STM32MP1软件架构

life.augmented

STM32MP1 Flash内存映射

life.augmented

Flash partitions (minimal)

Size	Component	Comment
Remaining area	userfs	包括用户数据以及STM32实例应用
768MB	rootfs	Linux根文件系统，包括所有用户空间的程序(可执行文件, 软件库, …) 以及内核模块
16MB	vendorfs	存放第三方专有二进制文件，使其与开源License例如GPL V3分开
64MB	bootfs	启动文件系统包括： - (可选) 初始RAM文件系统。该文件系统可以被复制到外部DDR，在还没有挂装大的根文件系统供linux使用 - Linux kernel device tree (可以做成统一的Flattened Image Tree - FIT) - Linux kernel U-Boot image (可以做成统一的Flattened Image Tree - FIT) - 除NOR之外的所有Flash: 由U-Boot显示的启动splash screen image - U-Boot distro 配置文件extlinux.conf (可以做成统一的Flattened Image Tree – FIT)
2MB	ssbl	存放第二级启动加载器Second Stage Boot Loader (SSBL)。这里是U-Boot, 包括附在文件尾的device tree blob (dtb)
256kB to 512kB (*)	fsbl	存放第一级启动加载器First Stage Boot Loader 。可以是ARM Trusted Firmware (TF-A) 或者U-Boot Secondary Program Loader (SPL)包括文件尾的device tree blob (dtb)。至少需要两份。 注意： 由于ROM代码的RAM要求，FSBL大小限制在247KB。

(*): the partition size depends on the flash technology, to be aligned on block erase size for NOR (256kB) / NAND (512kB)

life.augmented

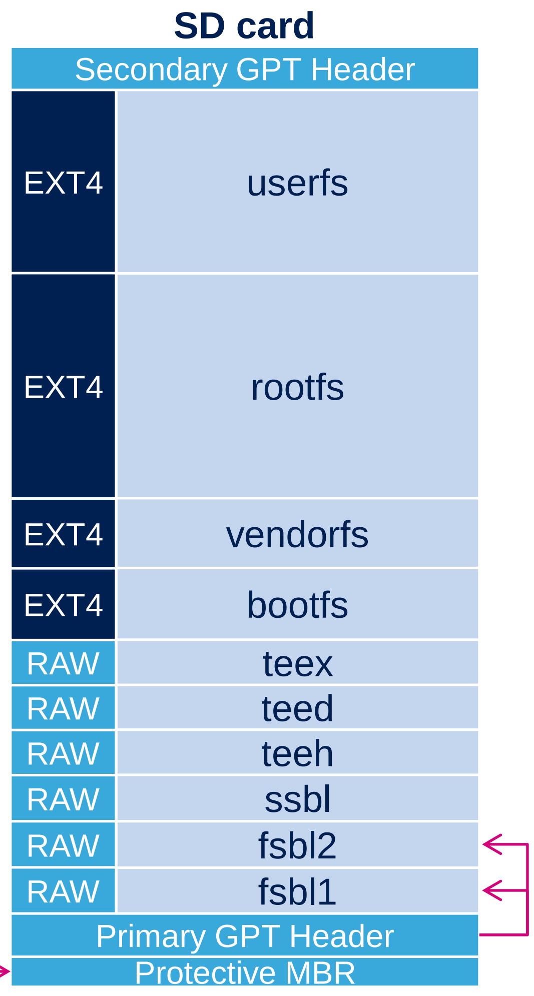
Flash partitions (可选)

Size	Component	Comment
256kB (*)	logo	供NOR flash存放boot loader splash screen image (对于其他的Flash, 该image存放在bootfs分区)
256kB to 512kB (*)	teeh	OP-TEE header
256kB to 512kB (*)	teed	OP-TEE pageable code and data
256kB to 512kB (*)	teex	OP-TEE pager

(*): the partition size depends on the flash technology, to be aligned on block erase size for NOR (256kB) / NAND (512kB)

SD card memory mapping

ROM代码首先查找在SD card头部的GPT 分区表

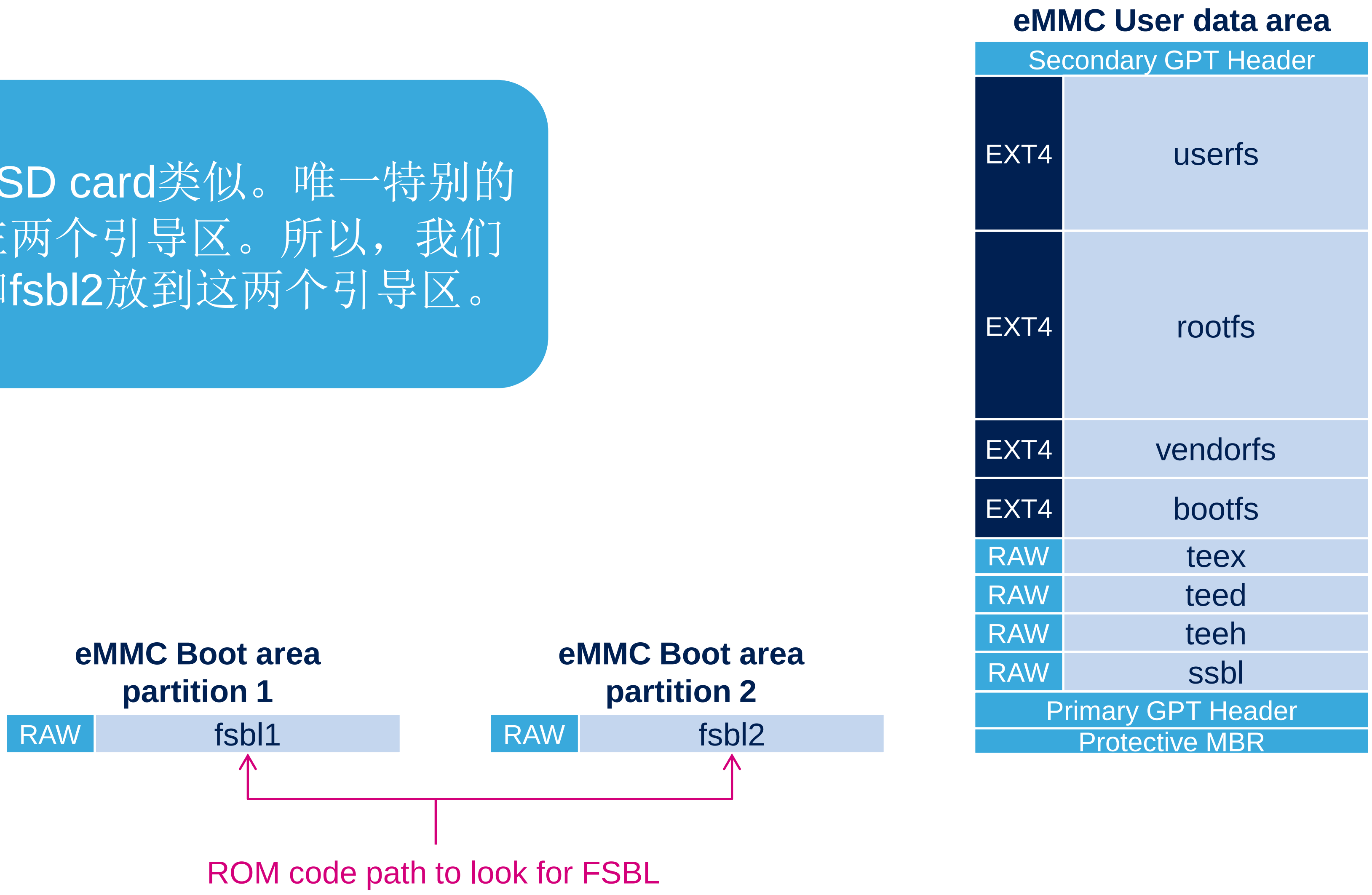


ROM code path to look for FSBL

- 两个fsbl分区时为了在更新时可以保证一个有效分区
- 存在ssbl和tee分区是因为fsbl不支持文件系统
- bootfs是个 EXT4分区。U-Boot使用该分区。
- rootfs和userfs是EXT4文件系统。Linux使用该分区。

eMMC memory mapping

eMMC与SD card类似。唯一特别的是它存在两个引导区。所以，我们讲fsbl1和fsbl2放到这两个引导区。



life.augmented

NOR memory mapping

SD card

Secondary GPT Header	
EXT4	userfs
EXT4	rootfs
EXT4	vendorfs
EXT4	bootfs
Primary GPT Header	
Protective MBR	

