Overview of CPU Power Consumption and Management in Smartphones

Prof. Sasu Tarkoma

University of Helsinki, Aalto University,
Helsinki Institute for Information Technology

Contents

- Modern smartphone SoC and CPUs
  - The CPU: power states
  - Power management basics
- Smartphone solutions
  - Linux CPU Frequency subsystem
  - Power models
- Intra-device task offloading
  - Sensor hub
  - Heterogeneous multiprocessing
- Computation offloading

Smartphones

- Smartphones have become hubs for applications and connecting with the Internet
- Cloud has emerged as a backend for mobile applications
- Mobile data and WiFi are the dominant protocols for connecting with Internet resources
- The next generation solutions are addressing limitations of the current smartphones
  - Coordination of resource usage
  - Offloading in its many forms
  - Heterogeneous environment and the emergence of IoT / M2M / wearables

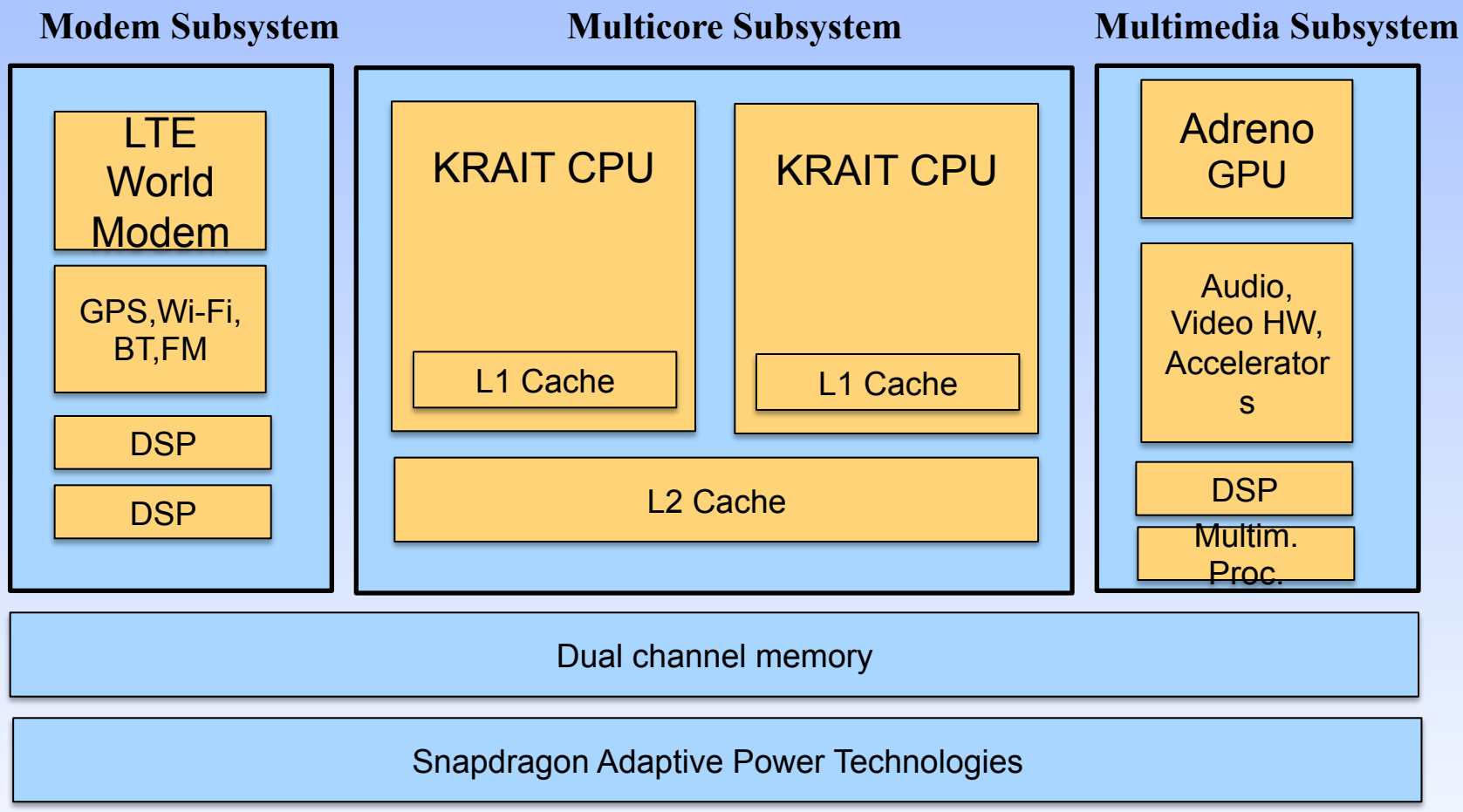
Observations

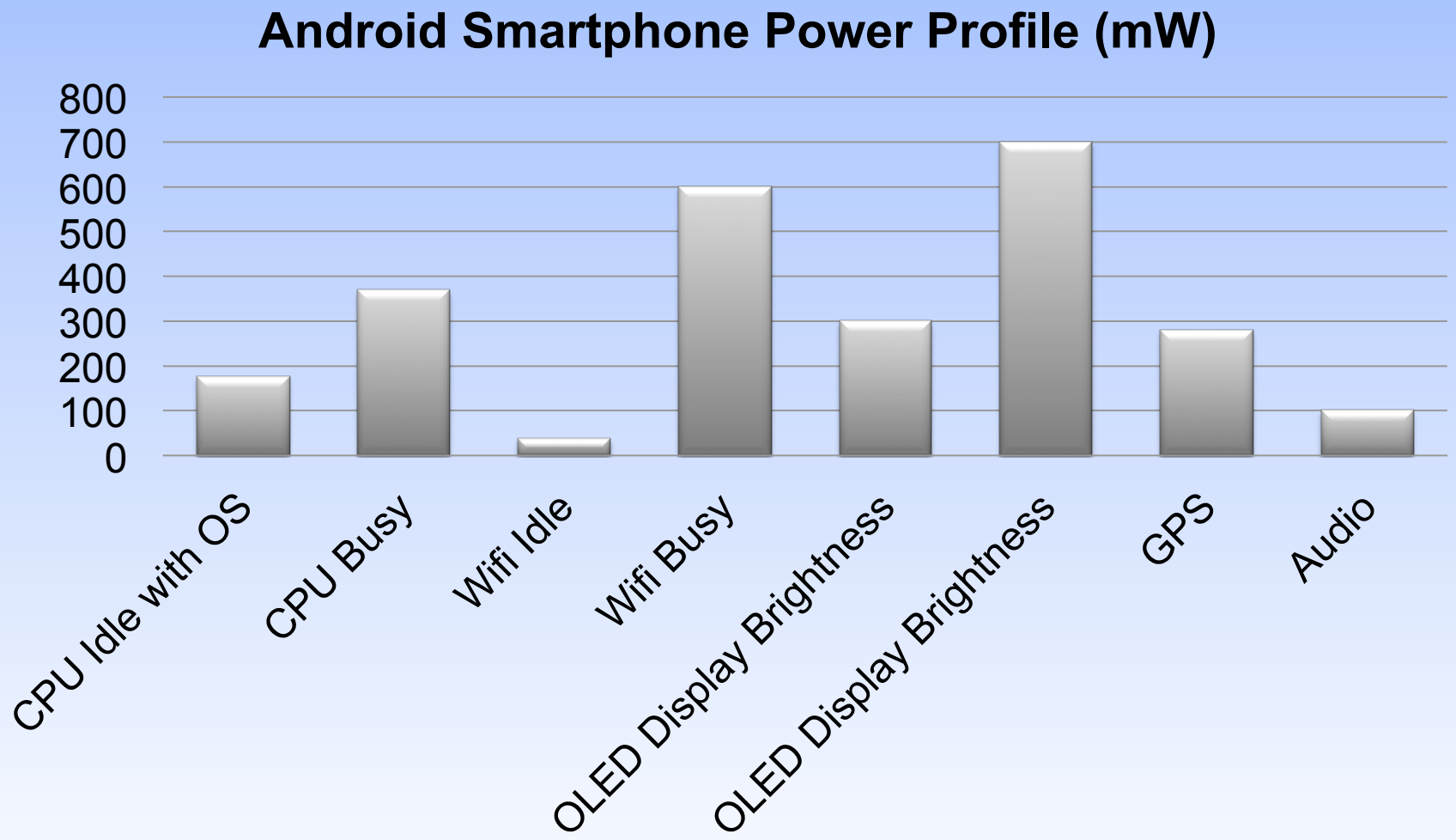
- Smartphone and mobile device hardware and software evolve rapidly
- Multiple wireless protocols
- Heterogeneous computing over multiple cores
  - Dedicated subsystems (sensor hubs)
  - Increasing number of sensing subsystems
  - Always-on sensing
- Battery technology has not kept pace with the development
- Software is not, in many cases, optimized
- Difficult to balance between local versus distributed processing
- Difficult to control traffic across interfaces

