Embedded Linux

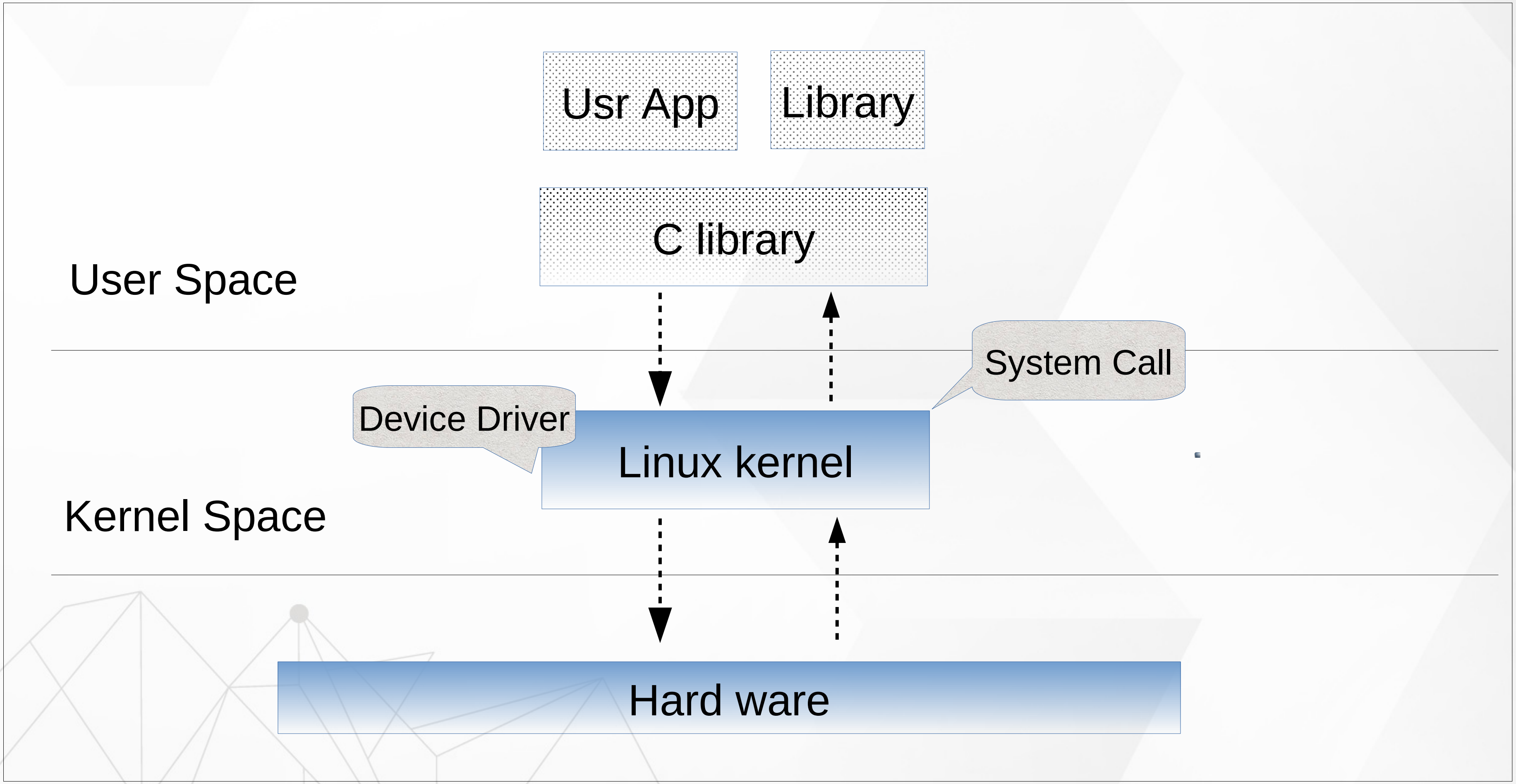
Various Layers within Linux

Various layers within Linux, also showing separation between the userland and kernel space

<table>
<tr><td rowspan="3">User mode</td><td>User applications</td><td colspan="5">bash, LibreOffice, GIMP, Blender, 0 A.D., Mozilla Firefox, ...</td></tr>
<tr><td>System components</td><td>init daemon: OpenRC, runit, systemd...</td><td>System daemons: polkitd, smbd, sshd, udevd...</td><td>Window manager: X11, Wayland, SurfaceFlinger (Android)</td><td>Graphics: Mesa, AMD Catalyst, ...</td><td>Other libraries: GTK, Qt, EFL, SDL, SFML, FLTK, GNUstep, ...</td></tr>
<tr><td>C standard library</td><td colspan="5"><code>fopen</code>, <code>execv</code>, <code>malloc</code>, <code>memcpy</code>, <code>localtime</code>, <code>pthread_create</code>... (up to 2000 subroutines)
glibc aims to be fast, musl aims to be lightweight, uClibc targets embedded systems, bionic was written for Android, etc. All aim to be POSIX/SUS-compatible.</td></tr>
<tr><td rowspan="3">Kernel mode</td><td rowspan="3">Linux kernel</td><td colspan="5"><code>stat</code>, <code>splice</code>, <code>dup</code>, <code>read</code>, <code>open</code>, <code>ioctl</code>, <code>write</code>, <code>mmap</code>, <code>close</code>, <code>exit</code>, etc. (about 380 system calls)
The Linux kernel System Call Interface (SCI), aims to be POSIX/SUS-compatible[2]</td></tr>
<tr><td>Process scheduling subsystem</td><td>IPC subsystem</td><td>Memory management subsystem</td><td>Virtual files subsystem</td><td>Network subsystem</td></tr>
<tr><td colspan="5">Other components: ALSA, DRI, evdev, klibc, LVM, device mapper, Linux Network Scheduler, Netfilter
Linux Security Modules: SELinux, TOMOYO, AppArmor, Smack</td></tr>
<tr><td colspan="7">Hardware (CPU, main memory, data storage devices, etc.)</td></tr>
</table>