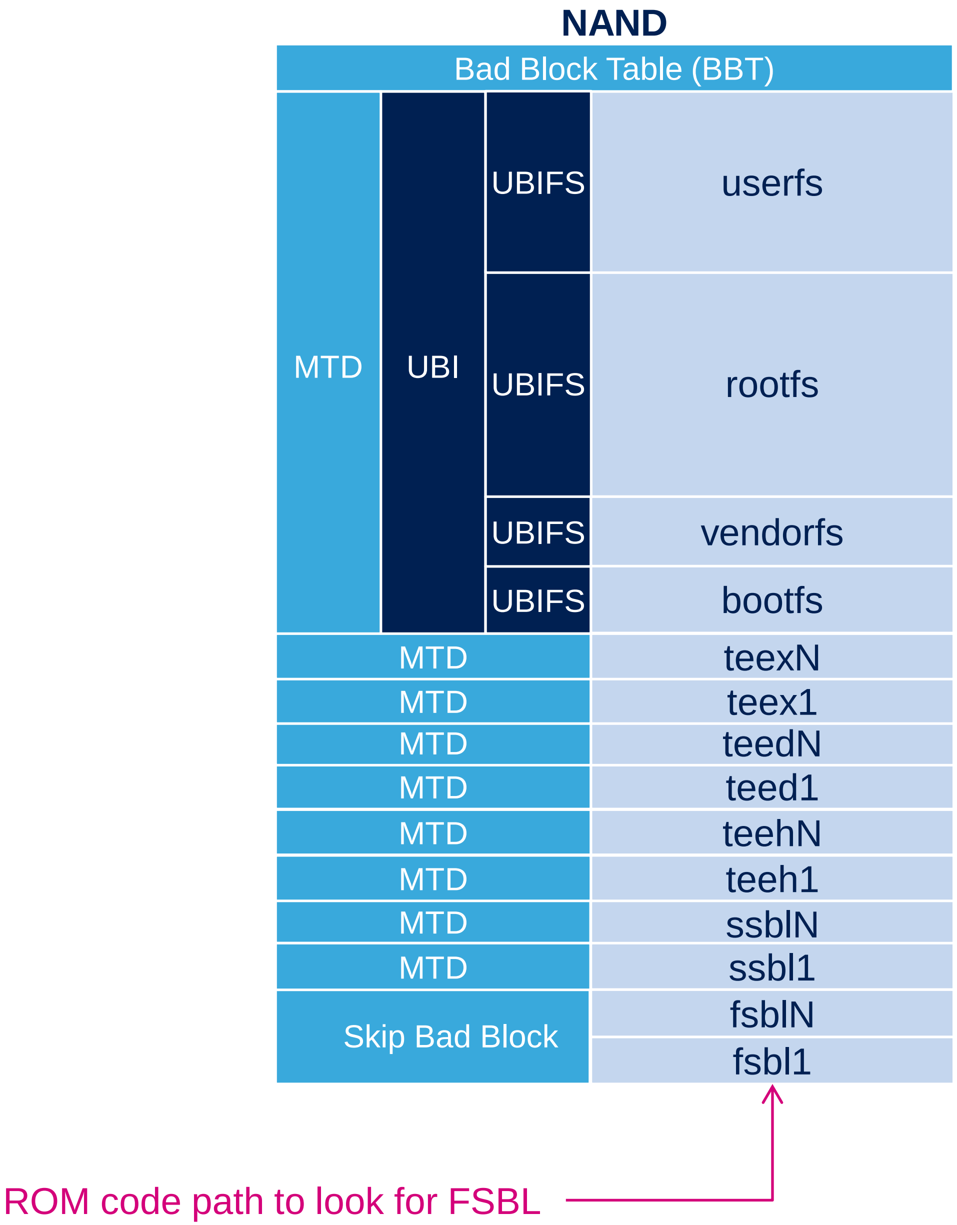
注意： 用户基于成本考虑，NOR Flash一般不大，需要第二级存储空间，例如SD卡。也可以使用eMMC or NAND.

QSPI NOR

RAW	teex	
RAW	teed	
RAW	teeh	
RAW	logo	
RAW	ssbl	
RAW	fsbl2	Offset 256kB
RAW	fsbl1	Offset 0

