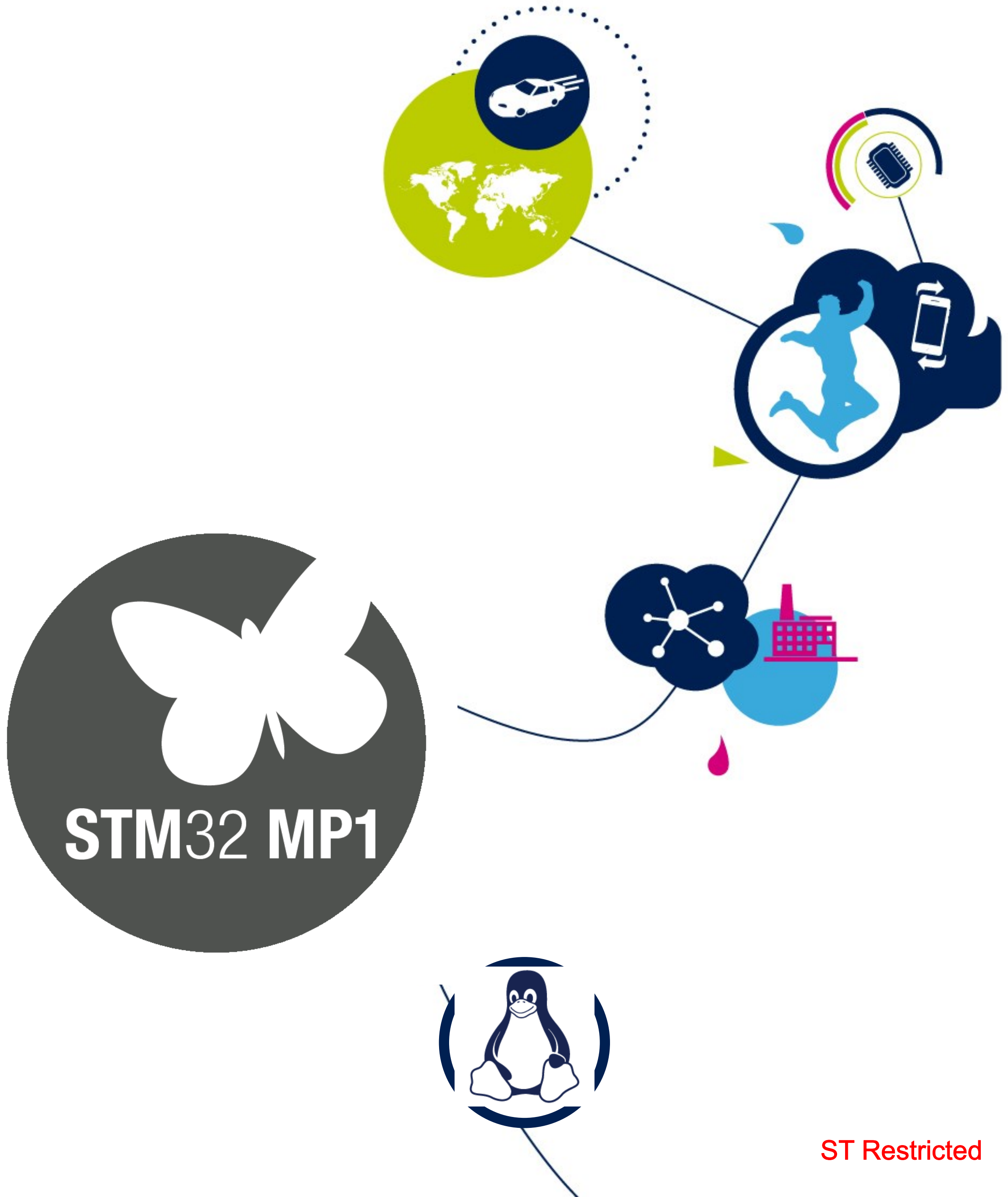
STM32MP1

Boot chain and security



life.augmented

Agenda

Presentation

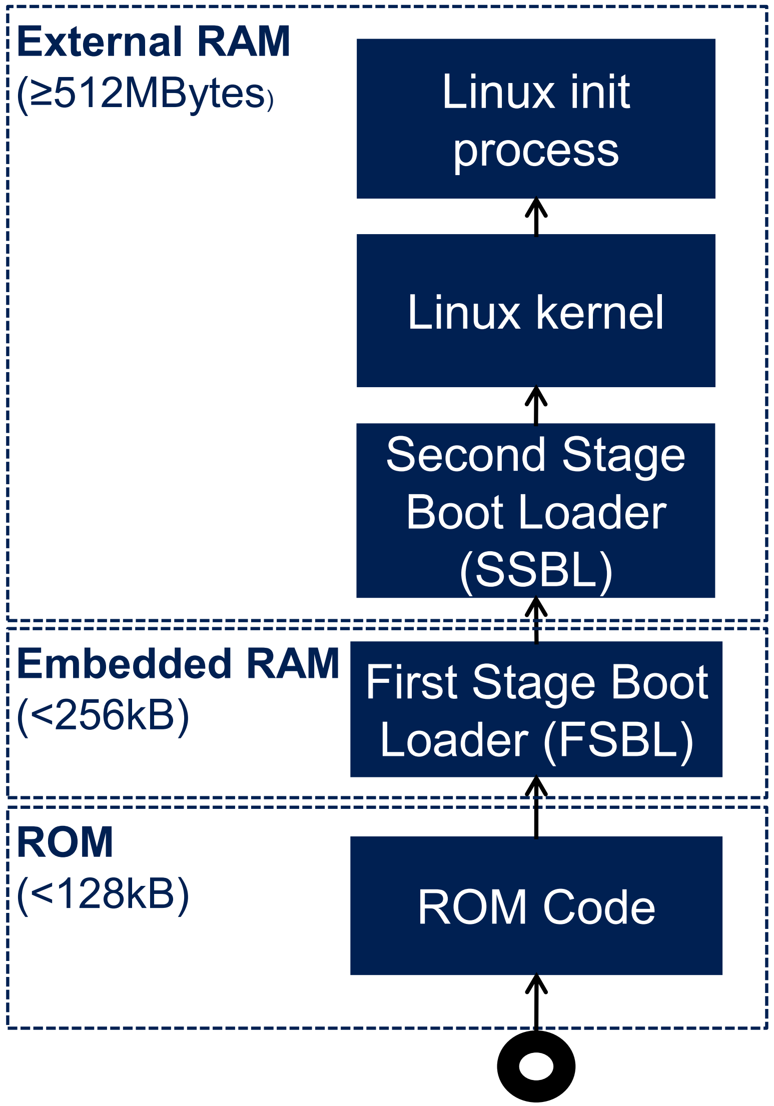
40min

- Boot chain
- Security on STM32MP1

Boot chain overview

Linux boot loader

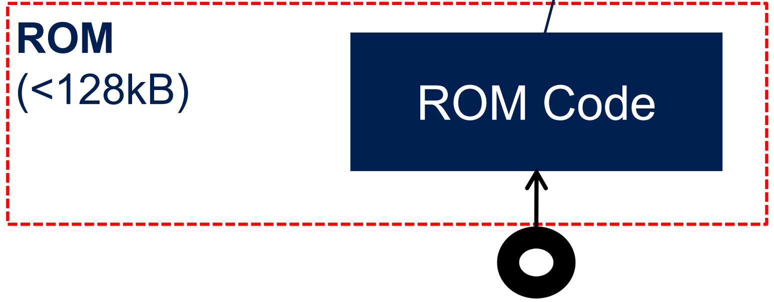
Standard Linux Boot Chain



life.augmented

Linux boot loader

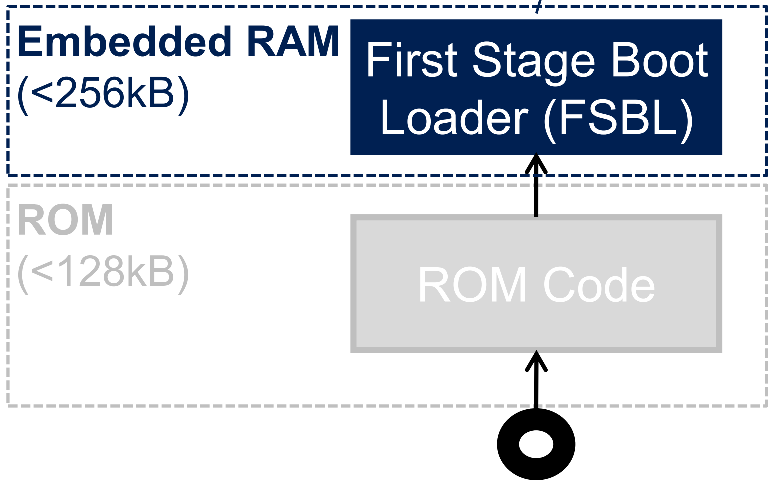
Standard Linux Boot Chain



ROM code :
- **Basic clock tree** initialization
- **FSBL** Int-RAM loading from the **boot device** (mass storage or serial link)
