博士論文本審査発表資料

高速なRUBY用仮想マシンの開発

EFFICIENT IMPLEMENTATION OF RUBY VIRTUAL MACHINE

東京大学大学院
情報理工学系研究科　特任助教
笹田耕一

Agenda

- 背景と目的
- 高速なRuby用仮想マシン
- Rubyの並列化
- まとめ

背景

- スクリプト言語の必要性の増大
  - スクリプト言語の重要性増大
    - 計算機の性能向上，スクリプト言語の機能向上
    - 利用シーンの増加
      - GUIプログラミング，ネットワークアプリケーション
      - ウェブアプリケーション
  - システムプログラミング言語 v.s. スクリプト言語
    - John K. Ousterhout, Scripting: Higher-Level Programming for the 21st Century, Computer, IEEE, 1998 より
    - 型付・静的 v.s. 型無・動的

      部品開発　　部品結合

      高い性能　　柔軟で迅速な開発

PL/I, FORTRAN v.s. JCL on OS/360
C v.s. Shellscript, sed, awk on UNIX
C, C++ v.s. Tcl, VB on GUI
Java v.s. Perl, Javascript on Internet

よく使われている プログラミング言語比較

言語	モデル	開発効率	機能	並列実行	性能	実行方式
C	静的	×	×	○	◎+	ネイティブ
Java	静的	△	○	○	◎	VM +JIT
sh	動的	△	×	×	×	Interpreter
Perl	動的	○	○	×	△	VM
Python	動的	○	○+	×	△	VM
Ruby	動的	○	◎	×	×	Interpreter

機能：オブジェクト指向，メタプログラミング，高階関数

背景: Ruby

- オブジェクト指向スクリプト言語Ruby
  - 近年注目されているスクリプト言語
  - 動的で柔軟な言語仕様
  - 高い記述性
- Ruby処理系（CRuby）
  - C言語で記述，リファレンス実装
- その他の処理系実装も存在
  - JRuby (Java)
  - IronRuby (.NET)
  - Rubinius (Ruby)

RubyConf 参加者数
2004　60人
2005 200人
2006 300人
2007 500人
RubyKaigi 参加者数
2006 200人
2007 450人
＊両年とも数時間で5千円チケット完売

CRubyの問題点

- 性能向上が困難な処理系の構造
  - 構文木を再帰関数でたどる処理系
  - 既知の処理系最適化技術が適用不可
- Rubyの並列実行が不可能
  - コモディティ化した計算機を活用不可
  - Rubyスレッドを並列実行できないスレッド処理機構

Rubyが遅いことに関して

We would be absolutely dead on Ruby.

Ruby Performance Revisited

http://joelonsoftware.com/items/2006/09/12.html

Joel Spolsky

本研究の目的

高速なRuby処理系の実現

RubyのVM化
↓
仮想マシンYARVの開発

Rubyの並列実行
↓
Rubyスレッドを並列実行

よく使われている プログラミング言語比較

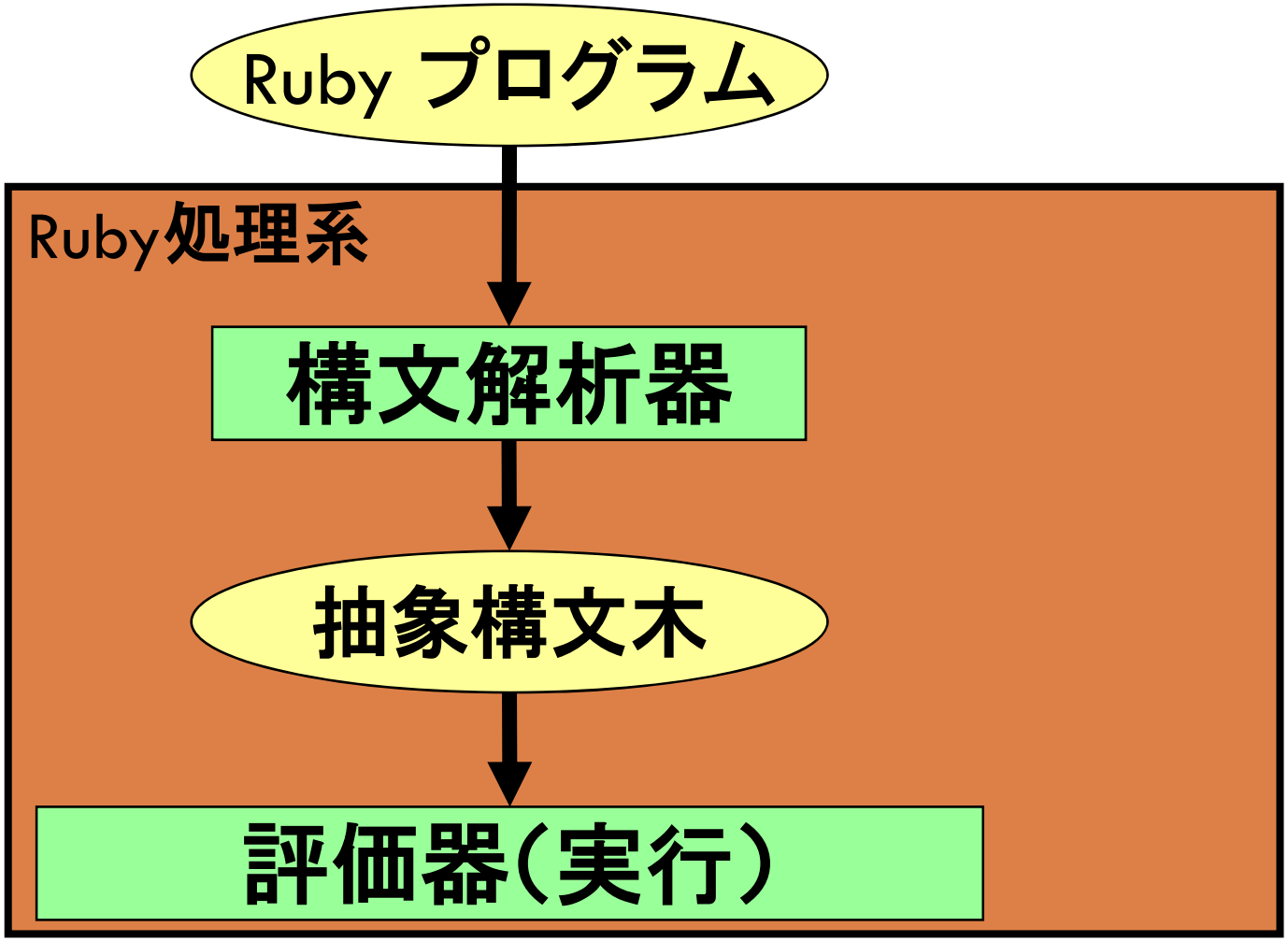
言語	モデル	開発効率	機能	並列実行	性能	実行方式
C	静的	×	×	○	◎+	ネイティブ
Java	静的	△	○	○	◎	VM +JIT
sh	動的	△	×	×	×	Interpreter
Perl	動的	○	○	×	△	VM
Python	動的	○	○+	×	△	VM
Ruby	動的	○	◎	×→○	×→○	Interpreter → VM

機能：オブジェクト指向，メタプログラミング，高階関数

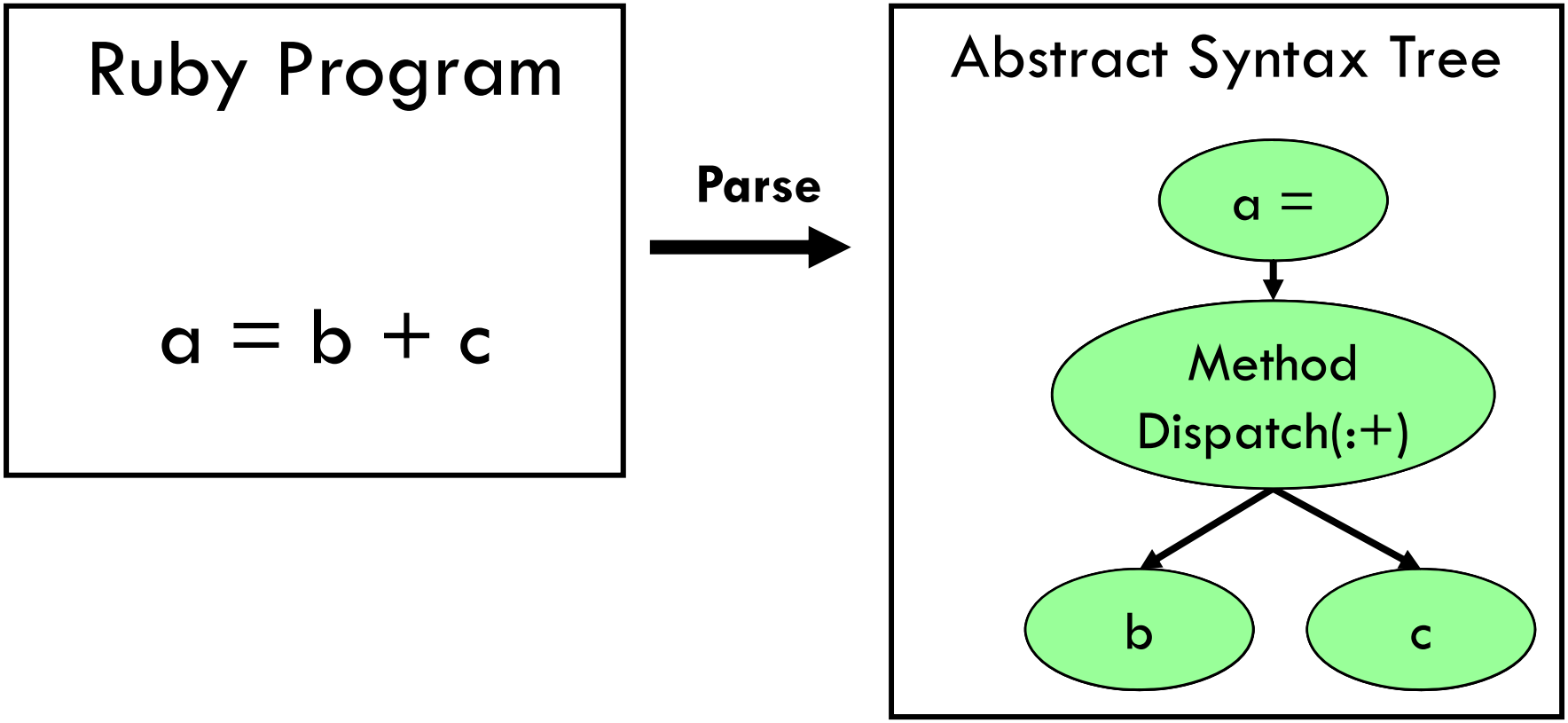
Agenda

- 背景と目的
- 高速なRuby用仮想マシン
- Rubyの並列化
- まとめ

CRubyの構成



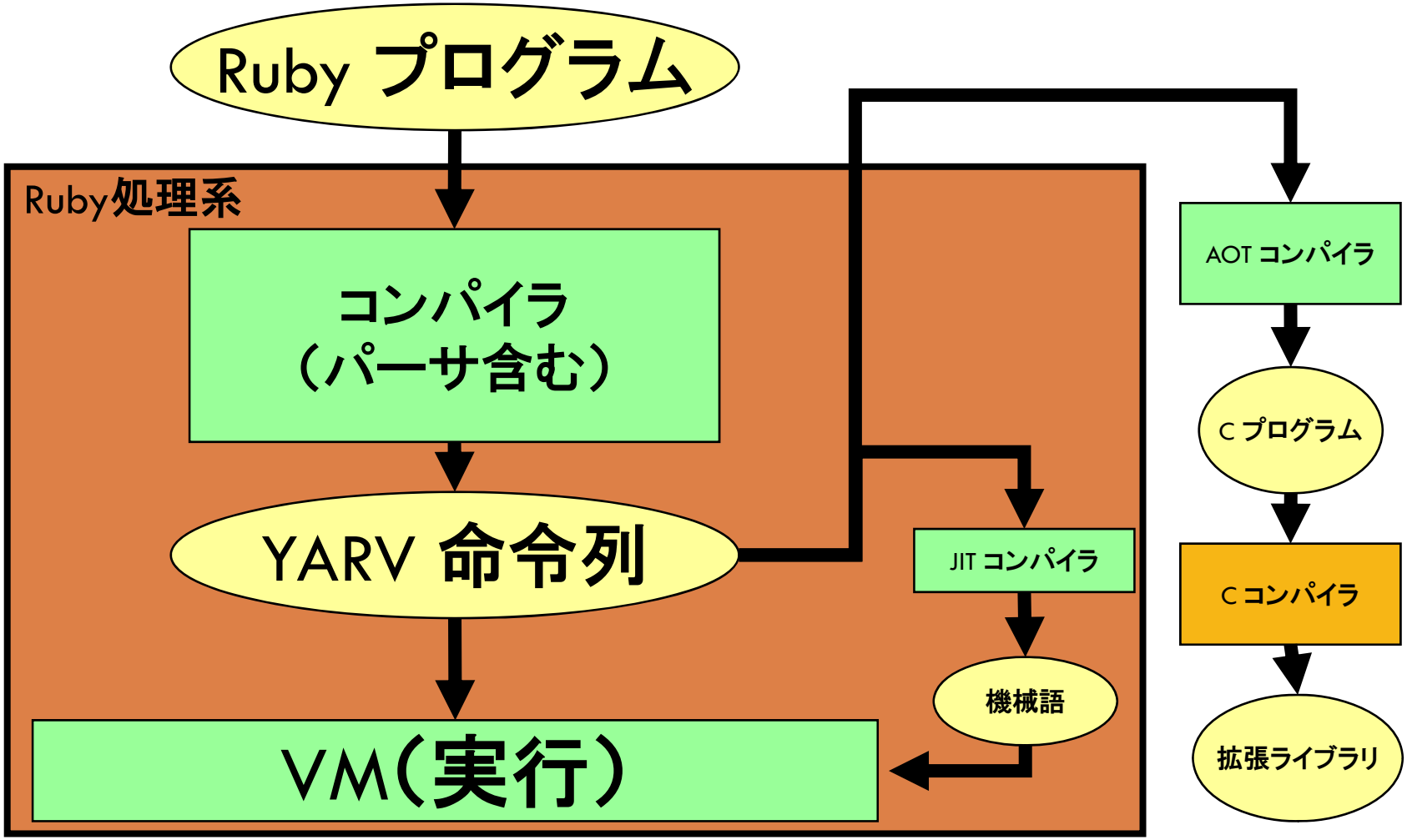
CRubyの処理


Ruby Program
a = b + c
Parse
Abstract Syntax Tree
a =
Method
Dispatch(:+)
b
c

Ruby向け仮想マシンYARV

- YARV: Yet Another RubyVM
  - Rubyプログラムを表現するYARV命令セット
  - コンパイラ
  - 仮想機械(VM)
    - スタックマシン
    - 各種最適化を適用
  - C言語で実装
    - まずは, どこでも速い, を目標に
- 簡単なVM生成系を利用した効率的な開発

YARVの構成


Ruby プログラム
Ruby処理系
コンパイラ
（パーサ含む）
YARV 命令列
VM（実行）
JIT コンパイラ
機械語
AOT コンパイラ
C プログラム
C コンパイラ
拡張ライブラリ

検討：仮想マシンの計算モデル スタックマシン v.s. レジスタマシン

- レジスタマシン型VMのほうが1.4倍高速
  - Yunhe Shi, et.al., Virtual machine showdown: stack versus registers, 2005（Java, OCaml用VMで評価）
  - レジスタマシン
    ○命令数が少ない
    →命令ディスパッチ回数減少により，VM一般で有利
  - スタックマシン（古典的）
    ○ コンパイル時間が短い，実装が容易
    インタプリタとして大事
  - Rubyはメソッド呼び出し多発
    → スタックマシンのほうが有利と判断

命令セット・コンパイラ

- YARV命令セット
  - スタックマシンモデル
  - Rubyプログラムを十分表現可能な命令セット
  - 55命令
- Rubyパーサ
  - Rubyプログラム → 抽象構文木
  - CRubyのものを利用
- コンパイラ
  - 抽象構文木 → YARV 命令列
  - ピープホール最適化
  - 最適化命令へ変換

YARVの処理

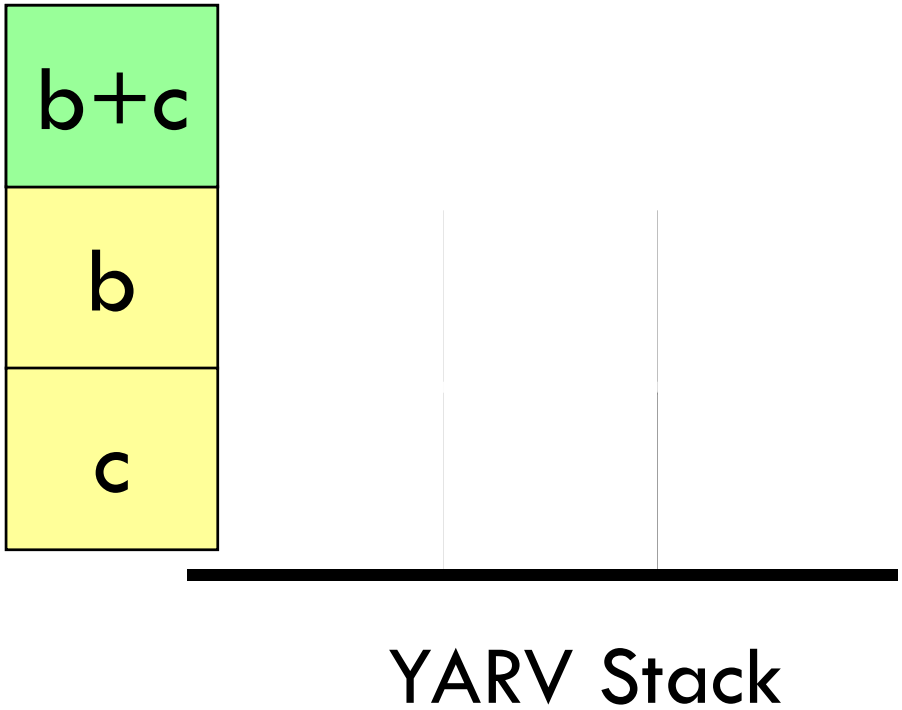
Ruby Program

a = b + c

Compile

YARV Instructions

getlocal	b
getlocal	c
send	+
setlocal	a



VM

- オブジェクト管理，GC
  - CRubyと同等
- スタックマシン
  - 仮想レジスタ：PC, SP, LFP, DFP, CFP, …
  - 値用スタックと制御フレーム用スタックの2本立て
    - 環境をヒープにコピーするClosureの実現が容易
  - スコープ変更 → 制御フレーム用スタック push
    - メソッド呼び出し，ブロック呼び出し（yield），クラス定義文
- 表引き法による例外処理
  - 例外が発生しない限り高速
  - ただし，setjmp/longjmp も利用

VM命令の実行と最適化

- 命令の実行手順
  1. 命令列から命令のフェッチ
  2. 命令へ分岐
  3. 命令列から命令オペランドのフェッチ
  4. スタックオペランドのフェッチ
  5. 命令処理の実行
  6. スタックへ結果を書き込み
  7. プログラムカウンタ(PC)を進め，1 へ戻る

ダイレクトスレッデッドコード

命令融合

スタックキャッシング

特化命令

インラインキャッシュ

ネイティブコンパイラ(開発中)
プロファイラ
コンパイル時ののぞき穴最適化

最適化：特化命令

- 問題：メソッド呼び出しのオーバヘッドが特に大きな処理
  - 「メソッド呼び出しのオーバヘッド」>>「処理のオーバヘッド」
  - Integer#+ などの数値計算
- 解決：特定のメソッド呼び出しパターンを特化命令へ置換
  - 特定 selector、特定の引数の数のメソッド呼び出しを置換
  - 例）a + b
    - send :+,1 → opt_plus
  - 型（クラス）のチェック
  - 動的特性を満たすため，再定義のチェック

Python では
チェックしない

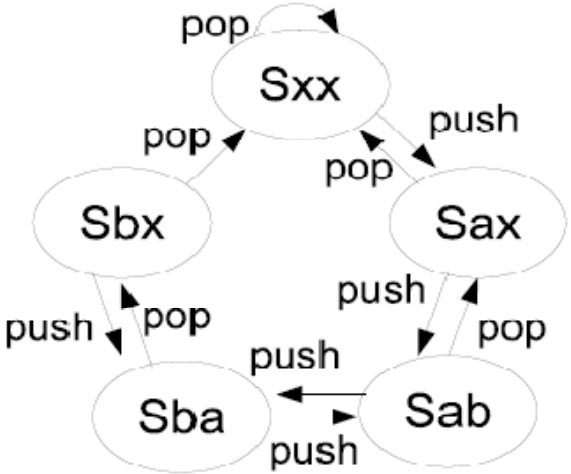
```
opt_plus:
    if(a と b は整数？)
        if 整数についての + メソッドは再定義されていないか？
            return a+b
        else return 通常のメソッド呼び出し（a.+(b)）
```

最適化：オペランド・命令融合

- 問題：メモリ上の命令列からのフェッチオーバヘッド
- 解決：融合操作
  - オペランド融合
    - insnA op1 → insnA_op1
    - op1 が頻出する場合、ひとつの命令に融合（例：putobject_true）
  - 命令融合
    - insnA insnB → insnA_insnB
    - insnA, insnB という並びが頻出する場合、ひとつの命令に融合（a.k.a superinstruction）
  - 専用命令の作成
  - 処理に埋め込まれているため，メモリ上の命令列からのフェッチオーバヘッド削減

最適化：静的スタックキャッシング

- 問題：スタック操作のオーバヘッド
- 解決：スタックキャッシングの利用
  - スタックトップレジスタ（SReg）を用意
  - スタックへプッシュする代わりにSregへアクセス
  - 5状態、2レジスタの静的スタックキャッシング
  - 5状態なので、同じ命令につき5命令必要
    - putobject（スタックにひとつ積む）
      - putobject_xx_ax
      - putobject_ax_ab
      - putobject_bx_ba
      - putobject_ab_ba
      - putobject_ba_ab



  - M. Anton Ertl , Stack caching for interpreters, PLDI, 1995

最適化：静的スタックキャッシング

Ruby プログラム

v1 = v2 + v3

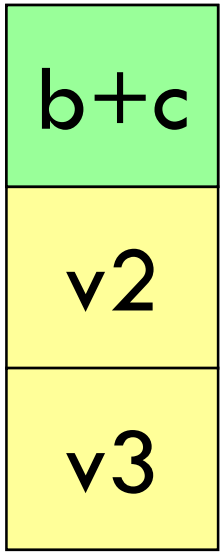
YARV命令列(SC)

