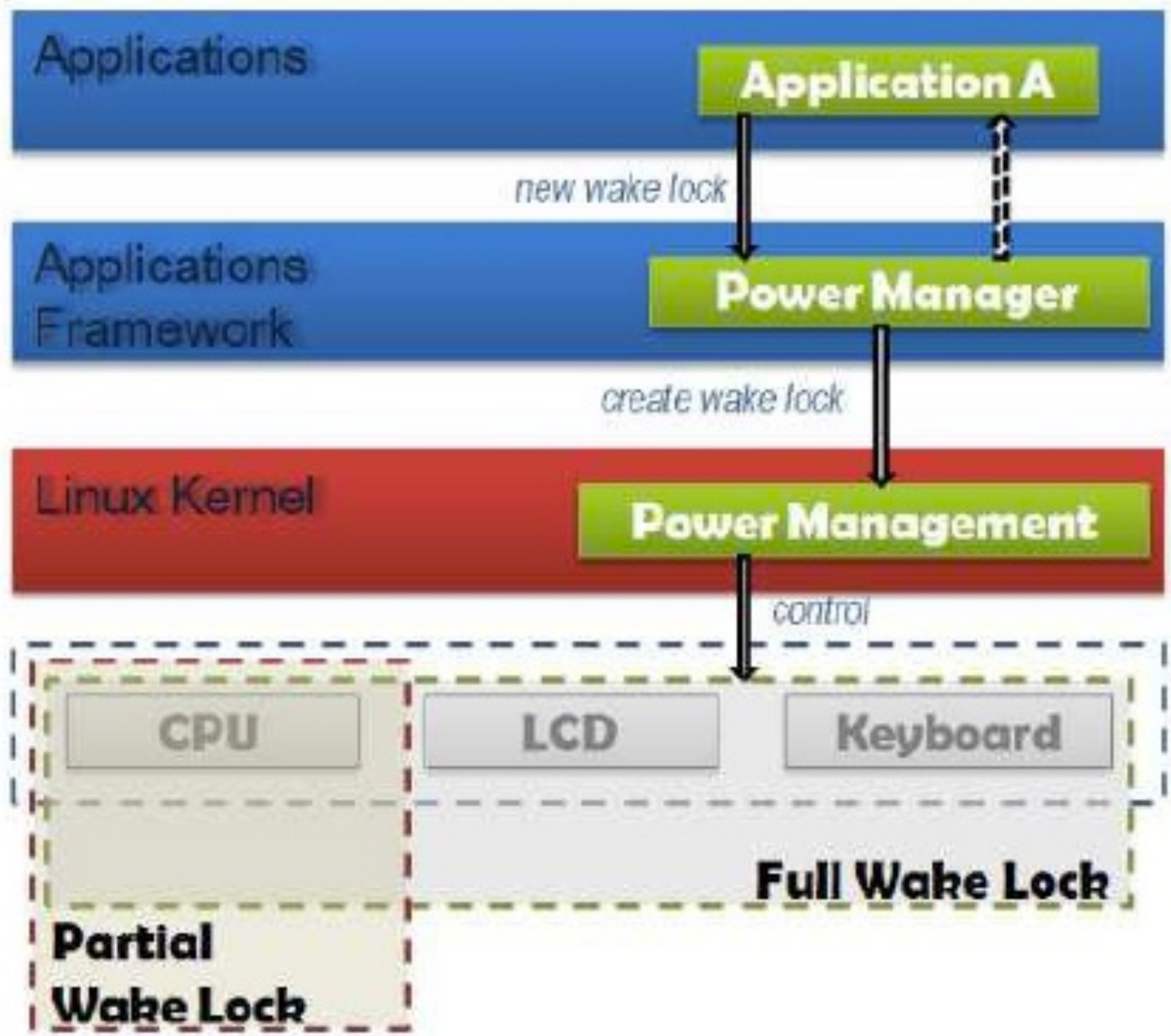
G54MDP
Mobile Device Programming

Lecture 18 – 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

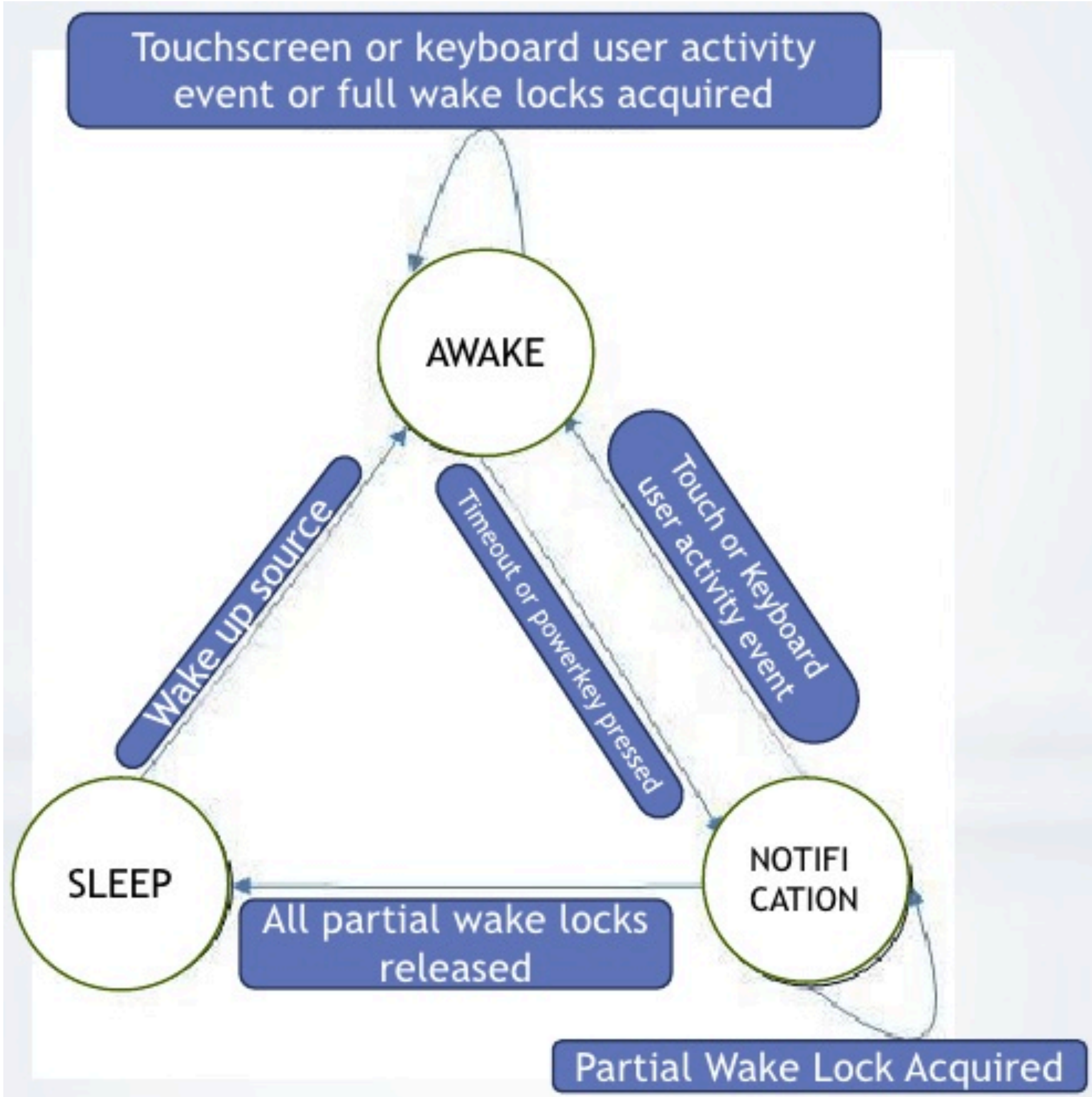
Android Power Management

- 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


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 periodic 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

