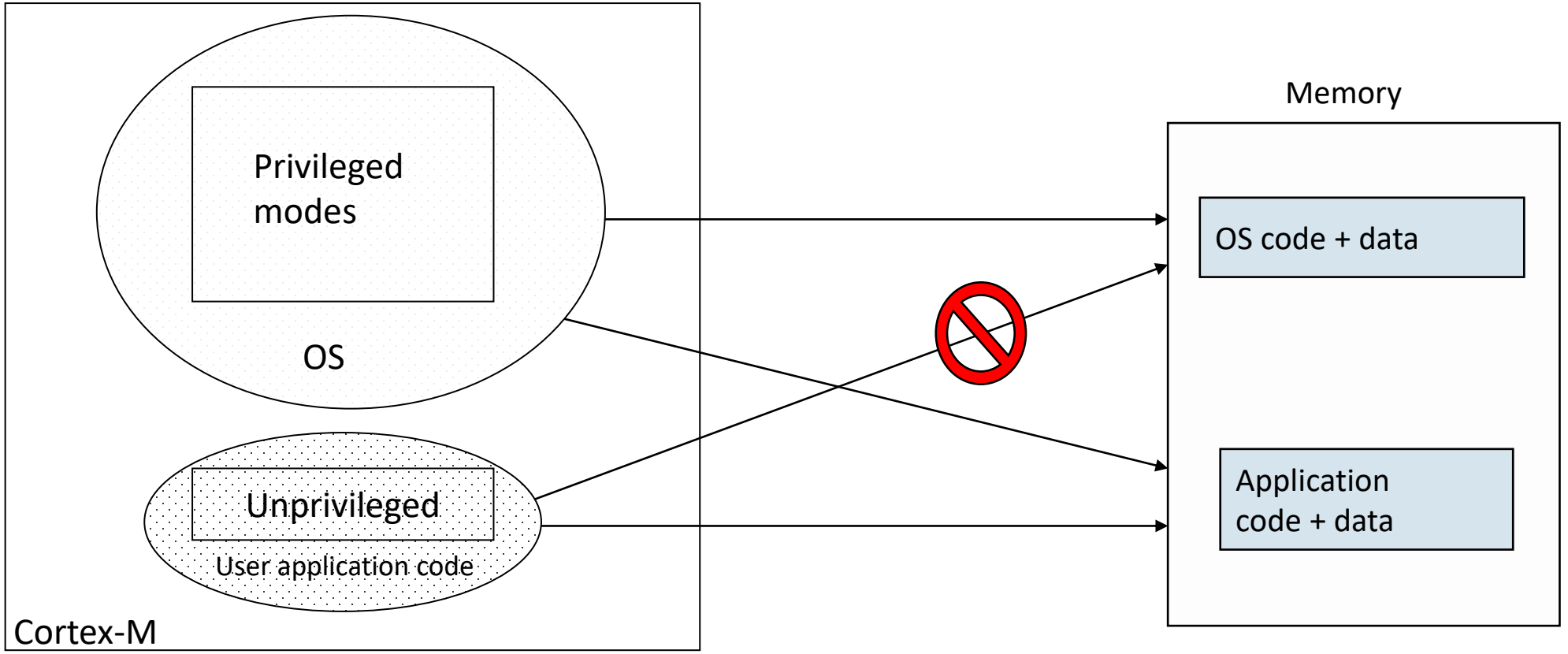
Armv8-M Mainline Memory Protection

Learning objectives

At the end of this module you will be able to:

- Describe the need for a Memory Protection Unit (MPU)
- Understand how the Armv8-M MPU differs to previous Armv7-M and Arm6-M MPUs
- Describe the different memory-mapped MPU registers
- Configure memory regions by programming the MPU registers

Motivation: memory protection



Memory protection controls accesses to the address space

- This is needed due to the various security and privilege states that the core can operate in and the importance of restricting sensitive data to certain states

Memory protection and security attribution

Memory protection consists of:

