# Power, Energy and Performance

Kunal Korgaonkar
Prof. Tajana Simunic Rosing
Department of Computer Science and Engineering
UCSD

# Motivation for Power Management

- Power consumption is a critical issue in system design today
  - Mobile systems: maximize battery life
  - High performance systems: minimize operational costs

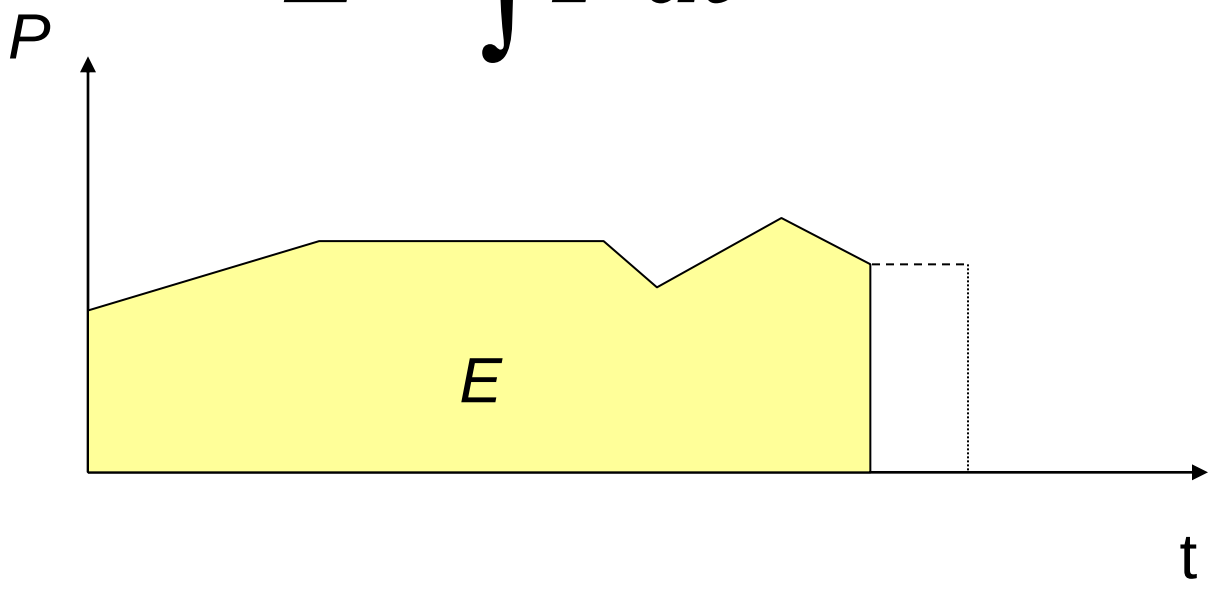


1.2% of total electrical use in US devoted for powering and cooling data centers

$1 operation => $1 cooling

# Power and Energy Relationship

$$E = \int P\,dt$$



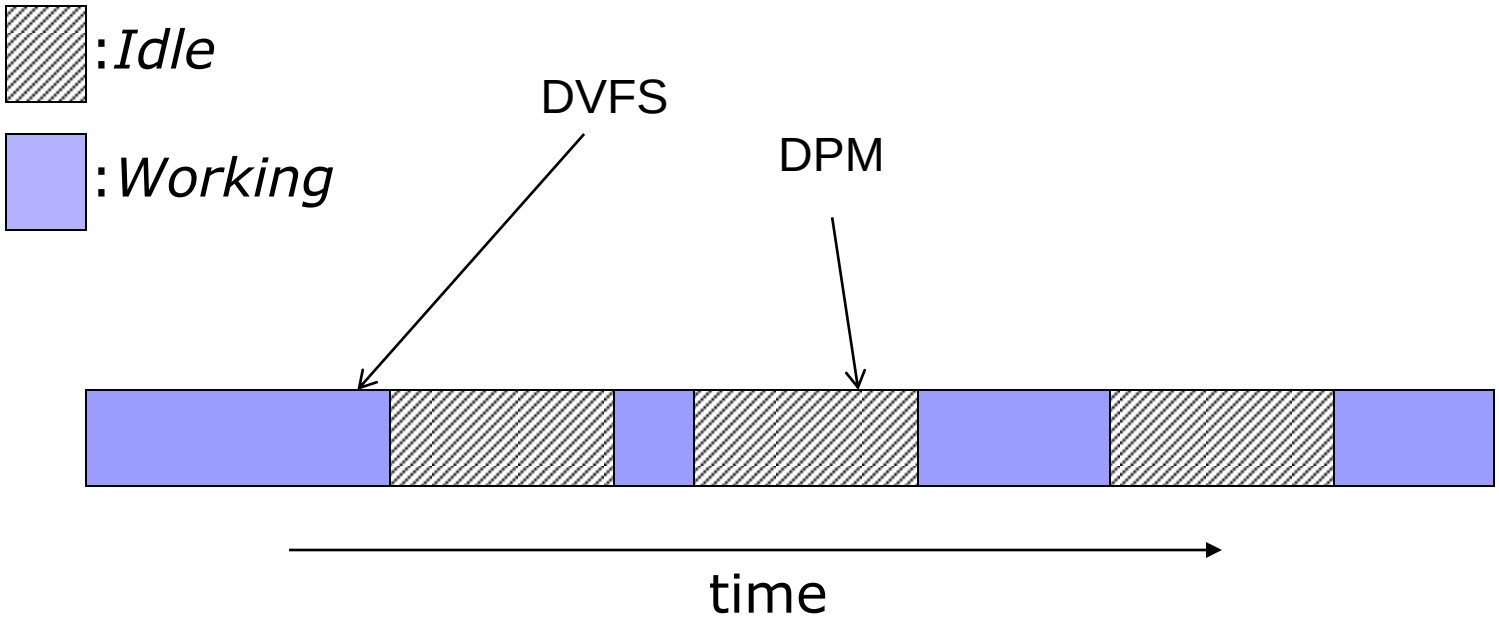
# Low Power vs. Low Energy

- Minimizing the **power consumption** is important for
  - the design of the power supply
  - the design of voltage regulators
  - the dimensioning of interconnect
  - short term cooling
- Minimizing the **energy consumption** is important due to
  - restricted availability of energy (mobile systems)
    - limited battery capacities (only slowly improving)
    - very high costs of energy (solar panels, in space)
  - cooling
    - high costs
    - limited space
  - dependability
  - long lifetimes, low temperatures

# Intuition

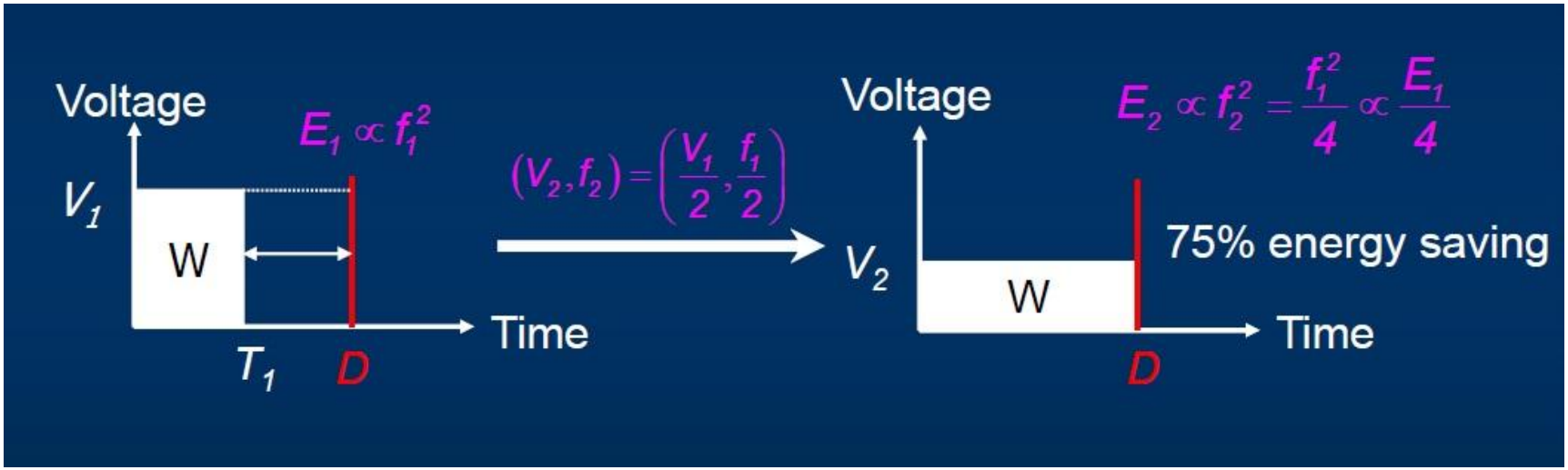
- System and components are:
  - Designed to deliver *peak performance*, but …
  - Not needing peak performance most of the time
- Dynamic Power Management (DPM)
  - Shut down components during idle times
- Dynamic Voltage Frequency Scaling (DVFS)
  - Reduce voltage and frequency of components
- System Level Power Management Policies
  - Manage devices with different power management capabilities
  - Understand tradeoff between DPM and DVFS

# DPM/DVFS




DVFS during active phases and
DPM during idle phases.

