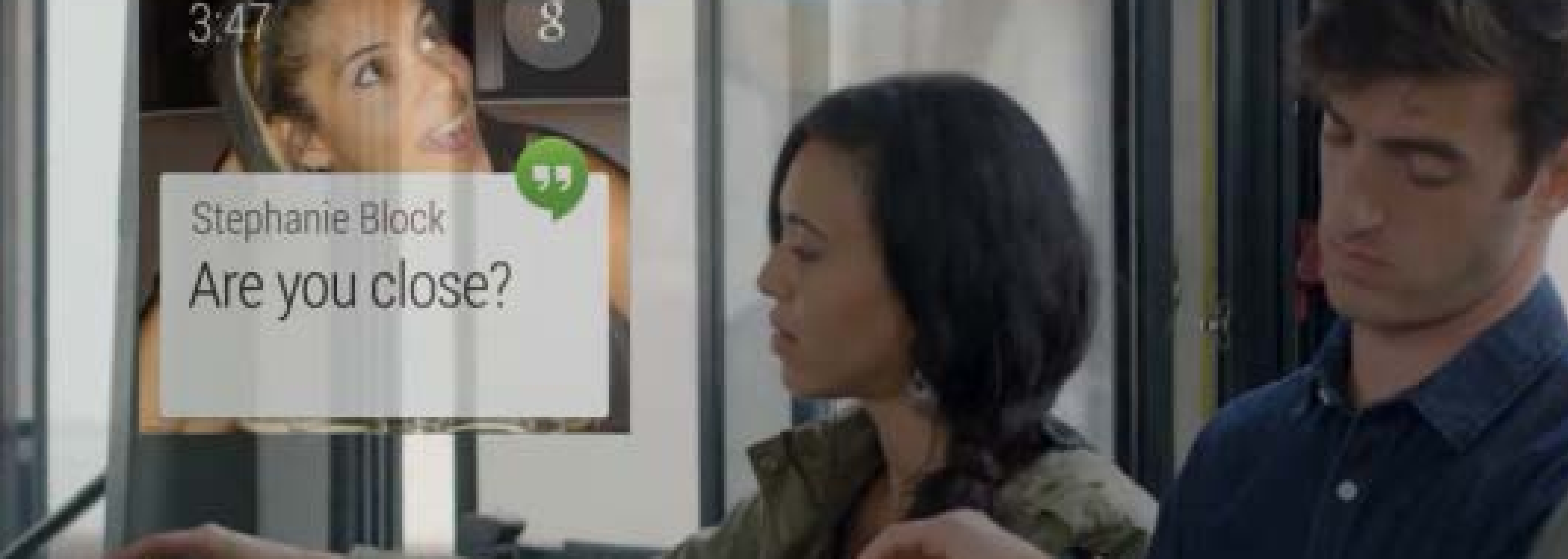
Decelerating Suspend and Resume in Operating Systems

