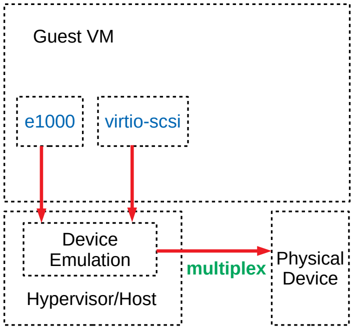
To Trap or Not To Trap: The PCI Passthrough

Dongli Zhang

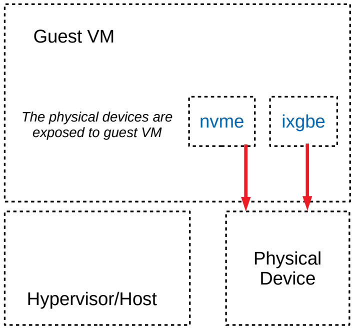
January 6, 2021

What is PCI passthrough?

- To assign full and direct access of PCI device to VM



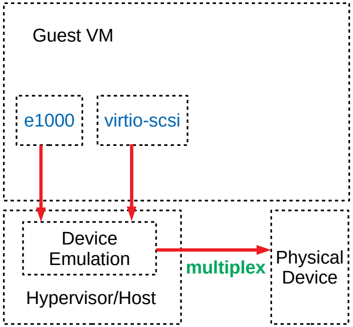
Device Emulation & Pavavirtualized Device



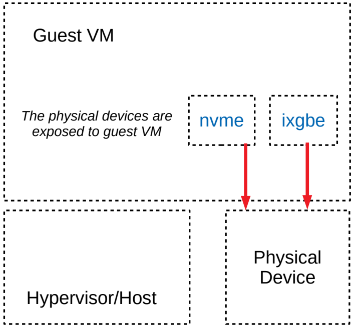
PCI Device (PF/VF) Passthrough

What is PCI passthrough?

- To assign full and direct access of PCI device to VM
- How to assign bugs related to NVMe PCI passthrough?



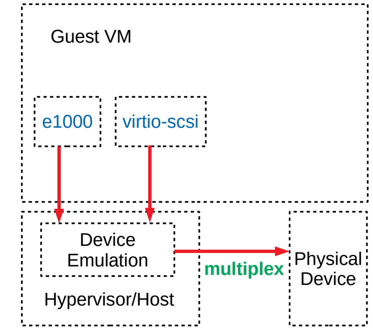
Device Emulation & Pavavirtualized Device



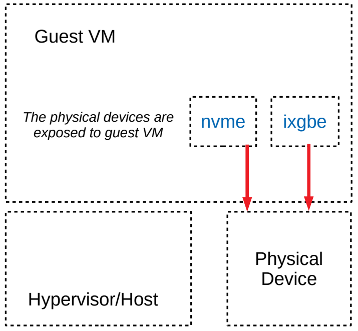
PCI Device (PF/VF) Passthrough

What is PCI passthrough?

- To assign full and direct access of PCI device to VM
- How to assign bugs related to NVMe PCI passthrough?
  - The NVMe developers may blame virtualization developers



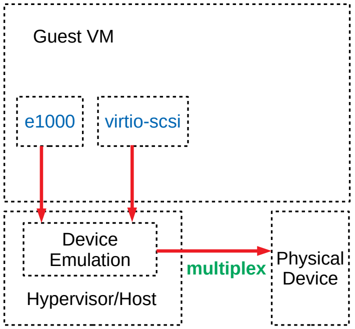
Device Emulation & Pavavirtualized Device



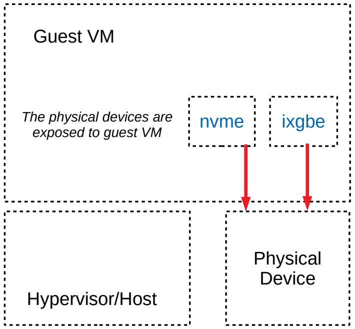
PCI Device (PF/VF) Passthrough

What is PCI passthrough?

- To assign full and direct access of PCI device to VM
- How to assign bugs related to NVMe PCI passthrough?
  - The NVMe developers may blame virtualization developers
  - The virtualization developers may blame hardware/firmware



Device Emulation & Pavavirtualized Device



PCI Device (PF/VF) Passthrough

KVM PCI Passthrough

STEP 1: *To unbind* **01:00.0** *from* **NVMe** *driver and register to* **vfio-pci** *driver:*

01:00.0 Non-Volatile memory controller: Intel Corporation Device f1a6 (rev 03)

```
host# echo 0000:01:00.0 > /sys/bus/pci/devices/0000\:01\:00.0/driver/unbind
host# lspci -ns 0000:01:00.0
01:00.0 0108: 8086:f1a6 (rev 03)
host# echo "8086 f1a6" > /sys/bus/pci/drivers/vfio-pci/new_id
```

KVM PCI Passthrough

STEP 1: *To unbind* **01:00.0** *from* **NVMe** *driver and register to* **vfio-pci** *driver:*

01:00.0 Non-Volatile memory controller: Intel Corporation Device f1a6 (rev 03)

```
host# echo 0000:01:00.0 > /sys/bus/pci/devices/0000\:01\:00.0/driver/unbind
host# lspci -ns 0000:01:00.0
01:00.0 0108: 8086:f1a6 (rev 03)
host# echo "8086 f1a6" > /sys/bus/pci/drivers/vfio-pci/new_id
```

