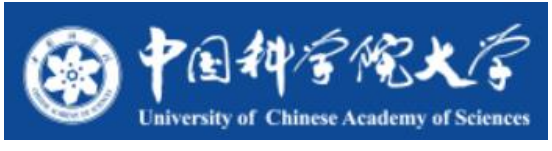
ISCAS
中国科学院软件研究所
Institute of Software,Chinese Academy of Sciences
中国科学院大学
University of Chinese Academy of Sciences

I/O虚拟化

改编声明

- 本课程教学及PPT内容基于**上海交通大学并行与分布式系统研究所**发布的操作系统课程修改，原课程官网：
  - https://ipads.se.sjtu.edu.cn/courses/os/index.shtml
- 本课程修改人为**中国科学院软件研究所**，用于国科大操作系统课程教学。

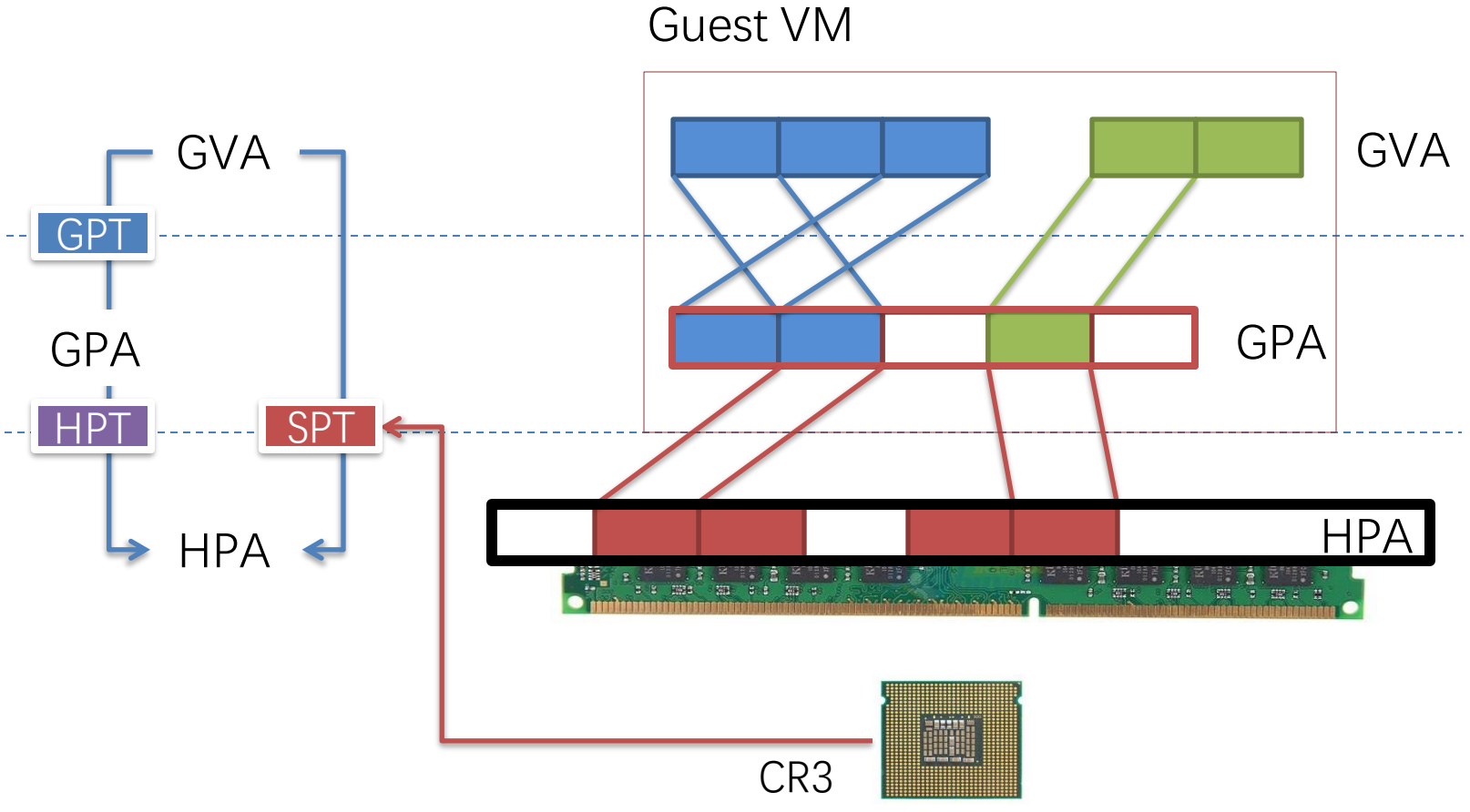


三种地址

- **客户虚拟地址(Guest Virtual Address, GVA)**
  - 虚拟机内进程使用的虚拟地址
- **客户物理地址(Guest Physical Address, GPA)**
  - 虚拟机内使用的“假”物理地址
- **主机物理地址(Host Physical Address, HPA)**
  - 真实寻址的物理地址
  - GPA需要翻译成HPA才能访存

VMM管理

影子页表



2、Direct Paging (Para-virtualization)

- **Modify the guest OS**
  - No GPA is needed, just GVA and HPA
  - Guest OS directly manages its HPA space
  - Use *hypercall* to let the VMM update the page table
  - The hardware CR3 will point to guest page table
- **VMM will check all the page table operations**
  - The guest page tables are read-only to the guest

2、Direct Paging (Para-virtualization)

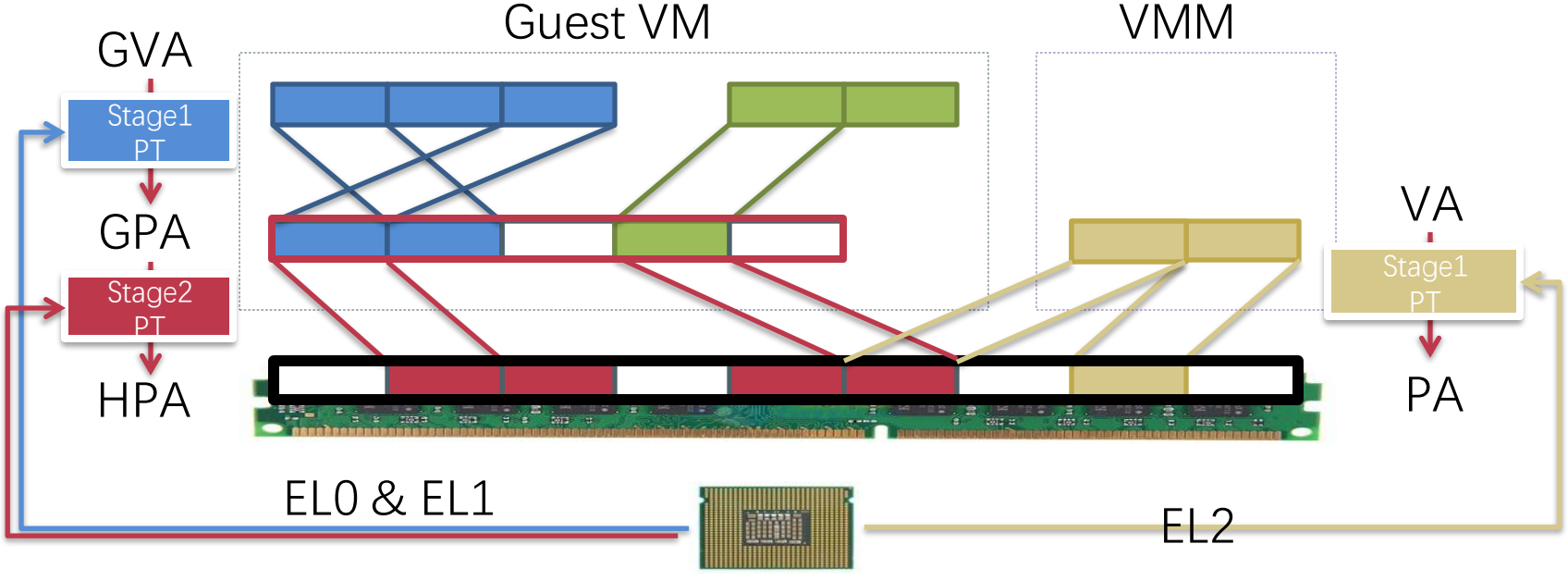
- **Positive**
  - Easy to implement and more clear architecture
  - Better performance: guest can batch to reduce trap
- **Negatives**
  - Not transparent to the guest OS
  - The guest now knows much info, e.g., HPA
    - May use such info to trigger *rowhammer* attacks