- FSBL launch

Linux boot loader

Standard Linux Boot Chain

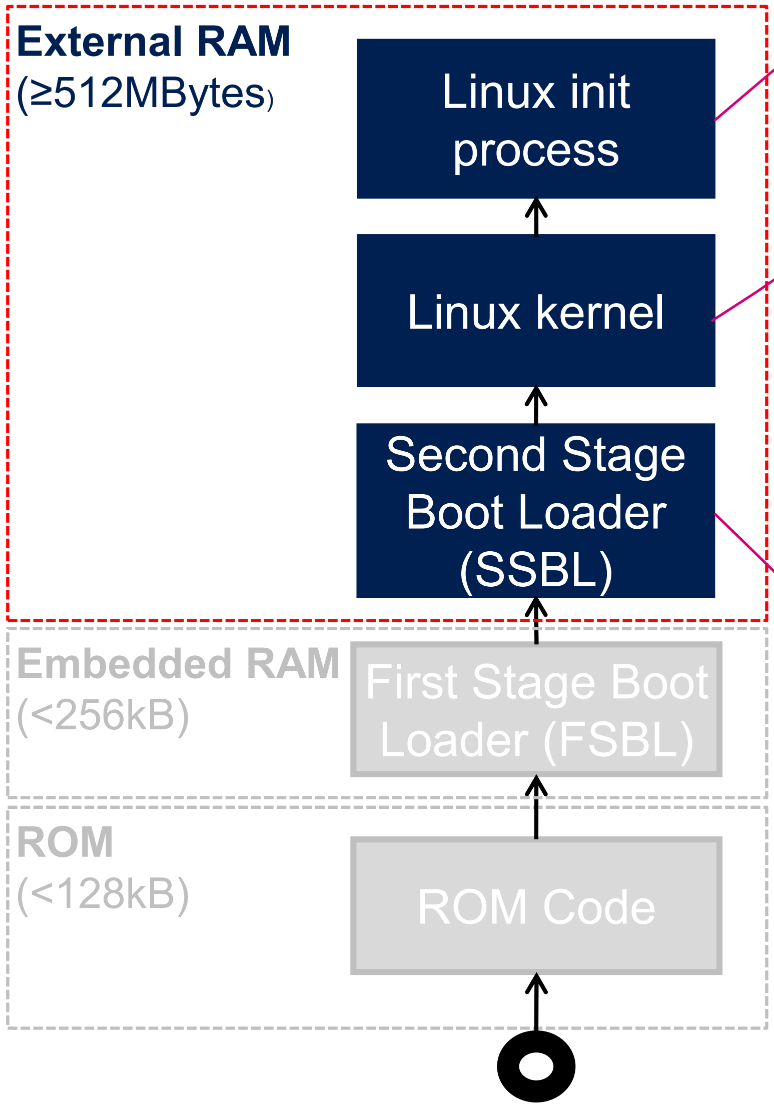


FSBL :

- **Complete clock tree (Pll)** initialization
- **External RAM** (DDR, LpDDR) controller initialization
- **Boot device init** (mass storage or serial link)
- **SSBL loading** from the **boot device** SSBL launch

Linux boot loader

Standard Linux Boot Chain



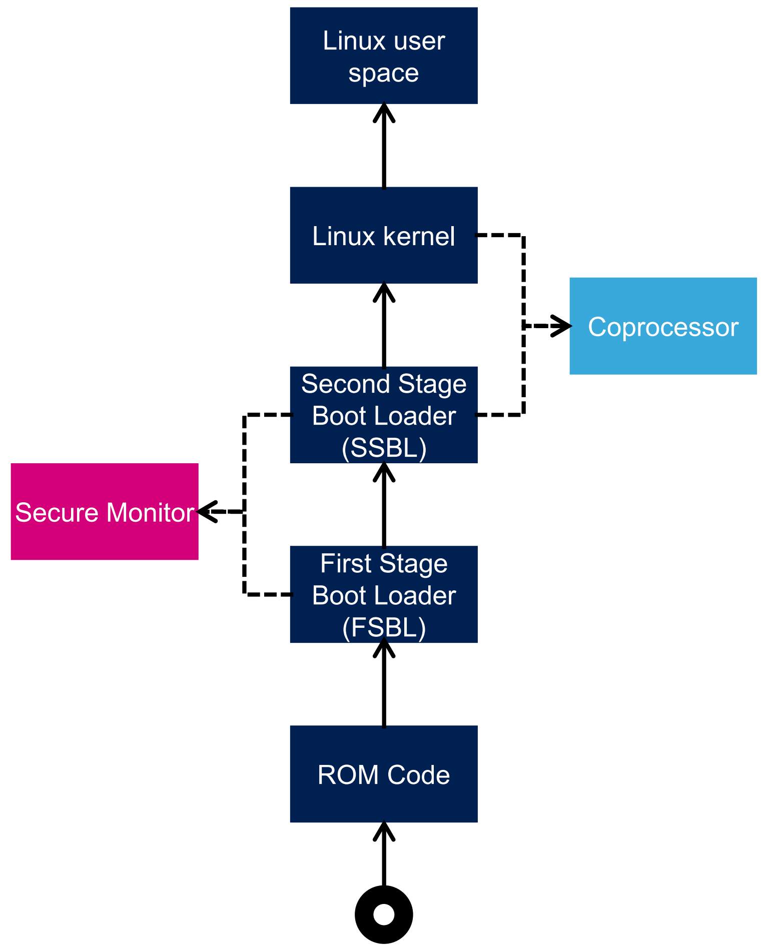
Linux User
- User space services and applications launch

