虚拟化技术与云计算

火龙果•整理
uml.org.cn

教学内容

- 绪论
- 计算体系架构与虚拟化技术
  - 虚拟化概述
  - **CPU的虚拟化**
  - 内存虚拟化
  - I/O虚拟化
  - 虚拟化实施与应用
  - 虚拟化的现状与未来

火龙果•整理
uml.org.cn

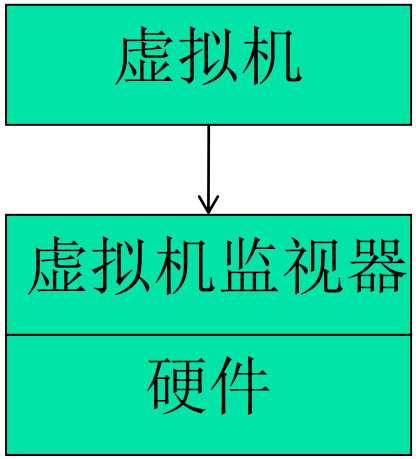
CPU虚拟化

- 处理器虚拟化概述
  - 可虚拟化条件
  - 指令模拟
- CPU软件虚拟化方案
- CPU硬件虚拟化方案

火龙果•整理
uml.org.cn

虚拟环境与可虚拟化条件

- 虚拟环境由三部分组成：硬件、虚拟机监视器（VMM）及虚拟机
- 虚拟机可以视为物理机的一种高效隔离的复制：同质、高效、资源受控
- 虚拟机的三个特点决定了不是所有体系结构都可以虚拟化



火龙果•整理
uml.org.cn

特权指令与敏感指令

- 多数计算机体系结构都有2个或以上的特权级，用来分隔系统软件和应用软件。
- 系统中有一些操作和管理关键系统资源的指令会被定为特权指令
  - 特权指令只有在最高特权级上能够正确执行
  - 如果在非最高特权级上运行，特权指令会引发异常，处理器会陷入到最高特权级
- 在虚拟化后，有一类指令称为敏感指令，即**操作特权资源的指令**
- 所有特权指令都是敏感指令；并不是所有敏感指令都是特权指令

火龙果•整理
uml.org.cn

陷入再模拟

- 为使VMM完全控制系统资源，不允许直接执行虚拟机操作系统（客户OS）的敏感指令
- 陷入再模拟
  - 系统的所有敏感指令都是特权指令（前提）
  - 将VMM运行在最高特权级，客户OS运行在非最高特权级
  - 陷入再模拟：当客户OS因执行敏感指令（此时也是特权指令）而陷入到VMM时，VMM模拟执行引起异常的敏感指令

火龙果•整理
uml.org.cn

虚拟化漏洞

- 一个系统是否可虚拟化：关键在该系统对敏感指令的支持上
  - 所有指令都是特权指令：可虚拟化的结构
  - 无法支持在所有敏感指令上触发异常：不是可虚拟化的结构，即存在“虚拟化漏洞”
- 一个结构存在虚拟化漏洞：存在敏感指令不属于特权指令

火龙果•整理
uml.org.cn

存在虚拟化漏洞系统的虚拟化方案

- 可采用一些方法弥补或避免虚拟化漏洞
- 模拟的方法：最直接
  - 解释执行：取一条指令，模拟出这条指令执行的效果，再取下一条指令
  - 不存在陷入问题，避免了虚拟化漏洞
  - 性能差
- 既要填补漏洞，又要保证性能，只能使用辅助手段：
  - 直接在硬件层面填补虚拟化漏洞
  - 通过软件避免虚拟机中使用到无法陷入的敏感指令

火龙果•整理
uml.org.cn

3个概念1：虚拟寄存器

- 未虚拟化的系统中，OS直接访问物理处理器，处于最高特权级，可以控制系统的所有关键资源：寄存器、内存、I/O外设等
- 虚拟化后，VMM接管物理处理器，以前的OS变成客户OS降级到非最高特权级。试图访问关键资源的指令成为敏感指令。
- VMM通过多种方法保证这些敏感指令的执行能够触发异常，从而陷入到VMM进行模拟
- 当客户OS试图访问关键资源时，该请求并不会真正发生在物理寄存器。VMM会通过准确模拟物理处理器的行为，将其访问定位到VMM为其设计与物理寄存器对应的“虚拟”的寄存器上。

火龙果•整理
uml.org.cn

3个概念2：上下文

- 进程上下文：某个时刻物理处理器中与运算相关的寄存器的状态构成了当前进程上下文
- 当OS进行调度时，当前进程的上下文（即寄存器状态）被保存在特定的内存区域中，而下一个进程的上下文被恢复到相应的寄存器中，从进程的角度看就好像从未被中断一样。
- 虚拟处理器上下文更复杂：客户OS本身包含许多敏感指令，会试图访问和修改物理处理器上定义的所有寄存器，这种访问和修改会被VMM重定位到虚拟处理器上。
- 对虚拟处理器，其上下文包括了更多的系统寄存器。

火龙果•整理
uml.org.cn

3个概念3：虚拟处理器

- 从客户OS来讲，其运行的虚拟处理器需要具备与其“期望”的物理处理器一致的功能和行为
- 从VMM角度，虚拟处理器是其需要模拟完成的一组功能集合。虚拟处理器的功能可以由物理处理器和VMM共同完成
  - 对非敏感指令，物理处理器直接解码处理其请求，并将相关效果直接反映到物理寄存器上；
  - 对于敏感指令，VMM负责陷入再模拟，从程序角度也就是一组数据结构与相关处理代码的集合
    - 数据结构用于存储虚拟寄存器的内容
    - 相关处理代码负责按照物理处理器的行为将效果反映到虚拟寄存器上。

火龙果•整理
uml.org.cn

VMM的陷入模式

- VMM陷入利用了处理器的保护机制，使用中断和异常来完成的，
- 基于处理器保护机制触发的异常
  - 敏感指令的执行
- 虚拟机主动触发异常。即陷阱
- 异步中断，包括处理器内部中断源和外部的设备中断源

火龙果•整理
uml.org.cn

CPU虚拟化方案

- CPU软件虚拟化方案
  - **模拟执行<->基于软件的完全虚拟化**
    - 解释执行
    - 代码扫描与修补
    - 二进制代码翻译
  - 直接源代码改写<->类虚拟化
- CPU硬件虚拟化方案

火龙果•整理
uml.org.cn

CPU软件虚拟化方案

- 在硬件设计未考虑虚拟化而导致虚拟化漏洞时，首先考虑到基于软件方式实现虚拟化
- 两种可行的基于软件的虚拟化解决方案：
  - 模拟执行<->基于软件的完全虚拟化
  - 直接源代码改写<->类虚拟化

