Introduction to power saving in mobile devices

Outline

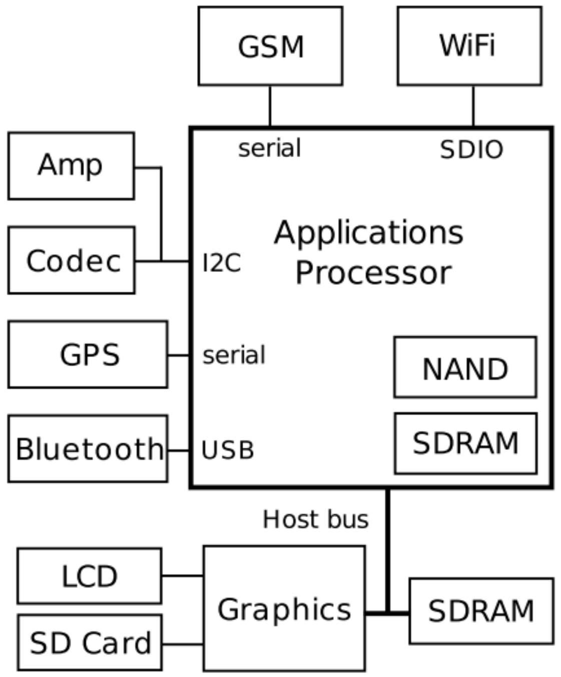
- How is energy spent in a smartphone?
- More energy-efficient CPUs
- Energy Aware Scheduling
- Mechanisms at the OS-Application level

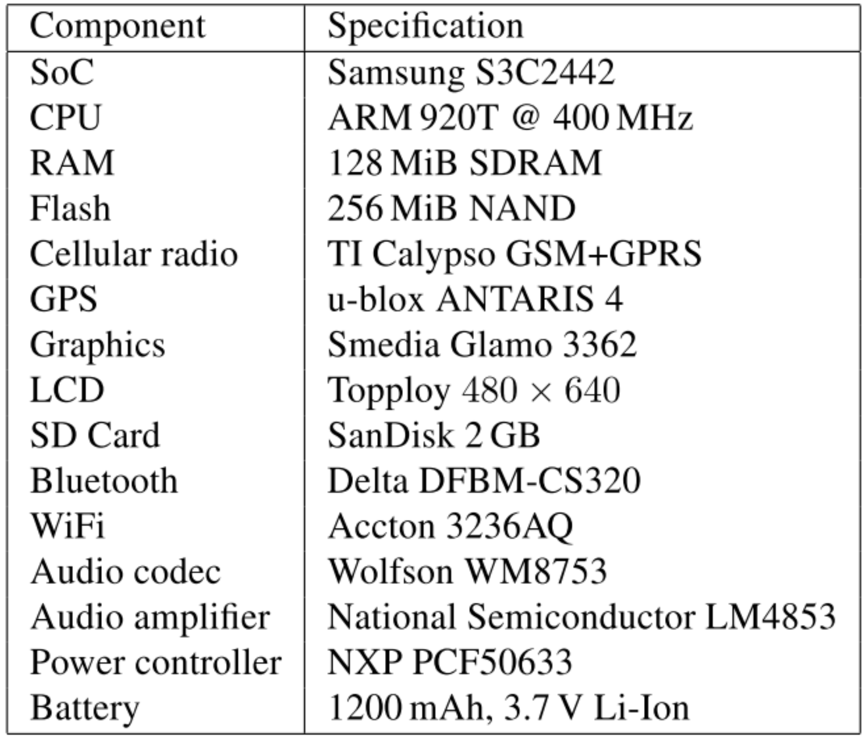
How is energy spent in a smartphone?

- Limited battery life is one of the main factors adversely affecting the mobile experience of smartphone users
- The first step to improve the battery life of a smartphone is to understand how energy is spent
- We will see two studies:
  - “An Analysis of Power Consumption in a Smartphone”, A. Carrol, G. Heiser, Proceedings of the USENIX conference
  - “Smartphone Energy Drain in the Wild: Analysis and Implications”, Chen et al., Proceedings of the ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems

An Analysis of Power Consumption in a Smartphone

- Device under test: Openmoko Neo 2.5G smartphone
- Schematics freely available





Component	Specification
SoC	Samsung S3C2442
CPU	ARM 920T @ 400 MHz
RAM	128 MiB SDRAM
Flash	256 MiB NAND
Cellular radio	TI Calypso GSM+GPRS
GPS	u-blox ANTARIS 4
Graphics	Smedia Glamo 3362
LCD	Topploy 480 × 640
SD Card	SanDisk 2 GB
Bluetooth	Delta DFBM-CS320
WiFi	Accton 3236AQ
Audio codec	Wolfson WM8753
Audio amplifier	National Semiconductor LM4853
Power controller	NXP PCF50633
Battery	1200 mAh, 3.7 V Li-Ion

An Analysis of Power Consumption in a Smartphone

- To calculate the power consumed by any component, both the supply voltage and current must be determined
    - current: sense resistors inserted on the power supply rails of components
    - supply voltages: measured relative to ground from the component side of the resistors

- Benchmarks based on real usage scenarios
    - low-interactivity applications (e.g. music playback): simply launched from the command line
    - interactive applications, such as web browsing: a trace-based approach

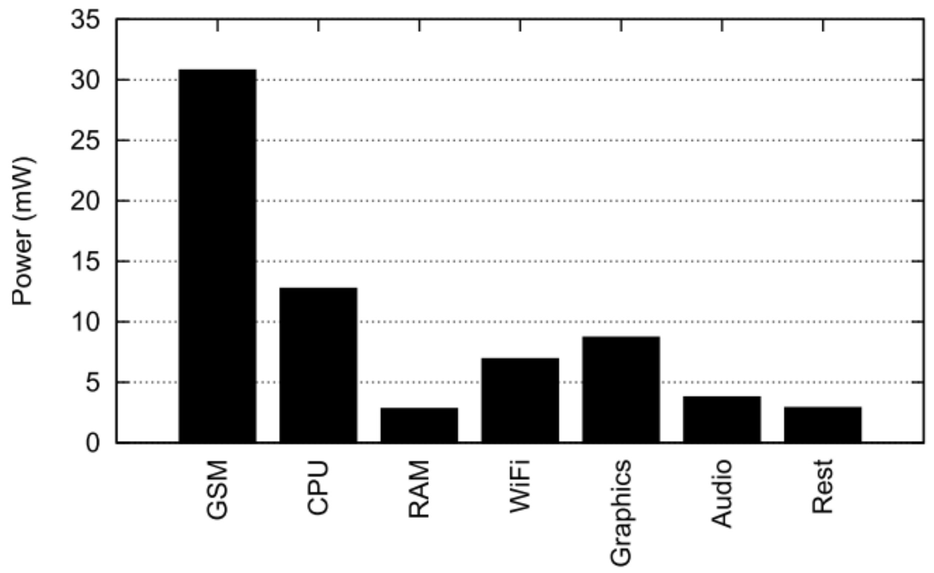
An Analysis of Power Consumption in a Smartphone

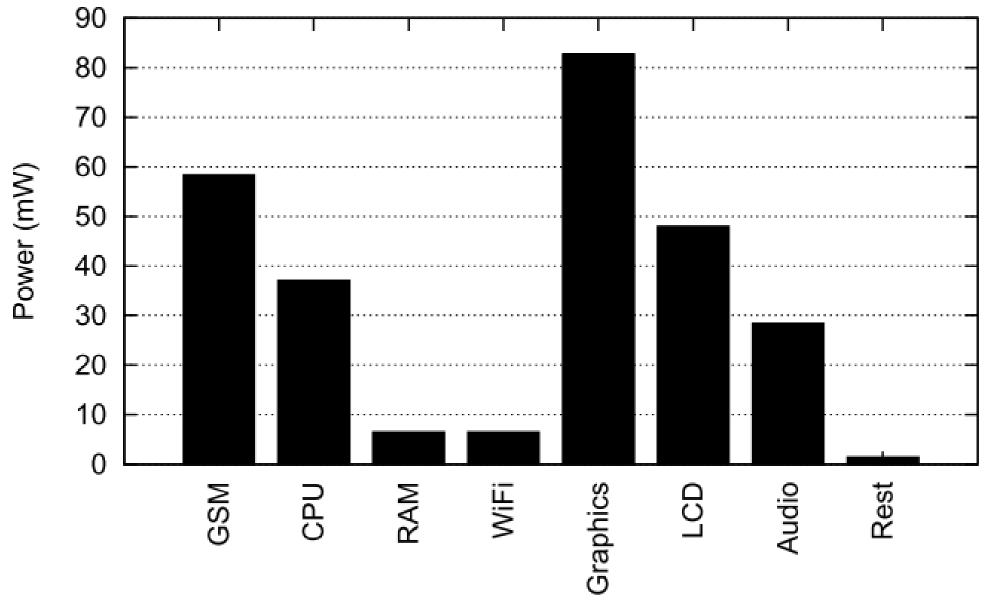
- Baseline power state of the device, when no applications are running
- A mobile phone will spend a large amount of time in a state where it is not actively used
- Two different cases to consider:
  - *Suspended*:
    - Application processor is in a lower power state
    - Communications processor low level of activity, it must remain connected to the network to receive calls, SMSs, etc.
    - Android aggressively suspends to RAM during idle periods: necessary state is written to RAM and the devices are put into low-power sleep modes
  - *Idle*: It is fully awake (not suspended) but no applications are active

An Analysis of Power Consumption in a Smartphone

- *Suspeded*:
  - GSM module highest consumption
  - RAM negligible despite mantaining full state
  - total power: 69 mW

- *Idle*:
  - measured with backlight off
  - display-related subsystems highest consumption
  - total power 269 mW

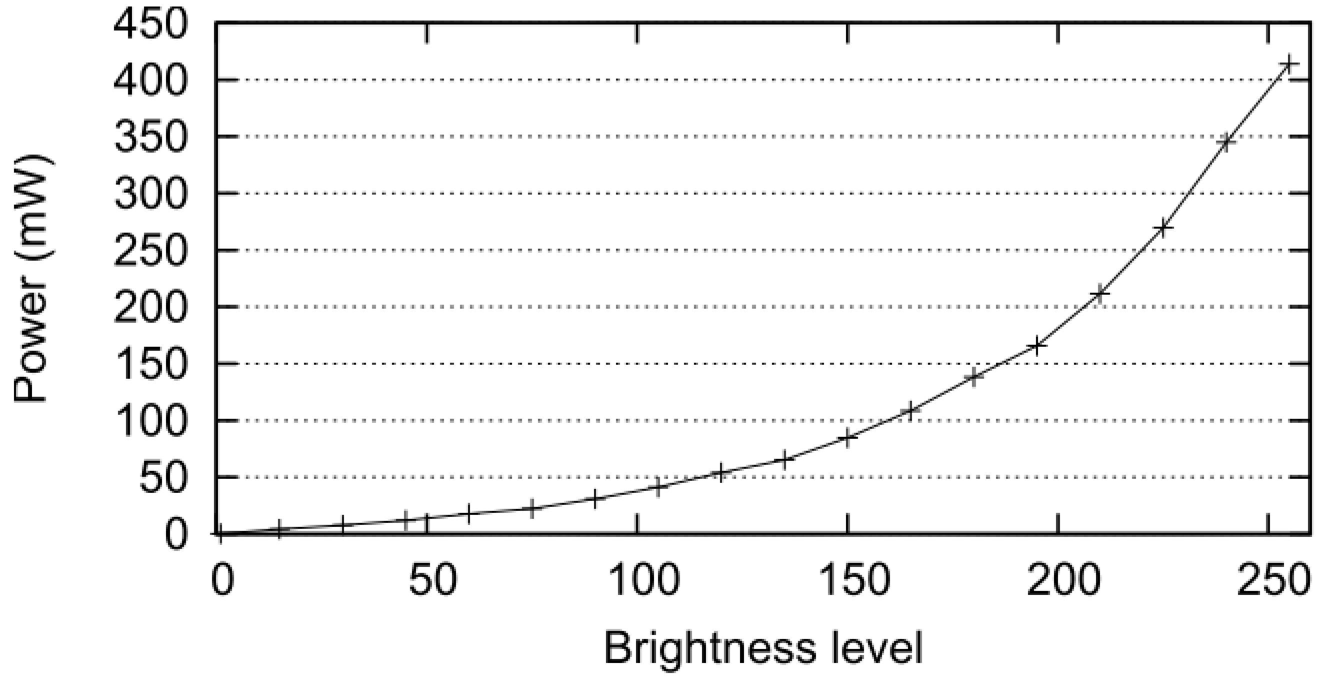




An Analysis of Power Consumption in a Smartphone

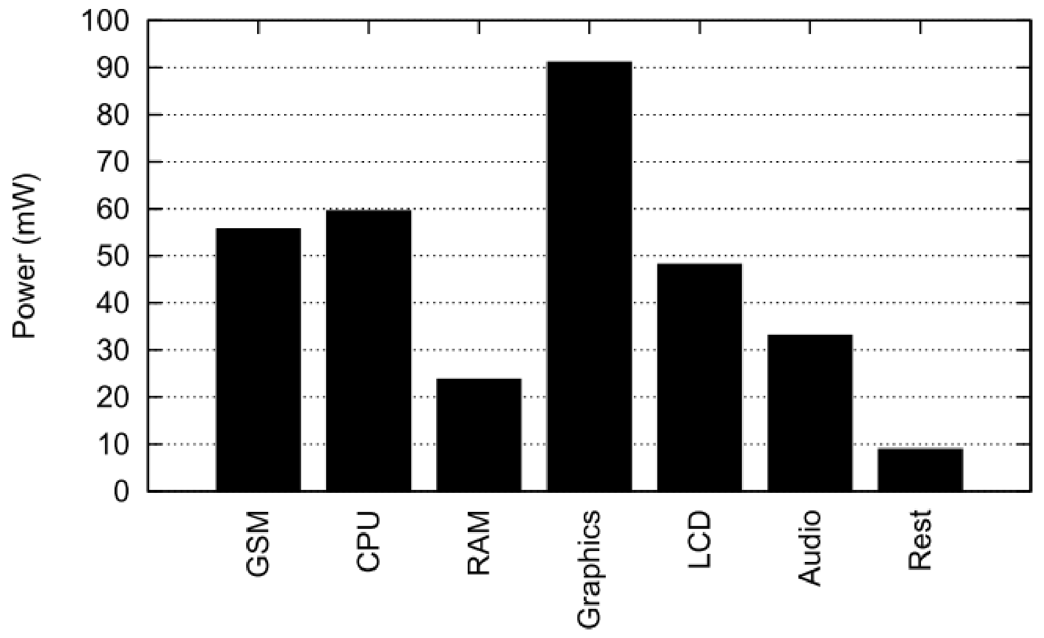
- Brightness level: an integer between 1 and 255
- Consumption: min 7.8mW, max 414mW
  - half-scale corresponds to a brightness level of 143, consumption: 75mW

Content displayed on the LCD affected its power consumption: 33.1mW for a completely white screen, and 74.2mW for a a black screen



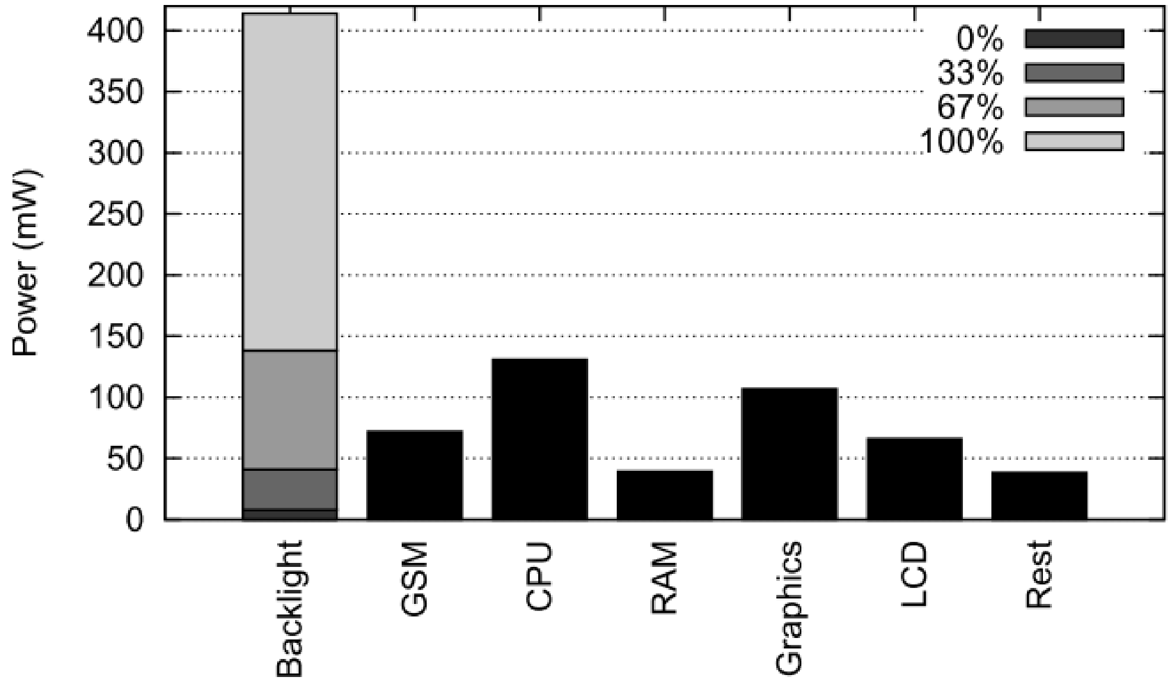
An Analysis of Power Consumption in a Smartphone

