ARM-KVM: Weather Report

Korea Linux Forum

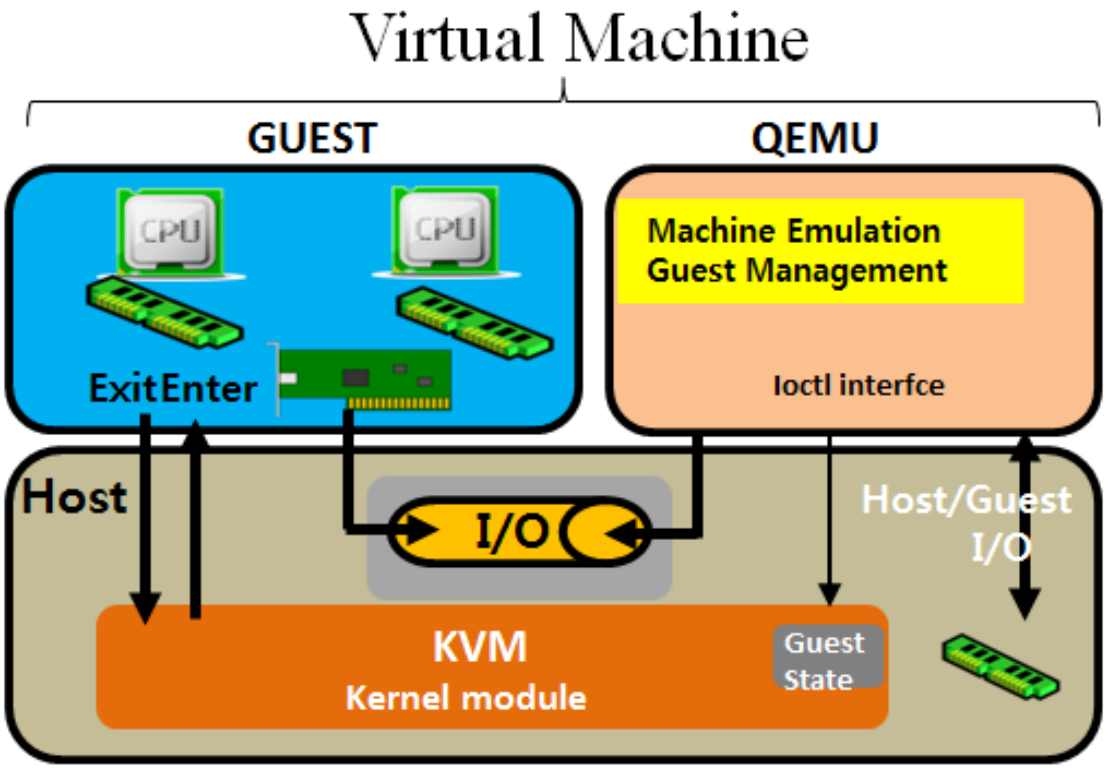
Mario Smarduch
Samsung Open Source Group
Senior Virtualization Architect
m.smarduch@samsung.com

ARM-KVM This Year

- Key contributors Linaro, ARM
  - Access to documentation & specialized HW an issue
  - ARM64 subtree – 12+ hw vendors

- Some of the new features added since last year:
  - QEMU/Guest – cache-coherency resolved
  - GICv2m – interrupt controller (GICv3 spec not public)
  - Device Pass-through
  - Virtual Platforms with kernel platform selection
  - 16-k page size support
  - Guest Debug Support

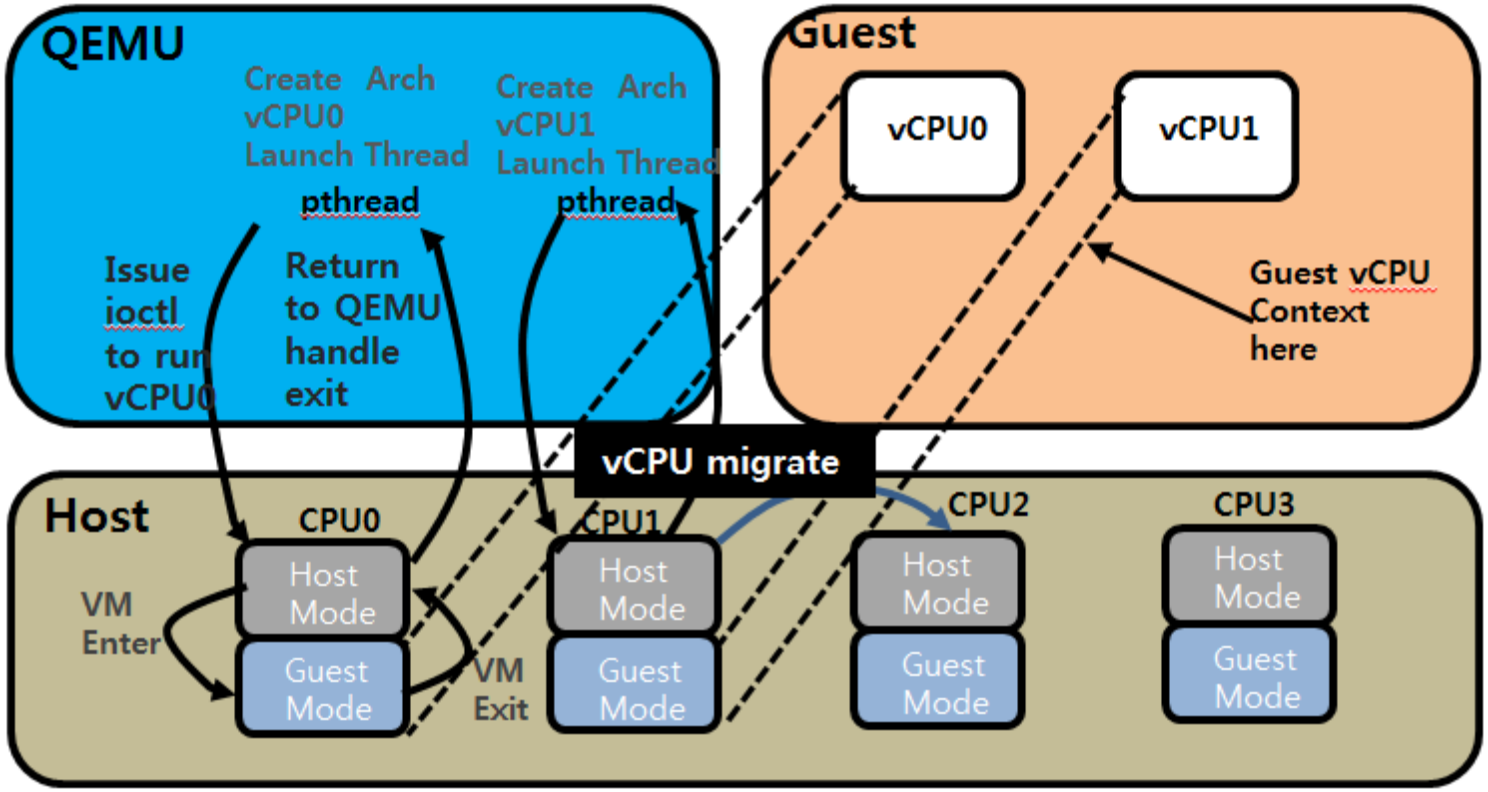
What is KVM?


Virtual Machine
GUEST
QEMU
CPU
CPU
ExitEnter
Machine Emulation
Guest Management
Ioctl interfce
Host
I/O
Host/Guest
I/O
KVM
Kernel module
Guest
State

Where is KVM in the Cloud?

