KVM on ARM

Zhao Zhenlong
<zhaozhl@inspur.com>

Outline

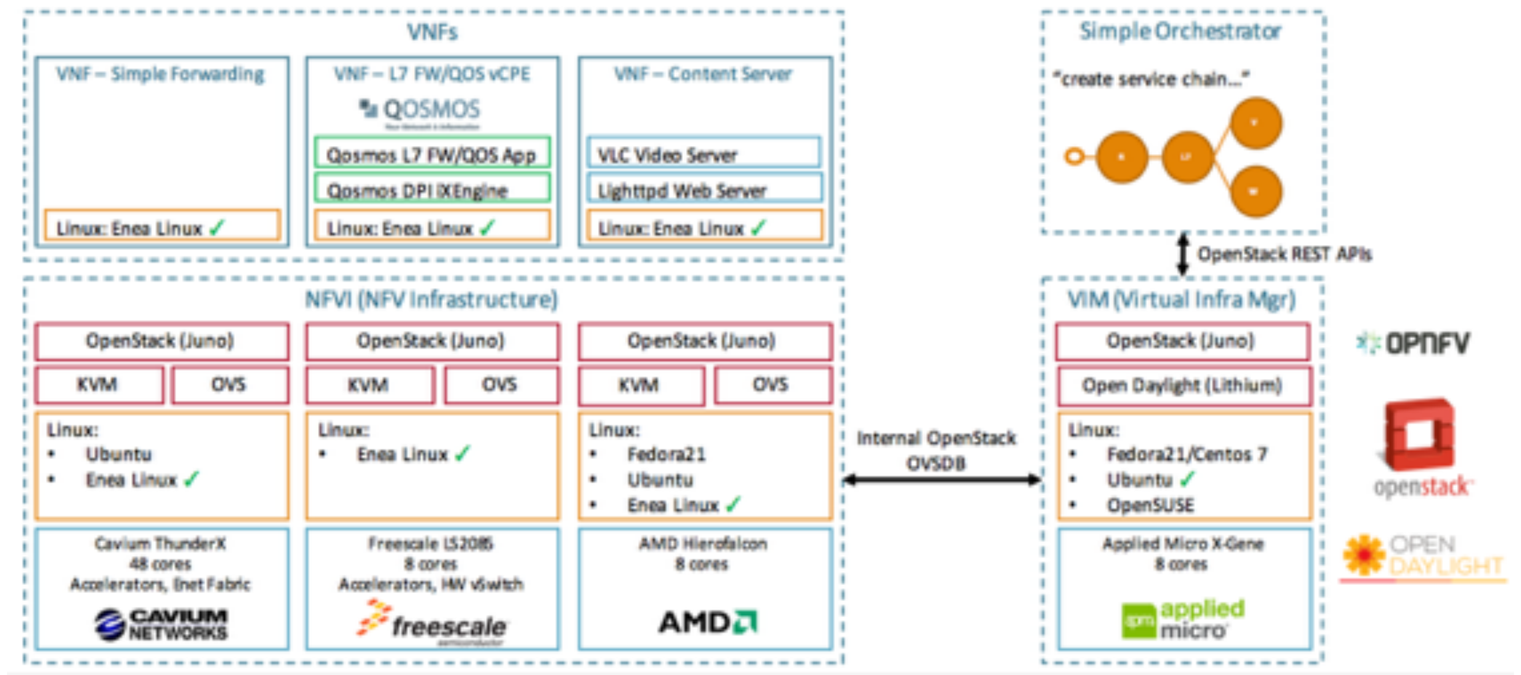
- Background
- ARM Virtualization Extension
- KVM on ARM
- Current State

Background

- More powerful ARM processor
- 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, 1974
- 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

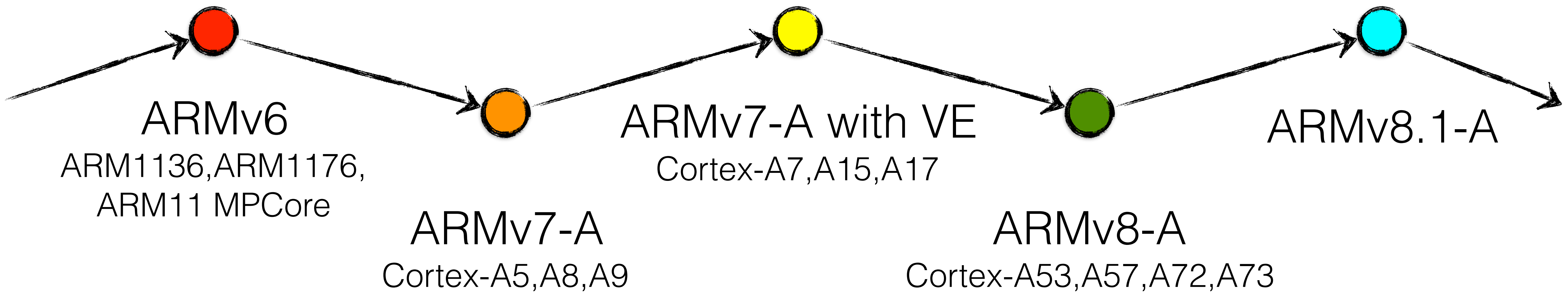
- SMP
- Security extension

- 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

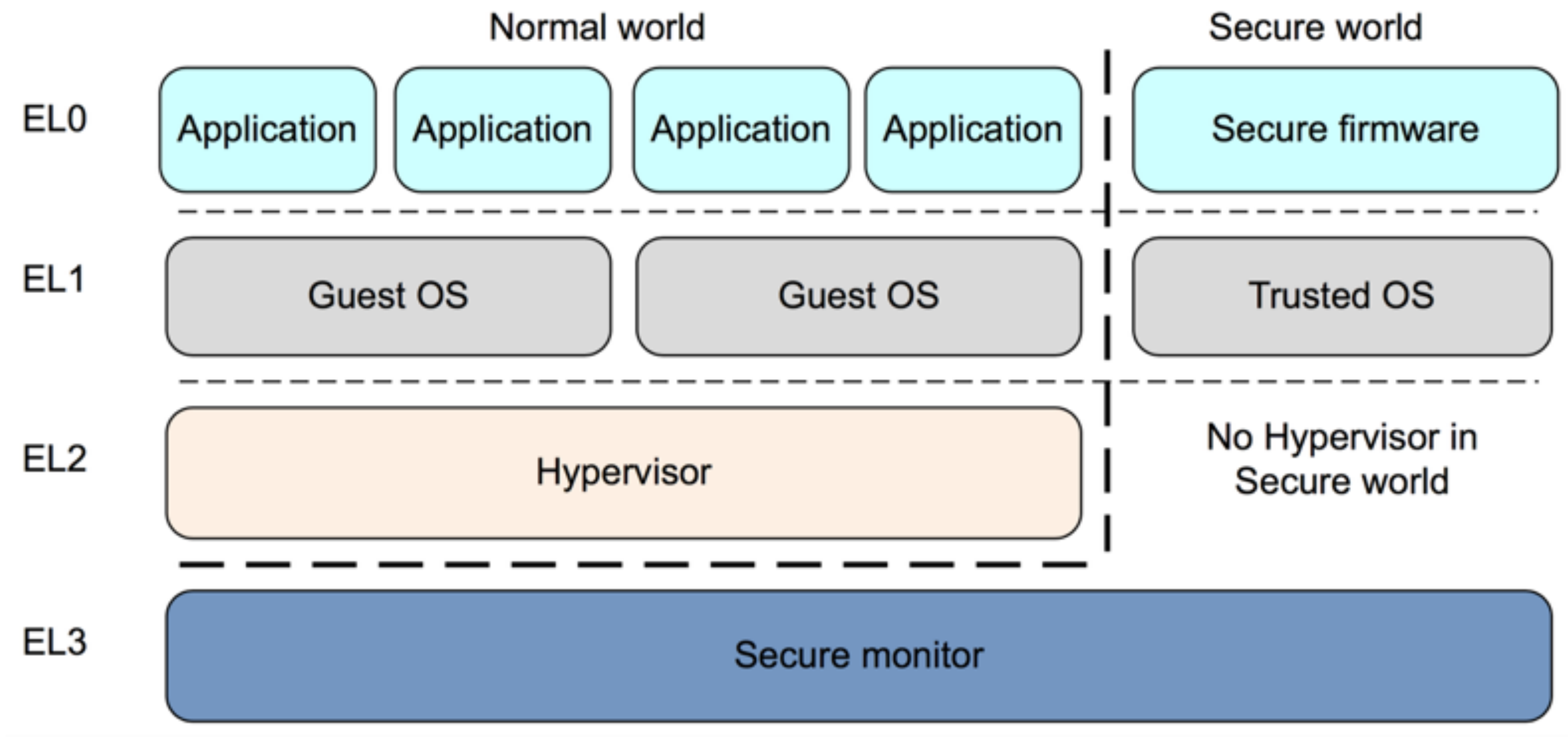
- New atomic inst
- New SIMD inst
- MM improves
- VMIDs expanded
- VHE



