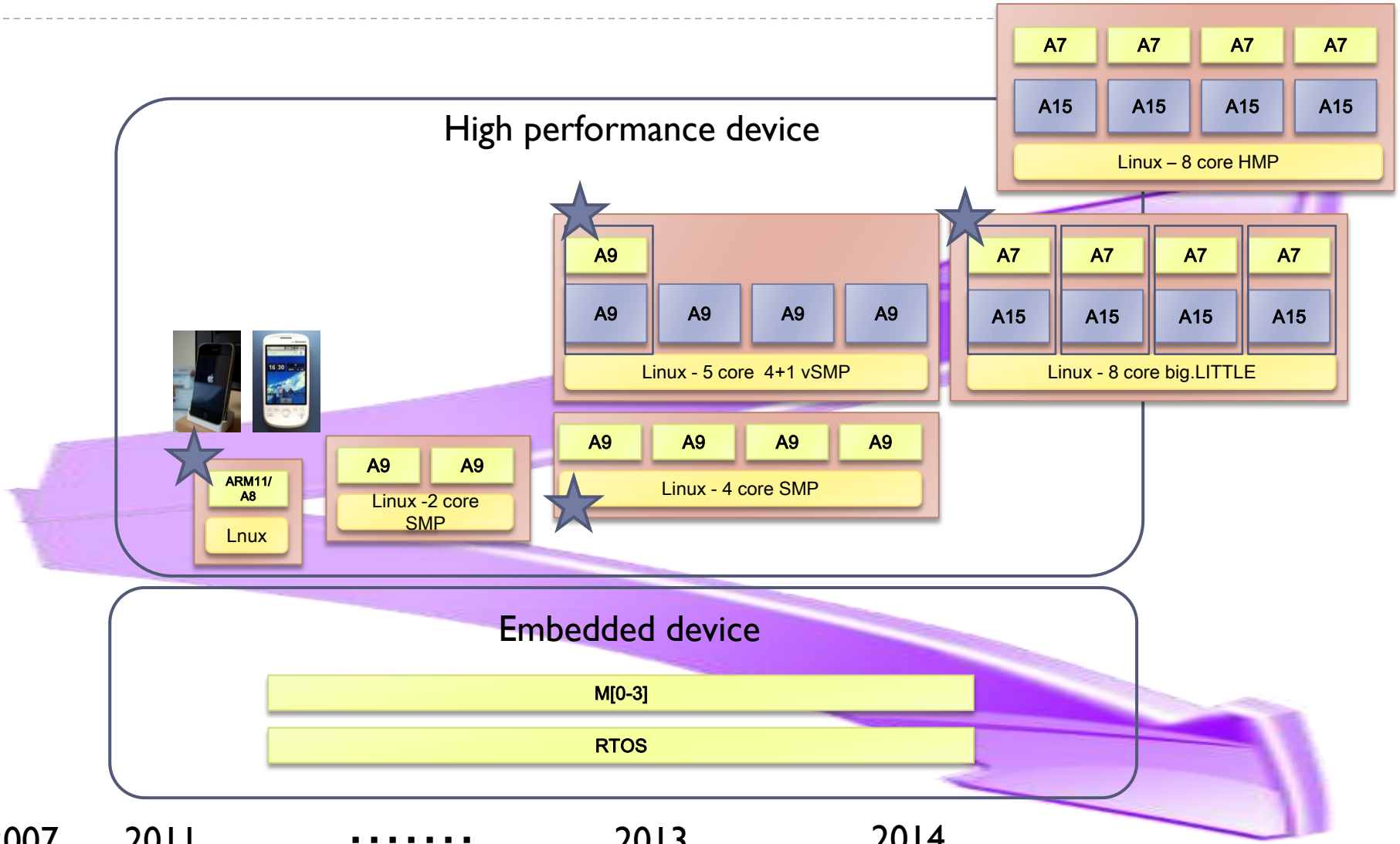
民生組込み機器における低消費電力化の手法

2014/10/22

ESS2014 企画セッション

山本英雄(早稲田大)

組み込みプラットフォームの2極化

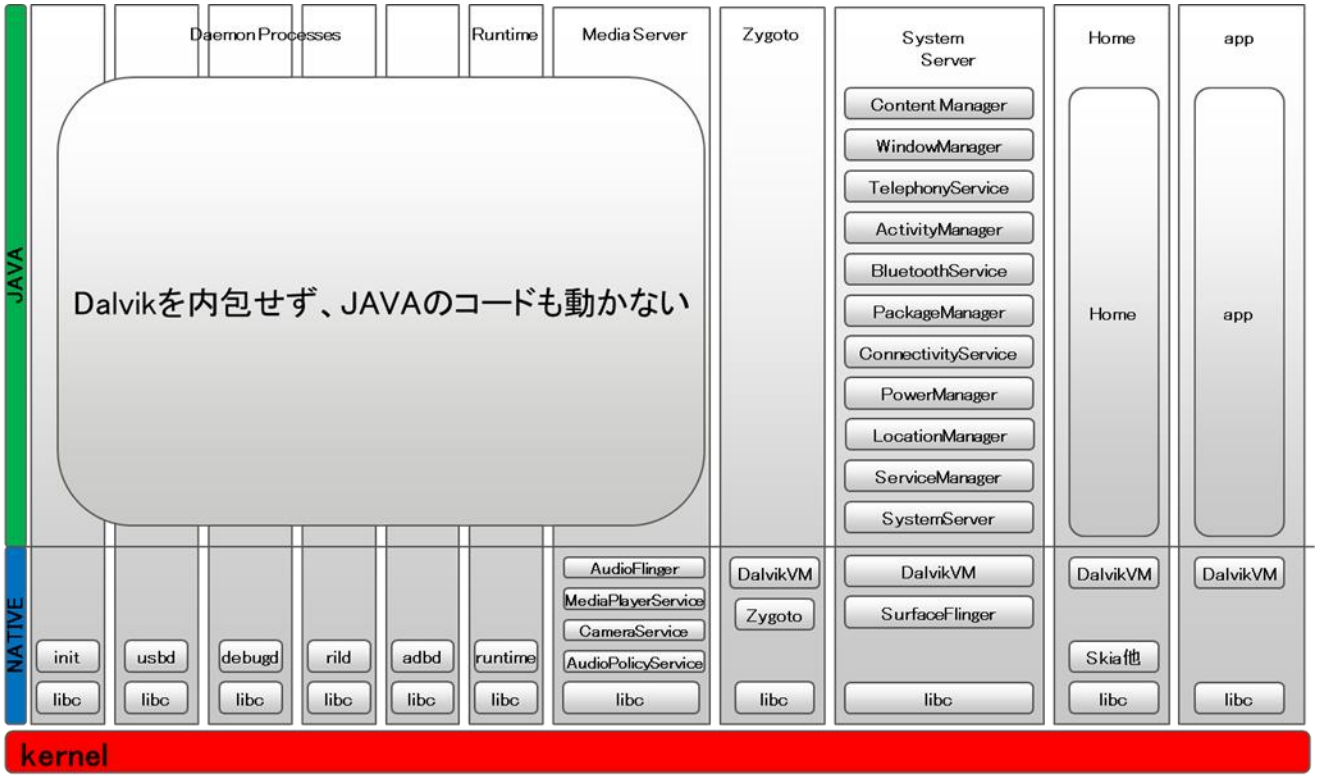


汎用OSを用いた組み込み機器

- フルセットのネットワーク機能
- Java,c,c++とそれらの標準ランタイムのサポート
- コーデック
- 3D描画
- 過去のソフトウェア資産

AndroidとLinuxの関係

- Linuxカーネルへの追加は僅少 - 共用メモリ、binder
- 一般的なアプリはDalvik (java VM)上で動作
- 電力管理他、空間の保護、スケジューラーもLinuxの機能に依存

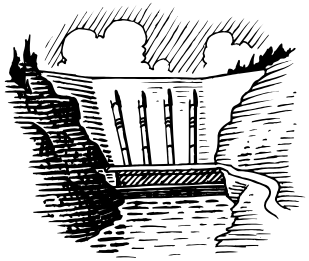


前提とするHW/SWの基礎

HW:オームの法則からSW:プロセスの状態遷移

電力って何

- P:電力、V:電圧、I:電流、R:直流抵抗、C x F:交流抵抗(逆数)
- V=IR　I=V/R
  - 電圧が一定の時、抵抗小なら電流大
  - 抵抗が一定の時、電圧大なら電流大
  - 電力P=VI なので、I=V/RでIを置き換えて$P=V^2/R$



- 直流で考えると
  - Rが一定の時、電圧を2倍で電力は4倍
  - Rが一定の時、電圧を1/2倍で電力は1/4倍
- 交流で考えると
  - C キャパシタンス、F 周波数のときC x Fの逆数が交流抵抗
  - Cが一定の時、周波数高なら抵抗小で電流大
  - $P=CFV^2$

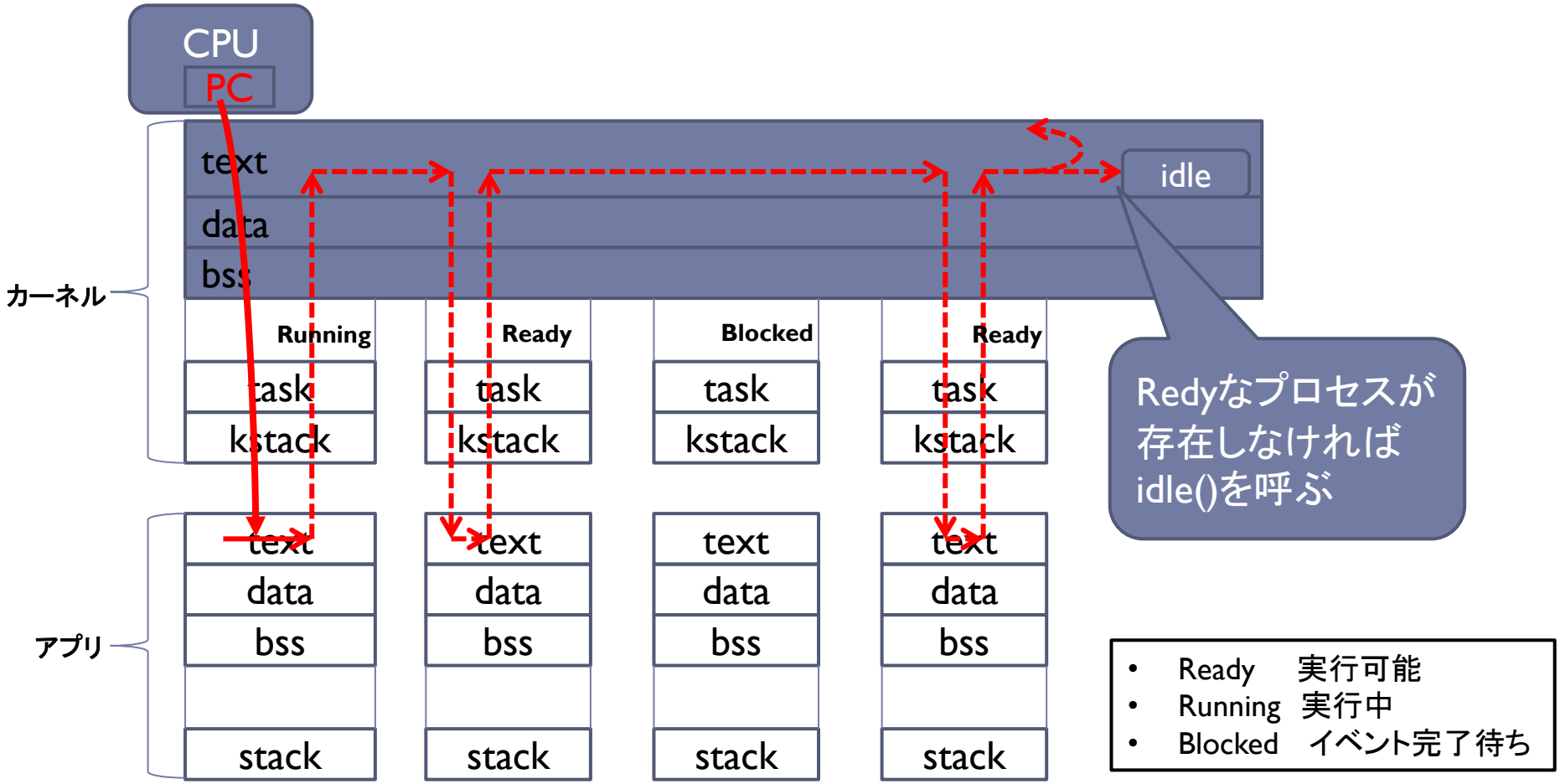
電力削減のためのハードウェア機構

CPUコア電力のモデル式
P = 交流成分＋直流成分 ＝ F x C x V^2 ＋ V^2/R
F:動作周波数を上げるには、電圧も上げる必要がある
F:動作周波数が0なら交流成分の電力は0

