Speedup Ruby Interpreter

Koichi Sasada

<ko1@heroku.com>

Today’s talk

- Ruby 2.1 and Ruby 2.2
- How to speed up Ruby interpreter?
  - Evaluator
  - Threading
  - Object management / Garbage collection

Koichi Sasada as a Japanese

- Koichi Sasada a.k.a. ko1
- From Japan
- 笹田 (family name) 耕一 (given name) in Kanji character
  - "Ichi" means "1" or first
  - This naming rule represents I'm the first son of my parents
  - Ko"ichi" → ko1

Koichi Sasada as a Programmer

- CRuby/MRI committer
  - Virtual machine (YARV) from Ruby 1.9
  - YARV development since 2004/1/1
  - Recently, improving GC performance
- Matz team at Heroku, Inc.
  - Full-time CRuby developer
  - Working in Japan
- Director of Ruby Association

Ruby Association

The Ruby Association was founded to further development of the programming language Ruby.

The goals of the Ruby Association are to improve relationship between Ruby-related projects, communities and businesses, and to address issues connected with using Ruby in an enterprise environment.

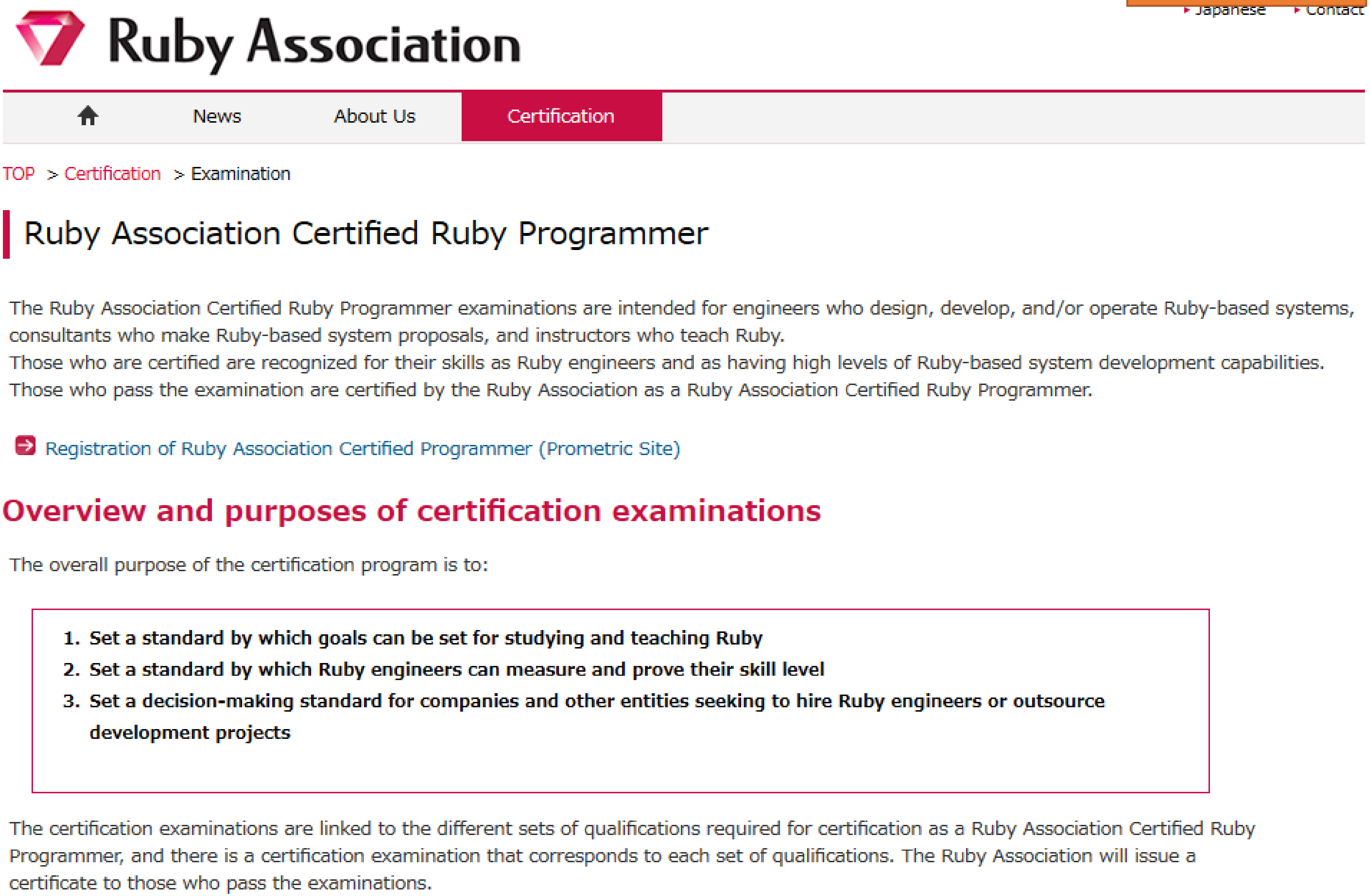
Quoted from http://www.ruby.or.jp/en/

Advertisement

Ruby Association

- Foundation to encourage Ruby dev. and communities
- Activities
  - Ruby programmer certification program
    - http://www.ruby.or.jp/en/certification/examination/ in English
  - Grant project. We have selected **3 proposals** in 2013
  - Ruby Prize
    - To recognize the efforts of “New members” to the Ruby community
    - http://www.ruby.or.jp/en/news/20140627.html
  - Maintenance of Ruby (Cruby) interpreter
    - Now, it is for Ruby 2.0.0
  - Events, especially RubyWorld Conference
    - http://www.rubyworld-conf.org/
  - **Donation** for Ruby developments and communities

Advertisement


Japanese
Contact
Ruby Association
News
About Us
Certification
TOP > Certification > Examination
Registration of Ruby Association Certified Programmer (Prometric Site)

Ruby Association Certified Ruby Programmer

The Ruby Association Certified Ruby Programmer examinations are intended for engineers who design, develop, and/or operate Ruby-based systems, consultants who make Ruby-based system proposals, and instructors who teach Ruby.
Those who are certified are recognized for their skills as Ruby engineers and as having high levels of Ruby-based system development capabilities.
Those who pass the examination are certified by the Ruby Association as a Ruby Association Certified Ruby Programmer.

Overview and purposes of certification examinations

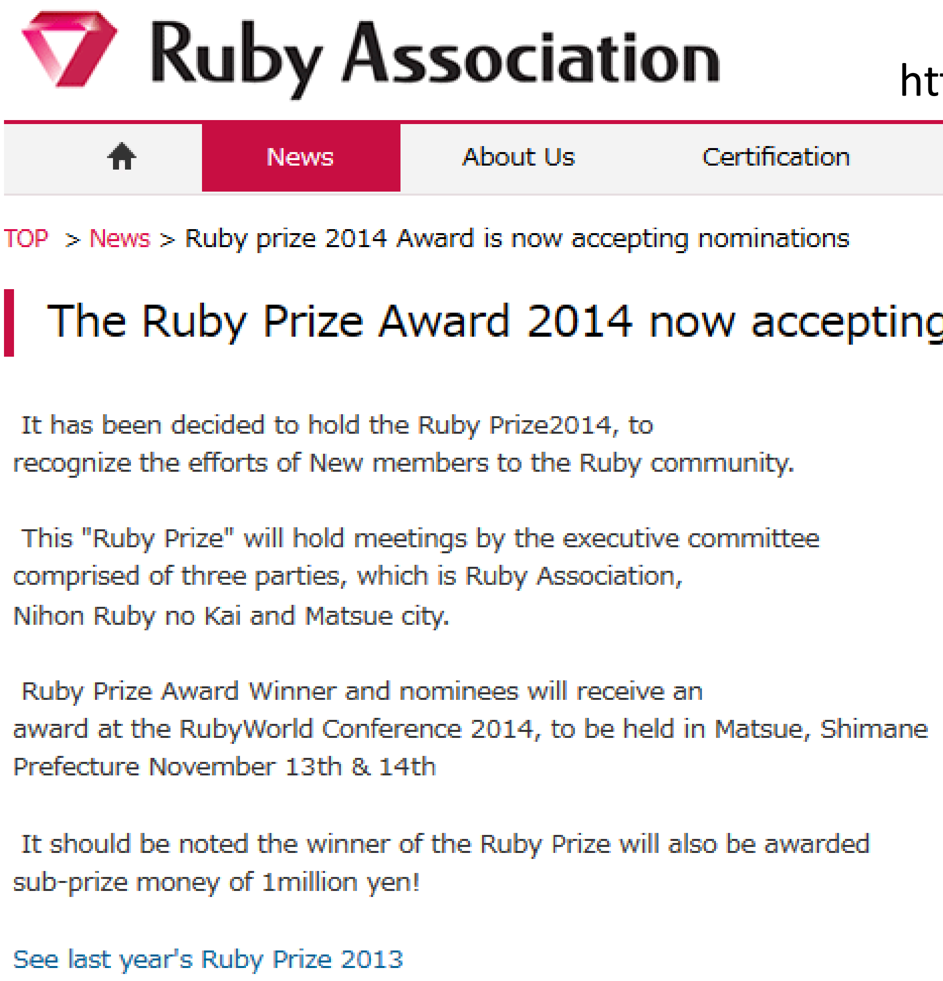
The overall purpose of the certification program is to:

1. **Set a standard by which goals can be set for studying and teaching Ruby**
2. **Set a standard by which Ruby engineers can measure and prove their skill level**
3. **Set a decision-making standard for companies and other entities seeking to hire Ruby engineers or outsource development projects**

The certification examinations are linked to the different sets of qualifications required for certification as a Ruby Association Certified Ruby Programmer, and there is a certification examination that corresponds to each set of qualifications. The Ruby Association will issue a certificate to those who pass the examinations.

Advertisement

http://www.ruby.or.jp/en/news/20140627.html



Ruby Association

TOP > News > Ruby prize 2014 Award is now accepting nominations

The Ruby Prize Award 2014 now accepting nominations

It has been decided to hold the Ruby Prize2014, to recognize the efforts of New members to the Ruby community.

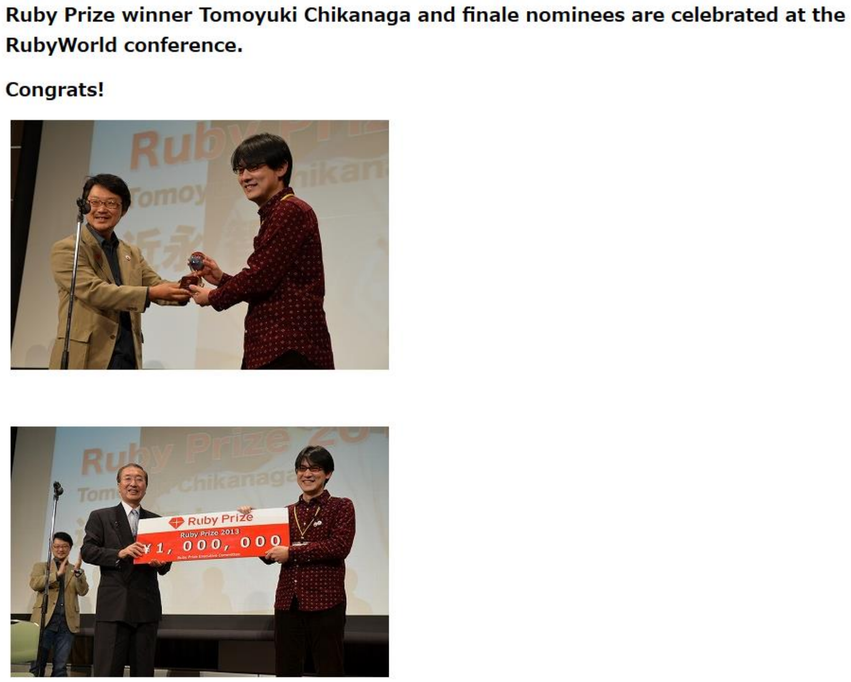
This "Ruby Prize" will hold meetings by the executive committee comprised of three parties, which is Ruby Association, Nihon Ruby no Kai and Matsue city.

Ruby Prize Award Winner and nominees will receive an award at the RubyWorld Conference 2014, to be held in Matsue, Shimane Prefecture November 13th & 14th

It should be noted the winner of the Ruby Prize will also be awarded sub-prize money of 1million yen!