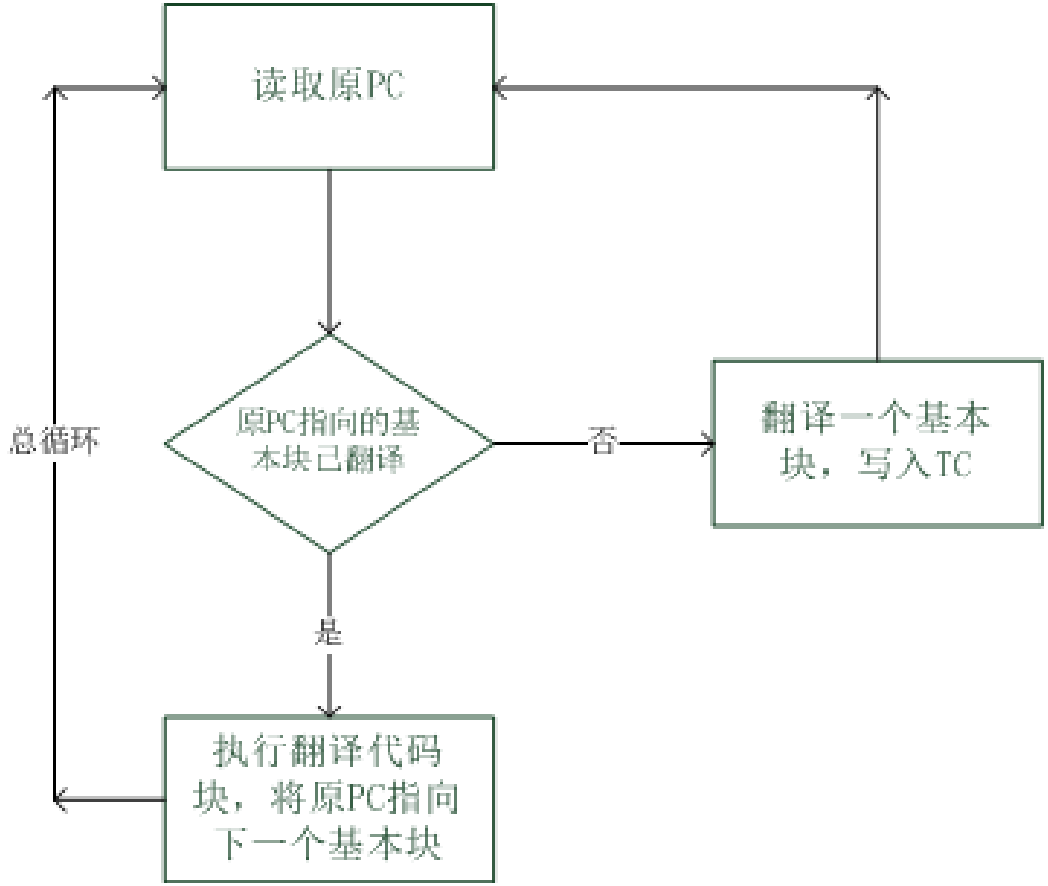
火龙果•整理
uml.org.cn

模拟技术

- 所有的虚拟化形式都可以用模拟来实现。三种常见的模拟技术：
  - 解释执行是最简单最直接的模拟实现方式
  - 代码扫描与修补
  - 二进制代码翻译
- 不同体系结构的虚拟化比较困难，通常使用解释执行
  - 在一种硬件体系结构上模拟出另一种不同硬件体系结构的运行环境
- 同一体系结构模拟：多数指令不需要被模拟执行而是直接在硬件上执行。可以使用效率更高的虚拟化方法：
  - 代码扫描与修补和二进制代码翻译。

