RVM: Rcore Virtual Machine

清华大学 计算机科学与技术系

贾越凯

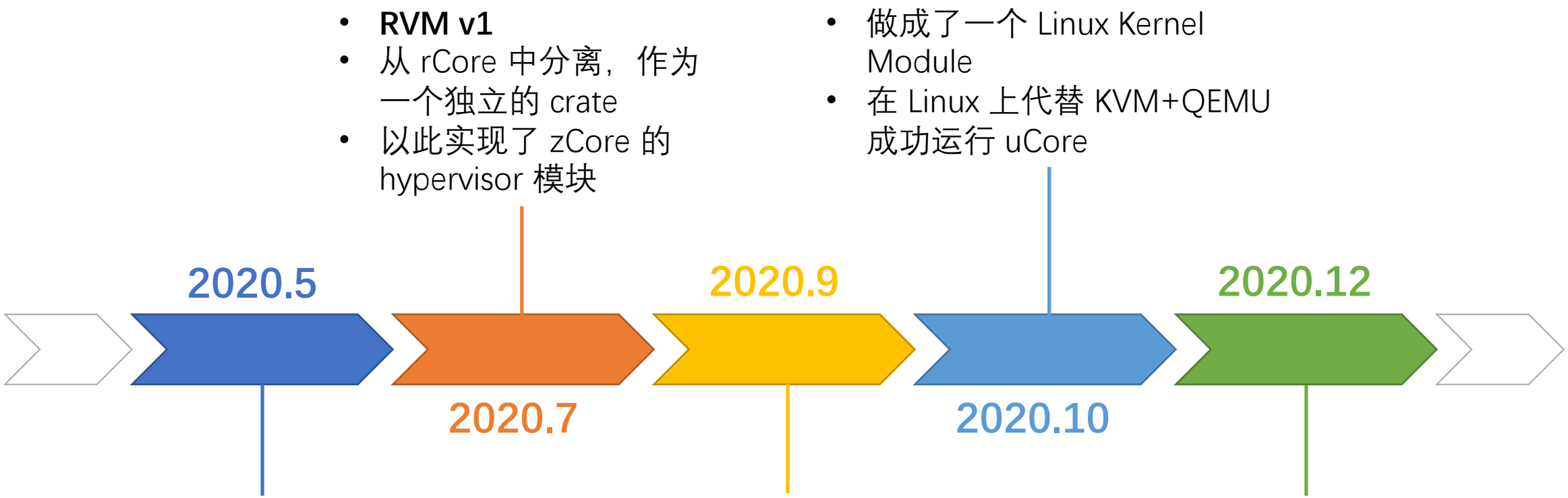
2020/12/26

项目介绍

- RVM 是什么？
  - **R**core/**R**ust **V**irtual **M**achine
  - Hypervisor、Virtual Machine Monitor (VMM)
- 项目定位
  - 用 Rust 语言写 hypervisor 的尝试
  - 适用于虚拟化技术的教学与研究

