



Virtualization in the ARM Architecture
Lecture for the Embedded Systems Course
CSD, University of Crete (May 20 & 23, 2022)

▶ Manolis Marazakis (maraz@ics.forth.gr)



Institute of Computer Science (ICS)
Foundation for Research and Technology – Hellas (FORTH)

Essentials of a hypervisor

- ▶ Parent partition (minimum-footprint OS) + Hypervisor
- ▶ **Hypervisor: Thin layer of software running on the hardware**
 - ▶ Supports creation of *partitions (virtual machines)*
 - ▶ Each partition has one or more *virtual processors*
 - ▶ Partitions can own or share hardware resources
- ▶ **Enforces memory access rules**
- ▶ **Enforces policy for CPU usage**
 - ▶ Virtual processors are scheduled on real processors
- ▶ **Enforces ownership of other devices**
- ▶ **Provides inter-partition messaging**
 - ▶ Messages appear as interrupts
- ▶ **Exposes simple programmatic interface: “hypercalls”**

Hypervisor functions (recap)

- ▶ Memory management
- ▶ Device emulation
- ▶ Device assignment
- ▶ Exception handling
- ▶ Instruction trapping
- ▶ Managing virtual exceptions
- ▶ Interrupt controller management
- ▶ Context switching
- ▶ Memory translation
- ▶ Managing multiple virtual address spaces

Traditional ARM Architecture: ARM11 (ARMv6)

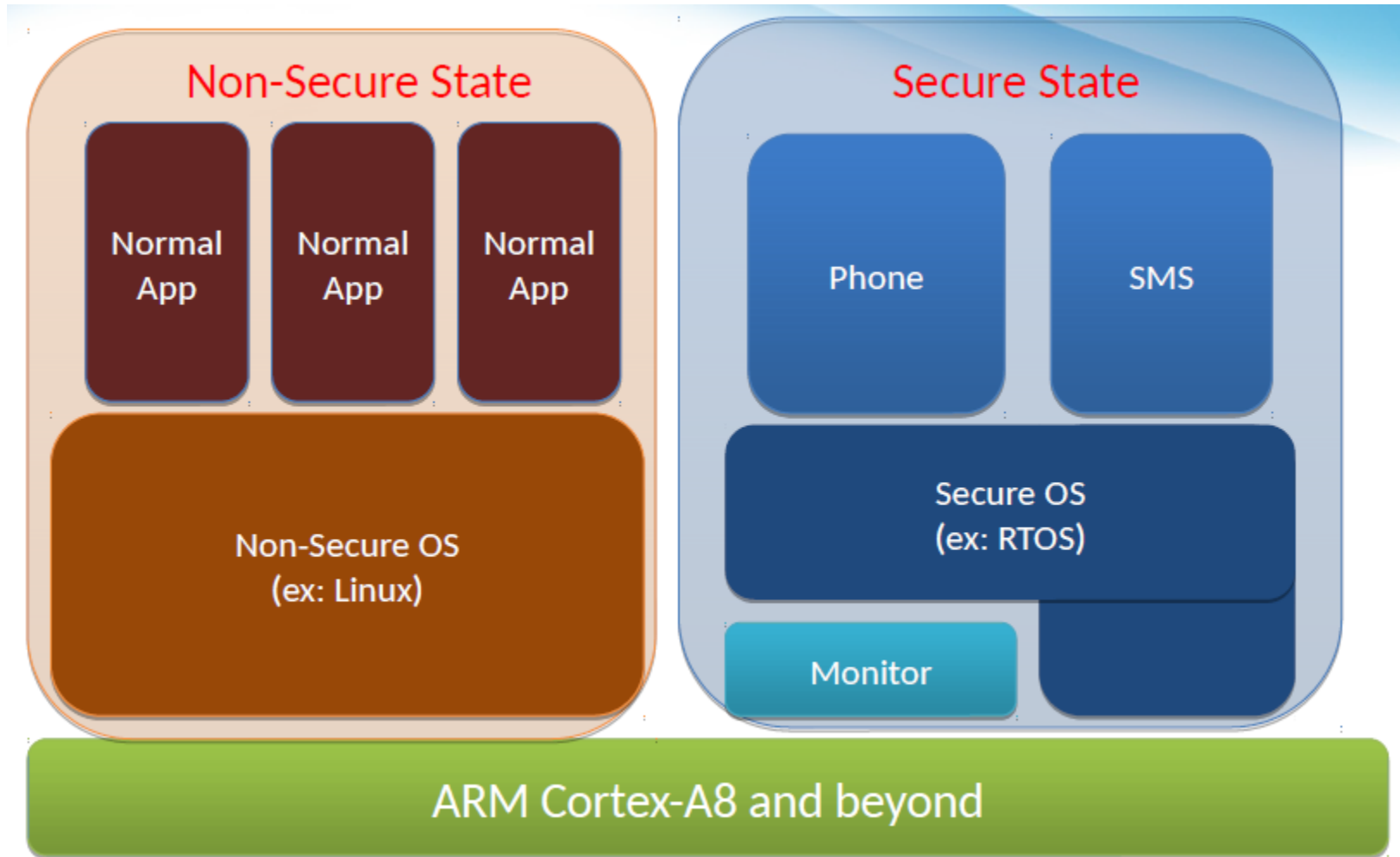
- ▶ Privilege Level 0
 - ▶ User mode
- ▶ Privilege Level 1
 - ▶ System
 - ▶ IRQ
 - ▶ FIQ
 - ▶ Supervisor
 - ▶ Abort

Virtualization Extensions (CPU, Memory, I/O) on the ARM architecture are based on Security Extensions (“Trust Zone”).

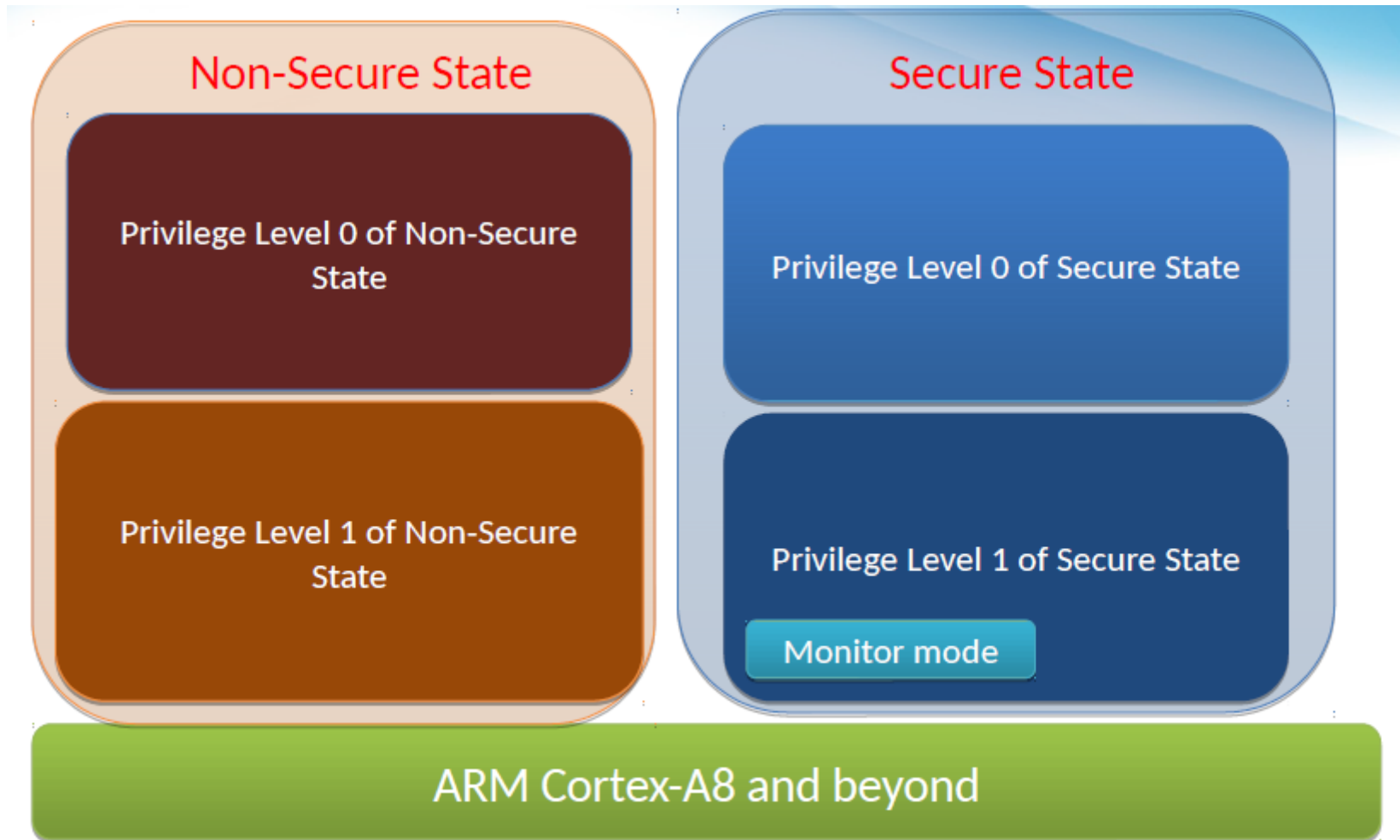
Overview of ARM Virtualization Extension

- ▶ Based on ARM TrustZone (TZ) Security Architecture
- ▶ CPU: new mode (HYP), and new Privilege Level (non-secure privilege level 2)
 - ▶ Support for sensitive instructions
 - ▶ Multiple interrupt vector tables
 - ▶ Hypervisor call
- ▶ Memory: Intermediate Physical Address (IPA)
 - ▶ Guest OS cannot access physical address space directly.
- ▶ I/O: Virtual Generic Interrupt Controller (vGIC)
 - ▶ Faster delivery of interrupts to VMs

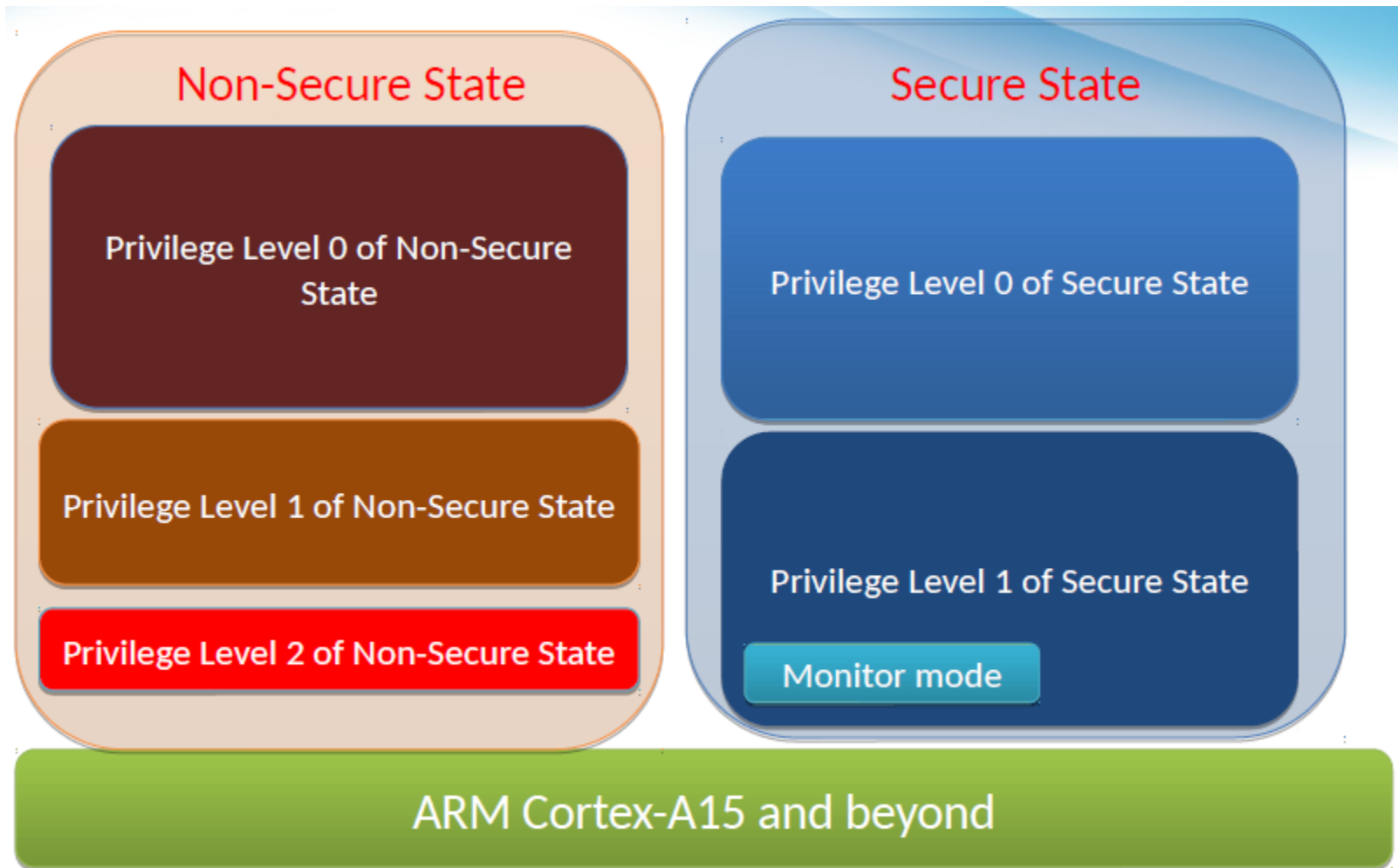
Security Extensions (“TrustZone”)



Privilege Levels in TrustZone [1/2]



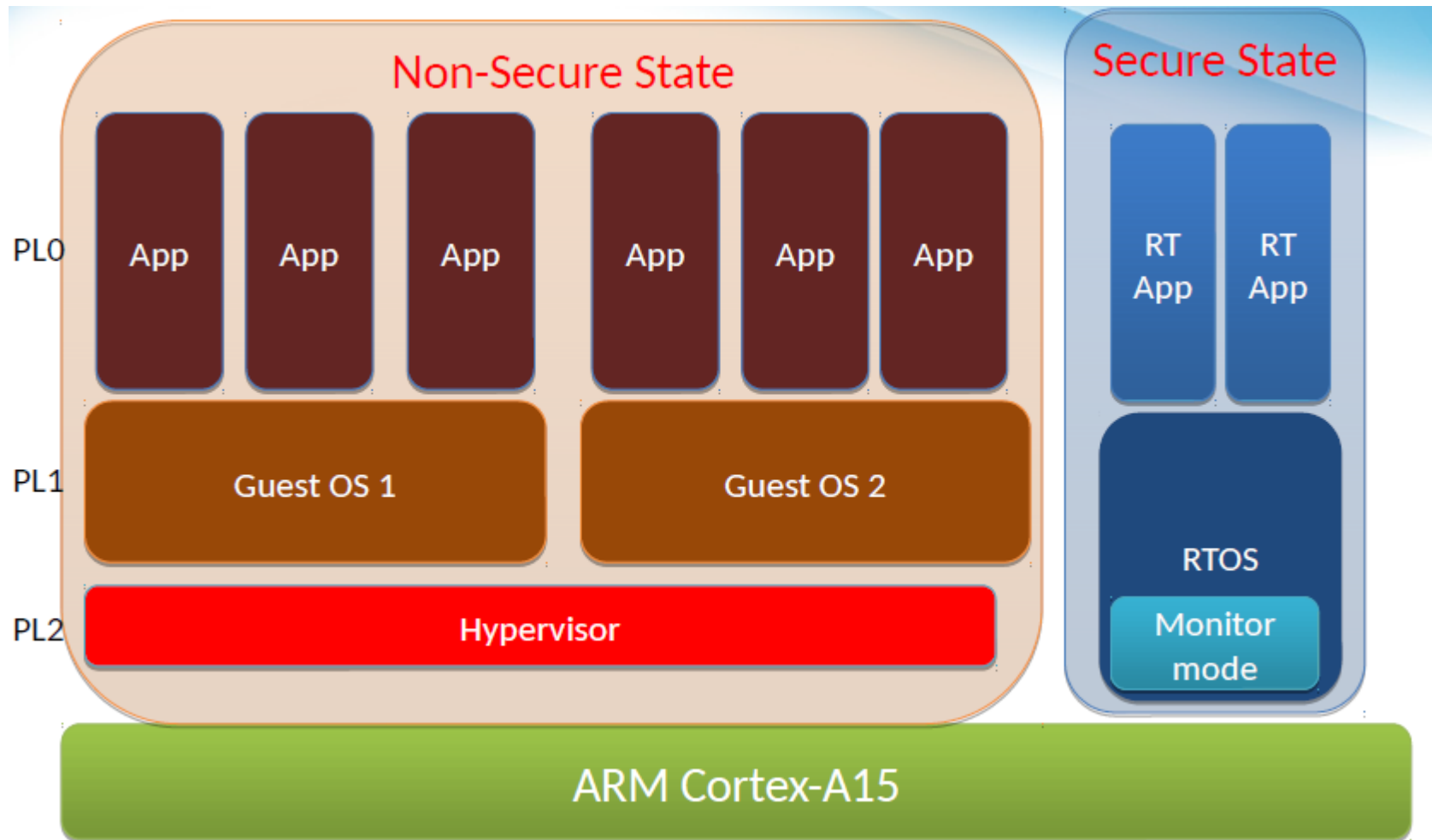
Privilege Levels in TrustZone [2/2]



