G54MDP
Mobile Device Programming

Lecture 17 –Gestures, Power

Single Touch

- So a touch gesture will be formed by
  - A single ACTION_DOWN
  - Zero or more ACTION_MOVE
  - An ACTION_UP to finish
- Can use the data from these to perform some interaction
  - Use position delta to move an object around the screen
    - Delta = change from original
  - Use movement velocity for a swipe / fling

Dragging / Scrolling

- Store original location of thing to move
  - Store x,y-pair from ACTION_DOWN
- Calculate delta from stored value and value returned from ACTION_MOVE or ACTION_UP
  - Change the location of thing being moved by adding delta to original location
  - Note: need to adjust as position returned is relative to View origin

Swipe / Fling

- Can do similar for a swipe / fling
- Rather than moving the object, calculate the velocity with which it is moving
  - Speed
  - Direction
- On ACTION_UP
  - Continue to move the object with that velocity
- Gives the user the impression of “flinging” UI elements across the screen
  - Obvious visual feedback

Multi Touch

- Very similar to single touch
- Same sequence of events as before with a few more events thrown in
  - ACTION_POINTER_DOWN and ACTION_POINTER_UP tell us that a **new** pointer has been pressed
- Support for 256, but some Android devices only support 2

Which finger?

- getActionIndex() tells us the index for the pointer caused this event for
  - ACTION_POINTER_DOWN/ACTION_POINTER_UP
  - However, number of pointers can change as fingers are lifted or placed
- Each pointer given an id that won't change
  - But its **index** within a bundle of movement events might
  - Need to track both the id and past locations of pointers to move things about

Which finger?

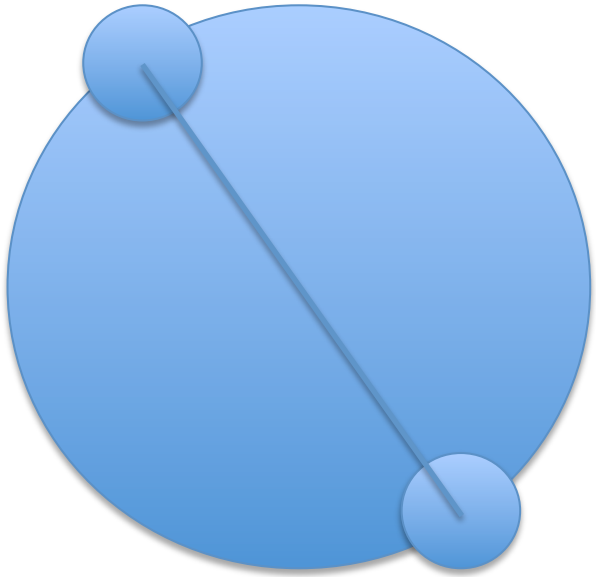
	Action	ID
#1 touch →	ACTION_DOWN	0
#2 touch →	ACTION_POINTER_DOWN	1
#3 touch →	ACTION_POINTER_DOWN	2
	ACTION_MOVE	0
#2 lift →	ACTION_POINTER_UP	1
#1 lift →	ACTION_POINTER_UP	0
#3 lift →	ACTION_UP	2

Pointers to Gestures

- Maintain state of pointer IDs
  - Track movement of multiple fingers
- How do we convert these into gestures?
  - Pinch to zoom
  - Two-finger rotation
- Little SDK support for specific gestures
  - Implement ourselves with some simple trigonometry

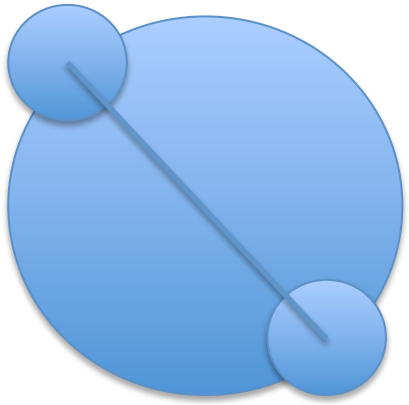
Pinch to Zoom

- Imagine the two points lie on the circumference of a circle
  - ACTION_POINTER_DOWN
- Can easily calculate the diameter of the circle
  - distance between the two points
- Use Pythagoras to calculate it