XSEL, Purdue ECE

Shuang Zhai, Liwei Guo, Xiangyu Li, and Felix Xiaozhu Lin

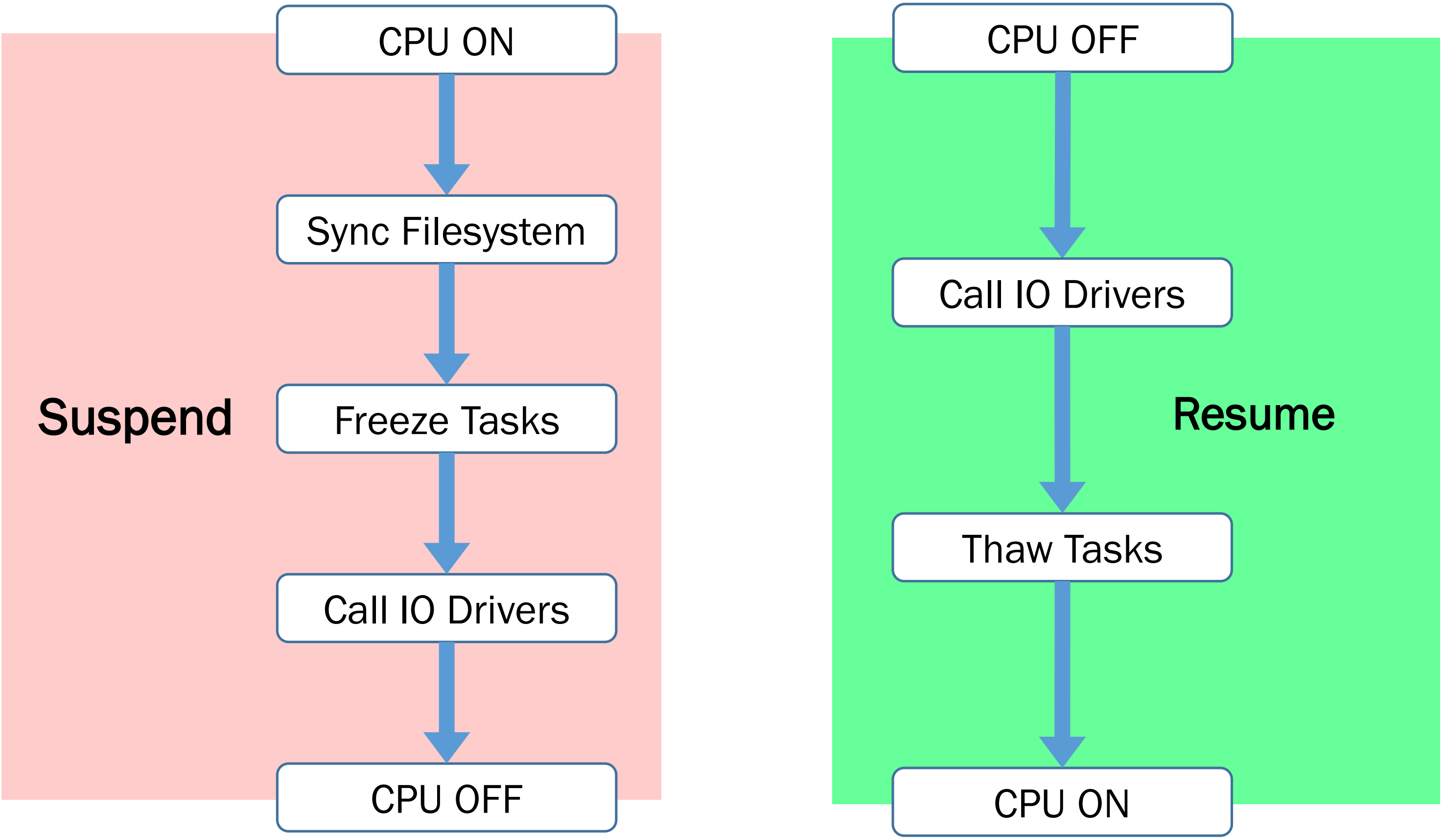
02/21/2017

http://xsel.rocks



- Mobile/IoT devices see many short-lived tasks
  - Sleeping for a long time; Waking up frequently
    - Smartwatch: 72 times per day
  - Each task is short-lived
    - Smartwatch: 10 secs
    - Background task: < 1 sec

Suspend/Resume OS Workflow



Suspend/Resume Is Expensive

- Slow suspend/resume is long known for desktop/server
  - Suspend/resume mostly slowed down by SATA and USB devices
  - These machines suspend/resume only occasionally

- Much worse on mobile/IoT due to **short-lived tasks**
  - Suspend/resume takes ~500 ms on Samsung Note4 Smartphone
  - E.g. consume 43% of total energy on sensing benchmark[1]

