修士学位論文発表

マルチスレッドアーキテクチャにおけるシステムソフトウェアの研究

A study on Systems Software
for Multithreaded Architecture

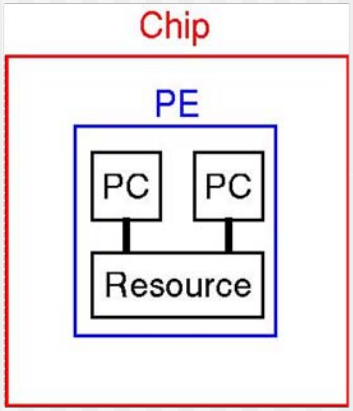
2004 2/12

東京農工大学大学院 工学研究科
情報コミュニケーション工学専攻 並木研究室
03646109 笹田 耕一

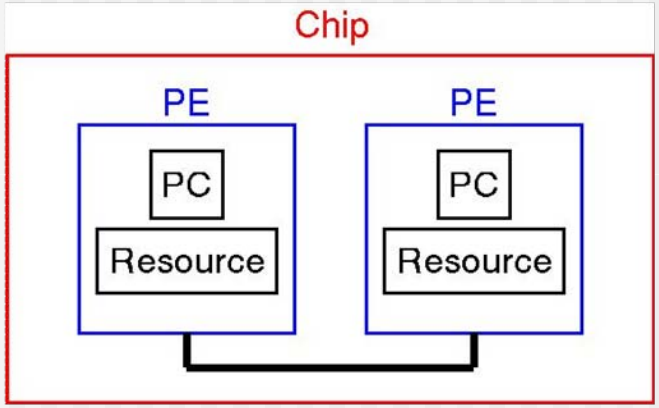
背景

- マルチスレッドアーキテクチャプロセッサ

SMT(Simultaneous Multithreading)



CMP(Chip Multi-Procesor)



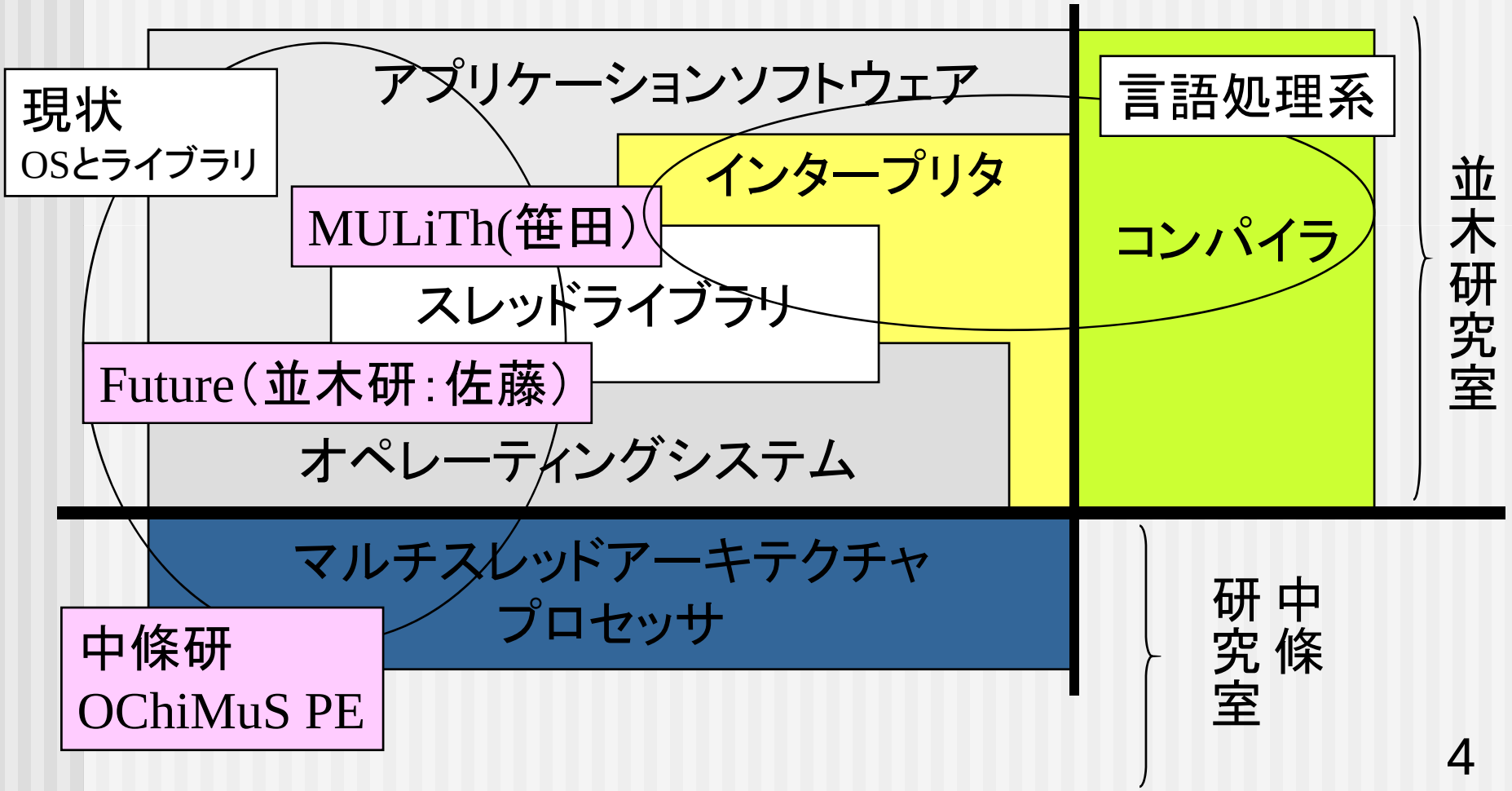
  - 1チップ上で複数の命令流(実スレッド)を並列実行
  - *ILPの追求から TLP の活用へ*
  - いくつかの製品
    - Intel社が製品化(Xeon / Pentium4 プロセッサ)
    - IBM Power4/Power5

問題点

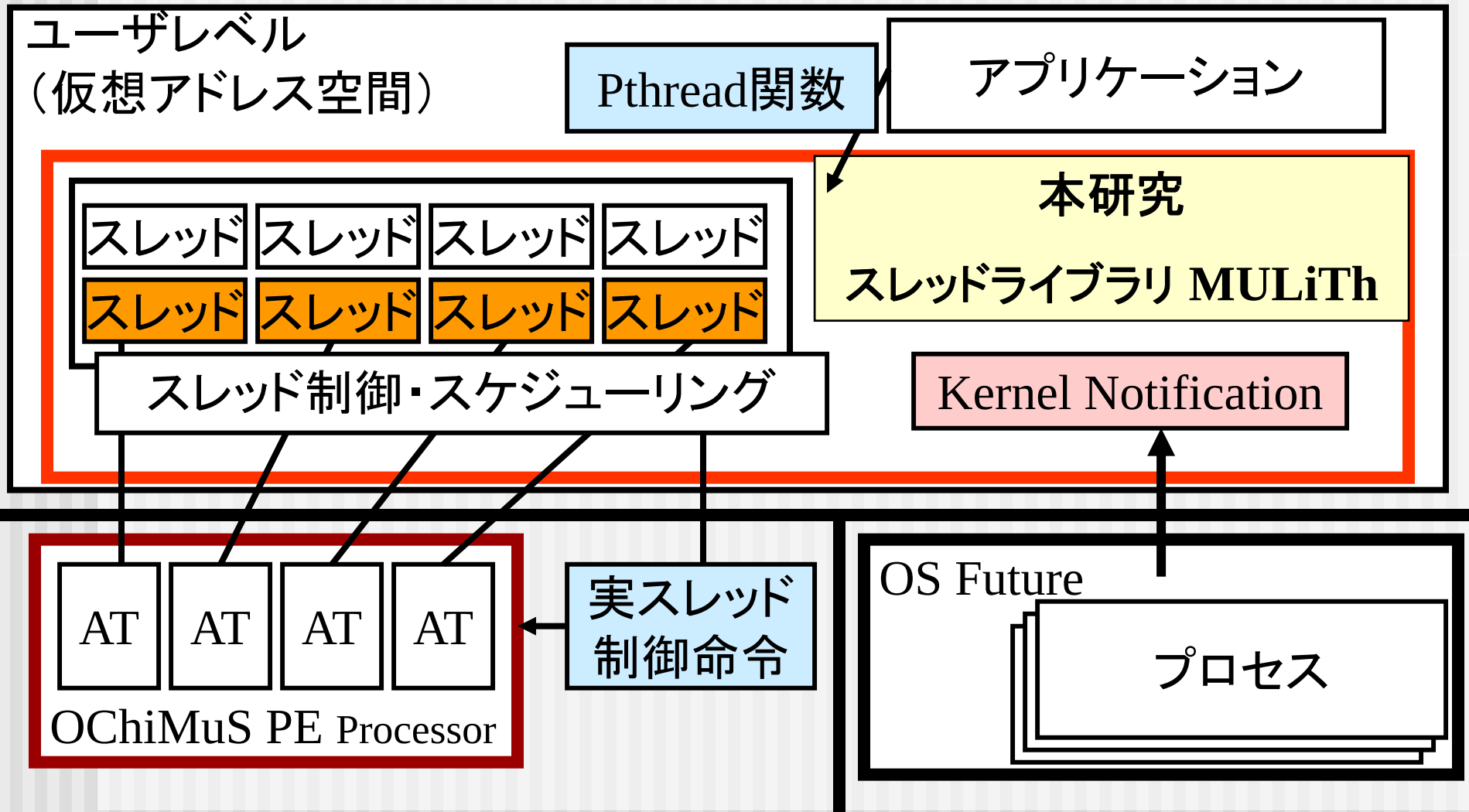
- 従来のSMP プロセッサ用システムでは不十分
  - 例：Xeon Processor + Linux or Windows
  - 実スレッド制御はOSのみ（システムコールが必要）
  - ワーキングセットの増大（複数プロセス同時実行）
  - 計算資源の共有と競合についての考慮無し
    - 演算器 / キャッシュメモリなど
  - 従来のOSからの事象通知機構（Scheduler Activations[Anderson’92]など) は非効率