- Host Kernel, KVM Module, QEMU, and Guest working together
  - Kernel – KVM reuses kernel MMU, synch, scheduling, timers, interrupts, etc.
  - Kernel matures – KVM reuses
  - KVM - runs vCPU loop, traps/fix/resume guest, emulate
  - QEMU/Kvmtool – platform emulation, Guest Management, I/O
  - Guest – kernel, disk image, I/O – unaware of virtual platforms

vCPU Scheduling

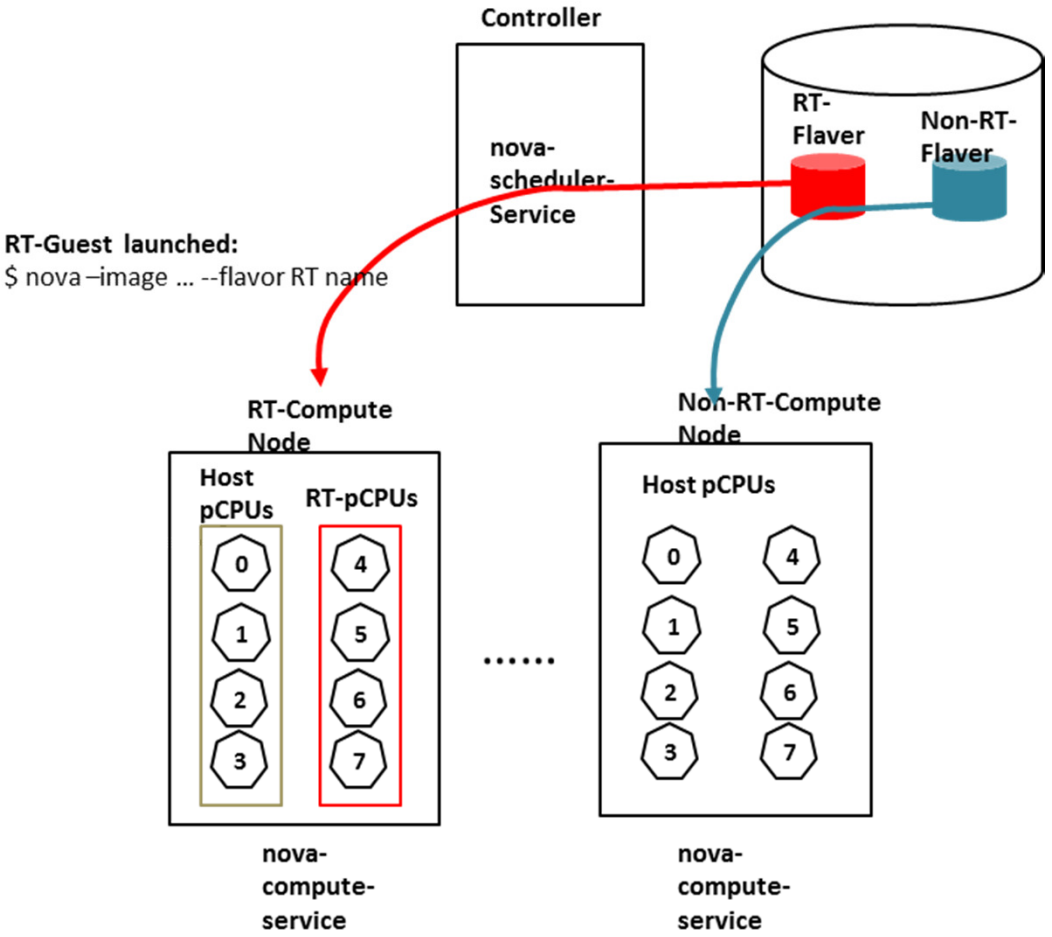


vCPU Scheduling

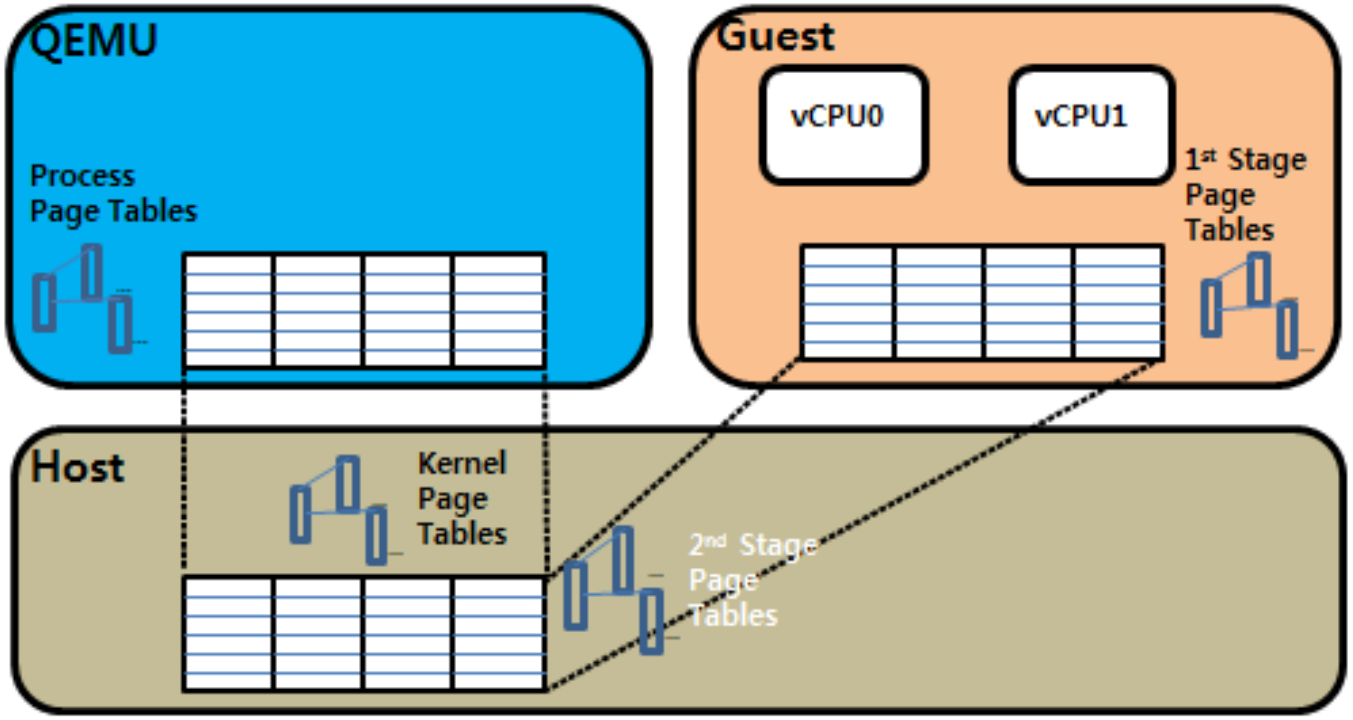
- Physical CPU – can be in host or guest mode
  - Guest mode uses HW Extension support
- Guest CPU – its a thread in Guest mode aka vCPU
- Transitions
  - Host > Guest a VM Enter - save host, load guest context
  - Guest > Host a VM Exit - save guest, restore host, resolve exit, and later VM Enter
- vCPUs are threads so you can:
  - Use taskset, chrt, numactl, ps
  - Use KVM to leverage kernel scheduler code for preempt notifiers and vCPU scheduling

