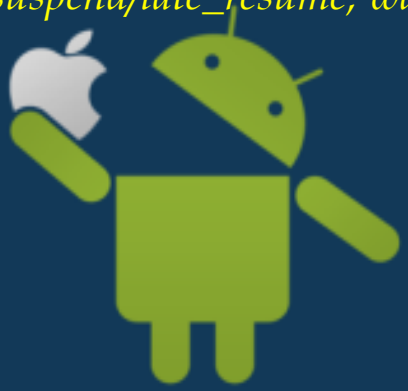
Android Power Management Hacks

: suspend/resume, early_suspend/late_resume, wakelock, battery service …

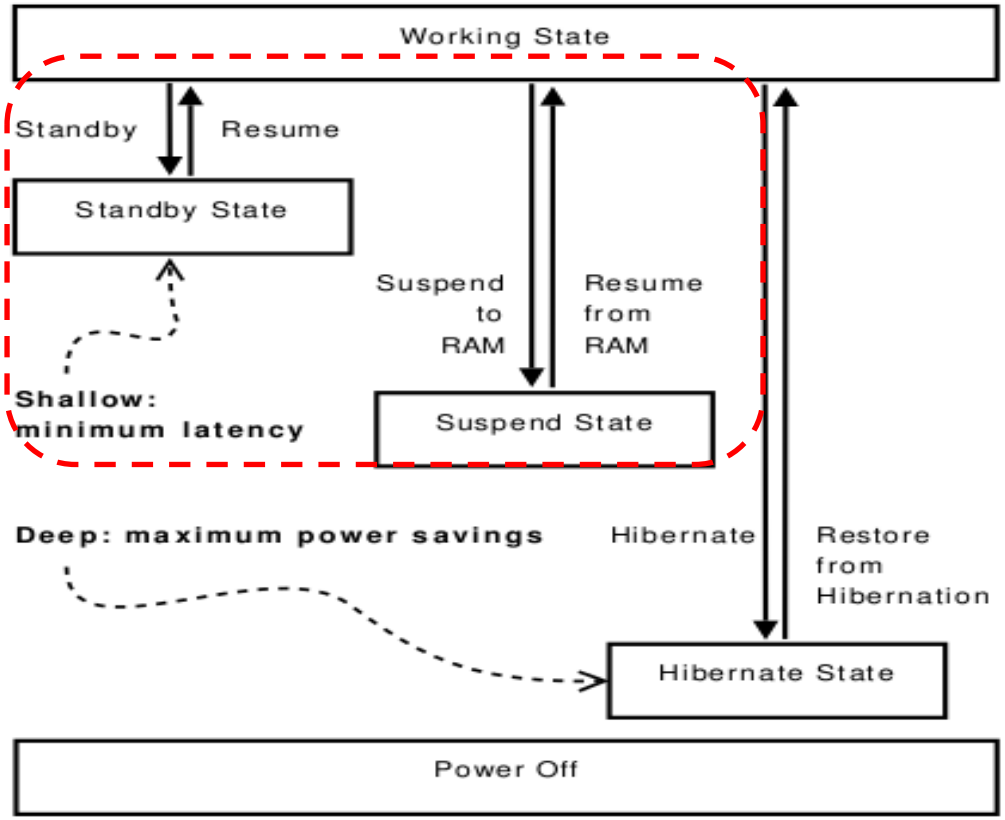


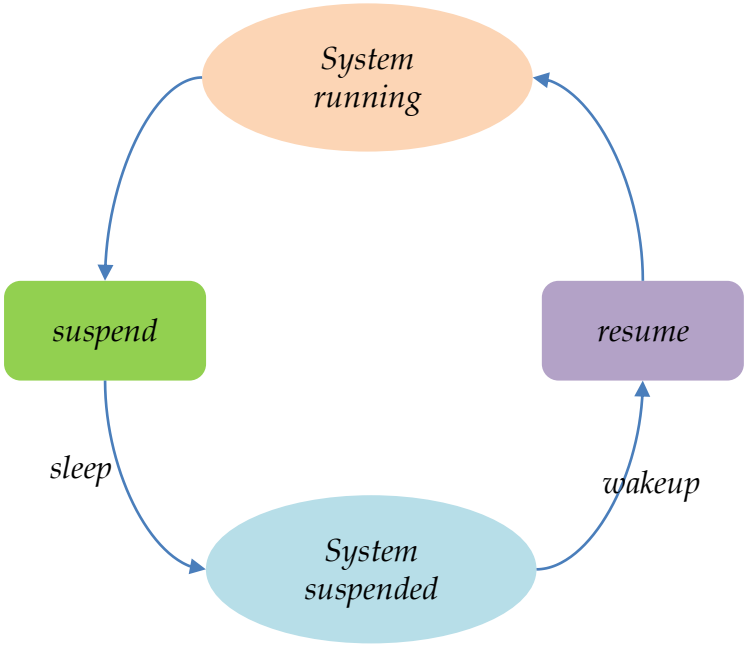
chunghan.yi@gmail.com, slowboot

Revision	작성자	비고
0.1	이 충 한	최초 작성 12/20/2011
0.6	이 충 한	12/21/2011 (1차 보강)
0.7	이 충 한	12/22/2011 (wakelock 보강)
0.8	이 충 한	12/26/2011 (정리)

목차

1. Linux Power Management(1) - *Overview*





(*) suspend/resume 관련 내용은 아래 파일에서 확인할 수 있다.
1) kernel/power/main.c
2) kernel/power/suspend.c
3) arch/arm/mach-xxx/pm..c
*4) drivers/base/power/**

1. Linux Power Management(2) – ***PM Core***

Power Management Core

➔ *Kernel 내에서 suspend/resume 제어 담당*
➔ *kernel/power/main.c*
➔ *Application 에서 아래 명령 수행으로 suspend 시작됨*
 # echo mem > /sys/power/state
➔ *main.c의 state_store() 함수내의 enter_state( ) 함수에 의해 시작 !!!*

struct platform_suspend_ops

➔ *arch/arm/mach-xxx/pm.c*

코드 예)

```
struct platform_suspend_ops msm_pm_ops = {
        .enter = msm_pm_enter,
        .valid = suspend_valid_only_mem,
};
```

platform_driver
or
device_driver

```
struct platform_suspend_ops {
    int (*valid)(suspend_state_t  state);
    int (*begin)(suspend_state_t  state);
    int (*prepare)(void);
    int (*prepare_late)(void);
    int (*enter)(suspend_state_t  state);
    void (*wake)(void);
    void (*finish)(void);
    void (*end)(void);
    void (*recover)(void);
};
```

```
struct platform_driver {
    int (*probe)(struct platform_device *);
    int (*remove)(struct platform_device *);
    int (*shutdown)(struct platform_device *);
    int (*suspend)(struct platform_device *, …);
    int (*suspend_late)(struct platform_device *, …);
    int (*resume_early)(struct platform_device *);
    int (*resume)(struct platform_device *);
};
```

1. Linux Power Management(3) – ***Power States***

Power States(/sys/power/state)

*1) **on***

➔ *켜져 있는 상태*

*2) **standby**(Standby)*

➔ *최소한도의 power saving 제공*

➔ *이 상태(standby)에서 ON 상태로 전환하는데, 1-2초 정도 소요*

*3) **mem**(**Suspend-to-RAM**)*

➔ *중요한 power saving 제공*

➔ *이 상태(Suspend-to-RAM)에서 ON 상태로 전환하는데, 3-5초 정도 소요*

4) disk(Suspend-to-Disk)

➔ *가장 많은 power saving 제공*

➔ *이 상태(Suspend-to-Disk)에서 ON 상태로 전환하는데, 대략 30초 소요*

Power State 확인

```
# cat /sys/power/state
```

Power State 변경

```
# echo mem > /sys/power/state
# echo standby > /sys/power/state
```

1. Linux Power Management(4) – ***Suspend/Resume***

Suspend 절차:

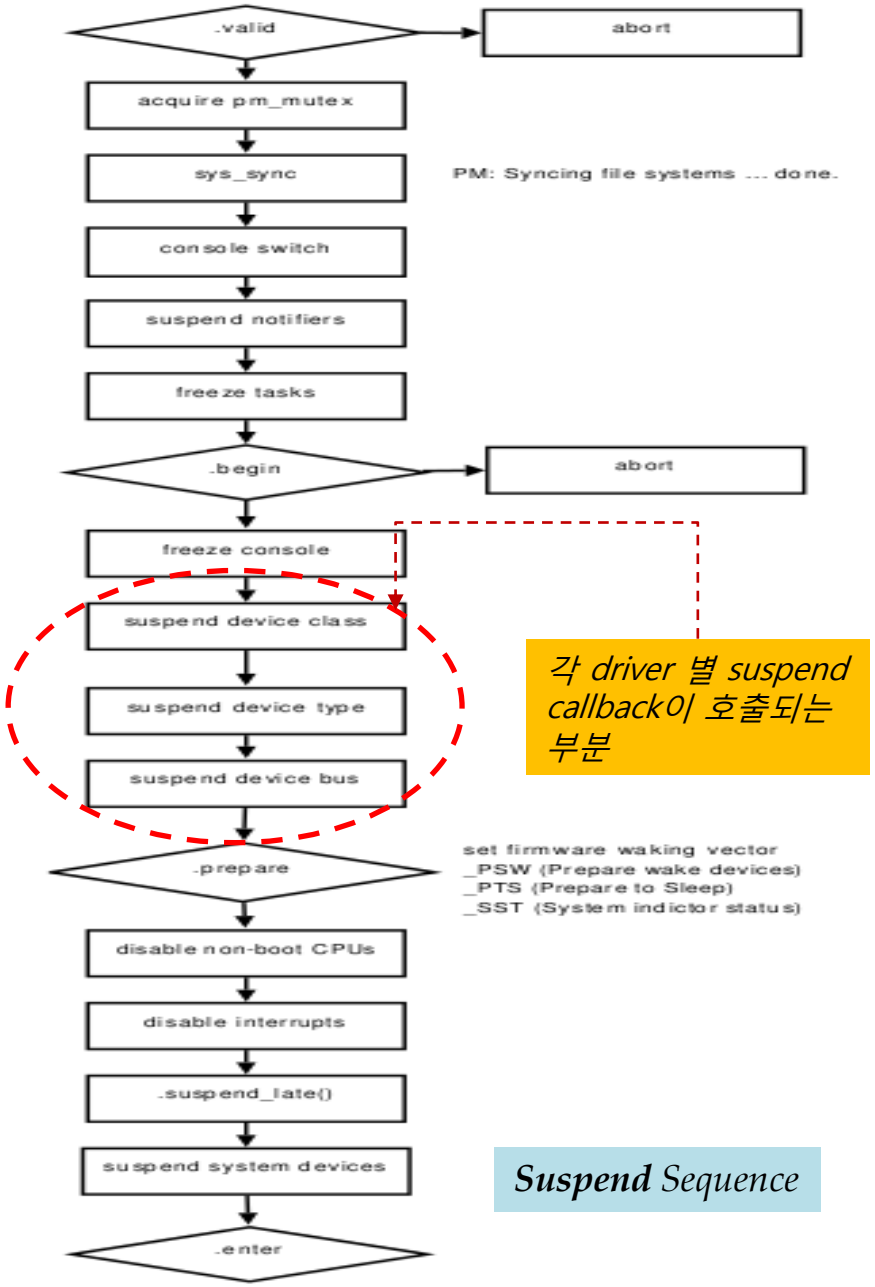
) **application으로 부터 "echo mem > /sys/power/state" 형태로 suspend 시작*

