National Yang Ming Chiao Tung University
Computer Architecture & System Lab

Bootloader

IOC5226 Operating System Capstone

Tsung Tai Yeh
Department of Computer Science
National Yang Ming Chiao Tung University

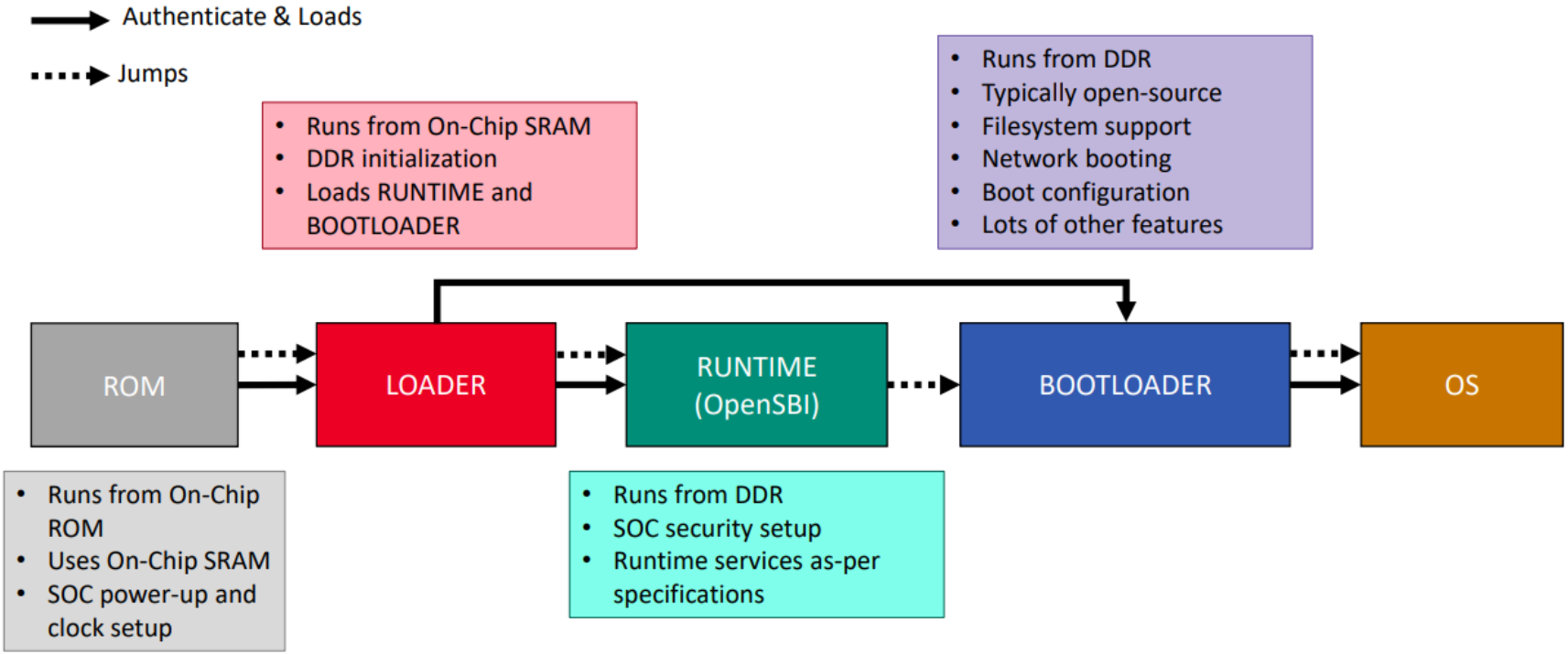
Acknowledgements and Disclaimer

- Slides were developed in the reference with
  - MIT 6.828 Operating system engineering class, 2018
  - MIT 6.004 Operating system, 2018
  - Remzi H. Arpaci-Dusseau etl. , Operating systems: Three easy pieces. WISC

