Hardware-Assisted Virtualization

温瑞林

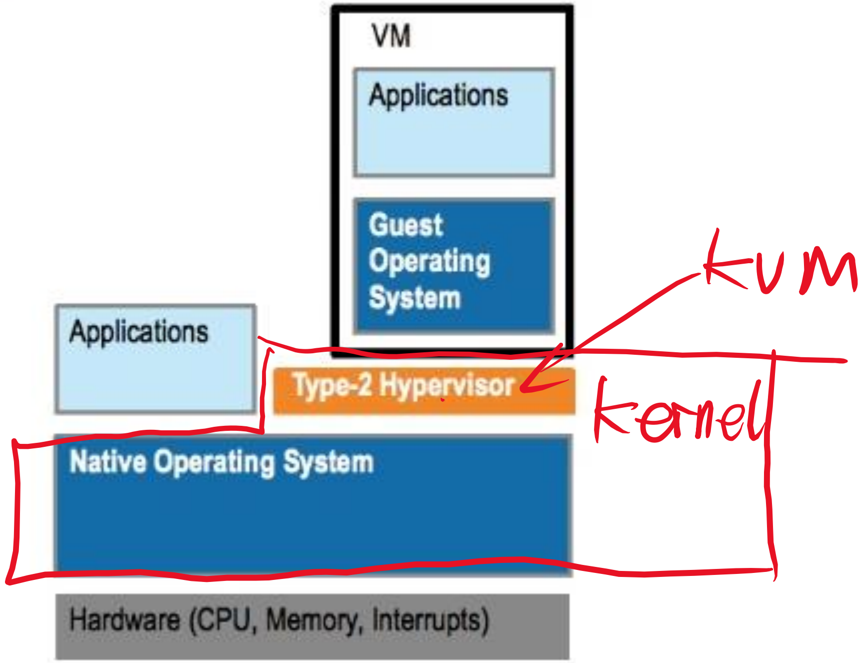
CONTENTS

1. What’s KVM?
2. Hardware Virtualization and Implementation of KVM
3. Dune: Safe User-level Access to Privileged CPU Features
4. Arrakis: The Operating System is the Control Plane
5. Summary

What’s KVM? /01

What's KVM?

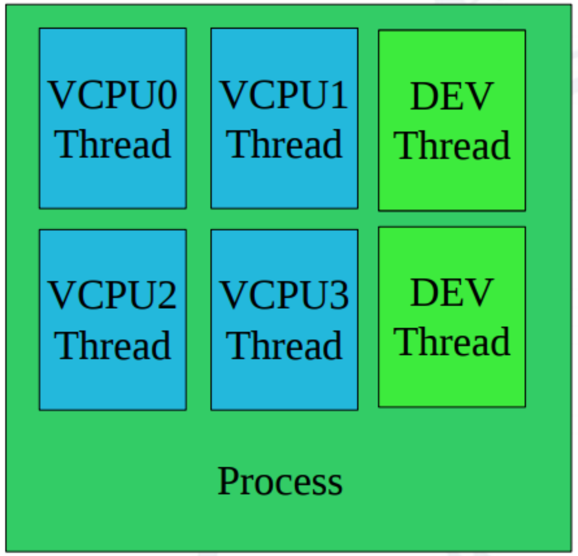
Type 2 Hypervisor



Abstraction Model

Design

1. One process per VM
2. One thread per-VCPU
3. Device models run concurrent in VCPU thread
4. Long running operations run in additional device thread