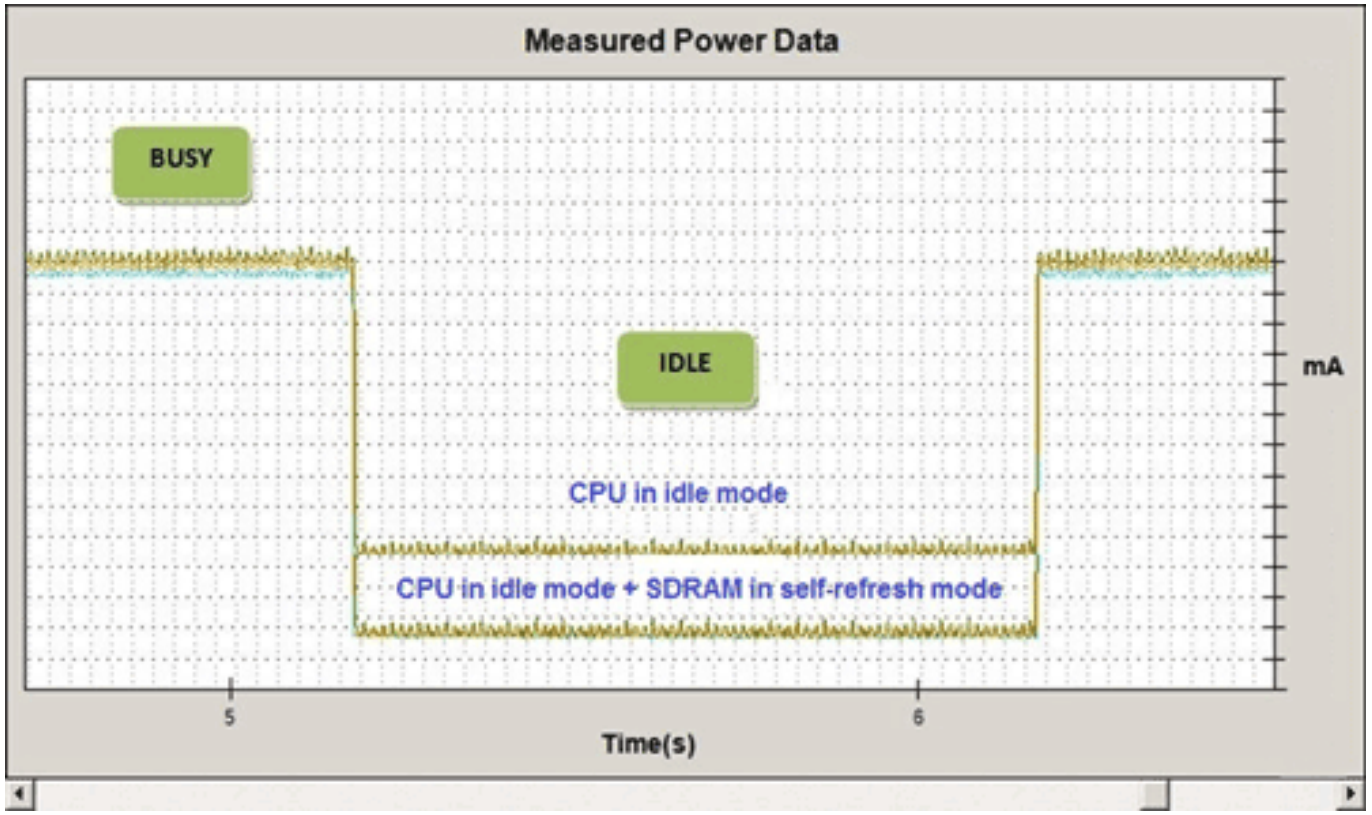
# DVFS




DVFS is an effective way of reducing the CPU energy consumption by providing “just-enough” computation power.

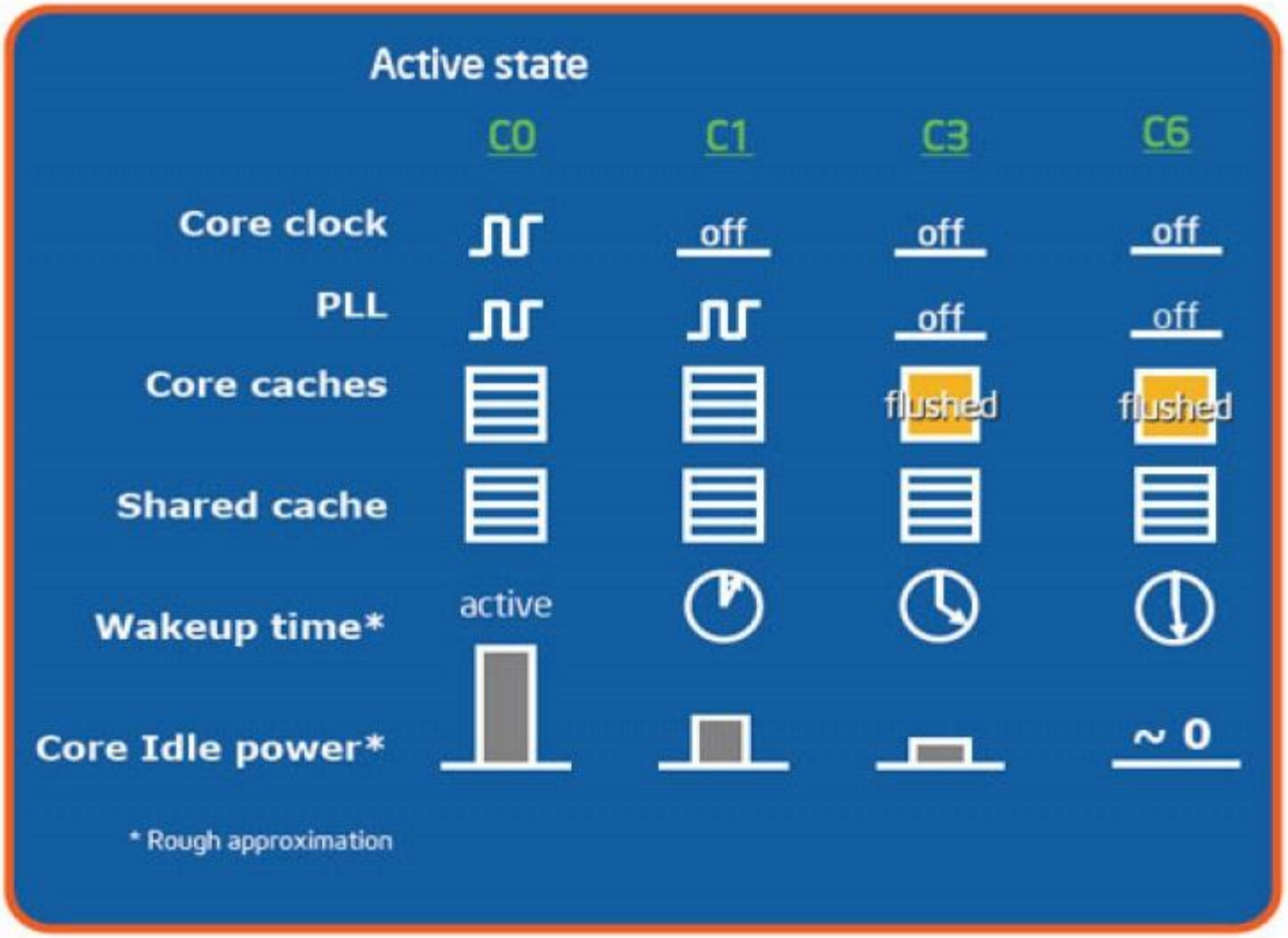
http://atrak.usc.edu/~massoud/Talks/Pedram-dvfs-Taiwan05.pdf

