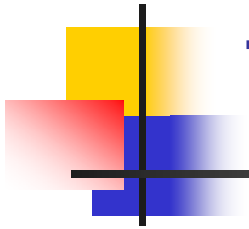
省電力計算機アーキテクチャとOSの資源管理

～細粒度PGアーキテクチャとOSの連携～

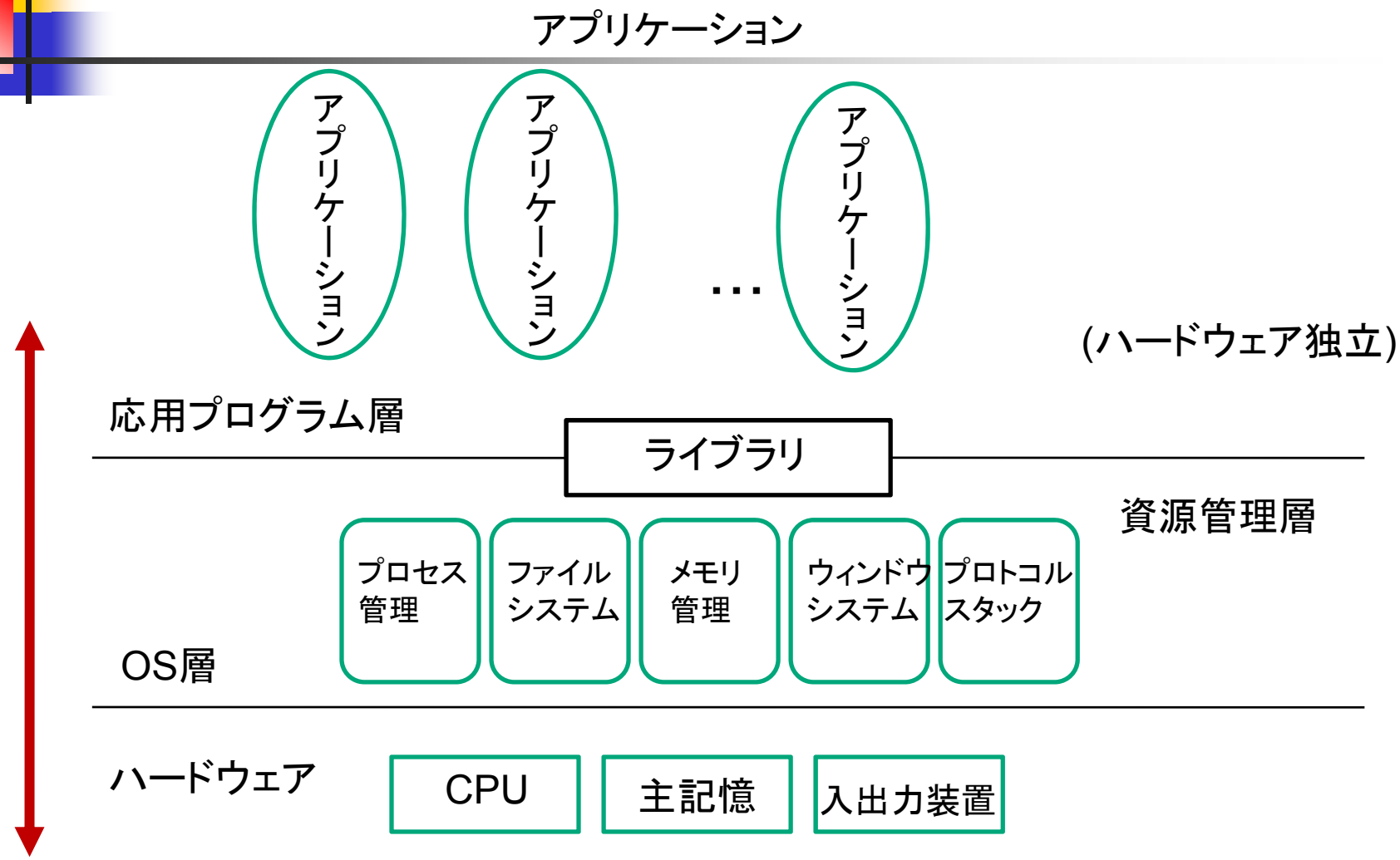
並木美太郎（東京農工大学）

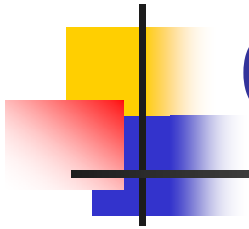


話者の生まれと育ち

- 1983年からOS屋
- プログラミングやソフトに興味を持っていた
- 特に、プログラムの実行基盤としてのOSなどのシステムソフトウェア
- OS以外にも、コンパイラ、インタプリタ、ライブラリ、ウィンドウシステム、プロトコルスタックなどのシステムソフト、UIなども
- 横断分野では、並列分散処理、情報教育なども
- ハードは好き(はんだごて、ラッピング…)
- 去年までIPSJ OS研の主査…

OS～偉大なる中間管理職～





OSの使命

- ハードウェアなどの仮想化：CPU、メモリ、I/O
- 資源管理：最適割当て
- 保護

←本質は概念創出、方式として汎化

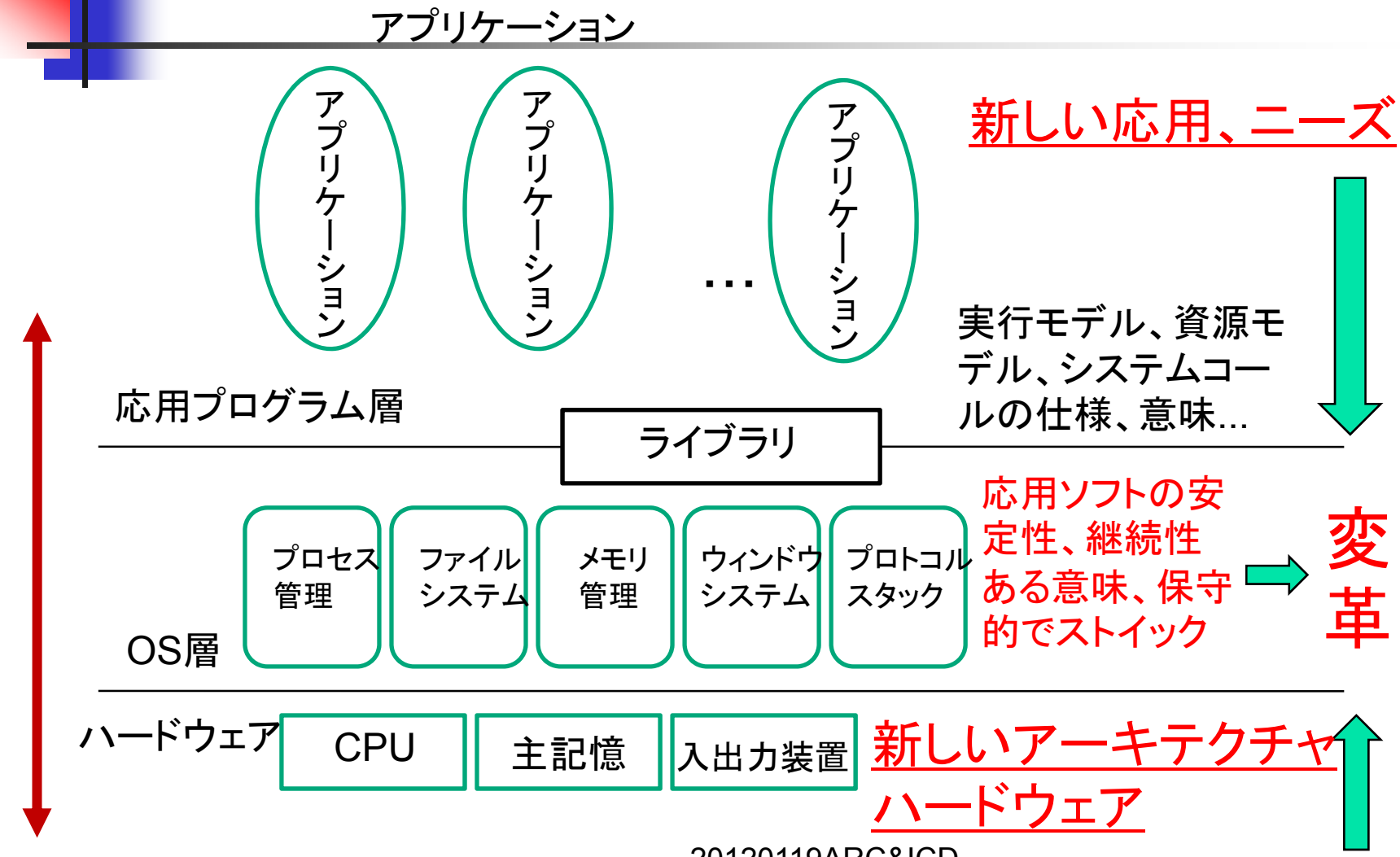
- これらをAPIとして、AP(応用プログラム)に提供すること
- 応用層の知識は必須
- ハードウェア、計算機アーキテクチャを知らないとできない

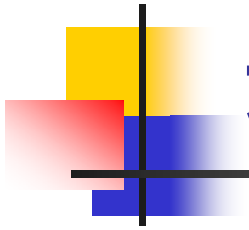
←自分はこの両方ができるから面白いと思った

「ハード、ソフトって分類、関係ないよね」

(人によってはソフトで仮想化と美の極致)

中間管理職は...