- DVFS dynamic voltage/frequency scaling
  - CPUのF:動作周波数とV:電圧を動的に変更する
- Clock Gateing - クロック遮断
  - F:動作周波数を0にする
- Power Gateing - 電力遮断
  - V:電圧を0にする

idleによる休眠状態

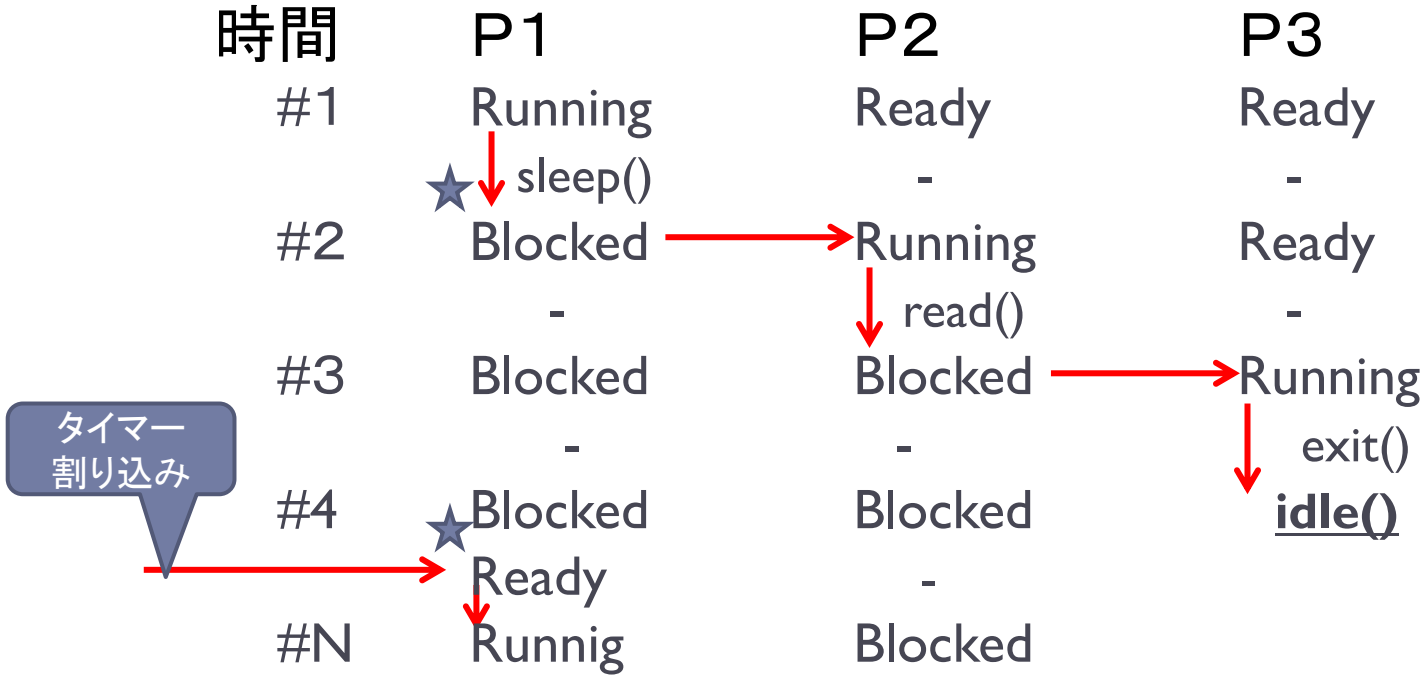
- 実行可能状態のプロセスを探して、コンテキストを復元
- 実行可能状態のプロセスが存在しない場合はidle()を呼ぶ



割り込み

- イベントに応じて、PCに任意のアドレスを書き込む
- HW割り込み
  - 外部の装置が生成する代表的なイベント
    - タイマ割り込み　　設定した時間の経過時に生成
    - IO割り込み　　DISK/LAN/keyboardなどIO装置が生成
- SW割り込み
  - プログラム自身が生成するイベント
    - 「システム呼び出し命令」*の実行
    - 0除算、不正アドレス参照などの例外

割り込みとidleの関係



- idleの割合がCPUの負荷を示す
- idleを呼ぶとOSがCPUを休眠状態（低電力）に遷移させる
- 割り込み発生時にHWが休眠状態から復帰させる

割り込みがidelからの復帰契機

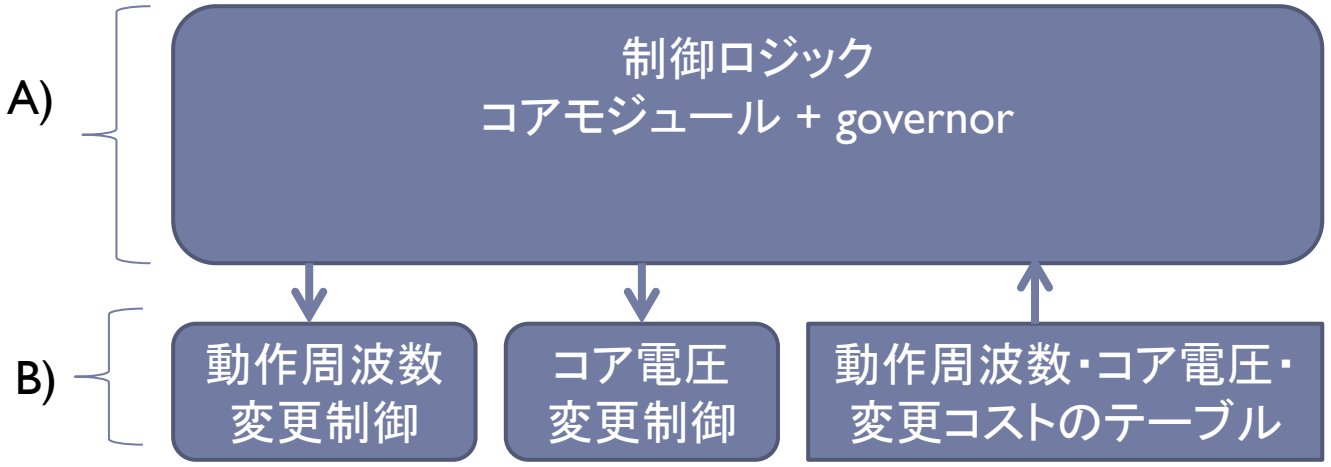
- idleを読んだ後、システムの状態は全て割り込みを契機に更新される

電力削減のためのOSの役割

A) 制御ロジック
- 例）CPU負荷など、システムの状態を判断して動作周波数を変更する

B) ハードウェアIF
- 例）SoCの仕様に従い、動作周波数と電圧のテーブルを用意し、周波数の対応した電圧を指定する

DVFSの例



代表的な制御ロジック

A) 負荷に応じて、動作周波数を調整する→DVFS
- 負荷に応じてClockとVoltageを変更
- 負荷はidle状態を評価

B) idle時の休眠状態を調整する→Clock Gating
- 深く寝ると復帰に時間がかかる

C) 負荷に応じて、動作コア数を調整する→Power Gating
- 過負荷を検出してコアを自動的に追加・削除

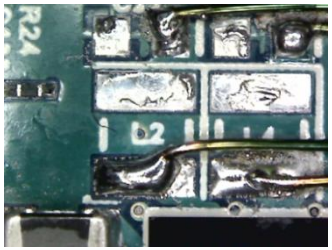
OSが制御する範囲で

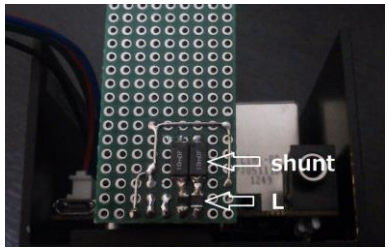
D) 電力と応答性のバランスを調整する→Tick Less
- OSが扱う時間はCPUの動作周波数とは別の時計で刻む – TICK
- TICKが進む都度、タイマ割り込みが発生して実行可能状態のプロセスをプライオリティ順に実行する

電力評価の手法

電力の測定方法

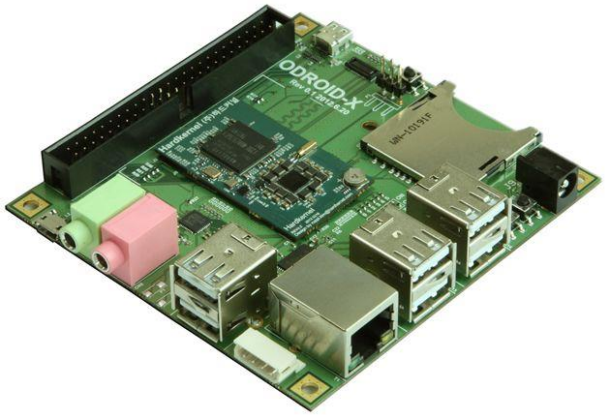
- 正確な値がわかっている抵抗を電源部に挿入し、抵抗での降下電圧を測定して電流を得る
  - I=V/R
  - Rは定数なので、Iは降下電圧:VをRで除算すれば求まる
- 電流:Iを降下後の電圧Vで乗算すれば電力が求まる