Outline

- BootROM
- OpenSBI
- DeviceTree
- U-Boot

Typical RISC-V Booting Flow



https://reurl.cc/zQ8Avp

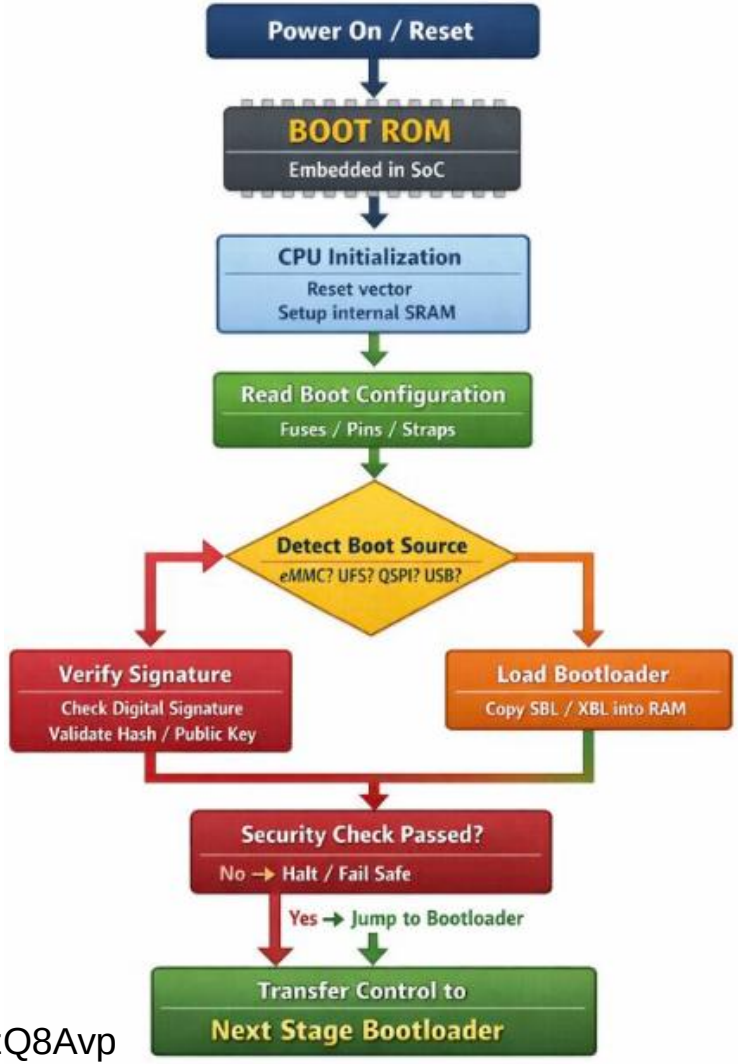
BootROM

- What is BootROM?
  - BootROM (Boot Read-Only Memory)
  - Small and immutable piece of code burned in the SoC silicon
  - The very first code that executes when the SoC powers on or resets
- Where BootROM lives?
  - Inside the SoC die
  - Implemented as Mask ROM

