

TRUSTZONE TECHNOLOGY

NOVEMBER, 2018



EXTERNAL USE



SECURE CONNECTIONS
FOR A SMARTER WORLD

Content

- TrustZone Technology Overview
- Memory Configuration
- Switching between Secure and Non-secure
- Exceptions

TRUSTZONE TECHNOLOGY OVERVIEW



IoT Will Be Everywhere

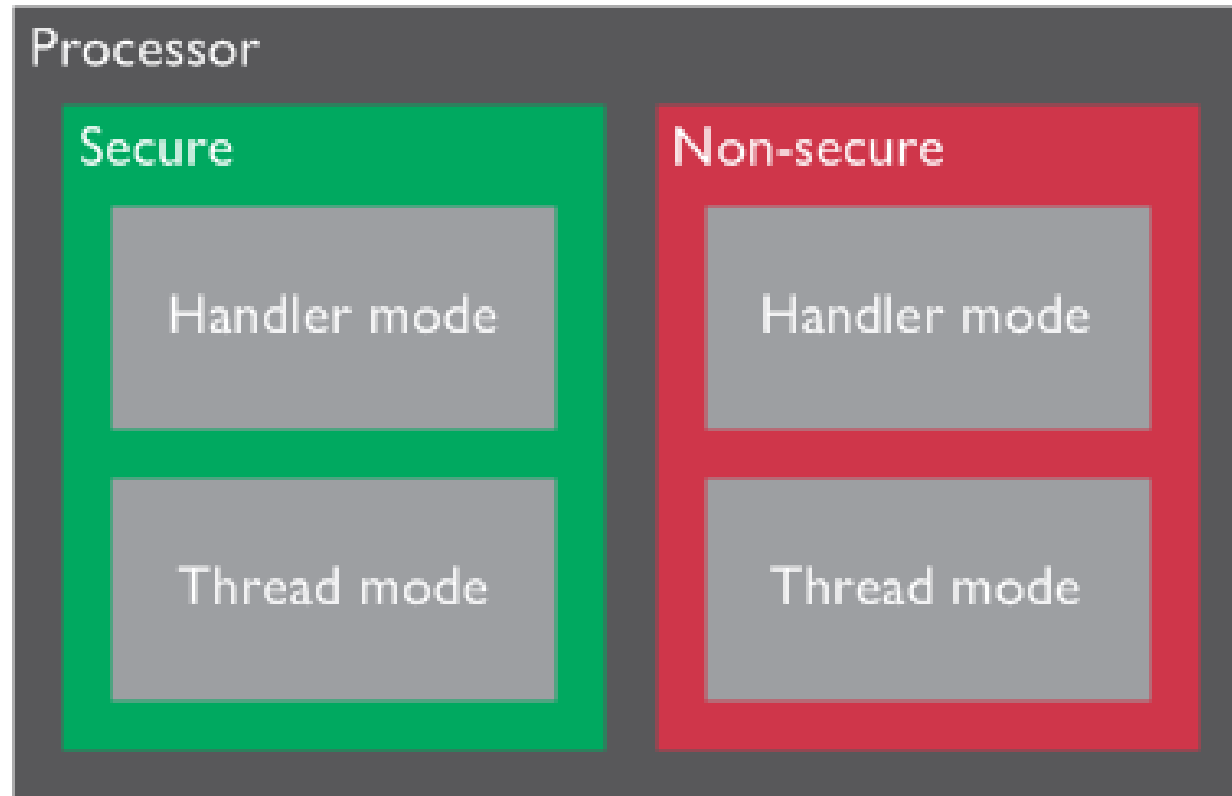
- In recent years, the Internet of Things (IoT) has become a hot topic for embedded system developers.



- IoT system products have become more complex, and better solutions are needed to ensure system security.
- **Traditional solution:** By dividing software into **privileged** and **non-privileged parts**, privileged software uses MPU to prevent unprivileged applications from accessing critical system resources, including security-sensitive information.

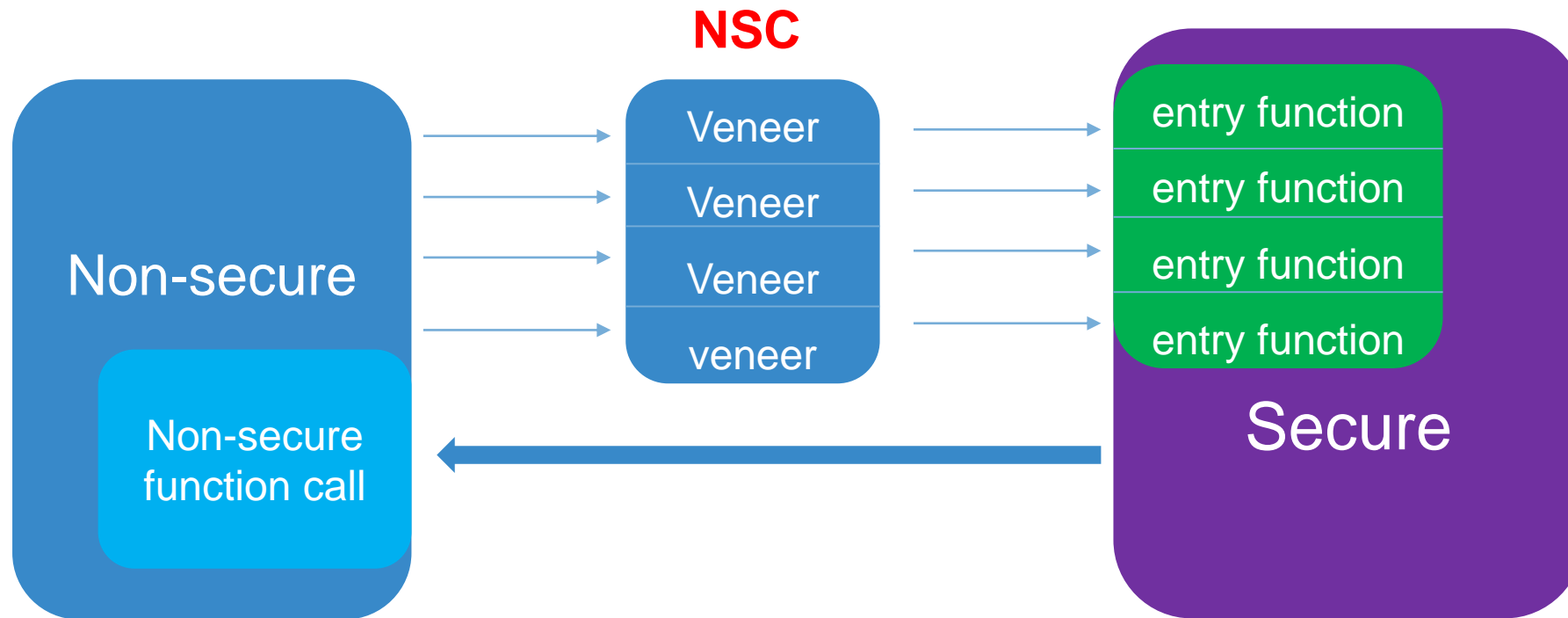
ARM TrustZone technology

- TrustZone technology for ARMv8-M is an optional Security Extension that is designed to provide a foundation for improved system security in a wide range of embedded applications.
- TrustZone technology divides the system into two states, **safe and non-secure**, and can switch between the two states through corresponding commands.



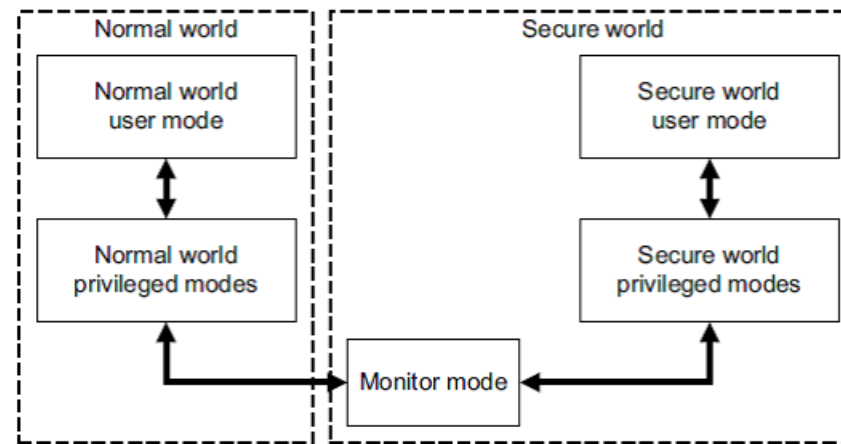
Switching between Secure and Non-secure

- The Secure memory space is further divided into two types:
 - **Secure and Non-secure Callable(NSC)**



Cortex-M and Cortex-A TrustZone technology comparison

- In both designs, the processor has **Secure and Non-secure states**.
- There are several differences in the implementation:
 - TrustZone technology for ARMv8-M supports **multiple Secure function entry points**, whereas in TrustZone technology for Cortex-A processors, **the Secure Monitor handler** is the sole entry point.
 - Non-secure interrupts can still be serviced when executing a Secure function.



Cortex-A

Features of TrustZone technology


- Allows user to divide memory map into **Secure and Non-Secure regions**
- Allows **debug to be blocked for Secure code/data** when not authenticated
- CPU includes Security Attribution Unit (**SAU**) as well as a duplication of **NVIC, MPU, SYSTICK, core control registers** etc. such that Secure/Non-Secure codes can have access to their own allocated resources
- Stack management expands from **two stack pointers in original Cortex-M** (Main Stack Pointer (MSP) and Process Stack Pointer (PSP)) **to four**, providing the above pair individually to both Secure and Non-Secure
- Introduces the concept of **Secure Gateway** opcode to allow secure code to define a strict set of entry points into it from Non-secure code.

