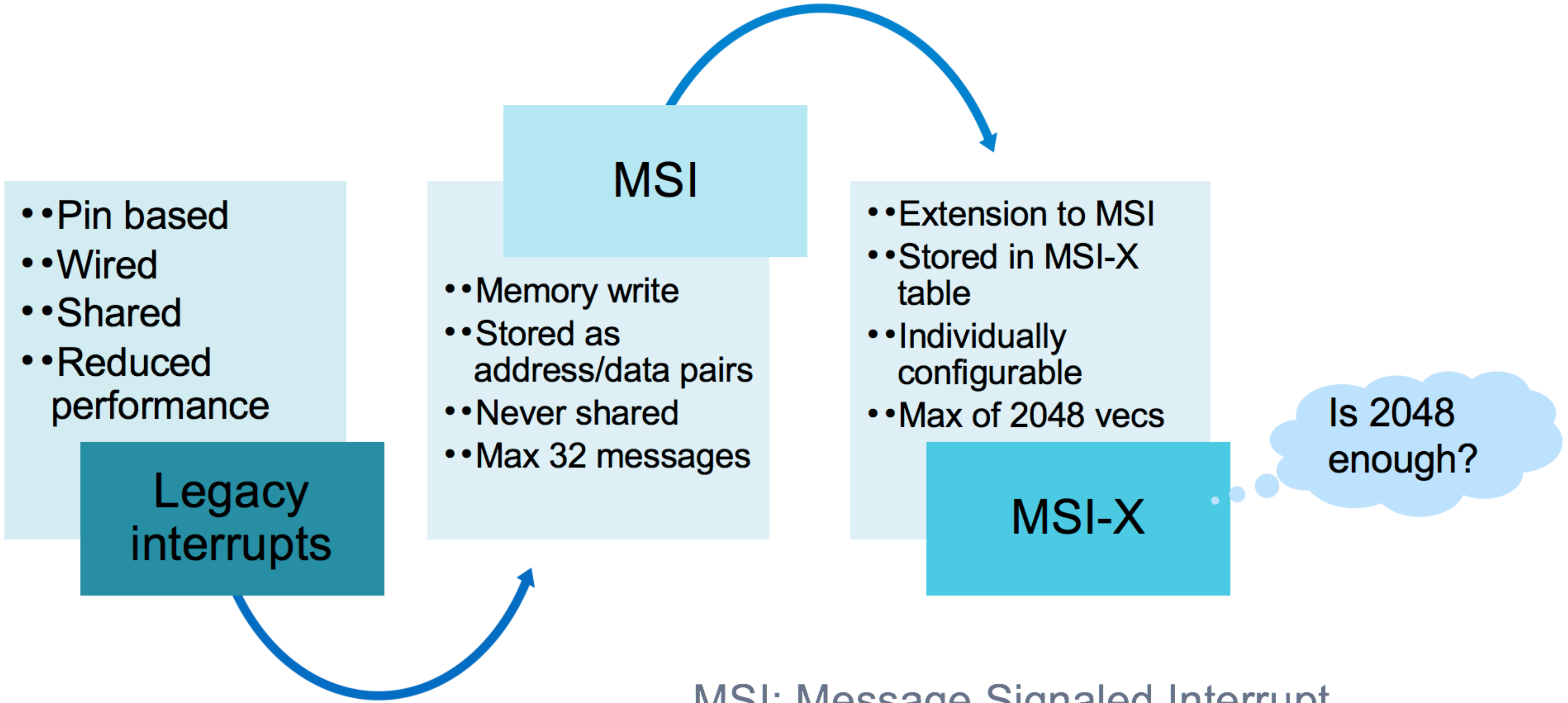
MSI/MSI-x及其虚拟化

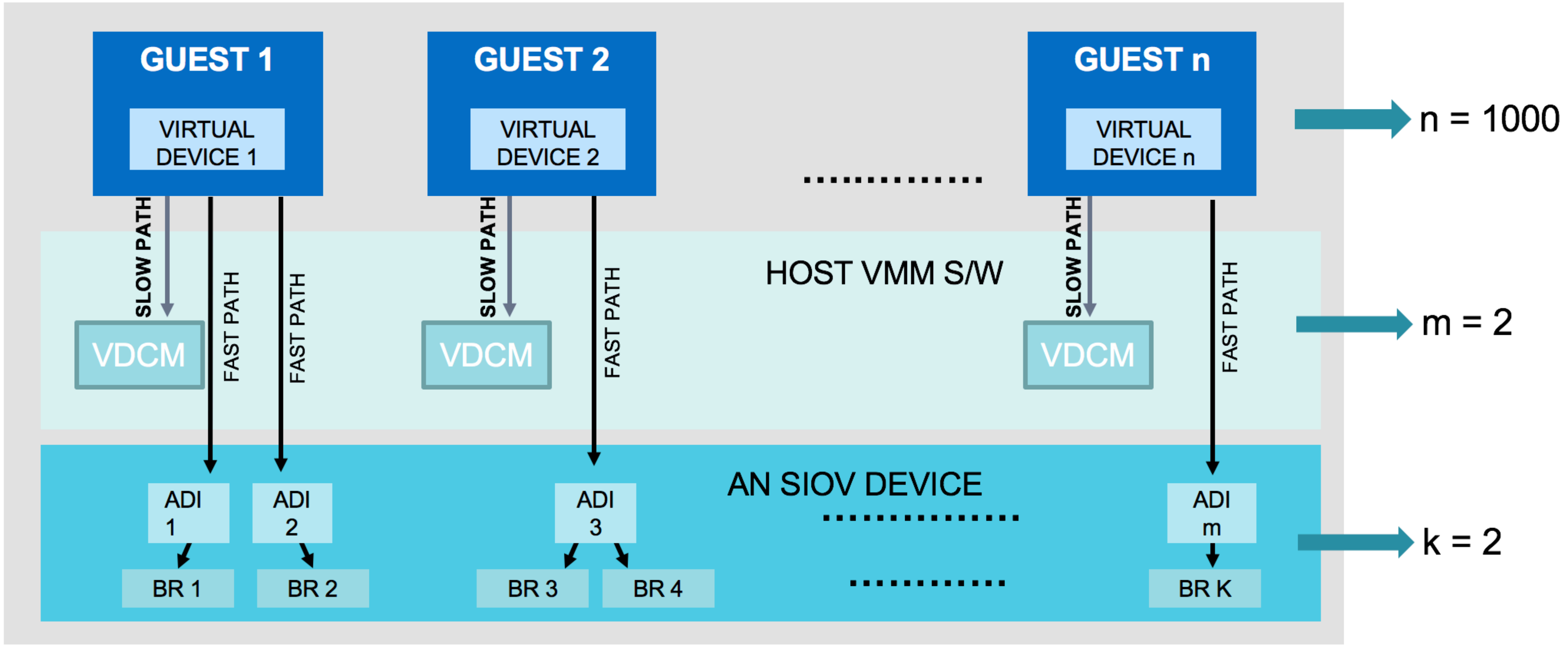
Liu, Junming

The Interrupt Story… So Far



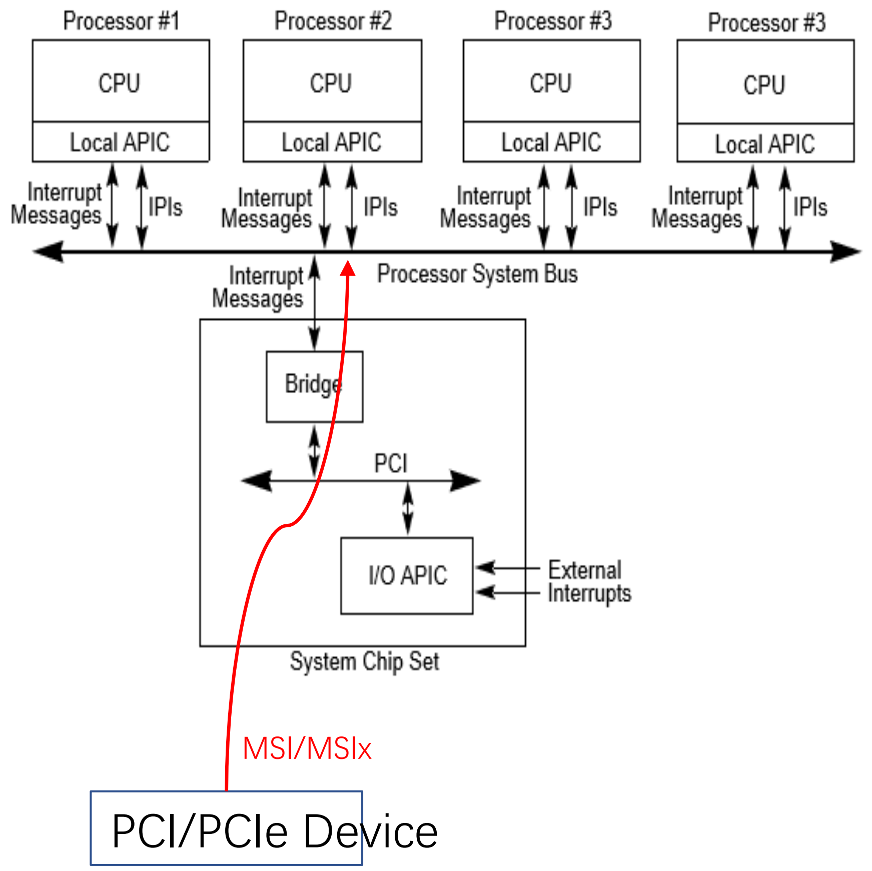