https://reurl.cc/zQ8Avp

BootROM

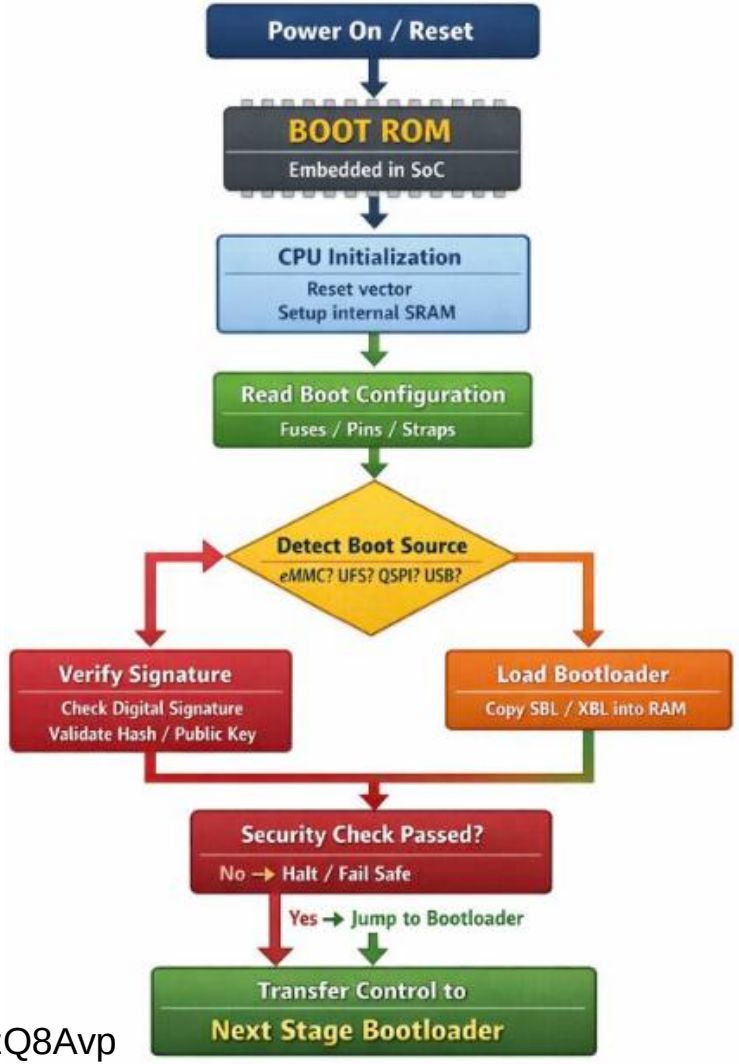
- What are jobs of BootROM?
  - Establish a known starting point
    - CPU starts execution from a fixed reset vector
    - Ensure execution begins from trusted code
  - Minimal hardware initialization
    - Configure CPU execution state
    - Initialize internal SRAM (TCM)
    - Set up clocks need for boot



https://reurl.cc/zQ8Avp

BootROM

- What are jobs of BootROM?
  - Load next stage bootloader
    - Copies next-stage bootloader into SRAM or DDR
    - Jumps to its entry point
    - BootROM execution ends here



https://reurl.cc/zQ8Avp

BootROM

- RISC-V Flat memory model on RootROM
  - BootROM direct mapping address
    - 0x0000_1000: BootROM start address (pc points to here)
    - 0x0200_0000: CLINT (Core Local Interrupt Controller) address
    - 0x8000_0000: External DRAM start address
    - RISC-V directly moves data from the bootloader on the ROM to RAM without passing any registers and translation table

https://reurl.cc/zQ8Avp

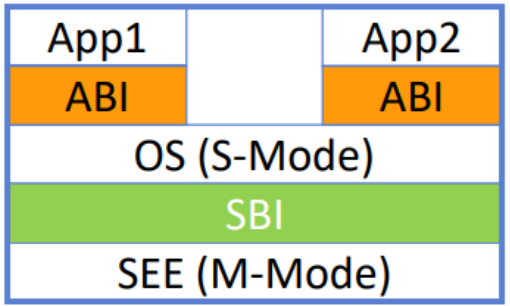
BootROM

- BootROM steps
  - Reset Vector
    - Reset CPU, pc is forced to set the fixed physical address (e.g. 0x1000)
  - Execute BootROM
  - Allocation PMP (Physical Memory Protection)
  - Load and Jump
    - Copy Bootloader to RAM using “jr” instruction
    - BootROM executes on M-mode and need to use “mret” to jump to S-mode by resetting of mstatus.MPP (Machine Previous Privilege mode)

