# KVM on ARM

Zhao Zhenlong
<zhaozhl@inspur.com>

# Outline

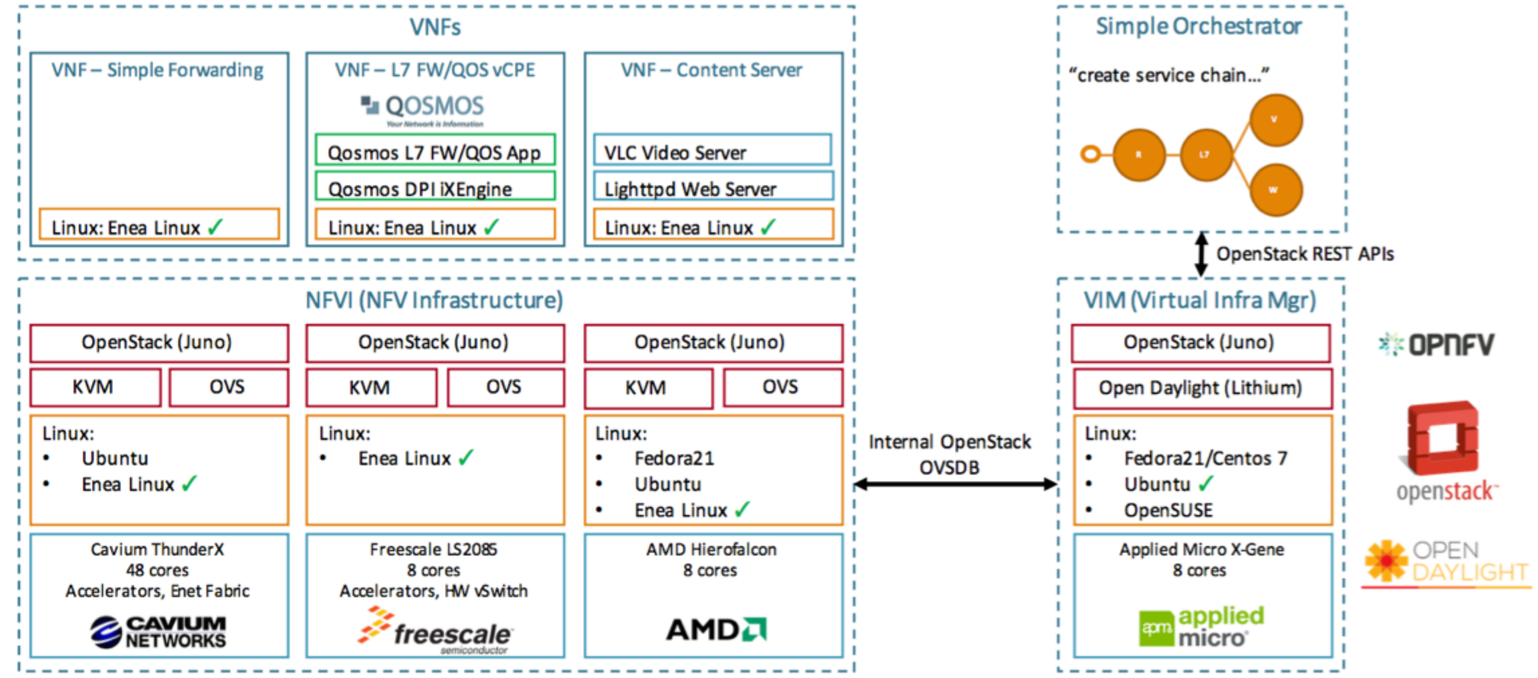
- Background
- ARM Virtualization Extension
- KVM on ARM
- Current Status

# Background

- ARM processor becomes more powerful
- 64-bit support
- Server virtualization benefits
  - High Availability
  - Server Consolidation
  - Load Balance
  - Isolation and Security

# More

- Debug
- Testing environment
- OPNFV on ARM
- Phone virtualization
  - run iOS on Android
  - Legacy system support




OPNFV on ARM: Software Components

# About Virtualization

- Formal Requirements for Virtualizable Third Generation Architectures, Popek and Goldberg
- Essentially Identical
  - A program running under the hypervisor should exhibit a behaviour essentially identical to that demonstrated when running on an equivalent machine directly.
- Efficiency
  - A statistically dominant fraction of machine instructions must be executed without hypervisor intervention.
- Resource control
  - The hypervisor should be in complete control of virtualized resources

