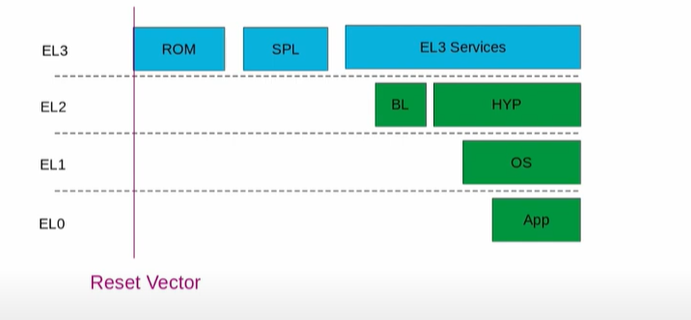
Reference : - https://www.youtube.com/watch?v=GXFw8SV-51g



First it starts with EL3 i.e. the highest exception level.

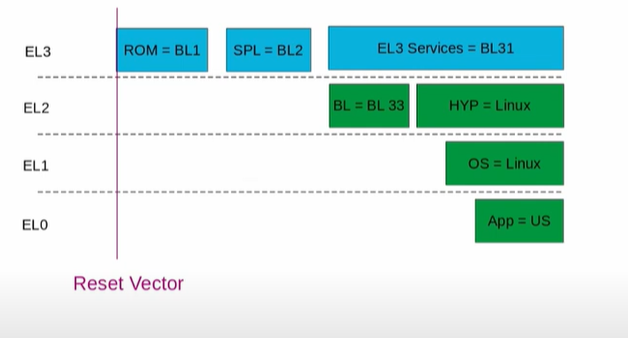
ROM code is vendor implementation, and it is fused into SOC.

SPL – Second stage Bootloader

Then EL3 services run.

The EL3 services then finally starts full strength boot loader. It will then handover to kernel and will provide hypervisor facilities if enabled and put itself into exception level 1.

Finally, kernel started up and user space is available, we will run our application.



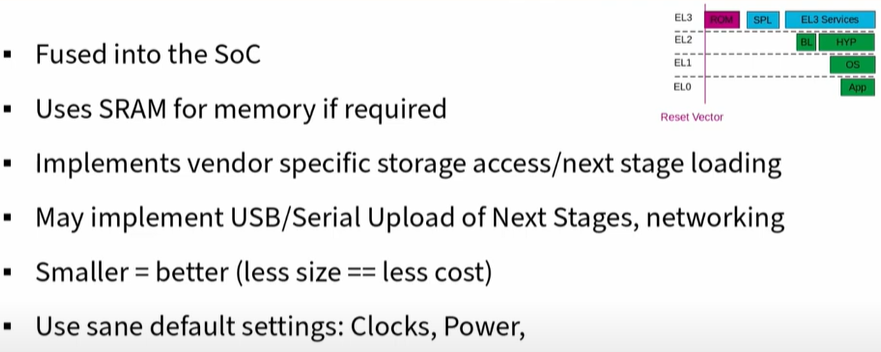
ROM code is also named as BL1. SPL is named as BL2.

EL3 services running in EL3 is named as BL31 i.e. 3rd stage but the first part, which is going to run permanently in EL3, while the kernel is also running parallel in the system.

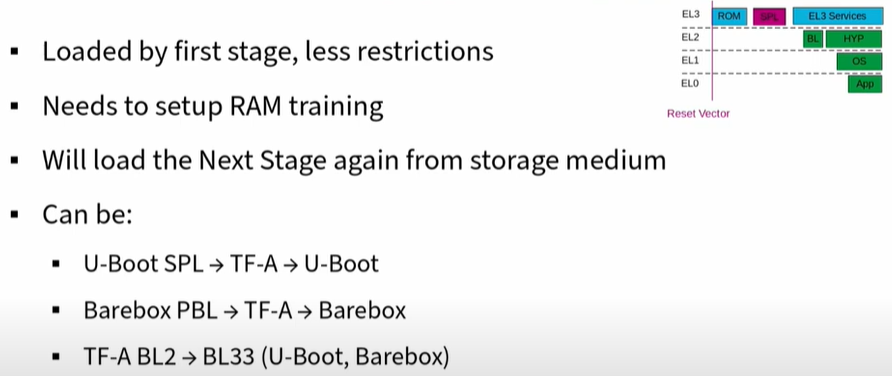
Full strength boot loader (BL) running in EL2 is also named as BL33. Finally the control is handed over Hypervisor (in the form of Linux kernel) or Linux kernel in EL1.

Finally, the application runs. APP=US for User space

# First stage BL1 ROM Code: -

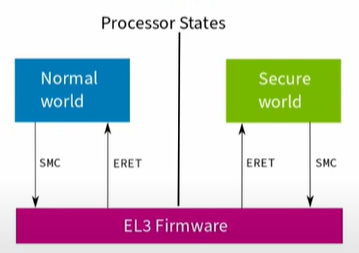


# Second stage (BL2): TF-A/U-Boot SPL/Barebox PBL

****

Boot flow can be: U-Boot SPL -> ARM Trusted Firmware -> U-Boot boot loader

# ARM SMC calling convention



**How do exception levels communicate?**

SMC – Secure Monitor Call

ERET – Exception Return

# Full strength Boot loader: BL33: Bare box proper

Full strength Boot loader BL33 runs in EL2.

Barebox provides additional services: -

* Networking/NFS boot
* Bootspec parsing
* USB Gadget support for: -
* Serial
* Mass storage
* Fastboot

# BL33: Kernel Start

* Decompress the kernel
* AArch64 does not implement decompression in the kernel
* Copy DTB into memory
* Mask interrupts (Boot should not be interrupted)
* Initialize standard ARM timer, but keep its interrupts OFF

# Bl33: Kernel Start 2

* Load kernel at offset defined in header
* Disable MMU and Flush Data cache
* Initialize CPU registers for either EL2 or EL1
* Ex. – X0 for Device Tree Blob
* Jump to kernel