https://reurl.cc/zQ8Avp

OpenSBI

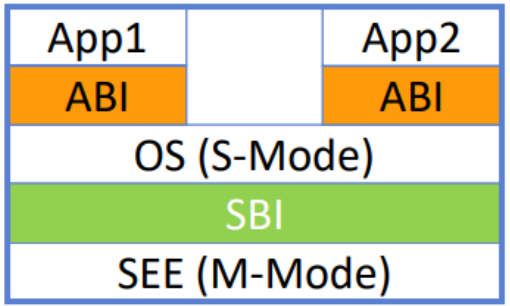
- What is the role of OpenSBI on the booting?
  - SBI serves as a bootloader and firmware for a RISC-V platform
  - Aimed at providing RUNTIME services in M-mode
    - Typically used in boot stage folling ROM/LOADER
- What are jobs of OpenSBI ?
  - Loads further stages of S-mode bootloaders
  - Hosts M-mode traps (timer, pseudo-instruction emulation)
  - Provide service routines (ecall from S-mode)



https://reurl.cc/ovL5Vj

OpenSBI

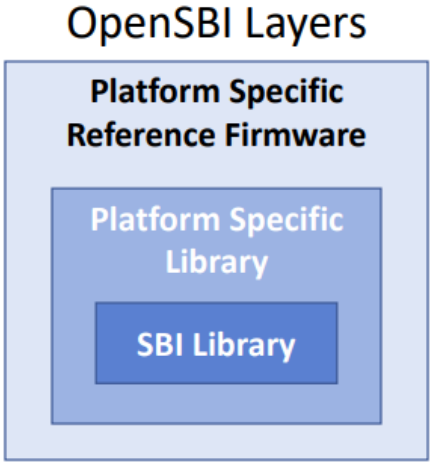
- Provides support for reference platforms
  - Generic simple drivers included for M-mode to operate
    - PLIC, CLINT, UART …
    - Other platforms can reuse the command code and add needed drivers



https://reurl.cc/ovL5Vj

OpenSBI

- Important features of OpenSBI
  - Layered structure to accommodate various use cases
  - Generic SBI library
    - EDK2 (Unified Extensible Firmware Interface (UEFI) implementation)
  - Platform Specific Library
    - Include platform specific drivers
  - Platform specific reference firmware
  - Hardware features supported
    - RV32/64, CSR emulation
    - Protects firmware using PMP support



https://reurl.cc/ovL5Vj

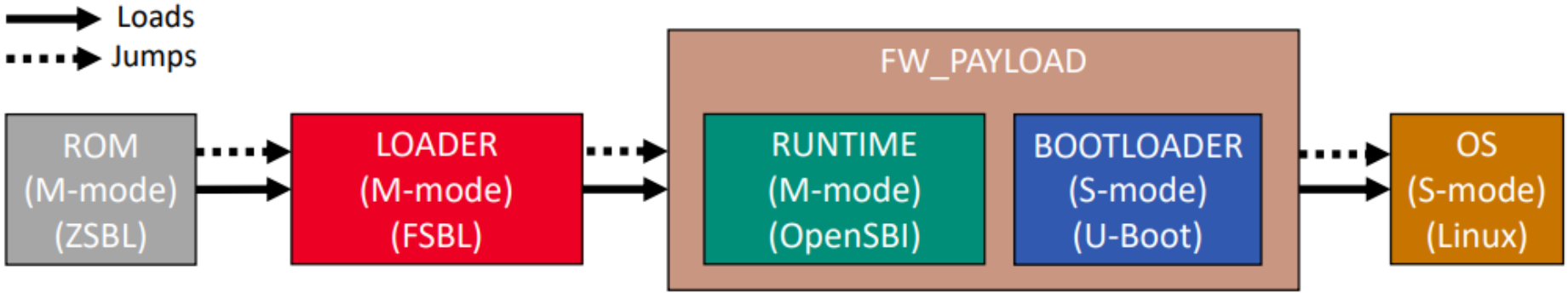
OpenSBI

- Using OpenSBI as reference firmware
  - **FW_PAYLOAD**: with the next booting stage as a payload
  - **FW_JUMP**: with static jump address to the next booting stage
  - **FW_DYNAMIC**: dynamic information on the next booting stage
- SoC vendors may choose
  - Use one of OpenSBI firmwares as M-mode RUNTIME firmware
  - Build M-mode RUNTIME firmware from scratch with OpenSBI as library
  - Extend M-mode firmwares (U-Boot_M_mode/EDK2) with OpenSBI as library