- Audio playback, total power 320 mW
- Screen turned off
- Audio subsystem accounts for ~12% of power consumed
- Maintaining a connection to the GSM network requires significant power
- MP3 file is loaded from the SD card, the cost of doing so is negligible at < 2% of total power



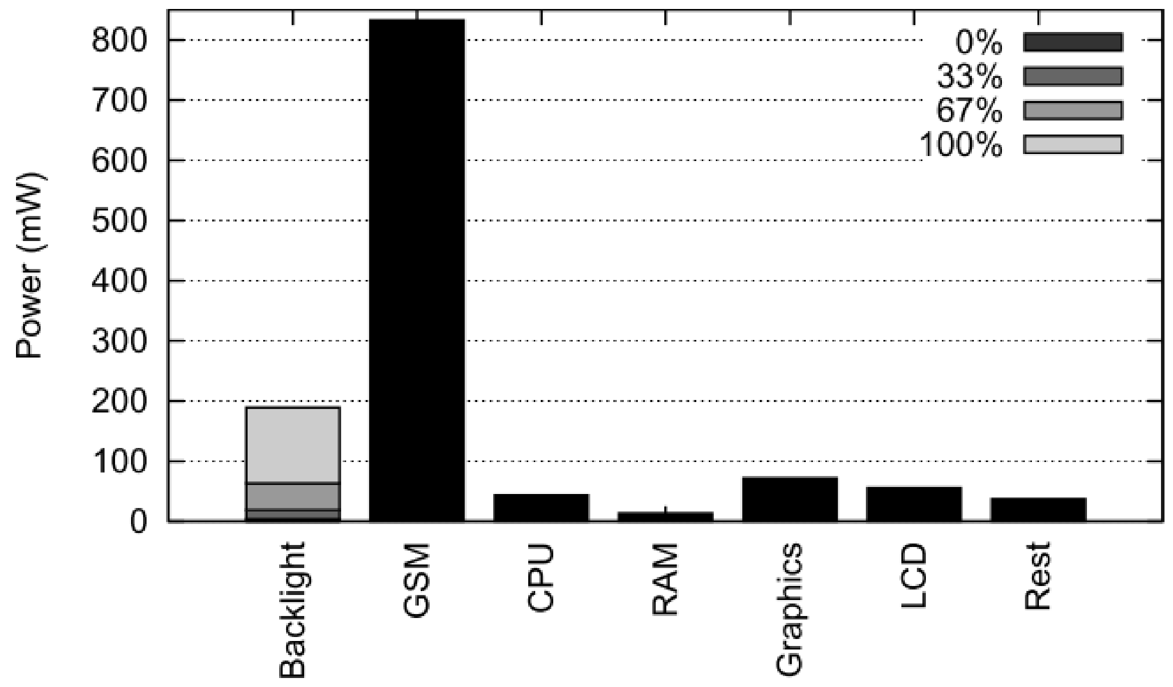
An Analysis of Power Consumption in a Smartphone

- Video playback, total power 453 mW (no backlight)
- CPU is the biggest single consumer of power (other than backlight)
- Display subsystems still account for a large fraction
- The energy cost of loading the video from the SD card is negligible



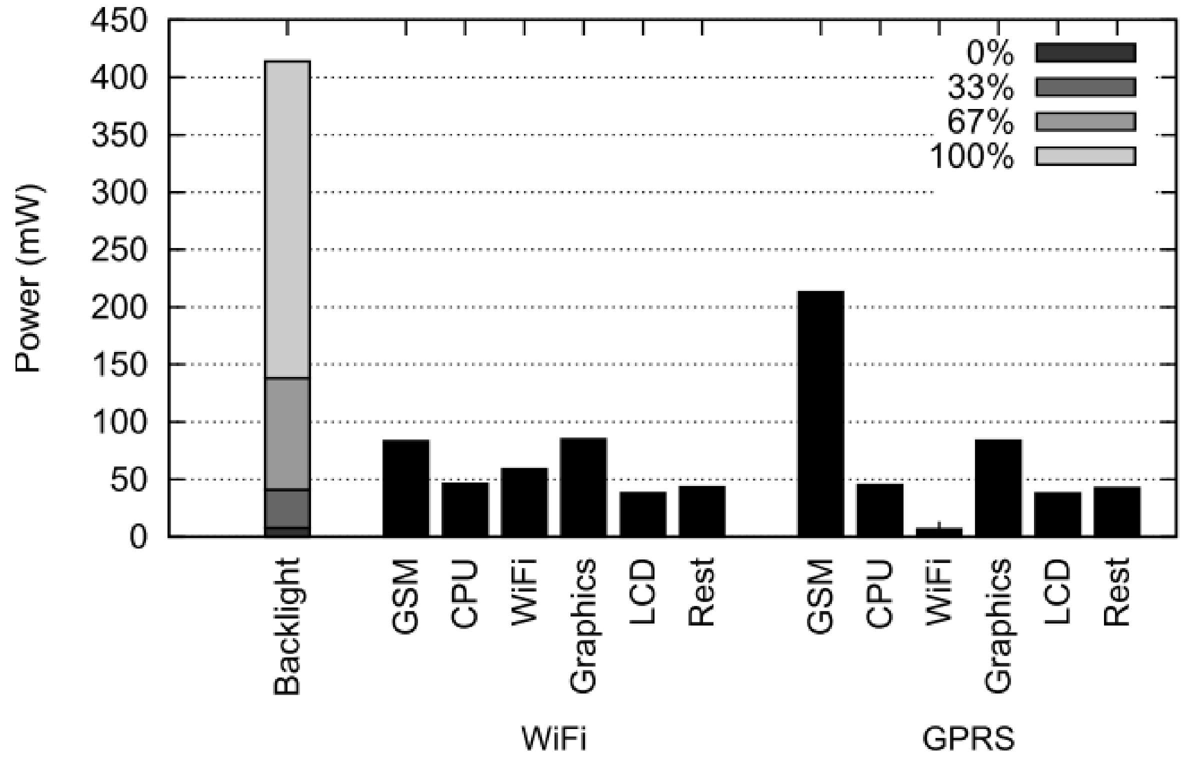
An Analysis of Power Consumption in a Smartphone

- Phone call, total power 1054 mW, excluding backlight
- Backlight turned off after few seconds



An Analysis of Power Consumption in a Smartphone

- Web browsing, total power 352 mW for WiFi, 429 mW for GPRS
- GPRS consumes more power than WiFi by a factor of 2.5



An Analysis of Power Consumption in a Smartphone

- Where does the energy go?
  - Excluding intensive cellular communication, brightness is the largest factor in power consumption
  - Just maintaining a connection with the network consumes a significant fraction of power. In a call, GSM consumes more than 800mW, the single largest power drain
  - Static contribution to system power consumption is substantial

- **Aggressive backlight dimming can save lots of energy**
- **Inclusion of ambient light and proximity sensors in mobile devices to assist with selecting appropriate brightness**
- **Selecting color scheme can reduce energy consumption**

- **Wi-Fi needs less power than cellular, use Wi-Fi whenever possible**

Shut down unused components and disable their power supplies (where possible)

Smartphone Energy Drain in the Wild: Analysis and Implications

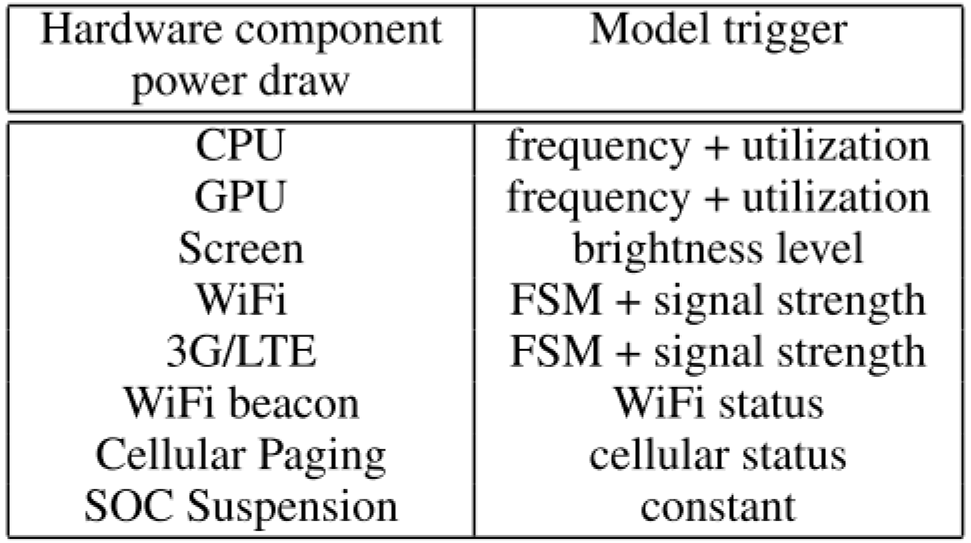
- More recent work
- Power consumption is measured according to a less accurate method, but in a more realistic scenario

Smartphone Energy Drain in the Wild: Analysis and Implications

- Basic ideas:
  - Power model for estimating energy drain of the hardware components + apps and services running on the smartphones in the wild
  - Triggers for driving FSM transitions on stock Android phones (not easy on non rooted devices)
    - On a rooted phone this could be easily done by instrumenting all relevant OS calls
  - Free app that performs low overhead logging, collect all the triggers needed to drive the power model
  - ~1500 users, all equipped with the same devices (Samsung S3 and S4)

Smartphone Energy Drain in the Wild: Analysis and Implications

- Modeling



Hardware component power draw	Model trigger
CPU	frequency + utilization
GPU	frequency + utilization
Screen	brightness level
WiFi	FSM + signal strength
3G/LTE	FSM + signal strength
WiFi beacon	WiFi status
Cellular Paging	cellular status
SOC Suspension	constant

$$P_{CPU} = P_{B,N_c} + \sum_i^{N_c} u_i \cdot P_\Delta(f_i)$$

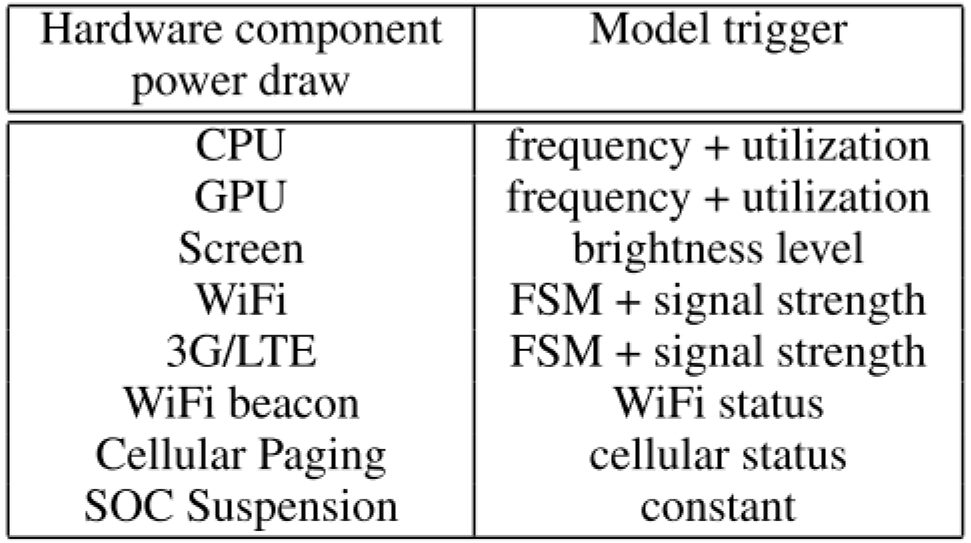
$P_{B,Nc}$ is the baseline CPU power with Nc enabled cores
$P_\Delta(f_i)$ is the power increment of core *i* at frequency *f_i*,
and *u_i* is its utilization.

Measured in lab, always the same devices

At runtime, the energy of each app was estimated using the logged CPU frequency and utilization

Smartphone Energy Drain in the Wild: Analysis and Implications

- Modeling

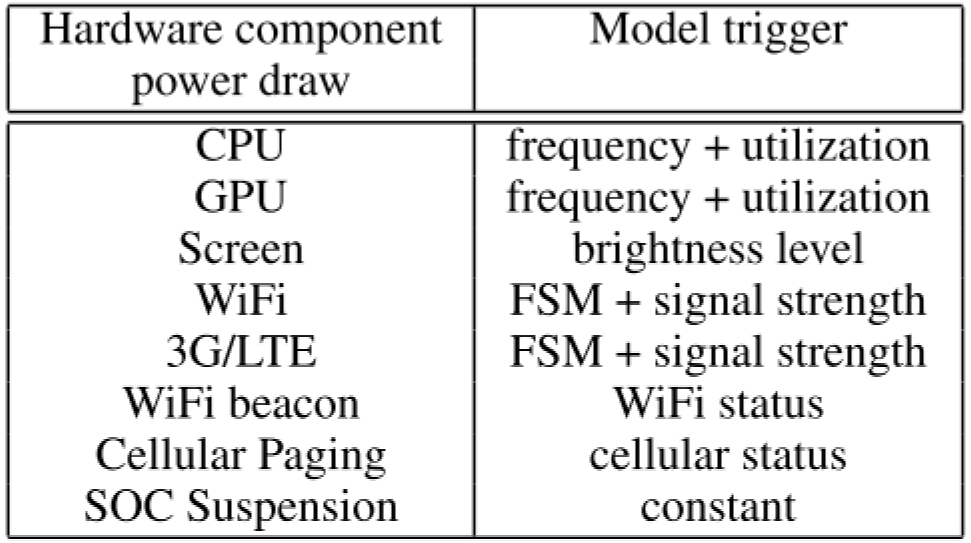


Hardware component power draw	Model trigger
CPU	frequency + utilization
GPU	frequency + utilization
Screen	brightness level
WiFi	FSM + signal strength
3G/LTE	FSM + signal strength
WiFi beacon	WiFi status
Cellular Paging	cellular status
SOC Suspension	constant

Similar approach also for GPU

Smartphone Energy Drain in the Wild: Analysis and Implications

- Modeling



Hardware component power draw	Model trigger
CPU	frequency + utilization
GPU	frequency + utilization
Screen	brightness level
WiFi	FSM + signal strength
3G/LTE	FSM + signal strength
WiFi beacon	WiFi status
Cellular Paging	cellular status
SOC Suspension	constant

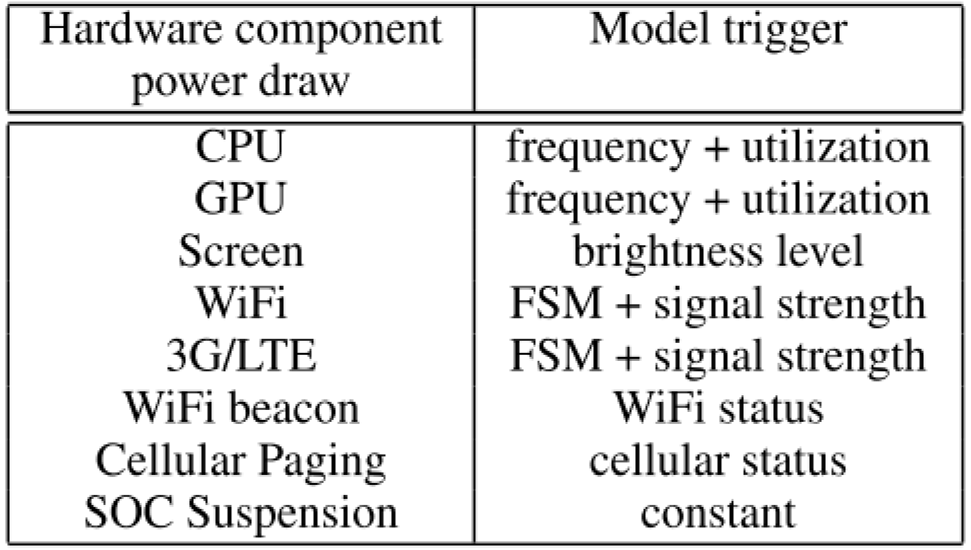
Consumption of different brightness level
Measured in lab
Displayed content not considered