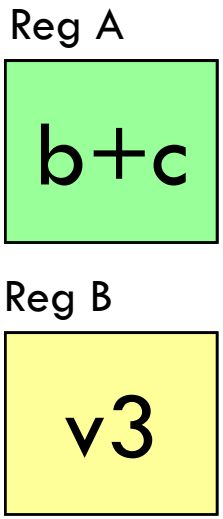
getlocal_xx_ax v2

getlocal_xx_ab

send_ab_ax +

setlocal_ax_xx v





スタック操作無し！

YARV
タックマシン

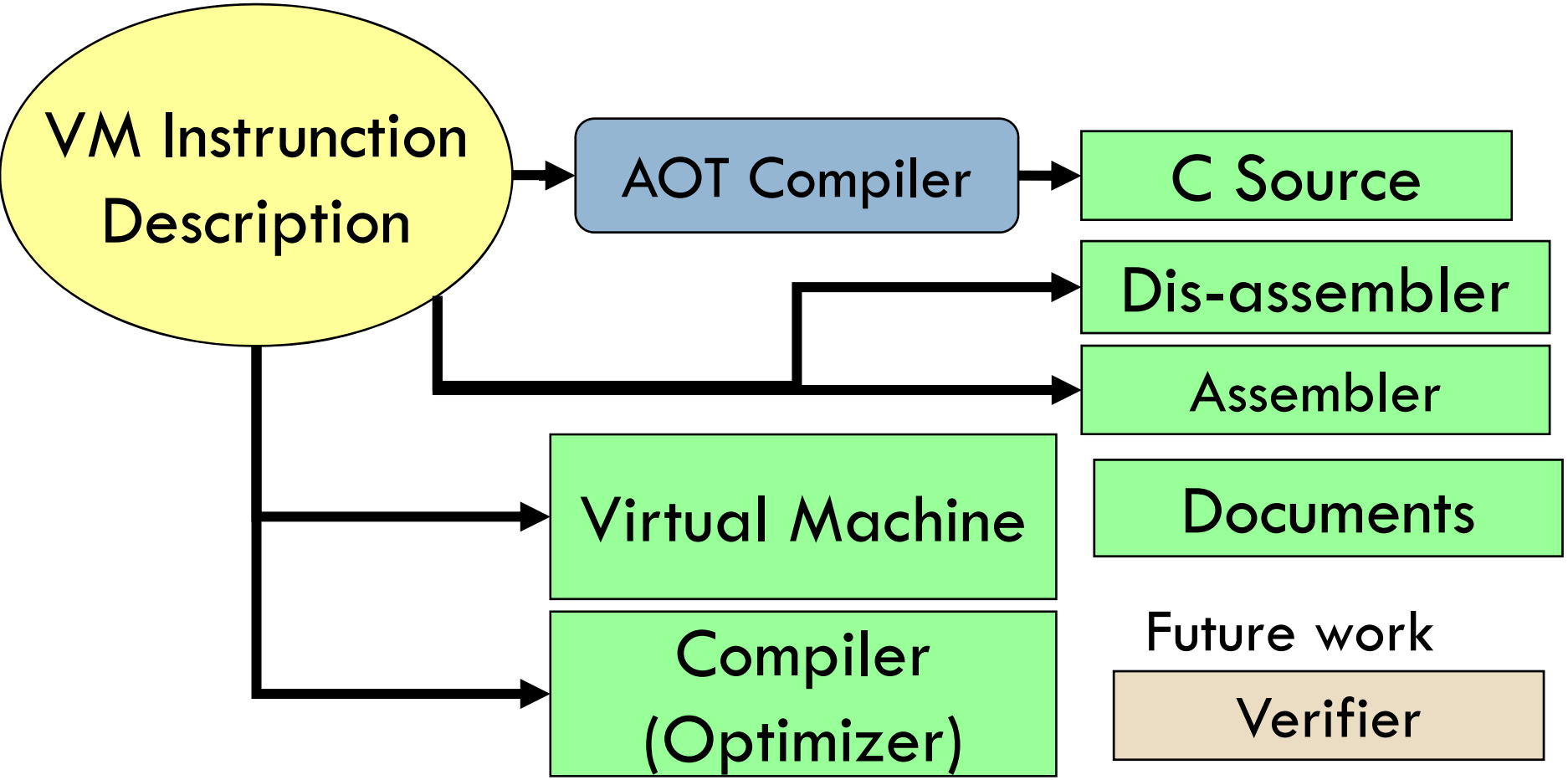
VM生成系の利用

- 仮想マシンは複雑なソフトウェア
  - 各命令ごとのスタックのpush/pop, pc 増加, etc
  - 最適化用命令が必要
    - オペランド・命令融合のための命令の用意
    - スタックキャッシュ用命令の用意（基本の5倍必要）
- VMの多くの処理は機械的に生成可能

→ VM生成系の利用

  - 関連研究
    - M. Anton Ertl et.al: vmgen: a generator of efficient virtual machine interpreters, 2002 ← ベース．命令オペランドの融合には対応しない
    - 内山等: 仮想機械の仕様記述に基づくバイトコードインタプリタ生成系, 2005 ← 厳密な解析を行うため，C言語呼び出しに未対応

VM生成系


VM Instrunction Description
AOT Compiler
C Source
Dis-assembler
Assembler
Virtual Machine
Documents
Compiler (Optimizer)
Future work
Verifier

VM生成系：命令記述と生成例

- 値の宣言と、ロジックの記述（C言語）

（1）命令オペランド（2）スタックオペランド→前処理

（3）スタックへプッシュする値→後処理

```
DEFINE_INSN
mult_plusConst
(int c)          //  (1) 命令オペランド定義
(int x, int y) // (2) スタックオペランド定義
(int ans)       // (3) 命令の返り値定義
{
  // C言語による命令ロジック記述部分
  ans = x * y + c;
}
```

→

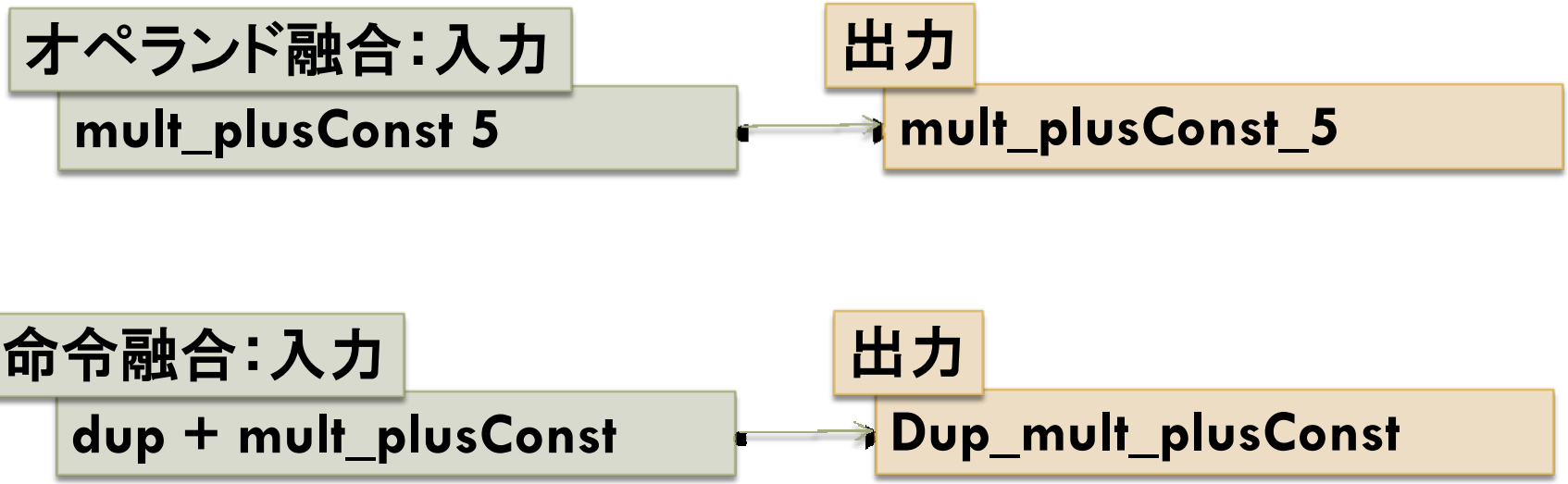
```
mult_plusConst:{
  int c = *(PC+1);
  int y = *(SP-1);
  int x = *(SP-2);
  int ans;
  PC += 2;
  SP -= 2;
{
  ans = x * y + c;
}
  *(SP) = ans; SP += 1;
  goto **PC;}
```

前処理

後処理

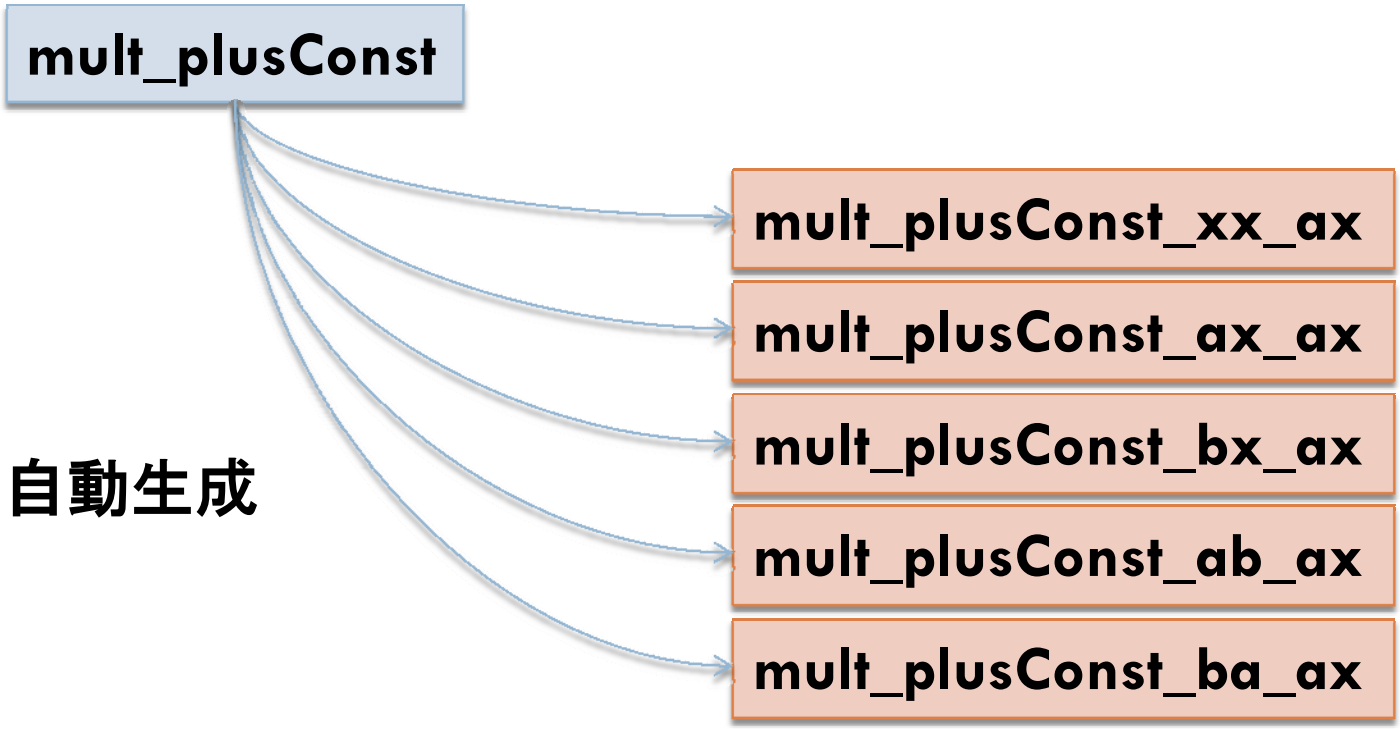
VM生成系：融合命令の生成

- 開発者が融合対象を指定
- VM生成系は融合命令のVMコード，コンパイラパターンマッチ用コードを自動生成



VM生成系：スタックキャッシング用命令生成

- プッシュ・ポップ処理をSC用レジスタアクセスにして自動生成
- コンパイラで遷移状態に応じた命令に変換



VM生成系に関する考察

- 実装は容易
  - 主に文字列置換
- 手作業では困難な最適化を適用
  - 前田他, Scheme インタプリタにおける仮想マシンアーキテクチャの最適化, IPSJ SIGPRO, 2003. →手作業での融合での高速化
  - 手作業によるバグの混入防止
- コード量が容易に増加
  - 最適化によっては膨大なコード量に
    - 基本命令：52命令 → 生成後：875命令
    - 命令キャッシュミス, ビルド時間が問題
  - 命令数削減が必要

評価：評価環境

- 評価環境

  Pentium-M 753（1.2GHz，L1I/D 32KB，L2S 2MB）、メモリ 1GB，WindowsXP + cygwin，GCC 3.4.4

- 比較言語処理系
  - Ruby 1.9.0 (2005-03-04)
  - Perl 5.8.6
  - Python 2.4.1
  - Gauche 0.8.4（Scheme 処理系）

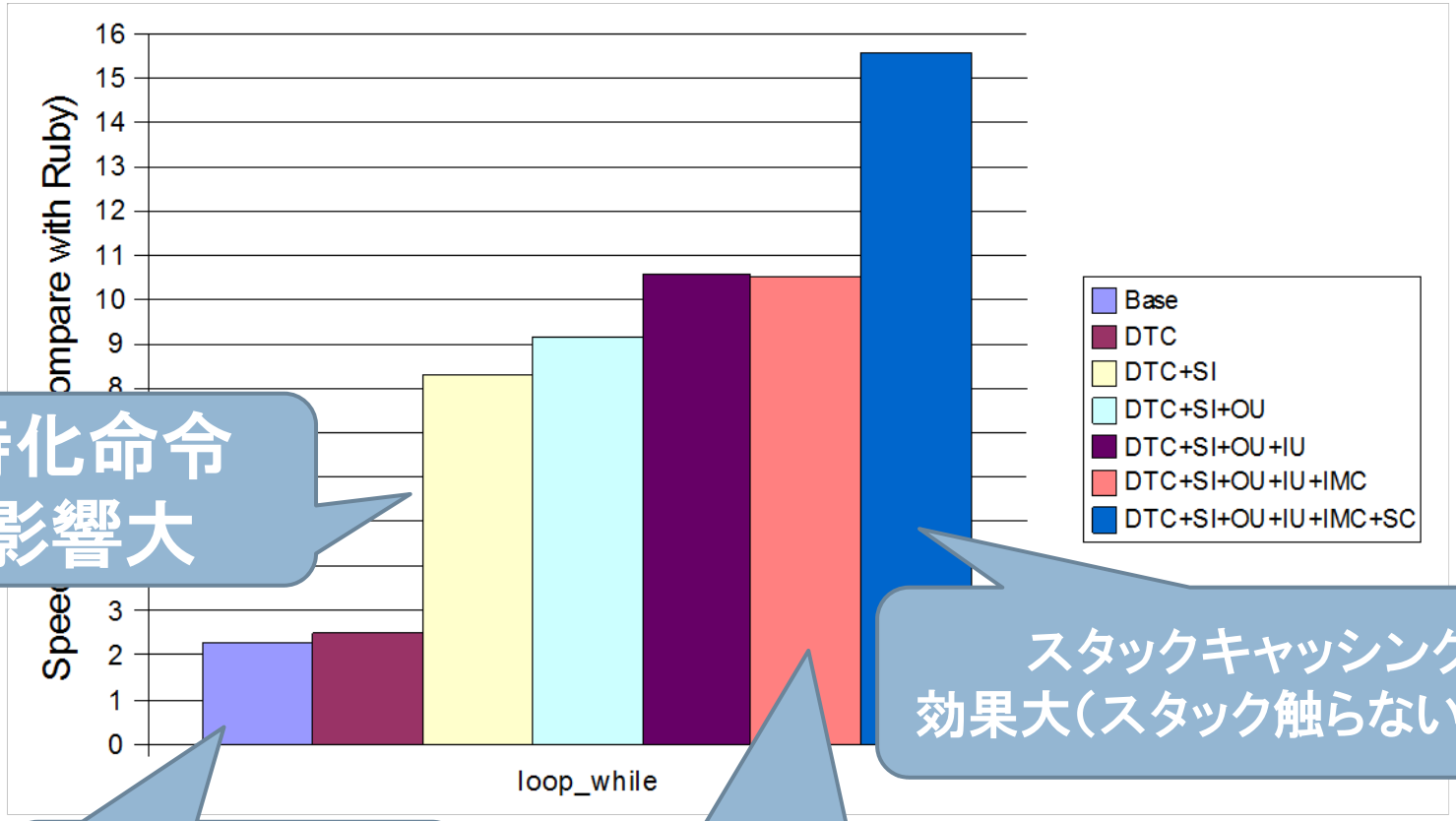
評価：例外処理

- 表引き法により，例外発生しない場合は高速に実行
- 例外が発生すると，例外表の検査のためオーバヘッド
  → 一般的に，例外発生は例外的なので本方式が有利

	30万回のループ	Ruby (sec)	YARV (sec)	Speedup
例外発生なし	ensure	1.77	0	-
	rescue	4.67	0.15	31.11
例外発生あり	rescue	3.65	5.51	0.66

```
x.times{begin; rescue/ensure; end}
```

評価：while loop



特化命令
影響大

スタックキャッシング
効果大（スタック触らないため）

VM化だけで
2倍高速化

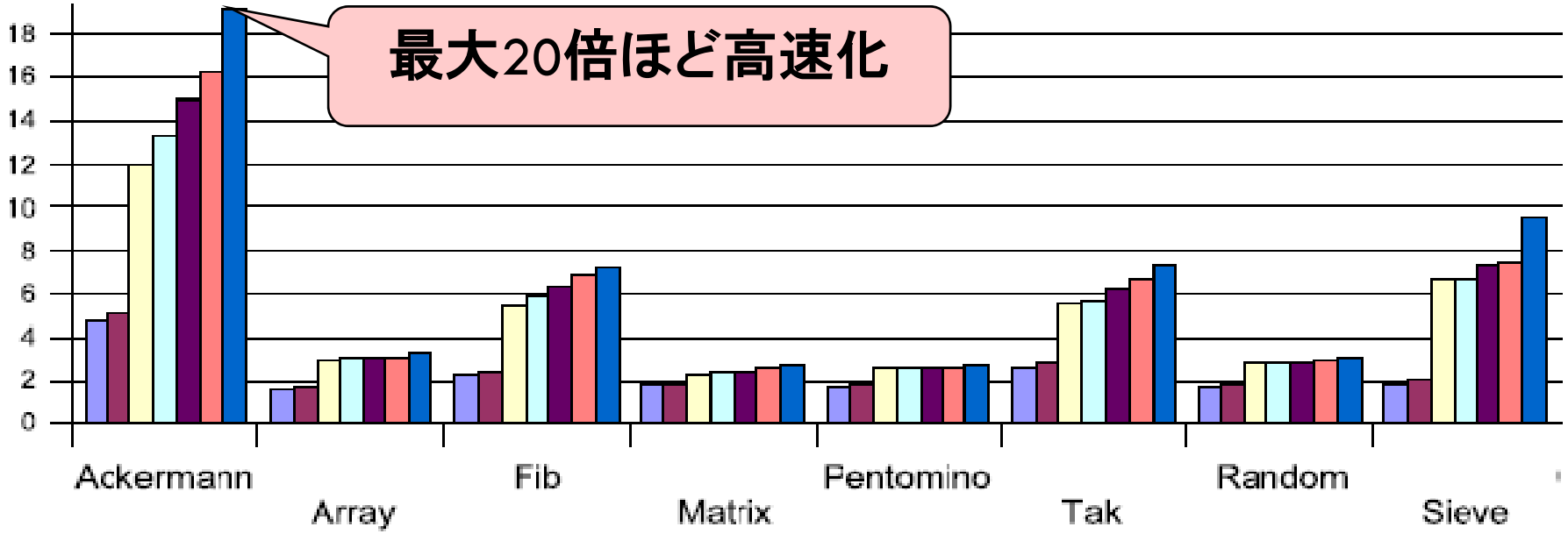
メソッド呼び出しが無い
のでインラインメソッド
キャッシュ効果なし

Base: V
DTC: D
SI: Spe

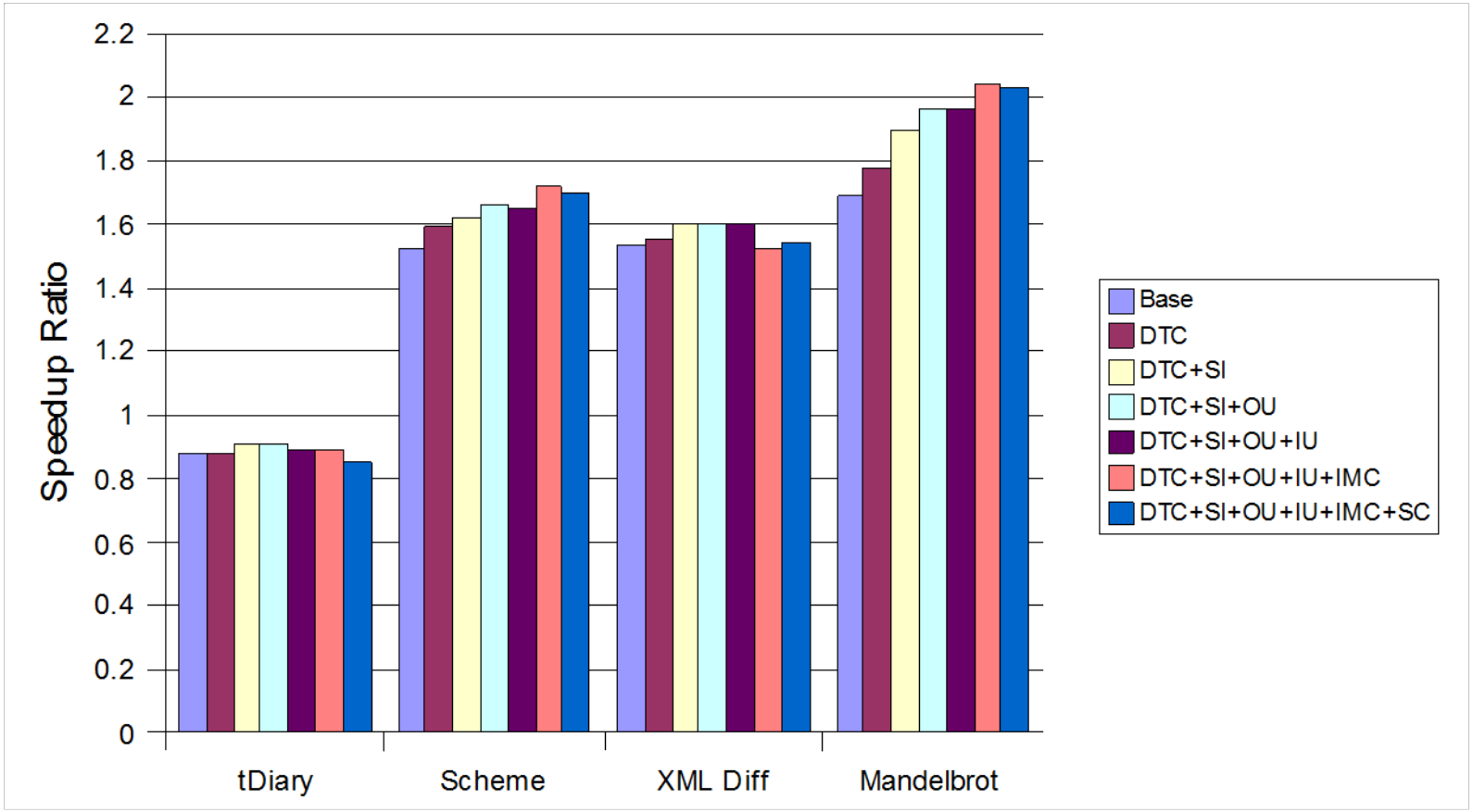
OU: C
IU: Ins

e Method Cache
Caching

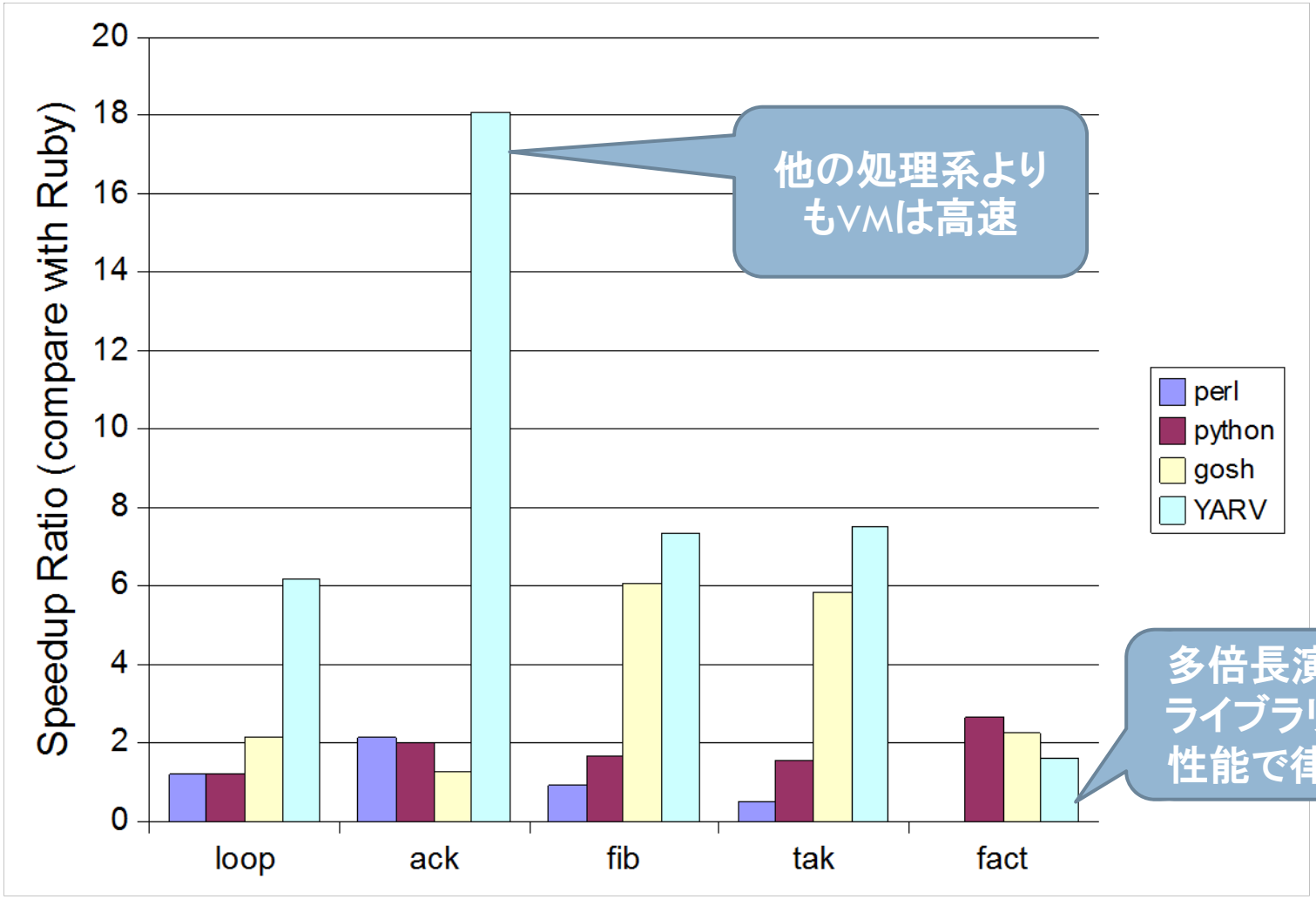
評価：高速化が有効な例


最大20倍ほど高速化
18
16
14
12
10
8
6
4
2
0
Ackermann
Array
Fib
Matrix
Pentomino
Tak
Random
Sieve