- An optional Memory Protection Unit (MPU)
  - Based on the Protected Memory System Architecture (PMSAv8)
- An optional Security Attribution Unit (SAU)
  - Available if the Security Extension is implemented

The number of supported regions is implementation defined

- Each region applies to both instruction and data accesses (unified regions)

The MPU provides full support for:

- Protection regions, access permissions and exports memory attributes to the system

MPU mismatches and permission violations invoke the MemManage handler if it is enabled

- Otherwise HardFault

The SAU is a programmable unit that determines the security of an address

- The SAU is not covered in this module

If no protection units are implemented, the processor uses the default memory map

Default system address map

By default, the address space defines eight 0.5GB regions

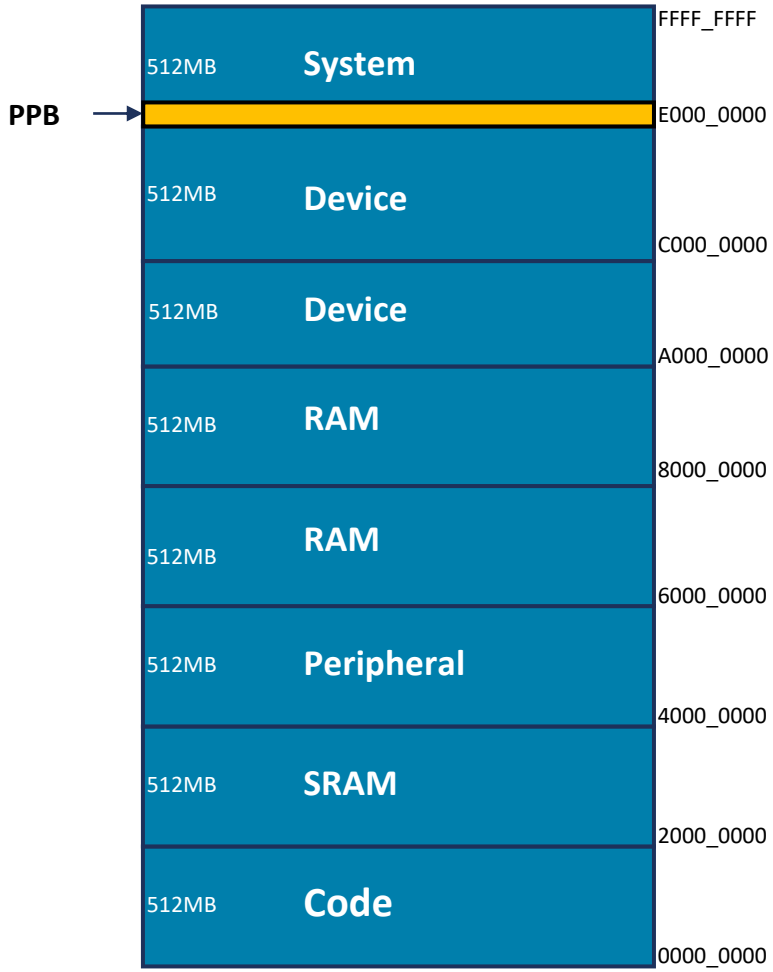
The default system address map is applied when:

The Memory Protection Unit is not implemented

The MPU is implemented but is disabled

The MPU is implemented and enabled but the system

- Accesses the PPB address space
- Reads the vector table on exception entry
- Performs a privileged access to a region that is not enabled, and MPU_CTRL.PRIVDEFENA is set
- Executes an NMI or HardFault handler with MPU_CTRL.HFNMIENA=0



Memory Protection Unit (1)