3、硬件虚拟化对内存翻译的支持

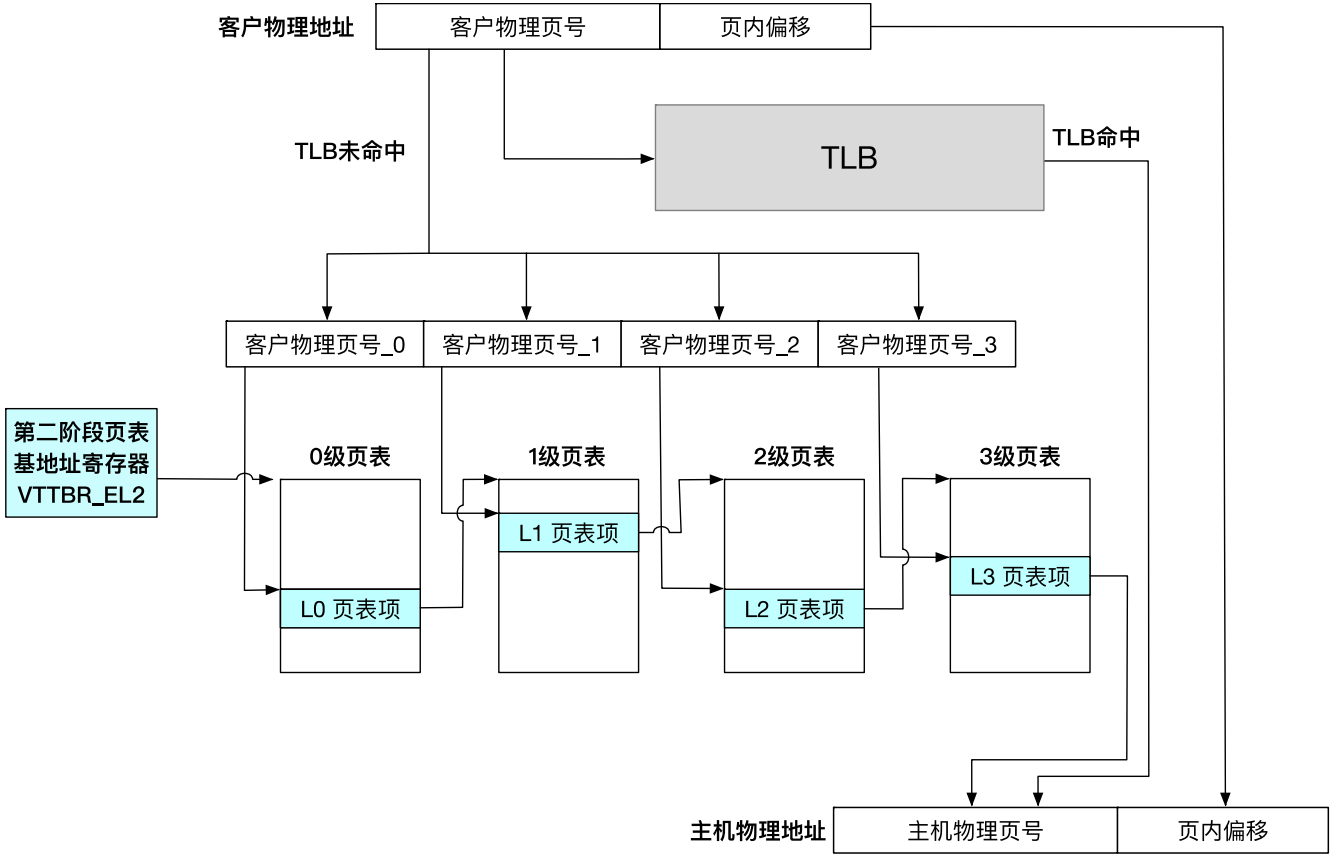
- **Intel VT-x和RISC-V硬件虚拟化都有对应的内存虚拟化**
  - Intel Extended Page Table (EPT)
  - RISC-V Hypervisor Page Table (HPT)
- **新的页表**
  - 将GPA翻译成HPA
  - 此表被VMM直接控制
  - 每一个VM有一个对应的页表

第二阶段页表

- 第一阶段页表：虚拟机内虚拟地址翻译（GVA->GPA）
- 第二阶段页表：虚拟机客户物理地址翻译（GPA->HPA）



第二阶段4级页表

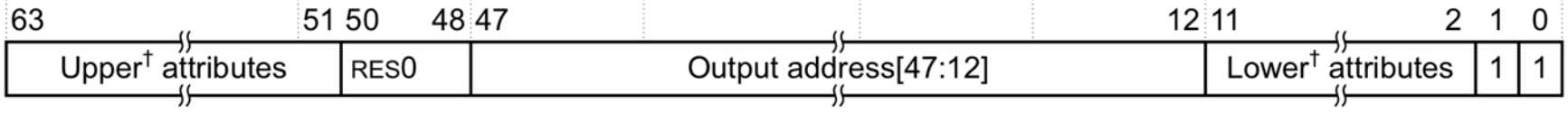

客户物理地址
客户物理页号
页内偏移
TLB未命中
TLB
TLB命中
客户物理页号_0
客户物理页号_1
客户物理页号_2
客户物理页号_3
第二阶段页表
基地址寄存器
VTTBR_EL2
0级页表
1级页表
2级页表
3级页表
L0 页表项
L1 页表项
L2 页表项
L3 页表项
主机物理地址
主机物理页号
页内偏移

VTTBR_EL2

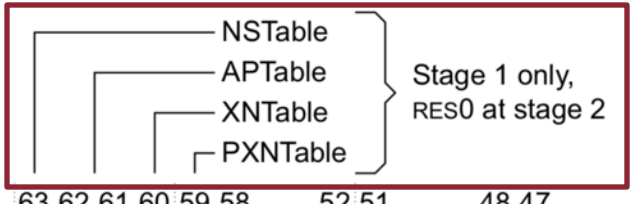
- **存储虚拟机第二阶段页表基地址**
  - 只有1个寄存器：VTTBR_EL2
- **对比第一阶段页表**
  - 有2个页表基地址寄存器：TTBR0_EL1、TTBR1_EL1
- **VMM在调度VM之前需要在VTTBR_EL2中写入此VM的第二阶段页表基地址**
- **第二阶段页表使能**
  - HCR_EL2第0位

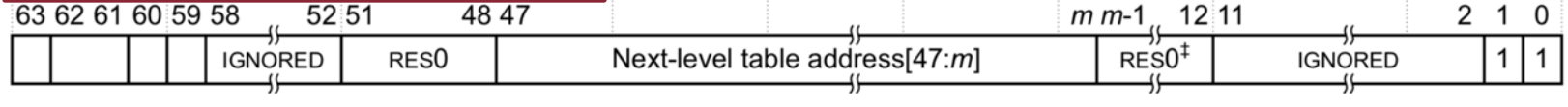
第二阶段页表项

- **第3级页表页中的页表项**
  - 与第一阶段页表完全一致



- **第0-2级页表页中的页表项**
  - 与第一阶段在高位有不同

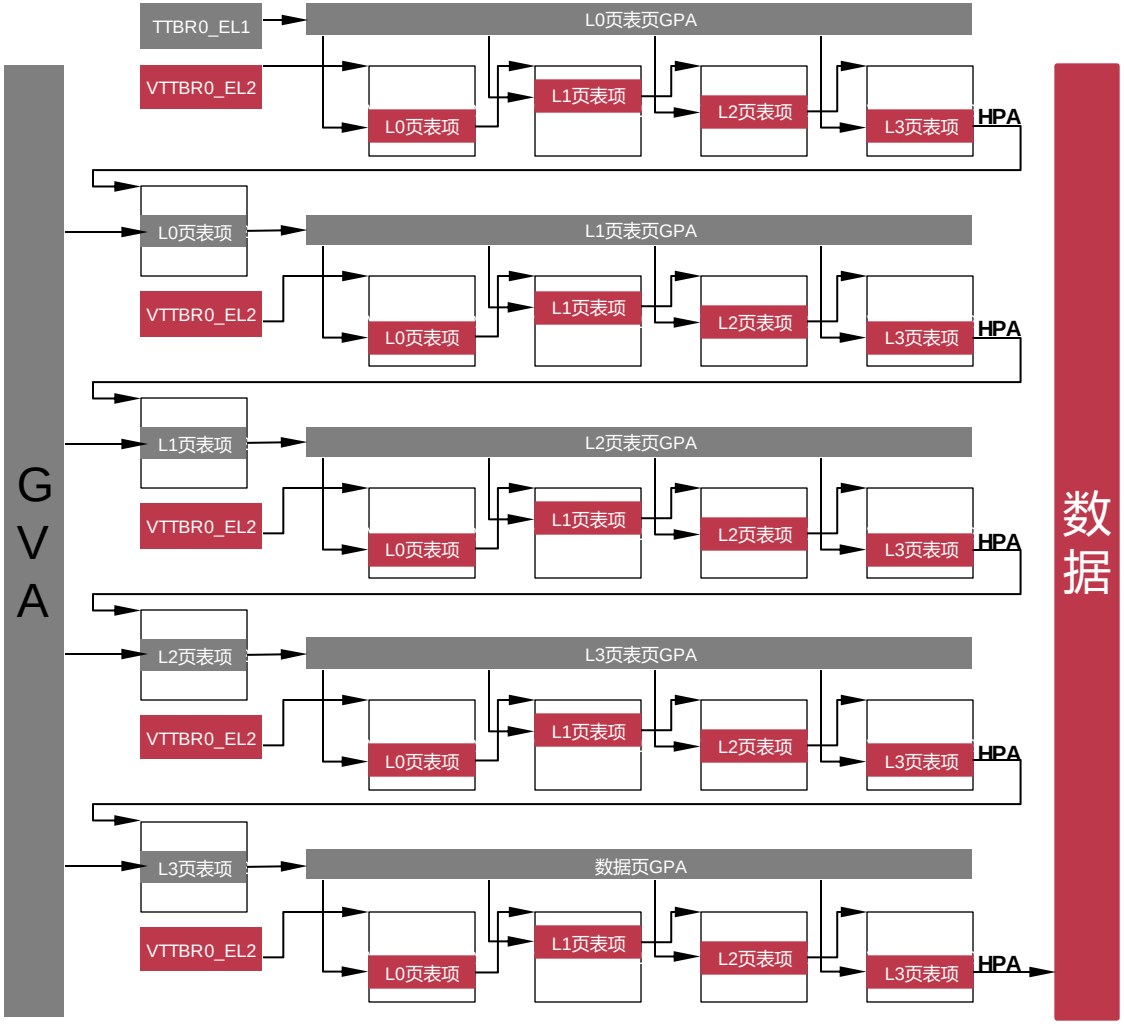




- **NSTable：与TrustZone有关**
- **APTable：读写权限**
- **XNTable：执行权限**
- **PXNTable：特权级别软件的执行权限**

