Linaro Connect Sept 2021

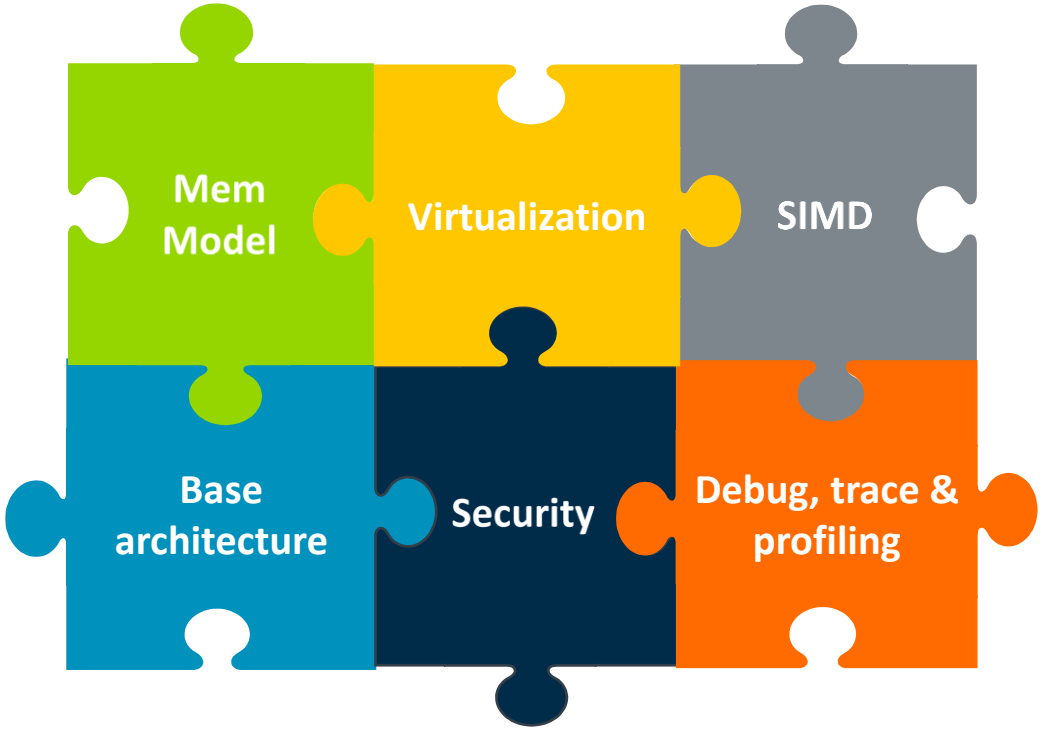
Arm Architecture 2021 Extensions

Martin Weidmann
Director Product Management, ATG ARM

LVC21F-113

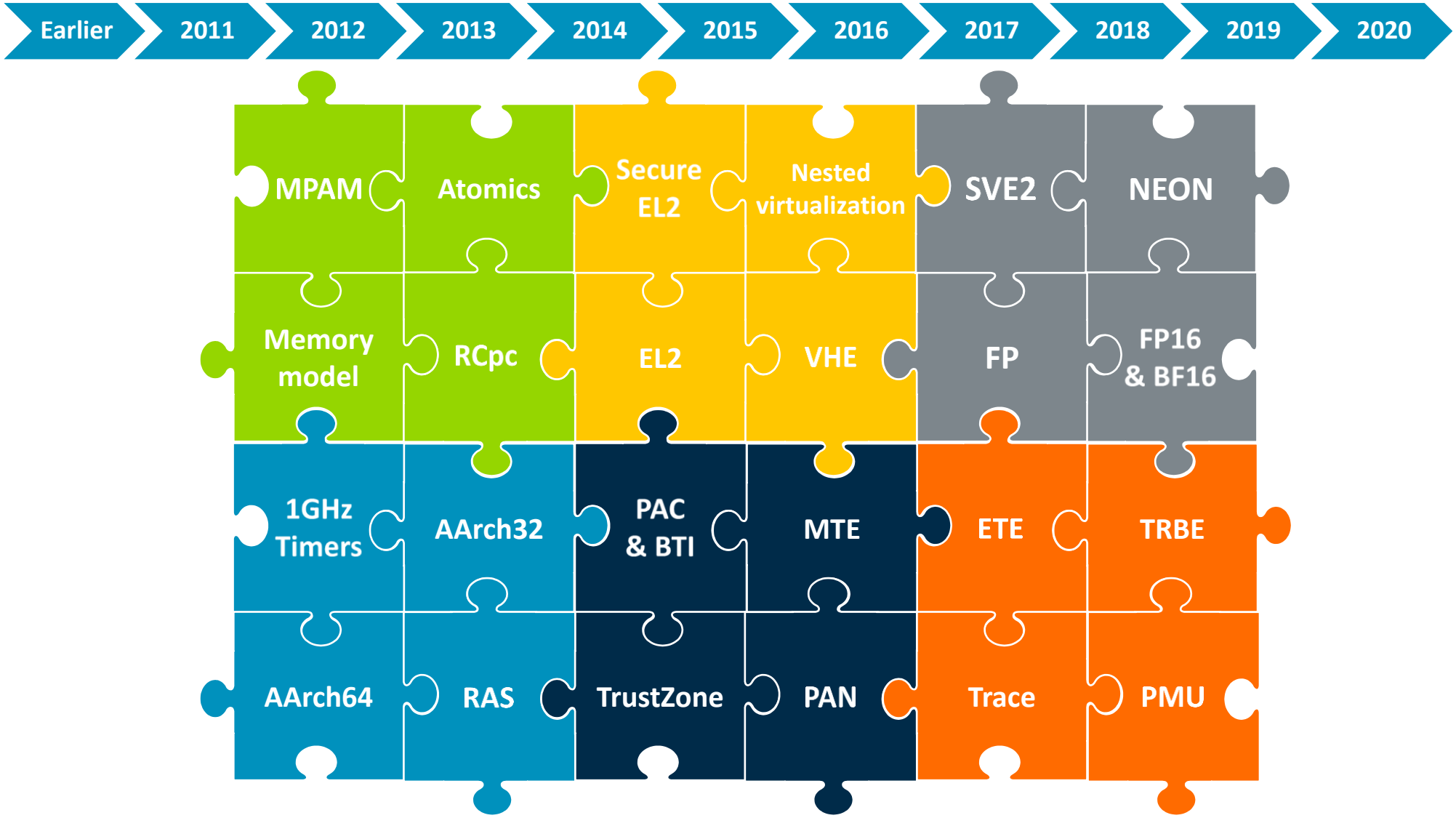
© 2021 Arm

Features in the architecture



© 2021 Arm

Features up to 2020



© 2021 Arm

What's new in 2021?

Earlier | 2011 | 2012 | 2013 | 2014 | 2015 | 2016 | 2017 | 2018 | 2019 | 2020 | **2021**

memcpy

- Instructions optimized for memcpy()/memset() family of functions
- Enabling a standard optimized implementation across different micro-architectures

NMI

- Support for non-maskable interrupts in the CPU and interrupt controller
- Flexible support that works with existing priority schemes

EL0 cache op permissions

Hinted branches

PMU cache events

PMU thresholds

EL3 RNG traps

QARAM3

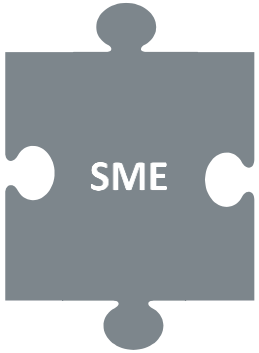
© 2021 Arm

Already announced in 2021

Realm Management Extension

- Architecture extension for Arm's Confidential Compute Architecture (Arm CCA)
  - Adds new Security states and physical address spaces
- Find out more on:
  - https://www.linaro.org/events/linaro-and-arm-cca-tech-day-deep-dive-into-arm-confidential-compute-architecture/

