中山大學 软件工程学院

SUN YAT-SEN UNIVERSITY SCHOOL OF SOFTWARE ENGINEERING

1924-2024 中山大學 世纪华诞 100th ANNIVERSARY SUN YAT-SEN UNIVERSITY

系统虚拟化II

SSE202/204: 操作系统原理

苏玉鑫

suyx35@mail.sysu.edu.cn

助教：龙玉丹 单诗雯 毛晨希 沈志轩 郑灿峰 胡伟峰

版权信息

- 部分内容来自：上海交通大学并行与分布式系统研究所操作系统课件
  - https://ipads.se.sjtu.edu.cn/courses/os/
- 其它参考资料：
  - 清华大学操作系统公开课
    - https://open.163.com/newview/movie/courseintro?newurl=ME1NSA351
    - 介绍标准内容，适合考研
  - 南京大学计算机软件研究所
    - http://jyywiki.cn/OS/2025/
    - https://space.bilibili.com/202224425/channel/collectiondetail?sid=192498
    - 比较有趣

大纲

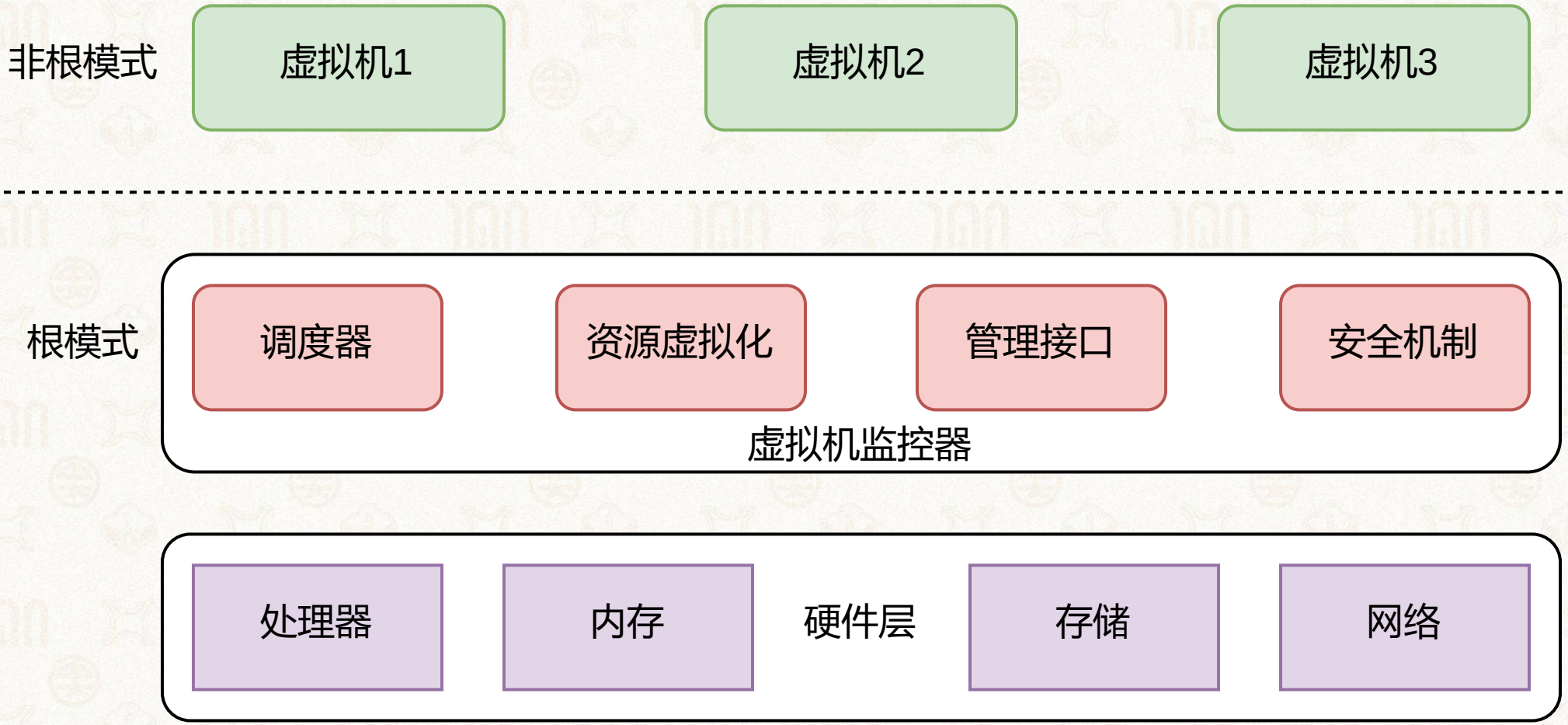
- 虚拟化概述
  - 为什么要用虚拟化
  - 虚拟化的优势
- 什么是系统虚拟化
  - 虚拟机监控器
  - 虚拟化的类型
- CPU虚拟化
  - 下陷
  - 三种软件虚拟化方法
  - 硬件虚拟化
- 内存虚拟化
  - 影子页表
  - 直接页表
  - 硬件虚拟化
- I/O 虚拟化
  - 设备模拟
  - 半虚拟化
  - 设备直通
- 案例：QEMU/KVM

方法4：硬件虚拟化

- x86和ARM都引入了全新的虚拟化特权级

- x86引入了root模式和non-root模式
  - Intel推出了VT-x硬件虚拟化扩展
  - Root模式是最高特权级别，控制物理资源
  - VMM运行在root模式，虚拟机运行在non-root模式
  - 两个模式内都有4个特权级别：Ring0~Ring3

Intel VT-x硬件虚拟化架构



虚拟机控制结构

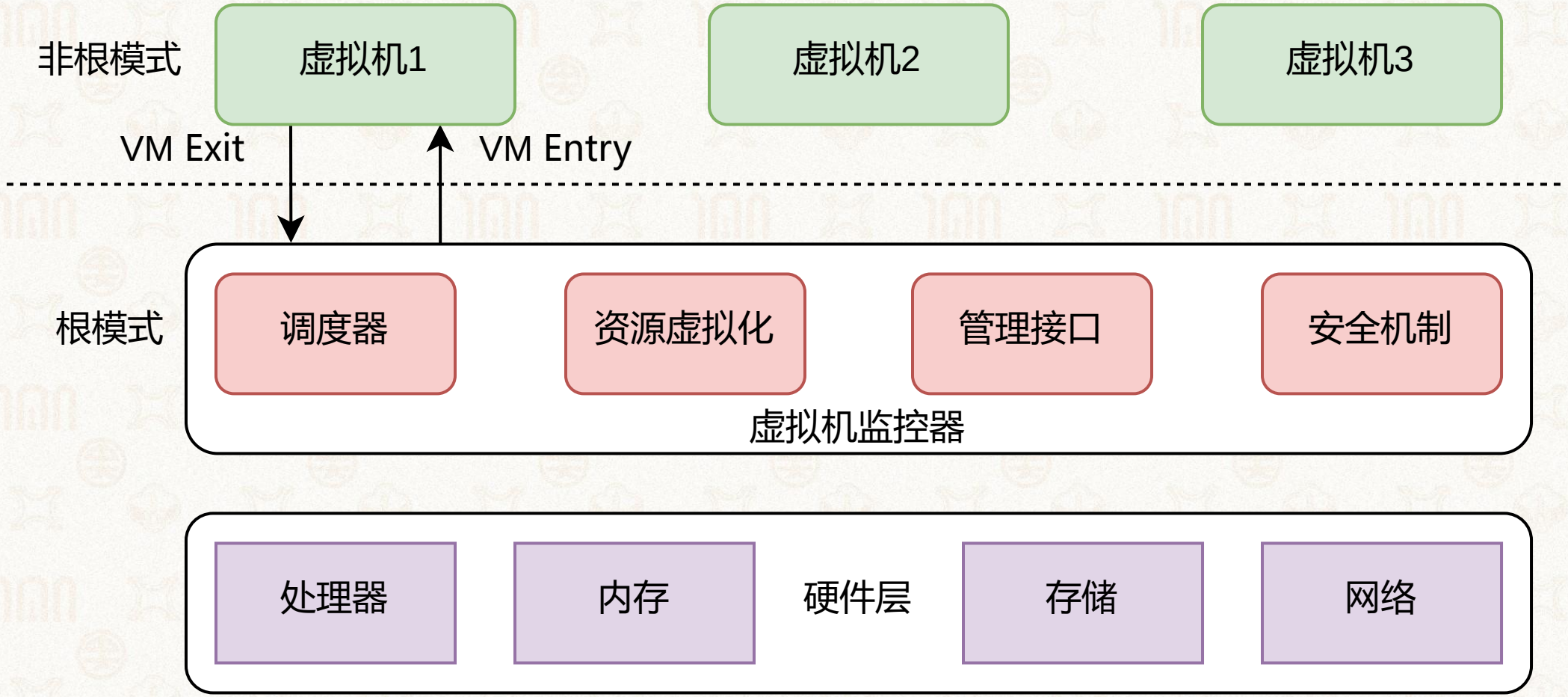
- Virtual Machine Control Structure, VMCS

- VMM提供给硬件的内存页（4KB）
  - 记录与当前VM运行相关的所有状态

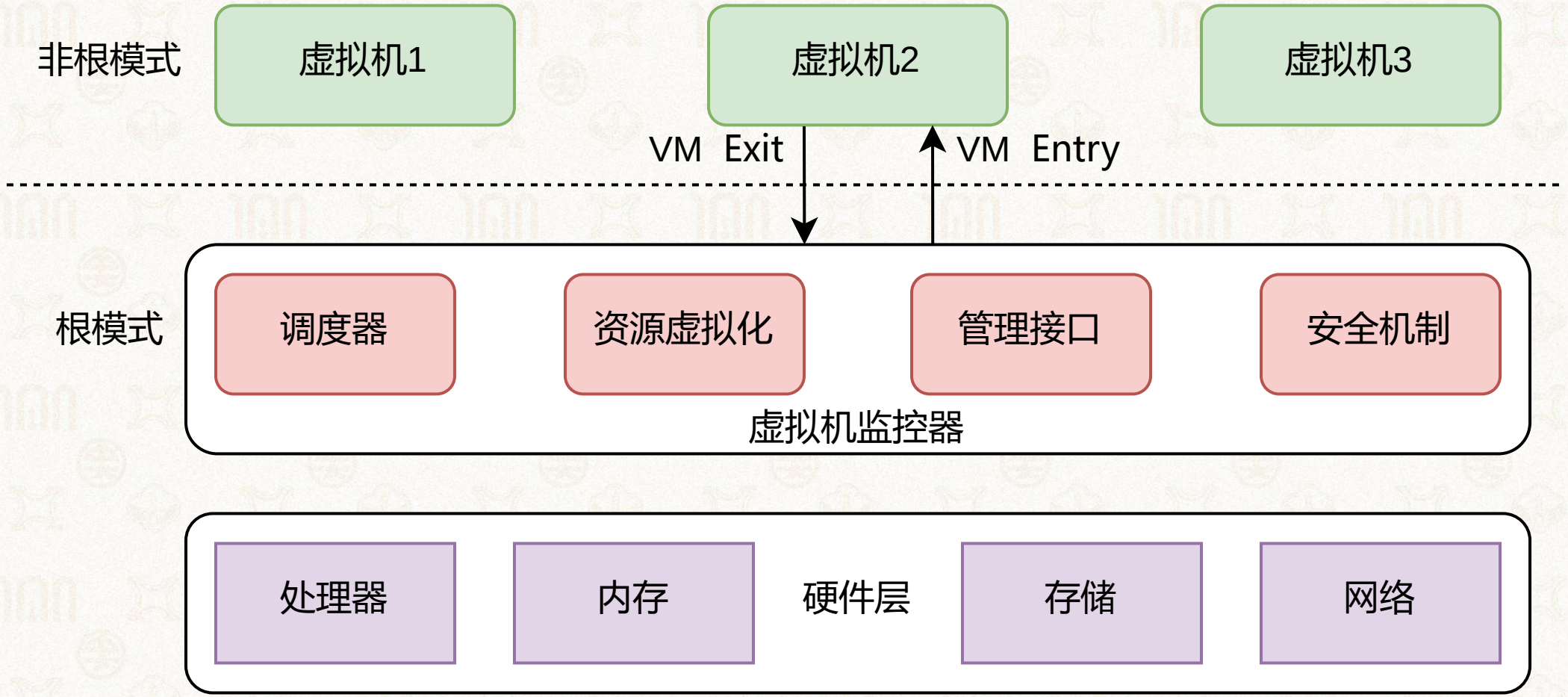
- VM Entry
  - 硬件自动将当前CPU中的VMM状态保存至VMCS
  - 硬件硬件自动从VMCS中加载VM状态至CPU中

