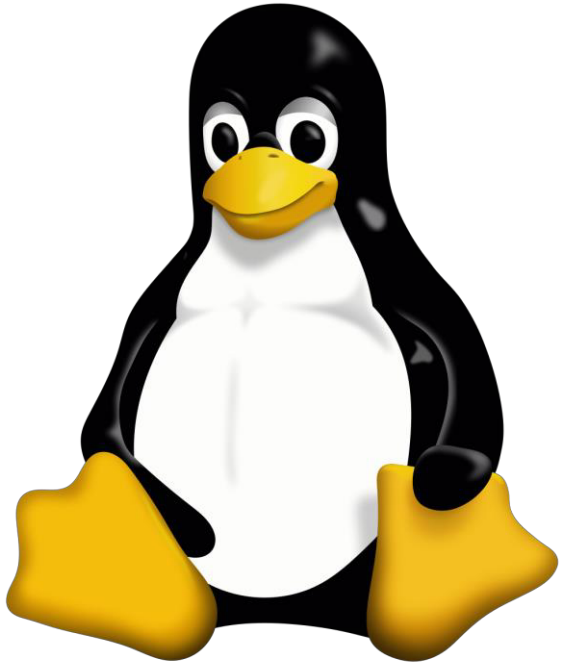
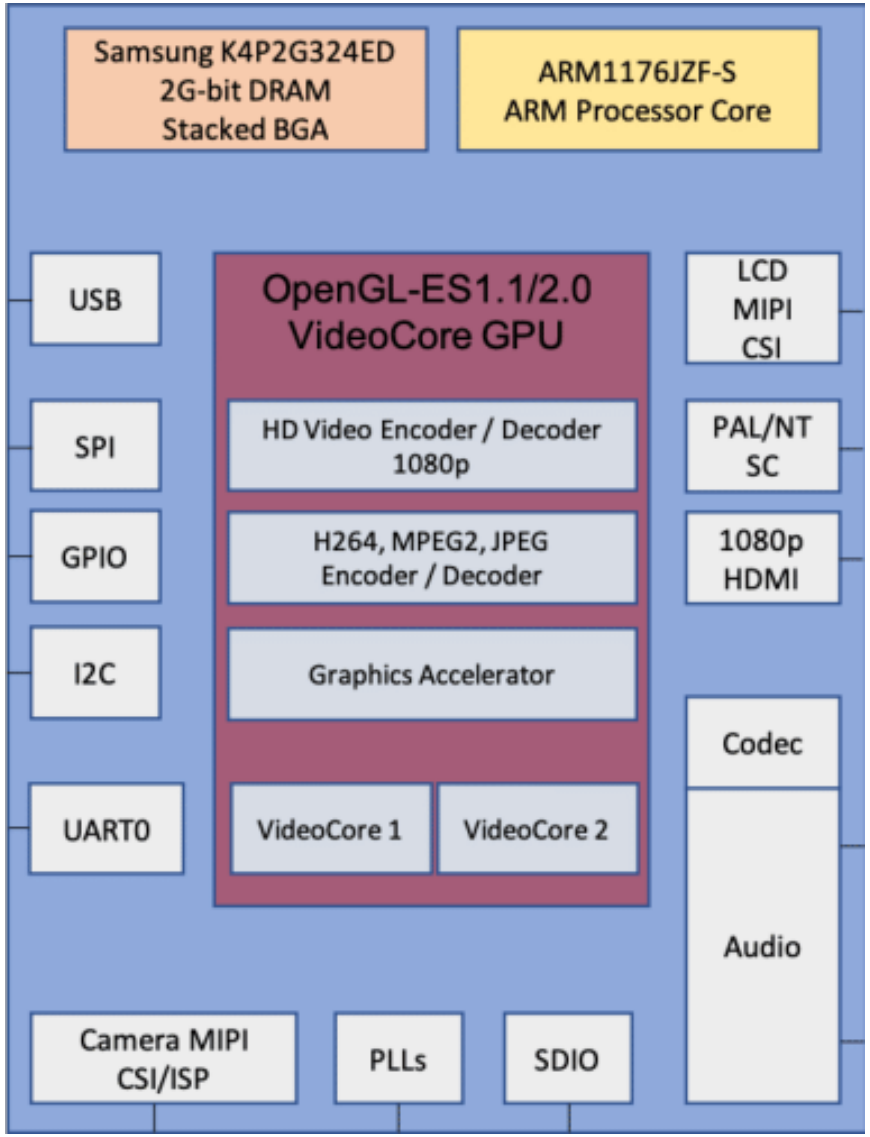


Linux Character Device Driver

Sunbeam Infotech

BCM2835 & AM335x

RPi



BBB →

ARM®
Cortex®-A8
Up to 1 Ghz

32KB and 32KB L1 + SED
256KB L2 + ECC
176KB ROM | 64KB RAM

Graphics
PowerVR
SGX
3D GFX

Crypto
64KB
shared
RAM

Display
24-bit LCD controller
Touch screen controller

PRU-ICSS
EtherCAT, PROFINET,
EtherNet/IP,
and more

L3 and L4 interconnect

Serial
UART x6
SPI x2
I²C x3
McASP x2
(4 channel)
CAN x2
(Ver. 2 A and B)
USB 2.0 HS
DRD + PHY x2