# ARM Virtualization Extensions

# Evolution of ARM VE

- SMP
- Security extension

- HW TLB broadcast
- HW cache coherency
- Thumb-2 ISA

- LPAE
- Virtualization extensions
- Generic timers

- New AArch64 mode
- New A64 ISA
- 64bit all the way
- Full 32bit compatibility

- New atomic inst
- New SIMD inst
- MM improvements
- VMIDs expanded
- VHE

ARMv6
ARM1136,ARM1176,
ARM11 MPCore

ARMv7-A
Cortex-A5,A8,A9

ARMv7-A with VE
Cortex-A7,A15,A17

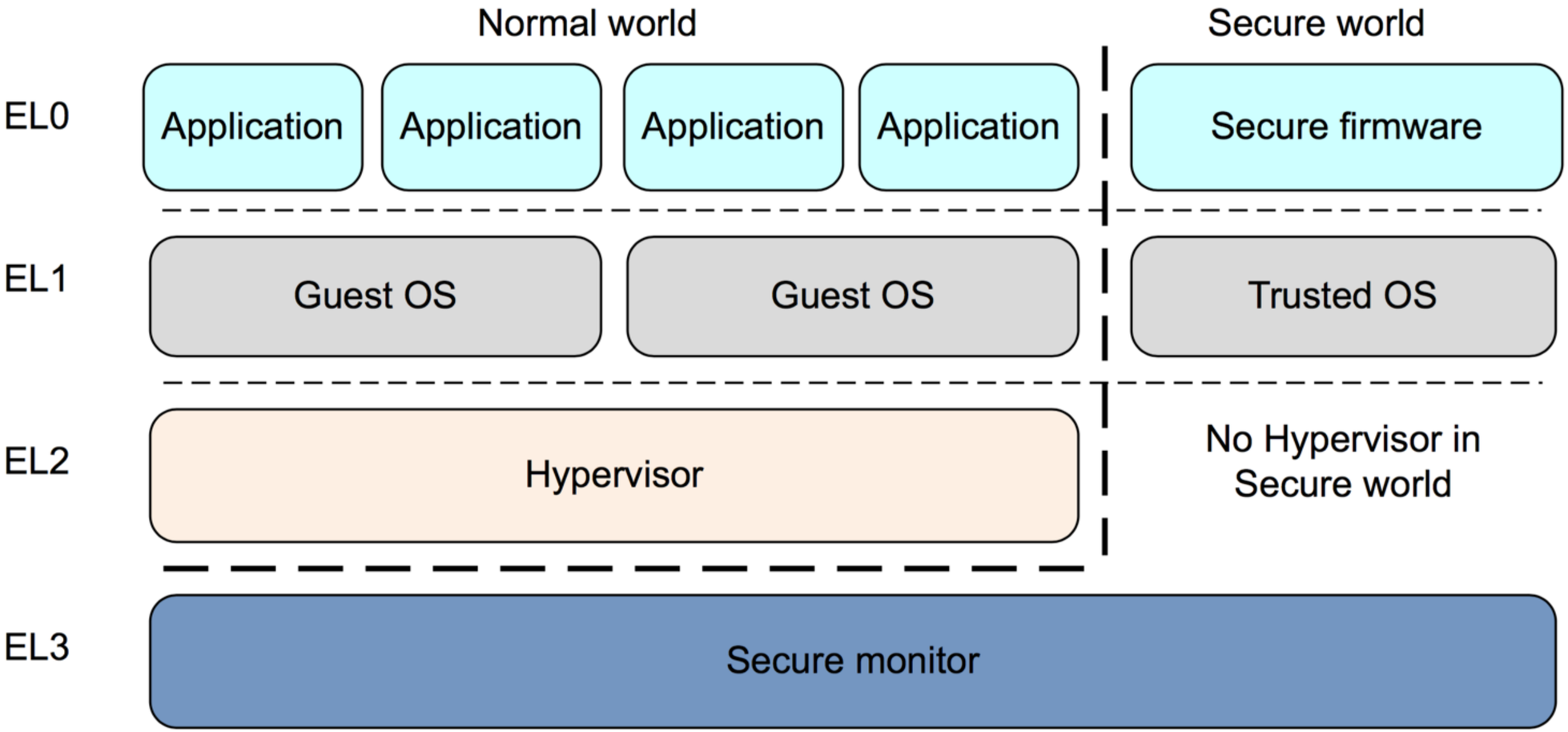
ARMv8-A
Cortex-A53,A57,A72,A73

ARMv8.1-A

# ARM Virtualization Extensions

- EL2 for hypervisor
  - Scheduling and resource sharing
  - Support Trap-and-Emulation
    - Hypervisor decides what to trap for efficiency and syndrome support for trapping key instructions
  - Guests can call into EL2 using HVC instruction for para-virtualization
- Second stage translation
  - Adds an extra level IPA between VA and PA using nested page tables
  - TLBs are tagged by Virtual Machine ID (VMID)
  - System MMU aids memory management
- Virtualization support is considered in standard architecture peripherals designing
  - GIC and timers

