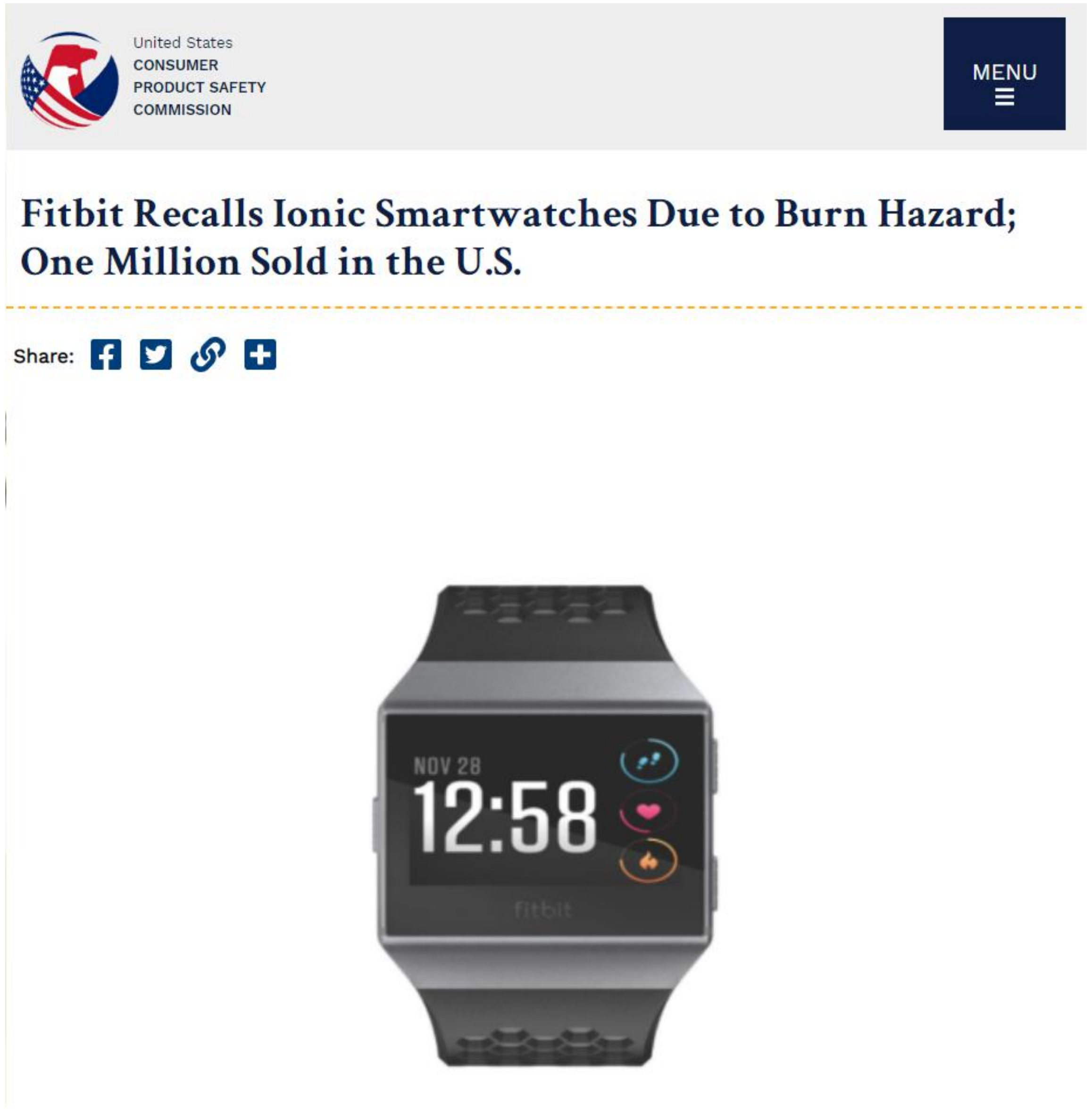
Energy Efficiency in Mobile Computing

Heat dissipation in wearables

- Close proximity of the wearables to the human body
  - Airflow is low, limiting cooling
  - Strict limitation on temperature
- Careful design of software necessary to help avoid many heat generation crises.



Fitbit Recalls Ionic Smartwatches Due to Burn Hazard; One Million Sold in the U.S.

- 115 reports of the watch's battery overheating in the US, and 59 internationally.
- There were 78 reports of burn injuries in the US, including two reports of third-degree burns, and four reports of second-degree burns.
- 40 burns were reported internationally.

Problem: Battery Power is Scarce

- Battery energy is most constraining resource on mobile device
- Most resources (CPU, RAM, WiFi speed, etc) increasing exponentially *except* battery energy (ref. Starner, IEEE Pervasive Computing, Dec 2003)

