



life.augmented

STP32MP1 coprocessor development classroom

Johnson Zhang

March 2021

- 1 STP32MP1 coprocessor overview
- 2 STP32MP1 coprocessor development
- 2 STP32MP1 inter-processor communication

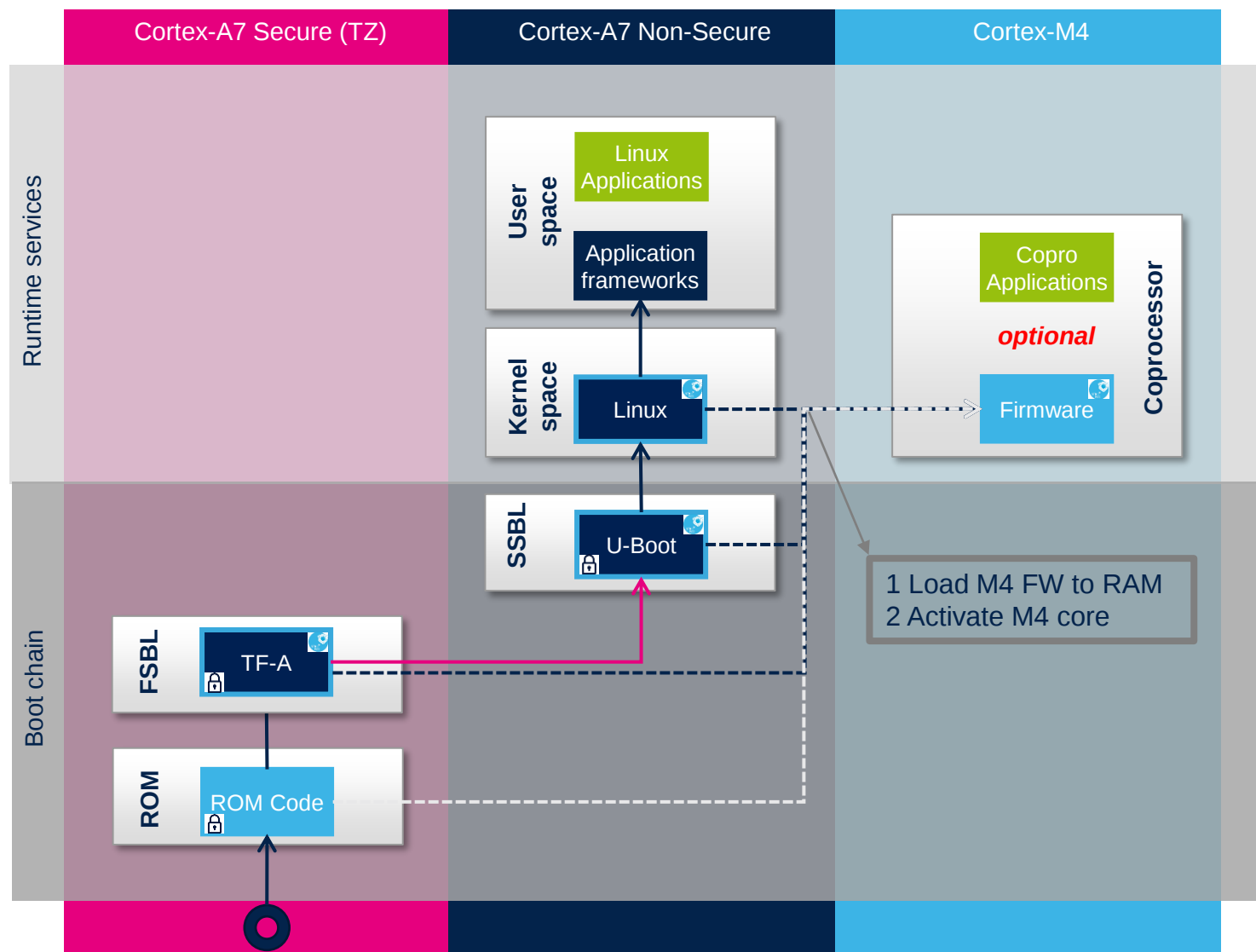
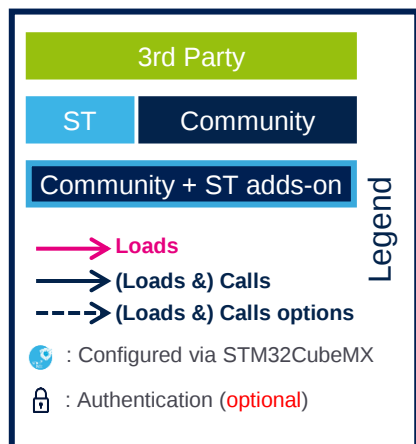
STP32MP1 coprocessor overview



STM32MP1 Block Diagram



M4 boot



STP32MP1 coprocessor development

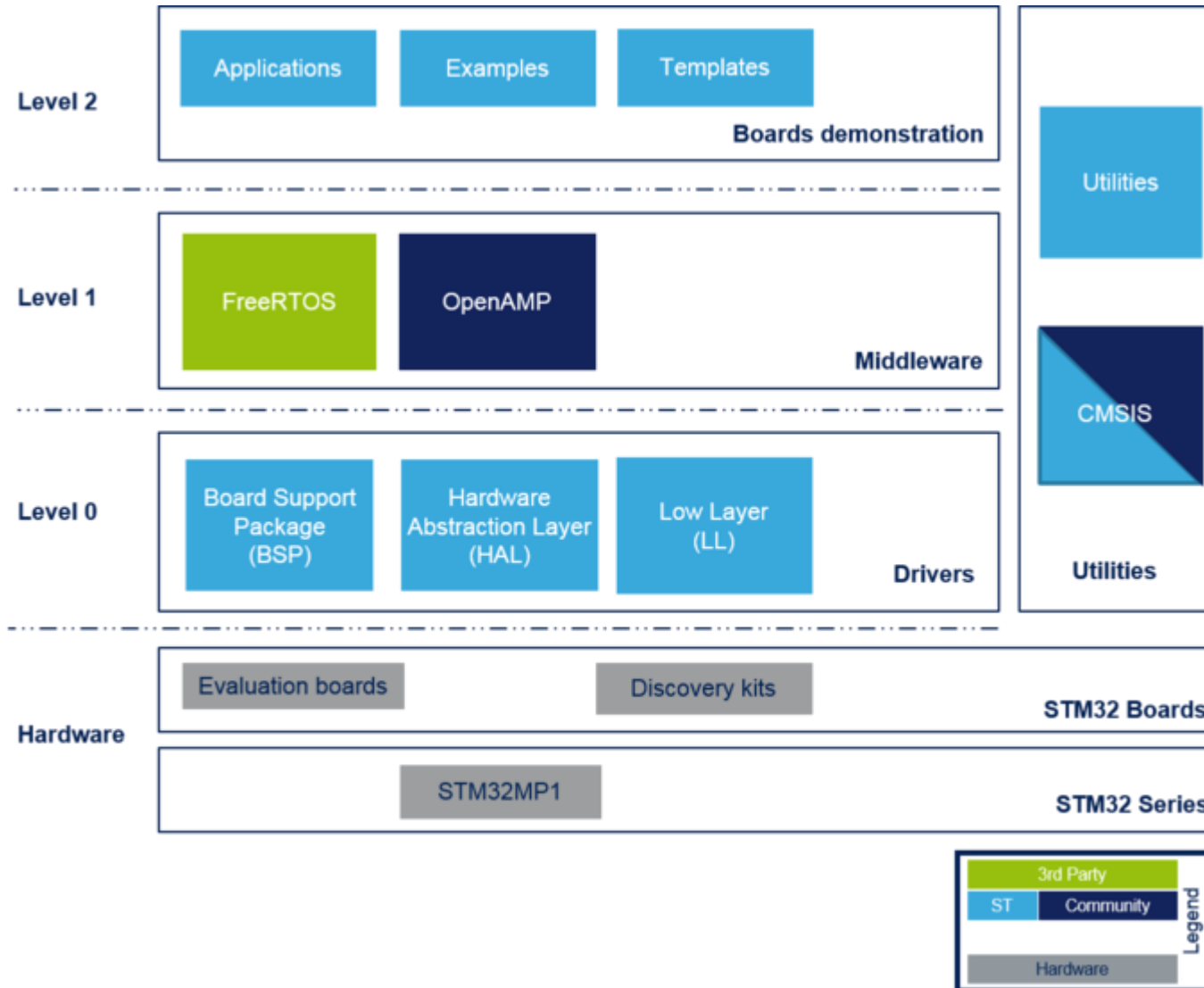


Develop on Arm® Cortex®-M4

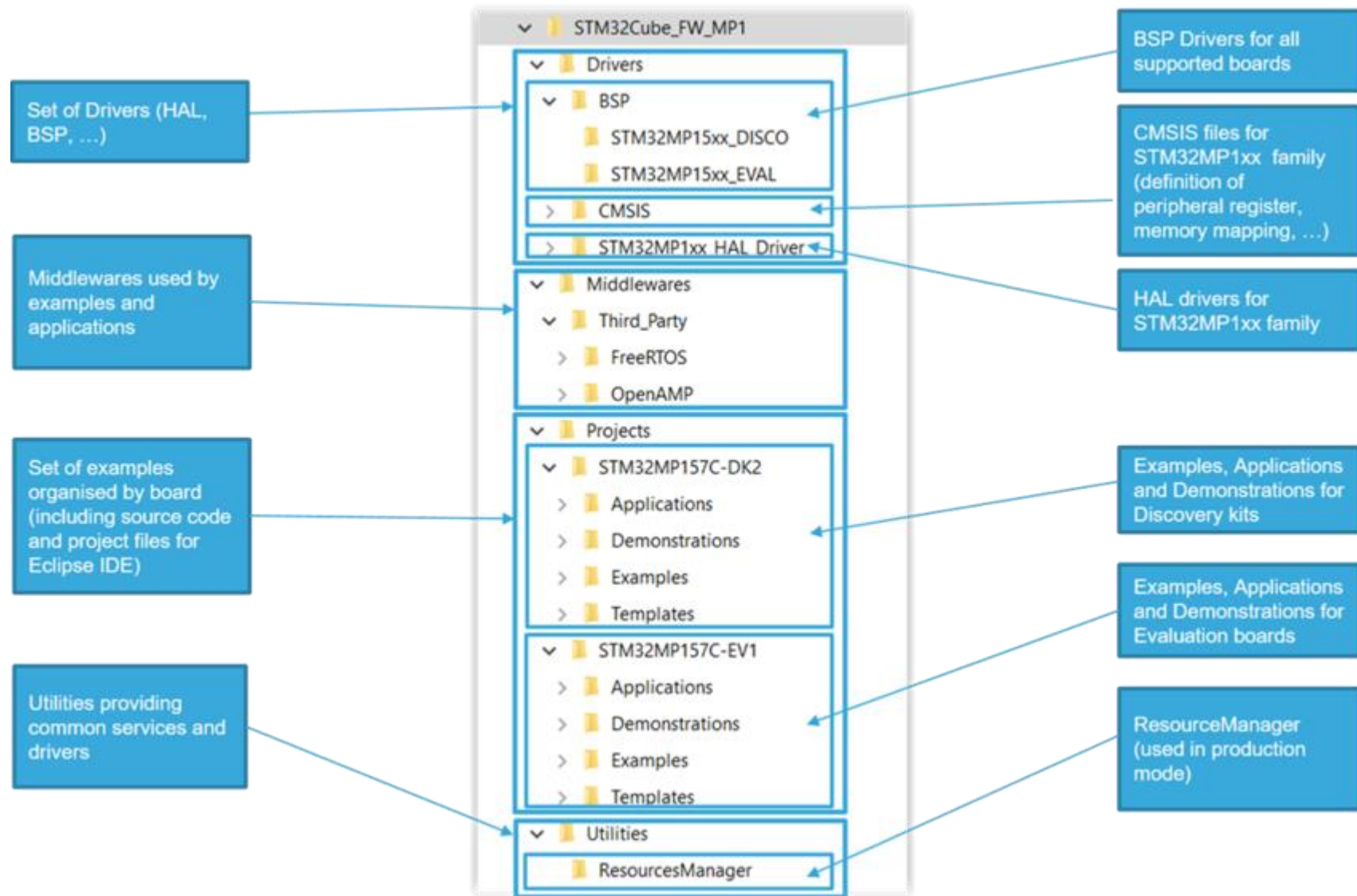


The STM32MP1 Developer Package contains (for the Arm® Cortex®-M4 side) the STM32Cube MPU Package containing all pieces of software (BSP, HAL, middlewares and applications) in source code.