Pinch to Zoom

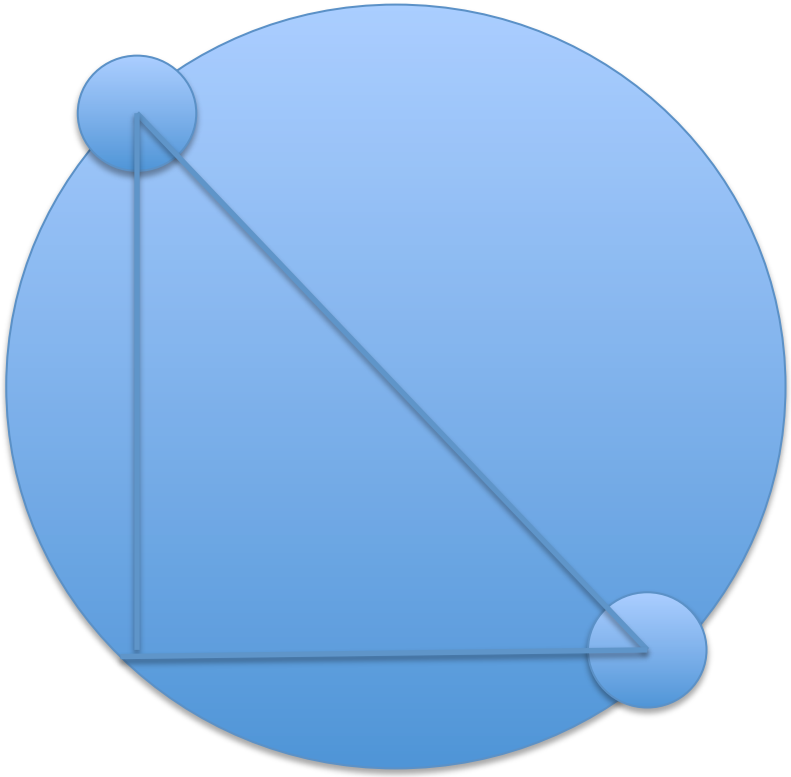
- As fingers move, diameter of the circle will change
  - store the initial diameter of the circle
- The ratio of new diameter to old diameter will give the zoom ratio
- With this and the original size of the item
  - calculate how to rescale the new item



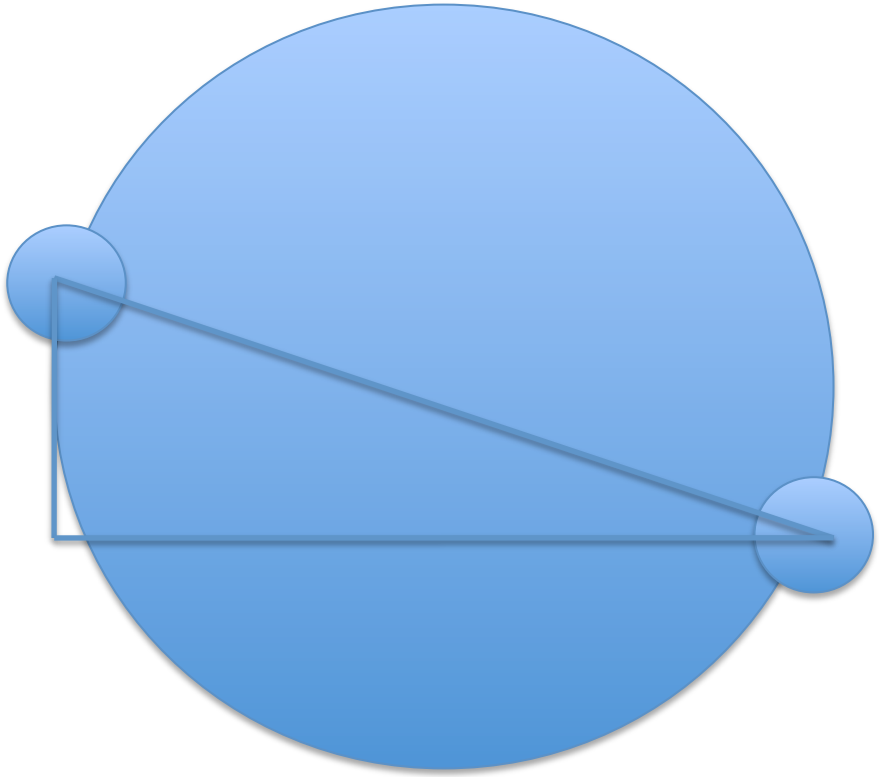
Two-finger rotation

- Very similar approach for rotation
- The difference in position can (using basic trigonometry) give us the angle of the line bisecting the circle between the two points
- As the points move the angle will change
- Can use the difference between this and the original angle to work out the rotation
  - Magnitude may not be as important as direction
    - Rotate a photograph, document