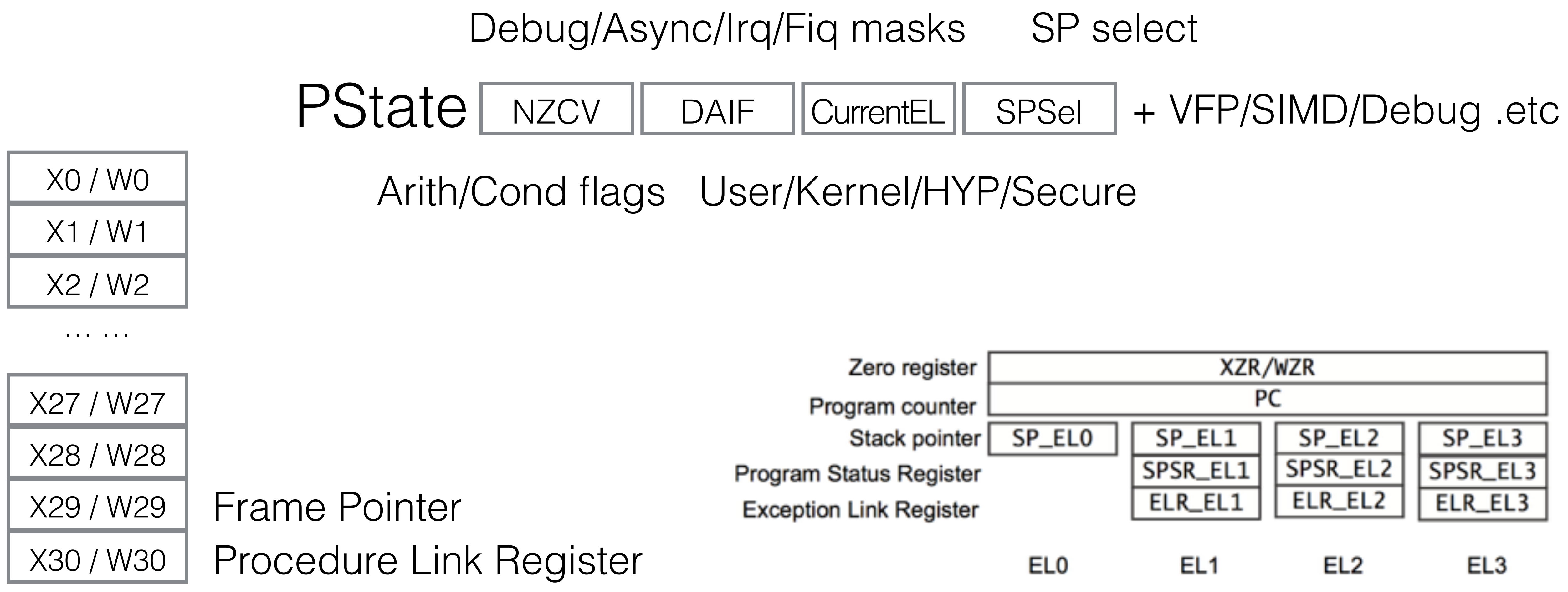
ARM VE

- 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



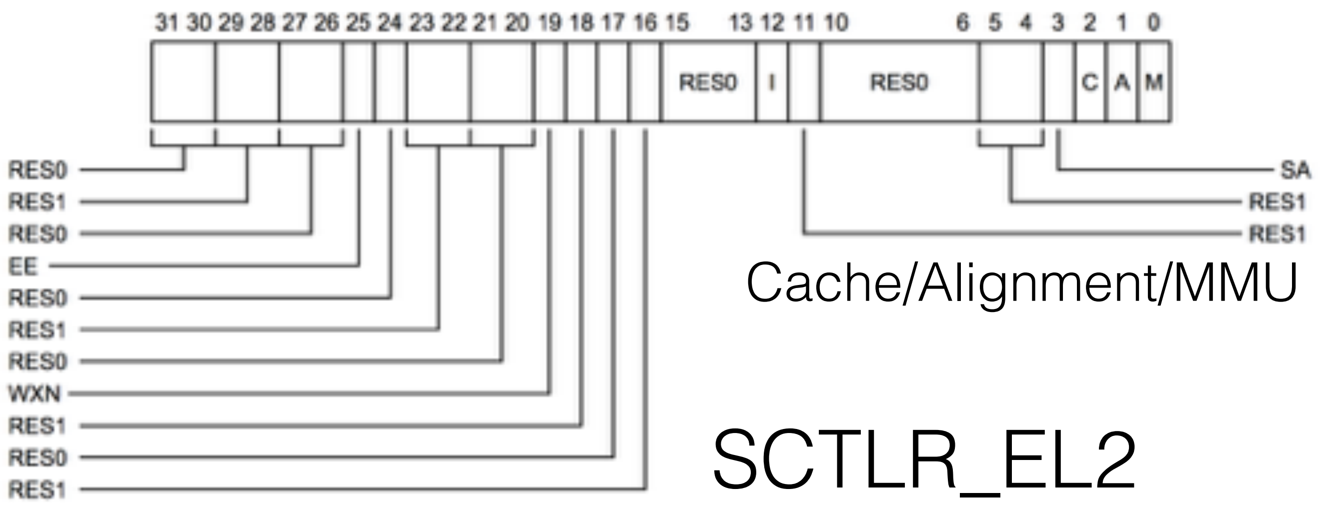
AArch64 Registers



general-purpose register

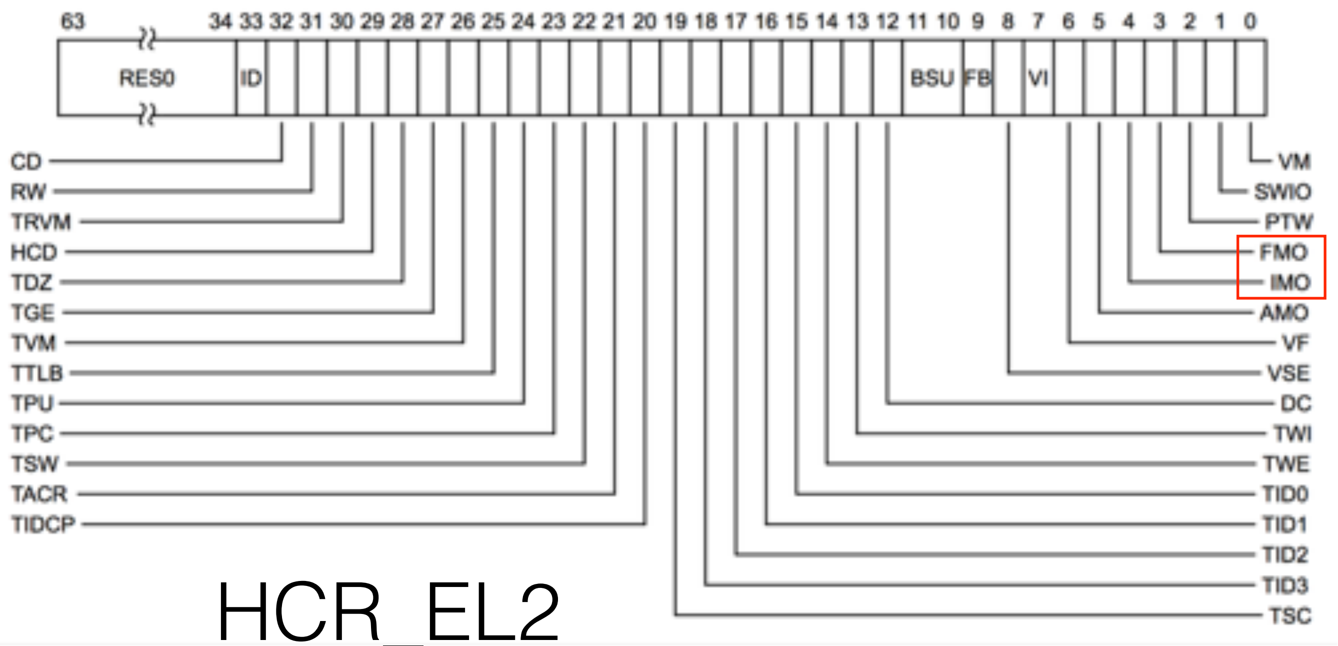
special register

AArch64 Registers Cont.