NFV Example

LTE Network Element - Isolation



Guest Memory Management



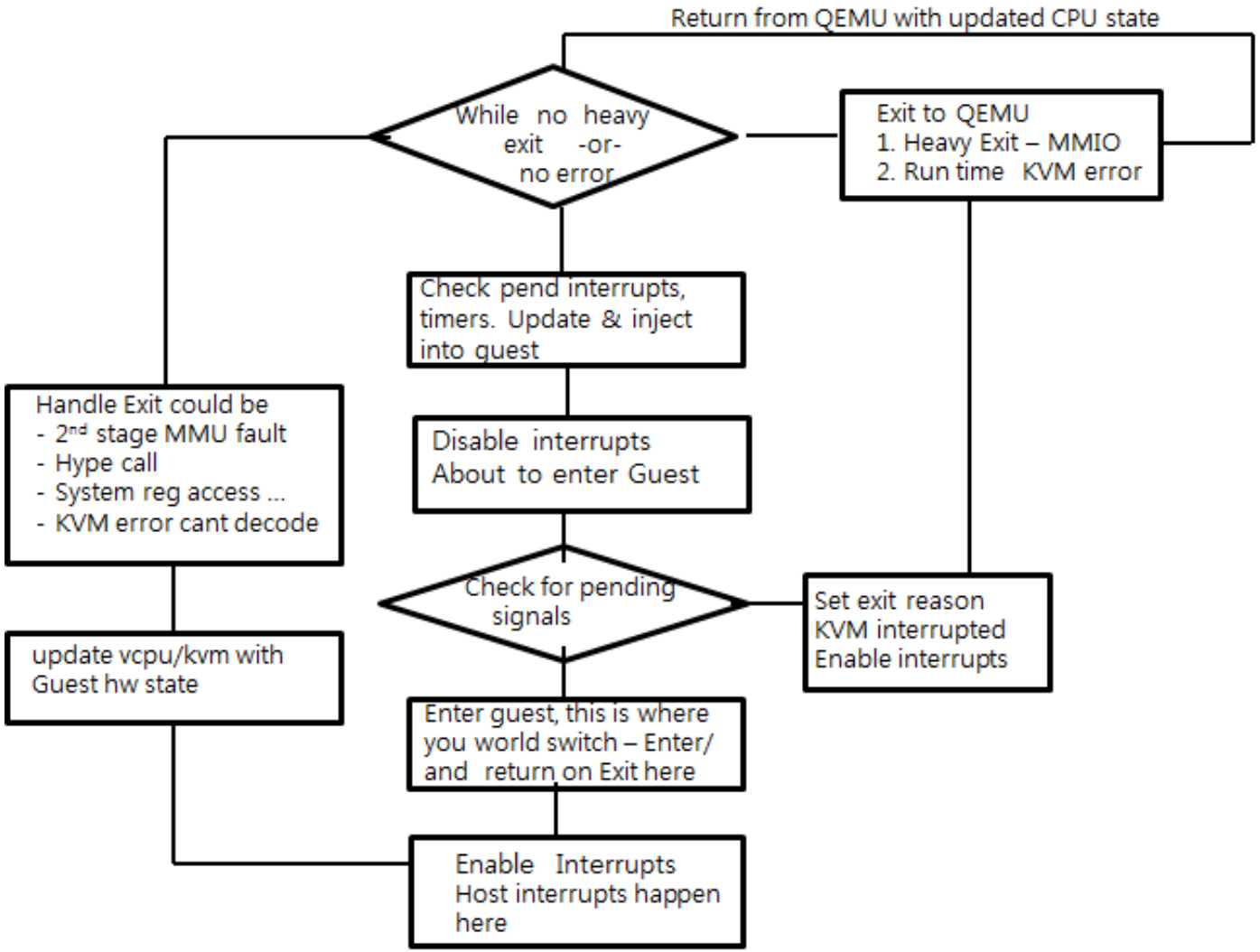
Guest Memory Management

- QEMU backs guest memory with mmap() region
  - Register QEMU VA/GPA range with KVM
- Guest access – 2nd stage fault
  - KVM – (1) GPA > QEMU VA > get a page > update 2nd stage, QEMU
  - Guest resolves stage 1
- 4 – tables
  - QEMU process, Kernel, 1st, 2nd stage tables
- KVM leverages kernel MMU code
  - paging, mmu notifiers, page allocation, and topology (flat, numa)

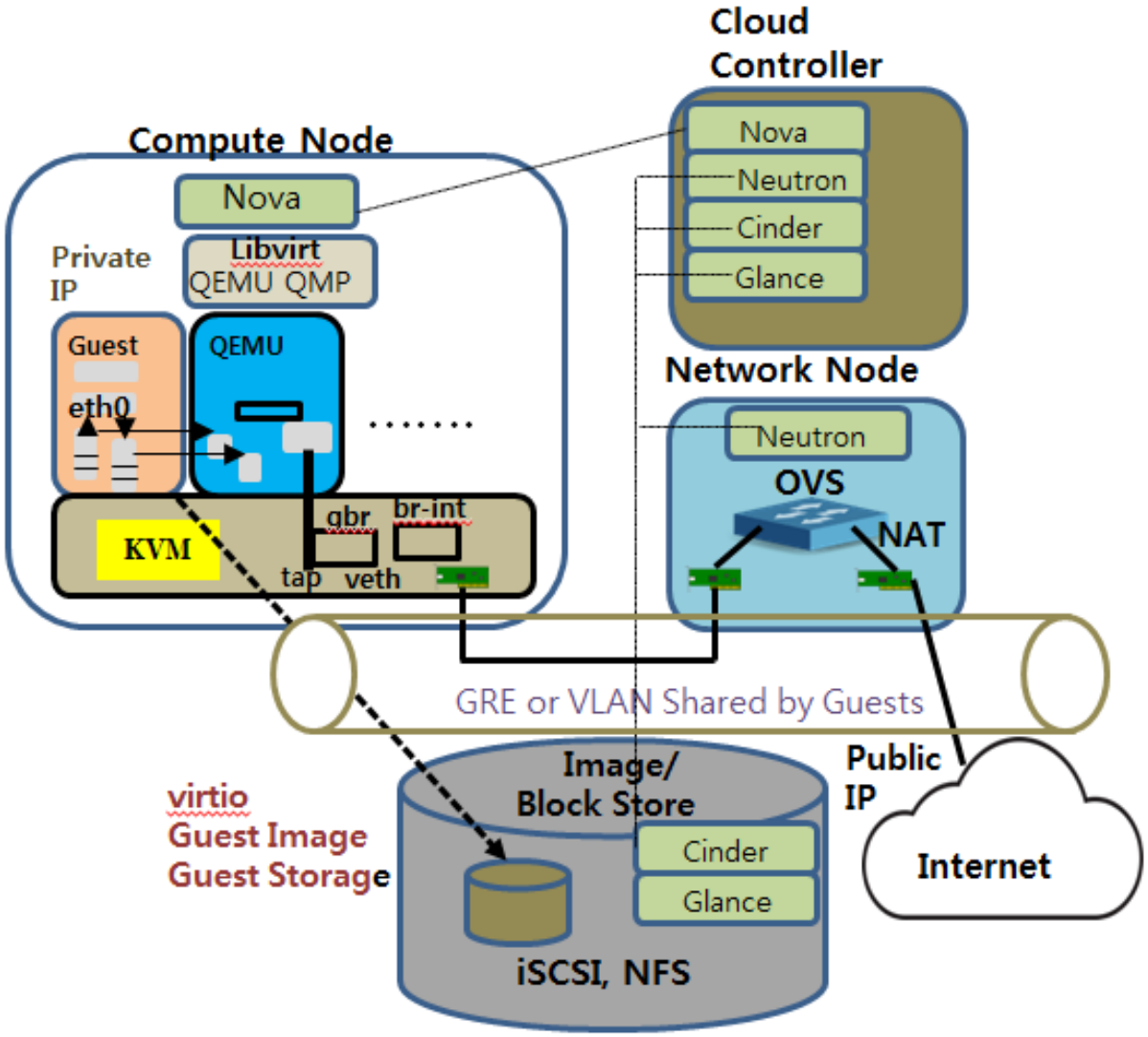
I/O

- Virtio – Dominant in cloud
  - QEMU/Guest map the same memory
  - Tx, Rx, Ctrl – Virt-Qs used
  - QEMU translates GPA to/from QEMU VA
- QEMU MT – vCPUs + IO Thread(s)
  - IO thead – frontend – virt-q & backend host OS transport

