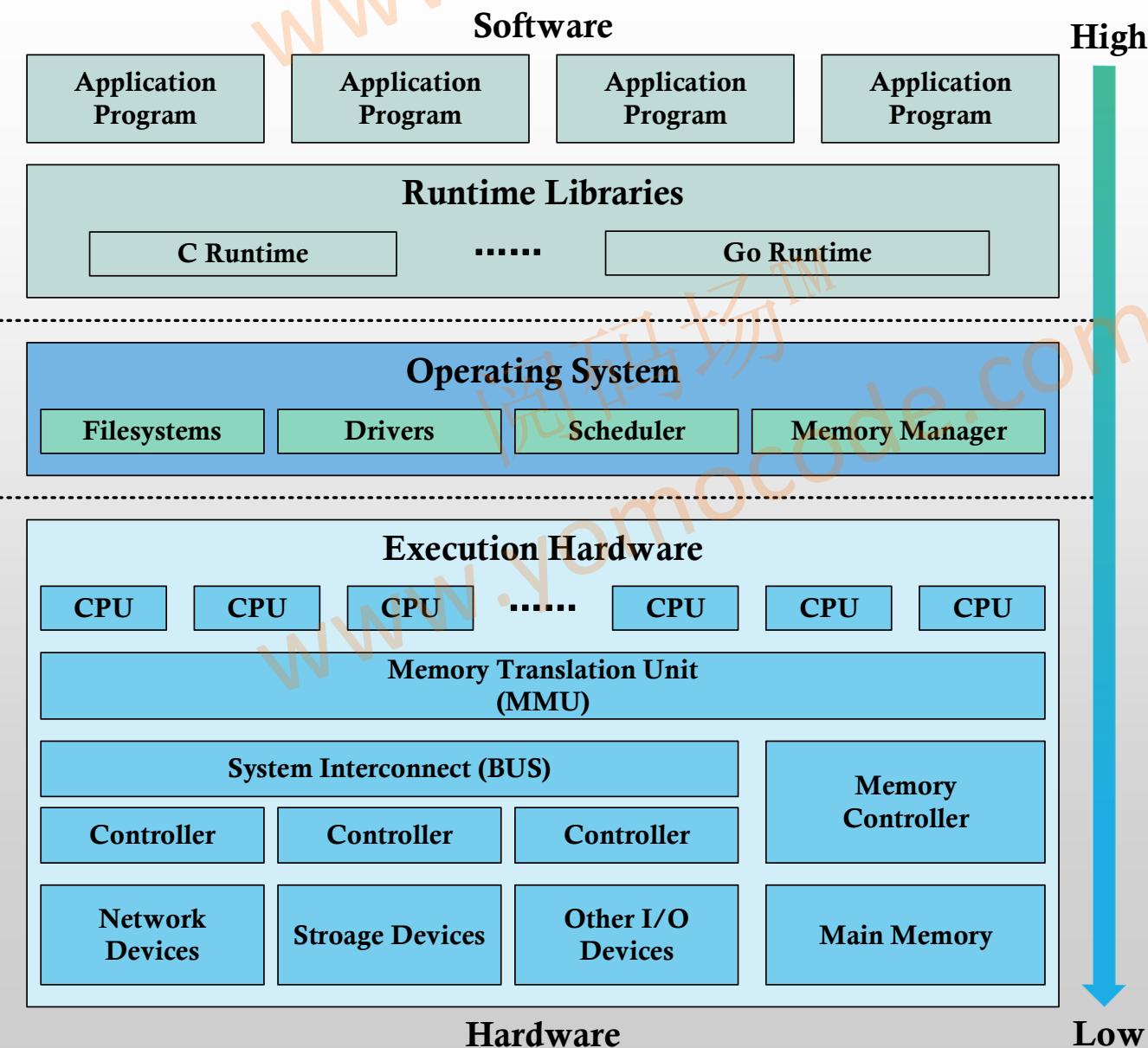


# **OVERVIEW OF ARM VIRTUALIZATION**



**YOMOCODE, JEDIX81@GMAIL.COM, 2019**

# COMPUTER SYSTEM ABSTRACTION

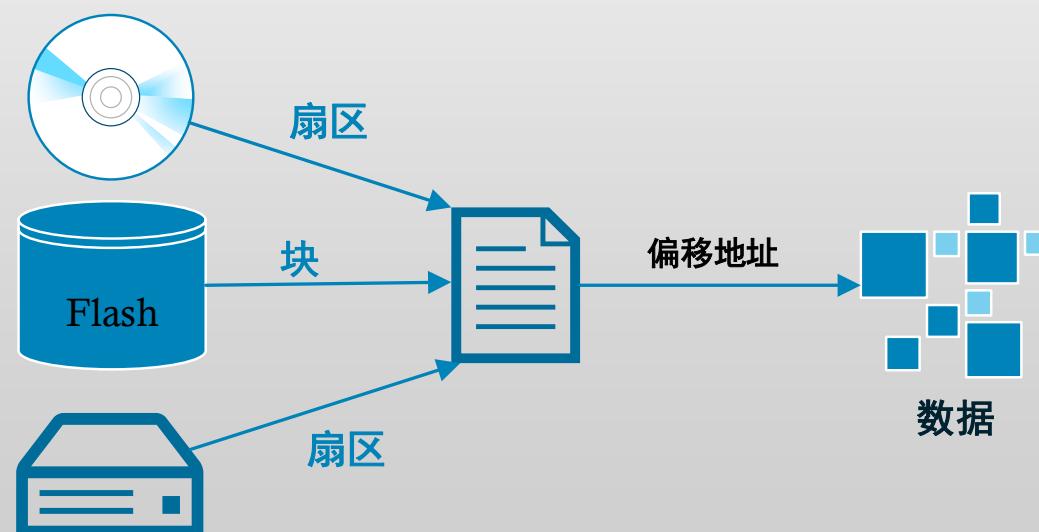


计算机抽象：

- 计算机系统由从低到高的抽象层构建而成
- 较高级别的抽象层隐藏了低级别抽象层的细节

一个例子：

- 文件就是一种对磁盘的抽象



# WHAT IS VIRTUALIZATION?

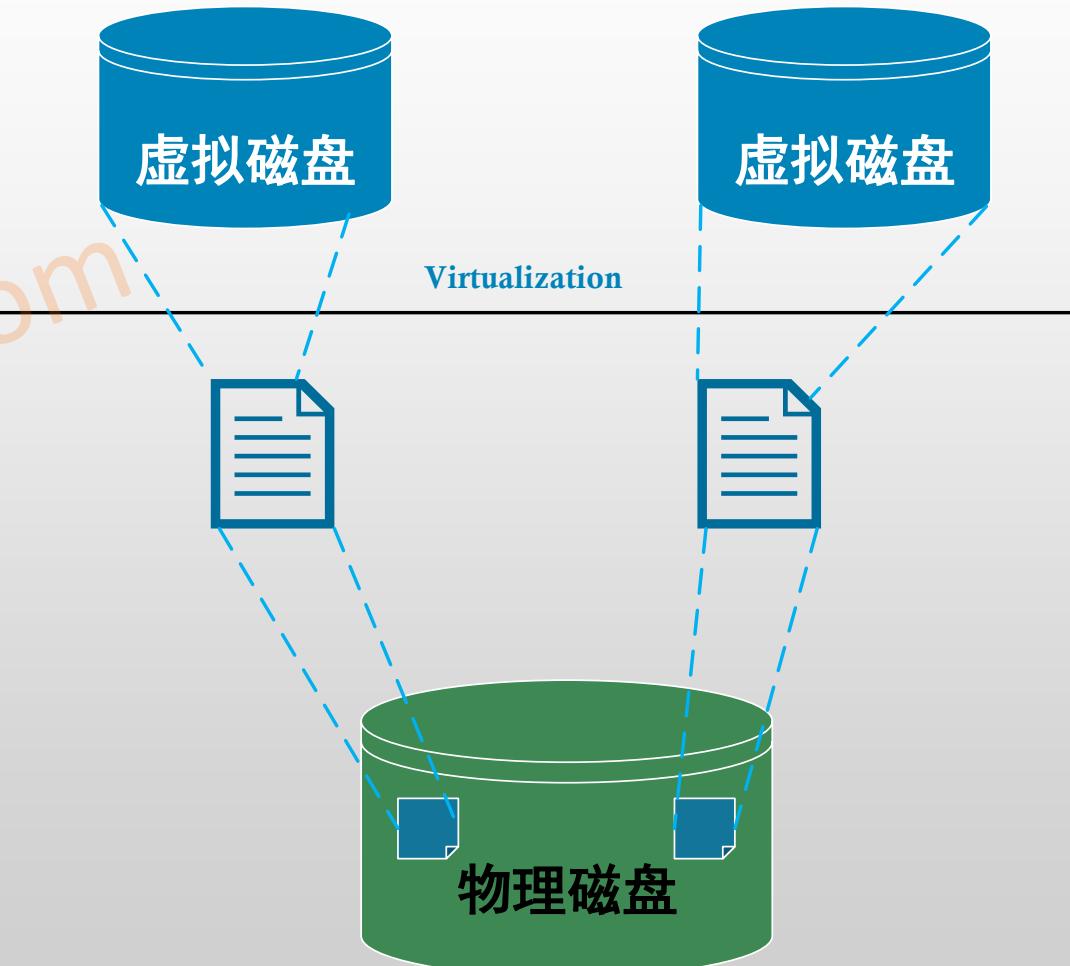
虚拟化和抽象非常的类似，除了：

- 它不需要隐藏底层的细节

虚拟磁盘暴露的虚拟硬件细节：

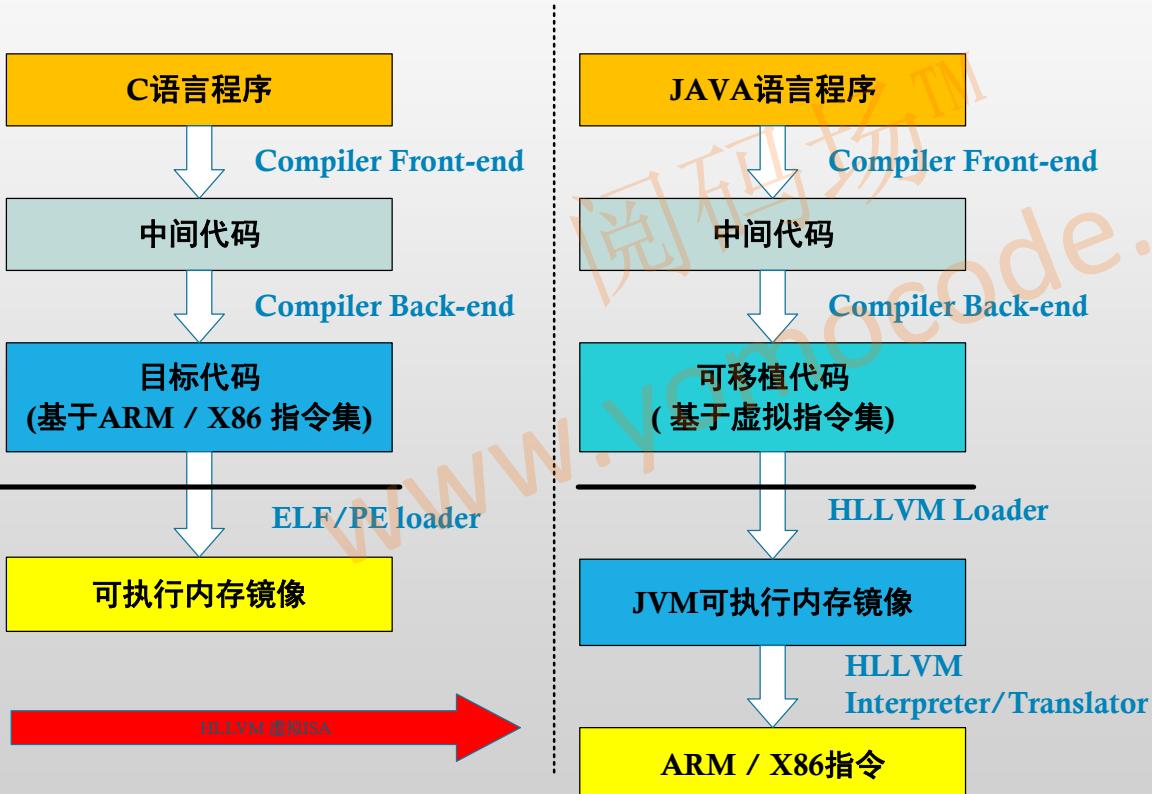
1. 接口类型, **SCSI, IDE**
2. 总线地址, **LUN-ID, IDEO**
3. 寻址方式, **LBA, CHS**
4. 磁盘的容量大小 **500GB, 2TB**
5. 磁盘的限制, 扇区大小, 擦除块大小
6. 磁盘的其他特性, **trim, APM**