1) 프로세스와 task를 freezing 시키고,
2) 모든 device driver의 suspend callback 함수 호출
3) CPU와 core device를 suspend 시킴

Resume 절차:

) **interrupt 등에 의해 시작됨(예: power key 누름, 전화 걸려옴 ...).*
따라서 이와 관련된 장치는 항상 wakeup 상태로 있어야 함.

1) System 장치(/sys/devices/system)를 먼저 깨우고,
2) IRQ 활성화, CPU 활성화
3) 나머지 모든 장치를 깨우고(resume callback 함수 호출), freezing되어 있는 프로세스와 task를 깨움.


.valid
abort
acquire pm_mutex
sys_sync
PM: Syncing file systems ... done.
console switch
suspend notifiers
freeze tasks
.begin
abort
freeze console
suspend device class
suspend device type
suspend device bus
각 driver 별 suspend callback이 호출되는 부분
.prepare
set firmware waking vector
_PSW (Prepare wake devices)
_PTS (Prepare to Sleep)
_SST (System indictor status)
disable non-boot CPUs
disable interrupts
.suspend_late()
suspend system devices
.enter
Suspend Sequence

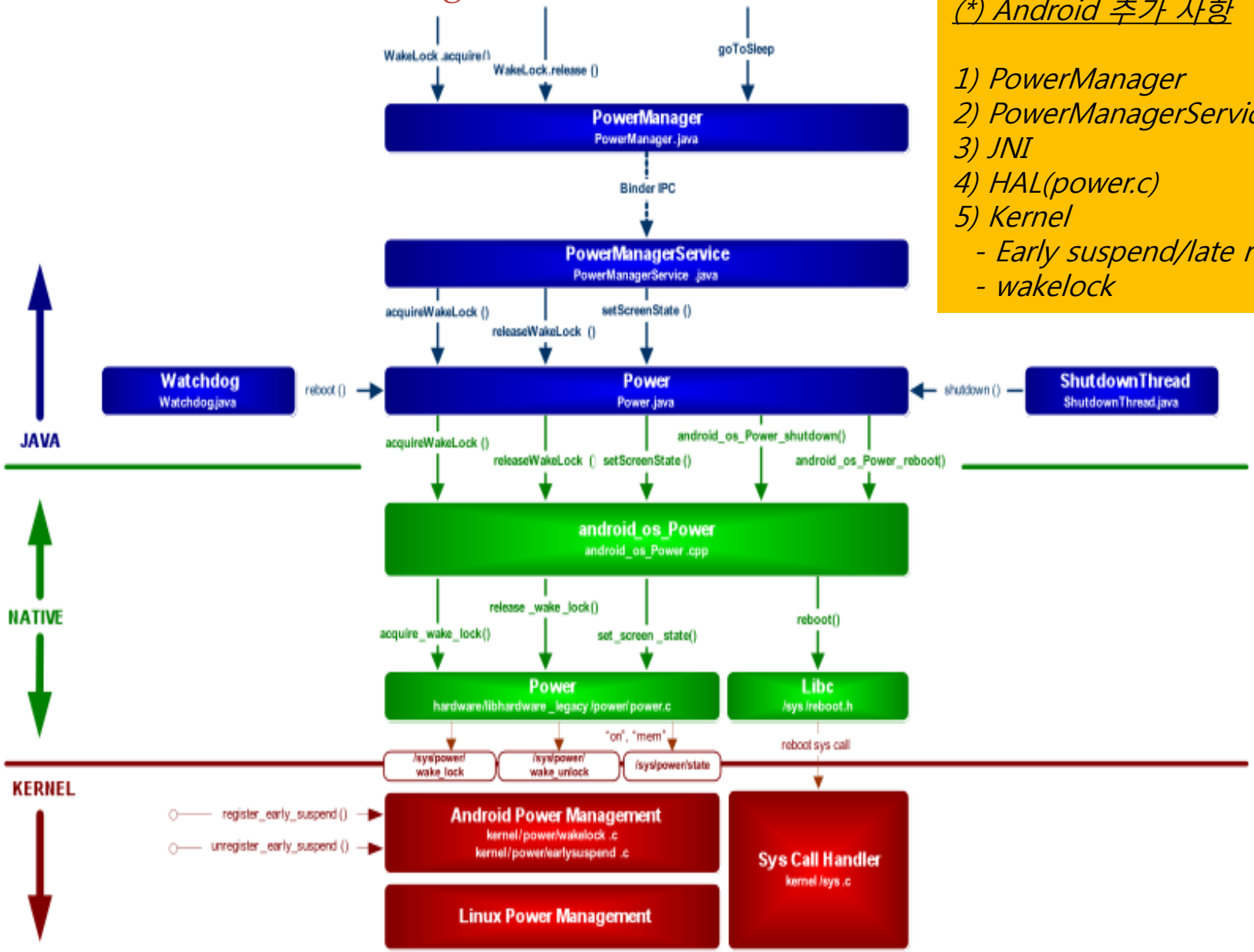

resume at wake vector
initialize CPU
.enter
.resume system devices
.resume_early()
resume non-boot CPUs
.finish
_WAK (wakup)
.resume device bus
.resume device type
.resume device class
각 driver 별 resume callback이 호출되는 부분
unblock console
.end
forget target state
thaw tasks
resume notifiers
console switch
release pm_mutex
Resume Sequence

<참고사항: platform_drivers 혹은 device_drivers의
suspend callback 함수 호출 flow 분석>

1) state_store( ) *kernel/power/main.c*
2) enter_state( ) *kernel/power/suspend.c*
➔ *android early suspend patch가 적용된 경우에는 진입 point가 request_suspend_state( ) 함수로 변경됨.*
3) suspend_devices_and_enter( ) *kernel/power/suspend.c*
4) dpm_suspend_start( ) *drivers/base/power/main.c*
5) dpm_suspend( )
➔ *"Execute <suspend> callbacks for all non-sysdev devices."*
6) device_suspend( )
7) __device_suspend( ) *drivers/base/power/main.c*
8) legacy_suspend(dev, state, ***dev->class->suspend****);*
…

(*) 사용자가 정의한 driver내의 suspend callback 함수는 대략 위의 순서를 따라 호출된다.
(*) 한편, platform_suspend_ops data structure는 arch/arm/mach-xxx/pm.c에서 초기화되며,
앞 페이지의 그림에서 처럼, 적당한 시점(?)에서 callback 함수(.valid, .enter, .begin 등)가 호출된다.

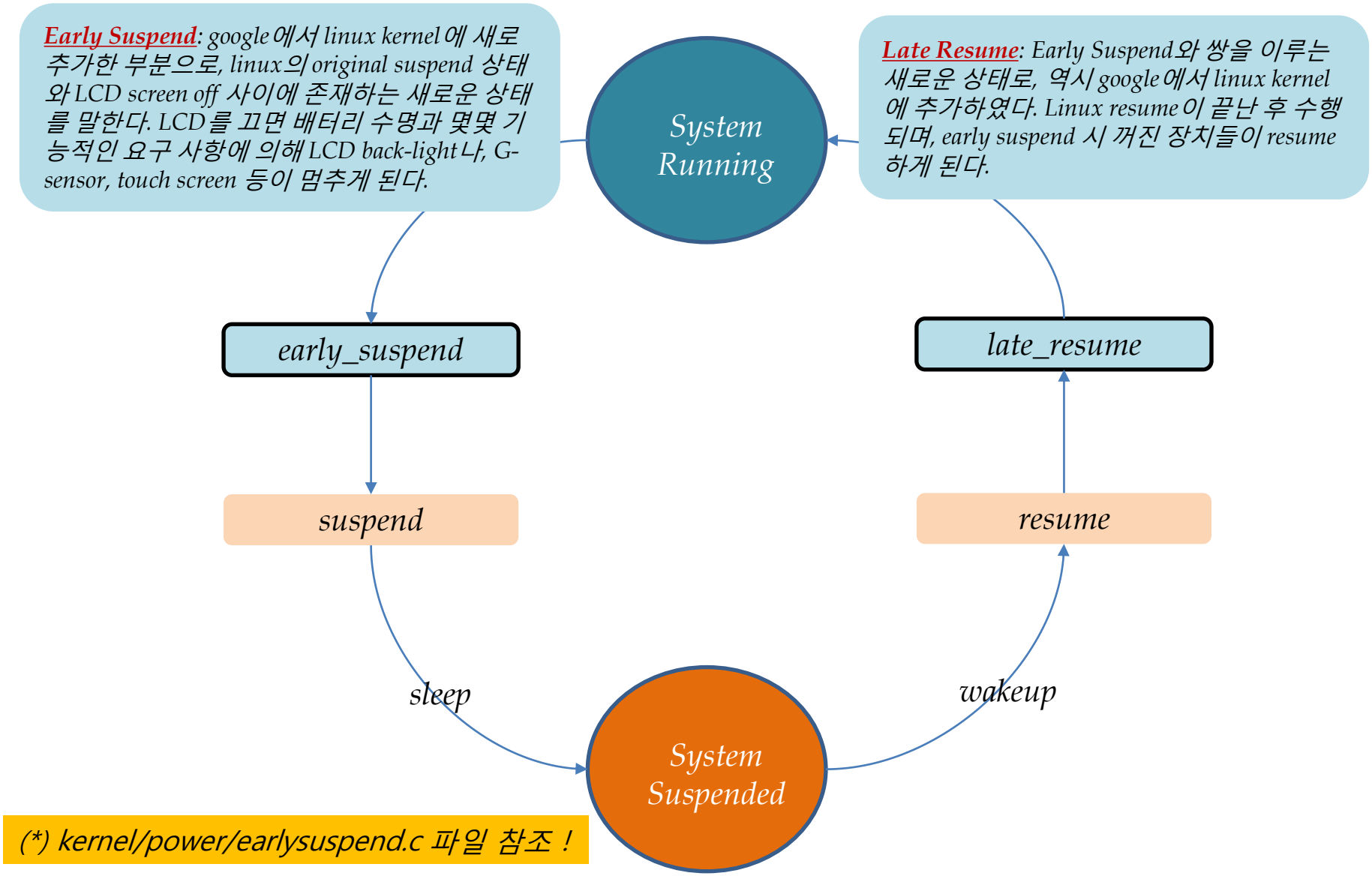
2. Android Power Management - *Overview*