STM32CubeMP1 architecture



STM32CubeMP1 source tree



M4 example source tree



M4 development steps

1 Create a project for your board:

- Use CubeMX to generate a chip/board based project
- Use IDE to generate an original project
- Import an existed project to the IDE

2 Adding middleware to your project (optional) :

- openAMP
- FreeRTOS

3 Configure the firmware components :

- xxx_conf.h

4 Start the HAL Library :

- HAL_Init()
 - SysTick
 - Set NVIC Group Priority
 - HAL_MspInit()

5 Configure the system clock (Engineering mode only) :

- HAL_RCC_OscConfig()
- HAL_RCC_ClockConfig()

6 Initialize the peripheral :

- HAL_PPP_MspInit()
- required interrupt handlers
- process complete callback functions

7 Develop your application :

- HAL APIs
- STM32CubeMP1 Package examples and applications

Engineering mode VS Production mode

Functions under *if(IS_ENGINEERING_BOOT_MODE())* :

- **System clock configuration** (*SystemClock_Config()*)
- **Clock source** (*HAL_RCCEx_PeriphCLKConfig()*)
- **openAMP**(*MX_OPENAMP_Init()*)

Functions need PERIPH_LOCK protection:

- **GPIO configuration** (*HAL_GPIO_Init/HAL_GPIO_DeInit()*)
- **EXTI configuration** (*HAL_EXTI_SetConfigLine()*)

Notice:

- Engineering mode is for debug only

STP32MP1 inter-processor communication



Agenda

1 Introduction

5 Data exchange

2 Hardware solution

6 Configuration

3 Software solution

7 References

4 Boot

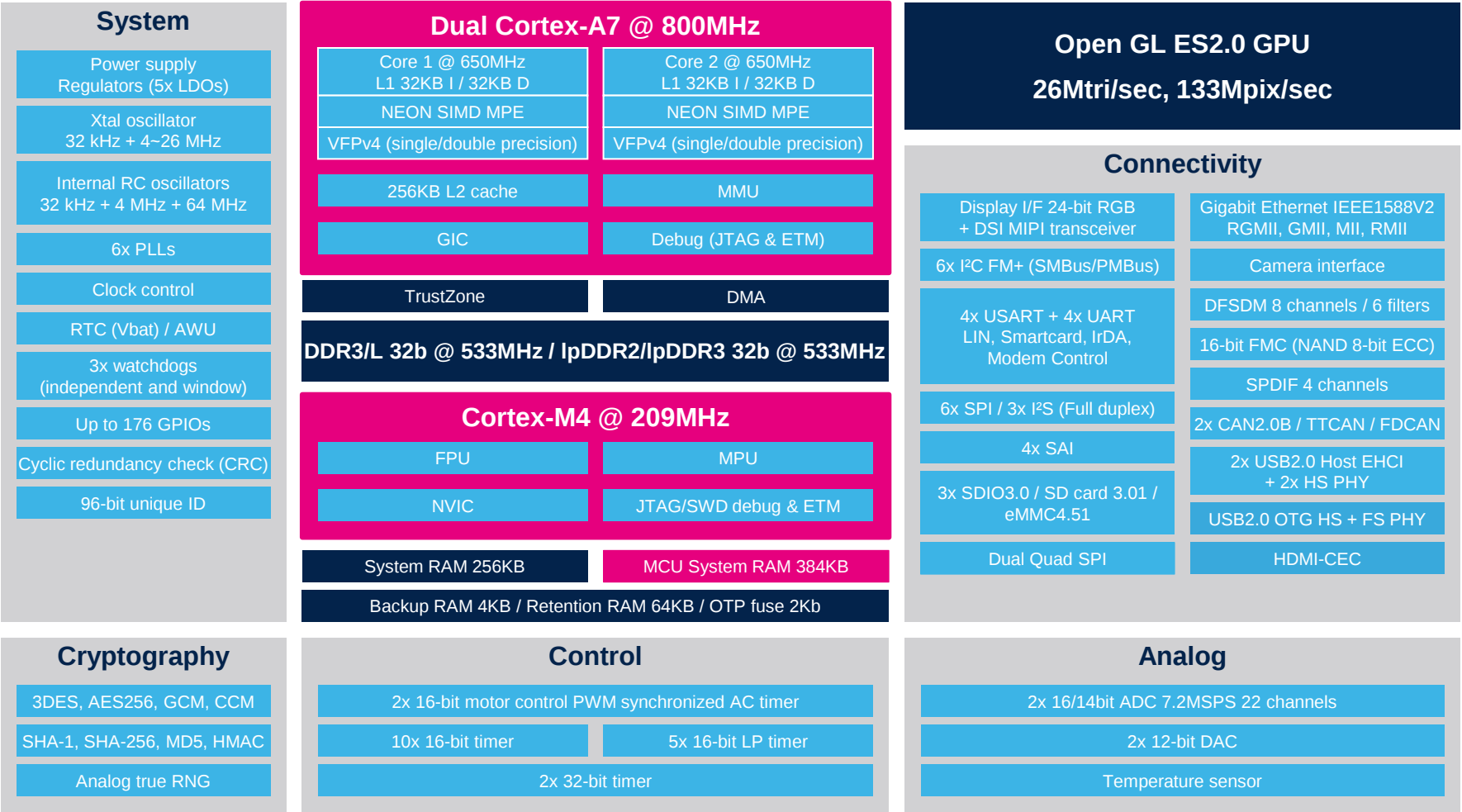
Introduction

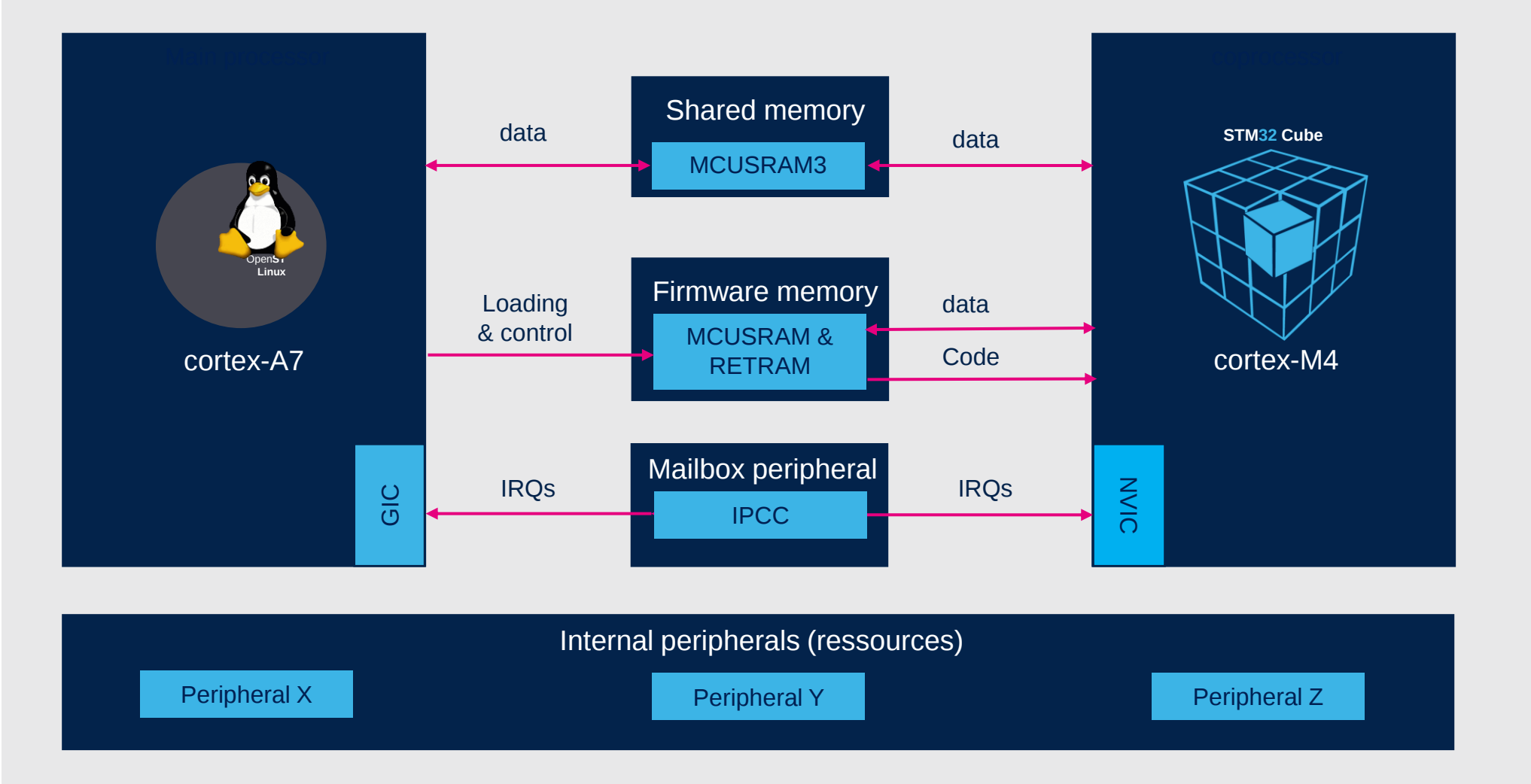
- Coprocessing management is a mechanism put in place to handle multicore in heterogeneous asymmetric architecture.
- On the STM32MP15x the Cortex®-A7 is the main processor (master) and the Cortex®-M4 is the coprocessor.
- In order to manage coprocessor, a list of services are needed :
 - Load and control Cortex®-M4 firmware, by Cortex®-A7.
 - Inter-processor communication based on messages and mailboxes.
 - Resources management:
 - Peripheral assignment to a Core (with access right check)
 - System resources management (resources that are common to both core)
- Feature can be enable/disable by declaring structures in **resource table**.