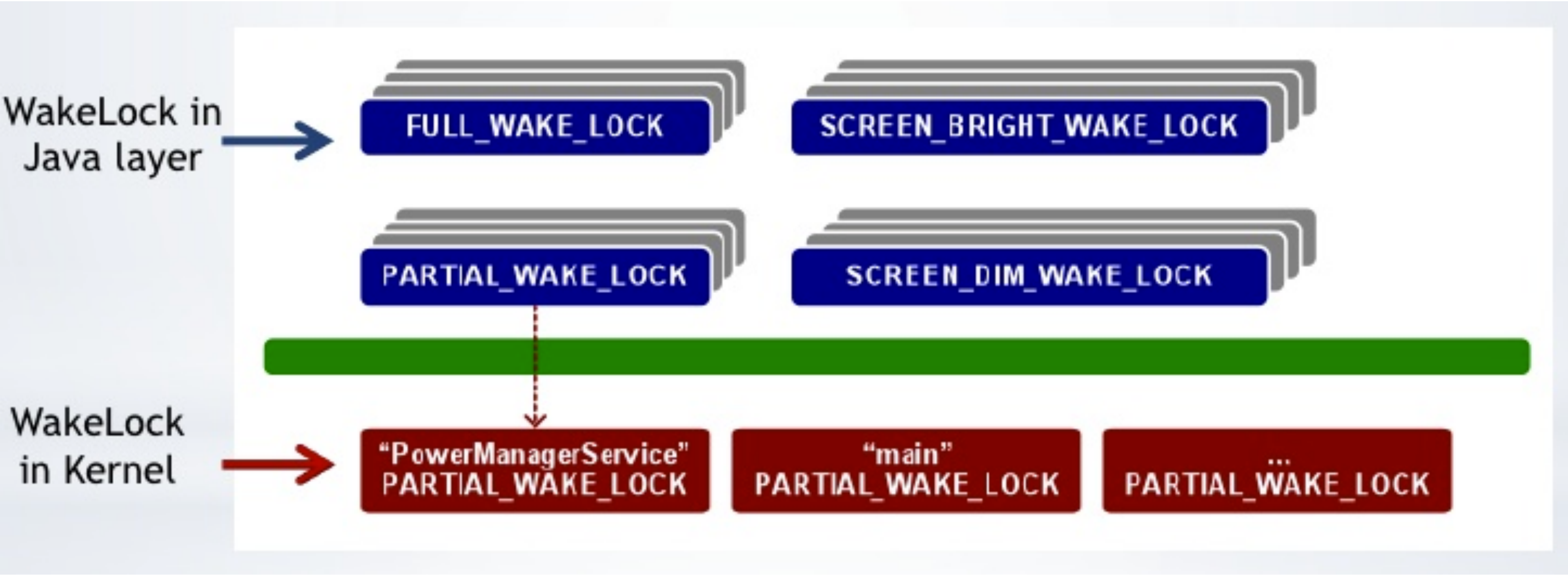

Touchscreen or keyboard user activity event or full wake locks acquired
AWAKE
Wake up source
Timeout or powerkey pressed
Touch or Keyboard user activity event
SLEEP
NOTIFI CATION
All partial wake locks released
Partial Wake Lock Acquired

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

Coding for Battery Life

- Android constantly tries to suspend
  – Extends battery life on average
- Apps allow us to **do things** with the phone
  – Use the CPU, use the radio / network
    - reduce battery life
- Code with battery usage in mind
  – Work with, not against the system
  – Pragmatism over principle
    - Efficiency over code elegance and good OO practice
    - Pretend that you're a C programmer

Waking up the phone

- Imagine an email application
  - Checks for new emails every 10 minutes
  - Takes 10 seconds to check for new emails
    - Wakes up CPU using an alarm
    - Wakes up the network device
    - Makes a DNS request
    - Pulls down and parses data
  - 350mA current used
- Cost during a given hour:
  - Sleeping 3600 seconds * 5mA = 5mAh
  - 6 times * 10 seconds * 350mA = 5.8mAh
- Double the battery usage
- Waking up the phone can cause a cascade of updates
  - Some services only run when the phone is awake

Using Alarms

- Multiple scheduled alarms waking the phone
  - Suspending and waking the phone takes power
  - More efficient to schedule multiple alarms at the same time
    - Wake once, do several tasks, sleep once
    - Wake, task, sleep, wake, task, sleep, wake, task, sleep…
- setInexactRepeating()
  - An alarm that repeats once an hour, but not necessarily on the hour
  - Time between two alarms may vary
    - Jitter – some may be acceptable
  - Allow the system to schedule multiple similar alarms at the same time

Rules of thumb

- Speed = Efficiency
  - The CPU runs at a certain rate
    - Instructions per second
  - The faster we can perform our work, the more time the CPU can idle
    - Idle at reduced power
    - More efficient use of instructions
  - The faster we can perform our work, the more quickly the CPU can go to sleep
    - Sleeping at reduced power consumption
- Waking up / Running services = Costs power
  - Assume we are not the only application in use
- Byproduct
  - A fast app feels more responsive
    - Users are less likely to use an app that is slow
    - Majority of apps are kept / uninstalled after first run

