

Cinder: Rethinking the Handset Operating System

Stephen Rumble, Ryan Stutsman, Philip Levis, and David Mazières
Computer Systems Lab
Stanford University

UNIX, a Life



The Progression



1971



2009

POSIX



1988



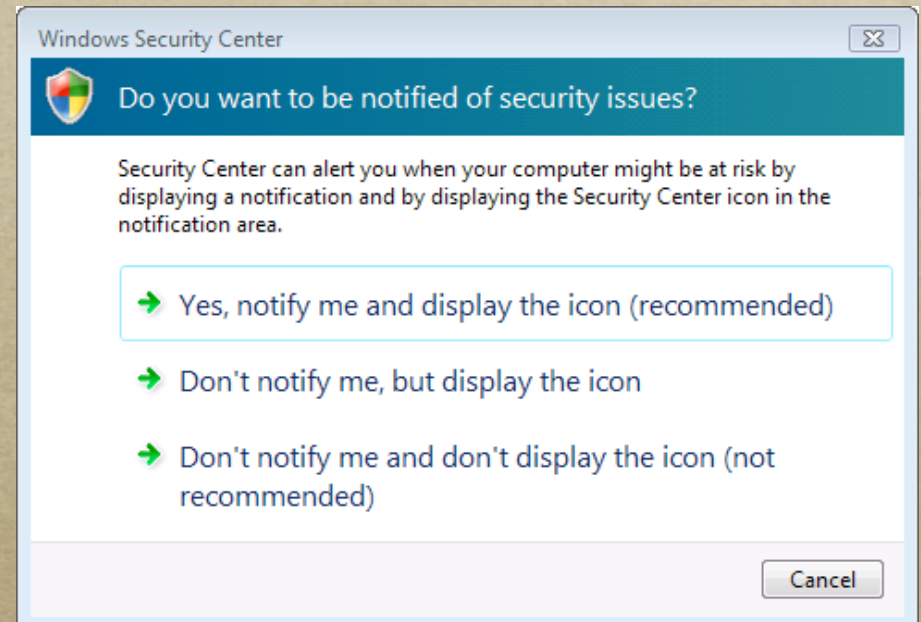
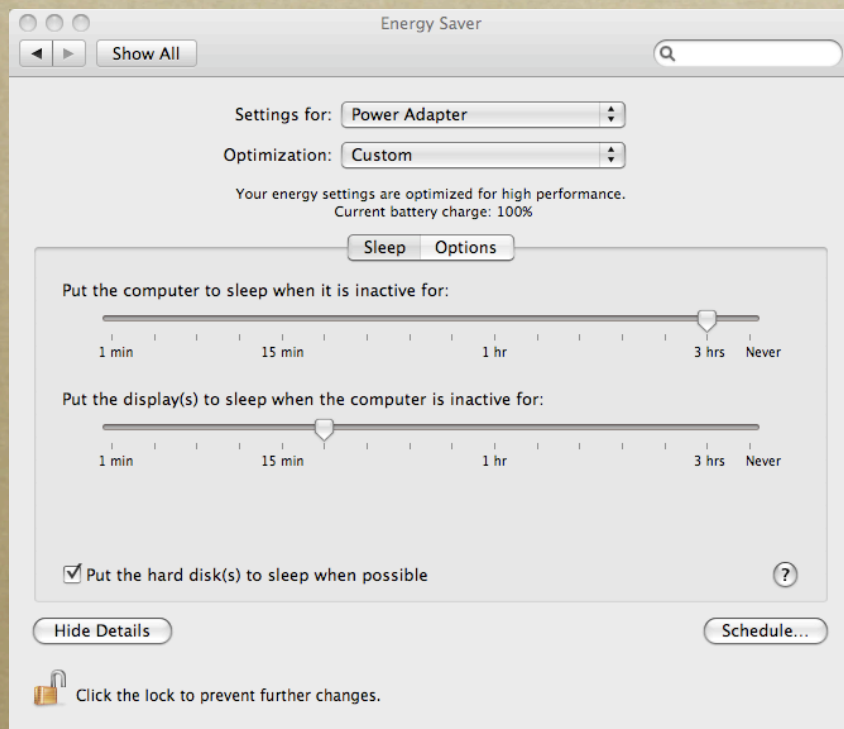
1989