(*) Android 추가 사항

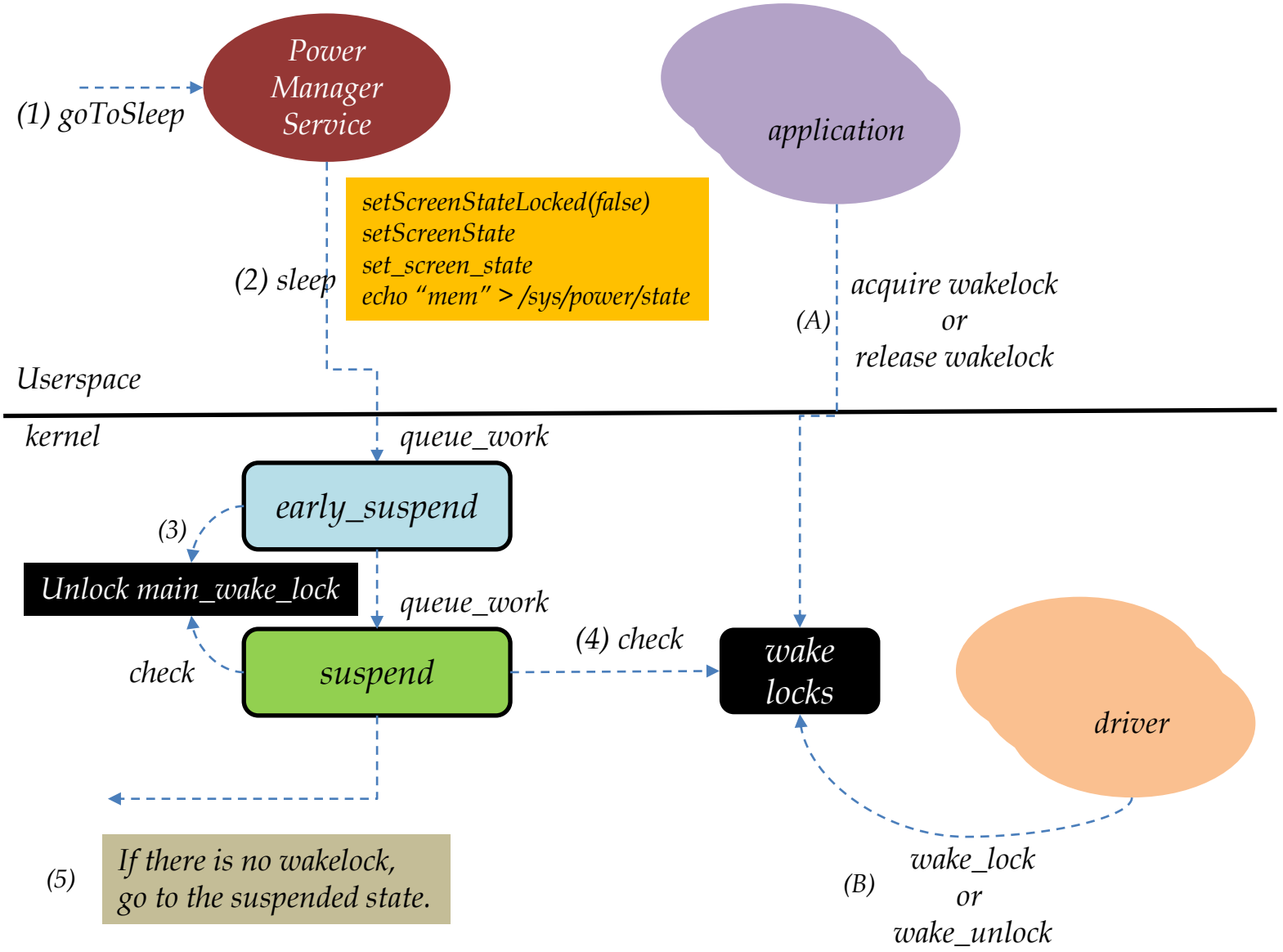
1) PowerManager
2) PowerManagerService
3) JNI
4) HAL(power.c)
5) Kernel
- *Early suspend/late resume*
- *wakelock*

2. Android Power Management – *Early Suspend/Late Resume*



(*) kernel/power/earlysuspend.c 파일 참조 !

2. Android Power Management – ***Early Suspend/1***



2. Android Power Management – *Early Suspend/2*

<Early Suspend & Suspend Step>

(A) Android application에서 application 수행 중, 임의의 시점에서 wakelock을 건다.
(B) Linux device driver에서 driver 동작 중, 임의의 시점에서 wakelock을 건다.

(1) Sleep 요청(goToSleep)이 들어온다.
(2) PowerManager -> PowerManagerService -> HAL(Power.c)을 거쳐 kernel로 suspend(=sleep) 요청이 내려간다.
- ➔ *화면이 off일 때, Power Manager Service는 early suspend를 요청한다.*
- ➔ *PARITAL_WAKE_LOCK을 제외한 wake lock이 잡혀 있을 경우에는 early suspend를 요청하지 않는다.*
- ➔ *"mem" 값을 "/sys/power/state"에 기록한다.*

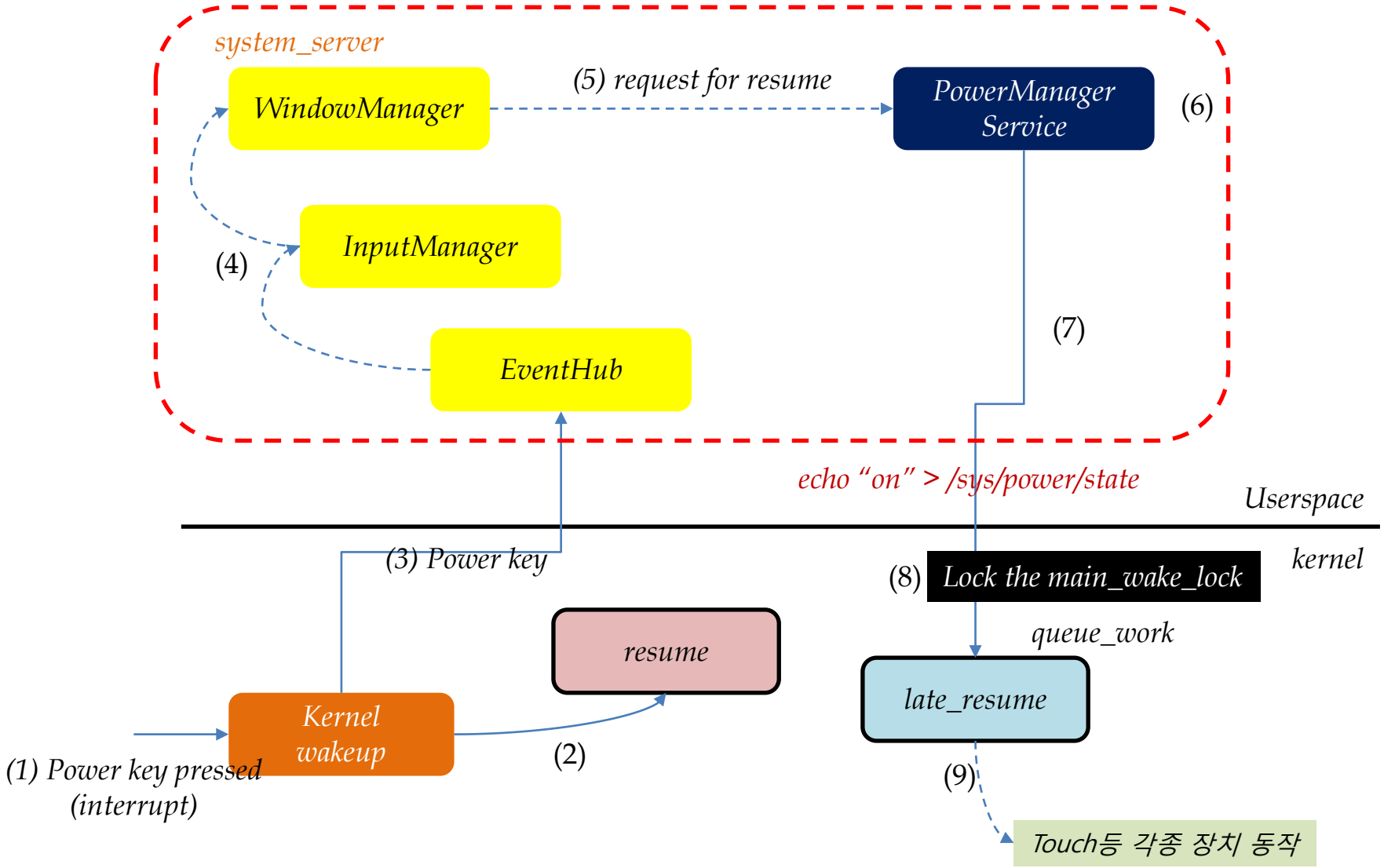
(3) early_suspend를 처리하기 위한 work(early_suspend_work)을 <suspend> work queue에 전달한다.