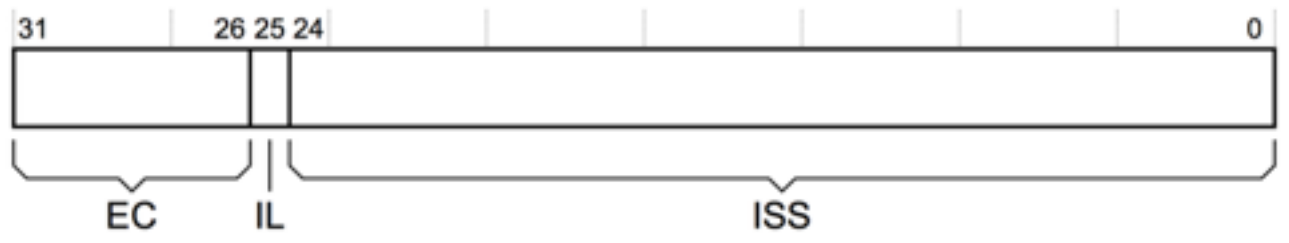
Cache/Alignment/MMU

SCTLR_EL2



HCR_EL2

ESR_ELx



Exception Class : cause of exception
Instruction Length : 0 for16-bit, 1 for 32-bit
Instruction Specific Syndrome : EC information

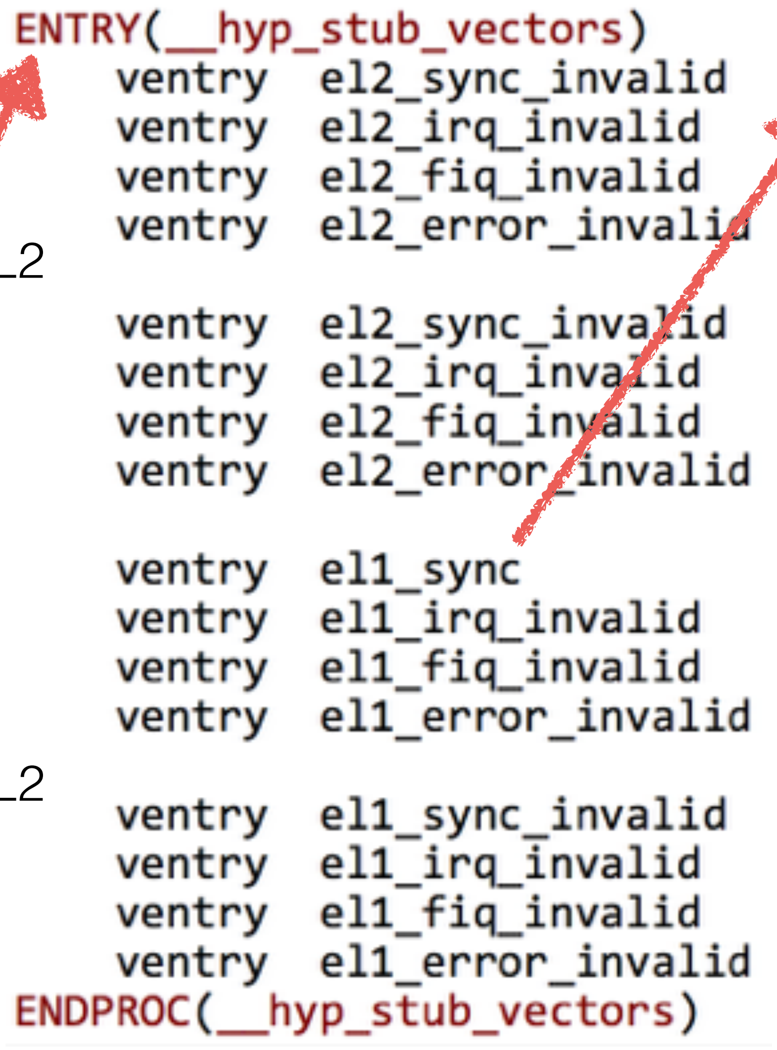
system register

EL2

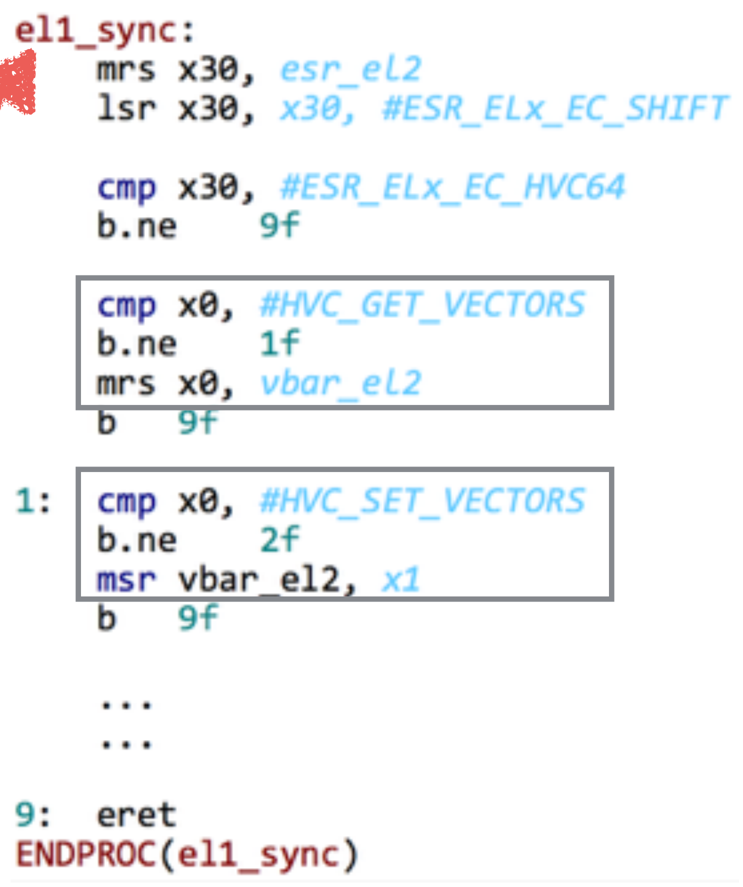
- Access to hypervisor control registers
- Controls stage-2 translations
- Entry through exception
- Exit by exception return

Booting at EL2

- el2_setup in arch/arm64/kernel/head.S
  - if CurrentEL == #CurrentEL_EL2 :
    - if CONFIG_ARM64_VHE :
      - mark #BOOT_CPU_MODE_EL2
    - else :
      - install_el2_stub
        - **set vbar_el2 to __hyp_stub_vectors**
      - mark #BOOT_CPU_MODE_EL2
      - eret back to EL1
  - else :
    - mark #BOOT_CPU_MODE_EL1

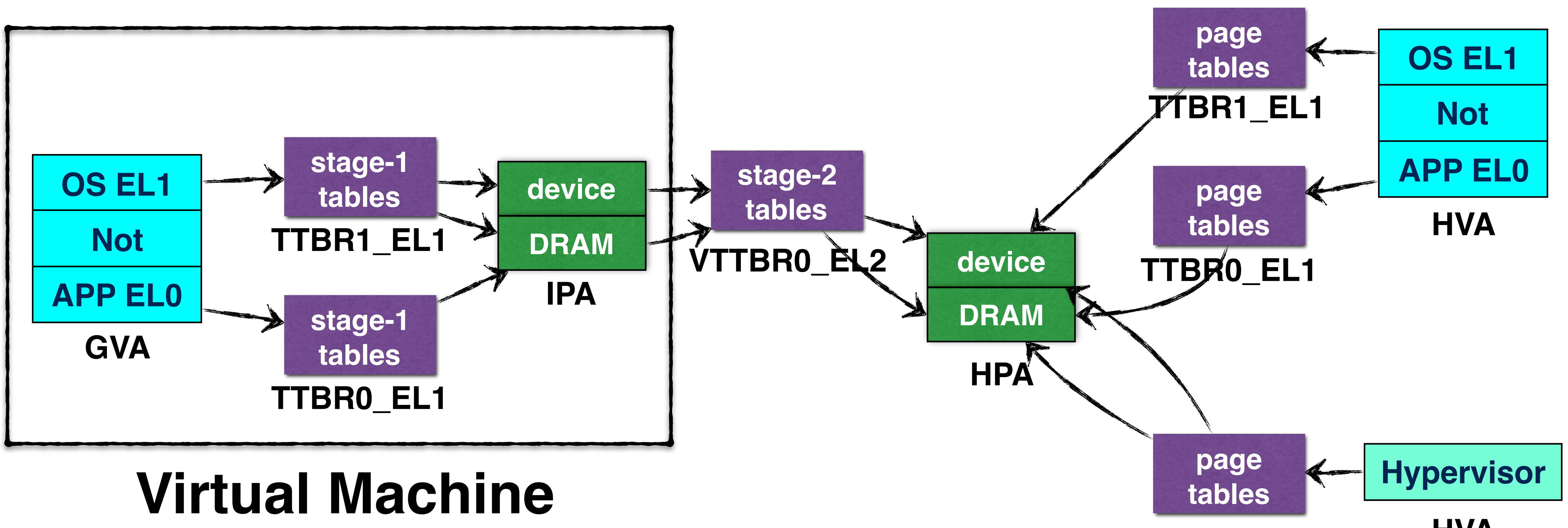


```
ENTRY(__hyp_stub_vectors)
	ventry	el2_sync_invalid
	ventry	el2_irq_invalid
	ventry	el2_fiq_invalid
	ventry	el2_error_invalid

	ventry	el2_sync_invalid
	ventry	el2_irq_invalid
	ventry	el2_fiq_invalid
	ventry	el2_error_invalid

	ventry	el1_sync
	ventry	el1_irq_invalid
	ventry	el1_fiq_invalid
	ventry	el1_error_invalid

	ventry	el1_sync_invalid
	ventry	el1_irq_invalid
	ventry	el1_fiq_invalid
	ventry	el1_error_invalid
ENDPROC(__hyp_stub_vectors)
```