- Need to understand suspend/resume on mobile/IoT devices

1. M. Lentz, J. Litton, and B. Bhattacharjee. Drowsy power management. SOSP, 2015

Profiling Suspend/Resume

Nexus 5

Samsung Gear

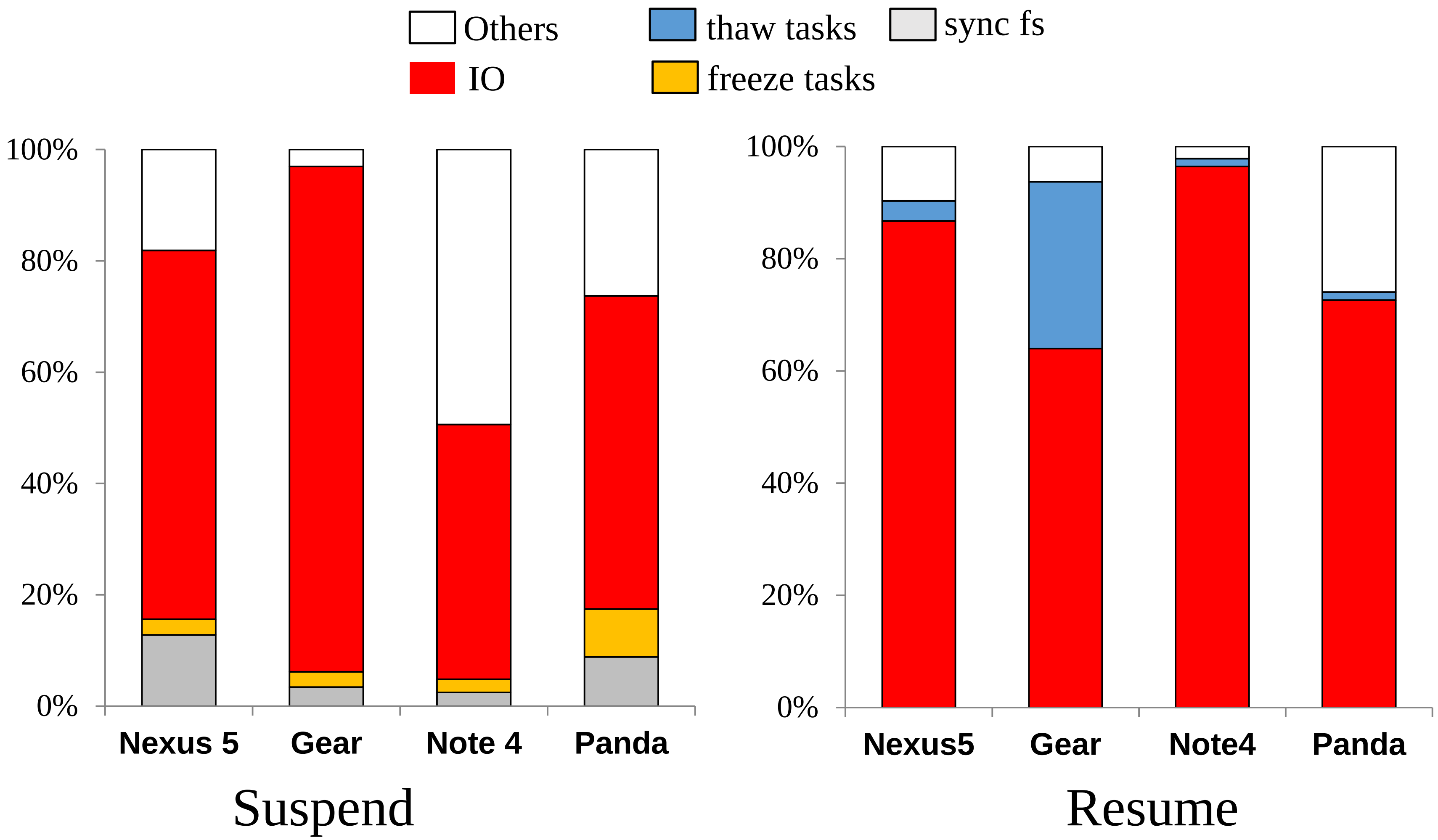
Samsung Note 4

Panda Board

Suspend/Resume on Mobile SoC Is Slow

	Nexus 5	Gear	Note 4	Panda
Suspend	119 ms	191 ms	231 ms	262 ms
Resume	88 ms	159 ms	316 ms	492 ms

Main Reason: IO Power Transitions Are Slow



Slow IOs Are Various and Diverse

Nexus 5	Gear	Note 4
serial_hsl mmc_host	mdss mmc_host	pcieh dwmmc2

Top IO devices

Nexus 5	Gear	Note 4
187	120	116

Number of IO drivers for each platform

Alternative Solution: Async IO Power Transitions

- Asynchronous PM <u>overlaps</u> power transitions of multiple IO devices
- Key difficulty: dependencies among hundreds of IO devices
  - Subtle and implicit
  - OS may not know them
- Linux kernel community has a long debate
- Still very conservative about async PM