- VM Exit
  - 硬件自动将当前CPU中的VM状态保存至VMCS
  - 硬件自动从VMCS加载VMM状态至CPU中

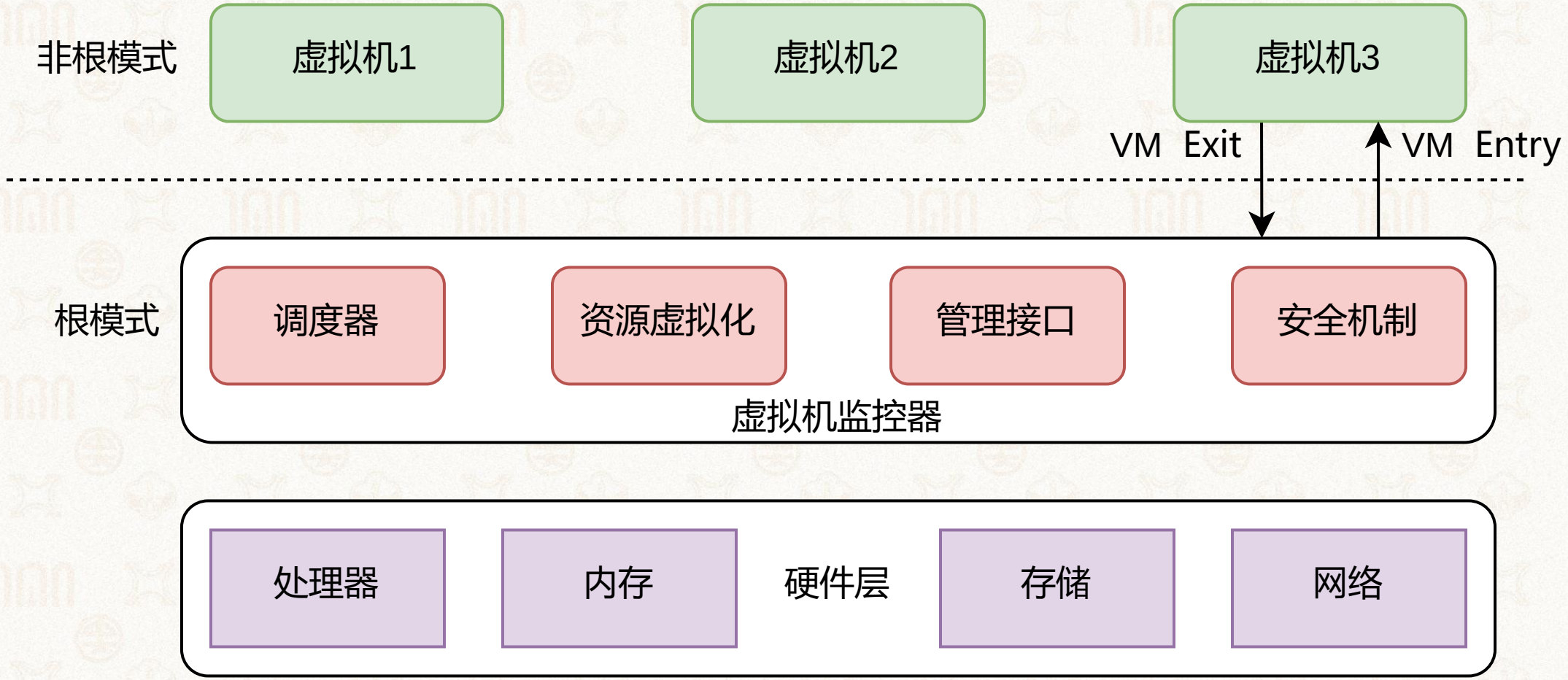
Intel VT-x硬件虚拟化架构



Intel VT-x硬件虚拟化架构



Intel VT-x硬件虚拟化架构

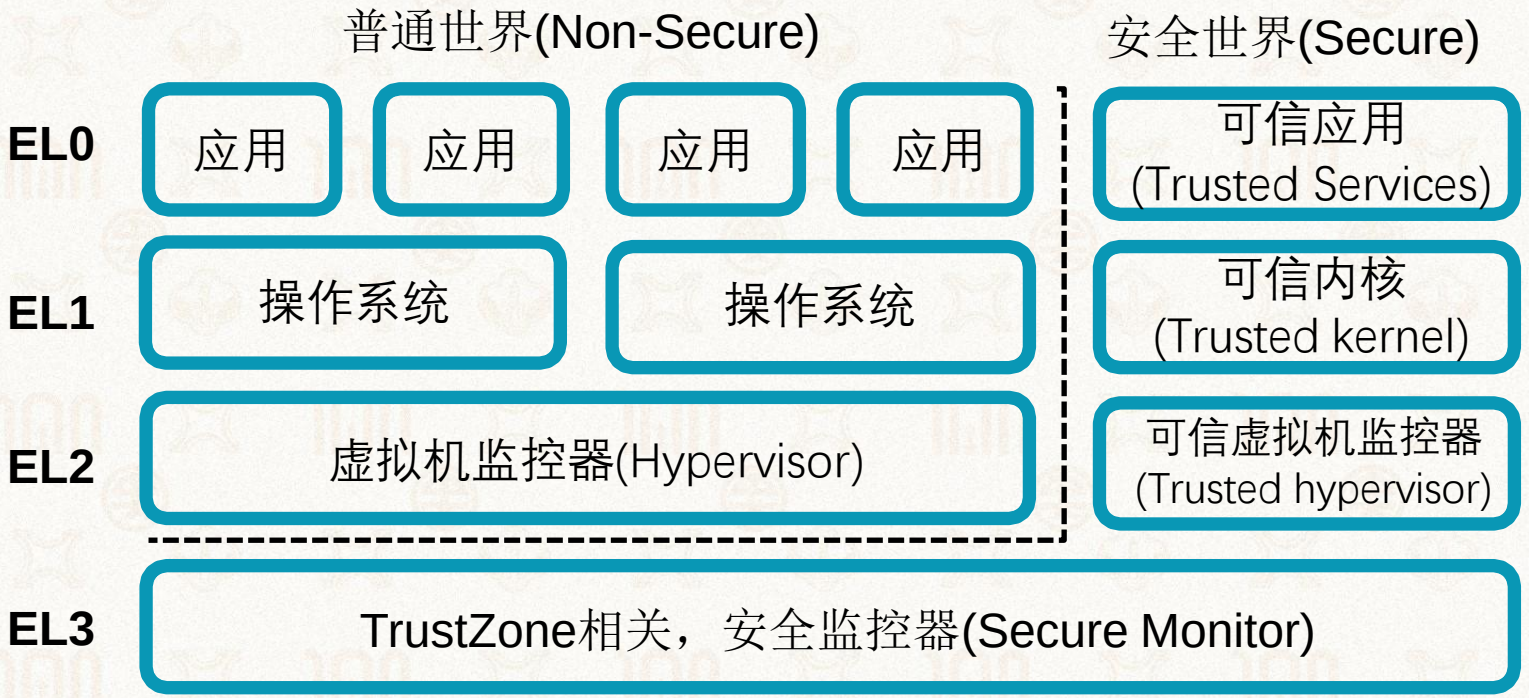


虚拟机控制结构

- 包含6个部分
- Guest-state area:
  - 发生VM exit时，CPU的状态会被硬件自动保存至该区域;
  - 发生VM Entry时，硬件自动从该区域加载状态至CPU中
- Host-state area:
  - 发生VM exit时，硬件自动从该区域加载状态至CPU中;
  - 发生VM Entry时，CPU的状态会被自动保存至该区域
- VM-execution control fields： 控制Non-root模式中虚拟机的行为
- VM-exit control fields：控制VM exit的行为
- VM-entry control fields：控制VM entry的行为
- VM-exit information fields：VM Exit的原因和相关信息（只读区域）

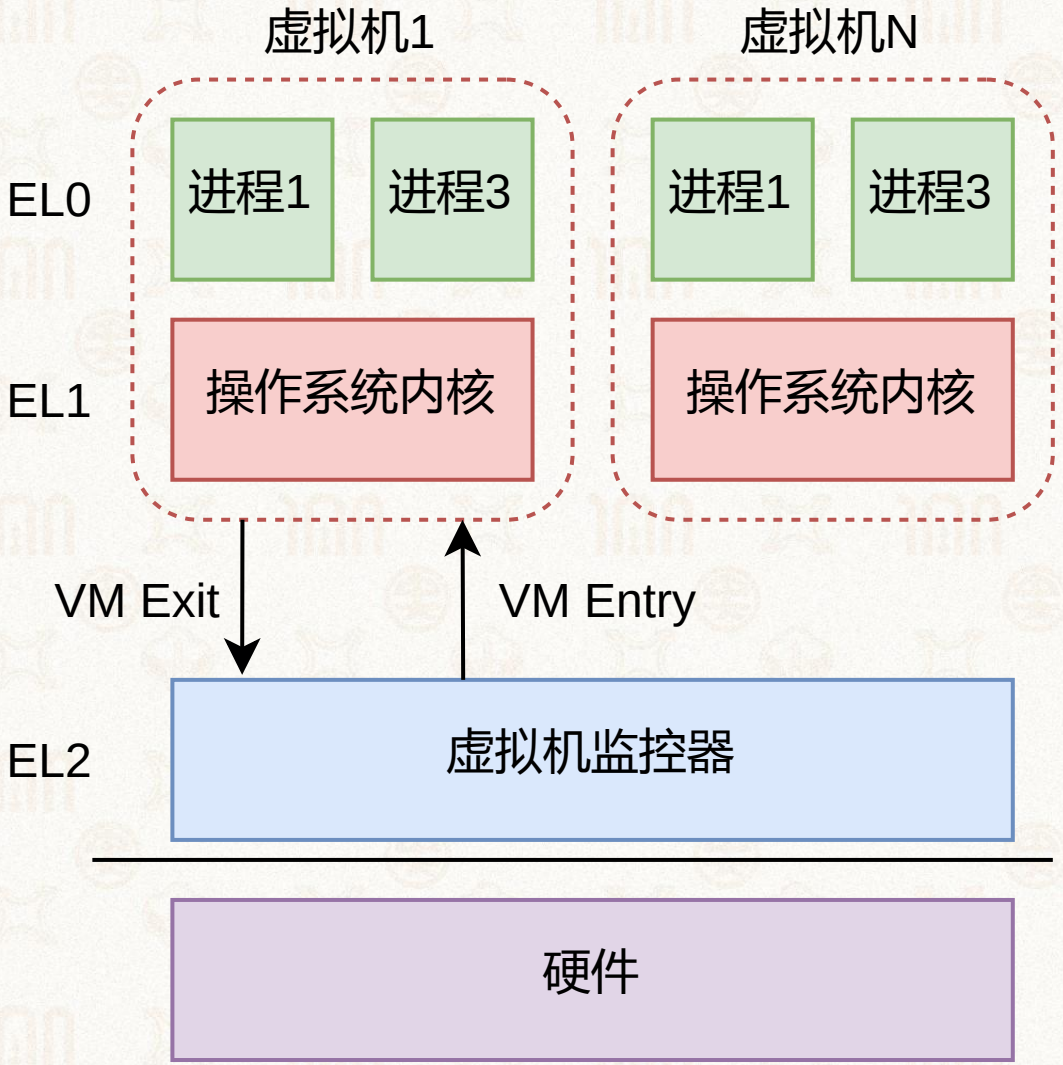
方法4：硬件虚拟化

- ARM引入了EL2
  - VMM运行在EL2
  - EL2是最高特权级别，控制物理资源
  - VMM的操作系统和应用程序分别运行在EL1和EL0



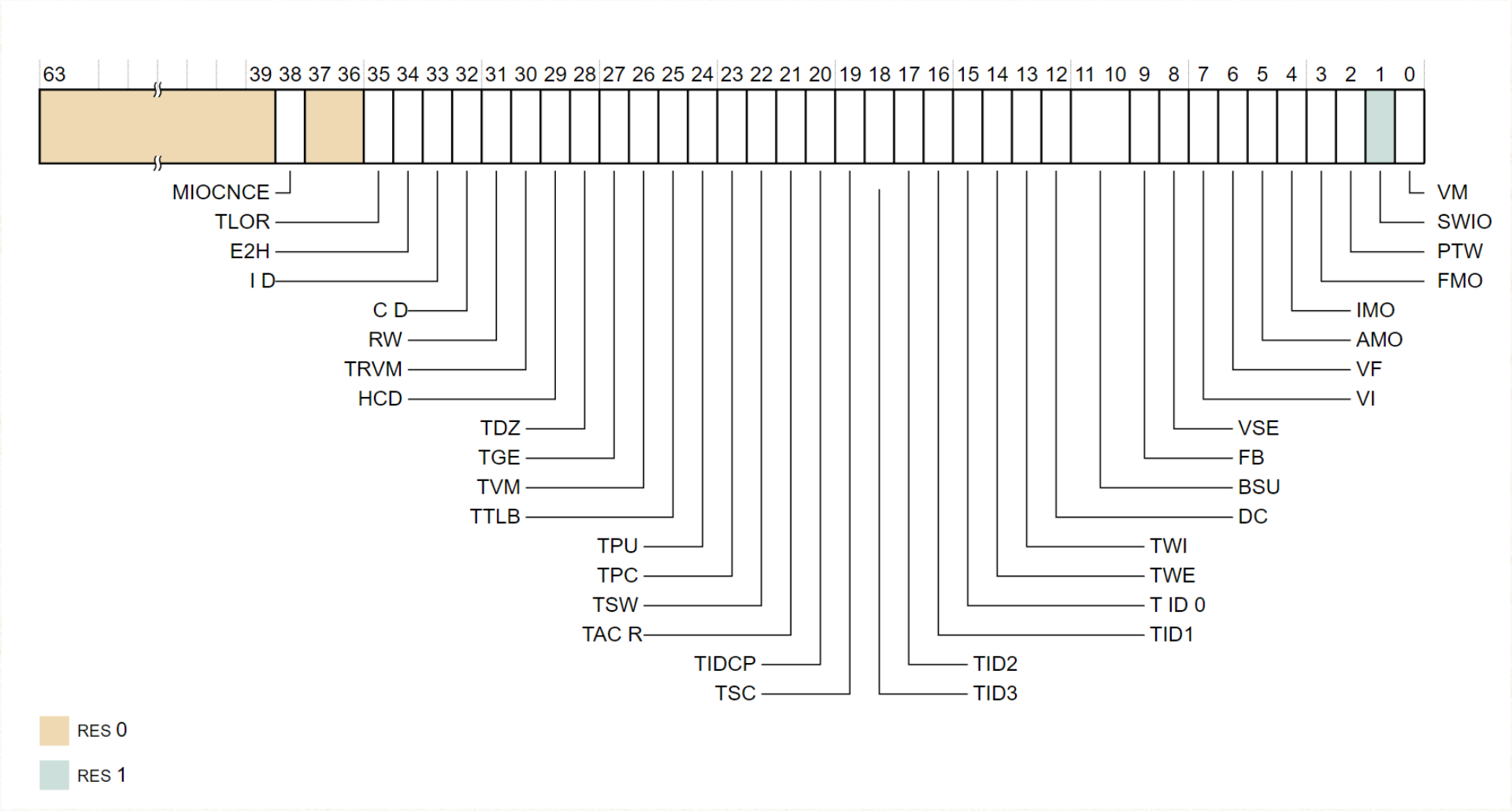
ARM的处理器虚拟化

- VM Entry
  - 使用ERET指令从VMM进入VM
  - 在进入VM之前，VMM需要主动加载VM状态
    - VM内状态：通用寄存器、系统寄存器、
    - VM的控制状态：HCR_EL2、VTTBR_EL2等
- VM Exit
  - 虚拟机执行敏感指令或收到中断等
  - 以Exception、IRQ、FIQ的形式回到VMM
    - 调用VMM记录在vbar_el2中的相关处理函数
  - 下陷第一步：VMM主动保存所有VM的状态



HCR_EL2 系统寄存器

- Hypervisor Configuration Register, EL2



https://developer.arm.com/documentation/ddi0595/2020-12/AArch64-Registers/HCR-EL2--Hypervisor-Configuration-Register?lang=en

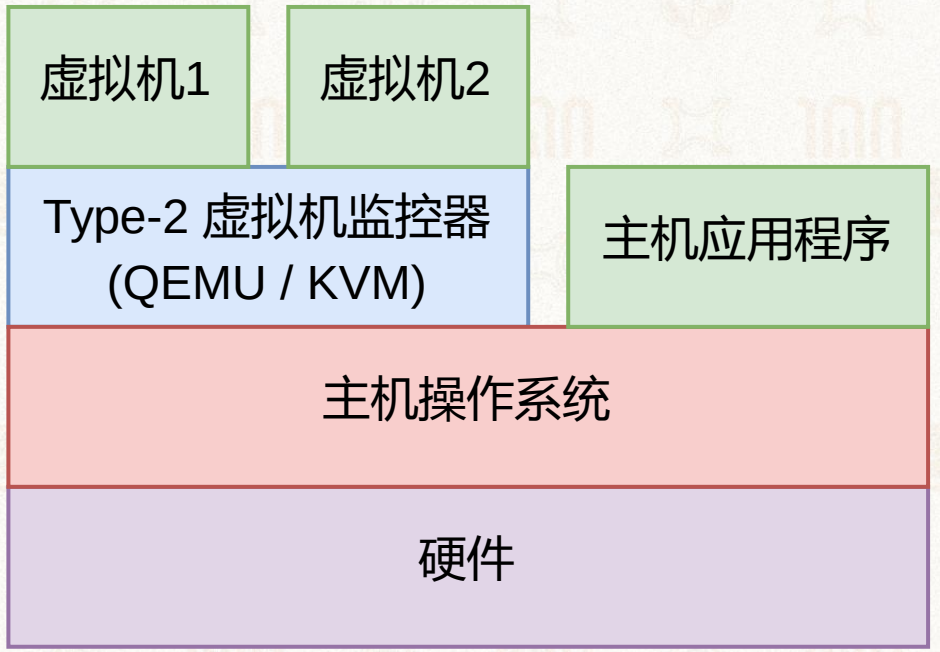
HCR_EL2 系统寄存器

- HCR_EL2：VMM控制VM行为的系统寄存器
  - VMM有选择地决定VM在某些情况时下陷
  - 和VT-x VMCS中VM-execution control area类似