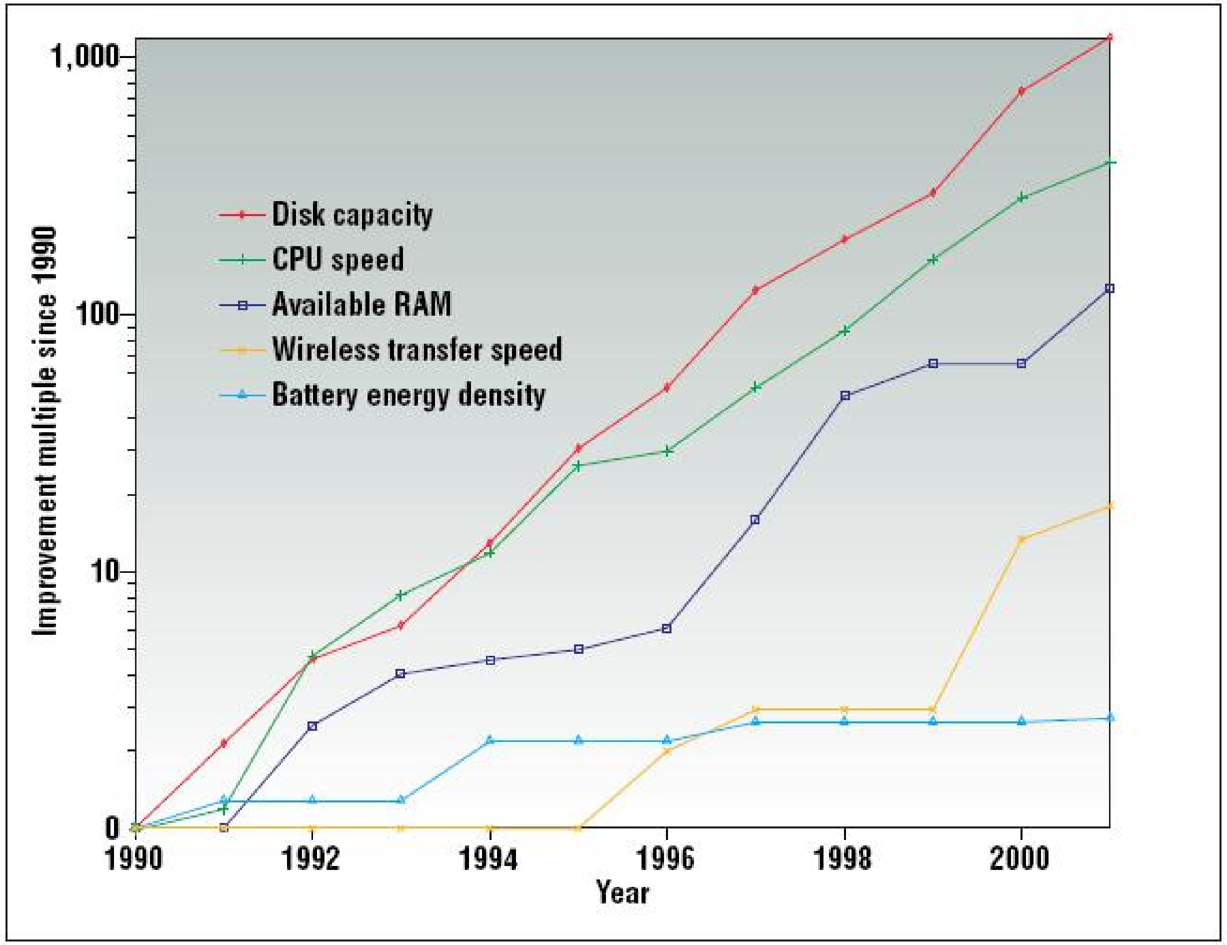


Figure 1. Improvements in laptop technology from 1990–2001.

Ideal wearables (Recap)

- Persist and provide constant access to information services.
- Sense and model context.
- Adapt interaction modalities based on the user’s context.

Ideal wearables: challenges

- Persist and provide constant access to information services.
  - Challenge: supply the energy to support all-time access
- Sense and model context.
  - Challenge: power enough sensors
- Adapt interaction modalities based on the user’s context.
  - Challenge: power sufficient computing capacity for context learning

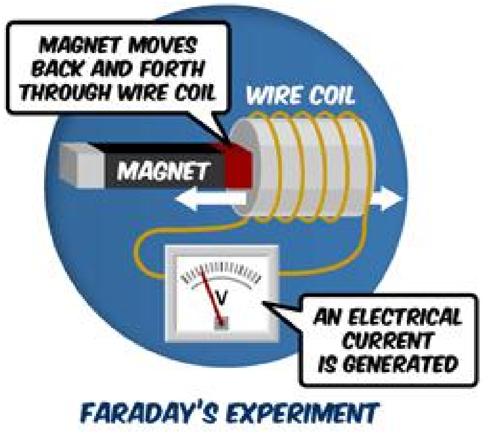
Summary: battery capacity continues to bottleneck the ideal wearables

Energy Harvesting for Wearable Systems

Energy Harvesting

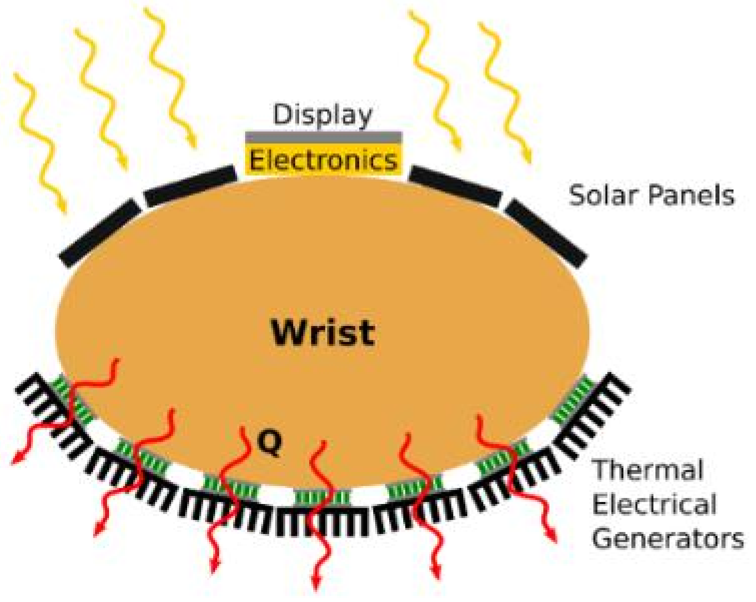
- What is it?
- Macro scale- wind and solar
- Micro scale- electromagnetic and piezoelectricity

[3]


MAGNET MOVES BACK AND FORTH THROUGH WIRE COIL
WIRE COIL
MAGNET
V
AN ELECTRICAL CURRENT IS GENERATED
FARADAY'S EXPERIMENT

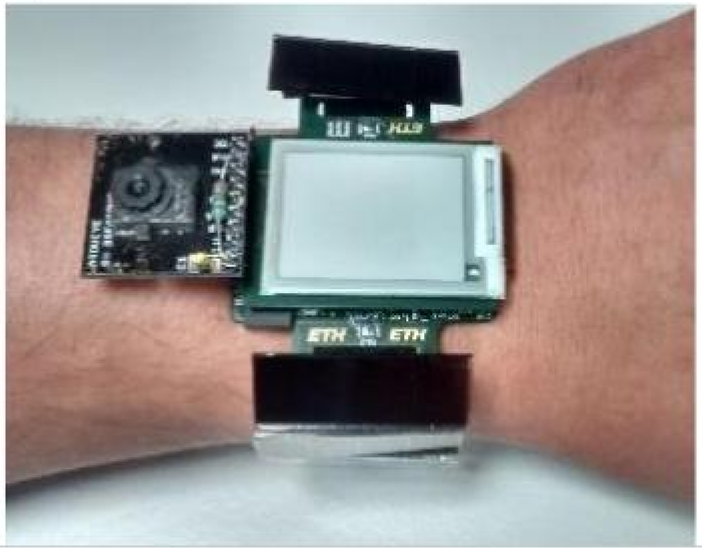
Thermal Electrical Generator

Havest body heat and convert it to electricity



Texas Instruments Innovation Challenge: Europe Design Contest 2015

Self-Sustainable Smart Wearable Device with Energy Harvesting for Context Recognition Applications



