Linux on AArch64
ARM 64-bit Architecture

Catalin Marinas

LinuxCon North America 2012

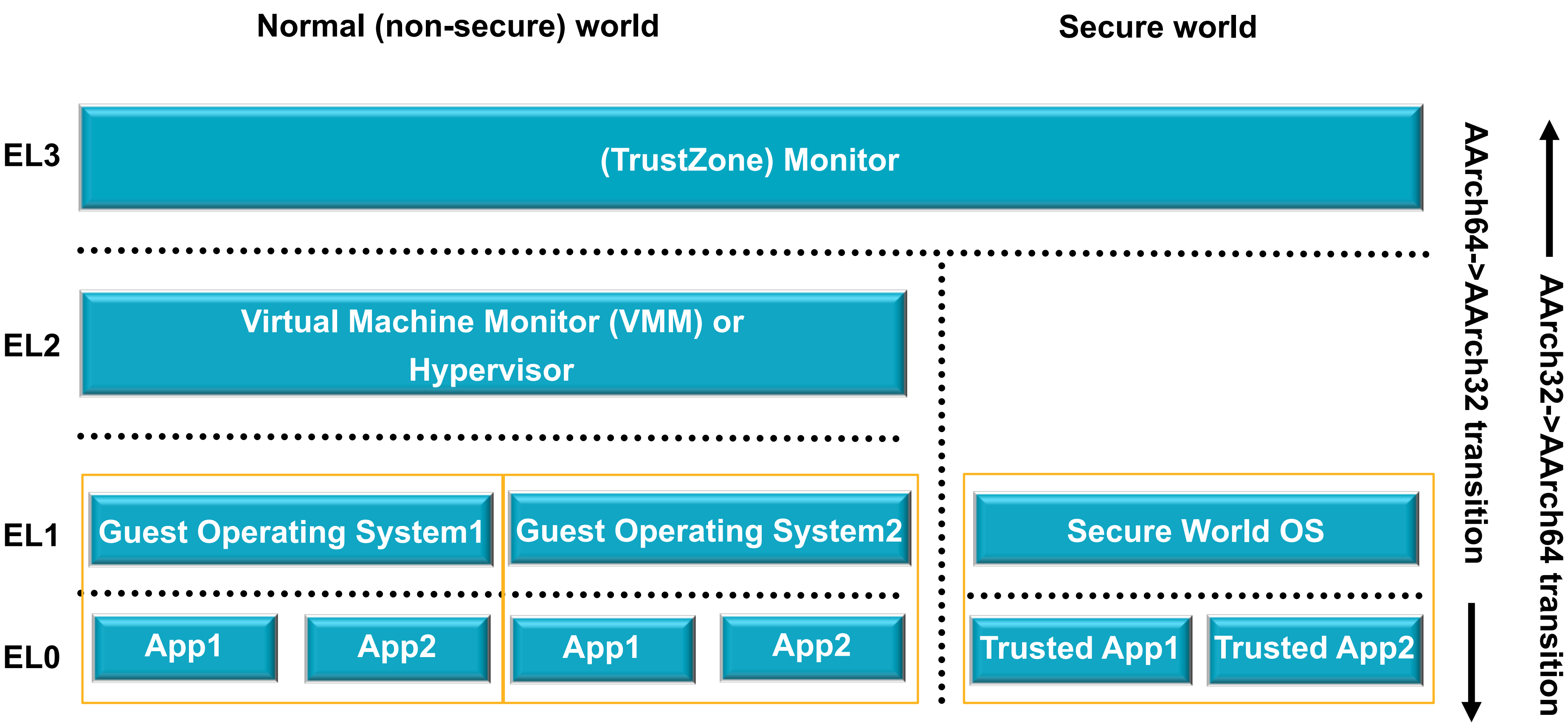
Introduction

- Previous ARM architecture, ARMv7, is 32-bit only
  - Cortex-* processors family
  - LPAE and virtualisation support
- The latest ARM architecture, ARMv8, introduces 64-bit capability alongside the existing 32-bit mode
  - First release covers the Applications processor profile
  - Addresses the need for larger virtual address space and high performance
  - Targets both mobile and server markets
- ARMv8 has two execution modes
  - AArch64 – 64-bit registers and memory accesses, new instruction set
  - AArch32 (optional) – backwards compatible with ARMv7-A
    - Few additional enhancements