Memory Use

- Mobiles have limited RAM
  - Typically <512MB
  - iPhone3G used 102MB of 128MB for "system"
  - No virtual memory
  - Be frugal with memory usage
    - Memory allocation / garbage collection takes time, uses power
- Avoid allocation of memory, objects where possible
  - Yes we have a garbage collector
    - Takes time, uses power. Generate less garbage
  - A **String** is an object
    - Return **substring** rather than a new **String** – pointer to the same memory

Throwing away OO

- An array of ints vs an array of Integer objects
  - Which is more efficient in terms of allocation and memory usage?
- Avoid boxing
  - int x[1024], y[1024];
  - class Point { int x; int y; public getX() }
- More efficient (CPU, memory) to use the former
  - More care required when coding
- Getter / Setter methods
  - Good OO practice to provide these at the class interface
  - Access member variables directly where possible
    - 3x – 7x speed improvement, removing virtual method calls

Throwing away OO

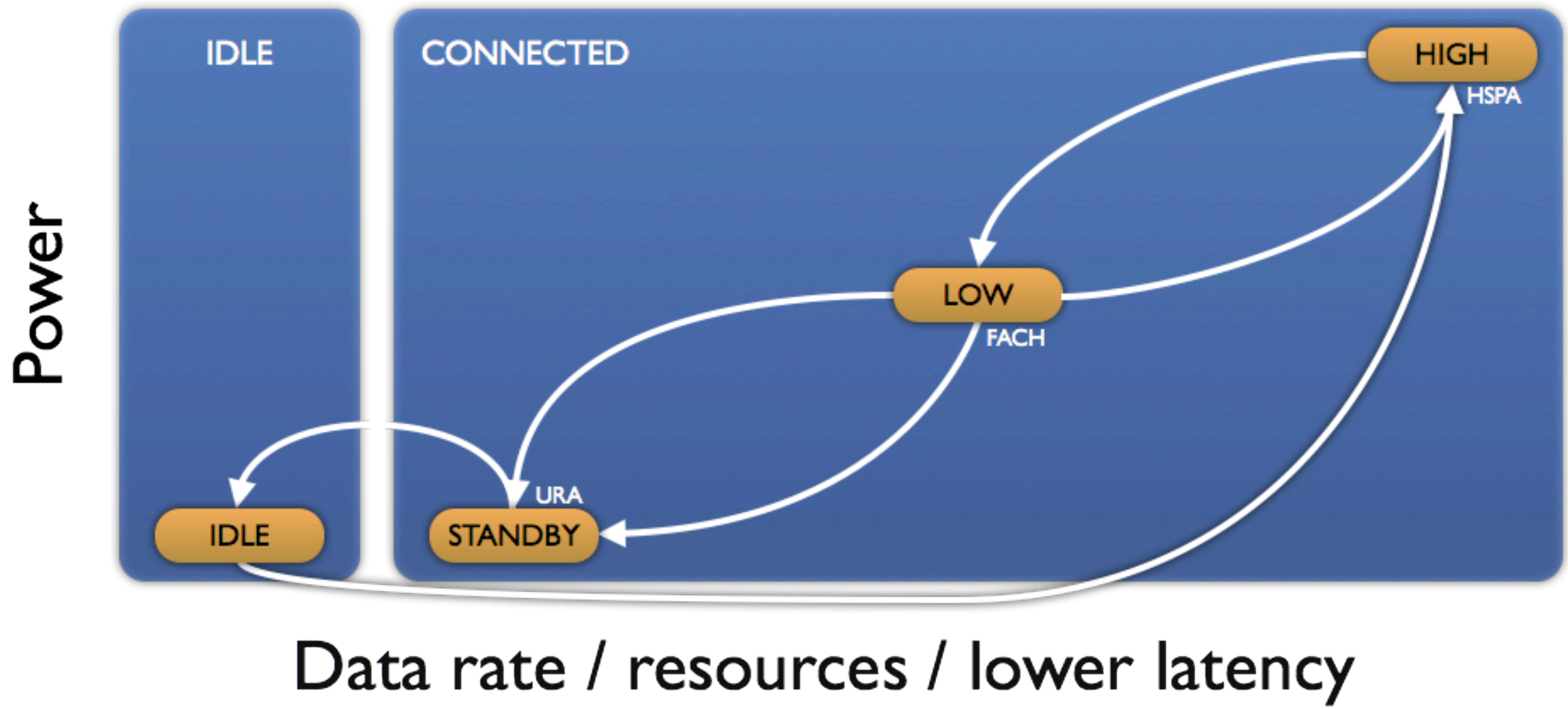
- Static methods belong to a class
  - public static void foo() { … }
- Virtual methods act on an object
  - public void foo() { … }
- When can we use static?
  - When no member variables are accessed
  - Pass variables as parameters instead
  - Makes method invocations ~15% - 20% faster
    - Discuss!
- Constants
  - Should be declared **static final**
  - Just **static** causes class initialiser method to be run
  - Make them **static final** the variable lookup vanishes
    - Replaced by a constant