アプリケーション
アプリケーション
アプリケーション
…
新しい応用、ニーズ
実行モデル、資源モデル、システムコールの仕様、意味...
応用プログラム層
ライブラリ
プロセス管理
ファイルシステム
メモリ管理
ウィンドウシステム
プロトコルスタック
OS層
応用ソフトの安定性、継続性ある意味、保守的でストイック
変革
ハードウェア
CPU
主記憶
入出力装置
新しいアーキテクチャハードウェア



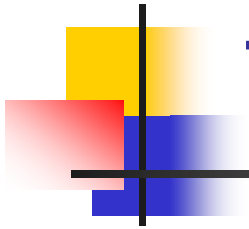
近年のOS研究のトレンド

- 上位層から
  クラウド、セキュリティ→分散、VMM(仮想化)
  モバイル、ユビキタス→携帯、センサ系
- 計算機アーキテクチャから
  マルチコア、メニーコア→HPC、組込み
  省電力

- 電源ON/OFF
- クロック
- 電圧
- 割当て

高速小容量高消費電力 vs
低速大容量低消費電力

←どう組み合わせて隠蔽するか
バランスさせるか
どう調停するか



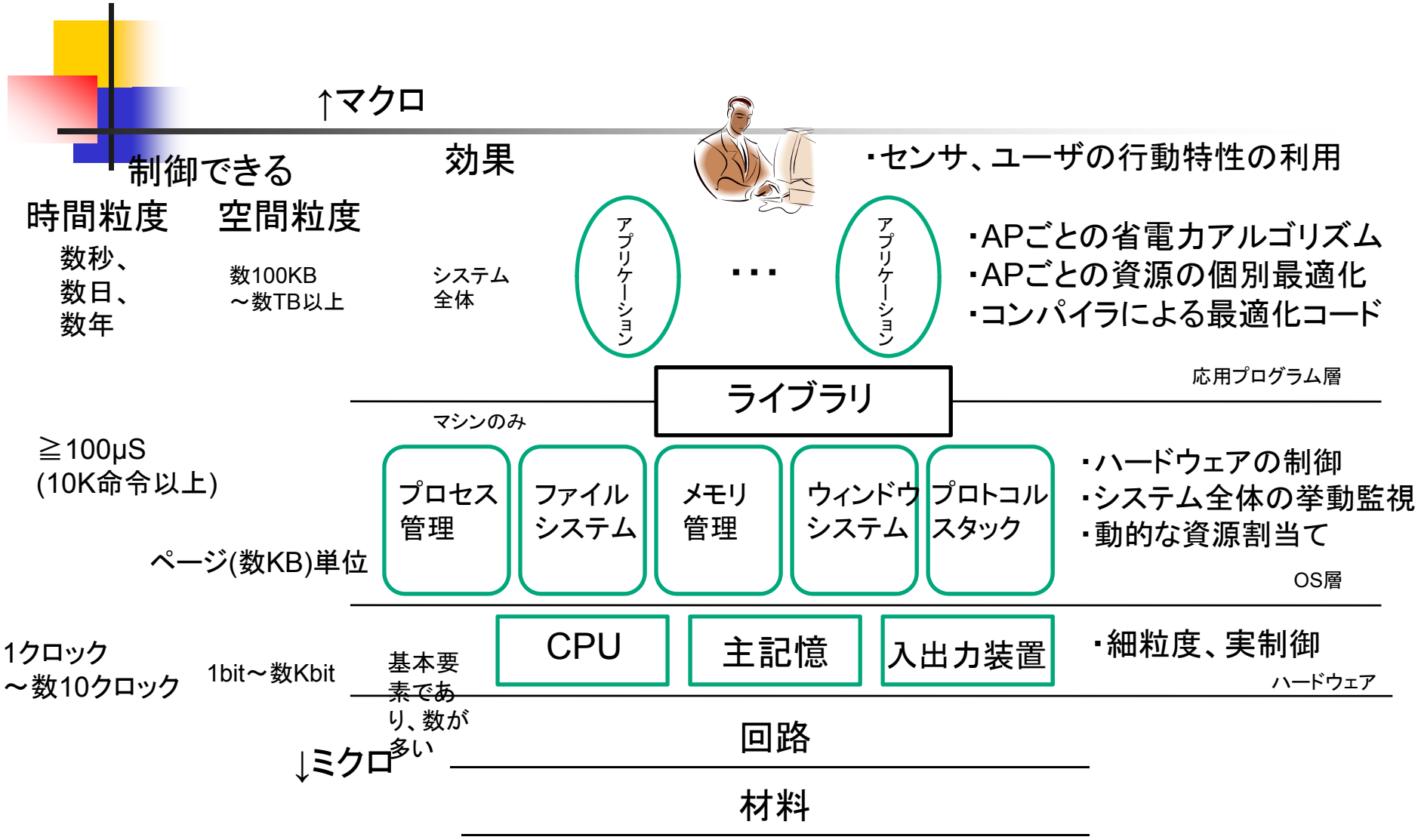
計算機の電源管理

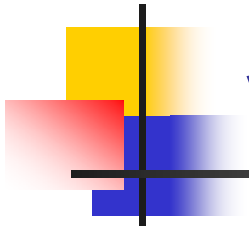
- 単位：ラック、マシン全体、CPU/メモリ/IOなどの構成部品、個々の構成要素
- 構成要素の挙動特性
- ユースケース
- 時間粒度、空間粒度

階層によって異なる

→研究分野でも切れている

階層ごとの特徴：役割分担は変わる





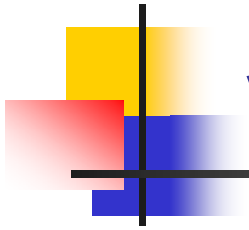
省電力とOS

- サーバ系

- DNSラウンドロビン
- VMM(仮想機械モニタ)

を使ったサーバ集約

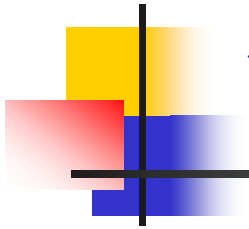
- データセンターで実用化
- 負荷が少ないときにマシンの電源を落とす
- 判断はログベース、統計的解析と予測、学習
- 数分から1時間程度の単位



省電力とOS

- PC系：ACPI(Advanced Configuration and Power Interface)の利用　suspendの状態
  - ディスプレイ、HDDの自動電源断/再起動
  - 電源制御用の8bit μPはいつも通電
- 組込み
- センサネットワーク系：TinyOSでも

  バッテリ駆動の要求から、イベント駆動で間欠的なプログラム実行



資源管理からは

- プロセス実行のDVFS(Dynamic Voltage and Frequency Scaling)

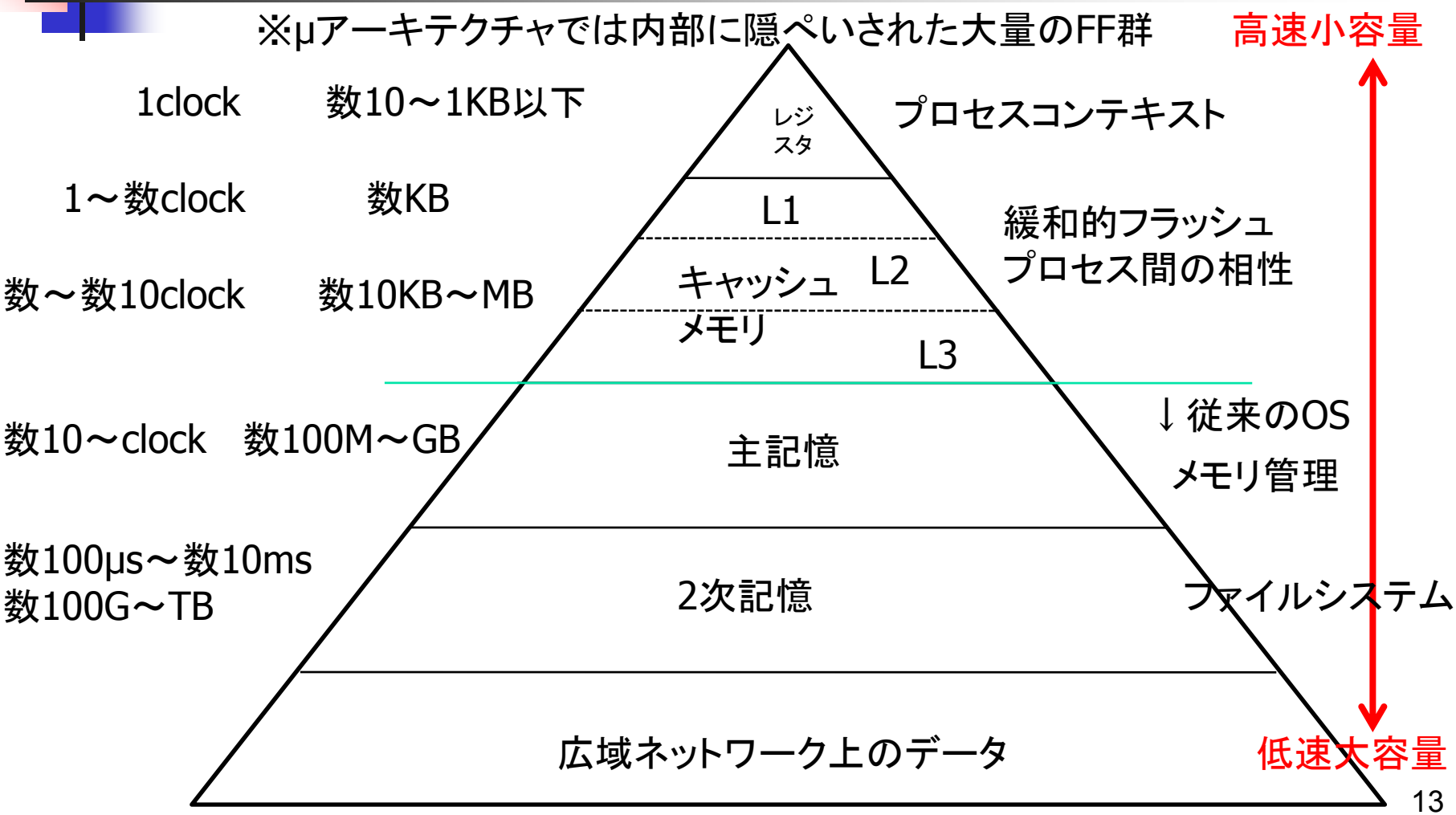
  挙動予測から省電力になるようCPUをDVFS制御

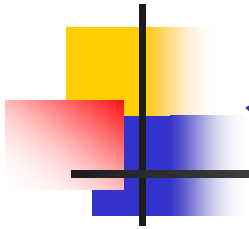
- マルチコア制御

  低負荷時のコア電源断、DVFS制御

これらをサーバ、PC、組込みシステム、リアルタイムシステムに適用した例

メモリ管理：計算機の記憶階層のどこで、どうやるか（パネルででも）

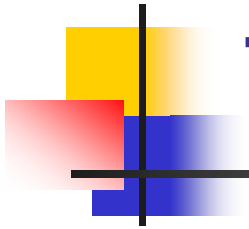




メモリ管理

- 低電力なキャッシュ割当て
- SPM(Scratch Pad Memory)の利用

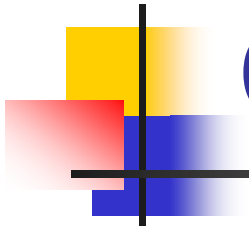
  必要な部分をSPMに移動して、主記憶の電源断を行う



ファイルシステム

- SSD(Solid State Drive)の利用

  HDDに対するアクセスの高速化、低電力化
- ファイルシステム(ログベース)、ブロックデバイスとして利用
- 記憶域の有効利用、書込み時間、書換え上限が課題



OSによる電力制御

- 観測と制御
  - OSはシステム全体の利用状況はわかる
  - 資源割当てと調停の全権←トレードオフ
- 時間の単位は比較的粗粒度：≧1ms(頑張って100μs)
- 記憶の割当てはページサイズ程度(数KB)
- 例外 or system callで起動され、実行される
- これ以上は、パフォーマンスカウンタからの観測結果から戦略を決定

