



# CS353 Linux Kernel

---

Chentao Wu 吴晨涛  
Associate Professor  
Dept. of CSE, SJTU  
[wuct@cs.sjtu.edu.cn](mailto:wuct@cs.sjtu.edu.cn)



# Power Management

From Linux Kernel to Android



# Outline


---

- ❑ Introduction on Power Management
- ❑ Linux Power Management
- ❑ Android Power Management
- ❑ Wake Lock
- ❑ System Sleep (Suspend)
- ❑ Battery Service





# Introduction

- ❑ Power Management (PM) in Android is designed for mobile devices
- ❑ Goal: prolong the life of battery 
- ❑ Build on top of Linux Power Management
  - ❑ Not directly suitable for a mobile device
- ❑ Designed for devices which have a 'default-off' behavior
  - ❑ 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** behavior





# Linux Power Management

---

- ❑ Two popular power management standards in Linux
  - ❑ APM (Advanced Power Management)
  - ❑ ACPI (Advanced Configuration and Power Interface)





# APM (Advanced Power Management)

---

- ❑ 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 (e.g. USB, IEEE 1394)





# APM (contd.)

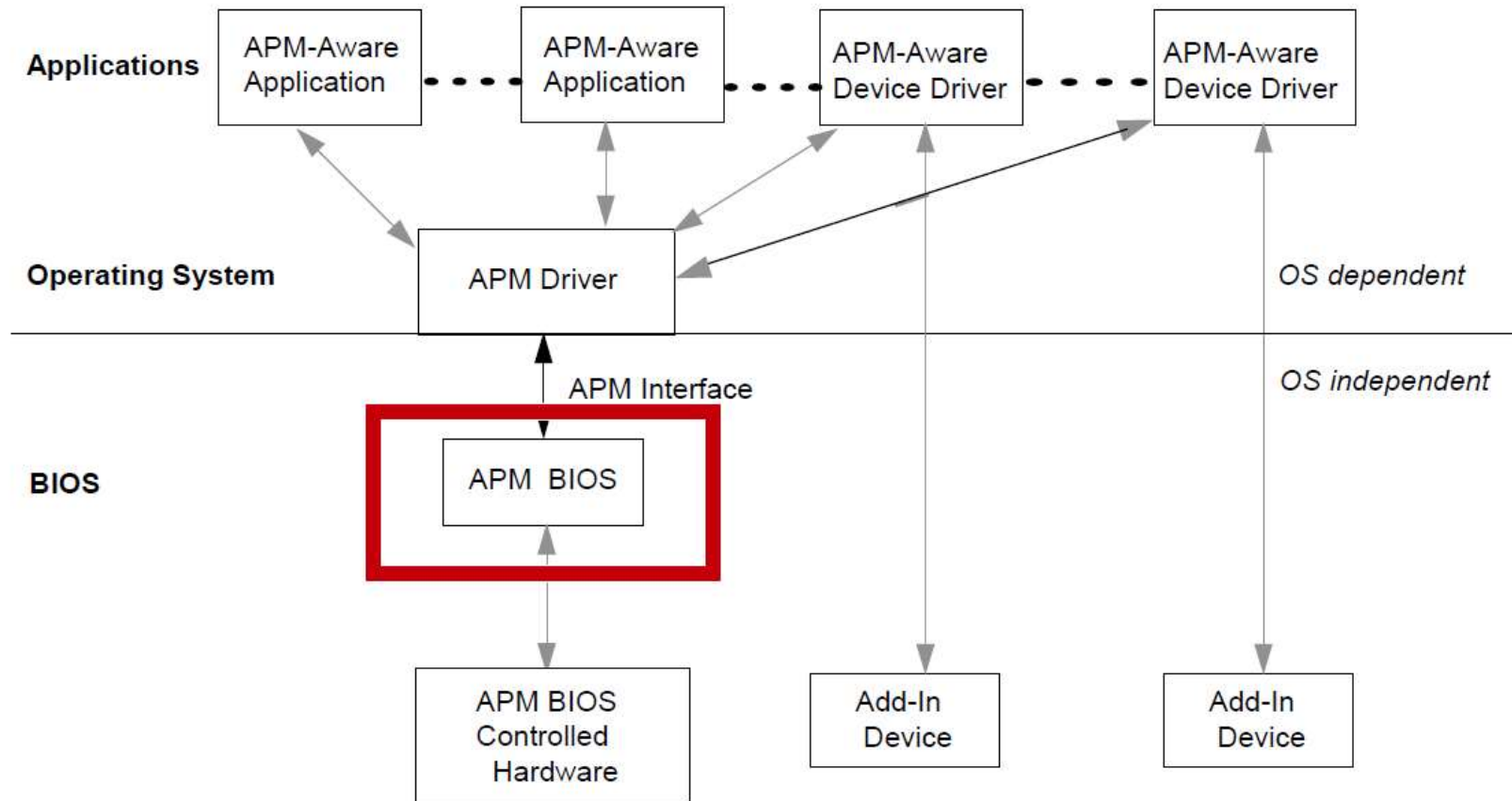
---

- 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





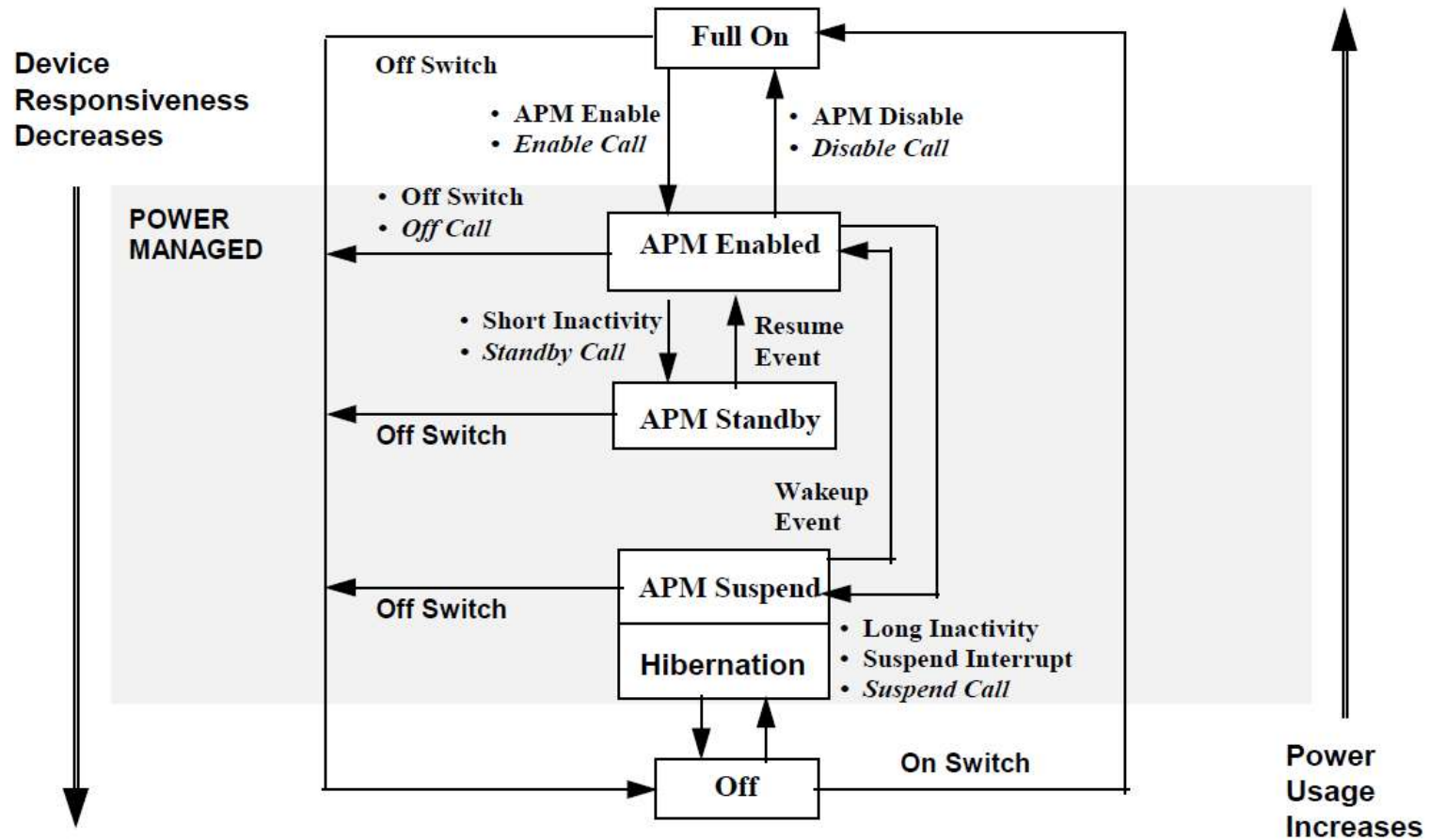
# APM (contd.)







