Android Device Driver Collage

: LCD, Touchscreen, Sensors, Wi-Fi, Bluetooth, NFC…

chunghan.yi@gmail.com, slowboot

본 문서에서는 …

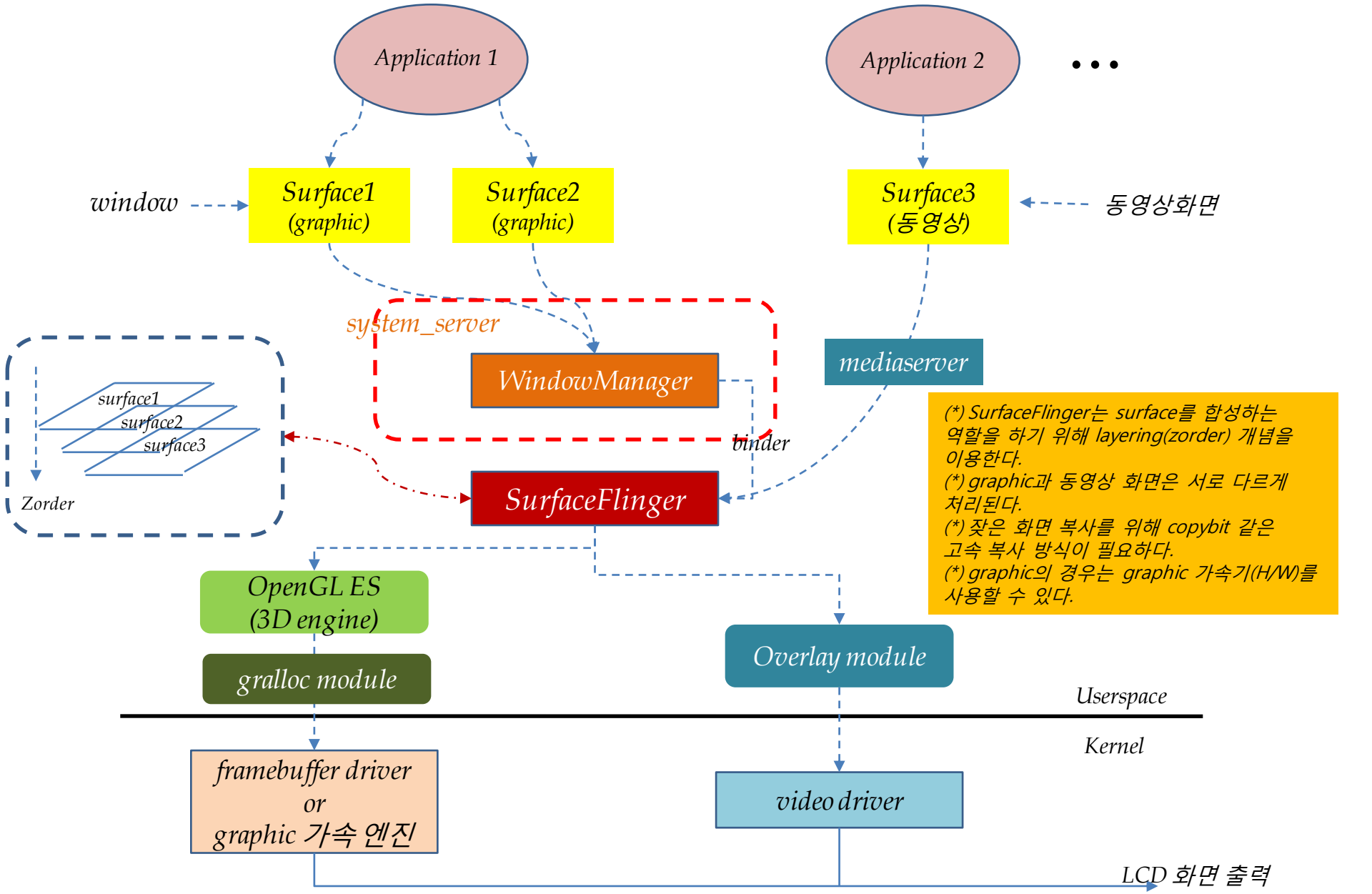
- *Smart Phone Device Driver를 구성하는 여러 내용 중, 아래 4), 5)번 부분을 중점적으로 소개하고자 한다.*

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

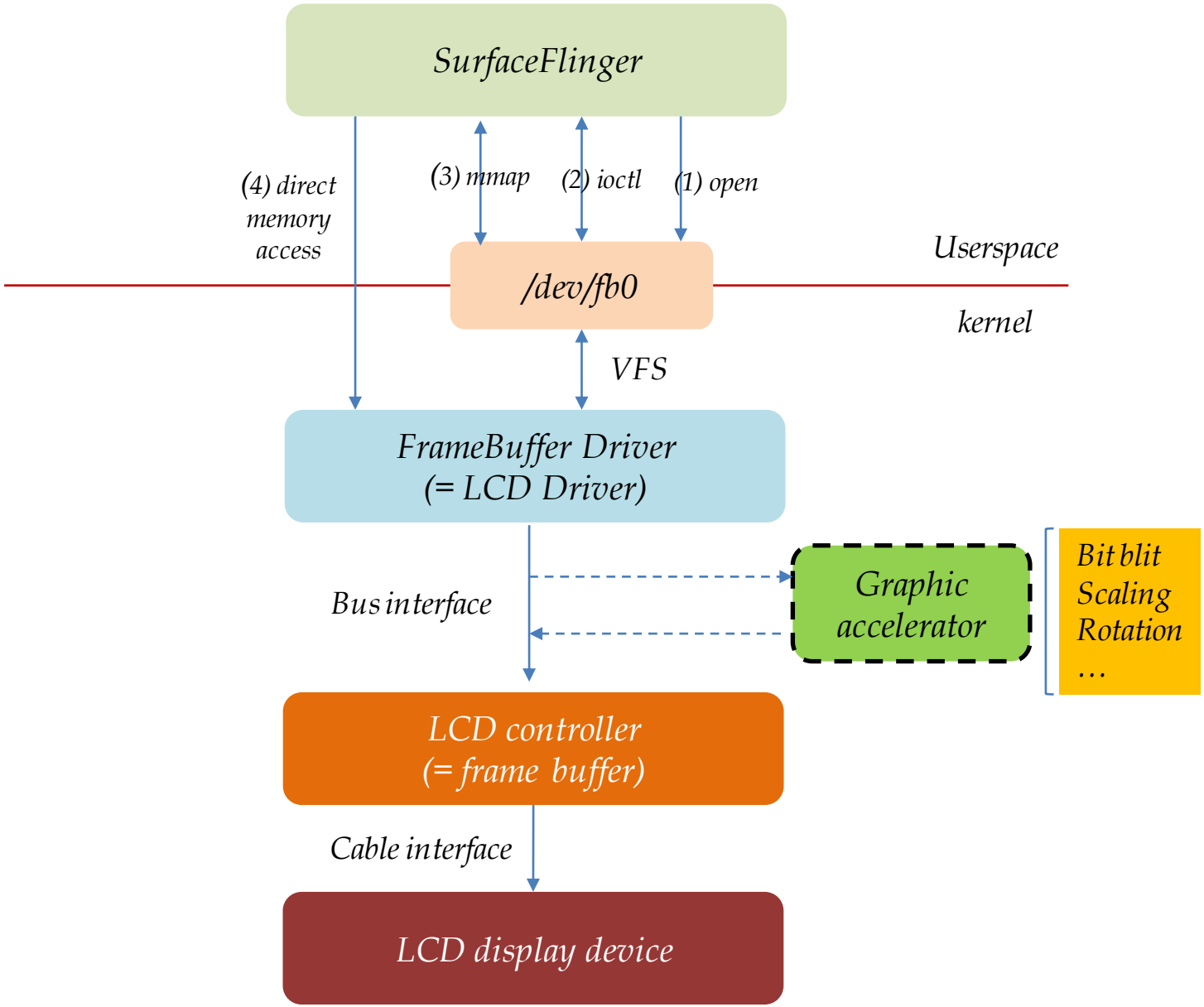
목차

1. LCD Driver(FrameBuffer Driver)

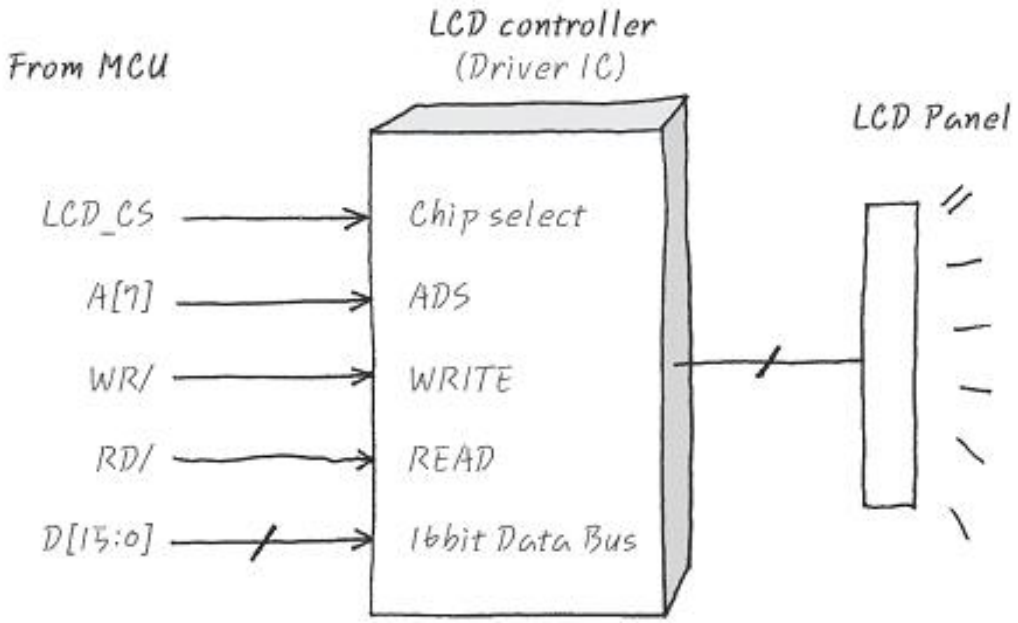
1. Android Display Overview



2. Framebuffer Driver & LCD Controller(1): ***Overview***



2. Framebuffer Driver & LCD Controller(2): ***LCD Controller***



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

3. FrameBuffer API(1)

<LCD 정보>
(*) x-resolution, x-resolution(virtual)
(*) y-resolution, y-resolution(virtual)
(*) bpp(bit per pixel)
(*) length of framebuffer memory

```
struct fb_var_screeninfo {
    __u32 xres;             /* Visible resolution in the X axis */
    __u32 yres;             /* Visible resolution in the Y axis */
    /* ... */
    __u32 bits_per_pixel;   /* Number of bits required to hold a
                               pixel */
    /* ... */
    __u32 pixclock;         /* Pixel clock in picoseconds */
    __u32 left_margin;      /* Time from sync to picture */
    __u32 right_margin;     /* Time from picture to sync */
    /* ... */
    __u32 hsync_len;        /* Length of horizontal sync */
    __u32 vsync_len;        /* Length of vertical sync */
    /* ... */
};
```

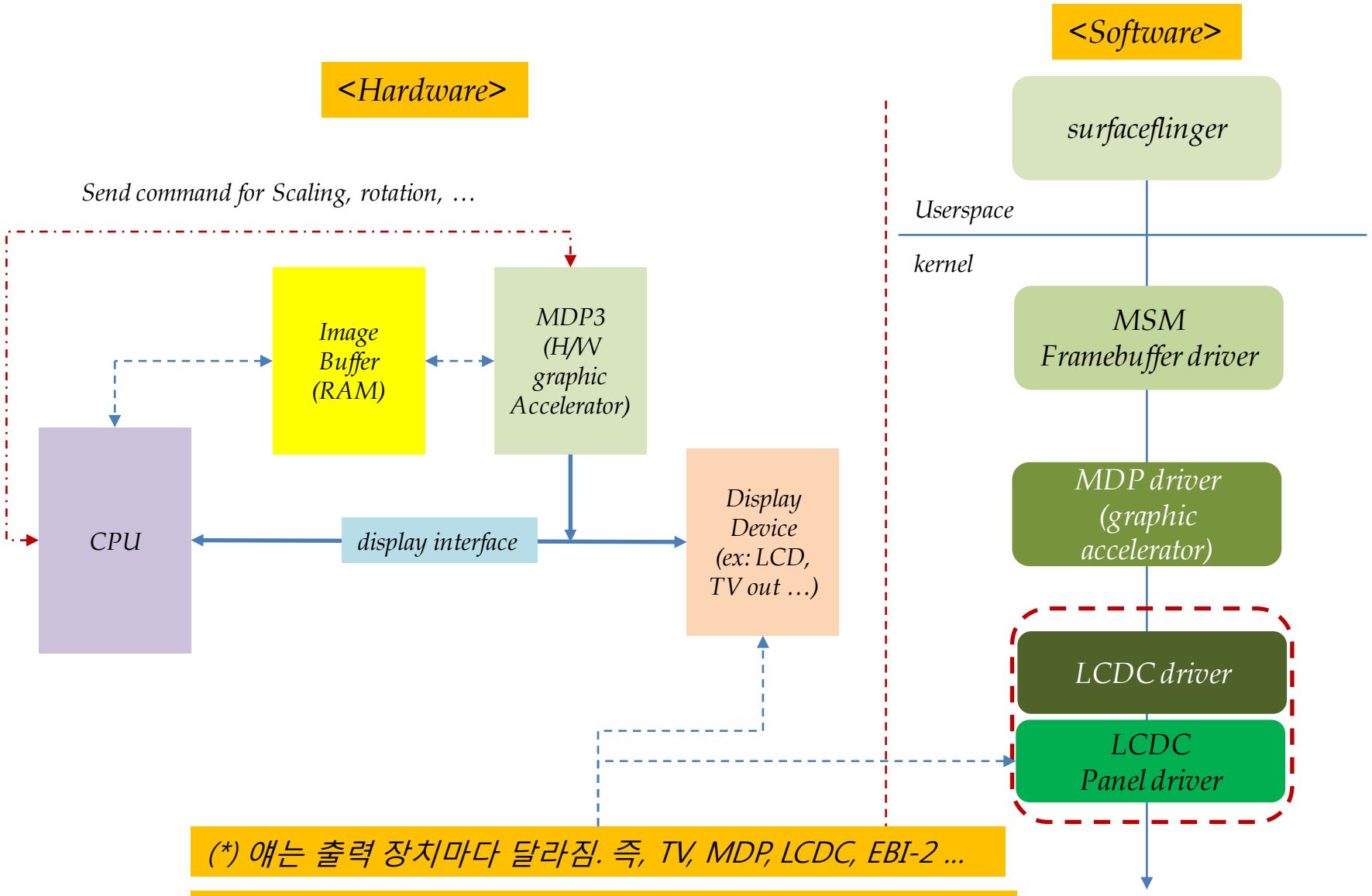
```
struct fb_fix_screeninfo {
    char id[16];                /* Identification string */
    unsigned long smem_start;   /* Start of frame buffer memory */
    __u32 smem_len;             /* Length of frame buffer memory */
    /* ... */
};
```

```
struct fb_info {
    kdev_t node;
    int flags;
    int open;
    struct fb_var_screeninfo var;
    struct fb_fix_screeninfo fix;
    struct fb_monspecs monspecs;
    struct fbcursor cursor;
    struct fb_cmap cmap;
    struct fb_ops *fbops;
    struct pm_dev *pm_fb;
    char *screen_base;
    wait_queue_head_t wait;
    devfs_handle_t devfs_handle;
    devfs_handle_t devfs_lhandle;
    void *pseudo_palette;
#ifdef CONFIG_MTRR
    int mtrr_handle;
#endif
    void *par;
}
```

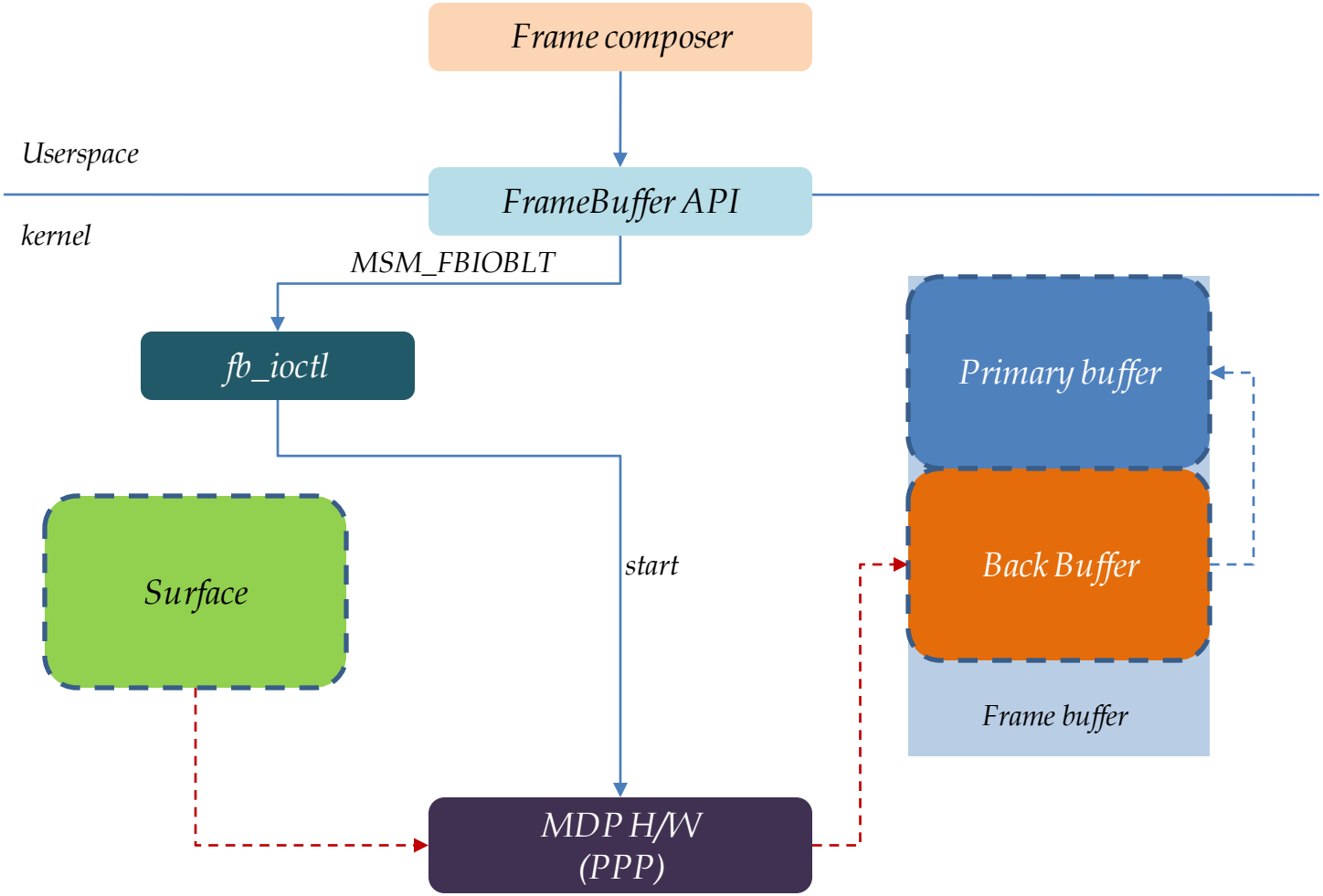
3. FrameBuffer API(2)

```
struct fb_ops {
  struct module *owner;
  /* Driver open */
  int (*fb_open)(struct fb_info *info, int user);
  /* Driver close */
  int (*fb_release)(struct fb_info *info, int user);
  /* ... */
  /* Sanity check on video parameters */
  int (*fb_check_var)(struct fb_var_screeninfo *var,
                      struct fb_info *info);
  /* Configure the video controller registers */
  int (*fb_set_par)(struct fb_info *info);
  /* Create pseudo color palette map */
  int (*fb_setcolreg)(unsigned regno, unsigned red,
                unsigned green, unsigned blue,
                unsigned transp, struct fb_info *info);
  /* Blank/unblank display */
  int (*fb_blank)(int blank, struct fb_info *info);
  /* ... */
  /* Accelerated method to fill a rectangle with pixel lines */
  void (*fb_fillrect)(struct fb_info *info,
                      const struct fb_fillrect *rect);
  /* Accelerated method to copy a rectangular area from one
     screen region to another */
  void (*fb_copyarea)(struct fb_info *info,
                      const struct fb_copyarea *region);
  /* Accelerated method to draw an image to the display */
  void (*fb_imageblit)(struct fb_info *info,
                       const struct fb_image *image);
  /* Accelerated method to rotate the display */
  void (*fb_rotate)(struct fb_info *info, int angle);
  /* Ioctl interface to support device-specific commands */
  int (*fb_ioctl)(struct fb_info *info, unsigned int cmd,
                  unsigned long arg);
  /* ... */
};
```

4. Qualcomm Framebuffer Driver with MDP(Graphic Accelerator)(1)



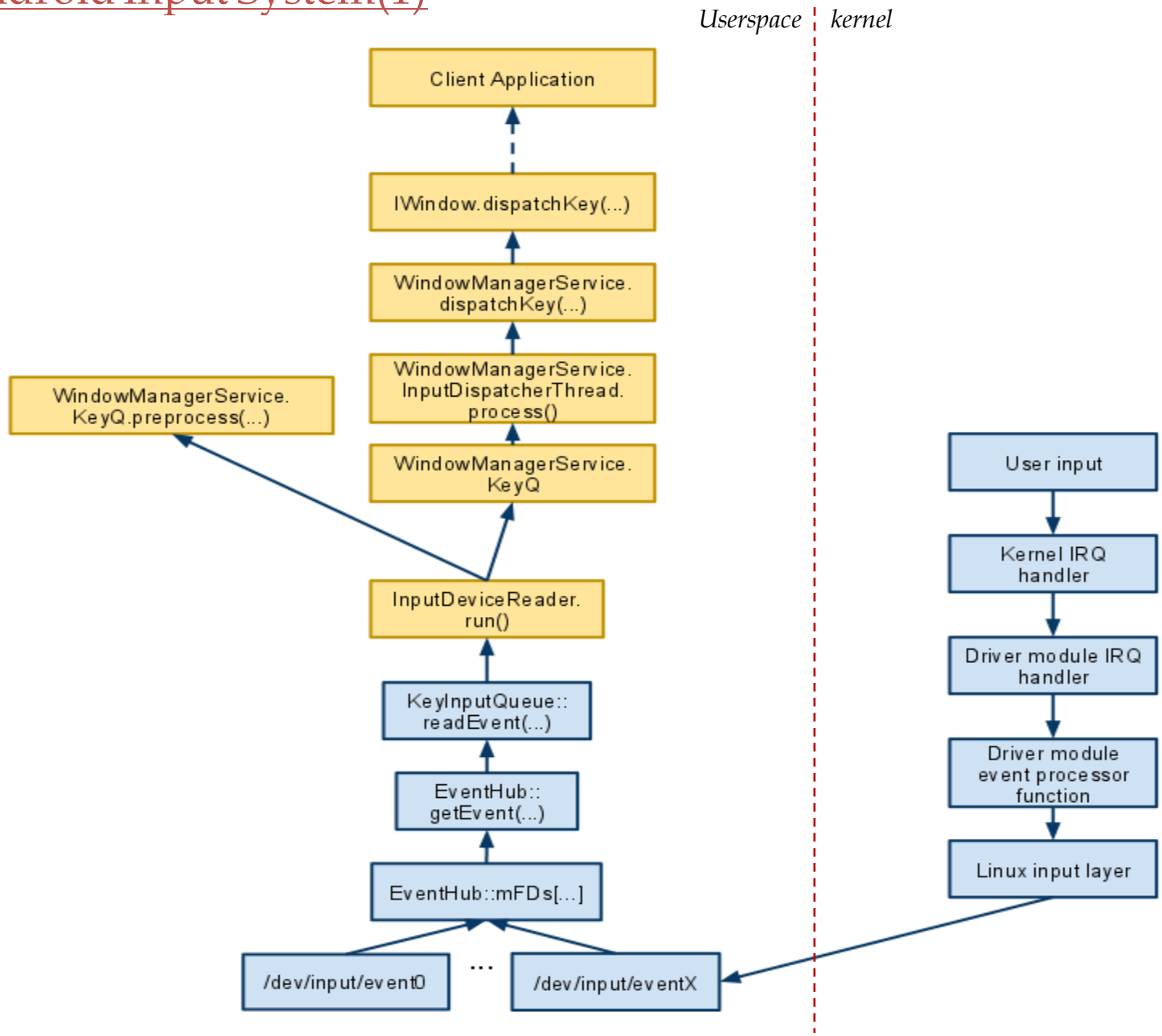
4. Qualcomm Framebuffer Driver with MDP(Graphic Accelerator)(2)



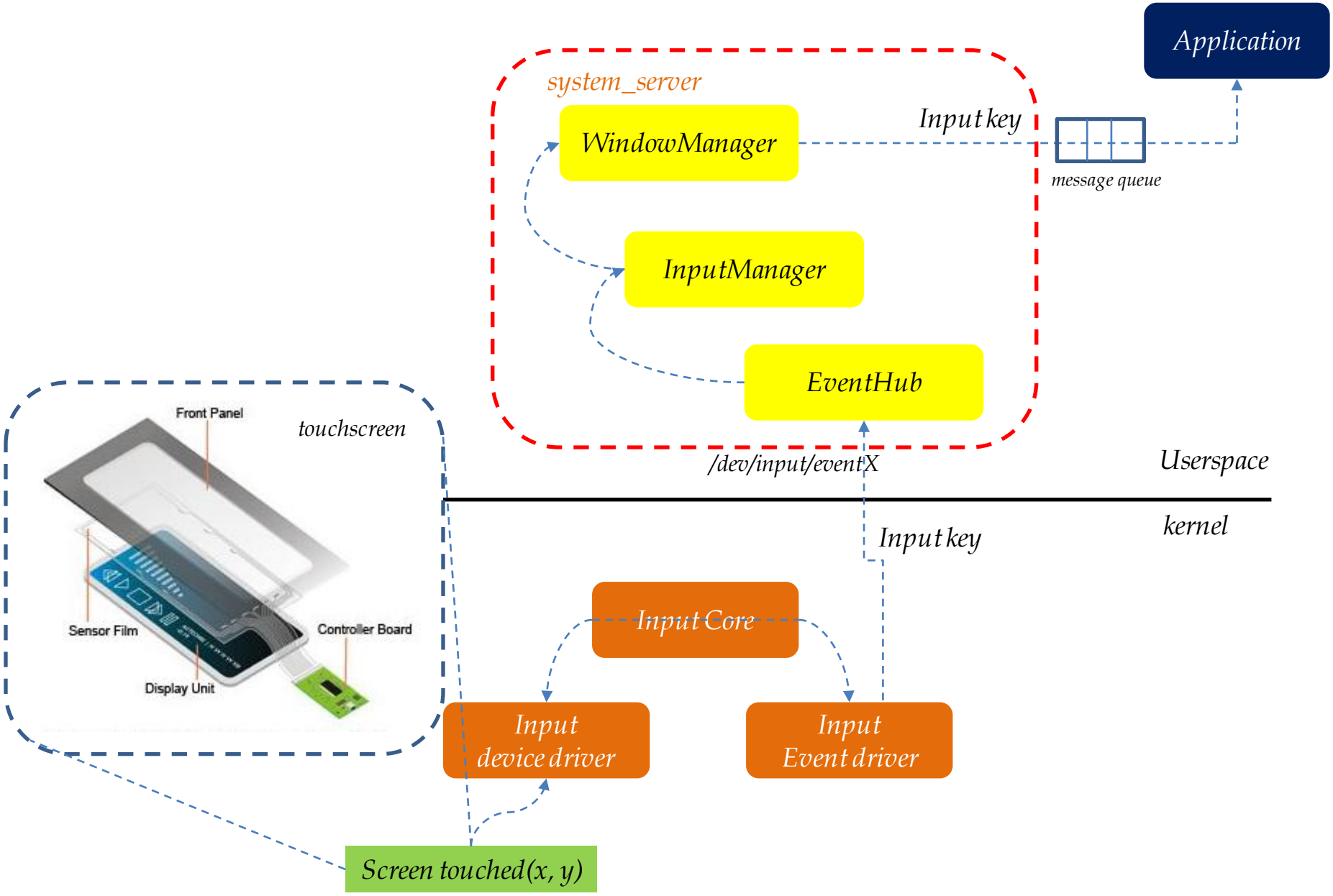
(*) MDP HW를 통해 bit blit등의 hardware graphic 가속 처리가 이루어진다.
(*) 화면 떨림을 없애기 위해 framebuffer 상에 back buffer를 두고 있으며, 모든 surface는 일차적으로 MDP를 거쳐 back buffer로 출력되고, 최종적으로 merge된 화면은 다시 primary buffer로 출력되는 과정을 통해 화면에 보여지게 된다.
(*) framebuffer 관련 기본 operation은 위에서 보는 바와 같이 fb_ioctl() 함수를 통해 이루어 진다.