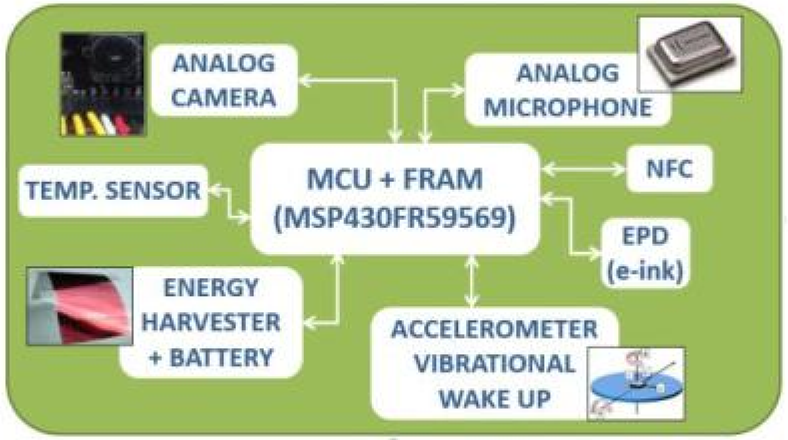
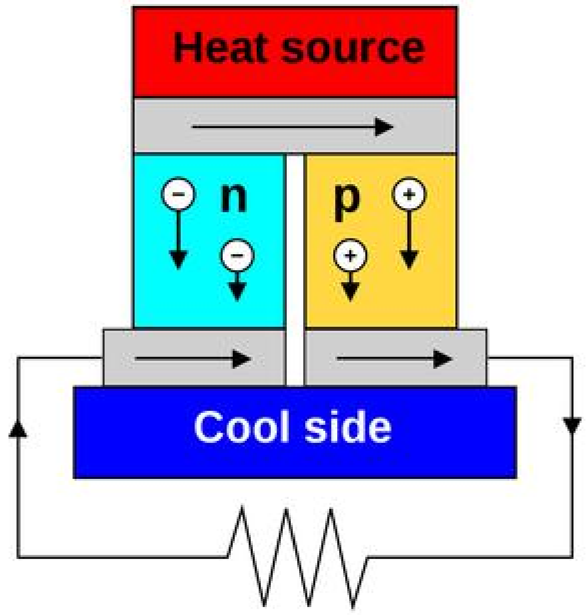

ANALOG CAMERA
ANALOG MICROPHONE
TEMP. SENSOR
MCU + FRAM (MSP430FR59569)
NFC
EPD (e-ink)
ENERGY HARVESTER + BATTERY
ACCELEROMETER VIBRATIONAL WAKE UP

Fig. 2: Block diagram of the smartwatch

- A thin conductive material takes advantage of the temperature difference between its two sides to produce electricity.
- This is known as the Seebeck effect.



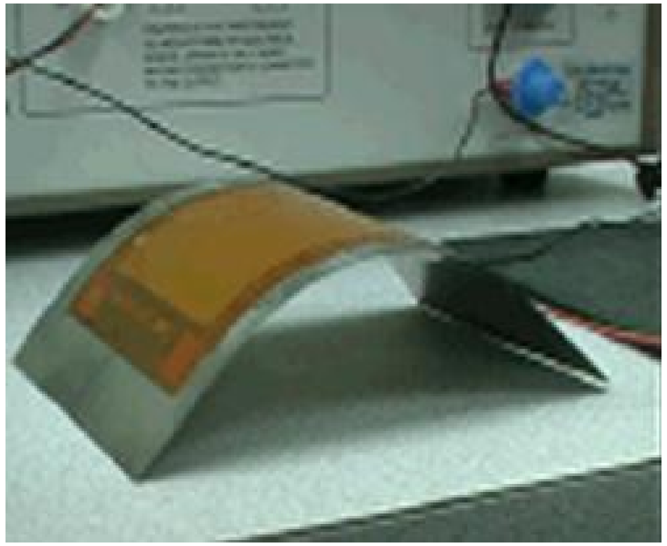
- If a thermoelectric device is located on skin, it will produce power as long as the ambient air is at a temperature that is lower than the skin.
- A device that is one square centimeter in area can yield up to 30 microwatts.
- If these generators are located side by side, the amount of power being harvested is increased.

Piezoelectricity

- Deform to get voltage

Apply voltage to deform

[4]



[5]

Piezoelectricity

- What is it?
- Occurs in polycrystalline materials
  - (For example: Quartz, lead-zirconate-titanate, Rochelle salt, etc.)

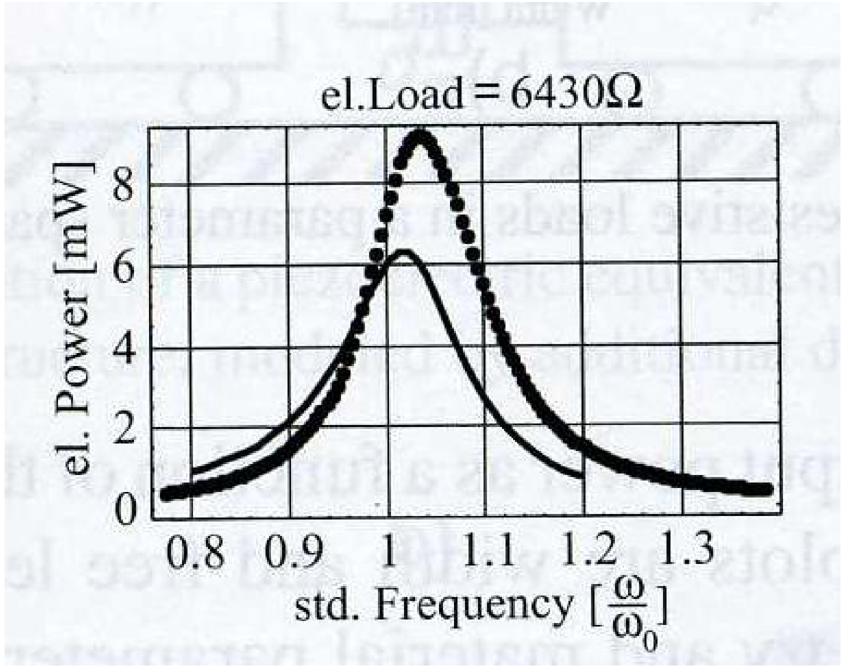
[4]

Piezoelectricity- Output

Output always different

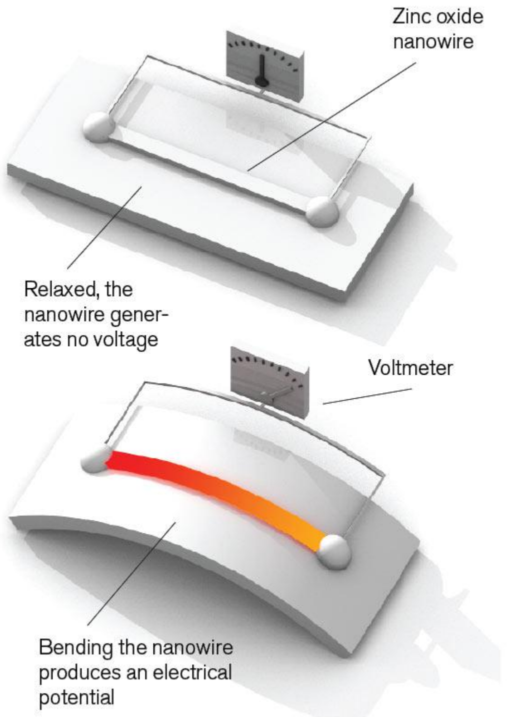
High voltage, low current

Sample output:

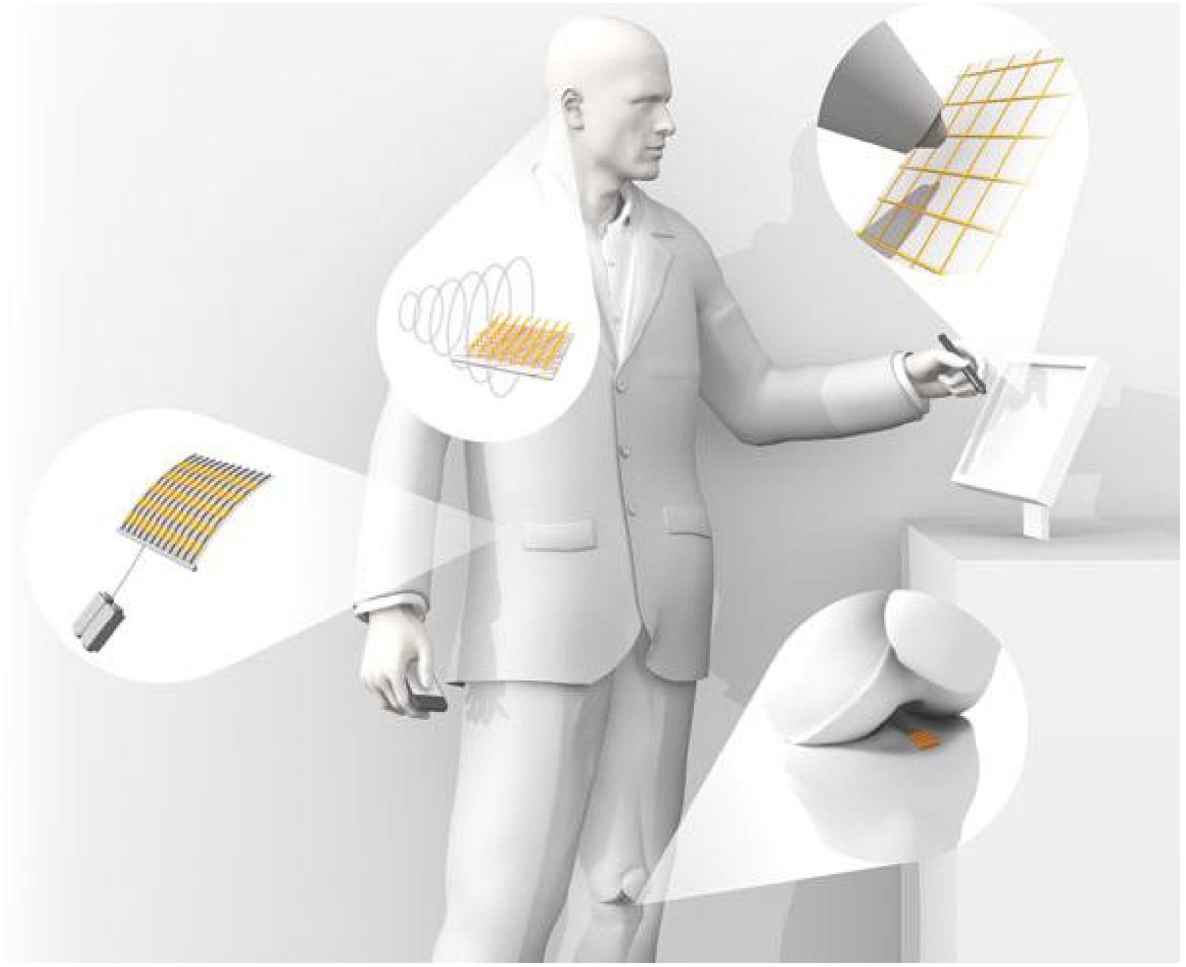


[6]

Future Outlook


Zinc oxide nanowire
Relaxed, the nanowire generates no voltage
Voltmeter
Bending the nanowire produces an electrical potential

[7]



Power Management in Android

Android Power Management Design

- Apps and services must request CPU resource with ‘wake locks’ through the Android application framework 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 is managed.

Wakelock

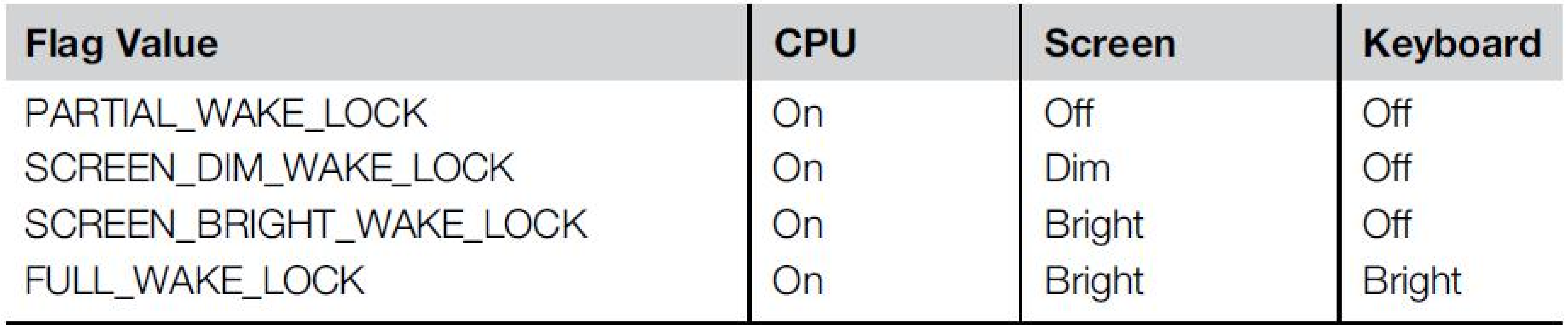
- By default, Android tries to put the system into a sleep or better a suspend mode as soon as possible
- Some applications need to assure that the screen stays on or the CPU stays awake to react quickly to inputs
  - To do so, we use wakelock
  - Without active wake locks, the CPU will be turned off
  - Partial wakelocks can turn off screen or keyboards.

Android Wakelock Architecture

- Wakelocks are power-managing software mechanisms that make sure that your Android device doesn’t go into deep sleep
- Because a given application needs to use your system resources.
- Class PowerManager.WakeLock is mechanism to indicate that the application requires the device to stay on.

