Building the Ruby Interpreter

What is easy and what is difficult?

Koichi Sasada

ko1@heroku.net

2014

Very important year for me

10th Anniversary

10th Anniversary

YARV development (2004/01-)

Ruby no Kai (2004/07-)

Rubyist Magazine (2004/09-)

10th Anniversary

Continuous efforts on development of Ruby

Today's talk

Ruby development

Easy part

何が簡単なの？

Difficult part

何が難しいの？

Who am I?
A Programmer

- CRuby developer (a committer)
  - Interpreter core such as VM, GC, and so on
  - Join 2007-
- Member of Heroku Matz team
  - **Full time** CRuby developer with Matz and Nobu
  - Join at 2012-
- One of the directors of Ruby Association
  - Join at 2013

Who am I?
Contributions

- YARV: Yet Another RubyVM (Ruby 1.9)
- Native Thread strategy (Ruby 1.9)
- Fiber (Ruby 1.9)
- Flonum (on 64bit CPU) (Ruby 2.0)
- New method cache (Ruby 2.0)
- RGenGC: Restricted Generational GC (Ruby 2.1)
- RincGC: Restricted incremental GC (Ruby 2.2?)
- Research projects
- Community activities

Contributions
YARV: Yet Another RubyVM (1.9-)

Ruby (Rails) app

i gigantum umeris insidentes
Standing on the shoulders of giants

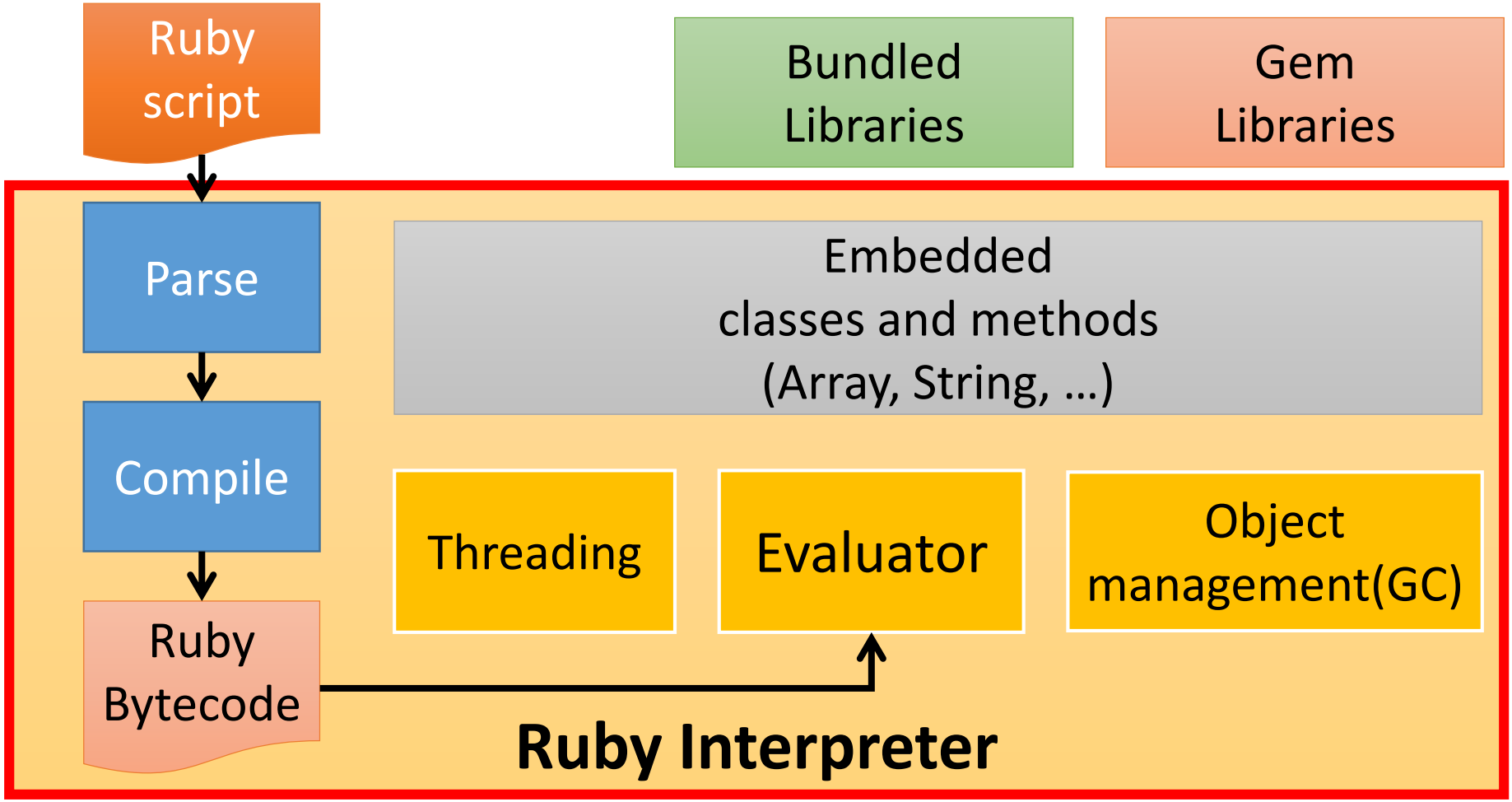
So many gems

such as Rails, pry, thin, … and so on.