A Memory Protection Unit (MPU) provides basic memory management

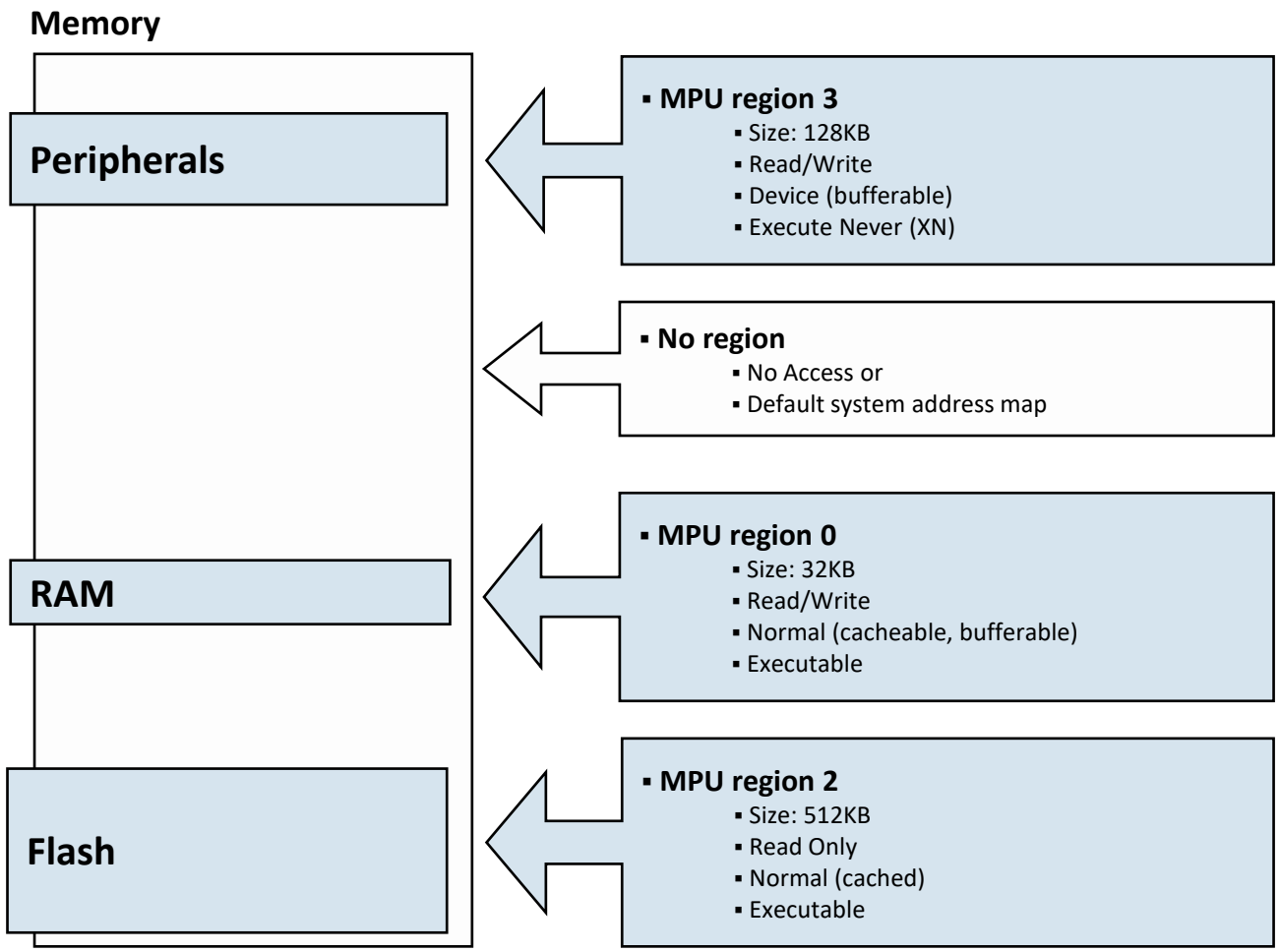
- Allows attributes to be applied to different address regions
- All accesses checked against MPU regions

Each region consists of

- Base address
- Limit address
- Attributes (e.g. type, size, access permissions)

Available on

- Cortex-M23, Cortex-M33, Cortex-M35P, Cortex-M55



Memory Protection Unit (2)

The MPU can be used to allow privileged software to control access to physical memory