https://en.wikipedia.org/wiki/User_space_and_kernel_space

Embedded Linux System

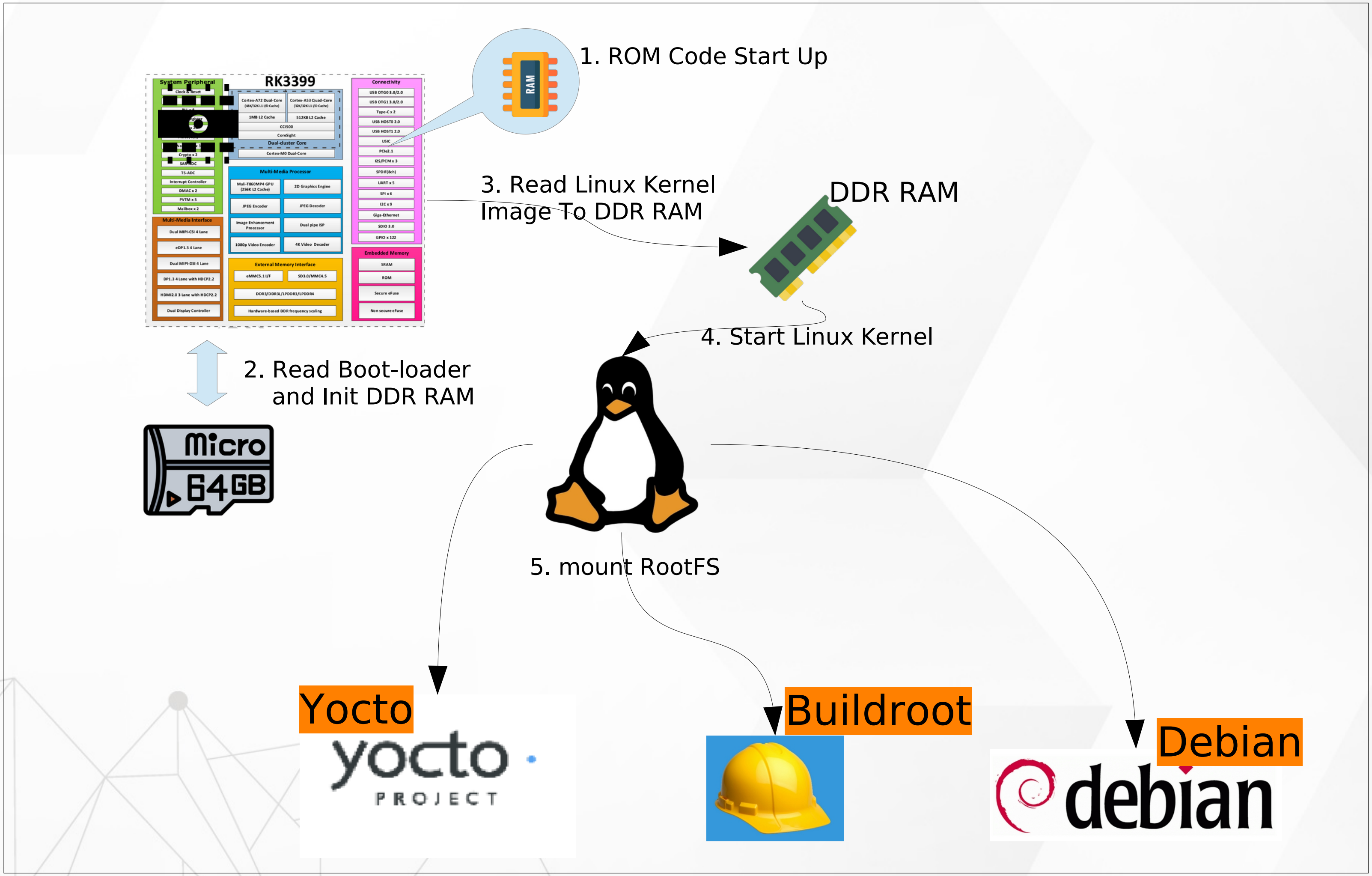


Linux kernel key features

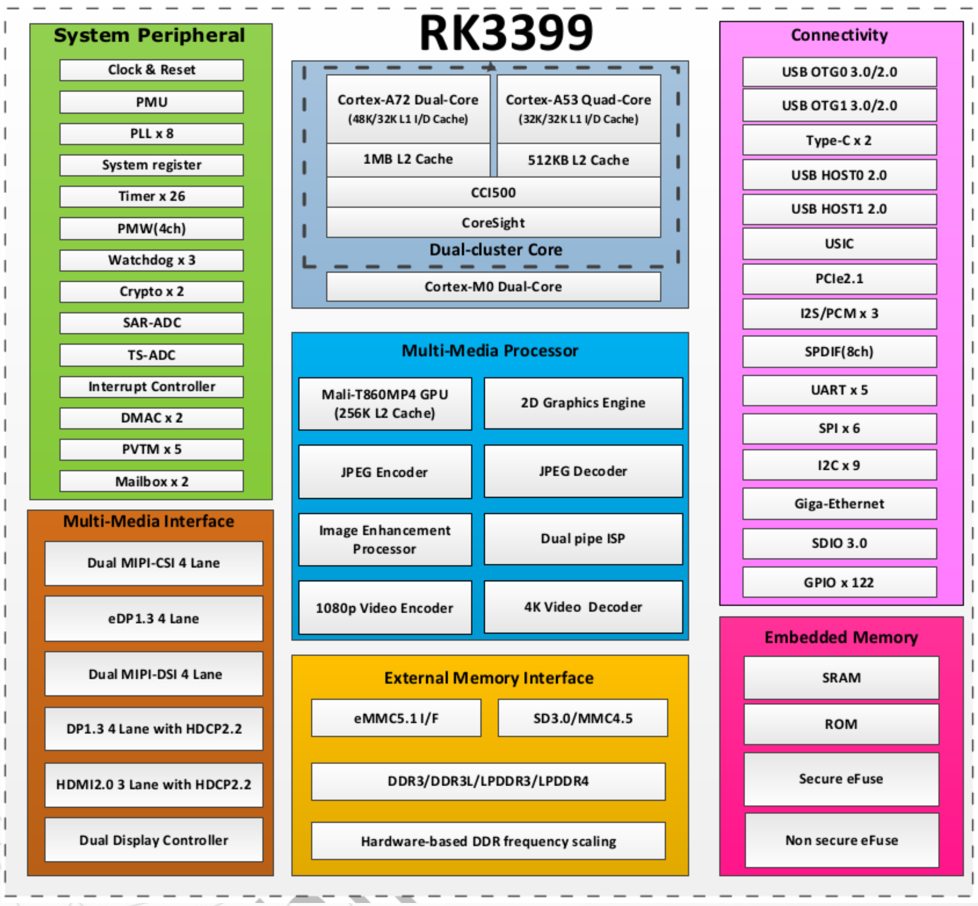
- Portability and hardware support
- Scalability
- Exhaustive networking support
- Stability and reliability
- Modularity
- Easy to program.