See last year's Ruby Prize 2013

http://www.ruby.or.jp/en/news/20131018e.html

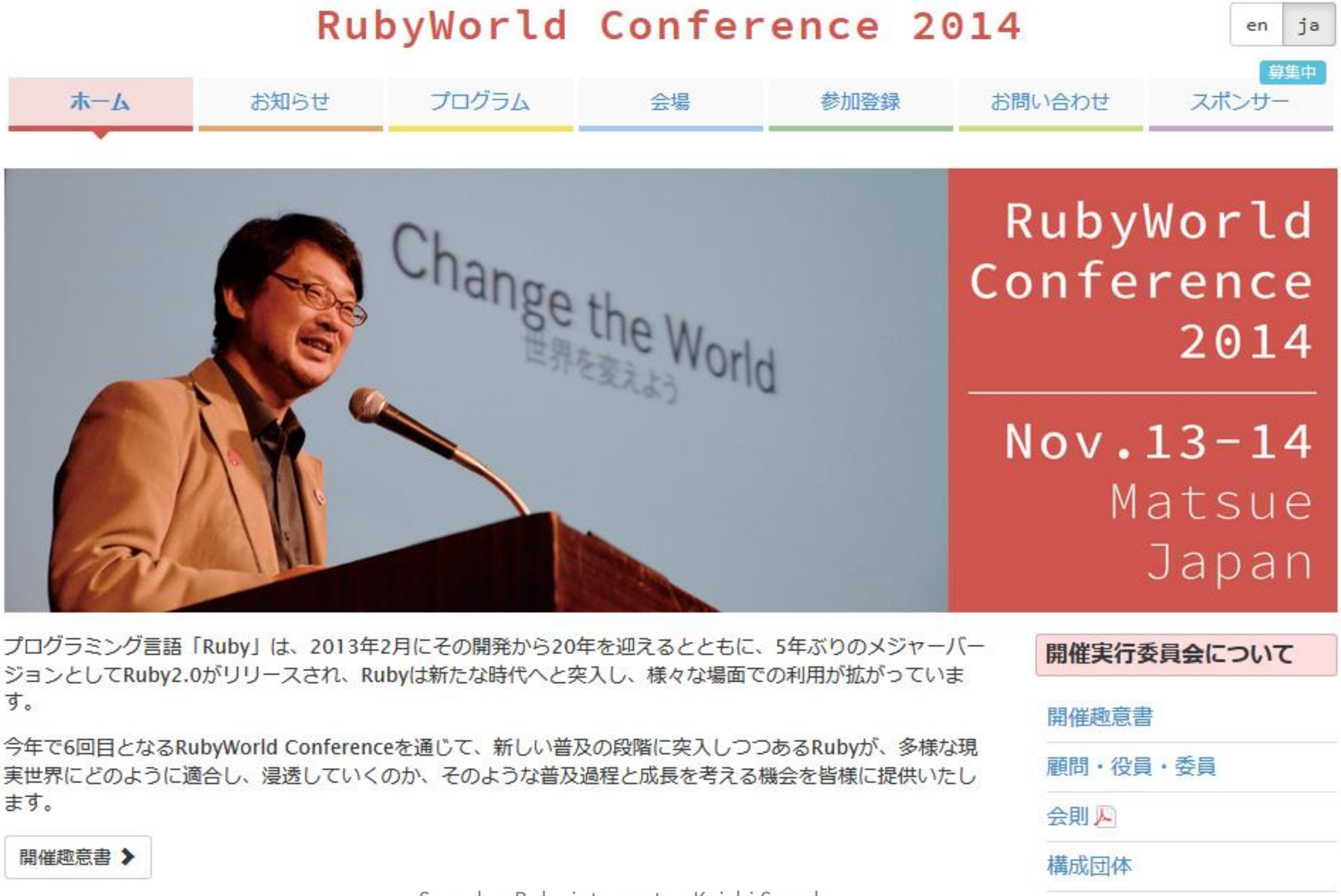


Ruby Prize winner Tomoyuki Chikanaga and finale nominees are celebrated at the RubyWorld conference.

Congrats!

Advertisement

http://www.rubyworld-conf.org/



RubyWorld Conference 2014

en ja

ホーム お知らせ プログラム 会場 参加登録 お問い合わせ 募集中 スポンサー

RubyWorld Conference 2014

Nov.13-14
Matsue
Japan

プログラミング言語「Ruby」は、2013年2月にその開発から20年を迎えるとともに、5年ぶりのメジャーバージョンとしてRuby2.0がリリースされ、Rubyは新たな時代へと突入し、様々な場面での利用が拡がっています。

今年で6回目となるRubyWorld Conferenceを通じて、新しい普及の段階に突入しつつあるRubyが、多様な現実世界にどのように適合し、浸透していくのか、そのような普及過程と成長を考える機会を皆様に提供いたします。

開催趣意書 ❯

開催実行委員会について

開催趣意書

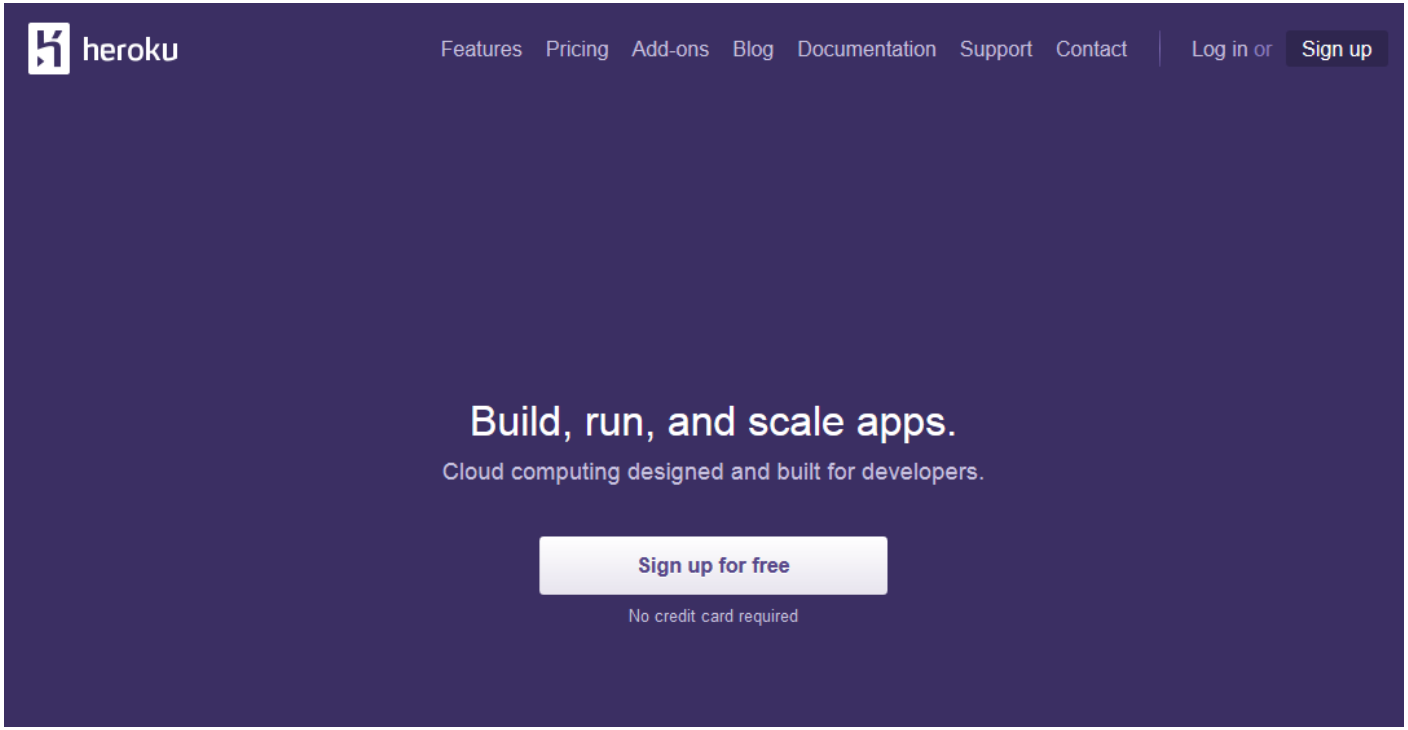
顧問・役員・委員

会則

構成団体

Heroku, Inc. http://www.heroku.com

You should know about Heroku!!

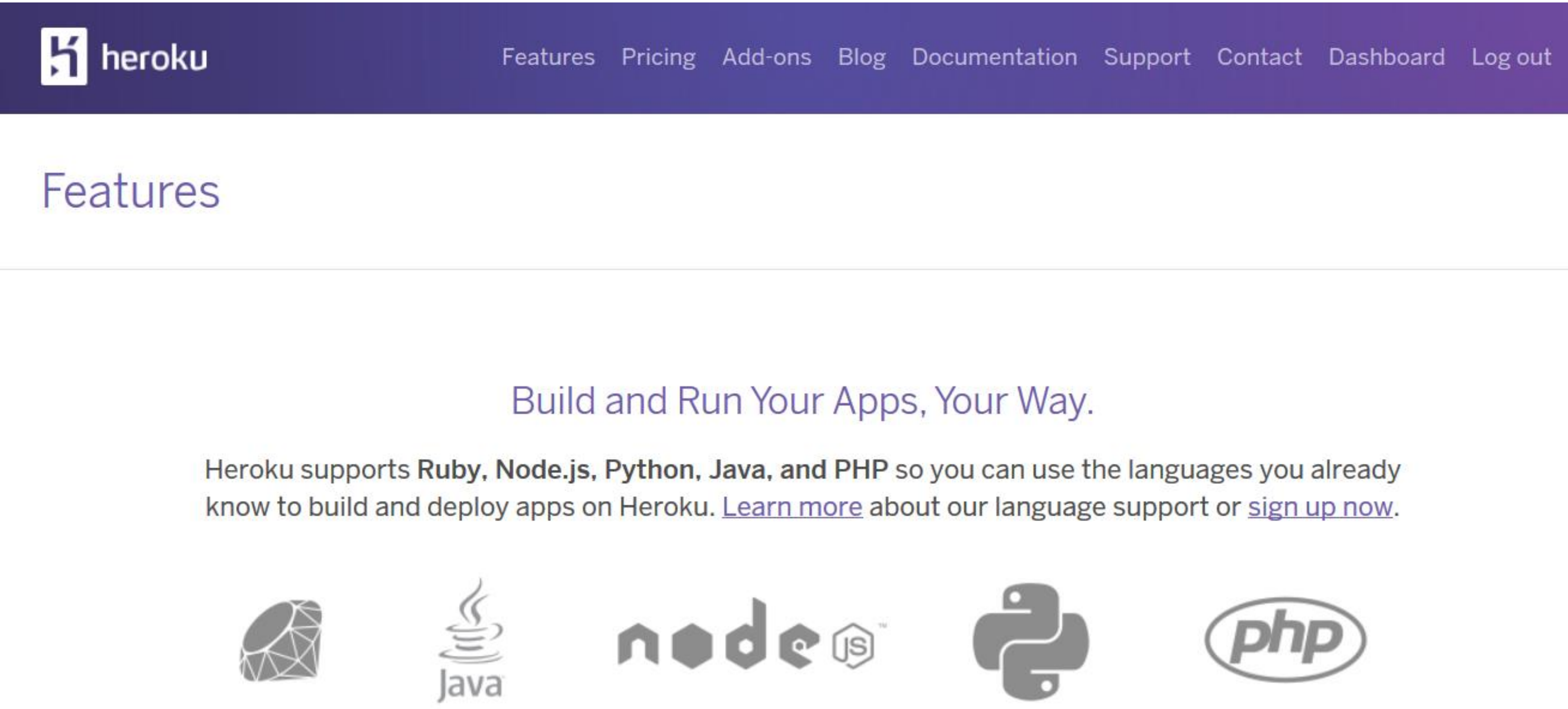


Build, run, and scale apps.

Cloud computing designed and built for developers.

No credit card required

Advertisement



Features

Build and Run Your Apps, Your Way.

Heroku supports **Ruby, Node.js, Python, Java, and PHP** so you can use the languages you already know to build and deploy apps on Heroku. Learn more about our language support or sign up now.

Check https://www.heroku.com/features

heroku

- Heroku, Inc. http://www.heroku.com
- Heroku supports OSSs / Ruby development
  - Many talents for Ruby, and also other languages
  - Heroku employs 3 **Ruby interpreter core developers**
    - Matz
    - Nobu
    - Ko1 (me)
  - We name our group “Matz team”

“Matz team”

Nobu @ Tochigi
Patch monster

札幌
青森
秋田
盛岡
日本海
日本
仙台
新潟
福島
富山
長野
金沢
甲府
東京
国
松江
鳥取
名古屋
大阪
津
静岡
山口
広島
岡山
福岡
大分
松山
高知
長崎
熊本
宮崎
鹿児島

Matz @ Shimane
Title collector

ko1 @ Tokyo
EDD developer

Matz

Title collector

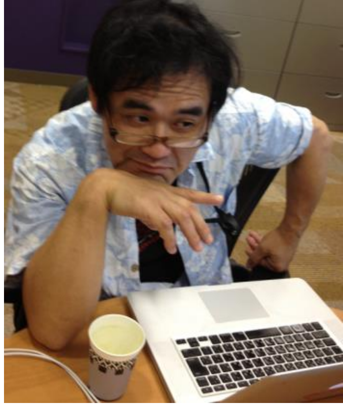
- He has so many (job) title
  - Chairman - Ruby Association
  - Fellow - NaCl
  - Chief architect, Ruby - Heroku
  - Research institute fellow – Rakuten
  - Chairman – NPO mruby Forum
  - Senior researcher – Kadokawa Ascii Research Lab
  - Visiting professor – Shimane University
  - Honorable citizen (living) – Matsue city
  - Honorable member – Nihon Ruby no Kai
  - …
- This margin is too narrow to contain