Our Key idea

Objective

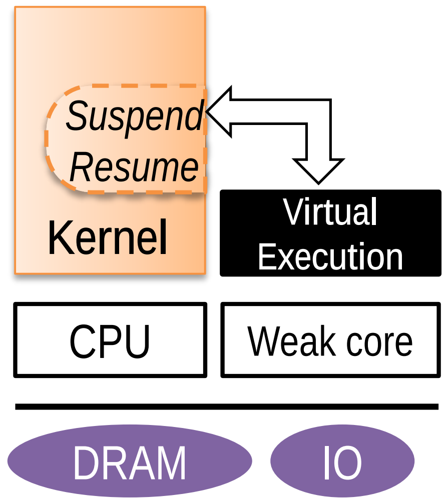
Offloading suspend/resume to a very weak core

Hardware support

A weak core (common on mobile SoCs)

Software support

A baremetal virtual executor on the weak core



Offloading suspend/resume via virtualization

Weak Core on Modern SoCs

- Low power cores already exist on modern SoCs
  - E.g. Apple motion coprocessor (Cortex-M3)
  - Shared memory and IO bus; incoherent cache domain
  - Heterogeneous but similar ISA (ARMv7/8 vs ARMv7m)

Weak Core on Modern SoCs

- Low power cores already exist on modern SoCs
  - E.g. Apple motion coprocessor (Cortex-M3)
  - Shared memory and IO bus; incoherent cache domain
  - Heterogeneous but similar ISA (ARMv7/8 vs ARMv7m)

- Weak cores are ideal for executing OS suspend/resume
  - Idle power 3.8 mW vs 30 mW
  - Kernel execution favors weak cores [1]
    - Small code working set
    - Less predictable control flow

1. J. Mogul, J. Mudigonda, N. Binkert, P. Ranganathan, and V. Talwar. Using asymmetric single-isa cmps to save energy on operating systems. Micro, IEEE, 2008

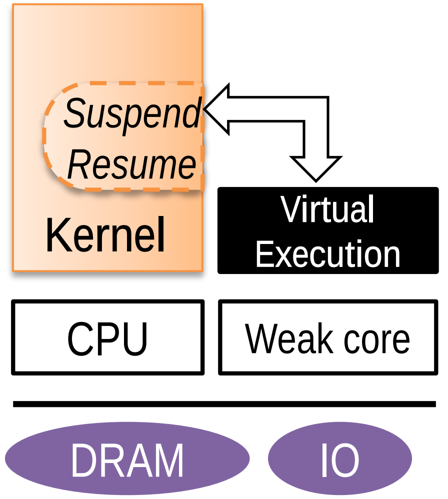
Software Challenges

- Objective: Execute the kernel suspend/resume path on a weak core, without cache coherence and without a unified ISA
- Manually partitioning mature kernels is infeasible
  - Modern kernels are beasts
    - Windows: 45M SLoC[1]
    - Linux 4.4: 16M SLoC[2]
  - Suspend resume code is complicate (30k SLoC in Linux)
- Commodity kernels are rapidly evolving

1. https://www.facebook.com/windows/posts/155741344475532
2. https://www.linuxcounter.net/statistics/kernel