今日は何度も出ている式

- $Pon_i = AnCV^2f + nI_{leak}V$

  $P = \Sigma Pon_i$

- 空間分割は、構成方式やアーキテクチャ、OSやランタイムの分割方式
- 時間軸制御：ユニットのon/off制御

→これをソフトとハードでどうするか

新しいハードウェアとOS

- 細粒度PG　Geyser CPUとそのOS
  ←演算器でのPG、一過的
- マルチコア/メニーコアの省電力制御
  ←4～8コアはAMD Phenomでやってみた
  アクセラレータ系はいずれまた
- 状態・記憶系
  ←ノーマリオフ：状態保持、パネルで

OSと省電力アーキテクチャの協調事例 ～細粒度PGの例～

戦略的創造研究推進事業CREST

「情報システムの超低消費電力化を目指した技術革新と統合化技術」において

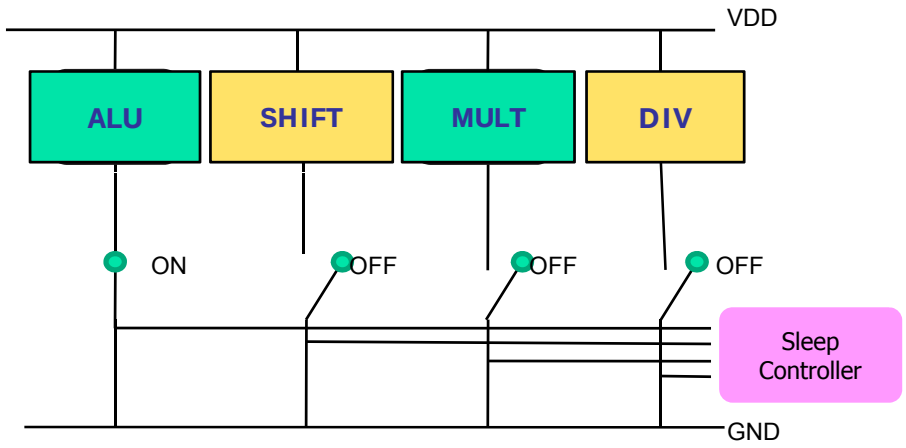
「革新的電源制御による次世代超低電力高性能システムLSIの研究」

回路 (芝工大 宇佐美研)	アーキテクチャ (東大 中村研)	アクセラレータ、CPU実装 (慶応大 天野研)	コンパイラ (東大 中村研 電通大 近藤研)	OS 並木研担当

回路からOSまで協調

細粒度PG Geyser CPU

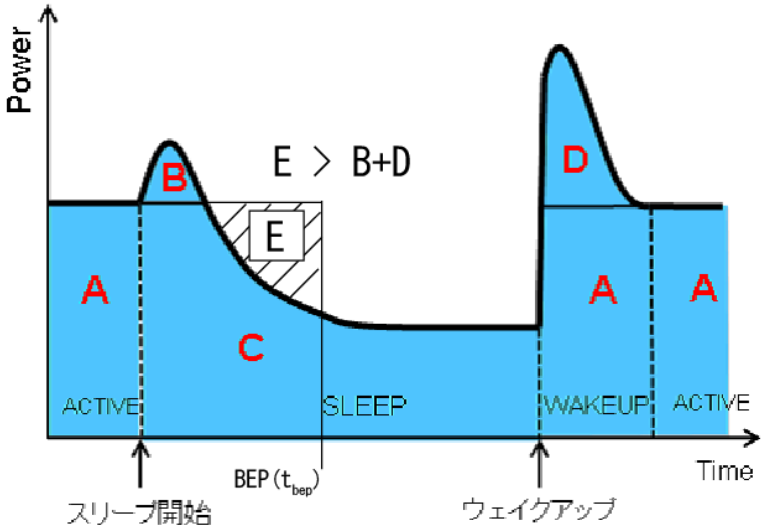
- 細粒度PG(Power Gating)が一つのテーマ
- Geyser CPU：MIPS R3000をベース
- ALU、SHIFT、MUL、DIVの4ユニットそれぞれで、1命令ごとにユニットを使わなければ電源を切る



細粒度PGの課題(1)

(1) 適切なスリープポリシーの設定

- スリープ時の電力モデル
  - オーバヘッド(B,D)よりも削減電力(E)が大きくなるスリープ期間(損益分岐点)をBEP(Break Even Point)と定義
  - 短い期間でON/OFF繰返すと電力増大→PGせず、アクティブの方が良い
  - BEPを下回るスリープ(BEPミス)が頻発すると電力ロスにつながる



最小1命令サイクルでスリープ

→適切なスリープポリシー、コードの利用が必要

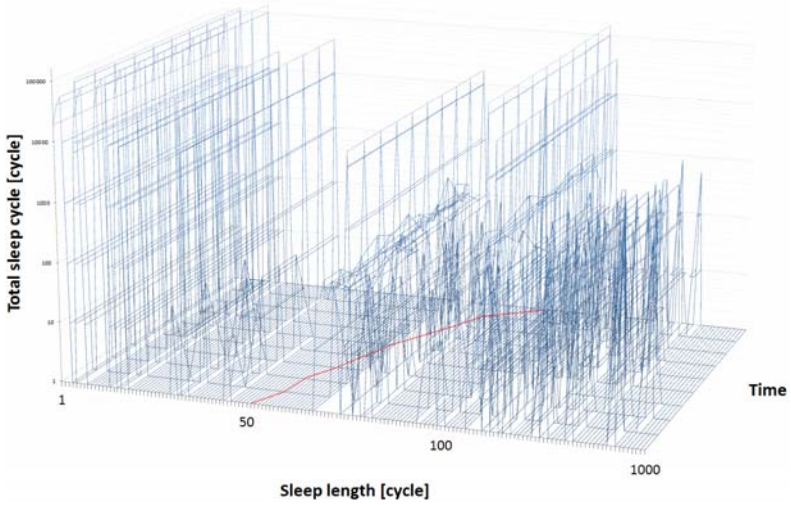
細粒度PGの課題(2)

(2) 温度によるBEPの変化

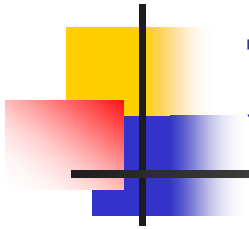
- リーク電力は温度によって変わる

  →BEPも変わる

  →実行時に適切なスリープポリシーを選択する必要有

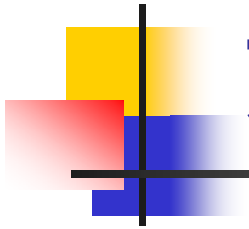


例　SHIFTのスリープ時間の分布



方針(1)

- ソフトとハードの役割分担
  - 細粒度は回路とハードで頑張る、粗い流度と比較的長い時間(≧1ms程度)をOSから制御
  - ソフトは観測結果から基本方針/戦略の選択制御
  - ソフトウェア制御可能な機能をCPUに追加



方針(2)

- 実行時(動的)情報を活用
- 静的解析可能なものはコンパイラで、実行時の情報を基にOSが制御
- 実行時情報の収集機能の追加

  各種パフォーマンスカウンタ

  キャッシュヒット/ミスなどのほかに、

  リークモニタ(温度)、PG統計採取をハードで追加

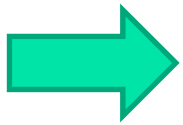
Geyser CPUの持つPG制御機能(1)

- PG実施方針（スリープポリシー）

  (1) 動的にパワーゲーティング

  (2) キャッシュミス時のみスリープ

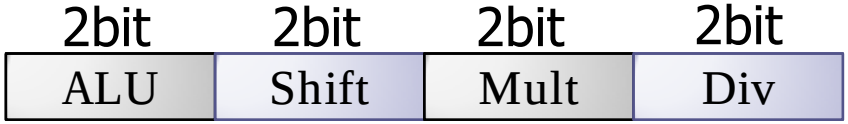
  (3) 常にアクティブ（スリープしない）



ソフトウェア(OS)で
スリープポリシーを制御

- スリープポリシーを演算器ごとに制御可能
  - 細粒度PGのインタフェース（特権レジスタ: PGStatus)

    CP0レジスタの一つとして実装



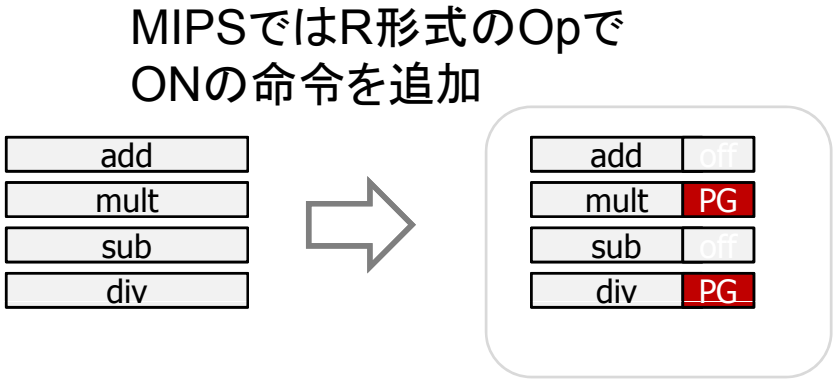
00 : 動的PG(通常ポリシー)

