Kernel Protection Using Hardware-Based Virtualization

Jun Nakajima and Sainath Grandhi

Intel Software

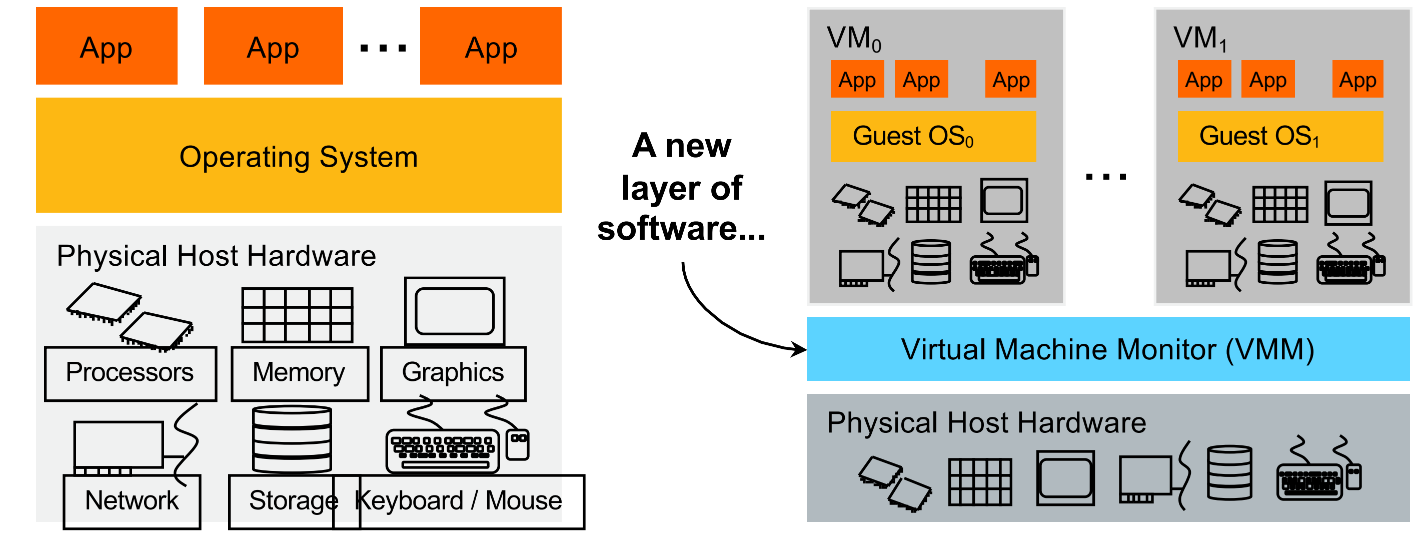
Intel OpenSource TECHNOLOGY CENTER

Legal Disclaimer

Agenda

- Hardware-Based Virtualization
- Monitoring/Protecting the Kernel in Virtualization
- Policy and Incident Handling
- Architecture and Implementation
- VM and Bare Metal
- Beyond Kernel Protection

