Embedded OS in China
国产嵌入式操作系统产业论坛

Rust-Shyper：基于Rust语言的高可靠、嵌入式Hypervisor

王雷

北京航空航天大学计算机学院

目 录

CONTENTS

研究背景及意义

常见的虚拟化方案

Xen Project

StratoVirt

QEMU

Linux KVM

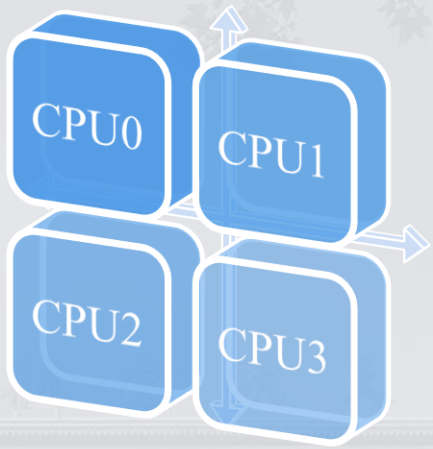
Xvisor
eXtensible Versatile hypervISOR

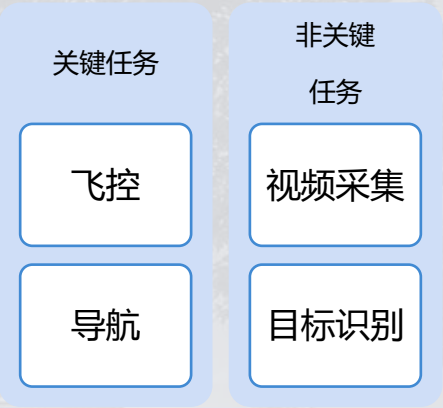
ACRN

- Jailhouse Hypervisor
- Bao Hypervisor
- Etc.

主要面向的场景

- 车载、智能座舱、无人机等混合关键系统嵌入式虚拟化





专用->通用

图形化用户界面、传感器数据采集、人机交互、路径规划等

单核->多核

硬件处理器包含多个核心从而可以同时运行多个任务

简单一致->混合关键性

需要同时运行和管理关键任务和非关键任务

要求和挑战

轻量

- 最小化可信计算基，增加系统的可靠性和安全性

高性能高实时

- 嵌入式对实时性能敏感，需要保障上层VM的实时性，并尽量减少VM-Exit

要求和挑战

- 各个VM的硬件资源、故障等的隔离，不能相互影响

隔离性

- 最小化系统资源和性能的开销

效率

常见虚拟化方案

- **多分区实时操作系统**
  - LynxOS、INTEGRITY、VxWorks
  - **满足 ARINC 653 等时间域和空间域隔离设计的软件标准**
  - **依赖于特定的RTOS；多平台移植能力差；闭源且收费昂贵**

✈基于服务器的虚拟化技术

- Xen、KVM
- 在桌面、服务器和云计算等领域获得巨大的成功
- 可信计算基较大，难以通过 ARINC 653 等工业安全认证
- 无法保证关键任务满足必要的实时性约束
- 使用传统C语言编程实现

✈新型嵌入式虚拟化技术

- Xvisor、XtratuM、Jailhouse、ACRN
- 拥有较小的可信计算基
- 通过资源固定分配的策略以及实时虚拟化技术保证关键任务的实时性约束
- 资源利用率较低、未能完全解决实时性虚拟化的问题
- 使用传统C语言编程实现

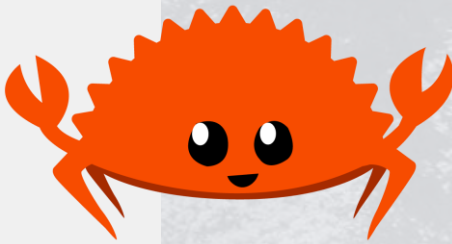
系统软件

- 系统软件：
  - 操作系统内核（rCore）
  - Hypervisor
  - 网络协议栈
  - ……
- 需求：
  - **性能；**
  - **安全性；**

Rust的优势

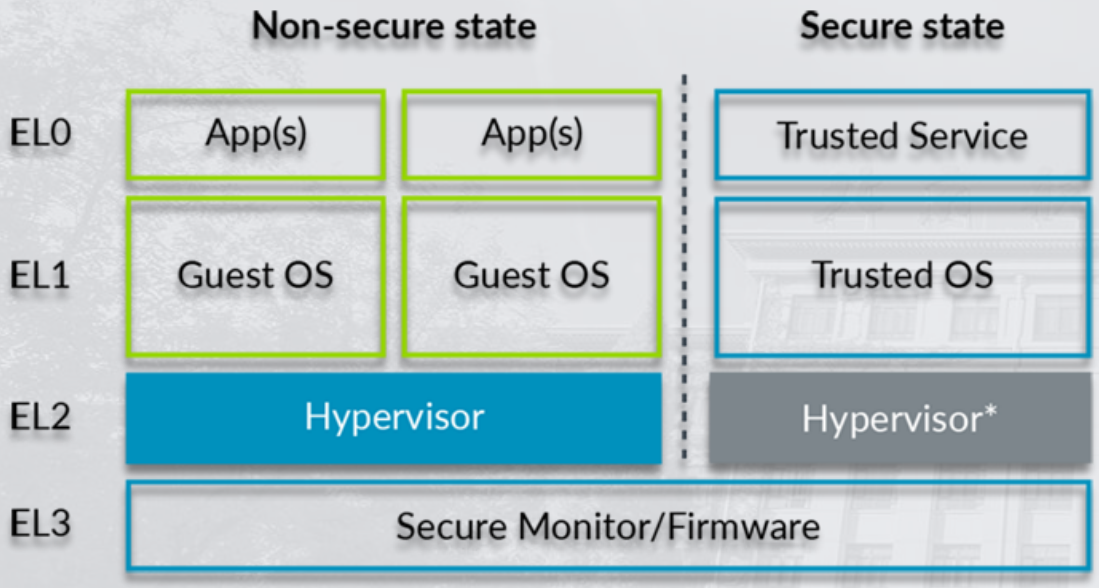
高安全；高性能；社区生态；很好用的包管理；良好的FFI兼容和支持；

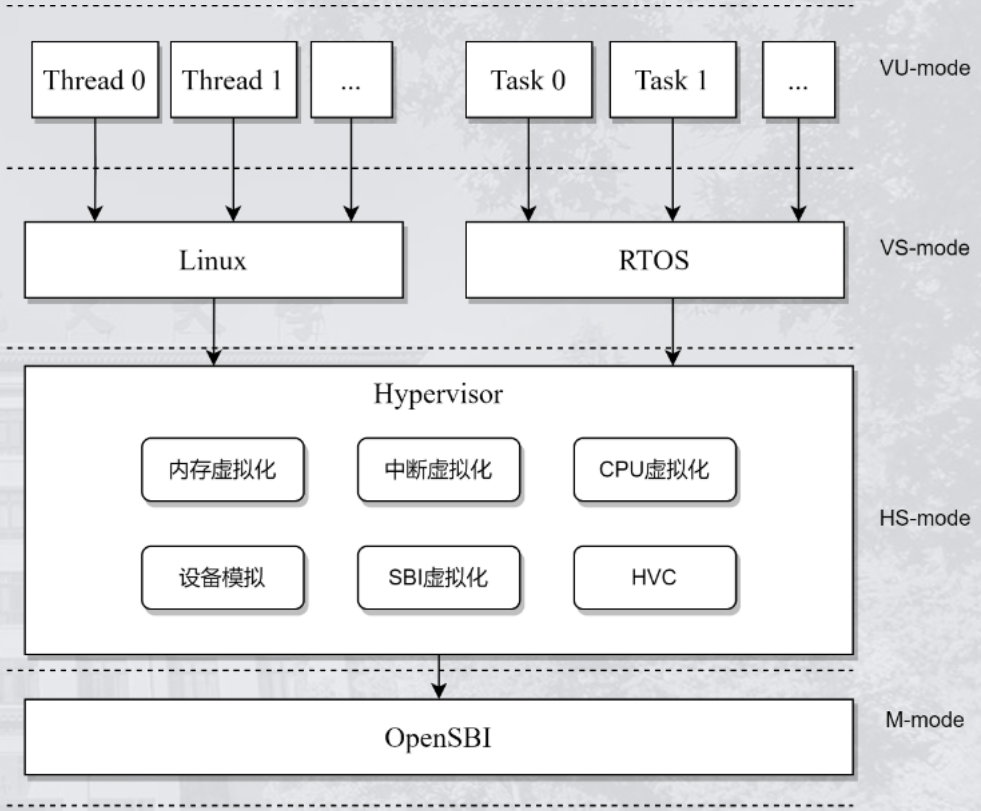
- **内存安全**：
  - 解决空指针、裸指针等；
  - 借用检查、生命周期概念等；
- **线程安全**：
  - 内置安全并发模型；
  - 语言级的多线程同步原言；
- **性能**：
  - 零成本抽象；
  - 泛型与静态分派；
  - 确定性内存管理，无GC；
- **其他现代特性：**包管理、函数式等



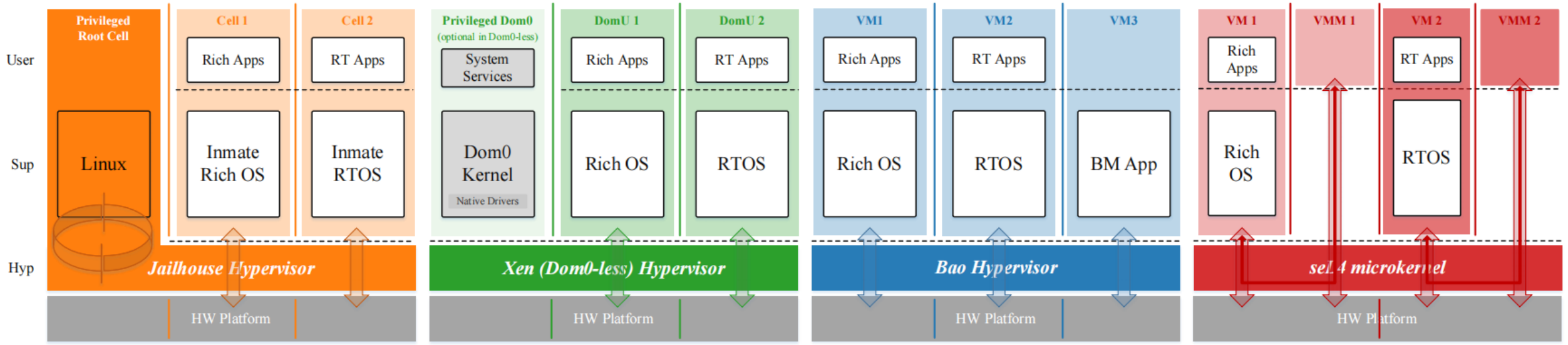
ARM&RISC-V平台系统设计

对虚拟化的支持


Non-secure state
Secure state
EL0
App(s)
App(s)
Trusted Service
EL1
Guest OS
Guest OS
Trusted OS
EL2
Hypervisor
Hypervisor*
EL3
Secure Monitor/Firmware

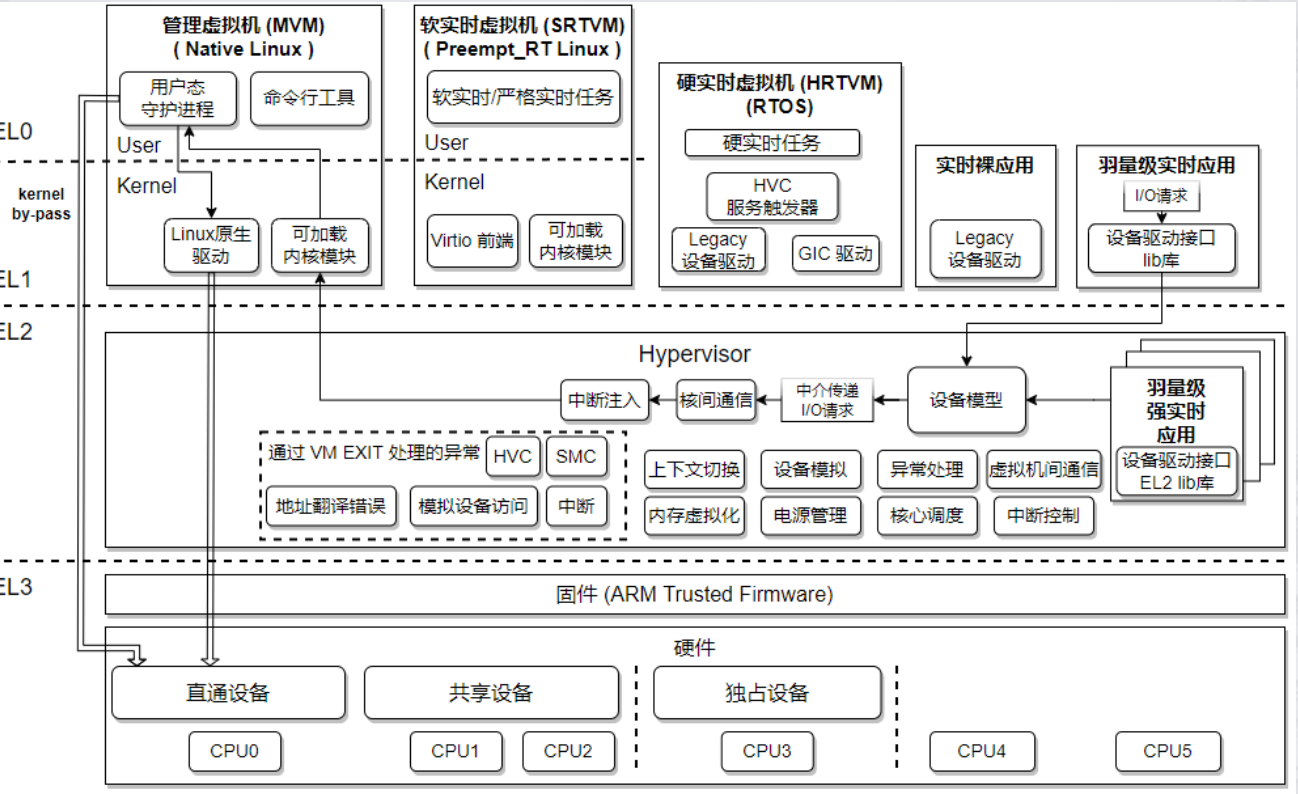

Thread 0
Thread 1
...
Task 0
Task 1
...
VU-mode
Linux
RTOS
VS-mode
Hypervisor
内存虚拟化
中断虚拟化
CPU虚拟化
设备模拟
SBI虚拟化
HVC
HS-mode
OpenSBI
M-mode

常见的虚拟化方案



arXiv:2303.11186v2 [cs.OS] 23 Mar 2023

系统架构设计



Rust-Shyper特点

- 基于Rust语言的Type I虚拟机监控器，具备更好的隔离性和更轻量的代码实现。基于AArch64&RISC-V、能够支持Linux、裸应用等系统软件的运行
- 灵活、动态创建并配置虚拟机，能够为实时操作系统创建实时虚拟机，保障嵌入式场景的实时需求
- 具备资源隔离属性，保障不同虚拟机间的隔离性
- 实现多种模拟设备，提供基本的磁盘、网络、串口等设备的虚拟化实现，提高资源利用率
- 提供传统虚拟机迁移机制以及本地热更新机制，保障虚拟机监控器正常运行过程中的可靠性

Rust-Shyper目前支持的平台

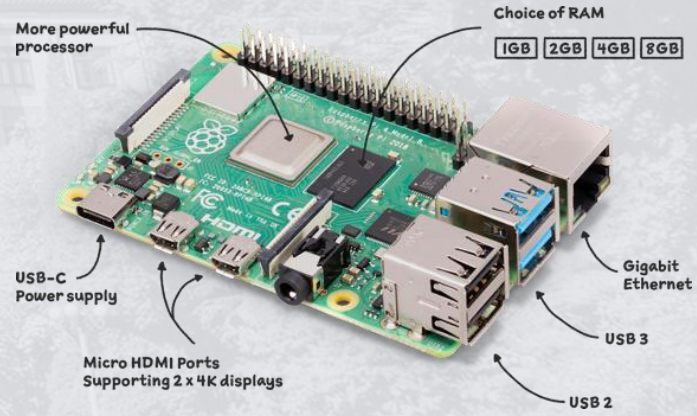
- Firefly ROC-RK3588S-PC

- QEMU (for RISCV64)