01 : キャッシュミス時のみスリープ

11 : 常にアクティブ

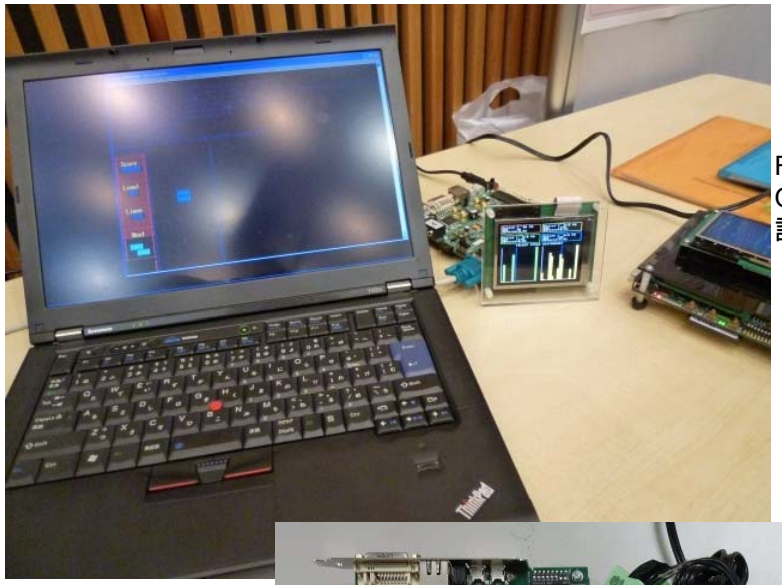
Geyser CPUの持つPG制御機能(2)

- CP0の他に、R形式の命令形式でスリープしないものを追加 (Opを別の値でFunctは同じ)
- コンパイラの研究用

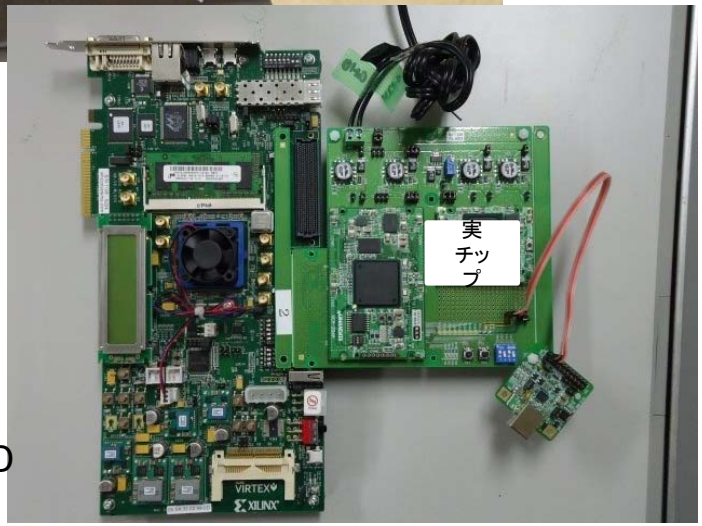


Geyser CPUの実装とテストベッド

- Geyser-0,1,2,3 と三つのバージョン
- VDECで実チップ化
- 現在、Geyser-3のテスト中
- 本体は主として慶應大学が実装
- 並木研でCP0を追加修正、テストと全体の修正、OS実装
- FPGA上にも移植：独自組込みOS(開聞)、Linuxが稼働