評価：マクロベンチマーク



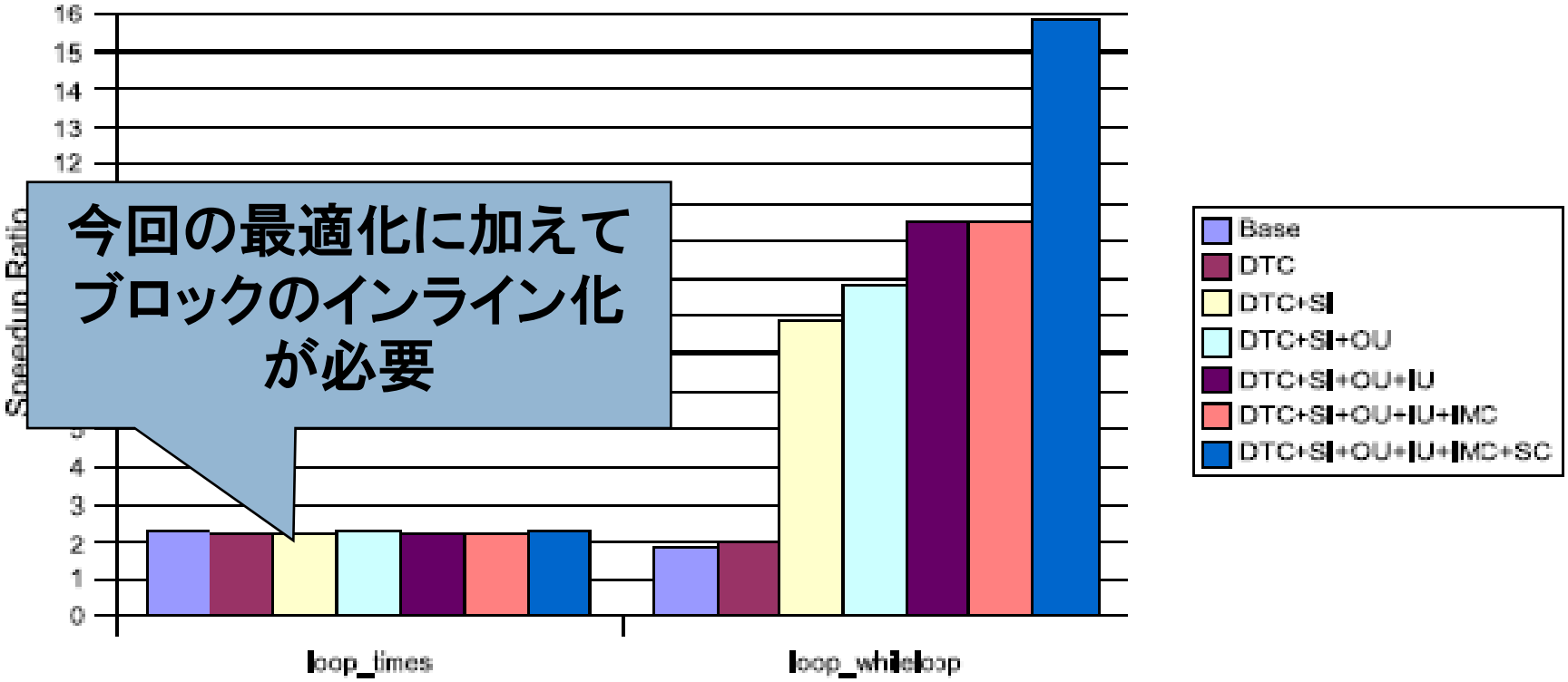
評価：他言語との比較


Speedup Ratio (compare with Ruby)
20
18
16
14
12
10
8
6
4
2
0
loop
ack
fib
tak
fact
perl
python
gosh
YARV
他の処理系より
もVMは高速
多倍長演算
ライブラリの
性能で律速

分析1

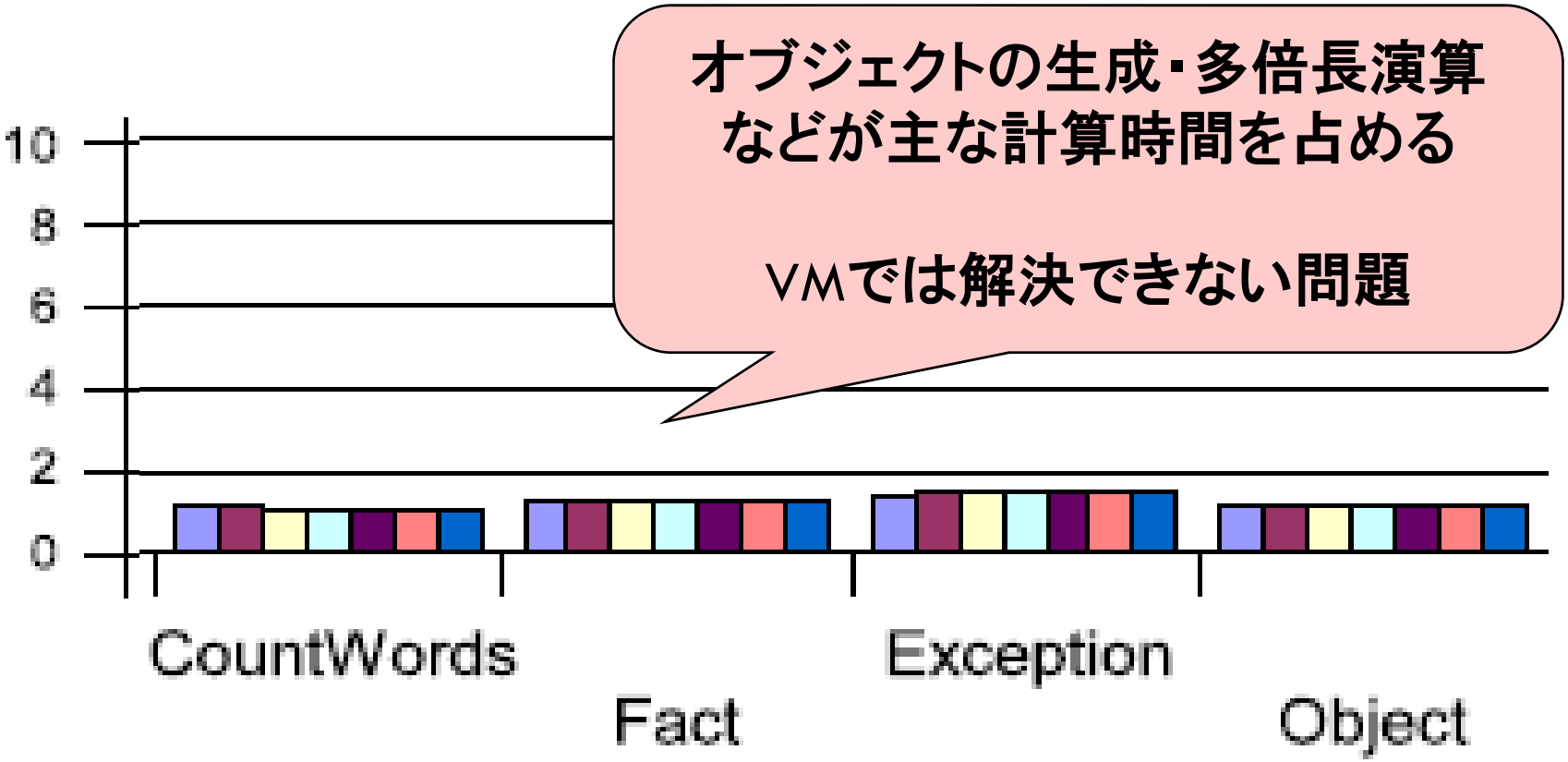
- VM化(Base)だけで性能向上を確認
  - どのベンチマークでも1.5倍～2倍
- 各最適化で性能向上を確認
  - 数値計算，記号処理などが高速化
  - 規模が小さいプログラムでスタックキャッシングが有利
  - 整数計算で特化命令が有利
- 融合操作のさらなる性能向上の可能性
  - マイクロベンチマークを基準に融合命令を選定
- tDiary の性能低下
  - eval() を利用しているため
  - そもそもVM化では速くならないプログラム
    - ライブラリを多用しているため

評価：繰り返し（w/block）



今回の最適化に加えて
ブロックのインライン化
が必要

評価：高速化の効果が小さい例



分析2

- ボトルネックがVM以外では各最適化効果なし
  - ライブラリの利用（文字列演算，多倍長演算）
  - オブジェクトの利用
  - 例外の発生
- ブロック呼び出しには各最適化効果なし
  - VM化による高速化は効果があった
  - Rubyではブロック呼び出しが多用されるため問題
  - ブロック呼び出し処理のオーバヘッド＞最適化効果
    →今後の課題

Agenda

- 背景と目的
- 高速なRuby用仮想マシン
- Rubyの並列化
- まとめ

Rubyの並列化

- 「並列」実行による高速化
  - 同時に複数のRubyプログラムを並列計算機上で動作
  - fork による並列実行はすでに可能
    - データ共有が難しく，表現できるプログラムに制限
    - 参考：dRuby – 手軽にRPC を実現
- Rubyは言語レベルでスレッドに対応
  - 「並行」実行をサポート

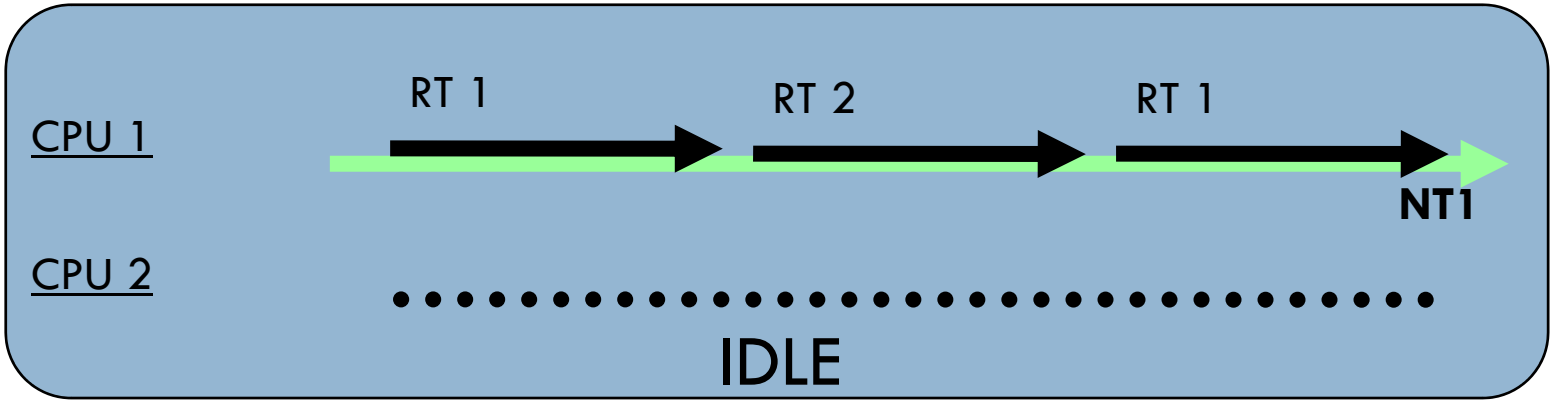
  → 並列実行への拡張は自然

目標と課題

- Rubyスレッドの並列実行によるRubyの並列化
  - ネイティブスレッド(NT)の利用
    - どのようにRubyスレッドとのマッピングするか？
  - VMでの適切な排他制御の導入
    - メモリ保護違反などを起こさない「安全」な実行に必須
    - 排他制御オーバヘッド削減が必要

現在のRubyスレッド(RT)処理機構とその問題点

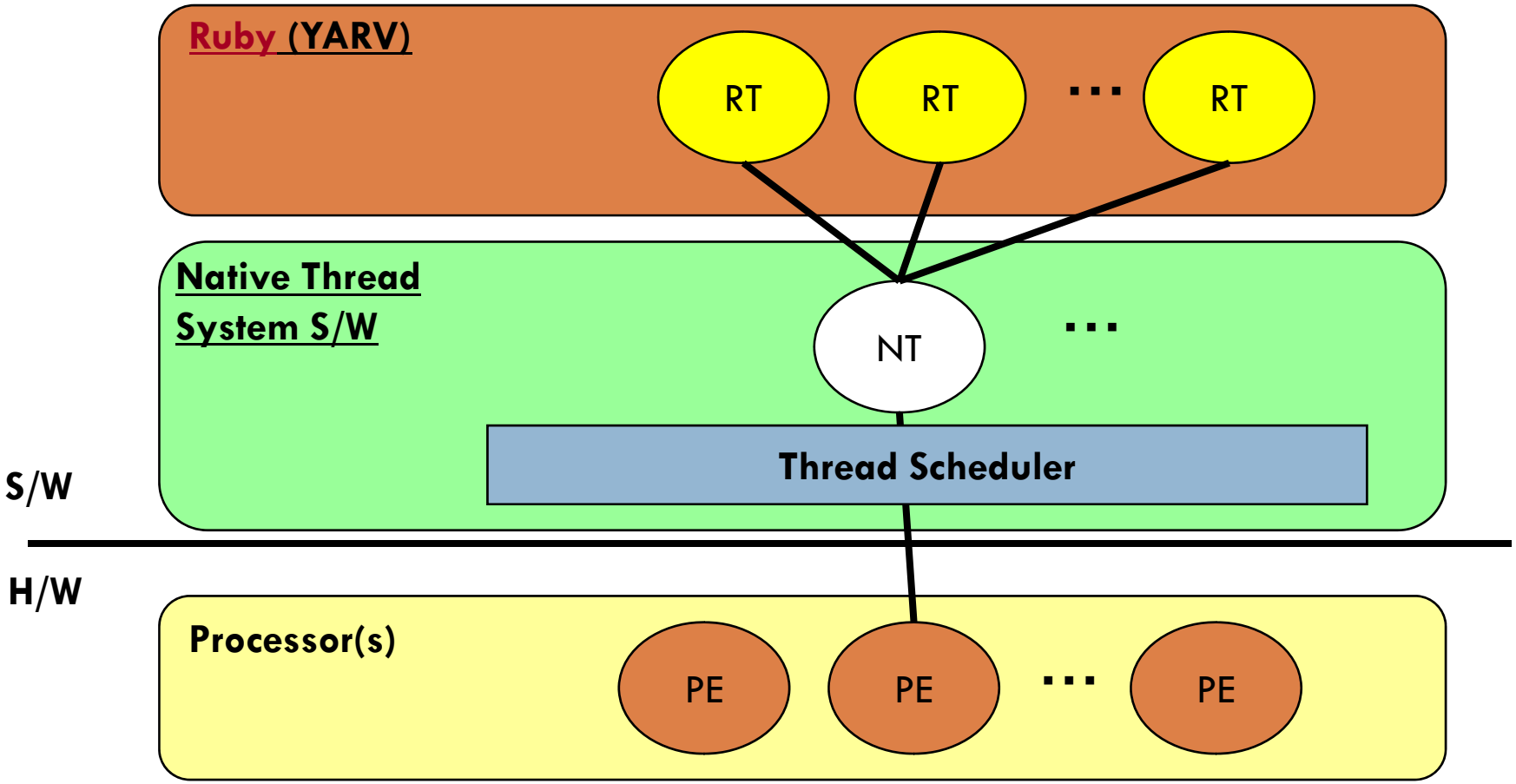
- 独自のユーザレベルスレッド
  - 並列実行不可
- スタックコピーによるスレッド切り替え
  - 切り替え時間がスタックの深さに比例



ネイティブスレッドを利用したRubyスレッド処理機構

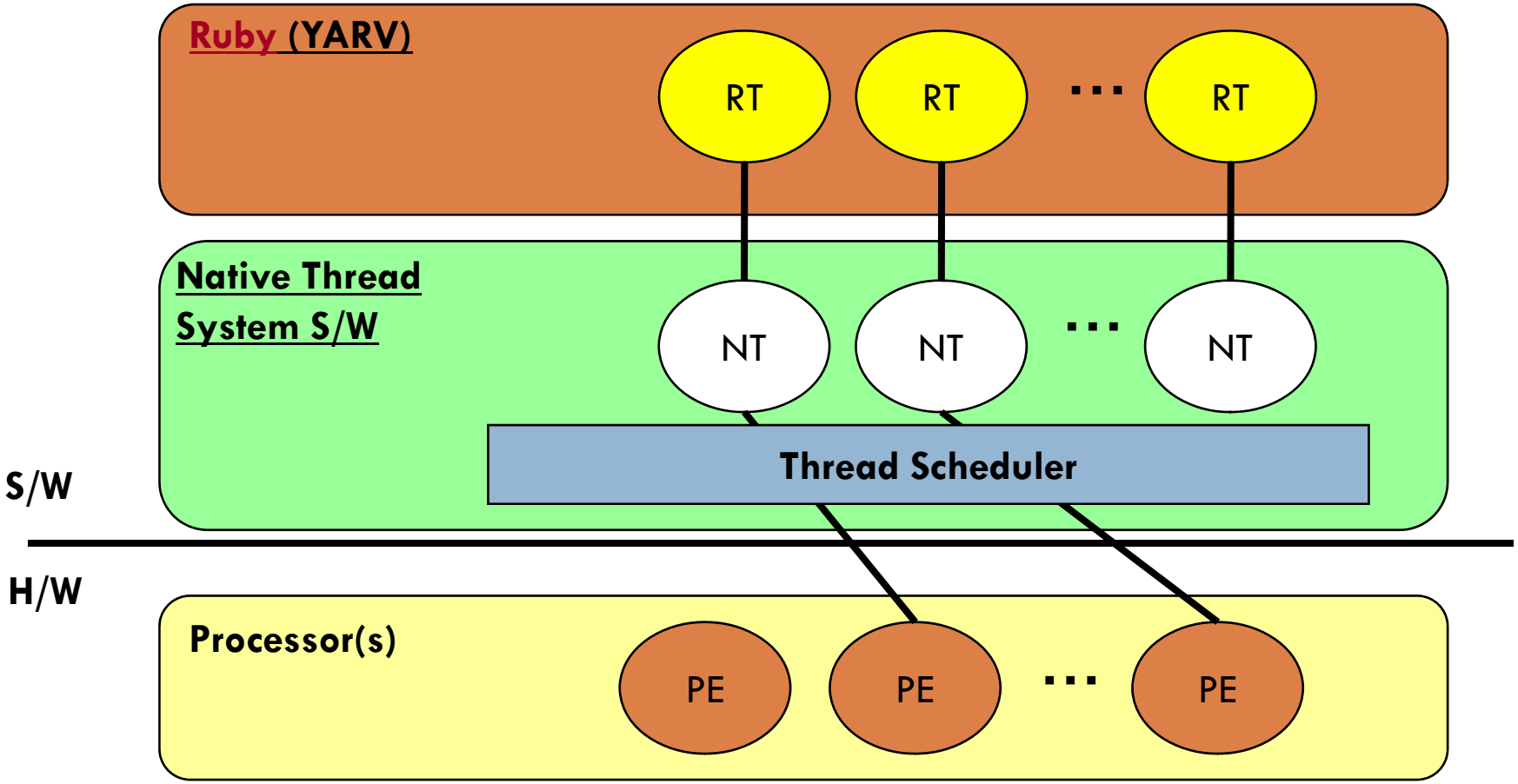
- ネイティブスレッドを利用して並列化
  - POSIX Thread (Pthread)
  - Windows Thread
- ネイティブスレッドとRubyスレッドのマッピング
  - 1:N → ユーザレベルスレッド
  - 1:1→ スレッド管理を全てNTで
  - M:N → 切り替えなどをユーザレベルで

検討(1)
Rubyスレッドとネイティブスレッド(1:N):CRuby



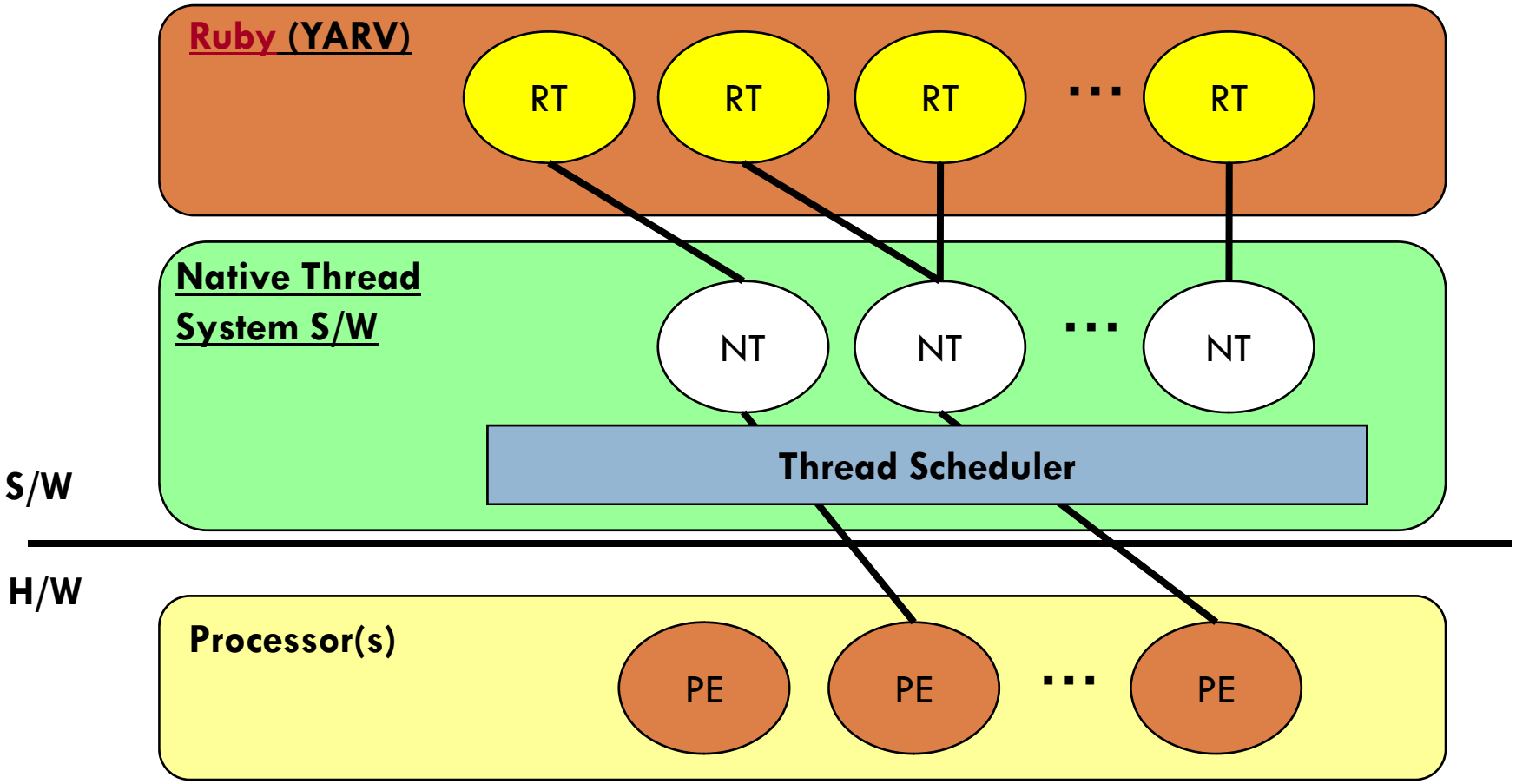
PE: Processor Element, UL: User Level, KL: Kernel Level