Virtualization extensions to the ARMv7-A architecture

- ▶ Virtualization extensions to the ARMv7-A architecture:
 - ▶ Available in Cortex A-15 / A-7 CPUs
 - ▶ Hyp - New privilege level (for hypervisor)
 - ▶ GuestOS: SVC privilege level, Applications: USR privilege level
 - ▶ 2-stage address translation (for OS and hypervisor levels)
 - ▶ Complementary to TrustZone security extensions
- ▶ Mechanisms to minimize hypervisor intervention for “routine” GuestOS tasks:
 - ▶ Page table management
 - ▶ Interrupt masking & Communication with the interrupt controller (GIC)
 - ▶ Device drivers (hypervisor memory relocation)
 - ▶ Emulation of Load/Store accesses and trapped instructions
 - ▶ Hypervisor Syndrome Register: Hyp mode entry reason (syndrome)
- ▶ Traps into Hyp mode for ID register accesses & idling (WFI/WFE)
- ▶ System instructions to read/write key registers

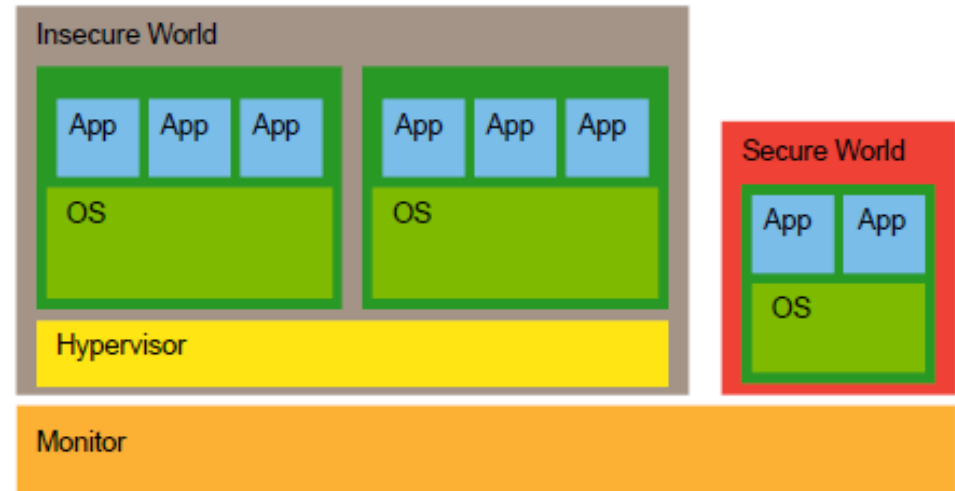
ARMv7 CPU Virtualization



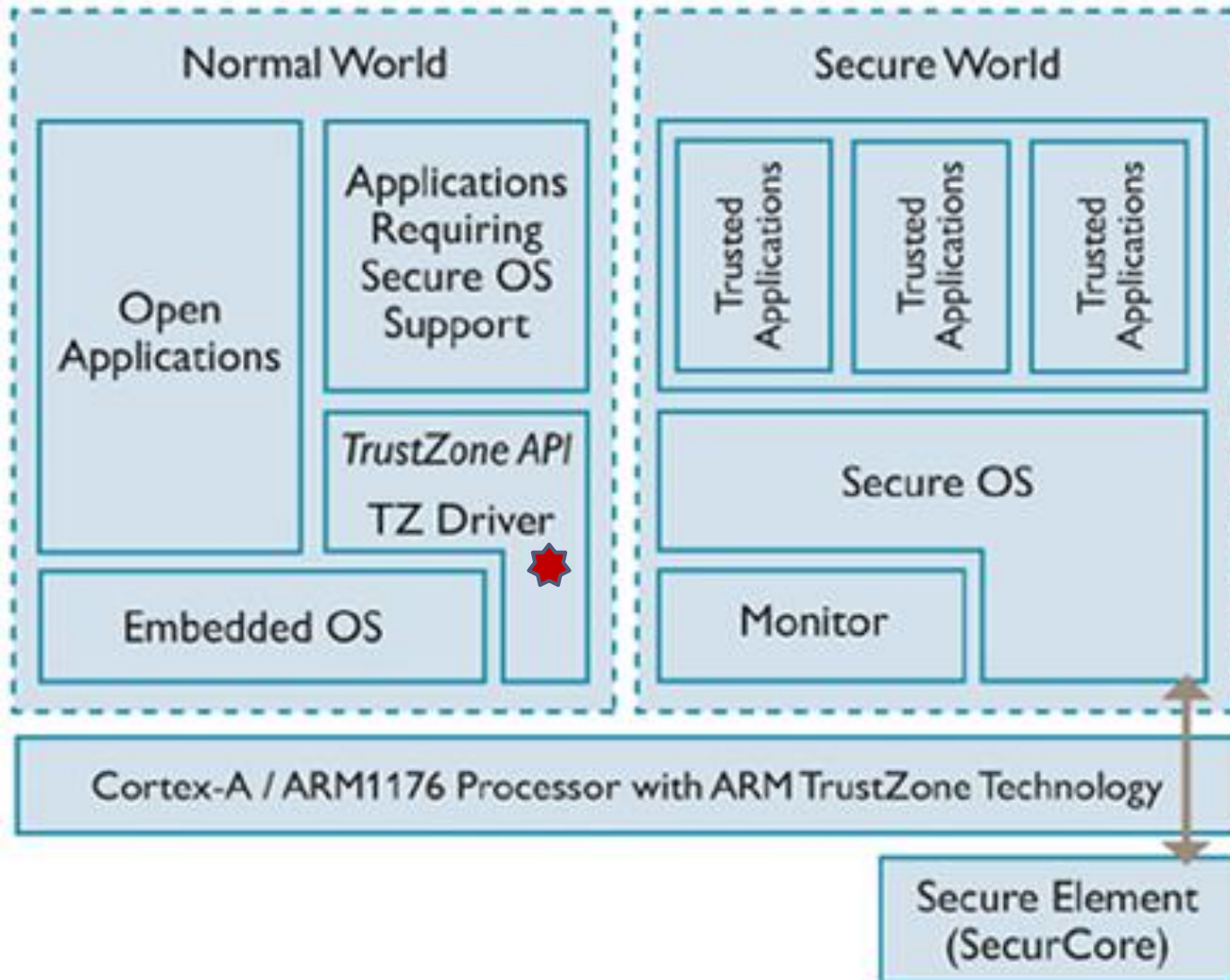
ARM TrustZone (Secure System Partitioning) [1/2]

→ ARM TrustZone extensions introduce:

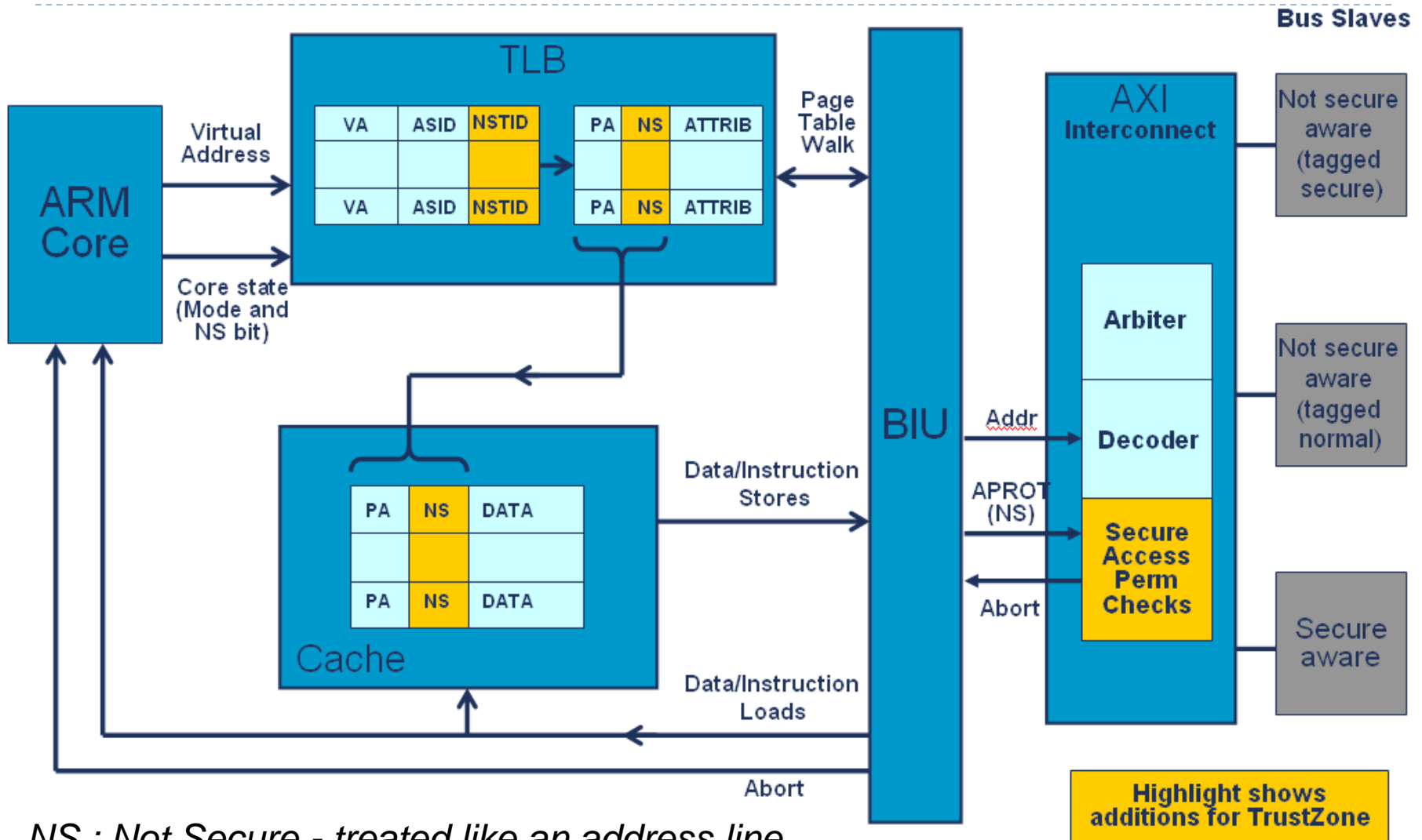
- new processor mode: *monitor*
 - similar to VT-x root mode
 - banked registers (PC, LR)
 - can run unmodified guest OS binary in non-monitor kernel mode
- new privileged instruction: SMI
 - enters monitor mode
- new processor status: *secure*
- partitioning of resources
 - memory and devices marked secure or insecure
 - in secure mode, processor has access to all resources
 - in insecure mode, processor has access to insecure resources only
- monitor switches world (secure ↔ insecure)
- really only supports one virtual machine (guest in insecure mode)
 - need another hypervisor and para-virtualization for multiple guests



ARM TrustZone [2/2]