Two-finger rotation



Two-finger rotation



Android Support

- We could implement any number of complex gestures
- Android provides built-in callbacks for common gestures via GestureDetector.SimpleOnGestureListener class
  - onSingleTap
  - onDoubleTap
  - onShowPress
  - onLongPress
  - onScroll
  - onFling
- Can create and save custom gestures as binary resources
  - A pattern of movements
- GestureLibrary attempts to recognise the gesture
  - Detects attempt at complex gesture
  - Returns predications as to which gesture the input may match
  - Inspect confidence of a match
    - Gestures are an imprecise science

Let’s have a look…

References

- http://developer.android.com/training/gestures/index.html
- http://developer.android.com/training/gestures/multi.html


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
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

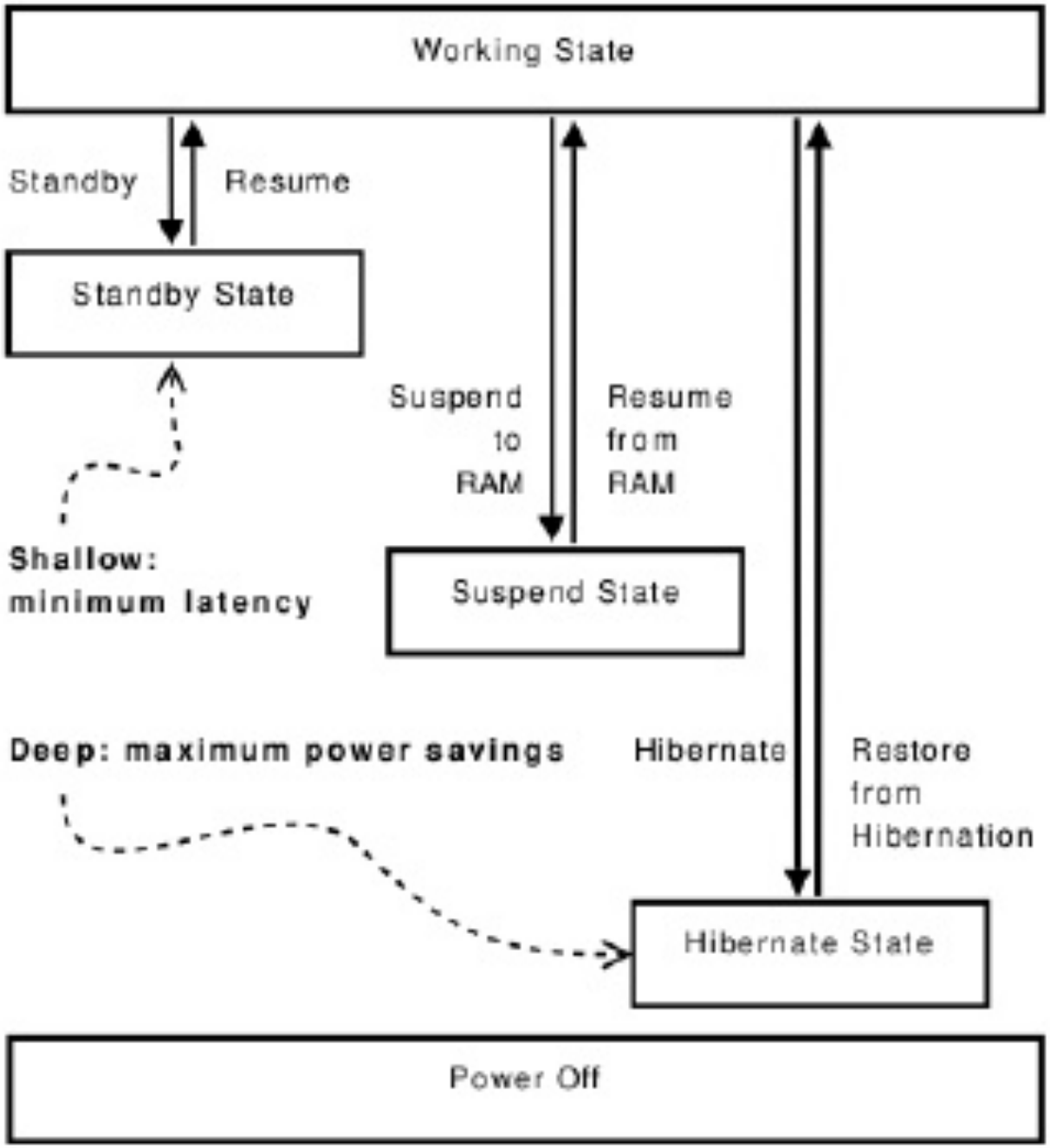
- Batteries have a limited power capacity
  - Power is the rate at which energy is used
  - Capacity measured in milliamp hours (mAh)
  - The amount of current that the battery can provide for one hour, before running out of charge
    - More “powerful” components draw more current
- 1000mAh battery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