- 電力回りの回路はPMIC＋L＋Cなので、回路図無しでも解析は容易で複数の改造事例あり

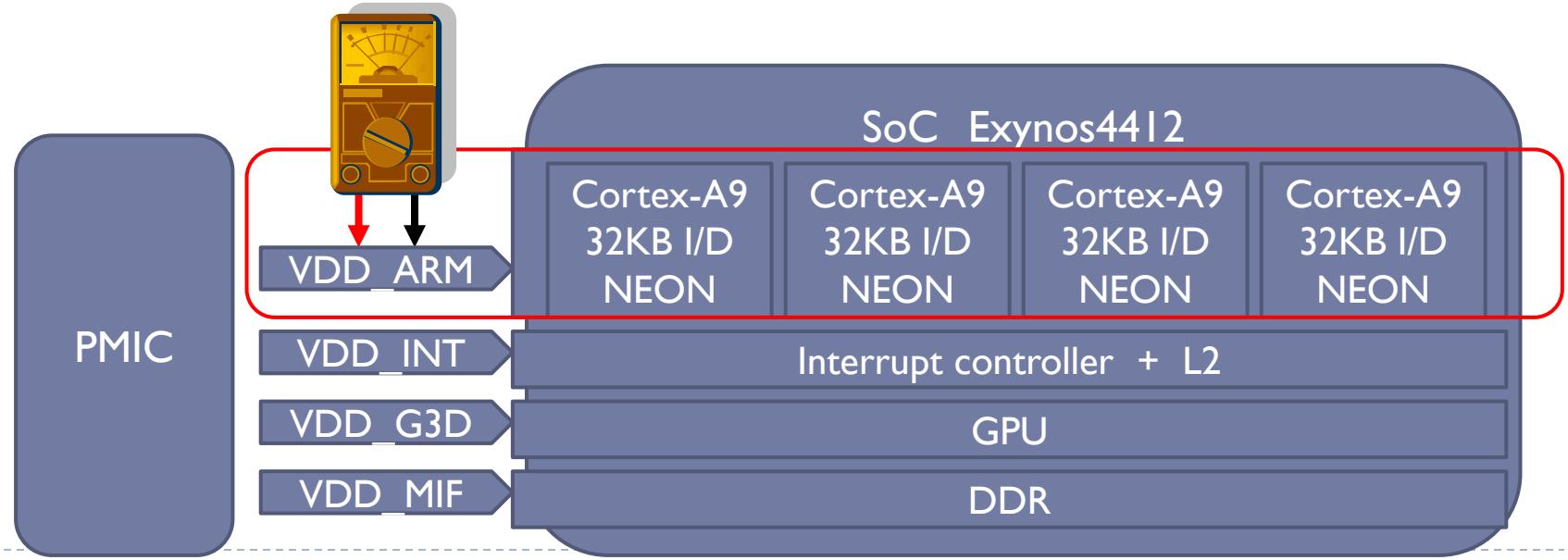
Exynos4412の例



- ODROID-X2　hardkernel社
- Samsung Exynos4412 Prime
  - ARM Cortex-A9 Quad core
  - 最大 1.7GHz
- 回路図とAndroid/linuxのソースコードは公開
  - シリアル番号をメールで送ると回路図が送られてくる

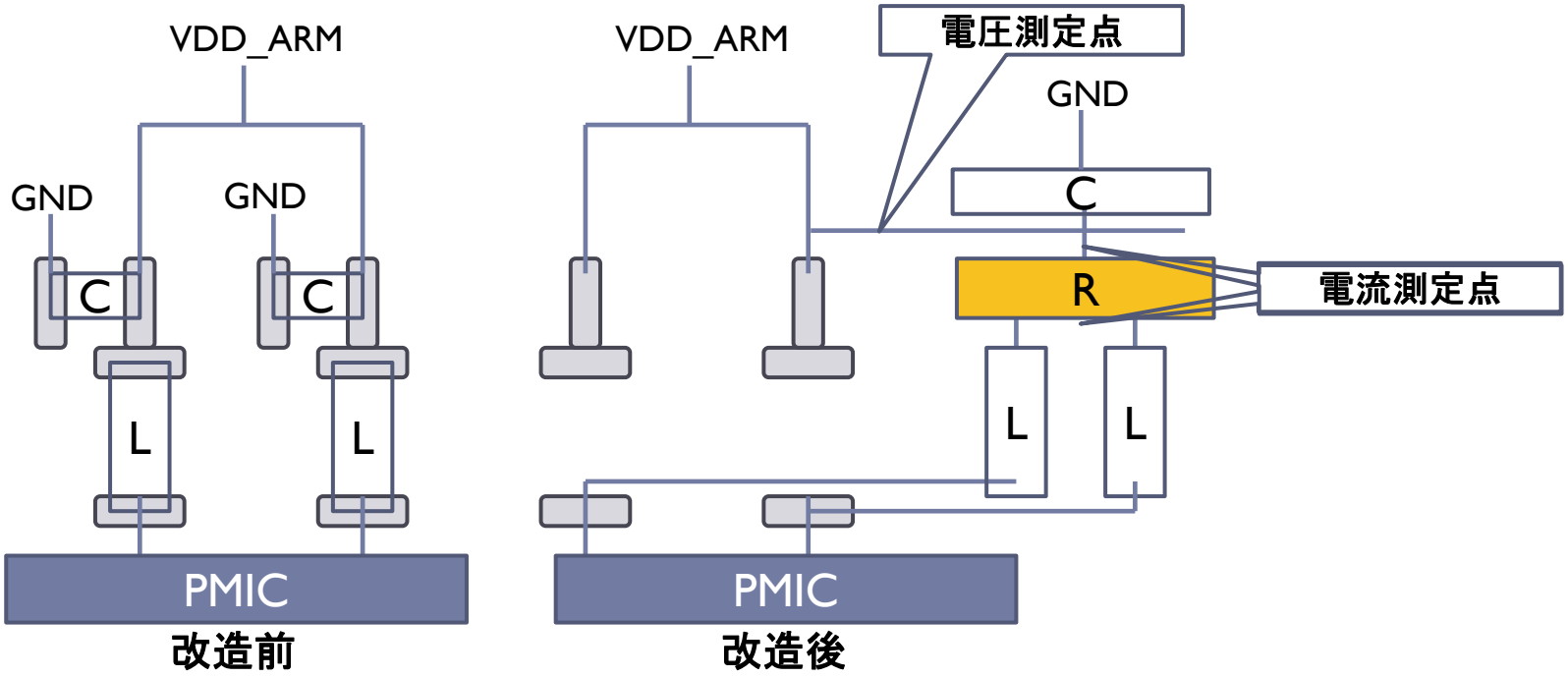
Power Railの例　Exynos4412の場合

- PMICはExynos4412 に4種類の電源を供給
  - VDD_ARM　　CORE
  - VDD_INT　　Interrupt controller and L2
  - VDD_G3D　　GPU
  - VDD_MIF　　DDR Memory
- VDD_ARM (CORE) を測定対象にした



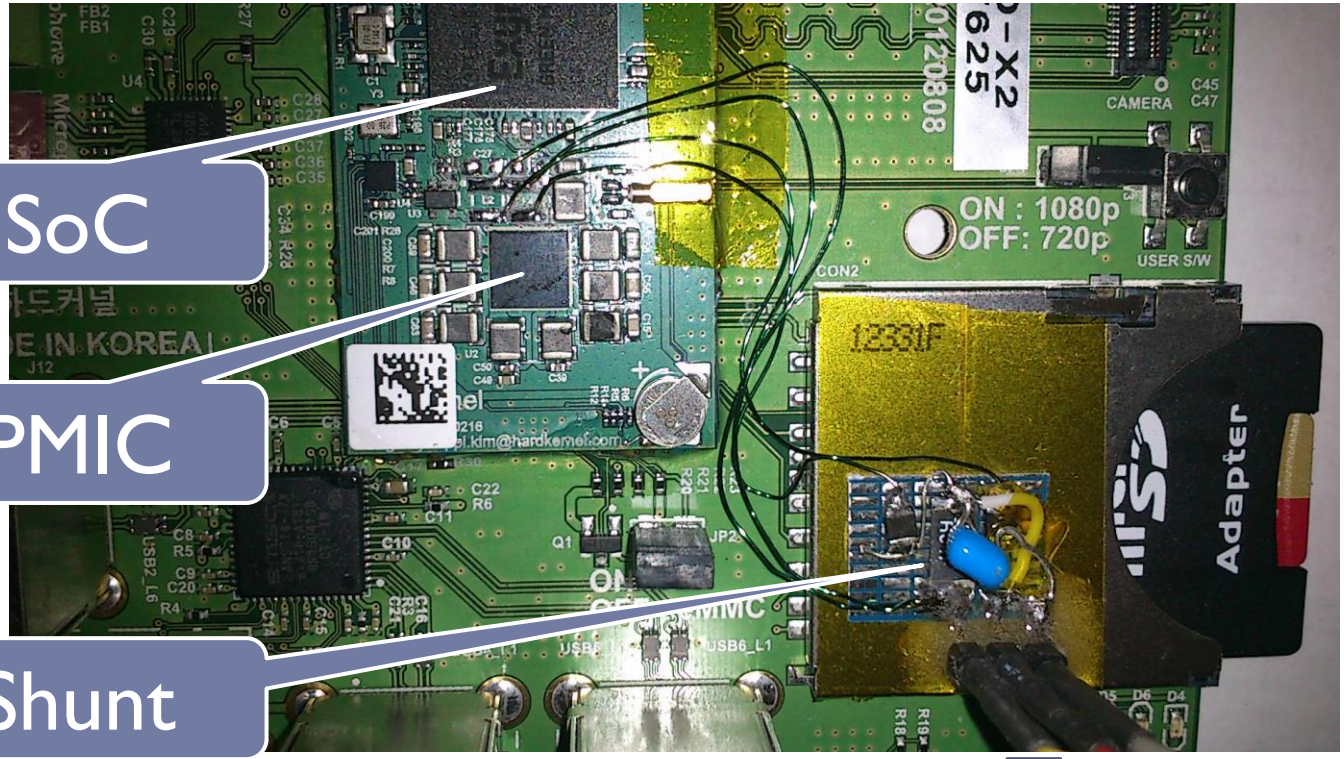
回路パターンの変更例　Exynos4412の場合

- PMIC(電力制御IC)からCPUにつながる回路を加工

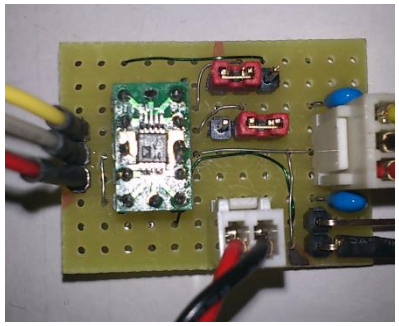


- Rでの電位差は微小なため、専用の測定器または計装アンプが必要

Exynos4412の改造例



オシロ・ADCなど



計装アンプ

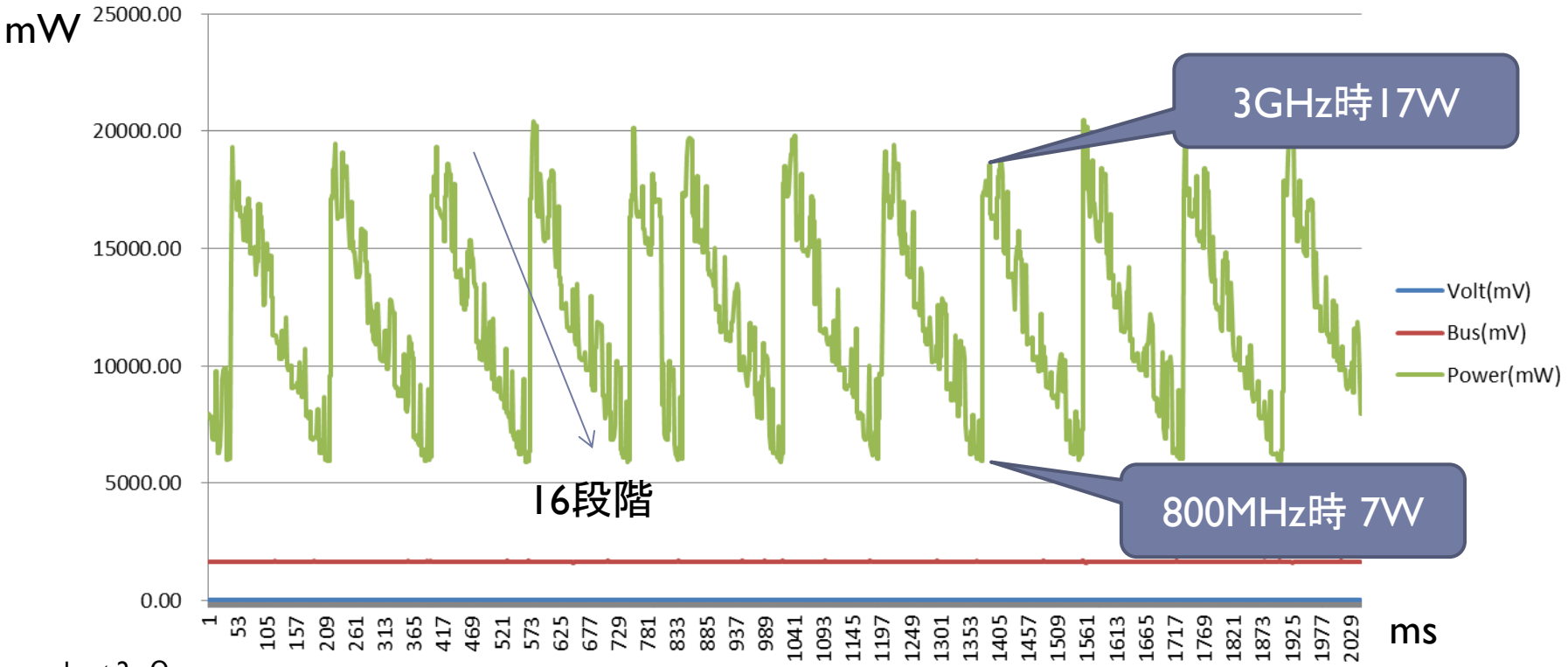
Shunt電圧

ハードウェアIF層での電力削減の効果

実測を結果から

DVFSの効果 (Intel Haswell)