目標とするシステム

- OChiMuS Project



CPU・OS・ライブラリの全体像



OChiMuS PE(processor)

- SMTプロセッサ
  - 複数の実スレッドを並列実行可能
- MIPS命令セット
- LTNによる実スレッドの抽象化
- 全ての実スレッドは同一アドレス空間

OChiMuS: On Chip Multi SMT Processor
PE: Processor Element

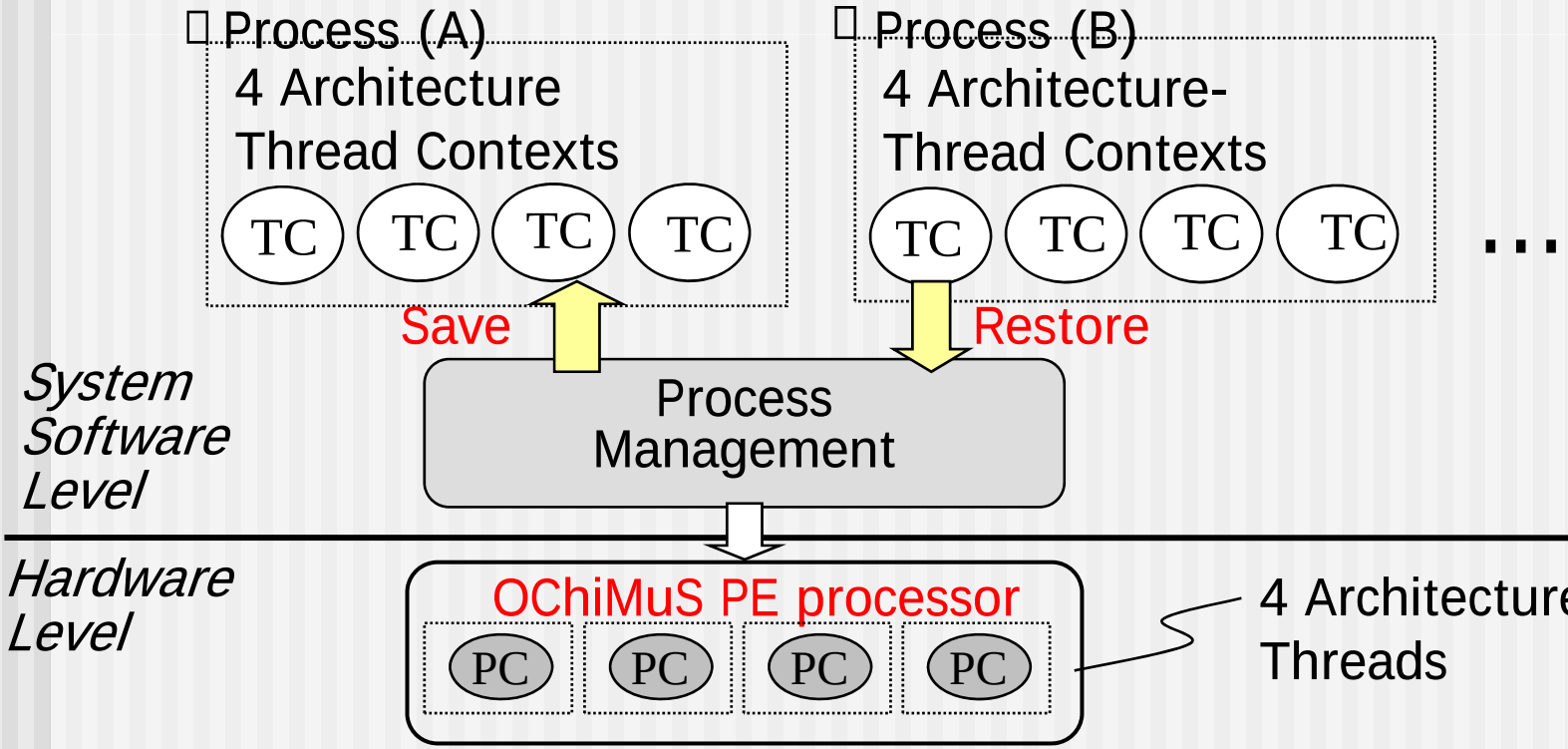
河原章二, 佐藤未来子, 並木美太郎, 中條拓伯:
システムソフトウェアとの協調を目指す
オンチップマルチスレッドアーキテクチャの構想,
コンピュータシステムシンポジウム, Vol.~2002, No.~18, pp. 1-8 (2002).

OChiMuS PEの実スレッド

- 各実スレッドのハードウェアリソース
  - プログラムカウンタ、汎用レジスタなど
  - LTN(Logical Thread Number)レジスタ
- 実スレッド制御命令
  - スレッド制御命令は*ユーザ命令*
  - 実スレッド割り当て命令で*LTNを設定*
  - 以降、*LTNで制御対象の実スレッドを指定*
- 実スレッドの状態
  - 停止状態・一時停止状態・通常状態

システムソフトウェア
OS Future

- プロセス管理
  - 複数ある実スレッドコンテキストの退避・復帰
  - 実スレッド管理はスレッドライブラリが担当



システムソフトウェア
スレッドライブラリMULiTh

- SMTプロセッサ実スレッドにスレッド割り当て
  - 複数の実スレッド上でスレッドを並列実行
  - ユーザレベルで実スレッド制御命令を利用した軽量なスレッド制御・排他制御・同期
- OSとの連携（Kernel Notification）
  - OS Future からの事象通知
- 標準的なPOSIX Thread 仕様

MULiTh: Userlevel Thread Library for Multithreaded architecture

MULiThにおけるスレッドの管理

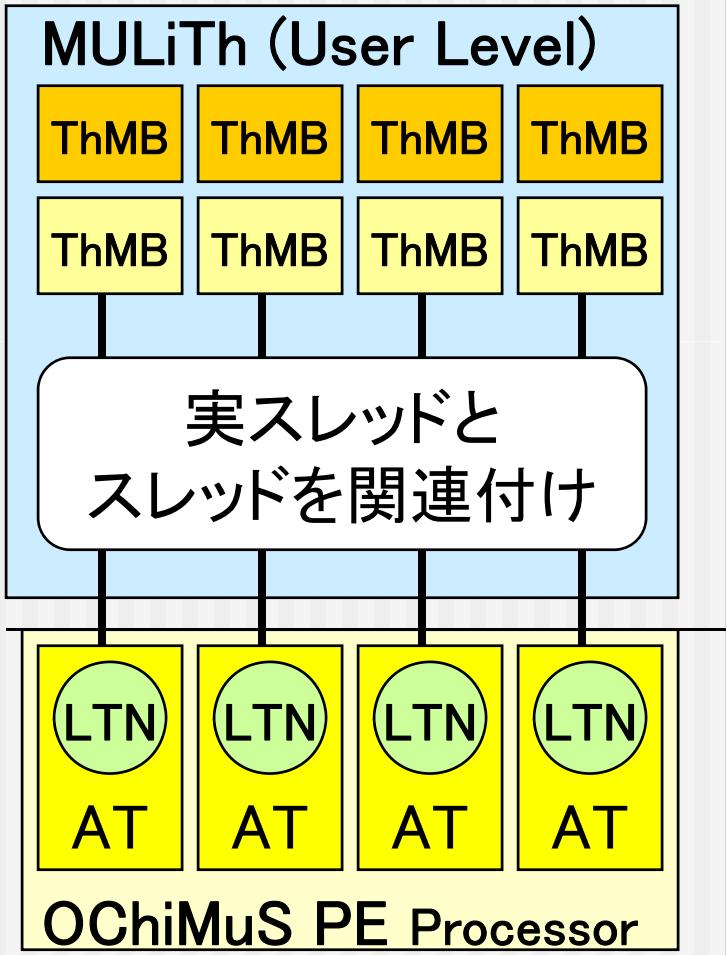
- スレッド管理ブロック(ThMB)
  - 各スレッドごとの情報を保持
  - コンテキスト・属性
- スレッド識別子