RISC-V®

- NVIDIA Jetson TX2

- Raspberry Pi 4 Model B



资源隔离策略

Rust-Shyper资源隔离策略

01 空间域隔离：

空间域隔离是保证VM访问仅限于分配给自身资源的的重要特性。

- 处理器核心隔离
- 地址空间隔离

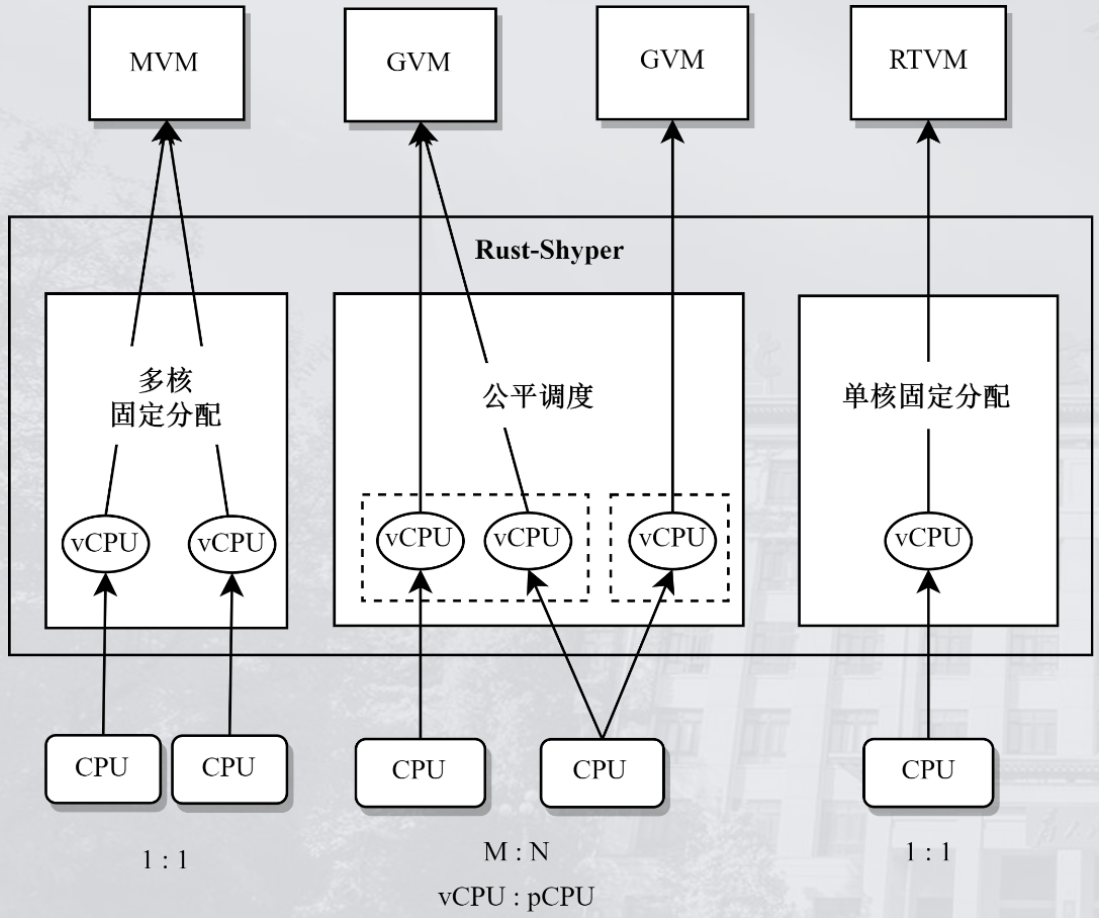
02 时间域隔离：

时间域隔离用于确保不同关键性的VM能够获得正确的CPU指令周期分配。

- 中断隔离
- I/O设备隔离
- 网络隔离

空间域隔离

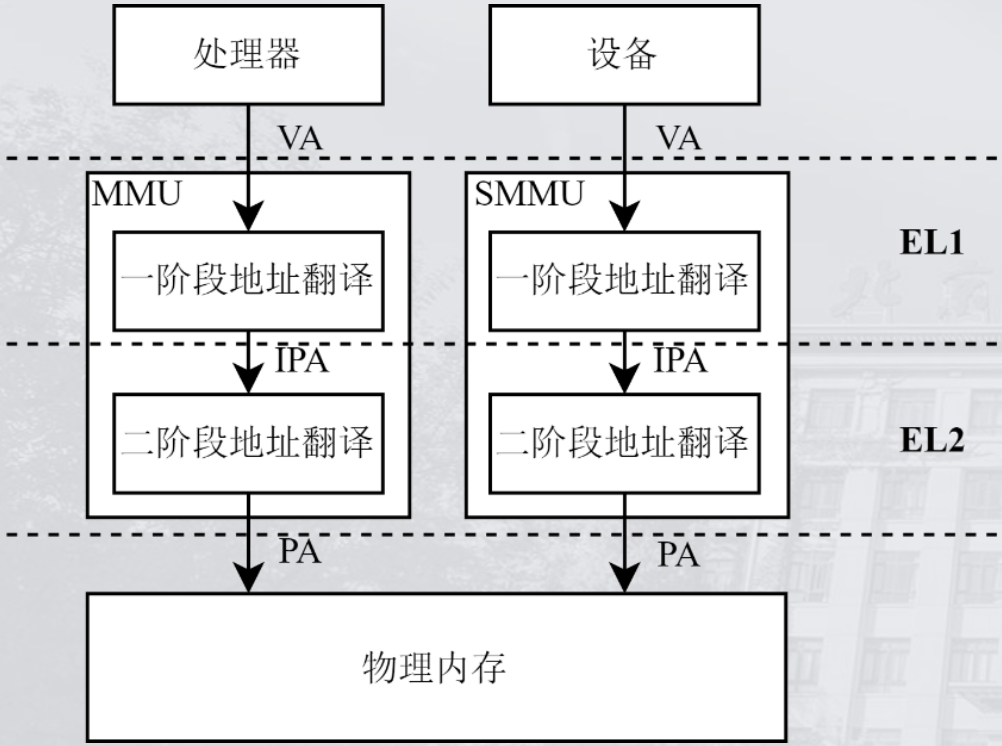
- 处理器核心隔离



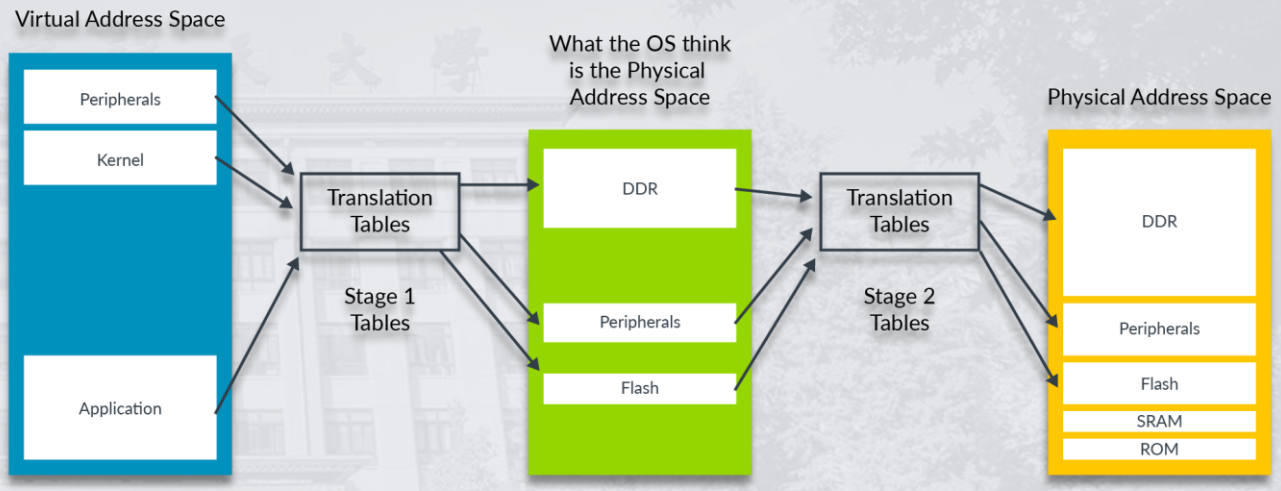
- 核心隔离需要保障虚拟机的vCPU运行状态不应受到其他虚拟机的篡改
- 不同物理核心上vCPU的隔离与同一物理核心上vCPU的隔离

