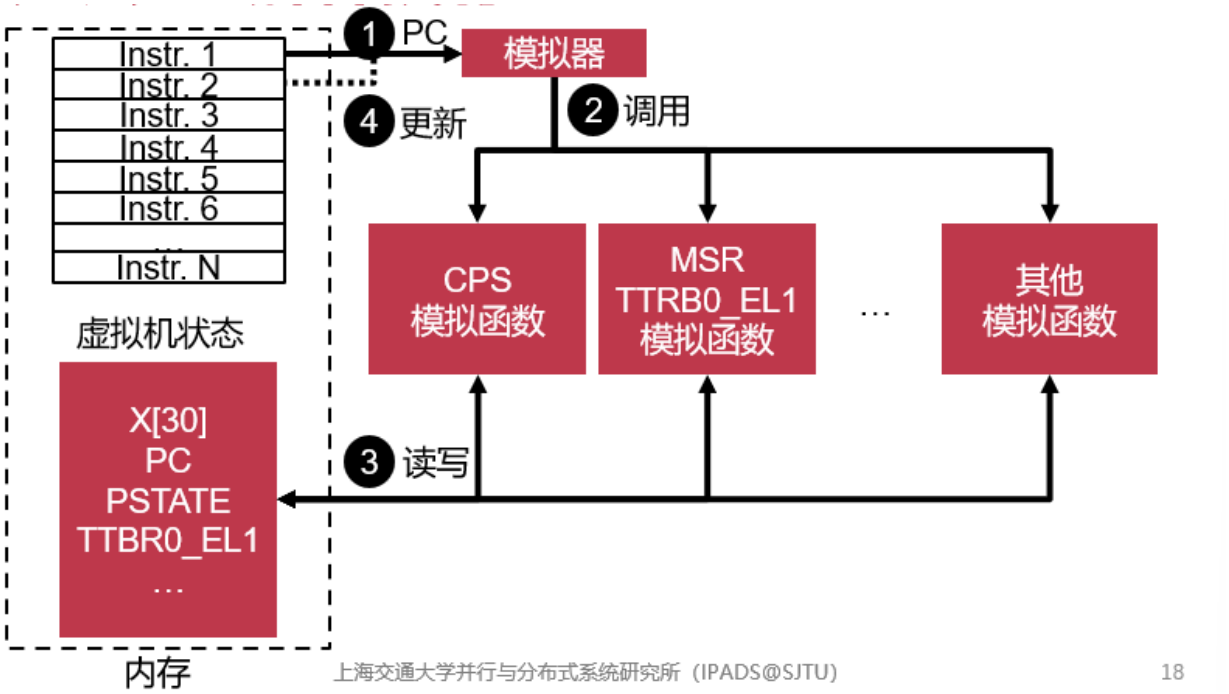
Lecutre20 CPU Virtualization

1. 如何处理不会下陷的敏感指令

> - 方法1：解释执行
> - 方法2：二进制翻译
> - 方法3：半虚拟化
> - 方法4：硬件虚拟化（改硬件）

方法1：解释执行

- 使用软件方法一条条对虚拟机代码进行模拟
  - 不区分敏感指令还是其它指令
  - 没有虚拟机指令直接在硬件上执行
- 使用内存维护虚拟机状态
  - 例如：使用uint64_t x[30]数组保存所有通用寄存器的值



- **优缺点**
  - 优点
    1. 解决了敏感函数不下陷的问题
    2. 可以模拟不同ISA的虚拟机
    3. 易于实现、复杂度低
  - 缺点
    - 非常慢：任何一条虚拟机指令都会被转换成多条模拟指令

方法2：二进制翻译

- 提出两个加速技术
  - 在执行前**批量翻译**虚拟机指令
  - **缓存**已翻译完成的指令
- 使用基本块(Basic Block)的翻译粒度
  - 每一个基本块被翻译后叫代码补丁

> Q：为什么以Basic Block为翻译粒度?
>
> A：不包含if-else分支，执行速度快

- 如何翻译?

  把一些敏感的指令换成另外的函数（HADLE_XXX）
- **二进制翻译的缺点**
  1. 不能处理自修改的代码(Self-modifying Code)
  2. 中断插入的粒度变大
     - 模拟执行可以在任意指令位置插入虚拟中断
     - **二进制翻译时只能在Basic Block边界插入虚拟中断**（？？？）

       > 因为敏感指令都被换成了不敏感的指令

方法3：半虚拟化(Para-virtualization)

