2011 ARM® TechCon™
Join the community defining the future

ARMv8 Technology Preview

By Richard Grisenthwaite
Lead Architect and Fellow. ARM

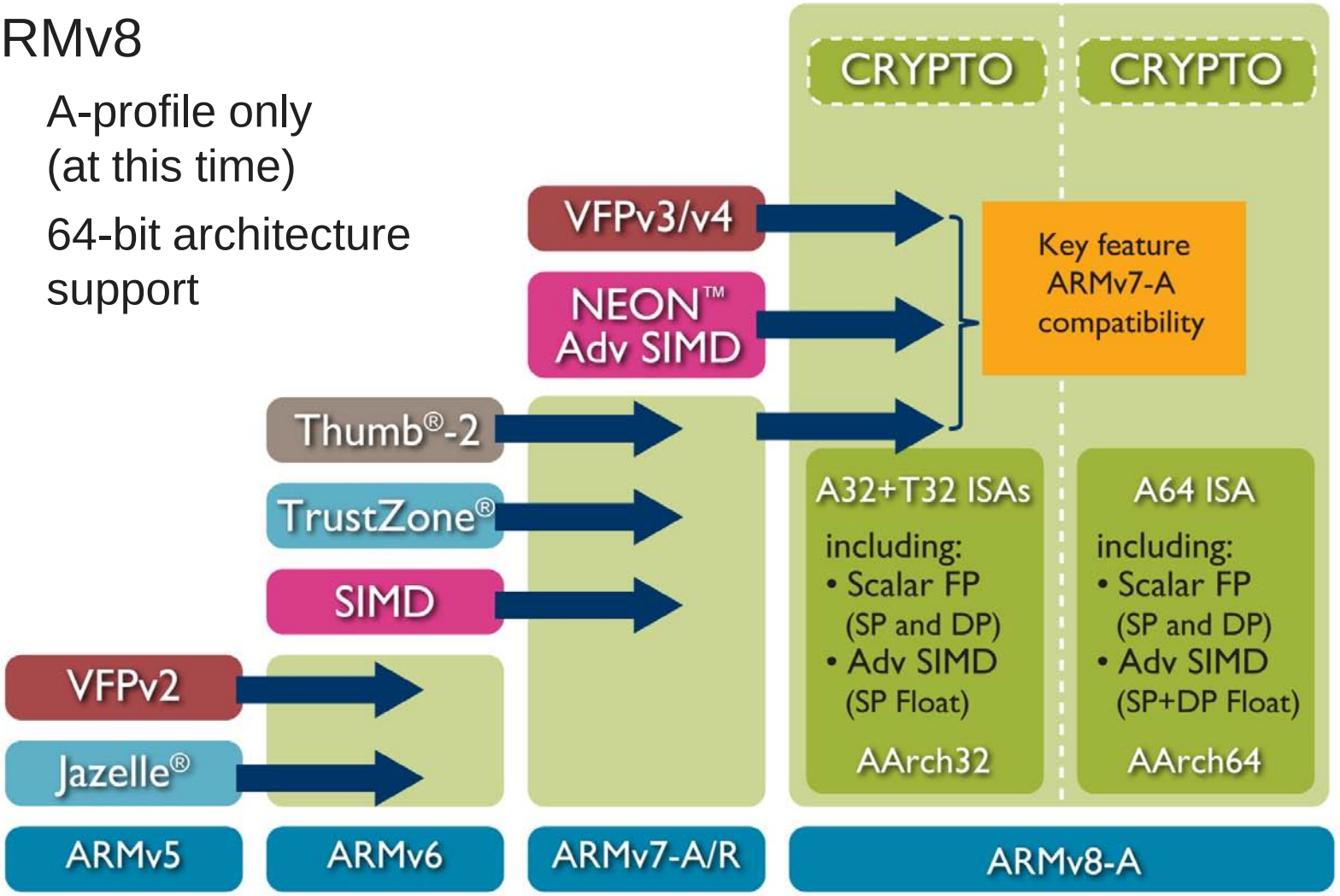
What is ARMv8?