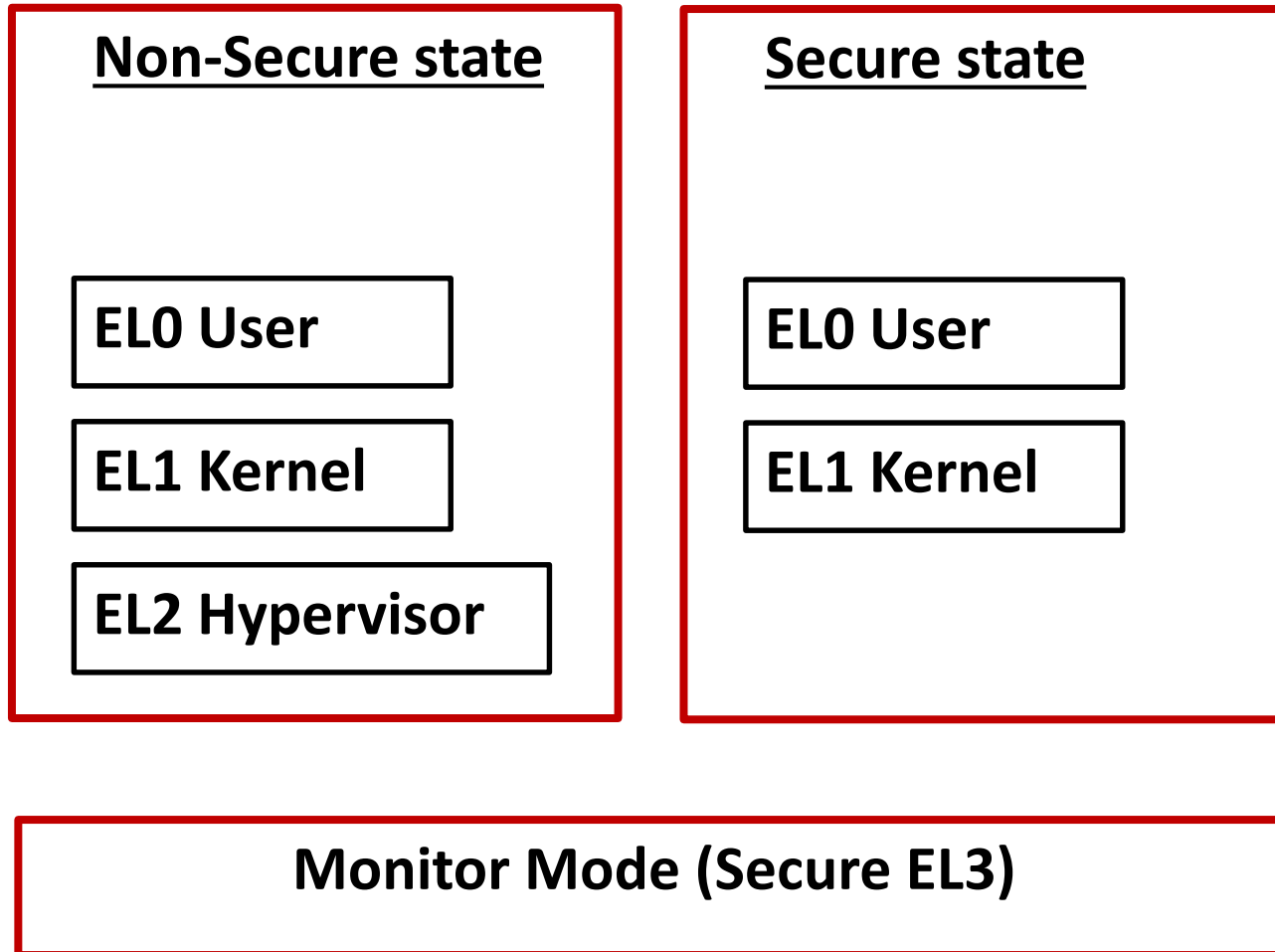
Propagation of System Security Mode



NS : Not Secure - treated like an address line



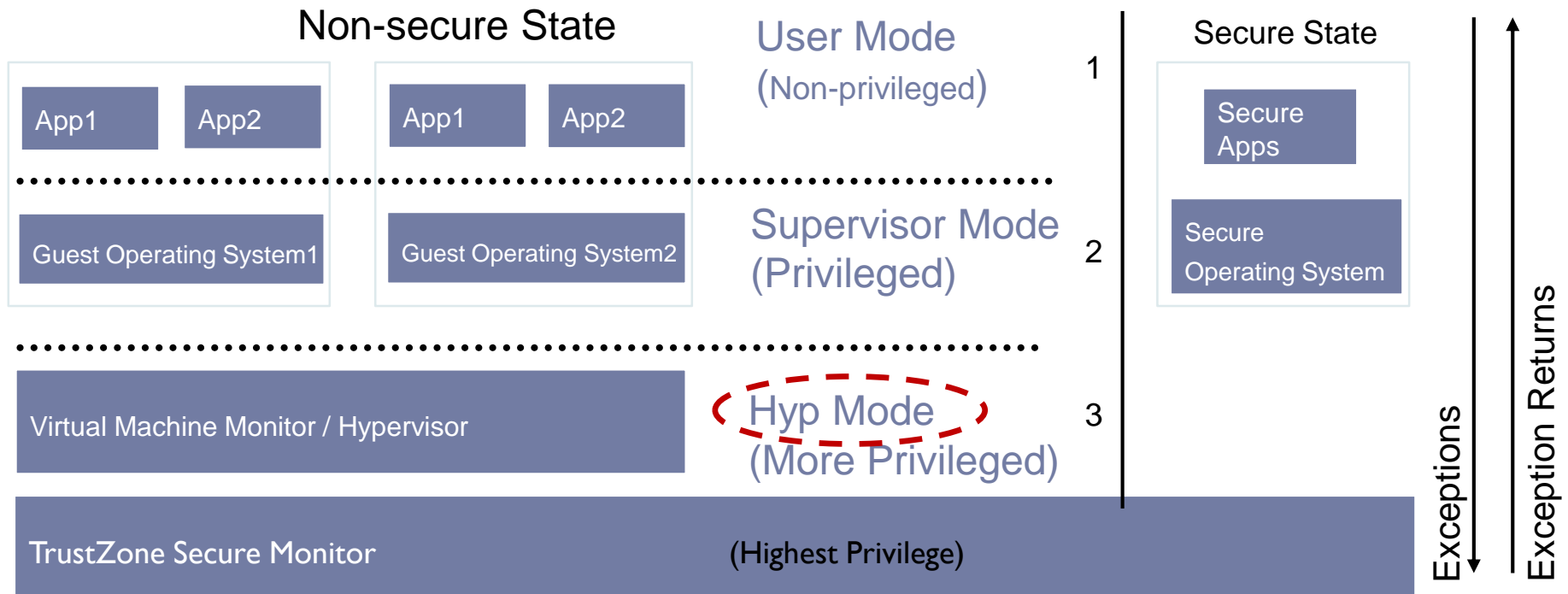
ARMv7 processor modes



Privilege levels

- ▶ Guest OS: same kernel/user privilege structure
- ▶ HYP mode: higher privilege than OS kernel level
 - ▶ hvc instruction (hypercall)
 - ▶ VMM controls wide range of OS accesses
- ▶ Hardware maintains TZ security (4th privilege level)

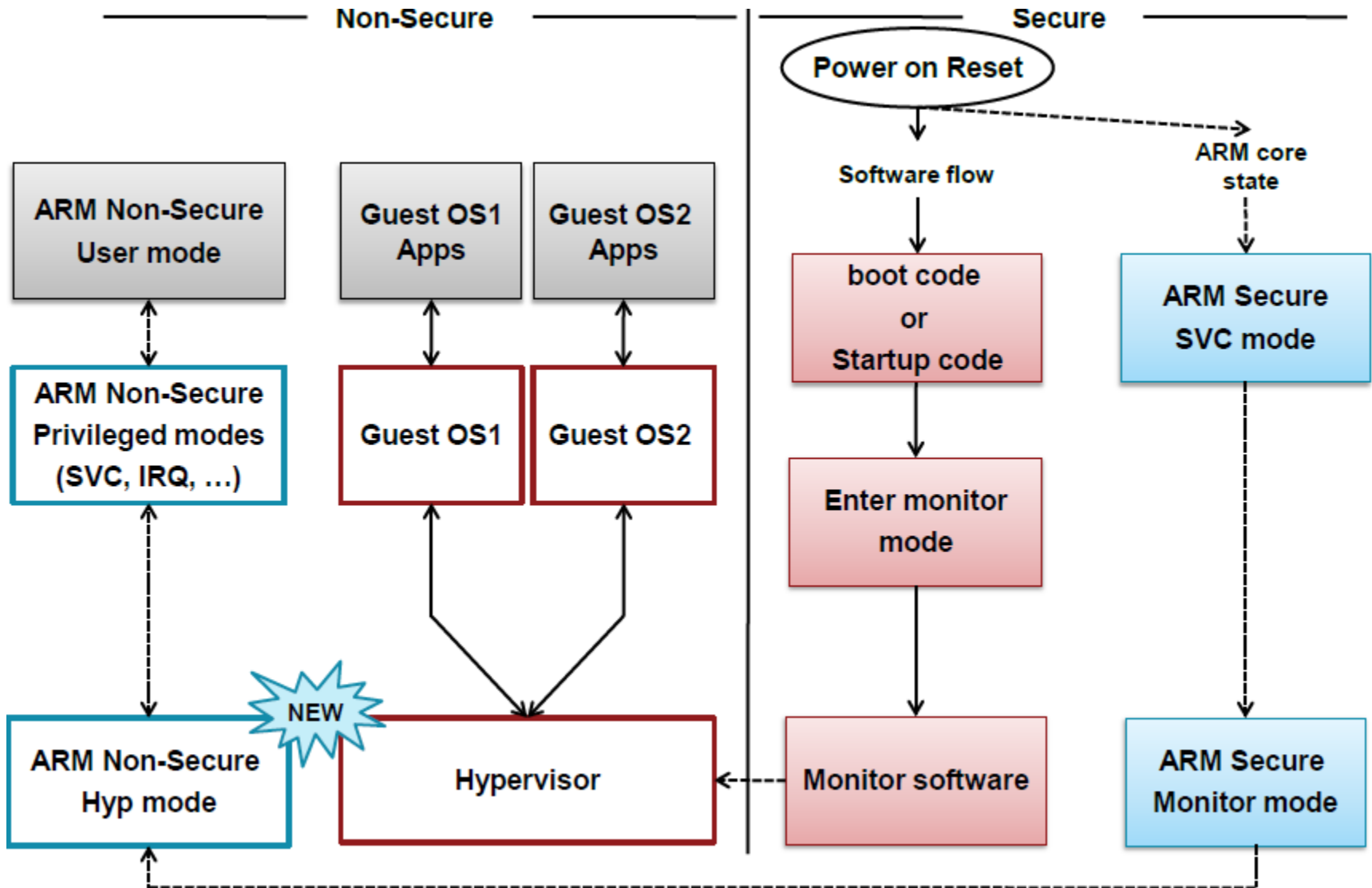
By default, HYP mode is disabled.
- Should be explicitly enabled in secure bootloader code.



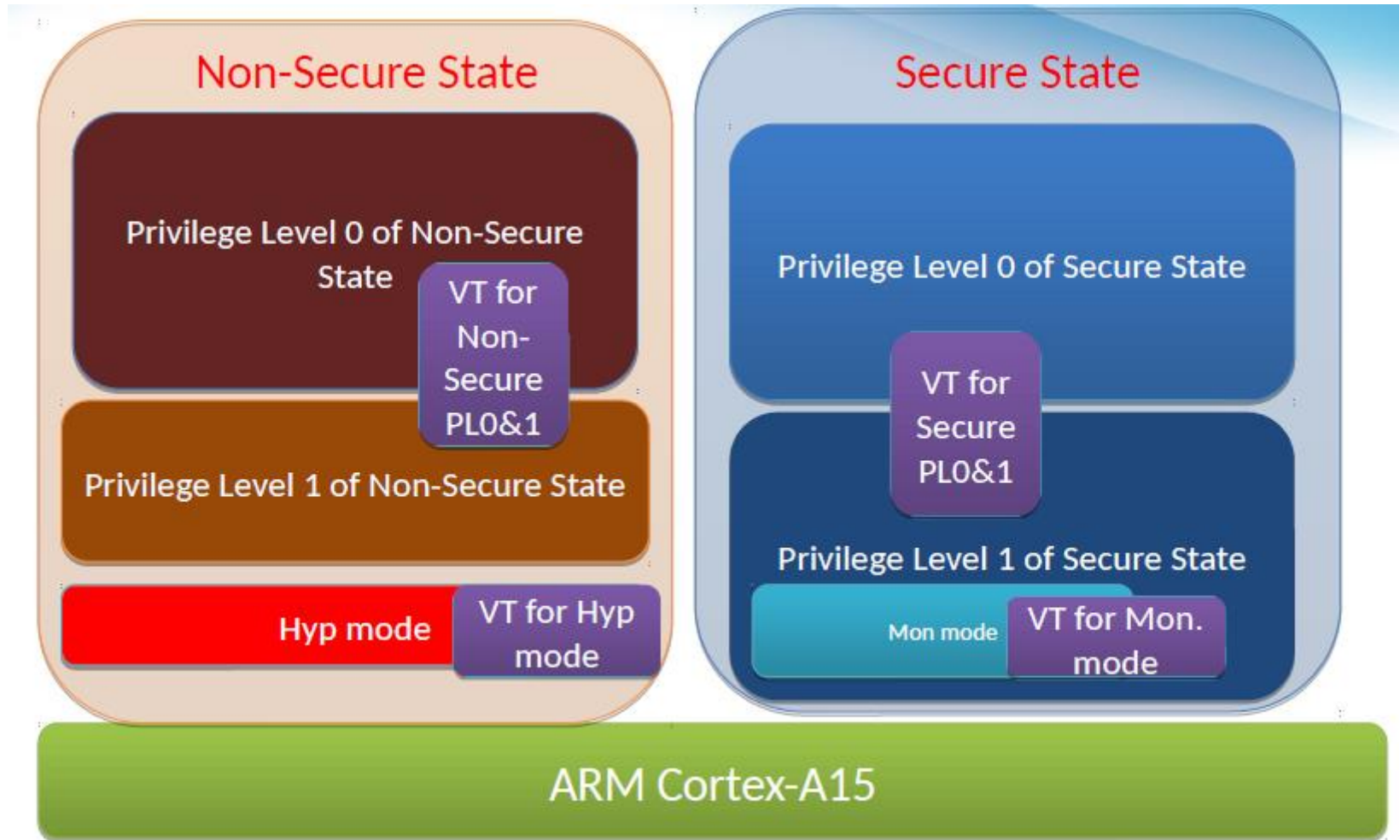
Differences with Intel's VT-x

- ▶ VT-x: Root-Mode is orthogonal to privilege levels (“rings”)
- ▶ ARM HYP: “just one more processor mode”
 - ▶ More privileged than existing “kernel” modes
 - ▶ Hypervisor must save guest VM’s register state
 - ▶ With VT-x, this is done automatically (by hardware)

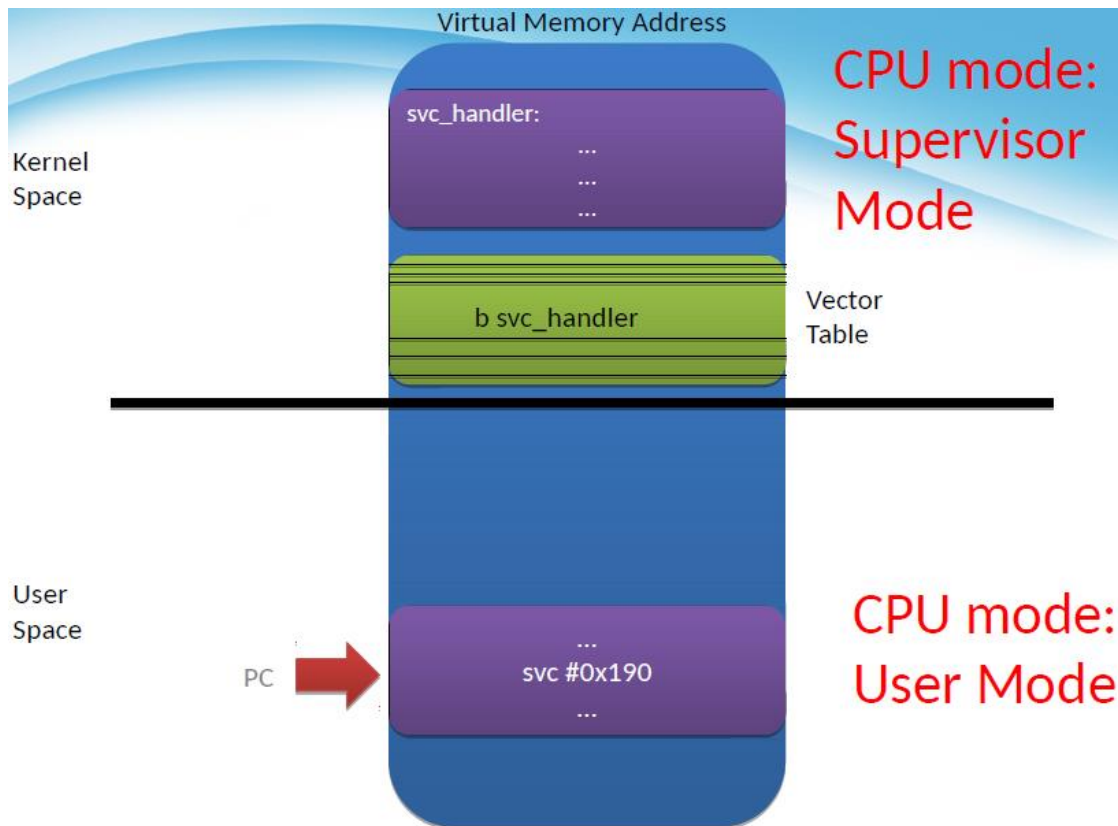
Boot sequence with Hypervisor



Vector Tables

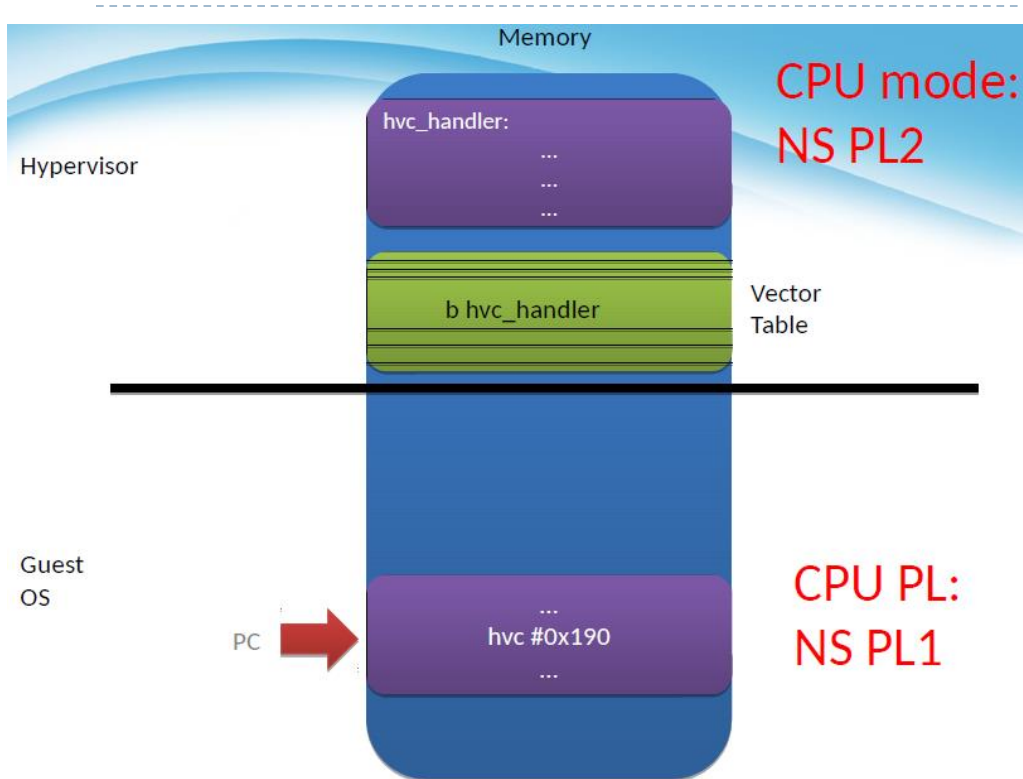


SVC vs. HVC [1/2]