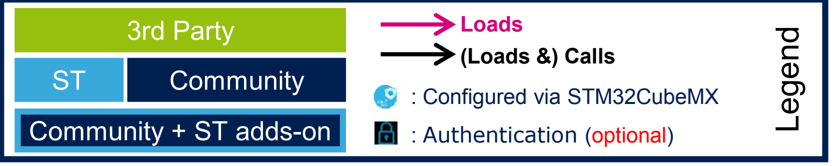
Linux Kernel
- Linux kernel initialisation (platform device drivers, etc.)
- **Root file system** (rootfs) mounting
- User space **init process** launch (/sbin/init)

SSBL:
- **Boot file system** (bootfs) **Ext-RAM loading** from boot device (can be also USB mass storage or Ethernet)
- User feedback with boot loader splash screen
- **Linux kernel** (uImage) **launch** with its device tree blob (*.dtb)

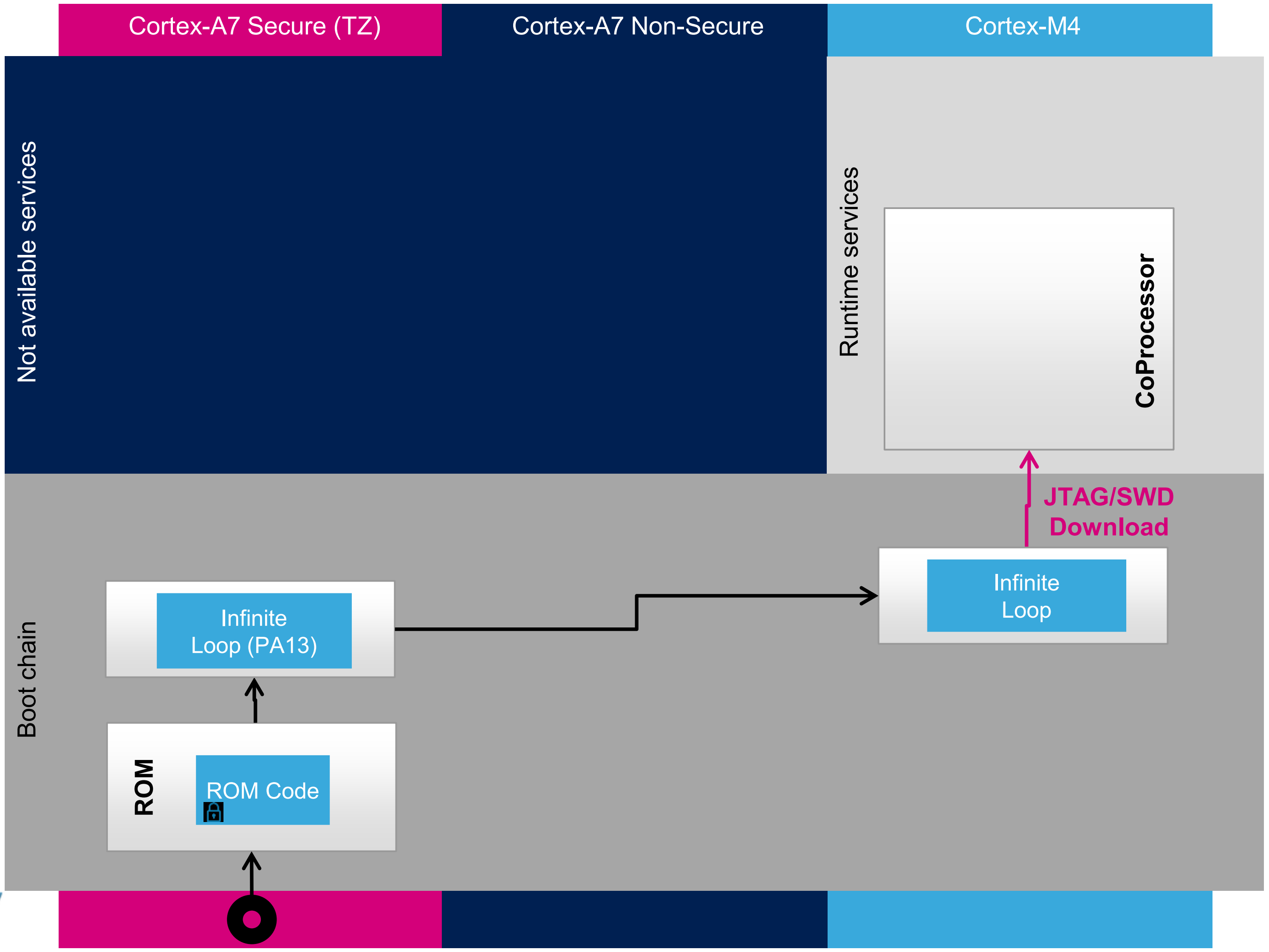
life.augmented

STM32MP1 boot chain


Linux user space
Linux kernel
Coprocessor
Second Stage Boot Loader (SSBL)
Secure Monitor
First Stage Boot Loader (FSBL)
ROM Code



Engineering Boot chain



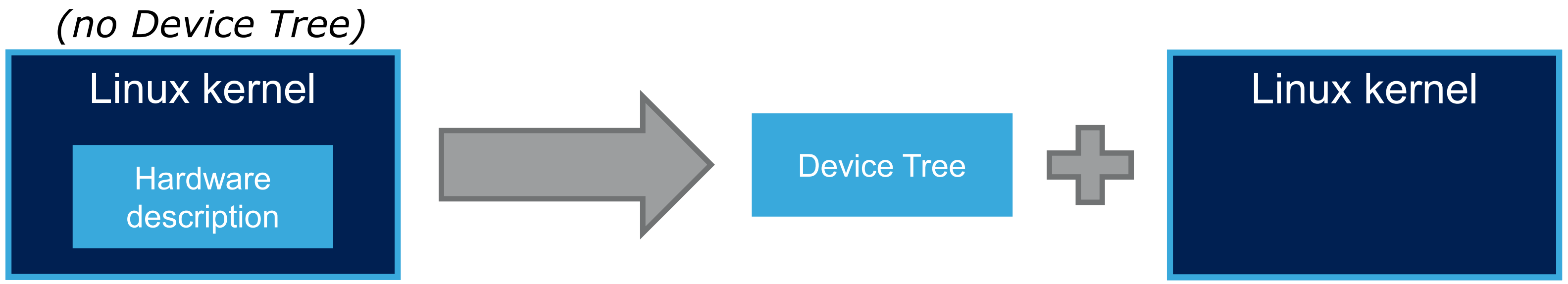
Boot Pins: “0b100”

- Used in M4-only preliminary debug – no Linux boot image needed
- M4 Firmware Load is done via JTAG/SWD
- Need to address clocks and pio alternate function set-up

Boot chain configuration

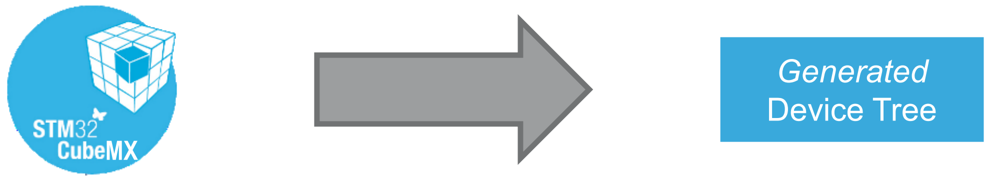
Variability management via device tree

- Former **Linux** kernel used to embed the hardware description of the supported board in the same binary. Current kernels put this information in a separate binary, the **device tree blob** (dtb). As a consequence, a unique kernel binary can support different chips and boards. **U-Boot** also adopted the same solution.



