YARV

Past, Present and Future

Koichi “ko1” Sasada

ささだこういち

ko1 at atdot dot net

RUBY MEETS VM

Koichi Sasada
ko1 at atdot dot net

Point of This Presentation

Merging YARV is not a goal,
but a start

YARV: Yet Another RubyVM

Notice

- I can’t speak English well, so I write down all things what I want to say.
  - Do you get ready for opera glasses?
  - Unfortunately, some slides are written in Japanese
- You can ask questions with

  Japanese, C, Ruby, …, or slow/short English.
- “How to impl. Ruby”, not “How to use Ruby”
- “x50” is too big mouth
  - Maybe x20

Self Introduction

- Koichi (given name) Sasada (family name)
  - ささだ（family name) こういち (given name)
  - 笹田 (family name) 耕一 (given name)
- Lecturer @ University of Tokyo (Feb. 2008-)
- Only VM developer
  - Don’t have compatibility with Matz
  - Please call me “ko-i-chi”

Agenda

- History of YARV
- Advanced VM Topics
  - Performance
    - Parallel Thread Execution
    - Embedding Float Value
    - JIT Compiler
    - Pre-Compiler
      - “Ruby to Compiled file” Compiler
      - “Ruby to C” Compiler
  - New Feature
    - Multi-VM Creation
    - Customizable Ruby Core
    - Debug/Profile support feature
- Summary

History of YARV

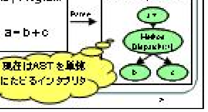
- 4 Years
  - **1, Jan 2004 Project Start**
  - 2004-2005 VM Core, Optimization
    - Supported by MITO youth Project (IPA)
  - 2005-2006 Thread, etc
    - Supported by MITO Project (IPA)
    - 1, Apr 2006 Got a Job (Assistant on U-Tokyo)
  - 2006-2007 etc, etc
    - Supported by MITO Project (IPA)
    - 25, Dec 2007 Got a Ph.D
  - **25, Dec 2007 (GMT) 1.9 Release**

FYI

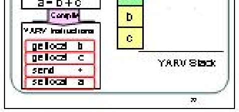
- Ruby 2.0 – since 2003 3/31
- Perl 6 – since 2003 4/1

YARV Policy

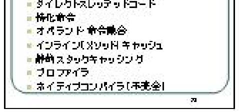
- Performance
  - Speed, Speed, Speed
  - Applied many many many optimization Tech.
- Compatibility
  - C extension API
  - Not language compatibility
- Auto-generation
  - VM description to Concrete VM source code



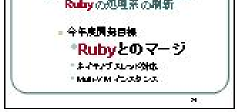
21



22

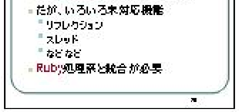

- ダイレクトスレッデッドコード
- 特化命令
- オペランド・命令融合
- インライン(メソッド)キャッシュ
- 静的スタックキャッシング
- プロファイラ
- ネイティブコンパイラ(不完全)

23

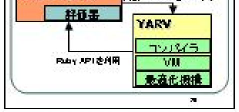

Rubyの処理系の刷新

- 今年度開発目標
  - Rubyとのマージ
  - ネイティブスレッド対応
  - Multi-VM インスタンス

24

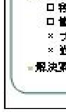

- だが、いろいろ未対応機能
  - リフレクション
  - スレッド
  - などなど
- Ruby処理系と統合が必要

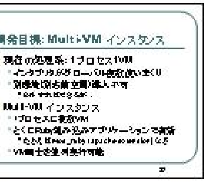
25



26

27



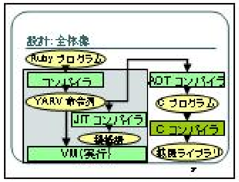

開発目標: Multi-VM インスタンス

- 現在の処理系: 1プロセス1VM
  - インタプリタがグローバル変数使いまくり
  - 別環境(別名前空間)導入不可
- Multi-VM インスタンス
  - 1プロセスに複数VM
  - とくにRuby組み込みアプリケーションで有効
    - たとえばmod_ruby (apache extension) など
  - VM同士を並列実行可能

31

開発目標: Multi-VM インスタンス イメージ図

32


設計: 全体像

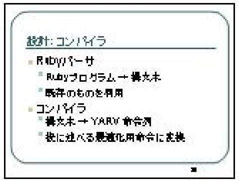
33

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変換・最適化命令を C
- プリミティブ型がないため、数値演算命令無し

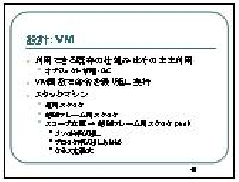
```
a = recv.method(b)
getlocal 2 # push recv
getlocal 1 # push b
send method, 1
setlocal 1 #
```

34


設計: コンパイラ

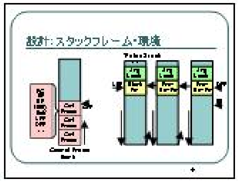
- Rubyパーサ
  - Rubyプログラム → 構文木
  - 既存のものを利用
- コンパイラ
  - 構文木 → YARV 命令列
  - 後に述べる最適化用命令に変換

35

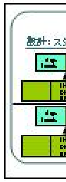

設計: VM

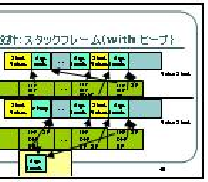
- 利用できる既存の仕組みはそのまま利用
  - オブジェクト管理・GC
- VM関数で命令を繰り返し実行
- スタックマシン

36

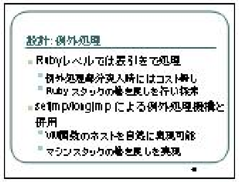

設計: スタックフレーム・環境

37



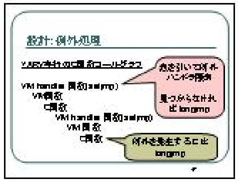

設計: スタックフレーム(with ヒープ)

41


設計: 例外処理

- Rubyレベルでは表引きで処理
  - 例外処理範囲突入時にはコスト無し
  - Ruby スタックの巻き戻しを行い検索
- 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実現可能
  - マシンスタックの巻き戻しを実現

42

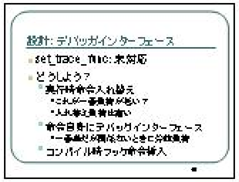

設計: 例外処理

43

設計: データ構造

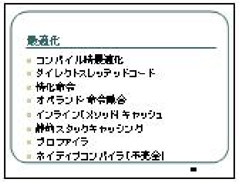
- Rubyマージのために刷新
  - とくに評価器
  - グローバル変数の排除(Thread へ格納)
  - スタックフレームのスリム化

44


設計: デバッガインターフェース

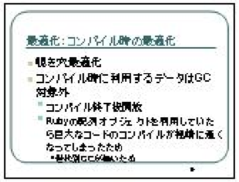
- set_trace_func: 未対応
- どうしよう?
  - 実行時命令入れ替え
  - 命令自体にデバッグインターフェース
  - コンパイル時フック命令挿入

45


最適化

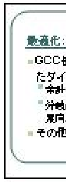
- コンパイル時最適化
- ダイレクトスレッデッドコード
- 特化命令
- オペランド・命令融合
- インライン(メソッド)キャッシュ
- 静的スタックキャッシング
- プロファイラ
- ネイティブコンパイラ(不完全)

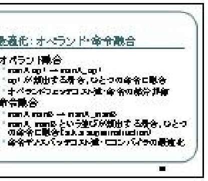
46


最適化: コンパイル時の最適化

- 覗き穴最適化
- コンパイル時に利用するデータはGC対象外
  - コンパイル終了後開放
  - Rubyの既存オブジェクトを利用していたら巨大なコードのコンパイルが極端に遅くなってしまったため
    - 世代別GCが欲しかった

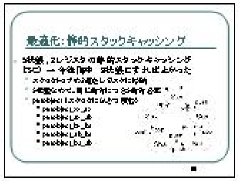
47



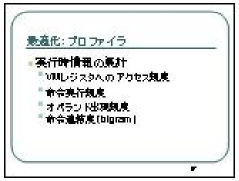

最適化: オペランド・命令融合

- 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
  - オペランドフェッチコスト減・命令の動的特化
- 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)
  - 命令ディスパッチコスト減・Cコンパイラでの最適化

51

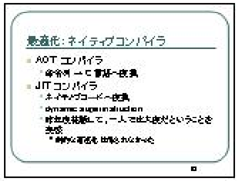

最適化: 静的スタックキャッシング

52


最適化: プロファイラ

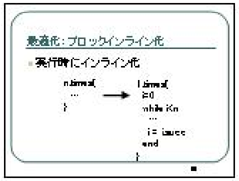
- 実行時情報の集計
  - VMレジスタへのアクセス頻度
  - 命令実行頻度
  - オペランド出現頻度
  - 命令連続度 (bigram)

53


最適化: ネイティブコンパイラ

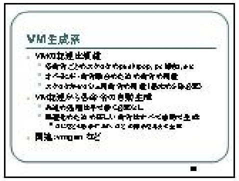
- AOT コンパイラ
  - 命令列 → C 言語へ変換
- JIT コンパイラ
  - ネイティブコードへ変換
  - dynamic superinstruction

54


最適化: ブロックインライン化

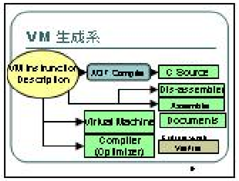
- 実行時にインライン化

55


VM生成系

- VMの記述を簡潔
- VM記述から各種命令の自動生成
- 関連: vmgen など

56

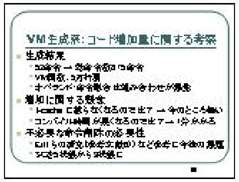

VM 生成系

57

VM生成系: スタックキャッシング用命令生成

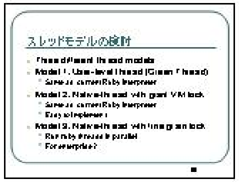
- プッシュ・ポップ処理をSC用レジスタアクセスに
- コンパイラは命令列を状態遷移に則り変換

61


VM生成系: コード増加量に関する考察

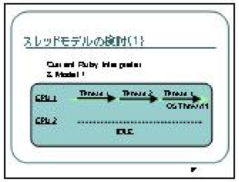
- 生成結果
- 増加に関する懸念
- 不必要な命令削除の必要性

62


スレッドモデルの検討

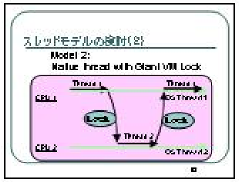
- Thread Model
  - Model 1: User-level thread (Green Thread)
    - Same as current Ruby Interpreter
  - Model 2: Native thread with giant VM lock
    - Same as current Ruby Interpreter
    - Easy to implement
  - Model 3: Native thread with fine grain lock
    - Run ruby threads in parallel
    - For enterprise?

63


スレッドモデルの検討(1)

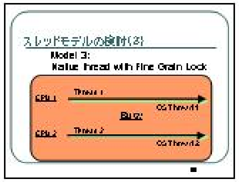
Current Ruby Interpreter & Model 1

64


スレッドモデルの検討(2)

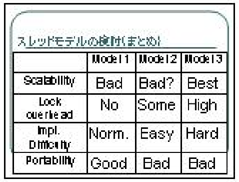
Model 2: Native thread with Giant VM Lock

65


スレッドモデルの検討(3)

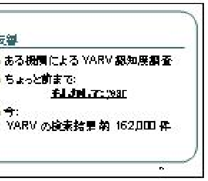
Model 3: Native thread with Fine Grain Lock

66


スレッドモデルの検討(まとめ)

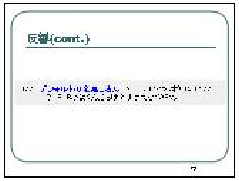
	Model 1	Model 2	Model 3
Scalability	Bad	Bad?	Best
Lock overhead	No	Some	High
Impl. Difficulty	Norm.	Easy	Hard
Portability	Good	Bad	Bad

67

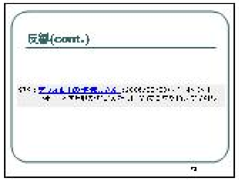

反響

- ある機関による YARV 認知度調査
- ちょっと前まで:
  - もしかして: year
- 今:
  - YARV の検索結果 約 162,000 件

71


反響(cont.)

72

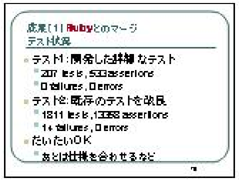

反響(cont.)

73

成果(1) Rubyとのマージ

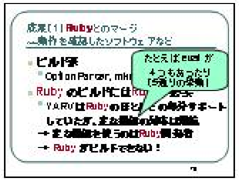
- 現在のRubyの機能を「ほぼ」網羅
- YARV 0.4.0 リリース (2006/02/22)
  - ビルド・インストール可能
  - Ruby 1.9 (2006/02/14) 版に追従

74


成果(1) Rubyとのマージ
テスト状況

- テスト1: 開発した詳細なテスト
  - 207 tests, 533 assertions
  - 0 failures, 0 errors
- テスト2: 既存のテストを改良
  - 1811 tests, 13358 assertions
  - 14 failures, 0 errors
- だいたいOK
  - あとは仕様を合わせるなど

75


成果(1) Rubyとのマージ
—動作を確認したソフトウェアなど

- ビルド系
  - OptionParser, mk…
- Ruby のビルドには Ru…
  - YARV は Ruby のほとんどの動作をサポートしていたが, まだ…
  - → Ruby がビルドできない!