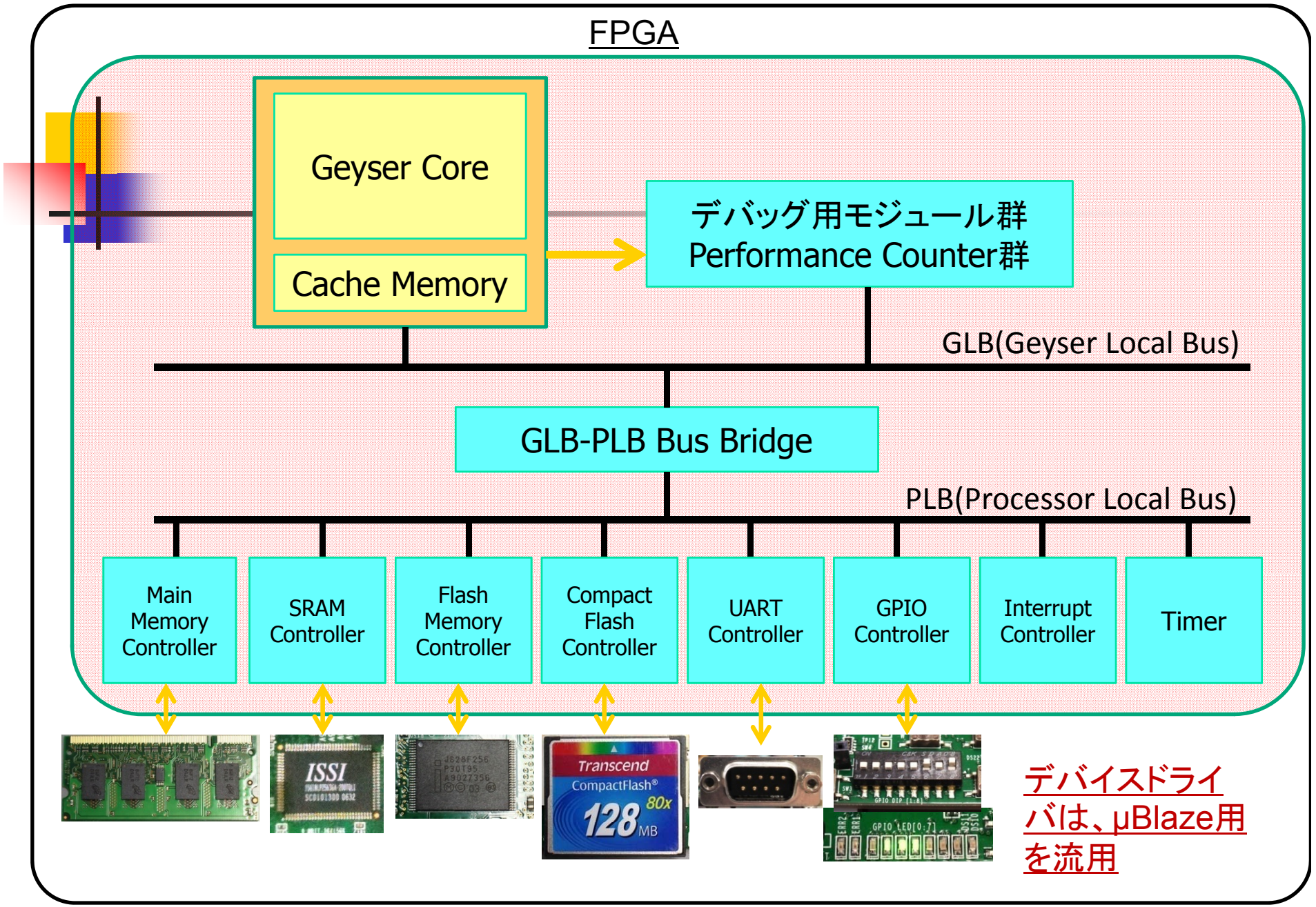


FPGA版でOS開発と評価



実チップはI/O、主記憶をFPGAで

ML501,ML605 Evaluation Platform, Spartan-6

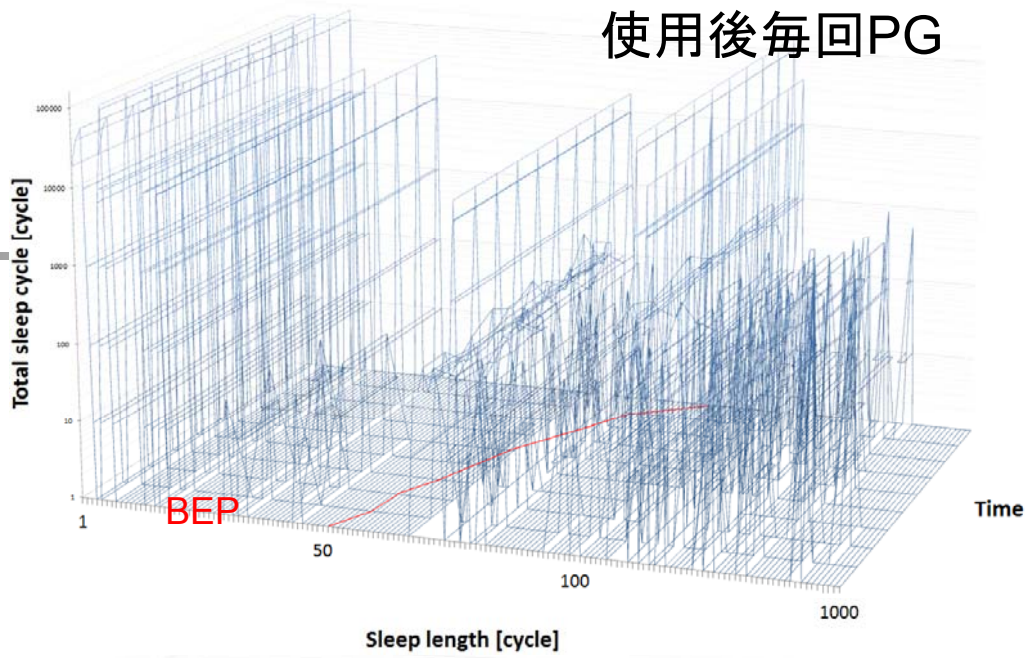


FPGA上で実行しながら
実時間でデータを採取、
OSで活用できる

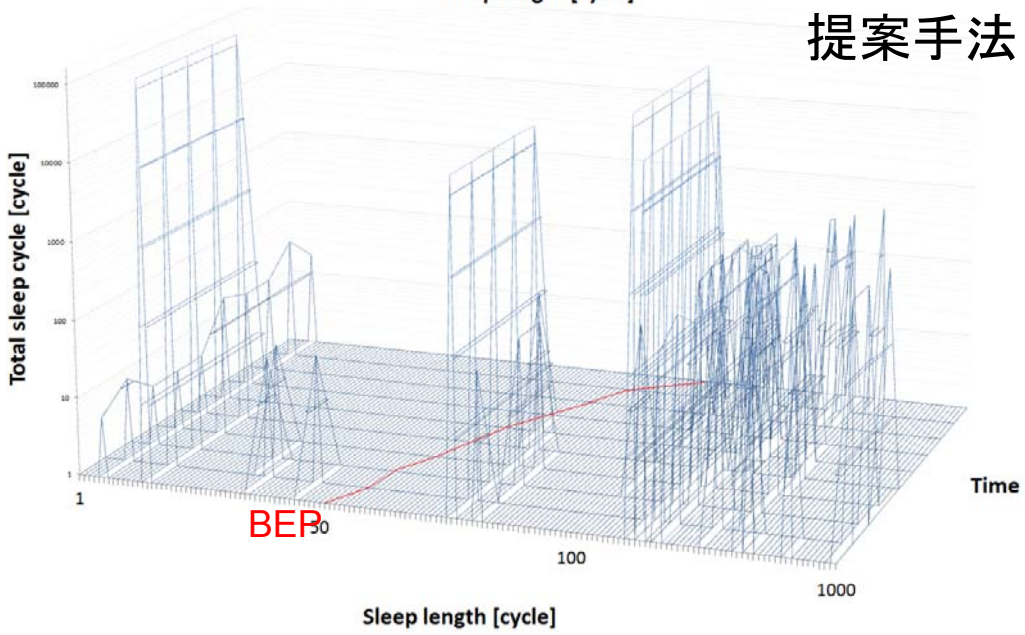
Shifter

ベンチマーク: Blowfish
熱環境: 56-77 [℃]
リーク電力削減率: 36.1%

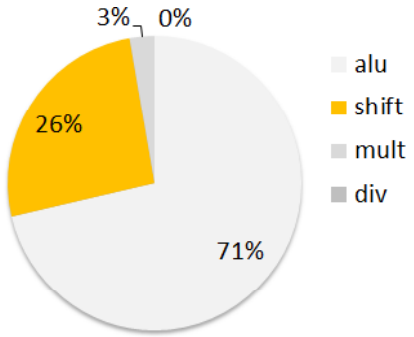
使用後毎回PG



提案手法



各ユニットの使用頻度



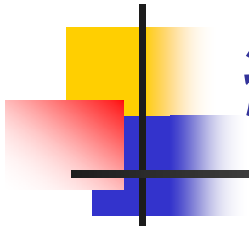
リークモニタ、パフォーマンスカウンタなど

- リークモニタ(担当：芝工大)はon chip化、CPUから読出せる設計
- PG状況：プロセッサ内部のPG信号線をチップ外まで引き回して、4ユニットのON/OFF状況を外部から計測可能

→FPGAでカウンタ：nクロックのスリープがm回
の頻度情報(m=C(n))を作成、OSから読み出す設計

- これら以外も大幅にRTLを修正

OS屋がアーキテクチャを直接変更できるメリット
(ただし、実チップの配置配線は餅は餅屋)



細粒度PGプロセッサ向けのOS研究

- BEPミス率を減らすスケジューラ(開聞,Linux)
- 温度に適応するスケジューラ(開聞,Linux*)
- 温度に適応する最適化コードを選択するOSメモリ管理(開聞,Linux*)

  ↑細粒度PGの効果保証をOSで
- リアルタイムOSとPG制御(開聞*)

(*は近日中に発表予定)

BEPミス率を減らすOSスケジューラ

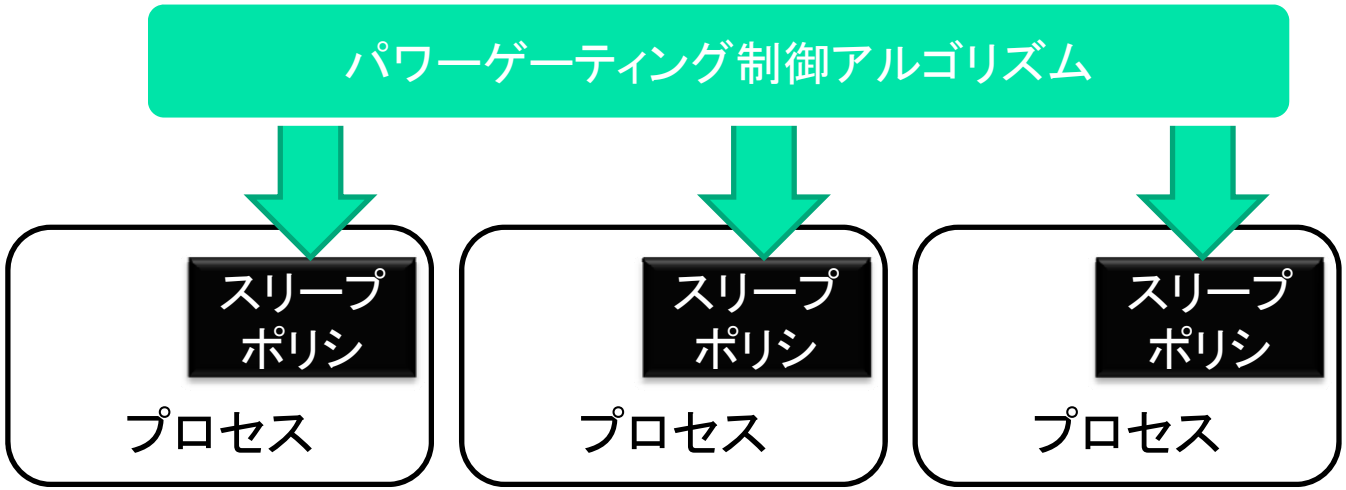
BEPミス率を勘案したOSスケジューラ

- 電力的に不利となるパワーゲーティングがどれくらいあるかの指標
  - パワーゲーティングによりスリープした全サイクル数のうち、BEPを越えないサイクル数の割合
  - BEPミス率が大　→　電力的に不利となる細かいON/OFFが多い

$$\text{BEPミス率} = \frac{\sum_{i=1}^{\text{BEPサイクル}} \text{iサイクルスリープした回数} \times i}{\sum_{i} \text{iサイクルスリープした回数} \times i}$$

プロセスごとのパワーゲーティング制御

- プロセスは処理内容によって使用する命令などの特徴に偏りがある
  - ユニットの使用頻度はプロセスによって変化する
    →プロセスごとにパワーゲーティングを制御
- プロセスごとにそれぞれスリープポリシを持たせる
  - プロセスのコンテキストスイッチにより切り替え

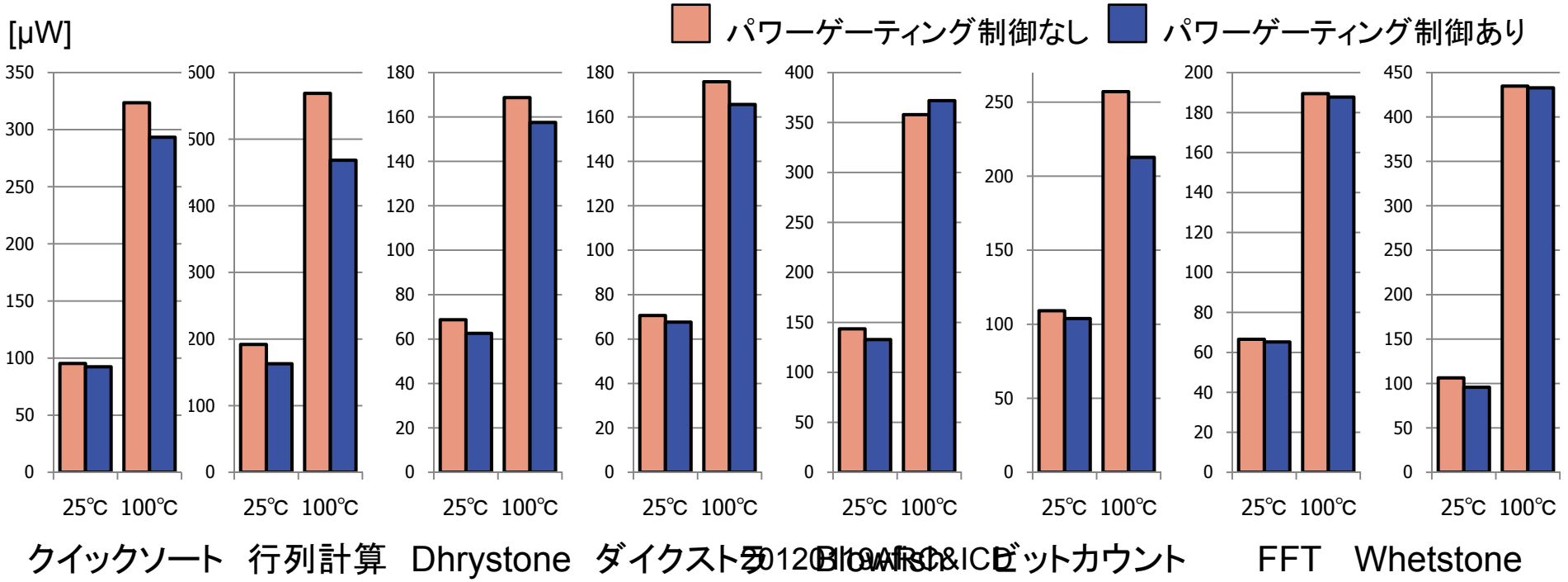


PG制御アルゴリズム

- パワーゲーティング制御アルゴリズム
  - スリープのパフォーマンスカウンタからBEPミス率を取得
  - BEPミス率と閾値を比較して、スリープポリシを決定
    - BEPミス率が閾値を上回る
      →電力的に不利なパワーゲーティングが多い
      →パワーゲーティング動作を粗粒度にするようOSで制御
    - BEPミス率が閾値を下回る
      →パワーゲーティング動作を細粒度にするようOSで制御

評価：スリープ時平均リーク電力

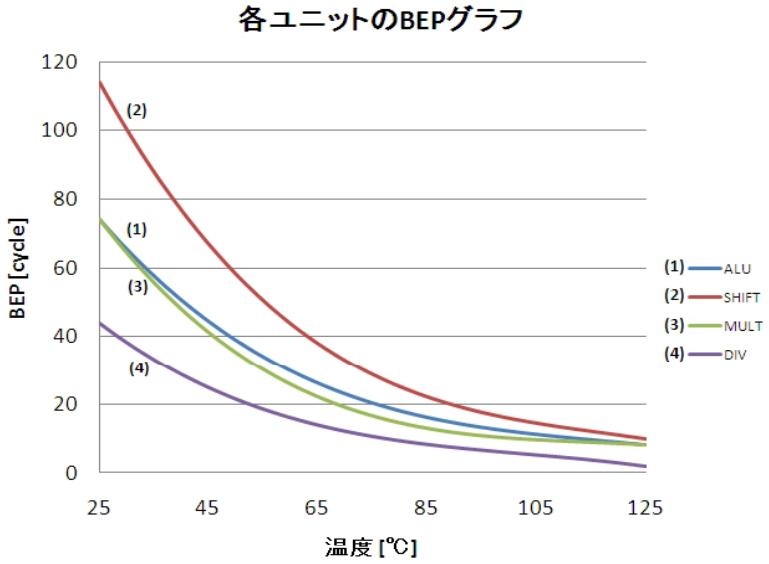
- スリープ時平均リーク電力を評価
  - ベンチマークをシングルタスクで実行
- 制御なし時に比べ3.0～23.2%削減
  - とくに行列計算の電力削減効果が大