System
eDMA
Timers x8
WDT
RTC
eHRPWM x3
eQEP x3
PRCM

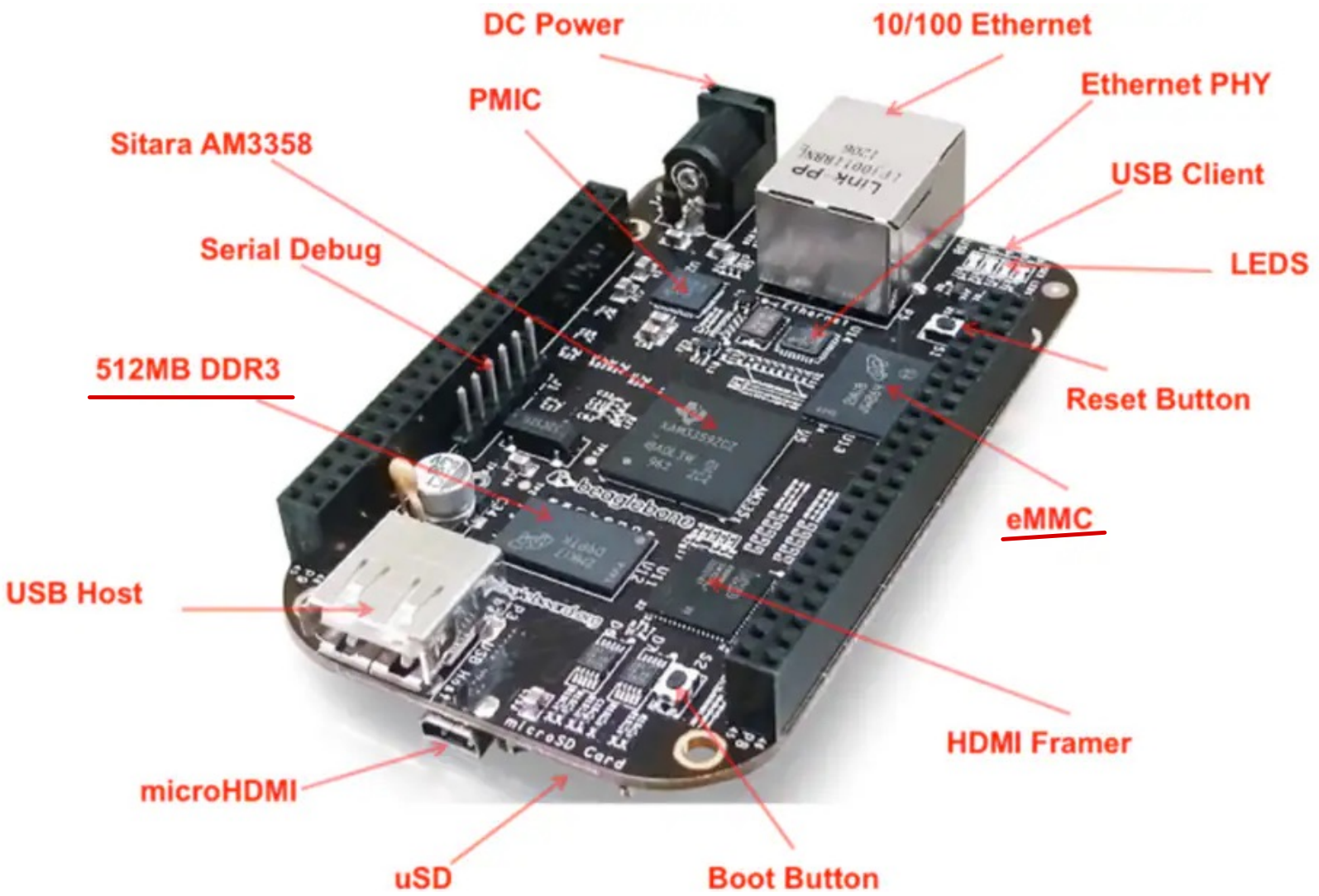
eCAP x3
ADC (8 channel)
12-bit SAR
JTAG
Crystal
Oscillator x2

Parallel
MMC, SD and
SDIO x3
GPIO

EMAC (2-port) 10M, 100M, 1G
IEEE 1588v2, and switch
(MII, RMII, RGMII)

Memory interface
mDDR(LPDDR), DDR2,
DDR3, DDR3L
(16-bit; 200, 266, 400, 400 MHz)
NAND and NOR (16-bit ECC)

BeagleBone Black



Cortex-A8

ROM - 176 KB

RAM - 64 KB
+ 64 KB

External RAM - DDR3
- 512 MB

eMMC - 4GB card.

External MMC card
- 8GB / 16 GB...