- ▶ **SVC: Supervisor Call**
 - ▶ Software Interrupt (SWI) instruction: allow program to actively trigger an event to enter supervisor mode (from user mode)
 - ▶ Processor jumps to SVC vector stub (in kernel space)

SVC vs. HVC [2/2]



- ▶ **HVC: Hypervisor Call**
 - ▶ instruction that allows program to actively trigger an event to enter hypervisor mode
 - ▶ Non-Secure PL1 → Non-Secure PL2
- ▶ Processor jumps to HVC vector stub (0x14)

Vector Table **with** virtualization extension

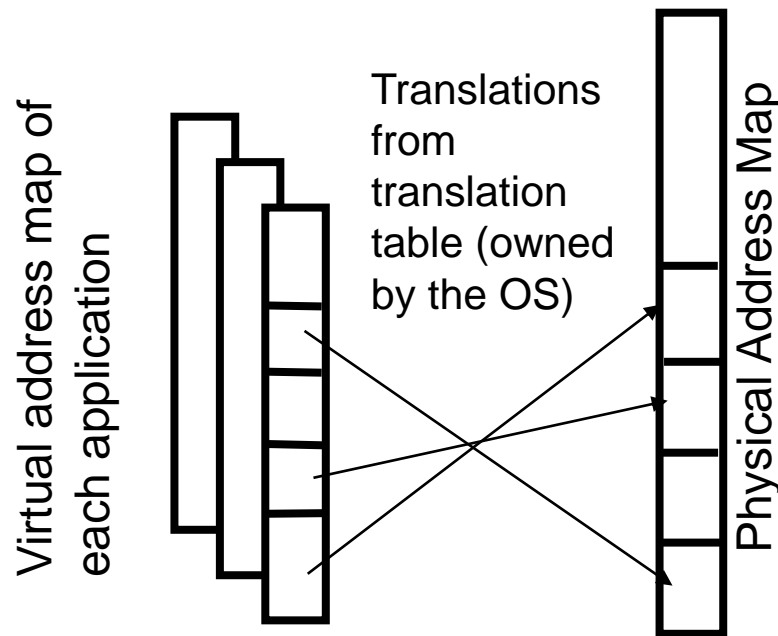
0x1C	FIQ
0x18	IRQ
0x14	Hypervisor Call
0x10	Data Abort
0x0C	Prefetch Abort
0x08	Supervisor Call
0x04	Undefined Instruction
0x00	Reset

Large Physical Address Extension

- ▶ Large Physical Address Extension
 - ▶ 64-bit descriptor entries, up to 3 levels
 - ▶ Input/Output addresses: up to 40 bits
- ▶ VMID: Virtual Machine Identifier
 - ▶ Identifies current VM, with its own Address Space Identifier (ASID – part of TLB maintenance tags)
- ▶ VTTBR: Virtual Translation Table Base Register
 - ▶ 8-bit VMID

Virtual Memory (1-stage translation)

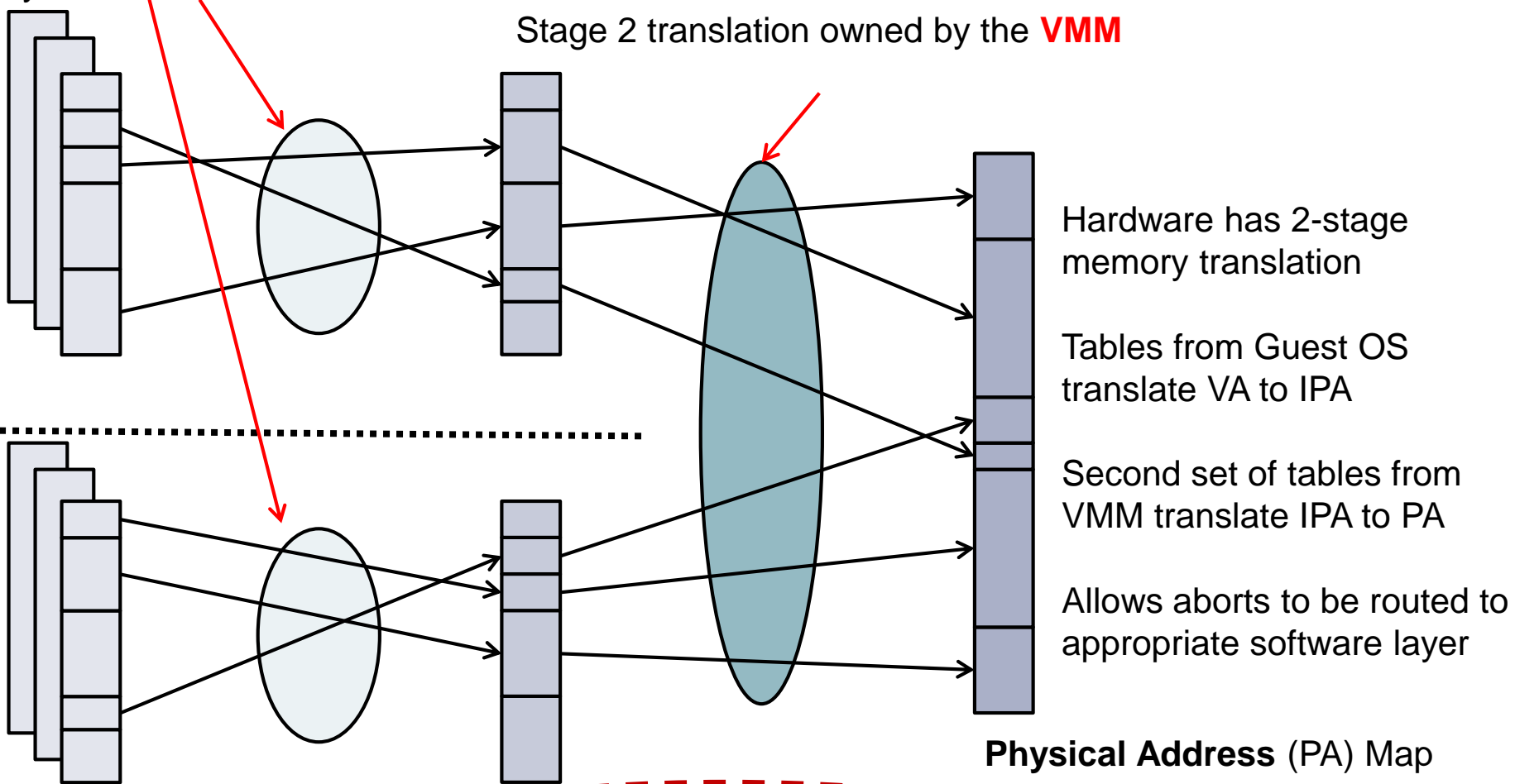
- ▶ Without virtualisation, the OS owns the memory
 - ▶ Allocates areas of memory to the different applications
 - ▶ Virtual Memory commonly used in “rich” operating systems



Virtual Memory (2-stage translation)

Stage 1 translation owned
by each Guest OS

Stage 2 translation owned by the **VMM**

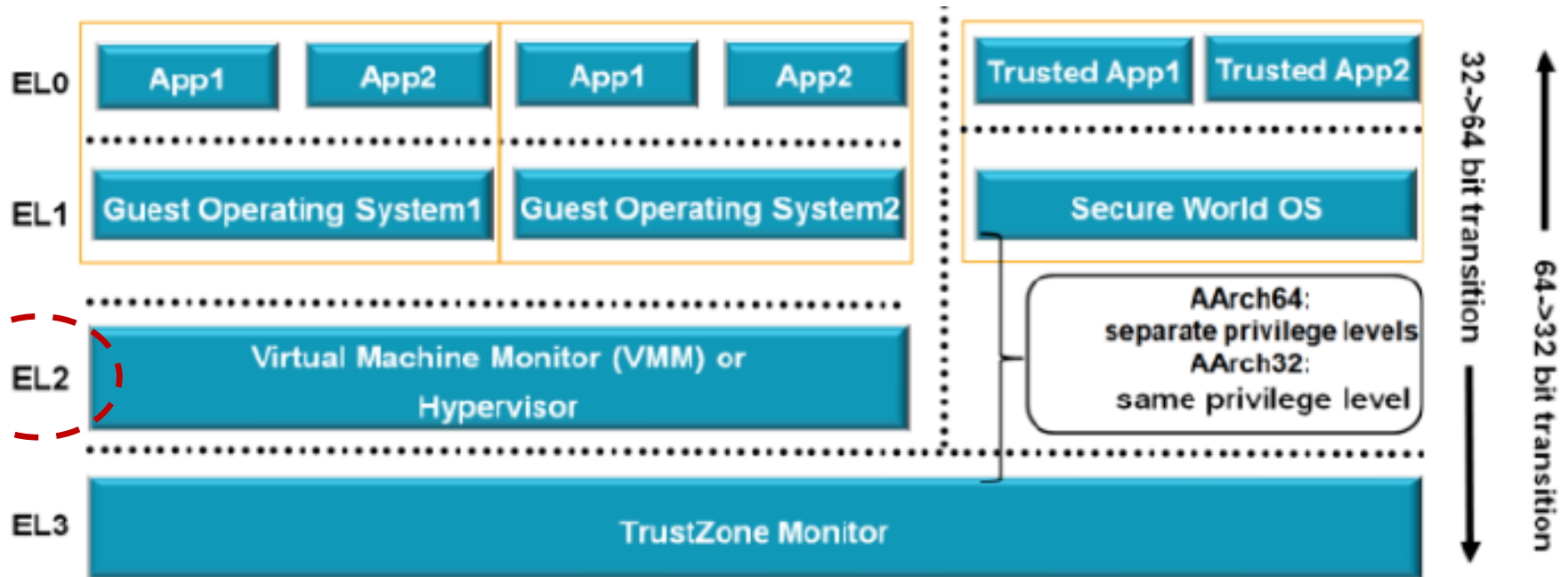


Virtual address (VA) map of
each App on each Guest OS

**“Intermediate Physical” address
map of each Guest OS (IPA)**

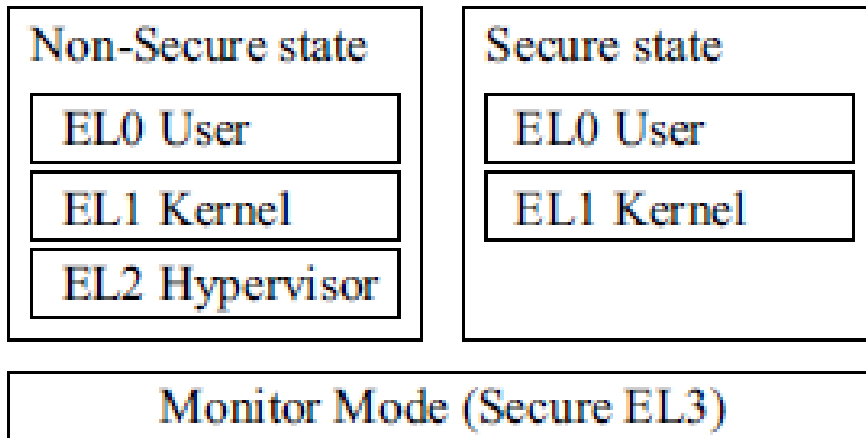


ARMv8 Privilege Model



- Each processor mode has its own linear address space, defined by a distinct page table.
- **EL2** has its own translation regime → tag in TLB entries
- - No need to flush TLB in transitions bet. EL2 and other CPU modes.

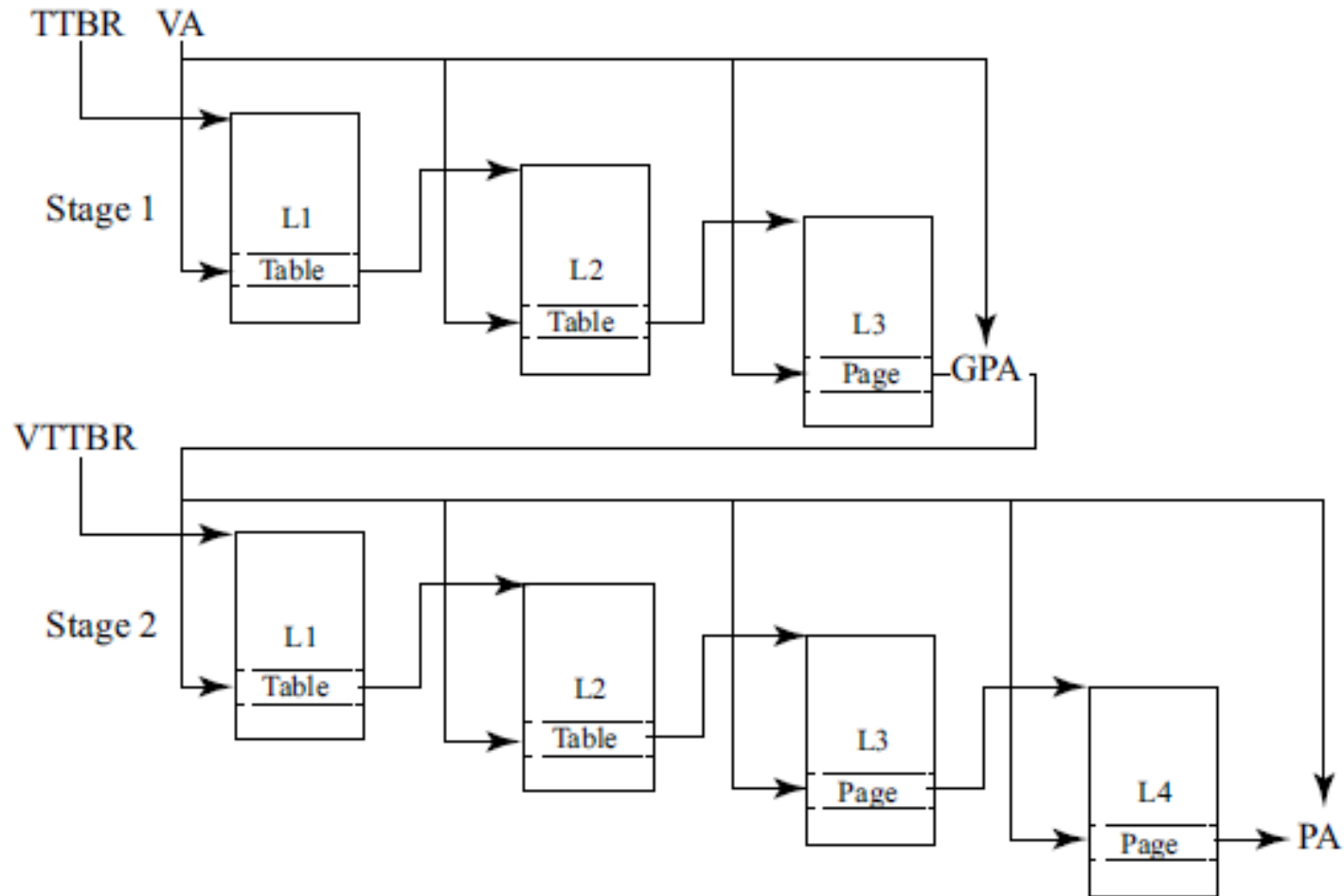
ARM processor modes



EL2: simply a more privileged CPU mode
(i.e. can be used for purposes other than VM support)