Hardware



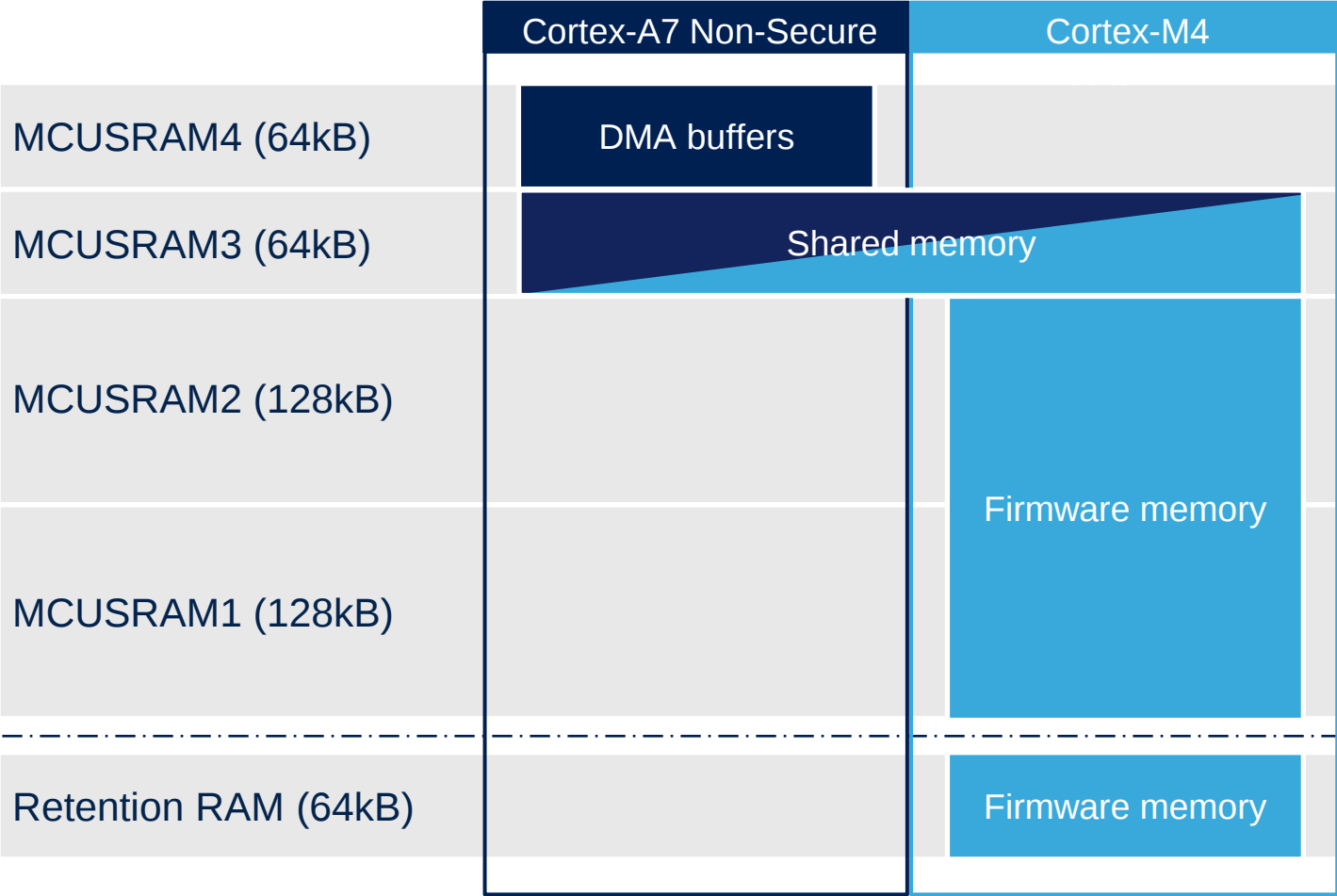
Hardware involved





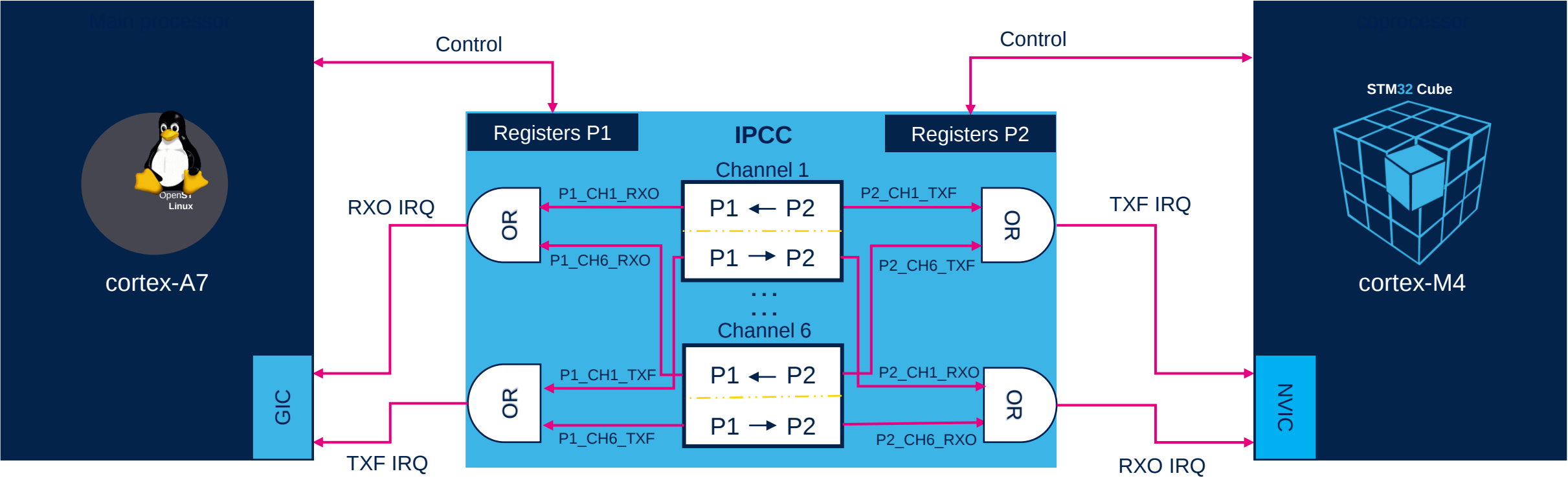
MCU RAM mapping

Cortex-M4 firmware will always start at the beginning of the RETRAM (address 0x0000 0000 for the cortex M4)



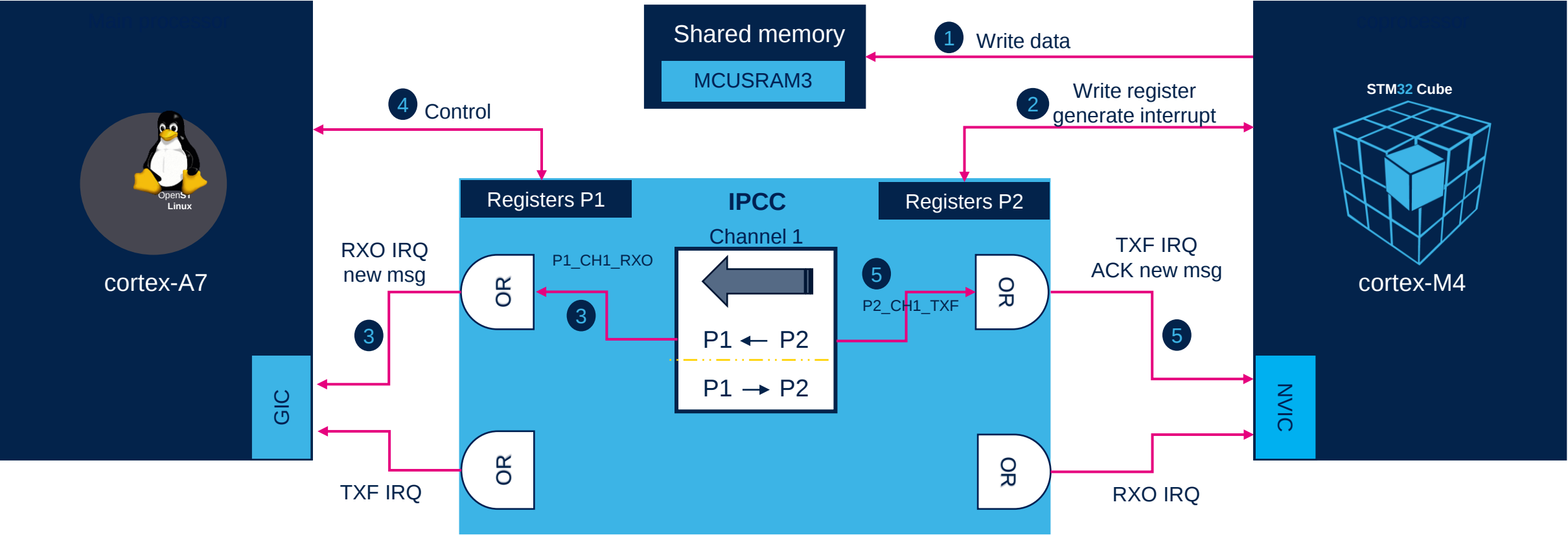
IPCC Signaling

- Inter-processor **signaling** is handled thanks to the Inter-Processor Communication Controller (IPCC).
- IPCC peripheral provides signaling for **6 bidirectional channels**.
- Each channel can be divided in 2 sub-signaling-channels, one for each direction :
 - P1_TO_P2 sub-signaling-channel
 - P2_TO_P1 sub-signaling-channel
- On Cortex-A, IPCC is controlled by mailbox software framework.
- On Cortex-M, IPCC is controlled by HAL_IPCC software driver.
- **2 channels are used** for the RPMsg protocol:
 - Channel 1: used for message from Cortex-M to Cortex-A (data in one direction, signaling in both direction).
 - Channel 2: used for message from Cortex-A to Cortex-M (data in one direction, signaling in both direction).
- **1 channel is used** for the Remote Proc protocol:
 - Channel 3: used to shutdown the Cortex-M from Cortex-A (unidirectional, simplex).



RXO : RX channel Occupied
TXF : TX channel Free
P1 ← P2 : Signaling direction, signals are going from P1 to P2

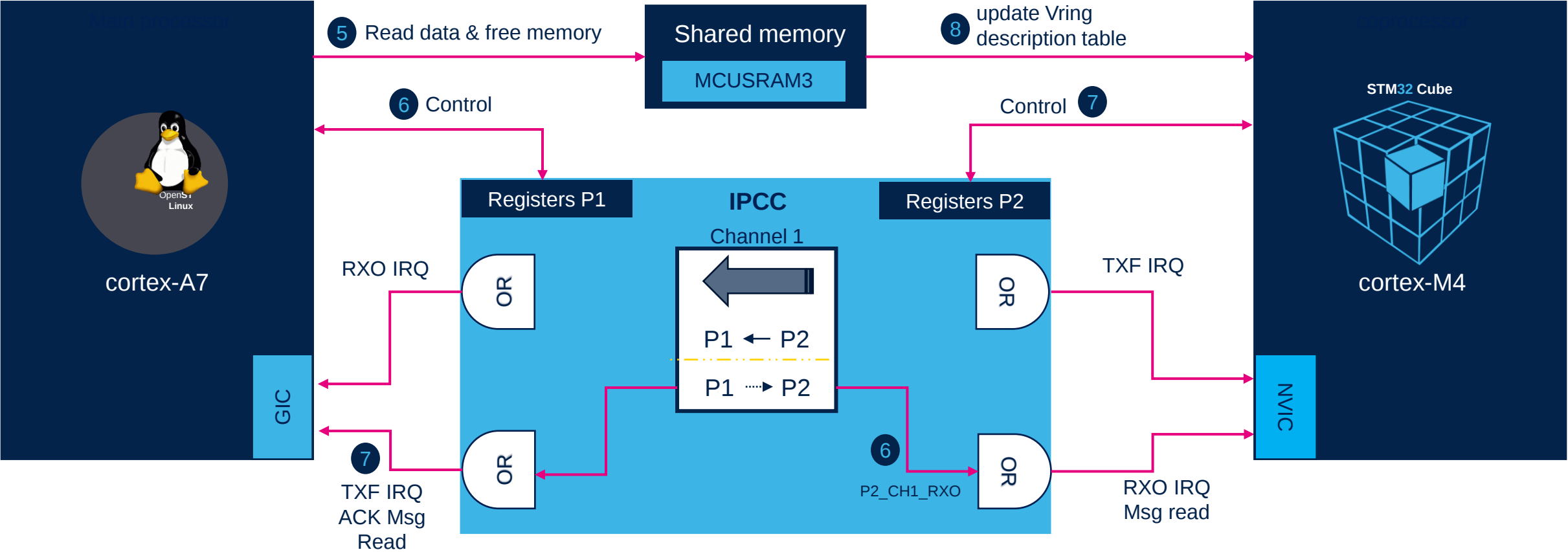
Example : data sent by M4



➡ : Data Flow

NB : same mechanism when data are sent by A7 using the channel 2 instead of channel 1

