Android Power Management

Jerrin Shaji George

Agenda

- Concept
- Linux Power Management
- Android Power Management Design
- Wake Locks
- System Sleep (Suspend)
- Battery Service

Concept

- Designed for mobile devices
- Goal is to prolong battery life
- Build on top of Linux Power Management
  - Not directly suitable for a mobile device
- Designed for devices which have a 'default-off' behaviour
  - The phone is not supposed to be on when we do not want to use it
  - Powered on only when requested to be run, off by default
  - Unlike PC, which has a default on behaviour

Linux Power Management

Two popular power management standards

1. APM (Advanced Power Management)
2. ACPI (Advanced Configuration and Power Interface)

Linux Power Management

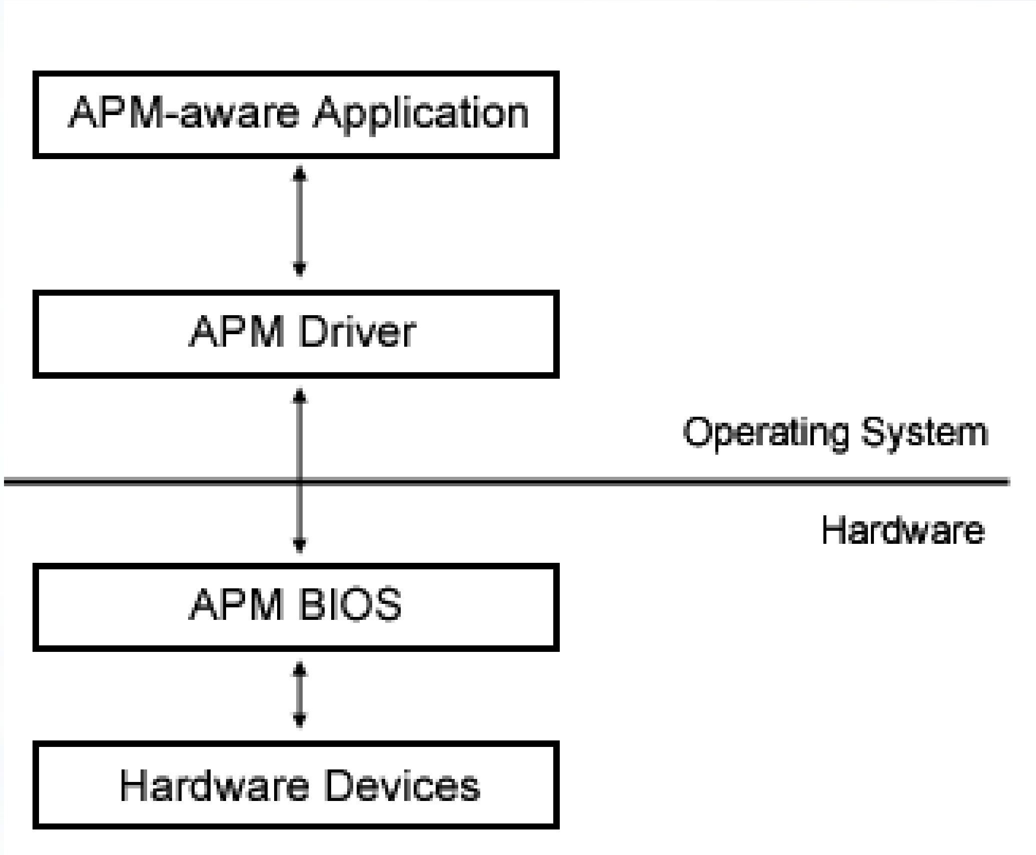
APM

- Control resides in BIOS
- Uses activity timeouts to determine when to power down a device
- BIOS rarely used in embedded systems
- Makes power-management decisions without informing OS or individual applications
- No knowledge of add-in cards or new devices

Linux Power Management

APM

- Uses layered approach to manage devices
- APM-aware applications (including device drivers) talk to an OS-specific APM driver
- The driver communicates to the APM-aware BIOS, which controls the hardware

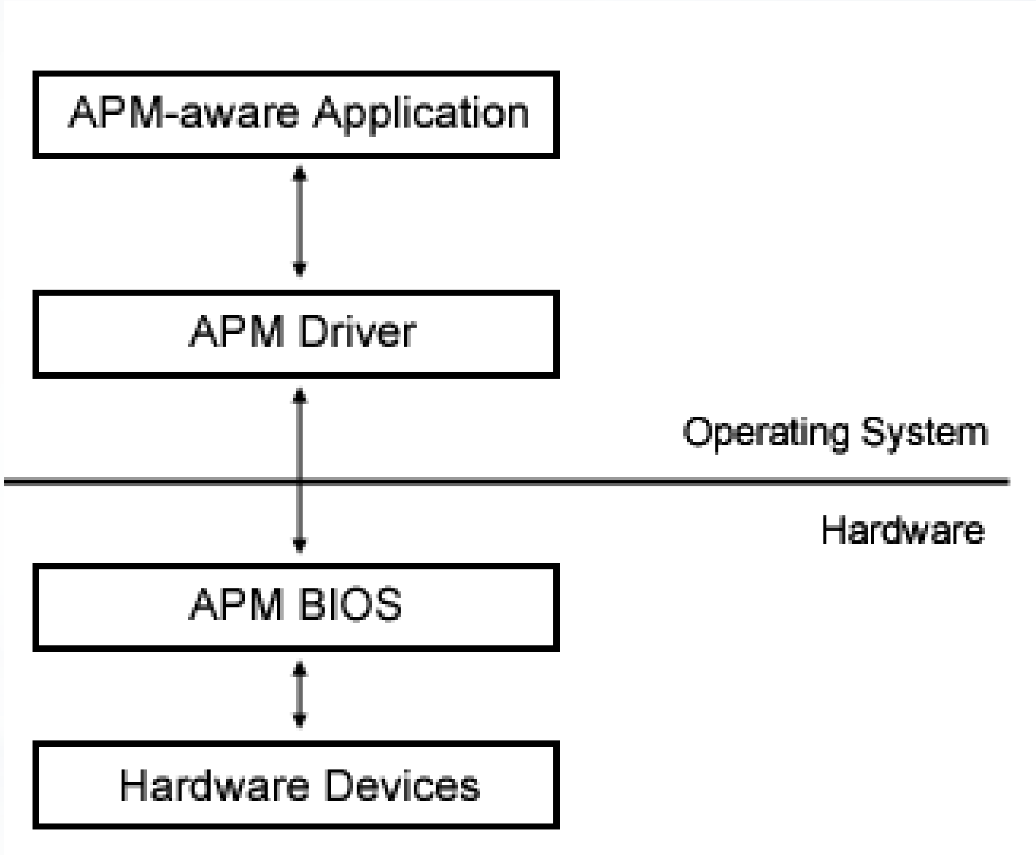


Linux Power Management

APM

- Communication occurs in both directions; power management events are sent from the BIOS to the APM driver, and the APM driver sends information and requests to the BIOS via function calls

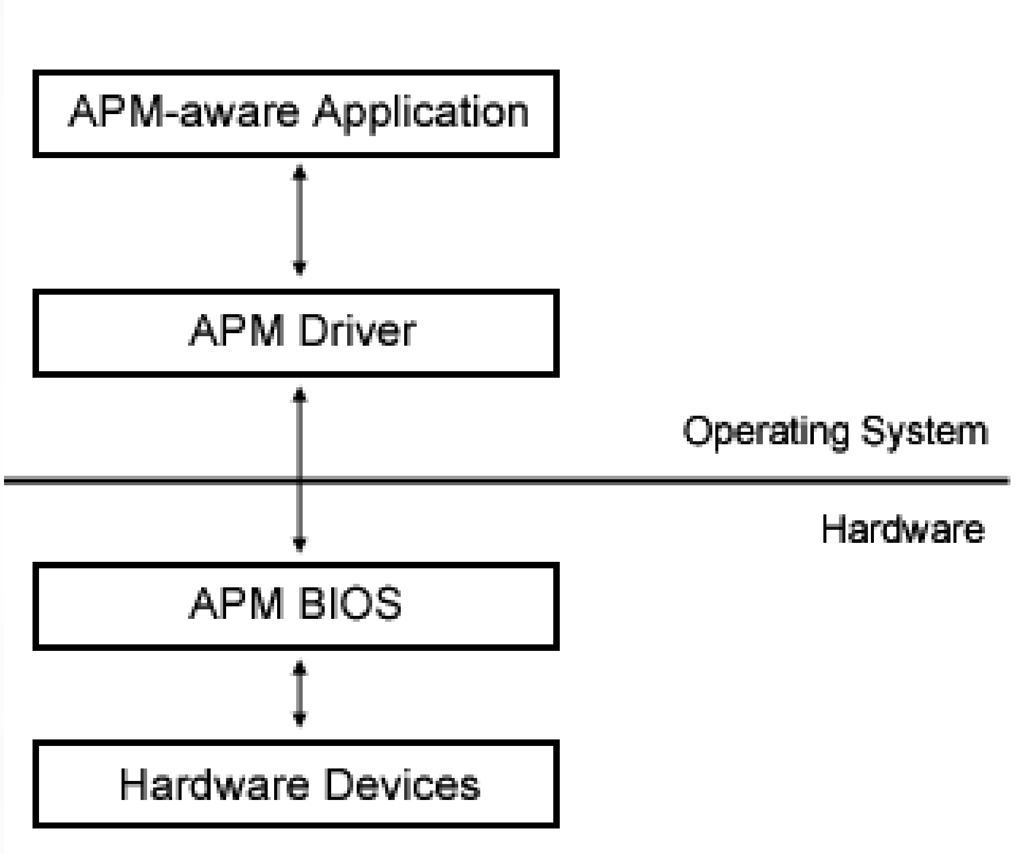
- In this way the APM driver is an intermediary between the BIOS and the operating system