2. Touchscreen Driver

1. Android Input System(1)

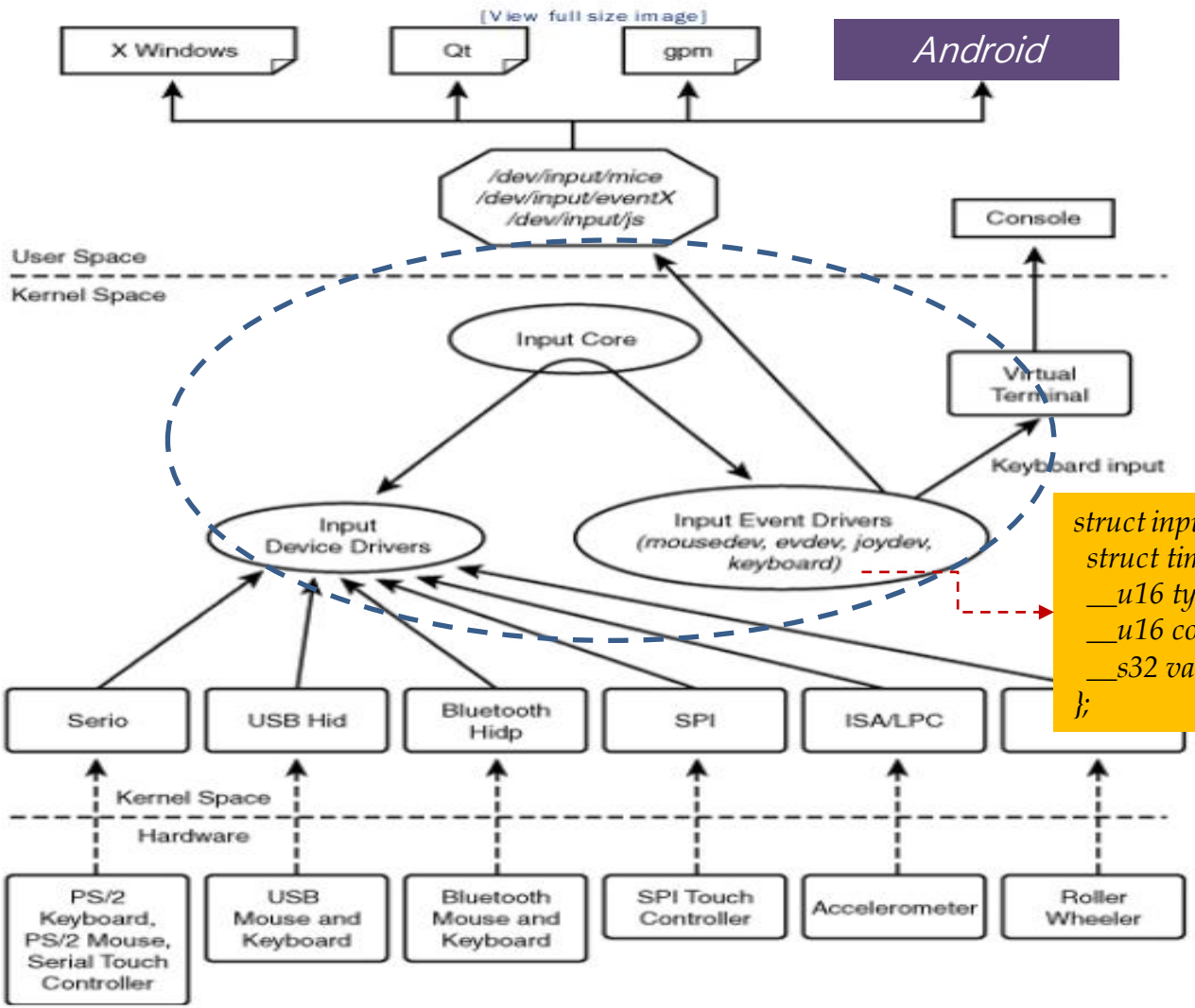


1. Android Input System(2)



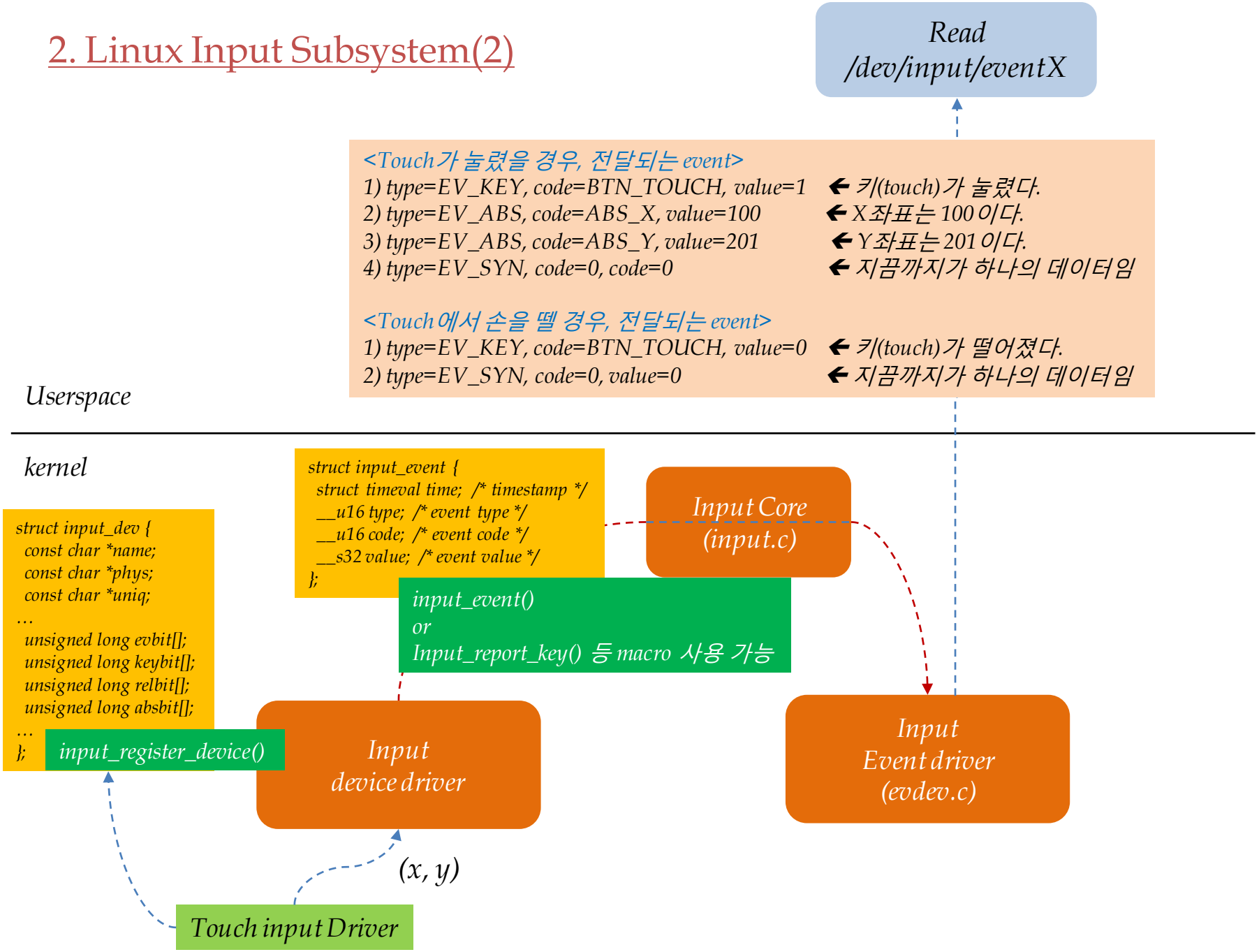
2. Linux Input Subsystem(1)

(*) 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)

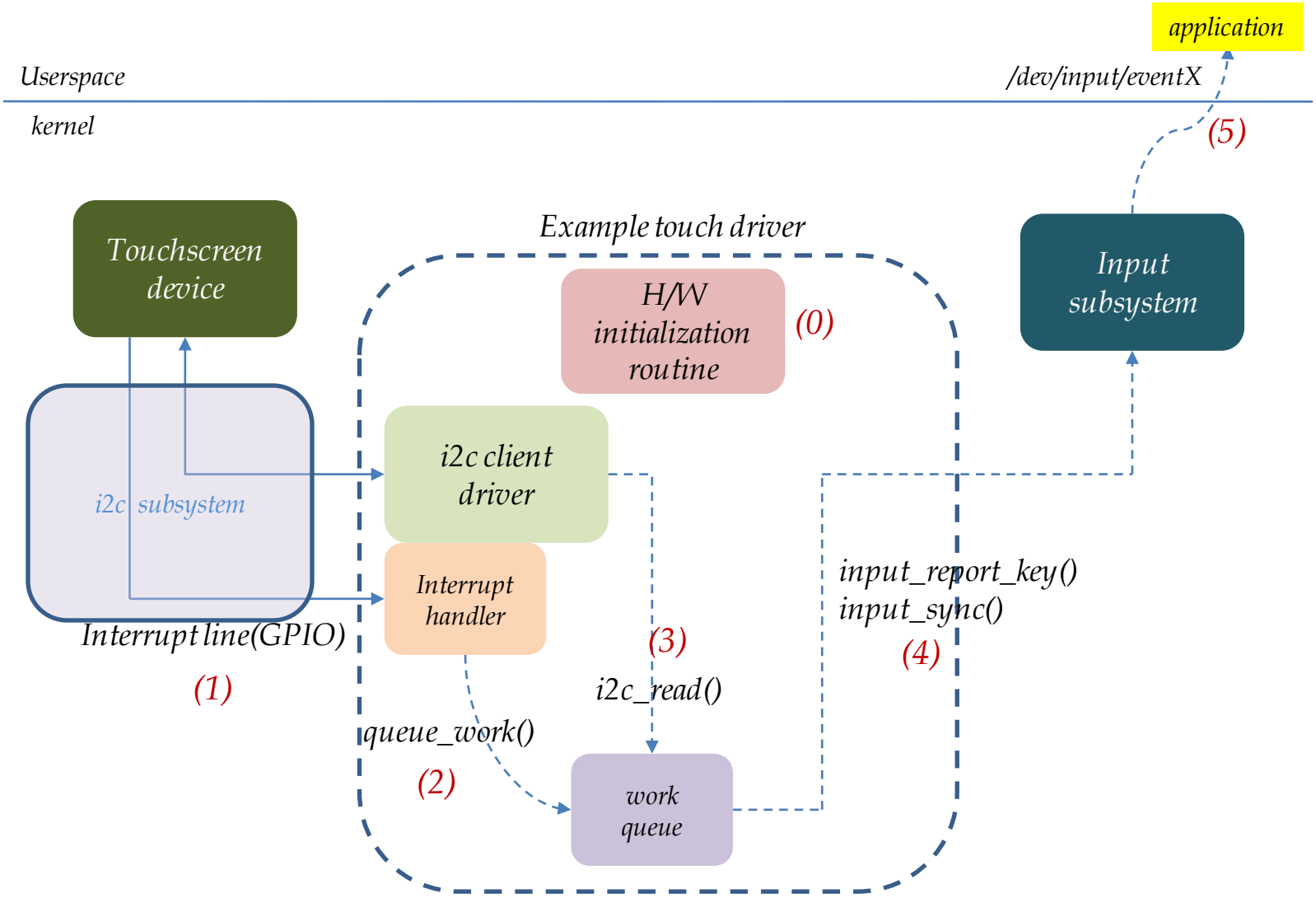
static inline void input_report_rel(struct input_dev *dev, unsigned int code, int value);
내부적으로 input_event 함수를 이용하여 이동 된 크기 이벤트를 전달한다.

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.

3. Example Touch Driver(1)



3. Example Touch Driver(2)

<Touch Input Flow>

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

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.

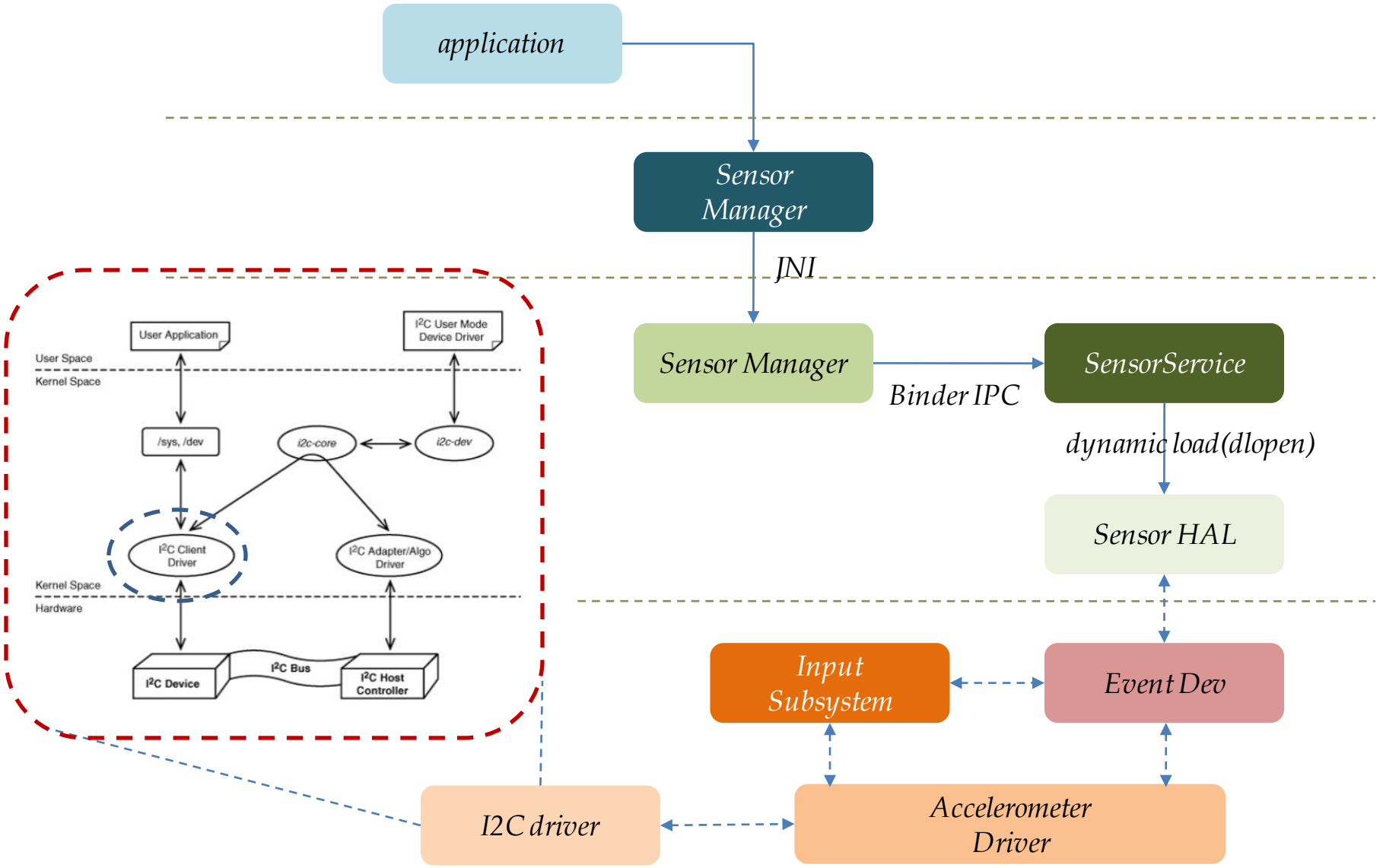
3) Work queue routine은 i2c_read( ) 함수를 이용하여 i2c로 입력된 정보(touch 좌표 정보)를 읽어 들인다.

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. Sensor Drivers

1. Android Sensor Architecture: ***Overview***



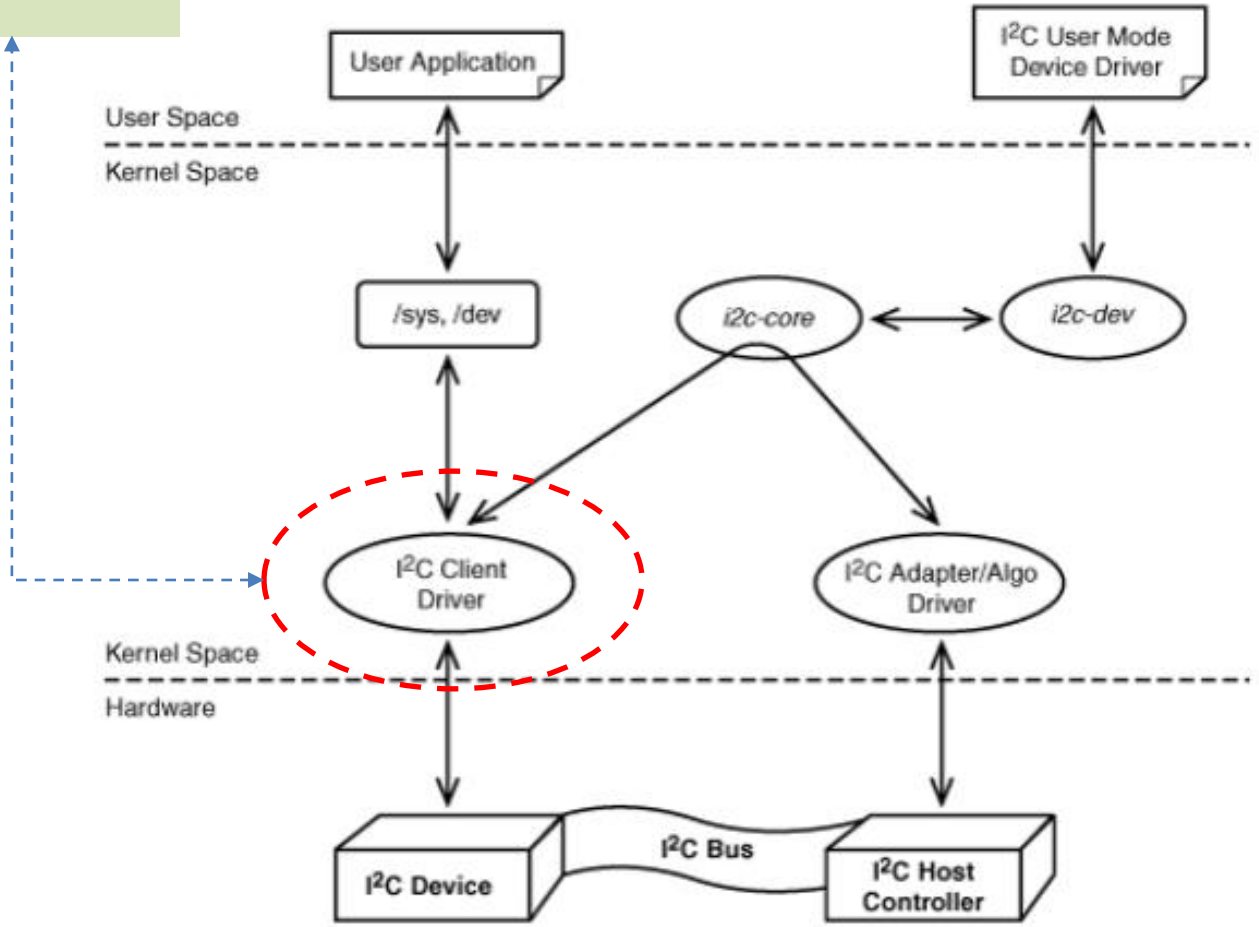
2. Sensor 종류

- *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*

3. Sensor I2C Interface: *I2C Client Driver*

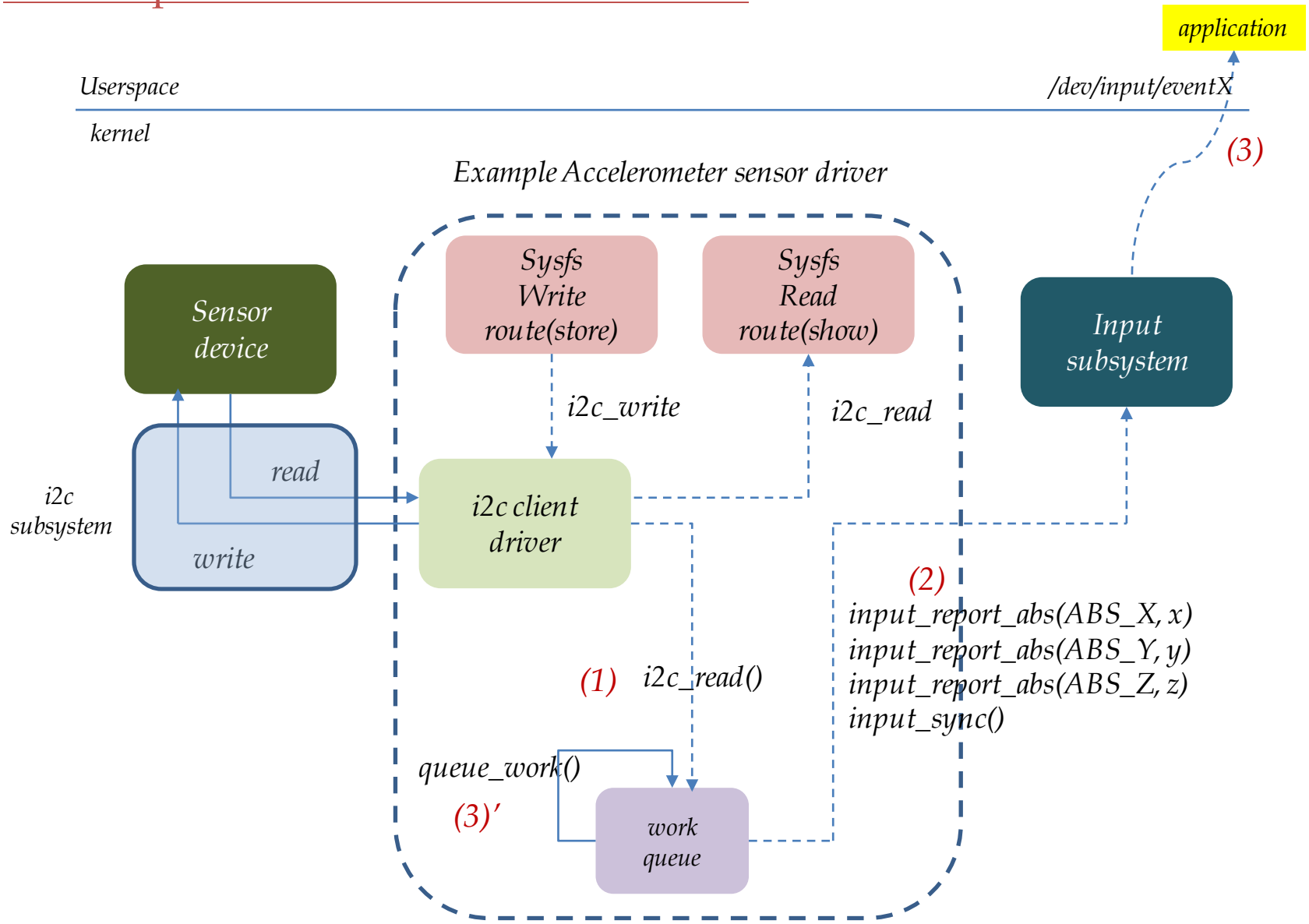
1) Acceleration Sensor
2) Geomagnetic Sensor
3) Proximity Sensor
...

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

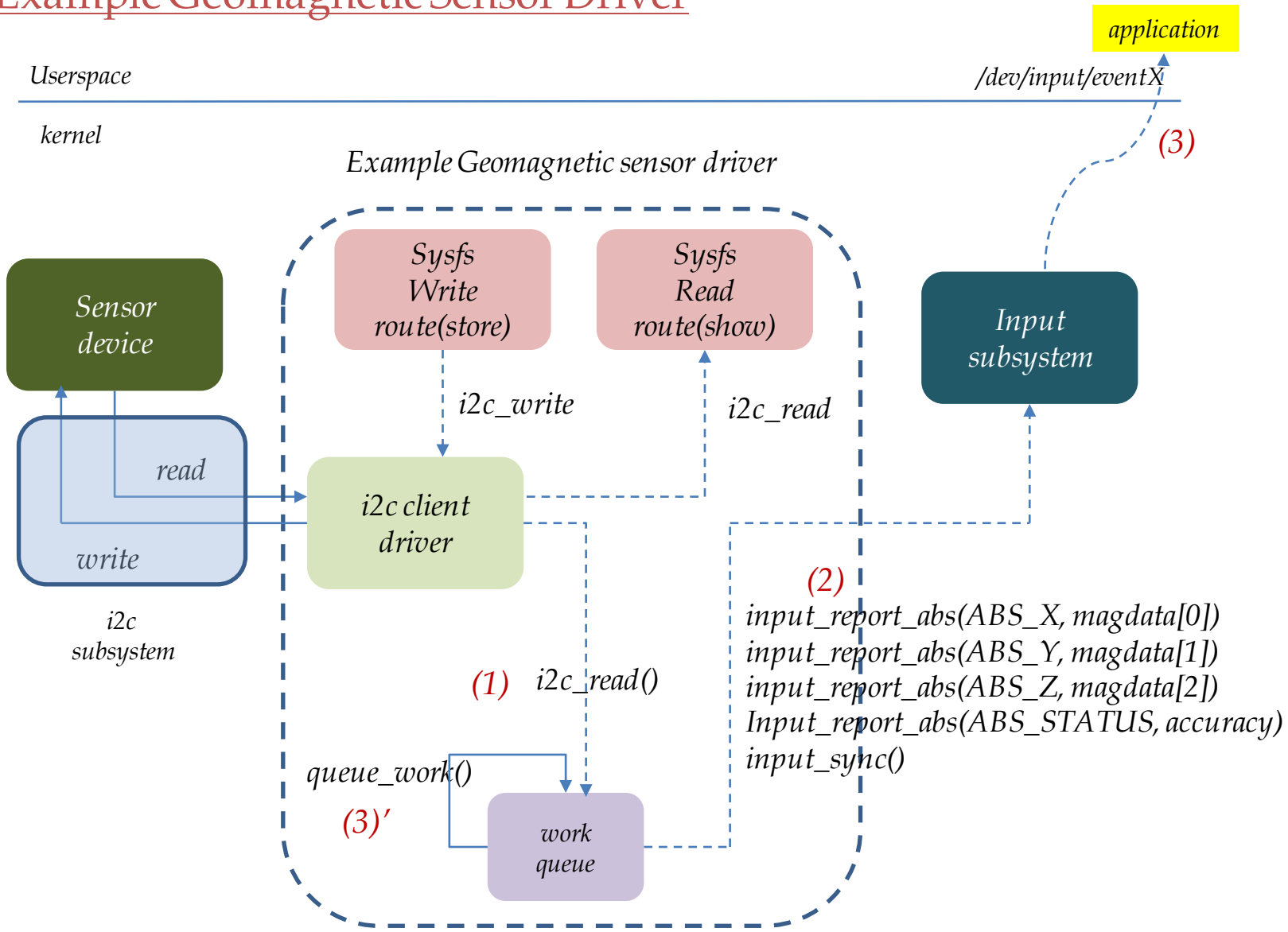


(*) 위 그림은 참고 문서[1]에서 복사해 온 것임.