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

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

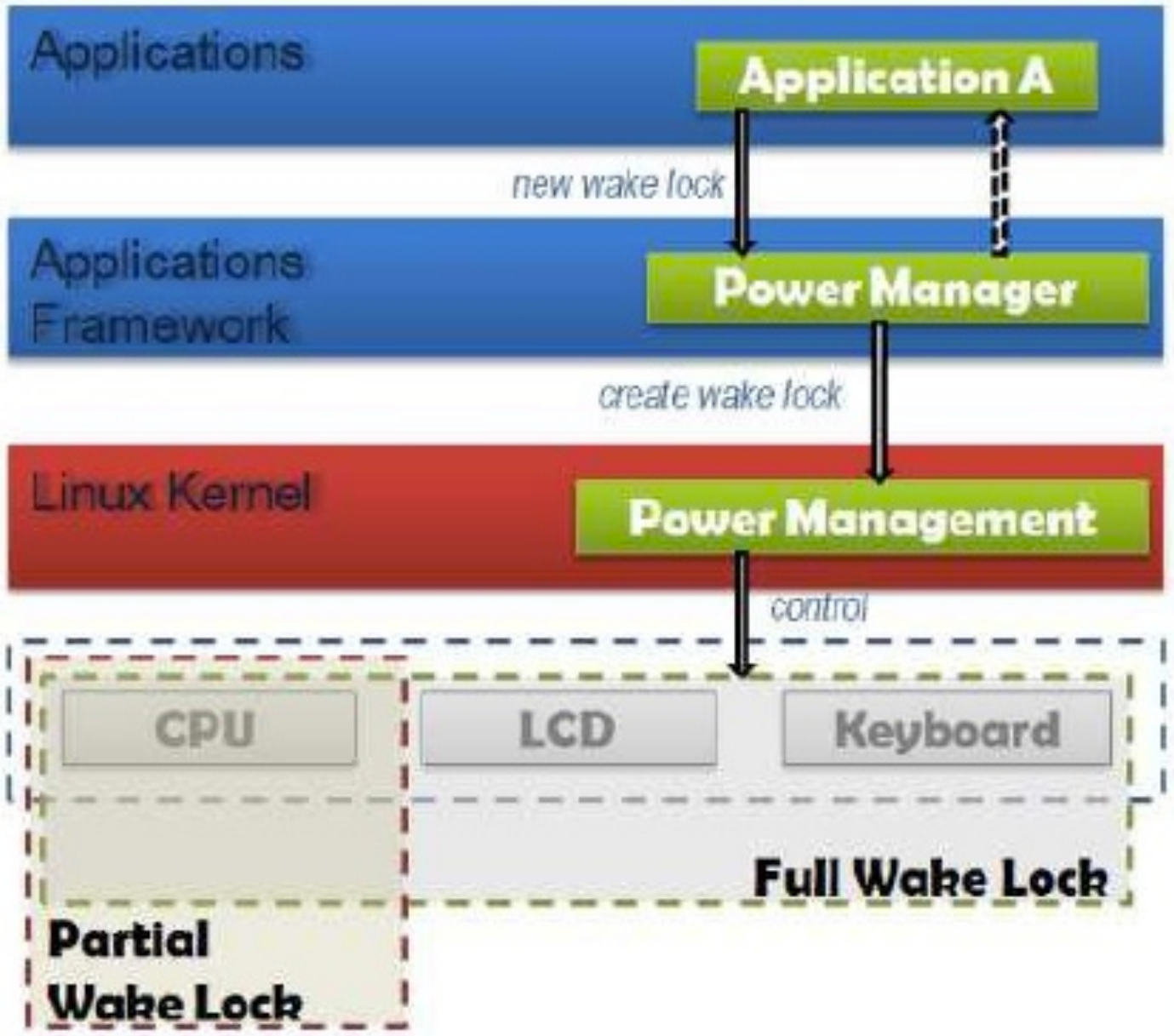
- Built as a wrapper around Linux Power Management
- In the kernel
  - Added **Early Suspend** mechanism
  - Added **Partial Wake Lock** mechanism
- Apps and services must request CPU resource in order to keep power on
  - Otherwise Android will shut down the CPU
  - Suspend operational RAM to NAND
- 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