KVM vCPU Loop



KVM in the Cloud

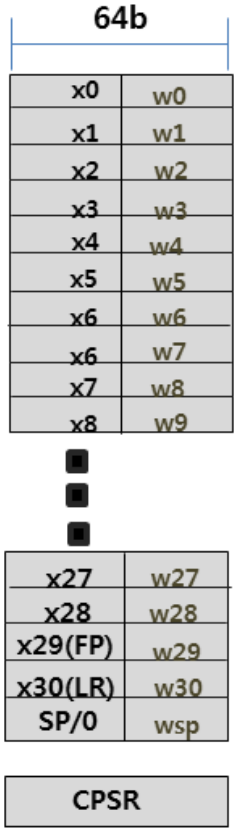


KVM in the Cloud

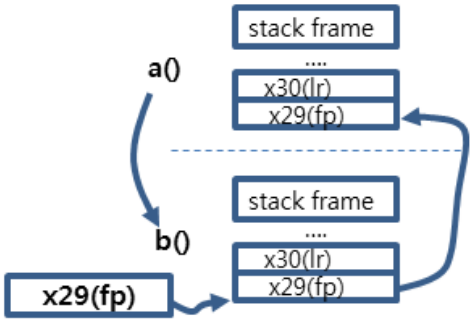
- IaaS Admin – Compute node provides access to
  - Create private/public networks
  - Install Images, create block storage
  - Backend/mgmt network access
- QEMU/KVM on Compute node
  - Cloud Controller interfaces with Libvirt
  - Libvirt launches guest, QEMU, and Image
    - Virtio: attached nework, storage
  - Libvirt uses QMP for QEMU mgmt
    - halt, mem balloon – infl/defl

