Linux Boot Process, systemd, and the Device Tree

- Booting Linux : also see Molloy (2nd Ed.) pp. 74 – 83
  - we will describe how the Linux OS launches on Arm-based embedded platforms
  - for reference, note that on your PC or host machine, which is typically an Intel- or AMD-based system, there is BIOS (Basic Input-Output System) which configures the low-level (device driver) aspects of the system and then launches the OS (Windows, Ubuntu or MacOS).
    → the BIOS (or aspects of it) remain on the system even after the OS launches
  - on your embedded Linux system, there is no BIOS and the boot process is different than on your PC
    → there are three phases:

      PHASE ① : ROM Code
      PHASE ② : Secondary Program Loader (SPL)
      PHASE ③ : third-Stage Program Loader (TPL)

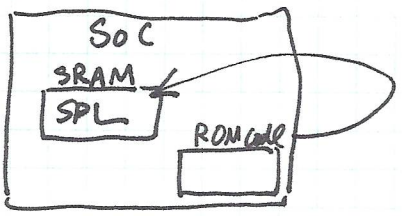
Boot Process on Embedded Linux Platforms

- the bootloader is a major element of embedded Linux.
- the bootloader starts up the system and loads the kernel
- it passes control of the system to the kernel using a data structure called a device tree, also known as a flattened device tree or FDT.

- on power up, the system is in a very minimal state — few resources are available, e.g. the DRAM controller has not yet been set up so main memory is not yet accessible.
- typically, right after power up, the only resources available are a single CPU core and some on-chip static memory.
- as a result, the system boots in stages, with each stage bringing more of system up...

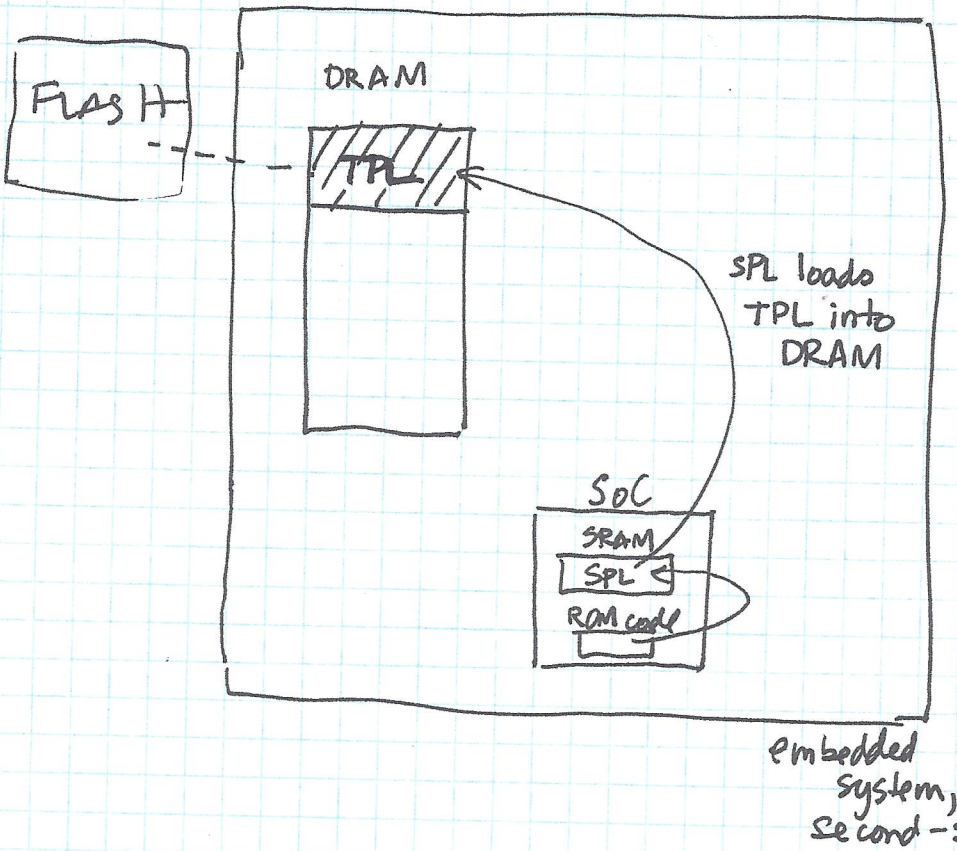
PHASE 1: ROM CODE

- located on the system-on-a-chip (SoC)
- ROM code (on-chip) gets loaded into on-chip static memory (SRAM)
- the size of this SRAM is typically small from 4 kibytes to 128 kibytes
- generally the ROM code loads a secondary bootloader, known as the secondary program loader (SPL) into SRAM