- Next version of the ARM architecture
  - First release covers the Applications profile only
- Addition of a 64-bit operating capability alongside 32-bit execution
  - AArch64 state alongside AArch32 state
  - Focus on power efficient architecture advantages in both states
- Definition of relationship between AArch32 state and AArch64 state
- Enhancement to the AArch32 functionality
  - Relatively small scale additions reflecting demand
  - Maintaining full compatibility with ARMv7

ARMv8-A – Context

ARMv8

- A-profile only (at this time)
- 64-bit architecture support



AArch64 - Motivation

- Work on 64-bit architecture started in 2007
- Fundamental motivation is evolution into 64-bit
  - Ability to access a large virtual address space
  - Foresee a future need in ARM's traditional markets
  - Enables expansion of ARM market presence
- Developing ecosystem takes time
  - Development started ahead of strong demand
  - ARM now seeing strong partner interest in 64-bit
    - Though still some years from "must have" status

AArch64 State Fundamentals

- New instruction set (A64)
- Revised exception handling for exceptions in AArch64 state
  - Fewer banked registers and modes
- Support for all the same architectural capabilities as in ARMv7
  - TrustZone
  - Virtualization
- Memory translation system based on the ARMv7 LPAE table format
  - LPAE format was designed to be easily extendable to AArch64-bit
  - Up to 48 bits of virtual address from a translation table base register

A64 New Instruction Set - 1

- New fixed length Instruction set
  - Instructions are 32-bits in size
  - Clean decode table based on a 5-bit register specifiers
- Instruction semantics broadly the same as in AArch32
  - Changes only where there is a compelling reason to do so
- 31 general purpose registers accessible at all times
  - Improved performance and energy
  - General purpose registers are 64-bits wide
  - No banking of general purpose registers
  - Stack pointer is not a general purpose register
  - PC is not a general purpose register
  - Additional dedicated zero register available for most instructions

A64 – Key differences from A32

- New instructions to support 64-bit operands
  - Most instructions can have 32-bit or 64-bit arguments
  - Addresses assumed to be 64-bits in size
    - LP64 and LLP64 are the primary data models targeted
- Far fewer conditional instructions than in AArch32
  - Conditional {branches, compares, selects}
- No arbitrary length load/store multiple instructions
  - LD/ST ‘P’ for handling pairs of registers added

A64 Advanced SIMD and FP Instruction Set

- A64 Advanced SIMD and FP semantically similar to A32
  - Advanced SIMD shares the floating-point register file as in AArch32
- A64 provides 3 major functional enhancements:
  - More 128 bit registers: 32 x 128 bit wide registers
    - Can be viewed as 64-bit wide registers
  - Advanced SIMD supports DP floating-point execution
  - Advanced SIMD support full IEEE 754 execution
    - Rounding-modes, Denorms, NaN handling
- Register packing model in A64 is different from A32
  - 64-bit register view fit in bottom of the 128-bit registers
- Some Additional floating-point instructions for IEEE754-2008
  - MaxNum/MinNum instructions, Float to Integer conversions with RoundTiesAway

Cryptography Support

- Instruction level support for Cryptography
  - Not intended to replace hardware accelerators in an SoC
- AES
  - 2 encode and 2 decode instructions
    - Work on the Advanced SIMD 128-bit registers
    - 2 instructions encode/decode a single round of AES
- SHA-1 and SHA-256 support
  - Keep running hash in two 128 bit wide registers
  - Hash in 4 new data words each instruction
  - Instructions also accelerate key generation

AArch64 – Unbanked Registers

64-bit General Purpose Register file used for:

- Scalar Integer computation
  - 32-bit and 64-bit
- Address computation
  - 64-bit

Media Register File used for:

- Scalar Single and Double Precision FP
  - 32-bit and 64-bit
- Advanced SIMD for Integer and FP
  - 64- or 128-bit wide vectors
- Cryptography