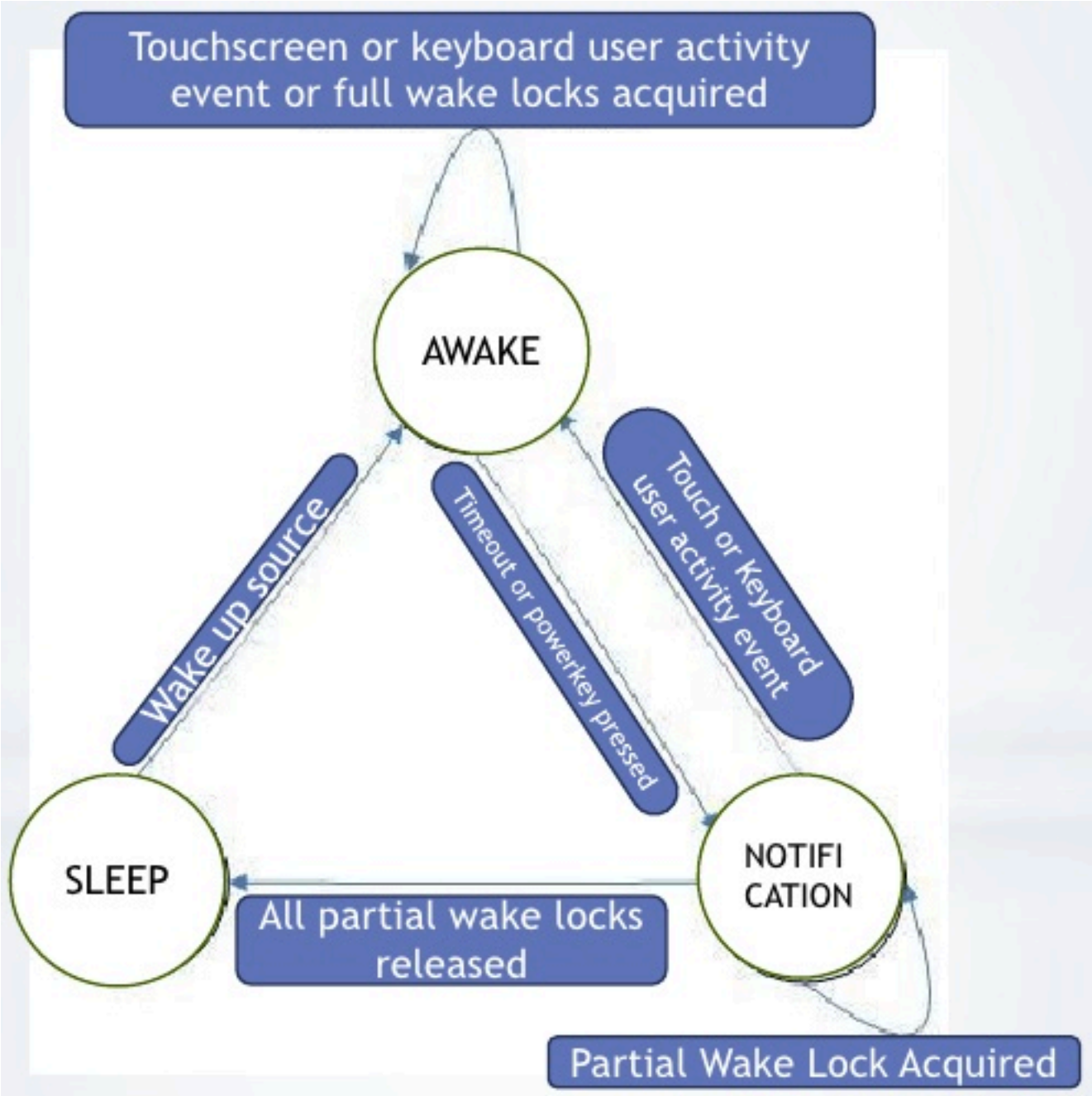
Wake Locks

- 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

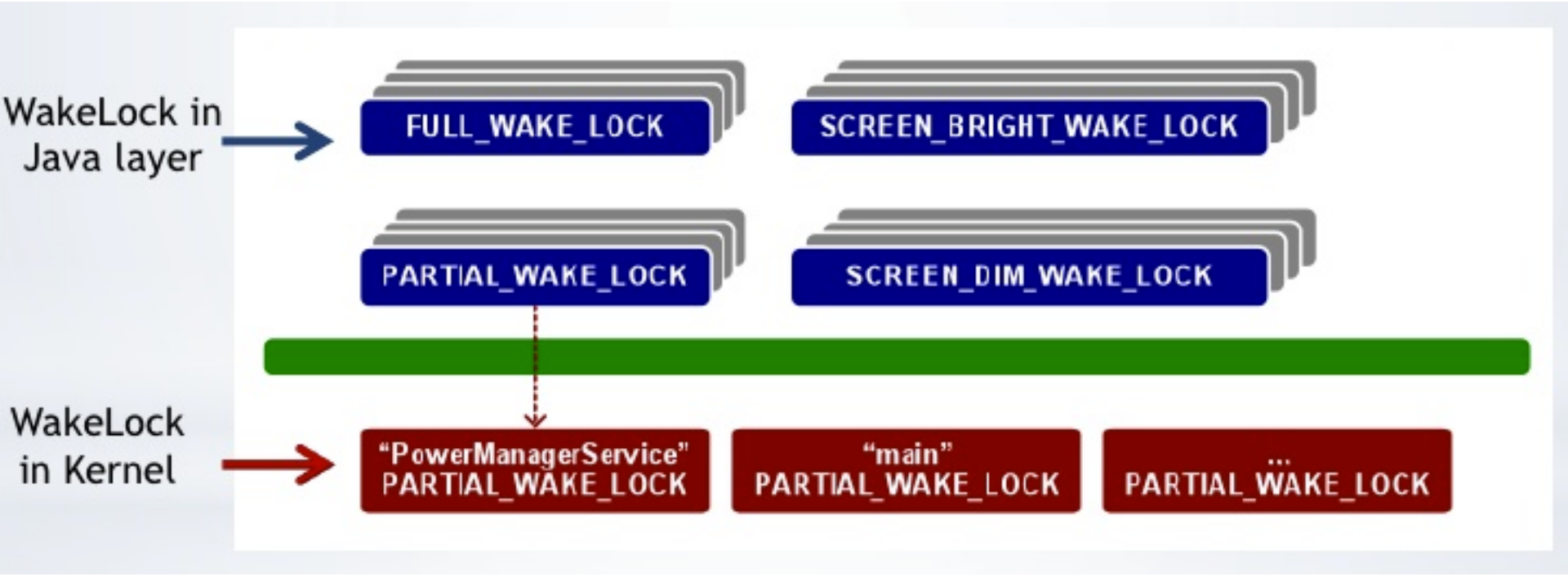

Touchscreen or keyboard user activity event or full wake locks acquired
AWAKE
Wake up source
Timeout or powerkey pressed
Touch or Keyboard user activity event
SLEEP
NOTIFI CATION
All partial wake locks released
Partial Wake Lock Acquired

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

AlarmManager

- Schedule an application to be run at some point in the future
  - As usual, specify an Intent to be broadcast at some time
    - At a regular interval, after an elapsed time
  - Extend Broadcast Receiver
    - onReceive method is called when the alarm goes off
- AlarmManager is triggered regularly by the real time clock device in the kernel
  - Alarms go off even when the CPU is asleep
  - AlarmManager holds a partial wake lock while onReceive is executing
    - If the alarm starts a service, need to acquire our own partial wake lock, as the system may go to sleep while the service is starting
    - Default off

References

- http://www.google.com/events/io/2009/sessions/CodingLifeBatteryLife.html
- http://developer.sonymobile.com/2010/08/23/android-tutorial-reducing-power-consumption-of-connected-apps/
- http://www.slideshare.net/EricssonLabs/droidcon-understanding-smartphone-traffic