ARM64 Memory Refresh

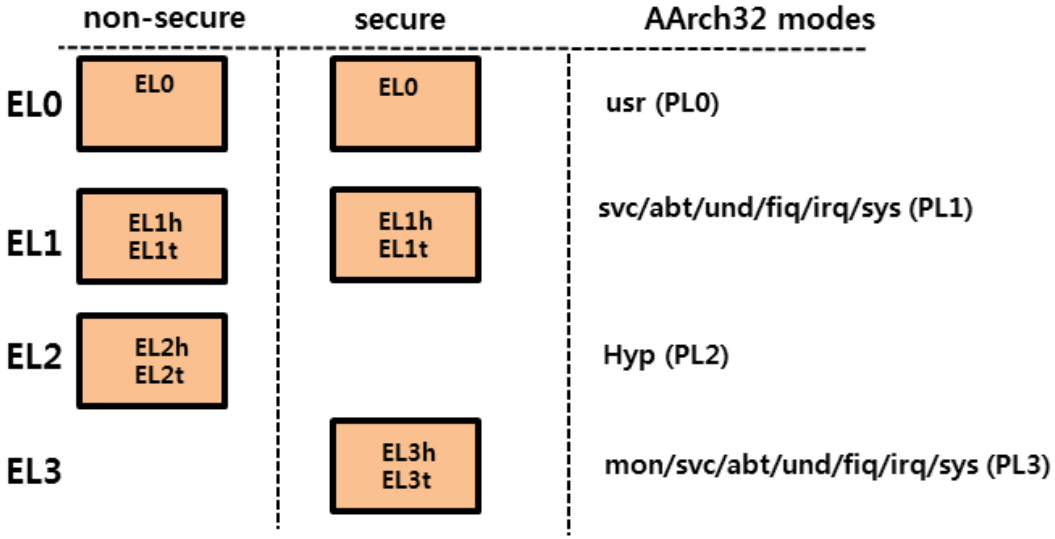
Register Set



Basic Procedure Call

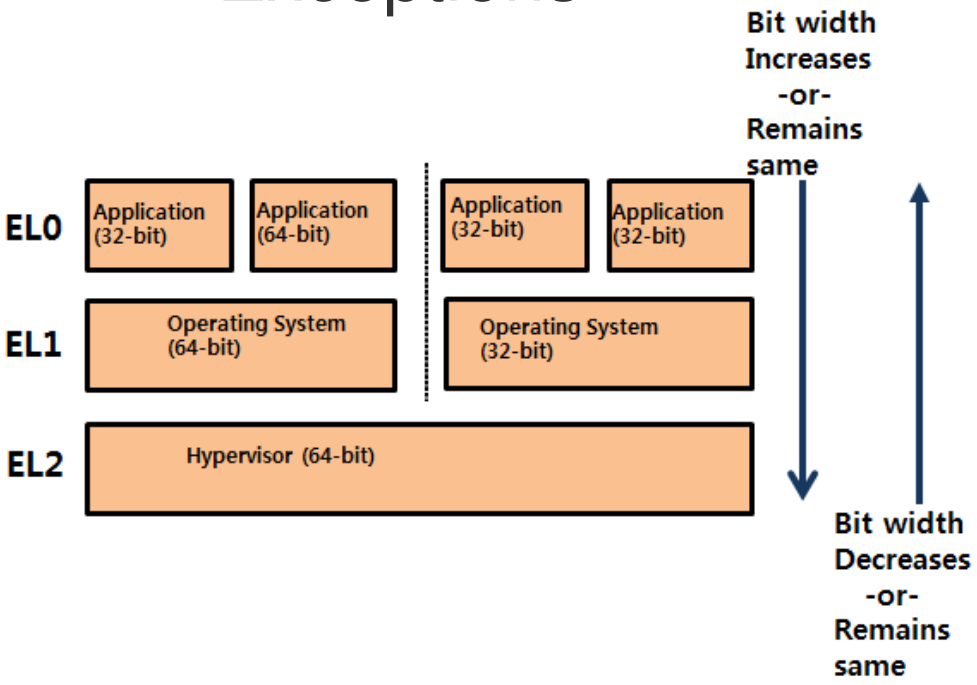


Exception Model

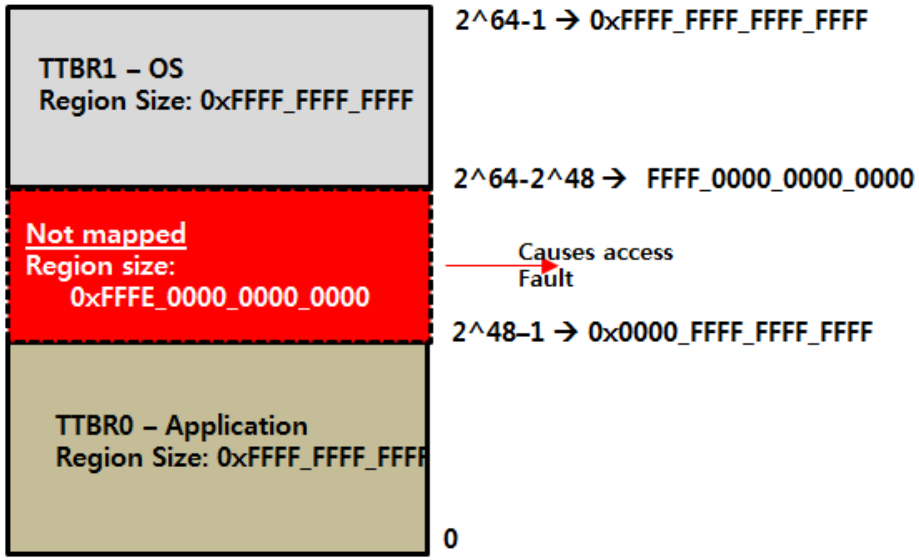


ARM64 Memory Refresh

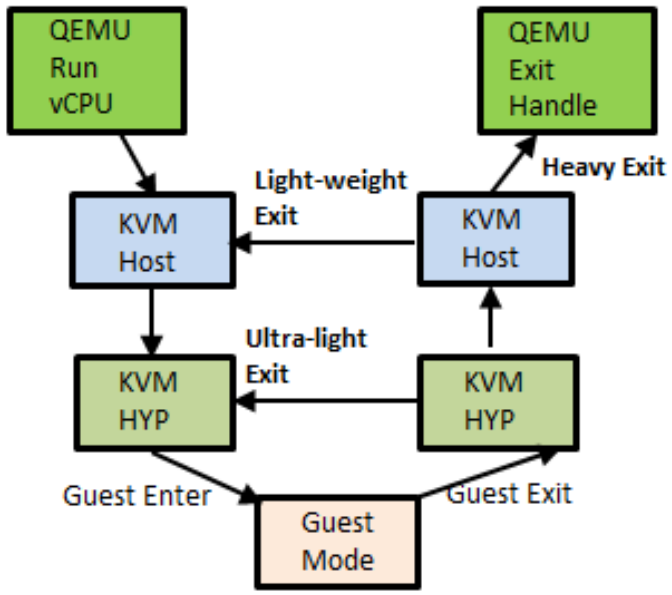
Bit Width and Exceptions



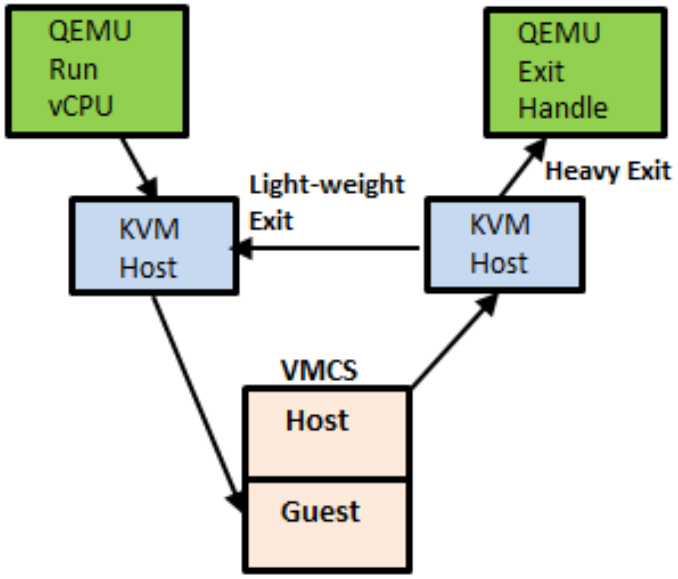
Address Size



ARM and x86

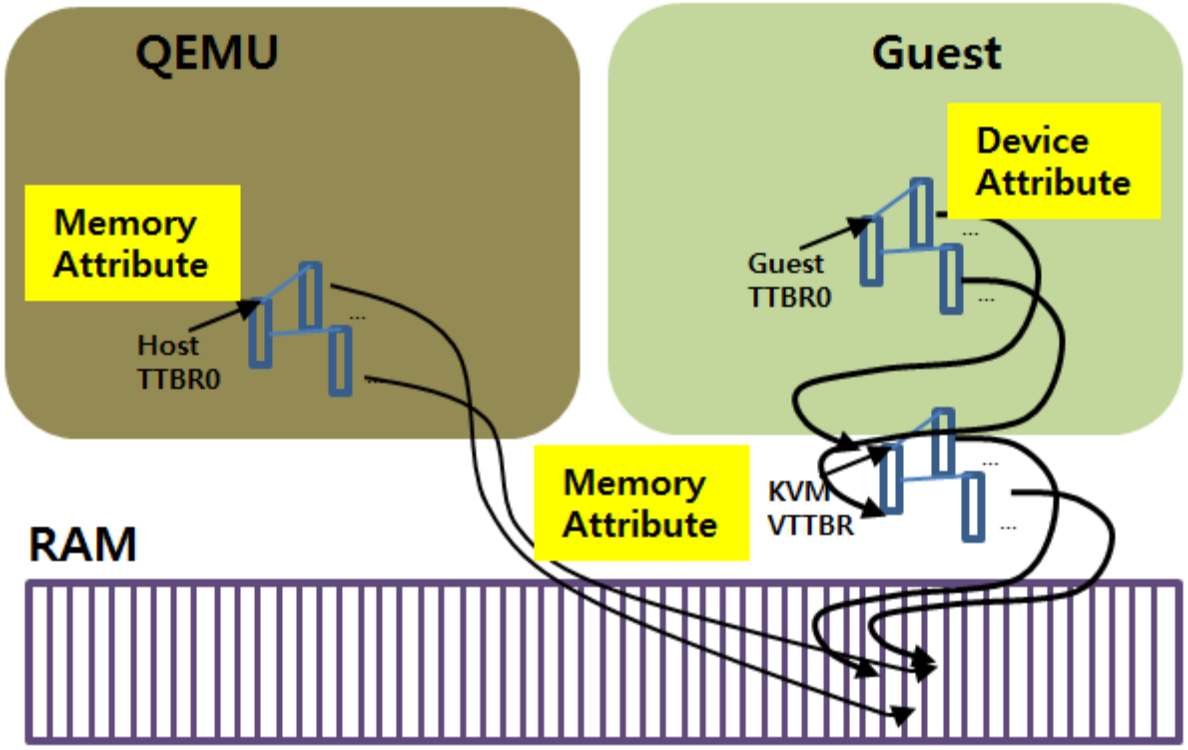


ARM HYP/EL2 Mode – Guest Context Switch



x86 VMCS – Guest Context Switch

Guest/QEMU Coherency



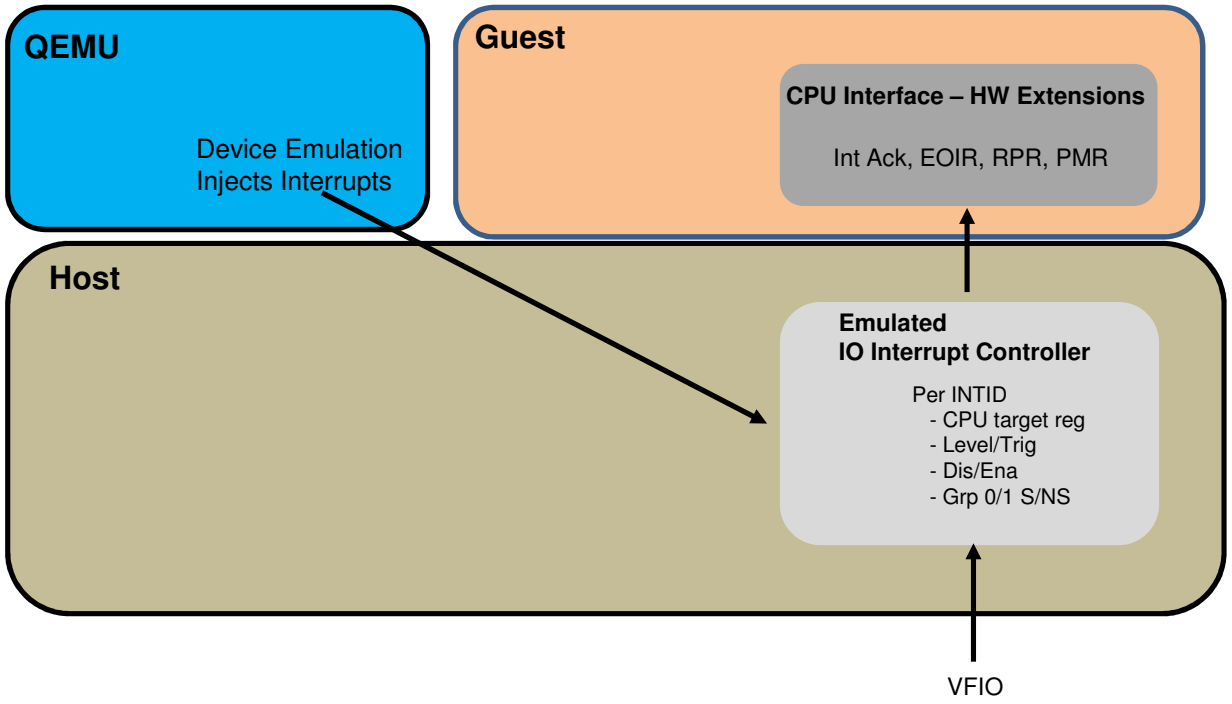
Guest/QEMU Coherency

- Blocked progress for some areas
- Strict guest device attributes prevail
  - Dealing with normal memory
  - Devices break emulation
  - Driver observes device, QEMU memory attr.
    - In-coherent view
- LCD
  - Guest updates not observed by QEMU

An Issue with Coherency

- Flash emulation broke
  - Reads from memory
  - Writes mmio unlock/write/lock