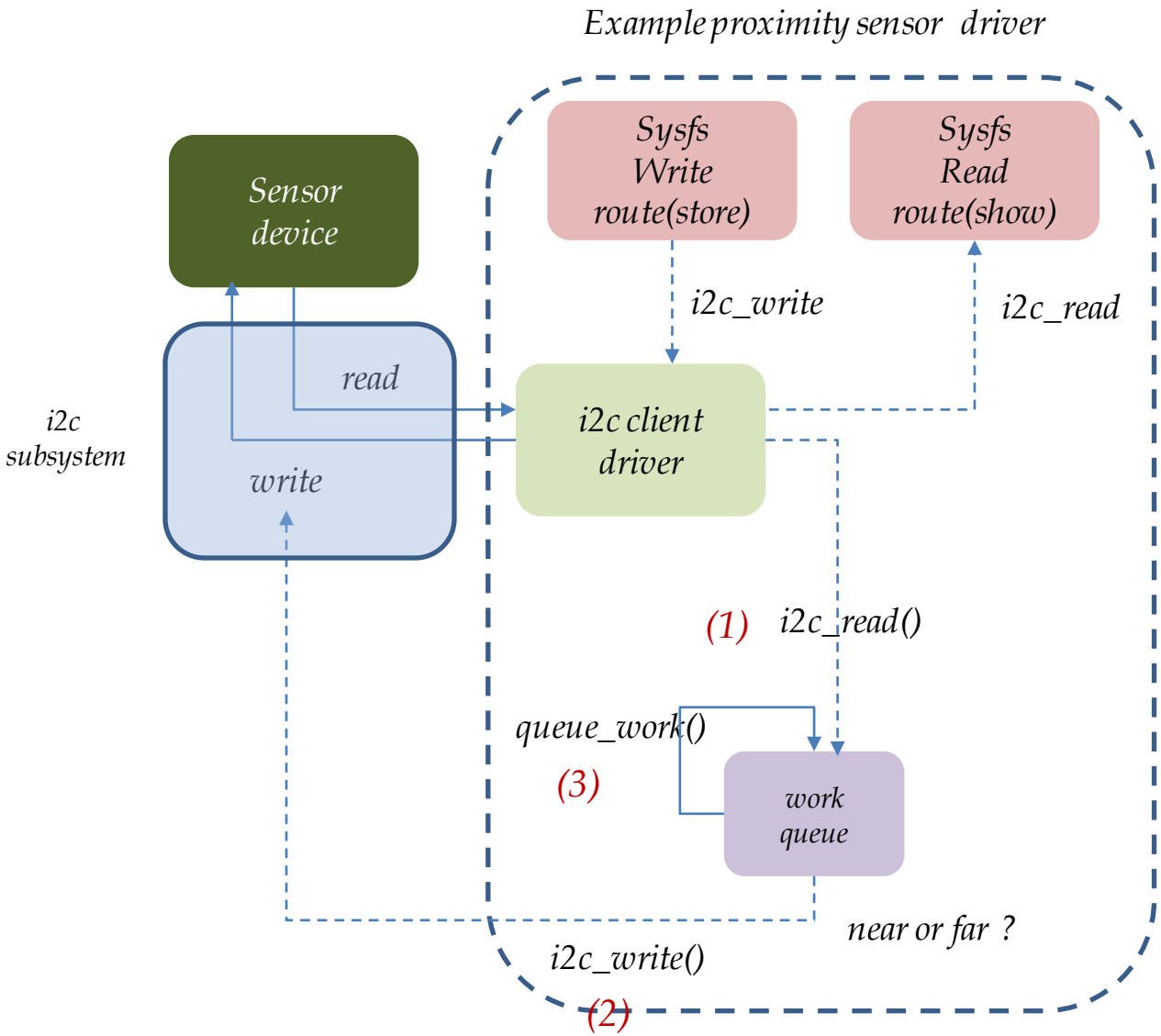
4. Example Accelerometer Sensor Driver



5. Example Geomagnetic Sensor Driver



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



4. Vibrator Driver

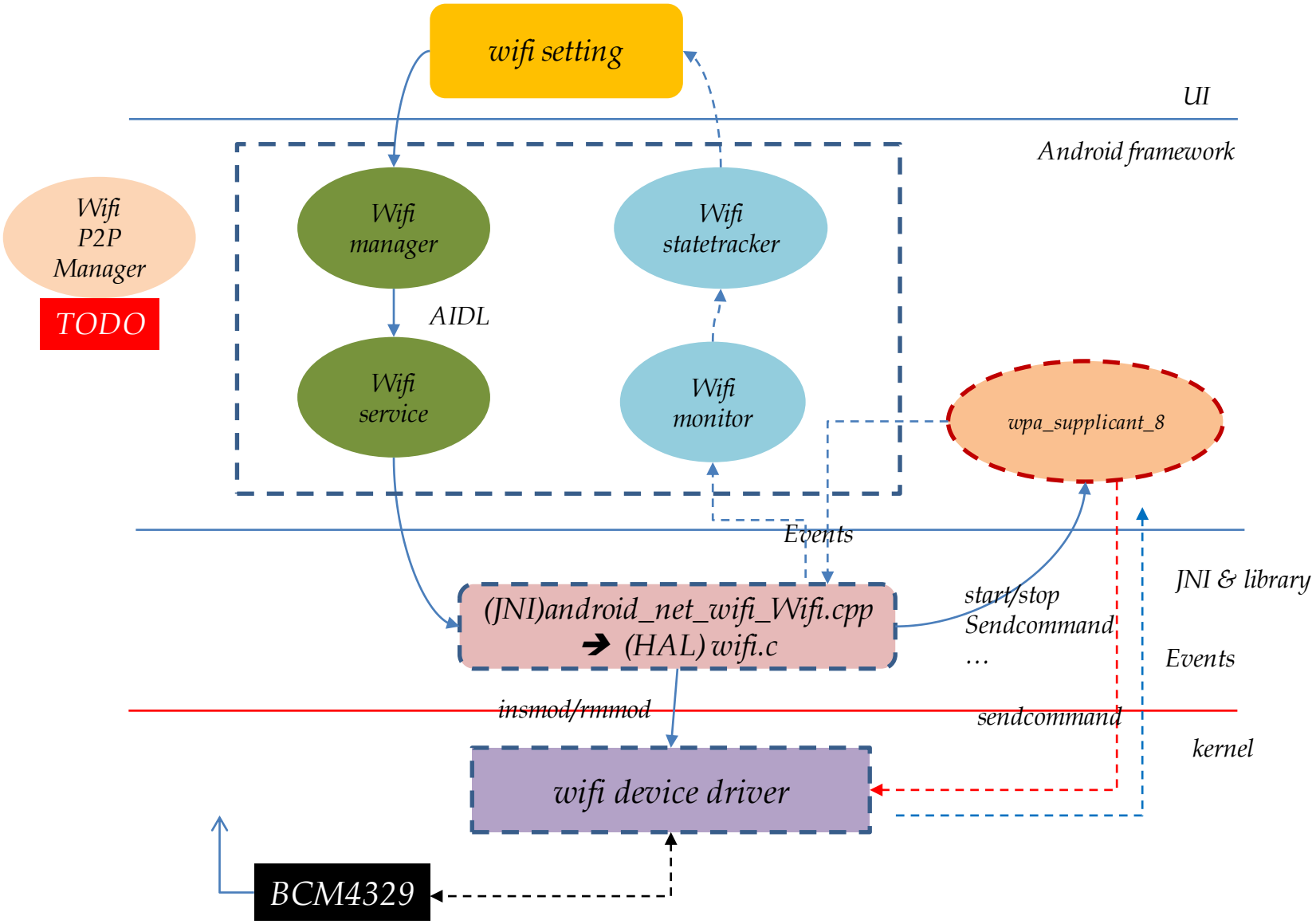
1. Vibrator Driver Overview

- ***<TODO>***

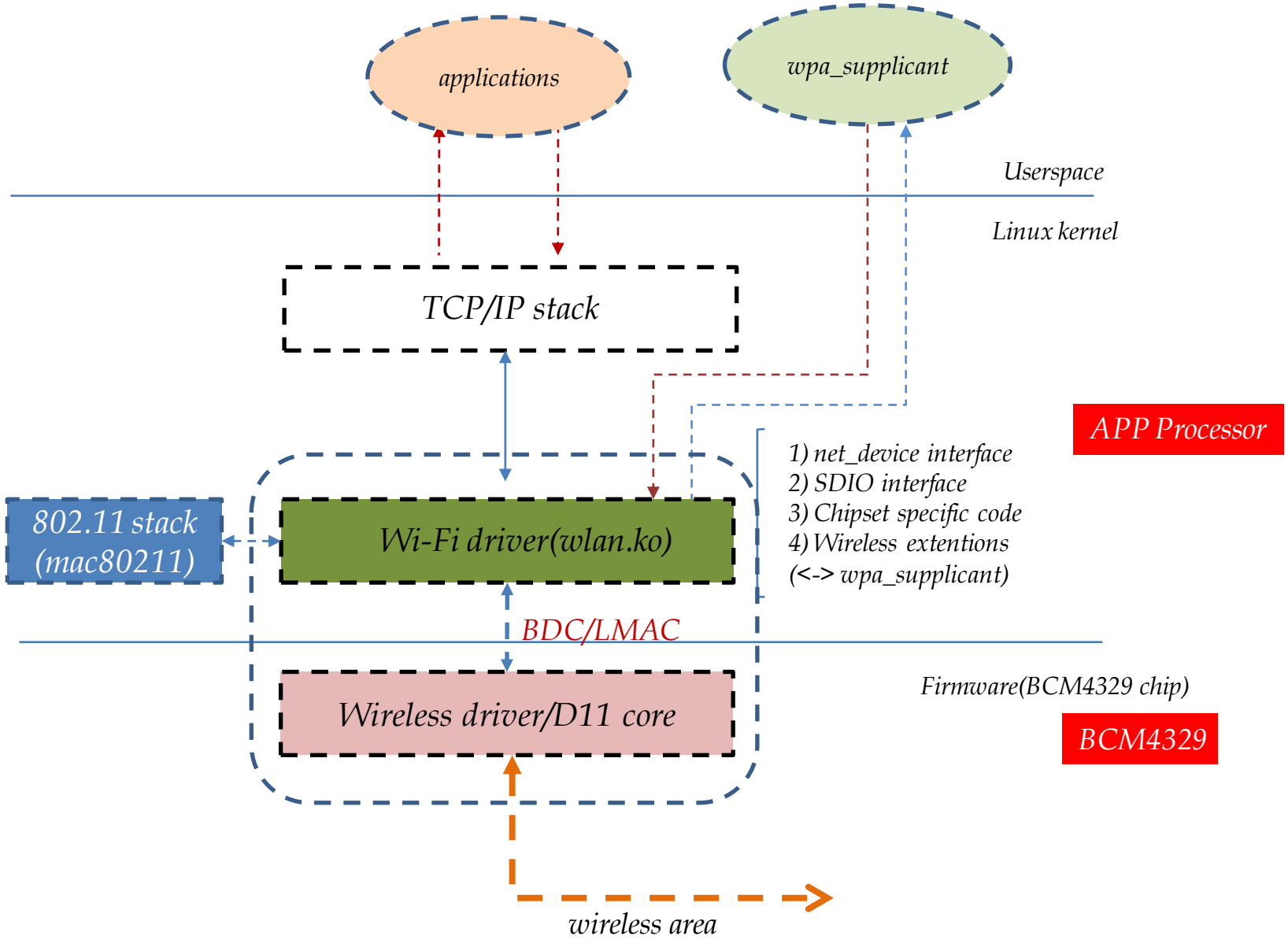
5. Wi-Fi Driver

based on BCM4329

1. Android Wi-Fi Overview

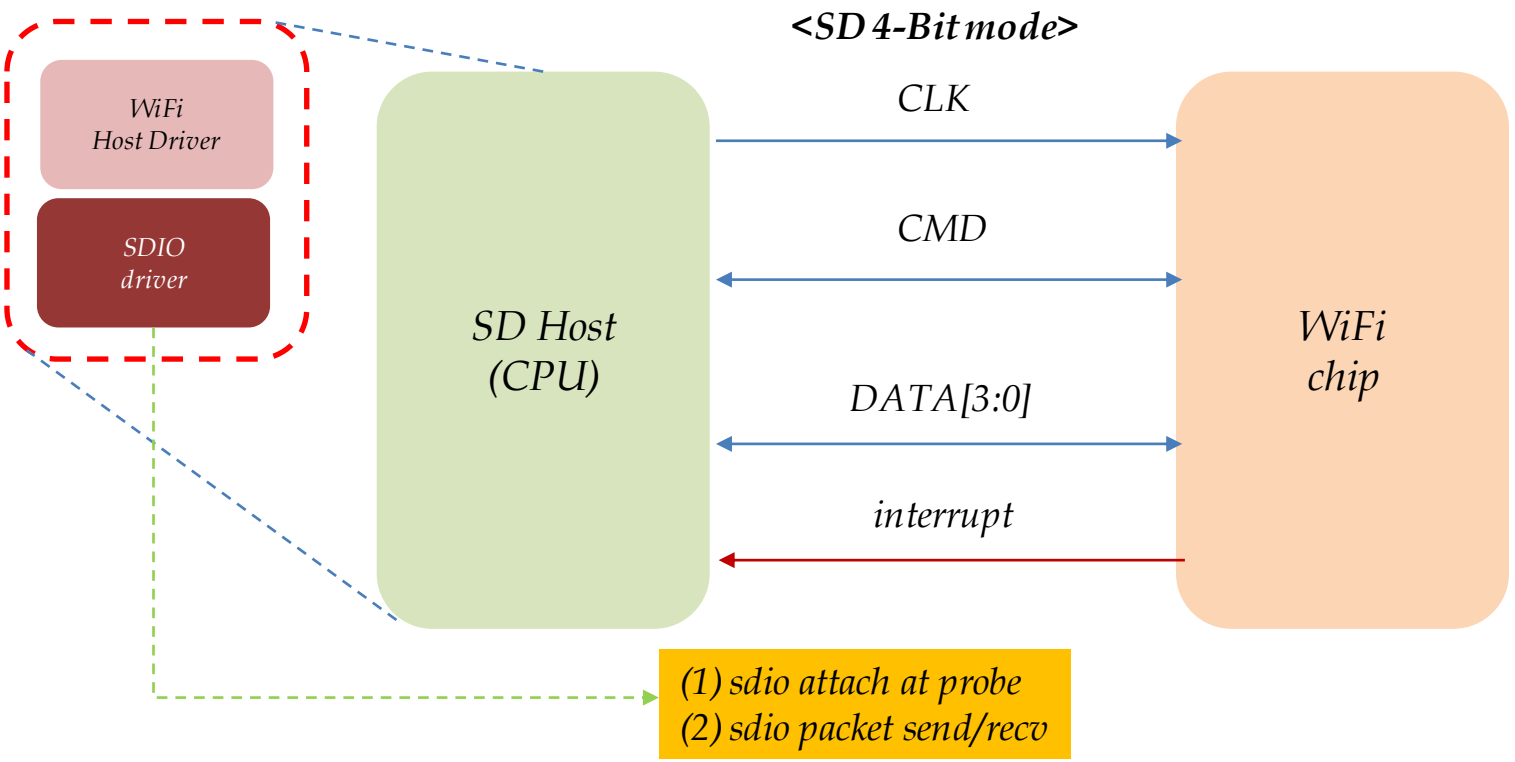


2. Wi-Fi 드라이버 구조(1)

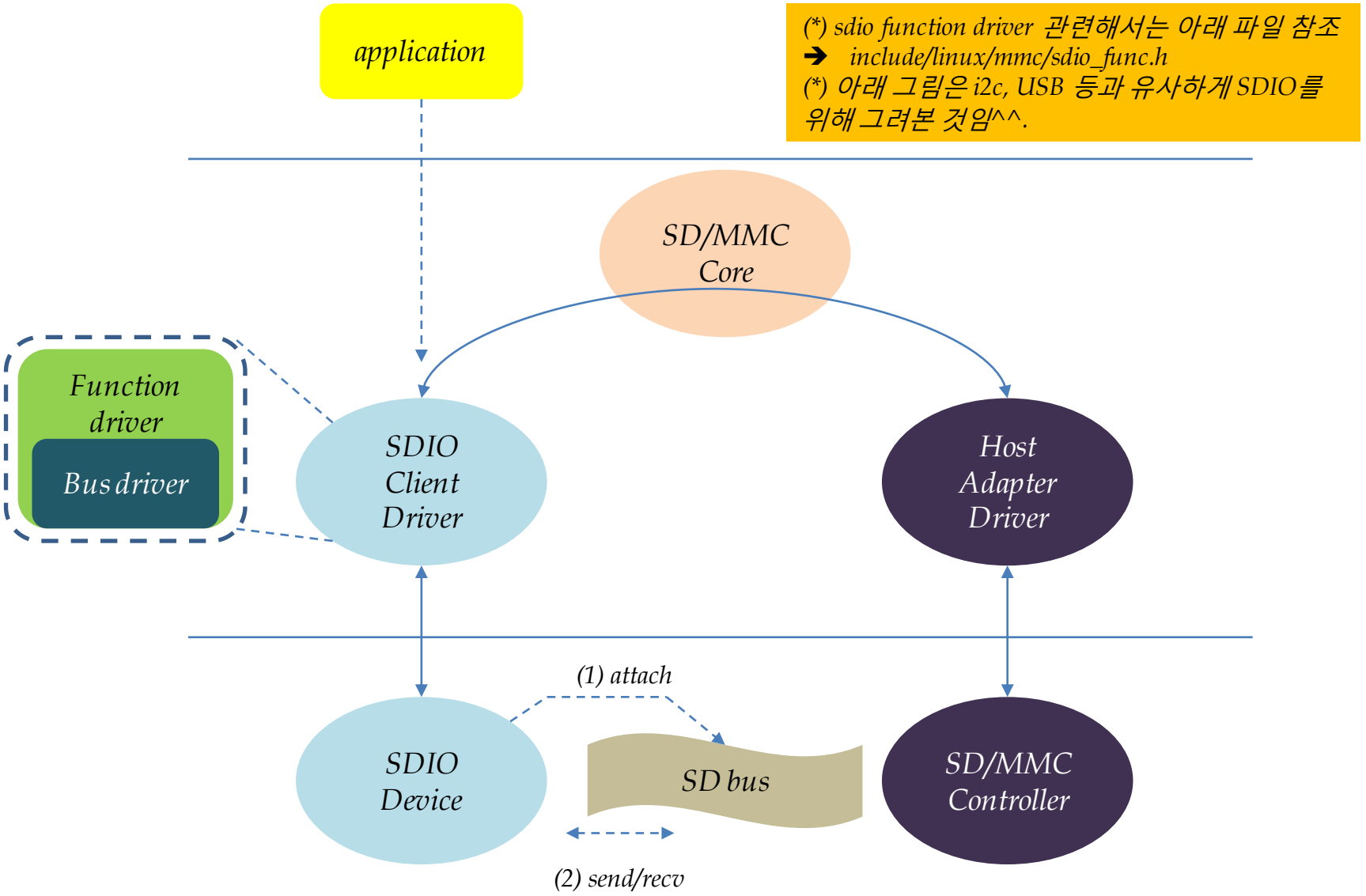


2. Wi-Fi 드라이버 구조(2)

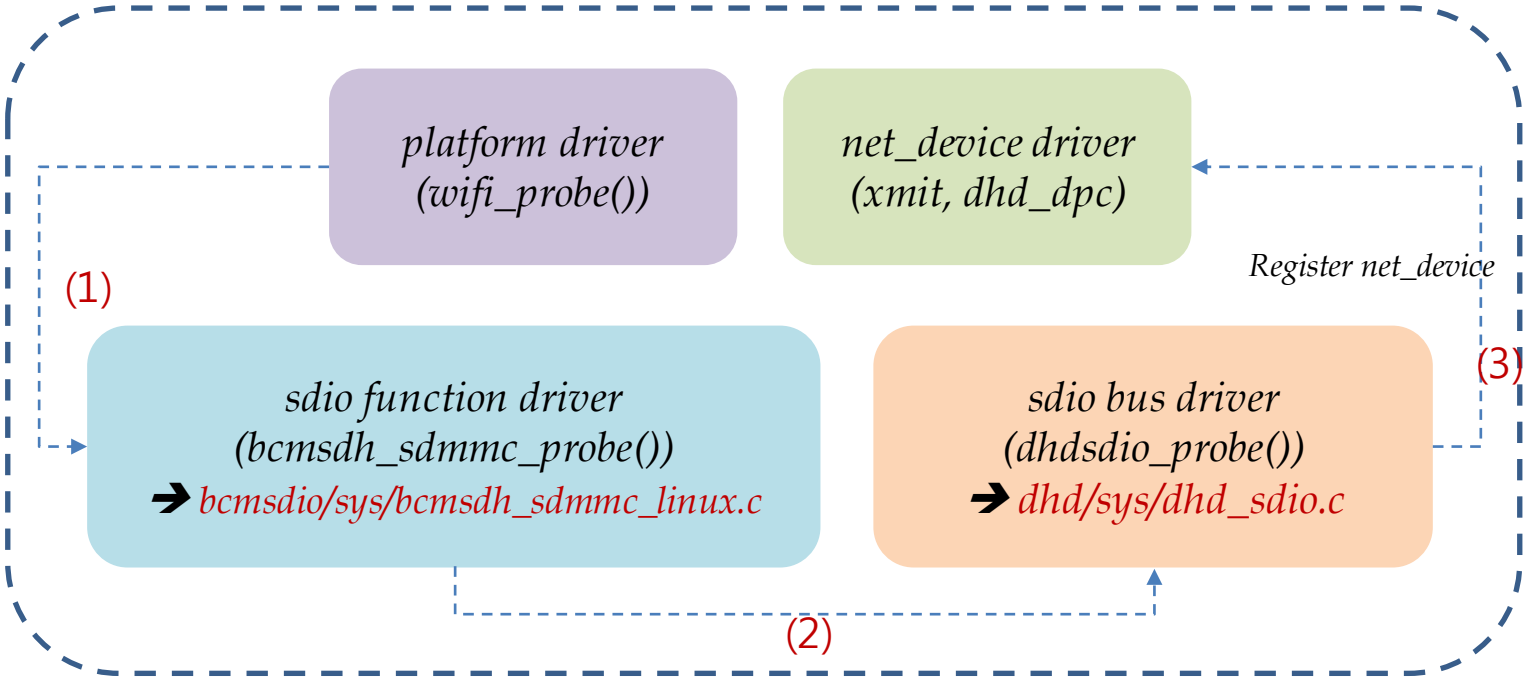
(*) 아래 그림은 WiFi 칩셋과 CPU 간의 SDIO 인터페이스(SD 4-bit mode) 사용 예를 보여준다.



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

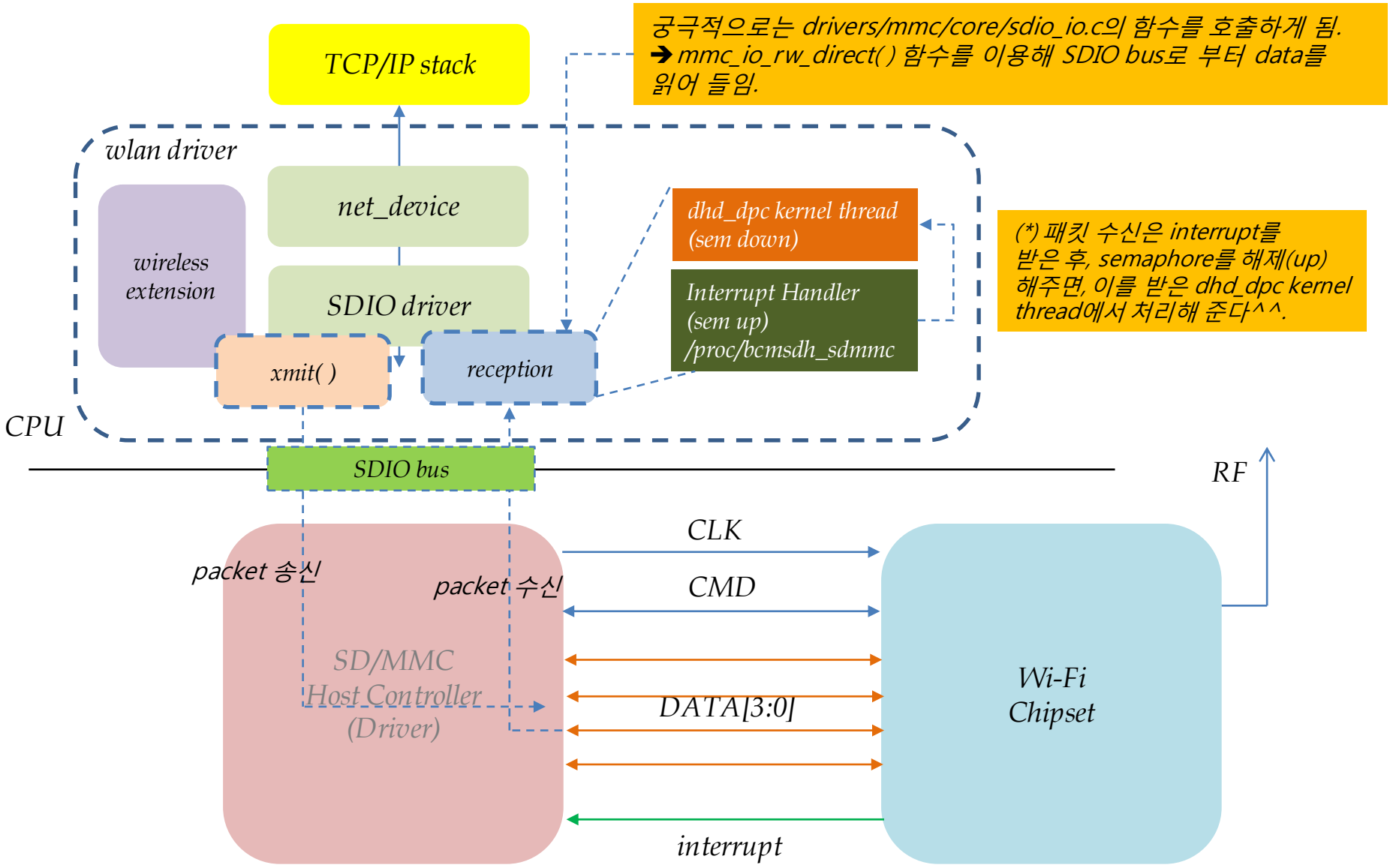


2. Wi-Fi 드라이버 구조(4) – ***bcm4329 driver overview(1)***

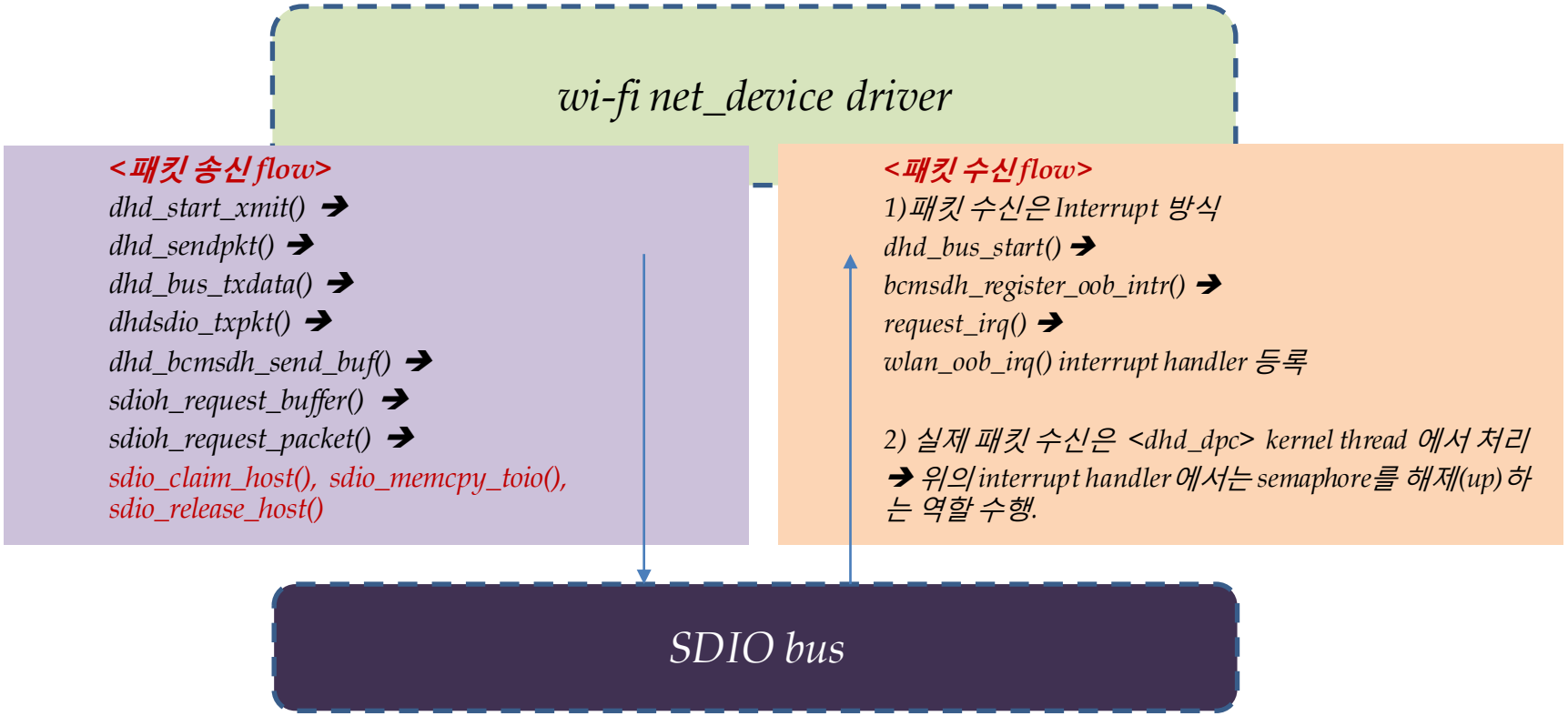


(*) BCM4329 driver는
1) Platform driver,
2) net_device(network) driver,
3) sdio function driver 이며,
4) sdio bus driver 이다.