System Start Up

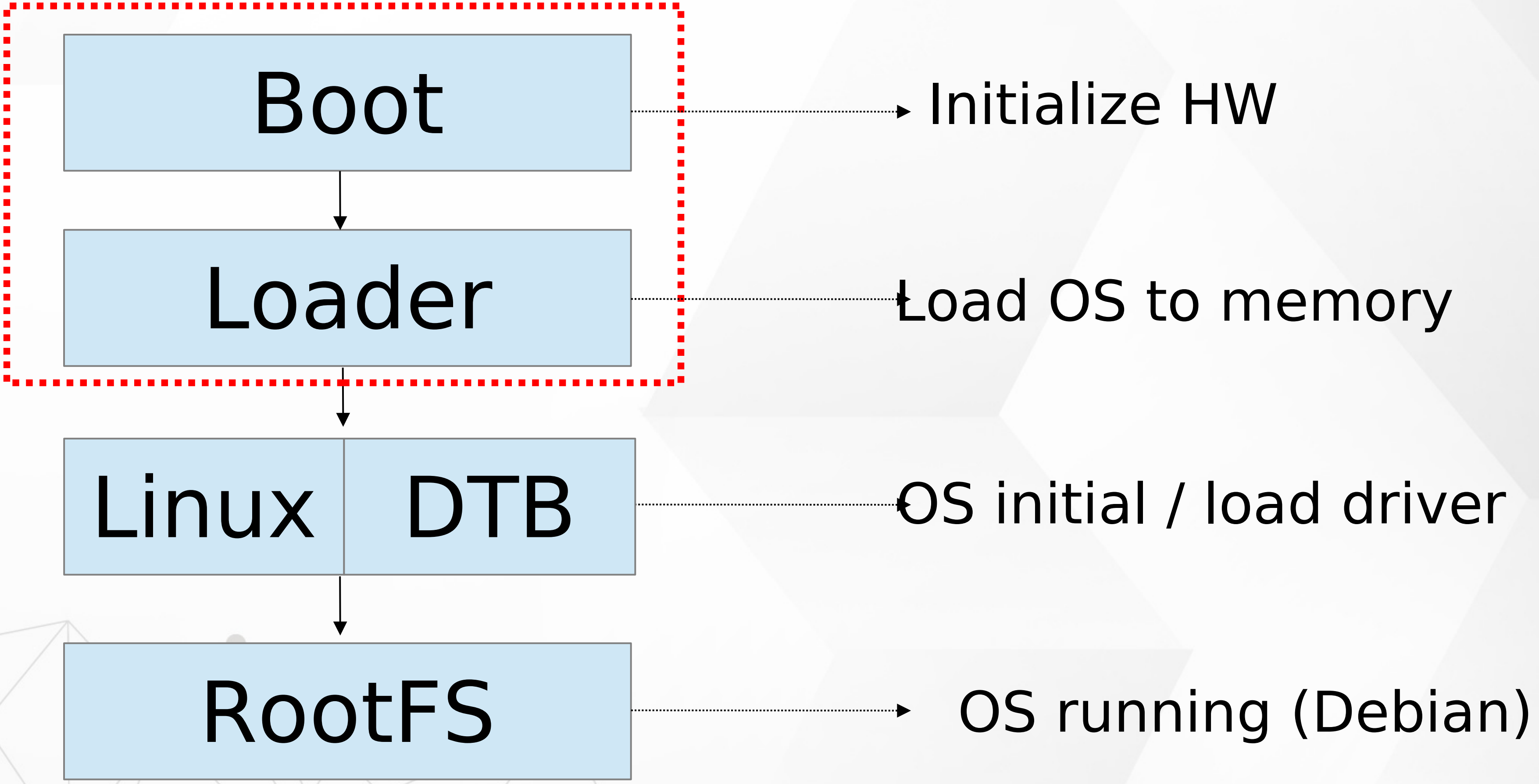
Linux Start Up



SOC RK3399


RK3399
System Peripheral
Clock & Reset
PMU
PLL x 8
System register
Timer x 26
PMW(4ch)
Watchdog x 3
Crypto x 2
SAR-ADC
TS-ADC
Interrupt Controller
DMAC x 2
PVTM x 5
Mailbox x 2
Cortex-A72 Dual-Core
(48K/32K L1 I/D Cache)
Cortex-A53 Quad-Core
(32K/32K L1 I/D Cache)
1MB L2 Cache
512KB L2 Cache
CCI500
CoreSight
Dual-cluster Core
Cortex-M0 Dual-Core
Connectivity
USB OTG0 3.0/2.0
USB OTG1 3.0/2.0
Type-C x 2
USB HOST0 2.0
USB HOST1 2.0
USIC
PCIe2.1
I2S/PCM x 3
SPDIF(8ch)
UART x 5
SPI x 6
I2C x 9
Giga-Ethernet
SDIO 3.0
GPIO x 122
Multi-Media Processor
Mali-T860MP4 GPU
(256K L2 Cache)