火龙果•整理
uml.org.cn

模拟器架构



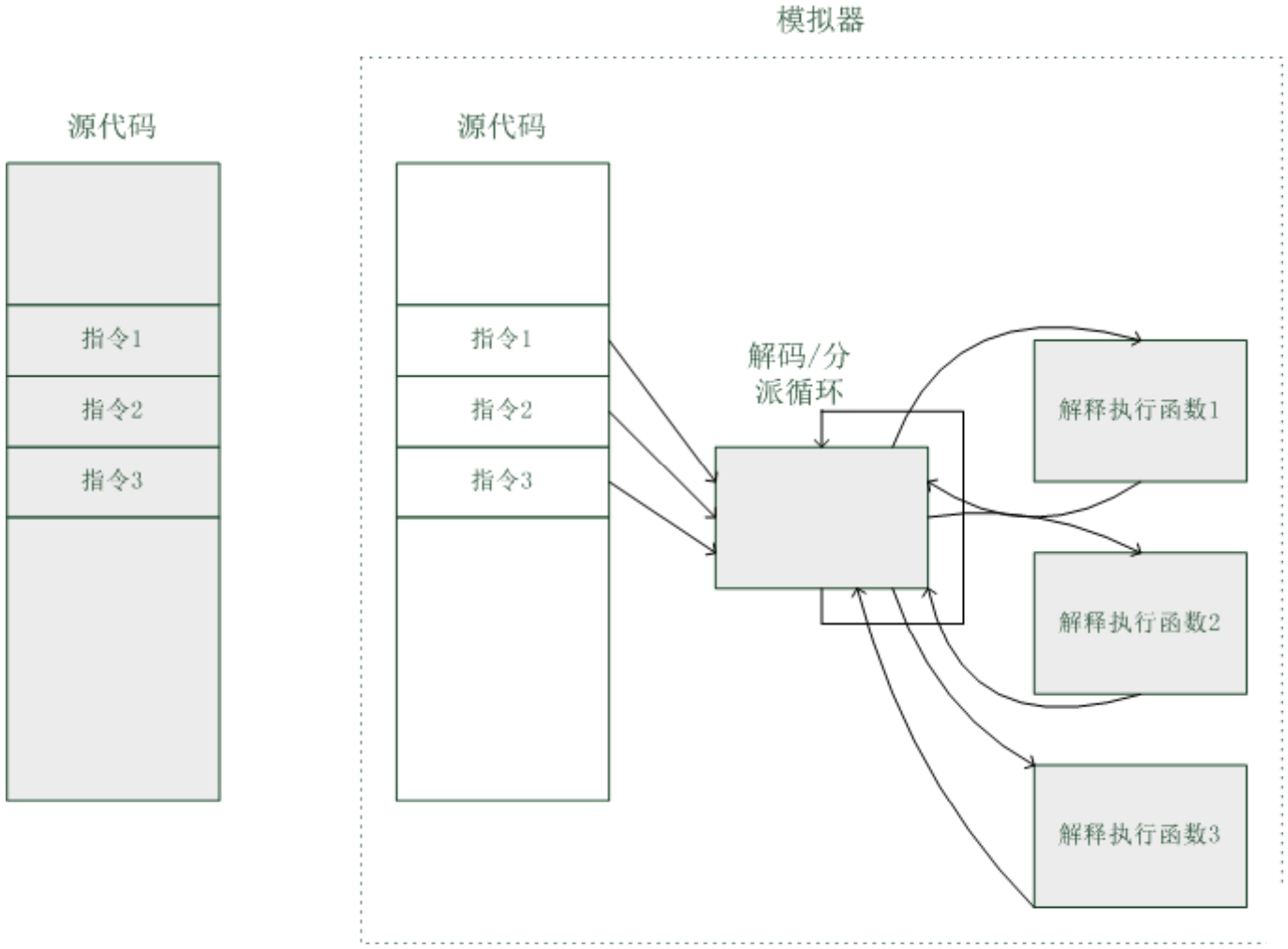
火龙果•整理
uml.org.cn

解释执行

- 解释执行是最简单的一种模拟技术
- 取一条执行，模拟出这条指令执行的效果，再取一条指令，周而复始。
- 从一定程度上解决陷入再模拟

火龙果•整理
uml.org.cn

指令的正常执行与解释执行



火龙果•整理
uml.org.cn

解释执行优缺点

- 模拟 取指/译码/执行 软件流水线
- 优点
  - 容易实现
  - 复杂度最低
- 缺点
  - 慢

火龙果•整理
uml.org.cn

扫描与修补

- 虚拟机中模拟的CPU与物理CPU体系结构相同时，大多数指令可以映射到物理CPU而直接执行
- 扫描与修补：多数指令直接在物理CPU执行，把OS代码中的敏感指令替换为跳转指令或会陷入到VMM中去的指令，一旦运行到敏感指令处控制流就会进入VMM中，由VMM代为模拟执行

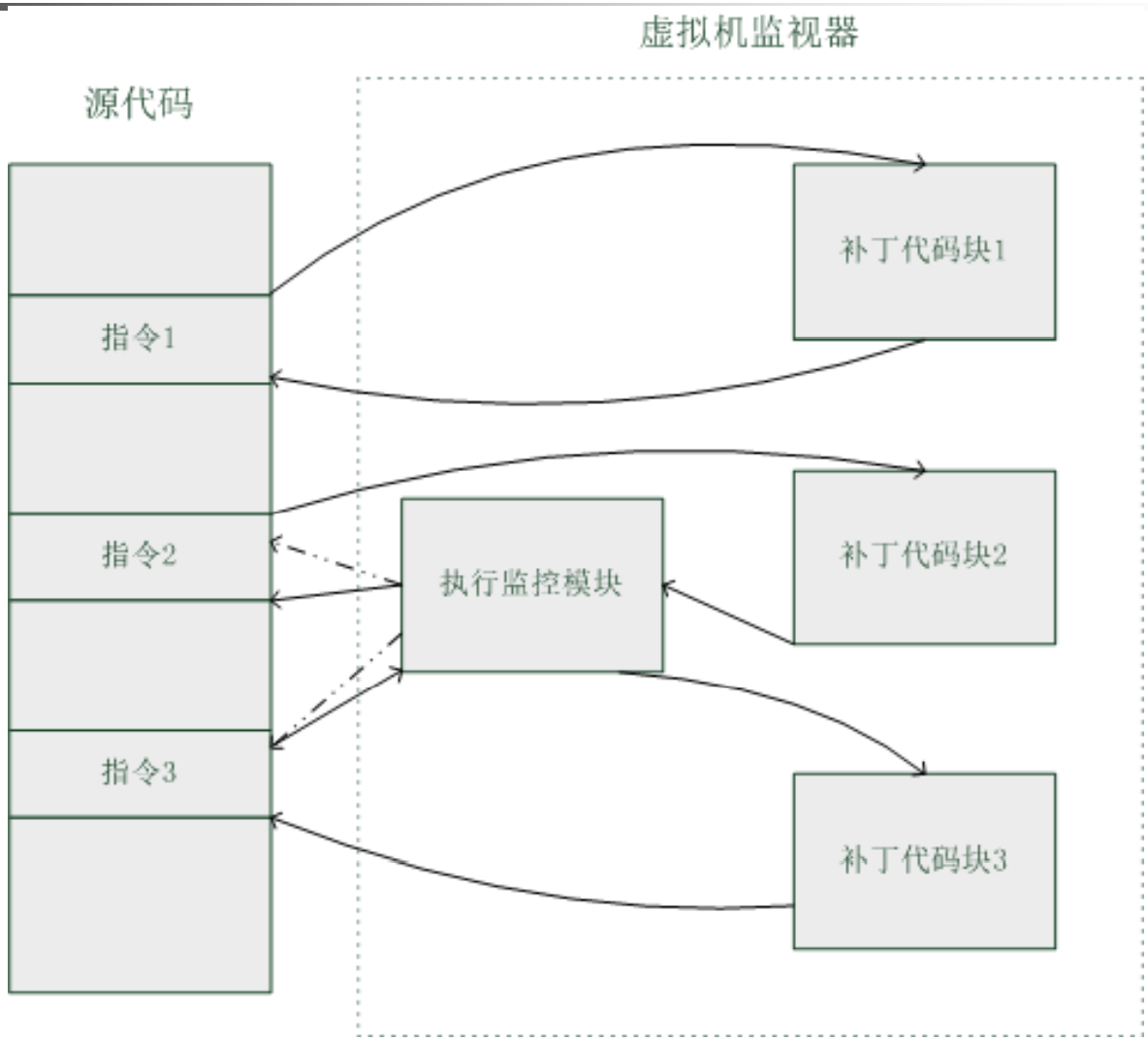
火龙果•整理
uml.org.cn

扫描与修补的流程

- VMM在虚拟机开始执行每段代码之前对其进行扫描，解析每一条指令，查找到特权指令和敏感指令
- 补丁代码会在VMM中动态生成，通常每一个需要修补的指令会对应一块补丁代码
- 敏感指令被替换成一个外跳转，从虚拟机跳转到VMM的空间里，在VMM中执行动态生成的补丁代码
- 当补丁代码执行完成后，执行流再跳转回虚拟机中的下一条代码继续执行

火龙果•整理
uml.org.cn

扫描与修补原理



火龙果•整理
uml.org.cn

扫描与修补的优缺点

- 异常的处理较简单
- 实现较简单
- 性能损失小
- 特权指令和敏感指令都被模拟执行，各条指令的模拟执行时间可能会很短，也可能会较长
- 每个补丁都引入额外的跳转，会降低代码的局部性
- 扫描与修补直接在虚拟机内存中进行代码修补，需维护一份与补丁对应的原始代码的备份，以在需要时将代码恢复

火龙果•整理
uml.org.cn

二进制代码翻译

- 二进制代码翻译（BT）在VMM中开辟一块代码缓存，将代码翻译好放在其中
- 客户OS代码不会直接被物理CPU执行，所有要被执行的代码都在代码缓存中

火龙果•整理
uml.org.cn

二进制翻译的单位：基本块

- BT以基本块为单位翻译代码
  - 模拟器动态按需读入二进制代码进行翻译，将翻译好的目标代码存放在模拟器开辟的内存空间（称为代码缓存）
  - 动态基本块
- 源代码中的指令与翻译后的代码用某种映射关系进行关联
  - 常用的是哈希表：由源代码的PC值通过哈希函数计算查表得到其在代码缓冲区中的位置