# APM (contd.)



# ACPI (Advanced Configuration and Power Interface)

---

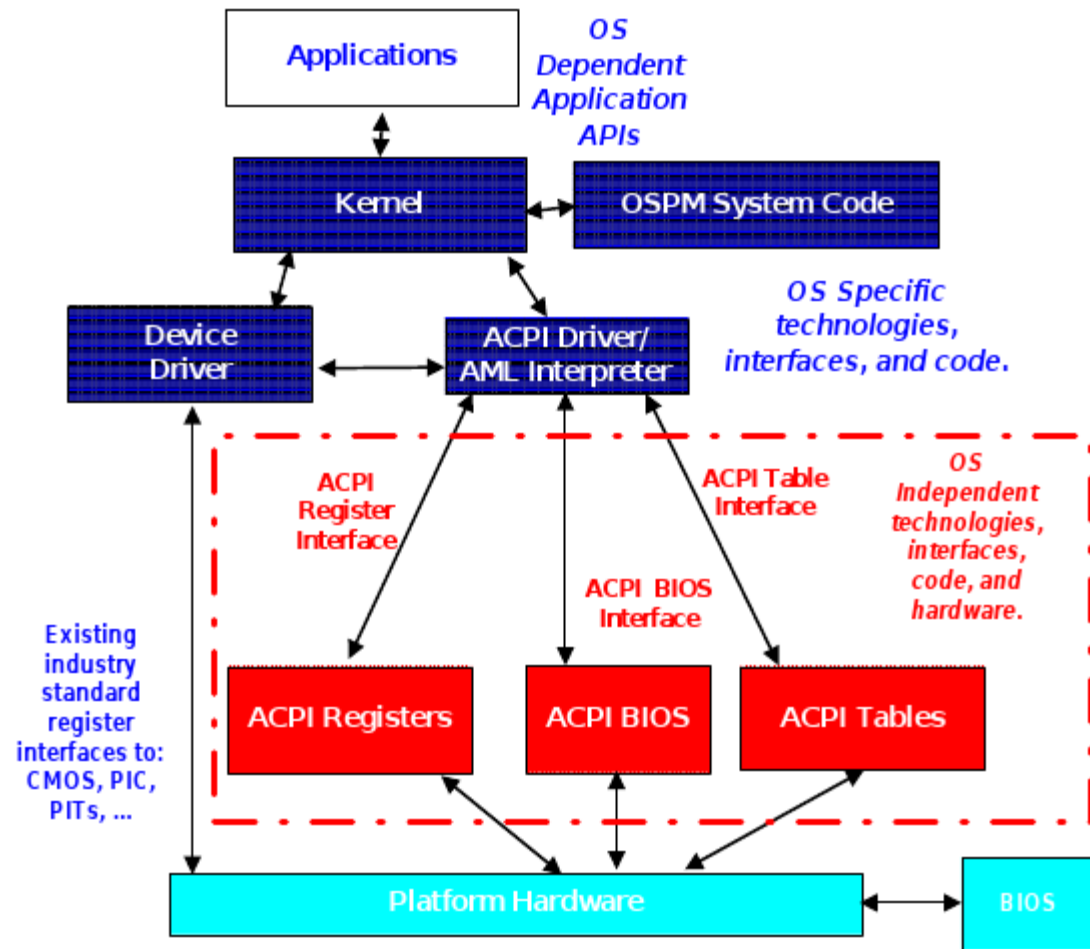


- ❑ Control divided between BIOS and OS
- ❑ Decisions managed through the OS
- ❑ Enables sophisticated power policies for general-purpose computers with standard usage patterns and hardware
- ❑ No knowledge of device-specific scenarios (e.g. need to provide predictable response times or to respond to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





# ACPI (contd.)





# ACPI (contd.)

---

- ❑ 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





# ACPI (contd.)

---

- ❑ 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(memory): 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





# ACPI (contd.)

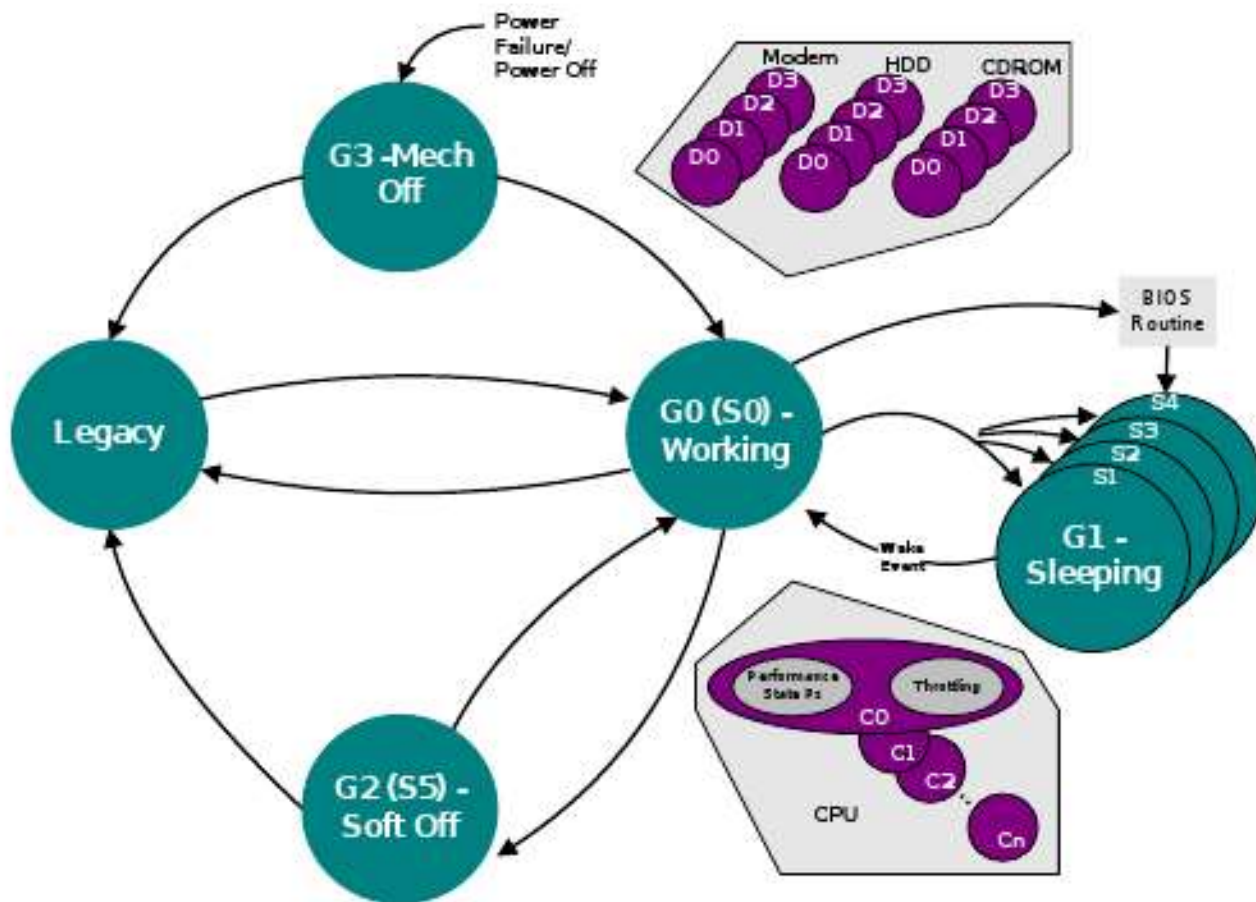
---

- 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.