Computing has Changed

- Commodity devices
- Rich, user-centric applications
- Battery-powered
- Untrustworthy programs



The State of the Art Today



Cinder

- A new operating system designed for mobile phones
- Goal: be able to download and run any code without worry
 - Security
 - Solve the morass of current approaches
 - Track data, not code
 - Enables simple and easy user policies
 - Energy Efficiency
 - Allow the OS to energy-limit applications
 - Allow programs to see, in detail, where energy is going
- Start with a clean design, then seek backwards compatibility

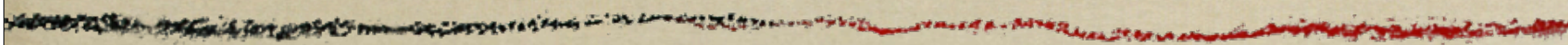


Cinder

- A new operating system designed for mobile phones
- Goal: be able to download and run any code without worry
 - Security
 - Solve the morass of current approaches
 - Track data, not code
 - Enables simple and easy user policies
 - Energy Efficiency
 - Allow the OS to energy-limit applications
 - Allow programs to see, in detail, where energy is going
- Start with a clean design, then seek backwards compatibility

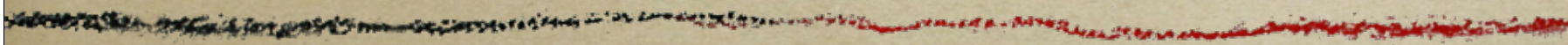


Framing the Problem



- Energy efficiency
 - Energy efficiency trades off with time to completion
 - Never doing something costs no energy
- Energy accounting
 - Track what is spending it
- Energy limiting
 - Limit the energy consumption of a particular application
 - Protects system lifetime from bad applications

Energy Efficiency



Device Control

- Primary challenge is knowing the future workload
 - Do you power down a device and later spend the cost for warm-up?
- Made more difficult by traditional I/O pattern
 - Control returns to application after I/O, when it might request again

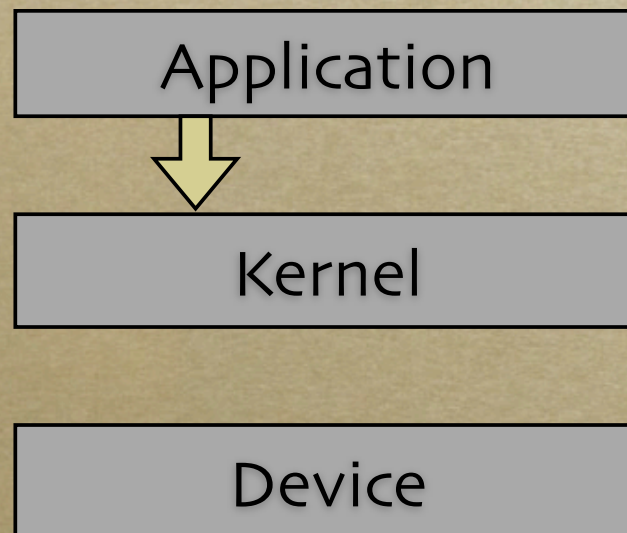
Application

Kernel

Device

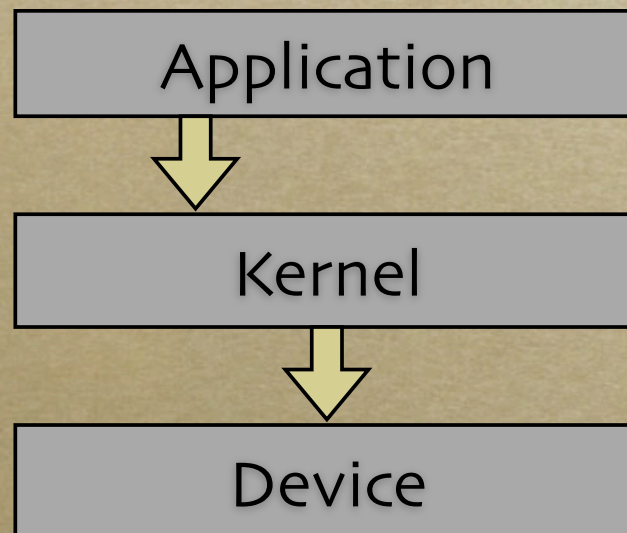
Device Control

- Primary challenge is knowing the future workload
 - Do you power down a device and later spend the cost for warm-up?
- Made more difficult by traditional I/O pattern
 - Control returns to application after I/O, when it might request again