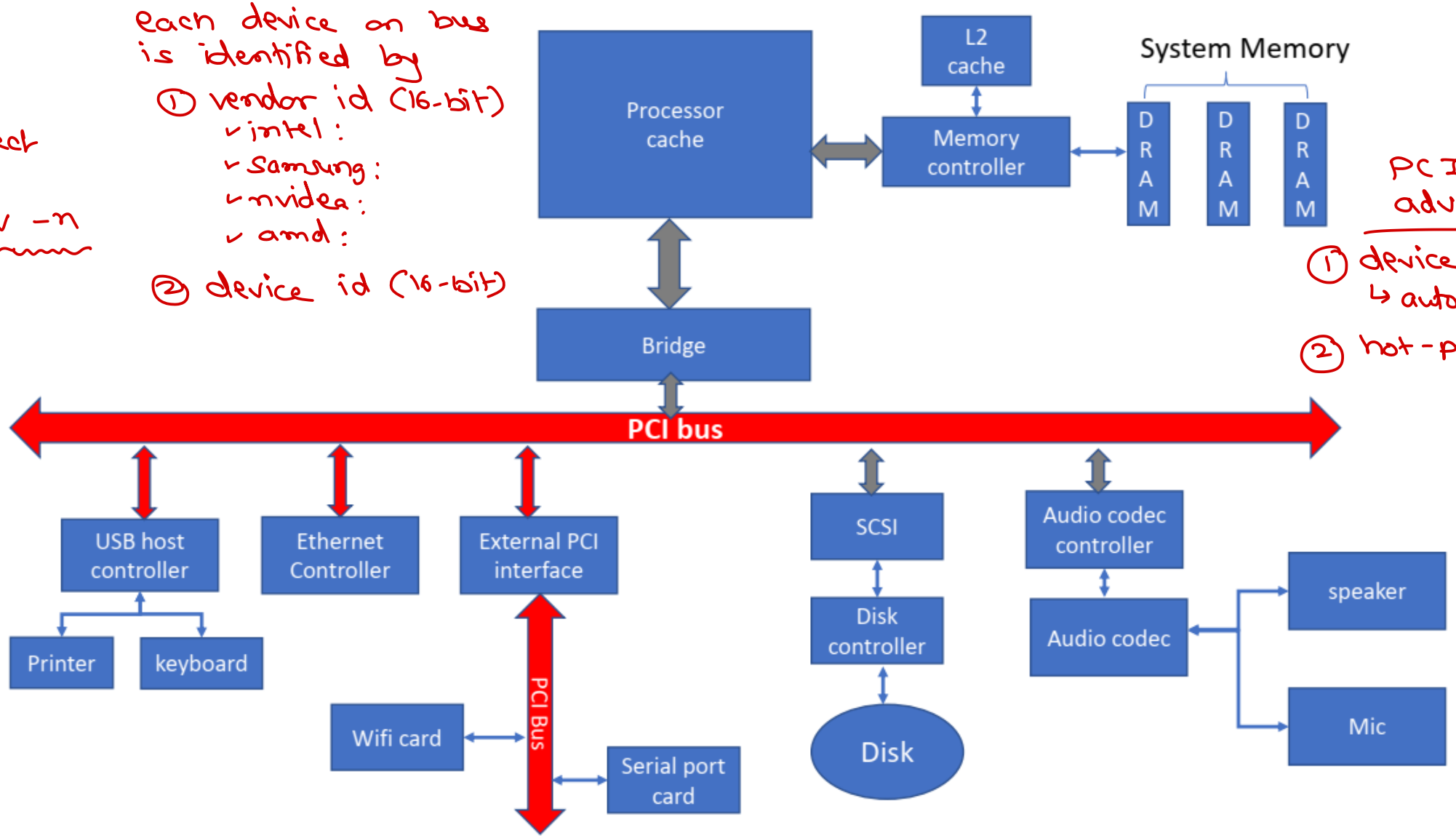
PCI bus – PC architecture

Peripheral Component Inter Connect

lspci -v -n

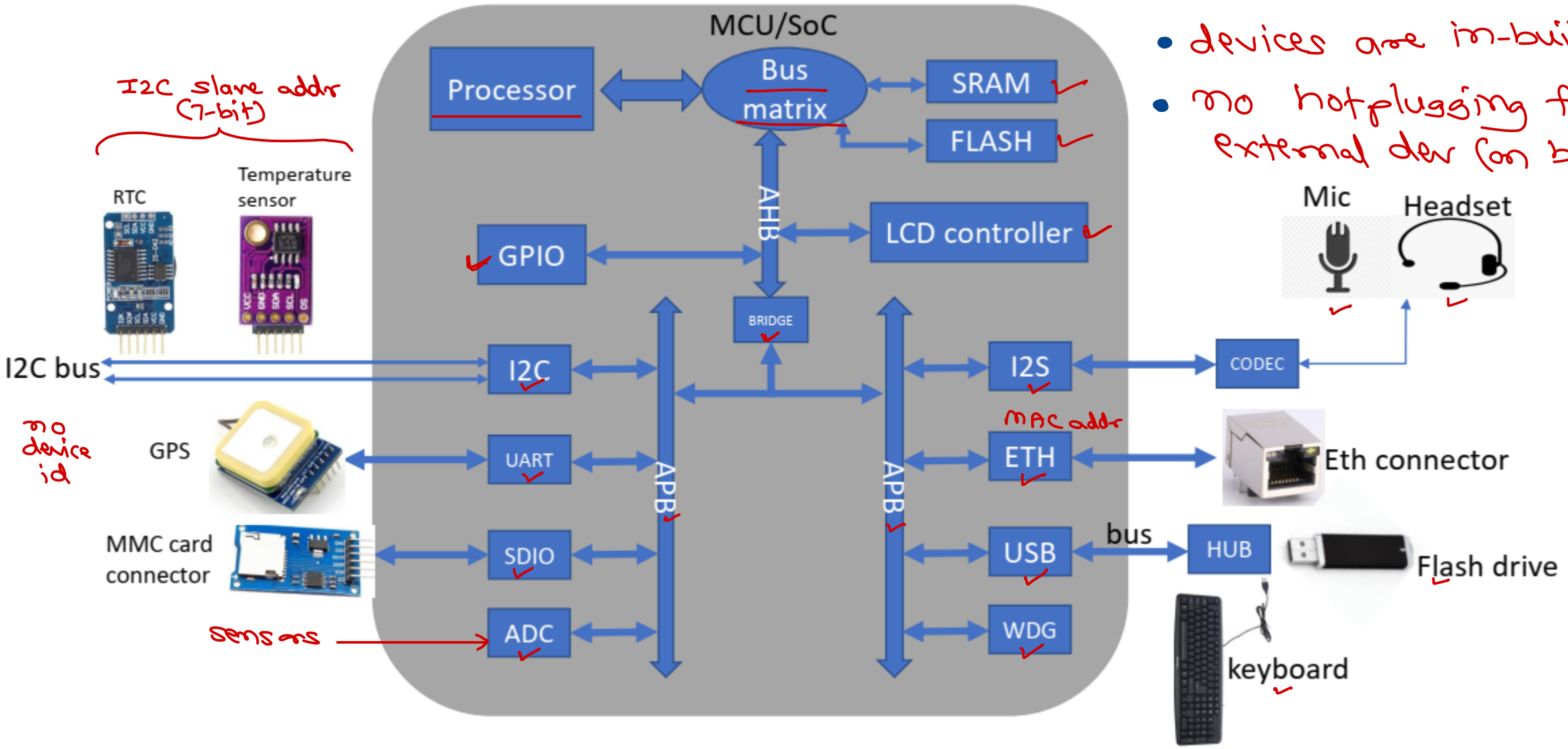
each device on bus is identified by
① vendor id (16-bit)
- intel :
- samsung :
- nvidea :
- amd :