Example : data sent by M4



 : Data Flow

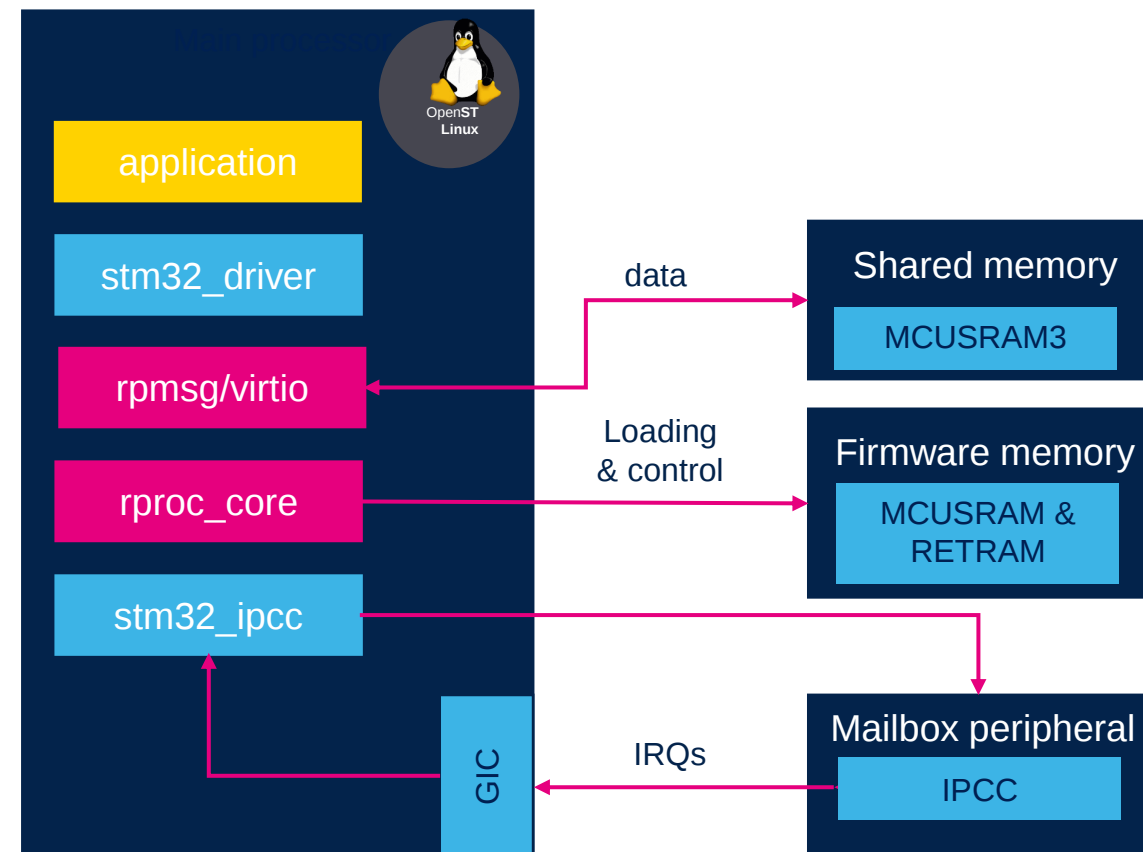
NB : same mechanism when data are sent by A7 using the channel 2 instead of channel 1

Software



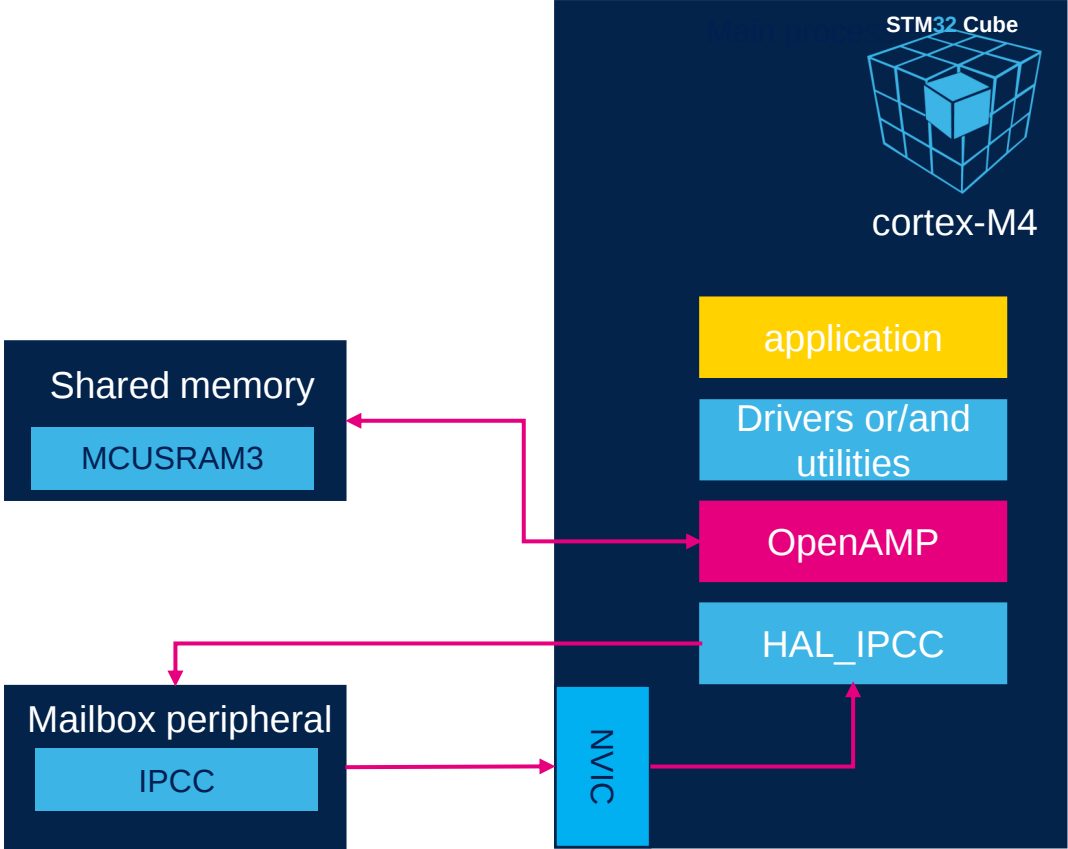
Software on Cortex-A7

- Linux OS integrates frameworks that allow to control and communicate with remote processor:
 - RemoteProc** - The generic Remote Processor framework allows to control (power on, load firmware, power off) remote processors.
 - RPMsg** - the Remote Processor Messaging is a VirtIO-based messaging bus that allows kernel drivers to communicate with remote processors available on the system.
 - VirtIO** - VirtIO framework that supports virtualization. It provides an efficient transport layer based on shared ring buffer (Vring).



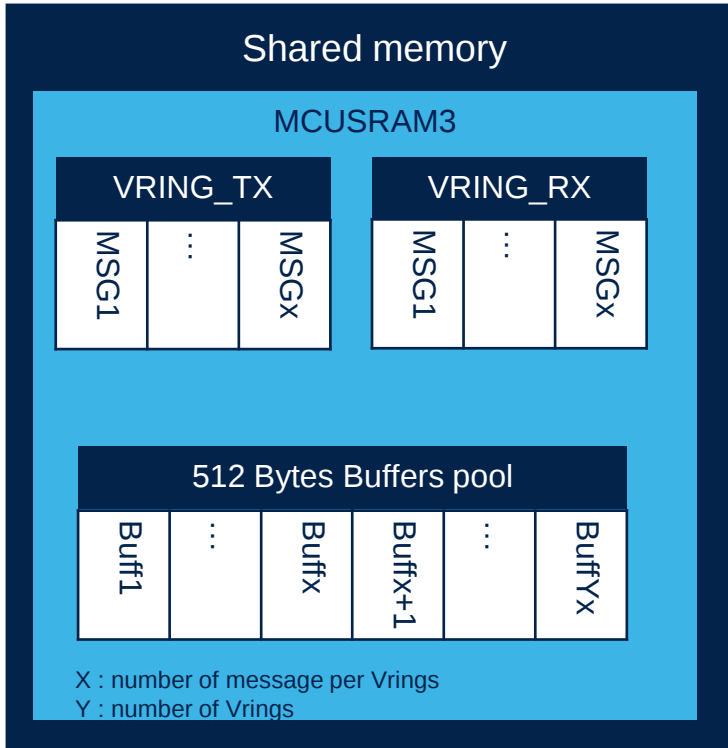
Software on Cortex-M4

- **OpenAMP** is a library integrating RemoteProc, Virtio and RPMsg frameworks for baremetal or RTOS running on Cortex-M4.
- Official website :
 - <https://www.openampproject.org/>
 - <https://github.com/OpenAMP/open-amp>
- From [openampproject.org](https://www.openampproject.org/) :
 - *OpenAMP (Open Asymmetric Multi-Processing) seeks to standardize the interactions between operating environments in a heterogeneous embedded system through open source solutions for Asymmetric MultiProcessing (AMP).*

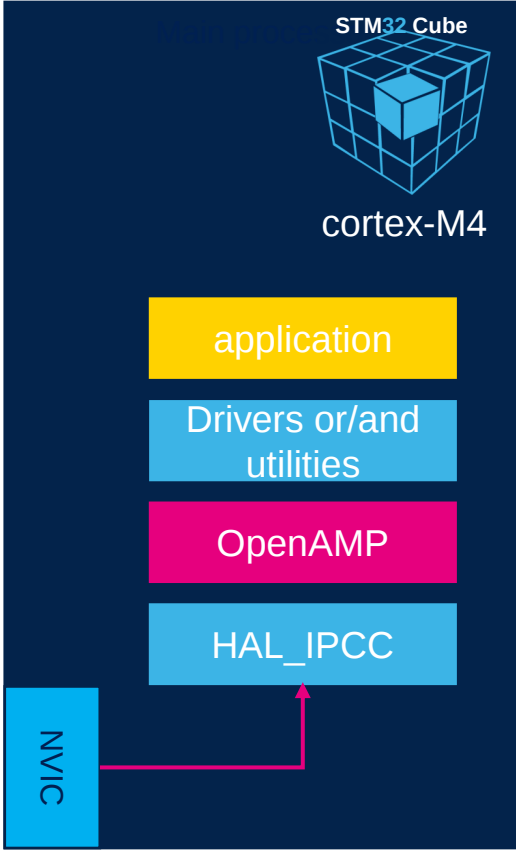
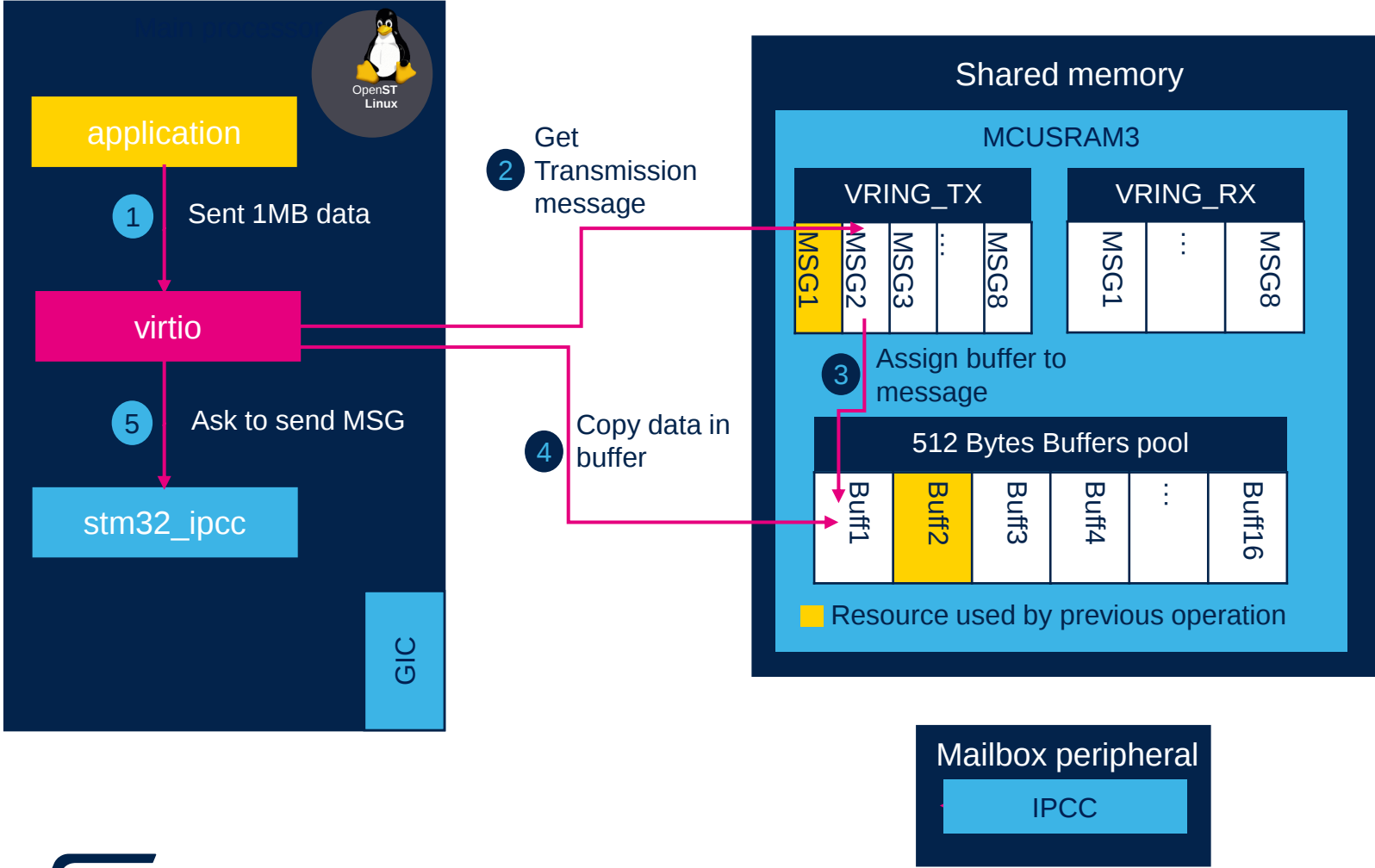