Stack limit checking

- As part of ARM TrustZone technology for ARMv8-M, there is also a **stack limit checking feature**. For **ARMv8-M Mainline**, all stack pointers have corresponding stack limit registers.

```
101
102; Reset Handler
103
104Reset_Handler  PROC
105                EXPORT Reset_Handler            [WEAK]
106                IMPORT SystemInit
107                IMPORT __main
108
109                LDR    R0, =__stack_limit
110                MSR   MSPLIM, R0
111
112                LDR    R0, =SystemInit
113                BLX   R0
114                LDR    R0, =__main
115                BX    R0
116                ENDP
117
```

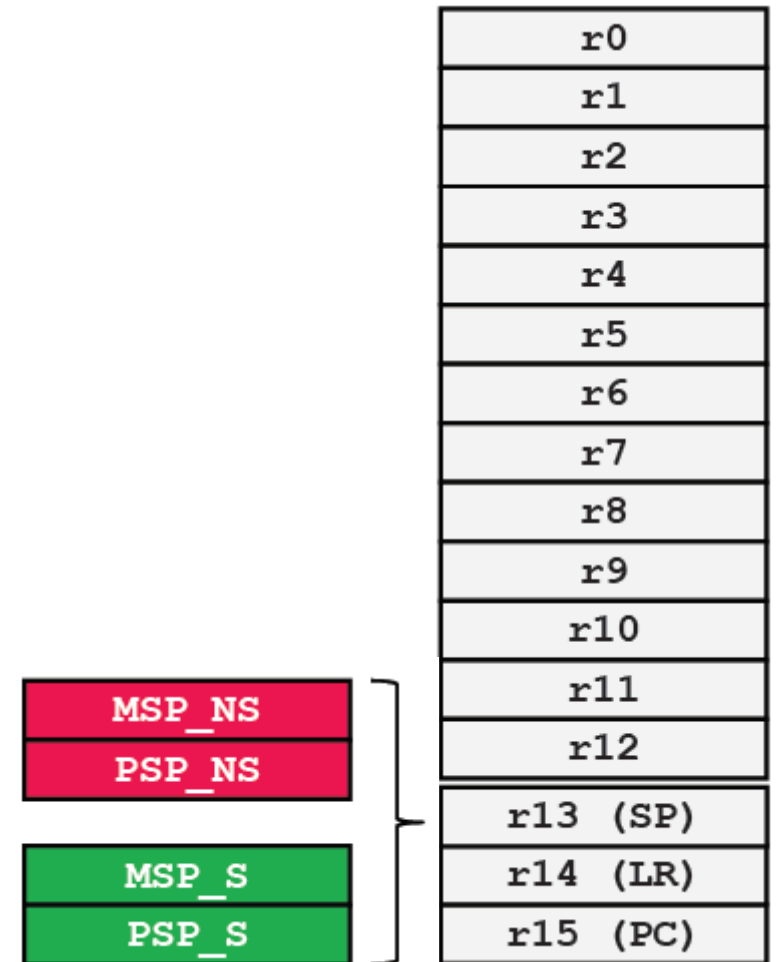
; Non-secure version of MSPLIM is RAZ/WI



The address at the bottom of the stack is stored in the MSPLIM register

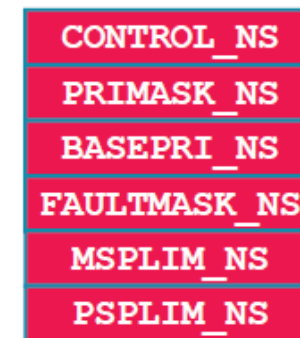
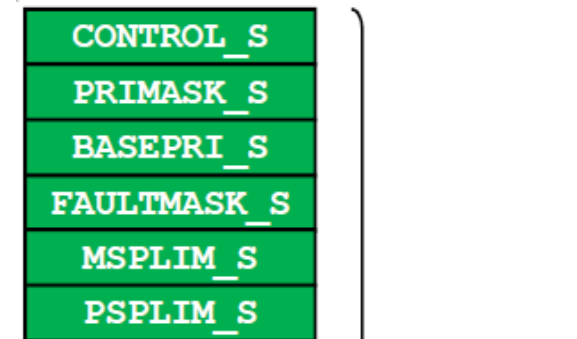
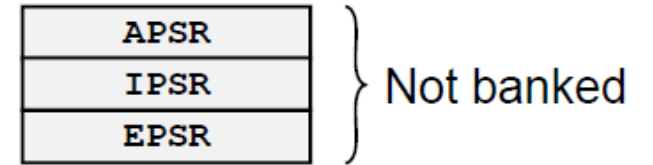
General-purpose register banking

- Most general-purpose registers are common to both security states
 - Registers R0-R7
 - Accessible to all instructions
 - Registers R8-R12
 - Accessible to a few 16-bit instructions
 - Accessible to all 32-bit instructions
 - R14 is the link register(LR)
 - R15 is the program counter(PC)
- R13 is **the stack pointer(SP)**
 - Banked by security state
- Floating-point register D0-D15 are not banked
- CONTROL and some other special-purpose registers are also banked by security...



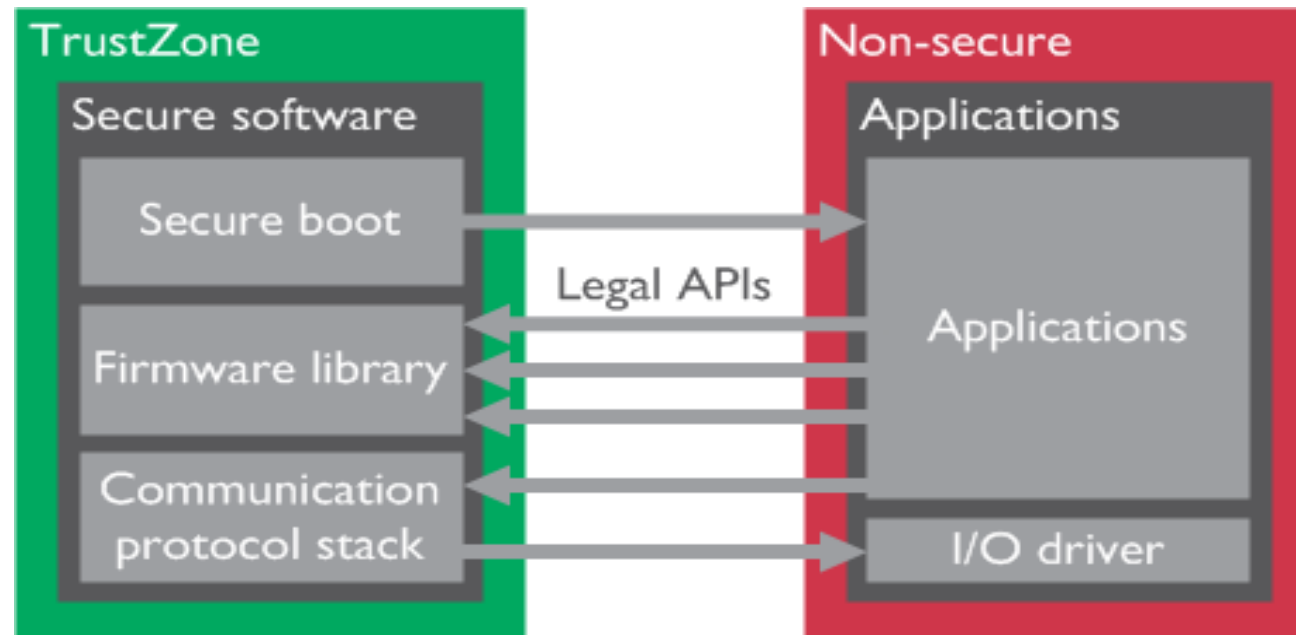
Special-purpose register banking

- Special-purpose registers are accessed using special instructions
 - MSR/MRS/CPS
- Some registers are security banked
- Non-secure code can only access Non-secure registers
- Secure code can access Secure and Non-secure instances



Security requirements addressed by TrustZone technology

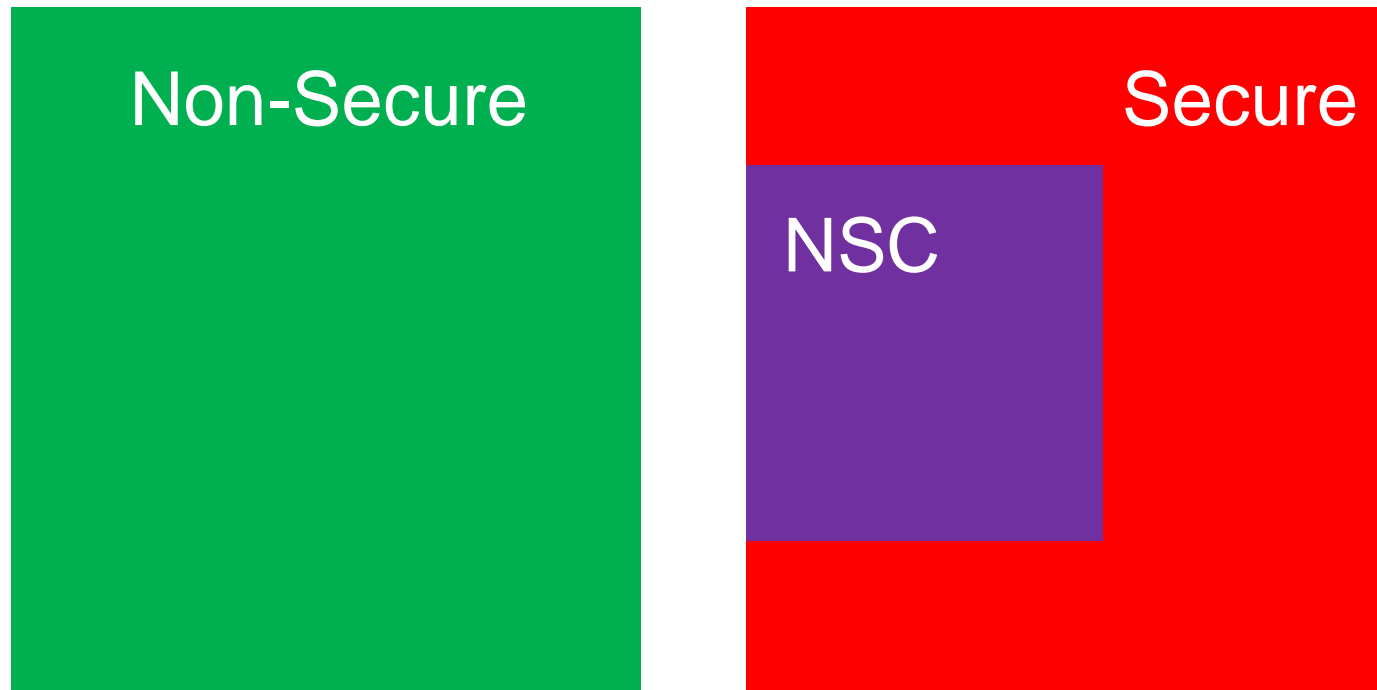
- **Data protection**
- **Firmware protection**
- **Operation protection**
- **Secure boot**



MEMORY CONFIGURATION

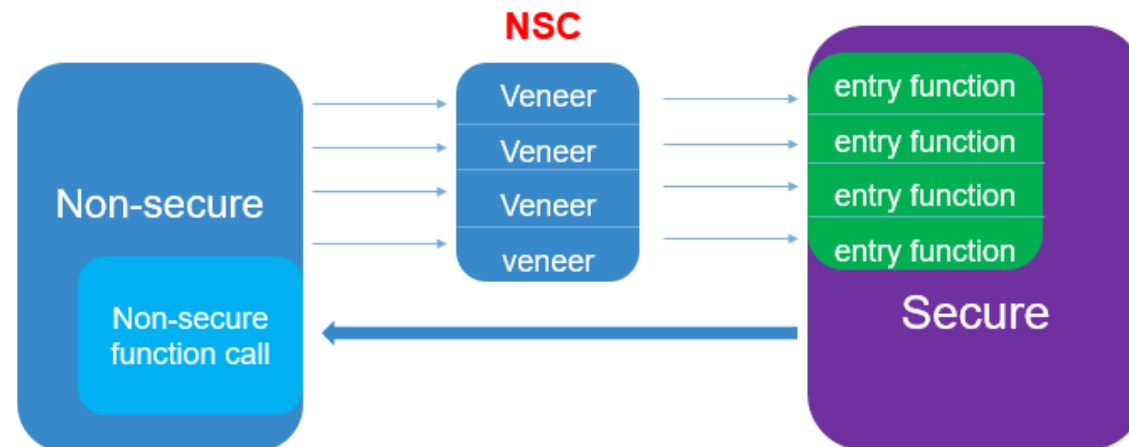
Memory system and memory partitioning

- If the **Security Extension** is implemented the 4GB memory space is partitioned into **Secure and Non-secure** memory regions.
- The Secure memory space is further divided into two types:
 - **Secure and Non-secure Callable(NSC)**