VCPU0 Thread

VCPU1 Thread

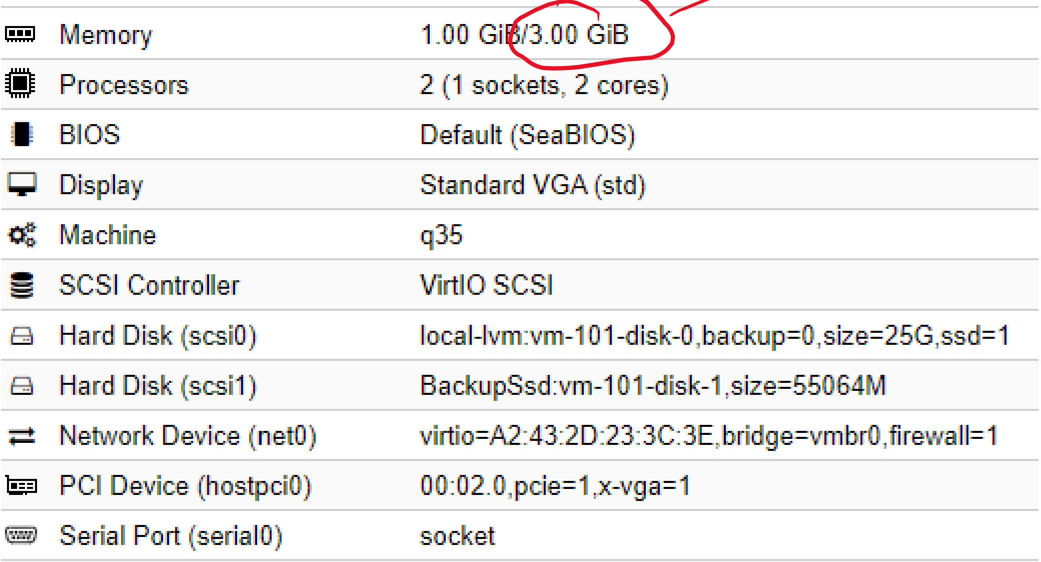
DEV Thread

VCPU2 Thread

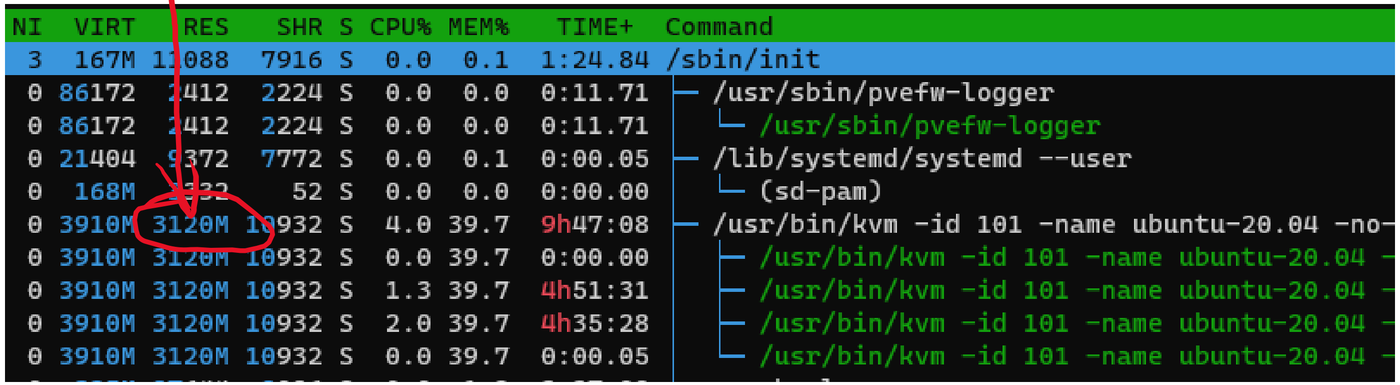
VCPU3 Thread

DEV Thread

Process



Memory	1.00 GiB/3.00 GiB
Processors	2 (1 sockets, 2 cores)
BIOS	Default (SeaBIOS)
Display	Standard VGA (std)
Machine	q35
SCSI Controller	VirtIO SCSI
Hard Disk (scsi0)	local-lvm:vm-101-disk-0,backup=0,size=25G,ssd=1
Hard Disk (scsi1)	BackupSsd:vm-101-disk-1,size=55064M
Network Device (net0)	virtio=A2:43:2D:23:3C:3E,bridge=vmbr0,firewall=1
PCI Device (hostpci0)	00:02.0,pcie=1,x-vga=1
Serial Port (serial0)	socket

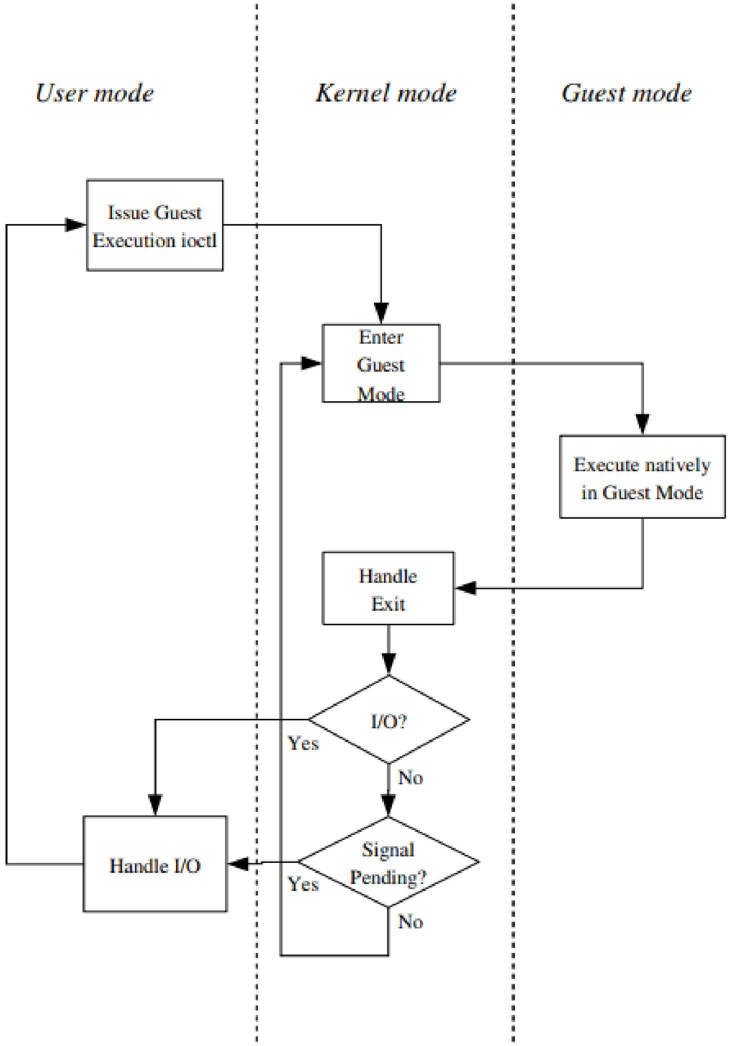


```
NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 3  167M 11088  7916 S  0.0  0.1  1:24.84 /sbin/init
 0 86172  2412  2224 S  0.0  0.0  0:11.71 ├─ /usr/sbin/pvefw-logger
 0 86172  2412  2224 S  0.0  0.0  0:11.71 │  └─ /usr/sbin/pvefw-logger
 0 21404  9372  7772 S  0.0  0.1  0:00.05 ├─ /lib/systemd/systemd --user
 0  168M  1332    52 S  0.0  0.0  0:00.00 │  └─ (sd-pam)
 0 3910M 3120M 10932 S  4.0 39.7  9h47:08 ├─ /usr/bin/kvm -id 101 -name ubuntu-20.04 -no-
 0 3910M 3120M 10932 S  0.0 39.7  0:00.00 │  ├─ /usr/bin/kvm -id 101 -name ubuntu-20.04 -
 0 3910M 3120M 10932 S  1.3 39.7  4h51:31 │  ├─ /usr/bin/kvm -id 101 -name ubuntu-20.04 -
 0 3910M 3120M 10932 S  2.0 39.7  4h35:28 │  ├─ /usr/bin/kvm -id 101 -name ubuntu-20.04 -
 0 3910M 3120M 10932 S  0.0 39.7  0:00.05 │  └─ /usr/bin/kvm -id 101 -name ubuntu-20.04 -
```