- Device Tree documentation is available on http://elinux.org/Device_Tree, starting from Device Tree for Dummies from Thomas Petazzoni.

- Linux developers manually edit device tree source files (dts): STMicroelectronics enables this **generation from STM32CubeMX** to ease new comers hands-on!



Device tree example for STM32MP1

stm32-usart.c

```
static const struct of_device_id stm32_match[] = {
                              { .compatible = "st,stm32h7-uart", .data = &stm32h7_info},
```

stm32mp157c.dtsi

```
uart4: serial@40010000 {
    compatible = "st,stm32h7-uart";
    reg = <0x40010000 0x400>;
    interrupts-extended = <&intc GIC_SPI 52 IRQ_TYPE_NONE>, <&exti 30 1>;
    clocks = <&rcc_clk UART4_K>;
    status = "disabled";
};
```

stm32mp157-pinctrl.dtsi

```
uart4_pins_a: uart4@0 {
    pins1 {
    pinmux = <STM32_PINMUX('G', 11, AF6)>; /* UART4_TX */
    bias-disable;
    drive-push-pull;
    slew-rate = <0>;
};
…
```

stm32mp157c-ed1.dts

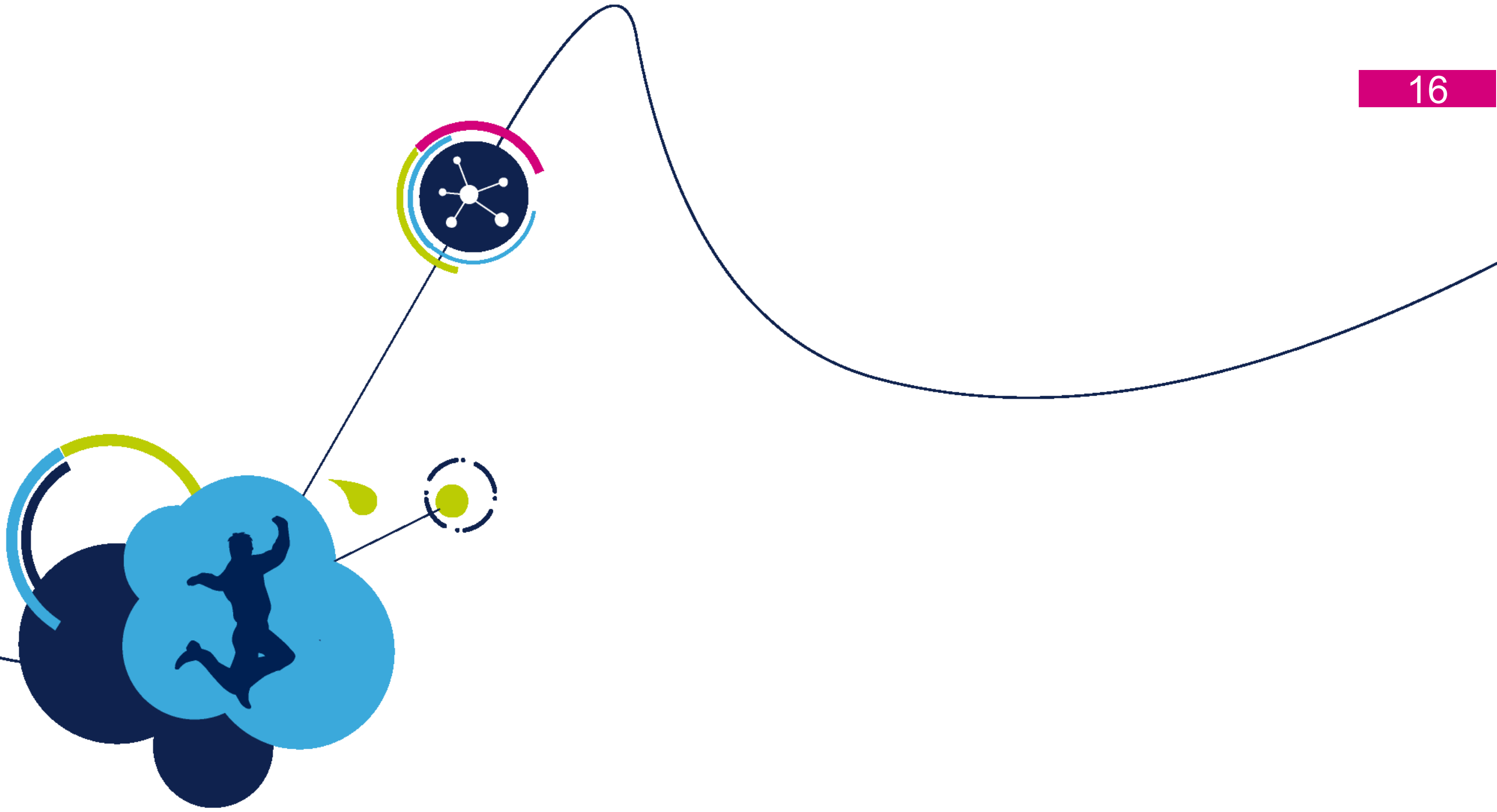
```
&uart4 {
                    pinctrl-names = "default";
                    pinctrl-0 = <&uart4_pins_a>;
                    status = "okay";
};
```

Different ROMCode boot devices

BOOT pins	TAMP_REG[20] (Force Serial)	OTP WORD 3 Primary boot source	OTP WORD 3 Secondary boot source	Boot source #1	Boot source #2 if #1 fails	Boot source if #2 fails
b000	x (don't care)	x (don't care)	x (don't care)	Serial	-	-
b001	!= 0xFF	0 (virgin)	0 (virgin)	QSPI NOR	Serial	-
b010	!= 0xFF	0 (virgin)	0 (virgin)	eMMC	Serial	-
b011	!= 0xFF	0 (virgin)	0 (virgin)	FMC NAND	Serial	-
b100	x (don't care)	x (don't care)	x (don't care)	NoBoot	-	-
b101	!= 0xFF	0 (virgin)	0 (virgin)	SD-Card	Serial	-
b110	!= 0xFF	0 (virgin)	0 (virgin)	Serial	-	-
b111	!= 0xFF	0 (virgin)	0 (virgin)	QSPI NAND	Serial	-
!= b100	!= 0xFF	Primary[1]	0 (virgin)	Primary[1]	Serial	-
!= b100	!= 0xFF	0 (virgin)	Secondary[1]	Secondary[1]	Serial	-
!= b100	!= 0xFF	Primary[1]	Secondary[1]	Primary[1]	Secondary[1]	Serial
!= b100	0xFF	x (don't care)	x (don't care)	Serial	-	-

[1]Primary and Secondary are fields of OTP WORD3.

