G54MDP
Mobile Device Programming

Power and Batteries


APPLE 821-1033-A
Authorized Service Provider Only
WARNING
Potential for fire or burning. Do not disassemble, puncture, crush, heat, or burn.
Li-ion Polymer Battery
3.7V 5.25Whr
APN: 616-0513
VPN: LIS1445APPC
Assembled in China
CE
Li-ion

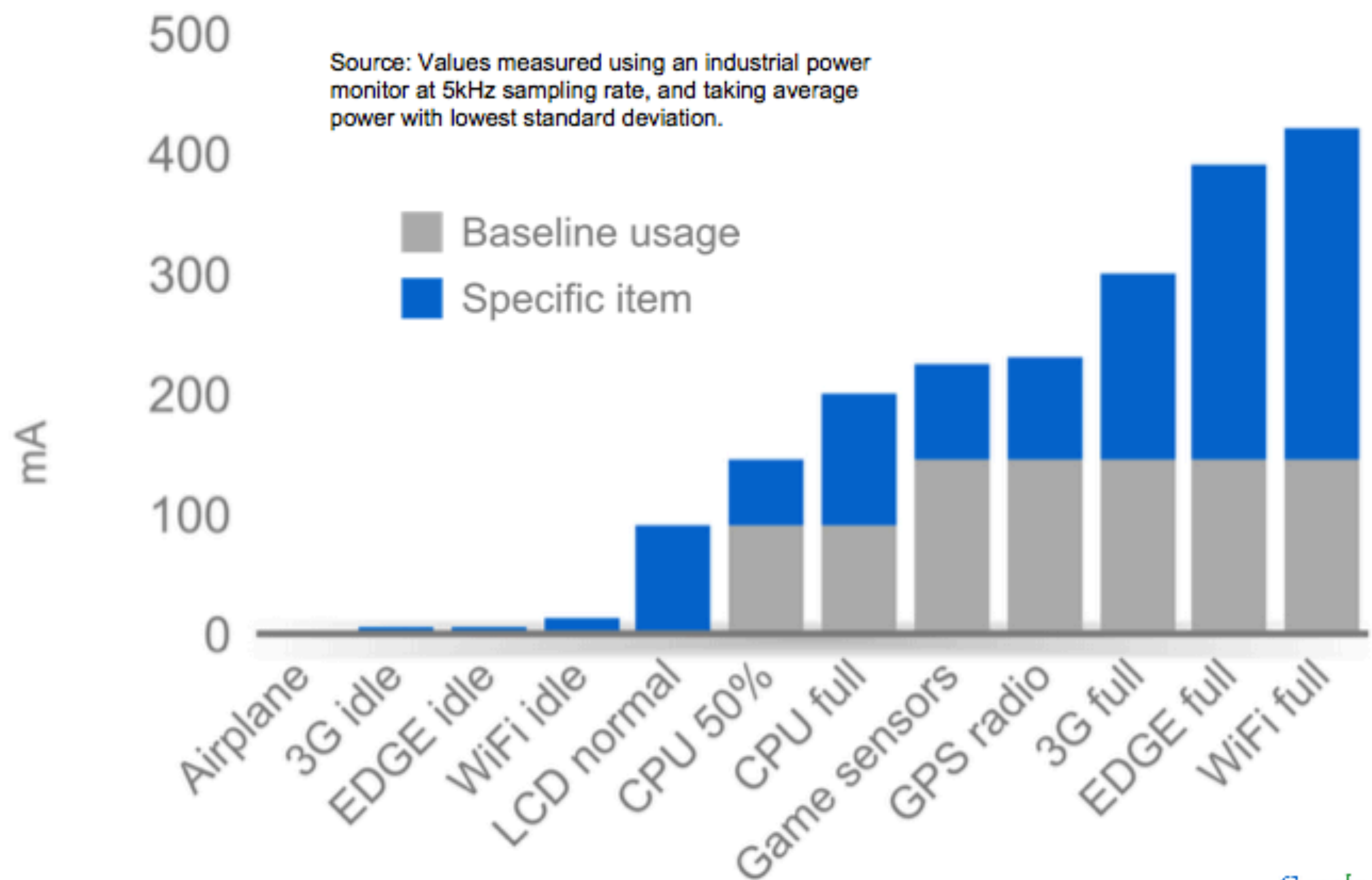
Batteries

- Mobile devices get their power from a battery
- More sophisticated devices require more power
  - Larger screens
  - Faster CPUs
  - Faster network communications
- … however battery technology evolving relatively slowly