AArch64 Overview

- New instruction set (A64)
  - 32-bit opcodes
  - Can have 32-bit or 64-bit arguments
  - Addresses assumed to be 64-bit
    - Primarily targeting LP64 and LLP64 data models
  - Only conditional branches, compares and selects
  - No LDM/STM (only pair load/store – LDP/STP)
  - Load-acquire/store-release exclusive accesses (implicit barrier)
  - Advanced SIMD and FP support
    - FP mandated by the ABI
  - Cryptography support
- 31 general purpose 64-bit registers (X0-X30)
  - PC, SP are special registers
  - Dedicated zero register (Xzr)

AArch64 Exception Model

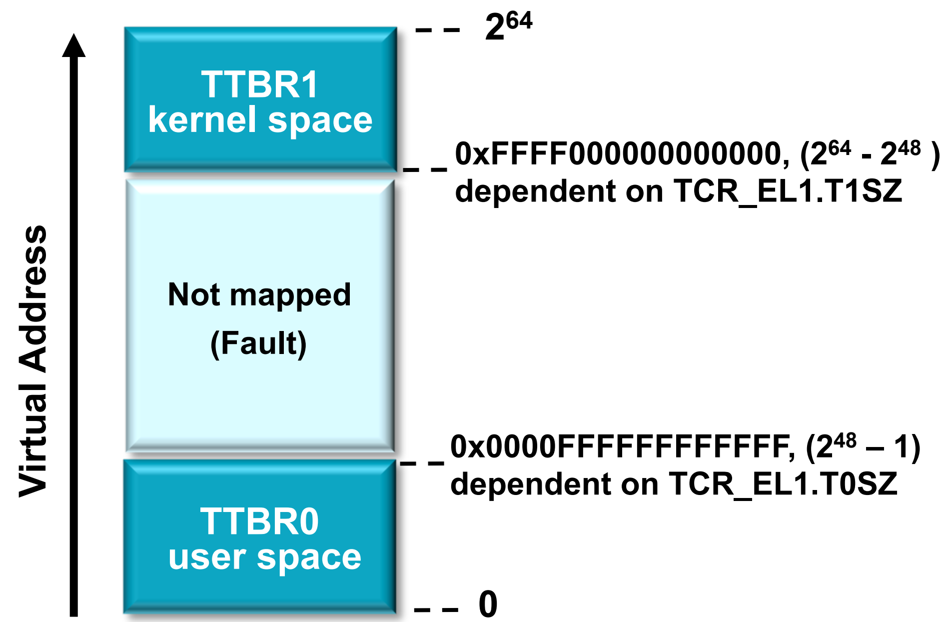


AArch64 Exception Model

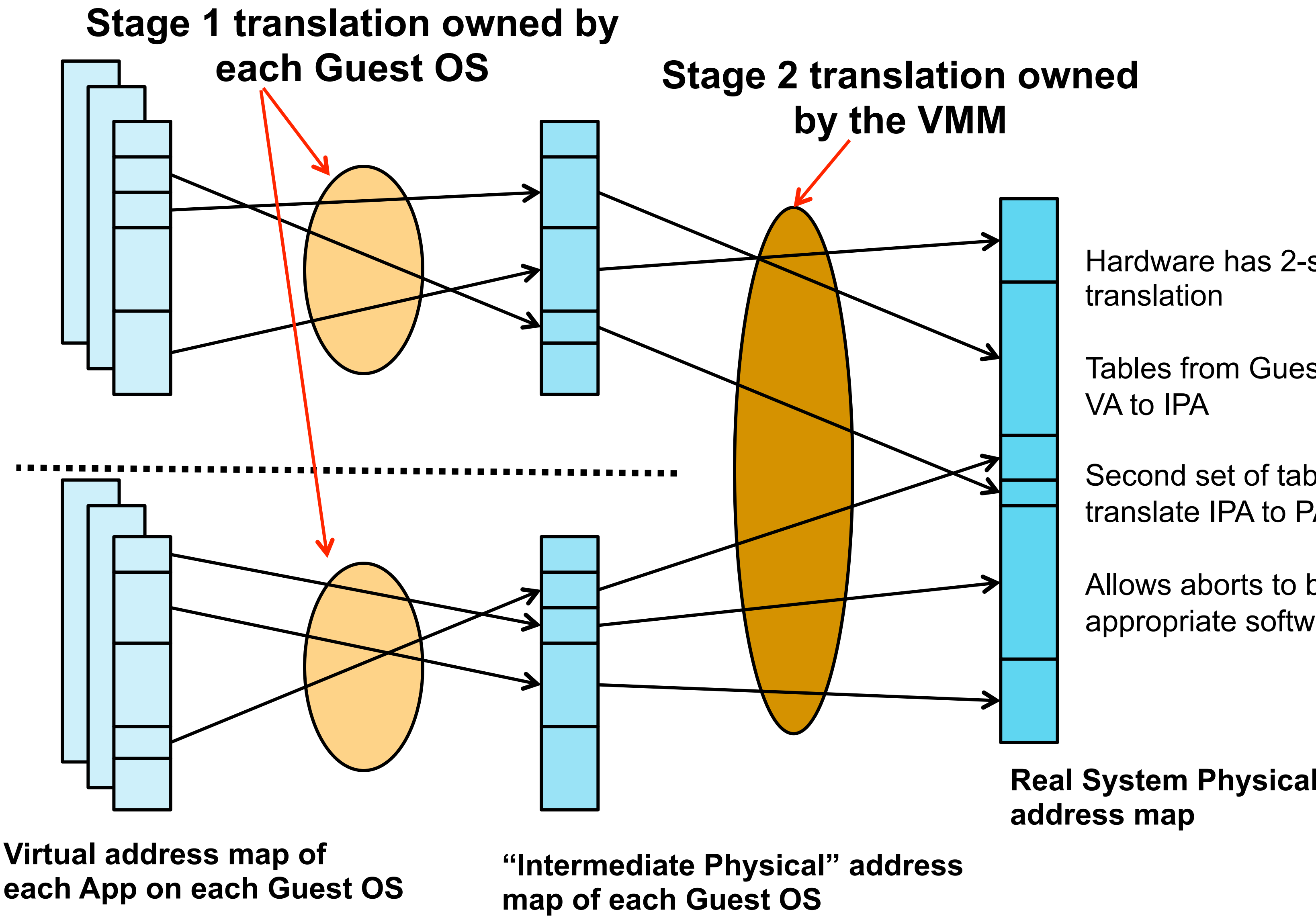
- Privilege levels: EL3 – highest, EL0 – lowest
- Transition to higher levels via exceptions
  - Interrupts, page faults etc.
  - SVC for transition to EL1 (system calls)
  - HVC for transition to EL2 (hypervisor calls)
  - SMC for transition to EL3 (secure monitor call)
  - Dedicated ELR register for the return address (banked at each EL)
- Transition to lower levels using the ERET instruction
- Register width cannot be higher in lower levels
  - E.g. no 64-bit EL0 with 32-bit EL1
- Transition between AArch32 and AArch64 via exceptions
  - AArch32/AArch64 interworking not possible
- Separate stack pointer (SP) at each EL

AArch64 MMU Support

- Separate TTBR register for user and kernel
  - Selection based on higher bits of the virtual address
  - Maximum 48-bit virtual address for each TTBR
- Upper 8 bits of the address can be configured for Tagged Pointers
  - Linux does not currently use them
- Maximum 48-bit physical address
- 2-stage translation