- 在VM Entry之前设置相关位，控制虚拟机行为
  - TRVM(32位)和TVM(26位): VM读写内存控制寄存器是否下陷，例如SCTRL_EL1、TTBR0_EL1
  - TWE(14位)和TWI(13位)：执行WFE和WFI指令是否下陷
  - AMO(6位)/IMO(5位)/FMO(4位)：Exception/IRQ/FIQ是否下陷
  - VM(0位): 是否打开第二阶段地址翻译

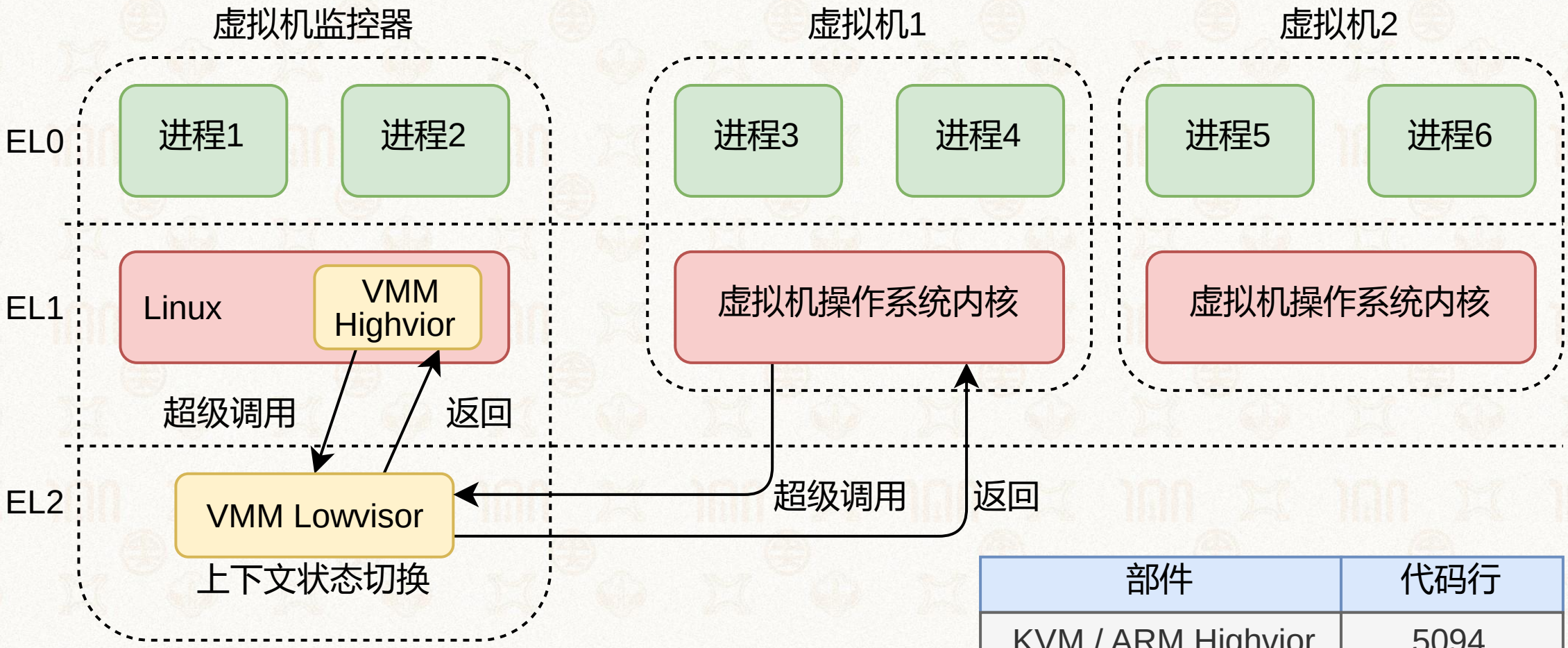
ARM硬件虚拟化: Type-2虚拟化

- 增加EL2特权级
- EL2只能运行VMM，不能运行一般操作系统内核
  - OS一般只使用EL1的寄存器，在EL2中不存在对应的寄存器
    - EL1：TTBR0_EL1、TTBR1_EL1
    - EL2：TTBR_EL2
  - EL2不能与EL0共享内存
- 因此：无法在EL2中运行Type-2虚拟机监控器的Host OS
  - 或者说，Host OS需要大量修改才能运行在EL2



ARM硬件虚拟化: Type-2虚拟化

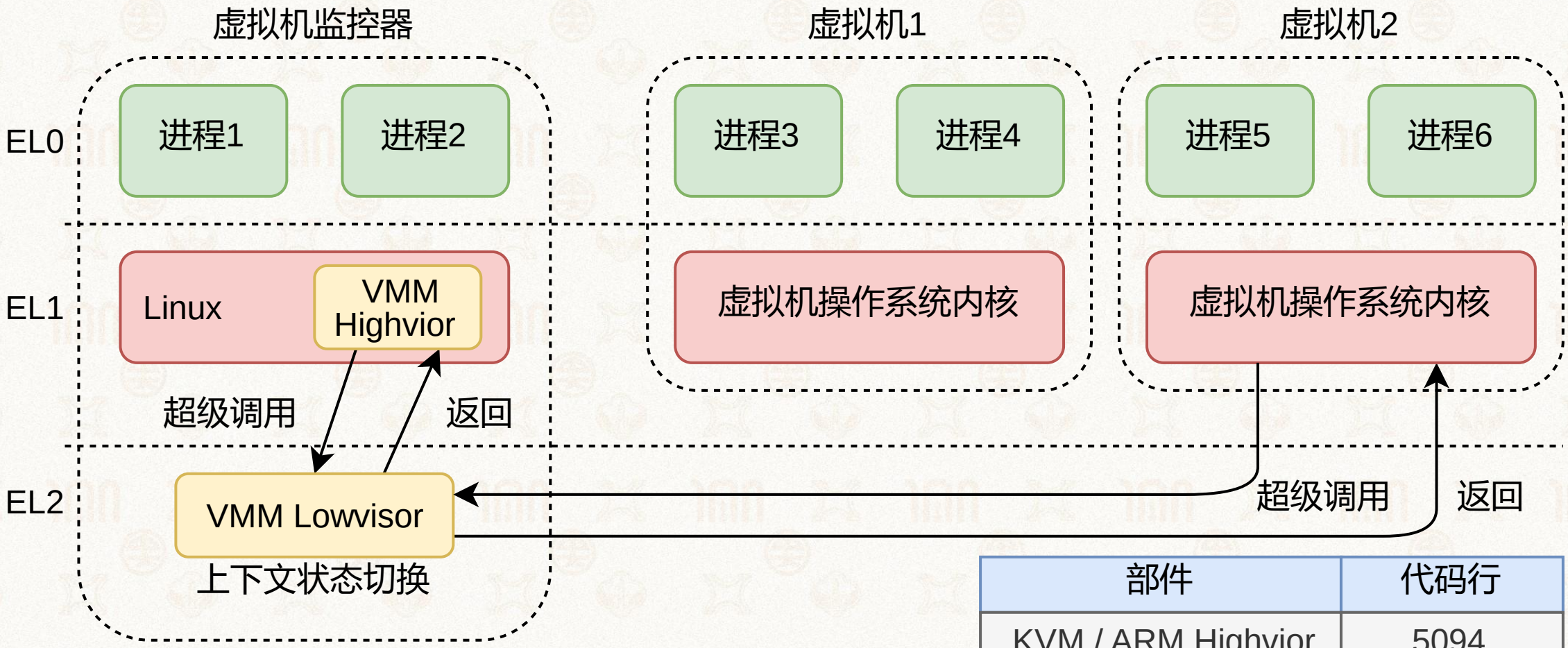
- EL2只能运行较少内容



部件	代码行
KVM / ARM Highvior	5094
KVM / ARM Lowvior	718

ARM硬件虚拟化: Type-2虚拟化

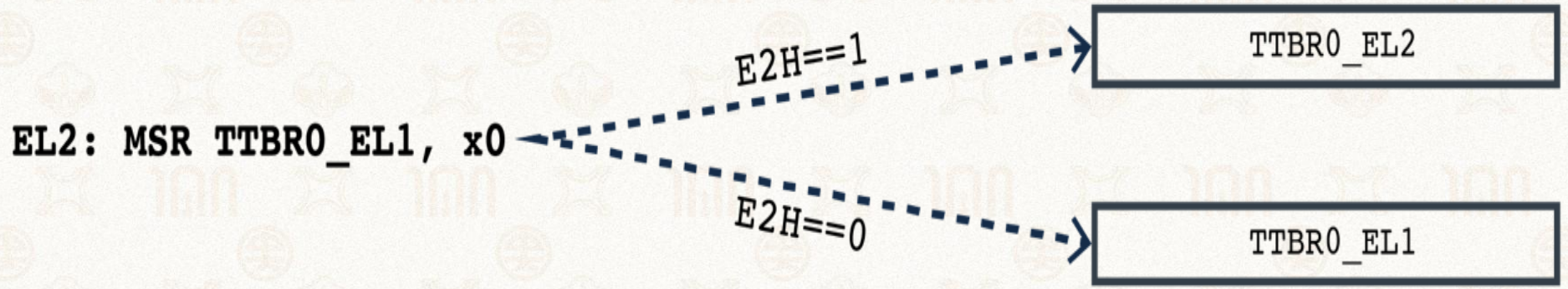
- EL2只能运行较少内容



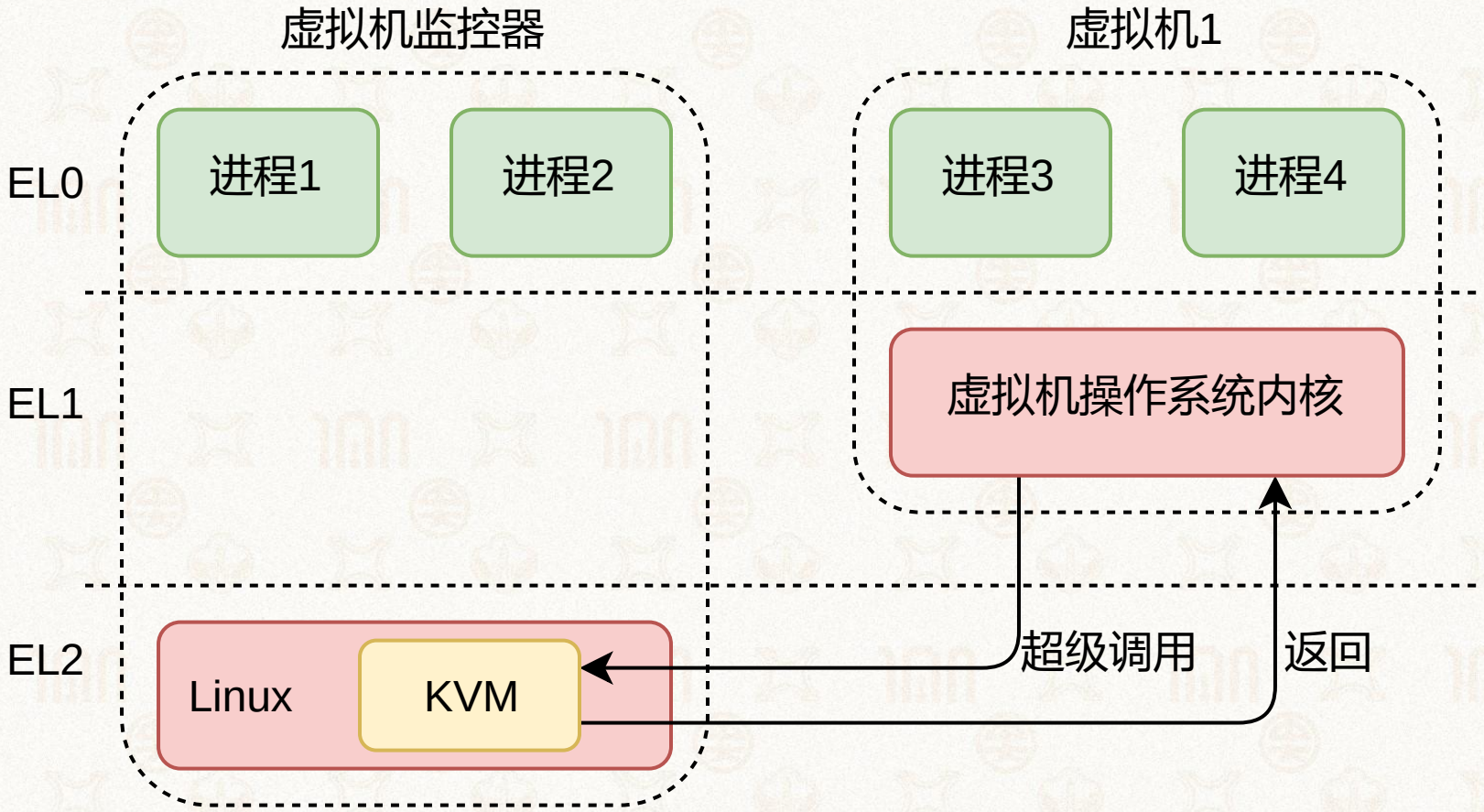
部件	代码行
KVM / ARM Highvior	5094
KVM / ARM Lowvior	718

ARMv8.1中的Type-2 VMM架构

- ARMv8.1
  - 推出Virtualization Host Extensions(VHE)，在HCR_EL2.E2H打开
    - 寄存器映射
    - 允许与EL0共享内存
  - 使EL2中可直接运行未修改的操作系统内核（Host OS）



ARMv8.1中的Type-2 VMM架构



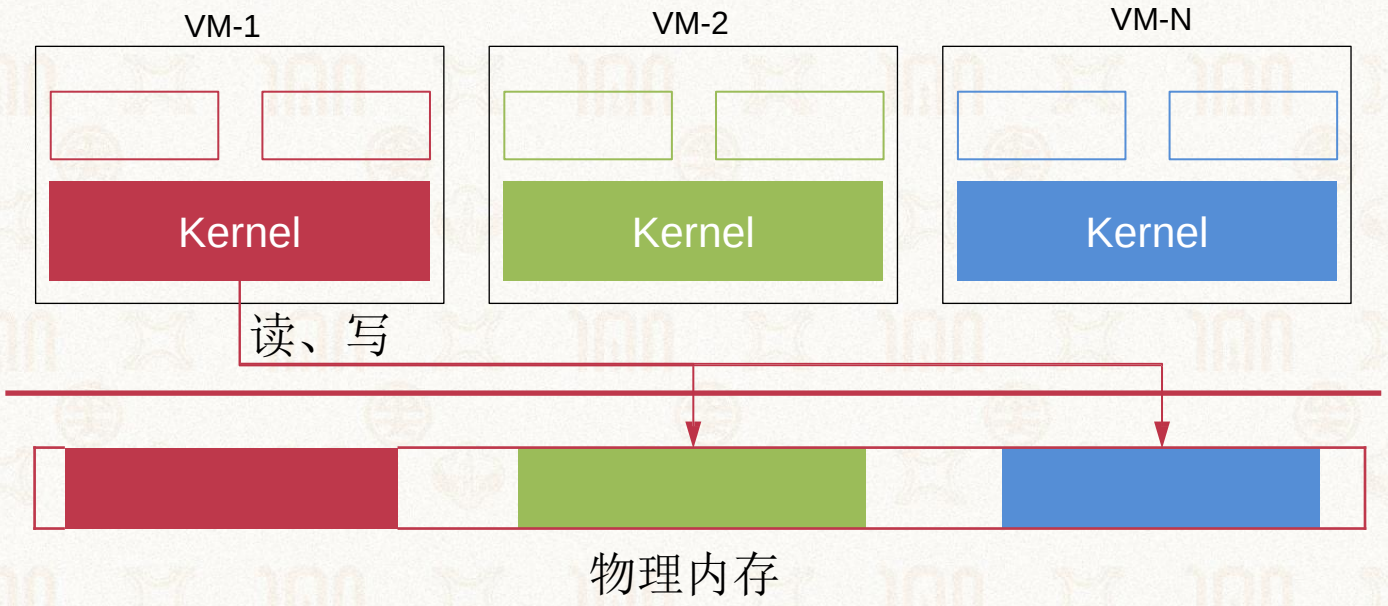
大纲

- 虚拟化概述
  - 为什么要用虚拟化
  - 虚拟化的优势
- 什么是系统虚拟化
  - 虚拟机监控器
  - 虚拟化的类型
- CPU虚拟化
  - 下陷
  - 三种软件虚拟化方法
  - 硬件虚拟化
- 内存虚拟化
  - 影子页表
  - 直接页表
  - 硬件虚拟化
- I/O 虚拟化
  - 设备模拟
  - 半虚拟化
  - 设备直通
- 案例：QEMU/KVM

为什么需要内存虚拟化?