ROM code path to look for FSBL

NAND memory mapping

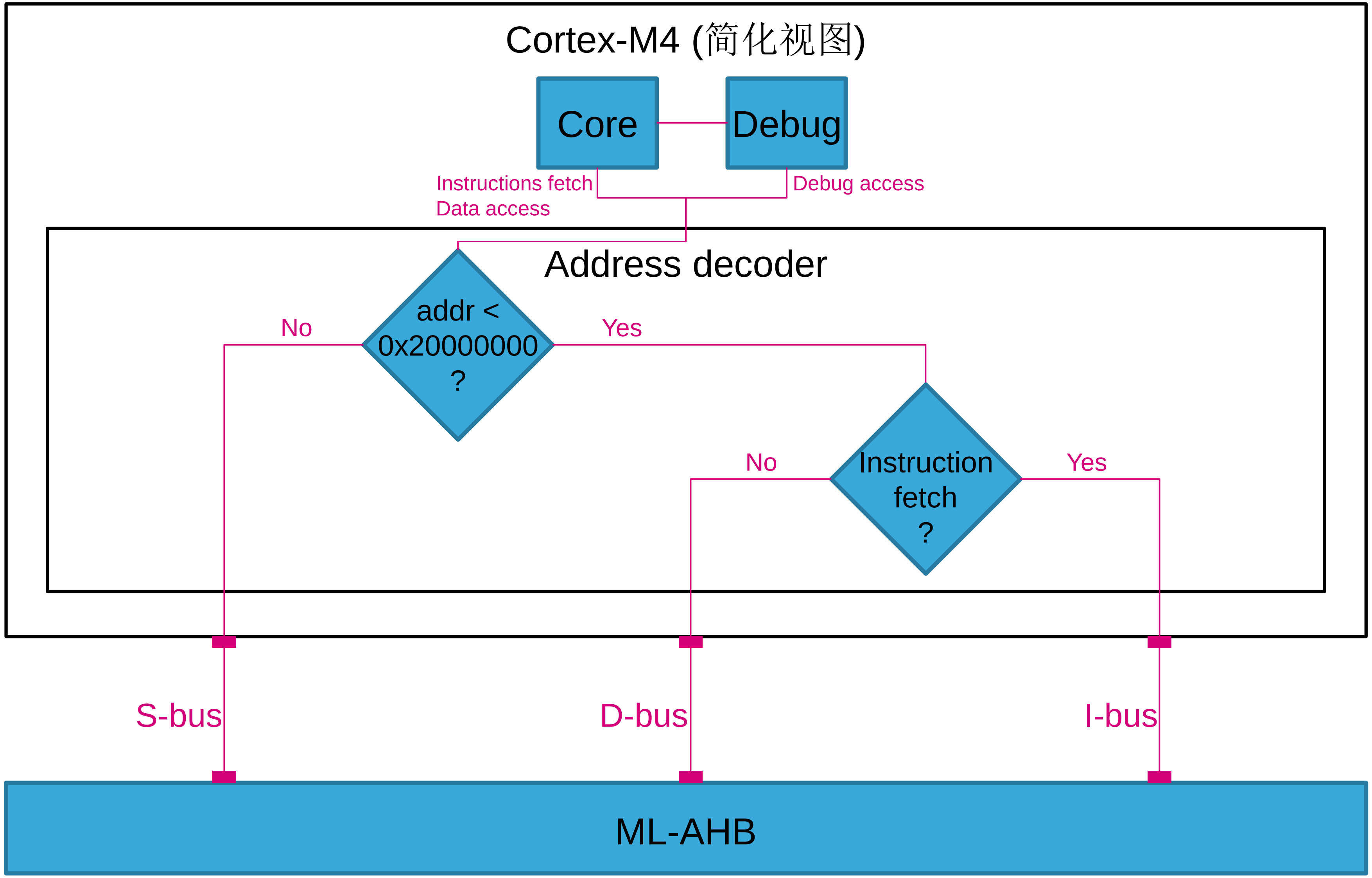


- NAND成本低但需要注意坏块管理
- NAND由其物理性质本身容易引起位翻转，故ECC是必须的

STM32MP1 RAM内存映射

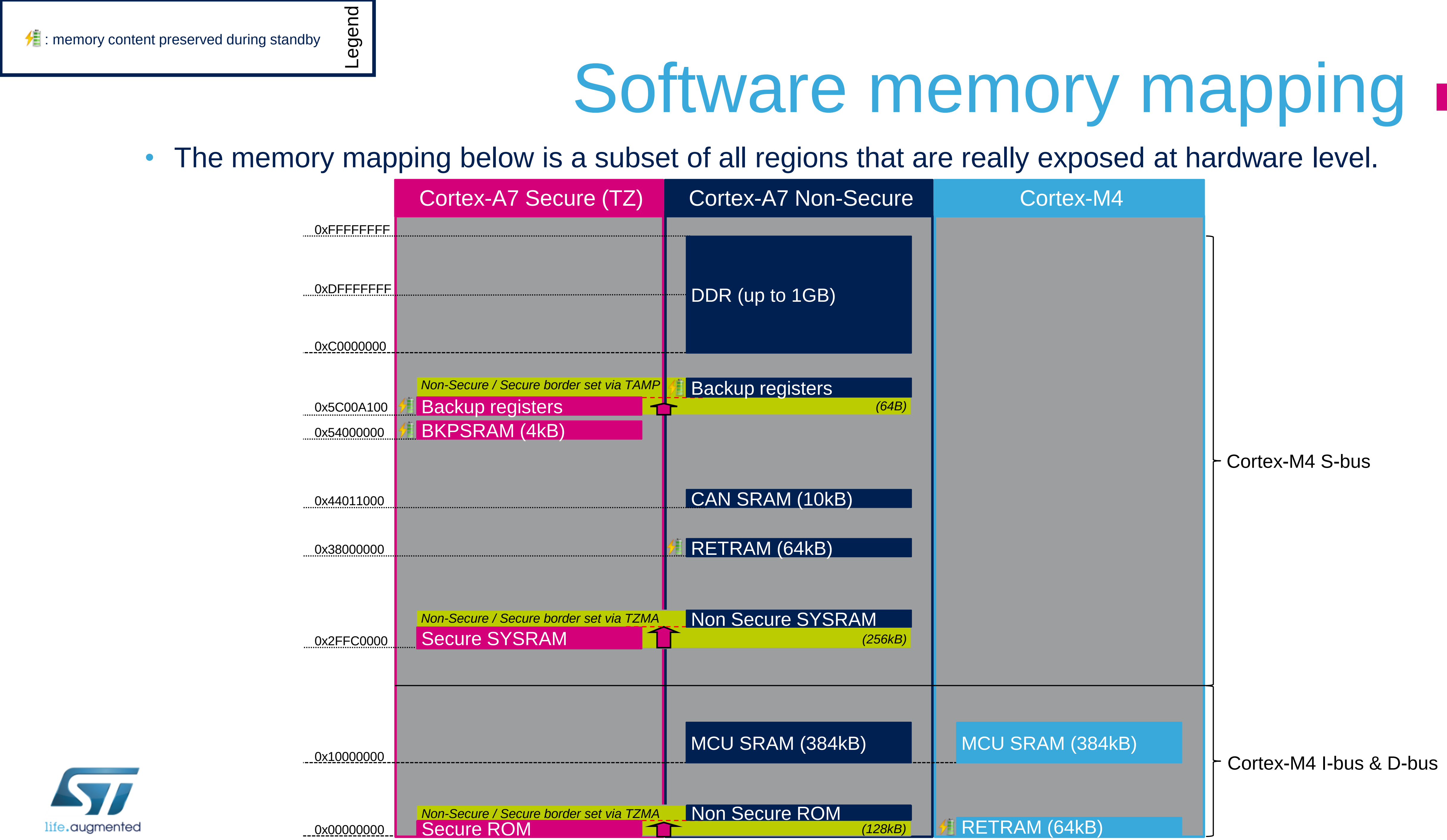
life.augmented

Cortex-M4 ports



Software memory mapping

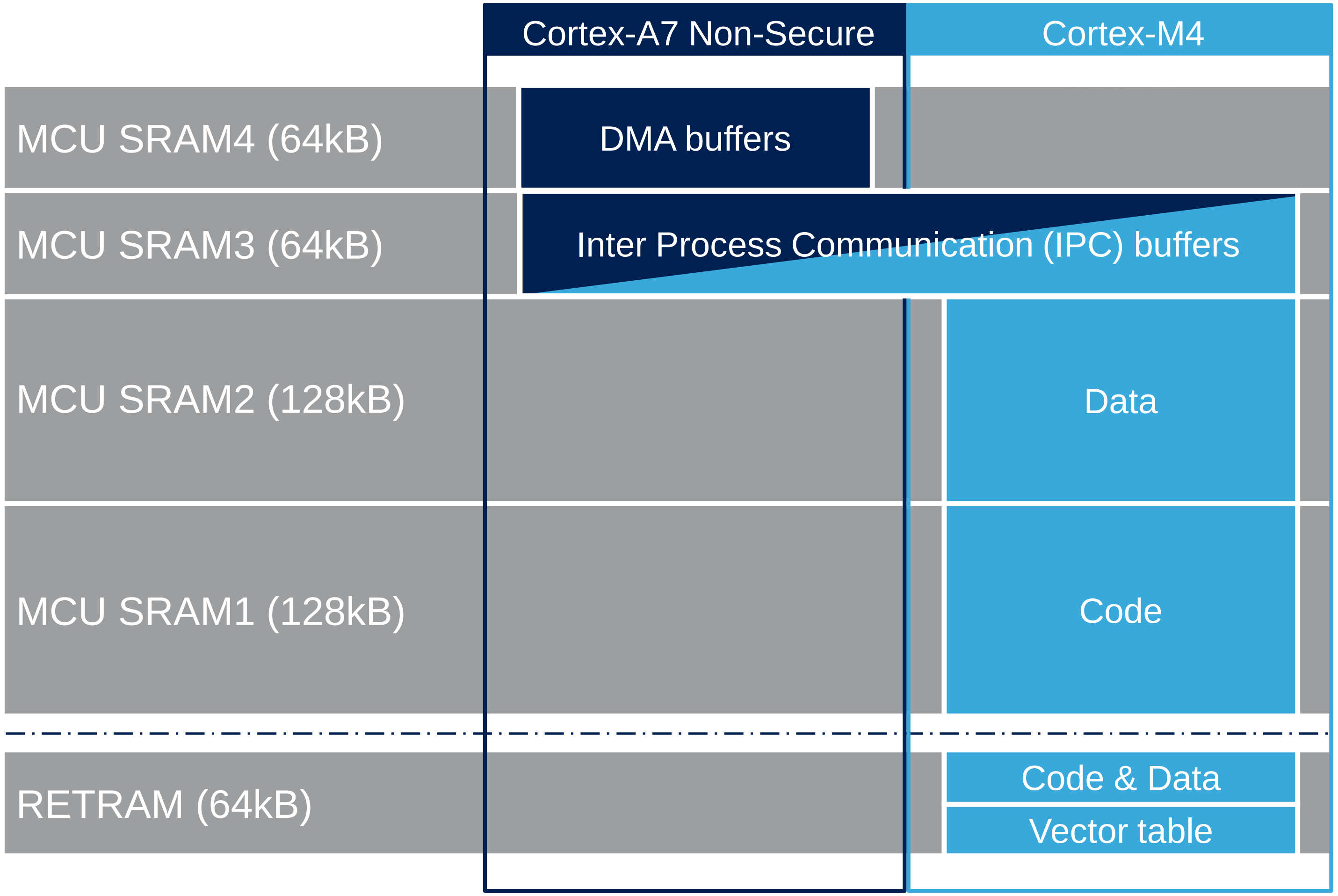
- The memory mapping below is a subset of all regions that are really exposed at hardware level.



life.augmented

Shared RAM memory mapping

- Notice that each core may not see the same regions at the same address, as already explained on previous slide

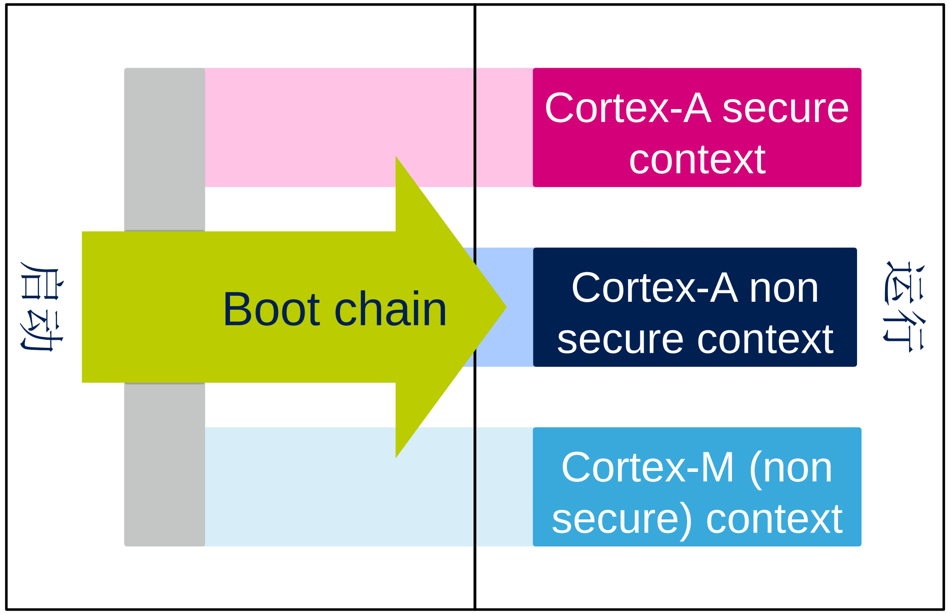


- Each customer can of course tune this mapping (regions location and sizes) to fit with his product needs