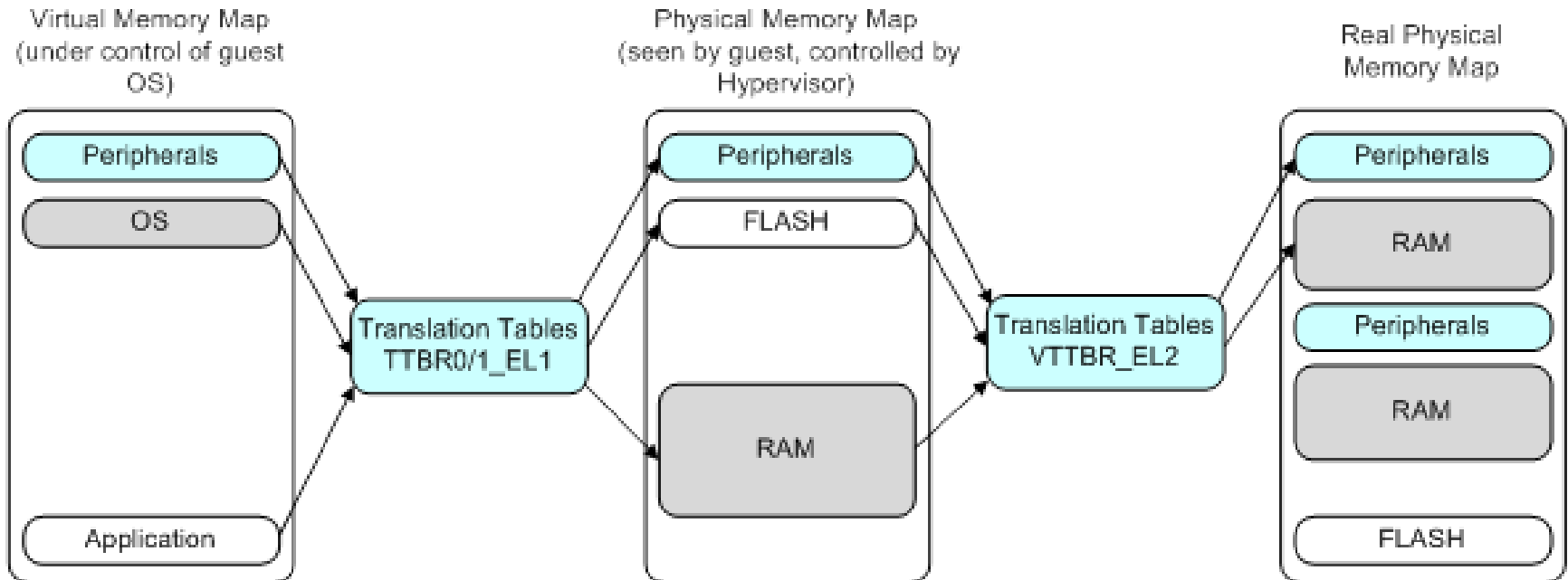
- ▶ The hypervisor enables the processor's virtualization features in EL2 before switching to a VM. The VM will then execute normally, in EL0 and EL1
 - ▶ ... until some condition is reached that requires the intervention of the hypervisor → trap into EL2
- ▶ Each “interrupt” (IRQ, FIRQ) can be configured to trap directly into a VM's EL1
 - ▶ Instead of going through EL2 - e.g. system calls, page faults
- ▶ Any state that needs to be saved & restored must be handled explicitly
 - ▶ Contrast: Intel VT-x → VM control block is automatically saved & restored, with a single instruction, when switching bet. Root – Non-Root modes.

Stage-1 and Stage-2 page table walk



VA (virtual address in VM) → gPA (guest physical address) → hPA (host physical address)

2-stage address translation



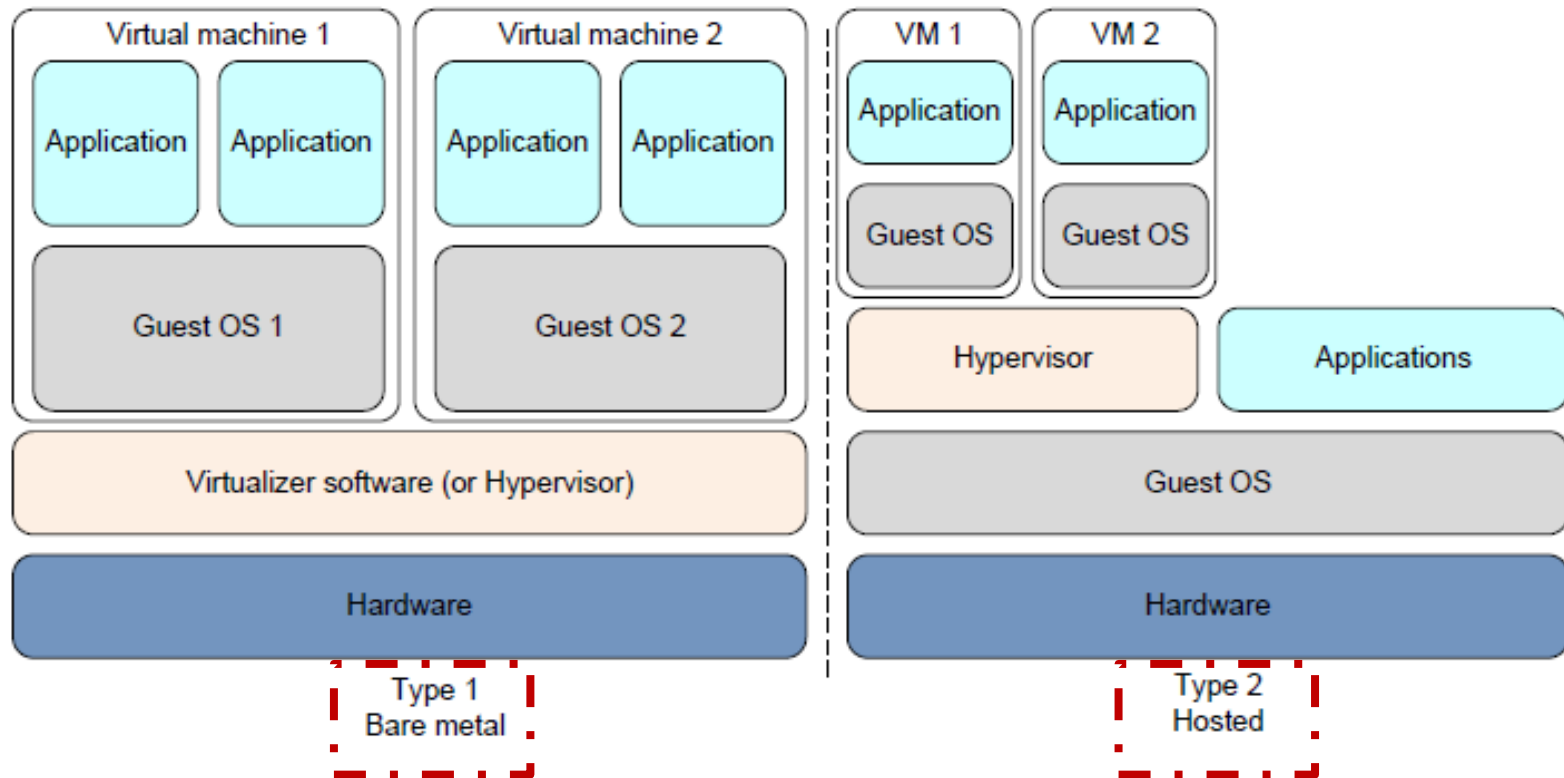
Steps for World Switch

1. Store all host GP registers onto EL2 stack
2. Configure vGIC and virtual timers for VM
3. Save host-specific CSRs onto EL2 stack
4. Load VM's CSR's (no impact on current execution, as EL2 uses its own CSRs – separate from host state)
5. Configure EL2 to trap FP operations for “lazy” context switching of VFP registers, trap interrupts, trap WFI/WFE (CPU halt) instructions, trap SMC instructions & specific CSR & debug-register accesses
6. Write VM-specific IDs into shadow ID registers
7. Set Stage-2 page table base register (VTTBR), enable Stage-2 address translation
8. Restore all Guest GP registers, and trap into either user or kernel mode for the VM.

VM and Host State (Cortex-A15)

Action	Nr.	State
Context Switch	38	General Purpose (GP) Registers
	26	Control Registers
	16	VGIC Control Registers
	4	VGIC List Registers
	2	Arch. Timer Control Registers
	32	64-bit VFP Registers
	4	32-bit VFP Control Registers
Trap-and-Emulate	-	CP14 Trace Registers
	-	WFI Instructions
	-	SMC Instructions
	-	ACTLR Access
	-	Cache Ops. by Set/Way
	-	L2CTLR / L2ECTLR Registers

Types of Hypervisors [1/2]



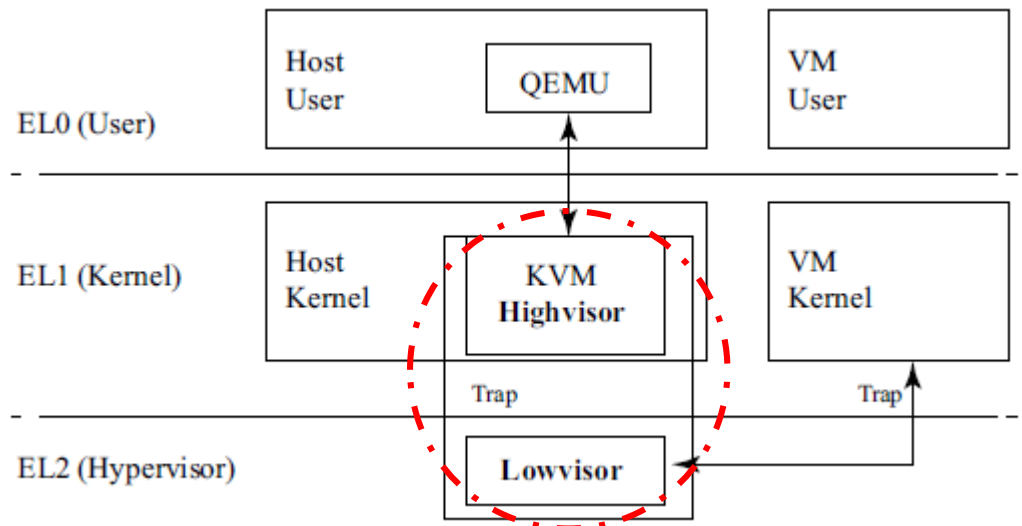
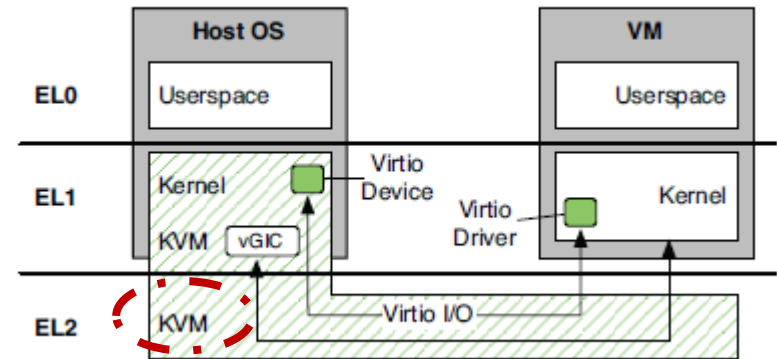
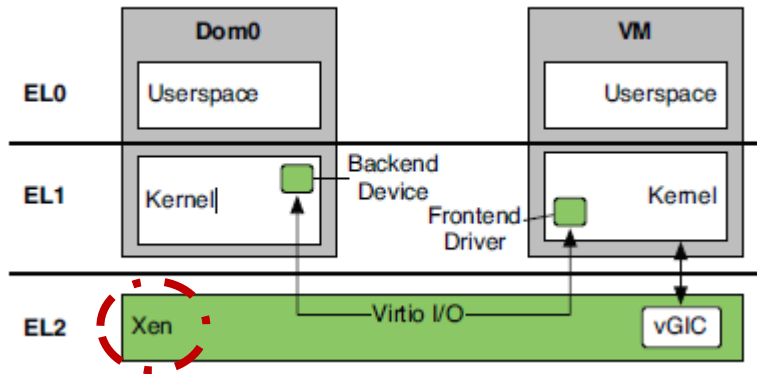
Hypervisor support in ARMv8 (aarch64):

- Dedicated exception level (EL2) for hypervisor
- Trapping exceptions that change core context/state
- Routing of exceptions and virtual interrupts
- 2-stage memory translation
- Dedicated exception (HVC) for Hypervisor Call

Types of Hypervisors [2/2]

- ▶ **Full virtualization**
 - ▶ Unmodified Guest OS
 - ▶ Guest I/O
 - ▶ emulated
 - ▶ ... or handled by virtualization-aware HW
- ▶ **Para-virtualization**
 - ▶ Guest OS is aware of the hypervisor
 - ▶ Privileged instructions are replaced by VMM hooks
- ▶ **Hybrid**
 - ▶ Use as much as possible hardware-assisted virtualization
 - ▶ Devices (network, block) para-virtualized or passthrough'ed
 - ▶ Use of emulation is very limited

Common hypervisors on ARM architecture



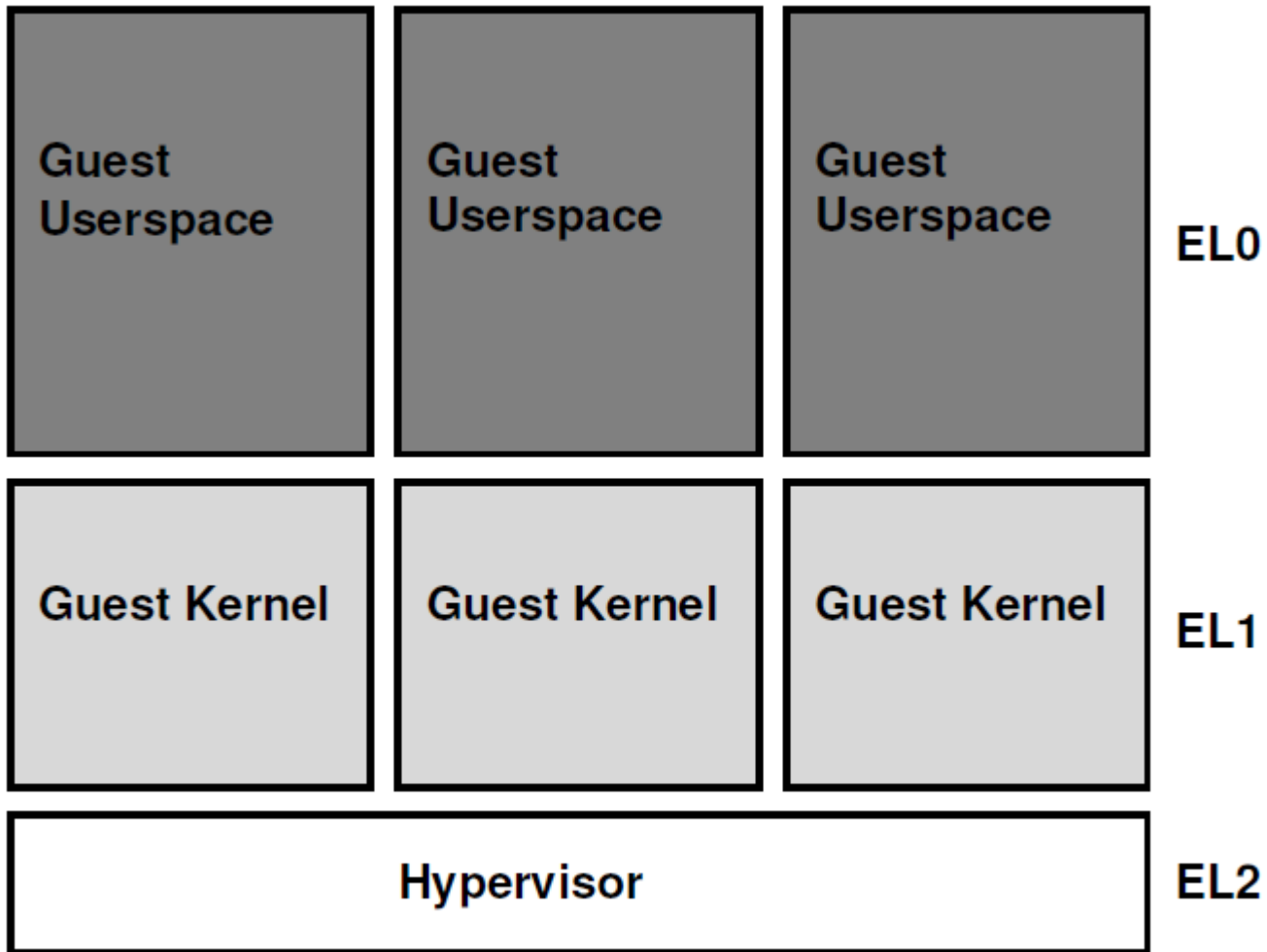
ARMv8 Virtualization Features

- ▶ **2nd stage of memory translation**
 - ▶ Adds extra level of indirection between guest & physical memory
 - ▶ TLBs are tagged by Virtual Machine ID (VMID: 8-/16-bit)
- ▶ **Ability to trap access of most system registers**
 - ▶ The hypervisor decides what it wants to trap
- ▶ **Can handle IRQs, FIQs and asynchronous aborts**
 - ▶ The guest doesn't see physical interrupts firing
- ▶ **Guests can call into EL2 mode (HVC instruction)**
 - ▶ Allows para-virtualized services
- ▶ **Standard architecture peripherals are virtualization-aware**
 - ▶ GIC and timer have specific features to help virtualization