- 操作系统内核直接管理物理内存
  - 物理地址从0开始连续增长
  - 向上层进程提供虚拟内存的抽象

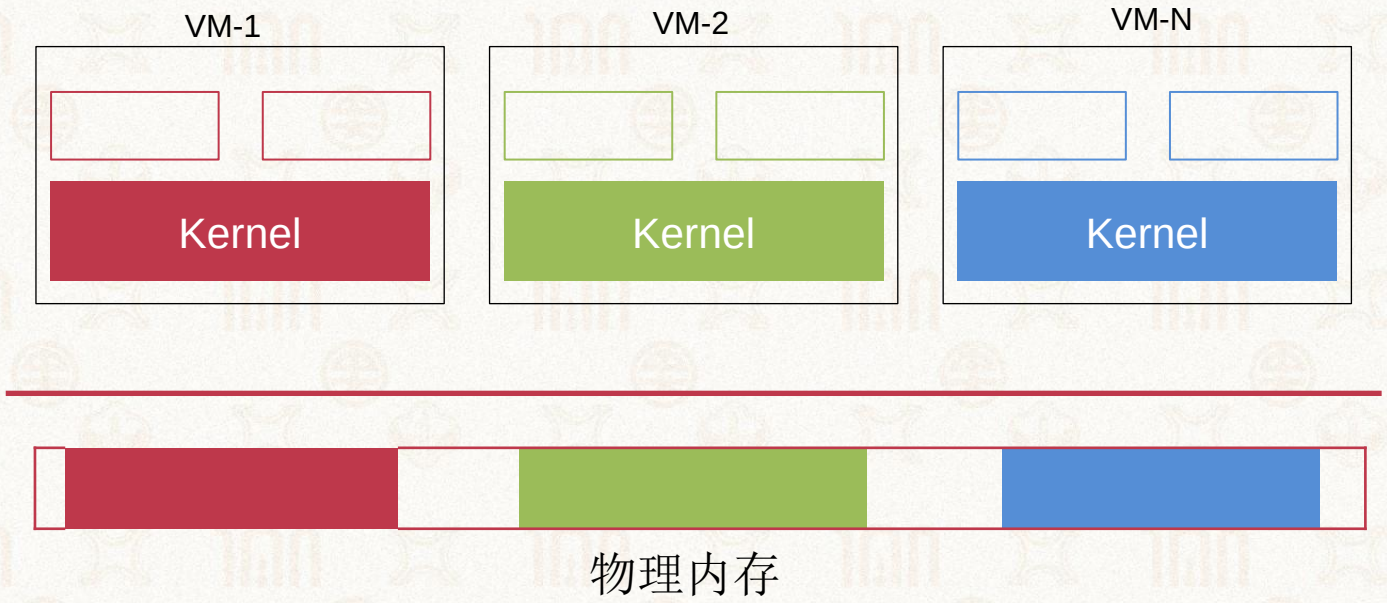
- 如果VM使用的是真实物理地址:



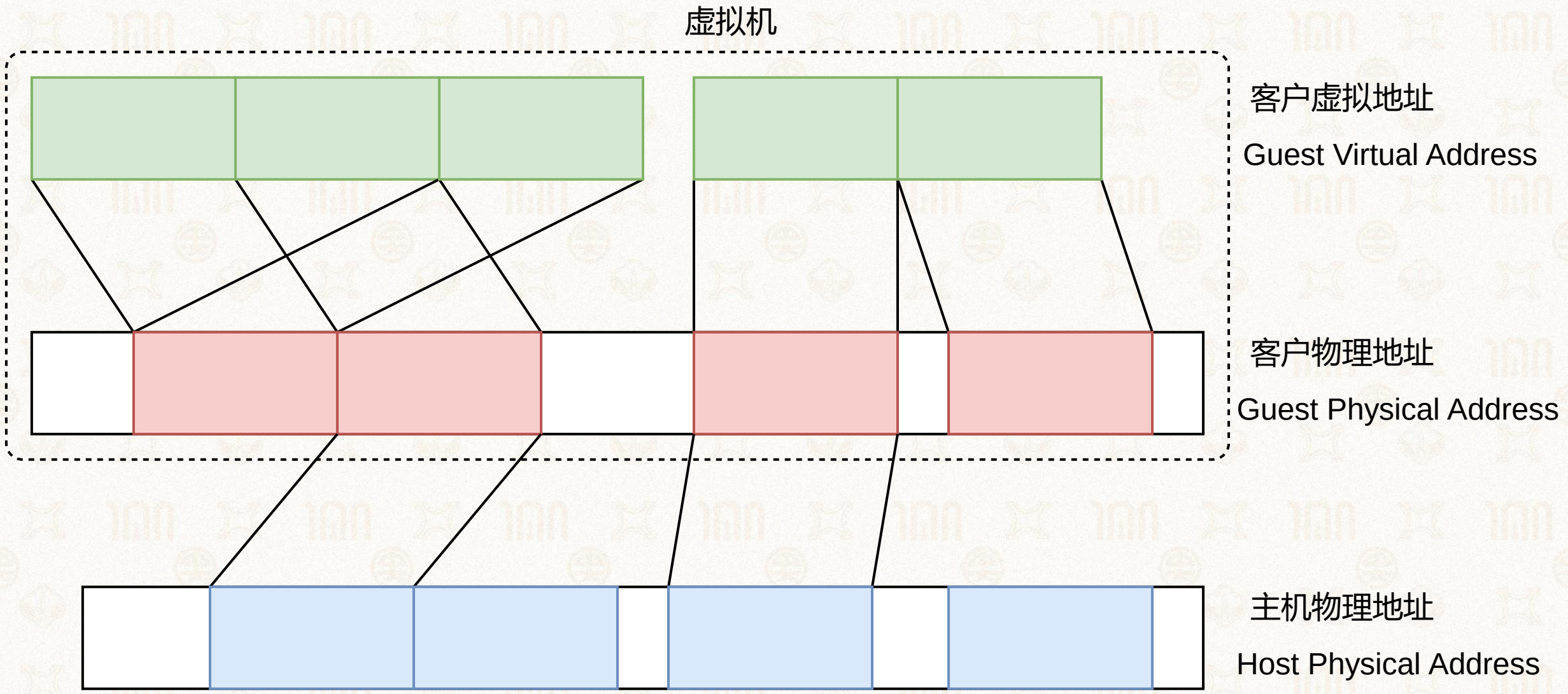
内存虚拟化的目标

- 为虚拟机提供虚拟的物理地址空间
  - 物理地址从0开始连续增长

- 隔离不同虚拟机的物理地址空间
  - VM-1无法访问其他的内存



三种地址


虚拟机
客户虚拟地址
Guest Virtual Address
客户物理地址
Guest Physical Address
主机物理地址
Host Physical Address

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

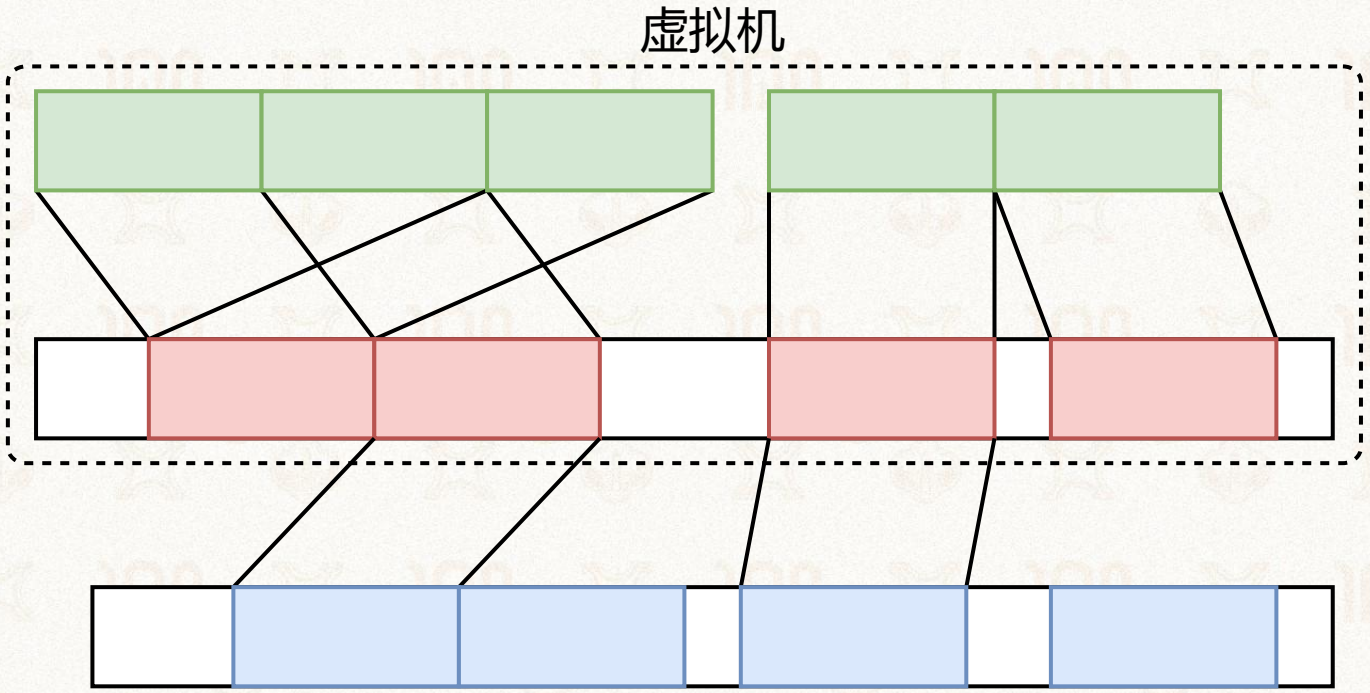
VMM管理

大纲

- 虚拟化概述
  - 为什么要用虚拟化
  - 虚拟化的优势
- 什么是系统虚拟化
  - 虚拟机监控器
  - 虚拟化的类型
- CPU虚拟化
  - 下陷
  - 三种软件虚拟化方法
  - 硬件虚拟化
- 内存虚拟化
  - 影子页表
  - 直接页表
  - 硬件虚拟化
- I/O 虚拟化
  - 设备模拟
  - 半虚拟化
  - 设备直通
- 案例：QEMU/KVM

怎么实现内存虚拟化?

- 影子页表(Shadow Page Table)
  - “费力地”骗虚拟机内的系统：“你在真实系统里！”
- 直接页表(Direct Page Table)
  - “你在虚拟机里！”，因为“你的物理地址不是从0开始的！”
- 硬件虚拟化



硬件虚拟化对内存翻译的支持

- Intel VT-x和ARM硬件虚拟化都有对应的内存虚拟化
  - Intel Extended Page Table (EPT)
  - ARM Stage-2 Page Table (第二阶段页表)

