RUBY CONTINUATION

Koichi Sasada

Department of Creative Informatics, Graduate School of Information Science and Technology, The University of Tokyo

RUBY CONTINUATION CONSIDERED HARMFUL

Koichi Sasada
ko1@atdot.net

Notice

- Language
- Place

Ruby

- Object Oriented Scripting Language
  - Minor Language here
  - Class based OO-System
  - Easy to write Closure (Block)
  - Dynamic Nature
  - From Japan
    - Designed/Implemented by Yukihiro Matsumoto
  - Have a Continuation class

FYI

- Ruby 2.0 – since 2003 3/31
  - Not released
- Perl 6 – since 2003 4/1
  - Not released

Ruby History

- 1993 2/24 Named “Ruby”
- 1995 Released via NetNews
- 1999 First Ruby book in Japanese
- 2001 First Ruby book in English
- 2004? Ruby on Rails
- 2007 12/25 Ruby 1.9.0-0 Released

YARV: Yet Another Ruby VM History

- 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
    - YARV is merged into Ruby Official Repository
  - **25, Dec 2007 (GMT) 1.9 Release**

Ruby Development History

- Ruby Development History
  ≠ Battle with Continuation bugs
  (+ other bugs)

Ruby Continuation

- Captured with “callcc” method
- Full-Continuation

```
# Sample Code
callcc{|cont|
  $cont = cont # “$cont” is global variable
  # …
}
$cont.call # call continuation
```