Mobile Evolution

	1995	2000	2005	2010	2015
Processor	Single	Single	Single	Dual-core	Quad-core and beyond, auxiliary processors, sensor hubs
Cellular generation	2G	2.5-3G	3.5G	Transition toward 4G	4G
Standard	GSM	GPRS	HSPA	HSPA, LTE	LTE, LTE-A
Downlink (Mb/s)	0.01	0.1	1	10	100
Display pixels (x1000)	4	16	64	256	1024
Communications modules	-	-	WiFi, Bluetooth	WiFi, Bluetooth	WiFi, Bluetooth LE, RFID
Battery capacity (Wh)	1	2	3	4	5
Software (MB)	0.1	1	10	100	1000

Example Smartphone SoC




Android Smartphone Power Profile (mW)
800
700
600
500
400
300
200
100
0
CPU Idle with OS
CPU Busy
Wifi Idle
Wifi Busy
OLED Display Brightness
OLED Display Brightness
GPS
Audio

CPU Total Power

- The total power of CMOS logic circuits are determined by the clock rate, the supply voltage, and the capacitances of the transistors
  - Switching power and static power:

$$P = P_{switch} + P_{static}$$

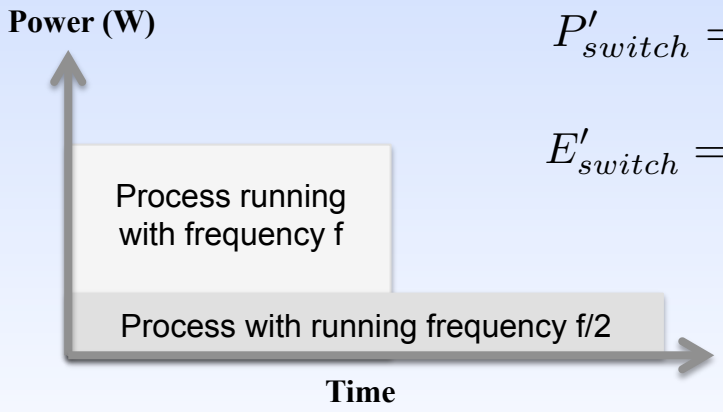
- Dynamic switching loss is given by:

$$P_{switch} = \alpha \times C \times V^2 \times f$$

- By varying clock rate and supply voltage, it is possible to obtain linear and quadratic improvements

CPU Power Saving

- Running a task at a slower speed saves energy; however, it will lake longer to execute the task thus affecting the performance
- Dynamic voltage and frequency scaling favor parallelism and having multiple voltage/frequency scaled cores executing tasks.



$$P'_{switch} = \alpha \times C \times (\frac{V}{2})^2 \times \frac{f}{2} = \frac{1}{8} P_{switch}$$

$$E'_{switch} = P'_{switch} \times T \times 2 = \frac{1}{8} P_{switch} \times T \times 2 = \frac{1}{4} P_{switch} \times T$$

¼ energy savings with the doubling of the execution time

Scaling SoCs

- Static power leakage is affecting the feasibility of voltage scaling.
  - A linear decrease of voltage will result in a quadratic decrease of the switching loss; however, it will only result in a linear decrease of the static leakage.
  - Static leakage can become dominant with low voltage integrated circuits with high density of transistors
- To address leakage, SoCs use power gating to allow portions of the system to be switched off when necessary.
- Low power requirements are also driving the trend for multicore systems with many voltage and frequency controlled cores.

CPU Frequency and Relative Power Consumption

Clock frequency (MHz	Core voltage (V)	Relative power consumption
250	1.075	100%
500	1.2	249%
550	1.275	309%
600	1.35	378%

Source: J. Kurtto, “Mapping and improving the energy efficiency of the Nokia N900”. M.Sc. Thesis. University of Helsinki, Department of Computer Science.

Power Optimization Levels

- Smartphone and mobile device power optimization happens on multiple levels:
  - **Silicon-level**, in which the transistor capacitance and the chip design affect the energy efficiency
  - **SoC-level**, in which multiple power/voltage/clock domains can be used to support granular power management with the help of software and DVFS
  - **Software-level**, in which various power managers monitor and control the energy and power settings.

- A high-level framework is needed to perform system-wide tuning and optimization.

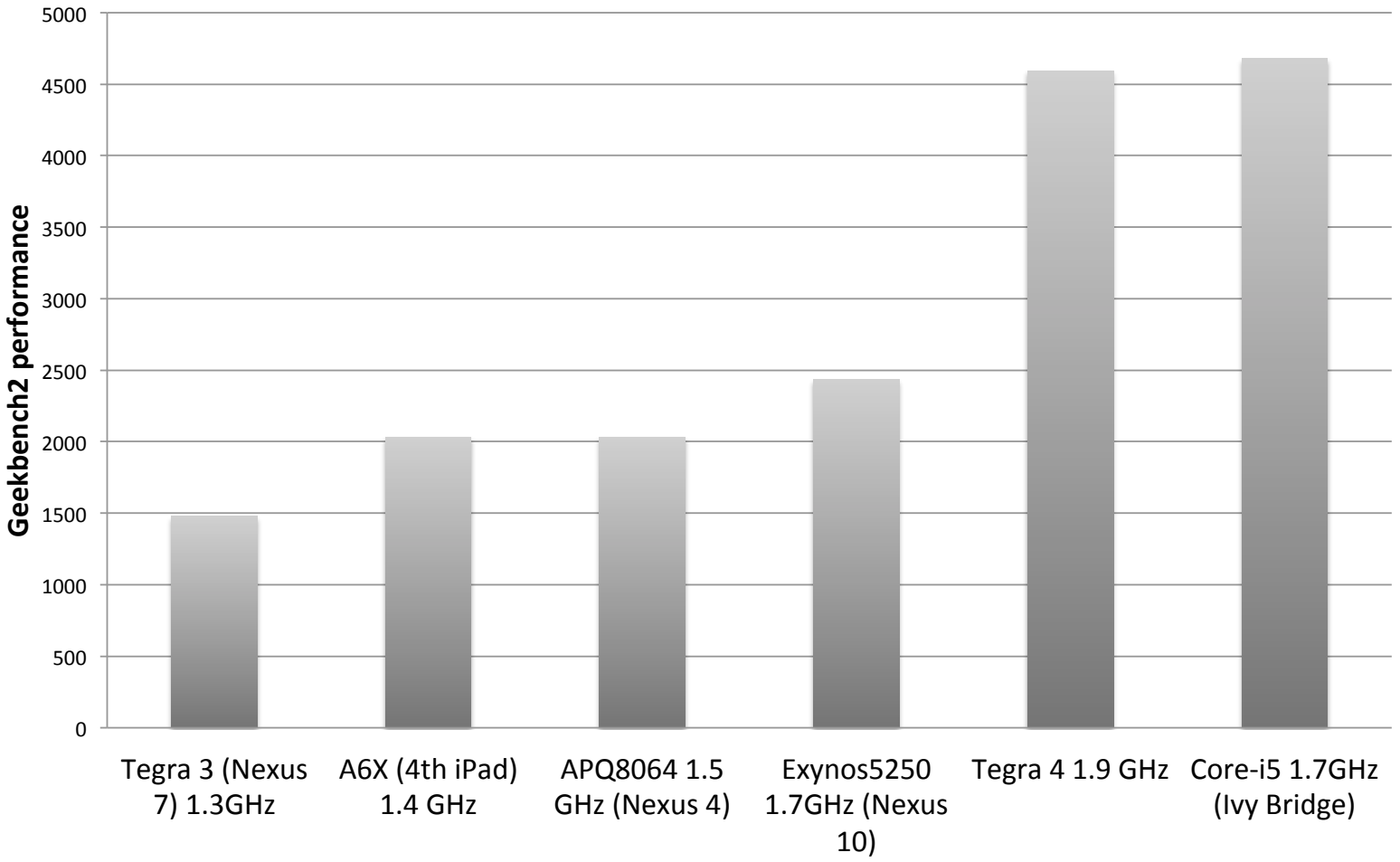
APM and ACPI

- Advanced Power Management (APM)
  - Firmware and BIOS level
  - Apps and drivers -> APM Driver-> BIOS APM -> hardware
  - Power management via device all and automatic based on device activity
  - Power management events
- Advanced Configuration and Power Interface (ACPI)
  - APM is replaced by ACPI
  - OS has the control
  - Power states: global, device, processor, performance