<work queue 내부 처리>
- ➔ *early_suspend( ) 함수 내에서는 사용자가 등록한 early_suspend callback 함수를 모두 호출한다.*
- ➔ *마지막으로 main_wake_lock을 풀어준다.*

(4) 현재 동작중인 wake lock이 존재하는지 확인(세 군데)하여, 없을 경우 suspend를 처리하기 위한 work(suspend_work)을 <suspend> work queue에 전달한다.
- ➔ *wake lock 존재 여부 검사는 timer를 이용하여 주기적으로 수행하는 것은 물론이고, wake_lock()/wake_unlock()함수 호출 시에도 행해진다. Wakelock이 존재하지 않을 경우, 이 곳에서 suspend work을 <suspend> work queue로 던지게 됨^^.*
- ➔ *Wake lock은 application 및 linux device driver에서 생성 가능하다.*
- ➔ *또한, suspend/resume을 관리하기 위한 main_wake_lock이 존재한다.*

(5) 시스템이 마침내 suspend 상태로 바뀐다.

2. Android Power Management – ***Late Resume/1***



2. Android Power Management – *Late Resume/2*

<Resume & Late Resume Step>

(1) 사용자가 power key를 누른다.

➔ *System을 resume 시키는 방법은 이 외에도 여러 가지가 있을 수 있음(ex: 전화가 온다).*

(2) (interrupt 처리와 유사하게) kernel이 resume 루틴을 구동시킨다.

➔ *이 부분은 좀 더 확인해 볼 필요가 있겠음^^*

(3) Power key는 input driver를 경유하여, android framework으로 전달된다.

(4) Android framework으로 전달된 power key는 다시 (system_server 내의)EventHub -> Input Manager -> Window Manager 순으로 차례로 전달된다.

(5) Window Manager는 power key를 인식하고, 이를 PowerManager -> PowerManagerService에게 전달한다.

➔ *Power key가 아닌 경우에는 버린다.*

(6) PowerManagerService는 early suspend와 유사한 처리 과정을 거쳐 kernel에 late resume 요청을 한다.

(7) 이 과정에서 "on" 값을 "/sys/power/state"에 기록한다.

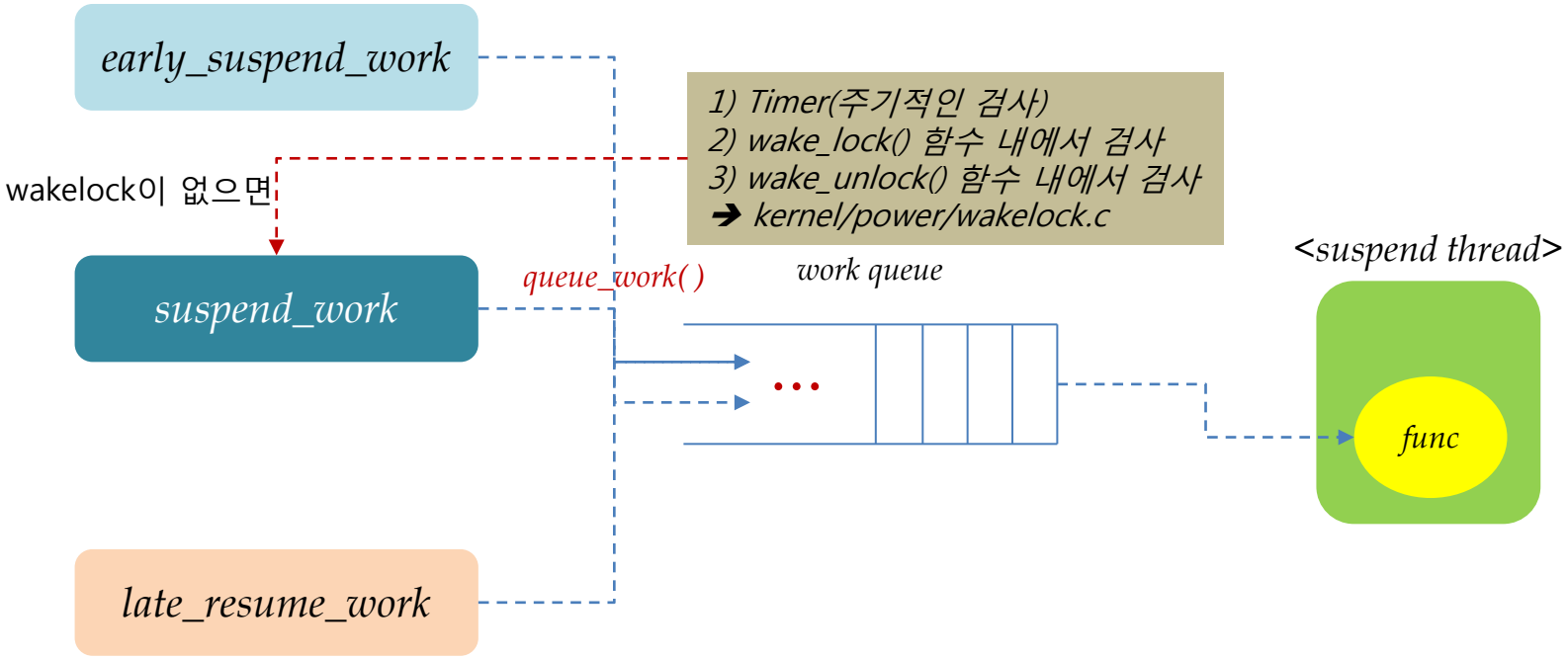
(8) late_resume을 처리하기에 앞서 "main_wake_lock"을 enable시킨다.

➔ *이 "main_wake_lock"은 early_suspend( ) 함수에서 다시 disable될 것이다.*

(9) Late resume을 처리하기 위한 work(late_resume_work)을 <suspend> work queue에 전달한다.

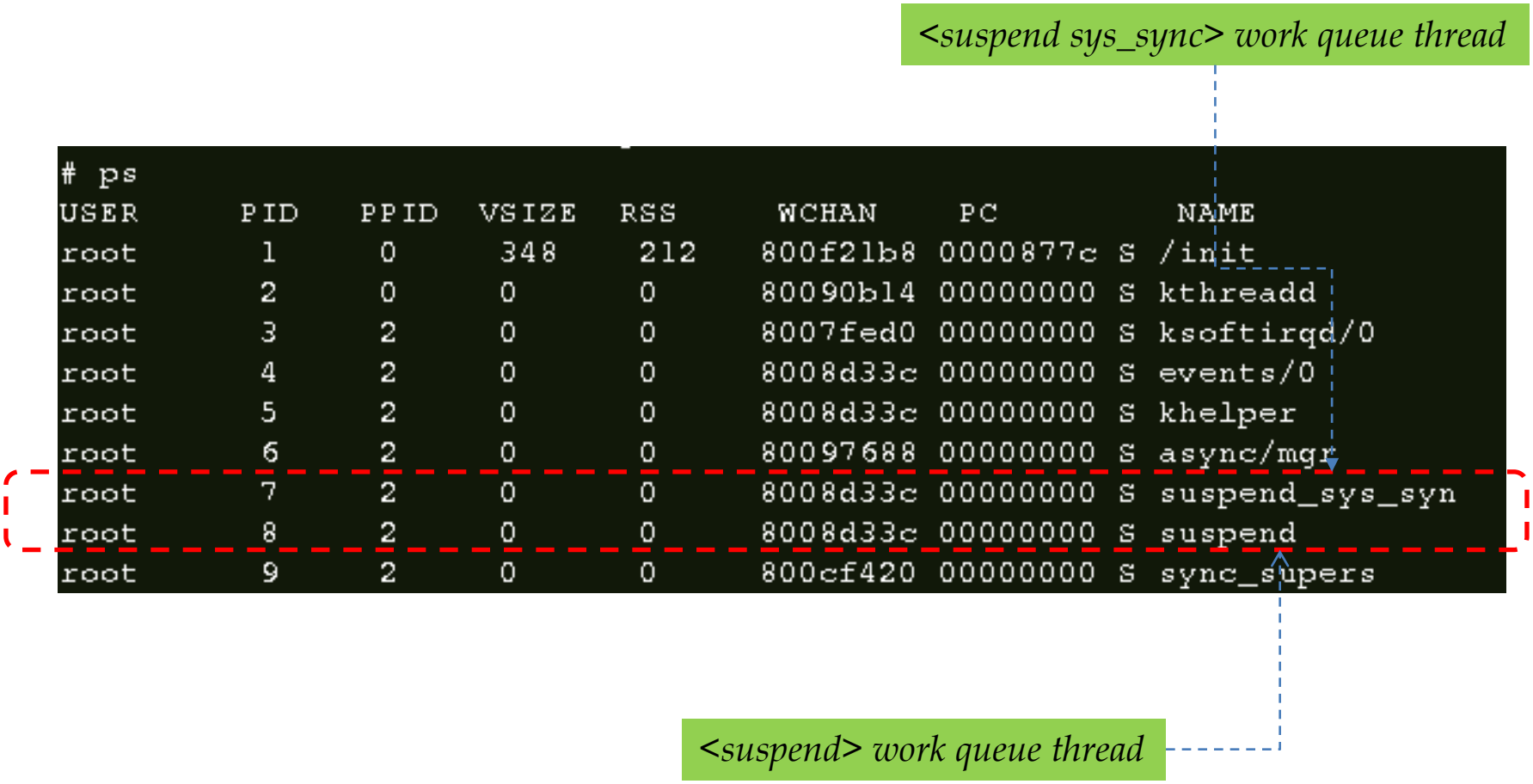
➔ *이후, early_suspend 시에 suspend되었던, 장치들이 다시 깨어나게 된다.*