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



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

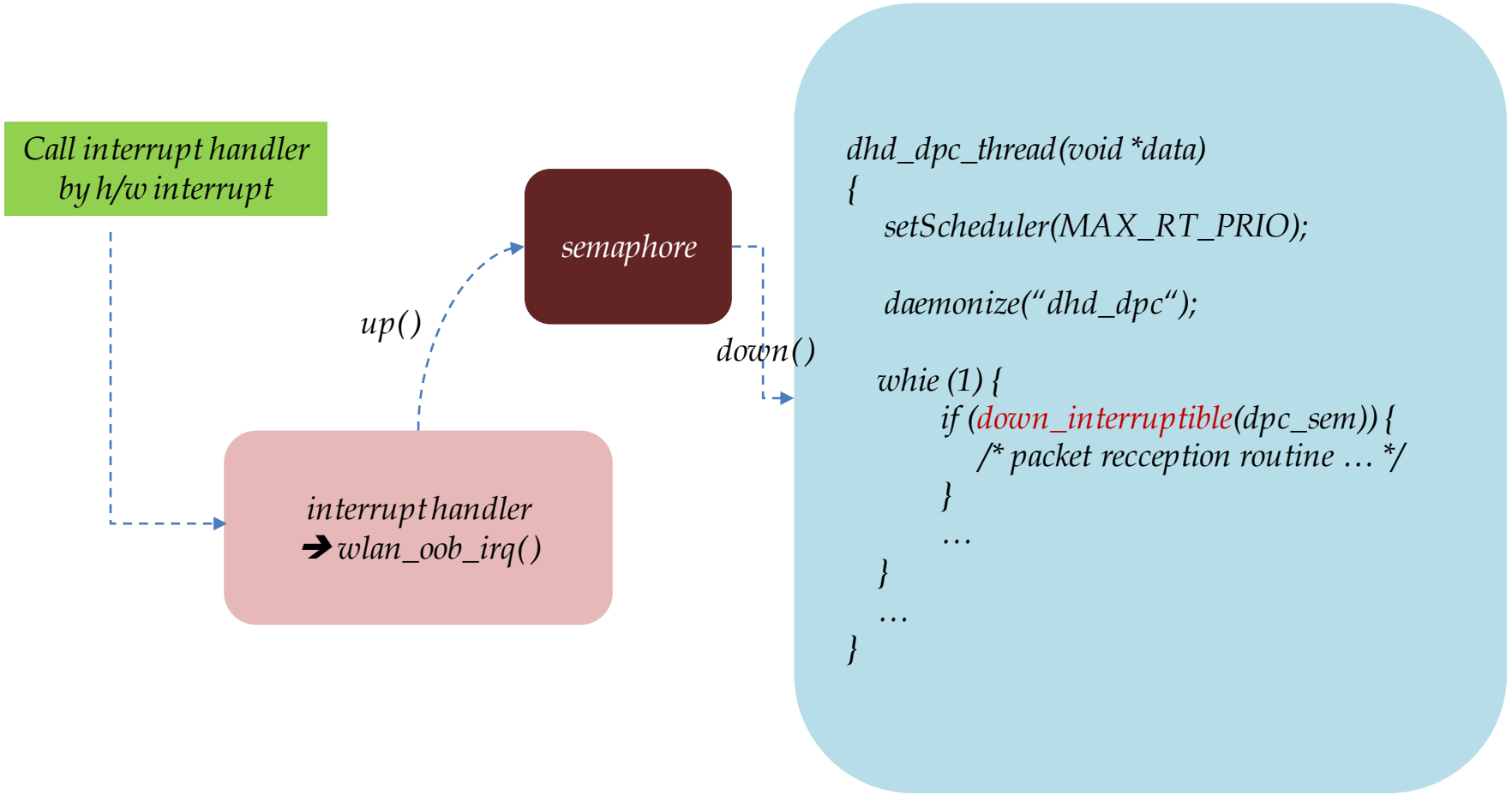


<참고 사항>

➔ 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

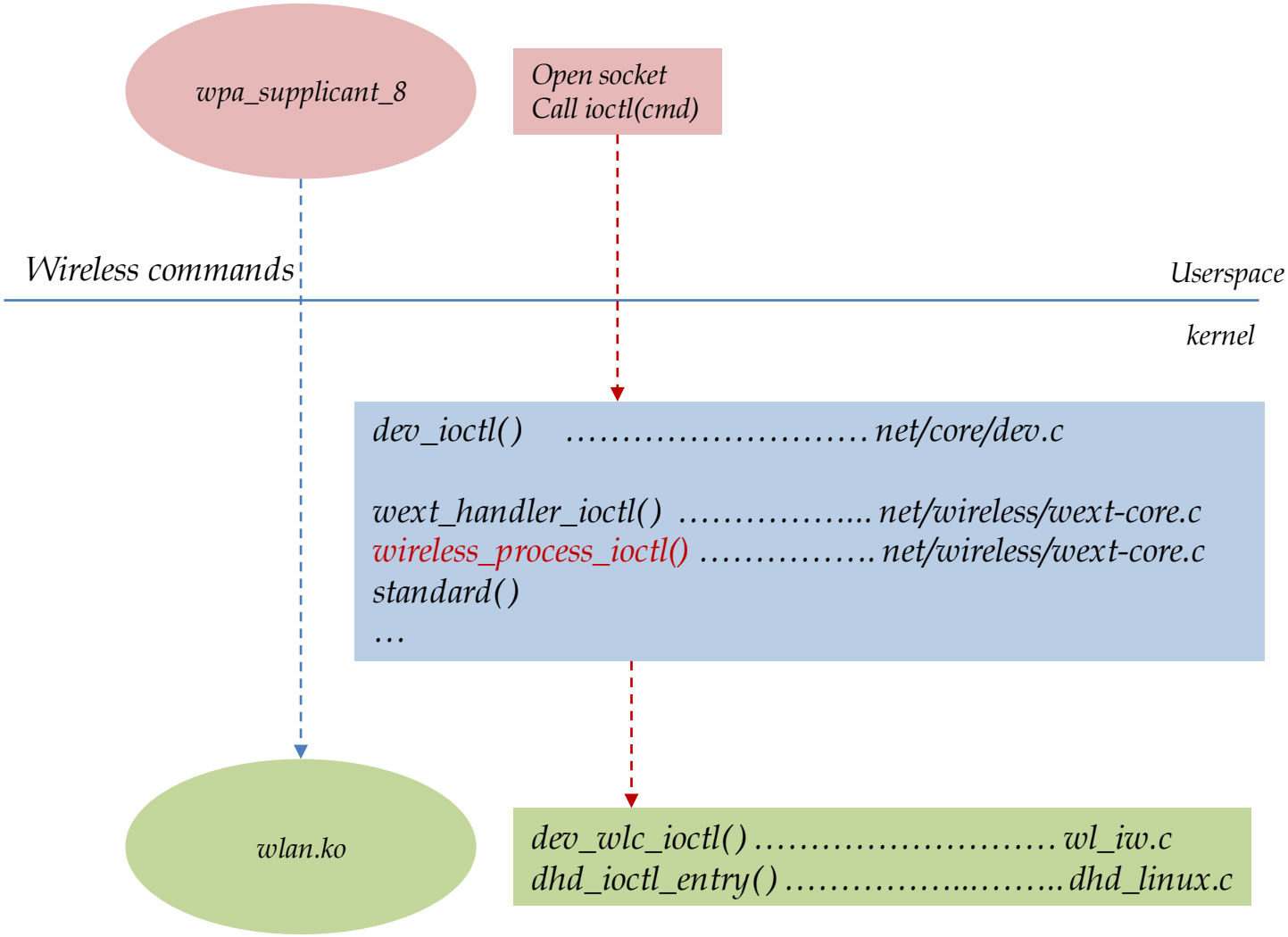
1-4) dhdsdio_probe_init

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)

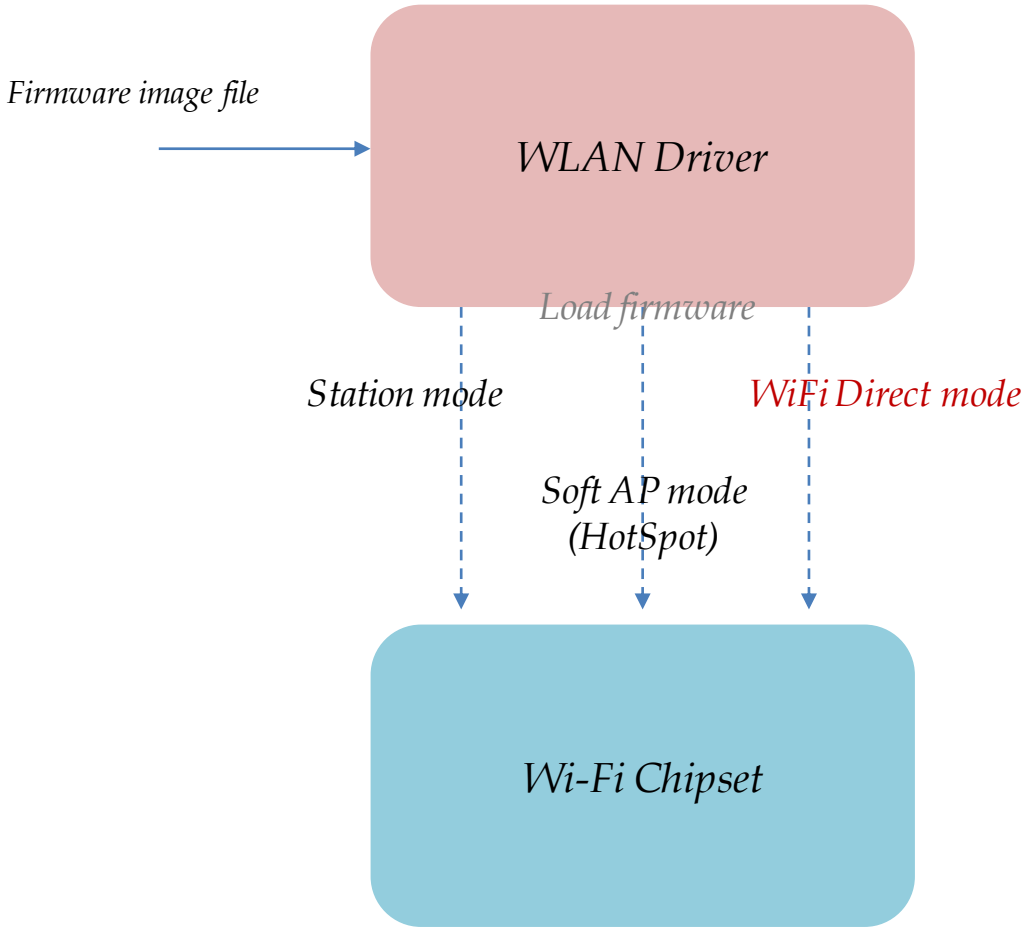
*1-7) *dhd_net_attach*
=> register_netdev() 호출하여, net_device로 등록함.

2. Wi-Fi 드라이버 구조(6): ***ioctl flow***

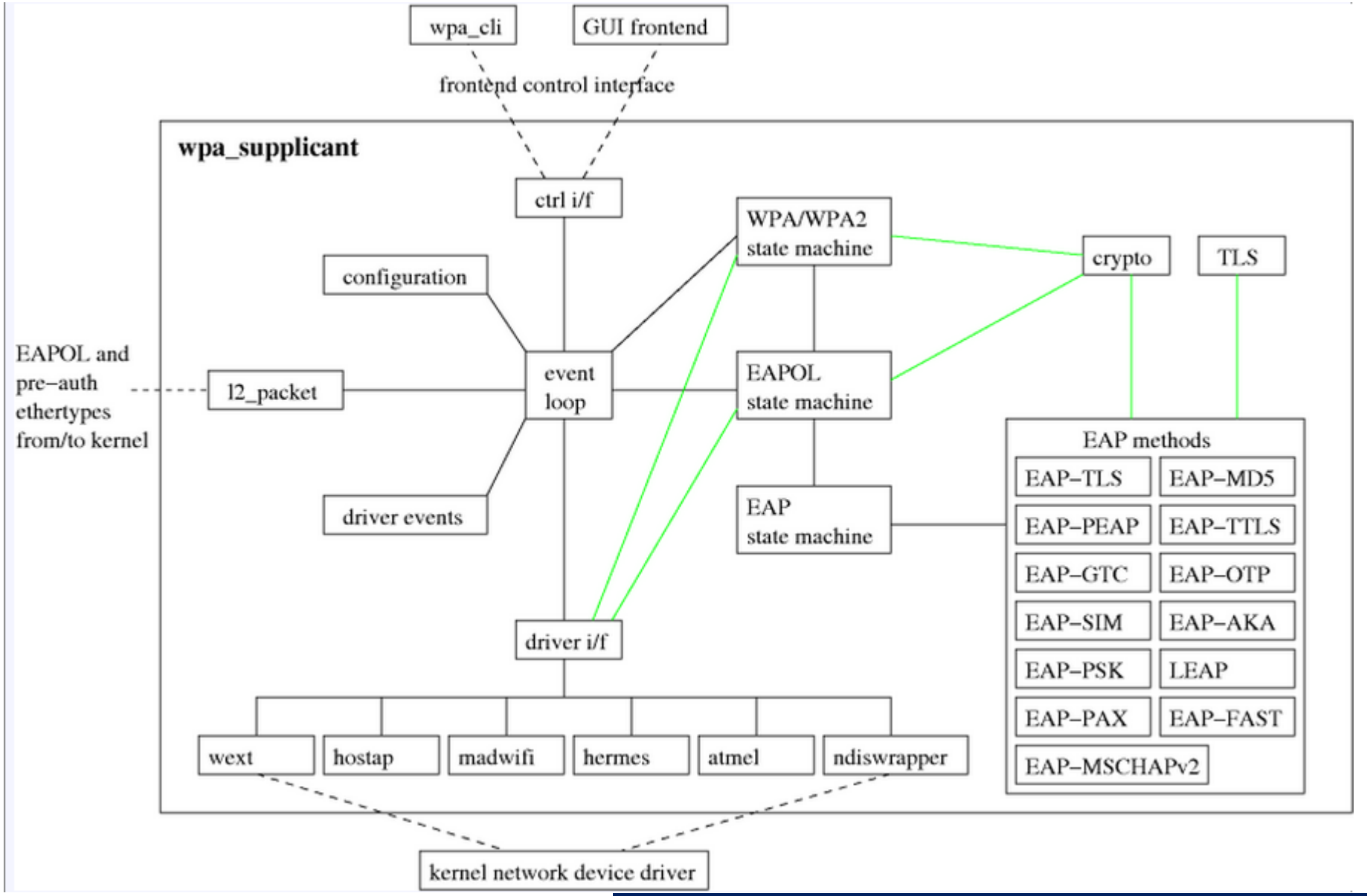


(*) 위의 과정을 통해 application으로 부터의 ioctl 명령이 wifi driver의 ioctl routine으로 전달된다.

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

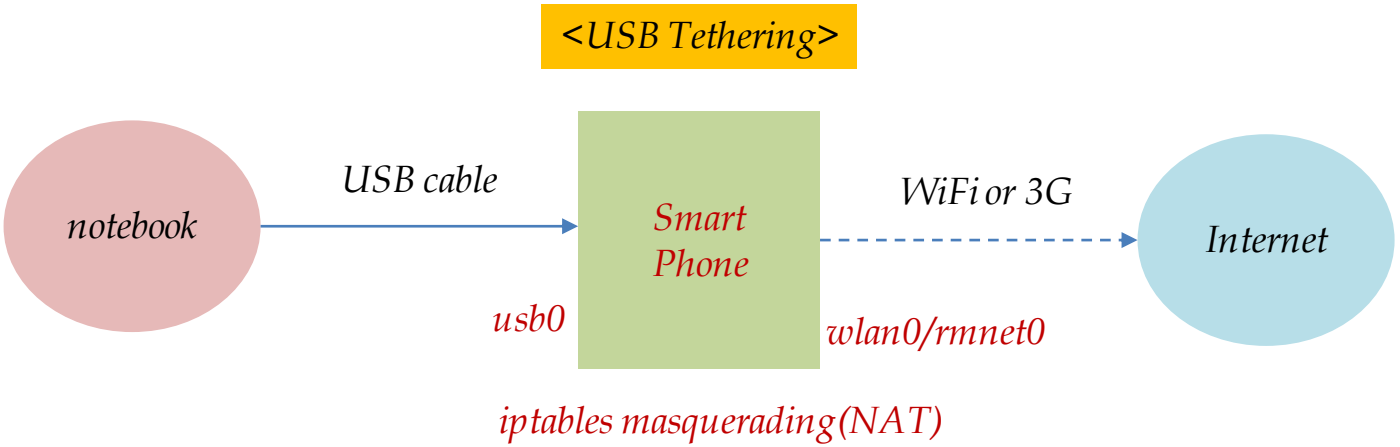


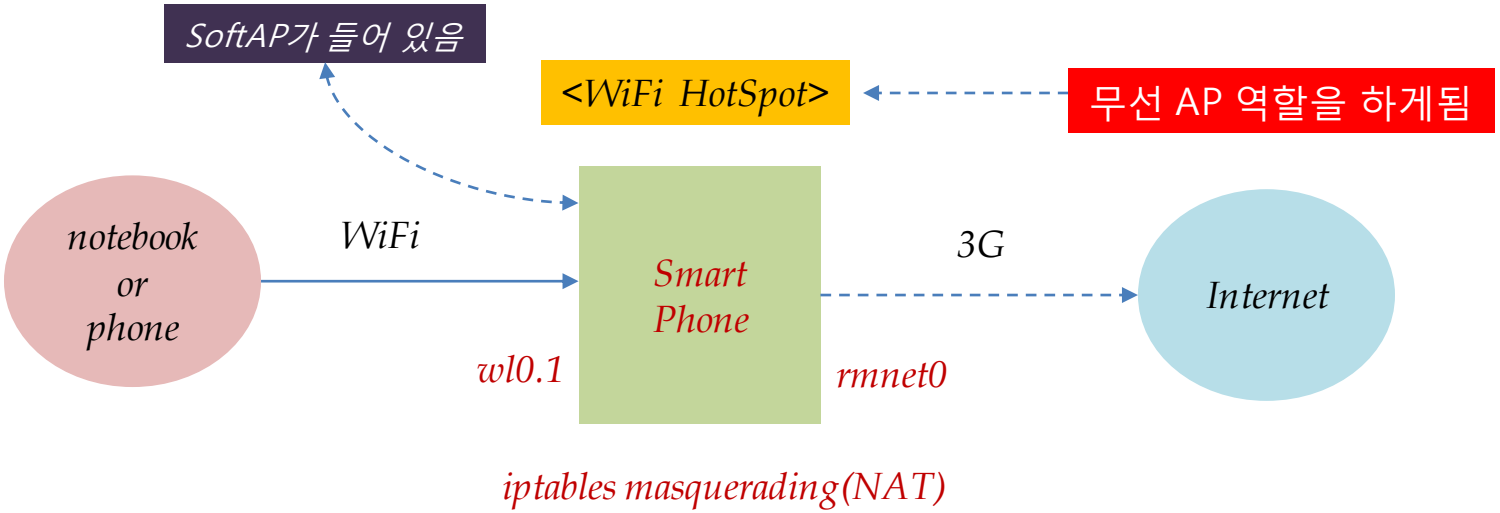
3. wpa_supplicant Architecture



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

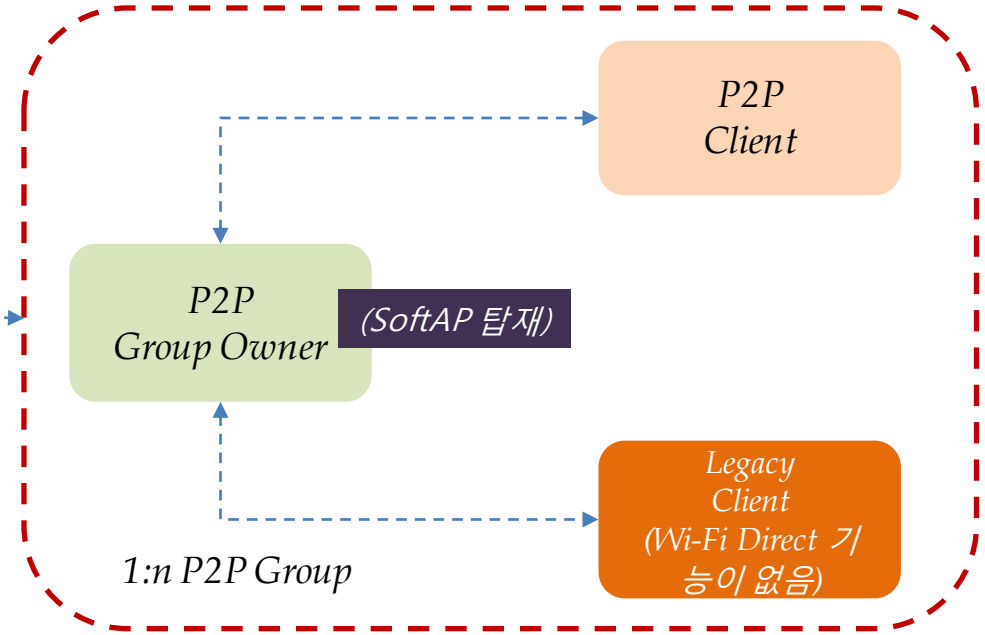
4. USB Tethering & WiFi HotSpot





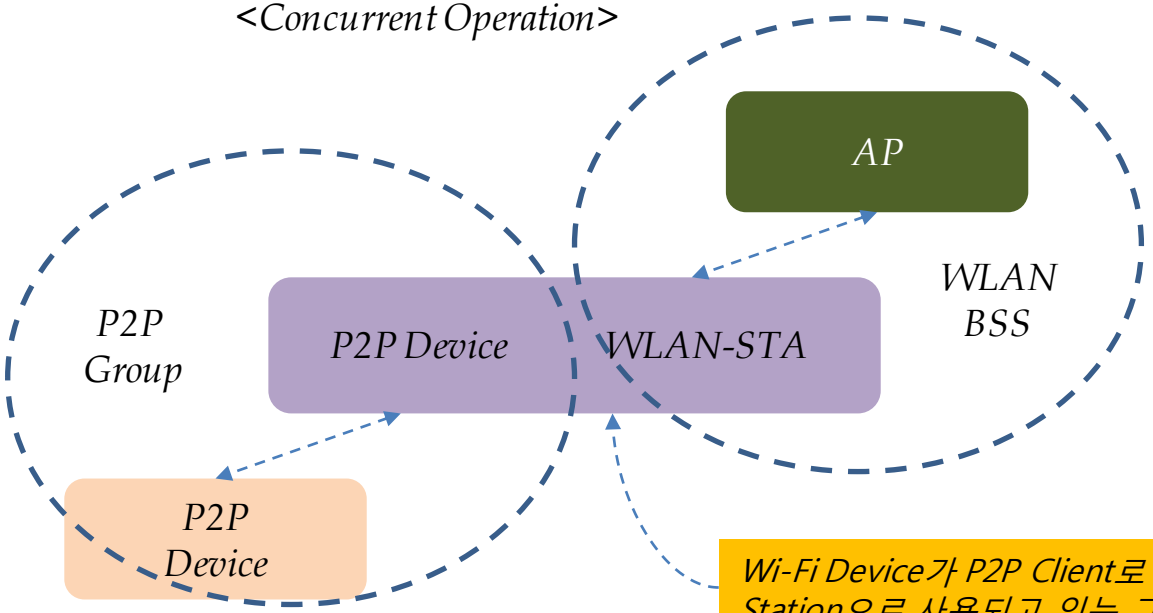
5. Wi-Fi Direct(Wi-Fi P2P)(1)

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*

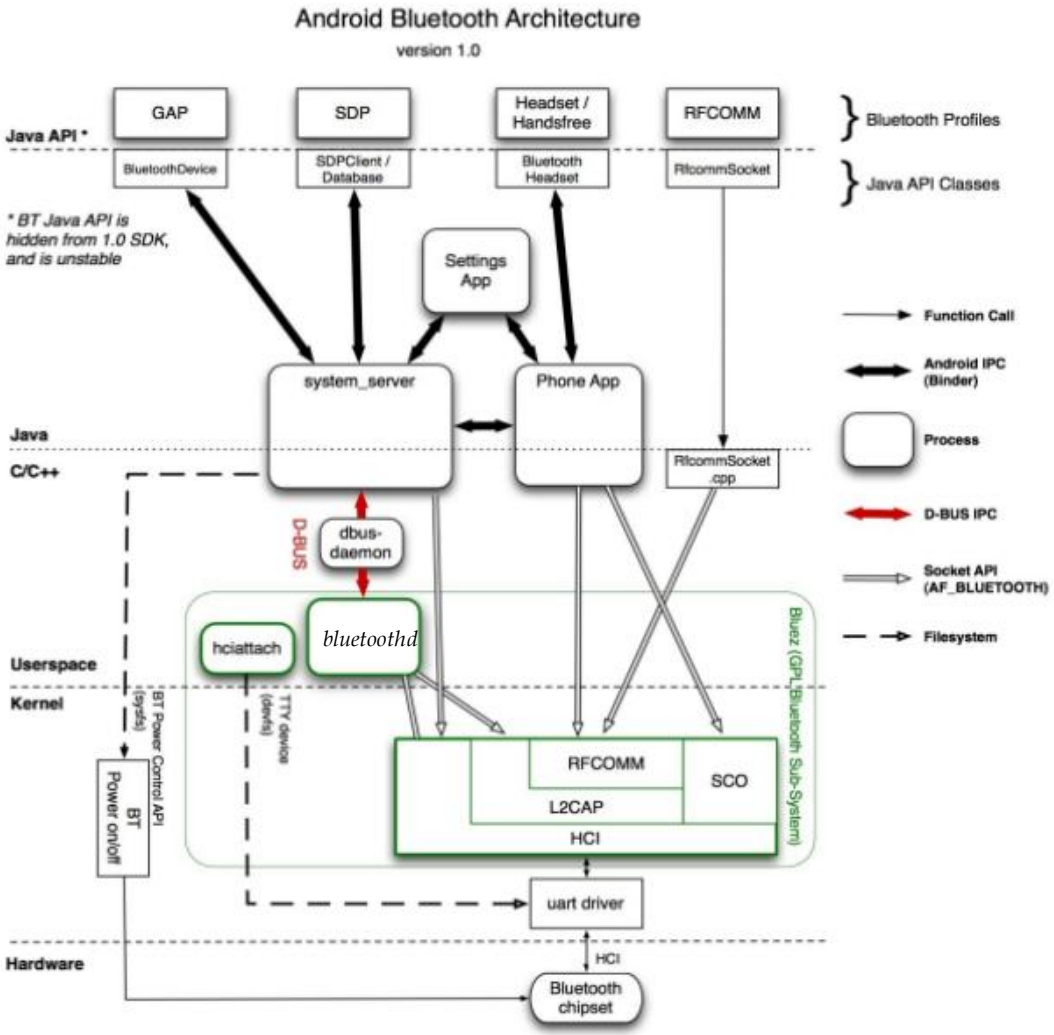
- *3) P2P Power Management*
- *➔ P2P 디바이스의 전력 관리 방법과 절전 모드 시점의 신호 처리 방법 규정*
- *(P2P GO sleep mode 진입 관련)*