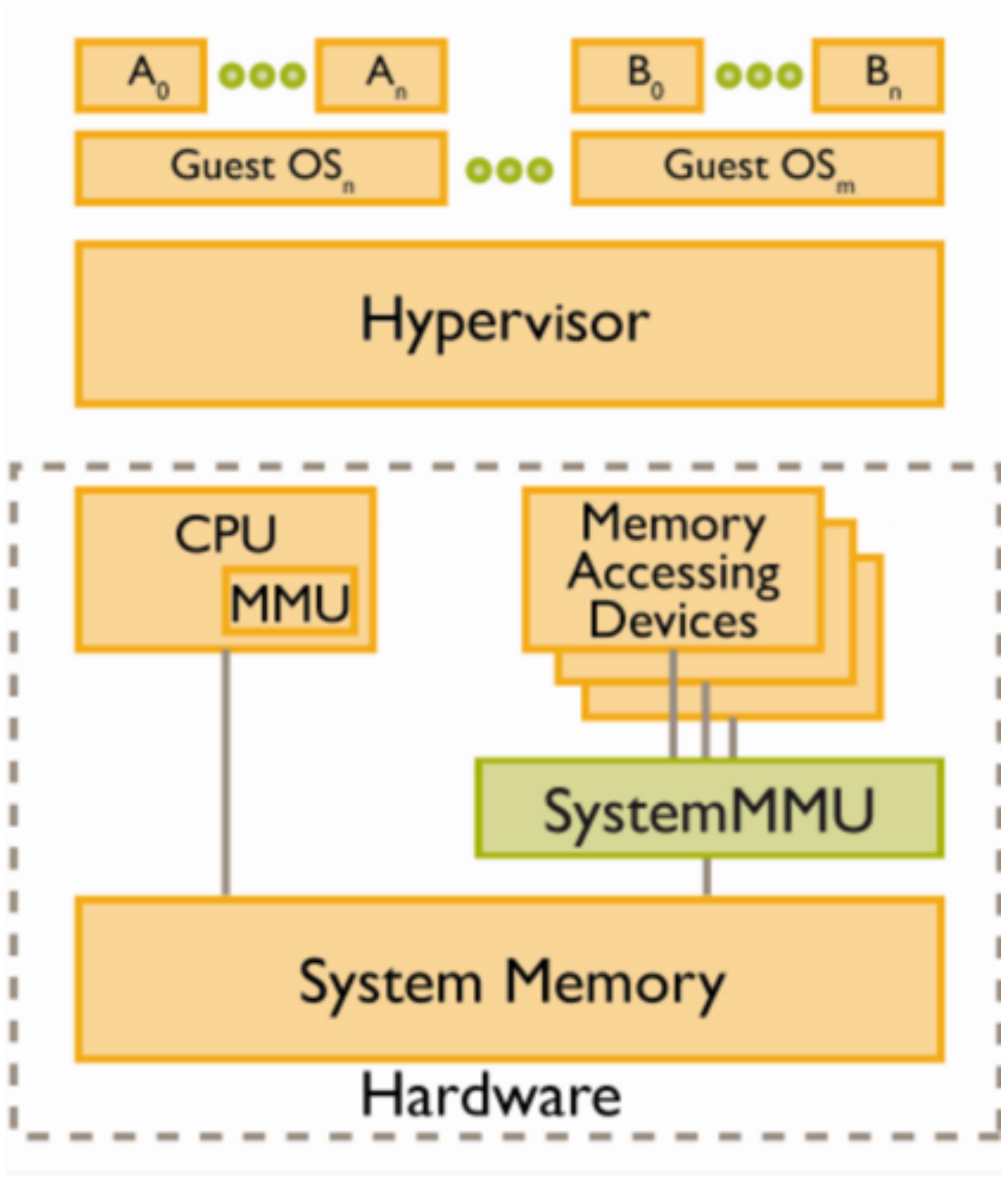
```
el1_sync:
	mrs x30, esr_el2
	lsr x30, x30, #ESR_ELx_EC_SHIFT

	cmp x30, #ESR_ELx_EC_HVC64
	b.ne	9f

	cmp x0, #HVC_GET_VECTORS
	b.ne	1f
	mrs x0, vbar_el2
	b	9f

1:	cmp x0, #HVC_SET_VECTORS
	b.ne	2f
	msr vbar_el2, x1
	b	9f

	...
	...

9:	eret
ENDPROC(el1_sync)
```

Stage-2 Translation



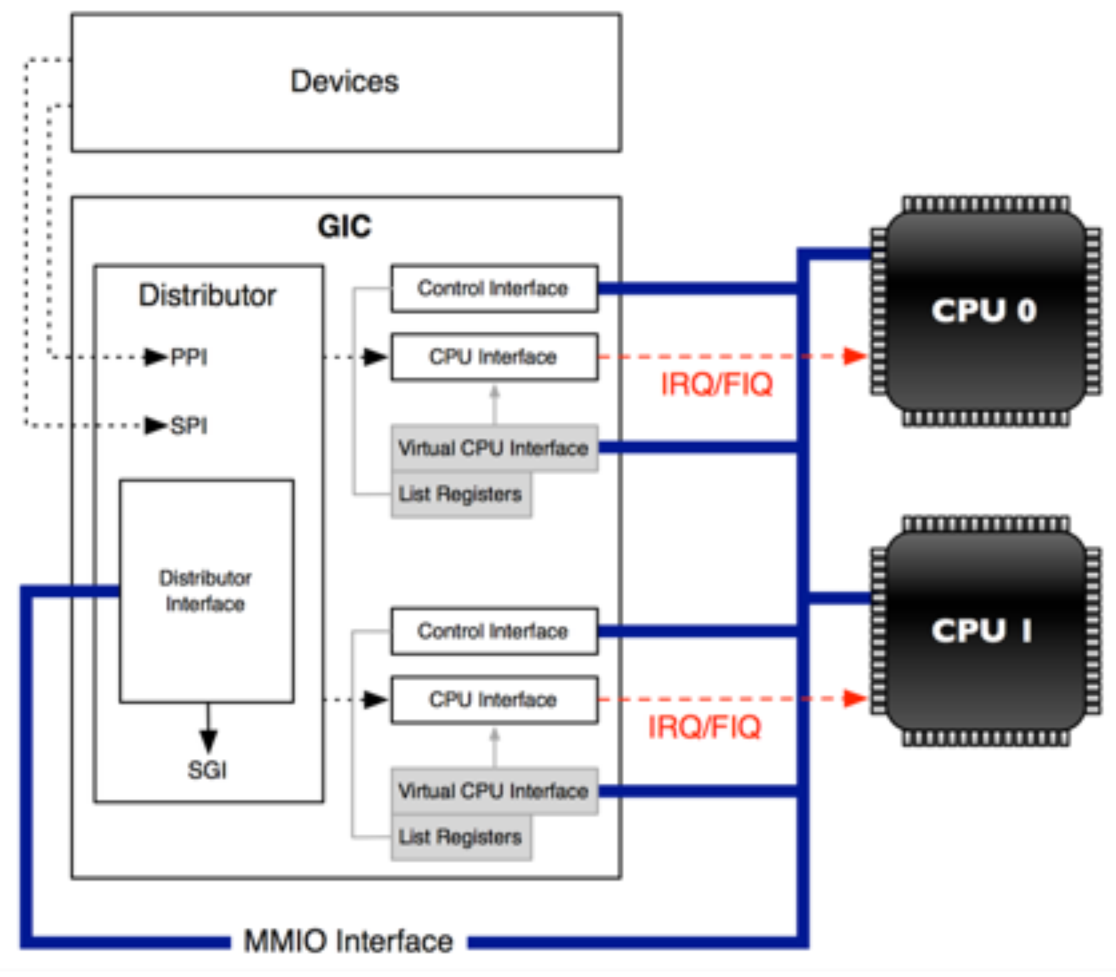
Stage 2 SMMU Translation

- 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



GICv2

- Hypervisor receives hardware interrupts and controls the guest view of injected interrupts using List Registers
- GIC hardware ensures interrupt ACK and EOI is handled without exiting the guest
- GIC v2.0 include hardware virtualization
    - Virtual CPU interface
    - List Register
    - Distributor needs emulation
- 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