X0	X8	X16	X24
X1	X9	X17	X25
X2	X10	X18	X26
X3	X11	X19	X27
X4	X12	X20	X28
X5	X13	X21	X29
X6	X14	X22	X30*
X7	X15	X23	

V0	V8	V16	V24
V1	V9	V17	V25
V2	V10	V18	V26
V3	V11	V19	V27
V4	V12	V20	V28
V5	V13	V21	V29
V6	V14	V22	V30
V7	V15	V23	V31

AArch64 Banked Registers

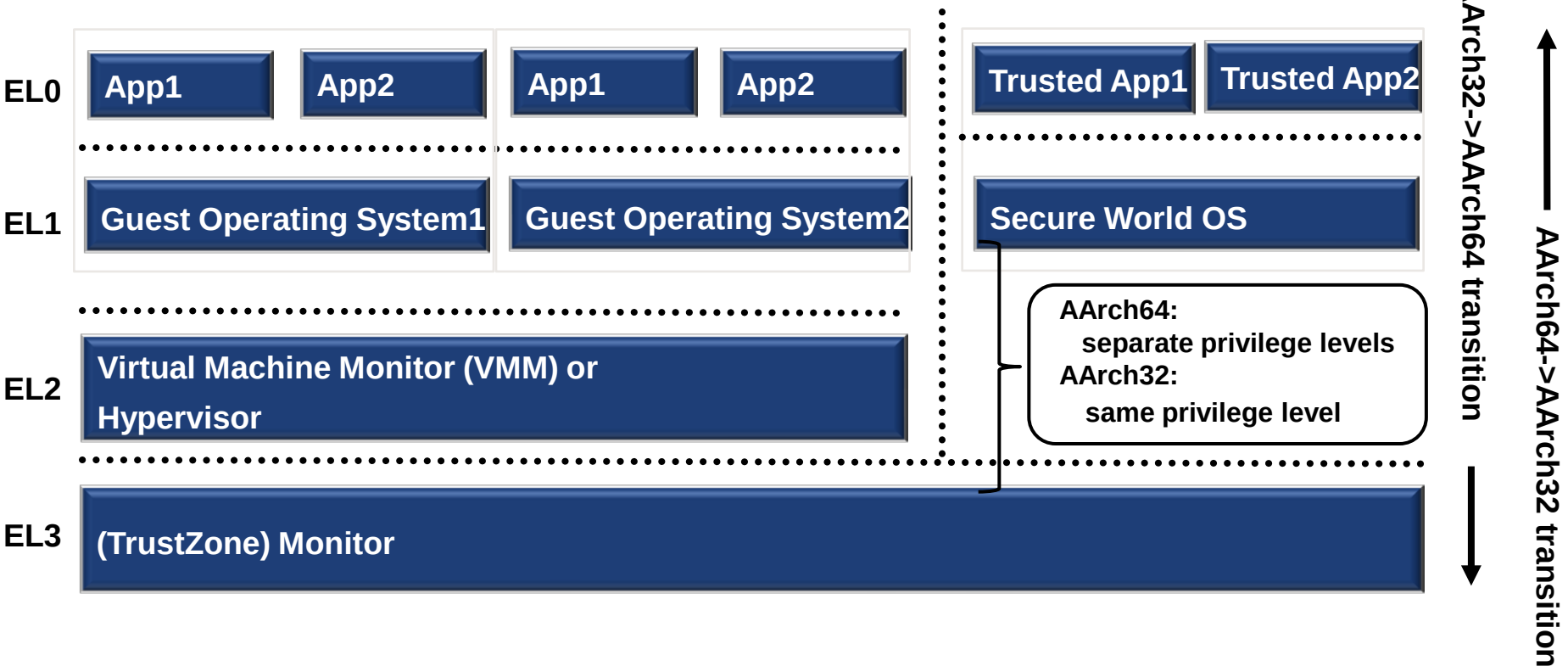
- AArch64 Banked registers are banked by exception level
- Used for exception return information and stack pointer
- EL0 Stack Pointer can be used by higher exception levels after exception taken

	EL0	EL1	EL2	EL3	
SP = Stack Ptr	SP_EL0	SP_EL1	SP_EL2	SP_EL3	
ELR = Exception Link Register		ELR_EL1	ELR_EL2	ELR_EL3	(PC)
Saved/Current Process Status Register		SPSR_EL1	SPSR_EL2	SPSR_EL3	(CPSR)

