Android Device Driver Collage2

: PM, Display, Input, Sensors, Network, Multimedia, Interconnections

chunghan.yi@gmail.com, slowboot

본 문서에서는 …

- *Smart Phone Device Driver를 구성하는 여러 내용 중, 아래 4), 5), 6)번 부분(파란색 부분)을 중점적으로 소개하고자 한다.*

- *1) 기본 Interface Drivers*
- *➔ UART, SPI, I2C, USB, SDIO, DMA …*
- *2) Storage Drivers*
- *➔ NAND, SD, eMMC …*
- *3) Power Management 관련 Drivers*
- *➔ PM & wakelock, battery(fuel gauges) & charger..*
- *4) LCD & Touchscreen Drivers, Some Sensors*
- *➔ LCD, Touch, keypad, Sensors, Vibrator, TVOUT/HDMI …*
- *5) 네트워크 Drivers*
- *➔WiFi, Bluetooth, NFC, RmNet(3G/4G data), GPS …*
- *6) Multimedia & Graphic 관련 Drivers*
- *➔ Video/Audio encoder/decoder, 2D/3G graphic accelerator, Sound Codec, Camera, TDMB …*
- *7) RIL & Modem Interface*

본 문서에서 소개하는 내용 중에는 Qualcomm(snapdragon), Samsung (exynos) 및 TI(OMAP) chip에 해당하는 것을 예로 든 부분이 있으나, 전체적인 개념을 소개하는 것이 목적인 만큼, 내용 중 일부는 사실과 다를 수 있음을 밝힌다.

목차

0. AP Review & BSP Work Area

: OMAP, SnapDragon, Exynos, Tegra

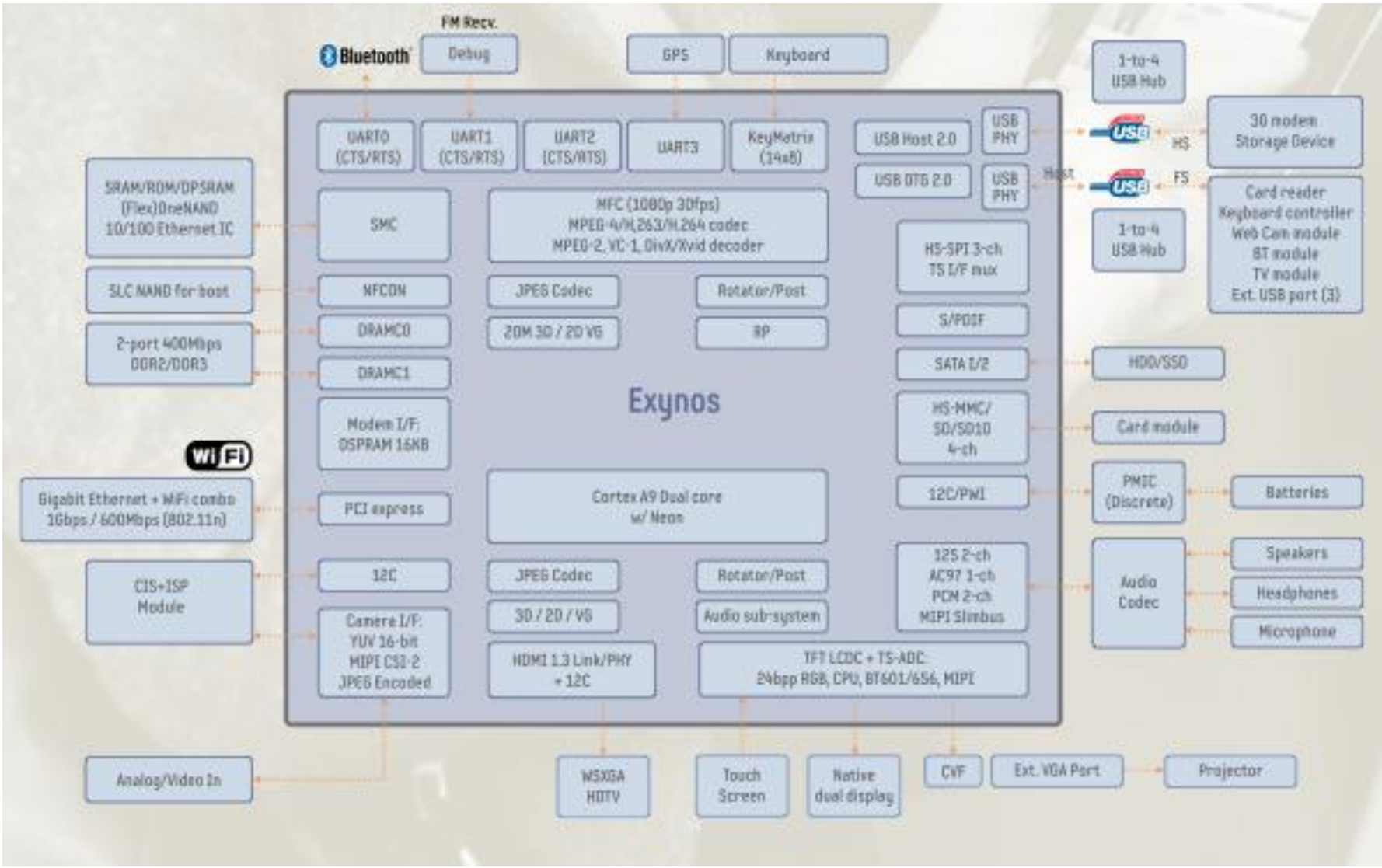
1. Nvidia Tegra

- *<TODO>*

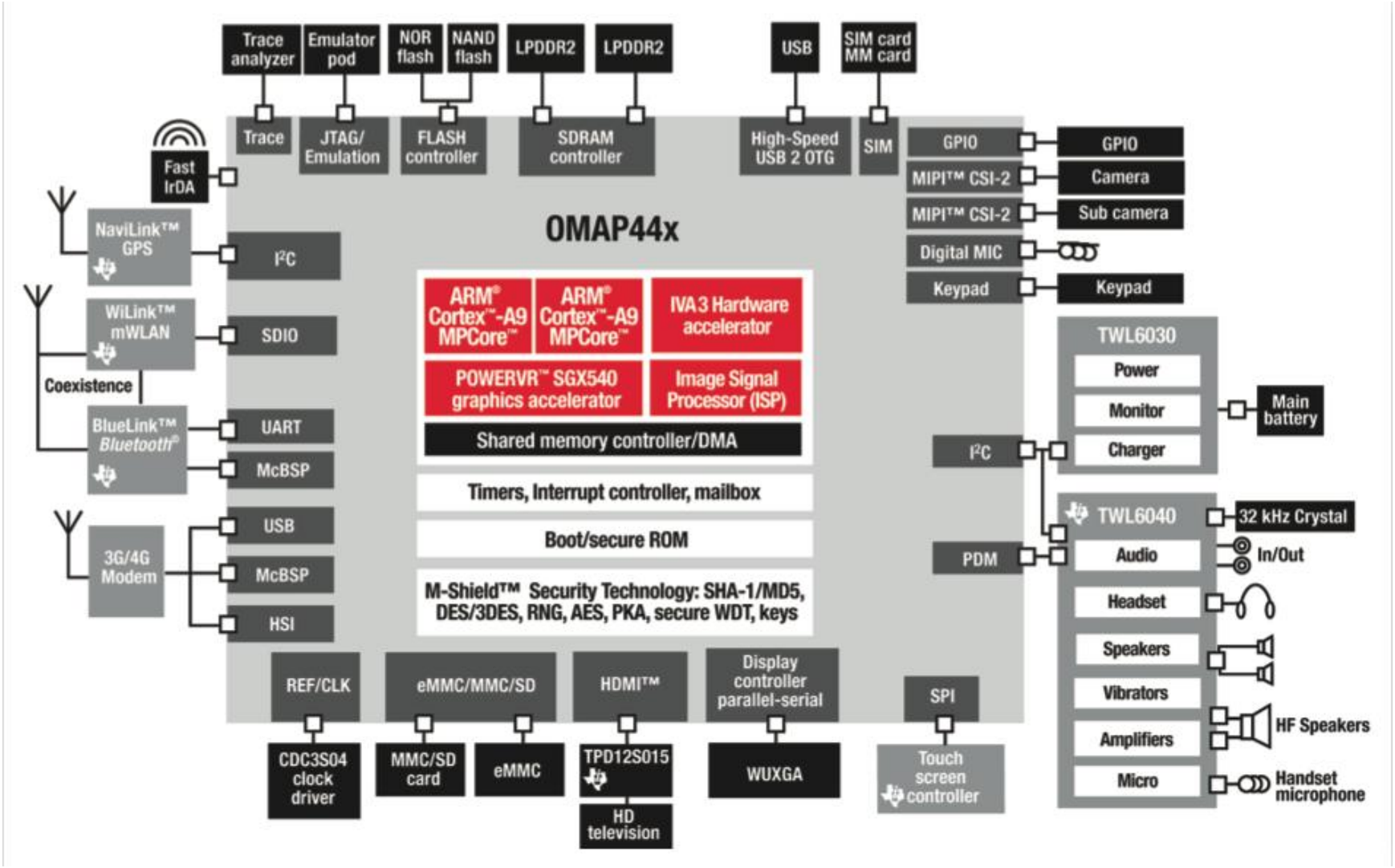
2. Qualcomm Snapdragon

- *<TODO>*

3. Samsung Exynos 4210



4. TI OMAP 44xx



5. BSP Work Area(1)

- *1)* ***Boot***
  – NAND/eMMC/Sdcard/RAM
- *2)* ***Display***
  – LCD/TVout/HDMI/Framebuffer/Graphics Accelerator …
- *3)* ***Sensors***
  – Touchscreen/Accelerometer/Geomagnetic/Proximity … /Vibrator
- *4)* ***Multimedia***
  – Video-Audio Encoder-Decoder/Audio Codec/Camera/DMB
- *5)* ***Network***
  – Wi-Fi/BT/GPS/FM Radio/NFC
- *6)* ***Modem***
  – Protocol area
- *7)* ***Power Management***
  – PMIC/Battery/Charger …
- *8)* ***Interconnection***
  – SPI/I2C/UART/USB/SD-MMC/… /GPIO/PWM/DMA/ … /IPC(processor communication)

5. BSP Work Area(2)

- *1)* ***Boot***

  – bootloader, kernel, boot sequence(init.rc, init.hardware.rc) …
- *2)* ***Display***

  – RGB LCD driver, backlight driver, frame buffer driver, 2D/3D graphic accel driver …
- *3)* ***Sensors***

  – ADC, SPI or I2C interface, Input device driver …
- *4)* ***Multimedia***

  – AV codec driver, Audio Codec(I2S, Speaker, MIC …), Camera sensor(i2c)/V4L2 driver, DMB(i2c)/stagefright extention ..
- *5)* ***Network***

  – wifi driver, bluetooth stack, gps driver …
- *6)* ***Modem***

  – Protocol area
- *7)* ***Power Management***

  – battery driver, charger driver, PMIC driver(chip dependent), clock, regulator, suspend/resume, wakelock …
- *8)* ***Interconnection***

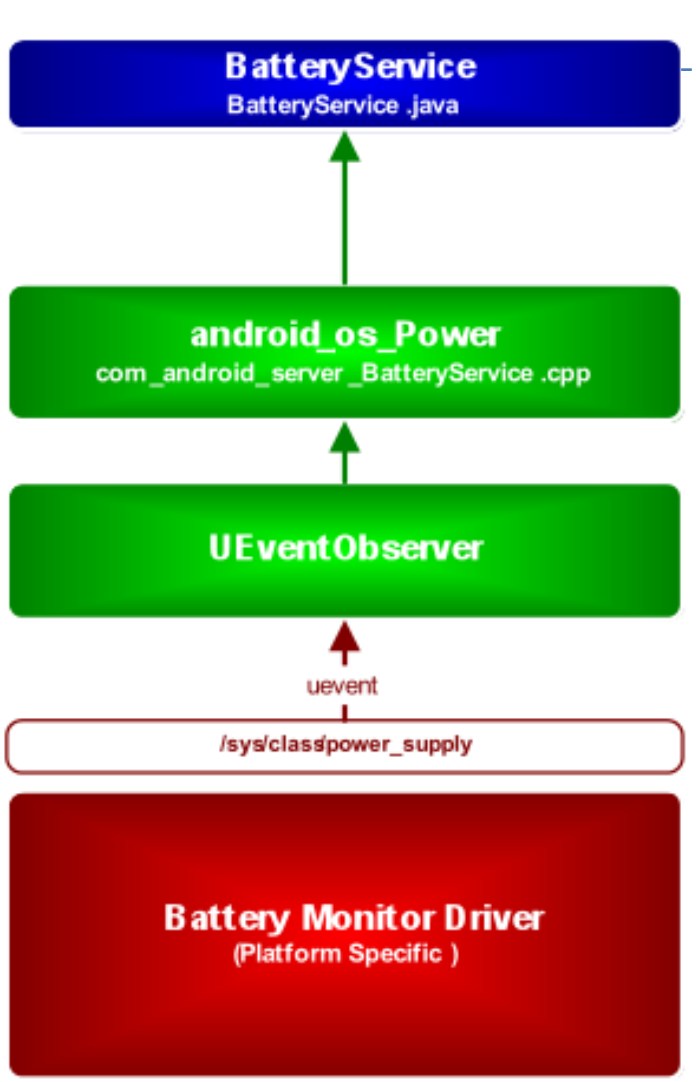
  – 기본 interface 이에도 chip vendor 에서 독자적으로 추가한 interface 다수 존재

  - CPU간 통신 방법 존재

1. Power Management

: Charger & Battery Driver, clock, regulator, gpio …

1. Android Battery Service Architecture



sendIntent(battery 정보 전달)

1) Linux kernel의 power supply class를 사용함
➔ /sys/class/power_supply

2) Battery 상태를 갱신하기 위하여 uevent 사용함.
➔ SUBSYSTEM=power_supply

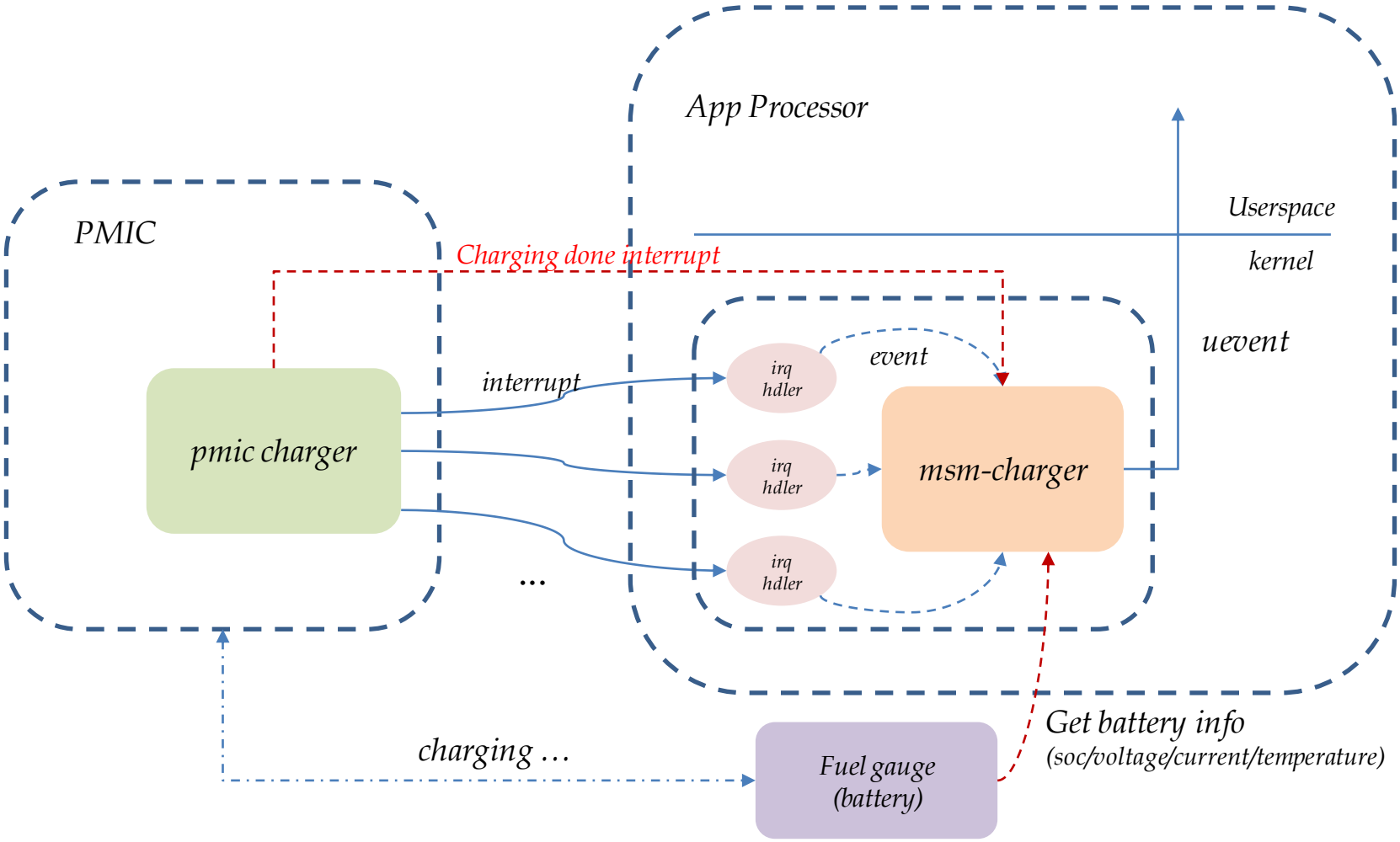
3) BatteryService가 battery 상태를 주기적으로 모니터링함. 새로운 상태를 다른 서비스로 전달(intent 이용)함.

<linux kernel battery driver>
(*) drivers/power/power_supply.c*
➔ power supply class를 위한 공통 파일

(*) drivers/power/ds2784_battery.c
➔ 이건 battery driver의 예임.
➔ 위의 power supply code와 연계됨

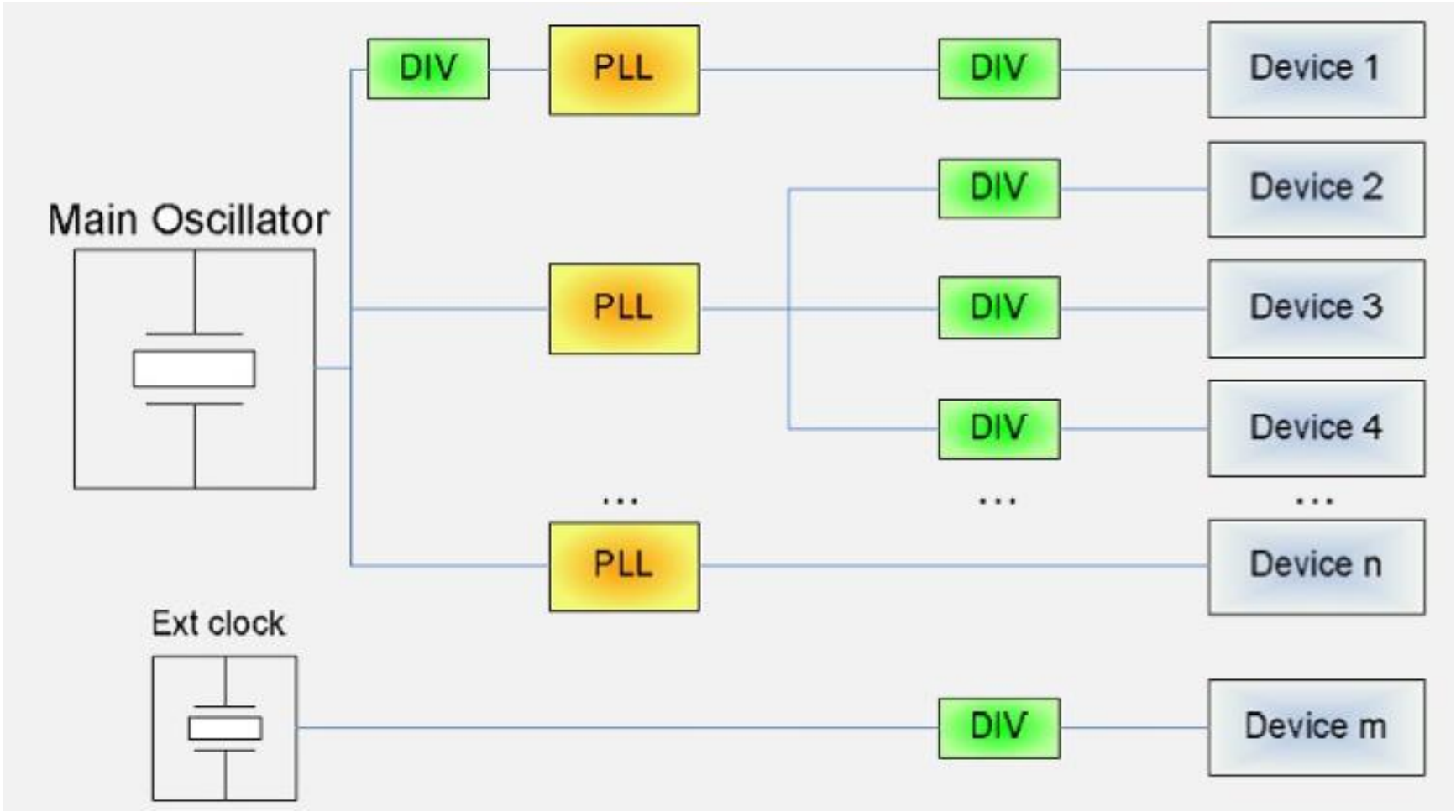
(*) 위의 이미지는 인터넷에서 copy한 것임[17].

2. Battery(Fuel Gauge) & Charger Driver



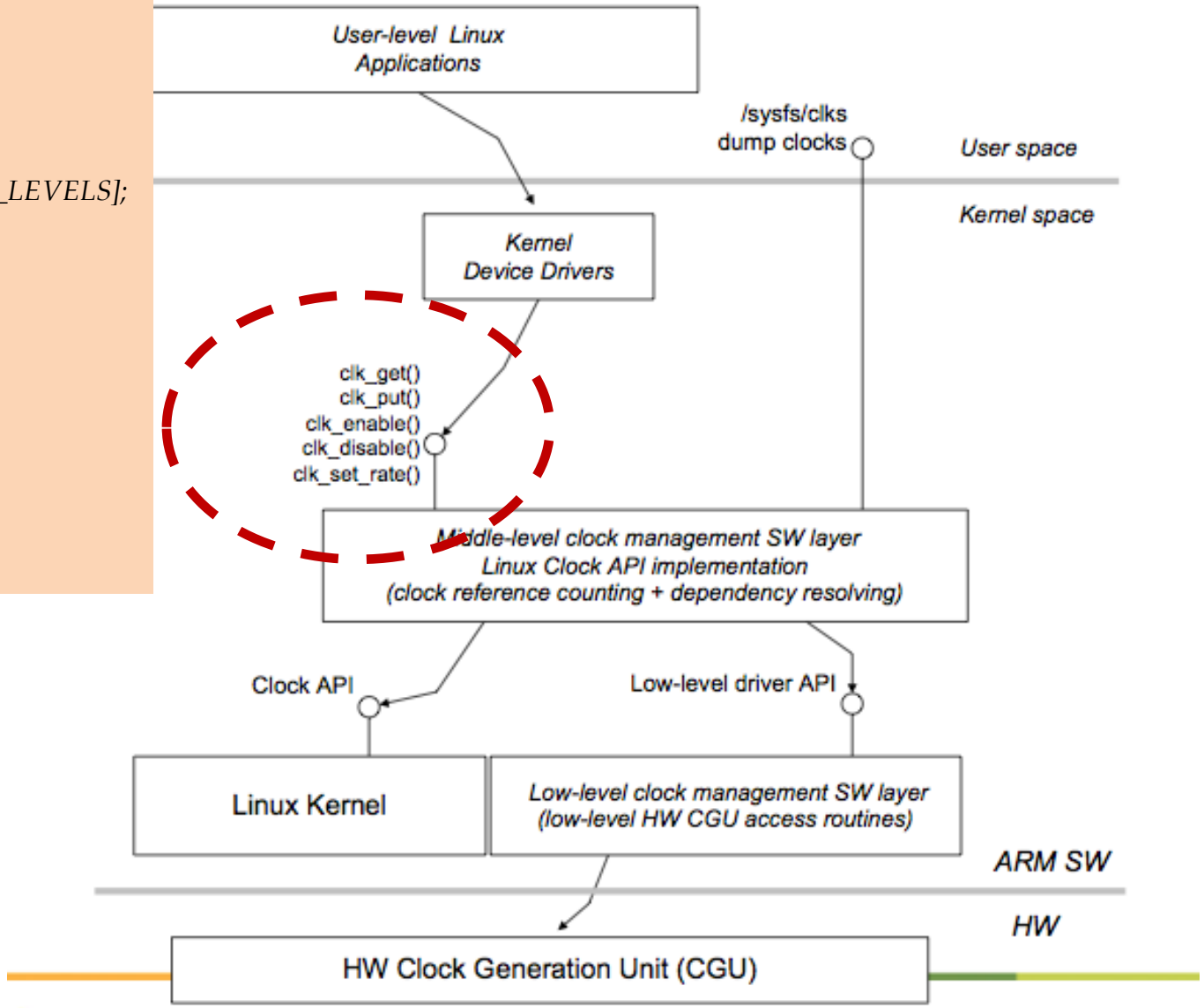
3. Clock Framework(1)

(*) 아래 그림에서는 clock이 4단계, 즉 Oscillator -> PLL -> Clock blocks(DIV) -> Device로 구성되어 있음을 보여준다.
(*) 이러한 clock은 보드 초기화 코드(가령, board-msm8x60.c -> clock.c/clock-8x60.c)에서 초기화되며, 각각의 드라이버에서 clock operation(다음 page)을 이용하여, 설정 변경될 수 있다.



3. Clock Framework(2)

```
struct clk {
    uint32_t flags;
    struct clk_ops *ops;
    const char *dbg_name;
    struct clk *depends;
    struct clk_vdd_class *vdd_class;
    unsigned long fmax[MAX_VDD_LEVELS];
    unsigned long rate;

    struct list_head children;
    struct list_head siblings;
#ifdef CONFIG_CLOCK_MAP
    unsigned id;
#endif

    unsigned count;
    spinlock_t lock;
};
```



3. Clock Framework(3)

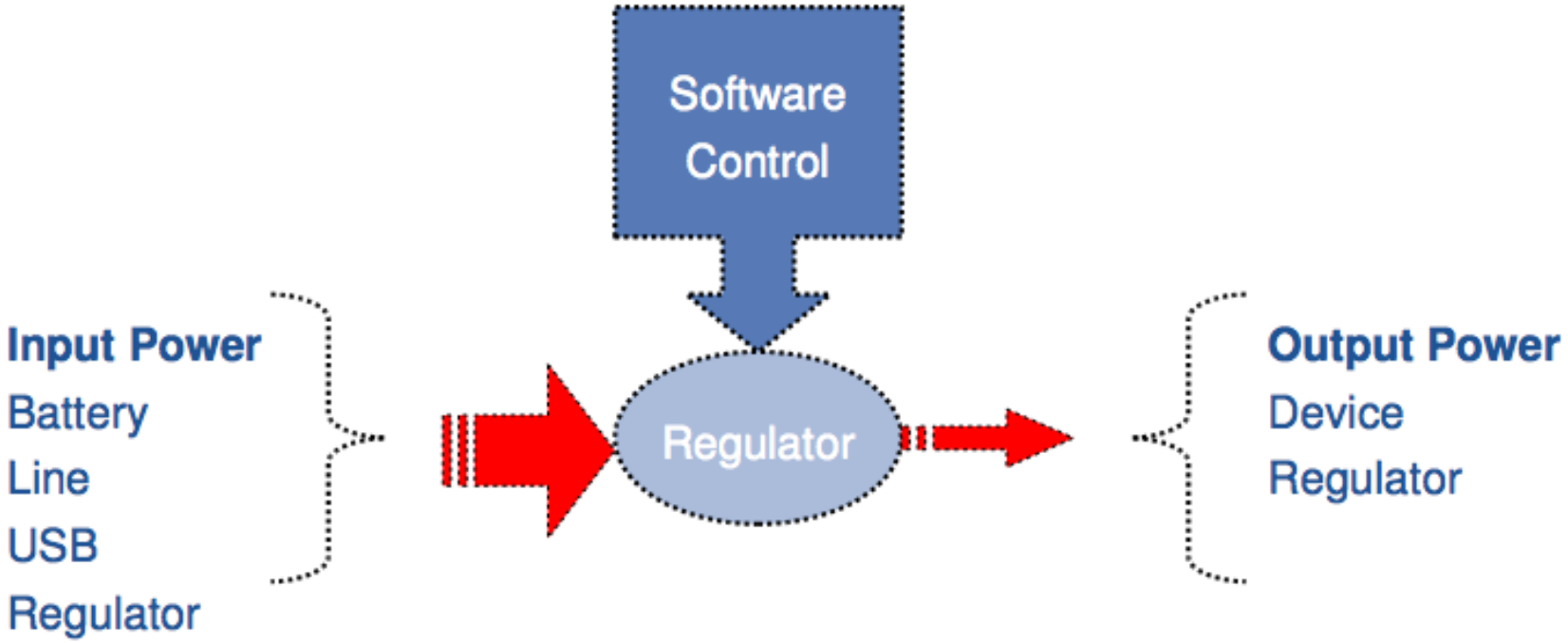
(*) 아래 함수는 clock을 제어하기 위해 linux에서 정의해 둔 API로, 함수 내부는 실제 chip에 맞게 구현해 주어야 한다(chip vendor마다 다르다).

- **clk_get** - lookup and obtain a reference to a clock producer
- **clk_enable** — inform the system when the clock source should be running.
- **clk_disable** — inform the system when the clock source is no longer required.
- **clk_put** — "free" the clock source

<일반적인 사용법>

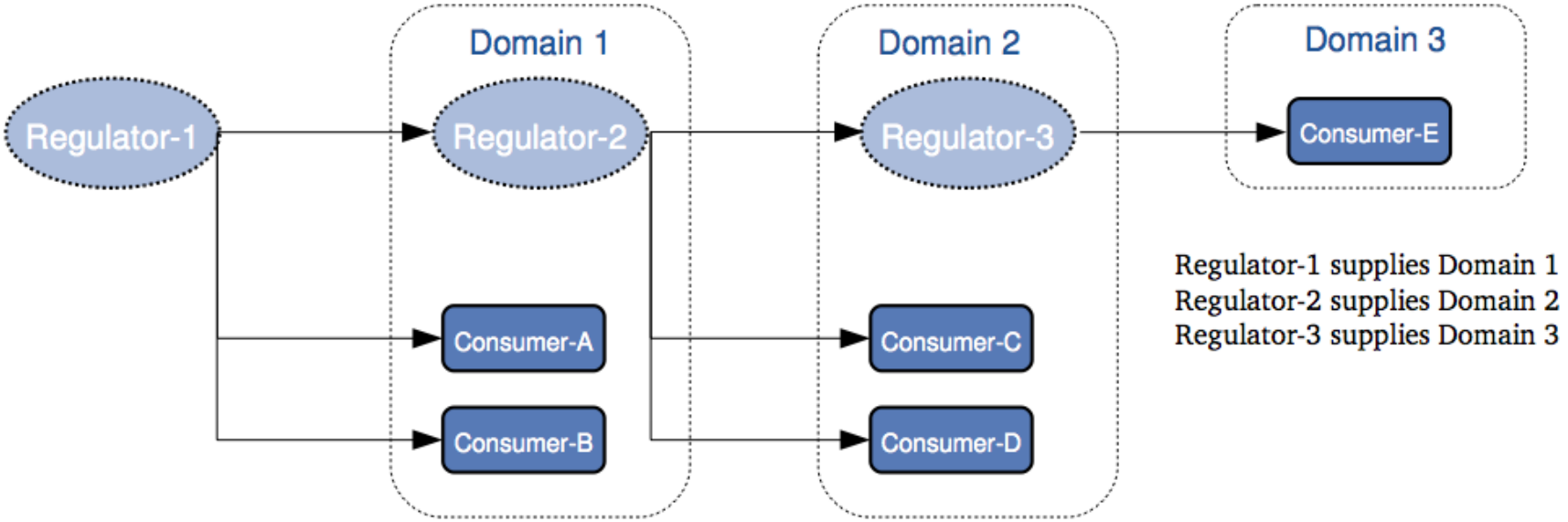
```
id = clk_get(string_name);
clk_enable(id);
clk_disable(id);
clk_set_rate(id, rate);
clk_get_rate(id);
clk_put(id);
```

4. Regulator Framework(1)



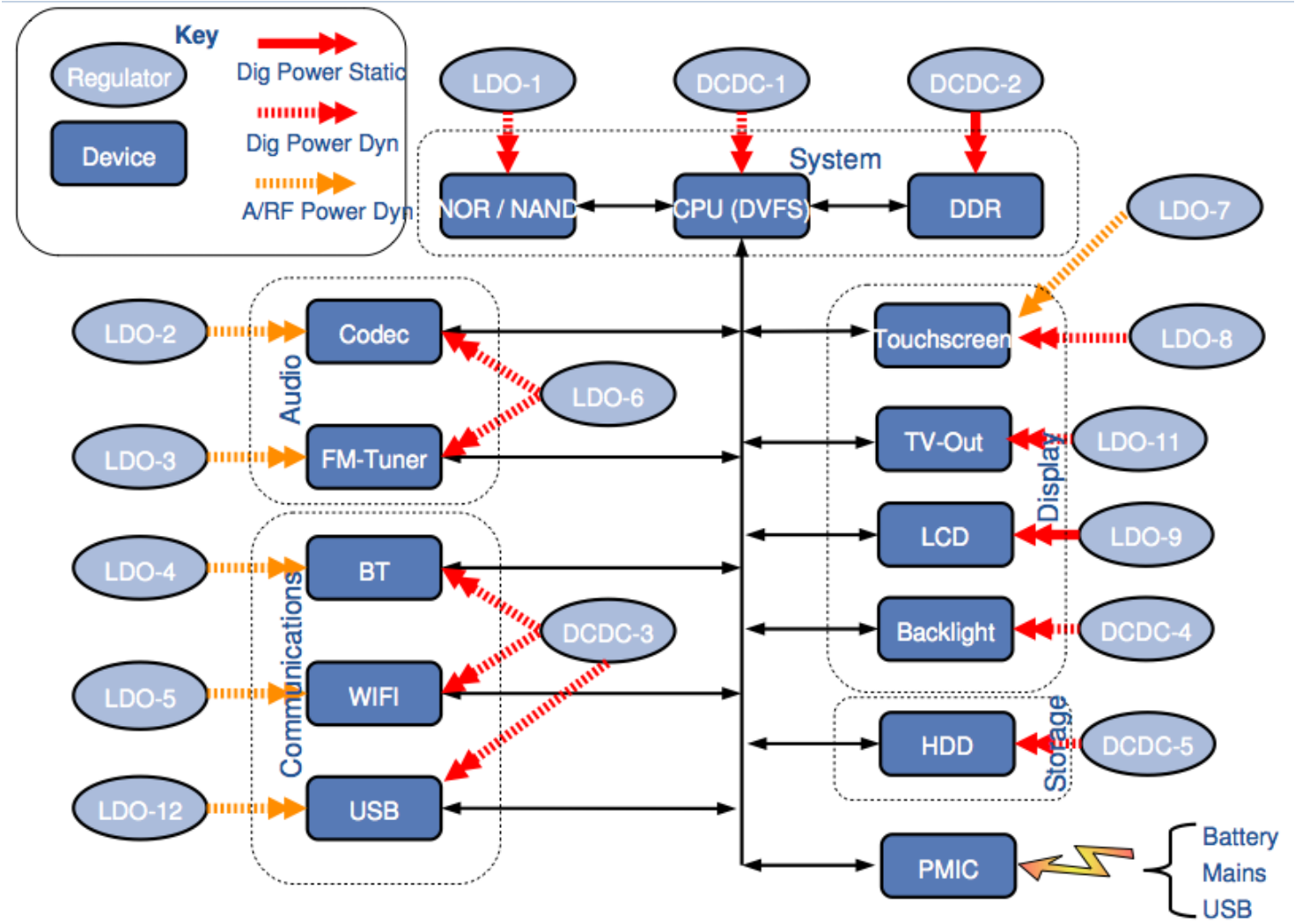
(*) Regulator는 입력 전압을 조정하여 원하는(낮거나 높은) 출력 전압을 만들어 주는 장치로, PMIC를 위시한 대부분의 장치에서 사용된다.

4. Regulator Framework(2)

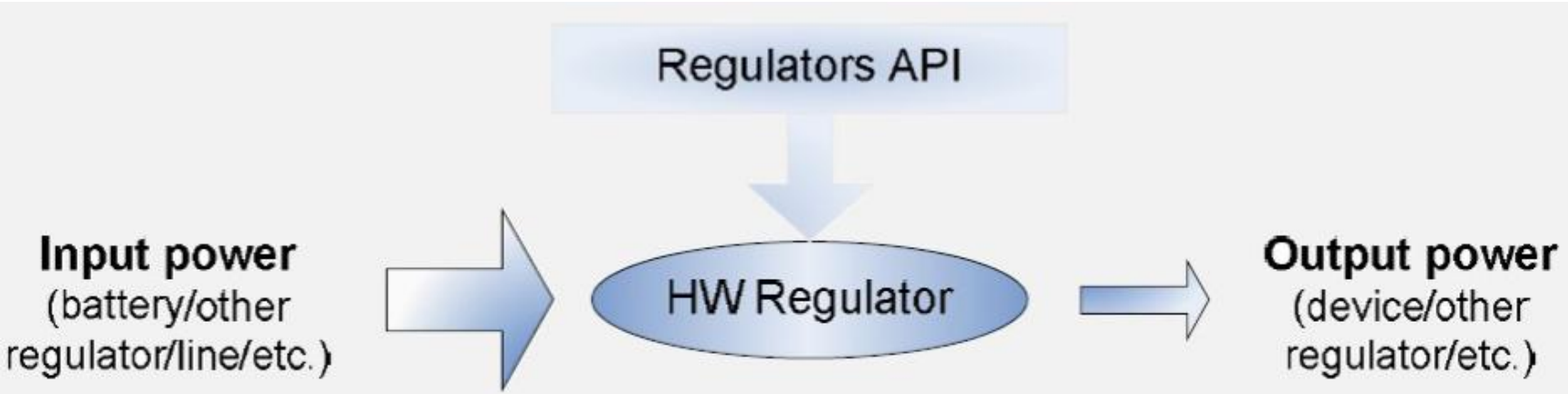


Regulator-1 supplies Domain 1
Regulator-2 supplies Domain 2
Regulator-3 supplies Domain 3

4. Regulator Framework(3)



4. Regulator Framework(4)



- Regulators have power constraints to protect hardware
- Sysfs interface (/sys/class/regulator/.../)
- Regulator independent abstraction (drivers/regulators/core.c)
- **regulator_get** - lookup and obtain a reference to a regulator
- **regulator_put** - "free" the regulator source
- **regulator_enable** - enable regulator output
- **regulator_disable** - disable regulator output

4. Regulator Framework(5) – *Consumer Interface*

- Consumer registration

  `regulator_get(), regulator_put()`

- Regulator output power control and status.