Reference: AN5031 - Getting started with STM32MP15 Series hardware development.
Wiki article: [STM32MP15_ROM_code_overview]



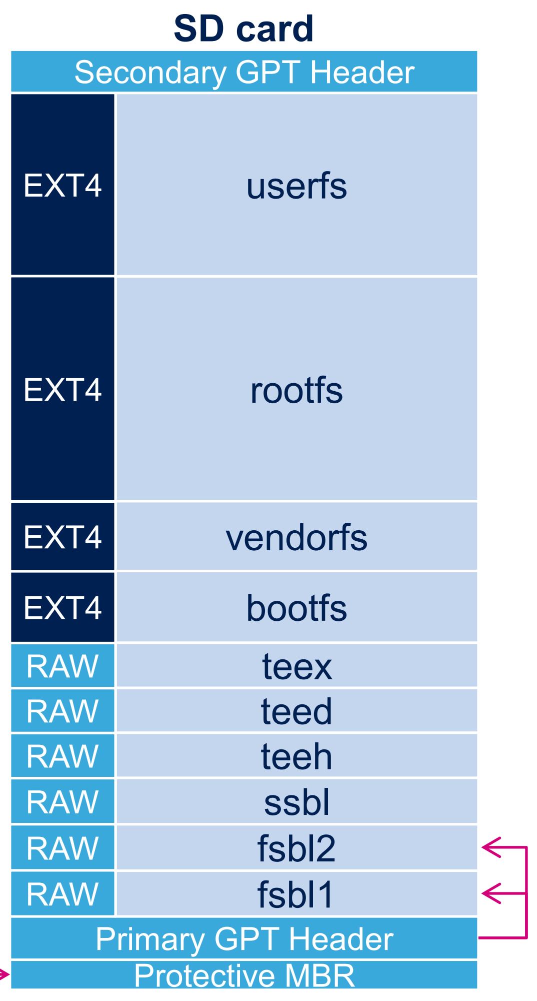
OpenSTLinux flash memory mapping

Flash partitions

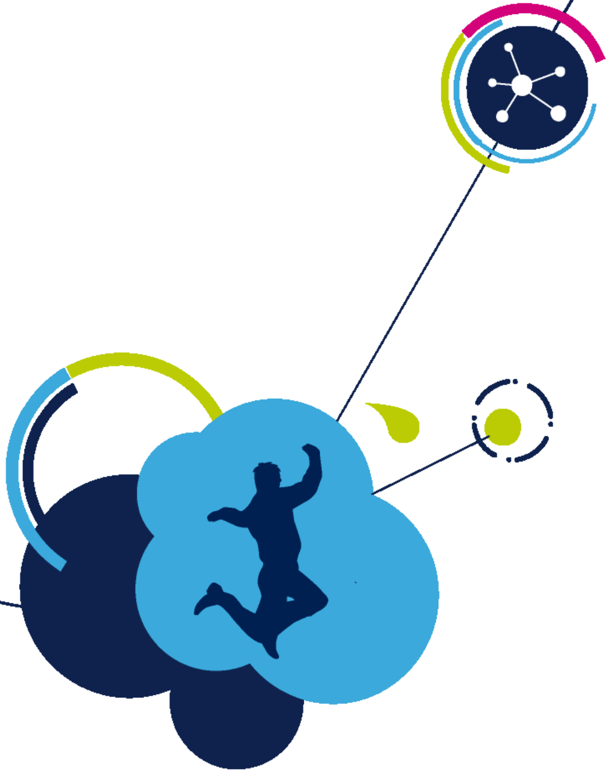
Size	Component	Comment
Remaining area	userfs	The user file system contains **user data and examples**
768MB	rootfs	**Linux root file** system contains all user space binaries (executable, libraries, …) and **kernel modules**
64MB	bootfs	The boot file system contains: - (option) the init ram file system, that can be copied to the external RAM and used by Linux before mounting a fatter rootfs - **Linux kernel device tree** (can be in a Flattened Image Tree - FIT) - **Linux kernel** U-Boot image (can be in a Flattened Image Tree - FIT) - The boot loader splash screen image, displayed by U-Boot - U-Boot distro config file extlinux.conf (can be in a Flattened Image Tree – FIT)
1MB	ssbl	The **Second Stage Boot Loader** (SSBL) **is U-Boot**, with its device tree blob (dtb) appended at the end
256kB	fsbl	The **First Stage Boot Loader** is ARM Trusted Firmware (TF-A) with its device tree blob (dtb) appended at the end. At least two copies are embedded. Note: due to ROM code RAM needs, FSBL payload is limited to 247kB.

life.augmented

SD card memory mapping



ROM code path to look for FSBL



Security on STM32MP1

Security

Based on 2 mechanisms

- MCU hardware isolation avoids A7 corruption of M4 peripherals configuration or M4 memory

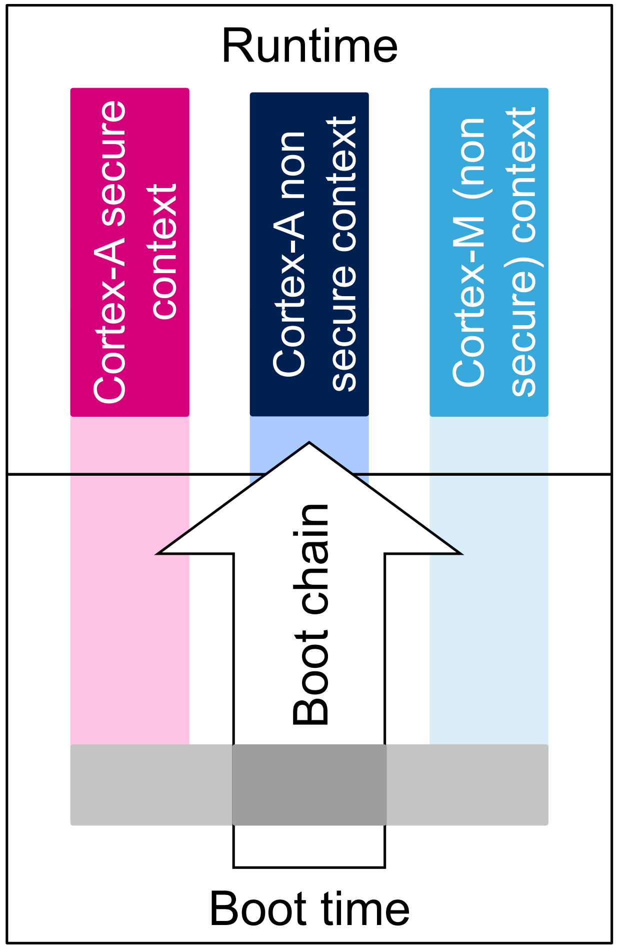
and

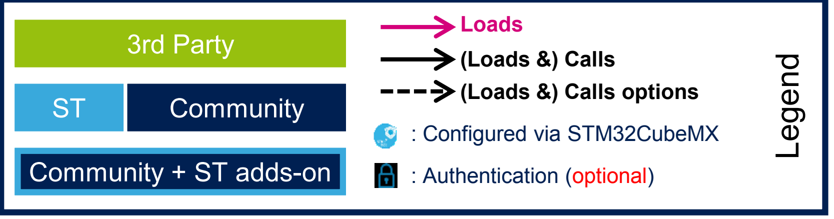
- A7 hardware isolation called Arm Trust-zone.
  Arm Trust-zone enables isolated hw execution context.
  A7 access to these peripherals & memories depends on this execution context.