Batteries

- Batteries have a limited power capacity
  - Power is the rate at which energy is used
  - Capacity measured in milliamp hours (mAh)
  - The amount of current that the battery can provide for one hour, before running out of charge
    - More "powerful" components draw more current
- 1000mAh battery can provide 1000mA (or 1A) for one hour
  - iPhone 4 has a 1420mAh battery
  - Laptop may have a 5800mAh battery
    - But more powerful components

Where does it all go?



Example Battery Usage

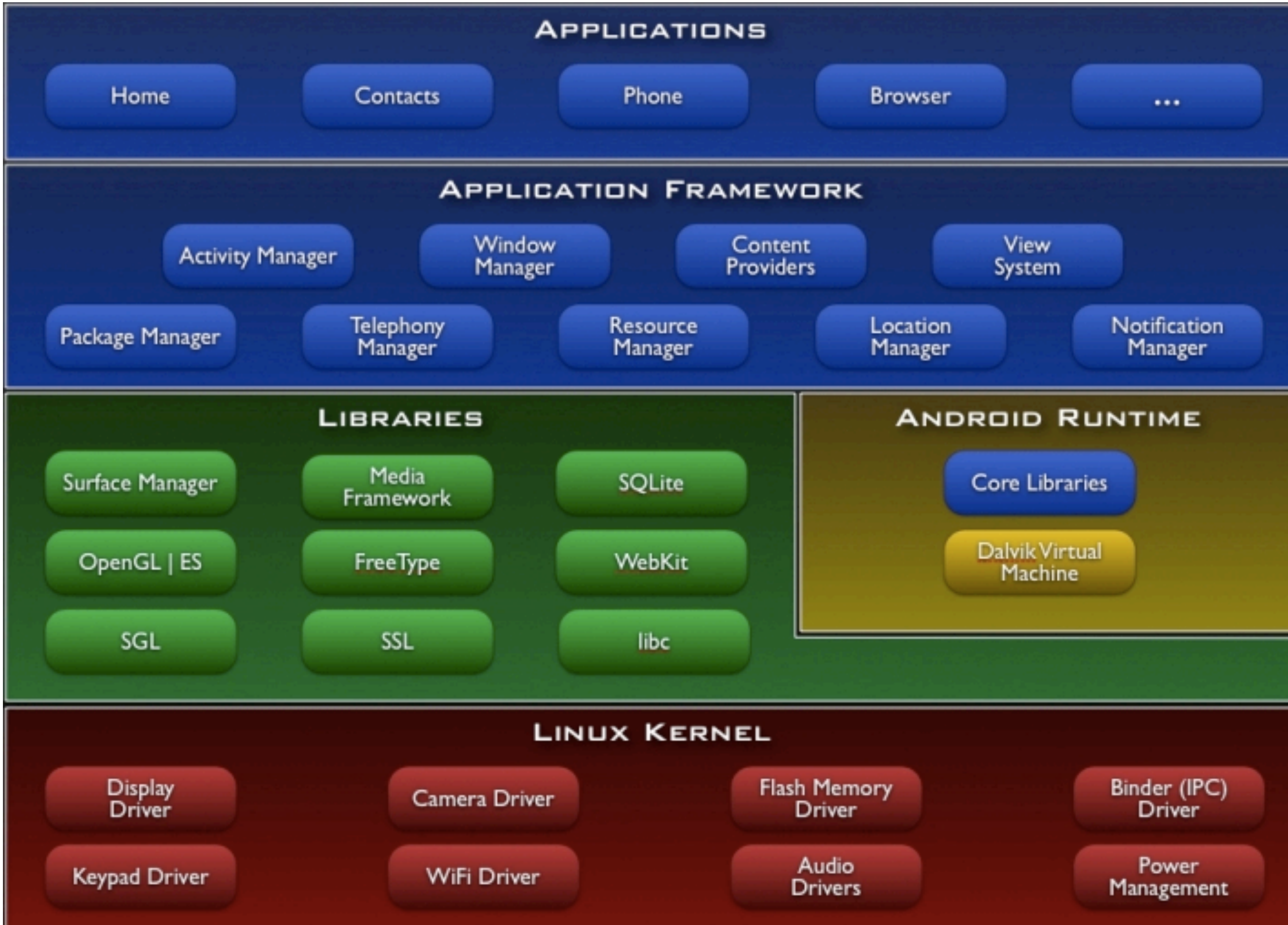
- Watching YouTube: 340mA = 3.4 hours
- Browsing 3G web: 225mA = 5 hours