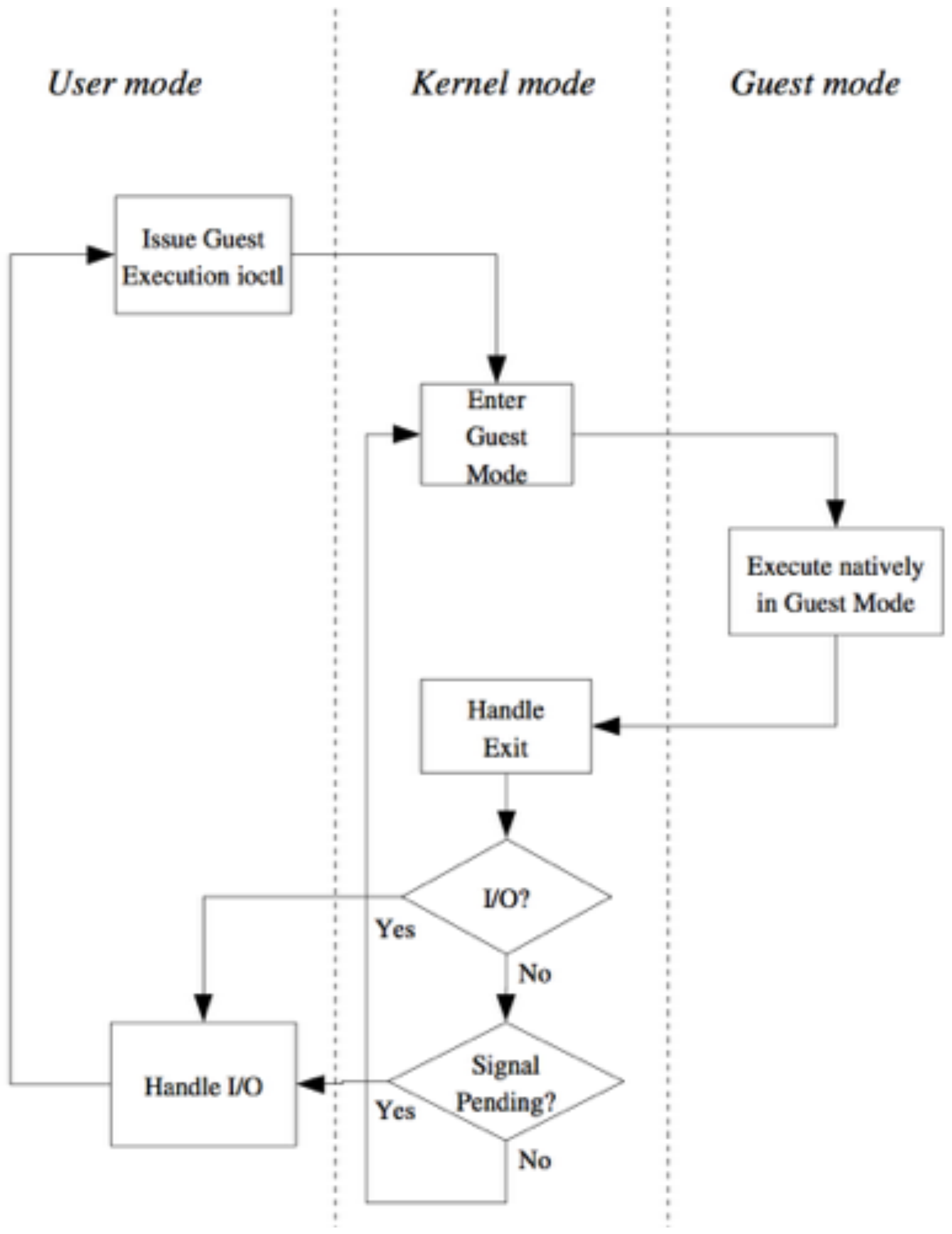
ARM and x86

- Separate mode vs Orthogonal privilege rings
    - EL2 is not a superset of normal privileged EL0/EL1
    - control registers must be programmed in EL2
    - overhead transitioning between EL2 and EL1
    - Linux kernel cannot directly run under EL2
- RISC-style vs CISC-style
    - x86 hardware automatically save to/restore from VMCS when context switching
    - ARM leaves it up to software to decide which state needs to be saved and restored
    - ARM has separate registers in VHE while x86 switching between root and non-root

KVM on ARM

KVM

- 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();



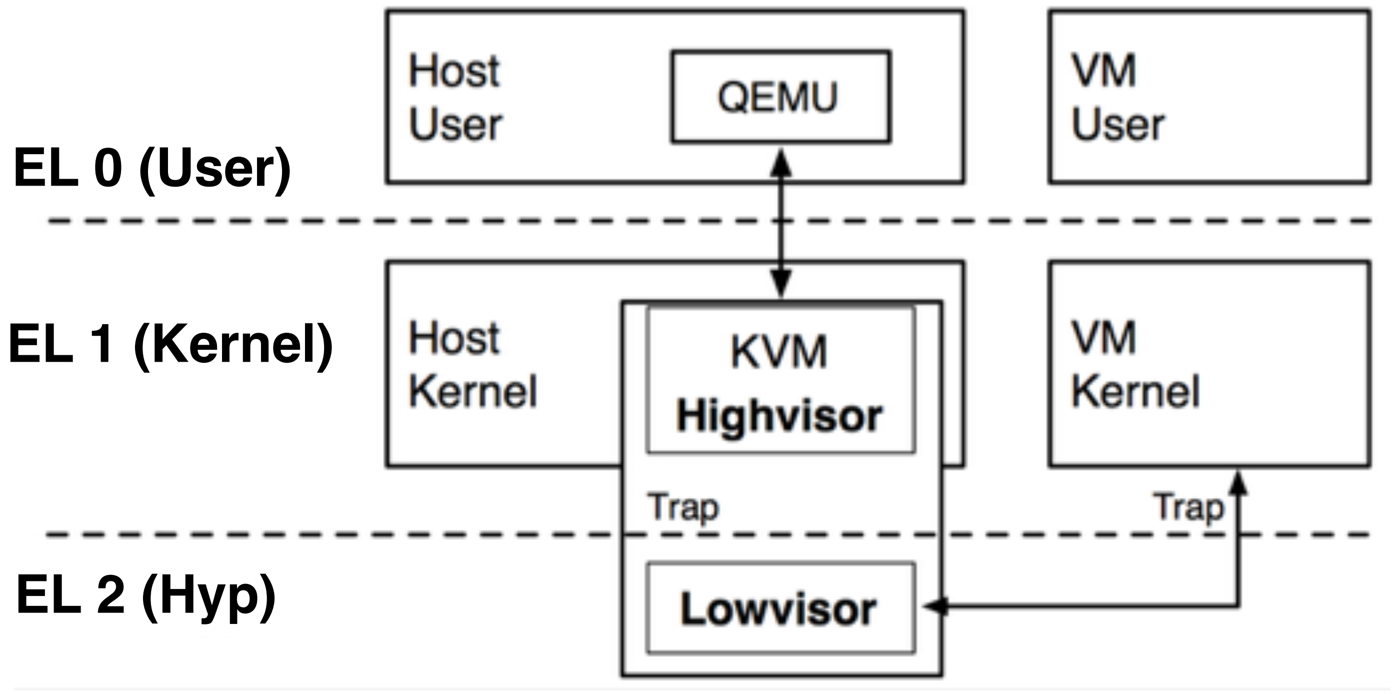
kvm: the Linux Virtual Machine Monitor

EL2 not superset of EL1

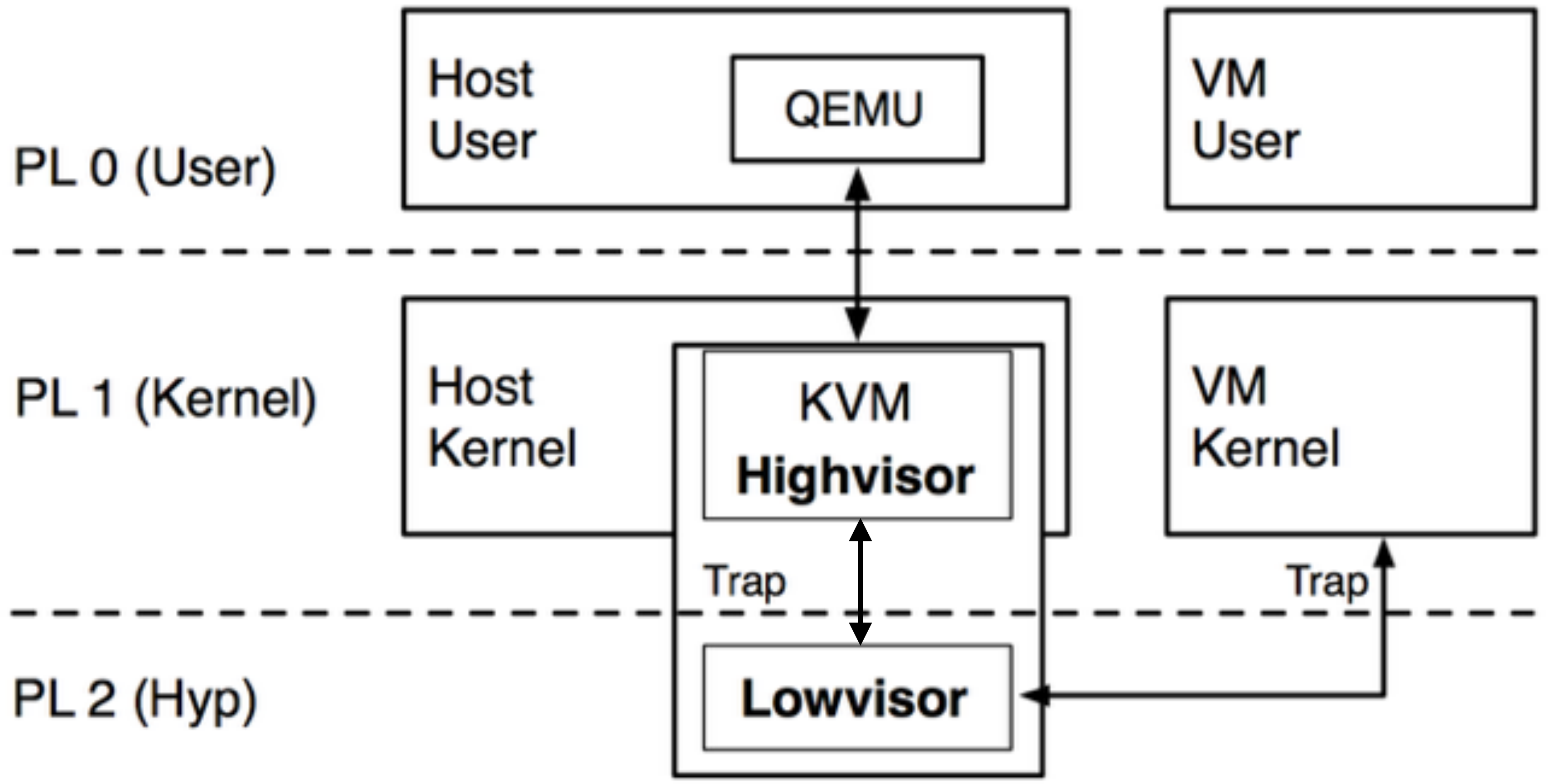
- EL2 is a completely different CPU mode from EL1
    - Different registers
    - Separate address space and different memory model
- Run Linux in EL2 requires too many changes
    - ARM Linux guest kernel should be able to run in EL1
- Potential performance problems in EL2
    - Only one Translation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