- **协同设计**
  - 让VMM提供接口给虚拟机，称为**Hypercall**
  - 修改操作系统源码，让其主动调用VMM接口
  - 还运用了Batch的思想，一次性进行大量修改
- **Hypercall可以理解为VMM提供的系统调用**
  - 在ARM中是HVC指令
- **将所有不引起下陷的敏感指令替换成超级调用**
- 优缺点：
  - 优点：
    - 解决了敏感函数不下陷的问题
    - 协同设计的思想可以提升某些场景下的系统性能
      - I/O等场景
  - 缺点：
    - 需要修改操作系统代码，难以用于闭源系统
    - 即使是开源系统，也难以同时在不同版本中实现

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
- 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

2. Intel VT-x

VT-x的处理器虚拟化

> Ring-0: Kernel Ring-3: User

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状态保存到VMCS
  - 硬件自动从VMCS中间加载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

VT-x VMCS的内容

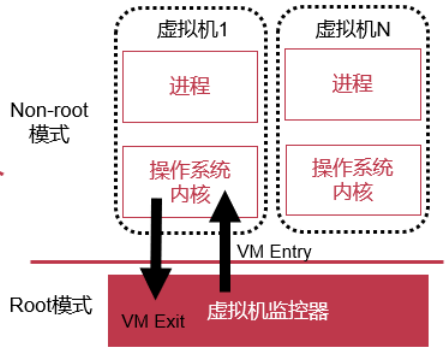
包含6个部分

- 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）

> 8core 10VM 4vCPU -> 需要40个vCPU

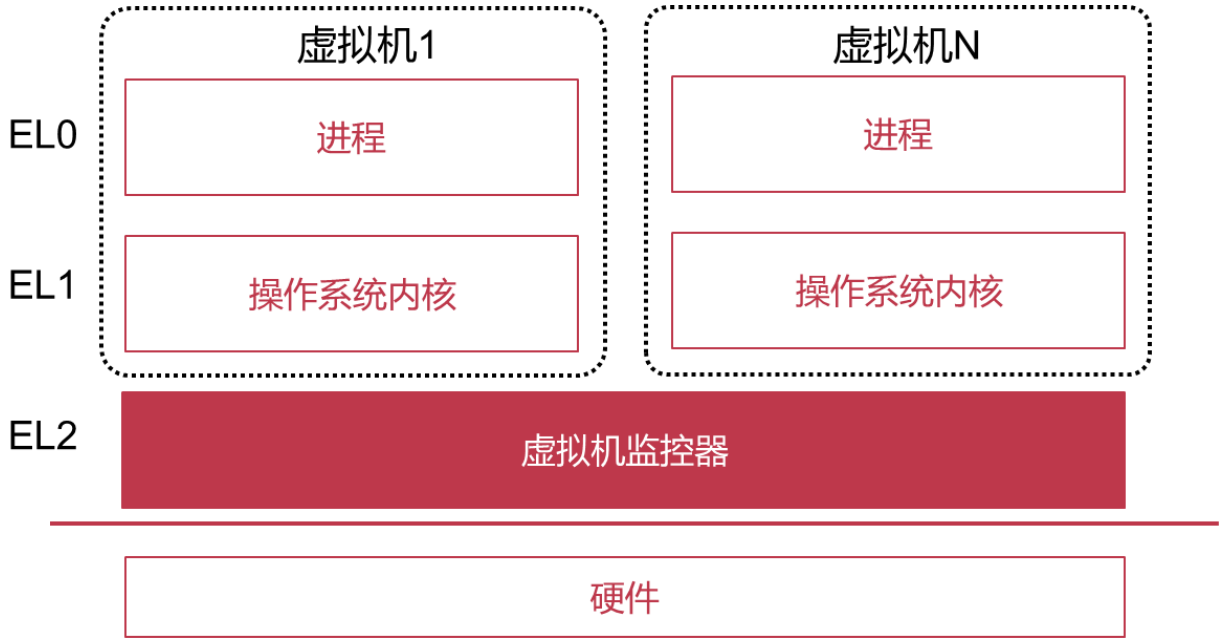
X86中的VM Entry和VM Exit

- **VM Entry**
  - 从VMM进入VM
  - 从Root模式切换到Non-root模式
  - 第一次启动虚拟机时使用VMLAUNCH指令
  - 后续的VM Entry使用VMRESUME指令
- **VM Exit**
  - 从VM回到VMM
  - 从Non-root模式切换到Root模式
  - 虚拟机执行敏感指令或发生事件(如外部中断)