AArch64 MMU Support



Hardware has 2-stage memory translation

Tables from Guest OS translate VA to IPA

Second set of tables from VMM translate IPA to PA

Allows aborts to be routed to appropriate software layer

AArch64 MMU Support

- Two different translation granules: 4KB and 64KB
  - The smallest page mapping supported
  - The size of a translation table
  - Can be independently configured for TTBR0 and TTBR1
- Number of translation tables and maximum VA range:
  - 4KB and 4 levels => 48-bit VA
  - 64KB and 3 levels => 48-bit VA (top table partially populated)
  - 4KB and 3 levels => 39-bit VA (currently used by AArch64 Linux)
  - 64KB and 2 levels => 42-bit VA
- Large page (block) mapping supported
  - 2MB and 1GB with 4KB page configuration
  - 512MB with 64KB page configuration

AArch32 Support

- AArch32 can be optionally present at any level
  - EL0 most likely for user application support
  - EL1 needed for 32-bit guest OS
- Execution state change only at exception entry/return
  - No branch and link (interworking) between AArch32 and AArch64
  - Increasing EL cannot decrease register width or vice versa
- Architected relationship between the AArch32 and AArch64 registers
  - AArch32 Rn registers accessed via corresponding AArch64 Xn registers

AArch64 Linux Overview

- New architecture port: arch/arm64/
- Re-using generic code and data
  - asm-generic/unistd.h
- Building requires aarch64-linux-gnu toolchain
  - No common AArch32/AArch64 toolchain
- Support for both AArch64 and AArch32 (compat) user applications
- VDSO
  - Signal return code
  - Optimised gettimeofday()
- Not fully optimised at this stage
  - Testing done on software model