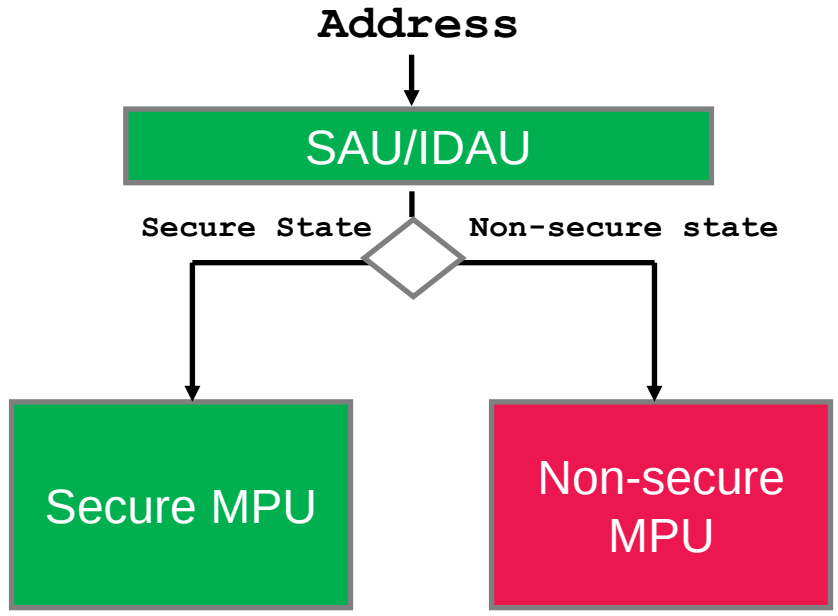
If implemented the MPU can

- Partition Physical Memory into configurable regions
- Change memory region type and attributes*
- Change peripheral and device regions to executable*
- Define region memory access permissions
- Checks instruction and data accesses to memory
- Trigger a fault/exception when access violations occur

When the Security Extension is implemented

- The MPU doesn't perform any security checks
- There are separate MPUs for secure and non-secure worlds which can be programmed independently

** There are exceptions to these, which will be discussed in later slides*



Armv8-M MPU compatibility with Armv7-M and Armv6-M

Protected Memory System Architecture (PMSAv8) adopts base and limit style comparators for regions

- Improved usability and flexibility compared to PMSAv7 and PMSAv6

More flexible MPU programming model

- Replaces previous power-of-two size, sized aligned scheme
- Accelerates programming, potentially reducing context switch times

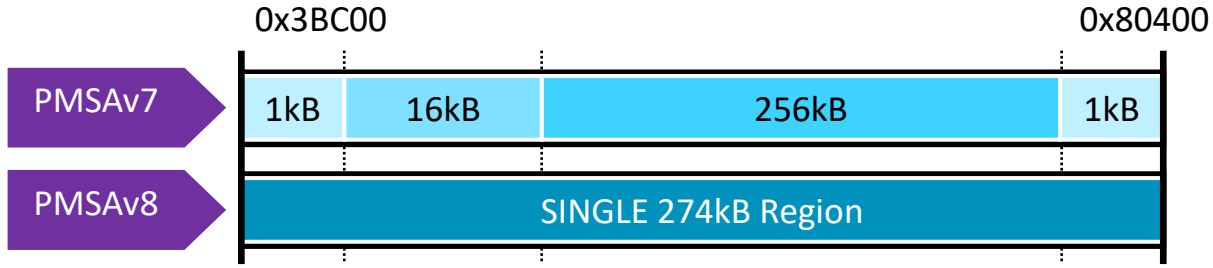
Overlapping regions produce MemManage Fault

- Armv7-M supported overlapping regions to overcome limitations of rigid MPU programming model

No support for Armv7-M subregions

- Flexibility in region addresses largely dispenses with its need

The smallest region size in PMSAv8 is 32 bytes



Memory regions overview (1)

Armv8-M Baseline/Mainline provide a default memory map identical to Armv6-M/Armv7-M, with:

- Device-nGnRE substituted for “Device” and Device-nGnRnE substituted for “Strongly-ordered”

Accesses to the PPB always uses the default system memory map

Exception vector reads from the Vector Address Table always use the default system memory map

The default system memory map can be configured to provide a background region for privileged accesses

Memory regions overview (2)

The architecture restricts how the MPU can change the default system memory map attributes for regions in System space, that is, for addresses 0xE0000000 and higher

- System space is always XN
- The MPU can map System space regions that default to Device-nGnRE to Device-nGnRnE
- The effect of remapping a System space region that defaults to Device memory as Normal memory is UNPREDICTABLE.