# DPM




DPM is an effective way of reducing the CPU energy consumption by shutting down (completely or partially) when components are not in use.

# Idle States




http://impact.asu.edu/cse591sp11/Nahelempm.pdf

# Power Supply

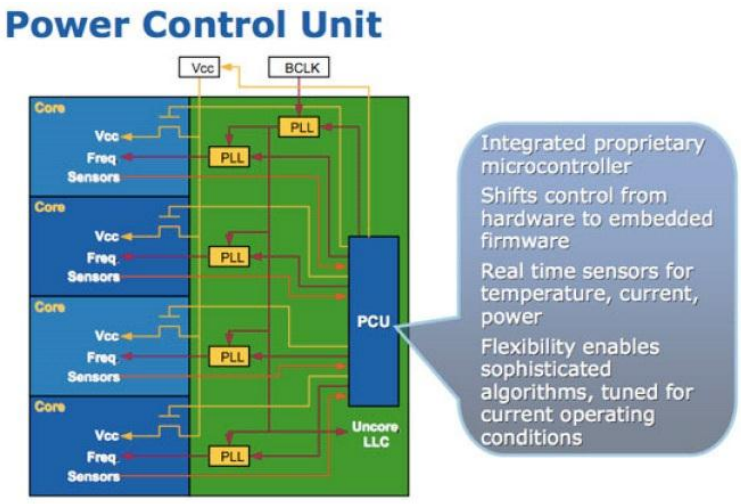
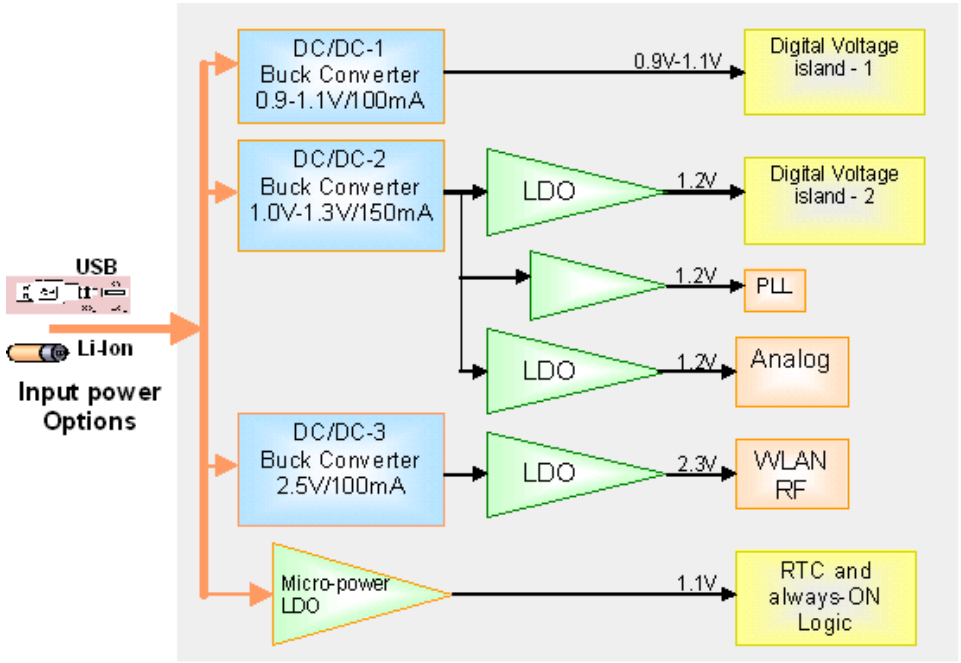