2D Graphics Engine
JPEG Encoder
JPEG Decoder
Image Enhancement Processor
Dual pipe ISP
1080p Video Encoder
4K Video Decoder
Multi-Media Interface
Dual MIPI-CSI 4 Lane
eDP1.3 4 Lane
Dual MIPI-DSI 4 Lane
DP1.3 4 Lane with HDCP2.2
HDMI2.0 3 Lane with HDCP2.2
Dual Display Controller
External Memory Interface
eMMC5.1 I/F
SD3.0/MMC4.5
DDR3/DDR3L/LPDDR3/LPDDR4
Hardware-based DDR frequency scaling
Embedded Memory
SRAM
ROM
Secure eFuse
Non secure eFuse

Embedded Linux System Booting



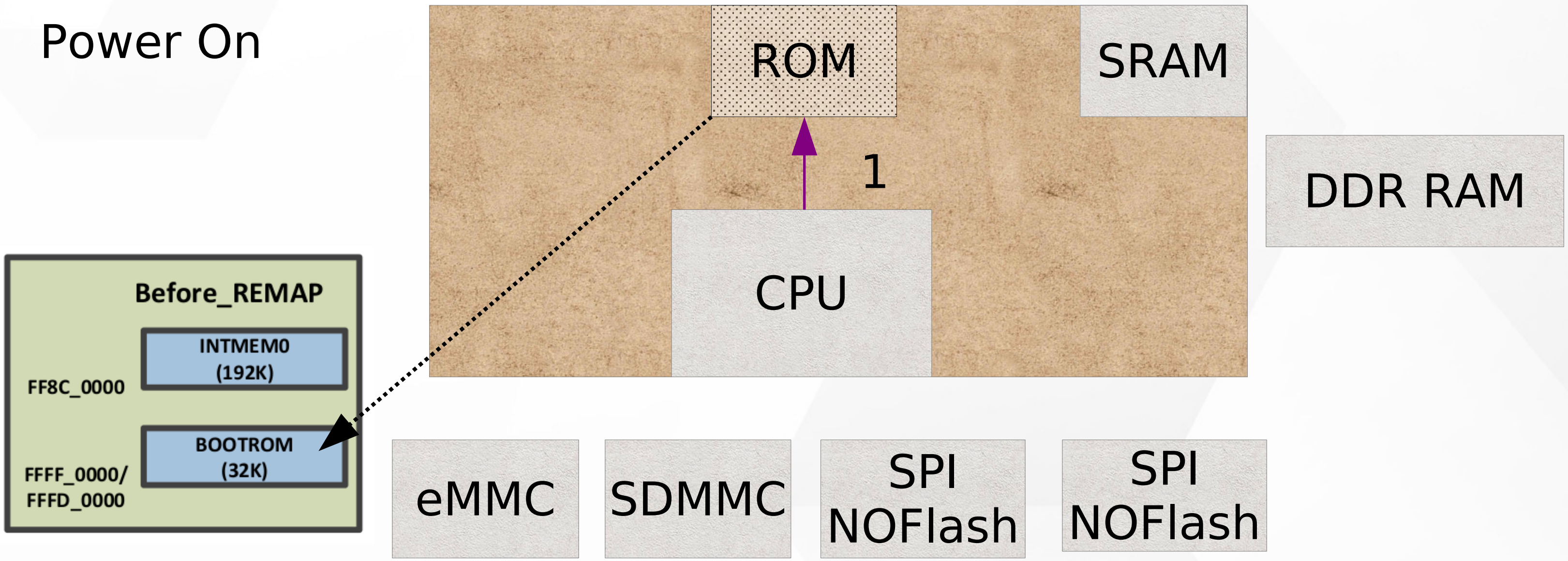
RK3399 System Boot

Reference:

Page 30 of Rockchip_RK3399TRM_V1.3_Part1.pdf

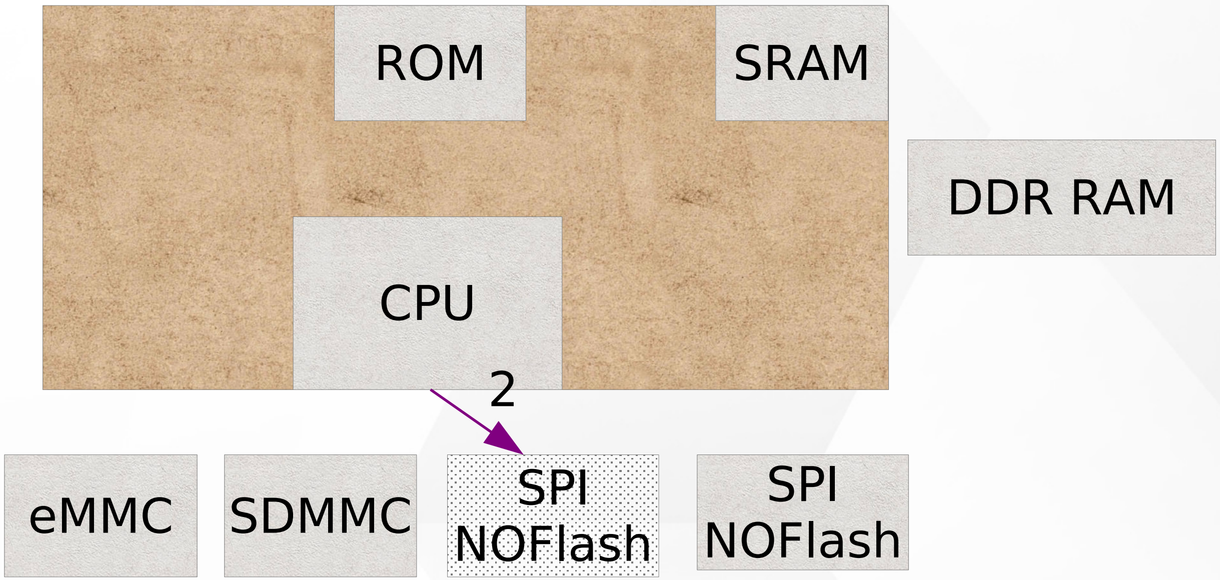
RK3399 System Boot (1)