空间域隔离

- 地址空间隔离



- Rust-Shyper通过二阶段地址映射提供的内存隔离来确保VM间内存数据无法被其他VM获取和修改



空间域隔离

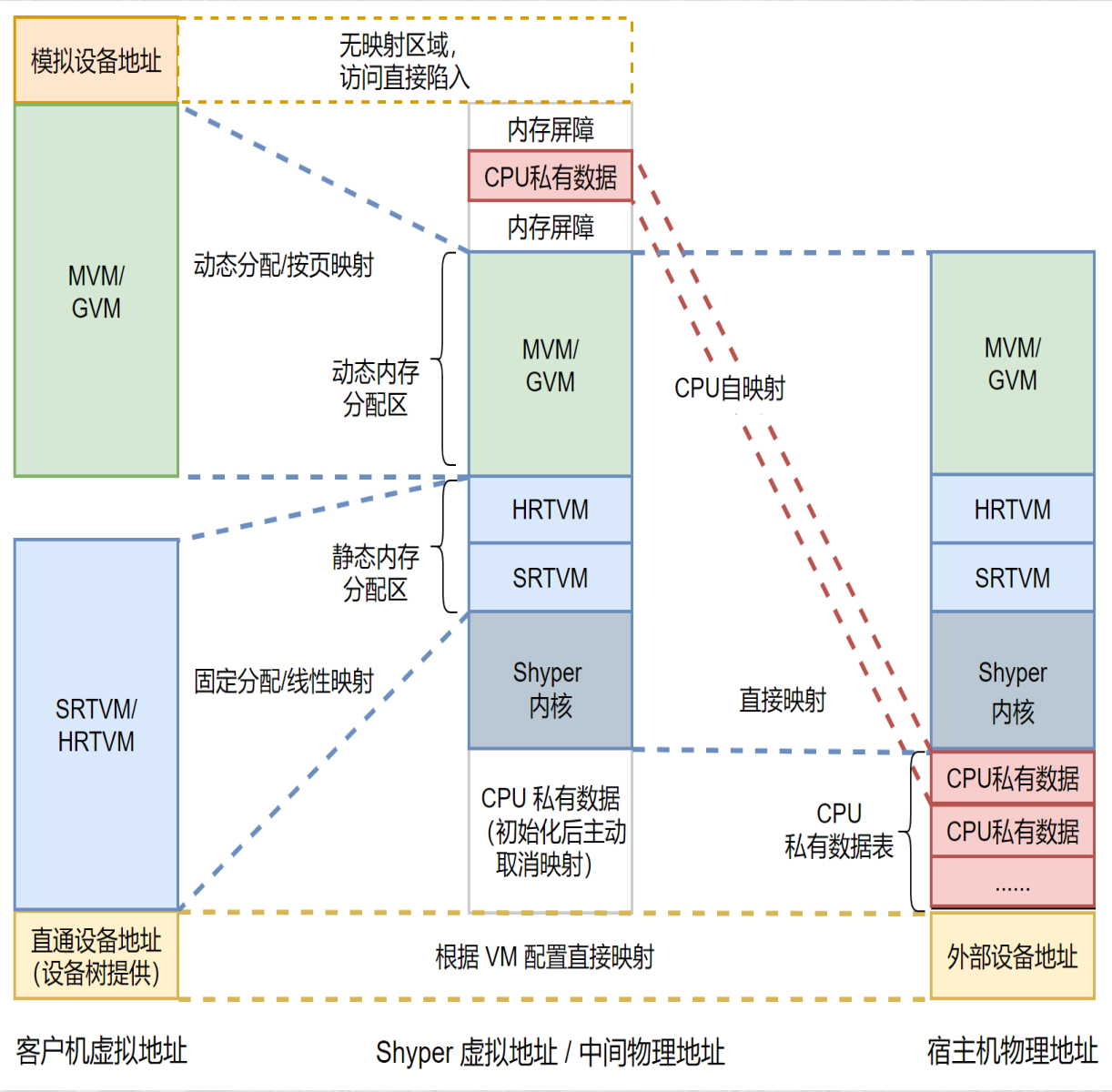
- 地址空间隔离

➢ 对于VM：

➢ 地址空间直接映射

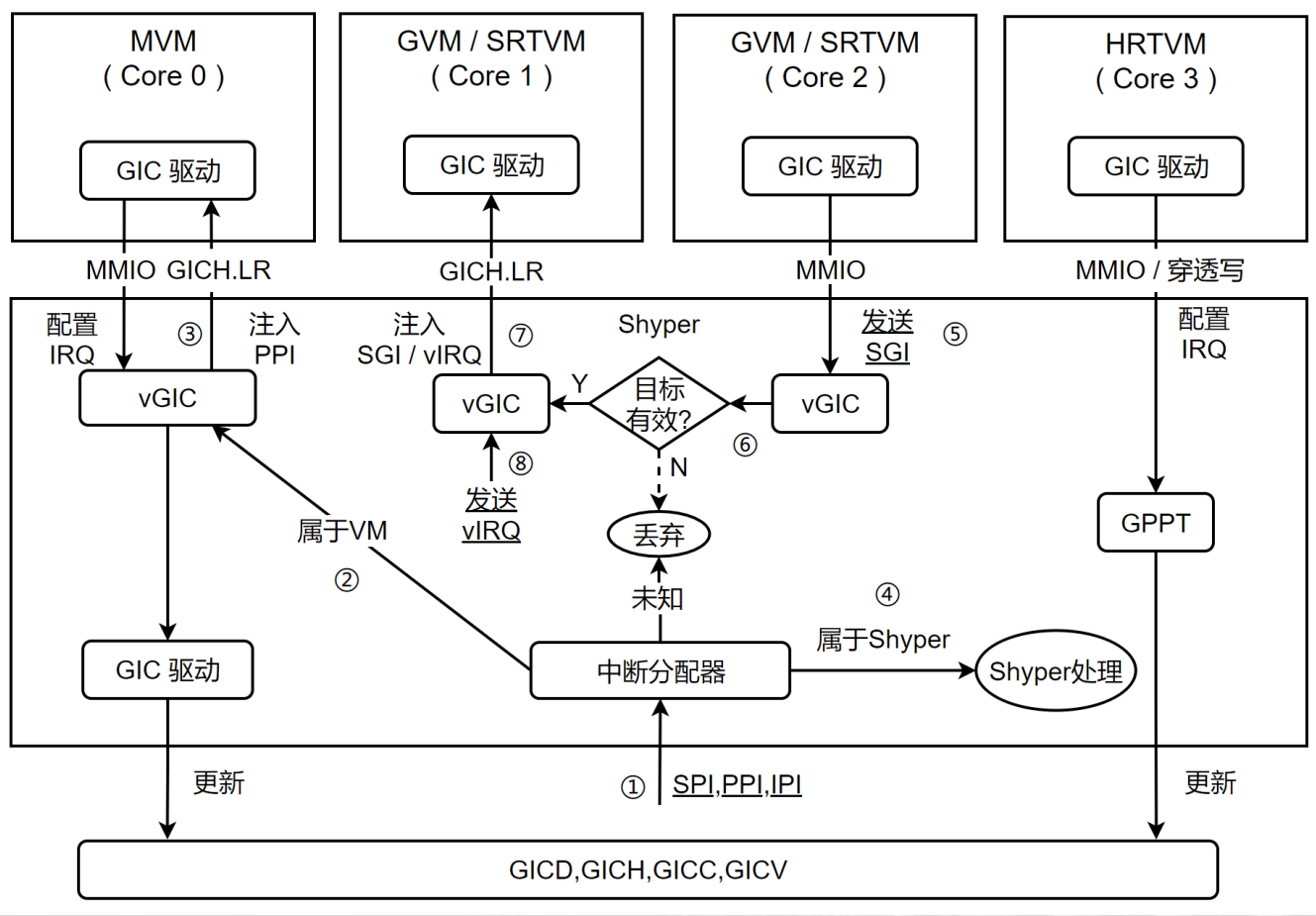
➢ VM启动前预先建立内存和设备地址的二阶段映射页表

➢ 对每一个CPU核心都分配独立的内核堆空间



时间域隔离

- 中断隔离

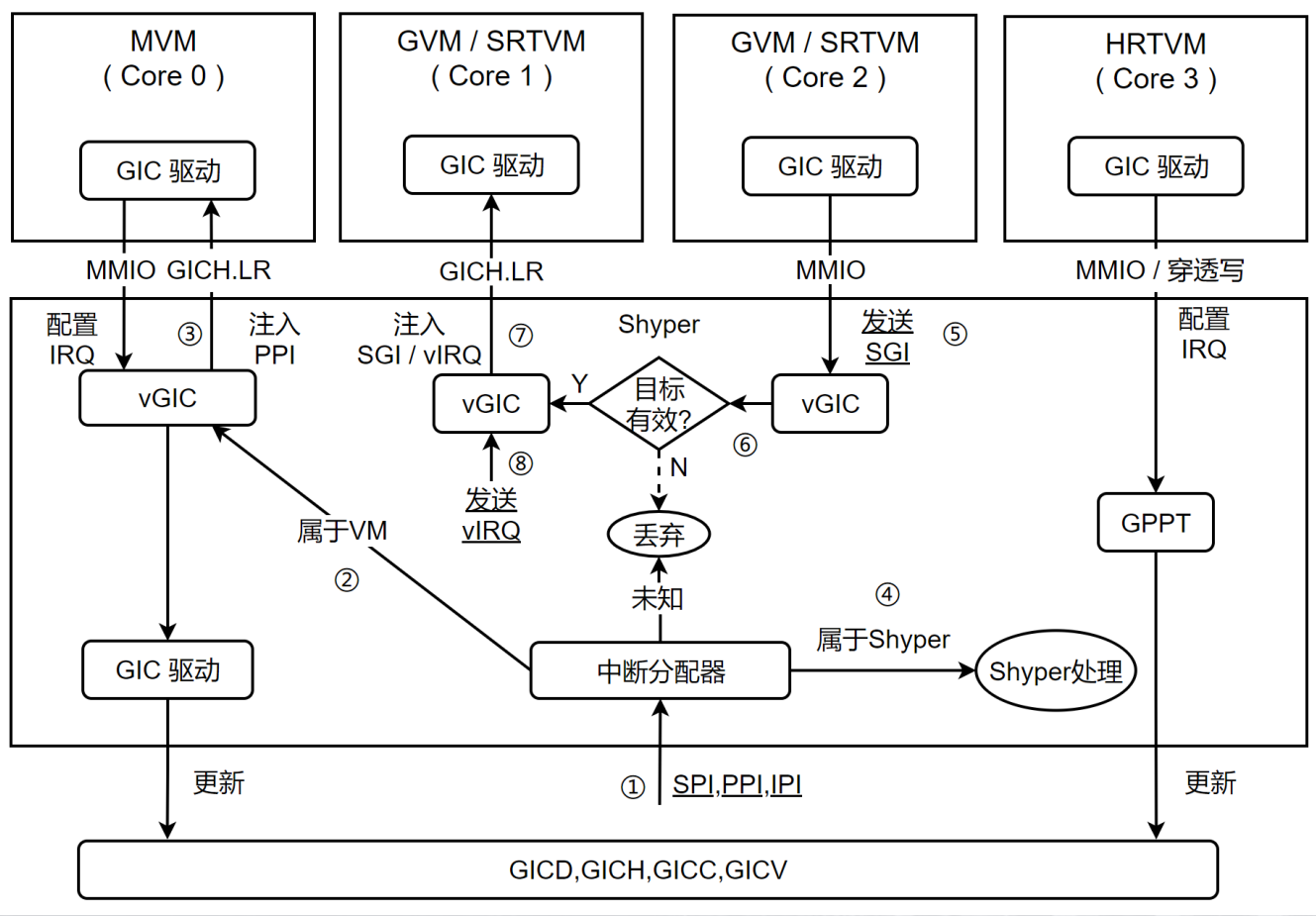


- 保证每一个虚拟机以及虚拟机监控器可以收到正确的中断
- 保证虚拟机之间无法通过发送软中断干扰运行
- 保证虚拟机的中断配置无法被其他虚拟机修改

SPI：共享外设中断
PPI：私有设备中断
SGI：软件产生中断
vIRQ：hypervisor产生的虚拟中断

时间域隔离

- 中断隔离

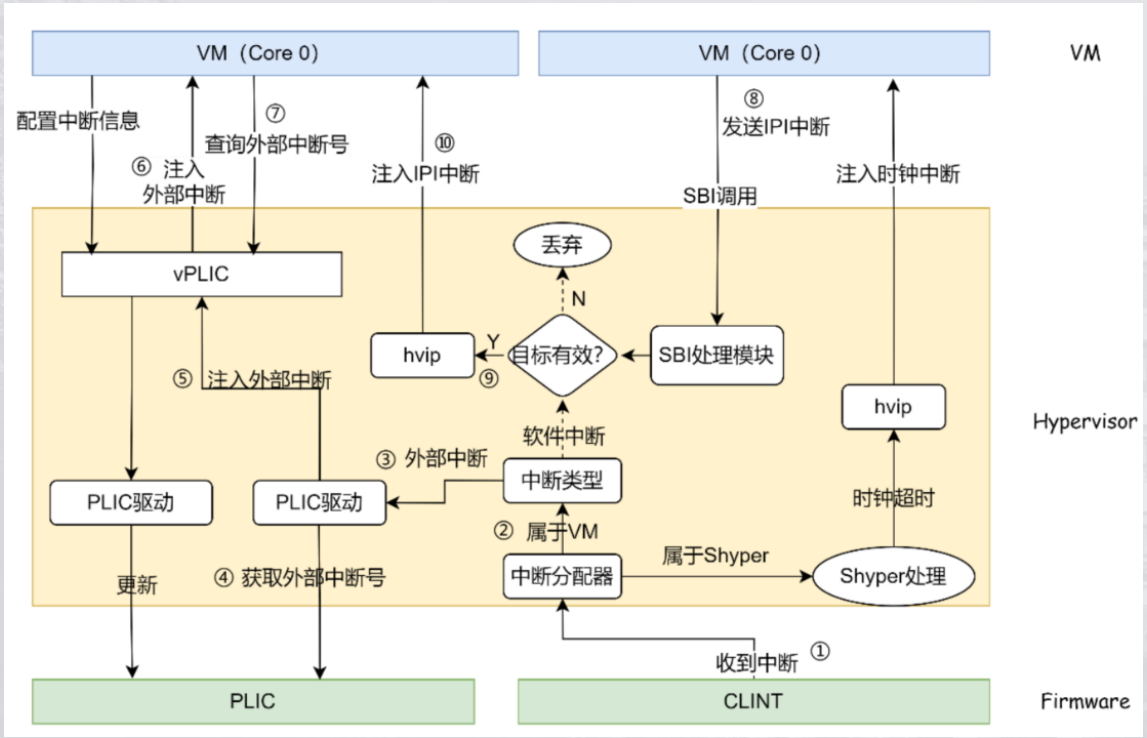


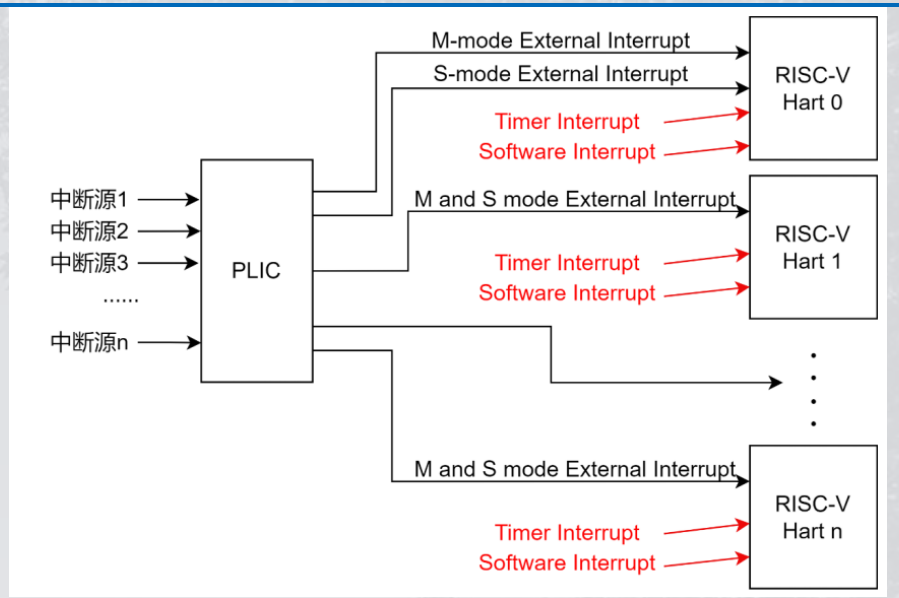
外部硬件中断：

- Rust-Shyper在EL2特权级拦截所有中断
- 在Step1中判断外部中断的归属
- 如果是上层VM的中断，通过Step2~3，将对应中断注入至对应的VM
- 如果是Shyper自身中断，通过Step4处理对应中断事件
- 否则，抛弃该中断

RISC-V核心级中断

- 时钟中断
- 核间中断
- 外部中断（由PLIC管理）





模拟PLIC

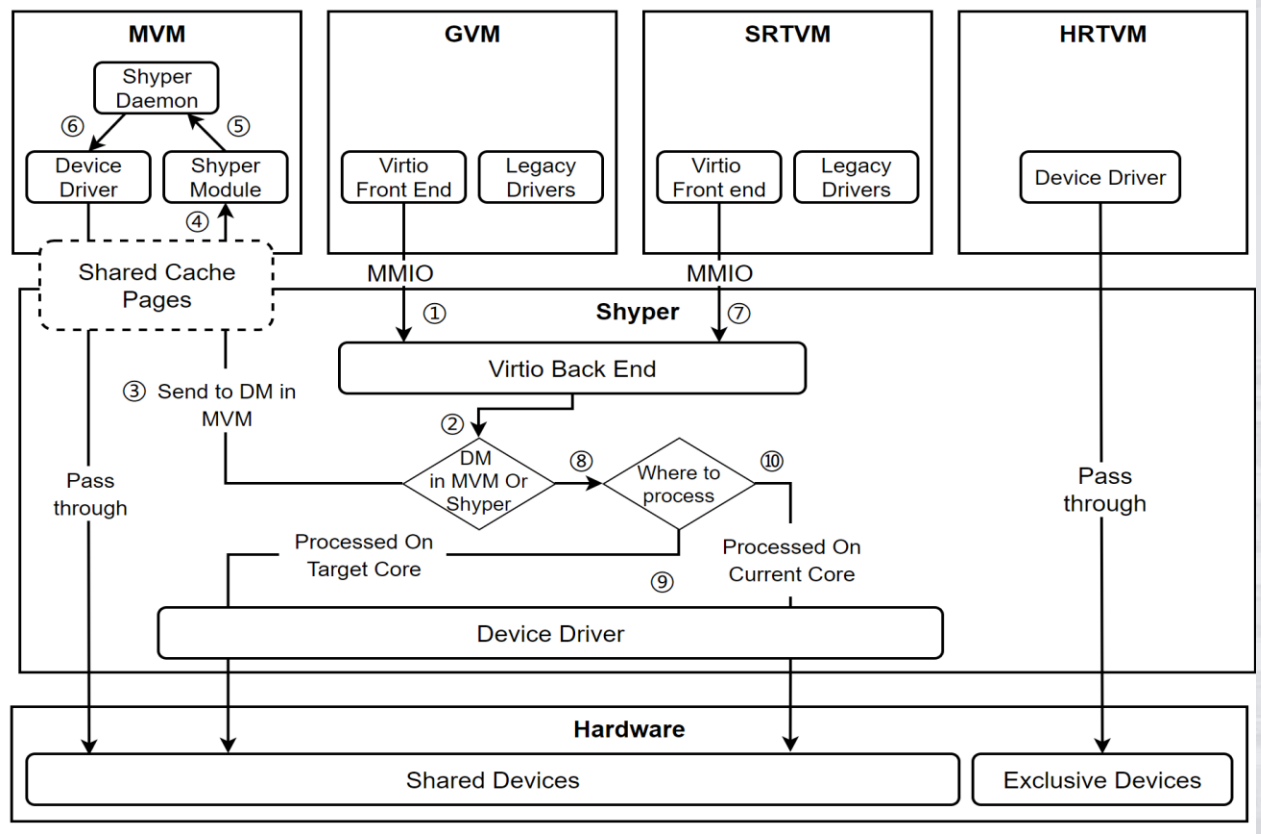
- 配置中断信息：更新到真实PLIC
- 发生外部中断时：注入外部中断
- Claim：返回当前VM的活跃外部中断号

核间中断

- 通过SBI调用发送
- Hypervisor检查发送对象的正确性

时间域隔离

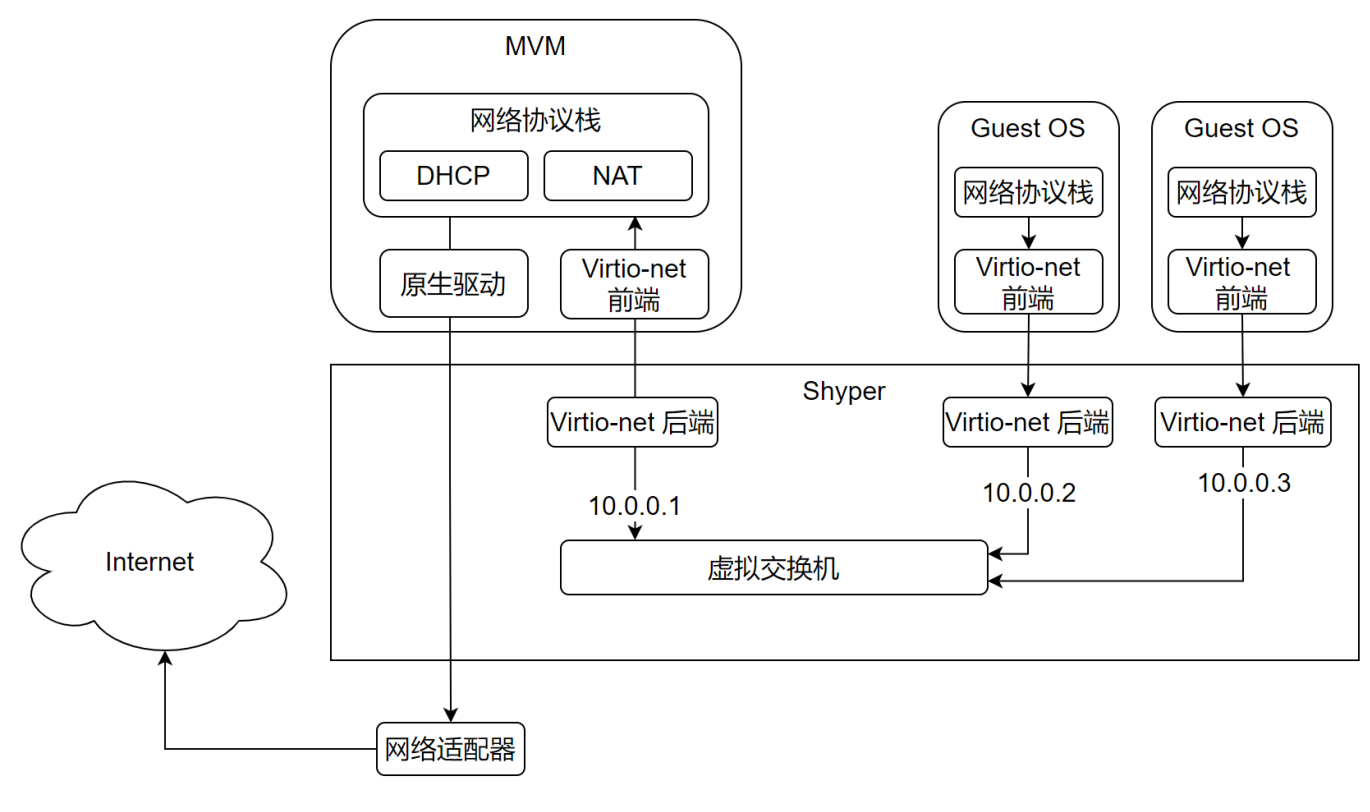
- I/O设备隔离



- 保证虚拟机无法通过伪造I/O请求获取不属于自身的数据或服务
- 设备直通
  - 为指定vm建立设备地址空间的二阶段地址翻译
- 中介传递
  - Rust_shyper从请求受理和中断注入的两个层面进行校验

时间域隔离

- 网络隔离



- 保证虚拟机之间网络服务不被相互访问和篡改
- 利用MVM的网络协议栈及原生网卡驱动为其他VM提供网络服务

设备虚拟化设计

1、中断虚拟化—全模拟设备

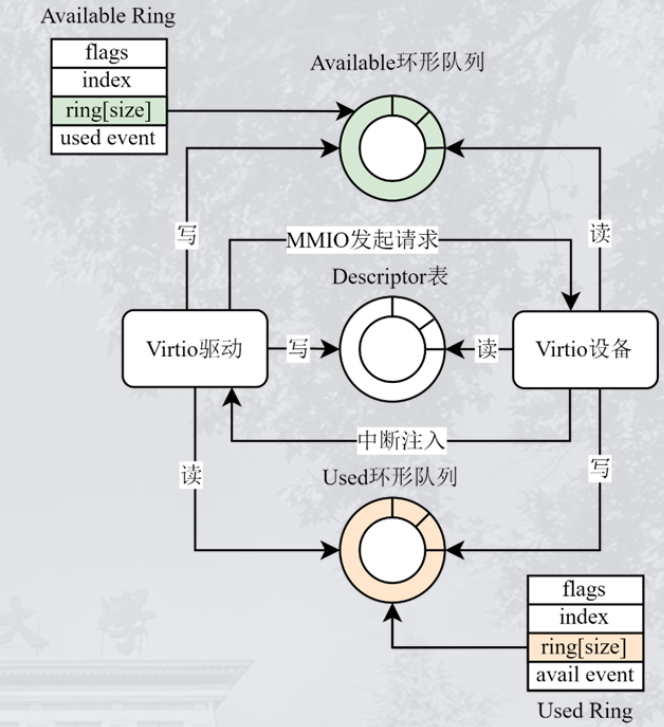
- Aarch64虚拟化扩展，把GICV直通给操作系统作为GICC
- Hypervisor模拟实现VGIC，作为GICD提供给操作系统

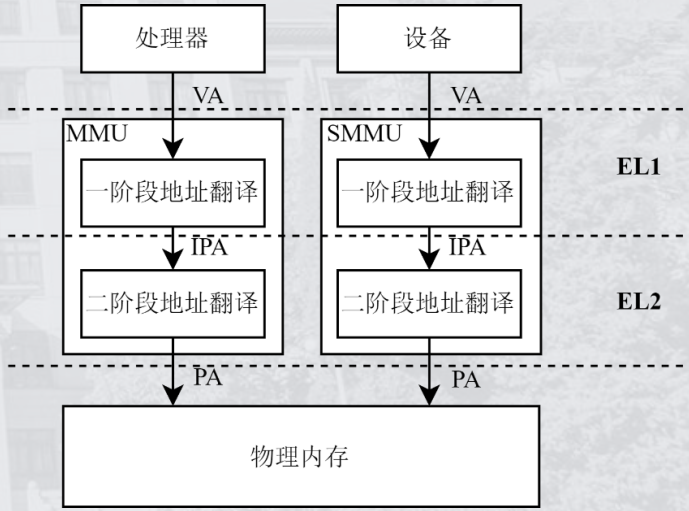
2、设备虚拟化—半模拟设备

- Virtio设备：广泛使用的半虚拟化协议框架；
- 支持多种类型的设备模拟，拥有良好的扩展性；

3、直通设备

- 需实现对应平台的IOMMU来控制DMA设备的访存行为；
- ARM平台对应的是SMMU；



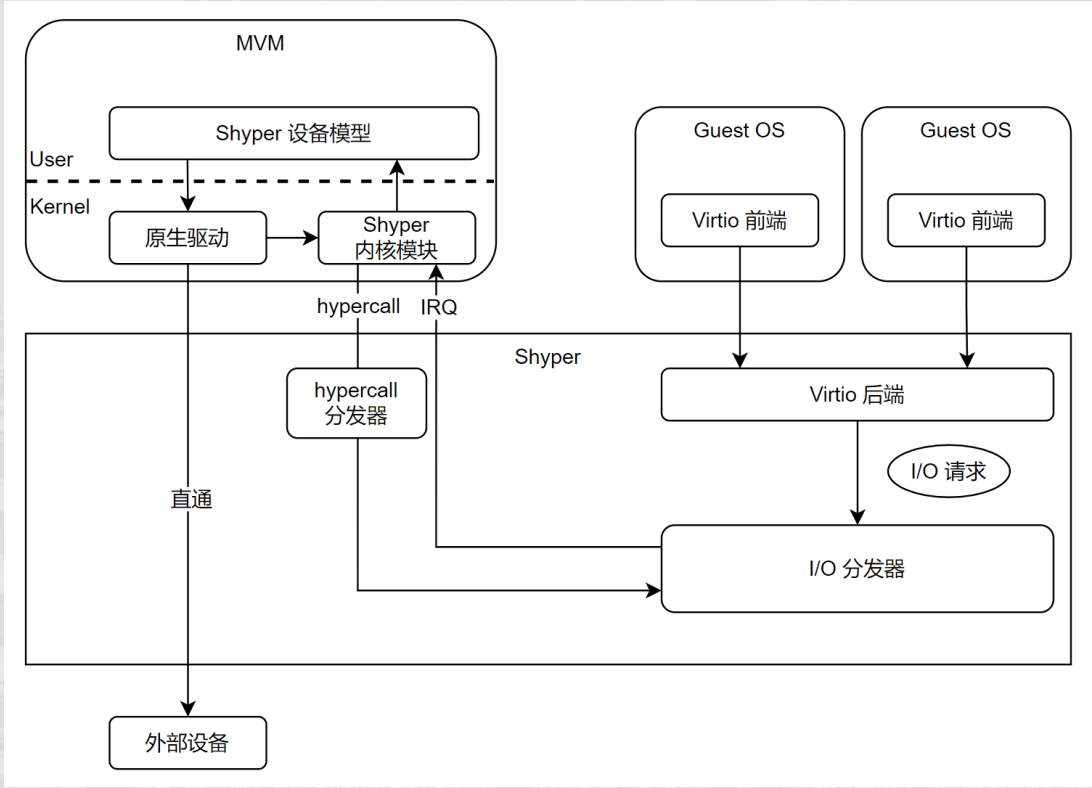


设备虚拟化

设备名称	设备虚拟化方法	虚拟机
中断控制器	直通	RTVM
	全模拟	MVM/GVM
串口	直通	MVM/RTVM
	Virtio	GVM
磁盘	直通	MVM
	Virtio	MVM/GVM
	中介传递	GVM/RTVM
网卡	直通	MVM
	Virtio	GVM/RTVM