RubyGems/Bundler

Ruby interpreter

Contributions
YARV: Yet Another RubyVM (1.9-)

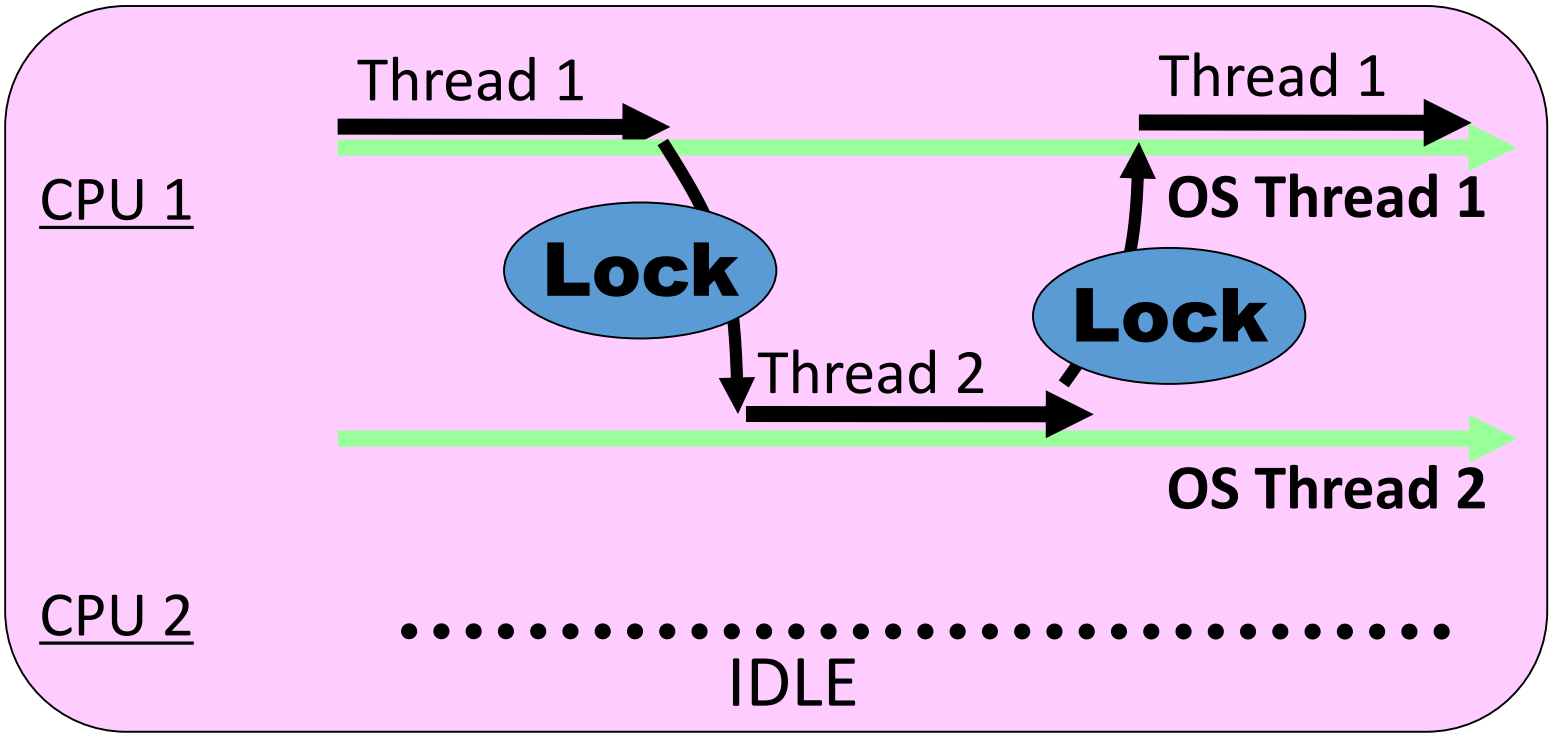


Contributions
YARV: Yet Another RubyVM (1.9-)

- Stack based virtual machine
  - Ruby specific bytecode
  - Compiler Ruby script to bytecode sequence
  - Bytecode interpreter
- Develop at 2004/01/01
  - I was 1st year doctor course student and had plenty time

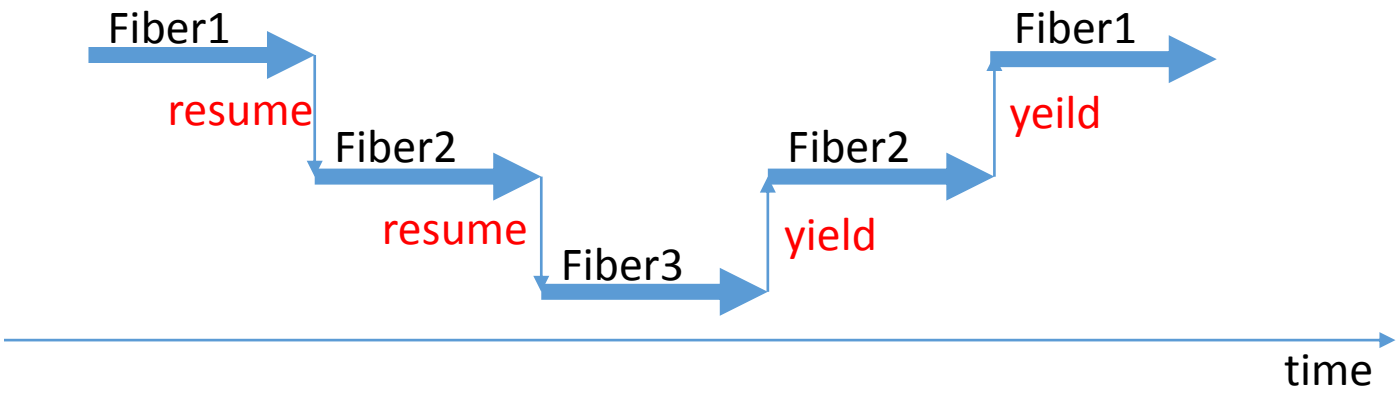
Contribution
Native thread strategy (1.9-)