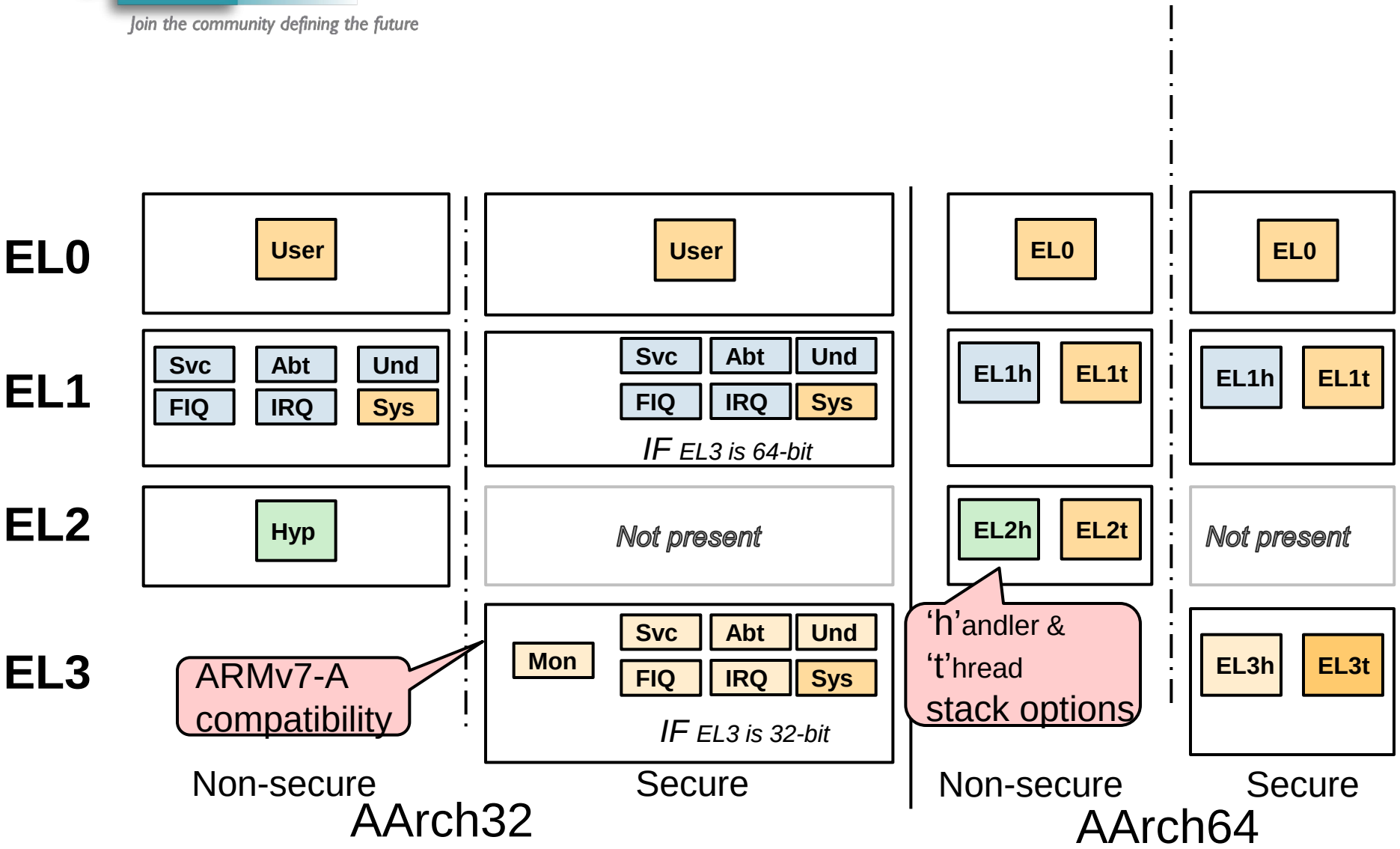
Exception Model for AArch64

- 4 exception levels: EL3-EL0
  - Forms a privilege hierarchy, EL0 the least privileged
- Exception Link Register written on exception entry
  - 32-bit to 64-bit exception zero-extends the Link Address
  - Interrupt masks set on exception entry
- Exceptions can be taken to the same or a higher exception level
  - Different Vector Base Address Registers for EL1, EL2, and EL3
- Vectors distinguish
  - Exception type: synchronous, IRQ, FIQ or System Error
  - Exception origin (same or lower exception level) and register width
- Syndrome register provides exception details
  - Exception class
  - Instruction length (AArch32)
  - Instruction specific information