STEP 2: *To passthrough the VFIO-managed NVMe to QEMU/KVM VM:*

```
# qemu-system-x86_64 -machine accel=kvm -vnc :0 -serial stdio -smp 4 -m 4096M \
-net nic -net user,hostfwd=tcp::5022-:22 \
-hda /home/user/img/boot.qcow2 \
-device vfio-pci,host=0000:01:00.0
```

Xen PCI Passthrough

STEP 1: *To unbind* **02:10.0** *from* **igb** *driver and register to* **xen-pciback** *driver:*

02:10.0 Ethernet controller: Intel Corporation I350 Gigabit Network Connection

```
host# echo 0000:02:10.0 > /sys/bus/pci/devices/0000\:02\:10.0/driver/unbind
host# echo 0000:02:10.0 > /sys/bus/pci/drivers/pciback/new_slot
host# echo 0000:02:10.0 > /sys/bus/pci/drivers/pciback/bind
```

Xen PCI Passthrough

STEP 1: *To unbind* **02:10.0** *from* **igb** *driver and register to* **xen-pciback** *driver:*

02:10.0 Ethernet controller: Intel Corporation I350 Gigabit Network Connection

```
host# echo 0000:02:10.0 > /sys/bus/pci/devices/0000\:02\:10.0/driver/unbind
host# echo 0000:02:10.0 > /sys/bus/pci/drivers/pciback/new_slot
host# echo 0000:02:10.0 > /sys/bus/pci/drivers/pciback/bind
```

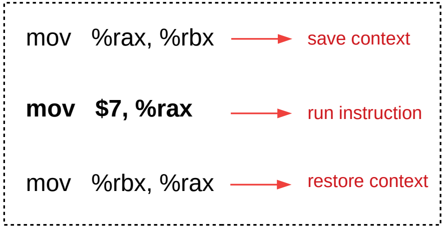
STEP 2: *To append below to Xen HVM config file:*

```
pci = [ '02:10.0' ]
```

Virtualization Demo

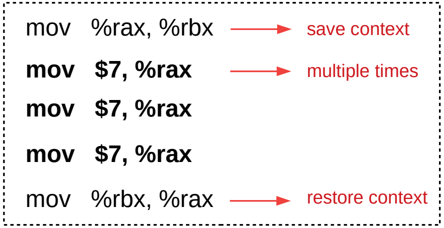
The virtualization demo for a new instruction set

- The registers (only one): **RAX**
- The instructions (only one): **mov $7, %rax**



```
mov   %rax, %rbx
mov   $7, %rax
mov   %rbx, %rax
```

demo 1: run instruction once



```
mov   %rax, %rbx
mov   $7, %rax
mov   $7, %rax
mov   $7, %rax
mov   %rbx, %rax
```

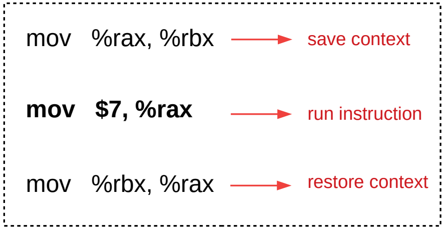
demo 2: run instruction multiple times

Virtualization Demo

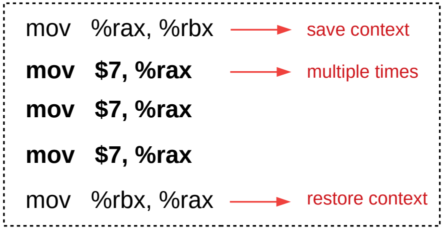
The virtualization demo for a new instruction set

- The registers (only one): **RAX**
- The instructions (only one): **mov $7, %rax**

How about privileged instructions?



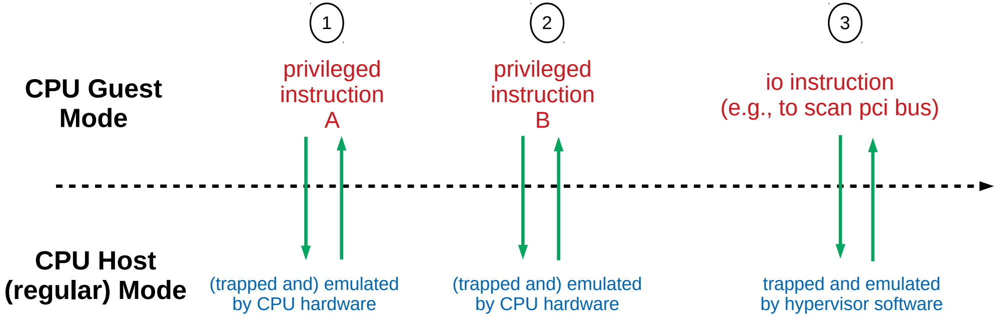
demo 1: run instruction once



demo 2: run instruction multiple times

CPU Virtualization

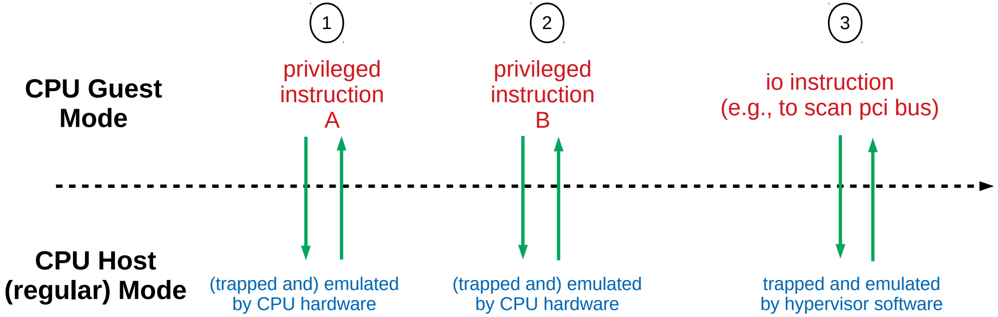
- The CPU is divided into **guest mode** and regular **host mode**
- Some privileged instructions are (trapped to and) emulated by **CPU host mode**
- Some privileged instructions are trapped to and emulated by **hypervisor software**