- Typical usage: 42mA average = 32 hours
- EDGE completely idle: 5mA = 9.5 days
- Airplane mode idle: 2mA = 24 days

- What is “typical” usage?

Android Power Management

- Designed specifically for mobile devices
  - Goal is to maximise battery life
    - How?
- Build on top of Linux Power Management
  - Not directly suitable for a mobile device
- Designed for devices that have a **default off** behaviour
  - The phone is not supposed to be on when not in use
    - Think about how often the phone is in a pocket / bag / etc
  - Powered on only when requested to be run
    - Off by default
  - Unlike a PC
    - **Default on** behaviour



APPLICATIONS
Home
Contacts
Phone
Browser
...
APPLICATION FRAMEWORK
Activity Manager
Window Manager
Content Providers
View System
Package Manager
Telephony Manager
Resource Manager
Location Manager
Notification Manager
LIBRARIES
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
libc
ANDROID RUNTIME
Core Libraries
Dalvik Virtual Machine
LINUX KERNEL
Display Driver
Camera Driver
Flash Memory Driver
Binder (IPC) Driver
Keypad Driver
WiFi Driver
Audio Drivers
Power Management

Linux Power Management

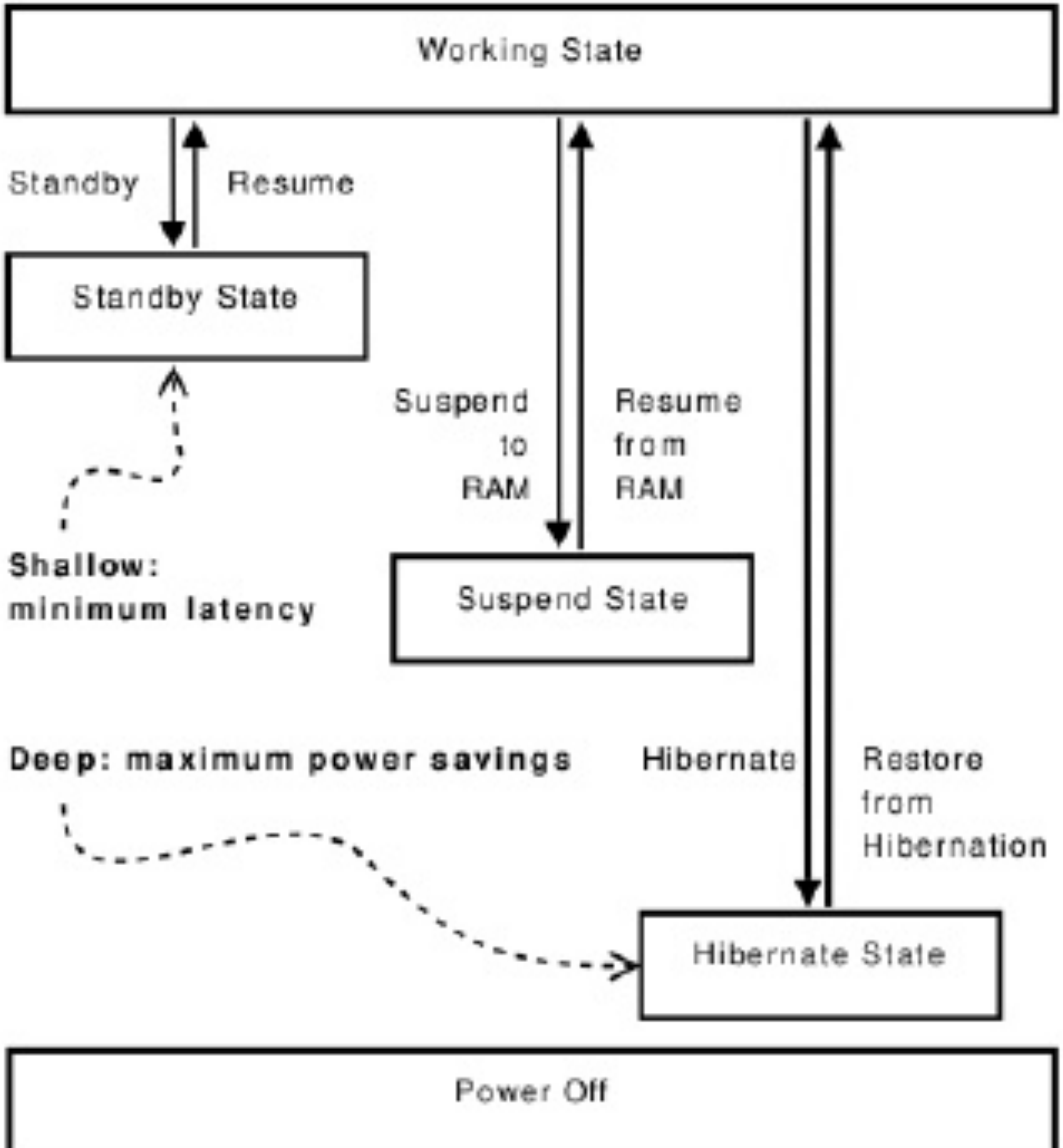
- APM - Advanced Power Management (1992)
- Power control resides in the PC BIOS
- Uses timeouts to determine when to power down a device
  - Monitor, HDD etc
- Makes power management decisions without informing the OS / individual applications

Linux Power Management

- ACPI – Advanced Configuration and Power Interface (1996)
  - Successor to APM
- Control divided between BIOS and OS
  - Decisions managed by the OS
- Enables power policies for general purpose computers with standard usage patterns and hardware
- No knowledge of device specific scenarios
  - Predictable response times
  - Respond to critical events over an extended period

Linux Power Management

- ACPI States
- G0 (working)
- G1 (sleeping)
  - S1 (CPU stops executing instructions, power to CPU and RAM maintained)
  - S2 (CPU powered off, cache is flushed)
  - S3 (Standby / sleep / suspend to powered RAM)
  - S4 (Hibernate / suspend to disk, RAM powered off)
- G2 (S5, soft off)
- G3 (mechanical off)


Working State
Standby
Resume
Standby State
Suspend to RAM
Resume from RAM
Shallow: minimum latency
Suspend State
Deep: maximum power savings
Hibernate
Restore from Hibernation
Hibernate State
Power Off