基本流程

- 如图所示，基于offload思想的设备驱动虚拟化流程由Hypervisor与MVM二者协同完成
- Hypervisor仅负责I/O请求的解析与传递，而主要的磁盘读写操作由MVM中的Linux原生驱动完成
- 整个流程通过Hypervisor内部的异常处理函数、Virtqueue解析函数、I/O分配器、虚拟中断注入、hypercall分配器，MVM内部的内核模块与用户态设备模型守护进程共同参与完成
- 由于Linux并不推荐在内核态进行文件读写操作，故需要用户态的守护进程使用C语言库函数fread、fwrite等来完成对磁盘文件的读写，这也能够为磁盘I/O提供一个原生的缓存机制

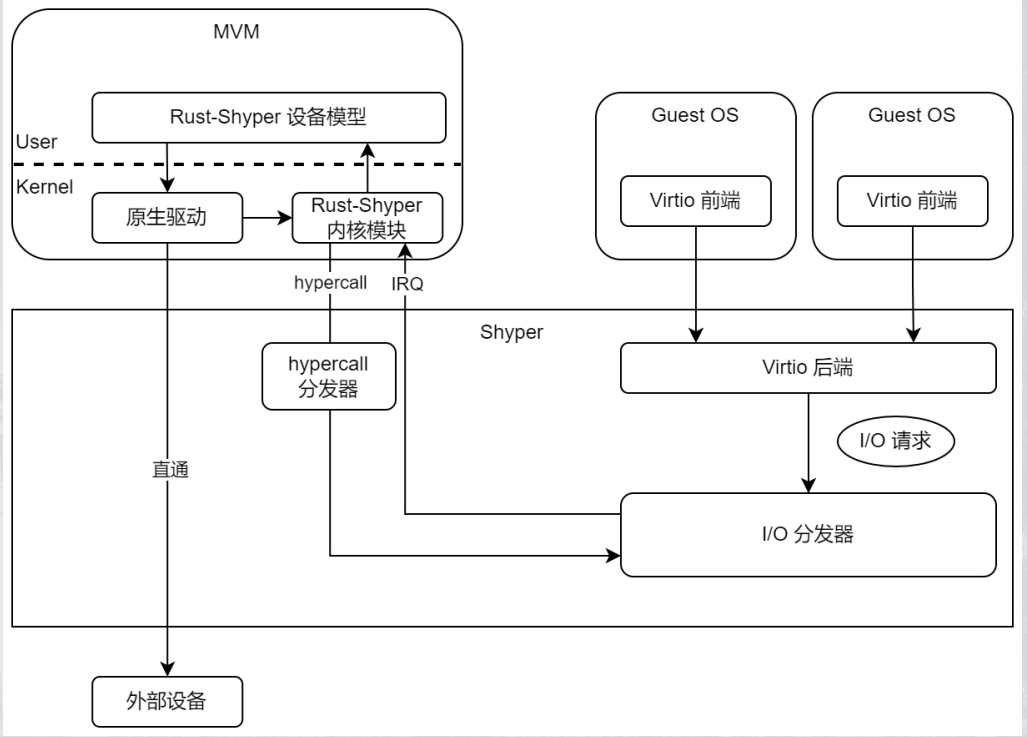


Rust-Shyper实时性优化

什么是中介传递设备

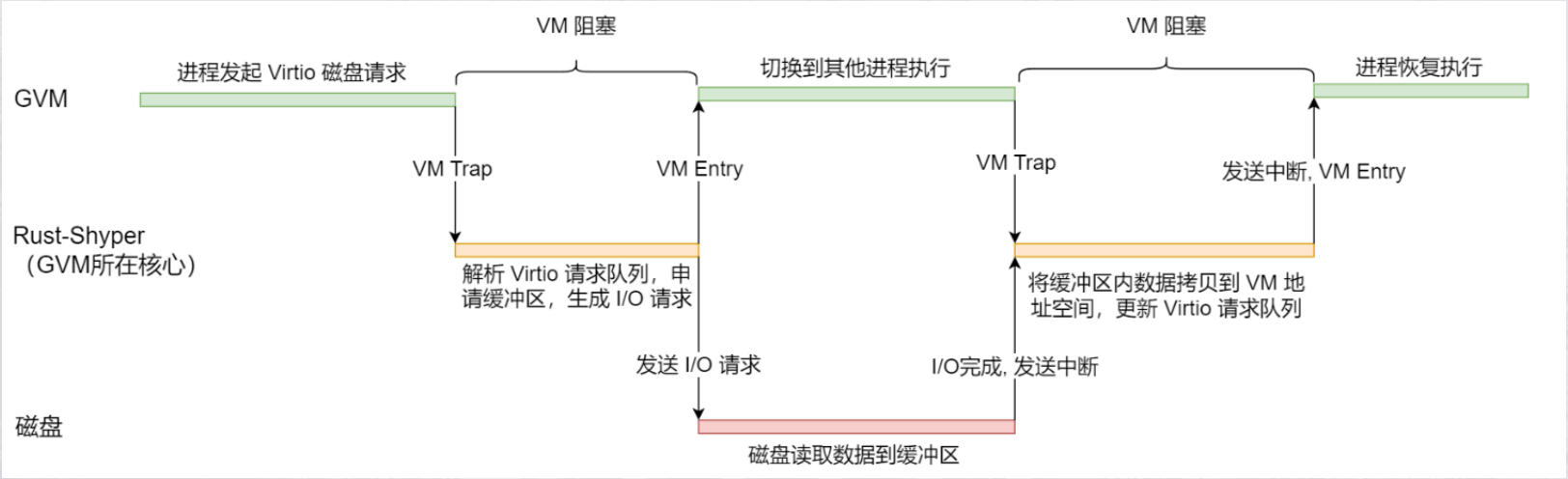
- 中介传递（Mediated Pass-through）设备驱动机制主要被用于GPU、磁盘等设备的虚拟化；
- 主要设计思路：把会影响性能的外设访问直接传递给虚拟机；把性能无关，功能性的MMIO访问进行拦截并在Hypervisor内模拟。

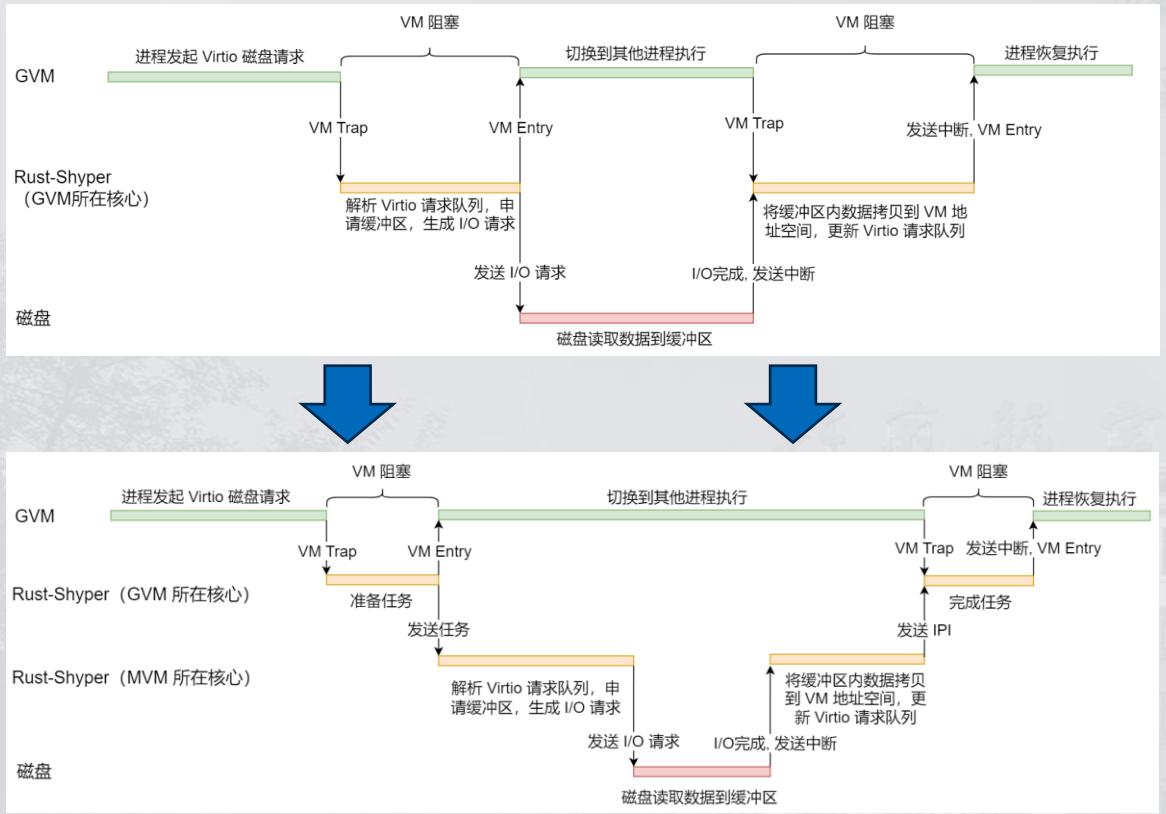
- 中介传递磁盘驱动虚拟化为例：由Rust-Shyper与MVM协同完成，Rust-Shyper仅负责I/O请求的解析与传递，而主要的磁盘读写操作由Service OS中的设备驱动完成。
- 整个流程由以下部分共同完成：
  - Rust-Shyper内部的异常处理函数；
  - Virtqueue队列解析函数；
  - I/O分配器、hypercall分配器；
  - MVM内部的内核模块与用户态设备模型守护进程；



传统的半模拟设备的处理流程

- 以基于Virtio的中介传递磁盘设备为例
- 存在的问题：长时间的陷入Hypervisor层。
- 处理流程中，有大量的Hypervisor处理流程，包括从VirtIO队列中提取请求、合并和缓冲区申请、数据拷贝以及VirtIO 队列的更新等，长时间的阻塞了GVM的正常运行。
- 此时上层VM中的全部进程都处于完全被阻塞的状态；
- 导致关键任务的实时性约束就有可能被破坏。





中介传递设备实时性优化

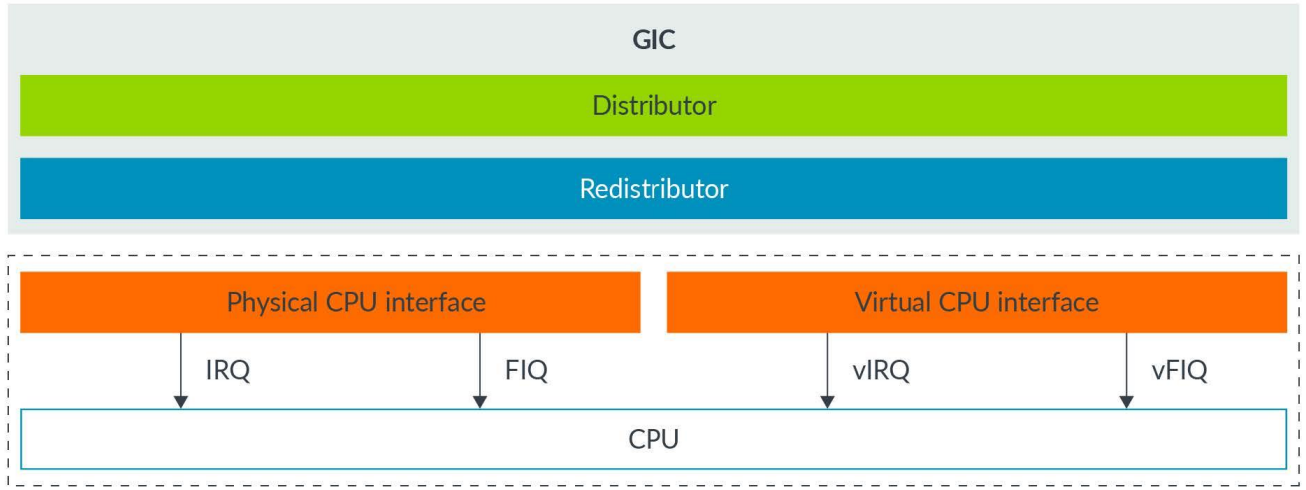
- 在系统运行过程中，Hypervisor需要处理来自上层VM的多种不同的同步异常陷入（设备模拟、hypercall等）。
- 优化方法：将Hypervisor的同步异常处理流程异步化，并将处理过程转移到特定的物理核心去完成。
- 中介异步任务组成部分：
  - 任务生成模块；
  - 任务调度器；
  - 任务缓冲队列；
  - 任务执行单元；

中断虚拟化

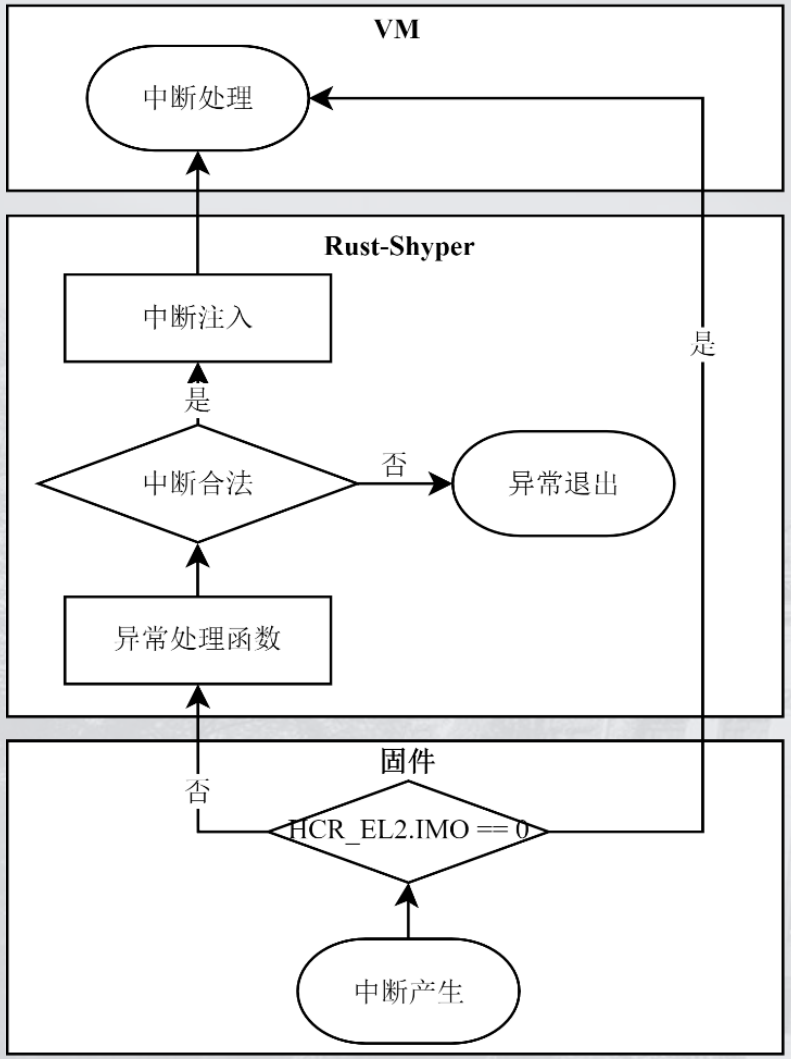
- 以GICv2的中断虚拟化为例：
- 虚拟中断控制器需要向虚拟机提供GICC、GICD两组寄存器的抽象；
- GICC由ARMv8提供的硬件虚拟化扩展支持；GICD的虚拟化需要依赖纯软件实现。
- 即：直通GICV，模拟GICD

中断虚拟化存在的问题

- Hypervisor拦截所有外部中断，导致频繁VM Exit；
- 所有虚拟中断依赖Hypervisor注入；有大量hypervisor的runtime开销，影响中断时延；
- 即使通过GIC硬件虚拟化扩展减少了中断虚拟化的开销，整个过程依然还是会引入一定的延迟：
- 与裸机相比，中断时延增大约10倍；



