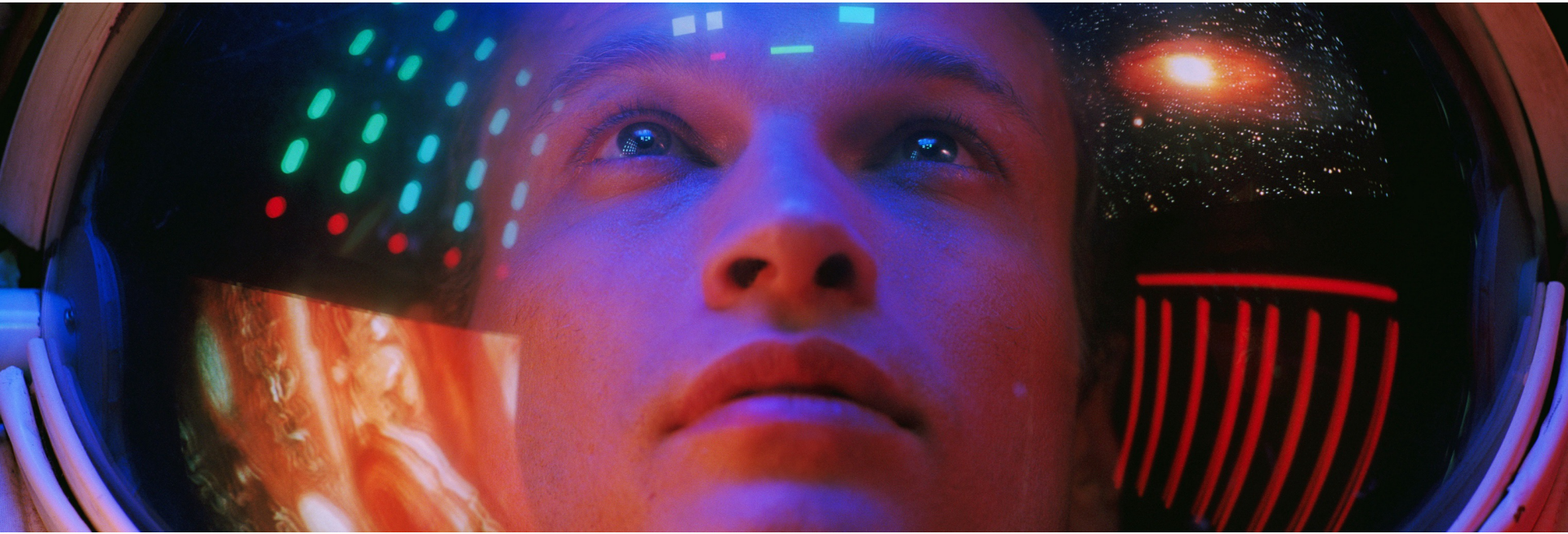
MSE | MASTER OF SCIENCE IN ENGINEERING

MA_EmbReal

Robust Patterns for Reliable Systems (IV)

Version: 1.4

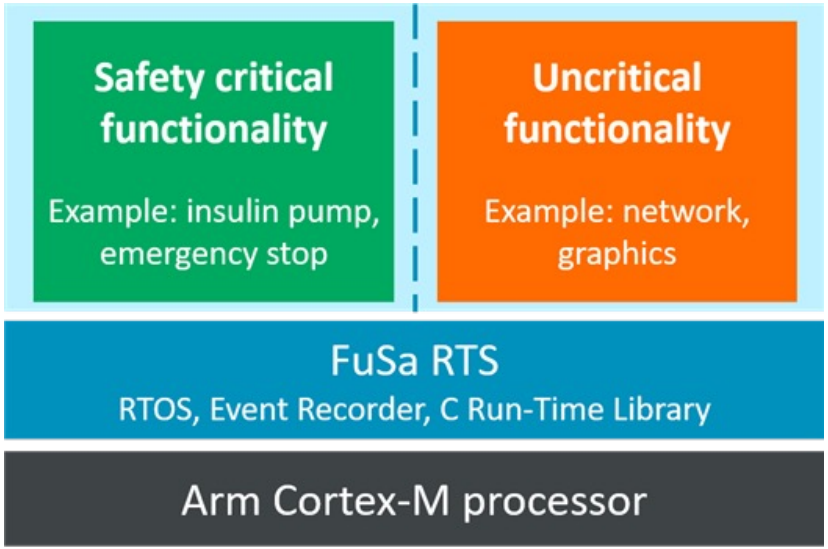
Serge Ayer – Luca Haab | 14.05.2024 | Cours MSE