A memory access with the MPU enabled generates a precise fault if:

- It is to an address that matches in more than one region
- It is to an address which does not match any region and if the background region is not enabled
- It does not match all access conditions for the region in which the address matches

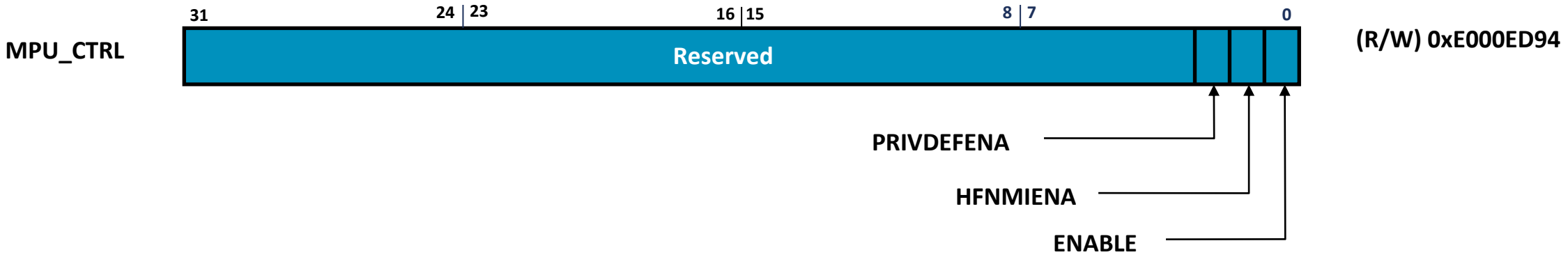
MPU Registers

Address	Name	Type	Description
0xE000ED90	MPU_TYPE	RO	MPU Type Register
0xE000ED94	MPU_CTRL	RW	MPU Control Register
0xE000ED98	MPU_RNR	RW	MPU Region Number Register
0xE000ED9C	MPU_RBAR	RW	MPU Region Base Address Register
0xE000EDA0	MPU_RLAR	RW	MPU Region Limit Address Register
0xE000EDA4	MPU_RBAR_A1	RW	MPU Region Base Address Register Alias 1
0xE000EDA8	MPU_RLAR_A1	RW	MPU Region Limit Address Register Alias 1
0xE000EDAC	MPU_RBAR_A2	RW	MPU Region Base Address Register Alias 2
0xE000EDB0	MPU_RLAR_A2	RW	MPU Region Limit Address Register Alias 2
0xE000EDB4	MPU_RBAR_A3	RW	MPU Region Base Address Register Alias 3
0xE000EDB8	MPU_RLAR_A3	RW	MPU Region Limit Address Register Alias 3
0xE000EDC0	MPU_MAIR0	RW	MPU Memory Attribute Indirection Register 0
0xE000EDC4	MPU_MAIR1	RW	MPU Memory Attribute Indirection Register 1

MPU_RBAR/RLAR Alias Registers

- Multiple regions can be accessed via a single **`STM`** or **`memcpy()`**

MPU Control Register – MPU_CTRL



PRIVDEFENA [2]– Privileged background region enable:

- 0: All accesses to background regions result in fault.
- 1: Allows privileged accesses to use the default memory map as a background region when the MPU is enabled

If set and no regions are enabled, then only privileged code can execute

HFNMIENA [1] – MPU Enable for HardFault and NMI, controls whether handlers executing with priority less than 0, access memory with the MPU enabled or with the MPU disabled