# Exception levels in AArch64



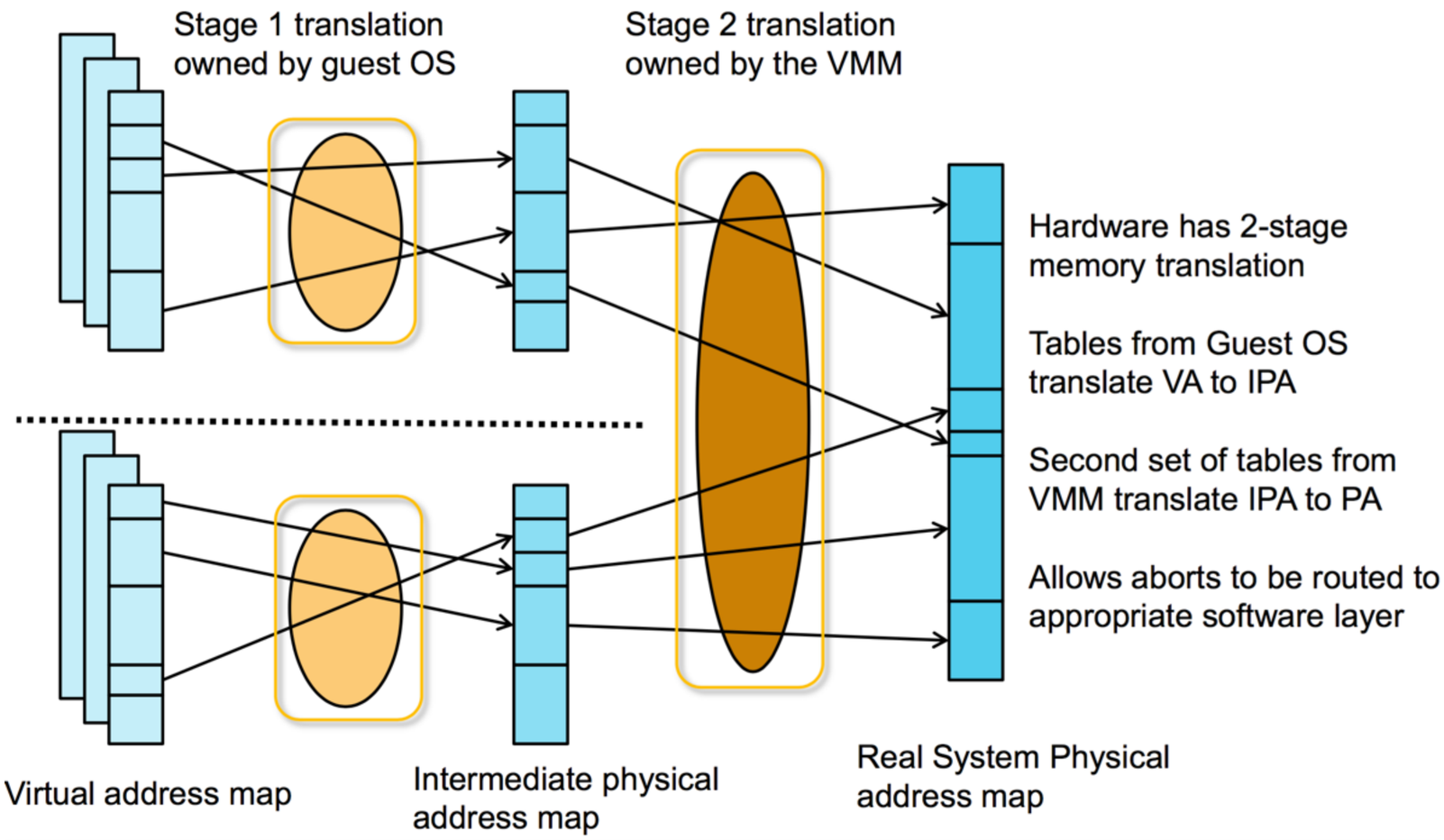
# EL2

- Access to hypervisor control registers
- Controls stage-2 translations
- Entry through exception
- Exit by exception return

# VM Exit Events

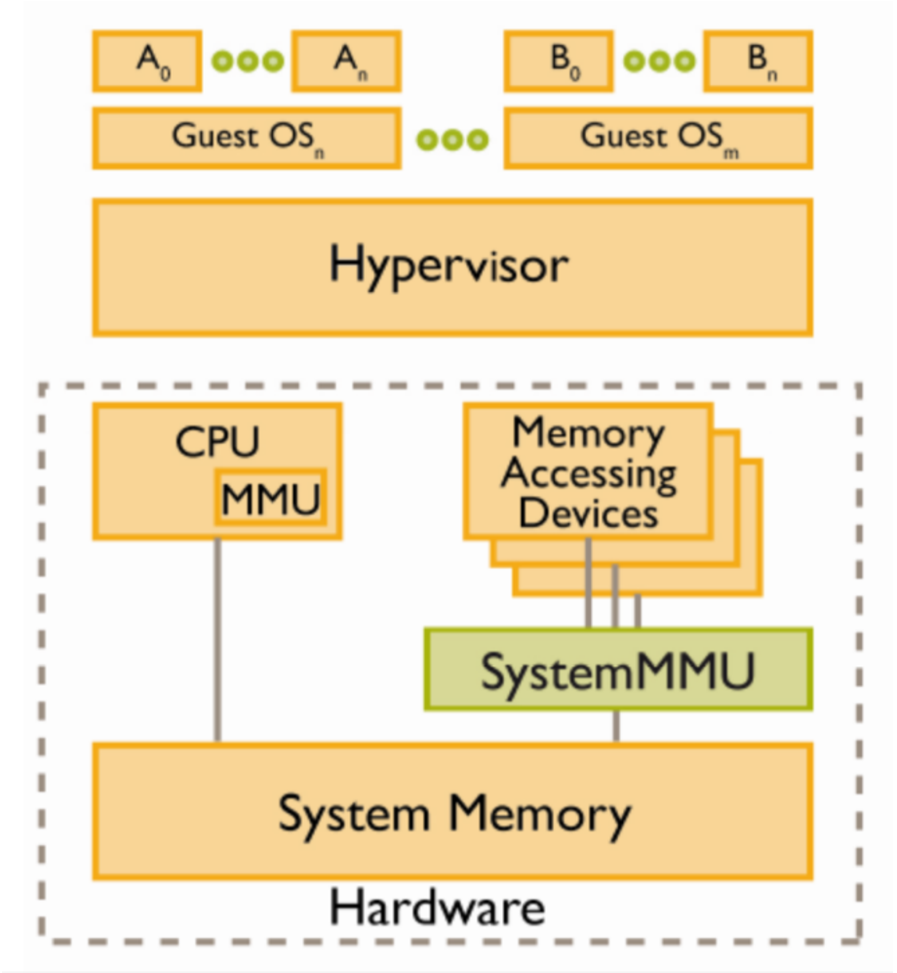
- Physical interrupt delivery
  - All the guest interrupts are virtual
  - Physical interrupts must be handled by the host
- Stage 2 fault
  - Can be either a translation, permission, or access fault
- HVC instruction
  - hypercalls
- SMC instruction
  - For secure mode services
- WFI (Wait For Interrupt) instruction
  - When the guest is waiting for an interrupt
- Blocking WFE (Wait For Event) instruction
  - To efficiently implement directed yield on blocking spinlock
- A few privileged system registers and operations