MSI: Message Signaled Interrupt

Need for IMS(Interrupt Message Store)



Total interrupt messages required = n * m * k = 1000 * 2 * 2 = 4000 (>> 2048)

What is MSI/MSI-x?



From the POV of TLP, MSI/MSI-X is memory write TLP from device

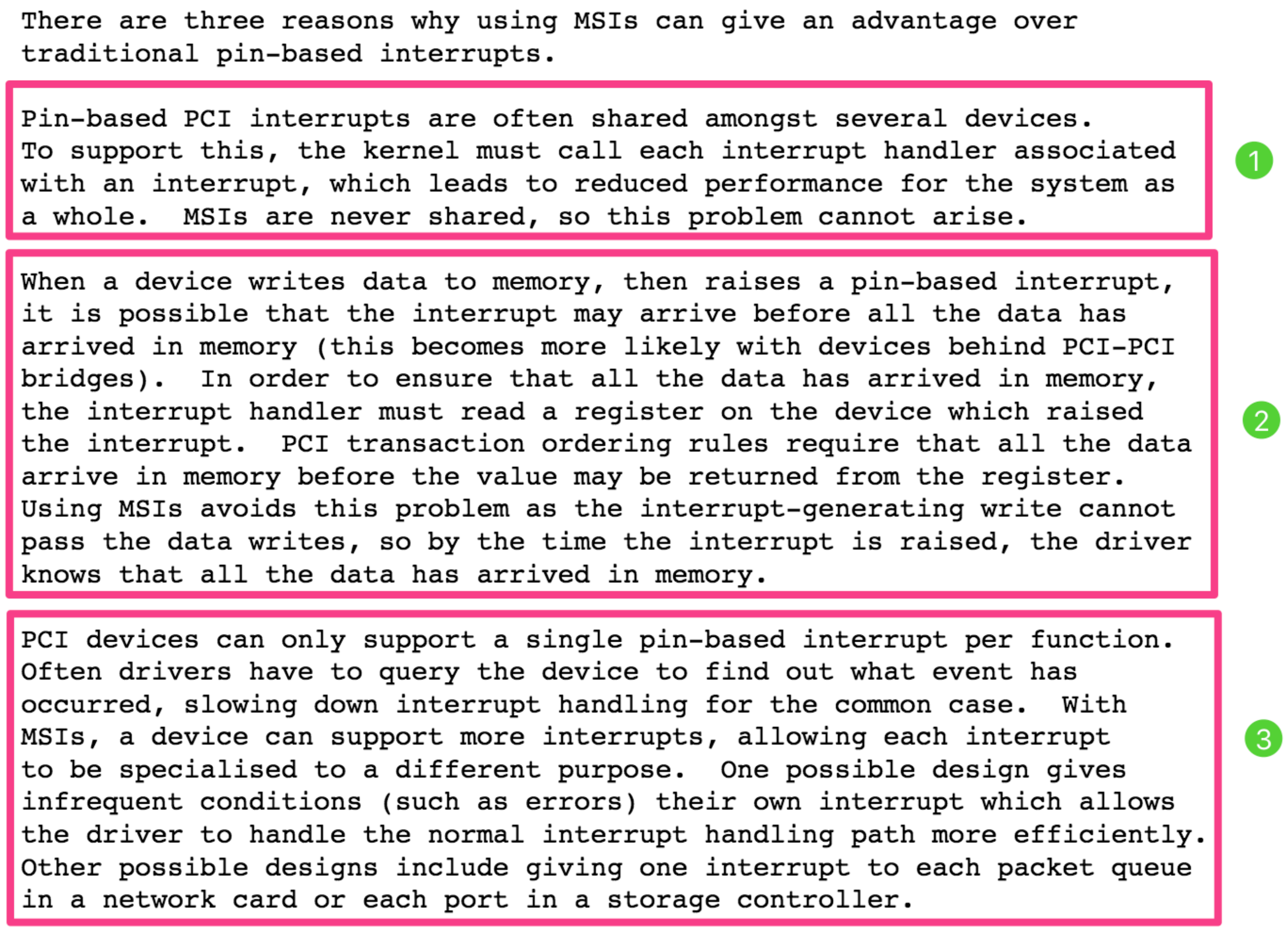
http://xillybus.com/tutorials/pci-express-tlp-pcie-primer-tutorial-guide-1/