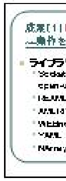
たとえば eval が 4つもあったり (5通りの挙動)

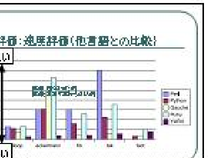
76

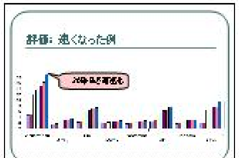
成果(1) Rubyとのマージ
—動作を確認したソフトウェアなどのごく一部

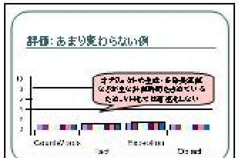
- Web系
  - Ruby on Rails
  - tDiary
  - ERB
  - Hiki
- 拡張系
  - OpenGL
  - Ruby/SDL
  - rcairo

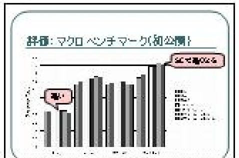
77

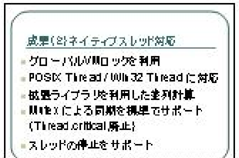



評価: 速度評価(他言語との比較)

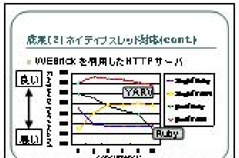

評価: 速くなった例


評価: あまり変わらない例


評価: マクロベンチマーク(初公開)


成果(2) ネイティブスレッド対応

- グローバルVMロックを利用
- POSIX Thread / Win32 Thread に対応
- 拡張ライブラリを利用した並列計算
- Mutex による同期を標準でサポート (Thread.critical 廃止)
- スレッドの停止をサポート

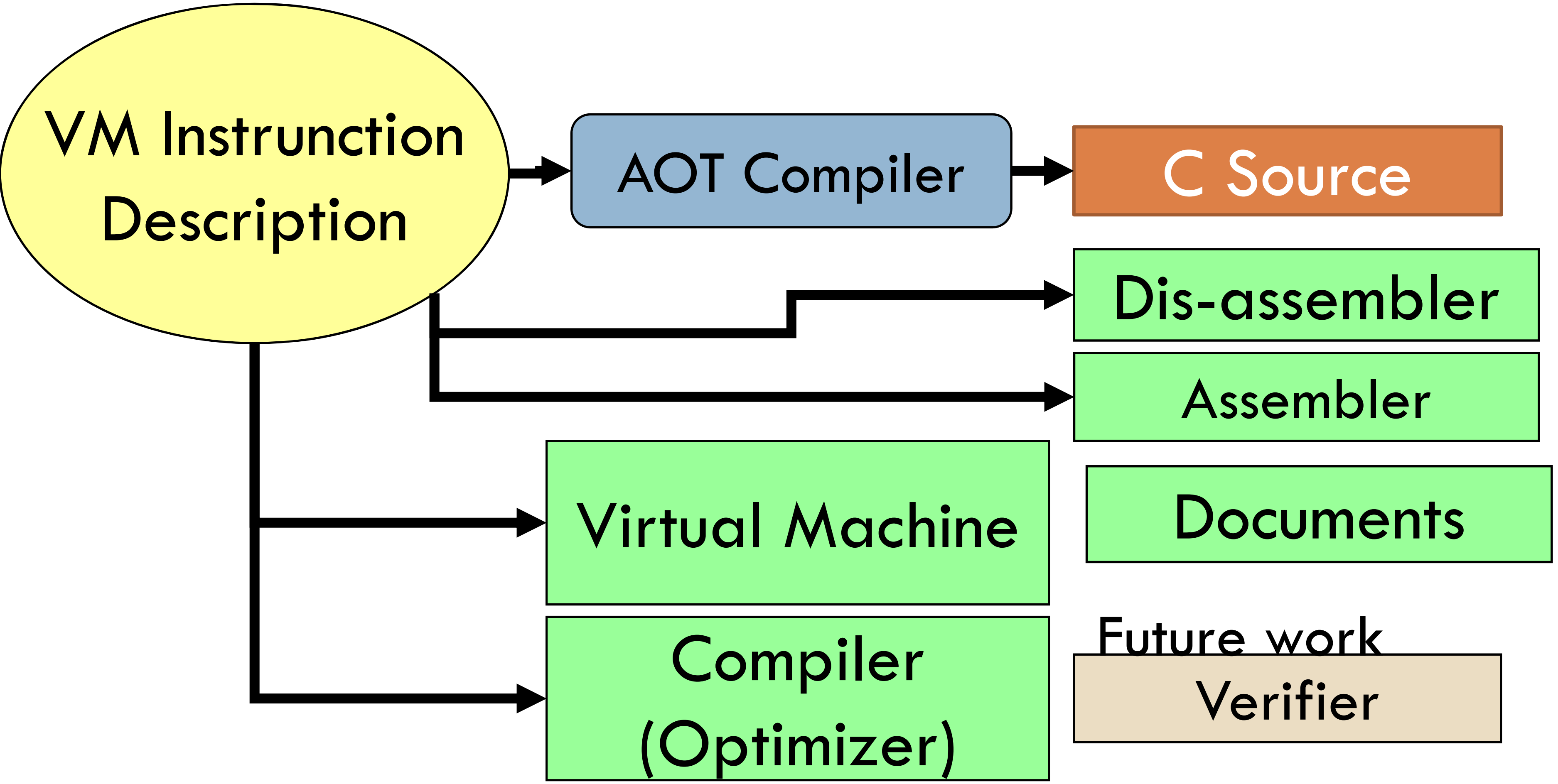

成果(2) ネイティブスレッド対応(cont.)

- WEBrick を利用したHTTPサーバ

成果(2) ネイティブスレッド対応(cont.)
拡張ライブラリと組み合わせた並列計算

Native thread with Giant VM Lock

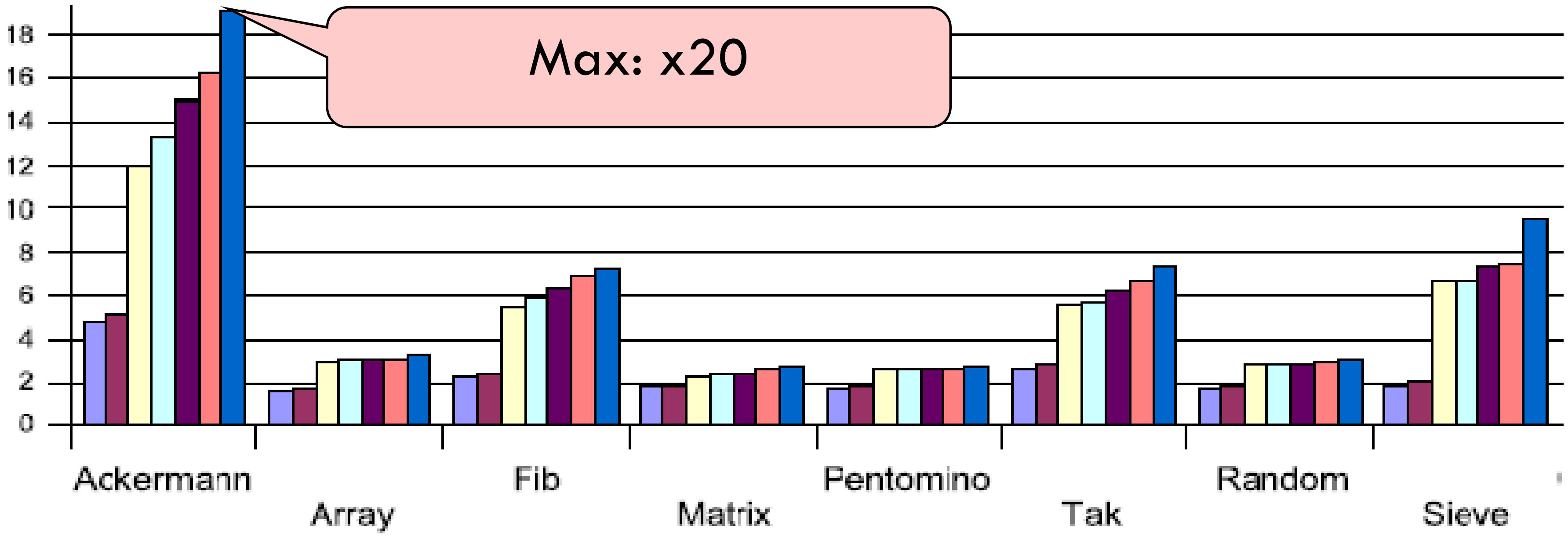
VM Generator



Enemies of YARV

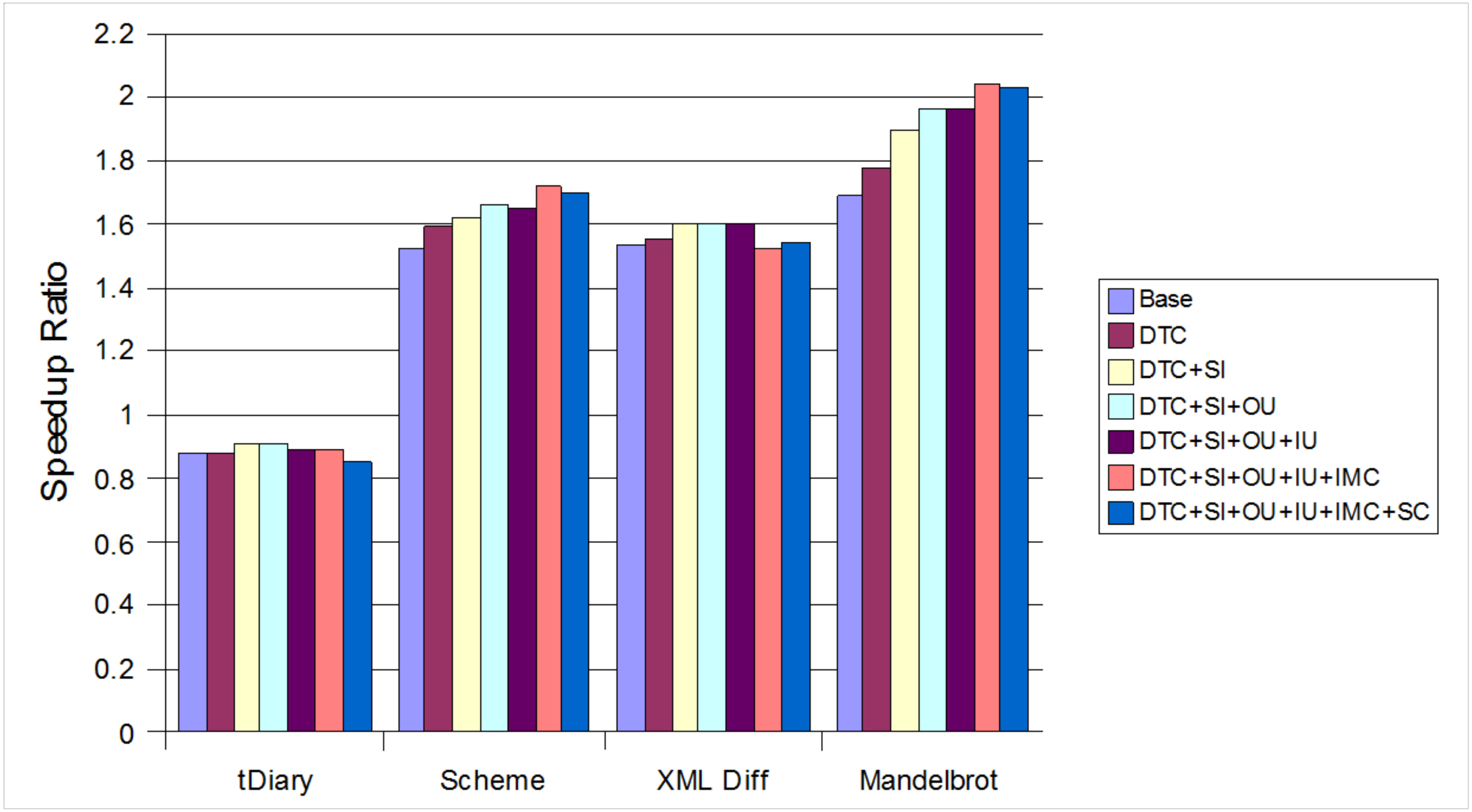
- Ruby Specification ≒ Matz
  - Ruby Spec kills many optimization techs
  - We love "Dynamic" "Meta" Programming, but…
- Changing Spec is also Nightmare
- Portability
  - We can't use system depending techs.
- Rivals **(not Enemy)**
  - Jruby, Rubinius, IronRuby, …
- Peggy work on my Job