# Stage-2 Address Translation



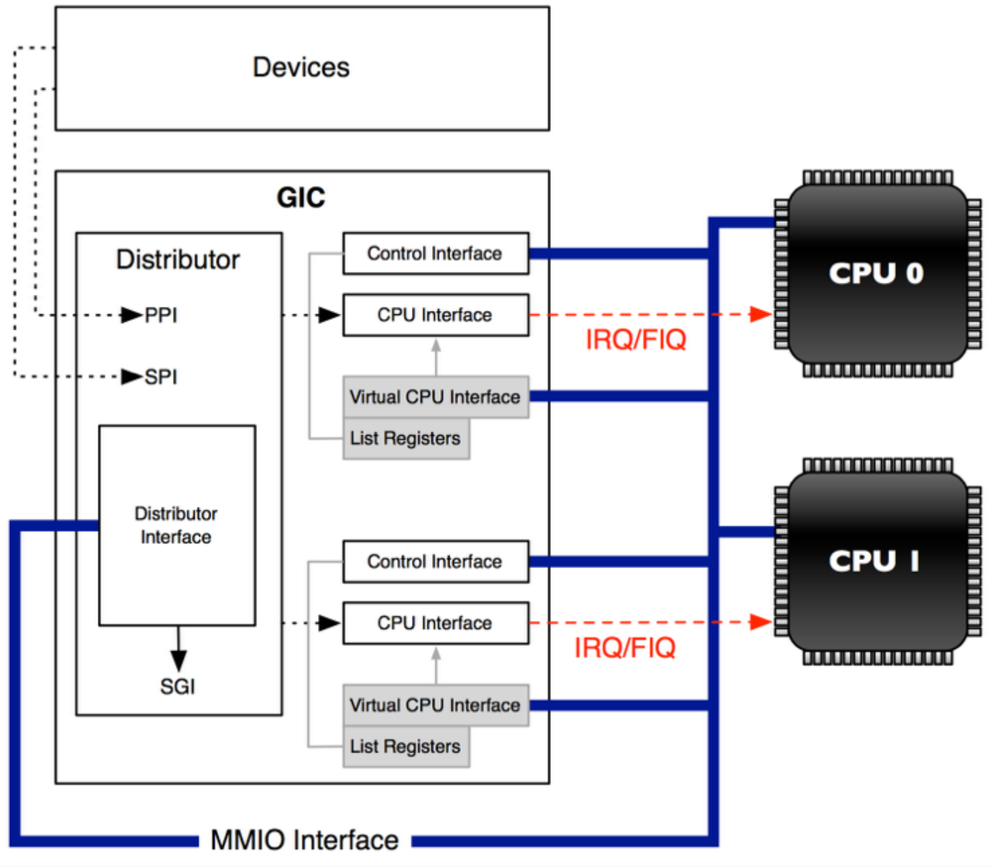
# Stage 2 SMMU Translation

- Stage 1 SMMU Translation
  - gather contiguous block of mem at VA space
  - dismiss bounce buffers
- Stage 2 SMMU Translation
  - Guest directly DMA
  - Guest device operating isolation
- SMMUv3 improvements
  - Larger number of devices
  - Support for ATS/PRI



# VGIC

- Hypervisor receives hardware interrupts and controls the guest view of injected interrupts using List Registers
- GIC hardware ensures interrupt acknowledgment and EOI is handled without exiting the guest
- GIC v2.0 include hardware virtualization
  - Virtual CPU interface
  - List Register
  - Distributor is emulated
- GIC revision constrains the size and capability of the VM
  - GICv2: maximum 8 vCPUs per VM
  - GICv3: zillions of vCPUs per VM, device isolation
  - GICv4: direct injection of MSI



# Timer

- Hardware assists time virtualization
- Global counter is always on with fixed frequency and has different view in different exception-level
  - EL2 view and physical EL1 view can both get physical counter
  - Virtual EL1 view, Virtual counter = Physical counter + offset
  - No trap when guest reads counter
- Per-CPU EL2 timer
- Two per-CPU EL1 timer
  - A physical timer
  - A virtual timer, with the same virtual offset