有了这些细节, **GuestOS**的磁盘驱动程序才能正确的识别磁盘, 并找到相应的驱动程序。文件系统根据这些细节才能够使用最优的参数配置。

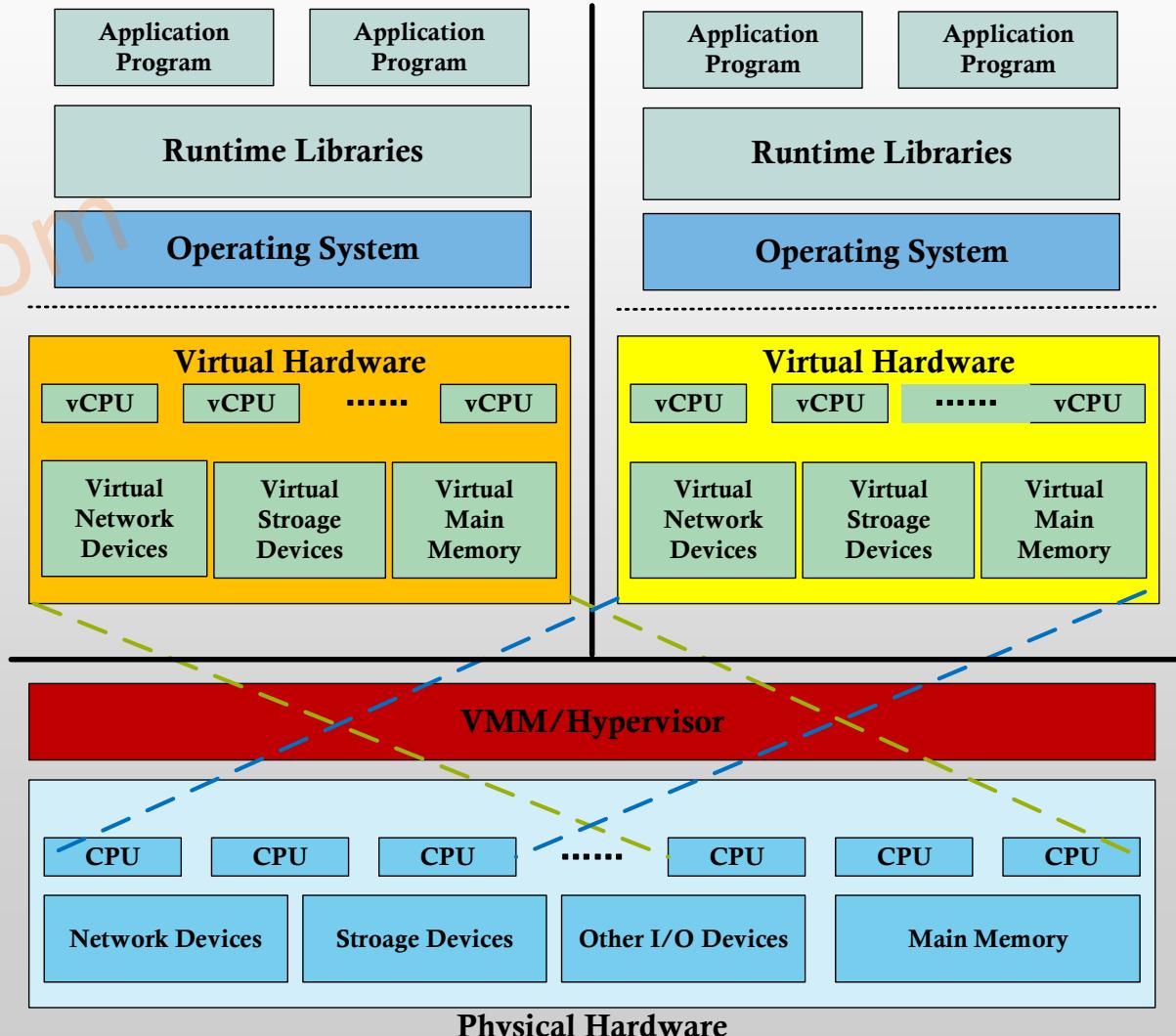


# VIRTUAL MACHINES?

高级语言虚拟机 (High Level Language Virtual Machine) - Java 虚拟机



系统虚拟机 (System Virtual Machine) :  
VMWARE/VIRTUALBOX/XEN/Linux-KVM



# SYSTEM VIRTUAL MACHINES

今天讨论虚拟化，大多都是在讨论系统虚拟机这一种类型。

系统虚拟机特性：

1. 构建在指令集这一个计算机抽象层：

执行**GuestOS**指令最常用的两种技术

➤ 指令模拟 (**Guest**和**Host**可以是不同的**ISA**)

➤ 本地指令直接执行 (**Guest**和**Host**必须是相同的**ISA**)

2. 系统资源虚拟化，提供一个系统环境

➤ **CPU**虚拟化

➤ **Memory** 虚拟化

➤ **IO**虚拟化

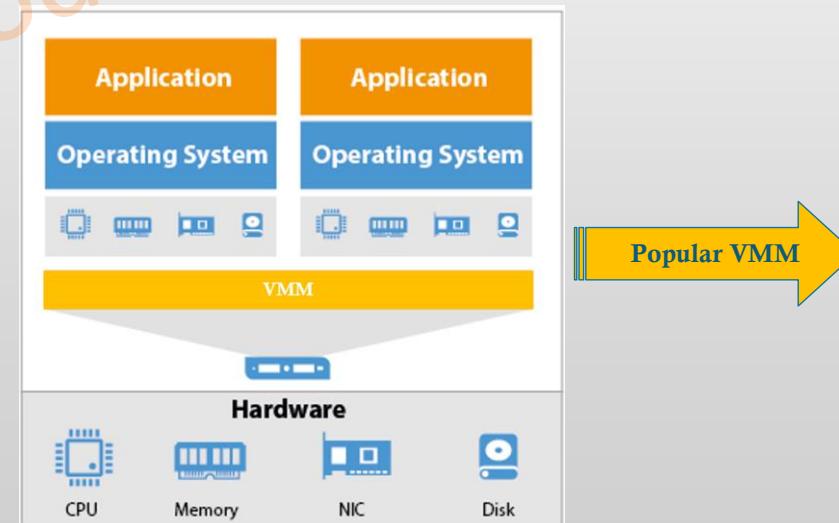
3. 持久化构建

➤ 虚拟机配置一次性创建完成不会因为进程运行结束而消亡。

➤ 虚拟机每次运行都是继承上一次的状态（对虚拟机的修改被保存）

常见的几种系统虚拟机软件：

**VMWARE/VirtualBox/Xen/Linux-KVM**



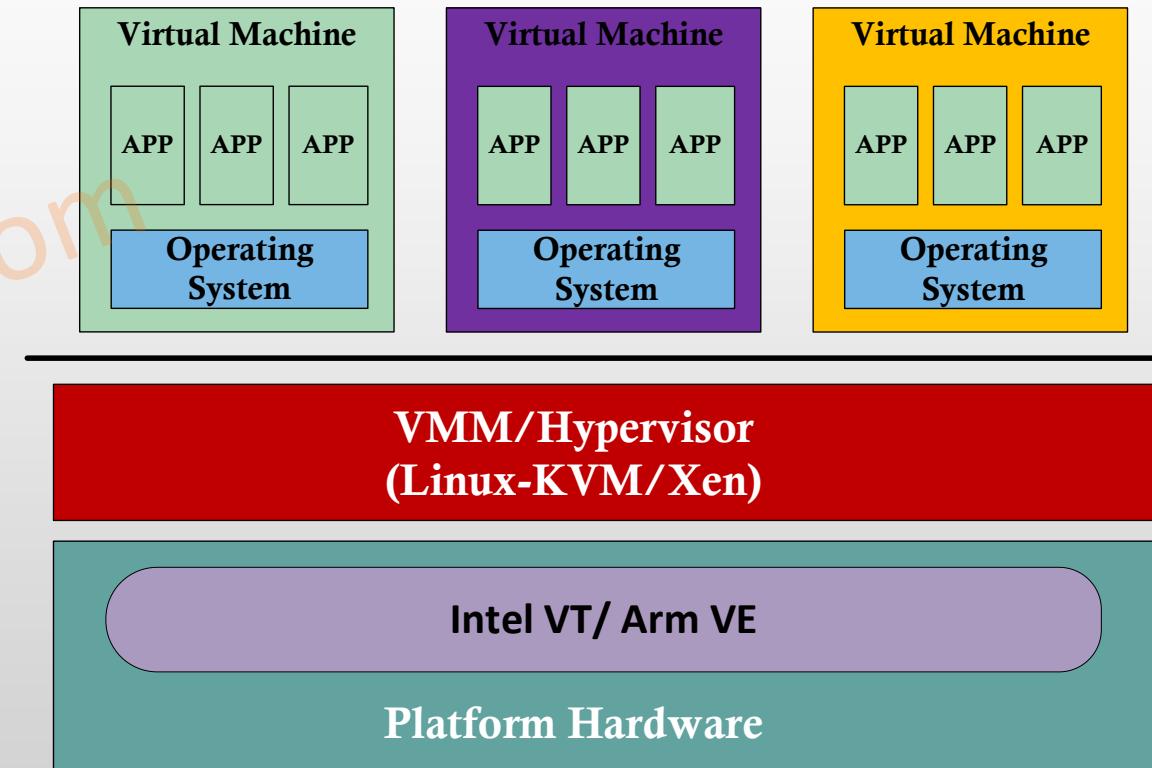
# WHAT IS VIRTUALIZATION?

虚拟化是一种能力：

- 它可以是一种软件能力 - **Emulator**
- 它可以是一种硬件内里 -**CPU** 微码翻译
- 它也是软硬结合的能力 - **Intel VT-X/Arm Virtualization Extension + Linux-KVM**

**Hypervisor/Virtual Machine Manager (VMM)**则是这种能力的一种呈现：

1. 控制所有的系统资源(**CPU**, 内存, 网络, 存储等)
2. 创建虚拟机并分配相应的资源
3. 管理虚拟机的生命周期



# WHY VIRTUALIZATION?

我们先来看看虚拟化有哪些优点：

## 1. 更高的系统安全性和可用性

- **VMM/Hypervisor**作为一个**monitor**, 运行在比传统操作系统的更高的特权层
- 控制和过滤虚拟机的行为
- 监控虚拟机状态, 故障快速恢复