https://reurl.cc/ovL5Vj

OpenSBI

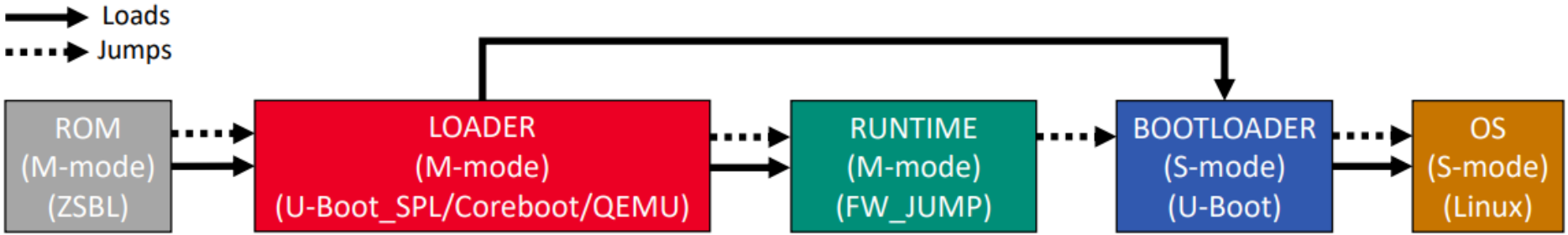
- **FW_PAYLOAD**: with the next booting stage as a payload
  - Any S-mode Bootloader/OS image as payload to OpenSPI FW_PAYLOAD
  - Allow overriding device tree blob (i.e. DTB)



https://reurl.cc/ovL5Vj

OpenSBI

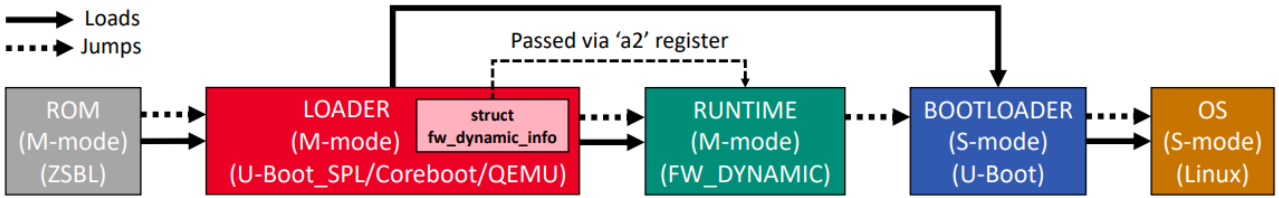
- **FW_JUMP**: with a fixed jump address to next booting stage
  - Next stage booting stage (i.e. BOOTLOADER) and FW_JUMP are loaded by the previous booting stage (i.e. LOADER)



https://reurl.cc/ovL5Vj

OpenSBI

- **FW_DYNAMIC**: with dynamic information about the next booting stage
  - Next stage booting stage (i.e. BOOTLOADER) and FW_DYNAMIC are loaded by the previous booting stage (i.e. LOADER)
  - The previous booting stage (i.e. LOADER) passes the location of struct fw_dymamic_info to FW_DYNAMIC via a2 register



https://reurl.cc/ovL5Vj

Device Tree

- What is DeviceTree?
  - Consist of a tree data structure describing hardware
  - Written in **Device Tree Source (.dts)** files
  - Compiled into **Device Tree Blob (.dtb)** format by the Device Tree Compiler (DTC)
  - The DTB can be directly linked within a bootloader or passed to the OS by the bootloader
  - The device tree is OS-agnostic
    - Found in arch/<ARCH>/boot/dts in Linux Kernel