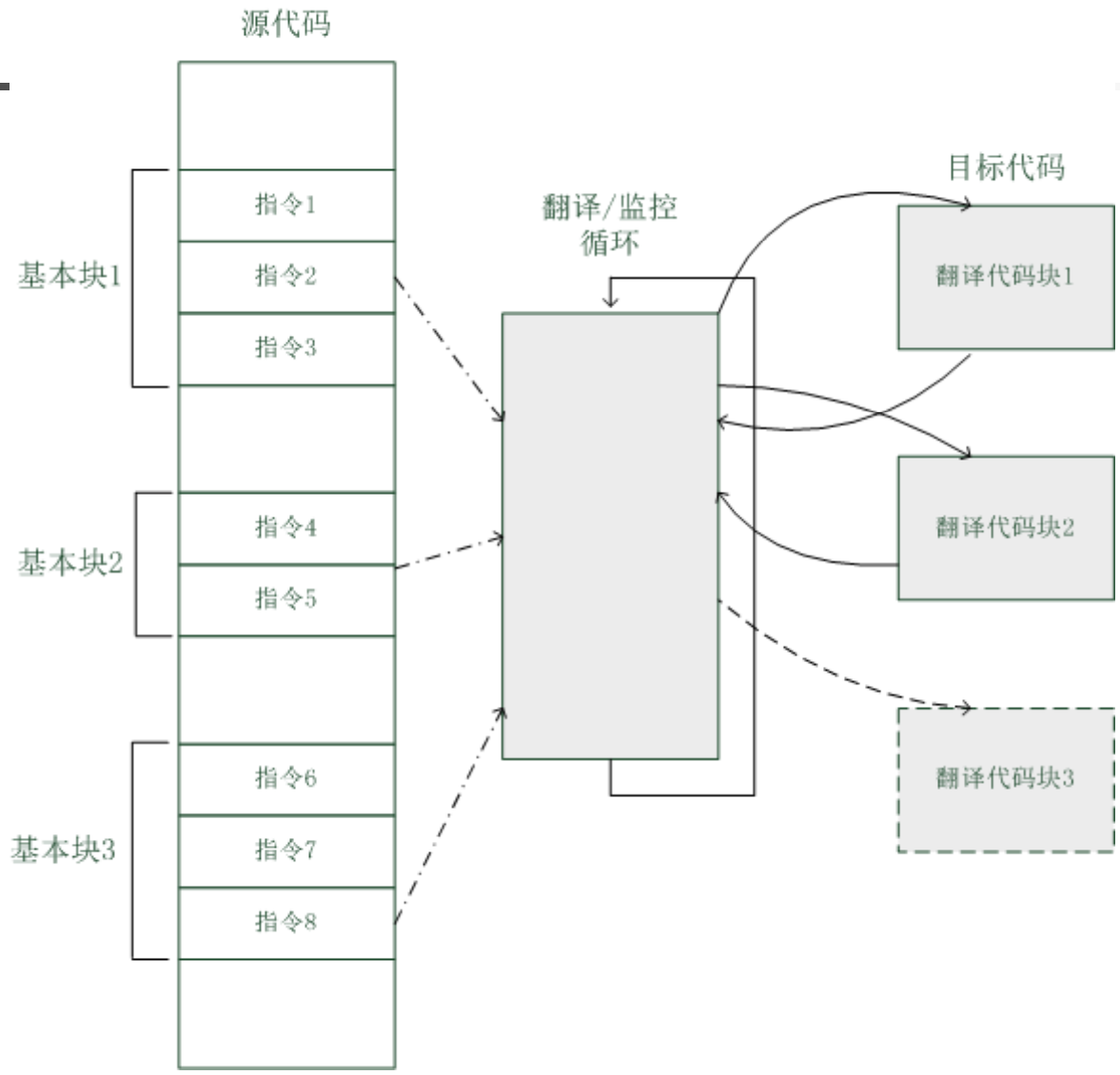
火龙果•整理
uml.org.cn

翻译

- 模拟器对读入的二进制代码不作限制
  - 应用程序的代码
  - OS内核代码
- 读入的二进制代码可能包含所有的x86体系结构的指令，模拟器将其翻译输出为x86指令的子集，即不包含特权指令和敏感指令，能够运行在用户态

火龙果•整理
uml.org.cn

二进制代码翻译



火龙果•整理
uml.org.cn

BT技术的难点

BT技术在处理过程中会遇到困难的几种情况：

- 自修改代码(Self Modifying Code)
- 自参考代码(Self Referential Code)
- 精确异常(Precise Exceptions)
- 实时代码

火龙果•整理
uml.org.cn

BT技术的优化

BT在发展过程中积累了以下优化技术来提高整体性能：

- 基本块串联
- 自适应翻译
- 指令缓存布局优化

火龙果•整理
uml.org.cn

模拟方式的优势与不足

- 模拟方式的优势
  - VMM可以将虚拟机整个执行过程置于控制中，VMM执行每一条指令都进行模拟，不会漏过需要模拟的敏感指令
  - 模拟技术应用范围广
    - 系统级
    - 应用程序级
- 模拟方式的缺陷
  - 效率较低

火龙果•整理
uml.org.cn

CPU的硬件辅助虚拟化

- 硬件辅助虚拟化
- **CPU虚拟化的硬件支持**
  - Intel VT-x的思想
    - 两种操作模式
    - VMCS
    - 虚拟化新指令
  - VMCS
    - 概念
    - VMCS与物理CPU的绑定
  - VMX操作模式
    - VM-Entry
    - VM-Exit
- CPU虚拟化的实现