  `regulator_enable(), regulator_disable(), regulator_force_disable(), regulator_is_enabled()`

- Regulator output voltage control and status

  `regulator_set_voltage(), regulator_get_voltage()`

- Regulator output current limit control and status

  `regulator_set_current_limit(), regulator_get_current_limit()`

- Regulator operating mode control and status

  `regulator_set_mode(), regulator_get_mode(), regulator_set_optimum_mode()`

- Regulator events

  `regulator_register_notifier(), regulator_unregister_notifier()`

5. GPIO

- 1) gpio 용도
  - 입력 값을 읽어 들이거나, 출력 값을 내보내는 역할
  (외부 장치로 부터 전달된 값을 읽거나, 외부 장치로 값을 쓸 경우)
  - Interrupt pin으로 사용
- 2) 관련 API(내부는 vendor마다 다르게 구현함)

  - *gpio_is_valid()*
  - *gpio_request()*
  - *gpio_free()*
  - *gpio_direction_input()*
  - *gpio_direction_output()*
  - *gpio_get_value()*
  - *gpio_set_value()*
  - *gpio_to_irq()*
  - …

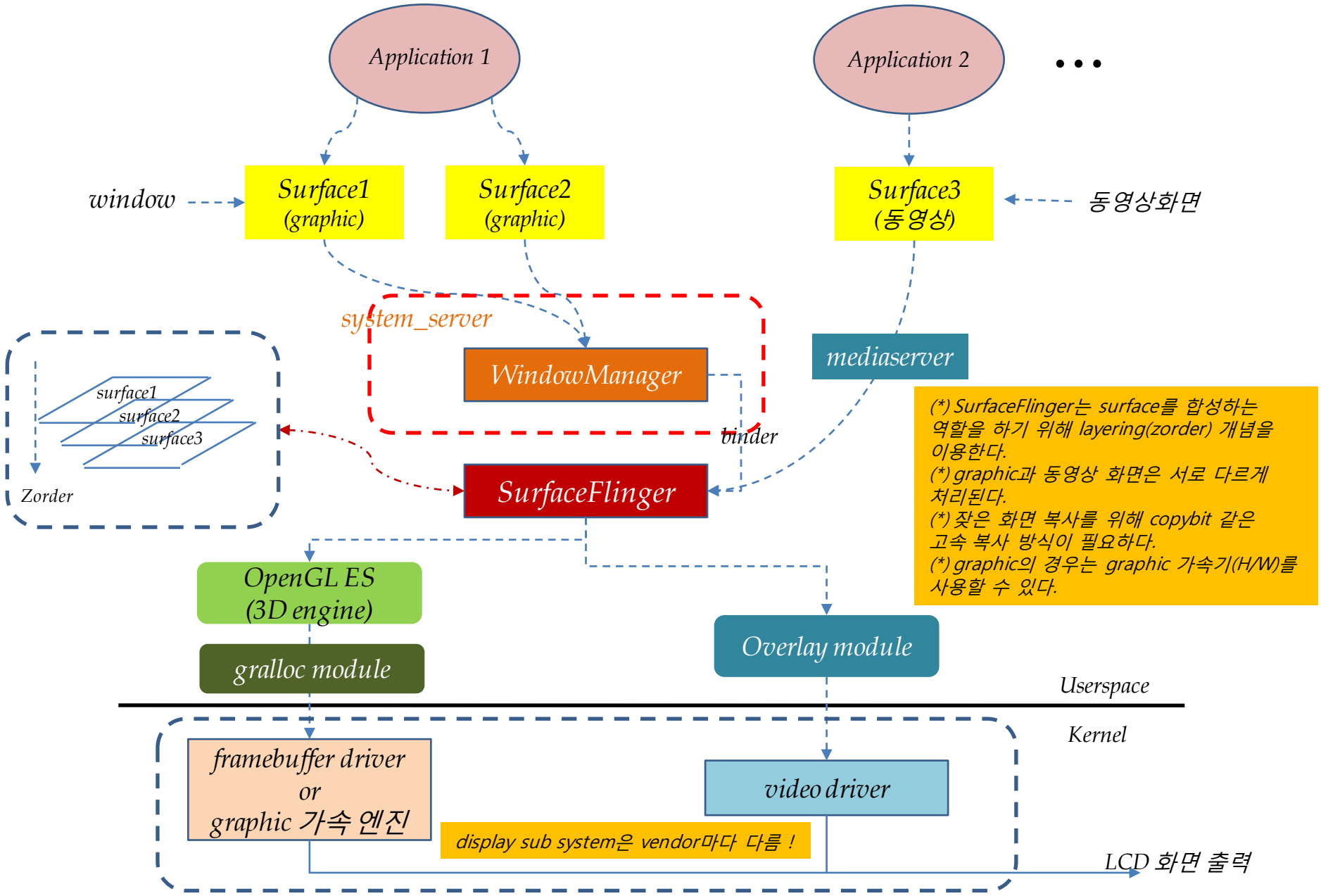
6. Suspend/Resume

- *Android_PM_Guide7.pdf 참조*

2. Display

: LCD interface, backlight, framebuffer

1. Android Display Overview

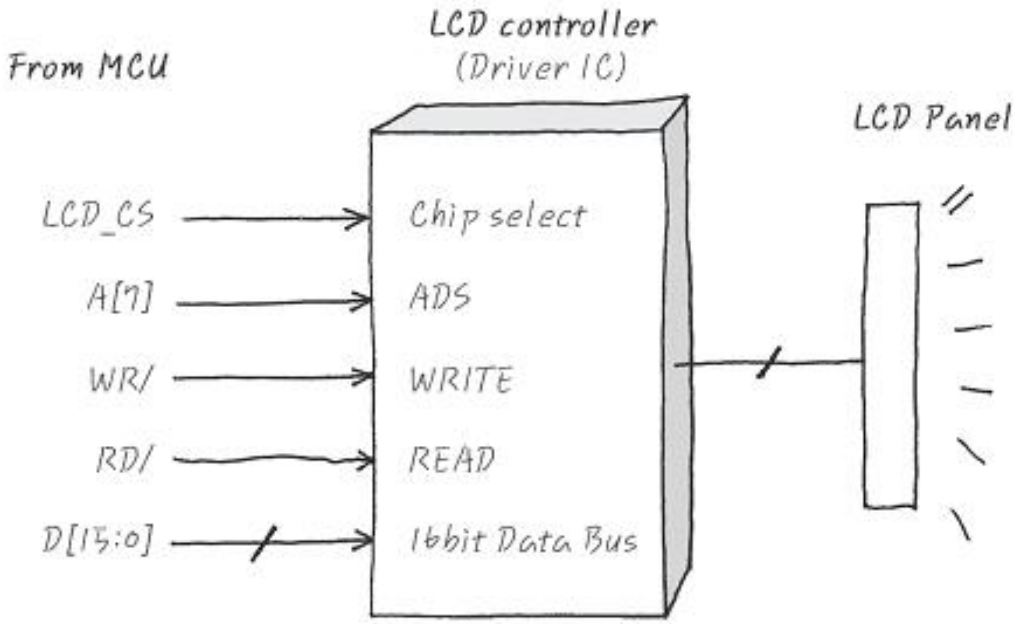


2. LCD Interface(1)

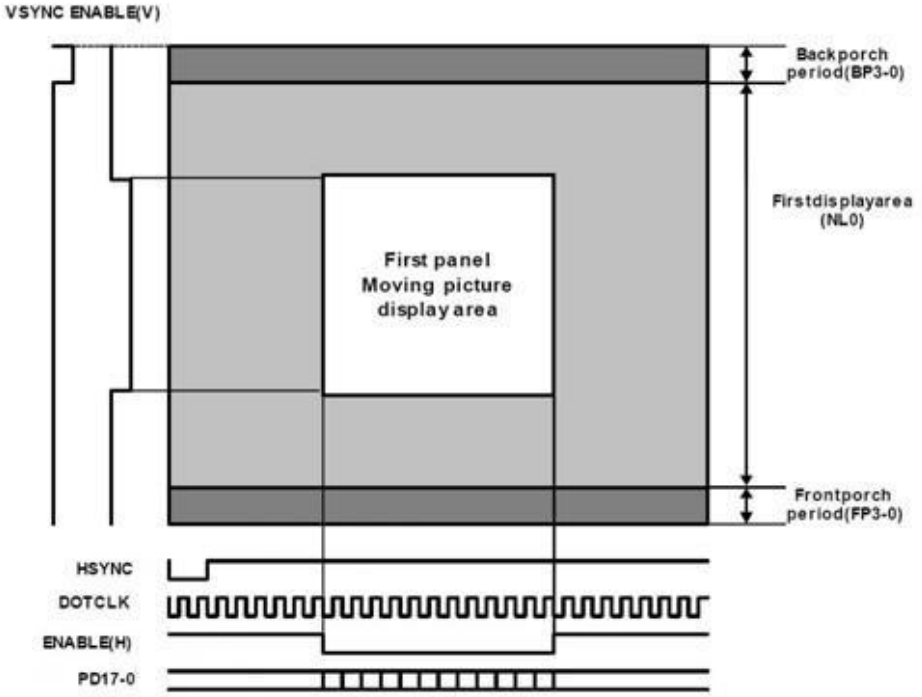
- *CPU interface*
- *RGB interface*
- *SPI interface*
- *MDDI interface*
- *MIPI interface*

Interface 종류	제어 신호
CPU Interface	*WR, RD, RS, CS*
RGB Interface	*VSYNC, HSYNC, Enable, DOTCLK*
Serial Interface	*SDI(MOSI), SDO(MISO), SCL, CS*

2. LCD Interface(2) – ***CPU Interface***



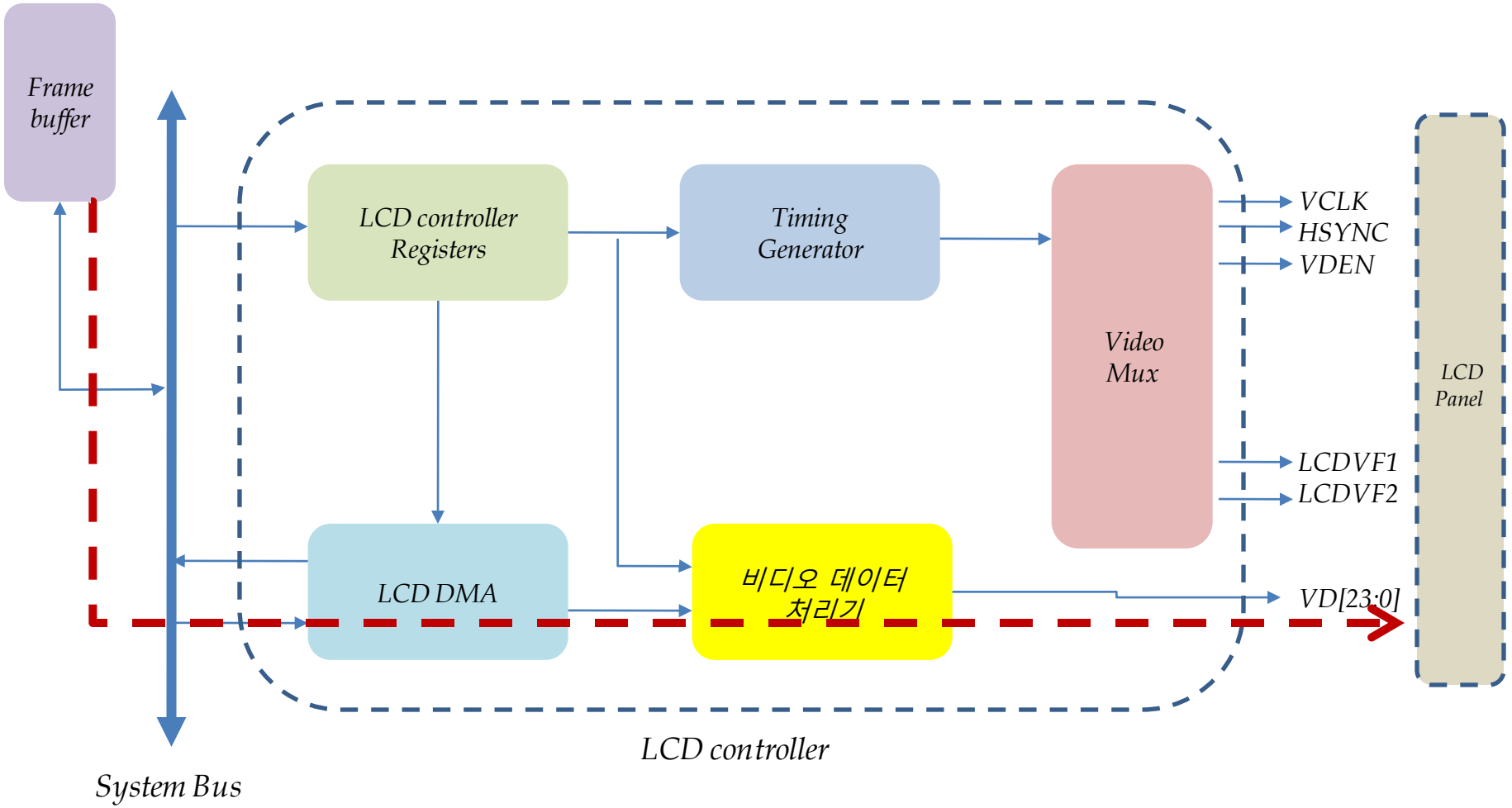
2. LCD Interface(3) – *RGB Interface/1*



Register Name	Used to Configure
SIZE_REG	LCD panel's maximum X and Y dimensions
HSYNC_REG	***HSYNC*** duration
VSYNC_REG	***VSYNC*** duration
CONF_REG	Bits per pixel, pixel polarity, clock dividers for generating pixclock, color/monochrome mode, and so on
CTRL_REG	Enable/disable LCD controller, clocks, and DMA
DMA_REG	Frame buffer's DMA start address, burst length, and watermark sizes
STATUS_REG	Status values
CONTRAST_REG	Contrast level

2. LCD Interface(3) – ***RGB Interface/2***

(*) 아래 그림은 framebuffer의 내용(pixel 정보)이 LCD controller 내의 DMA를 통해 LCD panel로 바로 출력되는 것을 보여준다. 따라서, 초기에 LCD controller를 제대로 설정해 놓기만 한다면, 이후 동작은 framebuffer에 값을 쓰는 것 만으로도 충분할 것이다.

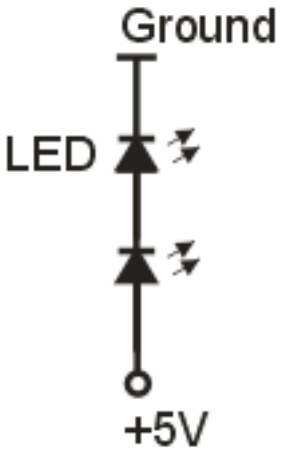


2. LCD Interface(4) – ***MIPI Interface***

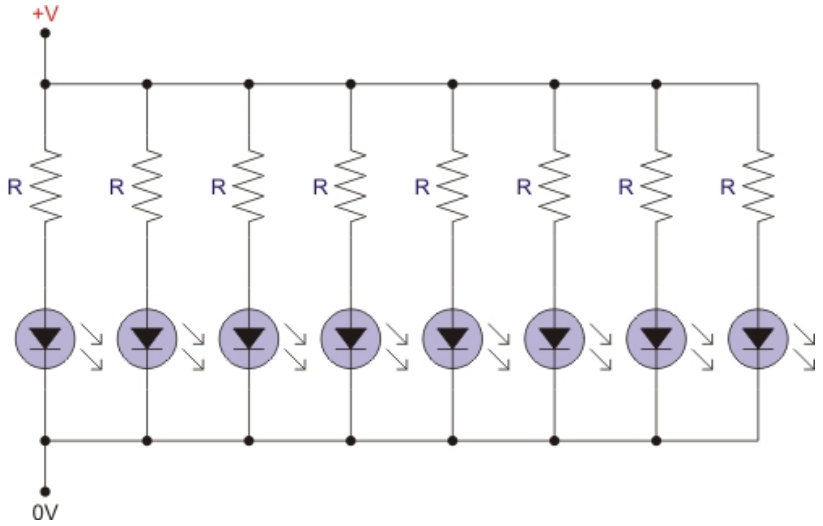
- <TODO>

3. LED Backlight Device & Driver(1)

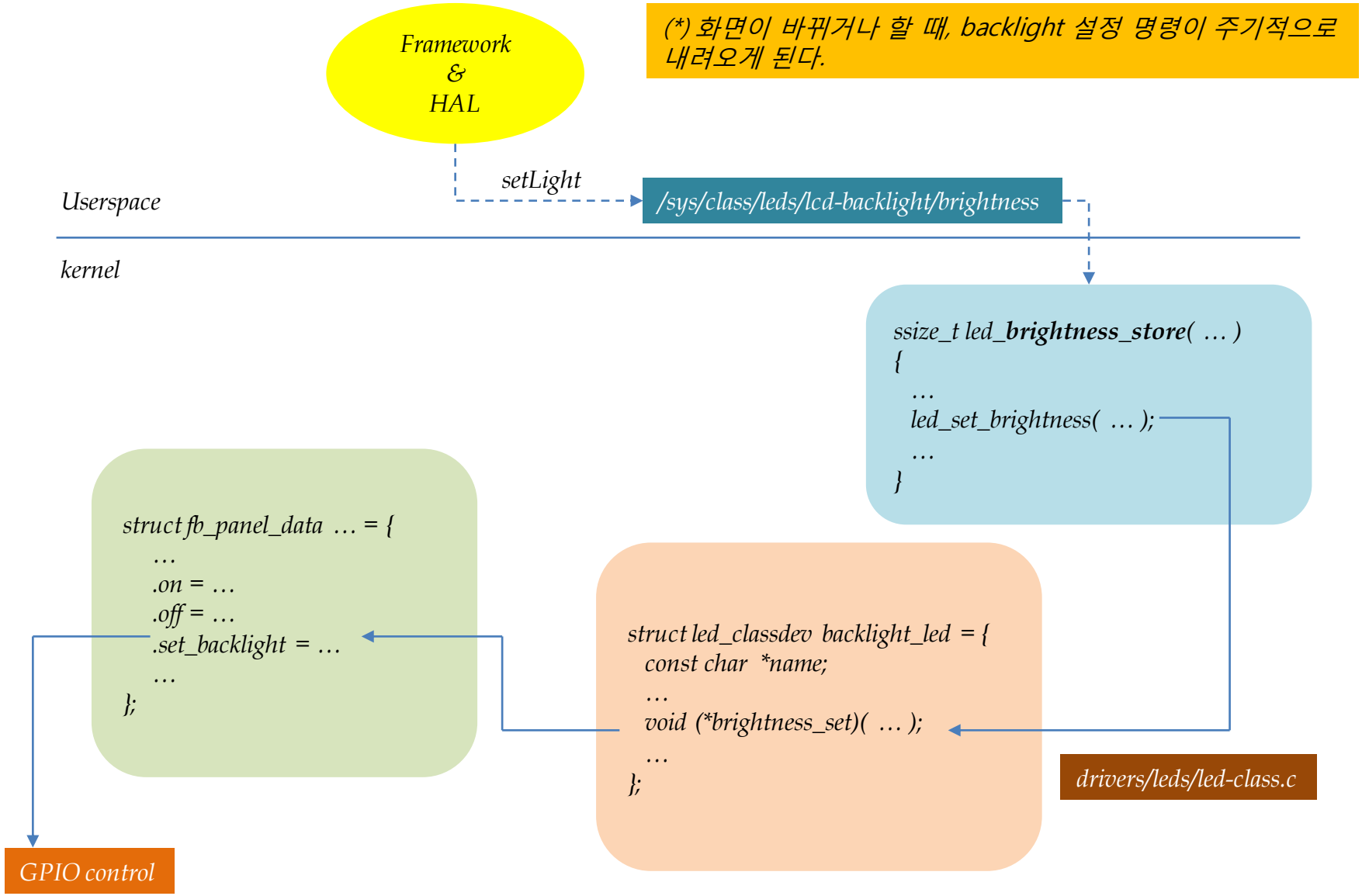
SERIAL



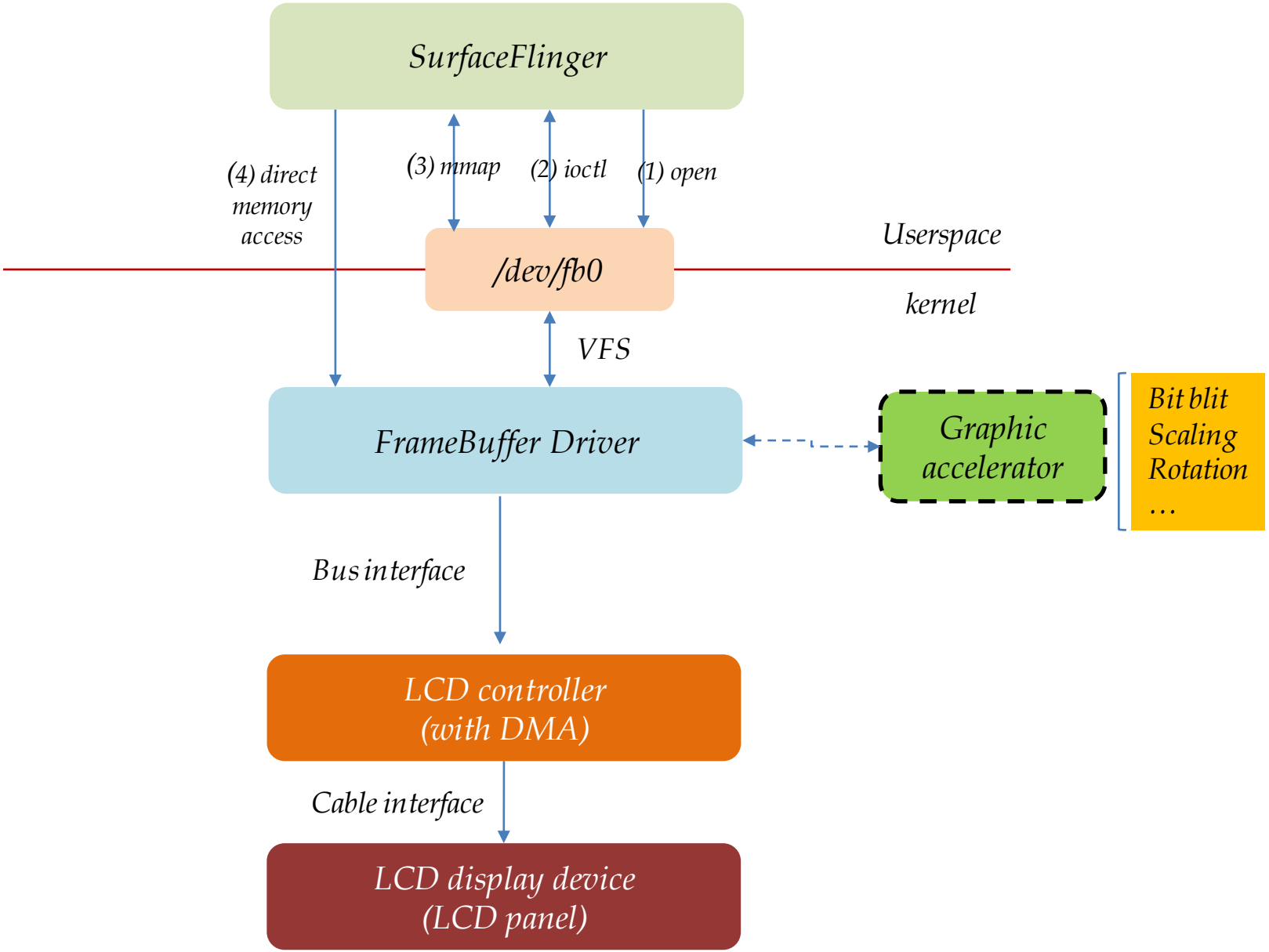
PARALLEL



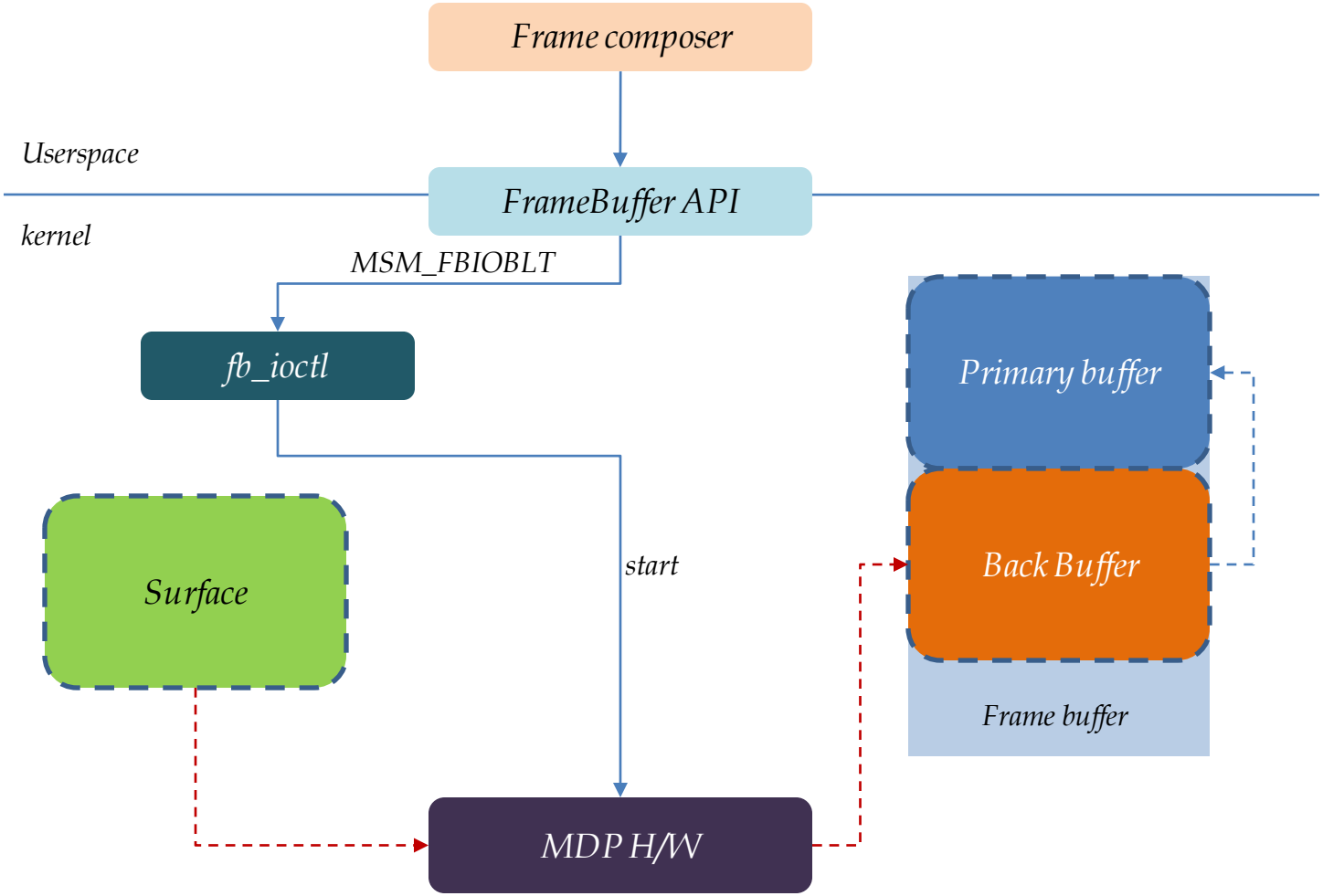
3. LED Backlight Device & Driver(2)



4. Display Subsystem(1) - ***Overview***

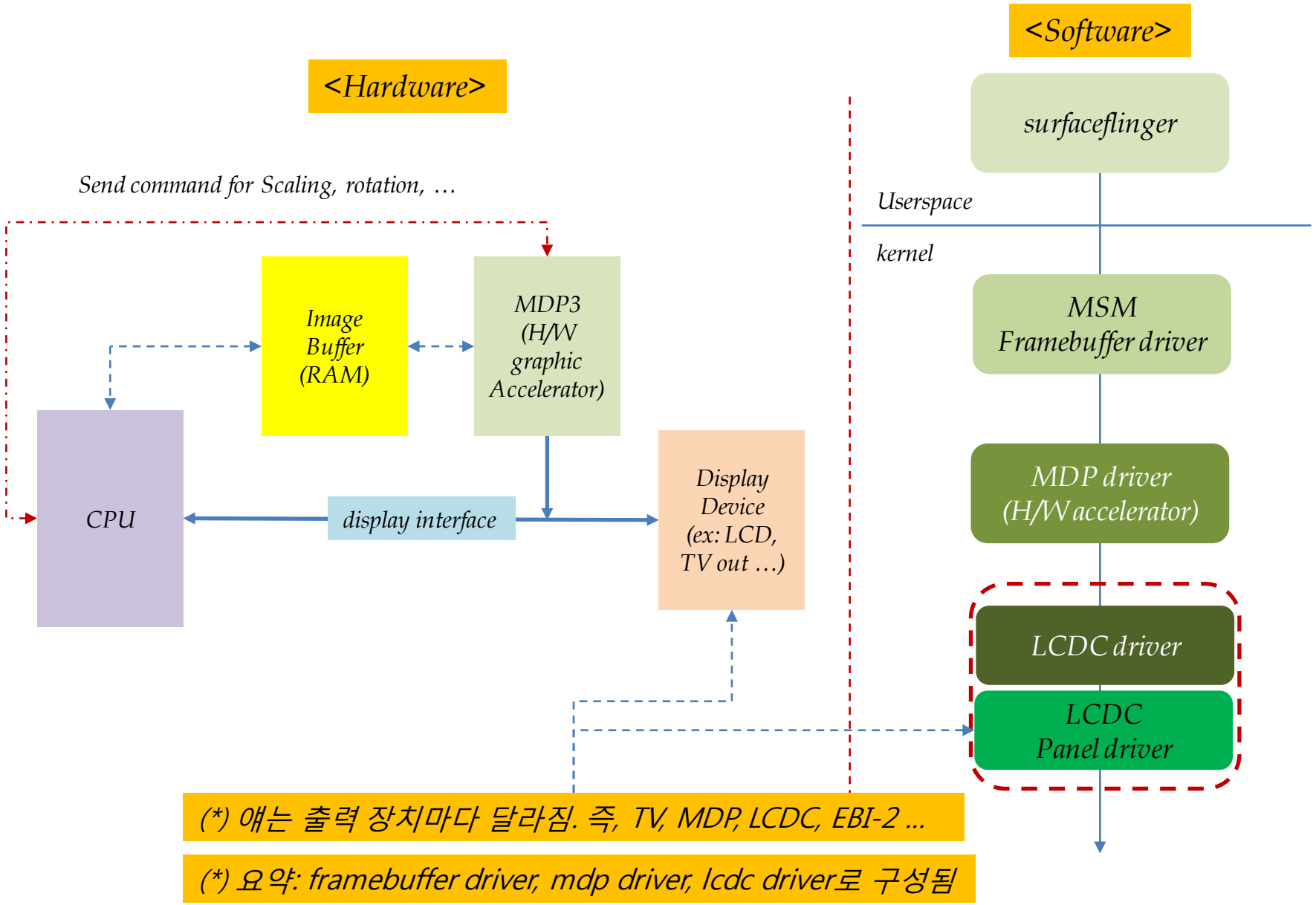


4. Display Subsystem(2) – Qualcomm MDP/1



(*) MDP HW를 통해 bit blit등의 hardware graphic 가속 처리가 이루어진다.
(*) 화면 떨림을 없애기 위해 framebuffer 상에 back buffer를 두고 있으며, 모든 surface는 일차적으로 MDP를 거쳐 back buffer로 출력되고, 최종적으로 merge된 화면은 다시 primary buffer로 출력되는 과정을 통해 화면에 보여지게 된다.
(*) framebuffer 관련 기본 operation은 위에서 보는 바와 같이 fb_ioctl() 함수를 통해 이루어 진다.

4. Display Subsystem(3) - ***Qualcomm MDP/2***



5. TVout/HDMI

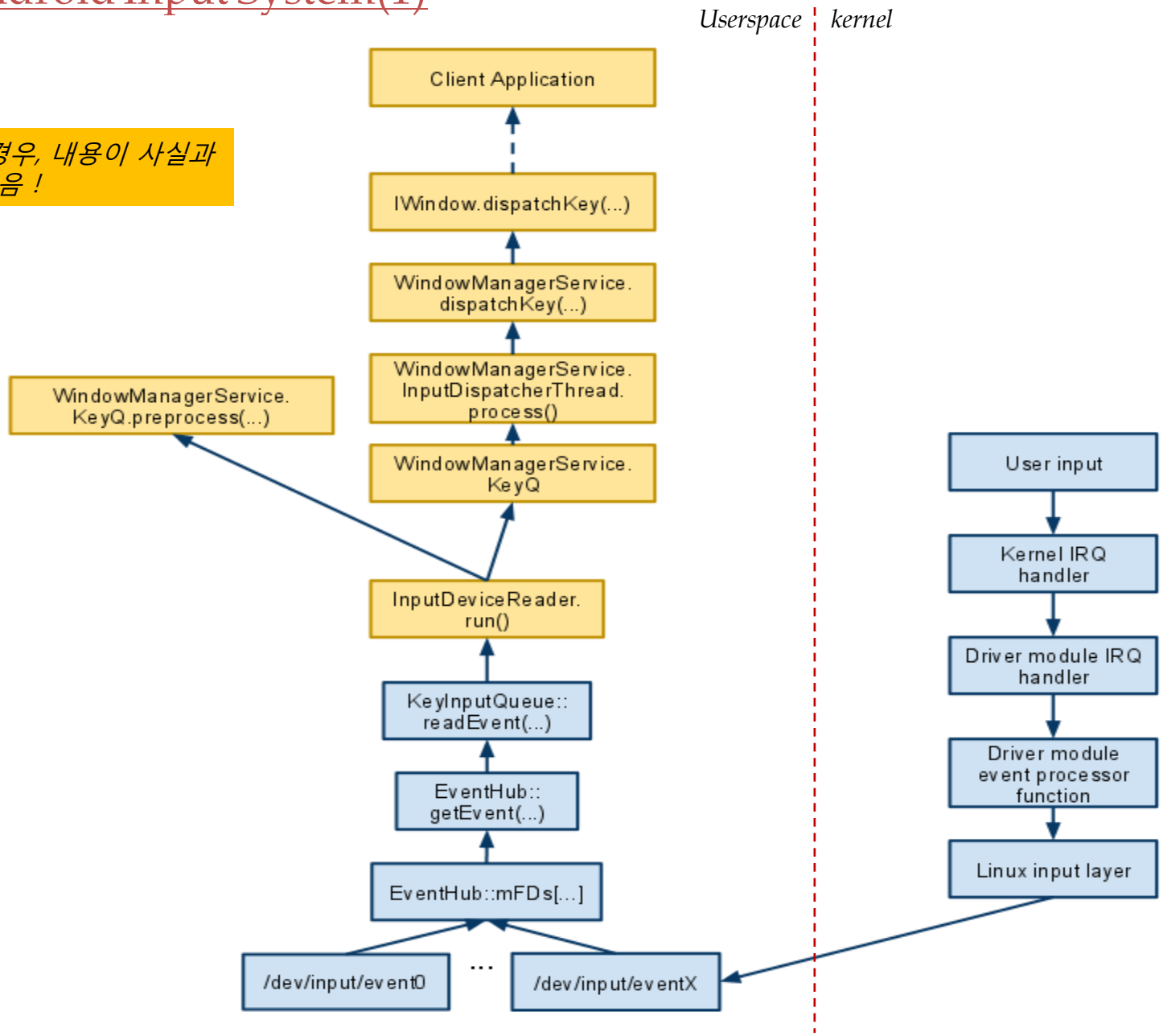
- *<TODO>*

3. Input Device

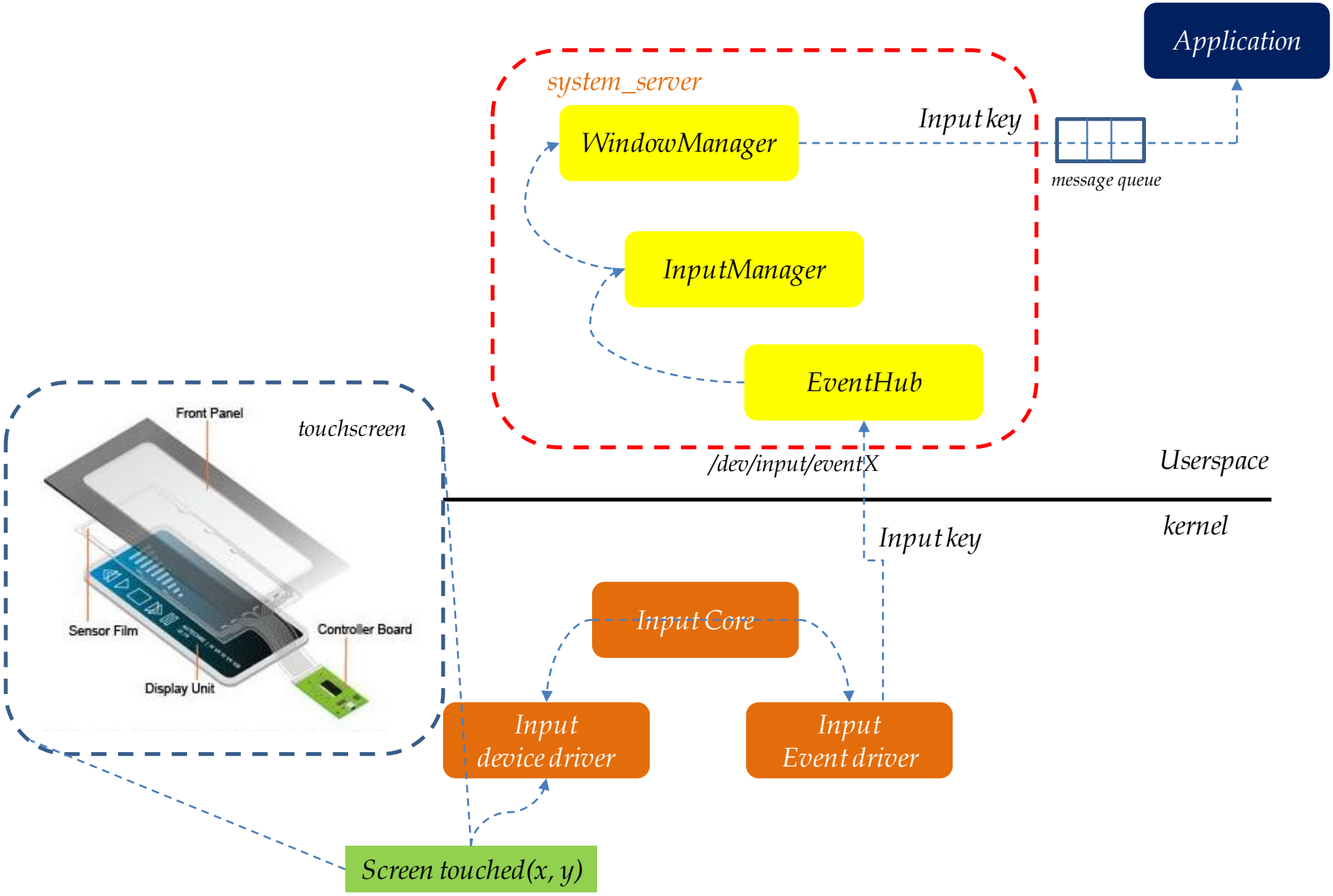
: Touchscreen & KeyPad Driver

1. Android Input System(1)

(*) ICS의 경우, 내용이 사실과 다를 수 있음 !

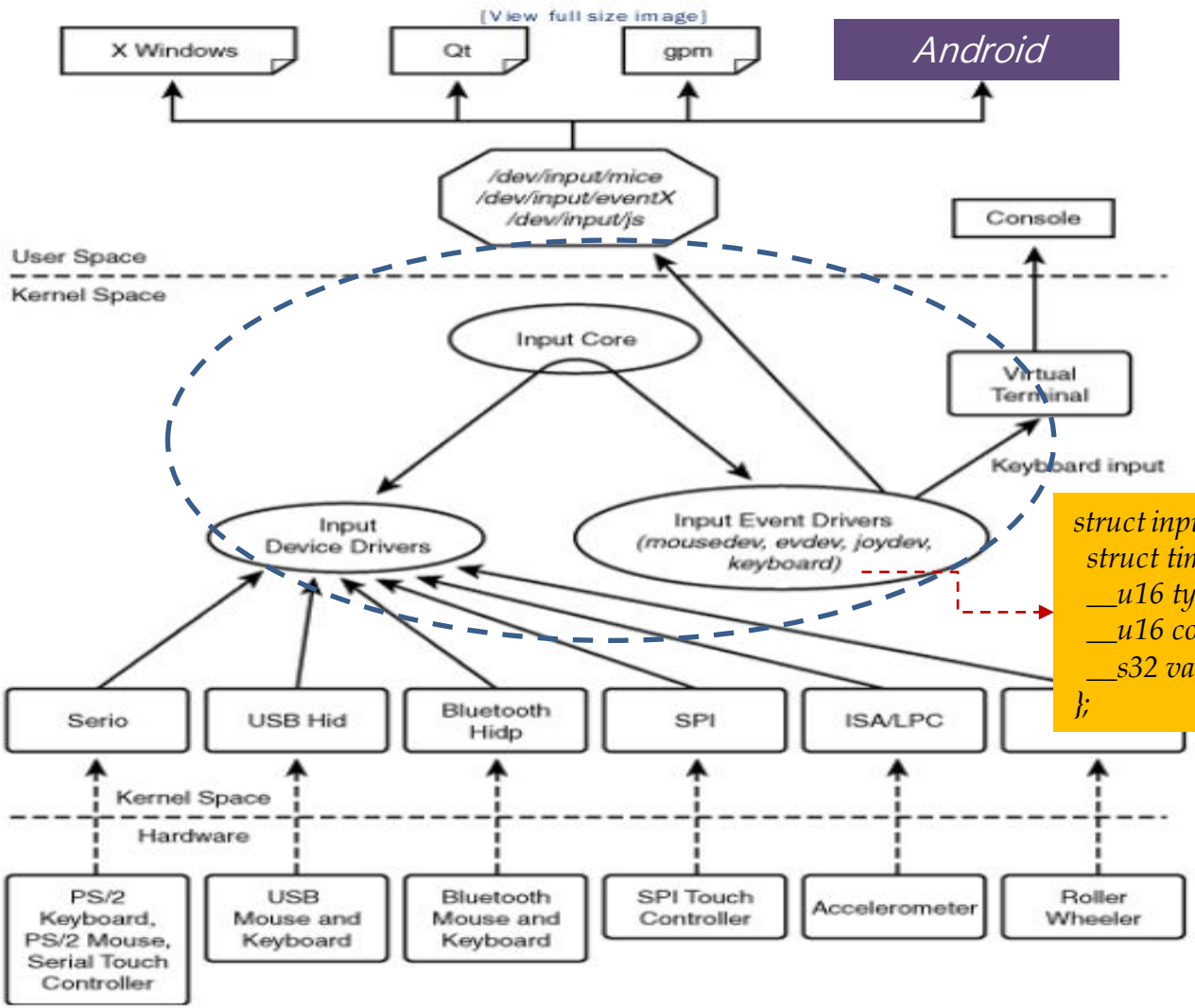


1. Android Input System(2)



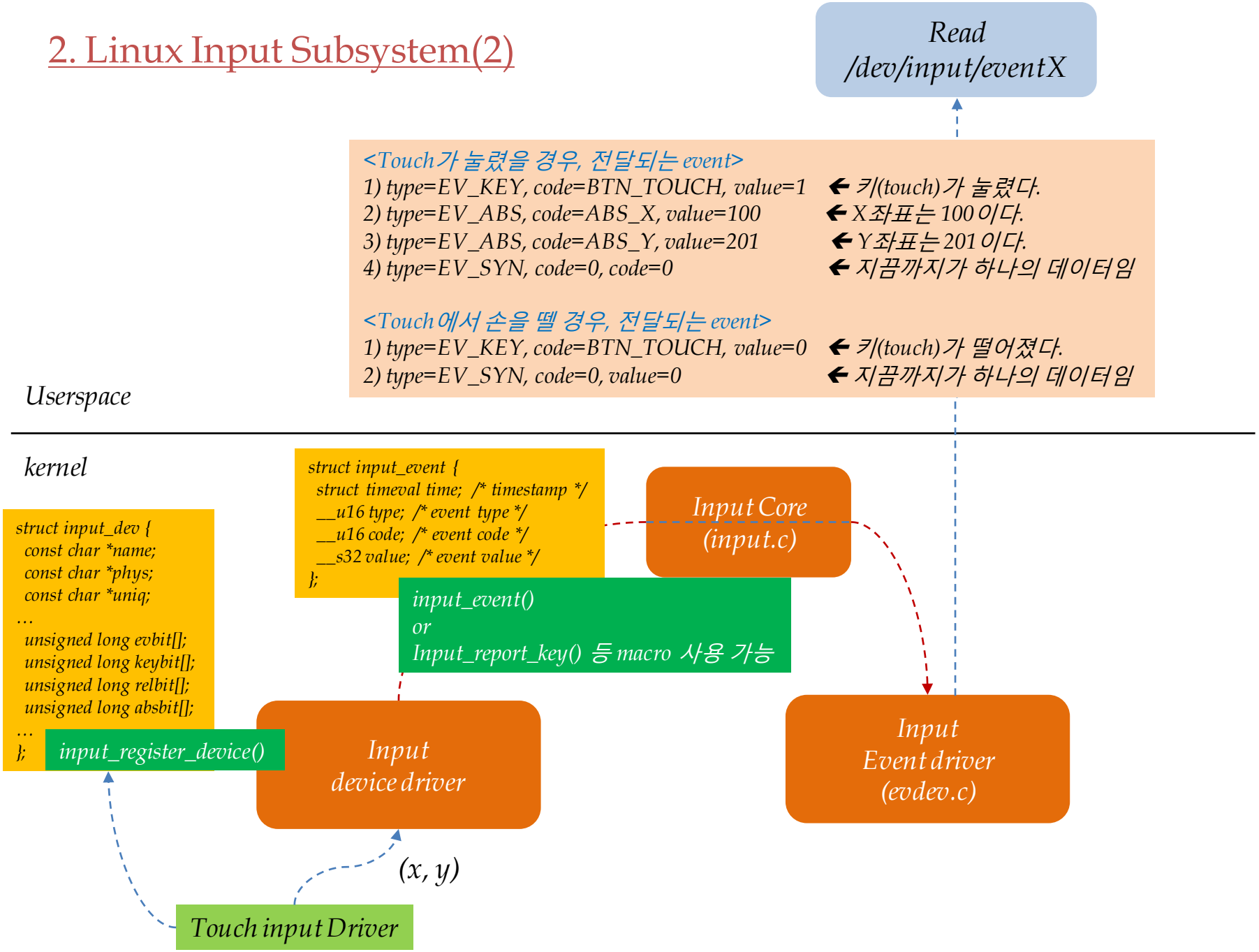
2. Linux Input Subsystem(1)

(*) 아래 그림은 참고 문서[1]에서 복사해 온 것임.




```
struct input_event {
  struct timeval time;  /* timestamp */
  __u16 type;  /* event type */
  __u16 code;  /* event code */
  __s32 value;  /* event value */
};
```

2. Linux Input Subsystem(2)



2. Linux Input Subsystem(3)

<Touch가 눌린 경우 event 전달 함수 호출 예>

```
1) input_report_key(&my_touch_drv,  BTN_TOUCH,  1);
2) input_report_abs(&my_touch_drv,  ABS_X,  100);
3) input_report_abs(&my_touch_drv,  ABS_Y,  201);
4) input_sync(&my_touch_drv);
```

<Touch에서 손을 뗄 경우 event 전달 함수 호출 예>

```
1) input_report_key(&my_touch_drv,  BTN_TOUCH,  0);
2) input_sync(&my_touch_drv);
```

(*) 위의 함수들은 모두 input_event() 함수를 호출하는 inline 함수들이다.