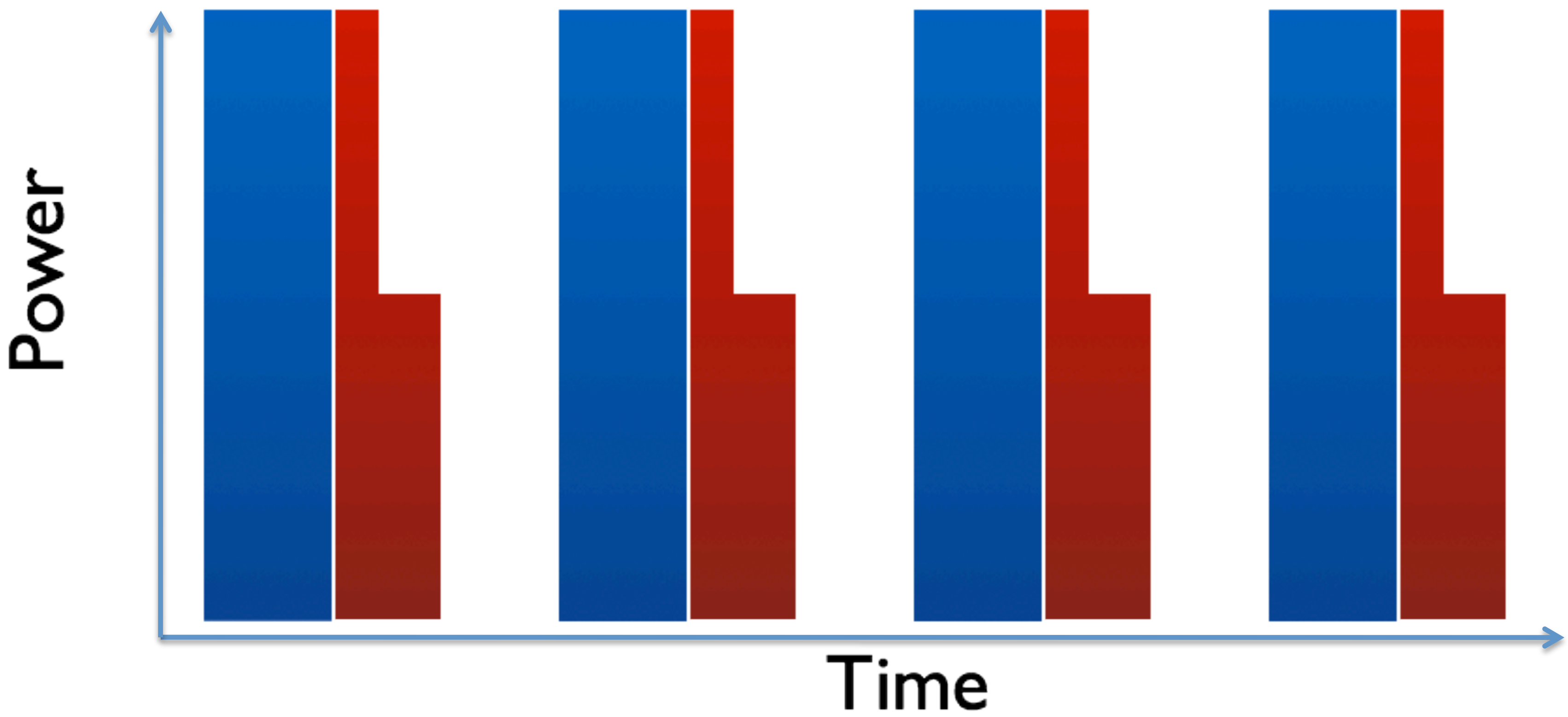
Other Chips

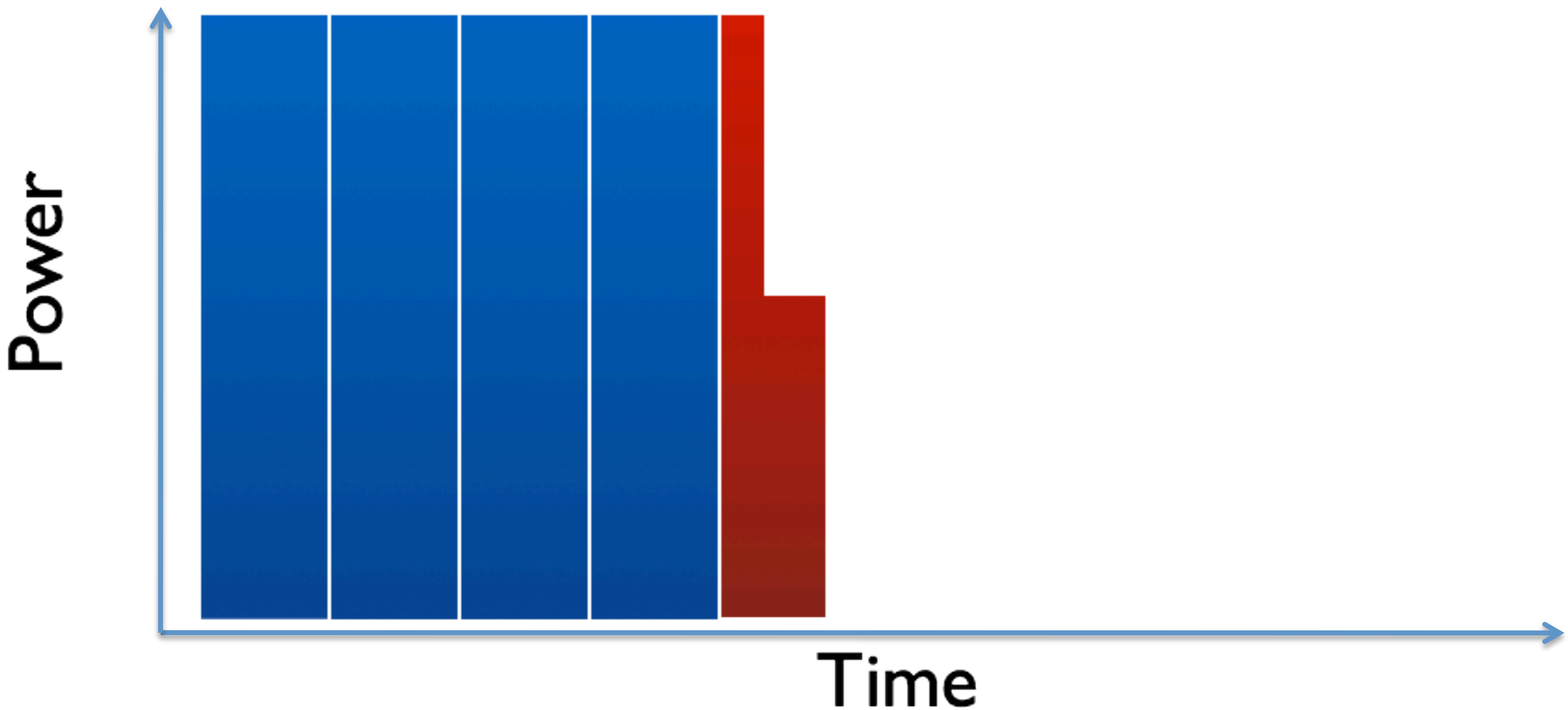
- CPU ~100mAh
- Other components use power too
  - Accelerometer
    - 10mA normal use – 80mA fastest / finest measurements
      - Choose most appropriate frequency
  - Location
    - Wifi basestations (~100m)
    - Cellphone tower triangulation (~500m – 3km)
    - GPS (~1-5m)
    - Select the most appropriate accuracy
      - GPS is very expensive in terms of battery usage, especially cold start
    - Register for updates appropriately
  - Radios (network connectivity, phone calls)