Our Solution

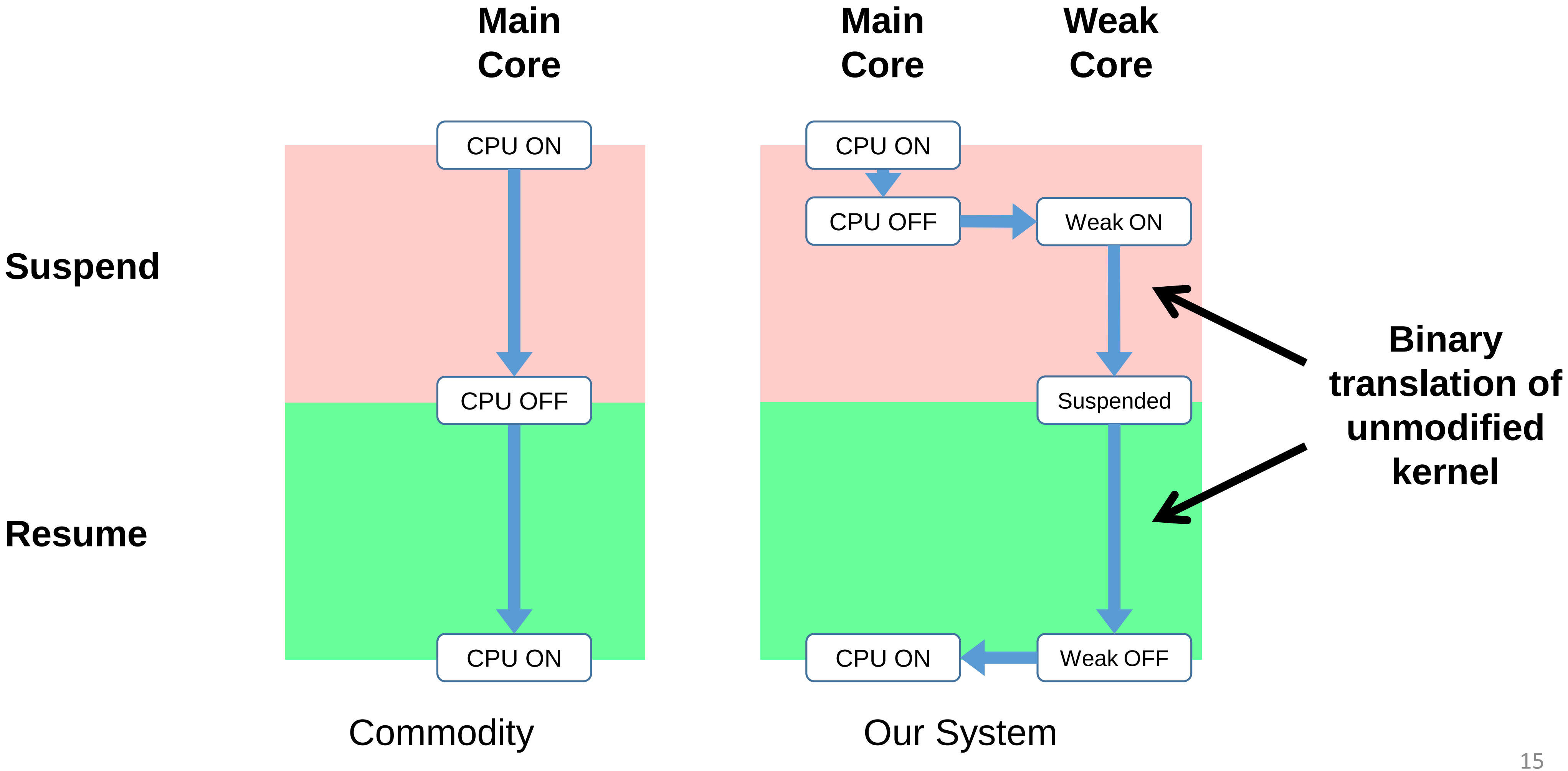
- Launching a virtual machine on the weak core to execute unmodified kernel binary for the main CPU



Offloading suspend/resume via virtualization

- This contrasts with traditional virtualization
  - Host is much more powerful than guest