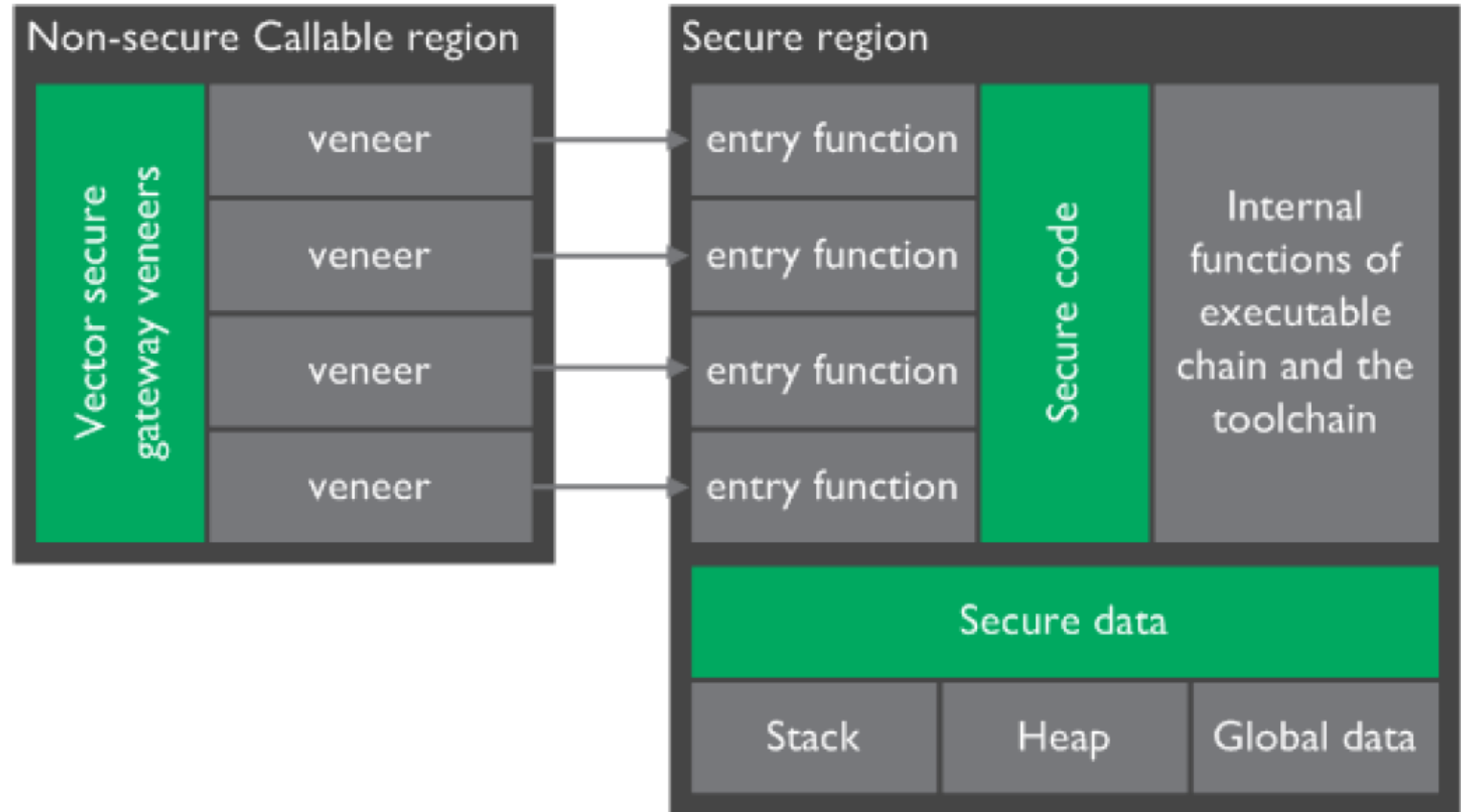
Trustzone Memory Regions : Secure/Non-Secure/Non-Secure Callable

- **Secure (S)** - For Secure code/data
 - Secure data can only be read by secure code
 - Secure code can only be executed by CPU in secure mode
- **Non-Secure (NS)** – For non-Secure code/data
 - NS Data can be accessed by both secure state and non-secure state CPU
 - **Cannot be executed by Secure code**
- **Non-Secure Callable (NSC)**
 - This is a special region for NS code to branch into and execute a Secure Gateway (**SG**) opcode.

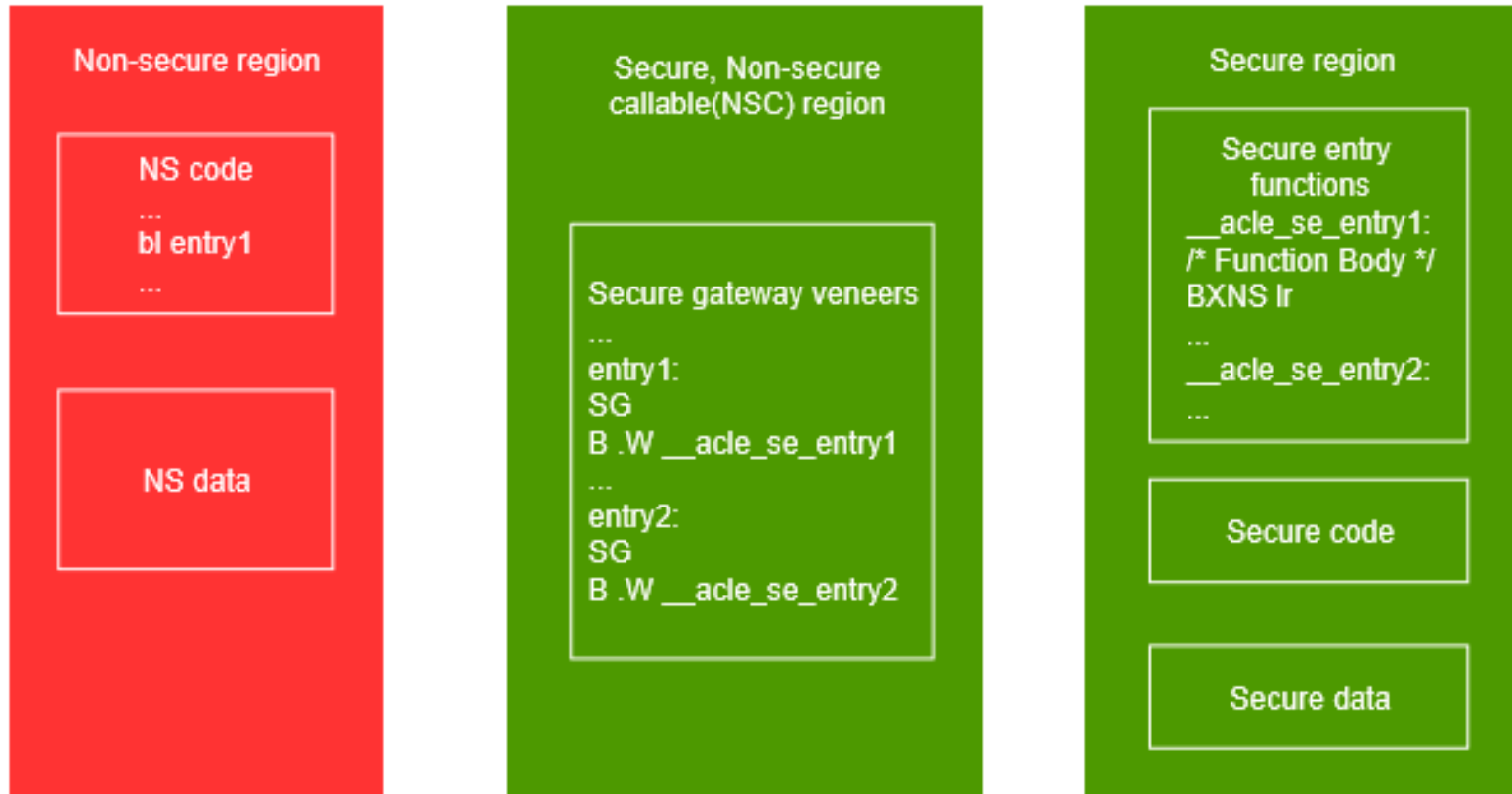


Non-Secure Callable (NSC) Memory

- Certain portion of Secure memory should be marked as **Non-Secure Callable (NSC)** memory for cross-domain calls.
- NSC memory regions contain tables of small **branch veneers** (entry points).
 - The first instruction in API must be **SG instruction**
 - NSC memory is to prevent hackers to use binary data matching SG opcode value