- 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

虚拟机

虚拟机监控器

客户虚拟地址

第一阶段页表

客户物理地址

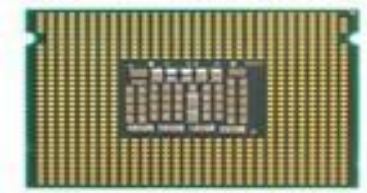
第二阶段页表

主机物理地址

虚拟地址

第一阶段页表

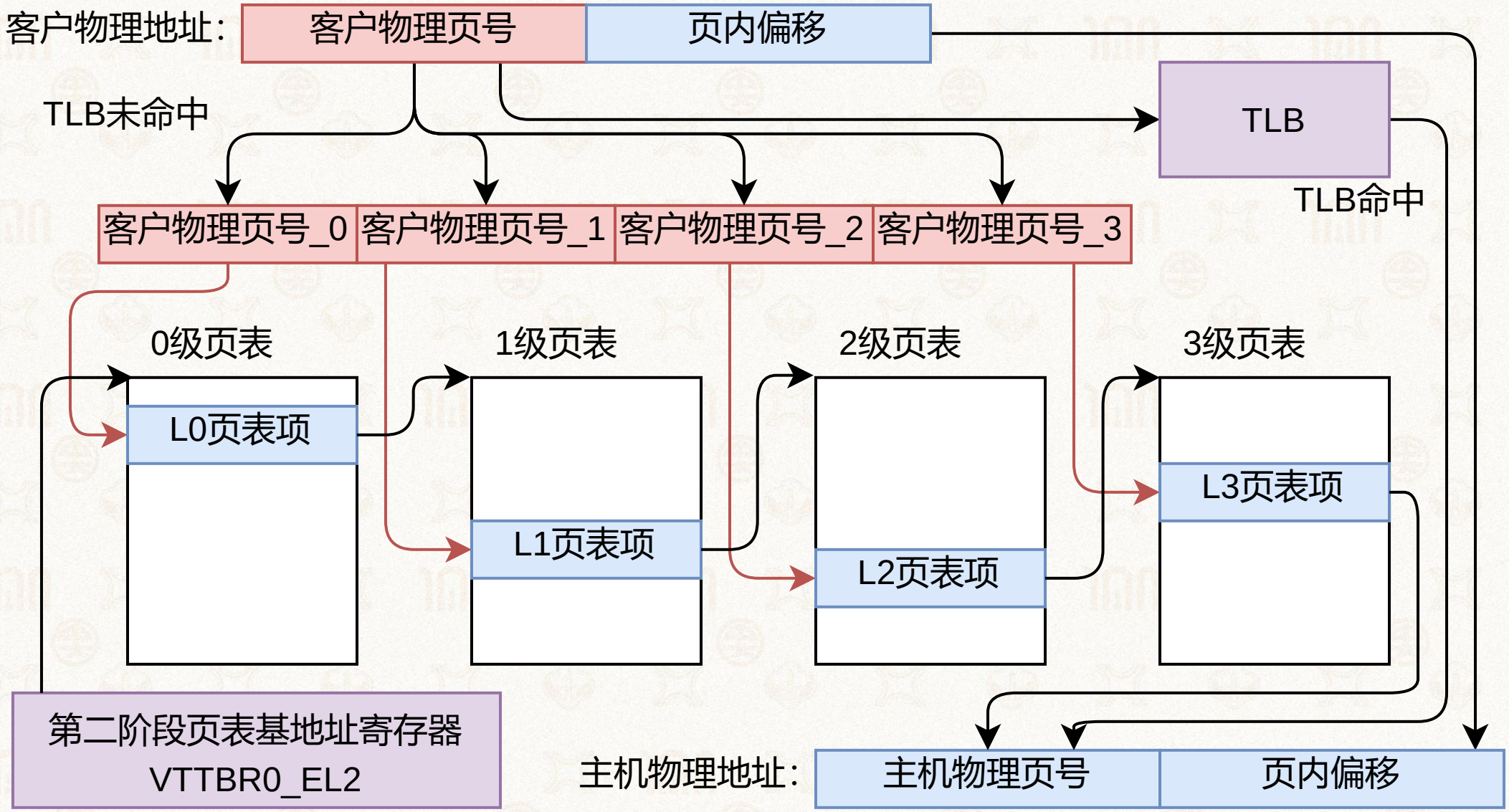
物理地址



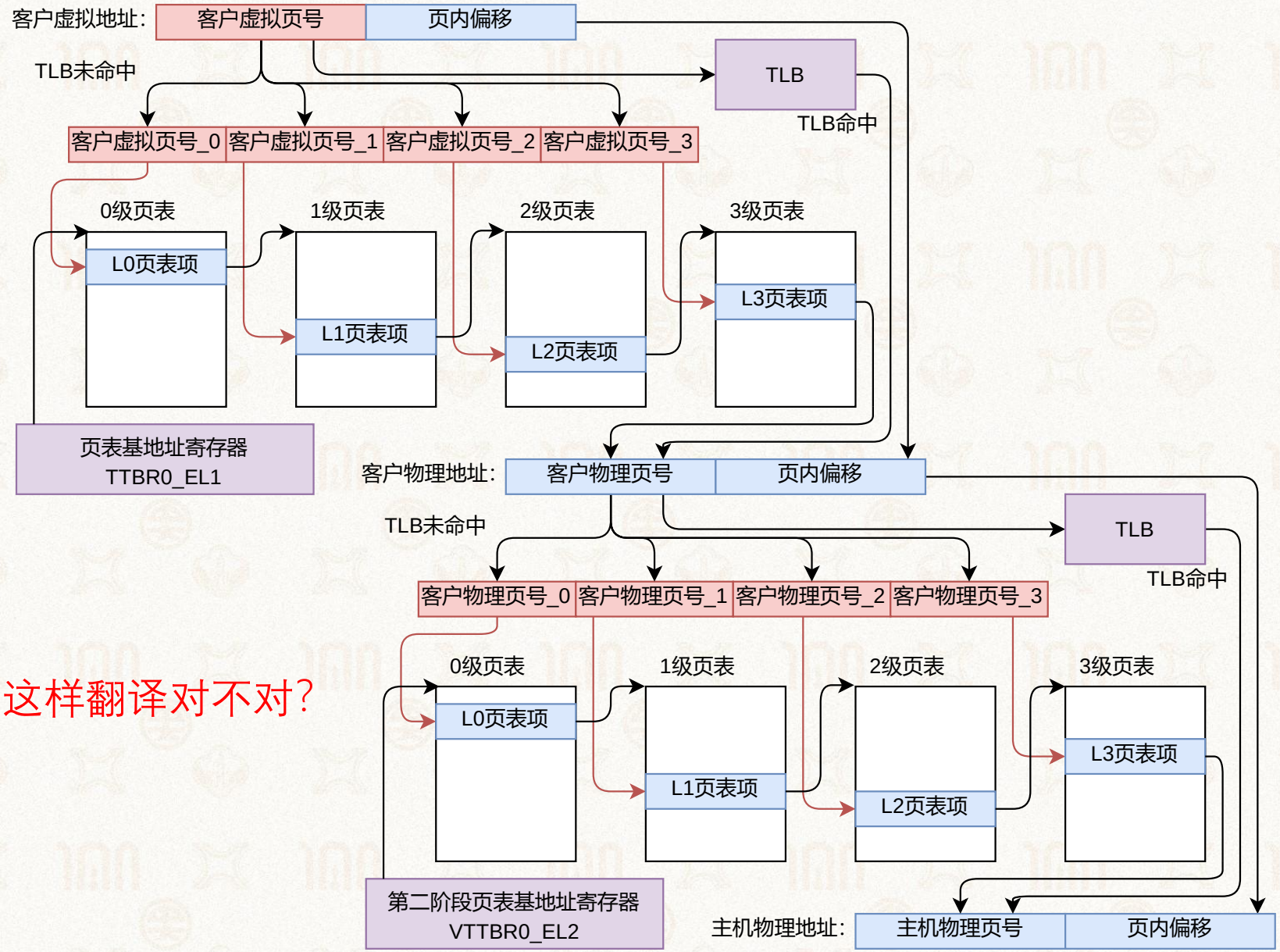
EL0 & EL1

EL2

第二阶段四级页表


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
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
第二阶段页表基地址寄存器
VTTBR0_EL2
主机物理地址：
主机物理页号
页内偏移

两阶段页表的地址翻译过程

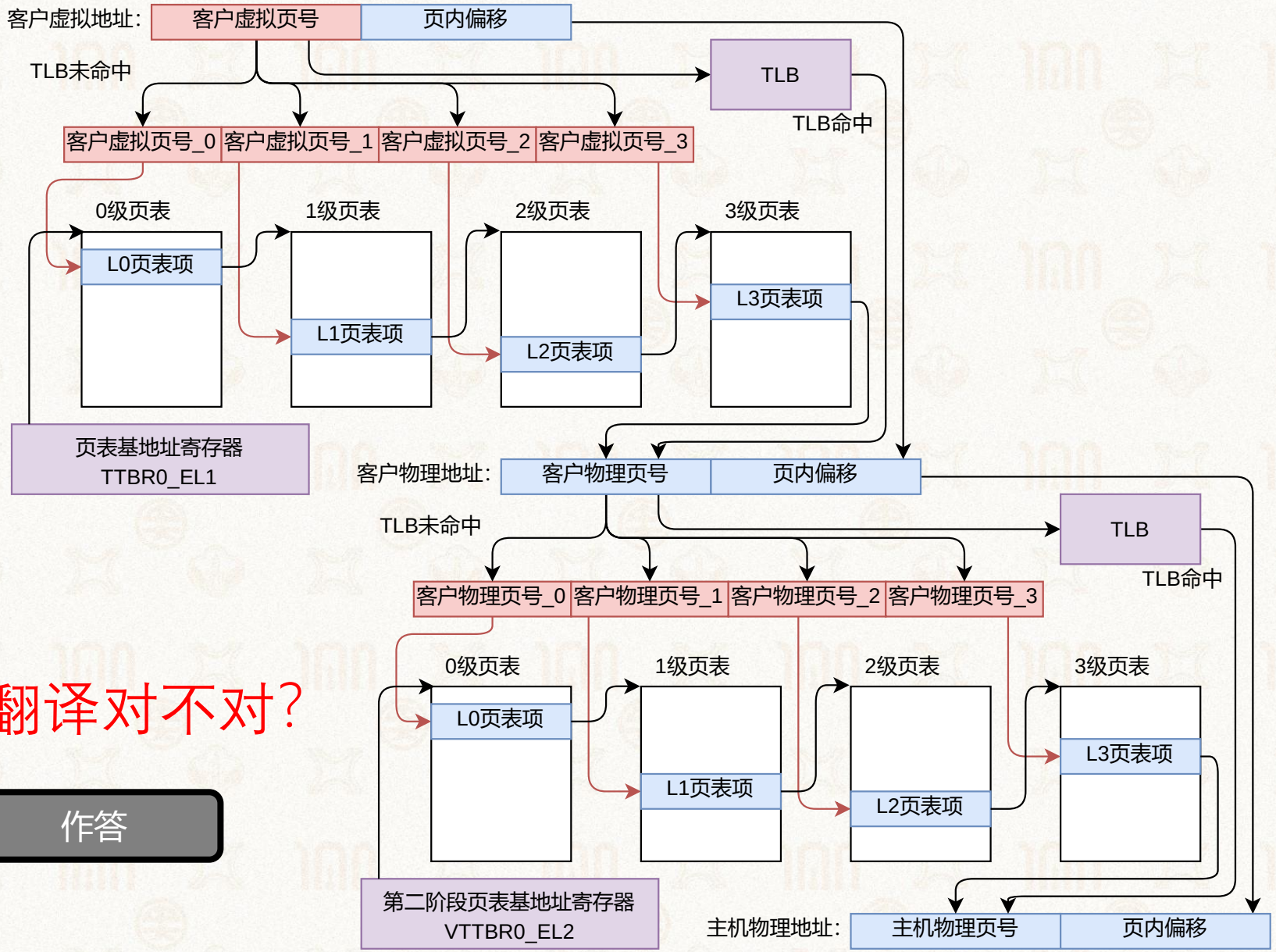


这样翻译对不对?

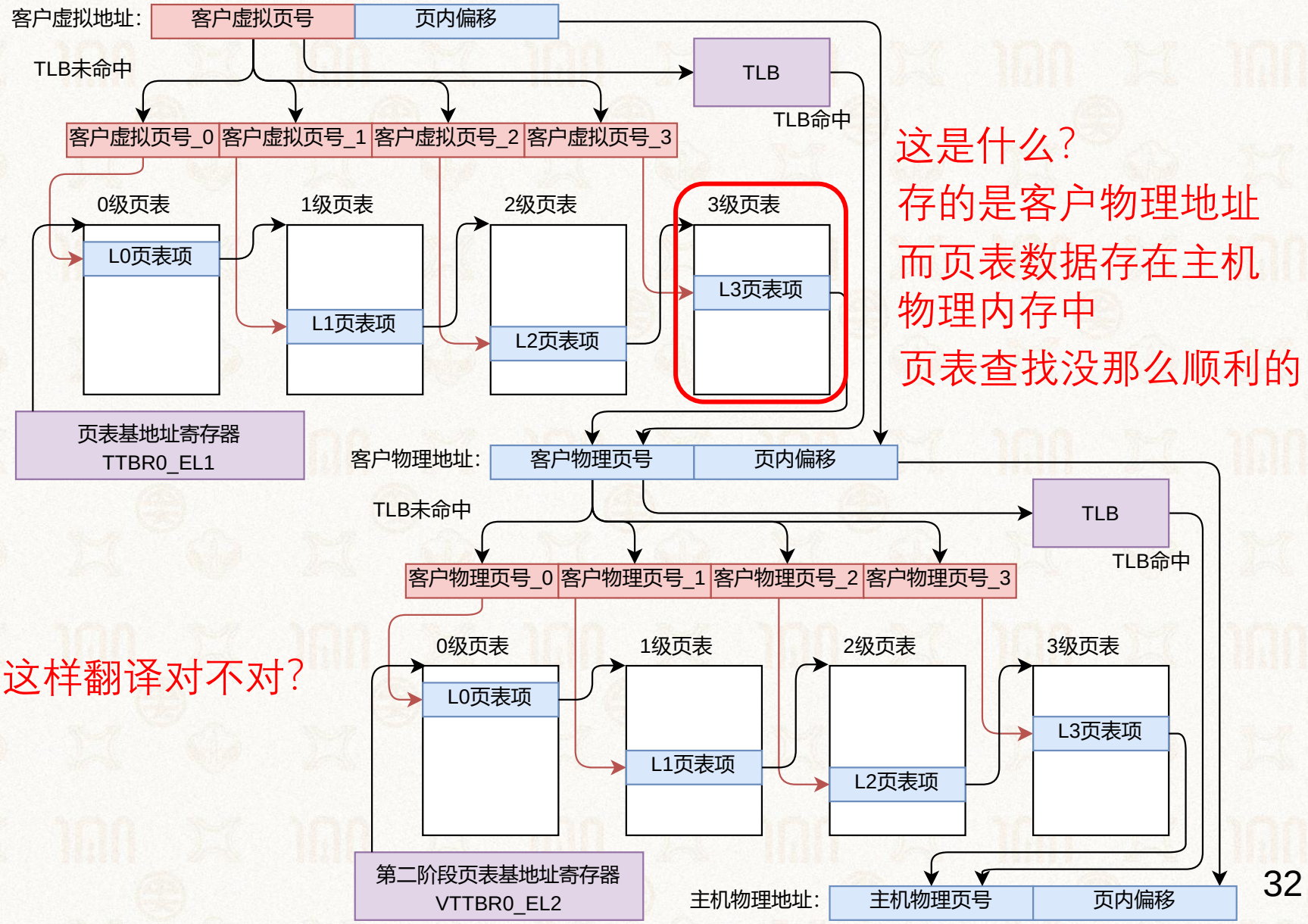
主观题 10分

这样翻译对不对？

作答



两阶段页表的地址翻译过程



这是什么？

存的是客户物理地址

而页表数据存在主机物理内存中

页表查找没那么顺利的

这样翻译对不对？

两阶段页表的地址翻译过程

客户虚拟地址

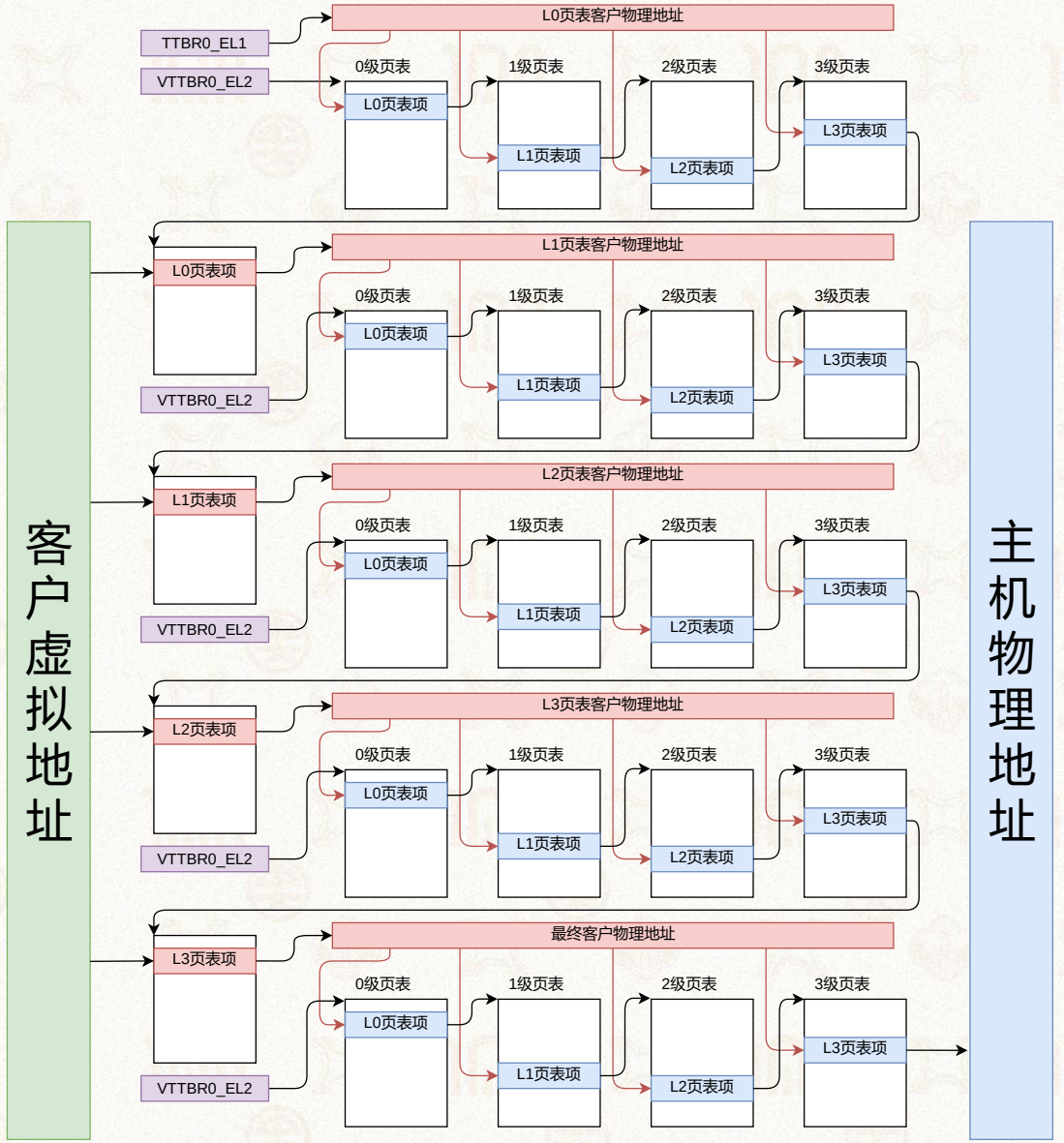
第一阶段页表

客户物理地址

第二阶段页表

主机物理地址

- 最差情况下，一次翻译总共需要访问主机物理内存24次
- TLB显得非常重要



https://gitee.com/yxsu/sse202-figures/blob/master/virt/two-stages-page-table.drawio