② device id (16-bit)



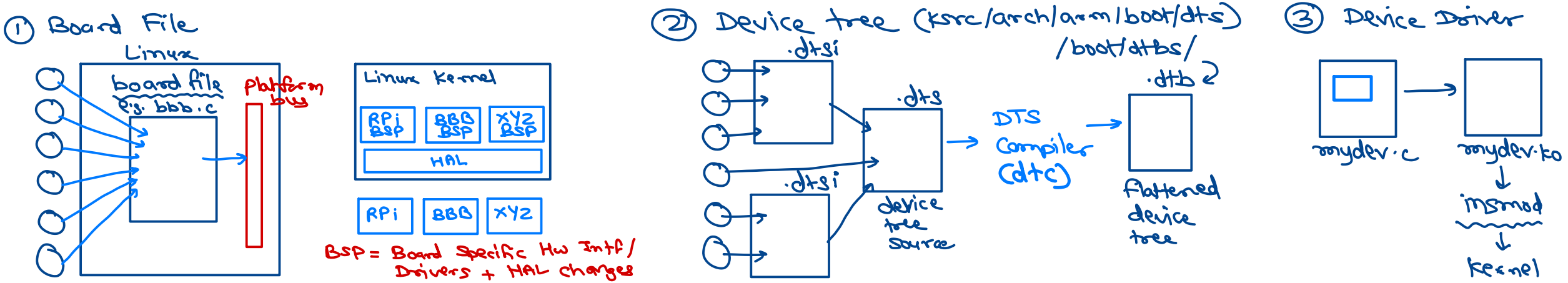
PCI bus advantages
① device identification
↳ auto-discoverable
② hot-plugging