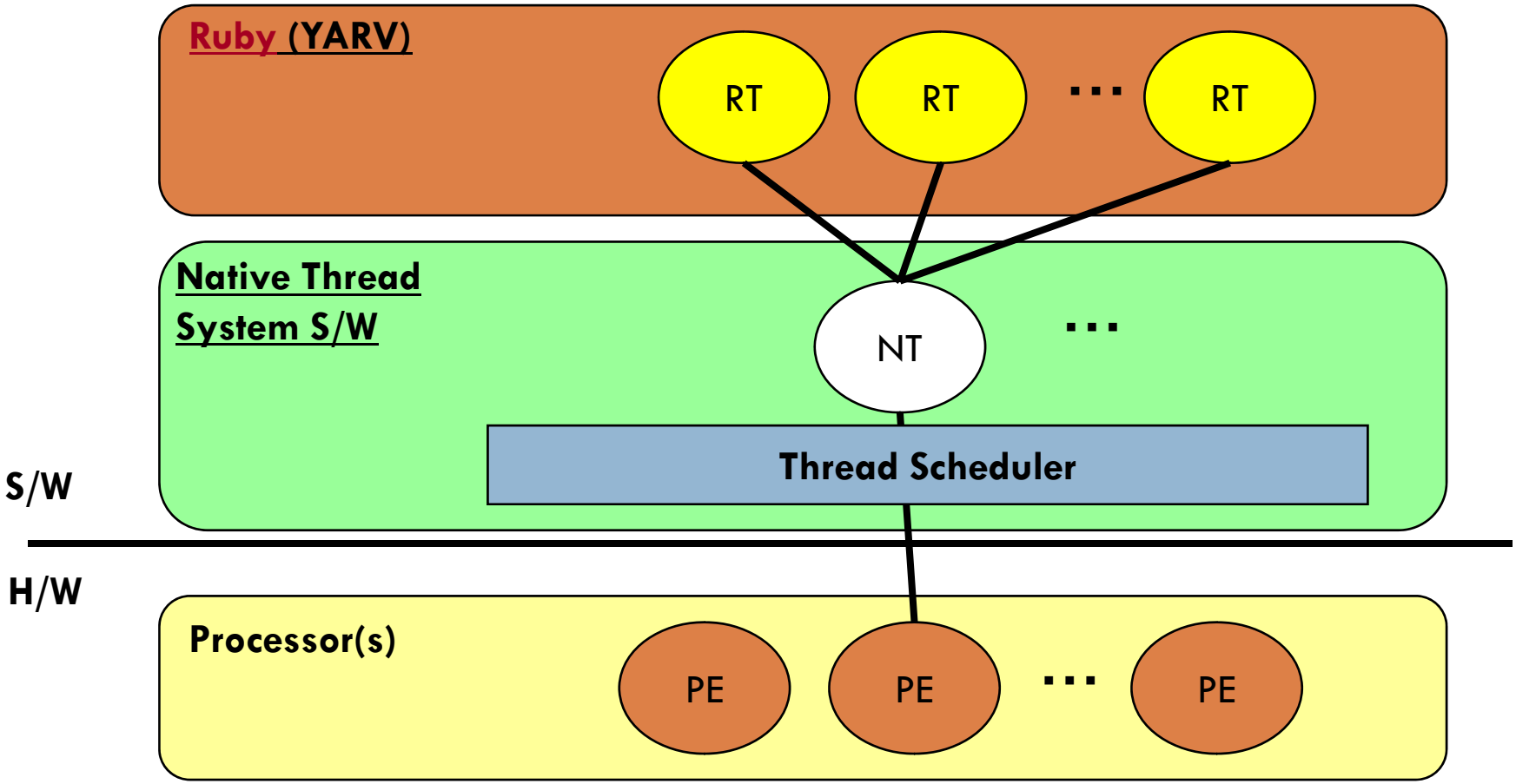
Ruby Continuation Situation

- To Achieve “Generator”
  - Ruby’s loop primitives are “inner-iterator”
    - Call Closure per iteration
  - E.g. same as Python’s “generator”
- Make a toy program (ex: amb)