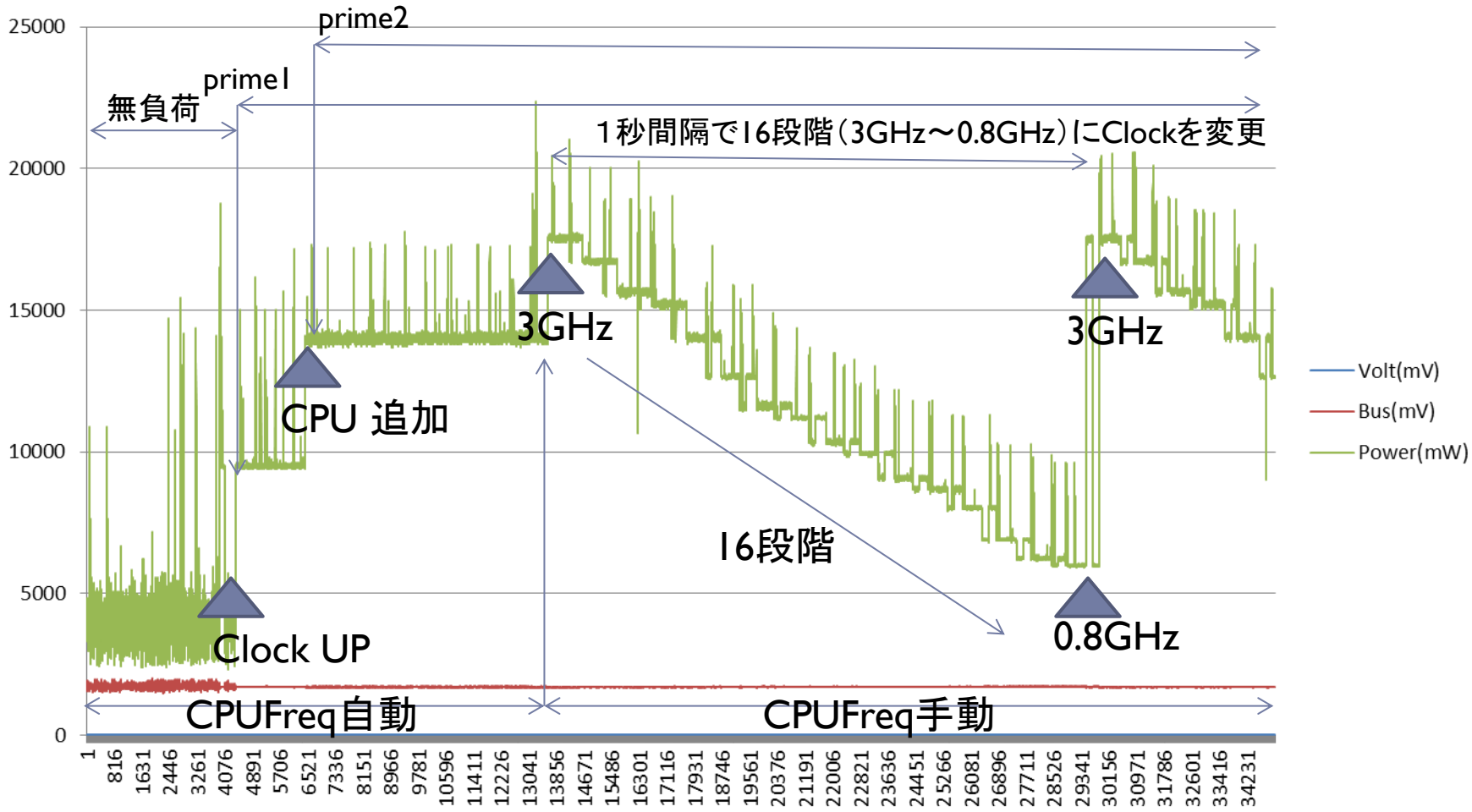
- 負荷を掛けた状態でクロックを変更
- 7W　800MHz時 ～ 17w 3GHz時



shunt:2mΩ
INA226を利用

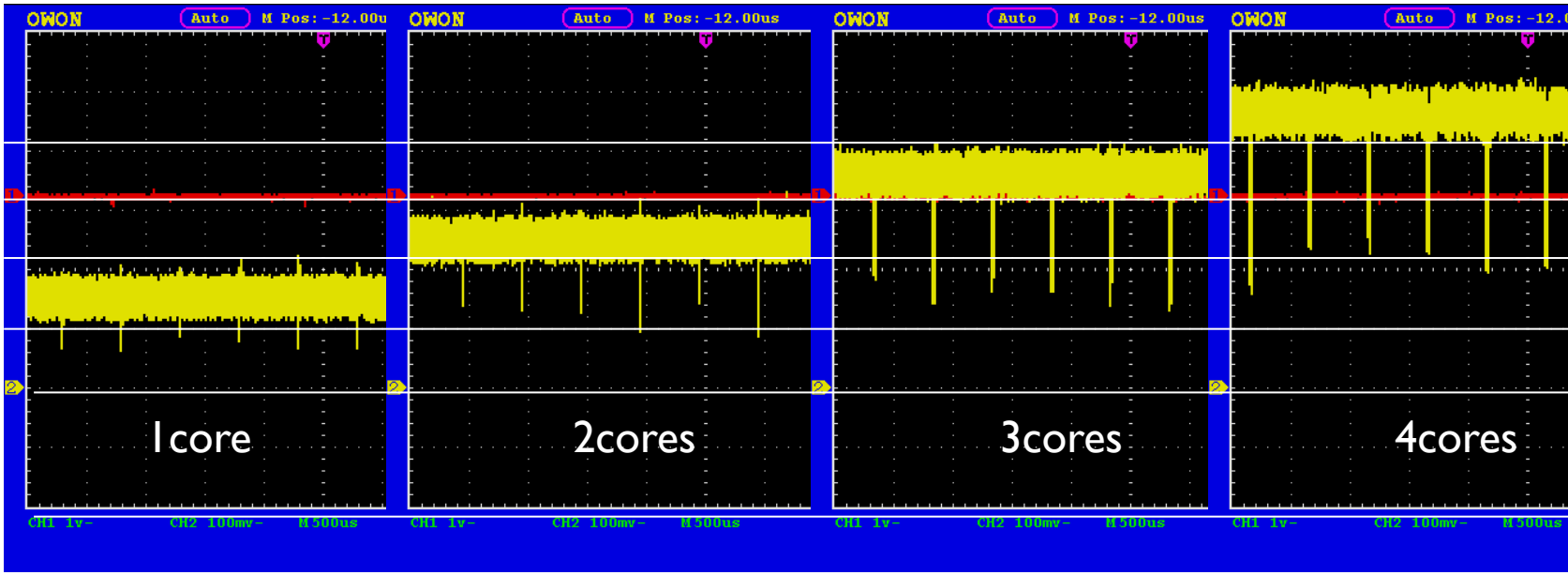
Intel Haswell 2 coreで測定

1Core-2Core-DVFS測定結果



Power Gatingの効果（ARM）

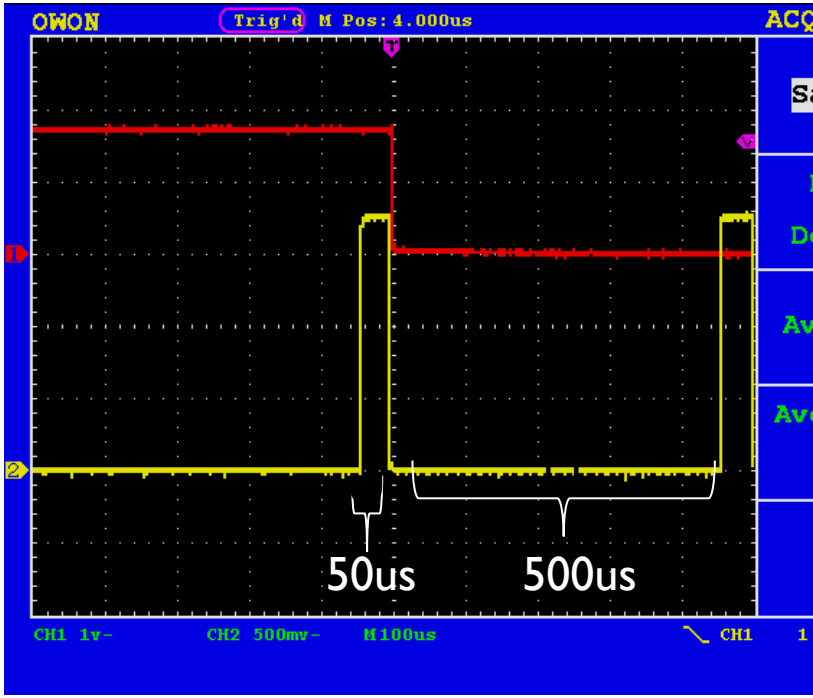
- 4多重の負荷を掛けた状態でコアを順次Pon



Exynos 4412 で測定

Tick値の影響　（ARM）

- 一定時間間隔でタイマ割り込みを発生させる頻度
  - 100HZから200HZ程度
  - 割り込み処理で、時間関連のソフトウェアイベントを処理
  - 割り込み処理のコストは一定
- 応答性能を向上するために
  Tickを200から2000に変更すると
  - 割り込みオーバヘッドが10%に
  - 従来は1%