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

6. Bluetooth Driver

1. Android Bluetooth Architecture

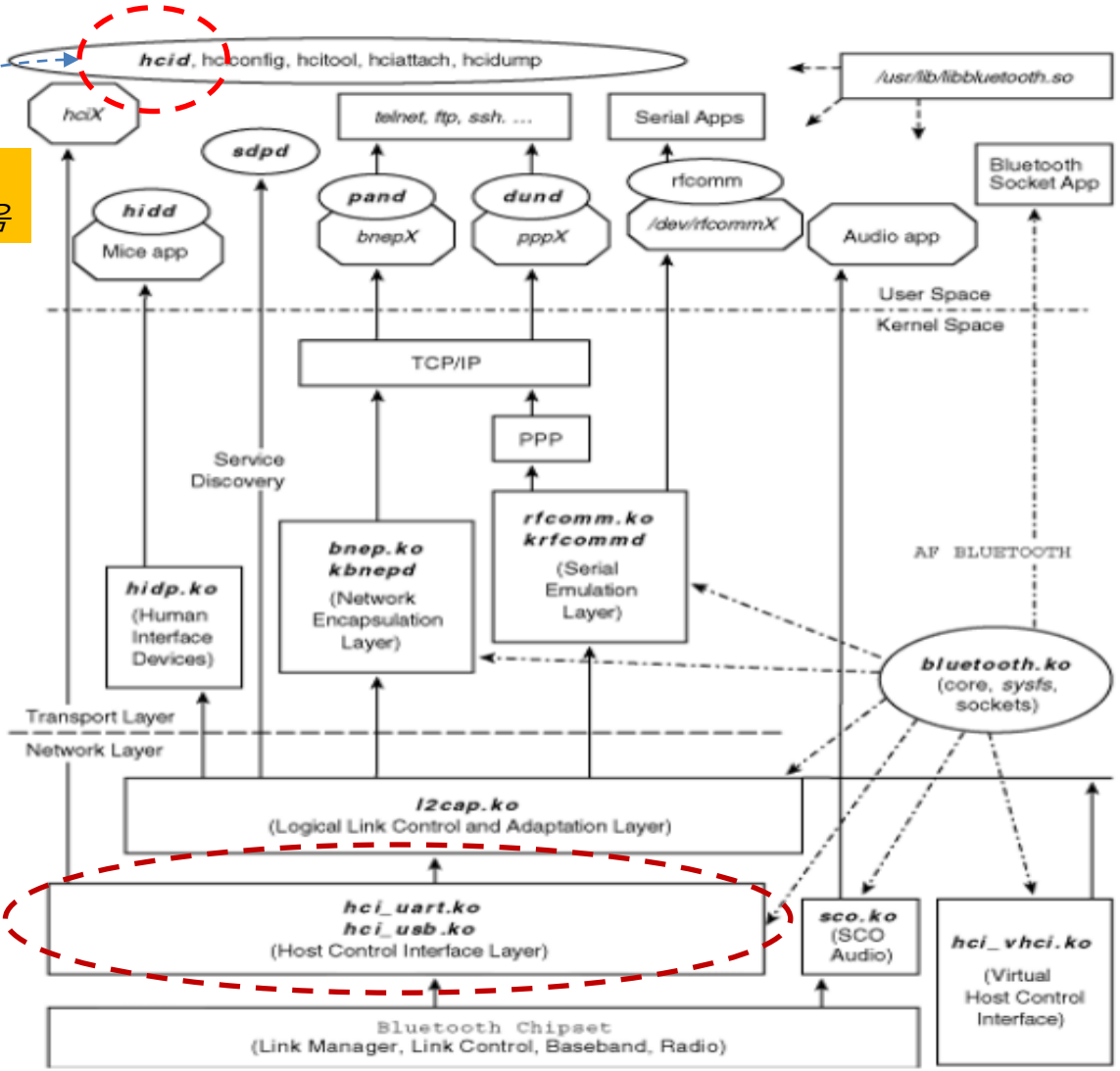
(*) 아래 그림은 인터넷에서 복사해 온 것임.



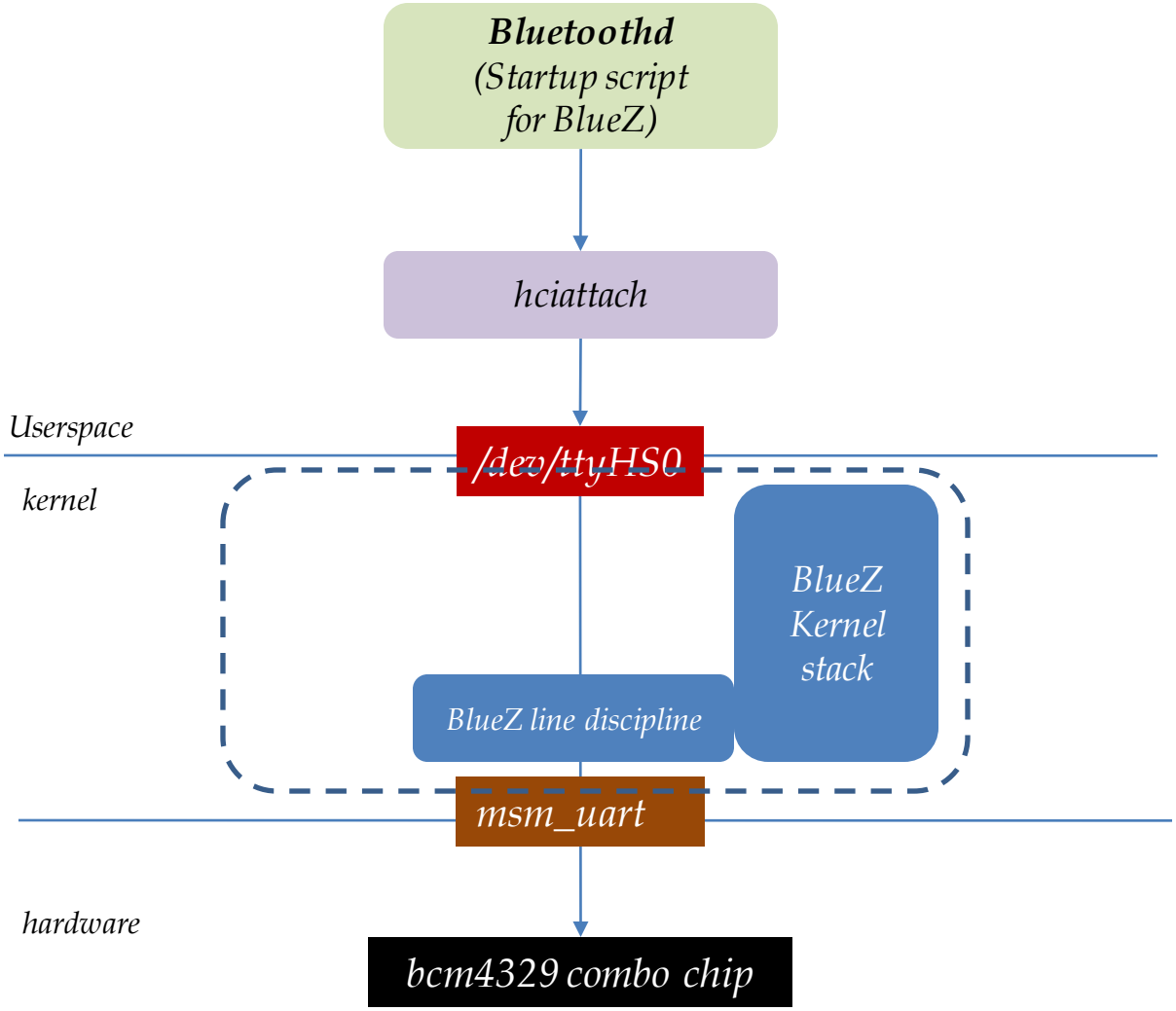
2. BlueZ Linux protocol stack

(*) 아래 그림은 참고 문서[1]에서 복사해 온 것임.

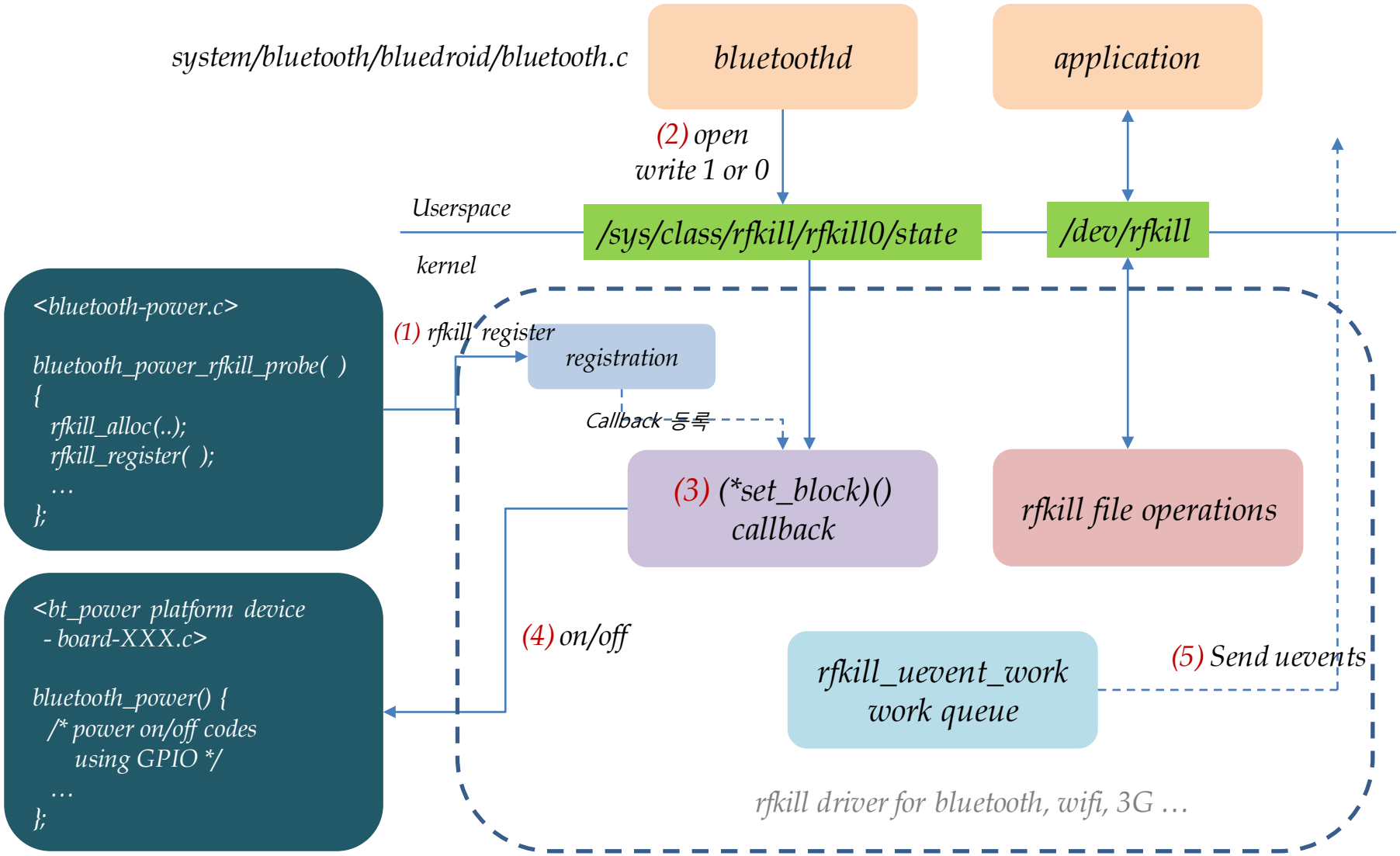
최신 버전에서는 bluetoothd로 바뀌었음



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.

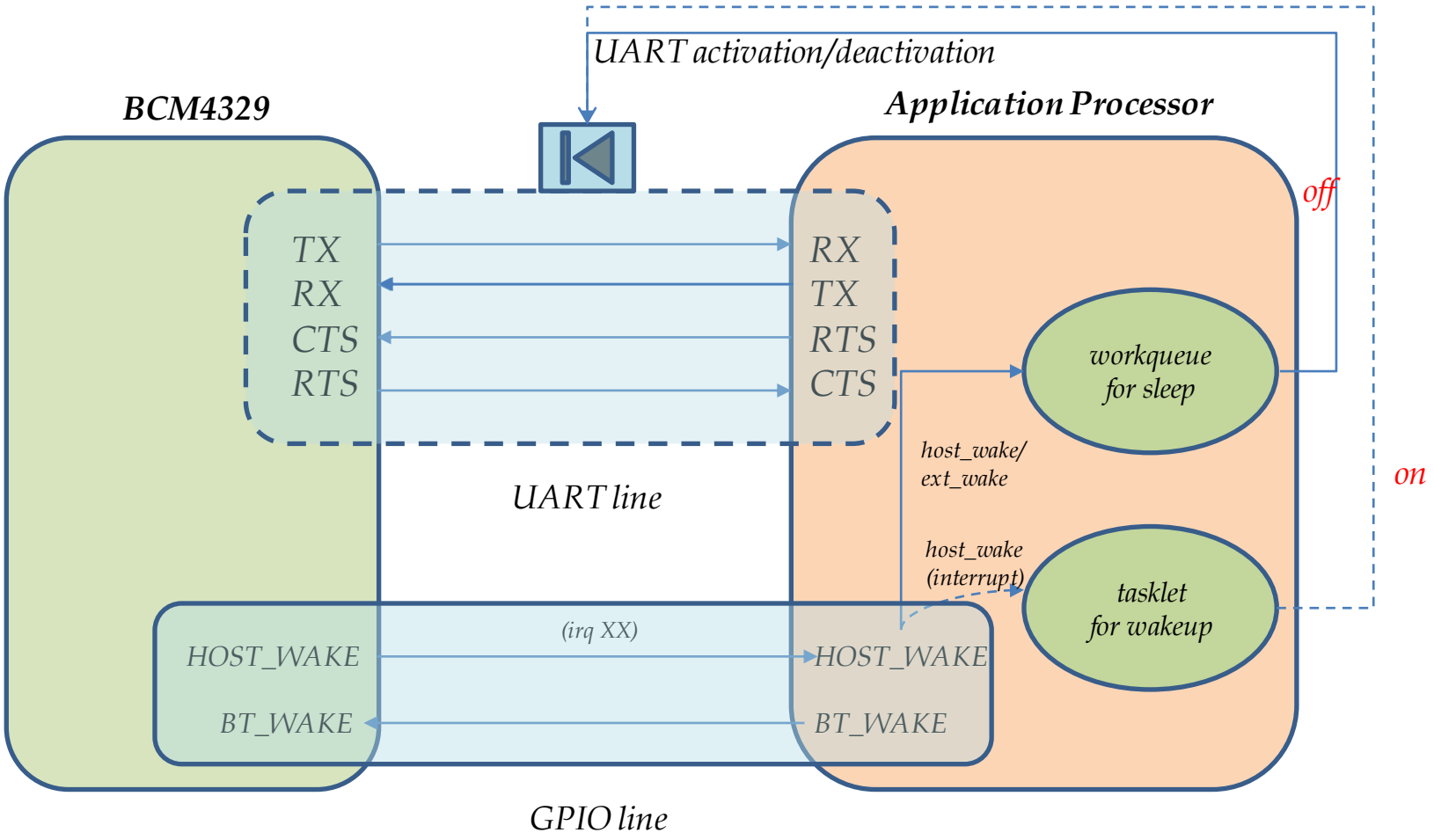
2) Application에서는 "/sys/class/rfkill/rfkillX/state" 파일을 open 한 후, power on의 경우 1을, power off의 경우 0을 write해 준다.

*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.

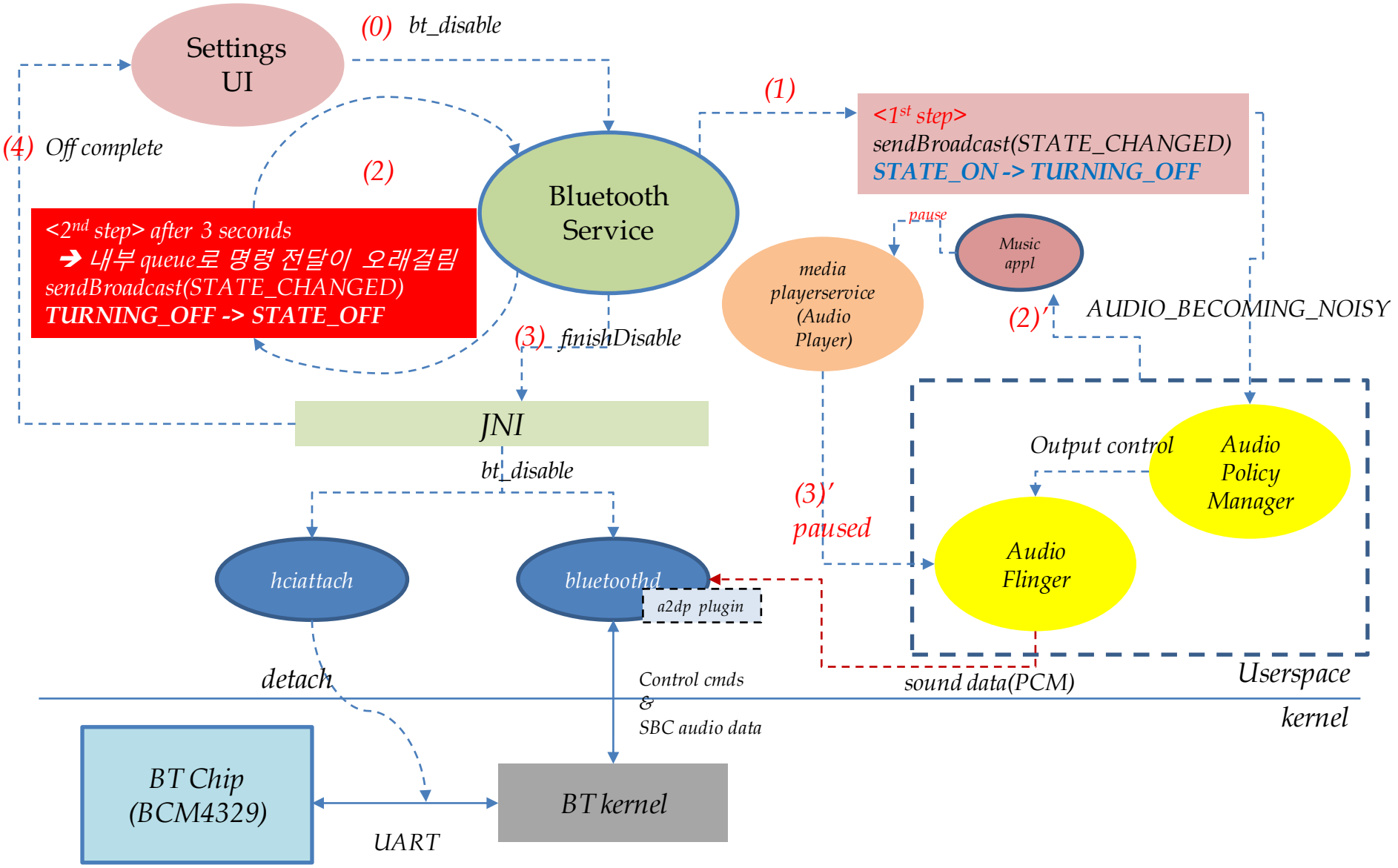
5) 상태 변화가 있을 때 마다 이를 uevent 형태로 application에 알린다.

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.