STM32MP1软件架构

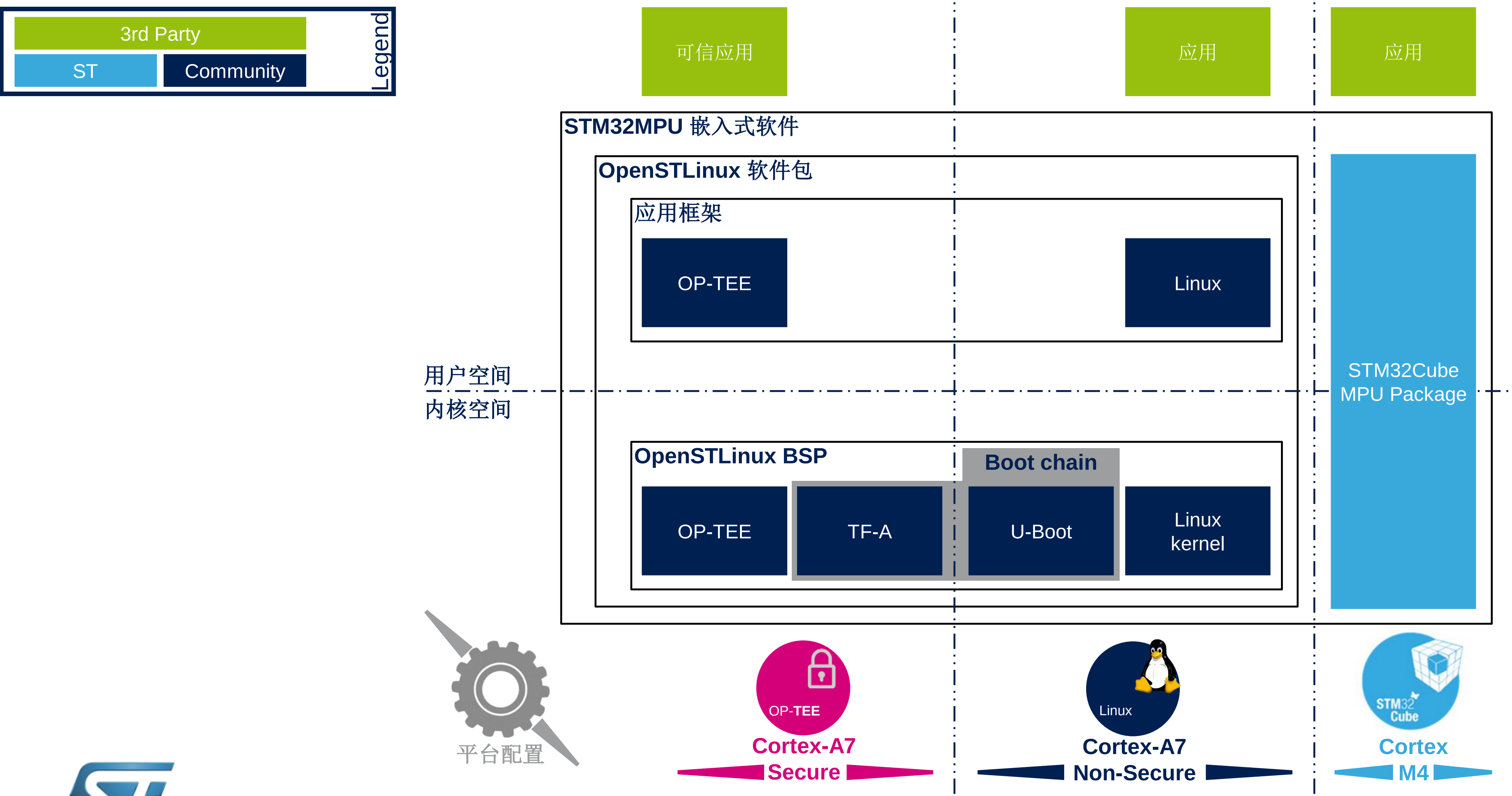
STM32MP1多核软件架构

- 系统软件运行情境划分
  - 按照内核
  - 按照安全(同一个内核的两种不同状态）
- 系统中同时存在多个软件运行情境
  - Arm Cortex-A secure (Trustzone) 运行OP-TEE
  - Arm Cortex-A non secure运行Linux
  - Arm Cortex-M (non secure)运行STM32Cube
- 运行时外设分配
  - 指定或者共享

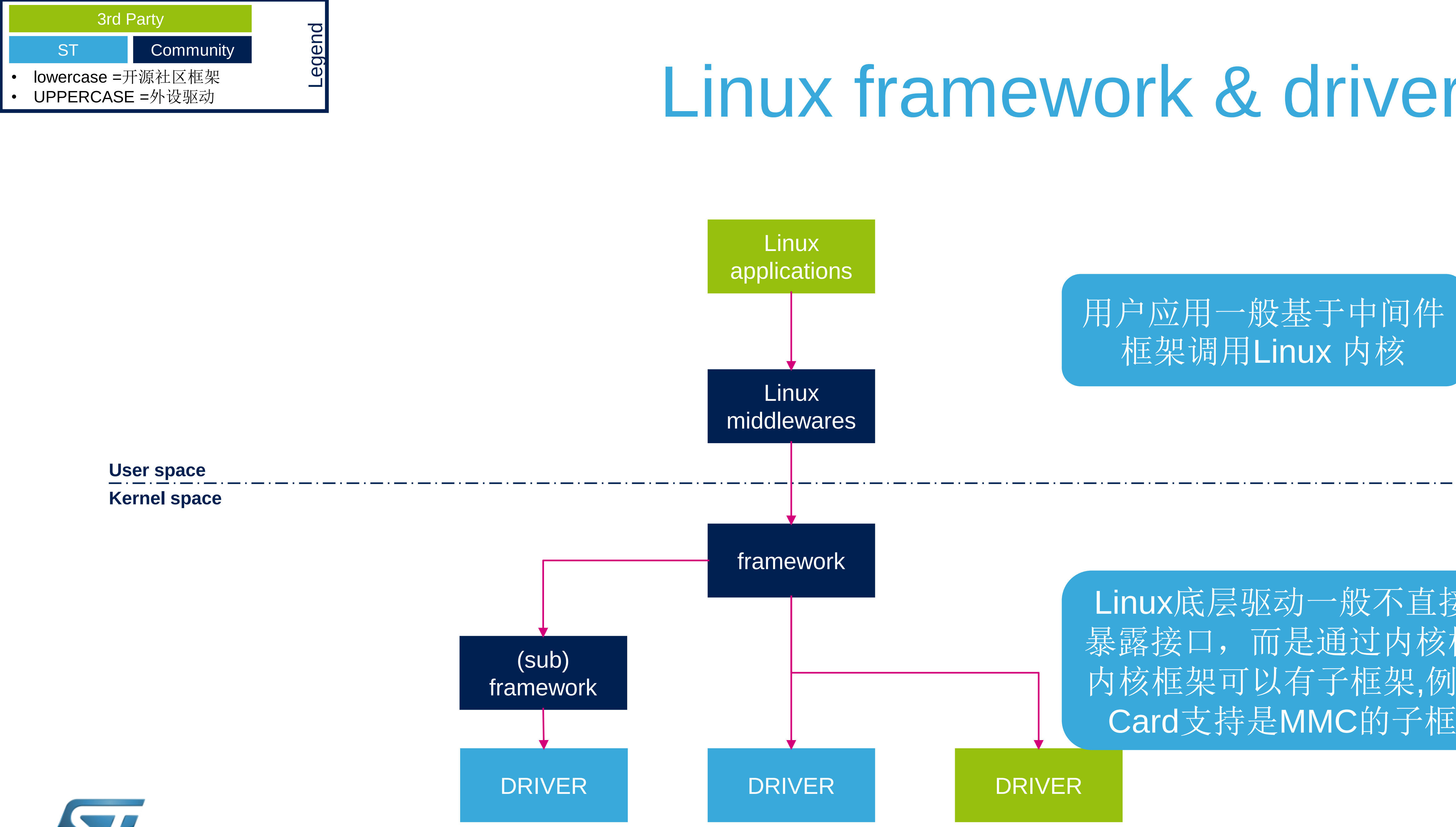


life.augmented

STM32MP1嵌入式软件组成



Linux framework & driver



用户应用一般基于中间件框架调用Linux 内核

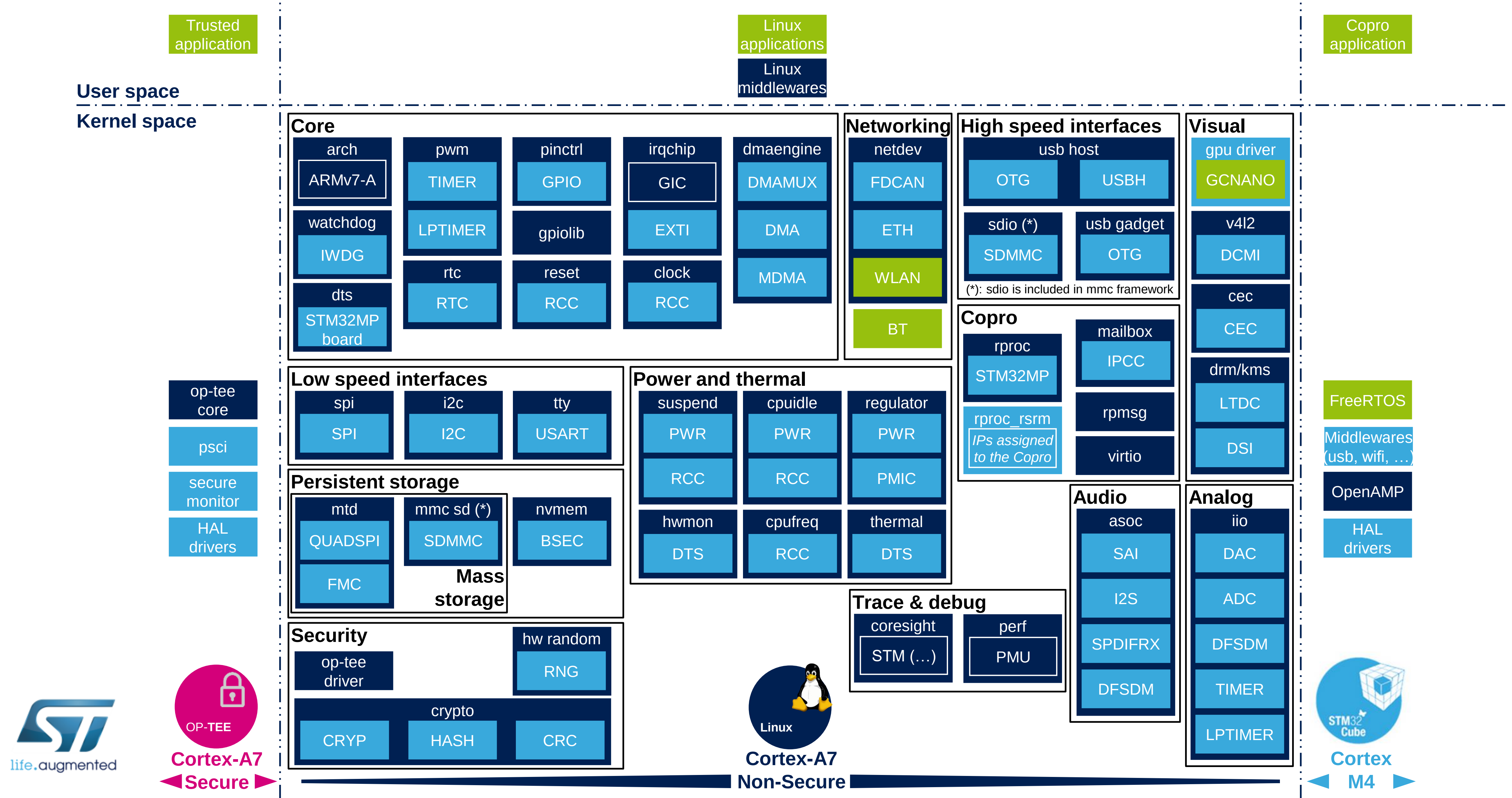
Linux底层驱动一般不直接对外暴露接口，而是通过内核框架。内核框架可以有子框架,例如:SD Card支持是MMC的子框架。