  ⇒ ThMBの先頭アドレス

  ↓

  LTNとして使用
- 実行中スレッドは

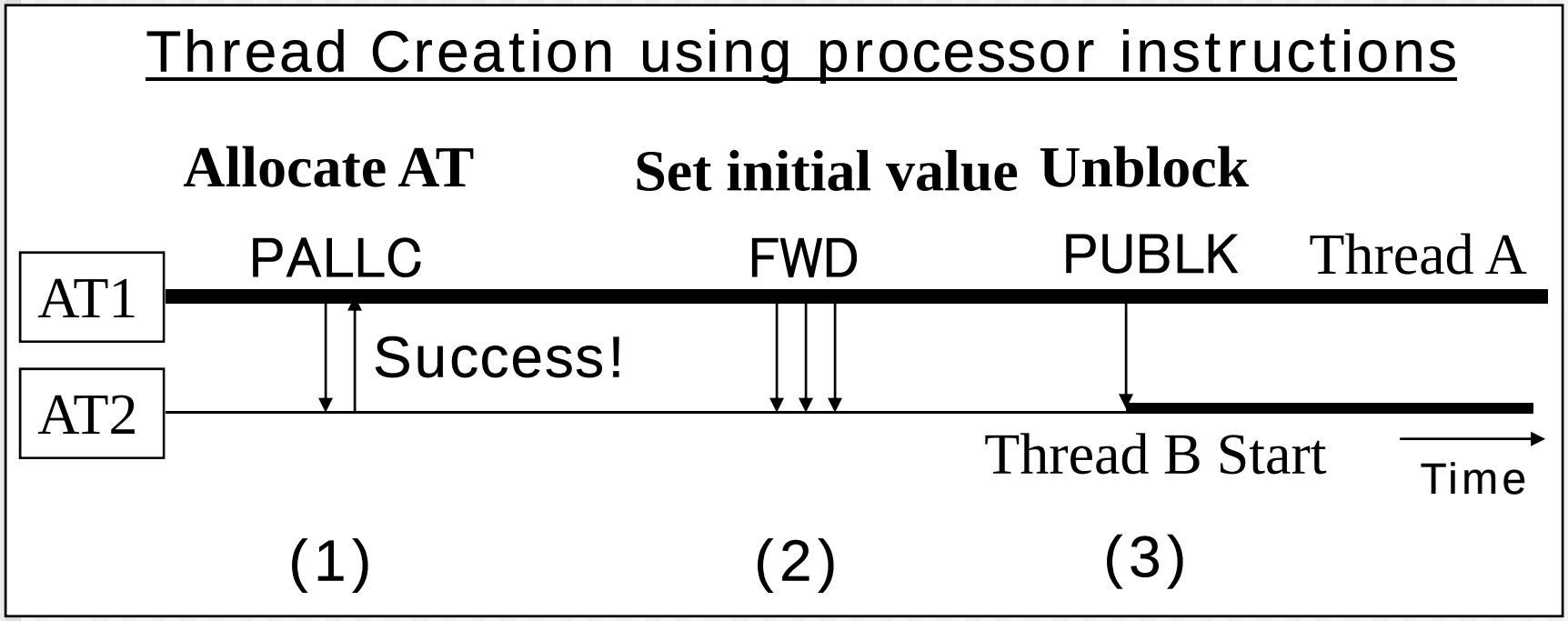
  プロセッサが把握



スレッドの制御(生成・同期)

- スレッド生成は実スレッド生成命令を利用
  - 並列実行する実スレッドを作成
  - *スレッドの仮想化コストを削減し高速化*
  - 空き状態の実スレッドがなければ待ちスレッドに
- 排他制御・同期は実スレッドを一時停止
  - OChiMuS PE のPBLK/PUBLK 命令を利用
  - スピンロック・スレッド切り替えが不要
  - *メモリアクセスを削減し性能向上*

スレッド生成



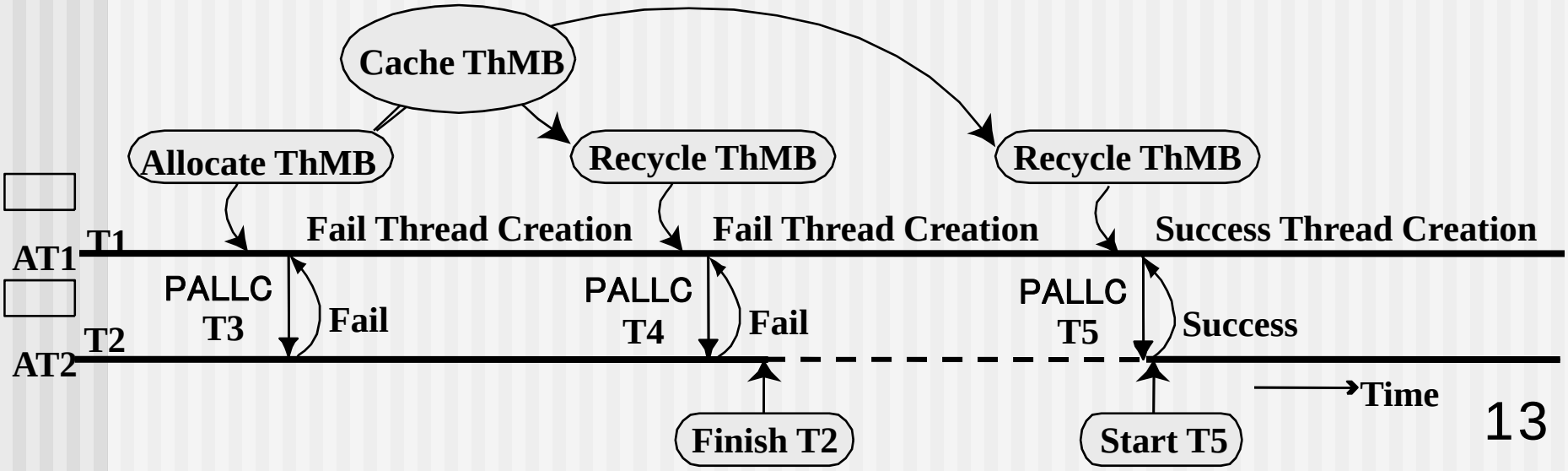
(1) 空き実スレッドがあるか？

(2) スレッド開始時の初期値設定
　　レジスタ転送命令を利用

(3) スレッド開始

細粒度スレッド生成サポート

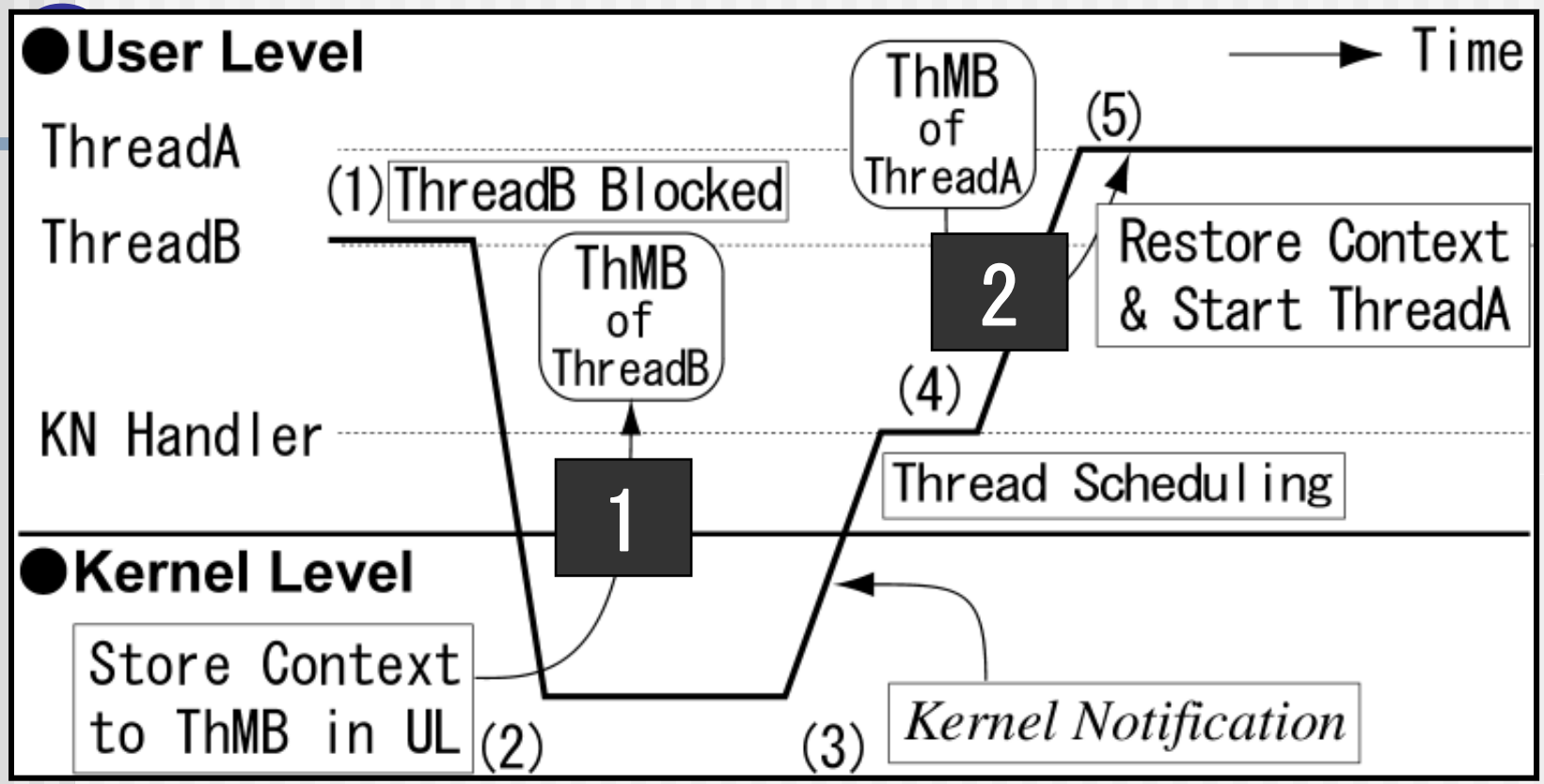
- 空き実スレッド無し → Creator に通知
  - 並列度向上ができないときの処理速度向上
- ThMB確保処理は同期が必要（重い処理）
  → ThMB 領域をキャッシュして後で利用