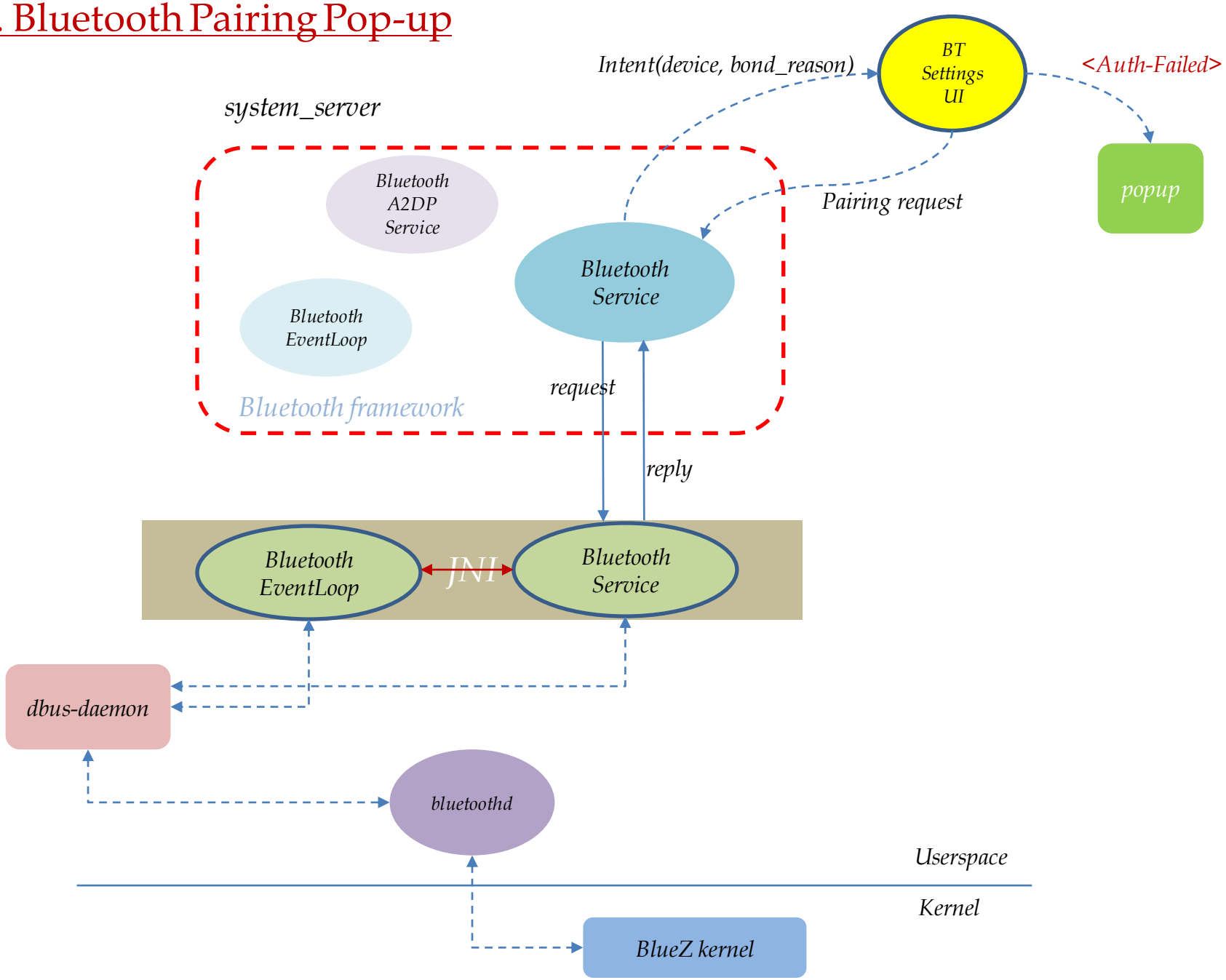
6. Bluetooth On/Off Flow



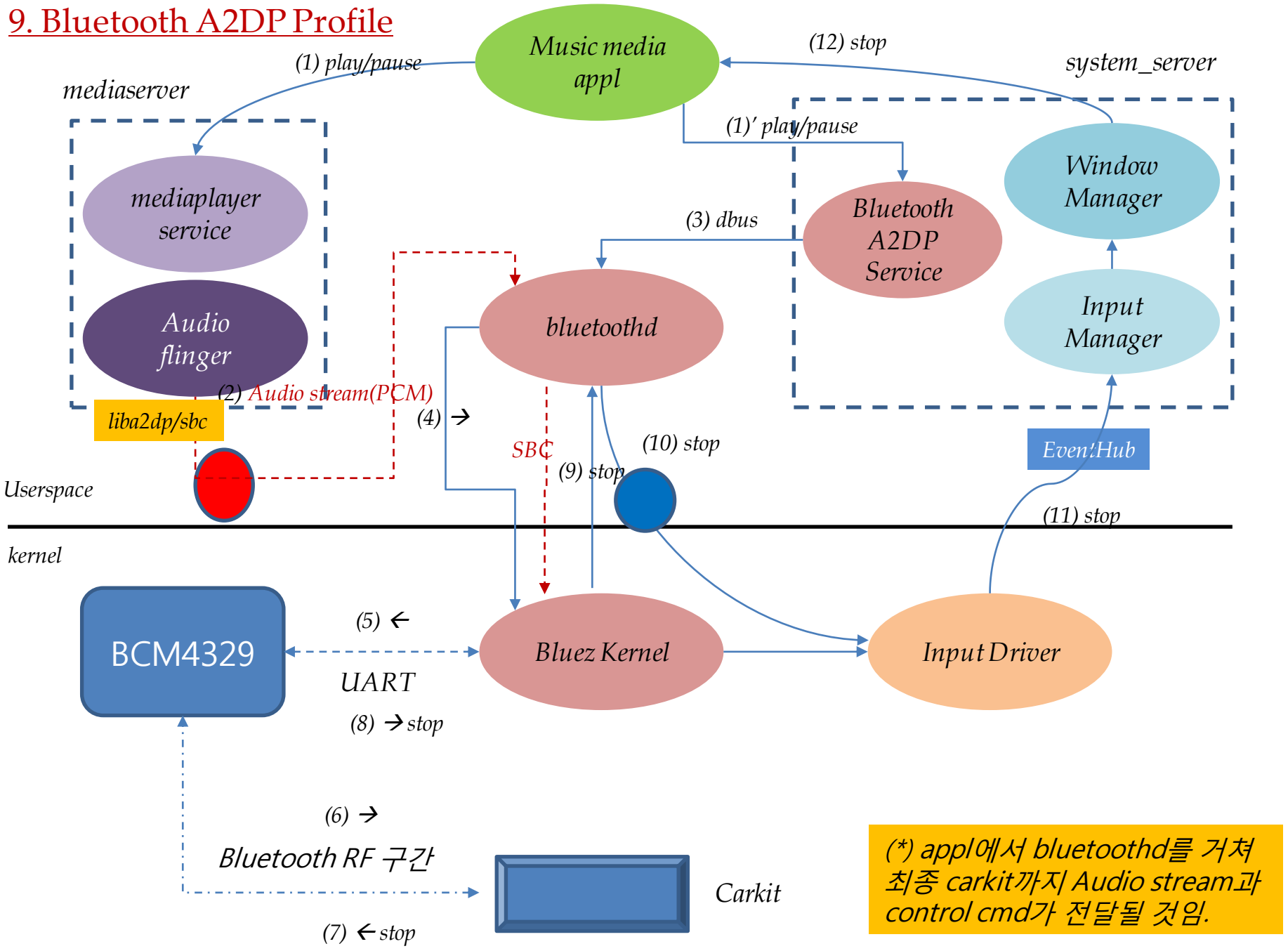
7. Bluetooth Profile

- 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

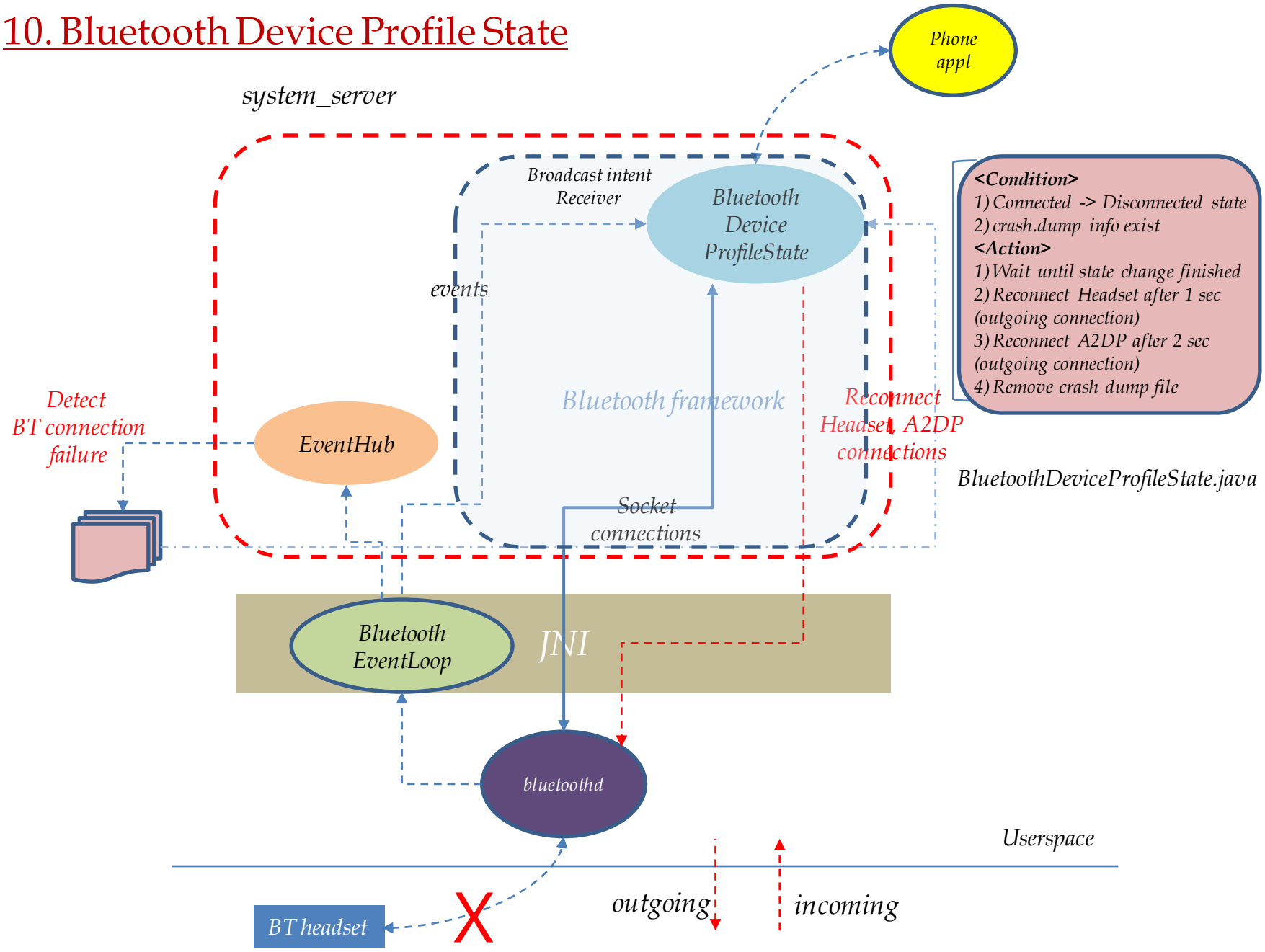


9. Bluetooth A2DP Profile

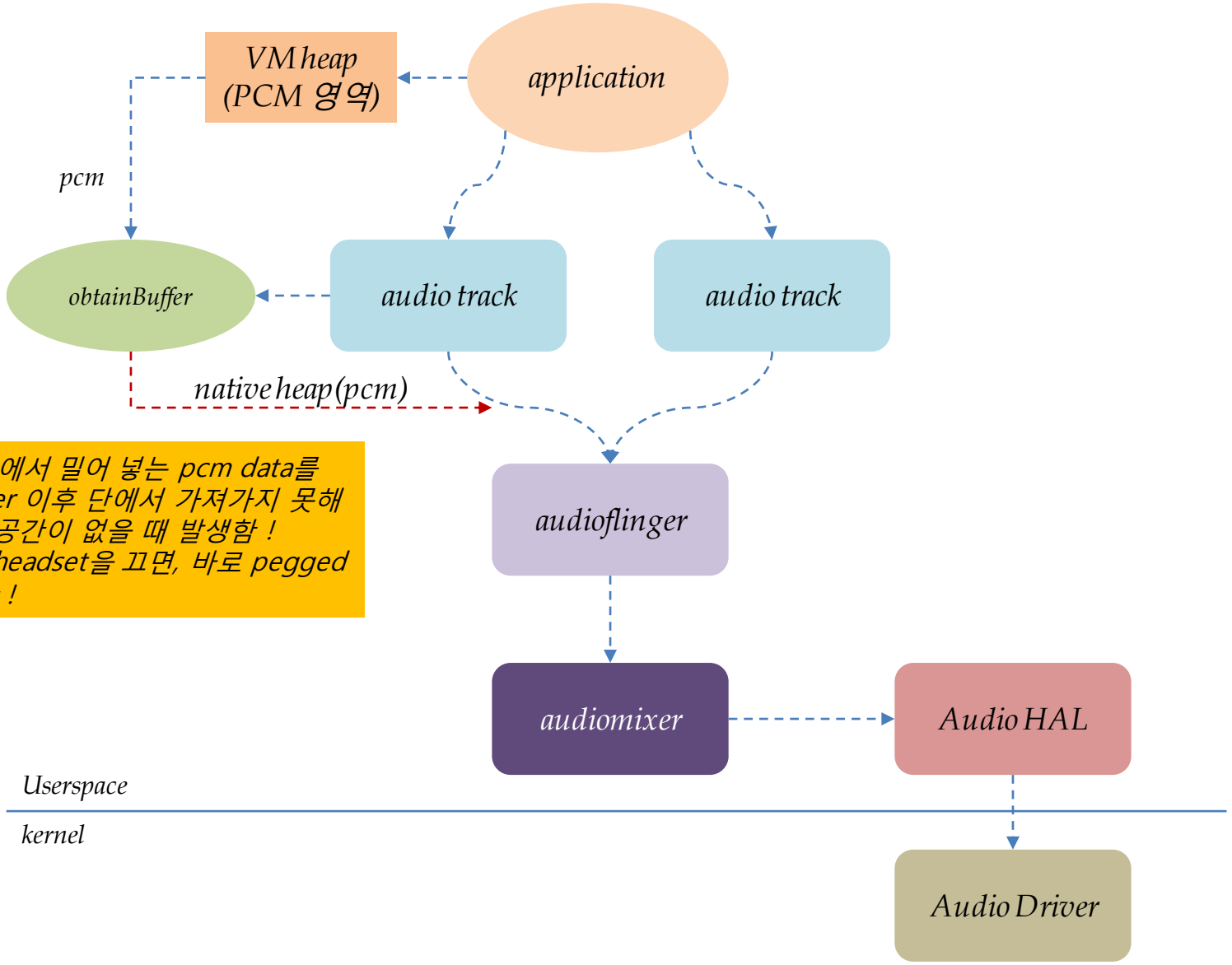


(*) appl에서 bluetoothd를 거쳐 최종 carkit까지 Audio stream과 control cmd가 전달될 것임.

10. Bluetooth Device Profile State



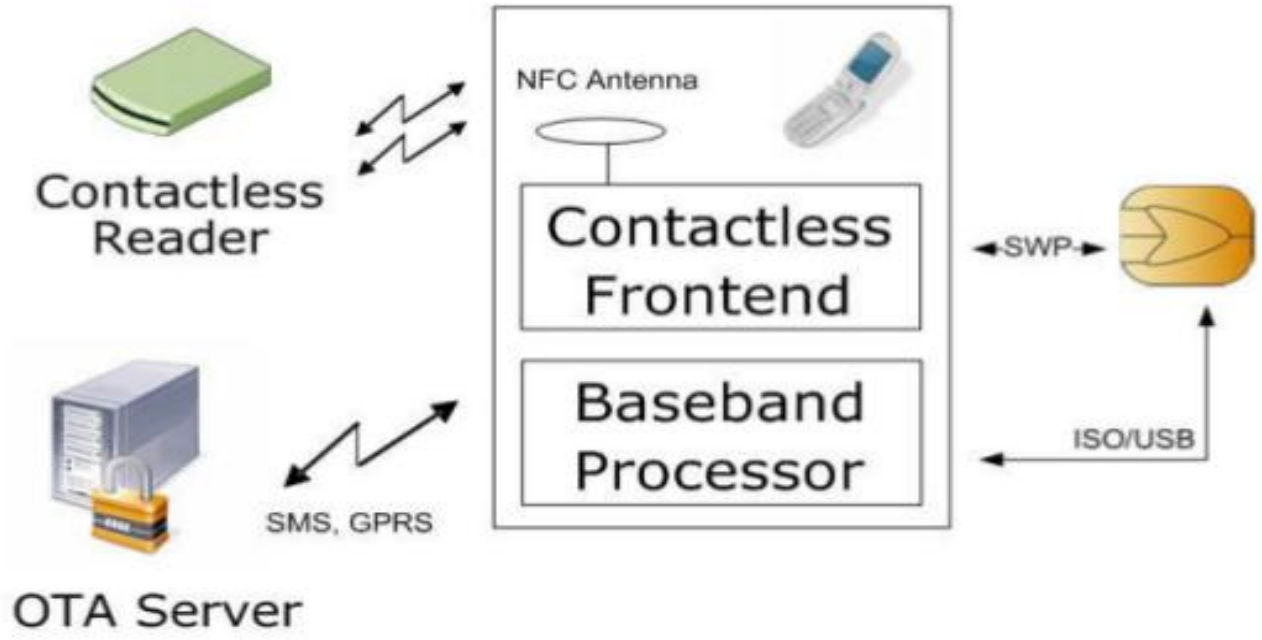
11. Case Study: *obtainBuffer timed out(is the CPU pegged?) issue*

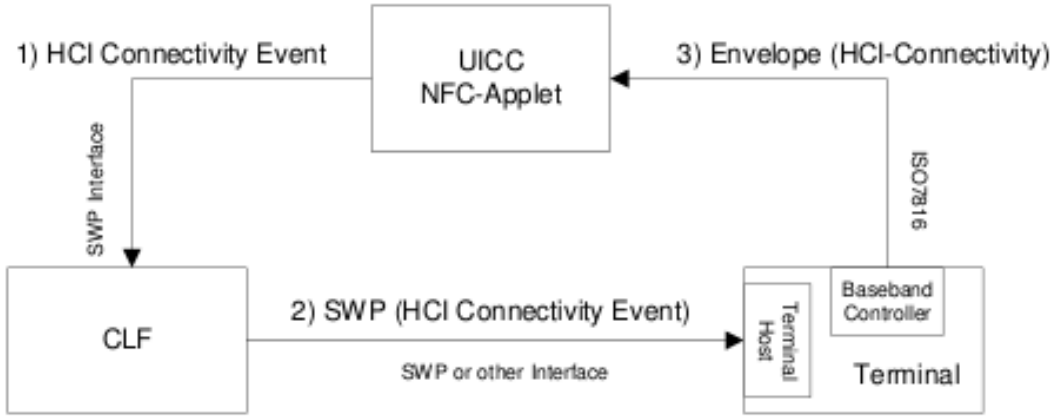


Application에서 밀어 넣는 pcm data를 Audio flinger 이후 단에서 가져가지 못해 frameAvail 공간이 없을 때 발생함 !
➔ 가령, BT headset을 끄면, 바로 pegged 에러 발생함 !

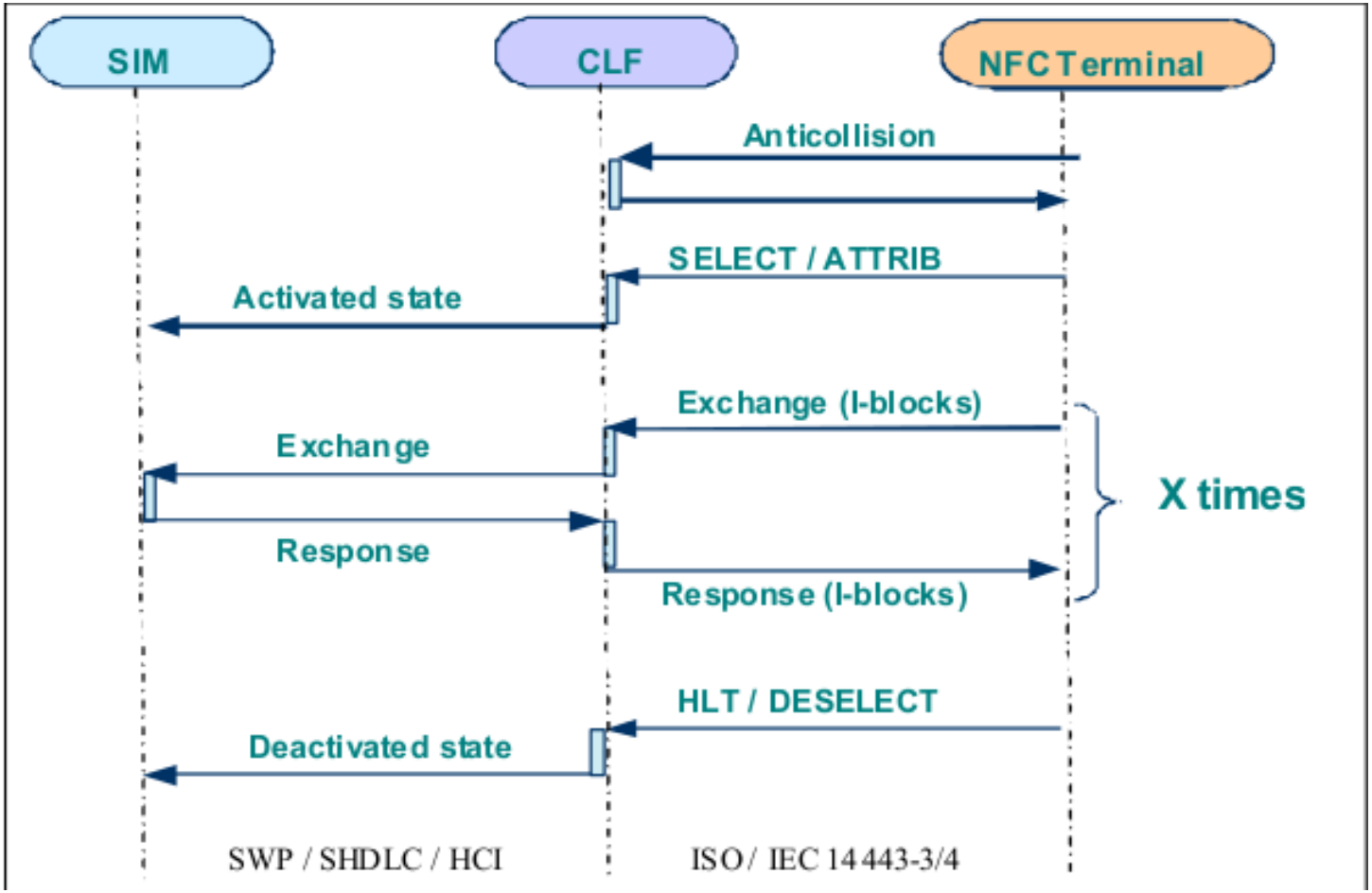
7. NFC Driver

1. NFC Overview(1)

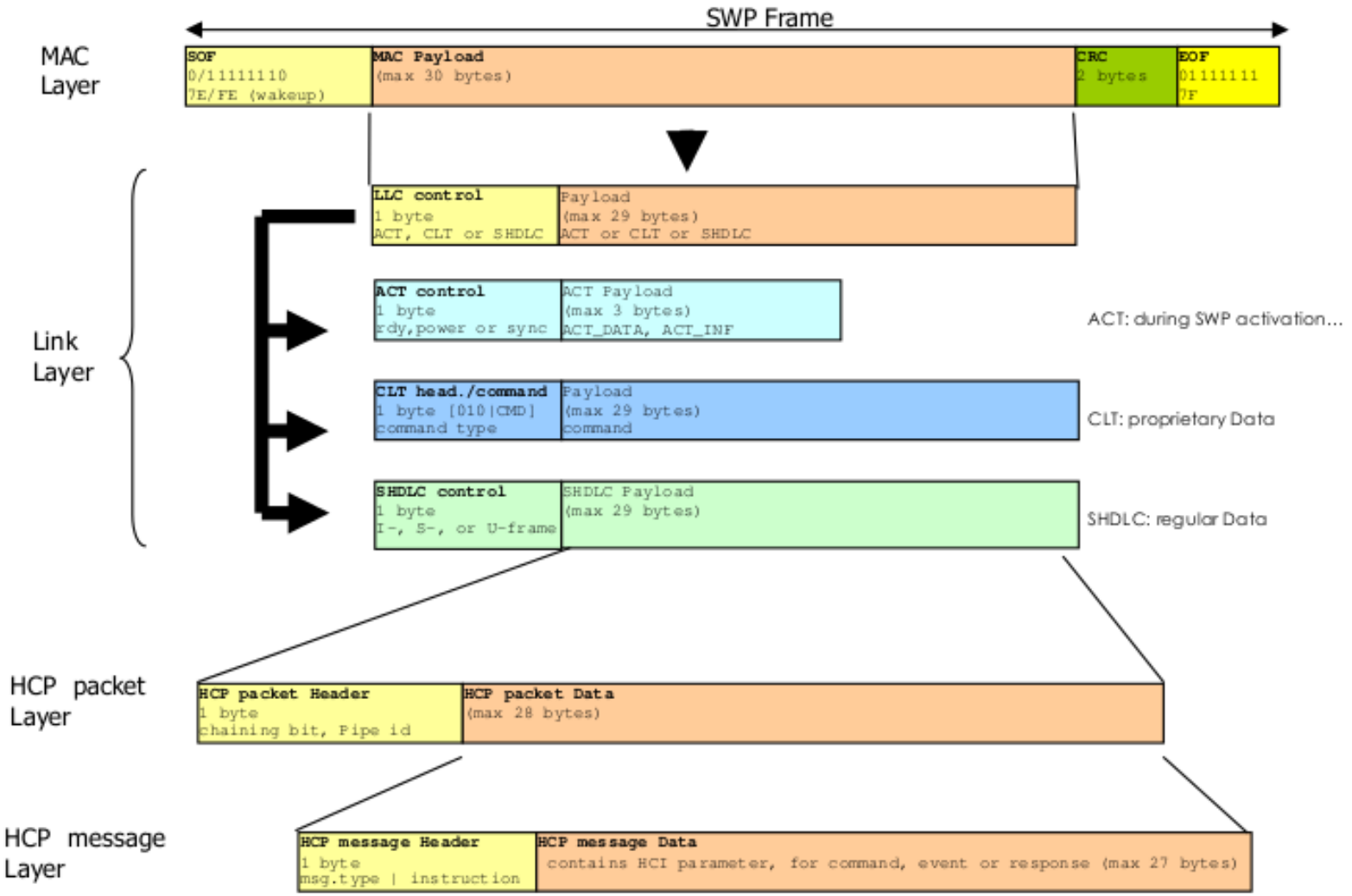




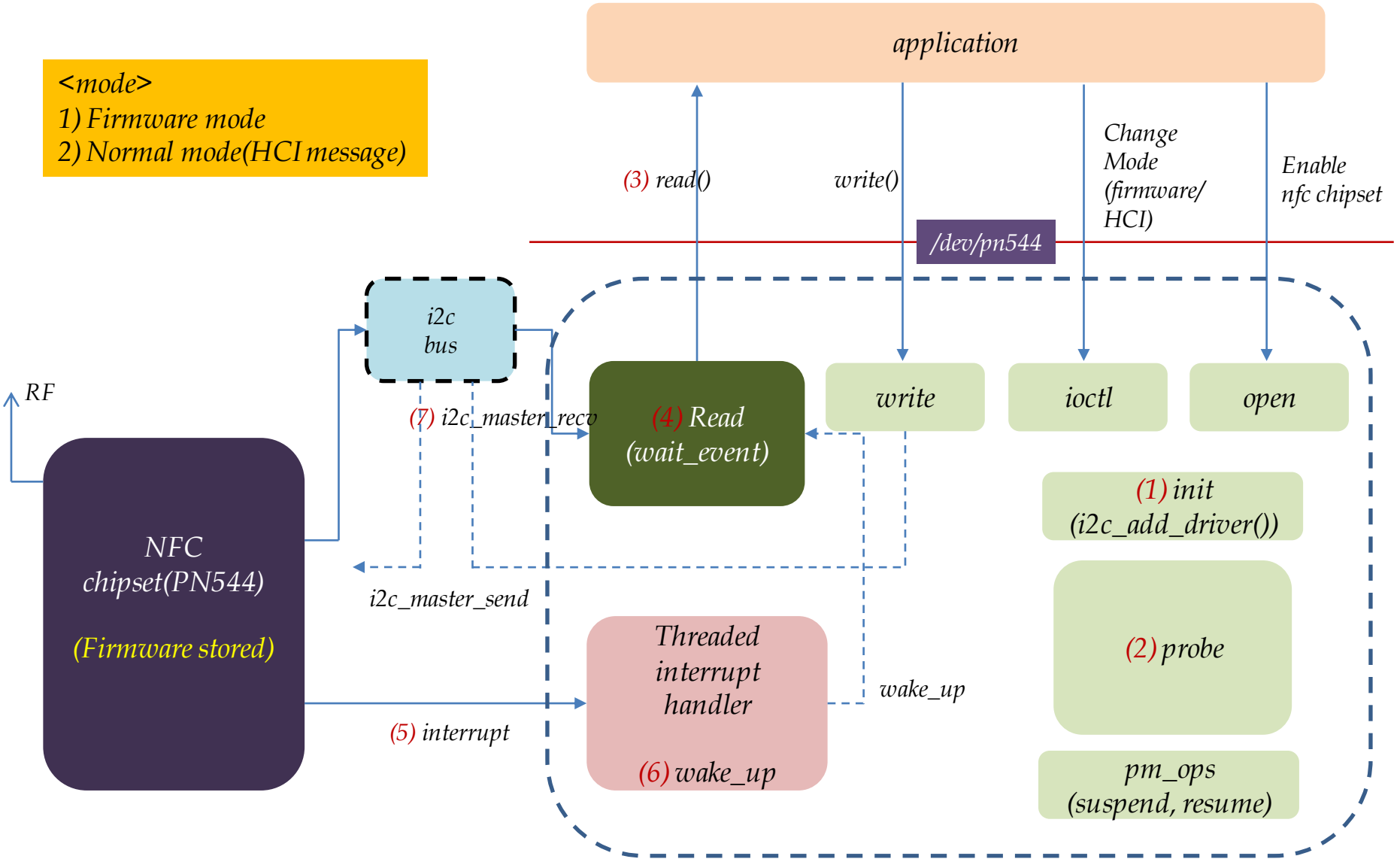
1. NFC Overview(2)



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.

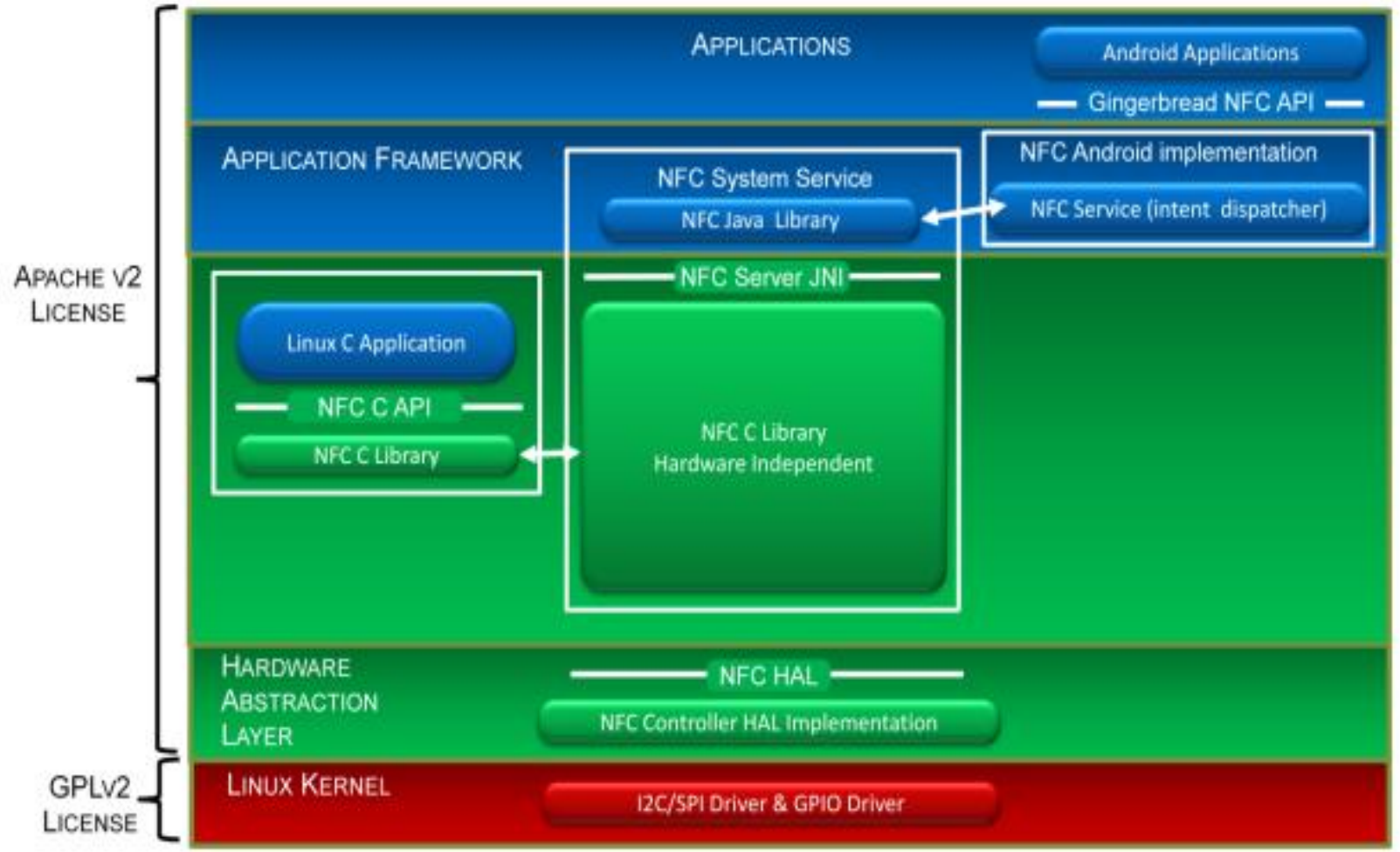
4) Application에서는 ioctl 함수를 사용하여, nfc chipset의 동작 방식을 firmware update mode와 normal mode(HCI mode)로 바꾸어 준다.

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.