Programming Model

1. Event-Loop like model
2. Hide the detail of cross-platform hardware features

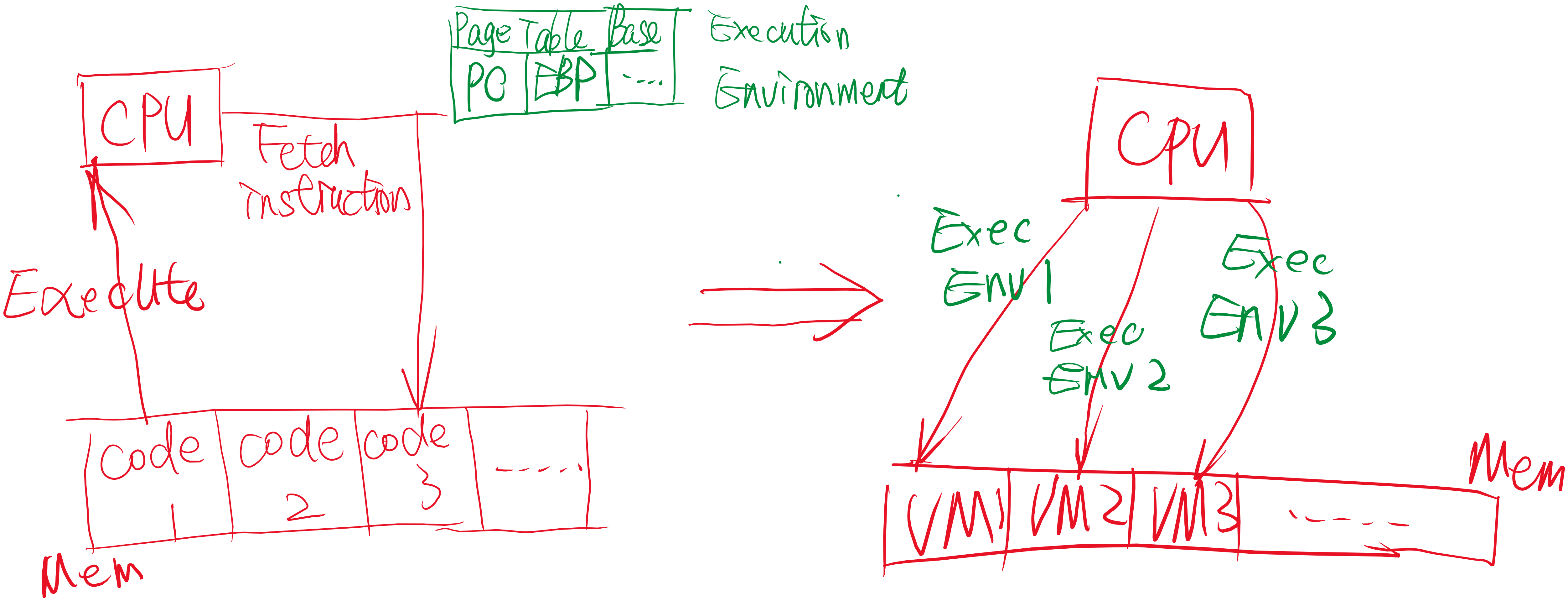
```
int runsz = ioctl(kvm_fd, KVM_GET_VCPU_MMAP_SIZE, 0);
struct kvm_run *run = (struct kvm_run *)
          mmap(     NULL, runsz,
                    PROT_READ | PROT_WRITE,
                    MAP_SHARED, vcpu_fd, 0);

for (;;) {
          ioctl(vcpu_fd, KVM_RUN, 0);
          switch (run->exit_reason) {
          case KVM_EXIT_IO:
                    printf(……);
                    break;
          case KVM_EXIT_SHUTDOWN:
                    return;
          }
}
```



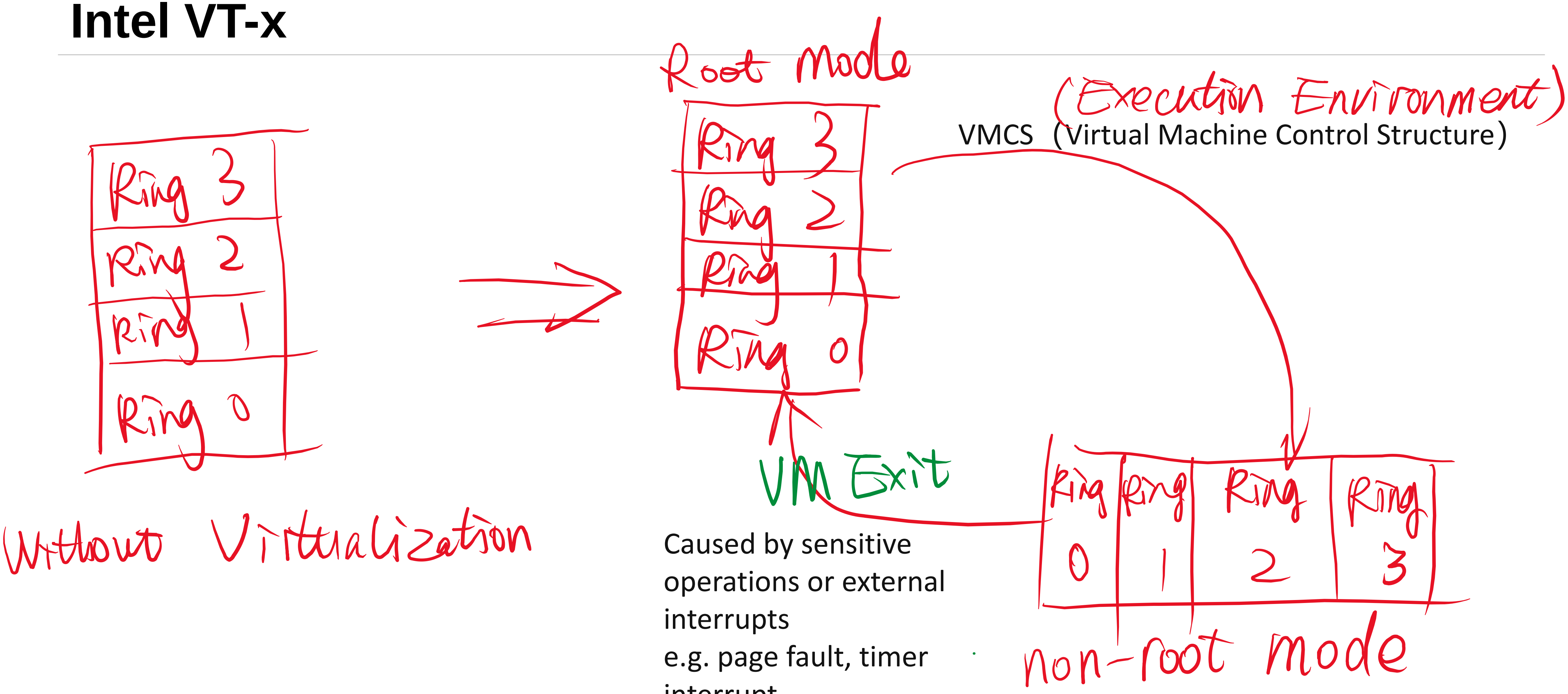
Hardware Virtualization and Implementation of KVM

/02

Hardware Virtualization and Implementation of KVM



Intel VT-x



VMCS (Virtual Machine Control Structure)

Caused by sensitive operations or external interrupts
e.g. page fault, timer interrupt

Intel VT-x

Root mode

Ring 3

Ring 2

Ring 1

Ring 0

(Execution Environment)