Kernel Notification(KN)

- OSからライブラリへの事象通知の必要性
  - I/O ブロック・ブロッキングの解除・シグナルなど
  - 複数回のコンテキストコピーなどがオーバーヘッド
- Kernel Notification機構による事象通知
  - ①カーネル遷移時、コンテキストをThMB に退避
    - ThMBのアドレスは 実スレッドLTN にあるため
  - ②ユーザレベルへの復帰時、ハンドラを起動

Kernel Notification による事象通知



- OSからの効率的な事象通知を実現
  - コンテキストのコピー回数が少ない
  - この機構でスレッドのプリエンプションを実現可

スレッドライブラリの実装と評価

- 実装
  - ライブラリはC言語 10ファイル/ 2500行
  - MIPS アセンブラでの記述が約40個所
    - プロセッサ実スレッド制御命令
    - コンテキスト復帰、退避
- 評価はシミュレータ上で実施
  - MUTHASI(MultiTHread Architecture Simulater)
  - OSは評価に利用する部分のみ実装

評価：スレッド制御の性能

単位:サイクル数

	本研究	従来	速度比
スレッド生成	84	135*	1.6倍
		10K**	120倍
		1.4K***	16.7倍
排他制御	41461	46656	1.3倍
OSからの通知	373	522	1.4倍

(*) 空き実スレッドがなかった場合

(**) Linux Thread (***)NTPL（Linux2.6 スレッドライブラリ)

評価：スレッド生成の性能（細粒度スレッド生成の評価）

単位：命令数

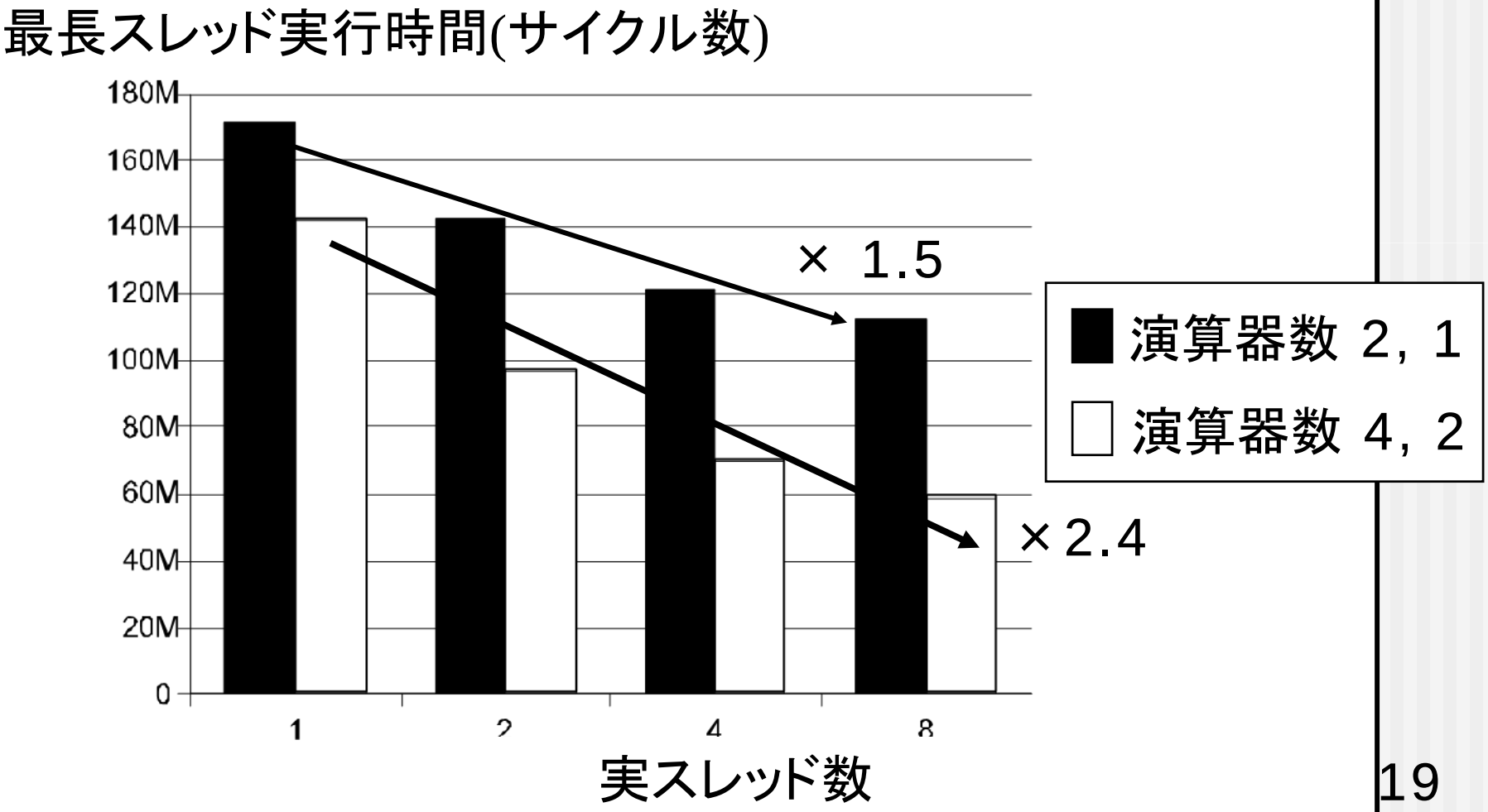
空き実スレッドがない場合のスレッド生成コスト比較	
待ちスレッドに登録	102
待ちスレッドに登録せず、Creator に失敗を通知	62
＋ThMB をキャッシュ	26

- ○：空き実スレッドがない場合のスレッド生成コストの大幅な削減
- ×：スレッドスケジューリングの責任をCreatorに

→大量細粒度スレッド生成プログラムでは問題無し

評価：アプリケーションの性能

並列化した画像縮小プログラム



本研究の成果

- マルチスレッドアーキテクチャにおけるシステムソフトウェアを考察
- スレッドライブラリ MULiTh の開発
  - 並列実行による性能向上を確認
  - 軽量なスレッド制御による性能向上
  - 細粒度スレッド生成サポート
  - 効率的なOSからの事象通知
  - Pthread仕様スレッドライブラリ

今後の課題

- OSとの連携を行うソフトウェアでの評価
  - システムコールや割り込みなど
- 適切なスケジューラの検討
  - マルチスレッドアーキテクチャの特性を利用したスレッドスケジューラが必要
- マルチスレッドアーキテクチャに適した言語処理系の検討
  - 並列化/最適化コンパイラ
  - インタプリタ

対外発表

- マルチスレッドアーキテクチャにおけるスレッドライブラリの実現と評価
  - 情報処理学会論文誌 ACS(2003. Aug)
  - SACSIS(2003. May)　優秀学生論文賞受賞 (Symposium on Advanced Computing Systems and Infrastructures 旧称JSPP)
  - PDPTA(2003. Jun) The 2003 International Conference on Parallel and Distributed Processing Techniques and Applications
- Ruby による JavaVM の実装
  - 情報処理学会第65回全国大会(2003 Mar)
- Ravaで見る Java仮想マシンのしくみ
  - JAVA PRESS Vol.31

- 以上

問題点：ユーザレベルライブラリ

- カーネルの事象をユーザライブラリへ伝達
  - I/O ブロック・ブロッキングの解除・シグナルなど
    - Scheduler Activations（’92）：カーネルが事象通知のためにユーザスレッドスケジューラを起動
    - 猪原ら(’95)：スレッド切り替え動作を最適化
  - 複数回のコンテキストコピーなどのオーバーヘッド
- 排他制御・同期機構
  - SMTなどではスピンロックが高負荷
- ライブラリインターフェース
  - 使いやすさ・過去の資産