# ACPI (contd.)





# Command of Linux PM

---

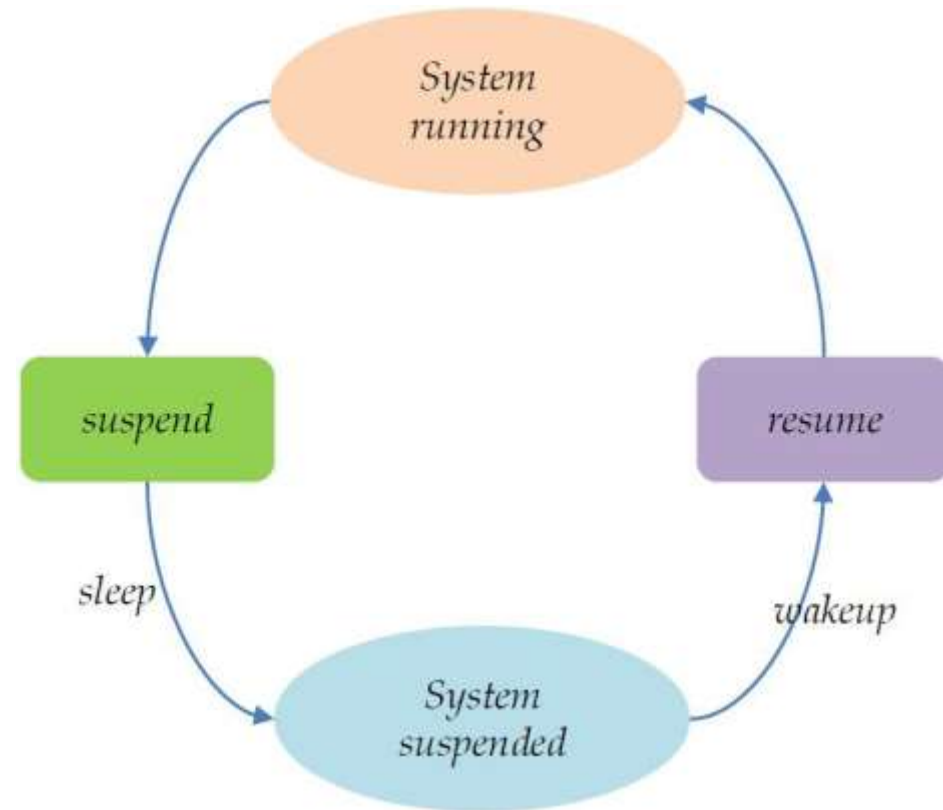
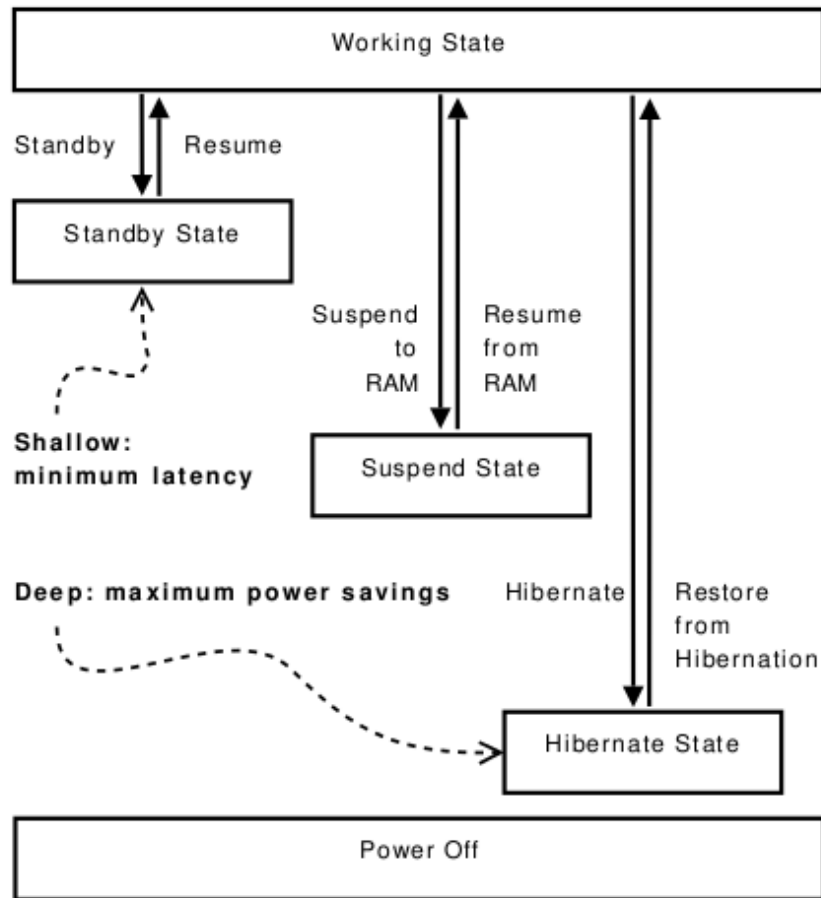
- Power mode interface is on sysfs
  - `/sys/power/state`
- sysfs is a virtual file system provided by Linux.
  - sysfs exports information about devices and drivers from the kernel device model to user space, and is also used for configuration
- Changing state done by
  - `# echo mem > /sys/power/state`
  - `# echo disk > /sys/power/state`
  - `# echo standby > /sys/power/state`