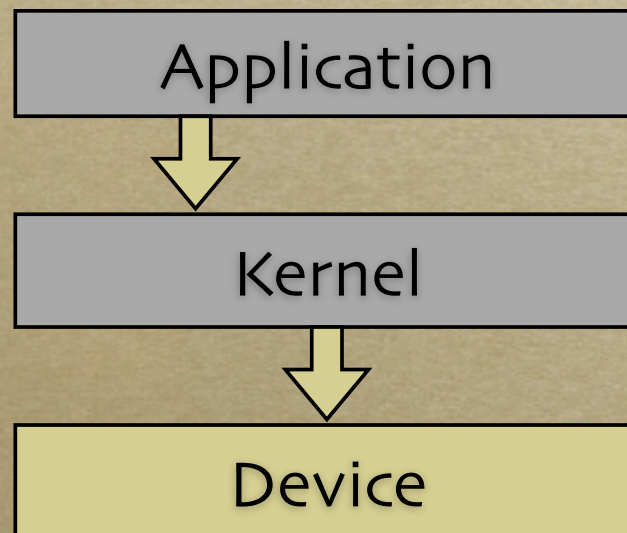
Device Control

- Primary challenge is knowing the future workload
 - Do you power down a device and later spend the cost for warm-up?
- Made more difficult by traditional I/O pattern
 - Control returns to application after I/O, when it might request again



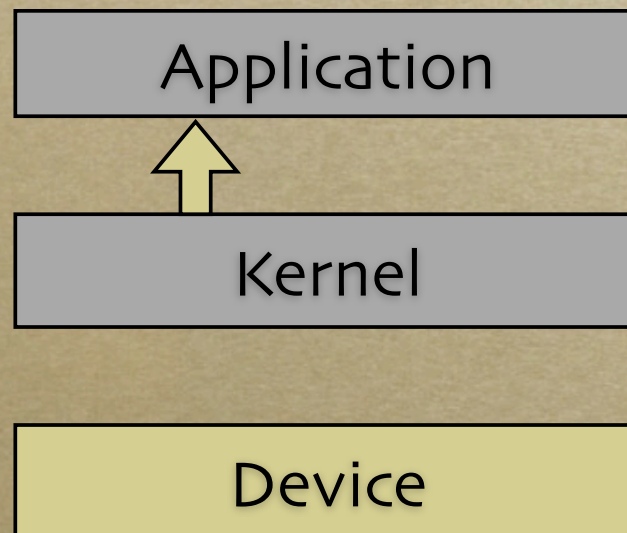
Device Control

- Primary challenge is knowing the future workload
 - Do you power down a device and later spend the cost for warm-up?
- Made more difficult by traditional I/O pattern
 - Control returns to application after I/O, when it might request again



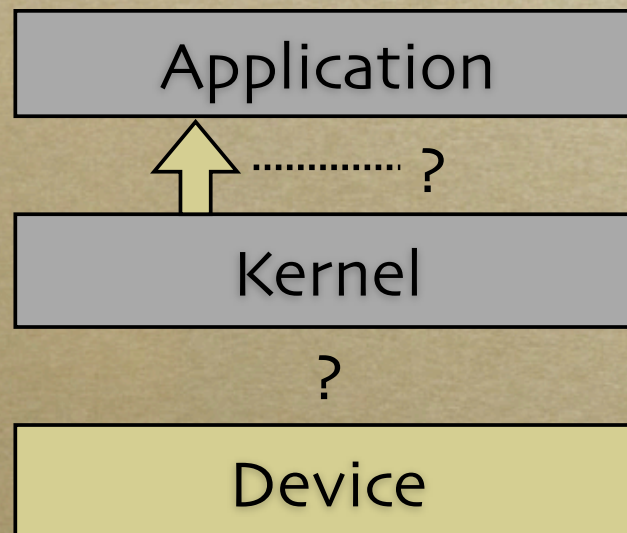
Device Control

- Primary challenge is knowing the future workload
 - Do you power down a device and later spend the cost for warm-up?
- Made more difficult by traditional I/O pattern
 - Control returns to application after I/O, when it might request again