Scalable Matrix Extension



- Builds on SVE and SVE2, improving efficiency and through put for matrix operations
- Find out more on:
  - https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/scalable-matrix-extension-armv9-a-architecture

© 2021 Arm

memcpy/memset

FEAT_MOPS

© 2021 Arm

Memcpy()/memset() family of functions

Enabling a standard optimal implementation across Arm platforms

- The memcpy() family of functions are used extensively within software
  - Good implementations are an important part of the performance and efficiency of a system
- Traditional RISC approach is to build these functions from standard instructions, but…
  - The optimal implementation differs depending on the micro-architecture, size, alignment, etc
    - Profiling shows large number of calls are on very small (or 0) amounts of data
    - Overhead of picking correct variants large compared to time needed to copy the data
  - Software burden on maintaining multiple variants of the function
- Arm is introducing new load/store instructions targeted at memcpy()/memset()/…
  - Enabling a standard optimal sequence
  - Hardware developers know what sequence to optimize for
  - Software developers don’t need to re-implement memcpy() for each new design

© 2021 Arm

New instructions

Optimized for memcpy()/memset()/…

memcpy()/memmove()

```
CPY[F]Px [dst]!, [src]!, num_bytes!
CPY[F]Mx [dst]!, [src]!, num_bytes!
CPY[F]Ex [dst]!, [src]!, num_bytes!
```

- Three instruction sequence:
  - `CPYP` performs pre-conditioning
  - `CPYM` performs the operation
  - `CPYE` performs finalizes operation
- Exceptions can be taken part way through copy
- Options to control direction, non-temporal and whether accesses are privileged

memset()

```
SETPx  [dst]!, num_bytes!, src_data
SETMx  [dst]!, num_bytes!, src_data
SETEx  [dst]!, num_bytes!, src_data
```

- As with CPY, three instruction sequence:
  - `SETP` performs pre-conditioning
  - `SETM` performs the operation
  - `SETE` performs finalizes operation
- Tag setting variants available