Android Power Management

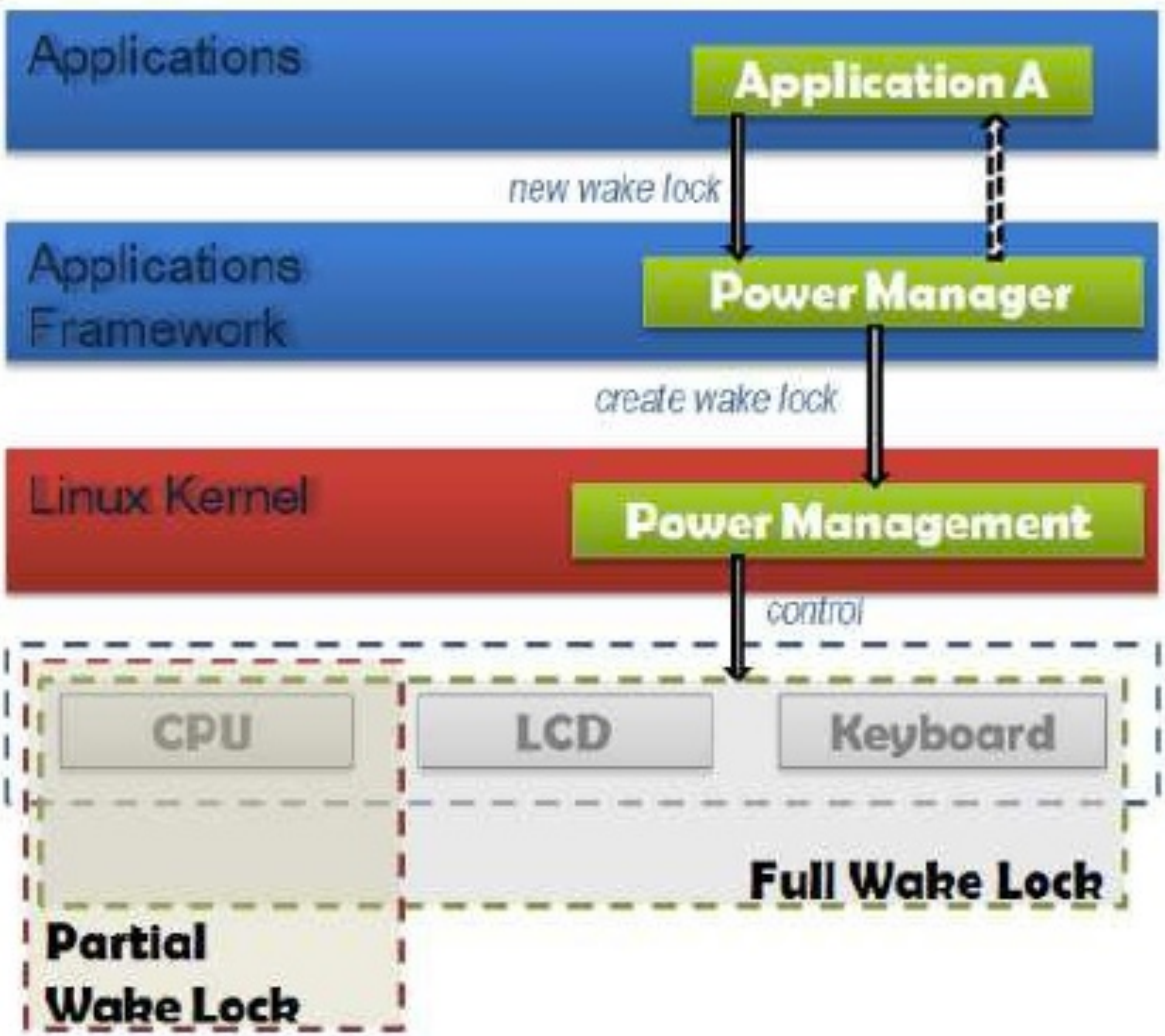
- Built as a wrapper around Linux Power Management
- In the kernel
  - Added **Early Suspend** mechanism
  - Added **Partial Wake Lock** mechanism
- Apps and services must request CPU resource in order to keep power on
  - Otherwise Android will shut down the CPU
  - Suspend operational RAM to NAND
- Wake locks and timeouts constantly switch the state of the system's power
  - Overall system power consumption decreases
  - "Better" use of battery capacity

Wake Locks

- By default Android tries to put the system into suspend mode as soon as possible
  - After a period of no activity / interaction
- Running apps can prevent the system from suspending
  - The screen stays on
  - The CPU stays awake to react quickly to interactions
- Applications ask for **wake locks**
  - If there are no **wake locks**, CPU will be turned off
  - If there are **partial wake locks**, display and touch screen will be turned off