Exynos 4412 1coreで測定

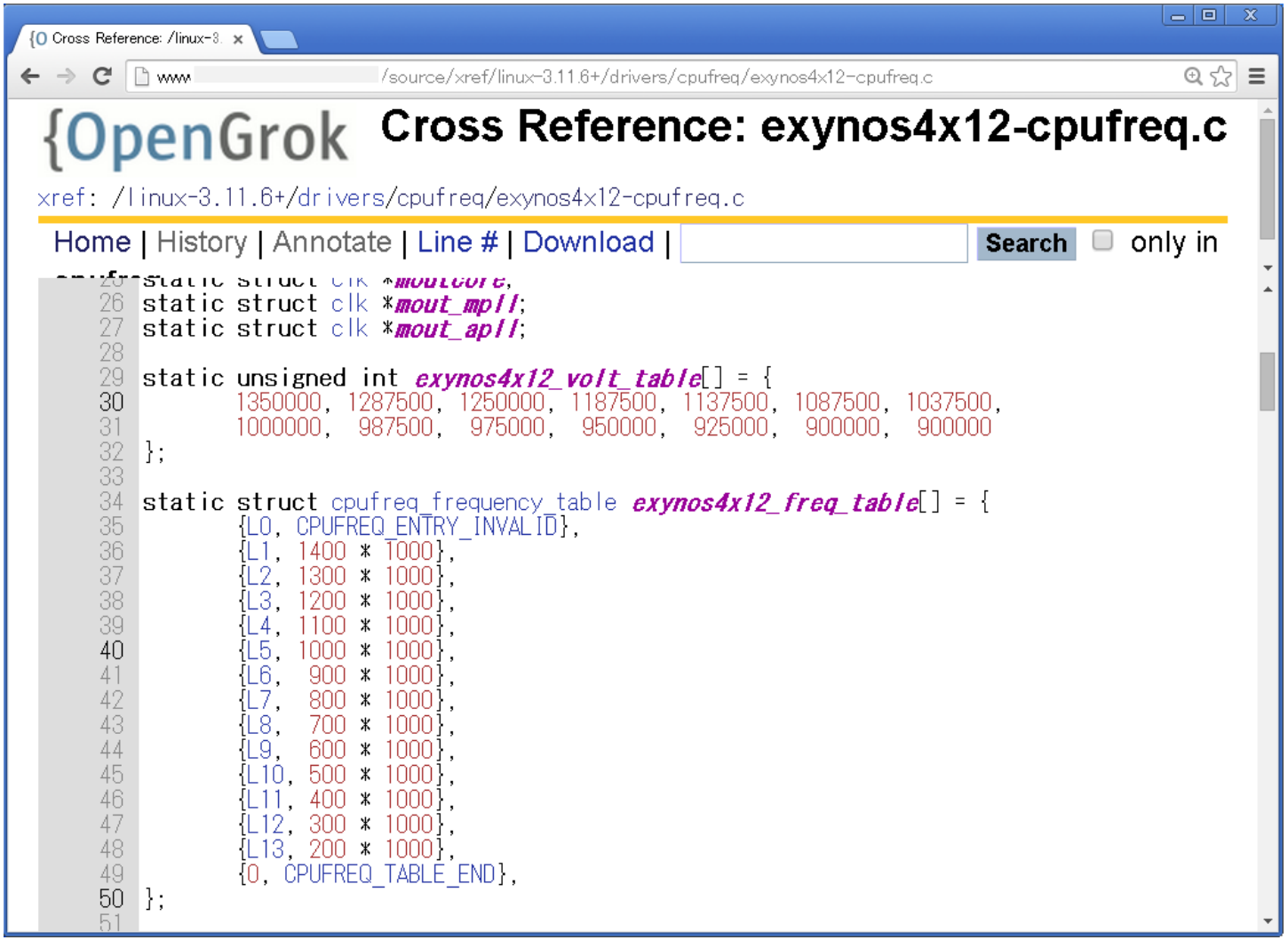
Linuxの電力管理

制御ロジック - governor

DVFSの制御 – CPUFreq

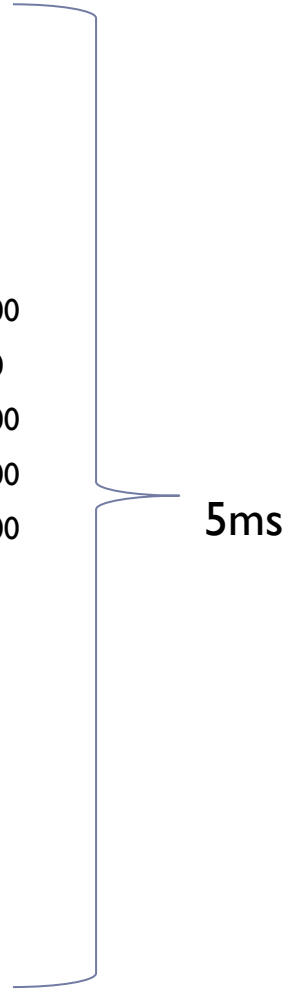
- 制御ロジックの実装 - drivers/cpufreq/
- Governorの種類
  - Powersave(cpufreq_powersave.c)　いつも最低
  - Performance(cpufreq_performance.c)　いつも最高
  - Conservative(cpufreq_conservative.c)　控え目（Step毎）に変更
  - Ondemand(cpufreq_ondemand.c)　一度に高クロックにあげる
  - Userspace(cpufreq_userspace.c)　sysfs経由でユーザーが操作
- Ondemandを標準とする場合が多い
  - 対話性能を重視
  - idleが80%（調整可）を超えると、Stepを無視して高クロックに変更

電圧・周波数テーブルの例



CPUFreq Clock up の遷移時間

```
<7>[  942.369161] notification 0 of frequency transition to 1200000 kHz
<7>[  942.369500] notification 0 of frequency transition to 1200000 kHz
<7>[  942.369685] notification 0 of frequency transition to 1200000 kHz
<7>[  942.370010] notification 0 of frequency transition to 1200000 kHz
<7>[  942.370193] cpufreq-tegra: transition: 340000 --> 1200000
<7>[  942.370555] regulator regulator.2: set_voltage: name=max77663_sd1, min_uV=1100000, max_uV=1350000
<7>[  942.371086] regulator regulator.1: set_voltage: name=max77663_sd0, min_uV=900000, max_uV=1250000
<7>[  942.371467] regulator regulator.2: set_voltage: name=max77663_sd1, min_uV=1200000, max_uV=1350000
<7>[  942.371985] regulator regulator.1: set_voltage: name=max77663_sd0, min_uV=1000000, max_uV=1250000
<7>[  942.372505] regulator regulator.1: set_voltage: name=max77663_sd0, min_uV=1025000, max_uV=1250000
<7>[  942.373135] notification 1 of frequency transition to 1200000 kHz
<7>[  942.373209] FREQ: 1200000 - CPU: 0
<7>[  942.373345] notification 1 of frequency transition to 1200000 kHz
<7>[  942.373483] FREQ: 1200000 - CPU: 1
<7>[  942.373561] notification 1 of frequency transition to 1200000 kHz
<7>[  942.373756] FREQ: 1200000 - CPU: 2
<7>[  942.373832] notification 1 of frequency transition to 1200000 kHz
<7>[  942.374027] FREQ: 1200000 - CPU: 3
```



Nexus7(2012)

CPUFreq Clock down 遷移時間

```
<7>[ 1035.045405] notification 0 of frequency transition to 1000000 kHz
<7>[ 1035.045529] notification 0 of frequency transition to 1000000 kHz
<7>[ 1035.045591] notification 0 of frequency transition to 1000000 kHz
<7>[ 1035.045702] notification 0 of frequency transition to 1000000 kHz
<7>[ 1035.045763] cpufreq-tegra: transition: 1200000 --> 1000000
<7>[ 1035.046042] regulator regulator.1: set_voltage: name=max77663_sd0, min_uV=975000, max_uV=1250000
<7>[ 1035.046315] notification 1 of frequency transition to 1000000 kHz
<7>[ 1035.046387] FREQ: 1000000 - CPU: 0
<7>[ 1035.046462] notification 1 of frequency transition to 1000000 kHz
<7>[ 1035.046593] FREQ: 1000000 - CPU: 1
<7>[ 1035.046669] notification 1 of frequency transition to 1000000 kHz
<7>[ 1035.046857] FREQ: 1000000 - CPU: 2
<7>[ 1035.046929] notification 1 of frequency transition to 1000000 kHz
<7>[ 1035.047116] FREQ: 1000000 - CPU: 3
<7>[ 1035.047352] regulator regulator.2: set_voltage: name=max77663_sd1, min_uV=1100000, max_uV=1350000
```

2ms

Nexus7(2012)

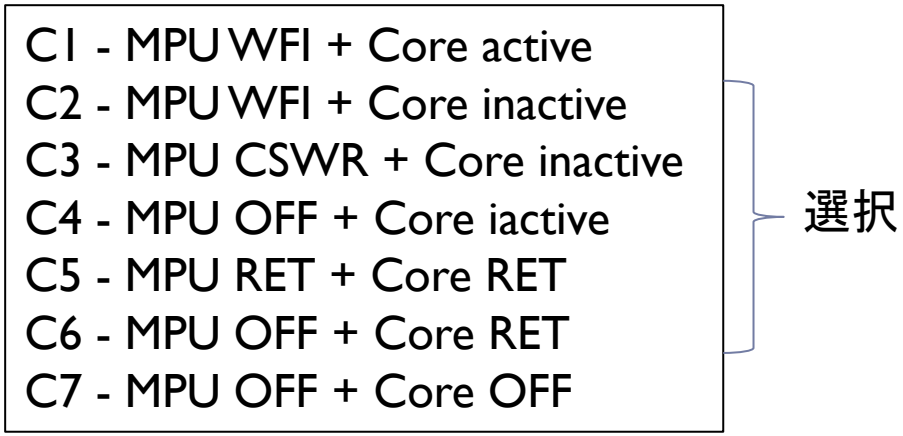
CPUFreqのまとめ

- 制御ロジックとハードウェアIF
- ポリシーベースのGovernor
- 遷移時間は数ms
  - 動作電圧の制御
    - 外付けレギュレータとの通信の時間
    - 電圧安定までの時間
  - 動作クロック(PLL)の制御

- 最新のCPUでは、レギュレータを内蔵するものが登場

Clock Gatingの制御 - CPUIdle

- 制御ロジックの実装 - drivers/cpuidle
  - CPUの休眠状態の深さを管理、休眠開始はidle()



Coreの状態 : omap3の例

- Governorの種類
  - Ladder(cpufreq_powersave.c)　　軽い休眠から重い休眠へ
  - Menu(cpufreq_performance.c)　　統計情報から特定の深さへ