Spatial Isolation

Tasks Spatial Isolation

Recap

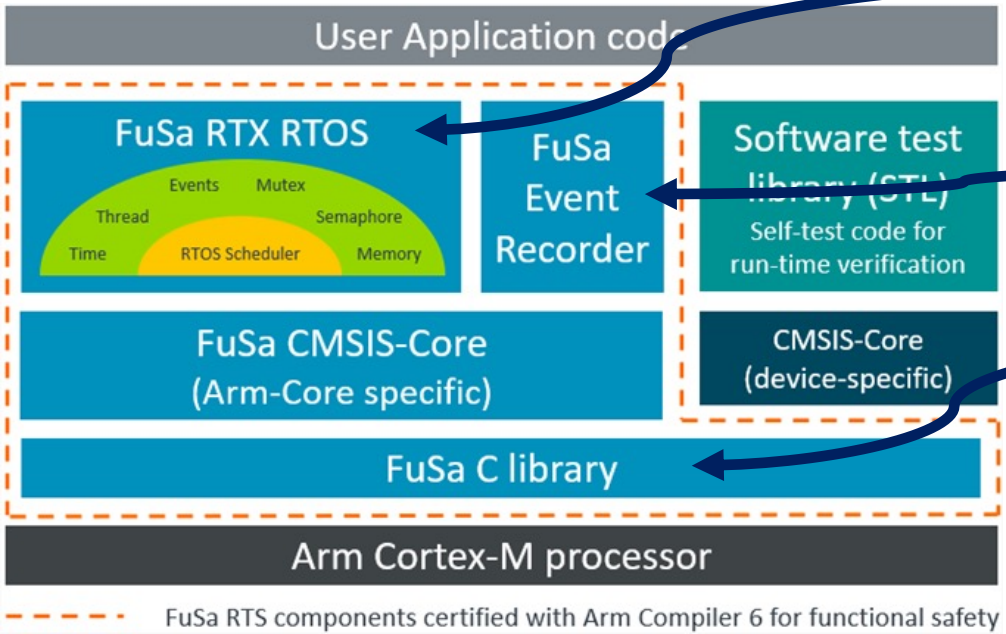


- FuSa allows task isolation through:
  - MPU protected zones
  - Safety classes
  - Thread watchdogs
  - System recovery

Source: https://community.arm.com/arm-community-blogs/b/tools-software-ides-blog/posts/process-isolation-with-fusa-rts

Tasks Spatial Isolation

Recap



Derivative of Keil RTX5, with additional safety components

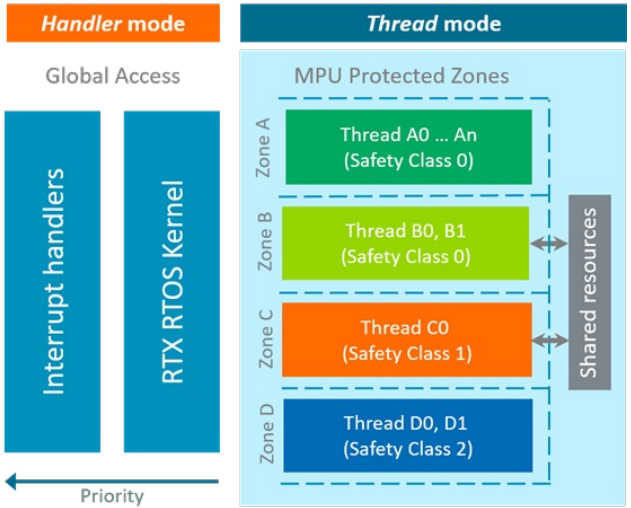
For dynamic analysis

Restricted subset of the C library (with usage guidelines)