https://github.com/rcore-os/RVM
https://github.com/rcore-os/RVM1.5

项目历史

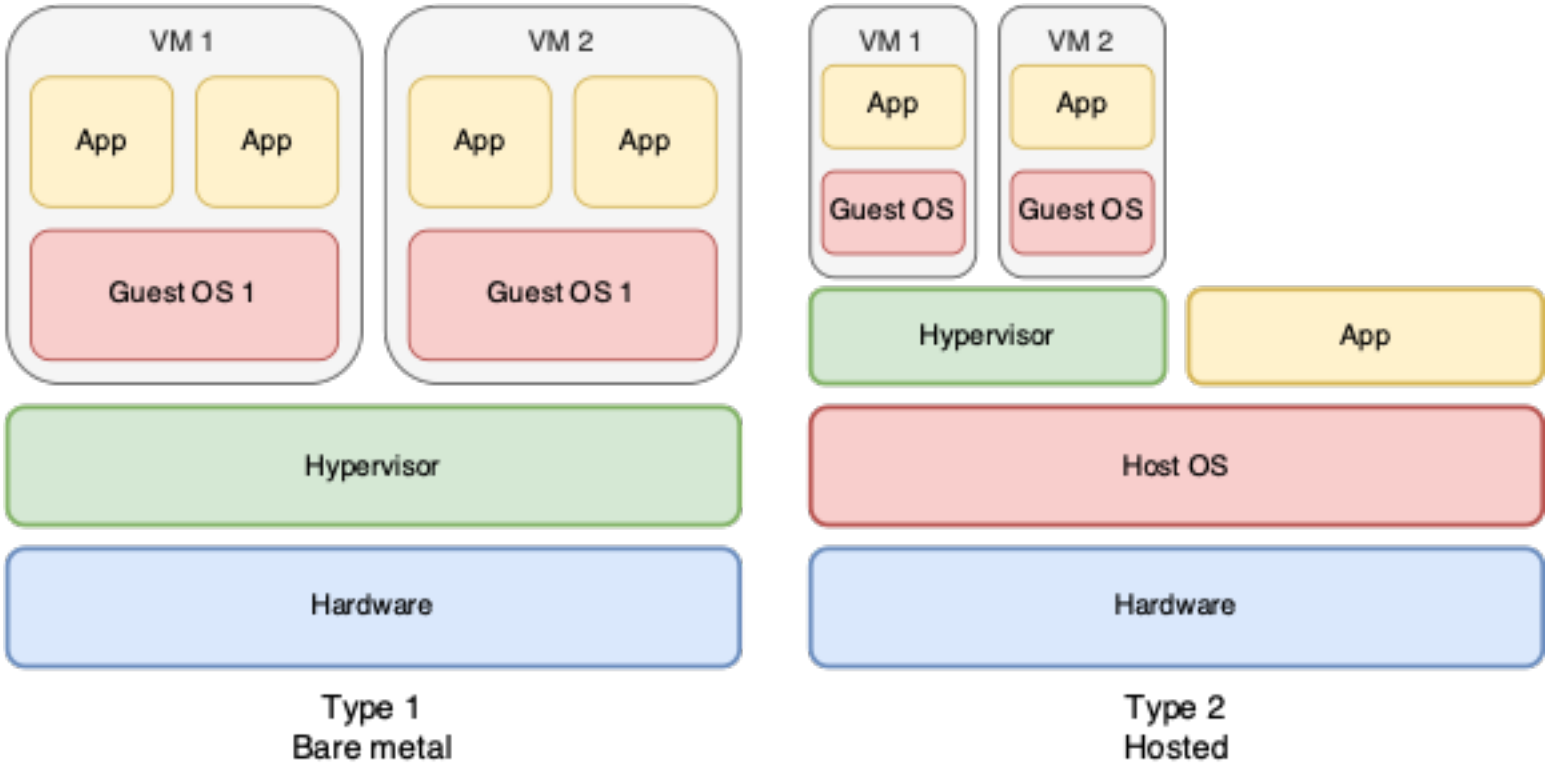

2020.5
2020.7
2020.9
2020.10
2020.12
RVM v0.1
rCore 子模块
成功在 rCore 上运行 uCore
RVM v1
从 rCore 中分离，作为一个独立的 crate
以此实现了 zCore 的 hypervisor 模块
通过调用 RVM crate 的方式，重新集成进 rCore 中
模块化程度更好
做成了一个 Linux Kernel Module
在 Linux 上代替 KVM+QEMU 成功运行 uCore
RVM v1.5
Type-1.5 hypervisor

Hypervisor 基本概念

- 虚拟化技术
  - 在一台计算机上模拟另一台计算机的执行
  - 系统级虚拟化
  - 广泛应用于云计算领域
- 基本概念
  - Host
  - Guest
  - Hypervisor / Virtual Machine Monitor(VMM)

Hypervisor 基本概念

- Type-1 与 Type-2

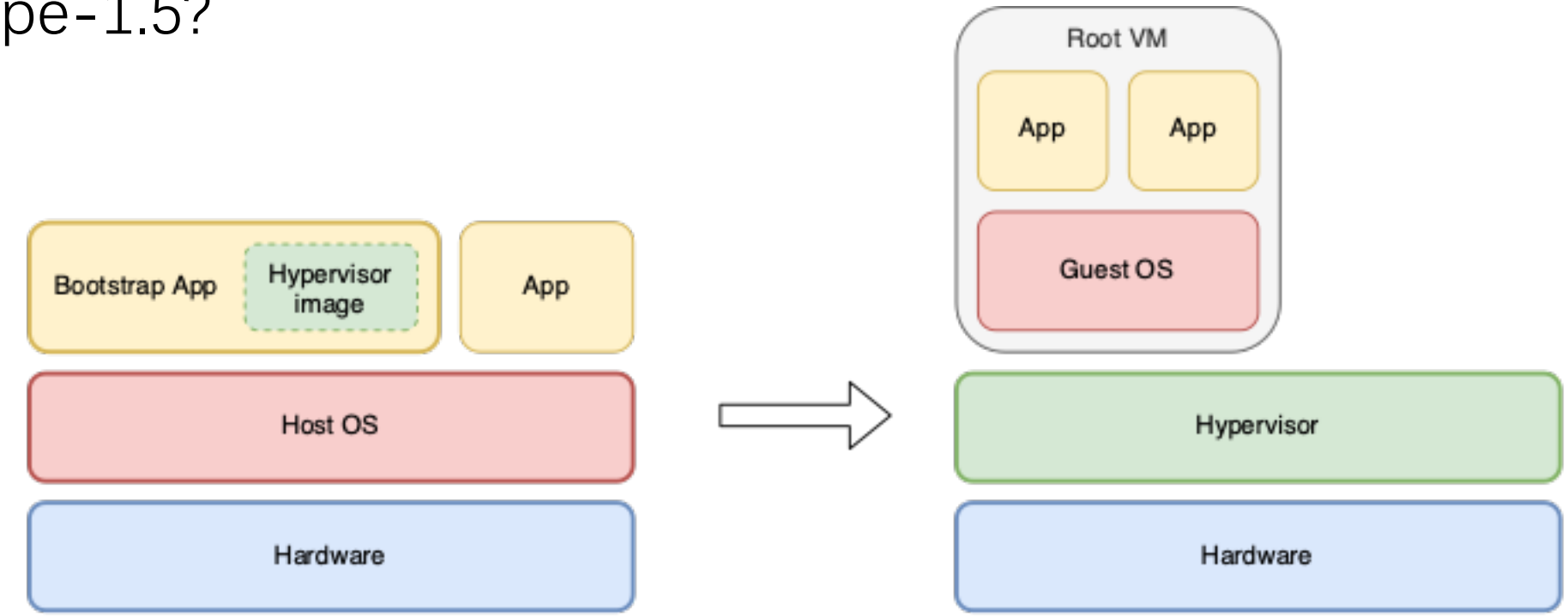


Xen

KVM、Virtual Box、QEMU

Hypervisor 基本概念

- Type-1.5?