Power On



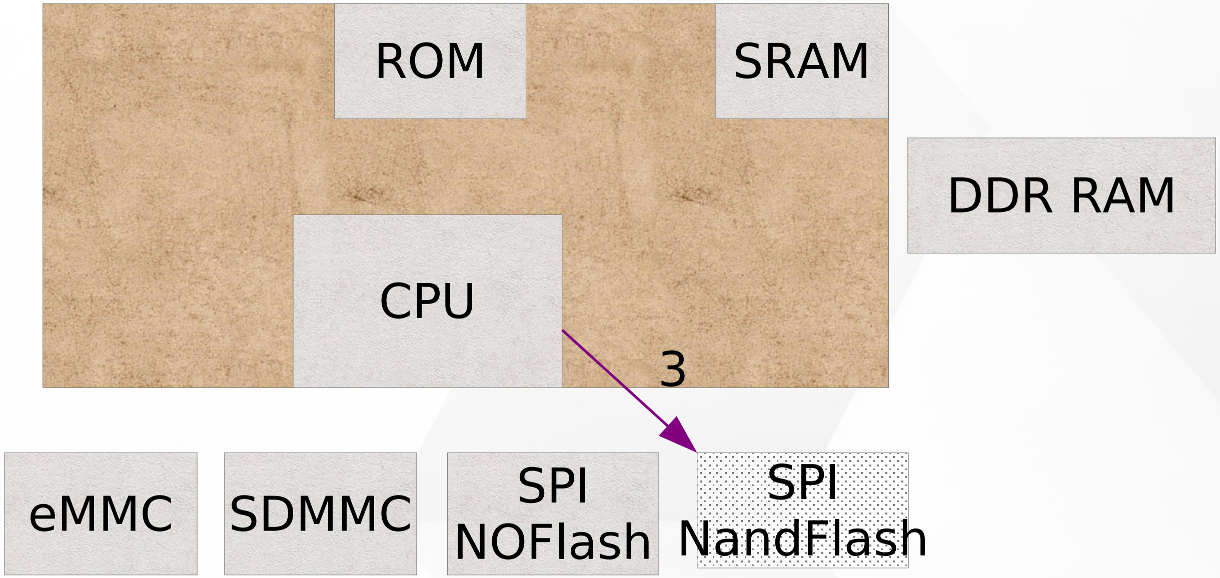
Cortex-A53 get first instruction
from address **0xffff0000 romcode**
start to run

RK3399 System Boot (2)



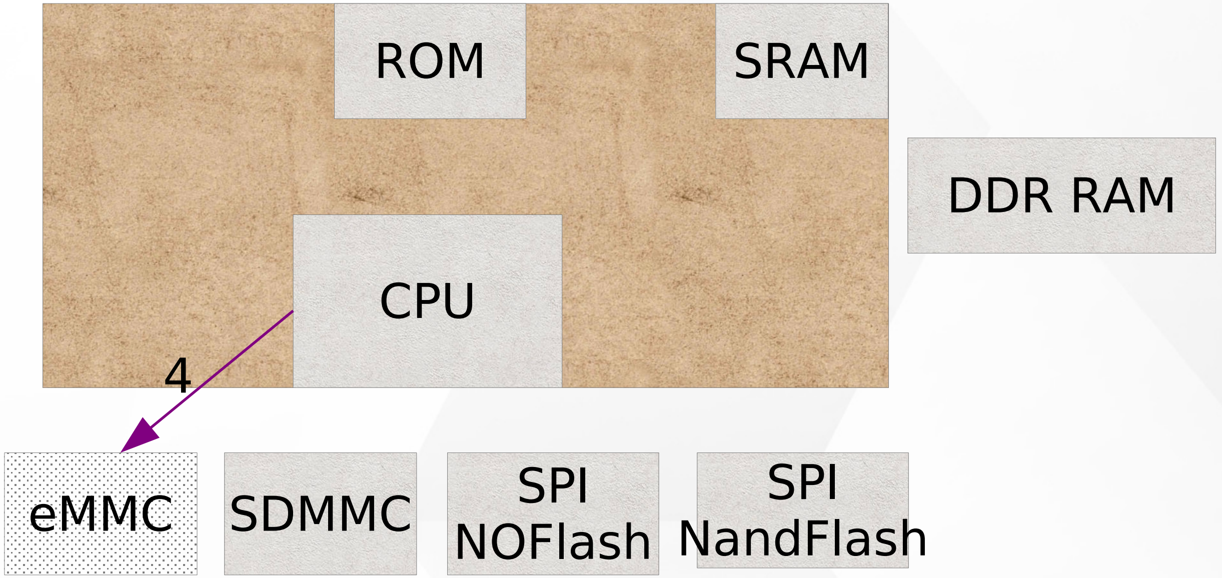
Check ID BLOCK from external **SPI Nor Flash**

RK3399 System Boot (3)