Source: https://community.arm.com/arm-community-blogs/b/tools-software-ides-blog/posts/process-isolation-with-fusa-rts

Tasks Spatial Isolation

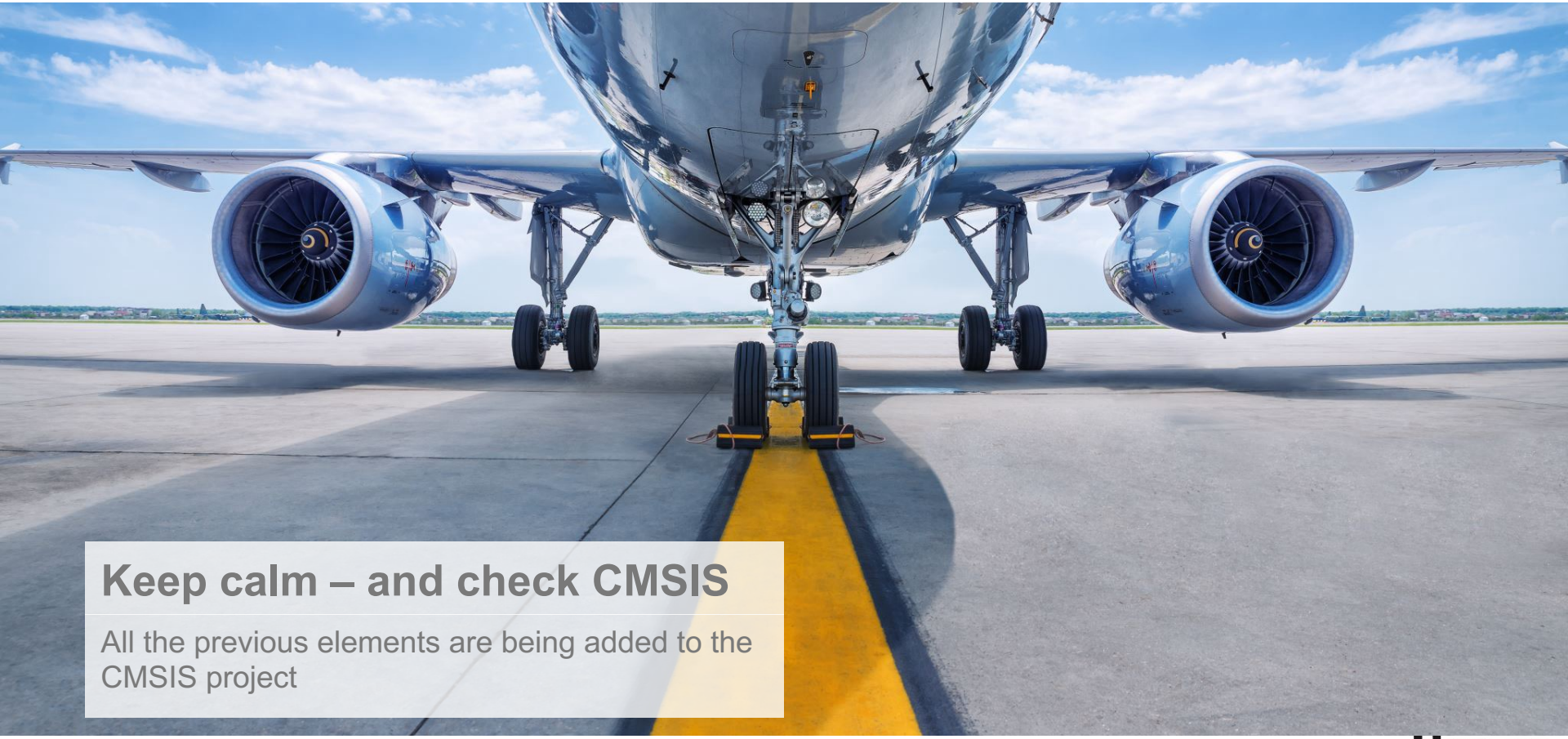
Recap



- Kernel is executed in handler mode
- MPU Protected Zones
  - Spatial isolation between tasks
  - Each FuSa RTX thread gets assigned to a MPU protected zone
- Safety classes
  - Enable access control to RTOS objects
- Application threads can individually switch between unprivileged or privileged modes with FuSa RTX

Source: https://community.arm.com/arm-community-blogs/b/tools-software-ides-blog/posts/process-isolation-with-fusa-rts

Wait, but we don’t have FuSa RTX!