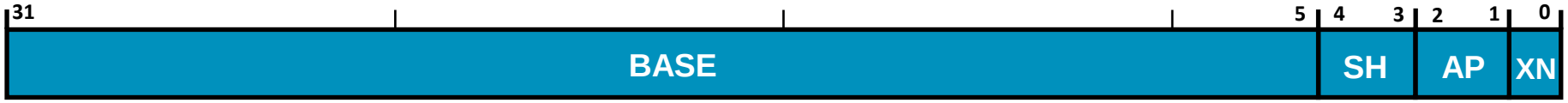
- 0: The MPU is disabled for these handlers and the default memory map is used
- 1: The MPU is enabled for these handlers

ENABLE [0] – Enables / Disable the MPU

- 1: MPU Enabled
- 0: MPU Disabled

MPU Region Base Address Register

MPU_RBAR (R/W) 0xE000ED9C



Shareability SH[4:3]	Normal memory
00	Non-shareable
01	Reserved
10	Outer Shareable
11	Inner Shareable

Access Permissions AP[2:1]	Access
00	Read/write, privileged code only
01	Read/write, any privilege level
10	Read-only, privileged code only
11	Read-only, any privilege level

Execute Never XN	Executable
0	Yes
1	No

BASE: Contains bits [31:5] of the lower inclusive limit of the selected MPU memory region

Bits [4:0] of the address are treated as 5’b00000

MPU Region Limit Address Register

LIMIT, bits[31:5]

- The upper inclusive limit of the selected MPU memory region
- The actual Limit address bits [4:0] are treated as 5’b11111 = 0x1F

AttrIndx, bits[3:1]

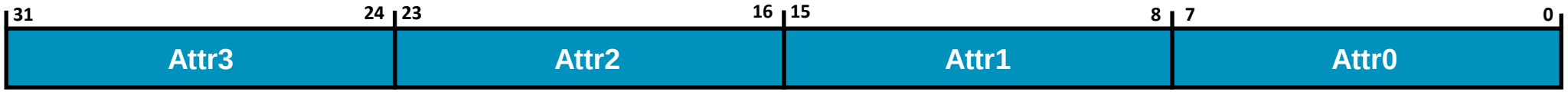
- Associates a set of attributes in the MPU_MAIR0/1 fields

EN, bit[0]

- Region enable

MPU Memory Attribute Indirection Register 0/1

MPU_MAIR0/1 **(R/W) 0xE000EDC0/4**



NORMAL MEMORY

(when Attr<n>[7:4] != 0000)

Attr<n>[7:4]	Attributes
0000	See Device memory
00RW *	Outer Write-through transient
0100	Outer Non-Cacheable
01RW *	Outer Write-back transient
10RW	Outer Write-through non-transient
11RW	Outer Write-back non-transient

Attr<n>[3:0]	Attributes
0000	UNPREDICTABLE
00RW *	Inner Write-through transient
0100	Inner Non-Cacheable
01RW *	Inner Write-back transient
10RW	Inner Write-through non-transient
11RW	Inner Write-back non-transient