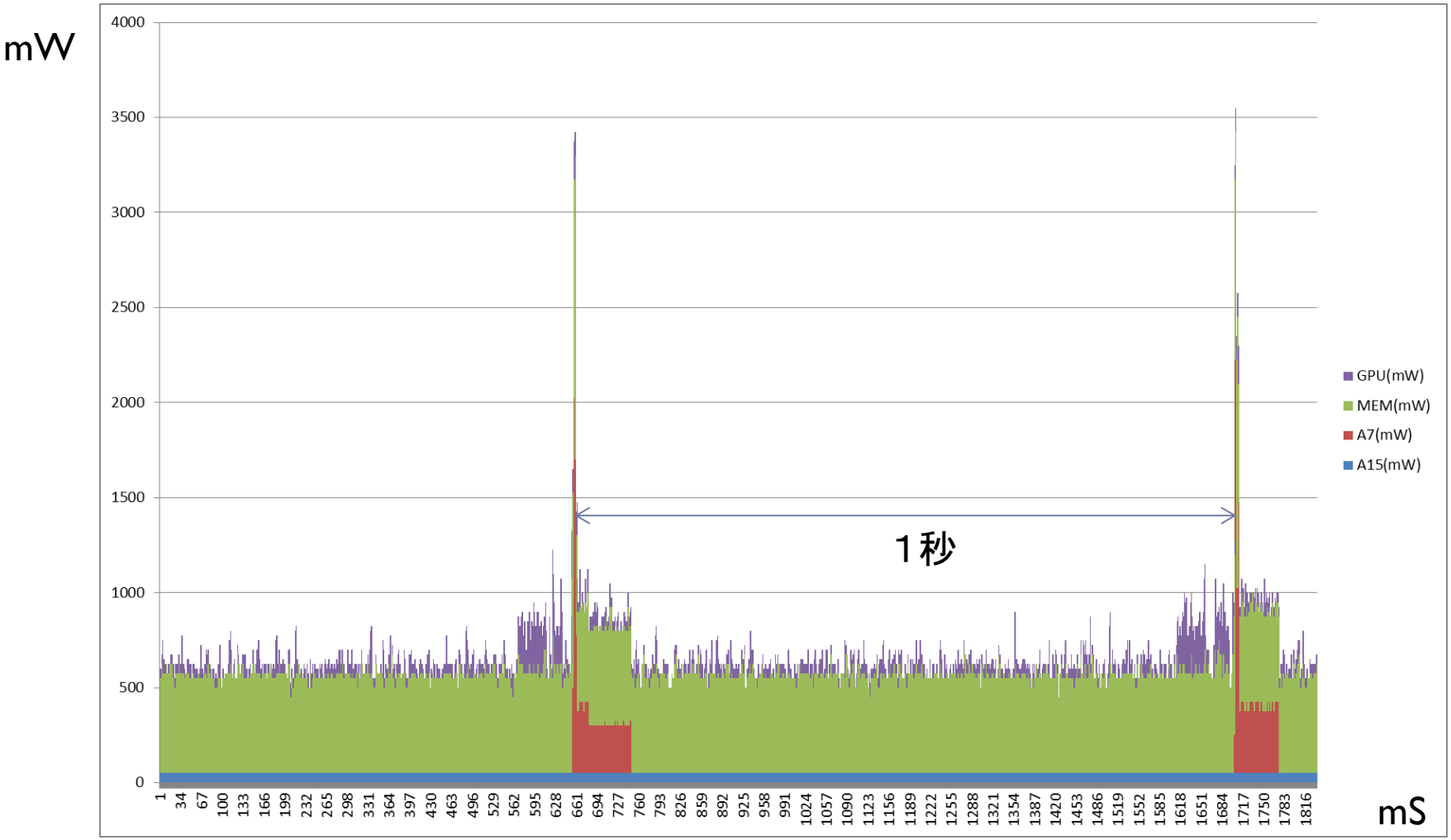
Power Gatingの制御 - hotplug

- マルチコアで利用
- Core0が他のCoreの起動・停止を制御
- 停止対象のCoreからプロセスのコンテキストを移動するコストが高い
- 詳細は後述

電力と応答性 – Tick Less

- 従来のTick
  - OS内部のソフトウェア時計
  - Kernel内の定数、10Hz(100ms)～100Hz(10ms)程度
  - 間隔が長いと休眠時間が長い反面、応答性が劣化
- Tick less（NOHZモード）
  - 制限付きでTickの値を変数に
    - 1プロセス、最大1秒
  - OSが休眠時間をタイマーに設定
    - マルチコアの場合にはコア毎にタイマーを持っている必要がある

Tick Lessの電力波形

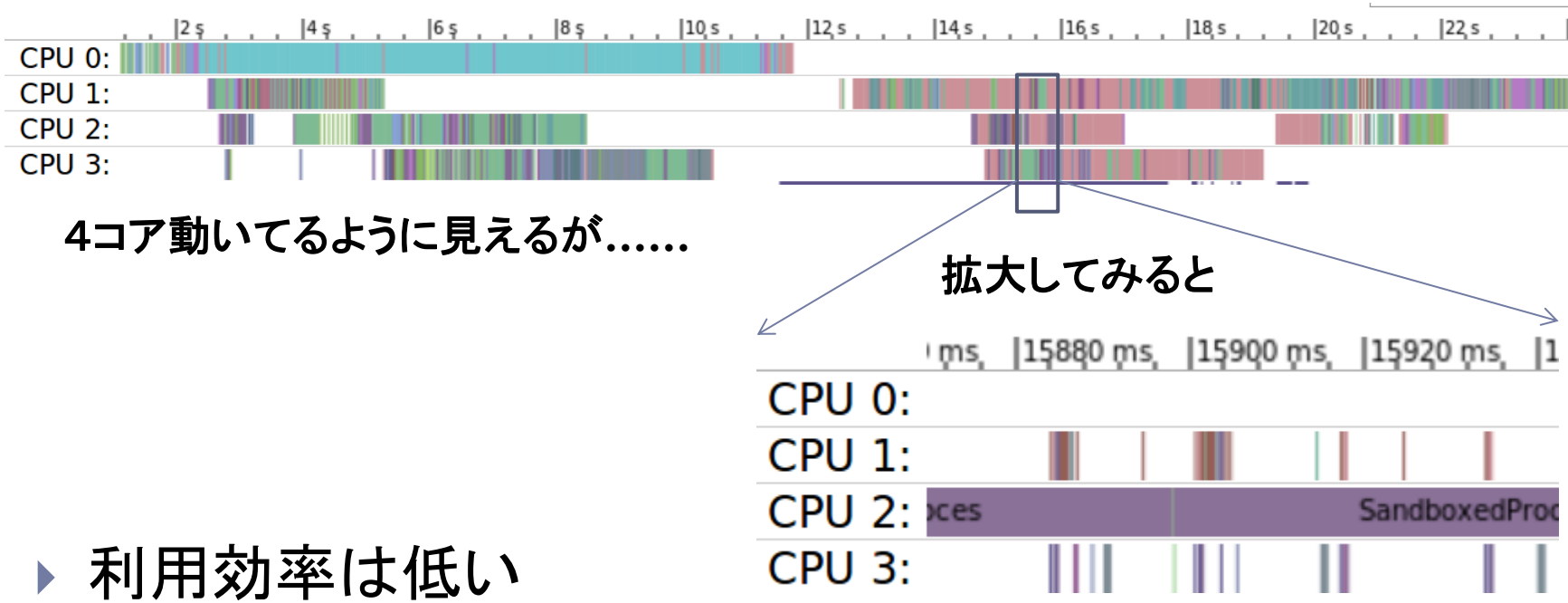


Exynos 5420

マルチコアLinuxの電力管理

マルチコアの利用の現状

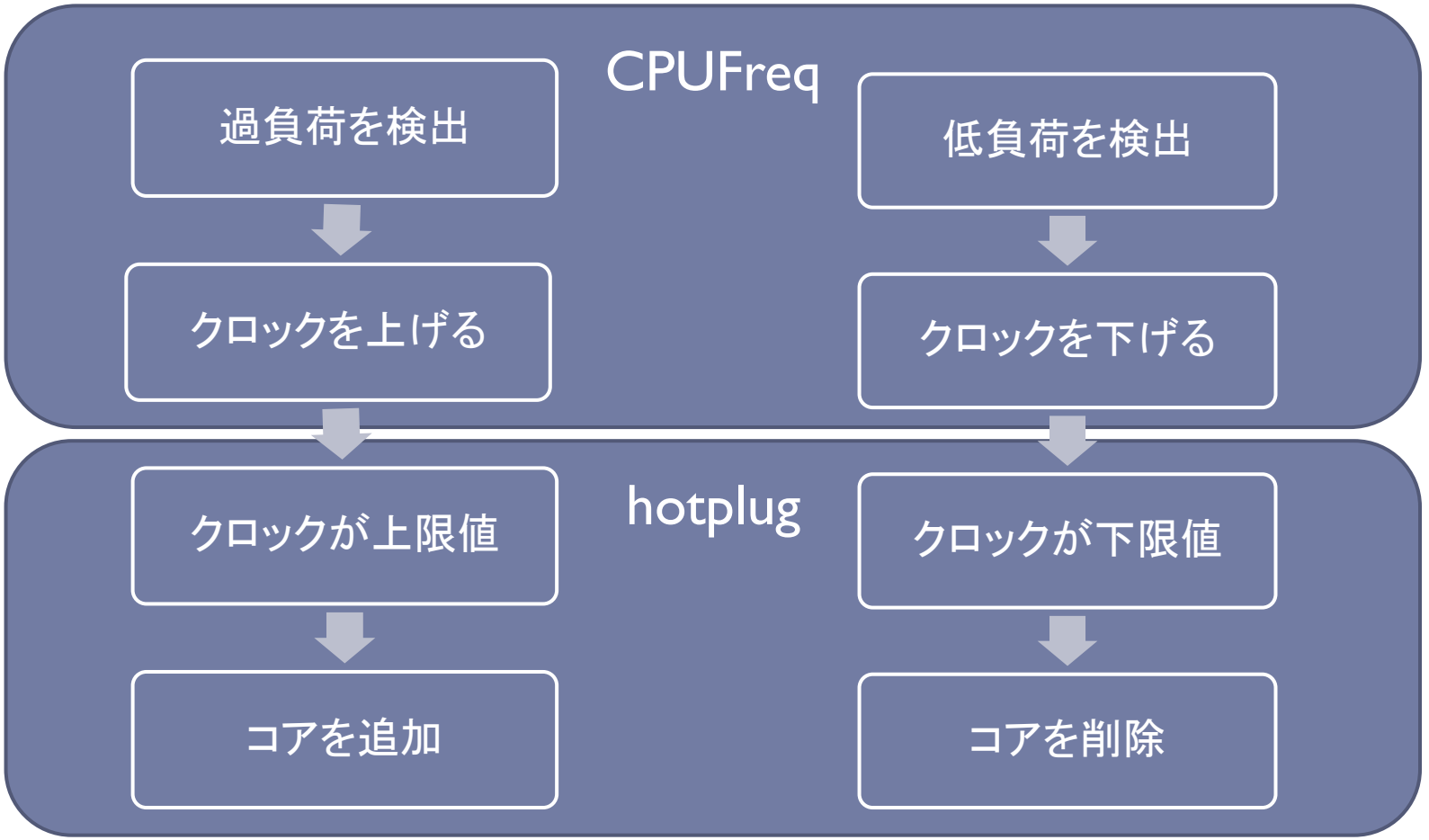
- Nexus7 (2012)
- TwitterクライアントとChrome動作時のsystrace



- 利用効率は低い

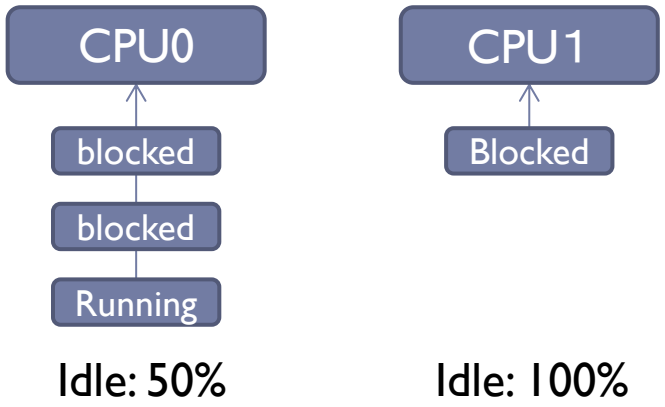
マルチコアの電力管理

- CPUの追加・削除をCPUFreqの拡張として実装　- hotplug



Hotplugの問題点

- プロセスの移動コスト、Pon/Poffコストが大きい



- CPU1をPoffする場合
  - CPU1が管理しているblockedプロセスをCPU1への移動が必要
- Pon時
  - 対象CPUのリセット割り込みから初期化
  - プロセスの移動
- 1～10ms程度を要するため、Pon/Poffは「慎重な判断」が必要

on/off lineのカーネルログ　Tegra3の場合

```
<4>[138006.624075] CPU1: Booted secondary processor
<6>[138006.634275] Switched to NOHz mode on CPU #1 B
<4>[138006.636903] CPU2: Booted secondary processor
<6>[138006.646632] Switched to NOHz mode on CPU #2 B
<4>[138006.650118] CPU3: Booted secondary processor
<6>[138006.657736] Switched to NOHz mode on CPU #3 B
<4>[138020.741637] stop_machine_cpu_stop smp=1
<4>[138020.741649] stop_machine_cpu_stop smp=3
<4>[138020.741659] stop_machine_cpu_stop smp=0
<4>[138020.741670] stop_machine_cpu_stop smp=2
<5>[138020.743037] CPU1: shutdown
<4>[138020.746183] stop_machine_cpu_stop smp=0
<4>[138020.746195] stop_machine_cpu_stop smp=2
<4>[138020.746206] stop_machine_cpu_stop smp=3
<5>[138020.747326] CPU2: shutdown
<4>[138021.227998] stop_machine_cpu_stop smp=0
<4>[138021.228010] stop_machine_cpu_stop smp=3
<5>[138021.228841] CPU3: shutdown
```