Shared memory structure (Virtio)

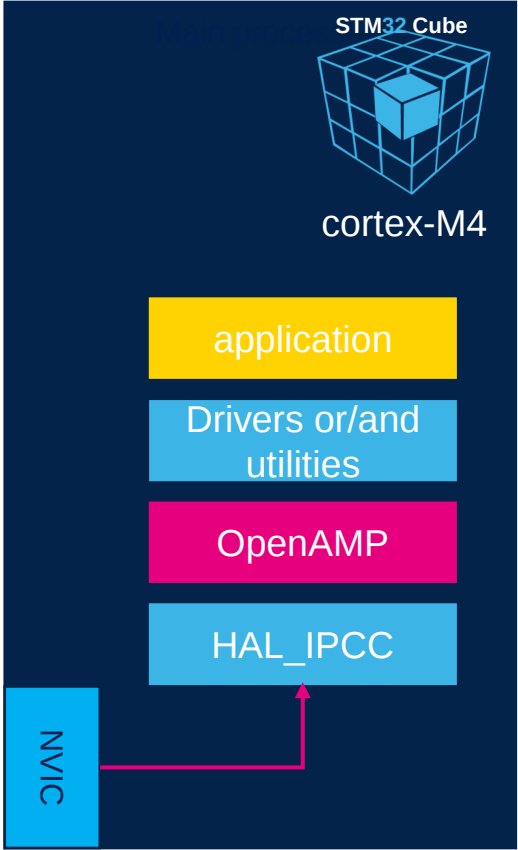
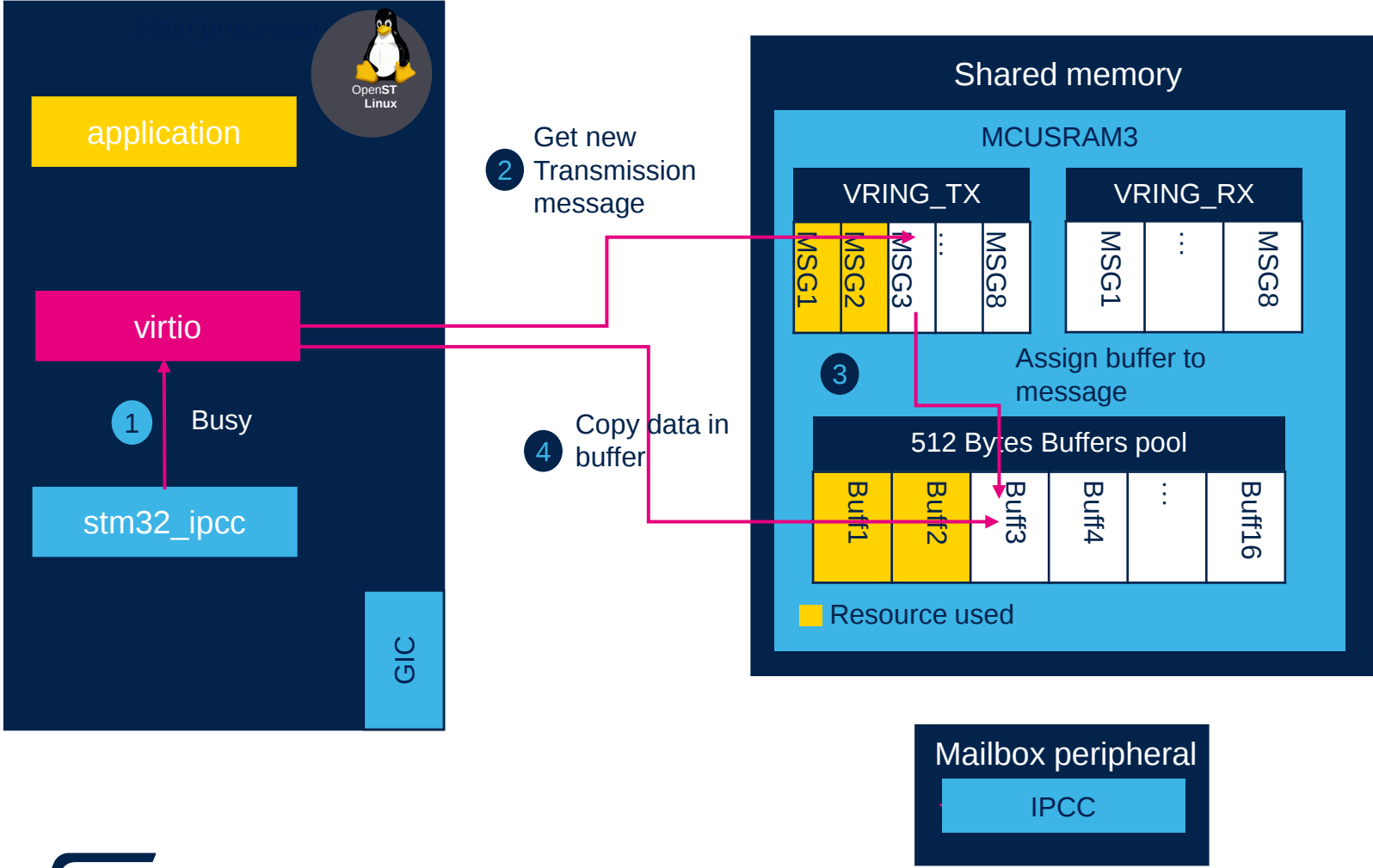
- VirtIO Vrings mechanism is used to send/receive messages to/from the remote CPU over shared memory (MCUSRAM).
- Vring is unidirectional
- Two Vrings are used in the STM32MP15x implementation :
 - One to send messages to remote processor, Vring_Tx.
 - One to receive message from the remote processor, Vring_Rx.
- A shared pool of buffers is created in memory space visible by both:
 - Number of message per Vring is customizable (max: 512, MP15x: 8)
 - Size of the message buffer is fixed to **512 Bytes**