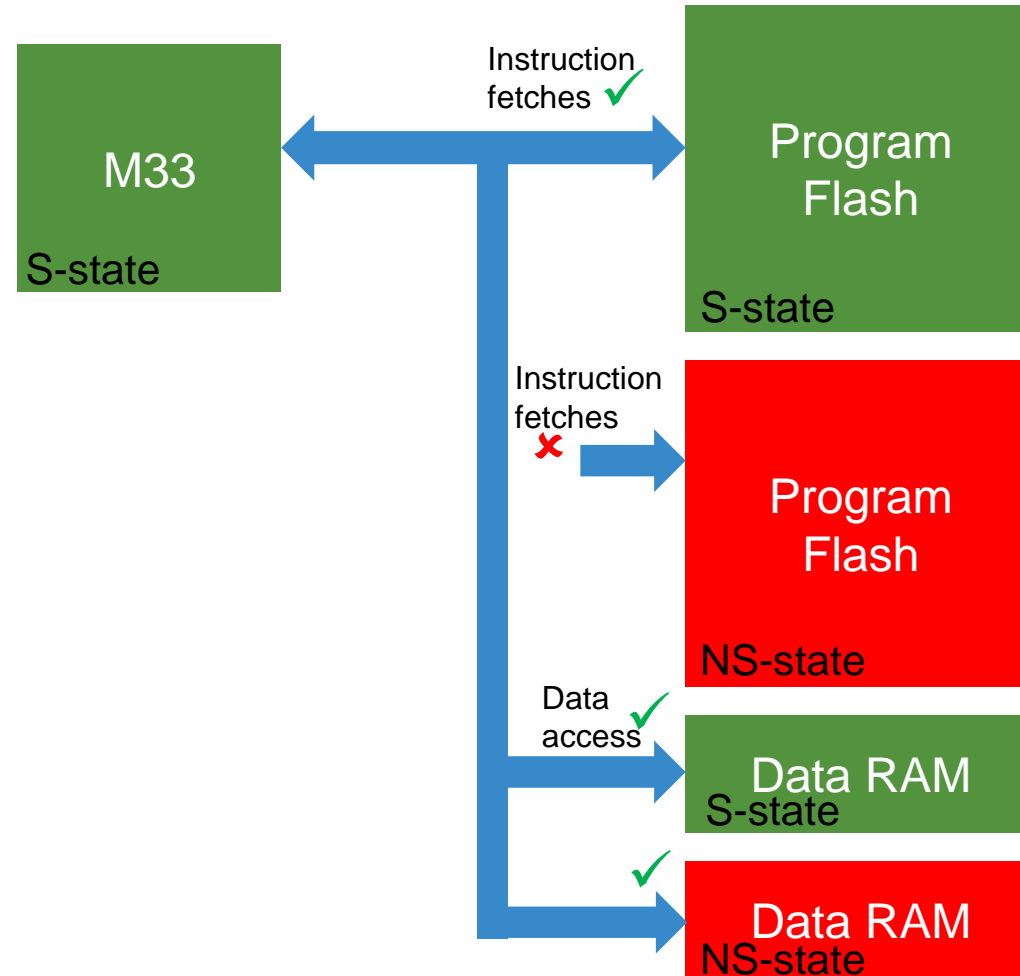


Secure gateway veneers



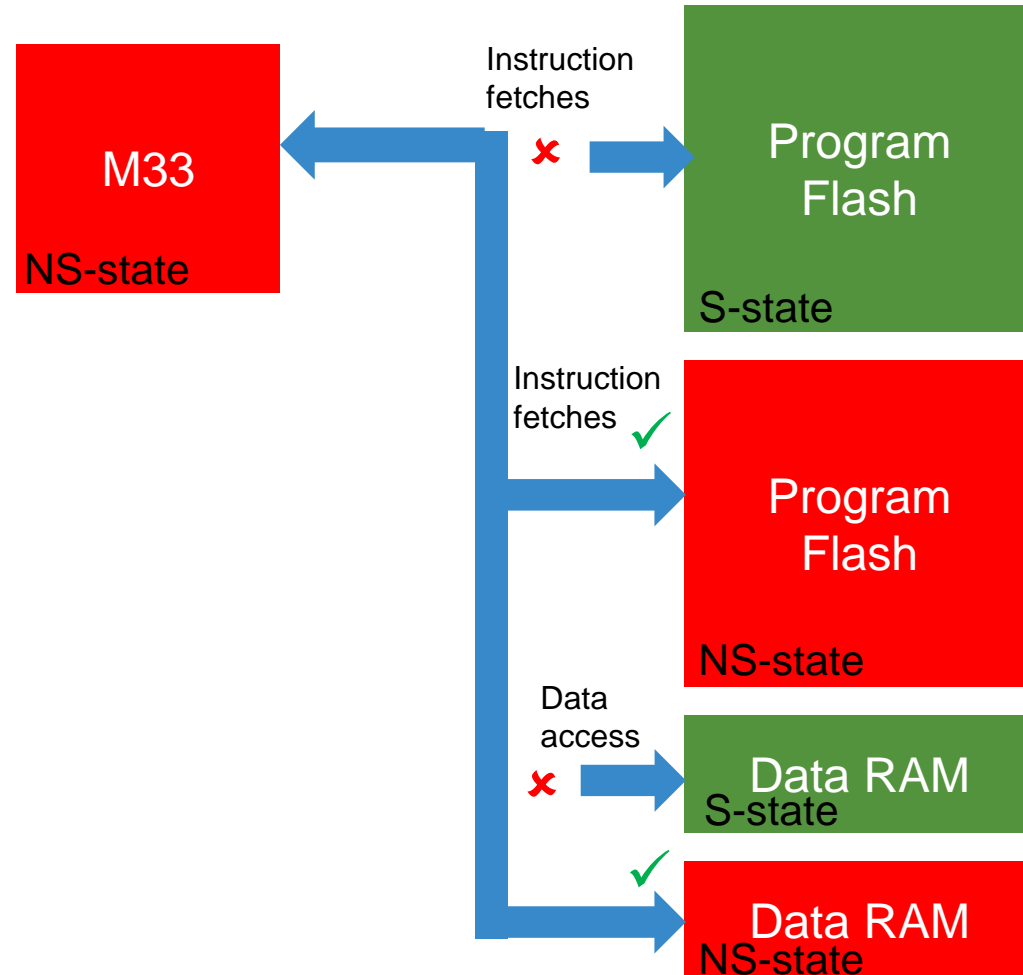
Secure CPU State

- CPU in secure state can only execute from Secure Program memory.
- CPU in secure state can access data from both secure and NS memory.



Non-Secure CPU State

- CPU in non-secure state can only execute from non-secure program memory.
- CPU in non-secure state can access data from NS memory only.



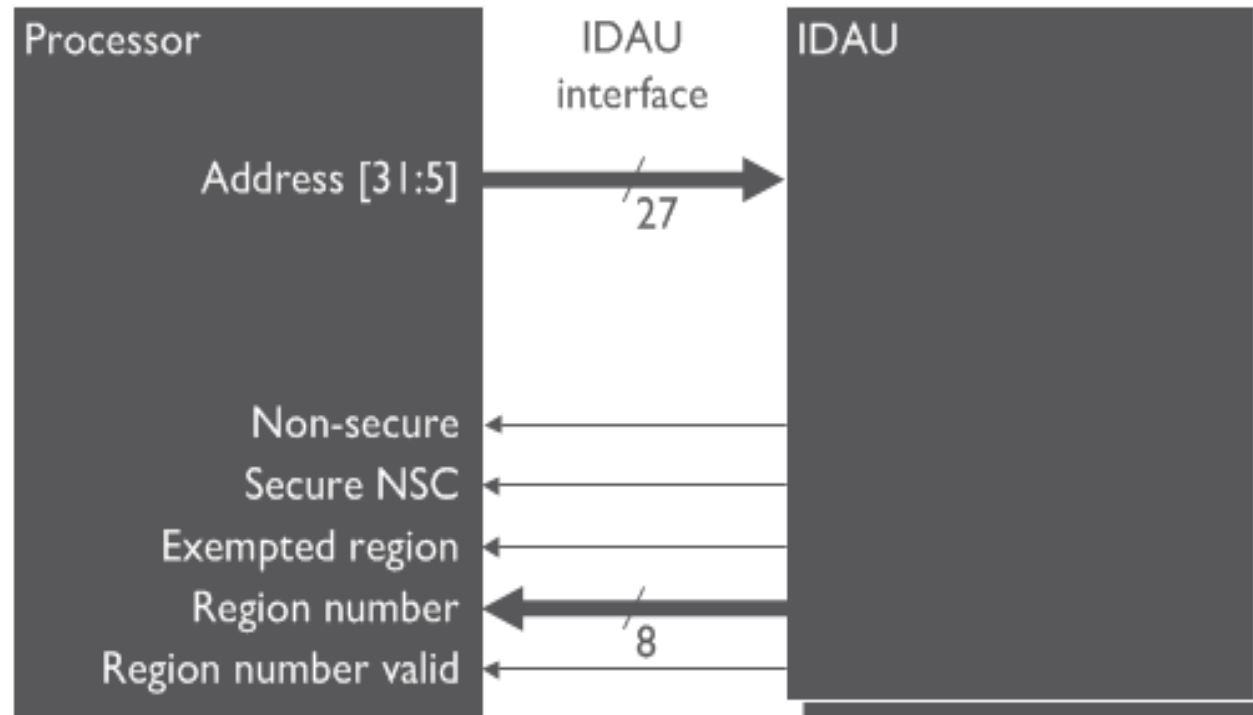
Memory security determination

- The security state of a memory region is controlled by a combination of the **internal *Secure Attribution Unit (SAU)*** or an **external *Implementation Defined Attribution Unit (IDAU)***.

SAU		IDAU		End result
Secure	+	Secure	=	Secure
NS	+	Secure	=	Secure
Secure	+	NS	=	Secure
NS	+	NS	=	NS
NSC	+	Secure	=	Secure
NSC	+	NS	=	NSC

IDAU

- The IDAU is used to indicate to the processor if a particular memory address is Secure, Non-secure Callable (NSC), or Non-secure, and provides the region number within which the memory address resides. It can also mark a memory region to be **exempted from security checking**, for example, a ROM table.



IDAU

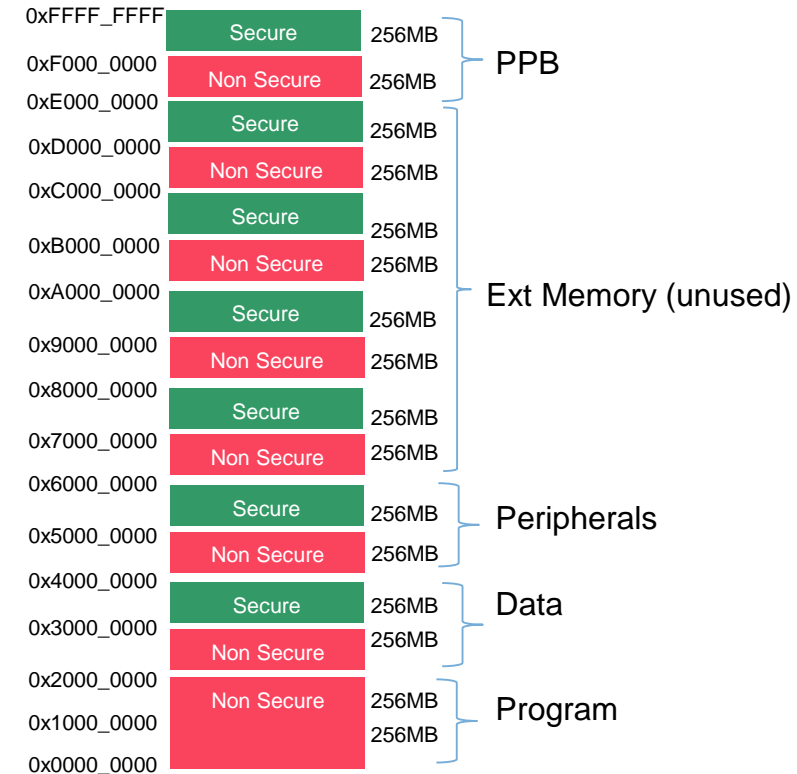
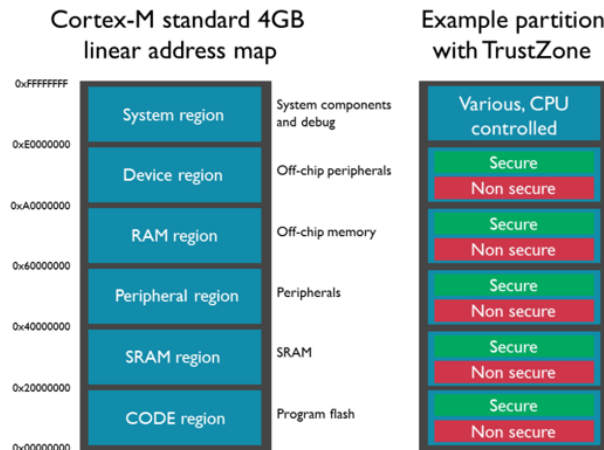
- For example, a designer could use bit [28] of the address to define if a memory is Secure or Non-secure, resulting in the following example memory map.

Address	Type	Security
0xFFFFFFFF	Device system	Various (CPU controlled)
0xF0000000		Secure
0xE0000000	Device system	Non-secure
0xD0000000		Secure
0xC0000000		Non-secure
0xB0000000		Secure
0xA0000000	RAM (WB)	Non-secure
0x90000000		Secure
0x80000000	RAM (WT)	Non-secure
0x70000000		Secure
0x60000000	Device	Non-secure
0x50000000		Secure
0x40000000	SRAM	Non-secure
0x30000000		Secure
0x20000000	Code	Non-secure
0x10000000		Secure
0x00000000		Non-secure



LPC55Sxx IDAU

- Simple IDAU, without creating a critical timing path. (CM33 does allow little for IDAU function)
 - **Addresses 0x0000_0000 to 0x1FFF_FFFF are NS**
 - Addresses 0x2000_0000 to 0xFFFF_FFFF
 - If Address Bit_28 = 0 Non-Secure
 - If Address Bit_28 = 1 Secure
- All peripherals and memories are aliased at two locations.