Smartphone CPU and SoC

- The five popular SoCs used by smartphones today are:
  - Qualcomm's **Snapdragon** consists of four versions from S1 to S4.
  - Texas Instrument's **OMAP** (Open Media Applications Platform).
  - Samsung's **Exynos** SoCs used in their smartphones and tablets. The latest version of Exynos is 5 Octa and it features a quad-core Cortex-A15 and a quad-core Cortex-A7, ARM Mali GPU, and auxiliary processors.
  - Nvidia's **Tegra** SoCs are multi-core and based on ARM cores with an ulta low- power (ULP) GeForce GPU. The latest version, Tegra 4, supports ARM- A15 cores in quad- or octa-core configurations.
  - **Apple's** CPUs and SoCs are designed by Apple based on the ARM architecture.

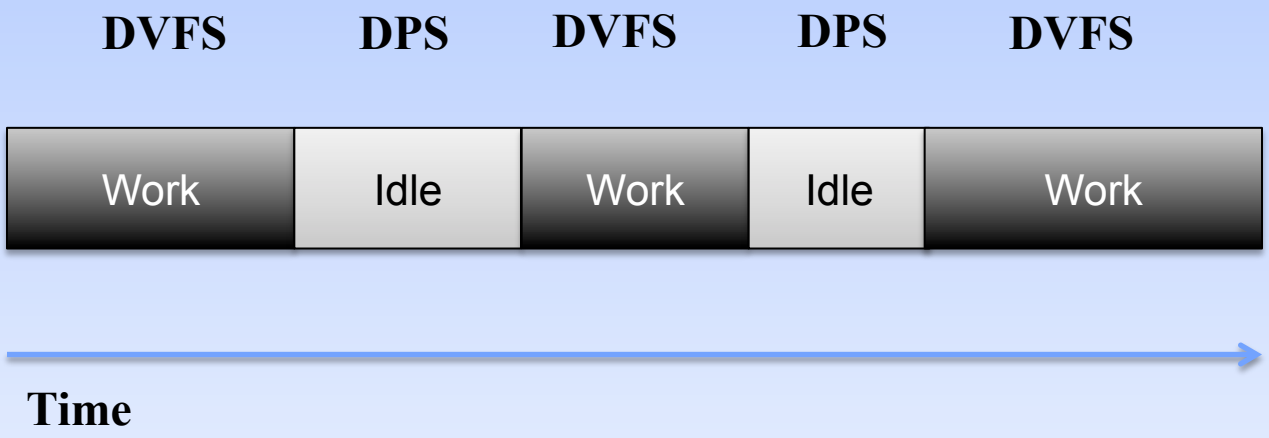
Mobile CPU Performance



Dynamic Power Management

- **Dynamic Power Management (DPM)** is a design methodology for energy and power management of dynamically reconfiguring systems. The goal for a DPM system is to provide the requested services and performance with a minimum power consumption.
- **DVFS (Dynamic Voltage and Frequency Scaling)**
  - Minimizes power needs by trying to reduce the operating frequency and the core voltage to levels sufficient to execute current tasks but with no excess resources left.
- **DPS (Dynamic Power Switching)**
  - DPS detects the true need of resource utilization in a CPU and, if there are no current computational tasks, forces the CPU into minimal power state.

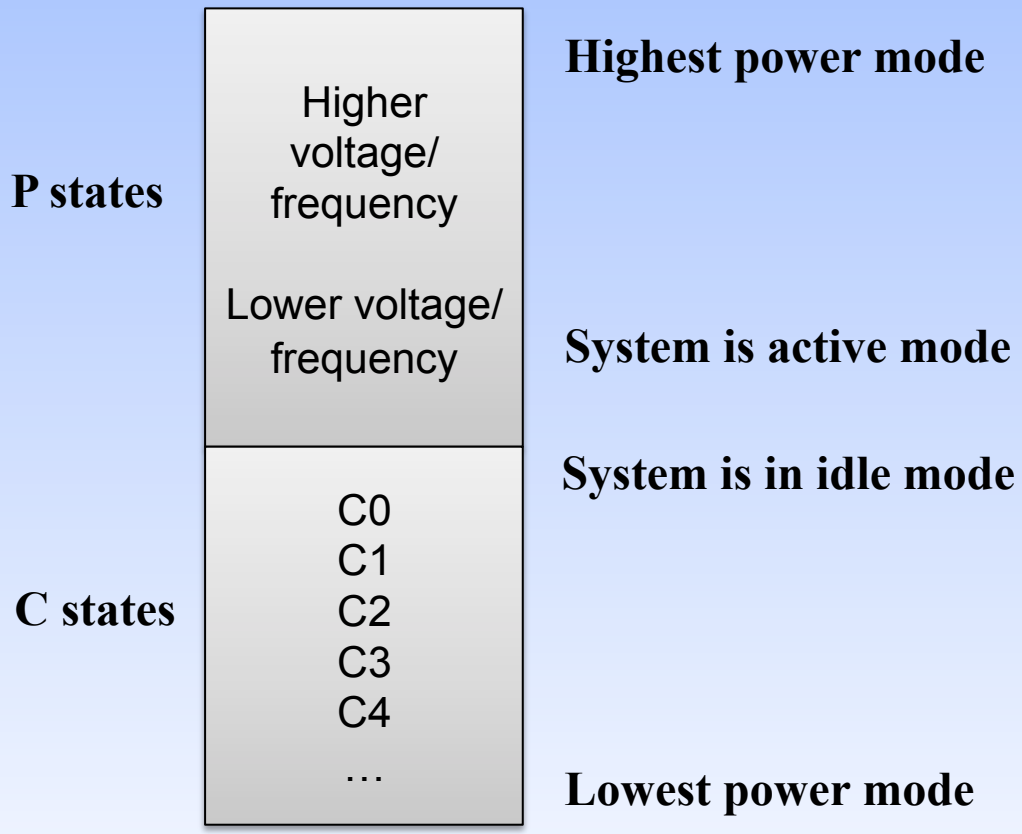
Example of Dynamic Power Management



P and C States

- At processor level we can manage power consumption with two different strategies:
  - Control of the CPU performance states using frequency and core voltage (**P-states**).
  - Use of processor operating states (**C-states**).
- **The Advanced Configuration and Power Interface (ACPI)** specification is the industry standard solution for power management used by the desktop and laptop industry.
  - Current smartphones use these states as well, but they are not based on ACPI.

Overview of P and C States



Power Management Components

- A **subsystem** for connecting the low-level technologies and policies at higher level.
- A system of in-kernel **policy governors**. They are essentially pre-configured power schemes with the ability to modify the clock frequency according to required needs. Typically these governors use the P-states to to change frequencies for lower power consumption. They will switch between clock frequencies, on the basis of current CPU utilization level trying to save power while not unduly losing performance. The governors are tunable allowing some customizing of the frequency scaling.
- **Drivers** implementing the technology in a CPU-specific way.