Evaluation: Improve case

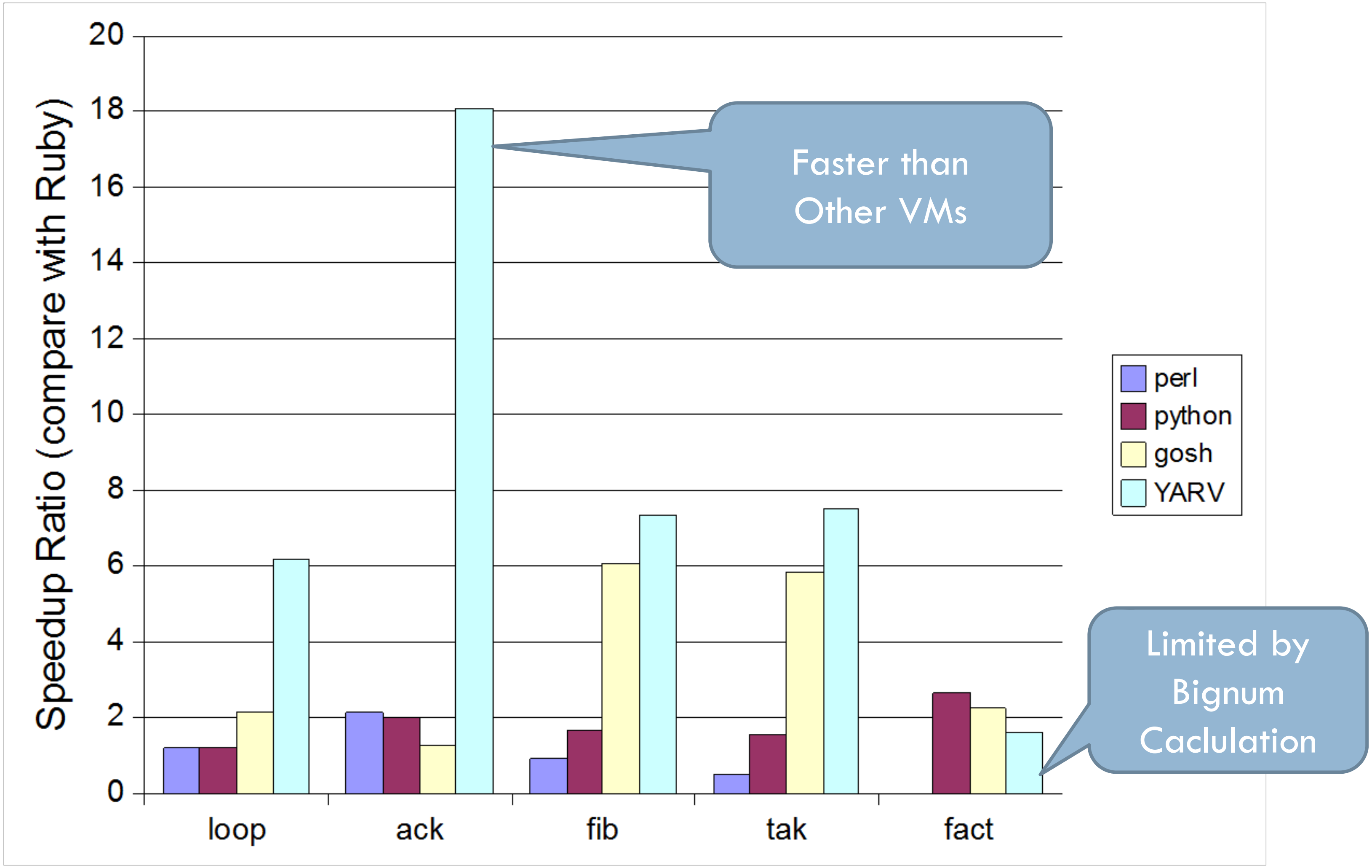


Max: x20

Evaluation: Macro-Benchmark



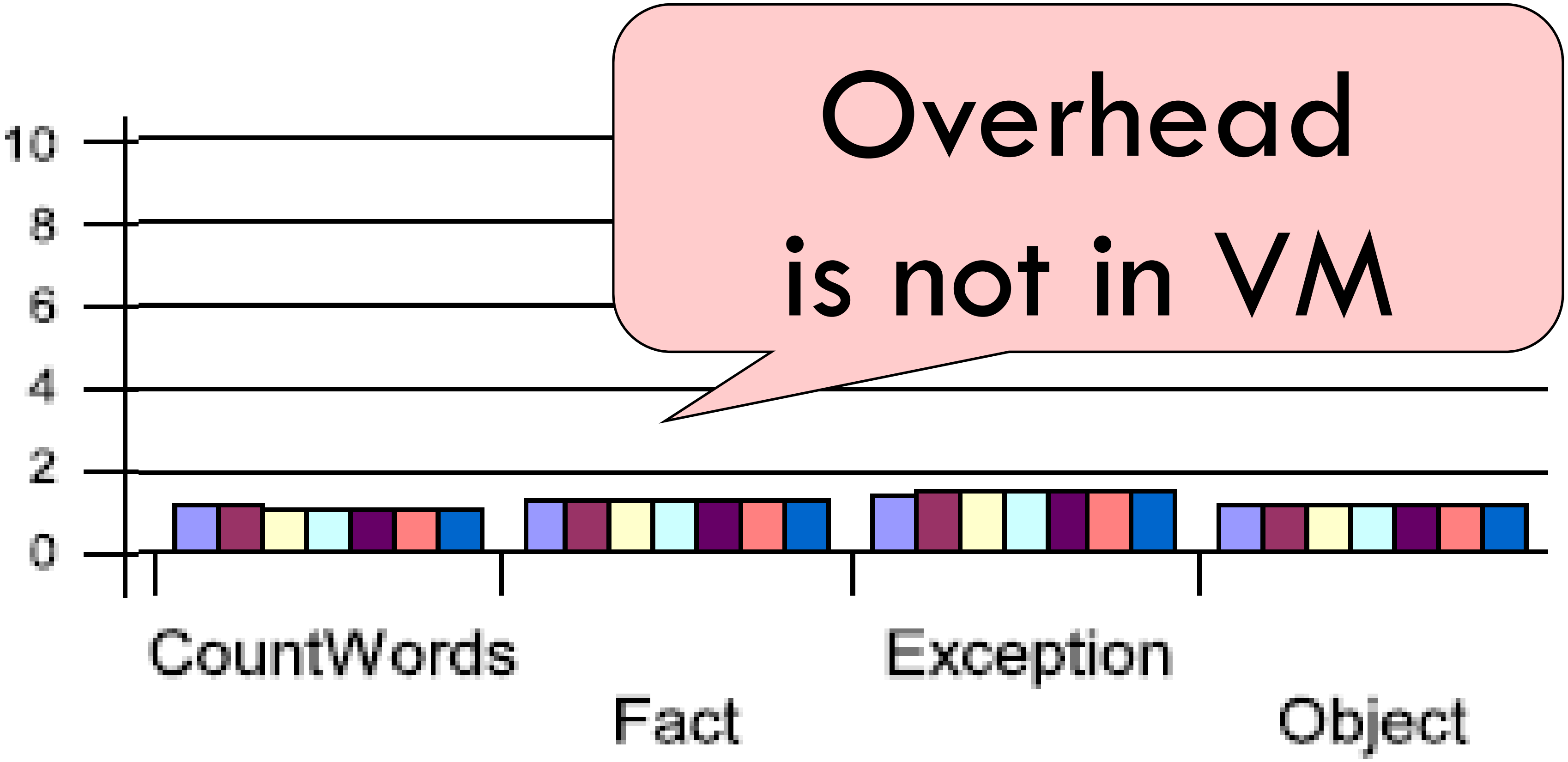
Evaluation: Compare with Other Languages



Faster than Other VMs

Limited by Bignum Caclulation

Evaluation: VM doesn't affect

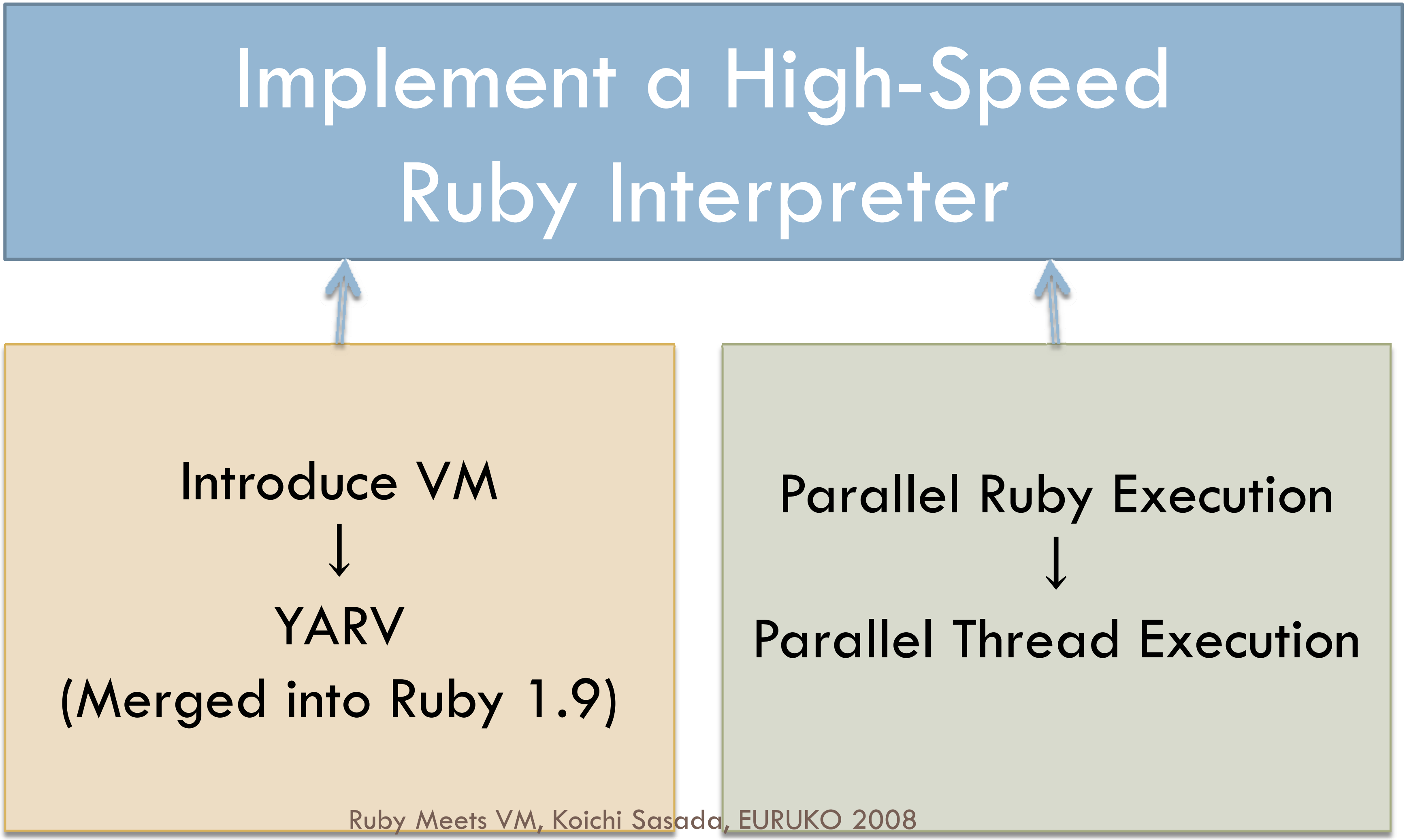


Overhead is not in VM

Advanced VM Topics

- Performance
  - Parallel Thread Execution
  - Embedding Float Value
  - JIT Compiler
  - Pre-Compiler
    - “Ruby to Compiled file” Compiler
    - “Ruby to C” Compiler
- New Feature
  - Multi-VM Creation
  - Customizable Ruby Core
  - Debug/Profile support feature

Ph.D Thesis
Efficient Implementation of Ruby Virtual Machine



Implement a High-Speed Ruby Interpreter

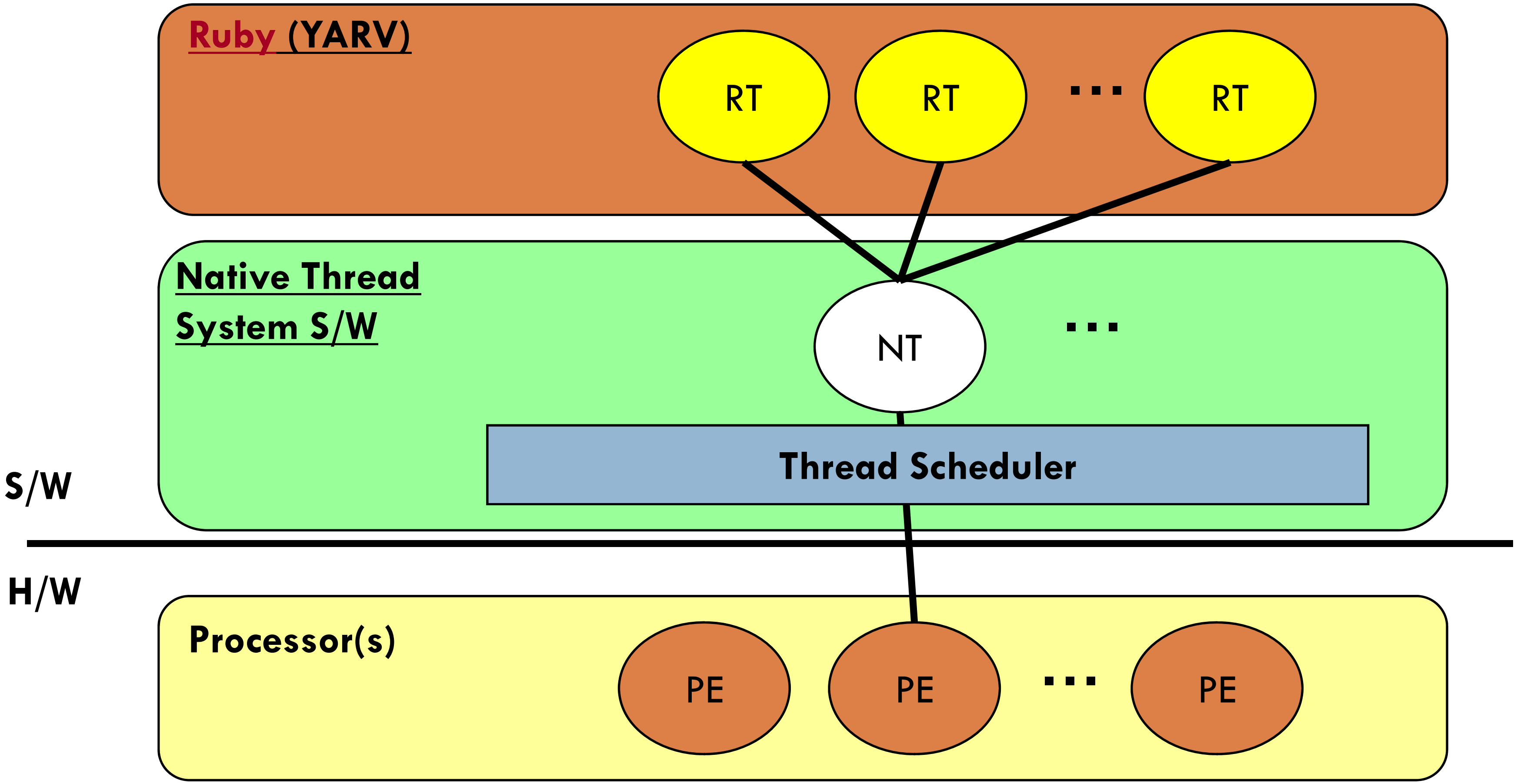
Introduce VM
↓
YARV
(Merged into Ruby 1.9)