2. Android Power Management – *suspend work queue/1*

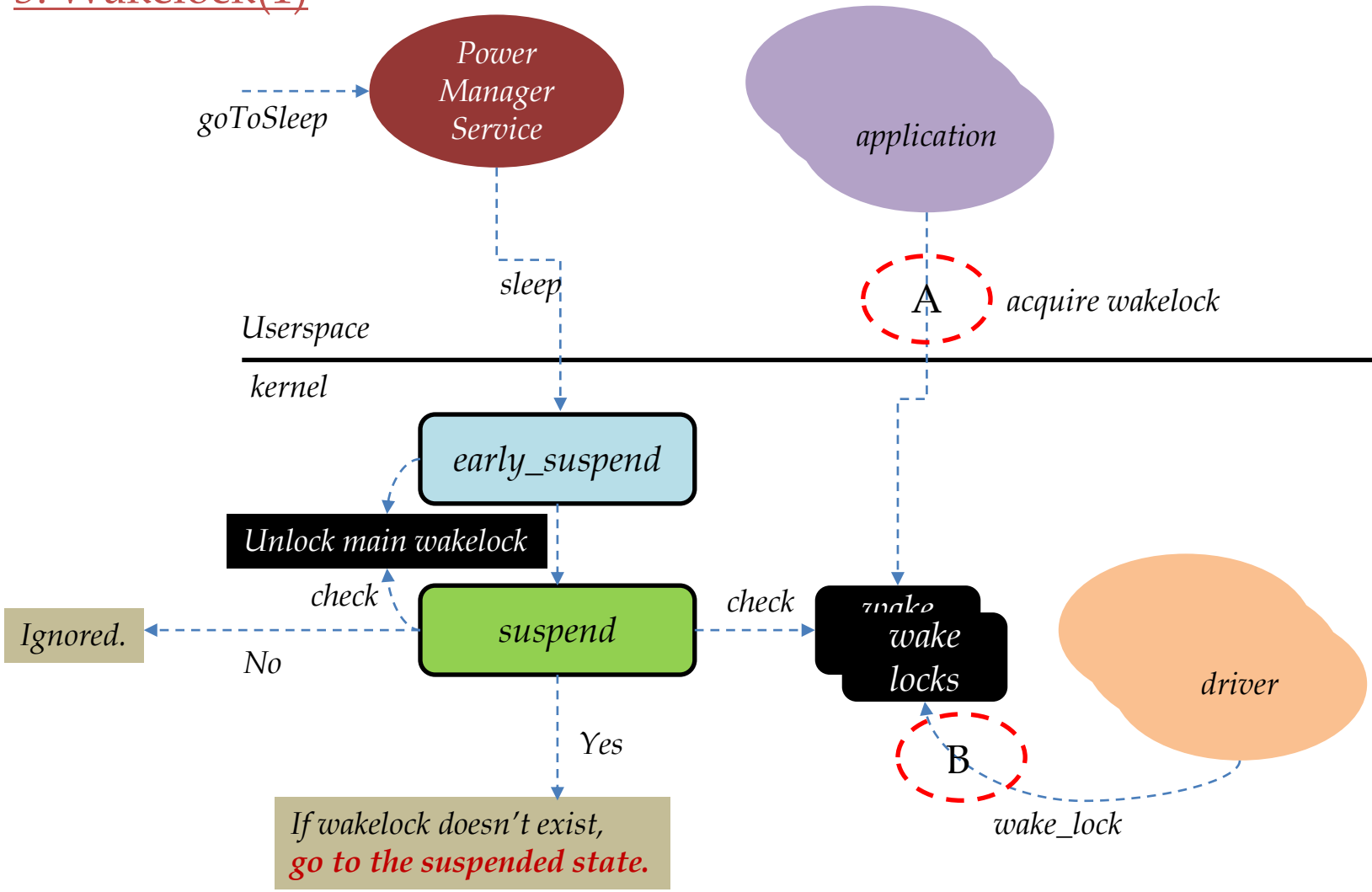


*(*) 앞서 절차 설명에서 언급한 바와 같이, early_suspend와 late_resume 처리를 위해 <suspend> work queue가 사용되고 있음^^*
(*) 또한, early_suspend 후, 실행되는 suspend 루틴도 work queue형태로 처리되고 있다.
(*) Android patch가 적용된 kernel의 경우는 suspend 진입 시점이 아래와 같이 존재할 수 있다.
a) timer로 주기적인 검사 도중, wake_lock이 하나도 존재하지 않음을 발견한 시점
b) wake_lock(), wake_unlock() 함수에서 검사했을 때, wake_lock이 존재하지 않는 시점
(*) 보다 자세한 사항은 kernel/power/earlysuspend.c, wakelock.c 파일 참조 !!!

2. Android Power Management – ***suspend work queue/2***



3. Wakelock(1)



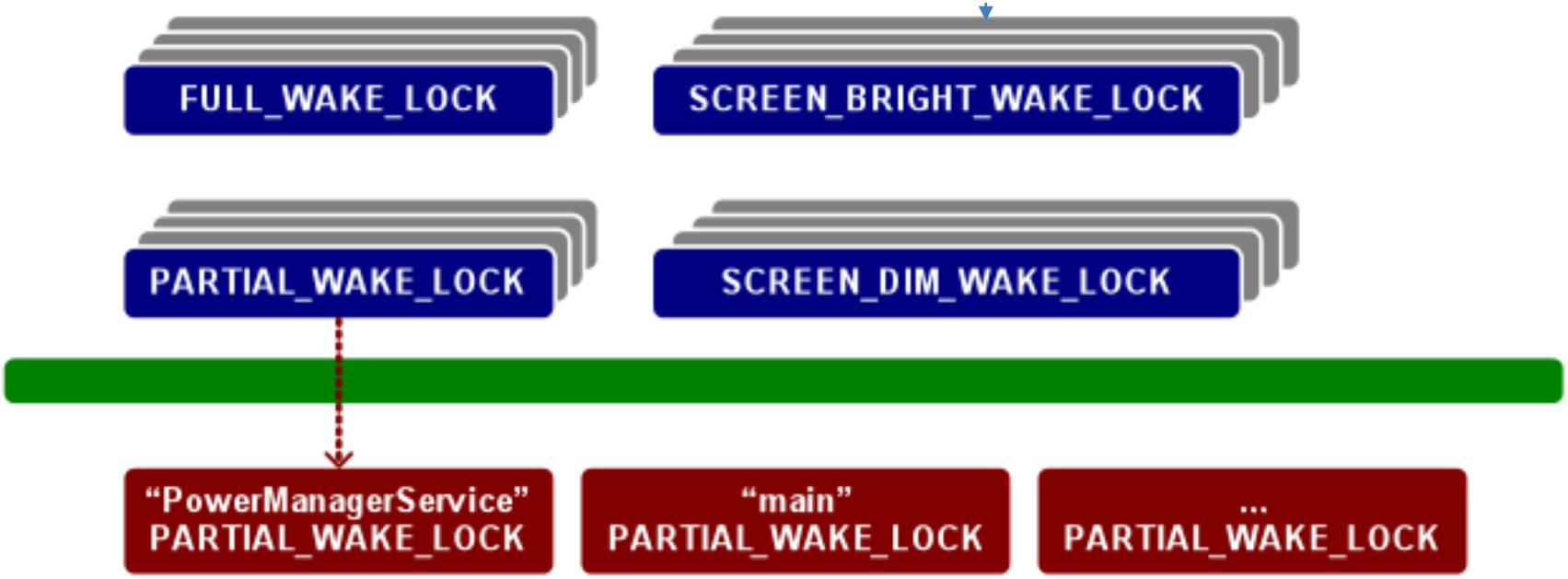
(*) android appl 및 linux device driver는 각각 독립적으로 wakelock을 설정할 수 있다.
(*) PowerManager로 부터 내려온 suspend 명령은 wakelock이 존재하는 상태에서는 무시되며, wakelock이 없을 경우, system이 suspend 상태로 들어가게 된다^^.

3. Wakelock(2) – *application wakelock*

(*) android appl에서 wakelock을 사용하는 예

```
PowerManager pm = (PowerManager)getSystemService(Context.POWER_SERVICE);
PowerManager.WakeLock wl =
        pm.newWakeLock(PowerManager.SCREEN_DIM_WAKE_LOCK, "My Tag");

wl.acquire();
/* screeen will stay on during this section ... */
wl.release();
```