Check ID BLOCK fromexternal **SPI Nand Flash**

RK3399 System Boot (4)



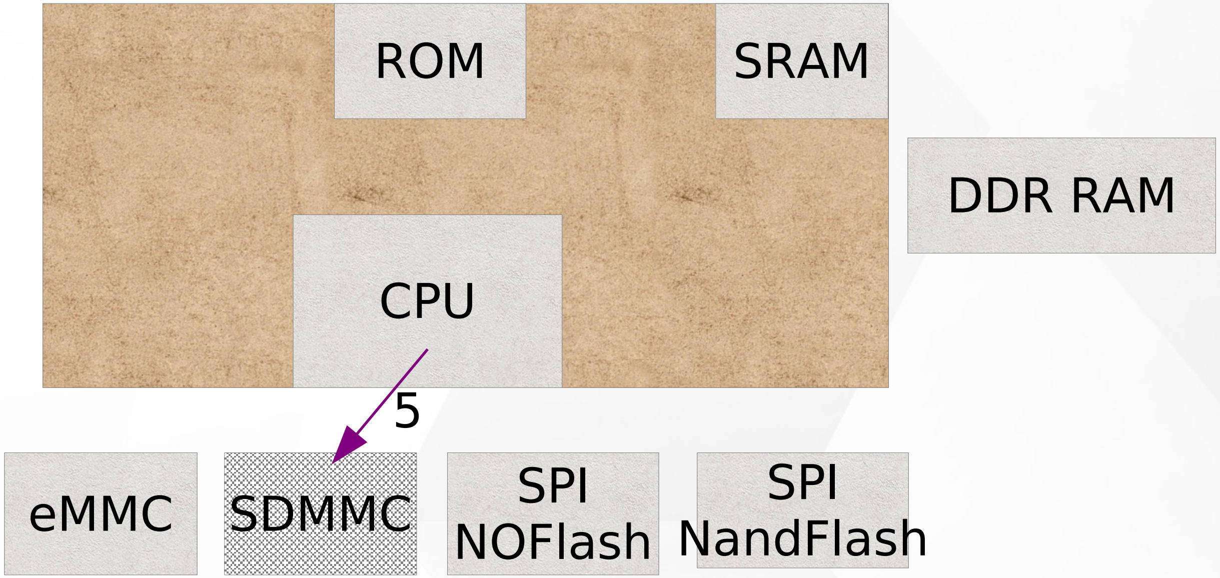
Check ID BLOCK from external **eMMC**

RK3399 System Boot (5)