Rust-Shype实时虚拟化-中断



中断虚拟化

- 设置HCR_EL2.IMO
- 直通GICV，模拟GICD
- 拦截虚拟机GICD请求，通过中断位图判断中断是否合法
- Rust-Shyper完成中断处理或中断注入

中断控制器直通策略

- 设置HCR_EL2.IMO；
- 直通GICC，透传GICD；
- 仅在配置GICD阶段陷入Hypervisor；
- 直通外设中断不再被Rust-Shyper拦截，由虚拟机直接完成中断处理；

多平台移植面临的问题

多平台移植的问题

硬件平台

- 体系结构不一致
- 板载设备型号不一致（如GIC版本，串口型号等）
- 设备内存地址不一致

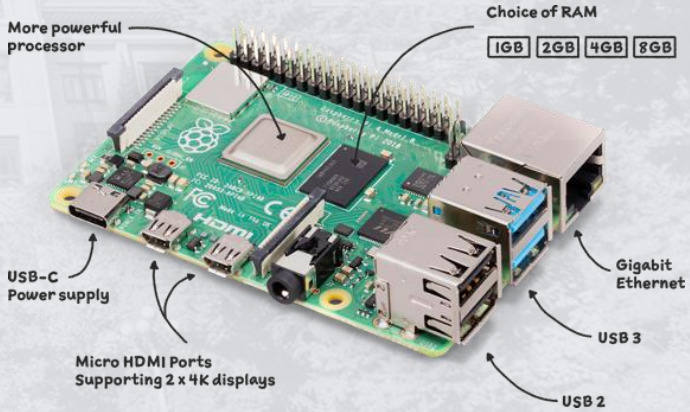
代码实现

- 复杂的驱动实现
- 启动引导程序不一致
- 底层接口抽象

QEMU

将大部分工作交给 MVM 完成

- 内存地址
- 串口驱动、中断控制器驱动
- 直通设备管理



Rust-Shyper 平台相关抽象

代码整体结构

代码结构

```
.
├── aarch64-pi4.json 编译器配置文件
├── aarch64-qemu.json
├── aarch64-tx2.json
├── aarch64-tx2-update.json
├── build.rs        Rust构建脚本
├── Cargo.lock      Rust包管理信息
├── Cargo.toml
├── doc
│   └── 基于Rust的嵌入式虚拟机监视器及热更新技术（王雷）.pdf
├── gdb
│   └── aarch64.gdb
├── image MVM（类Xen的Domain0）虚拟机的镜像，和hypervisor一同加载
│   ├── Image_pi4_5.4.83_tlb
│   ├── Image_vanilla
│   └── L4T
├── libfdt-binding 修改设备树C代码，通过Rust FFI bindgen引入
├── LICENSE
├── Makefile        编译用的makefile
├── README.ch.md
├── README.md
├── rust-toolchain Rust工具链配置信息，如rustc版本、需要rust-src等
├── src             核心代码
│   ├── arch        体系结构相关
│   ├── board       不同平台的配置
│   ├── config      动态配置虚拟机的代码，在MVM通过hypercall可动态配置
│   ├── device      Hypervisor提供的模拟设备，主要是virtio后端设备
│   ├── driver      驱动代码，如virtio前端、gpio等
│   ├── kernel      VM，Vcpu调度，内存管理，中断管理，热更新 // 解耦不够
│   ├── lib         一些util函数
│   ├── linkers     不同平台的链接脚本
│   ├── main.rs
│   ├── mm          Rust内存相关，主要是Rust堆，用的rCore的buddy system
│   ├── panic.rs
│   └── vmm         虚拟机管理相关代码，
└── tools           配套的内核模块和用户态commandline interface(Cli)，这两部分是没有开源的
    ├── shyper      用户态Cli
    └── shyper.ko   内核模块
```

体系结构抽象

```
Siran Li, 2 years ago | 1 author (Siran Li) | 1 implementation
pub trait ContextFrameTrait {
    fn new(pc: usize, sp: usize, arg: usize) -> Self;

    fn exception_pc(&self) -> usize;
    fn set_exception_pc(&mut self, pc: usize);
    fn stack_pointer(&self) -> usize;
    fn set_stack_pointer(&mut self, sp: usize);
    fn set_argument(&mut self, arg: usize);
    fn set_gpr(&mut self, index: usize, val: usize);
    fn gpr(&self, index: usize) -> usize;
}

Siran Li, 2 years ago | 1 author (Siran Li) | 1 implementation
pub trait ArchPageTableEntryTrait {
    fn from_pte(value: usize) -> Self;
    fn from_pa(pa: usize) -> Self;
    fn to_pte(&self) -> usize;
    fn to_pa(&self) -> usize;
    fn valid(&self) -> bool;
    fn entry(&self, index: usize) -> Self;
    fn set_entry(&self, index: usize, value: Self);
    fn make_table(frame_pa: usize) -> Self;
}
```

体系结构抽象

aarch64

```
Siran Li, 2 years ago | 1 author (Siran Li) | 1 implementation
pub trait ContextFrameTrait {
    fn new(pc: usize, sp: usize, arg: usize) -> Self;

    fn exception_pc(&self) -> usize;
    fn set_exception_pc(&mut self, pc: usize);
    fn stack_pointer(&self) -> usize;
    fn set_stack_pointer(&mut self, sp: usize);
    fn set_argument(&mut self, arg: usize);
    fn set_gpr(&mut self, index: usize, val: usize);
    fn gpr(&self, index: usize) -> usize;
}

Siran Li, 2 years ago | 1 author (Siran Li) | 1 implementation
pub trait ArchPageTableEntryTrait {
    fn from_pte(value: usize) -> Self;
    fn from_pa(pa: usize) -> Self;
    fn to_pte(&self) -> usize;
    fn to_pa(&self) -> usize;
    fn valid(&self) -> bool;
    fn entry(&self, index: usize) -> Self;
    fn set_entry(&self, index: usize, value: Self);
    fn make_table(frame_pa: usize) -> Self;
}
```

```
Siran Li, 18 months ago | 1 author (Siran Li)
#[repr(C)]
#[derive(Copy, Clone, Debug)]
6 implementations
pub struct Aarch64ContextFrame {
    gpr: [u64; 31],
    pub spsr: u64,
    elr: u64,
    sp: u64,
}

impl core::fmt::Display for Aarch64ContextFrame {
    fn fmt(&self, f: &mut Formatter<'_>) -> Result<(), core::fmt::Error> {
        for i: usize in 0..31 {
            write!(f, "x{:02}: {:016x}   ", i, self.gpr[i])?;
            if (i + 1) % 2 == 0 {
                write!(f, "\n")?;
            }
        }
        writeln!(f, "spsr:{:016x}", self.spsr)?;
        write!(f, "elr: {:016x}", self.elr)?;
        writeln!(f, "    sp:  {:016x}", self.sp)?;
        Ok(())
    }
}
48
49  impl crate::arch::ContextFrameTrait for Aarch64ContextFrame {
50      fn new(pc: usize, sp: usize, arg: usize) -> Self {…
65
66      fn exception_pc(&self) -> usize {…
69
70      fn set_exception_pc(&mut self, pc: usize) {…
73
74      fn stack_pointer(&self) -> usize {…
77
78      fn set_stack_pointer(&mut self, sp: usize) {…
81
82      fn set_argument(&mut self, arg: usize) {…
85
86      fn set_gpr(&mut self, index: usize, val: usize) {…
89
90      fn gpr(&self, index: usize) -> usize {…
93  }impl ContextFrameTrait for Aarch64ContextFrame
```

体系结构抽象

aarch64

```
Siran Li, 2 years ago | 1 author (Siran Li) | 1 implementation
pub trait ContextFrameTrait {
    fn new(pc: usize, sp: usize, arg: usize) -> Self;

    fn exception_pc(&self) -> usize;
    fn set_exception_pc(&mut self, pc: usize);
    fn stack_pointer(&self) -> usize;
    fn set_stack_pointer(&mut self, sp: usize);
    fn set_argument(&mut self, arg: usize);
    fn set_gpr(&mut self, index: usize, val: usize);
    fn gpr(&self, index: usize) -> usize;
}
```

```
Siran Li, 2 years ago | 1 author (Siran Li) | 1 implementation
pub trait ArchPageTableEntryTrait {
    fn from_pte(value: usize) -> Self;
    fn from_pa(pa: usize) -> Self;
    fn to_pte(&self) -> usize;
    fn to_pa(&self) -> usize;
    fn valid(&self) -> bool;
    fn entry(&self, index: usize) -> Self;
    fn set_entry(&self, index: usize, value: Self);
    fn make_table(frame_pa: usize) -> Self;
}
```

```
Siran Li, 2 years ago | 1 author (Siran Li)
#[repr(transparent)]
#[derive(Copy, Clone, Debug)]
4 implementations
pub struct Aarch64PageTableEntry(usize);

impl ArchPageTableEntryTrait for Aarch64PageTableEntry {
    fn from_pte(value: usize) -> Self {…
    fn from_pa(pa: usize) -> Self {…
    fn to_pte(&self) -> usize {…
    fn to_pa(&self) -> usize {…
    fn valid(&self) -> bool {…
    fn entry(&self, index: usize) -> Aarch64PageTableEntry {…
    fn set_entry(&self, index: usize, value: Aarch64PageTableEntry) {…
    fn make_table(frame_pa: usize) -> Self {…
}impl ArchPageTableEntryTrait for Aarch64PageTableEntry
```

```
moce0627, 5 months ago | 2 authors (Siran Li and others)
#[derive(Clone)]
2 implementations
pub struct PageTable {
    pub directory: Arc<PageFrame>,
    pub pages: Arc<Mutex<Vec<PageFrame>>>,
}
```

体系结构抽象

Siran Li, 2 years ago | 1 author (Siran Li) | 1 implementation

```rust
pub trait ContextFrameTrait {
    fn new(pc: usize, sp: usize, arg: usize) -> Self;

    fn exception_pc(&self) -> usize;
    fn set_exception_pc(&mut self, pc: usize);
    fn stack_pointer(&self) -> usize;
    fn set_stack_pointer(&mut self, sp: usize);
    fn set_argument(&mut self, arg: usize);
    fn set_gpr(&mut self, index: usize, val: usize);
    fn gpr(&self, index: usize) -> usize;
}
```

Siran Li, 2 years ago | 1 author (Siran Li) | 1 implementation

```rust
pub trait ArchPageTableEntryTrait {
    fn from_pte(value: usize) -> Self;
    fn from_pa(pa: usize) -> Self;
    fn to_pte(&self) -> usize;
    fn to_pa(&self) -> usize;
    fn valid(&self) -> bool;
    fn entry(&self, index: usize) -> Self;
    fn set_entry(&self, index: usize, value: Self);
    fn make_table(frame_pa: usize) -> Self;
}
```

aarch64

moce0627, 6 months ago | 2 authors (Siran Li and others)

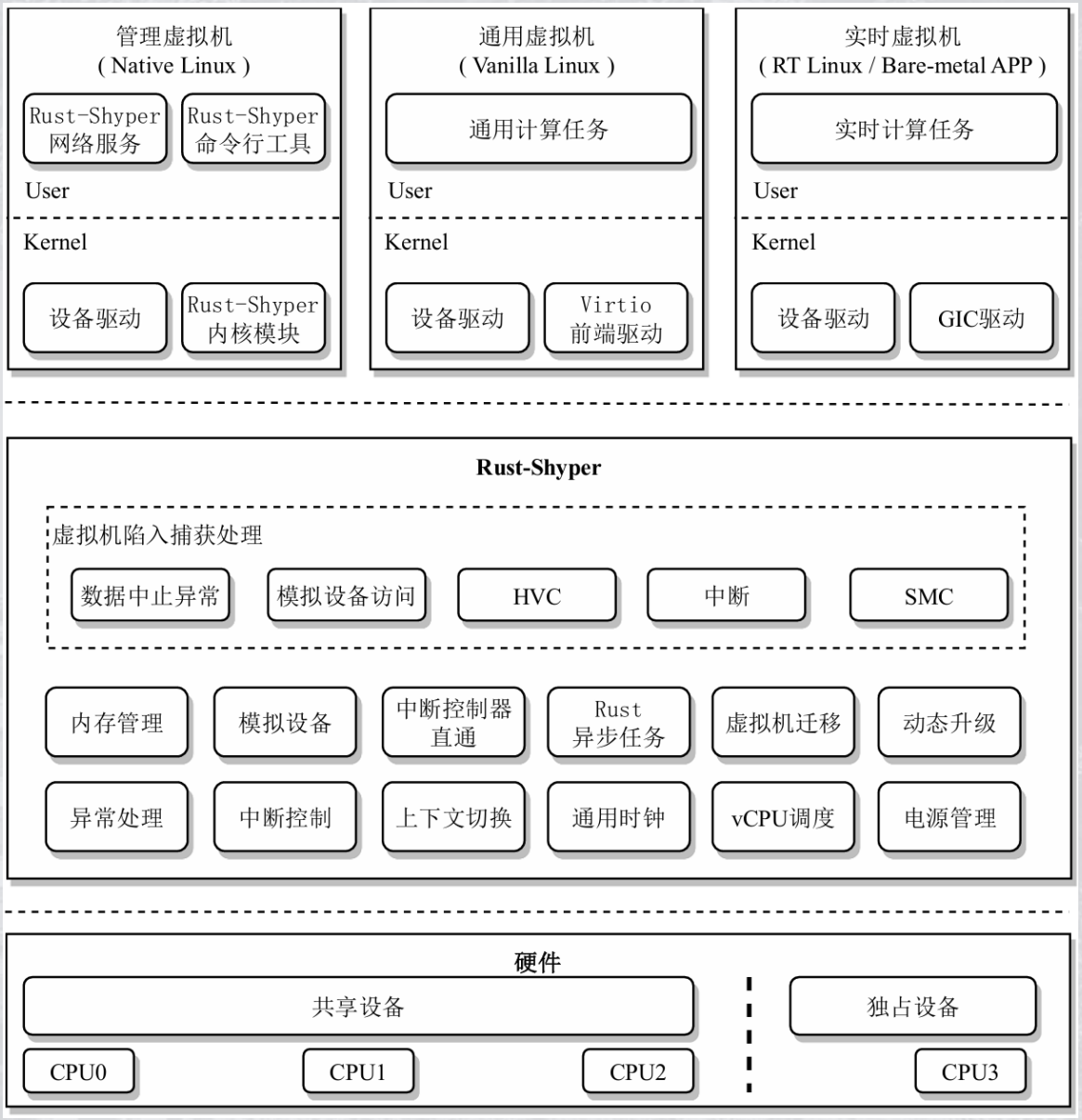
```rust
#[derive(Clone)]
2 implementations
pub struct PageTable {
    pub directory: Arc<PageFrame>,
    pub pages: Arc<Mutex<Vec<PageFrame>>>,
}

impl PageTable {
    pub fn new(directory: PageFrame) -> PageTable {…
    pub fn base_pa(&self) -> usize {…
    pub fn access_permission(&self, start_ipa: usize, len: usize, ap: usize) -> (usize, usize) {…
    pub fn map_2mb(&self, ipa: usize, pa: usize, pte: usize) {…
    pub fn unmap_2mb(&self, ipa: usize) {…
    pub fn map(&self, ipa: usize, pa: usize, pte: usize) {…
    pub fn unmap(&self, ipa: usize) {…
    pub fn map_range_2mb(&self, ipa: usize, len: usize, pa: usize, pte: usize) {…
    pub fn unmap_range_2mb(&self, ipa: usize, len: usize) {…
    pub fn map_range(&self, ipa: usize, len: usize, pa: usize, pte: usize) {…
    pub fn unmap_range(&self, ipa: usize, len: usize) {…
    pub fn show_pt(&self, ipa: usize) {…
    pub fn pt_map_range(&self, ipa: usize, len: usize, pa: usize, pte: usize, map_block: bool) {…
    pub fn pt_unmap_range(&self, ipa: usize, len: usize, map_block: bool) {…
} impl PageTable
```