## 2. 最大化硬件资源使用率

- 在一个物理主机上创建多个虚拟机共享主机资源, 节约硬件和能源成本

## 3. 系统易扩展性

- 修改虚拟机的配置来适应业务的负载变化
- **Aggregation**聚合技术

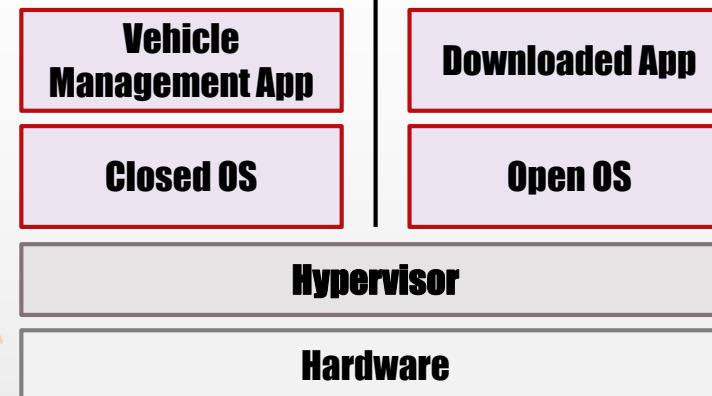
## 4. 方便的可迁移性特性

- 虚拟化的系统消除物理主机的硬件差异
- 虚拟机以文件镜像的格式封装

## 5. 硬件级别的隔离特性

- 借助硬件虚拟化技术, **VMM**可以提供硬件级别的隔离特性——硬件**sandbox**的基础

车载虚拟化场景例子：



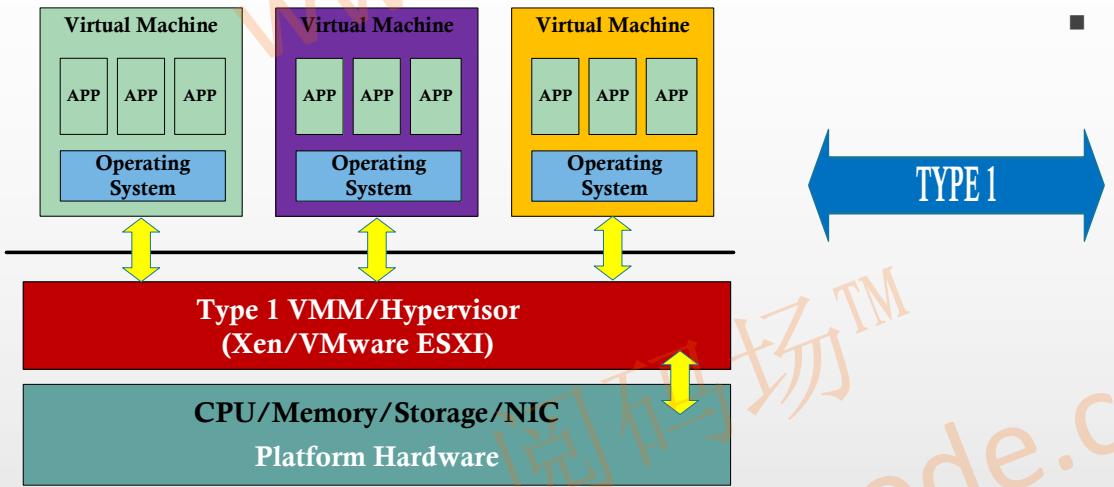
1. 封闭操作系统和开放操作系统运行在同一个硬件系统里。

2. 封闭操作系统运行一些比较关键的汽车功能软件, 这是一个功能安全的区域。

3. 开放操作运行一些非关键的软件, 同时运行用户安装他们自己的**APP**。

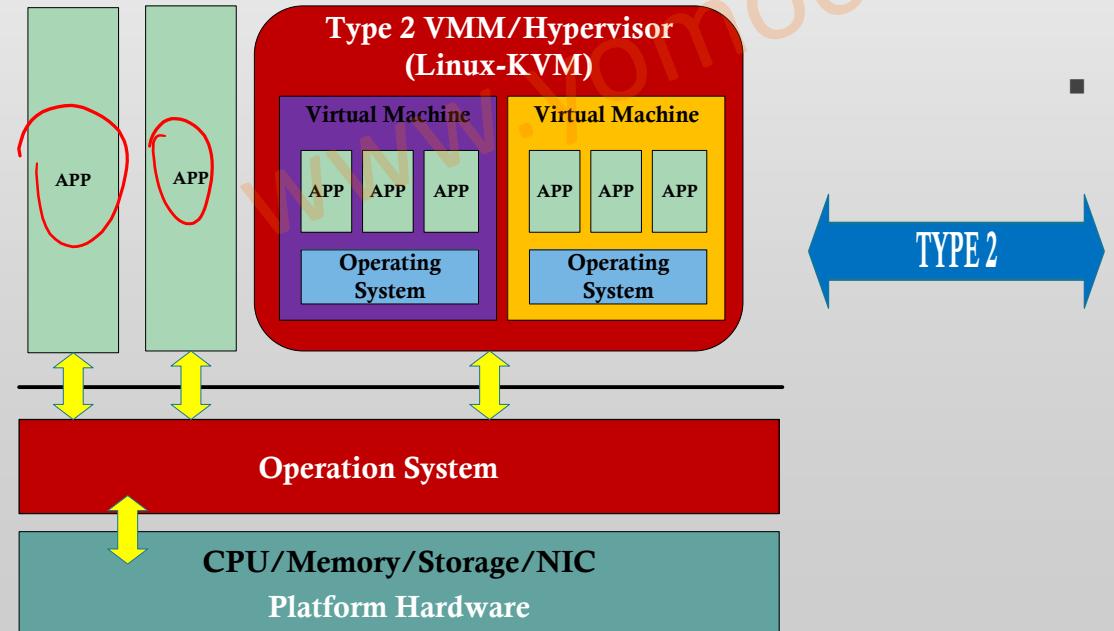
4. 虚拟化在这个场景里就利用它的硬件级别的隔离特性, 将用户安装的某些恶意软件隔离在开放操作系统里, 不会对汽车功能软件造成影响。

# TYPES OF HYPERVISOR/VMM



## Type 1 Hypervisor (or Native VMM, Bare-metal VMM)

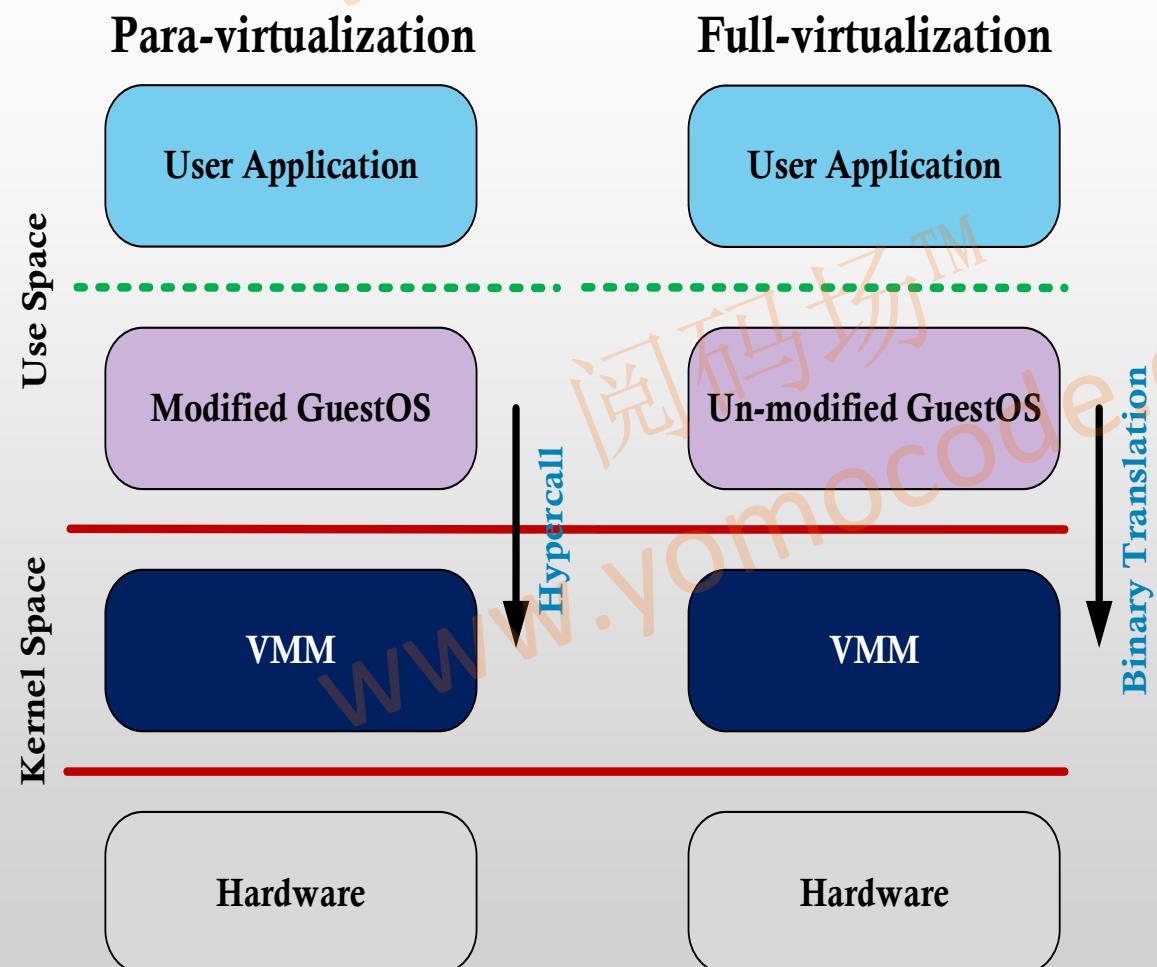
- 启动时**bootloader/BIOS**直接将执行权限交给**Hypervisor**
- 直接运行在硬件之上，不需要依赖基础操作系统，直接访问所有硬件资源。
- 可以控制所有的 **guest OS**
- 底噪小，交互少，性能好，稳定可靠易扩展
- 无法直接利用现有操作系统生态，硬件兼容性略差
- Example: Xen**



## Type 2 Hypervisor (or hosted-VMM)

- 启动时**bootloader/BIOS**先启动基础操作系统，**Hypervisor**相当于基础操作系统里安装的一个软件
- 需要通过请求基础操作系统来访问硬件资源。
- 只能控制 **guest OS**，不能控制基础操作系统中的其它程序
- 和基础操作系统间交互产生的**overhead**会影响性能
- 基础操作系统底噪大，攻击窗口多，安全性略低
- 可以直接利用现有操作系统生态，硬件兼容性好
- Example: Linux-KVM**

# VIRTUALIZATION WITHOUT HARDWARE ASSISTANT



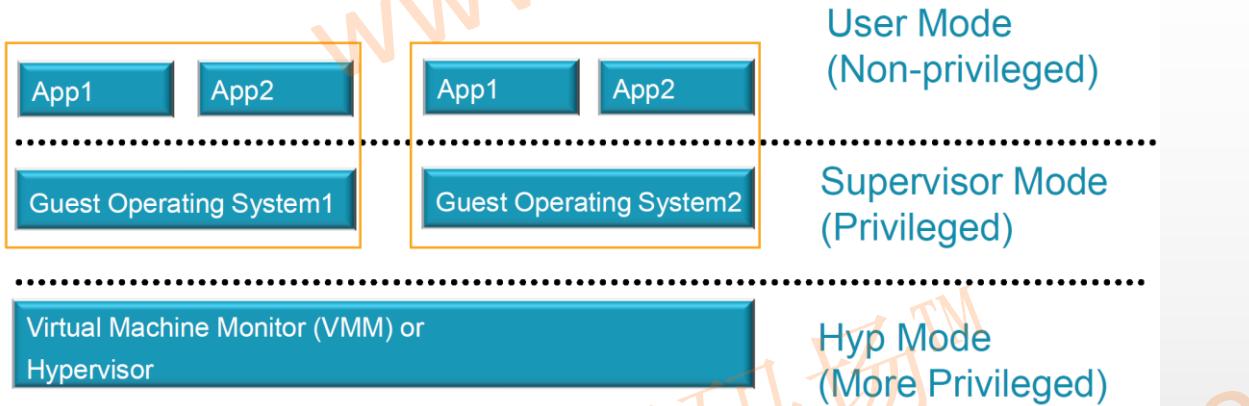
## 半虚拟化

- 将 **GuestOS** 降权，让它无法直接访问系统特权资源
- **VMM** 提供访问系统特权资源的 **Hypercall API**
- 修改 **GuestOS**，用 **Hypercall API** 访问系统特权资源
- 高效轻量的虚拟化技术，性能接近 **native performance**
- **GuestOS** 修改量比较大，使用不便

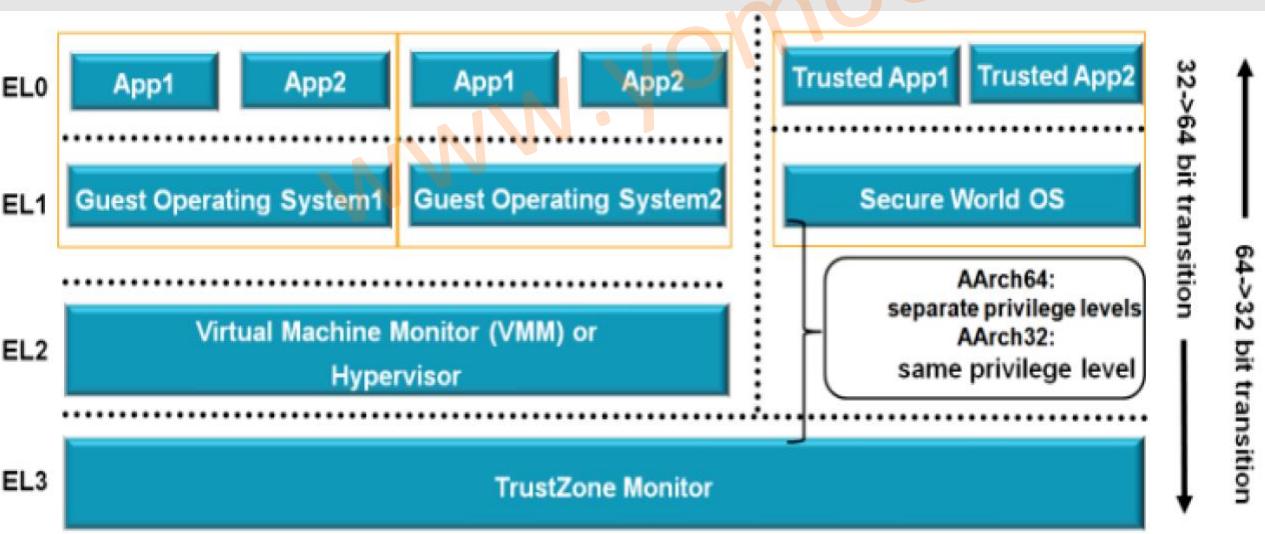
## 全虚拟化

- 将 **GuestOS** 运行在 **VMM** 创建的一个独立环境里
- **VMM** 通过代码块扫描，将内核特权访问操作翻译成一系列对 **VMM** 的请求，用户态代码直接执行
- **GuestOS** 对虚拟化环境不感知，不需要修改 **GuestOS**
- **VMM** 实现复杂
- **VMM** 和 **GuestOS** 之间翻译产生的 **overhead** 比较大