Type 1.5

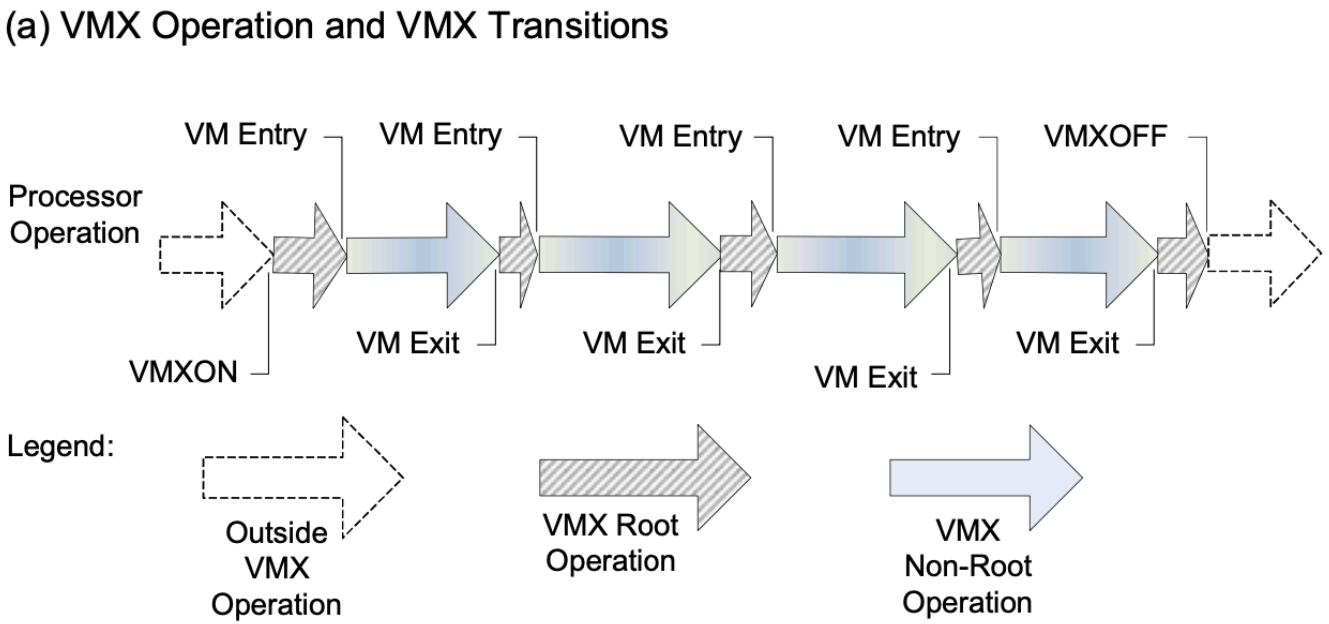
Jailhouse、Bareflank

Hypervisor 基本概念

- 高效实现虚拟化的基本条件
  - 对于 Guest 的特权指令，执行时要发生陷入(trap)，交给 hypervisor 处理
  - 对于 Guest 的非特权指令，要能被尽可能高效地执行
- 软件虚拟化
  - 二进制翻译、JIT
- 硬件虚拟化
  - x86：Intel VT-x、AMD SVM
  - ARMv8：EL2
  - RISC-V：H 扩展

Intel VT-x 指令扩展

- Virtual-Machine eXtensions (VMX)
- 开启 VMX 后的特殊模式：
  - VMX root operation
  - VMX non-root operation
- VM Entry/VM Exit

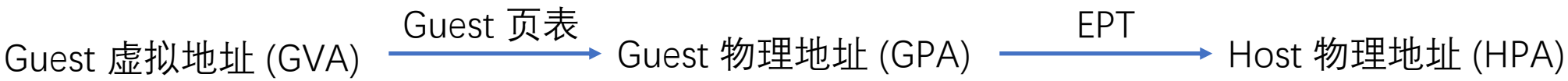


Virtual-Machine Control Structures (VMCS)

- Guest 特权状态 (VM entry 时载入，VM exit 时保存)
  - CR0、CR3、CR4
  - 段选择子、基址、权限：CS、SS、DS、ES、FS、GS、GDTR、IDTR
  - RSP、RIP、RFLAGS
- Host 特权状态 (VM exit 时载入，VM entry 时保存)
  - CR0、CR3、CR4
  - 段选择子、基址： CS、SS、DS、ES、FS、GS、GDTR、IDTR
  - RSP、RIP
- 控制字段
  - 对 VM entry/exit/execution 时的行为进行配置
- 只读字段
  - 提供 VM Exit 时的一些信息