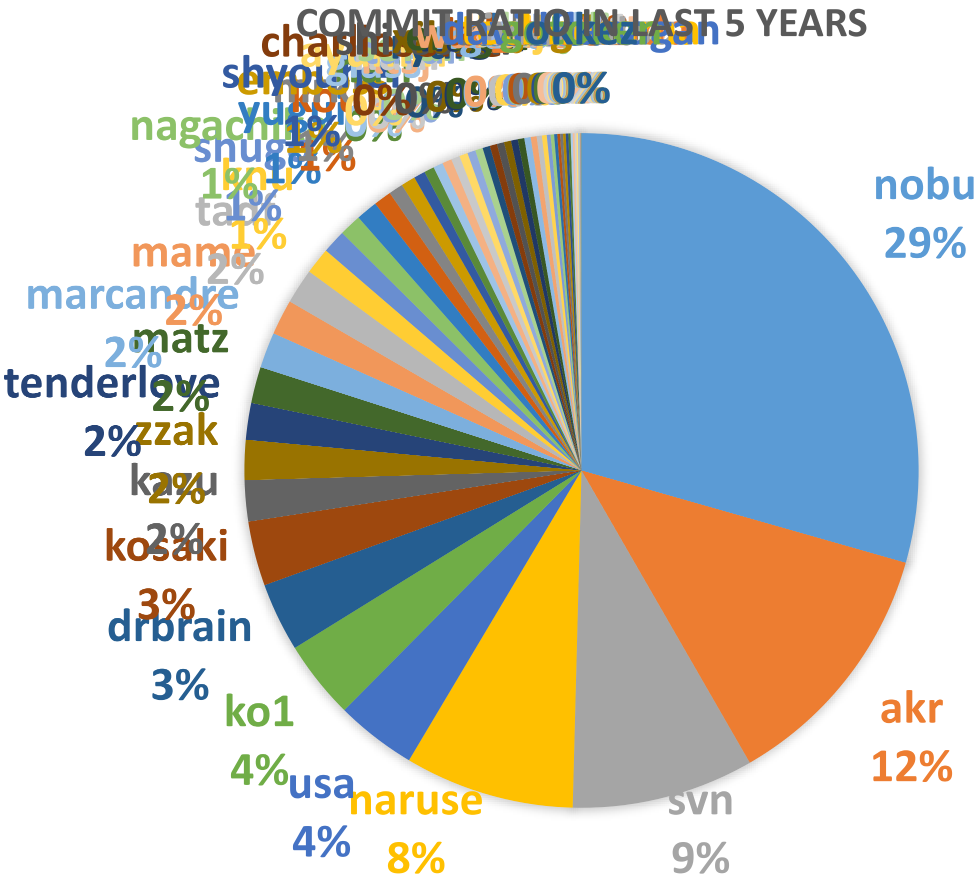
Nobu

Patch monster

- Great patch creator

Nobu is Great Patch Monster

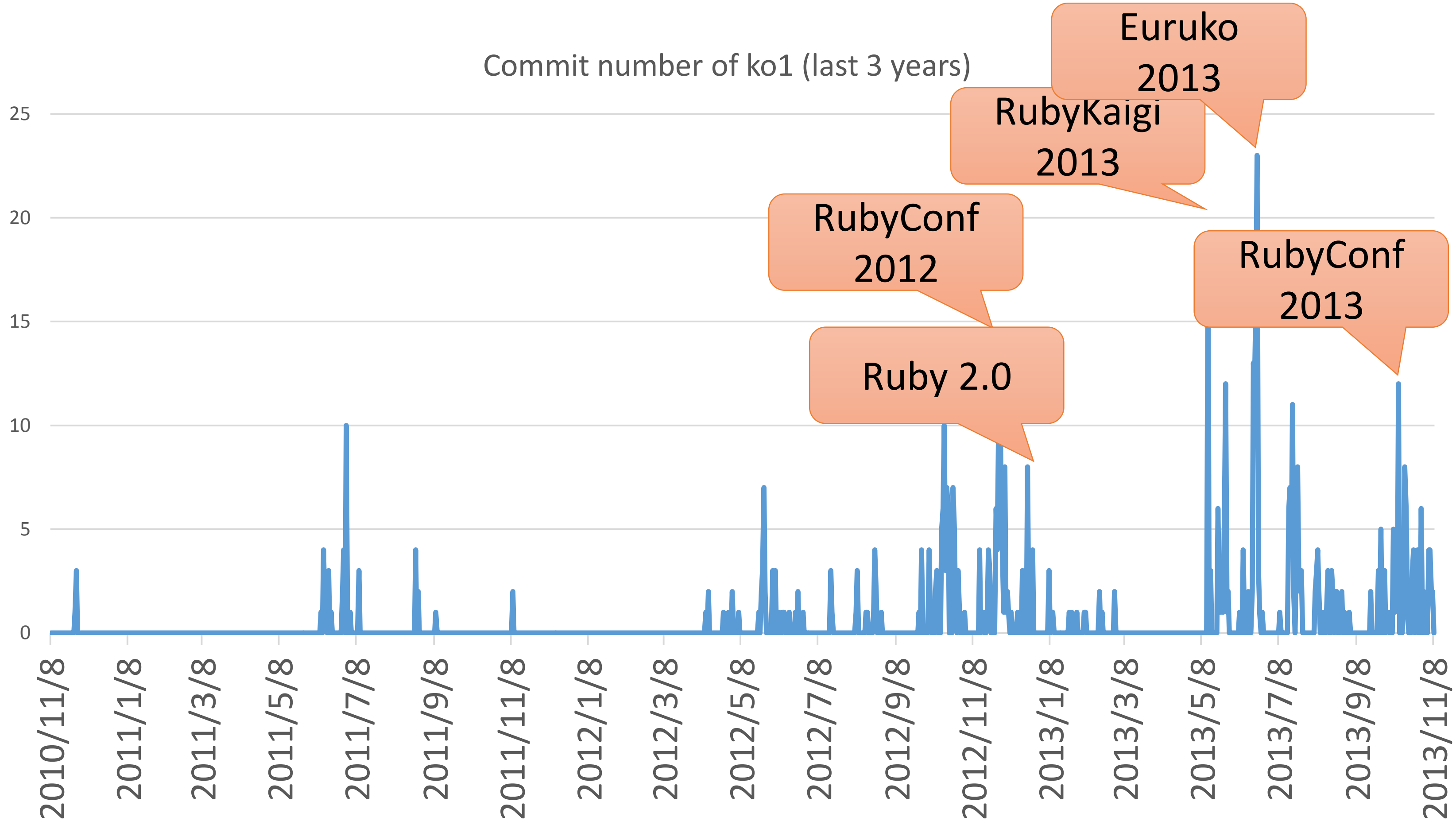




Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014

Ko1

EDD developer



EDD: Event Driven Development

“Mission of Matz team”

Improve quality of next version of CRuby

“Mission of Matz team”

- **Improve quality of next version of CRuby**
  - Matz decides a spec finally
  - Nobu fixed huge number of bugs
  - Ko1 improves the performance

- Next version of CRuby is “Ruby 2.2.0”

http://www.flickr.com/photos/loginesta/5266114104

Ruby 2.1
Current stable

Ruby 2.1
a bit old Ruby

- **Ruby 2.1.0** was released at **2013/12/25**
  - New features
  - Performance improvements
- **Ruby 2.1.1** was released at 2014/02/24
  - Includes many bug fixes found after 2.1.0 release
  - Introduce a new GC tuning parameter to change generational GC behavior (introduce it later)
- **Ruby 2.1.2** was released at **2014/05/09**
  - Solves critical bugs (OpenSSL and so on)

Ruby 2.1 the biggest change Version policy

- Change the versioning policy
  - Drop “patch level” in the version
  - **Major version**: Big language changes (or anniversary)
  - **Minor version**: minor language changes (or annually)
  - **Teeny version**: fixing bugs with compatibility
    - Release new teeny versions about every 3 month
    - Teeny upgrades keep compatibility

Ruby 2.1 New syntax

- New syntaxes
  - Required keyword parameter
  - Rational number literal
  - Complex number literal
  - `def’ returns symbol of method name

```
    **** COMMODORE 64 BASIC V2 ****
 64K RAM SYSTEM  38911 BASIC BYTES FREE
READY.
SHIFTRUNSTOP.CO.UK
?SYNTAX  ERROR
READY.
```

http://www.flickr.com/photos/rooreynolds/4133549889

Ruby 2.1 Syntax
Required keyword parameter

```
def foo(a:1, b:)
  …
end

foo(a: 1, b: 2)       # OK
foo()                 # NG
foo(a: 1)             # NG
```

Ruby 2.1 Syntax
Required keyword parameter

- Keyword argument (from Ruby 2.0.0)
  - def foo(a: 1, b: 2); end
  - `a’ and `b’ are optional parameters
  - OK: foo(); foo(a: 1); foo(a: 1, b: 2); foo(b: 2)
- Required keyword argument from 2.1
  - def foo(a: 1, b:)
  - `a’ is optional, but `b’ is required parameter
  - OK: foo(a: 1, b: 2); foo(b: 2)
  - NG: foo(); foo(a: 1)

Ruby 2.1 Syntax
Rational number literals

```
1/2r #=> Rational(1, 2)
```

Ruby 2.1 Syntax
Rational number literals

- To represent ½, in Ruby "Rational(1, 2)"

  → Too long!!

- Introduce "r" suffix

  ½ → 1/2r

- "[digits]r" represents "Rational([digits], 1)"
- ½ → 1/2r
  - 1/2r #=> 1/Rational(2, 1)
  - 1/Rational(2, 1) #=> Rational(1/2)

Ruby 2.1 Syntax
Complex number literals

```
1+2i #=> Complex(1, 2)
```

Ruby 2.1 Syntax
Complex number literals

- We already have “Integer#i” method to make imaginary number like “1+2.i”
- We already introduced “r” suffix for Rational

  → No reason to prohibit “i” suffix!!
- [digits]i represents “Complex(0, [digits])”
- 1+2i #=> 1+Complex(0, 2)
- 1+Complex(0, 2) #=> Complex(1, 2)

- You can mix “r” and “i” suffix

Ruby 2.1 Syntax
Return value of `def’ syntax

```
def foo()
  …
end
#=> :foo
```

Ruby 2.1 Syntax
Return value of `def’ syntax

- Return value of method definition
  - Method definition syntax returns symbol of defined method name
  - `def foo; …; end’ #=> :foo
- Method modifier methods
  - Example:
    - private def foo; …; end
    - public static void def main(args); …; end

Ruby 2.1 Runtime new features

- String#scrub
- Process.clock_gettime
- Binding#local_variable_get/set
- Bignum now uses GMP (if available)
- Extending ObjectSpace

Performance improvements

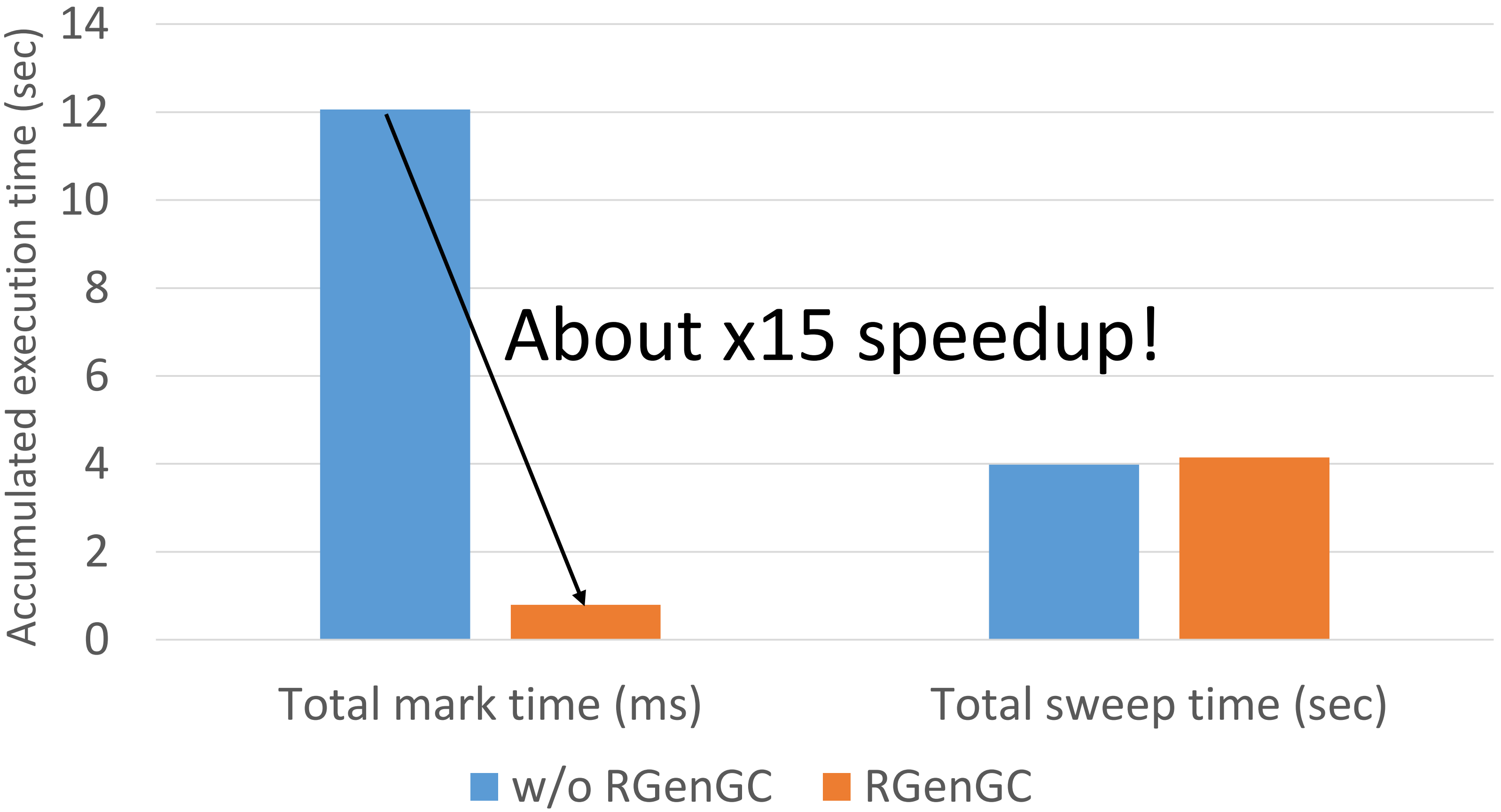
- Optimize “string literal”.freeze
- Sophisticated inline method cache
- Introducing Generational GC: RGenGC

RGenGC: Generational GC for Ruby

- RGenGC: Restricted Generational GC
  - Generational GC (minor/major GC uses M&S)
  - **Dramatically speedup for GC-bottleneck applications**
  - New generational GC algorithm allows mixing “Write-barrier protected objects” and “WB unprotected objects”

  → **No** (mostly) **compatibility issue** with C-exts
- Inserting WBs gradually
  - We can concentrate WB insertion efforts for major objects and major methods
  - Now, most of objects (such as Array, Hash, String, etc.) are WB protected
    - Array, Hash, Object, String objects are very popular in Ruby
    - Array objects using **RARRAY_PTR() change to WB unprotected** objects (called as Shady objects), so existing codes still works.