1
2
3
CPU Guest Mode
privileged instruction A
privileged instruction B
io instruction (e.g., to scan pci bus)
CPU Host (regular) Mode
(trapped and) emulated by CPU hardware
(trapped and) emulated by CPU hardware
trapped and emulated by hypervisor software

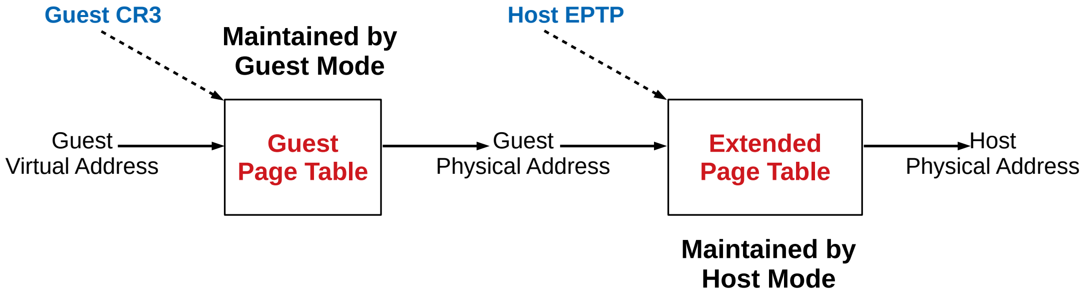
CPU Virtualization

- The CPU is divided into **guest mode** and regular **host mode**
- Some privileged instructions are (trapped to and) emulated by **CPU host mode**
- Some privileged instructions are trapped to and emulated by **hypervisor software**
- How about memory virtualization?


1
2
3
CPU Guest Mode
privileged instruction A
privileged instruction B
io instruction (e.g., to scan pci bus)
CPU Host (regular) Mode
(trapped and) emulated by CPU hardware
(trapped and) emulated by CPU hardware
trapped and emulated by hypervisor software

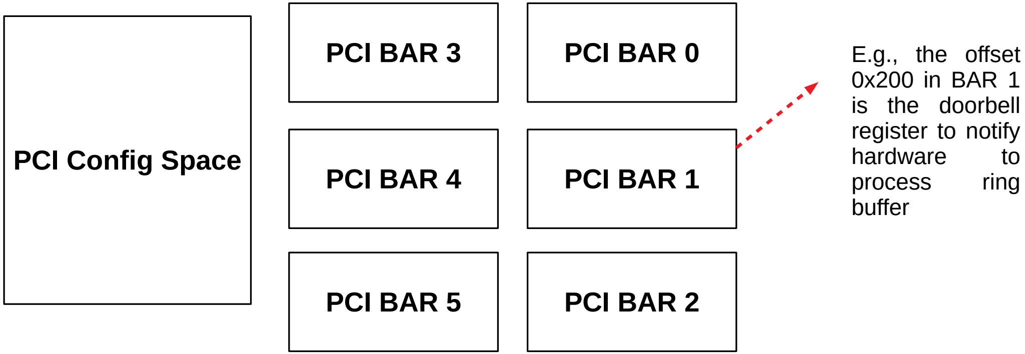
Memory Virtualization

- **Guest Page Table (CR3)**: Guest Virtual Address to Guest Physical Address
- **Host Page Table (EPTP)**: Guest Physical Address to Host Physical Address
- To set EPT entry as non-read, non-write or non-exec can trap guest VM memory access



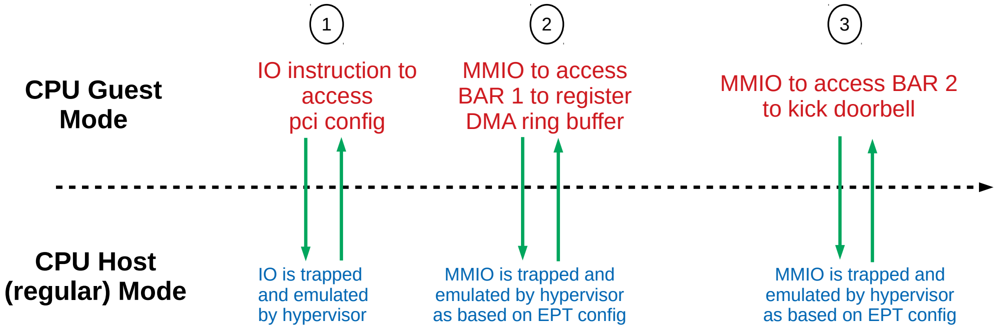
PCI Device Emulation 1/2

- PCI: **config space**, **capabilities** and **bar**
- The PCI Bar is usually to register DMA ring buffer, configure MSI-X and kick doorbell


PCI Config Space
PCI BAR 3
PCI BAR 0
PCI BAR 4
PCI BAR 1
PCI BAR 5
PCI BAR 2
E.g., the offset 0x200 in BAR 1 is the doorbell register to notify hardware to process ring buffer