翻译过程

- **总共24次内存访问**
  - 为什么?
  - 25-1
  - 读TTBR0_EL1无需内存访问



TLB：缓存地址翻译结果

- **回顾：TLB不仅可以缓存第一阶段地址翻译结果**
- **TLB也可以第二阶段地址翻译后的结果**
  - 包括第一阶段的翻译结果(GVA->GPA)
  - 包括第二阶段的翻译结果(GPA->HPA)
  - 大大提升GVA->HPA的翻译性能：不需要24次内存访问
- **切换VTTBR_EL2时**
  - 理论上应将前一个VM的TLB项全部刷掉

TLB刷新

- **刷TLB相关指令**
  - 清空全部
    - TLBI VMALLS12E1IS
  - 清空指定GVA
    - TLBI VAE1IS
  - 清空指定GPA
    - TLBI IPAS2E1IS
- **VMID (Virtual Machine IDentifier)**
  - VMM为不同进程分配8/16 VMID，将VMID填写在VTTBR_EL2的高8/16位
  - VMID位数由VTCR_EL2的第19位（VS位）决定
  - 避免刷新上个VM的TLB

如何处理缺页异常

- **两阶段翻译的缺页异常分开处理**
- **第一阶段缺页异常**
  - 直接调用VM的Page fault handler
  - 修改第一阶段页表**不会**引起任何虚拟机下陷
- **第二阶段缺页异常**
  - 虚拟机下陷，直接调用VMM的Page fault handler

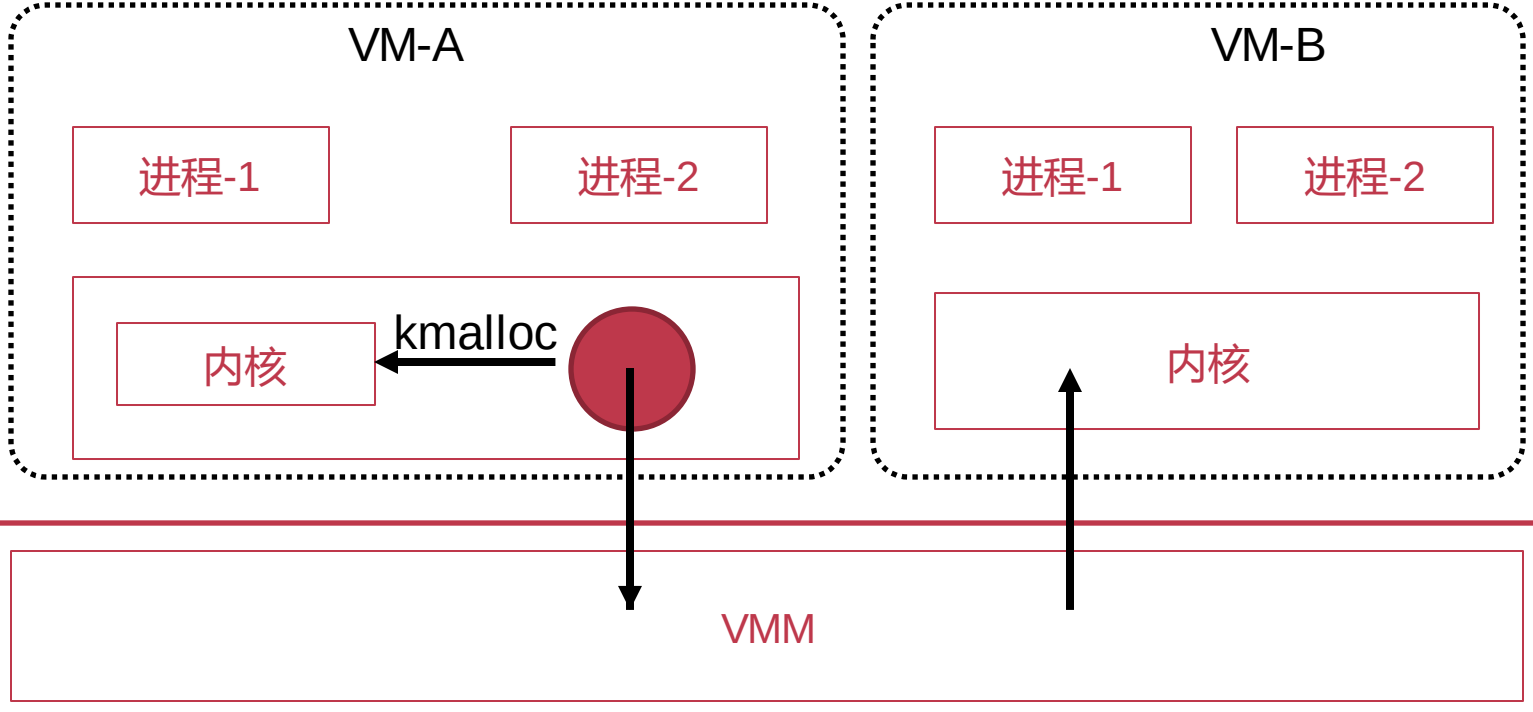
第二阶段页表的优缺点

- **优点**
  - VMM实现简单
  - 不需要捕捉Guest Page Table的更新
  - 减少内存开销：每个VM对应一个页表
- **缺点**
  - TLB miss时性能开销较大

如何实现虚拟机级别的内存换页?

- **具体场景：将虚拟机A的128MB内存转移到虚拟机B中**
  - 虚拟机A对内存的使用较少
  - 虚拟机B对内存需求较大
- **问题**
  - VMM无法识别虚拟机内存的语义
  - 两层内存换页机制
    - VM与VMM的换页机制可能彼此冲突，造成开销。

内存气球机制



I/O Virtualization

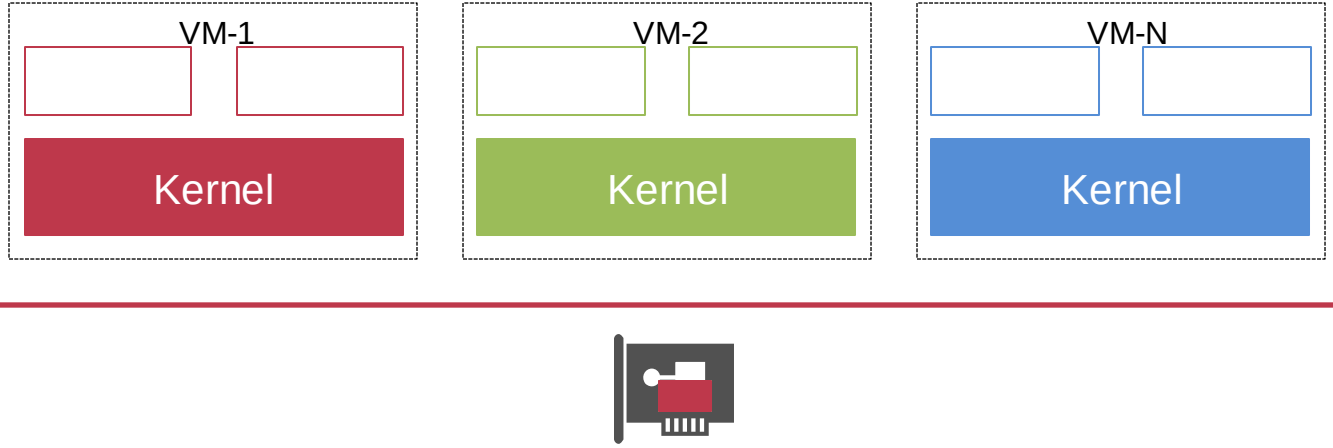
I/O虚拟化

为什么需要IO虚拟化

- **回顾：操作系统内核直接管理外部设备**
  - PIO/MMIO
  - DMA
  - Interrupt
- **如果VM能直接管理物理设备**
  - 会发生什么?