Smartphone Energy Drain in the Wild: Analysis and Implications

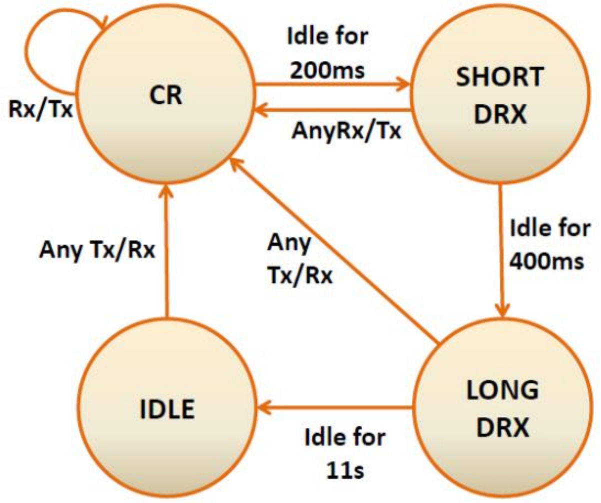
- Modeling

CR: When the UE sends or receives any data, the interface enters the *Continuous Reception* (CR) state and consumes high power

Short DRX: After the UE finishes data transfer and becomes idle for 200ms, the interface will enter the Short DRX state; little power but wakes up frequently to check for incoming traffic

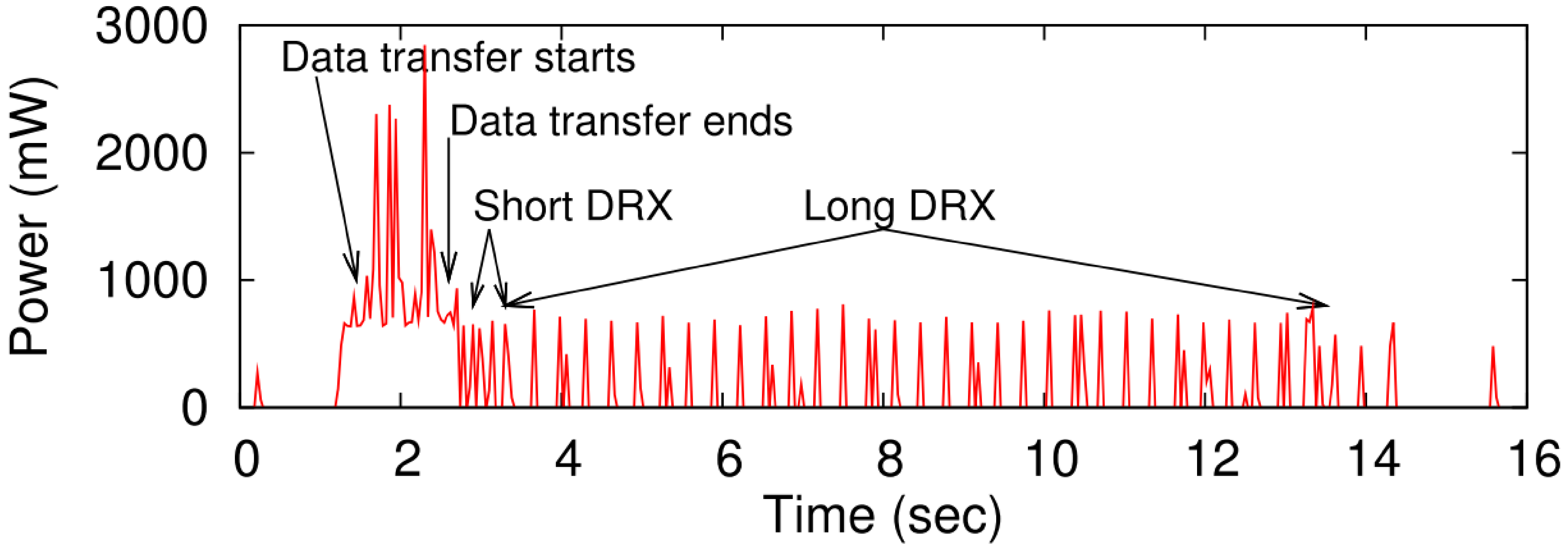


Hardware component power draw	Model trigger
CPU	frequency + utilization
GPU	frequency + utilization
Screen	brightness level
WiFi	FSM + signal strength
3G/LTE	FSM + signal strength
WiFi beacon	WiFi status
Cellular Paging	cellular status
SOC Suspension	constant



Long DRX: interface enters the Long DRX state after staying in Short DRX for 400ms without receiving any data; wakeup interval becomes longer

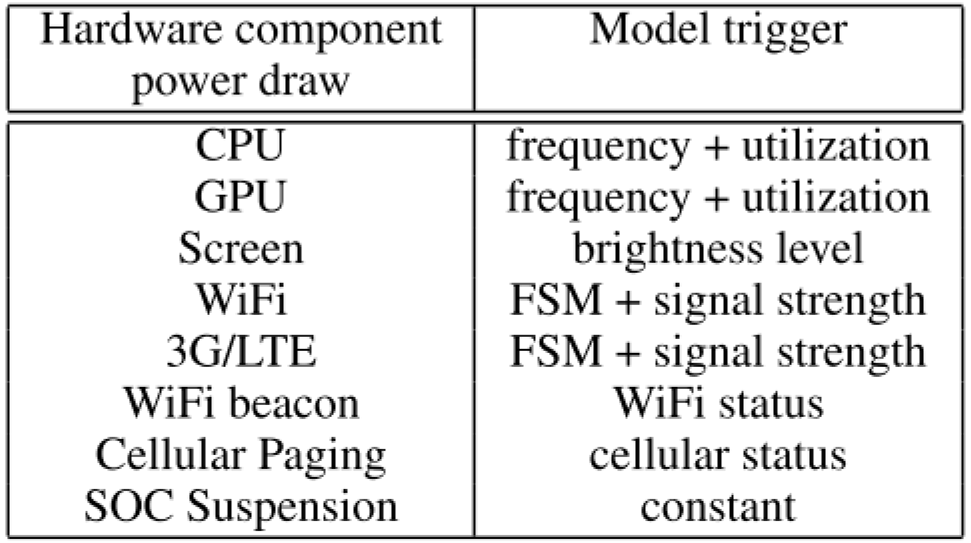
IDLE: The interface is in idle state when the UE does not send or receive any data; little power, periodically wakes up to check whether there are incoming data buffered at the network



Data transfer starts
Data transfer ends
Short DRX
Long DRX
Power (mW)
Time (sec)
0
1000
2000
3000
0
2
4
6
8
10
12
14
16

Smartphone Energy Drain in the Wild: Analysis and Implications

- Modeling



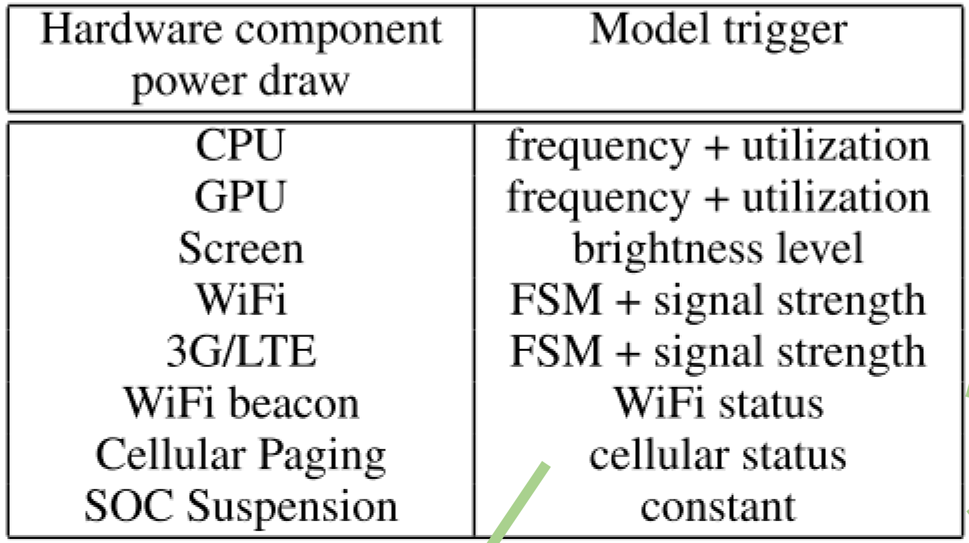
Hardware component power draw	Model trigger
CPU	frequency + utilization
GPU	frequency + utilization
Screen	brightness level
WiFi	FSM + signal strength
3G/LTE	FSM + signal strength
WiFi beacon	WiFi status
Cellular Paging	cellular status
SOC Suspension	constant

Approach similar to LTE

Smartphone Energy Drain in the Wild: Analysis and Implications

- Modeling

If WiFi radio is associated with an AP and in power saving mode, the WiFi radio wakes up at fixed intervals to receive beacons from the AP. Modeled as a constant current: 1.1 mA



Hardware component power draw	Model trigger
CPU	frequency + utilization
GPU	frequency + utilization
Screen	brightness level
WiFi	FSM + signal strength
3G/LTE	FSM + signal strength
WiFi beacon	WiFi status
Cellular Paging	cellular status
SOC Suspension	constant

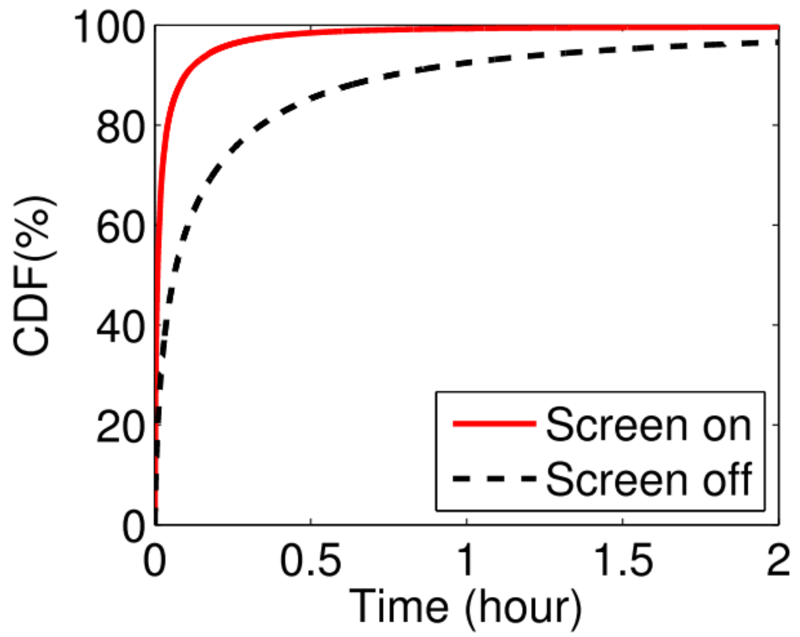
WiFi radio needs to search all possible channels until it finds one. For example, the 2.4 GHz band (802.11/bg) has 11 channels and the 5GHz (802.11ac) band has 22 channels. WiFi scanning on these two phones has a duration of 3.4 seconds and average power draw of 64 mA

On a cellular network, the base station periodically broadcasts a message during the 3G/LTE Idle state to signal incoming downlink data or voice call or SMS. A power spike on the phone modem every 1.28s. Average current: 8.3 mA on S3 and 2.3 mA on S4.

When the CPU and other hardware components are offline, the entire SOC is suspended and draws a constant current: 3.8 mA and 5.1 mA for Galaxy S3 and S4, respectively

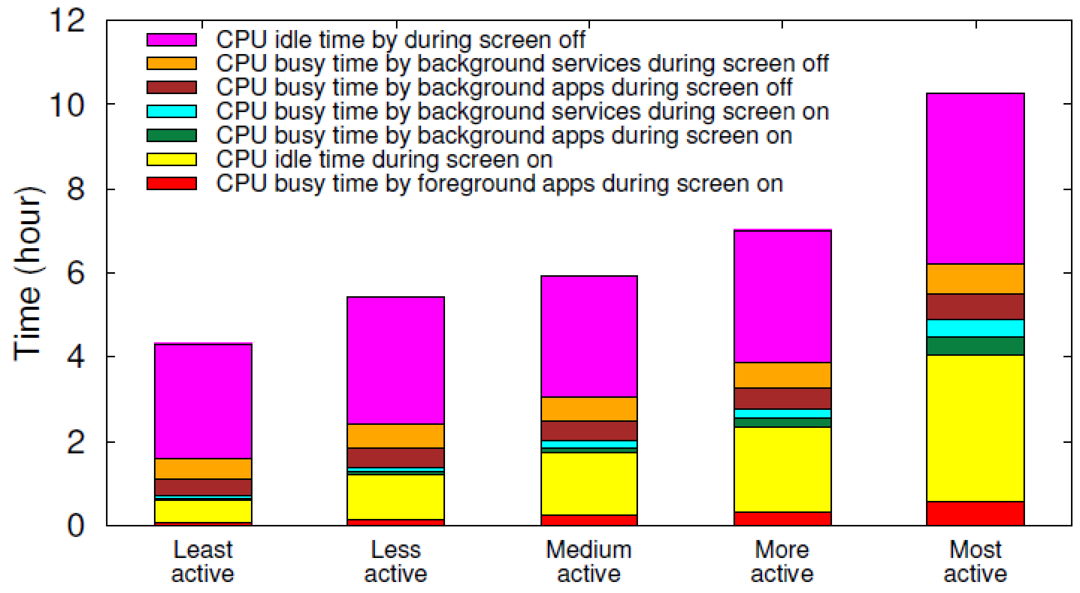
Smartphone Energy Drain in the Wild: Analysis and Implications

- Usage traces have been collected by means of an app
- 1520 users, average 32 days per user, total: 49k days, 56 countries, 191 MNOs
- Results: screen-on/screen-off periods

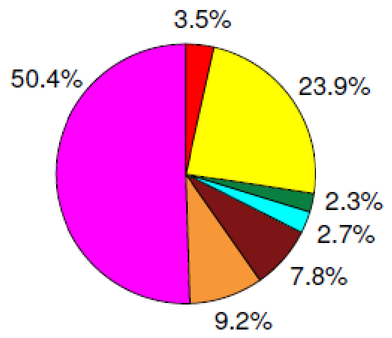


CDF of all screen off/on intervals for all users and all days

Smartphone Energy Drain in the Wild: Analysis and Implications



(a) Average daily CPU time breakdown of 5 groups of the 1520 users.

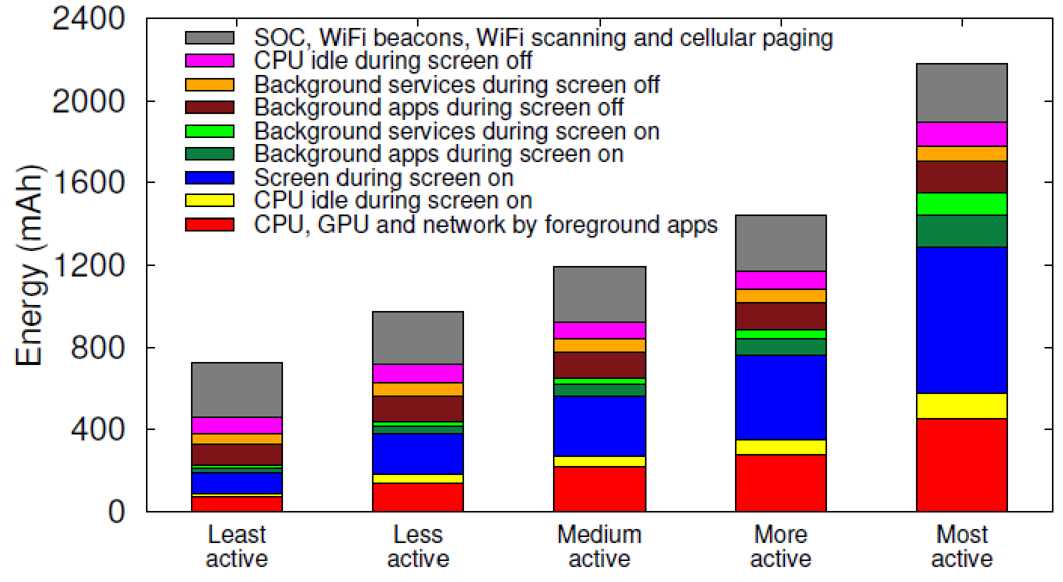


(b) Daily CPU time percentage breakdown, average over all users.

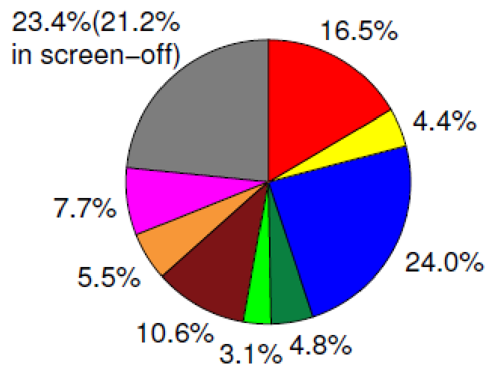
Figure 5: Daily CPU time breakdown.

- Users divided into 5 groups by total screen-on time
- In foreground, except for games that can be keeping CPU busy while the user is idle, **most apps are not using the CPU: users are idle for a significant portion of the time**, e.g., reading web pages, emails, or thinking.
- **A huge portion of screen-off CPU time is spent idle (50.4%)**

Smartphone Energy Drain in the Wild: Analysis and Implications



(a) Average daily energy drain breakdown of 5 groups of the 1520 users.