2. Linux Input Subsystem(4)

static inline void init_input_dev(struct input_dev *dev);
dev 에 전달된 구조체를 초기화 한다.

void input_register_device(struct input_dev *);
입력 장치를 등록한다.

void input_unregister_device(struct input_dev *);
입력 장치를 제거한다.

void input_event(struct input_dev *dev, unsigned int type, unsigned int code, int value);
이벤트를 이벤트 핸들러 디바이스 드라이버에 전달한다.

static inline void input_report_key(struct input_dev *dev, unsigned int code, int value);
내부적으로 input_event 함수를 이용하여 버튼 또는 키 이벤트를 전달한다.

2. Linux Input Subsystem(5)

static inline void input_report_rel(struct input_dev *dev, unsigned int code, int value);
내부적으로 input_event 함수를 이용하여 이동 된 크기 이벤트를 전달한다.

static inline void input_report_abs(struct input_dev *dev, unsigned int code, int value);
내부적으로 input_event 함수를 이용하여 이동 된 절대 좌표 이벤트를 전달한다.

static inline void input_sync(struct input_dev *dev);
내부적으로 input_event 함수를 이용하여 하나의 상태에 대한 여러 이벤트가 동기 되어야 할 필요가 있다는 것을 전달한다.

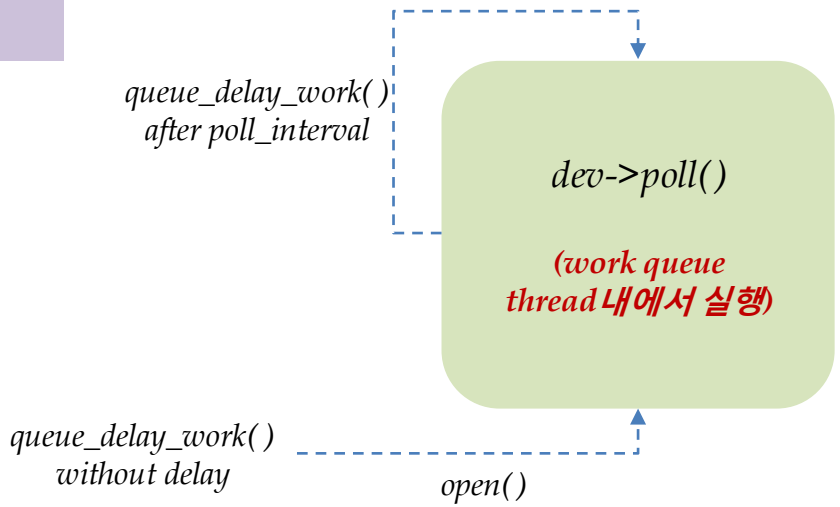
static inline void input_set_abs_params(struct input_dev *dev, int axis, int min, int max, int fuzz, int flat);
절대 좌표 값을 지정하는 경우 값의 최소 값과 최대 값을 지정한다.

2. Linux Input Subsystem(6)

```
struct input_polled_dev {
    void *private;

    void (*open)(struct input_polled_dev *dev);
    void (*close)(struct input_polled_dev *dev);
    void (*poll)(struct input_polled_dev *dev);
    unsigned int poll_interval; /* msec */
    unsigned int poll_interval_max;/* msec */
    unsigned int poll_interval_min;/* msec */

    struct input_dev *input;

/* private: */
    struct delayed_work work;
};
```

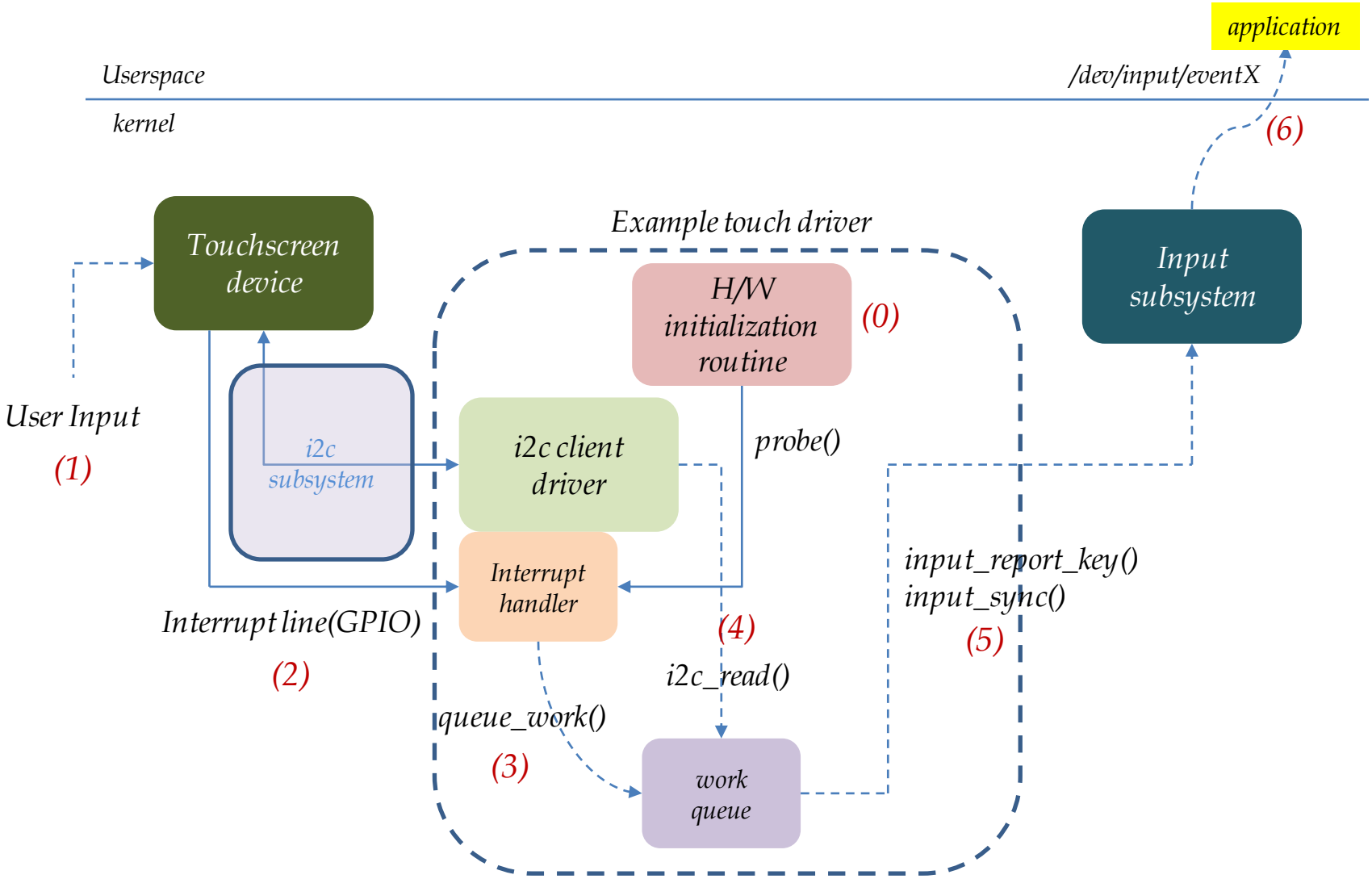
<include/linux/input-polldev.h>

(*) input_polled_dev는 user space에서 주기적으로 값을 읽어가는 것과는 달리,
(*) work queue를 이용하여 지정된 시간(polled_interval) 마다 값을 읽어 user space로 던져주는 방식이다.



<drivers/input/input-polldev.c>

3. Touch Driver(1) – *I2C Interface*



3. Touch Driver(2) – *I2C Interface*

<Touch Input Flow>

0) touchscreen driver의 경우 전송되는 data의 량이 많지 않으므로 i2c driver 형태로 구현함(이건 장치마다 서로 상이함. SPI로 구현하기도 함).

1) 사용자의 touch 입력에 대해 interrupt가 들어온다.

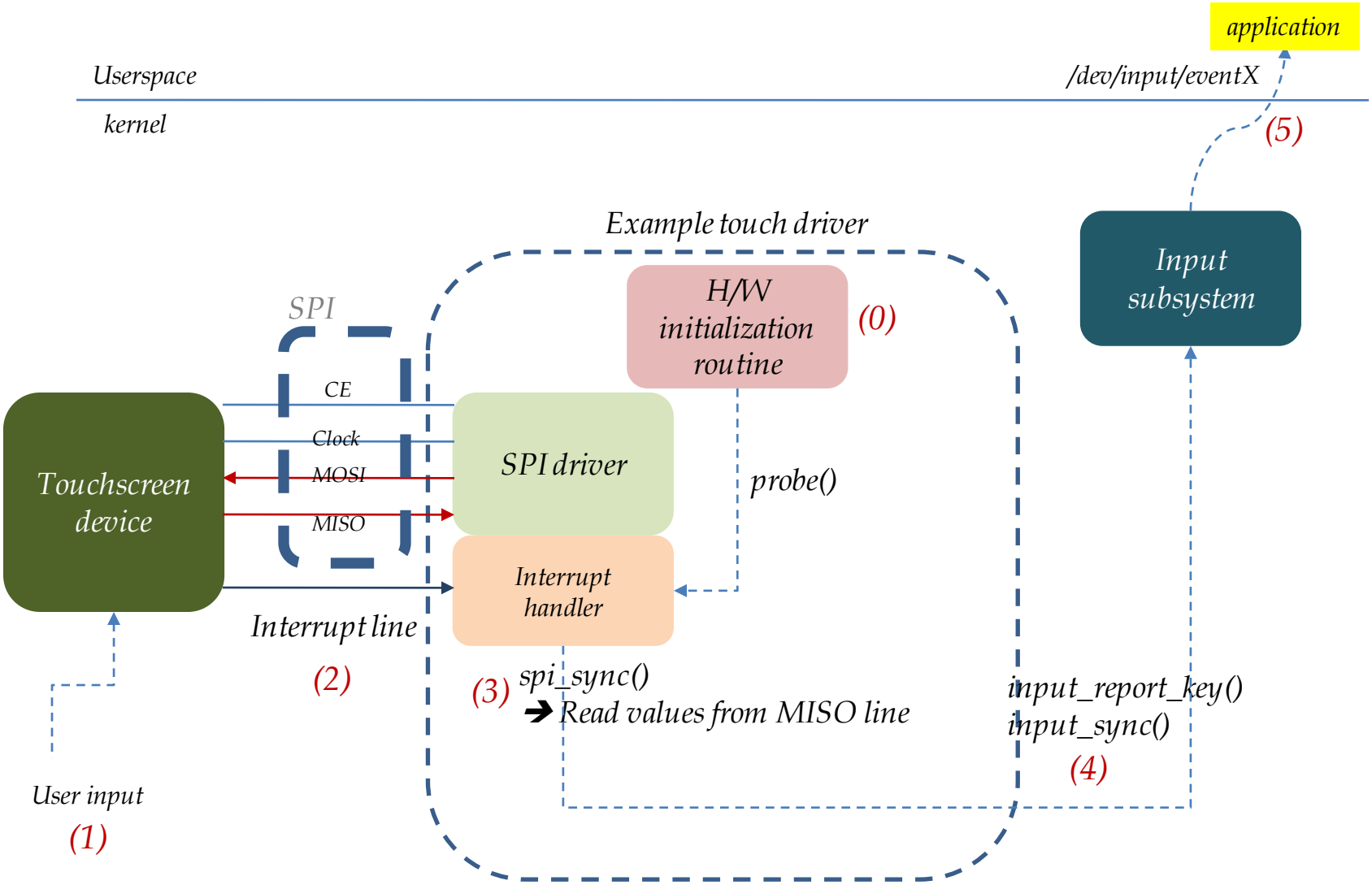
2) Interrupt handler의 내부는 지연 처리가 가능한 work queue 형태로 구현하였으며, 따라서 interrupt가 들어올 경우 work queue routine이 동작하도록 schedule해준다.

3) Work queue routine은 i2c_read( ) 함수를 이용하여 i2c로 입력된 정보(touch 좌표 정보)를 읽어 들인다.

4) 읽어들인 i2c data를 input subsystem에서 인식할 수 있는 정보로 변형하여 input subsystem을 전달한다.

5) Application은 /dev/input/eventX 장치 파일로 부터 touch 정보를 읽어 들인다.

3. Touch Driver(3) – ***SPI Interface***



4. KeyPad Driver - ***GPIO***

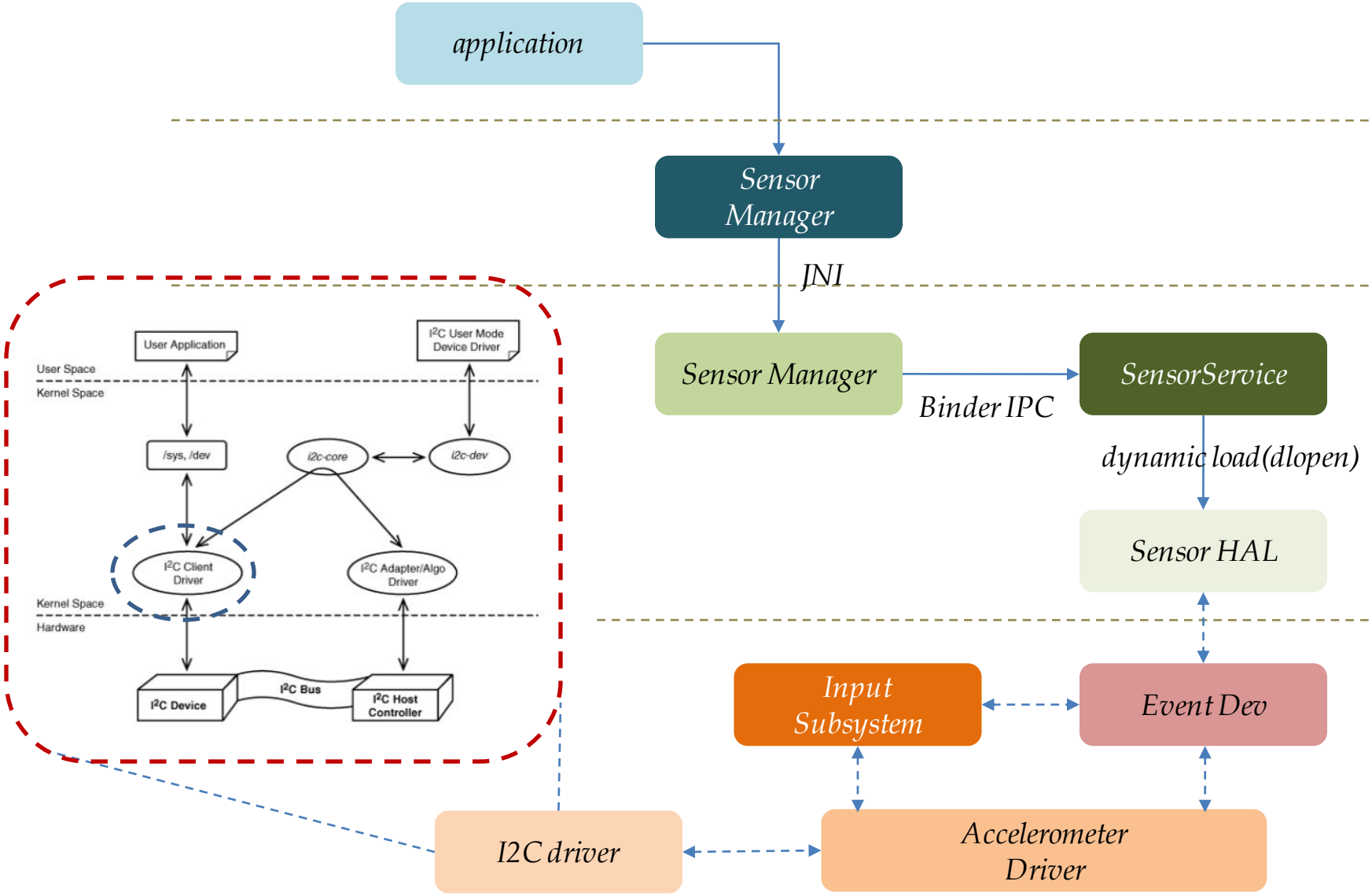
- <TODO>

4. Sensor Drivers

1. ADC(Analog-to-Digital Converter)

- *<TODO>*

2. Android Sensor Architecture: ***Overview***



3. Sensor 종류

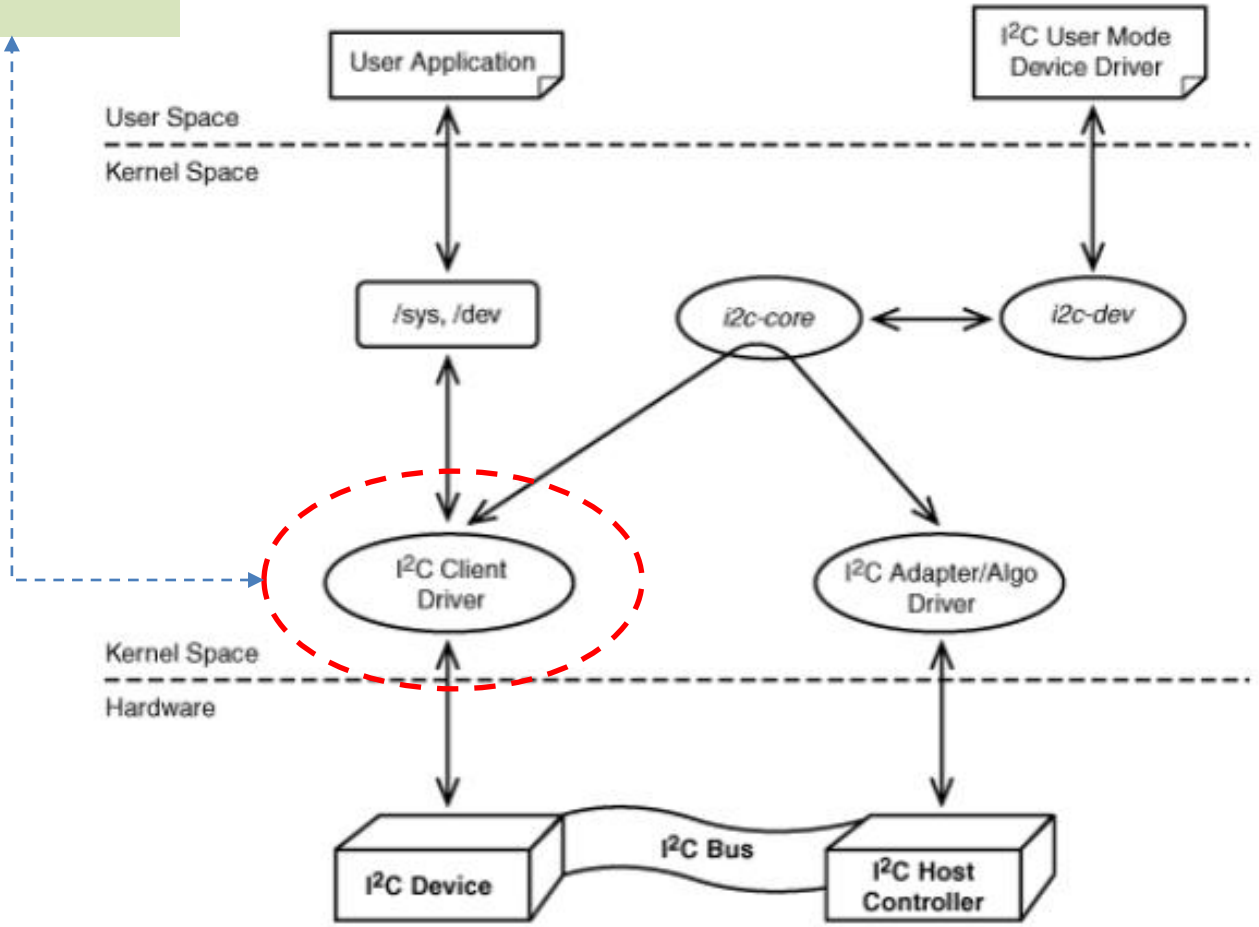
- *1) **accelerometer** : 가속도 감지(흔들림 감지) 센서*
- *2) **geomagnetic** : 주변의 자기장을 감지하는 센서*
- *3) **orientation** : 기기의 방향을 감지하는 센서*
- *4) **proximity** : 특정 물체와 근접한 정도를 감지하는 센서*
- *5) **gyroscope** : 모션 센서의 정밀한 교정을 위해 쓰이는 자이로스코프 센서*
- *6) **light** : 주변의 빛을 감지하는 센서*
- *7) **pressure** : 기기에 적용되는 압력을 감지하는 센서*
- *8) **temperature** : 기기 근처의 온도를 감지하는 센서*

(*) Sensor driver의 경우는 대부분은 i2c client driver 형태로 구현되며, input subsystem을 이용하여 정보를 user space로 전달하는 형태로 구성되어 있다.

4. Sensor I2C Interface: *I2C Client Driver*

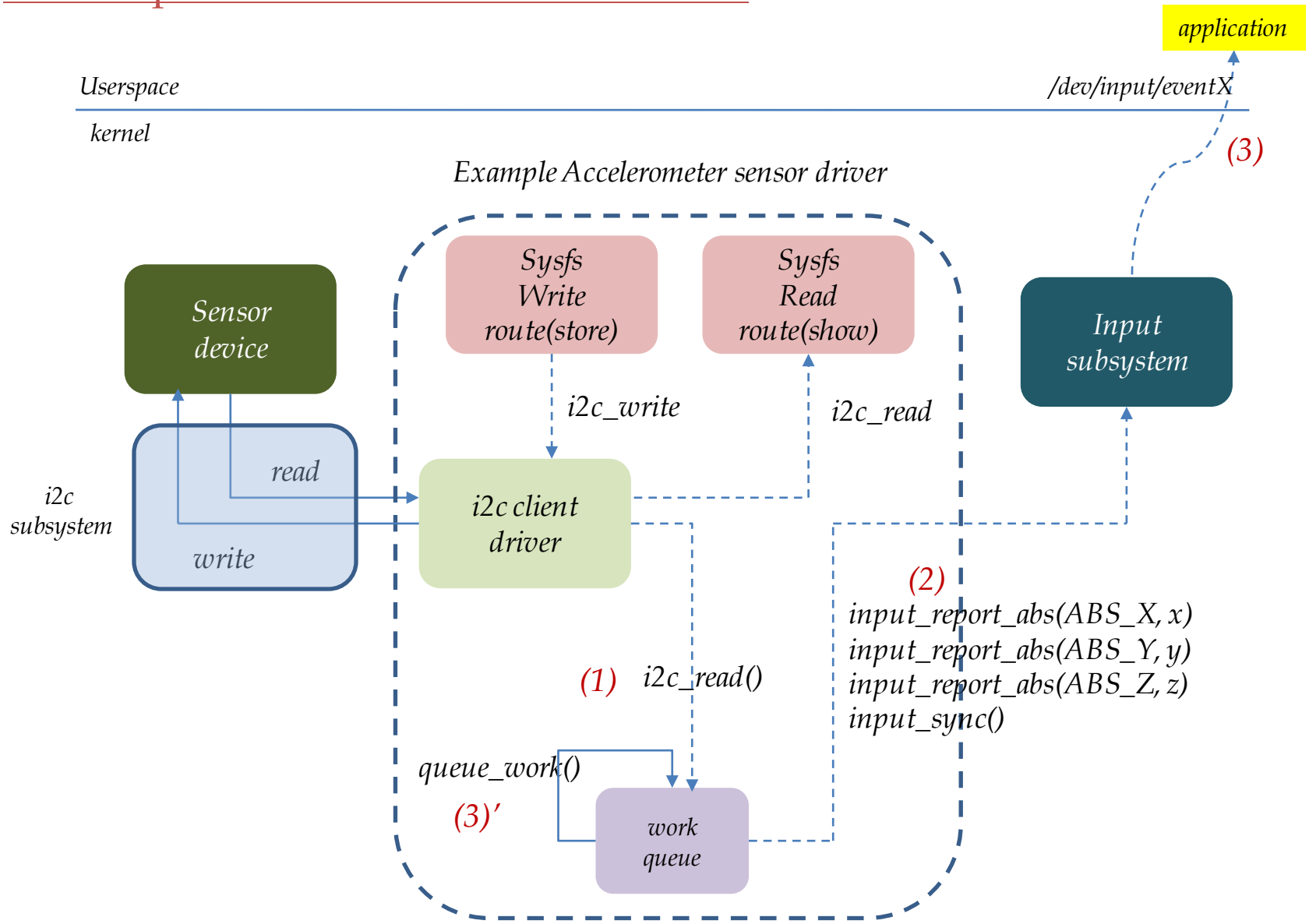
1) Acceleration Sensor
2) Geomagnetic Sensor
3) Proximity Sensor
...

(*) 아래 그림은 Sensor 장치의 인터페이스로 사용되는 i2c client 드라이버의 전체 구조를 정리한 것이다.

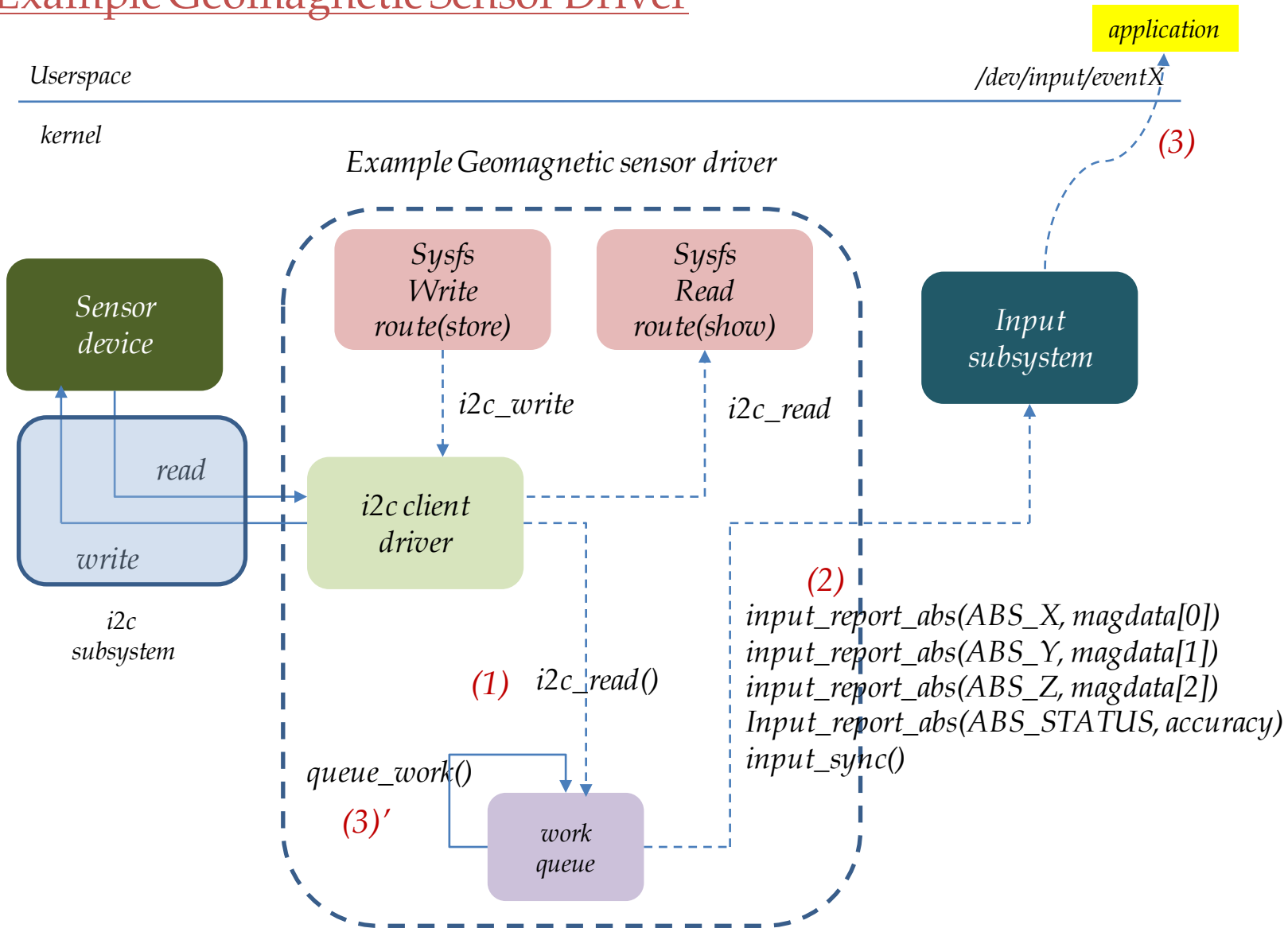


(*) 위 그림은 참고 문서[1]에서 복사해 온 것임.