两阶段页表的地址翻译过程

客户虚拟地址

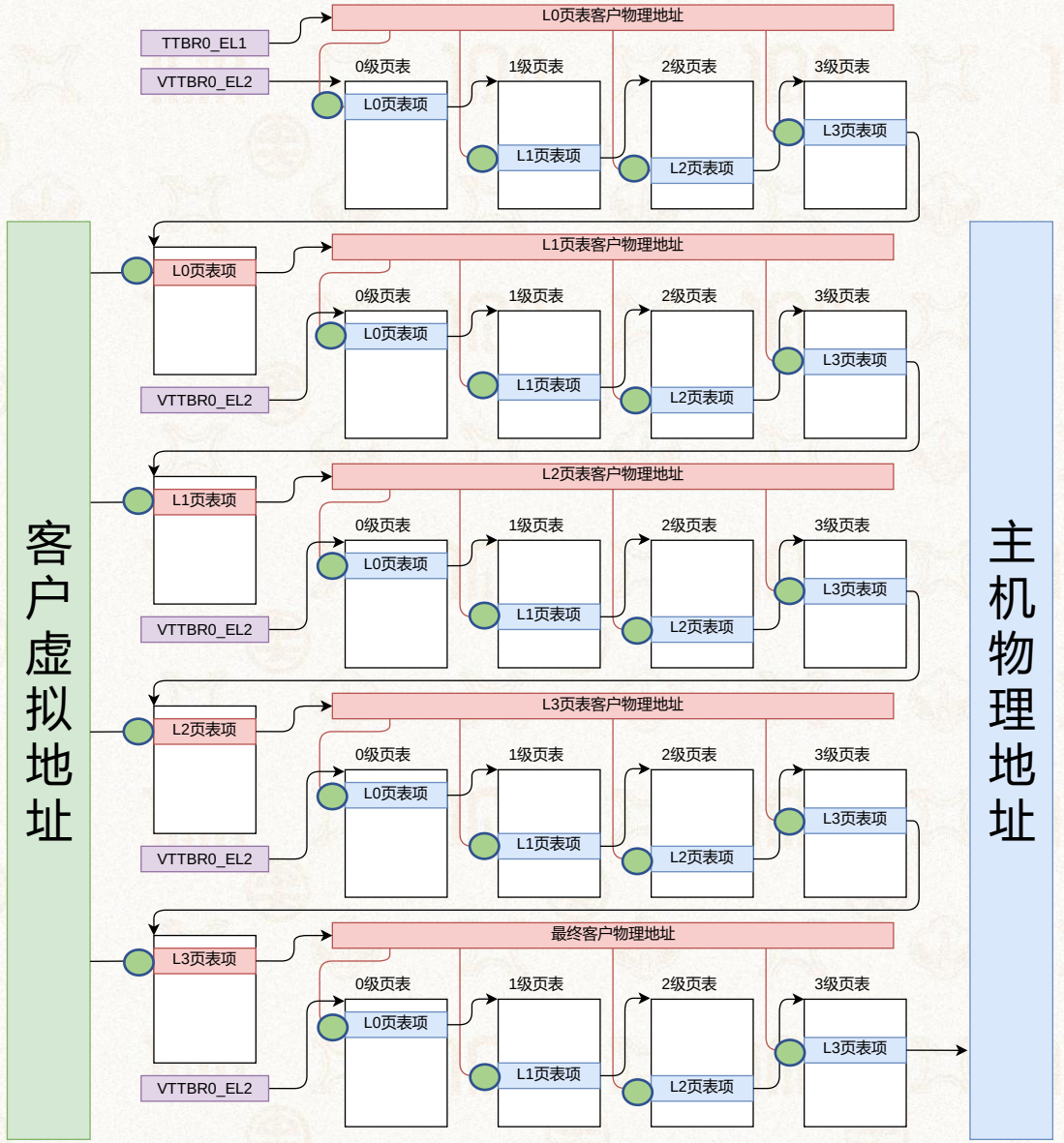
第一阶段页表

客户物理地址

第二阶段页表

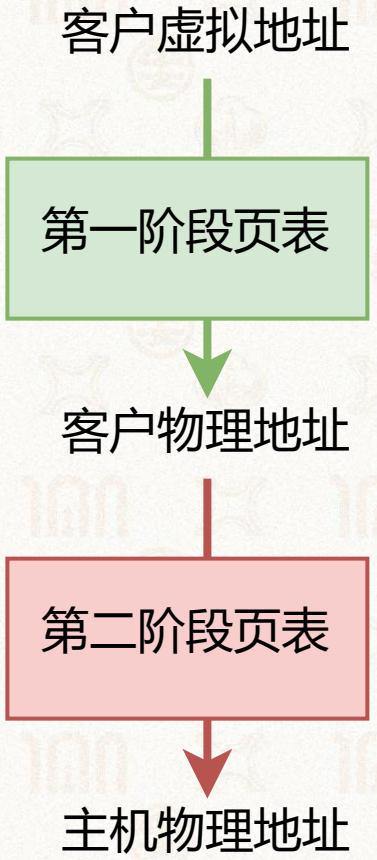
主机物理地址

- 最差情况下，一次翻译总共需要访问主机物理内存24次(5)
- TLB显得非常重要



https://gitee.com/yxsu/sse202-figures/blob/master/virt/two-stages-page-table.drawio

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，直接调用虚拟机监控器的Page fault handler

大纲

- 虚拟化概述
  - 为什么要用虚拟化
  - 虚拟化的优势
- 什么是系统虚拟化
  - 虚拟机监控器
  - 虚拟化的类型
- CPU虚拟化
  - 下陷
  - 三种软件虚拟化方法
  - 硬件虚拟化
- 内存虚拟化
  - 影子页表
  - 直接页表
  - 硬件虚拟化
- I/O 虚拟化
  - 设备模拟
  - 半虚拟化
  - 设备直通
- 案例：QEMU/KVM

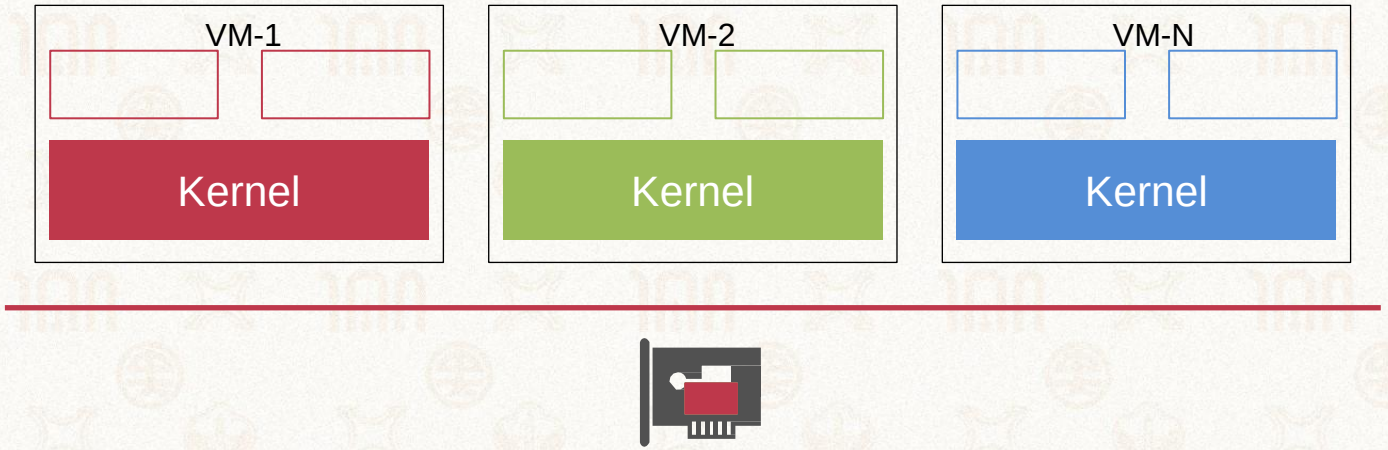
为什么需要IO虚拟化

- 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

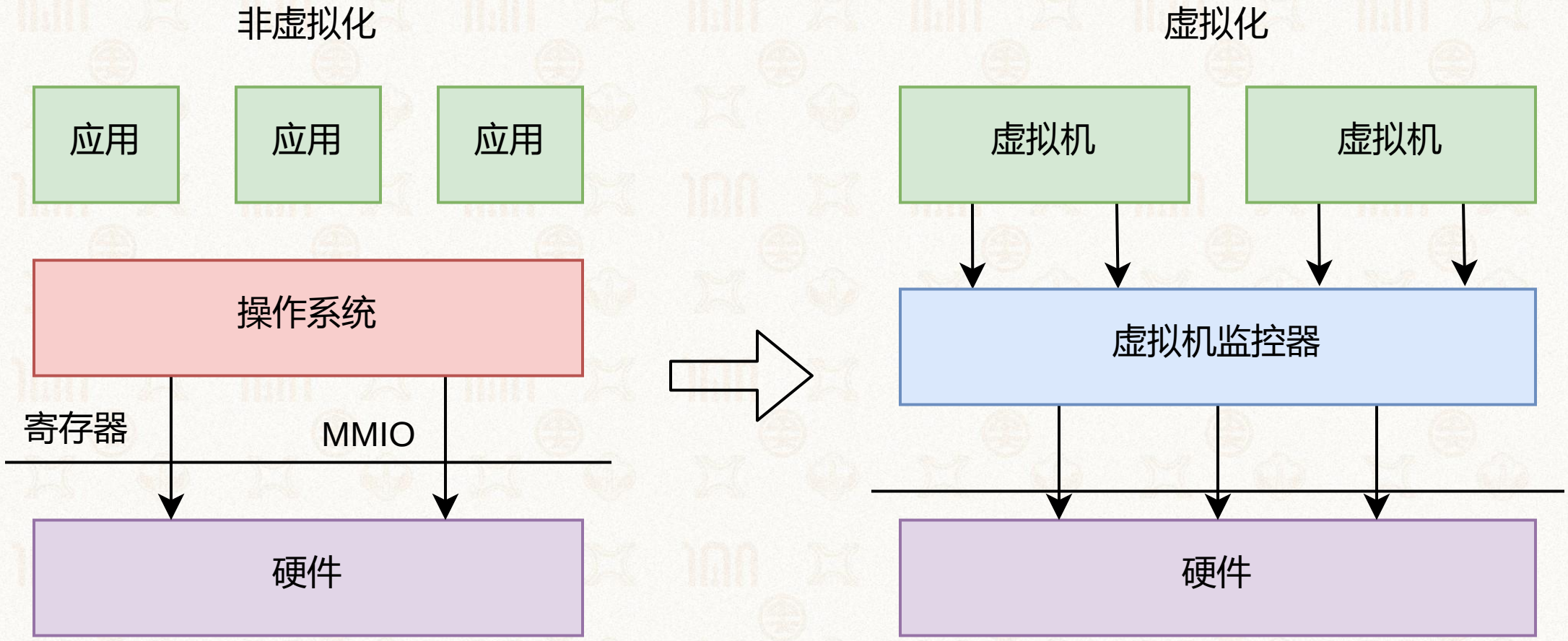
- 为虚拟机提供虚拟的外部设备
  - 虚拟机正常使用设备
- 隔离不同虚拟机对外部设备的直接访问
  - 实现I/O数据流和控制流的隔离
- 提高物理设备的利用资源
  - 多个VM同时使用，可以提高物理设备的资源利用率

怎么实现I/O虚拟化?