# HARDWARE ASSISTANT VIRTUALIZATION ON ARM



ARMv7/AArch32 Virtualization Extension



ARMv8/AArch64 New Exception Model

## Arm硬件虚拟化技术

### CPU特权层扩展

- 新增加一个特权模式来运行**VMM**
  - HYP mode (Arm32)**
  - EL2 Exception Level (Arm64)**
- GuestOS**和没有虚拟化时一样，运行在原有的特权模式不需要修改**GuestOS**
- VMM**所在特权层有更高的权限，可以控制**GuestOS**访问硬件的权限

### MMU虚拟化支持

- LPAE (Arm32), Stage 2 Translation**

### GIC虚拟化

- VCPU interface, Hypervisor interface**

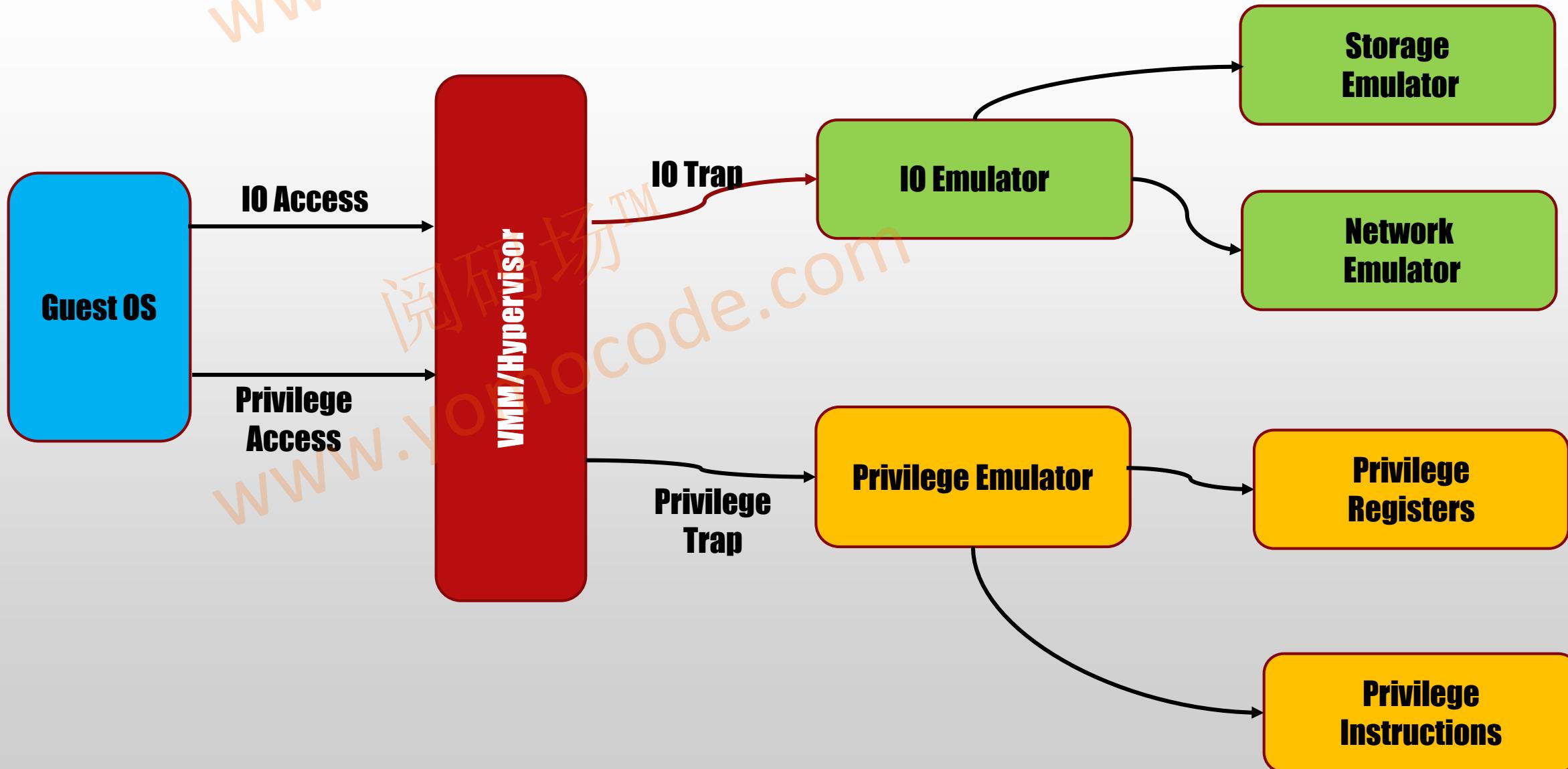
### Arch\_timer 虚拟化

- Hypervisor Timer, Virtual Timer**

### SMMU

- Stage 2 Translation for DMA**

# TRAP AND EMULATOR

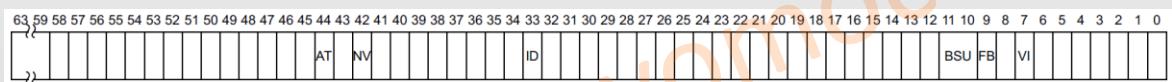


# CPU VIRTUALIZATION

Arm的CPU虚拟化通过硬件**trap**和软件模拟来完成

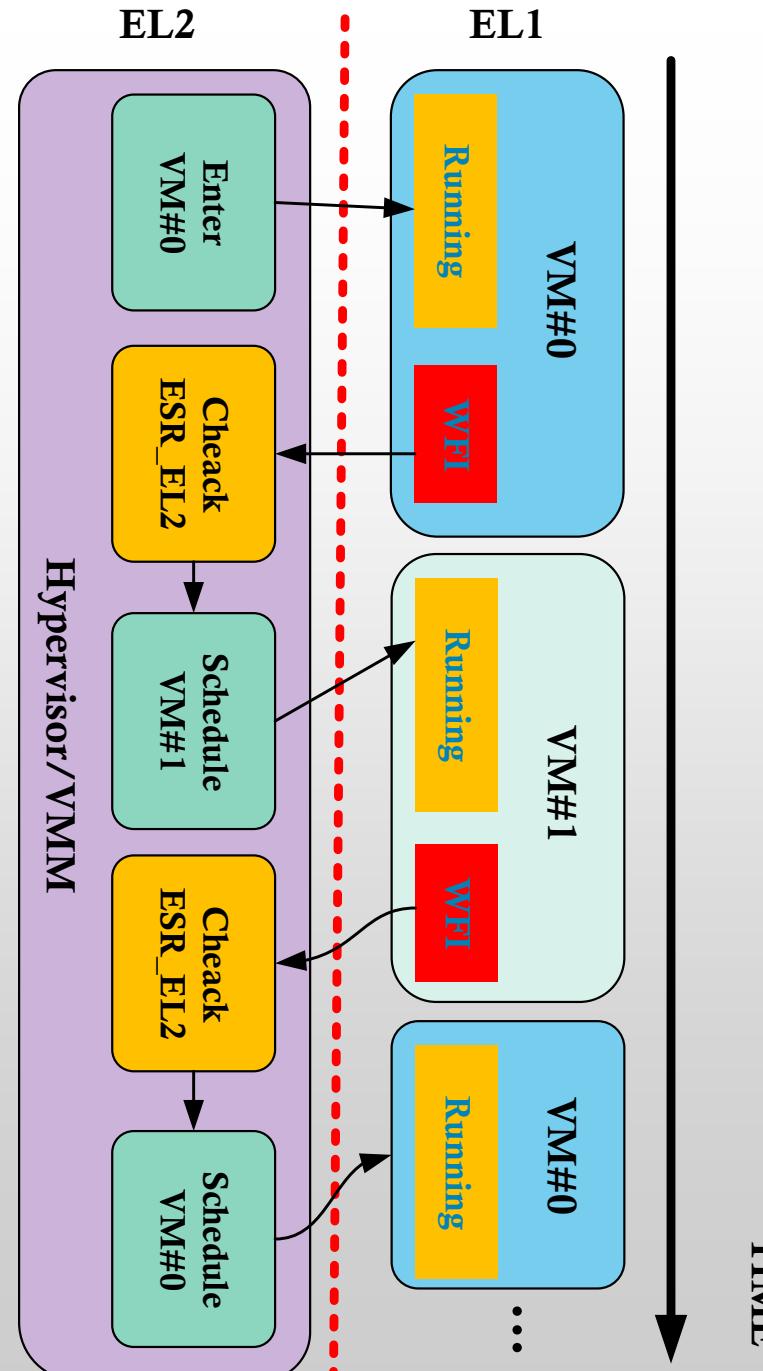
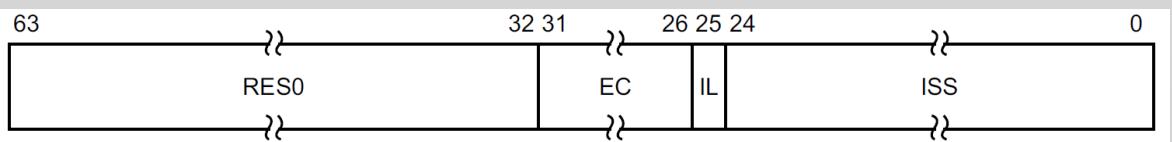
## 1. HCR\_EL2 Hypervisor Configuration Register

- 配置**VM**产生硬件**trap**的条件
- 有非常丰富的组合，如**TLB/cache**的操作，**ID**寄存器的访问和一些特殊指令  
**(TLB/cache ops, ID groups, instructions)**