- Native (OS) threads for each Ruby threads with GVL
- Fast context switch, easy to manage threads



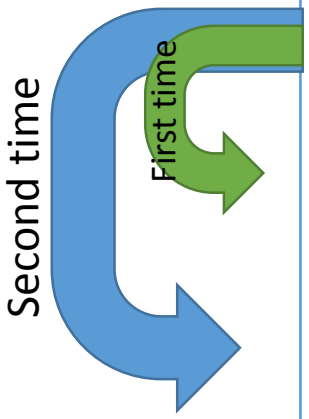
Contribution
Fiber (1.9-)

- Abstraction objects of execution contexts
  - Fiber is from Windows API
  - Cooperative thread
  - Coroutine (or Semi-Coroutine)
- Fast fiber context switch with non-portable methods



Contributions
New method cache (2.0-)

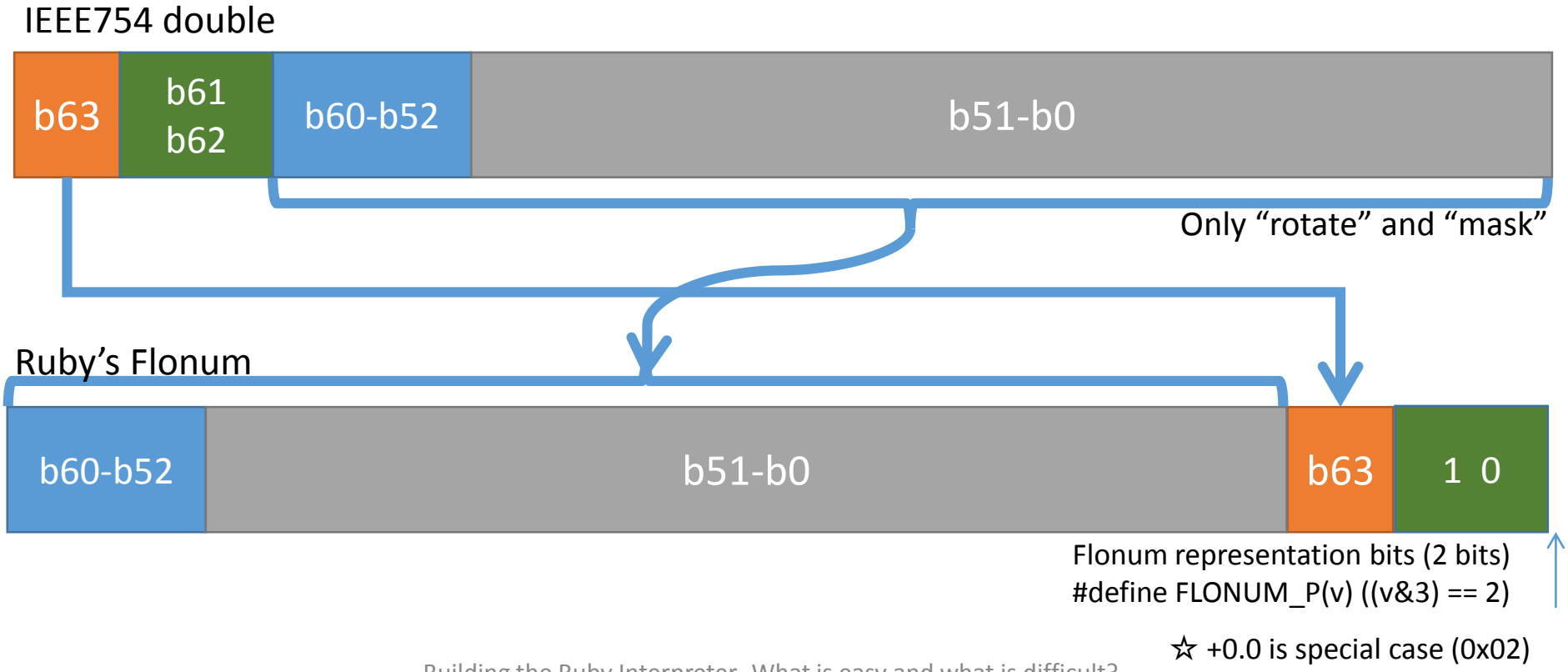
- Store checking results into method cache
- Eliminate method frame building



1. Check caller's arguments
2. Search method `body' `selector' from `klass'
3. Dispatch method with `body'
    1. **Check visibility and arity**
        1. **Cache result into inline method cache**
    2. Push new control frame
    3. Build `local environment'
    4. Initialize local variables by `nil'