評価：細粒度スレッド生成の性能

スレッド生成コスト比較（サイクル数）	
通常	102
失敗を知らせ	62
ThMB をキャッシュ	26

評価：細粒度スレッド生成の性能

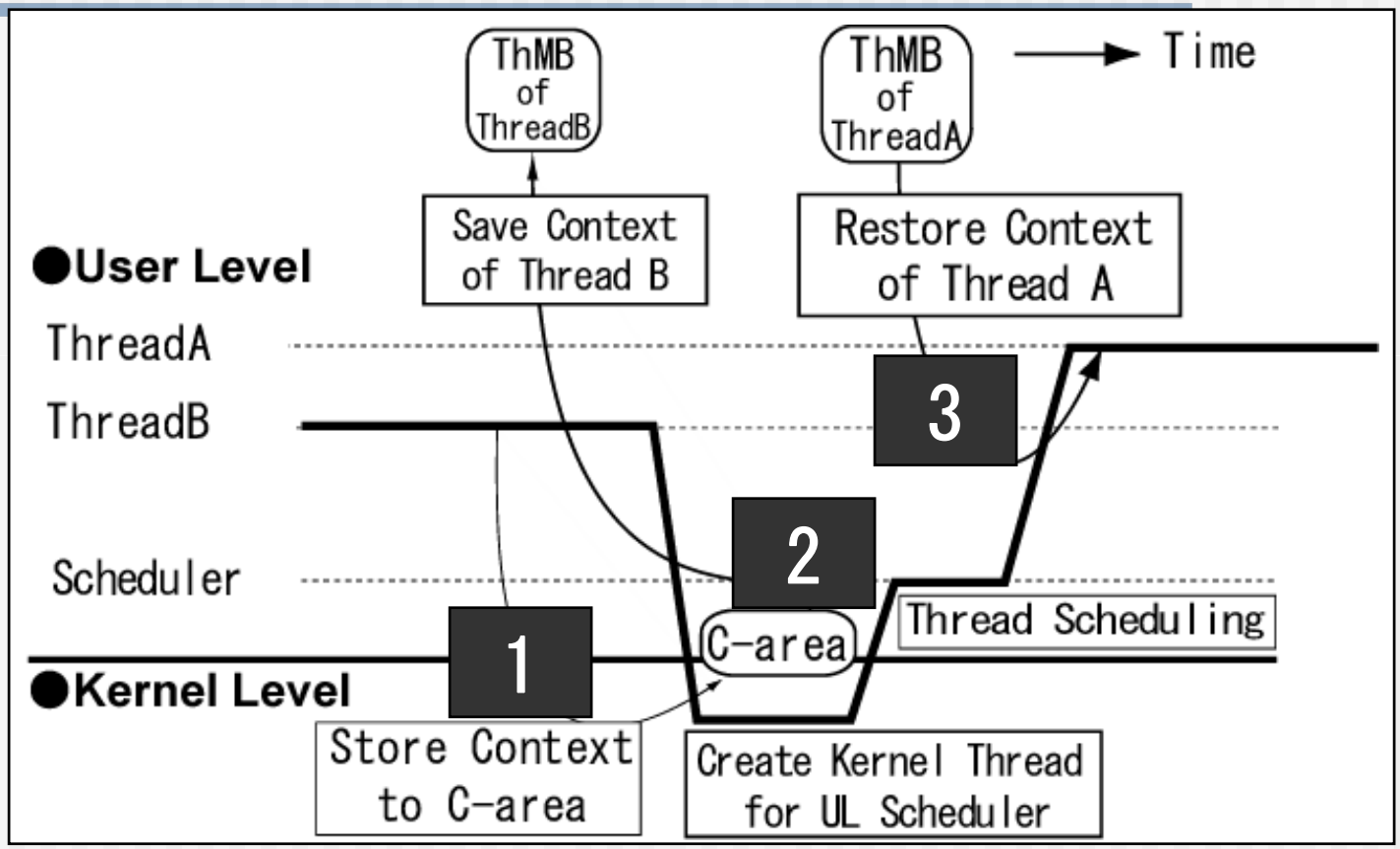
N番目のフィボナッチ数を求めるプログラムの速度向上率

実行方法	速度向上率
逐次実行	1.00
通常のMULiTh	0.25
細粒度スレッド生成を利用	1.24

フィボナッチ数を求めるプログラム

```
void *fib_th(void *p){
  return (void *)fib((int)p);}
int fib(int n){
  if(n <= 2) return 1;
  else{
    pthread_t t1;
    int a1 = 0, a2 = 0, err;
    err = th_create_fg(&t1, 0,
                       fib_th, (void*)n-1);
    if(err == E_AGAIN) a1 = fib(n-1);
    a2 = fib(n-2);
    if(a1 == 0)
  pthread_join(t1,(void*)&a1);
    return a1+a2;}}
```

従来型事象通知



3回のコンテキストコピー ＋ Kernel Thread 生成

猪原茂和, 益田隆司:情報処理学会論文誌, Vol.~36, No.~10, pp.¥ 2498-2510 (1995).
ユーザとカーネルの非同期的な協調機構によるスレッド切り替え動作の最適化

考察

- スレッド制御が軽量
  - ユーザレベルでのスレッド操作
  - プロセッサ制御命令を利用
  - 実スレッドブロック状態を利用
- 効率的なOSからの事象通知
  - 余計なコンテキストのコピーが無い
- 並列実行により性能向上
  - 実スレッドにスレッドを割り当て並列実行
  - CPUリソースの利用率が向上
  - 既存の Pthread アプリケーションが実行可能

実装した主なPthread関数

- pthread_create スレッド生成
- pthread_exit スレッド終了
- pthread_join スレッド合流（同期）
- pthread_mutex_lock / unlock 排他制御
- pthread_cond_wait / signal 同期機構

評価環境詳細

- Simple ALU：2 個
  - 一回の演算は1サイクル
- Complex ALU：1個
  - 掛け算 12サイクル
  - 割り算 32サイクル
- キャッシュ・TLBはなし

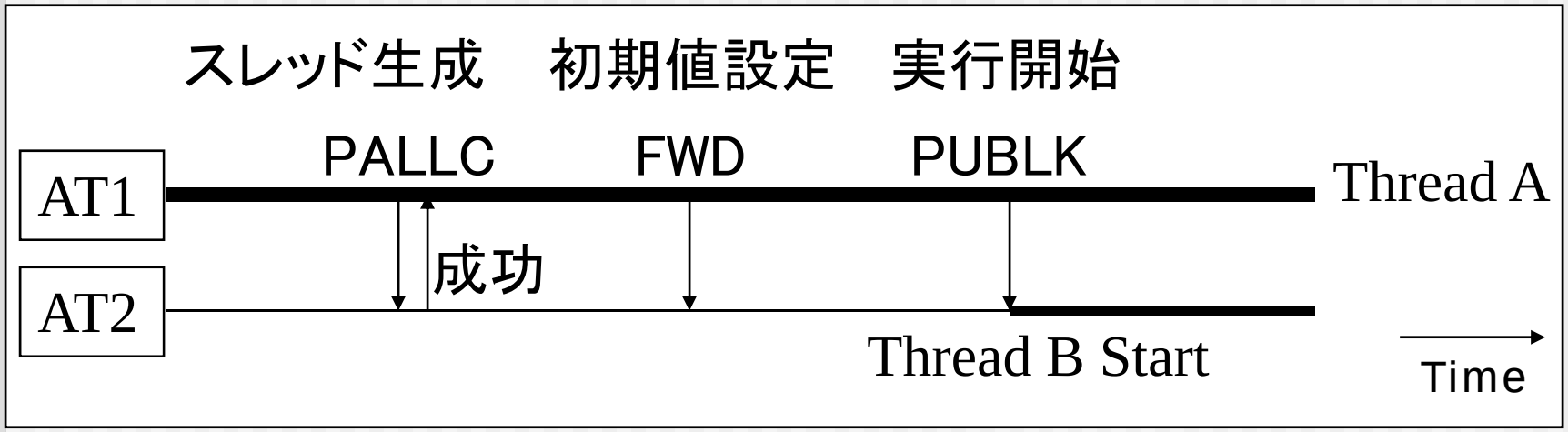
スレッドの作成

- プロセッサの実スレッド制御命令を利用
  - 並列実行するスレッドを生成
  - メモリアクセスなしでスレッド制御可
  - 命令が失敗したとき、従来のスレッド制御

  →軽量なスレッド生成

スレッドの作成

- プロセッサのスレッド制御命令(PALLC)を発行
  - 成功: PCS_HALT 状態の実スレッドが存在
    初期値を設定し、即座に並列実行
  - 制御命令: 空き状態の実スレッドなし
    従来どおり、スレッドを待ち状態へ遷移