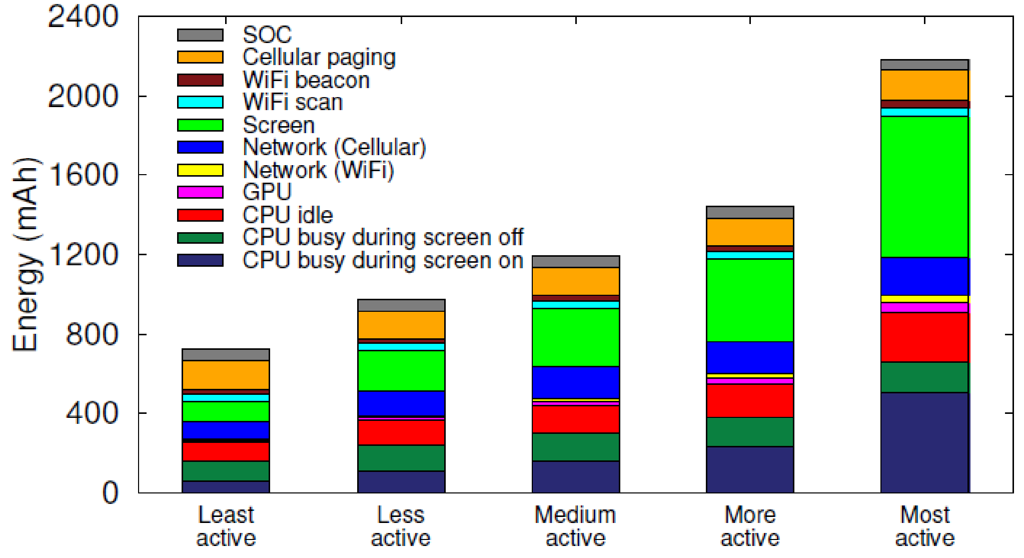


(b) Daily energy percentage breakdown, averaged over all users.

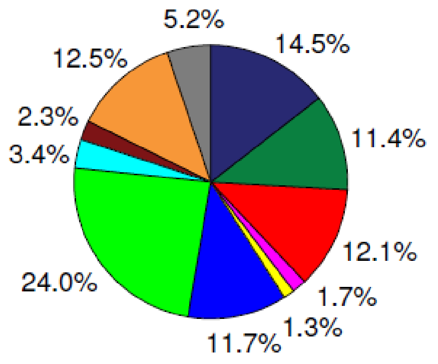
Figure 6: Daily energy breakdown by activities.

- **Energy breakdown by activities**
- **45.0% of the total energy drain in a day occurs during screen-off periods**
- 21.2% of the total energy drain due to SOC base power, WiFi beacon, WiFi scanning and cellular paging activities during screen-off periods does not contribute to any useful work
- Out of the 55.0% energy incurred during screen-on periods, 24.0% is spent in screen

Smartphone Energy Drain in the Wild: Analysis and Implications



(a) Average daily energy drain breakdown of 5 groups of the 1520 users.



(b) Daily energy percentage breakdown, averaged over all users.

Figure 7: Daily energy breakdown by components.

- **Energy breakdown by components**
- **Cellular paging vs. WiFi beacon:** The energy drain from cellular paging and WiFi beacon are 12.5% and 2.3%, respectively
- **Cellular vs. WiFi:** The energy drain over cellular (LTE and 3G) and over WiFi are 11.7% and 1.3%, respectively

Lessons learned from the two studies

- Screen is responsible for a large fraction of energy consumption
  - Sensor useful to detect the amount of light in the environment and automatically adjust brightness
  - Sensor useful to detect proximity of user’s face during calls and turn screen off
- Wi-Fi more energy efficient than cellular technologies
- **CPU idle most of the time (light computational load)**
- **Large amount of energy spent for background activities**

More efficient CPUs

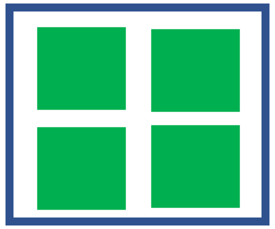
- Computational load is generally light
  - But with spikes, e.g. rendering of a web-page, computational photography
  - Sometimes high computational power needed, e.g. when playing games

- A high-performance CPU
  - **User is happy**
  - **Overkill most of the time, waste of energy**

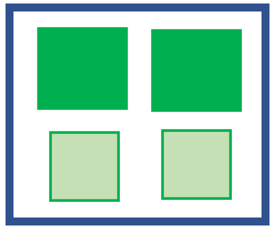
- A low-performance CPU
  - **Reduced power consumption**
  - **User's experience may be not excellent**

Heterogenous multi-processing

- Uniprocessor: performance cannot grow indefinitely
- Symmetric MultiProcessor (SMP): increased performance achieved by means of increased parallelism. All processors/cores are equal.



- **Hetherogenous MultiProcessor (HMP)**: increased performance achieved by parallelism. Power efficiency through differences in processors.



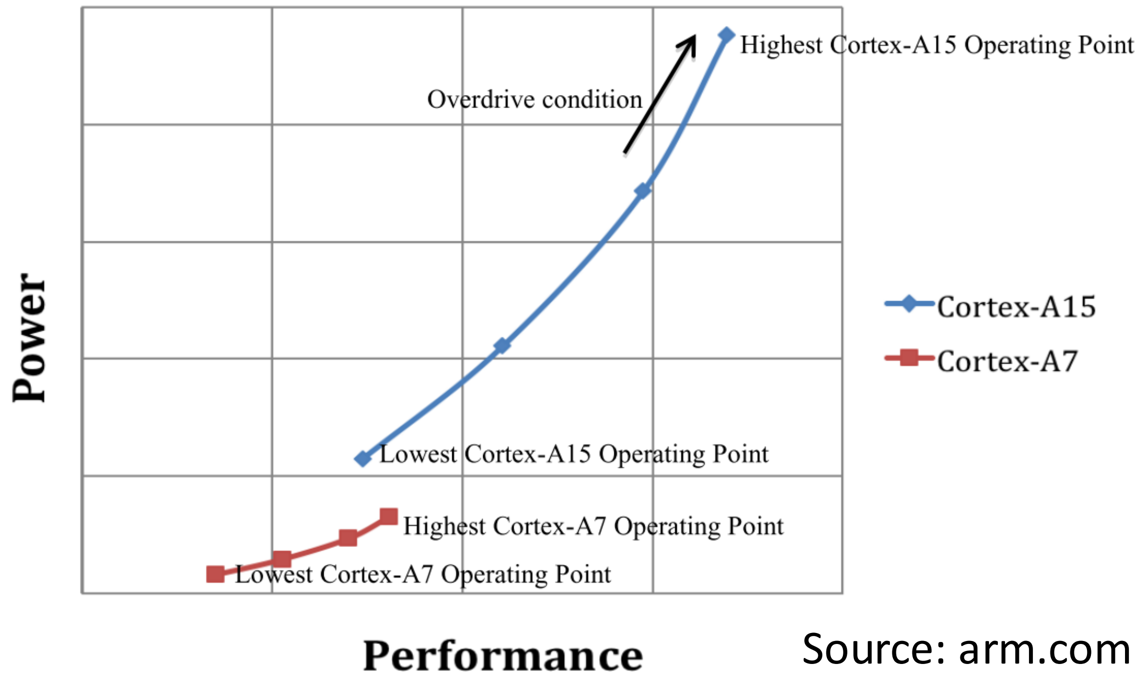
HMP: ARM's big.LITTLE

- ARM big.LITTLE combines the high-performance CPUs and smaller CPUs in one system
- Software dynamically moves to the right size processor for the required performance

- big: A75, A15, …
- LITTLE: A55, A7, …

- A15 vs A7:
  - A15: out-of-order execution, triple issue, pipeline 15-24 stages
  - A7: in-order execution, double issue, pipeline 8-10 stages
  - Performance: A15 vs A7 ~2x
  - Energy efficiency: A7 vs A15 ~3x
  - Same instruction set, same registers

Both use Dynamic Voltage and Frequency Scaling (DVFS)



Source: arm.com

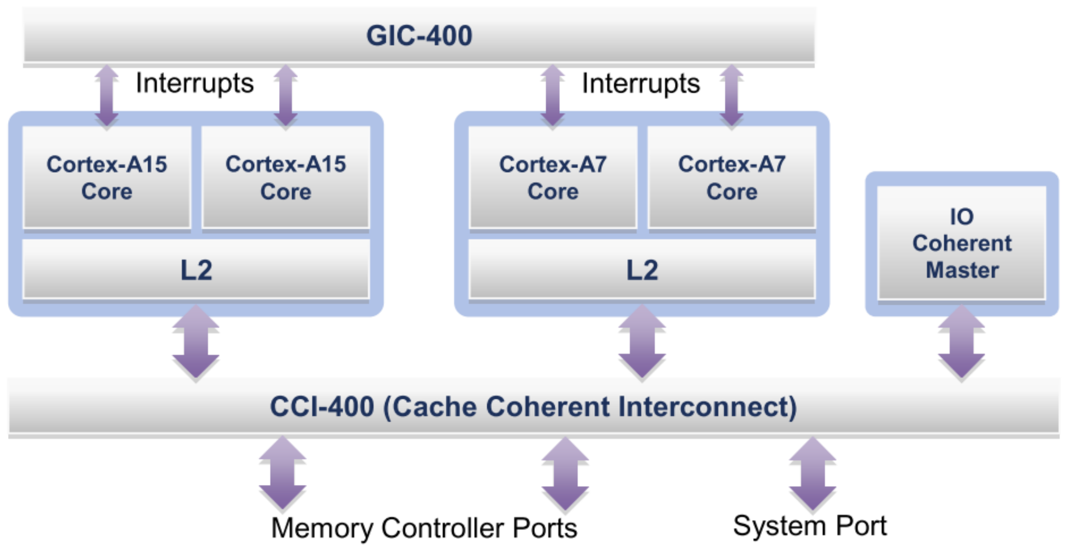
Dynamic Voltage and Frequency Scaling

- Energy saving technique that exploits:
  - linear relationship between power consumption and frequency
  - quadratic relationship between power consumption and voltage
- This relationship is given as:

  $P = C \times V^2 \times f$
- Where:

  P: power
  C: switching capacitance of the logic circuit
  V: operational voltage
  f: operational frequency
- At lower frequencies core can operate at lower voltages
- Operating Point: used frequency and voltage

ARM big.LITTLE



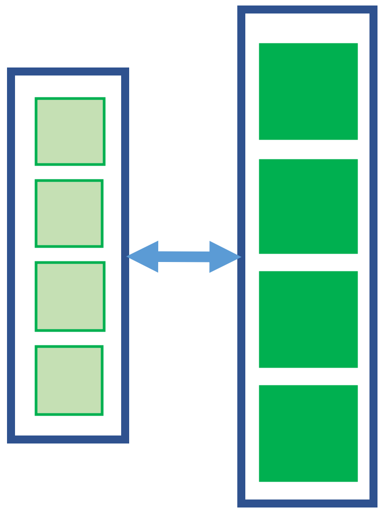
Source: arm.com

Three main task migration models are possible:

- Clustered switching
- CPU migration (in-kernel switching)
- Heterogeneous MultiProcessing (global task scheduling)

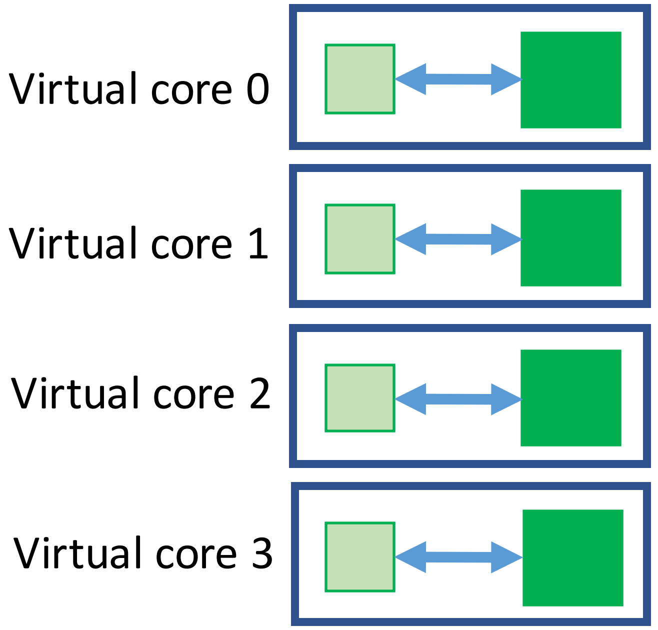
ARM big.LITTLE

- Clustered switching
  - Clusters of big and LITTLE core have the same size
  - A single cluster can be active at any time.
  - When the load becomes high, all tasks are moved to the big cluster and vice versa
  - The inactive cluster is powered off
  - Not all cores are used
  - Simple
  - Sub-optimal both in terms of energy and computing power



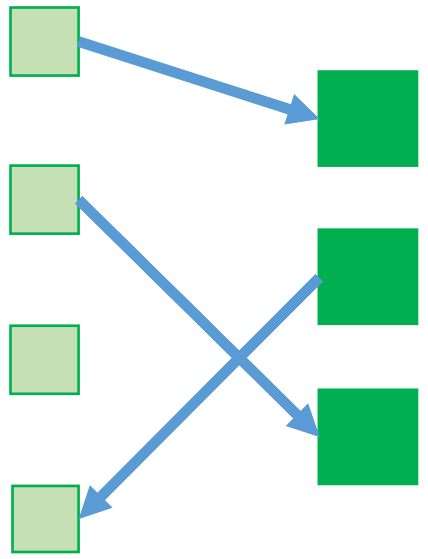
ARM big.LITTLE

- CPU migration
  - Clusters of big and LITTLE core have the same size
  - When the load on a CPU becomes high, tasks are moved to the big core and vice versa
  - The inactive real core is powered off
  - Not all cores are used
  - Simple
  - Sub-optimal both in terms of energy and computing power



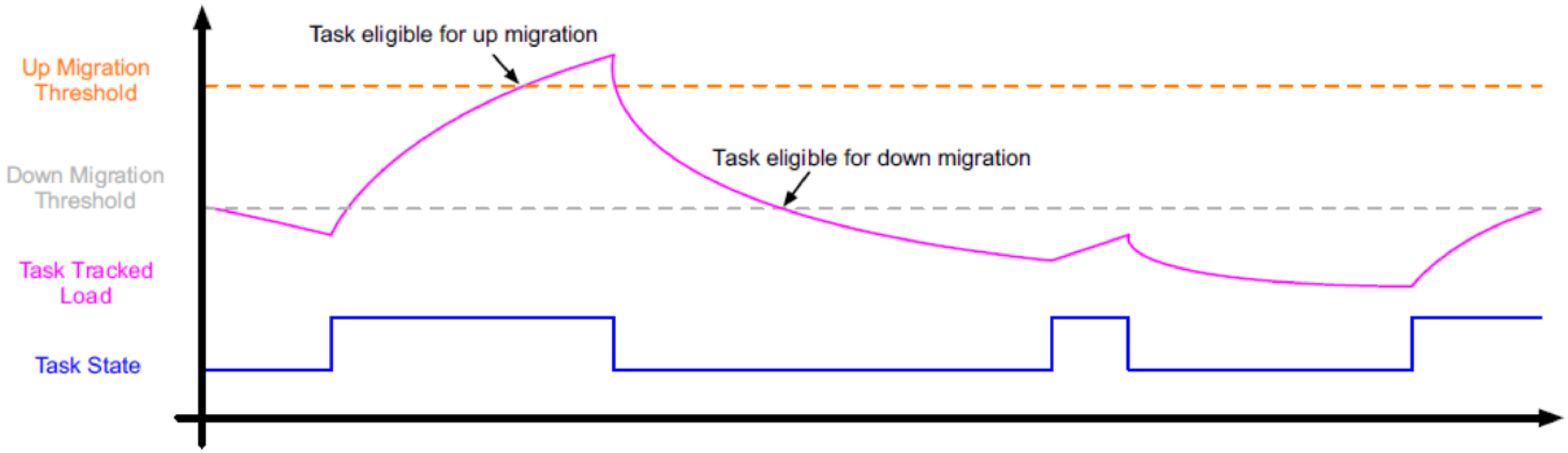
ARM big.LITTLE

- Global task scheduling
  - Numbers of big and LITTLE cores may be different
  - When the load on a LITTLE core becomes high, tasks are moved to a big core and vice versa
  - Inactive cores are powered off
  - All cores may be in use at the same time
  - More complex scheduling decisions