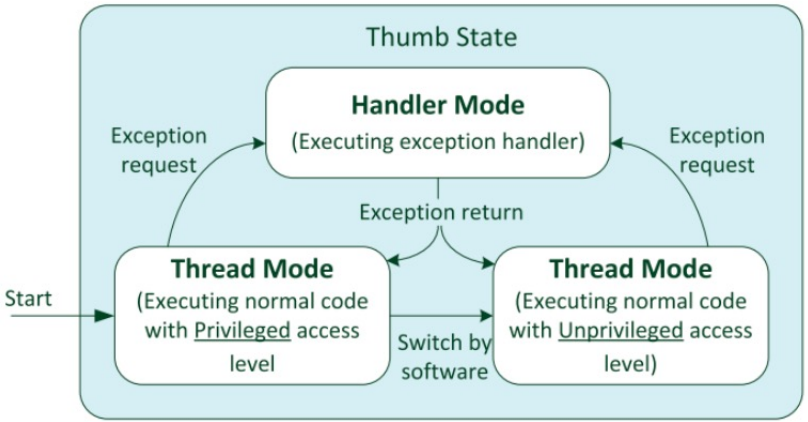


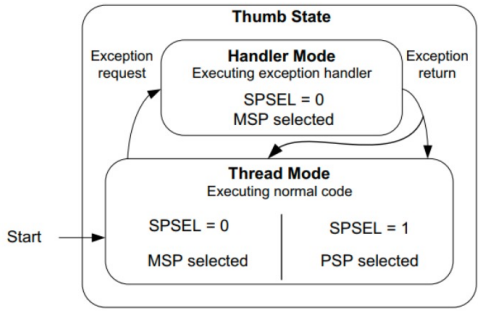
Keep calm – and check CMSIS

All the previous elements are being added to the CMSIS project

Kernel executed in handler mode

- Protects kernel memory
- Has dedicate Stack (MSP)





Source: "The Definitive Guide To ARM Cortex-M3 and ARM Cortex-M4 Processors Third Edition", J. Yiu, 2014

Processor mode and privilege levels

Thread mode

Used to execute application software. The processor enters Thread mode when it comes out of reset.

Handler mode

Used to handle exceptions. The processor returns to Thread mode when it has finished all exception processing.

The *privilege levels* for software execution are:

Unprivileged

The software:

- has limited access to the `MSR` and `MRS` instructions, and cannot use the `CPS` instruction
- cannot access the system timer, NVIC, or system control block
- might have restricted access to memory or peripherals.

Unprivileged software executes at the unprivileged level.