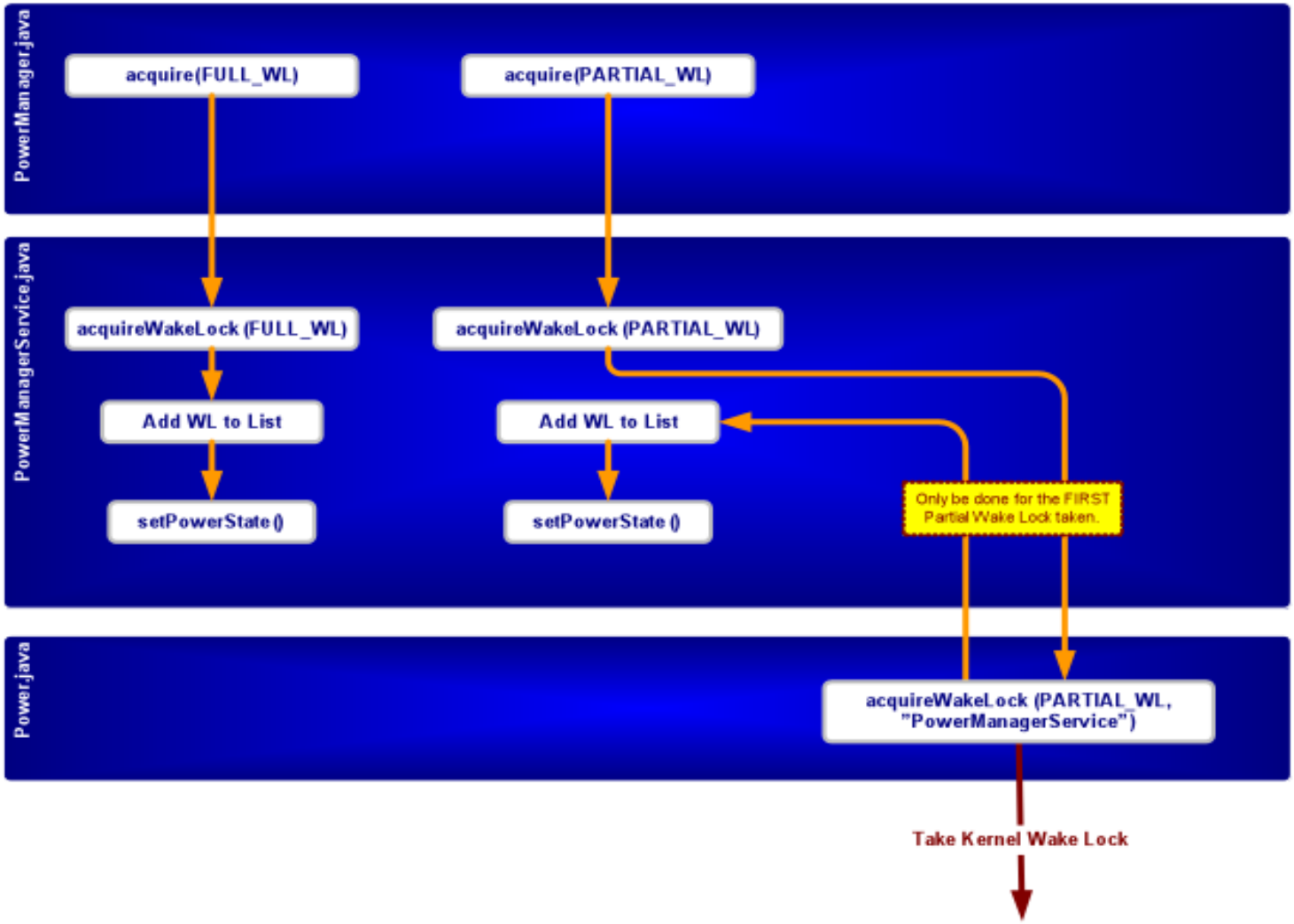


3. Wakelock(2) – *application wakelock: 4가지 Wakelock flag의 의미*

Wakelock flag	*의미(application code에서 사용함)*
1) PARTIAL_WAKE_LOCK	*CPU는 동작하고 있으나, screen은 on이 아닐 수 있는 상태.*
2) SCREEN_DIM_WAKE_LOCK	*Screen은 on 상태이나, keyboard backlight는 꺼질 수 있는 상태*
3) SCREEN_BRIGHT_WAKE_LOCK	*Screen은 가장 밝은 상태임. Keyboard backlight는 꺼질 수 있음.*
4) FULL_WAKE_LOCK	*Screen과 keyoard가 가장 밝은 상태에 있음.*

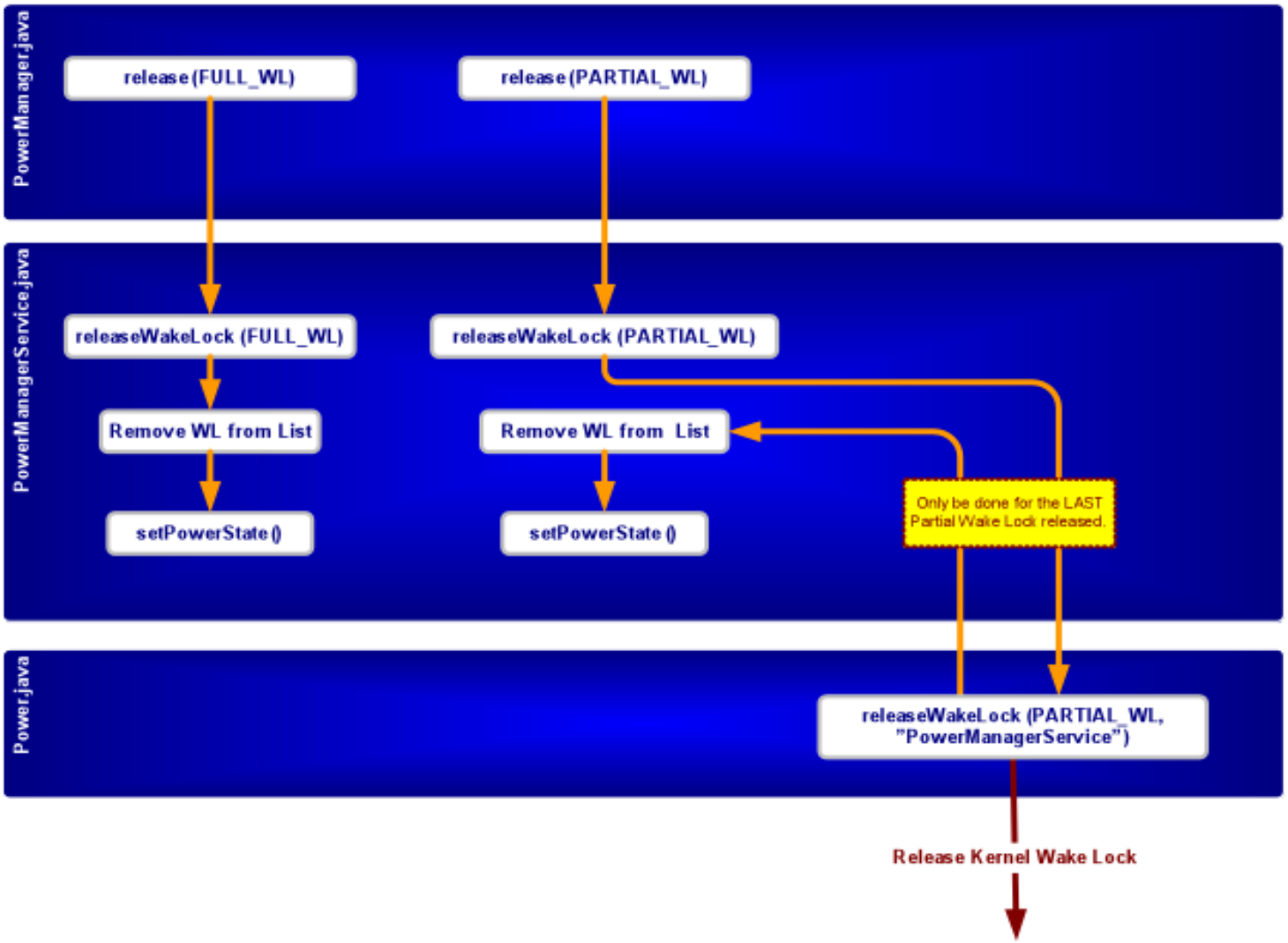
3. Wakelock(3) – *Acquiring wakelock by application*

(*) android appl에서 wakelock을 사용하는 예
➔ acquire 과정



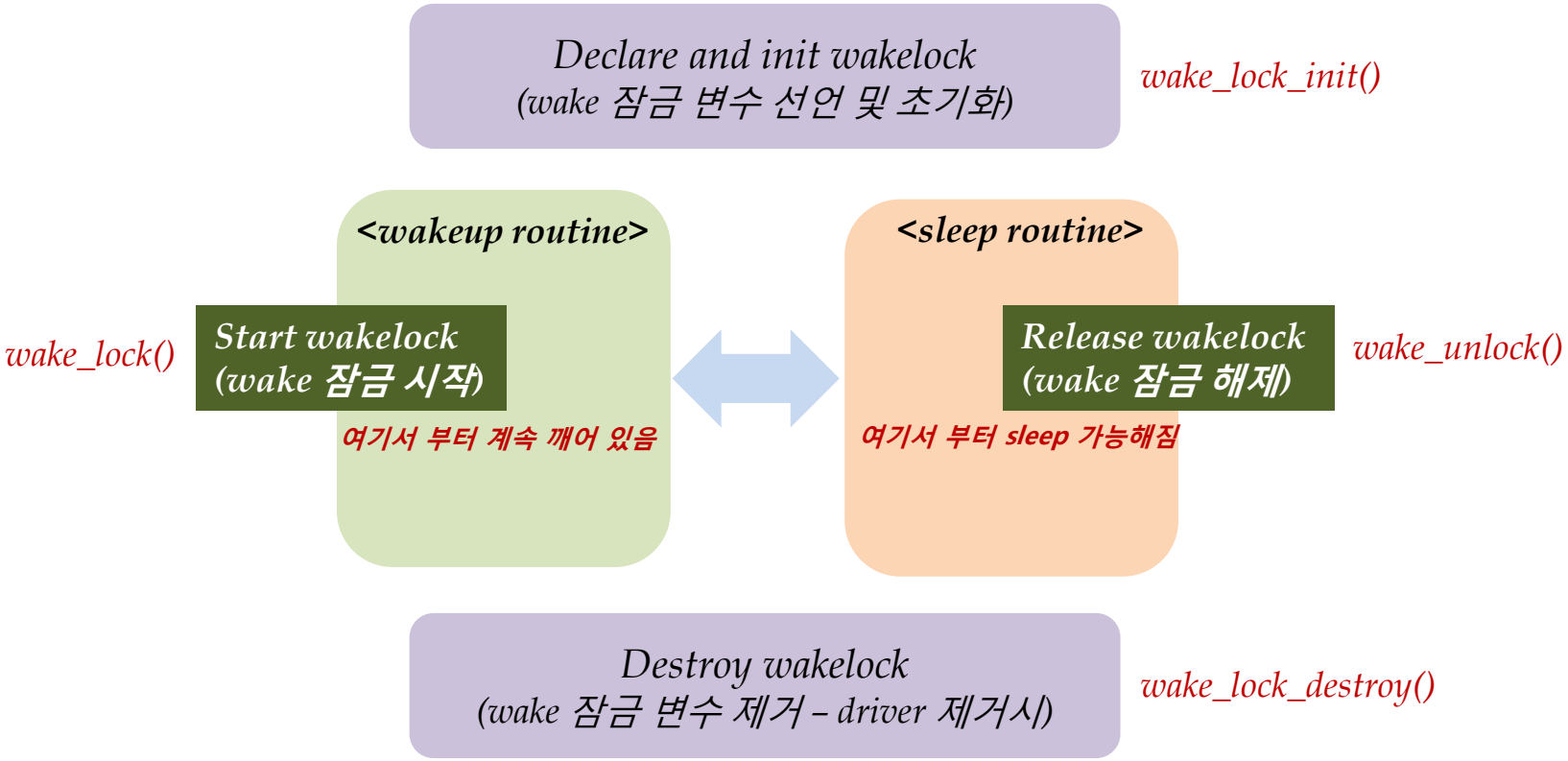
3. Wakelock(4) – *Releasing wakelock by application*

(*) android appl에서 wakelock을 사용하는 예
➔ release 과정



3. Wakelock(5) – ***Kernel wakelock***

*(*) **wakelock**: android 전원 관리 시스템의 핵심을 이루는 기능으로, 시스템이 suspend 혹은 low power state로 가는 것을 막아주는 메카니즘(google에서 만듦)이다.*
(*) Smart Phone은 전류를 많이 소모하므로, 항시 sleep mode로 빠질 준비를 해야 한다.
(*) wake_lock_init의 인자로 넘겨준, name 값은 /proc/wakelocks에서 확인 가능함.