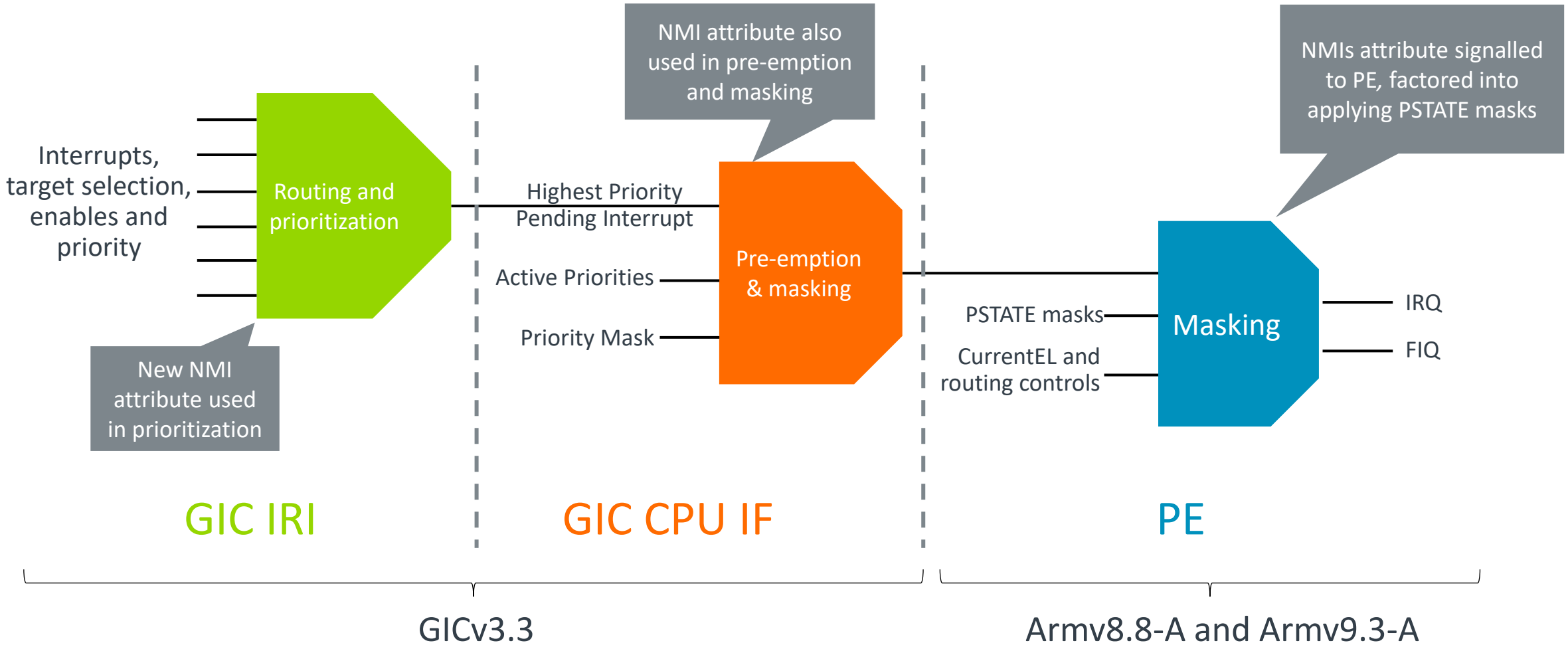
© 2021 Arm

Non-maskable interrupts

FEAT_NMI, FEAT_GICv3_NMI

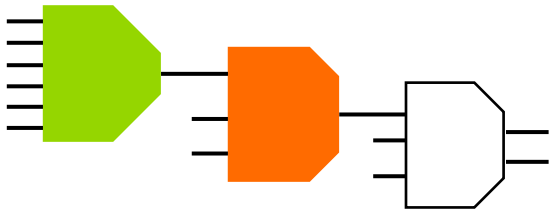
© 2021 Arm

NMI support

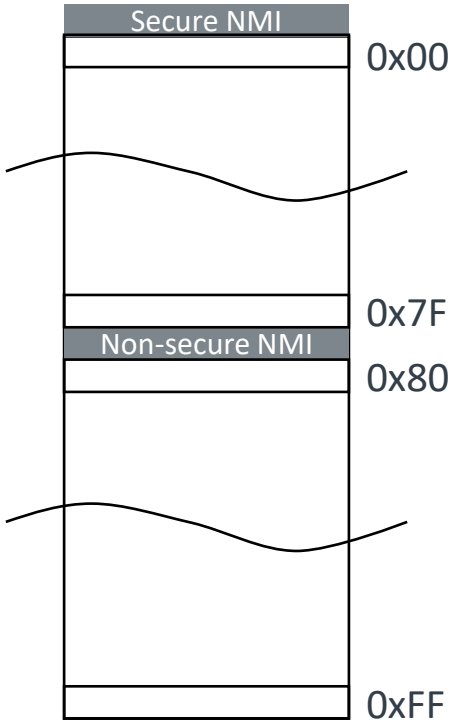


© 2021 Arm

GICv3.3 – Non-maskable property



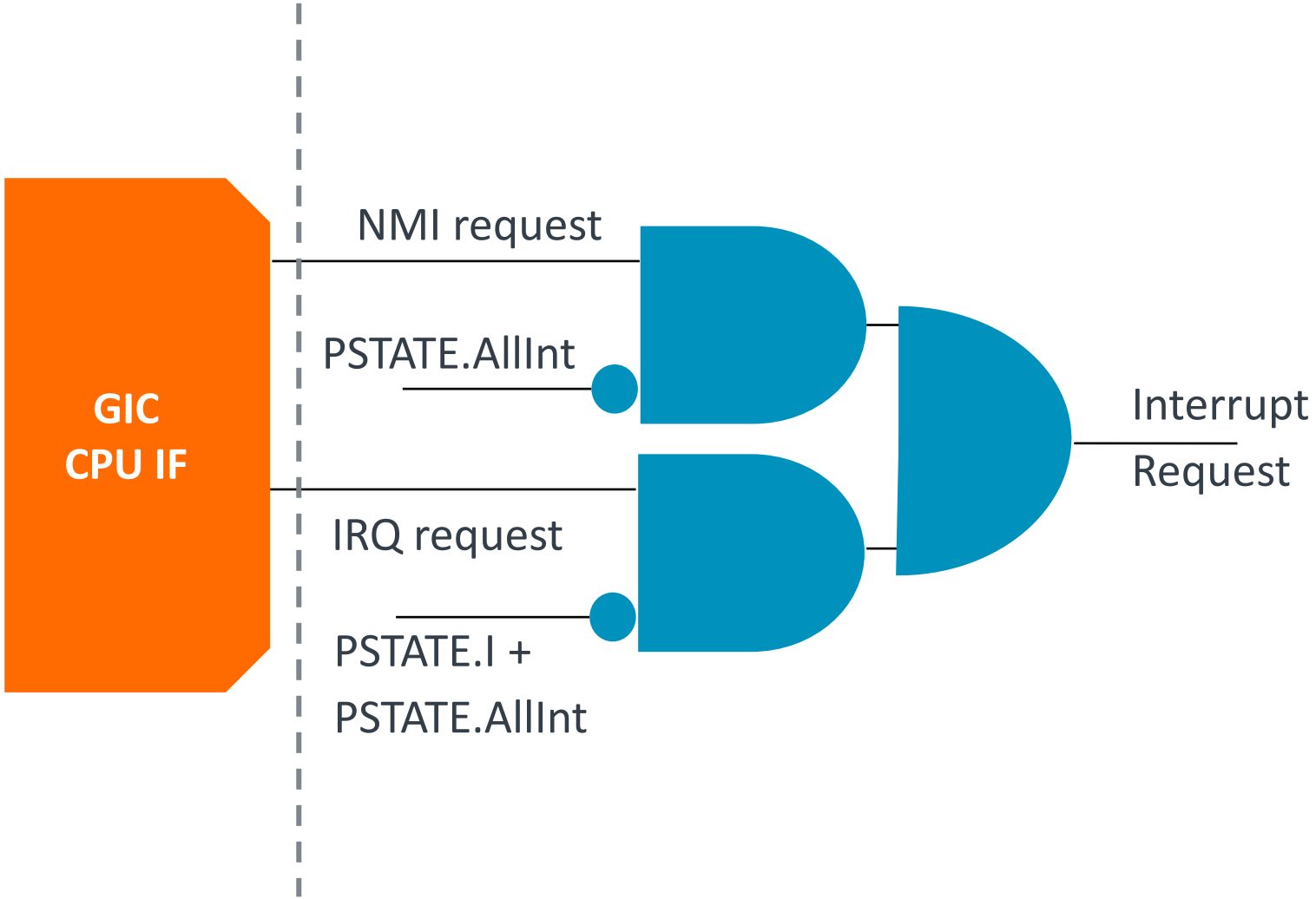
- Non-maskable property for SPIs, PPIs and SGIs
  - Supported for physical and virtual interrupts
  - Resets to “not NMI”
- NMIs are the highest priority interrupts for owning Security state
  - Two new priority levels added into existing priority scheme →
- NMIs are not maskable by owning Security state
  - Non-secure state cannot mask NMIs
  - Secure state cannot mask Secure NMIs
    - Secure state can mask Non-secure NMIs and control when they pre-empt
- Separate acknowledge register for NMIs
  - `ICC_IAR` returns new reserved INTID if the highest priority interrupt is an NMI
  - `ICC_NMIAR` can only acknowledge NMIs