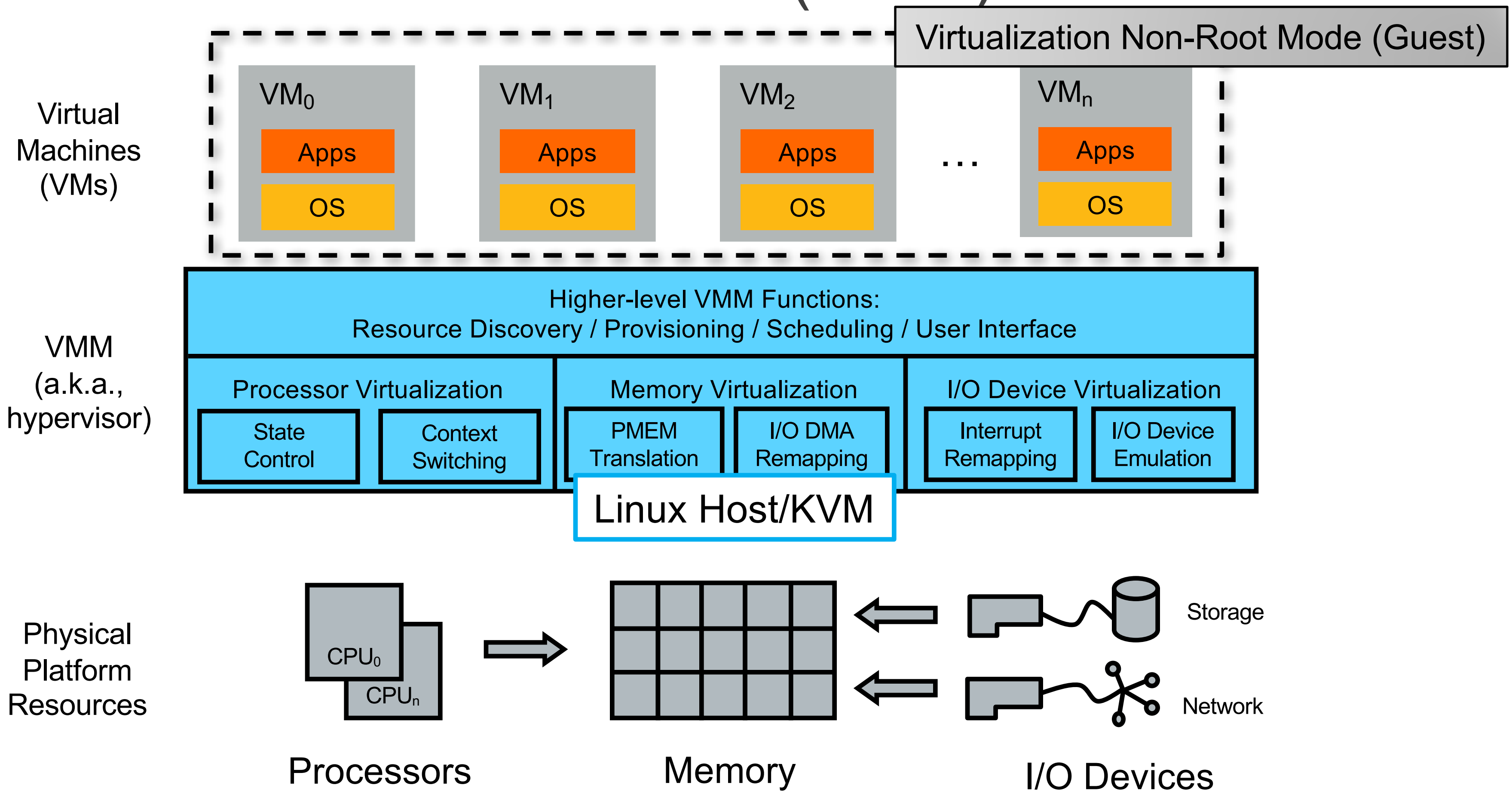
Hardware Virtual Machines (VMs)



Without VMs: Single OS owns all hardware resources

With VMs: Multiple OSes share hardware resources

Virtual Machine Monitor (VMM)