MSI-x table vector vs IDT vector

- MSI-x table最多支持2048个中断
- Intel IDT只有256个vector
- 如何分配呢？

每个物理CPU有256个vector，n个CPU就有256 * n个vector，当然够MSI-x table使用了！

Advantages over legacy interrupt



```
There are three reasons why using MSIs can give an advantage over
traditional pin-based interrupts.
```

```
Pin-based PCI interrupts are often shared amongst several devices.
To support this, the kernel must call each interrupt handler associated
with an interrupt, which leads to reduced performance for the system as
a whole.  MSIs are never shared, so this problem cannot arise.
```

1

```
When a device writes data to memory, then raises a pin-based interrupt,
it is possible that the interrupt may arrive before all the data has
arrived in memory (this becomes more likely with devices behind PCI-PCI
bridges).  In order to ensure that all the data has arrived in memory,
the interrupt handler must read a register on the device which raised
the interrupt.  PCI transaction ordering rules require that all the data
arrive in memory before the value may be returned from the register.
Using MSIs avoids this problem as the interrupt-generating write cannot
pass the data writes, so by the time the interrupt is raised, the driver
knows that all the data has arrived in memory.
```

2

```
PCI devices can only support a single pin-based interrupt per function.
Often drivers have to query the device to find out what event has
occurred, slowing down interrupt handling for the common case.  With
MSIs, a device can support more interrupts, allowing each interrupt
to be specialised to a different purpose.  One possible design gives
infrequent conditions (such as errors) their own interrupt which allows
the driver to handle the normal interrupt handling path more efficiently.
Other possible designs include giving one interrupt to each packet queue
in a network card or each port in a storage controller.
```