温度変化に適応するOSスケジューラ

本研究の目標

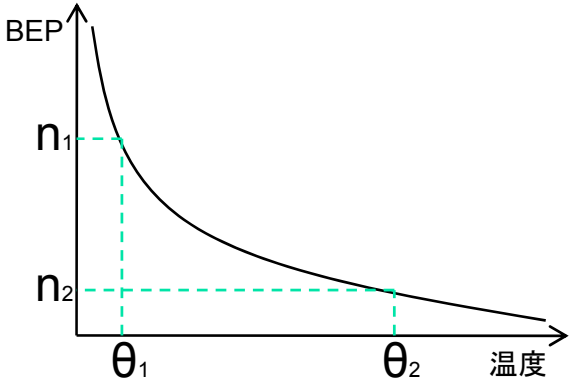
- 温度に適応するスリープポリシー制御方式の実現
  - 各演算ユニットのBEPが温度により変化する点に着目
  - 温度に適応するスリープポリシーを選択
  - 細粒度PGの効果を高める



- 温度が変化する条件を仮定した電力削減効果を評価
  - 温度の変化をエミュレーション
  - スリープポリシー制御手法の妥当性を実証

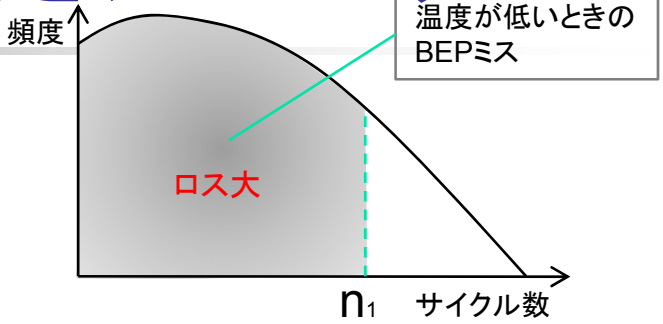
スリープポリシー決定アルゴリズム

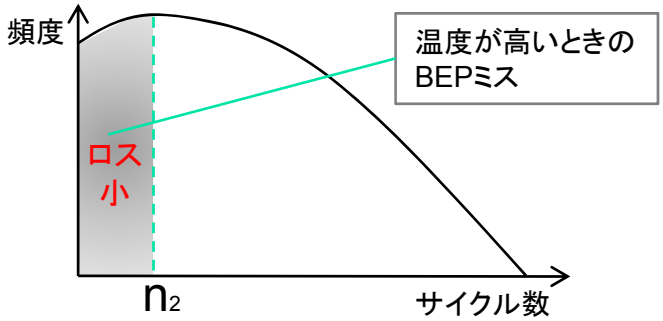
- 温度に適応するスリープポリシーの決定
  - BEPの温度変化を考慮し PGの時間粒度を決定



θ1

θ2





  - 温度閾値θTHを設定して「動的PG」と 「キャッシュミス時のみスリープ」を切り替え
- スリープポリシーの制御モジュール
  - 閾値温度設定部：スリープ頻度情報から閾値温度θTHを決定
  - スリープポリシー制御部：温度に適応するスリープポリシーをコアに設定

閾値決定方法

- 温度閾値θTHを決めるためのサイクル数nTHの設定方法（メリットとデメリット）
  - (方法1) アドミッションテストにより事前に得た情報を用いる方法
    - スリープ頻度の計測モジュールが必要ない
    - スリープ頻度特性の判定が不要
    - スリープ頻度特性がタスクにより変わるため効果が得にくい
  - (方法2) 実行時の情報を用いる動的方法
    - スリープ頻度の計測モジュールが必要
    - スリープ頻度特性の判定が必要
    - タスクの性質により動的に閾値を設定可能
- スリープ頻度から閾値温度を決定
  - スリープ頻度分布の中央値にBEPが一致する温度
  - ただし実行時にスリープ率の高いユニットは温度による制御をせず「動的PG」ポリシー固定（適用可否判定）

$$\sum_{n=1}^{n_{TH}} (m_n \times n) = \frac{1}{2} t_{sleepAll}$$

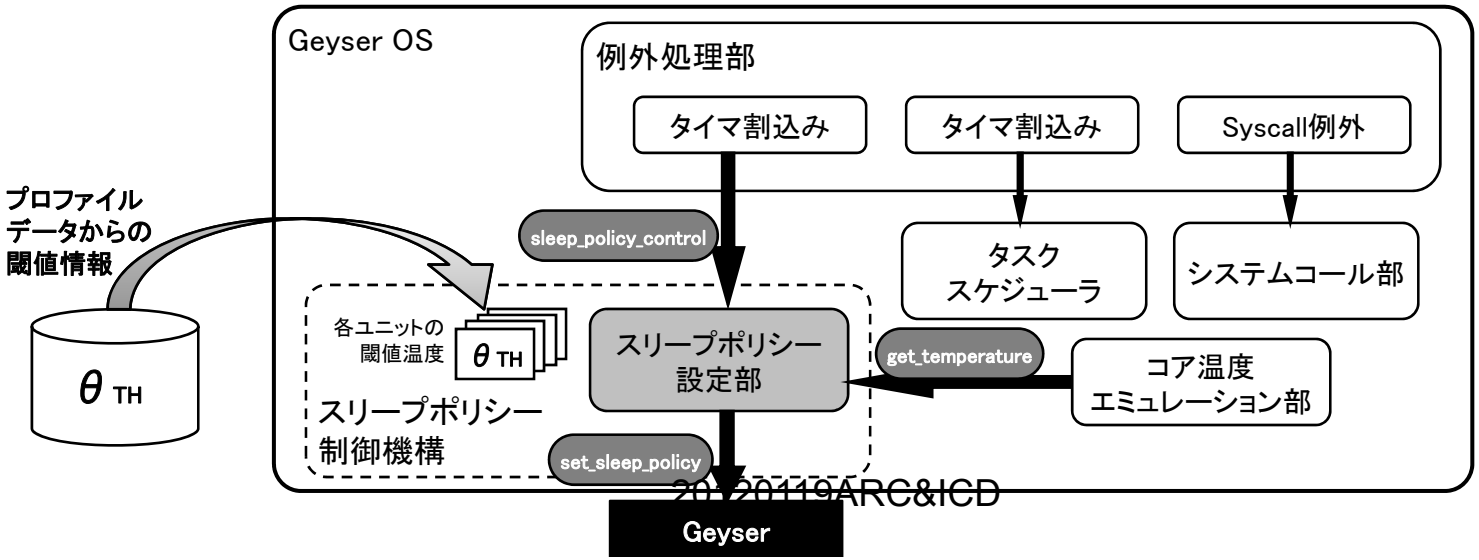
$t_{sleepAll}$: 全スリープサイクル数
m_n: nサイクルスリープの頻度

評価基盤への実装

(1) アドミッションテスト方式による
スリープ頻度制御機構の実装

- あらかじめ決めた閾値温度を保持
- タイマ割込みで起動
- 温度に適応したスリープポリシーを
ユニットごとに選択しGeyserコアのPGStatusレジスタに書き込み