System Overview



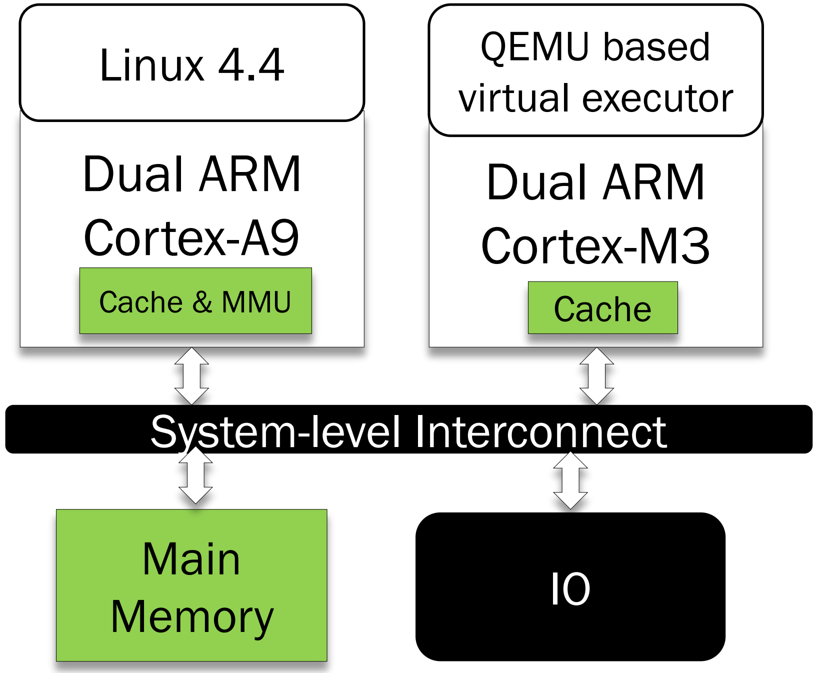
Does This Really Work?

- No one would believe binary translation works for us
  - We need aggressive optimizations
- ~20x slow down from initial implementation
- Reason: commodity binary translators are **generic** and **conservative**
  - Status register is emulated
  - Frequent Interrupt Check

Our Key Optimizations

- Exploit ISA similarity (ARMv7 *vs* ARMv7M)
- Baremetal stacks
- Relaxed handling of interrupts and exceptions
- Kernel virtual memory

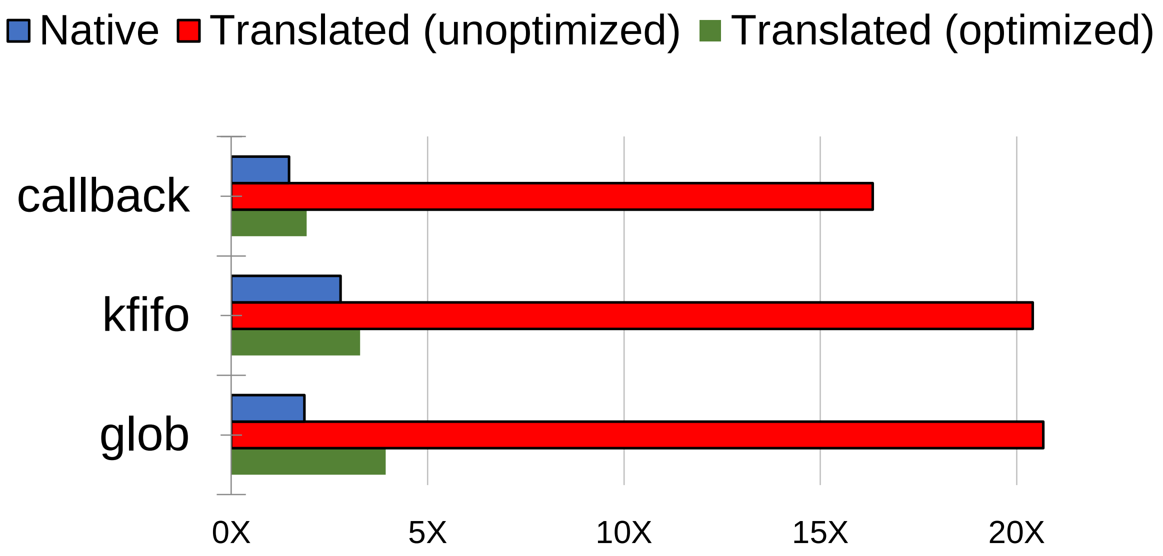
Current Implementation



TI OMAP4

- Platform: TI OMAP4 SoC
- Trimming down QEMU from 2.6M SLoC to 50.5K SLoc
- 4.5K SLoC new code
- A first-of-its-kind virtualization environment on an embedded core