検討(2)
Rubyスレッドとネイティブスレッド(1:1)



PE: Processor Element, UL: User Level, KL: Kernel Level

検討(3)
Rubyスレッドとネイティブスレッド(N:M)



PE: Processor Element, UL: User Level, KL: Kernel Level

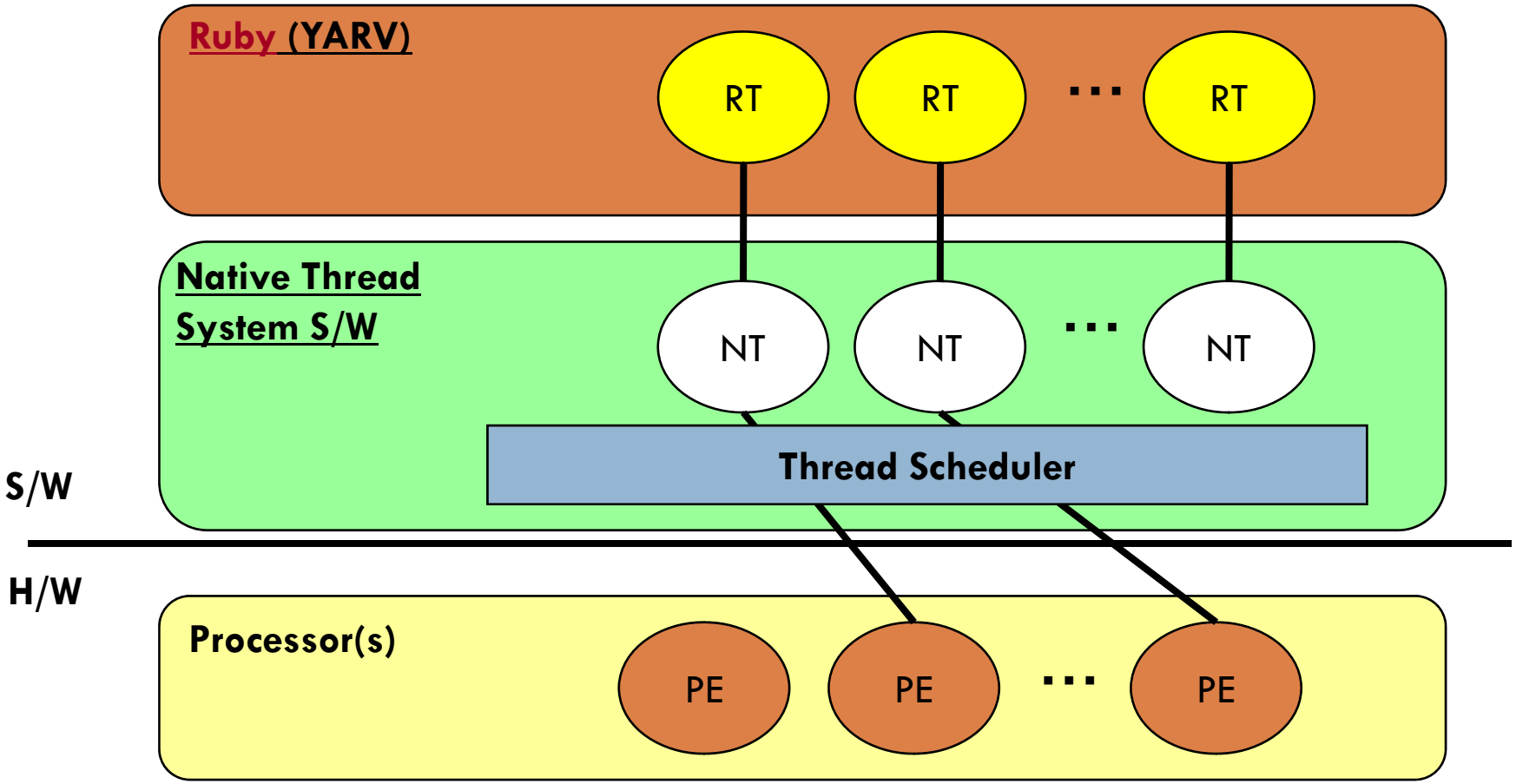
検討：ネイティブスレッド適用方式

- ネイティブスレッドとRubyスレッドのマッピング

	Pros	Cons
1:N	スレッド制御が軽い	並列実行できない
1:1	並列実行可能 シンプル,移植性が高い	スレッド制御が重い
N:M	スレッド制御が軽い 並列実行可能	複雑，移植性が悪い

- システムをシンプルにするために 1:1 モデルで
  - ネイティブスレッド処理機構にスレッド制御の性能を担保
    今後の研究開発に期待

システムの全体構成
Rubyスレッドとネイティブスレッド(1:1)



PE: Processor Element, UL: User Level, KL: Kernel Level

設計：Rubyスレッドのネイティブスレッド対応

- Rubyスレッドの生成・終了・排他制御スケジューリング
  - ネイティブスレッドに委譲
- ネイティブスレッド非対応機能の対応
  - Rubyスレッドの合流
  - 割り込み処理の対応

排他制御の導入

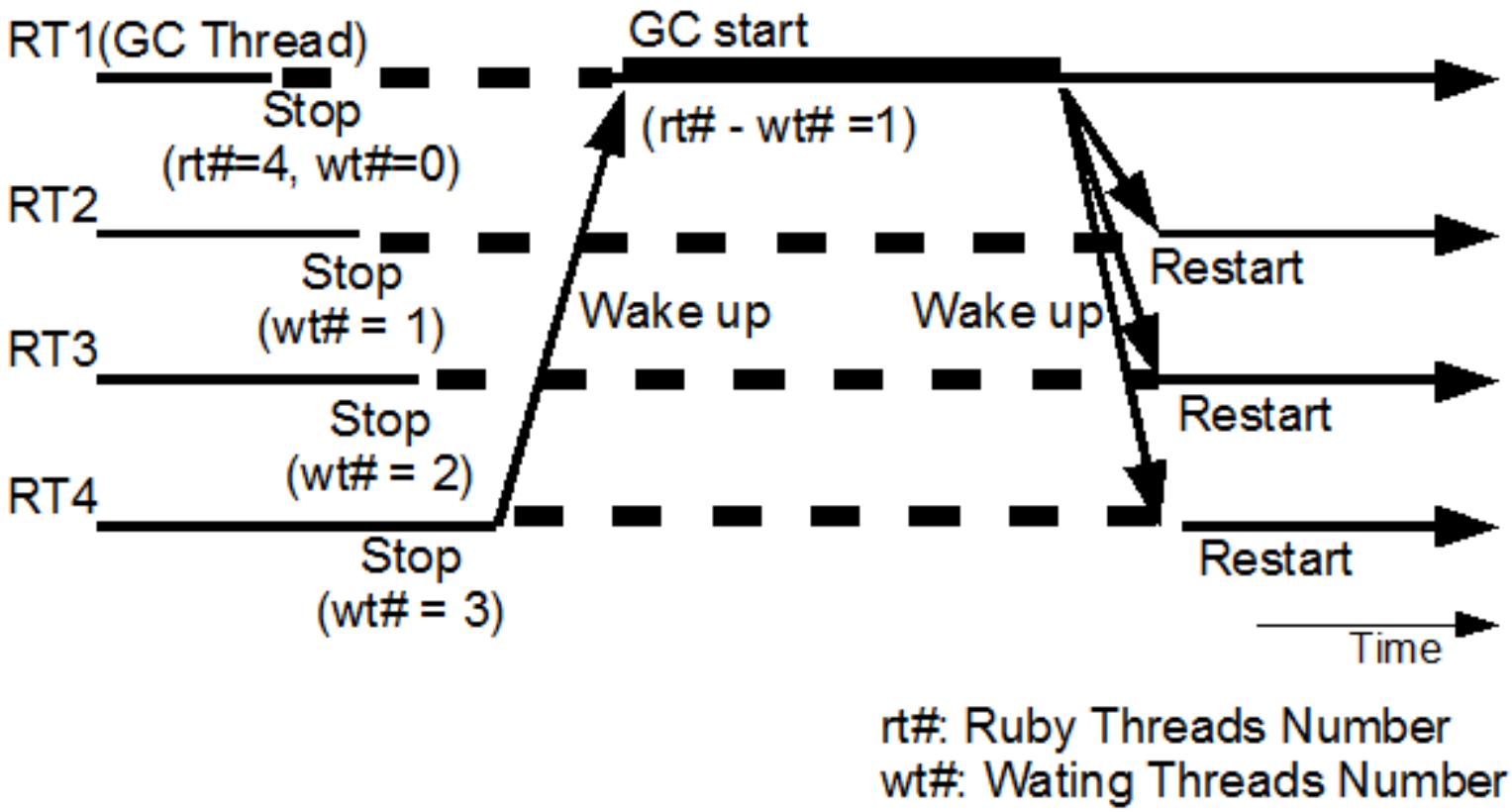
- VMに排他制御の導入が必要
  - Ruby VMは異常終了していはいけない
- 排他制御が必要な箇所
  1. スレッド間で共有するVM管理データ
  2. GCやオブジェクト管理
  3. インラインキャッシュ
  4. スレッドセーフでないネイティブメソッド

(1) スレッド間で共有する管理データ

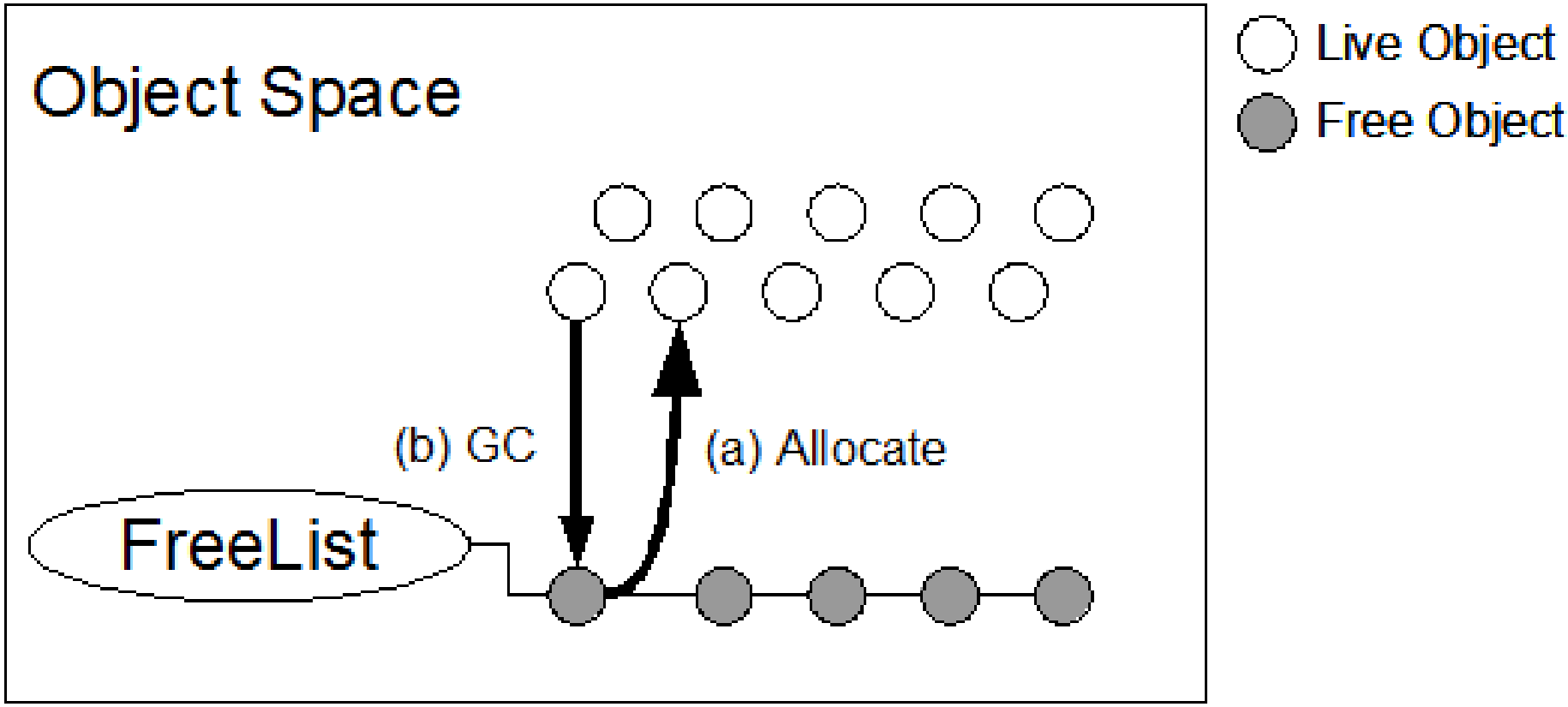
- 主に表で管理
  - グローバル変数名 → 実体
  - メソッド名 → メソッドボディ
  - …
- 表のアクセス時に排他制御

(2) GCやオブジェクト管理

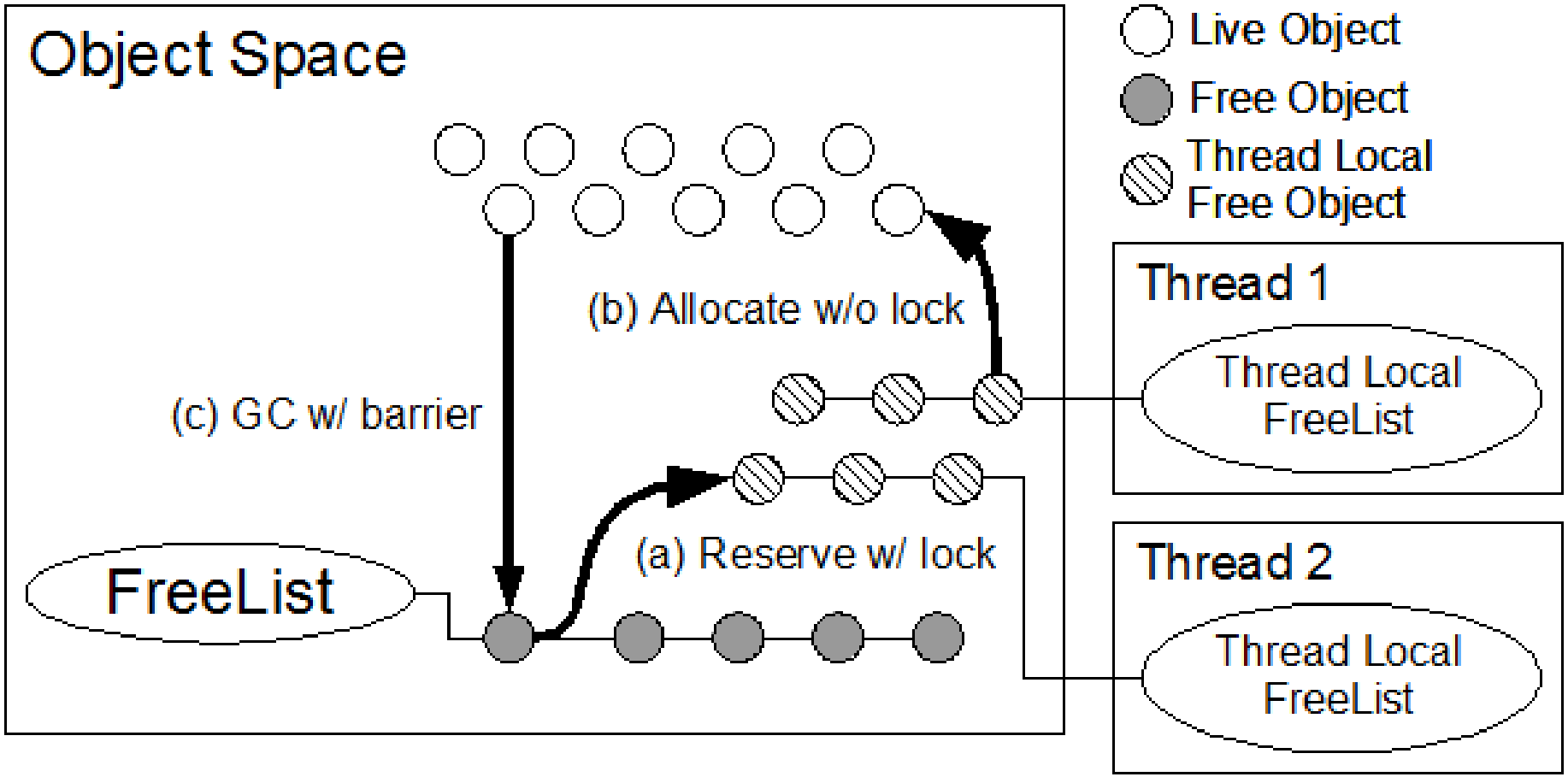
- GC時には全スレッドを停止し，逐次GC



(2) GCやオブジェクト管理
TLFLを用いた排他制御不要なオブジェクトの生成



(2) GCやオブジェクト管理
TLFLを用いた排他制御不要なオブジェクトの生成



(3) インラインキャッシュ

- VMの高速化に利用
  - キャッシュエントリを命令列に埋め込み
- 一貫性保持のための排他制御→性能に大問題
  - キャッシュエントリの「鍵」と「値」の一貫性
  - 高速化のための工夫が足を引っ張ってはいけない
- 排他制御不要なインラインキャッシュ
  - キャッシュミス時 → キャッシュエントリを生成
    - 古いキャッシュエントリはゴミに（GC任せ）
    - 生成オーバヘッド増，しかしミスは少ないため問題無し

(4) スレッドセーフでないネイティブメソッド

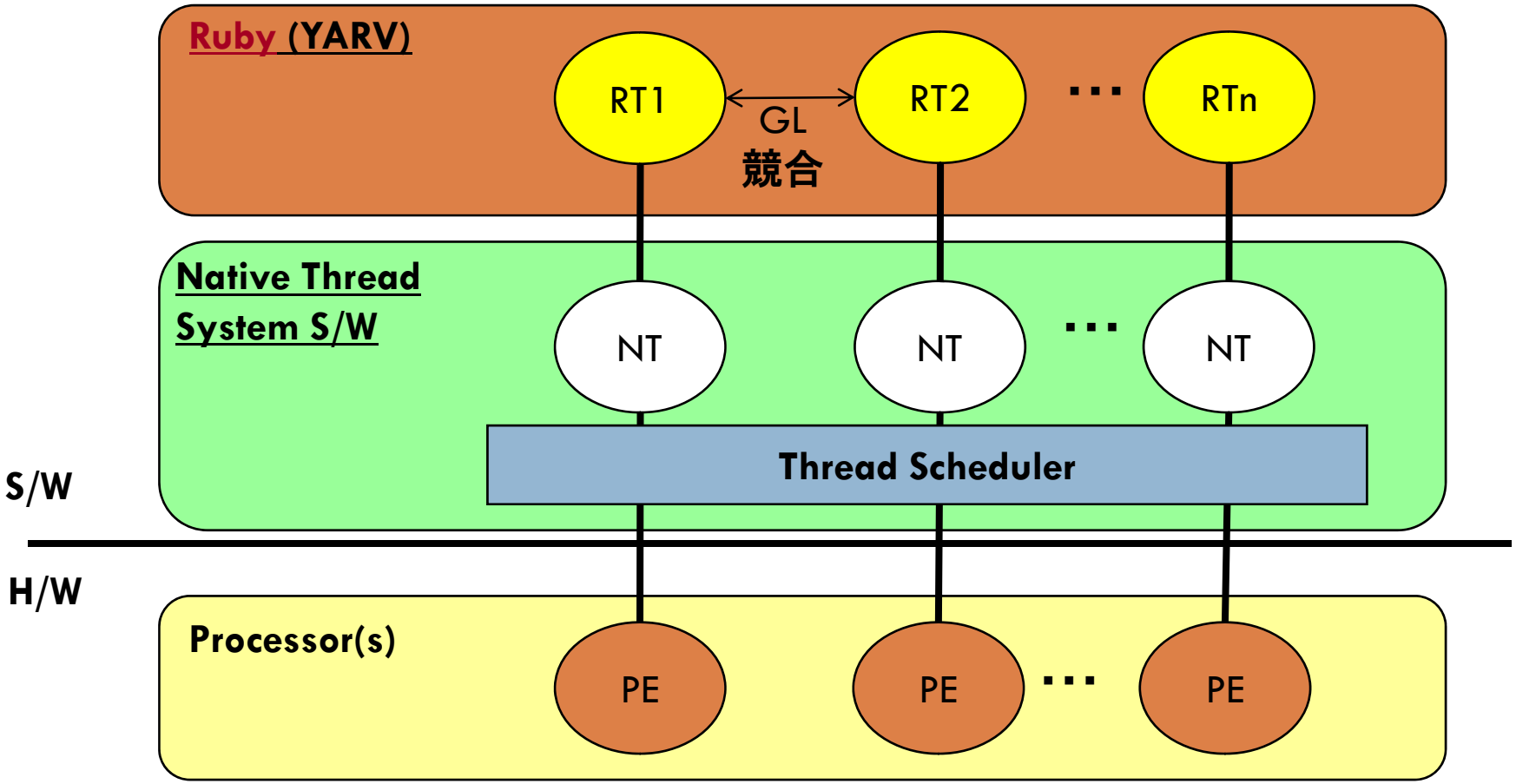
- CRuby にはCで記述したRubyを拡張するネイティブメソッドが多数存在→非スレッドセーフ
  - 並列実行していないから必要なかったため
- ジャイアントロックで保護
  - 過去のライブラリを使うため
- スレッドセーフに書き換え