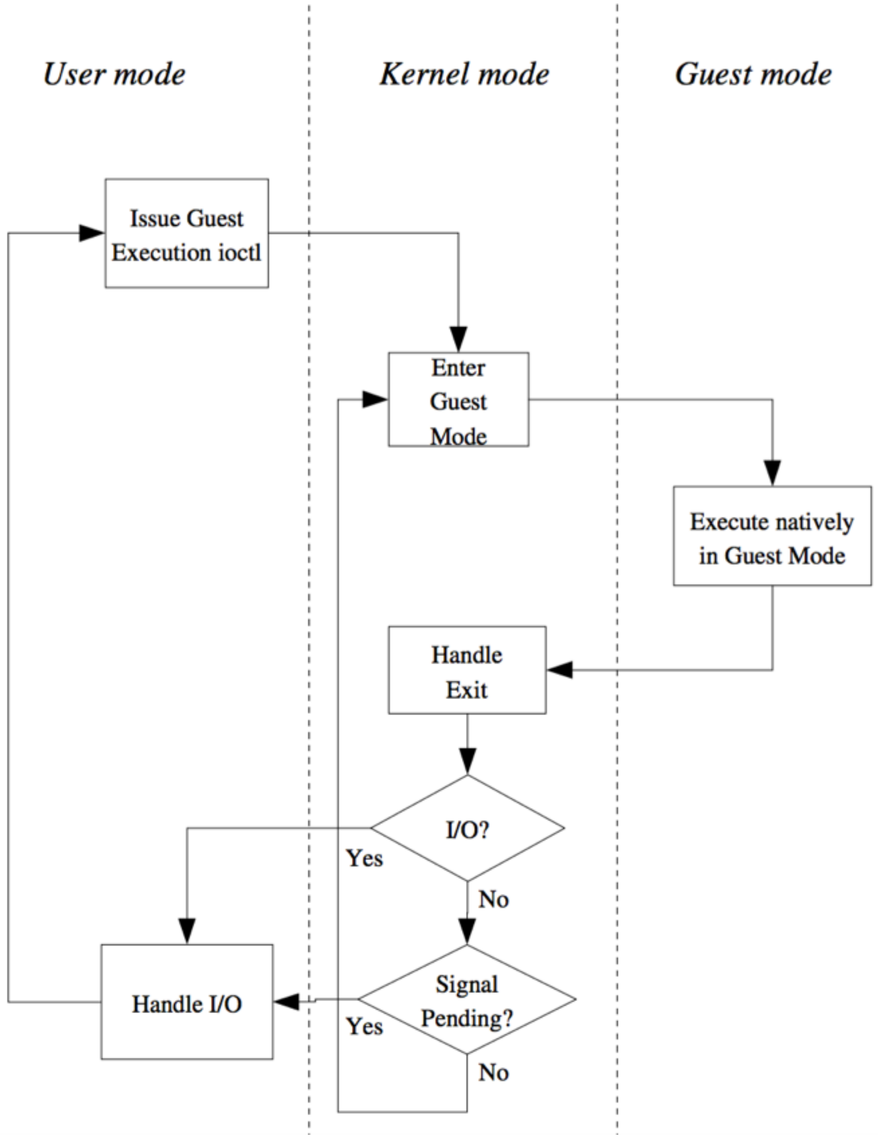
# ARM and x86

- Separate mode vs Orthogonal privilege rings
  - EL2 is not a superset of normal privileged ELn
  - Control registers must be programmed in EL2
  - Transitioning between EL2 and EL1 needs a mount of work
  - Kernel cannot directly run under EL2
- x86 hardware automatically save to/restore from VMCS when context switching
- while ARM leaves it up to software to decide which state needs to be saved and restored

# KVM on ARM

# KVM

- /dev/kvm userland interface
  - kvm_fd = open(“/dev/kvm”);
  - vm_fd = ioctl(kvm_fd, KVM_CREATE_VM, 0);
  - posix_memalign(&mem, PAGE_SIZE, 0x20000000);
  - ioctl(kvm_fd, KVM_SET_USER_MEMORY_REGION, mem);
  - vcpu_fd = ioctl(vm_fd, KVM_CREATE_VCPU, 0);
  - thread_start(run_vcpu(vcpu_fd));
  - while (1)
    - process_io();