Linux Power Management

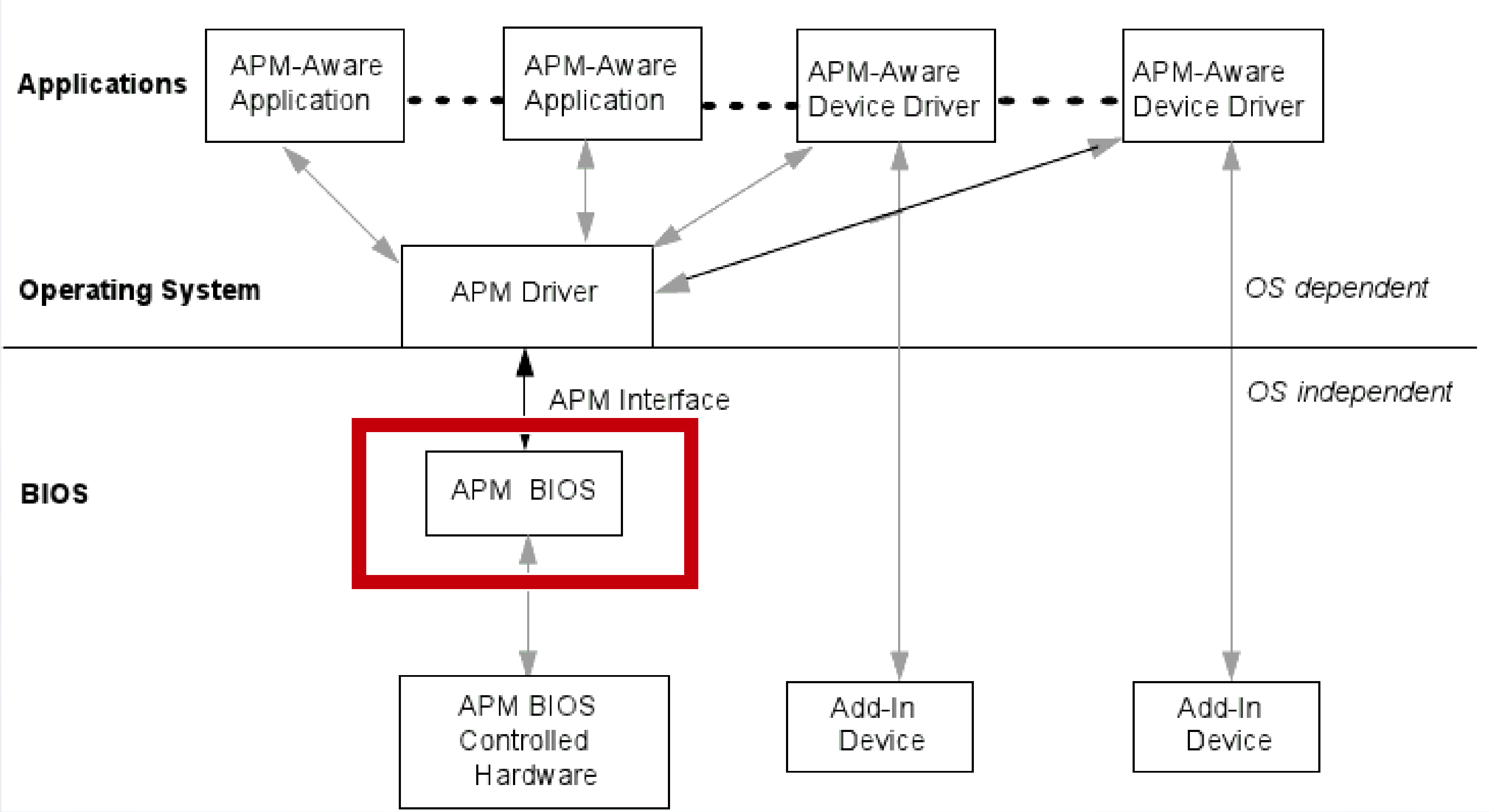
APM

- Power management happens in two ways; through function calls from the APM driver to the BIOS requesting power state changes, and automatically based on device activity



Linux Power Management

APM



Linux Power Management

ACPI

- Control divided between BIOS and OS
- Decisions managed through the OS
- Enables sophisticated power policies for general-purpose computers with standard usage patterns and hardware
- No knowledge of device-specific scenarios (e.g. need to provide predictable response times or to respond to critical events over extended period)

Linux Power Management

ACPI

ACPI specification defines the following four Global ‘Gx’ states and six Sleep ‘Sx’ states for an ACPI-compliant computer-system:

- G0 (S0)
  - Working
  - ‘Awaymode’ is a subset of S0, where monitor is off but background tasks are running

Linux Power Management

ACPI

- G1, Sleeping, subdivides into the four states S1 through S4:
  - S1 : All processor caches are flushed, and the CPU(s) stop executing instructions. Power to the CPU(s) and RAM is maintained; devices that do not indicate they must remain on may be powered down
  - S2: CPU powered off. Dirty cache is flushed to RAM
  - S3(mem): Commonly referred to as Standby, Sleep, or Suspend to RAM. RAM remains powered
  - S4: Hibernation/Suspend-to-Disk - All content of main memory is saved to non-volatile memory such as a hard drive, and is powered down

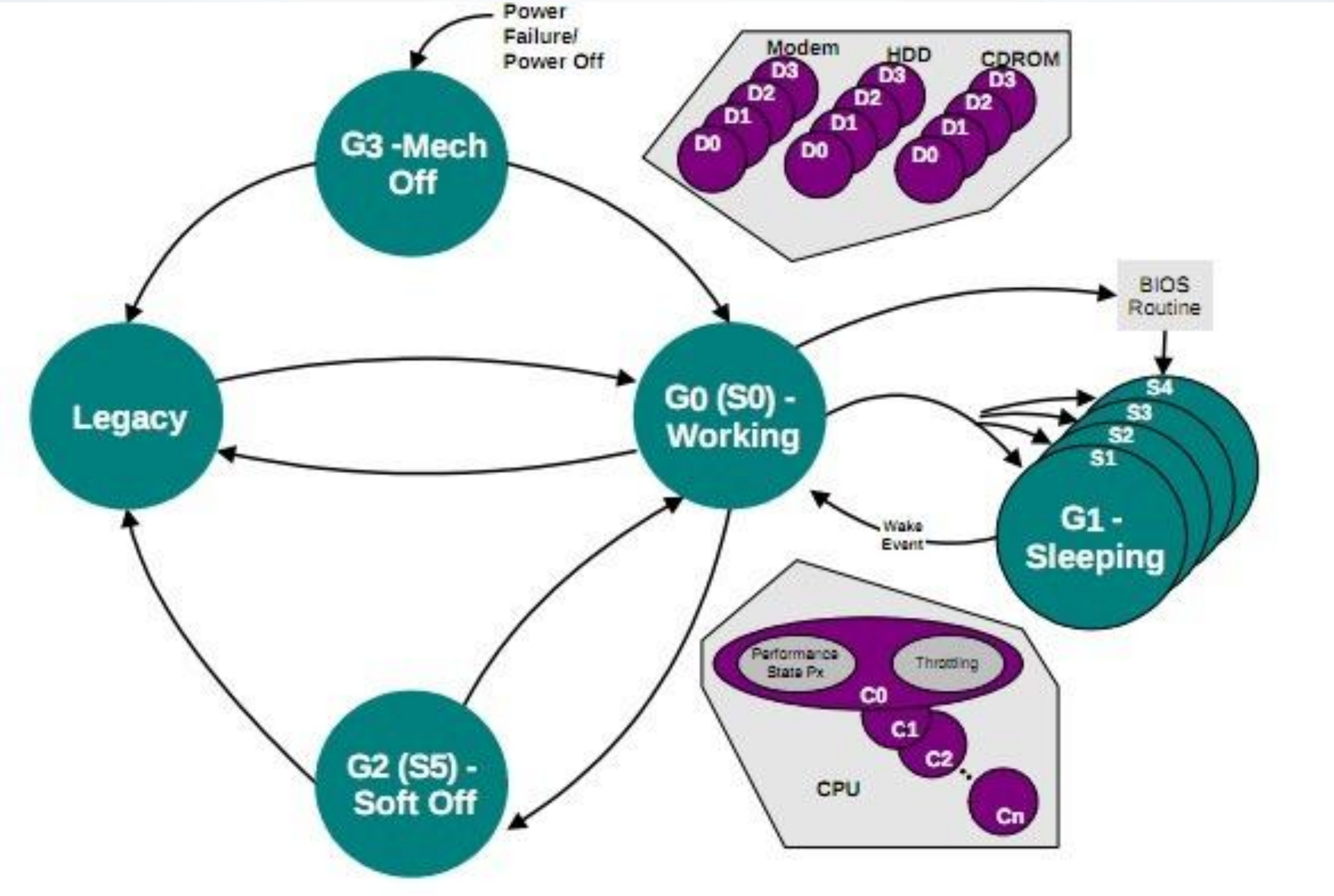
Linux Power Management

ACPI

- G2 (S5), Soft Off
- G3, Mechanical Off
  - The computer's power has been totally removed via a mechanical switch
- Legacy State : The state on an operating system which does not support ACPI. In this state, the hardware and power are not managed via ACPI, effectively disabling ACPI.

Linux Power Management

ACPI



Linux Power Management

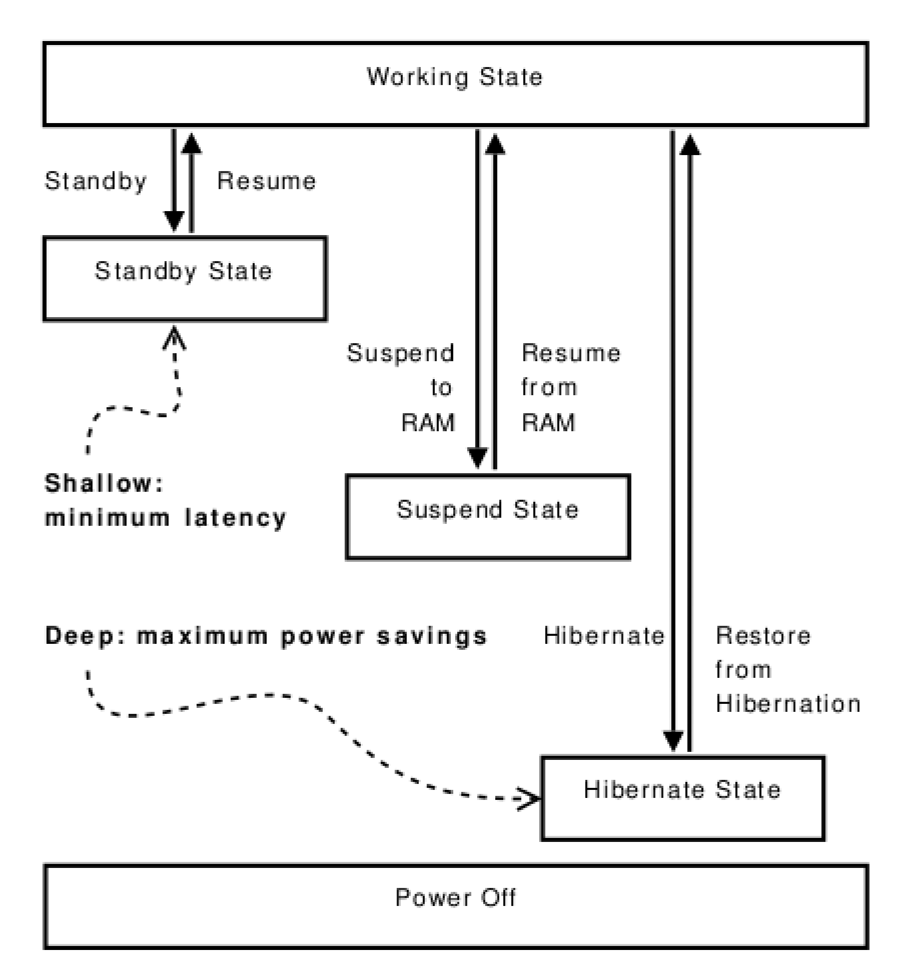
- Power mode interface is on sysfs
  - /sys/power/state