RGenGC
Performance evaluation (RDoc)



* Disabled lazy sweep to measure correctly.

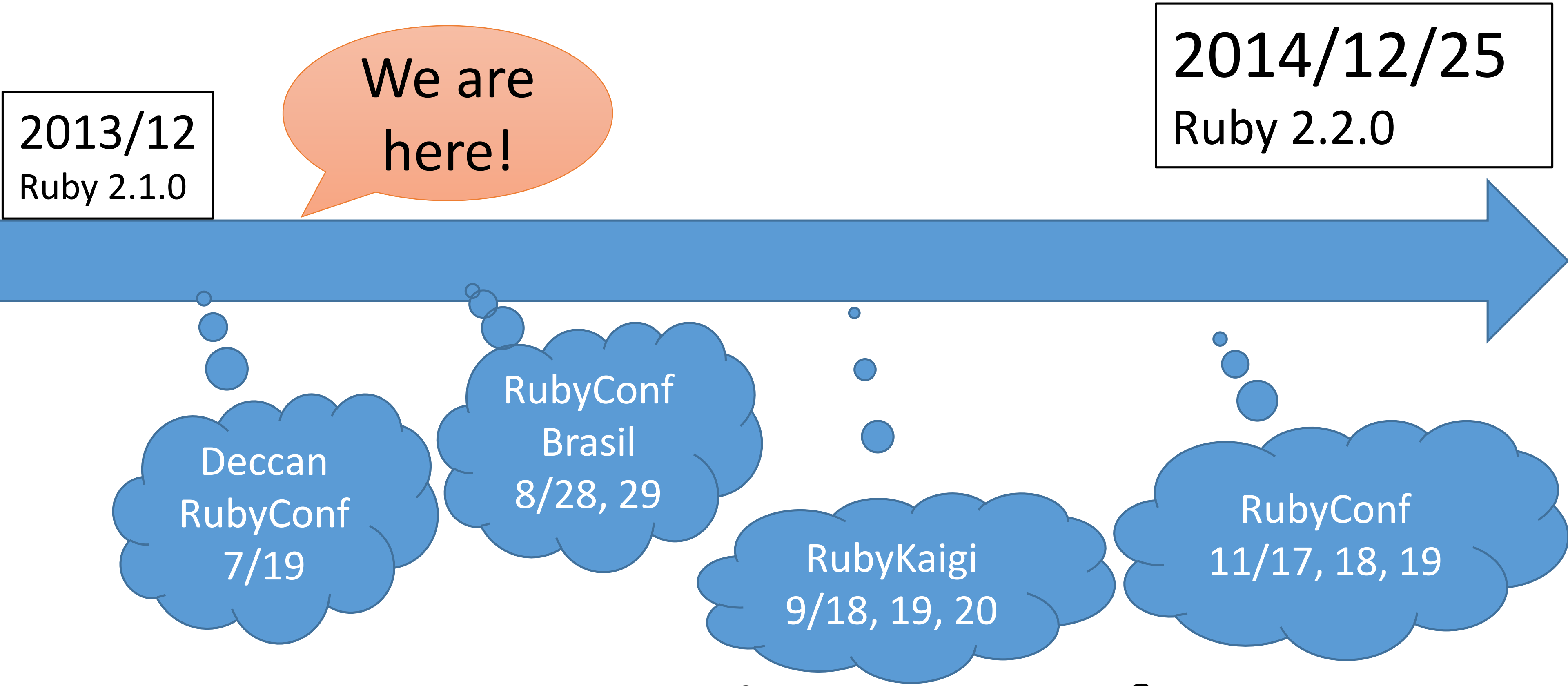
http://www.flickr.com/photos/adafruit/8483990604

Ruby 2.2
Next version

Schedule of Ruby 2.2

- Not published officially
- Schedule draft is available by Naruse-san
    - https://bugs.ruby-lang.org/projects/ruby-trunk/wiki/ReleaseEngineering22

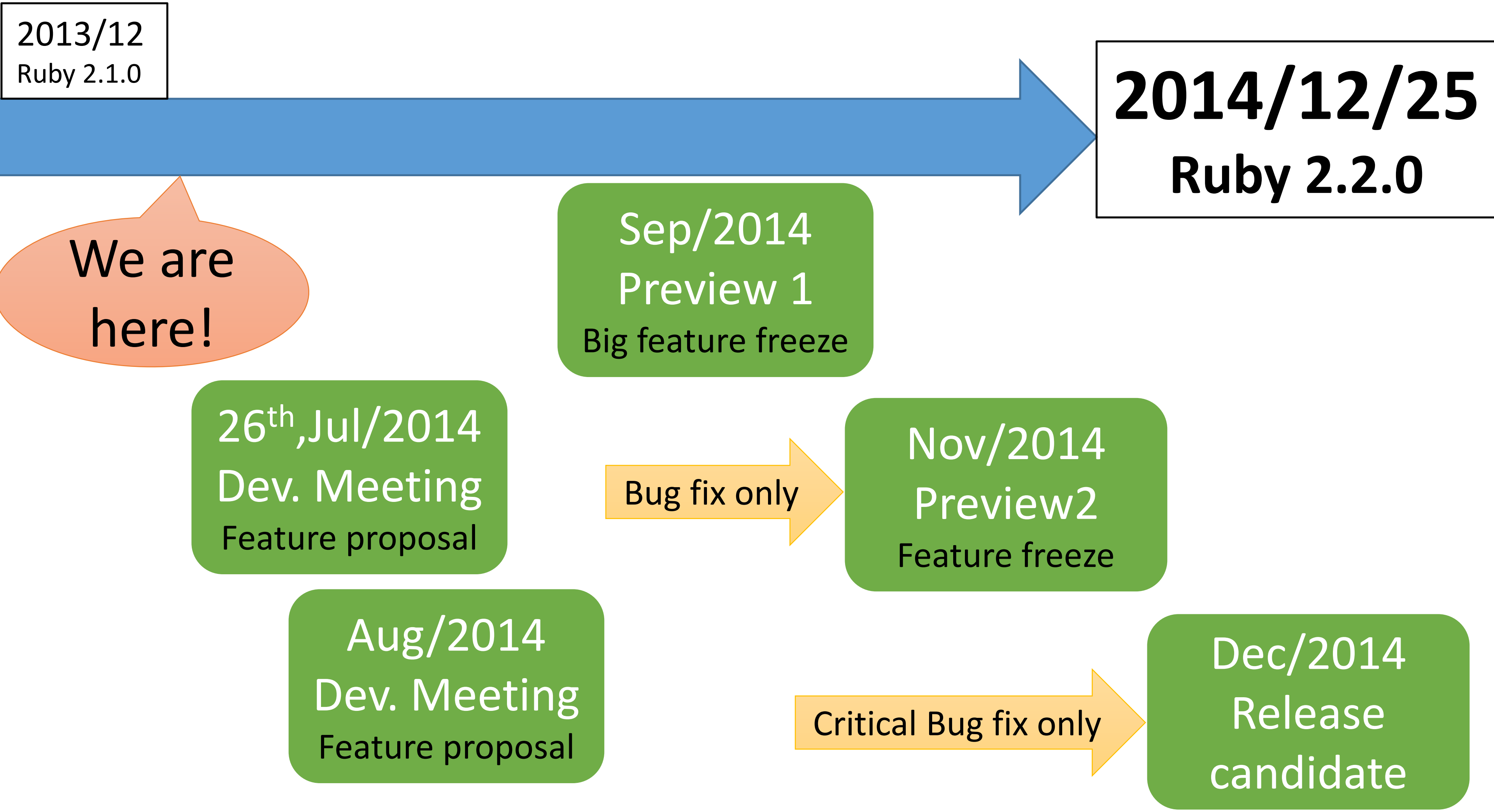
Ruby 2.2 schedule



Events are important for

EDD (Event Driven Development) Developers

Ruby 2.2 (rough) schedule



2013/12
Ruby 2.1.0

2014/12/25
Ruby 2.2.0

We are here!

Sep/2014
Preview 1
Big feature freeze

26th,Jul/2014
Dev. Meeting
Feature proposal

Bug fix only

Nov/2014
Preview2
Feature freeze

Aug/2014
Dev. Meeting
Feature proposal

Critical Bug fix only

Dec/2014
Release candidate

2.2 big features (planned)

- New syntax: not available now
- New method: no notable methods available now
- Libraries:
  - Minitest and test/unit will be removed (provided by bundled gem)

2.2 internal changes

- Internal
  - C APIs
    - Hide internal structures for Hash, Struct and so on
    - Remove obsolete APIs
  - GC
    - **Symbol GC (merged recently)**
    - **2age promotion strategy for RGenGC**
    - **Incremental GC** to reduce major GC pause time
  - VM
    - More sophisticated method cache

Ruby 2.2 internals
Symbol GC

```
1_000_000.times{|i| i.to_s.to_sym}
p Symbol.all_symbols.size
```

```
# Ruby 2.1
#=> 1,002,376
# Ruby 2.2 (dev)
#=> 25,412
```

Ruby 2.2 internals
Symbol GC

- Symbols remain forever → Security issue
  - “n.times{|i| i.to_s.to_sym}”
    creates “n” symbols and they are never collected
- Symbol GC: Collect dynamically created symbols

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

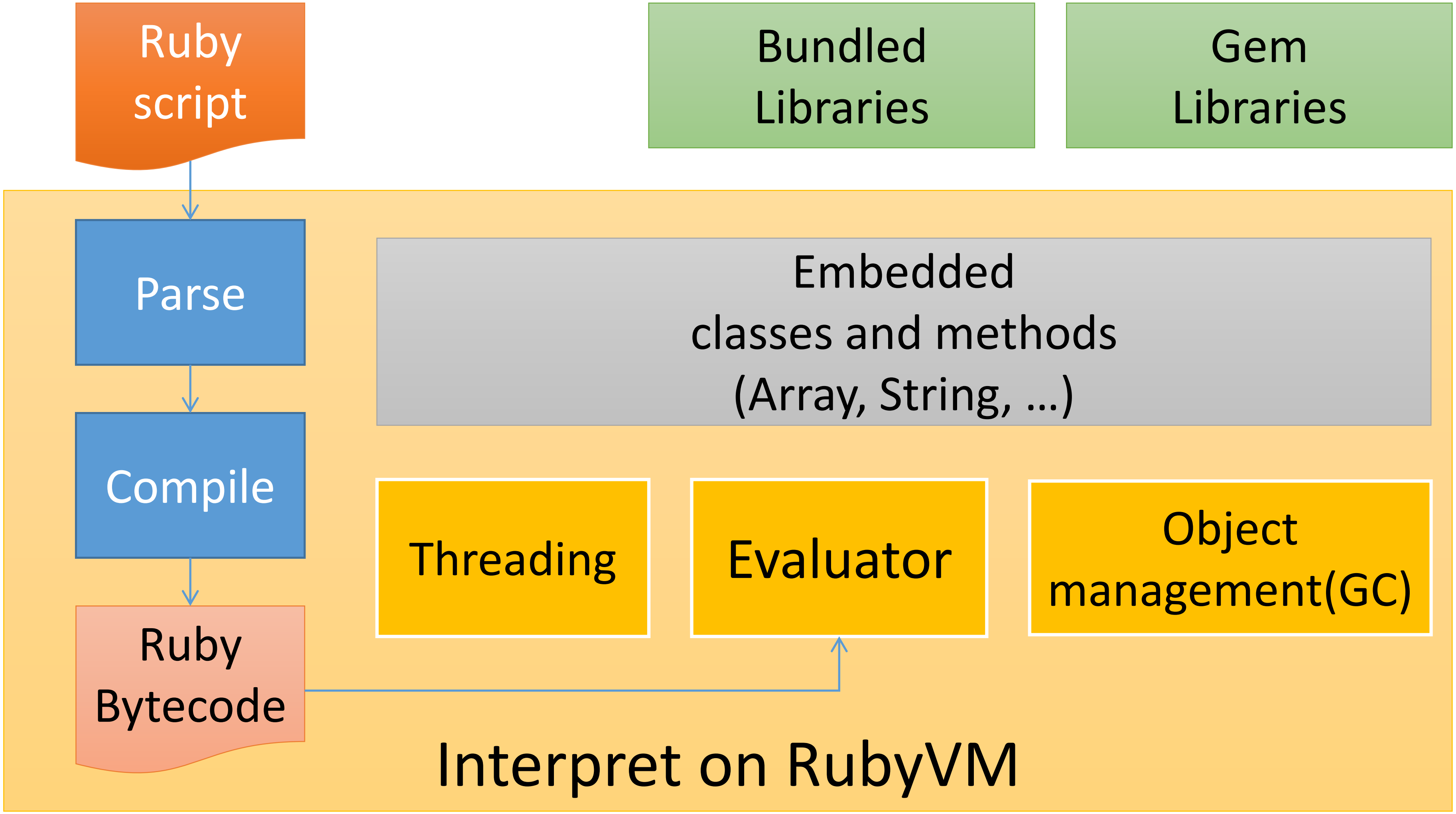
Break

Speedup Ruby Interpreter

How do we speed up Ruby interpreter?