3rd Party
ST | Community

- lowercase = 开源社区框架
- UPPERCASE = 外设驱动

Legend

OpenSTLinux + STM32Cube



Legend: 3rd Party | ST | Community

Open-Embedded 用户空间应用

- 这里仅列出了一部分应用
- 用户可以根据需要进行定制

Linux applications

Core
bash, busybox, grep, coreutils, kbd
apt, dpkg, elfutils, kmod, dbus
python, perl, binutils, blktool, memtester, libraries (glibc, …)

Networking
can-utils, dhcp
iproute, bluez5

Low speed interfaces
i2c-tools, minicom, ckermit

High speed interfaces
libusb

Visual
wayland
glmark2, weston, fb-test, libv4l
gcnano OpenGL ES, libdrm, libjpeg, gstreamer

Mass Storage
gptfdisk, mmcutils, mtdutils

Power & Thermal
cpufreq utils

Security
openssh, openssl, ca-certificates

Debug
gdb-server, lttng, blktrace

Audio
alsa

Analog
libiio

yocto PROJECT COMPATIBLE

openembedded

User space

Kernel space

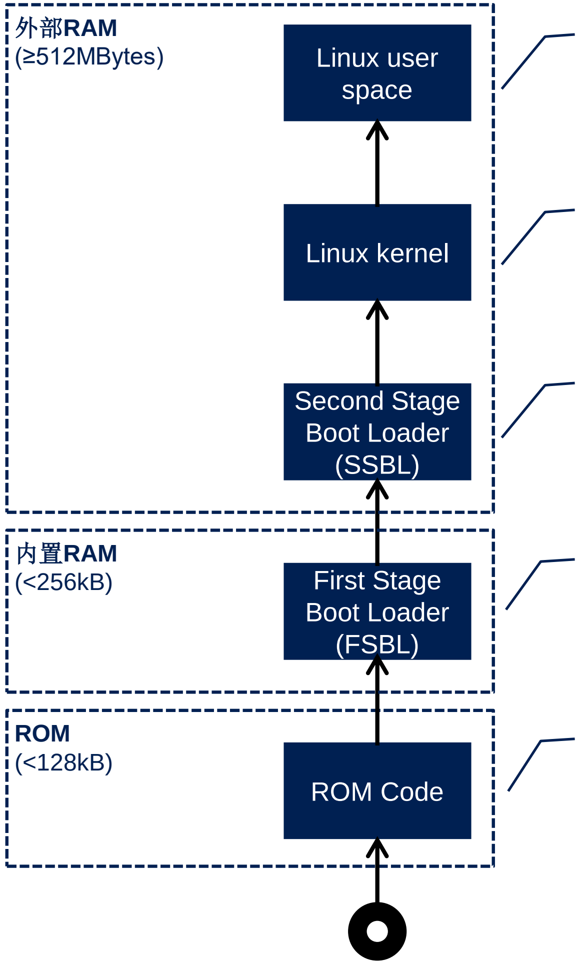
Linux

Cortex-A7
Non-Secure

life.augmented

STM32MP1 启动流程

标准Linux启动过程



- 加载用户空间服务和应用

- Linux kernel 初始化 (平台驱动等)
- 挂载根文件系统(rootfs)
- 加载用户空间初始化进程User space init process launch (/sbin/init)

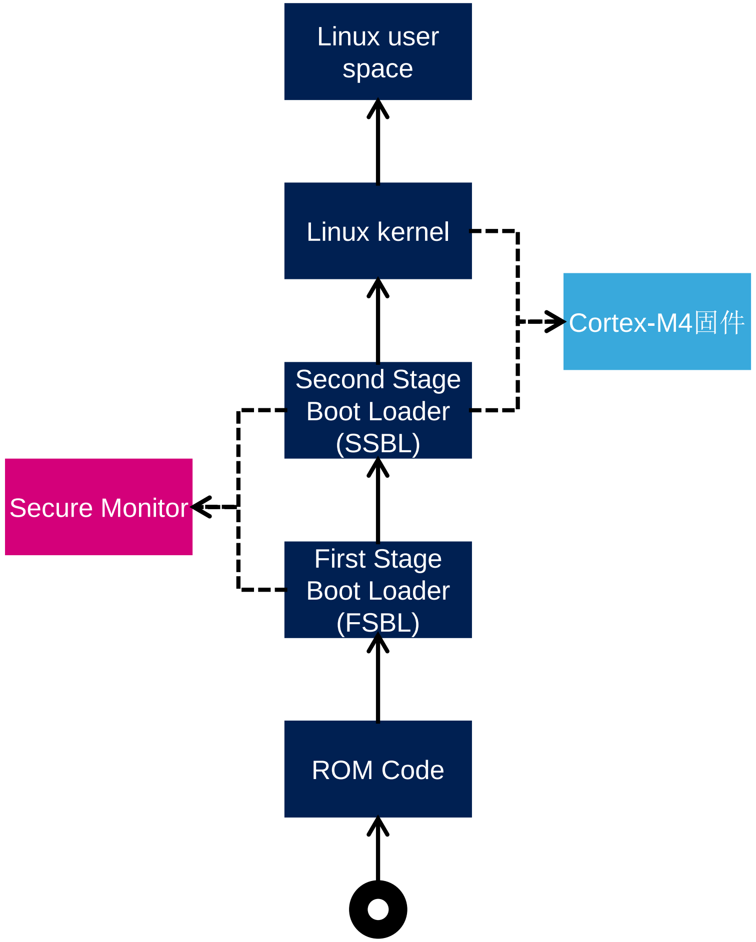
- 从存储器或者以太网加载启动文件系统(bootfs)
- 显示开机画面boot loader splash screen
- 加载Linux kernel (uImage)以及设备树device tree blob (*.dtb)

- 完成时钟树初始化
- 初始化外部RAM (DDR, LpDDR) 控制器
- 从启动设备（存储器或者串口）加载SSBL
- 加载SSBL

- 基本时钟树初始化
- 从启动设备（存储器或者串口）加载第一级启动加载程序FSBL
- 启动FSBL

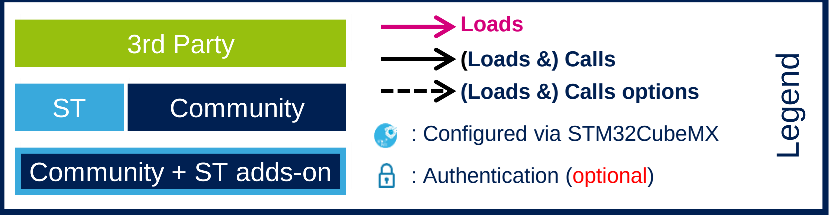
Linux启动如同发射火箭，是一个多级启动流程。

STM32MP1 启动流程



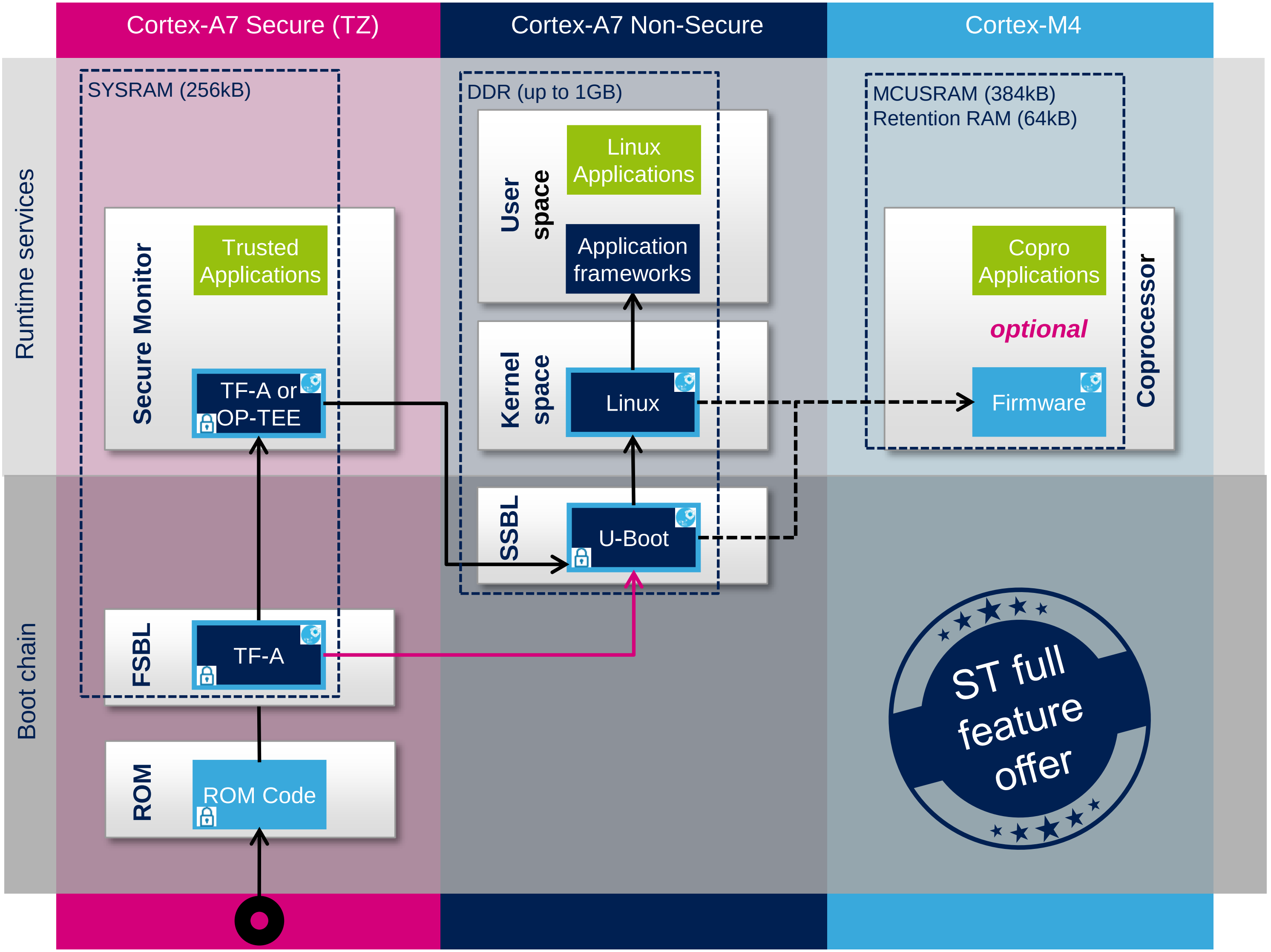
- Secure Monitor可以由FSBL或者SSBL启动
- Cortex-M4固件可以由SSBL或者Linux启动