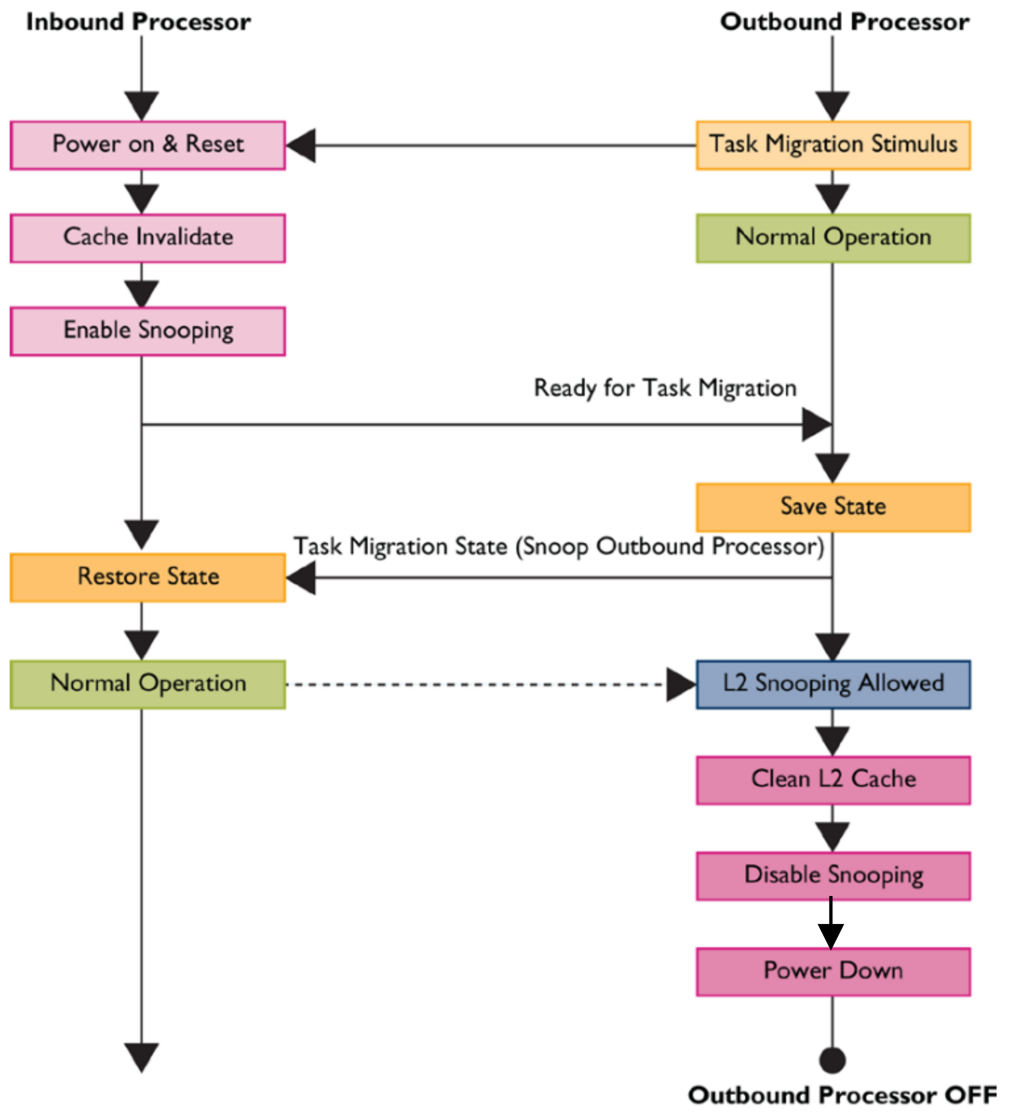
Global Task Scheduling

- In GTS the scheduler has to track the activity level of each task



Task migration

- To improve task migration time:
  - the state that is saved on the outbound processor is snooped and restored on the inbound processor rather than going via main memory
  - the level-2 cache of the outbound processor remains powered up after a task migration to improve the cache warming time of the inbound processor through snooping of data values



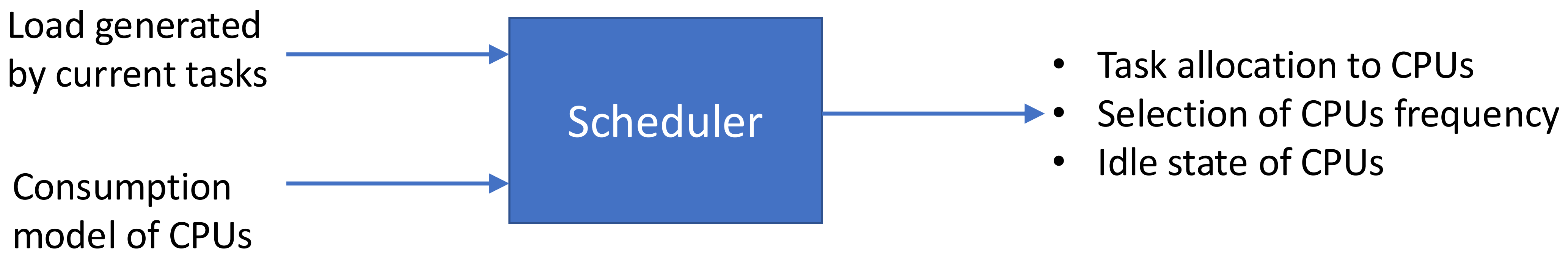
Scheduling in Linux (before EAS)

- The *Completely Fair Scheduler* aims to maximize throughput and performance of interactive applications
- Power is managed by two external components: *cpuidle* and *cpufreq*
- **cpuidle**:
  - CPUs may enter different low power modes called C states
  - The lower the power the longer the time to return operational
  - cpuidle is called when CPU is idle to select the C state to enter
  - Composed of drivers (which know the details of the CPU in terms of C states) and governors (where the policy is implemented)
- **cpufreq**:
  - Similar to cpuidle, but it determines the operational frequency of CPU
  - ondemand governor: if CPU utilization is above a given threshold frequency is increased, if below another threshold freqency is decreased

Energy Aware Scheduler (EAS)

- The scheduler is in a better position to take power-related decisions with respect to cpuidle and cpufreq
    - global view, per task information, already keeps track of tasks behavior
- EAS is an energy-aware scheduler used in recent versions of Android and now merged into the mainline Linux kernel
- EAS principles of operations:
    - A model of CPUs and clusters provides information about the power requirements at the different operational points
    - Tasks are tracked, in terms of load, to evaluate their impact on CPUs
    - When a task must be assigned to a CPU, the decision takes into account the energy impact
- EAS operates **only when there is spare capacity**
    - If the load is high there is nothing to save, behavior returns to standard CFS

EAS

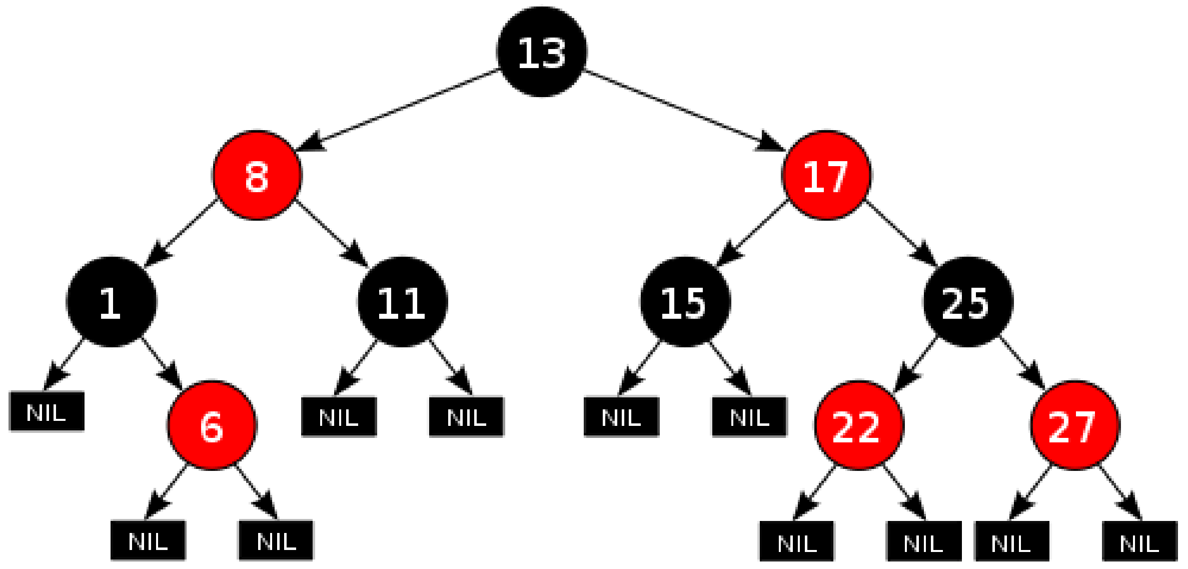

Load generated by current tasks
Consumption model of CPUs
Scheduler
• Task allocation to CPUs
• Selection of CPUs frequency
• Idle state of CPUs

Linux Scheduling

- Completely Fair Scheduler (CFS) is the "Desktop" scheduler, introduced in 2.6.23
- Tries to model an ideal, precise multi-tasking CPU on real HW
- “**Ideal** multi-tasking CPU” can run each task at precise equal speed, in parallel, each at 1/N speed. For example: if there are 2 tasks running, then it runs each at 50% physical power
- When the time for tasks is not balanced (one or more tasks are not given a fair amount of time relative to others), then these tasks should be given time to execute.
- CFS registers the amount of time provided to a given task (the *virtual runtime*)
- On *ideal* hardware, at any time all tasks would have the same virtual runtime — i.e., tasks would execute simultaneously and no task would ever get “out of balance” from the ideal share of CPU time
- The smaller a task's virtual runtime—meaning the smaller amount of time a task has been granted the CPU—the higher its need for the processor

CFS

- Tasks are stored in a red-black tree (not a queue) ordered in terms of virtual time
  - A red-black tree is roughly balanced: any path in the tree will never be more than twice as long as any other path.
  - Insert and deletion are O(log n)



CFS

- The scheduler picks the left-most node of the red-black tree.
- The task is assigned the CPU, then its execution time is added to the virtual runtime. Finally, it is inserted back into the tree if runnable.
- Once virtual runtime gets high enough so that another task becomes the “leftmost task” of the time-ordered rbtree (plus a small amount of “granularity” distance relative to the leftmost task so that we do not over-schedule tasks and trash the cache), then the new leftmost task is picked and the current task is preempted.
- CFS doesn't use priorities directly but instead uses them as a decay factor for the time a task is permitted to execute.
  - Lower-priority tasks have higher factors of decay, where higher-priority tasks have lower factors of decay.
  - This means that the time a task is permitted to execute dissipates more quickly for a lower-priority task than for a higher-priority task.
  - This avoids maintaining run queues per priority.

CFS

- *sched_lantency_ns*: target latency, period in which every runnable threads will put onto the CPU (24 ms)
- If there are N runnable processes, each process is executed for *sched_latency/N*
  - e.g. 4 runnable processes, each executed for 6 ms
- When the number of processes is high, context switches introduce too much overhead:
  - *min_granularity*: minimum time a runnable process is executed (3 ms)
  - e.g. if 12 processes are runnable *sched_latency/N* = 2 ms, which is lower than 3 ms. So each process is assigned the CPU for 3 ms.

CFS, weights

- Processes do not have all the same nice level
  - nice level: from -20 (max priority) to +19 (min priority)
- Assignment of CPU time is carried out considering their relative weights
- CPU time assigned to process i =

$$\frac{\text{sched\_latency} * \text{weight\_i}}{\text{sum all weights in runqueue}}$$

$$\text{weight} = 1024/1.25^{\text{nice\_level}}$$

From source code:
Nice levels are multiplicative, with a gentle 10% change for every nice level changed. I.e. when a CPU-bound task goes from nice 0 to nice 1, it will get ~10% less CPU time than another CPU-bound task that remained on nice 0. The "10% effect" is relative and cumulative: from_any_nicelevel, if you go up 1 level, it's -10% CPU usage, if you go down 1 level it's +10% CPU usage. (to achieve that we use a multiplier of 1.25. If a task goes up by ~10% and another task goes down by ~10% then the relative distance between them is ~25%.)

```
const int sched_prio_to_weight[40] = {
 /* -20 */     88761,     71755,     56483,     46273,     36291,
 /* -15 */     29154,     23254,     18705,     14949,     11916,
 /* -10 */      9548,      7620,      6100,      4904,      3906,
 /*  -5 */      3121,      2501,      1991,      1586,      1277,
 /*   0 */      1024,       820,       655,       526,       423,
 /*   5 */       335,       272,       215,       172,       137,
 /*  10 */       110,        87,        70,        56,        45,
 /*  15 */        36,        29,        23,        18,        15,
};
```

CFS, weights

- Two tasks running at nice 0 (weight of T1 and T2 is 1024)
  - T1 and T2 get 50% of time: 1024/(1024+1024)=0.5
- Task 1 is moved to nice -1:
  - T1: 1277/(1024+1277) is approximately 0.55
  - T2: 1024/(1024+1277) is approximately 0.45
- Task 2 is then moved to nice +1:
  - T1: 1277/(820+1277) is approximately 0.61
  - T2: 820/(820+1277) is approximately 0.39

CFS, weights

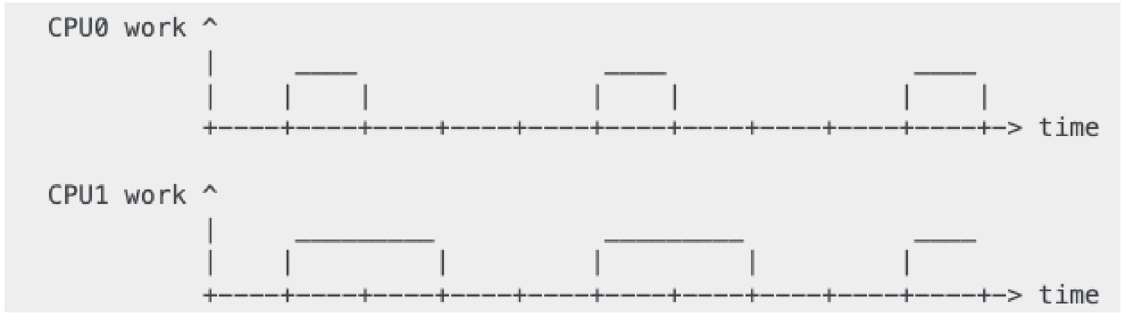
- If T1 has nice -1 and T2 has nice +1 weights are 0.61 and 0.39
- With sched_latency equal to 24 ms
  - T1 is executed for 14.64 ms
  - T2 is executed for 9.36 ms
  - suppose T1.vrtime = 5000 and T2.vrtime = 5001 (actually expressed in ns)
- When their virtual runtime is updated:
  - T1.vrtime = T1.vrtime + 14.64ms * (L0/T1.weight) = 5000 + 14.64*(0.5/0.61) = 5012
  - T2.vrtime = T2.vrtime + 9.36ms * (L0/T2.weight) = 5001 + 9.36*(0.5/0.39) = 5013

Capacity Awareness

- CPU capacity is a measure of the performance a CPU can reach, normalized against the most performant CPU in the system.
- Heterogeneous systems are also called asymmetric CPU capacity systems, as they contain CPUs of different capacities.
- Disparity in maximum attainable performance originates from two factors:
  - CPUs may have different microarchitectures
  - with Dynamic Voltage and Frequency Scaling (DVFS), not all CPUs may be physically able to attain the higher Operating Performance Points (OPP).
- capacity(cpu) = work_per_hz(cpu) * max_freq(cpu)

Capacity Awareness

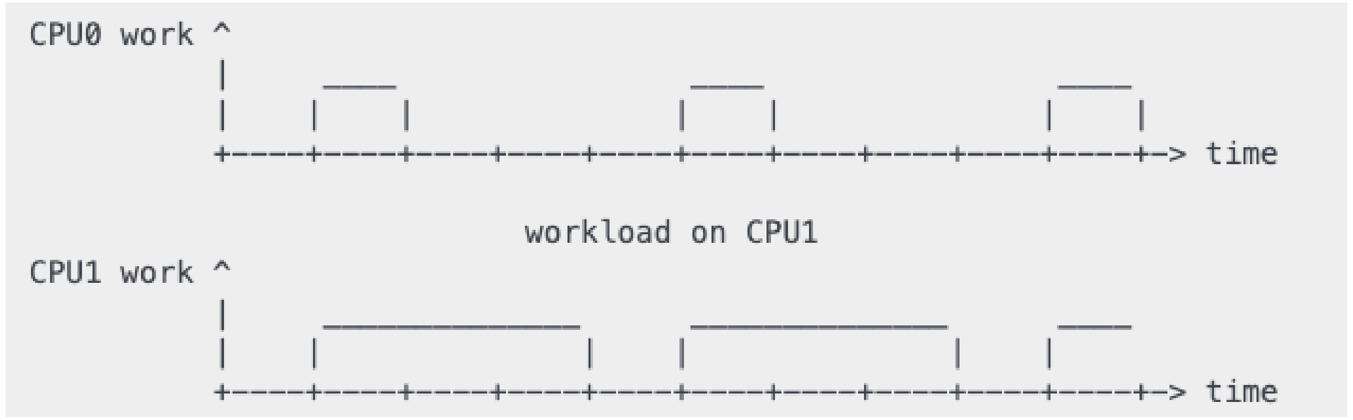
- Consider a dual-core asymmetric CPU capacity system where
  - work_per_hz(CPU0) = W
  - work_per_hz(CPU1) = W/2
- All CPUs are running at the same fixed frequency
- By the above definition of capacity:
  - capacity(CPU0) = C
  - capacity(CPU1) = C/2
- A periodical task, when executed on the two CPUs will require a different amount of time



```
CPU0 work ^
          |     ____                ____                ____
          |    |    |              |    |              |    |
          +----+----+----+----+----+----+----+----+----+----+-> time

CPU1 work ^
          |     _________           _________           ____
          |    |         |         |         |         |
          +----+----+----+----+----+----+----+----+----+----+-> time
```