如果VM直接管理物理网卡

- **正确性问题：所有VM都直接访问网卡**
  - 所有VM都有相同的MAC地址、IP地址，无法正常收发网络包
- **安全性问题：恶意VM可以直接读取其他VM的数据**
  - 除了直接读取所有网络包，还可能通过DMA访问其他内存



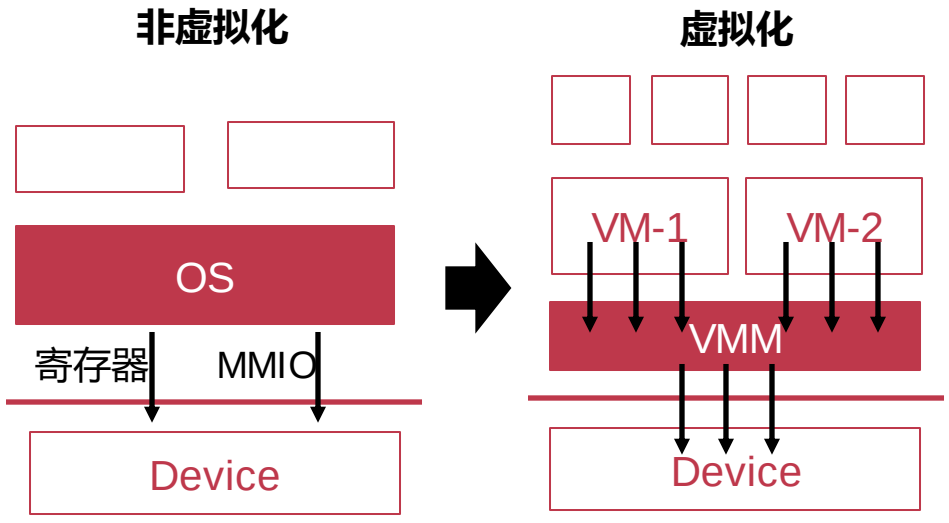
I/O虚拟化的目标

- **为虚拟机提供虚拟的外部设备**
  - 虚拟机正常使用设备
- **隔离不同虚拟机对外部设备的直接访问**
  - 实现I/O数据流和控制流的隔离
- **提高物理设备的利用资源**
  - 多个VM同时使用，可以提高物理设备的资源利用率

怎么实现I/O虚拟化?

- 1、设备模拟（Emulation）
- 2、半虚拟化方式（Para-virtualization）
- 3、设备直通（Pass-through）

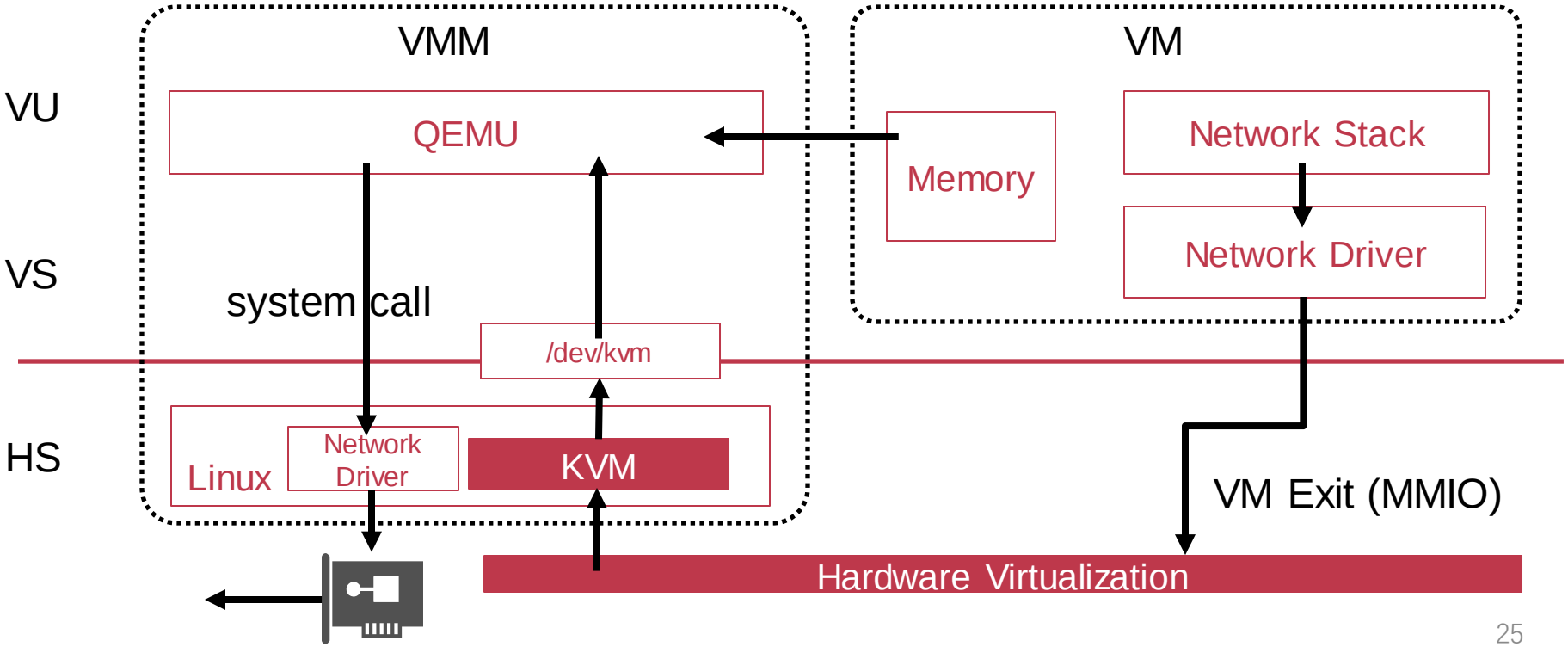
方法1：设备模拟



- **OS与设备交互的硬件接口**
  - 模拟寄存器(中断等)
  - 捕捉MMIO操作
- **硬件虚拟化的方式**
  - 硬件虚拟化捕捉PIO指令
  - MMIO对应内存在第二阶段页表中设置为invalid