Embedded – SoC

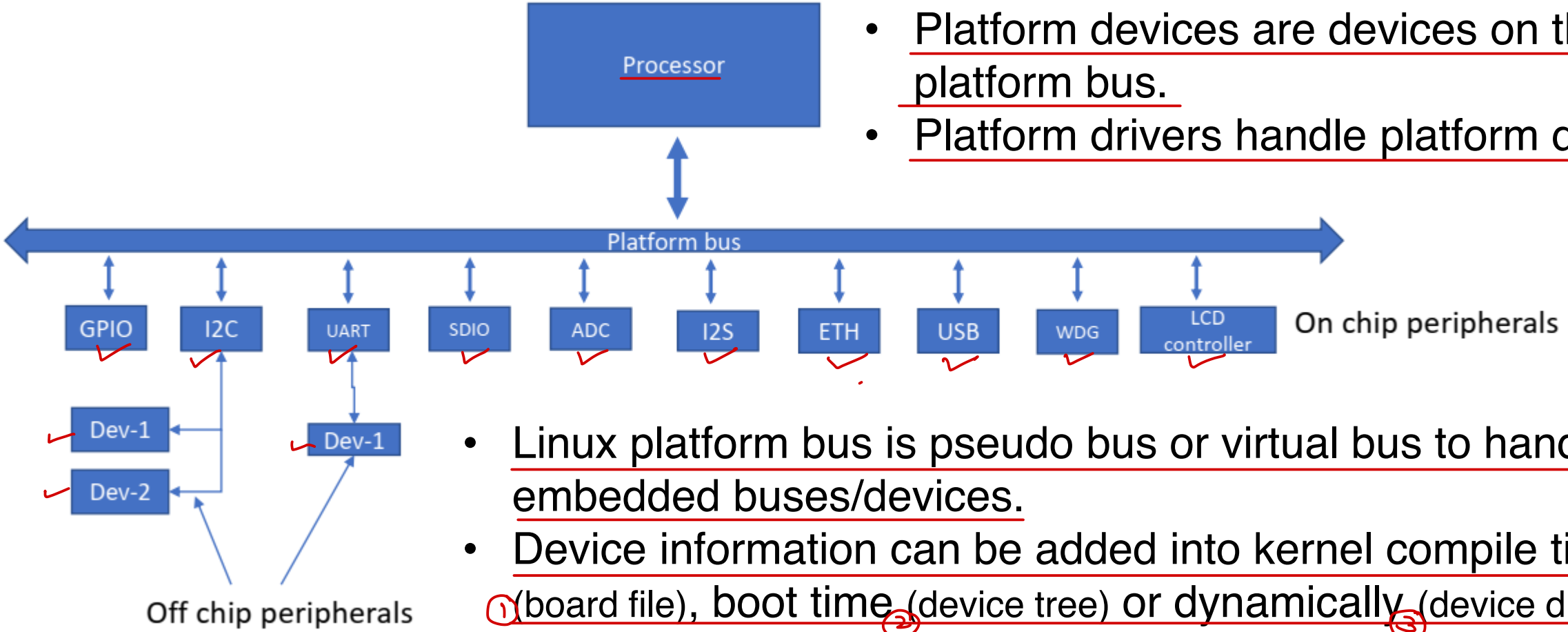


Platform bus, device and driver

- In Linux on PC architecture, most of the IO devices are connected over PCI and USB buses.
- PCI and USB buses are auto-discoverable (lspci, lsusb) and hot-pluggable (plug n play).
- Typical embedded Linux on ARM or other architecture do not have PCI bus.
- In embedded hardware (SoC) most of devices/buses are available on chip itself and are directly connected to CPU.
- Embedded buses like SPI, I2C, CAN, I2S are not discoverable/hot-pluggable.



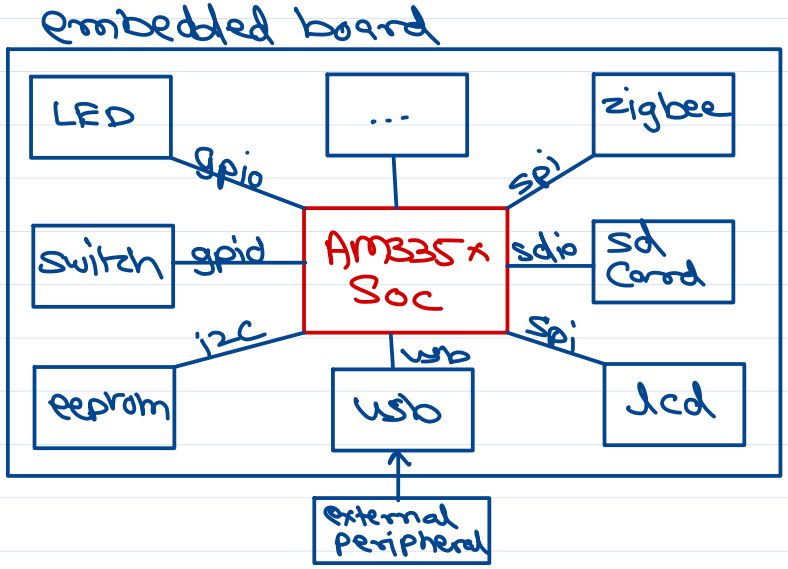
Platform bus



- Platform devices are devices on the platform bus.
- Platform drivers handle platform devices.

- Linux platform bus is pseudo bus or virtual bus to handle embedded buses/devices.
- Device information can be added into kernel compile time (board file), boot time (device tree) or dynamically (device driver).
  - Memory/IO address, IRQ number, Device Id, Device address, Pin configuration, Power/voltage information, etc.
- Device Tree: https://www.kernel.org/doc/Documentation/devicetree/usage-model.txt