Parallel Ruby Execution
↓
Parallel Thread Execution

Parallel Thread Execution

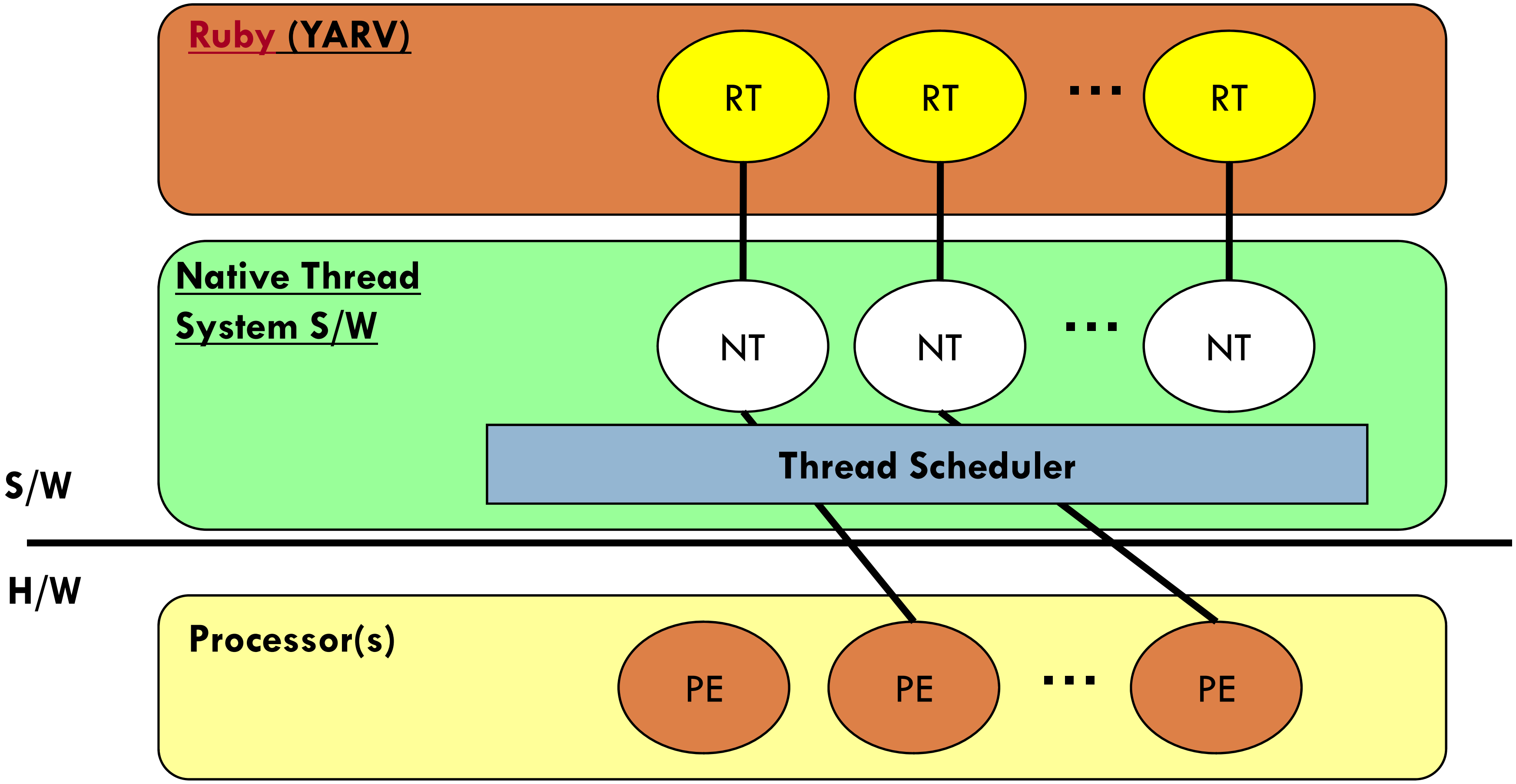
- Using Native Thread
- Get rid of Giant VM Lock

Method (1)
Ruby Thread and Native Thread (1:N) a.k.a -1.8 Ruby model



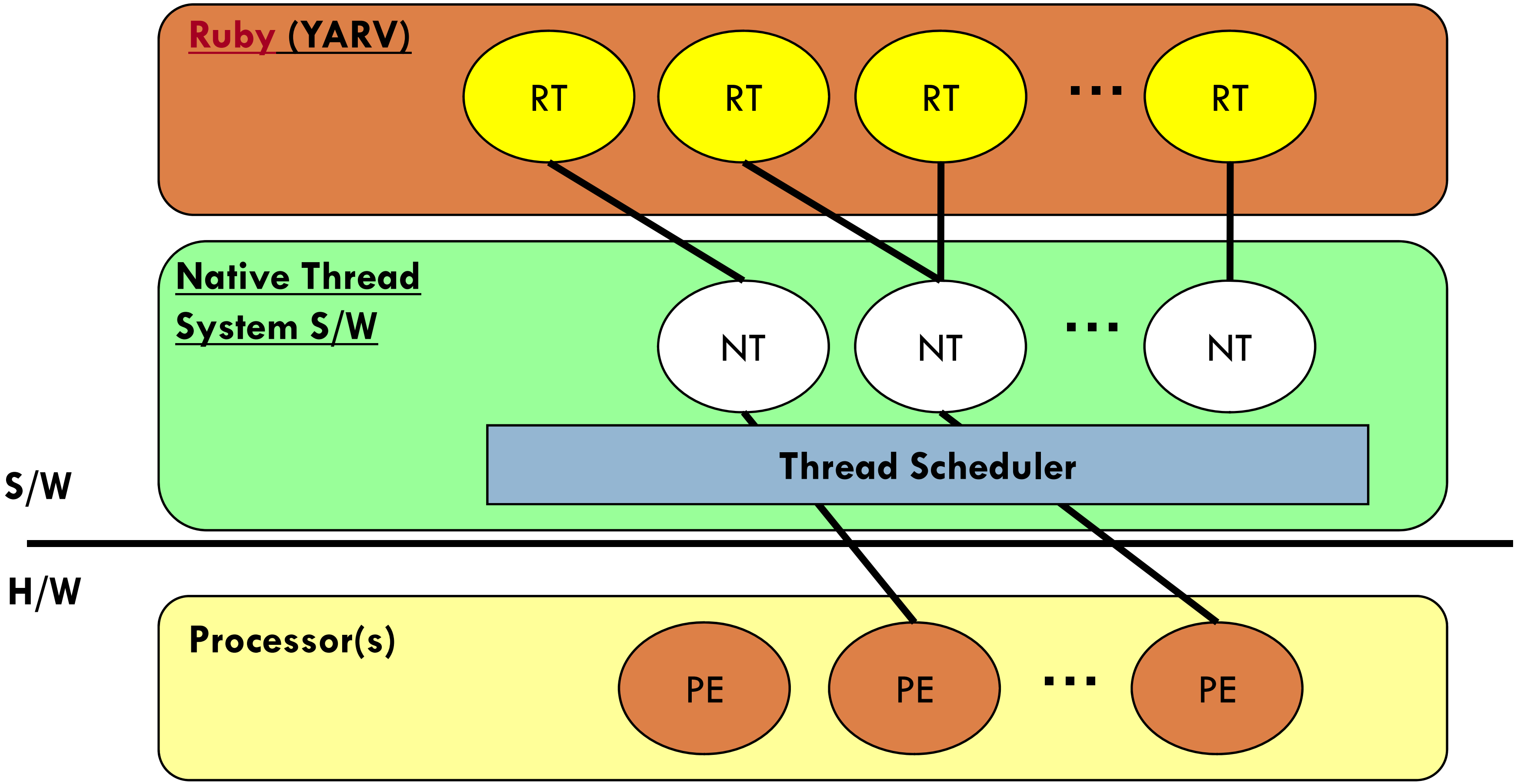
PE: Processor Element, UL: User Level, KL: Kernel Level

Method (2)
Ruby Thread and Native Thread (1:1)



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

Method (3)
Ruby Thread and Native Thread (N:M)



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

Discussion

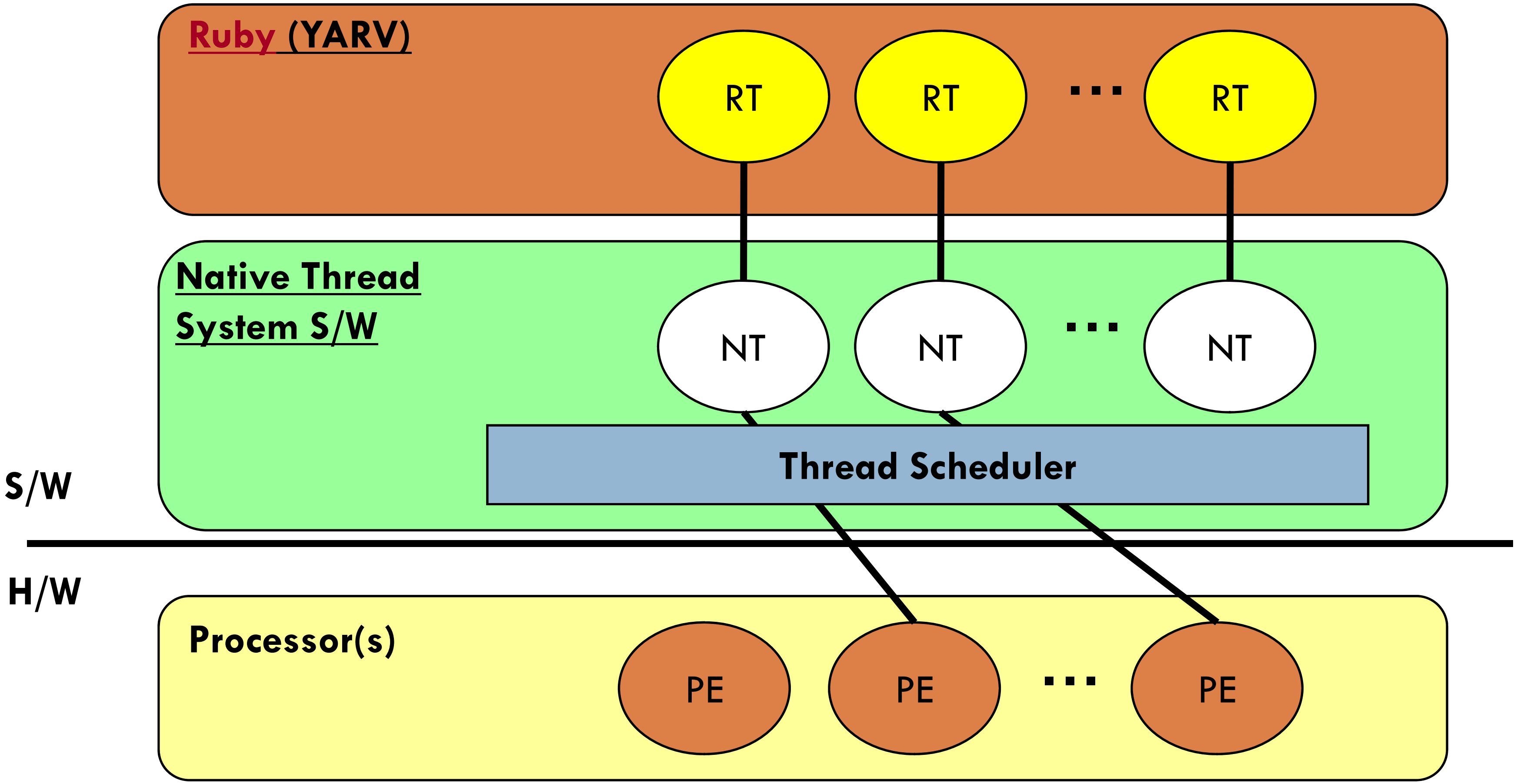
Ruby Thread and Native Thread

- Mapping with Native Thread and Ruby Thread

	Pros	Cons
1:N	Lightweight Thread Control	Can’t run in Parallel
1:1	Run in Parallel Simple, Portable	Heavyweight Thread Control (Creation, etc)
N:M	Lightweight Thread Control, Run in Parallel	Complication, Non-Portable

- Accept 1:1 model to make Ruby Simple
  - Depend Performance on Native Thread Libraries

Accepted Method:
Ruby Thread and Native Thread (1:1) ← Ruby 1.9/YARV



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

Introduction of Mutual Exclusion

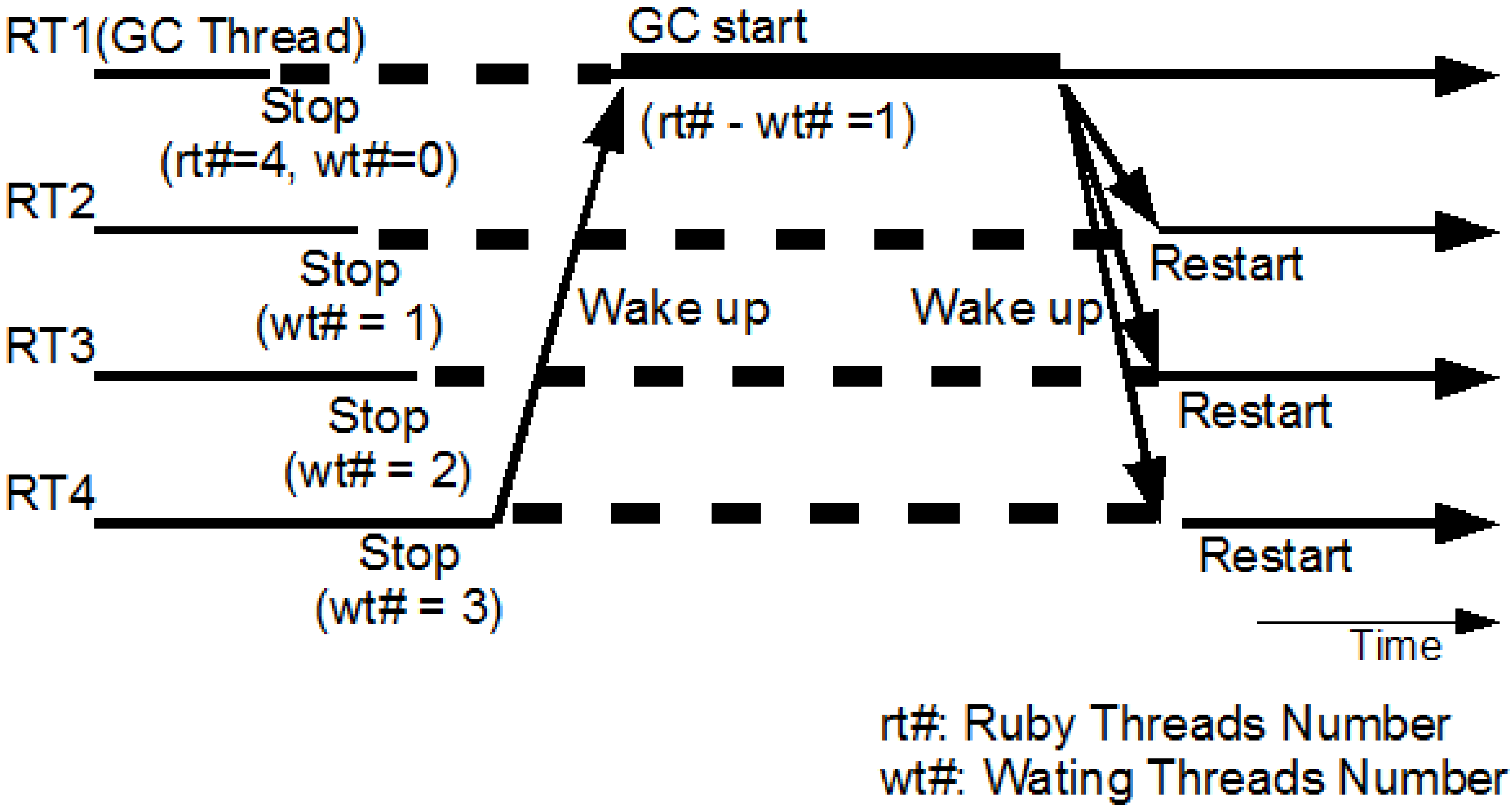
- Needed at
  1. Global VM Management Data
  2. Object Management / GC
  3. Inline Cache
  4. Thread Unsafe “C” methods