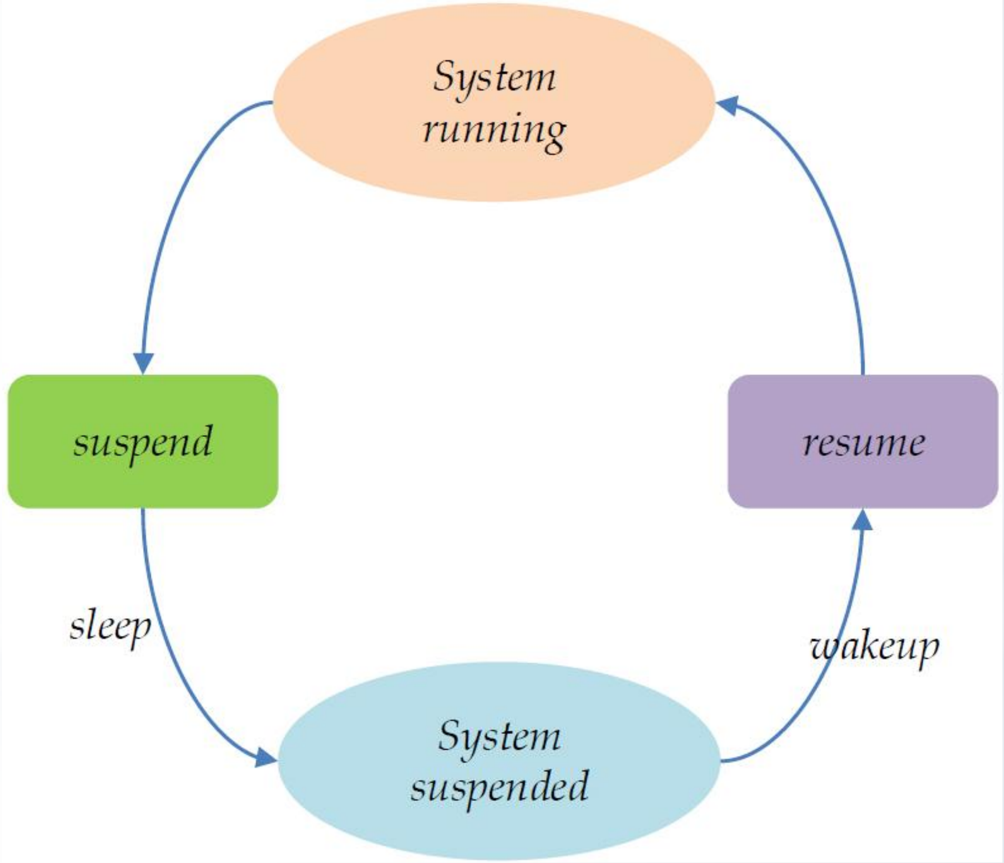
- sysfs is a virtual file system provided by Linux. sysfs exports information about devices and drivers from the kernel device model to user space, and is also used for configuration

- Changing state done by
  - # echo mem > /sys/power/state
  - # echo disk > /sys/power/state
  - # echo standby > /sys/power/state

Linux Power Management

Overview





Android PM Design

- Built as a wrapper to Linux Power Management
- In the Kernel
  - Added 'on' state in the power state
  - Added Early Suspend framework
  - Added Partial Wake Lock mechanism
- Apps and services must request CPU resource with ‘wake locks’ through the Android application framework and native Linux libraries in order to keep power on, otherwise Android will shut down the CPU
- Android PM uses wake locks and time out mechanism to switch state of system power, so that system power consumption decreases

Wake Locks

- By default, Android tries to put the system into a sleep or better a suspend mode as soon as possible
- Applications running in the Dalvik VM can prevent the system from entering a sleep or suspend state, i.e. applications can assure that the screen stays on or the CPU stays awake to react quickly to interrupts
- The means Android provides for this task is wake locks
- If there are no active wake locks, CPU will be turned off
- If there are partial wake locks, screen and keyboard will be turned off

Wake Locks

Types of Wake Locks

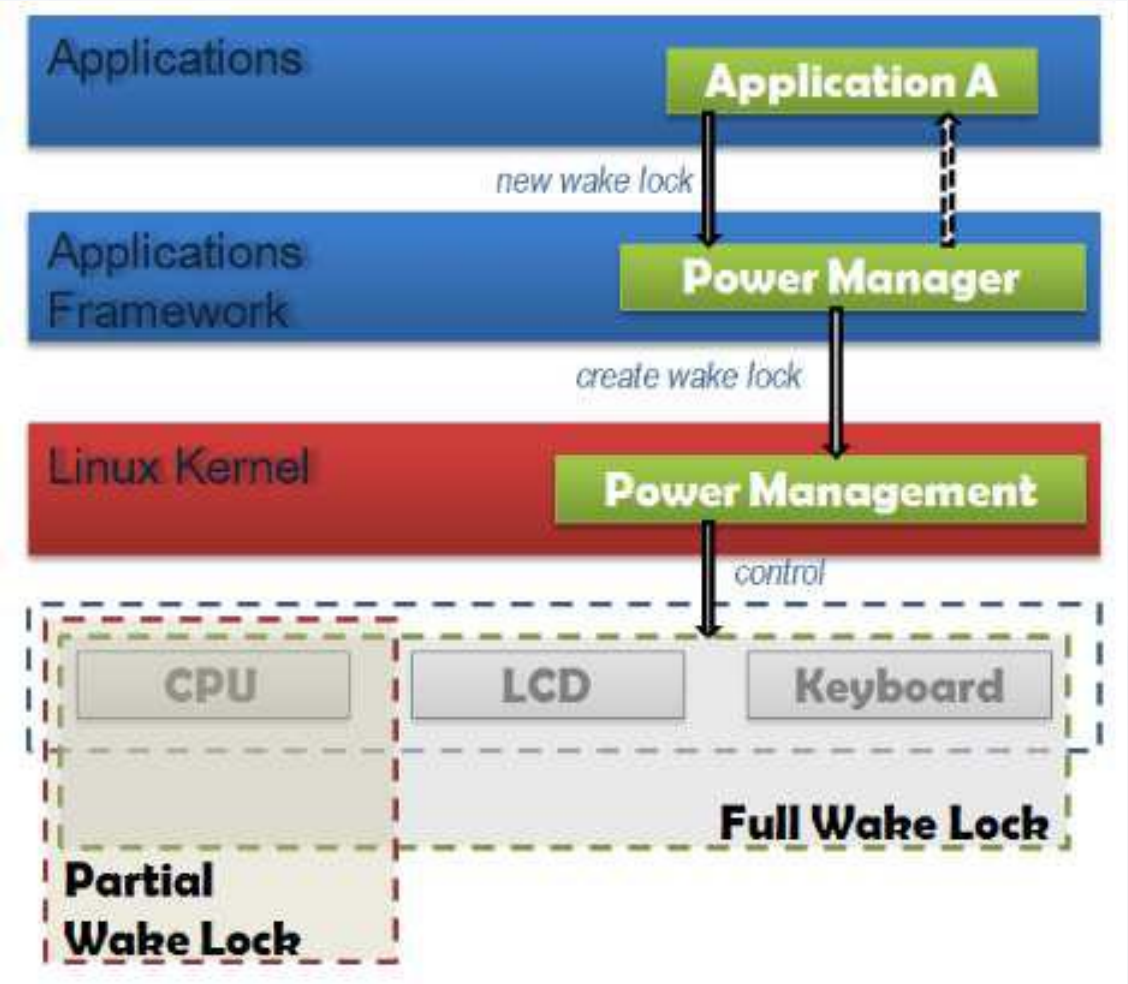
- PARTIAL_WAKE_LOCK
  - Ensures that the CPU is running
  - The screen might not be on
- SCREEN_DIM_WAKE_LOCK
  - Wake lock that ensures that the screen is on, but the keyboard backlight will be allowed to go off, and the screen backlight will be allowed to go dim
- SCREEN_BRIGHT_WAKE_LOCK
  - Wake lock that ensures that the screen is on at full brightness; the keyboard backlight will be allowed to go off
- FULL_WAKE_LOCK
  - Full device ON, including backlight and screen

Android PM Design

- Android implements an application framework on top of the kernel called Android Power Management Applications Framework
- The Android PM Framework is like a driver. It is written in Java which connects to Android power driver through JNI
- Currently Android only supports screen, keyboard, buttons backlight, and the brightness of screen