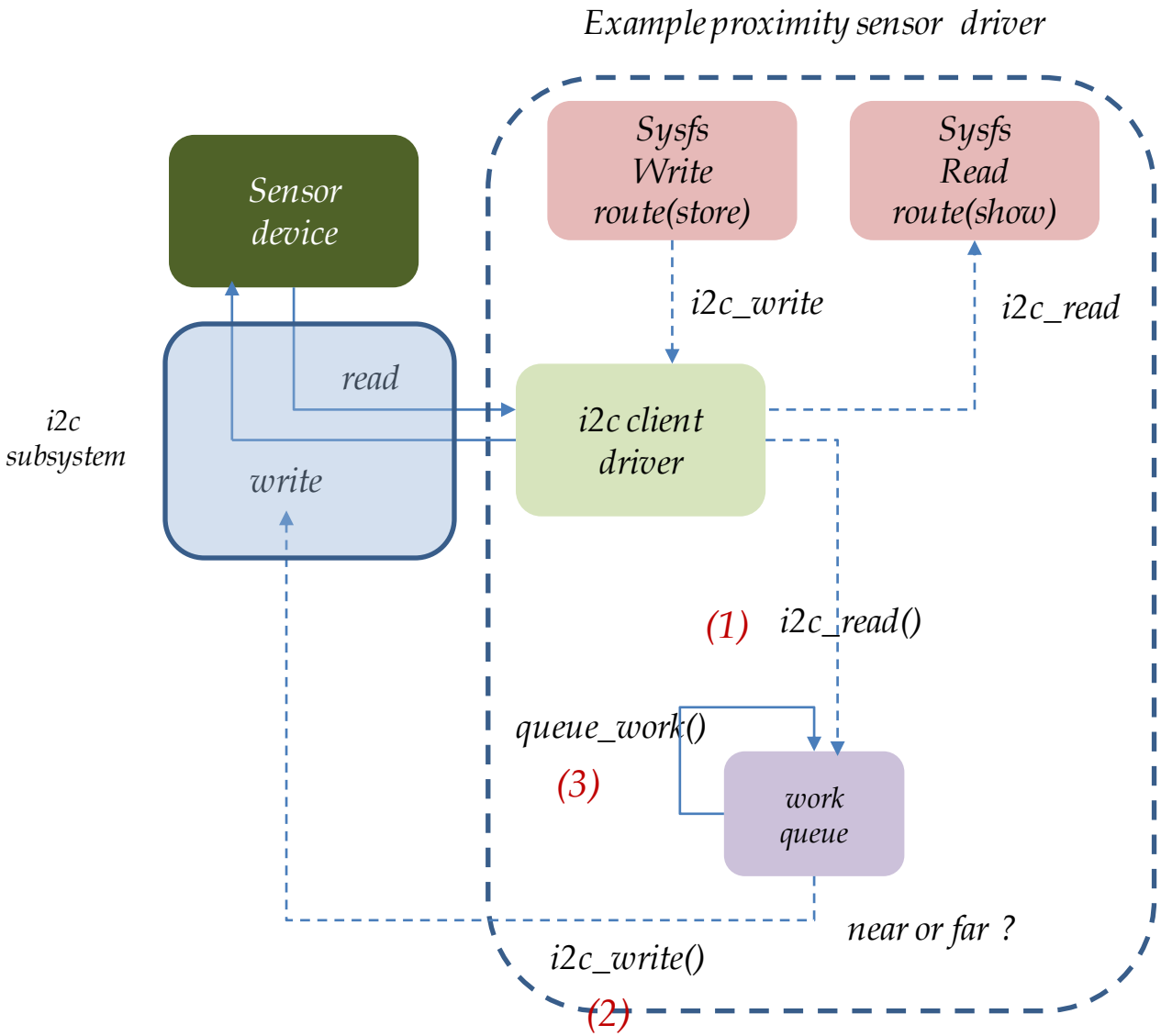
5. Example Accelerometer Sensor Driver



6. Example Geomagnetic Sensor Driver

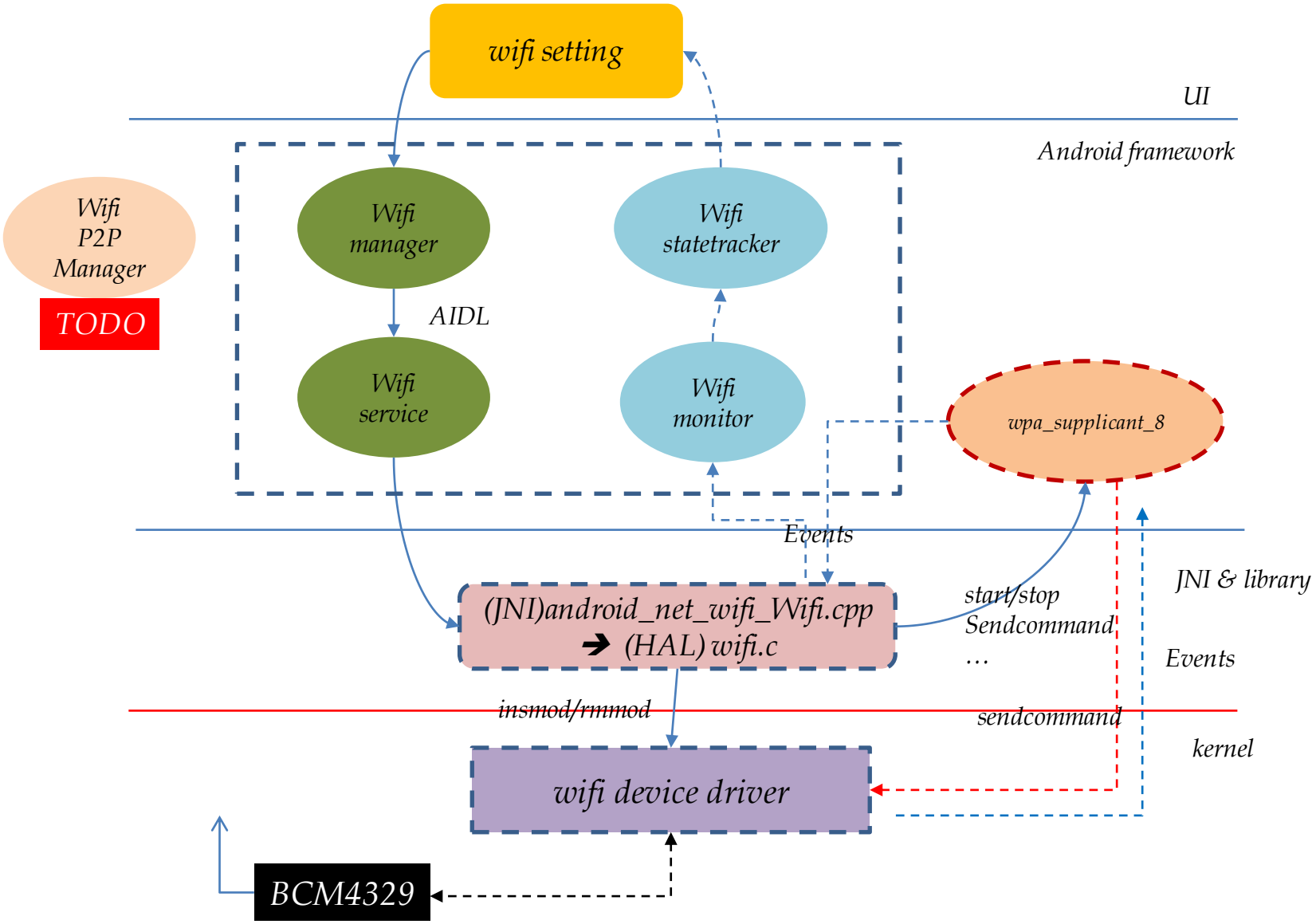


7. Example Proximity Sensor Driver: ***TODO***

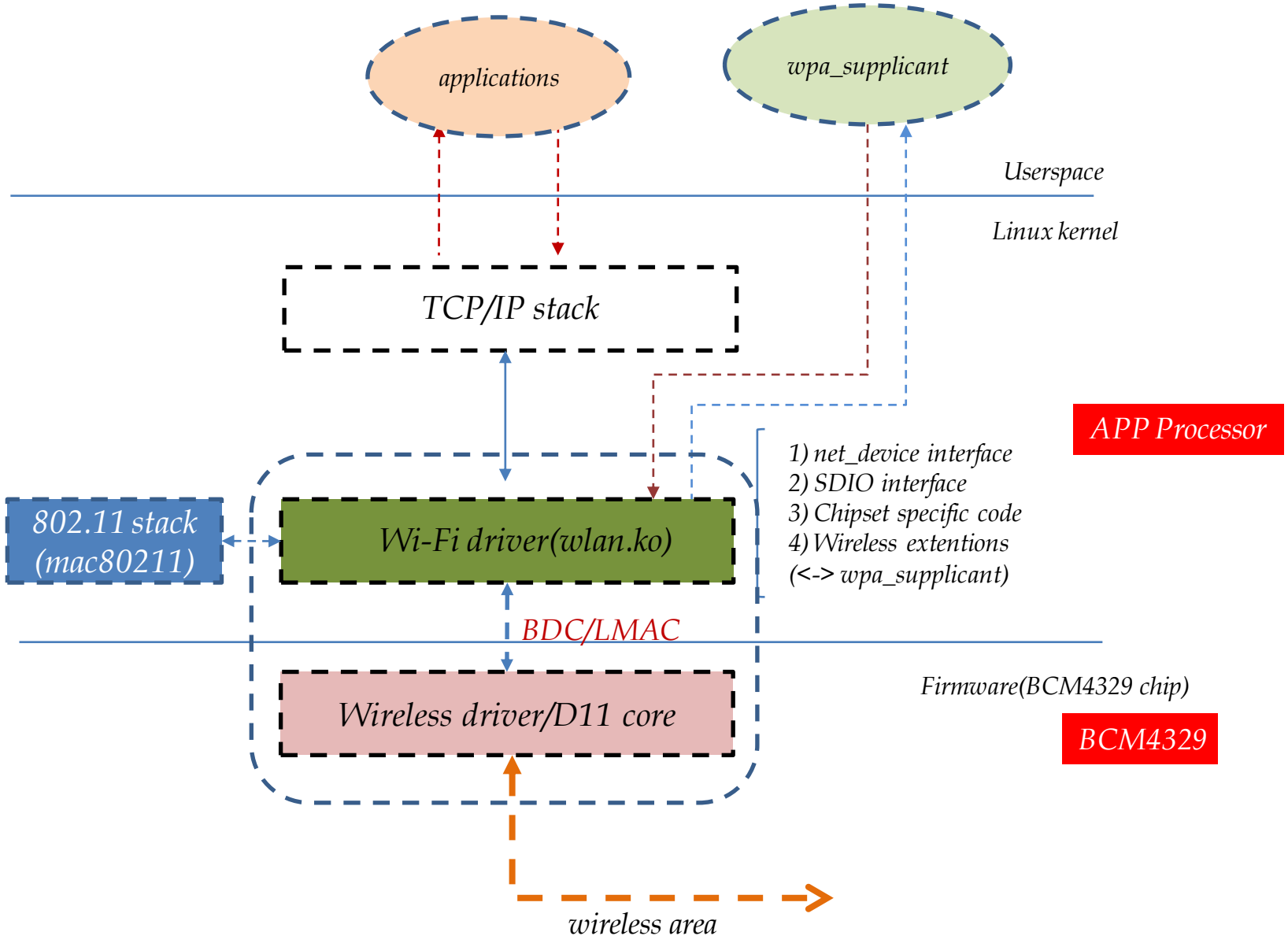


5. *Wi-Fi Driver*

1. Android Wi-Fi Overview

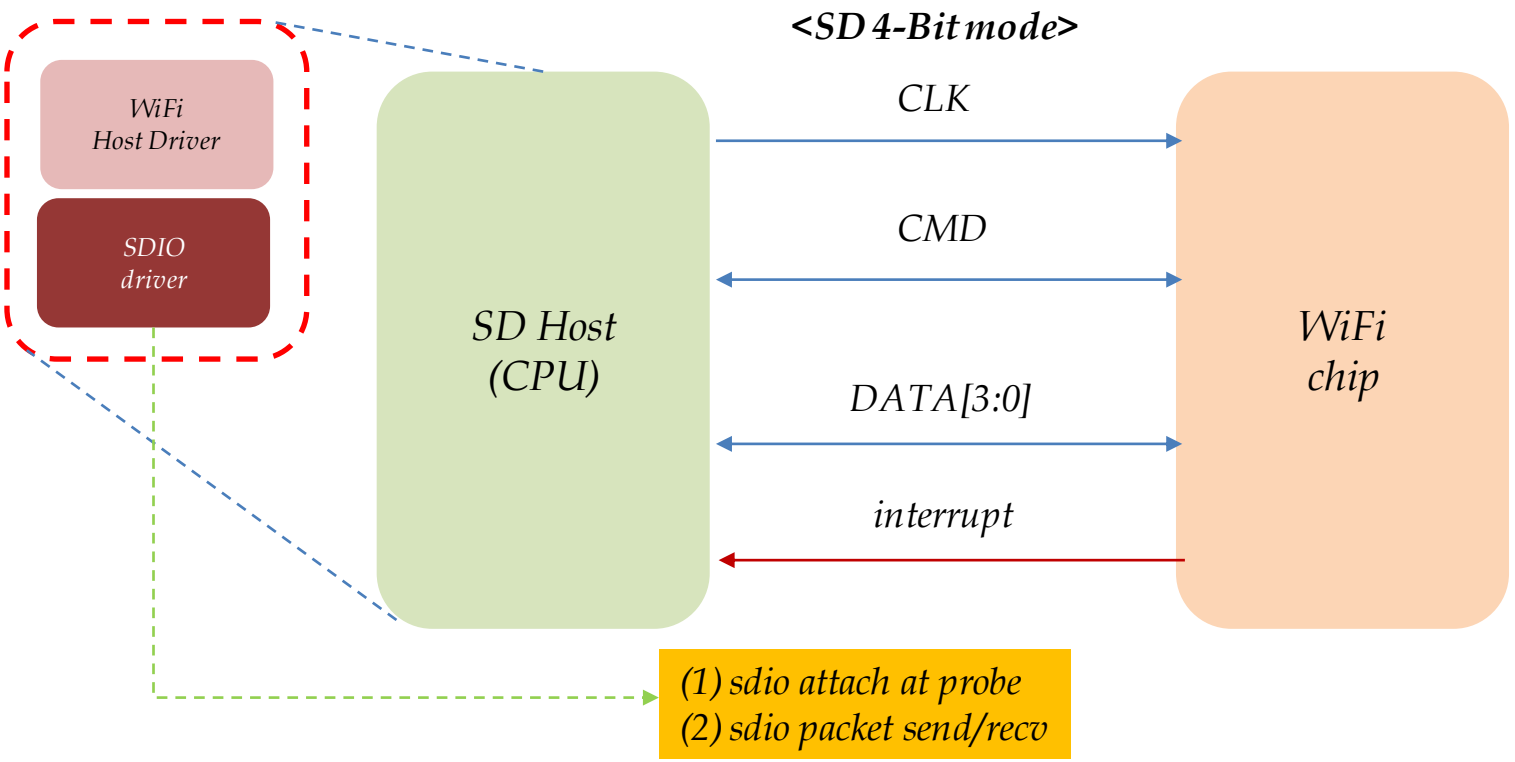


2. Wi-Fi 드라이버 구조(1)

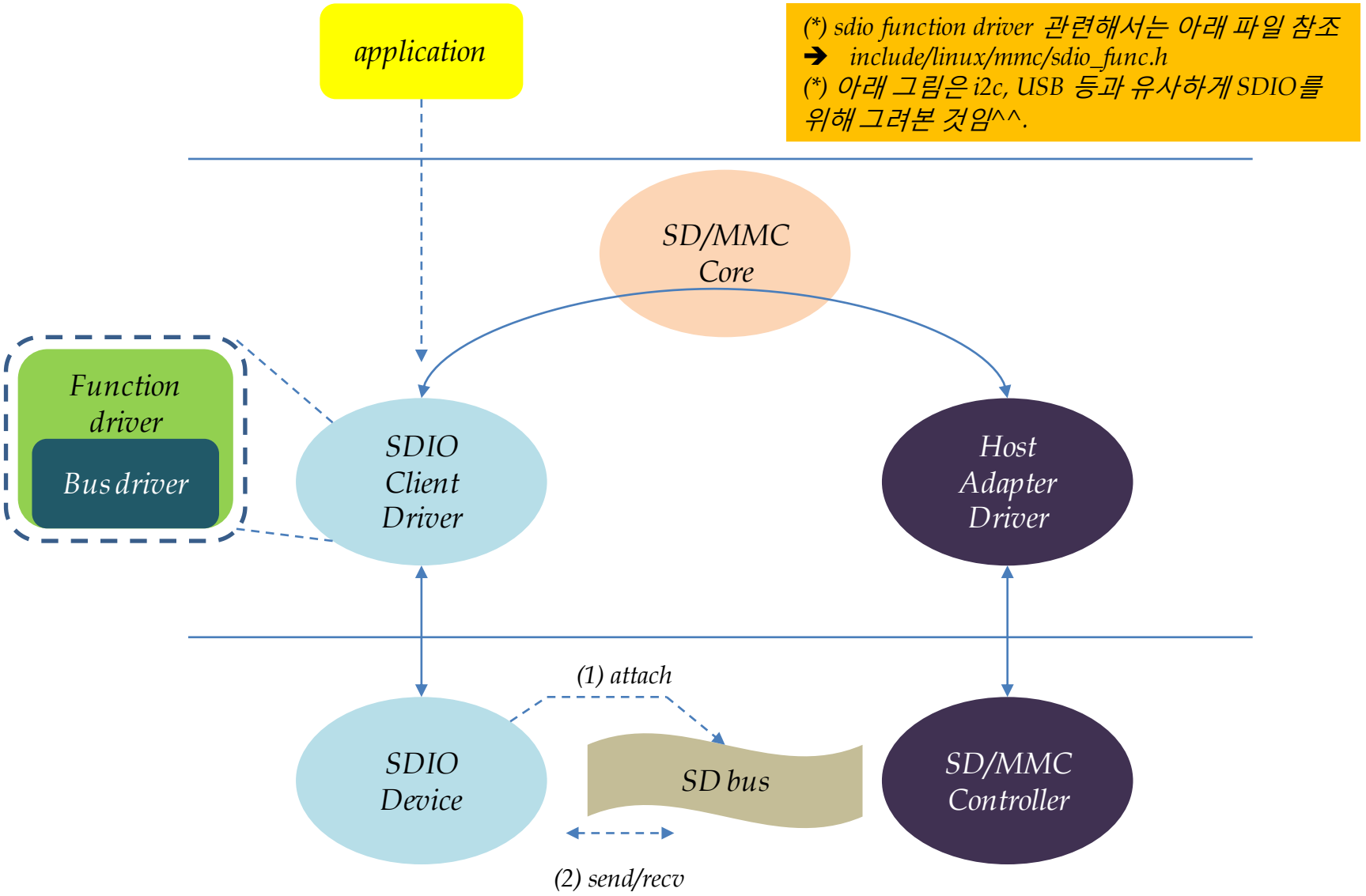


2. Wi-Fi 드라이버 구조(2)

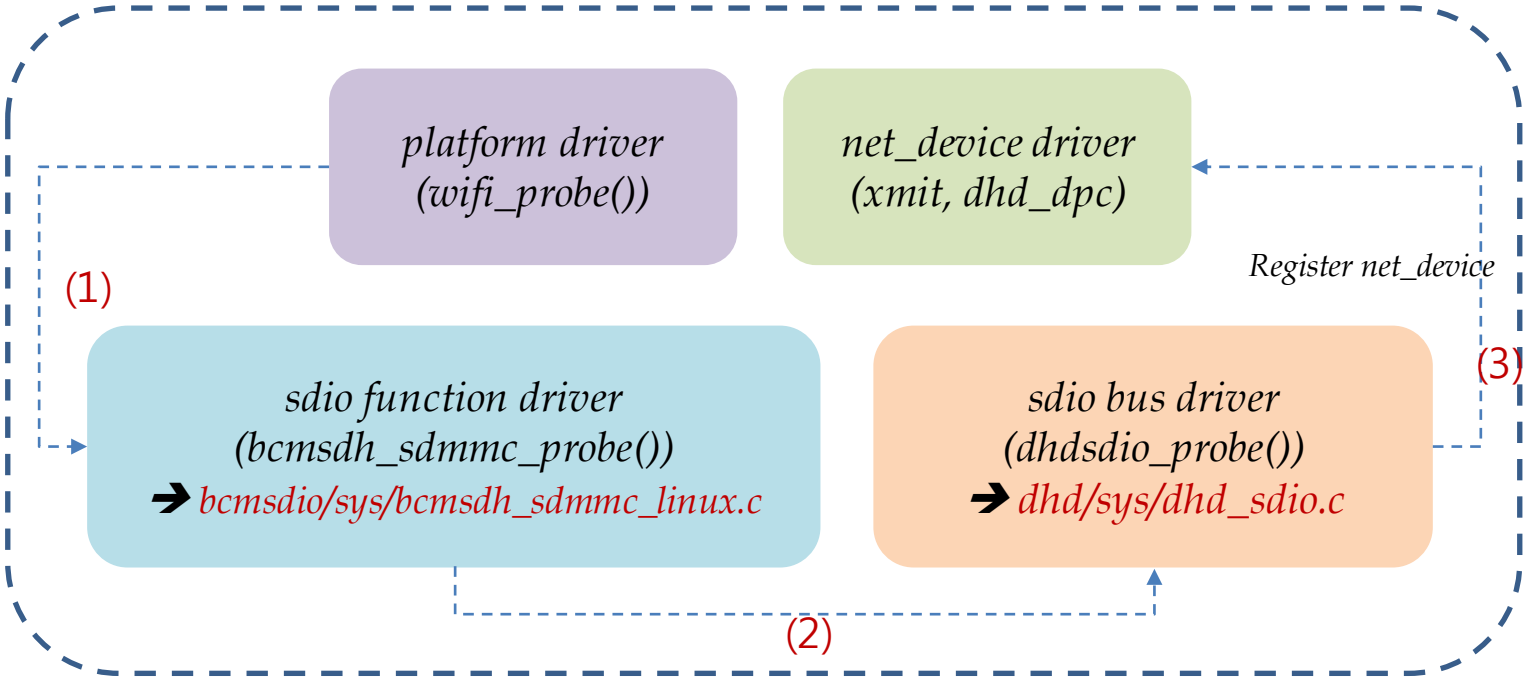
(*) 아래 그림은 WiFi 칩셋과 CPU 간의 SDIO 인터페이스(SD 4-bit mode) 사용 예를 보여준다.



2. Wi-Fi 드라이버 구조(3): ***SDIO Client Driver***

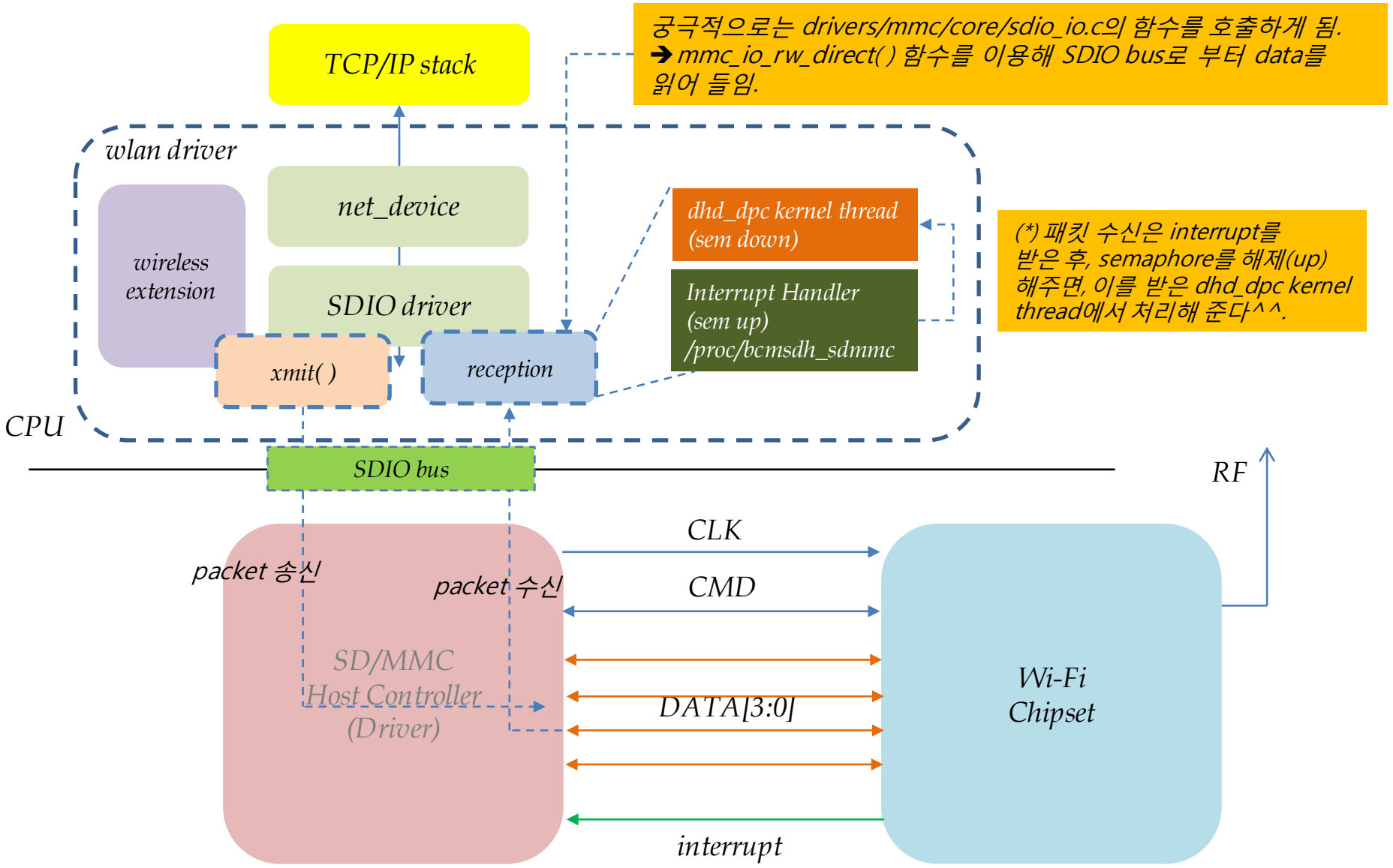


2. Wi-Fi 드라이버 구조(4) – ***bcm4329 driver overview(1)***

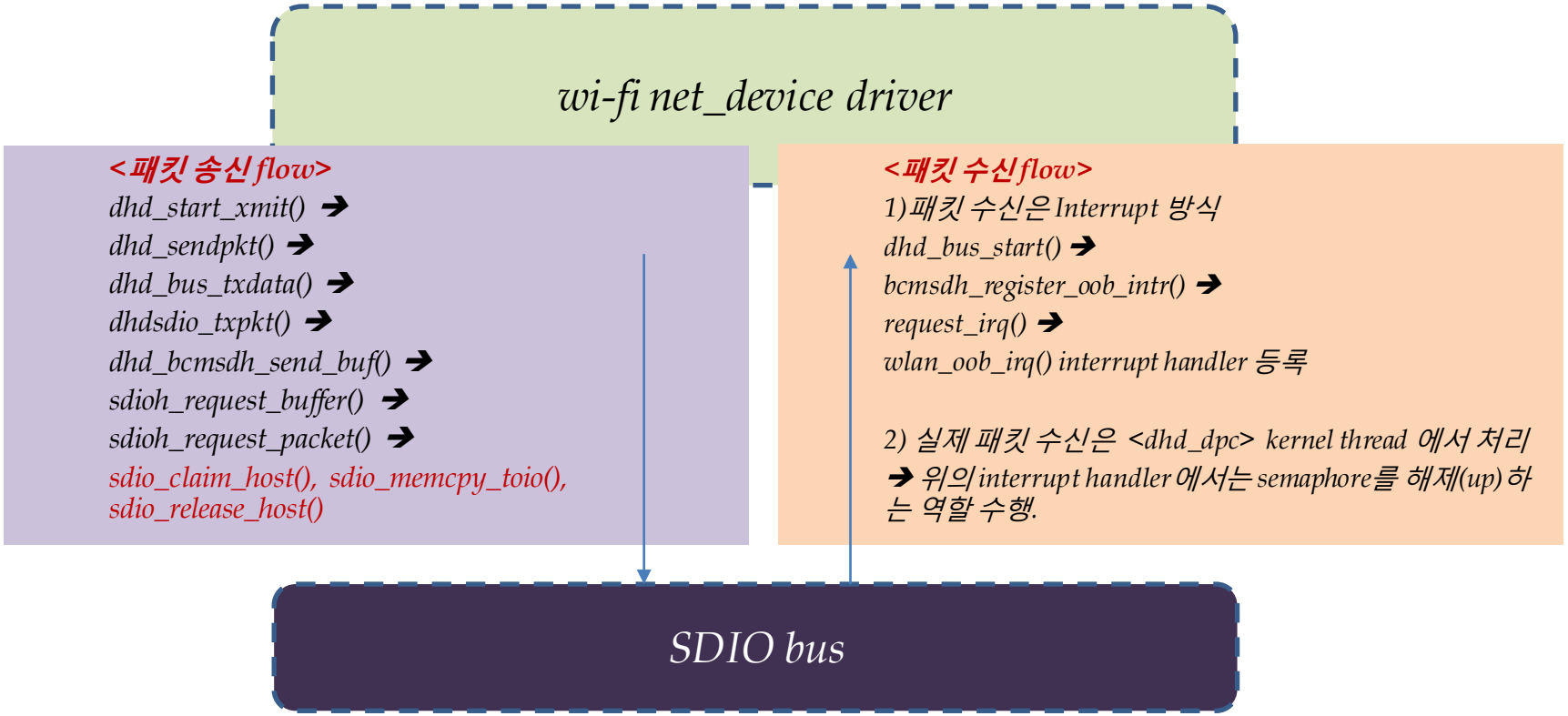


(*) BCM4329 driver는
1) Platform driver,
2) net_device(network) driver,
3) sdio function driver 이며,
4) sdio bus driver 이다.

2. Wi-Fi 드라이버 구조(4): *bcm4329 driver overview(2)*



2. Wi-Fi 드라이버 구조(4): *bcm4329 driver overview(3)*

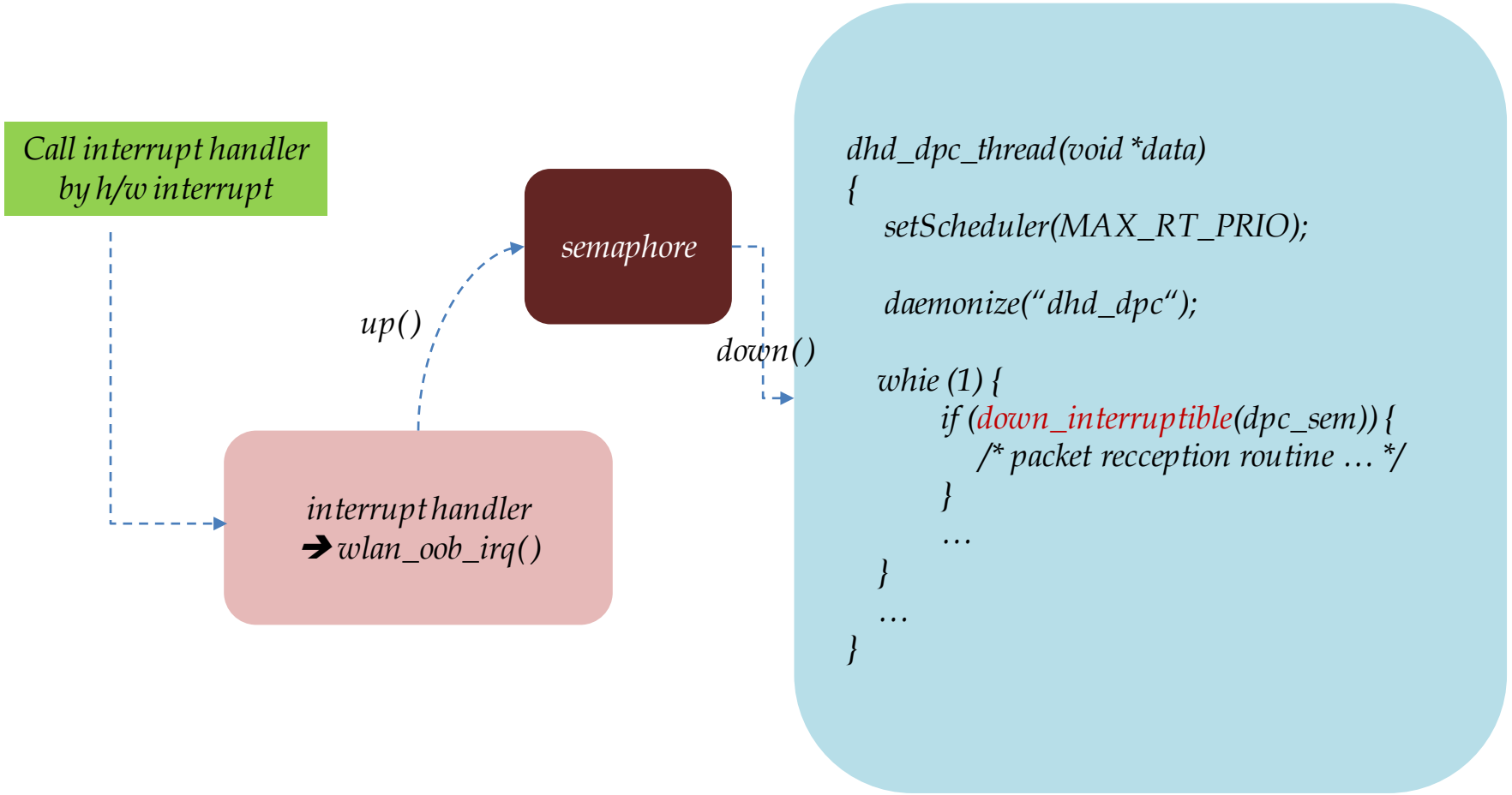


<참고 사항>

➔ SDIO bus로 부터 data를 읽거나 쓰고자 할 경우, 궁극적으로 아래와 같은 함수 호출 순서를 따르고 있다(bcmsdio/sys/bcmsdh_sdmmc.c 파일 참조)

1) sdio_claim_host( )
2) sdio_readb( ) or sdio_writeb( ) .... ← drivers/mmc/core/sdio_io.c에 있는 함수
3) sdio_release_host( )

2. Wi-Fi 드라이버 구조(4): *bcm4329 driver overview(4)*



(*) 위의 그림은 SDIO bus로 부터 들어오는 패킷을 수신하기 위한 그림으로 interrupt handler와 kernel thread 간의 관계를 표현하고 있다.

2. Wi-Fi 드라이버 구조(5) – *bcm4329 driver flow 분석(1)*

```
static struct platform_driver wifi_device = {
    .probe      = wifi_probe,
    .remove     = wifi_remove,
    .suspend    = wifi_suspend,
    .resume     = wifi_resume,
    .driver     = {
    .name  = "bcm4329_wlan",
    }
};
```

```
struct sdio_driver bcmsdh_sdmmc_driver = {
    .probe     = bcmsdh_sdmmc_probe,
    .remove    = bcmsdh_sdmmc_remove,
    .name      = "bcmsdh_sdmmc",
    .id_table  = bcmsdh_sdmmc_ids,
};
```

```
bcmsdh_driver_t dhd_sdio = {
    dhdsdio_probe,
    dhdsdio_disconnect
};
```

<module_init>

1) gpio를 이용하여 wi-fi chipset을 on한다.
2) platform_driver로 등록한다.
3) SDIO/MMC driver로 등록한다.
 3-1) dhd_bus_register( )
 3-2) bcmsdh_register( )
 3-3) sdio_function_init( )
 3-4) sdio_register_driver( ) <- drivers/mmc/core/sdio_bus.c
  ➔ sdio function driver로 등록함.
  ➔ bcmsdh_sdmmc_probe( ) 함수가 이 sdio function driver의 probe 함수임.

2. Wi-Fi 드라이버 구조(5) – *bcm4329 driver flow 분석(2)*

```
struct sdio_driver
bcmsdh_sdmmc_driver = {
    .probe      = bcmsdh_sdmmc_probe,
    .remove     = bcmsdh_sdmmc_remove,
    .name       = "bcmsdh_sdmmc",
    .id_table   = bcmsdh_sdmmc_ids,
};
```

(*) bcmsdh_sdmmc_probe flow

1) bcmsdh_probe

2-1) bcmsdh_attach

2-2) drvinfo.attach (= dhdsdio_probe)

➔ dhd_bus_register() 함수 안에서 아래 함수 호출 시 argument로 dhd_sdio가 전달되고, 이 것의 probe 함수가 여기서 다시 호출 됨.

bcmsdh_register(&dhd_sdio);

3) sdioh_attach <= 2-1) bcmsdh_attach() 가 호출함.

➔ sdioh_sdmmc_osinit(), sdio_set_block_size(), sdioh_sdmmc_card_enablefuncs() 함수 등 호출하고 마무리...

(*) 이 probe 함수에서는 sdio function driver 형태로 동작하기 위한 기본 처리 작업을 진행하고, 나머지는 sdio bus driver를 등록시켜 주는 역할을 수행함.

(*) sdio function driver는 실제 SDIO 장치에 data를 read/write 하기 위한 용도로 사용되는 듯.

(*) 다음 page의 dhdsdio_probe( ) 함수 호출 준비 ...

2. Wi-Fi 드라이버 구조(5) – *bcm4329 driver flow 분석(3)*

```
bcmsdh_driver_t dhd_sdio = {
    dhdsdio_probe,
    dhdsdio_disconnect
};
```

(*) dhdsdio_probe flow *<= 이함수가 실제로 sdio bus에 attach 시키는 루틴으로 보임.*

1-1) dhd_common_init
=> firmware path 초기화

1-2) dhdsdio_probe_attach
=> dongle에 attach 시도 ???

*1-3) *dhd_attach*
=> dhd driver의 main routine 수준이군.
=> dhd/OS/network interface에 attach ???
=> dhd_info_t data structure 변수 선언 및 초기화
=> net_device 추가(dhd_add_if)
=> dhd_prot_attach() 함수 호출
=> dhd_watchdog_thread kernel thread 생성
=> dhd_dpc_thread kernel thread 생성 ……..(A)
------> frame 송수신 관련 thread로 보임.
=> _dhd_sysioc_thread kernel thread 생성
=> 몇개의 wakelock 생성
------> dhd_wake_lock, dhd_wake_lock_link_dw_event,
dhd_wake_lock_link_pno_find_event

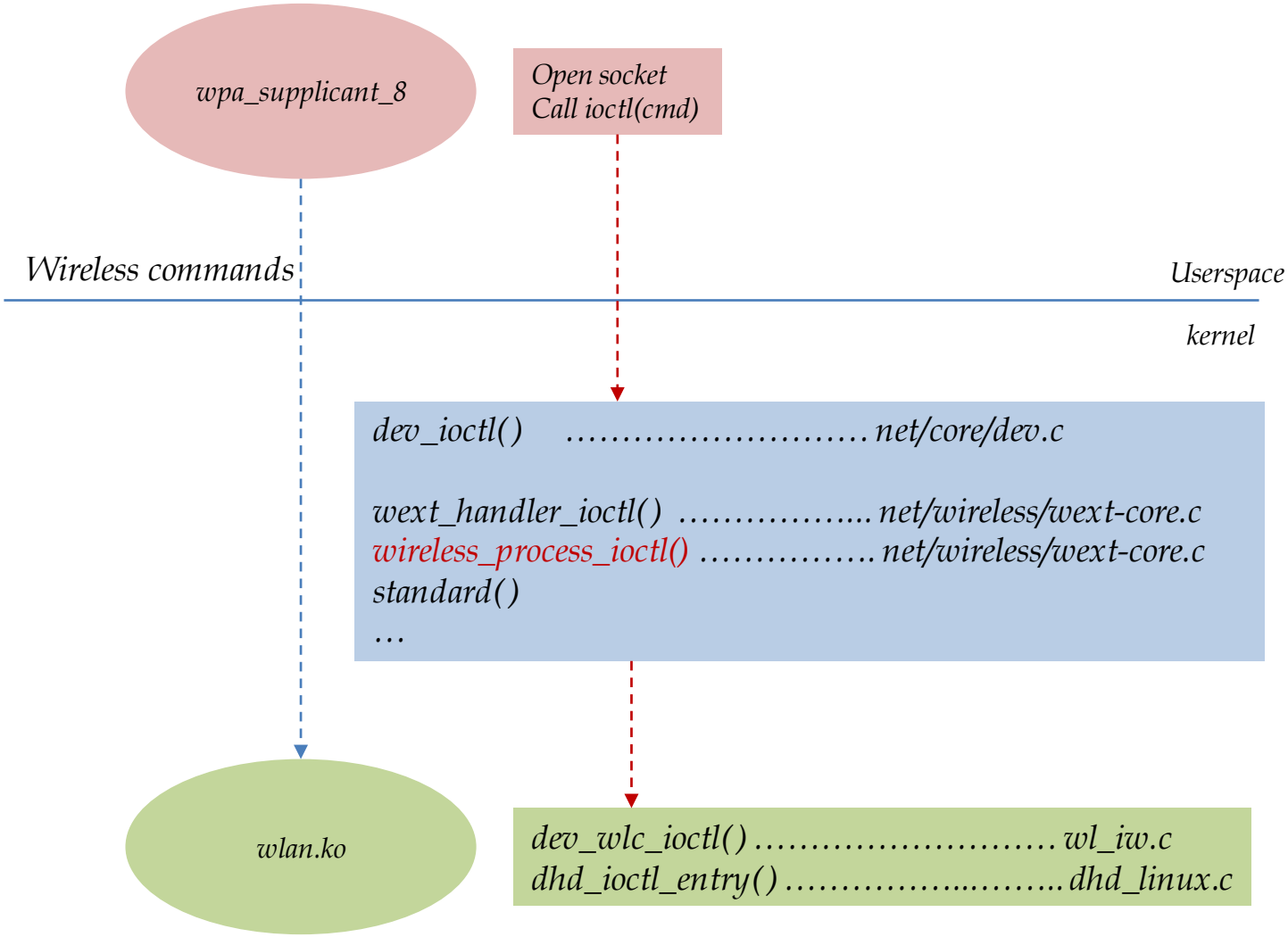
1-4) dhdsdio_probe_init

1-5) bcmsdh_intr_reg

*1-6) *dhd_bus_start*
=> firmware download 하고, bus를 start 시킨다.
=> 이 안에서 bcmsdh_register_oob_intr() 함수 호출하여
request_irq() interrupt handler 등록 ……(B)

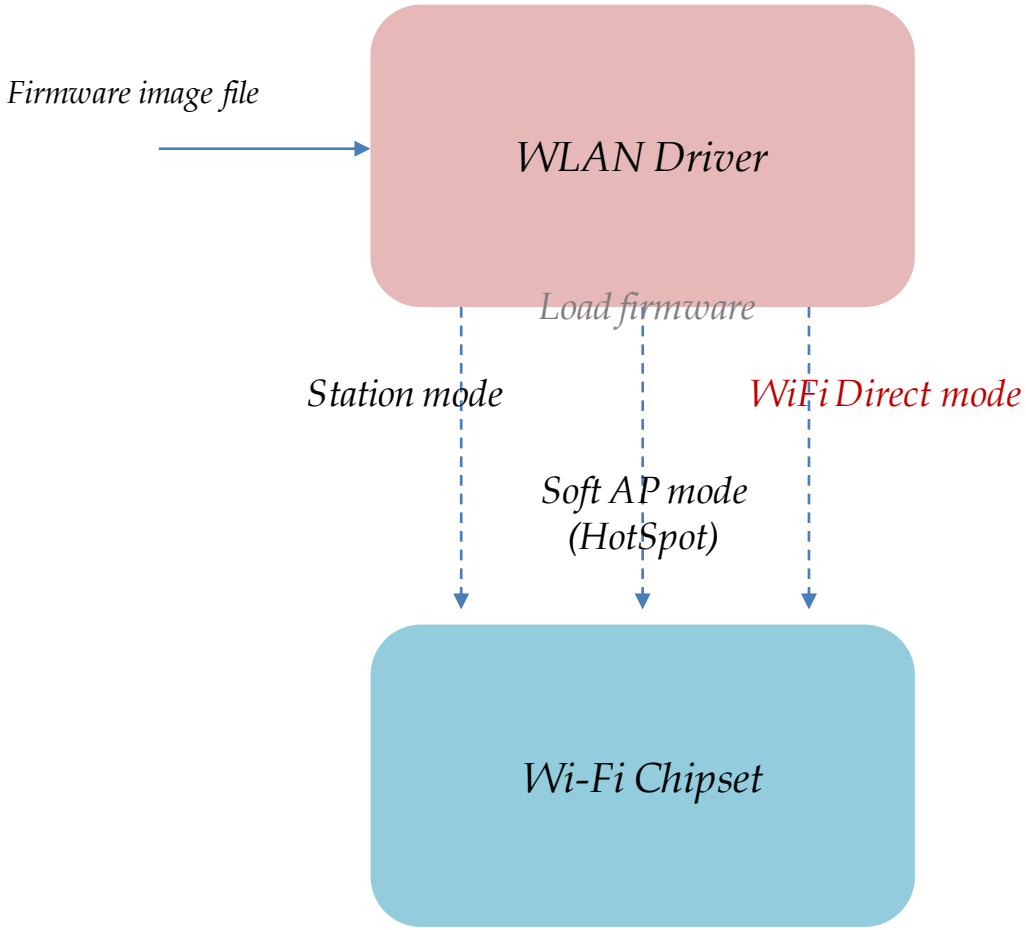
*1-7) *dhd_net_attach*
=> register_netdev() 호출하여, net_device로 등록함.

2. Wi-Fi 드라이버 구조(6): ***ioctl flow***

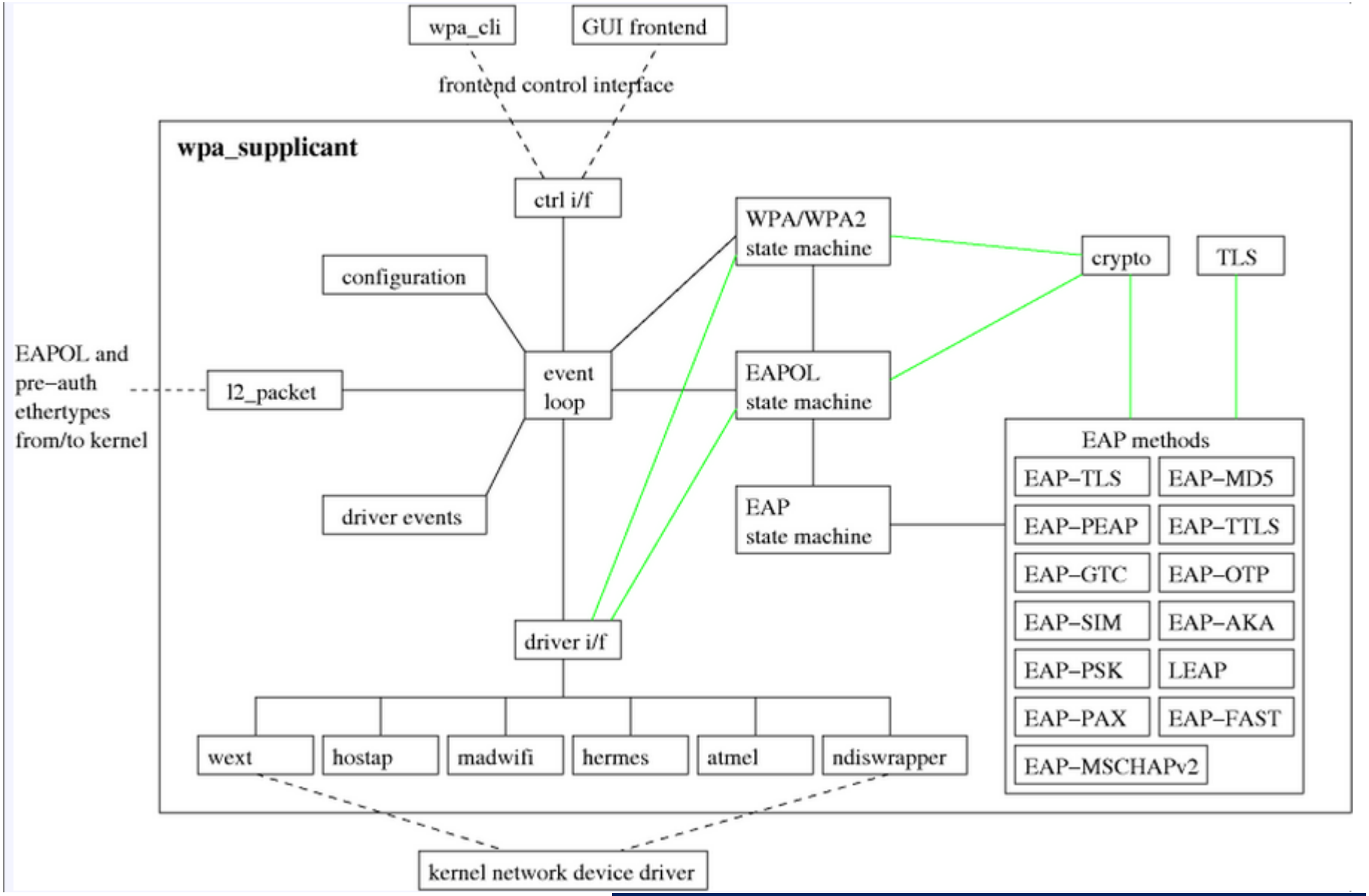


(*) 위의 과정을 통해 application으로 부터의 ioctl 명령이 wifi driver의 ioctl routine으로 전달된다.

2. Wi-Fi 드라이버 구조(7): *firmware loading*

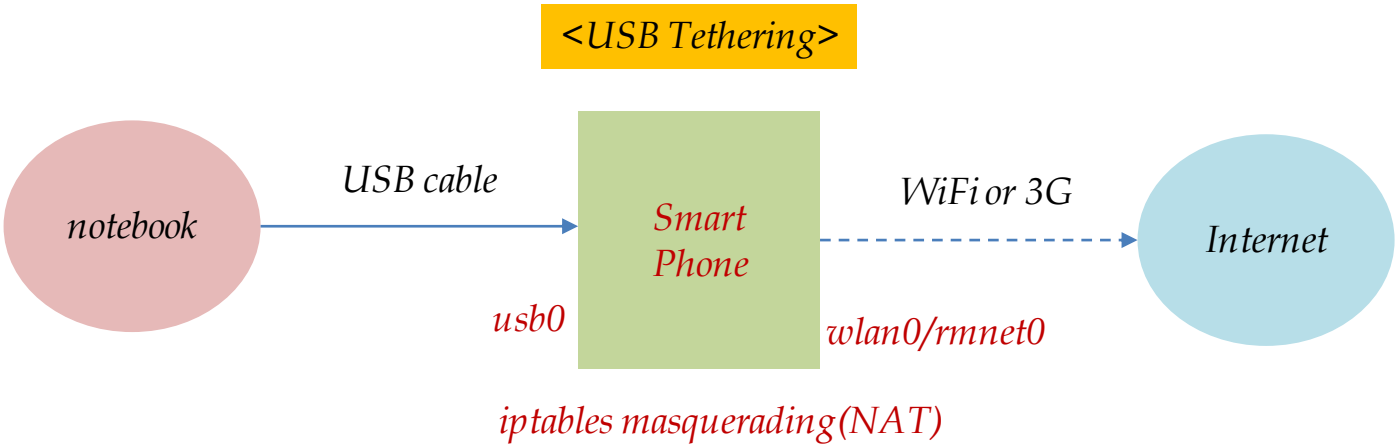


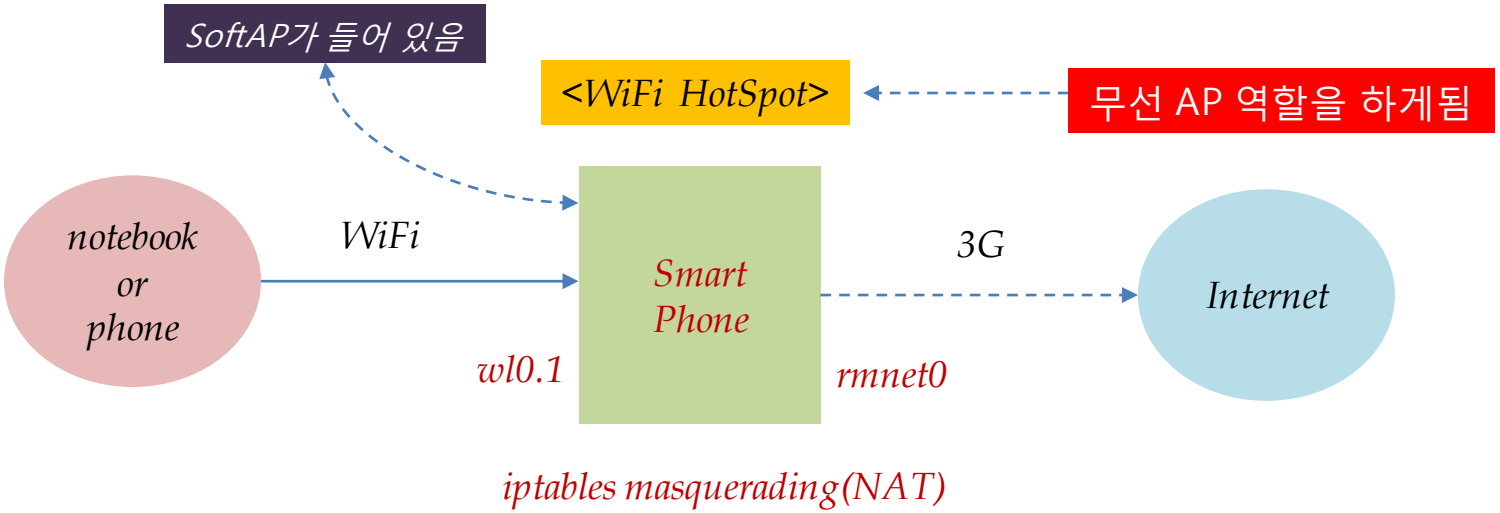
3. wpa_supplicant Architecture – *for Station(client)*



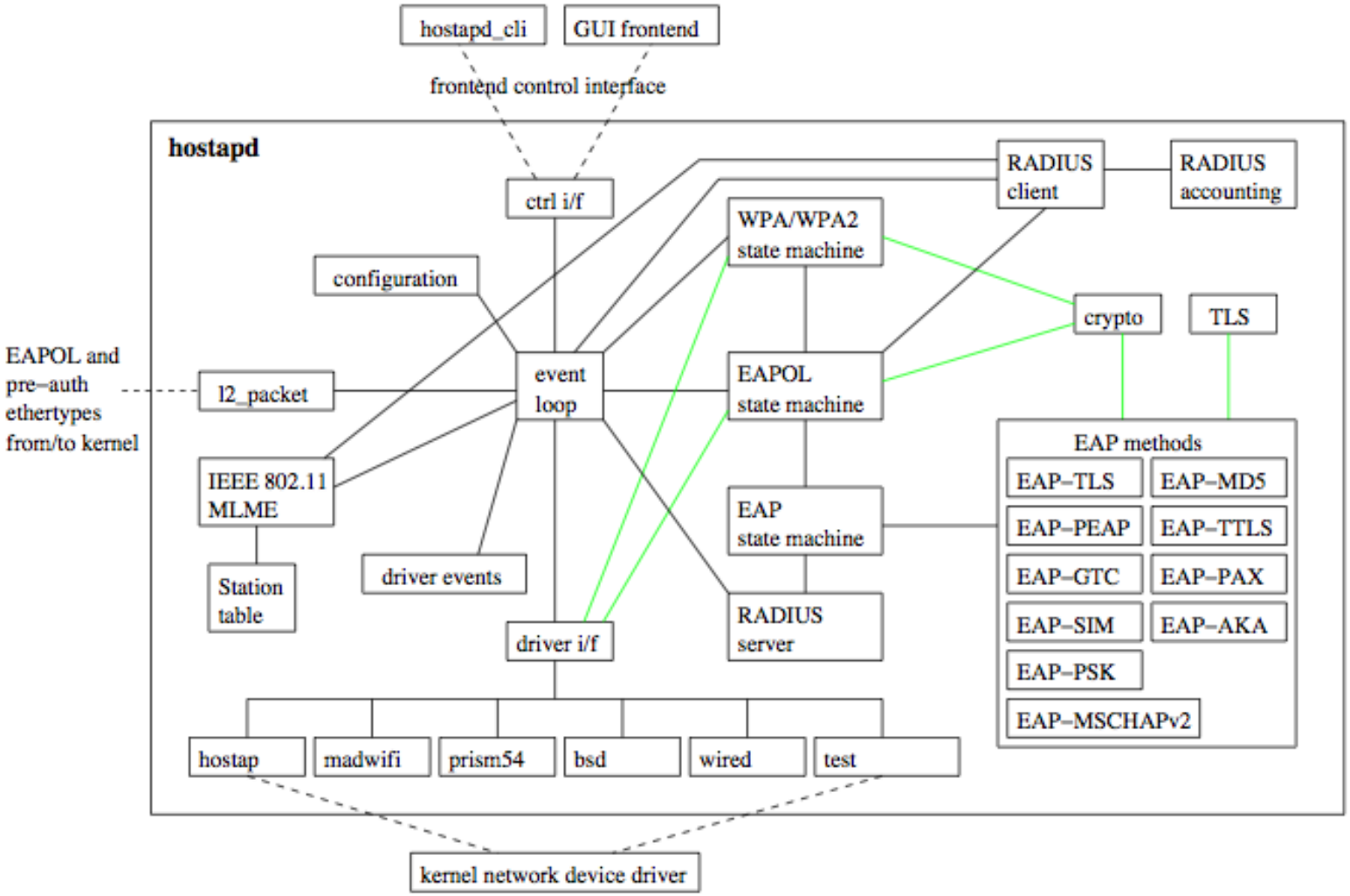
(*) 위 그림은 인터넷(wpa_supplicant 관련 site)에서 복사해 온 것임.