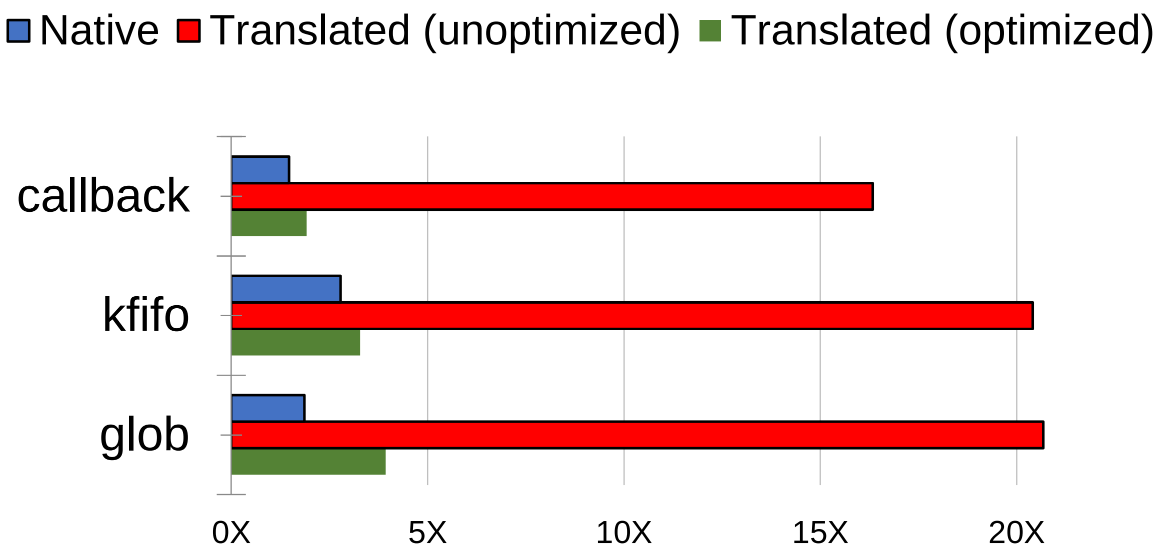
Microbenchmarks



Overhead in terms of cycle count

- **Performance metric:** # of CPU cycles
- **Baseline:**
  - native compilation & execution on the main CPU (Cortex-A9)
- **Native:**
  - native compilation & execution on weak core (Cortex-M3)
- **Translated** (unoptimized/optimized):
  - translated execution on weak core

Microbenchmarks



Overhead in terms of cycle count

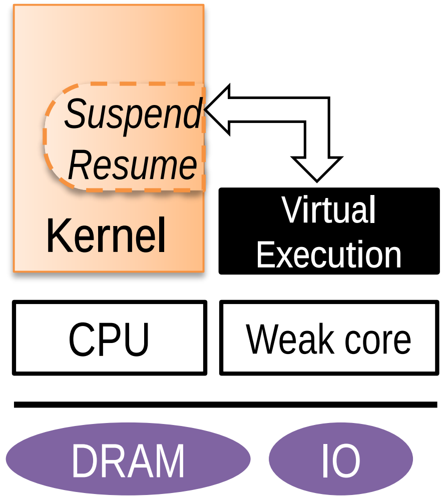
- **Performance metric:** # of CPU cycles
- **Baseline:**
  - native compilation & execution on the main CPU (Cortex-A9)
- **Native:**
  - native compilation & execution on weak core (Cortex-M3)
- **Translated** (unoptimized/optimized):
  - translated execution on weak core

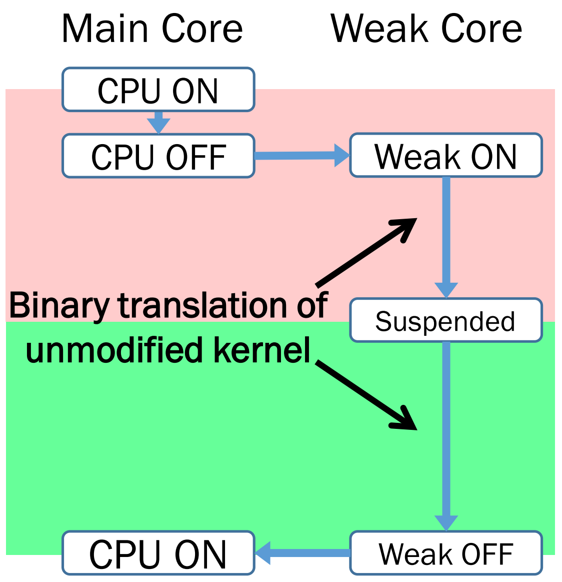
- **Optimization Result:**
  - 5x overhead reduction
  - 2x within native execution
- **Estimated Energy Saving:**
  - 70% energy reduced in suspend/resume
  - 30% overall battery life extended
    - Benchmark: Mobile sensing scenario [1]