Android PM Design

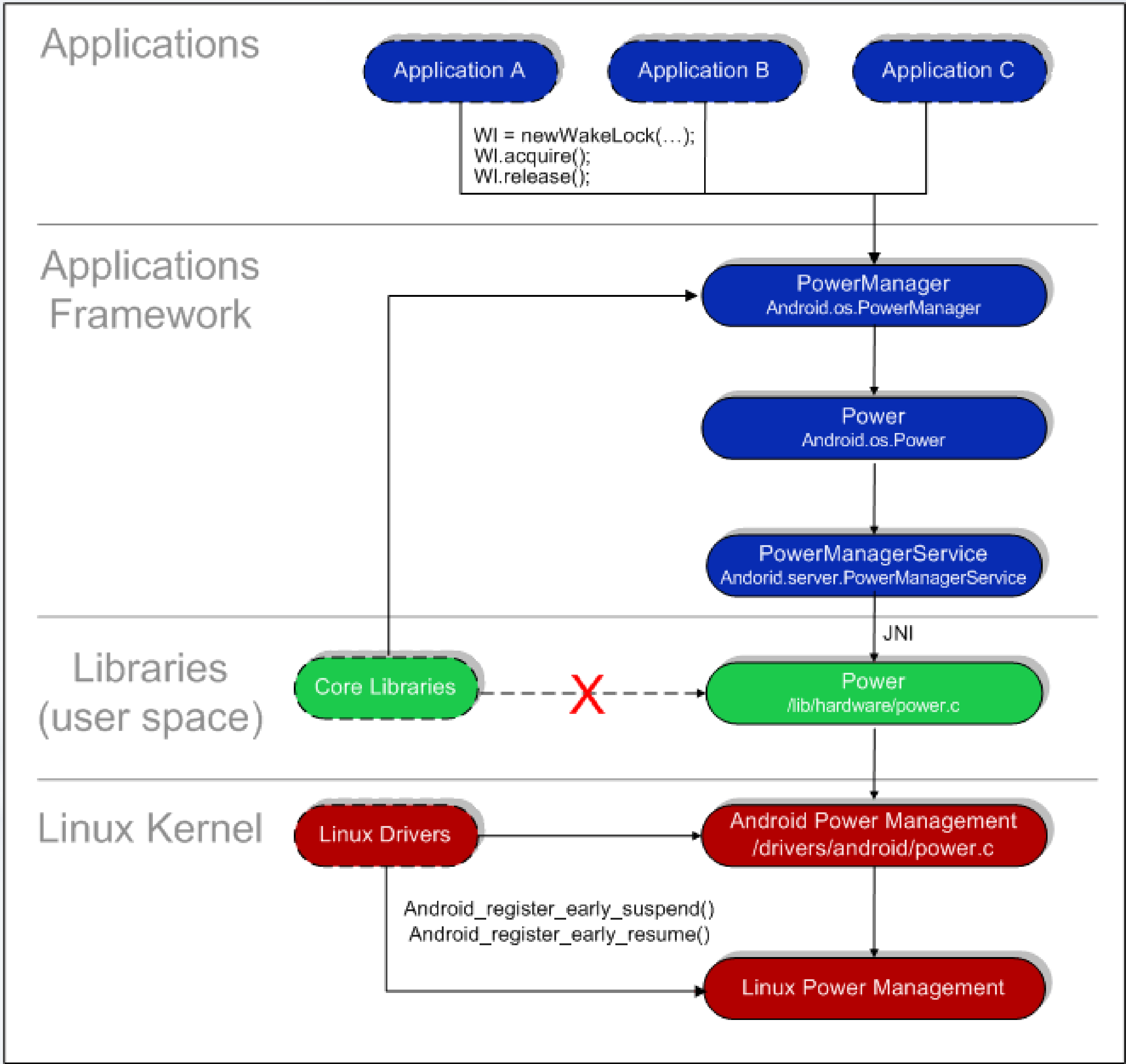
Through the framework, user space applications can use ‘PowerManger’ class to control the power state of the device



Green - Native

Blue - Java

Red - Kernel



Applications

Application A

Application B

Application C

```
Wl = newWakeLock(…);
Wl.acquire();
Wl.release();
```

Applications Framework

PowerManager
Android.os.PowerManager

Power
Android.os.Power

PowerManagerService
Andorid.server.PowerManagerService

JNI

Libraries (user space)

Core Libraries

X

Power
/lib/hardware/power.c

Linux Kernel

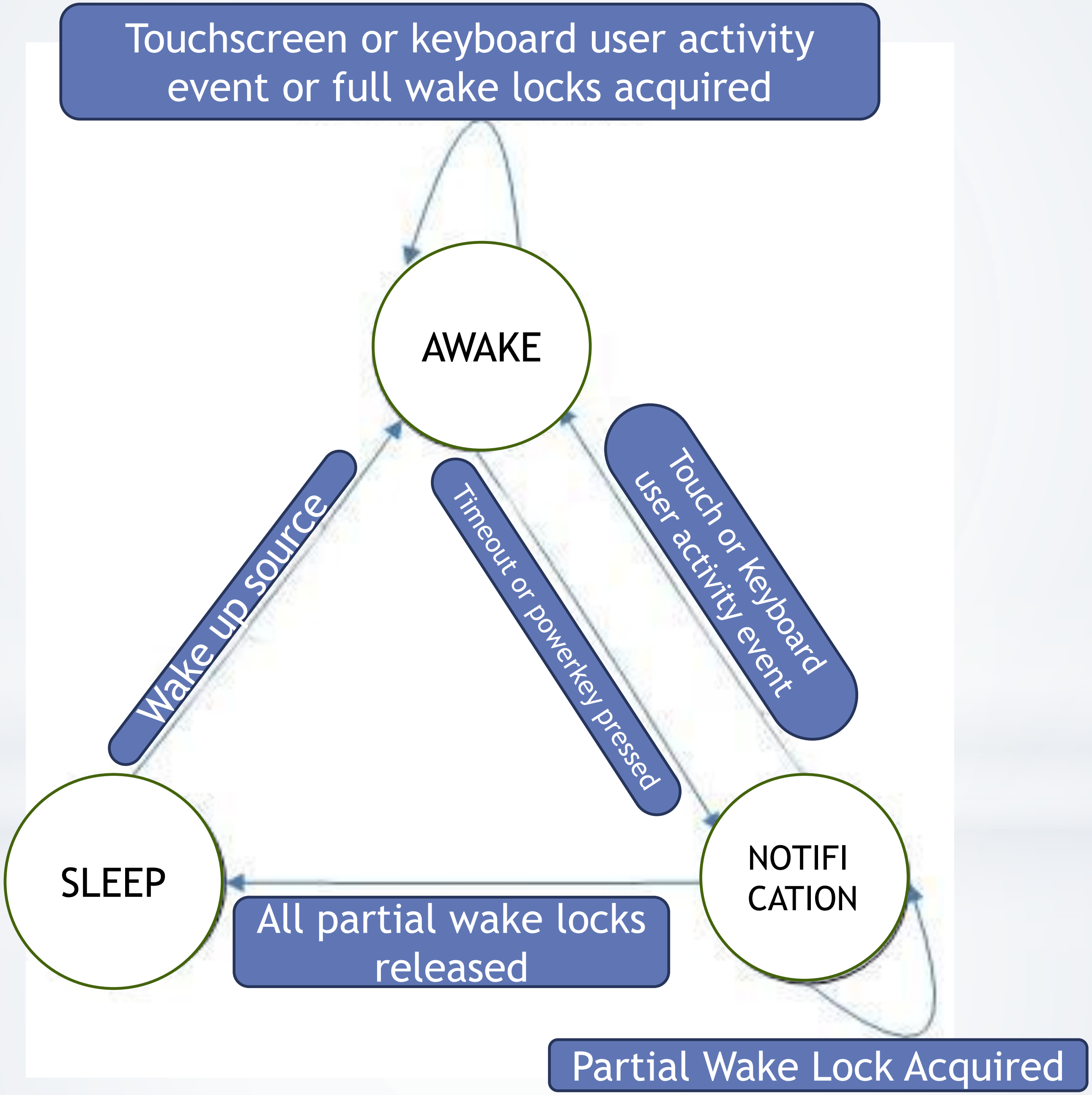
Linux Drivers

Android Power Management
/drivers/android/power.c

Android_register_early_suspend()
Android_register_early_resume()

Linux Power Management

A Finite State Machine of Android Power Management



Touchscreen or keyboard user activity event or full wake locks acquired

AWAKE
Wake up source
Timeout or powerkey pressed
Touch or Keyboard user activity event
SLEEP
NOTIFI CATION

All partial wake locks released

Partial Wake Lock Acquired

Android PM Design

- When a user application acquire full wake lock or screen/keyboard touch activity event occur, the machine will enter ‘AWAKE’ state
- If timeout happens or power key is pressed, the machine will enters ‘NOTIFICATION’ state
    - If partial wake locks are acquired, it will remain in ‘NOTIFICATION’
    - If all partial locks are released, the machine will go into ‘SLEEP’

Android PM Implementation

- Android PM Framework provides a service for user space applications through the class PowerManger to achieve power saving
- The flow of exploring Wake locks are :
  - Acquire handle to the PowerManager service by calling Context.getSystemService()
  - Create a wake lock and specify the power management flags for screen, timeout, etc.
  - Acquire wake lock
  - Perform operation such as play MP3
  - Release wake lock

Kernel Wake Lock

- Used to prevent system from entering suspend or low-power-state
- Partial Wake Lock behaviour
- Can be acquired/released from Native apps through Power.c interface
- Can be acquired/released internally from kernel

Wake Locks

How are Wake Locks Managed

- Wake Locks are mainly managed in Java layer
- When an android application takes a wake lock, a new instance of wake lock is registered in the PowerManagerService
  - PowerManagerService is running in the java layer
- Registered wake locks are put in a list

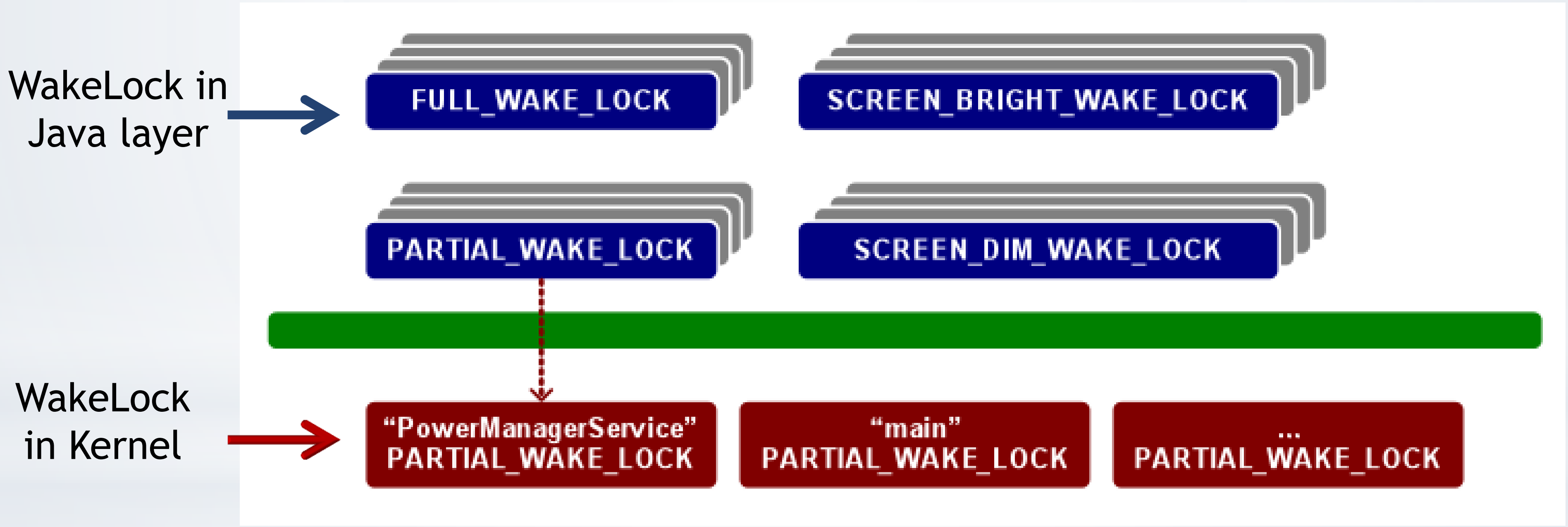
Wake Locks

How are Wake Locks Managed

- A Single Partial Wake Lock in Kernel is needed to protect multiple instance of Partial Wake Locks in Java
  - It is taken on behalf of PowerManagerService class with the name PowerManagerService
- Other wake lock residing in kernel side are either from Native code via Power.c API or taken internally in the Kernel
  - E.g. Partial wake lock for keyboard
- There is one main wake lock called ‘main’ in the kernel to keep the kernel awake
- It will be the last wake lock to be released when system goes to suspend