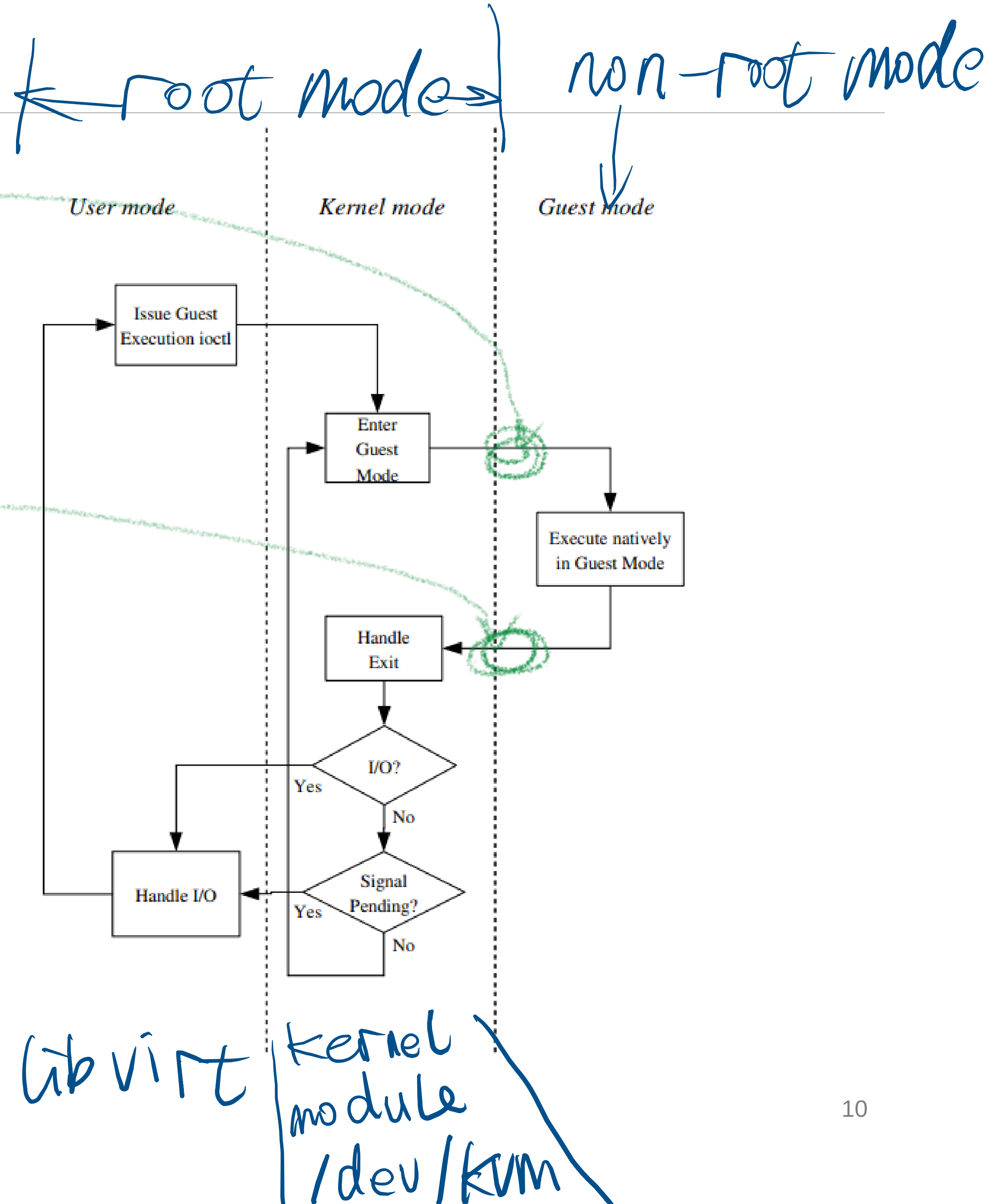
VMCS (Virtual Machine Control Structure)

VM Exit

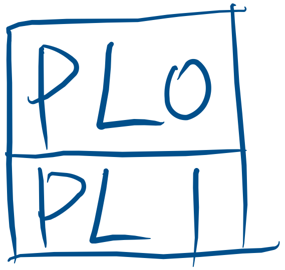
Ring 0 | Ring 1 | Ring 2 | Ring 3

non-root mode

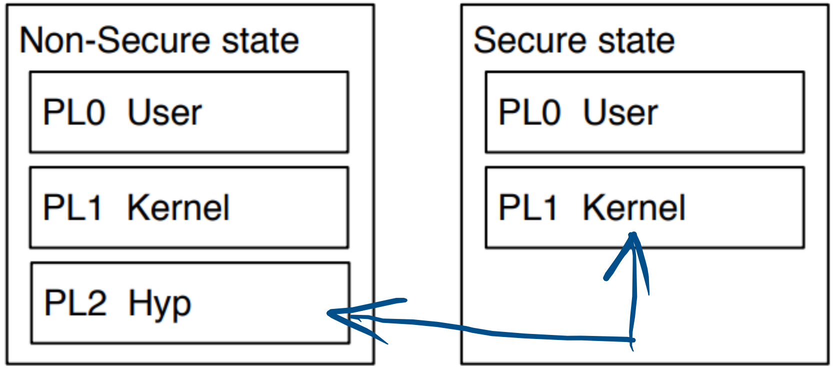
Caused by sensitive operations or external interrupts
e.g. page fault, timer interrupt