P and C States: Example

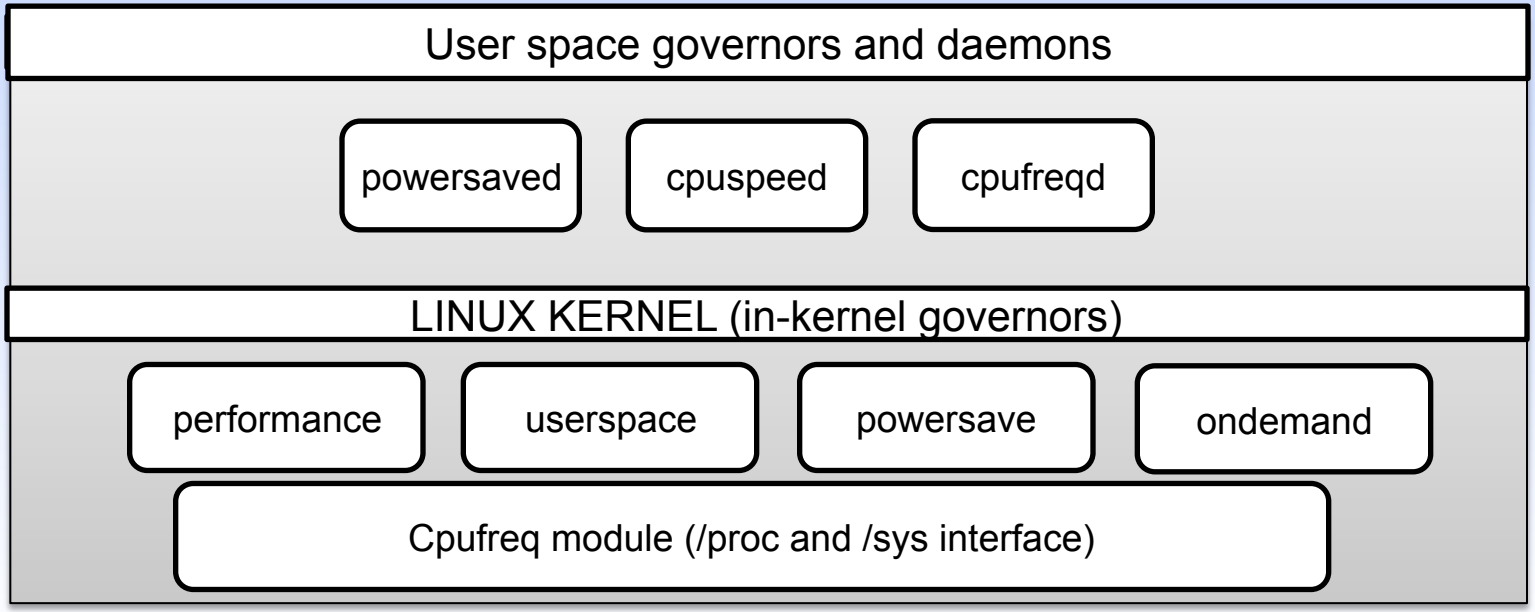
- The difference between C-states and P-states is one of operational level and can be summarized as follows:
  - In C-states starting from C1 the processor is idle but partially in use. P-state is an operational concept and defined solely by the clock frequency and core voltage.

State	Idle Power (mW)
C0	433
C1	390
C2	330
C3	200
Without idle states	1060

The Nexus 4 smartphone power states. Nexus 4 is based on the Snapdragon S4 SoC.

Linux CPU Frequency Subsystem

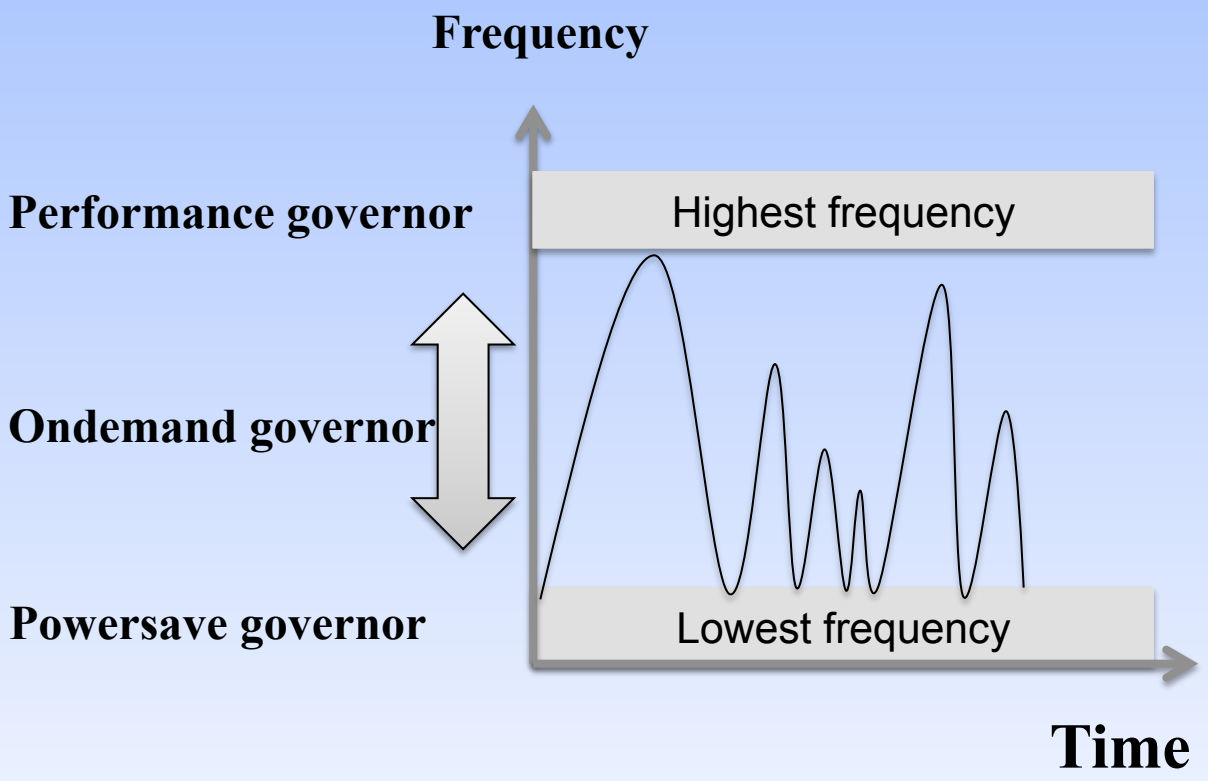
- **The Linux CPU frequency subsystem** that has supported dynamic processor frequencies since the 2.6.0 Linux kernel.
- The CPUfreq subsystem uses governors and daemons for implementing a static or dynamic power management policy.



Governors

- **Performance governor** that gives the highest CPU frequency and performance. This governor statically sets the highest frequency value and allows the tuning of this highest value.
- **Powersave governor** that sets the lowest CPU frequency and system speed.
- **Userspace governor** that allows the CPU frequency to be set manually. The component can be used to implement custom power policies.
- **Ondemand governor** is an in-kernel governor to dynamically set the CPU frequency based on CPU utilization.
- **Conservative governor** is similar to the on demand governor, but allows a more gradual increase of the power consumption.

Example: Governors



Time resolution

- The default governors use millisecond resolution for decisions, the thresholds are specified in microseconds
- Userspace governor can be implemented with finer-grained granularity (microseconds)
- Intel Speedstep Technology can switch frequency with latency of 10 microseconds
- Faster sampling results in quicker response time for changes in the workload

Ondemand Governor

- For each CPU:
  - Every X milliseconds:
    - Get Utilization since last check
    - If utilization > UP_THRESHOLD
      - Increase frequency to MAX
  - Every Y milliseconds:
    - Get utilization
    - If utilization < DOWN_THRESHOLD
      - Decrease frequency 20%
- Conservative is similar but more gradual increase
- Used on many Android devices, for example Samsung S4
- *V. Pallipadi and A. Starikovskiy, "The Ondemand Governor," The Linux Symposium, 2006.*

Linux cpufreq

- cpufreq-info
  - Info on the current governor and settings
- cpufreq-set allows setting of:
  - d minimum frequency,
  - u maximum frequency,
  - f specific frequency (userspace governor must be set first) and
  - g governor on a
  - c specific CPU
- Cpufreq allows activating governors on every available CPU (and core)
- Can access /sys/devices/system/cpu/cpu0/cpufreq and /proc/cpuinfo on Android (as root)