Device tree



Embedded Buses/Connectivity

1. USB
2. RS232
3. SPI
4. I2C
5. CAN

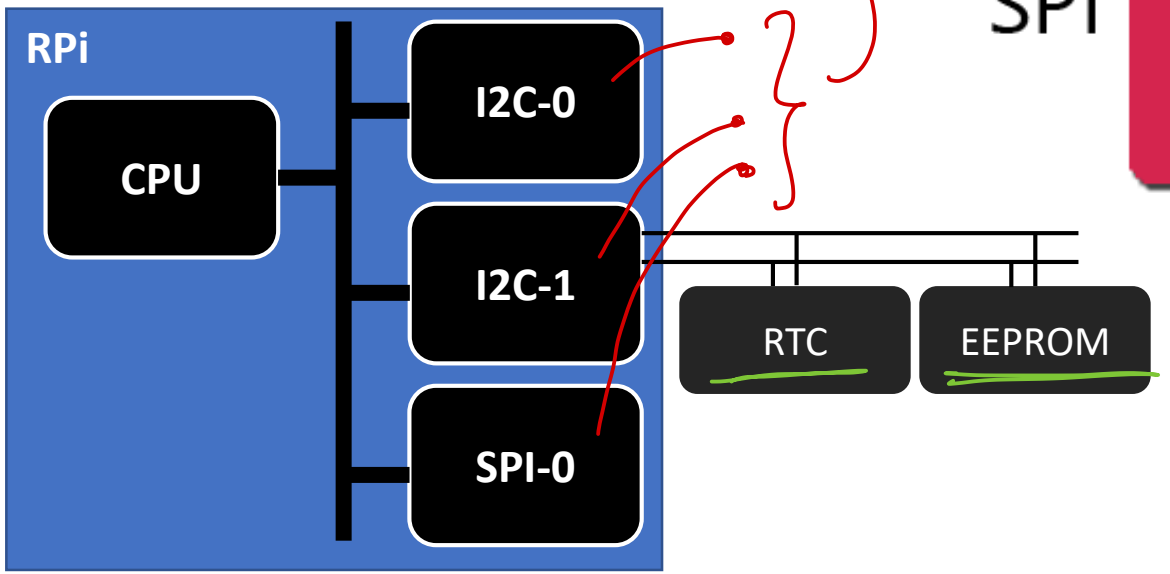
→ platform devices

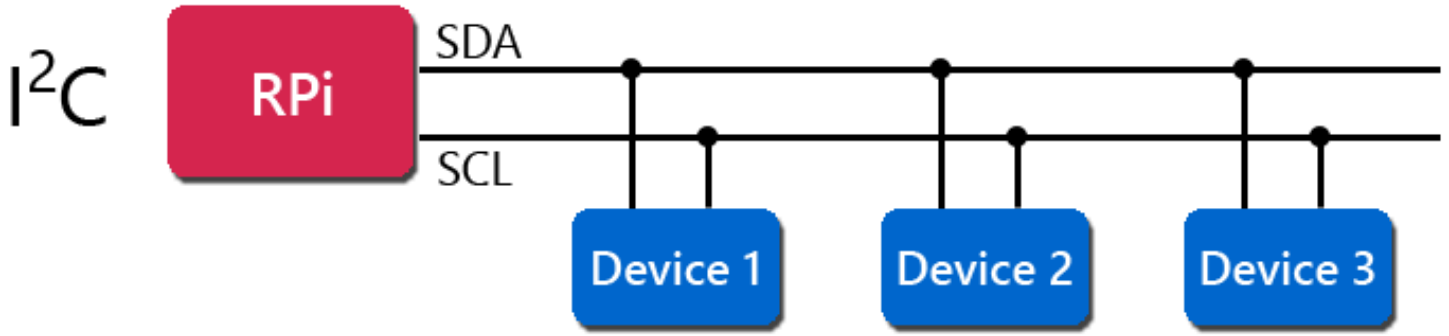
- board file
  - ↳ board specific + probe drivers (recompile kernel for each board).
  - my_board_init()
    - ✓ add_device_serial()
    - ✓ add_device_spi()
    - ✓ add_device_eth()
    - ✓ add_device_i2c()
    - ✓ add_device_gpio()
  - + device drivers (.ko)
- device tree source
  - ↳ board specific describe data struct.
  - .dts → DTS Compiler → .dtb

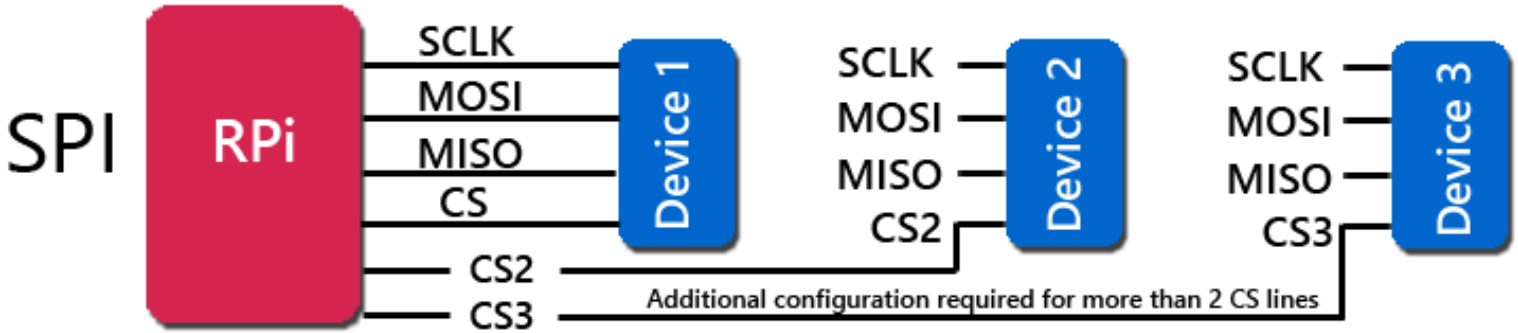
Same kernel can be used to init diff boards with diff dtb files.

Platform devices and drivers

- Embedded devices (on-board and off-board) are platform devices.
- Respective vendors usually provide platform drivers host/bus controllers called as "Controller drivers".







- Devices on I2C bus are I2C client devices.
- RPi provides i2c host controller driver and client drivers implemented in user or kernel space.

Platform Driver

- https://www.kernel.org/doc/Documentation/driver-model/platform.txt

```
struct platform_driver {
        int (*probe)(struct platform_device *);
        int (*remove)(struct platform_device *);
        void (*shutdown)(struct platform_device *);
        int (*suspend)(struct platform_device *, pm_message_t state);
        int (*resume)(struct platform_device *);
        struct device_driver driver;
        const struct platform_device_id *id_table;
};
```

called when device attached or driver loaded (matching).

called when device detached or driver unloaded (matching).

to shutdown/power off the device.

when device is suspended (idle) and resume. During suspension few dev can be kept in low power state.