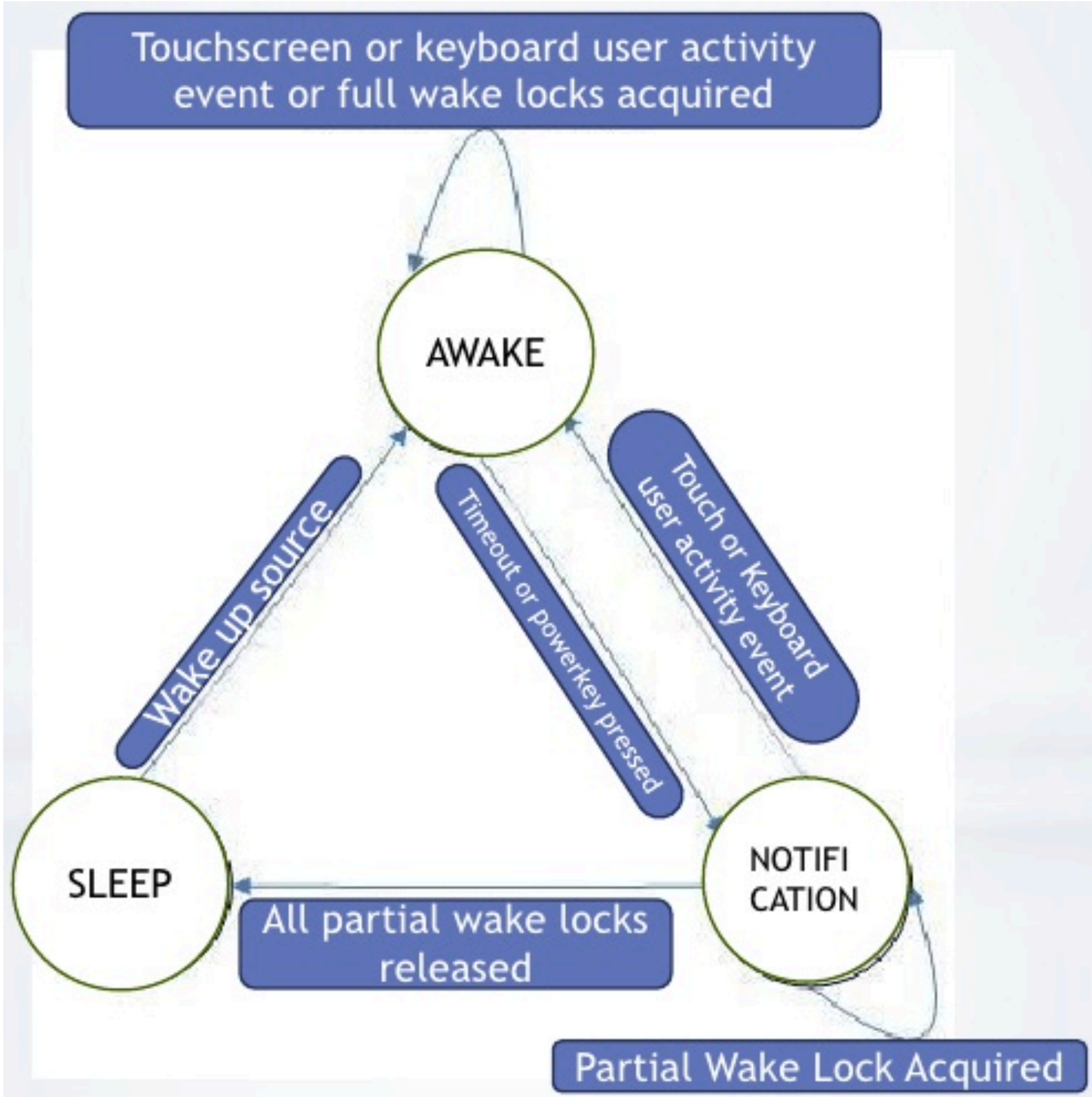
Wake Locks

- Types of Wake Lock
- PARTIAL_WAKE_LOCK
  - Ensures the the CPU is running
  - The screen might not be on (off after timeout)
- SCREEN_DIM_WAKE_LOCK
  - Ensures that the screen is on
  - Backlight will be allowed to go off (after timeout)
- SCREEN_BRIGHT_WAKE_LOCK
  - Screen is on at full brightness
  - Keyboard backlight will be allowed to go off
- FULL_WAKE_LOCK
  - Full device on, including backlight and screen