Split-Mode Virtualization

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



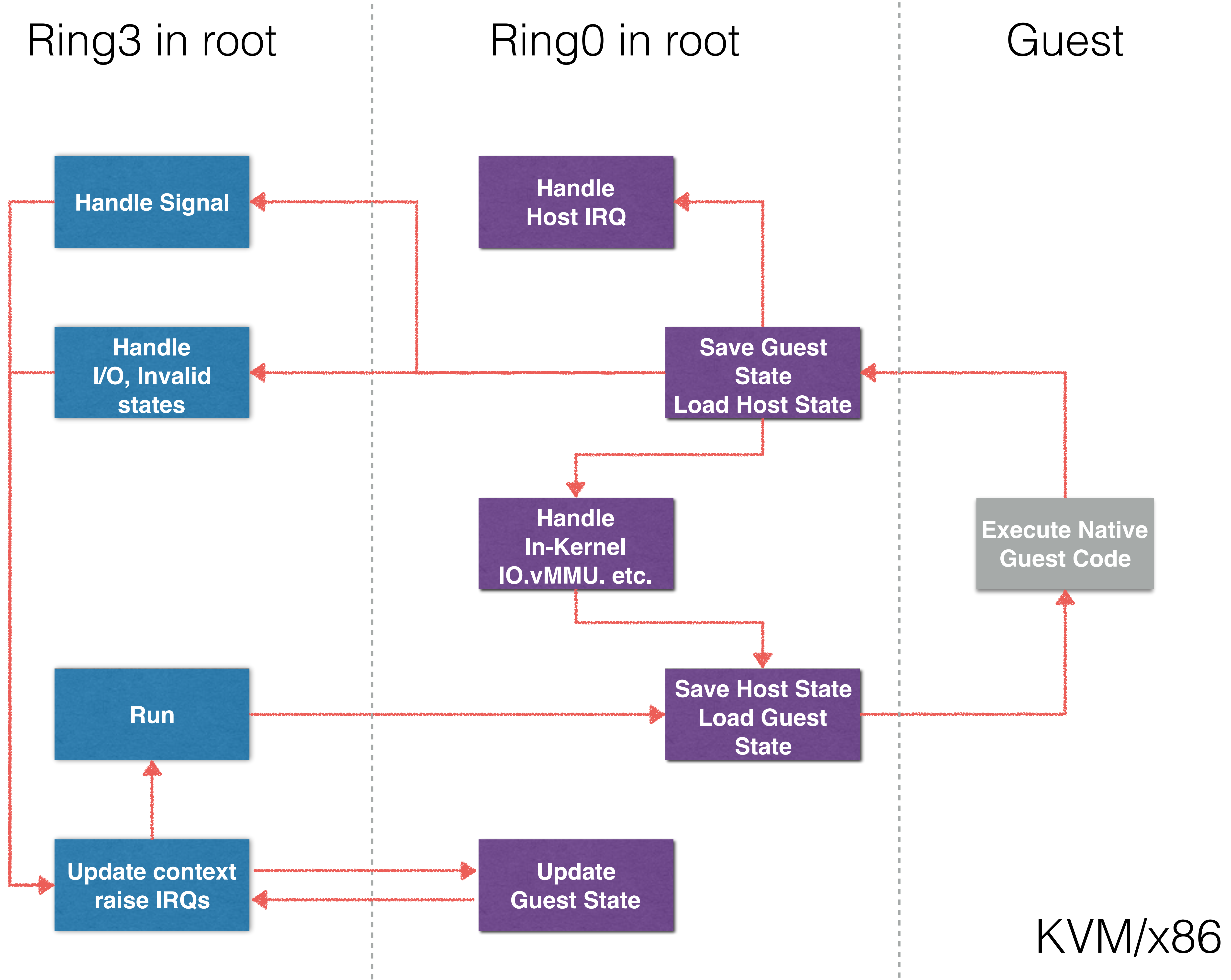
Split-Mode Virtualization



double-trap issue

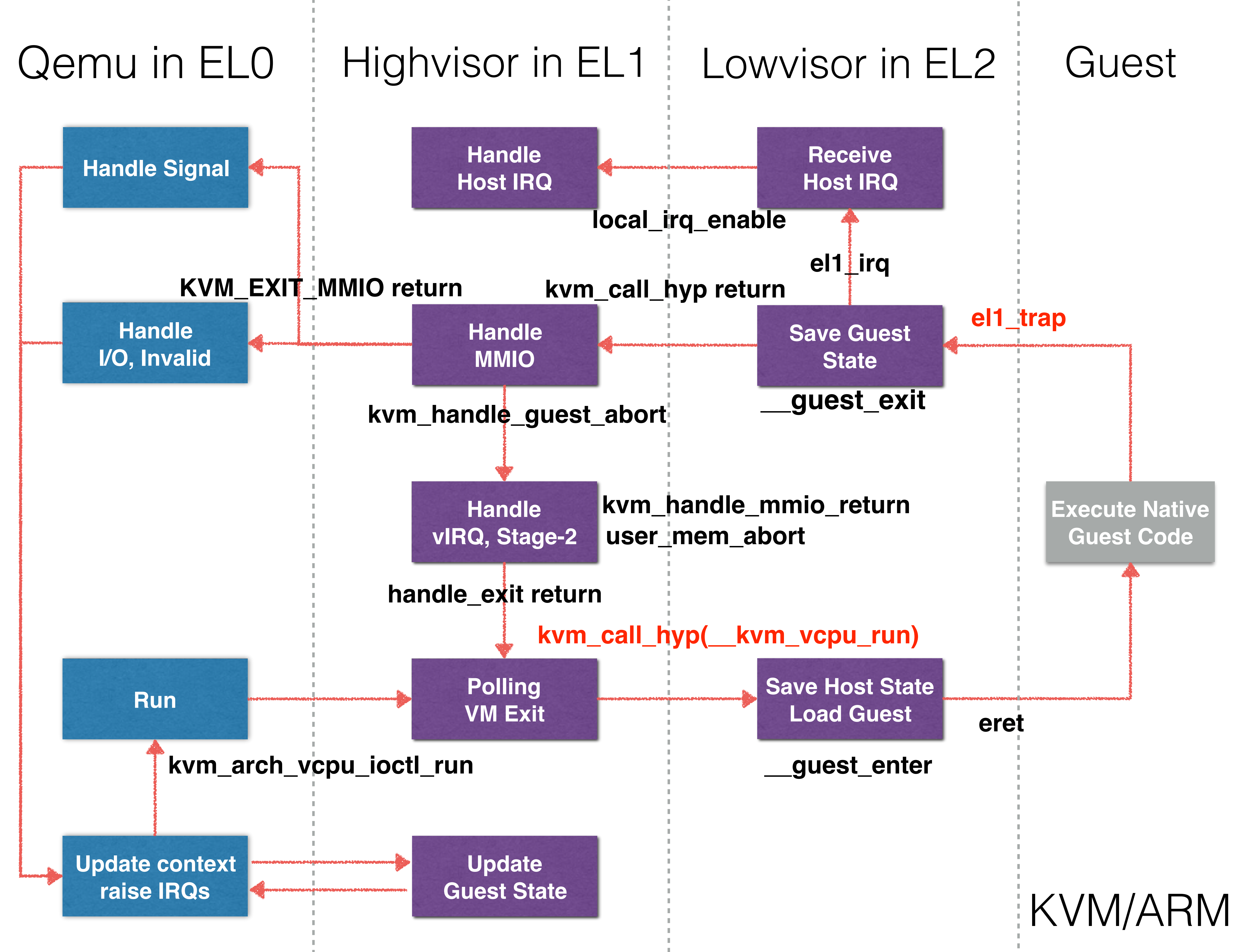
CPU Virtualization

- 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



Ring3 in root
Ring0 in root
Guest
Handle Signal
Handle Host IRQ
Handle I/O, Invalid states
Save Guest State Load Host State
Handle In-Kernel IO.vMMU. etc.
Execute Native Guest Code
Run
Save Host State Load Guest State
Update context raise IRQs
Update Guest State