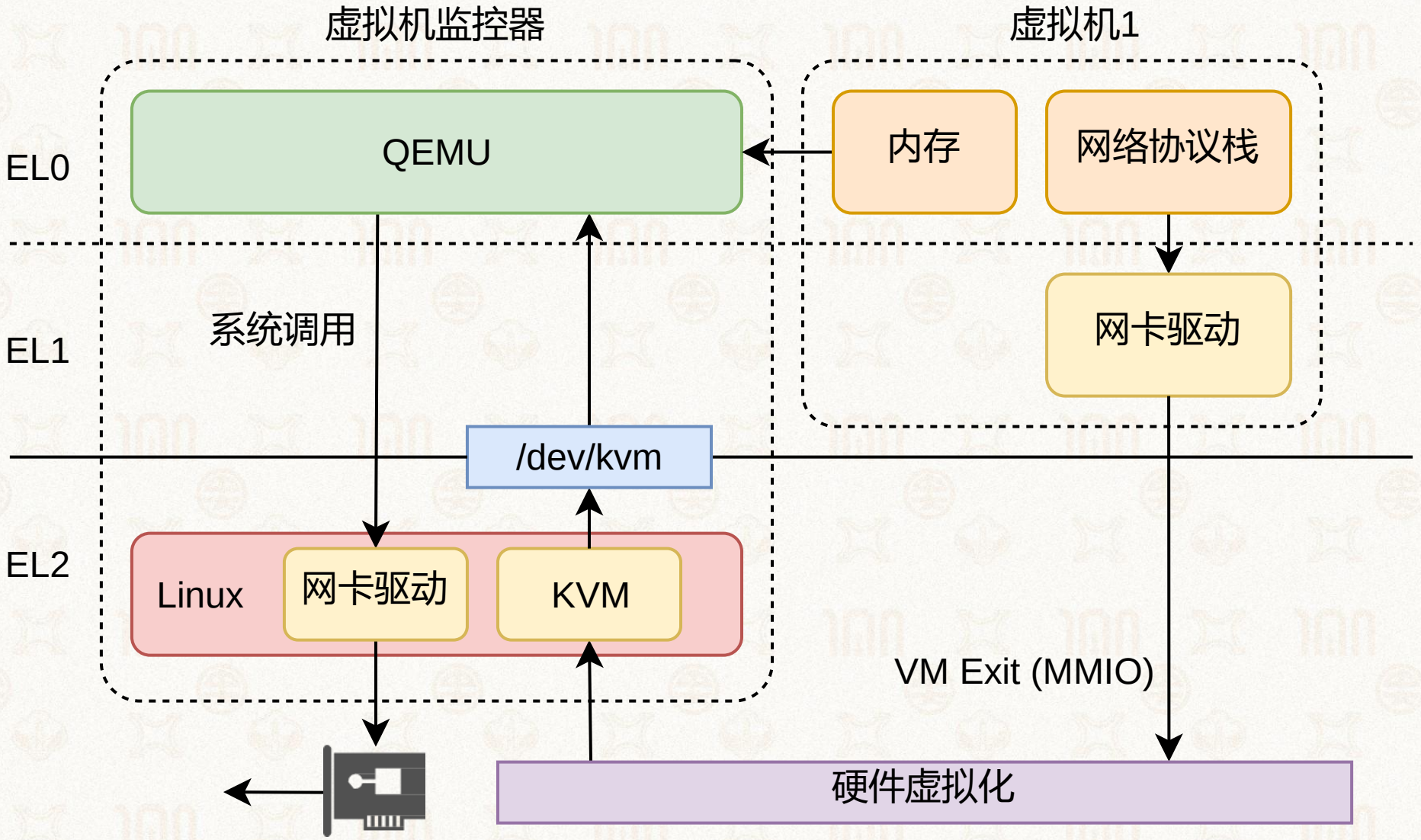
- 设备模拟（Emulation）
- 半虚拟化方式（Para-virtualization）
- 设备直通（Pass-through）

方法1：设备模拟

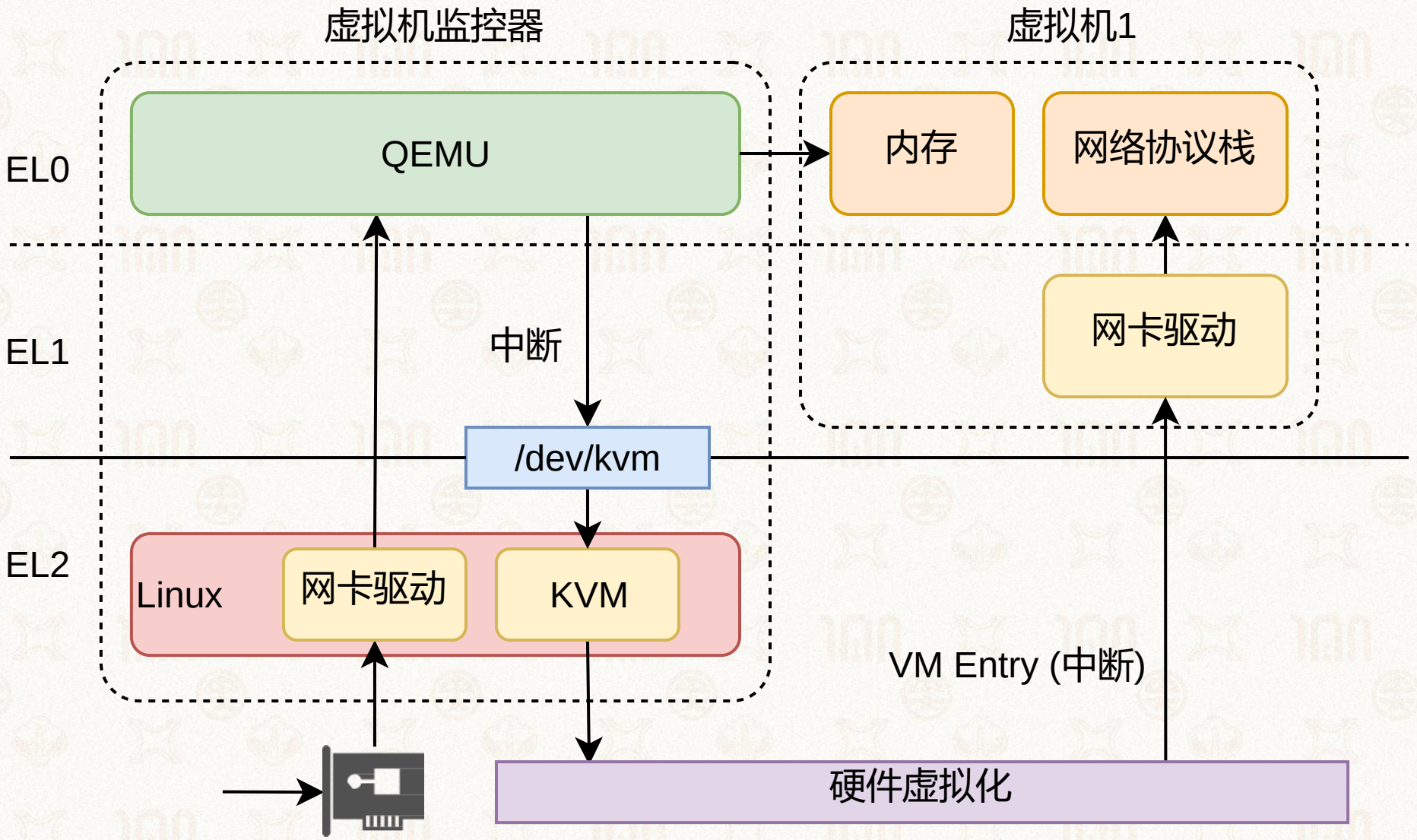
- OS与设备交互的硬件接口
  - 模拟寄存器(中断等)
  - 捕捉MMIO操作
- 硬件虚拟化的方式
  - 硬件虚拟化捕捉PIO指令
  - MMIO对应内存在第二阶段页表中设置为invalid



例：以虚拟网卡举例——发包过程



例：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

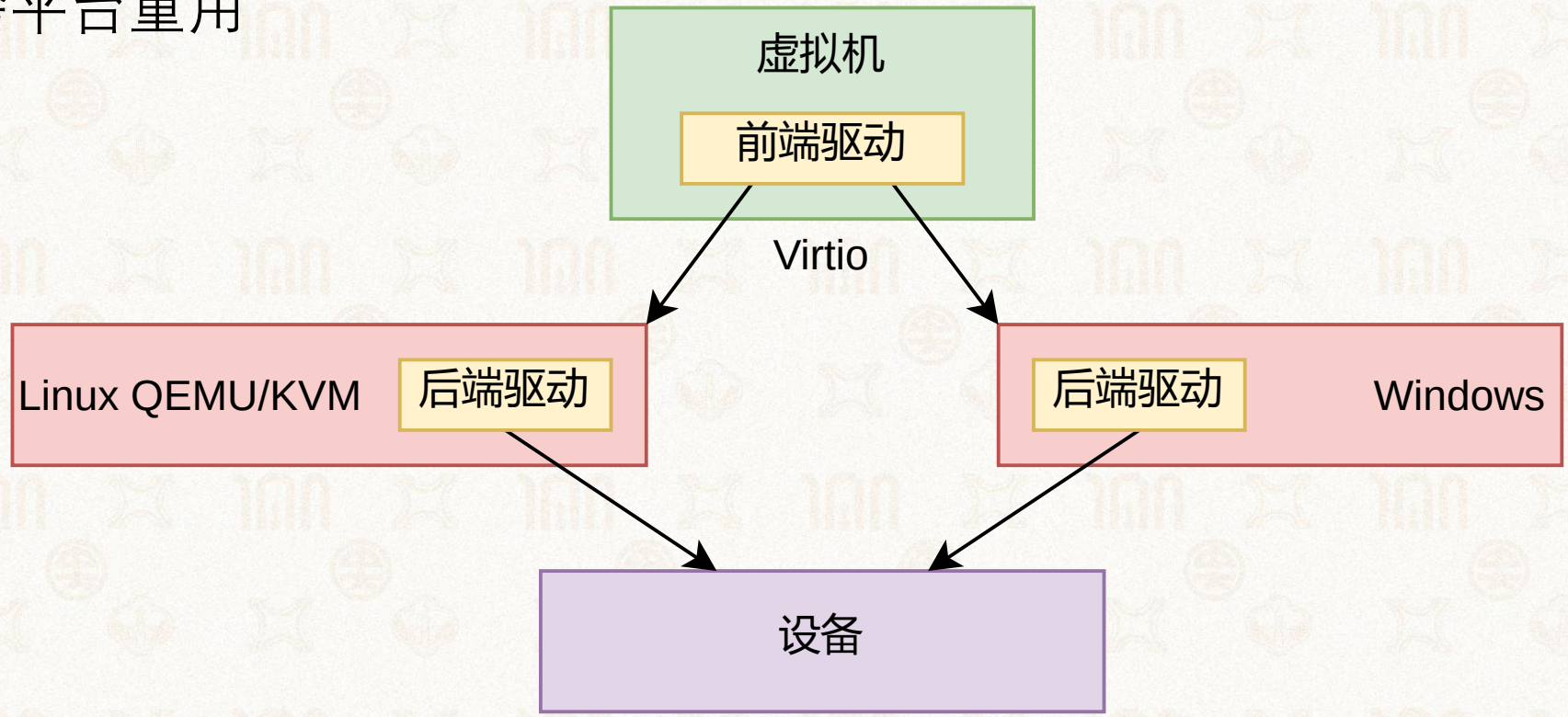
- 协同设计
  - 虚拟机“知道”自己运行在虚拟化环境
  - 虚拟机内运行前端(front-end)驱动
  - 虚拟机监控器内运行后端(back-end)驱动

- 虚拟机监控器主动提供超级调用给虚拟机

- 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

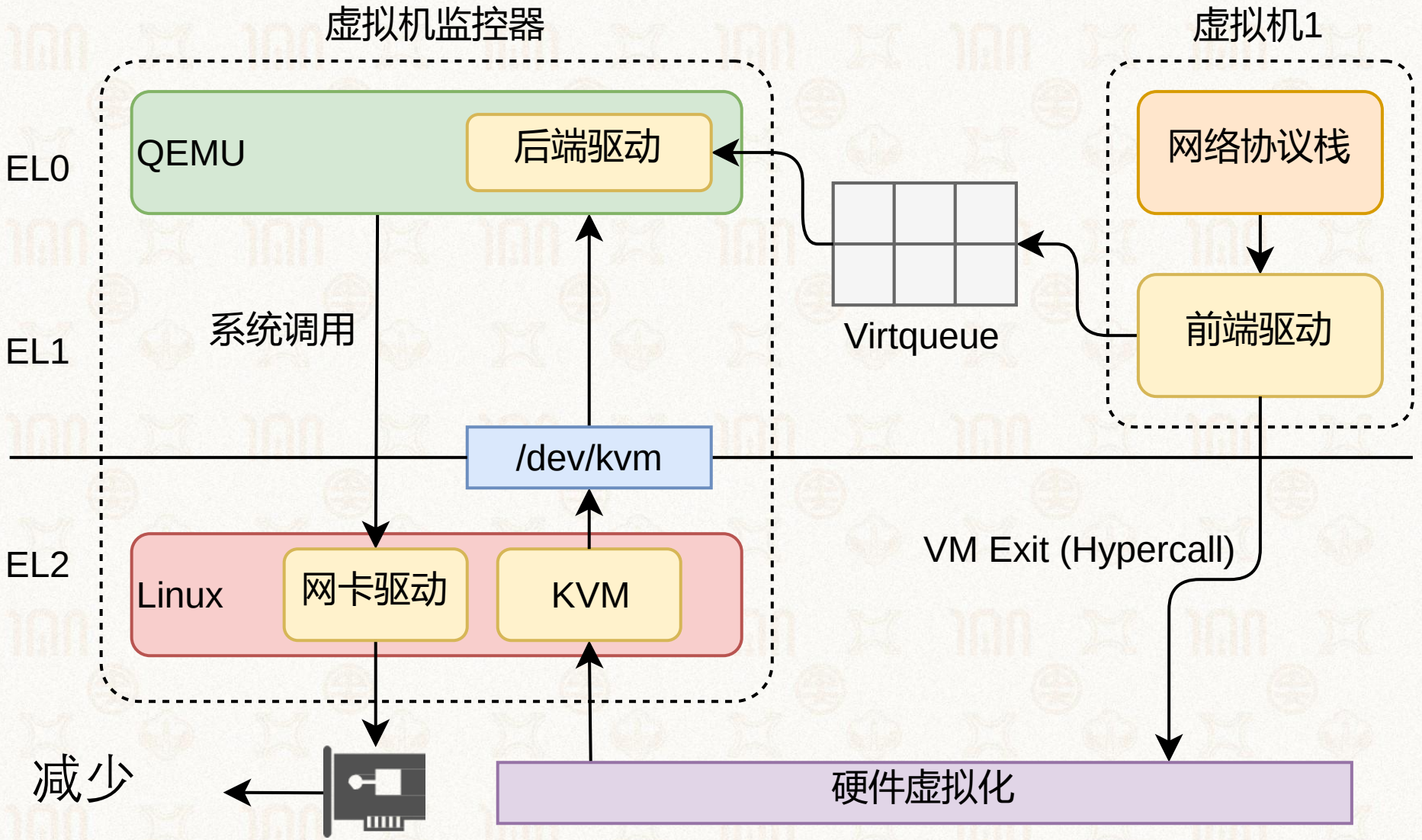


Virtqueue

- 前后端之间通过共享内存创建传递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



- 可以批处理，减少I/O次数

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
  - 虚拟机监控器实现简单，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

大纲

- 虚拟化概述
  - 为什么要用虚拟化
  - 虚拟化的优势
- 什么是系统虚拟化
  - 虚拟机监控器
  - 虚拟化的类型
- CPU虚拟化
  - 下陷
  - 三种软件虚拟化方法
  - 硬件虚拟化
- 内存虚拟化
  - 影子页表
  - 直接页表
  - 硬件虚拟化
- I/O 虚拟化
  - 设备模拟
  - 半虚拟化
  - 设备直通
- 案例：QEMU/KVM

案例：QEMU/KVM

- 2003年，法国程序员Fabrice Bellard发布了QEMU 0.1版本
  - 目标是在非x86机器上使用动态二进制翻译技术模拟x86机器

- 2003-2006年
  - 能模拟出多种不同架构的虚拟机，包括S390、ARM、MIPS、SPARC等
  - 在这阶段，QEMU一直使用软件方法进行模拟
    - 如二进制翻译技术