## 2. ESR\_EL2 Exception Syndrome Register

- 当**trap**发生时，确定**VM**产生硬件**trap**的原因



# PRIVILEGE REGISTERS

5 GROUPs of ID  
registers

MemTag

Software  
Context  
Number

RAS

Nested  
Virtualization

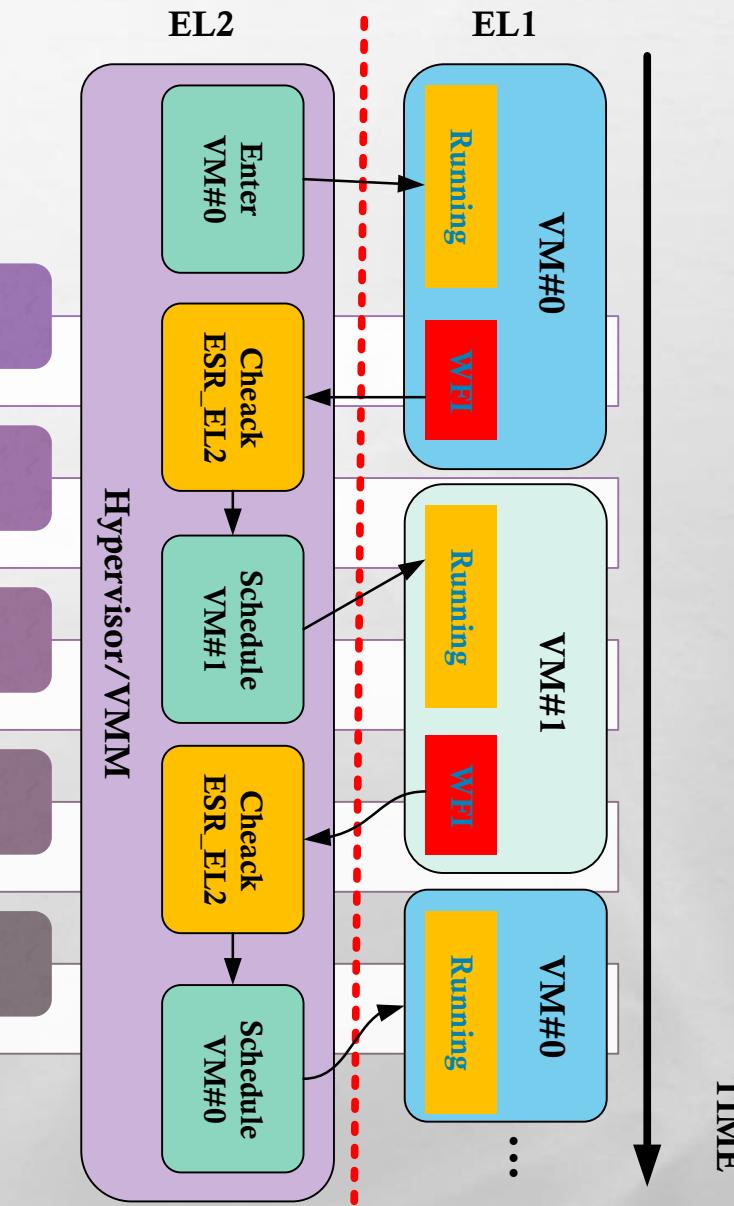
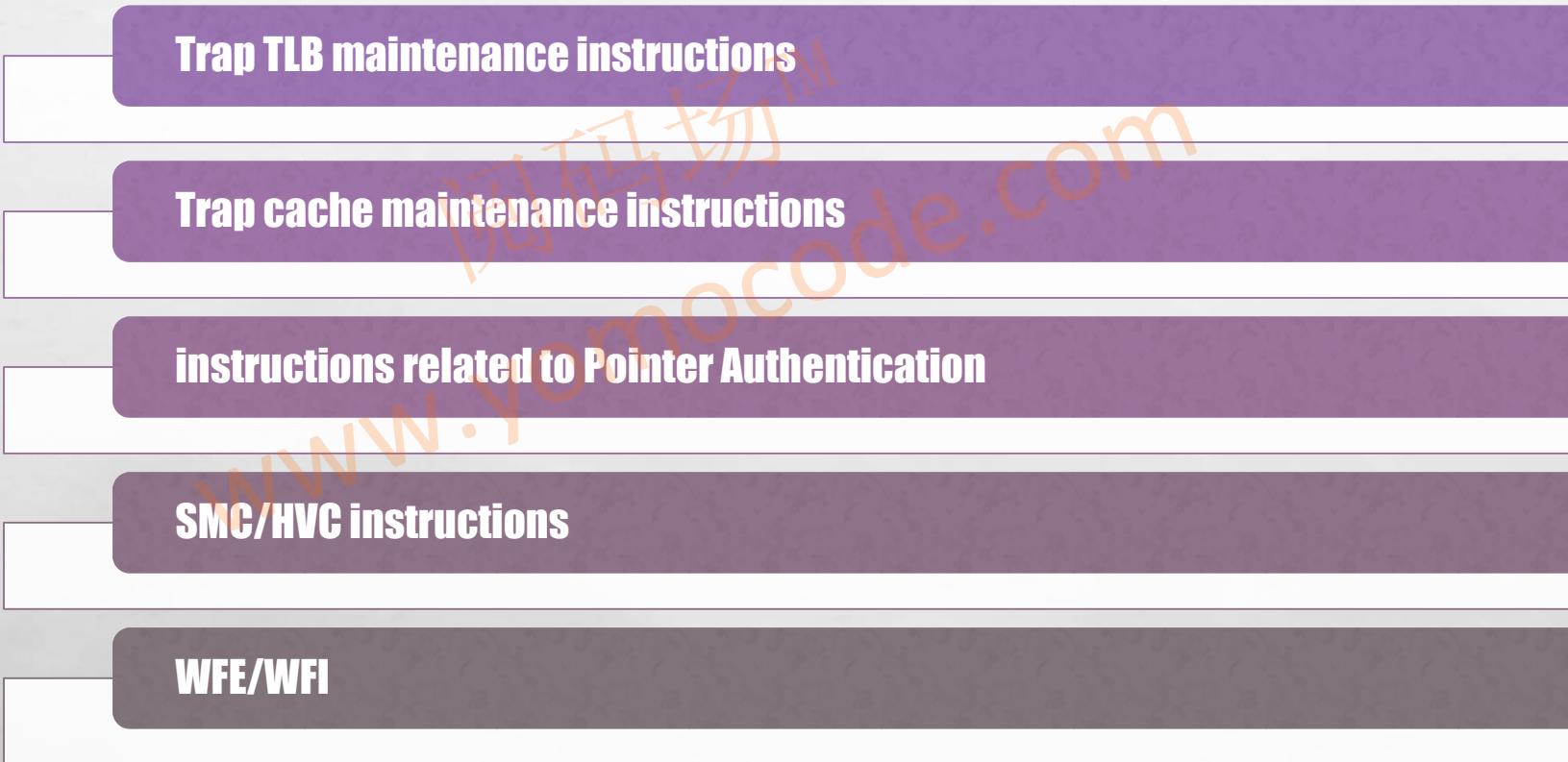
Pointer  
Authentication

LOR

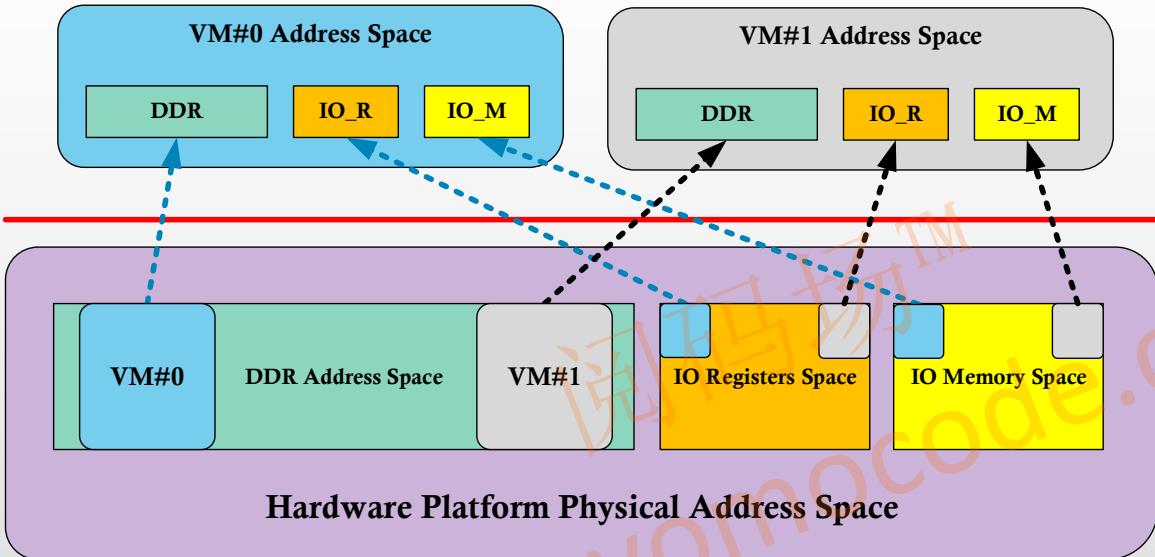
Virtual  
Memory

Auxiliary  
Control

# PRIVILEGE INSTRUCTIONS



# MEMORY VIRTUALIZATION



## 1. 内存虚拟化涵盖DDR内存和IO地址空间

- 地址空间虚拟化

## 2. 内存虚拟化的诉求

- 真实的物理地址空间是唯一的
- 每一个虚拟机都希望有个一个独立的物理地址空间

常用内存虚拟化的解决办法:

### 1. 地址空间分区

- 简单，但不灵活

### 2. 半虚拟化**DirectPaging / shadow page table**

- 虚拟地址模拟物理地址，性能好，需要修改**GuestOS**

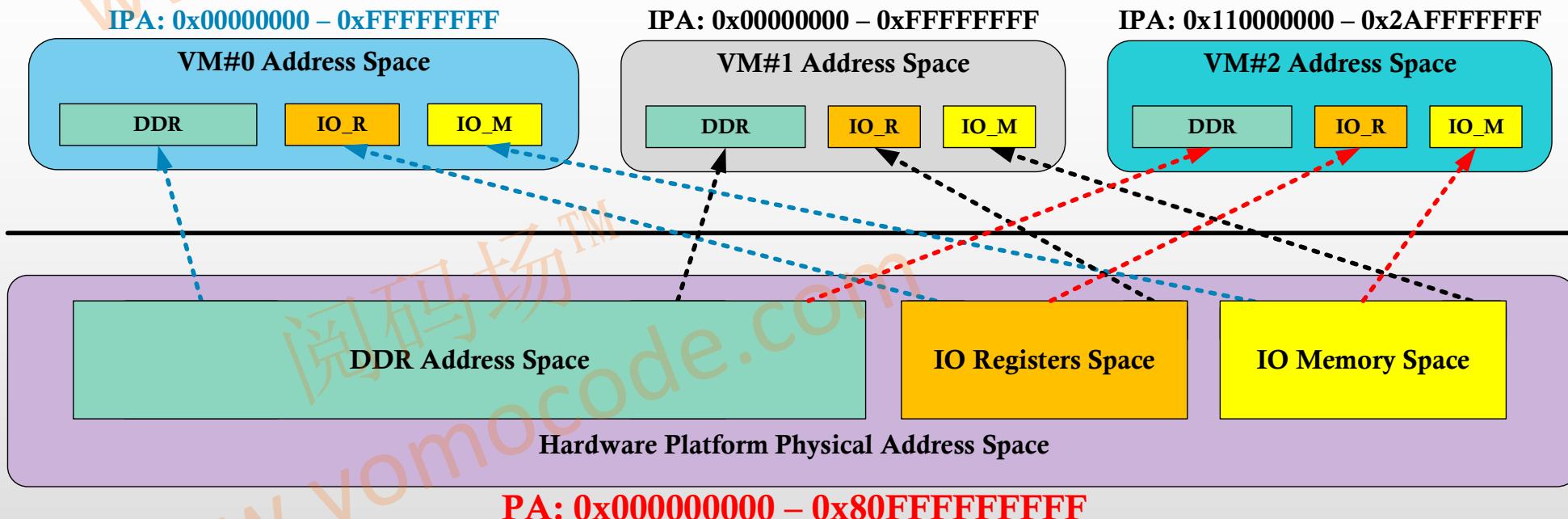
### 3. 二进制翻译地址转换

- 除了性能差和实现复杂，没什么其他缺点；）

### 4. **MMU**二级地址转换

- 通过**MMU**硬件进行地址翻译
- 对**GuestOS**保持透明，不需要修改原有**MMU**代码
- 运行时软件不介入翻译，简单高效

# 2 STAGE TRANSLATION



二级地址翻译：

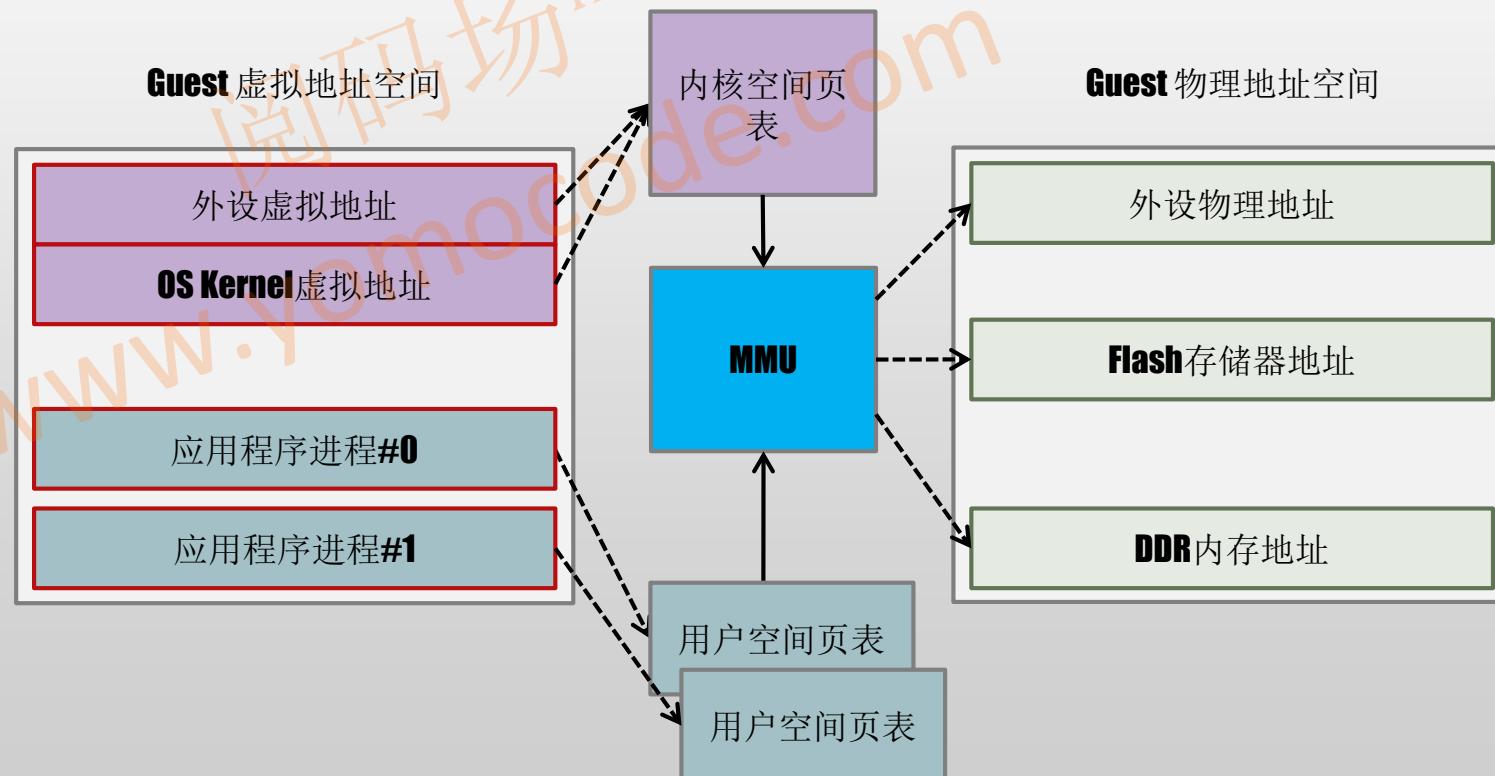
1. **Intermediate Physical Address (IPA) Guest物理地址**
2. 虚拟机的物理地址空间由**IPA**描述，不直接指向真实物理地址
3. 每个**VM**的**IPA**地址空间完全独立，可以相同的，可以重叠，也可以是完全不同
4. 通过两级地址翻译找到真实物理地址：