4. USB Tethering & WiFi HotSpot





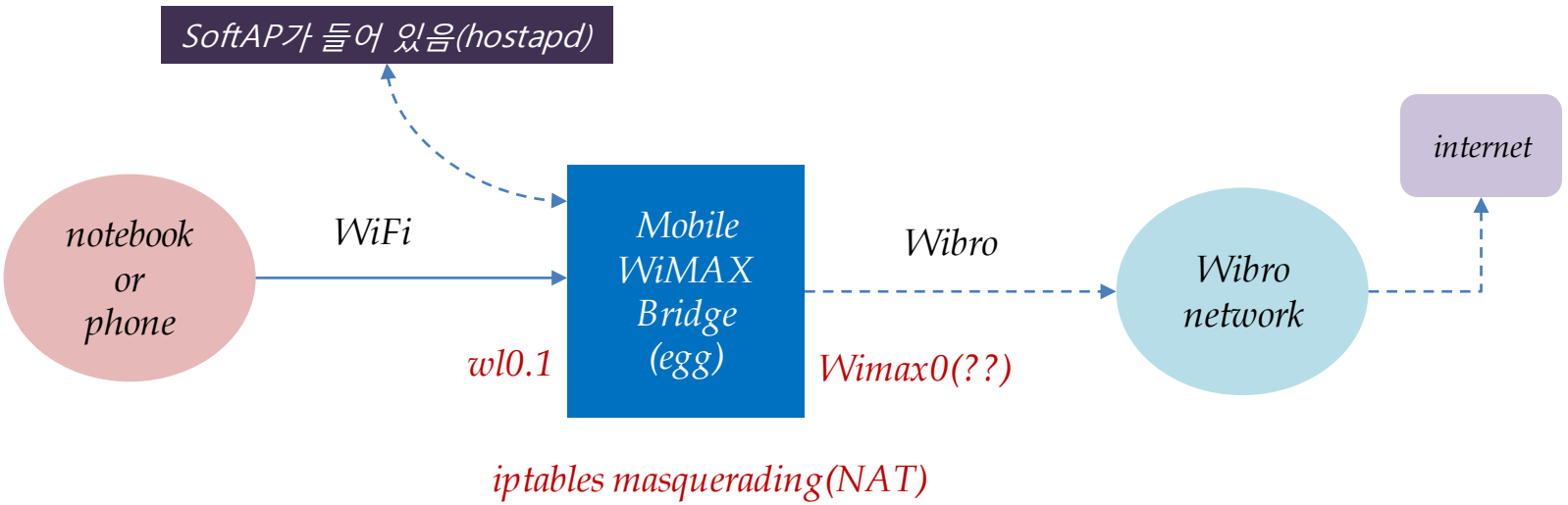
4. USB Tethering & WiFi HotSpot – *hostapd(for AP)*

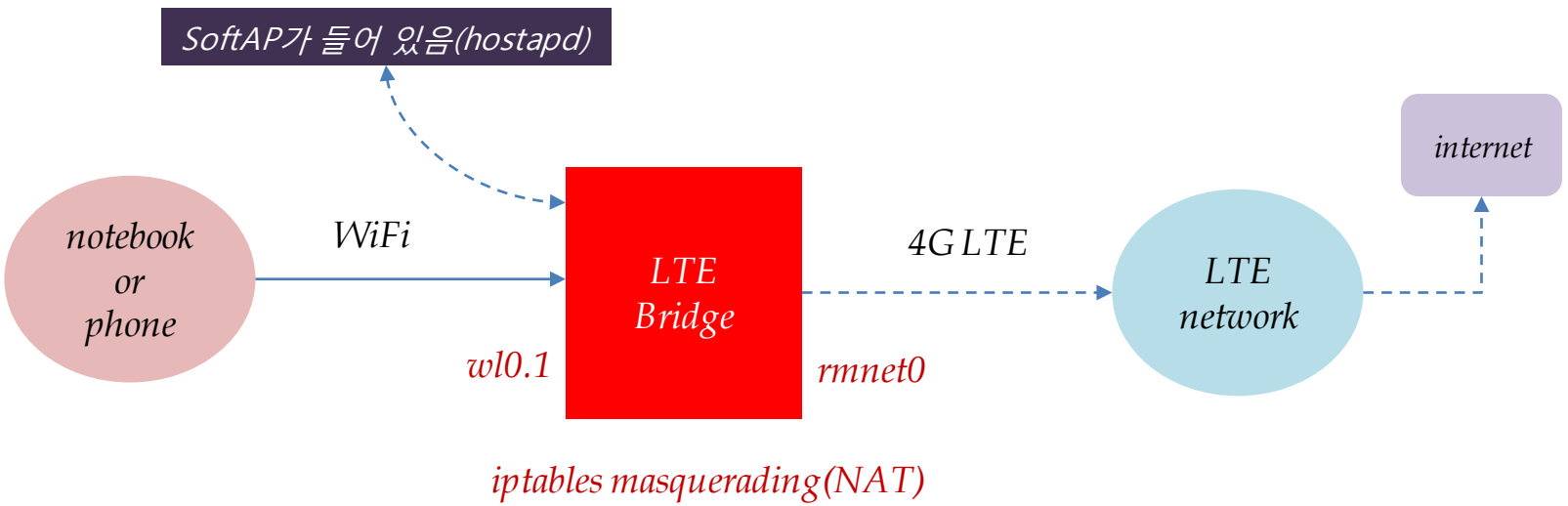


(*) HotSpot을 위해서는
1) WLAN firmware교체(AP용 firmware),
2) hostapd, dhcpcd, dnsmasq, iptables 등이 추가로 필요하다.
➔ wpa_supplicant는 station용으로 AP 환경에서는 사용되지 않음)

(*) 위 그림은 인터넷(wpa_supplicant 관련 site)에서 복사해 온 것임.

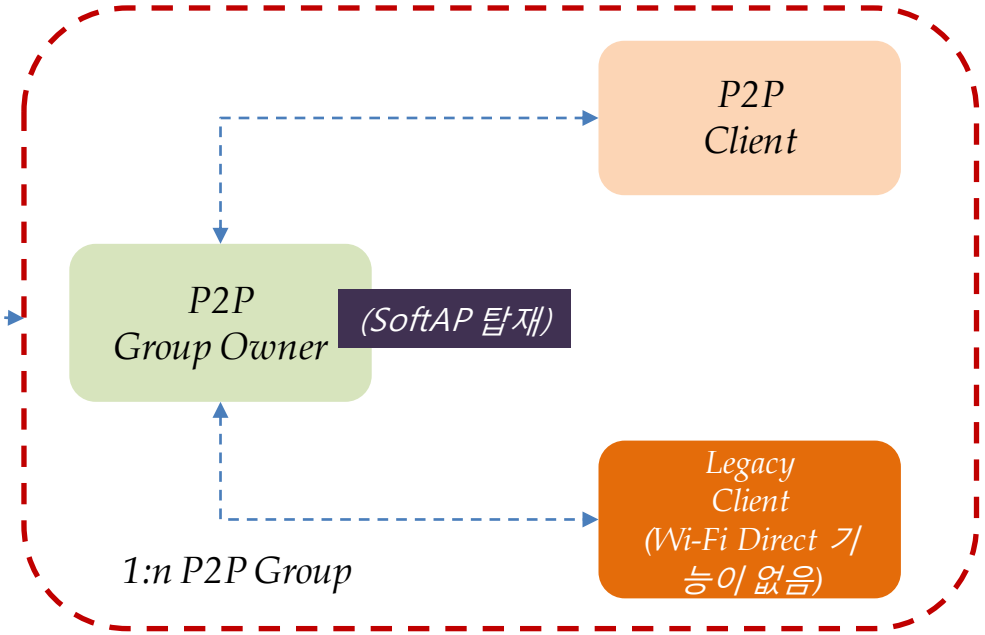
(*) Mobile WIMAX & LTE Bridge





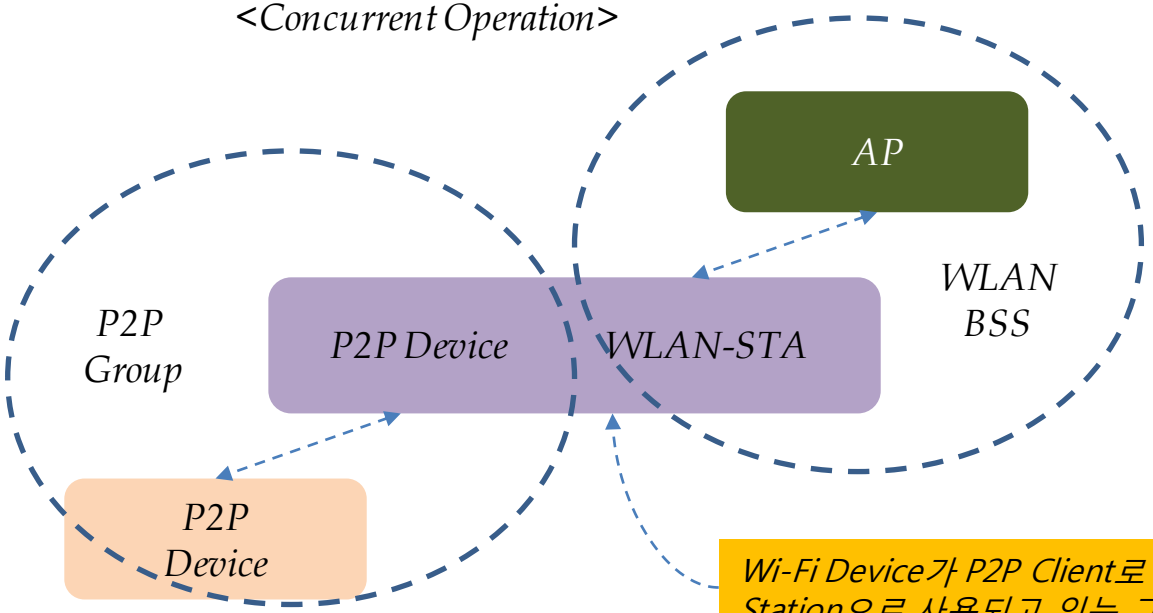
5. Wi-Fi Direct(Wi-Fi P2P)(1)

1) Wi-Fi Device가 P2P Client 혹은 P2P GO로 사용될 수 있음을 표현한 그림.
2) P2P GO로 선정되면, 무선 AP와 같은 동작을 수행해야만 함. 따라서 Legacy Client와 P2P Client가 P2P GO를 통해 상호 통신이 가능함.
3) 두 대의 P2P 디바이스가 통신할 경우, 둘 중 하나는 P2P GO로 동작함.



<P2P and Topology>

<Concurrent Operation>



Wi-Fi Device가 P2P Client로 사용되면서, 동시에 무선 AP에 접속하는 Station으로 사용되고 있는 그림임.

5. Wi-Fi Direct(Wi-Fi P2P)(2)

- **<P2P 기능>**
- *1) P2P Discovery*
- *➔ 디바이스 발견(device discovery), 서비스 발견(service discovery), 그룹 형성 (group formation), P2P 초대(P2P invitation) 등 담당*

- *2) P2P Group Operation*
- *➔ P2P Group 형성과 종료, P2P 그룹으로의 연결, P2P 그룹 내의 통신, P2P client 발견을 위한 서비스, 지속적 P2P 그룹의 동작 규정*

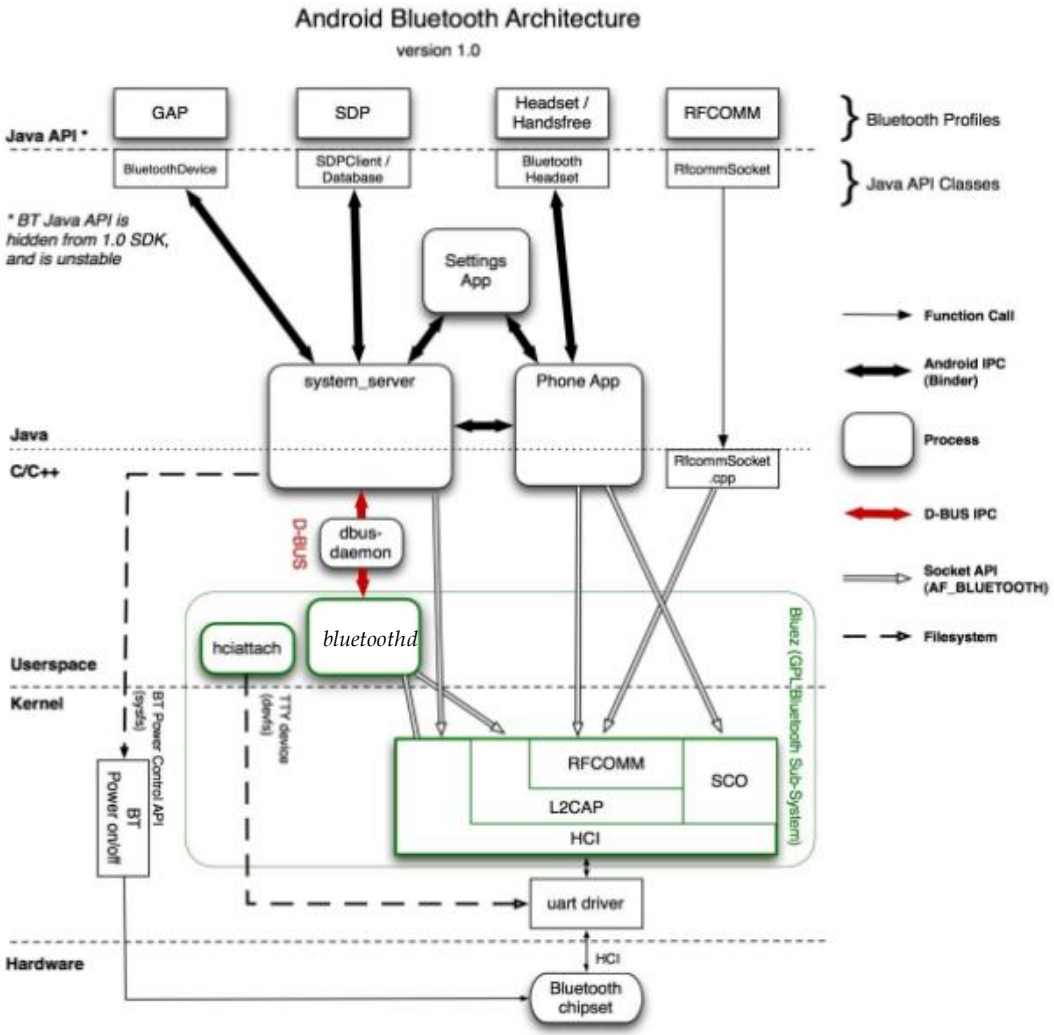
- *3) P2P Power Management*
- *➔ P2P 디바이스의 전력 관리 방법과 절전 모드 시점의 신호 처리 방법 규정*
- *(P2P GO sleep mode 진입 관련)*

- *4) Managed P2P Device*
- *➔ 한 개의 P2P 디바이스에서 P2 그룹을 형성하고, 동시에 WLAN AP에 접속하는 방법 규정*

6. Bluetooth Driver

1. Android Bluetooth Architecture

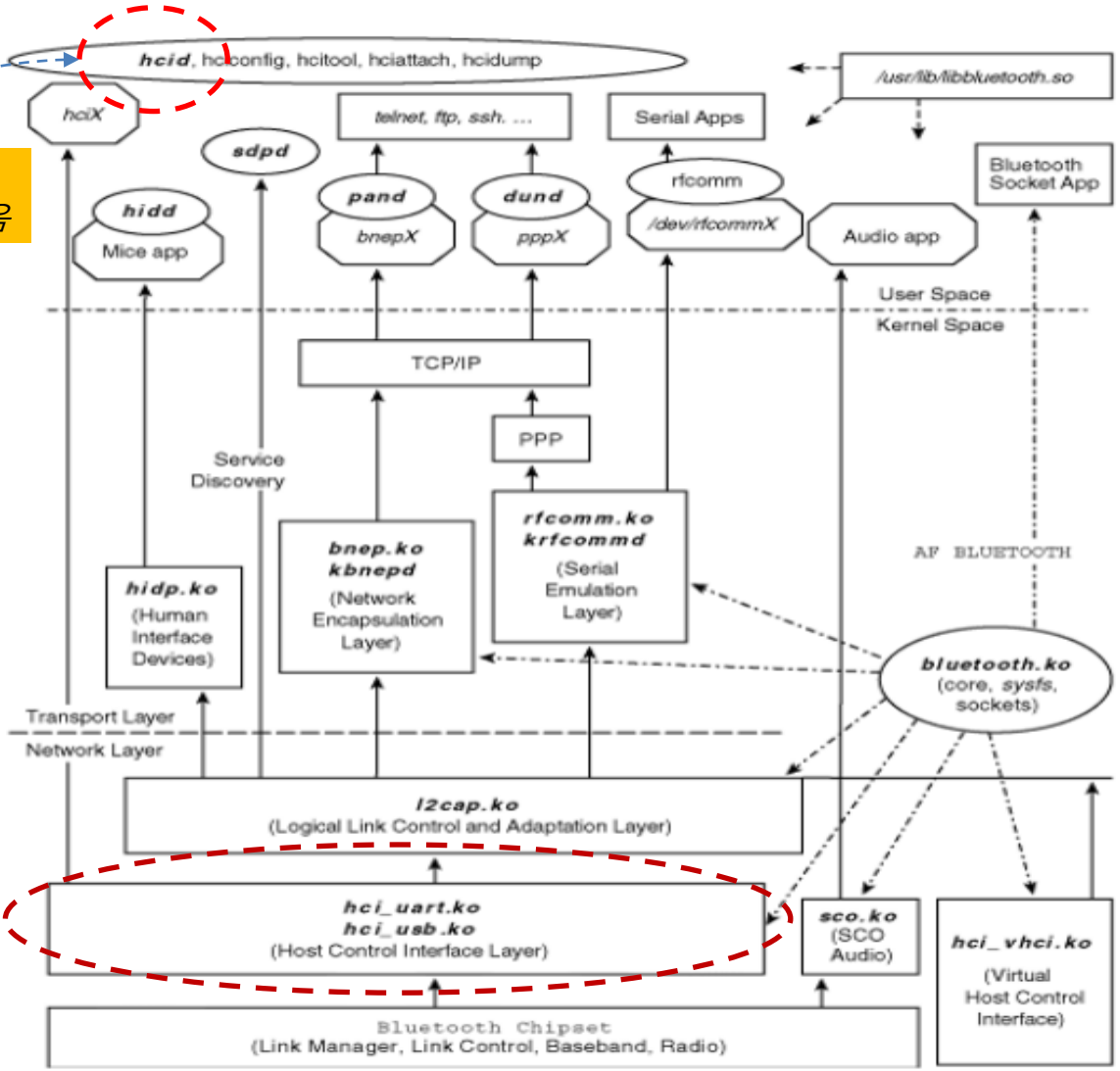
(*) 아래 그림은 인터넷에서 복사해 온 것임.



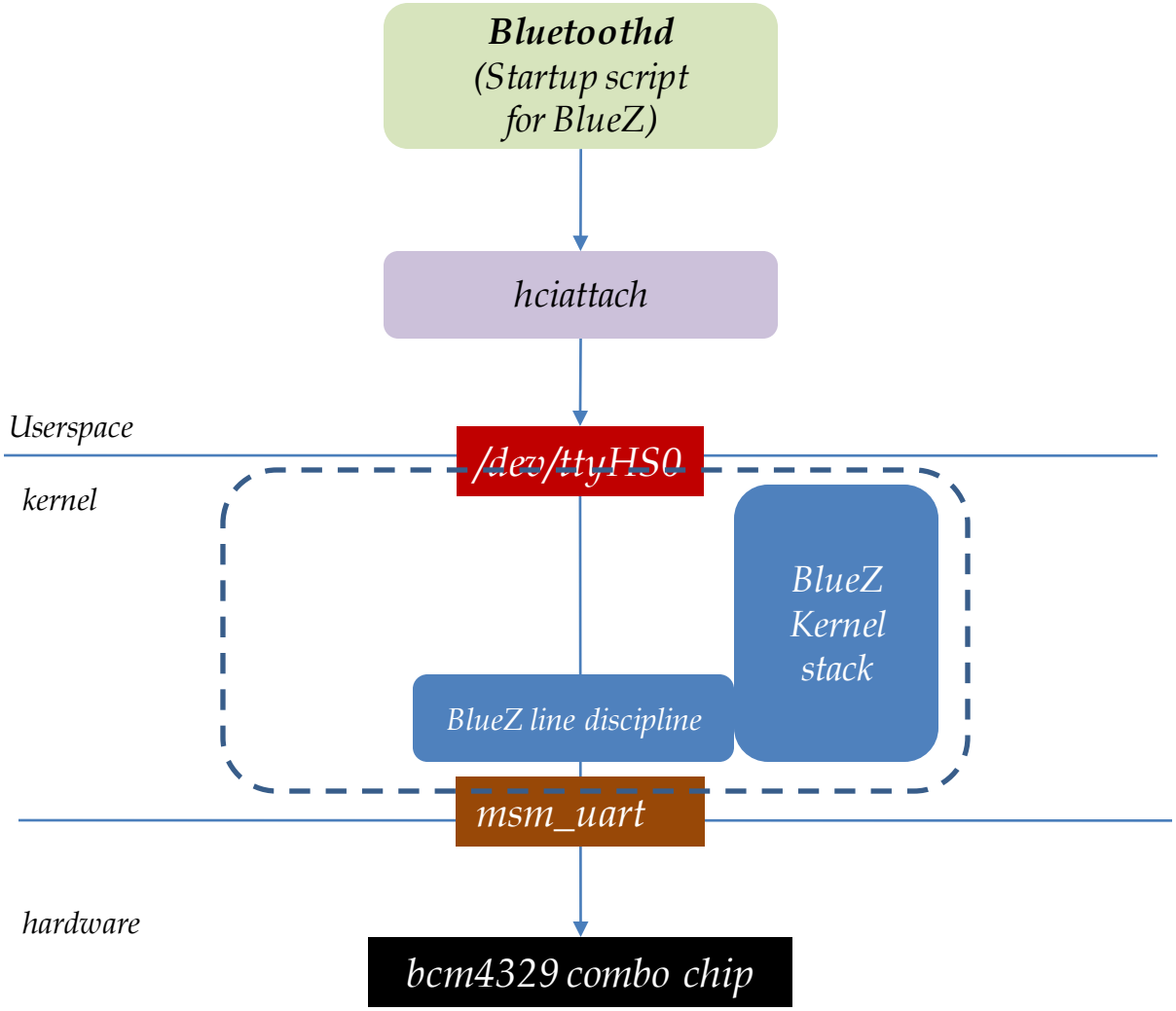
2. BlueZ Linux protocol stack

(*) 아래 그림은 참고 문서[1]에서 복사해 온 것임.

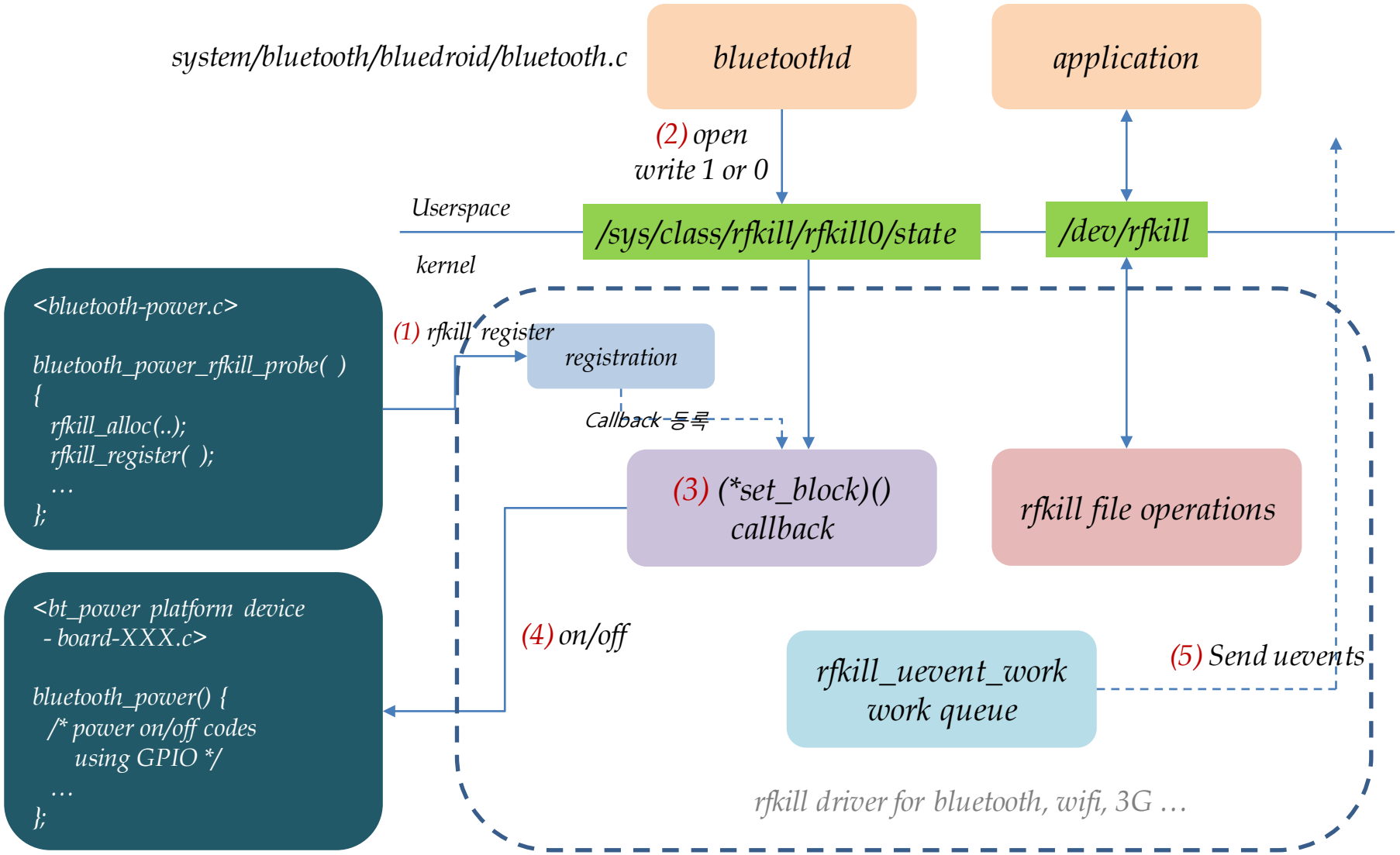
최신 버전에서는 bluetoothd로 바뀌었음



3. Bluetooth 초기화



4. Bluetooth Power On/Off: *rfkill driver(1)*



4. Bluetooth Power On/Off: *rfkill driver(2)*

<rfkill driver 개요 및 flow>

0) Wi-Fi, Bluetooth, 3G 등 전파 송수신 장치들에 대해 질의하고, 활성화하고, 비활성화할 수 있도록 해주는 interface를 제공해주는 kernel subsystem을 rfkill 드라이버라고 함.
➔ net/rfkill 디렉토리에 코드 있음.

1) rfkill을 사용하고자 하는 드라이버는 driver probe 단계에서 자신을 rfkill driver로 등록해 주어야 한다.
➔ 등록 단계에서 자신을 power on 혹은 off할 수 있는 callback 함수를 등록해 주어야 한함(이때 rfkill_alloc( ) 함수를 사용함).
➔ bluetooth의 경우는 board-XXX.c 파일에 bt_power platform device를 선언하면서 등록한 함수(platform_data = &bluetooth_power)를 위의 callback 함수로 사용하고 있음.

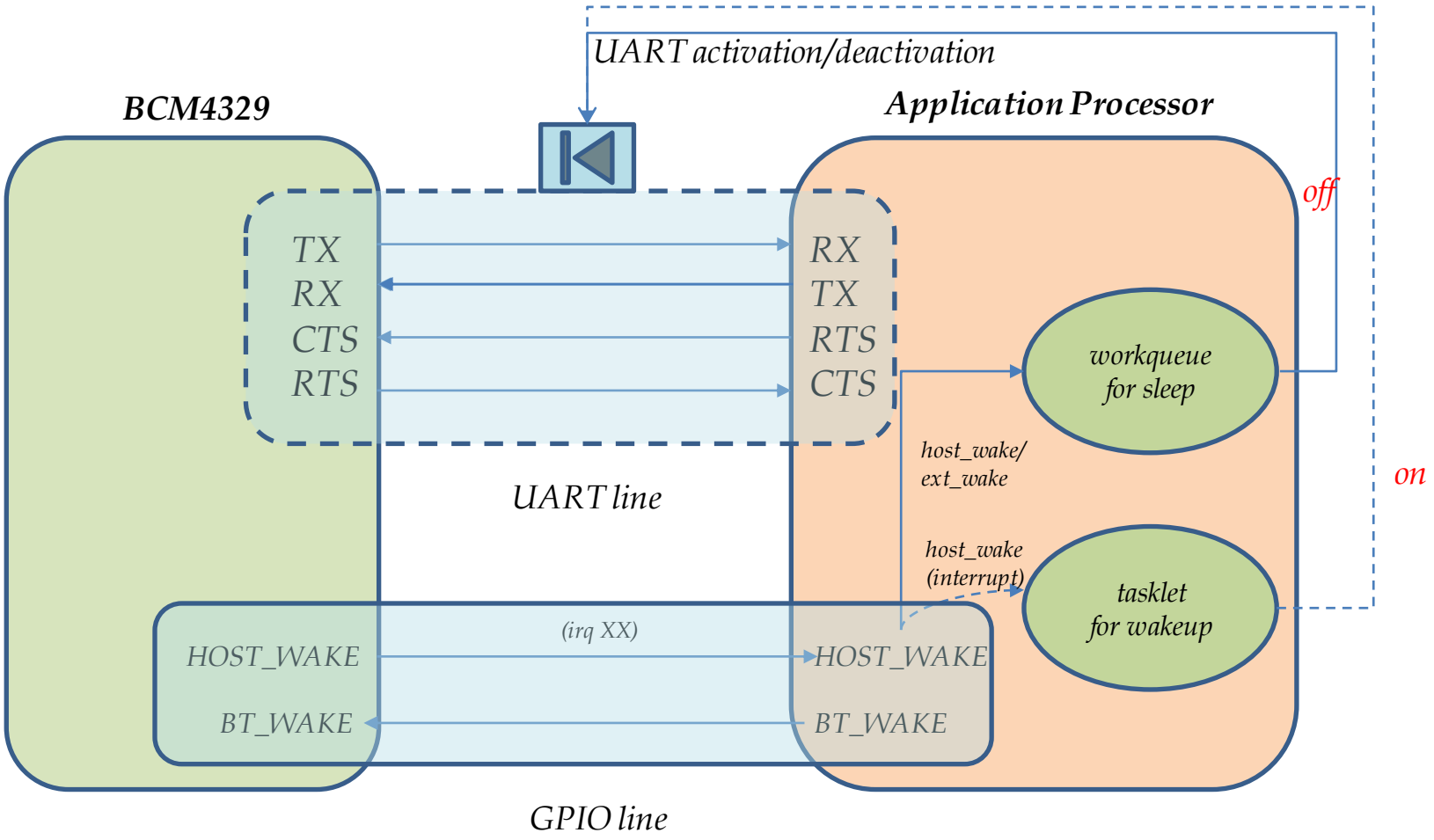
2) Application에서는 "/sys/class/rfkill/rfkillX/state" 파일을 open 한 후, power on의 경우 1을, power off의 경우 0을 write해 준다.

*3) rfill driver는 2)에서 요청한 값을 토대로, 1)에서 기 등록한 callback 함수(*set_block)를 호출해 준다.*

4) 3) 단계에 의해, 1)에서 등록한 실제 드라이버의 power control 함수가 호출되어, 실제 전파 송수신 장치(wi-fi, bluetooth, 3G ...)의 power가 on 혹은 off 된다.

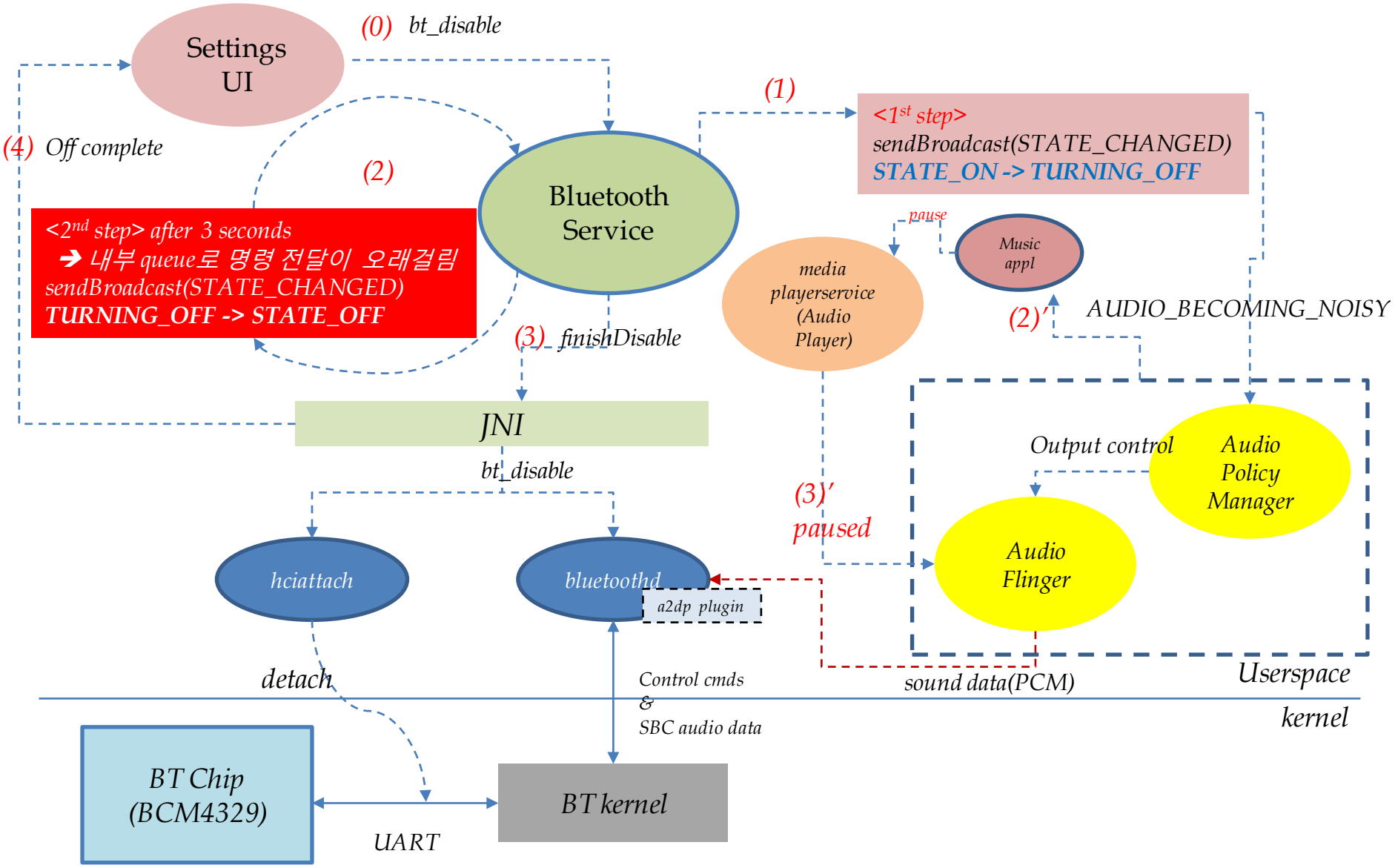
5) 상태 변화가 있을 때 마다 이를 uevent 형태로 application에 알린다.

5. Bluetooth Sleep Control



(*) bluetooth가 wakeup되는 조건은 위의 HOST_WAKE가 enable(interrupt)되는 것 이외에도 실제로 bluetooth packet이 나가고 나서 발생하는 HCI event(callback)에 기인하기도 한다.
(*) 전력 소모를 최소로 하기 위해, 틈만 나면(?) sleep mode로 진입해야 하며, HOST_WAKE 및 EXT_WAKE GPIO pin이 모두 사용중이지 않을 때(deasserted), sleep으로 들어가게 된다.

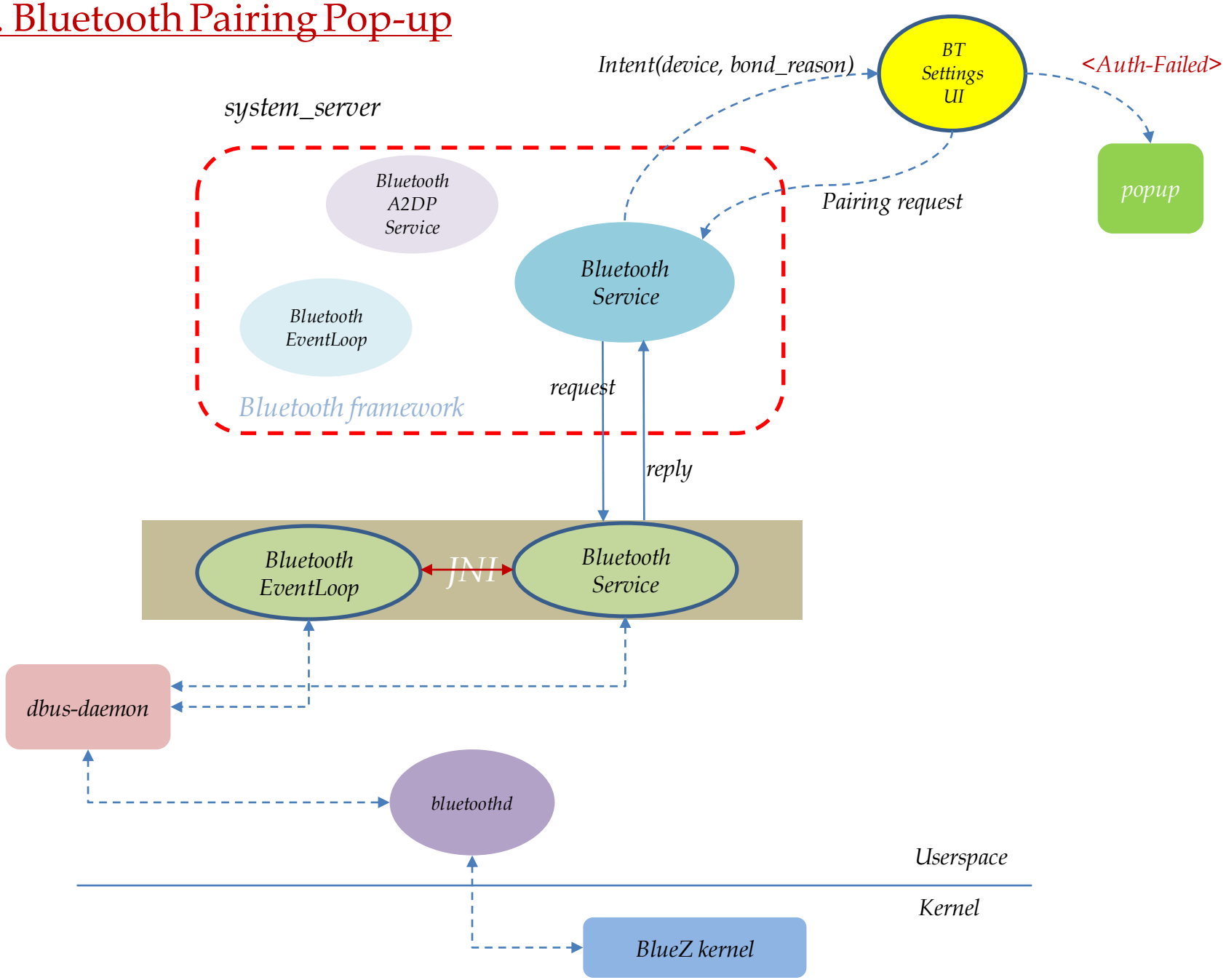
6. Bluetooth On/Off Flow



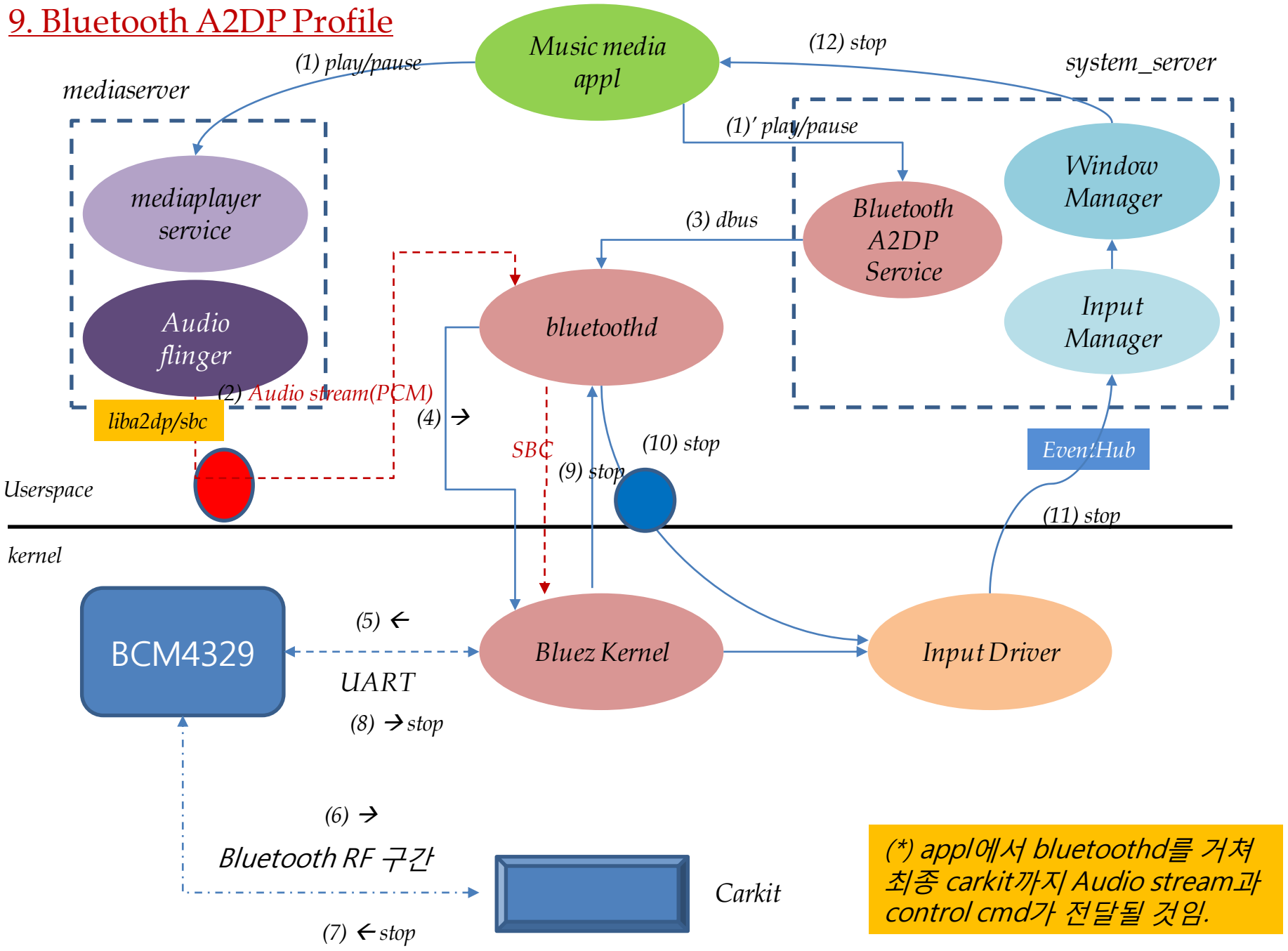
7. Bluetooth Profile

- 1) SPP/SDAP ← 기본 profile & BT 장치 검색 관련
  - SPP: Serial Port Profile(DUN, FAX, HSP, AVRCP의 기초가 되는 profile임)
  - SDAP: Service Discovery Application Profile
  - 기본 profile로 보임
- 2) HSP/HFP ← phone app/headset과 연관됨
  - HSP: Headset Profile
  - HFP: Hands-Free Profile
  - Headset을 사용하여 음악과 전화를 ...
- 3) GAVDP/AVRCP/A2DP ← media player app과 연관됨
  - GAVDP: Generic Audio/Video Distribution Profile
  - AVRCP: AV Remote Control Profile
  - A2DP: Advanced Audio Distribution Profile
  - Audio/Video playback 제어
- 4) OPP ← 파일 전송과 연관됨
  - OPP: Object Push Profile
  - File 전송 관련 ... OBEX와 연관됨
- 5) PBAP ← 전화번호부 전송 관련
  - PBAP: Phone Book Access Profile
  - Phone book object 교환(car kit <-> mobile phone)
- 6) HID ← BT 무선 키보드/마우스등 관련
  - Human Interface Device profile
  - Bluetooth keyboard/mouse/joypad ...
- 7) DUN ← BT로 인터넷 사용 관련
  - DUN: Dial-up Networking Profile
  - PC -> Phone -> Internet !!!