Capacity Awareness

- CPUs also have different maximum OPPs.
- Consider two CPUs with **same work_per_hz()** with:
  - max_freq(CPU0) = F
  - max_freq(CPU1) = 1/3 * F
- This yields:
  - capacity(CPU0) = C
  - capacity(CPU1) = C/3



```
CPU0 work ^
          |     ____                ____                ____
          |    |    |              |    |              |    |
          +----+----+----+----+----+----+----+----+----+----+-> time

                             workload on CPU1
CPU1 work ^
          |     ______________      ______________      ____
          |    |              |    |              |    |
          +----+----+----+----+----+----+----+----+----+----+-> time
```

Capacity Awareness

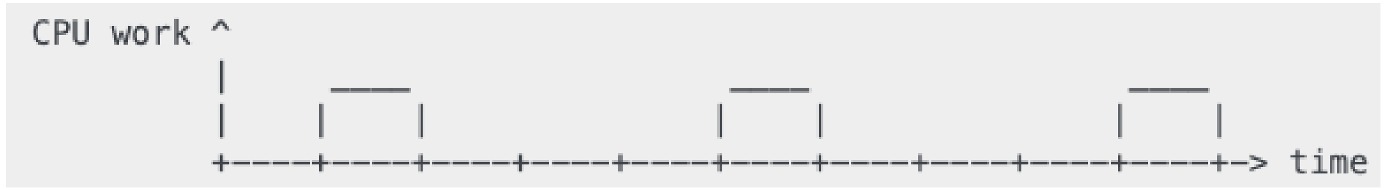
- Capacity aware scheduling requires an expression of a task’s requirements with regards to CPU capacity.
- Task utilization is a percentage meant to represent the throughput requirements of a task. A simple approximation of it is the task’s duty cycle, i.e.:

  task_util(p) = duty_cycle(p)
- On an SMP system with fixed frequencies, 100% utilization suggests the task is a busy loop. Conversely, 10% utilization hints it is a small periodic task that spends more time sleeping than executing.
- Variable CPU frequencies and asymmetric CPU capacities complexify this.

Capacity Awareness

Frequency invariance

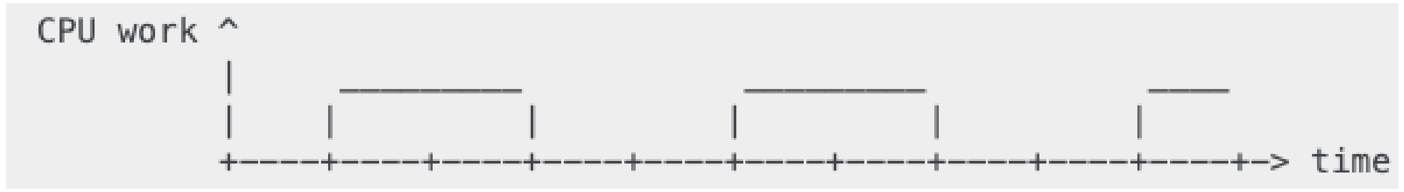
- Duty cycle is directly impacted by the current OPP the CPU is running at.
- Consider running a periodic workload at a given frequency F



```
CPU work ^
         |     ____                ____                ____
         |    |    |              |    |              |    |
         +----+----+----+----+----+----+----+----+----+----+-> time
```

  This yields duty_cycle(p) == 25%.

- Now, consider running the same workload at frequency F/2



```
CPU work ^
         |     _________           _________           ____
         |    |         |         |         |         |
         +----+----+----+----+----+----+----+----+----+----+-> time
```

This yields duty_cycle(p) == 50%, despite the task is the same (the same amount of work).

The task utilization signal can be made frequency invariant using the following formula:

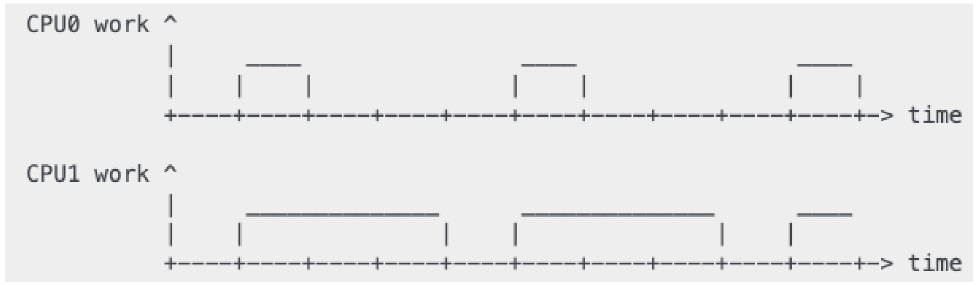
*task_util_freq_inv(p) = duty_cycle(p) * (curr_frequency(cpu) / max_frequency(cpu))*

Applying this formula to the two examples above yields a frequency invariant task utilization of 25%

Capacity Awareness

CPU invariance

- Consider a system where
  - capacity(CPU0) = C
  - capacity(CPU1) = C/3
- Executing a given periodic workload on each CPU at their maximum frequency would result in:



```
CPU0 work ^
          |     ____                ____                ____
          |    |    |              |    |              |    |
          +----+----+----+----+----+----+----+----+----+----+-> time

CPU1 work ^
          |     ______________      ______________      ____
          |    |              |    |              |    |
          +----+----+----+----+----+----+----+----+----+----+-> time
```

  - duty_cycle(p) == 25% if p runs on CPU0 at its maximum frequency
  - duty_cycle(p) == 75% if p runs on CPU1 at its maximum frequency
- The task utilization signal can be made CPU invariant using the following formula:

  *task_util_cpu_inv(p) = duty_cycle(p) * (capacity(cpu) / max_capacity)*

with *max_capacity* being the highest CPU capacity value in the system

Applying this formula to the above example above yields a CPU invariant task utilization of 25%.

Capacity Awareness

- Invariance to both frequency and CPU

```
                                   curr_frequency(cpu)   capacity(cpu)
task_util_inv(p) = duty_cycle(p) * ------------------- * -------------
                                   max_frequency(cpu)    max_capacity
```

as if it were running on the highest-capacity CPU in the system, running at its maximum frequency.

EAS

- The goal of EAS is to minimize energy, while still getting the job done, i.e. to maximize:

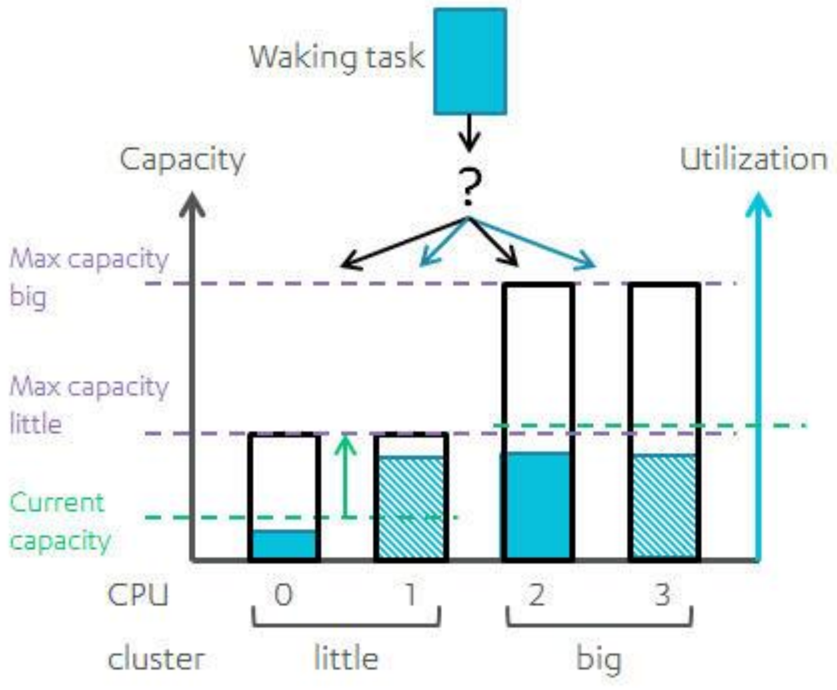
  performance [inst/s] /power [W]

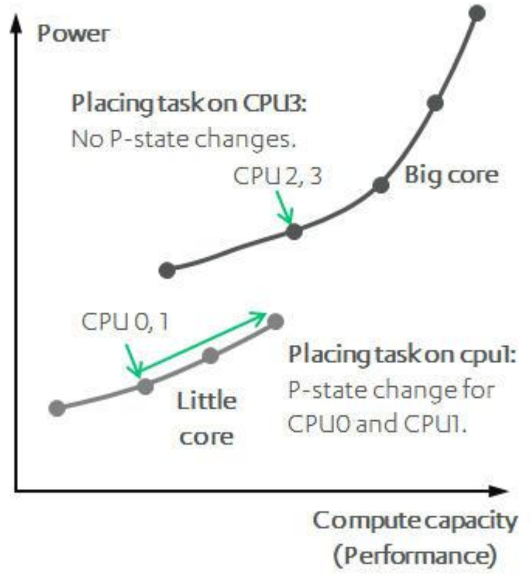
  which is equivalent to minimizing:

  energy [J] / instruction
- When it is time for the scheduler to decide where a task should run (during wake-up), the EM is used to break the tie between several good CPU candidates and pick the one that is predicted to yield the best energy consumption without harming the system's throughput.
- The predictions made by EAS rely on specific elements of knowledge about the platform's topology, which include the 'capacity' of CPUs, and their respective energy costs.
- Looks for the CPU with the highest spare capacity (CPU capacity - CPU utilization) in each performance domain, checks if placing the task there could save energy compared to leaving it the CPU where the task ran in its previous activation.

EAS

- Selection of CPU
- A new waking task can be placed on either of two CPUs - #1 and #3
- EAS estimates the energy costs associated to the two options:
  - CPU#1: operating point must be moved up for both CPU#0 and CPU#1
  - CPU#3: no operating point change, but higher power used

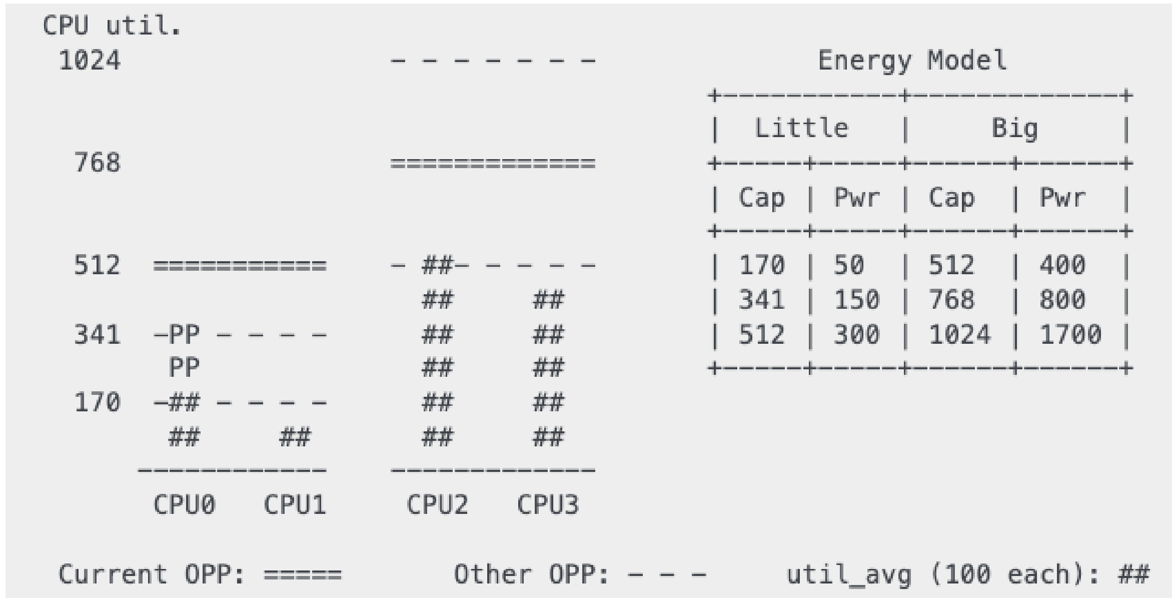




EAS will probably choose CPU#1 because the small additional energy cost of increasing the OPP of CPU#0 (and CPU#1 – since we suppose both CPUs are in the same frequency domain) is not significant compared with the better power efficiency of running the task on CPU#1 instead of CPU#3.

EAS

- Example: platform with 2 independent performance domains composed of two CPUs each.
  - CPU0 and CPU1 are little
  - CPU2 and CPU3 are big
- The scheduler must decide where to place a task P whose util_avg = 200 and prev_cpu = 0.
- The current utilization of the CPUs: CPUs 0-3 have a util_avg of 400, 100, 600 and 500 respectively
- Each performance domain has 3 OPPs.
- The CPU capacity and power cost associated with each OPP is listed in the Energy Model table. The util_avg of P is shown on the figures as 'PP'.



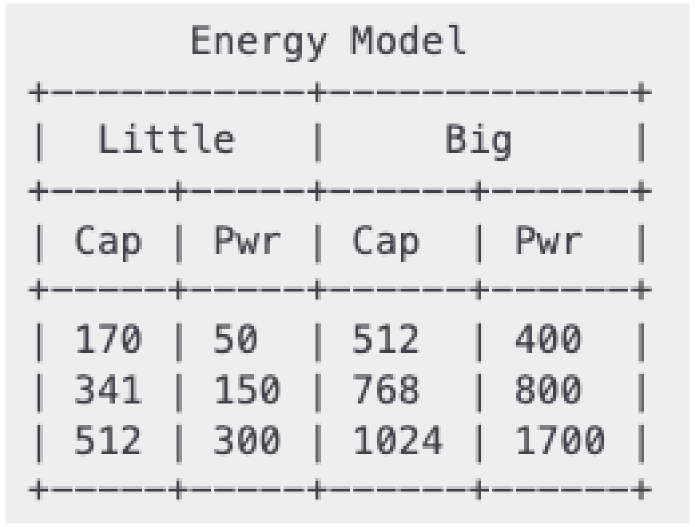
```
CPU util.
 1024                 - - - - - - -              Energy Model
                                          +-----------+-------------+
                                          |  Little   |     Big     |
  768                 =============       +-----+-----+------+------+
                                          | Cap | Pwr | Cap  | Pwr  |
                                          +-----+-----+------+------+
  512  ===========    - ##- - - - -       | 170 | 50  | 512  | 400  |
                        ##     ##         | 341 | 150 | 768  | 800  |
  341  -PP - - - -      ##     ##         | 512 | 300 | 1024 | 1700 |
        PP              ##     ##         +-----+-----+------+------+
  170  -## - - - -      ##     ##
        ##     ##       ##     ##
      ------------    -------------
       CPU0   CPU1     CPU2   CPU3

 Current OPP: =====       Other OPP: - - -     util_avg (100 each): ##
```

EAS

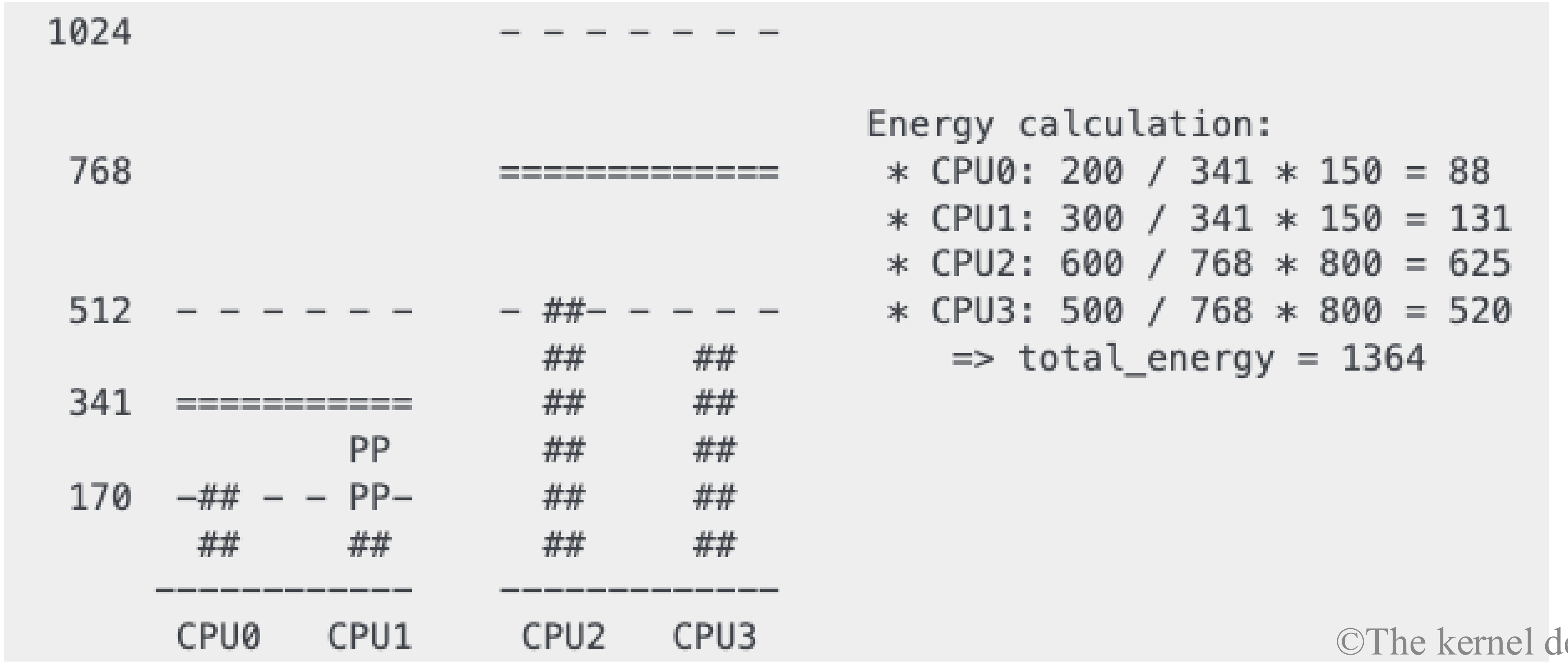
1. Look for the CPUs with the maximum spare capacity in the two performance domains. In this example, CPU1 and CPU3
2. Estimate the energy of the system if P was placed on either of them, and check if that would save some energy compared to leaving P on CPU0.