(1) Global VM Management Data

- Managed by Table
  - Variable Name → Value
  - Method Name → Method Body
  - …
- Introduce Synchronization at Table Operation
  - Get/Set
  - Easy

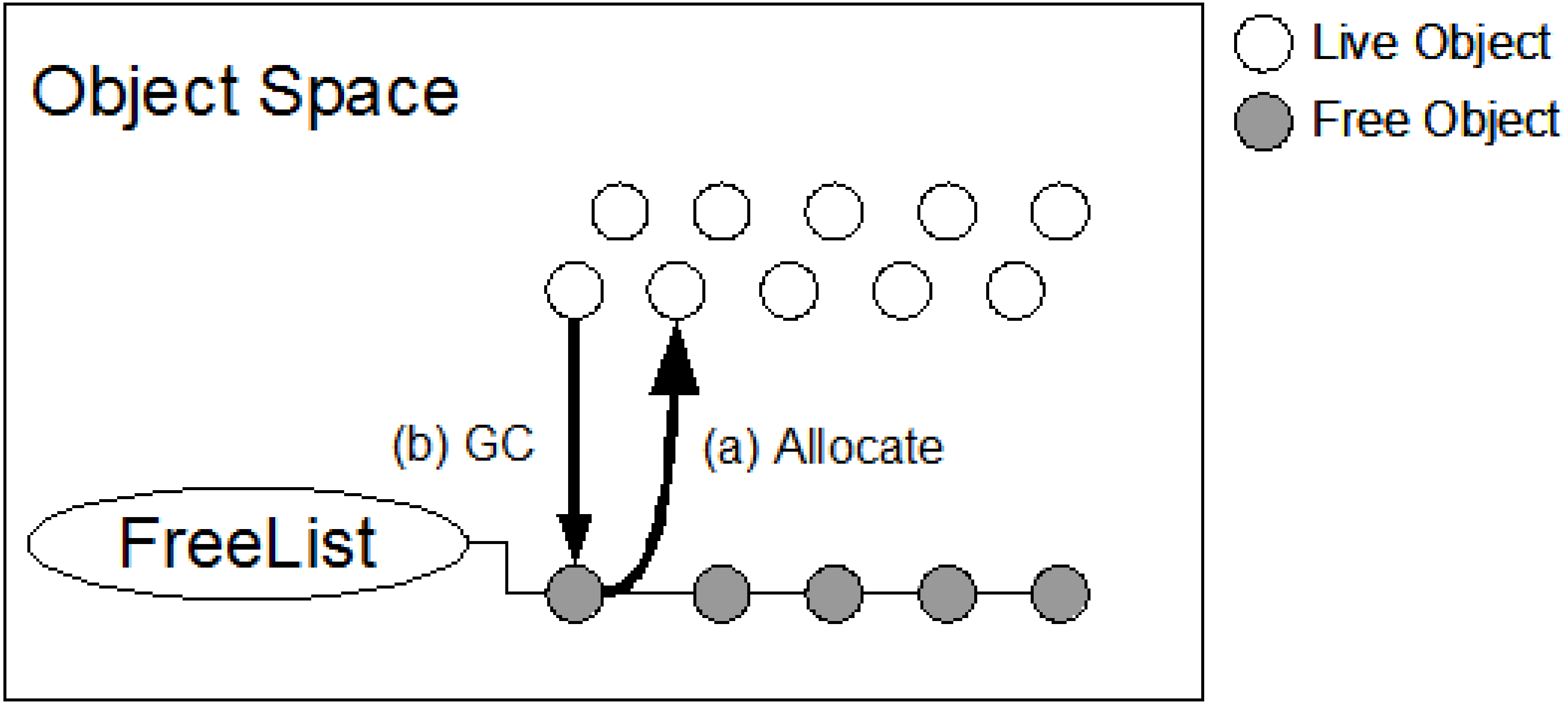
(2) Object Management/GC

- Synchronous GC

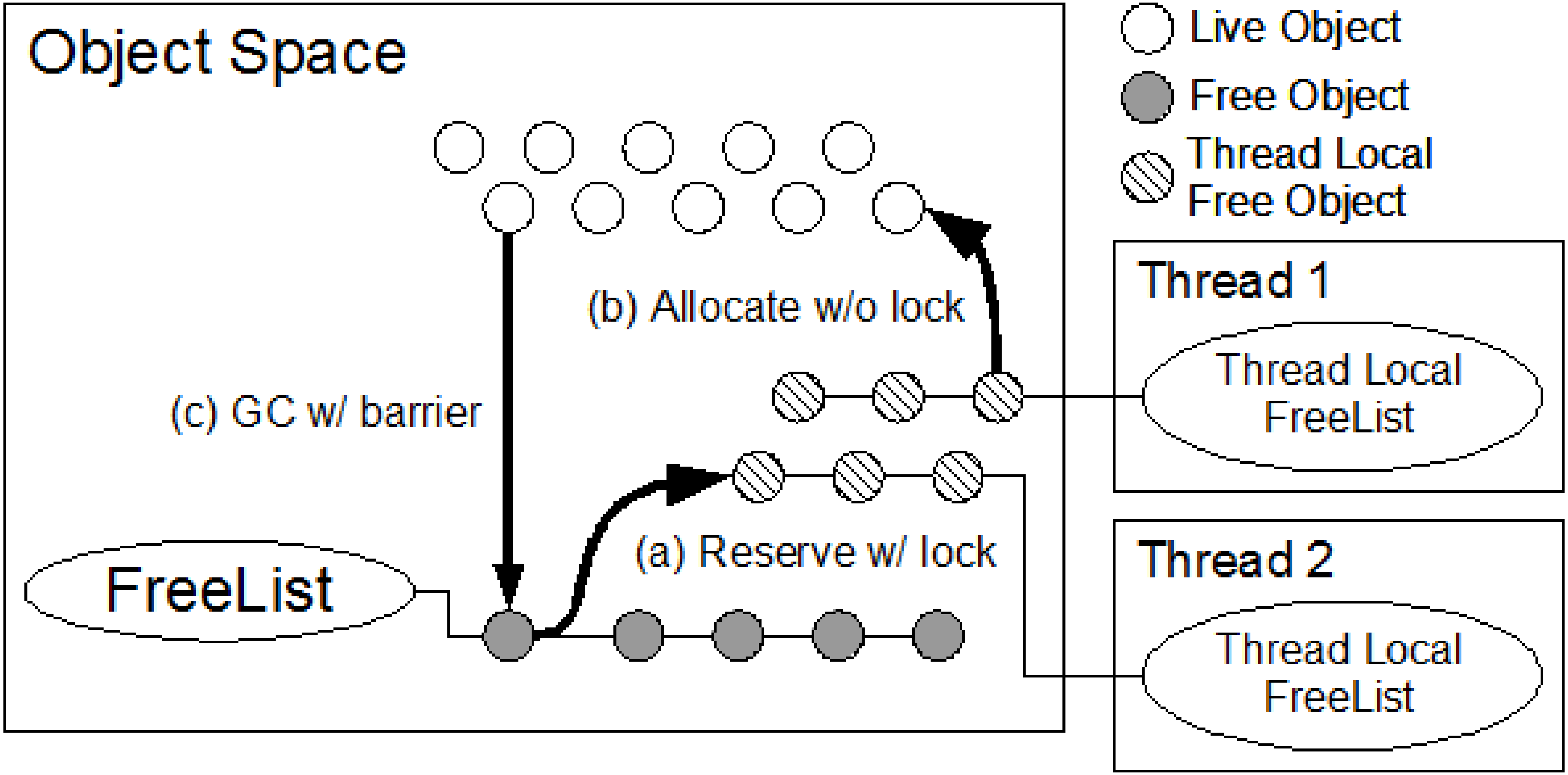


(2) Object Management

Lock-Free Object Allocation with Thread Local Free List



(2) Object Management
Lock-Free Object Allocation with Thread Local Free List



(3) Inline Cache

- Using by VM performance
  - Embed Cache Entries in Instruction Sequence
- Sync. For Coherence -> Performance Problem
  - Key and Value of Cache Entry
- Sync.-Free Inline Cache
  - Cache Miss -> Make a new entry
    - GC will clean-up old cache entries
    - Increase Miss-Penalty, but Good for average

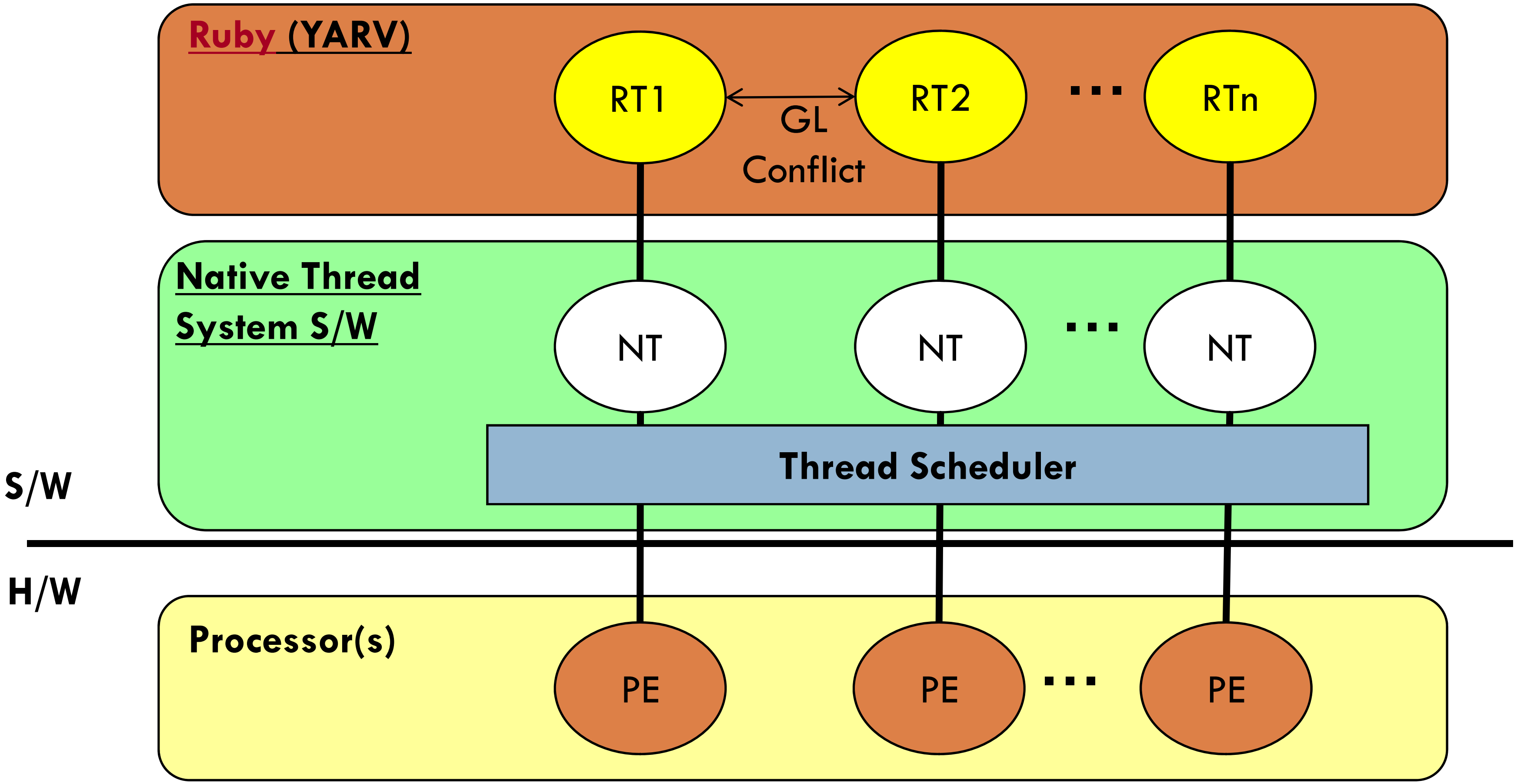
(4) Thread Unsafe “C” Methods

- CRuby has many many methods implemented by “C”
  - All of them are “Thread Unsafe”
  - Because -1.8 doesn’t support parallelization
- Basic Policy: Using Giant-Lock
  - Invoke old “C” method with Giant-Lock Acquire
  - Re-write C methods as Thread Safe, this method will be Giant-Lock free

Problem
Conflict of Giant-Lock Acquirement

- GL Conflict → Performance Decrement
- Limit Running CPU
  - Check GL Conflict Periodically
  - Limit their CPU
  - Using pthread_setaffinity_np on NTPL
    - SetThreadAffinityMask on Windows