Energy-aware scheduling

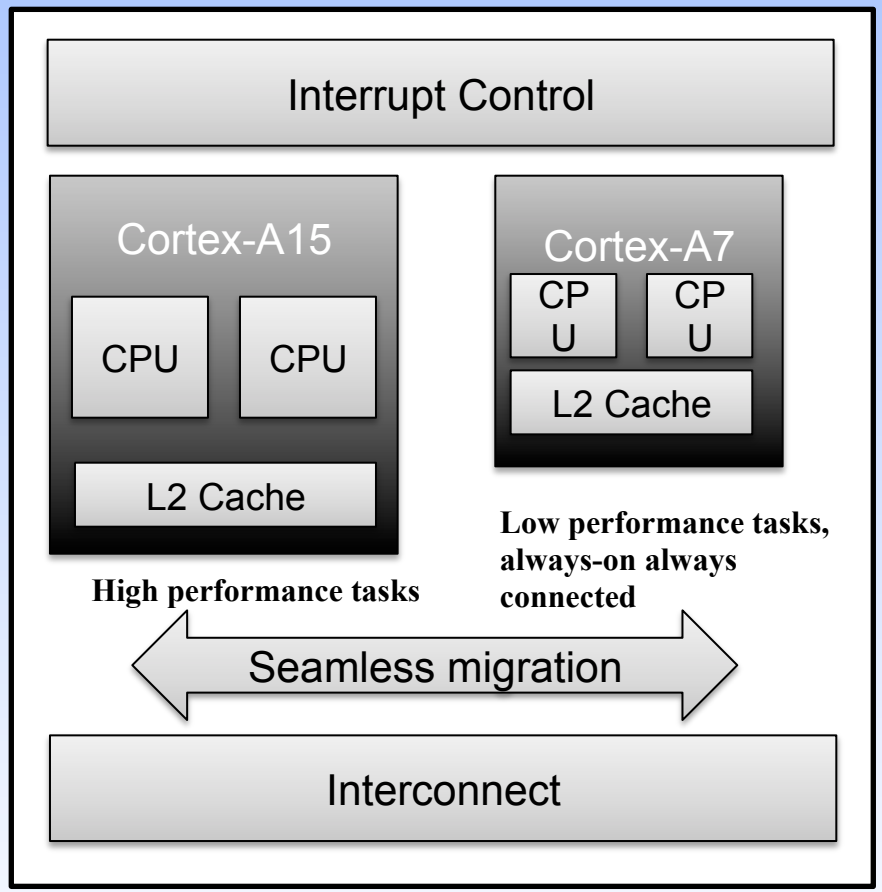
- Traditional schedulers do not take the multicore topology or energy issues into account

- Energy-aware aware schedulers
  - Distribute tasks across CPUs and cores to save energy
  - Underlying topology affects the strategy
  - For example: cluster tasks on a high performance CPU, then remaining CPUs can enter idle states

- Examples: ARM's big.LITTLE, NVIDIA battery saver core, …

ARM big.LITTLE

- The **ARM big.LITTLE** is a computing architecture that combines slow and low-power processor cores with faster and more power demanding cores
- This architecture is used, for example, by Samsung Galaxy Note 3 and S4 smartphones (Exynos 5 Octa).
- Extends DVFS with CPU migration.



big.LITTLE Models

- **Clustered model**, in which the OS scheduler observes one of the two processor clusters, and the scheduler transitions between the clusters based on the observed load.
- **In-kernel switcher** pairs a more powerful core with a less powerful core with the option of having many identical pairs on a chip. The active core is selected based on the load.
- **Heterogeneous multi-processing (MP)** enables the use of all physical cores simultaneously. High priority or computationally demanding threads are run by the powerful cores whereas low priority or less demanding threads are are run on the less powerful cores.

Big.LITTLE Experiments

- The framework allows seamless migration of tasks across the processor cores. The Cortex- A15-Cortex-A7 system is designed to migrate tasks between the processor clusters in less than 20 microseconds with 1GHz processors.
- ARM's energy efficiency comparison of Cortex-A15 and Cortex-A7 indicates significant power savings with a variety of benchmarks.
- For example, the Dhrystone benchmark gives energy efficiency benefit of 3.5x for A7 with performance benefit of 1.9x for A15. This motivates the use of the slower processor for lightweight tasks.

GPUs

- In modern smartphone designs graphics processing is typically offloaded to the GPU that has a high performance graphics processing pipeline.
- The four well-known mobile GPUs are:
  - Adreno GPU in the Qualcomm's Snapdragon line of SoCs.
  - PowerVR GPU used in TI's OMAP line of SoCs.
  - Mali in the ARM architecture.
  - GeForce ULP (ultra low-power) in the Tegra line of SoCs from Nvidia.
- GPU can be used for generic processing. For example, a Gabor face feature extraction algorithm was implemented with the Tegra GPU and OpenGL ES and the shader language.
  - The resulting GPU based algorithm achieved a 4.25 times speedup compared to the CPU based version

Modelling the CPU

- We outline the development of simple power models for the smartphone SoC and CPU based on utilization and linear regression. The development of our simple model proceeds in the following phases:
  - **Design of training phase** with different CPU loads. The loads should be realistic and reflect the real-life workloads on the device.
  - **Power measurement** of the training loads on the smartphone. An external power monitor tool is typically used for high accuracy measurements.
  - Creation of a **power model** for the CPU energy consumption.
- The training can model construction can happen offline, online or online with offline support.

Processor Power Model based on Counters

- Isci and Martanosi have proposed a power model for CPUs based on performance counters.
  - They correlate hardware performance counters and system log with total power measurements with an external power monitor to obtain a fine-grained view of energy consumption of the CPU. A similar approach can be used to model GPUs

$$P(C_i) = AccessRate(C_i) \times ArchitecturalScaling(C_i) \times MaxPower(C_i) + NonGatedClockPower(C_i).$$

$$P_{total} = \sum_{i=1}^{N} P(C_i) + Idle\ power.$$

C. Isci and M. Martonosi, “Runtime power monitoring in high-end processors: Methodology and empirical data,” in Proceedings of the 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003.

Single Core Regression Model

- Assuming a linear relationship between power consumption and CPU load, the linear regression model would be the following: $P_{cpu} = a \times U_{cpu} + b$
- Constants a and b are determined with regression
- The energy consumption of an application that uses the CPU in a dynamic manner can be determined with

$$E_{total} = \sum_{i} P_{cpu}^{i} \times \Delta T$$

- Given that CPU utilization is 20 % for a give duration T , we can determine the energy consumption with Etotal = (a20 + b)T.

Single Core Regression Model with DVFS

- DVFS can significantly improve energy efficiency.
- The effect of DVFS can be examined with the following simple equation:

$$E = P \times t + P_{idle} \times (t_{max} - t)$$

- where E gives the total energy of the workload, P is the average power over the workload, t is the execution time of the workload, P_{idle} is the idle power of the CPU, and t_{max} is the maximum running time of the workload over all frequencies.

A. Carroll and G. Heiser, “An analysis of power consumption in a smartphone,” in Proceedings of the 2010 USENIX conference.

Multicore Regression Model

- Yifan Zhang et al. studied the power modeling of multicore smartphone CPUs and they have identified that the traditional frequency and utilization based regression techniques are prone to errors in the multicore setting.

- Based on this observation they developed a new regression based power model for multicore CPUs based on the time spent in C states.

Y. Zhang, X. Wang, X. Liu, Y. Liu, L. Zhuang, and F. Zhao, “Towards better cpu power management on multicore smartphones,” in Proceedings of the Workshop on Power-Aware Computing and Systems, ser. HotPower ’13. New York, NY, USA: ACM, 2013.

Balancing Between Cores

- Given that we have a high performance core and a lower performance core for sensing tasks, it is clear that computationally heavy operations should be run on the high performance core.
- The low performance core, on the other hand, would be suitable for reading sensors and then handing over the data to the high performance core for intensive processing, such as speech recognition.
- Assuming that the low-cost processor is the most suitable for a specific computation stage i, we have the following bound for the slow-down of the stage:

$$s_i < \frac{P^M_{active} - P^M_{sleep}}{P^L_{active}} + \frac{E^{trans} / P^L_{active}}{T^M_i}$$