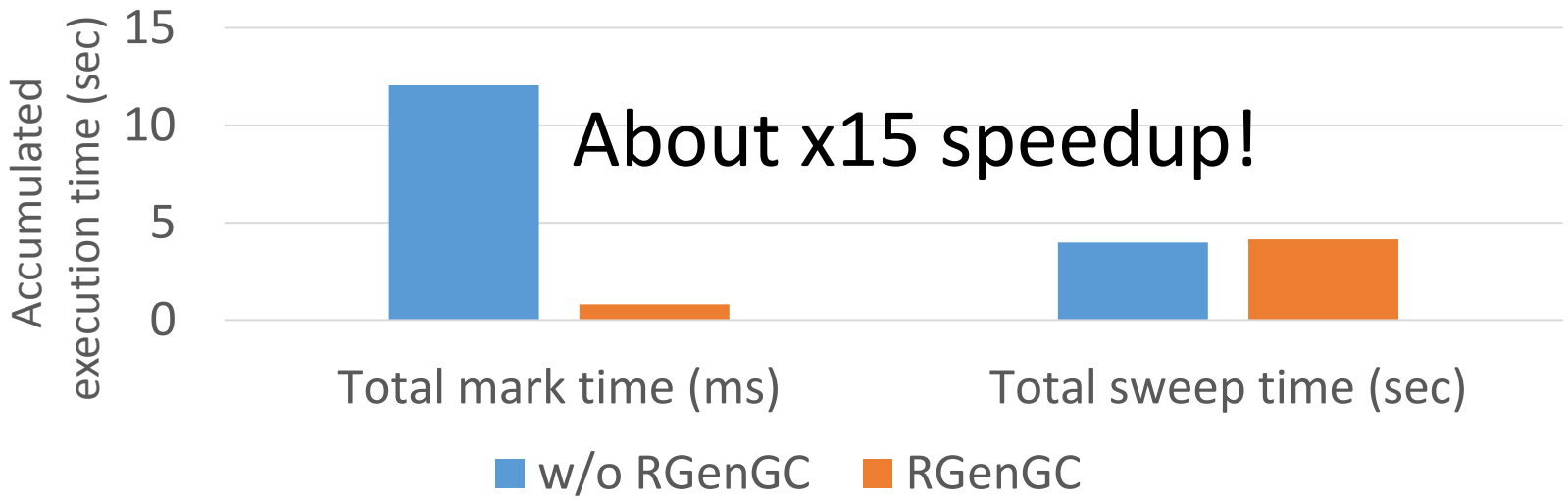
Contributions
Flonum (on 64bit CPU) (2.0-)

- Embedded “double” into VALUE like Fixnum
- About 2 times faster



Contributions
RGenGC: Generational GC for Ruby (2.1-)

- Introduce RGenGC by inventing “WB-unprotected” objects technique and reduce marking time dramatically
- Incremental GC by same technique (for Ruby 2.2)
  - See Rubyist Magazine vol. 0048 or attend RubyConf2014



* Disabled lazy sweep to measure correctly.

Contributions
Research projects

- Performance
  - Ruby to C compiler (3 versions)
  - Ruby to C# compiler
  - Ruby to X10 compiler
  - Regexp compiler
  - Mix Ruby and C program
  - Memory management with mmap
- Parallelization
  - Parallel threads CRuby
  - MVM: Multiple virtual machines
  - Inter-processes shared objects mechanism
- Profilers
  - Memory profiler
  - High-speed profiler
- And others….

Contributions
Community activities

- Nihon Ruby no Kai
  - Director (2004-2011)
  - Rubyist Magazine (2004-)
  - RubyKaigi (2006-2011)
- Ruby Association
  - Director (2012-)
- and other activities
  - see http://www.atdot.net/~ko1/activities/ for other activities

Contributions
Community activities

- Conference in the world
  - 2014/03 RubyConf Philippines 2014, Manila, Philippines
  - 2014/04 RubyConf Taiwan 2014, Taipei, Taiwan
  - 2014/05 Ogasawara, Tokyo, Japan (Honeymoon)
  - 2014/06 RedDotRubyConf 2014, Singapore
  - 2014/07 Deccan RubyConf 2014, Pune, India
  - 2014/08 RubyConf Brasil 2014, São Paulo, Brazil
  - 2014/09 RubyKaigi 2014 (NOW)
  - 2014/10 ?? (No plan, please invite me)
  - 2014/11 RubyConf2014, San Diego, US

http://www.flickr.com/photos/donkeyhotey/8422065722

Today's talk

Ruby development

Easy part

何が簡単なの？

Difficult part

何が難しいの？

Mission of
Ruby interpreter developers

Improve the

Quality

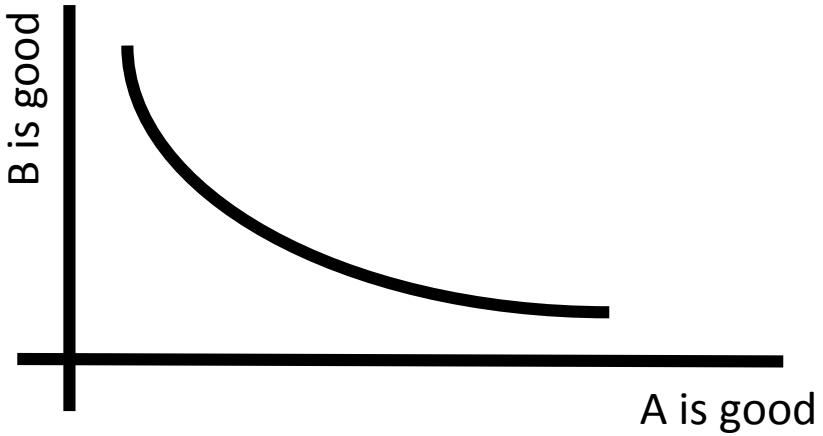
of the Ruby interpreter

Quality

- Reliability / Availability
  - Run Ruby program correctly
  - No bugs!!
- High performance
  - Nobody blames speed-up
- Low machine resources
  - Low memory, low energy, …
- Good compatibility
  - Ruby level and C-API level
- Extensibility
  - Productivity on Ruby interpreter development

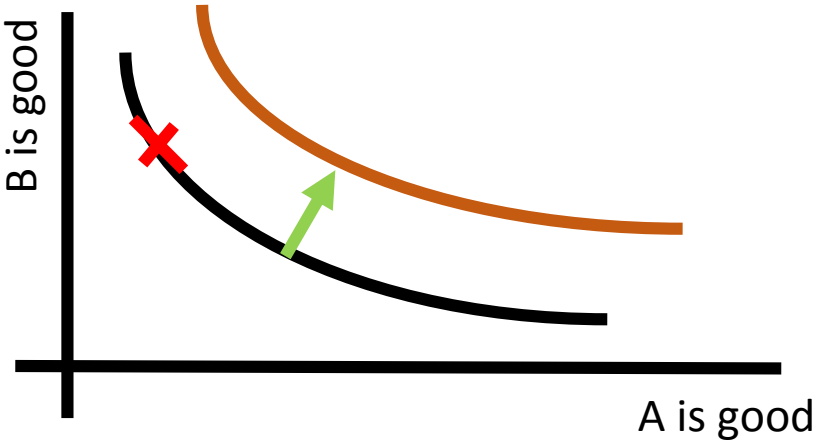
Trade-off

- Many trade-off, for example:
  - Performance <-> Reliability
  - Performance <-> Low resource
  - Performance <-> Compatibility
  - Performance <-> Extensibility