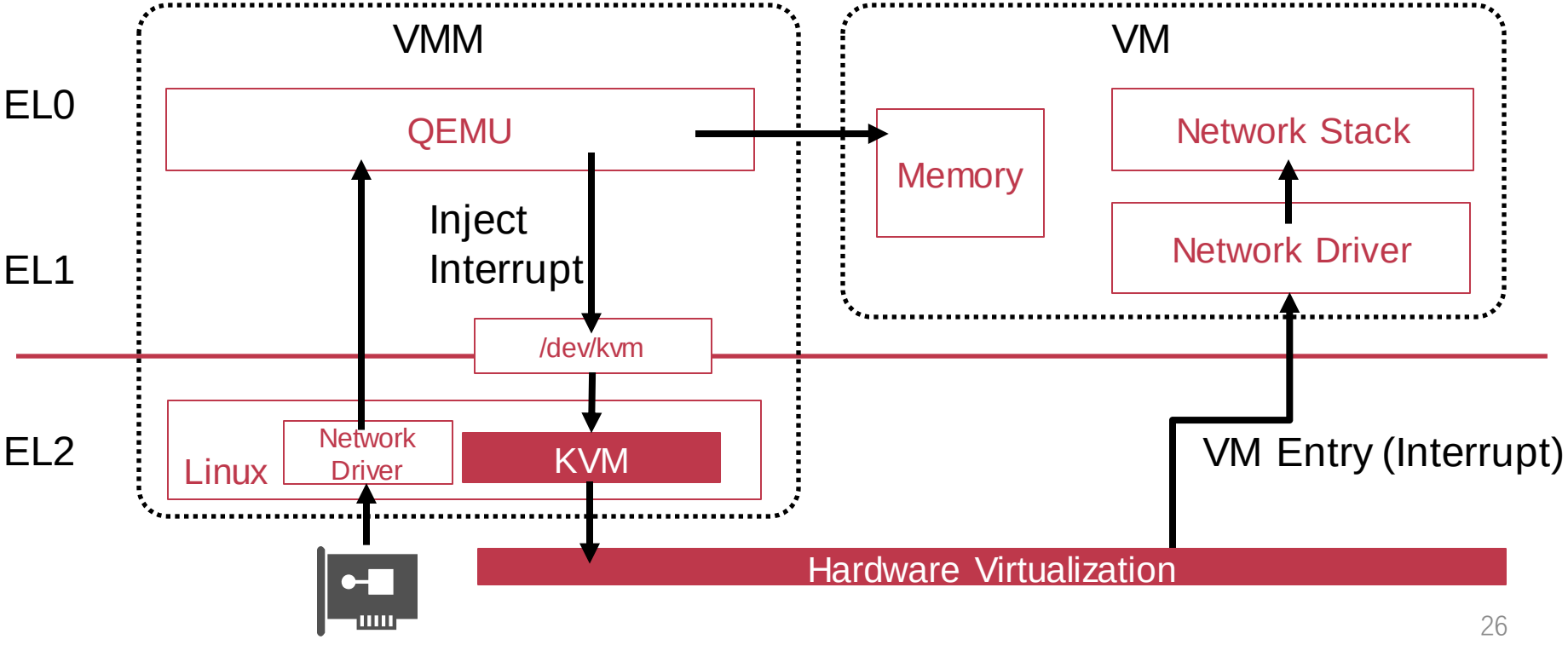
例：QEMU/KVM设备模拟1

- 以虚拟网卡举例——发包过程



例：QEMU/KVM设备模拟2

- 以虚拟网卡举例——收包过程



设备模拟的优缺点

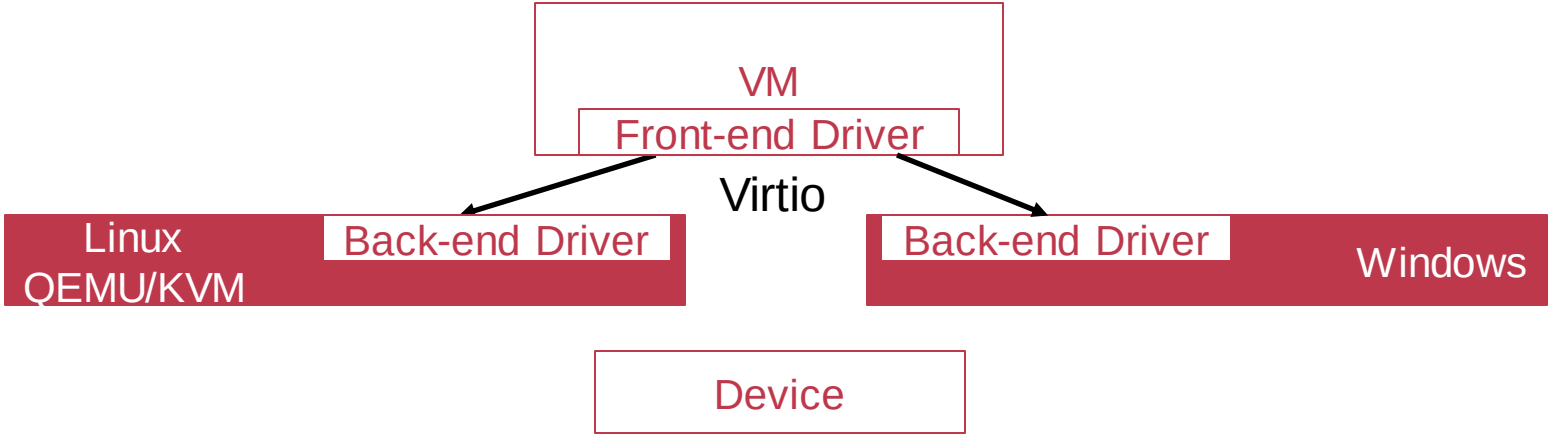
- **优点**
  - 可以模拟任意设备
    - 选择流行设备，支持较“久远”的OS（如e1000网卡)
  - 允许在中间拦截（Interposition）：
    - 例如在QEMU层面检查网络内容
  - 不需要硬件修改
- **缺点**
  - 性能不佳

方法2：半虚拟化方式

- **协同设计**
  - 虚拟机“知道”自己运行在虚拟化环境
  - 虚拟机内运行前端(front-end)驱动
  - VMM内运行后端(back-end)驱动
- **VMM主动提供Hypercall给VM**
- **通过共享内存传递指令和命令**

VirtIO: Unified Para-virtualized I/O

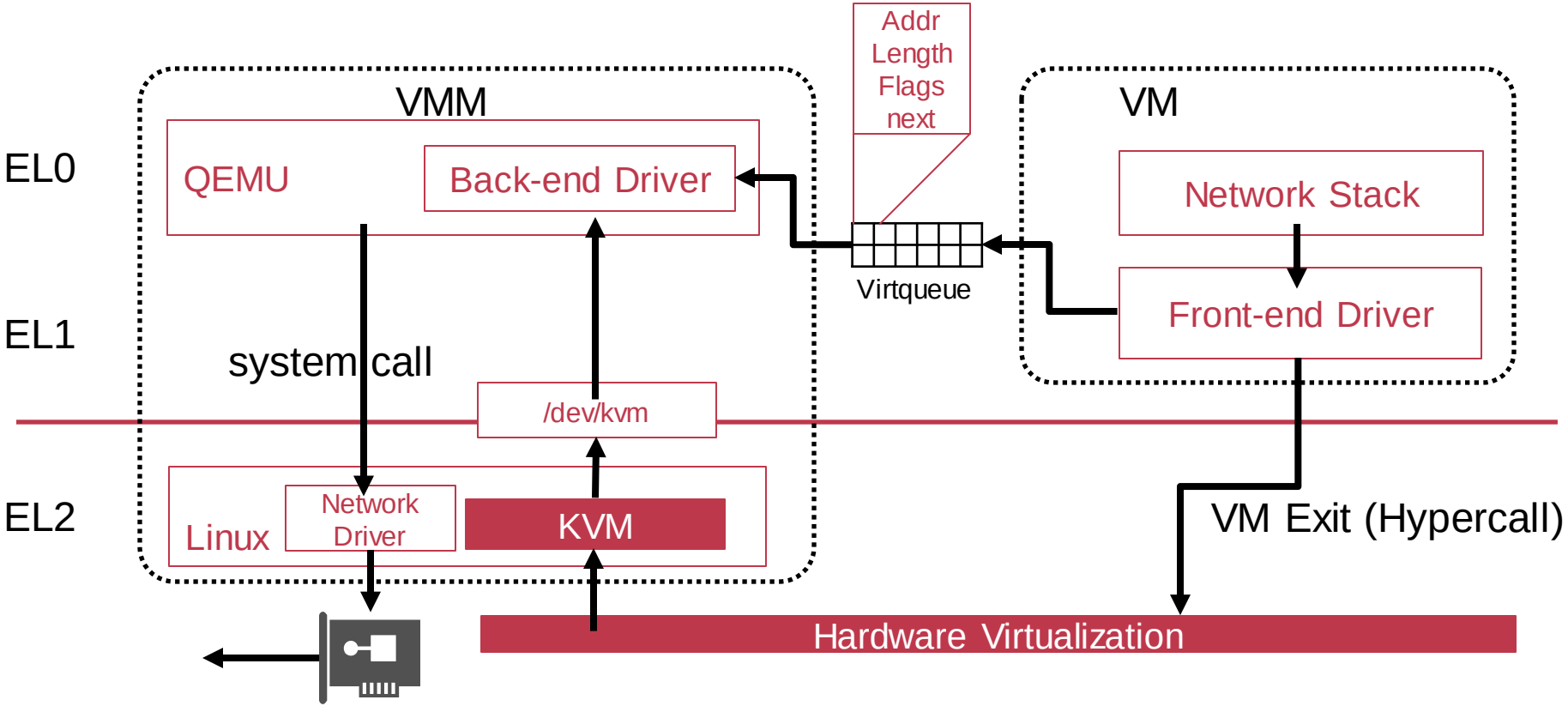
- **标准化的半虚拟化I/O框架**
  - 通用的前端抽象
  - 标准化接口
  - 增加代码的跨平台重用



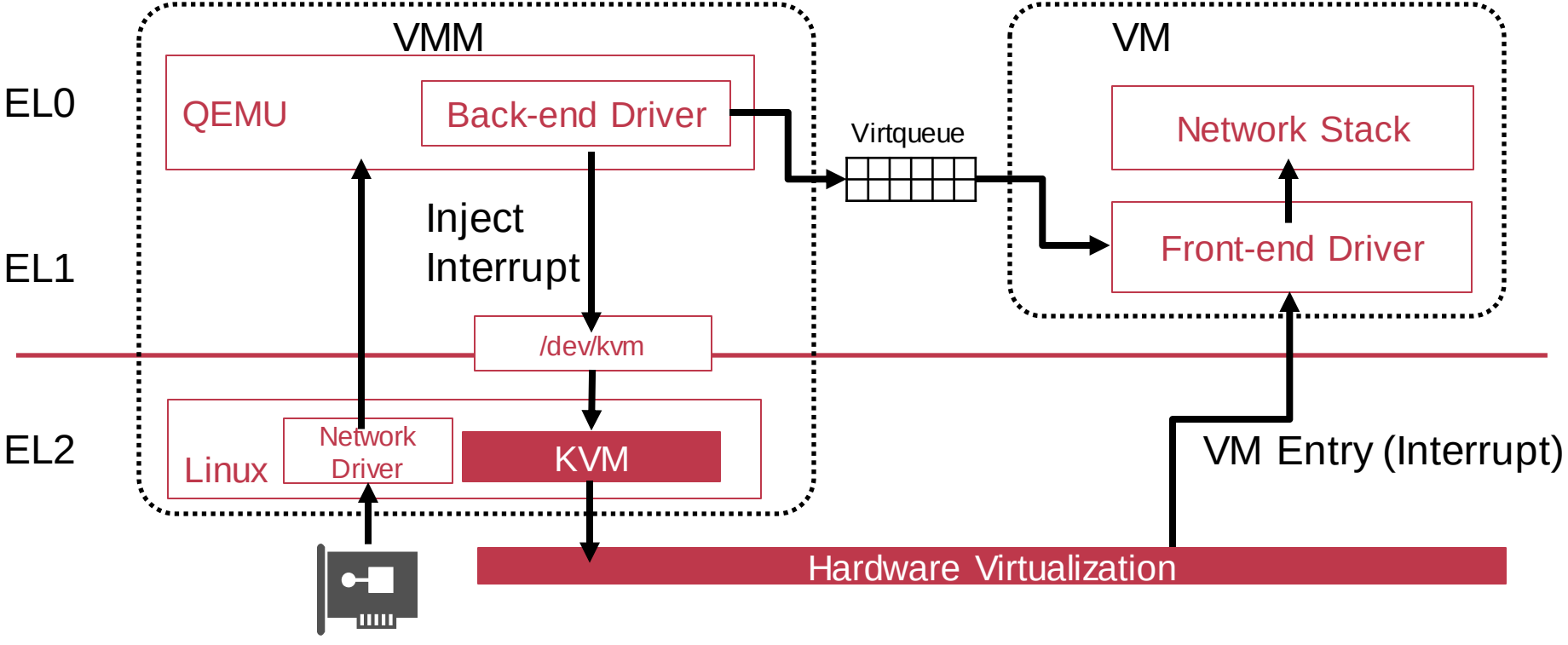
Virtqueue

- **VM和VMM之间传递I/O请求的队列**
- **3个部分**
  - Descriptor Table
    - 其中每一个descriptor描述了前后端共享的内存
    - 链表组织
  - Available Ring
    - 可用descriptor的索引，Ring Entry指向一个descriptor链表
  - Used Ring
    - 已用descriptor的索引