Wake Locks

How are Wake Locks Managed



Wake Locks

Working

- By default, a time out is set to off the screen
- If FULL_WAKE_LOCK or SCREEN_BRIGHT_WAKE_LOCK has been taken, when a request comes to the system to go to sleep, the system does not go to sleep
- If no locks are currently being taken, request is sent through JNI to suspend the device

Wake Locks

Special behaviour of Partial Wake Lock

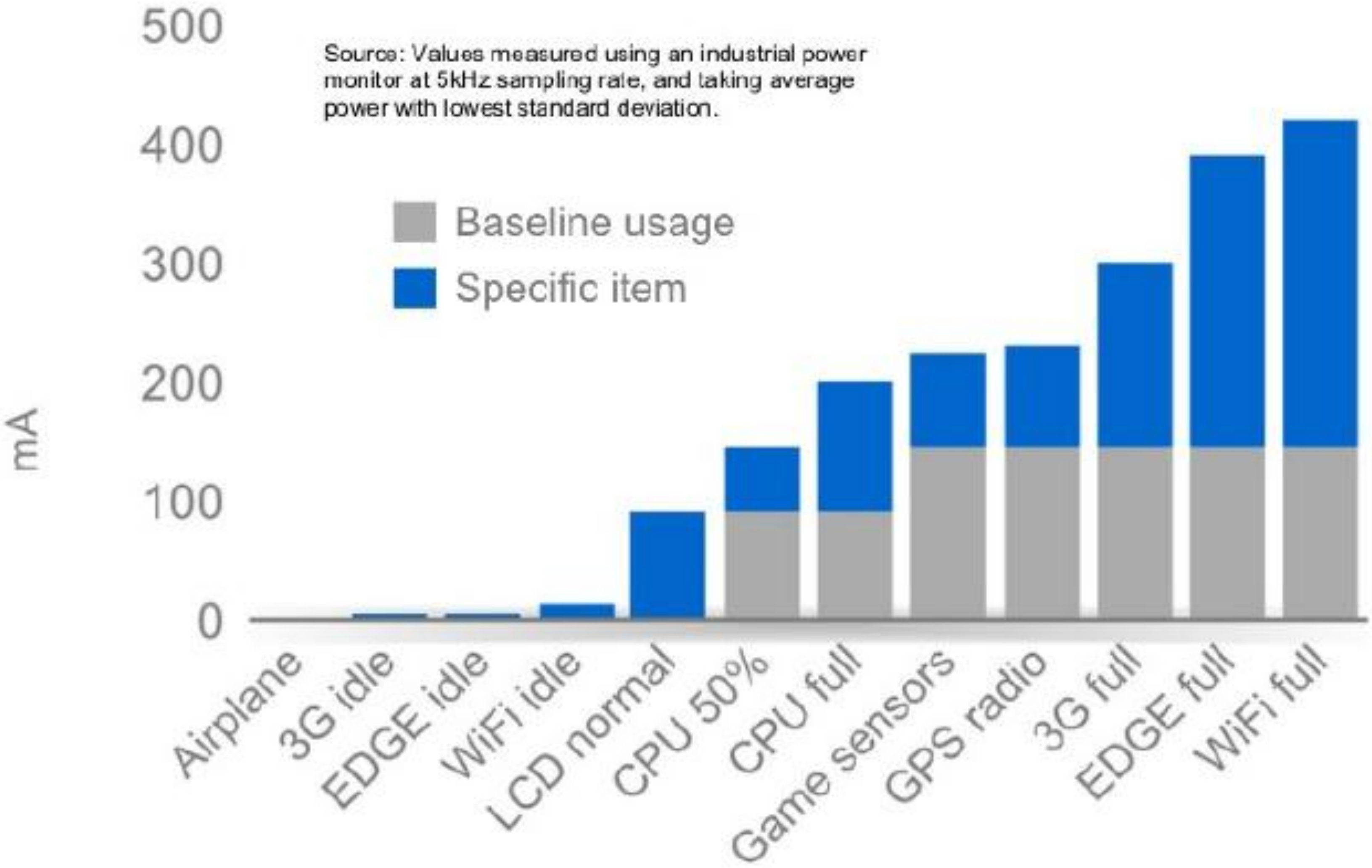
- PARTIAL_WAKE_LOCK is maintained in the kernel, not in Java
- When a PARTIAL_WAKE_LOCK in Java layer is taken, internally in the Kernel a PARTIAL_WAKE_LOCK is taken
- All of the PARTIAL_WAKE_LOCK in the Java layer is protected by one wake lock in the Kernel
- What is it used for ?
  - If a PARTIAL_WAKE_LOCK has been take in java, when system tries to go to sleep, the android will ask the kernel to go to sleep
  - But kernel will check if a PARTIAL_WAKE_LOCK has been taken. If so it will not suspend the CPU
  - CPU could run at a reduced frequency/low power mode for running the background app

Wake Locks

Special behaviour of Partial Wake Lock

- EG : Audio playback
  - When an audio is played, the audio handler, like an ALSA driver, will take a wake lock in the kernel
  - So whenever the device is turned off, we can still hear the audio because the kernel never fully suspend the audio processing

Battery consumption



Source: Values measured using an industrial power monitor at 5kHz sampling rate, and taking average power with lowest standard deviation.

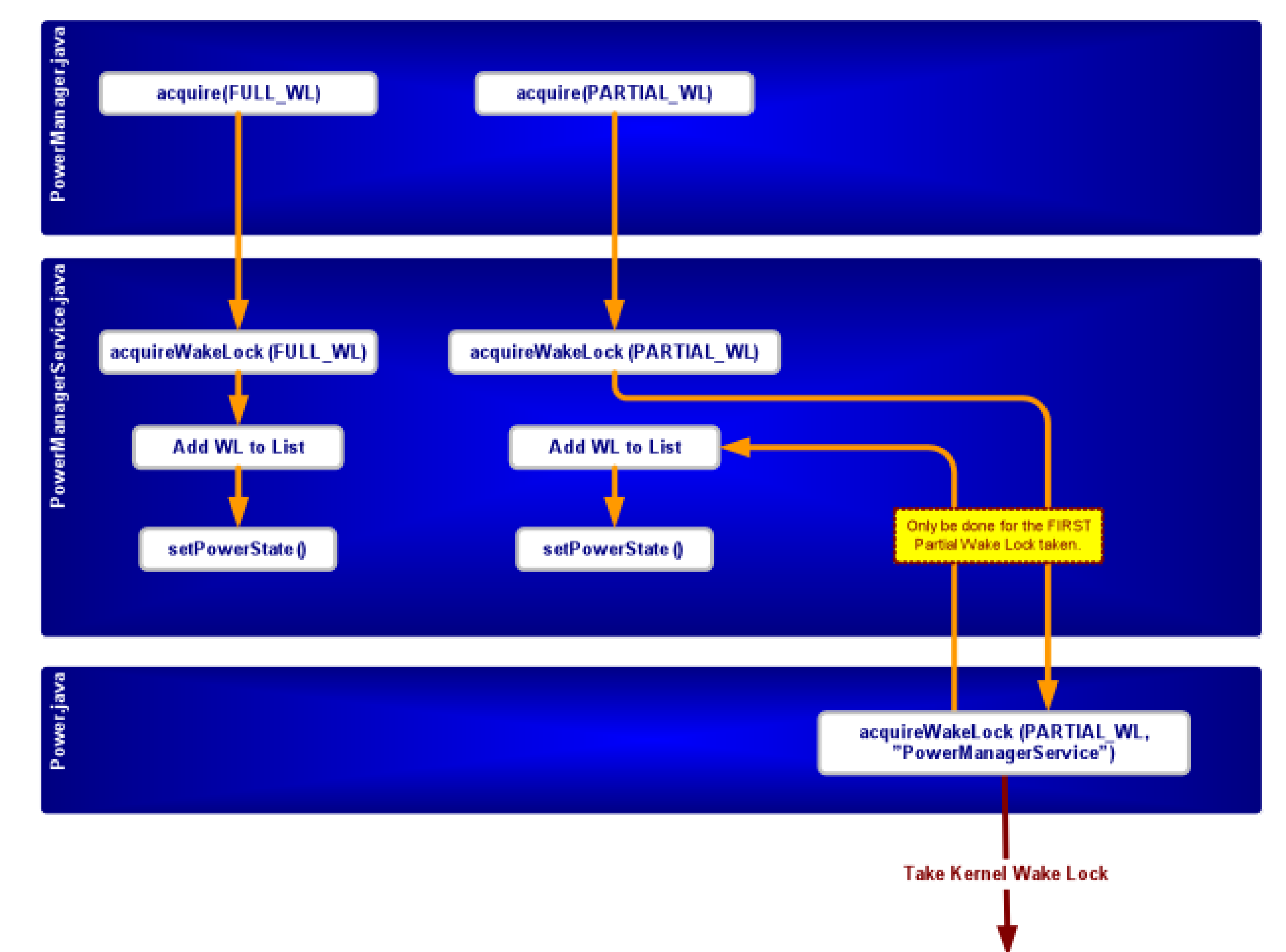
Acquiring Wake Lock

The flow when a Wake Lock is acquired

- Request sent to PowerManager to acquire a wake lock
- PowerManagerService to take a wake lock
- Add wake lock to the list
- Set the power state
  - For a FULL_WAKE_LOCK, PowerState would be set to ON
- For taking Partial wake lock, if it is the first partial wake lock, a kernel wake lock is taken. This will protect all the partial wake locks. For subsequent requests, kernel wake lock is not taken, but just added to the list