SAU

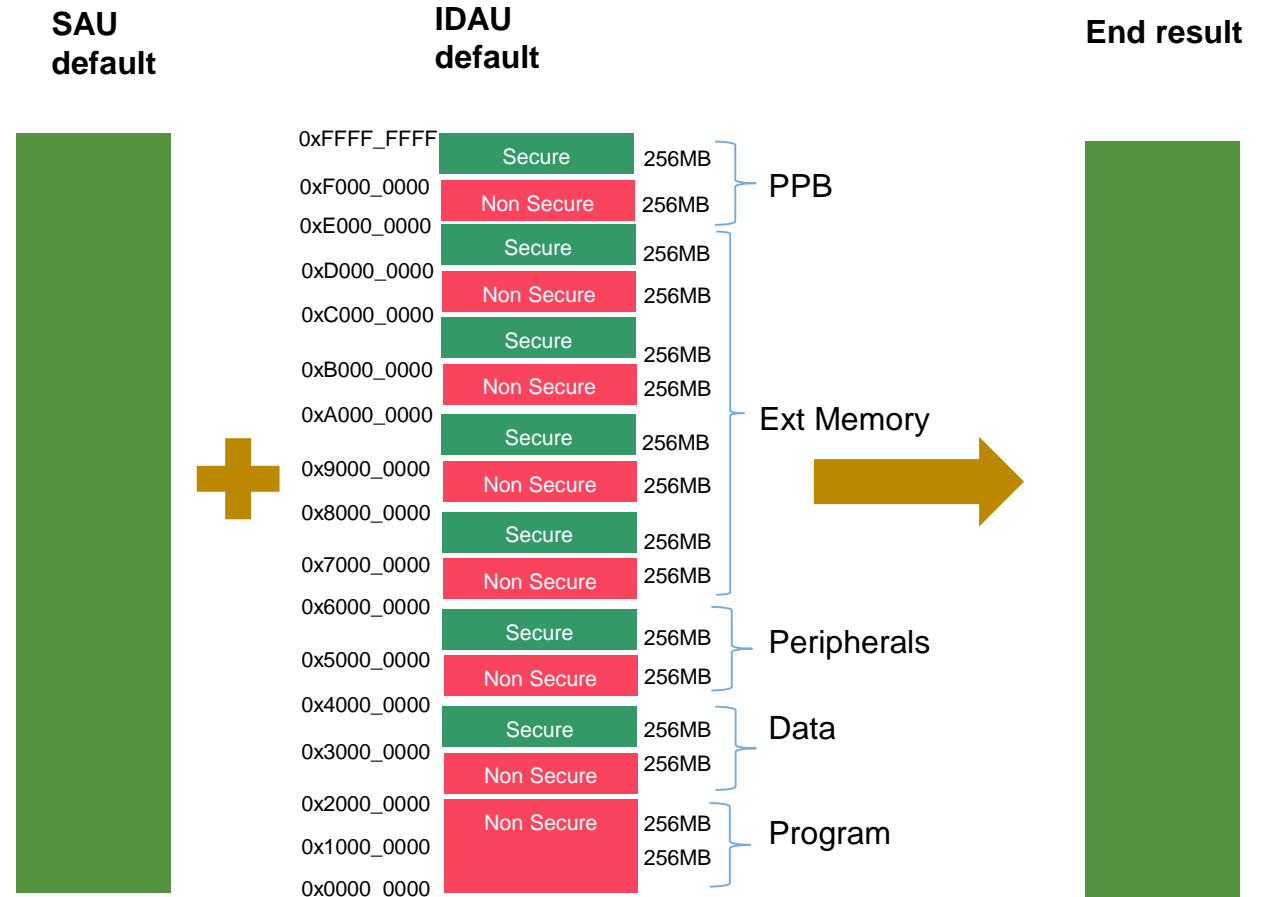
- The SAU define region numbers for each of the memory regions. **The region numbers** are 8-bit, and are used by **the Test Target(TT) instruction** to allow software to determine access permissions and security attribute of objects in memory.
- The number of regions that are included in the SAU can be configured to be either **0, 4 or 8**.

Note

When programming the SAU Non-secure regions, you must ensure that Secure data and code is not exposed to Non-secure applications.

Security Attribution Unit

- 8 regions are supported
- Used to override IDAU's fixed map
- Used to define NSC regions
- **By default all memory is set to secure**
 - At least one SAU descriptor should be used to make IDAU effective. Or set ALLNS bit in SAU control.



Security Attribution Unit Violation

- An attribution unit (AU) violation is raised by either the SAU or the IDAU.
 - All boundaries between address ranges with different security attributes must be **aligned to 32-byte boundaries**.
 - The behavior of the following address ranges is fixed, so SAU and IDAU can't change:
 - 0xF0000000 - 0xFFFFFFFF
 - On LPC55Sxx it is always marked as Secure and not Non-secure callable for instruction fetches.

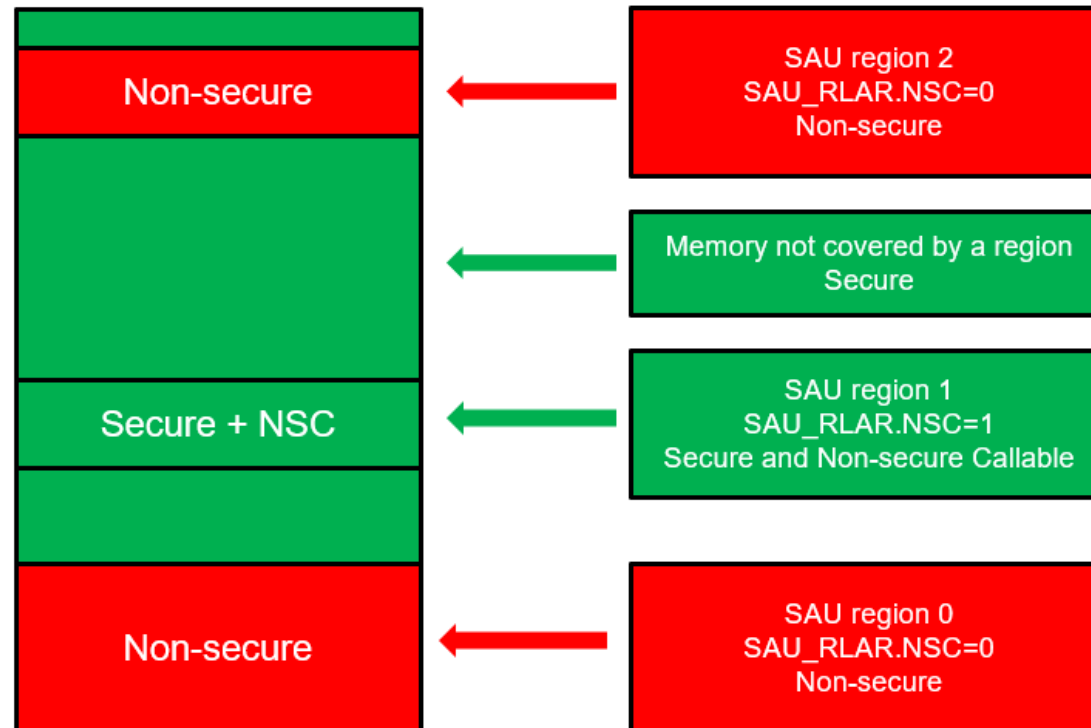
SAU Register

- The SAU can only be programmed in **Secure state**.

Address	Name	Description
0xE000EDD0	SAU_CTRL	SAU Control register
0xE000EDD4	SAU_TYPE	SAU Type register
0xE000EDD8	SAU_RNR	SAU Region Number Register
0xE000EDDC	SAU_RBAR	SAU Region Base Address Register
0xE000EDE0	SAU_RLAR	SAU Region Limit Address Register

SAU region configuration

- Regions are enabled individually using SAU_RLAR.
- The region is Non-secure when SAU_RLAR.ENABLE = 1 and SUA_RLAR.NSC=0.
- The region is Secure and Non-secure callable when SAU_RLAR.ENABLE = 1 and SUA_RLAR.NSC=1.



Configuring the SAU with CMSIS

- CMSIS-CORE now provide `partition_<device>.h`
 - `TZ_SAU_Setup()` used to configure SAU regions
- Some software tools provide SAU configuration wizards

Option	Value
<input checked="" type="checkbox"/> Initialize Security Attribution Unit (SAU) CTRL regi...	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> Initialize Security Attribution Unit (SAU) Address ...	<input checked="" type="checkbox"/>
<input checked="" type="checkbox"/> Initialize SAU Region 0	<input checked="" type="checkbox"/>
Start Address	0x0000 0000
End Address	0x001F FFFF
Region is	Secure, Non-Secure Callable
<input checked="" type="checkbox"/> Initialize SAU Region 1	<input checked="" type="checkbox"/>
Start Address	0x0020 0000
End Address	0x003F FFFF
Region is	Non-Secure
<input checked="" type="checkbox"/> Initialize SAU Region 2	<input checked="" type="checkbox"/>
Start Address	0x2020 0000
End Address	0x203F FFFF
Region is	Non-Secure
<input type="checkbox"/> Initialize SAU Region 3	<input type="checkbox"/>
<input type="checkbox"/> Initialize SAU Region 4	<input type="checkbox"/>

Initialize SAU Region 2
Setup SAU Region 2 memory attributes

Text Editor | **Configuration Wizard**

Build Output
*** Using Compiler 'V6.9', folder: 'C:\software_install\Keil5.25\ARM\ARMCLANG\Bin'

SAU Configuration example

The following example CMSIS code shows how the you can configure the SAU for two regions

```
// Configure SAU using CMSIS

// Configure SAU Region 0
// Start Address 0x00200000
// Limit Address 0x003FFFE0
// Secure non-secure callable

// Use CMSIS to access SAU Region Number Register (SAU_RNR)
// Select region 0
SAU->RNR = (0);
// Set SAU Region Base Address Register (SAU_RBAR)
SAU->RBAR = (0x00200000U & SAU_RBAR_BADDR_Msk);
// Set SAU Region Limit Address Register (SAU_RLAR)
SAU->RLAR = (0x003FFFE0U & SAU_RLAR_LADDR_Msk) | 1U << SAU_RLAR_NSC_Pos) &
SAU_RLAR_NSC_Msk) | 1U
```