Software consists of many components

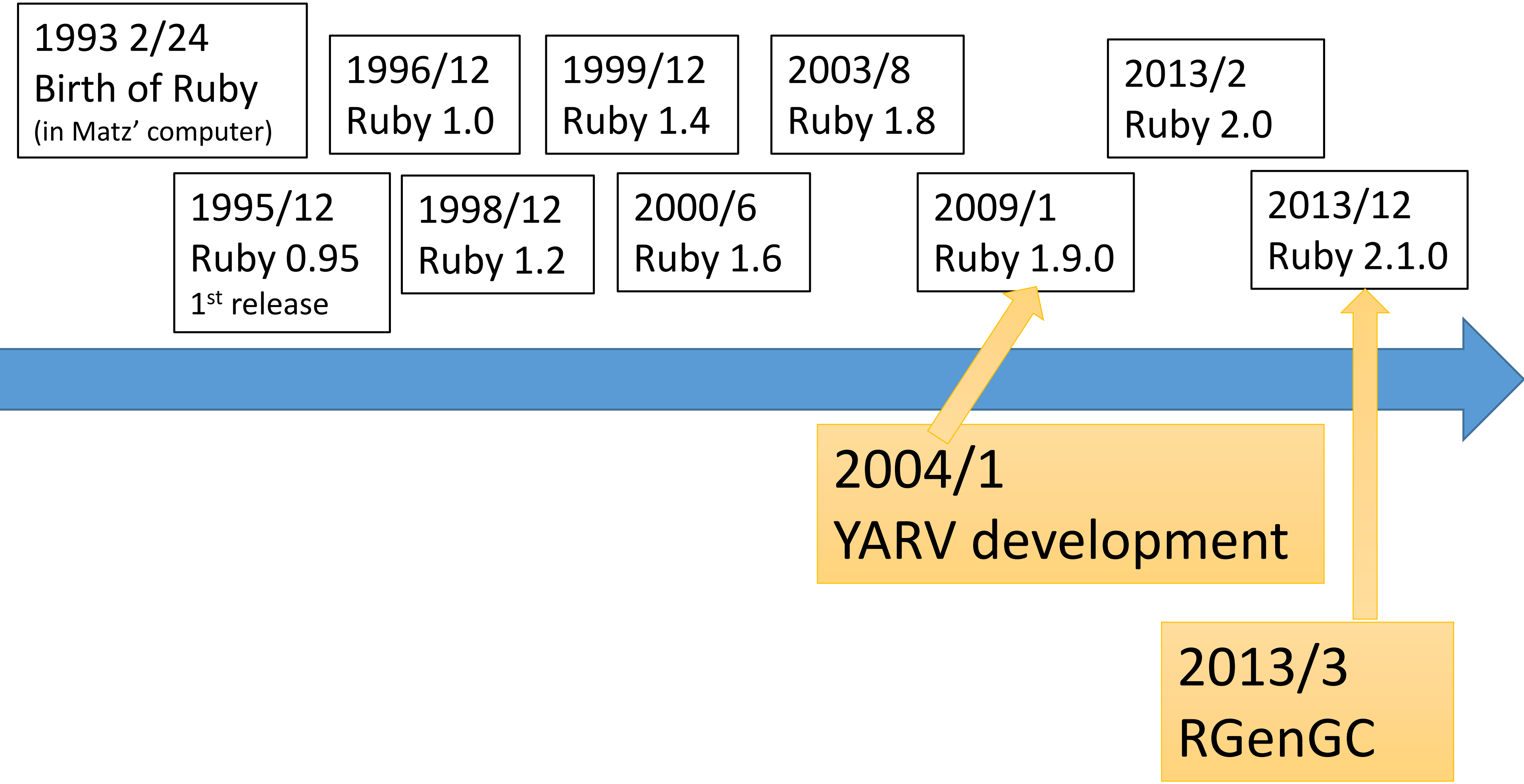
Ruby's components



Working for core components

- Core components I’m working for:
  - Evaluator (10 years)
  - Thread management (10 years)
  - Memory management (few years)

History of Ruby interpreter



1993 2/24
Birth of Ruby
(in Matz' computer)

1995/12
Ruby 0.95
1st release

1996/12
Ruby 1.0

1998/12
Ruby 1.2

1999/12
Ruby 1.4

2000/6
Ruby 1.6

2003/8
Ruby 1.8

2009/1
Ruby 1.9.0

2013/2
Ruby 2.0

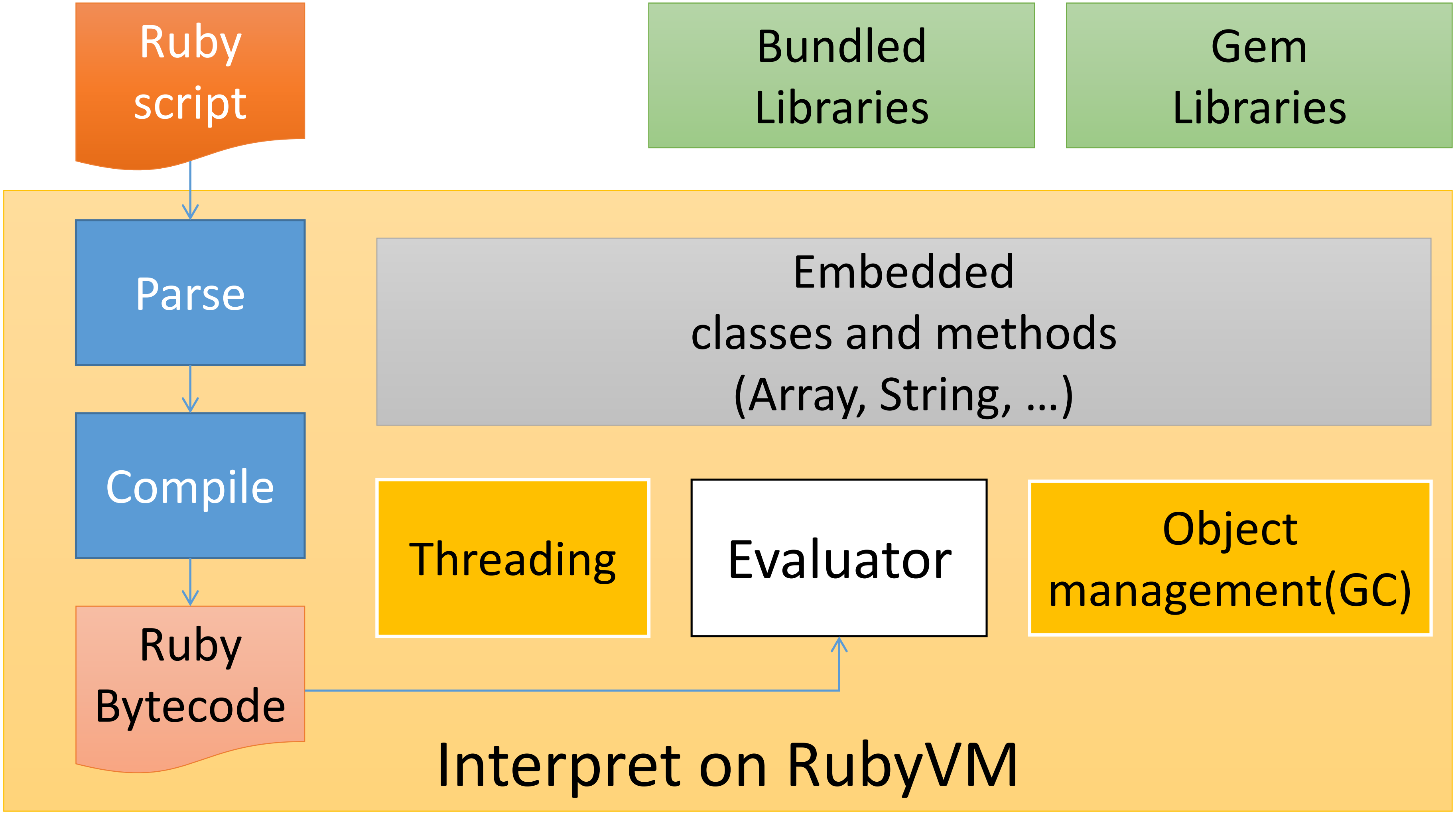
2013/12
Ruby 2.1.0

2004/1
YARV development

2013/3
RGenGC

Introduce our effort (especially my contributions) to speedup Ruby interpreter

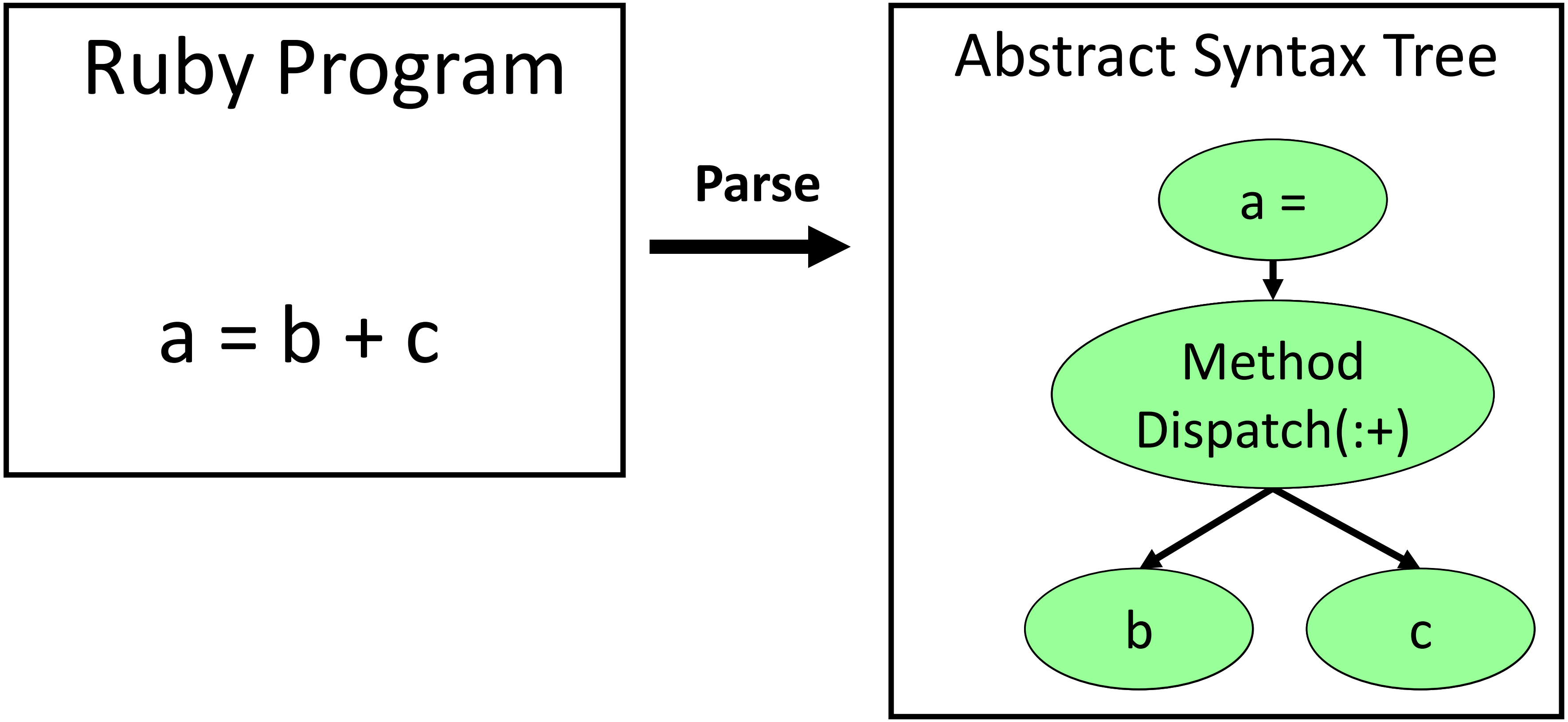
Evaluator


Ruby script
Bundled Libraries
Gem Libraries
Parse
Embedded classes and methods (Array, String, …)
Compile
Threading
Evaluator
Object management(GC)
Ruby Bytecode
Interpret on RubyVM