STM32MP1安全启动流程



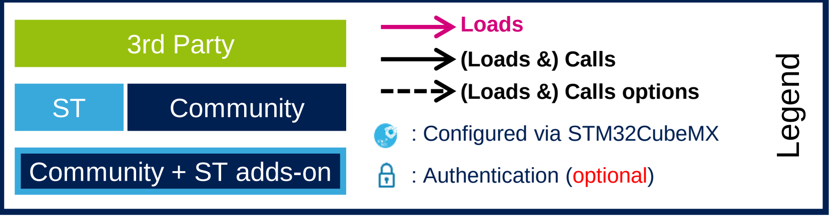
TF-A
- BSD许可，用户可根据需要定制
- ARM为信任执行而开发，适合安全敏感场景
- 功能已被ARMv8平台所验证

U-Boot
- GPL v2许可



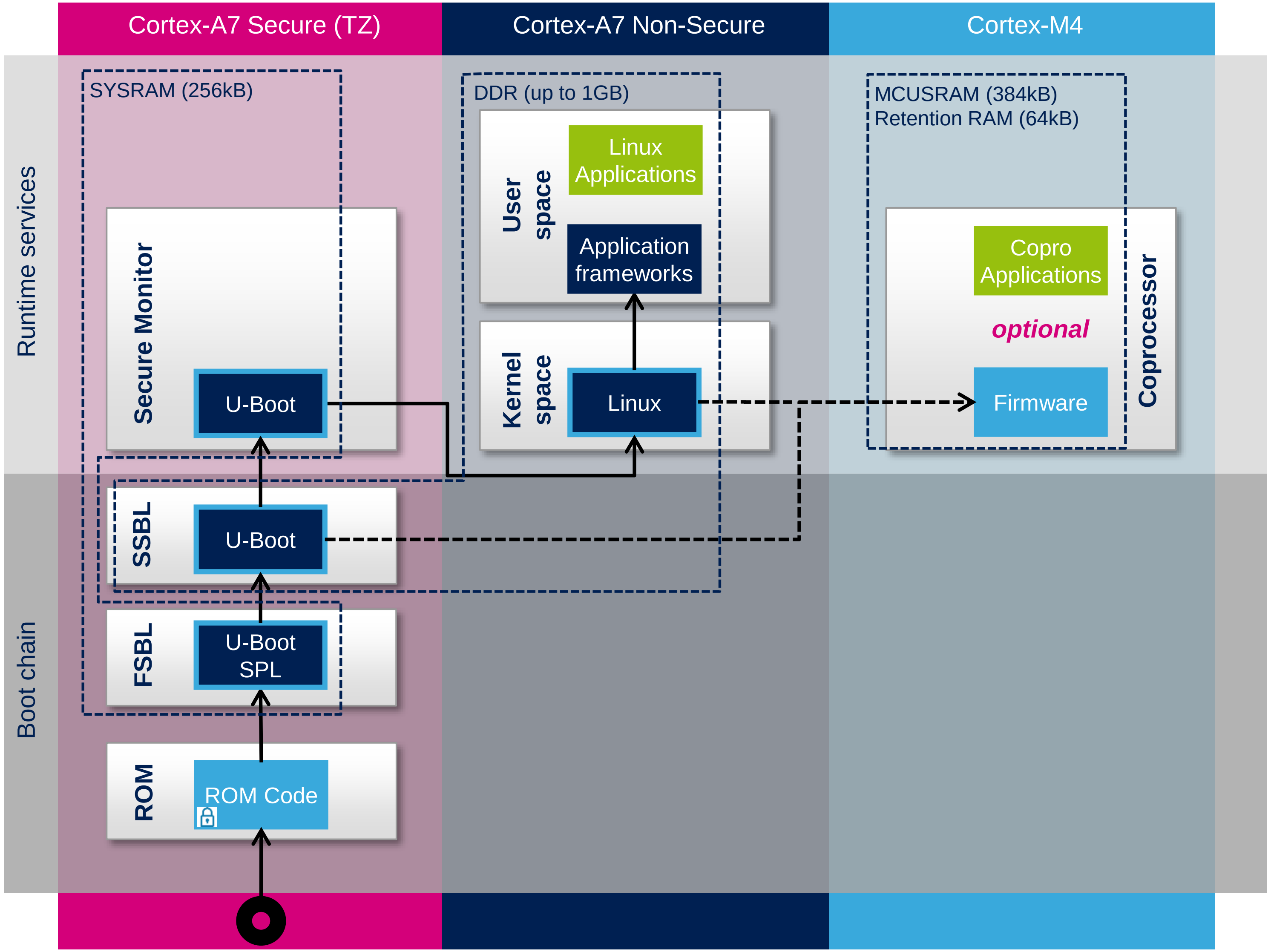
- 安全启动流程是默认方案
- Secure Monitor: 如果没有Secure OS (OP-TEE is optional)，则使用TF-A

life.augmented



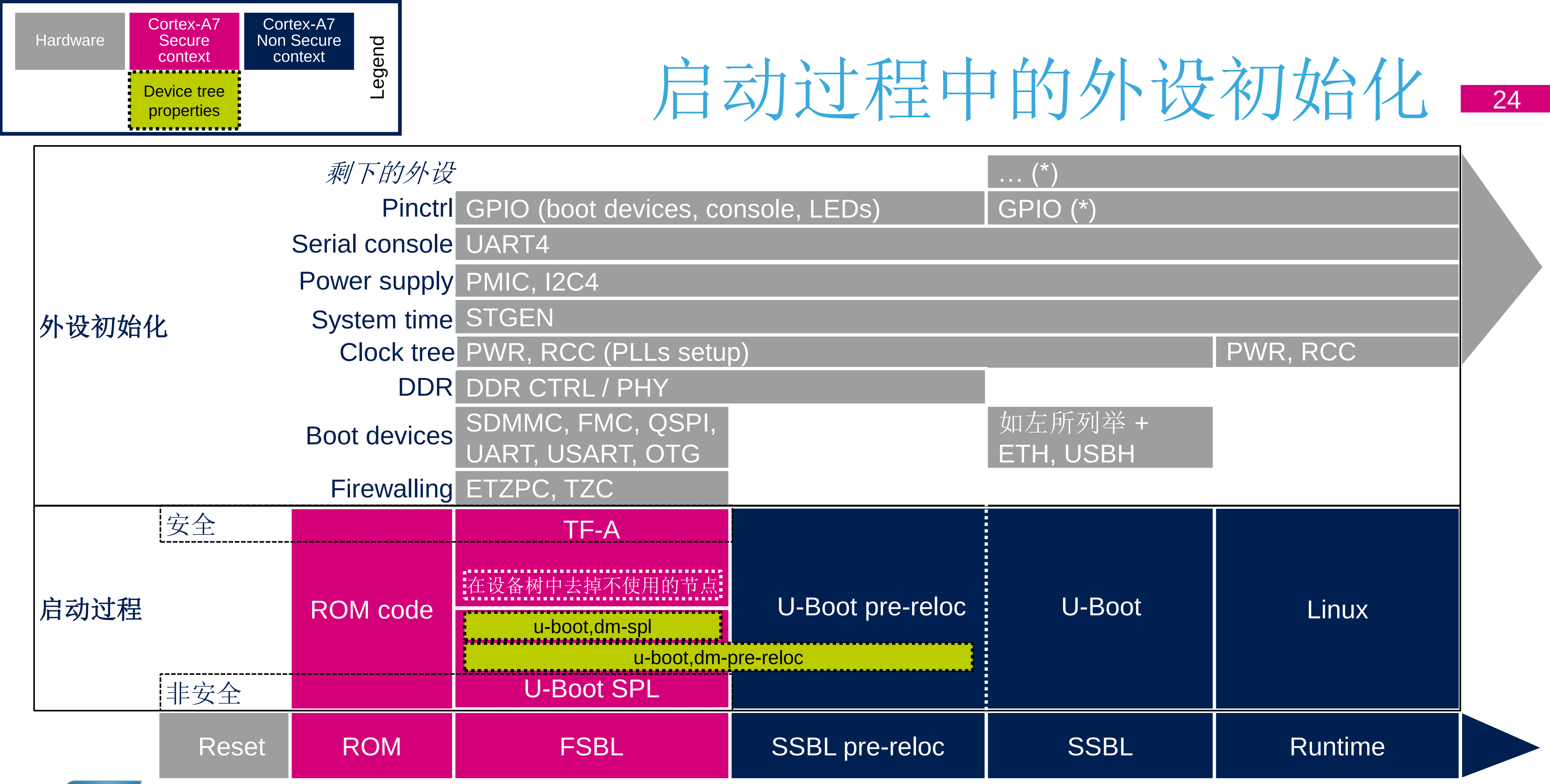
STM32MP1非安全启动流程

- U-Boot SPL来自同一U-Boot源代码，但是不同的配置。较小的U-Boot SPL用来加载较大的U-Boot。



life.augmented

启动过程中的外设初始化



(*) : U-Boot和Linux可以使用同一设备树，但是U-Boot只需一部分外设

STM32MP1启动模式配置

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.