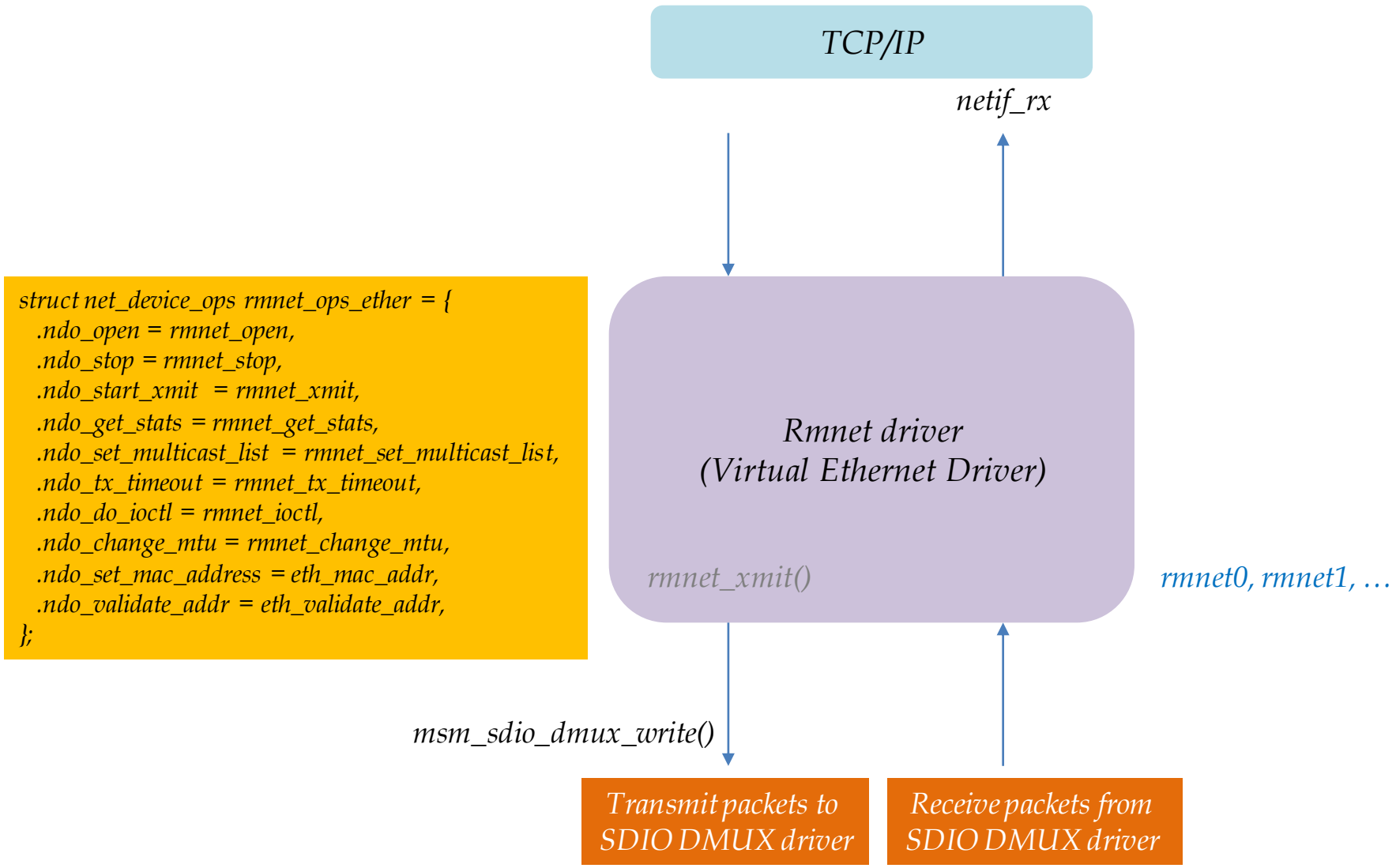
3. Android NFC Framework Overview(2)

<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

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);

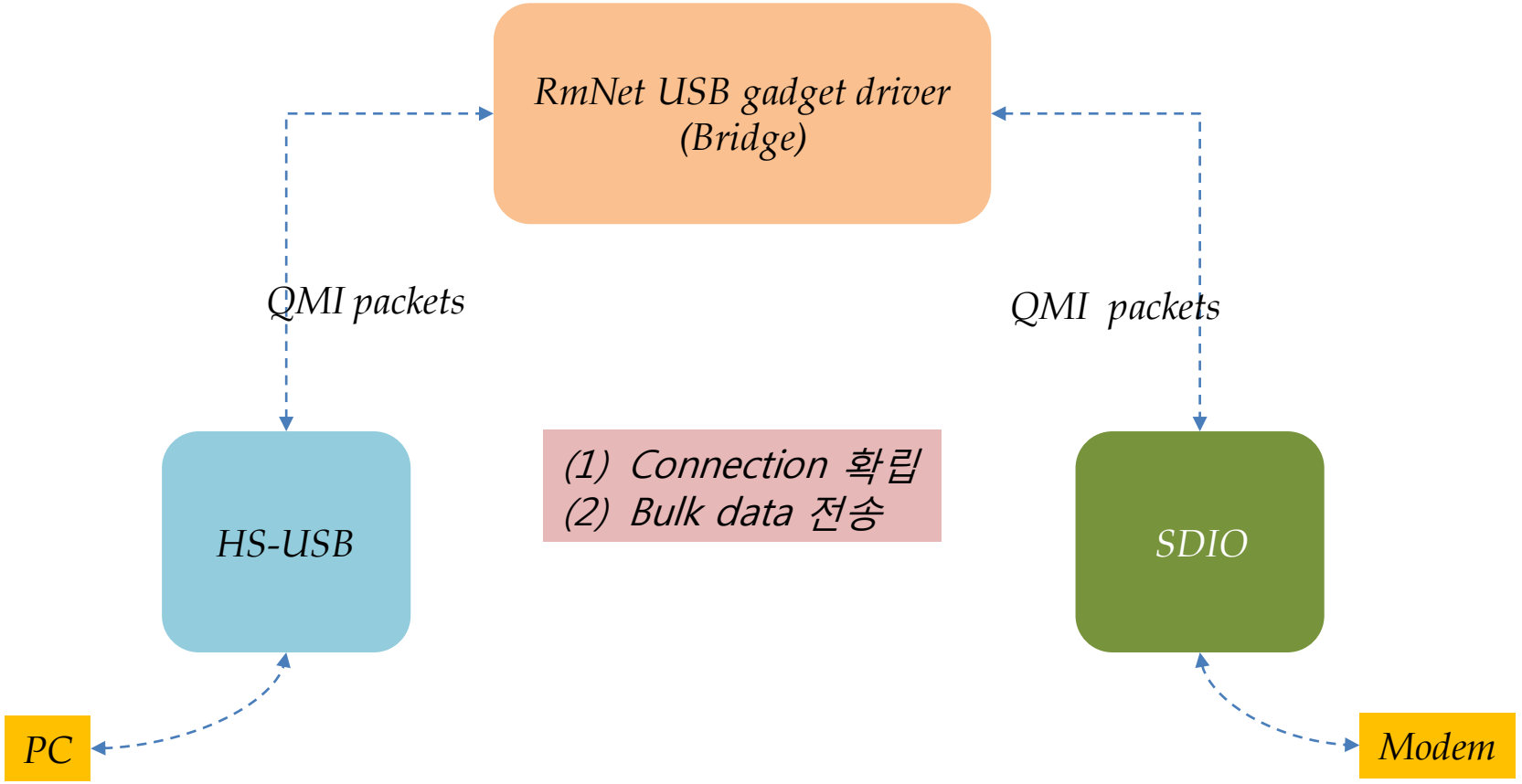
8) ret = register_netdev(dev);

9) sysfs entry 생성

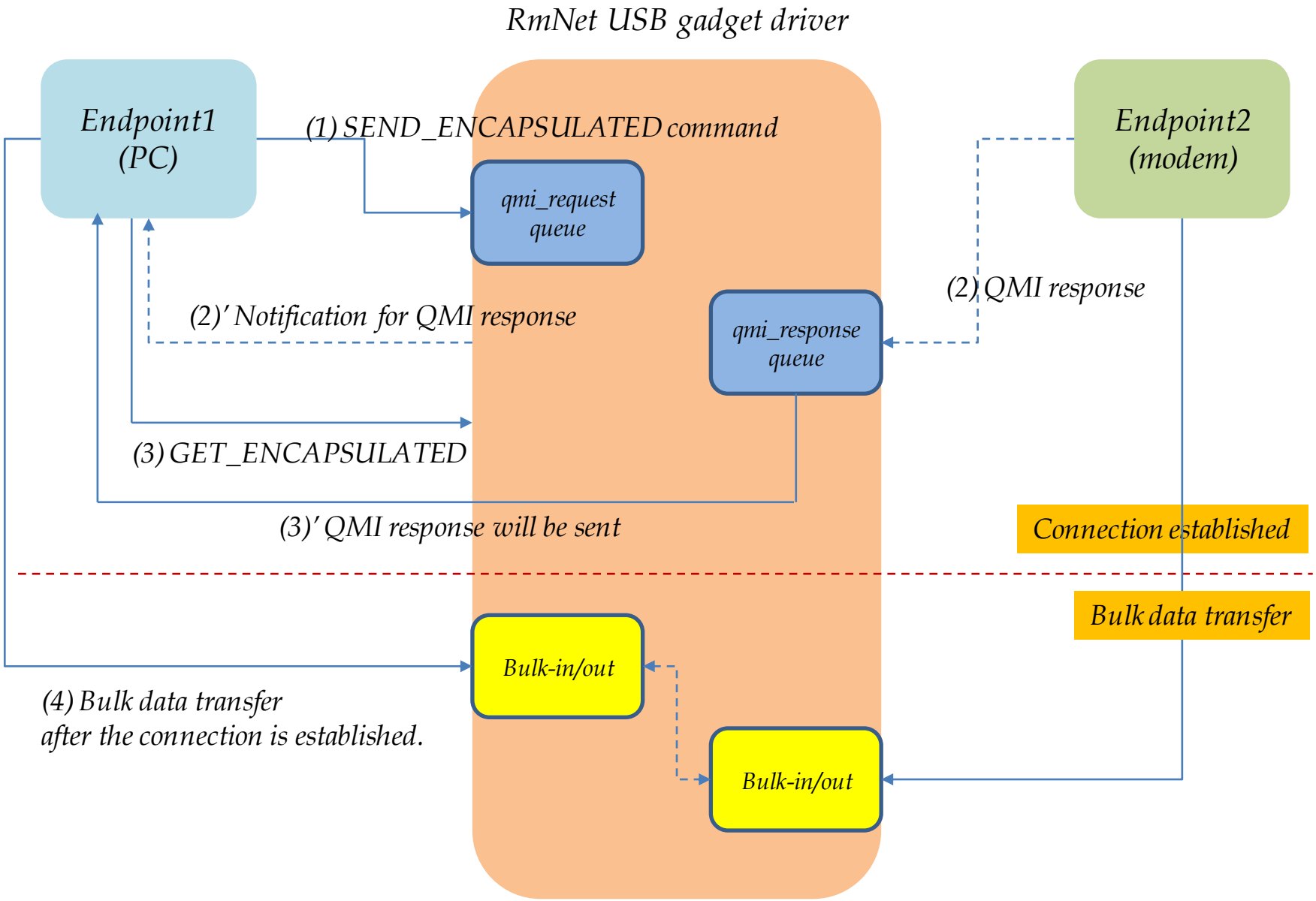
- *device_create_file(d, &dev_attr_timeout)*
- *device_create_file(d, &dev_attr_wakeups_xmit)*
- *device_create_file(d, &dev_attr_wakeups_rcv)*
- *device_create_file(d, &dev_attr_timeout_suspend)*

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

<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

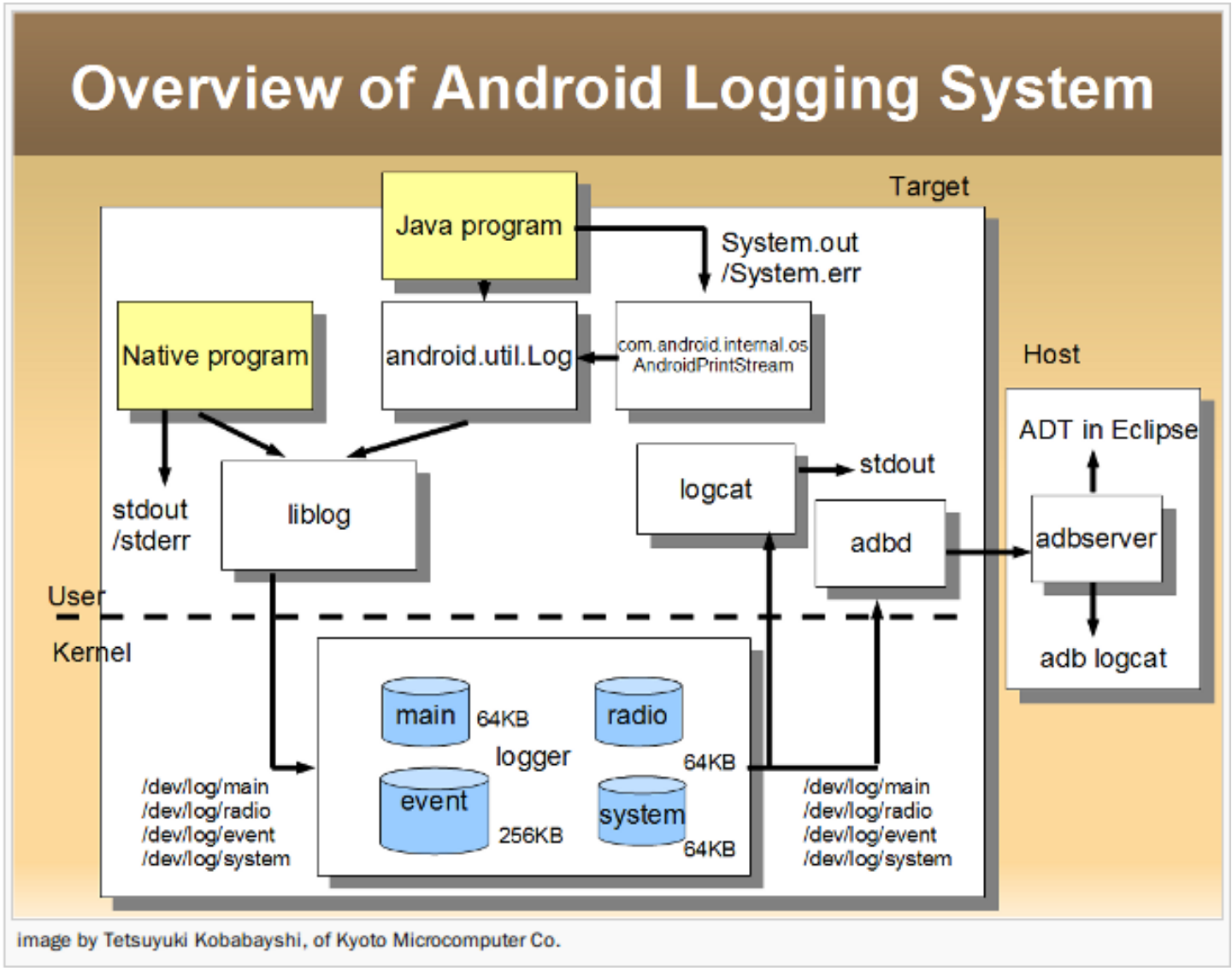
9. GPS Driver

1. GPS Driver Overview

- ***<TODO>***

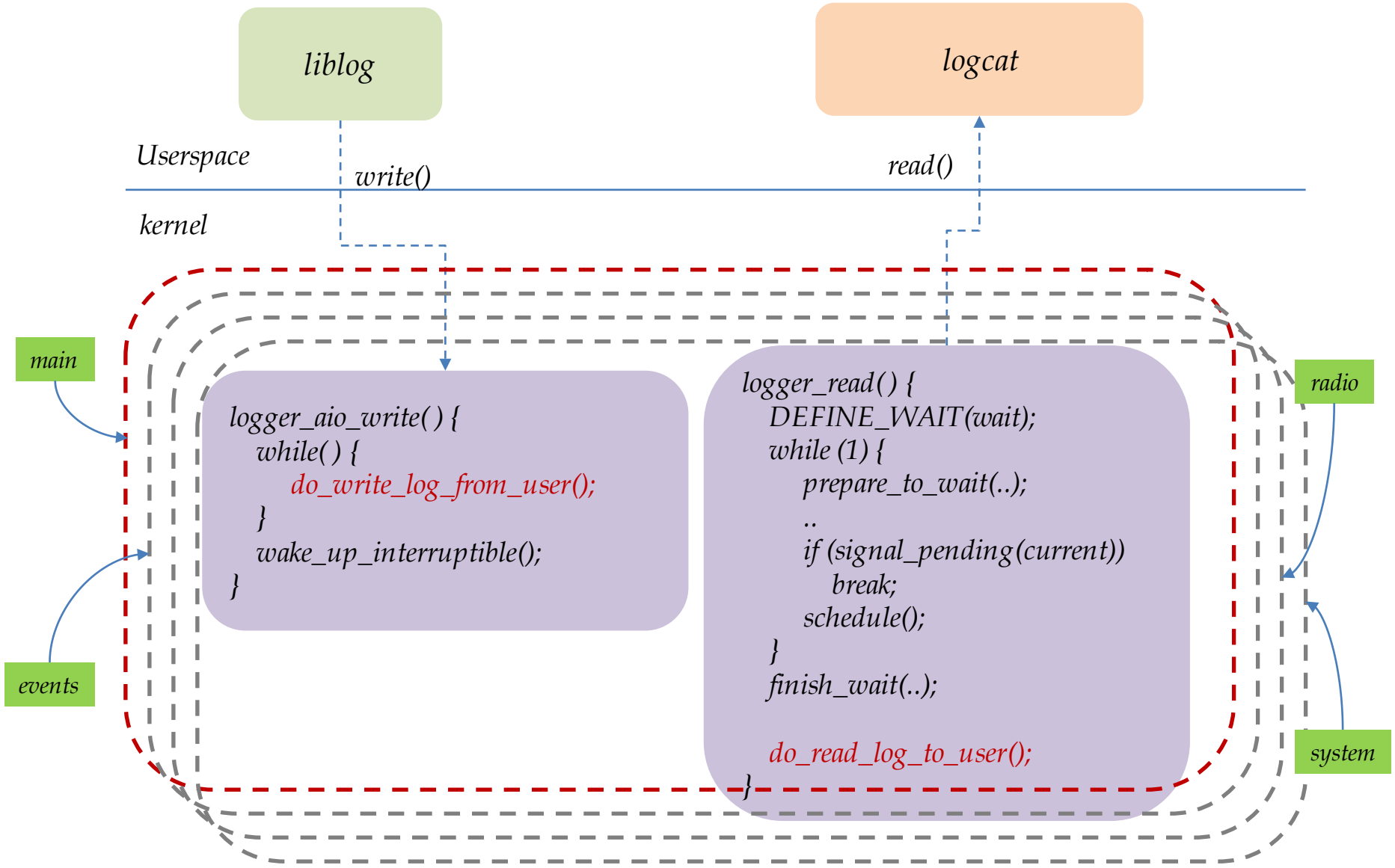
부록 1. Android Logging System

1. Android Logging System Overview



2. Android Logger Driver(1)

(*) logger driver는 다음 4개를 misc driver 형태로 등록하고 있음.
➔ log_main, log_events, log_radio, log_system



2. Android Logger Driver(2)

- 1) Android Logger Driver는 linux kernel log system과는 별도로 android log를 저장 관리하기 위한 kernel misc driver이다. 이 드라이버는 android log의 종류 별로 각각 아래와 같은 네가지의 log buffer를 보유하고 있다.
  - *Main – main application log*
  - *Events – system event information*
  - *Radio – radio 및 phone 관련 information*
  - *System – low level system message 및 debugging information*
- 2) User application에서는 /dev/log 아래의 4가지 device file을 open하여 logger driver에 접근할 수 있다.
  - *main, events, radio, system*
- 3) 동작 방식은 매우 단순한데 …
  - *3-1) liblog library를 통해 android log를 원하는 kernel buffer에 write한다. Write operation 후에는 logcat으로 부터의 read operation이 원할이 진행되도록 wake_up_interruptible() 함수를 호출한다.*
  - *3-2) logcat은 driver open 후, read 함수를 통해 원하는 log buffer의 내용을 읽어들인다. 단, 이때 아직 log buffer의 내용이 준비되어 있지 않을 수 있으므로(혹은, log를 write하는 중일 수 있으므로), wait queue에 대기하고 있다가, wake_up signal이 도착할 경우, wait queue를 빠져나와 buffer의 내용을 읽기 시작한다.*

(*) 보다 자세한 사항은 kernel/drivers/staging/android/logger.c 파일 및 http://elinux.org/Android_Logging_System site를 참조하기 바람^^.

3. 몇 가지 Tips(1)

1) kernel log를 file에 writing하기(init.rc에 추가)

```
service klog /system/bin/dd if=/proc/kmsg of=/data/kernel.log bs=1
    oneshot
```

2) android log를 file에 writing하기

```
service logcat /system/bin/logcat –r 10000 –f /data/log.txt
    oneshot
    user  root
```

3) android log를 linux kernel log와 함께 출력하기

```
service logcat /system/bin/logcat –f /dev/kmsg
    oneshot
```

(*) 위의 방법은 유용할 수도 그렇지 않을 수도 있으나, 참고 삼아 정리한 것임^^.

3. 몇 가지 Tips(2)

- 1) kernel에서 pr_debug()함수의 내용이 출력되지 않을 때
  - *C file의 맨 위에 "#define DEBUG"를 추가하면 됨.*
  - *이유는 include/linux/kernel.h 파일 참조*

- 2) Android에서 LOGE, LOGW, LOGI, LOGD method의 내용이 출력되지 않을 때
  - *C/C++ 파일의 맨 위에 "#define LOG_NDEBUG 1" 를 추가하면 됨.*
  - *이유는* system/core/include/cutils/log.h 파일 참조

부록 2. Android Build System

1. Android ICS build 절차 – *envsetup.sh & make*

<단계1>

 # . build/envsetup.sh

1) toolchain path 등 개발 환경 설정

2) 기타 아래의 몇 가지 유용한 명령어 제공

m *- build tree의 top에서 부터 make(build) 시작*

➔ *모든 Android.mk 파일을 찾아서 하나로 만든 후, make를 돌림.*

mm *– 현재 directory 아래의 모든 module을 build*

mmm *– 파라미터로 지정한 directory 아래의 모든 module을 build*

➔ *mmm test*

➔ *mmm test snod* ←*test 디렉토리 build 후, system image 생성*

croot *– tree의 top으로 이동*

cgrep *– 모든 C/C++ file에 대해 특정 패턴을 grep하고자 할 때 사용함.*

jgrep *– 모든 java file에 대해 특정 패턴을 grep하고자 할 때 사용함.*

…

<단계2>

choosecombo

1) 바이너리를 upload할 대상을 선택한다(emulator, device 중에서)

2) Build mode(release, debug)를 선택한다.

3) Product model을 선택한다.

4) 구동 mode(user, userdebug, engineer)를 선택한다.

…

<단계3>

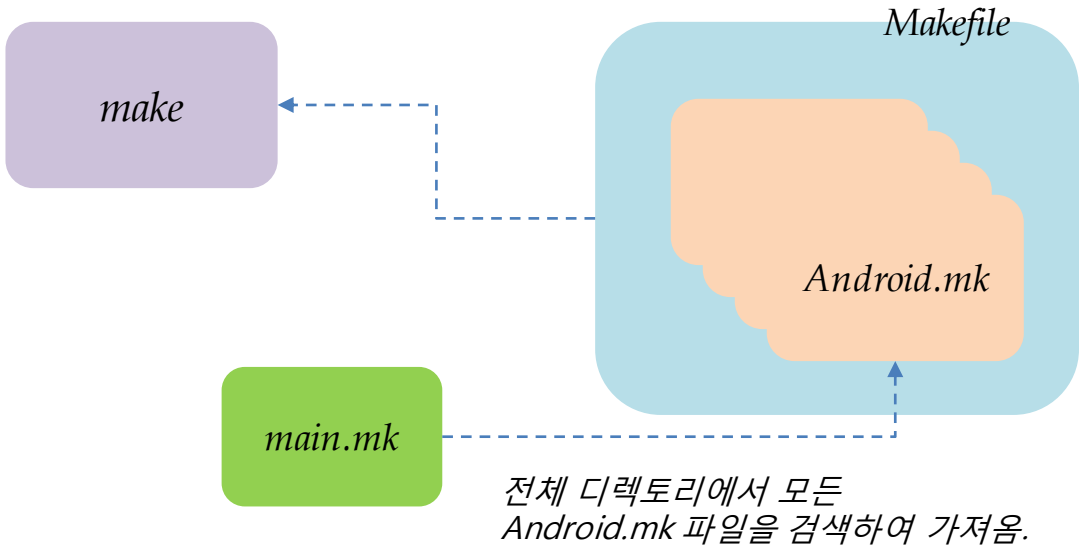
make ←***android/Makefile***

2. Android Build System Overview(1)

(*) android build system은 겉보기와는 달리, 일반적인 GNU make를 이용하여 구축되어 있다. 다만, 시스템 전체를 build하기 위한 관점에서 접근하고 있으며, 각각의 하위 모듈을 build하기 위해서 Android.mk라는 새로운 문법(sub makefile)을 정의하여 사용하고 있다.

<일반적인 GNU build system>

<Android GNU build system>

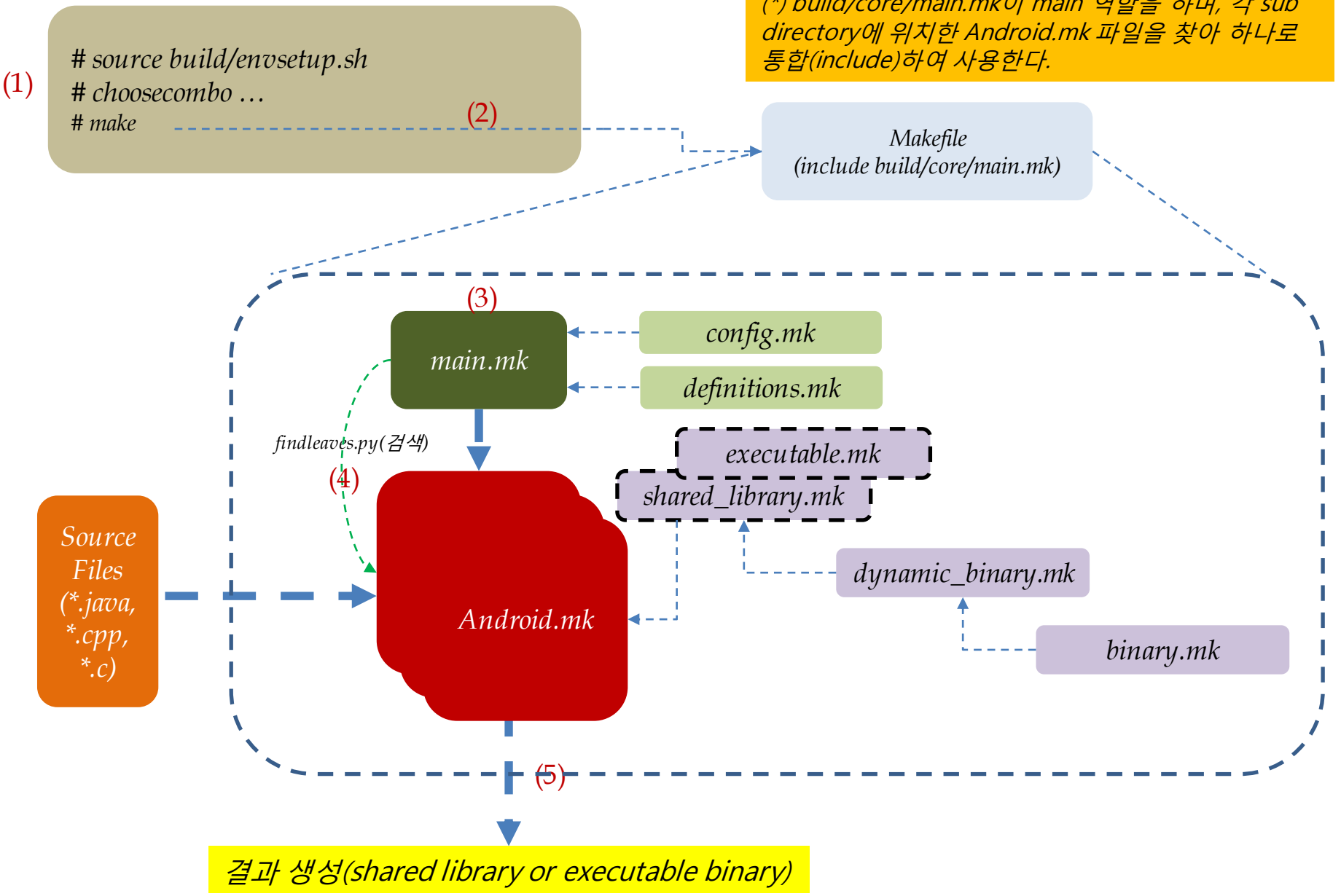


여러 개의 Android.mk 파일을 묶어 한 개의 거대한 Makefile 생성

전체 디렉토리에서 모든 Android.mk 파일을 검색하여 가져옴.