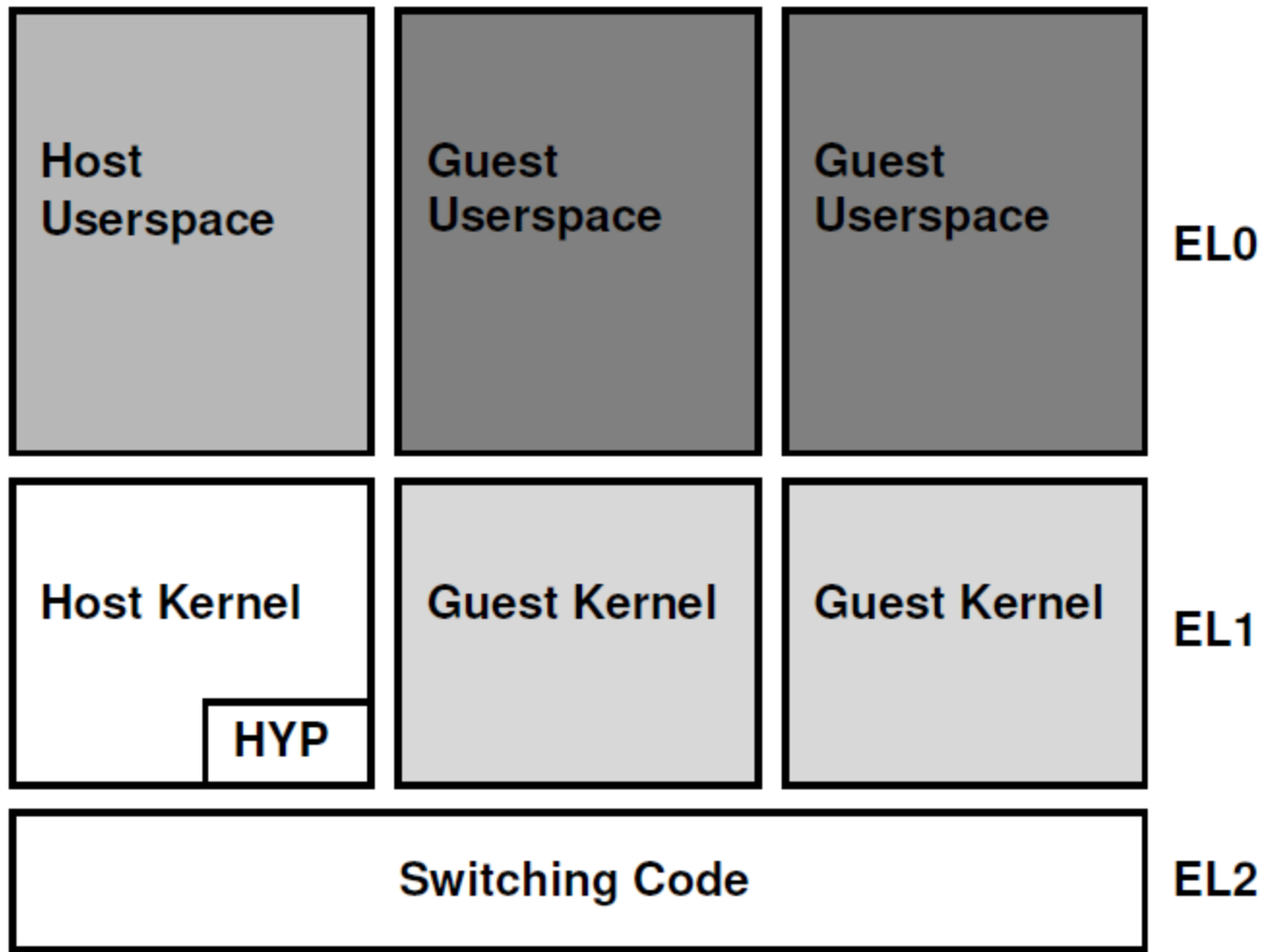
EL2 in ARMv8

- ▶ EL2 is not a superset of NS-EL1
 - ▶ Orthogonal mode to EL1
 - ▶ Allows multiplexing of NS-EL1 guests on the hardware
- ▶ Own translation regime
 - ▶ Separate Stage-1 translation, no Stage-2 translation
- ▶ It would be difficult to run Linux in EL2
 - ▶ too many changes to be practical
- ▶ EL2 could be used for "world switch"
 - ▶ Between guests (bare-metal hypervisor/Type I)
 - ▶ Between host and guest (hosted hypervisor/Type II)
 - ▶ *This makes the host a form of specialized guest ...*

Type-I Hypervisor



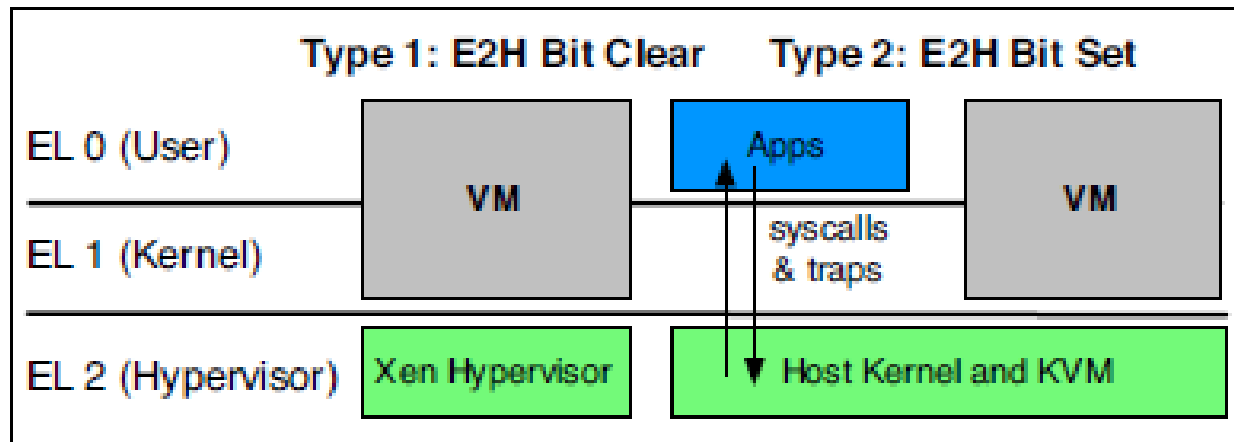
Type-II Hypervisor



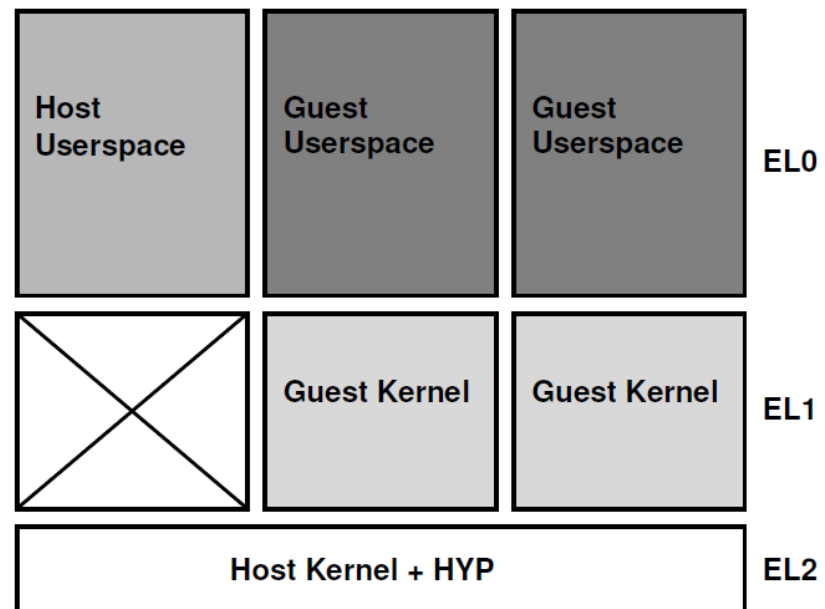
Virtualization Host Extensions (VHE)

- ▶ **Part of ARMv8.1 (AArch64 specific): expands the capability of EL2**
 - ▶ Designed to improve the support of the Type-II hypervisors
 - ▶ Allows the host OS to be run at EL2
 - ▶ Significantly reduces the number of system registers shared between Host and Guest
 - ▶ The host OS requires minimal changes to run at EL2
 - ▶ Use HYP timer interrupt, instead of Guest's timer interrupt
 - ▶ User-space still runs at EL0
 - ▶ Host has no software running at EL1
 - ▶ VHE provides a mechanism to access the extra EL2 register state transparently.
 - ▶ Simply providing extra EL2 registers is not sufficient to run unmodified Oses in EL2, because existing Oses are written to access EL1 registers.

Type-II Hypervisor with VHE



Type-II Hypervisor with VHE

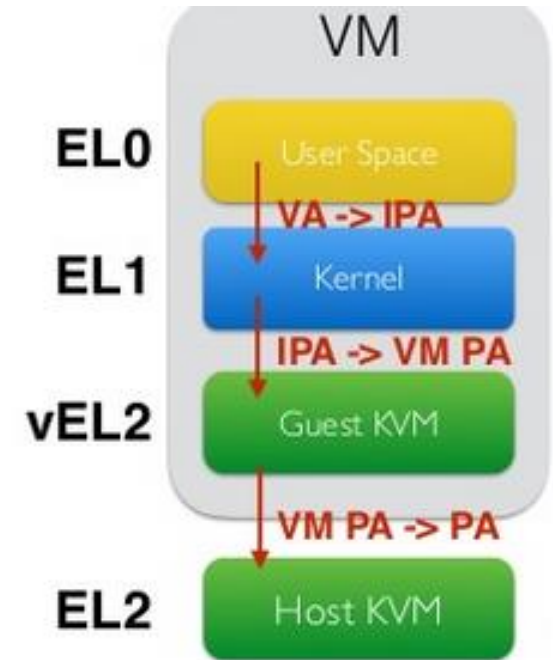


VHE: Impact on Hypervisor (kvm)

- ▶ Reduced save/restore of host system registers
 - ▶ Only 4 registers
- ▶ “Lazy” save/restore of Guest system registers
 - ▶ Deferred until actually needed by Host, or upon VM switch
- ▶ Reduced interrupt latency
 - ▶ Less state to be restored before handling the interrupt
- ▶ No trap to enter hypervisor (normal function call!)
- ▶ No HYP mappings needed
 - ▶ The kernel runs at EL2

Nested Virtualization

- ▶ Part of ARMv8.3
- ▶ Allows an hypervisor in a VM
 - ▶ Unmodified guest hypervisor running in NS EL1
 - ▶ VM thinks it runs at “virtual” EL2 (using VHE)
 - ▶ VM uses EL1 register accesses to access EL2 registers (HCR_EL2.NV, Shadow EL1 registers)
 - ▶ Very few traps needed (EL1 → EL2, “eret”)
- ▶ Implementation of a host hypervisor required
 - ▶ Running at EL2



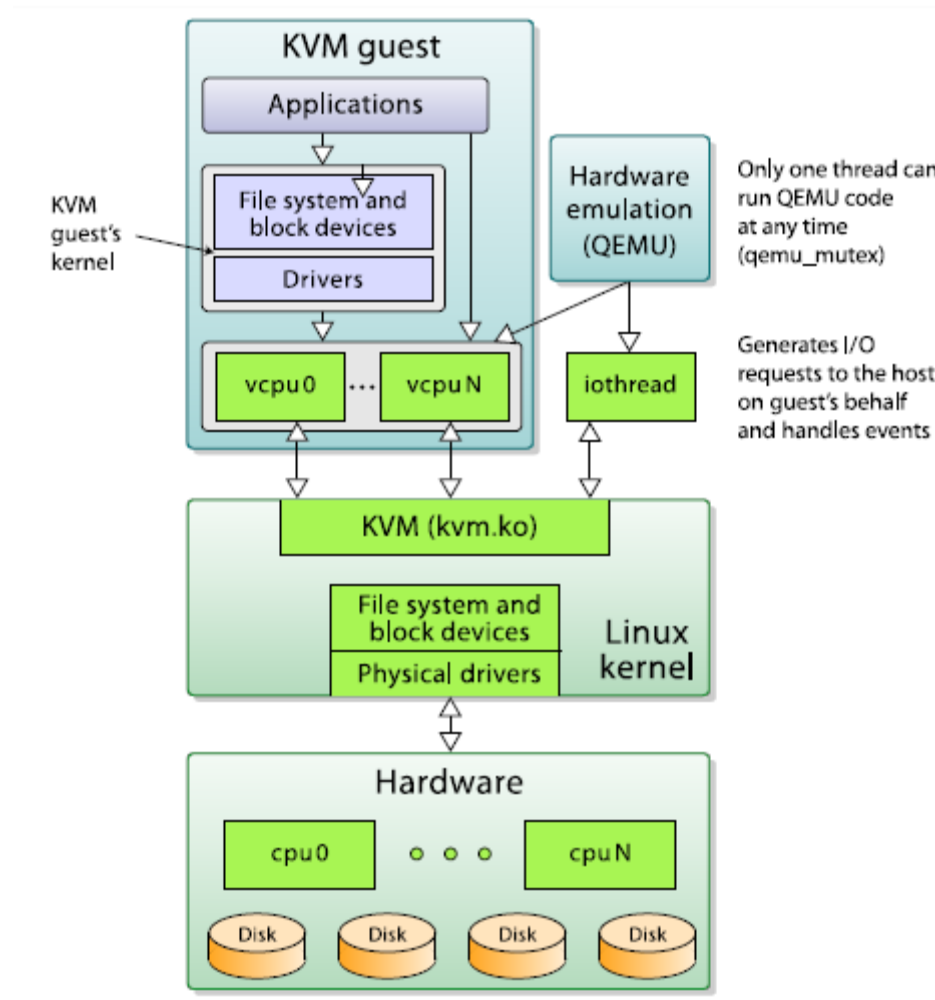
Use cases :