火龙果•整理
uml.org.cn

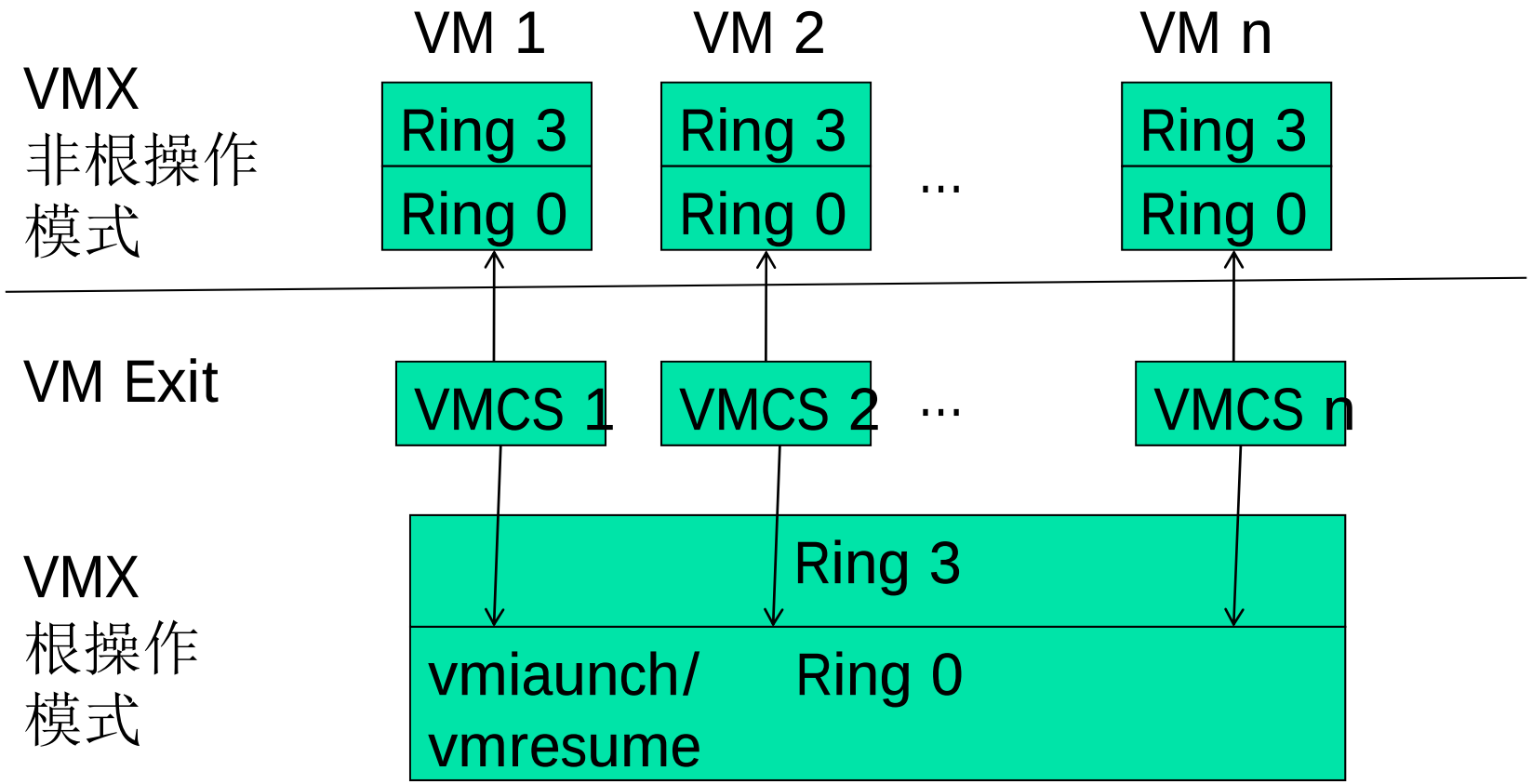
硬件辅助虚拟化

- 硬件辅助虚拟化：在CPU、芯片组以及I/O设备等硬件中加入专门针对虚拟化的支持，以简化虚拟化的实现，提高虚拟化的效率
- 在硬件中加入虚拟化支持的原因：
  - 原有硬件体系结构在虚拟化方面存在缺陷，导致单纯软件虚拟化方法存在问题；
  - 某些虚拟化功能尽管可用软件方法来实现，但实现复杂；
  - 某些通过软件方法实现的虚拟化功能性能不佳
- Intel提供了VT-x（Intel Virtualization Technology for x86）的CPU虚拟化技术

火龙果•整理
uml.org.cn

Intel VT-x架构

- VT-x的基本思想可如下如所示：



火龙果•整理
uml.org.cn

Intel VT-x新模式的引入

VT-x引入了新的模式：

- CPU指令的虚拟化通过“陷入再模拟”实现
- IA-32架构有些敏感指令不能通过“陷入再模拟”的方式处理，导致了虚拟化漏洞
- 新模式（非根模式）下所有敏感指令的行为都被重新定义，使它们**不经虚拟化就直接运行或通过“陷入再模拟”的方式处理**

火龙果•整理
uml.org.cn

Intel VT-x的两种模式

- VT-x引入了两种操作模式，统称为VMX操作模式：
  - 根操作模式（VMX Root Operation）：VMX运行所处的模式，简称根模式
  - 非根操作模式（VMX Non-Root Operation）：客户机运行所处的模式，简称非根模式
- 两种操作模式与IA32特权级0~特权级3是正交的

火龙果•整理
uml.org.cn

两种模式的切换

- VT-x中，非根模式下敏感指令引起的“陷入”被称为**VM-Exit**
- VM-Exit发生时，CPU自动从非根模式切换成根模式
- **VM-Entry**操作由VMM发起，通常是调度某个客户机运行
- VM-Entry发生时，CPU从根模式切换成非根模式

火龙果•整理
uml.org.cn

虚拟机控制结构&新指令

- VMCS：
  - VT-x引入VMCS（Virtual-Machine Control Structure，虚拟机控制结构），以更好地支持CPU虚拟化。VMCS保存虚拟CPU需要的相关状态，供CPU使用
- 新指令：
  - VT-x引入一组新的指令，包括VMLAUCH/VMRESUME用于发起VM-Entry，VMREAD/VMWRITE用于配置VMCS等

火龙果•整理
uml.org.cn

VMCS概念

- VMCS的概念与虚拟寄存器的概念类似，可看作是虚拟寄存器概念在硬件上的应用
  - 虚拟寄存器的操作和更改完全由软件执行
  - VMCS主要由CPU操作
- VMCS是保存在内存中的数据结构，包含了虚拟CPU的相关寄存器的内容和虚拟CPU相关的控制信息

火龙果•整理
uml.org.cn

VMCS与物理CPU的绑定

- 每个VMCS对应一个虚拟CPU。VMCS在使用时需要与物理CPU绑定
  - 任何给定时刻，VMCS与物理CPU是一对一的绑定关系：
    - 一个物理CPU只能绑定一个VMCS
    - 一个VMCS也只能与一个物理CPU绑定
- VT-x提供了两条指令用于VMCS的绑定与解除绑定
  - VMPTRLD<VMCS地址>：将指定的VMCS与执行该指令的物理CPU绑定
  - VMCLEAR：将执行该指令的物理CPU与它的VMCS解除绑定

火龙果•整理
uml.org.cn

VMCS数据域及访问

- VMCS信息存在放VMCS数据域
- VT-x提供2条指令访问VMCS:
  - VMREAD <索引>：读VMCS中“索引”指定的域
  - VMWRITE <索引> <数据>：写VMCS中“索引”指定的域
- VMCS数据域包括下列6大类信息：
  - 客户机状态域
  - 宿主机状态域
  - VM-Entry控制域
  - VM-Execution控制域
  - VM-Exit控制域
  - VM-Exit信息域