例：QEMU/KVM半虚拟化：发网络包



例：QEMU/KVM半虚拟化：收网络包

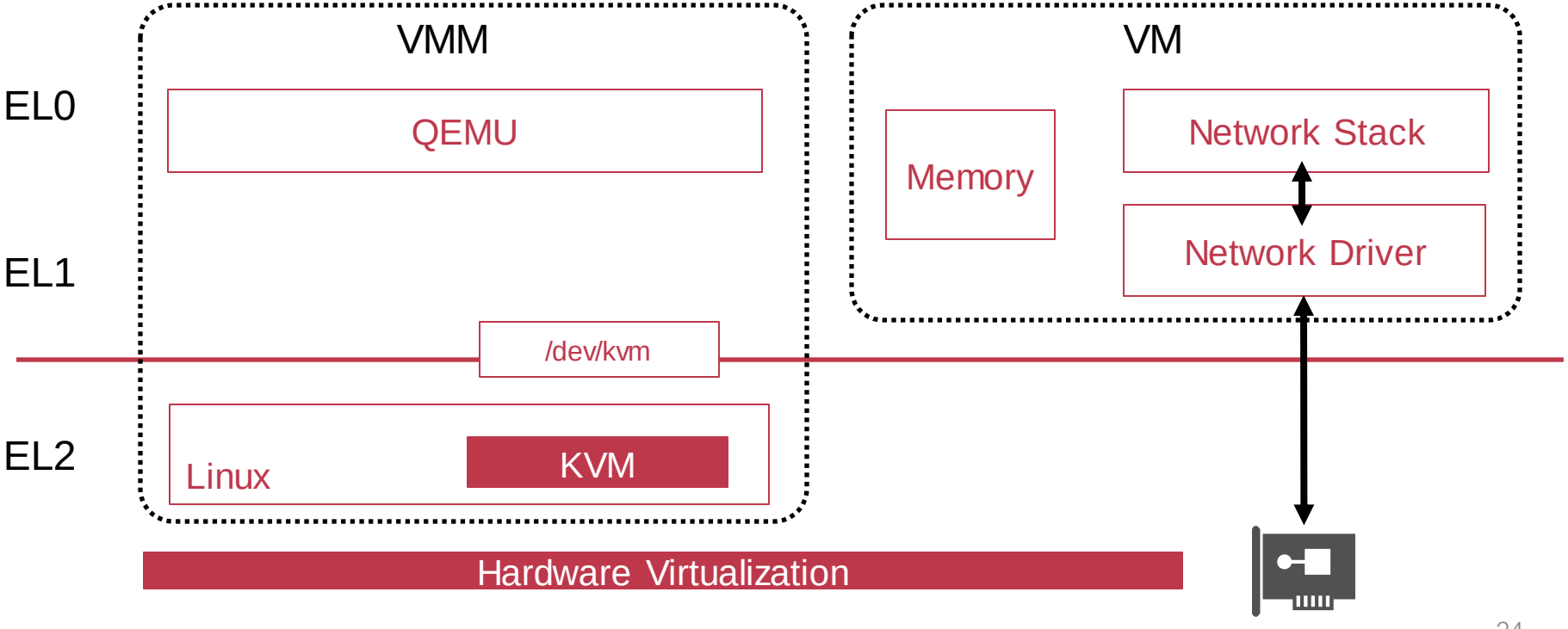


半虚拟化方式的优缺点

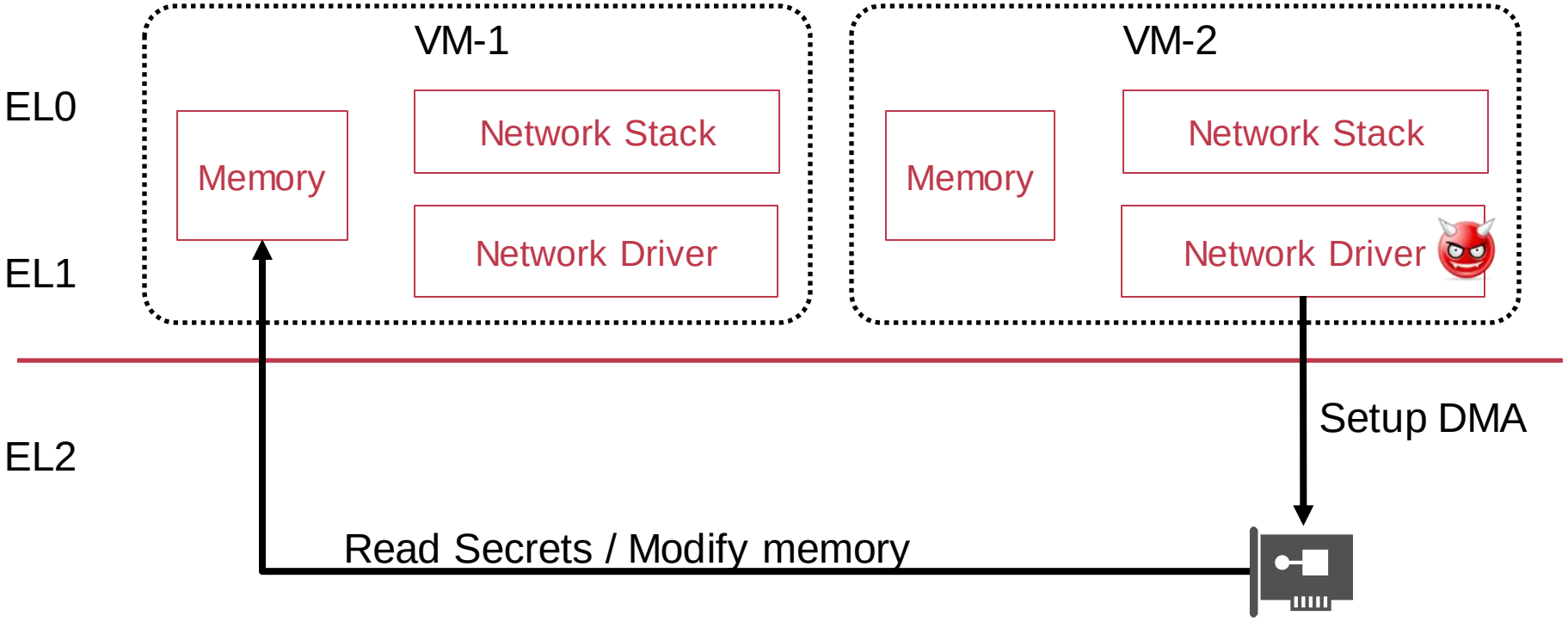
- **优点**
  - 性能优越
    - 多个MMIO/PIO指令可以整合成一次Hypercall
  - VMM实现简单，不再需要理解物理设备接口
- **缺点**
  - 需要修改虚拟机操作系统内核