**Level 1:** 虚拟地址(**VA**) → **Guest物理地址 (IPA)**

**Level 2:** **Guest物理地址 (IPA)** → 真实物理地址 (**PA**)

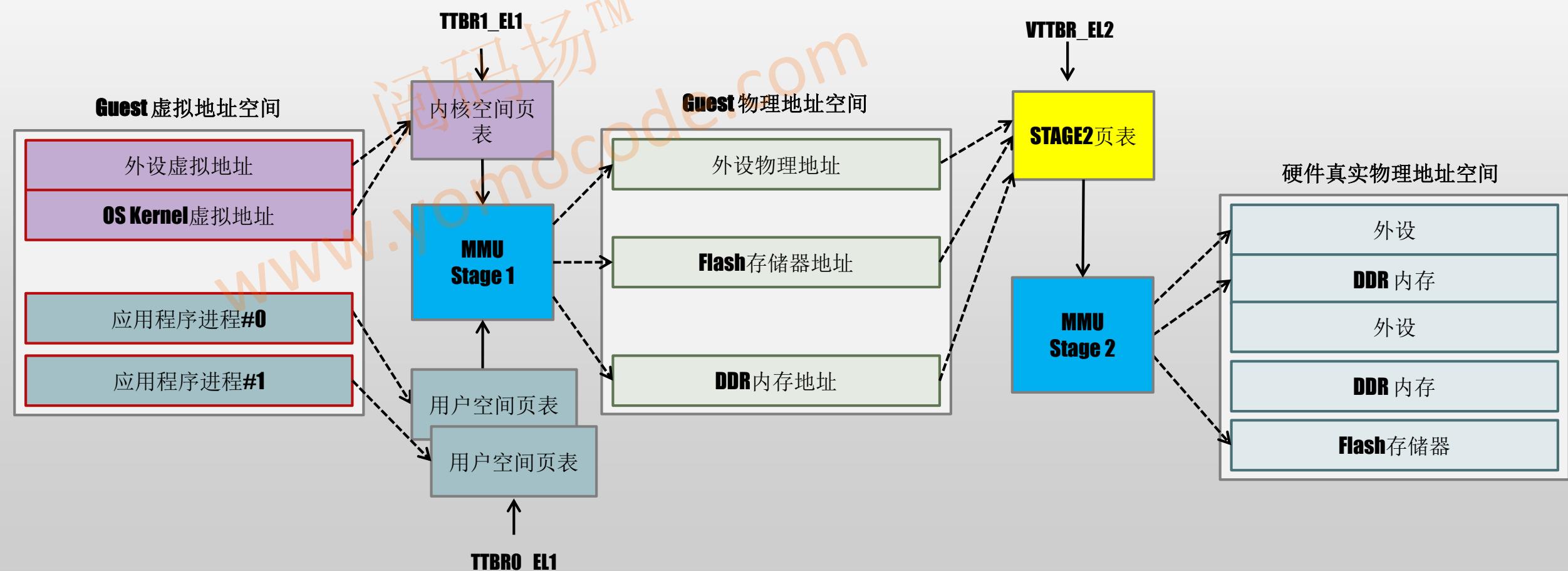
# STAGE#1 TRANSLATION

- 第一级翻译: 虚拟地址(VA) → Guest物理地址(IPA)
  - 这一级翻译仍然由GuestOS掌控, 和在物理机上运行的操作系统一样
  - 使用TTBRn\_EL1/TCR\_EL1寄存器和页表

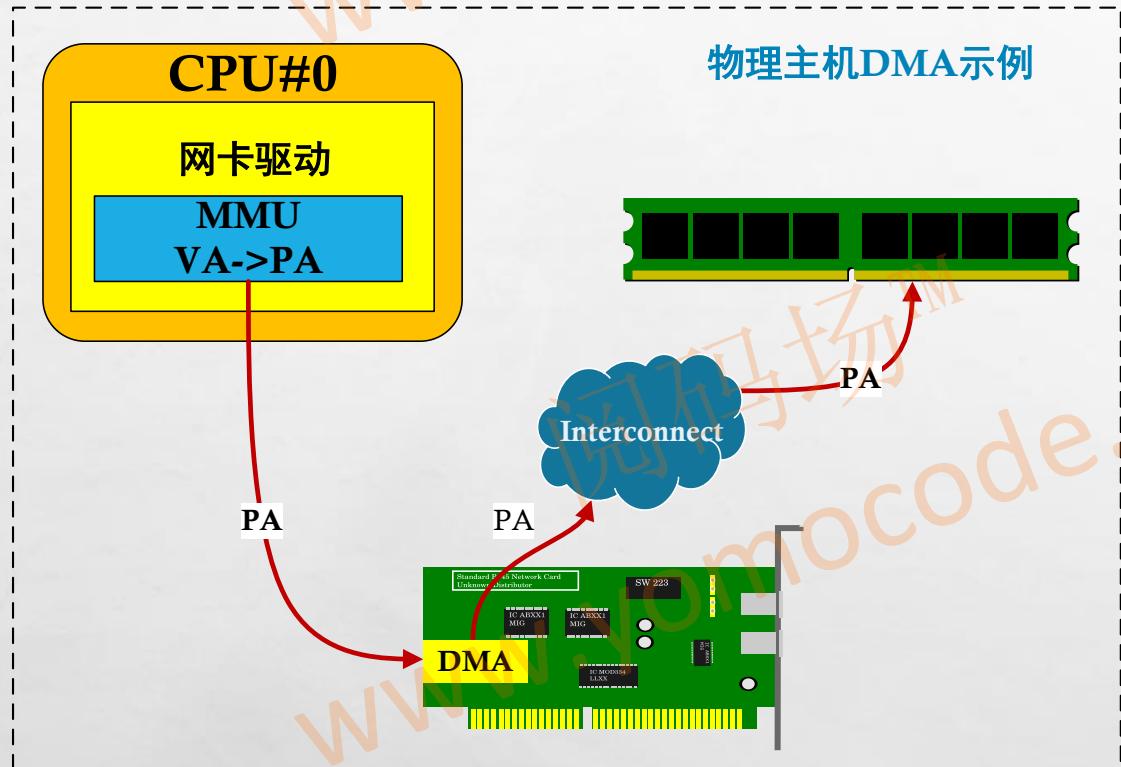


# STAGE#2 TRANSLATION

- 第二级翻译: Guest物理地址(IPA) → 物理地址(PA)
  - 这一级翻译由Hypervisor掌控
  - 使用VTTBR\_EL2/VTCR\_EL2寄存器和stage2页表