Acquiring Wake Lock

The flow when a Wake Lock is acquired



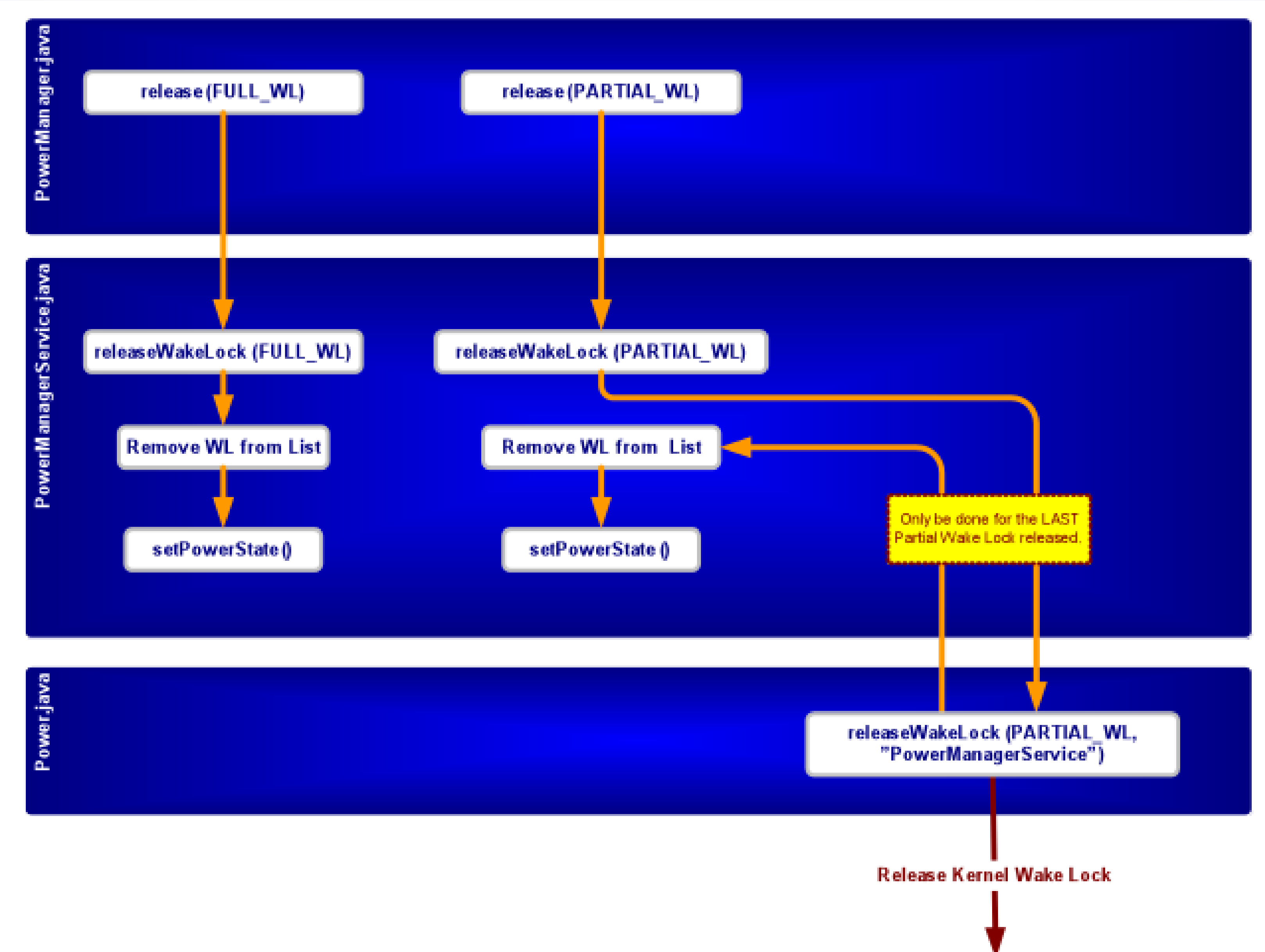
Releasing Wake Lock

The flow when a Wake Lock is released

- Request to release wake lock sent to PowerManager
- Wake Lock removed from the list
- For PARTIAL_WAKE_LOCK release, if the wake lock to be released is the last PARTIAL_WAKE_LOCK, PowerManagerService will also release the wake lock in the kernel. Will bring kernel to suspend
- setPowerState
  - If it is the last wake lock, power state will be set to mem, which will bring the device to standby

Releasing Wake Lock

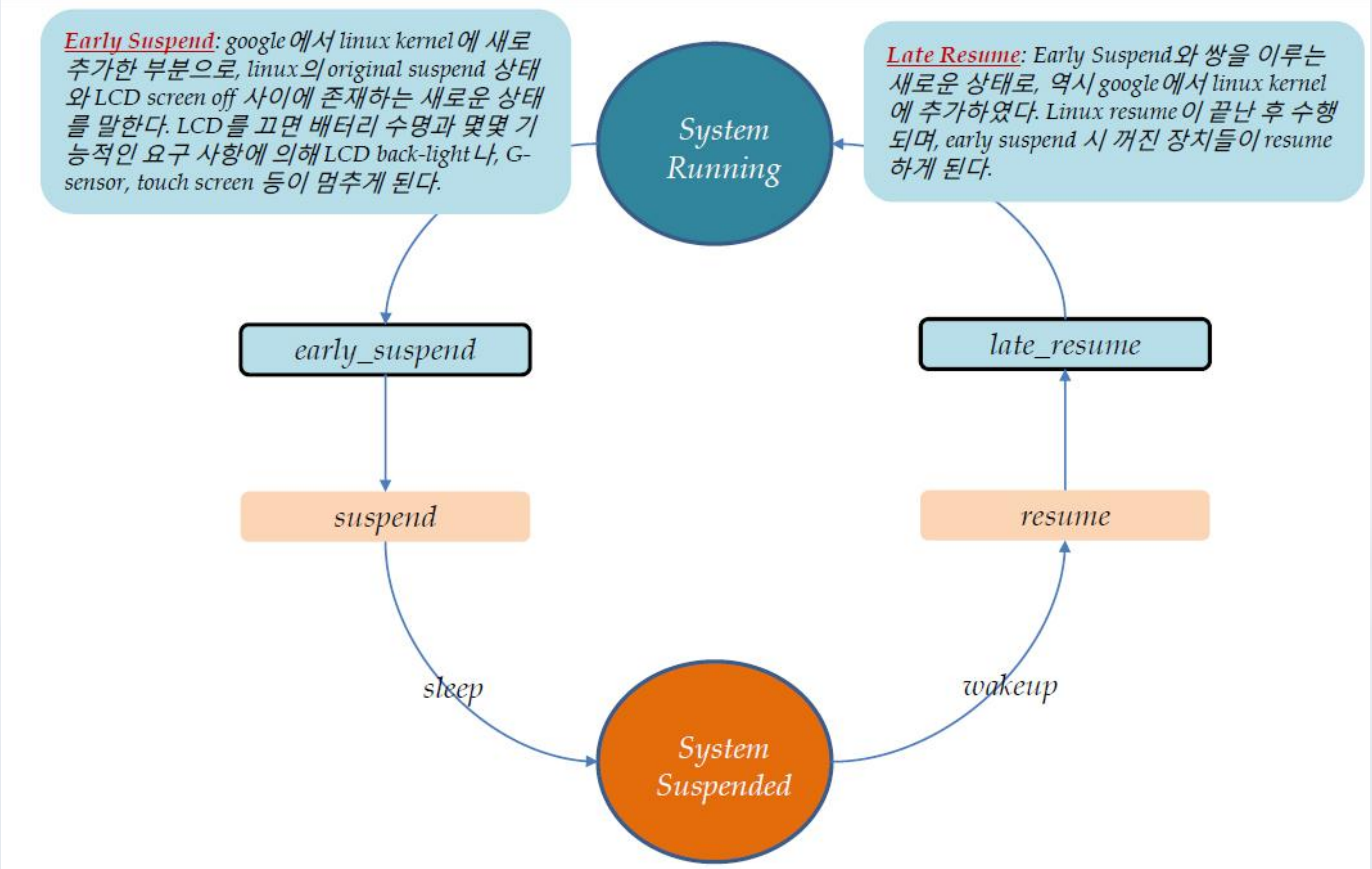
The flow when a Wake Lock is released



Early Suspend

- Extension of Linux Power Management Suspend Hooks
- Used by drivers that need to handle power mode settings to the device before kernel is suspended
- Used to turn off screen and non-wakeup source input devices
- Any driver can register its own early suspend and late_resume handler using register_early_suspend() API
- Unregistration is done using unregister_early_suspend() API
- When the system is brought to suspend mode, early suspend is called first. Depending on how the early suspend hook is implemented, various things can be done