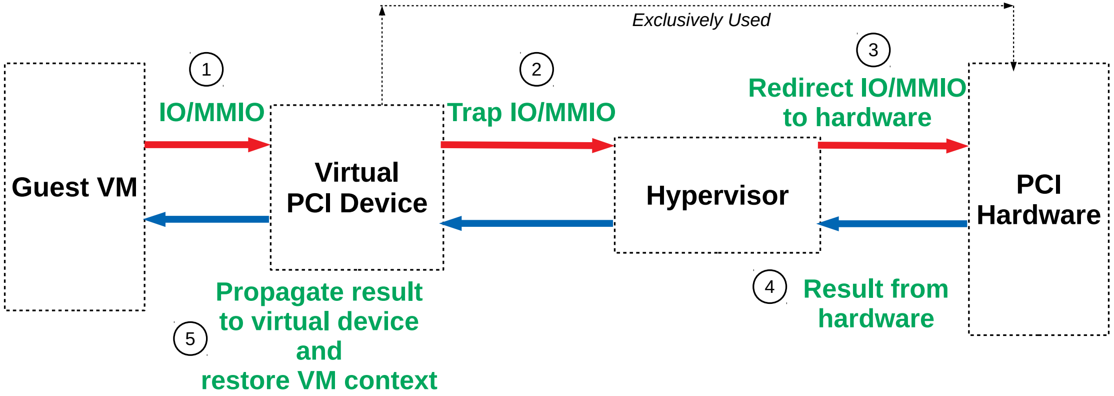
PCI Device Emulation 2/2

- EPT is configured on purpose to trap **MMIO** bar access
- PCI (config/bar) access by **IO** instruction is trapped to hypervisor
- Both **IO** and **MMIO** are emulated by hypervisor


1
2
3
CPU Guest Mode
IO instruction to access pci config
MMIO to access BAR 1 to register DMA ring buffer
MMIO to access BAR 2 to kick doorbell
CPU Host (regular) Mode
IO is trapped and emulated by hypervisor
MMIO is trapped and emulated by hypervisor as based on EPT config
MMIO is trapped and emulated by hypervisor as based on EPT config

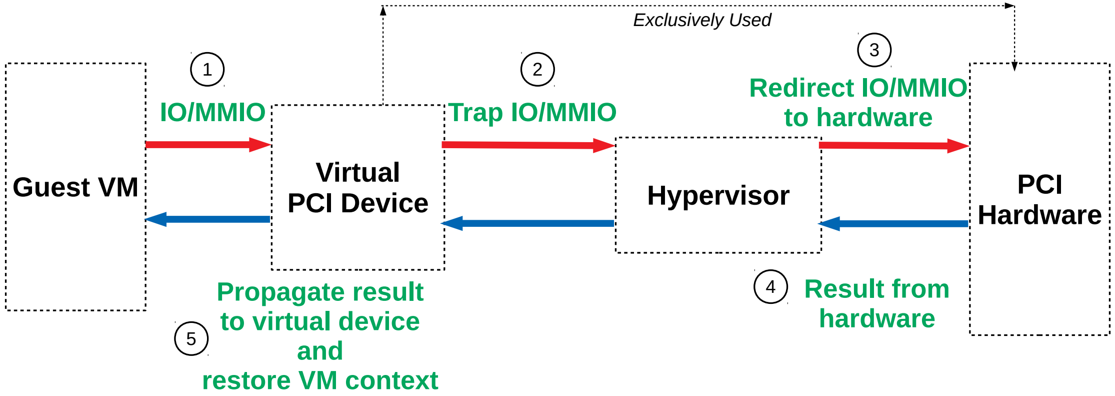
PCI Passthrough 1/4: Plan A

- Hypervisor as intermediate layer:
  - Access to PCI config/bar is always trapped to hypervisor
  - Hypervisor access PCI device and propagate result to VM



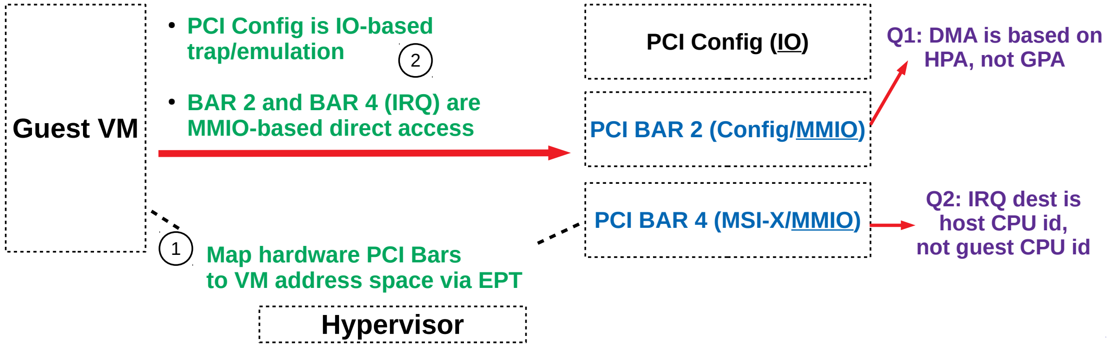
PCI Passthrough 1/4: Plan A

- Hypervisor as intermediate layer:
    - Access to PCI config/bar is always trapped to hypervisor
    - Hypervisor access PCI device and propagate result to VM
- Cons: too many **traps** to hypervisor!