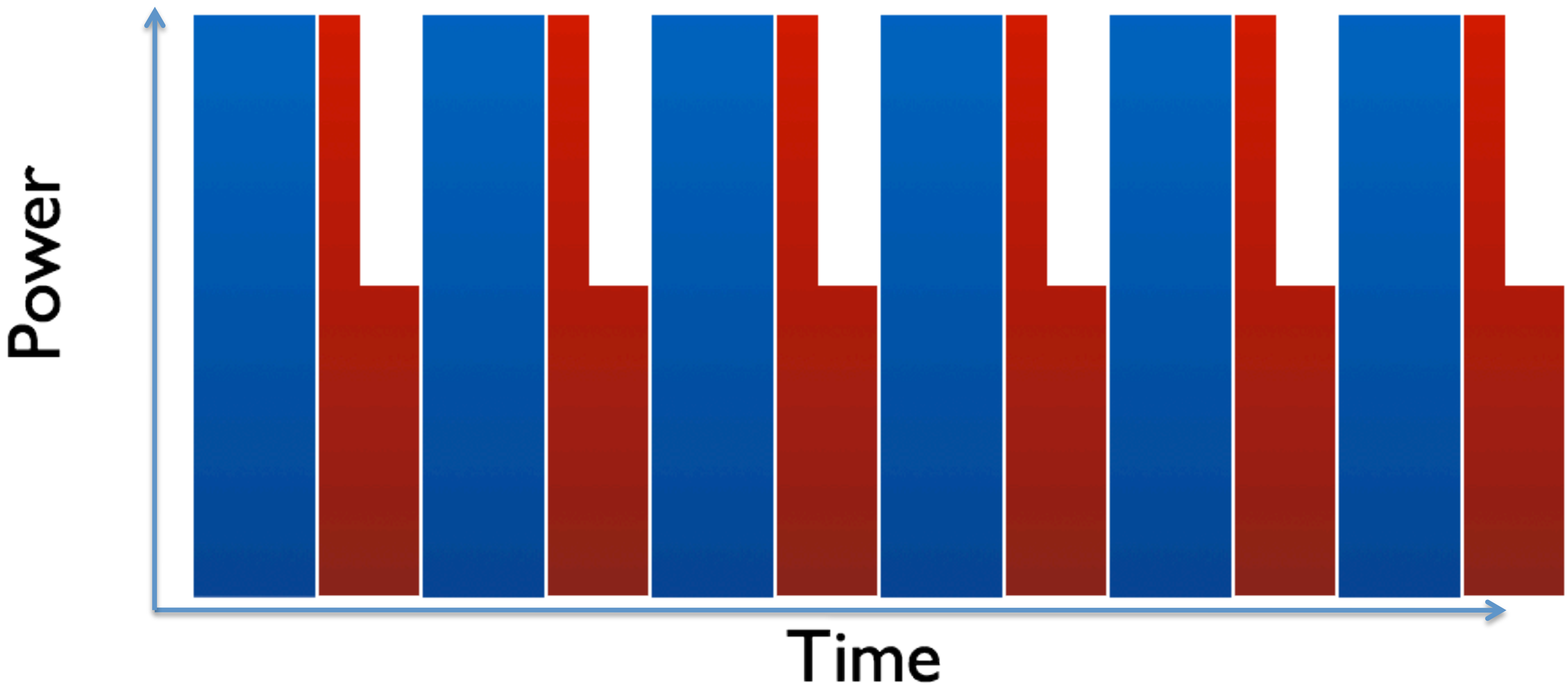
Radio / Network

- 3G chip has a number of states
  - URA – Connected but not sending data
  - FACH – Half power, small amount of data
  - HSPA – Full power, dedicated channel
- Cost / time to transition upwards
  - Ramp up power, negotiate channel
- In high power radio state
  - Delay to transmit is shorter
  - Device stays in high state for a short period of time following communication
- Regular polling keeps the radio transition between states
  - Pay the battery cost even if we transfer nothing
    - Synchronize polling – inExactAlarms
    - Coalesce data into large chunks
    - Small transfers will only transition up to low / FACH power state (~256 – 512 bytes)
  - Be careful of reusing libraries
    - Were they designed for 3G, or do they assume Ethernet


IDLE
CONNECTED
HIGH
HSPA
LOW
FACH
URA
IDLE
STANDBY
Power
Data rate / resources / lower latency