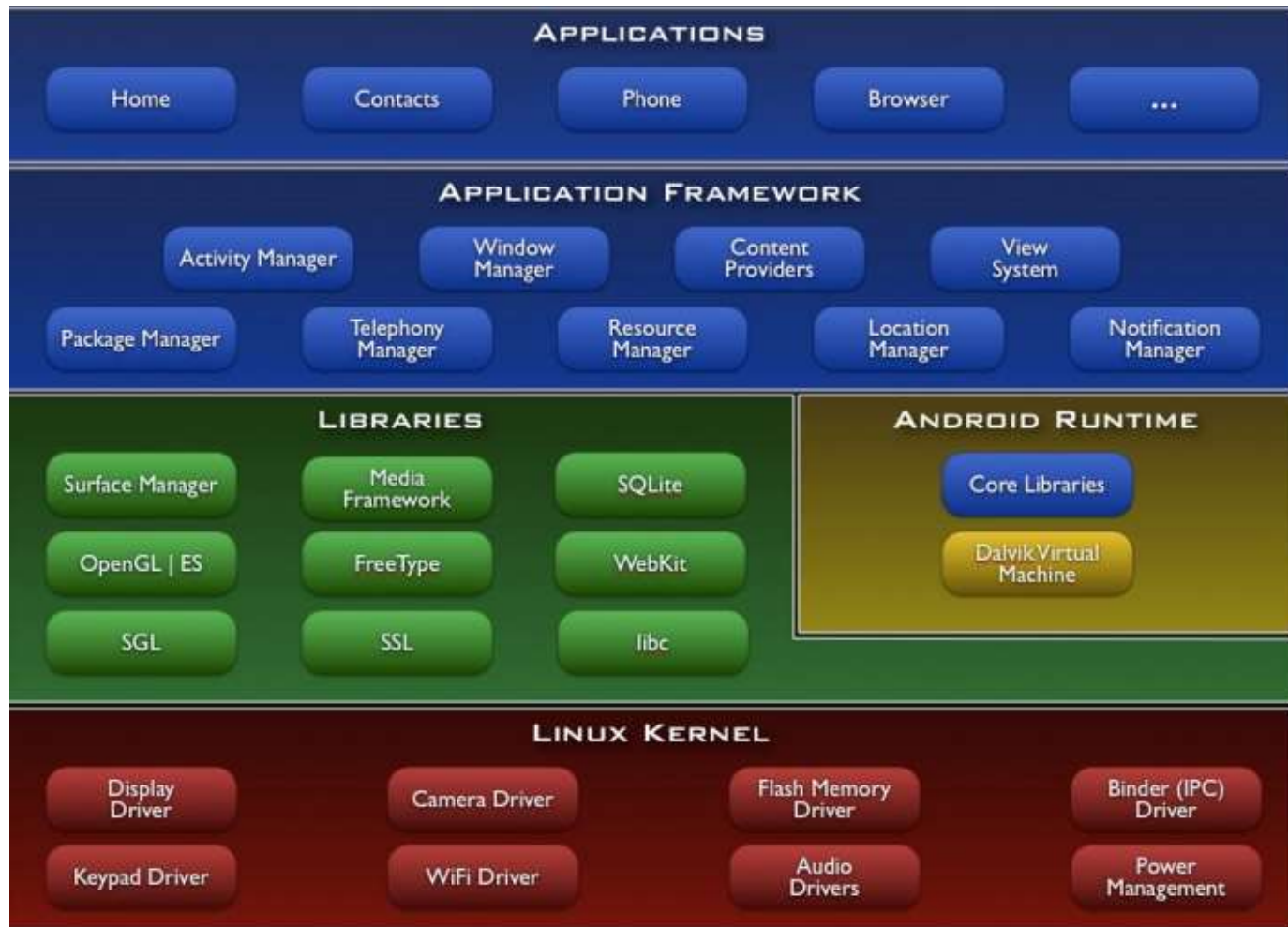


# Overview of Linux PM





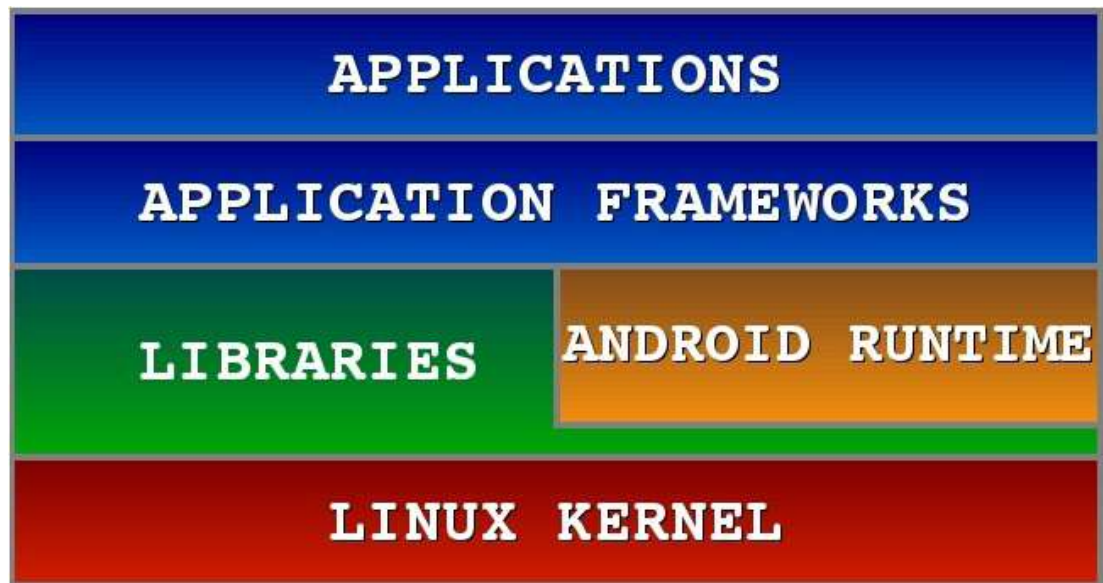
# Android Architecture





# In fact, power ghost exists everywhere

- ❑ Let “**grep -ri power**” tell you about the details!
- ❑ It is layered into several components, but implementation involves in some hacks.
- ❑ Check: CONFIG\_PM, sysfs, device drivers, libhardware\_legacy, libril, init.rc, powerd, alarm, vold, JNI, PowerManager, BatteryManager, ...





# Android PM Design

---

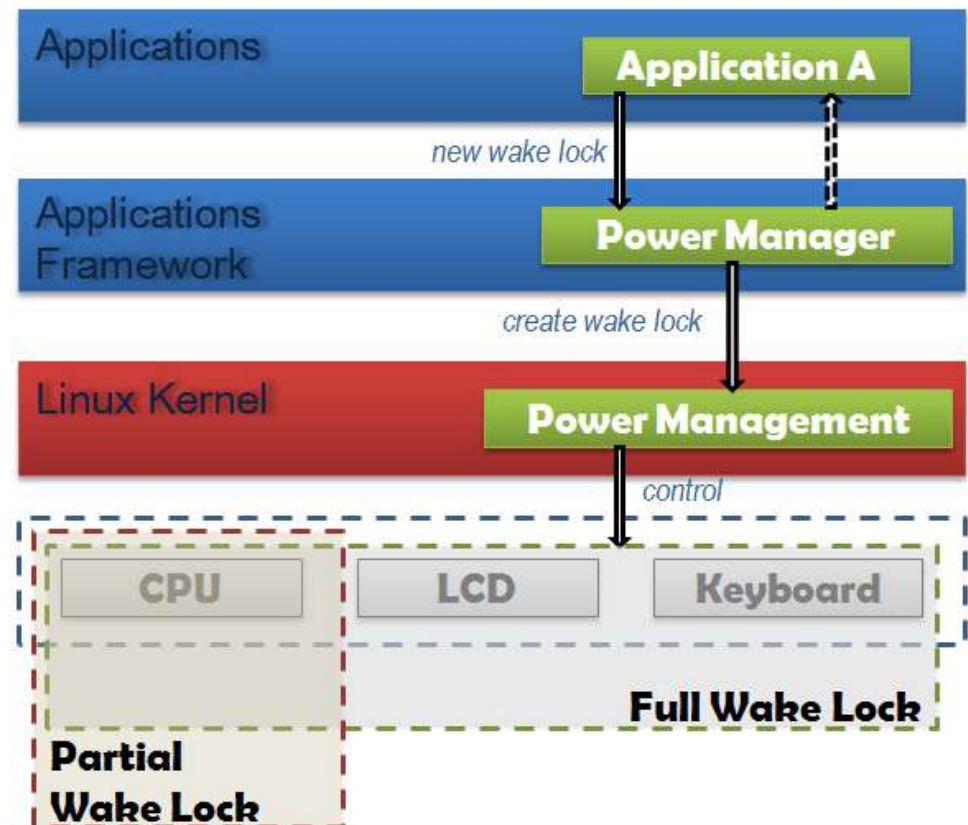
- ❑ Built as a wrapper to Linux Power Management
- ❑ In the Kernel
  - ❑ Added 'on' state in the power state
  - ❑ Added Early Suspend framework
  - ❑ Added Partial Wake Lock mechanism
- ❑ Apps and services must request CPU resource with '**wake locks**' through the Android application framework and 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





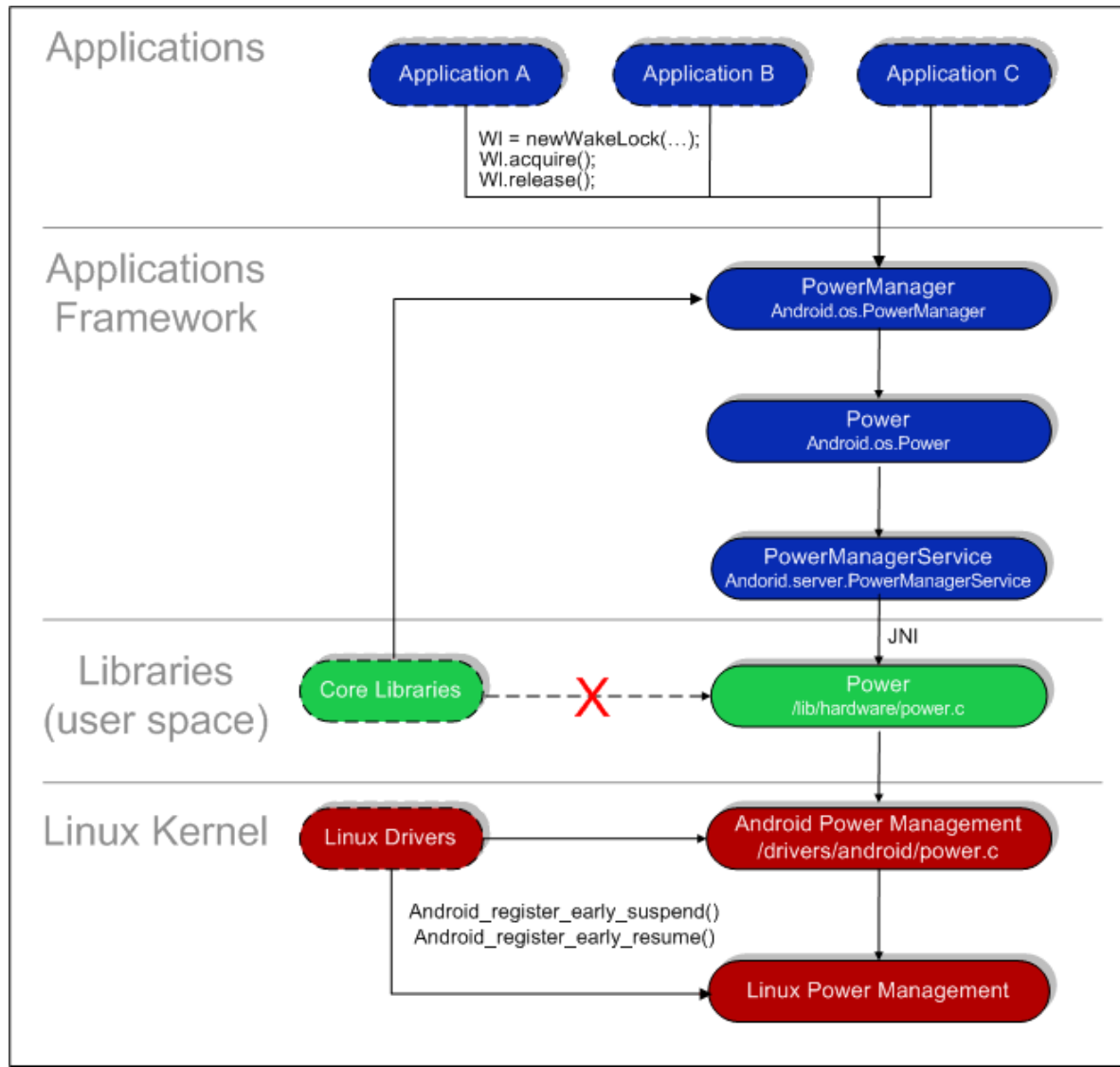
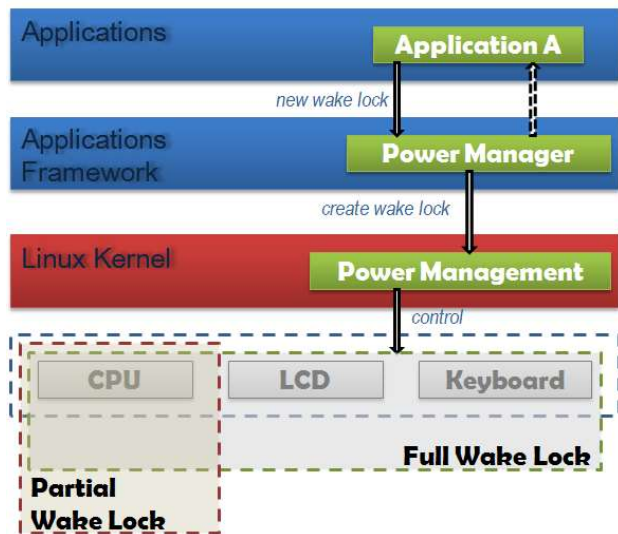
# Android PM Architecture

- Through the framework, user space applications can use '**PowerManger**' class to control the power state of the device
- If there are no active wake locks, CPU will be turned off.
- If there is are partial wake locks, screen and keyboard will be turned off.