PCI Passthrough 2/4: Plan B

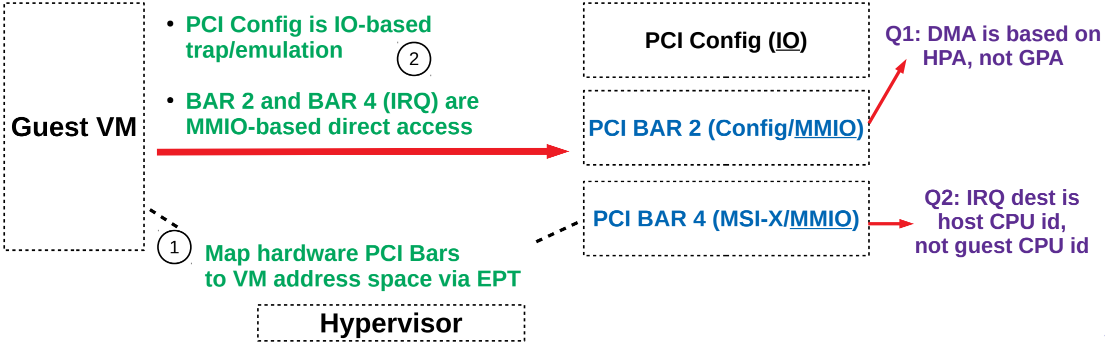
- Map all MMIO PCI BARs to VM address space via EPT
- Access to PCI config (and IO-based BAR) is trapped to hypervisor
- The VM has direct MMIO access to PCI BARs to avoid **trap**



Guest VM
• PCI Config is IO-based trap/emulation
2
• BAR 2 and BAR 4 (IRQ) are MMIO-based direct access
PCI Config (IO)
PCI BAR 2 (Config/MMIO)
PCI BAR 4 (MSI-X/MMIO)
Q1: DMA is based on HPA, not GPA
Q2: IRQ dest is host CPU id, not guest CPU id
1
Map hardware PCI Bars to VM address space via EPT
Hypervisor

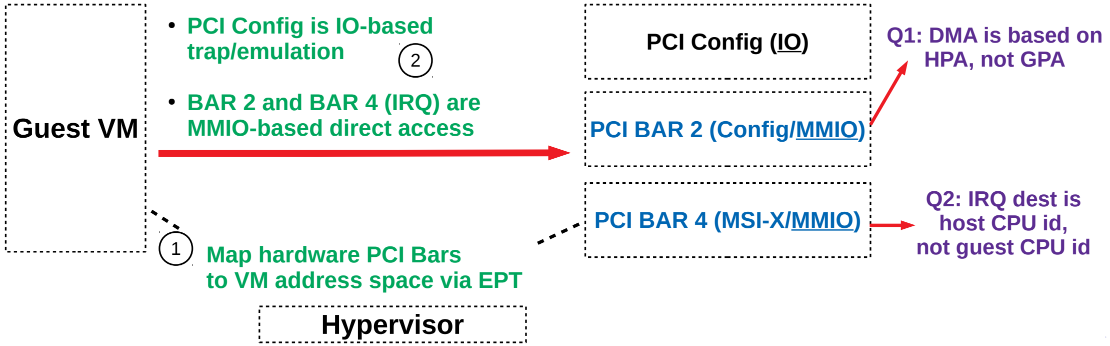
PCI Passthrough 2/4: Plan B

- Map all MMIO PCI BARs to VM address space via EPT
- Access to PCI config (and IO-based BAR) is trapped to hypervisor
- The VM has direct MMIO access to PCI BARs to avoid **trap**
- Q1: DMA is based on HPA, NOT GPA!


Guest VM
• PCI Config is IO-based trap/emulation
2
• BAR 2 and BAR 4 (IRQ) are MMIO-based direct access
1
Map hardware PCI Bars to VM address space via EPT
Hypervisor
PCI Config (IO)
PCI BAR 2 (Config/MMIO)
PCI BAR 4 (MSI-X/MMIO)
Q1: DMA is based on HPA, not GPA
Q2: IRQ dest is host CPU id, not guest CPU id

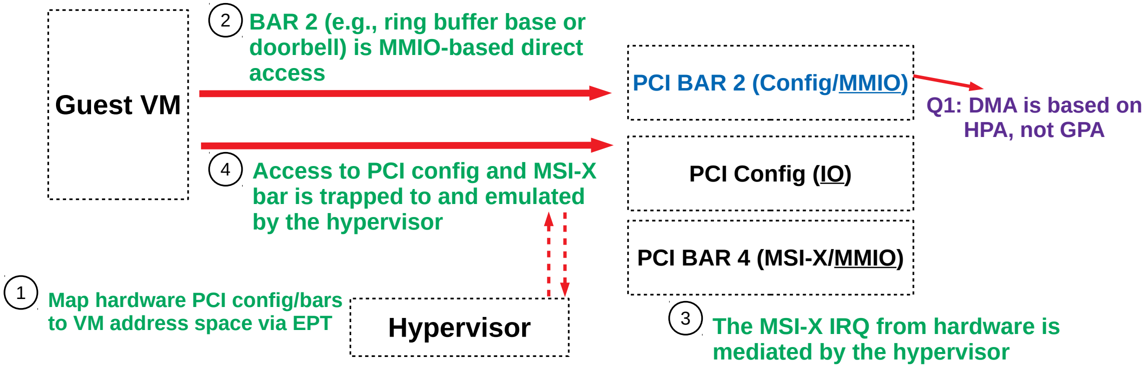
PCI Passthrough 2/4: Plan B

- Map all MMIO PCI BARs to VM address space via EPT
- Access to PCI config (and IO-based BAR) is trapped to hypervisor
- The VM has direct MMIO access to PCI BARs to avoid **trap**
- Q1: DMA is based on HPA, NOT GPA!
- Q2: MSI-X IRQ is based on host CPU ID, NOT guest CPU ID!