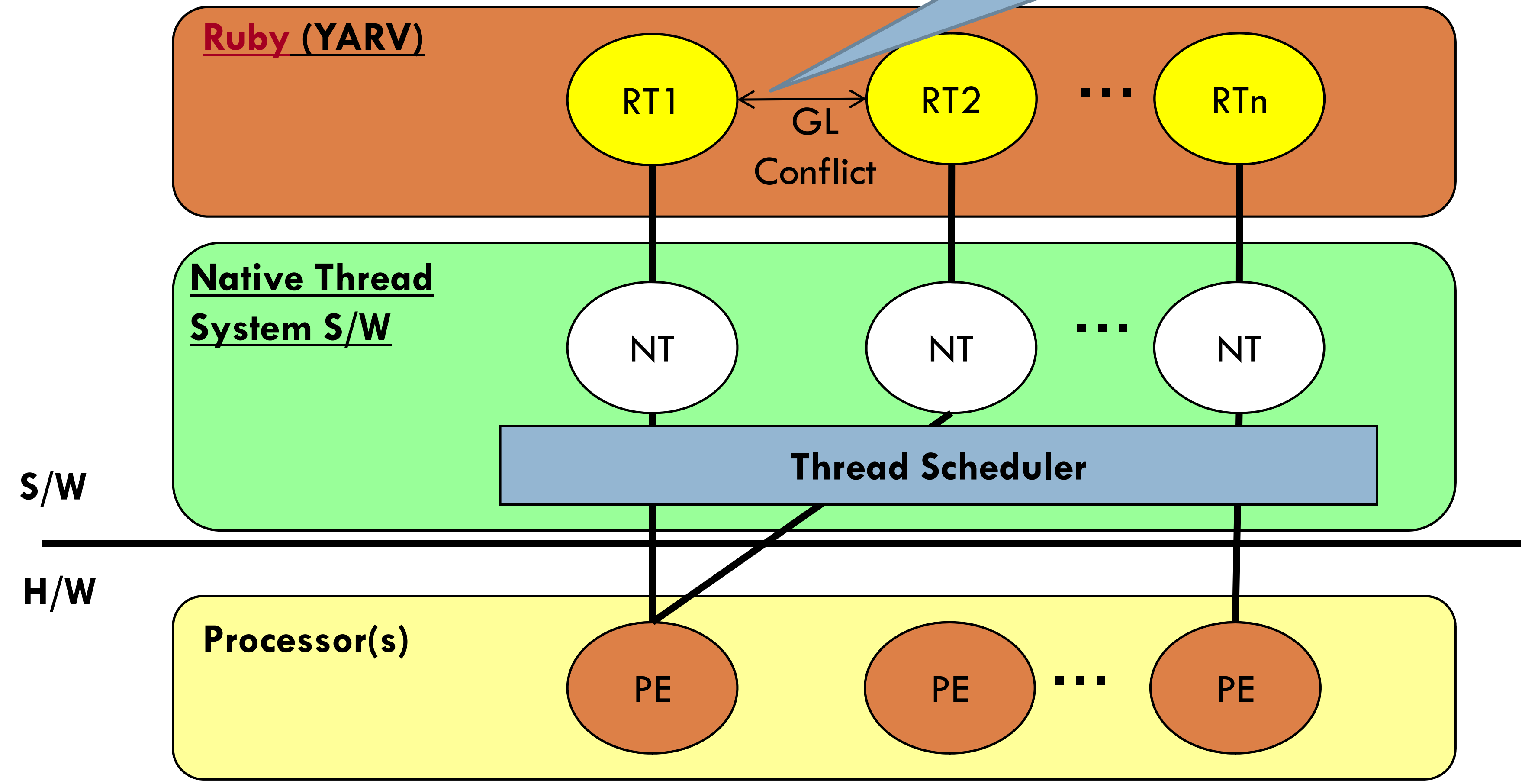
Running CPU Limitation



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

Running CPU Limitation

Force to run RT1 and RT2 in Serial
→Avoid Conflict



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

Performance Evaluation Environment

- Evaluation Environment
  - CPU: Intel Xeon CPU E5335 2.0GHz

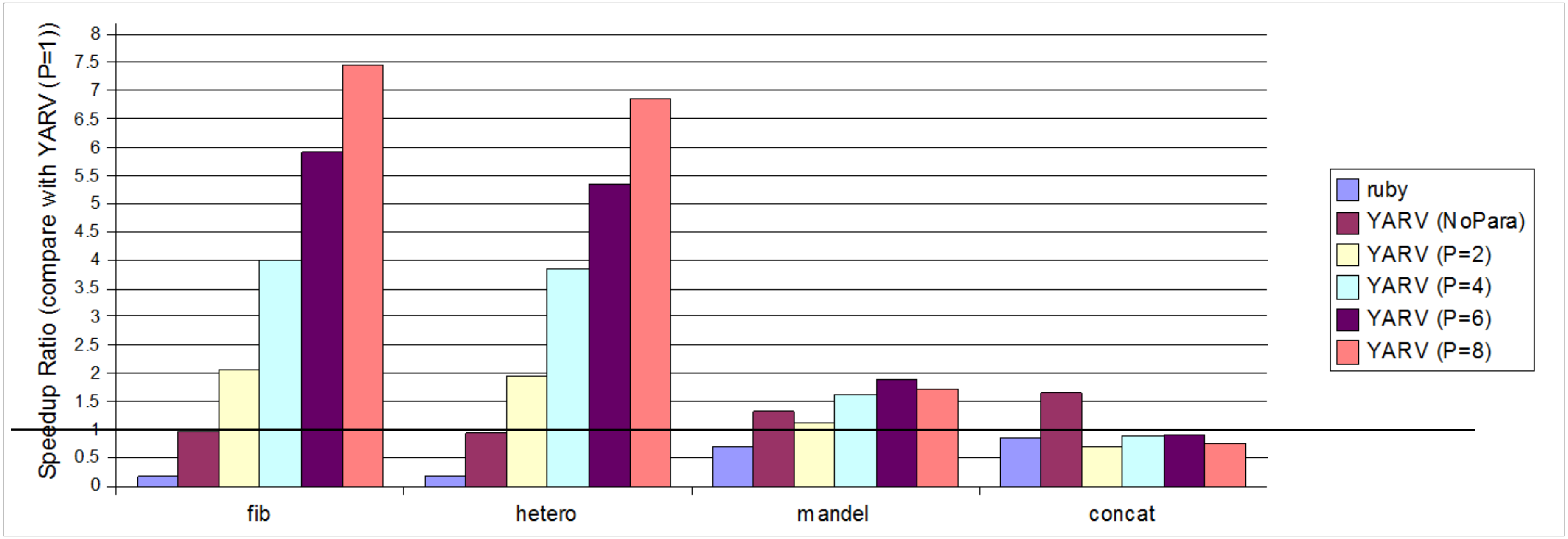
    Quad core x 2 = 8 core
  - OS: GNU/Linux 2.6.18 x86_64 SMP / NPTL
  - Compiler: gcc version 4.1.2
- Ruby
  - ruby 1.8.6 (2007-11-02) [x86_64-linux]
- YARV Optimization
  - All except Unification, Stack caching

Evaluation
Thread control Primitives

- Creation, Switch: 0.1M, Mutual Exclusion: 1M
- Low Performance for Creation/Join because of Native Thread
  - Native Thread Overhead
  - Memory Allocation Overhead
- High Performance Synchronization
- High Performance Thread Context Switch
  - Independent Stack-Depth (1.8 depends on depth)

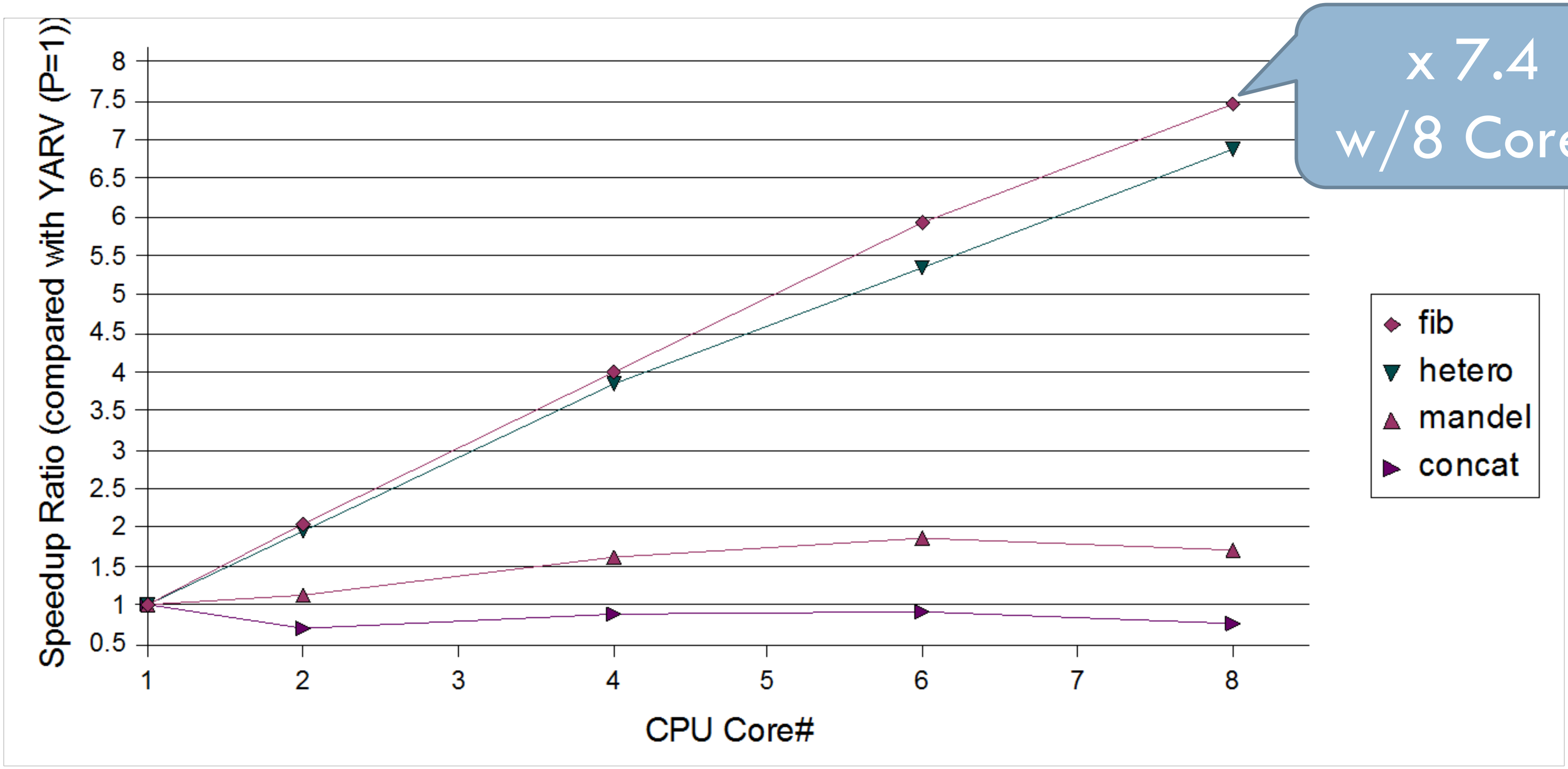
	Ruby (sec)	YARV (sec)	Ratio	NTPL (sec)
Creation	0.89	1.95	0.46	0.59
Mutual Exclusion	0.67	0.38	1.76	-
Switch (depth:1)	6.01	0.06	100.17	-
Switch (depth:16)	11.55	0.06	192.5	-

Evaluation
Result (Micro-benchmark)



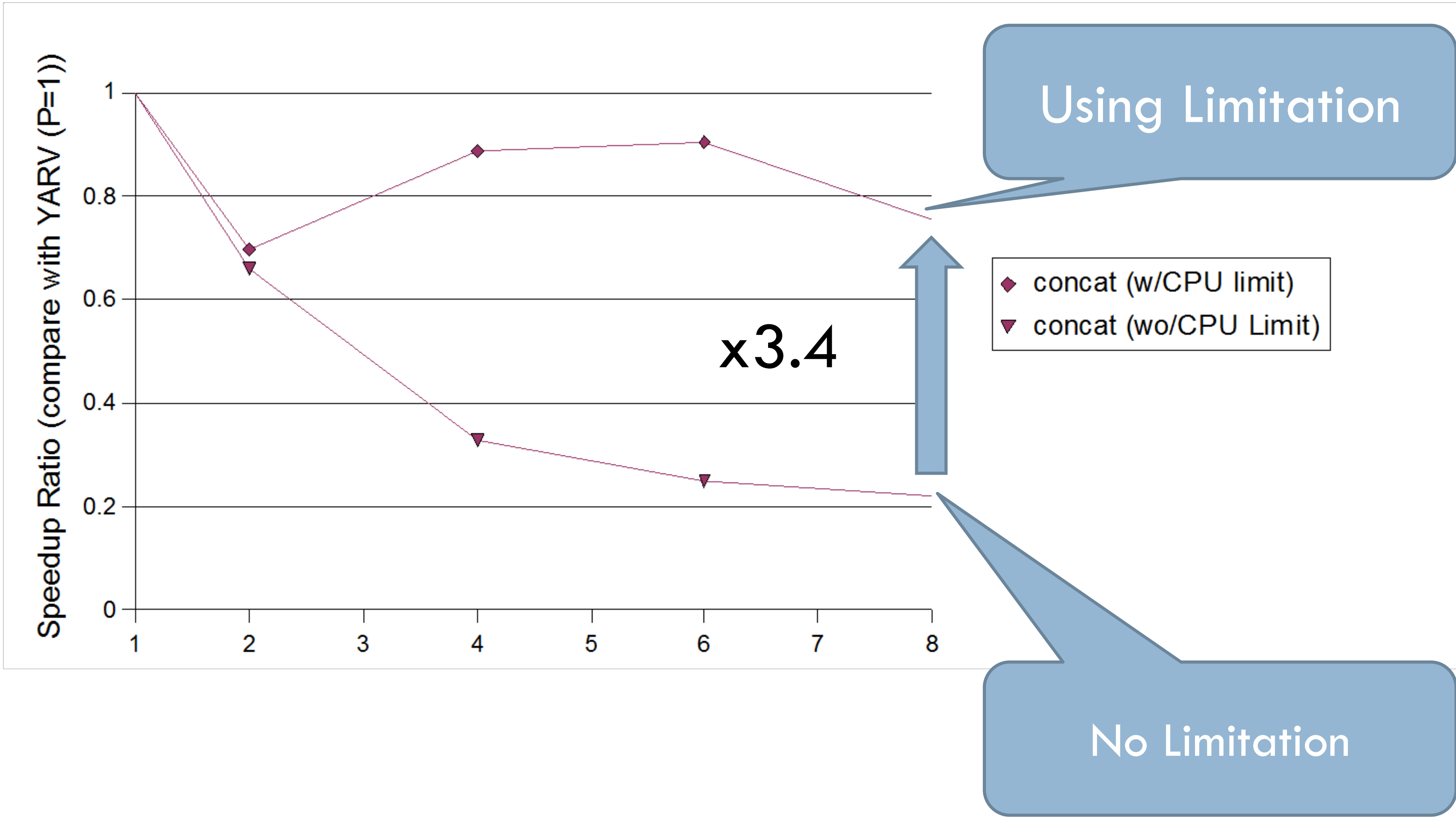
- •fib: fib(N) (Make new Thread if N>30)
- •hetero: fib + concat (1 thread)
- •mandel: Mandelbrot (Big GC overhead)
- •concat: String Concatenate (No Parallelism)