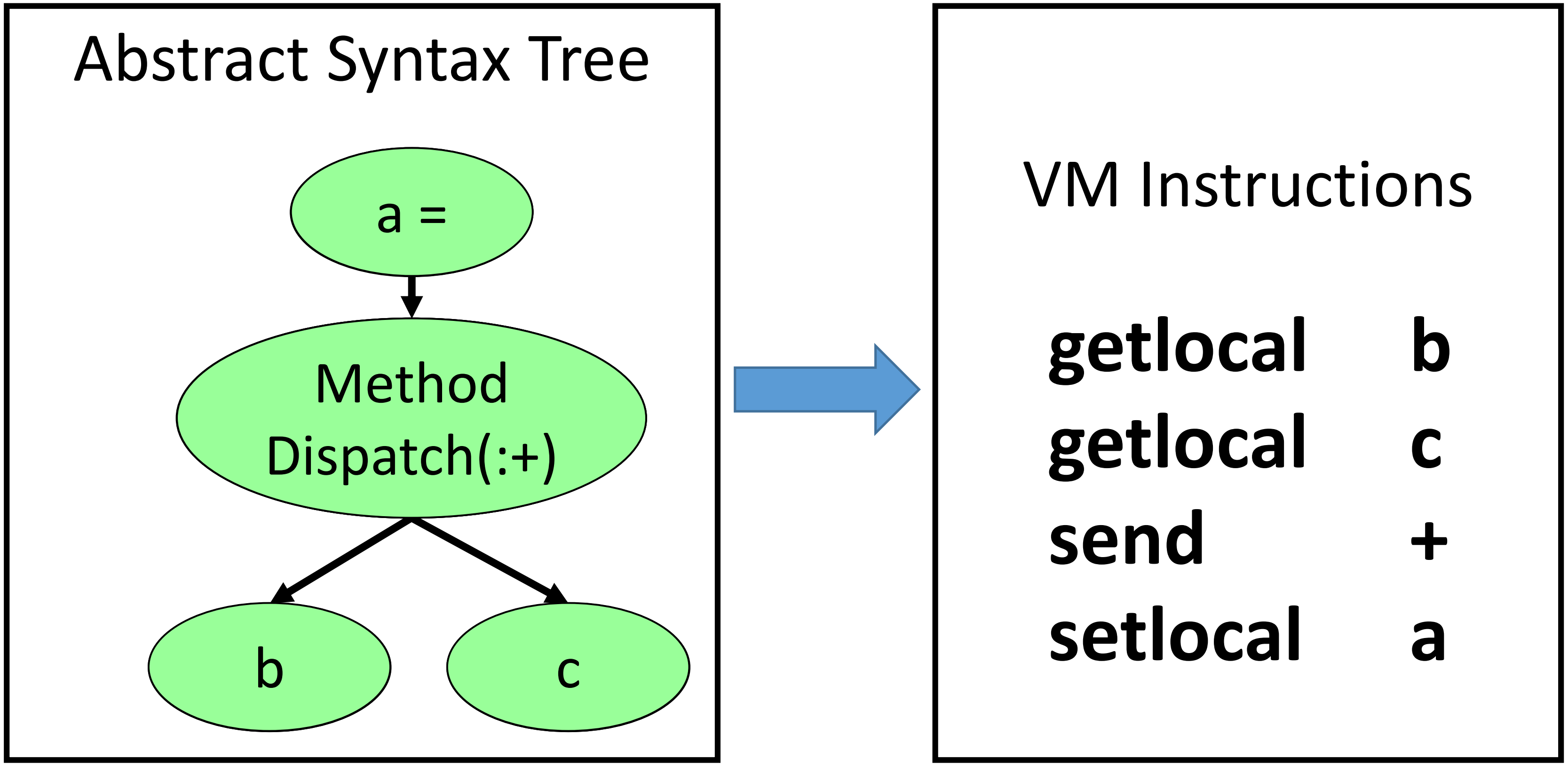
Evaluator

- Named YARV: Yet another RubyVM
  - Start until 10 years ago (2004/01/01)
  - Simple stack machine architecture
  - Execute each bytecode instructions one by one
- Apply many known optimization techniques

Evaluator
Compile Ruby to AST



Evaluator
Compile AST to Bytecode



Abstract Syntax Tree

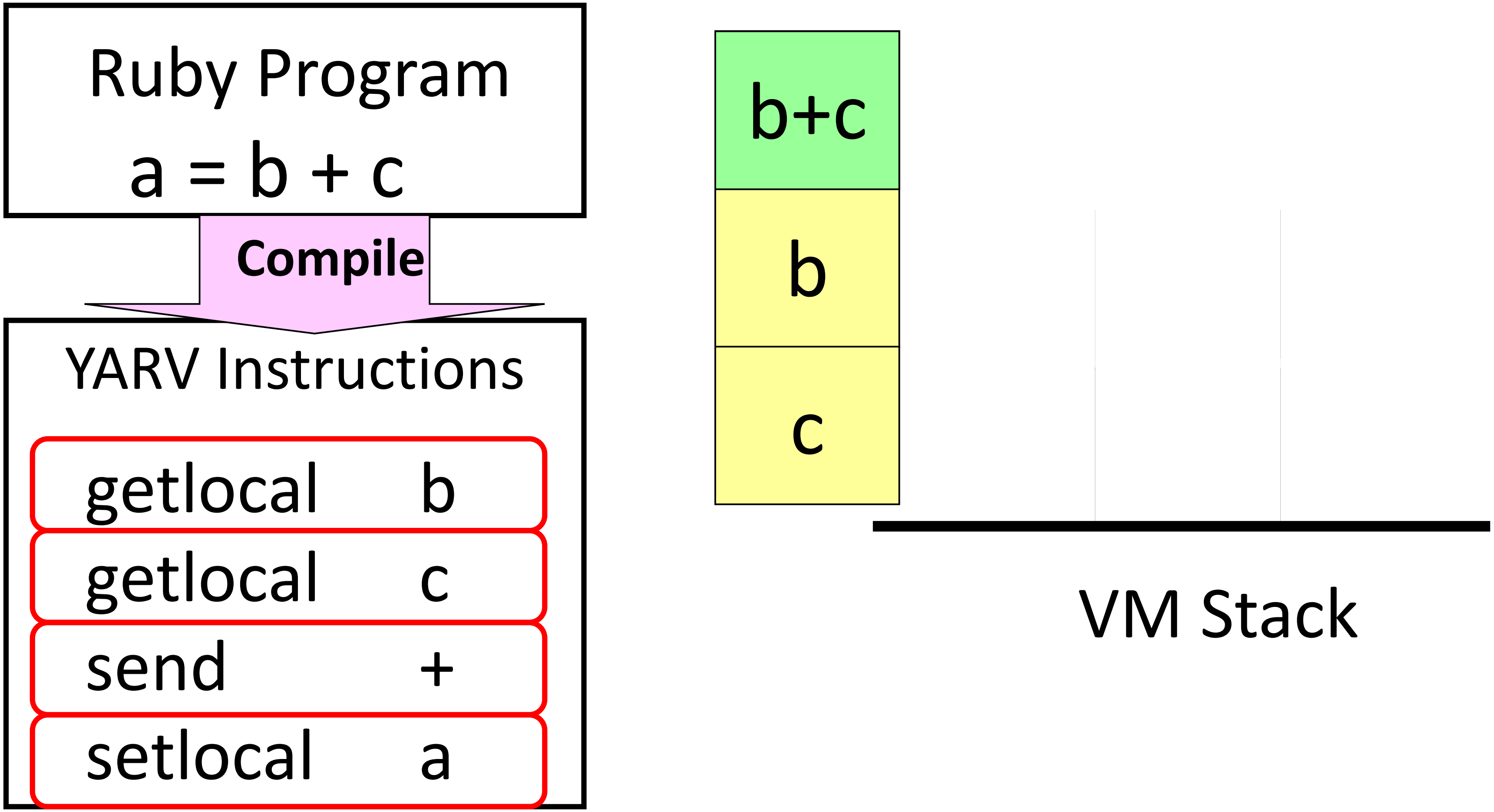
VM Instructions

```
getlocal    b
getlocal    c
send        +
setlocal    a
```

Tree data

Sequential data

Evaluator
Execution as stack machine



Ruby Program

a = b + c

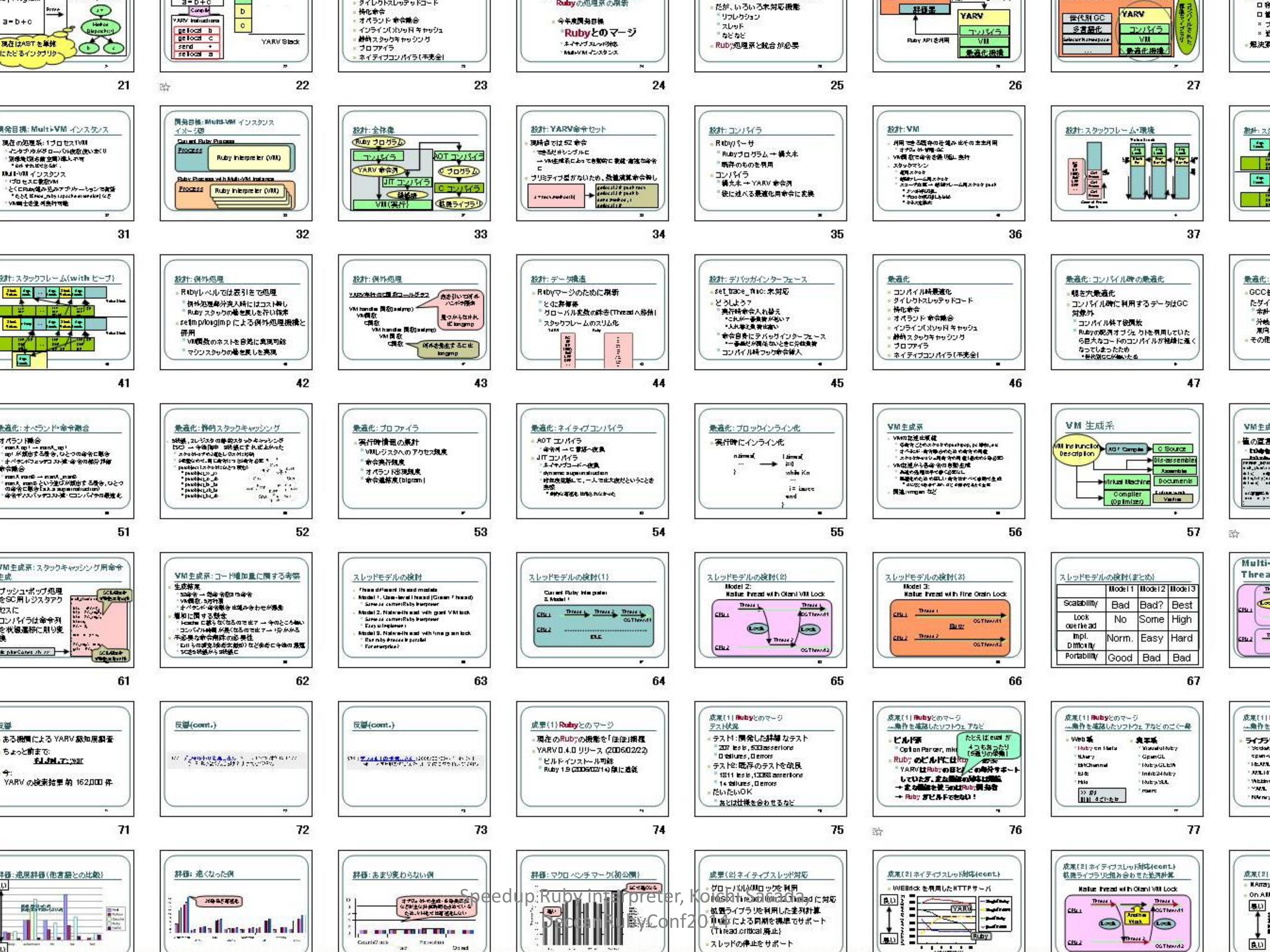
Compile

YARV Instructions

```
getlocal    b
getlocal    c
send        +
setlocal    a
```

Evaluator Optimizations

- Apply many techniques to improve performance
  - Peephole optimizations
  - Specialized instructions
  - Stack frame layout
  - Efficient exception handling
  - Efficient block representation
  - Direct threading
  - Stack caching
  - Instructions and operands unifications
  - …


21
22
23
24
25
26
27
31
32
33
34
35
36
37
41
42
43
44
45
46
47
51
52
53
54
55
56
57
61
62
63
64
65
66
67
71
72
73
74
75
76
77
開発目標: Multi-VM インスタンス
設計: 全体像
設計: YARV命令セット
設計: コンパイラ
設計: VM
設計: スタックフレーム・環境
設計: 例外処理
設計: データ構造
設計: デバッガインターフェース
最適化
最適化: コンパイル時の最適化
最適化: 静的スタックキャッシング
最適化: プロファイラ
最適化: ネイティブコンパイラ
最適化: ブロックインライン化
VM 生成系
VM生成系: コード増加量に関する考察
スレッドモデルの検討
スレッドモデルの検討(まとめ)
反響
反響(cont.)
成果(1) Rubyとのマージ
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
成果(2) ネイティブスレッド対応(cont.)