BL1

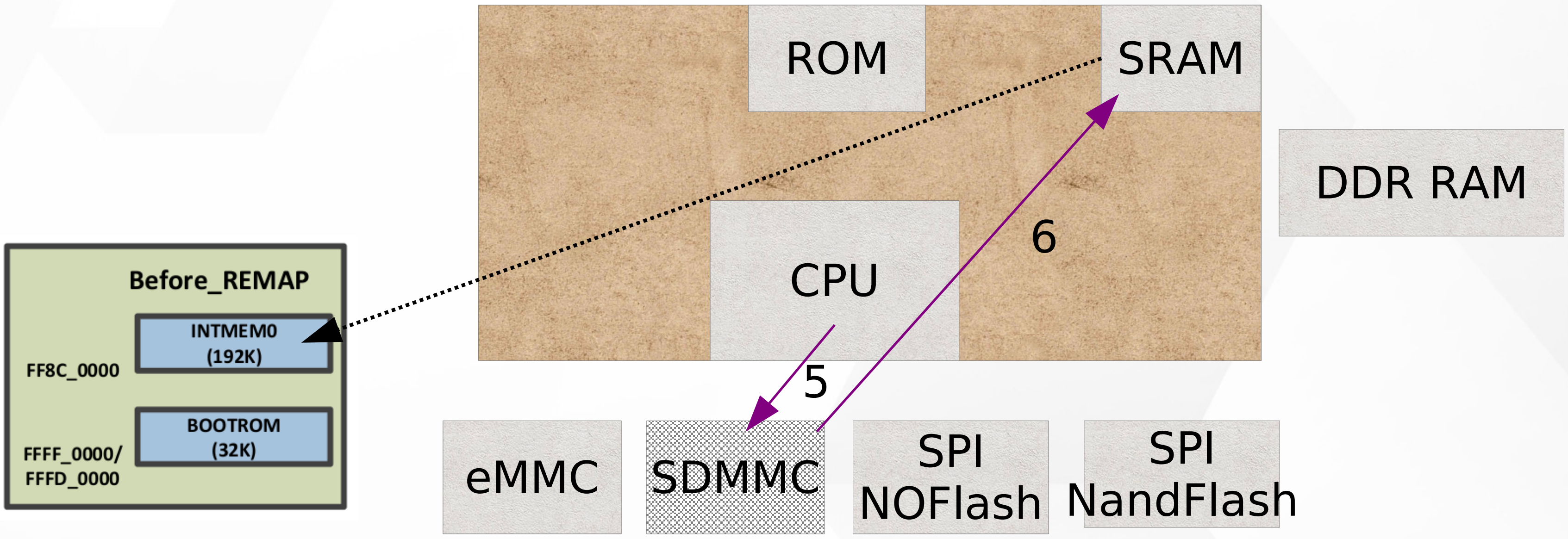
work in cache

IDB_Loader



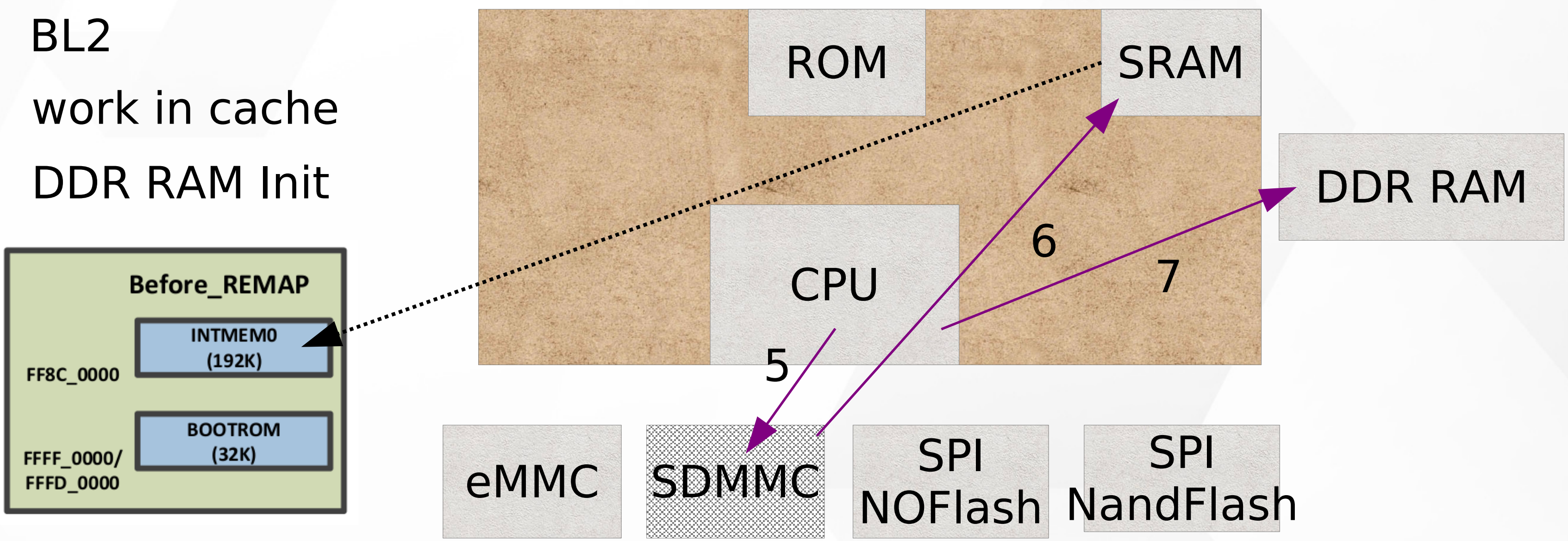
Check ID BLOCK from external **SDMMC**

RK3399 System Boot (6)



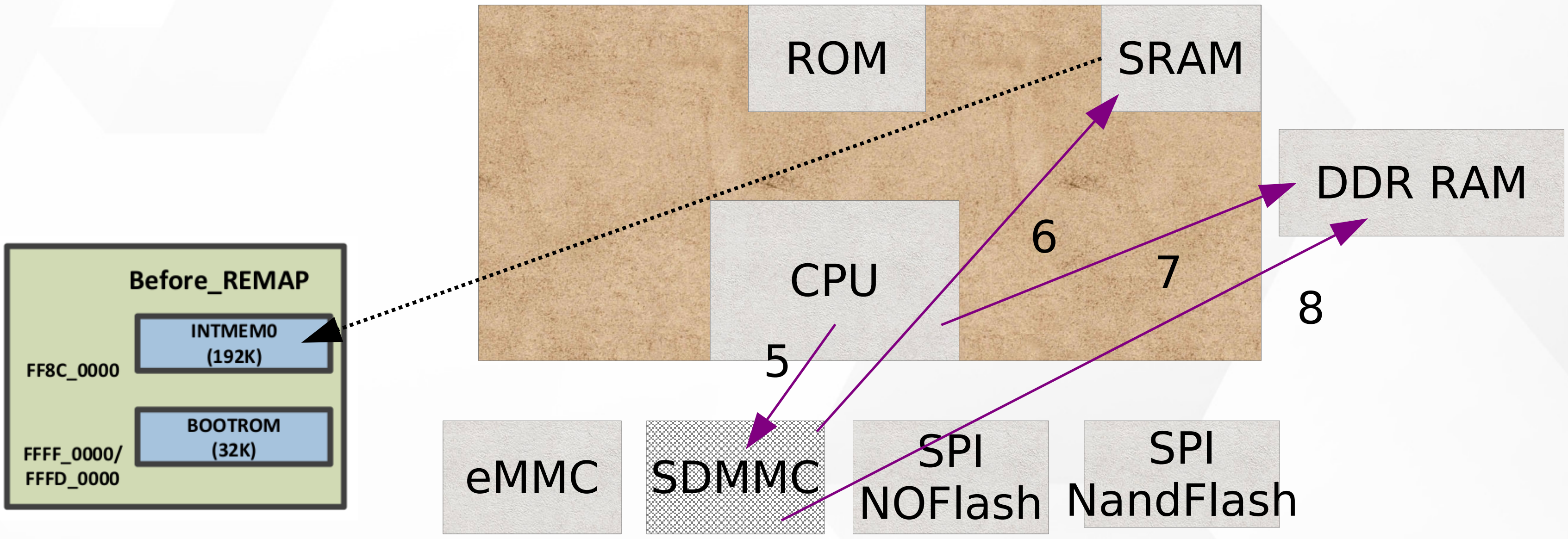
5. Check ID BLOCK from external **SDMMC**
6. Read **2nK** <u>SDRAM initialization image code</u> to **internal SRAM**

RK3399 System Boot (7)



5. Check ID BLOCK from external **SDMMC**
6. Read **2nK** <u>SDRAM initialization image code</u> to **internal SRAM**
7. Run boot code to do DDR initialization

RK3399 System Boot (8)