Device Control

- Primary challenge is knowing the future workload
 - Do you power down a device and later spend the cost for warm-up?
- Made more difficult by traditional I/O pattern
 - Control returns to application after I/O, when it might request again

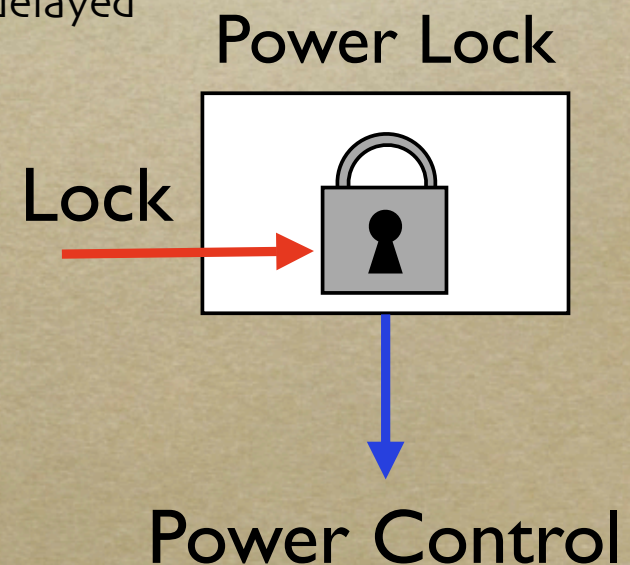


Asynchrony

- Asynchronous I/O: breaks 1-to-1 between threads and I/O
 - Key mechanism for high performance/highly parallel services
 - More I/O requests gives OS better scheduling flexibility
- Asynchronous I/O can improve energy efficiency
 - Given a workload, OS can schedule it
 - OS difficulty is when an energy resource goes idle

Power Locks

- Every device driver ends up with a specialized policy and mechanism for power management
- Insight: locks have knowledge of future workload
- Power locks integrate concurrency and power management
 - When lock falls idle, power down, perhaps delayed
- Power locks simplify device drivers
 - Single, general mechanism
- SOSP 2007



Energy Accounting

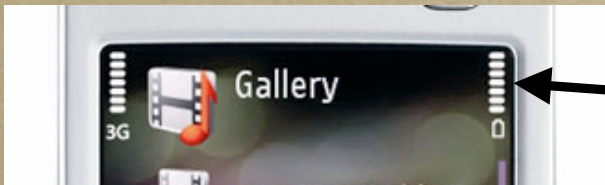
(Fonseca, Dutta, Levis and Stoica, OSDI 2008)

A Personal Story



Black Box

- No way for users or applications to account for energy use
- “What’s expensive” is a guessing game

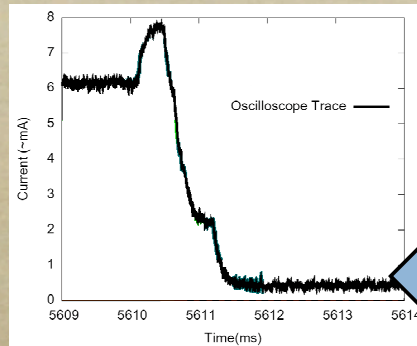


3 bits of information!

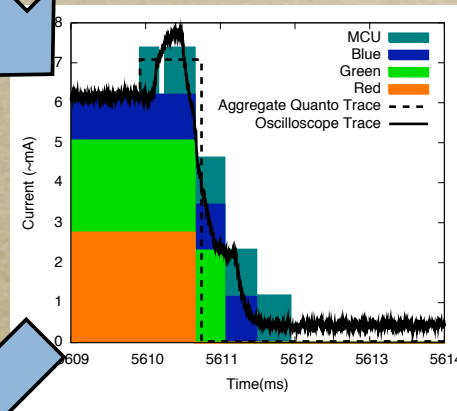
Accounting (for wireless sensors)

(OSDI 2008)