Guest VM
• PCI Config is IO-based trap/emulation
2
• BAR 2 and BAR 4 (IRQ) are MMIO-based direct access
1
Map hardware PCI Bars to VM address space via EPT
Hypervisor
PCI Config (IO)
PCI BAR 2 (Config/MMIO)
PCI BAR 4 (MSI-X/MMIO)
Q1: DMA is based on HPA, not GPA
Q2: IRQ dest is host CPU id, not guest CPU id

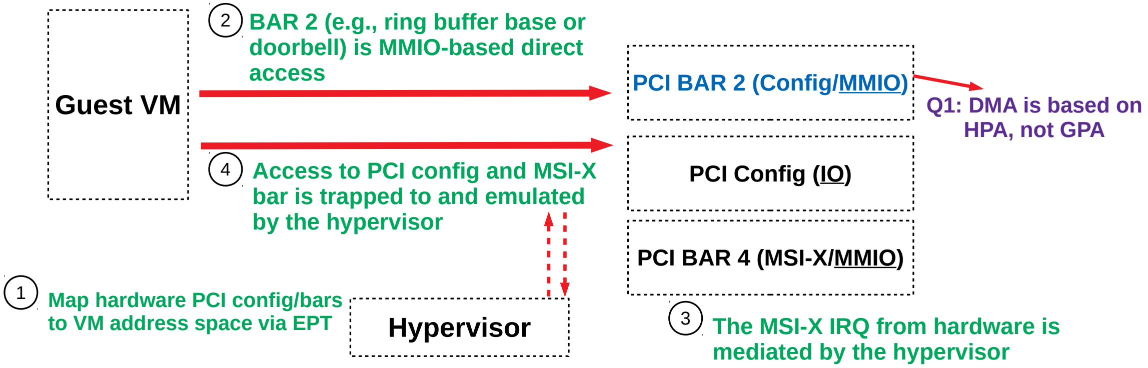
PCI Passthrough 3/4: Plan C

- Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space
- Access to PCI config (and IO-based BAR) and MSI-X table is trapped to hypervisor
- Redirect IRQ from PCI device to VM via hypervisor


Guest VM
2
BAR 2 (e.g., ring buffer base or doorbell) is MMIO-based direct access
PCI BAR 2 (Config/MMIO)
Q1: DMA is based on HPA, not GPA
PCI Config (IO)
4
Access to PCI config and MSI-X bar is trapped to and emulated by the hypervisor
PCI BAR 4 (MSI-X/MMIO)
1
Map hardware PCI config/bars to VM address space via EPT
Hypervisor
3
The MSI-X IRQ from hardware is mediated by the hypervisor

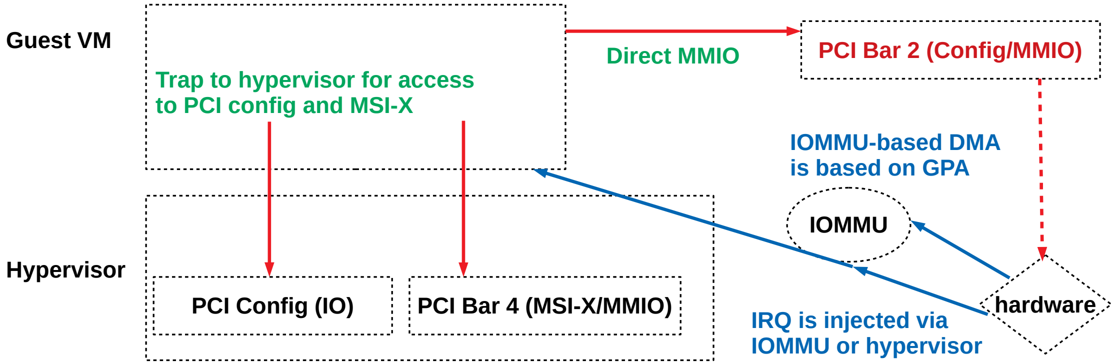
PCI Passthrough 3/4: Plan C

- Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space
- Access to PCI config (and IO-based BAR) and MSI-X table is trapped to hypervisor
- Redirect IRQ from PCI device to VM via hypervisor
- Q1: DMA is based on HPA, NOT GPA! (Device Isolation across VMs)


Guest VM
2 BAR 2 (e.g., ring buffer base or doorbell) is MMIO-based direct access
PCI BAR 2 (Config/MMIO)
Q1: DMA is based on HPA, not GPA
PCI Config (IO)
4 Access to PCI config and MSI-X bar is trapped to and emulated by the hypervisor
PCI BAR 4 (MSI-X/MMIO)
1 Map hardware PCI config/bars to VM address space via EPT
Hypervisor
3 The MSI-X IRQ from hardware is mediated by the hypervisor