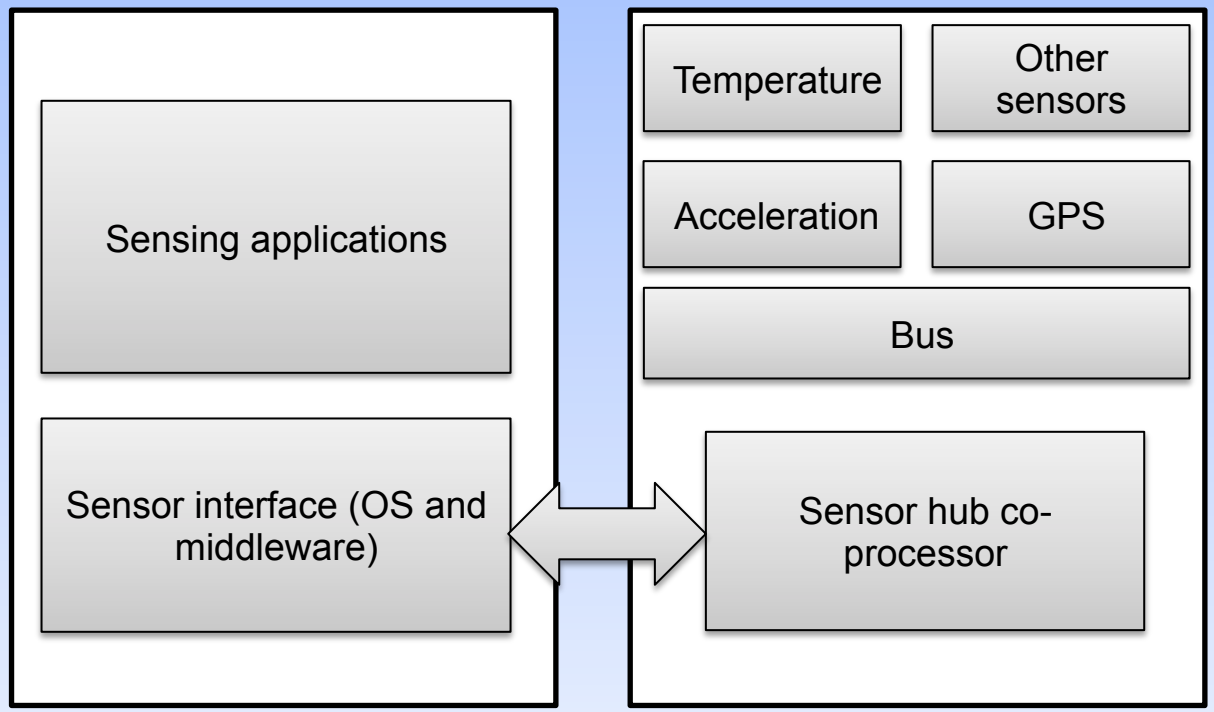
M.-R. Ra, B. Priyantha, A. Kansal, and J. Liu, “Improving energy efficiency personal sensing applications with heterogeneous multi-processors,” in Proceedings of the 2012 ACM Conference on Ubiquitous Computing.

Supporting Continuous Sensing

- Continuous sensing involves the constant monitoring of onboard sensors, such as acceleration, microphone or camera. Thus continuous sensing burdens the processor and uses a lot of energy.

- Galaxy S4 smartphone that has a hardware chip for aggregating and optimizing sensor data gathering and processing.

Sensorhub



Host CPU (maximize sleep)

Sensor hub (always-on)

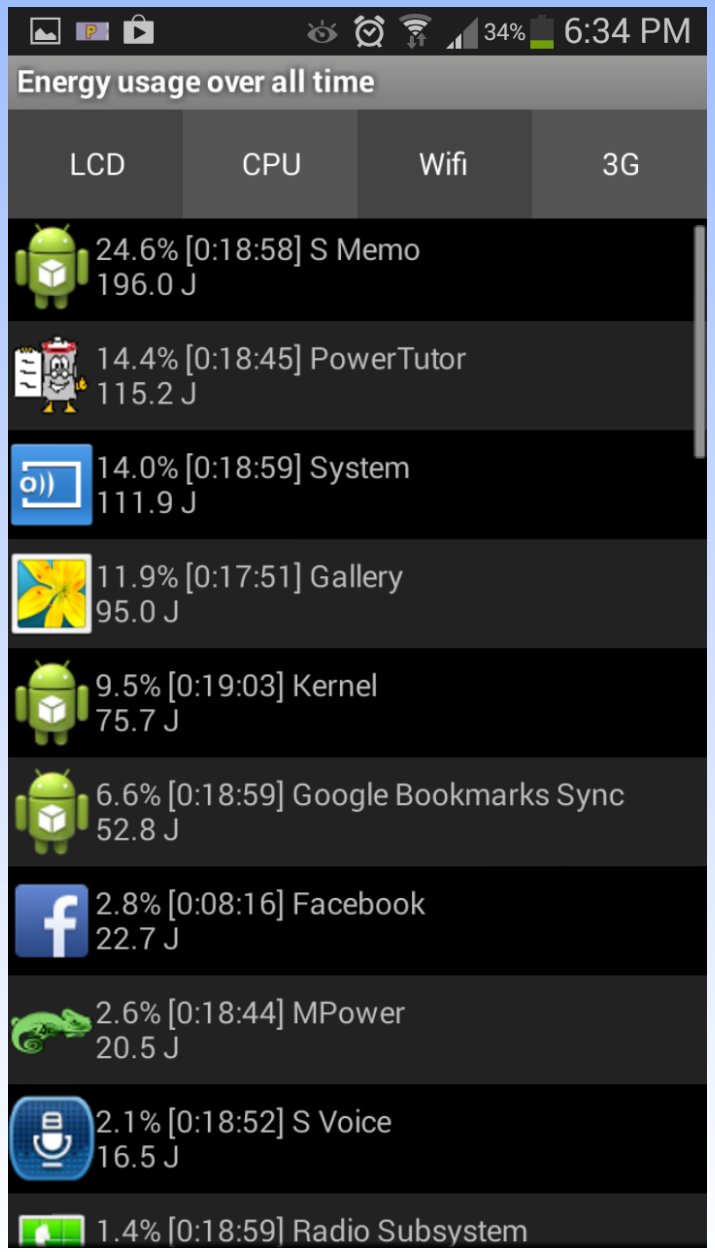
Smartphone Platform Overview

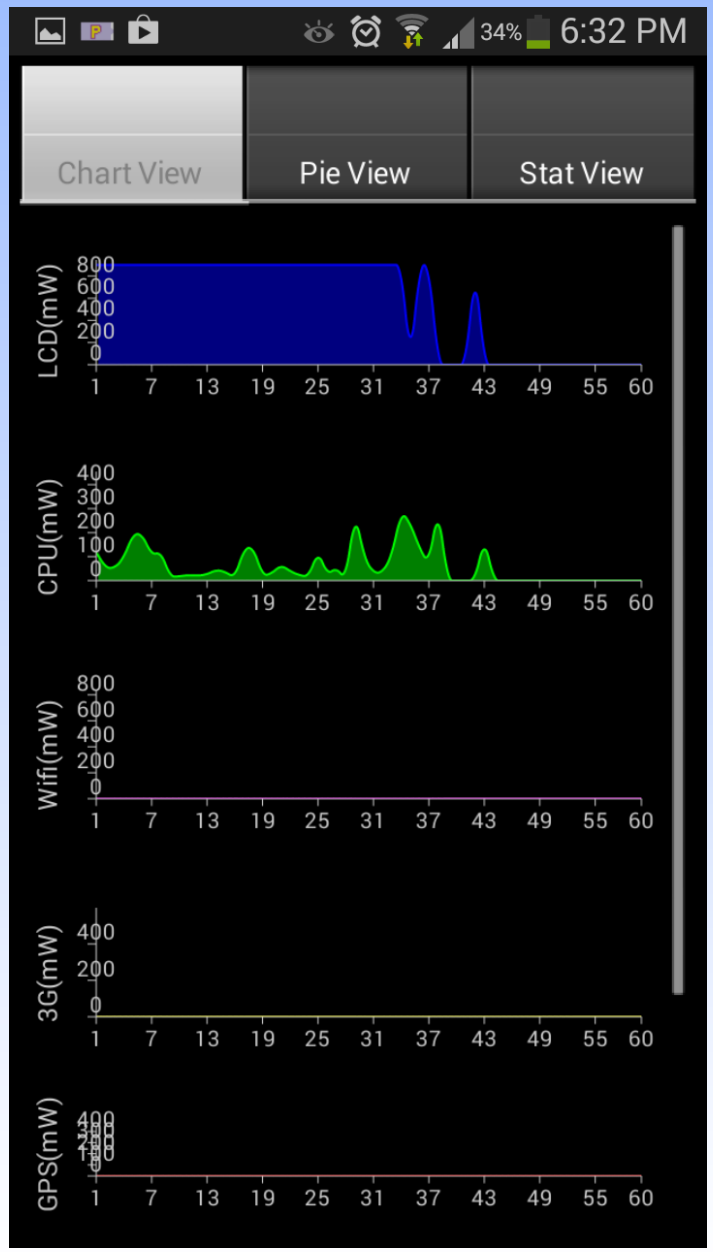
	Android Linux	iOS	Windows Phone 8	Firefox OS	Symbian Series 60
Low-level power management	Linux Power Management	iOS kernel	Windows NT	Linux Power Management (Gonk)	Kernel-side framework with power API (Power Manager), peripheral power on/ off
High-level power management	Java class PowerManager, JNI binding to OS. Key methods: goToSleep(long), newWakeLock(…), userActivity(long…) BatteryStats monitors energy consumption and uses device specific subsystem models.	I/O Framework	Run-time power management framework	Gaia (OS Shell) Gecko runtime: Power Management Web API is non-standard and reserved for pre-installed applications	Applications use domain manager that follows system-wide power-state policies. Nokia Energy Profiler API
Energy conservation patterns	Wake lock (partial, full) is used to ensure that device stays on. Methods: Create, acquire, release, sensor batching	Coding patterns, multitasking API (since iOS 4), push API, coalesced updates (since iOS7)	Multitasking API (tasks and push notification), asynchronous events	Asynchronous events (system messages) Resource lock in Power Management API	Active object, wakeup events, resource and domain manager
Policies	Wake lock specific flags and policies, system-wide power setting	Internal, multitasking API (since iOS4)	Internal	Gonk and Gecko level	Domain manager for system-wide and domain-wide policy. Domain-specific policies are possible
Battery information	The BatteryManager class contains strings and constants for different battery related notifications that applications can subscribe to, includes: battery level, temperature, voltage	iOS 3 and later: UIDevice Class allows to query/ subscribe battery info	Battery class provides the battery level, remaining operating time, and an event when battery is below 1%.	W3C Battery Status API	Battery API (charge level, external power). Nokia Energy Profiler