EAS

- Task is migrated on CPU1



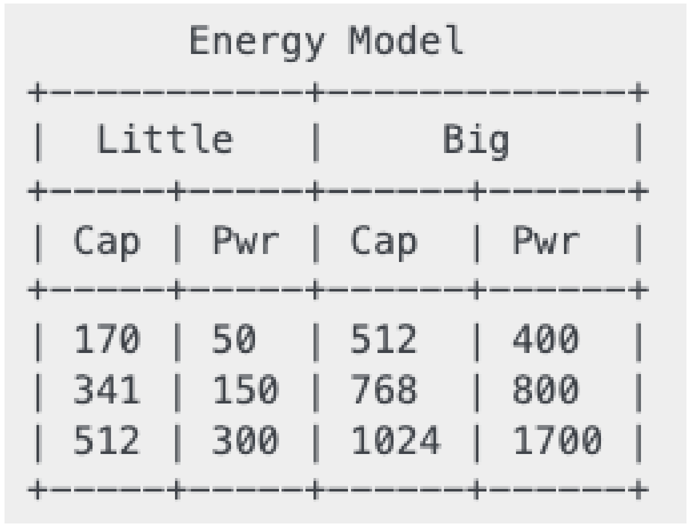
```
          Energy Model
+-----------+-------------+
|  Little   |     Big     |
+-----+-----+------+------+
| Cap | Pwr | Cap  | Pwr  |
+-----+-----+------+------+
| 170 | 50  | 512  | 400  |
| 341 | 150 | 768  | 800  |
| 512 | 300 | 1024 | 1700 |
+-----+-----+------+------+
```



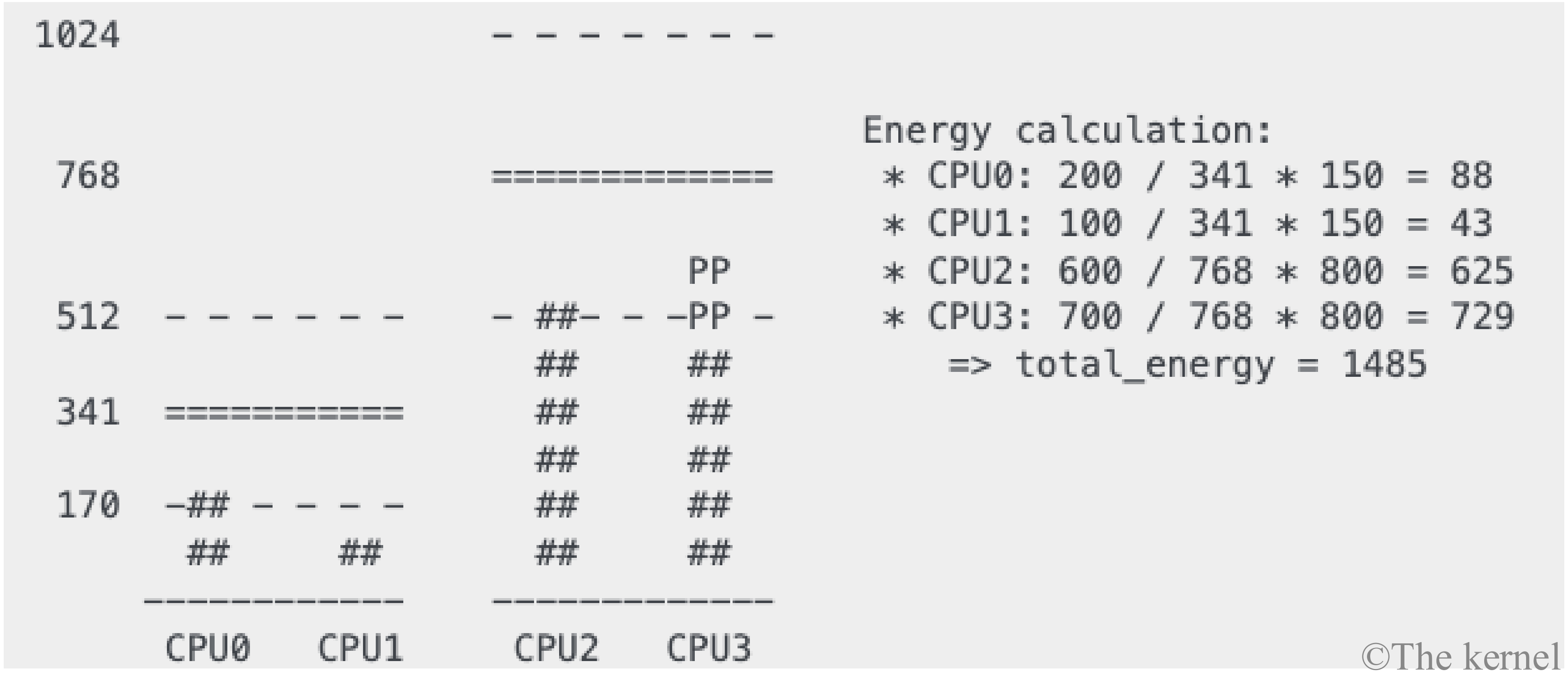
```
1024                 - - - - - - -

                                         Energy calculation:
 768                 =============        * CPU0: 200 / 341 * 150 = 88
                                          * CPU1: 300 / 341 * 150 = 131
                                          * CPU2: 600 / 768 * 800 = 625
 512  - - - - - -    - ##- - - - -        * CPU3: 500 / 768 * 800 = 520
                       ##      ##            => total_energy = 1364
 341  ===========      ##      ##
              PP       ##      ##
 170  -## - - PP-      ##      ##
       ##     ##       ##      ##
     ------------    -------------
      CPU0    CPU1     CPU2    CPU3
```

EAS

- Task is migrated on CPU3



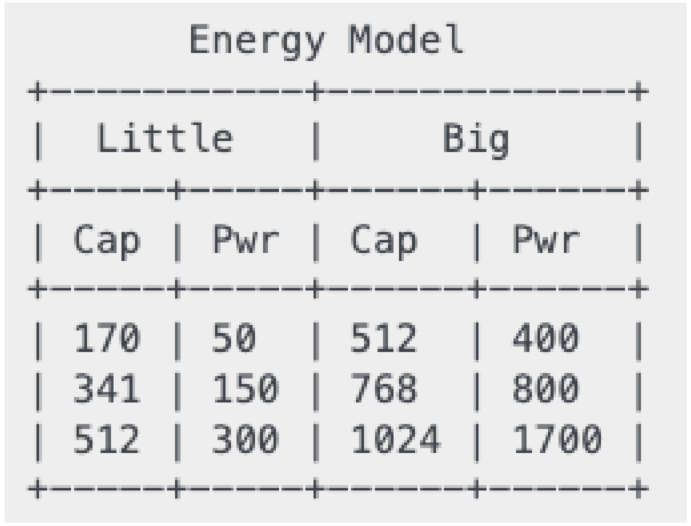
```
         Energy Model
+-----------+-------------+
|  Little   |     Big     |
+-----+-----+------+------+
| Cap | Pwr | Cap  | Pwr  |
+-----+-----+------+------+
| 170 | 50  | 512  | 400  |
| 341 | 150 | 768  | 800  |
| 512 | 300 | 1024 | 1700 |
+-----+-----+------+------+
```



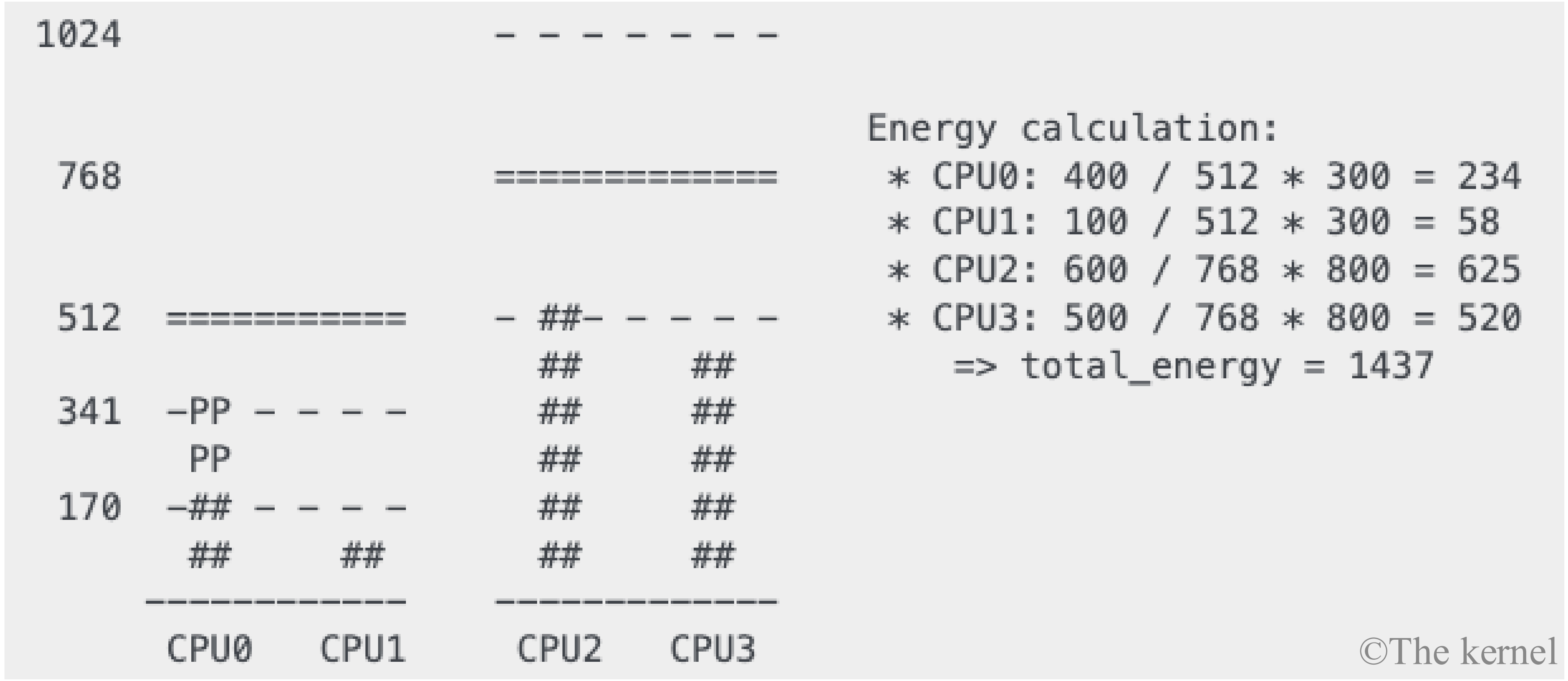
```
1024                 - - - - - - -
                                      Energy calculation:
 768                 =============     * CPU0: 200 / 341 * 150 = 88
                                       * CPU1: 100 / 341 * 150 = 43
                              PP       * CPU2: 600 / 768 * 800 = 625
 512  - - - - - -    - ##- - -PP -     * CPU3: 700 / 768 * 800 = 729
                       ##     ##          => total_energy = 1485
 341  ===========      ##     ##
                       ##     ##
 170  -## - - - -      ##     ##
       ##     ##       ##     ##
     ------------    --------------
      CPU0   CPU1     CPU2   CPU3
```

EAS

- Task remains on CPU0



```
        Energy Model
+-----------+-------------+
|  Little   |     Big     |
+-----+-----+------+------+
| Cap | Pwr | Cap  | Pwr  |
+-----+-----+------+------+
| 170 | 50  | 512  | 400  |
| 341 | 150 | 768  | 800  |
| 512 | 300 | 1024 | 1700 |
+-----+-----+------+------+
```



```
1024                 - - - - - - -

                                       Energy calculation:
 768                 =============      * CPU0: 400 / 512 * 300 = 234
                                        * CPU1: 100 / 512 * 300 = 58
                                        * CPU2: 600 / 768 * 800 = 625
 512  ===========    - ##- - - - -      * CPU3: 500 / 768 * 800 = 520
                       ##     ##           => total_energy = 1437
 341  -PP - - - -      ##     ##
        PP             ##     ##
 170  -## - - - -      ##     ##
        ##     ##      ##     ##
     ------------    -------------
      CPU0   CPU1     CPU2   CPU3
```

EAS

- In previous example, the task is executed on CPU1
- Little CPUs aren’t always necessarily more energy-efficient than big CPUs. For some systems, the high OPPs of the little CPUs can be less energy-efficient than the lowest OPPs of the big ones

EAS, over-utilization

- CPUs are flagged as ‘over-utilized’ as soon as they are used at more than 80% of their compute capacity.
- As long as no CPUs are over-utilized, load balancing is disabled and EAS overrides the wake-up balancing code.
- EAS is likely to load the most energy efficient CPUs of the system more than the others if that can be done without harming throughput.
- The load-balancer is disabled to prevent it from breaking the energy-efficient task placement found by EAS. If load is below 80%:
  - there is some idle time on all CPUs, so the utilization signals used by EAS are likely to accurately represent the ‘size’ of the various tasks in the system;
  - all tasks should already be provided with enough CPU capacity, regardless of their nice values;
  - since there is spare capacity all tasks must be blocking/sleeping regularly and balancing at wake-up is sufficient.

PELT

- In standard Linux kernel
- The main idea is that a process can contribute to load even if it is not actually running at the moment
- PELT tracks load on a per-entity basis
- L is the load of a given task

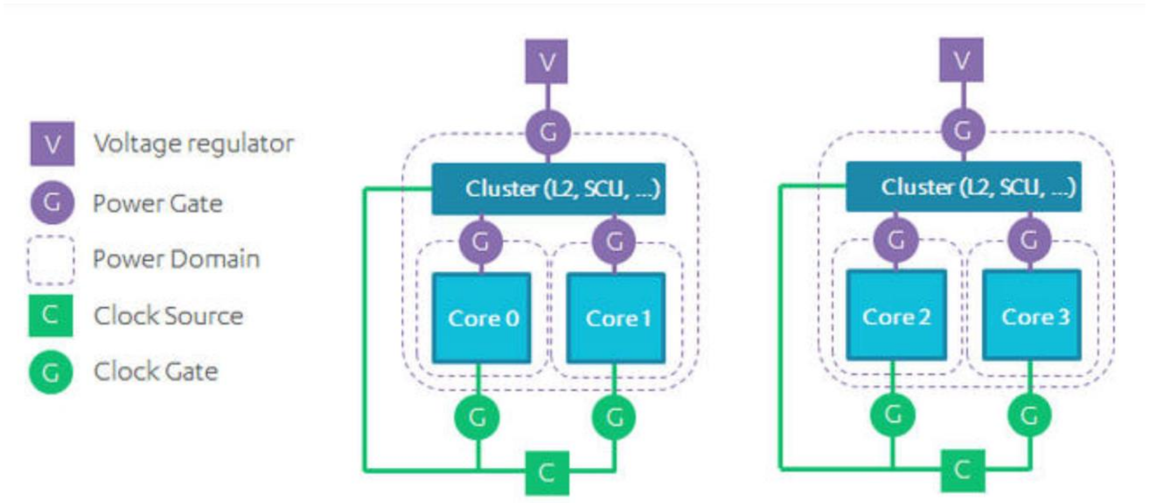
  $L = L_0 + L_1 q + L_2 q^2 + L_3 q^3 + \dots$

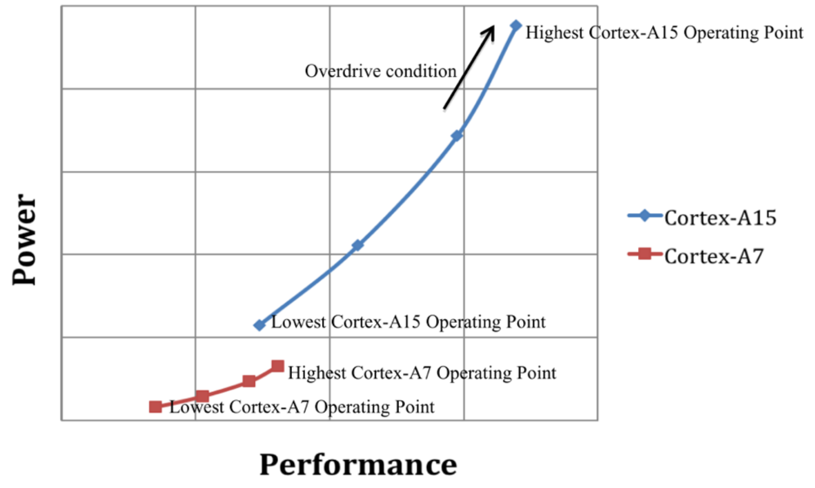
  where q is the decay factor

L_x is the observed load *x* periods in the past (portion of period it was running or runnable), period 1 ms, total 32 ms.