Early Suspend



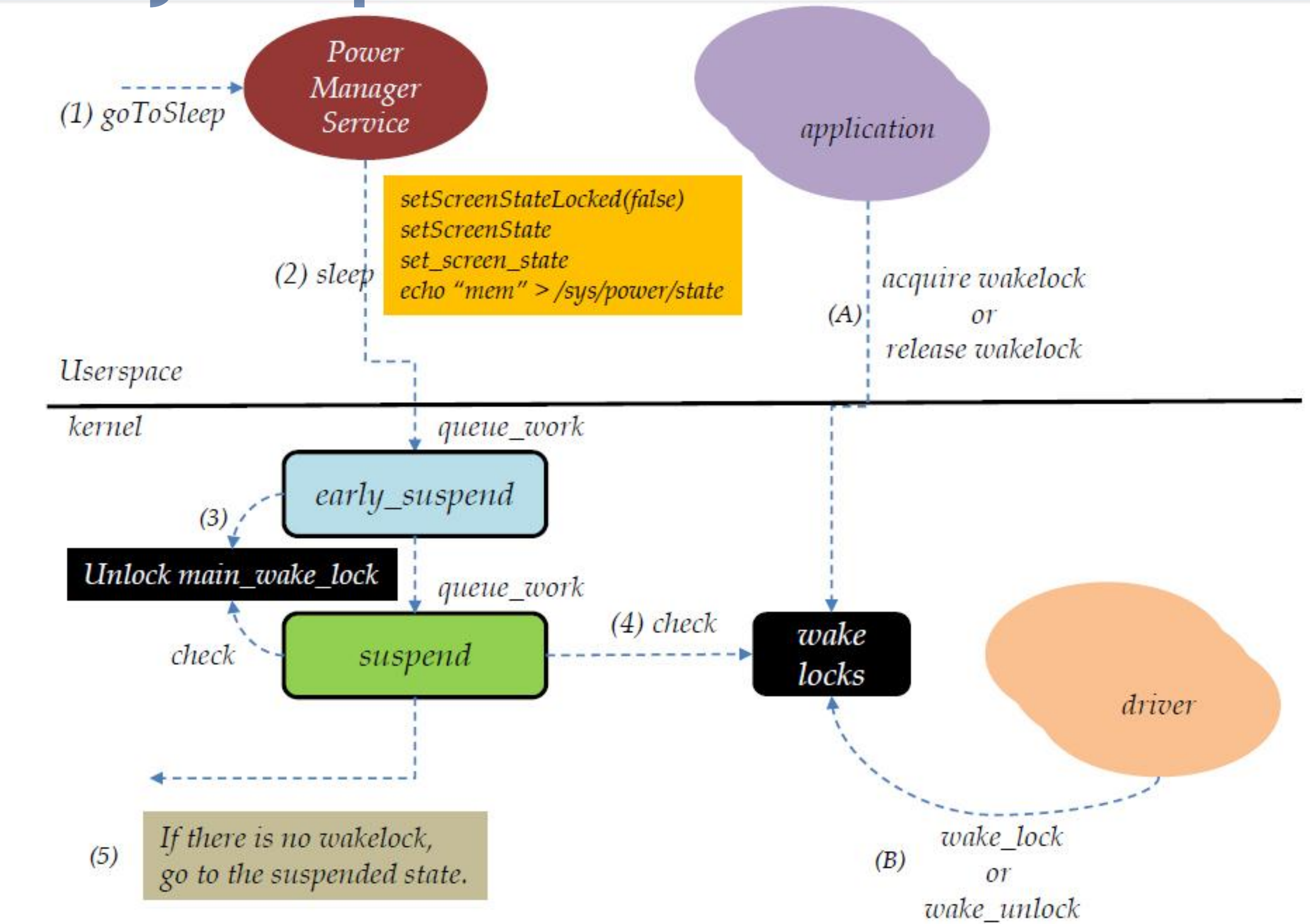
Early Suspend: google에서 linux kernel에 새로 추가한 부분으로, linux의 original suspend 상태와 LCD screen off 사이에 존재하는 새로운 상태를 말한다. LCD를 끄면 배터리 수명과 몇몇 기능적인 요구 사항에 의해 LCD back-light나, G-sensor, touch screen 등이 멈추게 된다.

Late Resume: Early Suspend와 쌍을 이루는 새로운 상태로, 역시 google에서 linux kernel에 추가하였다. Linux resume이 끝난 후 수행되며, early suspend 시 꺼진 장치들이 resume하게 된다.

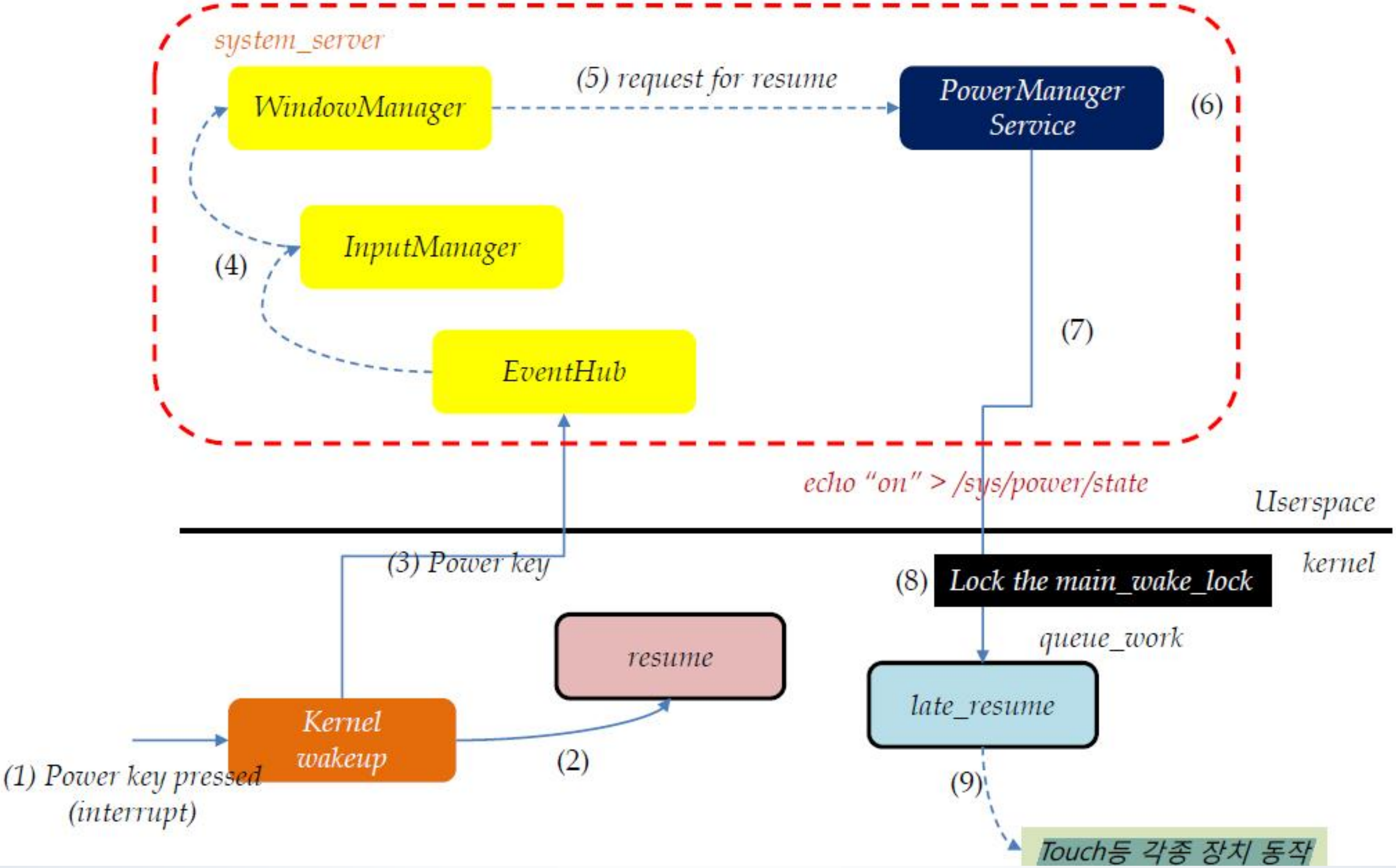
Early Suspend

- For e.g. consider a display driver
  - In early suspend, the screen can be turned off
  - In the suspend, other things like closing the driver can be done
- When system is resumed, resume is called first, followed by resume late

Early Suspend



Resume Late

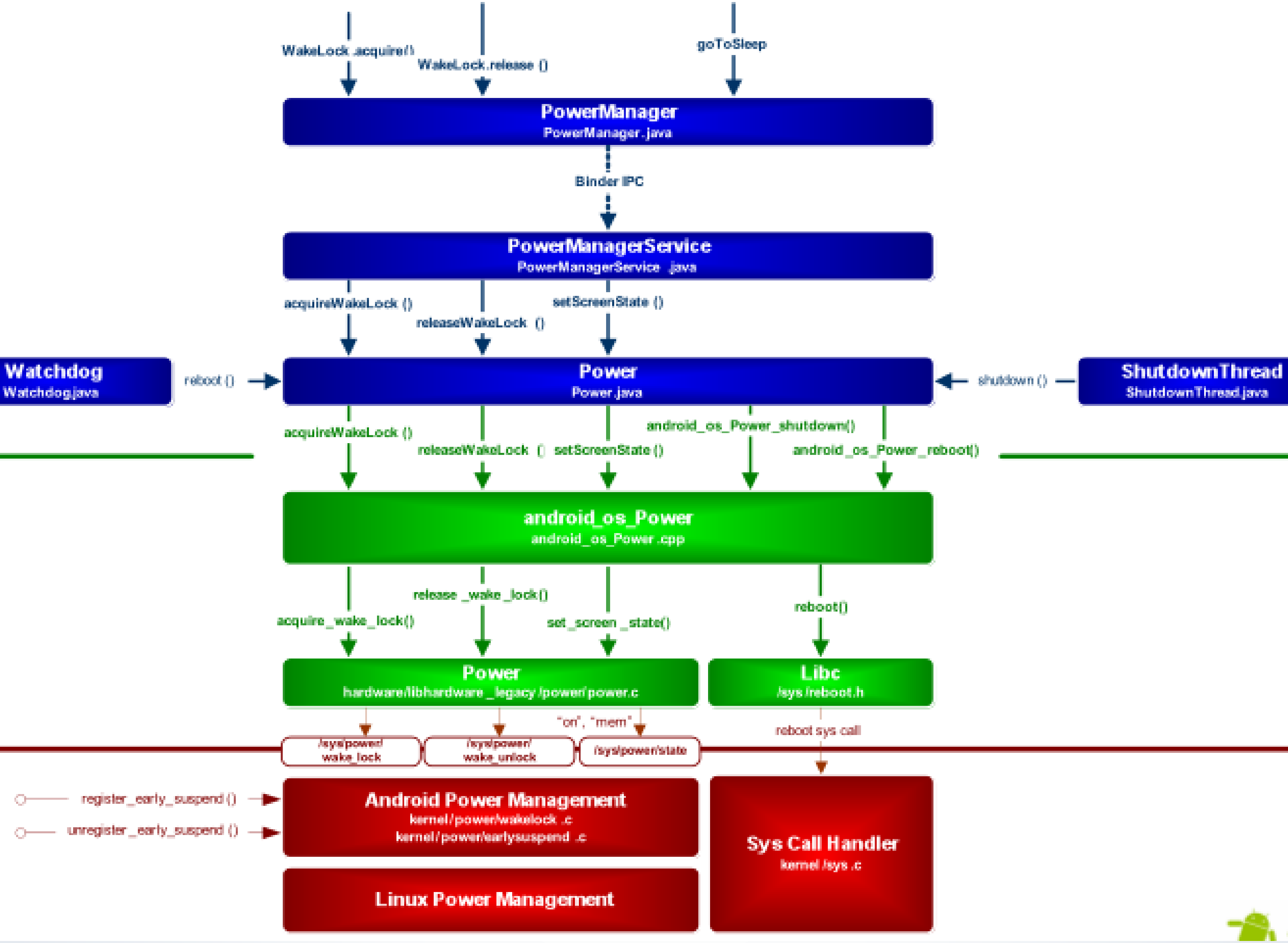

system_server
WindowManager
(5) request for resume
PowerManager Service
(6)
InputManager
(4)
EventHub
(7)
echo "on" > /sys/power/state
Userspace
kernel
(3) Power key
(8)
Lock the main_wake_lock
queue_work
resume
late_resume
Kernel wakeup
(1) Power key pressed (interrupt)
(2)
(9)
Touch등 각종 장치 동작