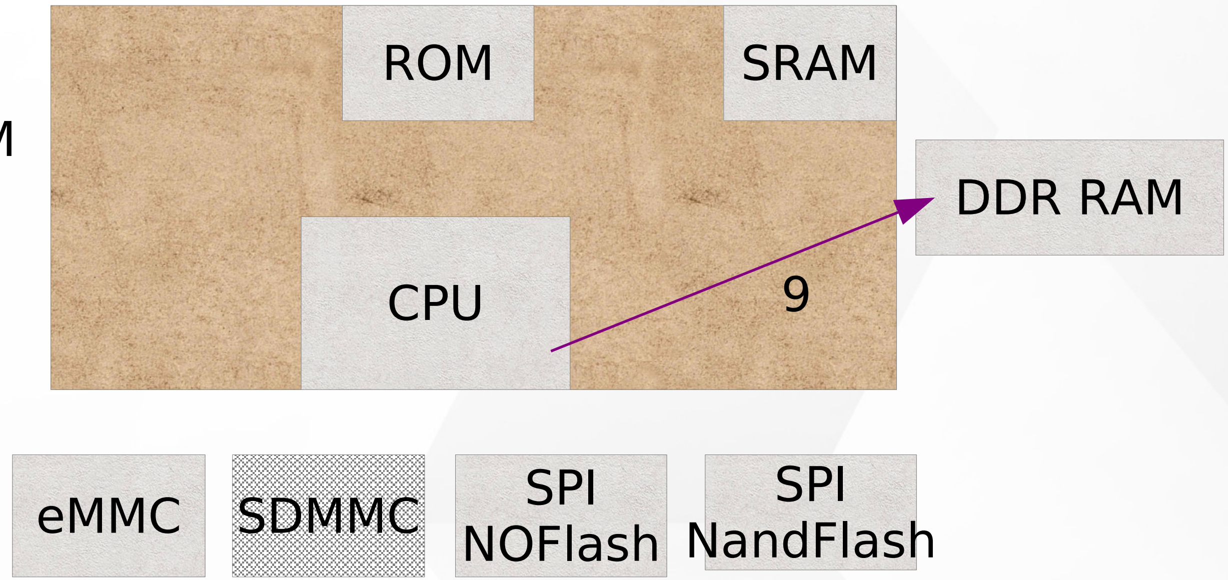
5. Check ID BLOCK from external **SDMMC**
6. Read **2nK** SDRAM initialization image code to **internal SRAM**
7. Run boot code to do DDR initialization
8. Transfer boot code to DDR then Run boot code

RK3399 System Boot (9)

BL3

work in DDR RAM

UBoot

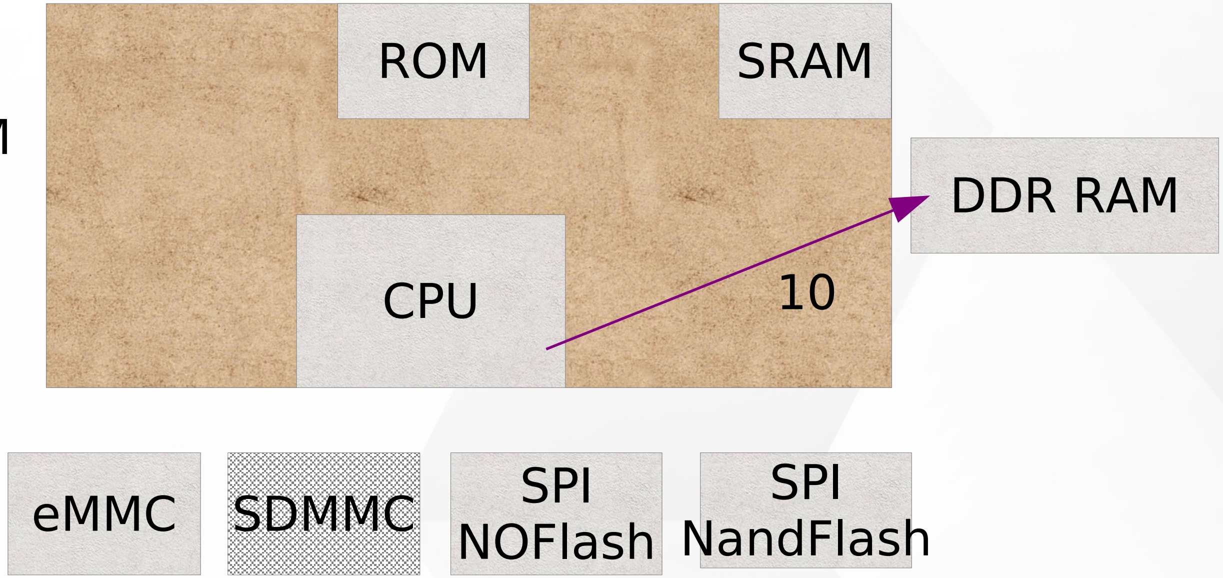


Run UBoot

RK3399 System Boot (10)

work in DDR RAM

Linux



Run Linux

Boot

- Power On BootROM code (work in **cache**)
  - Load BL1
- BL1 (work in **cache** - IDB_Loader)
  - **Initial simple exception vectors, PLL** (clock)
  - **Initial Multi-CPU**
  - Load BL2
- BL2 (work in **cache**)
  - **Initial DDR memory**
  - **Initial C environment** (stack, heap,)
  - Load BL31

Boot

- BL31 (**work in DDR**)
  - Initial exception vectors
  - Load BL32 (u-boot)
- BL32 U-boot (**work in DDR**)
  - **Initial storage device**
  - **Load Linux Kernel**
- Kernel (**work in DDR**)
  - kernel/Documentation/arm64/booting.txt
  - **Load RootFS**

Embedded Linux System

User land

C Library | Qt | OpenCV | GLib

Kernel

Virtual File System (VFS)

Linux System Call

Linux Device Driver

Hardware

Disk | Image Sensor | Keyboard | mouse | Panel