Wakelock Levels

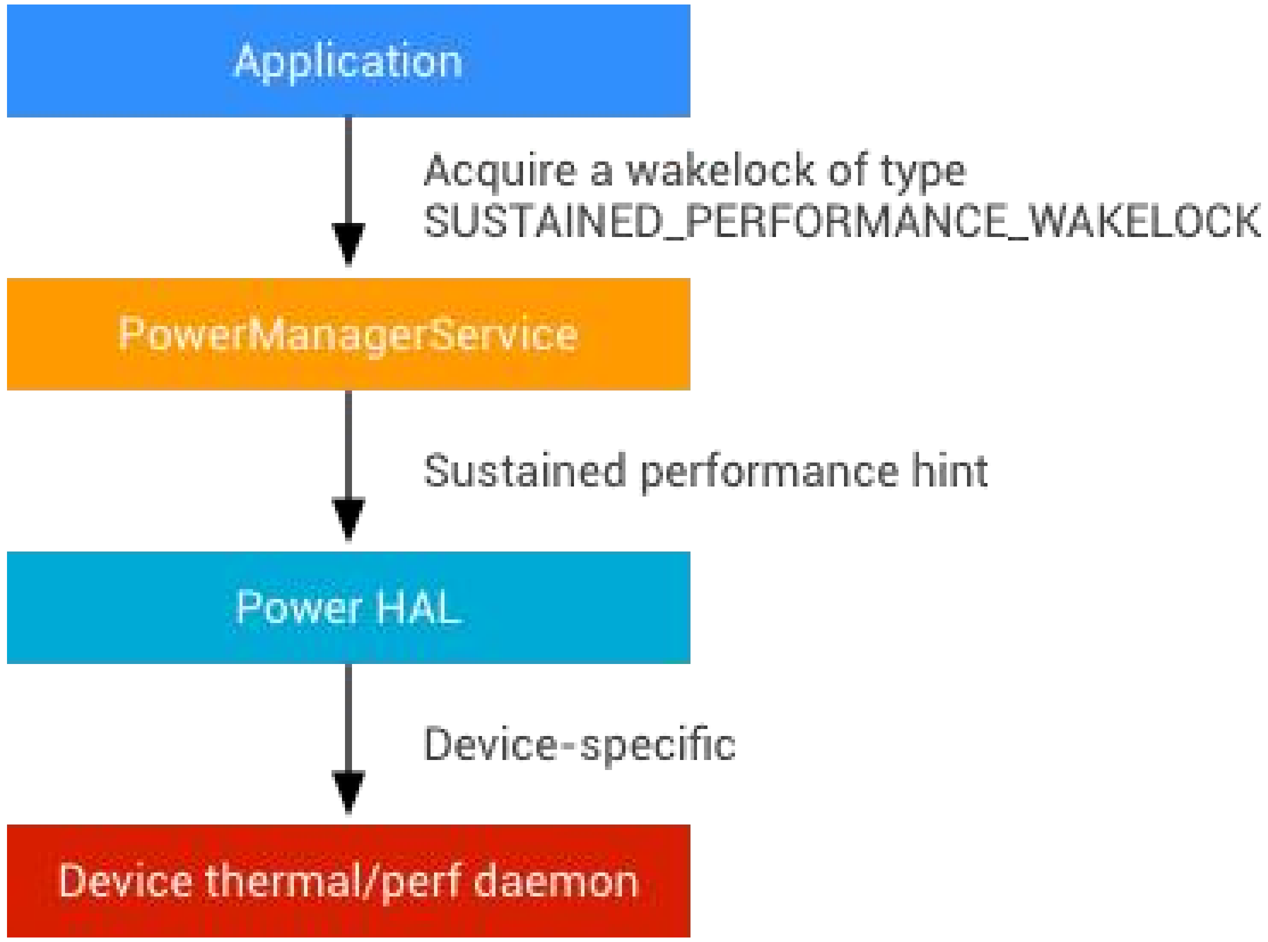
Experiment with it in your upcoming camera programming lab



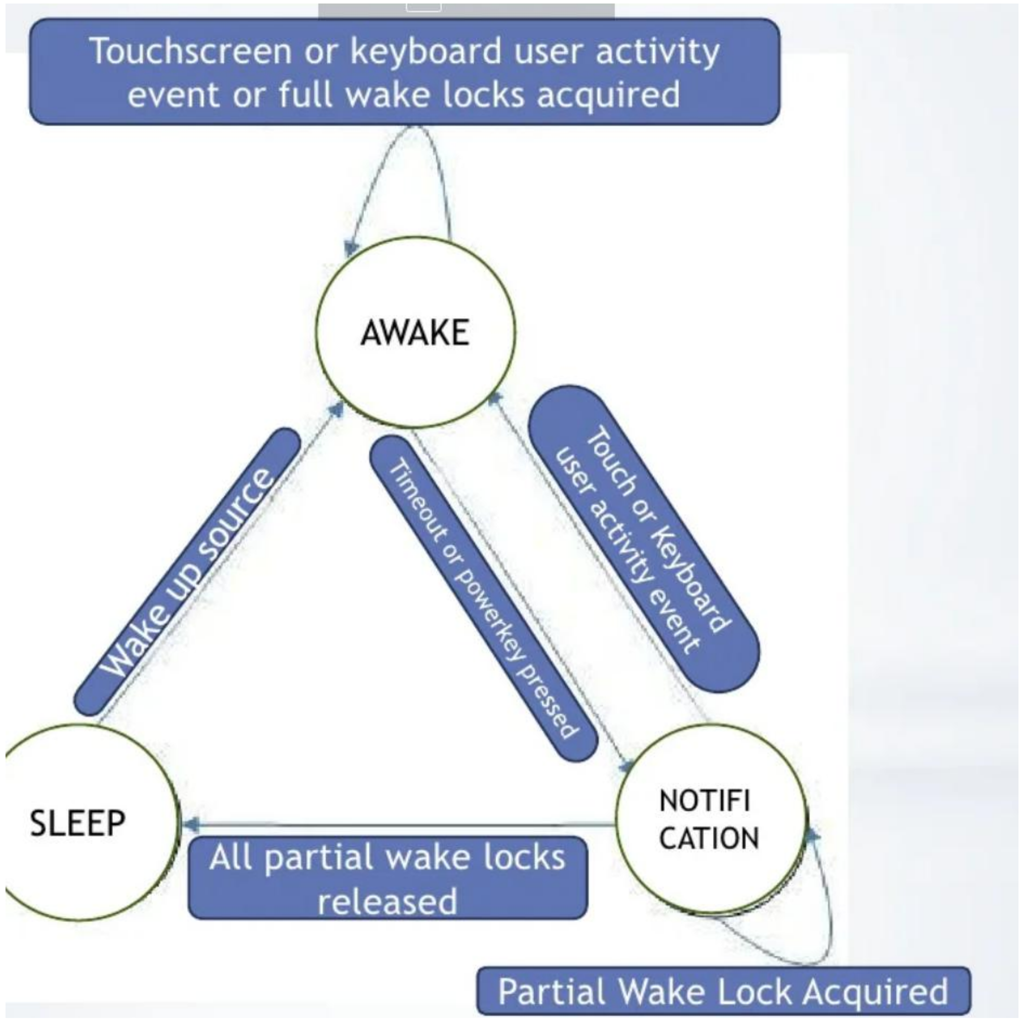
Flag Value	**CPU**	**Screen**	**Keyboard**
PARTIAL_WAKE_LOCK	On	Off	Off
SCREEN_DIM_WAKE_LOCK	On	Dim	Off
SCREEN_BRIGHT_WAKE_LOCK	On	Bright	Off
FULL_WAKE_LOCK	On	Bright	Bright