Trade-off

- We engineers/programmers need to:
  - Know trade-off
  - Consider trade-off
  - Overcome trade-off



Overcoming “trade-off” technology

- Object-oriented scripting language Ruby
  - Improve “Productivity” by overcoming trade-off between “Language power” and “Easy-to-read/write”
- Performance improvements of Ruby interpreter
  - Improve “Performance” by overcoming trade-off between “Productivity” and “Performance”

Ruby's Performance

Serial execution performance

Parallel execution performance

GC performance

Serial execution

- [EASY]
  - Introduce virtual machine (done)
  - Introduce (simple) JIT (AOT) compilation
- [Difficult]
  - Keep productivity, reliability, compatibility
  - Improve performance with aggressive optimization
  - Interoperability with C codes

Serial execution
Designing simple VM

- Add bytecode incrementally
  - Increase support ruby features
  - For YARV, ruby has an answer set! (test case)
- VM is simple and easy software
  - Loop fetch and execute instructions
- Details are not so easy
  - Implement block data structure is hell
  - But time can solve (maybe…)

Serial execution
Keeping productivity

- (1) VM code needs many similar codes

→ Solution: VM code generator

  - Generate VM related codes from simple definitions
  - No need to write complicated codes

- (2) Manipulate native code for more optimizations

→ Solution: similar code generation technique (planning)

Serial execution
Aggressive optimization

- For meaningful speed, aggressive optimizations are needed
  - Method/block inlining
  - Constant folding
  - Partial redundancy elimination (PRE)
  - Lambda lifting
  - … (many well-known traditional optimizations)
- Ruby is highly dynamically programming language
  - Method redefinition
  - Accessing local variables via “eval” method
- Key technique is [DE-OPTIMIZATION]
  - Revert aggressive optimizations
  - It is difficult to revert from mangled states to plain states dynamically

Serial execution
Interoperability with C codes

- C code is low-level, faster than Ruby's code
- However, C code doesn't have internal details which aggressive optimization requires
- Most of Java class libraries are written in Java program → Rubinius way (Ruby in Ruby)
- Ideas
  - (1) Rewrite with ruby
  - (2) Write annotations to C code
  - (3) Analyze C code with LLVM infrastructure and so on
  - (4) Mix C code with Ruby code

Performance

Serial execution performance

Parallel execution performance

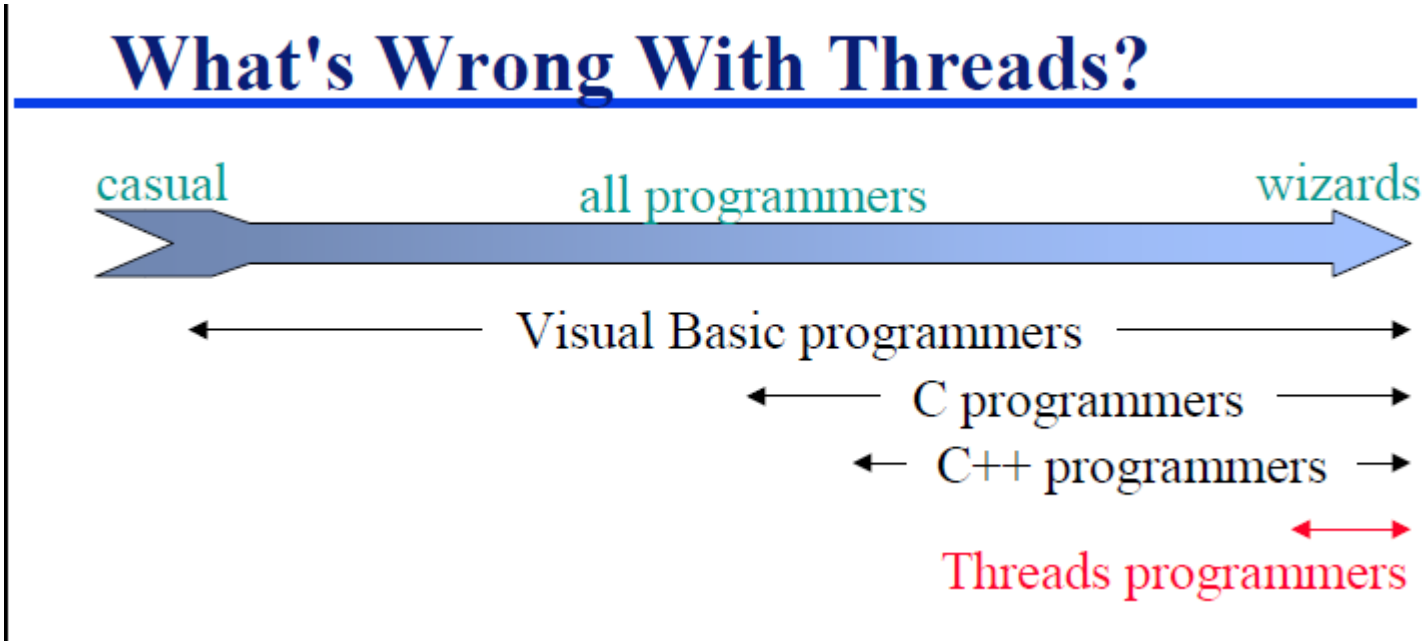
GC performance

Parallel execution

- [EASY]
  - Providing parallel threads
- [Difficult]
  - **Provide good programming experience**
    - **Programming model**
    - **Debugger**
  - Good serial performance
    - Need synchronization everywhere
  - Keep code quality, reliability, compatibility
    - Interpreter should be robust
    - Thread programming is needed!