  → GL 不要に
- 段階的な並列度向上が可能

利用CPU制限によるGL競合回避

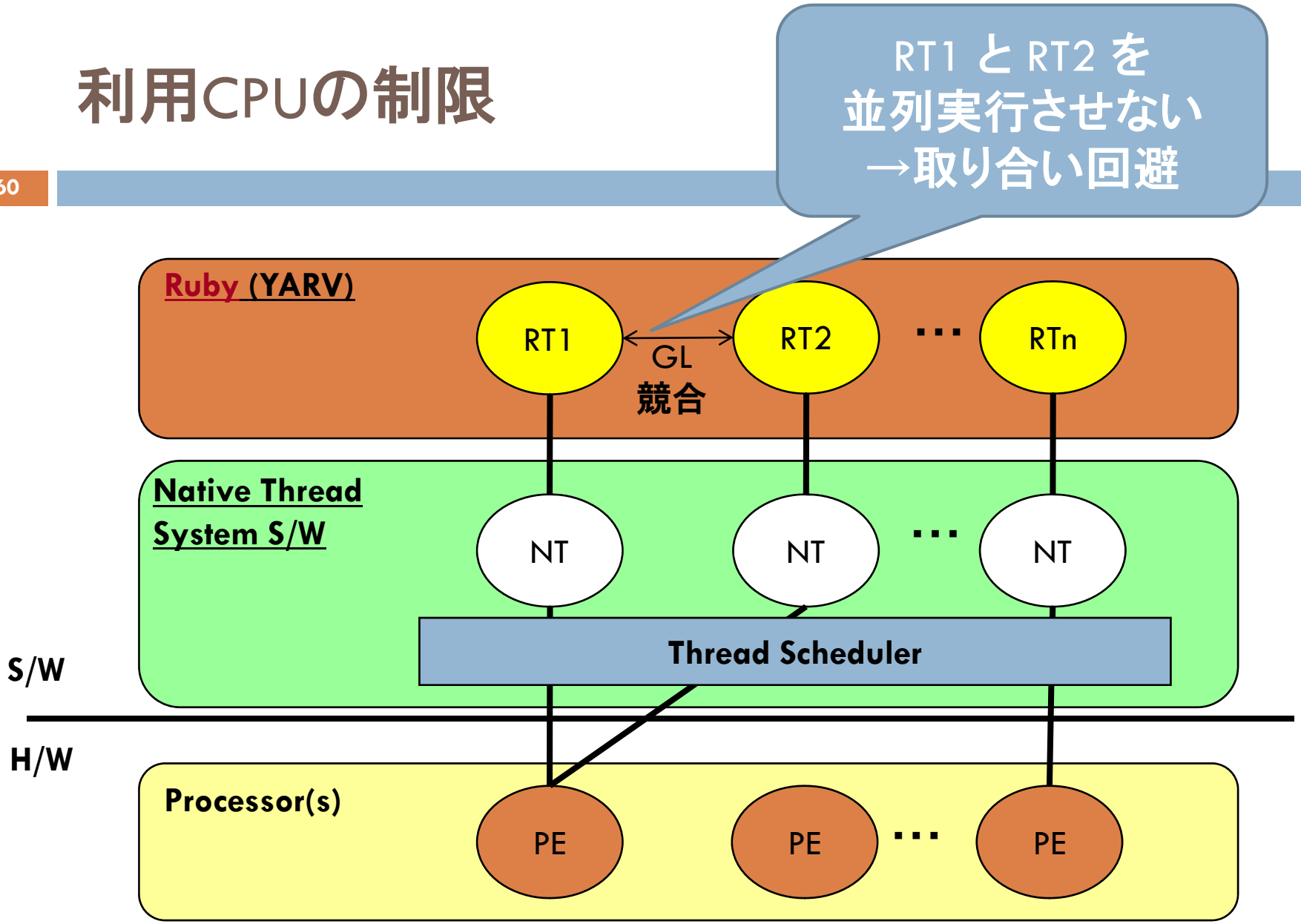
- GL競合(取り合い)による性能低下
  → 利用CPUの制限
  - GL競合回数を一定間隔で監視
  - 競合を繰り返すRubyスレッドの実行CPUを制限
  - NPTLのpthread_setaffinity_npを利用
    - Windows では SetThreadAffinityMask
  - 一定時間過ぎたら制限を解除

利用CPUの制限



PE: Processor Element, UL: User Level, KL: Kernel Level

利用CPUの制限



PE: Processor Element, UL: User Level, KL: Kernel Level

評価
評価環境

- 評価環境
  - CPU: Intel Xeon CPU E5335 2.0GHz

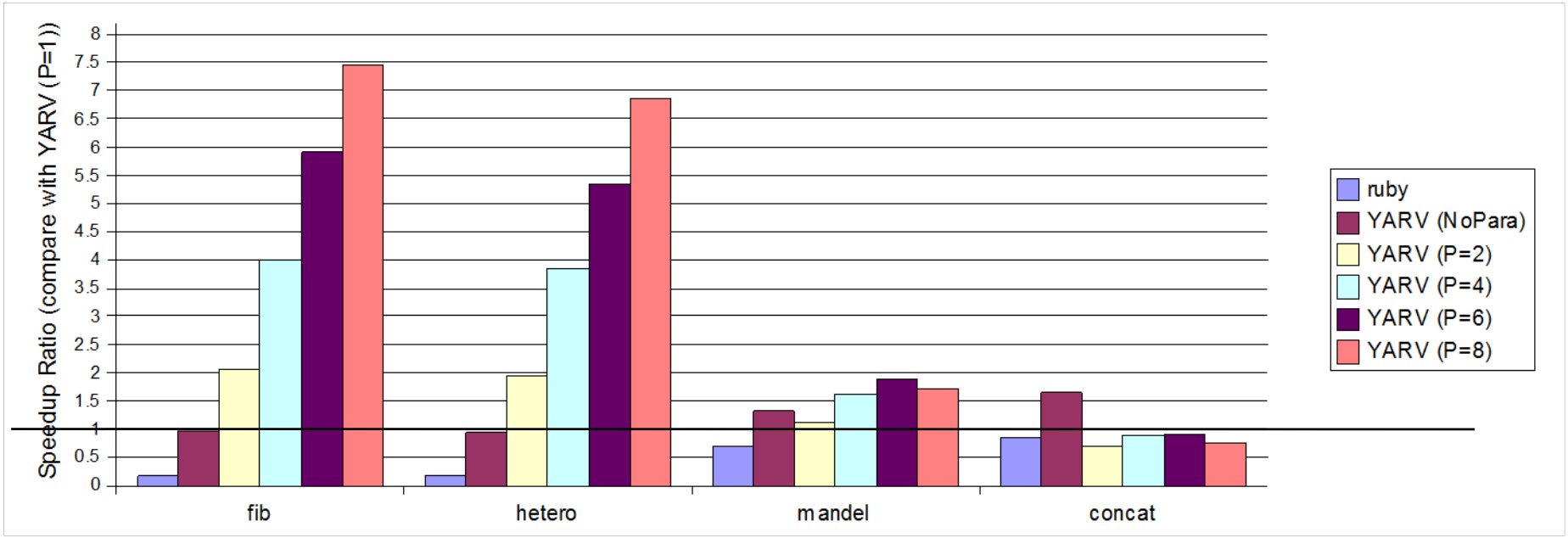
    Quad core x 2 = 8 core
  - OS: GNU/Linux 2.6.18 x86_64 SMP / NPTL
  - Compiler: gcc version 4.1.2
- 比較対象のRuby
  - ruby 1.8.6 (2007-11-02) [x86_64-linux]
- YARV最適化オプション
  - 融合操作とStack caching 以外すべて

評価
スレッド制御

- 生成，切り替え：10万回，排他制御：100万回
- 生成・合流はNT制御オーバヘッドで低速
  - ネイティブスレッド制御オーバヘッド
  - VM用スタック・スレッドローカルフリーリスト割り当てオーバヘッド
- 排他制御は高速
- ネイティブスレッドによるスタックの深さに依存しない高速なスレッド切り替え

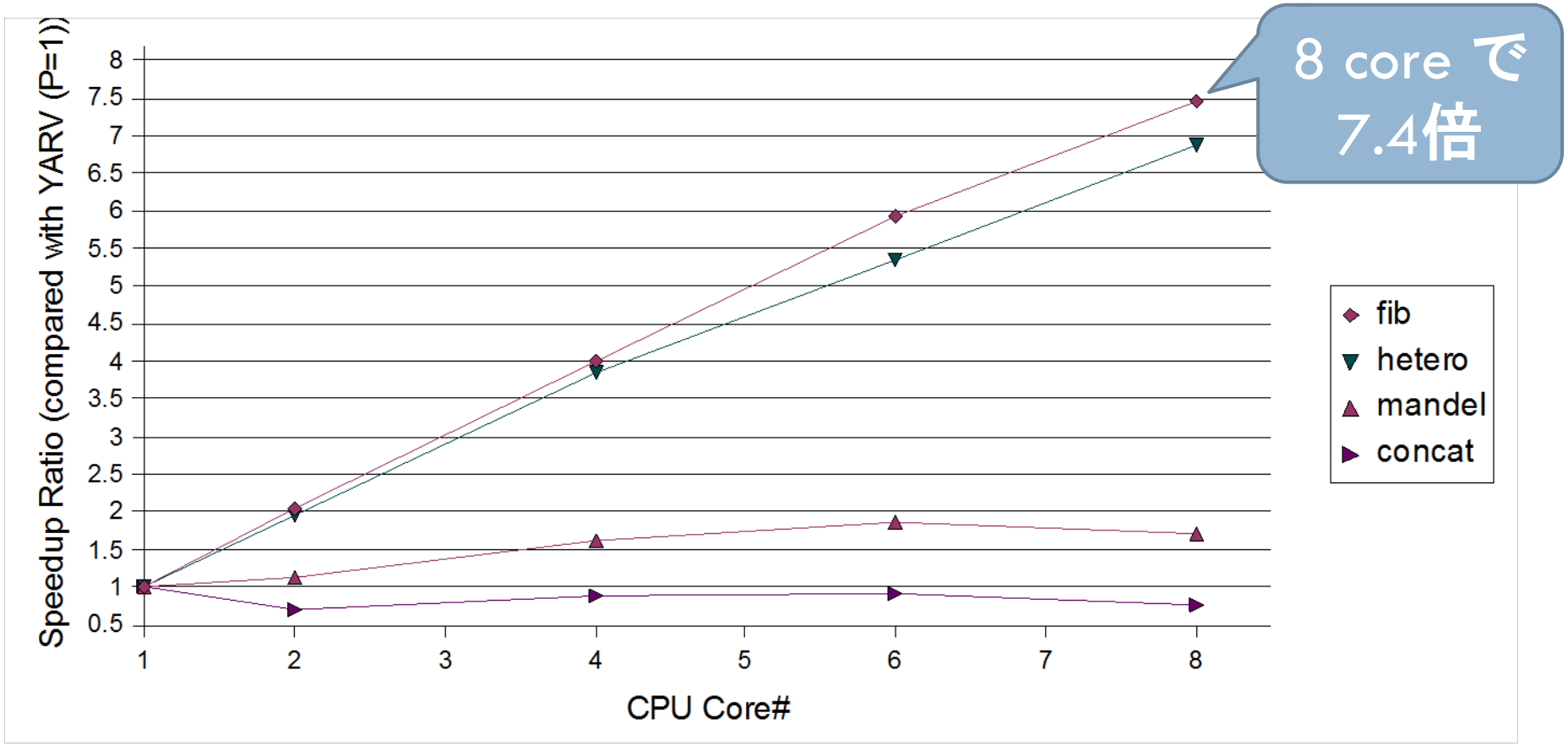
	Ruby (sec)	YARV (sec)	Ratio	NTPL (sec)
生成	0.89	1.95	0.46	0.59
排他制御	0.67	0.38	1.76	-
切り替え（深さ1）	6.01	0.06	100.17	-
切り替え（深さ16）	11.55	0.06	192.5	-

評価
並列実行(マイクロベンチマーク)

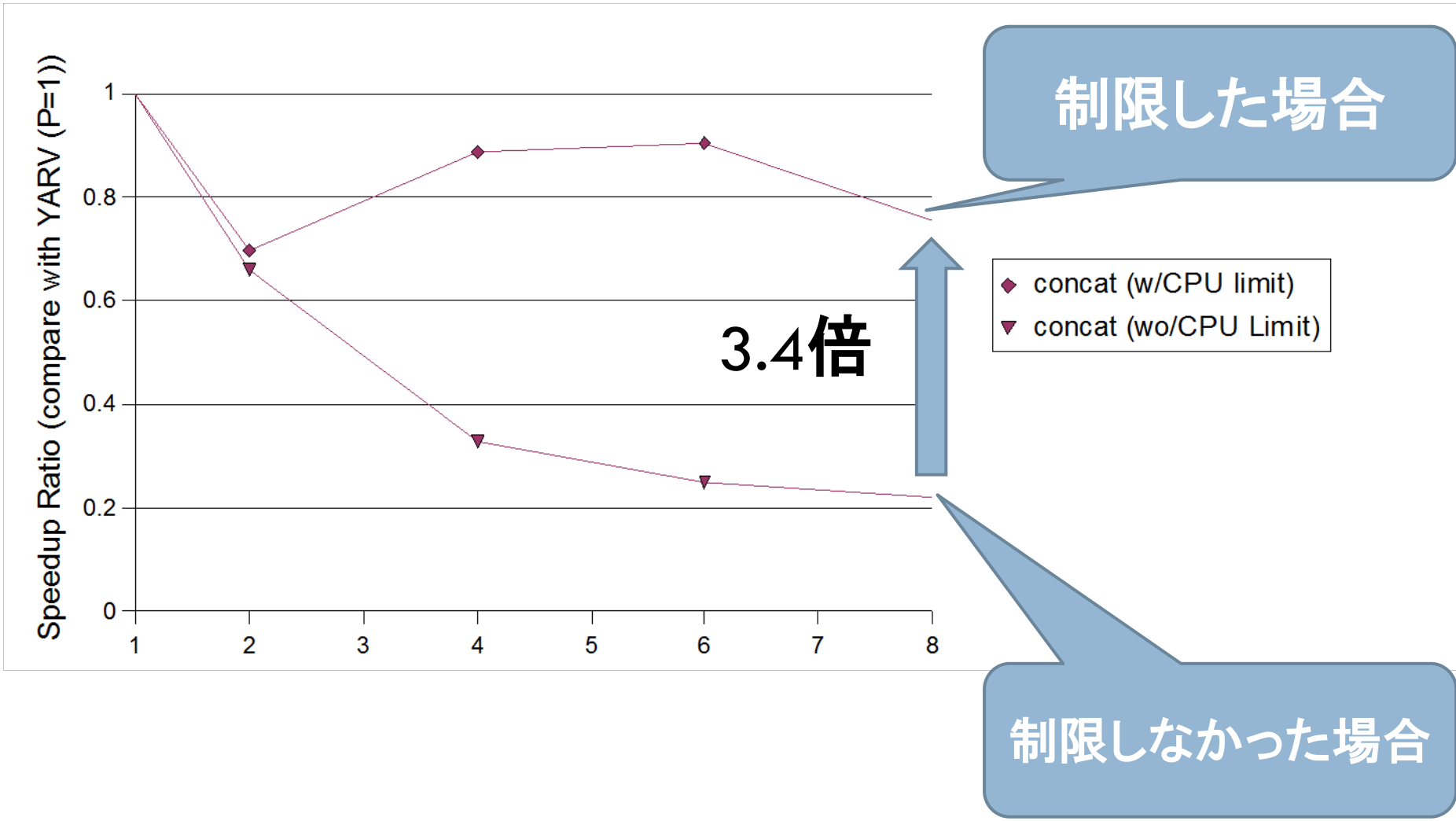


- fib: N番目のfib数を再帰で求める. N>30でスレッド生成
- hetero: fib + concat (1 スレッド)
- mandel: Mandelbrot 集合を求めるプログラム(GC多発)
- concat: 複数のスレッドで一つの文字列を追加
  (そもそも並列度無し・GL競合)

評価：マイクロベンチマーク



評価：CPU制限


Speedup Ratio (compare with YARV (P=1))
1
0.8
0.6
0.4
0.2
0
1
2
3
4
5
6
7
8
concat (w/CPU limit)
concat (wo/CPU Limit)
3.4倍
制限した場合
制限しなかった場合

分析

- GL, GCを利用しないプログラム，GL競合を起こさないプログラムでの台数効果を確認
- MandelはGC時間で律速
- GL競合によるオーバヘッド
  - 現在，GLを必要とするネイティブメソッドは多いため，この問題は頻出
  - 競合回避のためのCPU制限は有効

Agenda

- 背景と目的
- 高速なRuby用仮想マシン
- Rubyの並列化
- まとめ

成果

高速なRuby処理系の実現

仮想マシンYARVの開発

VMの設計
→1.5倍～4倍の性能向上
実行時最適化
→最大20倍
VM生成系の利用

Rubyの並列実行

Rubyスレッドを並列実行
ネイティブスレッドの利用
排他制御の導入
→ 8 core で最大 7.4 倍
排他制御オーバヘッド削減
→最悪値に比べ3.4倍高速

まとめ

- 高性能なRuby用仮想マシン構成
  - VM化による性能向上の確認
    - スタックマシン型バイトコードへコンパイルし実行するモデル
    - 例外表の利用
    - 1.5～4倍程度の性能向上を確認
  - 動的特性を考慮した最適化手法
    - 既知の最適化が最大20倍，マイクロベンチマークでとくに有効
  - VM生成系を利用したVM構成法
    - 最適化命令を容易に生成
    - 簡単な生成系でも有効

まとめ

- Rubyの並列化
  - Rubyスレッドの並列実行
    - 8 core で 7.4 倍の性能向上
  - ネイティブスレッドを適用(1:1)
    - 移植性，性能，システムの複雑度の妥協点
  - VMへの適切な排他制御の導入とその軽量化
    - VM管理データ，GC，インラインキャッシュ，…
    - ネイティブメソッド→ジャイアントロックの利用
      - GL競合による性能低下は著しいが，それを回避する手段により，ある程度性能低下を抑制

今後の課題

- VMの高速化
  - さらに高速な命令ディスパッチ手法の適用
  - JIT/AOTコンパイラの開発
  - 静的解析の活用
- 並列化
  - ネイティブメソッドのスレッドセーフ化
    - データ構造の検討
  - 並列GCの導入
  - 「スレッド」以外の並列化手法の提供
    - MVM
    - 分散拡張

成果(論文・発表)

- 主著論文
  - 笹田耕一, 松本行弘, 前田敦司, 並木美太郎: Ruby用仮想マシンYARVの実装と評価,情報処理学会論文誌(PRO), Vol.47, No.SIG 2(PRO28), pp.57-73 (2006.2).
  - 笹田耕一, 松本行弘, 前田敦司, 並木美太郎: Ruby用仮想マシンYARVにおける並列実行スレッドの実装, 情報処理学会論文誌, Vol.48, No.SIG10(PRO33), pp.1-16 (2007)
- 主著論文(参考)
  - 笹田耕一, 佐藤未来子, 河原章二, 加藤義人, 大和仁典, 中條拓伯, 並木美太郎: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, Vol.44, No.SIG11(ACS3), pp. 215--225 (2003).
  - 笹田耕一, 佐藤未来子, 内倉要, 小笠原嘉泰, 中條拓伯, 並木美太郎:SMTプロセッサ向けの軽量な同期機構, 情報処理学会論文誌, Vol.46, No.SIG16(ACS12), pp.14-27 (2005.12)
- 国際会議
  - K.Sasada, M.Sato, S.Kawahara, N.Kato, M.Yamato, H.Nakajo and M.Namiki: Implementation and Evaluation of a Thread Library for Multithreaded Architecture, The 2003 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'03), Vol.II, pp. 609--615 (2003).
  - K.Sasada: YARV: Yet Another RubyVM - Innovating the Ruby Interpreter, Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2005, pp.158--159 (2005).
  - RubyConf 2004 ~ 2006: Oral Presentation

研究業績(その他発表)

- 1)笹田耕一, 佐藤未来子, 河原章二, 加藤義人, 大和仁典, 中條拓伯, 並木美太郎: 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, SACSIS (Symposium on Advanced Computing Systems and Infrastructures) 2003, Vol.2003, No.8, pp. 13--20 (2003).
- 2)笹田耕一: Rubyプログラムを高速に実行するための処理系の開発, 情報処理学会第46回プログラミングシンポジウム報告集, pp. 1--10 (2005).
- 3)笹田耕一, 佐藤未来子, 内倉要, 加藤義人, 大和仁典, 中條拓伯, 並木美太郎: SMTプロセッサにおける細粒度最適化手法の検討, 情報処理学会研究報告(SWoPP青森2004), Vol.2004-ARC-159, pp. 37--42 (2004).
- 4)笹田耕一, 佐藤未来子, 内倉要, 加藤義人, 大和仁典, 中條拓伯, 並木美太郎: SMTプロセッサにおける同期方式の検討, 情報処理学会第154回計算機アーキテクチャ・第101回ハイパフォーマンスコンピューティング合同研究会(HOKKE-2005), Vol.2005-ARC-162, pp. 31--36 (2005).
- 5)笹田耕一, まつもとゆきひろ, 前田敦司, 並木美太郎: Ruby用仮想マシンYARVの実装と評価, 情報処理学会PRO研究会 SWOPP2005 (2005).
- 6)笹田耕一, まつもとゆきひろ, 前田敦司, 並木美太郎: Ruby用仮想マシンYARVにおける並列実行スレッドの実装, 第61回 情報処理学会PRO研究会 豊橋商工会議所 (2006.10).
- 7)笹田耕一, 並木美太郎: Ruby による JavaVM の実装, 情報処理学会第65回全国大会予稿集 (2003).
- 8)笹田耕一: プログラム言語 Ruby におけるメソッドキャッシング手法の検討, 情報処理学会第67回全国大会, Vol.1, pp. 305--306 (2005).

研究業績(その他)

- 解説論文
  - 1)ささだこういち: Ravaで見るJava仮想マシンのしくみ, Java Press, Vol.31, pp. 158--165 (2003).
  - 2)ささだこういち: 日本発LL—Rubyプログラミングステップアップ, Software Design (2006).
  - 3)ささだこういち: Parrotいんたーなる, まるごとPerl, Vol.1 (2006)
- その他
  - 1)ささだこういち: YARVについてLightning Talk, Lightweight Language Weekend (2004.8)
  - 2)ささだ こういち: 日本Rubyの会自主企画「RubyConf2004 に参加して」, 関西オープンソース 2004, 大阪産業創造館 (2004.10)
  - 3)ささだこういち: 日本Rubyの会自主企画「Ruby Hot Topics ～YARV and Rails～」YARV について, Open Source Conference 2005 (2005.3)
  - 4)ささだこういち: Ruby Language Update, Lightweight Language Day and Night (2005.8)
  - 5)ささだこういち: YARVについて(招待講演), Ruby勉強会@関西 (2005.11)
  - 6)笹田耕一:「Rubyを実装するということ」 ESPer2006 未踏集会 (2006.5)
  - 7)笹田耕一: YARVヒッチハイクガイド, 筑波大学大学院 システム情報工学研究科 コンピュータサイエンス専攻 コロキウム (2006.9)
  - 8)笹田耕一:Ruby 1.9実装の現状と今後 日本Ruby会議 2007 (2007.6)
  - 9) まつもとゆきひろ, 笹田耕一:Ruby Language Update Lightweight Language Spirits

研究業績(授賞等)

- 2003年 5月 SACSIS 2003 優秀学生論文賞受賞 「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装と評価」　笹田 耕一, 佐藤 未来子, 河原 章二, 加藤 義人, 大和 仁典, 中條 拓伯, 並木 美太郎
- 2004年 3月 東京農工大学　阿刀田基金 受給
- 2005年 5月 IPA 2004年度未踏ソフトウェア創造事業(未踏ユース)スーパークリエータ認定
- 2005年 5月 SACSIS 2005 優秀若手研究賞(共著) 「SMTプロセッサにおけるスレッドスケジューラの開発」　内倉要, 佐藤未来子, 笹田耕一, 加藤義人, 大和仁典, 中條拓伯, 並木美太郎
- 2005年5月SACSIS2005 最優秀独創的研究アイデア賞(共著) 「SMTプロセッサのFPGAへの実装と評価」　加藤義人, 大和仁典, 小笠原嘉泰,佐藤未来子, 笹田耕一, 内倉要, 中條拓伯, 並木美太郎
- 2007年10月IPA OSS貢献者賞受賞

成果(開発したソフトウェア)

- Ruby 1.9.1 (2007年 12月リリース予定…?)
  - 本研究の成果が取り込まれる予定
  - いくつかの最適化は制限
    - 命令融合, スタックキャッシング
    - 一般的に有利かどうかはまだわからないため
  - 並列化はなし(YARV (NoPara))
- IPA未踏ソフトウェア創造事業の支援
  - 2004年6月～2005年2月　IPA未踏ユース採択(PM:筧早稲田大学教授)
  - 2005年6月～2006年2月　IPA未踏採択(PM:千葉東工大助教授)
  - 2006年11月～2007年8月　IPA未踏採択(PM:千葉東工大助教授)

今後の展望

- Rubyの発展
  - 本研究によってRubyの利用可能範囲の拡大
- Rubyのような動的言語のさらなる需要大
  - 速度性能よりも開発速度重視の時代
  - 本研究で示したVMの実装・最適化技術が利用可能
- 並列計算機で簡単に性能を出すスクリプト言語
  - 簡単な記述で大きな計算パワーを利用
  - 本研究ではその1手段を提示

Rubyが遅いことに関して…続き

I understand that there are plans for Ruby to address performance with some kind of **bytecode compiler** *and that will be nice. When these things happen and when Ruby starts getting competitive benchmarks it will be a lot more* **appropriate for a lot more types of applications.**

Ruby Performance Revisited

http://joelonsoftware.com/items/2006/09/12.html

Joel Spolsky

以上

ご静聴ありがとうございました.