Android PM Design

The PowerManager class can control the power state of the device

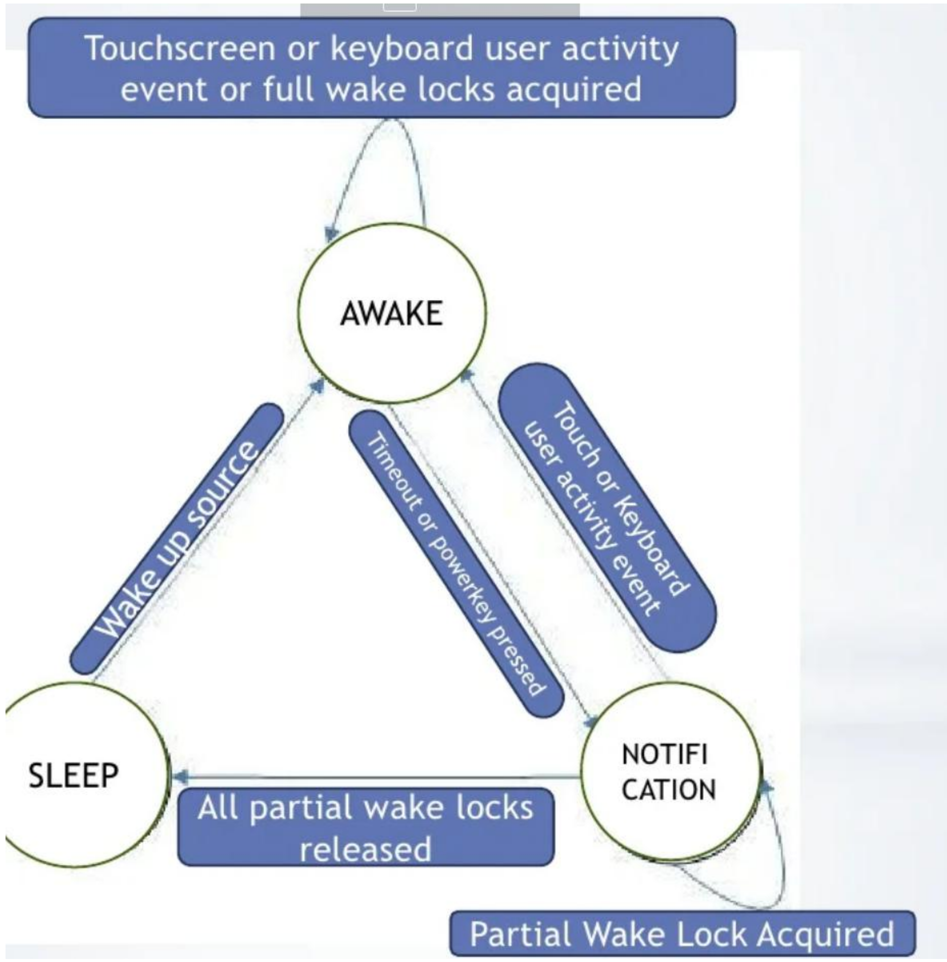

Application
Acquire a wakelock of type SUSTAINED_PERFORMANCE_WAKELOCK
PowerManagerService
Sustained performance hint
Power HAL
Device-specific
Device thermal/perf daemon

States for 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

States for Android Power Management

- When an application acquires full wake lock or an UI activity event occurs, the machine enters ‘wake’ state
- If timeout happens, the machine enters ‘notification’ state
  - If partial wake locks are acquired, it will remain in ‘Notification’
  - If all partial locks are released, the machine go into ‘sleep’


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

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 the power button is pressed, an API goToSleep() is called in the PowerManager
- goToSleep() will force release all wake locks

Software Techniques for Battery Conservation in Mobile Devices

- Three things to help
  - Make apps *Lazy First*
  - Take advantage of platform features
  - Use tools to identify battery draining components

Lazy First

- **Reduce**
  - Are there redundant operations your app can cut out?
- **Defer**
  - Does an app need to perform an action right away?
- **Coalesce**
  - Can work be batched, instead of putting the device into an active state many times?

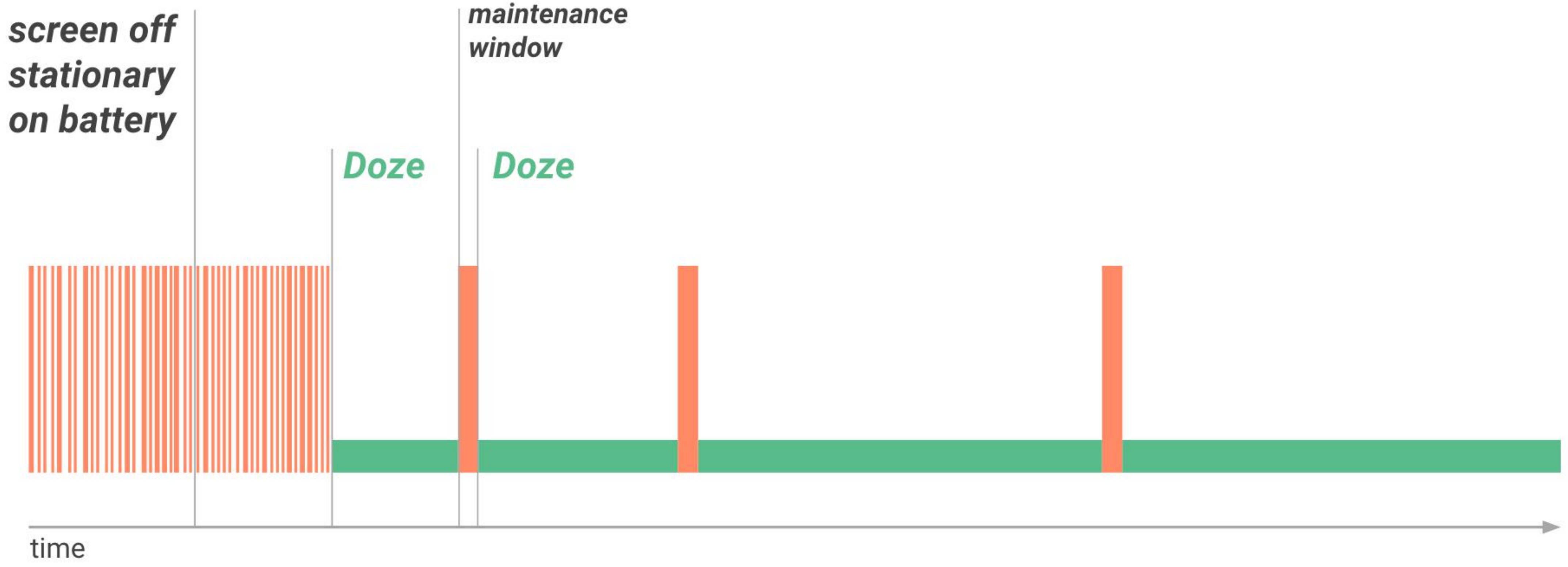
- Three things to help
    - Make apps *Lazy First*
    - **Take advantage of platform features**
    - Use tools to identify battery draining components

Android platform features

- **Doze** and **App Standby**:
  - Two power saving features
  - Manage how apps behave when a device is not connected to a power source
- Doze
  - Defers background CPU and network activities when the device is unused for a long time
- App Standby
  - Defers network activity when user has not recently interacted

Doze Mode

- Starts when a device is **unplugged** and **stationary** for **a period of time**, with the **screen off**
- Periodically, the system exits Doze to let apps complete their deferred activities, such as running pending syncs, jobs, alarms, and network access



Doze Mode

- Apps on your phone will have no network access, the system will ignore “wakelocks” when apps try to keep the device from going to sleep, and no background tasks will be allowed to run.
  - High-priority push messages will still show up.
  - So for example, a Hangouts message will appear on a device that’s in Doze mode.