DEVICE MEMORY

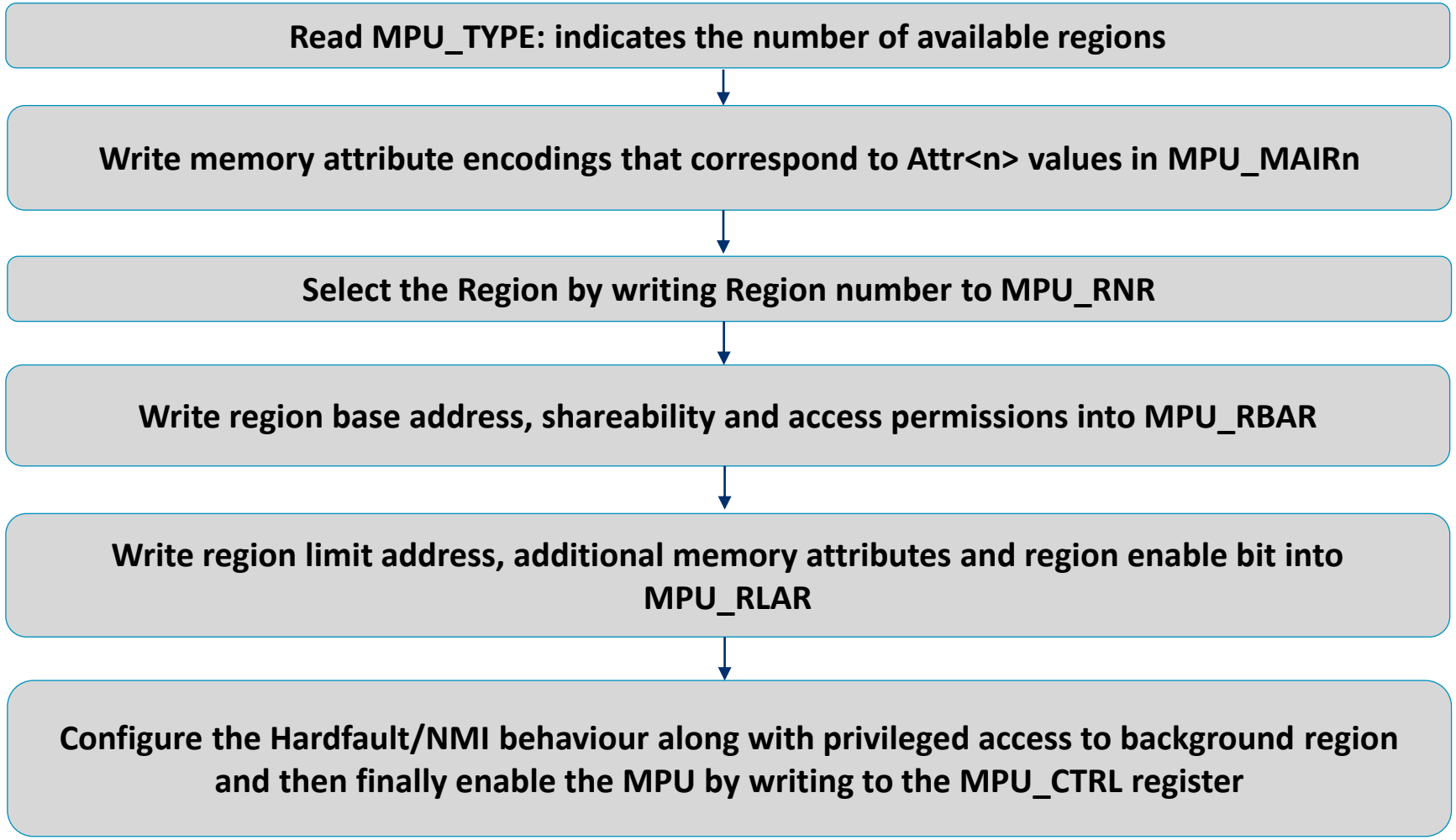
(when Attr<n>[7:4] == 0000)

Attr<n>[7:0]	Attributes
00000000	Device-nGnRnE memory
00000100	Device-nGnRE memory
00001000	Device-nGRE memory
00001100	Device-GRE memory
0000XXRW	UNPREDICTABLE (when RW != 00)