ARM



ARM without virtualization



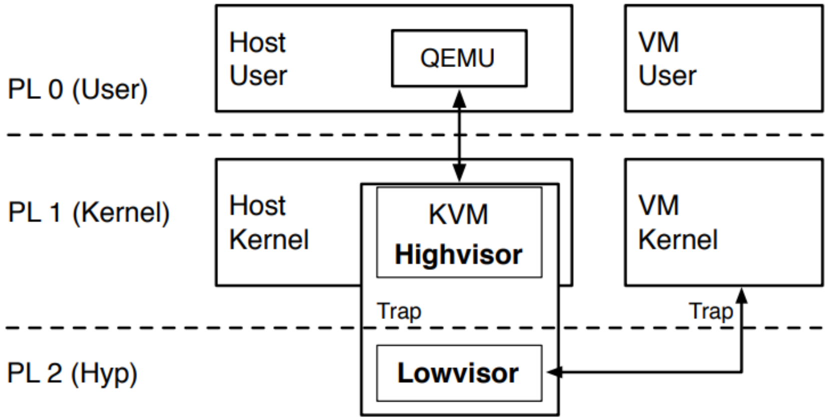
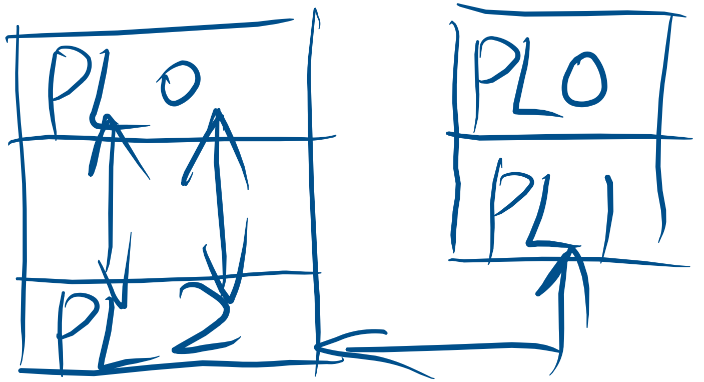
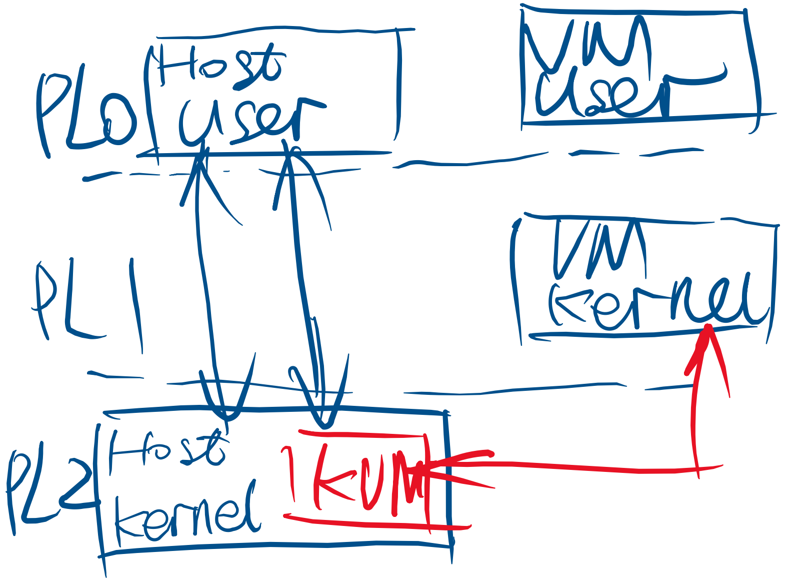


Figure 2: KVM/ARM System Architecture

Arm v7 & Arm v8



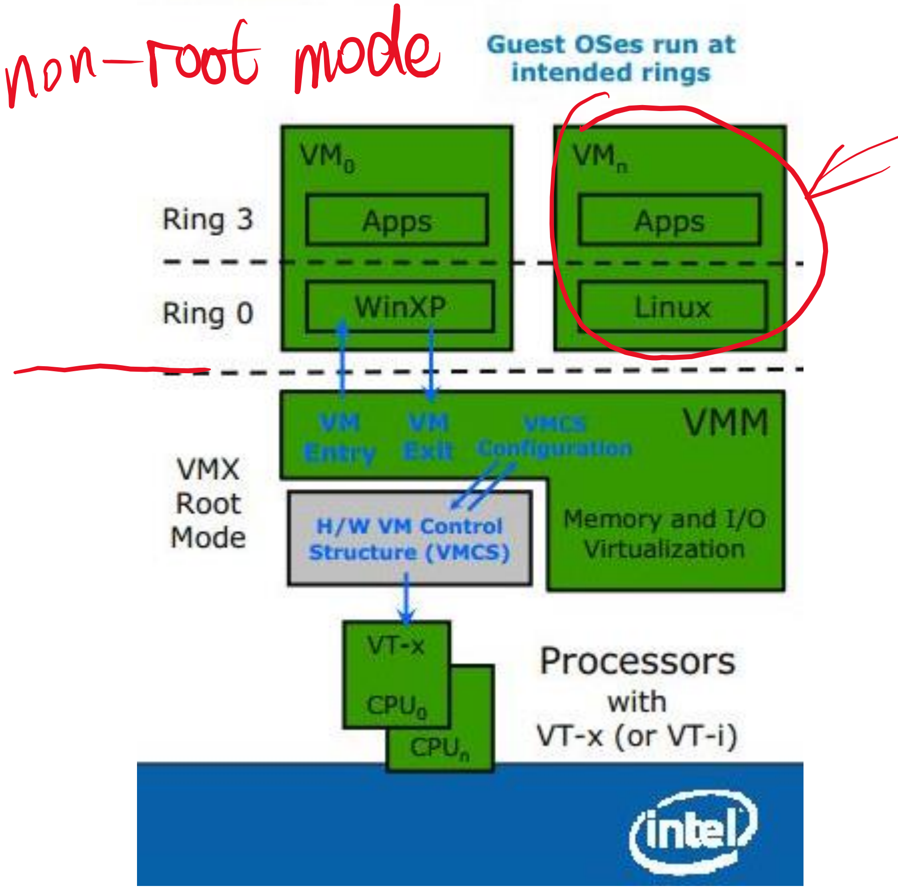


Arm v8.1 VHE

Dune: Safe User-level Access to Privileged CPU Features

/03

Idea



Can we replace it with a host process?