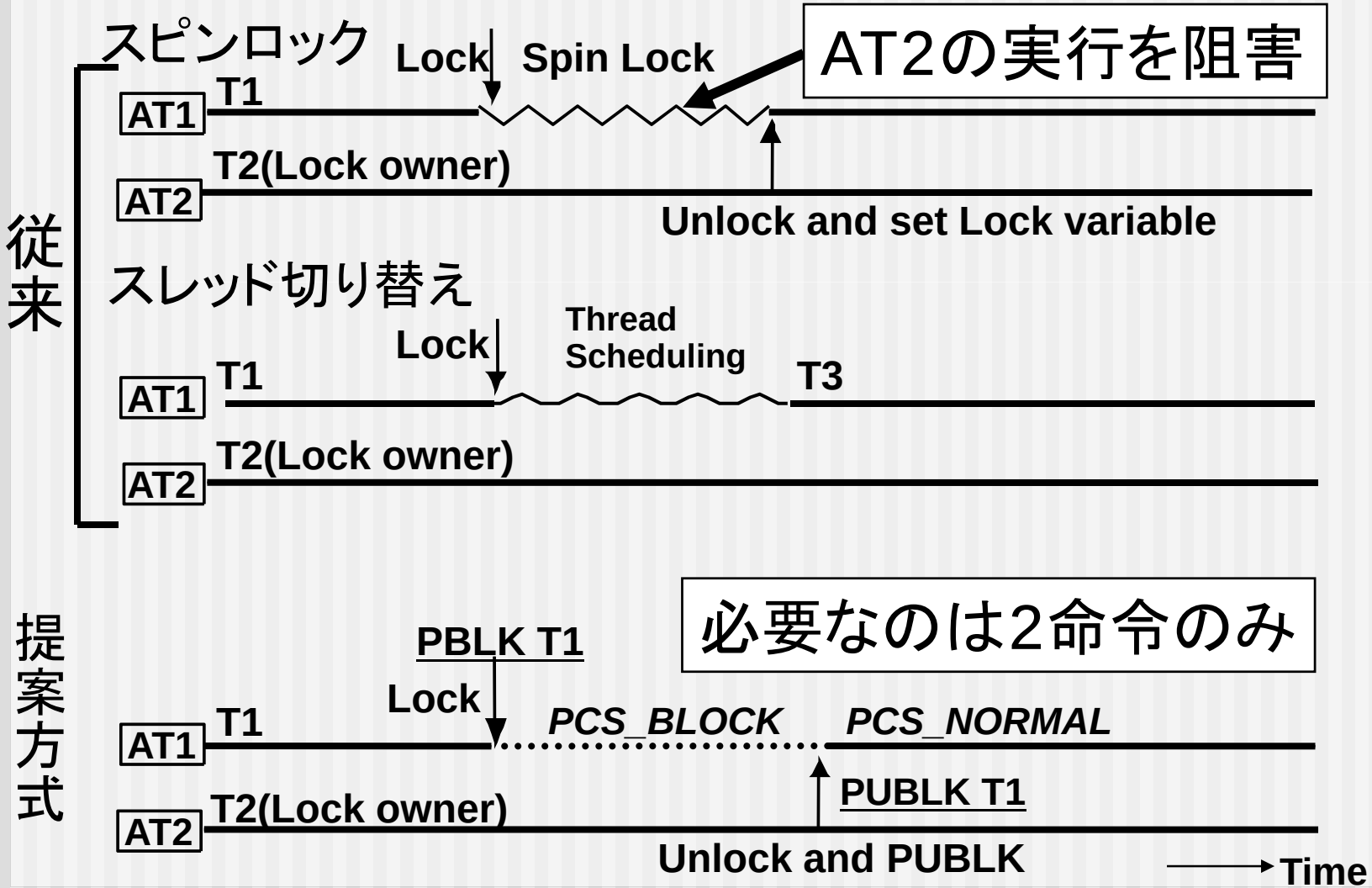
スレッドの作成

pallc dr,sr0,sr1 // 実スレッド生成命令

dr : 返り値を格納するレジスタ
- 成功
- 失敗
  ⇒停止状態の実スレッドが無い

sr0: スレッド開始位置
sr1: 設定したいLTN

スレッドの排他制御・同期



スレッドの排他制御・同期

- 問題点
  - 実スレッドの一時停止⇒並列度低下
- 解決案
  - ディスパッチ可能なスレッドがある場合スレッド切り替え
  - アダプティブロックの検討
    - あるスレッドが実行中かどうかはプロセッササポートにより知ることができる

      （あるLTNを持つ実スレッドが存在するか、を聞く）

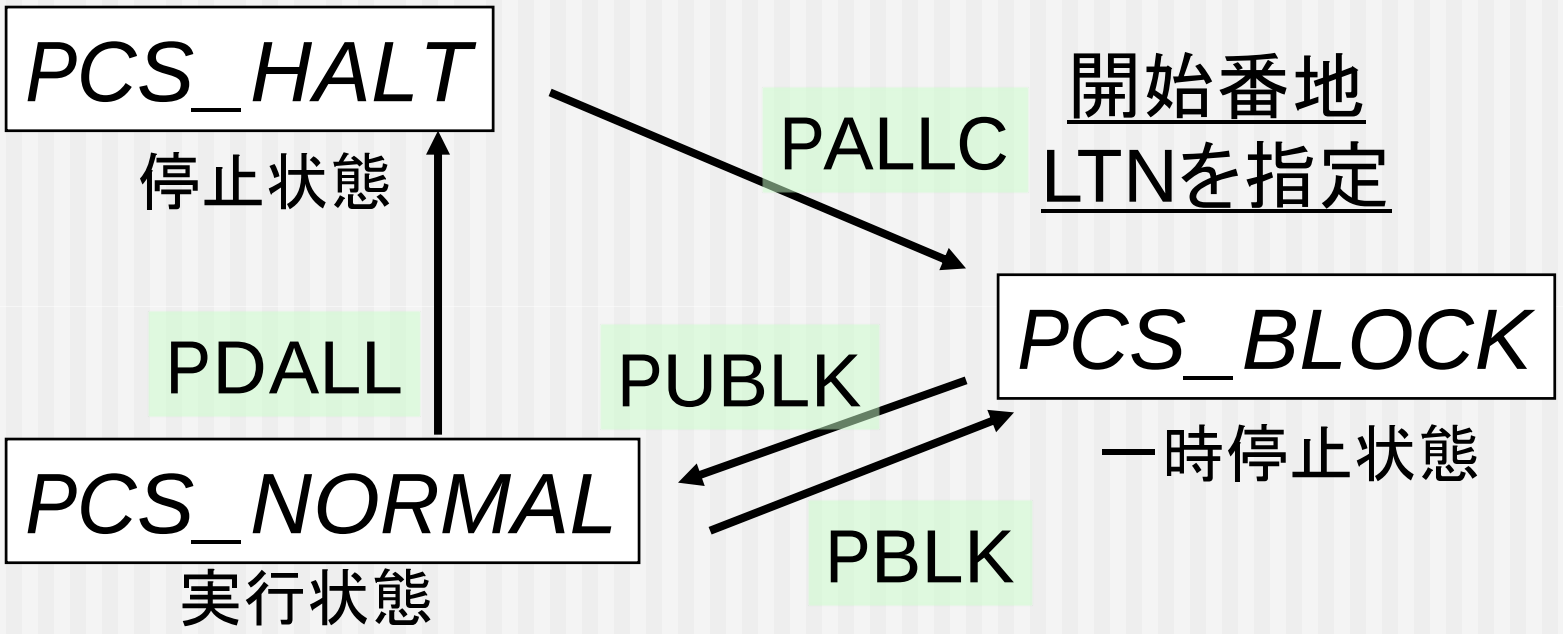
スレッドの排他制御・同期

pblk dr,sr　　// 実スレッド一時停止命令
publk dr,sr　　// 一時停止解除命令

dr : 返り値を格納するレジスタ
- 成功
- 失敗
⇒そんな LTN の実スレッドは無い
or　⇒例外状態なので実行不可

sr : 操作対象LTN

OChiMuS PE 実スレッド状態遷移



- PALLCで実スレッドに LTN を設定
- PALLC 以外の命令は LTN によってターゲットを指定
- 実スレッド間でのレジスタ転送命令 FWD がある

OChiMuS PEスレッドの状態

- 停止状態　　LTN割り当て無し
  - 割り当てを待っている状態
- 一時停止状態　　LTN割り当てあり
  - 解除すれば通常状態へ戻り実行を再開
  - プロセッサリソースを消費しない
- 通常状態　　LTN割り当てあり
  - プログラムを実行