Applications
Application A
new wake lock
Applications Framework
Power Manager
create wake lock
Linux Kernel
Power Management
control
CPU
LCD
Keyboard
Full Wake Lock
Partial Wake Lock

Suspended Android

- Running applications / services are suspended
- CPU is powered down
  - Phone is not off
- Other components (SOC) continue to operate
  - CPU is periodically woken to handle scheduled tasks
    - Real time clock manifests as /dev/alarm
    - AlarmManager – Alarms, email polling…
  - GSM modem will wake CPU on call / SMS notifications
- Why use a PARTIAL_WAKE_LOCK?
  - Playing music – does not require screen to be on
  - Avoid suspension during period tasks
    - Android will try to suspend even when it is checking whether the alarm clock should sound
    - AlarmManager acquires, then releases a PARTIAL_WAKE_LOCK

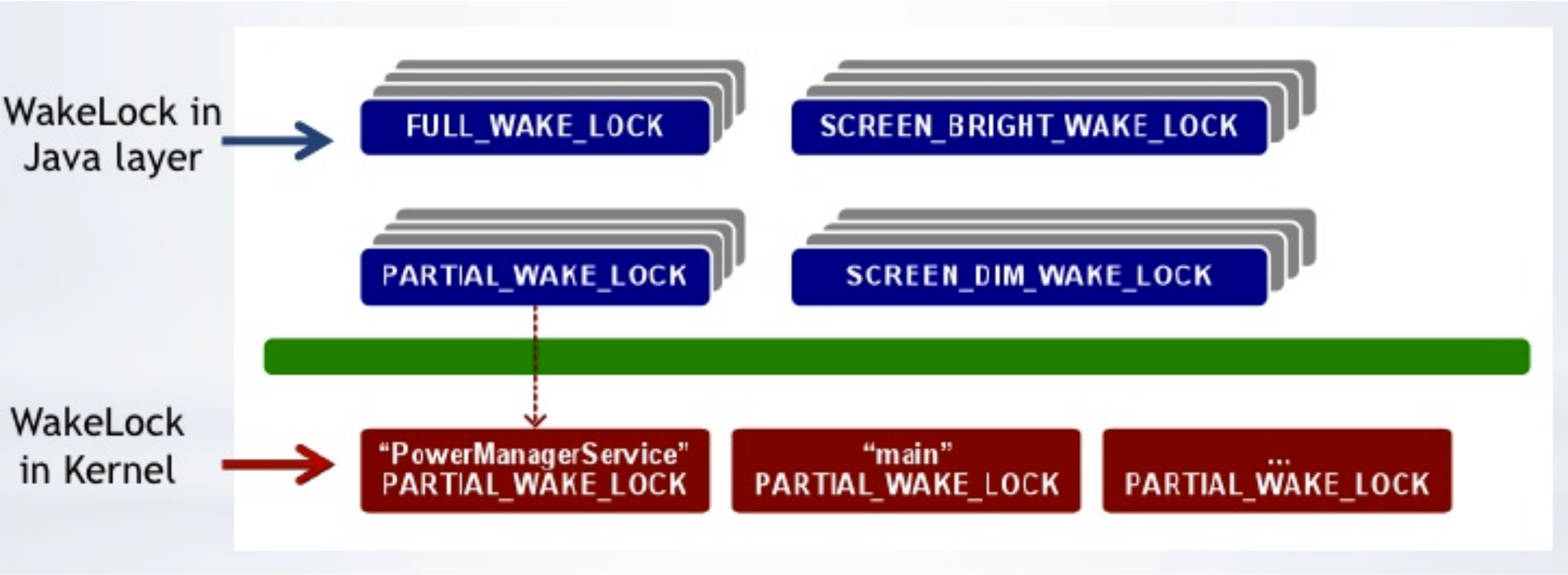

Touchscreen or keyboard user activity event or full wake locks acquired
AWAKE
Wake up source
Timeout or powerkey pressed
Touch or Keyboard user activity event
SLEEP
NOTIFI CATION
All partial wake locks released
Partial Wake Lock Acquired