hw execution context

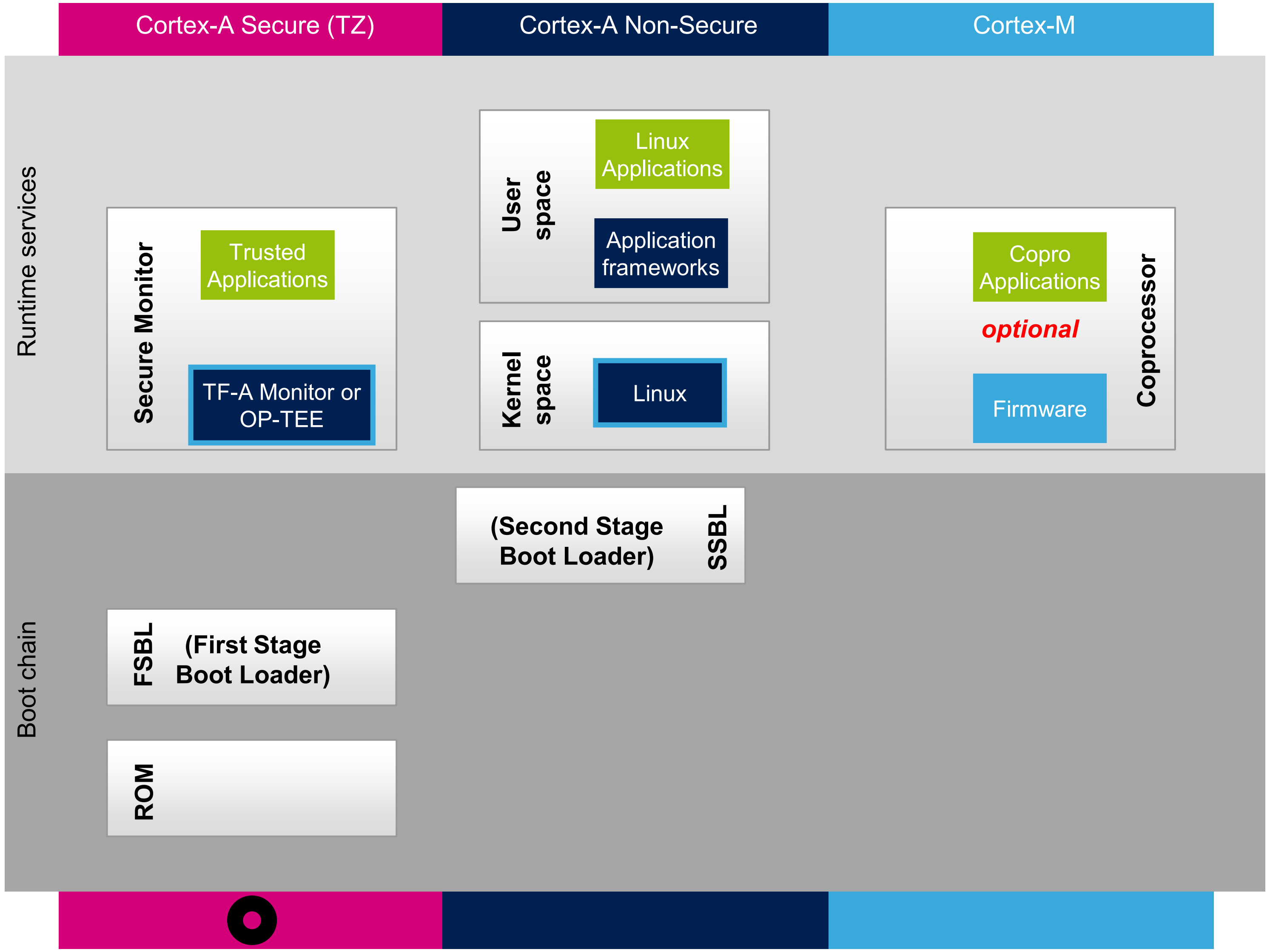
- Hardware execution context
  - « a core and a security mode »
  - Each Cortex-A core can run secure and non-secure contexts