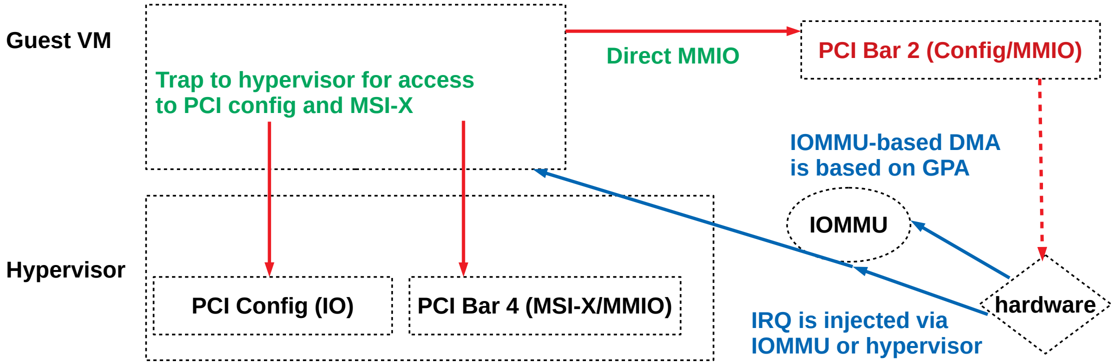
PCI Passthrough 4/4: Plan D

- Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space


Guest VM
Trap to hypervisor for access to PCI config and MSI-X
Direct MMIO
PCI Bar 2 (Config/MMIO)
IOMMU-based DMA is based on GPA
IOMMU
Hypervisor
PCI Config (IO)
PCI Bar 4 (MSI-X/MMIO)
hardware
IRQ is injected via IOMMU or hypervisor

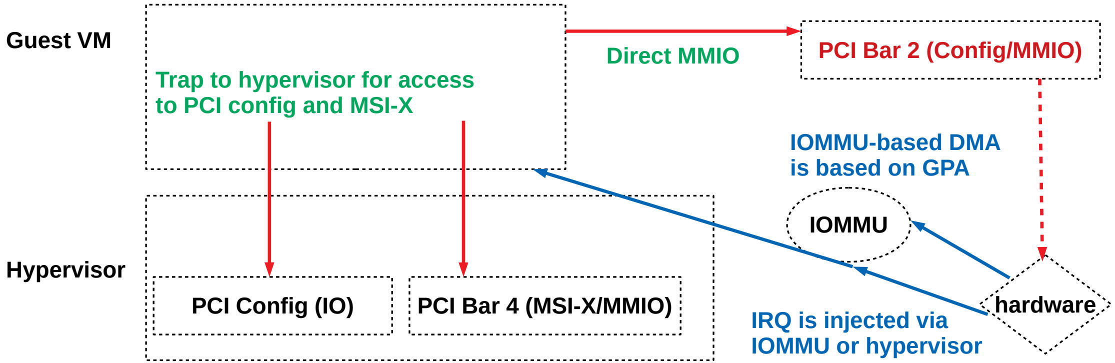
PCI Passthrough 4/4: Plan D

- Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space
- Access to PCI config (and IO-based BAR) and MSI-X table is trapped to hypervisor


Guest VM
Trap to hypervisor for access to PCI config and MSI-X
Direct MMIO
PCI Bar 2 (Config/MMIO)
IOMMU-based DMA is based on GPA
IOMMU
Hypervisor
PCI Config (IO)
PCI Bar 4 (MSI-X/MMIO)
hardware
IRQ is injected via IOMMU or hypervisor

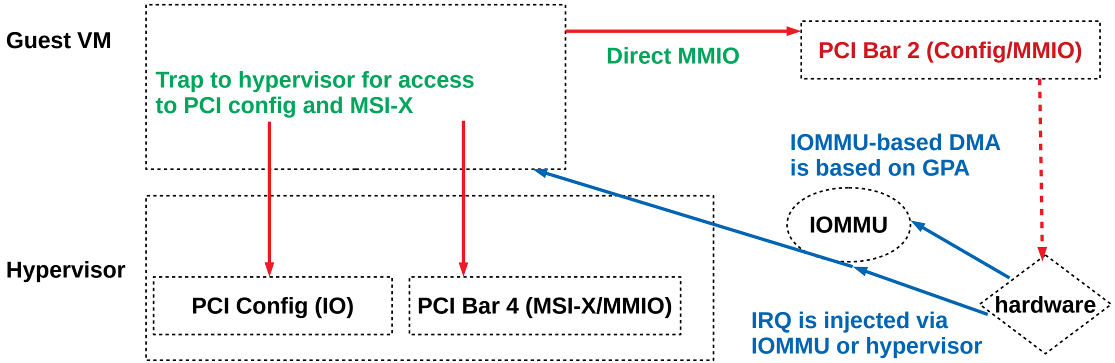
PCI Passthrough 4/4: Plan D

- Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space
- Access to PCI config (and IO-based BAR) and MSI-X table is trapped to hypervisor
- Redirect DMA GPA to HPA via IOMMU


Guest VM
Trap to hypervisor for access to PCI config and MSI-X
Direct MMIO
PCI Bar 2 (Config/MMIO)
IOMMU-based DMA is based on GPA
IOMMU
Hypervisor
PCI Config (IO)
PCI Bar 4 (MSI-X/MMIO)
hardware
IRQ is injected via IOMMU or hypervisor

PCI Passthrough 4/4: Plan D

- Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space
- Access to PCI config (and IO-based BAR) and MSI-X table is trapped to hypervisor
- Redirect DMA GPA to HPA via IOMMU
- Redirect IRQ from PCI device to VM via hypervisor/IOMMU


Guest VM
Trap to hypervisor for access to PCI config and MSI-X
Direct MMIO
PCI Bar 2 (Config/MMIO)
IOMMU-based DMA is based on GPA
IOMMU
Hypervisor
PCI Config (IO)
PCI Bar 4 (MSI-X/MMIO)
hardware
IRQ is injected via IOMMU or hypervisor