3

Advantages over legacy interrupt

- Legacy Interrupt

可能多个设备共享一个vector

同一个设备的多个事件
共享一个vector

DMA与pin based interrupt
存在竞态

- MSI/MSI-x

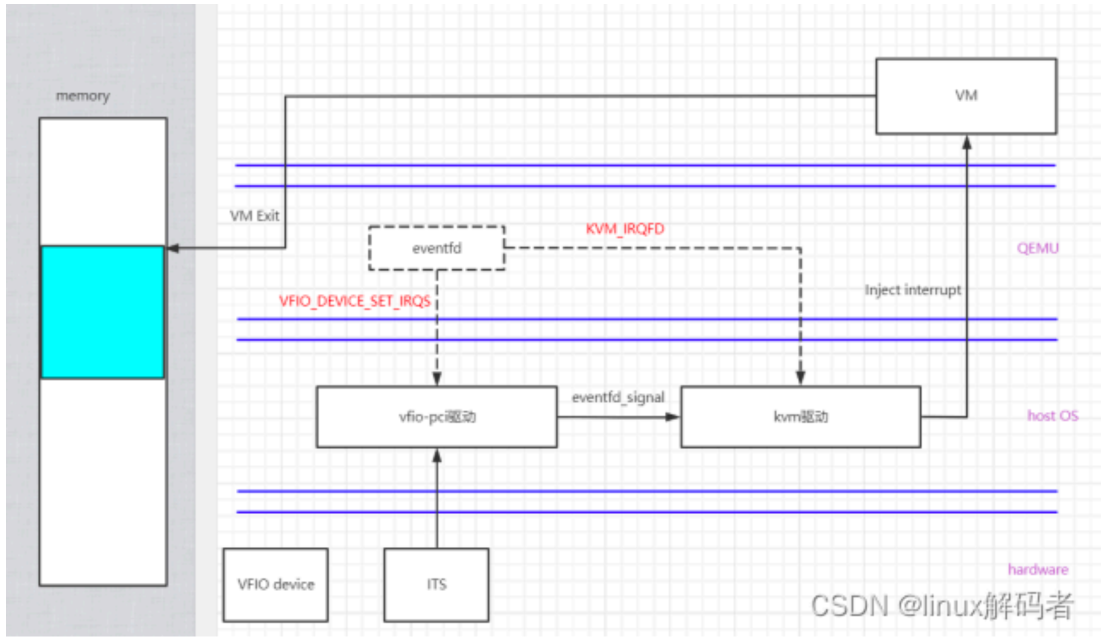
多个设备不共享vector

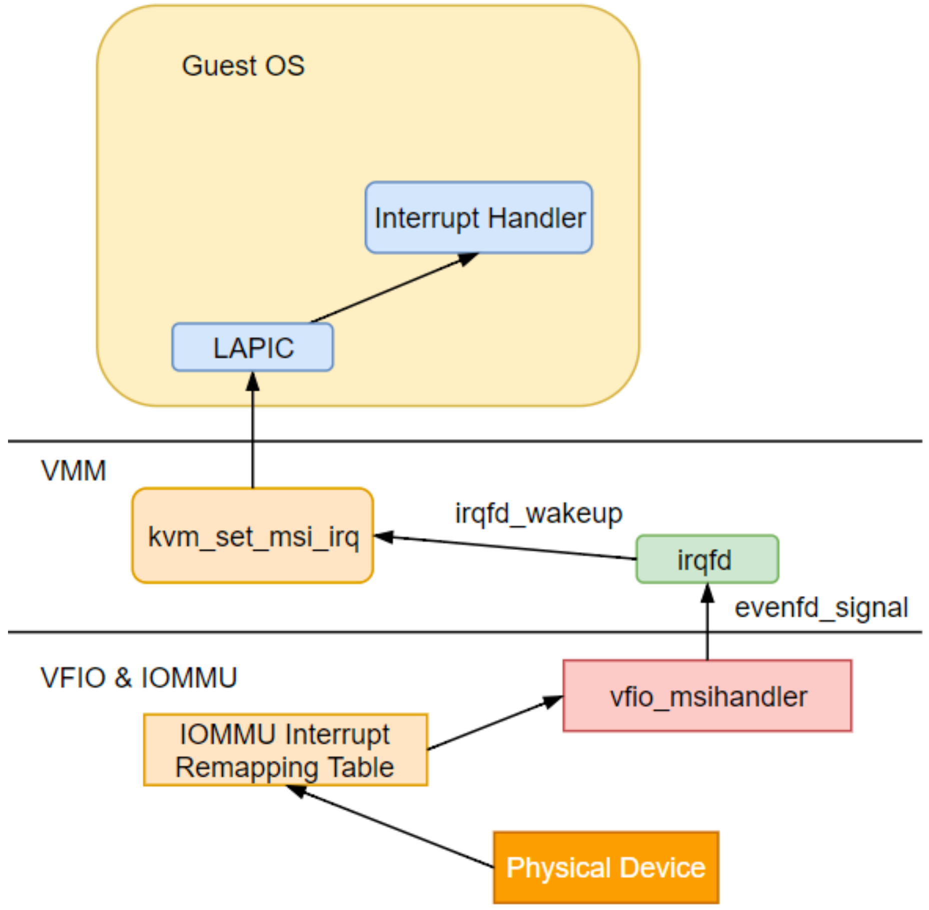
同一个设备的多个事件
拥有独立的vector

PCIe协议保证了
DMA(memory read/write TLP)与MSI(memory write TLP)
的顺序性

https://elixir.bootlin.com/linux/v5.3-rc4/source/Documentation/PCI/msi-howto.rst
https://en.wikipedia.org/wiki/Message_Signaled_Interrupts