火龙果•整理
uml.org.cn

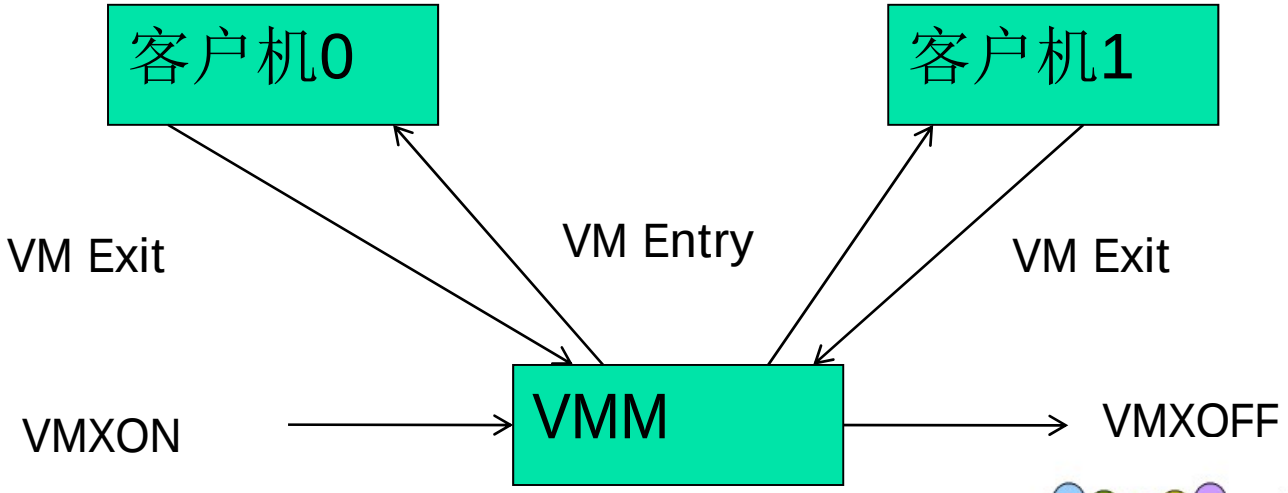
客户机状态域&宿主机状态域

- 客户机状态域：
  - 用于保存CPU在非根模式下运行时的CPU状态
  - 当发生VM-Entry时，CPU自动将客户机状态域保存的状态加载到CPU中
  - 当发生VM-Exit时，CPU自动将CPU的状态保存回客户机状态域
- 宿主机状态域：
  - 用于保存CPU在根模式下运行时的CPU状态。宿主机状态域只在VM-Exit时被恢复，在VM-Entry时不用保存

火龙果•整理
uml.org.cn

VMX操作模式

- OS没有虚拟化时，VMX操作模式默认关闭
- 当VMM需要使用这个功能时，可以使用VT-x提供的新指令来打开或关闭这个功能：
  - VMXON：打开VMX操作模式
  - VMXOFF：关闭VMX操作模式
- VMX操作模式



火龙果•整理
uml.org.cn

VM-Entry

- VM-Entry：VMM在机器加电引导后，进行初始化工作，准备就绪时通过VMXON指令进入根模式
  - VM-Entry是指CPU由根模式切换到非根模式
  - 从软件角度， VM-Entry是指CPU从VMM切换到客户机执行
  - VM-Entry操作通常由VMM主动发起
- VT-x为VM-Entry提供了两条指令：
  - VMLAUNCH：用于刚执行过VMCLEAR的VMCS的第一次VM-Entry
  - VMRESUME：用于执行过VMLAUNCH的VMCS的后续VM-Entry

火龙果•整理
uml.org.cn

VM-Entry的过程

当CPU执行VMLAUNCH/VMRESUME进行VM-Entry时，CPU要进行下面的步骤：

- 执行基本的检查来确保VM-Entry能开始
- 对VMCS中的宿主机状态域的有效性进行检查，以确保下一次VM-Exit发生时可以正确的从客户机环境切换到VMM环境
- 检查VMCS中客户机状态域的有效性；根据VMCS中客户机状态域区域来装载处理器的状态
- 根据VMCS中VM-Entry MSR-load区域装载MSR寄存器
- 根据VMCS中VM-Entry时间注入控制的配置，可能需要注入一个时间到客户机中

火龙果•整理
uml.org.cn

VM-Exit

- VM-Exit是指CPU从非根模式切换到根模式，从客户机切换到VMM的操作
- 引发VM-Exit的操作很多，如：
  - 在非根模式时执行了敏感指令
  - 发生了中断
- 处理VM-Exit事件是VMM虚拟指令、虚拟特权资源的一大任务

火龙果•整理
uml.org.cn

VM-Exit -2 非根模式下的敏感指令

- 当成功执行VM-Entry之后，CPU就进入了非根模式。若敏感指令运行在VMX非根操作模式，其行为可能发生变化，具体来说有三种可能：
  - 行为不变化，但不引起VM-Exit
  - 行为变化，产生VM-Exit
  - 行为变化，产生VM-Exit可控
- 由此可见，使用VT-x技术实现的VMM， 不需要对所有敏感指令进行模拟，减小了VMM实现的复杂性

火龙果•整理
uml.org.cn

VM-Exit -3 VM-Execution控制域

- VM-Execution控制域用来控制CPU在非根模式运行时的行为，根据虚拟机的实际应用，VMM可以通过配置VM-Execution控制域达到性能优化的目的。此控制域主要控制三个方面：
  - 控制某条敏感指令是否产生VM-Exit，如果产生VM-Exit，则由VMM模拟该指令
  - 在某些敏感指令不产生VM-Exit时，控制该指令的行为
  - 异常和中断是否产生VM-Exit

火龙果•整理
uml.org.cn

VM-Exit -4 VM-Exit 控制域和信息域

- VM-Exit控制域
  - 规定了VM-Exit发生时CPU时的行为；
- VM-Exit信息域
  - VMM通过VM-Exit信息域知道VM-Exit的相关信息（如退出原因）；
  - VM-Exit信息域提供的信息可以分为如下四类:
    - 基本的EM-Exit信息，包括Exit Reason、Exit qualification；
    - 事件触发导致的VM-Exit的信息；
    - 事件注入导致的VM-Exit的信息；
    - 执行指令导致的VM-Exit的信息；

火龙果•整理
uml.org.cn

VM-Exit的过程

- VM-Exit的大致流程基本为：
  - CPU首先将此次VM-Exit的原因信息记录到VMCS相应的信息域中，VM-Entry interruption-information字段的有效位（bit31）被清零；
  - CPU状态被保存到VMCS客户机状态机；
  - 根据VMCS中宿主机状态域和VM-Exit控制域中的设置，将宿主机状态加载到CPU相应寄存器；
  - CPU由非根模式切换到了根模式，从宿主机状态域中CS：RIP指定的VM-Exit入口函数开始执行。

火龙果•整理
uml.org.cn

CPU虚拟化的实现

- VCPU的概念及结构
- VCPU的创建
- VCPU的运行
  - 上下文切换
  - 上下文切换优化
  - 运行举例
- VCPU的退出
- VCPU的再运行