ARMv8 Exception Model

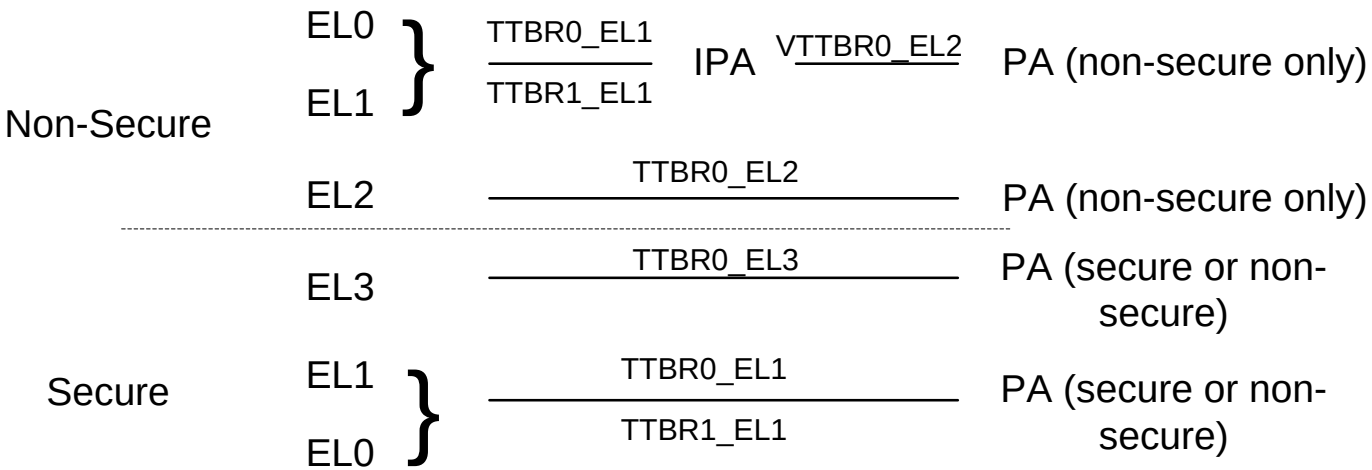


Exception model nomenclature

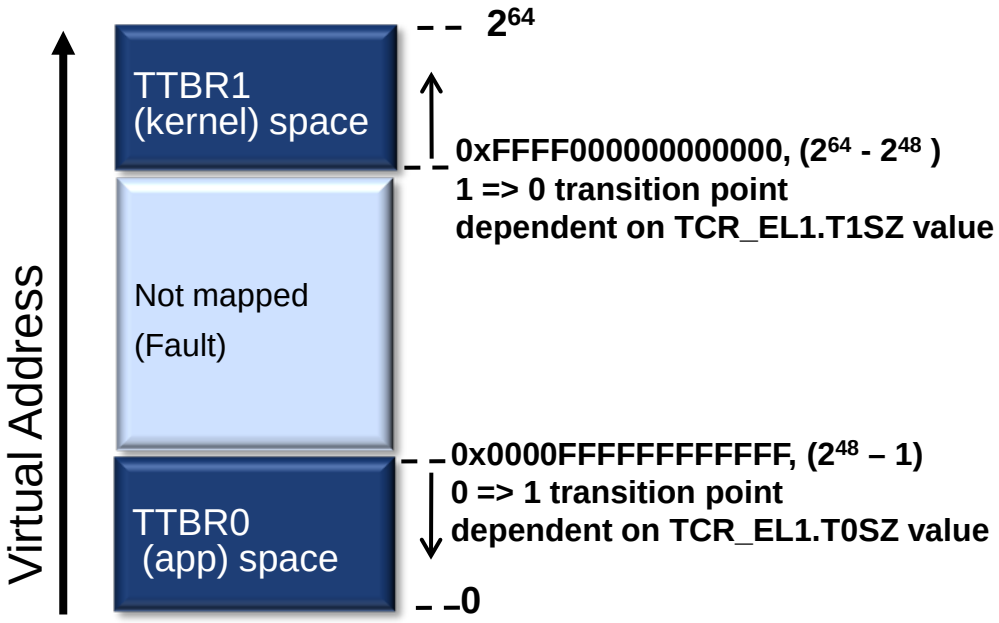


EL1/EL2/EL3 translation contexts

- Exception levels above EL0 manage their own translation context
  - Translation base address, control registers, exception syndrome etc
  - EL0 translation managed by EL1
- EL2 manages an additional stage2 of translation for EL1/EL0
  - For EL1/EL0 in the Non-secure state only



AArch64 MMU Support



- 64-bit architecture gives a larger address space
  - However little demand this time for all 16 Exabytes
- Supporting up to 48 bits of VA space for each TTBR
  - Actual size configurable at run-time
  - Number of levels of translation table walk depends on address size used
- Upper 8 bits of address can be configured for Tagged Pointers
  - Meaning interpreted by software
- IPA supports up to 48 bits on same basis
- Supporting up to 48 bits of PA space
  - Discoverable configuration option