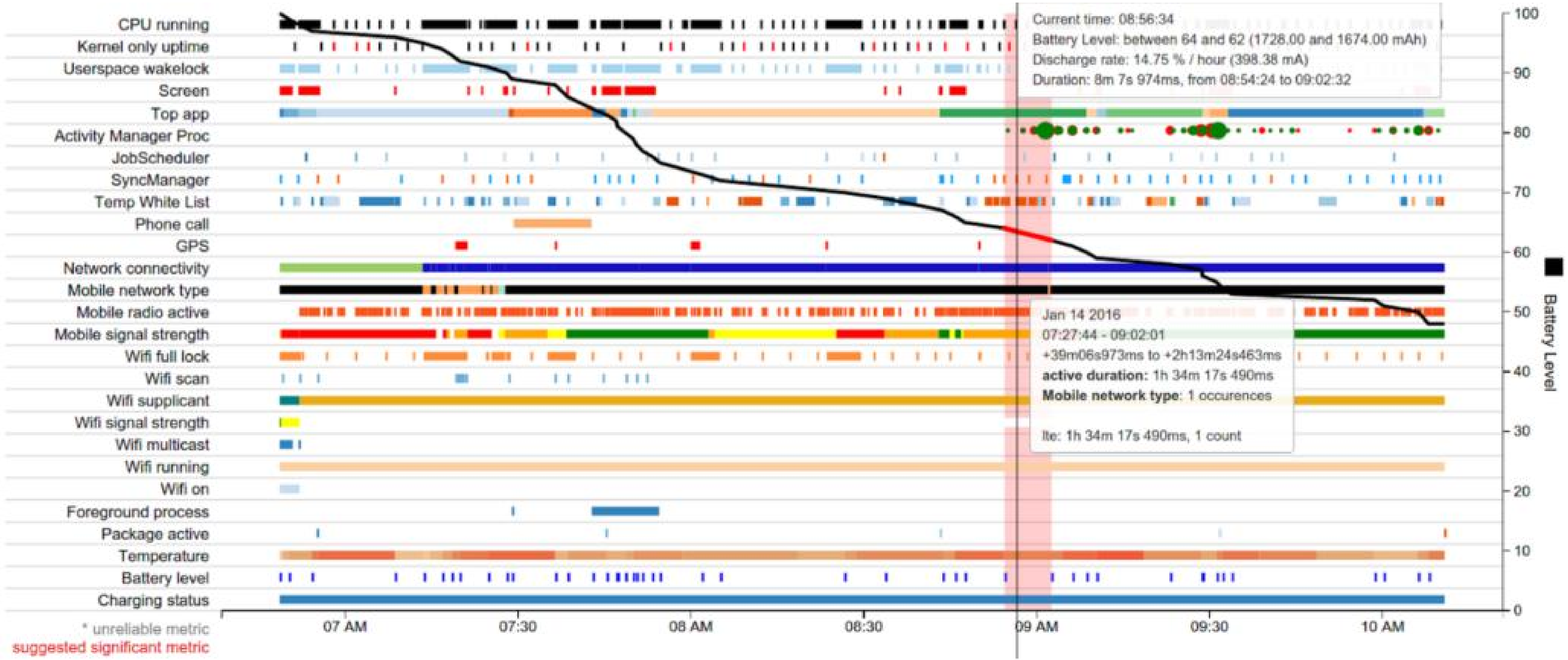
Android App Standby

- The system puts an app into Standby mode and defers its network access if for a certain period of time,
  - the user has not explicitly launched the app, and
  - the app doesn’t have a foreground process (activity), and
  - the app doesn’t generate notifications, and
  - the app is not an active device admin app.
- When the phone is plugged to power, apps are released from standby state
- If the device is idle for a long period of time, idle apps access network around once a day

- Three things to help
    - Make apps *Lazy First*
    - Take advantage of platform features
    - **Use tools to identify battery draining components**

Battery Historian: A tool to analyze power use in Android

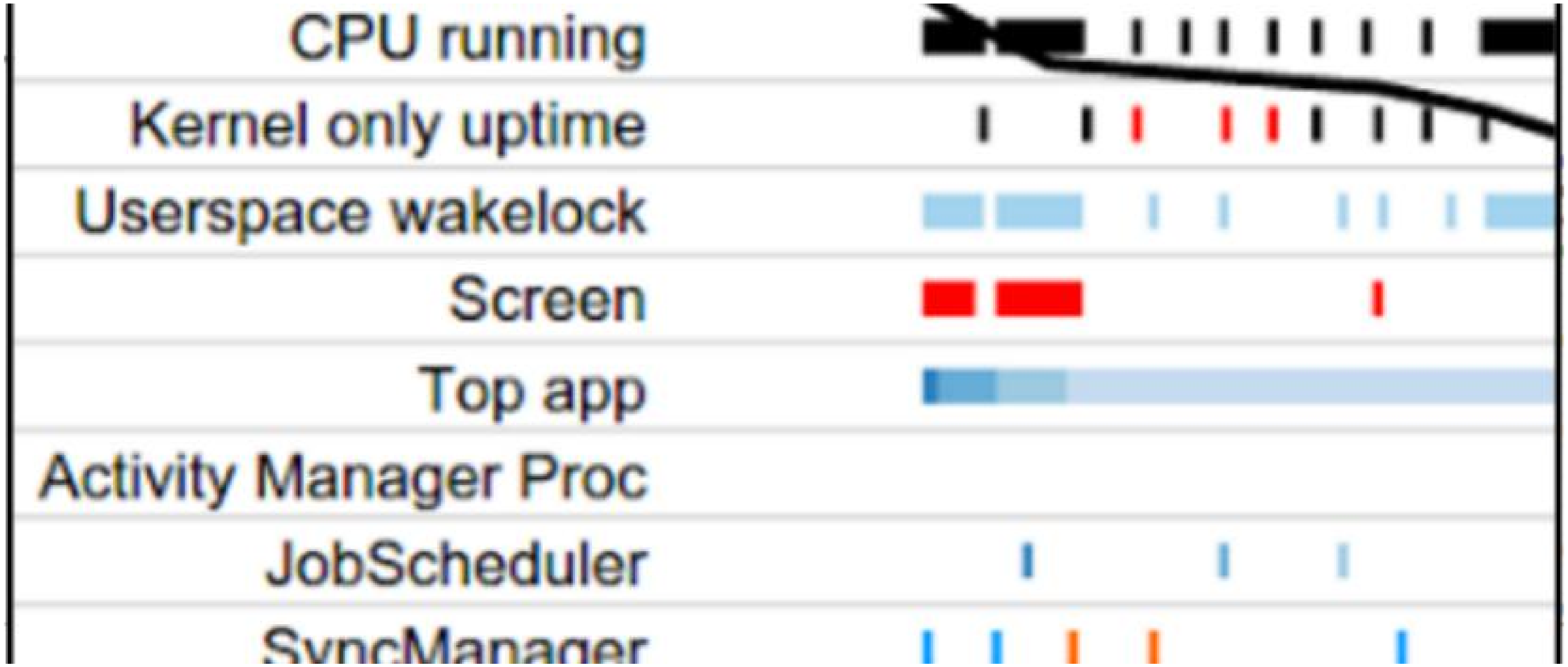
provides a system-wide visualization of various app and system behaviors, along with their correlation against battery consumption over time.