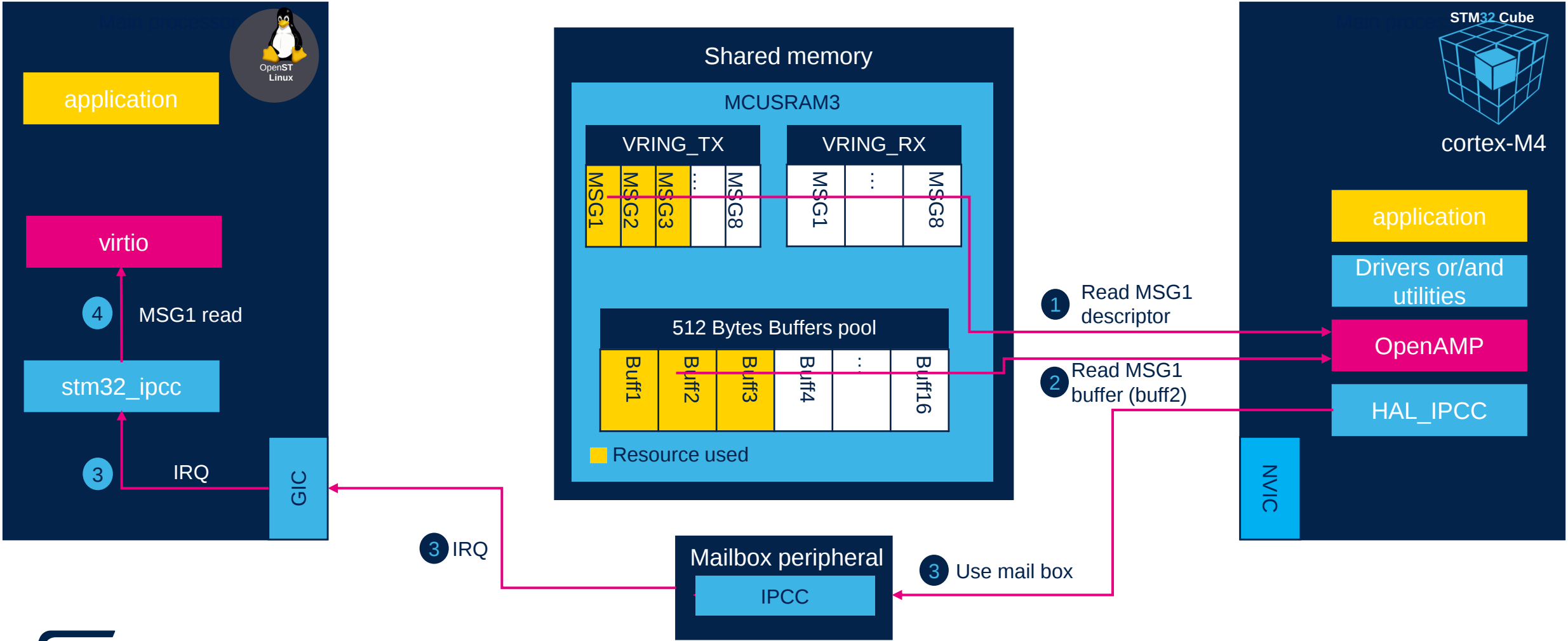
Virtio example



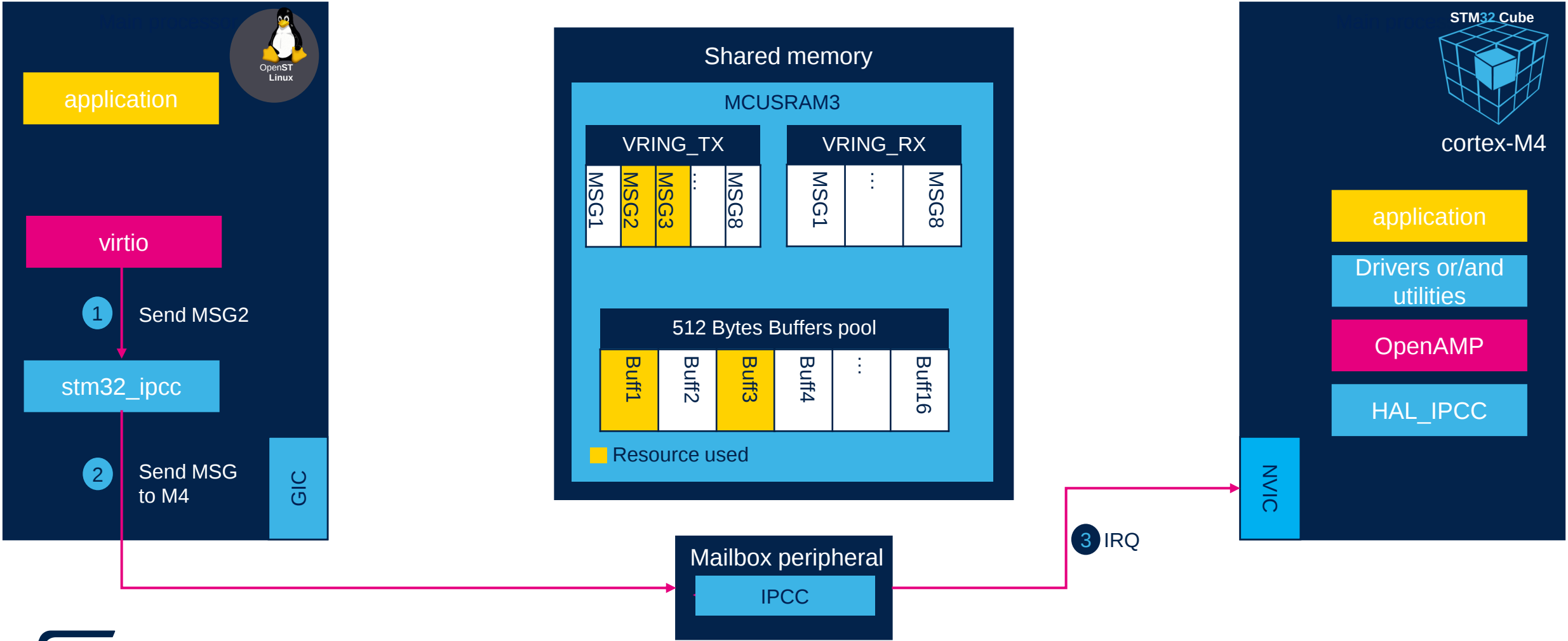
Virtio example



Virtio example

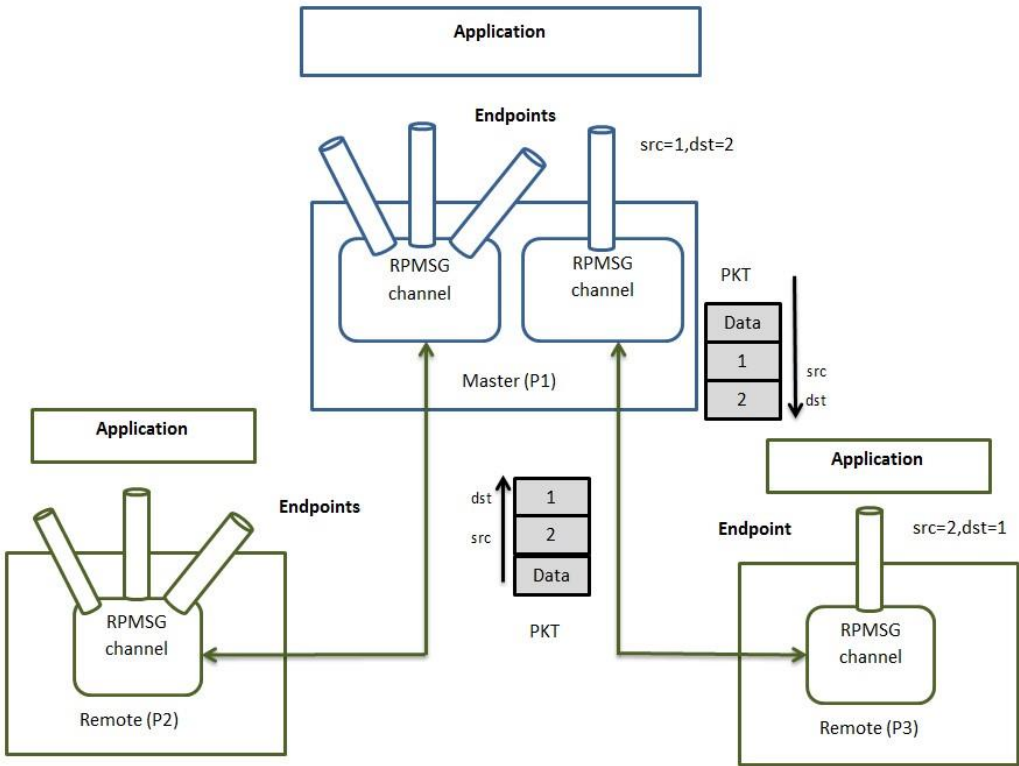


Virtio example



RPMMsg: Channel & endpoint

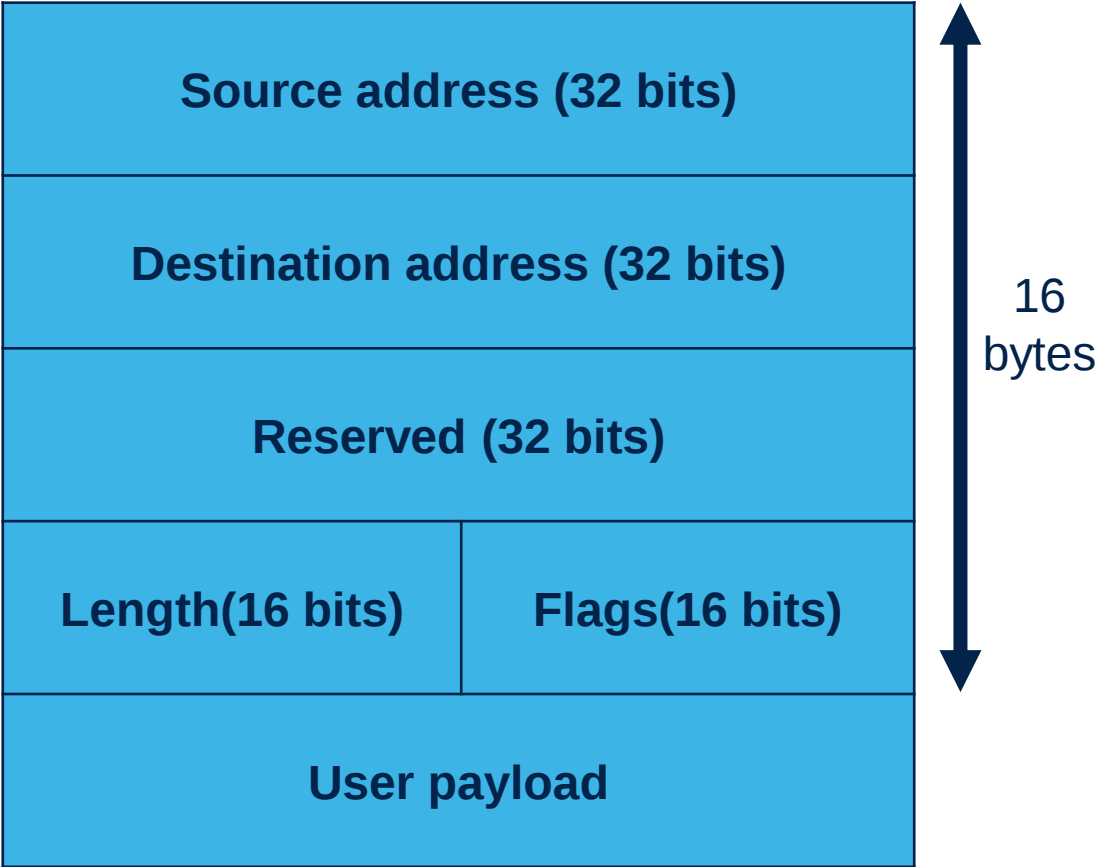
- A RPMMsg client is associated to a communication channel between master and remote processor. RPMMsg client is identified by the textual **service name**, registered in the RPMMsg framework. The communication channel is established when a match is found between the local service name registered and the remote service announced.
- The RPMMsg endpoints provide logical connections on top of RPMMsg channel. A RPMMsg endpoint has a unique address. Notice that every channel has a default endpoint which enables applications to communicate without even creating new endpoints.



RPMsg : Message

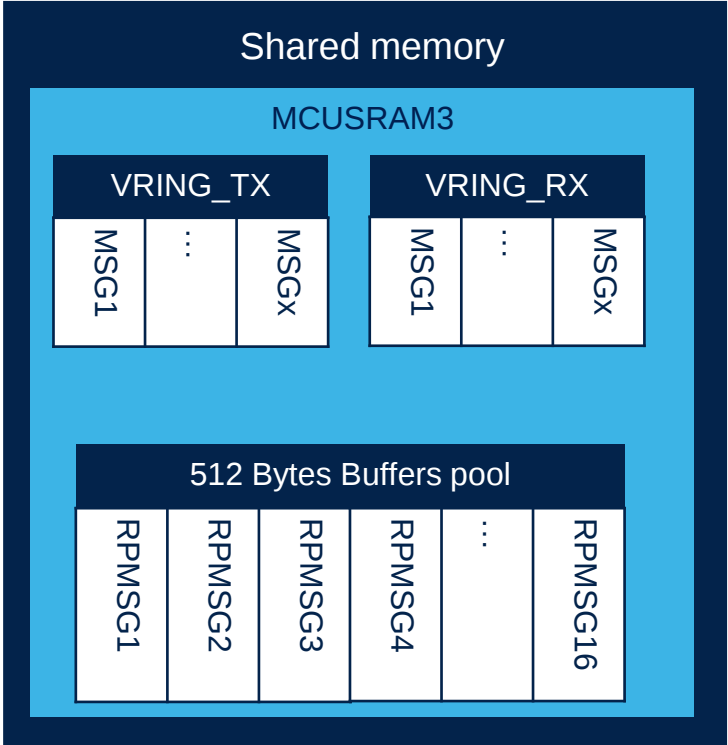
- The RPMsg header has a size of 16 bytes. And must be located at the begining of the RPMsg.
- The first word (32bits) is used as an address of the sender or source endpoint, next word is the address of the receiver or destination endpoint.
- There is a reserved field for alignment reasons (RPMsg header is thus 16 bytes aligned).
- Last two fields of the header are the length of the payload (16bit) and a 16-bit flags field.
- The user payload follows the RPMsg header

N.B.: The reserved field is not used to transmit data between cores and can be used internally in the RPMsg implementation.



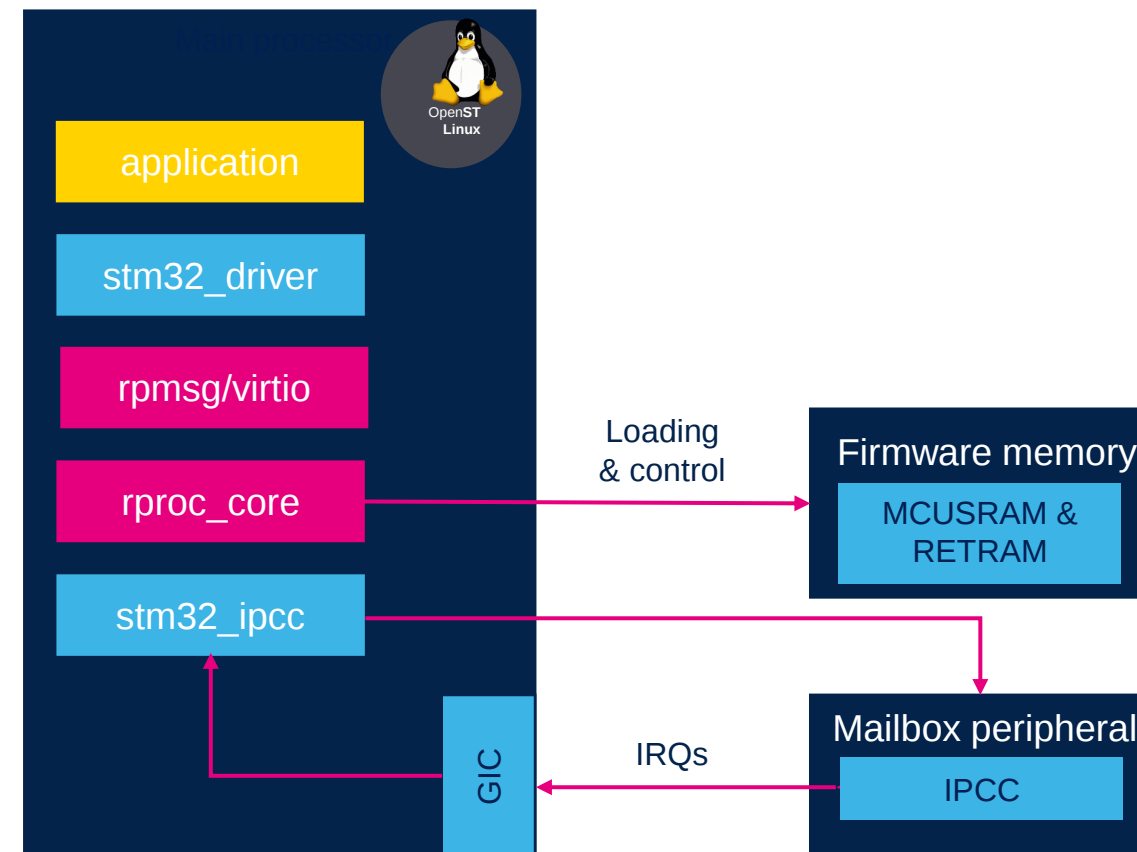
RPMMsg/Virtio: STM32MP15x

- Only one channel will be used inside the STM32MP1 as there is one CoProcessor (cortex-M4)
- Several End-points can be managed (example of the big data transfer).
- RPMMsg structure are stored in the shared memory (MCUSRAM3).One RPMSG per 512 Bytes buffer.