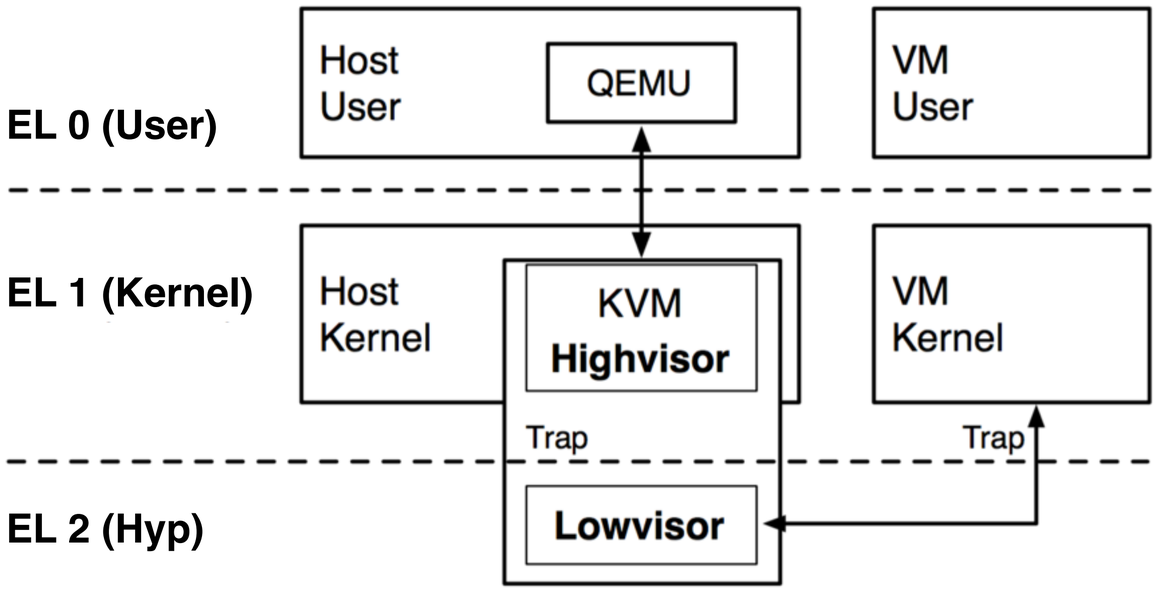

kvm: the Linux Virtual Machine Monitor

# EL2 not superset of EL1

- EL2 is a completely different CPU mode from EL1
  - Different registers
  - Separate address space and different memory model
- Run Linux in EL2 requires too many changes
  - ARM Linux guest kernel should be able to run in EL1
- Potential performance problems in EL2
  - Only one Translation Table Base Register (TTBR0_EL2)

# Split-Mode Virtualization

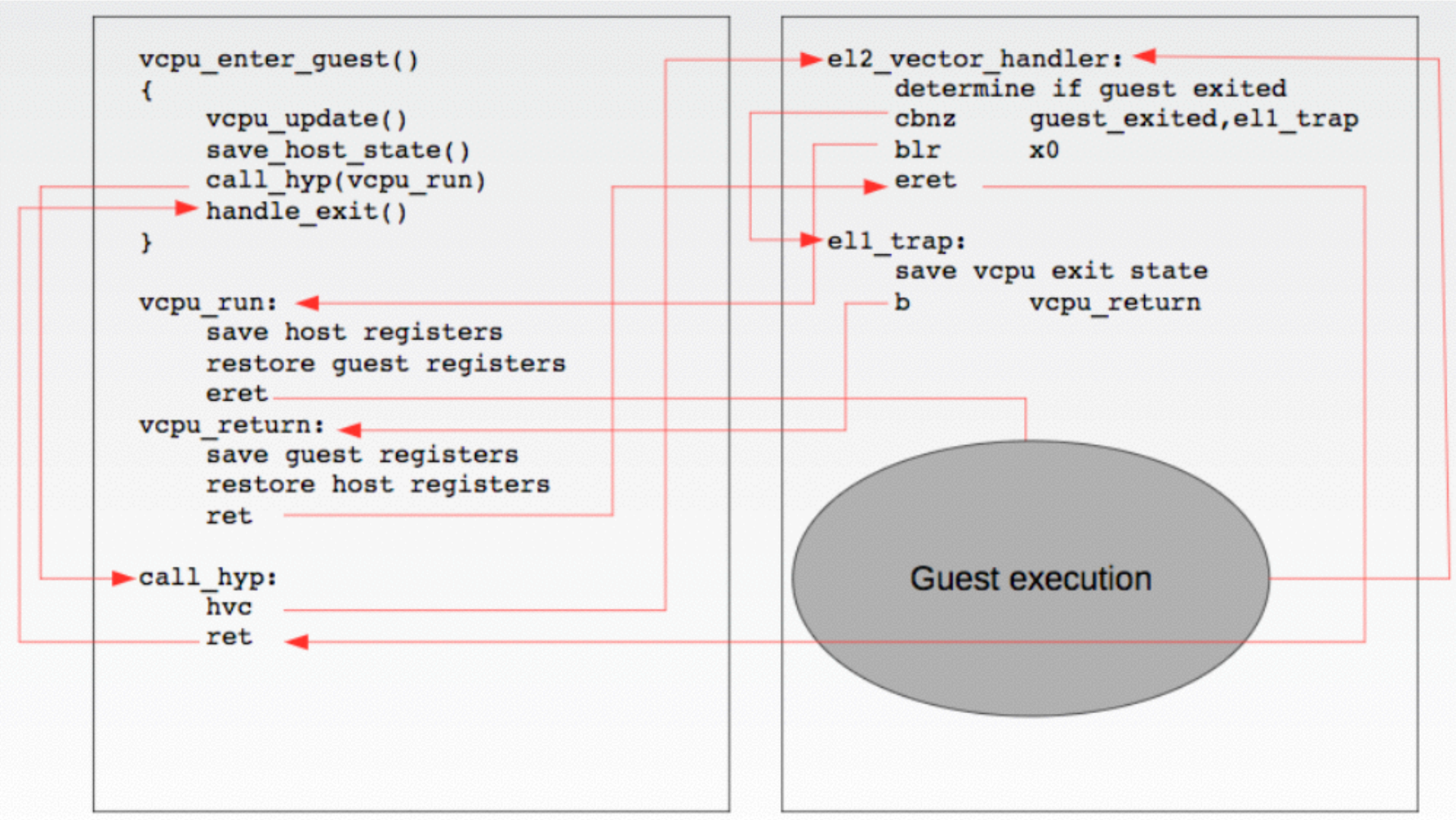
- Lowvisor in EL2
  - Configure hardware to set up execution context
  - Enforce protection and isolation
  - Context Switch
  - Handle interrupts and exceptions
- Highvisor in EL1
  - other functions in KVM