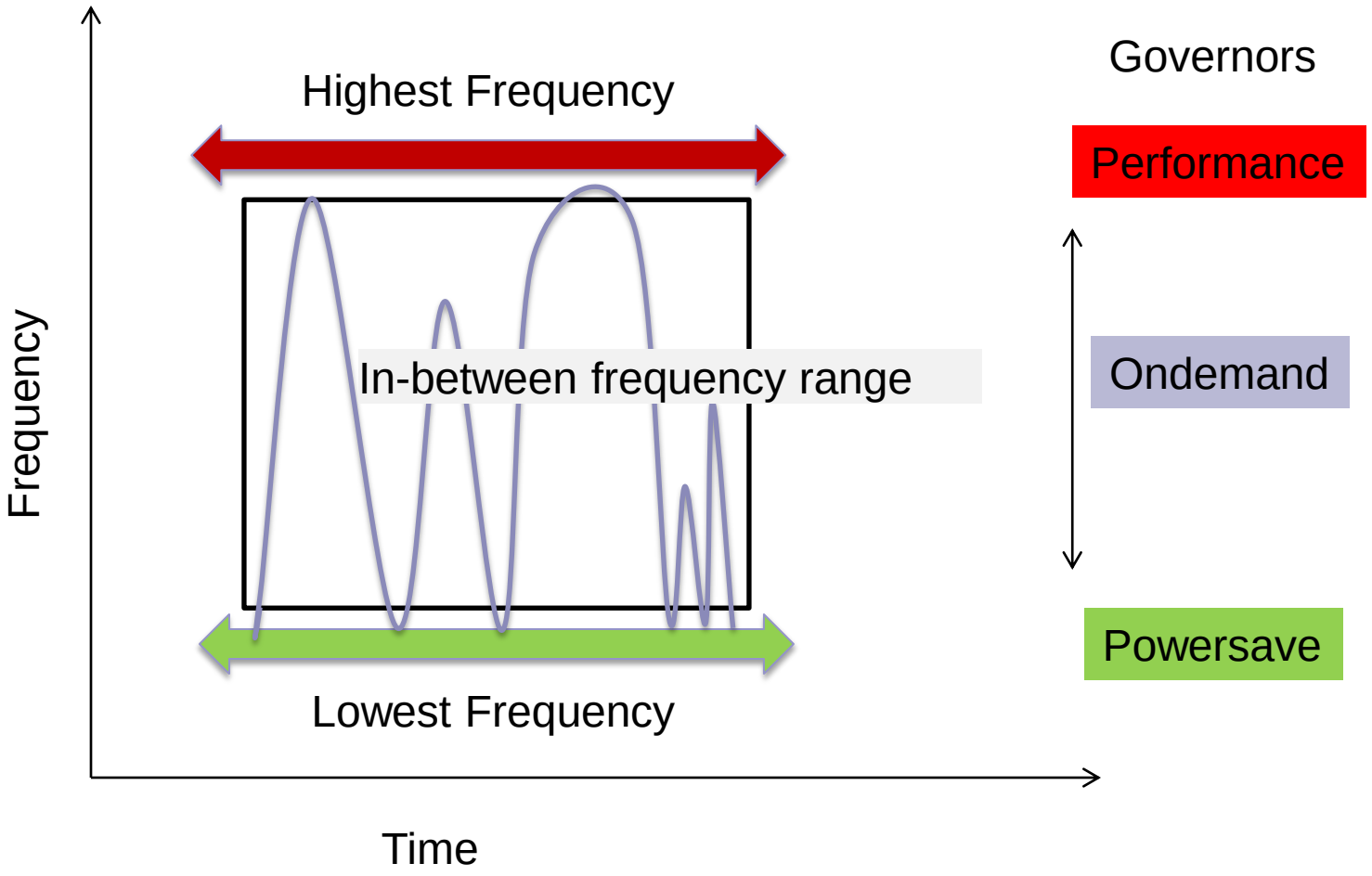

Figure 4: Nehalem's Power Control Unit (PCU)




http://impact.asu.edu/cse591sp11/Nahelempm.pdf
http://www.eetimes.com/design/automotive-design/4012133/Extracting-value-from-integrating-power-management?pageNumber=2&Ecosystem=eda-design