- To be proud of owning “callcc” method

Implementation of Ruby Thread on Ruby 1.8

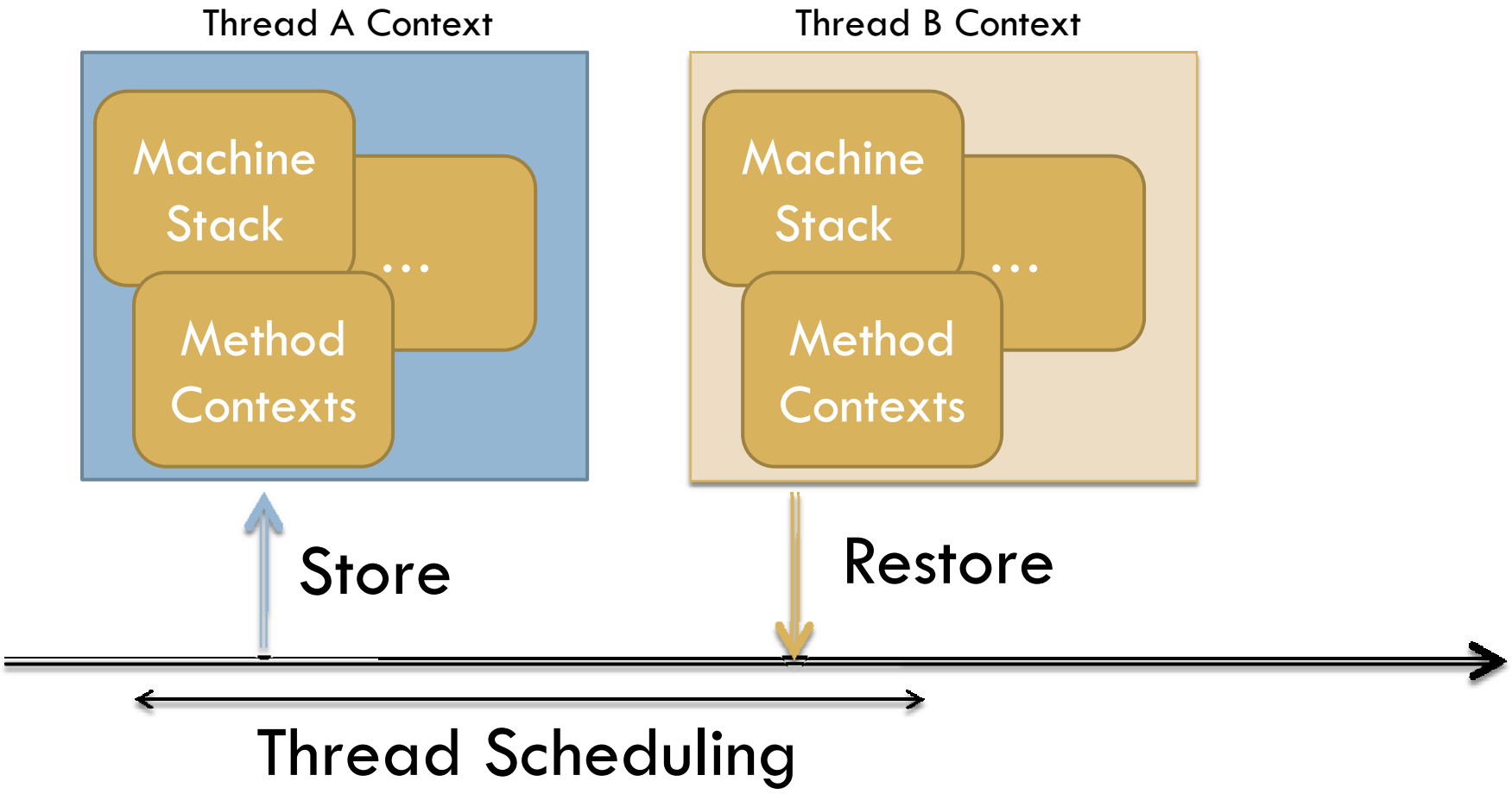
- Userlevel Thread model
- **Copy all machine stack** and swap all Interpreter contexts
- I/O check on thread scheduling