- ▶ Develop/test/debug hypervisor in VM
- ▶ laaS hosting private cloud

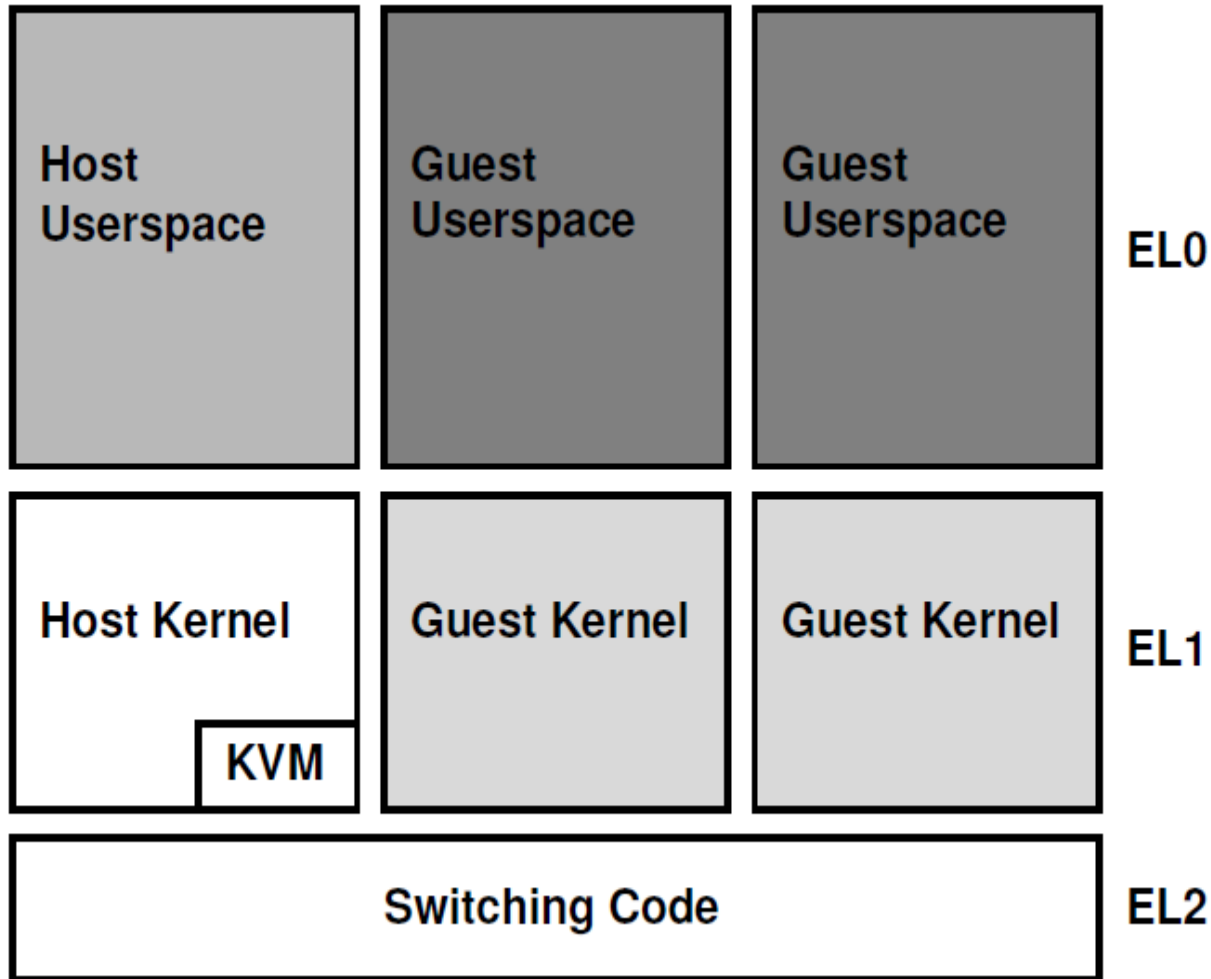
KVM: Kernel-based Virtual Machine

- ▶ Hosted (Type-II) hypervisor
 - ▶ Unlike Xen : Type-I
- ▶ Use of assisted hardware virtualization
- ▶ Devices are
 - ▶ emulated (QEMU)
 - ▶ para-virtualized (VIRTIO)
- ▶ All CPUs are using the same scheduler
 - ▶ guest vCPU is a task for the host OS
- ▶ Resource management can be done using cgroups
 - ▶ Standard way in Linux to control resources

KVM Architecture



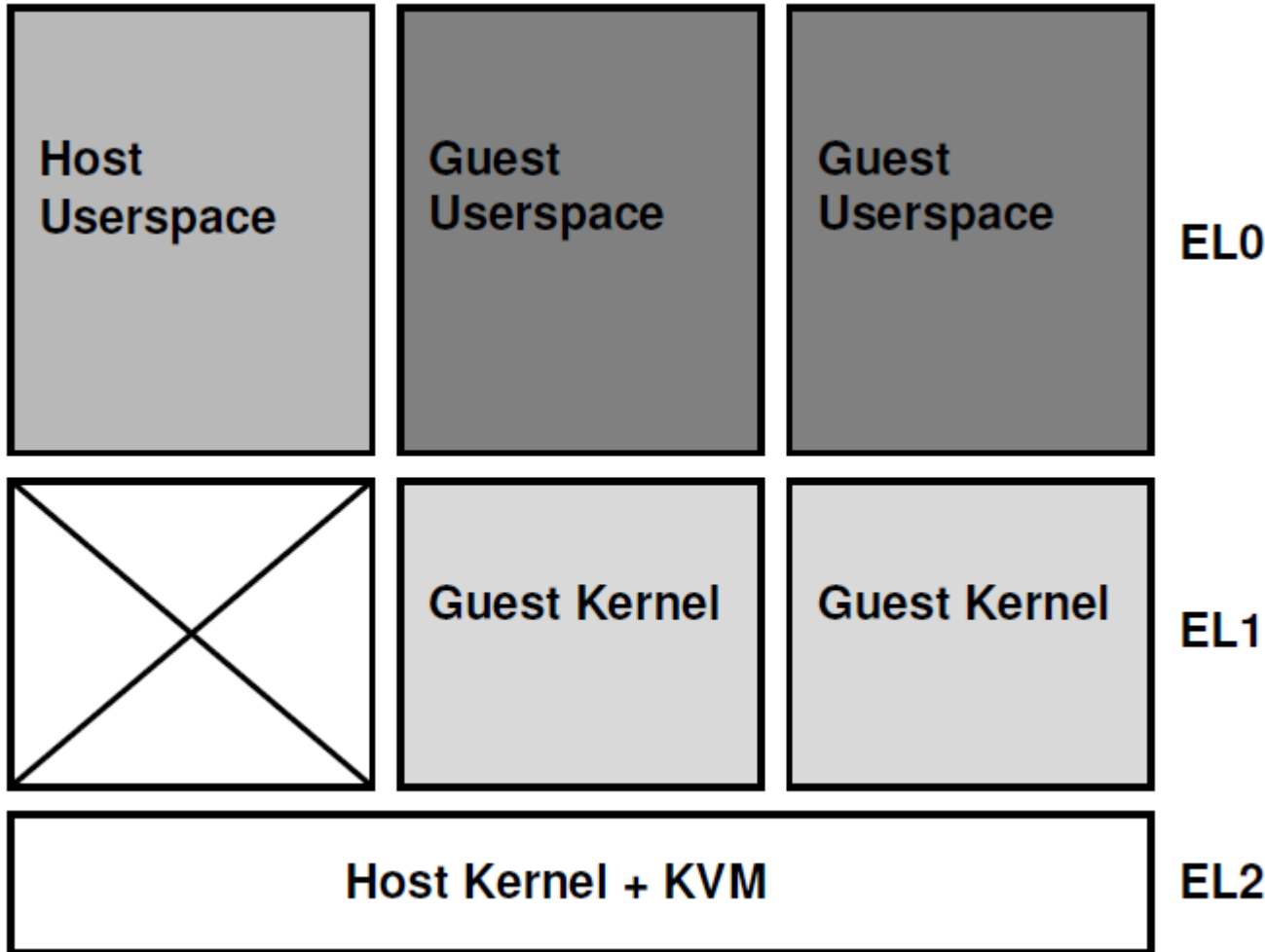
KVM architecture with ARMv8.0-A



State to save/restore:

- Stage-2 translation table
- Trap configuration
- General-purpose registers
- System control registers
- FP registers
- GIC configuration
- Timer configuration

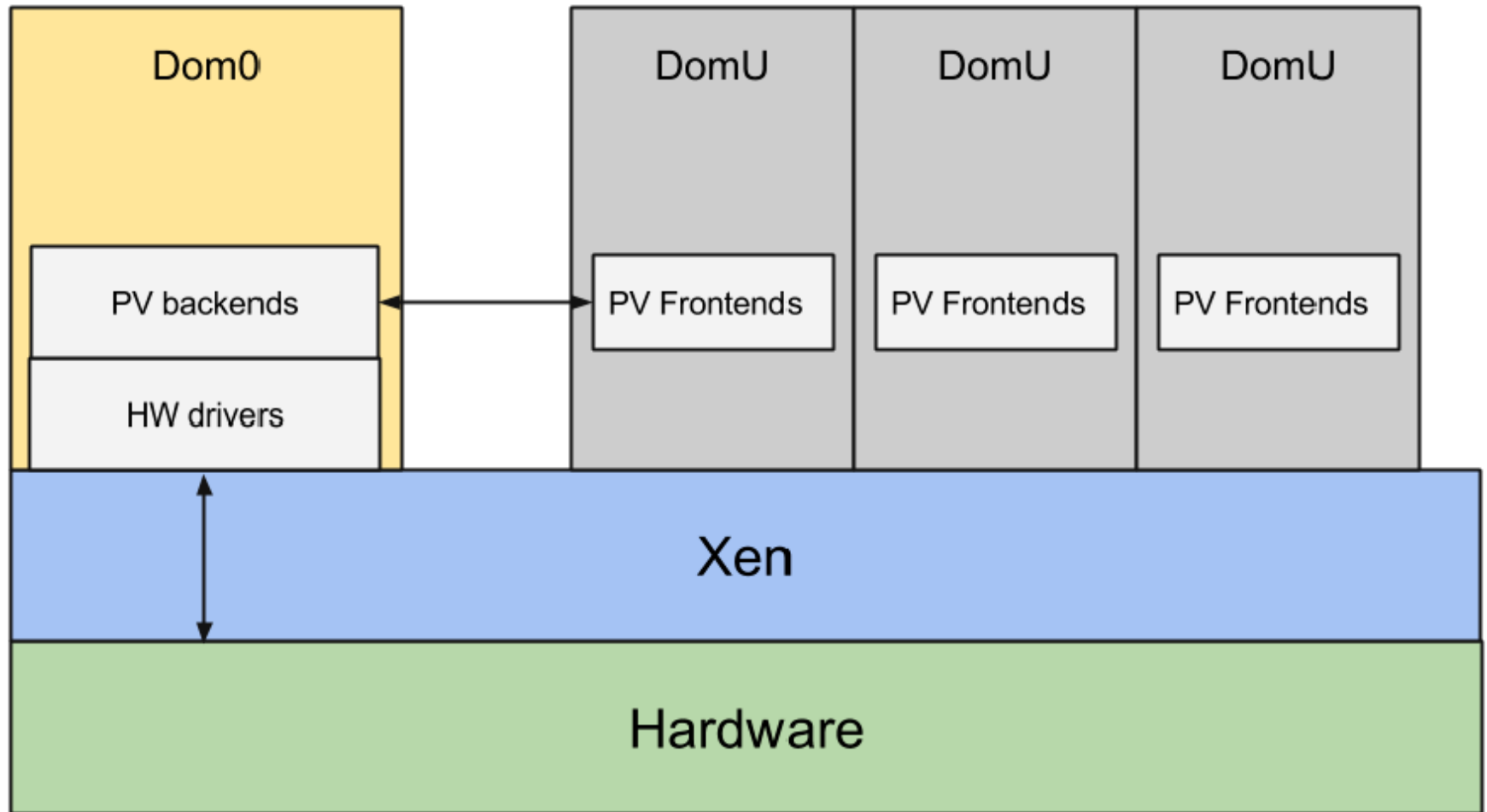
KVM architecture with ARMv8.1-A



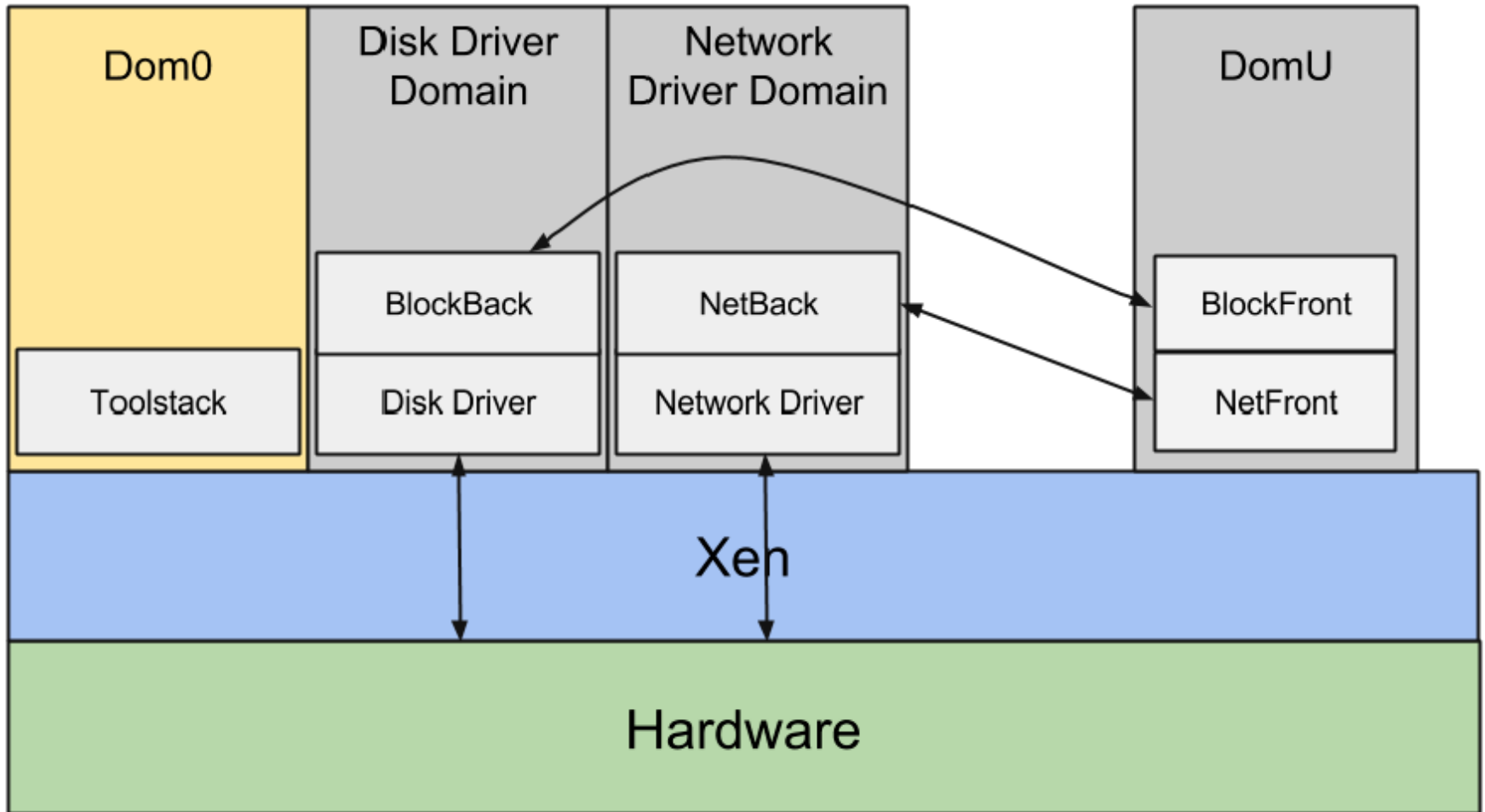
ARMv8.1 features:

- 16-bit VMIDs
- VHE (expanded EL2)