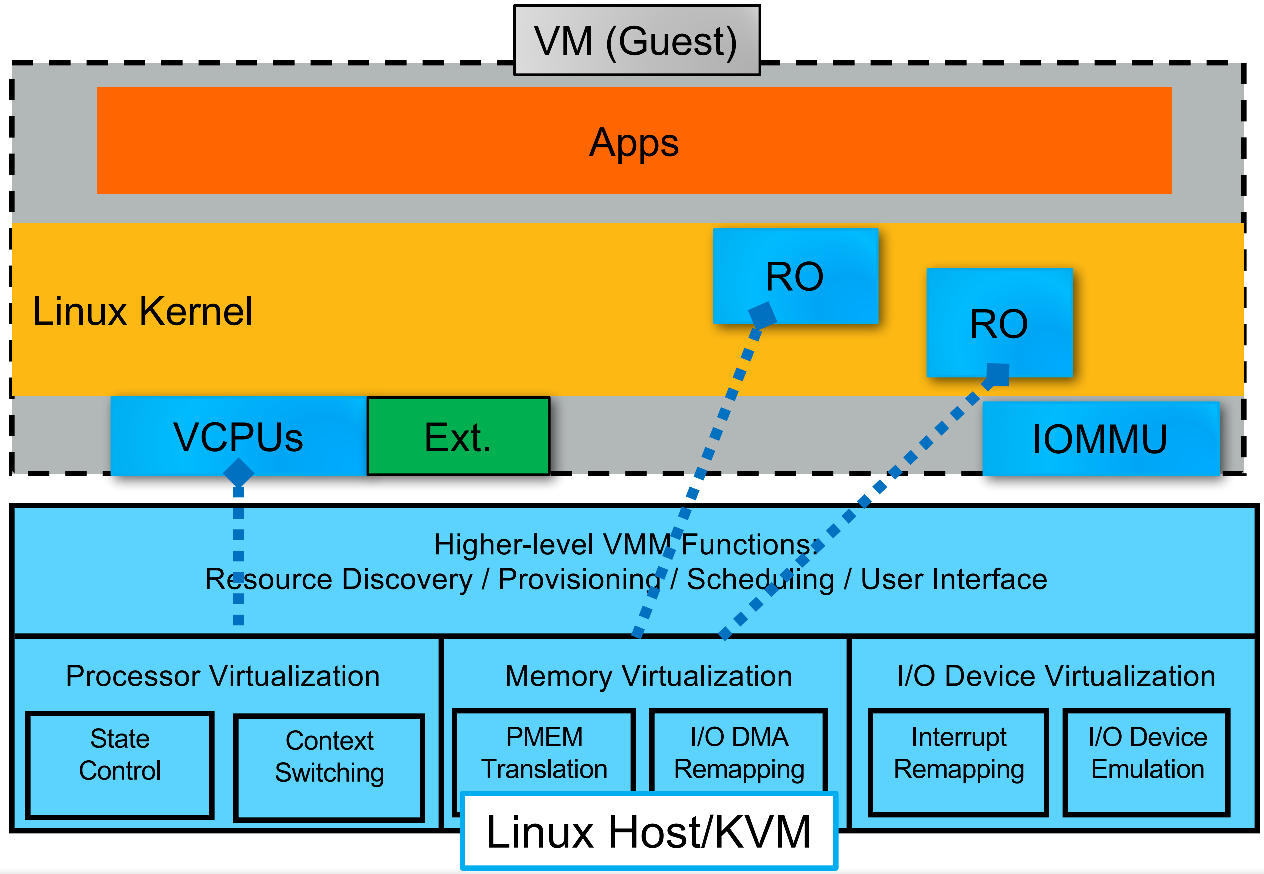


Overview of Kernel Protection

Memory:
- Monitoring
- Write-protection (RO)

Processor (VCPU):
- CPU control monitoring/locking
- Extensions for security

IOMMU:
- Monitoring
- Write-Protection



Benefits of Virtualization-Based Kernel Protection

More monitoring and isolation capabilities in virtualization than in native:

- Monitoring, isolation, and protection – Hypervisor as “Ring -1” or Virtualization Root Mode
- Security feature extensions to the CPUs so that the kernel can harden itself

No or minimal modifications to guest Linux kernel:

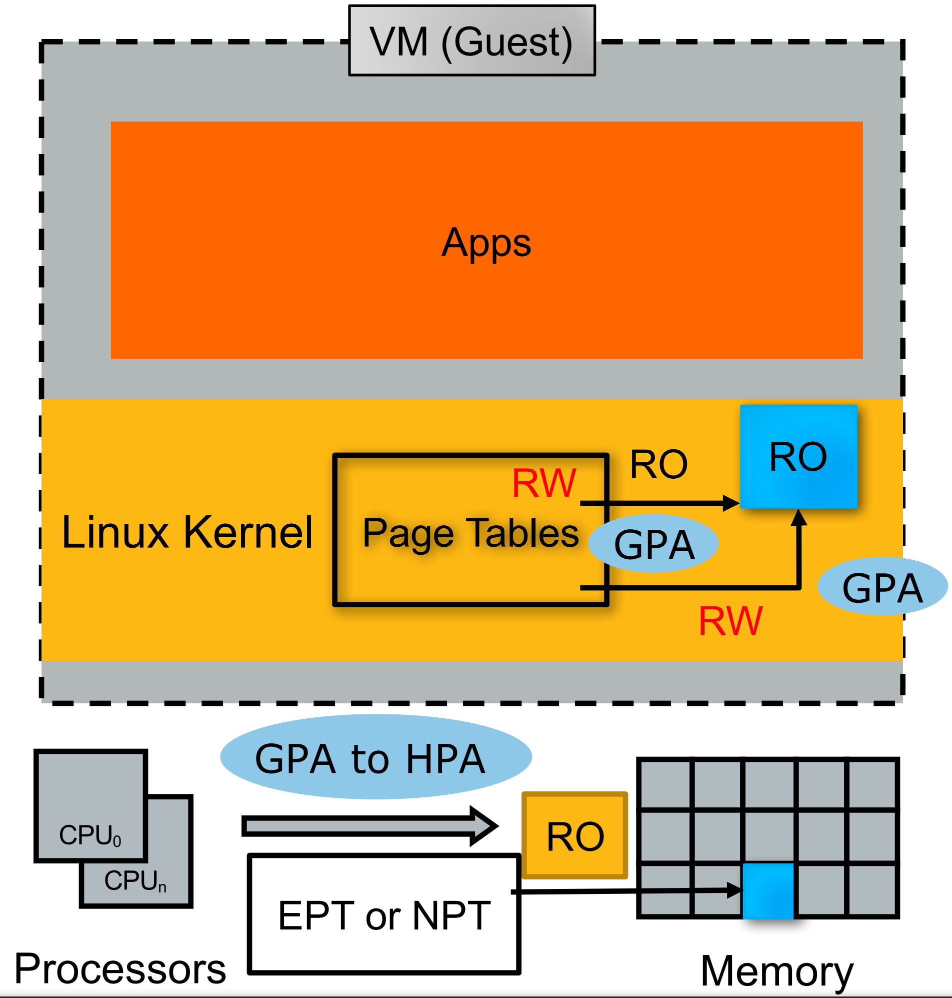
- Can be implemented inside the hypervisor (e.g. KVM)
- Hot patches

Applicable to bare metal kernel:

- Bare-metal Linux can de-privilege itself to become Virtualization Non-Root Mode
- Additional protection when running bare-metal containers, HPC without overhead

Kernel Memory Protection

- Kernel can write-protect its own code or data by RO (Read-Only) permission for the page
- But the page can be modified by:
  - Changing the permission, or
  - Establishing different mapping with RW permission
- H/W-based virtualization can add enforcement by:
  - RO permission for GPA* to HPA translation
  - VM exit upon attempt to write the page



*:GPA: Guest Physical Address, HPA: Host Physical Address

Kernel Memory Protection (cont.)