System Sleep

- API to bring device to sleep when we press the power button
- Require DEVICE_POWER permission
- Can only be called in system process context by verifying uid and pid
- When power button is presed, an API goToSleep() is called in the PowerManager

System Sleep

- goToSleep() will force release all wake locks
- When force releasing all locks, power state will be set to off
- In the JNI bridge there is a function setScreenState. setScreenState is set to off
- Then setPowerState to mem, ie write a mem to /sys/power/state



WakeLock.acquire()

WakeLock.release ()

goToSleep

PowerManager
PowerManager.java

Binder IPC

PowerManagerService
PowerManagerService.java

acquireWakeLock ()

releaseWakeLock ()

setScreenState ()

Watchdog
Watchdog.java

reboot ()

Power
Power.java

shutdown ()

ShutdownThread
ShutdownThread.java

acquireWakeLock ()

releaseWakeLock ()

setScreenState ()

android_os_Power_shutdown()

android_os_Power_reboot()

android_os_Power
android_os_Power.cpp

acquire_wake_lock()

release_wake_lock()

set_screen_state()

reboot()

Power
hardware/libhardware_legacy/power/power.c

Libc
/sys/reboot.h

"on", "mem"

reboot sys call

/sys/power/
wake_lock

/sys/power/
wake_unlock

/sys/power/state

register_early_suspend ()

unregister_early_suspend ()

Android Power Management
kernel/power/wakelock.c
kernel/power/earlysuspend.c

Linux Power Management

Sys Call Handler
kernel/sys.c

Battery Service

- The BatteryService monitors the battery status, level, temperature etc.
- A Battery Driver in the kernel interacts with the physical battery via ADC [to read battery voltage] and I^2C (Inter-Integrated Circuit: a multi-master serial single-ended computer bus used to attach low-speed peripherals to an electronic device)
- Whenever BatteryService receives information from the BatteryDriver, it will act accordingly

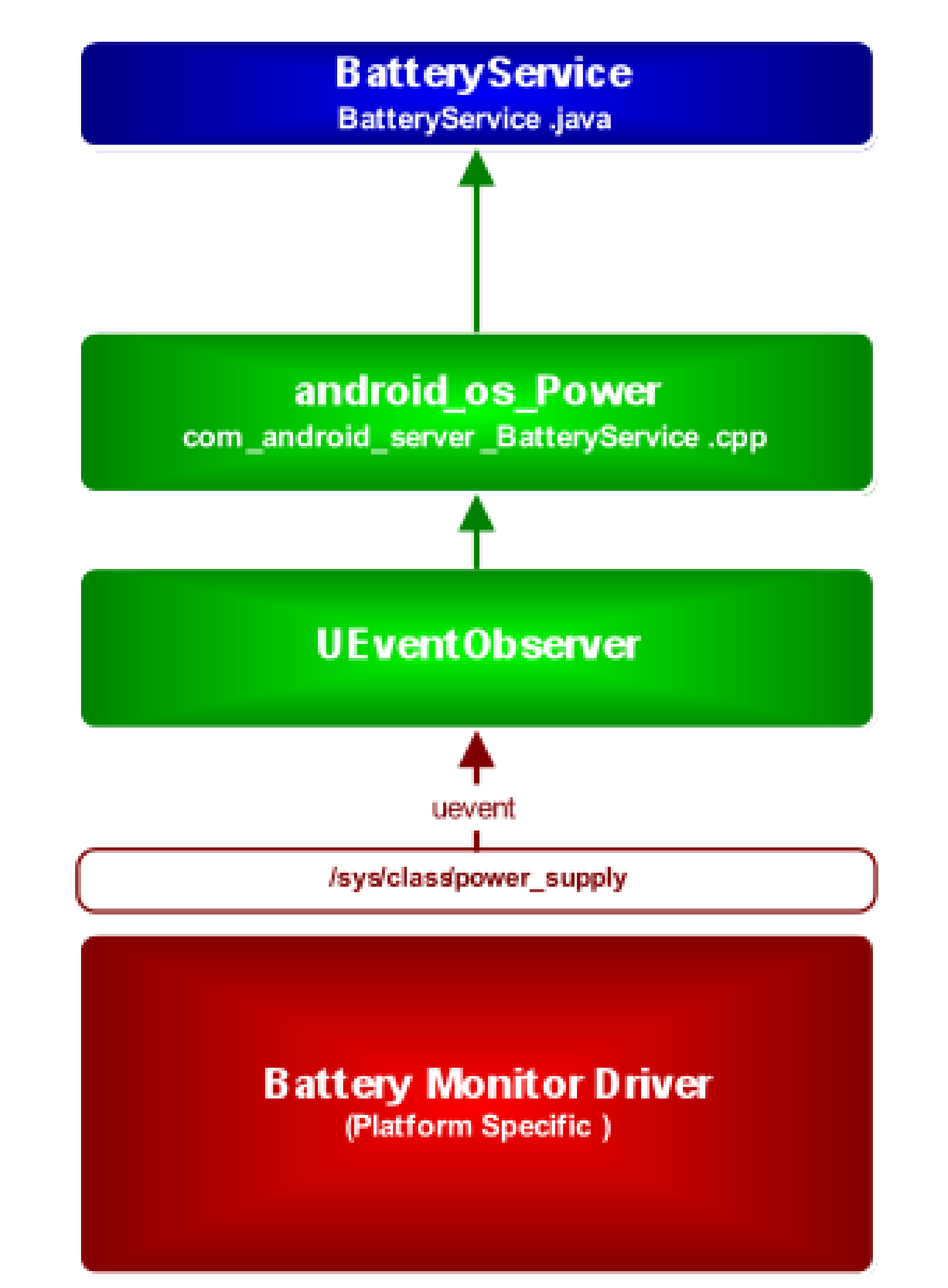
E.g. if battery level is low, it will ask system to shutdown

Battery Service

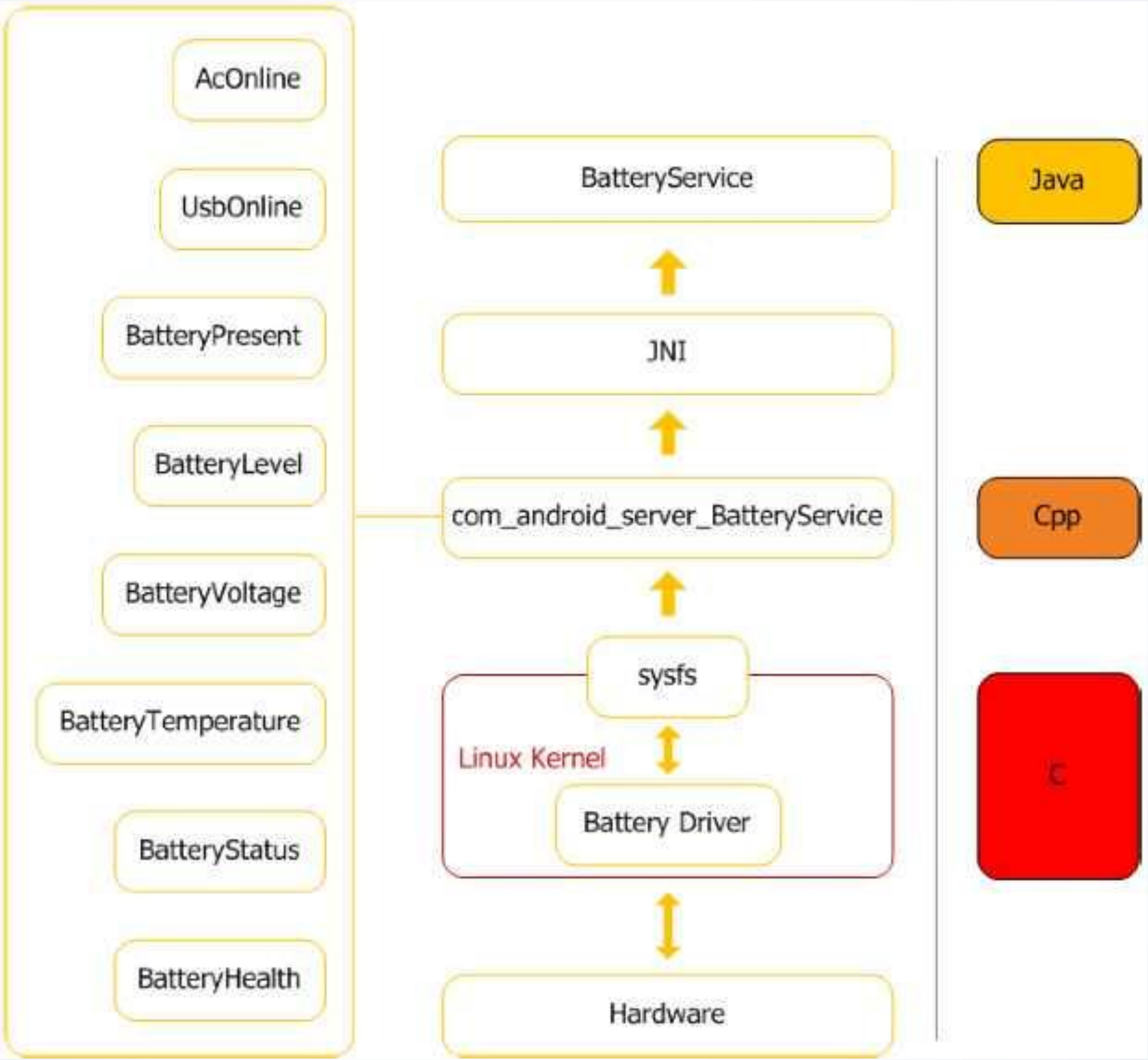
- Using power supply class in Linux Kernel

  /sys/class/power_supply
- Utilize uevent mechanism to update battery status
- uevent : An asynchronous communication channel for kernel
- Battery Service will monitor the battery status based on received uevent from the kernel

Battery Service



Battery Service



References

- Android Power Management Hacks, Slow Boot
- Power Management from Linux Kernel to Android, Matt Hsu & Jim Huang, 0xlab
- Analysis of the Android Architecture. Stefan Brahler