3. Wakelock(6) – *Kernel wakelock*

<Kernel Wakelock 관련 API 모음>

[변수 선언] *struct wakelock mywakelock;*

[초기화] wake_lock_init*(&mywakelock, int type, "wakelock_name");*

➔ *type :*

- ***WAKE_LOCK_SUSPEND****: 시스템이 suspend 상태(full system suspend)로 가는 것을 막음*
- ***WAKE_LOCK_IDLE****: 시스템이 low-power idle 상태로 가는 것을 막음.*

[To hold(wake 상태로 유지)] wake_lock*(&mywakelock);*

[To release(sleep 상태로 이동)] wake_unlock*(&mywakelock);*

[To release(일정 시간 후, sleep 상태로 이동)] wake_lock_timeout*(&mywakelock, HZ);*

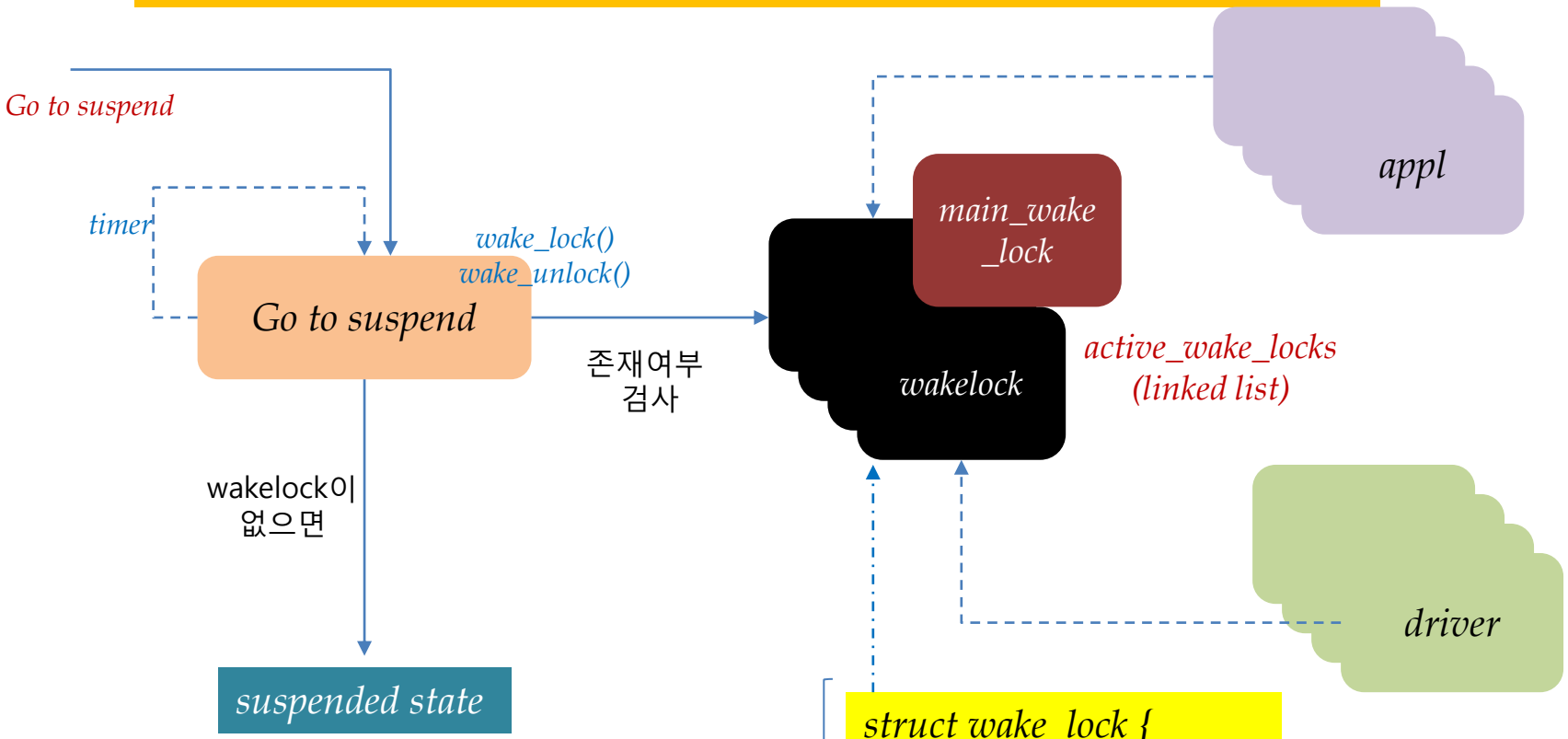
[제거] wake_lock_destroy *(&mywakelock);*

[기타] long has_wake_lock(int type);

➔ *한 개 이상의 wake lock이 사용 중이면 non-zero 값 return 함.*

3. Wakelock(7) – *Kernel wakelock internal/1*

(*) wakelock은 특별한 사항은 없고, 아래에서 보는 것 처럼, active_wake_locks list에 목록이 존재하지 않을 경우, suspend로 진입이 가능한 형태로 구성되어 있다.
(*) 뿐만 아니라, wake_lock 자체도 flags 및 expires 필드만을 가지고 운영되는 매우 간단한 구조체임을 알 수 있다(kernel/power/wakelock.c 파일 참조).




```
struct wake_lock {
  struct list_head link;
  int              flags;
  const char       *name;
  unsigned long    expires;
};
```

3. Wakelock(7) – *Kernel wakelock internal/2*

wakelocks_init ( ) 함수 분석[참고 사항]

(kernel/power/wakelock.c)

1) main_wake_lock, unknown_wakeup 등 두 개의 wakelock 초기화
2) 자신을 platform_device(power_device), platform_driver(power_driver) 형태로 등록

```
struct platform_driver power_driver = {
    .driver.name = "power",
    .driver.pm = &power_driver_pm_ops,
};
struct platform_device power_device = {
    .name = "power",
};
```

3) suspend_sys_sync 및 suspend work queue 생성
   ➔ *각각 sys_sync 및 early_suspend/suspend/late_resume 용 work queue*
   ➔ *sys_sync는 process freeze 시, 조건을 check하기 위해 사용*
4) suspend_sys_sync_comp completion 생성
   ➔ *kernel/power/process.c 파일의 freeze_processes( ) 함수 내에서 suspend_sys_sync_wait( ) 함수 호출시, wait_for_completion이 호출되고, suspend_sys_sync_timer handler내에서 complete를 호출해주게 됨.*
5) /proc/wakelocks 생성

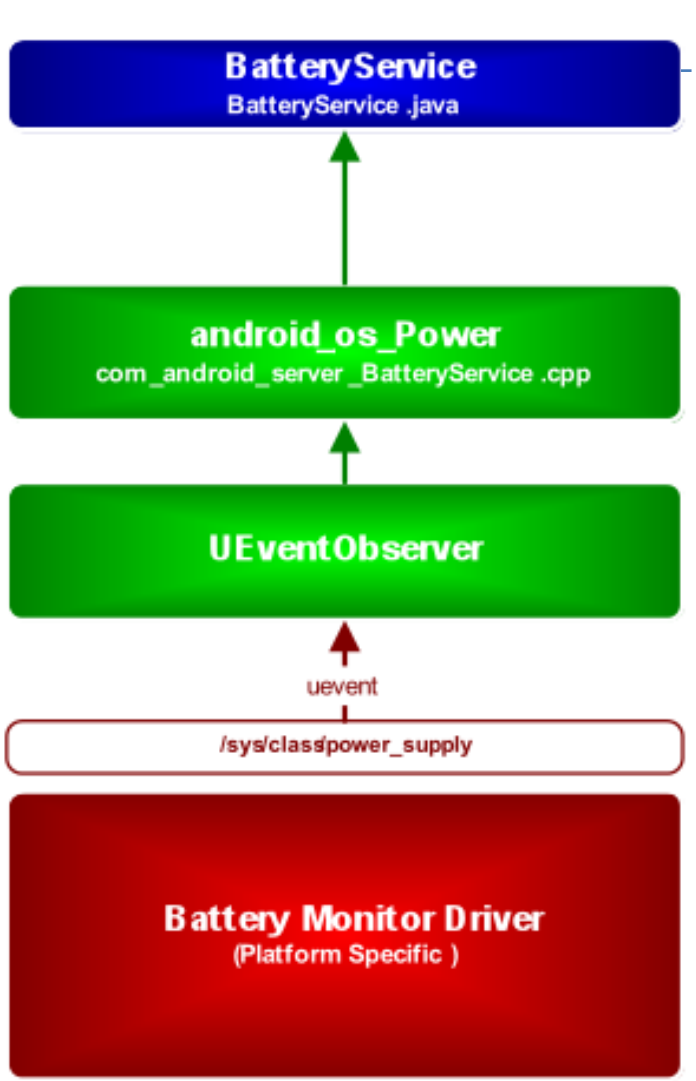
4. Linux Runtime PM

(*) I/O 장치들에 대해, run-time에 low-power state로 만들거나, wake-up 시키는 것을 run-time power management라고 함.
(*) PM core에서 아래의 callback 함수를 정의(등록)한 드라이버에 대해 작업을 수행한다.
(*) include/linux/pm.h 파일 참조

```
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 (*runtime_suspend)(struct device *dev);
   int (*runtime_resume)(struct device *dev);
   int (*runtime_idle)(struct device *dev);
};
```

(*) Runtime PM을 사용하기 위해서는 CONFIG_PM_RUNTIME=y 이 켜져 있어야 함.
(*) 보다 자세한 사항은 Documentation/power/runtime_pm.tx 파일 참조 !