# Android/Linux Power Management and Class Projects

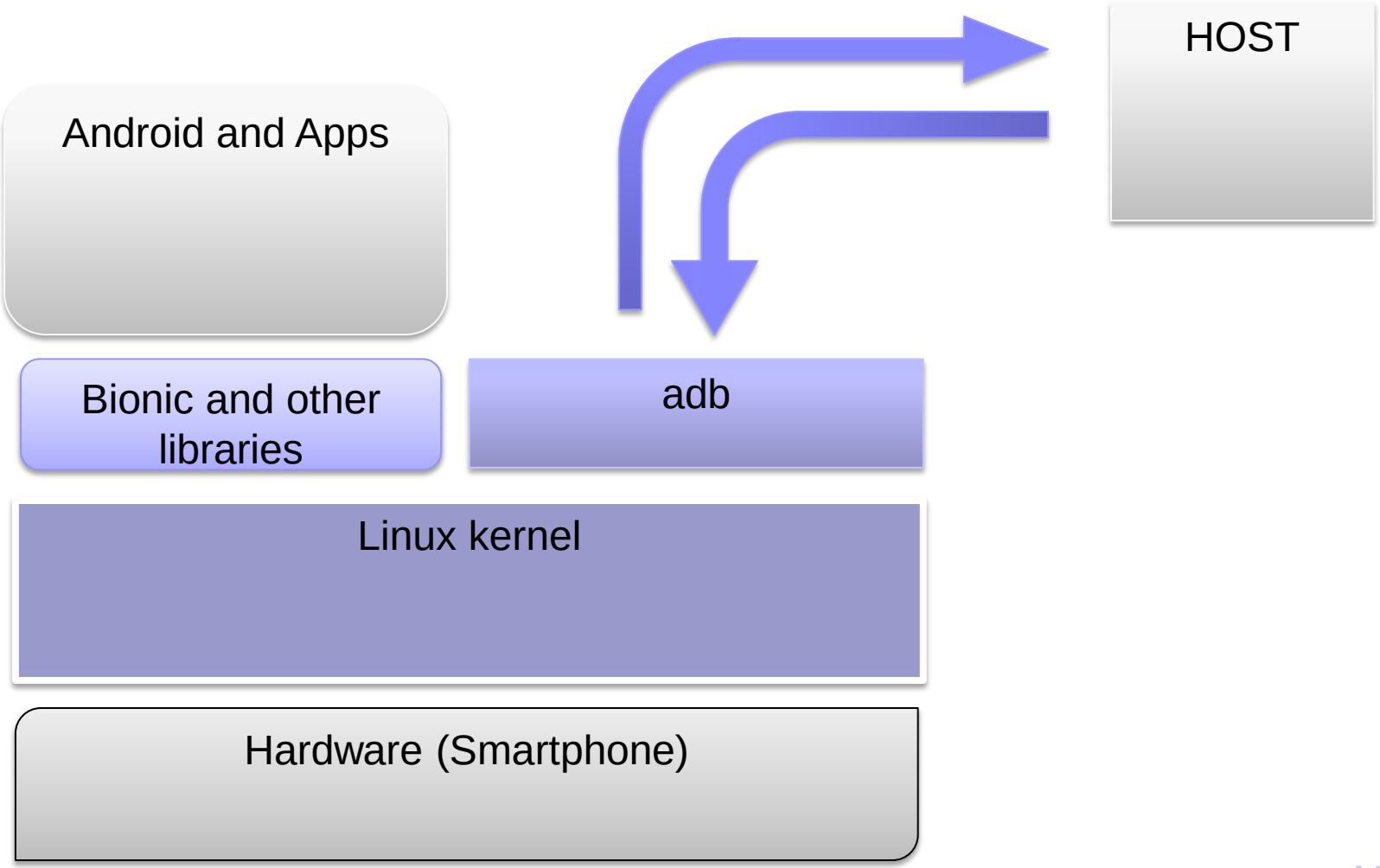
# Frequency Governors



# Project Part 2 – to help you get started

- Given a trace of cpu (IPC) and memory utilization (MPC) finding the best cpu frequency policy
- All on paper (no devices yet)
- Read about cpu frequency governors
  - http://docs.fedoraproject.org/en-US/Fedora/15/html/Power_Management_Guide/index.html
  - http://doc.opensuse.org/documentation/html/openSUSE/opensuse-tuning/cha.tuning.power.html