Power Profilers

Name / Authors	Year	Purpose
PowerScope	1999	Energy profiling of device and processes
Joule Watcher	2000	Fine-grained thread-level profiling
Nokia Energy Profiler	2006-2007	On-device standalone profiler
Shye et al.	2009	Energy profiling of device and components with a logger application
PowerTutor	2009	Hybrid profiler based on PowerBooter
PowerBooter	2009-2010	Short-term power model for components
BattOr	2011	Portable power monitor
Sesame	2011	Self-constructive on-device power model for device and components
PowerProf	2011	Self-constructive API-level power profiler
MobiBug	2011	Automatic diagnosis of application crashes
Carat	2012-2013	Application energy profiling and debugging
eProf	2012	Fine-grained power model for device, components and applications
DevScope	2012	Self-constructive power model for device and components
AppScope	2012	Fine-grained energy profiler for applications based on DevScope
eDoctor	2012	Automatic diagnosis of battery drain problems
V-Edge	2013	Self-constructive power model for device and components

PowerTutor

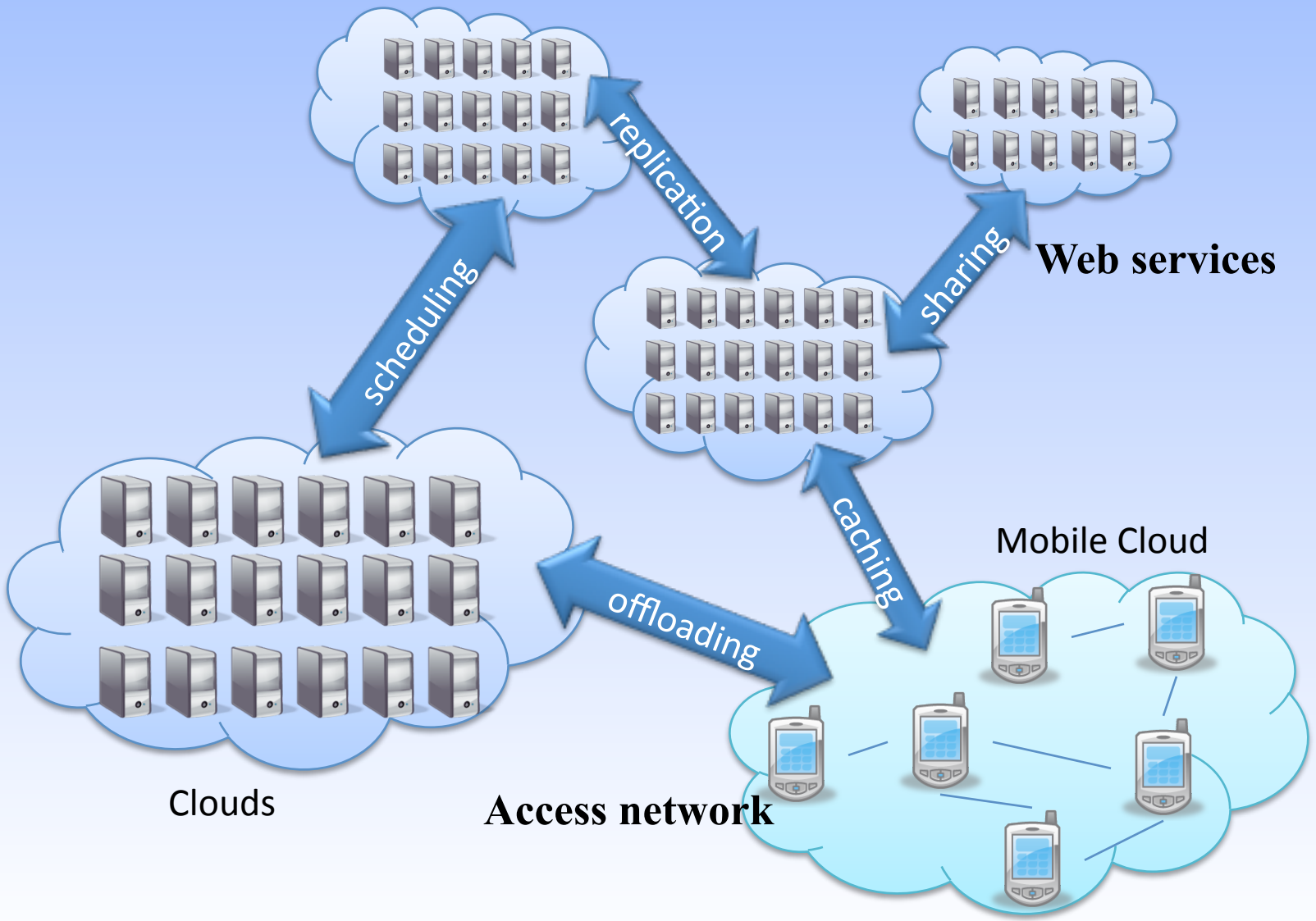

6:34 PM
Energy usage over all time
LCD
CPU
Wifi
3G
24.6% [0:18:58] S Memo
196.0 J
14.4% [0:18:45] PowerTutor
115.2 J
14.0% [0:18:59] System
111.9 J
11.9% [0:17:51] Gallery
95.0 J
9.5% [0:19:03] Kernel
75.7 J
6.6% [0:18:59] Google Bookmarks Sync
52.8 J
2.8% [0:08:16] Facebook
22.7 J
2.6% [0:18:44] MPower
20.5 J
2.1% [0:18:52] S Voice
16.5 J
1.4% [0:18:59] Radio Subsystem


6:32 PM
Chart View
Pie View
Stat View
LCD(mW)
CPU(mW)
Wifi(mW)
3G(mW)
GPS(mW)

Computation Offloading

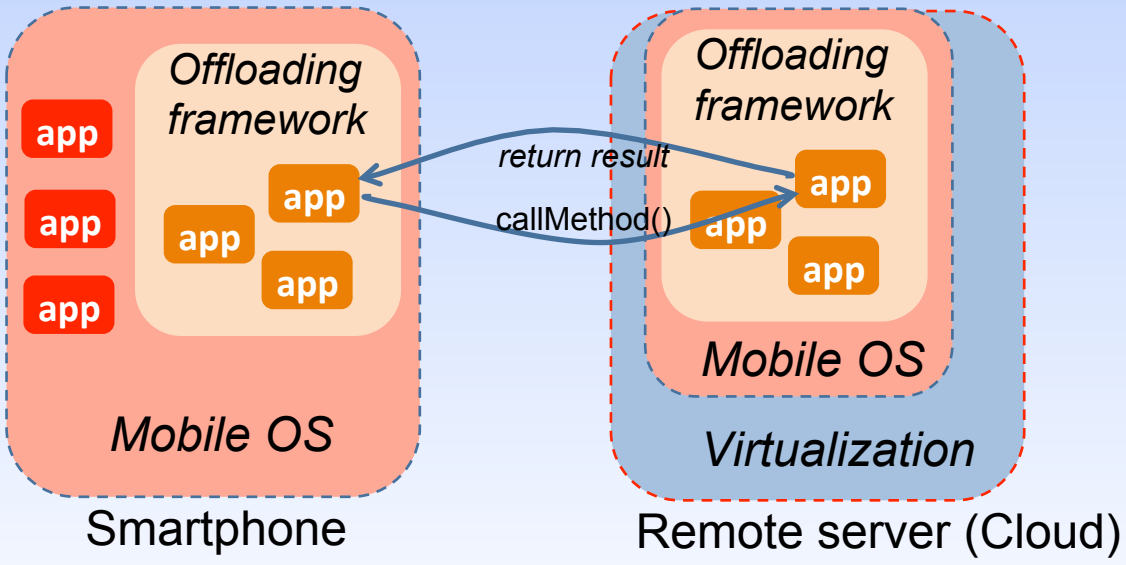
- Based on material contributed by Matti Siekkinen, Eemil Lagerspetz and Juhani Toivonen