8. Bluetooth Pairing Pop-up

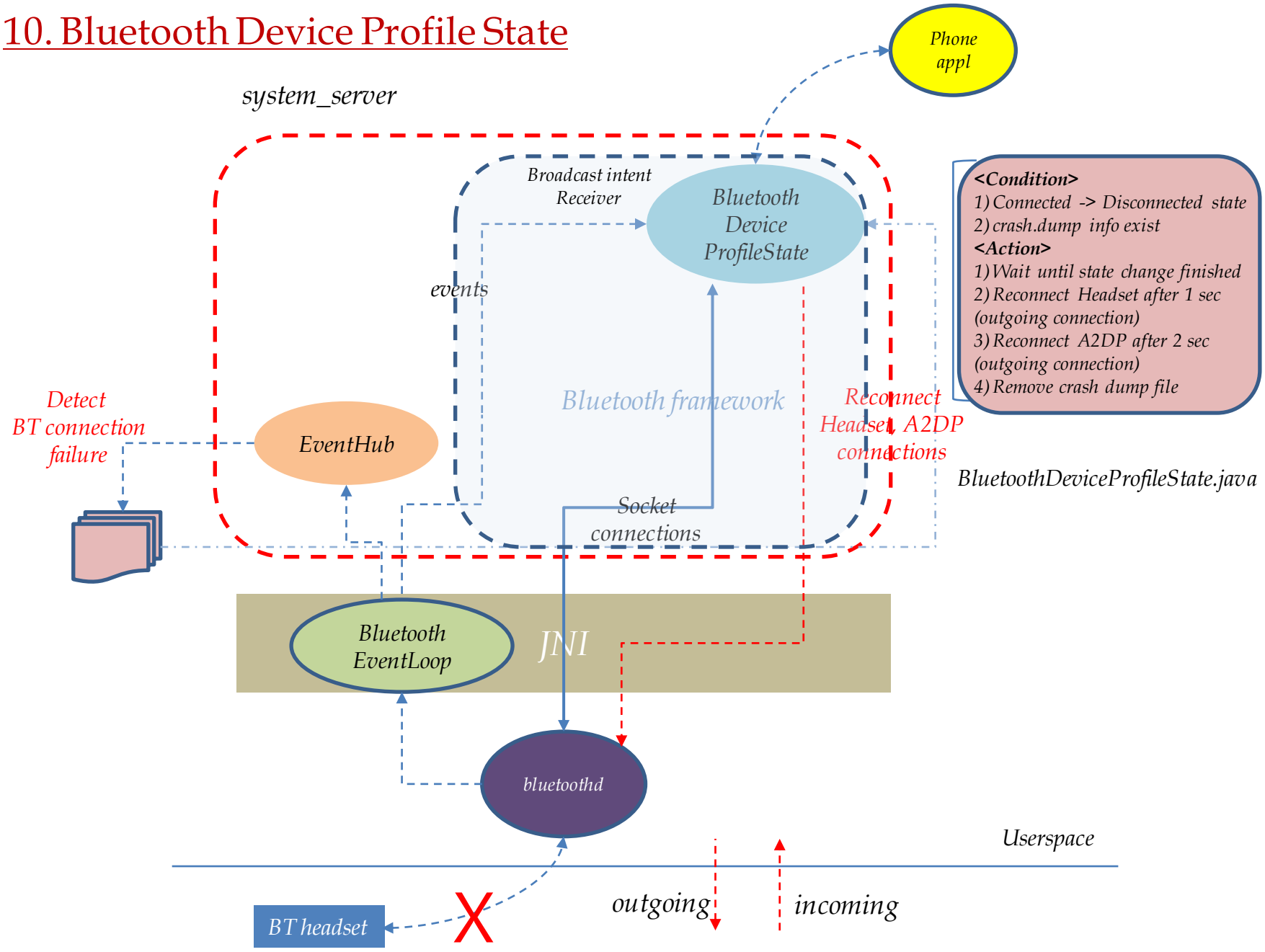


9. Bluetooth A2DP Profile

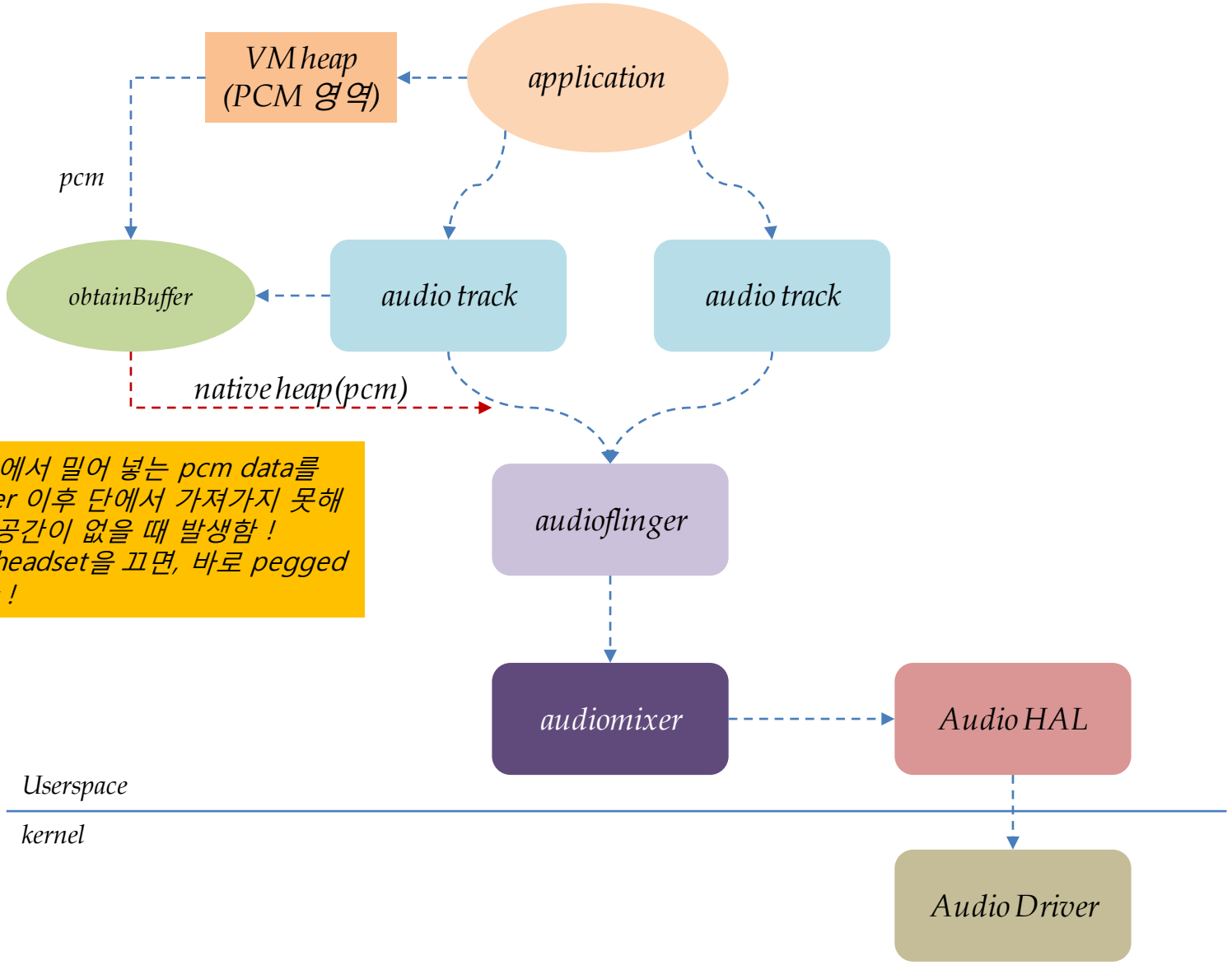


(*) appl에서 bluetoothd를 거쳐 최종 carkit까지 Audio stream과 control cmd가 전달될 것임.

10. Bluetooth Device Profile State



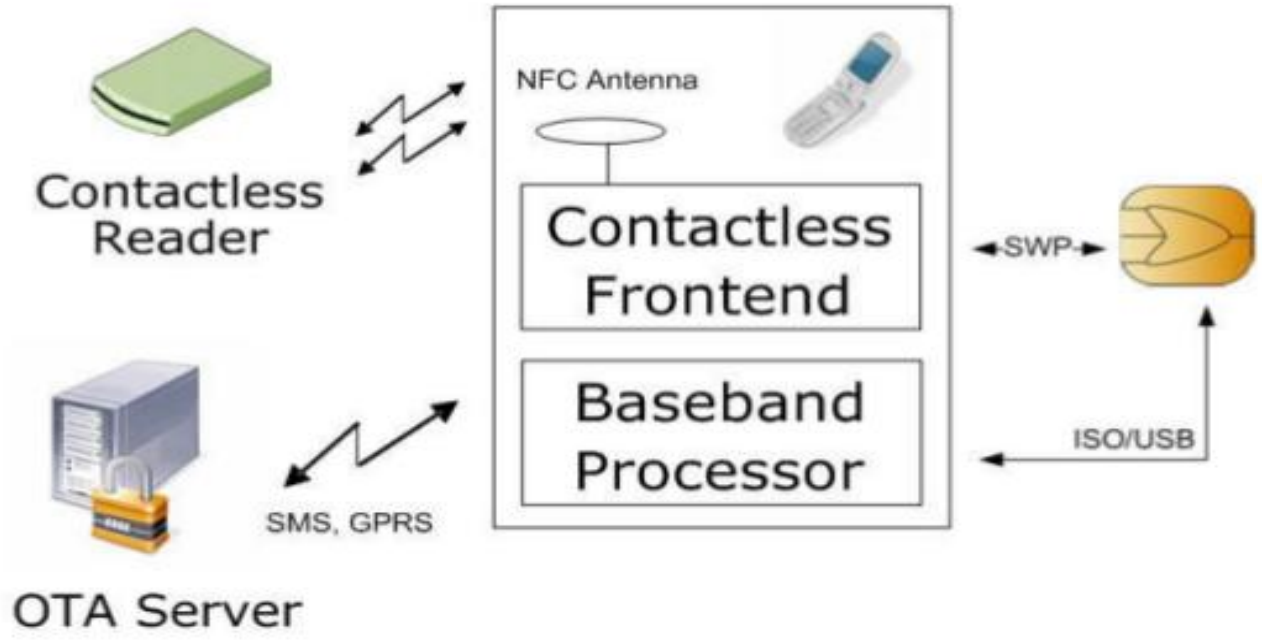
11. Case Study: *obtainBuffer timed out(is the CPU pegged?) issue*

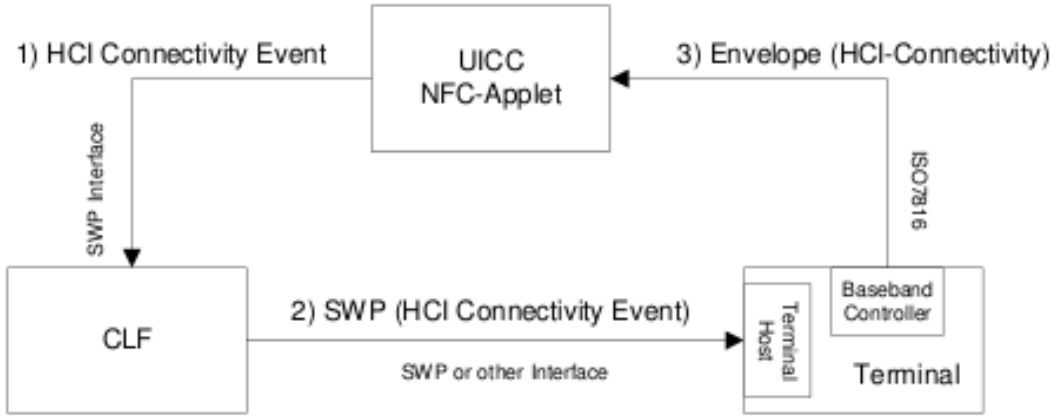


Application에서 밀어 넣는 pcm data를 Audio flinger 이후 단에서 가져가지 못해 frameAvail 공간이 없을 때 발생함 !
➔ 가령, BT headset을 끄면, 바로 pegged 에러 발생함 !

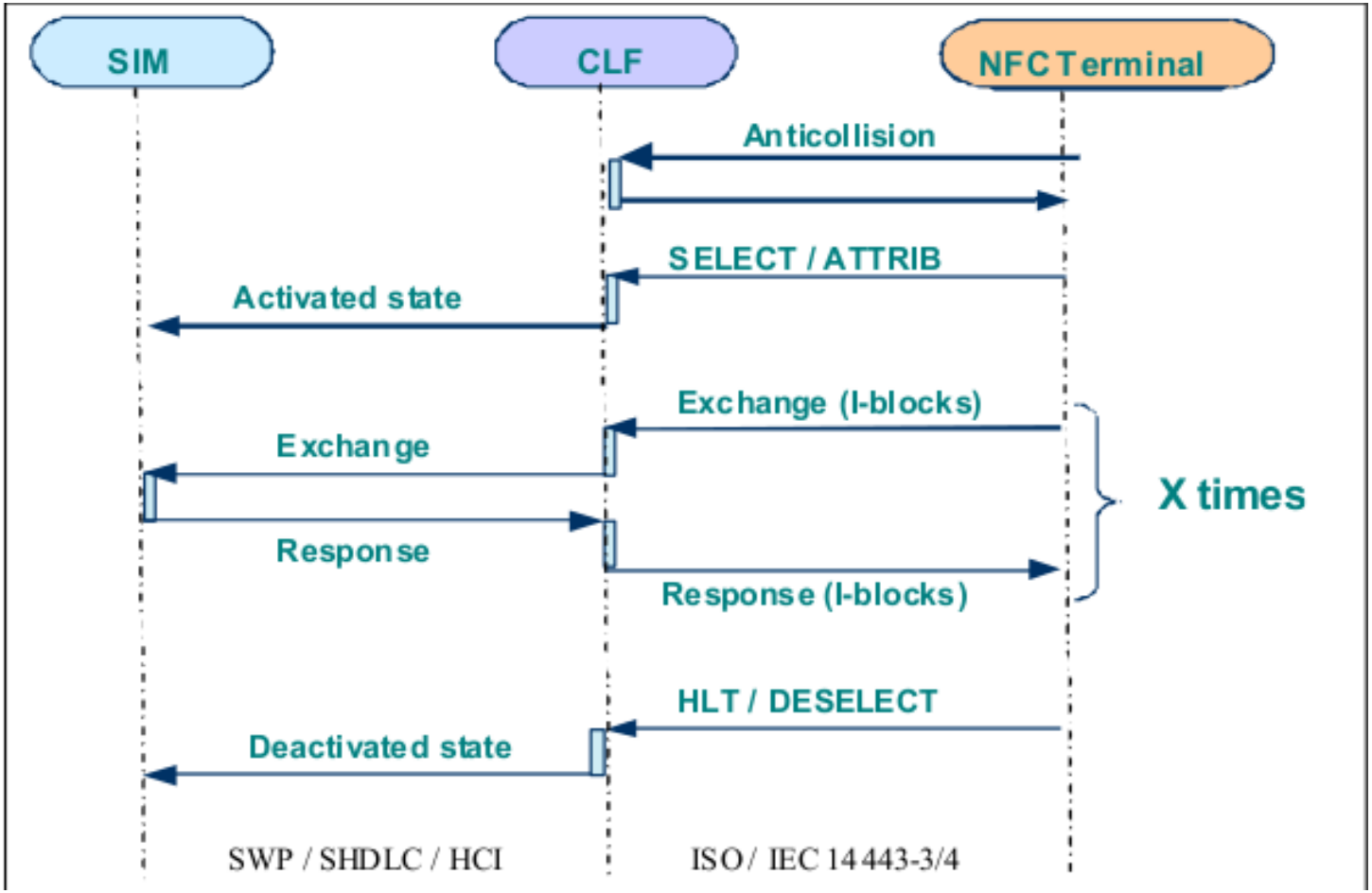
7. NFC Driver

1. NFC Overview(1)

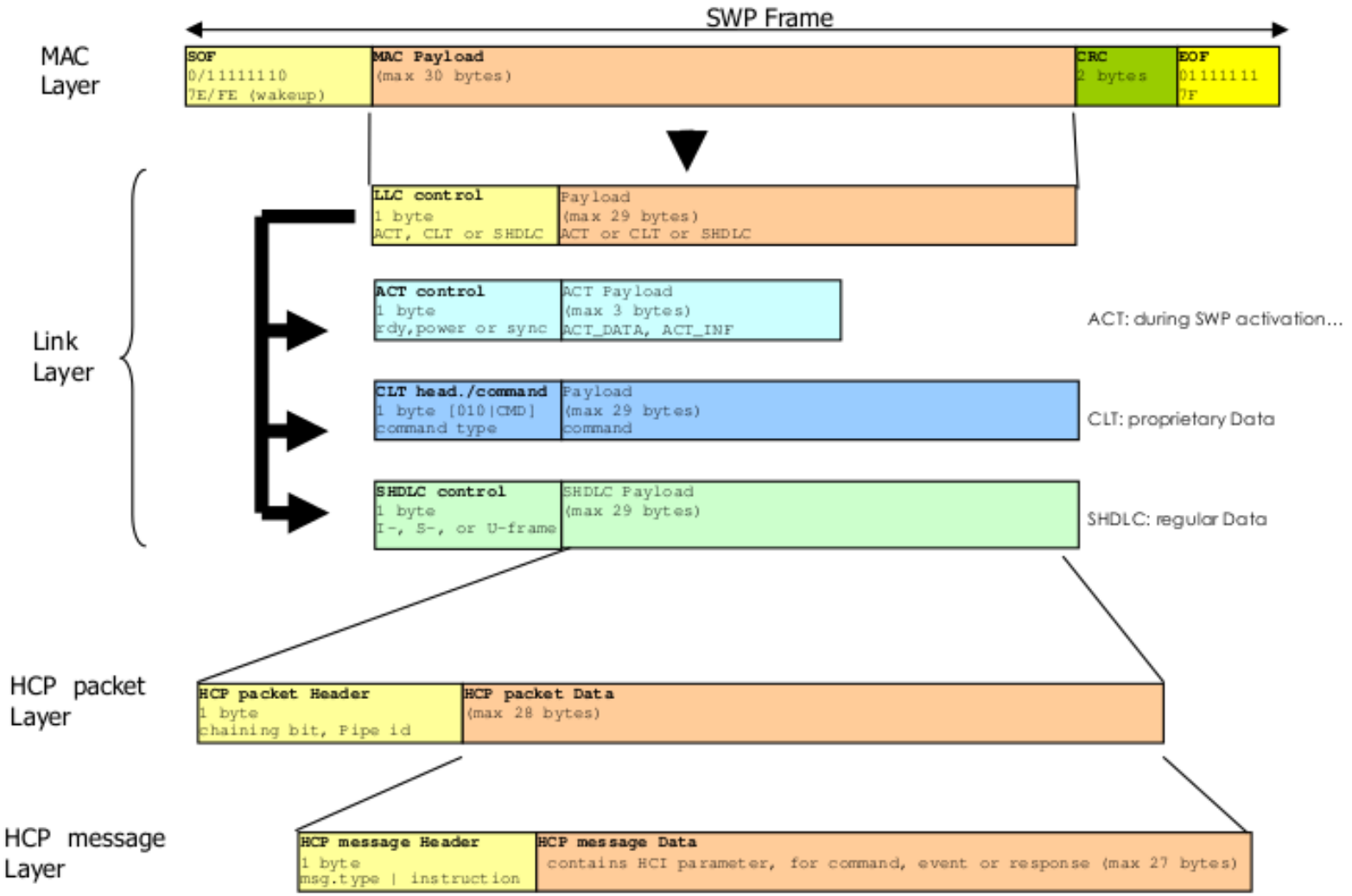




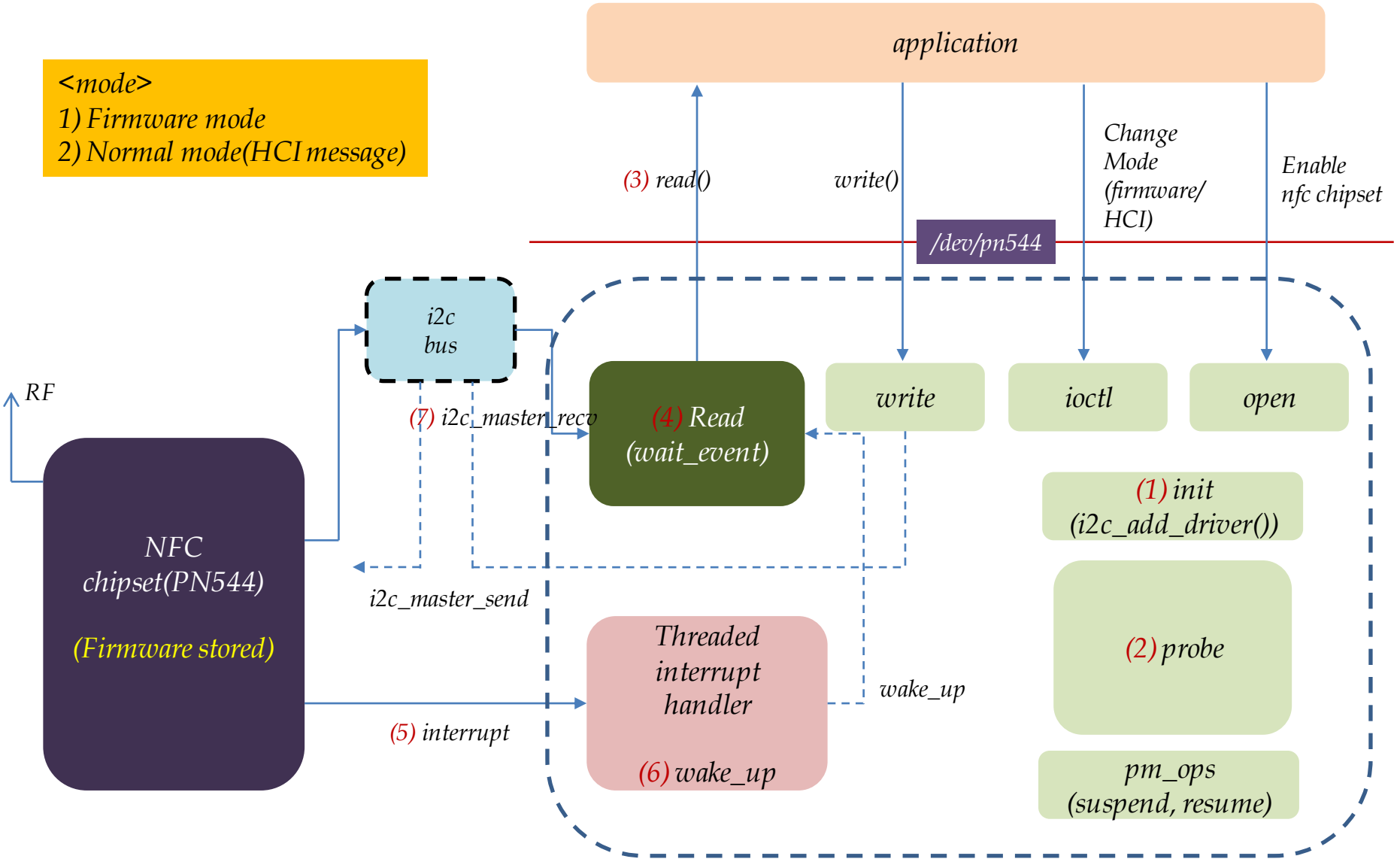
1. NFC Overview(2)



1. NFC Overview(3)



2. Example NFC Driver: ***PN544 NFC Driver(1)***



2. Example NFC Driver: *PN544 NFC Driver(2)*

1) pn544 NFC driver는 i2c client driver이다. 따라서 pn544_init()함수에서 i2c_add_driver() 함수를 호출하여 i2c client로 등록한다.

2) pn544_probe() 함수에서는

2-1) pn544 driver에서 사용하는 pn544_info data structure를 위한 buffer를 할당하고, 각각의 field를 초기화 한다.

2-2) 'read_wait' wait queue를 초기화한다(read 함수 <-> interrupt handler 간의 sync를 맞추기 위해 사용됨)

2-3) i2c_set_clientdata() 함수를 호출하여 i2c client를 위한 정보를 초기화한다.

2-4) nfc 장치로 부터 들어오는 interrupt 요청을 받아 처리하는 interrupt handler를 등록한다. pn544 driver는 이를 위해 특별히 threaded_interrupt handler 형태로 등록하고 있음.

2-5) 테스트 목적으로 sysfs에 파일을 생성함(pn544_attr).

2-6) misc driver 형태로 자신을 등록함.

3) application에서 read 함수를 호출할 경우, nfc chip으로 부터 i2c_master_recv() 함수를 사용하여 data를 읽어 들인다.

3-1) 단, 이때 nfc chip에 읽어 들일 data가 준비되지 않았을 수 있으므로, 읽기 작업 수행 전에 wait_event_interruptible() 함수를 호출하여 대기 상태로 들어간다.

3-2) probe함수에서 등록한 interrupt handler routine에서는 interrupt가 발생(nfc chip으로 부터 data 수신 가능 의미)할 경우, wake_up_interruptible() 함수를 호출하여, 대기 상태로 빠진 read 작업이 재개될 수 있도록 만들어 준다.

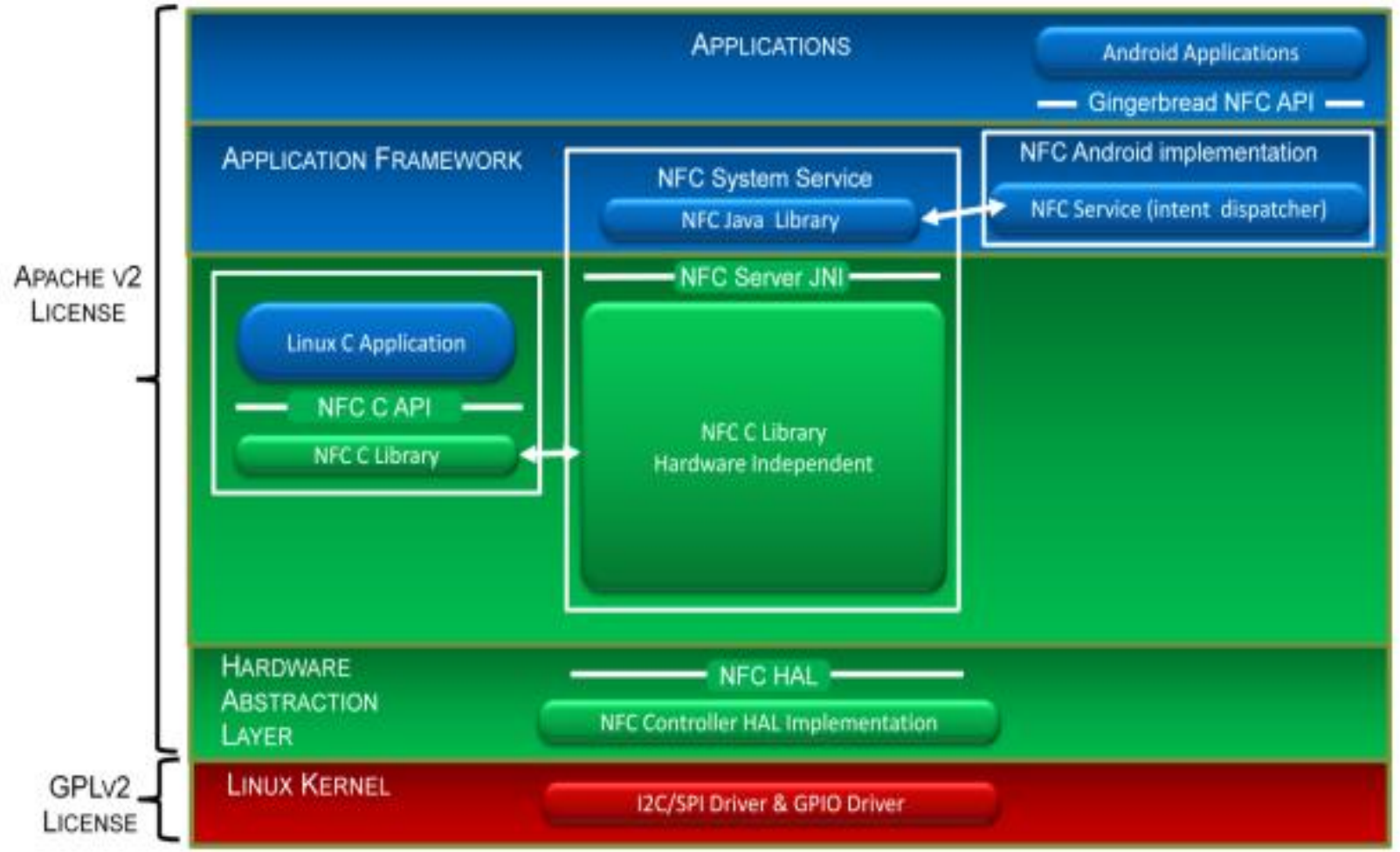
4) Application에서는 ioctl 함수를 사용하여, nfc chipset의 동작 방식을 firmware update mode와 normal mode(HCI mode)로 바꾸어 준다.

4-1) firmware update mode에서는 application에서 read 혹은 write 함수 호출시 firmware read 및 write 관련 작업이 수행되며,

4-2) normal HCI mode에서는 HCI message에 대한 송/수신이 가능하게 된다. HCI message(8bit header + body)는 최대 33bytes 이며, firmware message의 최대 길이는 1024bytes이다.

5) 자세한 것은 알 수 없으나, 무선 통신은 nfc chip 자체에서 수행하며, 이를 담당하는 firmware를 user application에서 교체할 수 있는 것으로 보인다.

3. Android NFC Framework Overview(1)



(*) 위의 그림은 Open NFC stack을 Android에 porting할 경우의 전체 구조를 그린 것으로, Android NFC Stack을 간접적으로 확인할 수 있다.

3. Android NFC Framework Overview(2)

<NFC 3 Operating Modes>

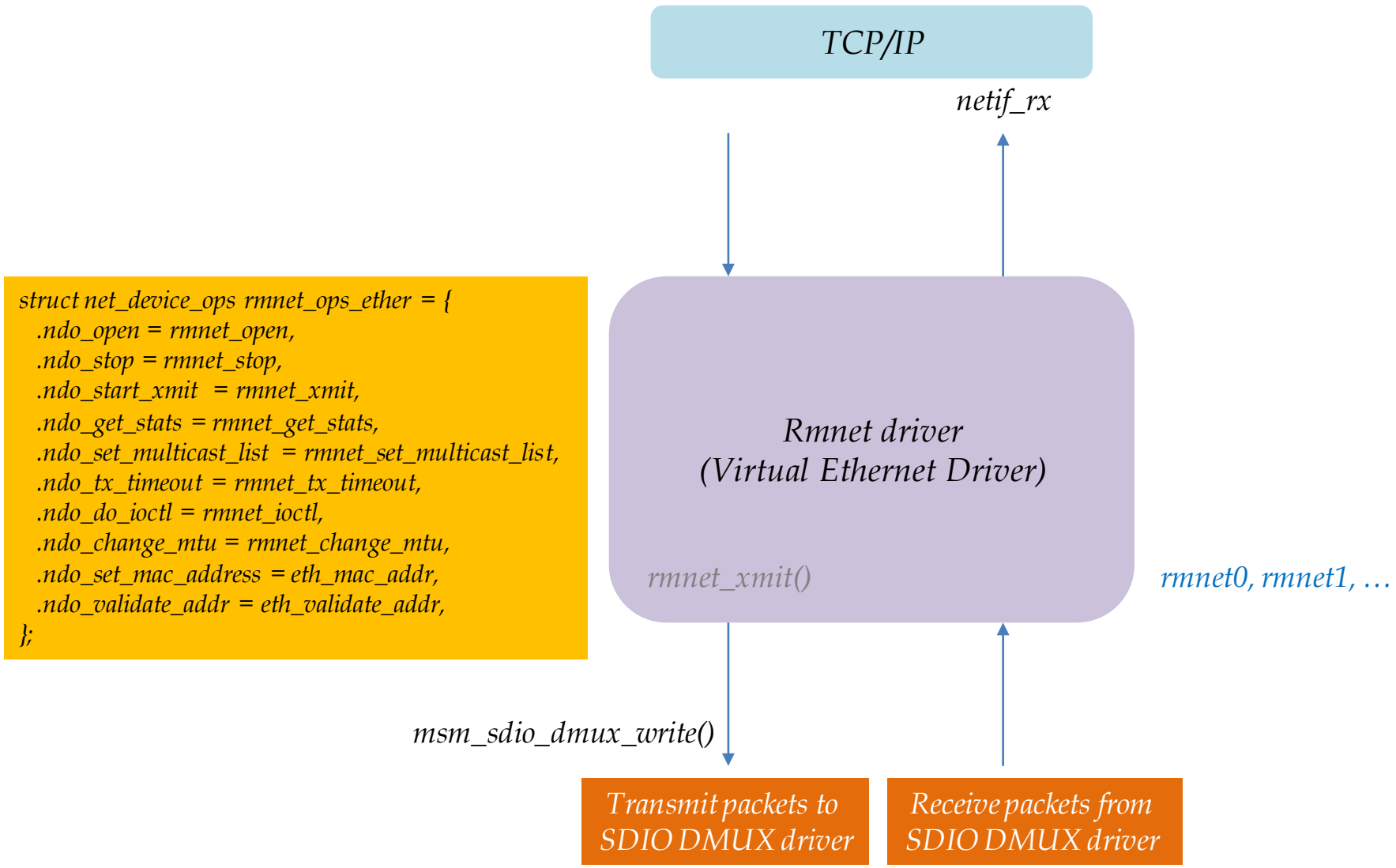
- 1) Card Emulation mode
- ➔ *Google Wallet이 필요하며, 아직 open되지 않았음.*
- 2) Reader Writer mode
- ➔ *Gingerbread 버전 부터 들어간 기능. Handset을 reader 혹은 writer로도 사용 가능 가능하도록 해 주는 기능(RFID와 차이점이기도 함).*
- 3) Peer-2-Peer mode
- ➔ *ICS에 추가된 Android Beam으로 알려진 기능(ISO 18092)*
- ➔ *상호 데이터 교환 가능.*

8. RmNet Driver

: 3G or 4G(app processor <-> modem processor)

(*) 실제 3G, 4G 통신은 modem processor에서 수행하며, 본장에서는 application processor와 modem processor간의 통신을 위한 driver를 소개하고자 한다.

1. RmNet Ethernet Driver: ***Virtual Ethernet Driver(1)***



1. RmNet Ethernet Driver: *Virtual Ethernet Driver(2)*

<rmnet_init() 함수 분석>

0) 변수 선언

```
struct device *d;
struct net_device *dev;
struct rmnet_private *p;
```

1) alloc_netdev() 함수 호출하여 net_device 할당. 이 줄을 포함하여 아래 step을 RMNET_DEVICE_COUNT(=8) 만큼 반복!

2) rmnet_private pointer(p) 값 초기화 및 몇개의 field 값 채움.

3) tasklet 하나 초기화

=> _rmnet_resume_flow() 함수가 나중에 호출될 것임.

4) wake_lock_init

5) completion 초기화

6) p->pdrv.probe = msm_rmnet_smd_probe;

7) ret = platform_driver_register(&p->pdrv);

8) ret = register_netdev(dev);

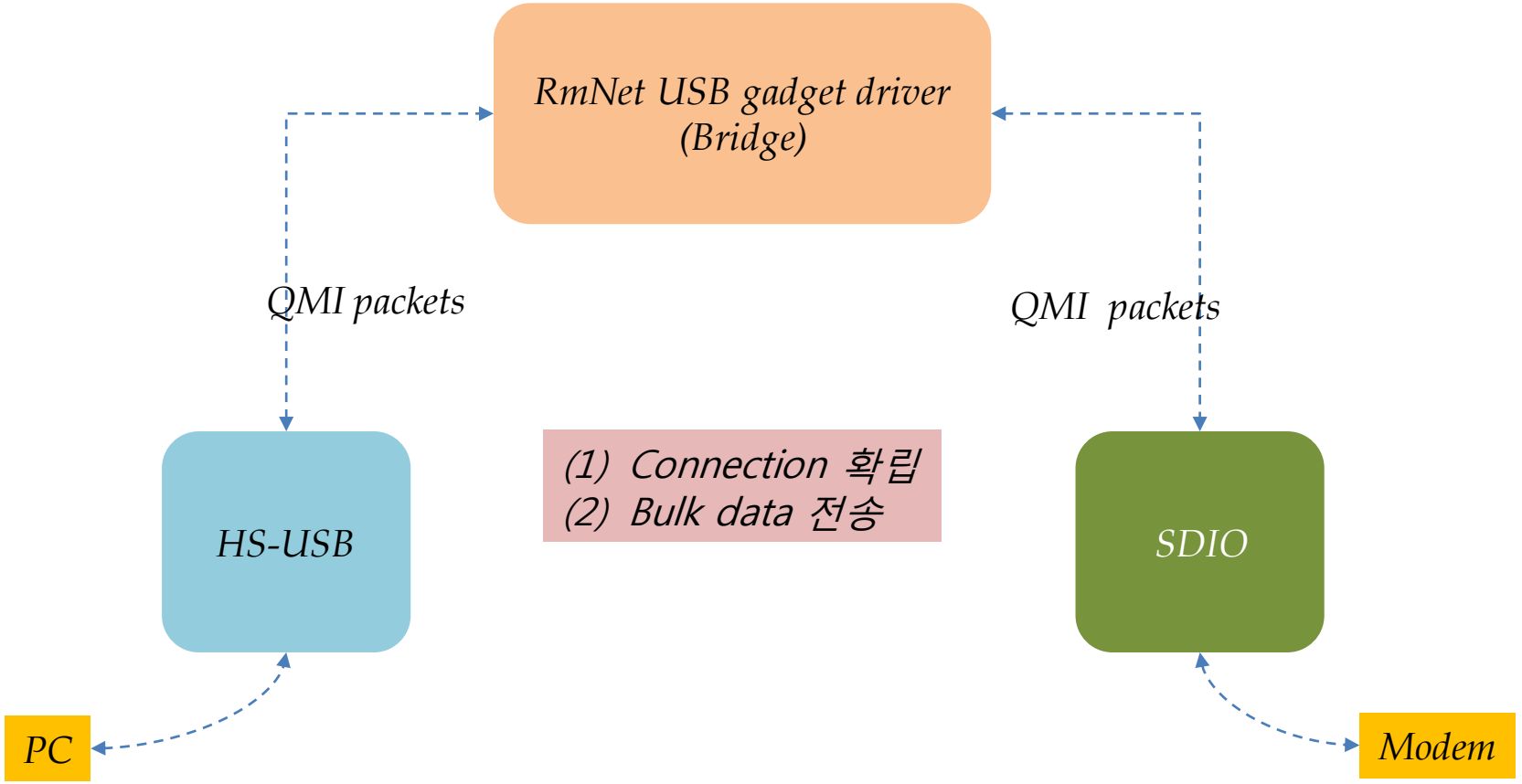
9) sysfs entry 생성

- *device_create_file(d, &dev_attr_timeout)*
- *device_create_file(d, &dev_attr_wakeups_xmit)*
- *device_create_file(d, &dev_attr_wakeups_rcv)*
- *device_create_file(d, &dev_attr_timeout_suspend)*

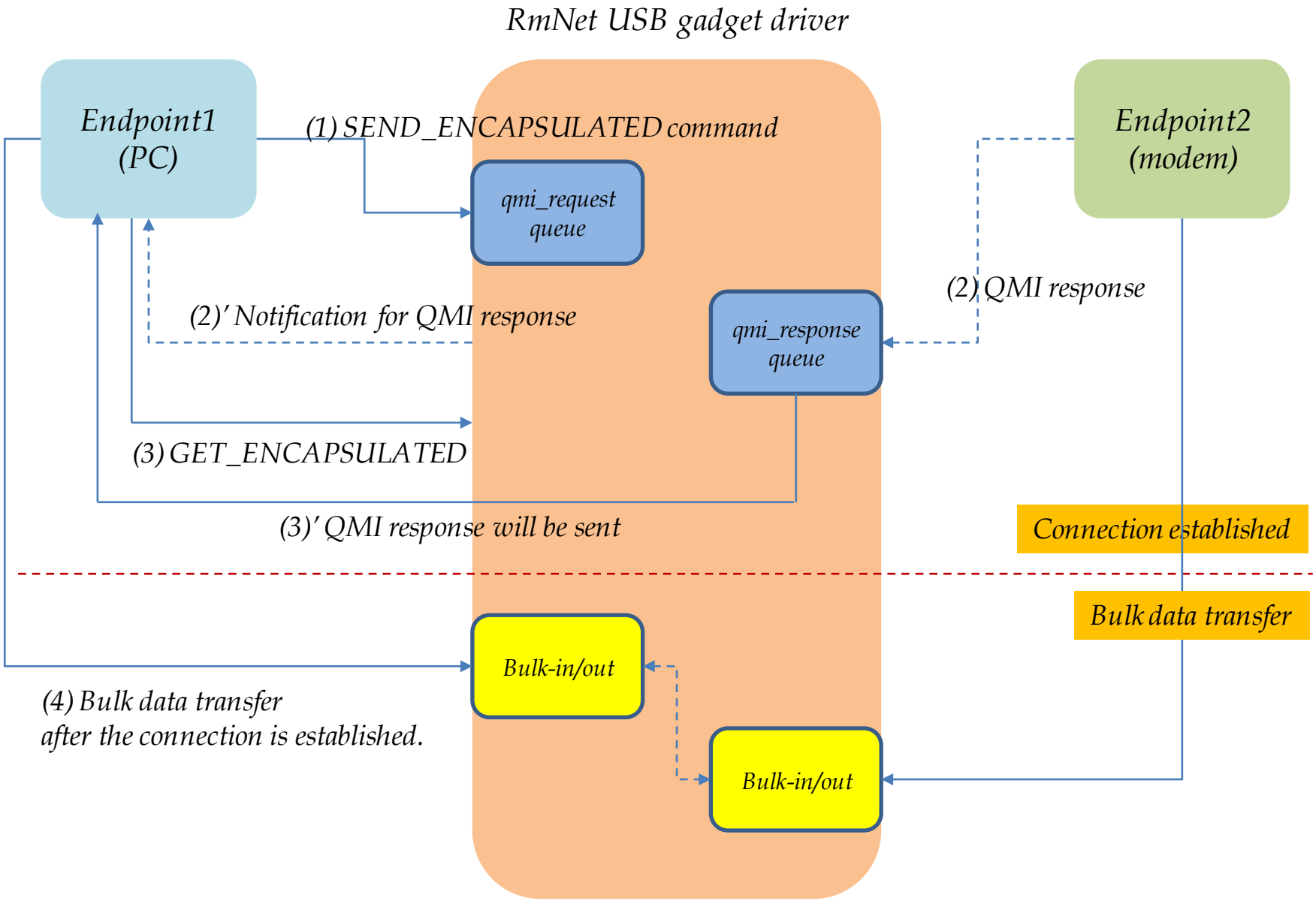
2. RmNet USB Gadget Driver: ***Bridge Driver(1)***

(*) gadget driver는 usb host(PC)와 통신하는 usb device 형태의 드라이버를 일컫는다.