QEMU/KVM架构

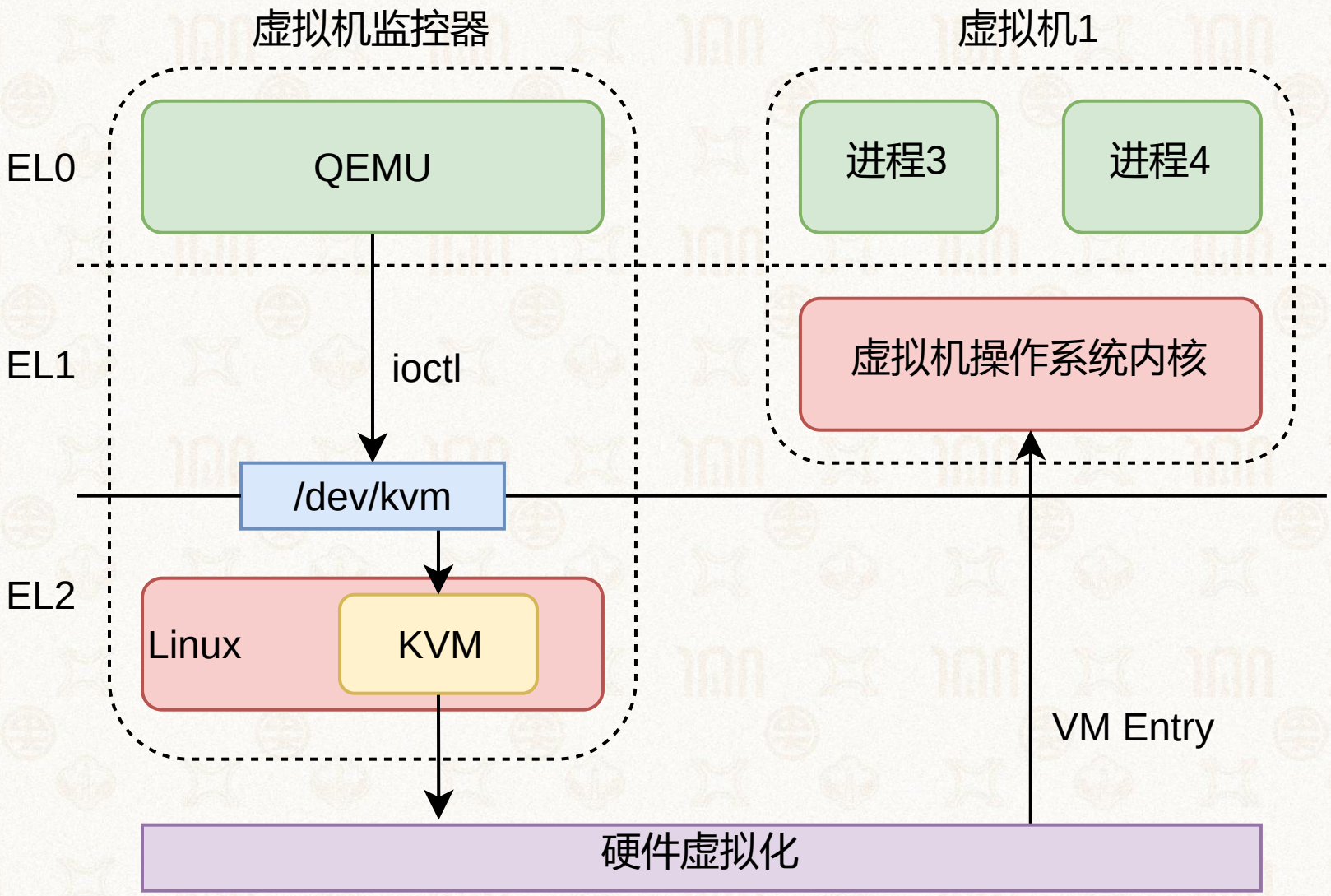
- QEMU运行在用户态，负责实现策略
  - 也提供虚拟设备的支持

- KVM以Linux内核模块运行，负责实现机制
  - 可以直接使用Linux的功能
  - 例如内存管理、进程调度
  - 使用硬件虚拟化功能

- 两部分合作
  - KVM捕捉所有敏感指令和事件，传递给QEMU
  - KVM不提供设备的虚拟化，需要使用QEMU的虚拟设备

QEMU使用KVM的用户态接口

- QEMU使用/dev/kvm与内核态的KVM通信
- 使用ioctl向KVM传递不同的命令：
  - CREATE_VM,
  - CREATE_VCPU,
  - KVM_RUN等



QEMU使用KVM的用户态接口

```
open("/dev/kvm")
ioctl(KVM_CREATE_VM)
ioctl(KVM_CREATE_VCPU)
while (true) {
    ioctl(KVM_RUN)
    exit_reason = get_exit_reason();
    switch (exit_reason) {
        case KVM_EXIT_IO:   /* ... */
            break;
        case KVM_EXIT_MMIO: /* ... */
            break;
    }
}
```

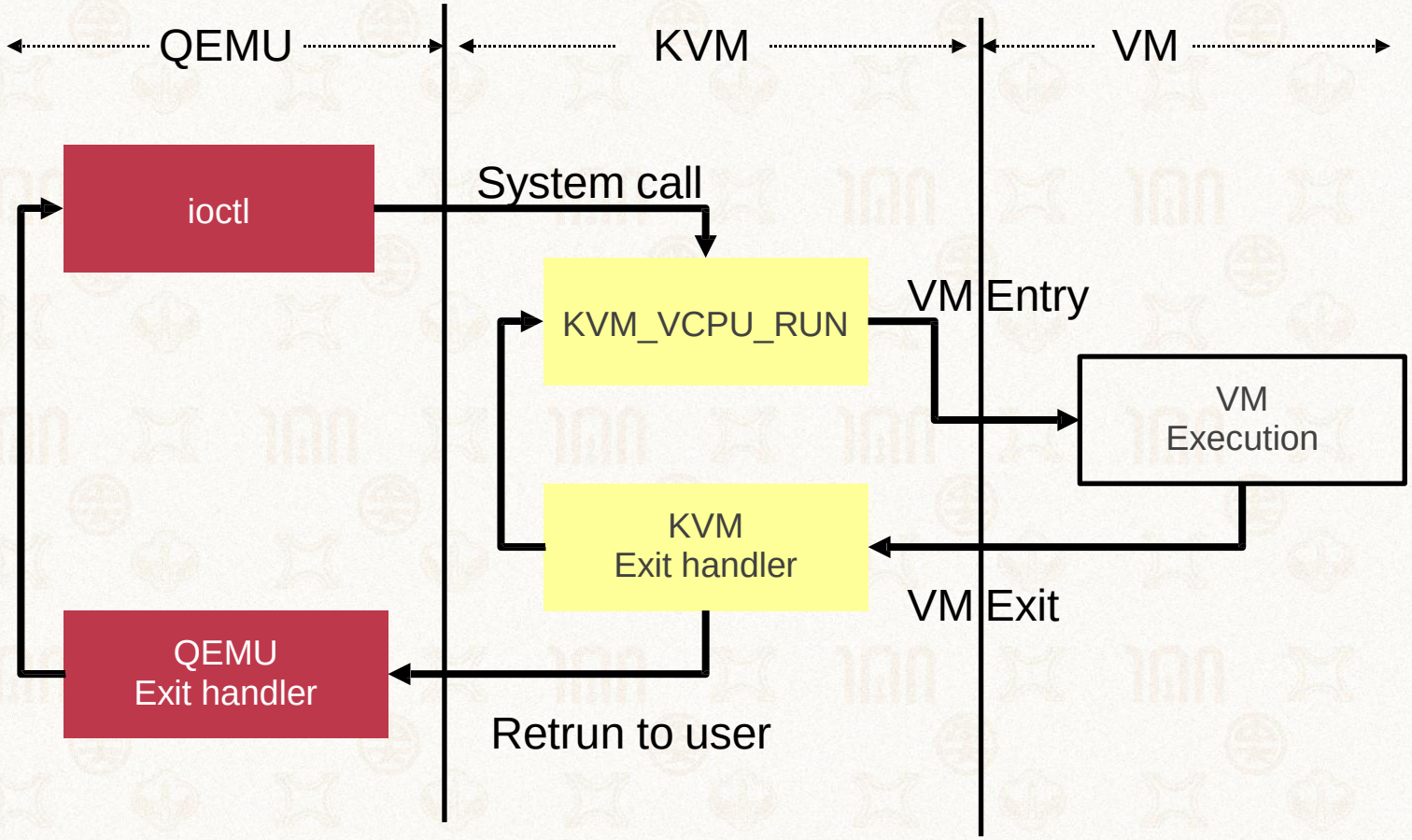
准备调度某个虚拟机实体

ioctl(KVM_RUN)时发生了什么

- x86中
  - KVM找到此VCPU对应的VMCS
  - 使用指令加载VMCS
  - VMLAUNCH/VMRESUME进入Non-root模式
    - 硬件自动同步状态
    - PC切换成VMCS->GUEST_RIP，开始执行

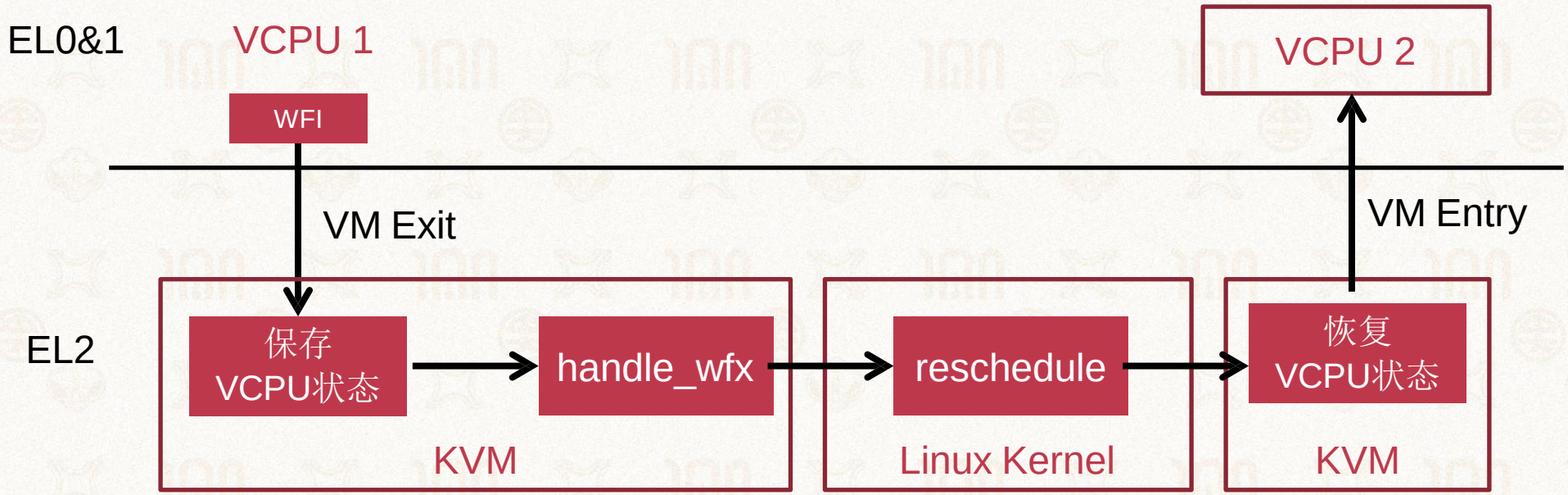
- ARM中
  - KVM主动加载VCPU对应的所有状态
  - 使用eret指令进入EL1
    - PC切换成ELR_EL2的值，开始执行

QEMU/KVM的流程


QEMU
KVM
VM
ioctl
System call
KVM_VCPU_RUN
VM Entry
VM
Execution
KVM
Exit handler
VM Exit
QEMU
Exit handler
Retrun to user

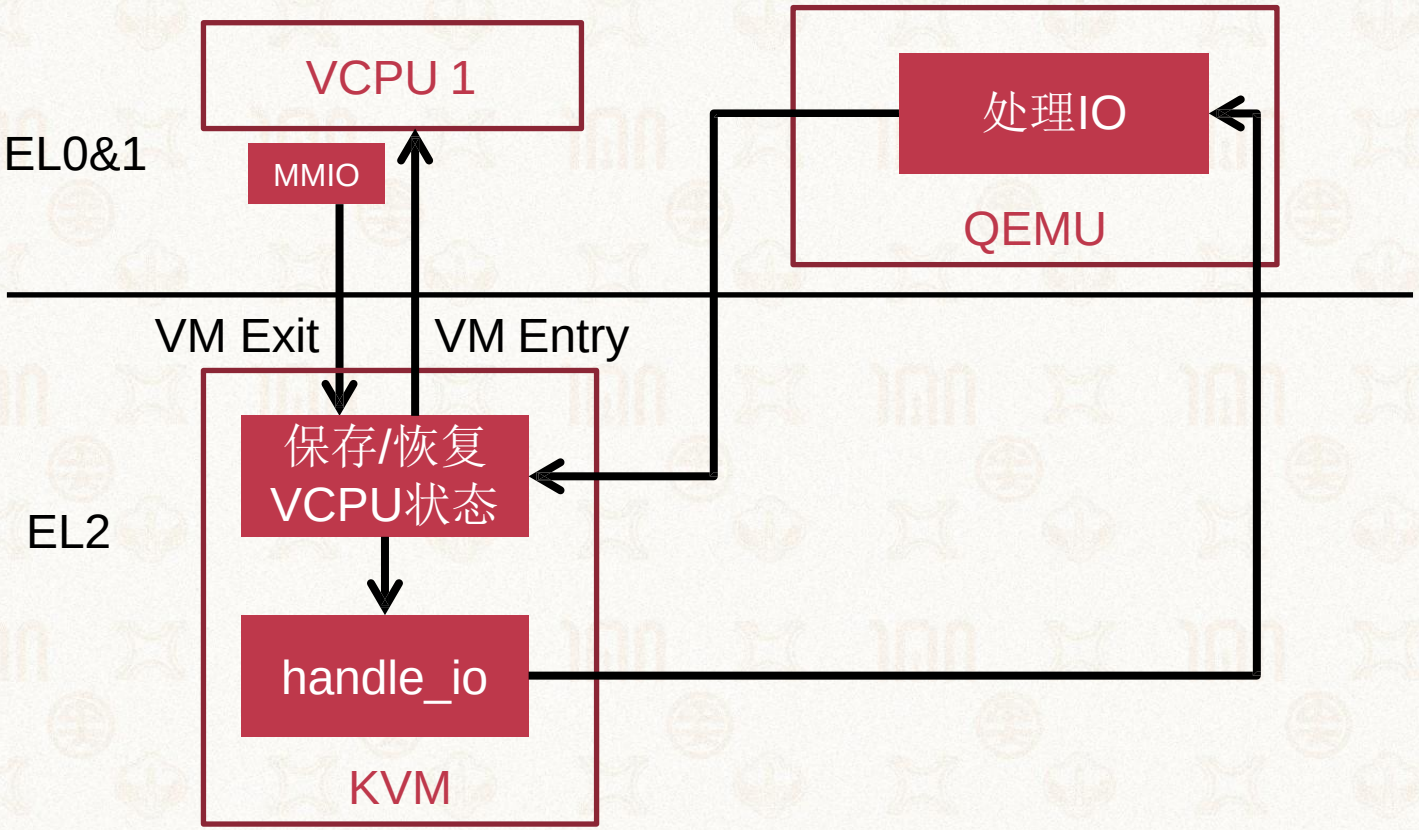
例：WFI指令VM Exit的处理流程

- VCPU1发出休眠指令，虚拟机监控器不再调度VCPU1



例：I/O指令VM Exit的处理流程

- QEMU中有完整的软件模拟设备



大纲

- 虚拟化概述
  - 为什么要用虚拟化
  - 虚拟化的优势
- 什么是系统虚拟化
  - 虚拟机监控器
  - 虚拟化的类型
- CPU虚拟化
  - 下陷
  - 三种软件虚拟化方法
  - 硬件虚拟化
- 内存虚拟化
  - 影子页表
  - 直接页表
  - 硬件虚拟化
- I/O 虚拟化
  - 设备模拟
  - 半虚拟化
  - 设备直通
- 案例：QEMU/KVM

谢谢

微信：suyuxin
钉钉：苏玉鑫
B站：https://space.bilibili.com/502854403
软工集市课程专区：https://ssemarket.cn/new/course
匿名提问箱：https://suask.me/ask-teacher/106/苏玉鑫