評価：スレッドの削除・同期

	本研究	従来	速度比
スレッド削除	51	223	4.3倍
同期	202	847	4.2倍

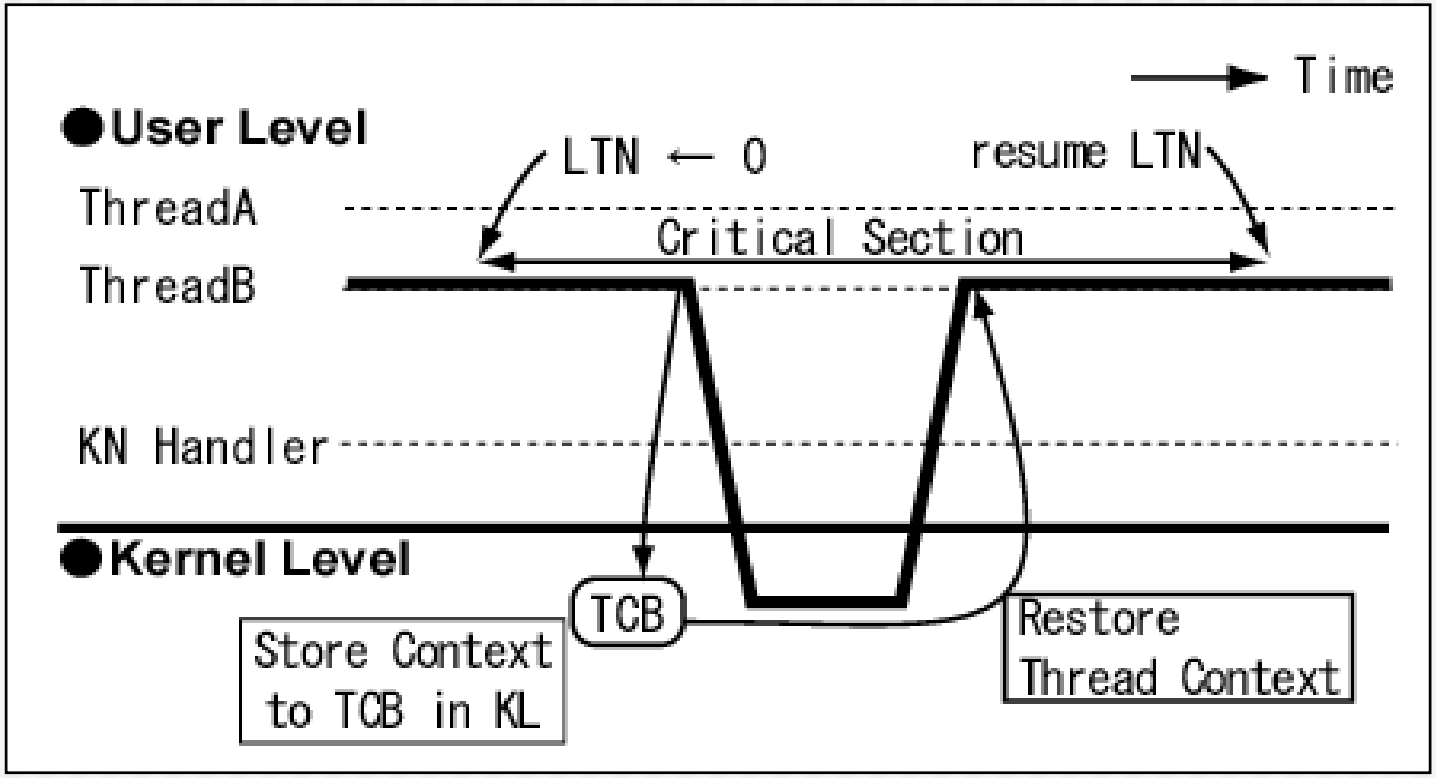
従来の1実スレッドCPUでは、
必ずスレッド切り替えが必要となる
本研究では、必ずしもそれが必要ではない

本研究の目標

- ユーザレベルでスレッド・実スレッドを管理
  - 実スレッドに割り当て、スレッドを並列実行
  - ユーザレベルで実行するので動作が高速
    - スレッド制御にシステムコール不要
    - プロセッサ実スレッド制御命令を利用
- OSからライブラリへの効率的な情報伝達
  - Scheduler Activations より効率的に
- スピンロックをしない排他制御・同期
- 一般的なスレッドライブラリインターフェース

KN:競合回避

- 実スレッドがLTN 0 である場合、カーネルは従来どおりコンテキストを復帰・退避する

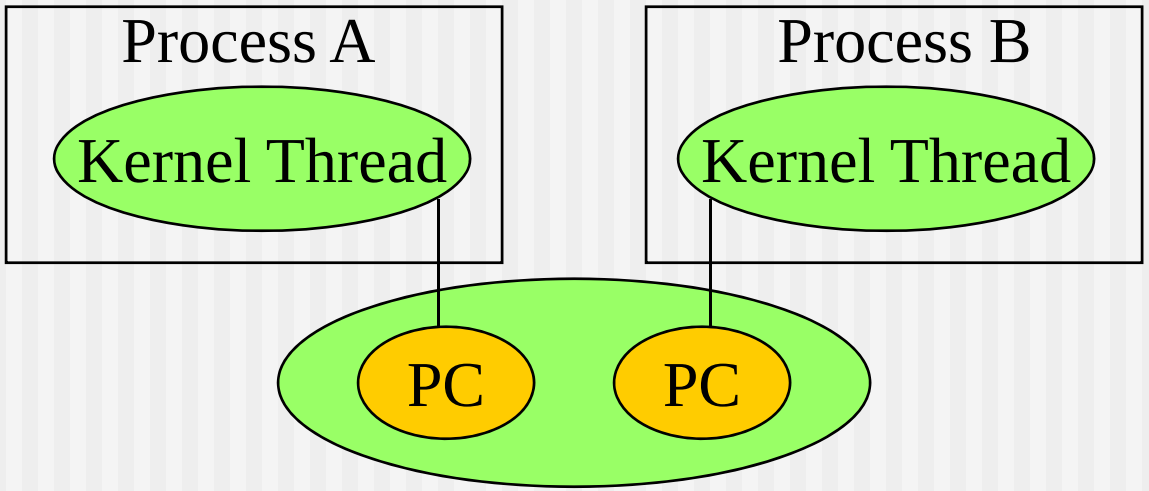


OS Future

- Future でのプロセス
  - アドレス空間管理、入出力管理など
  - 実スレッド管理はMULiThで行う
- Future でのプロセス切り替え
  - 複数の実スレッドの状態を退避・復帰を保証
  - Kernel Notification
  - 復帰は並列に実行

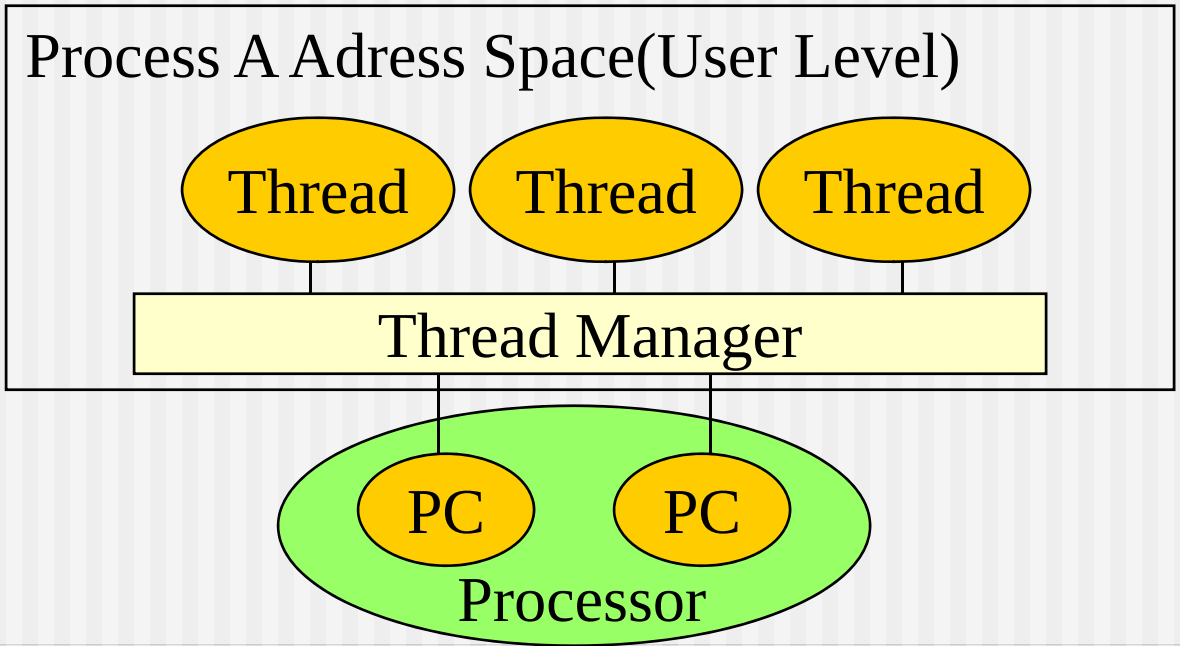
実スレッドの管理(1)

- 従来：OSがカーネルスレッドとして管理
  - 利点：SMP用カーネルが利用可能
  - 短所：ワーキングセット増大
    スレッド制御にシステムコールが必要