内存虚拟化

- 可以为每个 VM 划分地址空间
- Extended Page Table (EPT)
- 和页表结构相似，细节不同



	Page Table	Extended Page Table
地址转换	虚拟地址 → 物理地址	Guest 物理地址 → Host 物理地址
设置基址	CR3 寄存器	VMCS 的 EPT pointer 字段
访问不存在的页	Page Fault 异常 (#PF)	VM Exit：EPT violation

I/O 虚拟化

- CPU 通过 I/O 操作与各种 device 交互
- I/O 类型
  - PIO：**`IN`/`OUT`** 指令
  - MMIO：访问特殊内存区域

I/O 虚拟化

- 直连(pass-through)：
  - Guest 直接访问设备，无需经过 hypervisor
  - 不支持多个 guest 同时访问
  - IOMMU
- 模拟：
  - Guest 触发 VM exit
    - I/O instruction
    - EPT violation
  - Hypervisor 模拟 I/O 操作
  - 支持多个 guest
  - 实现复杂，速度慢

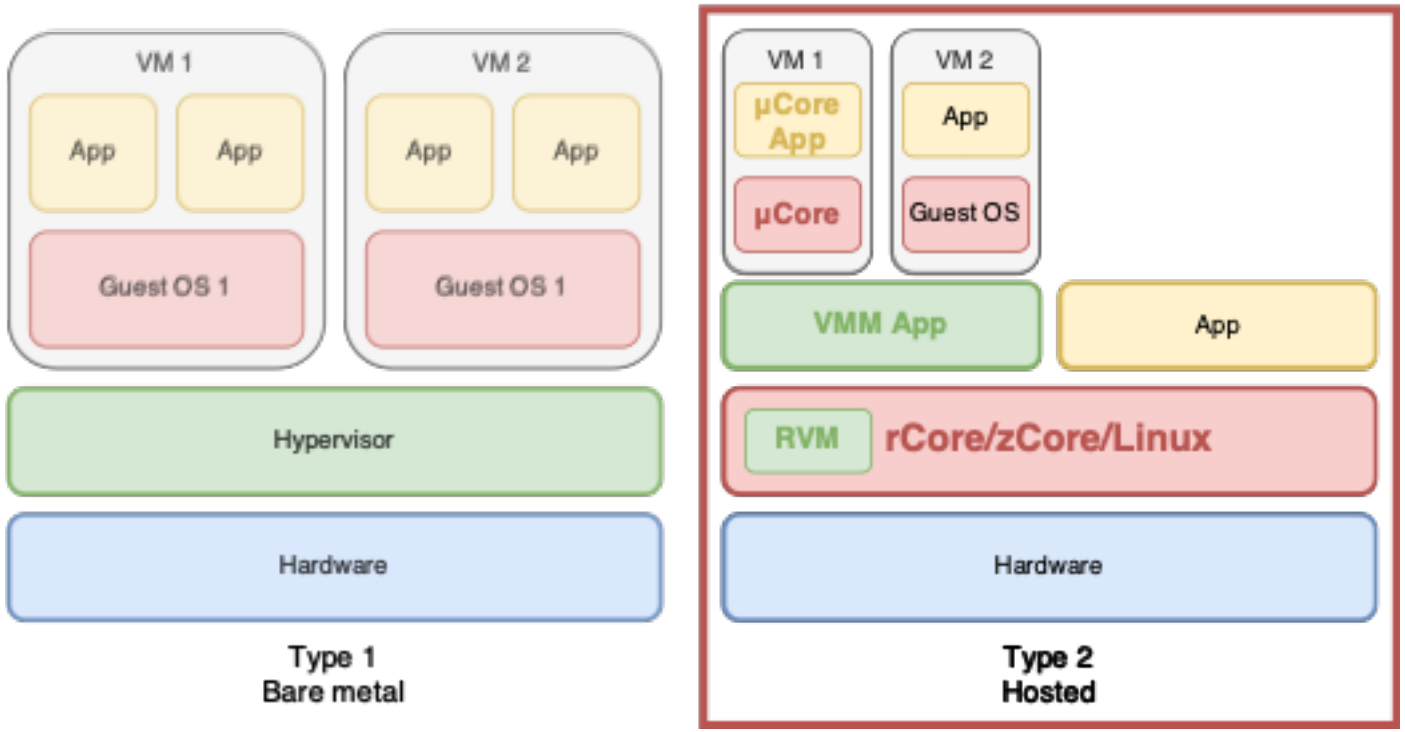
RVM v1

- 参考 Google Fuchsia 微内核 Zircon 的 hypervisor 模块
- Rust crate
  - 提供了对 Guest、vCPU 等结构的抽象
  - 可方便搭建 Type-1、Type-2 的 hypervisor
  - 可集成进其他 Rust 编写的底层系统软件中
- 目前已实现 (Type-2)：
  - 在 rCore 中使用，能运行 Guest uCore
  - 在 zCore 中使用，能通过 Zircon hypervisor 相关测例
  - 在 Linux 中使用，做成一个 Linux Kernel Module，代替 KVM+QEMU 成功运行 Guest uCore

https://github.com/rcore-os/RVM
https://github.com/rcore-os/RVM/wiki

基于 RVM v1 构建 Type 2 hypervisor

- 类似 KVM + QEMU
- 基本组件
  - RVM —— KVM
  - VMM app —— QEMU
  - Host OS: rCore/zCore/Linux
  - Guest OS: uCore