MVM 与多平台兼容

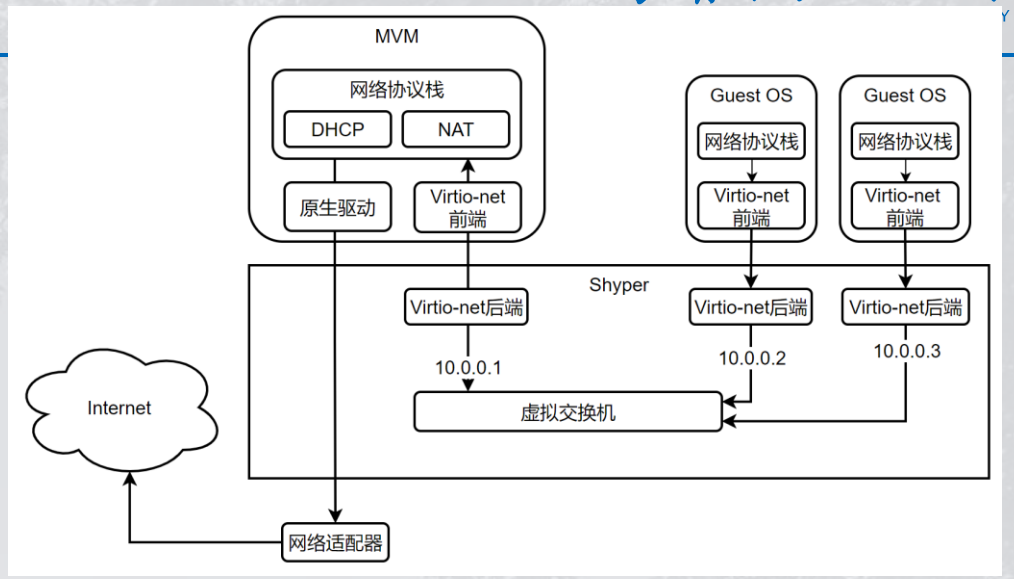
Rust-Shyper VM 类型

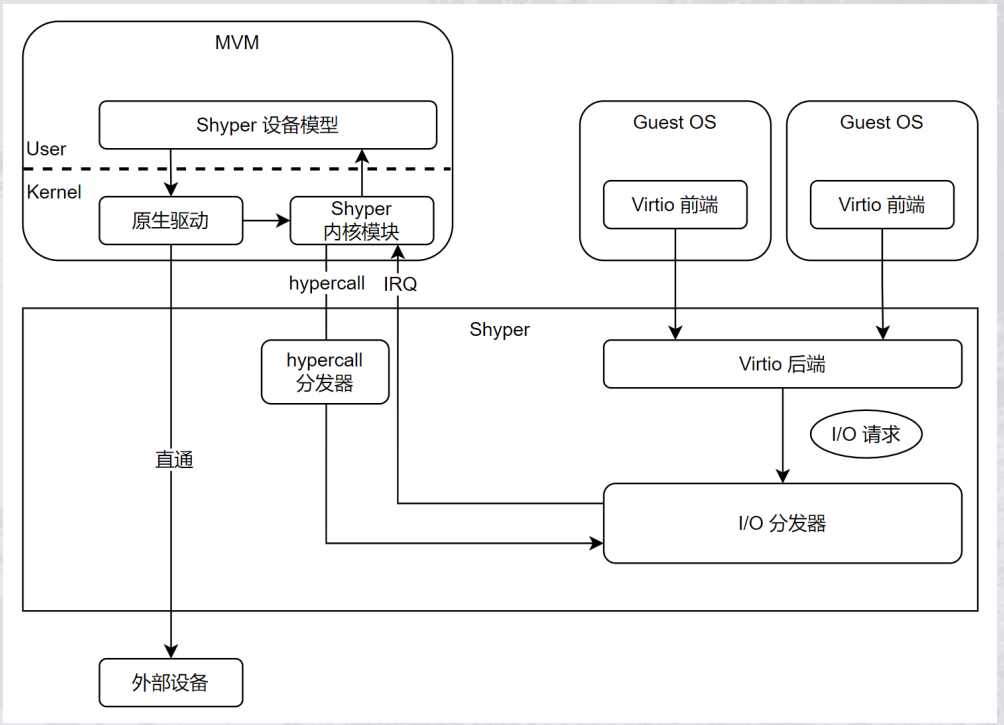
- MVM：一般来说为 VM[0]，通常为平台支持的原生Linux。通过调用hypervisor向下暴露的hvc接口，实现对其他VM的监控、管理，一般交付给管理员进行操作使用。
- VM[1..x]：依照提供的配置内容，可以运行各类系统或裸应用程序，例如Linux、RT Linux、RTOS、Bare-metal App。
- Rust-Shyper 保留静态配置设计，MVM 可以在配置中被移除，Rust-Shyper 上的所有 Guest 根据配置文件自动被启动



Rust-Shyper MVM 功能

- 负责接收用户输入，管理其他 Guest VM
- 实现硬件平台完善的驱动支持
- 用于支持其他 Guest VM 的网络访问请求：通过 NAT、IP路由表等配置，与 Virtio 虚拟网卡配合实现 Guest VM对网络资源的访问
- 完成基于 offload 思想的设备驱动虚拟化中，来自 Guest VM 的设备访问请求（以块设备为例）





Device Tree Source

- Device Tree是一种描述硬件的数据结构
- DTS即Device Tree Source 设备树源码
- DTC(Device Tree Compiler)负责将可读的DTS编译成机器处理的 DTB(Device Tree binary file)
- 在系统启动的时候，由启动程序负责为 Linux 准备 DTB，并告知所在地址

```
chosen {
    stdout-path = "/pl011@9100000";
    // bootargs = "earlycon console=ttyAMA0 rdinit=/sbin/init audit=0";
    bootargs = "earlycon console=ttyAMA0 root=/dev/vda rw audit=0";
    linux,initrd-start = <0x53000000>;
    linux,initrd-end = <0x5310dab1>;
};

virtio_mmio@a001000 {
    dma-coherent;
    compatible = "virtio,mmio";
    interrupts = <0x00 0x11 0x01>;
    reg = <0x00 0xa001000 0x00 0x1000>;
};
```

- Device Tree由一系列被命名的结点（node）和属性（property）组成，而结点本身可包含子结点
- 在Device Tree中，可描述的信息包括：
  - CPU的数量和类别
  - 内存基地址和大小
  - 总线和桥
  - 外设连接
  - 中断控制器和中断使用情况
  - GPIO控制器和GPIO使用情况
  - Clock控制器和Clock使用情况

MVM 设备树 setup

- 为 MVM 准备设备树文件
- 由 uboot 提供
- Hypervisor 根据需求对设备树进行相关修改
- 依赖libfdt

```
pub fn init_vm0_dtb(dtb: *mut fdt::myctypes::c_void) {
    #[cfg(feature = "tx2")]···
    #[cfg(feature = "pi4")]···
    #[cfg(feature = "qemu")]···
} fn init_vm0_dtb
```

- 后续的 GVM 设备树都由 Hypervisor 生成
- 与硬件平台无关

```
pub unsafe fn vmm_setup_fdt(vm: Vm) {
    use fdt::*;
    let config: VmConfigEntry = vm.config();
    match vm.dtb() {
        Some(dtb: *mut c_void) => {
            let mut mr: Vec<region> = Vec::new();
            for r: VmRegion in config.memory_region() {
                mr.push(region {
                    ipa_start: r.ipa_start as u64,
                    length: r.length as u64,
                });      Siran Li, 20 months ago • feat: add fdt & new virtio_net_dev
            }
            #[cfg(feature = "tx2")]
            fdt_set_memory(dtb, mr.len() as u64, mr.as_ptr(), "memory@90000000\0".as_ptr());
            #[cfg(feature = "pi4")]
            fdt_set_memory(fdt: dtb, region_num: mr.len() as u64, regions: mr.as_ptr(), node_name: "memory@200000\0".as_ptr());
            #[cfg(feature = "qemu")]
            fdt_set_memory(dtb, mr.len() as u64, mr.as_ptr(), "memory@50000000\0".as_ptr());
            // FDT+TIMER
            fdt_add_timer(fdt: dtb, trigger_lvl: 0x8);
            // FDT+BOOTCMD
            fdt_set_bootcmd(fdt: dtb, cmdline: config.cmdline.as_ptr());
            #[cfg(feature = "tx2")]
            fdt_set_stdout_path(dtb, "/serial@3100000\0".as_ptr());
            // #[cfg(feature = "pi4")]
            // fdt_set_stdout_path(dtb, "/serial@fe340000\0".as_ptr());

            if config.emulated_device_list().len() > 0 {
                for emu_cfg: VmEmulatedDeviceConfig in config.emulated_device_list() {
                    match emu_cfg.emu_type {
                        EmuDeviceTGicd => {···
                        EmuDeviceTVirtioNet | EmuDeviceTVirtioConsole => {···
                        EmuDeviceTShyper => {···
                        EmuDeviceTIOMMU => {···
                        _ => {
                            todo!();
                        }
                    }
                }
            }
            println!("after dtb size {}", fdt_size(dtb));
        }
        None => {
            println!("None dtb");
        }
    }
} fn vmm_setup_fdt
```

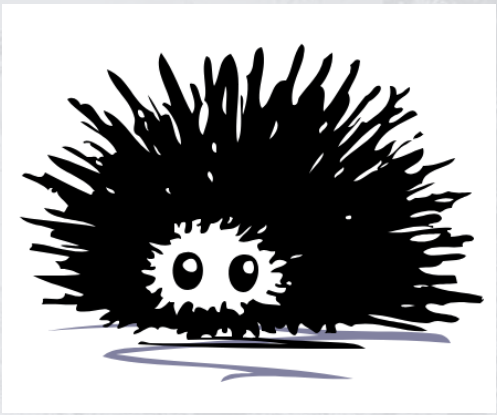
Rust语言unsafe代码

1、为什么需要unsafe?

- 强大但保守的编译器（能力有限）
  - Rust编译器拥有强大的静态检查能力；
  - 但是Rust编译器的静态检查在分析代码时，一些正确代码会因为编译器无法分析出它的所有正确性，导致编译报错。

Safe Rust

- 底层软件的编程需要（现实需要）
  - 计算机底层的一些硬件就是不安全的，如果 Rust 只允许你做安全的操作，那一些任务就无法完成。
  - 系统编程中unsafe必不可少。



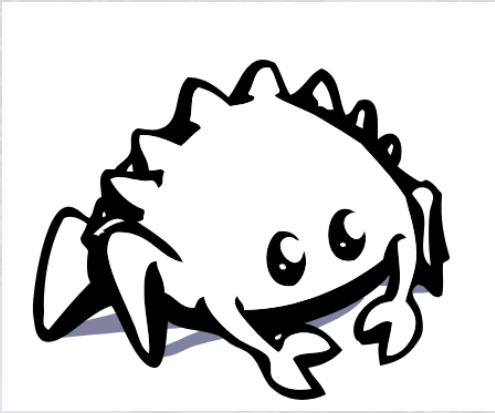
Unsafe Rust

2、unsafe的能力

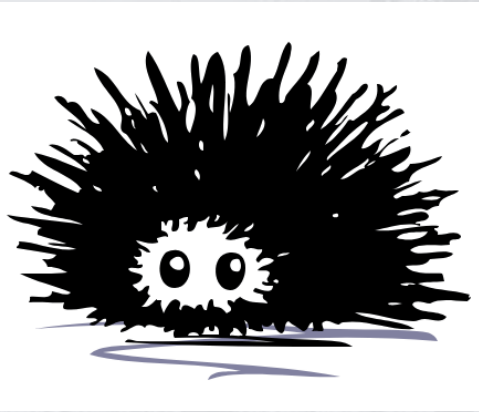
Safe Rust无法做到，只有unsafe才能做到的能力：

- 解引用裸指针raw pointer；
- 调用Unsafe函数（如跨语言调用汇编、C语言）；
- 访问或修改一个可变的静态变量；
- 实现一个 unsafe Trait；
- 访问 union 的字段

https://doc.rust-lang.org/book/ch19-01-unsafe-rust.html



Safe Rust



Unsafe Rust

2、unsafe的超能力（续）

- 解引用裸指针；
- 调用Unsafe函数（如跨语言调用汇编、C语言）；
- 访问或修改一个可变的静态变量；
- 实现一个 unsafe trait特征；
- 访问 union 的字段

以Rust-Shyper中用到的Unsafe举例：

```rust
impl core::ops::Deref for GicDistributor {
    type Target = GicDistributorBlock;
    fn deref(&self) -> &Self::Target {
        unsafe { &*(self.base_addr as *const Self::Target) }
    }
}
```

```rust
fn invalid_guest_ipa(ipa: usize) {
    unsafe {
        asm!(
            "dsb ish",
            "tlbi ipas2e1is, {0}",
            "dsb ish",
            "isb",
            in(reg) ipa >> 12,
            options(nostack)
        );
    }
}
```

2、unsafe的超能力（续）

- 解引用裸指针；
- 调用Unsafe函数（如跨语言调用汇编、C语言）；
- 访问或修改一个可变的静态变量；
- 实现一个 unsafe trait特征；
- 访问 union 的字段（很少使用）

以Rust-Shyper中用到的Unsafe举例：

```rust
static mut CPU_GLB_SYNC: CpuSyncToken = CpuSyncToken {
    n: PLAT_DESC.cpu_desc.num,
    count: AtomicUsize::new(0),
    ready: true,
};
pub fn barrier() {
    unsafe {
        let ori = CPU_GLB_SYNC.count.fetch_add(1, Ordering::Relaxed);
        let next_count = round_up(ori + 1, CPU_GLB_SYNC.n);
        while CPU_GLB_SYNC.count.load(Ordering::Acquire) < next_count {
            core::hint::spin_loop();
        }
    }
}
```

```rust
struct VgicIntInnerConst {
    id: u16,
    hw: Cell<bool>,
}

// SAFETY: VgicIntInnerConst hw is only set when initializing
unsafe impl Send for VgicIntInnerConst {}
unsafe impl Sync for VgicIntInnerConst {}
```

3、如何看待Unsafe Rust?

- Unsafe Rust也有安全检查：
  - 不能绕过 Rust 的借用检查、类型系统等；
  - 不能关闭 Rust 的安全检查规则；
- Unsafe Rust并非不安全：
  - unsafe不意味着块中的代码就一定是不安全的；
  - unsafe存在的意义是：程序员会确保 unsafe 块中的代码以有效正确的方式访问内存。
- Unsafe Rust是标准库的基石之一：
  - core等Rust标准库均包含大量Unsafe代码。
  - 没必要对Unsafe谈虎色变。

```rust
impl<T, A: Allocator> Vec<T, A> {
    #[stable(feature = "rust1", since = "1.0.0")]
    pub fn truncate(&mut self, len: usize) {
        // This is safe because:
        //
        // * the slice passed to `drop_in_place` is valid; the `len > self.len`
        //   case avoids creating an invalid slice, and
        // * the `len` of the vector is shrunk before calling `drop_in_place`,
        //   such that no value will be dropped twice in case `drop_in_place`
        //   were to panic once (if it panics twice, the program aborts).
        unsafe {
            // Note: It's intentional that this is `>` and not `>=`.
            //       Changing it to `>=` has negative performance
            //       implications in some cases. See #78884 for more.
            if len > self.len {
                return;
            }
            let remaining_len = self.len - len;
            let s = ptr::slice_from_raw_parts_mut(self.as_mut_ptr().add(len), remaining_len);
            self.len = len;
            ptr::drop_in_place(s);
        }
    }
}
```

4、如何处理unsafe Rust?

- 隔离，封装，最小化
- 形式化验证

Unsafe——合理使用

1、隔离

- 分离作用域：尽可能分离Safe Rust和unsafe的作用域；
- Unsafe函数尽可能只传递基础类型（如裸指针、整数）；

2、封装

- 将unsafe Rust封装成safe函数调用，并增加相应检查；
- 利用Rust宏等手段，集中unsafe代码的使用；

3、最小化

- 最小化unsafe 的出现频率：利用上述的隔离和封装等手段；
- 加速排错：当有程序错误发生时，任何与内存安全相关的错误必定位于 unsafe 块周围；

```rust
// WARNING: No Auto `drop` in this function
pub fn vcpu_run(announce: bool) -> ! {
    {
        let vcpu = current_cpu().active_vcpu.clone().unwrap();
        let vm = vcpu.vm().unwrap();

        current_cpu().cpu_state = CpuState::CpuRun;
        vm_if_set_state(active_vm_id(), super::VmState::VmActive);

        vcpu.context_vm_restore();
        if announce {
            crate::device::virtio_net_announce(vm);
        }
    }
```

```rust
// Move to ARM register from system coprocessor register.
// MRS Xd, sysreg "Xd = sysreg"
macro_rules! mrs {
    ($val: expr, $reg: expr, $asm_width:tt) => {
        unsafe {
            core::arch::asm!(concat!("mrs {0:", $asm_width, "}, ", stringify!($reg)), out(reg) $val,
options(nomem, nostack));
        }
    };
    ($val: expr, $reg: expr) => {
        unsafe {
            core::arch::asm!(concat!("mrs {0}, ", stringify!($reg)), out(reg) $val, options(nomem, nostack));
        }
    };
}
mrs!(self.cntvoff_el2, CNTVOFF_EL2);
mrs!(self.cntv_cval_el0, CNTV_CVAL_EL0);
mrs!(self.cntkctl_el1, CNTKCTL_EL1, "x");
mrs!(self.cntp_ctl_el0, CNTP_CTL_EL0, "x");
mrs!(self.cntv_ctl_el0, CNTV_CTL_EL0, "x");
```

```rust
pub fn memset_safe(s: *mut u8, c: i32, n: usize) -> *mut u8 {
    if (s as usize) < 0x1000 {
        panic!("illegal addr for memset s {:x}", s as usize);
    }
    unsafe { memset(s, c, n) }
}
```

实验评测

代码组成

Language	files	blank	comment	code
Rust	96	2414	2699	**18229**
Assembly	9	175	195	992
SUM:	105	2589	2894	19221

Unsafe Rust 代码量

Unsafe Rust使用：141 LoC

SyberX ISO-17961扫描结果

漏洞名称	严重等级	数量
CERT-5.14 空指针解引用	严重——高	1
CERT-5.44 使用无效的格式字符串	重要——中	60
CERT-5.06 带有不正确参数的函数调用	次要——中	18
CERT-5.10 整型和指针之间的相互转换	次要——中	29

SyberX MISRA_2004扫描结果

漏洞名称	严重等级	数量
MISRA_01.02不能有对未定义行为或未指定行为的依赖性	次要——中	3

SyberX C_CPP缺陷项扫描结果

漏洞名称	严重等级	数量
S_30_05 被赋值为NULL的指针发生空指针解引用	致命——高	1
S_26_02 参数数量不匹配	严重——高	3
S_26_04 参数类型不兼容	严重——高	57
S_30_07 可疑的空指针解引用	严重——高	2
S_30_06 指针解引用前未判空	重要——中	133
S_02_06 不同位数变量进行位运算	次要——中	17
S_02_12 函数通过值传递传占空间过大的参数	次要——中	1
S_02_21 无效的比较大小	次要——中	2
S_32_02 隐式类型转换可能丢失精度	次要——中	13
S_36_02用户函数返回值未测试	次要——中	2