Parallel execution
Provide good programing experience

- “Why Threads Are A Bad Idea (for most purposes)”
  - Quoted from John Ousterhout, 1995



Parallel execution
Provide good programing experience

- Shared everything and need correct synchronizations
- Typical bugs
  - Data race
  - Atomicity violation
  - Order violation
- Non-deterministic nature
  - Bugs are not reproducible
- Terrible experiences make Ruby programming unhappy
  - I hope Ruby programming is happy experience

Parallel execution
Provide good programing experience

- Educate programmers
- Provide good concurrent programming models
- Provide smart debugging tools

Parallel execution Concurrent programming model

- Approaches of other languages
  - Data models
    - Concurrent data: Java (java.util.concurrent)
    - Immutable (functional) data: Functional languages
    - STM: Clojure
    - Type system: D, Haskell
  - Execution models:
    - Actor: Erlang, Scala
    - CSP: Go-lang

- “Can write safe code” vs. “Must write safe code”

Parallel execution
Concurrent programming model

- Trade-off: Performance, Flexibility <-> Reliability
  - “Parallel threads” (shared everything) is very primitive
  - Enable to write best-speed programs
  - Difficult to debug because of non-deterministic nature
- Similar trade-off: free() vs. GC
  - Liberty vs. Restriction
  - Manual free() is high-performance
  - However, Ruby has good enough performance

Parallel execution Concurrent programming model

- Ideas
  - Better inter-process communication
  - MVM: Multiple virtual machines
    - Spawn full set Ruby virtual machine
    - Too big and performance neck
  - Smaller isolated ruby
    - Subset of Ruby
    - mruby?
  - Introduce “owner threads” for each objects
    - Detect “owner thread violation” dynamically
    - Pre-locked objects

Parallel execution
Make program deterministic

- Non-deterministic behavior kills programmers
- Many research on **thread debugging tools**
  - Detecting inter-thread conflicts
  - Change OS scheduler to make programs deterministic

Performance

Serial execution performance

Parallel execution performance

GC performance

GC Performance

- [Easy]
  - Write GC algorithms
- [Difficult]
  - Keep reliability
    - Non-deterministic behavior
  - Keeping compatibility
    - Lack of write-barriers
    - Conservative algorithms
      - Mostly copying/compaction GC

GC Performance
GC algorithm and implementation

- GC algorithms are simple, only a hundred of lines
  - Mark & Sweep
  - Copy, Compaction
  - Reference count
- Other than GC algorithm is very difficult
  - GC algorithm need assumptions, and **need to change all of Interpreter code**
  - Only one bug causes critical bugs
  - Also we need to care compatibility

GC Performance
Example: Write barrier

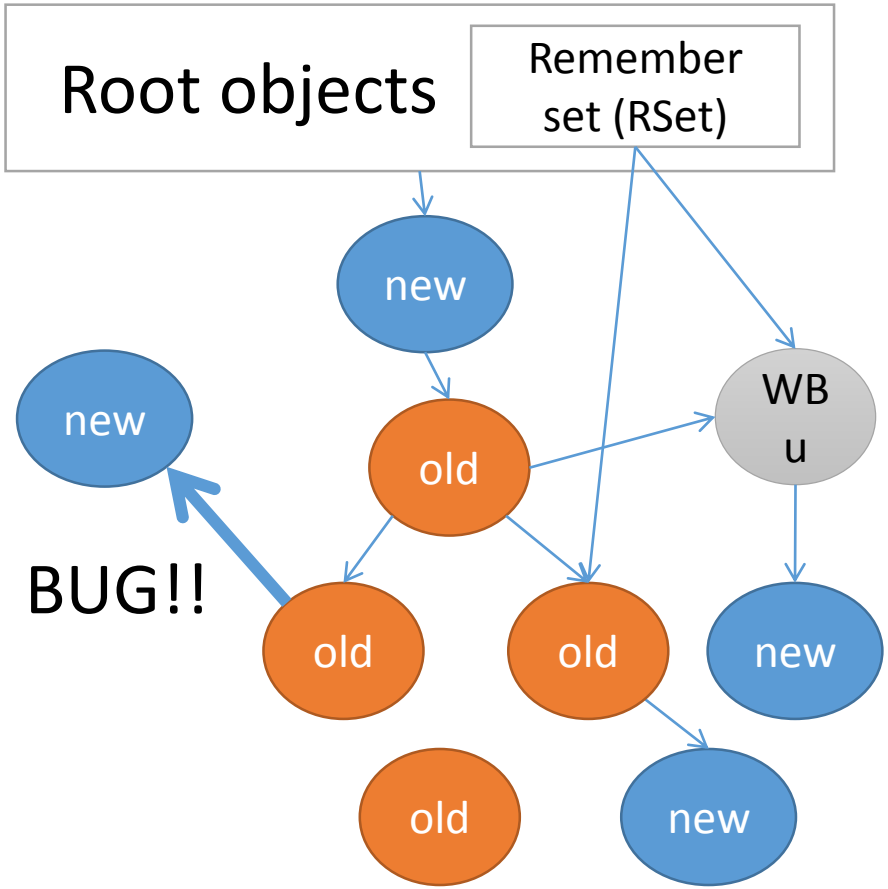
- **Write barrier** technique is required for many GC algorithms, but it is difficult to insert WBs correctly because of compatibility issue
- Solution: Invent new GC algorithm without enough WBs

GC Performance
Reliability

- Non-deterministic behavior
  - GC bugs appear in unexpected place
- Solution: Debugging feature
  - GC.stress: invoke GC many times forcibly
  - Check assertions
    - List up all assertions
    - Check assertions for debug

GC Performance
Example: detect write barrier miss

- Assertion (RGenGC):
  Old objects should not point new objects (without remember set)
- Traverse all objects and build objects relation graph
- Check assertion
- GC.verify_internal_consistency method



Measurement

- [EASY]
  - Measure execution time
- [Difficult]
  - Making periodically observing environment
  - What application should we measure?
  - What measurement should we measure by?