- PHASE 2 : SPL

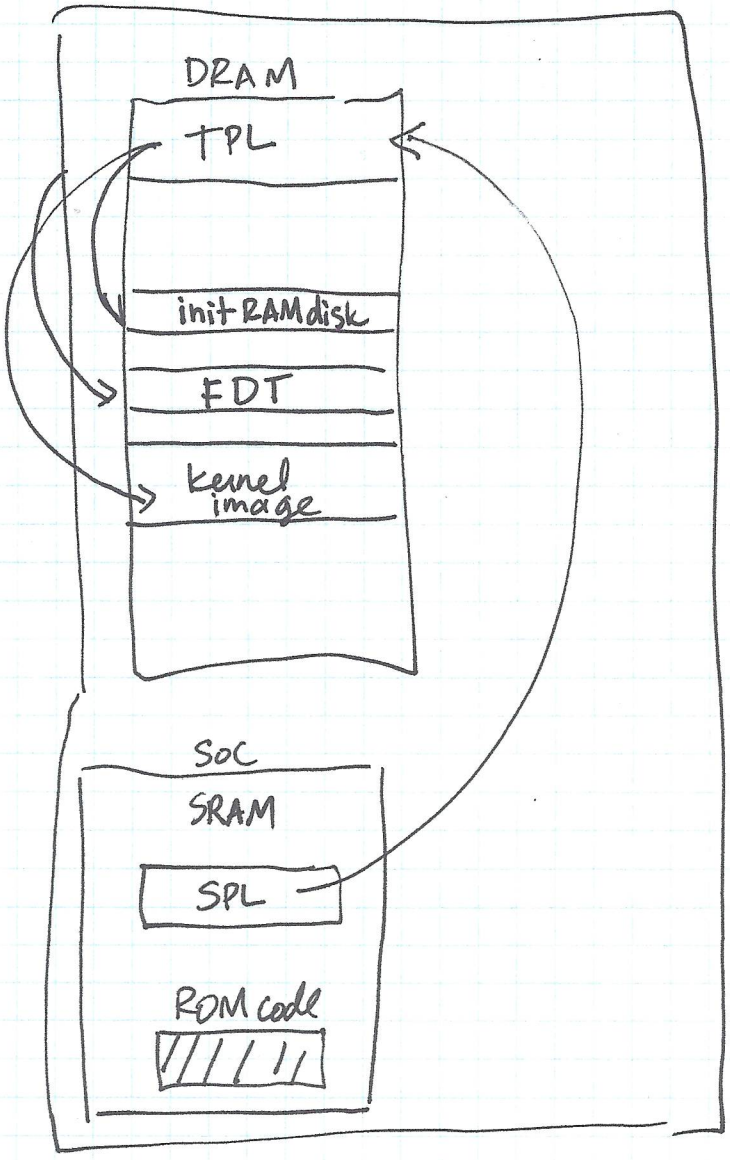
  - the SPL sets up the memory controller and other essential parts of the system in preparation for the larger third stage program loader (TPL)



  - SPL may be open-source (free software) or it may have proprietary components

- PHASE 3 : TPL

  - finally, we are running a full-fledged bootloader, like U-Boot or Barebox.
    → it even provides a command-line interface!!
  - this bootloader loads the kernel into main memory and performs the final preparations for the system
  - once it completes its task, bootloaders usually disappear from memory



Beaglebone Specifics

AM335x Boot ROM
- internal/first stage bootloader
- fixed at manufacture
- minimal peripheral configuration
- loads X-loader