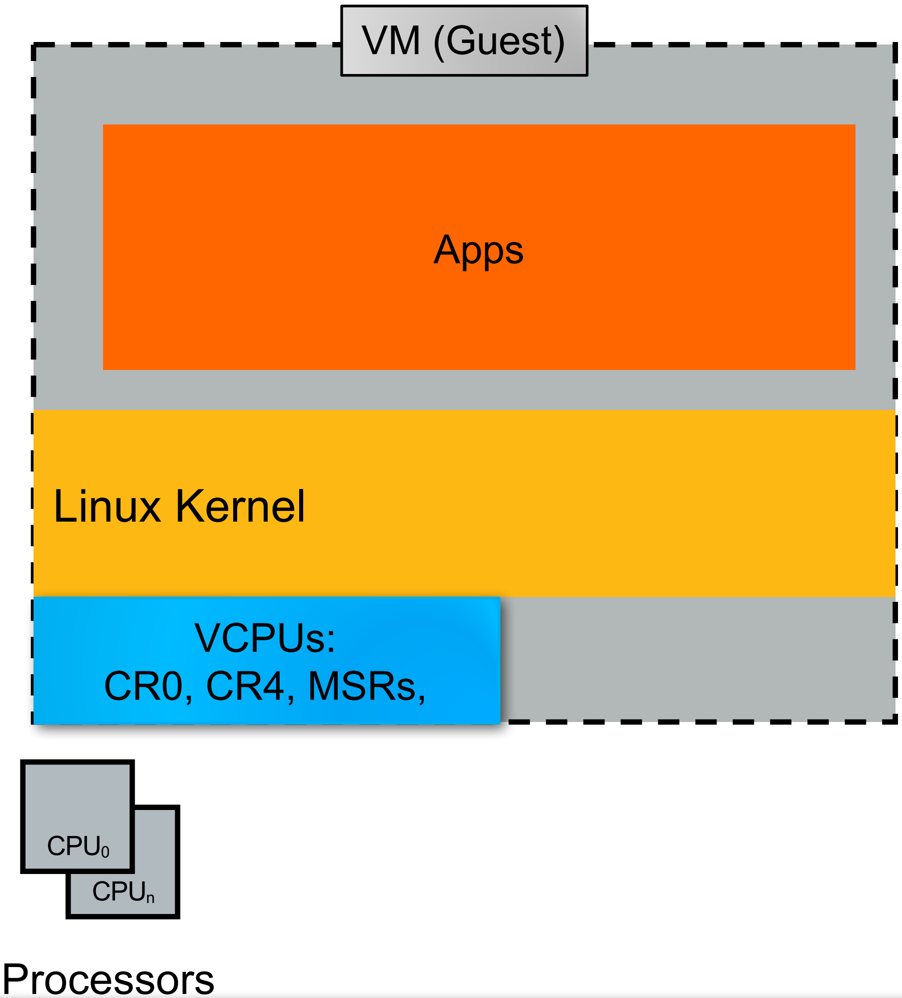
Examples of code/data to monitor or protect:

- Kernel code and page tables entries for such mappings
- Syscall table
- IDT (Interrupt Descriptor Table)
- …
- Various data structures, e.g. kernel data declared as “`const` ...”

Protecting CPU State Control

Linux kernel does not change the setting for CPU control at runtime:

- Control Registers
  - CR0 – PG, CD, WP, PE,
  - CR4 – UMIP, VMXE, SMXE, SMEP, SMAP, PKE,
- MSRs
  - EFER
  - PAT
  - MISC_ENBLE



Security Feature Extensions to CPUs

- Implement new or future H/W security features in virtualization so that the current or older CPUs can take advantage of them
  - Example: UMIP (User-Mode Instruction Prevention) – can be mostly emulated by the exiting H/W virtualization feature
- Para-virtualization
  - Requires modifications to the kernel

Protecting IOMMU State Control

Setup once and never modified:

- Root Table Address
- Invalidation Queue Address
- Interrupt Remapping Table Address

Feature Enabling:

- DMA Translation
- Interrupt Remapping
- Queued Invalidation

Policy and Incident Handling

Monitor and protect specific kernel data/code and system resources (assets):

