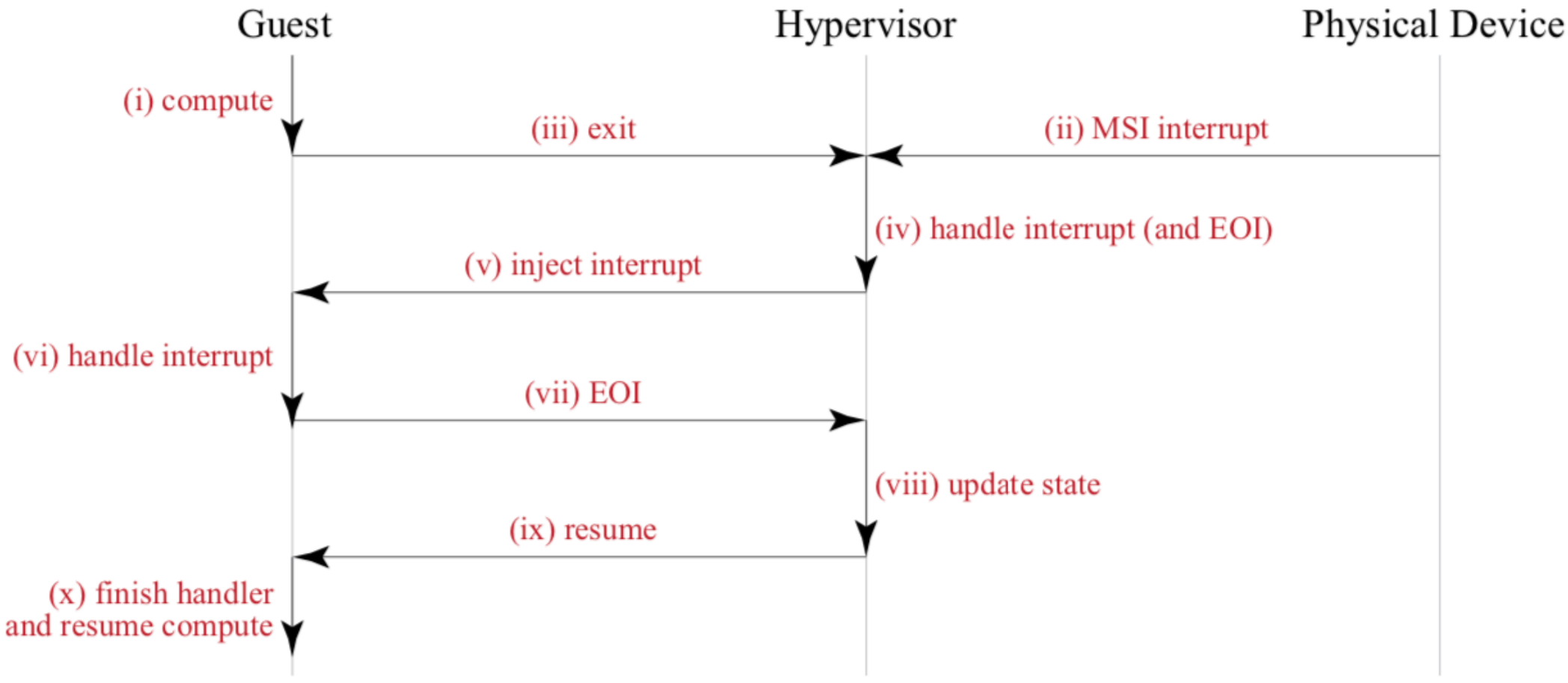


Interrupt-remapping



Accelerating IRQs in hardware

1. Assigned EOI Register
2. Exitless Interrupt
3. Posted Interrupt



Reference

1. KVM irqfd and ioeventfd
2. KVM: add ioeventfd support
3. KVM: irqfd
4. hardware and software support for virtualization
5. An introduction to PCI Device Assignment with VFIO