東京大学大学院情報理工学系研究科　特任助教
笹田耕一
ko1@atdot.net

研究業績(論文)

- 1)笹田耕一, 佐藤未来子, 河原章二, 加藤義人, 大和仁典, 中條拓伯, 並木美太郎: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, Vol.44, No.SIG11(ACS3), pp. 215--225 (2003).
- 2)笹田耕一, 佐藤未来子, 内倉要, 小笠原嘉泰, 中條拓伯, 並木美太郎:SMTプロセッサ向けの軽量な同期機構, 情報処理学会論文誌, Vol.46, No.SIG16(ACS12), pp.14-27 (2005.12)
- 3)笹田耕一, 松本行弘, 前田敦司, 並木美太郎: Ruby用仮想マシンYARVの実装と評価,情報処理学会論文誌(PRO), Vol.47, No.SIG 2(PRO28), pp.57-73 (2006.2).
- 4)笹田耕一, 松本行弘, 前田敦司, 並木美太郎: Ruby用仮想マシンYARVにおける並列実行スレッドの実装, 情報処理学会論文誌, Vol.48, No.SIG10(PRO33), pp.1-16 (2007)

研究業績(国際会議)

- 1)K.Sasada, M.Sato, S.Kawahara, N.Kato, M.Yamato, H.Nakajo and M.Namiki: Implementation and Evaluation of a Thread Library for Multithreaded Architecture, The 2003 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'03), Vol.II, pp. 609--615 (2003).
- 2)K.Sasada: YARV: Yet Another RubyVM - Innovating the Ruby Interpreter, Companion to the 20th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2005, pp.158--159 (2005).
- 3) RubyConf 2004 ~ 2006: Oral Presentation

研究業績(その他発表)

- 1)笹田耕一, 佐藤未来子, 河原章二, 加藤義人, 大和仁典, 中條拓伯, 並木美太郎: 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, SACSIS (Symposium on Advanced Computing Systems and Infrastructures) 2003, Vol.2003, No.8, pp. 13--20 (2003).
- 2)笹田耕一: Rubyプログラムを高速に実行するための処理系の開発, 情報処理学会第46回プログラミングシンポジウム報告集, pp. 1--10 (2005).
- 3)笹田耕一, 佐藤未来子, 内倉要, 加藤義人, 大和仁典, 中條拓伯, 並木美太郎: SMTプロセッサにおける細粒度最適化手法の検討, 情報処理学会研究報告(SWoPP青森2004), Vol.2004-ARC-159, pp. 37--42 (2004).
- 4)笹田耕一, 佐藤未来子, 内倉要, 加藤義人, 大和仁典, 中條拓伯, 並木美太郎: SMTプロセッサにおける同期方式の検討, 情報処理学会第154回計算機アーキテクチャ・第101回ハイパフォーマンスコンピューティング合同研究会(HOKKE-2005), Vol.2005-ARC-162, pp. 31--36 (2005).
- 5)笹田耕一, まつもとゆきひろ, 前田敦司, 並木美太郎: Ruby用仮想マシンYARVの実装と評価, 情報処理学会PRO研究会 SWOPP2005 (2005).
- 6)笹田耕一, まつもとゆきひろ, 前田敦司, 並木美太郎: Ruby用仮想マシンYARVにおける並列実行スレッドの実装, 第61回 情報処理学会PRO研究会 豊橋商工会議所 (2006.10).
- 7)笹田耕一, 並木美太郎: Ruby による JavaVM の実装, 情報処理学会第65回全国大会予稿集 (2003).
- 8)笹田耕一: プログラム言語 Ruby におけるメソッドキャッシング手法の検討, 情報処理学会第67回全国大会, Vol.1, pp. 305--306 (2005).

研究業績(その他)

- 解説論文
  - 1)ささだこういち: Ravaで見るJava仮想マシンのしくみ, Java Press, Vol.31, pp. 158--165 (2003).
  - 2)ささだこういち: 日本発LL—Rubyプログラミングステップアップ, Software Design (2006).
  - 3)ささだこういち: Parrotいんたーなる, まるごとPerl, Vol.1 (2006)
- その他
  - 1)ささだこういち: YARVについてLightning Talk, Lightweight Language Weekend (2004.8)
  - 2)ささだ こういち: 日本Rubyの会自主企画「RubyConf2004 に参加して」, 関西オープンソース 2004, 大阪産業創造館 (2004.10)
  - 3)ささだこういち: 日本Rubyの会自主企画「Ruby Hot Topics ～YARV and Rails～」YARV について, Open Source Conference 2005 (2005.3)
  - 4)ささだこういち: Ruby Language Update, Lightweight Language Day and Night (2005.8)
  - 5)ささだこういち: YARVについて(招待講演), Ruby勉強会@関西 (2005.11)
  - 6)笹田耕一:「Rubyを実装するということ」 ESPer2006 未踏集会 (2006.5)
  - 7)笹田耕一: YARVヒッチハイクガイド, 筑波大学大学院 システム情報工学研究科 コンピュータサイエンス専攻 コロキウム (2006.9)
  - 8)笹田耕一:Ruby 1.9実装の現状と今後 日本Ruby会議 2007 (2007.6)
  - 9) まつもとゆきひろ, 笹田耕一:Ruby Language Update Lightweight Language Spirits

研究業績(IPA未踏)

- IPA未踏ソフトウェア創造事業
  - ・2004年6月～2005年2月　IPA未踏ユース採択(PM：筧早稲田大学教授)
    - 「Rubyプログラムを高速に実行するための処理系の開発」
    - スーパークリエータ認定
  - ・2005年6月～2006年2月　IPA未踏ソフトウェア創造事業採択(PM：千葉東工大助教授)
    - 「オブジェクト指向スクリプト言語Rubyの処理系の刷新」
  - ・2006年11月～2007年8月　IPA未踏ソフトウェア創造事業採択(PM：千葉東工大助教授)
    - 「Ruby用仮想マシンYARVの完成度向上」

研究業績(授賞等)

- 2003年 5月 SACSIS 2003 優秀学生論文賞受賞 「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装と評価」　笹田 耕一, 佐藤 未来子, 河原 章二, 加藤 義人, 大和 仁典, 中條 拓伯, 並木 美太郎
- 2004年 3月 東京農工大学　阿刀田基金 受給
- 2005年 5月 IPA 2004年度未踏ソフトウェア創造事業(未踏ユース)スーパークリエータ認定
- 2005年 5月 SACSIS 2005 優秀若手研究賞(共著)「SMTプロセッサにおけるスレッドスケジューラの開発」　内倉要, 佐藤未来子, 笹田耕一, 加藤義人, 大和仁典, 中條拓伯, 並木美太郎
- 2005年5月SACSIS2005 最優秀独創的研究アイデア賞(共著)「SMTプロセッサのFPGAへの実装と評価」　加藤義人, 大和仁典, 小笠原嘉泰,佐藤未来子, 笹田耕一, 内倉要, 中條拓伯, 並木美太郎
- 2007年10月IPA OSS貢献者賞受賞

研究業績(その他の活動)

- 2003年4月～2006年3月 東京農工大学工学部　ティーチングアシスタント
- 2003年4月～2006年3月 東京農工大学工学部　プログラミング相談員
- 2004年7月～2005年3月 東京農工大学大学院　リサーチ・アシスタント
- 2004年8月 2日～7日 文部科学省ITスクール　チューター
- 2004年8月～ 日本Rubyの会 理事(会計)
- 2005年1月～2007年1月 情報処理学会 情報科学若手の会 幹事
- 2005年11月～ 日本Rubyカンファレンス2006 実行委員
- 2006年9月～ 日本Ruby会議2007 実行委員
- 2007年1月～ 情報処理学会夏のプログラミングシンポジウム 幹事
- 2007年4月～ 情報処理学会SIGOS 運営委員
- 2007年6月～ ソフトウェア科学会　PPL2008 プログラム委員

予備資料

設計：YARV命令セット

- 現時点では 52 命令
  - できるだけシンプルに
    → VM生成系によって自動的に複雑・高速な命令に
- プリミティブ型がないため、数値演算命令無し

```
a = recv.method(b)
```

→

```
getlocal 2 # push recv
getlocal 3 # push b
send :method, 1
setlocal 1 #
```

設計：例外処理

- Rubyレベルでは表引きで処理
  - 例外処理部分突入時にはコスト無し
  - Ruby スタックの巻き戻しを行い探索
- setjmp/longjmp による例外処理機構と併用
  - VM関数のネストを自然に表現可能
  - マシンスタックの巻き戻しを実現

設計：例外処理

<u>YARV実行のC関数コールグラフ</u>

VM handler 関数(setjmp)
    VM関数
        C関数
            VM handler 関数(setjmp)
                VM 関数
                C関数

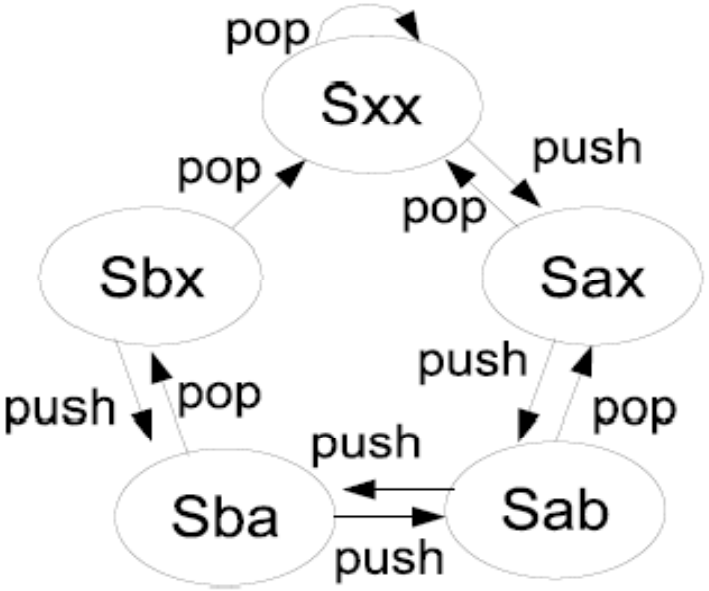
表を引いて例外ハンドラ探索

見つからなければ longjmp

例外を発生するには longjmp

最適化：静的スタックキャッシング

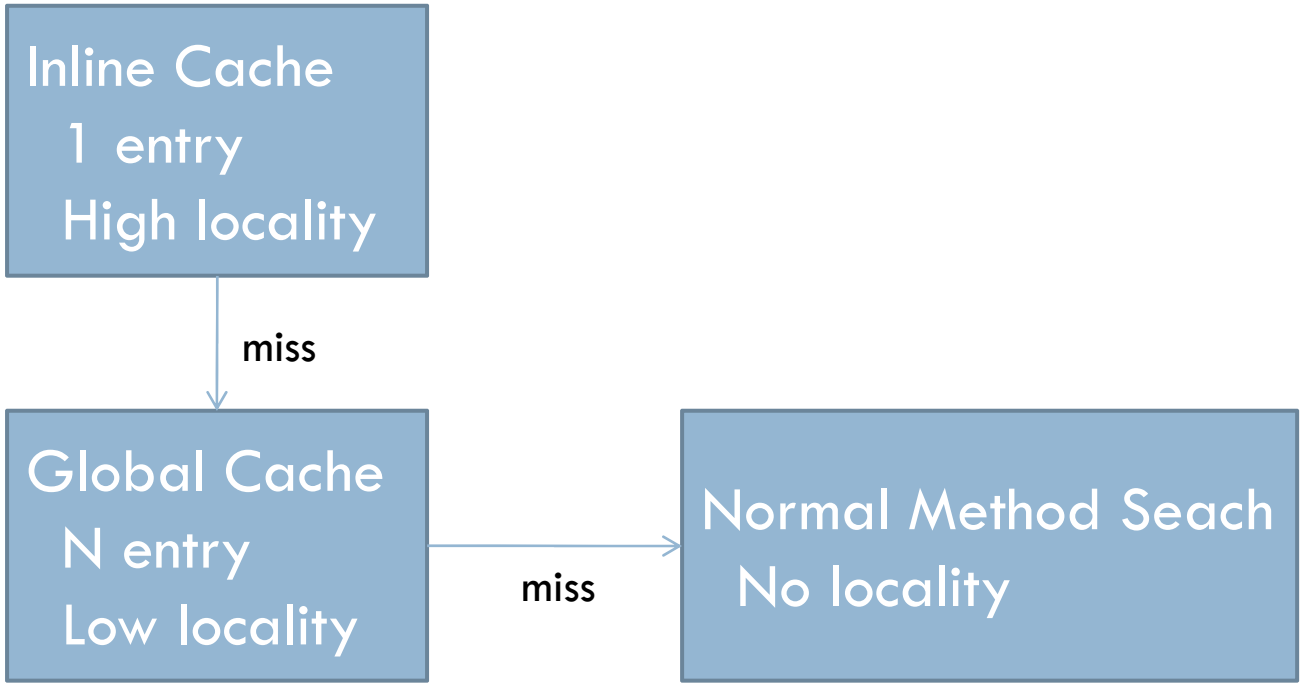
- 5状態、2レジスタの静的スタックキャッシング（SC）
  - スタックトップの2値をレジスタに格納
  - 5状態なので、同じ命令につき5命令必要
  - putobject（スタックにひとつ積む）
    - putobject_xx_ax
    - putobject_ax_ab
    - putobject_bx_ba
    - putobject_ab_ba
    - putobject_ba_ab



最適化：インラインキャッシュ

- インラインメソッドキャッシュ
  - メソッド検索結果を命令列にキャッシュ
- インラインキャッシュ
  - 検索結果のRuby定数値を命令列にキャッシュ
- キャッシュ時にVM状態カウンタも同時に格納
  - 現在の値と比較し、キャッシュの状態を確認
  - 定義・再定義が発生したときインクリメント

YARVのメソッド探索



VM生成系：命令記述と生成例

- 値の宣言と、ロジックの記述（C言語）
  - （1）命令オペランド（2）スタックオペランド（3）スタックへプッシ

```
DEFINE_INSN
mult_plusConst
(int c)          // 命令オペランド定義
(int x, int y) // スタックオペランド定義
(int ans)        // 命令の返り値定義
{
  // C言語による命令ロジック記述部分
  ans = x * y + c;
}
```

```
mult_plusC  n:{
  int c = *(PC+1);
  int y = *(SP-1);
  int x = *(SP-2);
  int ans;
  PC += 2;
  SP -= 2;
{
  ans = x * y + c;
}
  *(SP) = ans; SP += 1;
  goto **PC;}
```

命令オペランドフェッチ

スタックオペランドフェッチ

PC/SP設定

結果をpush

VM生成系：オペランド融合命令の生成

- 融合命令オペランドを #define で定義
- ロジック部分は変更無し

mult_plusConst 5

```
mult_plusConst
{
  #define c 5
  VALUE y = *(SP-1);
  VALUE x = *(SP-2);
  VALUE ans;
  PC += 1;
  SP -= 2;
{
  ans = x * y + c;
}
  #undef c
  *(SP) = ans; SP += 1;
  goto **PC;
}
```

オペランド定義

VM生成系：命令融合した命令の生成

dup + mult_plusConst

```
UNIFIED_dup_mult_plusConst:
{
  int c_1 = *(PC+1);
  int v_0 = *(SC-1);
  int ans;
  PC += 3;
  SP -= 1;
{ // dup
  #define v v_0
  #define v1 v1_0
  #define v2 v2_0
  v1 = v2 = v;
  #undef v
  #undef v1
  #undef v2
} // cont ->
```

利用する値を
適切に定義

命令を連結

```
// -> cont
{ // mult_plusConst
  #define c c_1
  #define x v1_0
  #define y v2_0
  ans = x * y + c;
  #undef c
  #undef x
  #undef y
}
  *(SP) = ans; SP += 1;
  goto **PC;
}
```

VM生成系：スタックキャッシング用命令生成

- プッシュ・ポップ処理をSC用レジスタアクセスに
- コンパイラは命令列を状態遷移に則り変換

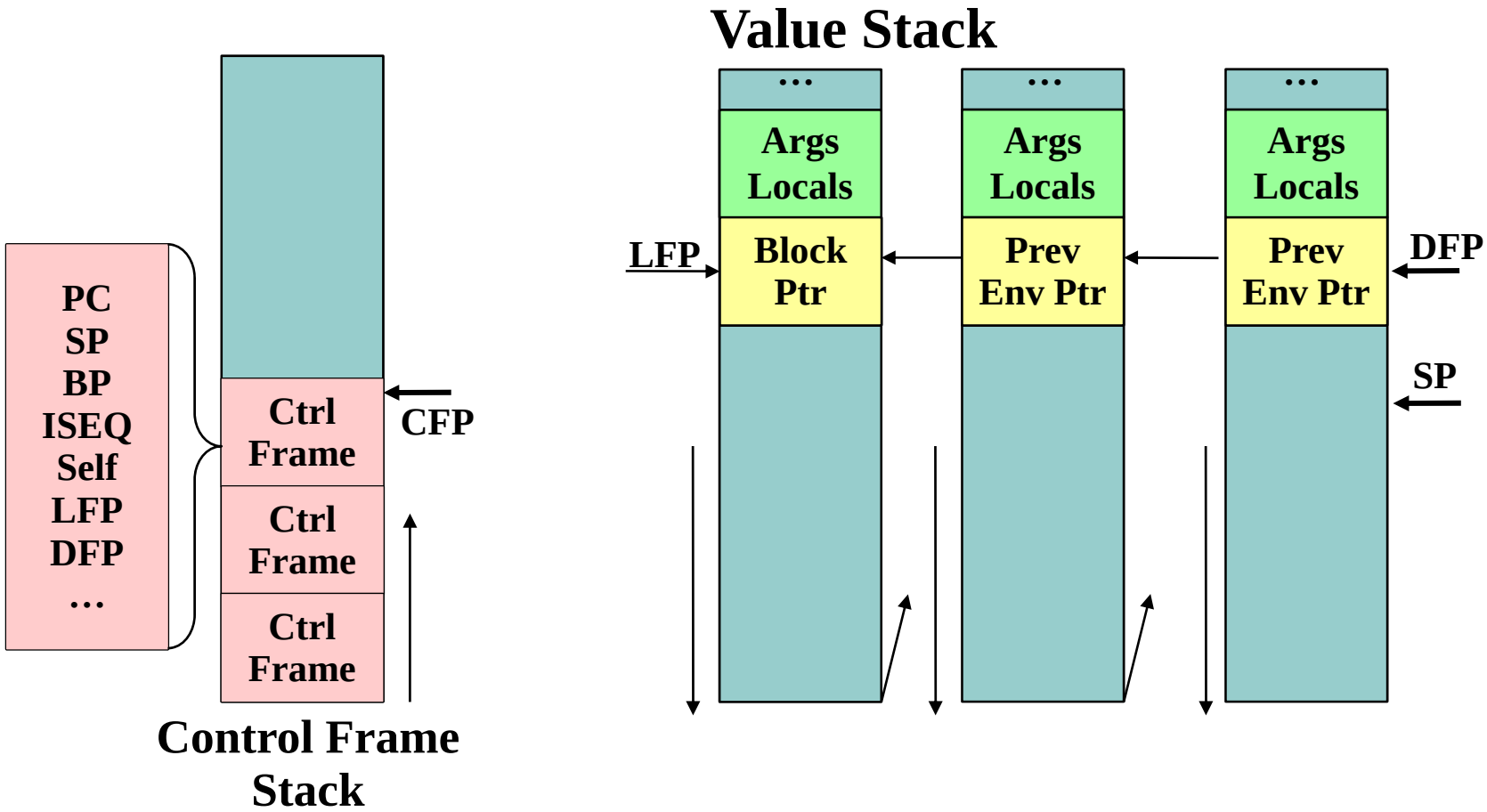
mult_plusConst_ab_ax

```
mult_plusConst_
{
  int c = *(PC+1);
  int y = SC_regB;
  int x = SC_regA;
  int ans;
  PC += 2;
{
  ans = x * y + c;
}
  SC_regA = ans;
  goto **PC;
}
```