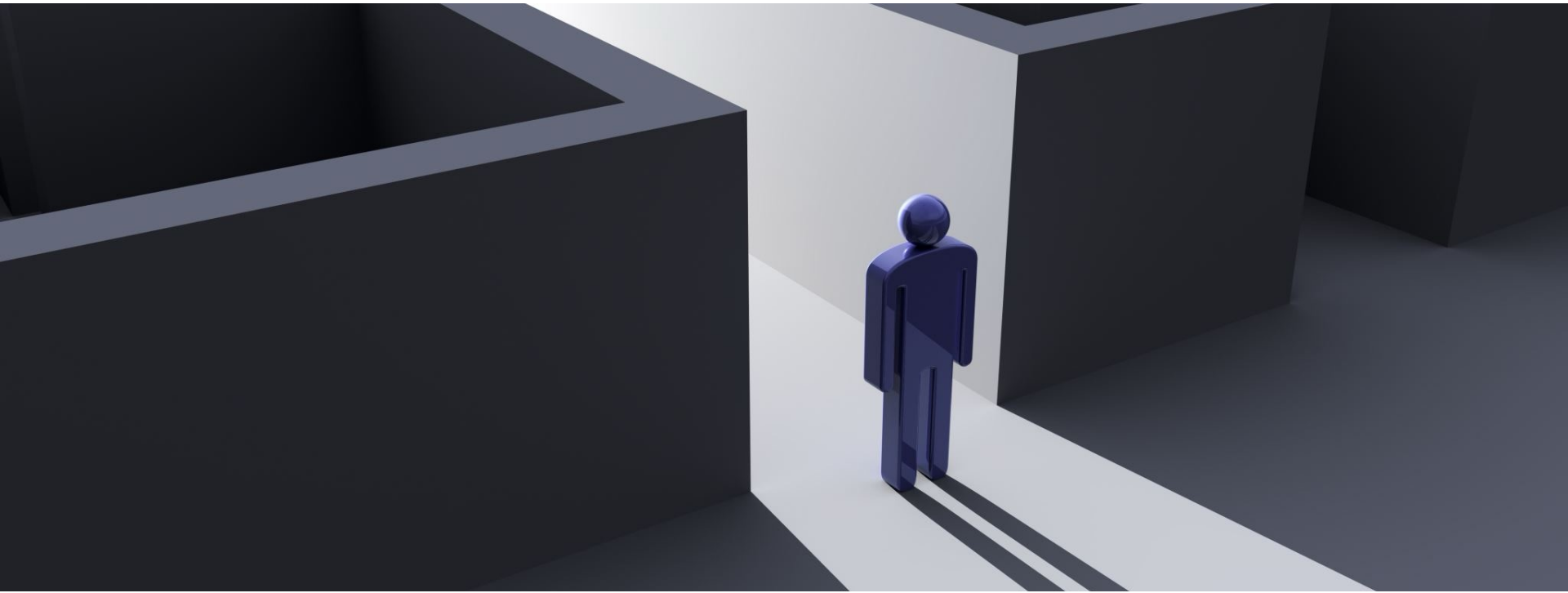
Privileged

The software can use all the instructions and has access to all resources.

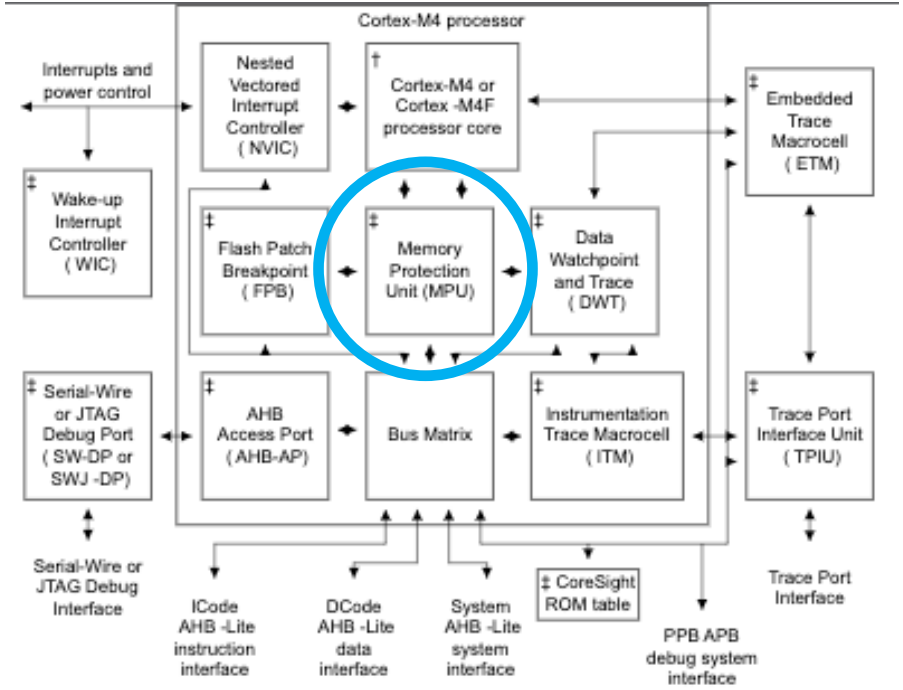
Privileged software executes at the privileged level.