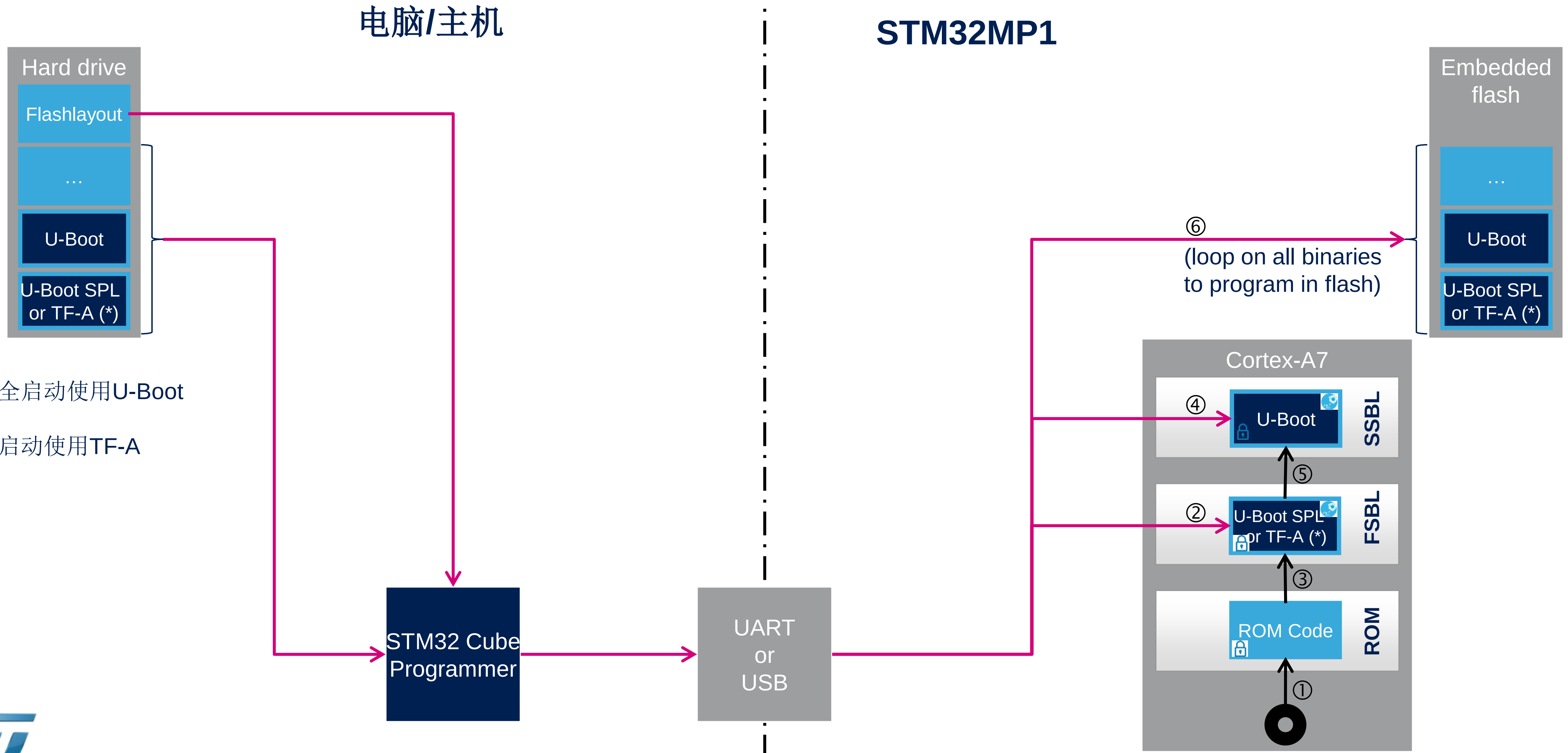
0	No secondary boot source is defined
1	FMC NAND
2	QSPI NOR
3	eMMC
4	SD
5	QSPI NAND