2. Android Build System Overview(2)

(*) Android.mk는 각각의 build 디렉토리에 있음.
(*) 아래에 기술되어 있는 모든 mk file은 build/core 디렉토리에 있음.
(*) build/core/main.mk이 main 역할을 하며, 각 sub directory에 위치한 Android.mk 파일을 찾아 하나로 통합(include)하여 사용한다.



2. Android Build System Overview(3) – *Android.mk 파일 문법*

<Android.mk 파일을 구성하는 field 설명>

➔ ***Android는 build를 위하여 GNU make를 사용하고 있으나, 앞서 기술한 것 처럼 자체 custom makefile을 운용하고 있다(그러나, 내부를 따라가 보면, 결국 기존 Makefile 형태임).***

1) **include $(CLEAR_VARS)**: build 관련 local 변수(아래 내용들)를 모두 clear해 줌.
   ➔ *build/core/clear_vars.mk 파일이 include될 것임.*
   ➔ *이 파일을 보면, Android.mk file에서 사용 가능한 local 변수를 확인할 수 있음.*

2) LOCAL variables

2-1) ***LOCAL_MODULE****: build하려는 module의 이름(결과 파일명)*
2-2) ***LOCAL_SRC_FILES****: build하려는 source 파일들*
2-3) ***LOCAL_STATIC_LIBRARIES****: 이 module(결과물)에 static하게 link하는 libraries*
2-4) ***LOCAL_SHARED_LIBRARIES****: 이 module에 link되는 shared libraries*
2-5) ***LOCAL_C_INCLUDES****: include file을 위한 path 지정(가령: $KERNEL_HEADERS)*
2-6) ***LOCAL_CFLAGS****: compiler에게 전달하는 추가 CFLAGS 지정*
2-7) ***LOCAL_LDFLAGS****: linker에게 전달하는 추가 LDFLAGS 지정*

3) Include BUILD rules

3-1) ***include $(BUILD_EXECUTABLE)***
   ➔ *실행파일을 build하는 rule이 추가됨(여기에서 build함 – 기존 Makefile format)*
   ➔ *build/core/executable.mk 파일이 include될 것임.*
3-2) ***include $(BUILD_SHARED_LIBRARY)****: shared library를 build하는 rule이 추가됨.*
3-3) ***include $(BUILD_STATIC_LIBRARY)****: static library를 build하는 rule이 추가됨.*
3-4) ***include $(BUILD_PREBUILT)****: prebuilt file을 복사하는 rule이 추가됨.*

2. Android Build System Overview(4) – *Android.mk 예(1)*

```
LOCAL_PATH:= $(call my-dir)

include $(CLEAR_VARS)
#Name of target to build
LOCAL_MODULE:= libmylibrary
#Source files to compile
LOCAL_SRC_FILES:= mysrcfile.c mysothersrcfile.c
#The shared libraries to link against
LOCAL_SHARED_LIBRARIES := libcutils
#No special headers needed
LOCAL_C_INCLUDES +=
#Prelink this library, also need to add it to the prelink map
LOCAL_PRELINK_MODULE := true
include $(BUILD_SHARED_LIBRARY)

#Clear variables and build the executable
include $(CLEAR_VARS)
LOCAL_MODULE:= myinfocmd
LOCAL_SRC_FILES:= mycmdsrcfile.c
include $(BUILD_EXECUTABLE)
```

libmylibrary.so를 만드는 예 (shared library)

myinfocmd 실행파일을 만드는 예

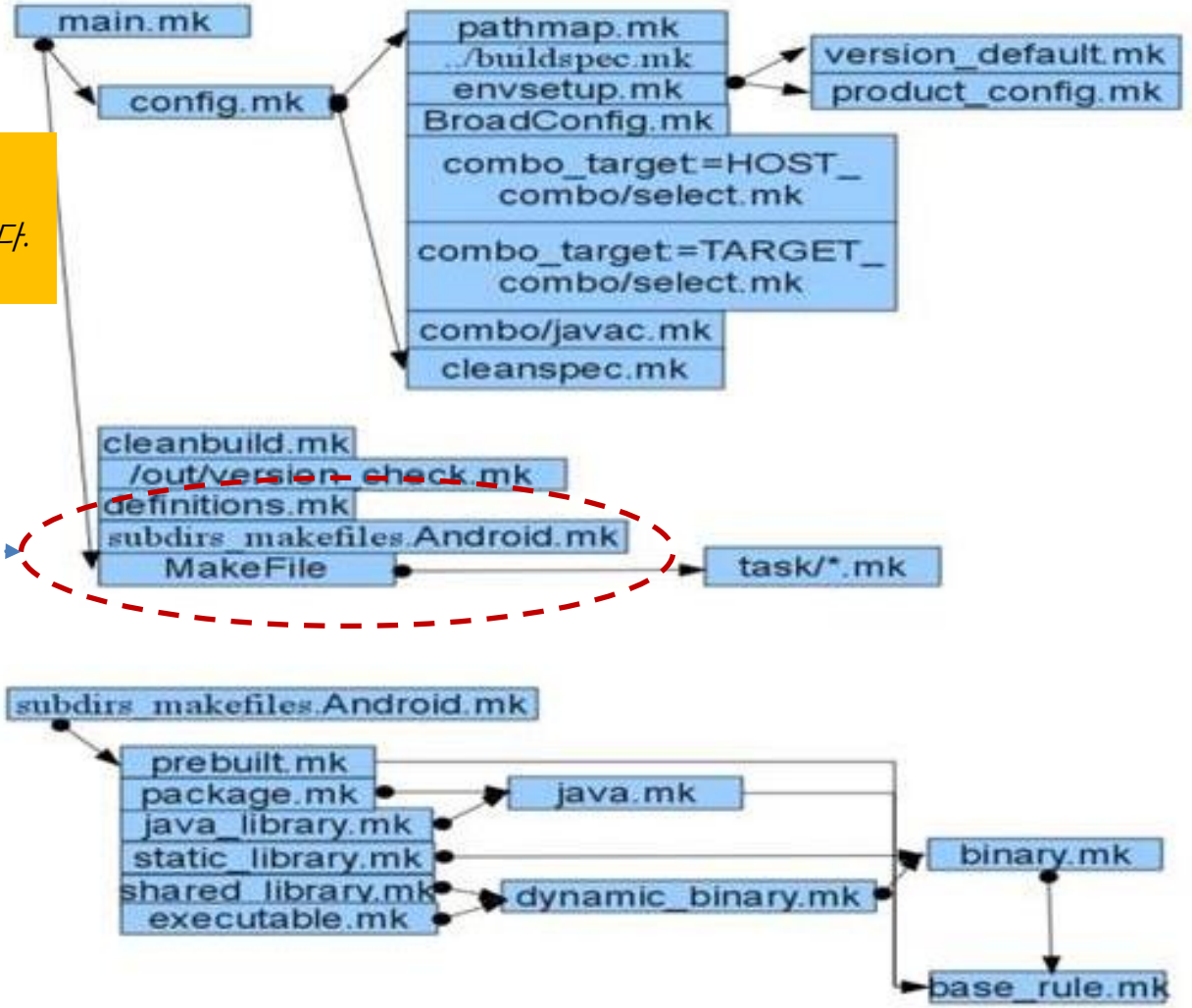
2. Android Build System Overview(4) – *Android.mk 예(2)*

```
LOCAL_PATH := $(call my-dir)
include $(CLEAR_VARS)

# Build all java files in the java subdirectory
LOCAL_SRC_FILES := $(call all-subdir-java-files)

# Name of the APK to build
LOCAL_PACKAGE_NAME := LocalPackage

LOCAL_CERTIFICATE := vendor/example/certs/app

# Tell it to build an APK
include $(BUILD_PACKAGE)
```

LocalPackage.apk 를 만드는 예

3. build/core/main.mk 파일 분석

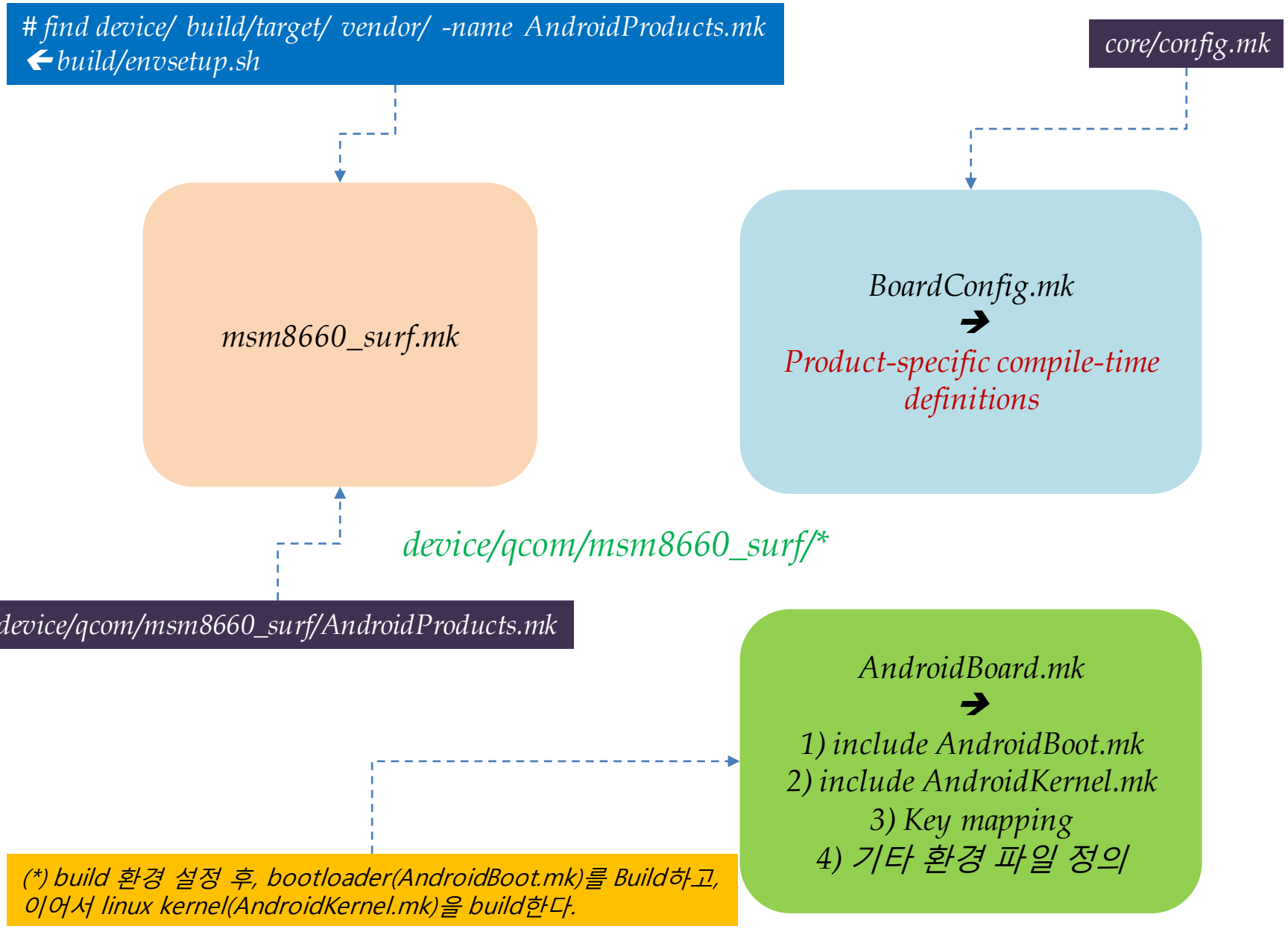
build/tools/findleaves.py script를 사용하여 각 폴더의 첫번째 Android.mk를 모두 찾아서 include 시킨다.
*➔ 하나의 거대한 Makefile을 만듦^^*



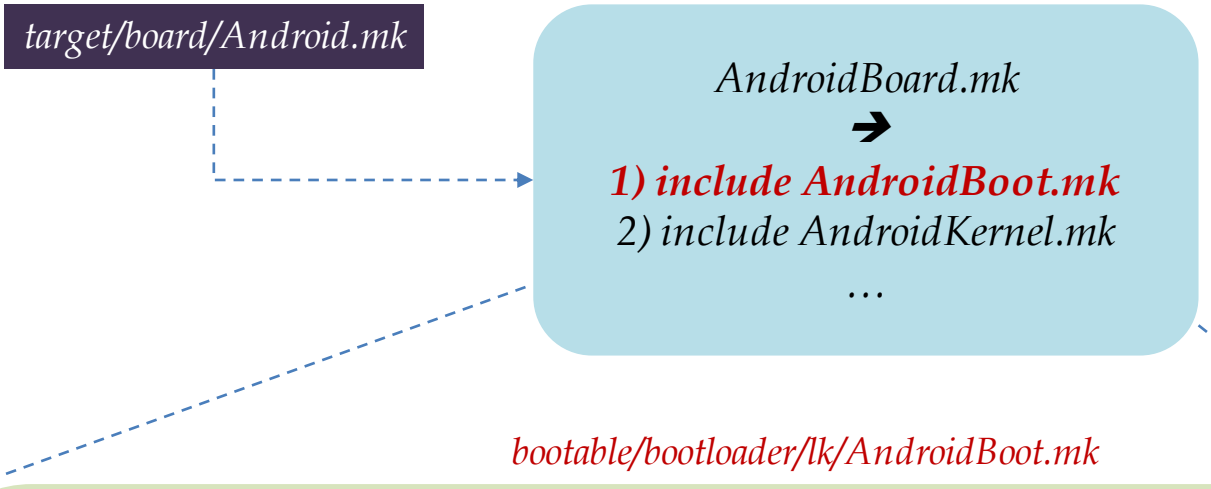
4. build/core/*.mk 주요 파일 정리: ***TODO***

- *1) config.mk*
- *2) definitions.mk*
- *3) envsetup.mk*
- *4) base_rules.mk*
- *…*

5. Target Board 파일 분석



6. Bootloader Makefile 파일 분석(1)



bootable/bootloader/lk/AndroidBoot.mk

➔ bootloader build Makefile
➔ NAND flash 용 output file: appsboot.mbn
➔ eMMC 용 output file: emmc_appsboot.mbn
➔ lk/ 디렉토리(현재 디렉토리) 아래의 makefile 을 이용하여 build 진행함.
➔ makefile 에서는 아래의 파일을 include 하여 사용하고 있음.

```
include project/$(PROJECT).mk
include target/$(TARGET)/rules.mk
include target/$(TARGET)/tools/makefile
include platform/$(PLATFORM)/rules.mk
include arch/$(ARCH)/rules.mk
include platform/rules.mk
include target/rules.mk
include kernel/rules.mk
include dev/rules.mk
include app/rules.mk
include make/module.mk
```

<bootable/bootloader/lk 디렉토리 내용 개략 정리>

1) bootable/bootloader/lk/project/msm8660_surf.mk*
   => project makefile 위치

2) bootable/bootloader/lk/app/aboot*
   => aboot.c : flash/mmc에서 kernel image 읽어 kernel loading 및 start하는 코드
   => fastboot.c: usb cable이용한 fastboot 처리 코드
   => recovery.c: recovery 관련 코드

3) bootable/bootloader/lk/arch*
   => arm/ 디렉토리 아래에 실제 arm 환경에서의 boot 관련 코드 위치함.
   (assembly code 다수 존재함)
   => 아래 (5)의 main(kmain) code를 호출하는 부분 존재함.

4) bootable/bootloader/lk/dev
   => device driver(fbcon, keys, net, pmic, ssbi, usb)

5) bootloader/lk/kernel*
   => boot main(kmain) 이 위치함.
   => kernel의 특성에 해당하는 코드(mutex, thread, timer, event ..)

6) bootloader/lk/lib
   => lk에서 사용하는 library codes

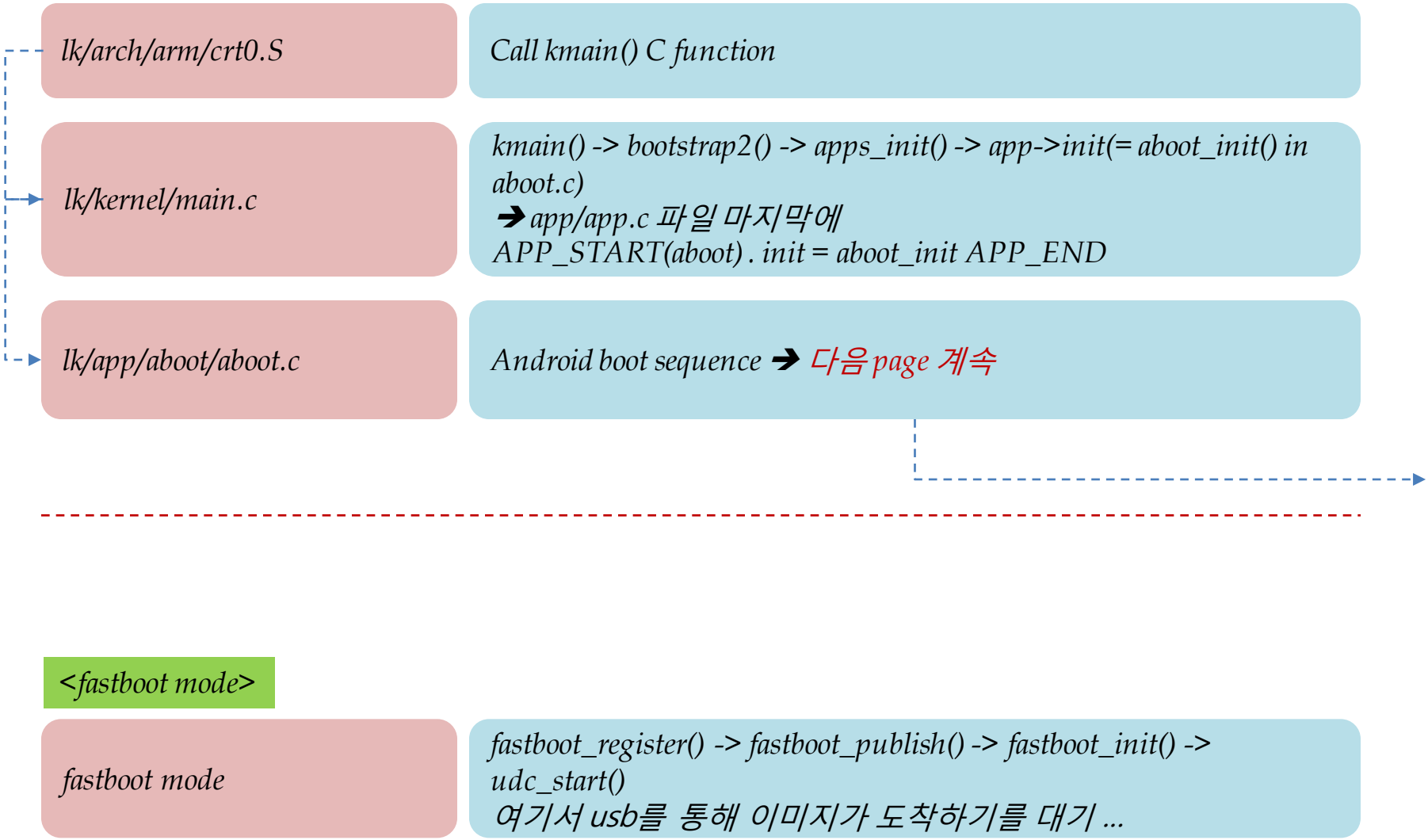
7) bootable/bootloader/lk/make

8) bootable/bootloader/lk/platform/msm8x60*
   => platform specific codes(acpuclock, gpio, hdmi, lcd panel, pmic, pmic battery alarm ...)

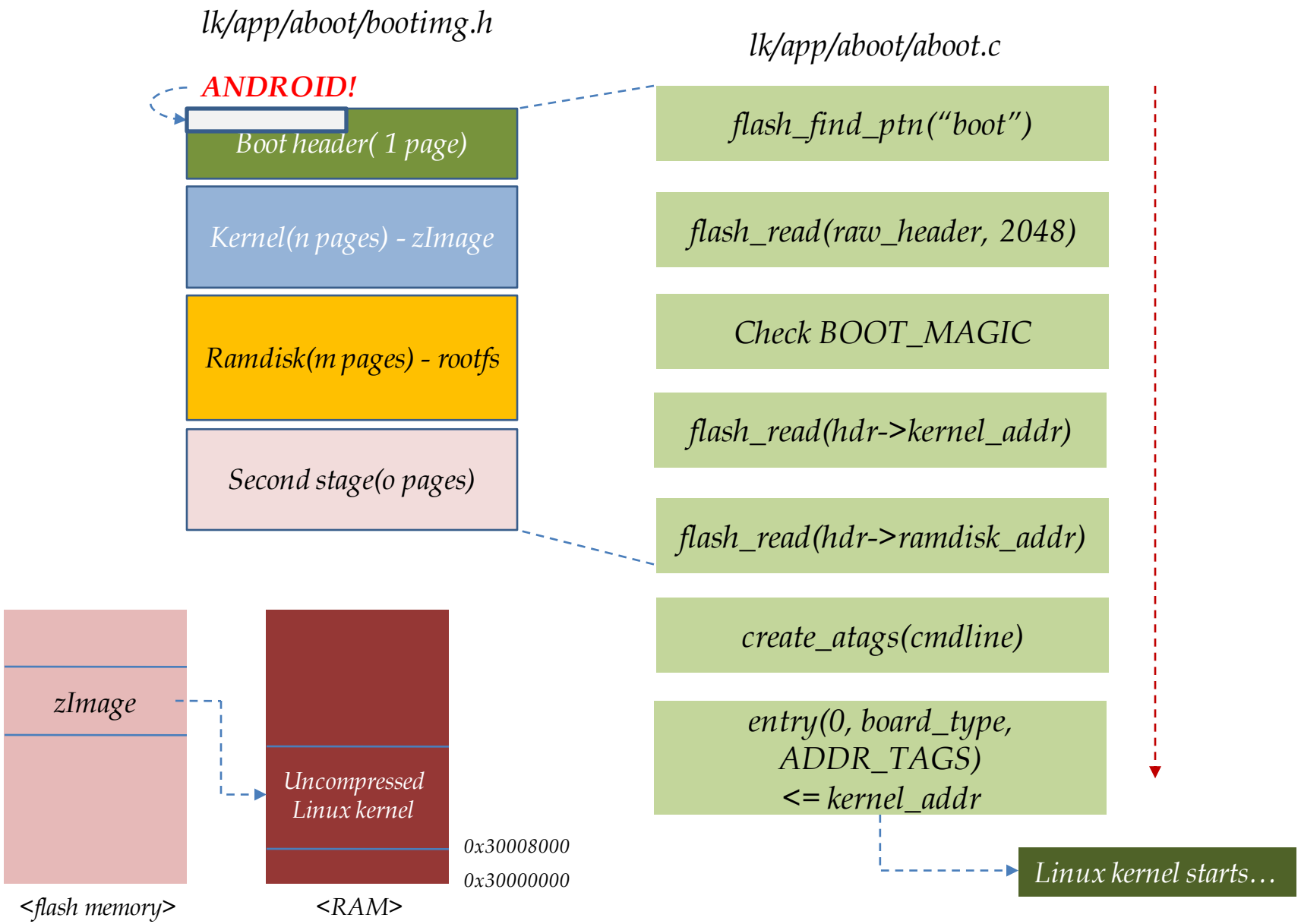
9) bootable/bootloader/lk/scripts
   => 몇가지 script 위치함

10) bootable/bootloader/lk/target*
   => msm8660_surf/ 디렉토리 아래에 target board와 관련한 몇가지 초기화 관련 코드 위치함.
   => target_init( ) 함수는 위의 (5)의 kmain( ) 함수에서 호출함.

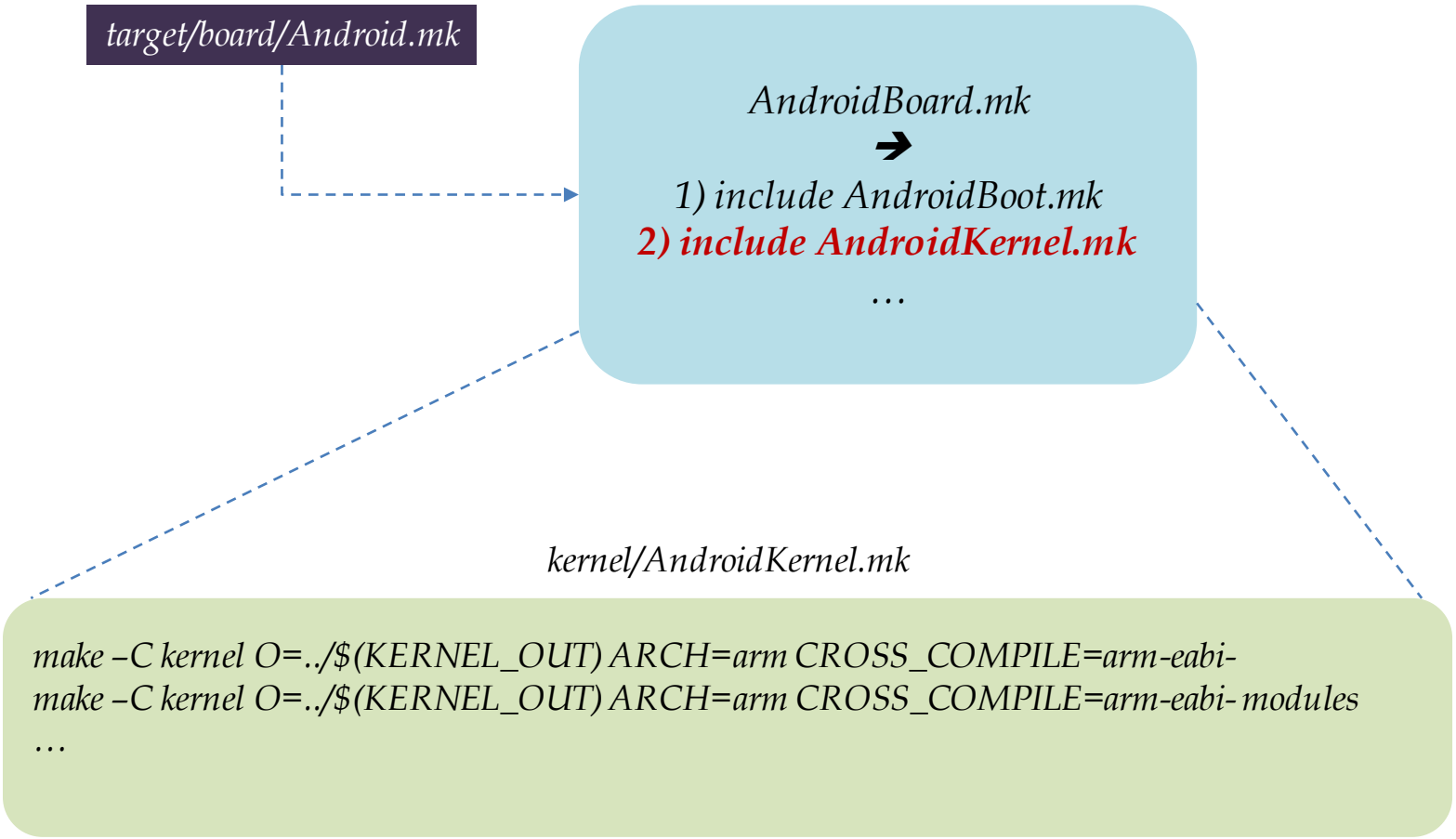
6. Bootloader Makefile 파일 분석(2) – ***Boot Sequence***



6. Bootloader Makefile 파일 분석(2) – ***Boot Sequence***



7. Kernel Makefile 파일 분석



References

- *1) Essential Linux Device Drivers …… [Sreekrishnan Venkateswaran]*
- *2) Android_Device_Driver_Guide_simple.pdf …….. [by me]*
- *3) Android_ICS_Porting_Guide.pdf …….. [by me]*
- *4) 안드로이드와 디바이스드라이버 적용 기법 … [FALINUX 유영창]*
- *5) Wi-Fi P2P 기술 분석(Understanding Wi-Fi P2P Technical Specification) … [ETRI]*
- *6) Wi-Fi Direct 기술 파급효과와 시사점 ……... [KT 종합기술원]*
- *7) Some Internet Articles …*

Thanks a lot !

SlowBoot