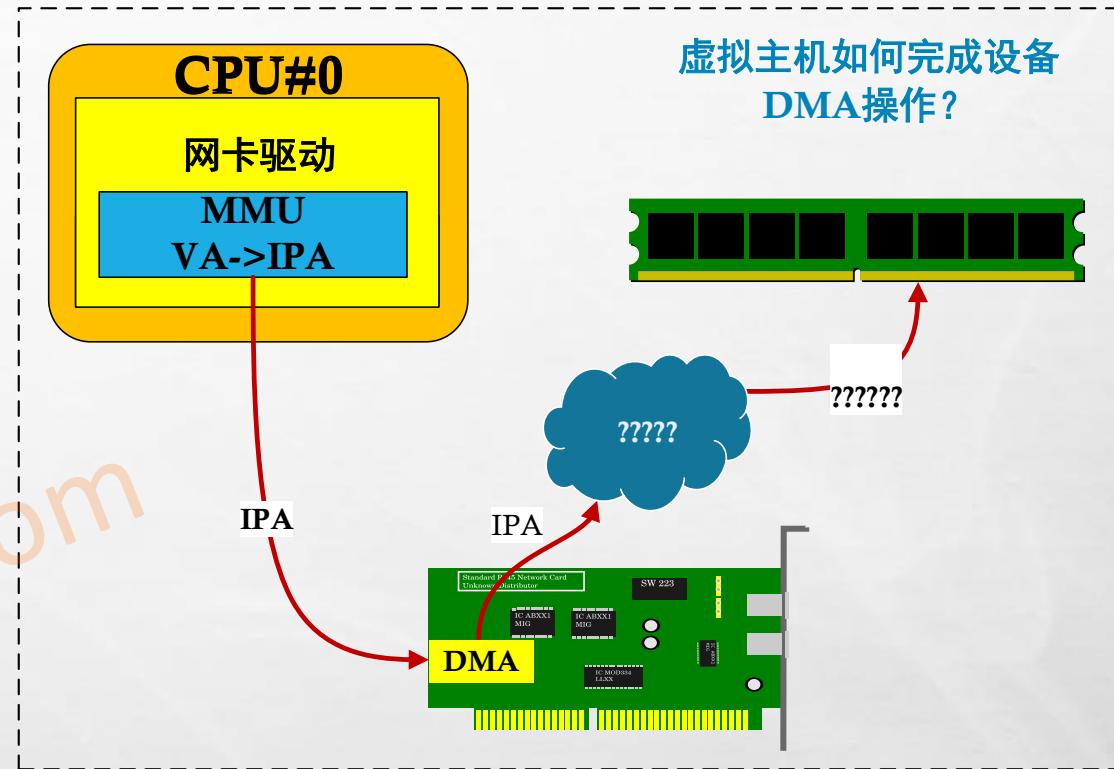


# DMA IN VIRTUAL MACHINE



## 物理机网卡DMA示例：

- 网卡驱动将数据的物理地址填入网卡**DMA**寄存器
- **DMA**模块通过物理地址访问物理内存



## 虚拟机网卡如何进行DMA？

- 网卡驱动将数据的**Guest**物理地址填入网卡**DMA**寄存器
- 通过**IPA**访问总线会产生故障

方法1：调用**hypervisor API** 直接转换**VA -> PA**

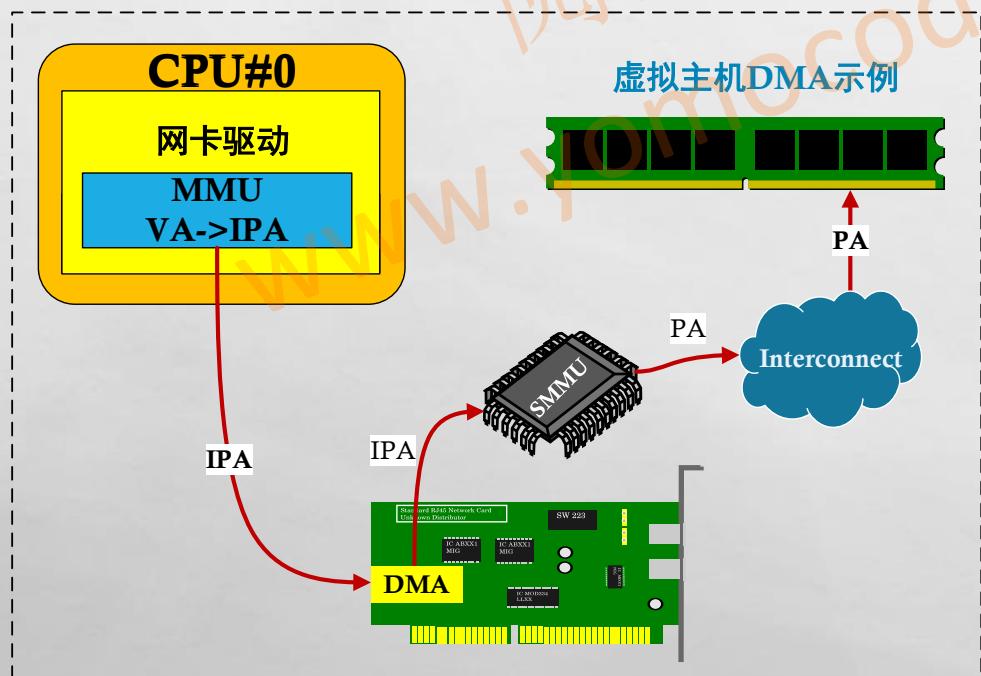
- 缺点：修改驱动，API调用的额外性能开销

方法2：为外设添加额外的虚拟化支持

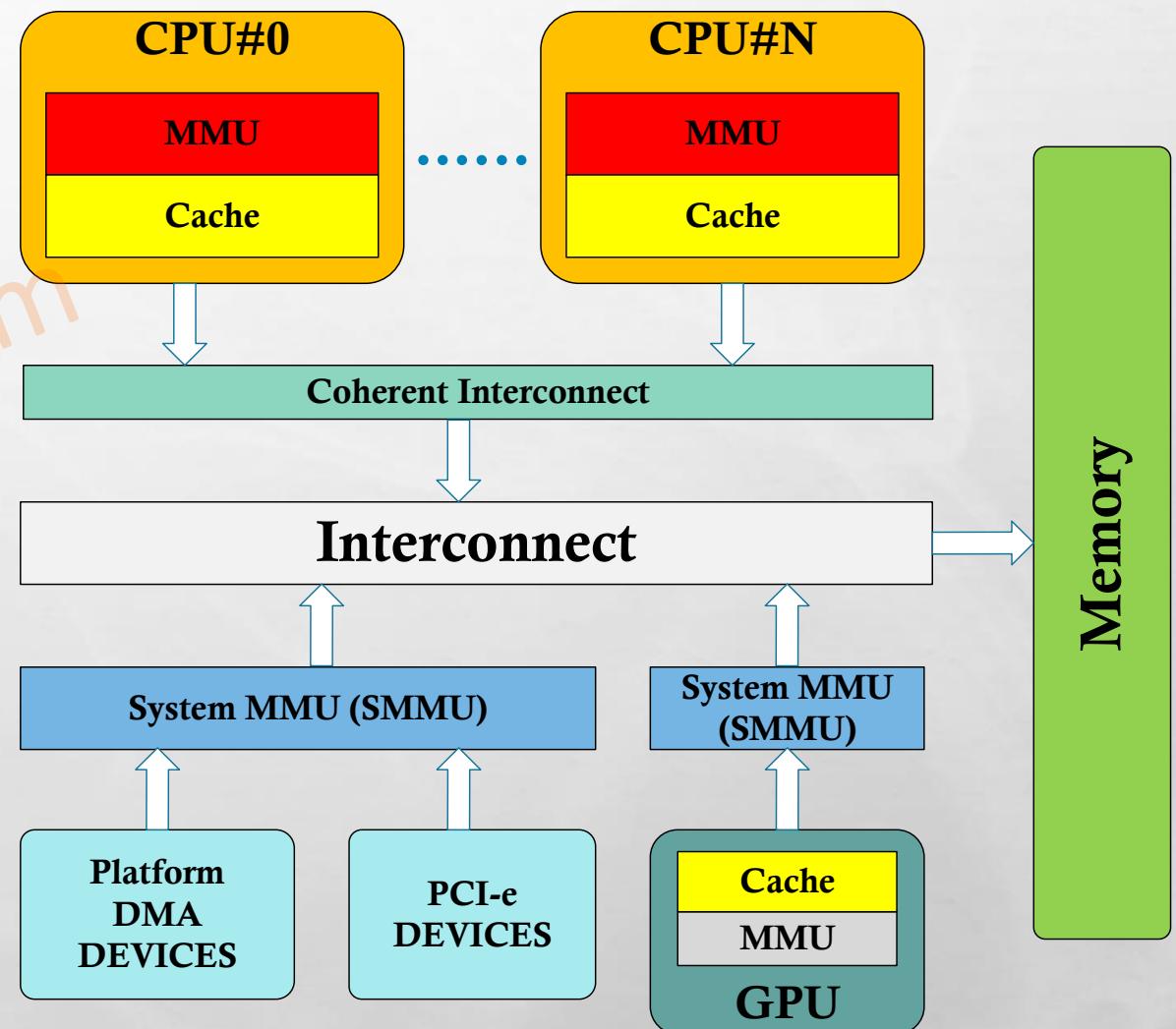
- 优点：驱动程序不感知，无额外API调用的性能开销

# SYSTEM MMU

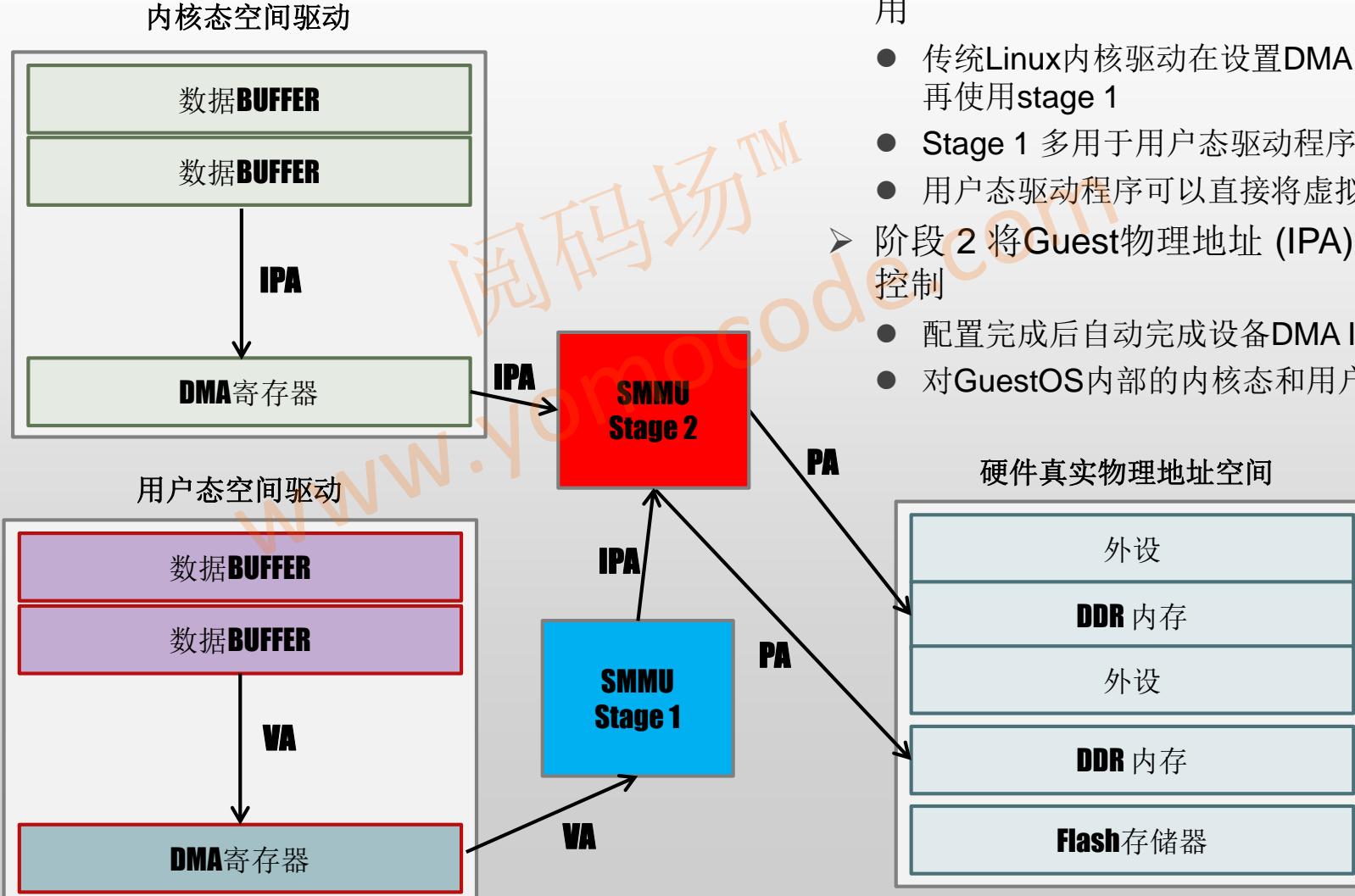
- MMU为CPU提供地址翻译服务和保护功能
- SMMU为系统中除CPU之外的任何具有DMA能力的设备提供地址翻译服务和保护功能
  - PCIe DMA设备
  - Platform DMA设备 (AMBA)
  - GPU/VE加速器



SOC MMU/SMMU连接示意图



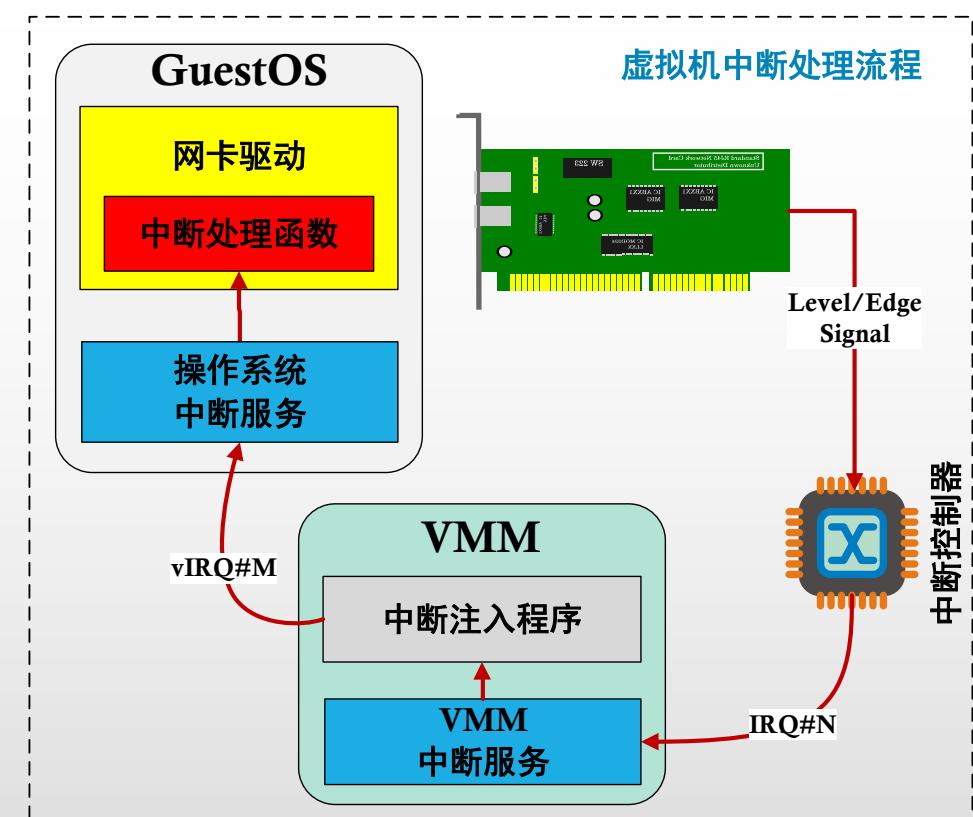
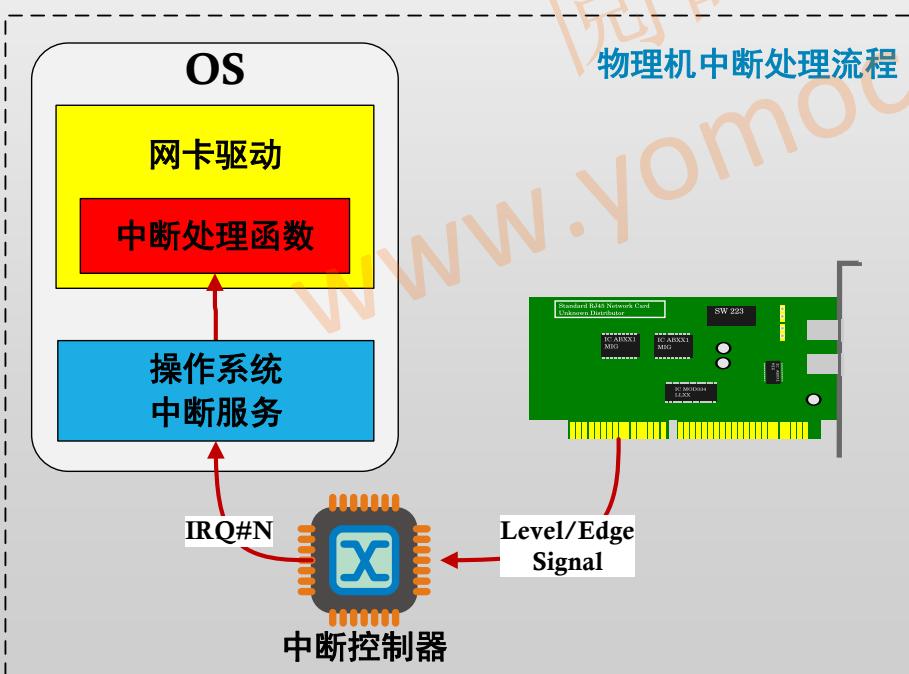
# SMMU 2 STAGE TRANSLATION