GIC interrupt priority space

© 2021 Arm

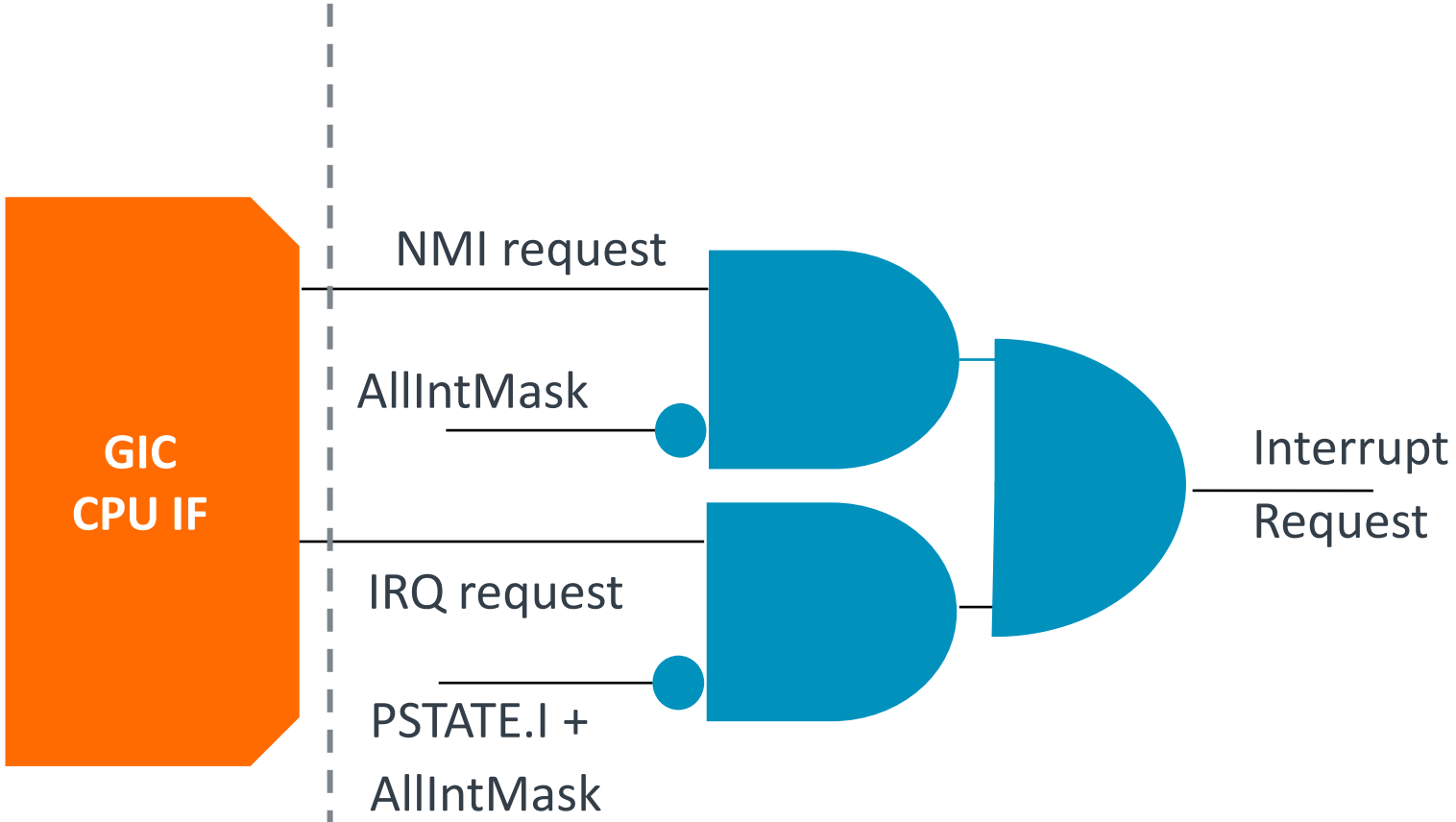
Masking with new PSTATE.AllInt



- NMI support controlled via `SCTLR_ELx`
- NMIs are not masked by `PSTATE.I/F`
- New `PSTATE.AllInt` masks affects all interrupts, including NMIs
  - New access instructions, which can be trapped to EL2
- Note: Masking rules due to routing controls still applies