KVM PCI Passthrough Memory Mapping 1/2

host# lspci -vv -s 03:00.3

```
03:00.3 Ethernet controller: Intel Corporation Ethernet Controller XL710 for 40GbE QSFP+ (rev 02)
        Subsystem: XXXXXX 10 Gb/40 Gb Ethernet Adapter
        Physical Slot: 3
        Control: I/O- Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR+ FastB2B- DisINTx+
        Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- INTx-
        Latency: 0, Cache Line Size: 32 bytes
        Interrupt: pin A routed to IRQ 24
        NUMA node: 0
        Region 0: Memory at c1800000 (64-bit, prefetchable) [size=8M]
        Region 3: Memory at c4000000 (64-bit, prefetchable) [size=32K]
… …
… …
        Capabilities: [70] MSI-X: Enable+ Count=129 Masked-
                Vector table: BAR=3 offset=00000000
                PBA: BAR=3 offset=00001000
… …
… …
```

$\longrightarrow$ **Bar 0 is mapped**

$\longrightarrow$ **BAR 3 MSI-X vector table is trapped**

$\searrow$ **BAR 3 MSI-X PBA table is mapped**

KVM PCI Passthrough Memory Mapping 2/2

(qemu) info mtree -f

```
FlatView #2
… …
… …
  00000000fe000000-00000000fe7fffff (prio 0, ramd): 0000:03:00.3 BAR 0 mmaps[0]
  00000000fe800000-00000000fe80080f (prio 0, i/o): msix-table
  00000000fe800810-00000000fe800fff (prio 0, i/o): 0000:03:00.3 BAR 3 @0000000000000810
  00000000fe801000-00000000fe807fff (prio 0, ramd): 0000:03:00.3 BAR 3 mmaps[0]
… …
… …
```

MSI-X Pending Bit Array (PBA)

- **ramd** : PCI bar/region is mapped to VM address space
- **i/o** : access is trapped to and emulated by hypervisor

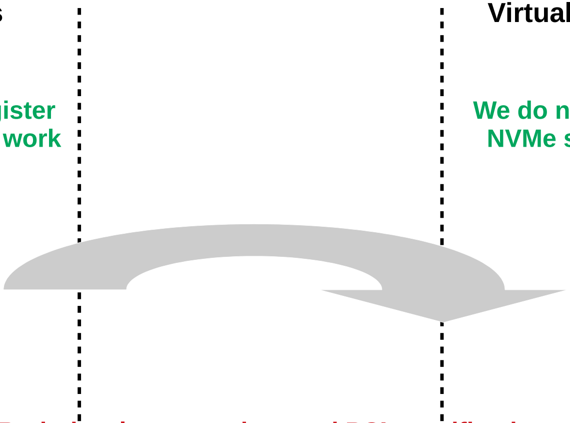
Bridge the Gap

NVMe Developers

Virtualization Developers

To kick the doorbell register for IO queue 3 does not work

We do not understand NVMe spec/manual!



- **Both developers understand PCI specification**
- **Write to BAR 0 offset 0x120 does not take effect**

Take-Home Message

- Virtualization is **trap and emulation** (explicitly by software or implicitly by hardware)

Take-Home Message

- Virtualization is **trap and emulation** (explicitly by software or implicitly by hardware)
- PCI Passthrough is **to minimize the cost that VM accesses PCI device**, via:
    - Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space via EPT

Take-Home Message

- Virtualization is **trap and emulation** (explicitly by software or implicitly by hardware)
- PCI Passthrough is **to minimize the cost that VM accesses PCI device**, via:
    - Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space via EPT
    - Access to PCI config (and IO-based BAR) and MSI-X table is trapped to and emulated by hypervisor

Take-Home Message

- Virtualization is **trap and emulation** (explicitly by software or implicitly by hardware)
- PCI Passthrough is **to minimize the cost that VM accesses PCI device**, via:
  - Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space via EPT
  - Access to PCI config (and IO-based BAR) and MSI-X table is trapped to and emulated by hypervisor
  - Redirect DMA GPA to HPA via IOMMU

Take-Home Message

- Virtualization is **trap and emulation** (explicitly by software or implicitly by hardware)
- PCI Passthrough is **to minimize the cost that VM accesses PCI device**, via:
  - Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space via EPT
  - Access to PCI config (and IO-based BAR) and MSI-X table is trapped to and emulated by hypervisor
  - Redirect DMA GPA to HPA via IOMMU
  - Redirect IRQ from PCI device to VM via hypervisor/IOMMU

Take-Home Message

- Virtualization is **trap and emulation** (explicitly by software or implicitly by hardware)
- PCI Passthrough is **to minimize the cost that VM accesses PCI device**, via:
    - Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space via EPT
    - Access to PCI config (and IO-based BAR) and MSI-X table is trapped to and emulated by hypervisor
    - Redirect DMA GPA to HPA via IOMMU
    - Redirect IRQ from PCI device to VM via hypervisor/IOMMU
- To bridge the gap (between NVMe/Ethernet/IB and virtualization) via PCI spec

Take-Home Message

- Virtualization is **trap and emulation** (explicitly by software or implicitly by hardware)
- PCI Passthrough is **to minimize the cost that VM accesses PCI device**, via:
  - Map only some PCI BARs (e.g., ring buffer base or doorbell) to VM address space via EPT
  - Access to PCI config (and IO-based BAR) and MSI-X table is trapped to and emulated by hypervisor
  - Redirect DMA GPA to HPA via IOMMU
  - Redirect IRQ from PCI device to VM via hypervisor/IOMMU
- To bridge the gap (between NVMe/Ethernet/IB and virtualization) via PCI spec
- Congratulations! You have learned about to implement virtualization demo from scratch!