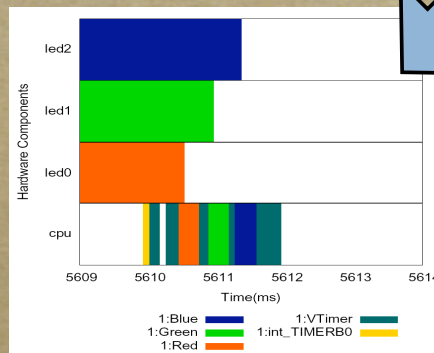
Metering



Breakdown

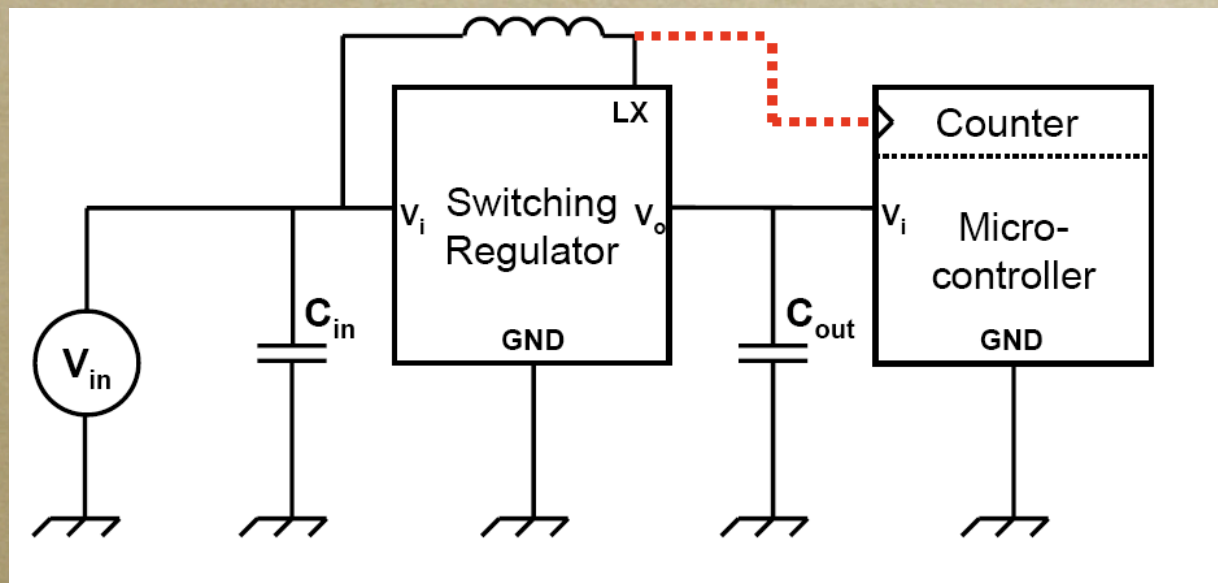


Accounting

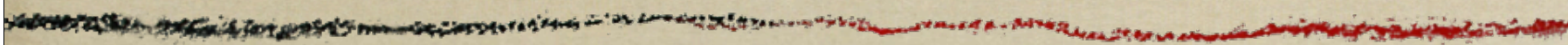


Metering

- Metering: iCount, using a PFM switching regulator
 - A single wire
 - Almost zero cost (24 instructions to sample, $1\mu\text{s}$ accuracy)



Virtual Metering



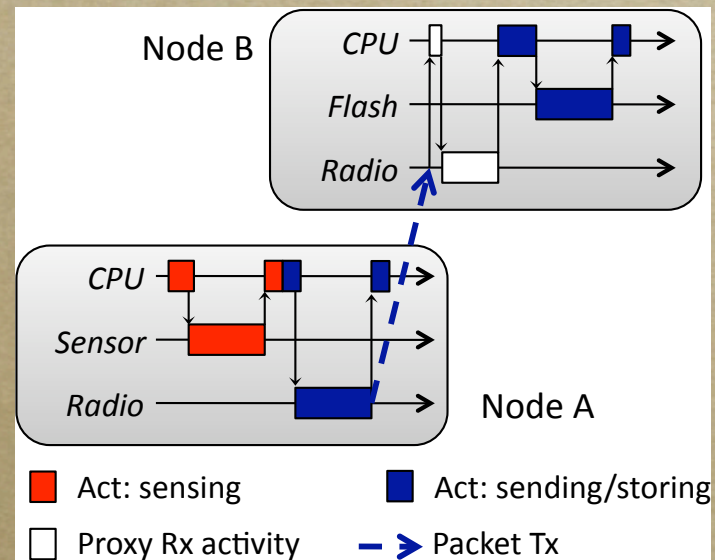
- Suzanne Rivoire (Sonoma State University) and Christos Kozyrakis (Stanford University)
- Use internal system performance counters
- Derive a model of energy consumption
 - Accurate to within 2% on most processors
- Extend to broader system