# CPU Virtualization

- Running VMs have to go through Hyp mode
- VM data structures must be mapped in Hyp mode
- Context switching guest accessible registers
- Trap access to other registers and emulate in software

# Guest Execution Process



# Hypervisor to VM Switch

- store all highvisor GP registers on the Hyp stack
- configure the VGIC and timer for the VM
- save all highvisor-specific configuration registers onto the Hyp stack
- load the VM's configuration registers onto the hardware
- configure Hyp mode to trap VM exit events
- write VM-specific IDs into shadow ID registers
- set the Stage-2 page table base register (VTTBR) and enable Stage-2 address translation
- restore all guest GP registers
- trap into either user or kernel mode.

# Memory Virtualization

- Based on Stage-2 translation
  - Enable when running in a VM
  - Disable when running in the highvisor and lowvisor
- Stage-2 translation page tables is managed by highvisor
  - Allocates memory for a VM
  - Accesses to unallocated addresses will cause stage-2 page faults

# VM Page Table Setup

- Create a blank stage-2 PGD
- Run vcpu
- VM generates stage-2 fault and trap to EL2
- Capture fault information
- Create stage-2 mapping
- Run vcpu

# I/O Virtualization

- ARM I/O are based on load/store to MMIO device regions
- A stage-2 abort not in RAM considered I/O operation, fault address can be routed to QEMU
- ESR_EL2 provides emulation hints

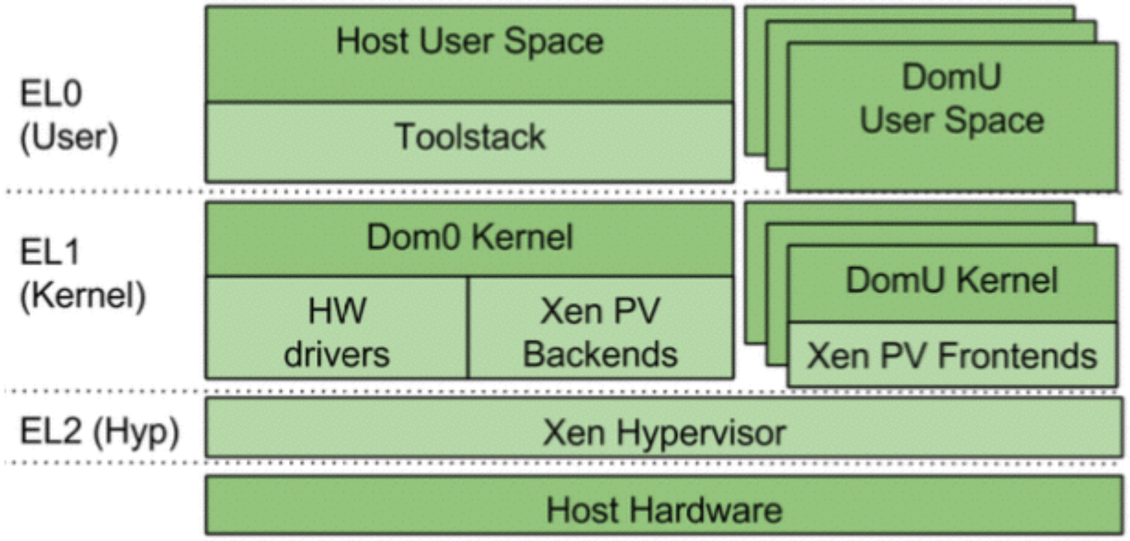
# Interrupt Virtualization

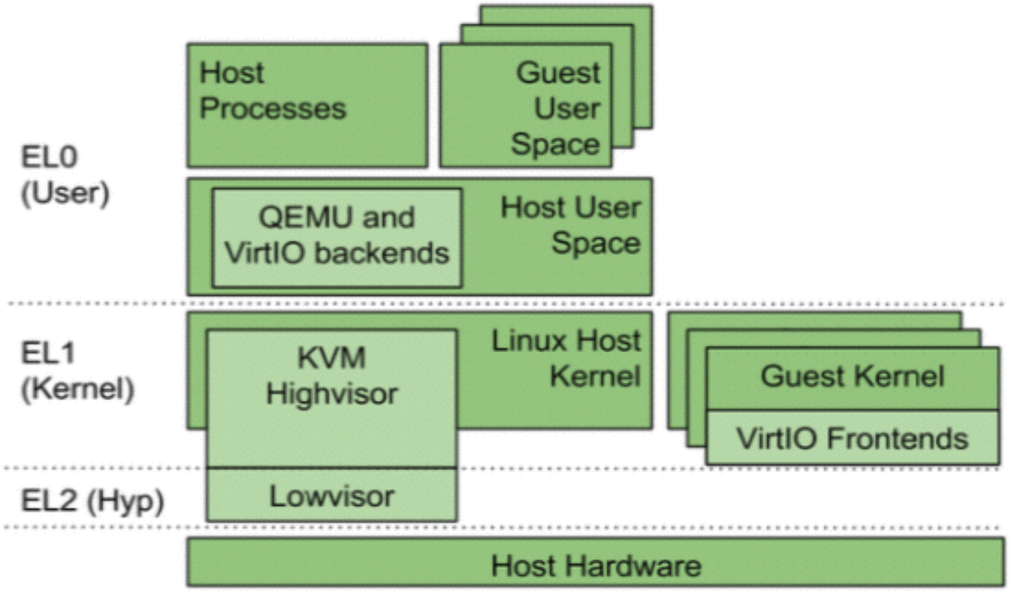
- All I/O is MMIO
- A stage-2 abort not in RAM considered I/O
- Emulation hints in HSR