R/W	Meaning
0	Do Not Allocate
1	Allocate

* ReadWrite Cache line Allocation hint bits (RW) != 00

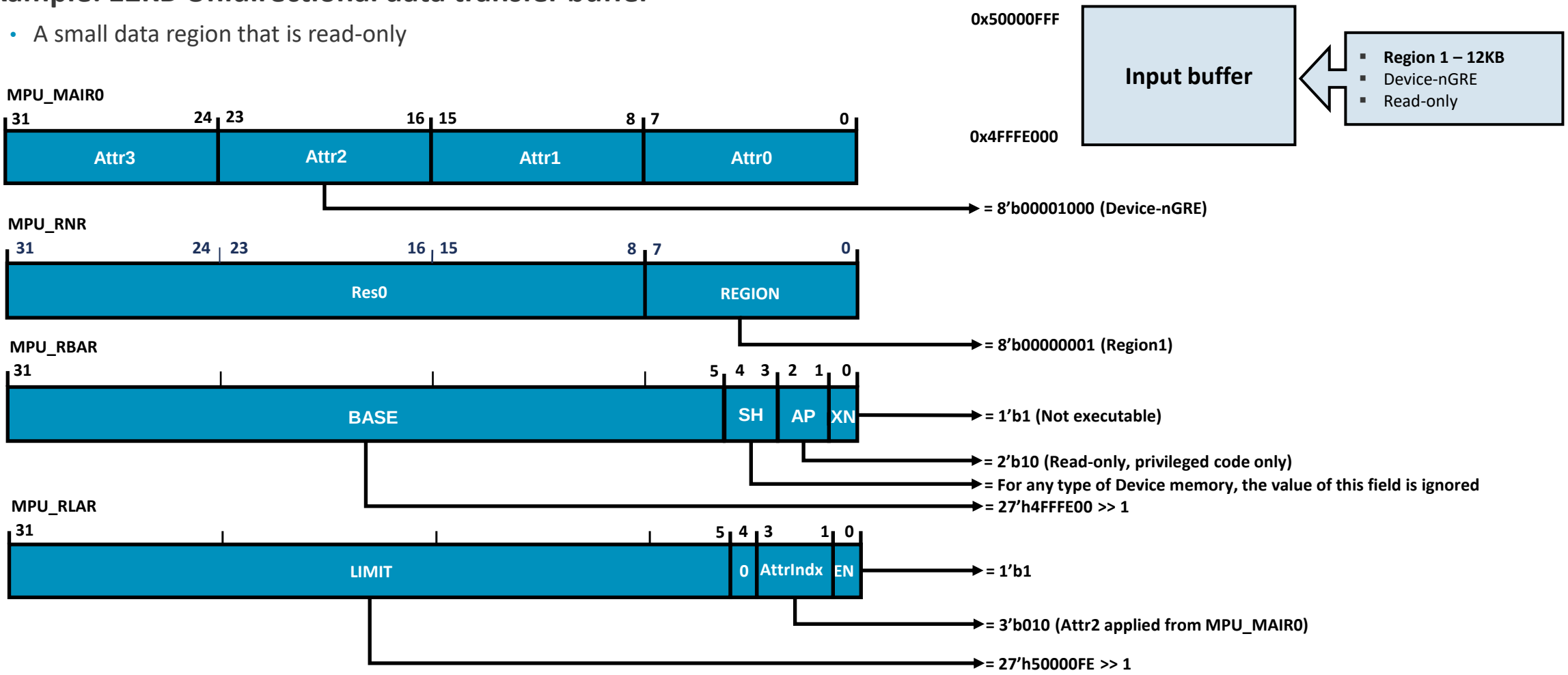
Configuring the MPU



Region programming

Example: 12KB Unidirectional data transfer buffer

- A small data region that is read-only



MemManage faults (Armv8-M Mainline only)

Type	MMFSR status bit	DEMCR vector catch bit	Conditions
Data access	DACCVIOL	VC_MMERR	Violation or fault on MPU as result of data access
Instruction access	IACCVIOL	VC_MMERR	Violation or fault on MPU as result of instruction address
Exception entry stack memory operations	MSTKERR	VC_INTERR	Failure on a hardware save of context, because of an MPU access violation. The processor does not update the MMAR
Exception return stack memory operations	MUNSTKERR	VC_INTERR	Failure on a hardware restore of context, because of an MPU access violation. The processor does not update the MMAR
Lazy state preservation error flag	MLSPERR	VC_INTERR	Records whether a MemManage fault occurred during FP lazy state preservation

Like Armv7-M, an implementation with the Main Extension provides:

- MemManage fault (all MPU faults are precise)
- MMFSR – MemManage Fault Status Register
- MMFAR – MemManage Fault Address Register
- DEMCR – Debug Exception and Monitor Control Register (contains MemManage vector catch fields)

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