Breakdown

- Instrument each energy sink (hardware resource) to track explicit power state
- Log aggregate energy consumed, interval length, and power state of all devices
- Use linear regression to derive the power draw of each device power state

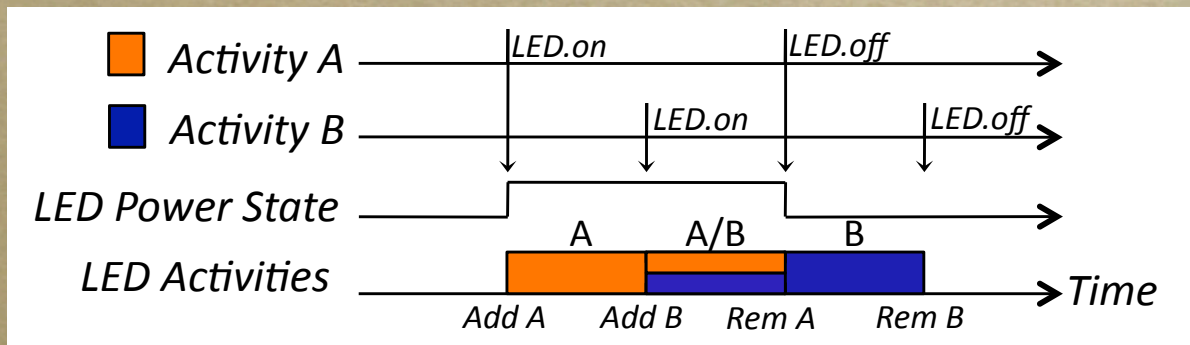
Activity Tracking

- Track in terms of “resource principals”
 - All uses of hardware resources have an associated principal
 - Use labels to track which principal is active
 - Simple API to change principal



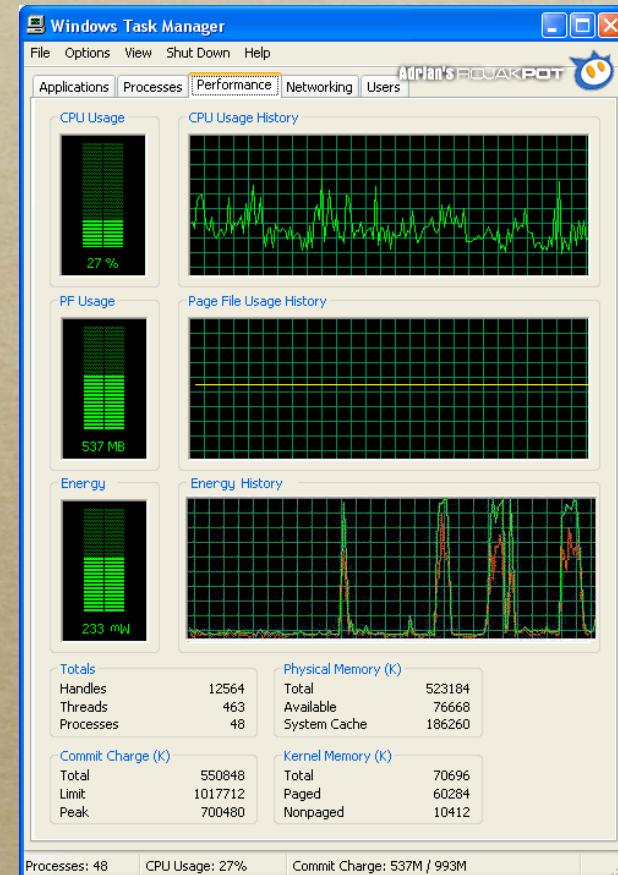
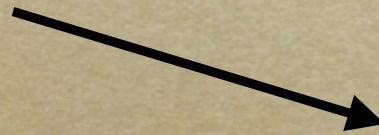
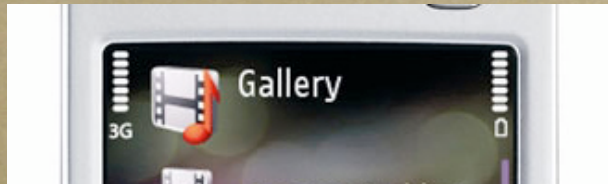
Activity Tracking