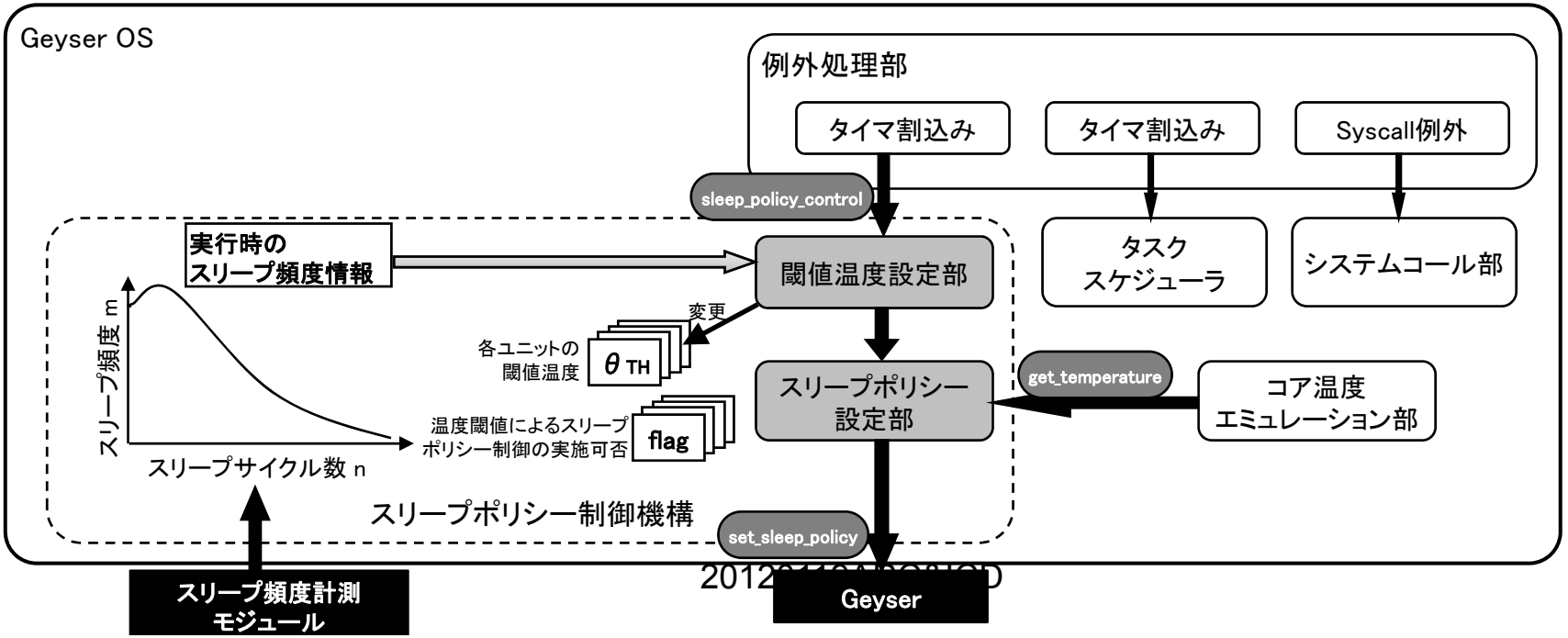
オーバヘッド
0.25%



評価基盤への実装

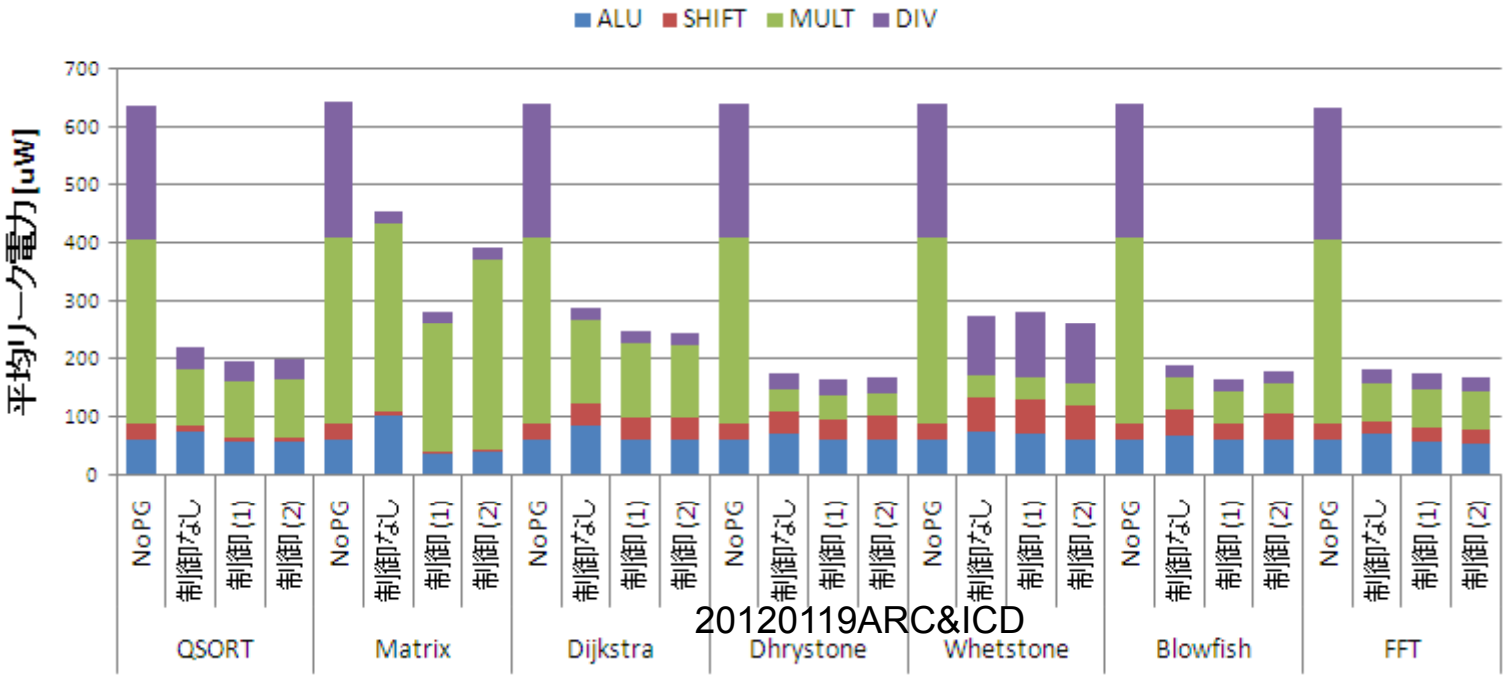
(2) 実行時情報を用いる方式による
スリープポリシー制御機構の実装

- タイマ割込みでスリープポリシー制御機構を起動
- スリープ頻度から閾値温度を設定
- 閾値温度と，適用可否判定のフラグを保持



評価結果(平均リーク電力)

- (1) アドミッションテスト方式
  - 最小 -2.5% (Whetstone)，最大 38% (Matrix) 平均リーク電力を削減
  - 演算ユニット全体でリーク電力を平均約12%削減
- (2) 動的決定方式
  - 最小 4.0% (Blowfish)，最大 16% (Dijkstra) 平均リーク電力を削減



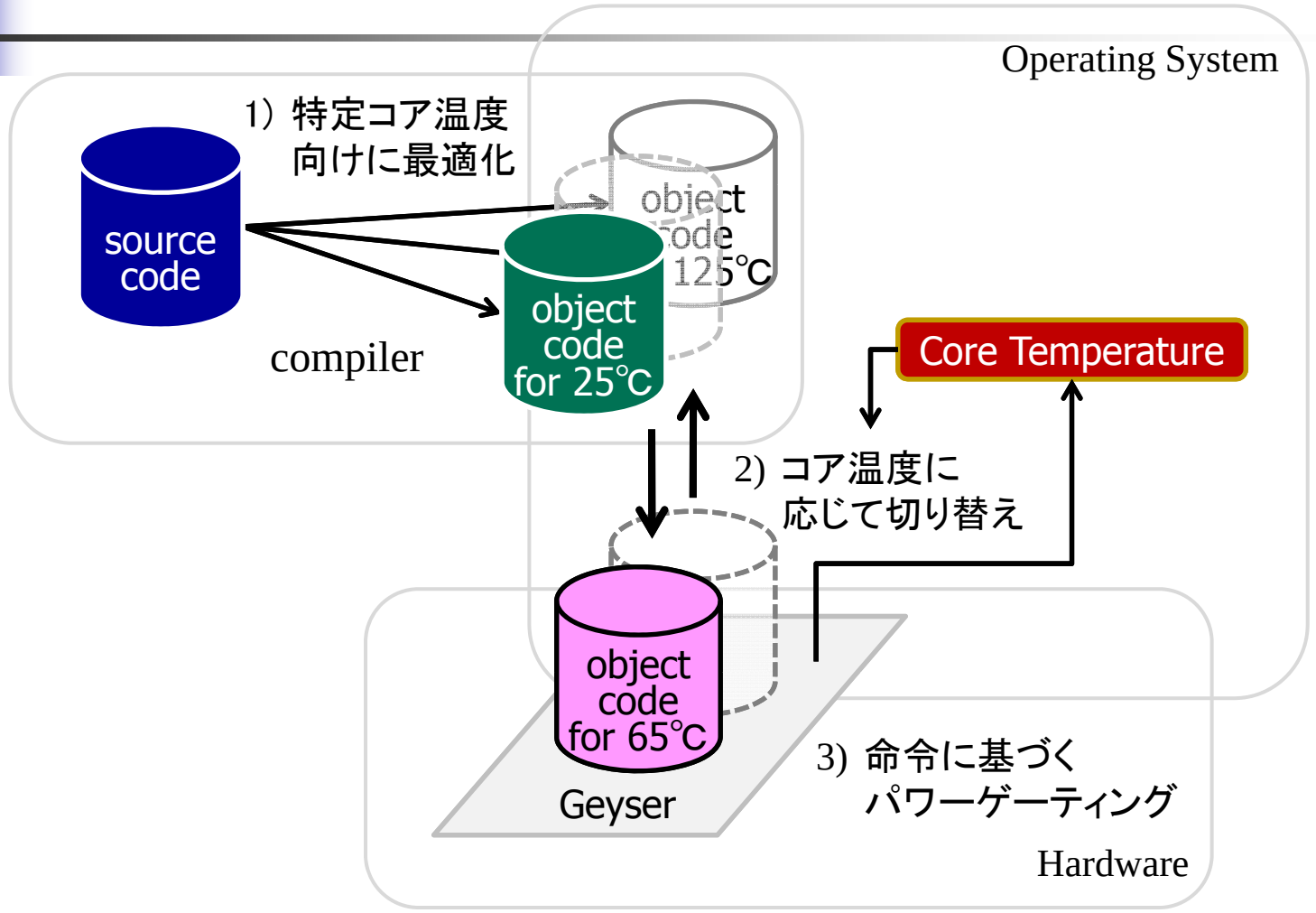
温度に適応する最適化コードを選択するOSメモリ管理

研究目標

- コア温度変化の影響の隠蔽
  - コア温度変化に伴うBEPの変動への対処
- コンパイラとOSとの協調動作
  - 静的解析と動的制御を組み合わせたPG制御
- 実行時コア温度変化に基づくオブジェクトコード管理
  - 実行時コア温度変化に基づいてOSがオブジェクトコードを切り替え実行

パワーゲーティングによる消費電力低減効果の向上

提案手法の概要

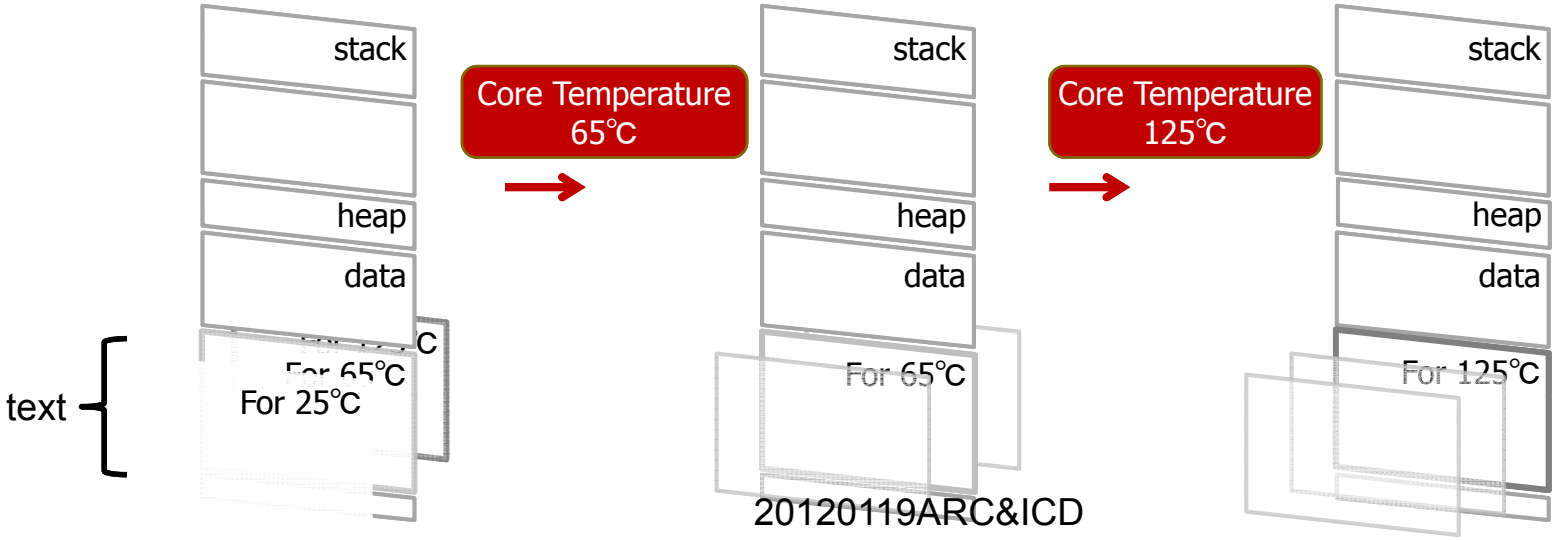

Operating System
1) 特定コア温度
向けに最適化
source code
object code 125℃
object code for 25℃
compiler
Core Temperature
2) コア温度に
応じて切り替え
object code for 65℃
Geyser
3) 命令に基づく
パワーゲーティング
Hardware