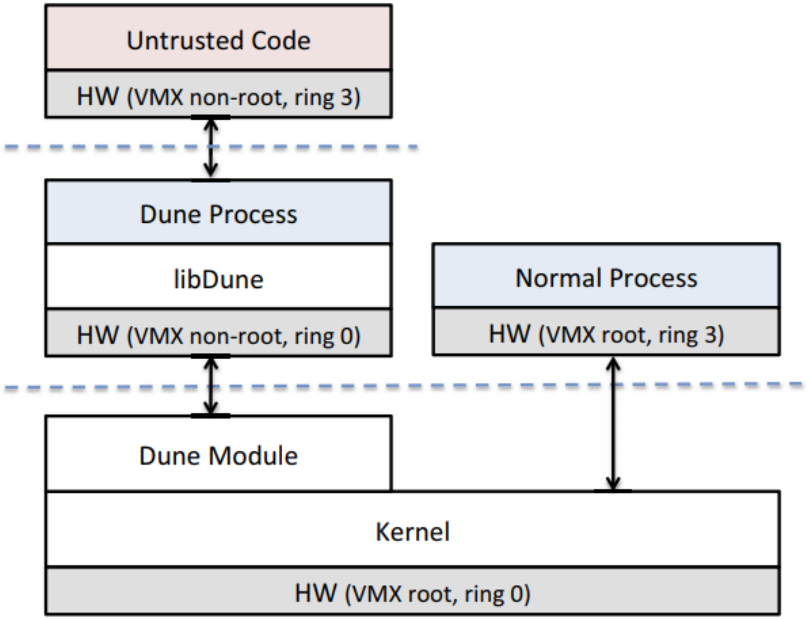
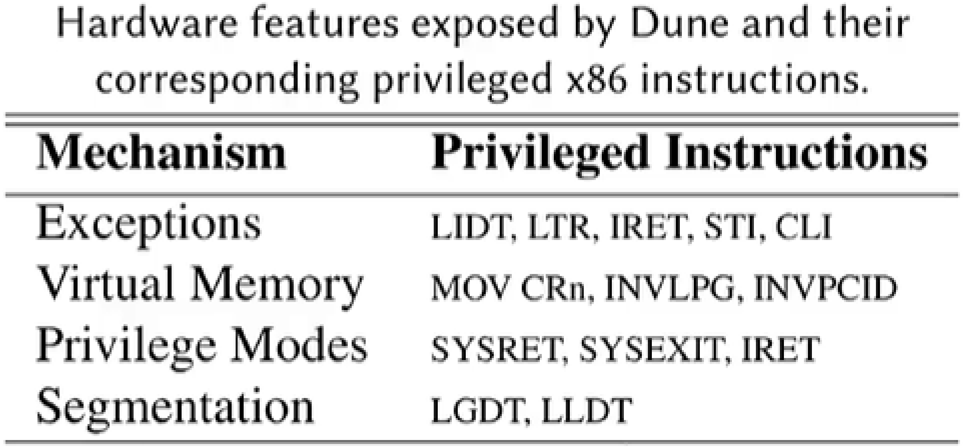


Figure 1: The Dune system architecture.

Benefits



Hardware features exposed by Dune and their corresponding privileged x86 instructions.

Mechanism	Privileged Instructions
Exceptions	LIDT, LTR, IRET, STI, CLI
Virtual Memory	MOV CRn, INVLPG, INVPCID
Privilege Modes	SYSRET, SYSEXIT, IRET
Segmentation	LGDT, LLDT

Allow some applications to improve their performance by exploiting the exposed hardware features

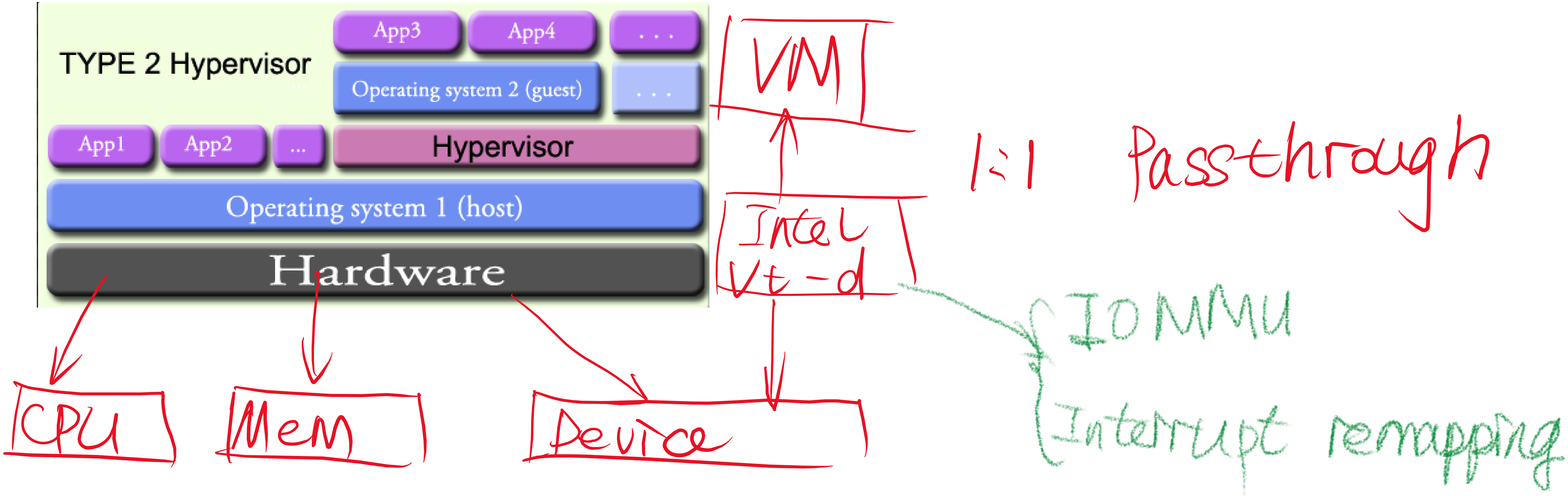
Application?

1. Faster GC
   1. No syscall or IRQ overhead
   2. More exploitable information(Dirty bits…)
   3. TLB flush batching
2. Sandbox
   1. Faster context switching
   2. More flexible filtering policies on sensitive operations without kernel modification
3. Trace
   1. A better way to bypass anti-debugging mechanism?