0	No primary boot source is defined
1	FMC NAND
2	QSPI NOR
3	eMMC
4	SD
5	QSPI NAND

life.augmented

例：若强制系统总是SD card启动，则可将OTP WORD 3的Primary boot source设为4

使用STM32CubeProgrammer进行烧录



(*):
- 非安全启动使用U-Boot SPL
- 安全启动使用TF-A

前提： 确保选择串行启动serial boot模式

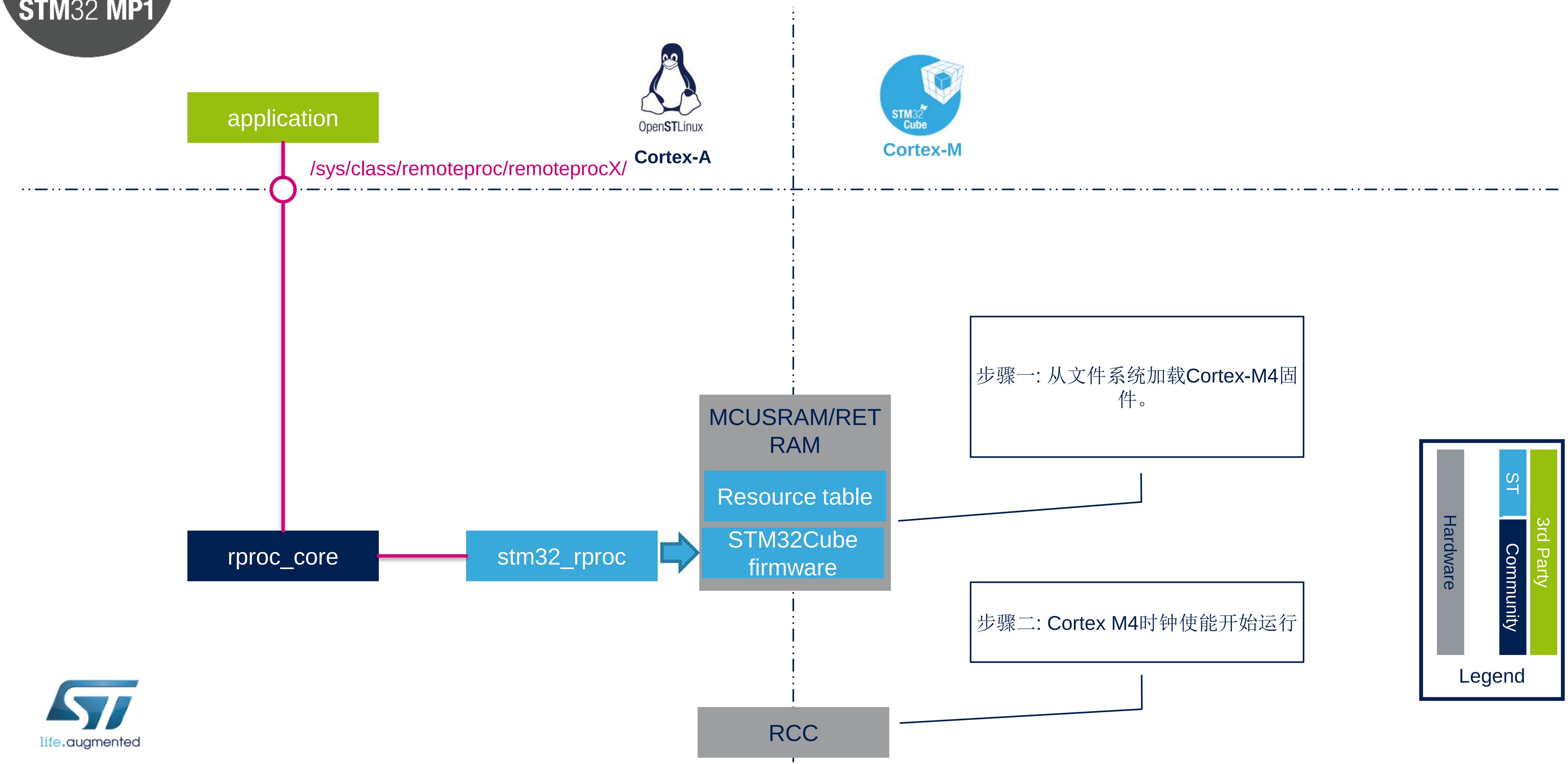
life.augmented

STM32MP1多核通信

多核通信的目标

- Cortex-M4的生命周期管理
- Cortex-A7与Cortex-M4的消息通讯

STM32MP1加载Cortex-M4固件



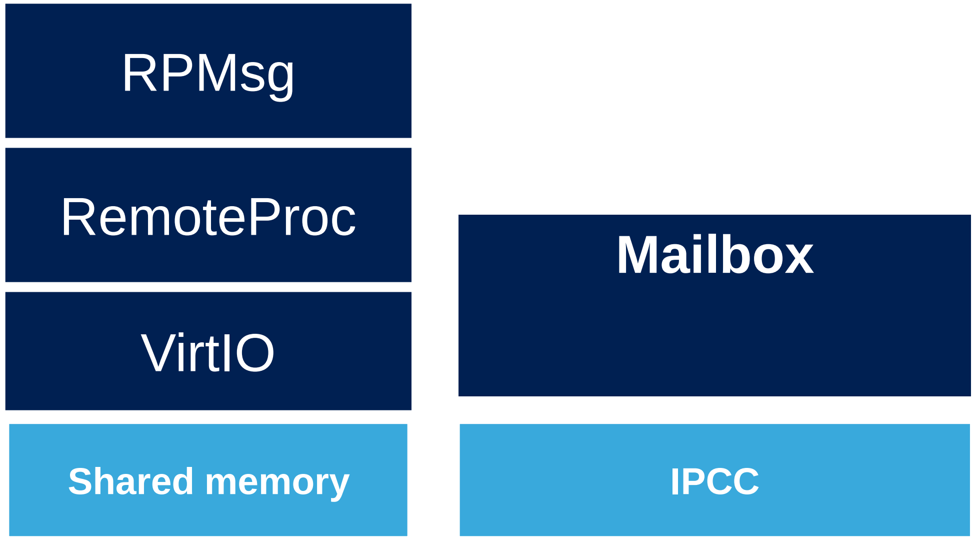
加载Cortex-M4固件

- Cortex-M4固件由Cortex-A使用Linux RemoteProc框架进行加载
- Cortex-M4固件在文件系统中的格式ELF
- 固件中包含Cortex-A7和Cortex-M4都可以访问的资源表“.resource_table”
  - Remote processor trace buffer.
  - Vring 队列和关联的Buffer
- 加载固件的两种方式
  - 自动: 在Linux启动时，lib/firmware存在设备树定义的名字
  - 手工: 使用sysfs接口 (推荐，因为文件系统可以后加载）

注意：根据需要可以在U-Boot中加载Cortex-M4固件

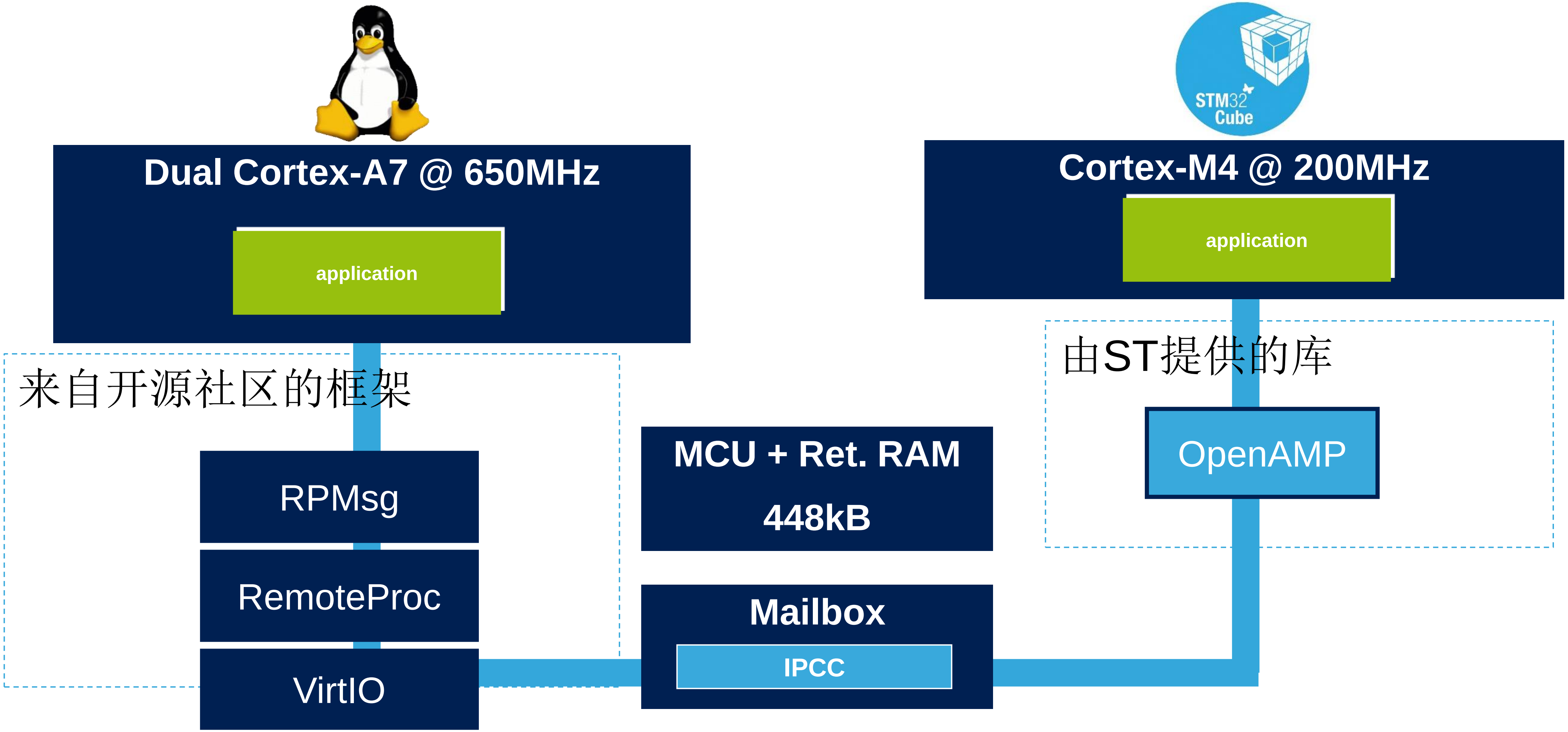
多核消息通信通用框架

- 消息服务基于共享内存，使用RPMsg和Virtio框架
- 信号通知/mailbox服务则基于内部IPCC外设

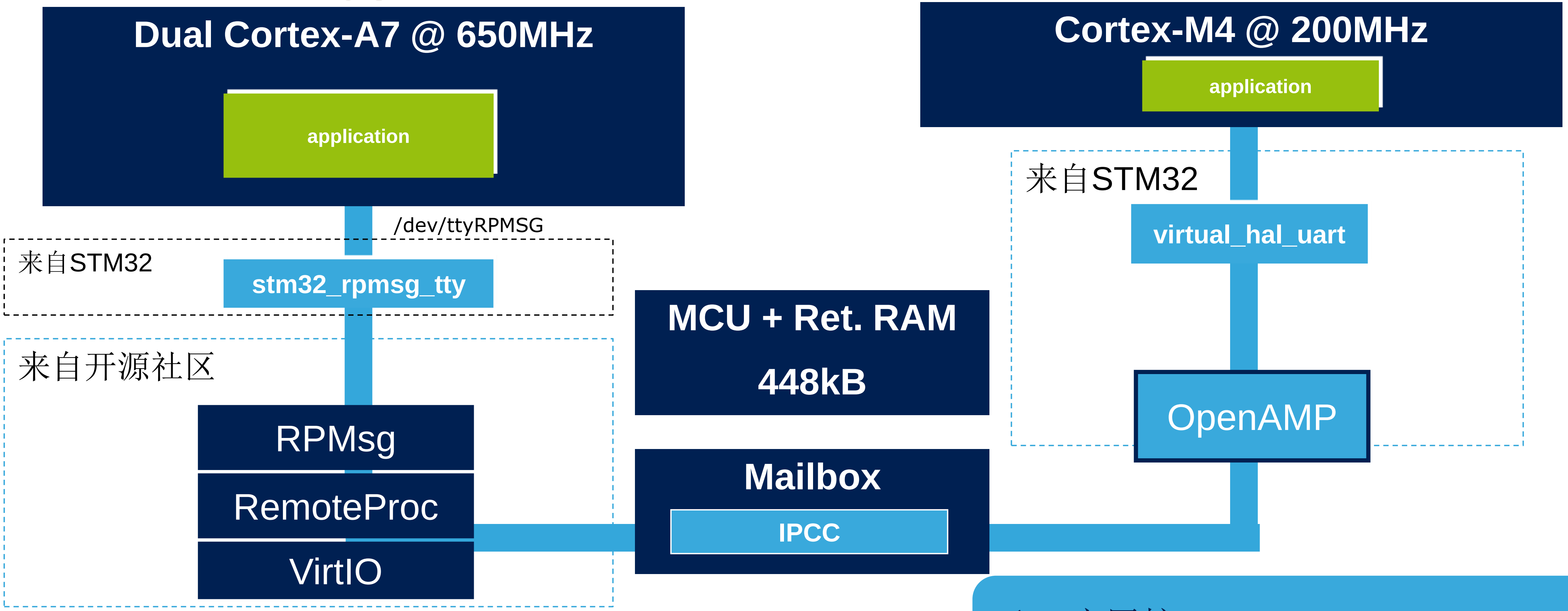


life.augmented

STM32MP1多核通信框架



STM32MP1多核消息通信应用接口



✓A7应用接口：/dev/ttyRPMSG
✓M4应用接口：virtual_hal_uart

STM32MP1软件架构总结

- STM32MP1 Flash内存布局
- STM32MP1 RAM内存布局
- STM32MP1 多核软件架构
  - Cortex-A7
    - Secure
    - Non-secure
      - Bootloader
      - Kernel space
      - User space
  - Cortex-M4
- STM32MP1启动流程
- STM32MP1多核通信