5. Battery Service(1) - *Overview*



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

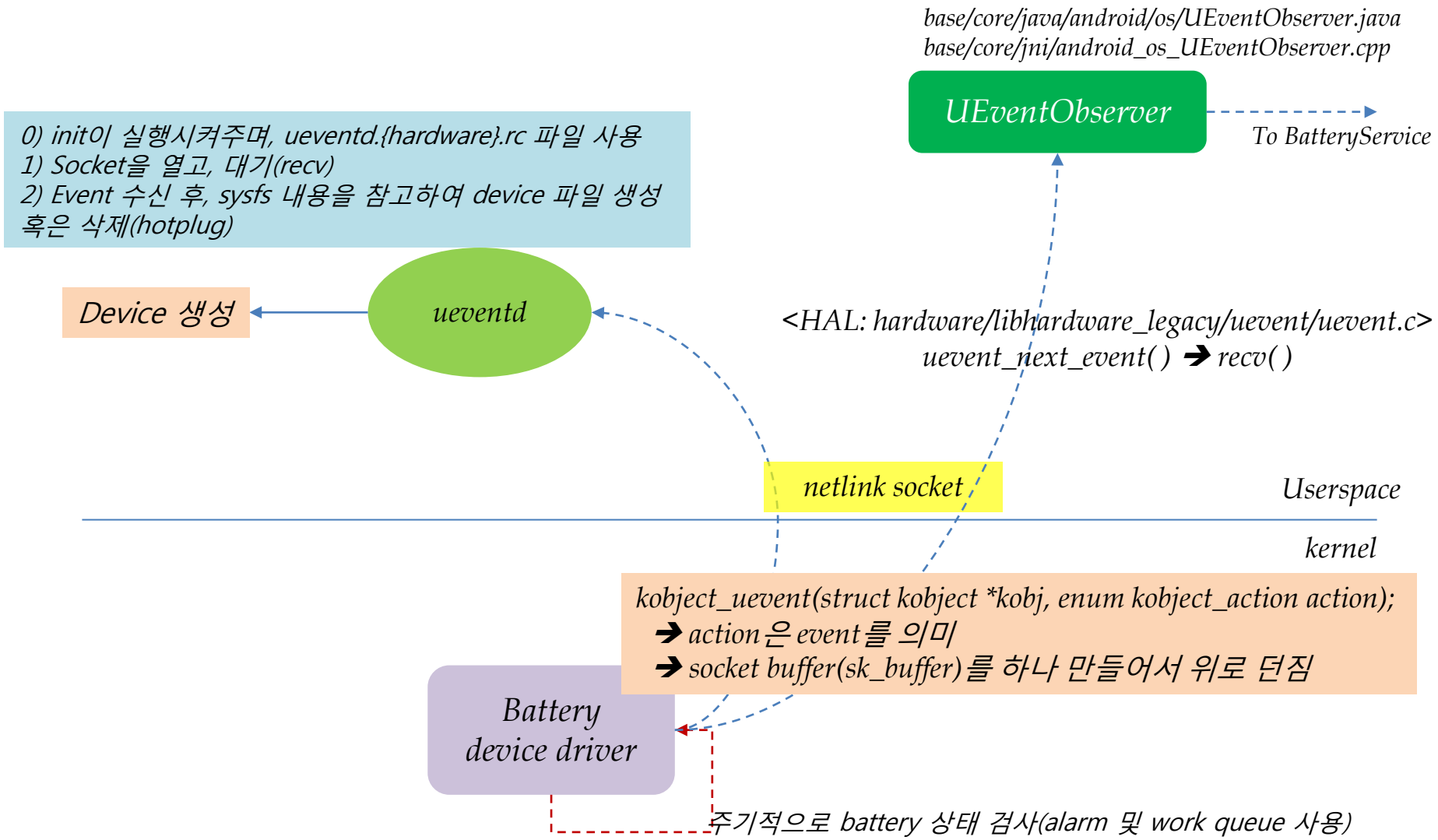
2) Battery 상태를 갱신하기 위하여 uevent 사용함.
➔ SUBSYSTEM=power_supply

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

5. Battery Service(2) – *uevent 전달 과정*

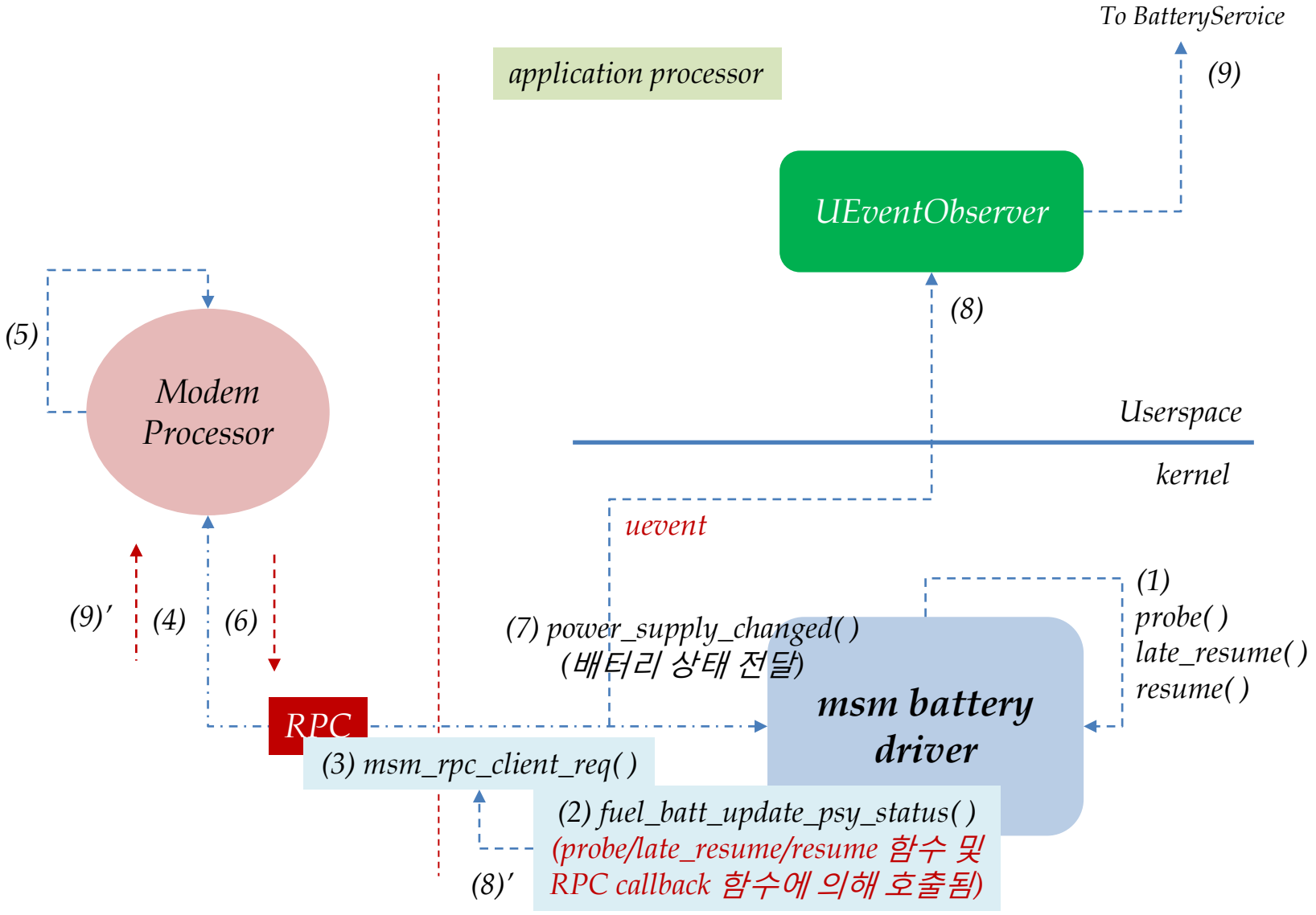


5. Battery Service(3) – *battery driver 예(1)*

<ds2784 battery driver probe 함수 소개>
(drivers/power/ds2784_battery.c)

(1) ds2784 battery driver는 platform driver 임.
(2) ds2784_device_info용 buffer를 하나 할당함.
(3) platform_set_drvdata( ) 함수를 호출하여, ds2784_device_info를 pdev (platform_device 포인터)에 저장함.
(4) ds2784_device_info data structure의 각 필드를 채움.
(5) power_supply_register 함수를 호출하여, power supply 드라이버로 등록함 (/sys/class/power_supply).
➔ *drivers/power/power_supply*.c 파일 참조*
(6) Battery monitoring 용 work queue(ds2784_battery_work)를 생성
➔ *이 monitoring 함수 내에서 battery의 상태를 모니터링하여, uevent를 날리게 됨.*
➔ *이 때, power_supply_core.c 파일을 이용함.*
[참고] ds2784_battery_work( ) => ds2784_battery_update_status( ) => power_supply_changed( ) => power_supply_changed_work( ) =>kobject_uevent( ) …
(7) wake_lock을 생성 및 초기화함(work_wake_lock).
(8) alarm를 초기화(ds2784_battery_alarm) 함.
➔ *이 alarm에서 주기적으로 battery monitoring work 을 호출해주게 됨.*
(9) (6)에서 생성한 work queue에 work을 하나 던짐(시작) …

5. Battery Service(3) – *battery driver 예(2)*



References

- *1) %233.GTUG-Android-Power Management.pdf … [Renaldo Noma 2010]*
- *2) Suspend-to-RAM in Linux …. [Proceedings of the Linux Symposium July 23rd-26th, 2008]*
- *3) Power Management & Battery Life … [The 7th Korea Android Conference, Kwangwoo LEE, kwangwoo.lee@gmail.com]*
- *4) Some Internet Articles …*

Thanks a lot !

SlowBoot