Xen: Para-virtualization

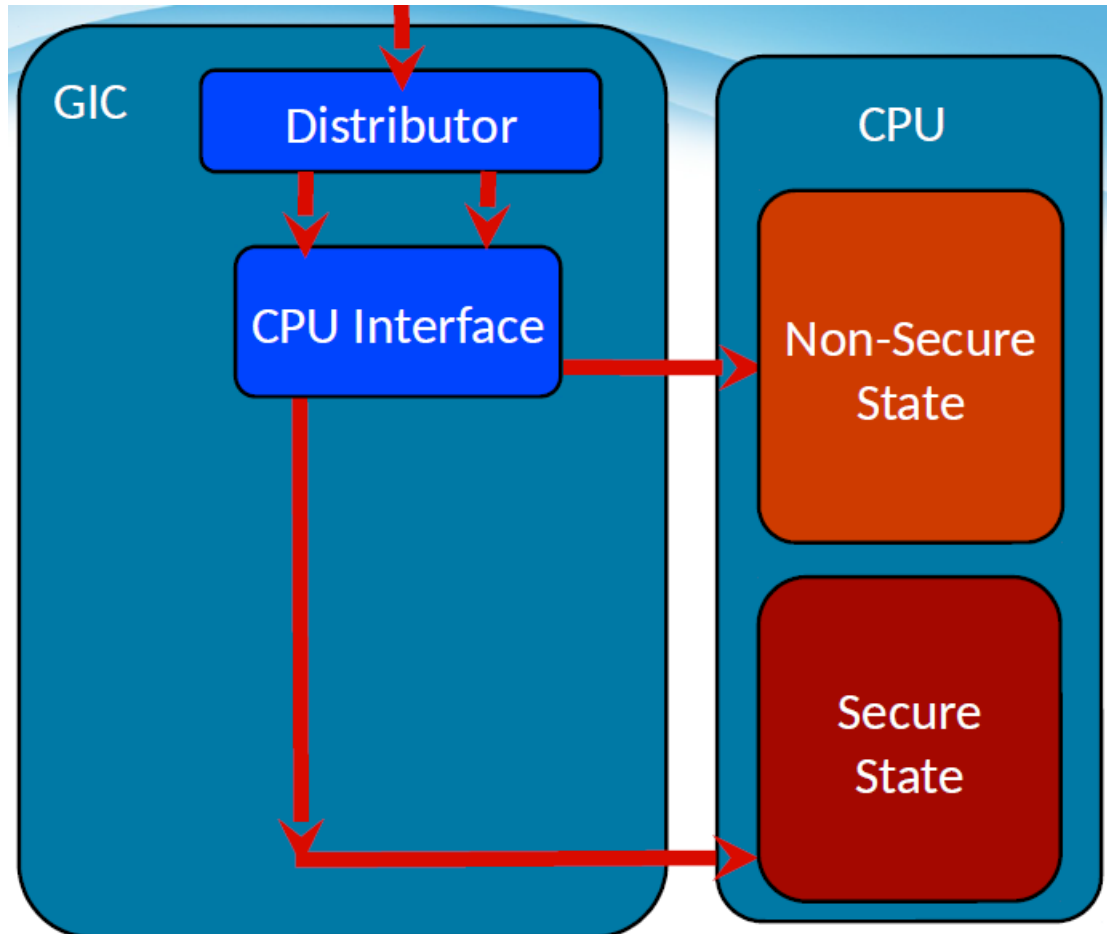


Xen: Driver Domains



GIC: Generic Interrupt Controller

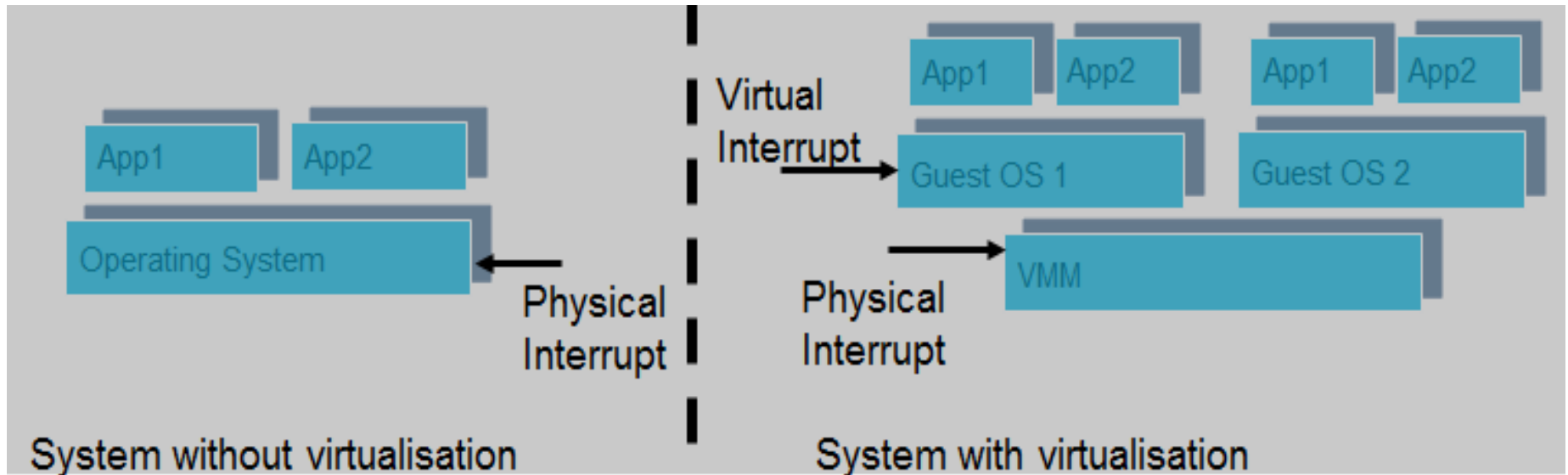
- ▶ Single interrupt controller in ARM architecture
- ▶ + Firmware: “Interrupt Distributor”



vGIC: virtual CPU interface
+ distributor

Virtualization of interrupts

- ▶ An interrupt might need to be routed to one of:
 - ▶ Current or different GuestOS
 - ▶ Hypervisor
 - ▶ OS/RTOS running in the secure TrustZone environment
- ▶ Physical interrupts are taken initially in the Hypervisor
 - ▶ If the Interrupt should go to a GuestOS :
 - ▶ Hypervisor maps a “virtual” interrupt for that GuestOS

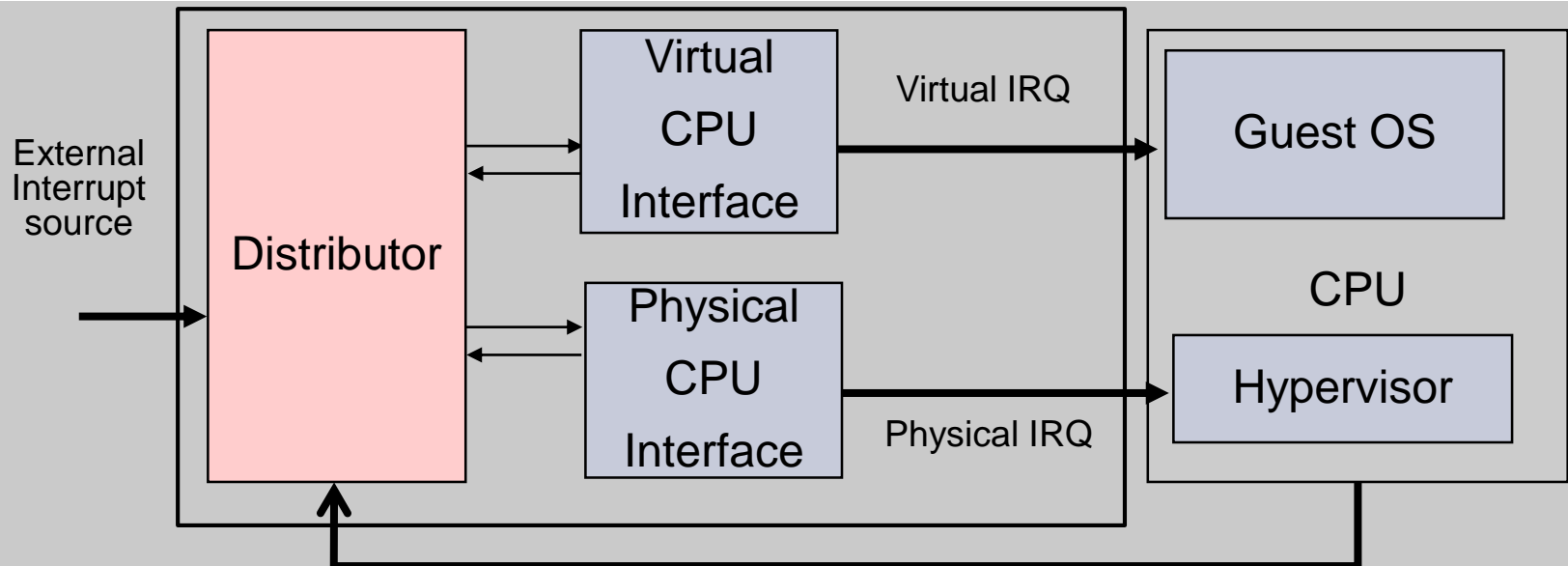


Virtual Interrupt Controller

- ▶ **ISR of GuestOS interacts with the virtual controller**
 - ▶ Pending and Active interrupt lists for each GuestOS
 - ▶ Interact with the physical GIC in hardware
 - ▶ Creates Virtual Interrupts only when priority indicates it is necessary
- ▶ **GuestOS ISRs therefore do not need calls for:**
 - ▶ Determining interrupt to take [Read of Interrupt Acknowledge]
 - ▶ Marking the end of an interrupt [Sending EOI]
 - ▶ Changing CPU Interrupt Priority Mask [Current Priority]
- ▶ **GIC has separate sets of internal registers:**
 - ▶ Physical registers and virtual registers
 - ▶ Non-virtualized system and hypervisor access the physical registers
 - ▶ Virtual machines access the virtual registers
 - ▶ Guest OS functionality does not change when accessing the vGIC
- ▶ **Hypervisor remaps virtual registers for use by GuestOS'es**
 - ▶ Interrupts generate a hypervisor trap

Virtual interrupt sequence

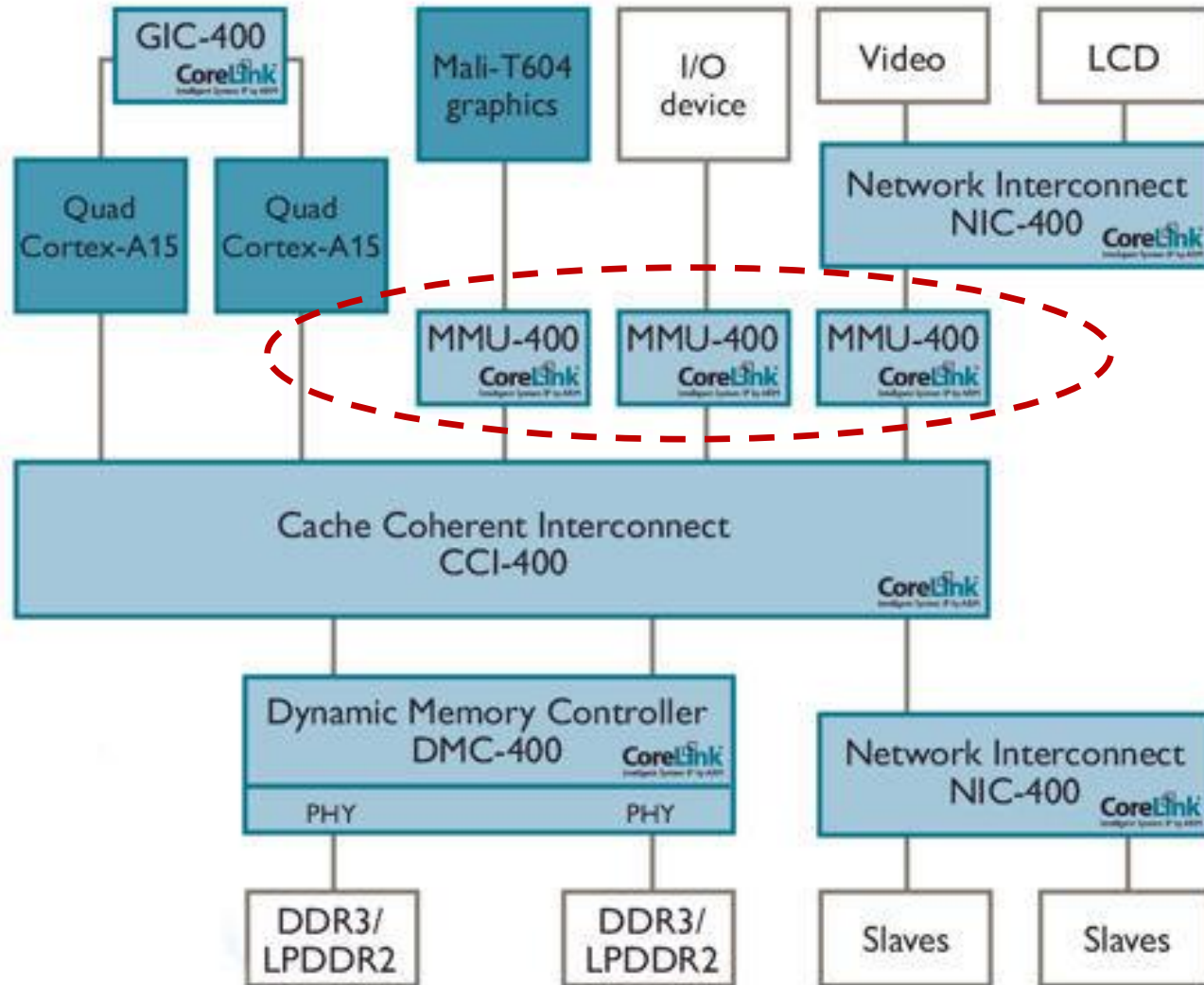
- ▶ External IRQ (configured as virtual by the hypervisor) arrives at the GIC
- ▶ GIC Distributor signals a Physical IRQ to the CPU
- ▶ CPU takes HYP trap, and Hypervisor reads the interrupt status from the Physical CPU Interface
- ▶ Hypervisor makes an entry in register list in the GIC
- ▶ GIC Distributor signals a Virtual IRQ to the CPU
- ▶ CPU takes an IRQ exception, and Guest OS running on the virtual machine reads the interrupt status from the Virtual CPU Interface



Virtual I/O devices

- ▶ Memory-mapped devices
 - ▶ Read/write accesses to device registers have specific side-effects
- ▶ Virtual devices → emulation
 - ▶ Typically, read/write accesses have to trap to Hypervisor
 - ▶ Fetch & interpretation of emulated load/stores is performance-intensive
 - ▶ Syndrome: key information about an instruction
 - ▶ Source/destination register, Size of data transfer, ...
 - ▶ Available for some loads/stores (on abort)
 - If not available, then it is required to fetch the instruction for full emulation
- ▶ System MMU: 2nd-stage address translation for devices
 - ▶ Allows devices to be programmed into Guest's VA space

System MMU (IO-MMU)



Device emulation

- ▶ Platform devices are memory-mapped, and guest accesses to devices are subject to at least Stage 2 translation when virtualization is in effect.
- ▶ Guests can detect a platform device by reading its ID register, or by interrogating registers mentioned in the device tree.
 - ▶ The hypervisor can return dummy values for Guest reads and ignore writes, effectively giving the Guest the impression that the device does not exist on the platform.
 - ▶ When the device model has some data to deliver, the hypervisor raises a virtual IRQ (vIRQ).
 - ▶ The Guest OS responds by attempting to read hardware registers. The hypervisor traps these accesses and provides simulated responses.

Device assignment

- ▶ Need to hide from the Guest the fact that the device is at a different physical address
 - ▶ Alternatively, the hypervisor might choose to hide a device from a set of guests - either because the device is not present or has already been assigned to a different guest.
- ▶ Generate a different interrupt ID than that which the guest is expecting.

Sources (1)

- ▶ David Brash, **Extensions to the ARMv7-A Architecture**, HotChips 2010
- ▶ John Goodacre, **Hardware accelerated Virtualization in the ARM Cortex Processors**, XenSummit Asia 2011
- ▶ Roberto Mijat and Andy Nightingale, **Virtualization is Coming to a Platform Near You: The ARM Architecture Virtualization Extensions and the importance of System MMU for virtualized solutions and beyond**, ARM White paper, 2011
- ▶ Christoffer Dall, Shih-Wei Li, Jin Tack Lim, Jason Nieh, and Georgios Koloventzos, **ARM Virtualization: Performance and Architectural Implications**, In Proceedings of the 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture

Sources (2)

- ▶ AArch64 virtualization Documentation
 - ▶ https://static.docs.arm.com/100942/0100/aarch64_virtualization_100942_0100_en.pdf?_ga=2.124504458.128789899.1557134708-1811630367.1537190275
- ▶ AArch64 virtualization: Hypervisor software
 - ▶ <https://developer.arm.com/docs/100942/latest/hypervisor-software>
- ▶ Julien Grall: **“Hypervisors on ARM: Overview and Design choices”**, Root Linux Conference 2017
 - ▶ Slides: https://www.slideshare.net/xen_com_mgr/rootlinux17-hypervisors-on-arm-overview-and-design-choices-by-julien-grall-arm
 - ▶ Video: <https://www.youtube.com/watch?v=jZNXtqFJpuc>
- ▶ ARMv8.1-A: <https://goo.gl/Ox4thV>
- ▶ ARMv8.2-A: <https://goo.gl/0Ns37U>
- ▶ ARMv8.3-A: <https://goo.gl/CJv1n0>