© 2021 Arm

Alternative masking model: PSTATE.SP



AllIntMask = PSTATE.AllInt ||
(SCTLR_ELx.SPisINTMASK && PSTATE.SP)

- Alternative model uses the selected stack pointer as an implicit mask
- When the exception stack pointer is selected, NMIs are masked
  - This maps on to exception entry and exit, where interrupts must be masked to prevent corruption of state

© 2021 Arm

PMU and BRBE changes

FEAT_PMUv3_TH and FEAT_BRBEv1p1

© 2021 Arm

PMU and BRBE changes

Improving the developer experience on Arm

Cache events

- Cache hit events to more accurately report where data comes from
- Cache line state tracking to profile the accuracy of prefetching

PMU counter thresholds

- Some PMU events can increment by more than 1 per cycle
  - Number of FP operations
- New threshold feature to allow analysis by creating histogram profiles

BRBE at EL3 (Armv9 only)

- When introduced in 2020, BRBE was limited to EL2 and below
- BRBE now also supported at EL3

© 2021 Arm

Other additions in 2021

© 2021 Arm

Other items

- Hinted conditional branches
  - Hint that branch is very consistent in its behaviour
- EL3 trap on RNDR and RNDRRS
- New algorithm, QARMA3, for Pointer Authentication
  - Enables lower latency on some micro-architectures
- EL1 and EL2 traps on use of IMPDEF functionality at EL0
- Controls for EL0 cache maintenance operations
  - In base architecture, I cache invalidate and D cache clean only require read permission at EL0
  - New mode introduced which requires write-permission