# Experimental Setup

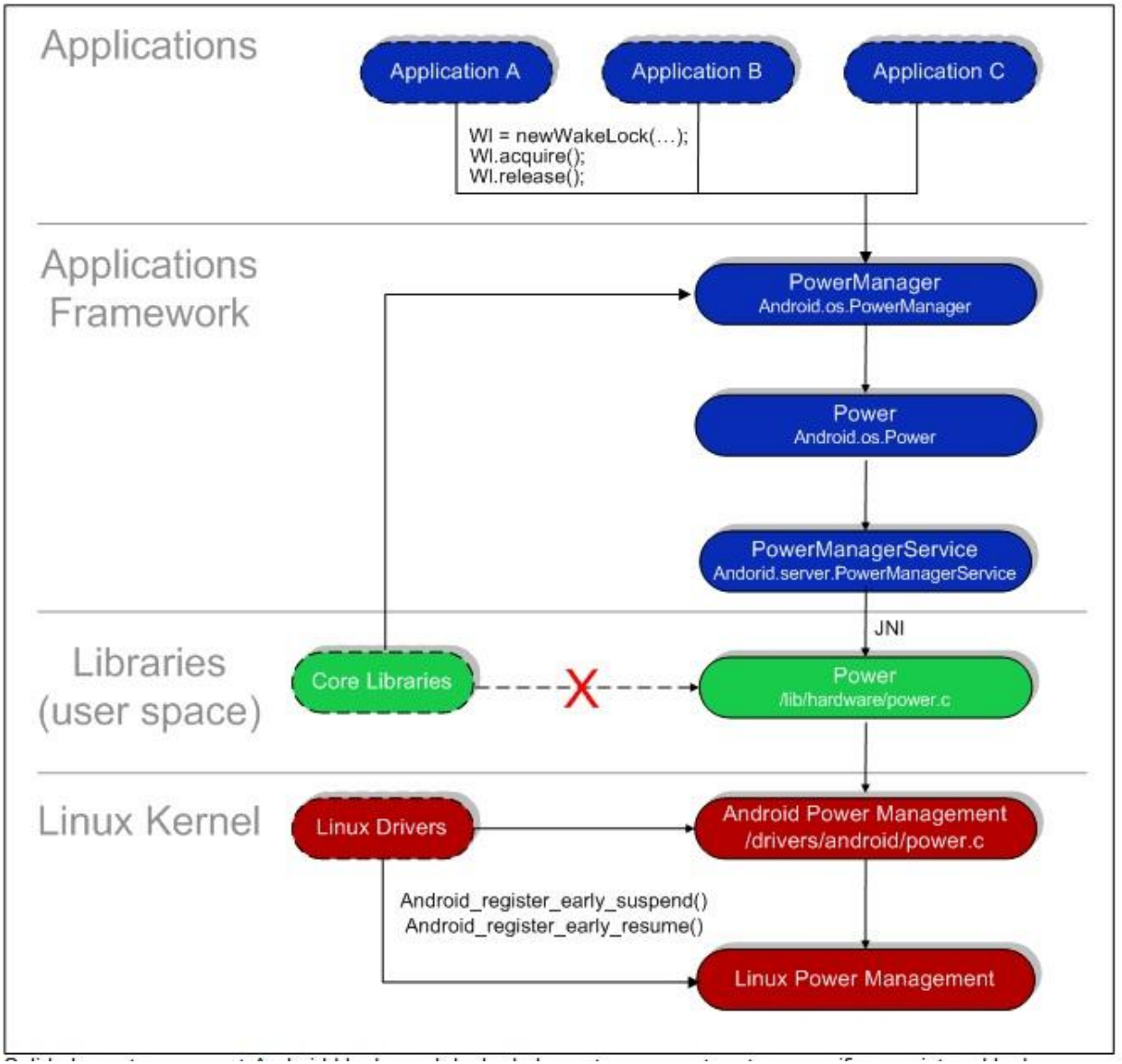


# Android power management

- Android requires that applications and services request CPU (and other hardware resources) with "wake locks" through the Android application framework and native Linux libraries
- Examples
  - WAKE_LOCK_SUSPEND: prevents a full system suspend
  - WAKE_LOCK_IDLE: low-power states, which often cause large interrupt latencies or that disable a set of interrupts, will not be entered from idle until the wake locks are released
  - SCREEN_BRIGHT_WAKE_LOCK: Wake lock that ensures that the screen is on at full brightness; the keyboard backlight will be allowed to go off

http://www.netmite.com/android/mydroid/development/pdk/docs/power_management.html,
http://processors.wiki.ti.com/index.php/TI-Android-GingerBread-2.3.4-DevKit-2.1_PortingGuides#Power_Management