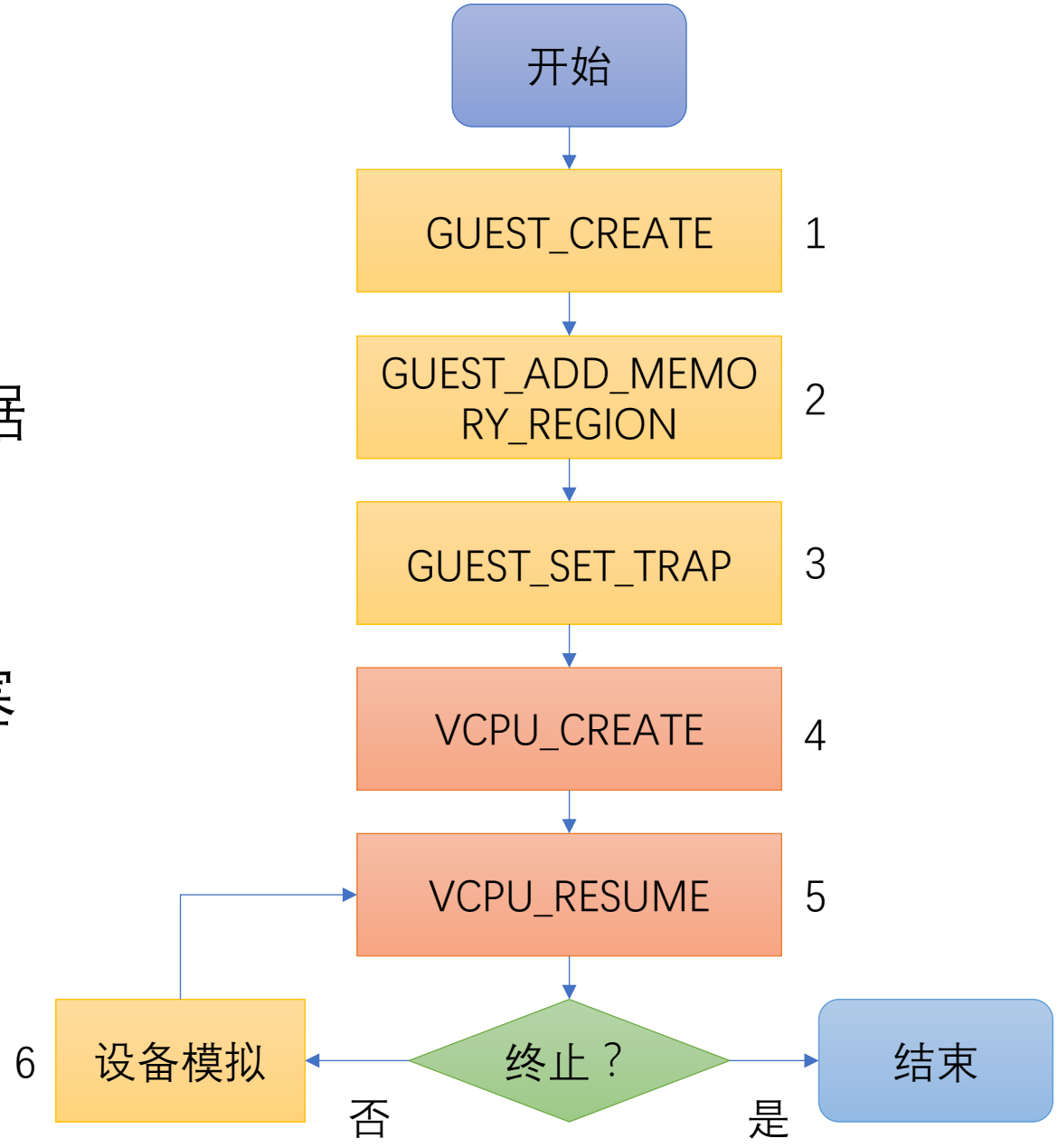
演示：在 rCore 中运行 uCore

RVM v1 API

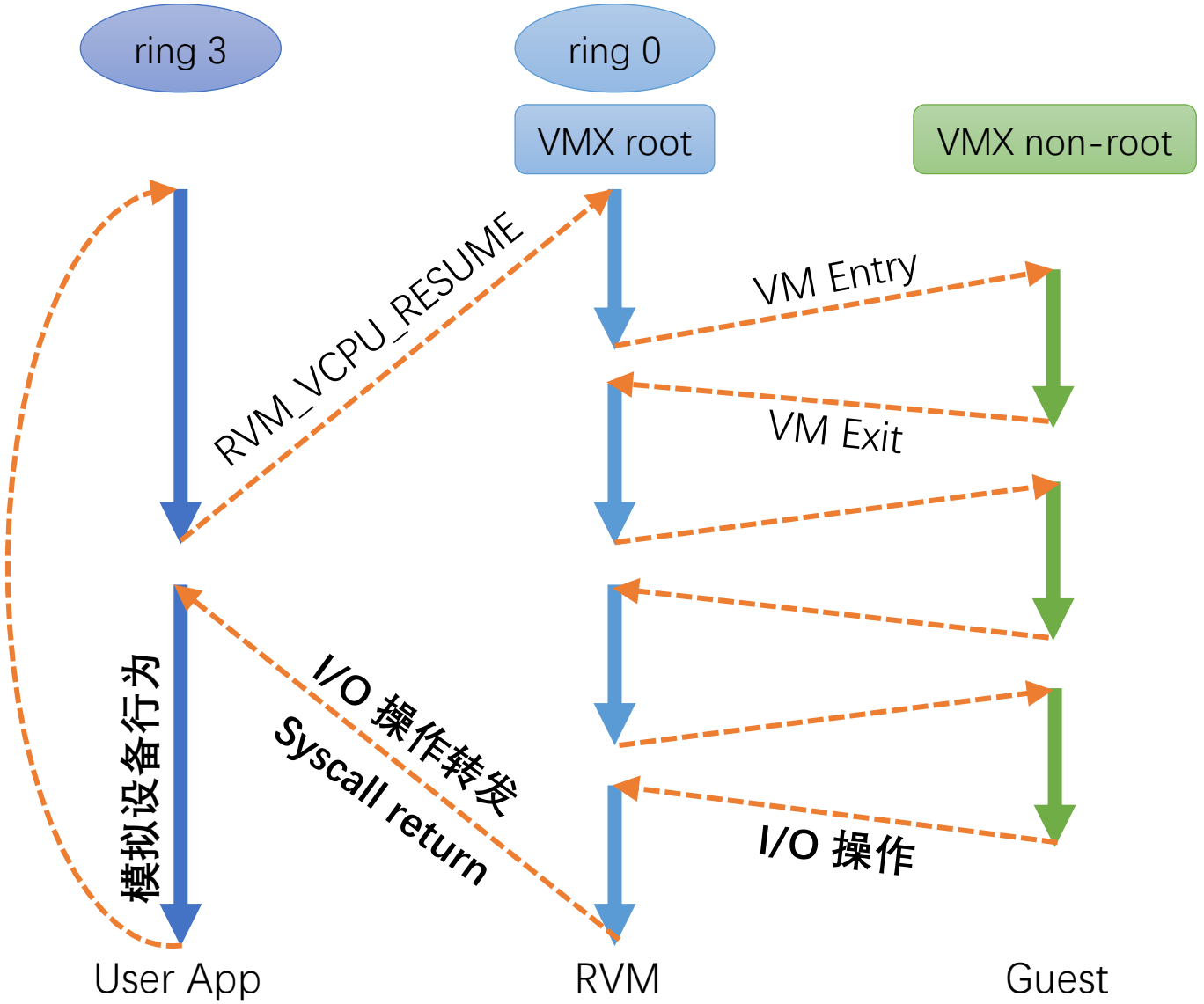
API 原语/rCore ioctl	zCore syscall	RVM API	说明
RVM_GUEST_CREATE	sys_guest_create	Guest::new()	创建一个 Guest
RVM_GUEST_ADD_MEMORY _REGION	sys_vmo_create sys_vmar_map	guest.add_memory_region()	添加 Guest 的物理内存段
RVM_GUEST_SET_TRAP	sys_guest_set_trap	guest.set_trap()	设置捕获哪些 I/O 操作
RVM_VCPU_CREATE	sys_vcpu_create	Vcpu::new()	创建一个 vCPU
RVM_VCPU_RESUME	sys_vcpu_resume	vcpu.resume()	(恢复)运行 vCPU，阻塞当前线程，直到需要用户态模拟
RVM_VCPU_READ_STATE	sys_vcpu_read_state	vcpu.read_state()	读 vCPU 状态(寄存器)
RVM_VCPU_WRITE_STATE	sys_vcpu_write_state	vcpu.write_state() vcpu.write_io_state()	写 vCPU 状态(寄存器和 I/O 指令结果)
RVM_VCPU_INTERRUPT	sys_vcpu_interrupt	vcpu.virtual_interrupt()	让 vCPU 产生一个中断

VMM App

1. 创建一个 guest
2. 添加物理内存段，写入必要的数据
3. 创建和设置相关虚拟设备
4. 创建 vCPU
5. 轮询调用 VCPU_RESUME，阻塞直到需要模拟设备
6. 设备模拟完毕，返回 5