Translation Granules

- AArch64 supports 2 different translation granules
  - 4KBytes or 64KBytes
  - Configurable for each TTBR
- Translation granule is:
  - Size of the translation tables in the memory system
  - Size of the smallest page supported
- Larger translation granule gives markedly flatter translation walk
  - Particularly where 2 stages of translation are in use

ARMv8-A: page table information

VA Bits <47:39>	VA Bits <38:30>	VA Bits <29:21>	VA Bits <20:12>	VA Bits <11:0>
Level 1 table index	Level 2 table index	Level 3 table index	Level 4 table (page) index	Page offset address

- 4-level lookup, 4KB translation granule, 48-bit address
  - 9 address bits per level

VA Bits <41:29>	VA Bits <28:16>	VA Bits <15:0>
Level 1 table index	Level 2 table (page) index	Page offset address

- 2-level lookup, 64KB page/page table size, 42-bit address
  - 13 address bits per level
  - 3 levels for 48 bits of VA – top level table is a partial table

63		52		48		12		2	1	0
Upper attributes		SBZ		Address out		SBZ	Lower attributes and validity			

- 64-bit Translation table entry format

AArch64 Memory model

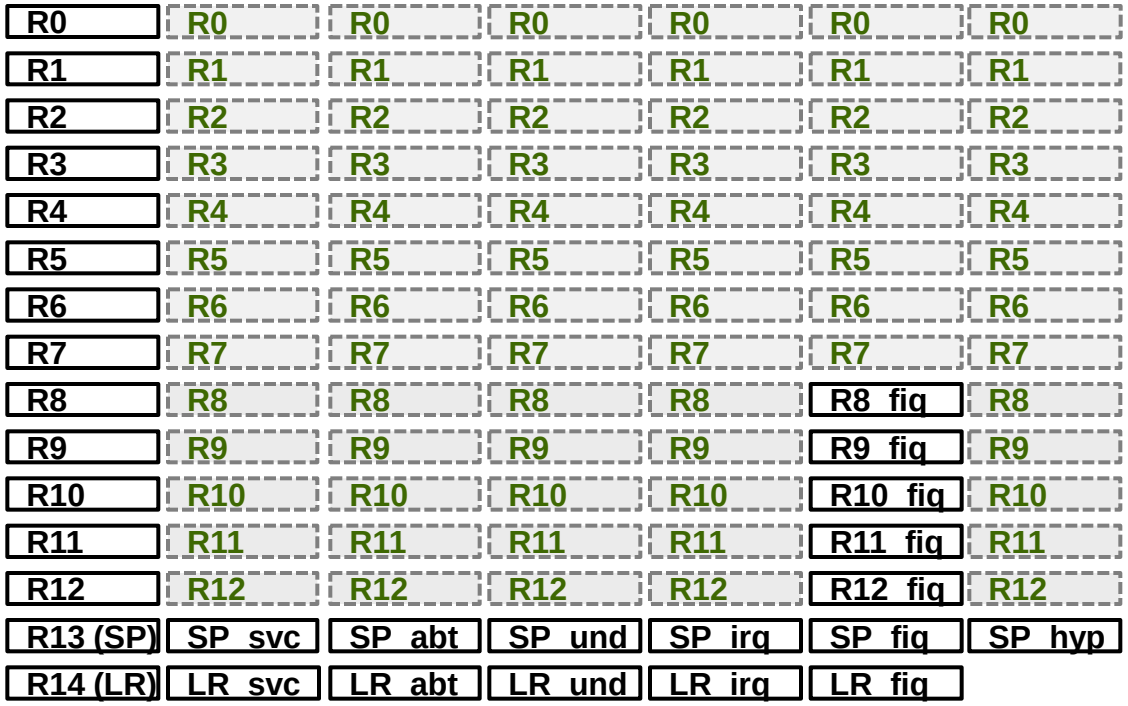
- ARM architecture has a Weak memory model
  - Good for energy
- Aligned with emerging language standardization
  - C++11/C1x memory models and related informal approach
- AArch64 adds load-acquire/store-release instructions
  - Added for all single general purpose register load/stores
  - Versions added for load-exclusive/store-exclusive as well
  - Follows RCsc model
    - Store-release -> Load-acquire is also ordered
  - Strong fit to the C++11/C1x SC Atomics
    - Best fit of any processor architecture