- Track in terms of “resource principals”
 - All uses of hardware resources have an associated principal
 - Use labels to track which principal is active
 - Simple API to change principal
- Account on power state or principal change
- Share costs across principals when needed

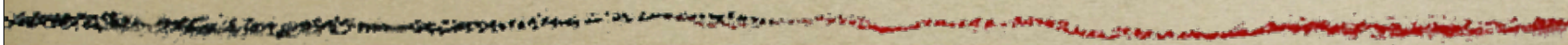


Power Summary

- Power locks to easily manage device power states
- Accurate accounting through metering, linear regression, and state tracking



Energy Limiting



Story, Revisited



Design Considerations

- Need to be able to rate-limit energy consumption
 - Control input power rate
 - Untrustworthy programs can use up entire battery
- Need to allow infrequent burst of high use
 - Build up energy
- Prevent starvation
 - An idle program should not hoard all unused energy

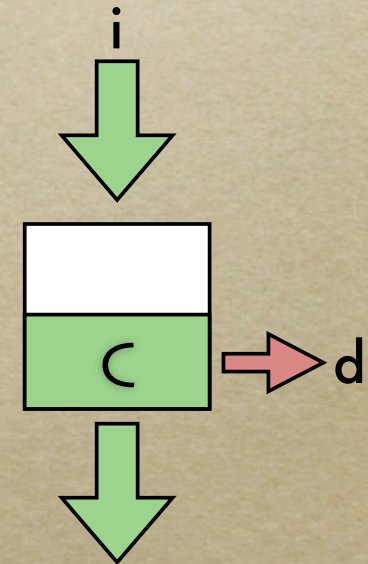
Software Capacitor

- A capacitor C has an energy input i and a decay rate d
- $C = C \cdot d + e$

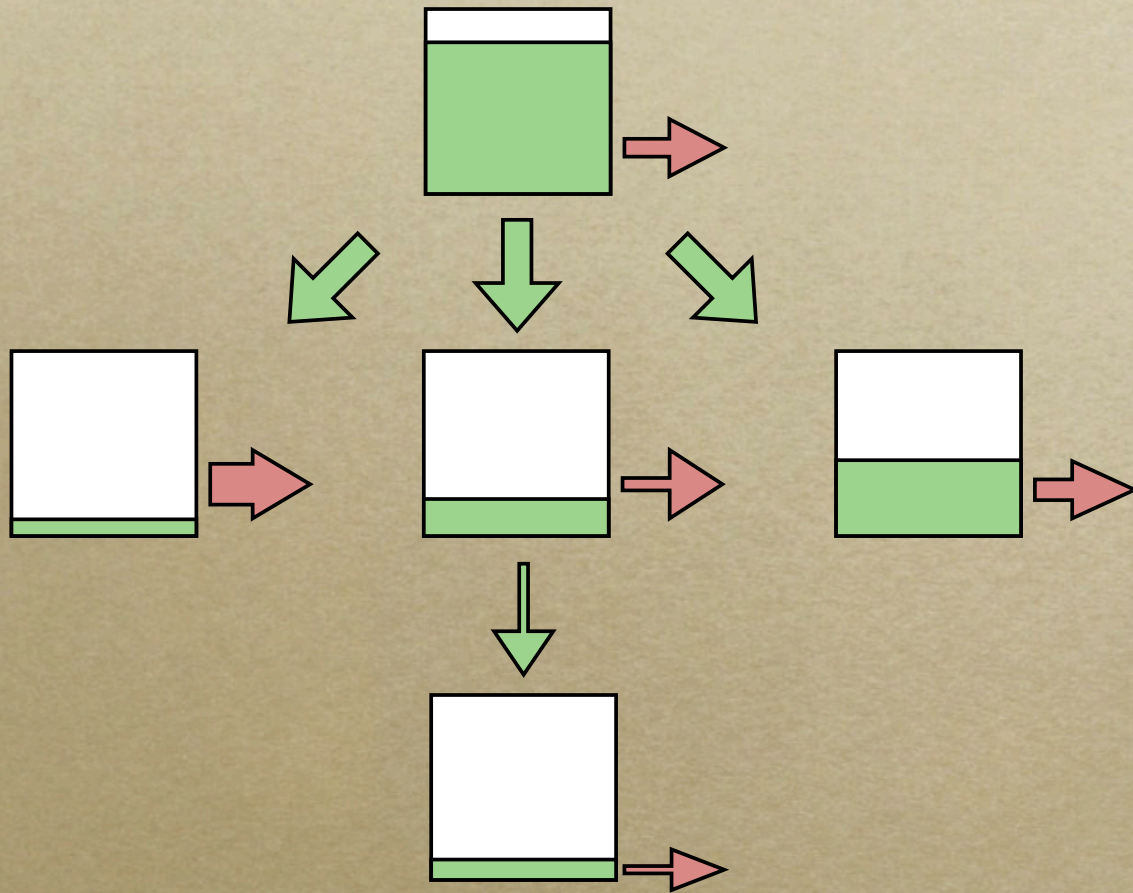
- A capacitor can create sub-capacitors
 - Sum of children inputs must be \leq parent input
 - The OS has a "root capacitor," the battery