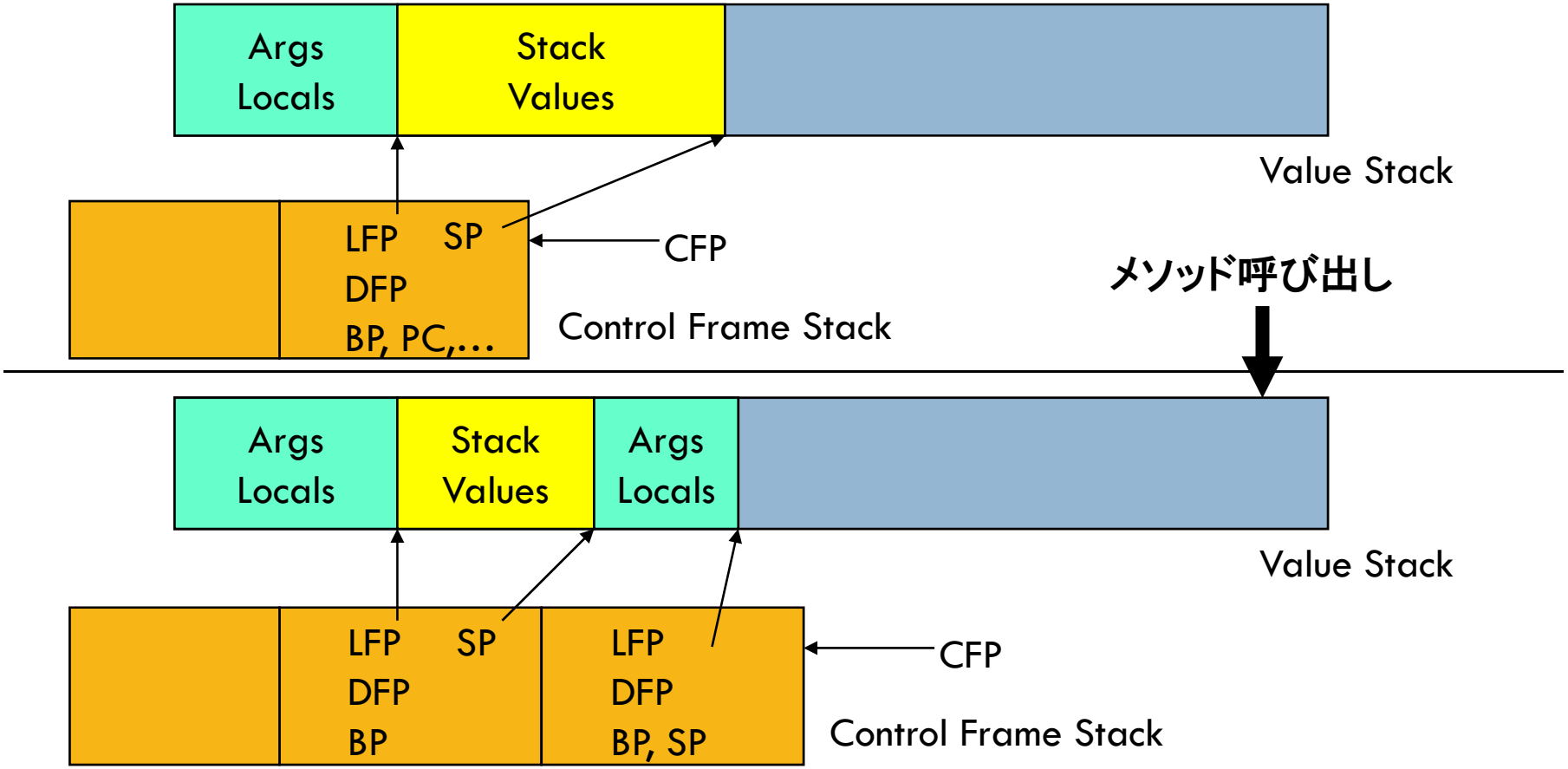
SCレジスタアクセス（pop）

SCレジスタアクセス（push）

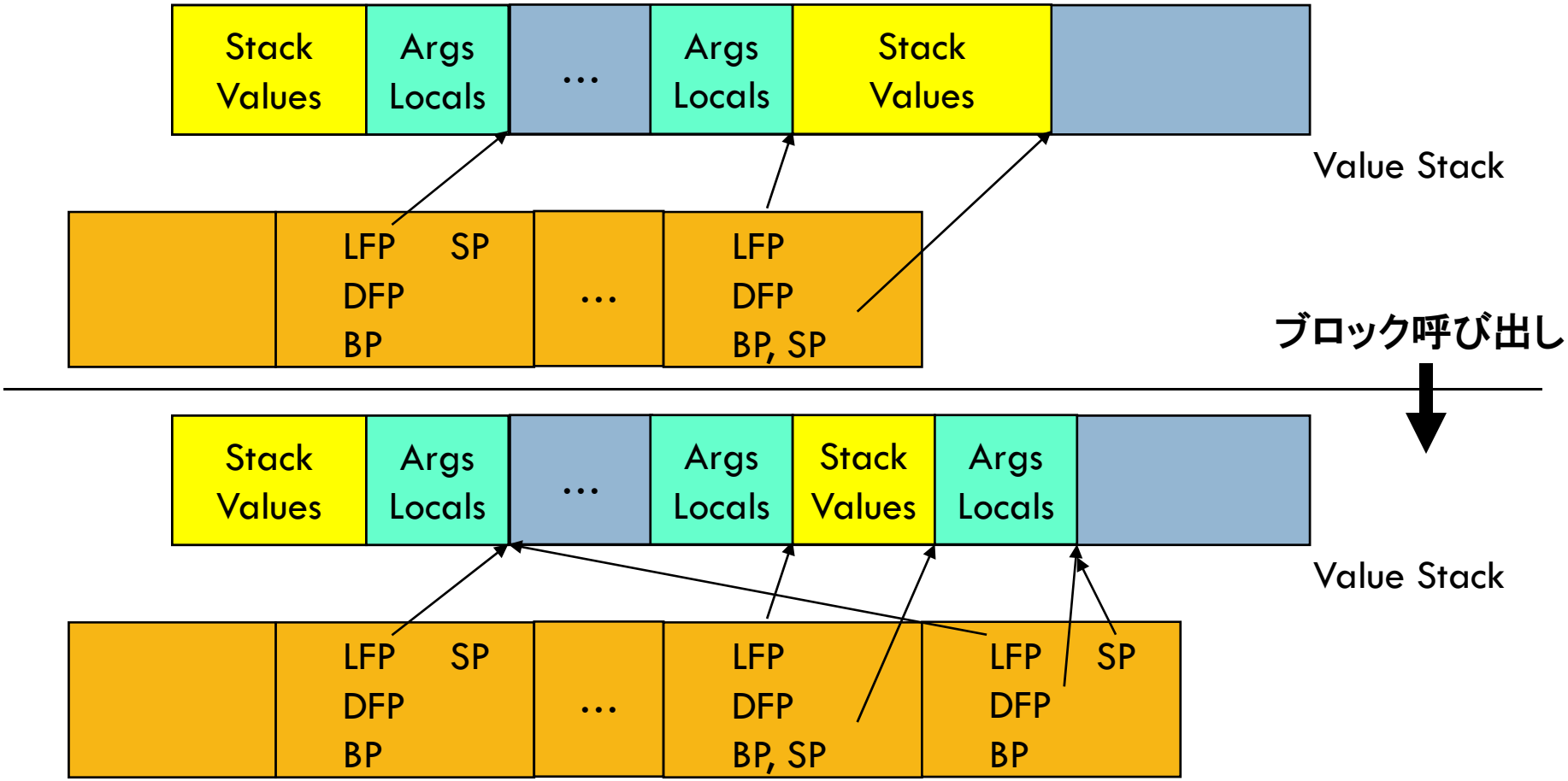
設計：スタックフレーム・環境



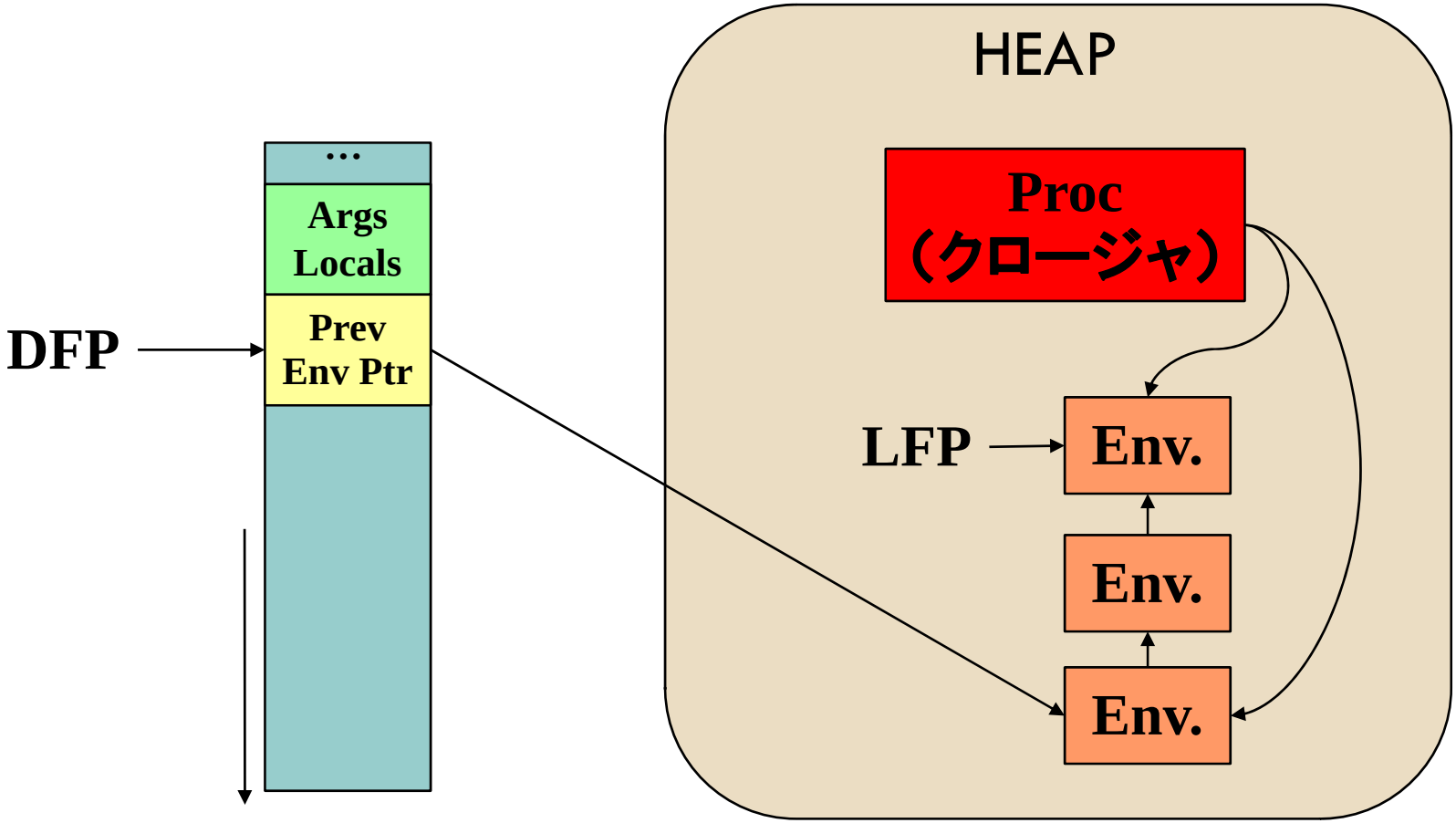
設計：スタックフレーム（メソッド呼び出し）



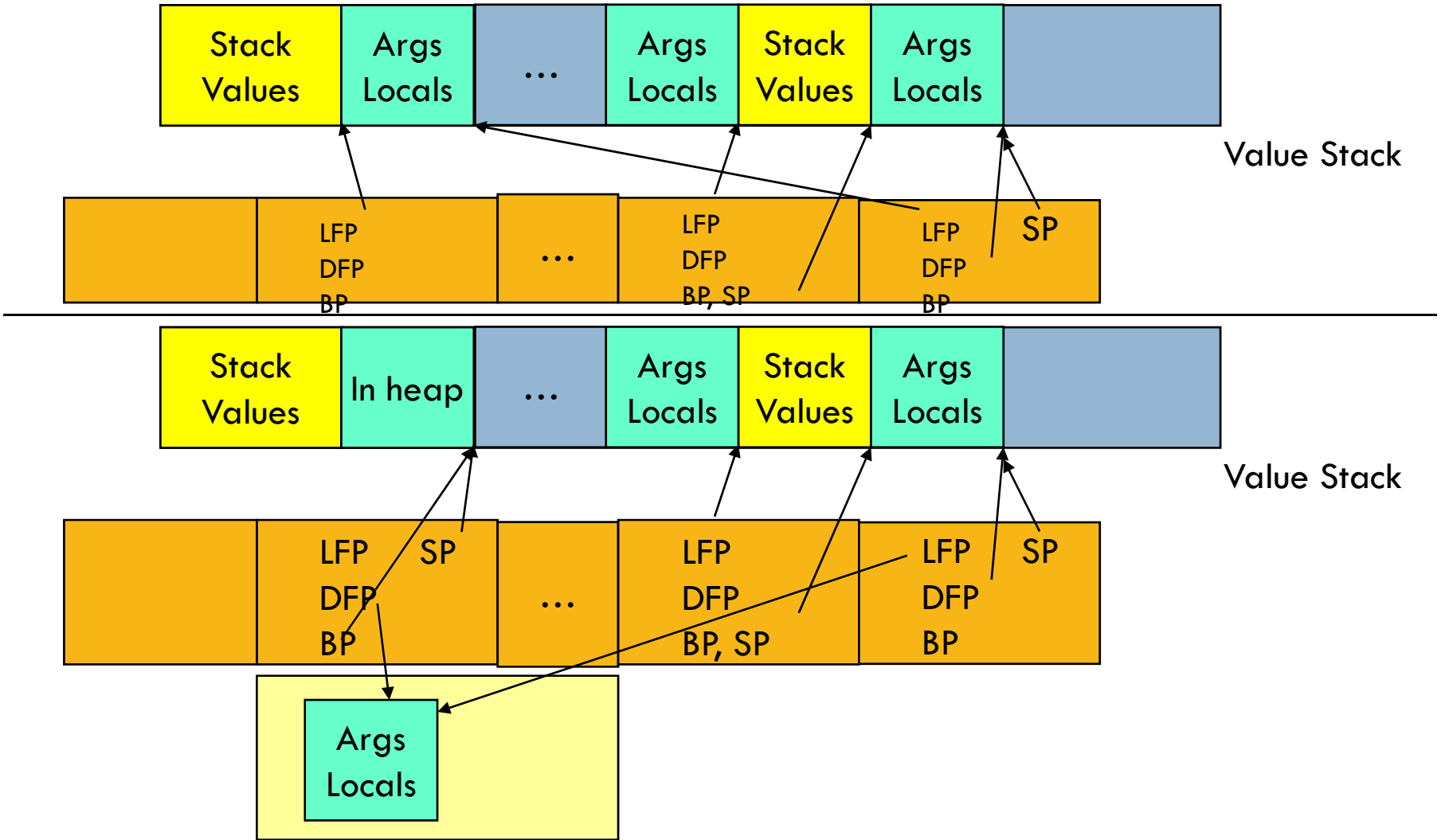
設計：スタックフレーム（ブロック呼び出し）



設計：スタックフレーム・環境（クロージャ）



設計：スタックフレーム（with ヒープ）



設計：Rubyスレッドのネイティブスレッド対応 Rubyスレッドへの割り込み

- Rubyスレッドは割り込み可能
  - 他スレッドに例外可能
    - Timeout 処理に利用
      特に I/O のタイムアウト
  - シグナルハンドラの実行

```
t = Thread.new{
  #...
}
t.raise(exception)
```

- ネイティブスレッドには「割り込み」インターフェースは無し

設計：Rubyスレッドのネイティブスレッド対応 Rubyスレッドへの割り込み(cont.)

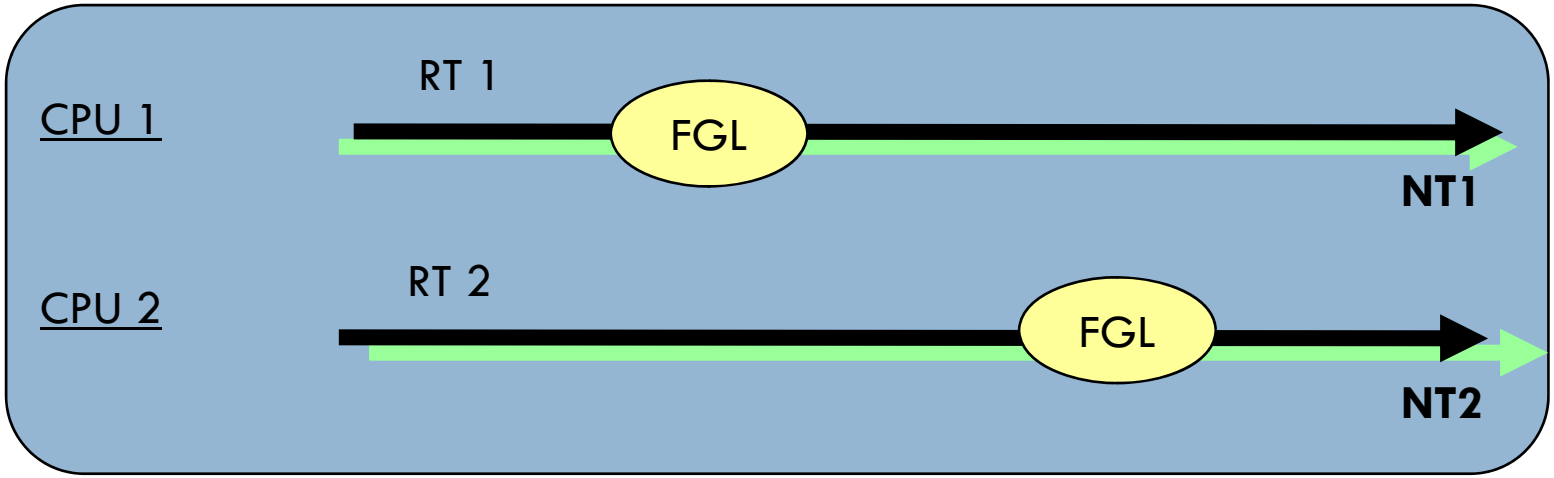
- (1) 割り込みフラグのポーリング
  - VM命令のいくつかの箇所でチェック
- (2) ブロック状態になる際には<u>ブロック解除関数</u>を登録
  - ポーリングしない処理部分（selectシステムコール発行時など）でも割り込みに対応
  - 例：selectシステムコールを中断させるブロック解除→当該スレッドにシグナル送信（select は EINTR で中断）

設計：Rubyスレッドのネイティブスレッド対応 管理スレッドの用意

- UNIXでのシグナル受信
  - Pthreadではどの NTに配送されるか不明→シグナル受信専用の管理スレッド
  - 他のNTはシグナルマスクで受信不可
  - 受信したらmainスレッドへRubyスレッドへの割り込み機能を利用して割り込み

検討：並列実行における排他制御
(a) 細粒度ロックのみ

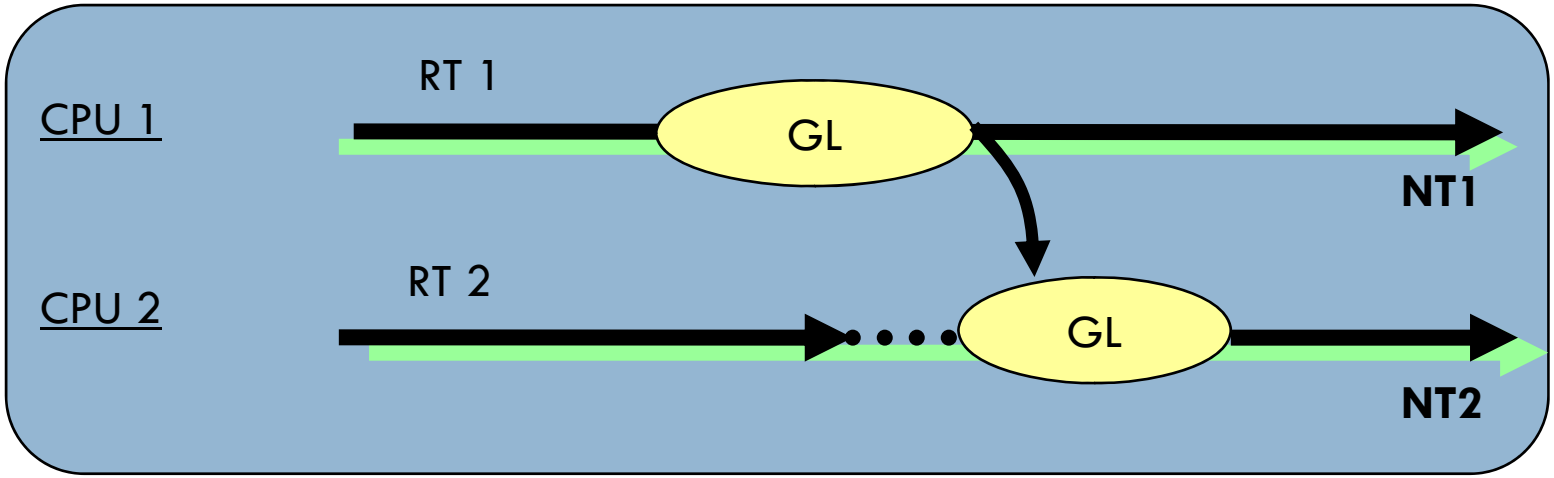
- ○ 十分な並列性を発揮(「普通」のアプローチ)
- × 全ての既存のコードに対し、FGLなどによるスレッドセーフ(TS)コードへ変更が必須
  →「既存の豊富なRubyライブラリ」が利用不可



FGL: Fine-Grained Lock

検討：並列実行における排他制御
(b) 細粒度ロック＋ジャイアントロック（GL）

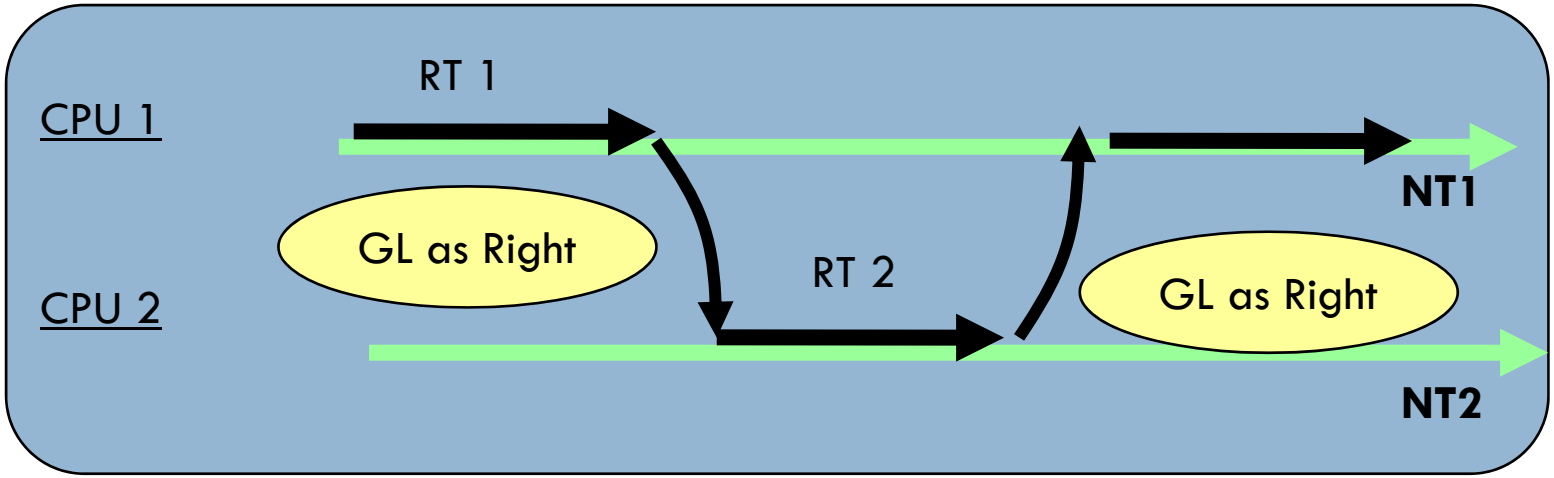
- TSでないCメソッドコールはGLで保護
  - ○ 既存のコードが利用可能（並列実行不可）
  - ○ とりあえず今までどおりCメソッド記述可能
  - ○ 必要に応じてFGLによるTSコードへ移行可



GL: Giant Lock

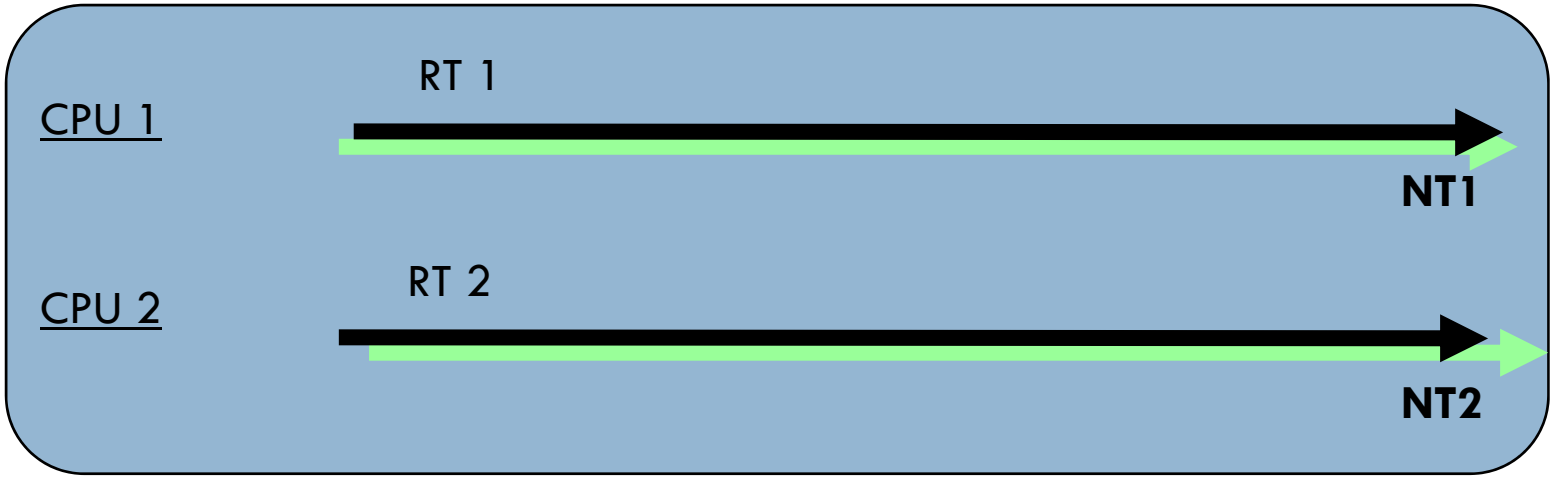
検討：並列実行における排他制御
(c) ジャイアントロックを持つRTのみ実行

- GLを実行権として扱い、持つRTのみ実行
  - ○ 排他制御不要
  - × 並列実行不可能
  - Python などで利用



検討：並列実行における排他制御
(d) 排他制御しない（Rubyプログラマ任せ）

- ○ 排他制御不要
- × 処理系不正処理（SEGV等）発生
- 低級言語的アプローチ



検討：
排他制御の各方式のまとめ

	スケーラビリティ（台数効果）	逐次処理の性能	既存コードの再利用	やさしさ
(a)	◎	○	×	○
(b)	○	○	○	○
(c)	×	◎	○	○
(d)	◎	◎	○	×

性能追求だけなら

Single Core用

Hacker用？

設計：Rubyスレッドの並列実行 スレッド情報へのポインタをTLSに

- スレッド情報へのポインタをネイティブスレッドが提供するTLSに格納
  - TLS: Thread Local Storage
    - GCC 拡張
    - Windows VC++ 拡張
    - pthread_getspecific の利用

設計：Rubyスレッドの並列実行
非TSなコードのための適切なGL制御

- 非TSなコード == 既存のCメソッド実装
  - メソッド起動時にGL獲得
- TSなコード ==
  対処したCメソッド or Ruby メソッド
  - GL獲得なしで起動可能
- CメソッドはRubyメソッド呼び出し（逆も）可能
  - メソッド呼び出し時、GL獲得状態によって処理分岐
- 例外発生時のGL獲得情報に注意

設計：Rubyスレッドの並列実行
非TSなコードのための適切なGL制御

Machine level Call Graph:

top

VM_eval_func (Ruby Method(s))

notTS C Method

VM_eval_func (Ruby Method(s))

TS C Method

notTS C Method

Locked w/GL

Locked w/GL

例外発生時の
大域ジャンプ例

メソッド呼び出し

設計：Rubyスレッドの並列実行 ロックコストを下げるための工夫(cont.)