方法3：设备直通

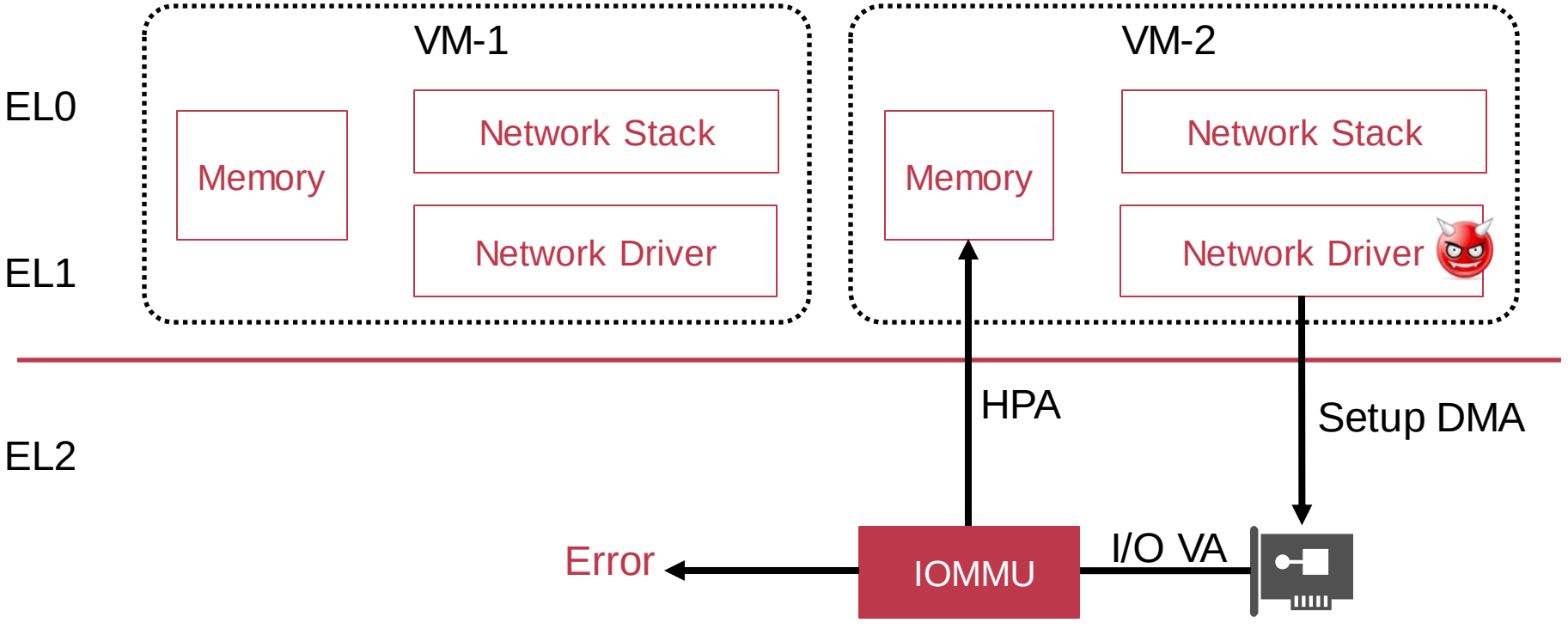
- 虚拟机直接管理物理设备



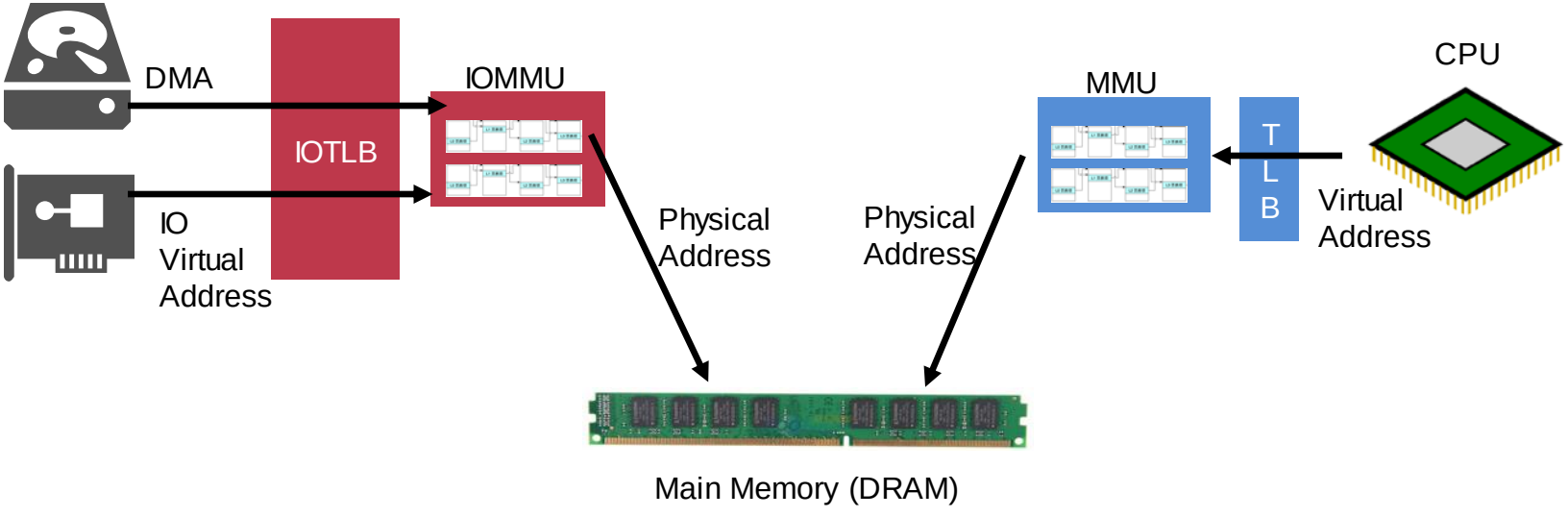
问题1：DMA恶意读写内存



使用IOMMU


VM-1
VM-2
EL0
EL1
EL2
Memory
Network Stack
Network Driver
Memory
Network Stack
Network Driver
HPA
Setup DMA
Error
IOMMU
I/O VA

IOMMU与MMU


DMA
IOTLB
IOMMU
IO
Virtual
Address
Physical
Address
Physical
Address
MMU
T
L
B
CPU
Virtual
Address
Main Memory (DRAM)

ARM SMMU

- **SMMU是ARM中IOMMU的实现**
  - System MMU
- **SMMU的设计与AARCH64 MMU一致**
  - 也存在两阶段地址翻译
  - 第一阶段：OS为进程配置：IOVA->GPA
  - 第二阶段：第一阶段翻译完之后进行第二阶段
    - VMM为VM配置：GPA->HPA

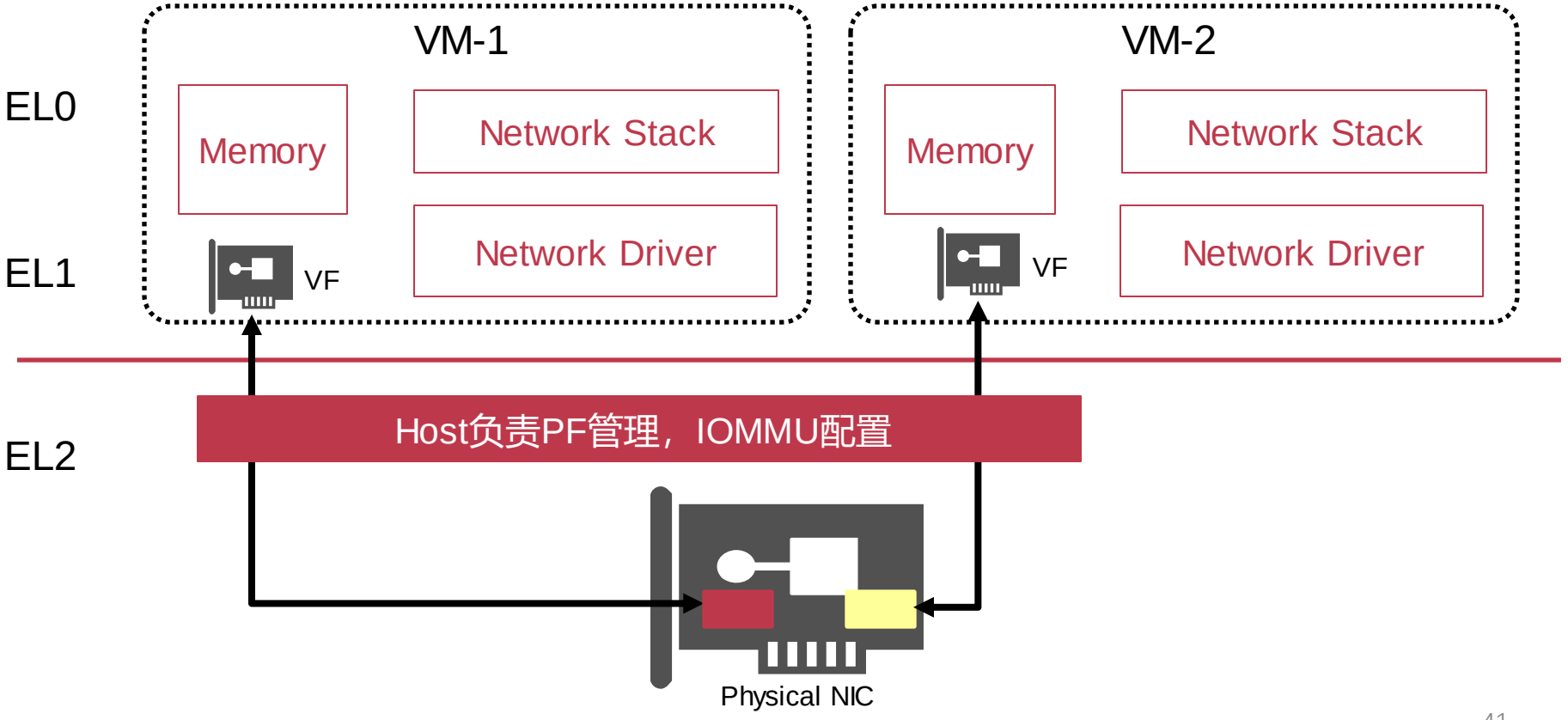
问题2：设备独占

- **Scalability不够**
  - 设备被VM-1独占后，就无法被VM-2使用
- **如果一台物理机上运行16个虚拟机**
  - 必须为这些虚拟机安装16个物理网卡

Single Root I/O Virtualization (SR-IOV)

- **SR-IOV是PCI-SIG组织确定的标准**
- **满足SRIOV标准的设备，在设备层实现设备复用**
  - 能够创建多个Virtual Function(VF)，每一个VF分配给一个VM
    - 负责进行数据传输，属于数据面（Data-plane）
  - 物理设备被称为Physical Function(PF)，由Host管理
    - 负责进行配置和管理，属于控制面（Control-plane）
- **设备的功能**
  - 确保VF之间的数据流和控制流彼此不影响

SR-IOV的使用


VM-1
VM-2
EL0
EL1
EL2
Memory
Network Stack
Network Driver
VF
Memory
Network Stack
Network Driver
VF
Host负责PF管理，IOMMU配置
Physical NIC

设备直通的优缺点

- **优点**
  - 性能优越
  - 简化VMM的设计与实现
- **缺点**
  - 需要特定硬件功能的支持（IOMMU、SRIOV等）
  - 不能实现Interposition：难以支持虚拟机热迁移

I/O虚拟化技术对比

	设备模拟	半虚拟化	设备直通
性能	差	中	好
修改虚拟机内核	否	驱动+修改	安装VF驱动
VMM复杂度	高	中	低
Interposition	有	有	无
是否依赖硬件功能	否	否	是
支持老版本OS	是	否	否