- Remote Proc Framework will provide the low level interface to the Virtio/RPMSG framework.
- Remote Proc Framework will also provide interface to load, start and stop the coprocessor.
- IPCC Channel 3 is used on remote Proc shutdown process.

RemoteProc Framework



Resource table

The resource table is a structure declared in the coprocessor firmware. This table contains resources that the remote and the master processors need before to powered on the remote one.

This table must be defined in a specific data section of the coprocessor firmware, parsed by the RemoteProc Linux framework during the firmware load phase to:

- Allocate memories defined in the resource table carveout section (not used in the ST implementation).
- **Load the RPMsg and Virtio frameworks to support messaging services (if not define in the table this service will be not used by Linux).**
- **Provide a user sysfs interface to access coprocessor traces for debug.**

Resource table

In the STM32MCU cube firmware package, the resource table is defined in rsc_table.c.

For the Cortex-M firmware that does not require interaction with the Linux OS, a default structure must be declared in the Cortex-M firmware.

```

struct remote_resource_table __resource __attribute__((used)) rproc_resource = {
    .version = 1,
    .num = 0,
    .reserved = {0, 0},
    .offset = { 0 },
};
    
```

Resource table

Activate the RPMG/Virtio communication between A7 and M4.

```
#define NUM_VRINGS 0x02 /* number of Vring used , must be fixed to 2 (one for TX one for RX) */
#define VRING_ALIGN 0x1000 /* must be fixed to 0x1000 (Linux constraint) */
#define VRING_TX -1 /* allocated by master processor */
#define VRING_RX -1 /* allocated by master processor */
#define VRING_SIZE 8 /* number of 512 bytes buffers associated to a Vring: can be customized */

struct remote_resource_table __resource __attribute__((used)) rproc_resource = {
    .version = 1,
    .num = 1, /* rely to number of offsetof structures declared in .offset */
    .reserved = {0, 0},
    .offset = {
        offsetof(struct remote_resource_table, rpsmsg_vdev),
    },
    /* Virtio device entry */
    .rpsmsg_vdev = {
        RSC_VDEV, VIRTIO_ID_RPSMSG, 0, RPSMSG_IPU_C0_FEATURES, 0, 0, 0,
        NUM_VRINGS, {0, 0},
    },
    /* Vring rsc entry - part of vdev rsc entry */
    .rpsmsg_vring0 = {VRING_TX, VRING_ALIGN, VRING_SIZE, 1, 0},
    .rpsmsg_vring1 = {VRING_RX, VRING_ALIGN, VRING_SIZE, 2, 0},
};
```

Resource table

Activate the log buffer exchange between A7 and M4.

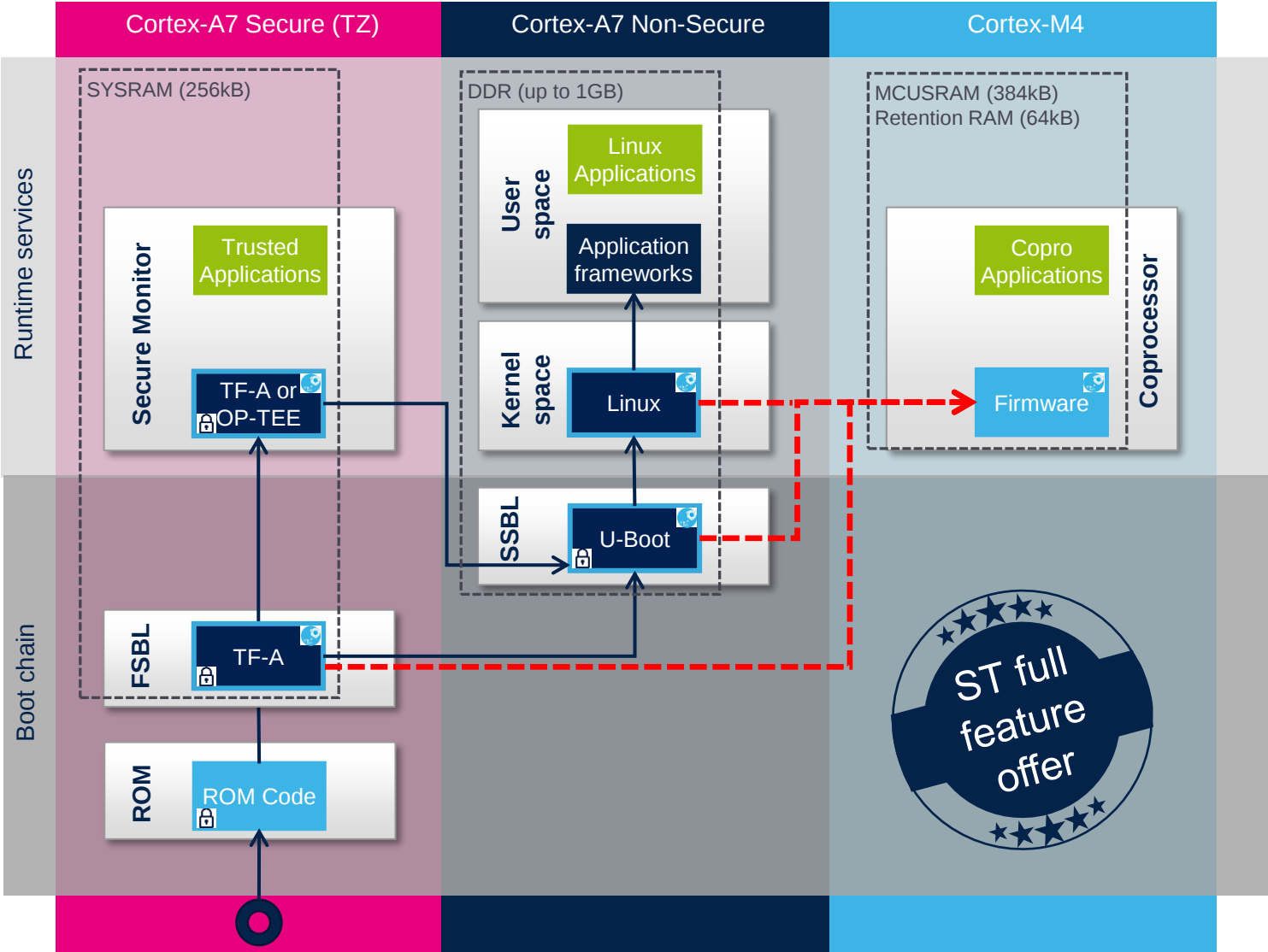
```
const struct shared_resource_table __resource __attribute__((used)) resource_table = {
    .version = 1,
    .num = 2, /* one for RPMSG, one for log */
    .reserved = {0, 0},
    .offset = {
        offsetof(struct shared_resource_table, vdev), /* RPMSG */
        offsetof(struct shared_resource_table, cm_trace), /* log */
    },
    ..... /* RPMSG config see previous slide */
    .cm_trace = {
        RSC_TRACE,
        (uint32_t)system_log_buf, SYSTEM_TRACE_BUF_SZ, 0, "cm4_log",
    },
};
```

Boot



The coprocessor firmware can be loaded and started by :

- TF-A
- the second stage bootloader, U-Boot.
- the Linux kernel (Thanks to the remoteProc)



M4 boot : firmware location

- TF-A boot :

The file format is a ELF file with a STM32 header.

For Emmc and SDmmc:

The firmware is stored in a GPT partition named “m4fw”.

For NOR:

The M4 firmware need to be set at a specific offset : “0x001800000”

For NAND:

The M4 firmware need to put in a specific MTD named : “m4fw”

- U-boot M4 boot :

The file format is an elf stored in the bootfs partition.

- Linux M4 boot:

By default Linux is looking for an elf file in /lib/firmware/

Flash type	TF-A boot M4	U-boot boot M4
Sdcard	750ms	6.93s
EMMC	450ms	4.55s
Raw Nand	350ms	10.8s
Nor flash	350ms	4.4s
Serial Nand	350ms	

Data exchange



Data exchange

The RPMsg provides, through the virtio framework, a basic transport layer based on a shared ring buffer:

- The buffers are prenegociated and preallocated during coprocessor loading (size , number of buffers).
- The buffers are allocated in non cacheable memory.
- There is no direct access from RPMsg client to these buffers. They are filled by a copy (no zero copy or DMA transfers).
- No bandwidth is guaranteed. Buffers can be shared between several RPMsg clients.
- The size of the buffers is hard-coded (512 bytes). However it is possible to customize the number of buffers used. Modifying this parameter impacts the number of buffers allocated in both direction.

The IRQ frequency can be a criteria for the decision. For instance, a 1 Mbyte/s transfer from Cortex-M4 to Cortex-A7 generates around 2000 IRQ per second on each Cortex, to transfer 512-bytes RPMsg buffers.

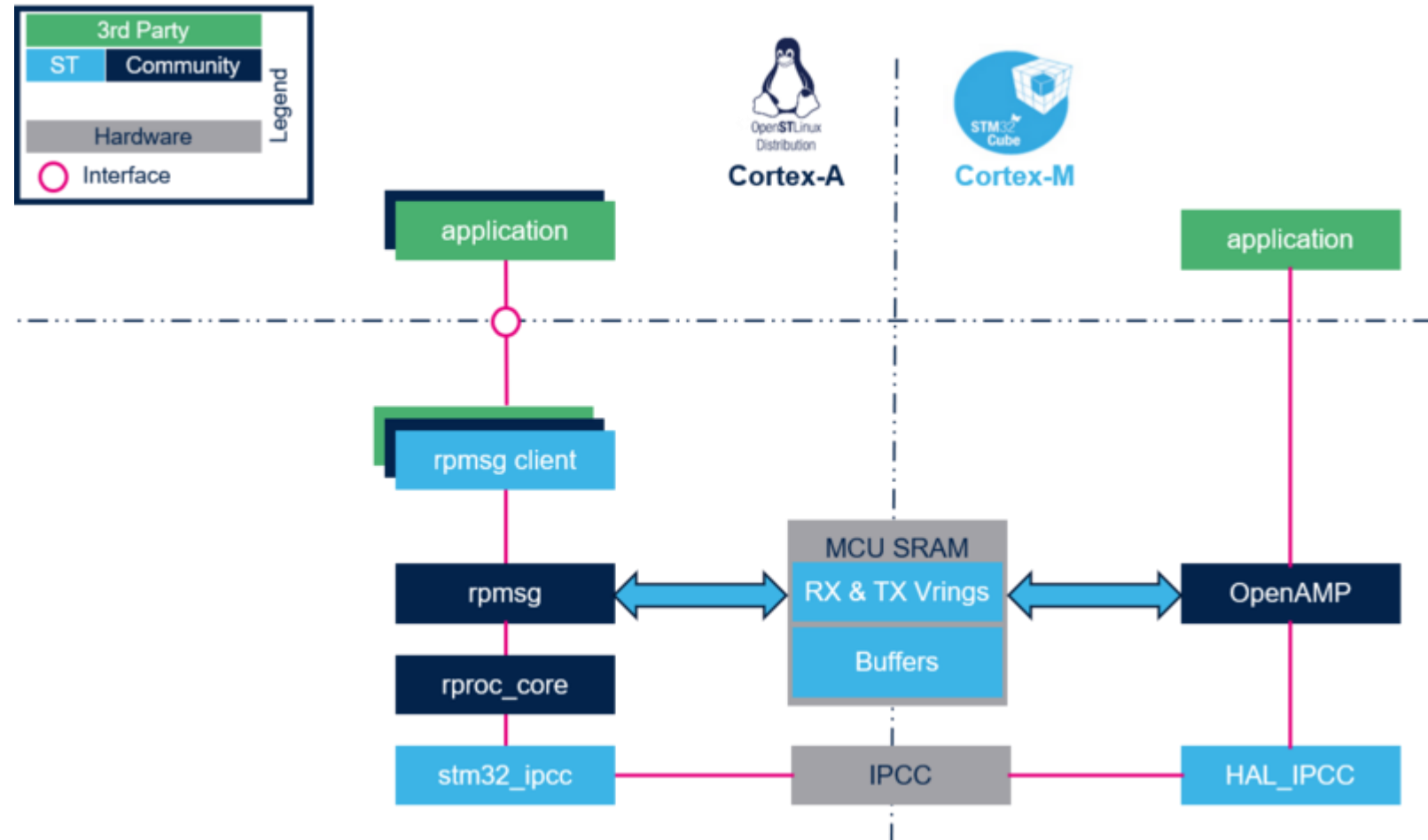
Due to these limitations ST provide two ways to exchange buffers between A7 and M4 :