Speedup Ruby interpreter, Koichi Sasada,
DeccanRubyConf2014

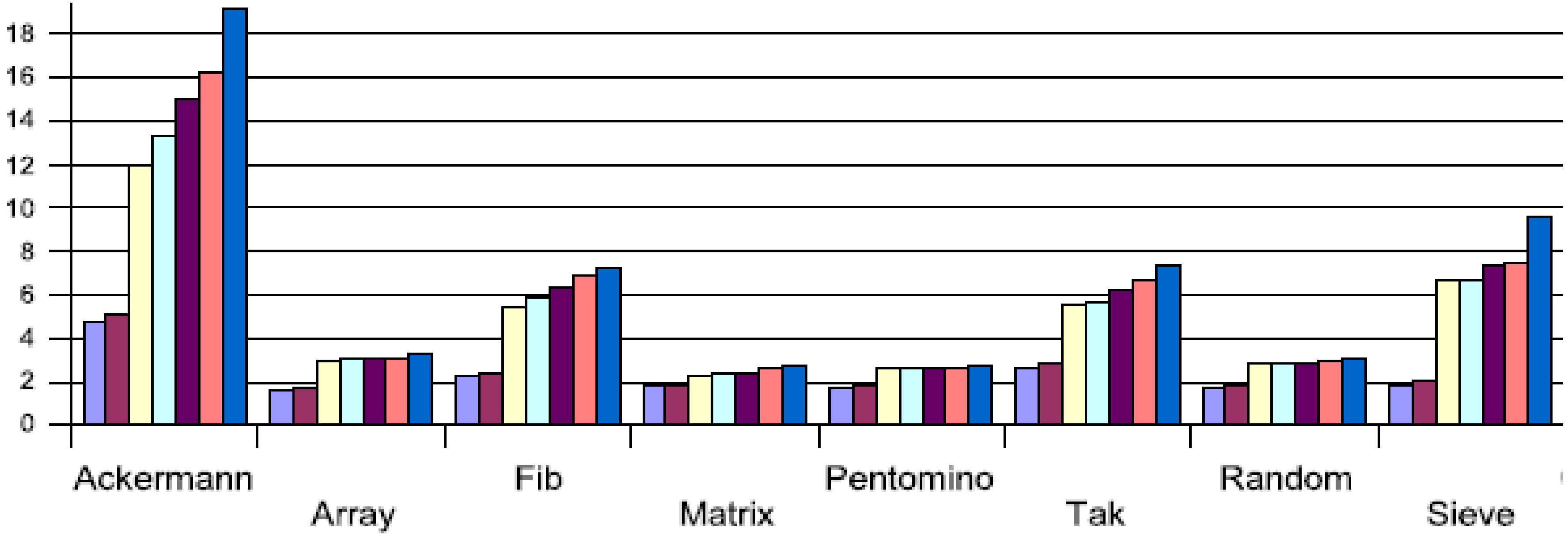
Evaluator Optimizations

- Analysis usage

  And optimize for frequent cases
- Example: Exception handling
  - Exceptions occur *EXCEPTIONAL* so optimize for no-exception control flow

Performance evaluation compare with Ruby 1.8

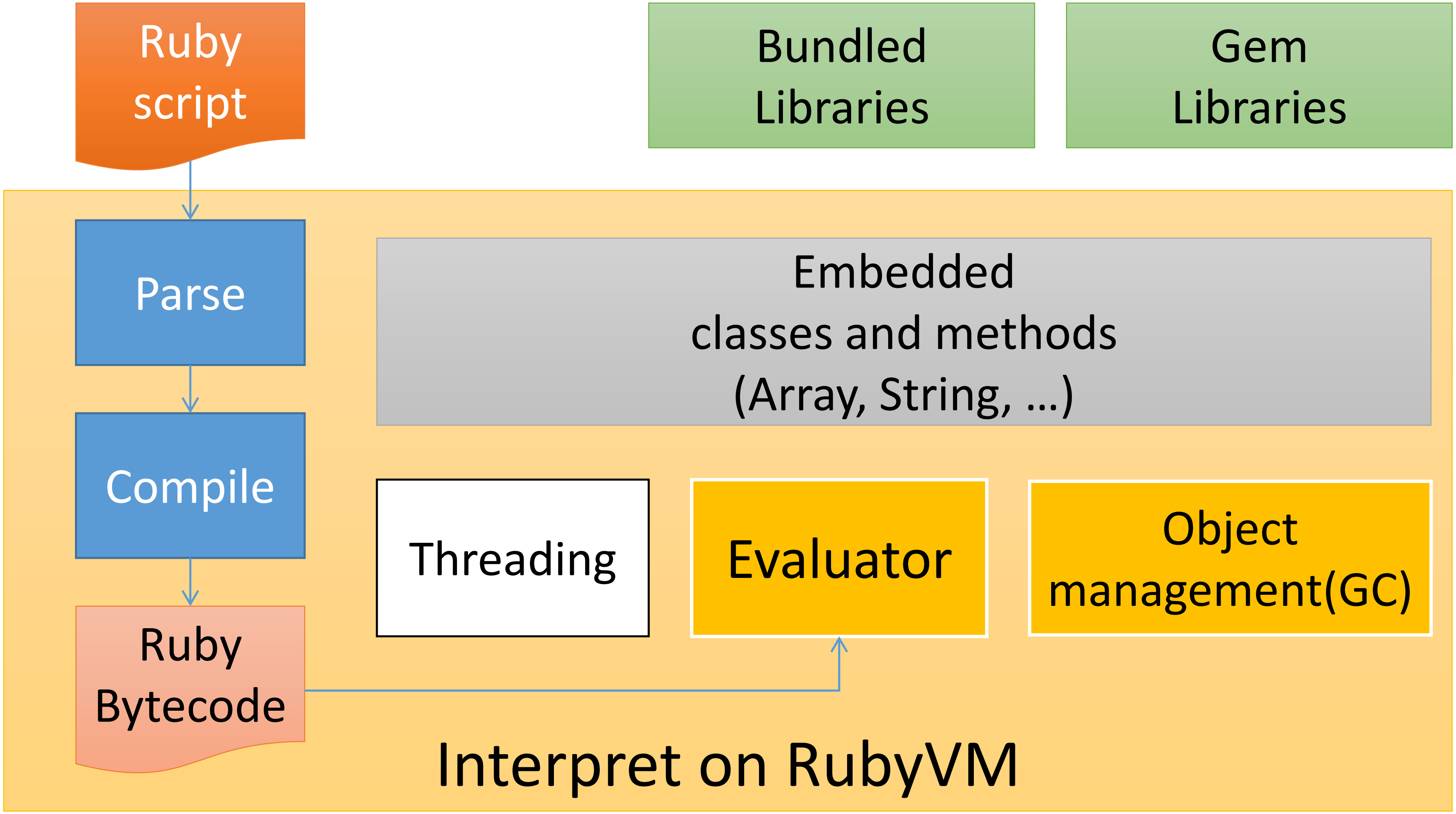
Higher is good



Main components

- Evaluator
- <u>Thread management</u>
- Memory management

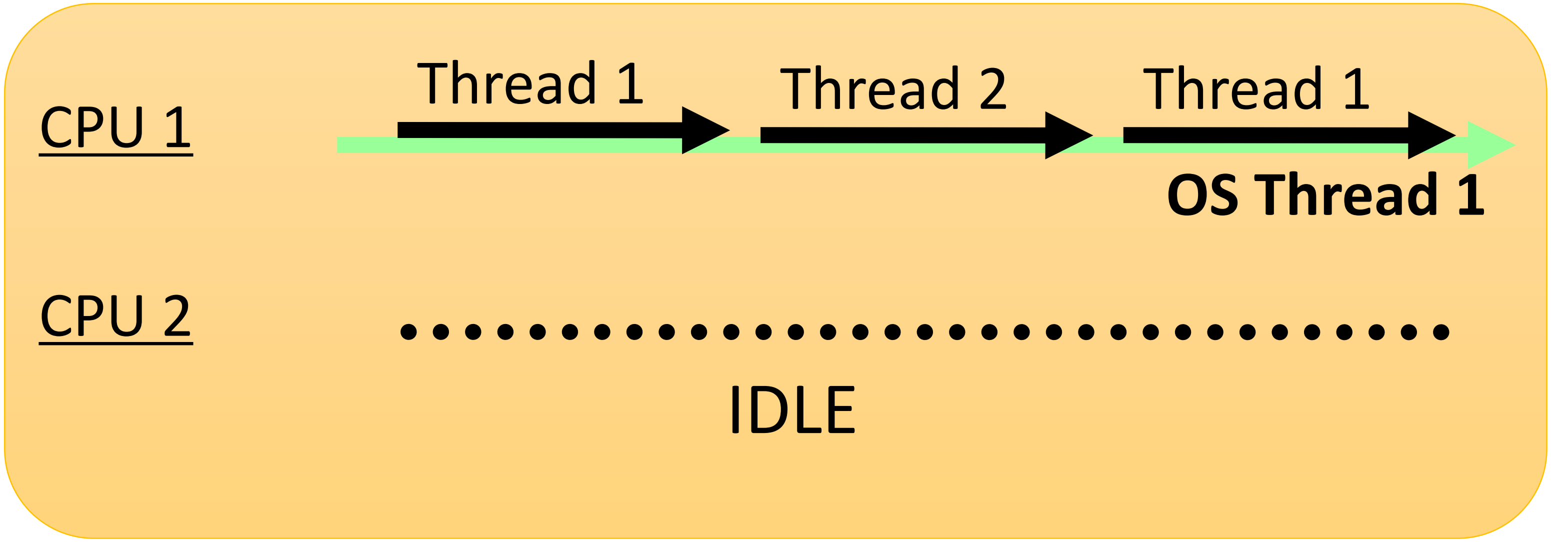
Threading


Ruby script
Bundled Libraries
Gem Libraries
Parse
Embedded classes and methods (Array, String, …)
Compile
Threading
Evaluator
Object management(GC)
Ruby Bytecode
Interpret on RubyVM

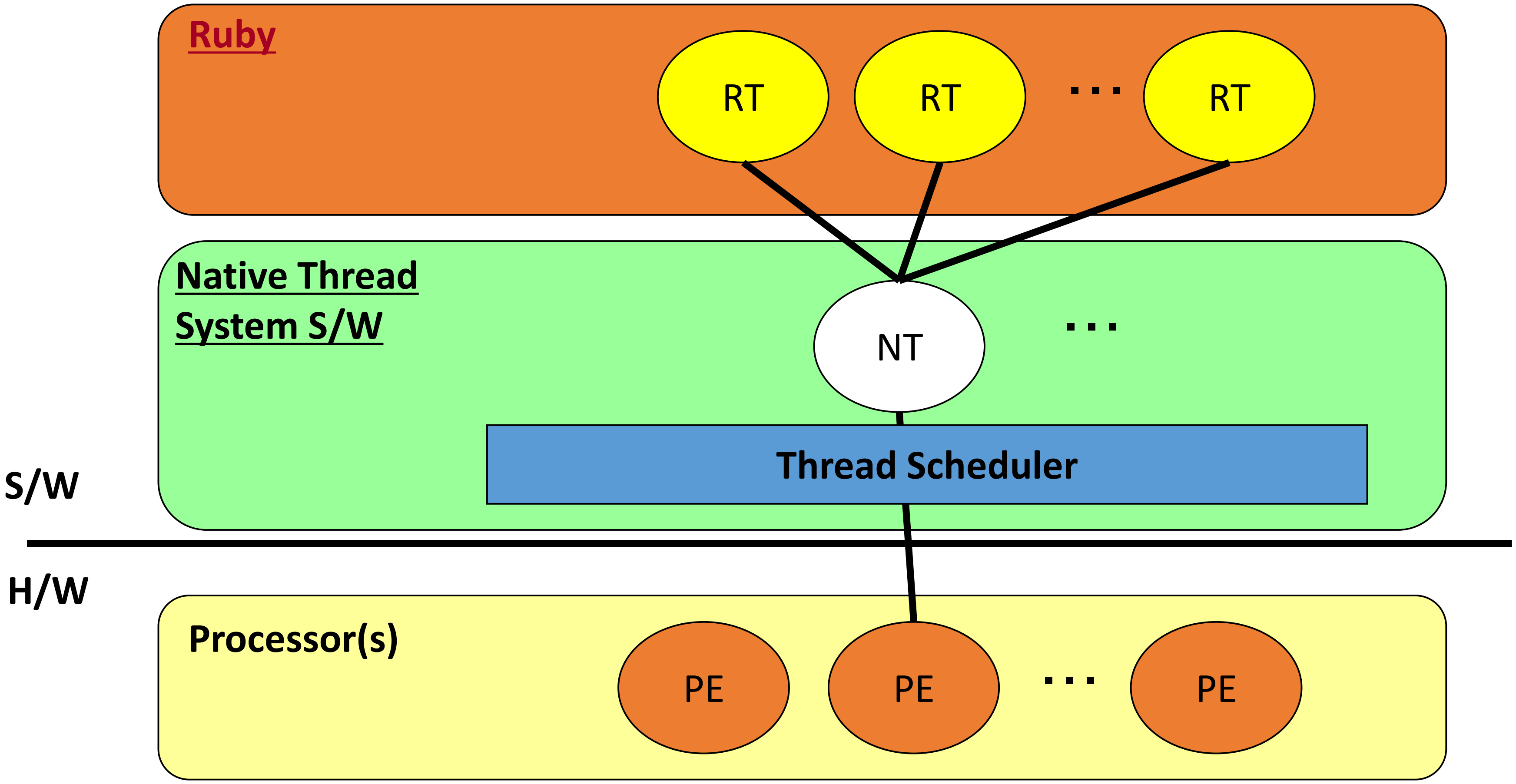
Threading

- Using native threads for each Ruby threads
- Parallel ruby execution is prohibited by GVL
    - You can free GVL if you write a code carefully in C level and run it in parallel