- ロックを不要に
  - (1) 単一スレッド実行時にはGL無視
  - (2) スレッドローカルヒープの用意
    - オブジェクトアロケーション時、ロック不要
  - (3) ロック不要なメソッドキャッシュ
    - キャッシュヒットすれば余計なコスト不要
    - キャッシュミス時、コスト増

設計：Rubyスレッドの並列実行 ロックコストを下げるための工夫(cont.)

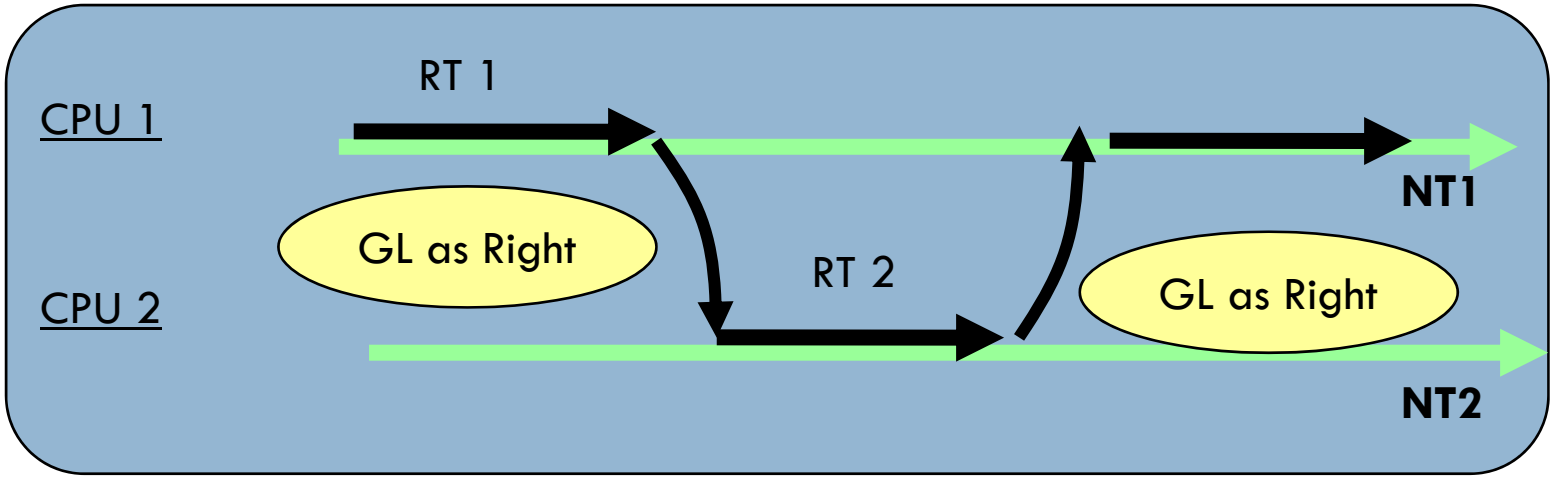
- (5) 利用CPUの制限
  - 競合を繰り返すRubyスレッドの実行CPUを制限
  - NPTLのpthread_setaffinity_npを利用
    - Windows では SetThreadAffinityMask
  - 一定時間過ぎたら制限を解除

    - ちょっと後ろ向きの工夫（v.s. TS なコードを増やす）

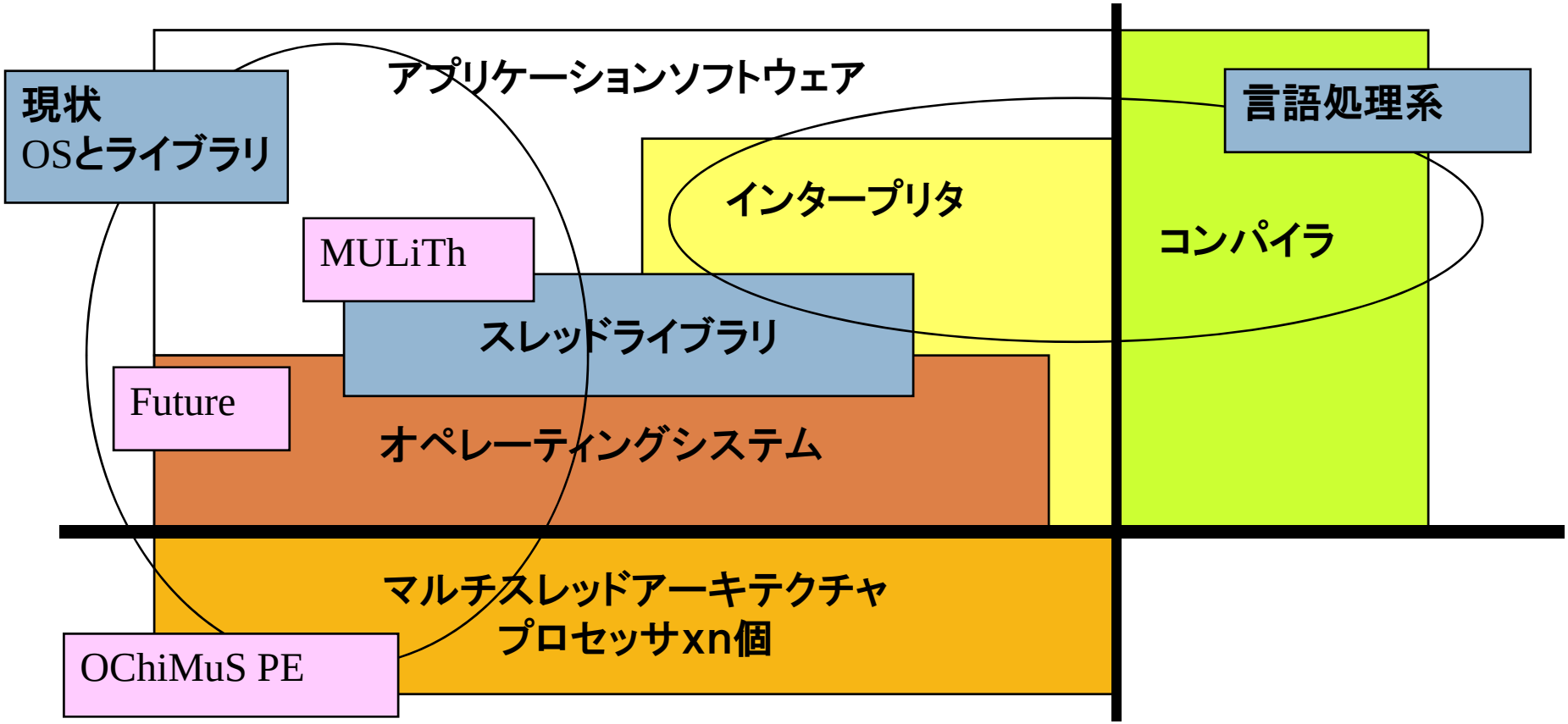
検討：並列実行における排他制御
(c) ジャイアントロックを持つRTのみ実行

- GLを実行権として扱い、持つRTのみ実行
- ○ 排他制御不要
- × 並列実行不可能

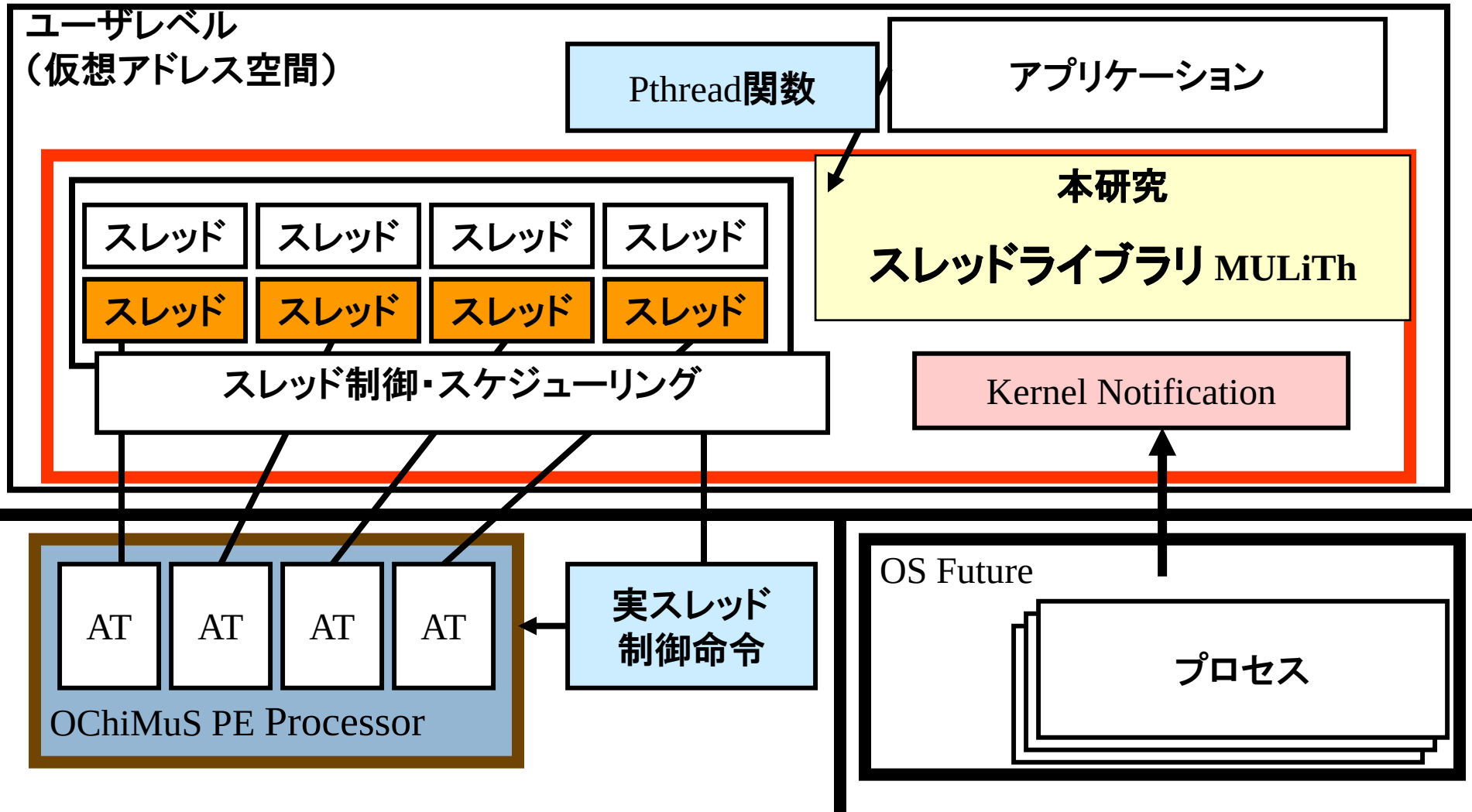
YARV (NoLock)



目標とするシステム

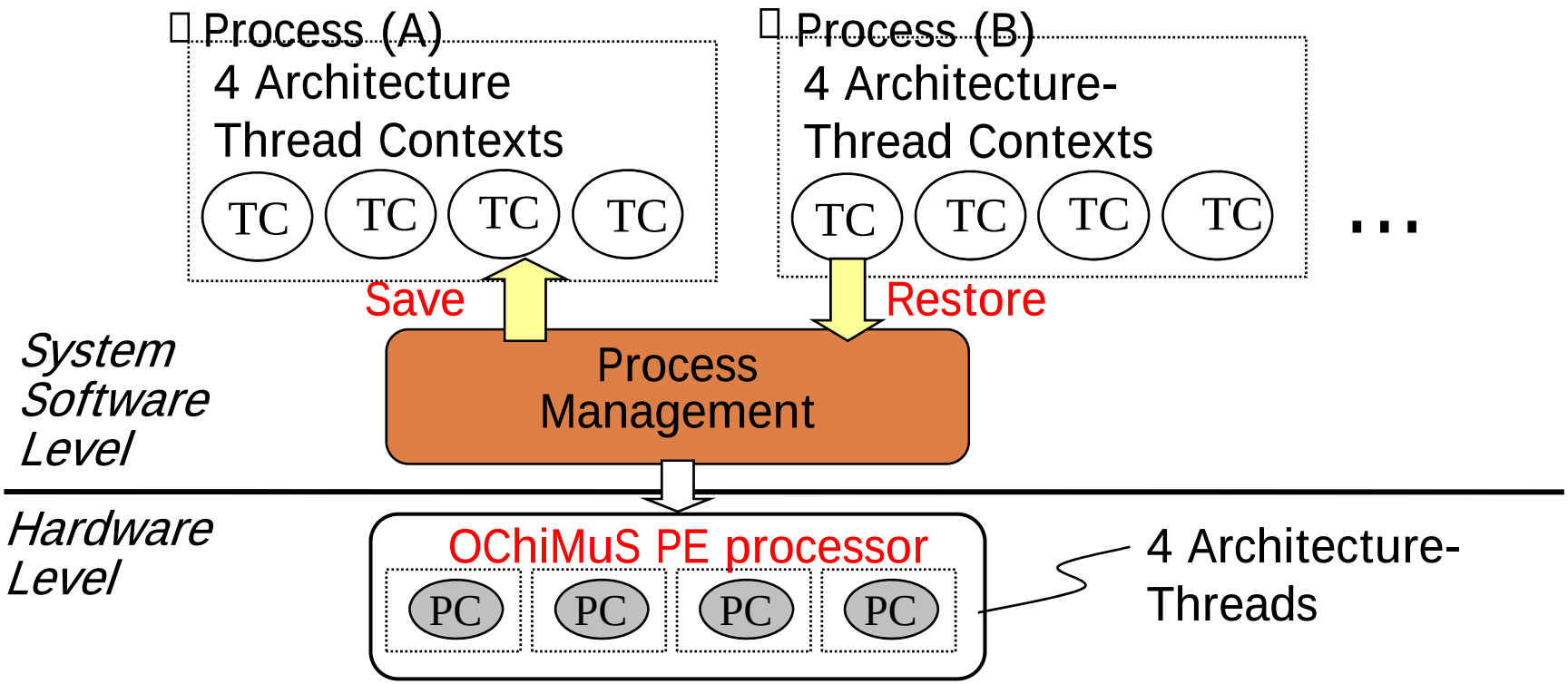

現状
OSとライブラリ
アプリケーションソフトウェア
言語処理系
インタープリタ
コンパイラ
MULiTh
スレッドライブラリ
Future
オペレーティングシステム
マルチスレッドアーキテクチャ
プロセッサxn個
OChiMuS PE

CPU・OS・ライブラリの全体像



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



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

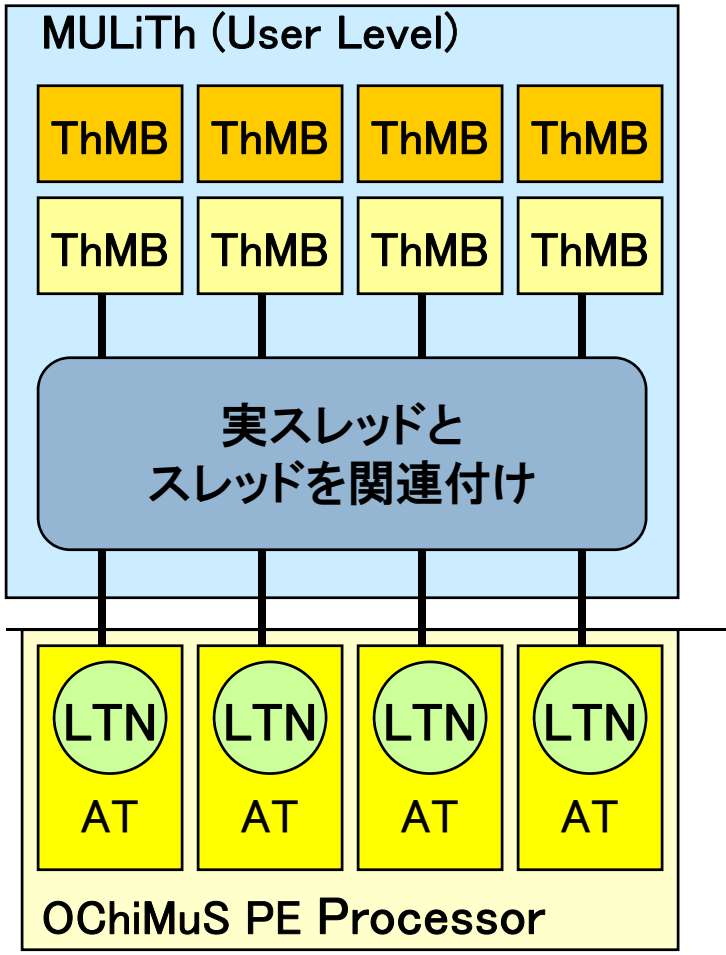
  ⇒ ThMBの先頭アドレス

  ↓

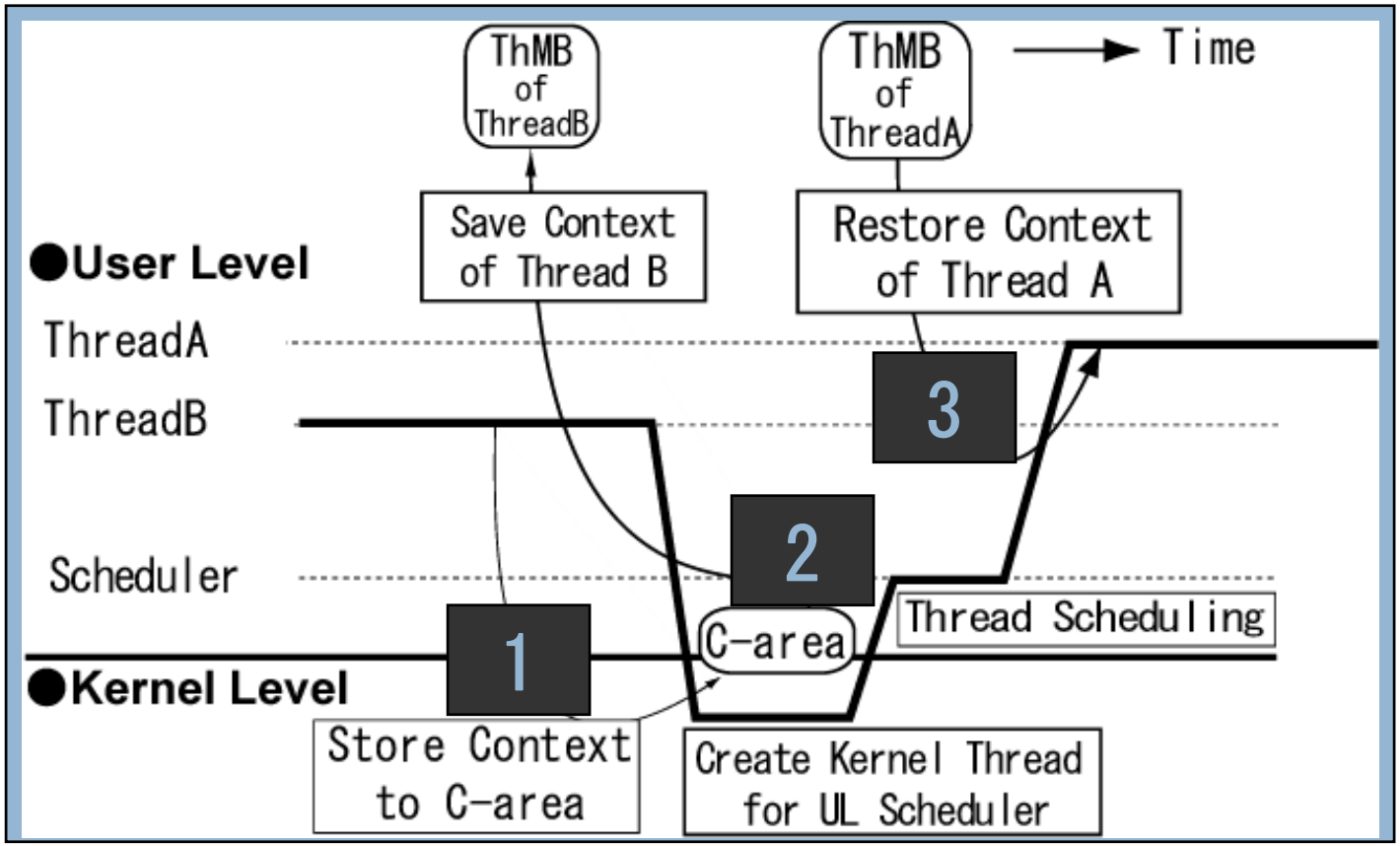
  LTNとして使用

- 実行中スレッドは

  プロセッサが把握



従来型事象通知

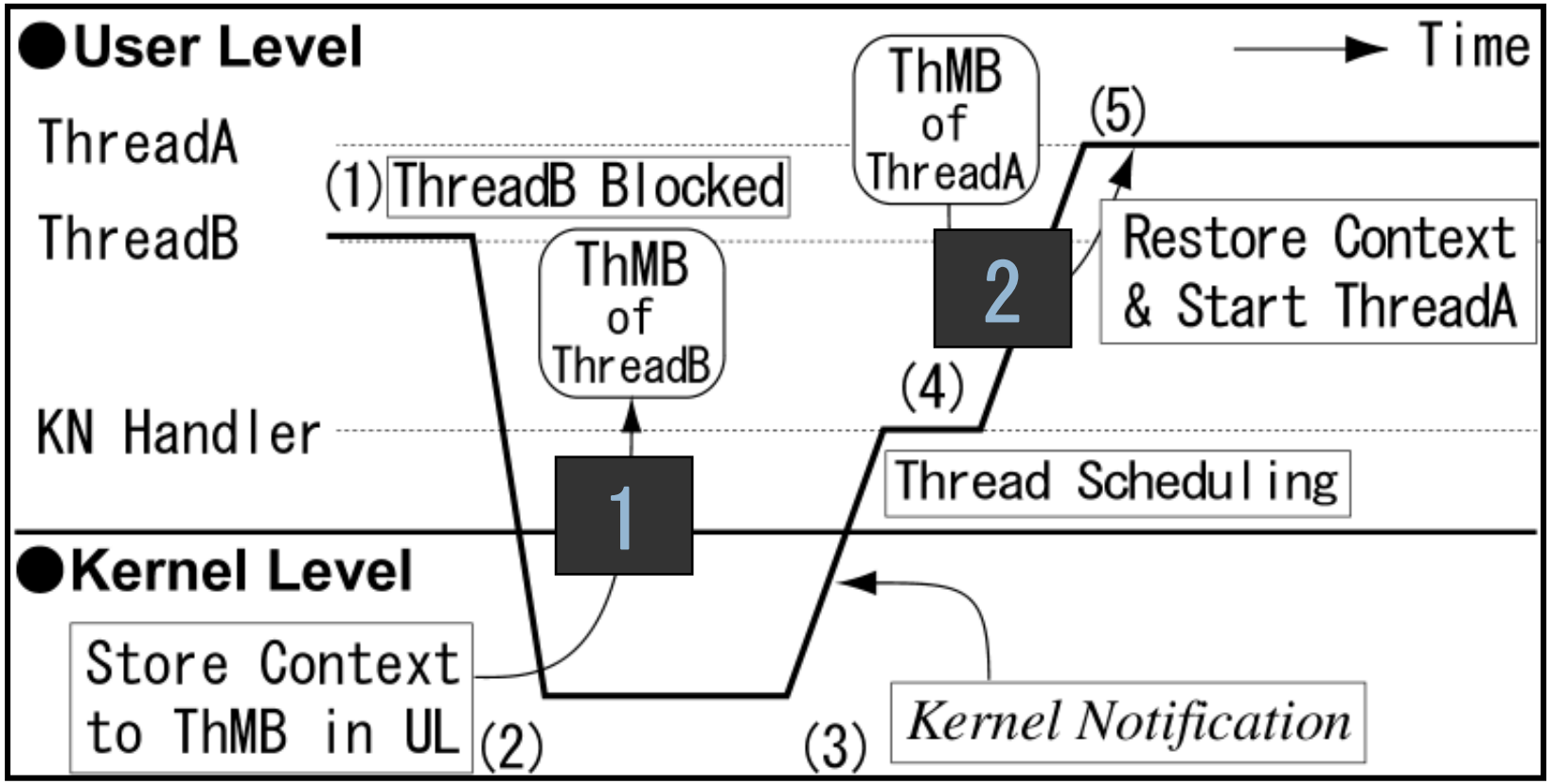


3回のコンテキストコピー ＋ Kernel Thread 生成

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

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

評価
スレッド切り替え時間