<rmnet usb gadget driver>
- drivers/usb/gadget/f_rmnet.c
- drivers/usb/gadget/f_rmnet_sdio.c (SDIO 사용시)
- drivers/usb/gadget/f_rmnet_smd.c (SMD 사용시)



2. RmNet USB Gadget Driver: ***Bridge Driver(2)***



2. RmNet USB Gadget Driver: *Bridge Driver(3)*

<rmnet_sdio_function_add() 분석>

1) struct rmnet_sdio_dev 변수 선언 및 buffer(dev) 할당.

---

2) k_rmnet_work work queue 생성

3) disconnect_work 초기화

4) set_modem_ctl_bits_work 초기화

5) ctl_rx_work 초기화

6) data_rx_work 초기화

7) sdio_open_work

8) sdio_close_work

---------- 까지 각각의 work queue의 용도 분석해야 함.

9) qmi_req_q: qmi request queue 초기화

10) qmi_resp_q: qmi response queue 초기화

11) tx_skb_queue socket buffer 초기화

12) rx_skb_queue socket buffer 초기화

13) 1)에서 할당한 dev 변수의 나머지 field 채움

=> 아래field가 gadget driver의 기본 요소로 보임.

```
dev->function.name = "rmnet_sdio";
dev->function.strings = rmnet_sdio_strings;
dev->function.descriptors = rmnet_sdio_fs_function;
dev->function.hs_descriptors = rmnet_sdio_hs_function;
dev->function.bind = rmnet_sdio_bind;
dev->function.unbind = rmnet_sdio_unbind;
dev->function.setup = rmnet_sdio_setup;
dev->function.set_alt = rmnet_sdio_set_alt;
dev->function.disable = rmnet_sdio_disable;
dev->function.suspend = rmnet_sdio_suspend;
```

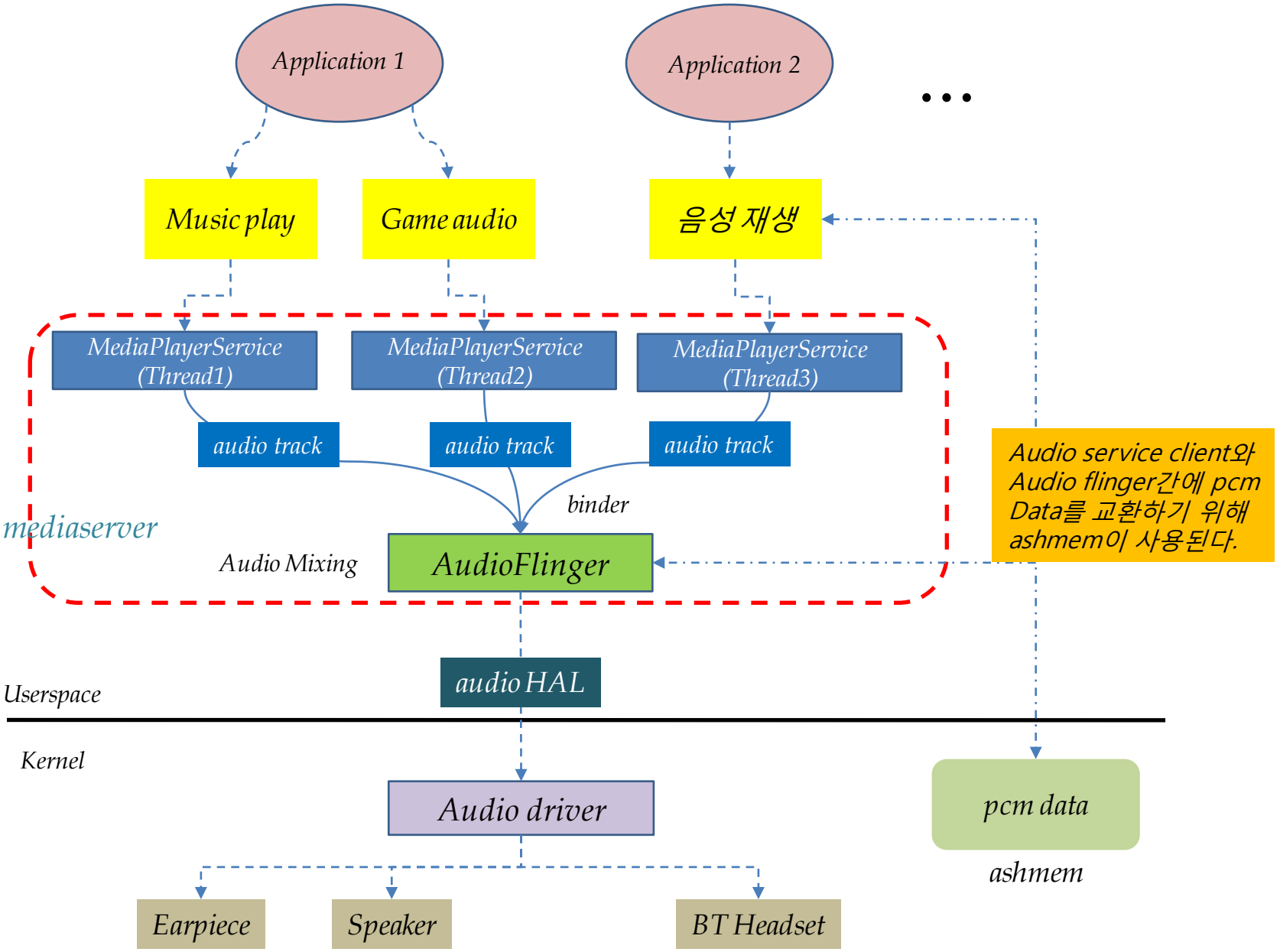
14) usb_add_function(c, &dev->function); 를 호출하여 gadget driver로 등록

15) rmnet_sdio 관련하여 debugfs에 항목 추가

9. Audio Codec Driver

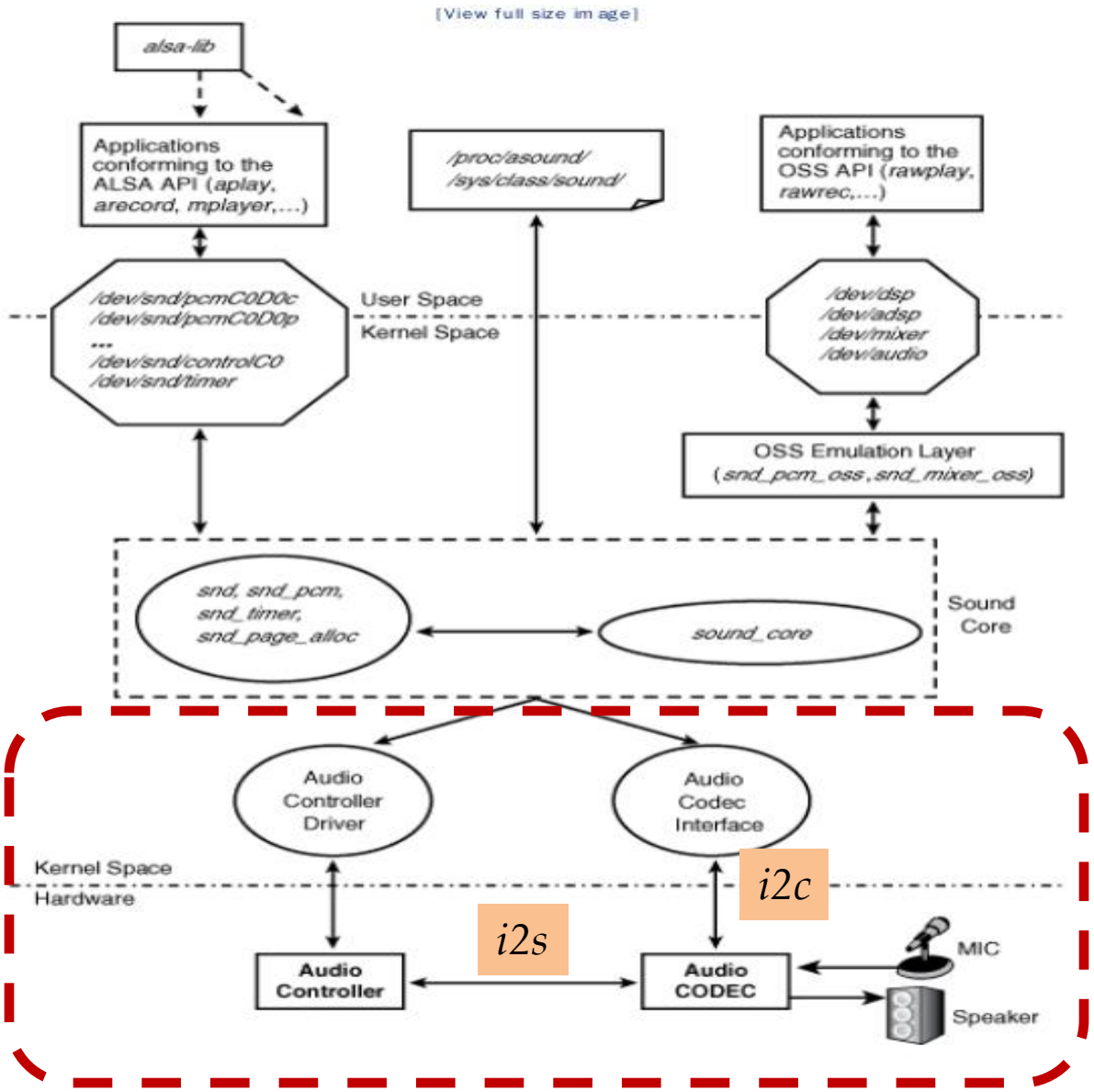
: ALSA, I2S, I2C(codec), DMA

1. Android Audio Architecture

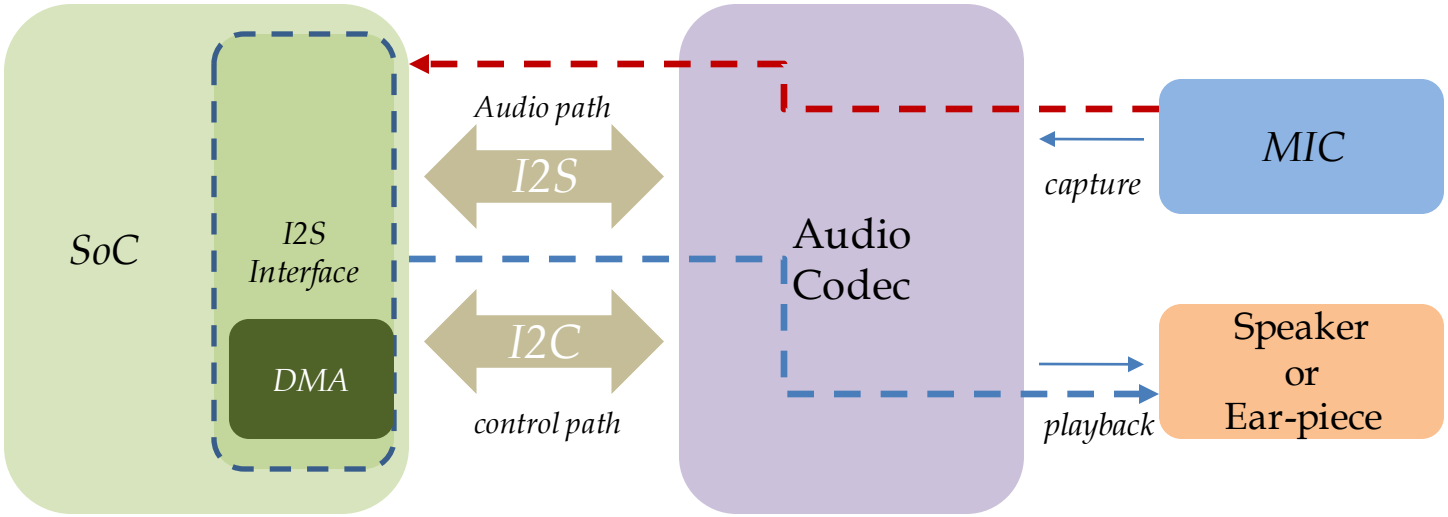


2. Audio Driver: *Linux ALSA Driver*

(*) 아래 그림은 참고 문서[1]에서 복사해 온 것임.

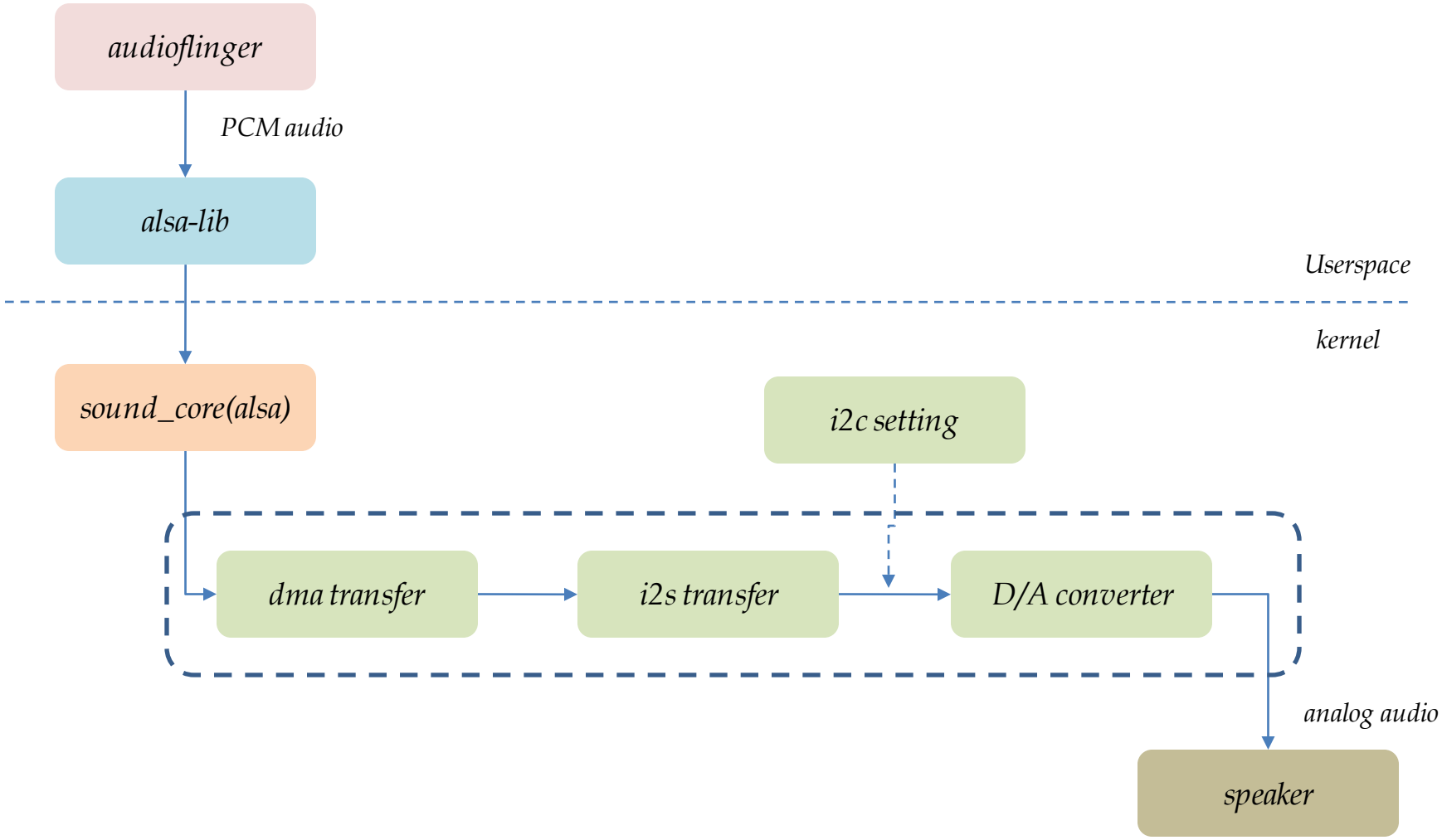


2. Audio Driver: ***I2S, I2C(codec)***



(*) decoding된 PCM audio data(audioflinger 출력)가 I2S interface를 통해 audio codec으로 전달된 후, analog로 변환되어 Speaker나 Earpiece로 출력된다.
(*) 한편, MIC로 부터 입력된 analog data는 Audio codec를 통해 digital로 변환된 뒤, CPU로 흘러 들어가게 된다(I2S).
(*) I2S로 전달되는 sound data는 경우에 따라서 빠른 전송을 보장하기 위하여 DMA와 연계하기도 한다.
(*) I2S는 audio data의 흐름과 관련이 있으며, I2C는 Audio Codec의 설정(register 값 변경)과 연관이 있다.

2. Audio Driver: ***PCM output flow***



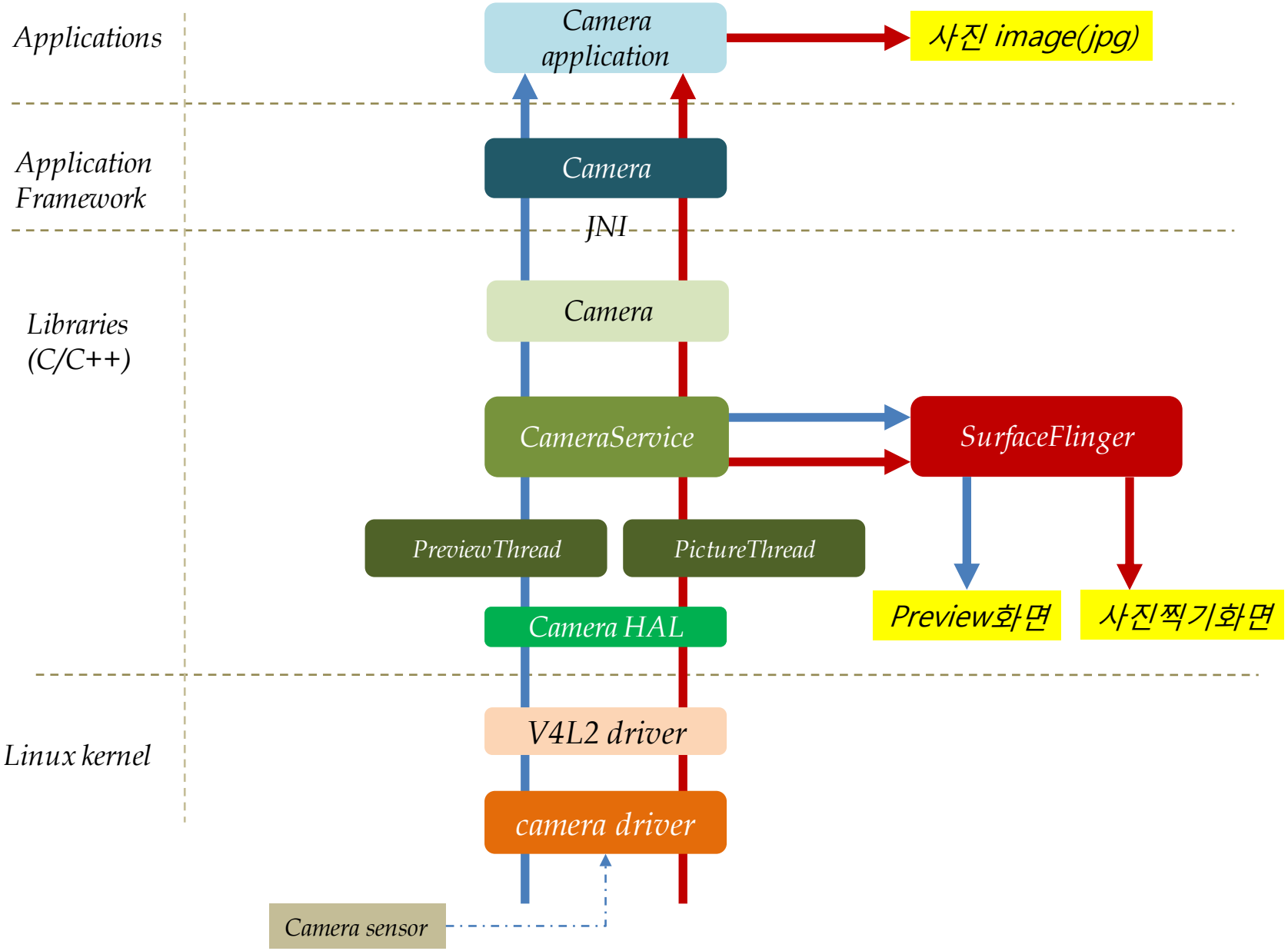
3. Audio Driver: ***ear-jack detection***

- *<TODO>*

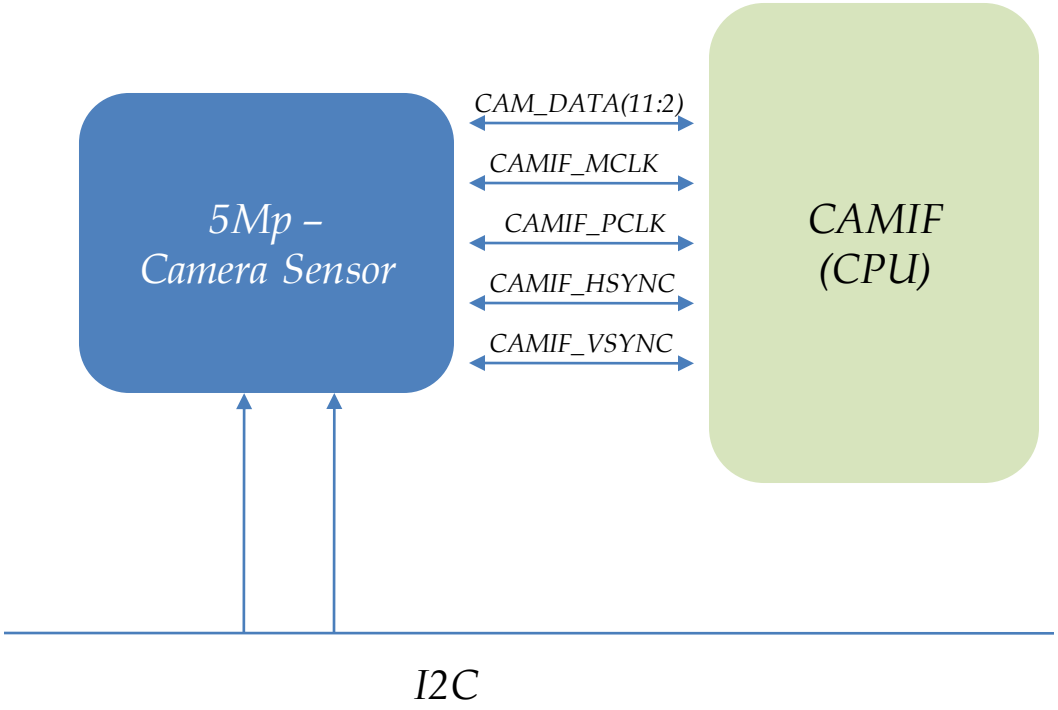
10. Camera Driver

: camera sensor(i2c), V4L2

1. Android Camera Service Architecture



2. Camera Concept(1)

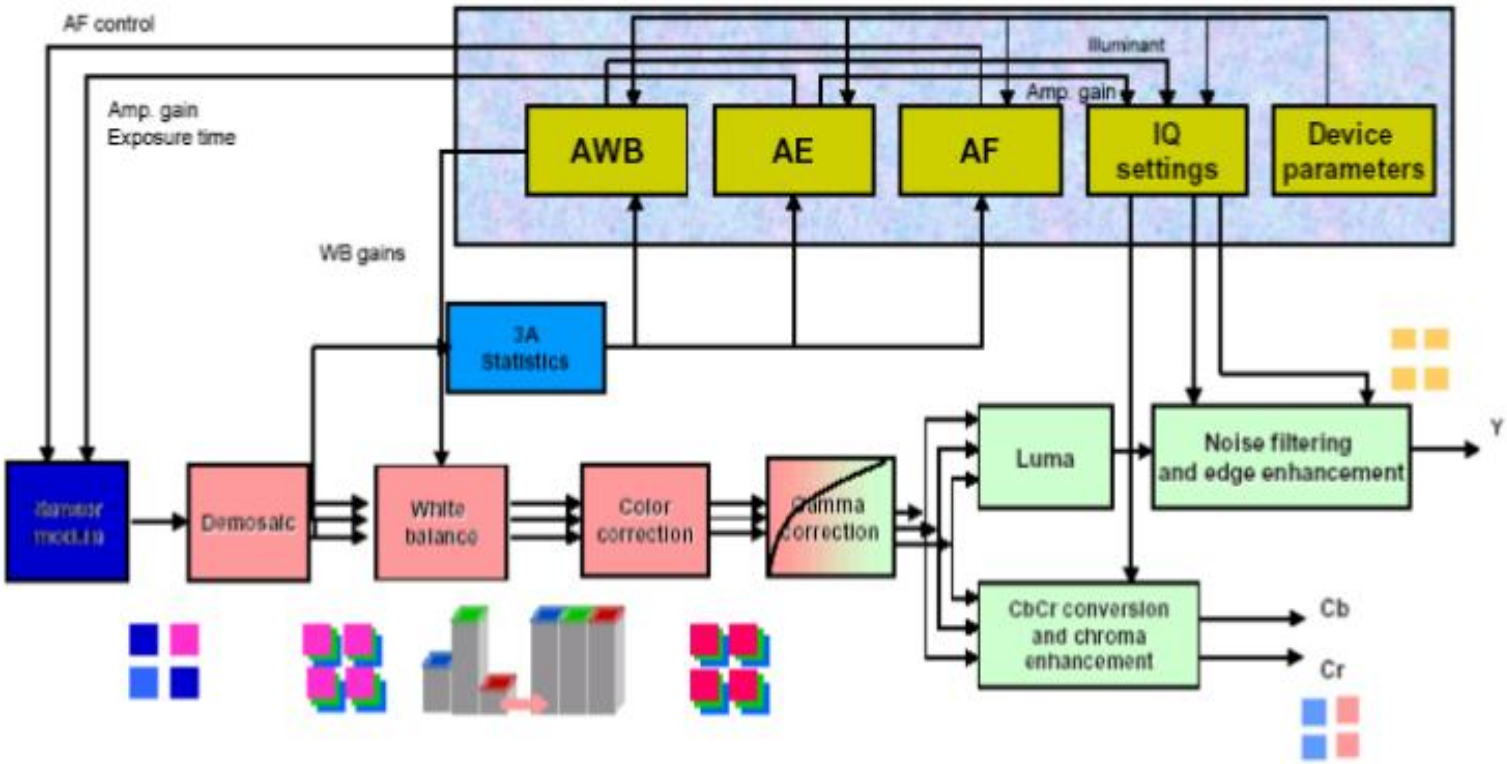


(*) Camera inteface ...
CPI(Camera Parallel Interface),
CSI(Camera Serial Inteface),
MIPI

2. Camera Concept(2)

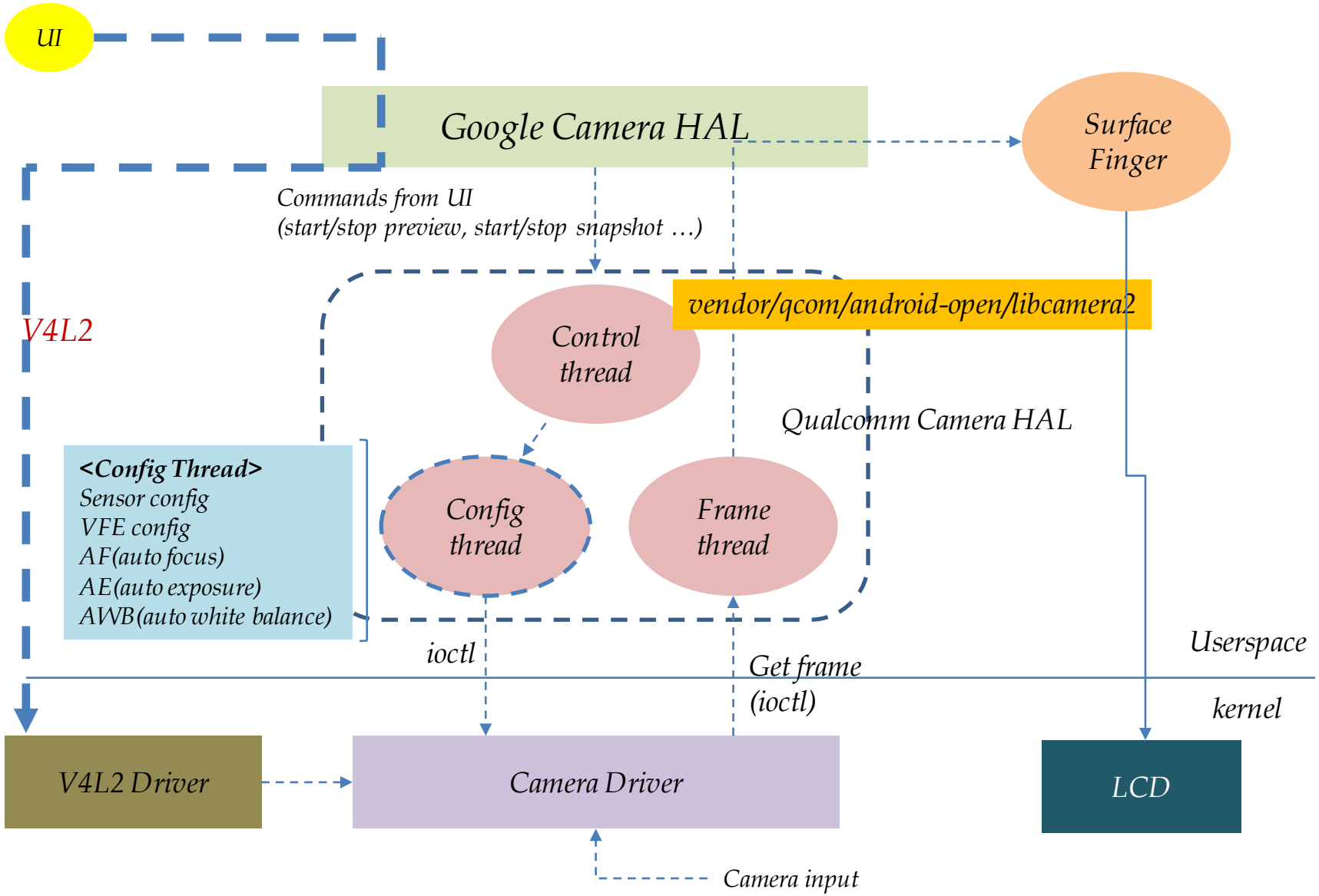
(*) 아래 그림은 인터넷에서 복사해 온 것임.

•Camera Signal Processing

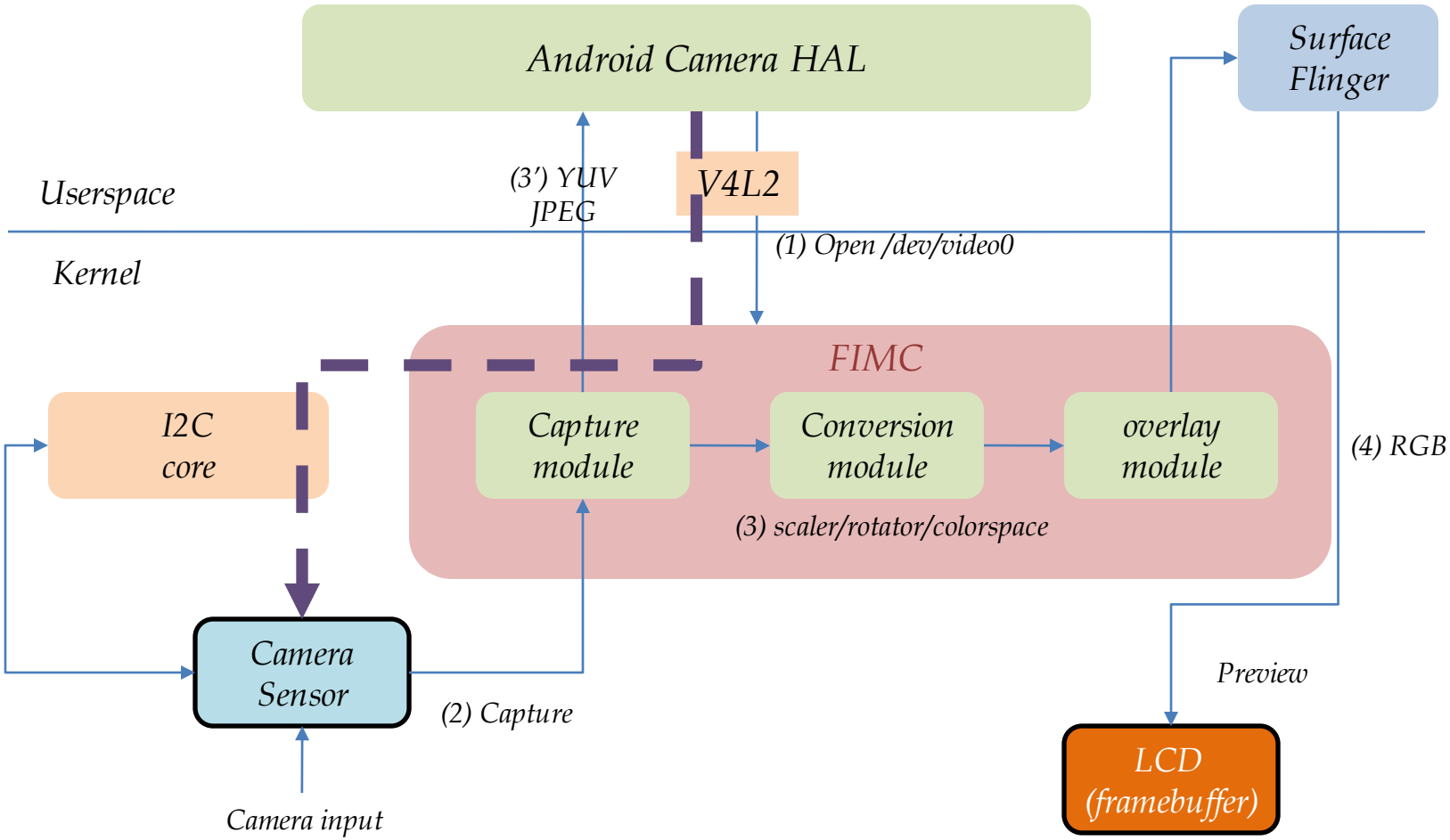


(*) 위의 그림은 camera sensor로 입력된 analog raw data를 3A(AWB, AE, AF) 처리를 통해, YUV(YCbCr)로 변환하는 과정을 표현한 것이다.

3. Camera Driver(1) – ***Qualcomm chip***



3. Camera Driver(2) – ***Samsung Chip***



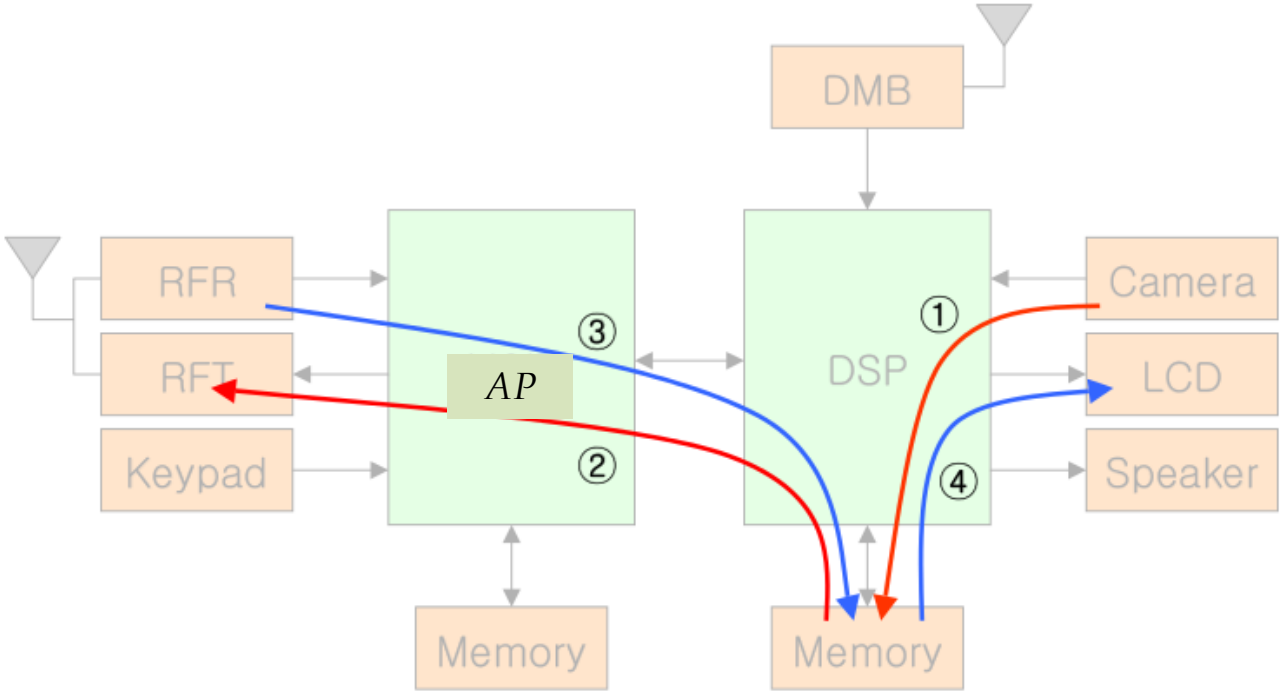
4. V4L2 Driver

- *<TODO>*

5. VT(Video Telephony) Architecture

(*) 아래 그림은 인터넷에서 복사해 온 것임.