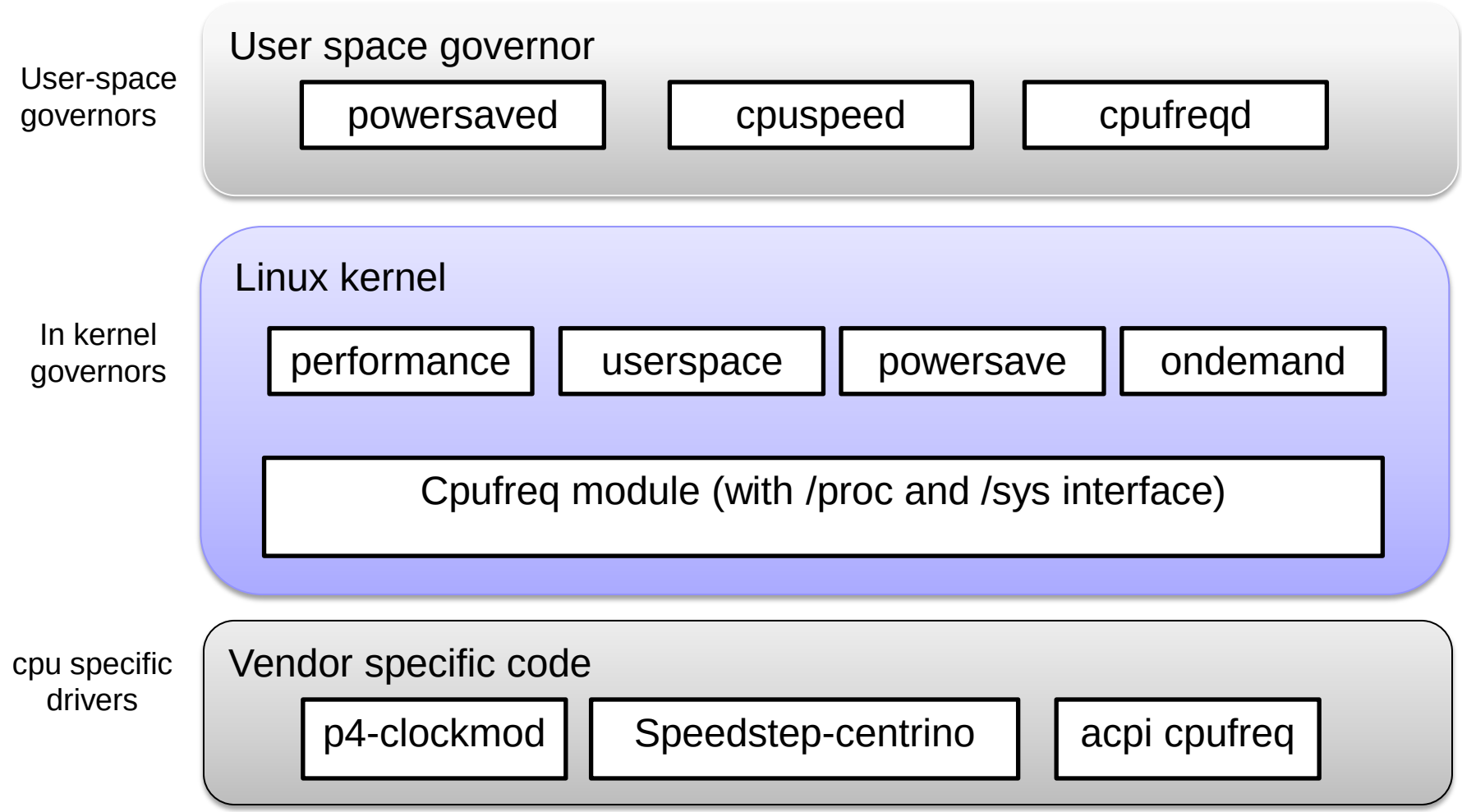
# Android Power Management



# How it works

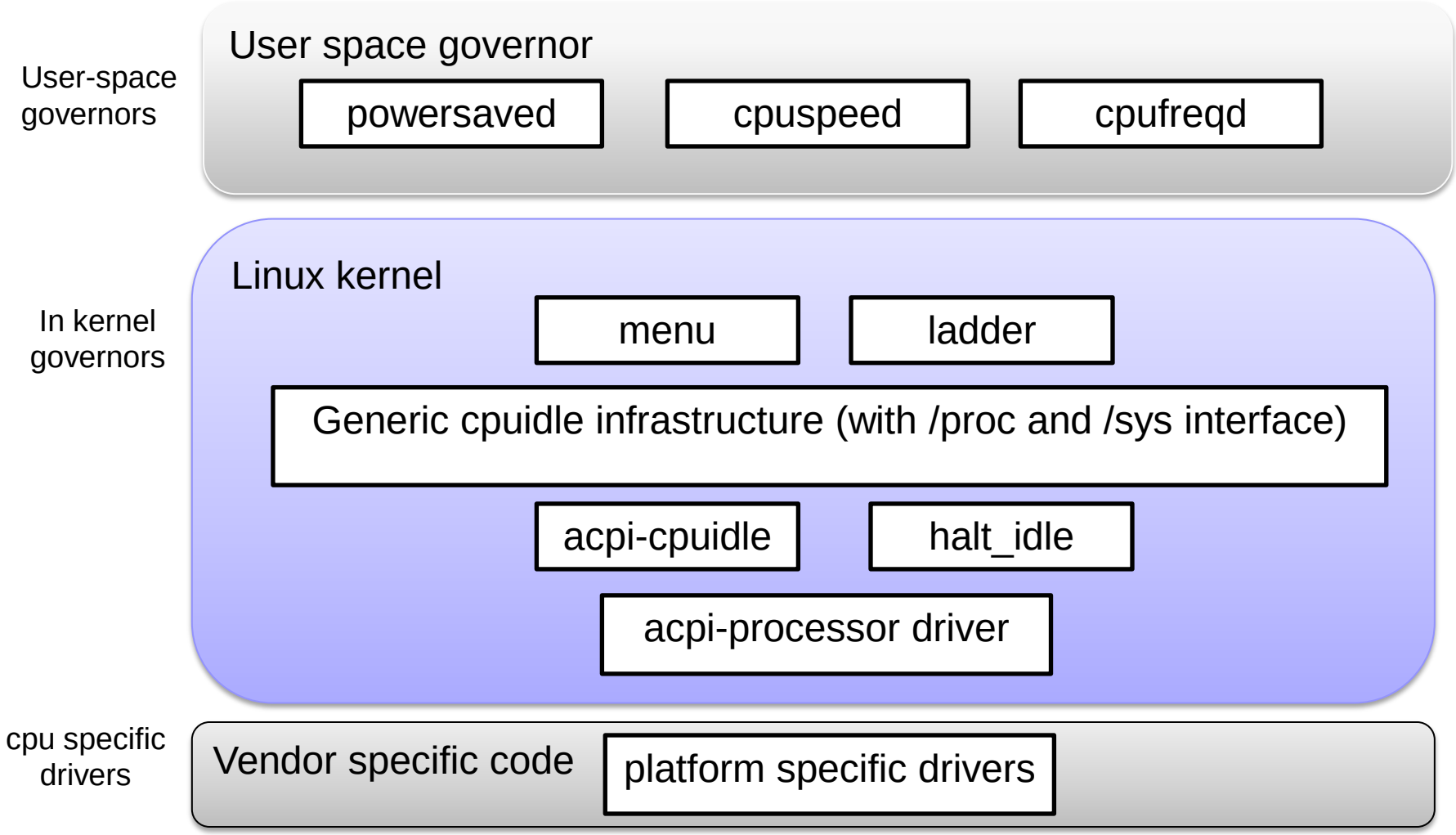
- **PowerManager class:**
  - The Android Framework exposes power management to services and applications through the PowerManager class
- **Registering Drivers with the PM Driver**
  - You can register Kernel-level drivers with the Android Power Manager driver so that they're notified immediately before power down or after power up
  - E.g. Display driver can be registered to completely power down when a request comes in to power down from the user space

# Linux CPU Frequency Subsystem




http://software.intel.com/en-us/articles/enhanced-intel-speedstepr-technology-and-demand-based-switching-on-linux/

# Linux Idle Subsystem




http://software.intel.com/en-us/articles/enhanced-intel-speedstepr-technology-and-demand-based-switching-on-linux/

# Workloads

- SPEC benchmarks
  - Standardized applications
  - But not mobile specific
- Apps
  - Written with real market needs in mind
  - But not standardized
- Micro-benchmarks
  - Can be used to understand/study a specific mobile SOC subsystem
  - Easy to debug

# Future Project Activities (all in parallel)

| |
|---|
| Data collection |
| Data Analysis |
| Understanding and coming up with power management policies |

# Project Activity: Data collection

- Session based
  - 2 to 5 mins
  - Multiple runs needed
  - For all three workloads
- Full ‘working day’ based
  - Full day logs
  - You will be the user

# Part Activity: Data Analysis

- Use logs of real apps whenever possible
- As instructed use
  - SPEC benchmarks and
  - Micro-benchmarks

# Project Activity: Understanding and coming up with your own power management policies

- Coming up with your own frequency governor
- TBA

# What you will learn through these projects

- You will get to learn about
  - Android and Linux kernel
  - Mobile SOC hardware capability
  - Linux CPU frequency subsystem
  - How to use cpu frequency subsystem for power management
  - How benchmarks and real apps behave
  - Understanding performance, power and ‘response time’ trade-offs