https://reurl.cc/GaVpQv

Device Tree

- Discoverable vs. Non-Discoverable-Hardware
  - Discoverable Hardware
    - Can enumerate and identify connected devices at runtime
    - E.g. PCI(e) and USB
  - Non-Discoverable Hardware
    - The system must know what devices are connected and how they are configured
    - E.g. I2C and SPI
    - Device Tree is useful on these type of buses

https://reurl.cc/GaVpQv

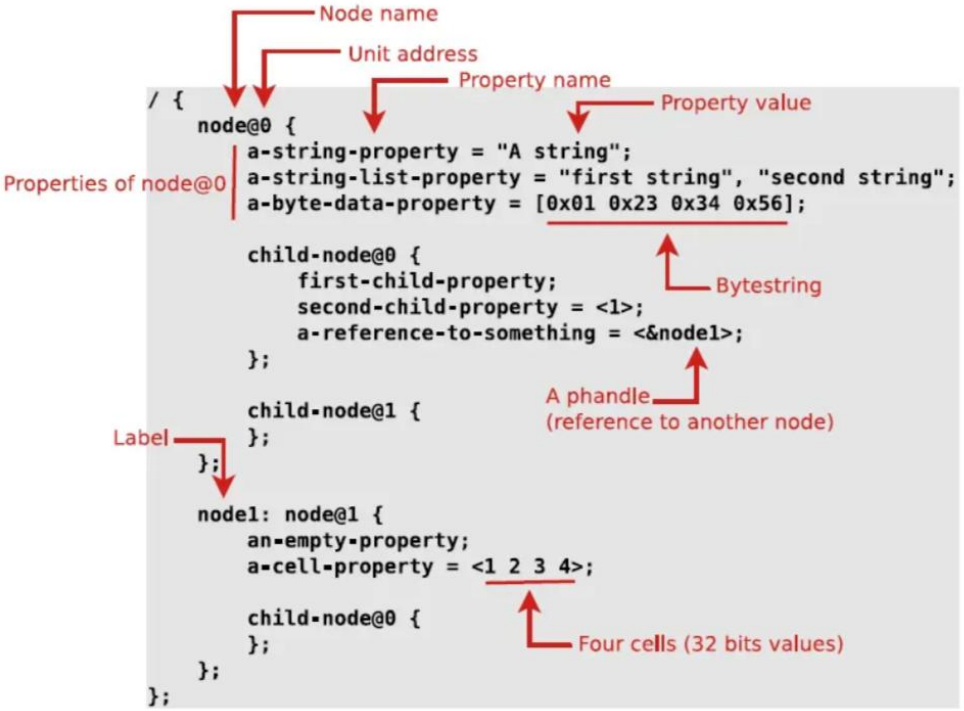
Device Tree

- Device Tree files convey details about the hardware layout, including
  - CPU cores
  - Memory mapped controllers
    - Memory addresses and IRQs of UART, I2C, and other controllers
  - Board-level components
    - E.g. audio codec, clock sources, and reset signals
  - These details make OS or bootloader not need to make a guess

https://reurl.cc/GaVpQv

Device Tree

- Syntax of Device Tree
  - **Nodes:** represent devices or IP blocks
  - **Properties:** define characteristics of devices
  - **Phandle:** Enable reference across the tree