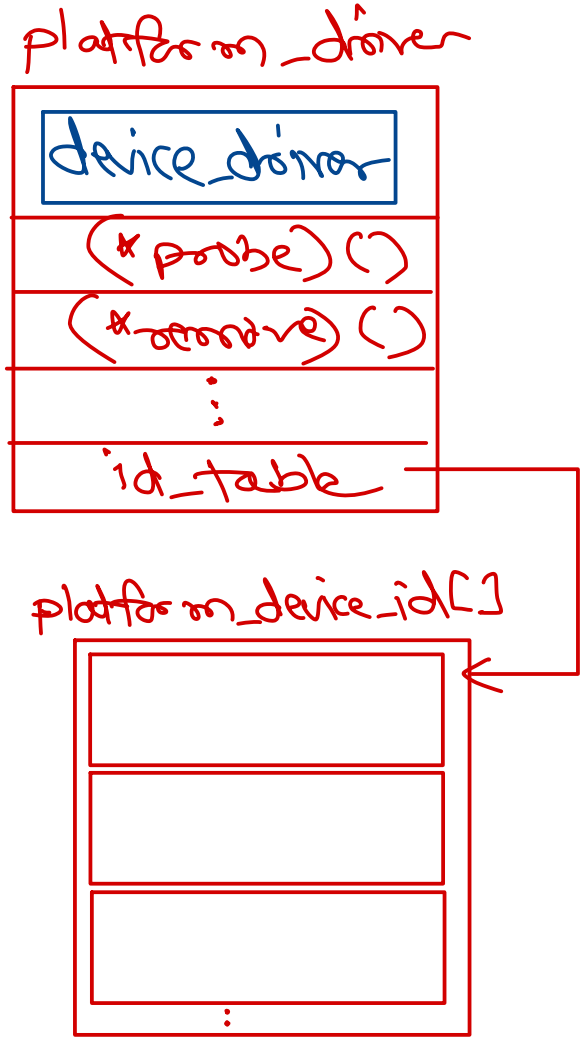
- To register platform driver
  - platform_driver_register(drv);

Platform Driver

- https://www.kernel.org/doc/Documentation/driver-model/platform.txt

```
struct platform_driver {
        int (*probe)(struct platform_device *);
        int (*remove)(struct platform_device *);
        void (*shutdown)(struct platform_device *);
        int (*suspend)(struct platform_device *, pm_message_t state);
        int (*resume)(struct platform_device *);
        struct device_driver driver;
        const struct platform_device_id *id_table;
};
```

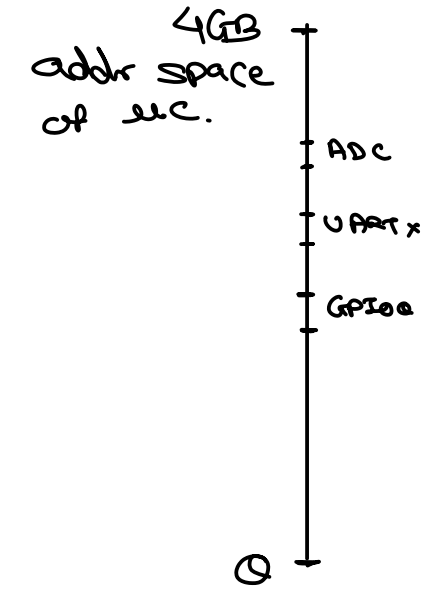
- To register platform driver
  - platform_driver_register(drv);

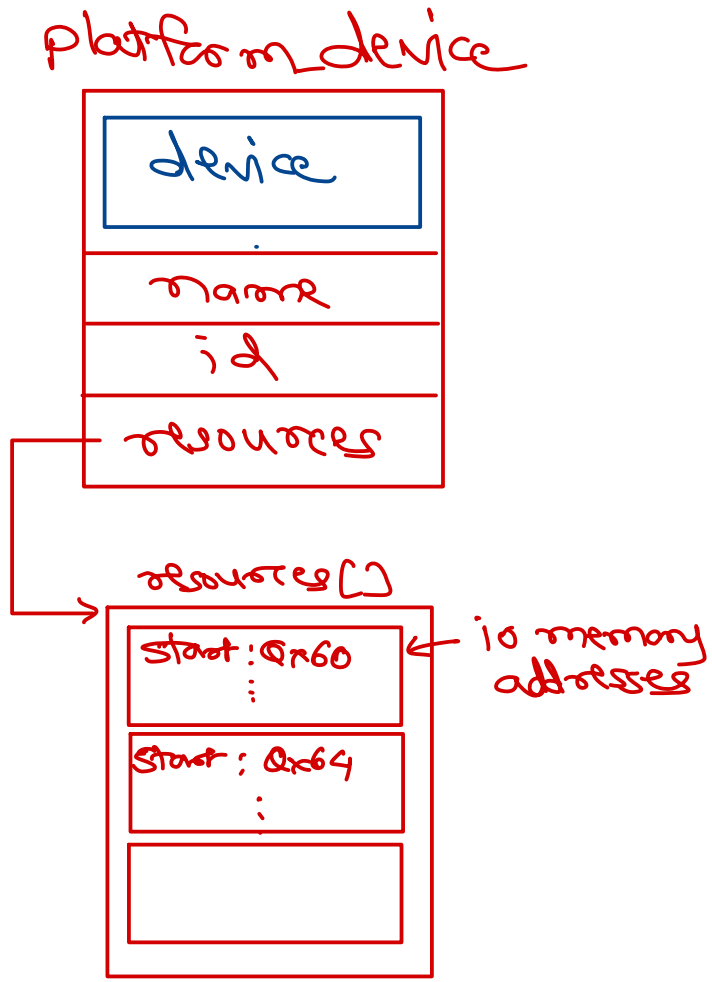


Platform Device

- Platform device is represented by struct platform_device.

```
struct platform_device {
        const char      *name;
        u32             id;
        struct device   dev;
        u32             num_resources;
        struct resource *resource;
};
```