12ms　ONline

11ms

1.4ms

1.1ms　OFFline

0.8ms

v SMP: Tegra3の事例

SMPVariable SMPVariable SMPVariable SMP (4 -PLUS PLUS -1 ™)より

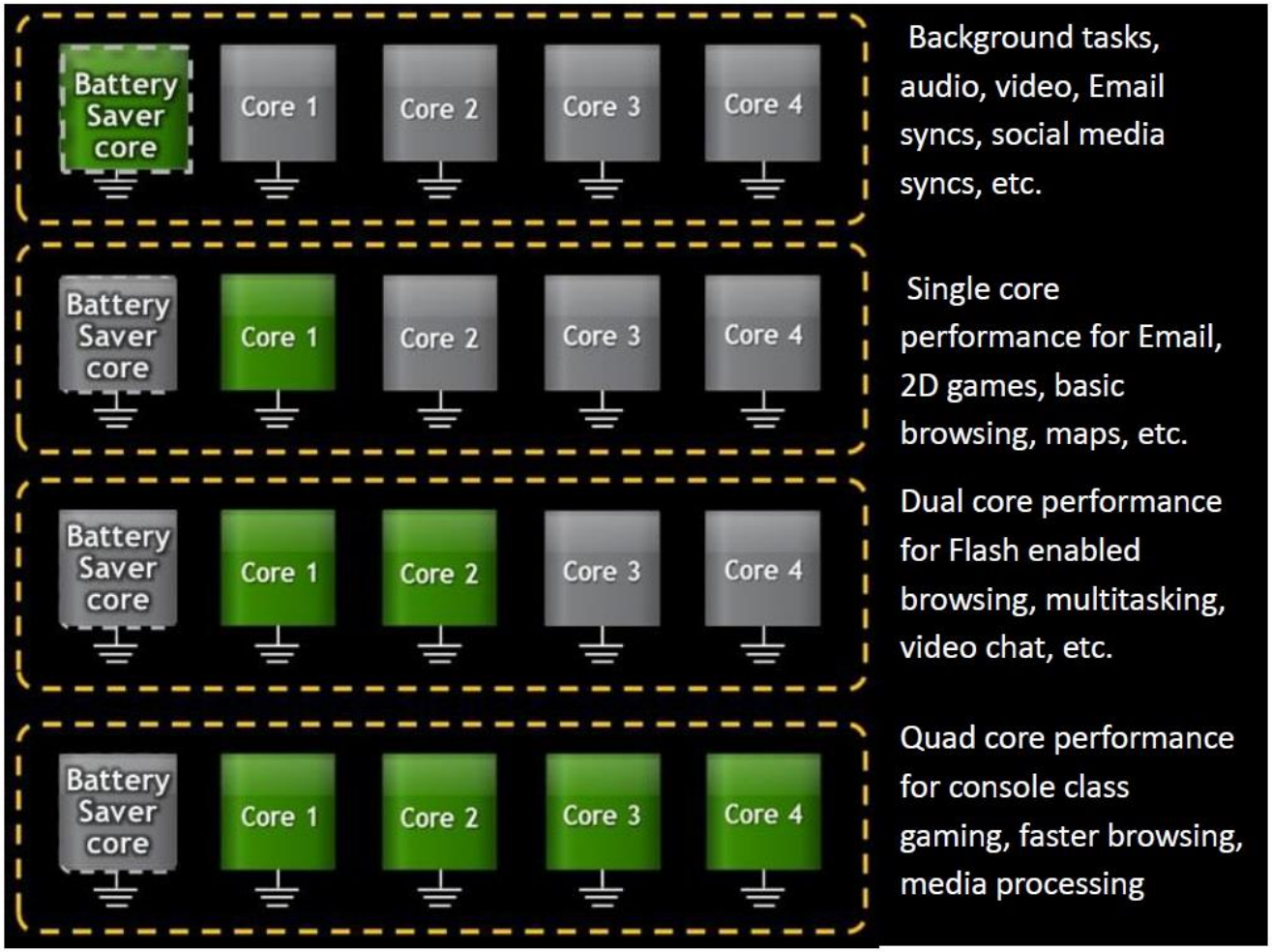
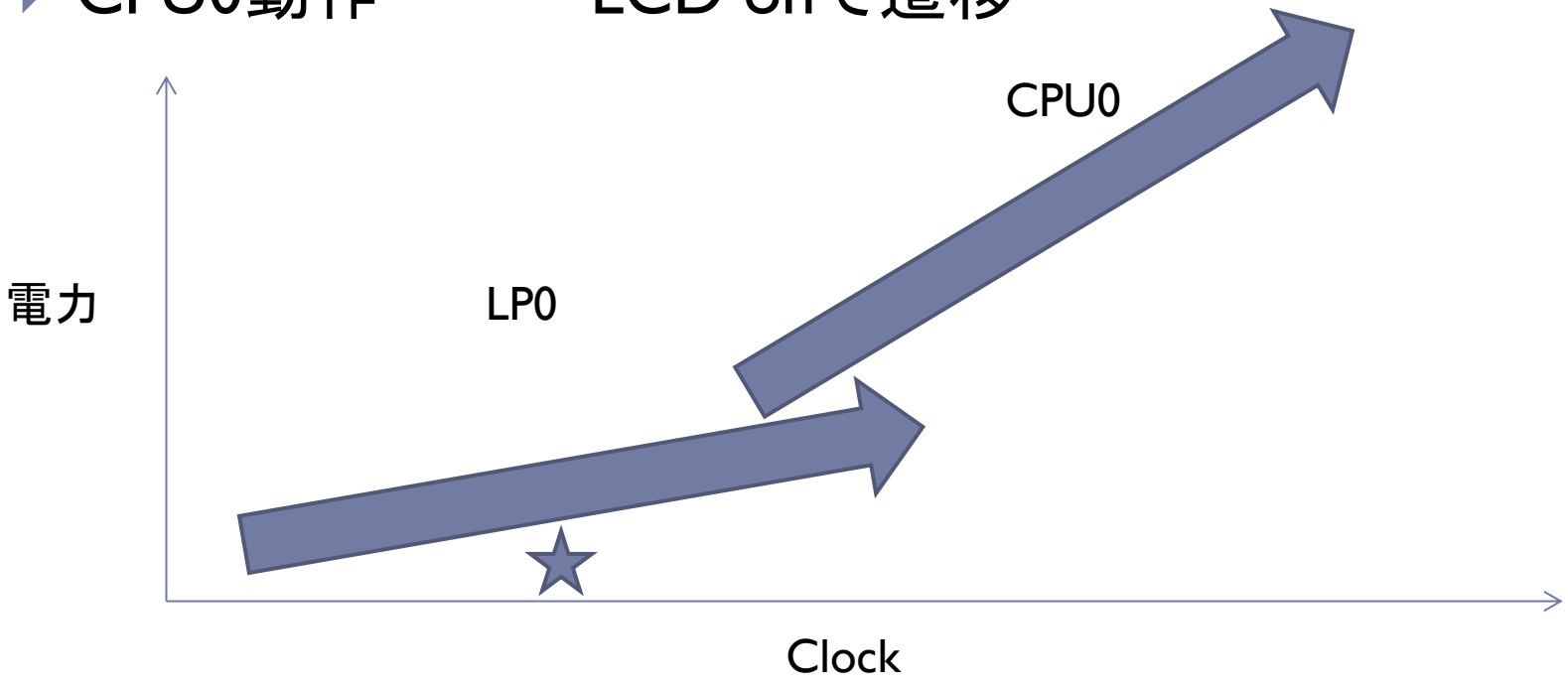


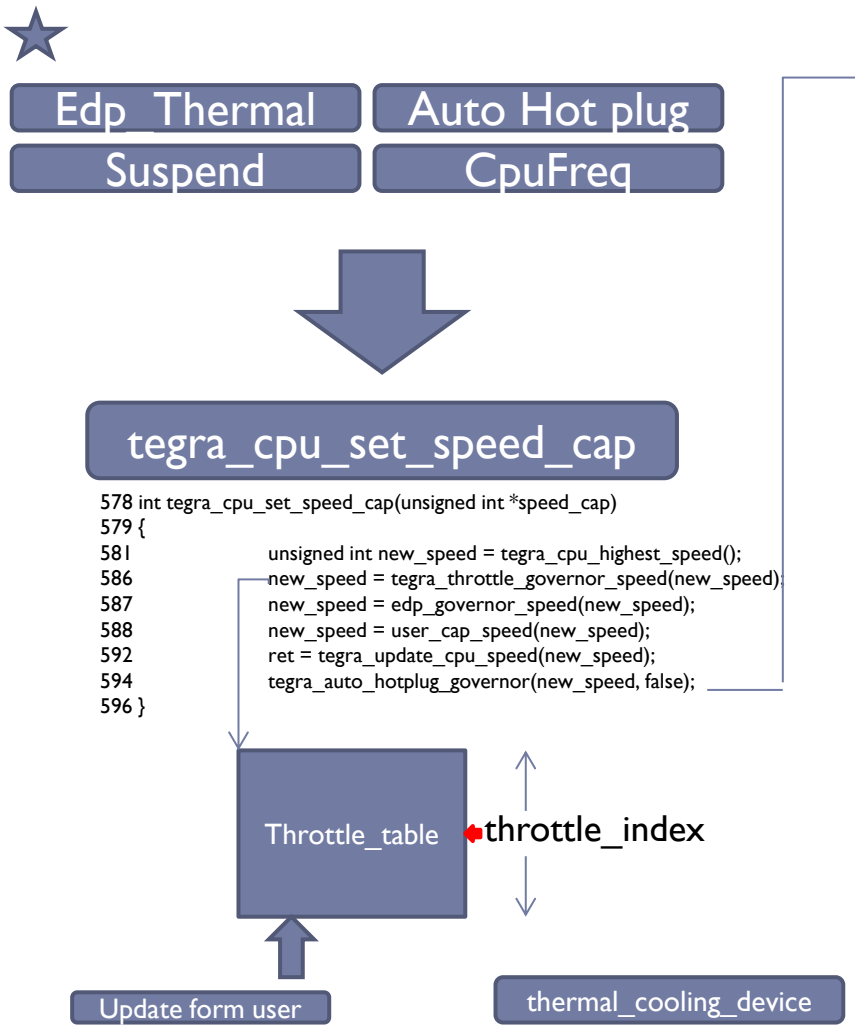
Figure 5 CPU core management based on workload

Battery Save Coreとは何か

- CPU0の定電圧動作時にCPU0に成り代わって透過的に動作するCPU（LP0）
- LP0動作　　LCD offで遷移
- CPU0動作　　LCD onで遷移