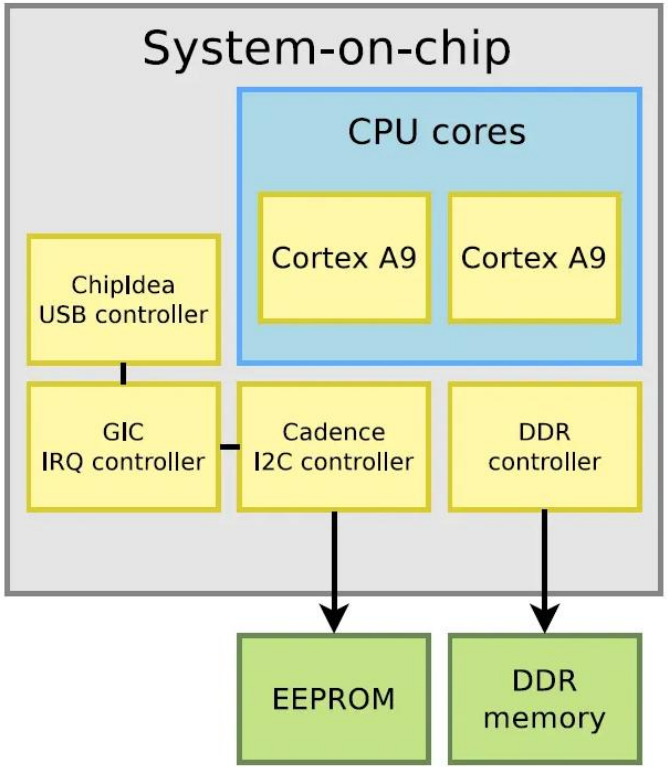


https://reurl.cc/GaVpQv

Device Tree

- A Simple Device Tree example
  - Base Structure

```
/ {
 #address-cells = <1>;
 #size-cells = <1>;
 compatible = "vendor1,board", "vendor2,soc";
 cpus { ... };
 memory@0 { ... };
 chosen { ... };
 soc {
  intc: interrupt-controller@f8f01000 { ... };
  i2c0: i2c@e0004000 { ... };
  usb0: usb@e0002000 { ... };
  };
};
```



https://reurl.cc/GaVpQv

Device Tree

- A Simple Device Tree example
  - CPU/memory

```
/ {
 cpus { ... };
 memory@0 {
  device_type = "memory";
  reg = <0x0 0x20000000>;
  };
 chosen {
  bootargs = "";
  stdout-path = "serial0:115200n8";
  };
  soc {
 intc: interrupt-controller@f8f01000 { ... };
 i2c0: i2c@e0004000 { ... };
 usb0: usb@e0002000 { ... };
 };
};
```

```
/ {
 cpus {
  #address-cells = <1>;
  #size-cells = <0>;
  cpu0: cpu@0 {
  compatible = "arm,cortex-a9";
  device_type = "cpu";
  reg = <0>;
  };
 cpu1: cpu@1 {
  compatible = "arm,cortex-a9";
  device_type = "cpu";
  reg = <1>;
  };
 };
 memory@0 { ... };
 chosen { ... };
 soc {
 intc: interrupt-controller@f8f01000 { ... };
 i2c0: i2c@e0004000 { ... };
 usb0: usb@e0002000 { ... };
 };
};
```

https://reurl.cc/GaVpQv

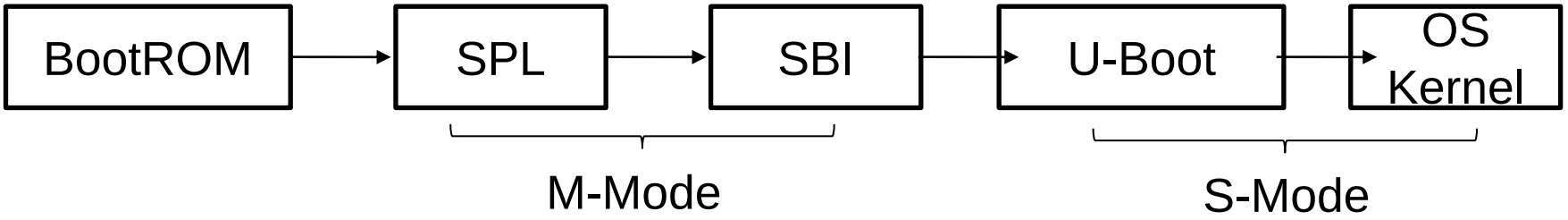
Device Tree

- Key Device Tree Properties
  - **reg:** Define base addresses and register sizes for memory-mapped devices
  - **interrupts:** Specifies the interrupt line used by a device and the interrupt controller it connects to
  - **clocks:** References the clock(s) used by the device
  - **dmas:** Lists DMA controllers and channels
  - **status:** okay indicates the device is active and ready for use