© 2021 Arm

arm

Find out more

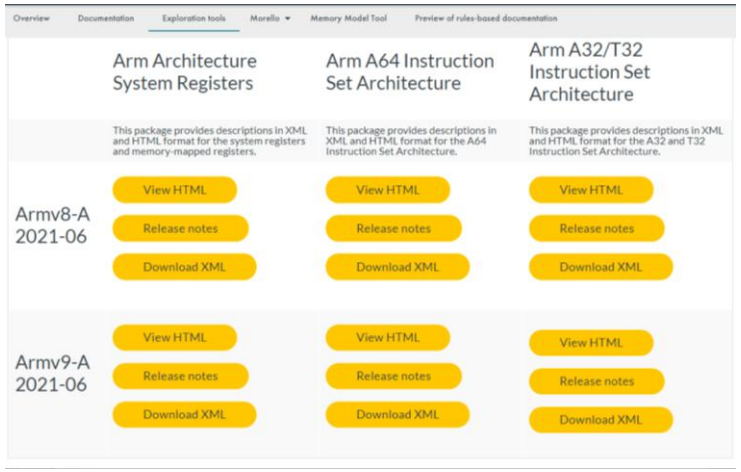
© 2021 Arm

Find out more

Architecture documentation

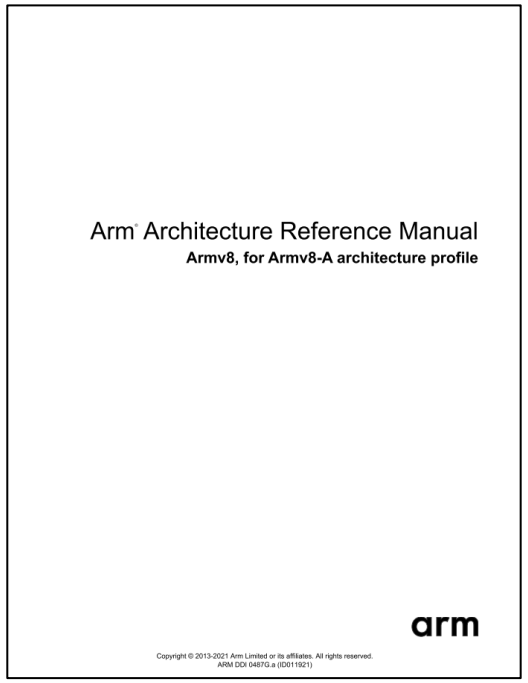
Register and ISA XML

- Available end of September



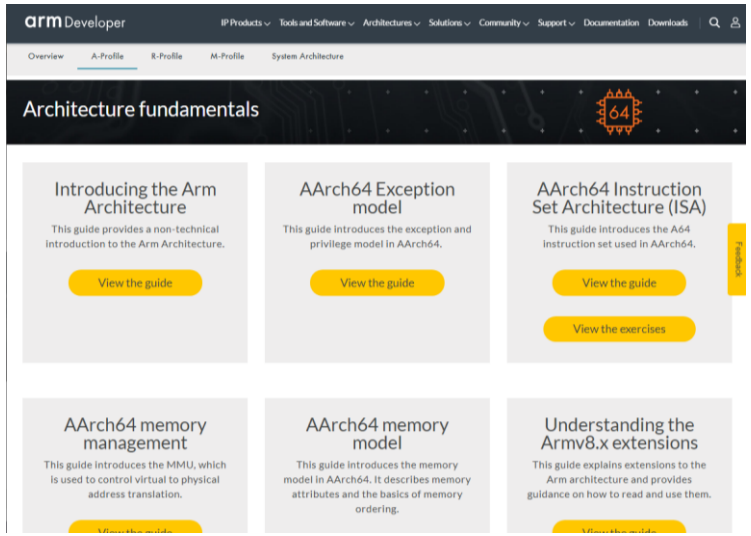
Architecture Reference Manual

- Available early 2022



Learn the Architecture

- Updates coming later this year



© 2021 Arm

One more thing… rules-based writing

https://developer.arm.com/architectures/cpu-architecture/a-profile/preview-of-rules-based-documentation

- The architecture documentation is moving to a rules-based writing style
- RME and SVE/SVE2 supplements are already written in this style
- For the existing material we are releasing preview versions of the re-written chapters on developer.arm.com
  - Currently AArch64 exception model chapter is available

Chapter B1. Exception levels
B1.1. Execution state

B1.1 Execution state

The architecture defines two *Execution states*, *AArch32 state* and *AArch64 state*.

R_{HTYJH} The architecture defines two *Execution states*. The AArch32 *Execution state* uses 32-bit processing. The AArch64 *Execution state* uses 64-bit processing.

I_{WKKGD} AArch32 *Execution state* is compatible with Armv7-A operation.

R_{NKNVR} An *Exception level* is described as:

- "Using AArch64" when execution in that *Exception level* is in the AArch64 *Execution state*.
- "Using AArch32" when execution in that *Exception level* is in the AArch32 *Execution state*.

R_{VZTWD} The *PE* can only change Execution state at Reset, or when changing *Exception level*.

R_{QNVJL} If an *Exception level* is using AArch32, then all lower *Exception levels* are using AArch32.

R_{BDMRT} If an *Exception level* is using AArch64, then all higher *Exception levels* are using AArch64.

R_{MWGKL} Interaction between the AArch64 and AArch32 Execution states is called "Interprocessing".

I_{KSMPY} In this chapter, the described behavior assumes the highest implemented *Exception level* is using AArch64.

© 2021 Arm

arm

Martin Weidmann

Director Product Management

martin.weidmann@arm.com

Thank You
Danke
Gracias
谢谢
ありがとう
Asante
Merci
감사합니다
धन्यवाद
Kiitos
شكرًا
ধন্যবাদ
תודה

© 2021 Arm

The Arm trademarks featured in this presentation are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere. All rights reserved. All other marks featured may be trademarks of their respective owners.

www.arm.com/company/policies/trademarks

© 2021 Arm