- **SMMU**设计为在虚拟化系统中使用，支持两个独立的地址转换阶段
  - 阶段 1 实施虚拟地址 (VA) 到 Guest 物理地址 (IPA) 的转换，由 GuestOS 使用
    - 传统 Linux 内核驱动在设置 DMA 前，已经通过 MMU 完成了 VA 到 IPA 的转换，一般不再使用 stage 1
    - Stage 1 多用于用户态驱动程序，比如 DPDK
    - 用户态驱动程序可以直接将虚拟地址填入 DMA 寄存器
  - 阶段 2 将 Guest 物理地址 (IPA) 转换为物理地址 (PA)，由 Hypervisor/VMM 控制
    - 配置完成后自动完成设备 DMA IPA 到 PA 的转换
    - 对 GuestOS 内部的内核态和用户态驱动都是透明的

# INTERRUPT VIRTUALIZATION

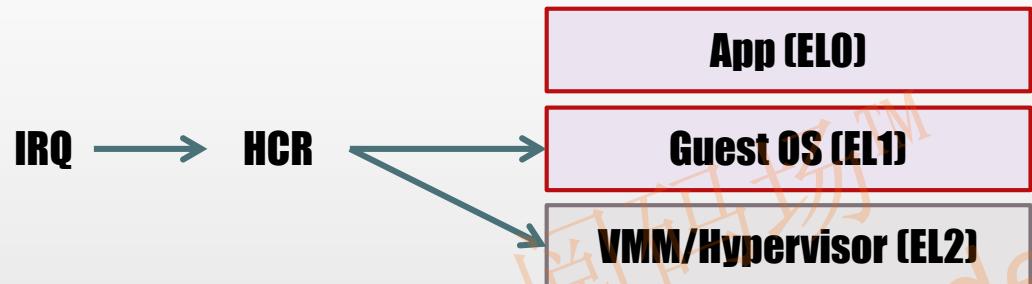
- 物理主机，设备中断的处理流程
  - 设备产生一个水平/边缘触发中断信号
  - 中断控制器响应该信号，让CPU EL1进入中断异常模式
  - OS调用中断服务程序，通过IRQ number找到对应的驱动中断处理函数
  - 完成中断处理



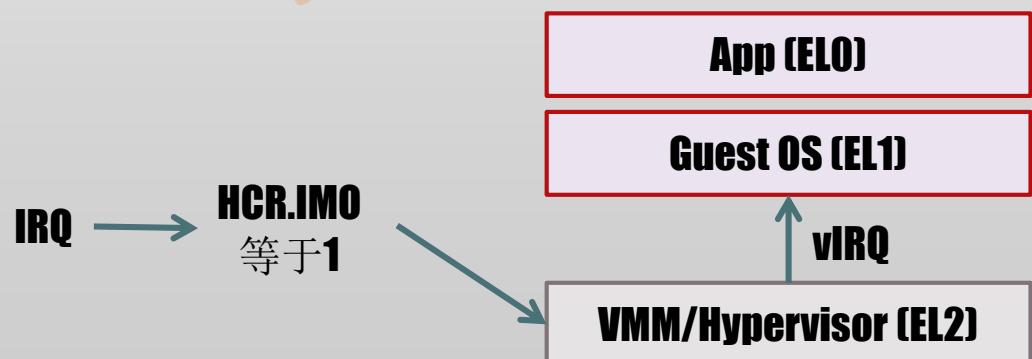
- 虚拟化系统中，设备中断的处理流程
  - 设备产生一个水平/边缘触发中断信号
  - 中断控制器响应该信号，让CPU EL2进入中断异常模式
  - CPU调用VMM 中断服务程序，通过IRQ number找打对应的GuestOS，通过中断注入程序向GuestOS注入virtual IRQ
  - GuestOS EL1 CPU进入中断异常模式
  - GuestOS调用中断服务程序，通过virtual IRQ number找到对应的驱动中断处理函数
  - 完成中断处理

# VIRTUAL EXCEPTIONS

在Armv8中，中断可以通过**HCR**寄存器的配置，选择让哪一个**Exception Level (EL)**来处理中断。



在典型的**Arm**虚拟化系统里，一般都是让**VMM**所在的**EL2**来处理来自真实设备的中断，然后通过虚拟化异常(**virtual exception**)向**GuestOS**注入中断。



## 1. Armv8支持3种virtual exceptions:

- 虚拟系统错误(**virtual Serror**)
- 虚拟中断(**virtual IRQ**)
- 虚拟快速中断(**virtual FIQ**)

2. **Virtual IRQ**只能由**EL2 hypervisor/VMM**发起，然后由**EL1 GuestOS**接受。

3. **Hypervisor**本身并不会收到**virtual IRQ**

## 4. Arm在硬件上提供了两种机制来发起**virtual IRQ**:

- 通过**GIC**中断控制器 (**Generic Interrupt Controller**)
- 通过**CPU EL2**的**hypervisor**配置寄存器**HCR\_EL2.VI**

# GIC VIRTUALIZATION EXTENSION

**GIC** 通用模块:

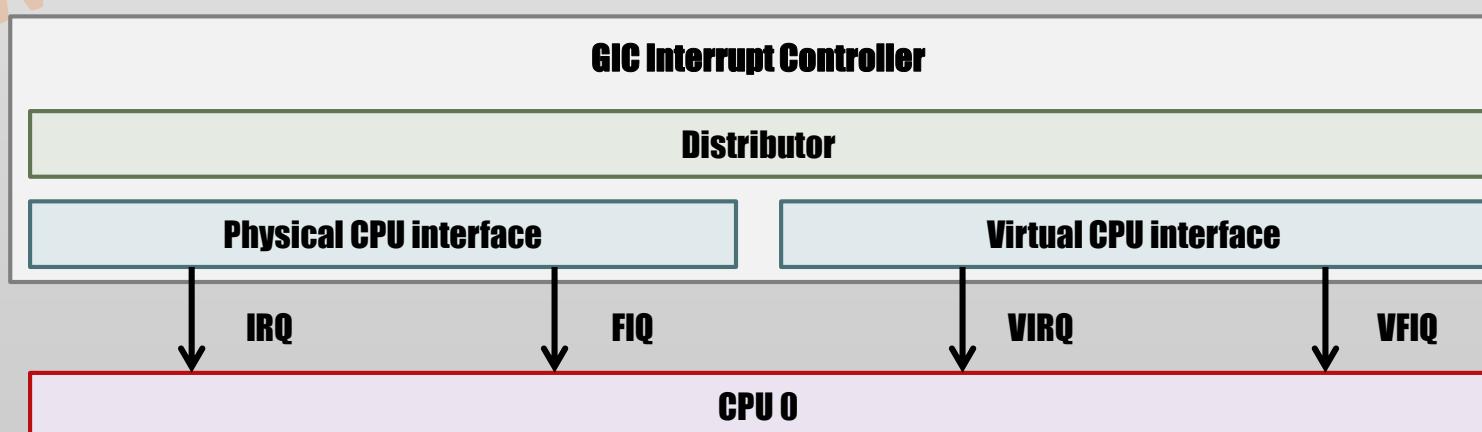
- **Distributor** – 和设备连接，用来配置中断的开关、属性等
- **Physical CPU interface** – 和**CPU**接口，触发**CPU**进入中断异常

**GIC** 虚拟化扩展模块:

- **Virtual CPU interface** – 和**CPU**的接口，触发虚拟中断
- **Hypervisor interface** – **VMM**用来给虚拟机配置**Virtual CPU interface**

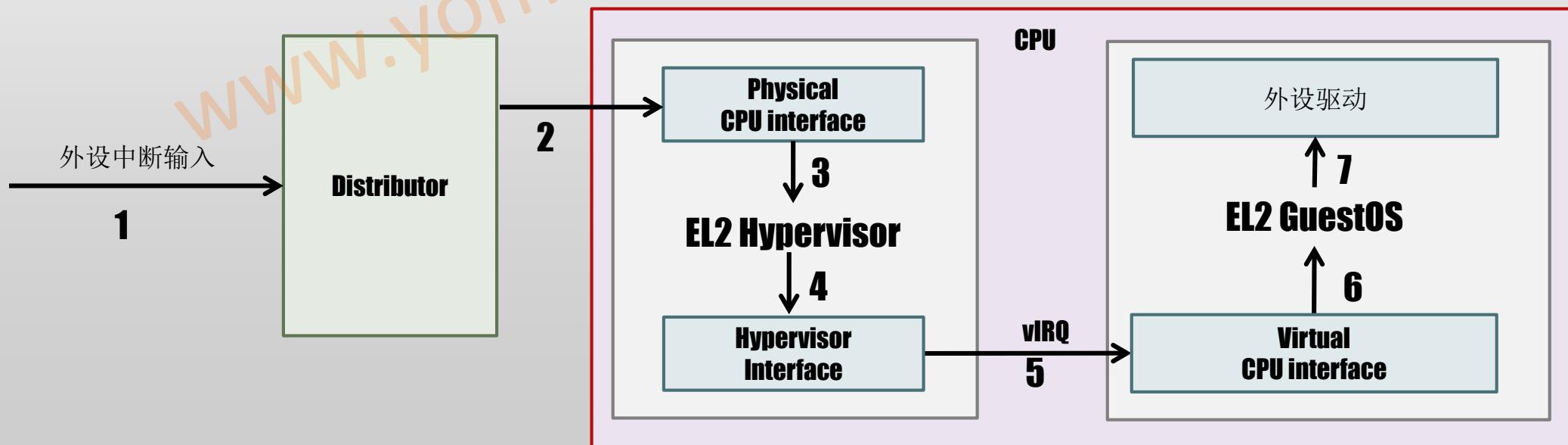
在虚拟化系统中，**physical CPU interface** 和 **hypervisor interface** 属于 **VMM** 控制

- **VMM** 通过 **physical CPU interface** 接收物理中断
- **VMM** 通过 **hypervisor interface** 配置 **virtual CPU interface** 向虚拟机注入虚拟中断
- **GuestOS** 通过 **virtual CPU interface** 接收虚拟中断并处理



# INJECT VIRTUAL IRQ - GIC

1. 外设中断输入源到达**GIC**
2. **GIC**向**CPU**发送一个物理中断信号
3. **CPU**会切换到**EL2**, 然后**HYPERVERISOR**通过**PHYSICAL CPU INTERFACE**获取中断的信息
4. 然后**HYPERVERISOR**通过**HYPERVERISOR INTERFACE**的**LR**寄存器写入对应虚拟中断的状态**(PENDING)**
5. **VIRTUAL CPU INTERFACE**根据这个**PENDING**会向**CPU**发送一个**VIRTUAL IRQ**
6. 虚拟机被调度, 虚拟机所在的**CPU**会获取这个**VIRTUAL IRQ**造成的中断异常
7. **GUESTOS**通过**VIRTUAL CPU INTERFACE**获取**VIRTUAL IRQ**的信息并处理



# EVOLUTION OF ARM VIRTUALIZATION

## Armv8.1 – VHE

改进了对**TYPE2  
hypervisors**的支持

## Armv8.3-A NV

加入了对嵌套虚拟化的支持

## Armv8.4-A SECURE EL2

在**Security State**引入了**EL2**, 支持在安全地址空间虚拟化

谢谢!  
**THANK YOU!**