实验环境配置

RK3588

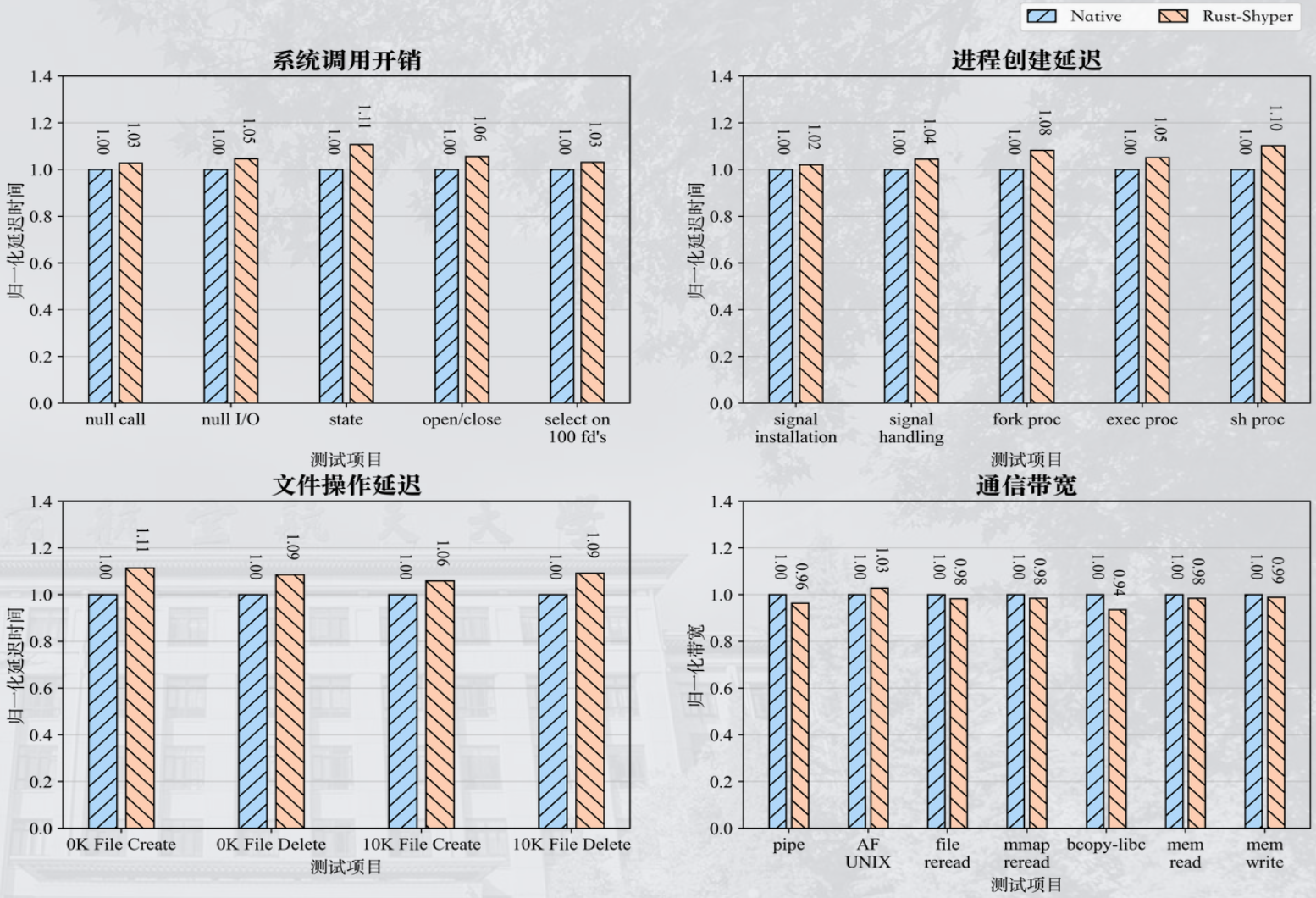
ROC-RK3588硬件信息	
处理器	4×Cortex-A76+4×Cortex-A55，主频 2.4GHz
内存	8GB 64bit LPDDR4
存储	64GB eMMC 5.1
网卡	1000Mbps 以太网（RJ45）
I/O插槽	PCIe 2.0、 SATA3.0、1 × USB3.0、3 × USB2.0

- 性能测试环境
  - 测试选用Firefly ROC-RK3588嵌入式开发板作为运行Rust-Shyper的硬件平台
  - 系统软件方面，Rust-Shyper选用Linux-5.10.160作为MVM和GVM系统镜像。
  - 本测试主要选用了原生Linux作为对比测试的对象

操作系统基本性能

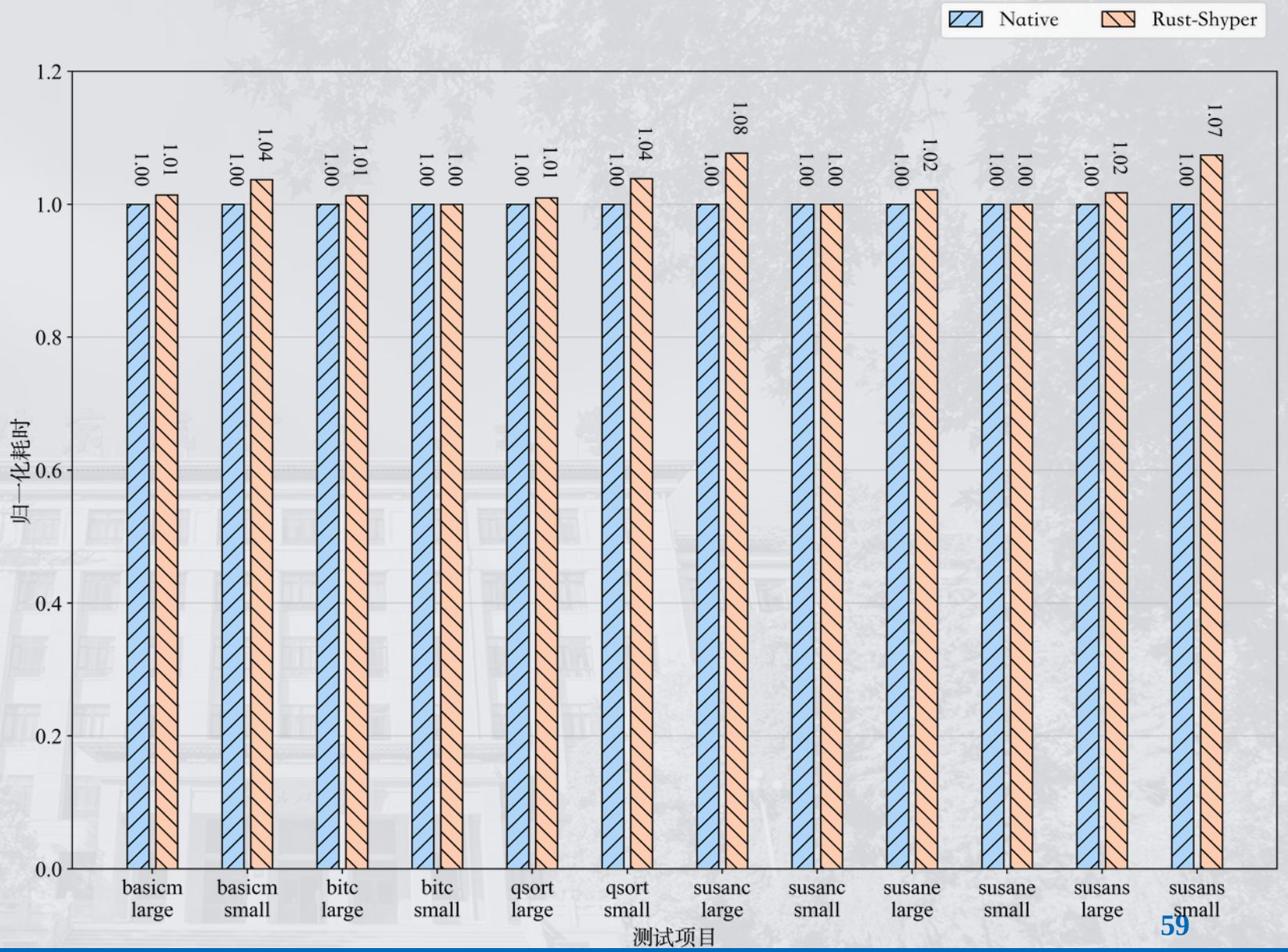
- 基本性能测试
  - 采用Lmbench测试集从文件操作延迟、通信带宽、系统调用开销、进程创建延迟四个方面进行测试

测试项目	平均性能开销
系统调用开销	5.35%
进程创建延迟	6%
文件操作延迟	8.79%
通信带宽	1.87%

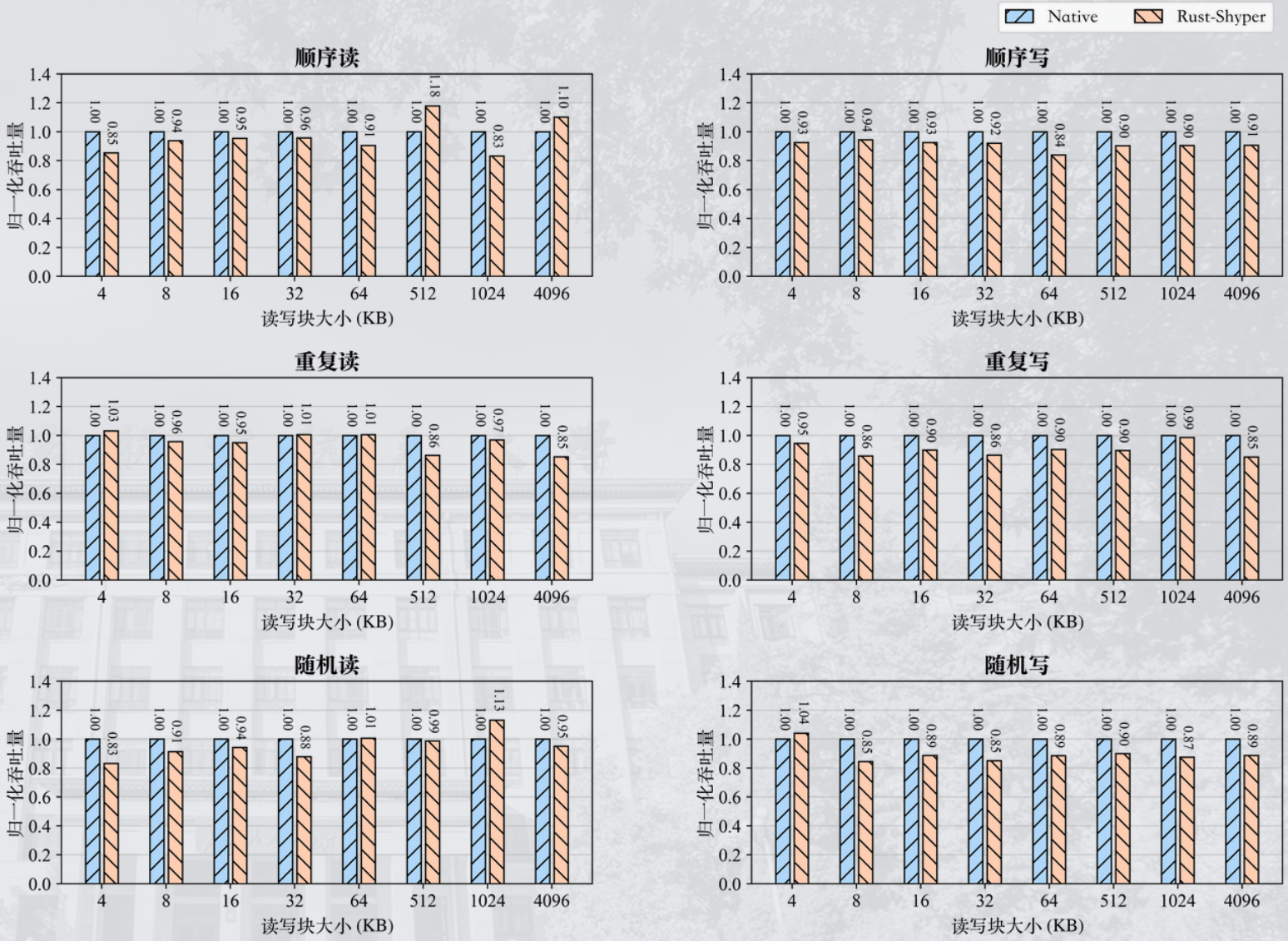


处理器与内存性能

- 处理器与内存性能测试
  - 测试采用MiBench嵌入式测试集，测试虚拟机数学运算、位运算、数据组织能力
  - 虚拟机的CPU计算性能以及内存访问性能均会对测试结果产生影响
  - 与原生 Linux 相比，Rust-Shyper VM 的平均性能损失为 2.52%，测试项目的最大性能损失为 8%



- I/O性能测试
  - IOZone对虚拟机磁盘I/O性能进行了全面的评估
  - 测试条目包括4K、8K、16K、64K、512K、1M、4M的随机读写、顺序读写、重复读写的吞吐量



- I/O性能测试
  - 于所有基准测试项，Rust-Shyper 在绝大部分情况下性能开销小于 10%
  - 性能符合项目指标要求

测试项目	平均性能开销
顺序读	4.15%
顺序写	9.23%
重复读	3.23%
重复写	9.83%
随机读	5.75%
随机写	9.34%

实验环境配置

- 实验平台：Qemu 8.2.0
- 对比对象：Native Linux、Rust-Shyper（RISC-V移植版）、KVM
- 测试OS：Ubuntu 22.04，Linux 6.1.90，2 vCPU、1G内存
- 功能测试分单VM启动、多VM启动、中介磁盘、网络通信四方面测试
- 性能测试主要使用LmBench和MiBench

操作系统基本性能

```
root@myhostname:~# free -h
               total        used        free      shared  buff/cache   available
Mem:           3.8Gi       156Mi       3.0Gi       0.0Ki       715Mi       3.5Gi
Swap:             0B          0B          0B
root@myhostname:~# uptime
 00:31:17 up 31 min,  2 users,  load average: 0.05, 0.22, 0.26
root@myhostname:~# cat /etc/os-release
PRETTY_NAME="Ubuntu 22.04.4 LTS"
NAME="Ubuntu"
VERSION_ID="22.04"
VERSION="22.04.4 LTS (Jammy Jellyfish)"
VERSION_CODENAME=jammy
ID=ubuntu
ID_LIKE=debian
HOME_URL="https://www.ubuntu.com/"
SUPPORT_URL="https://help.ubuntu.com/"
BUG_REPORT_URL="https://bugs.launchpad.net/ubuntu/"
```

单VM运

```
Welcome to Ubuntu 22.04.4 LTS (GNU/Linux 5.15.0-1028-generic riscv64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/pro

This system has been minimized by removing packages and content that are
not required on a system that users do not log into.

To restore this content, you can run the 'unminimize' command.

This is General Virtual Machine (GVM), VM1
GVM will use emulated devices maintained by MVM.

Last login: Fri May 17 21:23:03 CST 2024 on hvc1
COLUMNS=79;
LINES=25;
export COLUMNS LINES;
root@myhostname:~# free -h
               total        used        free      shared  buff/cache   availabe
Mem:           992Mi        60Mi       834Mi       2.0Mi        97Mi       913i
Swap:             0B          0B          0B
root@myhostname:~#
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.8 | VT102 | Offline | hvc1
```

多VM运行（使用minicom与GVM命令行交互）

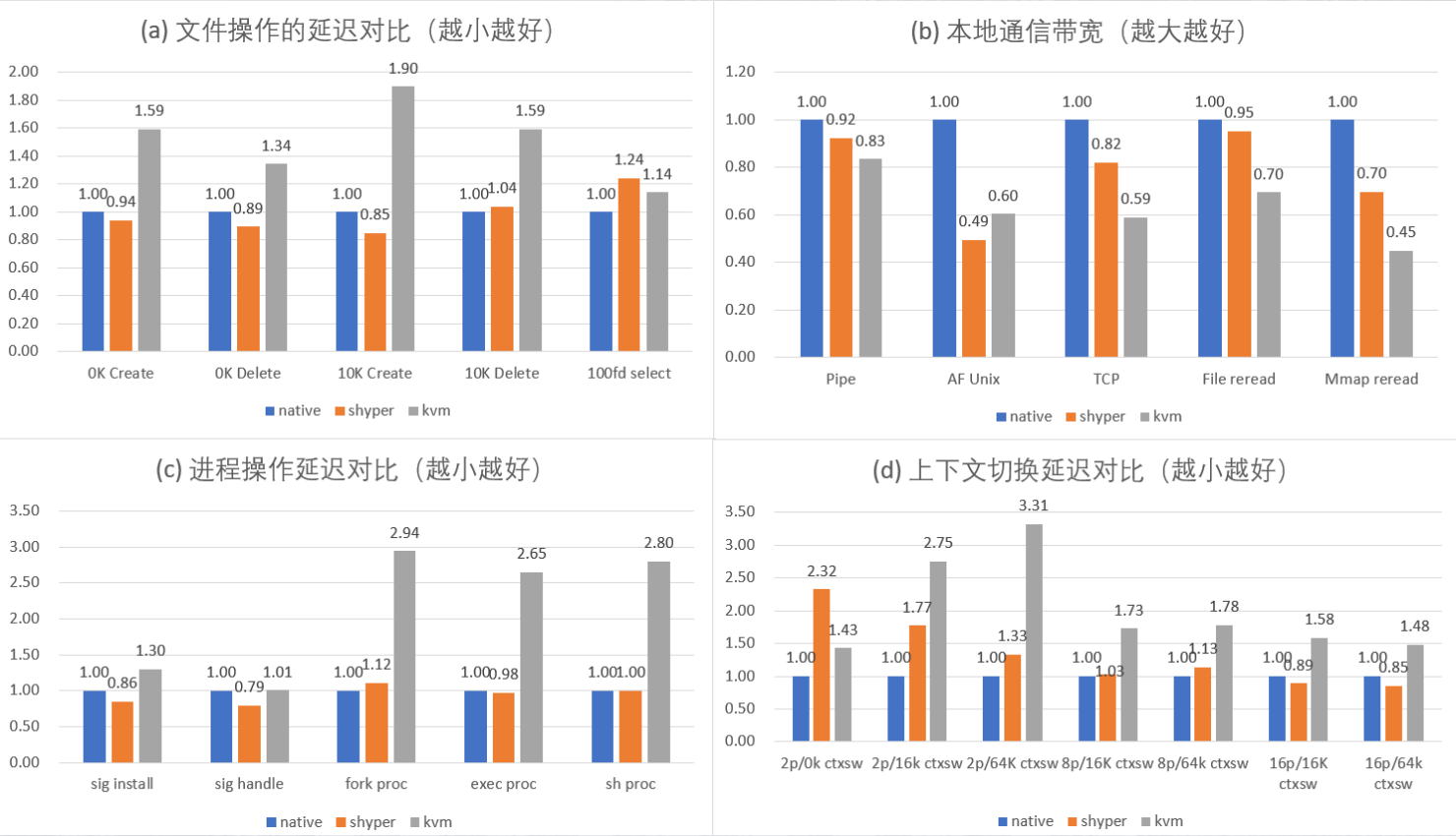
```
Auto Mode
Record Size 16384 kB
File size set to 1048576 kB
Command line used: ./iozone -a -i 0 -i 1 -i 2 -f /root/testfile -r 16m -s 1G
Output is in kBytes/sec
Time Resolution = 0.000001 seconds.
Processor cache size set to 1024 kBytes.
Processor cache line size set to 32 bytes.
File stride size set to 17 * record size.
                                                            random    random
      kB  reclen    write  rewrite    read    reread    read     write
 1048576   16384    81863   523002 2096871   1427216 2178890    556594

iozone test complete.
```