# Android PM Architecture (contd.)





# Type of Wake Locks

---

- ❑ **ACQUIRE\_CAUSES\_WAKEUP**
  - ❑ Normally wake locks don't actually wake the device, they just cause it to remain on once it's already on. Think of the video player app as the normal behavior.
- ❑ **FULL\_WAKE\_LOCK**
  - ❑ Wake lock that ensures that the screen and keyboard are on at full brightness.
- ❑ **ON\_AFTER\_RELEASE**
  - ❑ When this wake lock is released, poke the user activity timer so the screen stays on for a little longer.
- ❑ **PARTIAL\_WAKE\_LOCK**
  - ❑ Wake lock that ensures that the CPU is running. The screen might not be on.







# Type of Wake Locks (contd.)

---

## □ 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.

## □ 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.







# PM State Machine

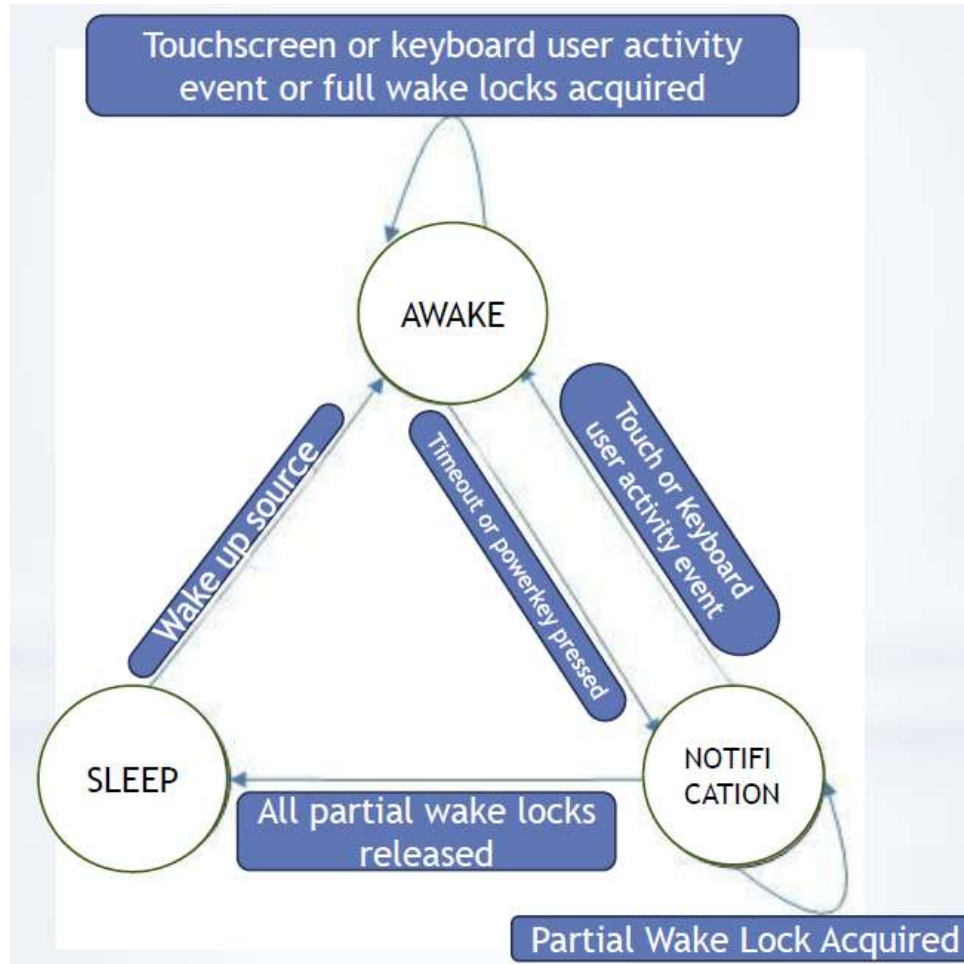
---

- ❑ 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'





# PM State Machine (contd.)





# Android PM Implementation

---

- ❑ Android PM Framework provides a service for user space applications through the class `PowerManager` 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 (1)

---

## 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 (2)

---

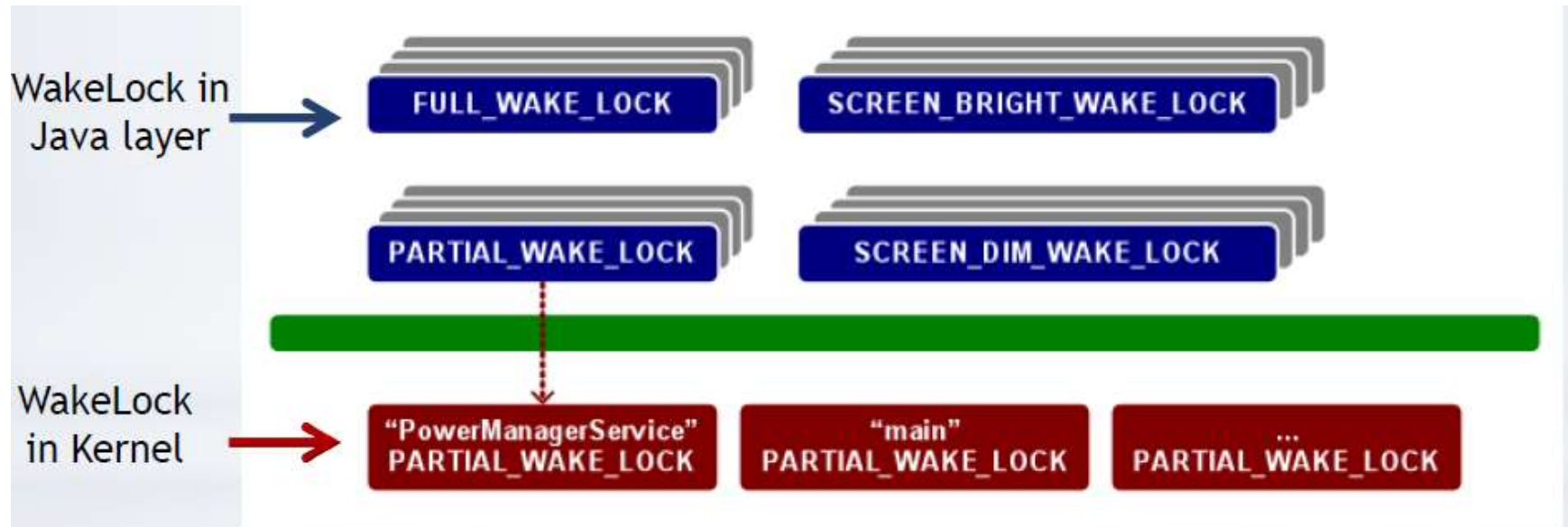
## How are Wake Locks Managed

- 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 (3)





# Wake Locks (4)

---

## 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







# Partial Wake Locks (1)

---

## Special behavior 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





# Partial Wake Locks (2)

---

## Example

### □ Audio playback

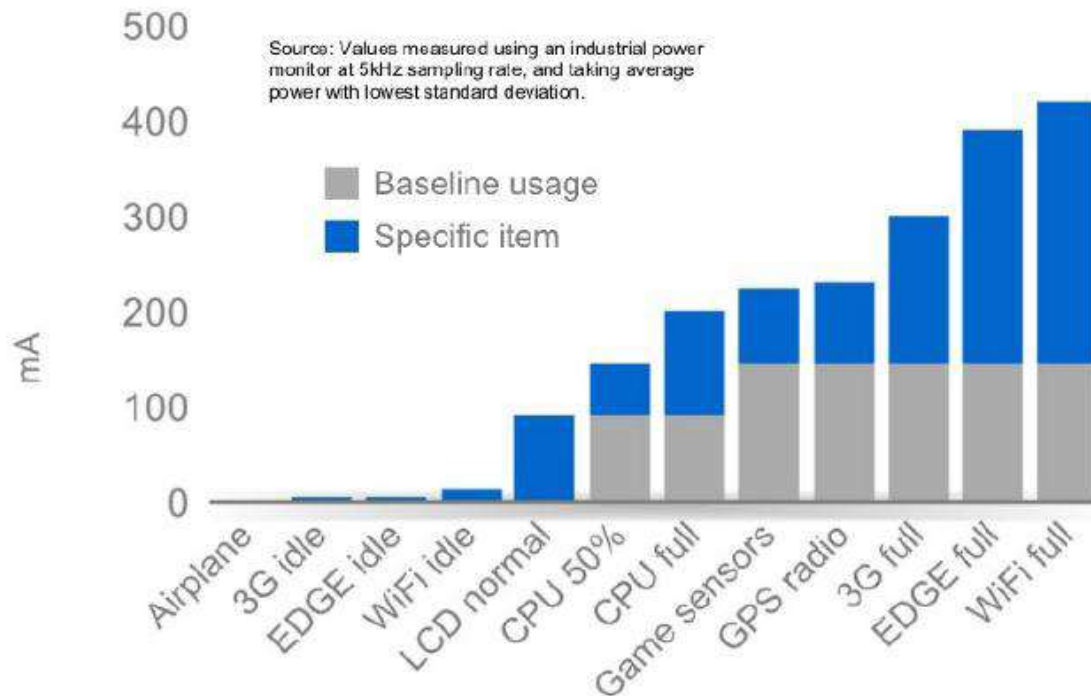
- When an audio is played, the audio handler, like an ALSA driver, will take a wake lock in the kernel
- So whenever the device is turned off, we can still hear the audio because the kernel never fully suspend the audio processing