User-level Thread Model



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

Context Switch on Ruby 1.8



Implementation of Ruby Continuation on Ruby 1.8

- Almost same as (1.8 Userlevel) Thread (== wrapper of Thread)
- Shugo Maeda (criminal) proposed that "Thread implementation can be used to make Continuation"
- Matz agreed this proposal (2nd criminal)

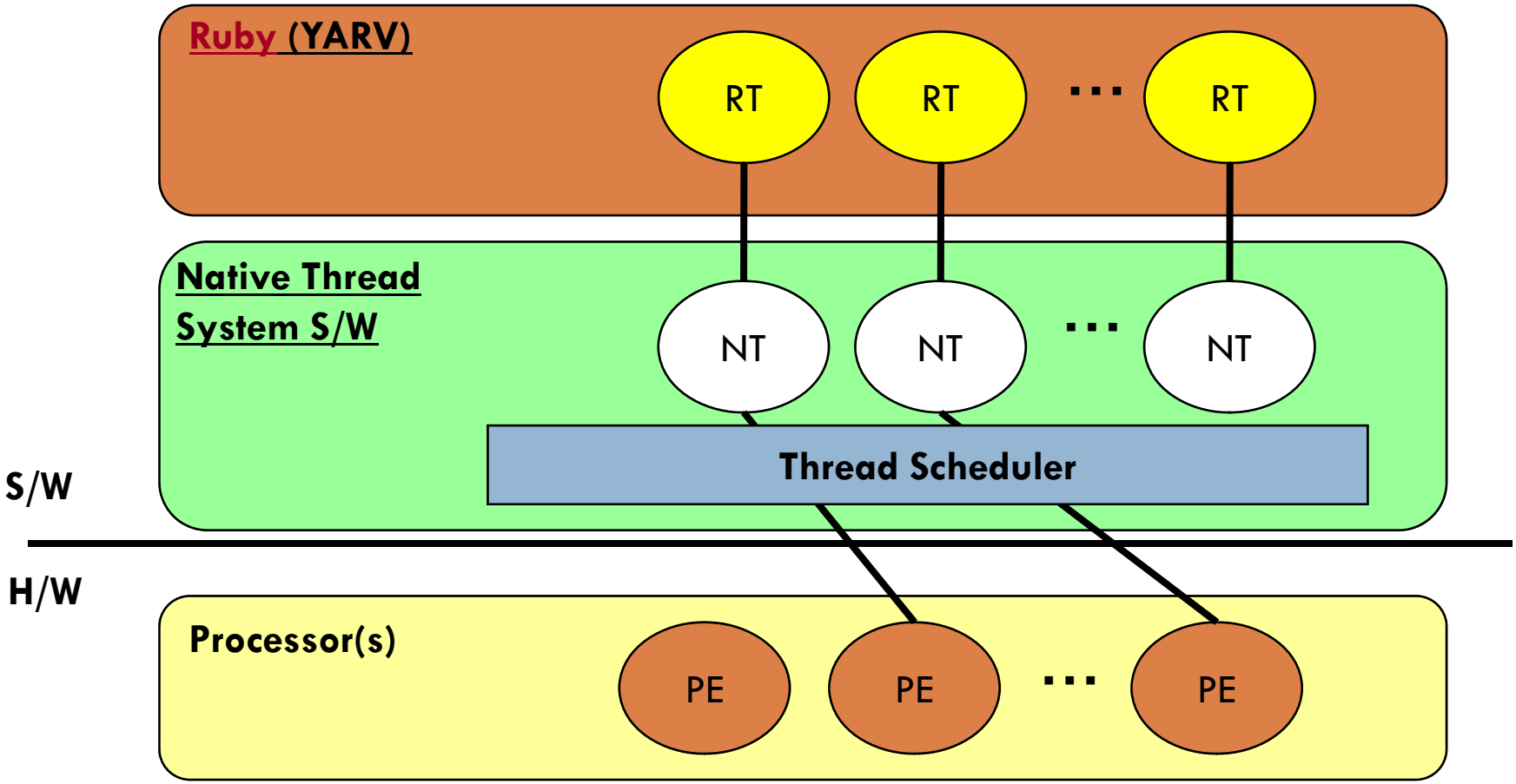
Problems of Ruby Continuation

- **Inconsistency on C Function's Context and Ruby's Context**
  - Save all "Machine Stack"
  - Produce Many Many BUG reports
    - [SEE blade]
- **NO "dynamic-wind" feature**
  - Can't make safety program with Continuation