SAU Configuration example

```
// Configure SAU Region 1
// Start Address 0x20200000
// Limit Address 0x203FFFE0
// Non-secure

// Select region 1
SAU->RNR = (1);
// Set SAU Region Base Address Register (SAU_RBAR)
SAU->RBAR = (0x20200000U & SAU_RBAR_BADDR_Msk);
// Set SAU Region Limit Address Register (SAU_RLAR)
SAU->RLAR = (0x203FFFE0U & SAU_RLAR_LADDR_Msk) | 0U << SAU_RLAR_NSC_Pos) &
SAU_RLAR_NSC_Msk) | 1U

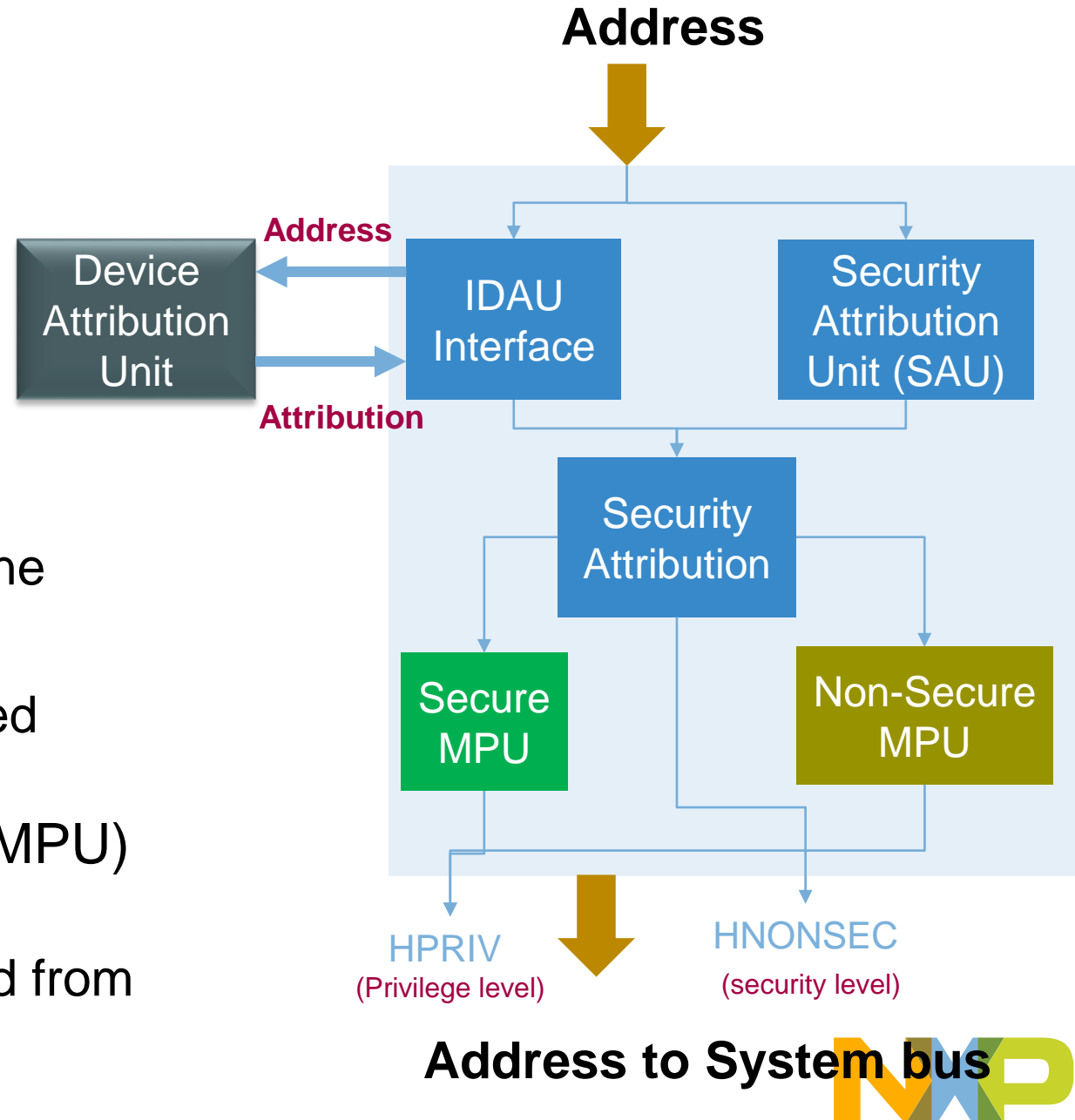
// Enable SAU
// Use CMSIS to access SAU Control Register (SAU_CTRL)
// Set ENABLE bit[0] to 1
// Set ALLNS bit[1] to 1 All memory is secure when SAU is disabled

SAU->CTRL = ((SAU_INIT_CTRL_ENABLE << SAU_CTRL_ENABLE_Pos) &
SAU->SAU_CTRL_ENABLE_Msk) |
            ((SAU_INIT_CTRL_ALLNS << SAU_CTRL_ALLNS_Pos) &
SAU_CTRL_ALLNS_Msk) ;
```



Security Defined by Address

- All address are either Secure or Non-secure.
- Security Attribution Unit (SAU)
 - SAU inside ARMv8M is similar to MPU
 - By default all memories are secure
 - LPC55S supports 8 SAU regions to define
- **NXP's Device Attribution Unit**
 - connects through Implementation Defined Attribution Unit (IDAU) interface
- Independent memory protection unit (MPU) per security state
 - Secure OS can be completely decoupled from



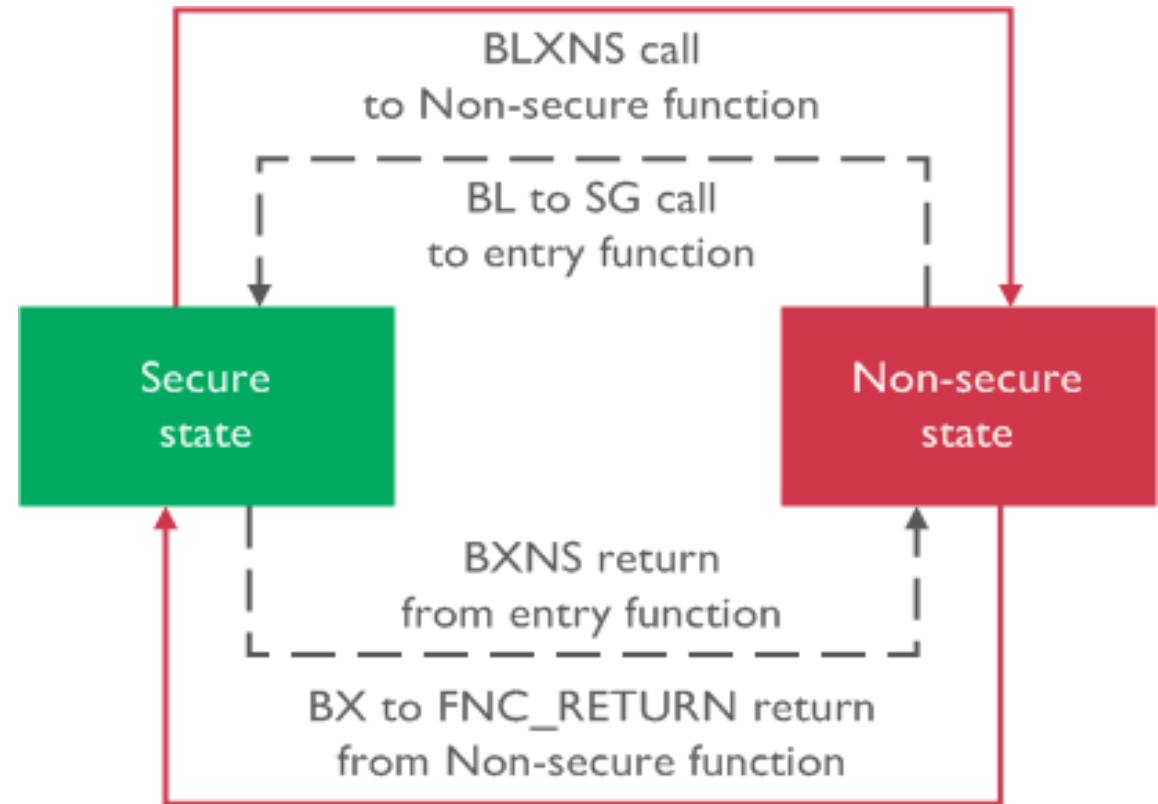
SWITCHING BETWEEN SECURE AND NON-SECURE STATES



Switching between Secure and Non-secure states

- The ARMv8-M Security Extensions allow direct calling between Secure and Non-secure software. **Several instructions** are available for state transition handling in ARMv8-M processors:

SG	Secure gateway. Used for switching from Non-secure to Secure state at the first instruction of Secure entry point.
BXNS	Branch with exchange to Non-secure state. Used by Secure software to branch or return to Non-secure program.
BLXNS	Branch with link and exchange to Non-secure state. Used by Secure software to call Non-secure functions.



Entry function

- An entry function can be called from non-secure state or secure state.
- `__attribute__((cmse_nonsecure_entry))`

```
/* Non-secure callable (entry) function */  
int func1(int x) __attribute__((cmse_nonsecure_entry)) {  
    return x+3;  
}
```

```
/* Call non-secure callable function func1 */  
val1 = func1 (1);
```

func1() is defined at secure project

func1() is executed in non-secure projects

Non-secure function call

- A call to a function that switches state from secure to non-secure is called a non-secure function call.
- Define a Non-secure function pointer using:

```
__attribute__((cmse_nonsecure_call))
```

For example

```
typedef void __attribute__((cmse_nonsecure_call)) nsfunc(void);
```

```
Nsfunc *FunctionPointer;
```

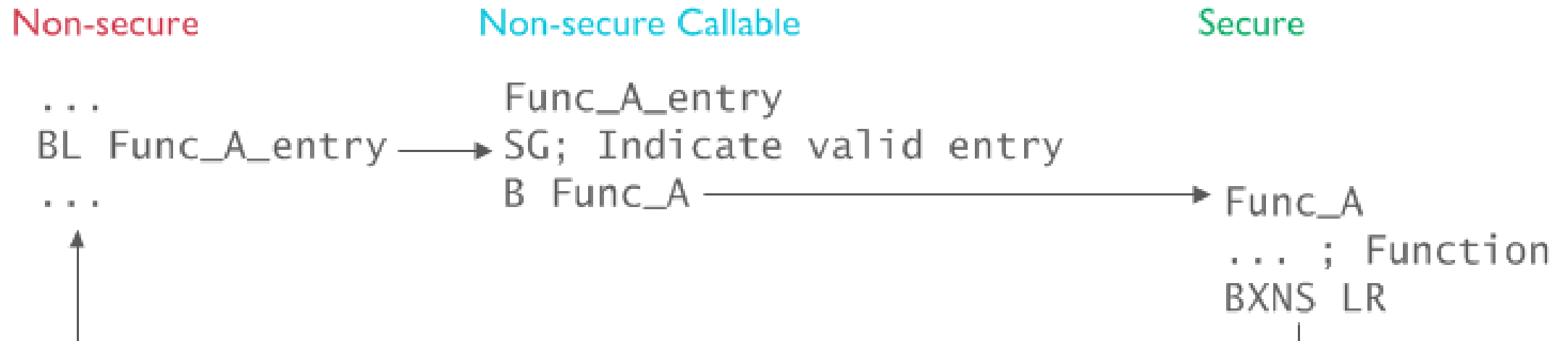
```
FunctionPointer=cmse_nsfptr_create((nsfunc *) (0x21000248u));
```

```
If(cmse_is_nsfptr(FunctionPointer))
```

```
FunctionPointer();
```