中断虚拟化

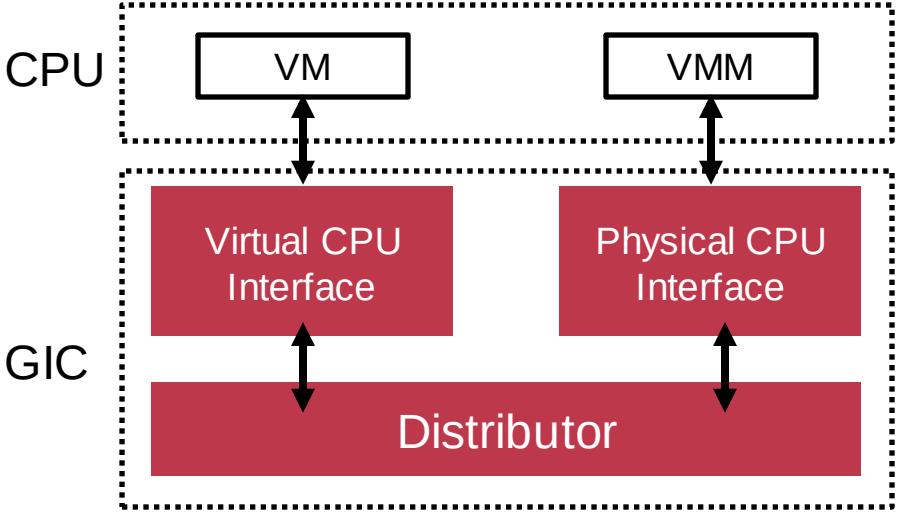
- **VMM在完成I/O操作后通知VM**
  - 例如在DMA操作之后
- **VMM在VM Entry时插入虚拟中断**
  - VM的中断处理函数会被调用
- **虚拟中断类型**
  - 时钟中断
  - 核间中断
  - 外部中断

RISC-V中断虚拟化的实现方法

- **打断虚拟机执行**
  - 通过VSIP寄存器注入虚拟的中断
  - 通过SBI接口注入核间中断和时钟中断
- **不打断虚拟机执行**
  - 通过PLIC/VirtIO插入中断

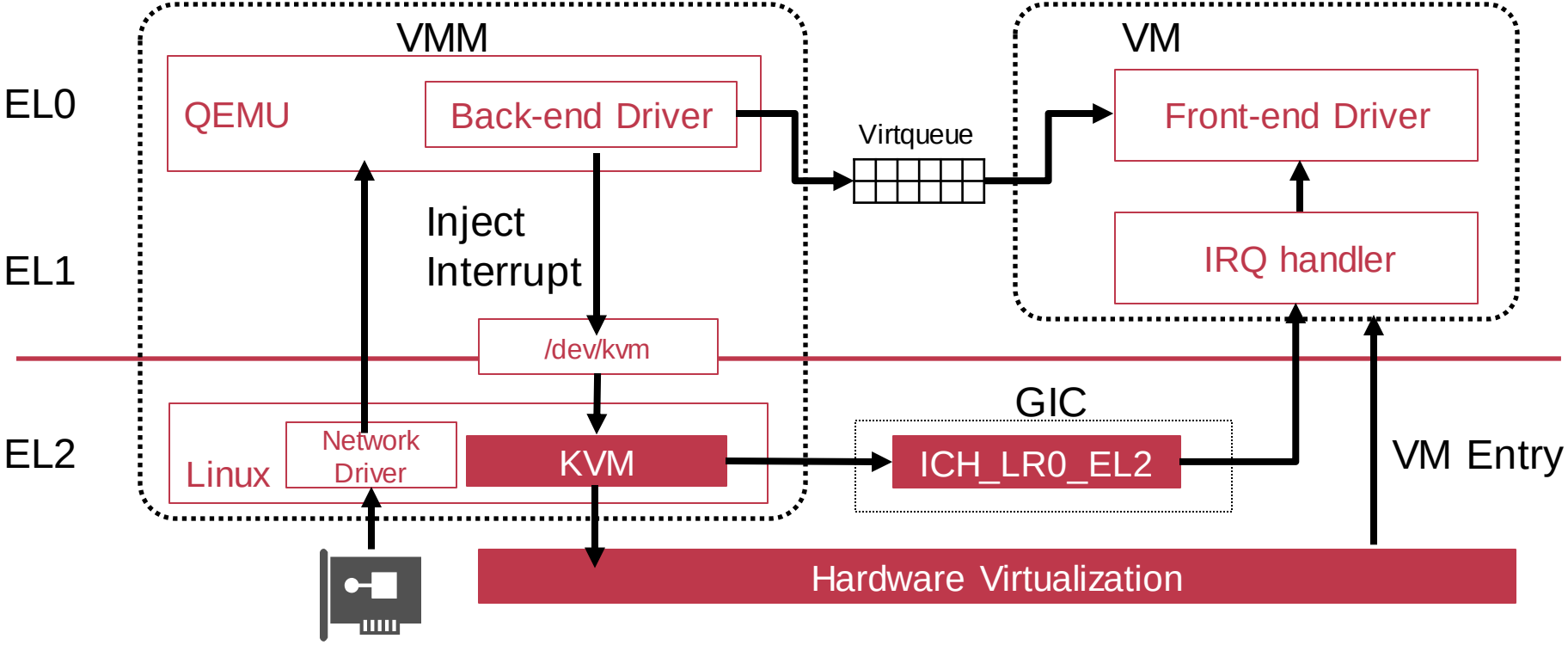
Virtual CPU Interface

- **GIC为虚拟机提供的硬件功能**
  - VM通过Virtual CPU Interface与GIC交互
  - VMM通过Physical CPU Interface与GIC交互



插入虚拟中断：以半虚拟化举例

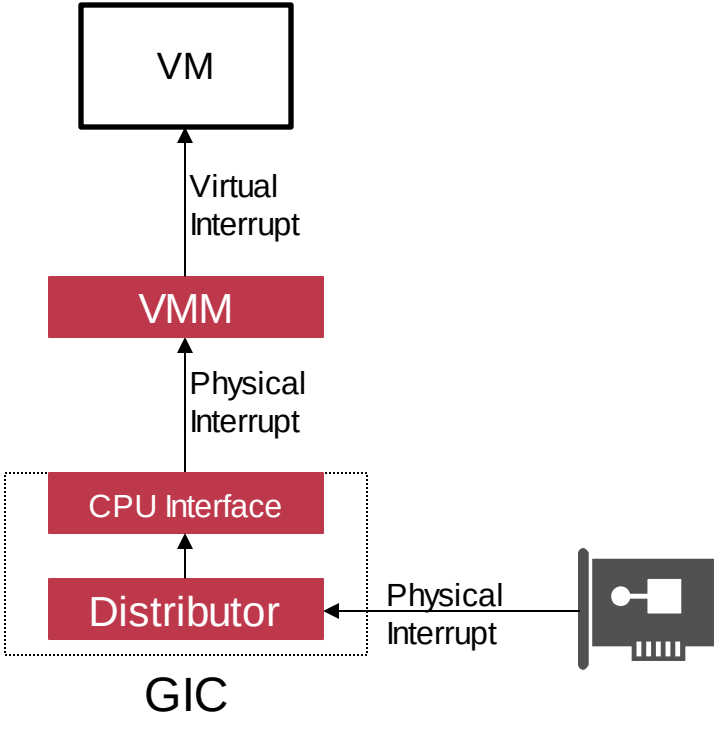
思考：这种方式有什么问题?



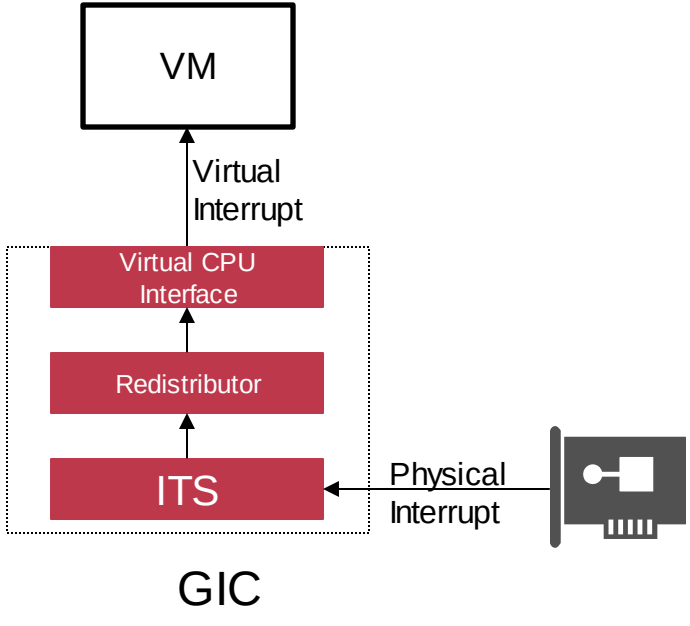
不打断虚拟机执行：GIC ITS

- **GIC第4版本推出了Direct injection of virtual interrupts**
  - 将物理设备的物理中断与虚拟中断绑定
  - 物理设备直接向虚拟机发送虚拟中断
- **VMM在运行VM前**
  - 配置GIC ITS (Interrupt Translation Service)
    - 建立物理中断与虚拟中断的映射
  - 映射内容
    - 设备与物理中断的映射
    - 分配虚拟中断号
    - 发送给哪些物理核上的虚拟处理器

虚拟中断的直接插入



GICv3中的物理中断插入



GICv4中的物理中断插入