AArch32 /AArch64 relationship

- Changes between AArch32 and AArch64 occur on exception/exception return only
  - Increasing exception level cannot decrease register width (or vice versa)
  - No Branch and Link between AArch32 and AArch64
- Allows AArch32 applications under AArch64 OS Kernel
  - Alongside AArch64 applications
- Allows AArch32 guest OS under AArch64 Hypervisor
  - Alongside AArch64 guest OS
- Allows AArch32 Secure side with AArch64 Non-secure side
  - Protects AArch32 Secure OS investments into ARMv8
- Requires architected relationship between AArch32 and AArch64 registers

AArch32/AArch64 relationship

Register State relationships below EL3



X0 ⇔ R0
X1 ⇔ R1
X2 ⇔ R2
X3 ⇔ R3
X4 ⇔ R4
X5 ⇔ R5
X6 ⇔ R6
X7 ⇔ R7
X8 ⇔ R8usr
X9 ⇔ R9usr
X10 ⇔ R10usr
X11 ⇔ R11usr
X12 ⇔ R12usr
X13 ⇔ R13usr
X14 ⇔ R14usr
X15 ⇔ R13_hyp

X16 ⇔ R14_irq
X17 ⇔ R13_irq
X18 ⇔ R14_svc
X19 ⇔ R13_svc
X20 ⇔ R14_abt
X21 ⇔ R13_abt
X22 ⇔ R14_und
X23 ⇔ R13_und
X24 ⇔ R8_fiq
X25 ⇔ R9_fiq
X26 ⇔ R10_fiq
X27 ⇔ R11_fiq
X28 ⇔ R12_fiq
X29 ⇔ R13_fiq
X30 ⇔ R14_fiq

SPSR_EL1 ⇔ SPSR_svc
SPSR_EL2 ⇔ SPSR_hyp
ELR_EL2 ⇔ ELR_hyp

AArch32

SPSR_svc SPSR_abt SPSR_und SPSR_irq SPSR_fiq SPSR_hyp ELR_hyp

AArch64

SP_EL0-2
ELR_EL1

AArch32 Enhancements

- ARMv8 includes enhancements to AArch32
  - Brings in new functionality independent of register width
  - **ARMv8 is not the end of the road for AArch32**
- Main enhancements:
  - Load acquire/store release and improved barriers
  - Cryptography instructions
  - Some additional improvements for IEEE754-2008

Debug in ARMv8

- ARM Hardware Debug support falls into 2 basic categories:
  - **Self-hosted** debug for debug facilities used by the operating system/hypervisor
  - **Halting** debug for external “target debug” where debug session is run on a separate host
- Self-hosted debug is basically part of the exception model
  - Hardware watchpoints and breakpoints to generate exceptions on debug events
  - Exceptions handled by a debug monitor alongside the OS Kernel or Hypervisor
  - AArch32 self-hosted (“monitor”) capability unchanged from ARMv7
- AArch64 self-hosted debug is strongly integrated into AArch64 exception model
  - Breakpoint and Watchpoint Addresses grow to 64-bits
  - Introduces an explicit hardware single step when debug monitor using AArch64
- Halting Debug view is not backwards compatible with ARMv7
  - External Debugger will need to change - even for fully AArch32 operation

Trace Support

- Embedded Trace in the Cortex-A profile limited to program flow trace
  - Shows the “waypoints” of instruction execution
  - Does not provide address or data value information
  - ARMv7 current position for Cortex-A9 and Cortex-A15
- New ETM protocol (ETMv4) works with ARMv8
  - Widens addresses to 64 bits
  - Better compression than ETMv3
  - For ARMv8 A-profile, will only support waypoint information

ARMv8-A rollout

- Plenty of headroom for ARMv7 in many markets
  - ARMv7-A is today, ARMv8-A is tomorrow
  - AArch64 ecosystem will take time to develop – need to start this process more widely
- TechCon 2011 – developer preview for ARMv8-A
  - Begins process of revealing ARMv8-A to wider developer community
  - Enables open and informed discussion of the topic – by ARM and partners
- 2012 – will start seeing up-streaming open-source materials
  - Detailed specifications planned to be released in the second half of 2012
- ARM is working with architectural partners and on its own implementations
- **No Product announcement from ARM for ARMv8-A at this time**

Conclusion

- ARM is well advanced in development of ARMv8-A
  - ARMv8-A is the largest architecture change in ARM's history
  - Positions ARM to continue servicing current markets as their needs grow
- Cortex-A15 & other ARMv7 parts are the top end for ARM today
  - Provide a lot of capability for the next few years
- Architectural roadmap into the future now clear