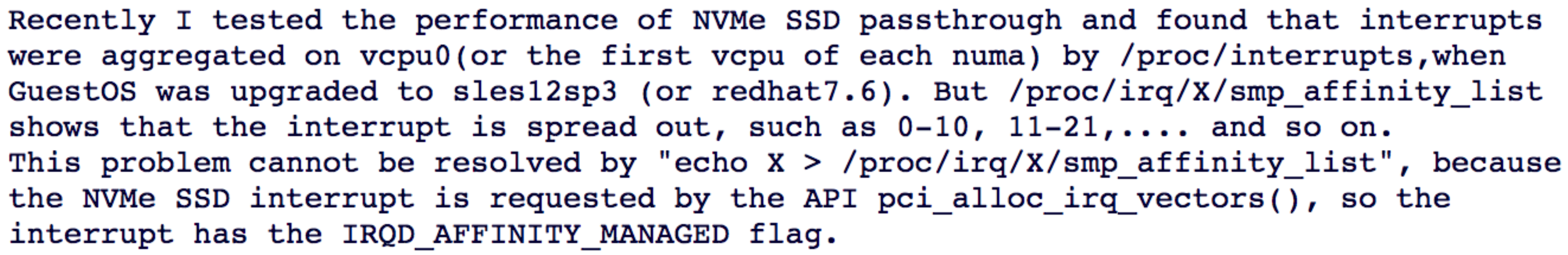
Interrupt remapping in VFIO



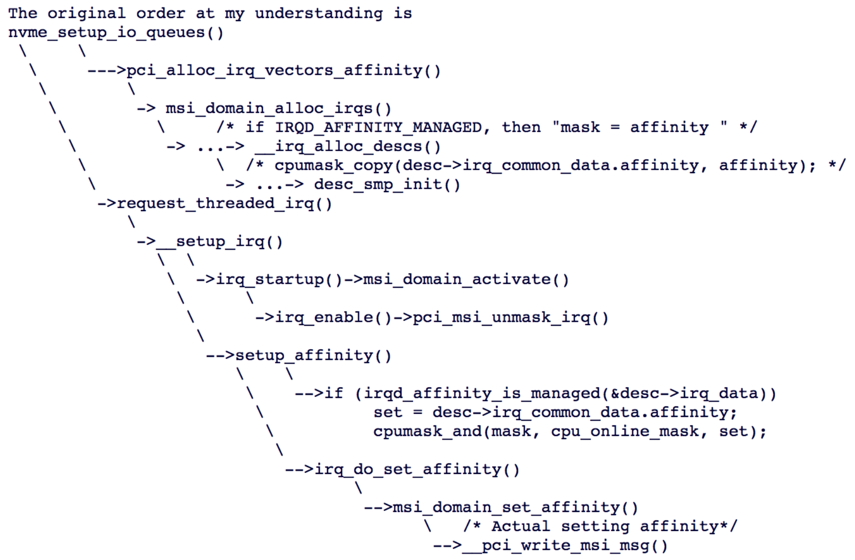


https://kernelgo.org/vtd_interrupt_remapping_code_analysis.html
https://blog.csdn.net/flyingnosky/article/details/123748153