Measurement
Periodical observing environment

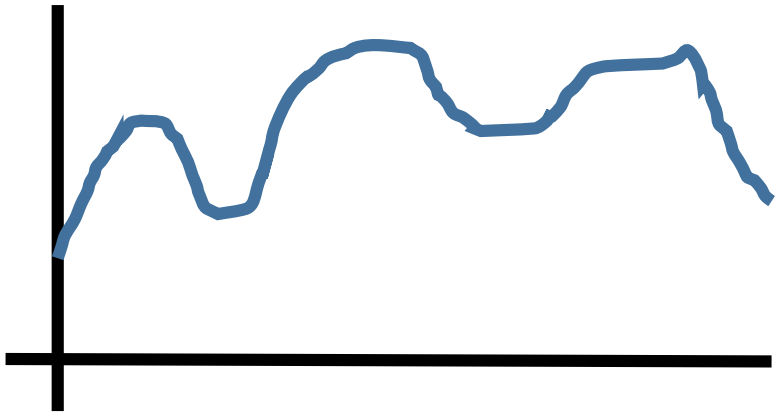
- To measure correct benchmark results, physical environments are required
  - We have a small rack space, cooperation by Prof. Sugaya, Shibaura Institute of Technology
  - Only two machines… ☹
- Ideal resources
  - Multiple OSs (linux, MacOSX, Windows, …), multiple architectures (Intel, ARM, …)
  - Multiple nodes for periodical benchmarking

Measurement Applications

- What applications should we measure?
  - Micro benchmarks
  - Rails application – discourse benchmark

Measurement Index

- What should we use measurement?
- Execution time
  - Which execution time?
  - Include launch time?
- Memory usage
  - Peak value?
  - Average/Median values?

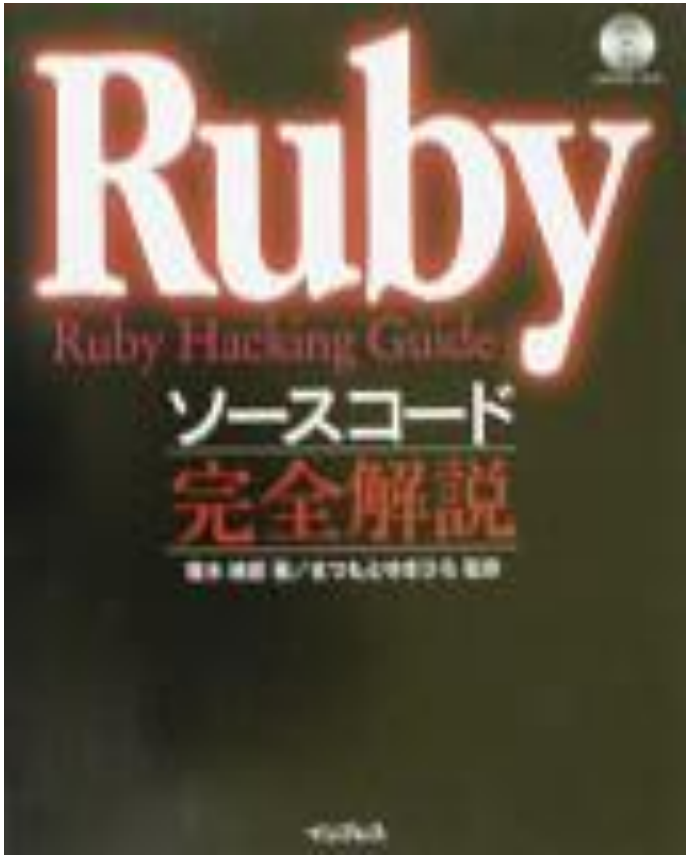


Development community

- [EASY]
  - Become a Ruby committer
- [Difficult]
  - Become a Ruby developer
  - Keep motivation and continuous development
  - Increase Ruby developers