Arrakis: The Operating System is the Control Plane

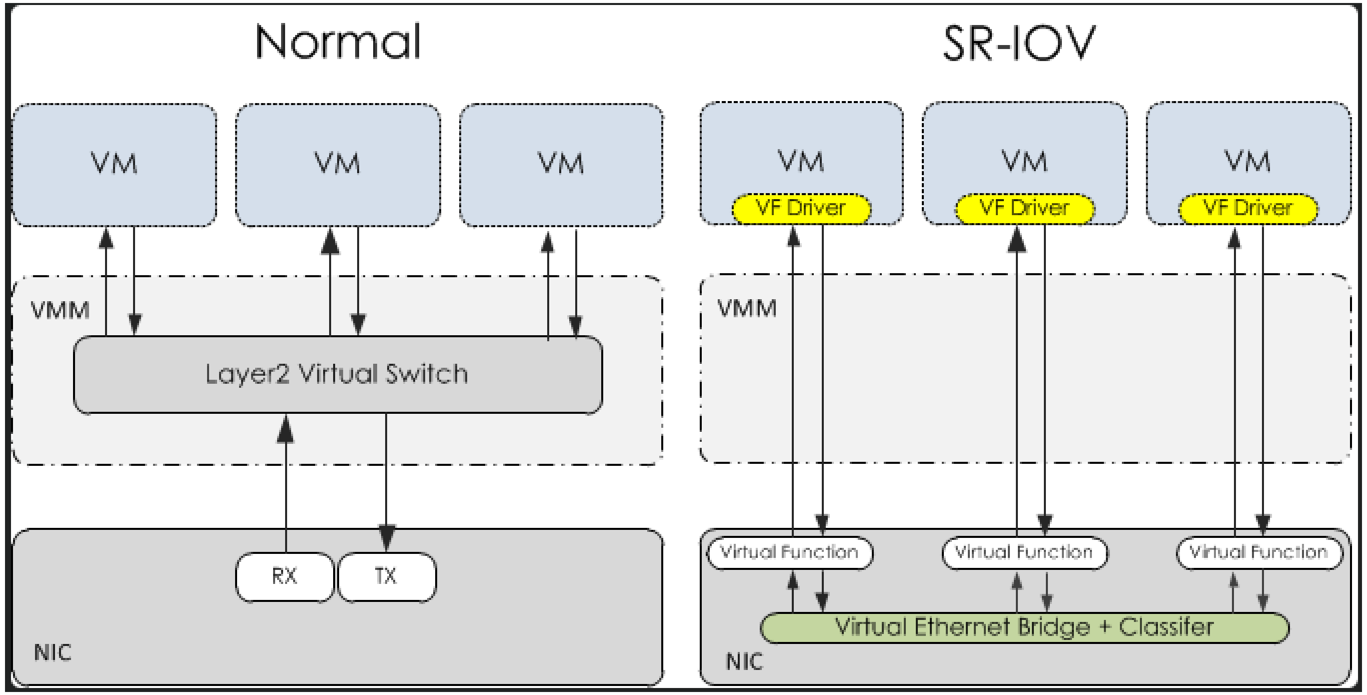
/04

Idea



But! No MULTIPLEXING!

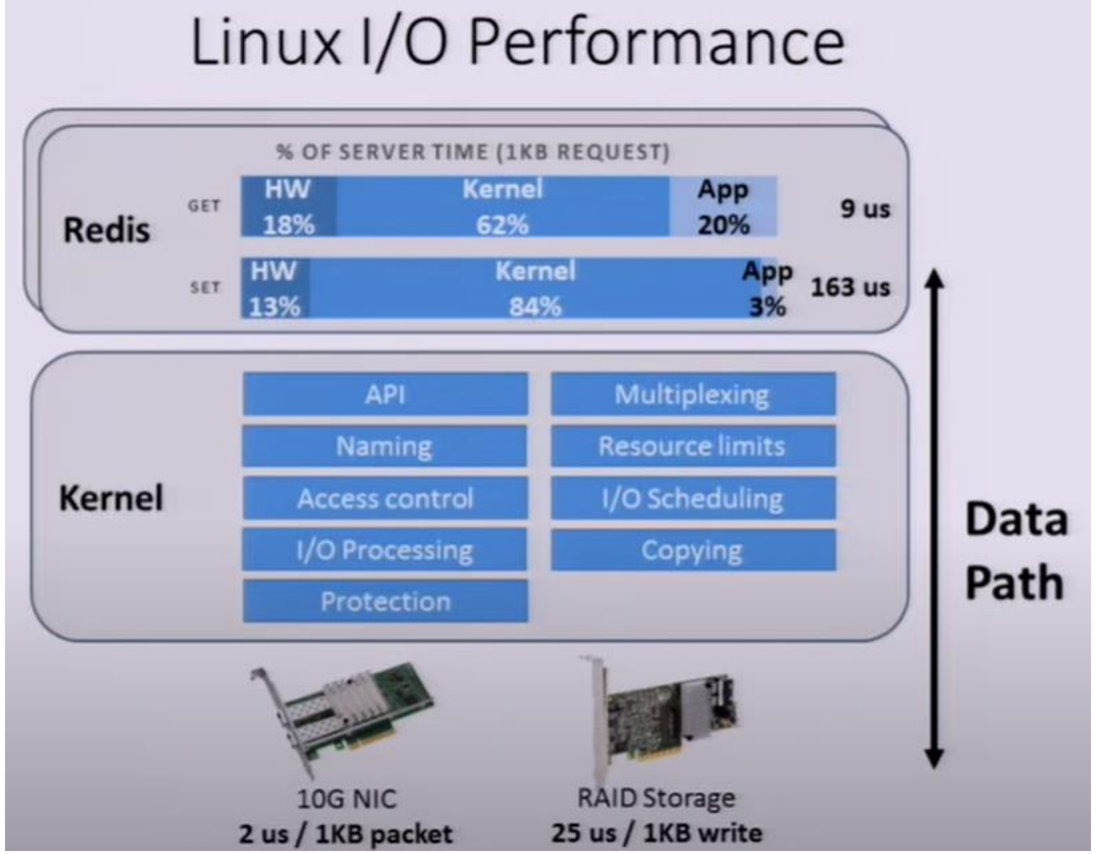
Idea



With SR-IOV, all hardware resources are able to be multiplexed at the hardware level.

So can we reorganize the OS now?

Idea



Linux I/O Performance

% OF SERVER TIME (1KB REQUEST)