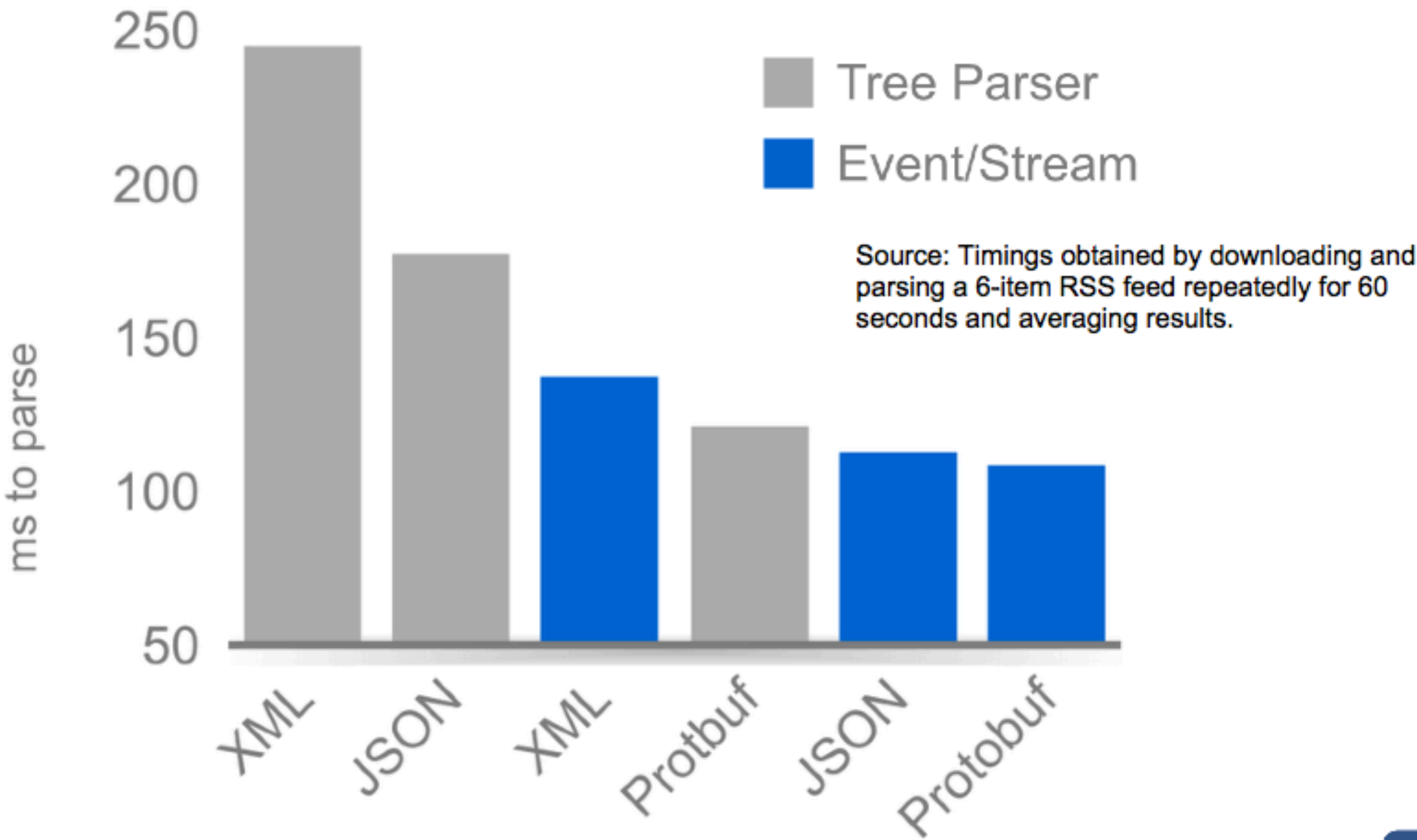

Power
Time


Power
Time


Power
Time

Data Transfer

- Battery cost per byte
  - Radio usage, CPU usage
  - Minimise the amount of data transferred
- Reduce signal-to-noise ratio
  - How much of the data describes the structure and not the data?
  - XML is bulkier than JSON
  - JSON is bulkier than binary
- Use Gzip compression where possible
  - Decompressor is native code
    - *Cost* to decompress is less than *cost* to send uncompressed
- Consider time taken to parse


250
200
150
100
50
ms to parse
Tree Parser
Event/Stream
Source: Timings obtained by downloading and parsing a 6-item RSS feed repeatedly for 60 seconds and averaging results.
XML
JSON
XML
Protbuf
JSON
Protobuf

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