IRQ balance



```
Recently I tested the performance of NVMe SSD passthrough and found that interrupts
were aggregated on vcpu0(or the first vcpu of each numa) by /proc/interrupts,when
GuestOS was upgraded to sles12sp3 (or redhat7.6). But /proc/irq/X/smp_affinity_list
shows that the interrupt is spread out, such as 0-10, 11-21,.... and so on.
This problem cannot be resolved by "echo X > /proc/irq/X/smp_affinity_list", because
the NVMe SSD interrupt is requested by the API pci_alloc_irq_vectors(), so the
interrupt has the IRQD_AFFINITY_MANAGED flag.
```



```
The original order at my understanding is
nvme_setup_io_queues()
 \     \
  \     --->pci_alloc_irq_vectors_affinity()
   \        \
    \        -> msi_domain_alloc_irqs()
     \         \     /* if IRQD_AFFINITY_MANAGED, then "mask = affinity " */
      \         -> ...-> __irq_alloc_descs()
       \             \  /* cpumask_copy(desc->irq_common_data.affinity, affinity); */
        \             -> ...-> desc_smp_init()
         ->request_threaded_irq()
            \
             ->__setup_irq()
               \  \
                \  ->irq_startup()->msi_domain_activate()
                 \      \
                  \      ->irq_enable()->pci_msi_unmask_irq()
                   \
                    -->setup_affinity()
                       \    \
                        \    -->if (irqd_affinity_is_managed(&desc->irq_data))
                         \           set = desc->irq_common_data.affinity;
                          \          cpumask_and(mask, cpu_online_mask, set);
                           \
                            -->irq_do_set_affinity()
                                   \
                                    -->msi_domain_set_affinity()
                                          \   /* Actual setting affinity*/
                                           -->__pci_write_msi_msg()
```

https://www.kernel.org/doc/html/latest/core-api/irq/irq-affinity.html
https://lore.kernel.org/qemu-devel/1554819296-14960-1-git-send-email-ann.zhuangyanying@huawei.com/?spm=ata.21736010.0.0.38547536YAcnc4

不能pass-thru MSI-X table

The format of the Message Address Register (lower 32-bits) is shown in Figure 10-24.

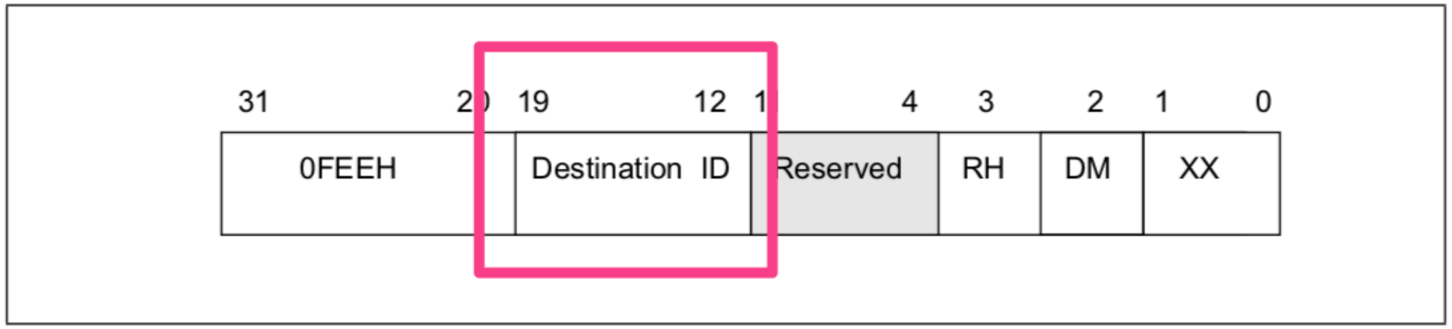
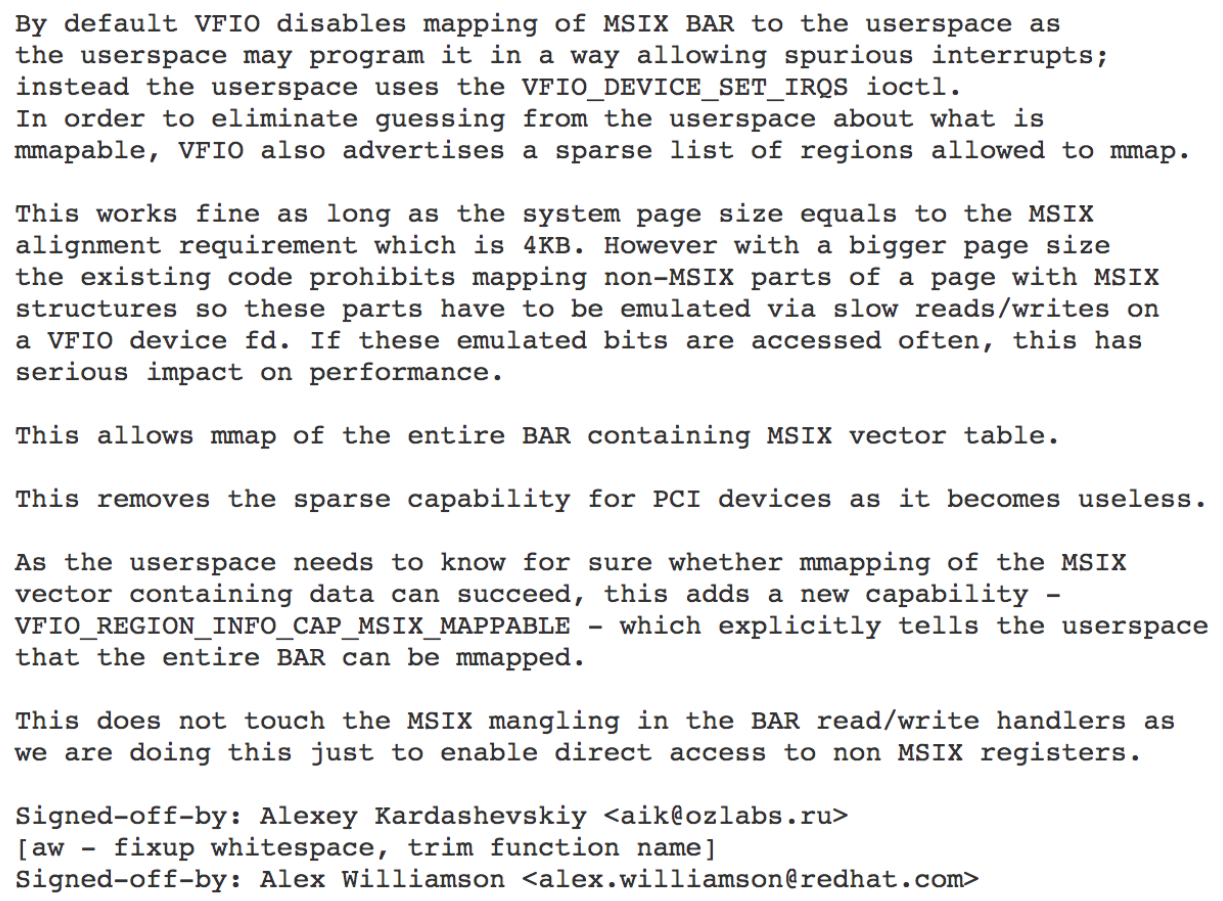