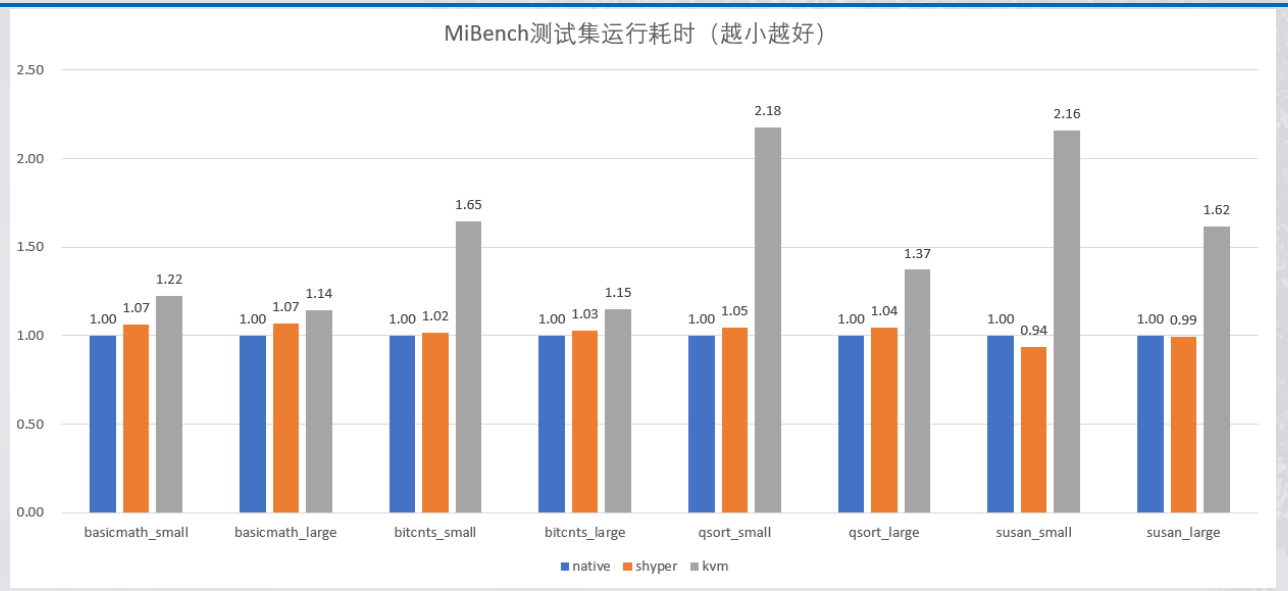
中介磁盘正确性测试（iozone）

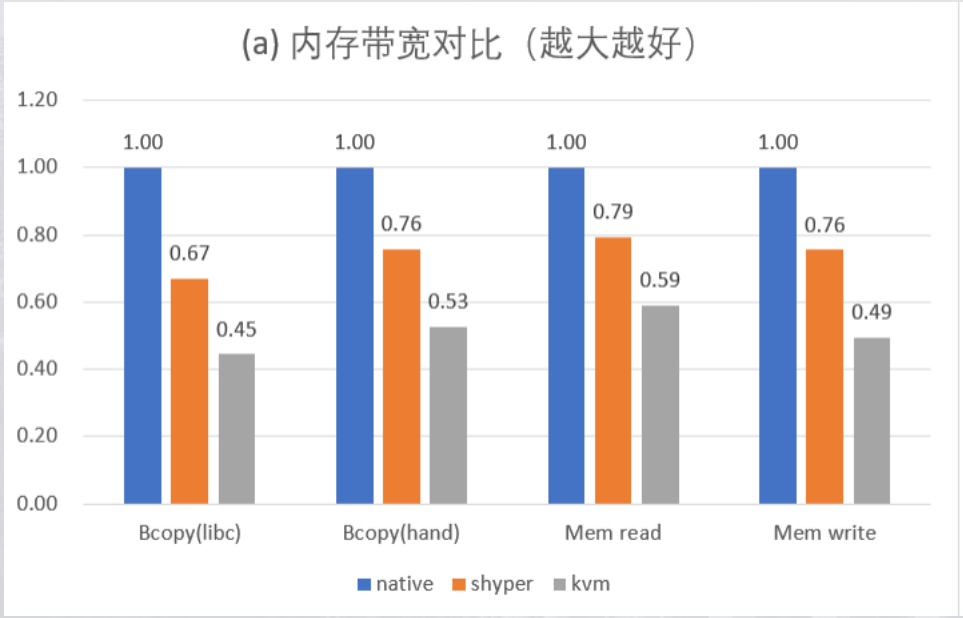
序号	来源 IP	目的 IP	能否连通
1	10.0.0.1	10.0.0.2	能
2	10.0.0.1	10.0.0.3	能
3	10.0.0.2	10.0.0.3	能
4	10.0.0.2	10.0.2.2	能
5	10.0.0.2	192.168.106.222	能
6	10.0.0.3	192.168.106.222	能

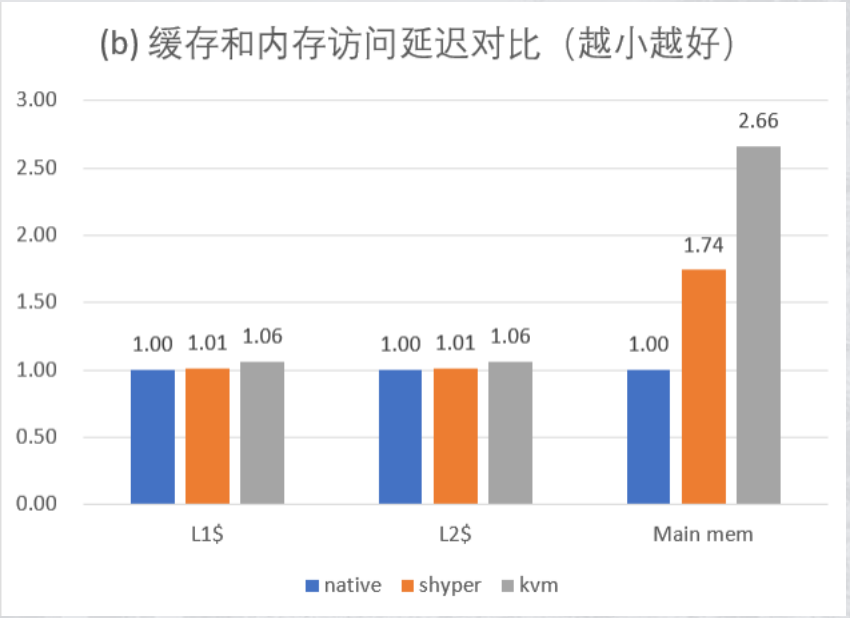
网络连通性测试



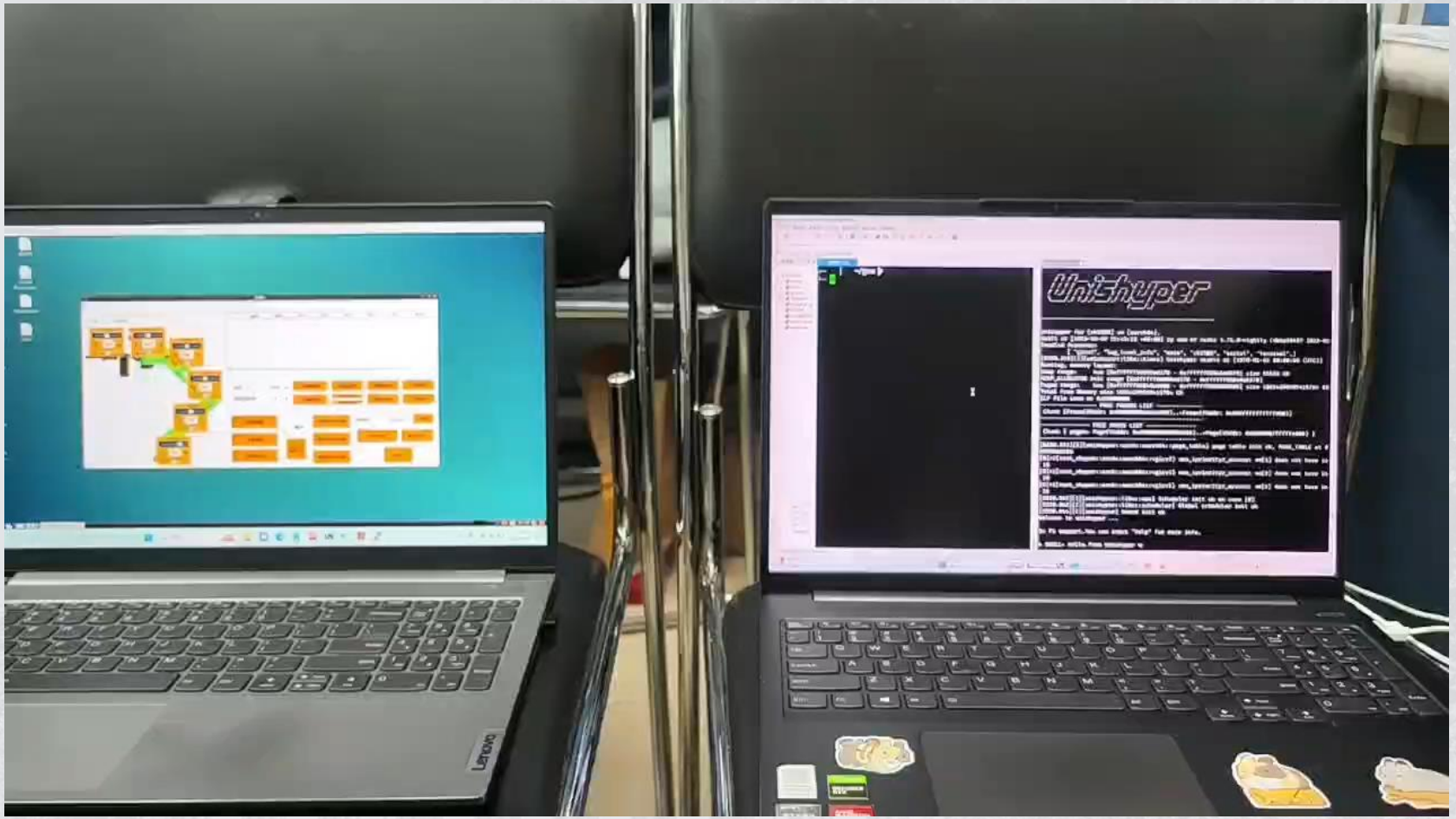
在关键指标上，Shyper相较于KVM的性能提升超过30%


MiBench测试集运行耗时（越小越好）
2.50
2.00
1.50
1.00
0.50
0.00
1.00 1.07 1.22
1.00 1.07 1.14
1.00 1.02 1.65
1.00 1.03 1.15
1.00 1.05 2.18
1.00 1.04 1.37
1.00 0.94 2.16
1.00 0.99 1.62
basicmath_small
basicmath_large
bitcnts_small
bitcnts_large
qsort_small
qsort_large
susan_small
susan_large
native
shyper
kvm


(a) 内存带宽对比（越大越好）
1.20
1.00
0.80
0.60
0.40
0.20
0.00
1.00 0.67 0.45
1.00 0.76 0.53
1.00 0.79 0.59
1.00 0.76 0.49
Bcopy(libc)
Bcopy(hand)
Mem read
Mem write
native
shyper
kvm


(b) 缓存和内存访问延迟对比（越小越好）
3.00
2.50
2.00
1.50
1.00
0.50
0.00
1.00 1.01 1.06
1.00 1.01 1.06
1.00 1.74 2.66
L1$
L2$
Main mem
native
shyper
kvm


Product Structure
3D Depth Camera
6DOF Vision Robot Arm
Intelligent Servo
6CH Far-field Microphone Array (optional)
7-inch LCD Screen (optional)
Pan-tilt Servo
Lidar (optional)
Speaker
Jetson Nano Control System
OLED Display Module
Anodized Metal Bracket
97mm Mecanum Wheel*4
USB-HUB Hub
ROS Robot Control Board
11.1V 6000mAh Lipo Battery
520 Encoder Motor*4
Pendulum Suspension Structure


Unitree
Lenovo

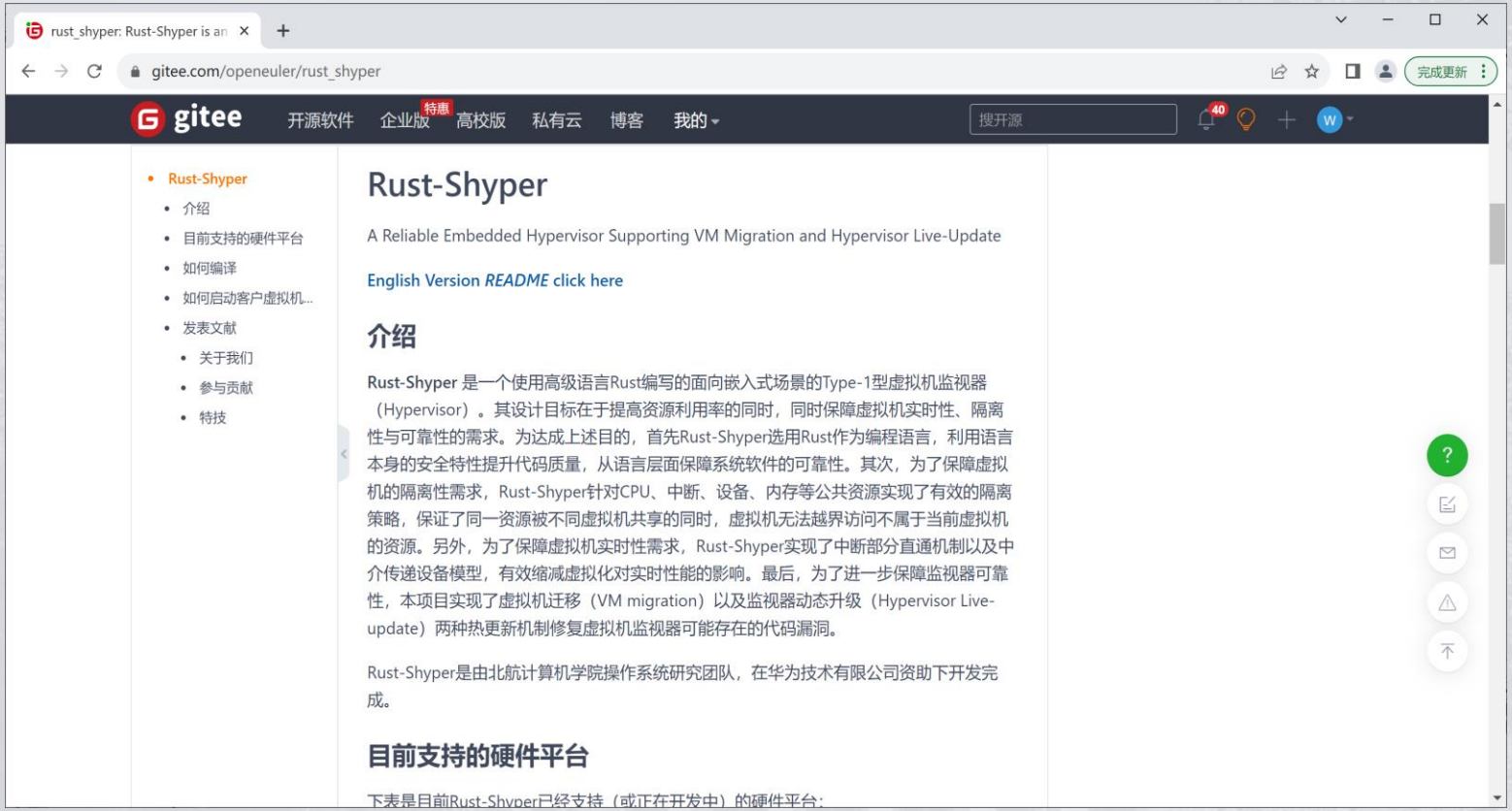
应用场景

第十一届国际智能网联汽车技术年会（CICV 2024），基于项目组的开源代码，**国家智能网联汽车创新中心、瑞芯微电子股份有限公司与国汽朴津智能科技有限公司，**共同推出虚拟机开源软件。

…该项目以Rust语言的所有权机制和生命周期等安全特性为基础，基于微虚拟机模型，并引入ArceOS开源软件的内核组件化技术和Rust-Shyper开源软件的Type 1虚拟化技术…

系统开源情况

- Rust-Shyper已经在openEuler社区开源
- 开源地址：https://gitee.com/openeuler/rust_shyper



THANKS

北京航空航天大學
BEIHANG UNIVERSITY