Redis	HW	Kernel	App	
GET	18%	62%	20%	9 us
SET	13%	84%	3%	163 us

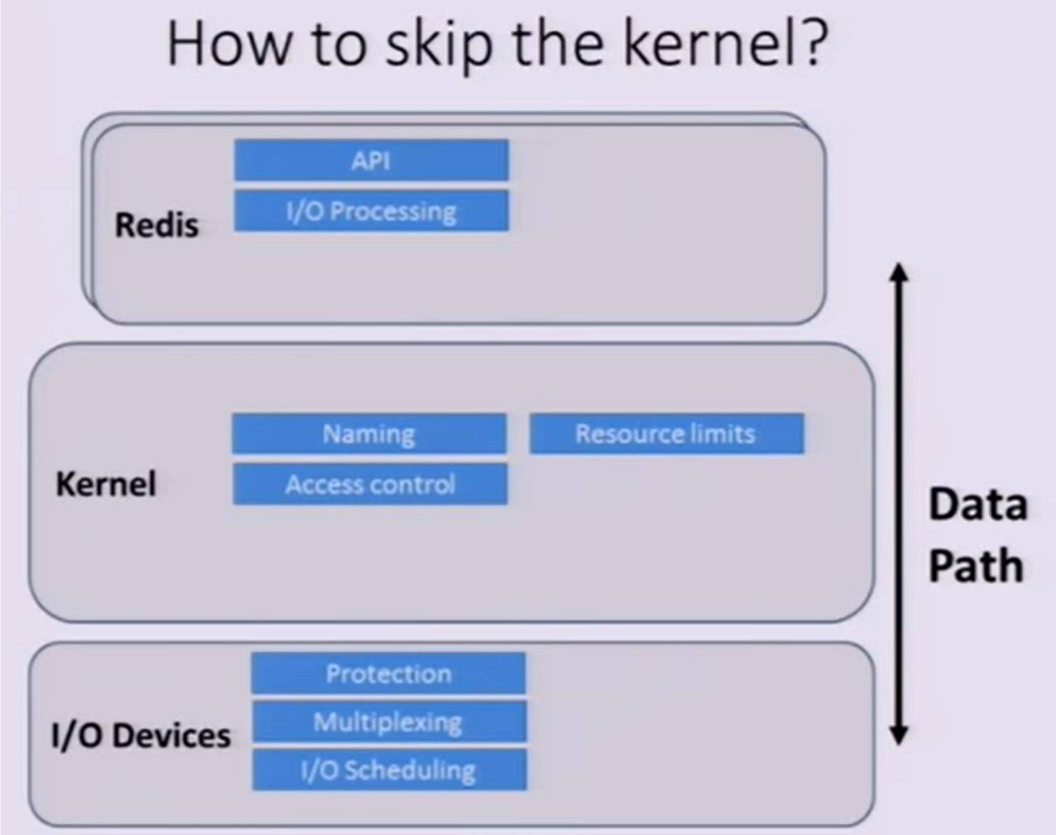
Kernel

API
Multiplexing
Naming
Resource limits
Access control
I/O Scheduling
I/O Processing
Copying
Protection

10G NIC
2 us / 1KB packet

RAID Storage
25 us / 1KB write

Data Path



How to skip the kernel?

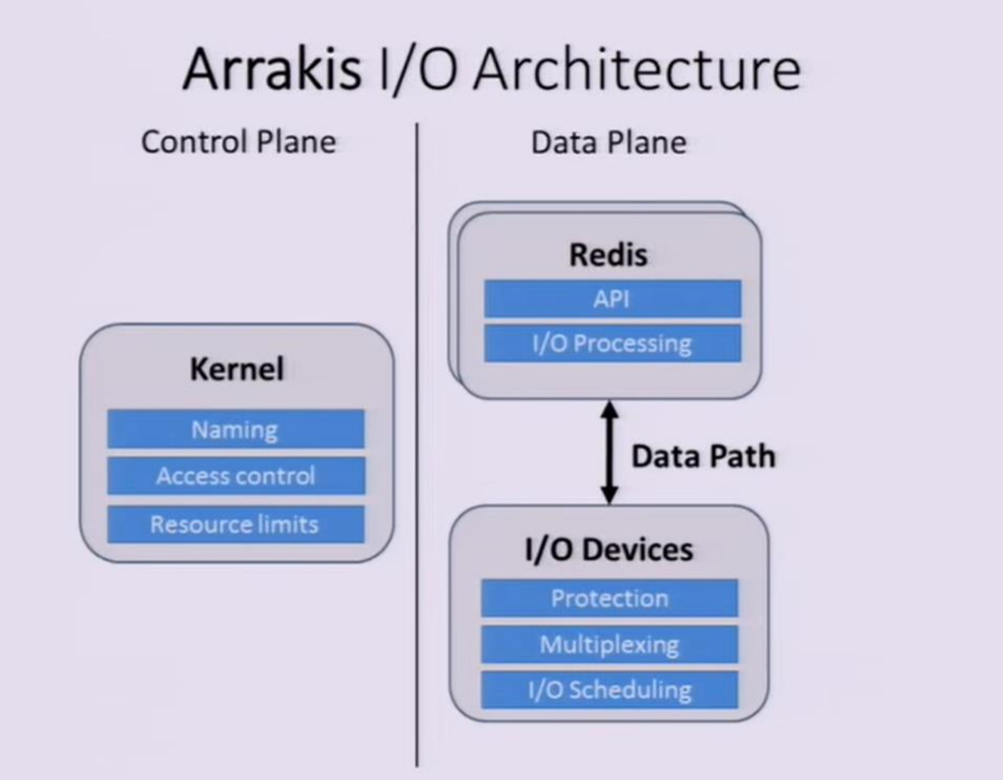
Redis
API
I/O Processing

Kernel
Naming
Resource limits
Access control

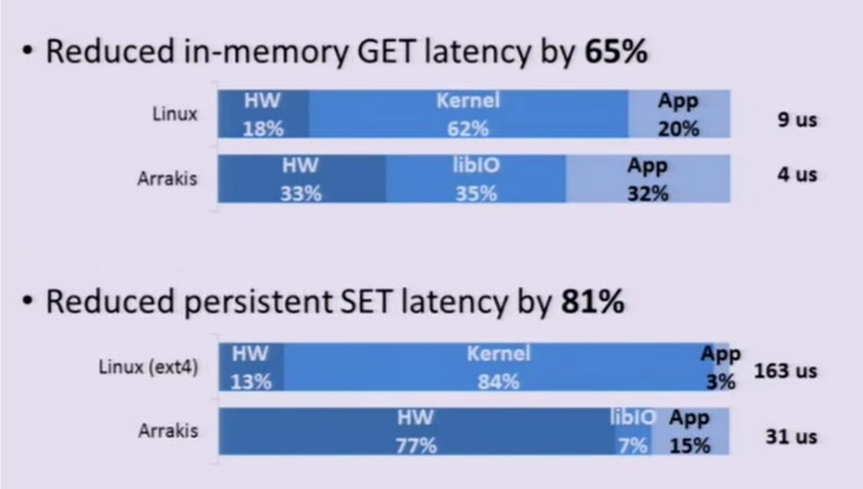
I/O Devices
Protection
Multiplexing
I/O Scheduling

Data Path

Design and Result



Arrakis I/O Architecture



- Reduced in-memory GET latency by **65%**

- Reduced persistent SET latency by **81%**

DPDK? Openwrt? ⇒ Arrakis provides a generic architecture

Summary /05

Summary

1. KVM and Dune - How to integrate with the existing system gracefully
2. Arrakis – How to reorganize the existing system gracefully

Thanks