Source: https://developer.arm.com/documentation/dui0552/a/the-cortex-m3-processor/programmers-model/processor-mode-and-privilege-levels-for-software-execution

Is this spatial isolation?



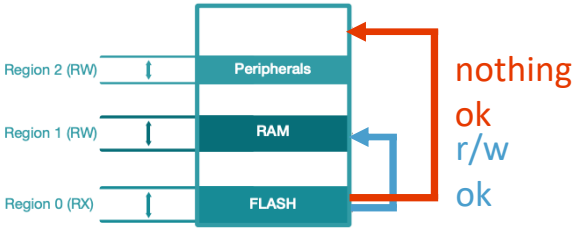
Enter the MPU

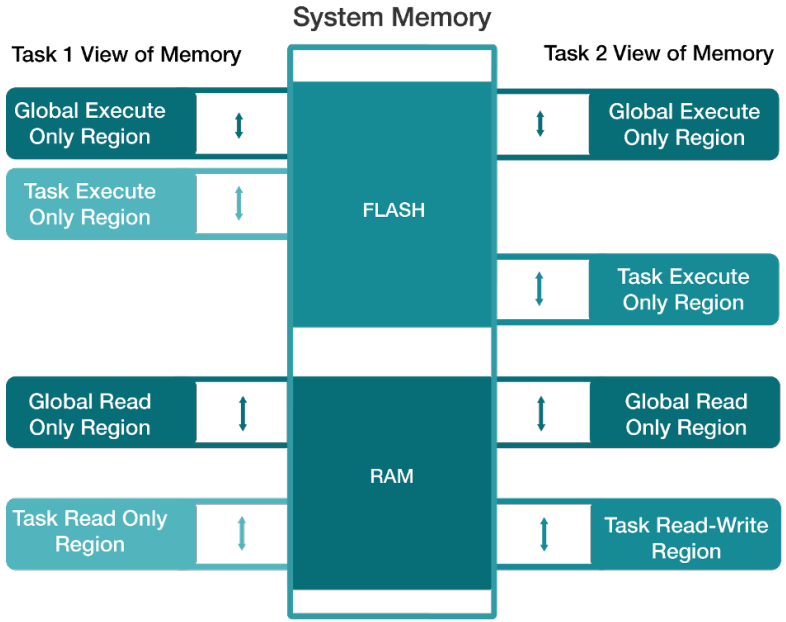


Source: https://developer.arm.com/documentation/ddi0439/b/Functional-Description/About-the-functions

Enter the MPU

- Allows setting up dedicated areas of memory
- Controls who and what accesses are allowed (r, w, x)
- Not respecting this will result in an exception
- Always good?





Source: https://www.freertos.org/fr-content-src/uploads/2021/02/MPU_regions.png

Let’s put that into practice

- Identify the resources
- Partion them
- Take the MPU limits into account

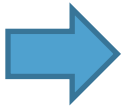
Name	Pe...	Size	Start	End
STM32L475VGTx				
Memory				
Main_Flash	rx	1 MB	0x08000000	0x080FFFFF
SRAM2	rwx	32 KB	0x10000000	0x10007FFF
SRAM1	rwx	96 KB	0x20000000	0x20017FFF
Peripherals				
ADC				
ADC1	rw	185 B	0x50040000	0x500400B8
ADC2	rw	185 B	0x50040100	0x500401B8
ADC3	rw	185 B	0x50040200	0x500402B8
ADC123_Common	rw	17 B	0x50040300	0x50040310