火龙果•整理
uml.org.cn

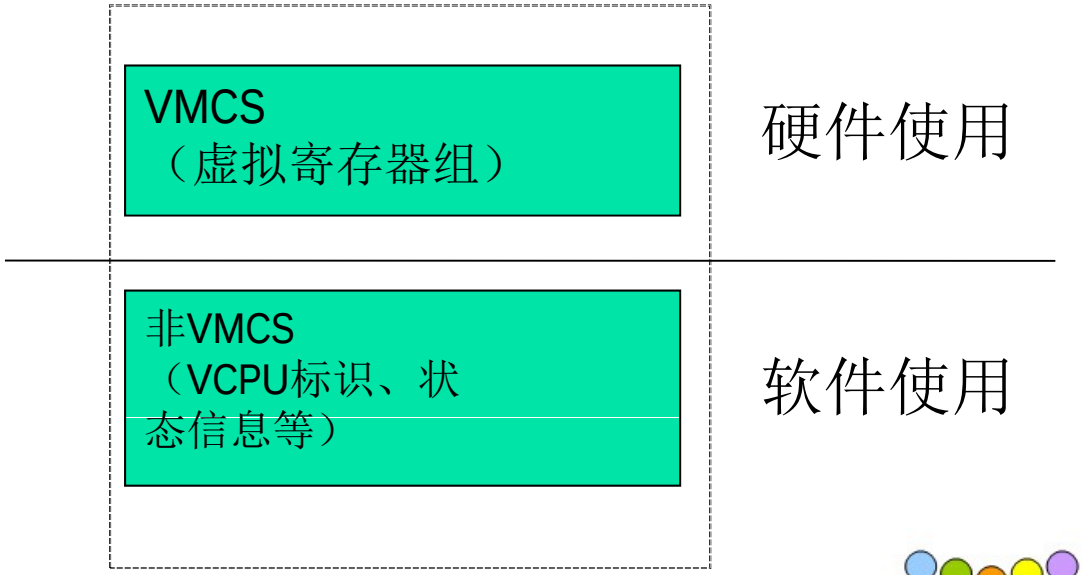
CPU虚拟化的实现

- 硬件虚拟化使用VCPU（Virtual CPU）描述符来描述虚拟CPU
- VCPU描述符类似操作系统中进程描述符（或进程控制块），本质是一个结构体，由下列几部分组成：
  - VCPU标识信息
  - 虚拟寄存器信息
  - VCPU状态信息
  - 额外寄存器/部件信息
  - 其他信息

火龙果•整理
uml.org.cn

VCPU结构

- Intel VT-x的VCPU可划分成两部分：
  - 以VMCS为主由硬件使用和更新的部分，这主要是虚拟寄存器；
  - 除VMCS之外，由VMM使用和更新的部分，主要指VMCS之外的部分；
- VCPU结构：



火龙果•整理
uml.org.cn

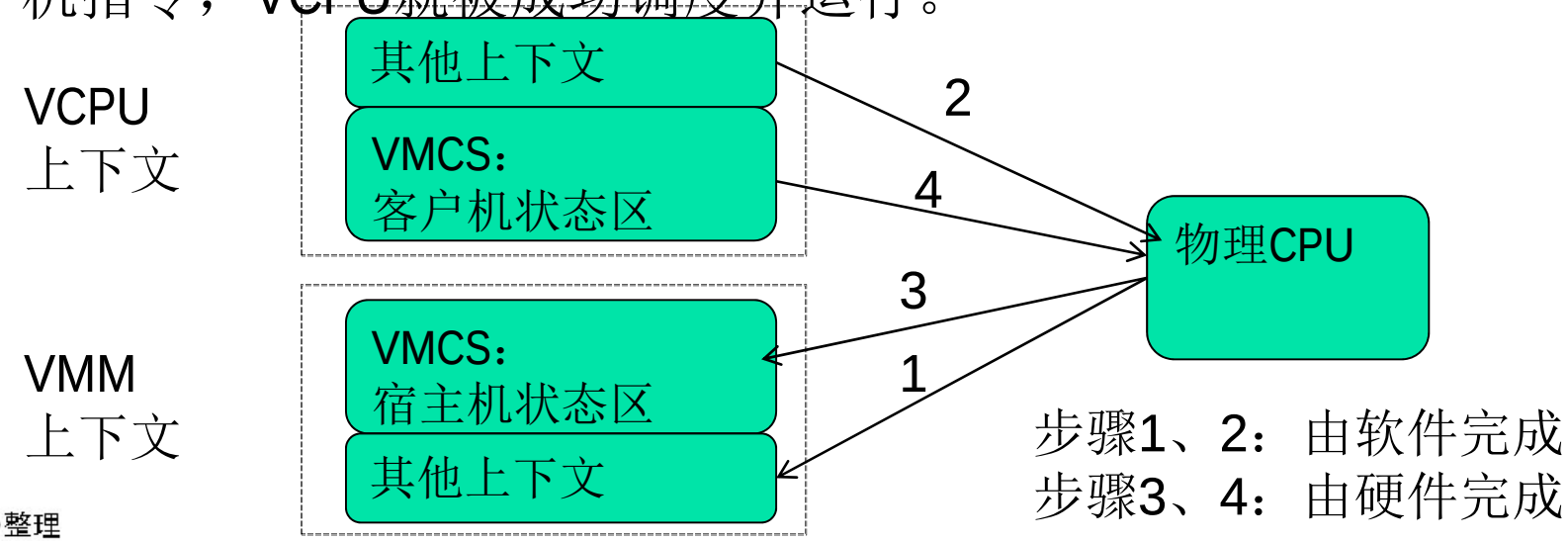
VCPU的创建

- 创建VCPU实际上是创建VCPU描述符，即分配相应大小的内存
- VCPU描述符创建之后，需要进一步初始化才能使用。通常初始化包含如下内容：
  - 分配VCPU标识
  - 初始化虚拟寄存器组
  - 初始化VCPU状态信息
  - 初始化额外部件
  - 初始化其他信息

火龙果•整理
uml.org.cn

上下文切换过程

- VT-x支持的CPU上下文切换的过程：
  - VMM保存自己的上下文，主要是保存VMCS不保存的寄存器，即宿主机状态域之外的部分；
  - VMM将保存在VCPU中的由软件切换的上下文加载到物理CPU中；
  - VMM执行VMRESUME/VMLAUNCH指令，触发VM-Entry，此时CPU自动将VCPU上下文中VMCS部分加载到物理CPU，CPU切换到非根模式。
- 此时，物理CPU已经处于客户机的运行环境了，开始执行客户机指令，VCPU就被成功调度并运行。