- Peripheral assignment to one hw execution context

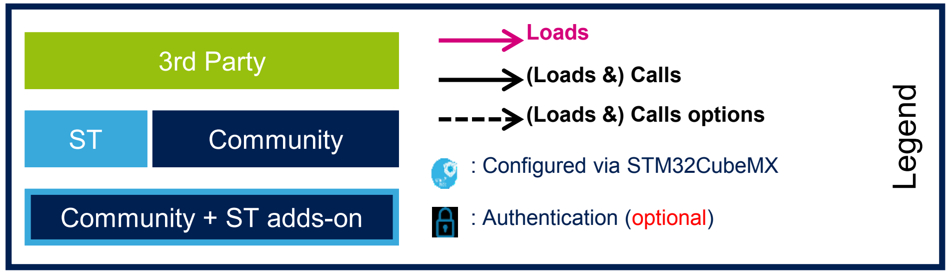


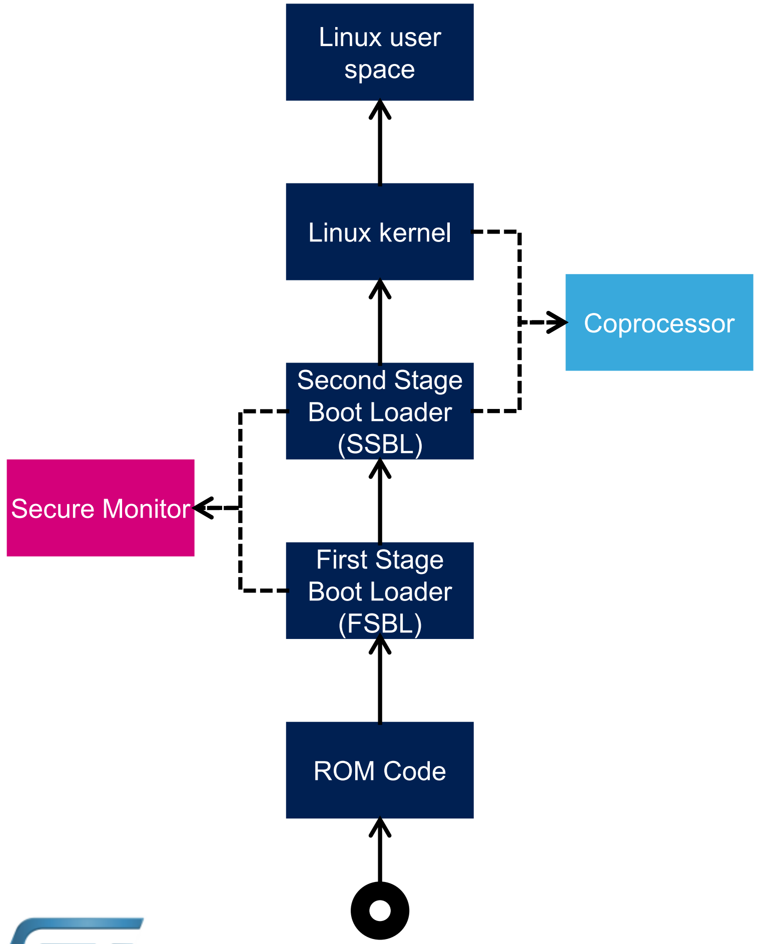


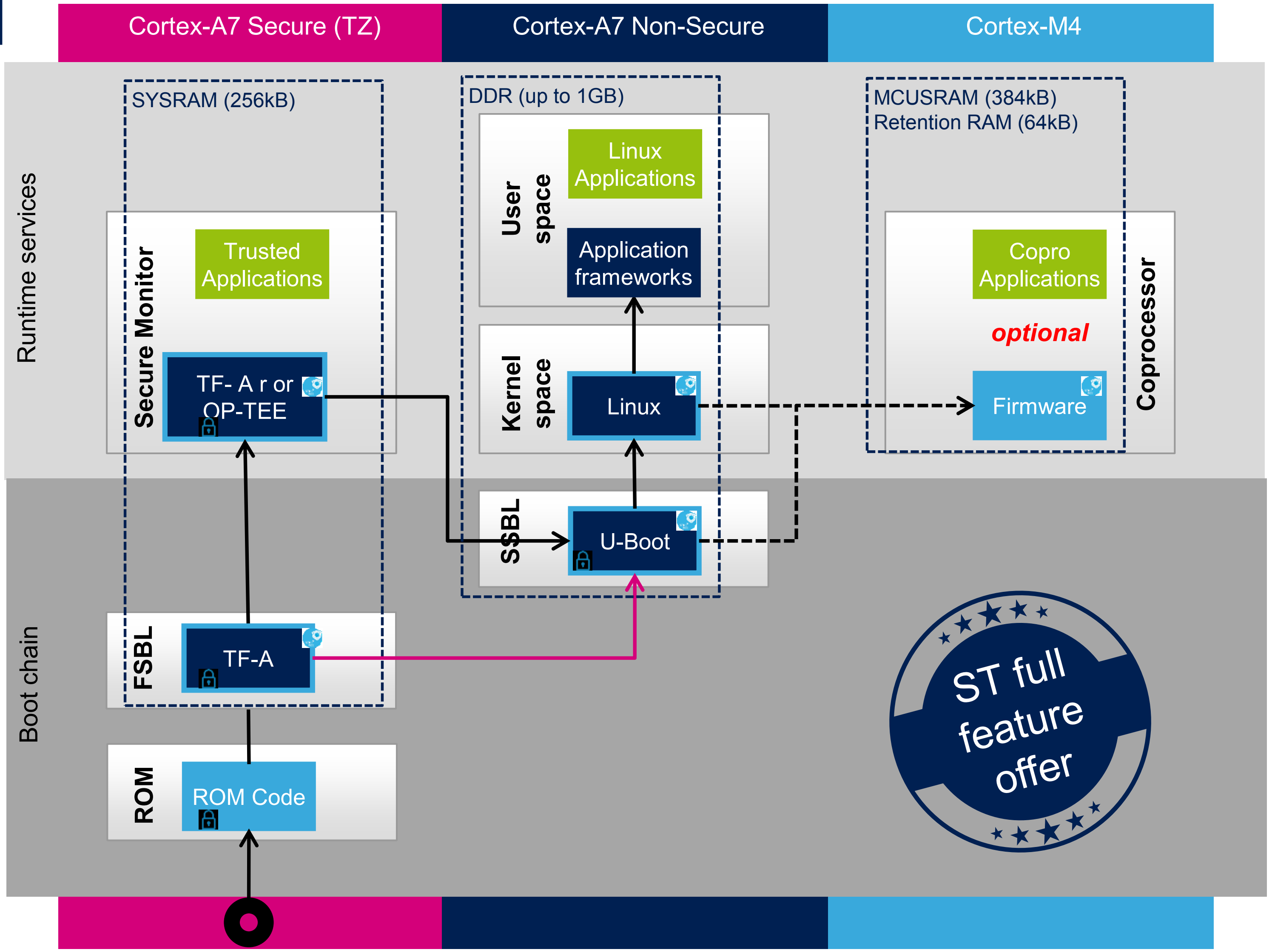
Hw execution contexts



A Trusted boot chain

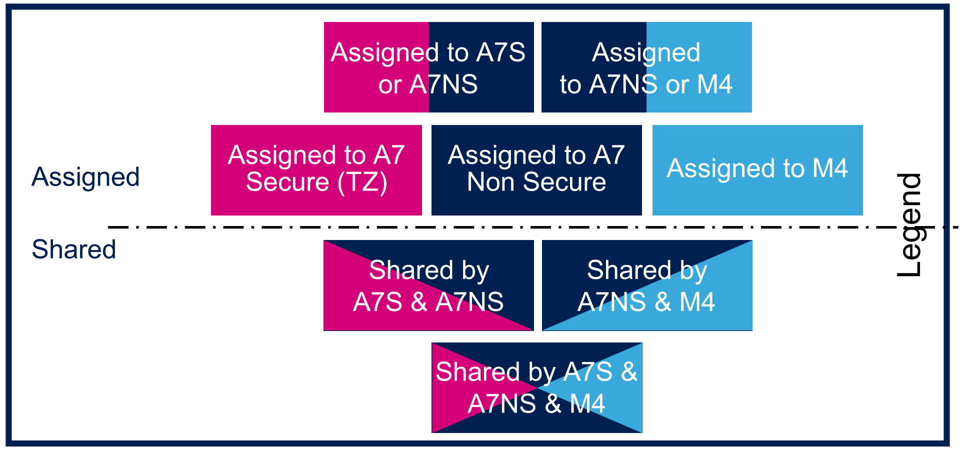



Linux user space
Linux kernel
Coprocessor
Second Stage Boot Loader (SSBL)
Secure Monitor
First Stage Boot Loader (FSBL)
ROM Code



Note : a Basic boot chain is also available, fully relying on U-Boot (instead of TF-A + U-Boot)

Peripheral (IP) assignment



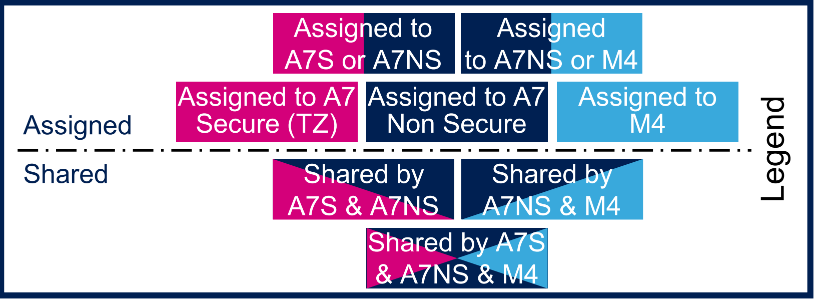
All IP are associated to an *hardware execution context*