Name	Permis...	Size	Start	End	Zone_APP	Zone_EVR	Zone_STL	Zone_IDLE	Zone_TIMER
STM32L475VGTx									
Memory									
Main_Flash	rx	1 MB	0x08000000	0x080FFFFF	☑	☑	☑	☑	☑
SRAM2	rwx	32 KB	0x10000000	0x10007FFF	☐	☐	☐	☐	☐
RAM_PRIV	rw,,p	32 KB	0x10000000	0x10007FFF	☐	☐	☐	☐	☐
SRAM1	rwx	96 KB	0x20000000	0x20017FFF	☐	☐	☐	☐	☐
RAM_APP	rw,,u	16 KB	0x20000000	0x20003FFF	☑	☐	☐	☐	☐
RAM_EVR	rw,,u	16 KB	0x20004000	0x20007FFF	☑	☑	☑	☑	☑
RAM_STL	rw,,p	16 KB	0x20010000	0x20013FFF	☐	☐	☑	☐	☐
RAM_IDLE	rw,,u	16 KB	0x20008000	0x2000BFFF	☐	☐	☐	☑	☐
RAM_TIMER	rw,,u	16 KB	0x20014000	0x20017FFF	☐	☐	☐	☐	☑
RAM_SHARED	rw,,u	16 KB	0x2000C000	0x2000FFFF	☑	☑	☑	☑	☑

Examples related to STM32L475VGTx

Getting a (very different) linker script

```
LR_IROM1 0x08000000 0x00100000  {    ; load region size_region
  ER_IROM1 0x08000000 0x00100000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO, +XO) ;Any read/execute only data, i.e. the code
  }
  RW_IRAM1 0x20000000 0x00018000  {
   .ANY (+RW +ZI)  ;standard RAM, with read/write access, zero initialized
  }
}
```



```
LR_Main_Flash REGION_MAIN_FLASH_START REGION_MAIN_FLASH_SIZE {
  ER_Main_Flash REGION_MAIN_FLASH_START REGION_MAIN_FLASH_SIZE {
    * (RESET,+FIRST)
    * (InRoot$$Sections)
    .ANY (+RO, +XO)
  }
  RW_RAM_PRIV REGION_RAM_PRIV_START REGION_RAM_PRIV_SIZE {
    * (.data.os*)
    * (.bss.os*)
  }
  RW_RAM_APP REGION_RAM_APP_START REGION_RAM_APP_SIZE {
    app.o (+RW +ZI)
  }
  RW_RAM_EVR REGION_RAM_EVR_START UNINIT REGION_RAM_EVR_SIZE {
    EventRecorder.o (+ZI)
  }
  RW_RAM_IDLE REGION_RAM_IDLE_START REGION_RAM_IDLE_SIZE {
    rtx_lib.o (.bss.os.thread.idle.stack)
  }
  RW_RAM_SHARED REGION_RAM_SHARED_START REGION_RAM_SHARED_SIZE {
    * (ram_shared)
  }
```

- LR_Main_Flash contains all the read/execute only code and the startup information
- RAM_PRIV is privileged and contains the kernel data and bss segments
- RAM_APP contains application data that is read/write and zero initialized
- RAM_EVR contains all objects produced by the EventRecorder object and is zero initialized
- RAM_IDLE contains the Idle thread stack
- RAM_SHARED contains shared data

MPU reconfiguration

- Kernel checks at every context switch whether the next task lies in a different zone
- If yes, `osZoneSetup` will be used to reconfigure it

-> it is up to the user to implement this

```
/* Update MPU settings for newly activating Zone */
void osZoneSetup_Callback (uint32_t zone) {
      if (zone >= ZONES_NUM) {
      // Here issue an error for incorrect zone value
      }
      ARM_MPU_Load(mpu_table[zone], MPU_REGIONS);
}
```

All nicely covered right?

Well, what about RTOS objects?

- RTOS Objects are… pointer based
- As we access those through system calls (thus in handler mode)… we may thus say ”Houston we have a problem”