  - QEMU/Guest coherency issue
  - Several attempts to resolve include using fake guest attributes, modifying QEMU MMU
  - KVM Forum Solution - Expose devices as DMA cacheable

Interrupts High Level



Interrupts

- ARM GICv2 interrupt IDs 16 - SGI, 16 - PPI, 992 – SPI
  - Interrupt Space Limited, no MSI support
- MSI/MSI-x
  - MSI up to 32 interrupts/function – address/data
  - MSI-x table up to 2048 entries address/data per entry
  - Edge triggered - re-enable delivery on device
  - Interrupt source identified easily
  - Messages instead of hw lines
  - Devices can target many CPUs & vectors. E.g. 8-CPUs, 128-Int IDs

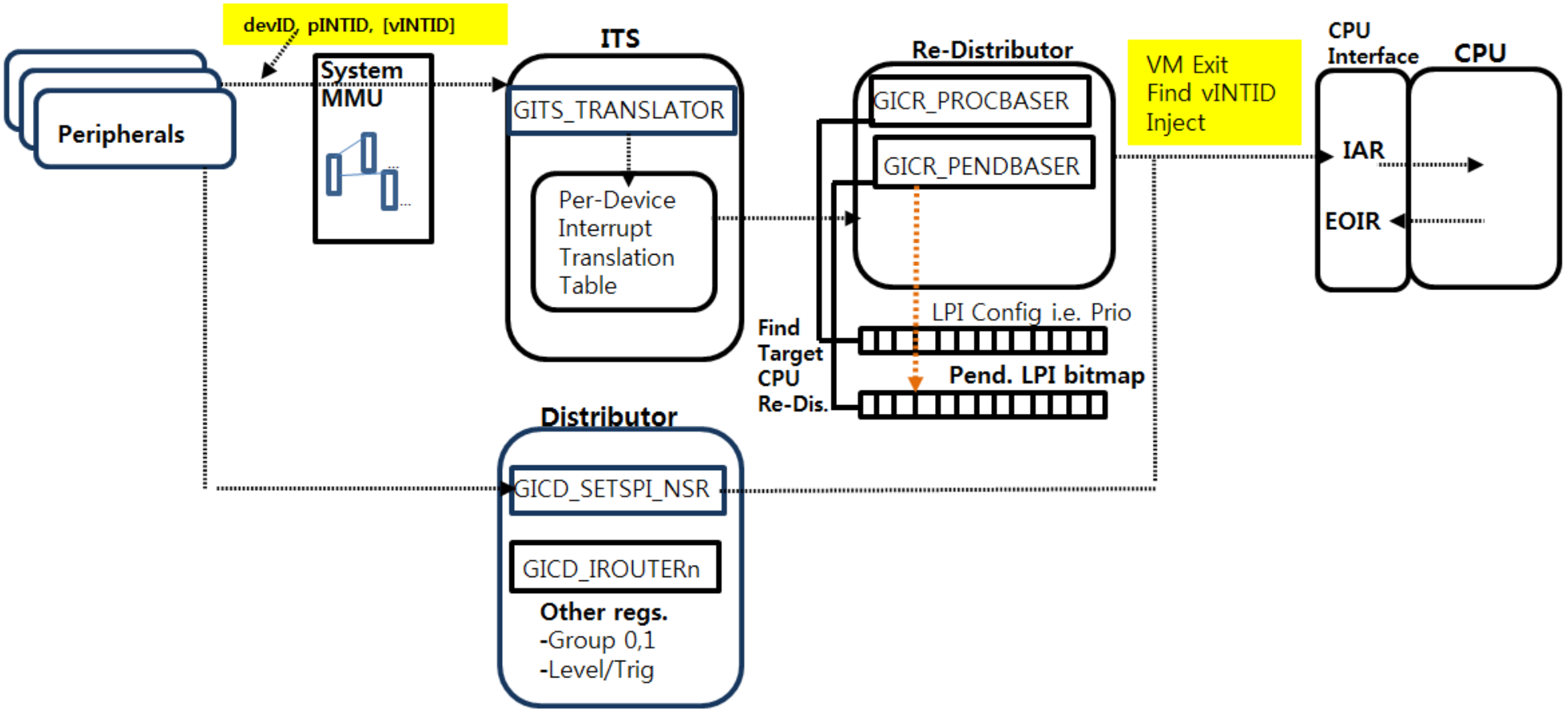
GICv2m

- MSI/MSI-x – using SPIs
  - Up to 32-clusters 8 CPUs/cluster
    - Affinity Routing enabled to target CPUs
  - Generate MSI/MSIx peripheral writes – using SPIs
    - GICD_{SET|CLR}SPI_NSR –
    - Few other regs to program