Figure 10-24. Layout of the MSI Message Address Register

虚拟机的Destination ID与物理的Destination ID
必须隔离好，否则会有安全漏洞！

vfio-pci: Allow mapping MSIX BAR



```
By default VFIO disables mapping of MSIX BAR to the userspace as
the userspace may program it in a way allowing spurious interrupts;
instead the userspace uses the VFIO_DEVICE_SET_IRQS ioctl.
In order to eliminate guessing from the userspace about what is
mmapable, VFIO also advertises a sparse list of regions allowed to mmap.

This works fine as long as the system page size equals to the MSIX
alignment requirement which is 4KB. However with a bigger page size
the existing code prohibits mapping non-MSIX parts of a page with MSIX
structures so these parts have to be emulated via slow reads/writes on
a VFIO device fd. If these emulated bits are accessed often, this has
serious impact on performance.

This allows mmap of the entire BAR containing MSIX vector table.

This removes the sparse capability for PCI devices as it becomes useless.

As the userspace needs to know for sure whether mmapping of the MSIX
vector containing data can succeed, this adds a new capability -
VFIO_REGION_INFO_CAP_MSIX_MAPPABLE - which explicitly tells the userspace
that the entire BAR can be mmapped.

This does not touch the MSIX mangling in the BAR read/write handlers as
we are doing this just to enable direct access to non MSIX registers.

Signed-off-by: Alexey Kardashevskiy <aik@ozlabs.ru>
[aw - fixup whitespace, trim function name]
Signed-off-by: Alex Williamson <alex.williamson@redhat.com>
```

https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=a32295c612c57990d17fb0f41e7134394b2f35f6