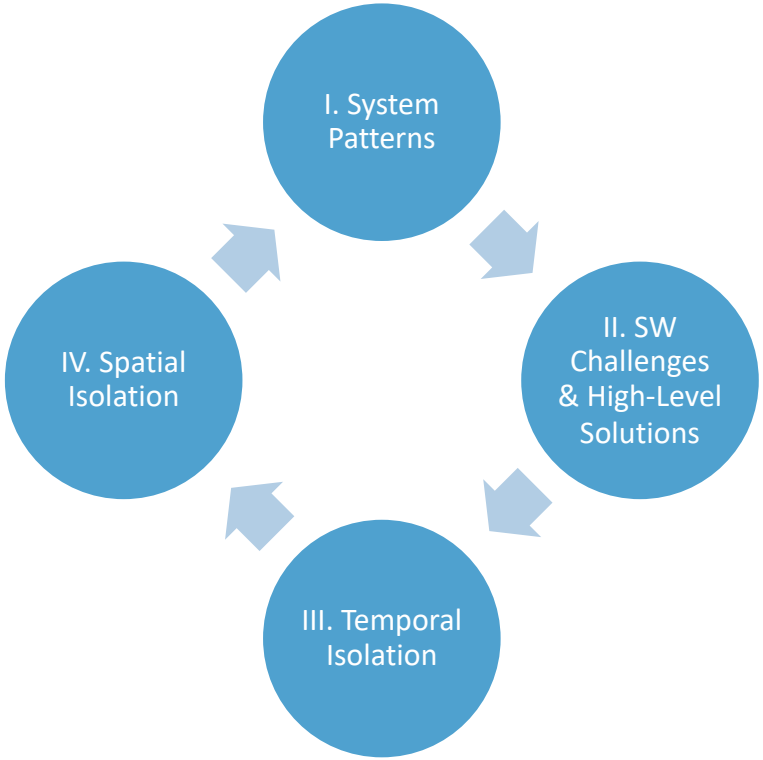
In come Safety Classes

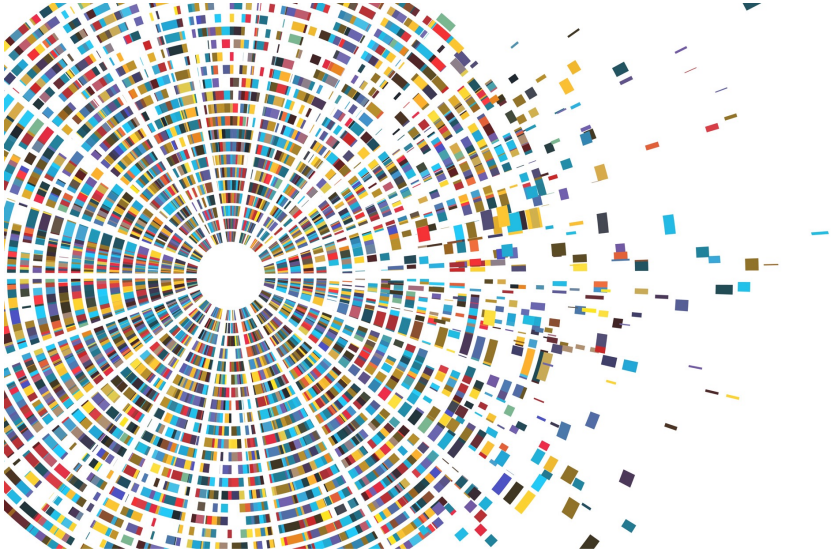
- Simple solution:
  objects cannot be modified by threads having a lower Safety Class assigned
- For example, it is not possible to change the priority or suspend a thread
- Threads of a higher safety class can create RTOS objects that belong to a lower or same safety class

Source: https://github.com/ARM-software/CMSIS_5/blob/96f949eeafcd44c39ac4f83312f41d2f8817bac5/CMSIS/DoxyGen/RTOS2/src/cmsis_os2_ProcessIsolation.txt

Robust Patterns: Recap

Robust Patterns: Full Circle





Let’s get dirty

https://embreal.isc.heia-fr.ch/exercises/robust-patterns-part3/

References

- https://arm-software.github.io/CMSIS_5/develop/Zone/html/pages.html
- https://github.com/ARM-software/CMSIS_5/blob/96f949eeafcd44c39ac4f83312f41d2f8817bac5/CMSIS/DoxyGen/RTOS2/src/cmsis_os2_ProcessIsolation.txt
- https://community.arm.com/arm-community-blogs/b/tools-software-ides-blog/posts/process-isolation-with-fusa-rts
- https://community.arm.com/arm-community-blogs/b/embedded-blog/posts/more-safety-and-engineering-efficiency-arms-new-runtime-software-system-to-accelerate-development-of-safety-applications-on-cortex-m-devices