Application Wake Locks

- Provides user-space (application) ability to manage power indirectly
  – Request a wake lock
- Application flow
  – Acquire a handle to the static PowerManager service with Context.getSystemService()
  – Create a wake lock and specify flags for screen, backlight etc
  – Acquire the wake lock
  – Perform the operation
    - Play MP3
  – Release the wake lock
- Must be used carefully
  – Keeping a wake lock for a long period of time will trash battery life
  – The CPU will not be allowed to sleep
- Tasks scheduled using the AlarmManager do not require a wake lock
  – AlarmManager acquires the lock while calling our scheduled task

Kernel Wake Locks

- Used to prevent the system entering suspended mode
  - Can be acquired and released by native code, or directly from within the kernel
  - Partial Wake Locks all reside in the kernel as they keep the CPU processing
- A single kernel wake lock manages multiple user mode (java) wake locks
  - PowerManagerService native kernel code partial wake lock
  - Audio driver partial wake lock while playing audio
  - Kernel has one last partial wake lock that exists to keep the kernel alive while other wake locks exist


WakeLock in Java layer
FULL_WAKE_LOCK
SCREEN_BRIGHT_WAKE_LOCK
PARTIAL_WAKE_LOCK
SCREEN_DIM_WAKE_LOCK
WakeLock in Kernel
"PowerManagerService" PARTIAL_WAKE_LOCK
"main" PARTIAL_WAKE_LOCK
... PARTIAL_WAKE_LOCK

Acquiring a Wake Lock

- Request sent to PowerManager (java) to acquire a wake lock
- PowerManagerService notified to take a wake lock
  - Add wake lock to an internal list
  - Set the requested power state
  - If this is the first partial wake lock take a kernel partial wake lock
    - This will protect all the partial wake locks
  - For subsequent wake locks simply add to the list

Releasing a Wake Lock

- Request sent to PowerManager (java) to release the wake lock
- Wake lock removed from the internal list
- If the wake lock is the last partial wake lock in the list
  - Release the kernel wake lock
- If kernel main wake lock is the only wake lock
  - Release main kernel wake lock
  - Device moves to suspend

Early Suspend / Late Resume

- More modifications to the Linux kernel
- In standard Linux all modules are suspended / resumed at the same time
  - Suspend
    - Freeze all user processes and kernel tasks
    - Call the suspend function for all devices
    - Suspend the kernel and suspend the CPU
  - Resume
    - Wake up the kernel
    - Wake up the registered devices
    - Unfreeze user processes and resume kernel tasks

Early Suspend / Late Resume

- Suspend as much as possible even if the kernel is still operating
- Early suspend
  - **Between** screen-off and full suspension
  - Tells devices to attempt to suspend even though a wake lock may be keeping the kernel awake
    - Stop screen, touch screen, backlight, close drivers
    - **Note** difference between "screen is on" and "kernel screen device is awake"!
- Cannot achieve full suspension (stop CPU, RAM -> NAND) until all wake locks are released
  - However attempts to suspend as much as possible
- Late resume
  - Kernel devices that were early_suspended are subsequently late_resumed
  - Can wake the kernel without waking up the entire device
  - Resume suspended devices once the kernel is awake and working

System Sleep

- API to put the device to sleep when the power button is pressed
- Requires DEVICE_POWER permission
- goToSleep()
  - Force release all wake locks
  - Turns off screen
  - The kernel immediately attempts to suspend

Summary

- Android Power Management
  - Why?
- Wake locks
- Early suspend, late resume
- System sleep

References

- http://developer.android.com/reference/android/os/PowerManager.html
- http://os.ibds.kit.edu/downloads/sa_2010_braehler-stefan_android-architecture.pdf