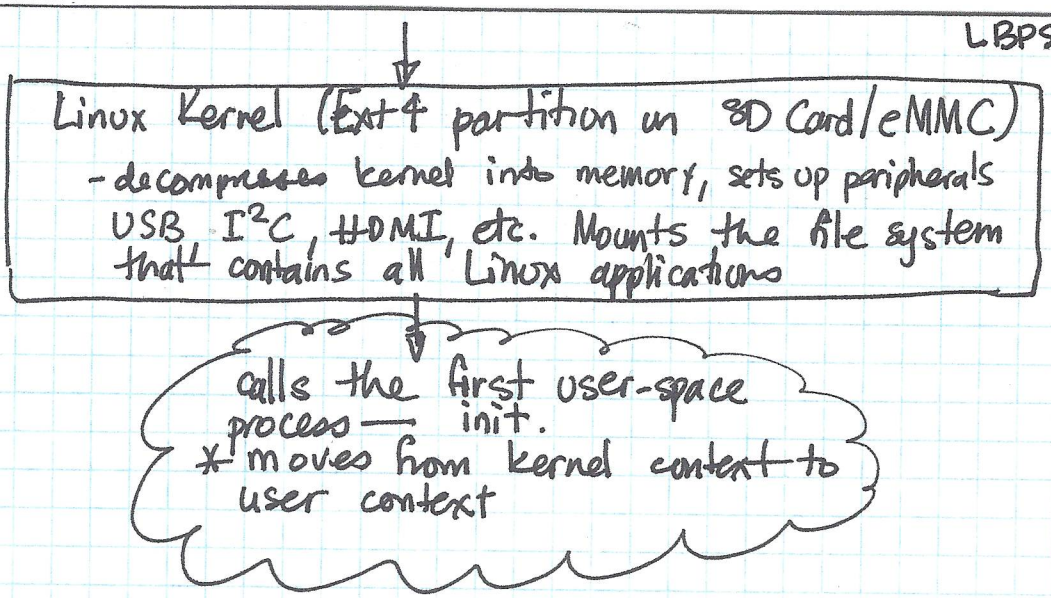
↓

X-Loader (MLO on the FAT partition)
- second-stage bootloader
- sets up pin muxing, clocks/memory, loads U-Boot

↓

U-Boot (u-boot.img on the FAT partition)
- third-stage boot loader
- specifies the root file system. Uses uEnv.txt configuration. Performs additional intialization, passes control to Linux Kernel

↓


Linux Kernel (Ext4 partition on SD Card/eMMC)
- decompresses kernel into memory, sets up peripherals USB, I²C, HDMI, etc. Mounts the file system that contains all Linux applications
calls the first user-space process — init.
* moves from kernel context to user context

systemd System and Service Manager

→ see Molloy (2nd Ed.) pp. 85-90

- "recent and somewhat controversial" addition to Linux, aiming to replace yet remain backward compatible with the older System V (SysV) init

  → SysV init was the parent process managing all systems and services for GNU-Linux

  - starting, stopping services like web servers and terminal command-shell servers depending on the current state or run level.

SysV init run level	description
0	halt the system
1	single-user mode (for admin)
2-5	multi-user modes (regular operation)
6	reboot the system
S	start-up services

→ a major drawback of SysV init is that it starts tasks in series, waiting for one task to complete before starting the next, which can lead to lengthy boot times.

→ systemd, by contrast, launches services in parallel, allowing shorter boot times.

→ use the systemctl command to interact with systemd

```
$ systemctl # lists all running services
```

NOTE: certain uses of systemctl may require the sudo prefix

```
$ systemctl start <service_name> # starts service
$ systemctl stop <service_name> # stops service
$ ~~sle~~ systemctl status <service_name> # check service status
```

→ see Table 3-1, pg. 87, for more uses of systemctl, from Molloy's 2nd Ed.

The Linux Device Tree

→ see Molloy 2nd Ed. (pp. 271-279)

→ a means by which embedded Linux systems specify their hardware configuration

→ NOTE: your desktop machine uses BIOS for this instead.

→ ARM devices running Linux make use of Flattened Device Tree (FDT) models to describe hardware/devices.

→ the device tree was conceived to be a human readable description that could by converted into efficient machine-readable format

→ the FDT is a human-readable data structure (somewhat resembling JSON format for non-relational databases)

→ can be used to describe properties of CPU, memory, GPIOs, PWMs, UARTs, ADCs & DACs, etc!

→ also see elinux.org/Device_Tree_Usage

→ these human readable FDT files are stored as "DTS" (.dts extensions) are converted into DTB (binary, machine-readable form) using the device tree compiler (DTC)

→ these compiled files are placed in your embedded Linux filesystem at /boot/dtbs/ → check it out on your Beaglebone or Pocket Beagle!

→ you can control which of those FDT files is incorporated into your system by modifying the configuration file, /boot/uEnv.txt