1. M. Lentz, J. Litton, and B. Bhattacharjee. Drowsy power management. SOSP, 2015

Summary

- **Observation**: Busy/idle waits for IOs bottleneck OS suspend/resume path
- **Goal**: Offloading suspend/resume to a weak core with incoherent cache and heterogenous ISAs
- **Key idea**: Binary translate and execute **unmodified** kernel on weak core
- **Highlight**: For the first time we run a virtual environment on an embedded core for offloading specific kernel paths





http://xsel.rocks

PURDUE ENGINEERING

Q/A

ARM big.LITTLE

SoC	Little Core Power	Big Core Power	Ratio
Exynos 5430	85 mW (Cortex A7)	750 mW (Cortex A15)	8.8
Exynos 5433	189 mW (Cortex A53)	1480 mW (Cortex A57)	7.8
OMAP 4460	21.1 mW (Cortex M3)	672 mW (Cortex A9)	31.8

Power Consumption Comparison between ARM big.LITTLE and OMAP4

	Performance	Energy	Performance/Energy
A15 (Exynox 5430)	99.69MB/s	19.75mWh	~5.04
A7 (Exynos 5430)	77.93MB/s	10.56mWh	~7.38
A57 (Exynos 5433)	155.29MB/s	27.72mWh	~5.60
A53 (Exynos 5433)	109.36MB/s	17.11mWh	~6.39

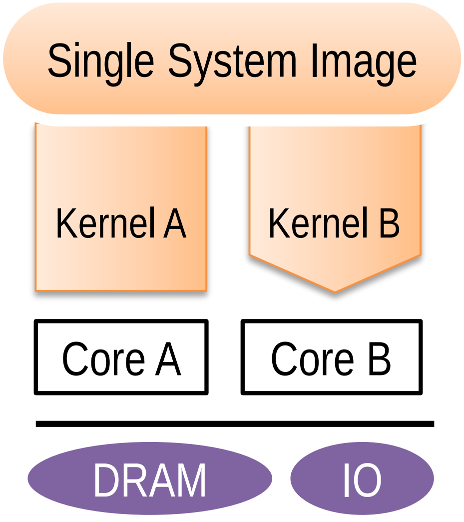
BaseMark OS II - XML Parsing Energy Efficiency

How do we estimate our energy saving

- Without offloading:
  - $E_{cpu} = (T_{busy\_exec} + T_{busy\_wait}) * P_{busy} + T_{idle} * P_{idle}$
- With offloading:
  - $E_{pm} = X * F * T_{busy\_exec} * P'_{busy} + T_{busy\_wait} * P'_{busy} + T_{idle} * P'_{idle}$

Prior Art: Multikernel OSes

- One kernel for each type of cores
  - Helios [1]
  - Barrelfish [2]
  - K2 [3]
  - Popcorn Linux [4]
- Kernels often pass messages to communicate
- They give up compatibility with commodity kernels



A multikernel OS

1. E. B. Nightingale, O. Hodson, R. McIlroy, C. Hawblitzel, and G. Hunt. Helios: heterogeneous multiprocessing with satellite kernels. SOSP, 2009.
2. Andrew Baumann, Paul Barham, Pierre-Evariste Dagand, Tim Harris, Rebecca Isaacs, Simon Peter, Timothy Roscoe, Adrian Schüpbach, and Akhilesh Singhania. The Multikernel: A new OS architecture for scalable multicore systems. SOSP, 2009.
3. F. X. Lin, Z. Wang, and L. Zhong. K2: A mobile operating system for heterogeneous coherence domains. ASPLOS, 2014
4. Antonio Barbalace, Marina Sadini, Saif B.M. Ansary, Christopher Jelesnianski, Akshay Ravichandran, Cagil Kendir, Alastair Murray and Binoy Ravindran,"Popcorn: Bridging the Programmability Gap in Heterogeneous-ISA Platforms“. EuroSys, 2015