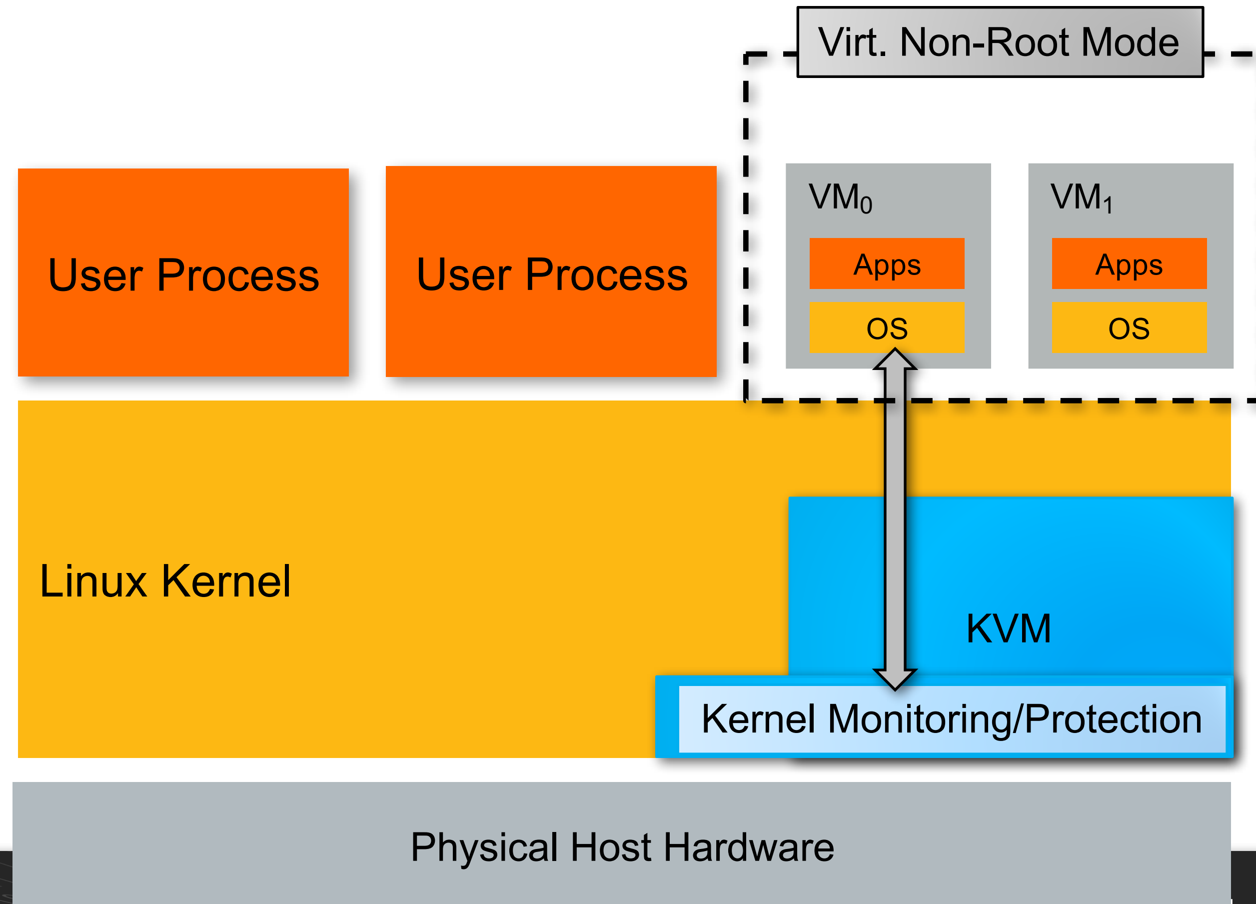
```
<Which asset to monitor>, <Permission>, <Action upon Permission Violation>
```

- <Which asset to monitor> := Bits of a control register, MSRs, memory pages, or I/O ports,
- <Permission> := RO (Read-Only), XO (Execution Only), NA (No Access Allowed)
- <Action(s)> := Omit the attempt and log, Allow the attempt and log,

Architecture Overview (KVM Guests Only)

Extend KVM:

- Kernel Monitoring/Protection



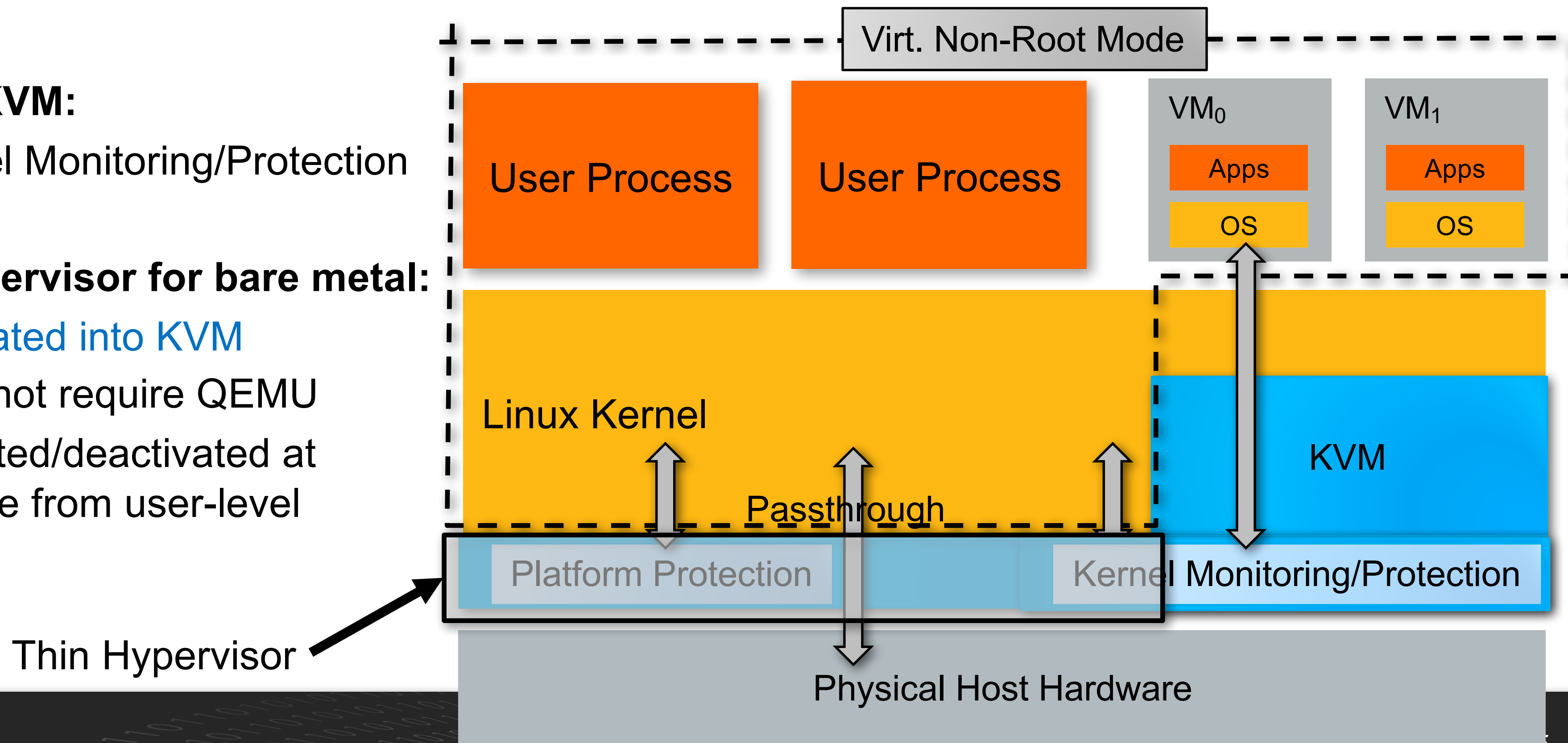
Architecture Overview (Host and KVM Guests)

Extend KVM:

- Kernel Monitoring/Protection

Thin Hypervisor for bare metal:

- Integrated into KVM
- Does not require QEMU
- Activated/deactivated at runtime from user-level



Bare-Metal Linux in Virtualization Non-Root Mode

Bare-metal Linux can run like the native with Virtualization Non-Root Mode enabled:

- Pass-through
  - I/O devices, interrupt controllers, timers, power management, – No VM exits (Done by “VM Exit Control”)
- Identity mapping (+ protection):
  - EPT (Extended page tables) – EPT(GPA) == HPA
  - Use the bare metal kernel – No additional memory for virtualization (except EPT)
- Platform protection
  - Prevent access to platform resources – Platform-specific MSRs, ports, I/O spaces