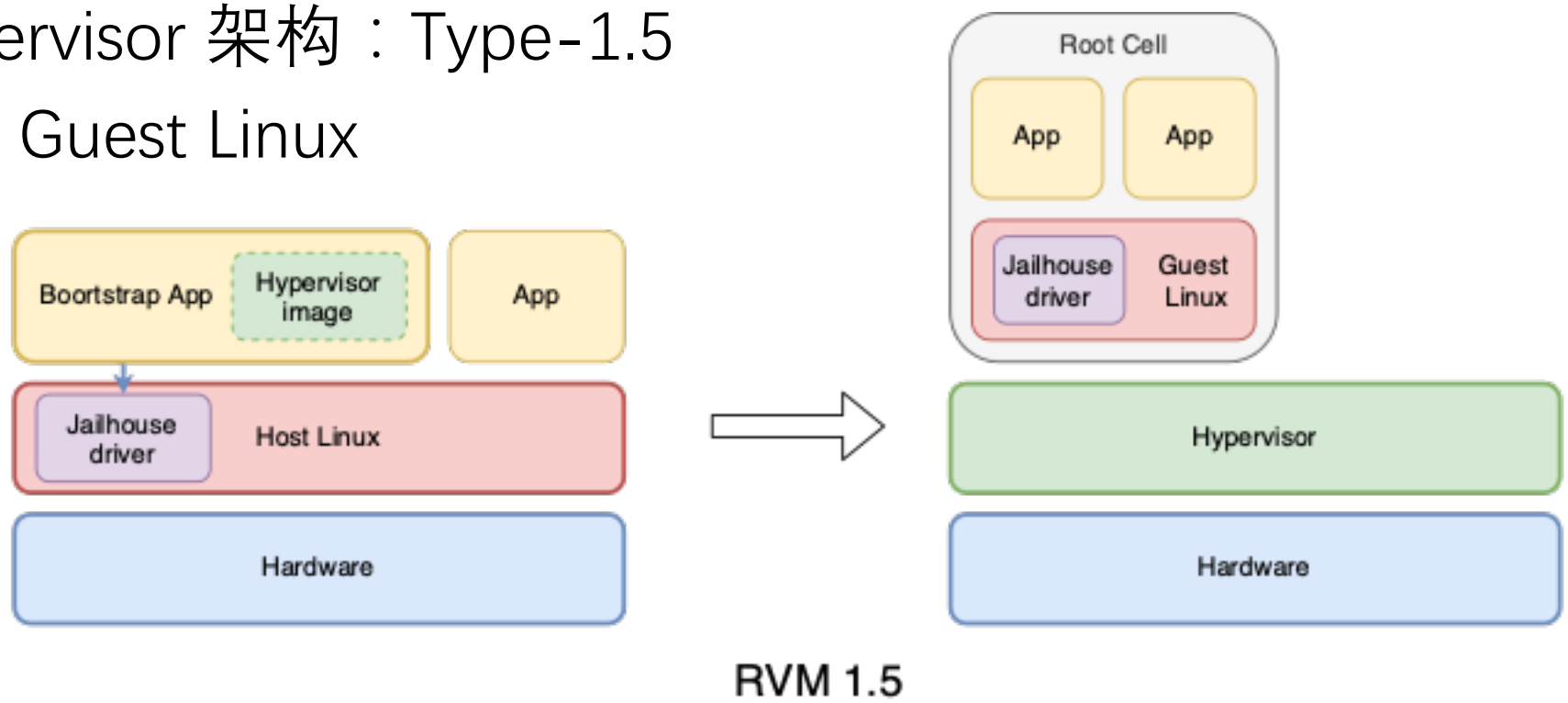


RVM v1 —— 设备模拟


ring 3
ring 0
VMX root
VMX non-root
RVM_VCPU_RESUME
VM Entry
VM Exit
模拟设备行为
I/O 操作转发
Syscall return
I/O 操作
User App
RVM
Guest

RVM v1.5：Type-1.5 hypervisor

- 参考 Jailhouse hypervisor
- 新型 hypervisor 架构：Type-1.5
- 支持运行 Guest Linux



RVM 1.5

https://github.com/rcore-os/RVM1.5

演示：RVM v1.5 启动和关闭 hypervisor

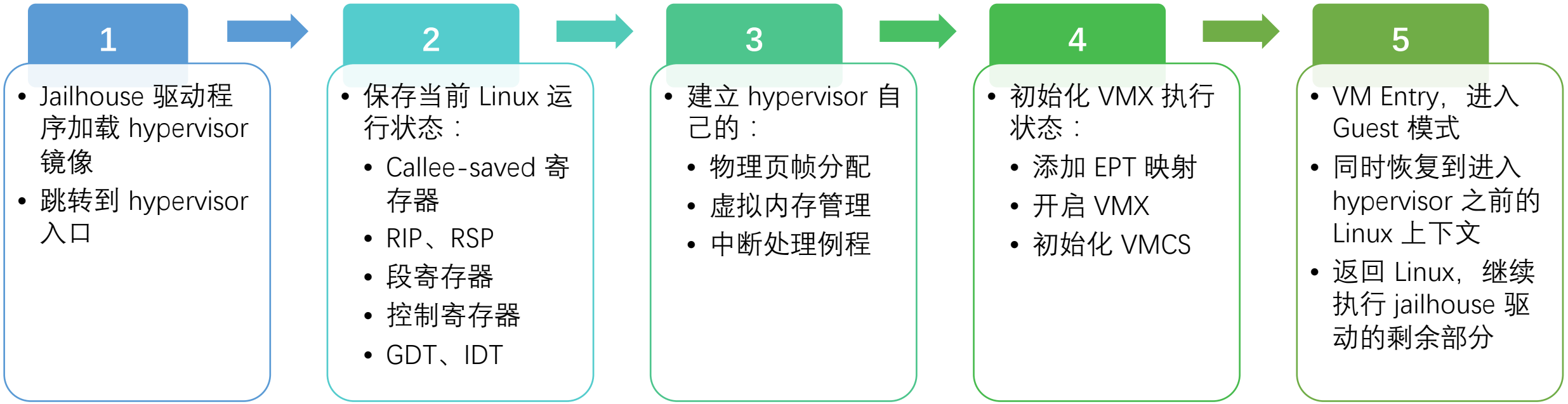
RVM v1.5 —— 特点

- 以 Type 2 的方式加载：
  - 先启动 Linux，再启动 hypervisor
  - 启动后自动将 Linux 降权为 Guest 模式
  - Linux 相当于 bootloader
- 以 Type 1 的方式运行：
  - 自行管理硬件，无需依赖 Linux
- 轻量化：
  - 直通对设备的访问，无需模拟
- 系统管理员被降权：
  - Linux 无法直接访问硬件和其他 VM
  - 只能通过 hypercall 与 hypervisor 交互