# Interrupt Virtualization

- Virtual interrupts
  - Generated by QEMU devices or virtual IPI
  - Raised to virtual CPU by programming list registers
- Distributor mechanism is needed between QEMU device and virtual CPU interface
  - Emulated distributor as part of highvisor schedules virtual interrupt
  - QEMU device raise virtual interrupt to emulated distributor by userspace interface
  - Pending interrupts are written into the list registers

# Xen and KVM


EL0
(User)
EL1
(Kernel)
EL2 (Hyp)
Host User Space
Toolstack
DomU
User Space
Dom0 Kernel
HW
drivers
Xen PV
Backends
DomU Kernel
Xen PV Frontends
Xen Hypervisor
Host Hardware


EL0
(User)
EL1
(Kernel)
EL2 (Hyp)
Host
Processes
Guest
User
Space
QEMU and
VirtIO backends
Host User
Space
KVM
Highvisor
Linux Host
Kernel
Guest Kernel
VirtIO Frontends
Lowvisor
Host Hardware

# Current State

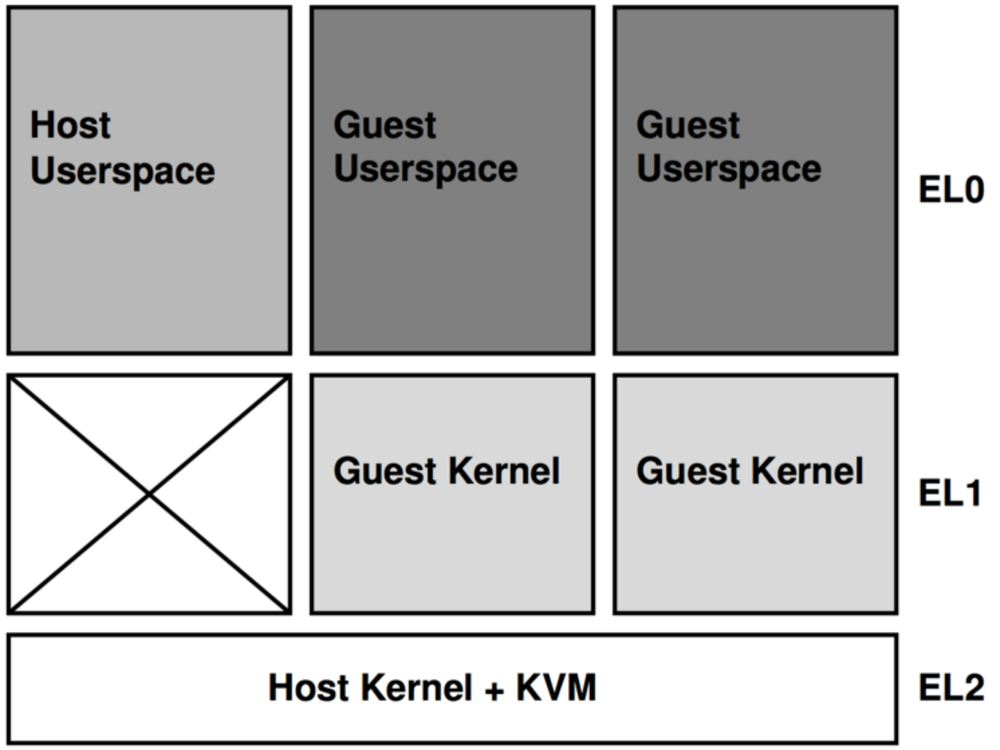
# State of KVM/ARM

- Debug support
- Virtualization Host Extensions (kernel at EL2)
- Virtual PMU
- 16K pages
- VGIC rewrite
- MSI support

from KVM Forum 2016 - Keynote

# VHE

- Virtualization Host Extensions
- EL2 becomes a strict superset of EL1
- Allows the host OS to be run at EL2 with minimal changes



# References


- ARM® Cortex®-A Series Programmer's Guide for ARMv8-A
- KVM/ARM: Experiences Building the Linux ARM Hypervisor
- ARM Virtualization: Performance and Architectural Implications
- Virtualization and the ARM architecture, XDS'15 by Marc Zyngier
- KVM: Has ARM done it better? Brno DevConf'15 by Drew Jones
- kvm: the Linux Virtual Machine Monitor, OLS'07 by Avi Kivity

# Thanks