Evaluation Result (Micro-benchmark)



x 7.4
w/8 Cores

CPU Limitation



Using Limitation

x3.4

No Limitation

Parallel Thread Execution Problem

- Unsafe “Old C” methods
  - Replacing all is not easy task
  - Man Power Problem?
- Programming Model
  - Is Parallel Thread Application easy to write?
- Ruby 1.9
  - 1.9 support Native Thread
  - 1.9 doesn’t support Parallel Thread Execution

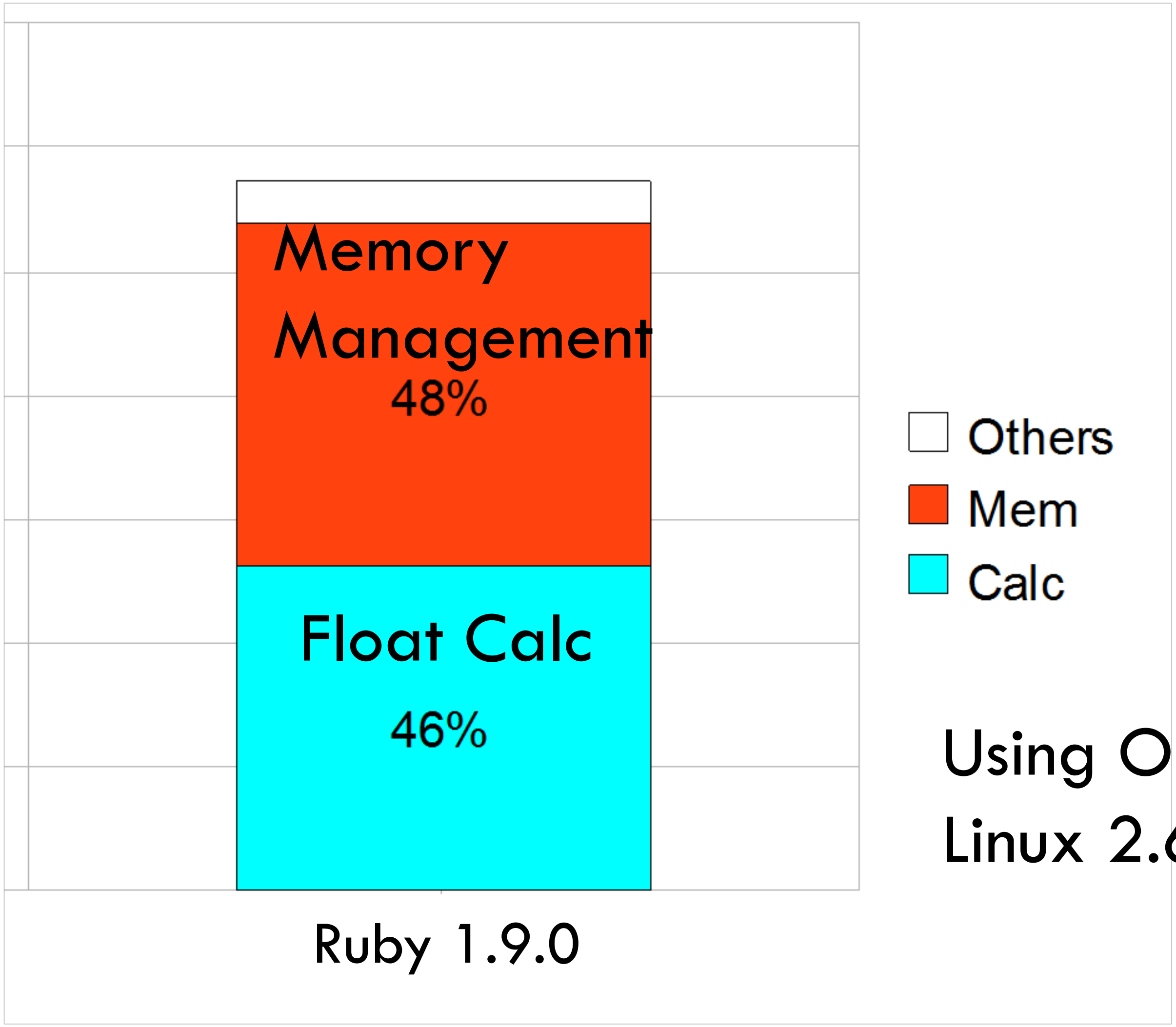
Embedded Float Representation

- Float object is not immediate type
  - This means that Float is “allocated” each time
  - Ex) Fixnum, Symbol, etc
- Half execution time of Float calculation is Memory management

Toy-Program

```
i = 0; f = 0.0
while i<30_000_000
  i += 1
  f += 0.1; f -= 0.1
  f += 0.1; f -= 0.1
  f += 0.1; f -= 0.1
  f += 0.1; f -= 0.1
end
```

List of Execution Time Toy-Program



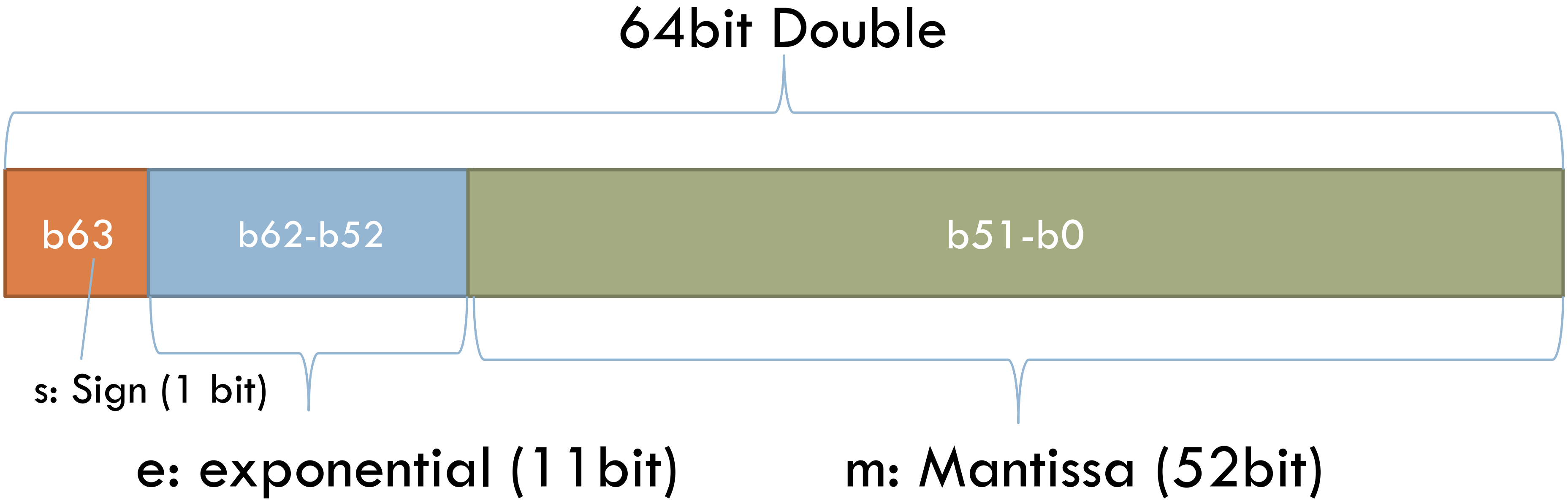
Using OProfile

Linux 2.6 (x86_64), Xeon

Embed Float Object as Fixnum

- Solution: Embed 64bit Float value to 64bit CPU Pointer type!

Review
IEEE 754 Double Precision Representation



$$-1^s 2^{e-1024} m \quad (m = 1 + \sum_{i=0}^{51} \frac{b_i}{2^{52-i}})$$

Discussion
How to Embed 64 bit Double?

- VALUE embed Object doesn't need memory overhead
- **64bit CPU have 64 bit pointer type**
  → Use 64 bit CPU
- At least we need 1 bit for TAG bit
  - From Mantissa?
    - Decrease Precision
  - From Exponential?
    - Decrease Representation Range

Proposal

- From Exponential

  But Store Float in Heap if it is Out of Range

  → **Save a Range and Precision**

  - Often used 01000000000b～10111111111b

    (2^{-512}~2^{511})

  - If Float is out of range, alloc from Heap
  - **Float Out of this Range is Rare Number on Numeric Application -> Practical Solution**

Proposal
Real Program

□ 1) Check the Range of “e” (512～1535)

□ 2) IEEE745 double -> Float (Encoding)

□ 3) Float -> IEEE745 double (Decoding)

Henry S. Warren, Jr. Hacker's Delight
ハッカーのたのしみ
本物のプログラマはいかにして問題を解くか
ヘンリー・S・ウォーレン, ジュニア

Proposal
Float Representation with Tag

- e: 10000000000b〜01111111111b
- Note that if “b62 != b61”, we can embed.
- On Ruby, Tag is at LSB

  → Left Rotate **3bit**
- b63 b62 b61 b60 ··· b0 → 3 bit rotate

  b60 ··· b0 b63 b62 b61

Proposal
Real Encoding Code

```
VALUE rb_float_new(double dbl) {
  VALUE v1 = BIT_DOUBLE2VALUE(dbl);
  VALUE v2 = BIT_ROTL(v1, 3);
  if ((v2 + 1) & 0x02) // check lower 2 bits
    return v2 | 0x01;     // Embed tag
  else {
    if (dbl == 0) // 0.0
      return ruby_float_zero;
    else                  // alloc from Heap
      return rb_float_new_in_heap(dbl);
  }}
```

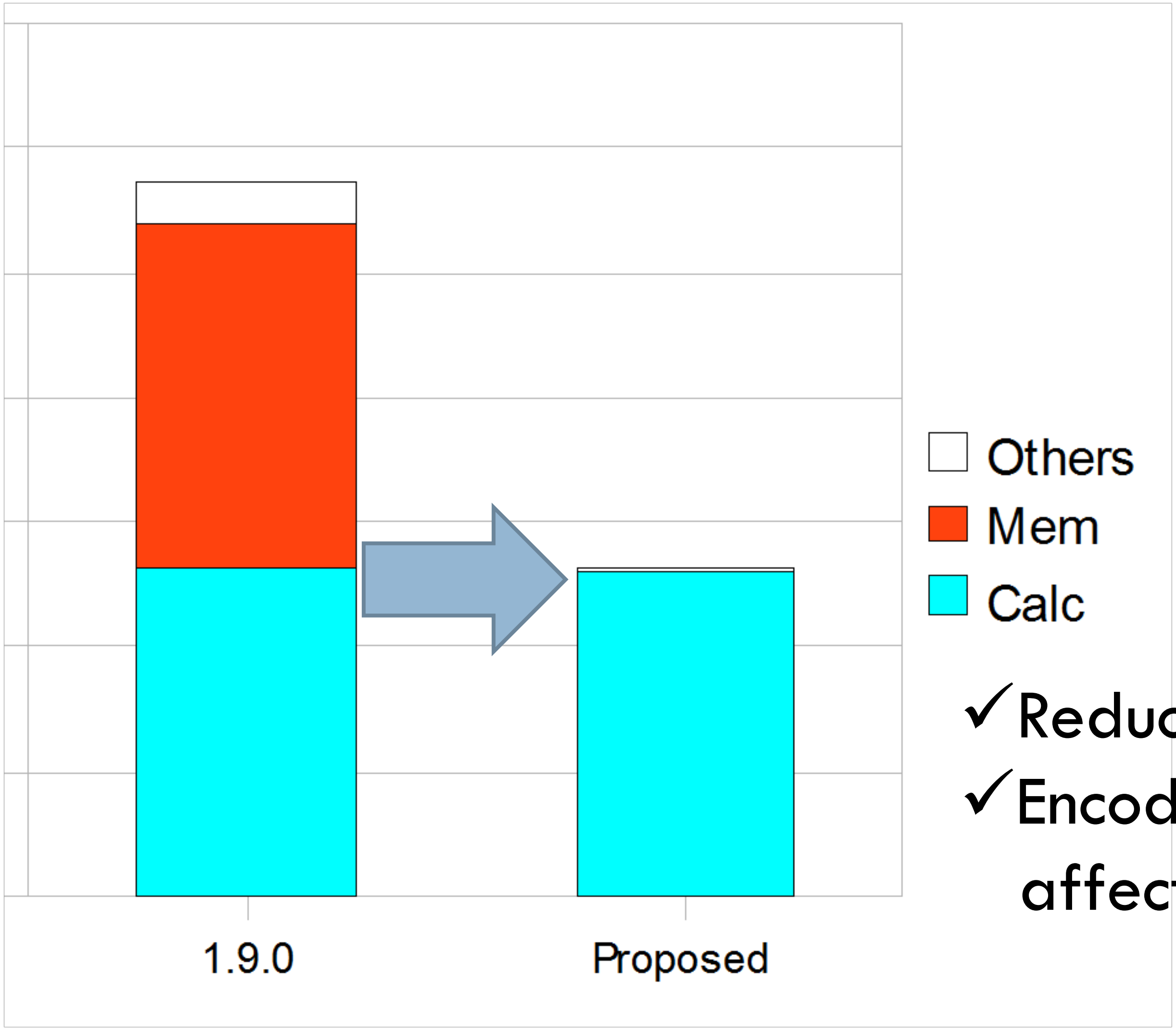
Proposal
Real Decode Code

```
double RFLOAT_VALUE(VALUE v) {
  if (v & 1) {
    VALUE v1 = v ^ ((v >> 1) & 1);
    VALUE v2 = BIT_ROTR(v1, 3);
    return BIT_VALUE2DOUBLE(v2);
  }
  else
    return RFLOAT(v)->float_value;
}
```