火龙果•整理
uml.org.cn

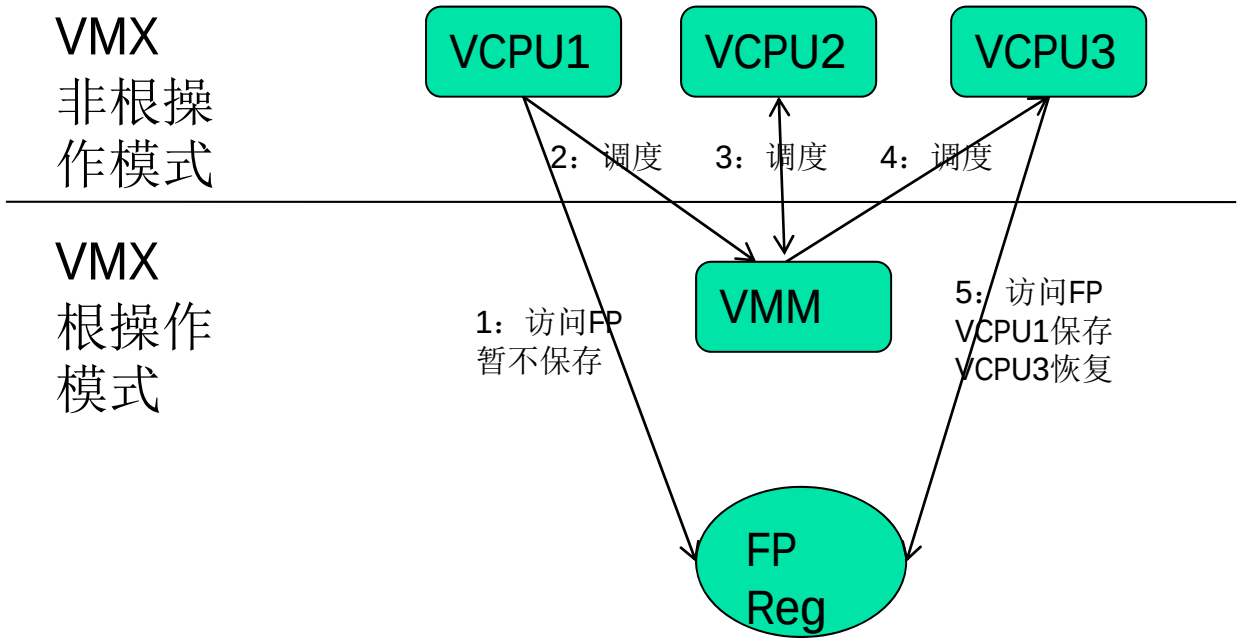
上下文切换优化

- VMM使用“惰性保存/恢复”的方法进行优化：尽量将寄存器的保存/恢复延迟到最后一刻，即其他VCPU或VMM需要用该寄存器的时候再保存/恢复
- 具体来说，VMM通过考察资源的用情况来实现“惰性保存/恢复”
  a) 对于VMM需要使用的寄存器，每次VCPU和VMM切换时都要保存/恢复。
  b) 对于VMM没有使用的寄存器，如果VMM无法知道VCPU是否在最近的执行中曾经修改了这个寄存器，那么在VCPU和VMM切换时，不需要对这个寄存器进行保存和恢复。
  c) 对于VMM没有使用的寄存器，如果VMM可以知道客户机是否在最近的执行中修改了这个寄存器（如浮点寄存器），还可以做进一步的优化。

火龙果•整理
uml.org.cn

VCPU的运行举例

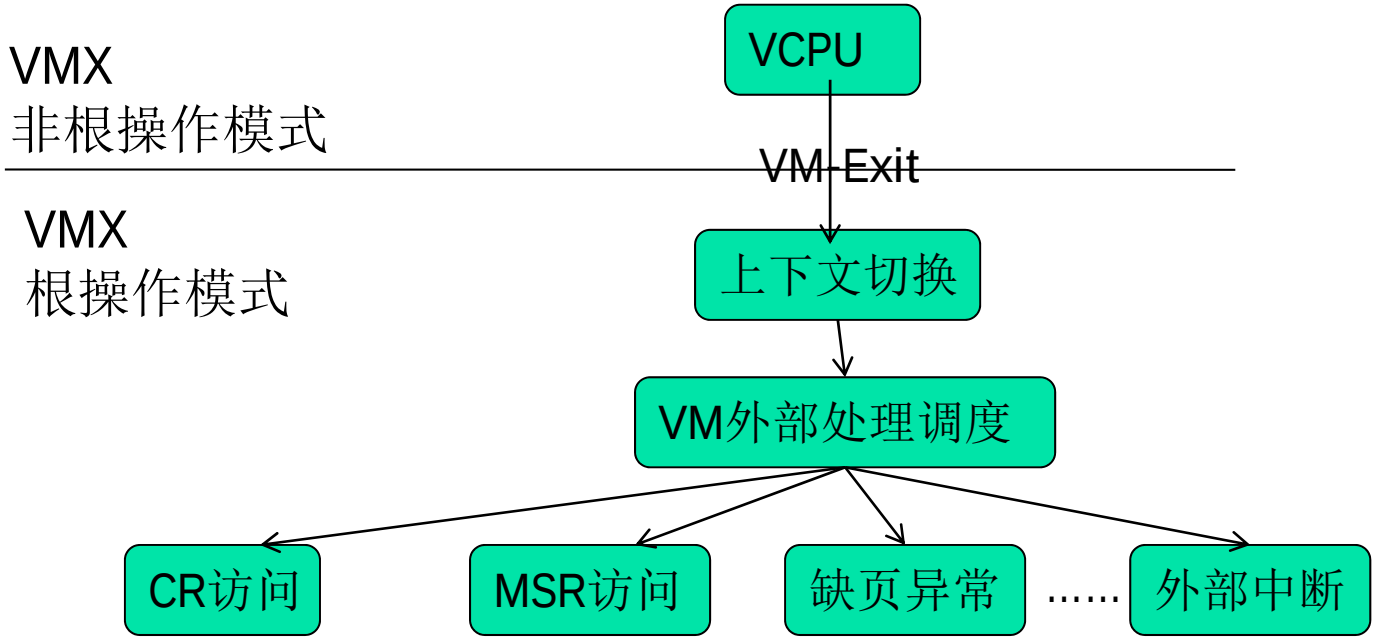
- 举例说明如下：



火龙果•整理
uml.org.cn

VCPU的退出

- 和进程一样，VCPU作为调度单位不可能永远运行，总会因为各种原因退出，例如执行了特权指令、发生了物理中断等。这种退出在VT-x中表现为发生VM-Exit
- 对VCPU退出的处理是VMM进行CPU虚拟化的核心，如模拟各种特权指令。
- VM-Exit的处理为



火龙果•整理
uml.org.cn

VCPU的再运行

VMM在处理完VCPU的退出后，会负责将VCPU投入再运行：

- 如果VCPU继续在相同的物理CPU上运行，可以用VMRESUME来实现VM-Entry
- 如果由于某种原因（如负载均衡），VCPU被调度程序迁移到了另外一个物理VCPU上，那么VMM需要做如下几件事情
  - 将VCPU对应的VMCS迁移到另外一个物理CPU
  - 迁移完成后，在重新绑定的物理CPU上执行**VMLAUNCH**发起VM-Entry

火龙果•整理
uml.org.cn