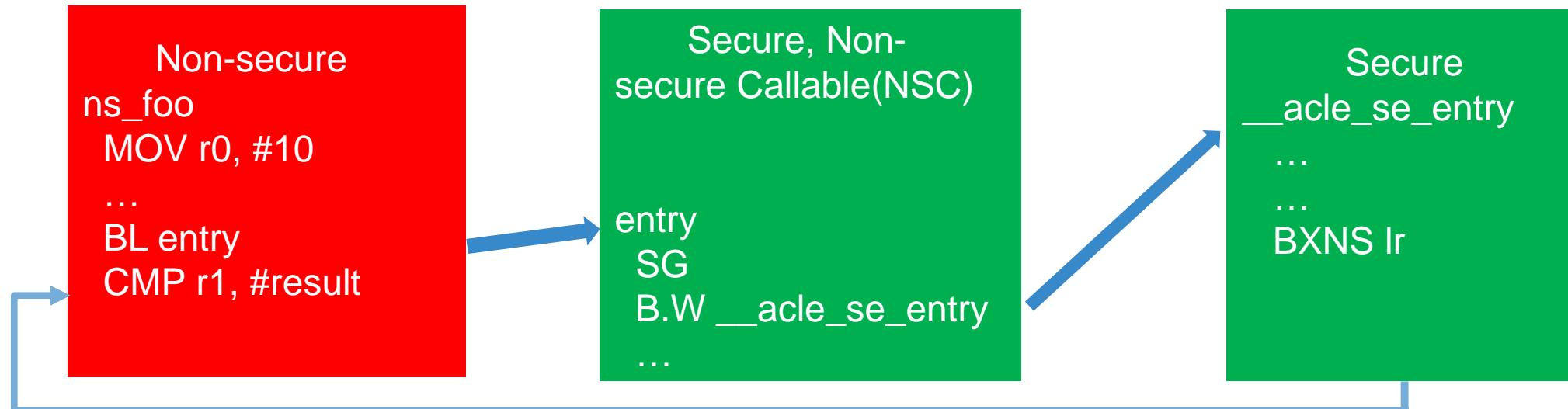
Calling secure code from non-secure code

- A direct API function call from Non-secure to Secure software entry points is allowed if the first instruction of the entry point is **SG**, and it is in a **Non-secure callable** memory location.



Calling secure code from non-secure code

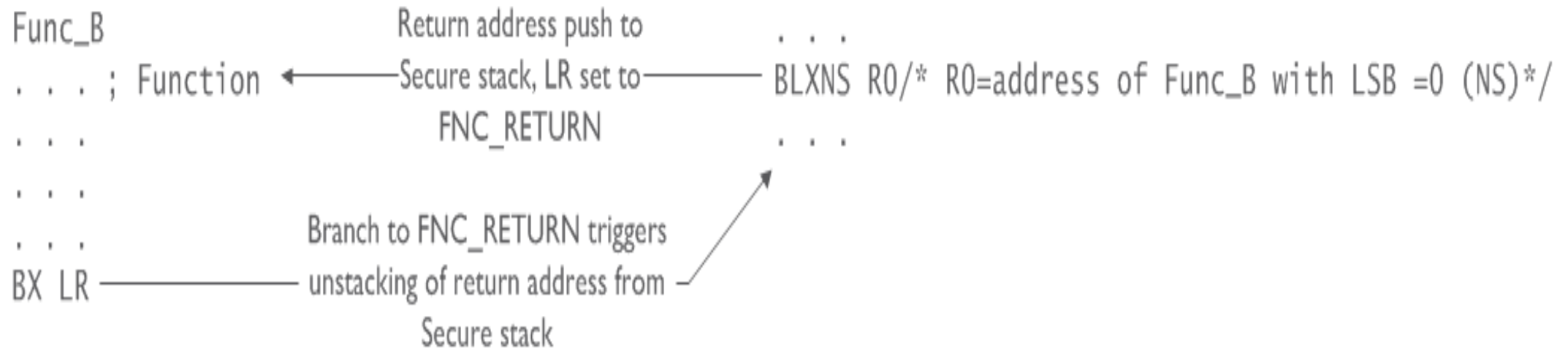
- Non-secure code can branch into secure code
 - This allows secure libraries to be used by non-secure applications
- The branch target address:
 - Must be mapped as **Secure and Non-secure Callable** by the SAU/IDAU
 - Must contain a **Secure Gateway(SG)** instruction
- When executed from Secure, NSC memory, the SG instruction
 - Changes the security state to Secure
 - Sets $Ir[0]$ to 0 to allow the secure code to return using a BXNS instruction



Calling non-secure code from secure code

Non-secure

Secure



Calling non-secure code from secure code

- Before executing a BXNS or BLXNS instruction
 1. Save all active non-banked registers by copying them to secure memory
 2. The branch target address must have the LSB set to 0 and reside in non-secure memory
 3. Clear all non-banked registers except:
 - Link register(BLXNS only)
 - Register that hold arguments for the call
 - Register that do not hold secret information

```
PUSH {r0-r12}
MOVW r0, #0x0
MOVT r0, #0x2100
MOV r1, r0
MOV r2, r0
MOV r3, r0
MOV r4, r0
MOV r5, r0
MOV r6, r0
MOV r7, r0
MOV r8, r0
MOV r9, r0
MOV r10, r0
MOV r11, r0
MOV r12, r0
MSR APSR_nzcvq, r0
BLXNS r0
```

Push r0-r12 onto secure stack

Move address into r0(LSB=0)


Clear registers(overwrite with Non-secure branch address)

Non-secure branch

TT Instruction – Test Target

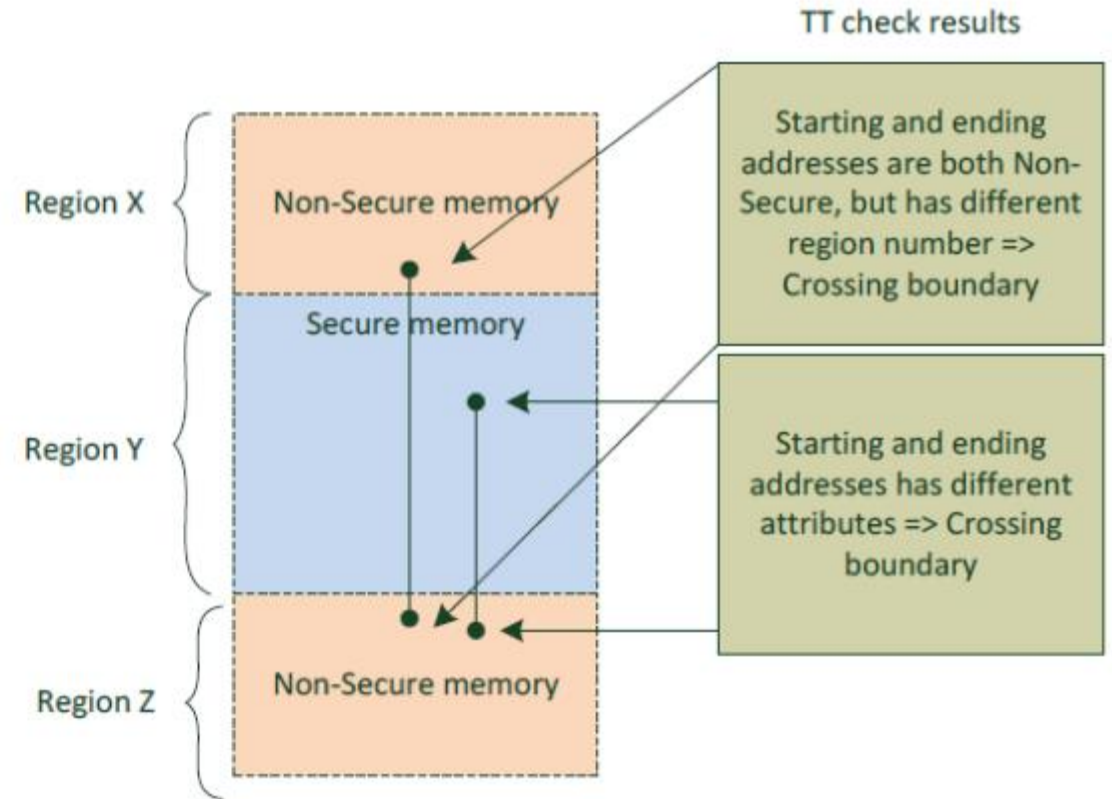
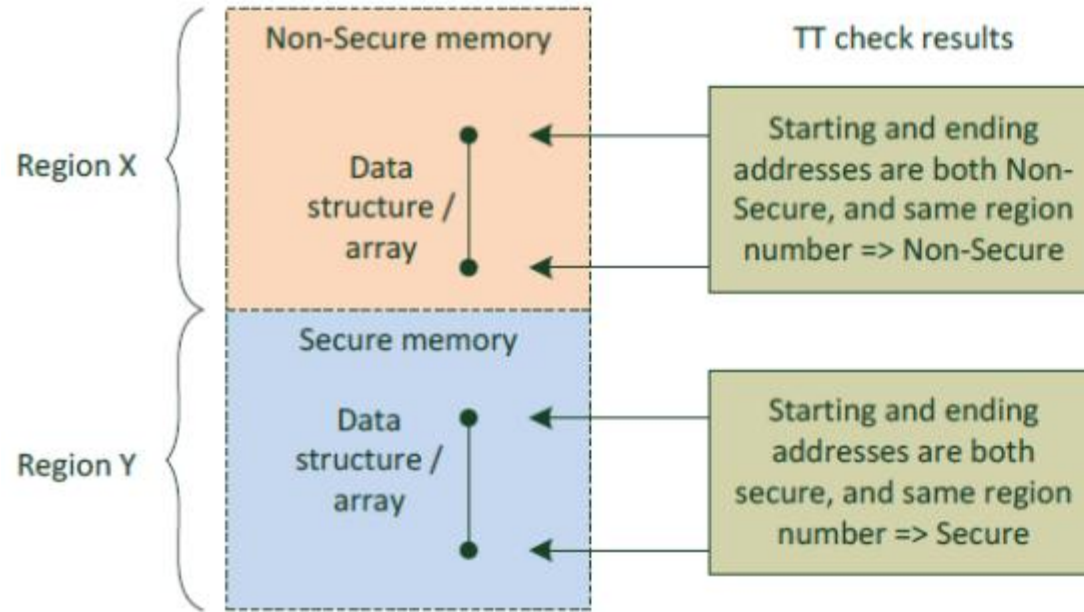
- SAU/IDAU generate a Region Number for each region.
 - Software can check a region to determine security
- TT returns RN and Attribute (NS or secure) on an address
 - Use TT on start and end address
 - Can determine whether memory has required security attributes.
- LPC55xxx IDAU returns region number as below
 - $\text{IDAU_RN}[7:0] = (\{4'h0, \text{idau_addr_a}[31:28]\})$
- Usage examples
 - When setting secure DMA for data transfers, secure code can check attributes of buffer pointer passed by NS code

TT instruction

- New **Test Target(TT)** instruction returns the MPU and SAU configuration for an address
- TT {cond} {q} Rd, Rn
 - Rn contains the address to be tested
 - Rd returns the attributes of the address 
- Secure functions may be passed pointers when called from Non-secure code
 - TT can validate that the calling function has the rights to access the memory
 - TT can validate that the address is Non-secure

[7:0]	MREGION	MPU region
[15:8]	SREGION	SAU region
[16]	MRVALID	Is MREGION valid?
[17]	SRVALID	Is SREGION valid?
[18]	R	Is address readable?
[19]	RW	Is address RW?
[20]	NSR	Is Non-secure && readable?
[21]	NSRW	Is Non-secure && RW?
[22]	S	Is address Secure?
[23]	IRVALID	Is IREGION valid?
[31:24]	IREGION	IDAU region


How TT works



Note

The MPU, SAU and IDAU in ARMv8-M do not allow regions to overlap.