# Partial Wake Locks (3)

## Battery consumption





# Acquiring Wake Lock (1)

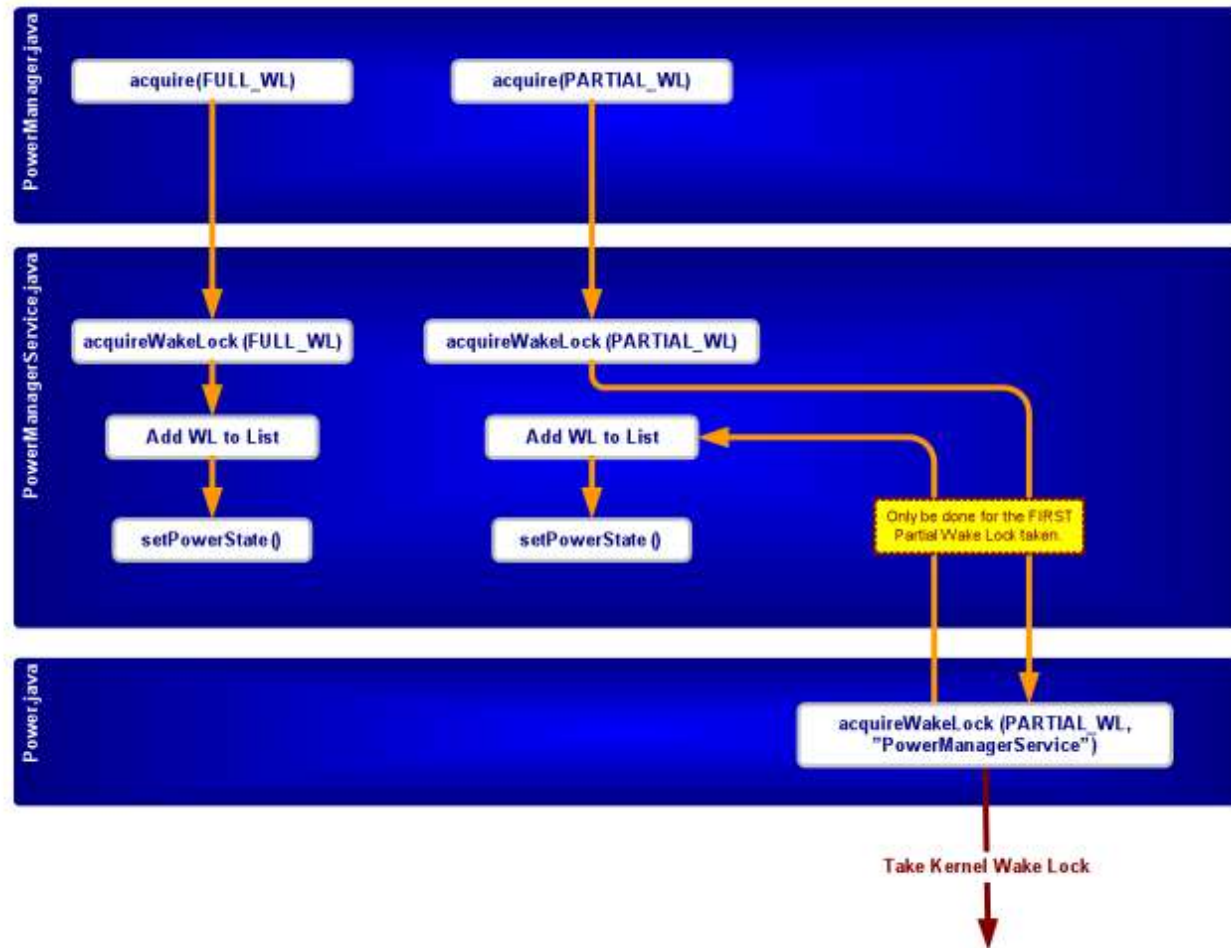
---

- ❑ 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 (2)





# Releasing Wake Lock (1)

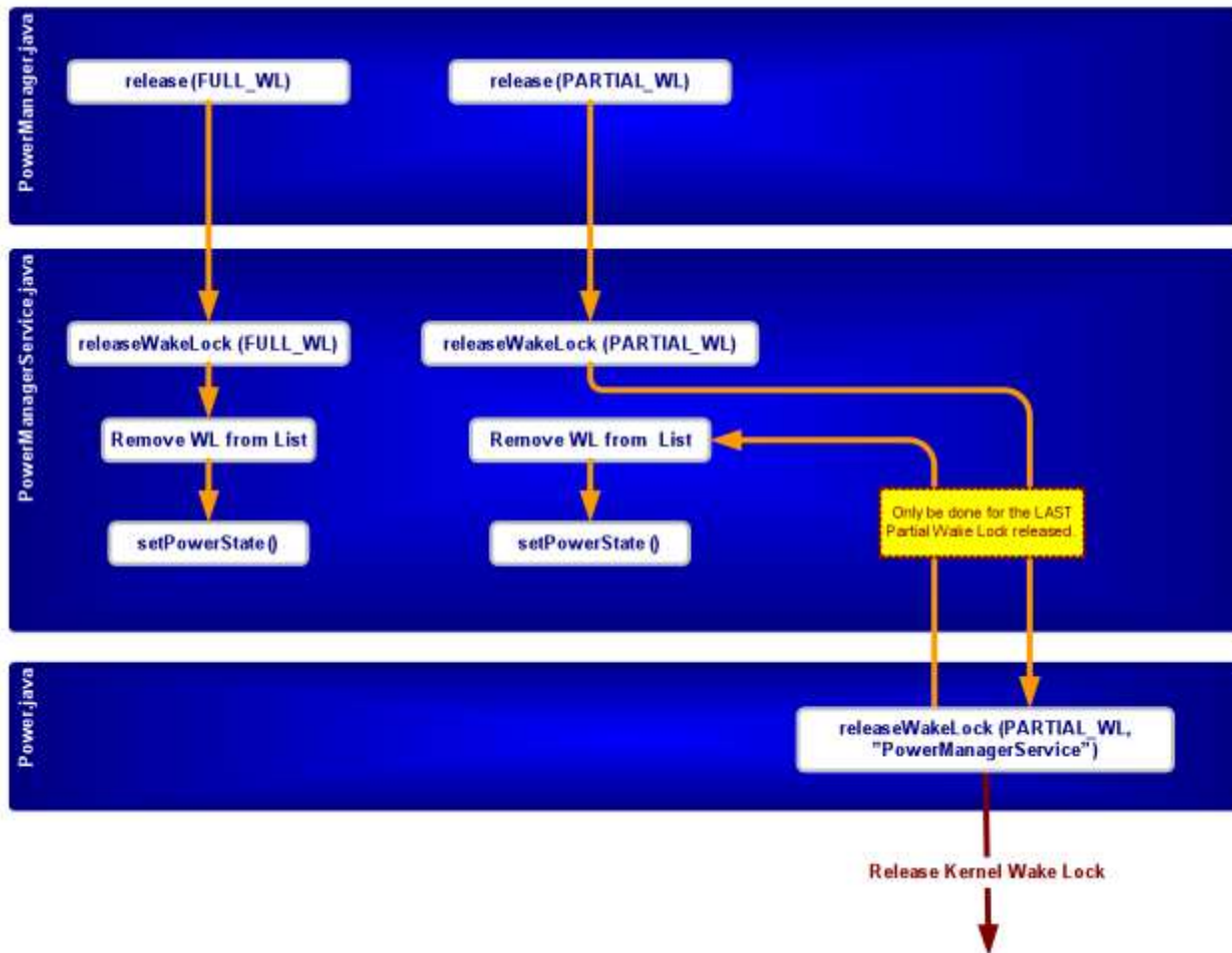
---

- ❑ 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 (2)