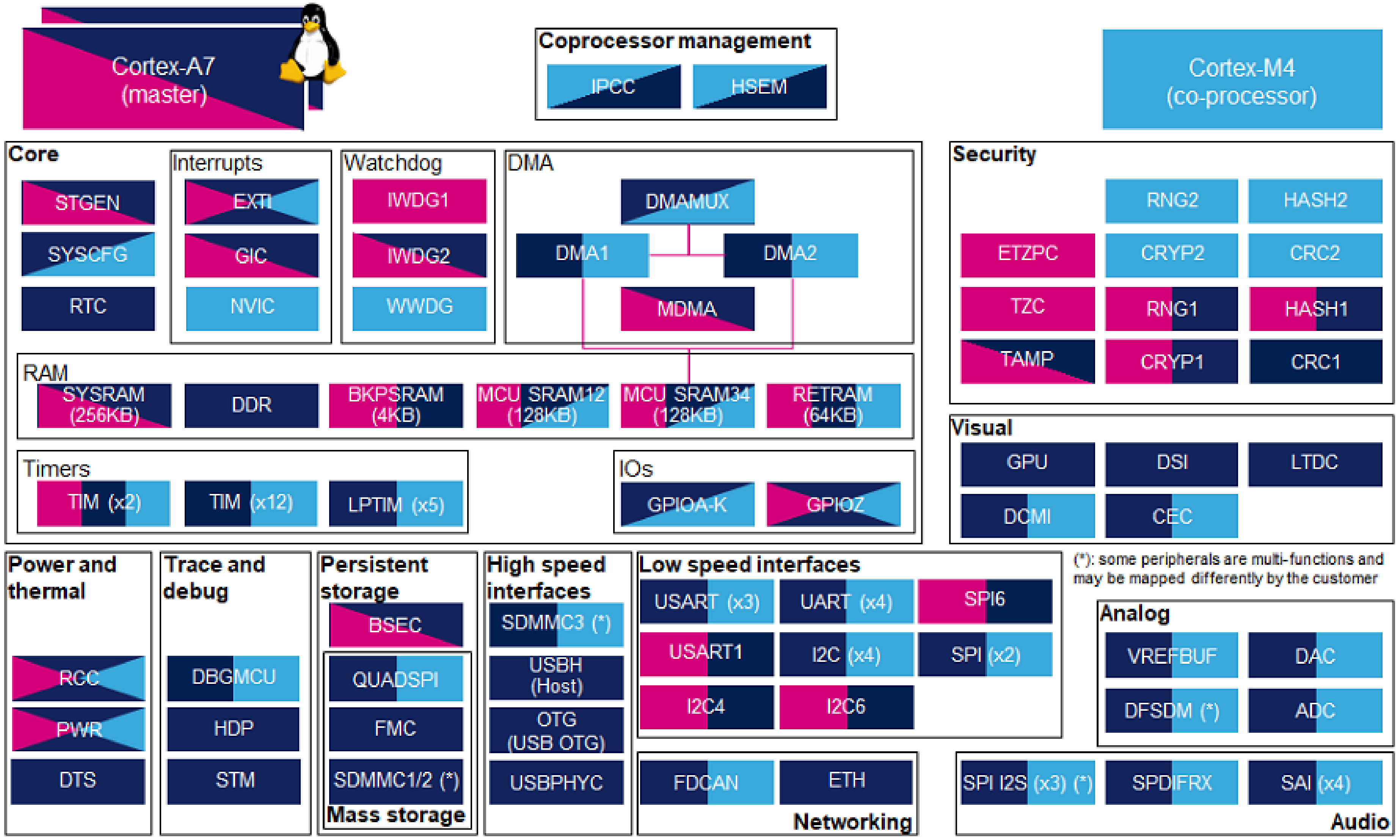
- IP can be “**assigned**” exclusively to an hw execution context
- IP can be “**shared**” between the 2 or 3 hw execution contexts

- When an IP is assigned to hw execution context
  - **Cortex-A secure**: IP register or mem accessible to A7 in secure mode (not by M4,A7-non-secure)
  - **Cortex-A non-secure**: IP register or mem accessible to A7S, A7NS, M4
  - **Cortex-M**: IP register or mem accessible by M4, A7 cannot access
- IP assigned ensured by ETZPC IP (Extended Trust Zone Protection Controller)
  – static configuration (TF-A device tree - manual or by CubeMx)

Peripherals sharing - M4 isolation

Source: ST Wiki article [STM32MP15 peripherals overview]

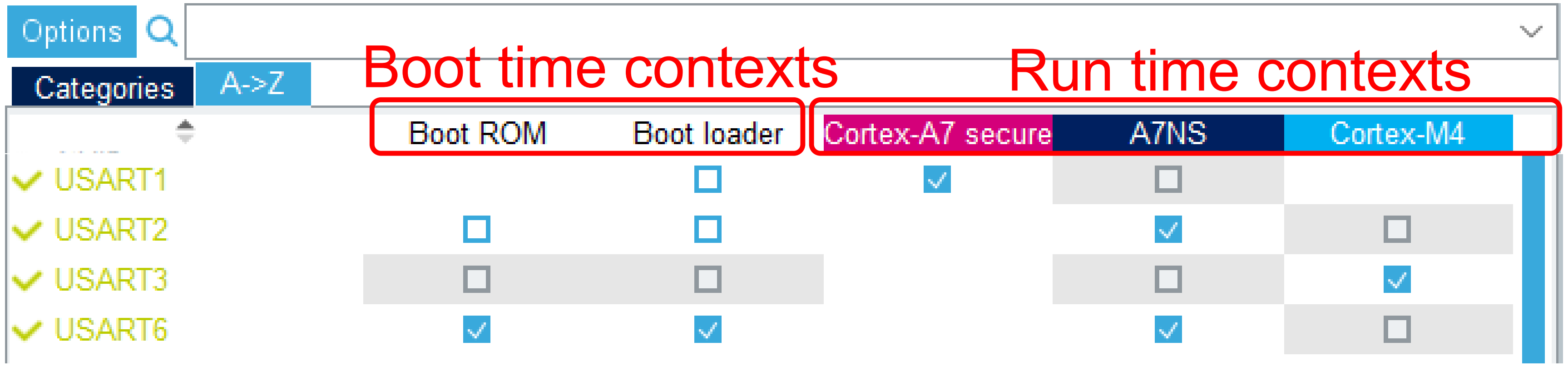




(*): some peripherals are multi-functions and may be mapped differently by the customer

life.augmented

Peripherals assignment via STM32CubeMX

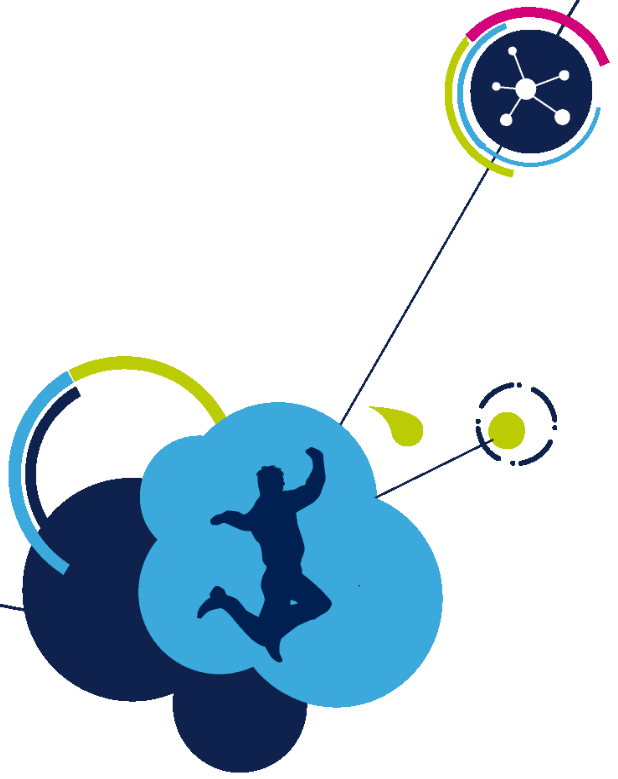


Options | Categories | A->Z

Boot time contexts

Run time contexts

	Boot ROM	Boot loader	Cortex-A7 secure	A7NS	Cortex-M4
✓ USART1		☐	✓	☐	
✓ USART2	☐	☐		✓	☐
✓ USART3	☐	☐		☐	✓
✓ USART6	✓	✓		✓	☐



Thanks