TT instruction example

- `PRINTF_NSE("Welcome in normal world!\r\n");` 
- */* Check whether string is located in non-secure memory */*
*if (cmse_check_address_range((void *)s, string_length, CMSE_NONSECURE |*
CMSE_MPU_READ) == NULL)
{
PRINTF("String is not located in normal world!\r\n");
abort();
}

`cmse_check_address_range()`  `TT instruction`

EXCEPTIONS

Interrupts and exceptions

- **Interrupt can be programmed as secure or non-secure interrupts**
- **Some system exceptions are banked(e.g. SysTick)**
- **New SecureFault exception**
- **Banked System Control Block(SCB) registers**
 - Two VTOR- Separate vector tables for Secure exceptions and Non-Secure exceptions
 - Non-Secure SCB registers can be accessed from Secure side via alias address
- **Priority of Secure exceptions/interrupts can share same levels as Non-secure's, or can be higher priority(programmable)**

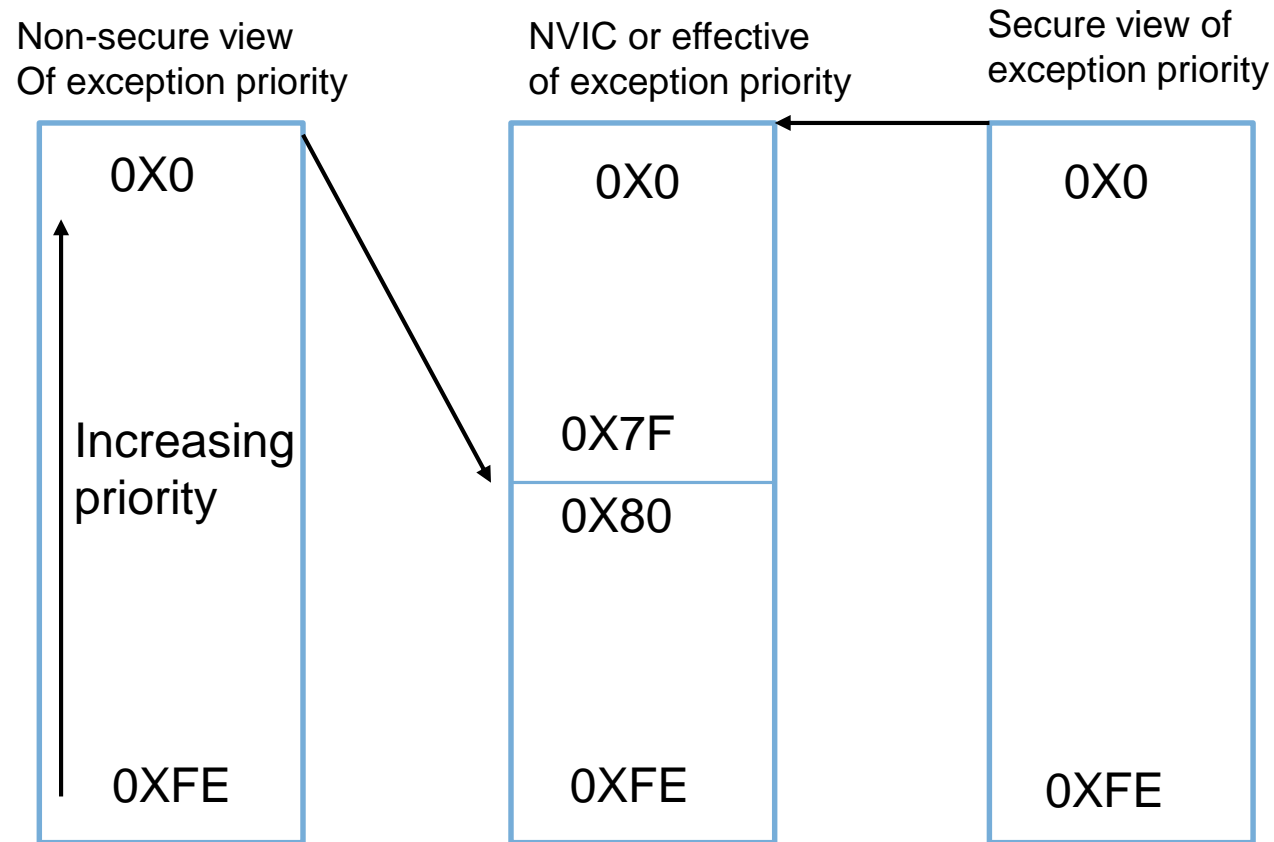
Exception priorities overview

Name	Exception #	Exception Priority #	Security
Interrupts #0-N	16 to 16+N	0-255(programmable)	Configurable
SysTick	15	0-255(programmable)	Banked
PendSV	14	0-255(programmable)	Banked
DebugMonitor	12	0-255(programmable)	Configurable
SVCall	11	0-255(programmable)	Banked
SecureFault	7	0-255(programmable)	Secure
UsageFault	6	0-255(programmable)	Banked
BusFault	5	0-255(programmable)	Configurable
MemManage	4	0-255(programmable)	Banked
Non-secure HardFault	3	-1	Non-secure
Secure HardFault	3	-3 or -1(programmable)	Secure
Non Maskable Interrupt (NMI)	2	-2	Configurable
Reset	1	-4	Secure

- The lower the priority number, the higher the priority level
- NMI, HardFault and BusFault default to be Secure, but can be set to NS

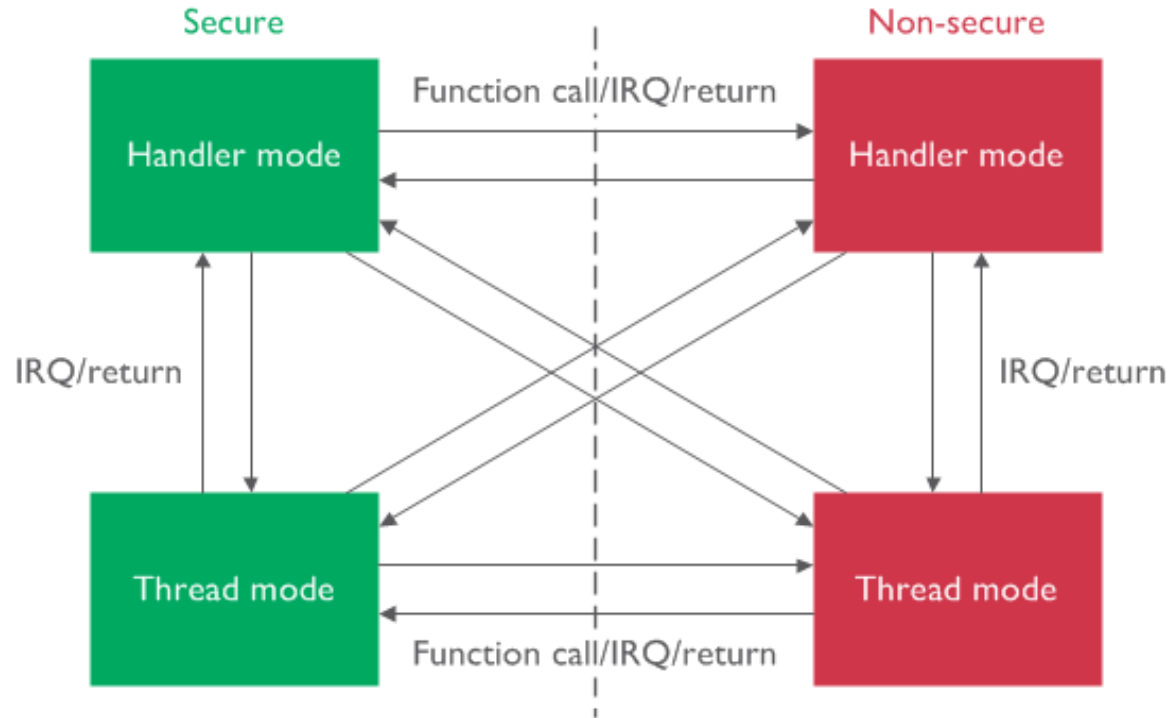
Secure exception prioritization

- Non-secure exception can be forced into the lower half of the priority range
 - Using AIRCR_S.PRIS



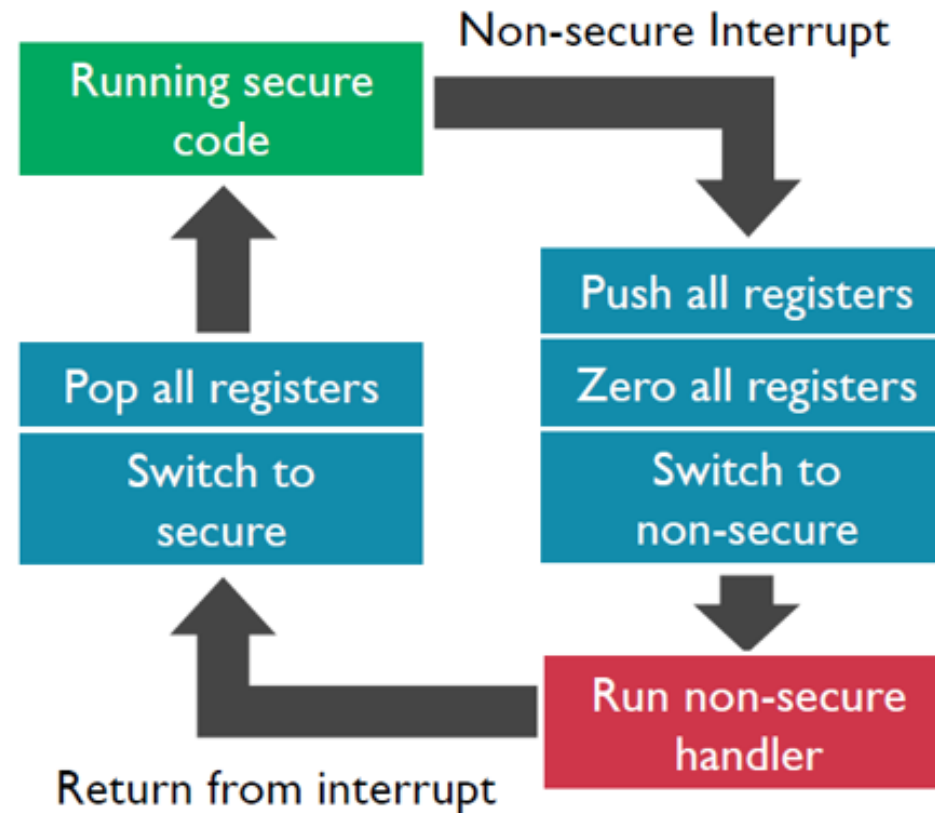
Interrupts and exceptions

- State transitions can also happen due to exceptions and interrupts.



TrustZone interrupt security

- The processor automatically pushes all Secure information onto the **Secure stack** and **erases** the contents from the **register banks**, therefore avoiding an **information leak**.



恩智浦MCU加油站

- 恩智浦工程师原创技术分享
- 欢迎关注，欢迎投稿





SECURE CONNECTIONS
FOR A SMARTER WORLD