# Early Suspend (1)

---

- ❑ 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 (2)

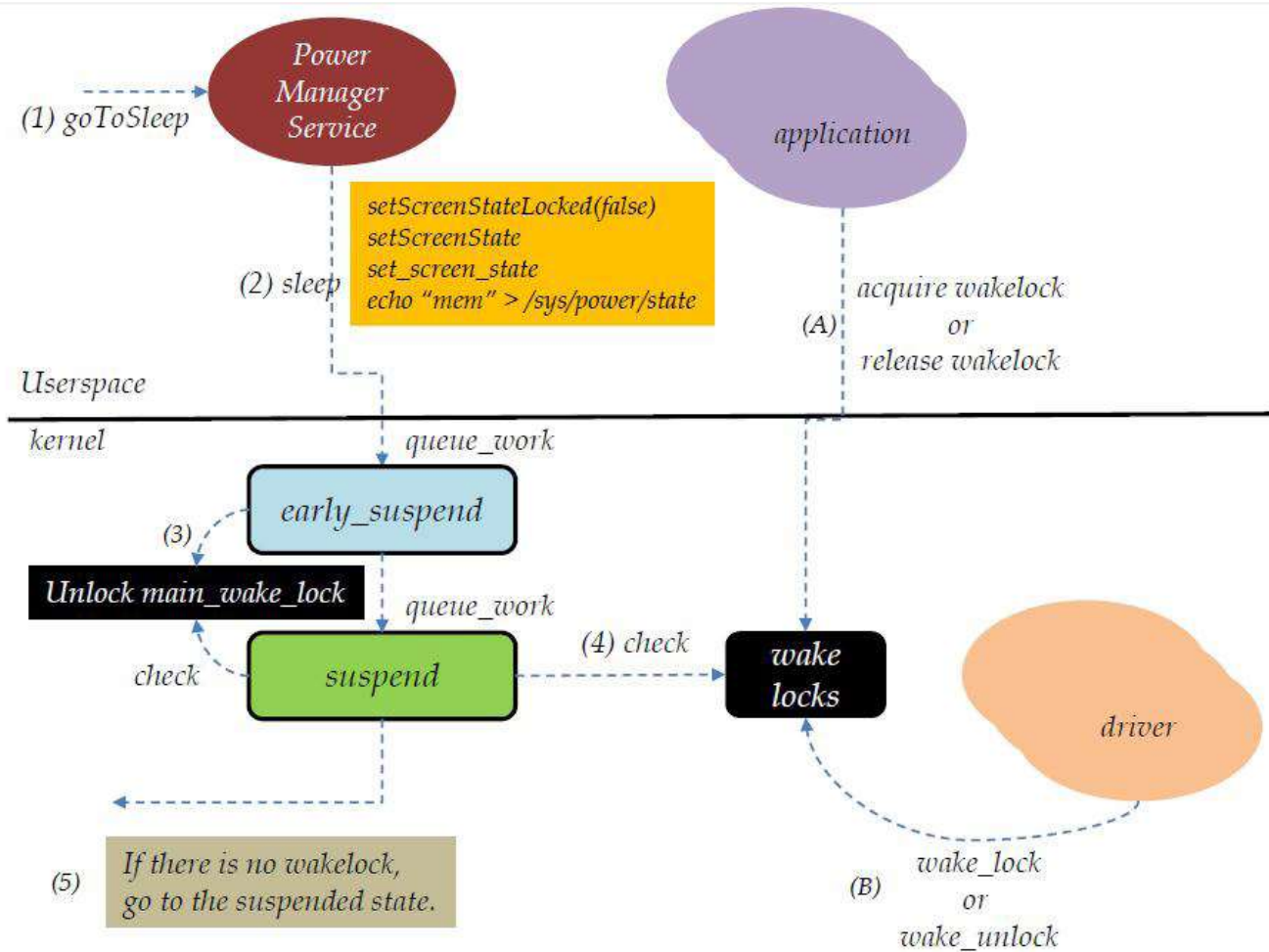
---

- 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





# Early Suspend (3)





# Resume Late (1)

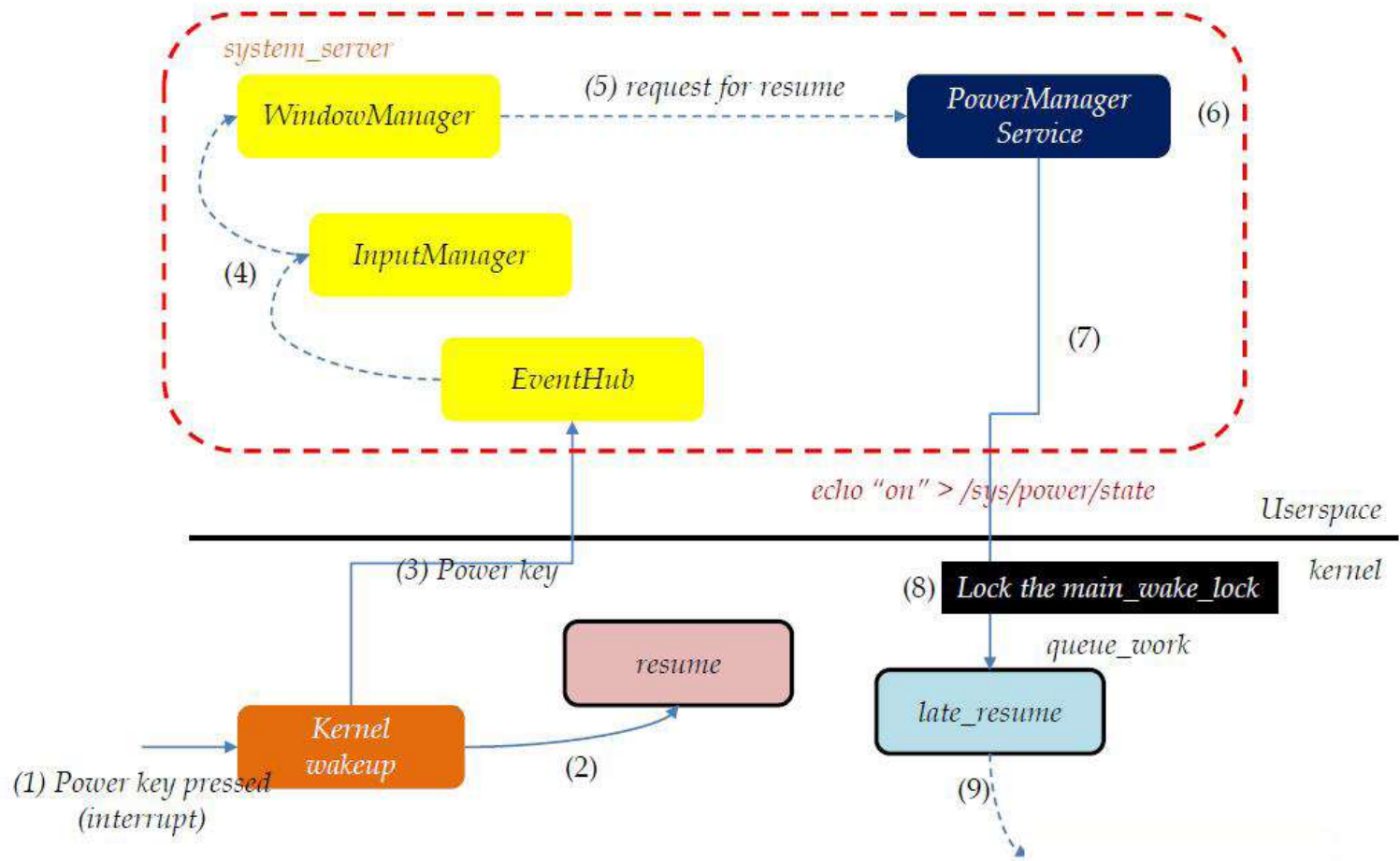
---

- When system is resumed, resume is called first, followed by resume late

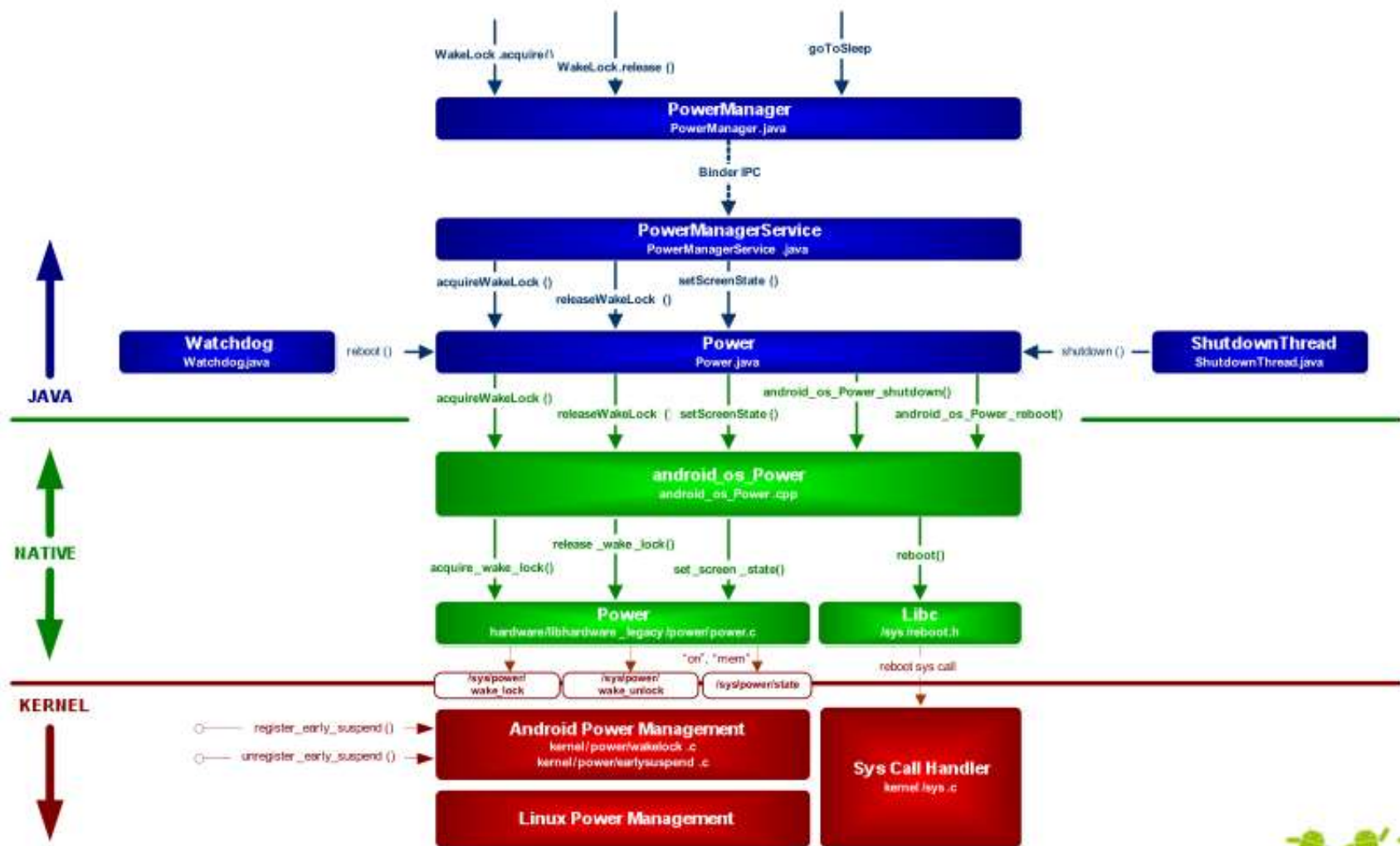