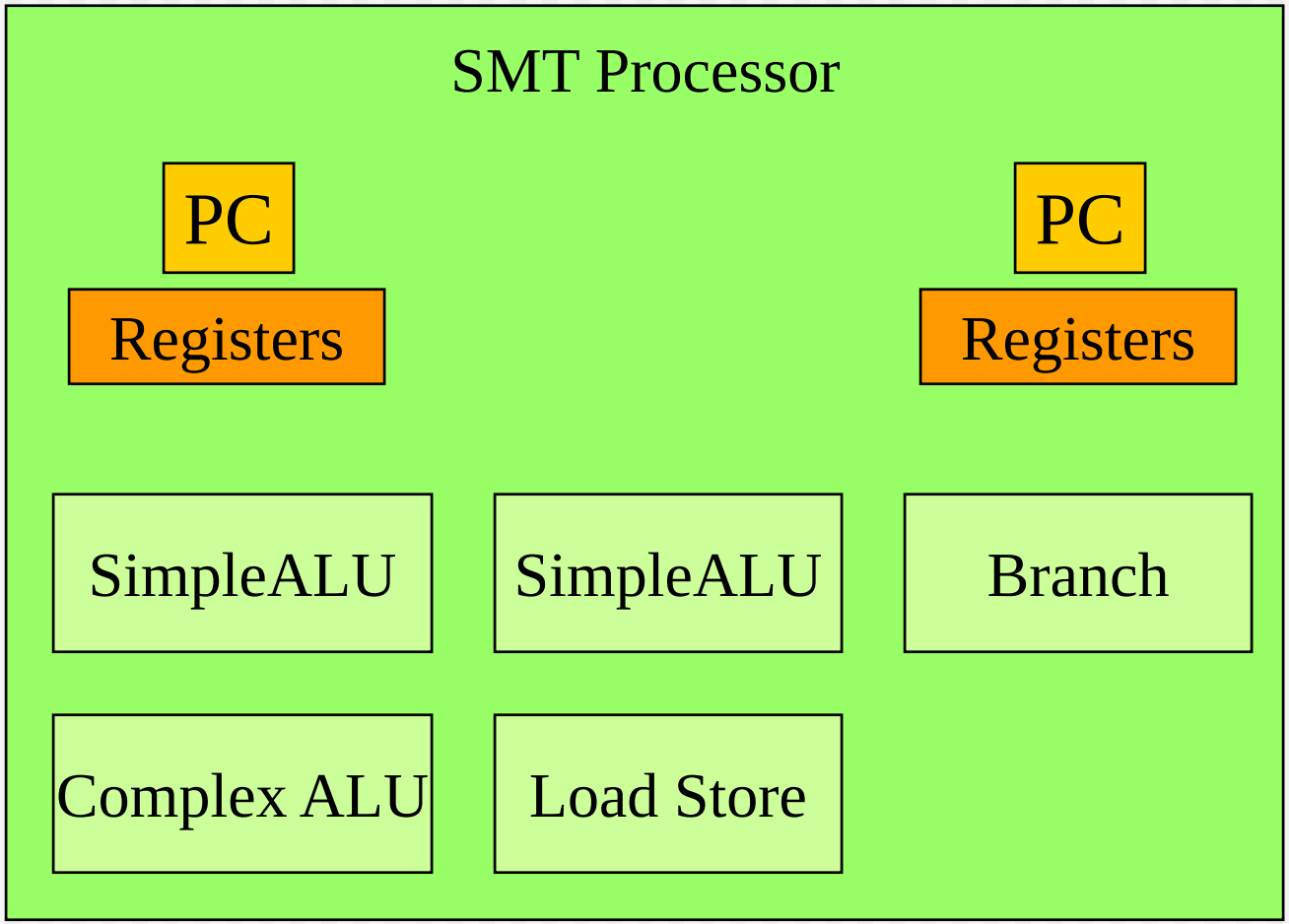


実スレッドの管理(2)

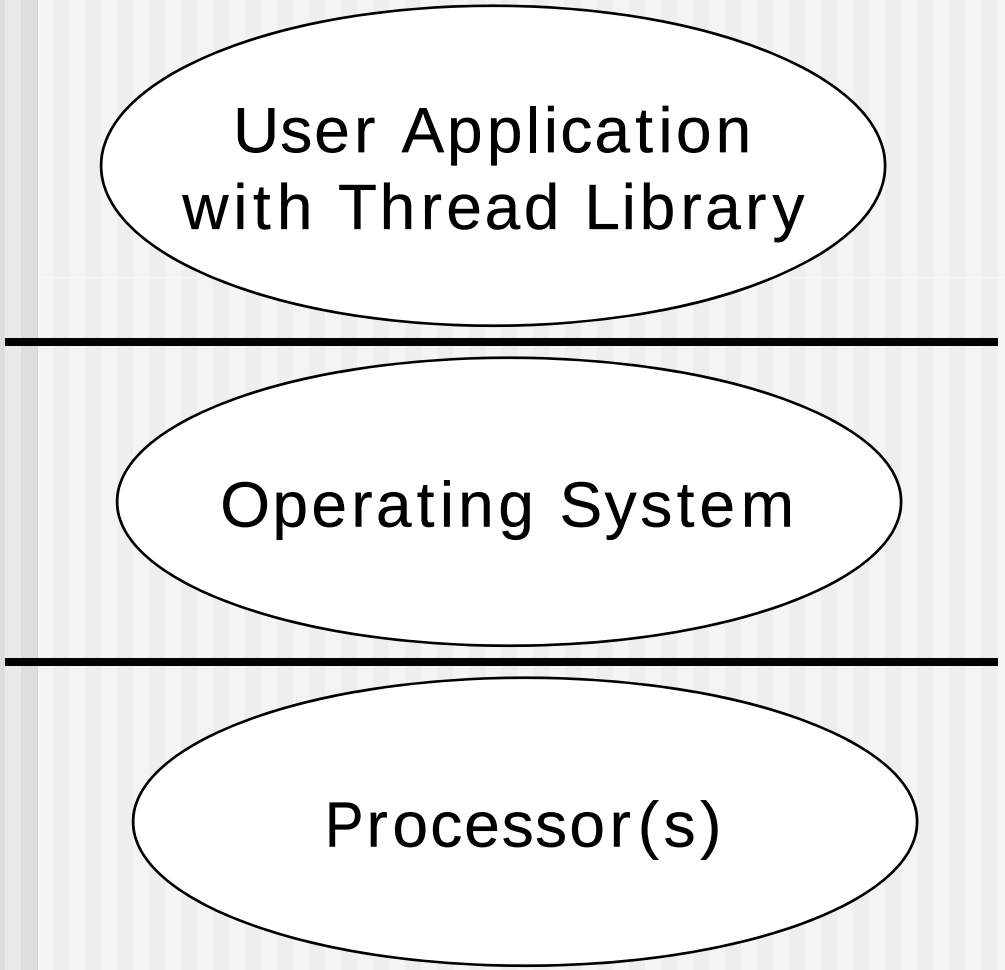
- 実スレッドをユーザレベルで管理
  - ユーザレベルで軽量なスレッド制御が可能
  - 専用システムソフトウェアが必要



SMTアーキテクチャ

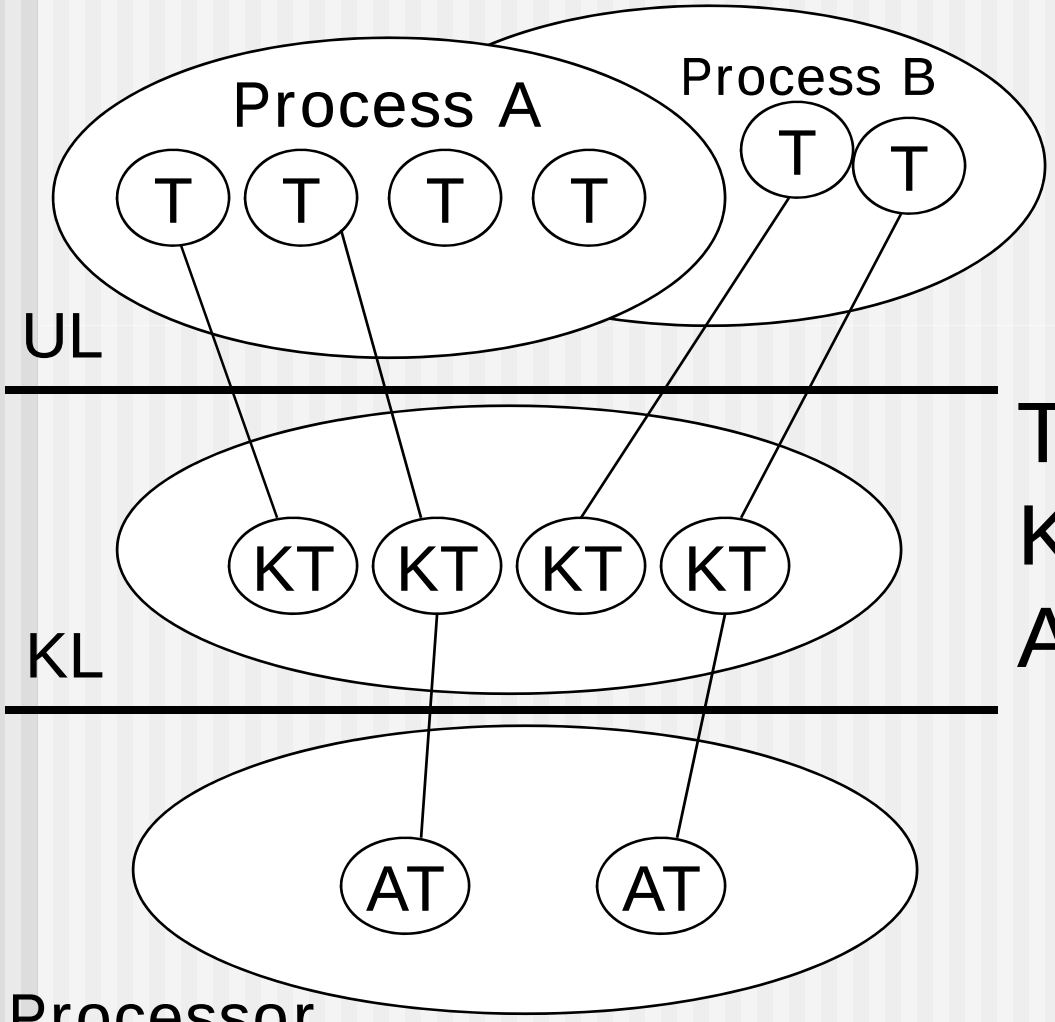

SMT Processor
PC
Registers
PC
Registers
SimpleALU
SimpleALU
Branch
Complex ALU
Load Store

従来のプロセッサ・OS・ライブラリ



ユーザライブラリは
プロセッサを
直接操作しない

従来のプロセッサ・OS・ライブラリ



T: ユーザスレッド
KT: カーネルスレッド
AT: 実スレッド