Tegra-3のhotplug governor



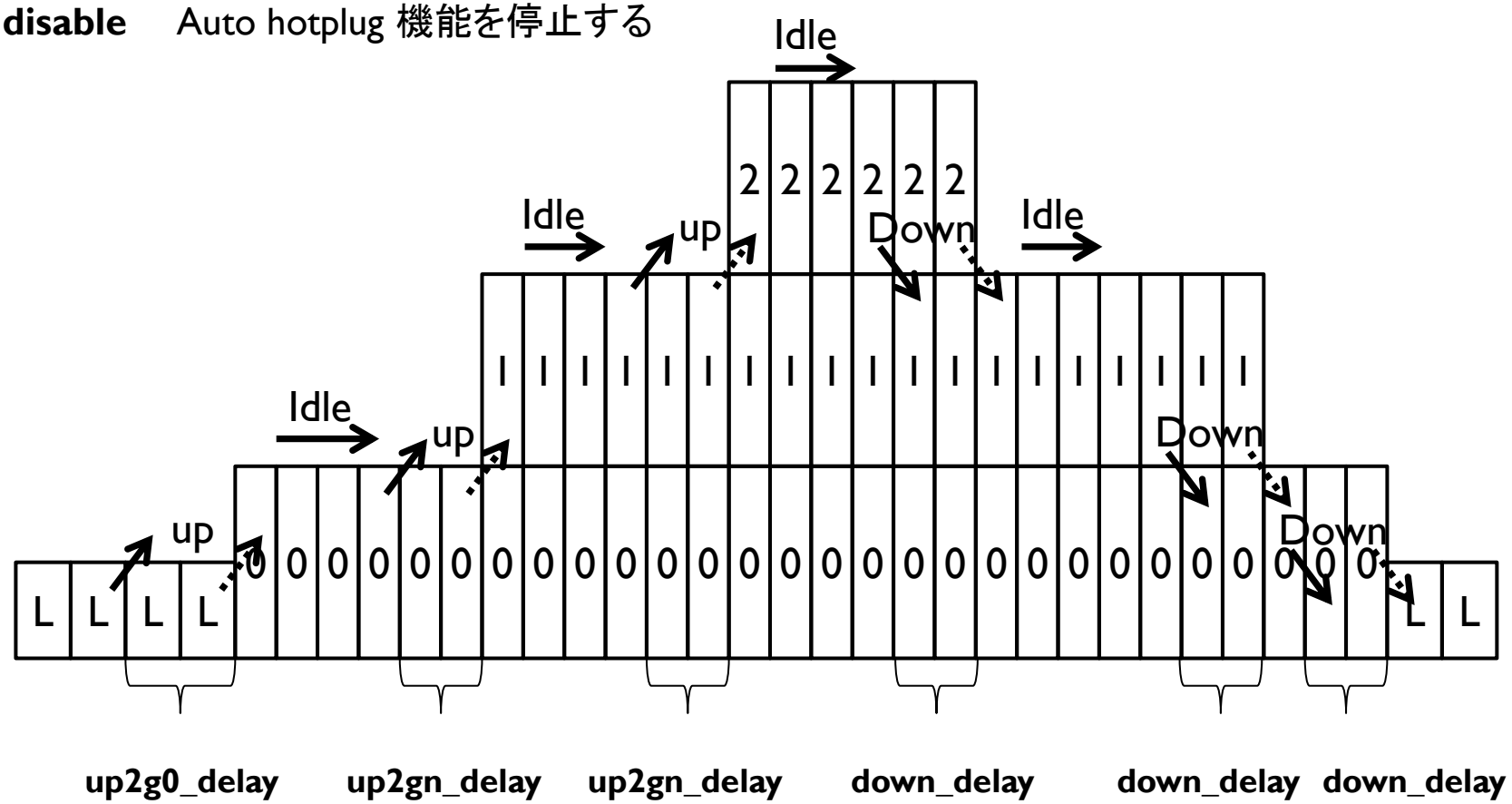
tegra_auto_hotplug_governor

parameters	LP-mode	GP-MODE
up_delay	up2g0_delay	up2dn_delay
down_delay	down_deley	down_delay
top_freq	idle_top_freq	idle_bottom_freq
botttom_freq	0	idle_bottom_freq

Current State	Compare with requested freq	New State	Delay to effecte
IDLE	> top_freq	UP	Up_delay
IDLE	<=bottom_freq	DOWN	Down_delay
DOWN	>top_freq	UP	Up_delay
DOWN	>bottom_freq	IDLE	NA
UP	<bottom_freq	DOWN	Down_delay
UP	<=top_freq	IDLE	ND

hotplug governorによる「慎重な判断」

up 動作周波数が上限値に達しているのでコアを追加
down 動作周波数が下限値に達しているのでコアを削除
idle 動作周波数が上限値よりも低く、下減値よりも高いのでコア個数の変更なし
disable Auto hotplug 機能を停止する

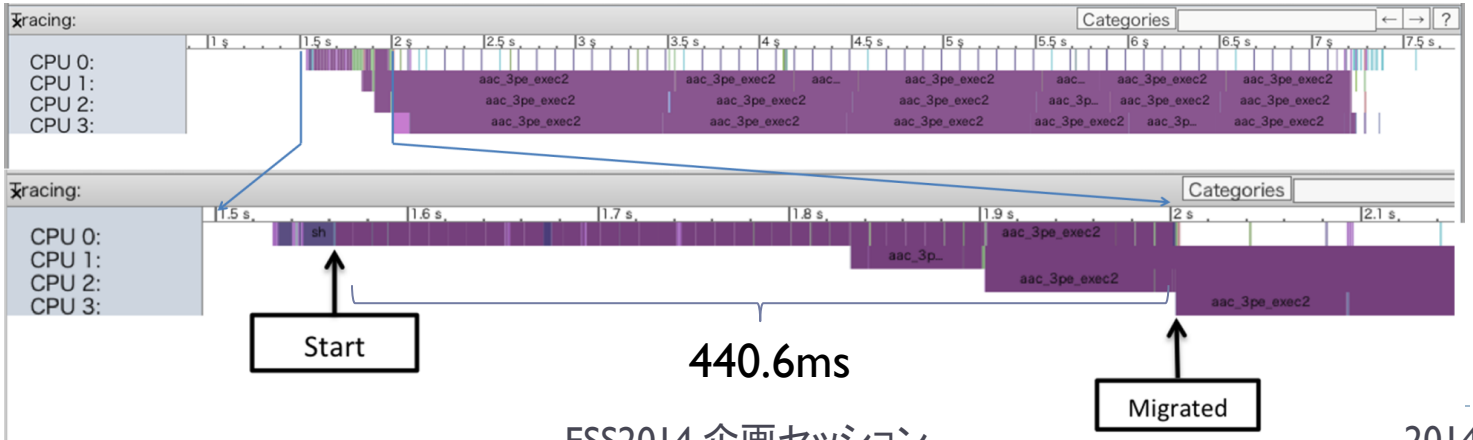


Hotplugが阻害する並列化プログラムの起動

- 並列化ベンチマークの結果
  - 実行時間にばらつき

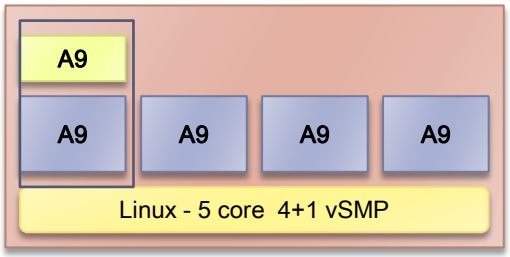
サンプル	1	2	3	4	5	6	7	8	9	10	最速	平均	最遅
実行時間(秒)	5.12	5.08	3.65	5.05	2.78	2.73	5.06	2.74	5.05	2.74	2.73	4.00	5.12

- Systraceによる解析結果
  - Thread生成からCPUのbindまでに遅延 (Start-Migrated 440.6ms)
  - CPUの自動ON/OFF line(Auto Hotplug)が影響

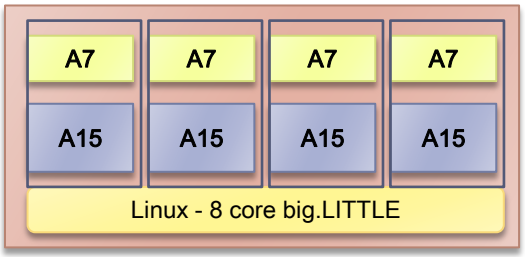


big.LITTLE: Exynos5 の事例

- vSMP の１＋４に対して, 4+4で動作



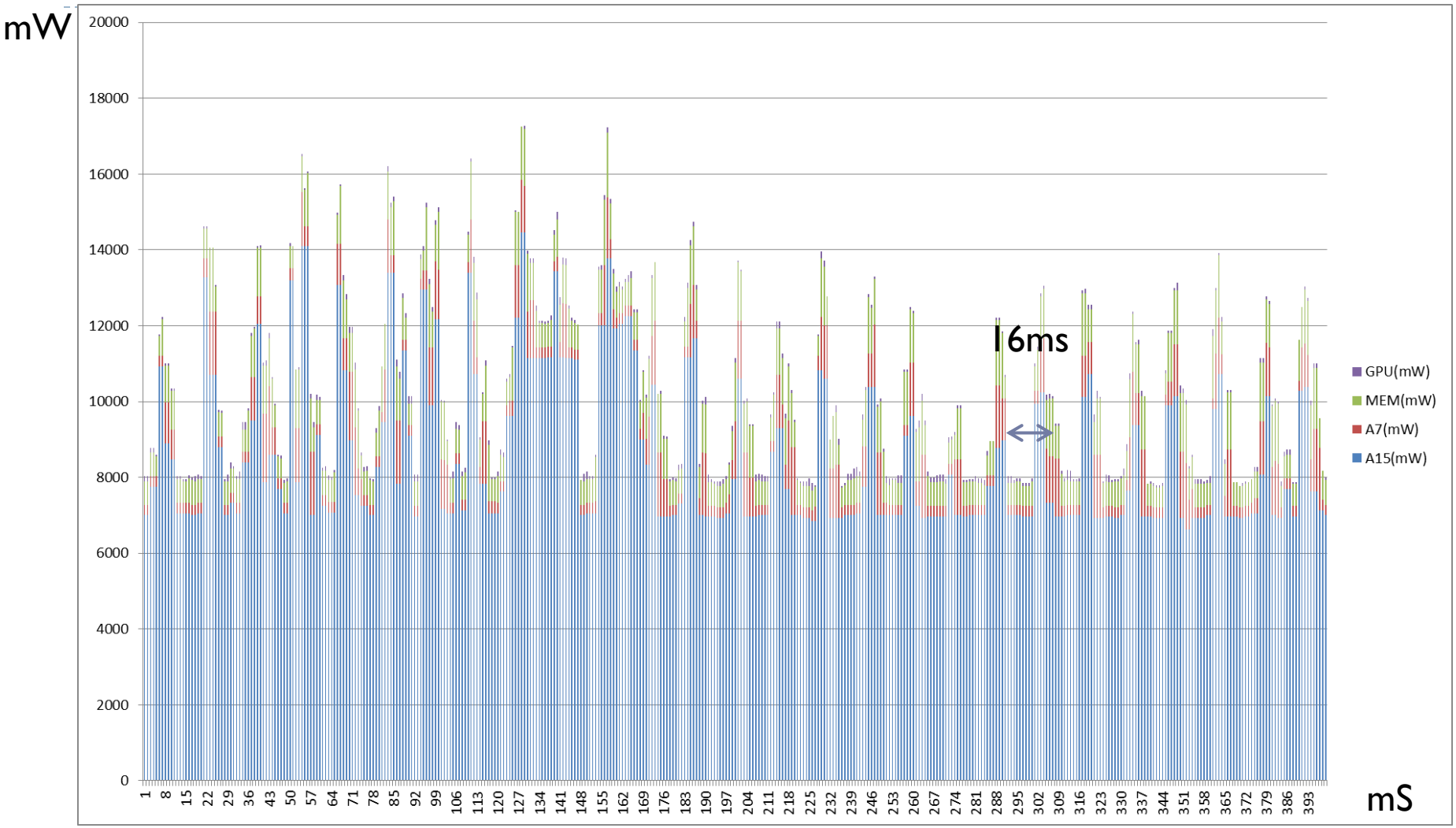
vSMPの構成



Big.LITTLEの構成

- vSMPのA9+A9に対して、A7+A15★
- 利用者からは4コアに見えて、クロックに応じて内部でA7-A15を切り替えて動作
- 制御はCPUFreq

big.LITTLEの電力波形



Exynos 5420

まとめ

自動制御の限界と期待

- 自動制御の限界
  - 過去の状態から将来の状態を予測
  - 予測は当たらない
  - 例）
    - 並列性の負荷が低くてもコアを追加する→無駄
    - 応答性を上げるために投機的にクロック上げる→無駄
  - 実装はCPUの機能を対応したハードウェアIFの追加に限定

⇒制御ロジックはCPUFreqのまま

- 新しい試みへの期待
  - コンパイラによる電力管理コードの出力
  - プロファイルベースのスケジューリング
  - 省電力機構の全面刷新