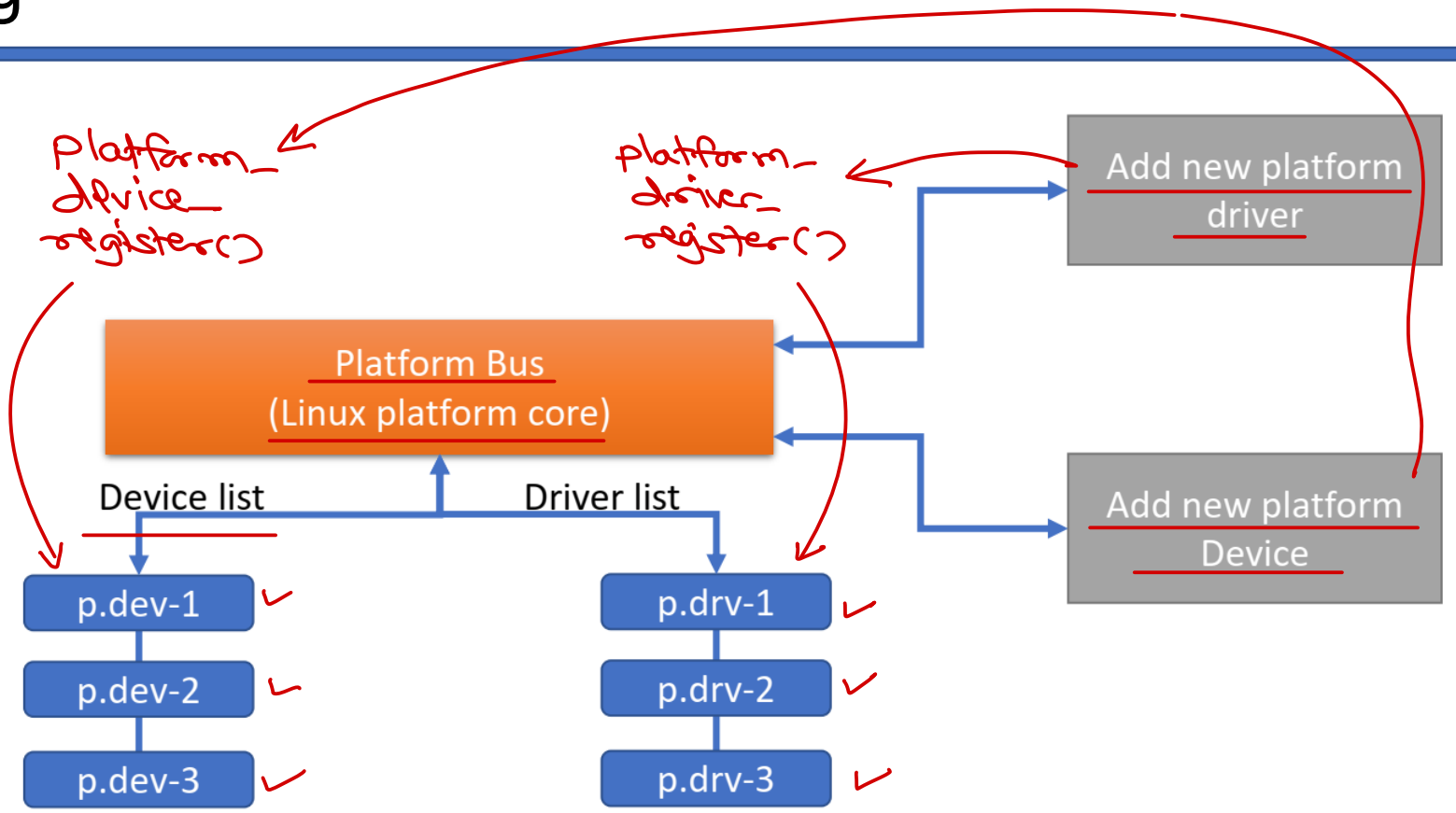




- The device is registered using: platform_device_register(dev);
  - From board file (compile time) or device driver (dynamically) – deprecated.
  - Devices are now registered using device tree.

Device and Driver matching

- Platform device and Platform driver are matched by the bus core matching mechanism.
  - Driver can detect the matching device added into the system.
  - Correct driver is auto-loaded when new device is added into the system.
- Each bus type has its match function that scans device and driver list.
- Linux platform core maintains platform device and driver list. It is auto updated when device or driver is added. e.g. /sys/bus/i2c – devices/drivers



- Match is done by name or ids.
- Upon match, probe() of driver is called by the core.
- When device/driver is removed, remove() is called.

platform_driver operations

- Platform driver must implement and register these method while platform_driver_register().
- When matching is done by the core, probe() will be called with platform_device as argument.
- probe() is responsible for
  - Device detection (verify) and initialization
  - Mapping IO memory and Register ISRs
  - Create user space access points (/dev or /sys)
  - Register device to the kernel framework
- When device or driver is removed, remove() will be called by the core.
- remove() is responsible for
  - Free memory and ISRs.
  - Shut-down or de-initialize the device.
  - Unregister device from the kernel framework
- suspend() is called to put device is pause/sleep (low power) state.
- resume() is called to set device in normal state (from sleep state).
- shutdown() is called to stop the device during system shutdown.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>