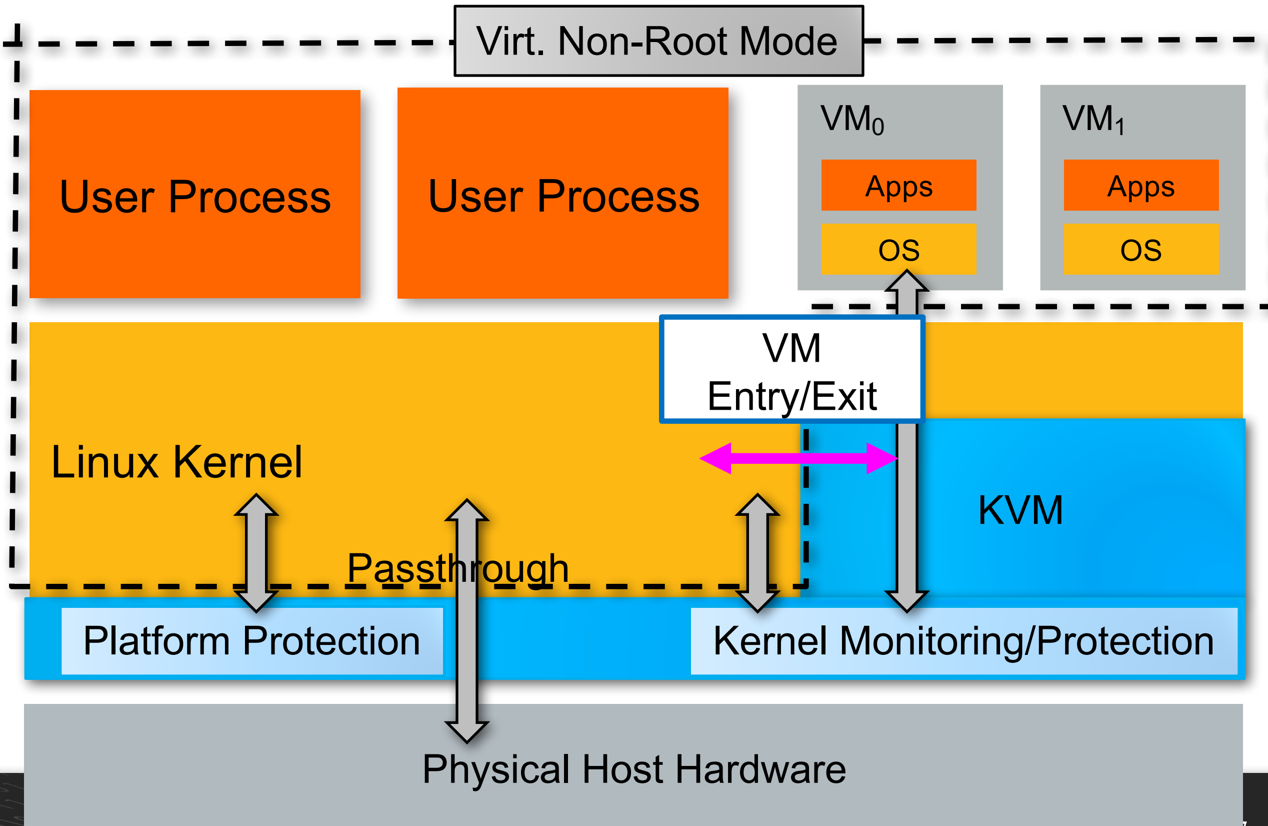
Switching from Virt. Non-Root to KVM (Virt. Root)

Go back to Virtualization Root Mode to run guests on top of KVM:

- Avoid nested virtualization

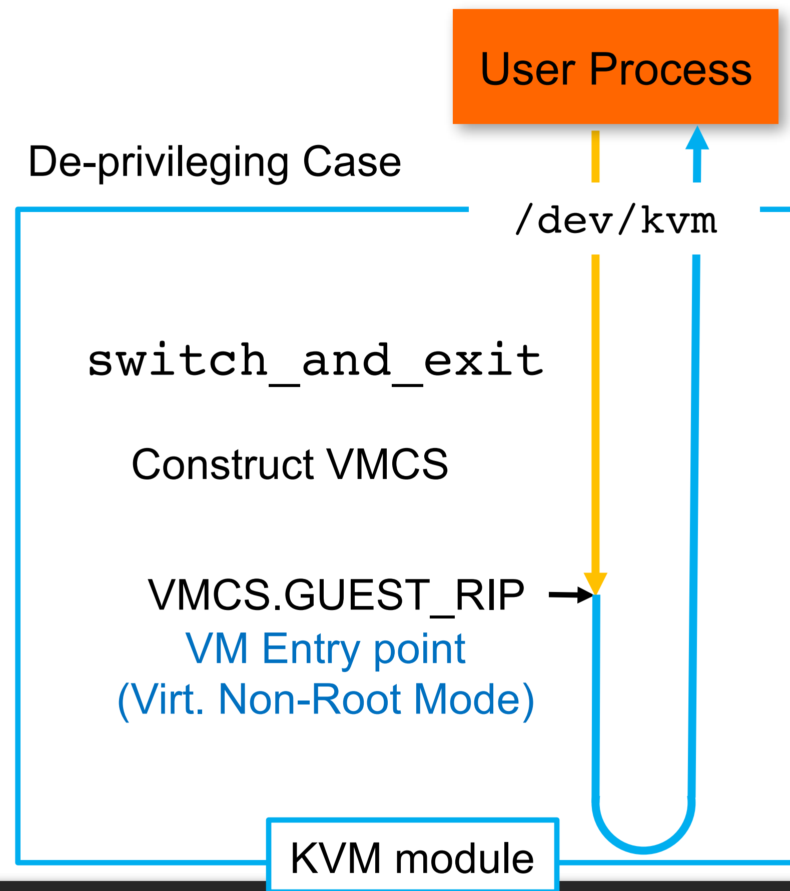
Current Implementation:

1. VM Exit in the kernel (e.g `VMXOFF` instruction)
2. VM Exit handler for the bare-metal kernel
3. IRET to the next instruction that caused the VM exit (one after `VMXOFF`)



Prototype Implementation of “Non-Root Mode Bare-Metal Linux”

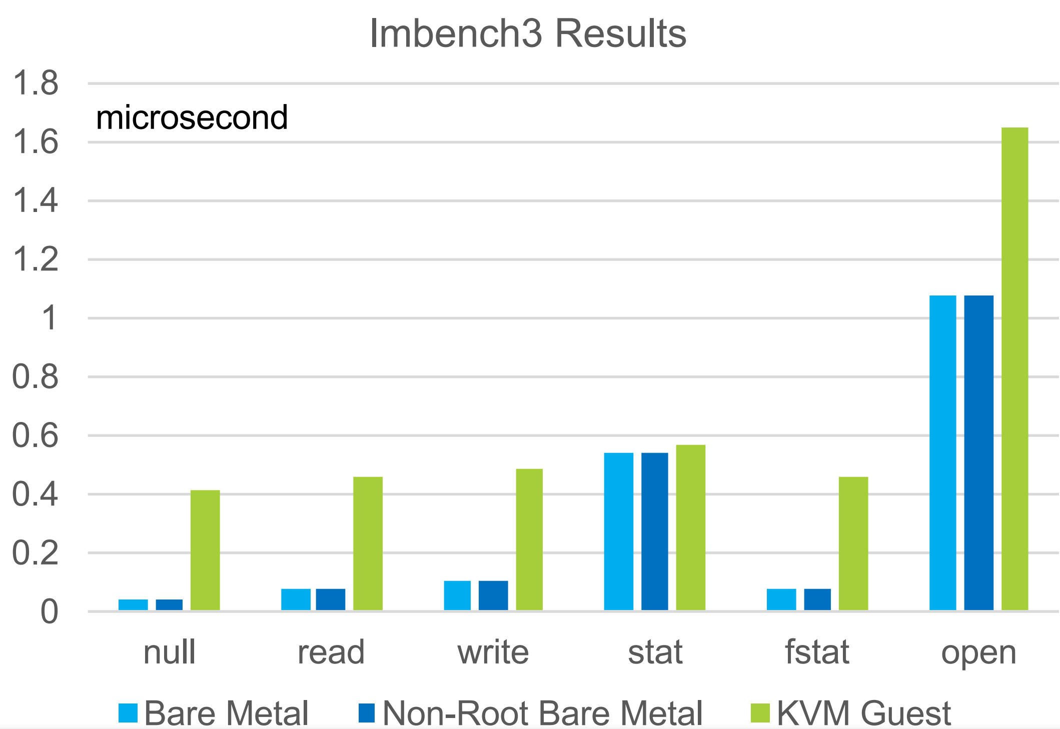
- Add new IOCTLs to KVM
  - De-privilege and privilege the current CPU (`switch_and_exit`)
  - Start running in Virtualization Non-Root Mode from the next instruction in the KVM module
  - Generate a dedicated VM exit to go back to Virtualization Root Mode
- Separate VM exit handler
  - Monitoring and protection
  - EPT is constructed in advance or at runtime (optional)
- Code changes are well contained in KVM module



Comparison of Overhead

Using lmbench (micro benchmark) and kernel build

- lmbench



- Kernel build
  - 1.2 % overhead with bare-metal kernel in Virtualization Non-Root Mode

lat_sys_call -P 1 -W 1000000 -N 1000 null

*KVM guest - qemu-system-x86_64 -enable-kvm -cpu host -smp 4 -m 4096 -hda image_file -serial stdio

Beyond Kernel Protection

Debugging:

- Monitor specific behaviors or events for debugging

More operations are available in virtualization (Virtualization Non-Root Mode):

- PML (Page Modification Logging)
  - Can be used to monitor memory activities, which guest physical memory pages are modified frequently
- #VE (Virtualization Exception)
  - Additional exception regarding GPA to HPA translation (access to non-present guest physical memory)

Hot patching and Intercepting exceptions (examples):

  - Intercept #DE in the kernel (oftentimes used as DoS) – Patching in the KVM module without modifying the kernel code

Current Status and Next Step

Current Status:

- PoC has been done (< 1000 lines of code changes to KVM module only)
- Adding policies and actions
- Planning to share the patches and findings with the community
- Feedbacks are welcome

Next Step:

- Reflect feedback to the design and patches
- Send out RFC

Q & A