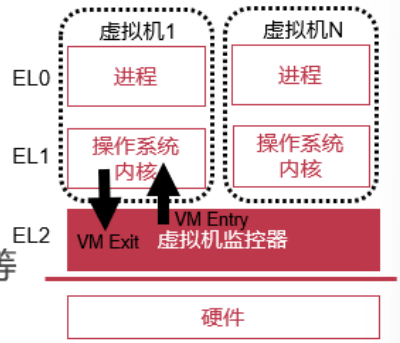
3. ARM的虚拟化技术

ARM处理器的虚拟化



ARM的VM Entry和VM Exit

- **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



ARM硬件虚拟化的新功能

- **ARM中没有VMCS**
- **VM能直接控制EL1和EL0的状态**
  - 自由地修改PSTATE(VMM不需要捕捉CPS指令)
  - 可以读写TTBR0_EL1/SCTRL_EL1/TCR_EL1等寄存器
- **VM Exit时VMM依然可以直接访问VM的EL0和EL1寄存器**

> Q：为什么ARM中可以不需要VMCS?
>
> A：因为ARM为EL1和EL2提供了两套系统寄存器，因此在下陷时VMM无需保存虚拟机在EL1中使用的系统寄存器

> Q：ARM中没有VMCS，对于VMM的设计和实现来说有什么优缺点?

> A：优点是节约了内存空间，省去了维护、读写VMCS的性能开销；缺点是一些x86架构的VMM可能无法在ARM上移植，需要额外的虚拟化扩展如VHE来实现虚拟化，增加了工作量

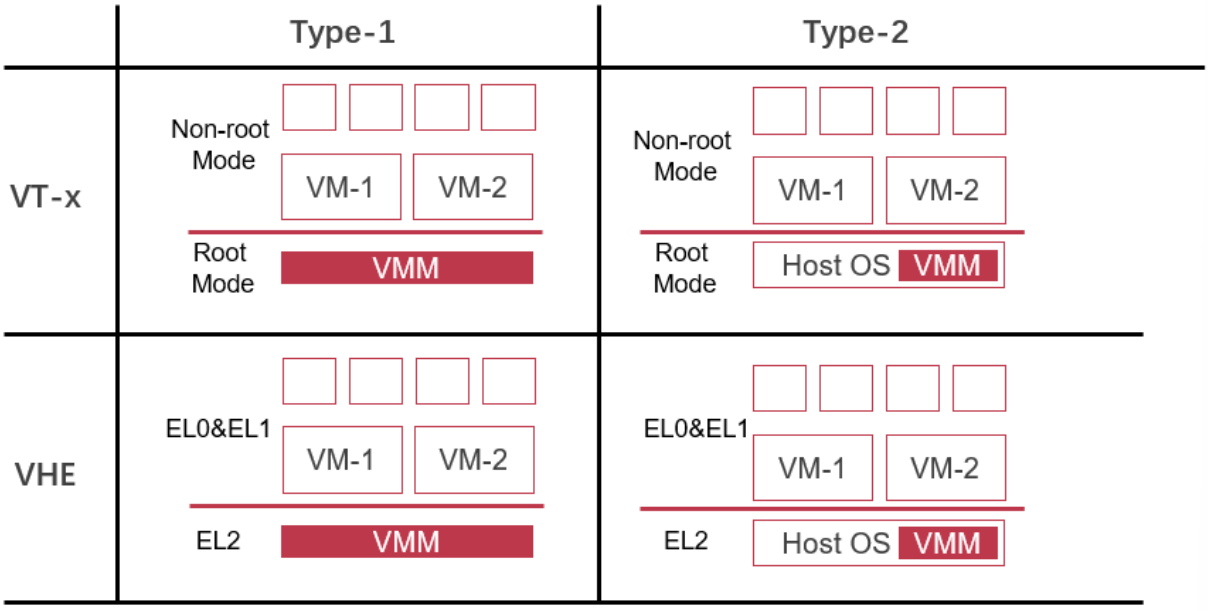
HCR_EL2寄存器简介

- **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

VT-x和VHE对比

	VT-x	VHE
新特权级	Root和Non-root	EL2
是否有VMCS?	是	否
VM Entry/Exit时硬件自动保存状态?	是	否
是否引入新的指令?	是(多)	是(少)
是否引入新的系统寄存器?	否	是(多)
是否有扩展页表(第二阶段页表)?	是	是

Type-1和Type-2在VT-x和VHE下架构



4. 案例：QEMU/KVM

QEMU发展历史

> 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

- 在2003-2006年，QEMU一直使用软件方法进行模拟

KVM发展历史

- 2005年11月，Intel发布带有VT-x的两款Pentium 4处理器
- 2006年中期，Qumranet公司在内部开发KVM(Kernel-based Virtual Machine)，并于11月发布
- 2007年，KVM被整合进Linux 2.6.20
- 2008年9月，Redhat出资1亿多美元收购Qumranet
- 2009年，QEMU 0.10.1开始使用KVM，以替代其软件模拟的方案

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捕捉所有的敏感指令和事件，传递给QEMU
    - KVM不提供设备的虚拟化，需要使用QEMU的虚拟设备