Linux Kernel Booting

- Linux required to run in Normal (Non-secure) mode
  - Virtualisation extensions not available in secure mode
- Host OS must be started in EL2 mode for virtualisation support
  - Switches to EL1 shortly after boot, the default kernel execution mode
- Guest operating systems start in EL1
  - The Linux kernel automatically detects the exception level
- Kernel image loaded at a pre-defined address
  - Image file header contains the relevant information
- Standard booting protocol
  - Currently driven by FDT (mailbox address for CPU release)
  - Proposed standard secure API (SMC) for CPU boot, reset and power management (Power State Coordination Interface)

Linux MMU Handling

- 39-bit virtual address for both user and kernel
  - `0000000000000000-0000007fffffffff` (512GB): user
  - [architectural gap]
  - `ffffff8000000000-ffffffbbfffeffff` (~240MB): vmalloc
  - `ffffffbbffff0000-ffffffbcffffffff` (64KB): [guard]
  - `ffffffbc00000000-ffffffbdffffffff` (8GB): vmemmap
  - `ffffffbe00000000-ffffffbffbffffff` (~8GB): [guard]
  - `ffffffbffc000000-ffffffbfffffffff` (64MB): modules
  - `ffffffc000000000-ffffffffffffffff` (256GB): mapped RAM
- 4KB page configuration
  - 3 levels of page tables (pgtable-nopud.h)
  - Linear mapping using 4KB, 2MB or 1GB blocks
  - AArch32 (compat) supported

Linux MMU Handling

- 64KB page configuration
  - 2 levels of page tables (pgtable-nopmd.h)
  - Linear mapping using 64KB or 512MB blocks
  - AArch32 (compat) not supported because the 32-bit ABI assumes 4KB pages
- SPARSEMEM support
  - SPARSEMEM_VMEMMAP optimisation for virtual mapping of the struct page array (mem_map)
- Huge pages
  - Hugetlbfs
  - Transparent huge pages

Linux Exception Handling

- SP1 register used for the kernel stack (running in EL1)
  - Default 8KB size
- SP0 used for the user stack (running in EL0)
- Returning to user is done with the ERET instruction
  - Registers restored from the kernel stack (pt_regs) by the return code
  - Return address automatically restored from the ELR register
  - PSTATE automatically returned from SPSR
  - Mode switching to EL0_SP0
  - AArch64/AArch32 execution state selected by the PSTATE.nRW bit
- Kernel entered at EL1 as a result of an exception
  - Mode switching to EL1_SP1 and AArch64
  - Return address automatically saved to ELR
  - PSTATE automatically saved to SPSR

Linux Exception Handling

  - The general purpose registers saved onto the kernel stack (pt_regs) by the exception entry code
- Exceptions while in kernel similar to the user->kernel transition
  - No mode switching (no IRQ etc. modes)
  - Using the current stack
  - Returning with ERET but to the same exception level and stack

AArch32 (compat) Support

- Must support the ARMv7 Linux EABI for compat tasks
  - Different set of system calls (unistd32.h)
  - Compat user structures
  - No SWP instruction, no unaligned LDM/STM access
- Supports both ARM and Thumb-2 32-bit user tasks
- Supports 32-bit ptrace
- Address space limited to 4GB
- Emulated vectors page
  - ARM Linux EABI expects helper routines in the vectors page accessible by user tasks

Platform (SoC) Support

- Different targets: embedded systems and servers
- FDT currently mandated for new platforms
  - ACPI may be required, especially for servers
- Minimal platform code
  - Most code under drivers/
  - FDT for platform description
- Standardised firmware interface
  - Booting protocol
  - SMC API for CPU power management
- Generic (architected) timers
- Generic interrupt controller (GIC)

AArch64 Linux Roadmap

- AArch64 Linux kernel currently under public review
  - Initially only the core architecture support
- GCC and binutils patches published
- Collaborate with Linaro and the Linux community to bring broader filesystem and applications support to AArch64
- SoC support
  - Future ACPI support
- New features
  - Huge pages
  - KVM
  - NUMA
- Power and performance improvements
  - When hardware becomes available

Reference

- AArch64 Linux Git tree
  - git://git.kernel.org/pub/scm/linux/kernel/git/cmarinas/linux-aarch64.git
- AArch64 instruction set
  - http://infocenter.arm.com/help/topic/com.arm.doc.genc010197a/index.html
- AArch64 ABI (PCS, ELF, DWARF, C++)
  - http://infocenter.arm.com/help/topic/com.arm.doc.ihi0059a/index.html
- Power State Coordination Interface
  - http://infocenter.arm.com/help/topic/com.arm.doc.den0022a/index.html

Questions