Current time: 08:56:34
Battery Level: between 64 and 62 (1728.00 and 1674.00 mAh)
Discharge rate: 14.75 % / hour (398.38 mA)
Duration: 8m 7s 974ms, from 08:54:24 to 09:02:32

Jan 14 2016
07:27:44 - 09:02:01
+39m06s973ms to +2h13m24s463ms
active duration: 1h 34m 17s 490ms
Mobile network type: 1 occurences

lte: 1h 34m 17s 490ms, 1 count

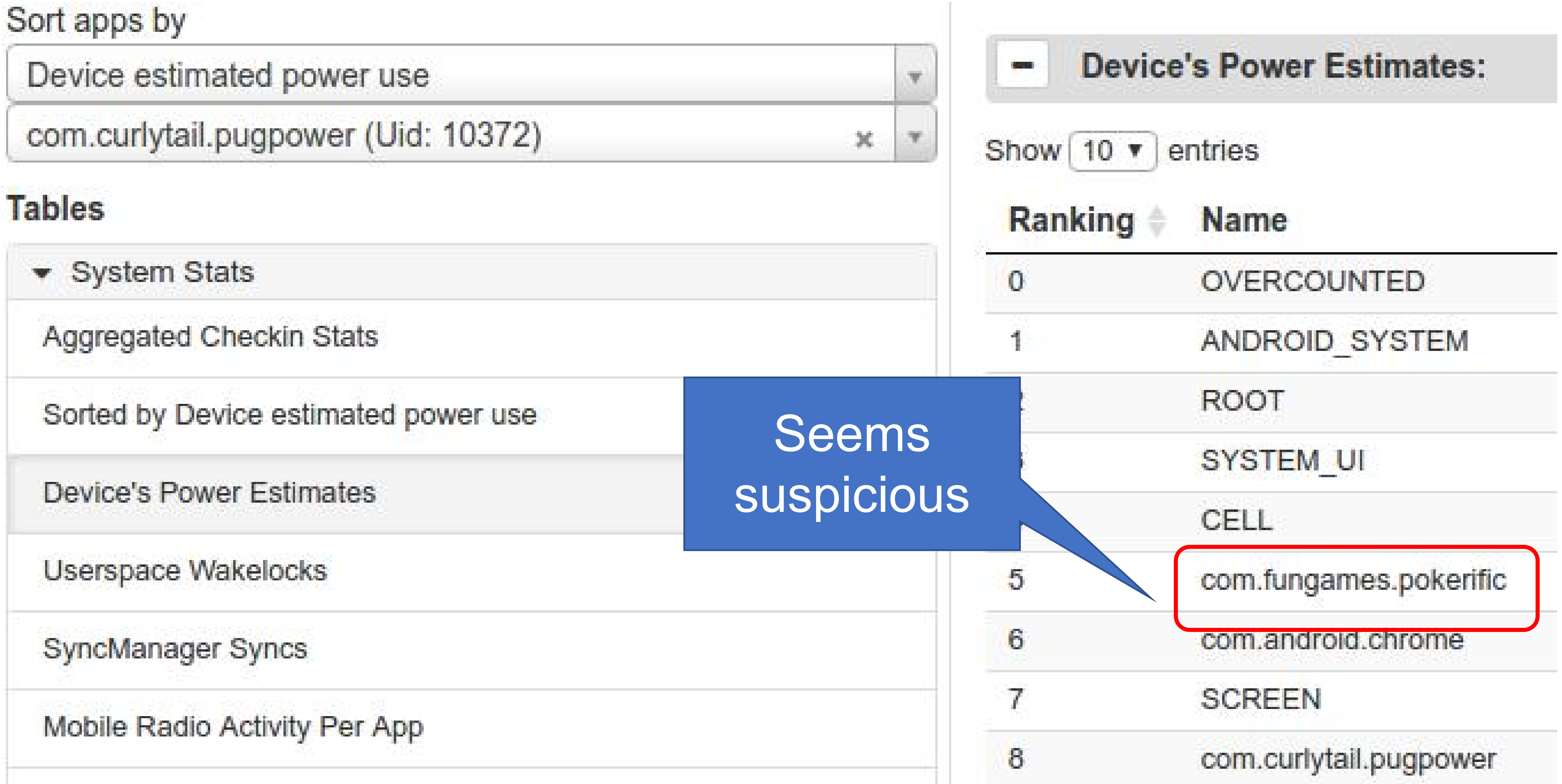
A closeup look



Battery historian: App-specific view

- Provides tables for the following data:
  - The app’s estimated power use on the device.
  - Network information.
  - Wakelocks.
  - Services.
  - Process info.

Display the ranking of app powers



Sort apps by

Device estimated power use

com.curlytail.pugpower (Uid: 10372)

Tables

- System Stats
  - Aggregated Checkin Stats
  - Sorted by Device estimated power use
  - Device's Power Estimates
  - Userspace Wakelocks
  - SyncManager Syncs
  - Mobile Radio Activity Per App

Device's Power Estimates:

Show 10 entries

Ranking	Name
0	OVERCOUNTED
1	ANDROID_SYSTEM
2	ROOT
3	SYSTEM_UI
	CELL
5	com.fungames.pokerific
6	com.android.chrome
7	SCREEN
8	com.curlytail.pugpower

Seems suspicious

Other cases where Battery Historian can help

- Examples:
    - Firing wakeup alarms overly frequently (every 10 seconds or less).
    - Continuously holding a GPS lock.
    - Scheduling jobs every 30 seconds or less.
    - Scheduling syncs every 30 seconds or less.
    - Using the cellular radio more frequently than you expect.