- direct buffer using RPMSG/Virtio to transfer Data
- Indirect buffer using DMA to transfer data

Data exchange : direct buffer

The Direct buffer exchange implementation is recommended:

- for control message, for instance to control a remote processor application,
- to exchange low data rate stream (similar to a slow data rate bus).

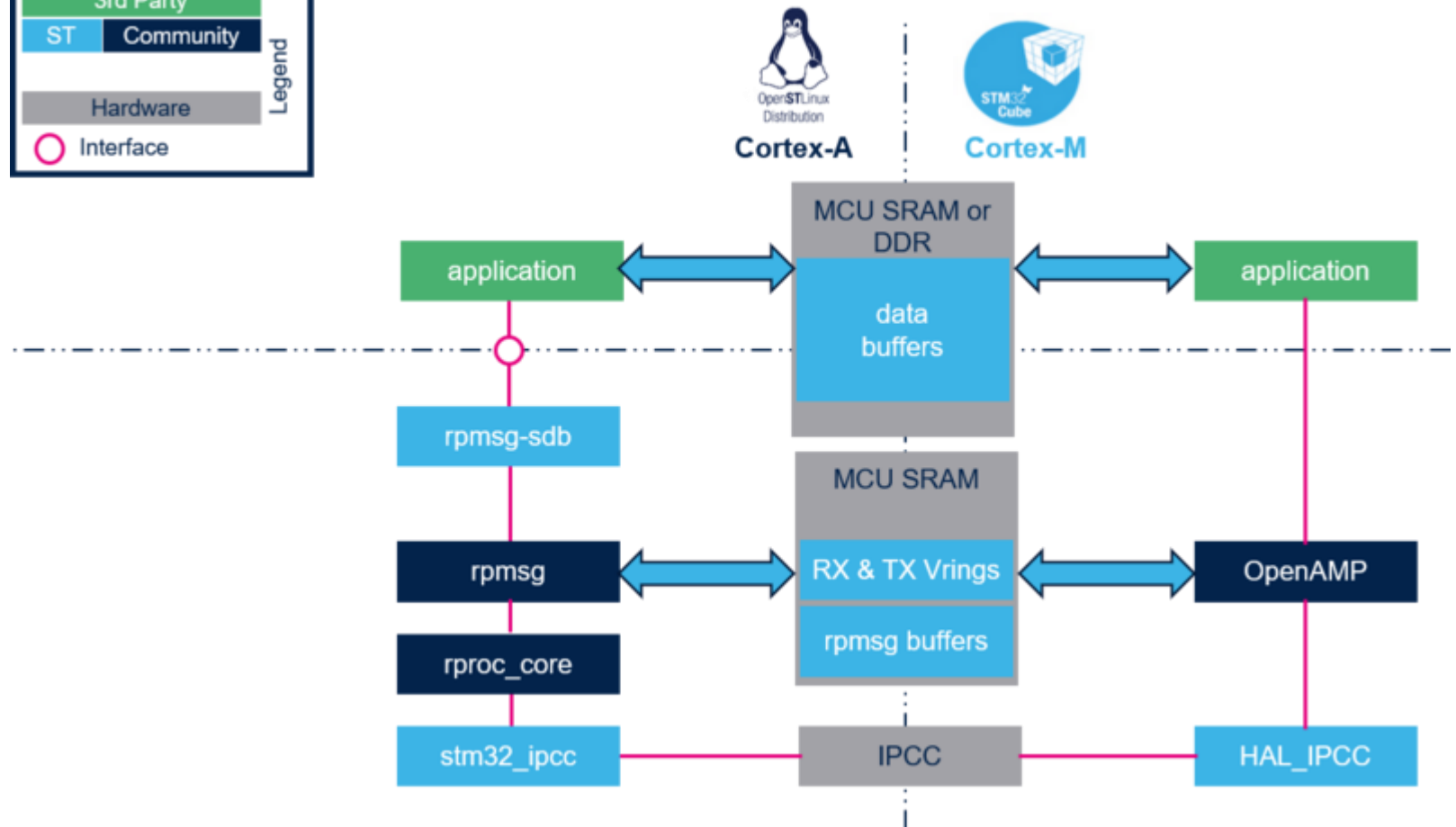
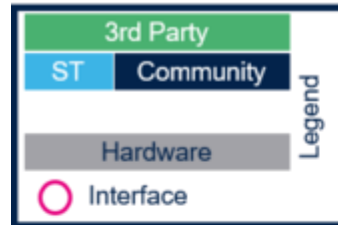


Data exchange : indirect buffer

The indirect buffer exchange implementation is recommended:

- for high bit rate transfer,
- for real time transfer (e.g. audio buffers)
- to privilege dynamic buffers allocation and/or minimize copies.
- to adapt to existing Linux framework or application

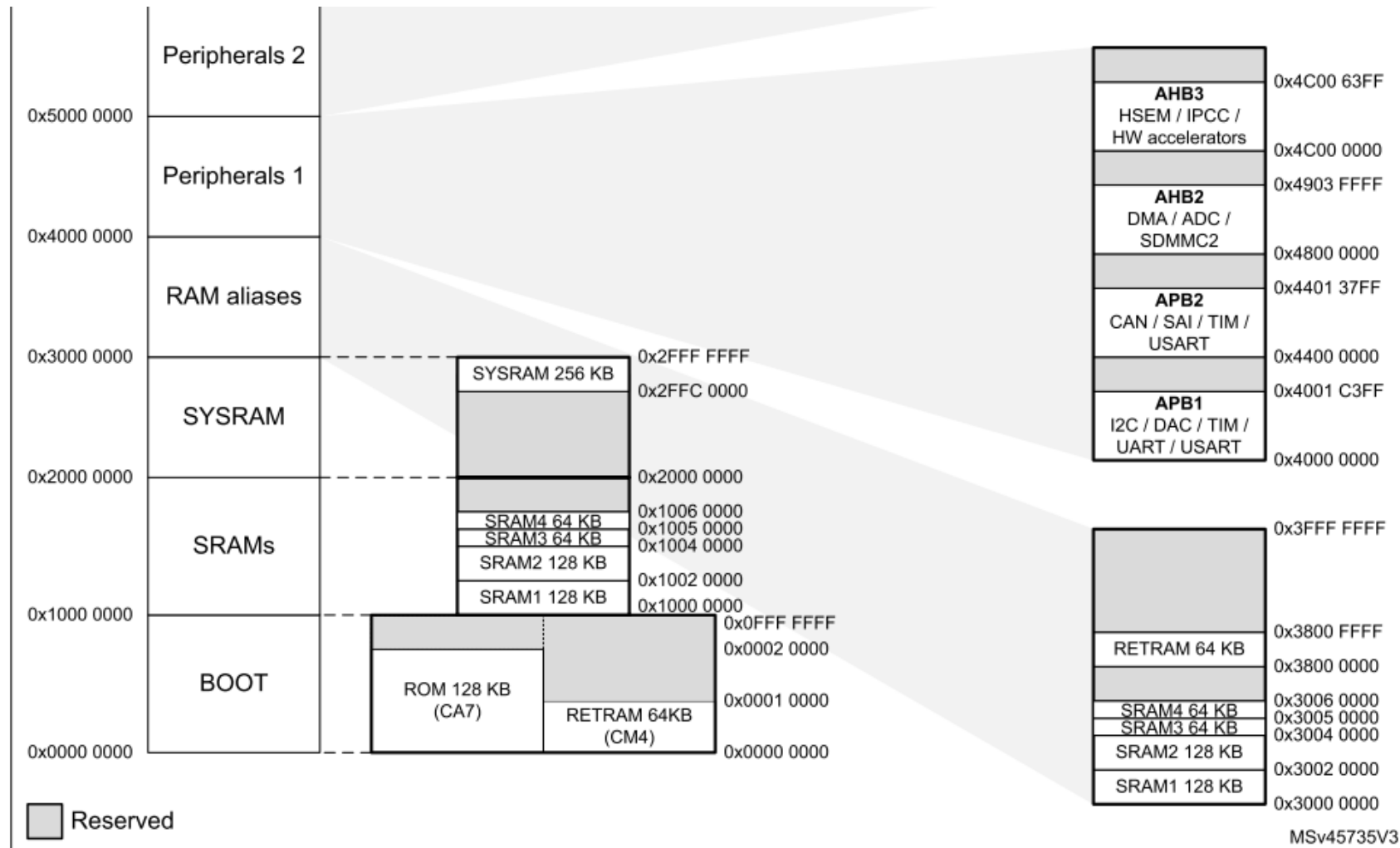
The data flow today is unidirectional M4 → A7



Configuration



Memory mapping



configuration

- m4_rproc: m4@10000000 {
- compatible = "st,stm32mp1-m4";
- reg = <0x10000000 0x40000>,
- <0x30000000 0x40000>,
- <0x38000000 0x10000>;
- resets = <&scmi0_reset RST_SCMI0_MCU>;
- st,syscfg-holdboot = <&rcc 0x10C 0x1>;
- st,syscfg-tz = <&rcc 0x000 0x1>;
- st,syscfg-rsc-tbl = <&tamp 0x144 0xFFFFFFFF>;
- st,syscfg-copro-state = <&tamp 0x148 0xFFFFFFFF>;
- st,syscfg-pdds = <&pwr_mcu 0x0 0x1>;
- status = "disabled";

configuration

- &m4_rproc {
- memory-region = <&retram>, <&mcuram>, <&mcuram2>, <&vdev0vring0>,
- <&vdev0vring1>, <&vdev0buffer>;
- mboxes = <&ipcc 0>, <&ipcc 1>, <&ipcc 2>;
- mbox-names = "vq0", "vq1", "shutdown";
- interrupt-parent = <&exti>;
- interrupts = <68 1>;
- wakeup-source;
- status = "okay";

- mcuram2: mcuram2@10000000 {
- compatible = "shared-dma-pool";
- reg = <0x10000000 0x40000>;
- no-map;
- };
- vdev0vring0: vdev0vring0@10040000 {
- compatible = "shared-dma-pool";
- reg = <0x10040000 0x1000>;
- no-map;

References



Coprocessor Management:

https://wiki.stmicroelectronics.cn/stm32mpu/wiki/Category:Coprocessor_management_STM32Cube

Wiki Android how to :

https://wiki.stmicroelectronics.cn/stm32mpu/wiki/Category:How_to_Android

Thank you