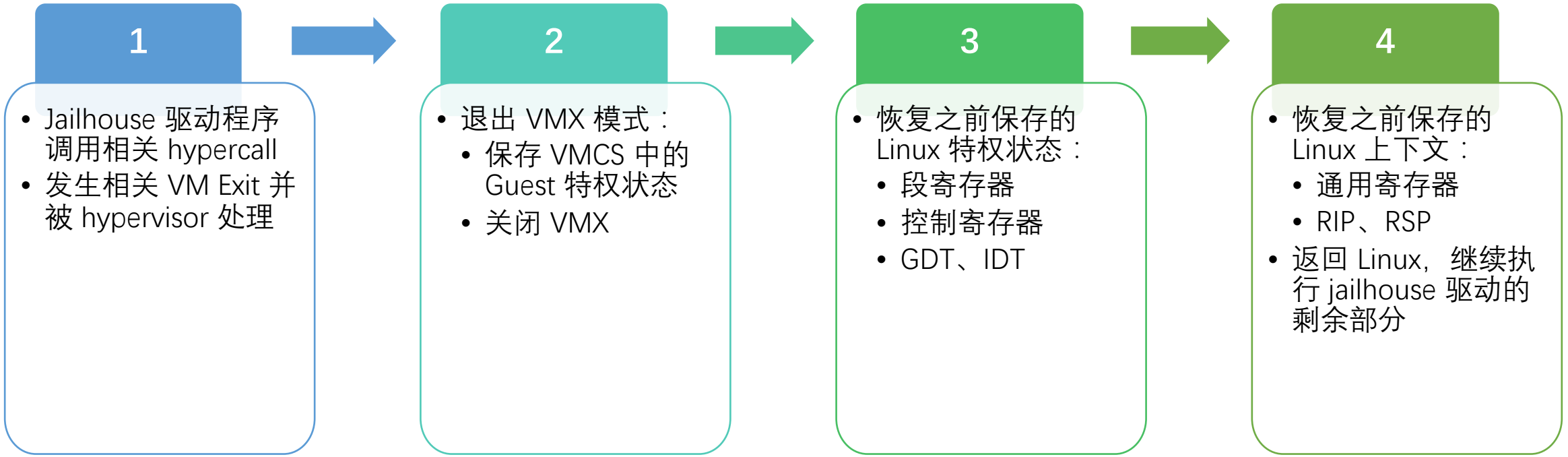
RVM v1.5 —— Hypercall

- 类似于系统调用 (syscall)
- 主动发生 VM Exit，hypervisor 根据传入参数进行相应处理
- 用于 Guest Linux 与 hypervisor 的交互：
  1. 关闭 hypervsior，让 Linux 重新回到 Host 模式
  2. 创建和管理其他 VM

RVM v1.5 —— 启动流程


1
Jailhouse 驱动程序加载 hypervisor 镜像
跳转到 hypervisor 入口
2
保存当前 Linux 运行状态：
Callee-saved 寄存器
RIP、RSP
段寄存器
控制寄存器
GDT、IDT
3
建立 hypervisor 自己的：
物理页帧分配
虚拟内存管理
中断处理例程
4
初始化 VMX 执行状态：
添加 EPT 映射
开启 VMX
初始化 VMCS
5
VM Entry，进入 Guest 模式
同时恢复到进入 hypervisor 之前的 Linux 上下文
返回 Linux，继续执行 jailhouse 驱动的剩余部分

RVM v1.5 —— 关闭流程


1
• Jailhouse 驱动程序调用相关 hypercall
• 发生相关 VM Exit 并被 hypervisor 处理
2
• 退出 VMX 模式：
  • 保存 VMCS 中的 Guest 特权状态
  • 关闭 VMX
3
• 恢复之前保存的 Linux 特权状态：
  • 段寄存器
  • 控制寄存器
  • GDT、IDT
4
• 恢复之前保存的 Linux 上下文：
  • 通用寄存器
  • RIP、RSP
• 返回 Linux，继续执行 jailhouse 驱动的剩余部分

RVM v1 与 RVM v1.5 实现对比

	RVM v1	RVM v1.5
类型	Type-2	Type-1.5
启动方式	在 Host OS 里启动	在 Host OS 里启动
硬件资源管理	通过 Host OS 提供的 API	自行实现
运行依赖	Host OS 提供的部分服务	无
支持的 Host OS	rCore/zCore/Linux	Linux
支持的 Guest OS	uCore	Linux (从 Host Linux 自动降权)
对 Guest I/O 的处理	模拟	直通
对 Guest 中断的处理	部分转发给 Guest	直通

Thanks