Environment



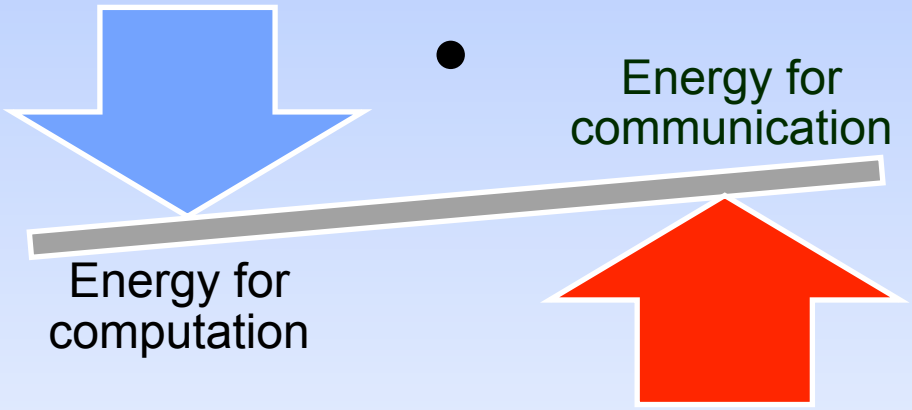
What is offloading?

- Consider apps designed and implemented to be run on standalone mobile OS
- Execute part of the application code in a remote machine



Offloading work to save energy

- Main objective is to save energy
  - Tradeoff: less computing with some extra communication
  - Transfer state back and forth between smartphone and cloud



- Often involves dynamic decision making because the tradeoff is not constant
- May also improve other performance metrics (response time)
  - High performance computing in cloud

Offloading

- The offloading of a mobile computing task is a trade-off between the energy used for local processing and the energy required for offloading the task, uploading its data, and downloading the result, if necessary.

- One can express the offloading energy trade-off as follows:
  - **Etrade = Elocal − Edelegate > 0**, where Elocal is the energy used for complete local execution, and Edelegate is the energy used if the task is offloaded from the perspective of the mobile device.

- If Etrade is greater than zero, then there is an energy benefit for delegating the task to the cloud.

Offloading frameworks

- Most rely on having source code available
  - MAUI at Mobisys'10
    - *Cuervo et al.* from Duke, UMass, UCLA, MSR
  - Cuckoo at MobiCASE'10
    - *Kemp et al.* from Vrije Universiteit
  - ThinkAir at Infocom'12
    - Kosta et al. from DT Labs, Cambridge, Nottingham, Huawei
- One modifies the underlying system (VM)
  - CloneCloud at EuroSys'11
    - *Chun et al.* from Intel Labs, Princeton
- Testing with computationally intensive apps delivers impressive results
  - 45% energy savings for chess AI [MAUI]
  - 20x speedup and energy savings for a large image search [CloneCloud]

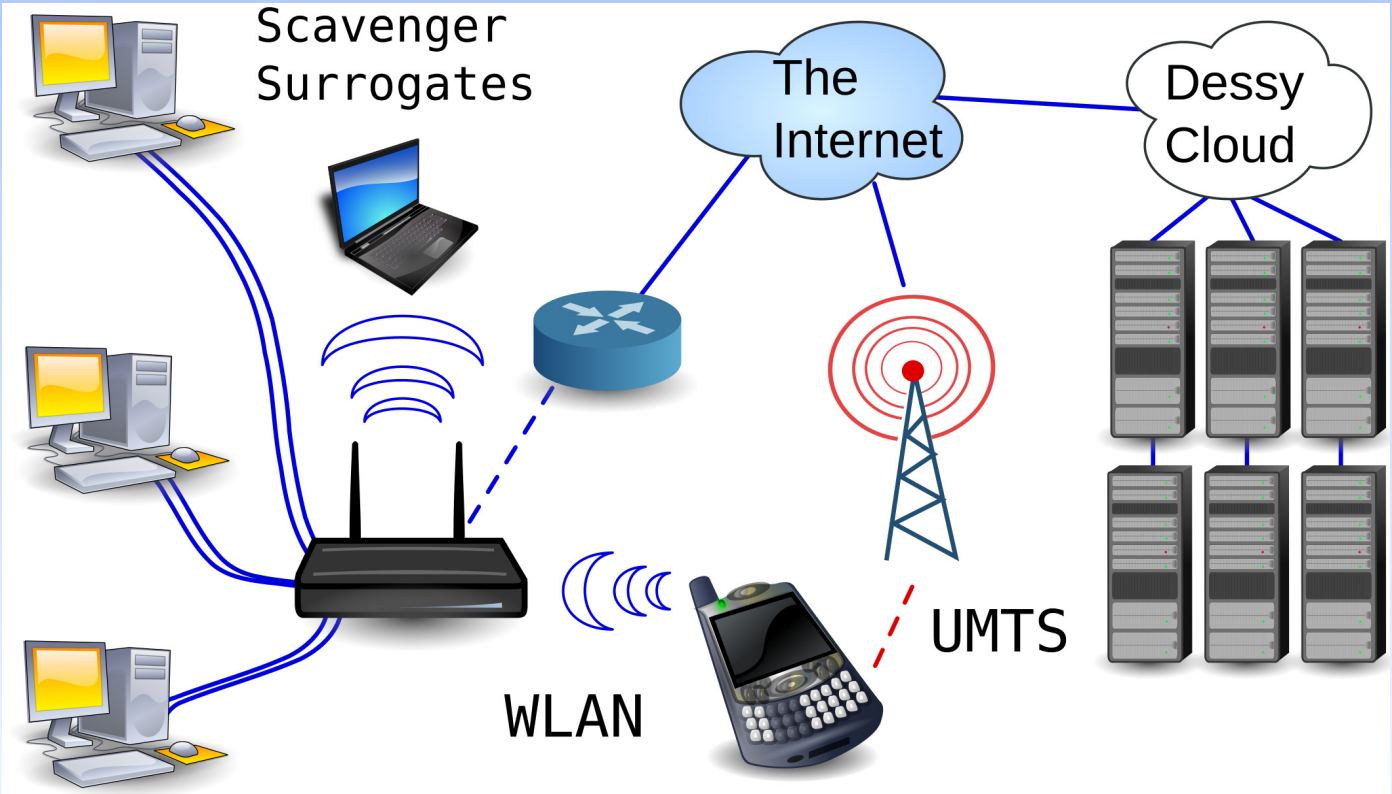
What can be offloaded?

- Content processing and transformations
  - Example: Javascript processing in OperaMini
- Completion notification and mobile push
- Application execution
  - Google docs, Windows Office
- Connection management
  - BitTorrent!
  - Large downloads
- Speech recognition
  - Siri
- Positioning (A-GPS)
- NVIDIA cloud enhanced 3D graphics

Example of Offloading: Indexing

- Implemented with Dessy: mobile desktop search
- To offload indexing
  - Transmit entire file to the cloud service
  - Wait for response
  - Receive file summary
- High energy savings can be obtained when offloading CPU-intensive tasks
- With N900 and WLAN 700kB/s: 96.5% savings!
  - 200 000 words, 1 MB file
  - With WLAN 100kB/s this is reduced to 83.7%

Dessy Offloading



Cloudlets

Cloudlet architecture from CMU consists of customized ephemeral virtual machines with soft state, and a platform for running them

Deploy applications near the users to avoid latency and bandwidth problems

Facilitates elastic and mobile execution of network components in base stations

Network support for computation offloading?