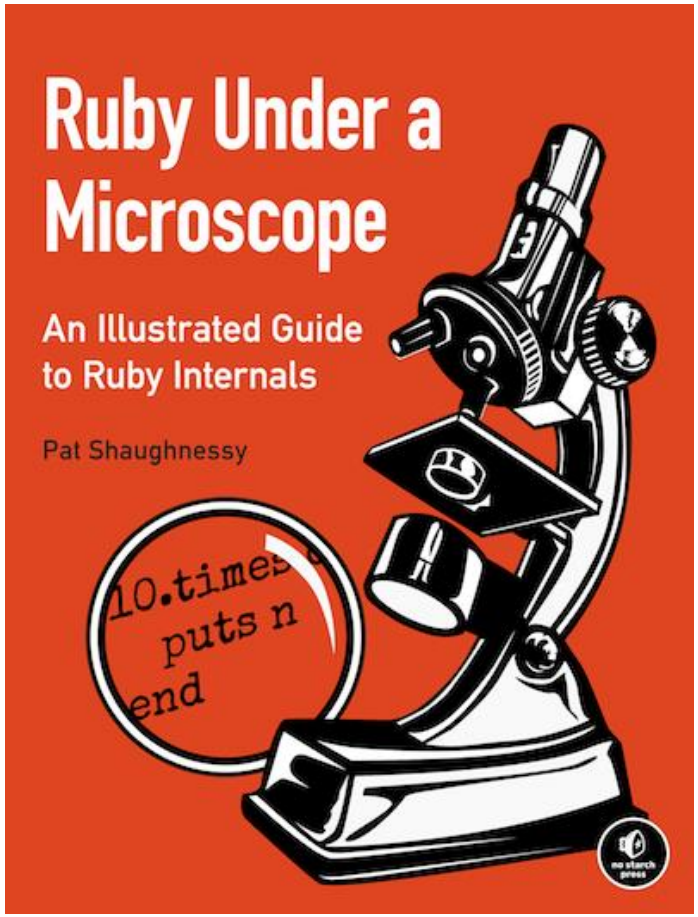
Community
Become a Ruby developer

- Ruby Hacking guide
  - Published in Japanese
  - Translated into English



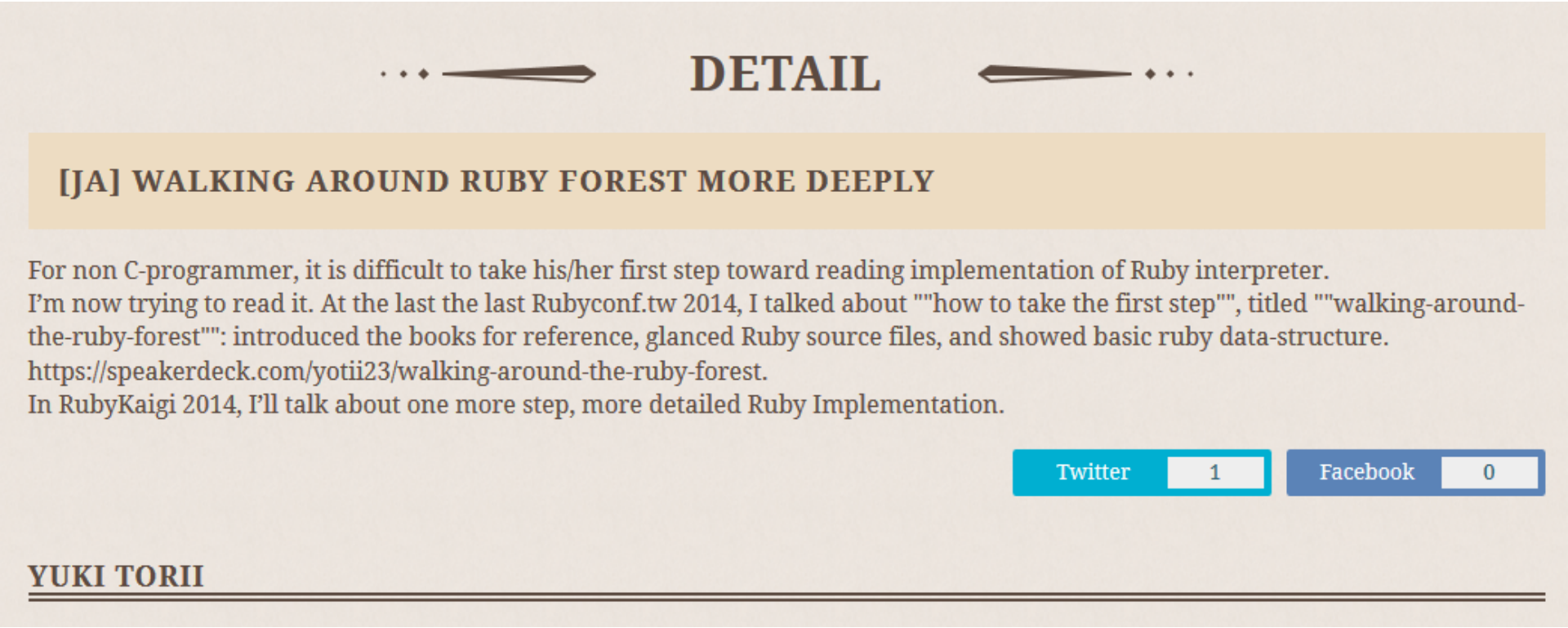
Community
Become a Ruby developer

- Ruby Under a Microscope
  - Published in English
  - (Translating into Japanese)



Community
Become a Ruby developer

- 2014/09/20 15:30- @ Hall B



DETAIL

[JA] WALKING AROUND RUBY FOREST MORE DEEPLY

For non C-programmer, it is difficult to take his/her first step toward reading implementation of Ruby interpreter.
I'm now trying to read it. At the last the last Rubyconf.tw 2014, I talked about ""how to take the first step"", titled ""walking-around-the-ruby-forest"": introduced the books for reference, glanced Ruby source files, and showed basic ruby data-structure.
https://speakerdeck.com/yotii23/walking-around-the-ruby-forest.
In RubyKaigi 2014, I'll talk about one more step, more detailed Ruby Implementation.

Twitter 1 Facebook 0

YUKI TORII

Community
Continuous development

- Survey new technologies
  - Blogs
  - Meet-up
  - Academic papers
- Consideration, Discussion
  - Thinking on the desk
  - Chatting on SNS
  - Developer's meeting
  - Talking at conferences
- Implementation and Evaluation

… and overcome trade-off

Message

We are facing with large blue ocean yet.

Join us for your profession and fun!

Today's talk

Ruby development

Easy part

何が簡単なの？

Difficult part

何が難しいの？

Thank you for your attention

Koichi Sasada

<ko1@heroku.com>

heroku