GICv2m

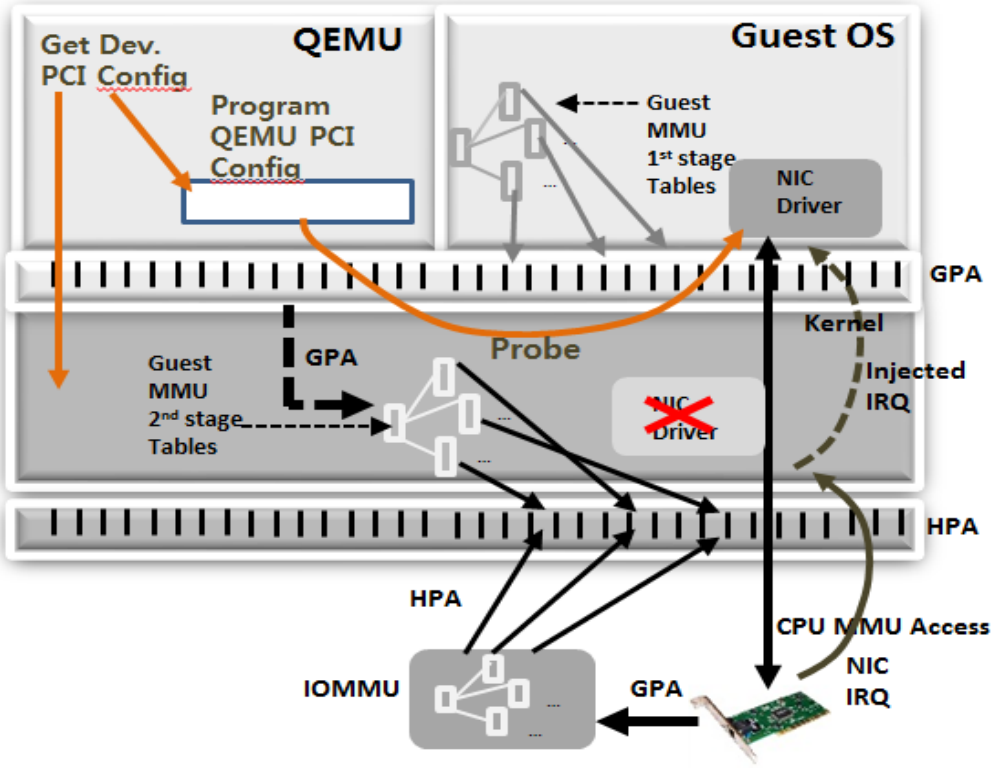
- MSI/MSI-x – using LPIs
  - Interrupt Translation Services
    - Huge LPI space of 57K+ interrupt IDs
    - GITS_TRANSLATER – dev id + LPI id – generate INTID
    - Device can target many CPUs & vectors. E.g. 16-CPUs, 128-Int IDs each
    - ITS – Guest programs peripherals directly
    - ITS translates from virt interrupt id to phys interrupt id
    - KVM injects virt interrupt
  - For Guest support must emulate Re-Dis, ITS, & Distributor

GICv2m


devID, pINTID, [vINTID]
Peripherals
System MMU
ITS
GITS_TRANSLATOR
Per-Device Interrupt Translation Table
Re-Distributor
GICR_PROCBASER
GICR_PENDBASER
LPI Config i.e. Prio
Pend. LPI bitmap
Find Target CPU Re-Dis.
VM Exit
Find vINTID
Inject
CPU Interface
CPU
IAR
EOIR
Distributor
GICD_SETSPI_NSR
GICD_IROUTERn
Other regs.
-Group 0,1
-Level/Trig

Device Pass-Through

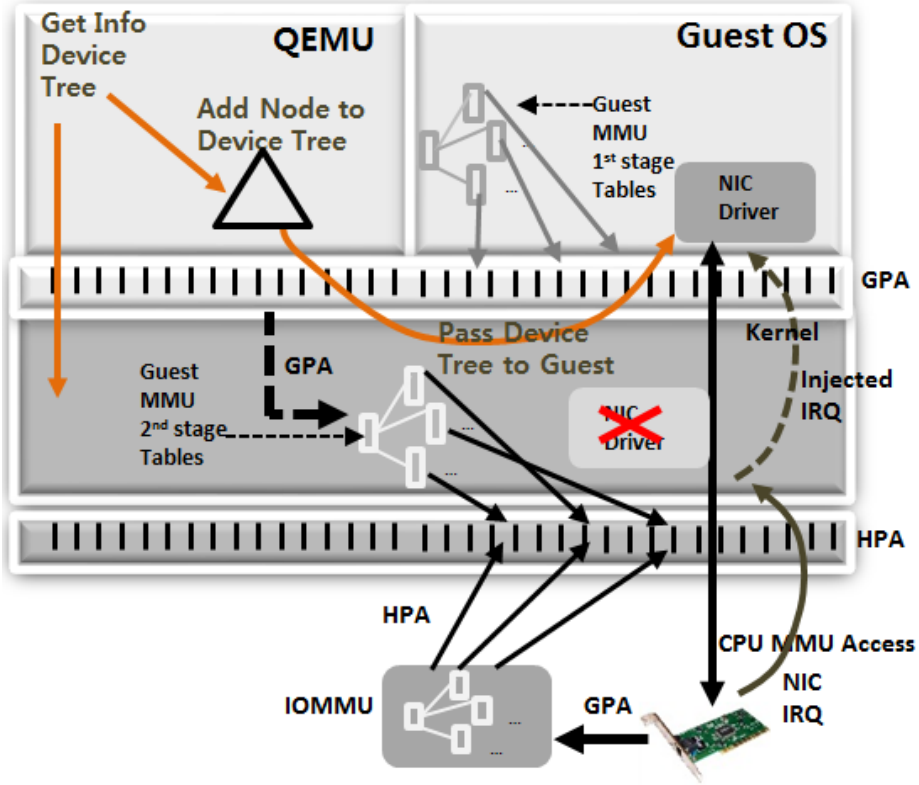
- Device pass through using – PCI
  - PCI pass through – 'device vfio-pci,host=xx.xx.xx'



- QEMU
  - Reads device PCI Config from kernel i.e xx.xx.xx
  - Qemu Picks B/D/F programs it
  - Guest enumerates – accesses PCI Config
  - Maps memory – BARs i.e. 2nd stage
  - IOMMU – guest memory
  - Sets up interrupts

Device Pass-Through

- Device Pass Through using device tree
  - -device vfio-<device name>



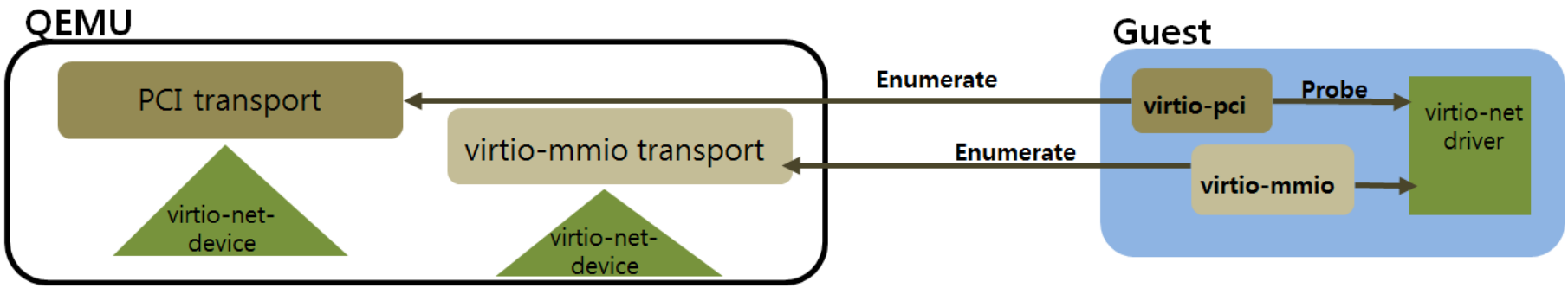
- QEMU enhancements
  - Add device handler – handle – device option
  - Gather device info – create node
  - Add to Guest device tree
  - Guest parses and accesses device
  - From node – i.e. mmio regions, irq, ..
  - SMMU map guest
  - Setup interrupt pass through