https://reurl.cc/GaVpQv

U-Boot

- What is U-Boot (Universal Bootloader) ?
  - A portable bootloader used on RISC-V/ARM/MIPS … boards
  - Sit between BootROM code and OS kernel
  - Secondary Program Loader (SPL)
    - Initialize DDR and loads U-Boot to DDR



https://reurl.cc/A9q6xe

U-Boot

- In the booting procedure
  - First stage (in silicon): BootROM/PBL (Primary Boot Loader)
    - Boot pins, picks a boot device, read a small image into on-chip SRAM
  - Second stage: U-boot (SPL)
    - Sets up DDR, clocks, PMIC
    - The U-boot SPL loads a FIT image (u-boot.itb), which contains a firmware (fw_dynamic)
    - Providing the SBI and U-Boot (U-Boot) running in S-mode
    - FIT image bundle kernel + fdt + ramdisk with signatures

https://reurl.cc/A9q6xe

U-Boot

- What happens when you press the power button?
  - **Step 0:** Power button -> PMIC & rails
    - PMIC (Power-Management IC) asserts power rails in sequence (core, DDR, I/O)
    - “POWER_GOOD” goes high; reset is released
  - **Step 1:** BootROM executes
    - Read strap pins/eFuses to choose boot media
    - Initializes a tiny interface, reads a small image (U-Boot SPL/FSBL (First Stage Boot Loader)) into on-chip SRAM

https://reurl.cc/A9q6xe

U-Boot

- What happens when you press the power button?
  - **Step 3: SPL minimal init to reach DRAM**
    - Configures clocks (PLLs) and initializes DDR (timings, PHY)
    - Sets up enough drivers to load the next stage from storage (eMMC/SD/Flash) or network/USB
  - **Step 4: SPL loads U-Boot proper**
    - SPL reads U-Boot proper image (full features) and jumps to it in DDR memory
    - Have a rich shell, file system drivers (FAT/ext4), network stack …

https://reurl.cc/A9q6xe

U-Boot

- What happens when you press the power button?
  - **Step 5: U-Boot board init & environment**
    - Probes devices (MMC, USB, Ethernet PHY, PCIe/NVMe)
    - Chooses a boot path via “bootcmd” scripts
    - Read env variables (e.g. bootargs, bootdelay, …)
  - **Step 6: Locate kernel + device tree (+initramfs)**
    - U-Boot loads
      - Kernel (Image/zImage)
      - FDT (.dtb device tree)
      - Optional initramfs (cpio) in DRAM addresses

https://reurl.cc/A9q6xe

U-Boot

- What happens when you press the power button?
  - **Step 7: Launch the kernel (S-mode RISC-V)**
    - U-Boot calls bootm/booti to jump to the kernel entry
    - Passing FDT (Flattened Device Tree), memory map, and bootargs
  - **Step 8: Kernel early boot (mounts rootfs)**
    - Kernel sets up MMU/VM, driver, mount root, pivots to userspace PID 1
  - **Step 9: Userspace & services**
    - Services start (networking, containers, app ..)

https://reurl.cc/A9q6xe

Bootloader on RISC-V Multi-Core

- Hart Lottery on Bootstrap Processor (BSP)
  - RISC-V employs Hart (Hardware Thread) Lottery to pick one core to be BSP
  - After the selection of one BSP, the BSP will do
    - Get Hart ID: use mhartid CSR register to gain its Hart ID
    - Allocate Stack memory for each Hart and calculate offset of stack memory using “mhartid”
    - Initialize OpenSBI for Hart State Management (HSM)
    - When OS (S-mode) needs more core, BSP use SBI ecall (sbi_hsm_hart_start) to warm boot cores

Takeaway Questions

- What are jobs of BootROM?
  - (A) load OS kernel
  - (B) Choose boot media
  - (C) Loads FIT image
- What are jobs of Device Tree?
  - (A) Boot the computer
  - (B) Describe the system hardware layout
  - (C) Perform interrupt system call