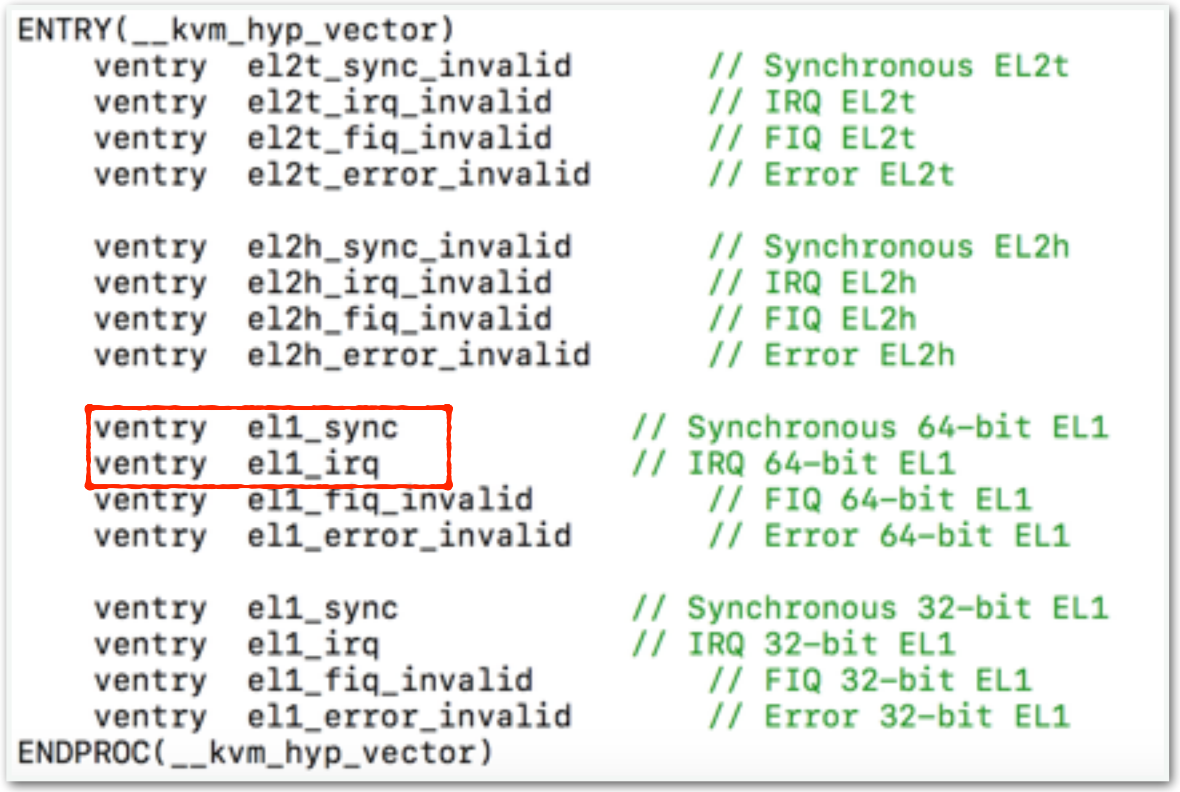
KVM/x86


Qemu in EL0
Highvisor in EL1
Lowvisor in EL2
Guest
Handle Signal
Handle Host IRQ
Receive Host IRQ
local_irq_enable
el1_irq
KVM_EXIT_MMIO return
kvm_call_hyp return
el1_trap
Handle I/O, Invalid
Handle MMIO
Save Guest State
__guest_exit
kvm_handle_guest_abort
Handle vIRQ, Stage-2
kvm_handle_mmio_return
user_mem_abort
Execute Native Guest Code
handle_exit return
kvm_call_hyp(__kvm_vcpu_run)
Run
Polling VM Exit
Save Host State Load Guest
eret
kvm_arch_vcpu_ioctl_run
__guest_enter
Update context raise IRQs
Update Guest State
KVM/ARM

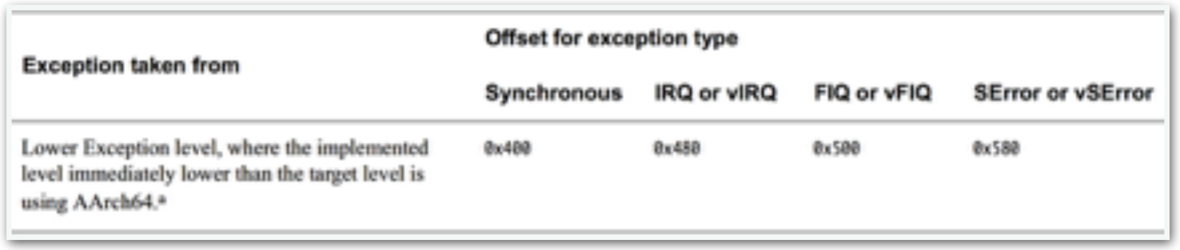
Guest Exit Events

- 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 / SMC instruction
  - hypercalls / 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

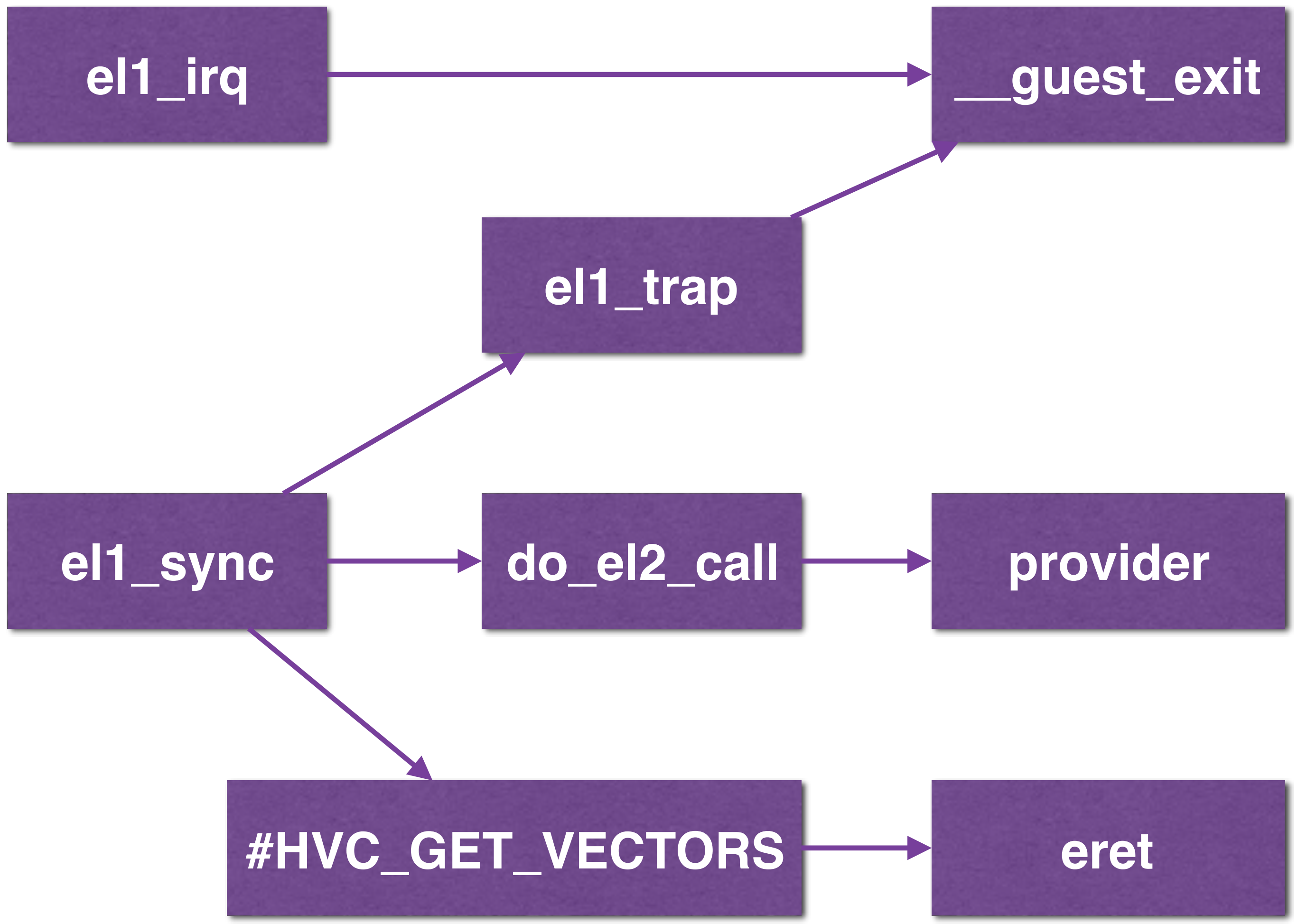
Exception Handling



```
ENTRY(__kvm_hyp_vector)
    ventry  el2t_sync_invalid       // Synchronous EL2t
    ventry  el2t_irq_invalid        // IRQ EL2t
    ventry  el2t_fiq_invalid        // FIQ EL2t
    ventry  el2t_error_invalid      // Error EL2t

    ventry  el2h_sync_invalid       // Synchronous EL2h
    ventry  el2h_irq_invalid        // IRQ EL2h
    ventry  el2h_fiq_invalid        // FIQ EL2h
    ventry  el2h_error_invalid      // Error EL2h

    ventry  el1_sync            // Synchronous 64-bit EL1
    ventry  el1_irq             // IRQ 64-bit EL1
    ventry  el1_fiq_invalid         // FIQ 64-bit EL1
    ventry  el1_error_invalid       // Error 64-bit EL1

    ventry  el1_sync            // Synchronous 32-bit EL1
    ventry  el1_irq             // IRQ 32-bit EL1
    ventry  el1_fiq_invalid         // FIQ 32-bit EL1
    ventry  el1_error_invalid       // Error 32-bit EL1
ENDPROC(__kvm_hyp_vector)
```