Virt Machine Model

- -M virt
  - Kernel builds against “Dummy Virtual Machine” – ARCH_VIRT
  - Supports arm32/arm64 guests
- Instantiates a FDT, no need to pass dtb file
- Defines physical map for
  - Flash – bios
  - GICv2, GICv2m, GCIv3UART, RTC
  - Platform bus device pass-through
  - UART
  - Builds ACPI tables i.e. hw discovery

Virt Machine Model

- virtio_mmio: for virtio transport enable virtio-mmio in kernel
  - The backend is agnostic to transport
  - The guest finds mmio transport



Virt Machine Model

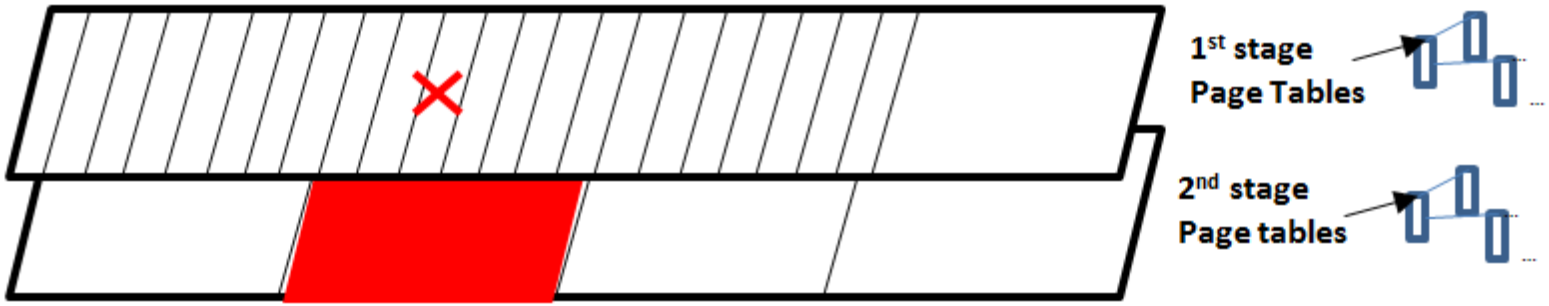
- Boot loaders for arm32/arm64
  - Tiny boot loader support
  - Will boot an Image, Image.gz, zImage, uImage
  - quick boot
  - Few devices emulated - low mmio exits

Several Page Sizes

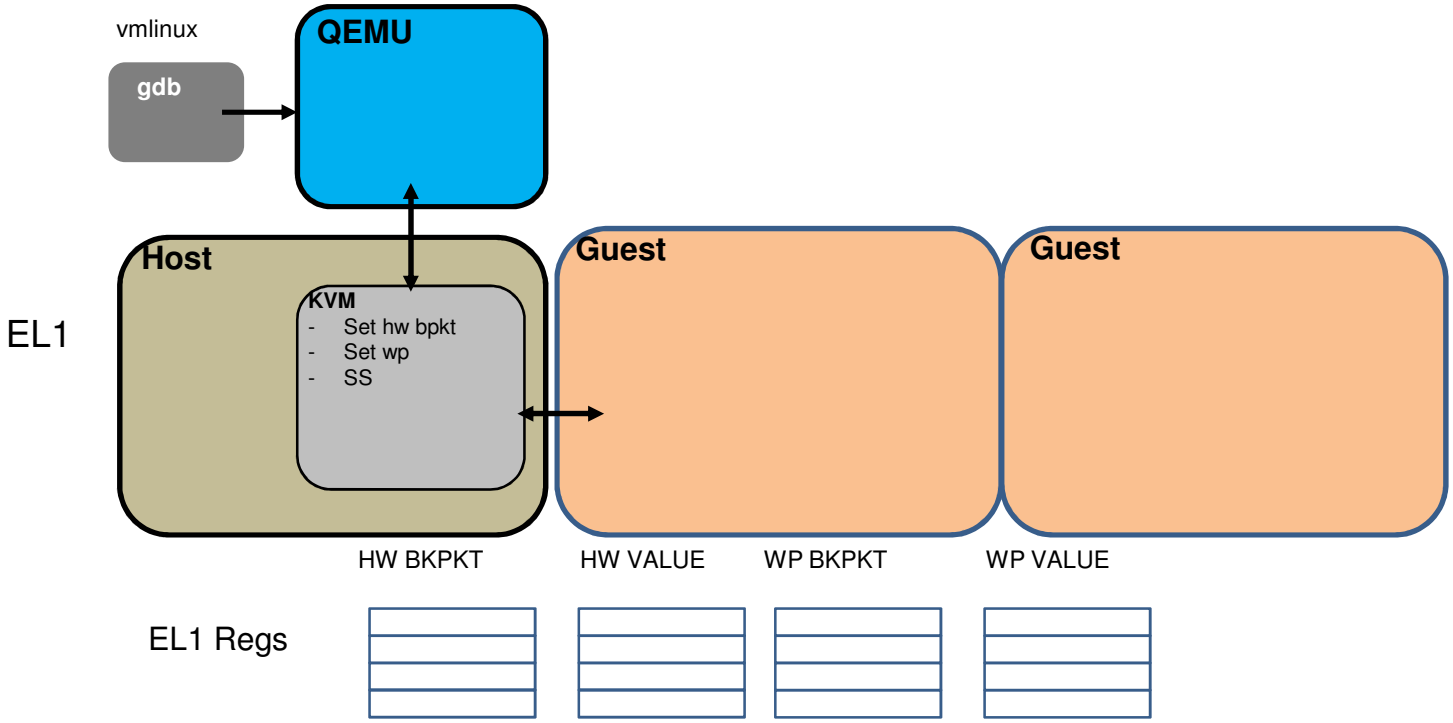
- 4K, 64K – page sizes
  - Huge page – 2MB, 512MB
- Now 16K page size – added
  - Huge Page – 32MB
- More flexibility in the future
  - 4k guest on 64k host – without huge pages
  - Or 16k page guest
  - Good for TLBs

Several Page Sizes

- Live migration & dirty page logging
  - 64k hosts are a good option due to less memory copy



Guest Debug Support



Guest Debug Support

- QEMU has a gdb server (connect -gdb tcp::… , -S stop cpu)
  - gdb <vmlinux> > connect remote:…
- Hyp debug support extensions to trap on debug events
- Arm64 provides a variety of self hosted debug regs
  - Paired HW control and value registers
    - Control – VA, CONTEXID/VMID match
    - Value reg – VA, VMID, CONTEXID
  - Paired watch point control and value regs
    - Control – on load/store, byte selects
    - Value – VA
  - Single Stepping – PSTATE, debug control reg.

Guest Debug Support

- Complex integration into QEMU gdb server infrastructure
  - Accept SS, bkpt, watch point commands
  - Take debug exit on bpkt and return state to QEMU
  - Handle concurrent guest/host QEMU debug

Questions?

Thank You!

Mario Smarduch
Samsung Open Source Group
Senior Virtualization Architect
m.smarduch@samsung.com