Implementation

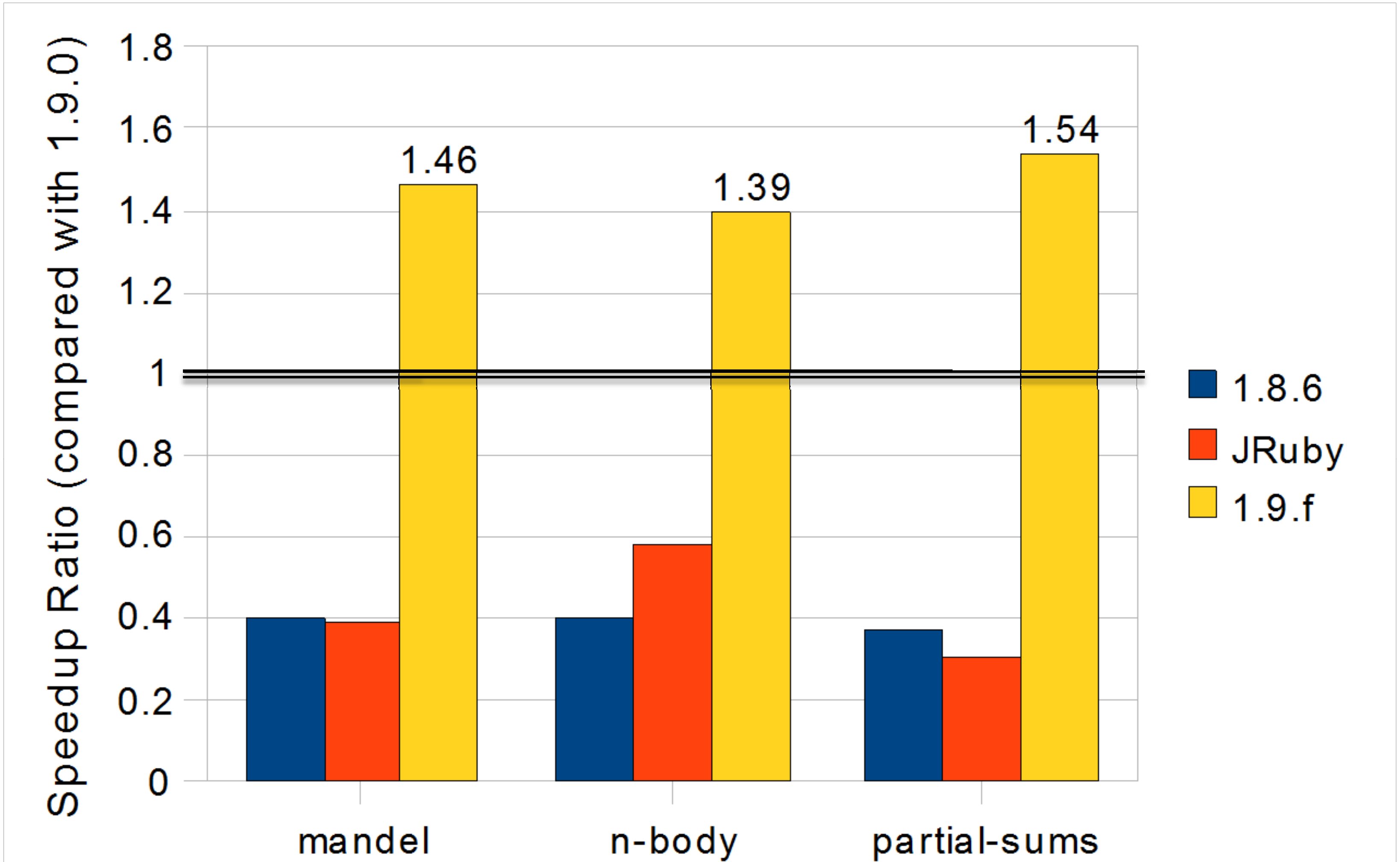
- Ruby 1.9.0-0
- Easy to Implementation
- No Spec Changes

Evaluation
Toy-Program



✓Reduce Mem Time
✓Encode/Decode don't affect to Performance

Evaluation
Compared with other Ruby Impl.



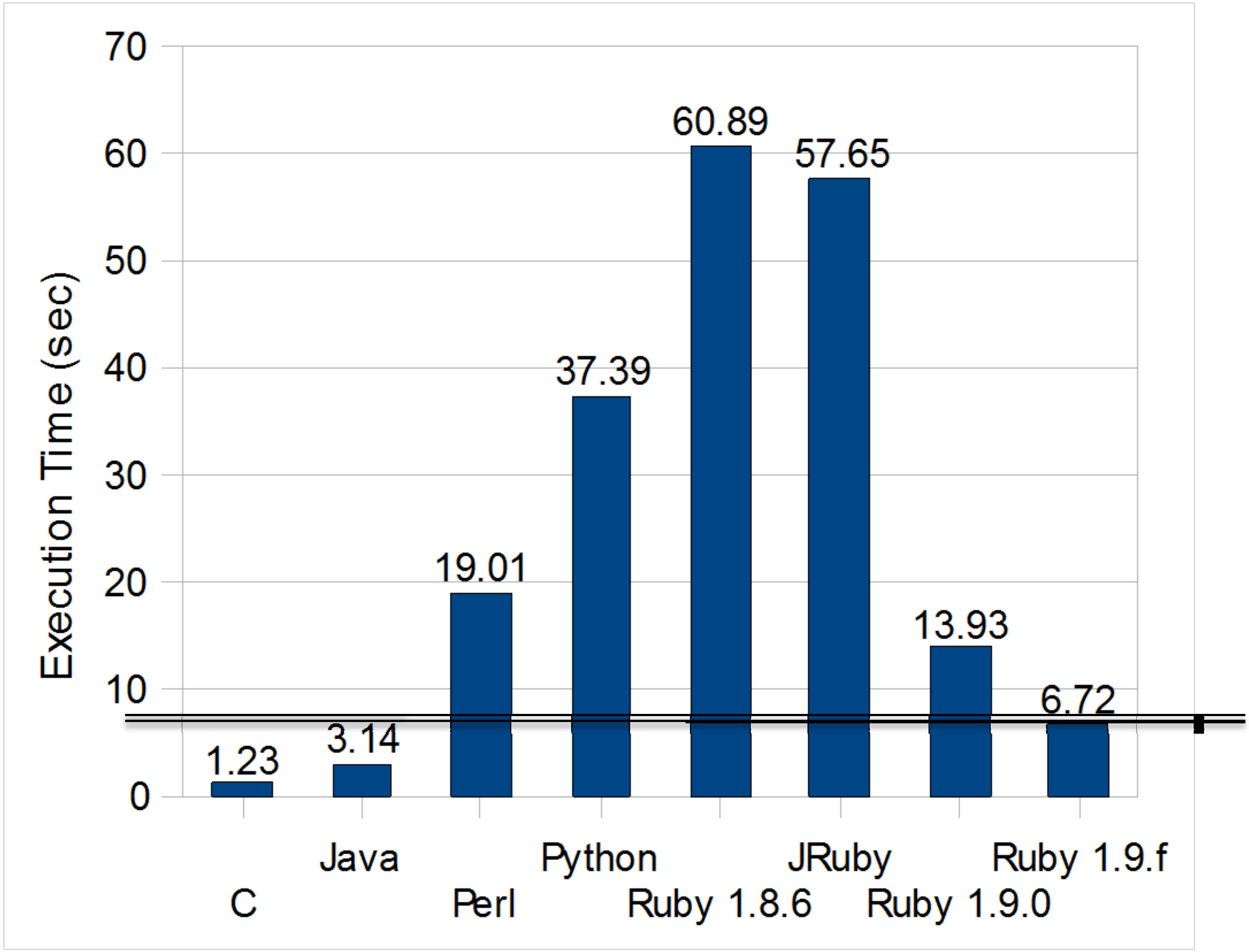
From Comp. Lang. Shootout [4]

Evaluation
Compared with Other Languages

- Evaluate with other languages
- Note that C/Java use “volatile” to avoid optimization

```
i = 0; f = 0.0
while i<30_000_000
   i += 1
    f += 0.1; f -= 0.1
    f += 0.1; f -= 0.1
    f += 0.1; f -= 0.1
    f += 0.1; f -= 0.1
end
```

Evaluation
Compared with Other Languages



[PLAN] JIT Compiler

- I’m re-designing to reduce VM instructions to impl. it easy
  - Current VM has about 50 instructions
  - Ex) “definemethod” move to “Method”

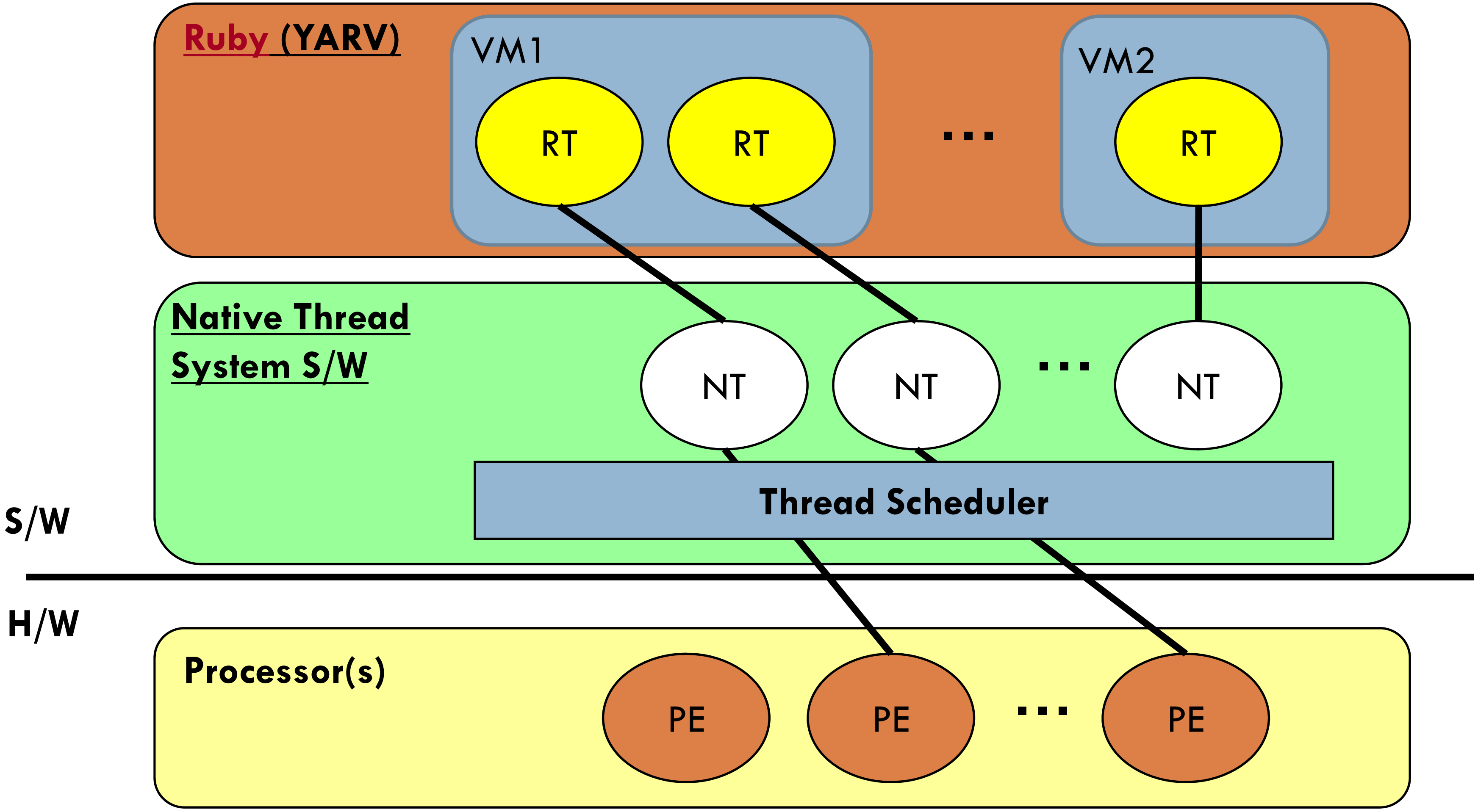
[PLAN]
Pre-Compiler

- YARV VM Generator helps us
  - Ruby to “Pre-compiled”
  - Ruby to “C”
- Purpose
  - Eliminate Loading-Time
  - More aggressive optimization
  - Obfuscation (?)

[PLAN]
Multi-VM Creation

- Purpose
  - Embed Ruby into Application
    - mod_ruby, …
  - Sand-box

Multi-VM Overview



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

Multi-VM Points

- How to control VMs?
  - C Level? → Designed with Nobuyoshi Nakada
    - Making new VM is need only 3 lines
  - Ruby Level?
- How to share environments Inter VM
  - Trade off between Isolation and Util.

[PLAN] Customizable VM Core

- Ruby is tooooo FAT to use XXX purpose
  - Many Many Convenience Methods/Feature
- Need Re-design Ruby Core

[PLAN] Debug/Profile Support Feature

- Only cheep Debugger/Profiler API
  - set_trace_func, Thread#set_trace_func
- Introduce "break" instruction?

Future Work
Benchmark

- Current Benchmark suits is for checking YARV Performance
  - Focus to YARV optimization
  - Toy benchmarks
- We need more pragmatic benchmarks

Summary

- YARV Merged into Ruby 1.9
- I’m working at Advanced VM Topics
  - Performance
    - Parallel Thread Execution
    - Embedding Float Value
    - JIT Compiler
    - Pre-Compiler
      - “Ruby to Compiled file” Compiler
      - “Ruby to C” Compiler
  - New Feature
    - Multi-VM Creation
    - Customizable Ruby Core
    - Debug/Profile support feature

Summary

Merging YARV is not a goal, but a start

VM is a very flexible infrastructure to hack

One more thing…

Sasada Lab@U-Tokyo

- I’ll make a laboratory from 2009, Apr
  - Department of Creative Informatics, Graduate School of Information Science and Technology, The University of Tokyo
  - Graduate School
  - Lab is at Akihabara, Tokyo, Japan
- Unfortunately I can’t employ you as Research Assistant
  - There are not enough grants in Japan…

Research Topics

- Ruby, Ruby, Ruby, PHP, Ruby, Ruby
- Ruby, Ruby, Ruby, Ruby, Python, Ruby
- Ruby, Perl, Ruby, Ruby, Ruby, Ruby
- Ruby, Ruby, Ruby, Ruby, Lua, Ruby
- Ruby, Ruby, Java, Ruby, Ruby, Ruby
- Implementation of Programming Language
- Operating System / Processor Architecture
- Software development

Sasada Lab.

□

```
if you.have_interest(
  :Japan, Tokyo, :Akihabara,
  :Japanese,
  :Ruby, :Research, :Development
)
  you.send_mail_to “ko1 at atdot dot net”
end
```

Thank you for your attention!
Any Questions?

ささだ こういち
Koichi Sasada
Ko1 at atdot dot net