Implementation of Ruby Thread on Ruby 1.9

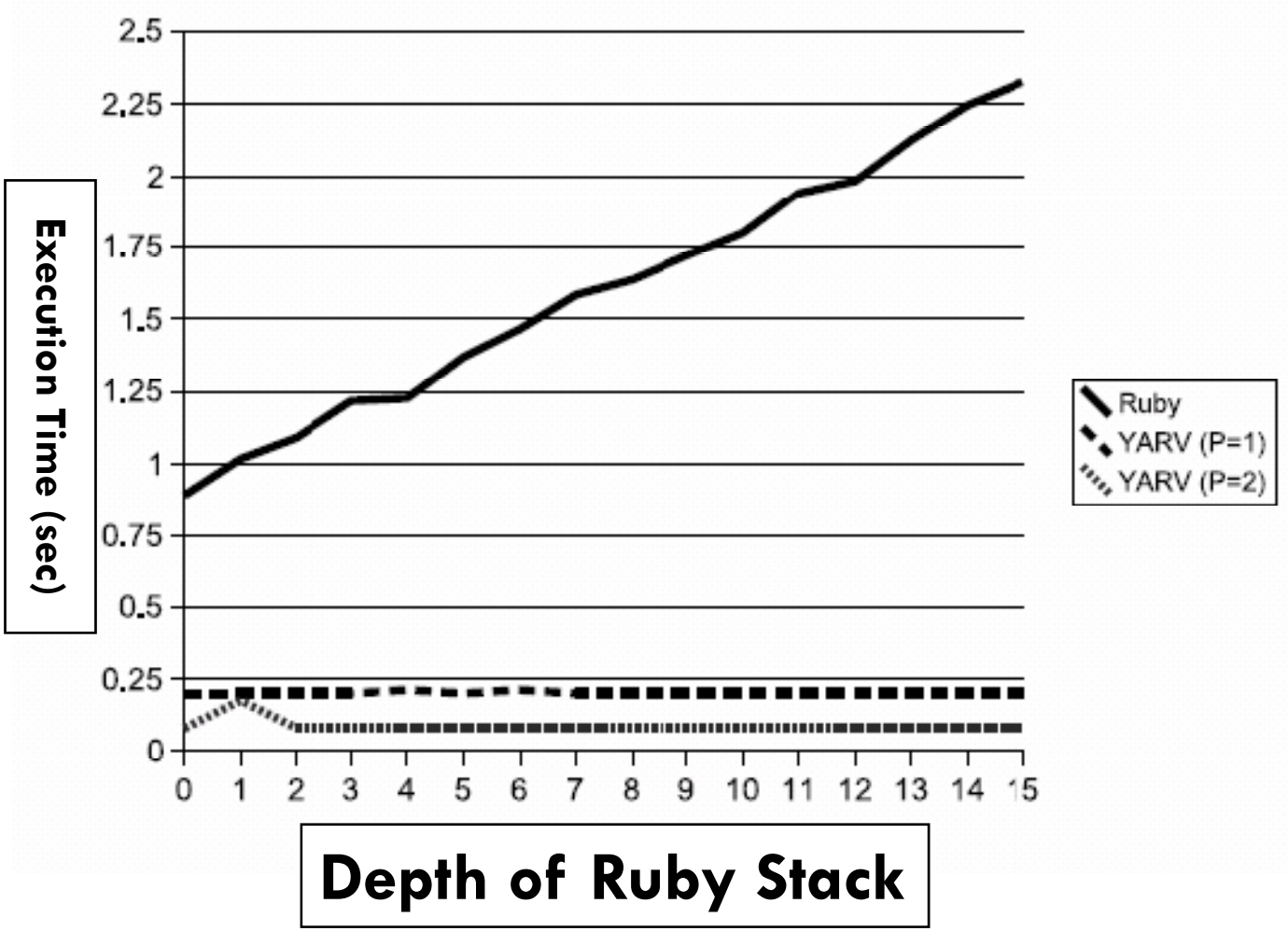
- Using Native Thread
- Can't Run in Parallel
  - With Giant VM Lock
  - CRuby has many "Thread-unsafe" C functions

1:1 Thread Model



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

FYI: Context Switch Performance



Implementation of Continuation on Ruby 1.9

- Make 1.8 Userlevel Thread without Thread Scheduler

Solution against Continuation issues on Ruby 1.9

- To use “callcc” method, need a library
  - require ‘continuation’
    callcc{|…| …}
  - Require library is only enable “callcc” method
  - At last, I want to kick out to gem
- Support “Fiber”
  - Fiber == Coroutine
  - FAQ: Why Fiber? → It sounds good
  - **Generator can be impl. with Fiber**

Other Ideas

- Name long method name or library name
  - call_with_current_continuation{|…| …}
  - require 'call_with_current_continuation_feature'

Other Ideas (cont.)

- Add a Agreement sentence to method name or library name (Same as “long name” solution)
  - call_with_current_continuation_if_you_call_this_method_this_interpreter_we_cant_guarantee_normal_execution{|…| …}
  - require ‘call_with_current_continuation_feature_if_you_call_this_method_this_interpreter_we_cant_guarantee_normal_execution’

Summary

- Ruby's Full-Continuation Considered Harmful
  - Produce many many bugs
  - Nobody use "callcc" method except experimental code
- Ruby 1.9 or later solves this issue with kick out "callcc" feature to Library
  - Ruby 2.0 includes this library?

Thank you for your attention!
Any Questions?

ささだ こういち
Koichi Sasada
ko1@atdot.net