- Programs with an empty capacitor do not run
 - A lack of energy manifests as poor performance

- Prevent an app from consuming all the battery

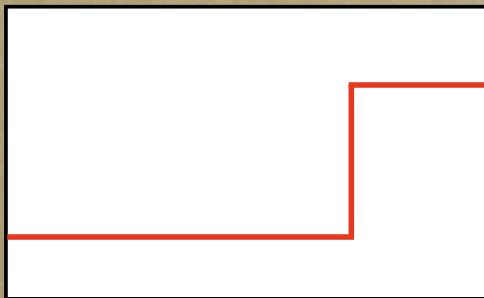


Chaining

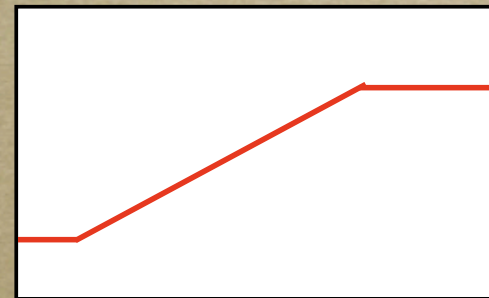


Why Decay?

- Another approach: hard caps
 - After capacitor reaches maximum capacity, all input leaks
- Decay has two advantages
 - Integrates rate with capacity (simpler management)
 - Provides a smoother signal (simpler control systems)



Cap

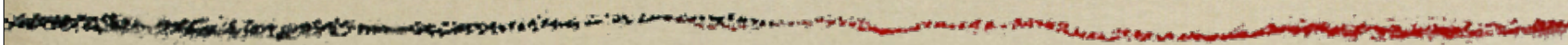


Decay

Task Profiles

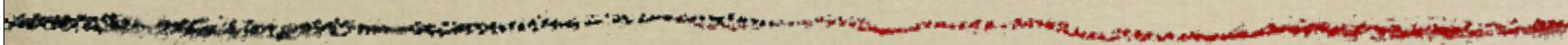
- Capacitors are a mechanism for enforcement
- Need system for policy
- Explicit statement of user intentions
 - "I expect my maps application to last 2 hours"
 - "I expect to be able to watch movies for 4 hours"
- Capacitors provide direct feedback
 - Limits resource use implicitly (e.g., only gets 20% of CPU)
 - Application adapts its performance accordingly

Malicious Code



- Emergence of third-party application markets
 - Walled gardens
- Prevent an application from running down the battery using a task profile

Information Flow



- Energy has implications to security
- Decay rate depends on quantity C
 - Decaying energy is a form of information flow
 - Requires labels/taint
- Decayed energy returns to the root capacitor
 - As input is constant, there is no information flow until system runs out of energy
 - Inputs must therefore be constant: a contract

Steps

- Port HiStar to ARM processors (almost done)
- Add capacitors (done)
 - Fit nicely into HiStar's container hierarchy
- Run HiStar on Android phones
- Incorporate energy accounting into HiStar
 - Metering, virtual metering
- Explore mechanisms
- Explore policies and user interfaces

Cinder

- A new operating system designed for mobile phones
- Goal: be able to download and run any code without worry
 - Security
 - Solve the morass of current approaches
 - Track data, not code
 - Enables simple and easy user policies
 - Energy Efficiency
 - Allow the OS to energy-limit applications
 - Allow programs to see, in detail, where energy is going
- Start with a clean design, then seek backwards compatibility



Questions and Feedback