•H.324/M + H.263 QCIF video + AMR speech

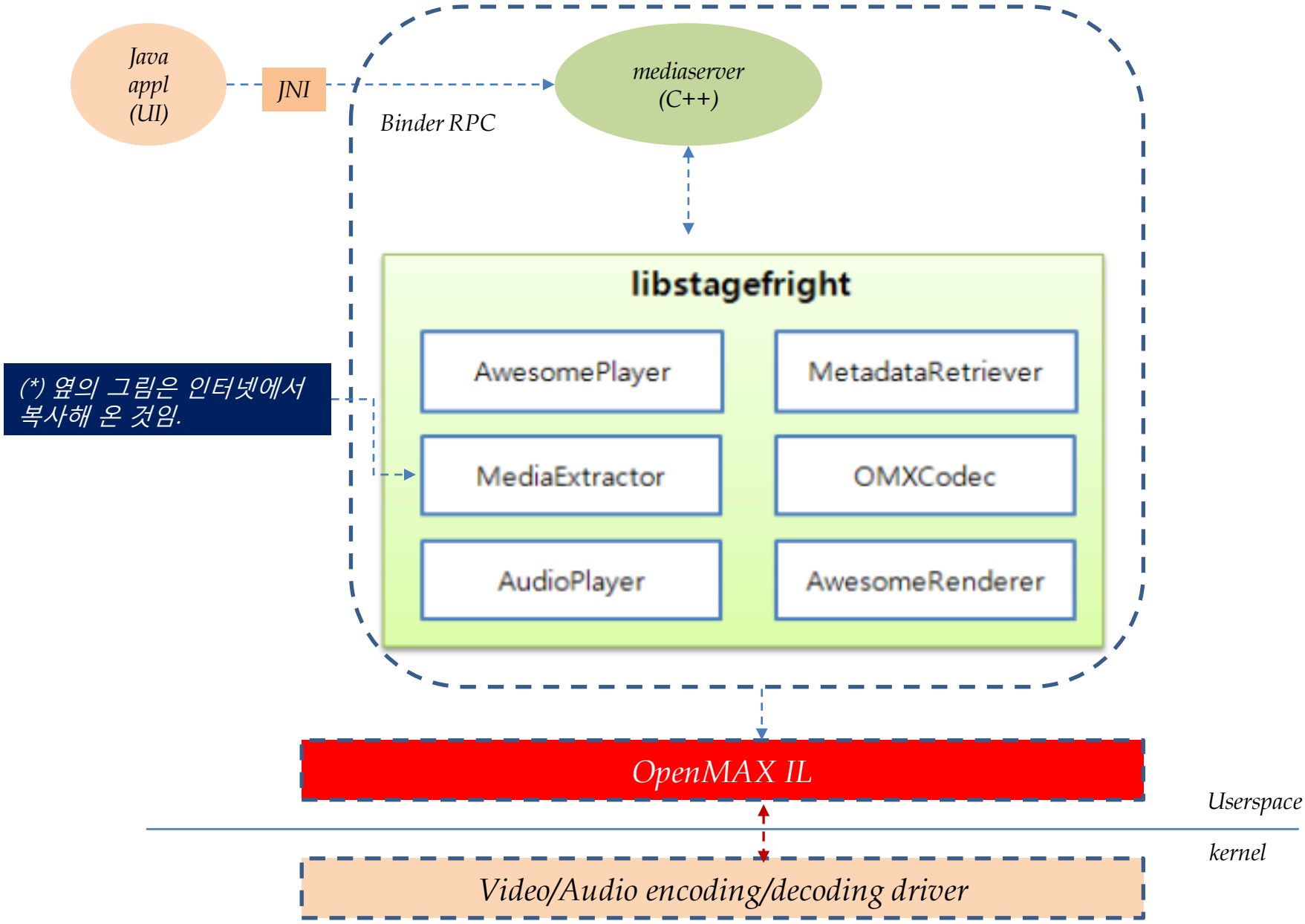


VT = Camera + Telephony + Video/AudioEncoding/Decoding

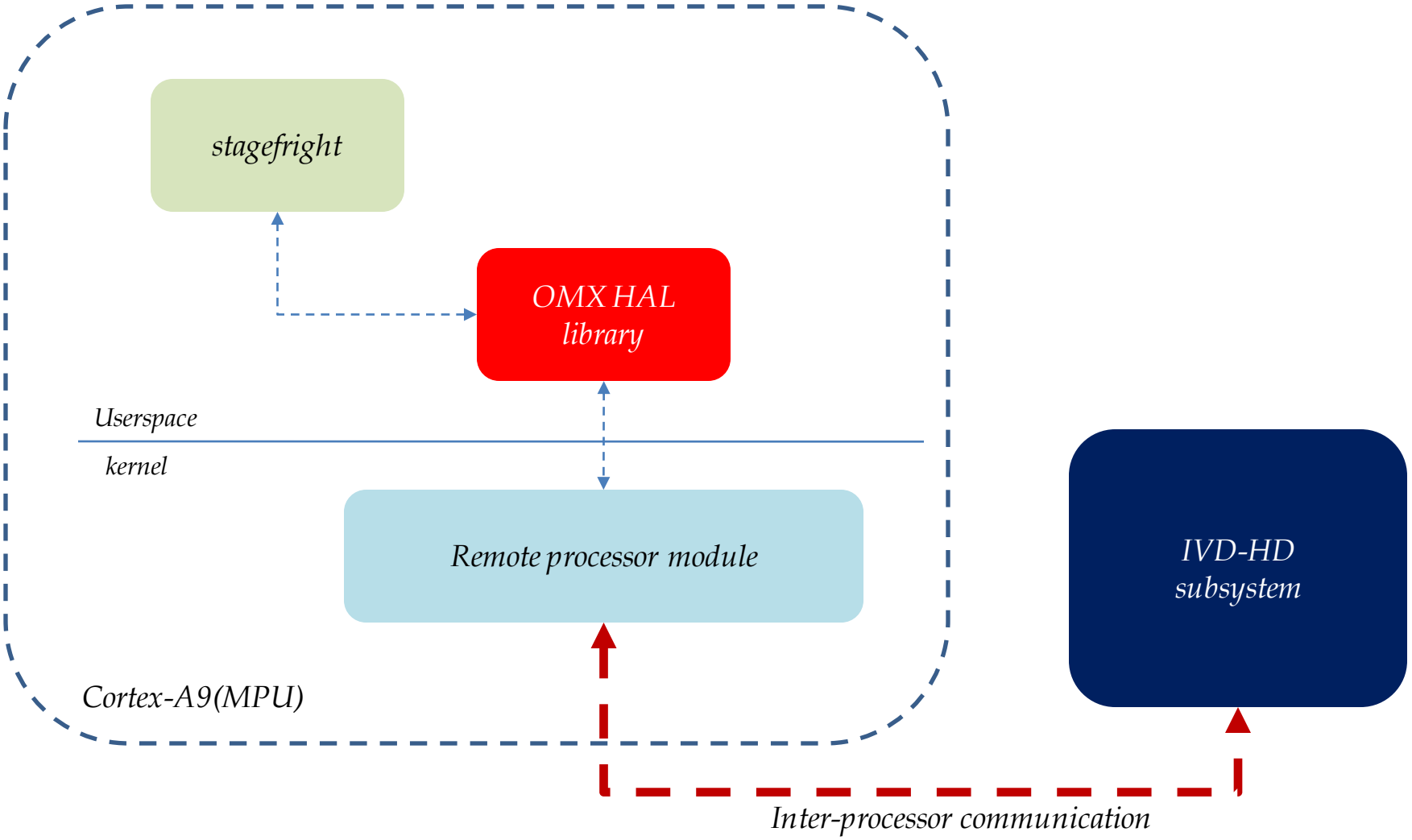
11. A/V Codec Driver

: Audio/Video Encoder/Decoder

1. Audio/Video Codec Architecture(1) - *Overview*



1. Audio/Video Codec Architecture(2) - *Video H/W Acceleration(TI OMAP)*

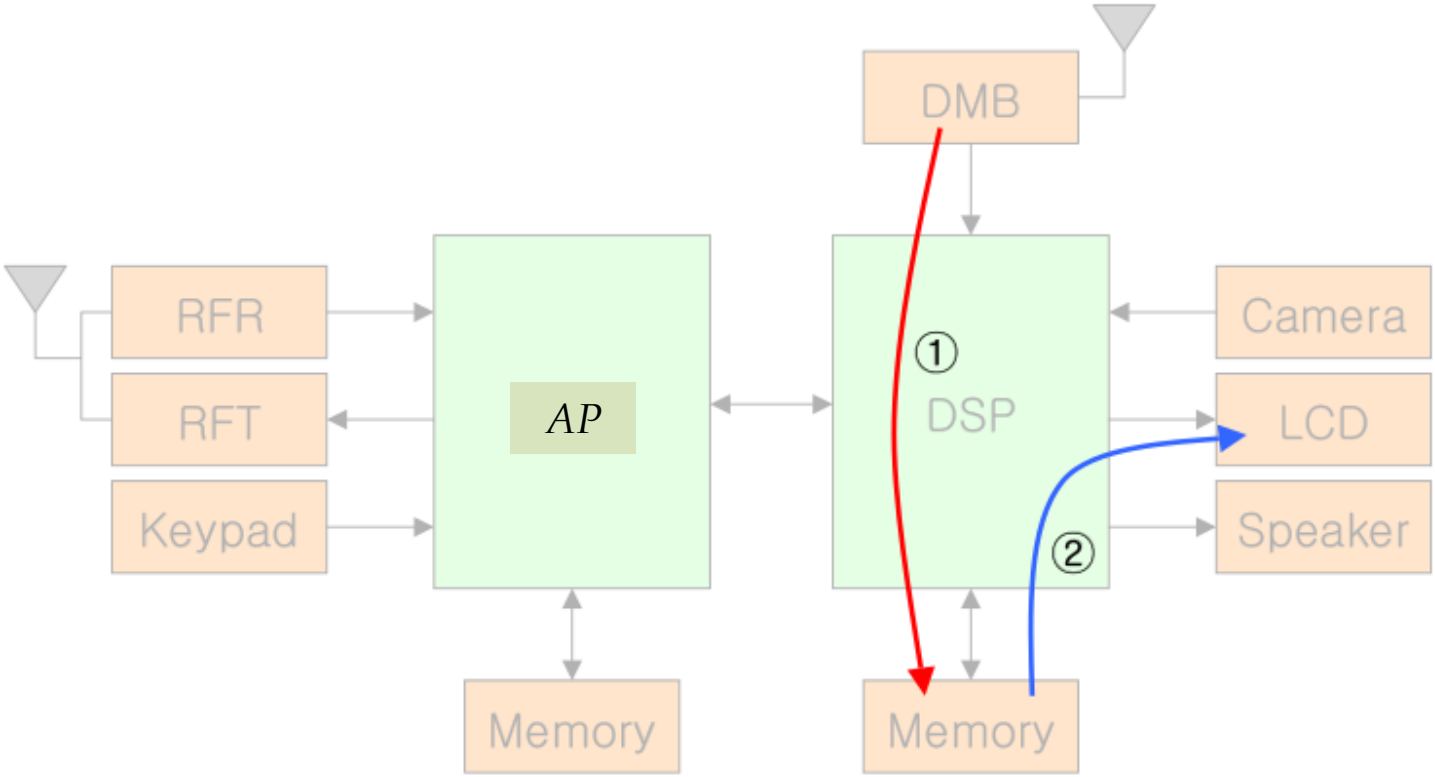


12. T-DMB

1. DMB Architecture(1)

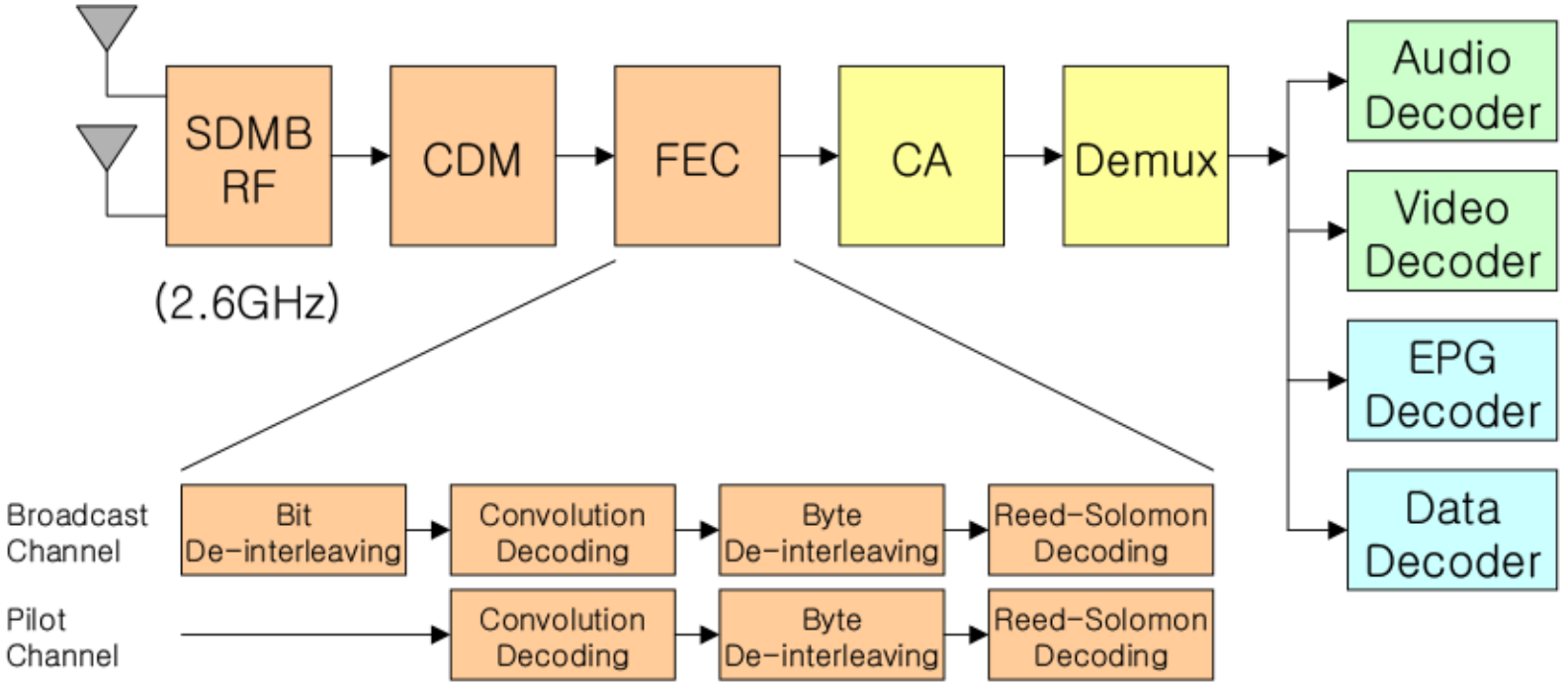
(*) 아래 그림은 인터넷에서 복사해 온 것임.

•MPEG-2 TS + H.264 QVGA video + AACplus audio



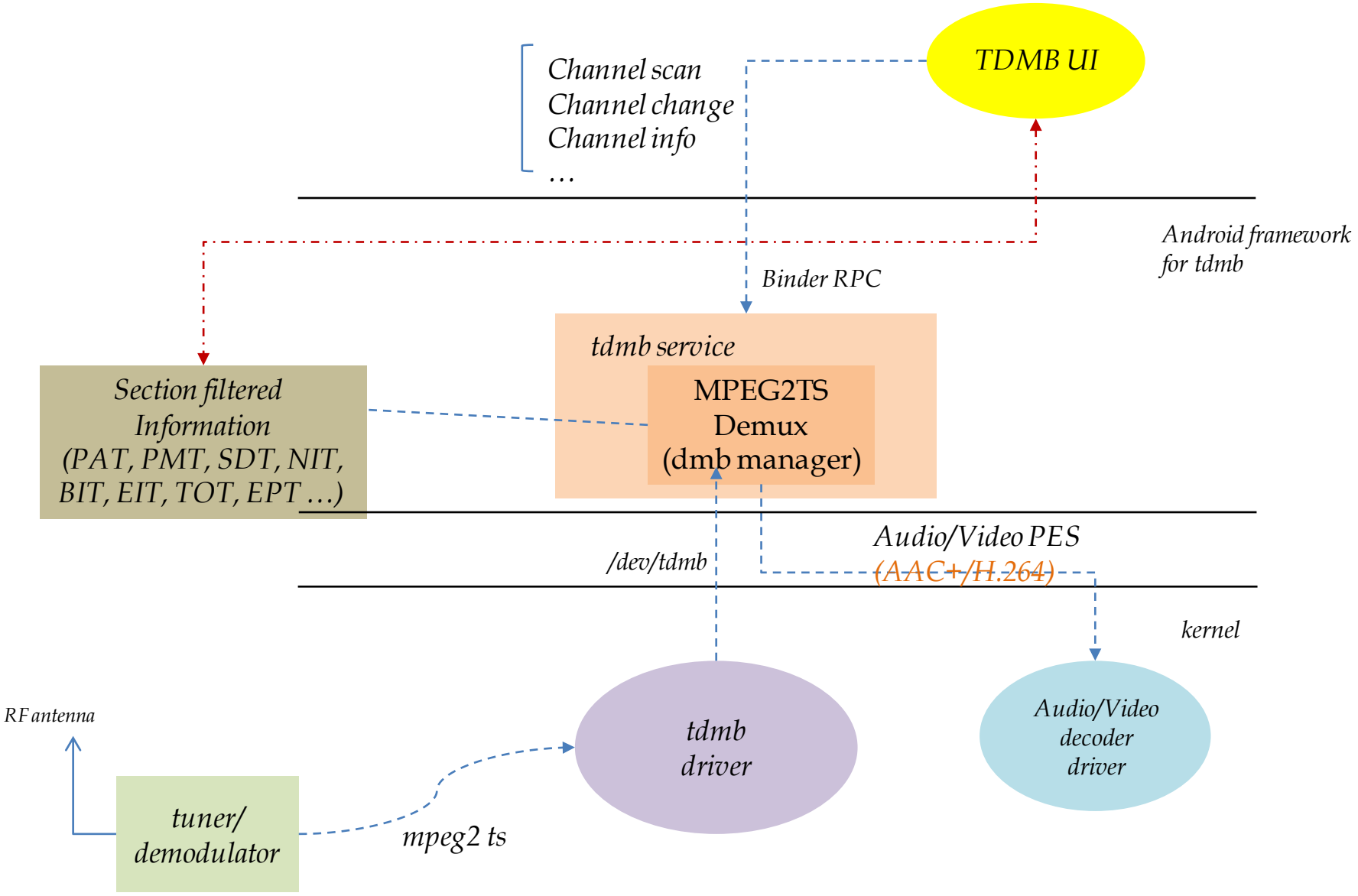
1. DMB Architecture(2)

(*) 아래 그림은 인터넷에서 복사해 온 것임.



(*) SDMB 그림이나, TDMB의 경우도 유사함.

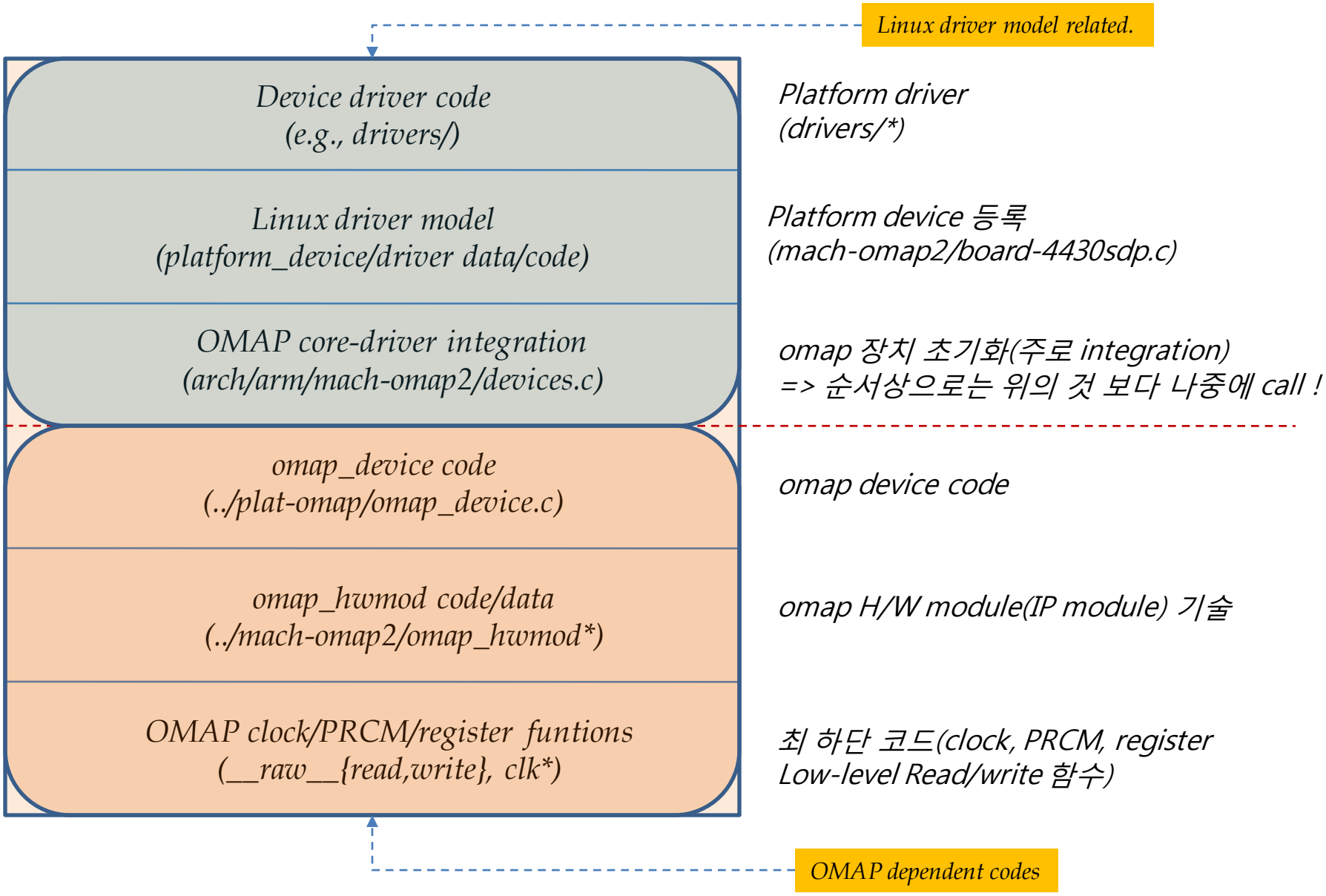
1. DMB Architecture(3)



부록 1 - Kernel Machine Code Review

: arch/arm/mach-omap2, plat-omap

1. OMAP Machine Code Architecture



2. Board(Machine) initialization routine(1)

```
MACHINE_START(OMAP_4430SDP, "OMAP4 blaze board")
        /* Maintainer: Santosh Shilimkar - Texas Instruments Inc */
        .boot_params    = 0x80000100,
        .reserve        = omap_4430sdp_reserve,
        .map_io         = omap_4430sdp_map_io,
        .init_early     = omap_4430sdp_init_early,
        .init_irq       = gic_init_irq,
        .init_machine   = omap_4430sdp_init,
        .timer          = &omap_timer,
MACHINE_END
```

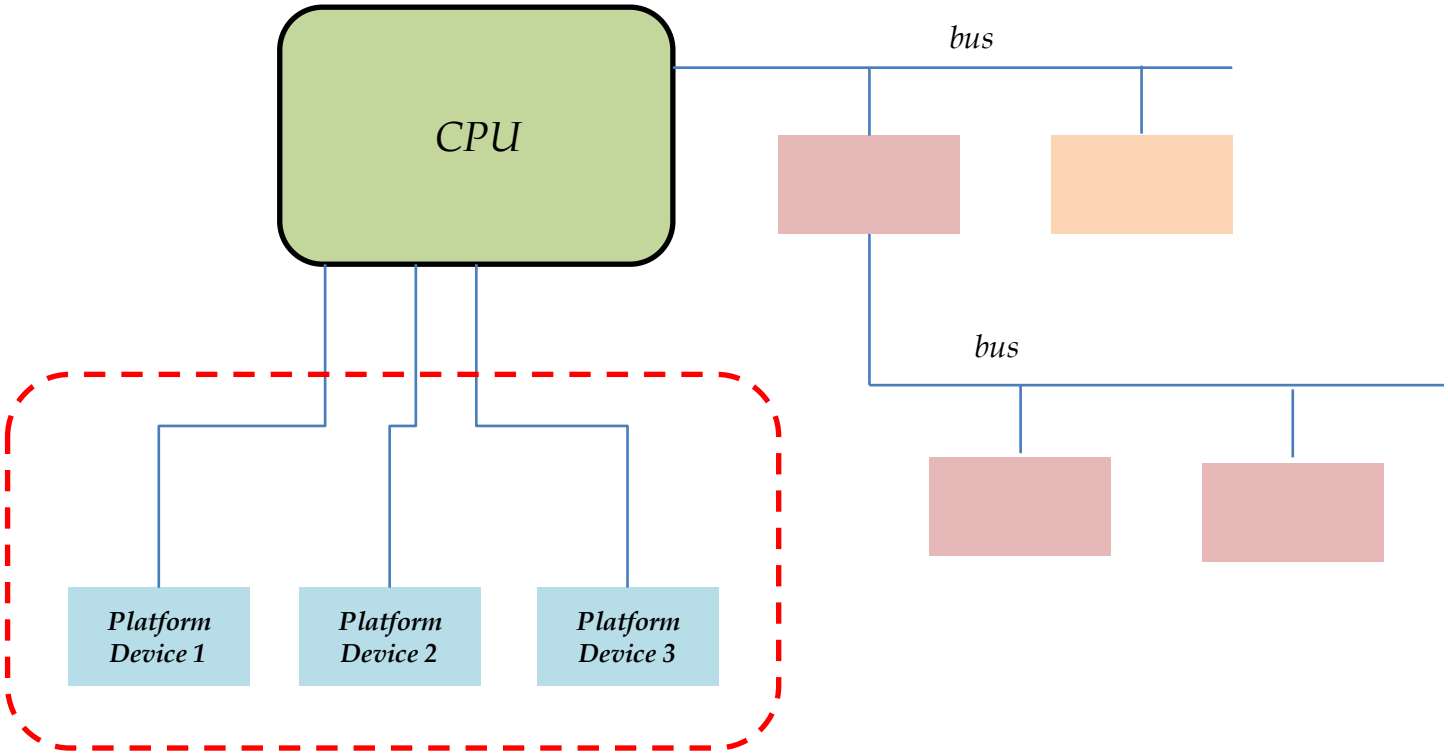
2. Board(Machine) initialization routine(2)

- **vmlinux.lds 파일 내용**
  => MACHINE_START에서 등록한 내용은 .arch.info.init와 연관 있으며,
  => devices.c에서 선언한 arch_initcall(omap2_init_devices); 등은 그 아래 INITCALLS에서 호출된다.

```
.init : {      /* Init code and data          */
            INIT_TEXT
        _einittext = .;
        __proc_info_begin = .;
            *(.proc.info.init)
        __proc_info_end = .;
        __arch_info_begin = .;
            *(.arch.info.init)
        __arch_info_end = .;
        __tagtable_begin = .;
            *(.taglist.init)
        __tagtable_end = .;
        . = ALIGN(16);
        __setup_start = .;
            *(.init.setup)
        __setup_end = .;
        __early_begin = .;
            *(.early_param.init)
        __early_end = .;
        __initcall_start = .;
            INITCALLS
        __initcall_end = .;

        .....
    }
```

3. Platform Device & Driver(1) - *개념*

- 1) Embedded system의 시스템의 경우, bus를 통해 device를 연결하지 않는 경우가 있음.
  ➔ *bus는 확장성(enumeration), hot-plugging, unique identifier를 허용함에도 불구하고 …*
- 2) platform driver/platform device infrastructure를 사용하여 이를 해결할 수 있음.
- ➔ *platform device란, 별도의 bus를 거치지 않고, CPU에 직접 연결되는 장치를 일컬음.*



3. Platform Device & Driver(2) - 개념

- platform_device 정의 및 초기화
- resource 정의

(arch/arm/mach-msm/board-XXXX.c 파일에 위치함)

<예 – bluetooth sleep device>

```
struct platform_device  my_bluesleep_device = {
    .name  = "bluesleep",
    .id      = 0,
    .num_resources = ARRAY_SIZE(bluesleep_resources),
    .resource = bluesleep_resources,
};
```

.name 필드("bluesleep")로 상호 연결(binding)

- platform_driver 정의 및 초기화
- probe/remove

(drivers/XXXX/xxxx.c 등에 위치함)

```
struct platform_driver  bluesleep_driver = {
   .remove = bluesleep_remove,
   .driver = {
       .name = "bluesleep",
       .owner = THIS_MODULE,
    },
};
```

(*) drivers/base/platform.c
(*) include/linux/platform_device.h 참조
(*) Documentation/driver-model/platform.txt 참조

3. Platform Device & Driver(3) – *platform device data structure*

다음 page 참조

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

    const struct platform_device_id *id_entry;

    /* arch specific additions */
    struct pdev_archdata    archdata;
};
```

```
struct resource {
    resource_size_t  start;
    resource_size_t  end;
    const char *name;
    unsigned long flags;
    struct resource *parent, *sibling,
*child;
};
```

```
struct platform_device_id {
    char
name[PLATFORM_NAME_SIZE];
    kernel_ulong_t driver_data

__attribute__((aligned(sizeof(kernel_ulo
ng_t))));
};
```

(*) 디바이스는 고유의 명칭(id)있는데, platform device의 경우는 platform_device.dev.bus_id가 device를 구분하는 값(canonical name)임.
(*) 이는 platform_device.name과 platform_device.id로 만들어지게 됨.

```
struct device {
    struct device       *parent;

    struct device_private    *p;

    struct kobject kobj;
    const char      *init_name; /* initial name of the device */
    struct device_type  *type;

    struct mutex        mutex;  /* mutex to synchronize calls to
                 * its driver.
                 */

    struct bus_type *bus;       /* type of bus device is on */
    struct device_driver *driver;   /* which driver has allocated
this
                   device */
    void        *platform_data; /* Platform specific data, device
                   core doesn't touch it */
    struct dev_pm_info  power;

#ifdef CONFIG_NUMA
    int     numa_node;  /* NUMA node this device is close to */
#endif
    u64     *dma_mask;  /* dma mask (if dma'able device) */
    u64     coherent_dma_mask;/* Like dma_mask, but for
                     alloc_coherent mappings as
                     not all hardware supports
                     64 bit addresses for consistent
                     allocations such descriptors. */

    struct device_dma_parameters *dma_parms;

    struct list_head    dma_pools;  /* dma pools (if dma'ble)
*/

    struct dma_coherent_mem *dma_mem; /* internal for
coherent mem
                     override */
    /* arch specific additions */
    struct dev_archdata archdata;
#ifdef CONFIG_OF
    struct device_node  *of_node;
#endif

    dev_t           devt;   /* dev_t, creates the sysfs "dev" */

    spinlock_t      devres_lock;
    struct list_head devres_head;

    struct klist_node knode_class;
    struct class  *class;
    const struct attribute_group  **groups;   /* optional
groups */

    void (*release)(struct device *dev);
};
```

3. Platform Device & Driver(4) – *platform driver data structure*

```
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;
};
```

```
struct device_driver {
    const char      *name;
    struct bus_type     *bus;

    struct module       *owner;
    const char      *mod_name;  /* used for built-in modules */

    bool suppress_bind_attrs;   /* disables bind/unbind via sysfs
*/

#if defined(CONFIG_OF)
    const struct of_device_id   *of_match_table;
#endif

    int (*probe) (struct device *dev);
    int (*remove) (struct device *dev);
    void (*shutdown) (struct device *dev);
    int (*suspend) (struct device *dev, pm_message_t state);
    int (*resume) (struct device *dev);
    const struct attribute_group **groups;

    const struct dev_pm_ops *pm;

    struct driver_private *p;
};
```

```
struct dev_pm_ops {
    int (*prepare)(struct device *dev);
    void (*complete)(struct device *dev);
    int (*suspend)(struct device *dev);
    int (*resume)(struct device *dev);
    int (*freeze)(struct device *dev);
    int (*thaw)(struct device *dev);
    int (*poweroff)(struct device *dev);
    int (*restore)(struct device *dev);
    int (*suspend_noirq)(struct device *dev);
    int (*resume_noirq)(struct device *dev);
    int (*freeze_noirq)(struct device *dev);
    int (*thaw_noirq)(struct device *dev);
    int (*poweroff_noirq)(struct device *dev);
    int (*restore_noirq)(struct device *dev);
    int (*runtime_suspend)(struct device *dev);
    int (*runtime_resume)(struct device *dev);
    int (*runtime_idle)(struct device *dev);
};
```

(*) platform driver와 power management간의 관계를
보기 위해 정리한 것임.
(*) suspend/resume callback 함수가 세군데나 있음.
드라이버 초기화시, 실제로 할당된 callback만이 사용될 것임^^

4. Platform Device(1)

- *(*) 플랫폼 디바이스는 동적으로 감지(detection)가 될 수 없으므로, static하게 지정해 주어야 함. static하게 지정하는 방식은 chip 마다 다를 수 있는데, ARM의 경우는 board specific code (arch/arm/mach-imx/mx1ads.c)에서 객체화 및 초기화(instantiation)를 진행하게 됨.*

- *(*) Platform 디바이스와 Platform 드라이버를 matching시키기 위해서는 name(아래의 경우는 "imx-uart")을 이용함.*

```
static struct platform_device imx_uart1_device = {
        .name           = "imx-uart",
        .id             = 0,
        .num_resources  = ARRAY_SIZE(imx_uart1_resources),
        .resource       = imx_uart1_resources,
        .dev = {
                .platform_data = &uart_pdata,
        }
};
```

4. Platform Device(2) - ***초기화***

- *(*) platform device는 아래 list에 추가되어야 함.*

```
static struct platform_device *devices[] __initdata = {
        &cs89x0_device,
        &imx_uart1_device,
        &imx_uart2_device,
};
```

- *(*) platform_add_devices() 함수를 통해서 실제로 시스템에 추가됨.*

```
static void __init mx1ads_init(void)
{
        [...]
        platform_add_devices(devices, ARRAY_SIZE(devices));
        [...]
}

MACHINE_START(MX1ADS, "Freescale MX1ADS")
        [...]
        .init_machine   = mx1ads_init,
MACHINE_END
```

4. Platform Device(3) - ***resource***

- *(*) 특정 드라이버가 관리하는 각 장치(device)는 서로 다른 H/W 리소스를 사용하게 됨.*
  ➔ I/O 레지스터 주소, DMA 채널, IRQ line 등이 서로 상이함.

  (*) 이러한 정보는 struct resource data structure를 사용하여 표현되며, 이들 resource 배열은 platform device 정의 부분과 결합되어 있음.
- *(*) platform driver내에서 platform_device 정보(pointer)를 이용하여 resource를 얻어오기 위해서는 platform_get_resource_byname(...) 함수가 사용될 수 있음.*

```
static struct resource imx_uart1_resources[] = {
        [0] = {
                .start  = 0x00206000,
                .end    = 0x002060FF,
                .flags  = IORESOURCE_MEM,
        },
        [1] = {
                .start  = (UART1_MINT_RX),
                .end    = (UART1_MINT_RX),
                .flags  = IORESOURCE_IRQ,
        },
};
```

4. Platform Device(4) - ***device specific data***

- *(*) 앞서 설명한 resource data structure 외에도, 드라이버에 따라서는 자신만의 환경 혹은 데이터(configuration)을 원할 수 있음.이는 struct platform_device 내의 platform_data를 사용하여 지정 가능함.*
  *(*) platfor_data는 void * pointer로 되어 있으므로, 드라이버에 임의의 형식의 데이타 전달이 가능함.*
  (*) iMX 드라이버의 경우는 struct imxuart platform data가 platform_data로 사용되고 있음.

```
static struct imxuart_platform_data uart_pdata = {
        .flags = IMXUART_HAVE_RTSCTS,
};
```

5. Platform Driver(1)

- *(*) drivers/serial/imx.c file에 있는 iMX serial port driver를 예로써 소개하고자 함. 이 드라이버는 platform_driver structure를 초기화함.*

```
static struct platform_driver serial_imx_driver = {
        .probe          = serial_imx_probe,
        .remove         = serial_imx_remove,
        .driver         = {
                .name   = "imx-uart",
                .owner  = THIS_MODULE,
        },
};
```

- *(*) init/cleanup시, register/unregister 하기*

```
static int __init imx_serial_init(void)
{
        platform_driver_register(&serial_imx_driver);
}
static void __ext imx_serial_cleanup(void)
{
        platform_driver_unregister(&serial_imx_driver);
}
```

5. Platform Driver(2) – *probe, remove*

- *(*) 보통의 probe함수 처럼, 인자로 platform_device에의 pointer를 넘겨 받으며, 관련 resource를 찾기 위해 다른 utility 함수를 사용하고, 상위 layer로 해당 디바이스를 등록함. 한편 별도의 그림으로 표현하지는 않았으나, probe의 반대 개념으로 드라이버 제거 시에는 remove 함수가 사용됨.*

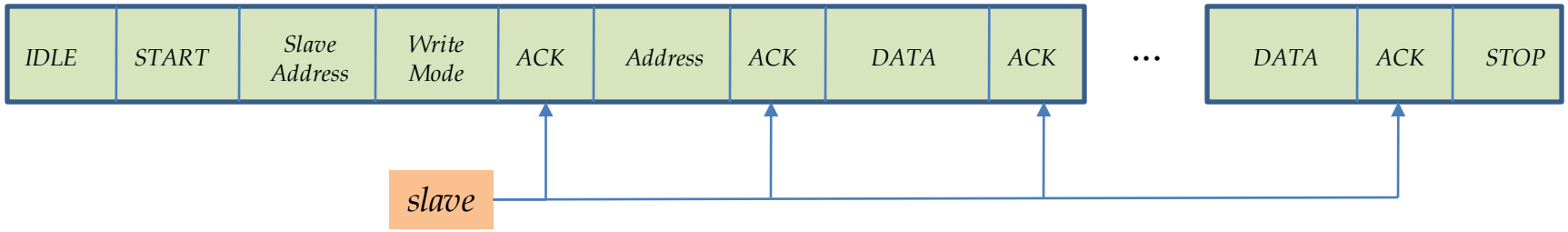
```
static int serial_imx_probe(struct platform_device *pdev)
{
        struct imx_port *sport;
        struct imxuart_platform_data *pdata;
        void __iomem *base;
        struct resource *res;

        sport = kzalloc(sizeof(*sport), GFP_KERNEL);
        res = platform_get_resource(pdev, IORESOURCE_MEM, 0);
        base = ioremap(res->start, PAGE_SIZE);

        sport->port.dev = &pdev->dev;
        sport->port.mapbase = res->start;
        sport->port.membase = base;
        sport->port.type = PORT_IMX,
        sport->port.iotype = UPIO_MEM;
        sport->port.irq = platform_get_irq(pdev, 0);
        sport->rxirq = platform_get_irq(pdev, 0);
        sport->txirq = platform_get_irq(pdev, 1);
        sport->rtsirq = platform_get_irq(pdev, 2);

        [...]
```

부록 2

: i2c, spi, usb APIs

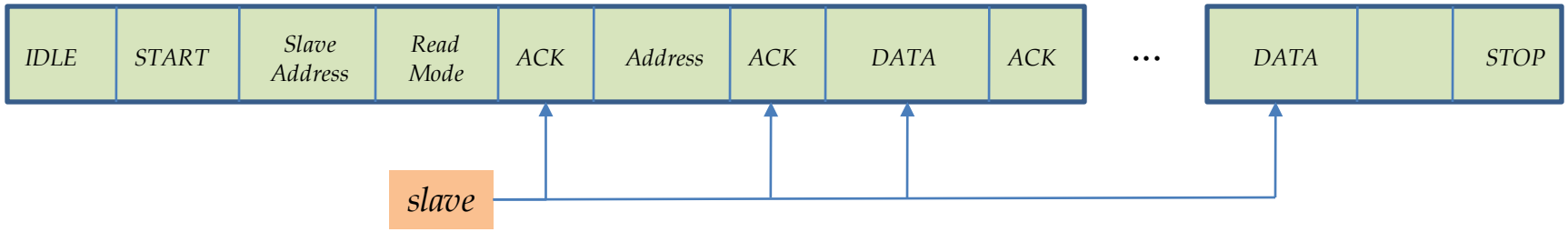
1. i2c kernel API(1)

- **I2C data write**
- *1) Master signals a START condition.*
- *2) Master sends the address of the slave it wishes to send data to and sends write mode of transfer.*
- *3) Slave sends an acknowledgement to the master.*
- *4) Master sends the address where the data has to be written on the slave device.*
- *5) Slave sends an acknowledgment to the master.*
- *6) Master sends data to be written on the SDA bus.*
- *7) At the end of the byte transfer, the slave sends an acknowledgment bit.*
- *8) The above two steps are again performed until all the required bytes are written. The write address is automatically incremented.*
- *9) Master signals a STOP condition.*



1. i2c kernel API(2)

- **I2C data read**
- *1) Master signals a START condition.*
- *2) Master sends the address of the slave it wishes to send data to and sends the mode of transfer to read.*
- *3) Slave sends an acknowledgement to the master.*
- *4) Master sends the address from where the data has to be read on the slave device.*
- *5) Slave sends an acknowledgment to the master.*
- *6) Slave sends the data to be read on the SDA bus.*
- *7) At the end of the byte transfer, the master sends an acknowledgment bit.*
- *8) The above two steps are again performed until all the required bytes are written. The read address is automatically incremented. However, for the last byte the master does not send an acknowledgment. This prevents the slave from sending any more data on the bus.*
- *9) Master signals a STOP condition.*

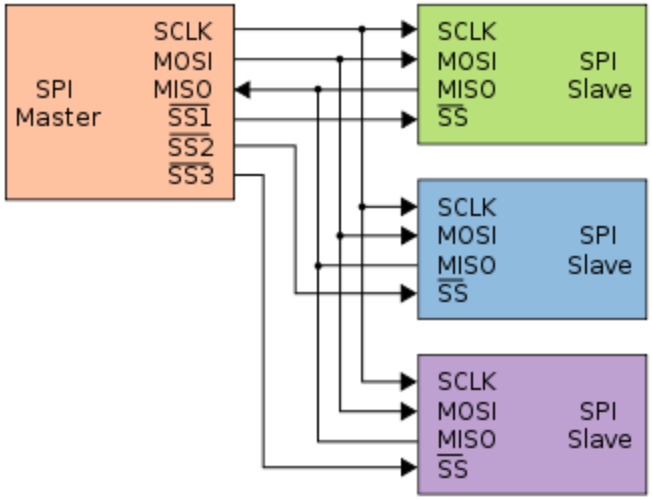


1. i2c kernel API(3)

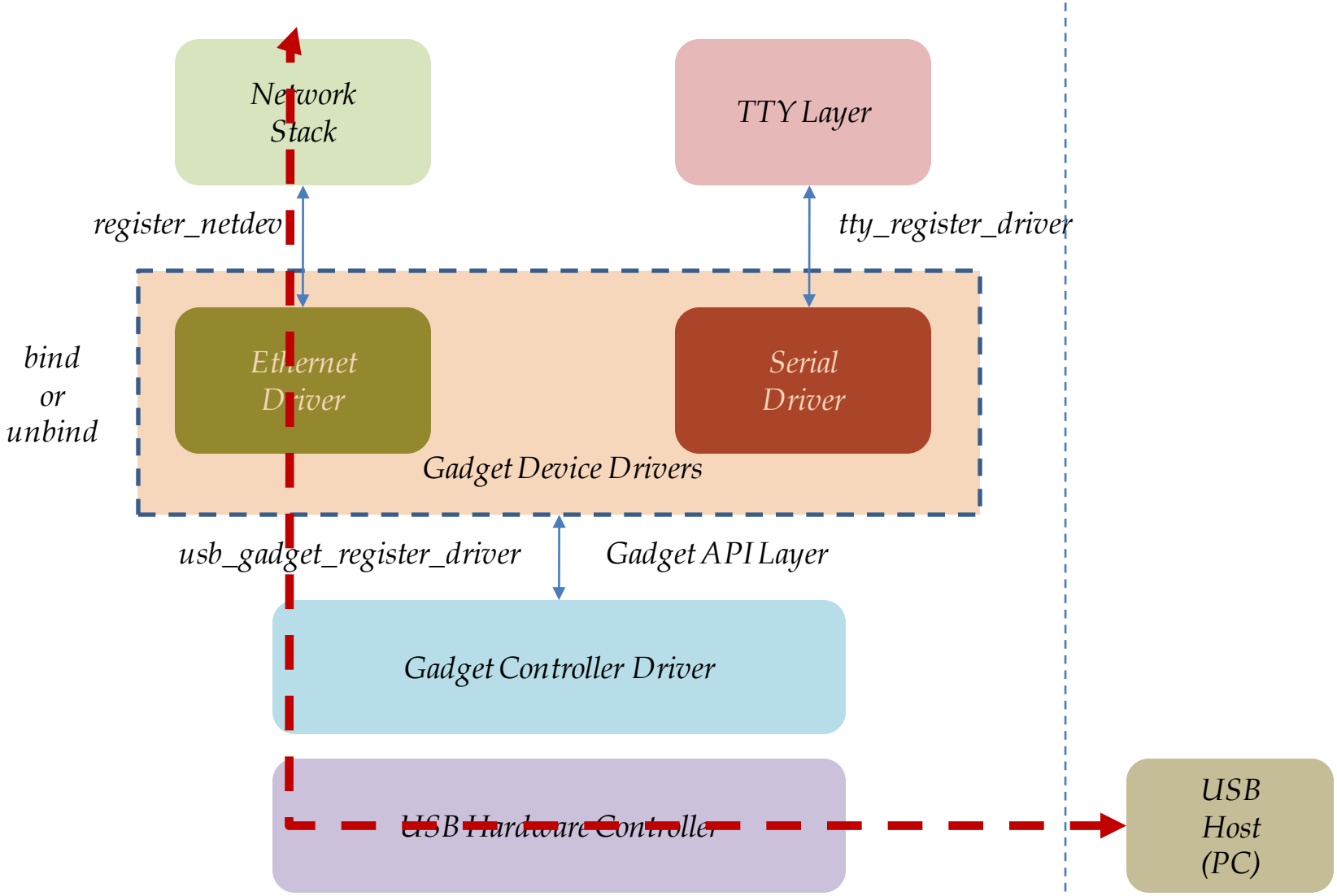
- *i2c_add_driver()*
- *i2c_del_driver()*
- *i2c_probe()*
- *i2c_attach_client()*
- *i2c_detach_client()*
- *i2c_check_functionality()*
- *i2c_get_functionality()*
- *i2c_add_adapter()*
- *i2c_del_adapter()*
- *i2c_transfer()*
- …

2. SPI kernel API

- *spi_register_driver()*
- *spi_unregister_driver()*
- *spi_message_init()*
- *spi_message_add_tail()*
- *spi_sync()*
- *spi_async()*
- …



3. USB API(1) - gadget driver



(*) USB gadget은 host가 아닌 device로 동작하는 방식임 !

3. USB API(2)

- *usb_register()*
- *usb_deregister()*
- *usb_set_intfdata()*
- *usb_get_intfdata()*
- *usb_register_dev()*
- *usb_deregister_dev()*

- *usb_alloc_urb()*
- *usb_fill_[control | int | bulk]_urb()*
- *usb_[control | interrupt | bulk]_msg()*
- *usb_submit_urb()*
- *usb_free_urb()*
- *usb_unlink_urb()*
- *usb_[rcv | snd][ctrl | int | bulk | isoc]pipe()*
- *usb_find_interface()*

- *usb_buffer_alloc()*
- *usb_buffer_free()*
- …

4. DMA(Direct Memory Access)

- *1) LCD controller내에서 사용*

  ➔ *FrameBuffer 내용이 LCD panel로 빠르게 전송되게 하기 위하여 사용*
- *2) I2S interface에서 사용*

  ➔ *memory에 존재하는 audio sound data와 audio codec 간에 빠른 data 전송이 가능하도록 하기 위해 사용*
- *3) UART -> HS-UART*

  ➔ *UART data를 빠르게 전송시키기 위하여 사용(memory <-> device)*

- *(*) 이 밖에도 memory와 device 간의 빠른 data 전송을 위해 다양한 곳에서 DMA가 사용되고 있다.*

References

- *1) Essential Linux Device Drivers …… [Sreekrishnan Venkateswaran]*
- *2) Android_Device_Driver_Guide_simple.pdf …….. [이충한]*
- *3) Android_ICS_Porting_Guide.pdf …….. [이충한]*
- *4) A Dynamic Voltage and Current Regulator Control Interface for the Linux Kernel ..... [Liam Girdwood]*
- *5) 안드로이드와 디바이스드라이버 적용 기법 … [FALINUX 유영창]*
- *6) Wi-Fi P2P 기술 분석(Understanding Wi-Fi P2P Technical Specification) … [ETRI]*
- *7) Wi-Fi Direct 기술 파급효과와 시사점 ……... [KT 종합기술원]*
- *8) Embedded Linux System Design and Development … [P.Raghavan]*
- *9) Some Internet Articles …*

Thanks a lot !

SlowBoot