温度に適合したコードを実行するメモリ管理

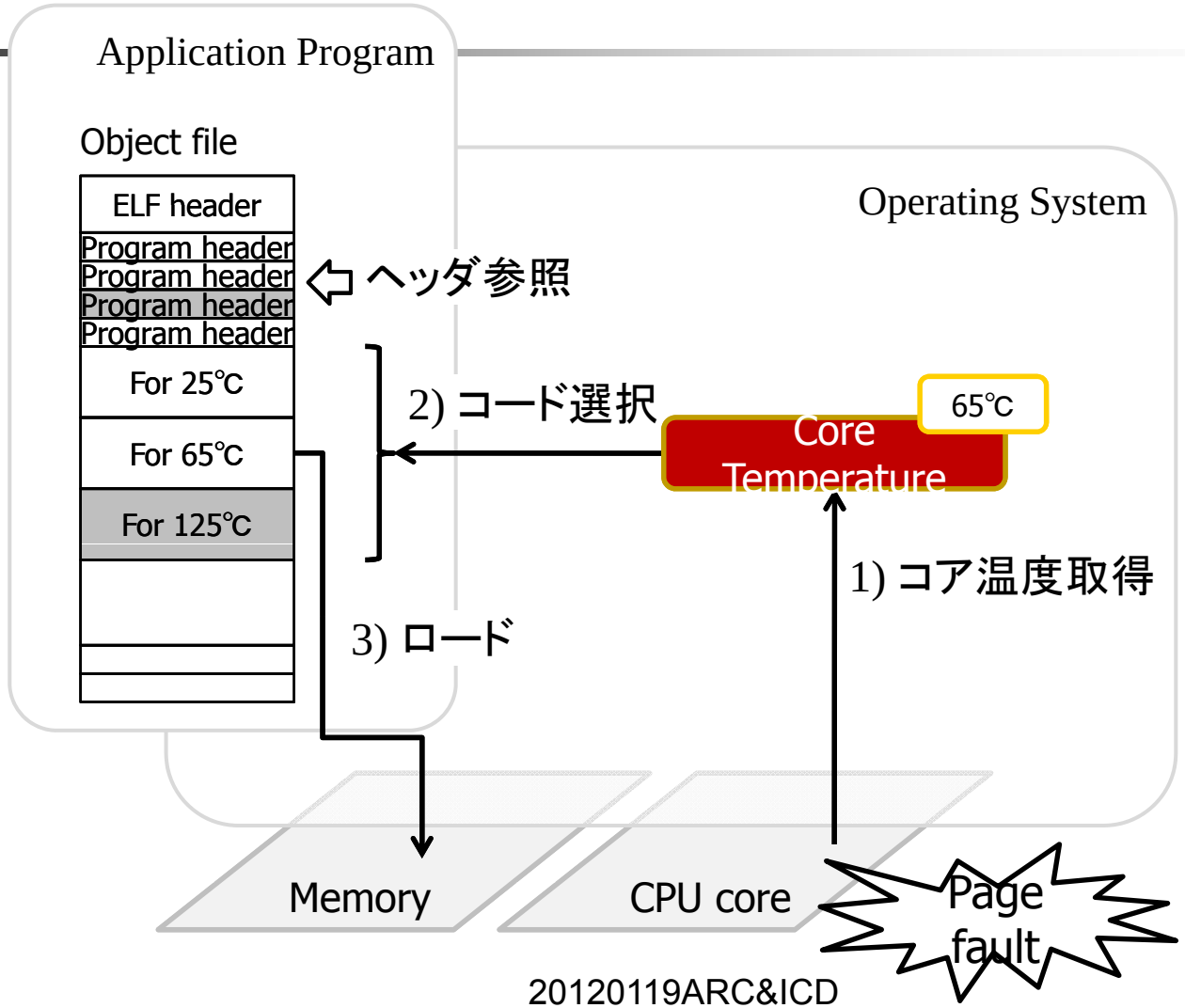
仮想アドレス空間

仮想記憶により各タスクに一意の仮想アドレス空間を提供

- テキスト領域の多重化
  - 各コア温度向けに一意のテキスト領域を提供
  - コア温度変化に基づくテキスト領域の動的切り替え



コード管理:コアごとに応じたコードの切り替え

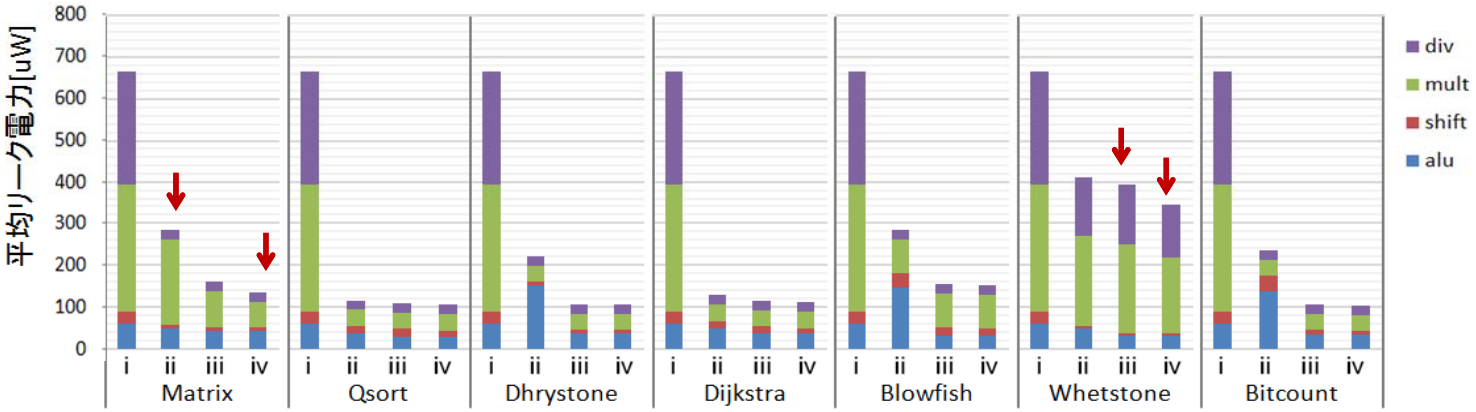


評価(1)

消費リーク電力の評価(1)

実行時オーバヘッドは3.2%

- 全温度平均
  - PGを行うことで消費リーク電力を大幅に削減
    最も効果の高い提案手法（iv）では平均77.3%削減



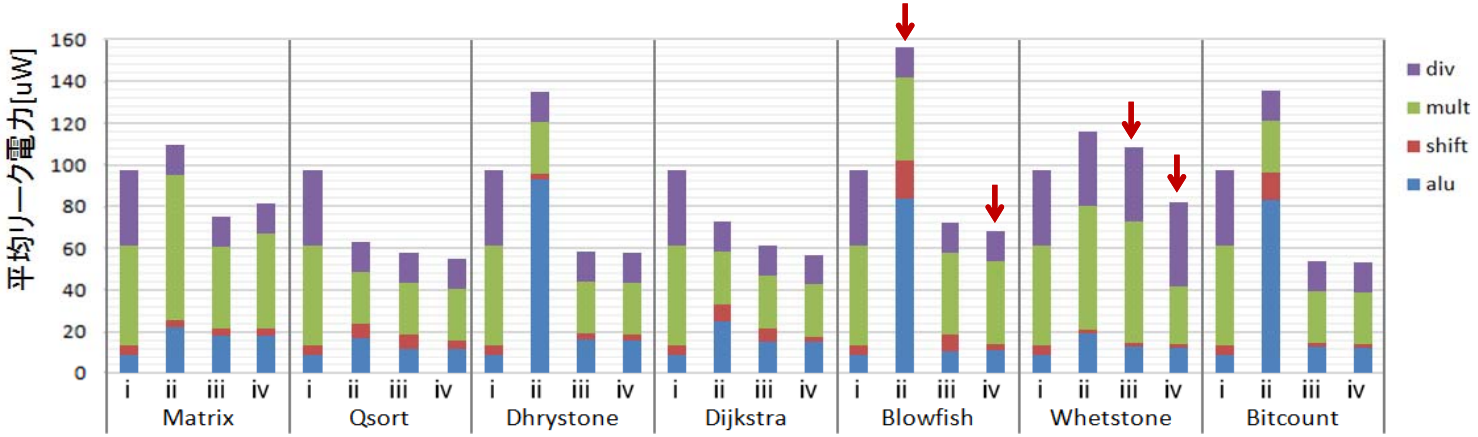
提案方式(iv)

- ハードウェア自律（ii）との比較　：最高55.7%,平均35.0%削減
- コンパイラ単体（iii）との比較　：最高15.6%,平均5.5%削減

評価(2)

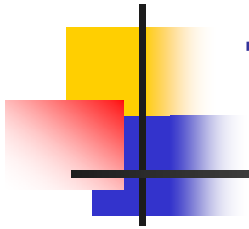
消費リーク電力の評価(2)

- 25℃
  - 一般的に低温下ではBEPが短いためPGによる省電力効果は低い
  - 手法 iii , iv は低温下でも効果を発揮



提案方式(iv)

➢ハードウェア自律(ii)との比較　　：最高60.7%,平均37.6%削減
➢コンパイラ単体(iii)との比較　　：最高24.5%,平均4.9%削減



まとめ

- 計算機システムの省電力化においても、OSの役割は重要
- ただし、粗粒度
- 細粒度PG：資源管理のモデルと実行時情報に基づく資源管理手法
- 階層を越えた研究
- 新しいハードウェアへの対応

さらなる効果を目指すには

- 今日の話は演算器系のみ
- 普通のCPUには：キャッシュ,TLB,回路内FF
- 状態保持の多い制御系と

  一過性の計算主体の演算系の徹底分離

  onにすべきところだけon, そこを減らす

  →メニーコアのアクセラレータ

  　いずれ、また(格闘中)
- メモリ、どうする？　　→パネルへ続く