Enegy Model (EM)

- The consumption model consists of:
  - topology
  - power consumption for each supported operating point
  - power consumption for each C-state (the idle power management state)
  - wake-up energy cost for each C-state
  - the model contains data for CPUs and clusters





EM

```
cpus {
  . . .
  A57_0: cpu@0 {
           compatible = "arm,cortex-a57","arm,armv8";
           ...
           clocks = <&scpi_dvfs 0>;
           cpu-idle-states = <&CPU_SLEEP_0 &CLUSTER_SLEEP_0>;
           sched-energy-costs = <&CPU_COST_0 &CLUSTER_COST_0>;
  };

  A57_1: cpu@1 {
           compatible = "arm,cortex-a57","arm,armv8";
           ...
           clocks = <&scpi_dvfs 0>;
           cpu-idle-states = <&CPU_SLEEP_0 &CLUSTER_SLEEP_0>;
           sched-energy-costs = <&CPU_COST_0 &CLUSTER_COST_0>;
  };
  A53_0: cpu@100 {
           compatible = "arm,cortex-a53","arm,armv8";
           ...
           clocks = <&scpi_dvfs 1>;
           cpu-idle-states = <&CPU_SLEEP_0 &CLUSTER_SLEEP_0>;
           sched-energy-costs = <&CPU_COST_1 &CLUSTER_COST_1>;
  };
  A53_1: cpu@101 {
           compatible = "arm,cortex-a53","arm,armv8";
           ...
           clocks = <&scpi_dvfs 1>;
           cpu-idle-states = <&CPU_SLEEP_0 &CLUSTER_SLEEP_0>;
           sched-energy-costs = <&CPU_COST_1 &CLUSTER_COST_1>;
};
```

```
energy-costs {
  CPU_COST_0: core-cost0 {
           busy-cost-data = <
                      417 168
                      579 251
                      744 359
                      883 479
                      1024 616 >;
           idle-cost-data = <
                      15
                      0 >;
  };
  CPU_COST_1: core-cost1 {
           busy-cost-data = <
                      235 33
                      302 46
                      368 61
                      406 76
                      447 93 >;
           idle-cost-data = <
                      6
                      0 >;
  };
  CLUSTER_COST_0: cluster-cost0 {
           busy-cost-data = <
                      417 24
                      579 32
                      744 43
                      883 49
                      1024 64 >;
           idle-cost-data = <
                      65
                      24 >;
  };
  CLUSTER_COST_1: cluster-cost1 {
   ...
```

SchedTune

- EAS can be tuned using information that is available in the user-space
- Using cgroups
  - tasks can be included in boost groups
  - tasks can be flagged to prefer an idle CPU
- boost groups: a margin can be added to the utilization of a task to promote/demote its execution
- Android
  - Top-app: prefer idle, boost of 10%
  - Foreground: prefer idle, no boost
  - Background: prefer pack, no boost
  - System background: prefer pack, no boost

EAS

- CFS already includes a Per Entity Load Tracking (PELT) functionality
- EAS may use a variation called Window Assisted Load Tracking (WALT)
  - According to some benchmarks it provides some benefits w.r.t PELT
  - Time is divided in windows
  - The load generated by a task is evaluated for each window
  - The predicted load of a task is equal to
    ```
    max(load_during_last_window, avg(last_N_windows))
    ```
  - Windows do not exist when a task is blocked: a heavy task exiting from sleep is immediately classified as demanding

Power-saving at a higher-level

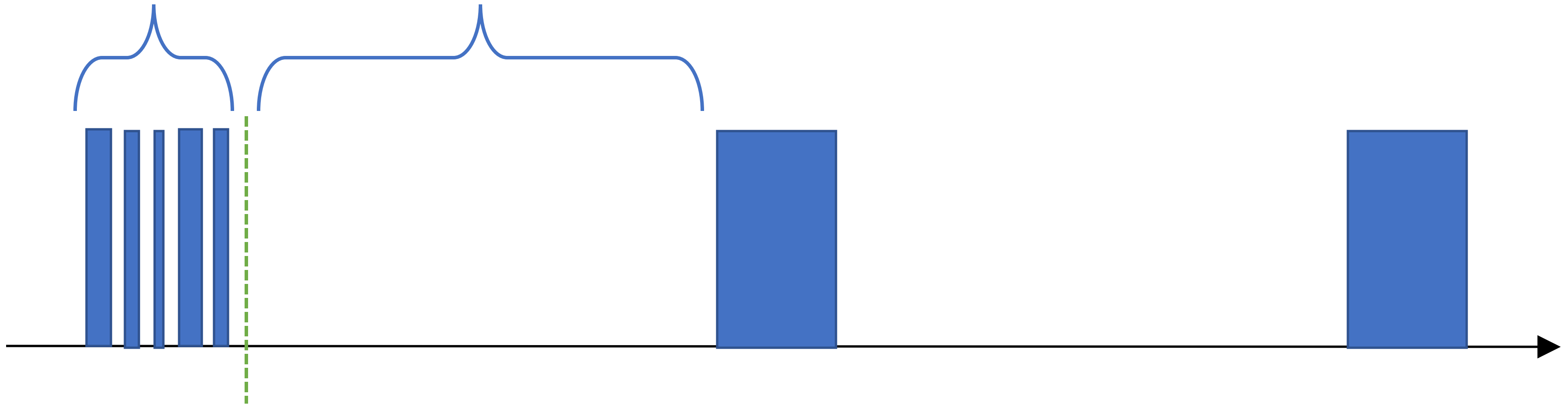
- Android includes some power-saving mechanisms:
  - **Doze** mode
  - App standby
- Doze:
  - Activated when the device is not connected to a power source and left stationary for a long period of time
  - background CPU and network activity are deferred
- App standby:
  - Android limits the resources available to apps depending on how frequently and how recently they have been used

Doze mode

Small CPU bursts, preventing the CPU from entering low power states

During doze:

- Network access is suspended
- WiFi scan is not performed
- Standard alarms and periodic activities are deferred
- Wake-locks are ignored



Device is stationary, not connected to a power source and screen is off

Doze mode activated

Maintenance window

Device exits from doze mode when moved, connected to a charger, or screen turned on

App standby

- Apps are assigned to priority buckets:
  - *Active*: the app is currently used (or it has been used very recently);
  - *Working set*: regularly used;
  - *Frequent*: often used, but not daily;
  - *Rare*: not frequently used.
- When the device is not connected to a power supply, restrictions are imposed to apps depending on their bucket

Bucket	Jobs	Alarms	Network	FCM
Active:	No restriction	No restriction	No restriction	No restriction
Working set:	Deferred up to 2 hours	Deferred up to 6 minutes	No restriction	No restriction
Frequent:	Deferred up to 8 hours	Deferred up to 30 minutes	No restriction	High priority: 10/day
Rare:	Deferred up to 24 hours	Deferred up to 2 hours	Deferred up to 24 hours	High priority: 5/day

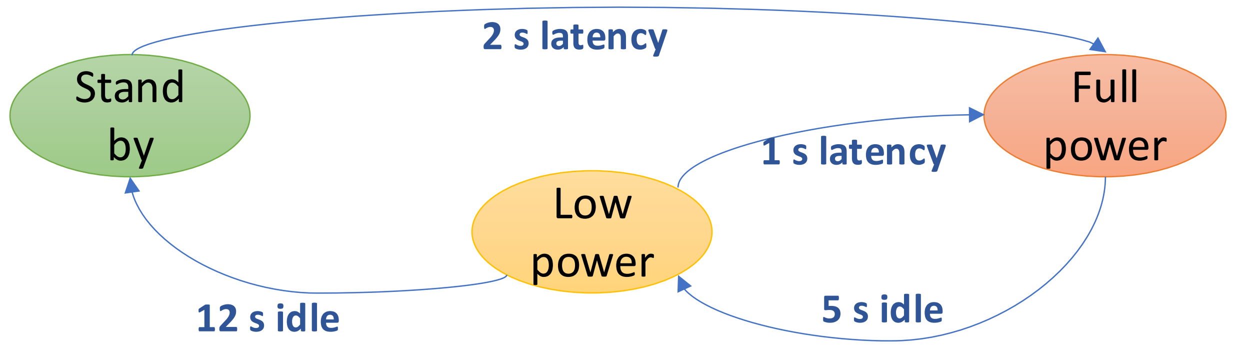
Related concepts

- *Works/Jobs*: operations to be executed when specific conditions are met (e.g. network is available, device is charging, timeout)
- *Alarms*: mechanisms useful to perform time-based operations even when an app is not running.
- *Firebase Cloud Messaging*: a mechanism useful to send messages to android smartphones (e.g. new data on a server is available)

Reducing the energy consumption due to communications

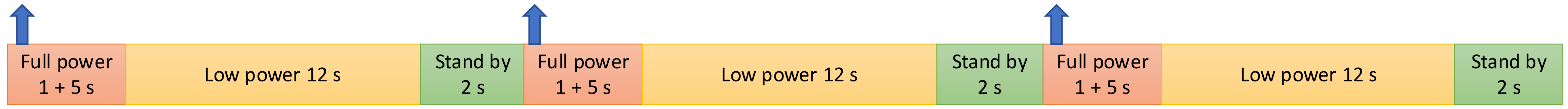
- Communication scheme of apps drives the FSM of cellular interface
- The energy needed by a a 3G radio depends on its state:
  - **Full power**: a connection is active, and data can be sent or received, needs high power
  - **Low power**: data cannot be transferred, power consumption is around 50%, reduced latency to go back to full power
  - **Stand by**: data cannot be transferred, power consumption is minimal, more time to go back to full power

Actual power and transition times depend on the MNO settings

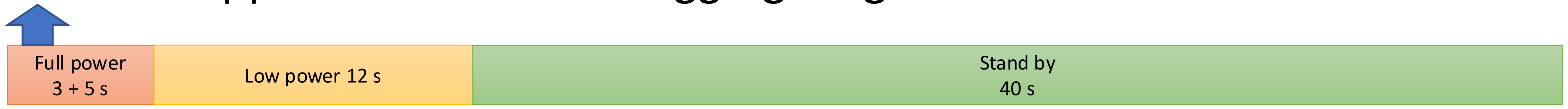


Reducing the energy consumption due to communications

- Suppose an app transfer data for 1 second, 3 times per minute (latency not considered)



- Same app that takes care of aggregating transfers



Reducing the energy consumption due to communications

- Transfer more data all together, but less frequently
- Can be easy to do when the data transfer is dependent on the app logic
  - sending logs
  - synchronizing a database
- More difficult when transfer depends on user's actions
  - keeping the application responsive is more important than reducing energy
  - not all applications are suitable for such an optimization
- Examples
  - in a blog-like or social network app, do not download each entry separately
  - download a number of entries all together and show them when user scrolls
  - tradeoff between aggregating and downloading unnecessary content (never used)
  - in a video streaming app, download video chunks at regular intervals, prefetch the amount of video the user is going actually view (e.g. few minutes)

Reducing the energy consumption due to communications

- Communication that is driven by app logic
  - use the *WorkManager* class to carry out some operations only when specific conditions are met

```
Constraints constraints = new Constraints.Builder()
        .setRequiredNetworkType(NetworkType.UNMETERED)
        .setRequiresBatteryNotLow(true)
        .build();
WorkRequest request = new
PeriodicWorkRequest.Builder(DownloadSetOfPostsWorker.class, 15, TimeUnit.MINUTES)
        .setBackoffCriteria(BackoffPolicy.LINEAR, 30L, TimeUnit.SECONDS)
        .setInputData(
                new Data.Builder()
                        .putString("URLS_TO_BE_DOWNLOADED", "http://...")
                        .build())
        .build();
WorkManager.getInstance(this).enqueue(request);
```

```java
public class DownloadSetOfPostsWorker extends Worker {
   public DownloadSetOfPostsWorker(@NonNull Context context,
                                   @NonNull WorkerParameters workerParams) {
      super(context, workerParams);
   }

   public Result doWork() {
      String inputData = getInputData().getString("URLS_TO_BE_DOWNLOADED");
      if(inputData == null) {
         return Result.failure();
      }
      // Do the work in other method
      downloadASetOfPosts(inputData);

      // Indicate whether the work finished successfully with the Result
      return Result.success();
      /* or Result.retry(): The work failed and should be retried. */
   }
   private void downloadASetOfPosts(String s) {
      // Actual network operations ...
      Log.i("NETOPT", "inputData: " + s);
   }
}
```

Reducing the energy consumption due to communications

- Other strategies for user's initiated network operations, change the amount of prefetch depending on connection technology

```
private int prefetchSize = 10;
private boolean wifiAvailable = false;
private boolean cellularAvailable = false;

public void go2(View v) {
  if (ActivityCompat.checkSelfPermission(this, Manifest.permission.READ_PHONE_STATE) !=
    PackageManager.PERMISSION_GRANTED) {
    ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.READ_PHONE_STATE}, 1);
    return;
  }
  ConnectivityManager cm =
      (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);
  cm.registerDefaultNetworkCallback(networkCallback);
}
```

```java
private ConnectivityManager.NetworkCallback networkCallback = new ConnectivityManager.NetworkCallback() {
    @Override
    public void onCapabilitiesChanged(
            @NonNull Network network,
            @NonNull NetworkCapabilities networkCapabilities
    ) {
        super.onCapabilitiesChanged(network, networkCapabilities);
        TelephonyManager tm =
                (TelephonyManager) getSystemService(Context.TELEPHONY_SERVICE);

        cellularAvailable = networkCapabilities
                .hasTransport(NetworkCapabilities.TRANSPORT_CELLULAR);
        wifiAvailable = networkCapabilities
                .hasTransport(NetworkCapabilities.TRANSPORT_WIFI);
        if(wifiAvailable) {
            prefetchSize = 10;
            return;
        }
        int nt = tm.getDataNetworkType();
        switch (nt) {
            case TelephonyManager.NETWORK_TYPE_LTE:
                prefetchSize = 8;
                break;
            case TelephonyManager.NETWORK_TYPE_EDGE:
                prefetchSize = 4;
                break;
            default:
                prefetchSize = 1;
                break;
        }

    }
};
```

Reducing the energy consumption due to communications

- Cache data, using the network requires more energy than reading from local storage
- The OkHttp library includes caching
- ... or you can build your app-specific mechanisms

```
private OkHttpClient client;
private Cache cc;

public void go3(View v) {
  if(client == null) {
    Log.i("NETOPT", "Creating client...");
    File cacheDir = getCacheDir();
    long cacheSize = 10L * 1024L * 1024L; // 10 MiB
    cc = new Cache(cacheDir, cacheSize);
    client = new OkHttpClient.Builder()
          .cache(cc)
          .build();
  }

  Request request = new Request.Builder()
          .url("https://publicobject.com/helloworld.txt")
          .build();
```

Reducing the energy consumption due to communications

```
  client.newCall(request).enqueue(new Callback() {    // Remember, no network operations in main thread
    @Override
    public void onFailure(Call call, IOException e) {
      Log.e("NETOPT failure", e.getMessage());
    }
    @Override
    public void onResponse(Call call, Response response) {
      try {
        Log.i("NETOPT", "requests: " + cc.requestCount() + ", hit: " + cc.hitCount() + " , net: " + cc.networkCount());
        ResponseBody responseBody = response.body();
        if (!response.isSuccessful()) {
          Log.e("NETOPT", "Response: " + response);
        }
        Log.i("NETOPT", responseBody.string());
      } catch (Exception e) {
        Log.e("NETOPT", e.getMessage());
      }
    }
  });
}
```

References

- The Linux kernel, https://docs.kernel.org/
- EAS Overview and Integration Guide, ARM