# Resume Late (2)



# Summary on PM





# Battery Service (1)

---

- ❑ 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<sup>2</sup>C ( 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 (2)

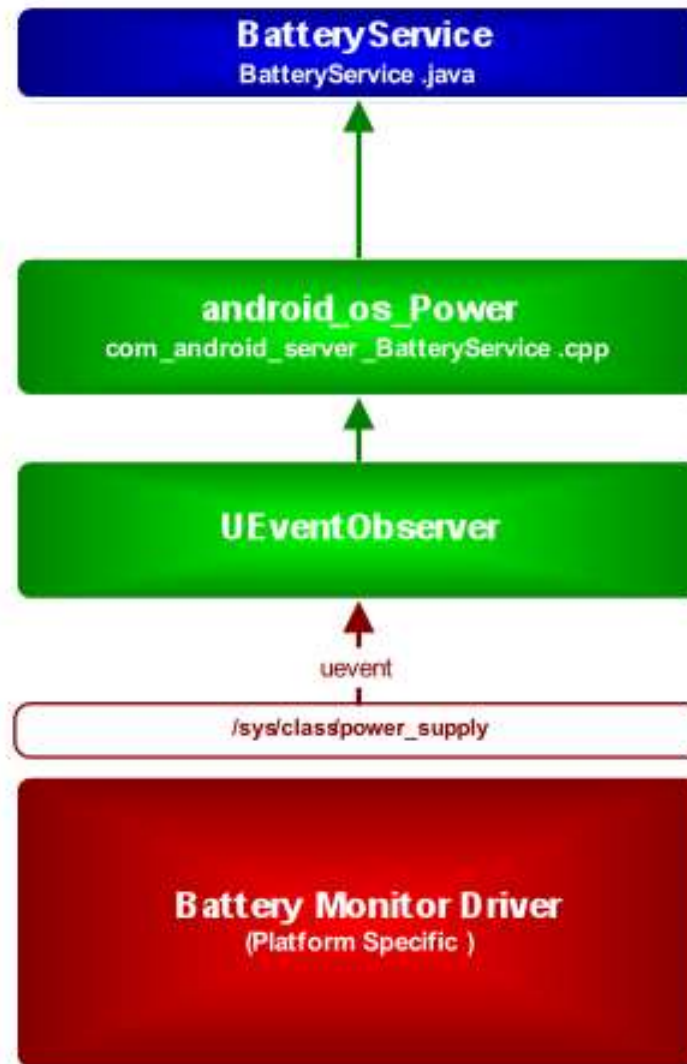
---

- ❑ 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





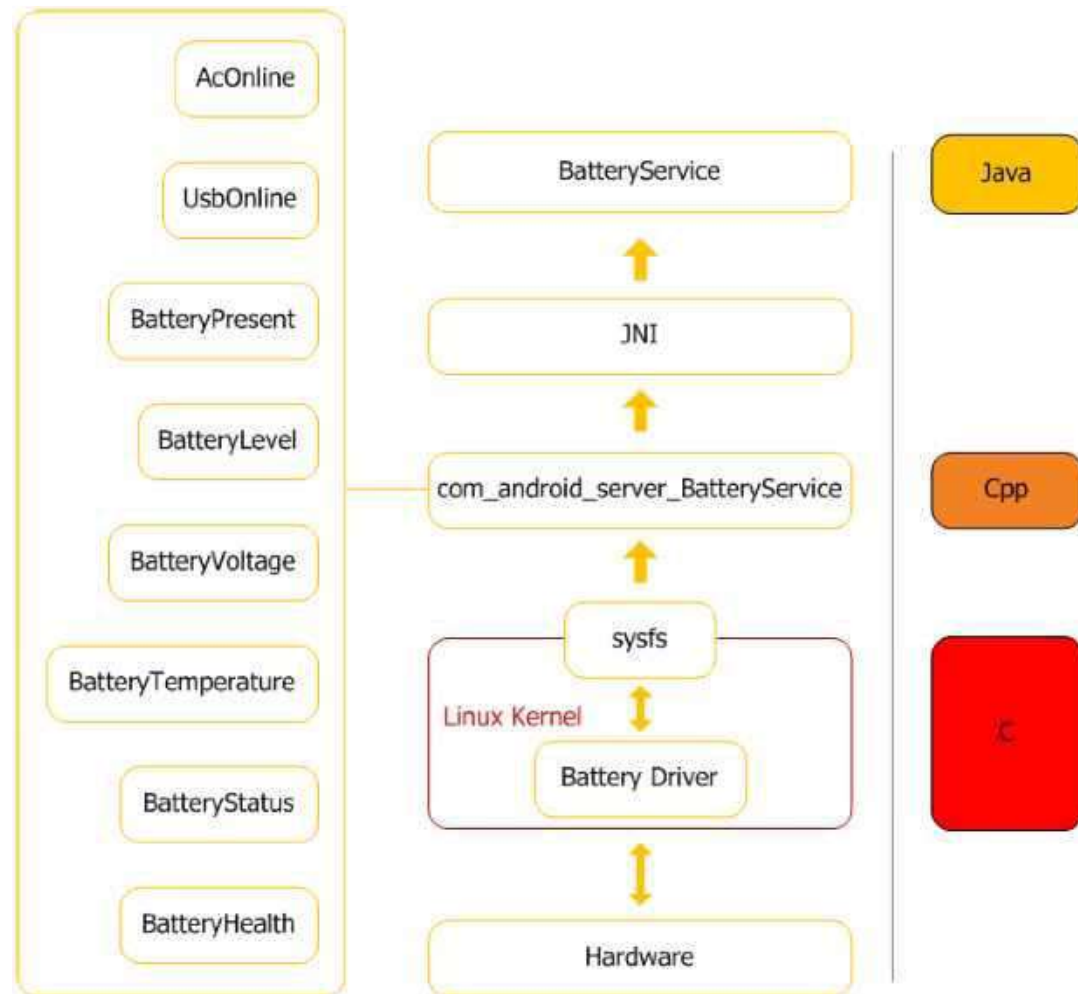
# Battery Service (3)







# Battery Service (4)





# Questions?

---