Exception taken from	Offset for exception type			
	Synchronous	IRQ or vIRQ	FIQ or vFIQ	SError or vSError
Lower Exception level, where the implemented level immediately lower than the target level is using AArch64.ᵃ	0x400	0x480	0x500	0x580



Hypervisor to VM Switch

- store all highvisor GP registers on the Hyp stack
- save all highvisor-specific configuration registers onto the Hyp stack
- configure Hyp mode to trap VM exit events
- set VTTBR and enable Stage-2 address translation
- configure the VGIC and timer for the VM
- load the VM’s configuration registers onto the hardware
- 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
    - EC = 0x24(ESR_ELx_EC_DABT_LOW) in ESR
        - missing stage-2 translation table entry
- Capture fault information
    - vcpu->arch.fault.hpfar_el2 / ESR_ELx_WNR
- Create stage-2 mapping
    - kvm_handle_guest_abort
    - if not MMIO, fix it using **user_mem_abort(vcpu, fault_ipa, memslot, hva, fault_status)**
- Run vcpu

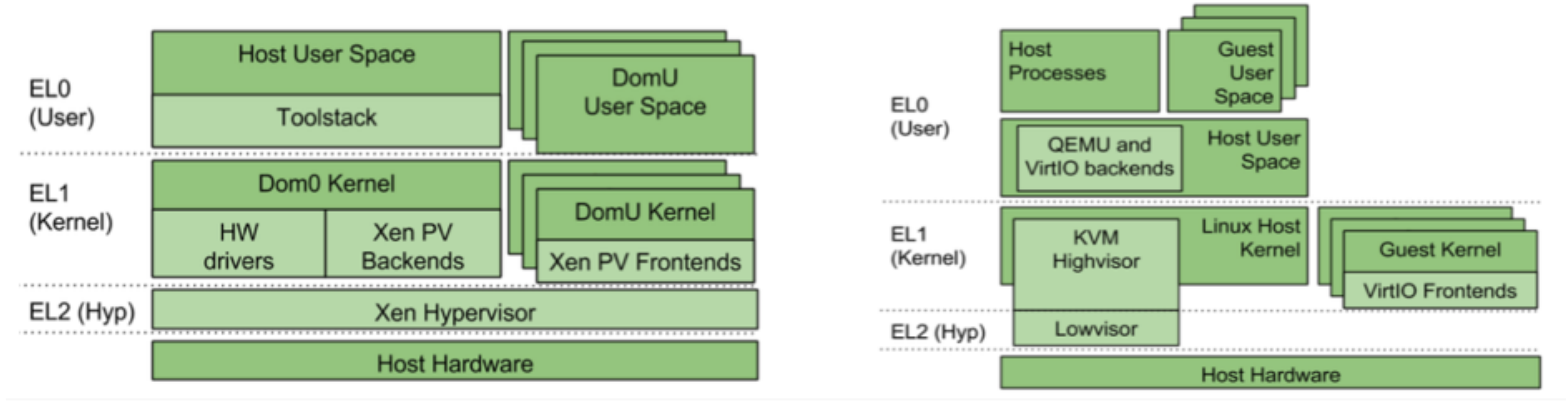
I/O Virtualization

- All I/O are load/store to MMIO device regions
- Stage-2 tranlation fault
    - io_mem_abort(vcpu, run, fault_ipa)
        - not in RAM considered I/O operation
        - emulations in kernel-space (e.g. vgic-v2/v3)
        - fault address can be routed to user-space
- ESR_EL2 provides emulation hints
    - decode_hsr(vcpu, &is_write, &len)

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



- Xen hypercall
  - HVC Instruction
  - Xen handler
  - Return to VM

- KVM hypercall
  - HVC Instruction
    - Switch to EL1
    - Return to host kernel
  - KVM handler
    - HVC Instruction
    - Switch to EL2
  - Return to VM

Scheduling

	KVM	**Xen**
save/ restore overhead	heavy host state	light-weight hypervisor state
VM-to-VM switch	trap to EL2, switch to Host OS trap to EL2, switch to Guest OS	trap to EL2 switch two VMs' EL1 context
scheduler position	call KVM_RUN usually using pthread kernel scheduler in EL1	Xen own scheduler in EL2
scheduling mechanism	top-to-bottom polling based active scheduling	bottom-to-top event driven passive scheduling

IO Performance

	KVM	Xen
device emulation	direct access hardware and VM memory	in Dom0 with virtualized memory
vGIC emulation	in EL1 with double trap	direct in EL2
output path	traps to EL2, then to EL1, sync send data (more fast)	traps to EL2, async signal Dom0 send data, slow signaling between VMs
input path	kernel receives data and wakes up receiving VM's vcpu thread by physical IPI	traps from Dom0 in EL1 to EL2, signaling VM to receive data by physical IPI

Current State

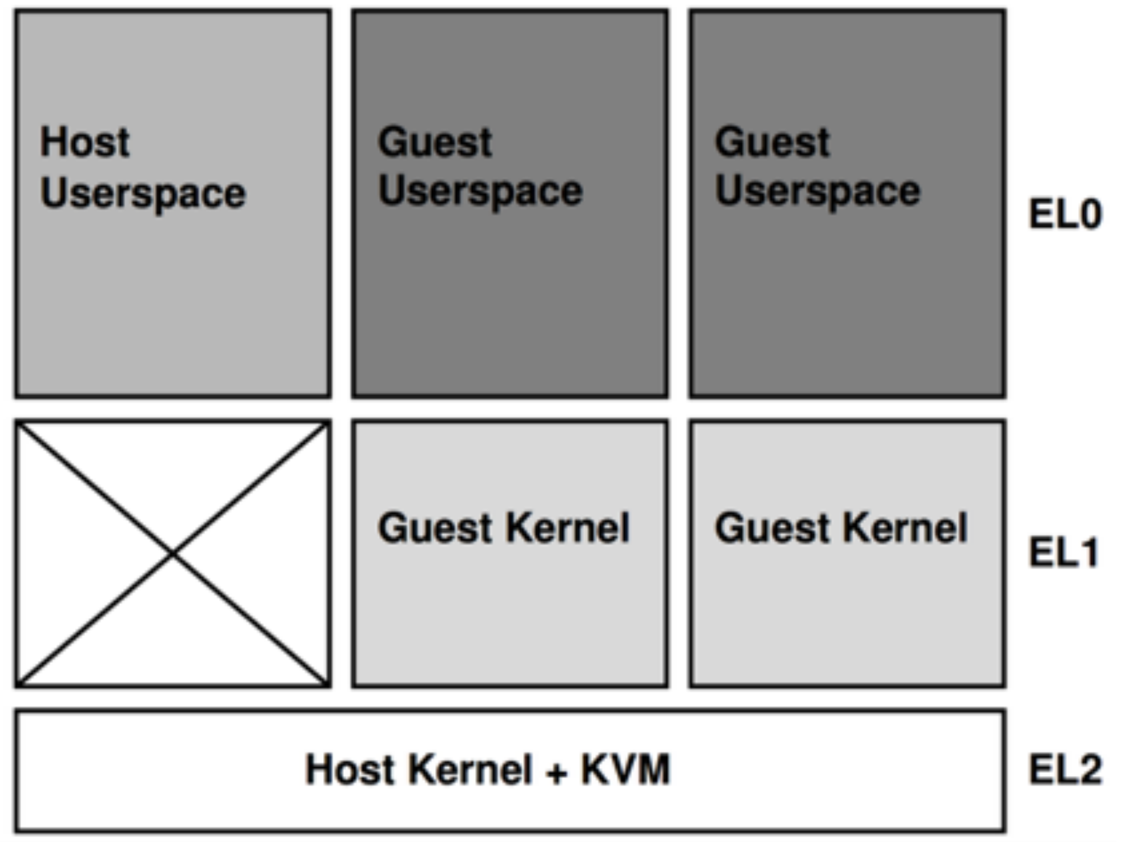
State of KVM/ARM

- 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
    - E2H bit
- EL2 becomes a strict superset of EL1
    - almost no system register sharing reduces switch overhead
    - some trap become functions call
- Access extra EL2 registers transparently
    - actually EL2 registers when access EL1 registers
- Expand memory translation capabilities of EL2
    - no need HYP mappings
- host OS to be run at EL2 with minimal changes
    - but still lot of work to do



References

- mainline kernel source code (current 4.8)
- ARM® Cortex®-A Series Programmer's Guide for ARMv8-A
- ARM® ARM ARMv8, for ARMv8-A architecture profile
- KVM/ARM: Experiences Building the Linux ARM Hypervisor, ASPLOS'14
- ARM Virtualization: Performance and Architectural Implications, ISCA'16
- Virtualization and the ARM architecture, XDS'15 by Marc Zyngier
- KVM/arm64 Architectural Evolutions, LCA'15 by Marc Zyngier
- kvm: the Linux Virtual Machine Monitor, OLS'07 by Avi Kivity

Thanks