Threading
Ruby 1.8 and before

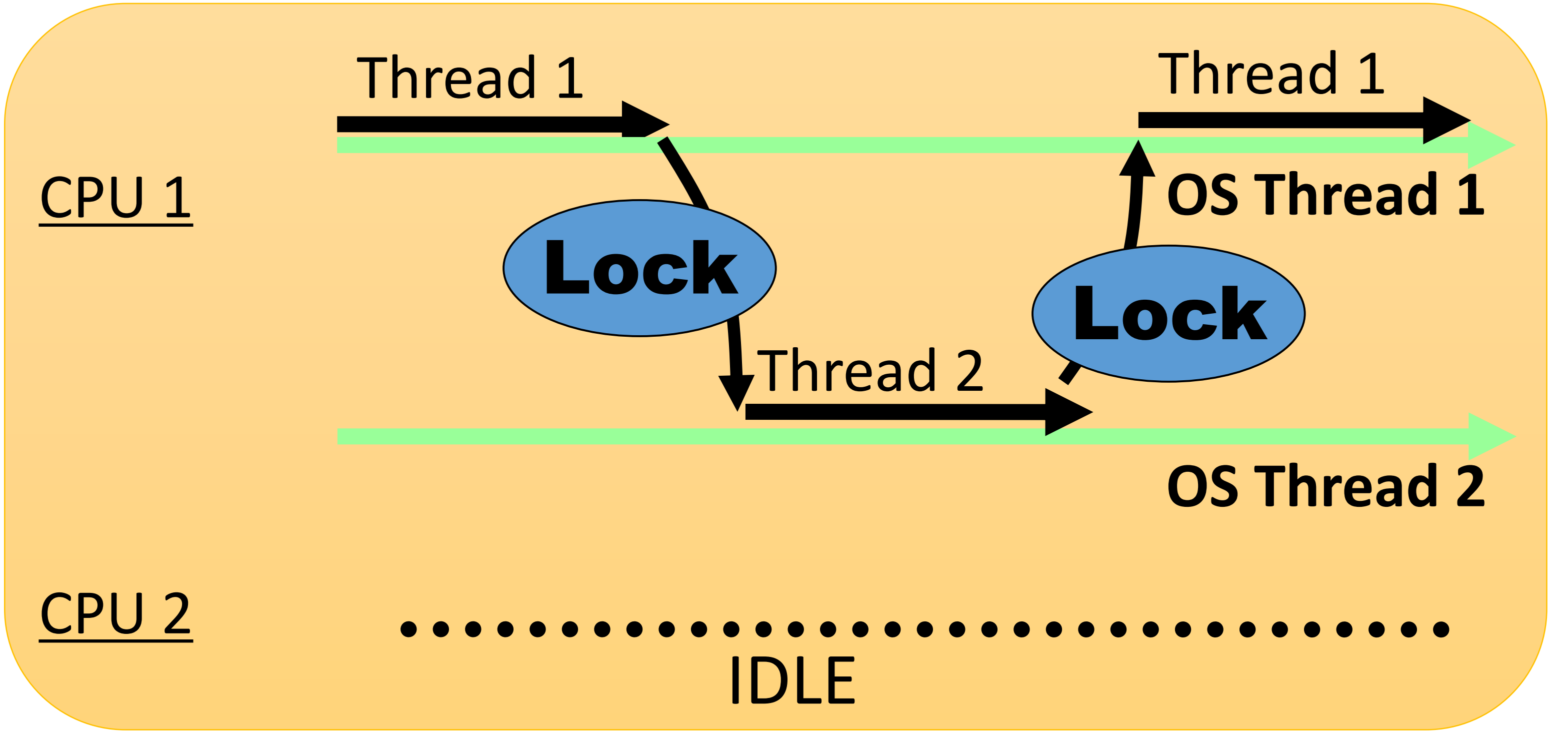


Threading
Layered view

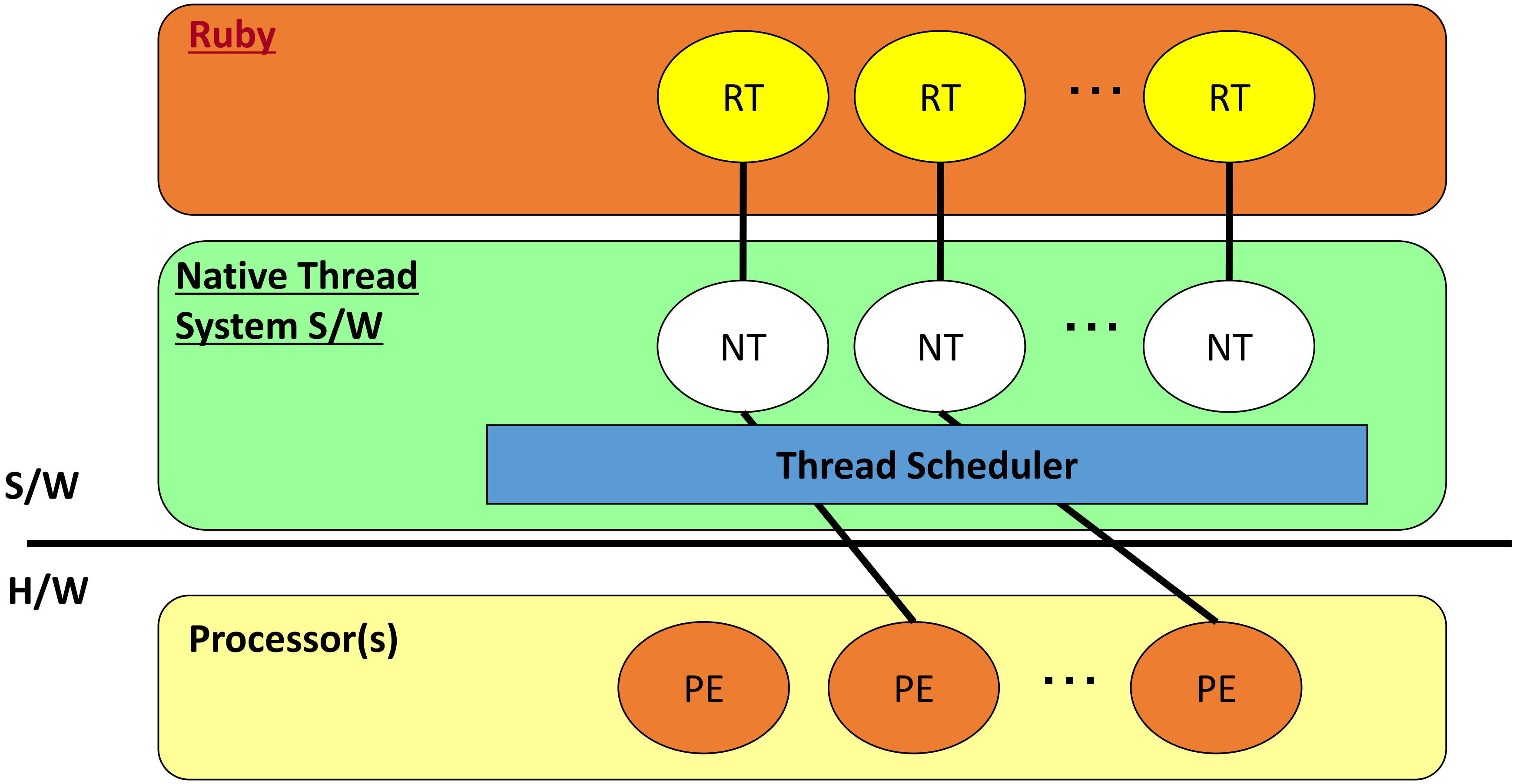


Threading
Ruby 1.9 and later

Native threads with Giant VM Lock



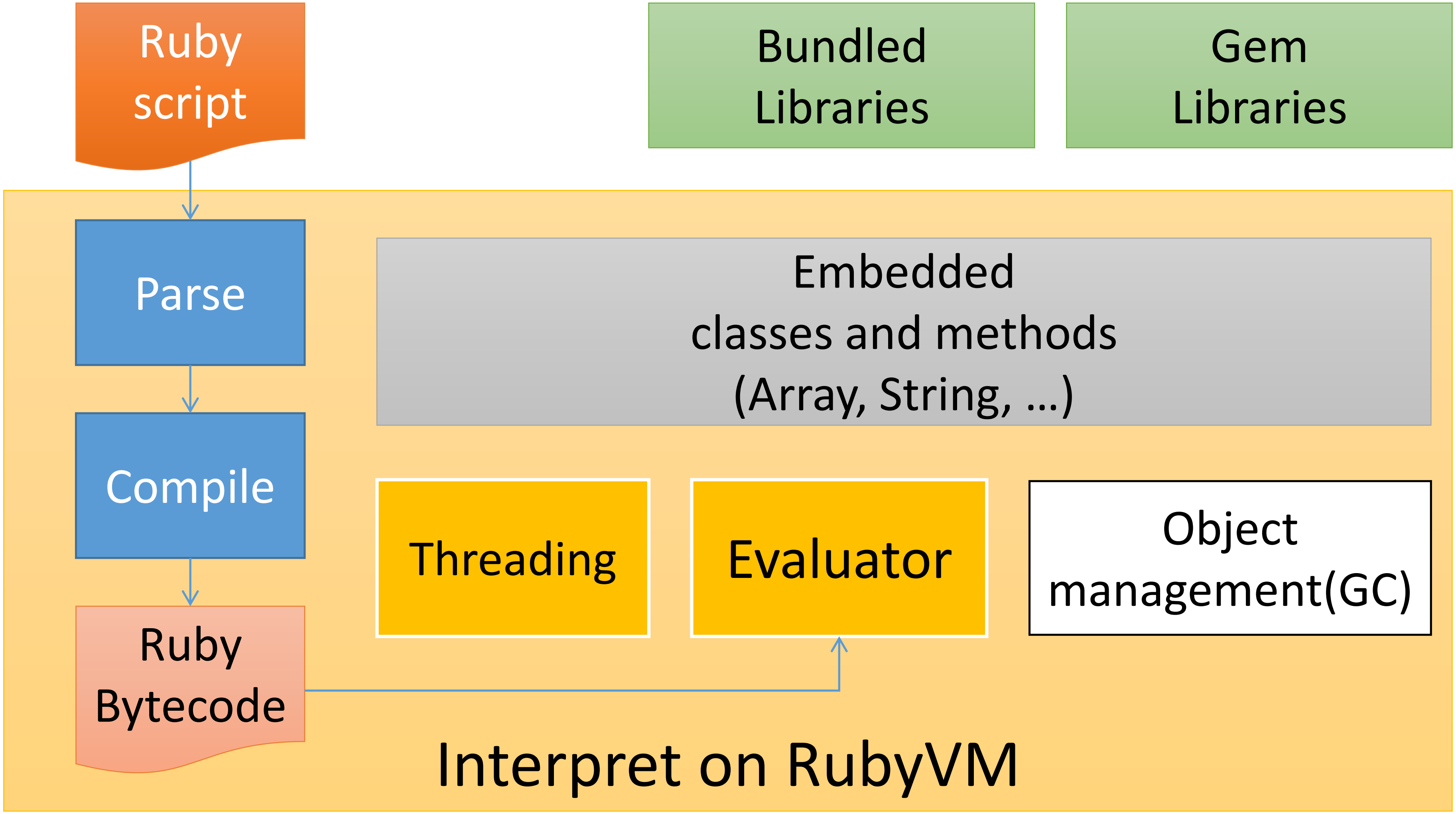
Threading
Layered view



Threading
Why GVL?

- To protect Ruby users from nightmare debugging
    - Shared parallel threading can make non deterministic bugs which is too hard to debug
- Disadvantage
    - CRITICAL ISSUE: No parallel programming in Ruby
    - Need another programming model for parallel
        - Current ***SHARED EVERYTHING*** model is not match
        - Correct isolation level for each parallel execution units

Object management (GC)



Object and memory management

- "Object.new" allocate a new object
    - "foo" (string literal) also allocate a new object
    - Everything are objects in Ruby!
- We don't need to **"de-allocate"** objects manually

Garbage collection

The automatic memory management

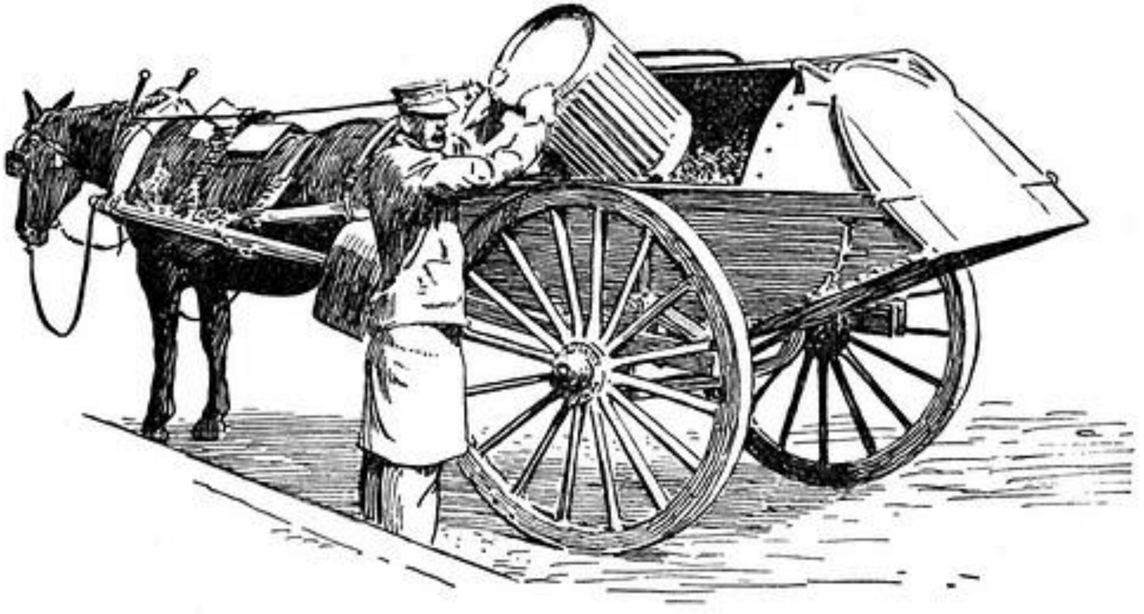


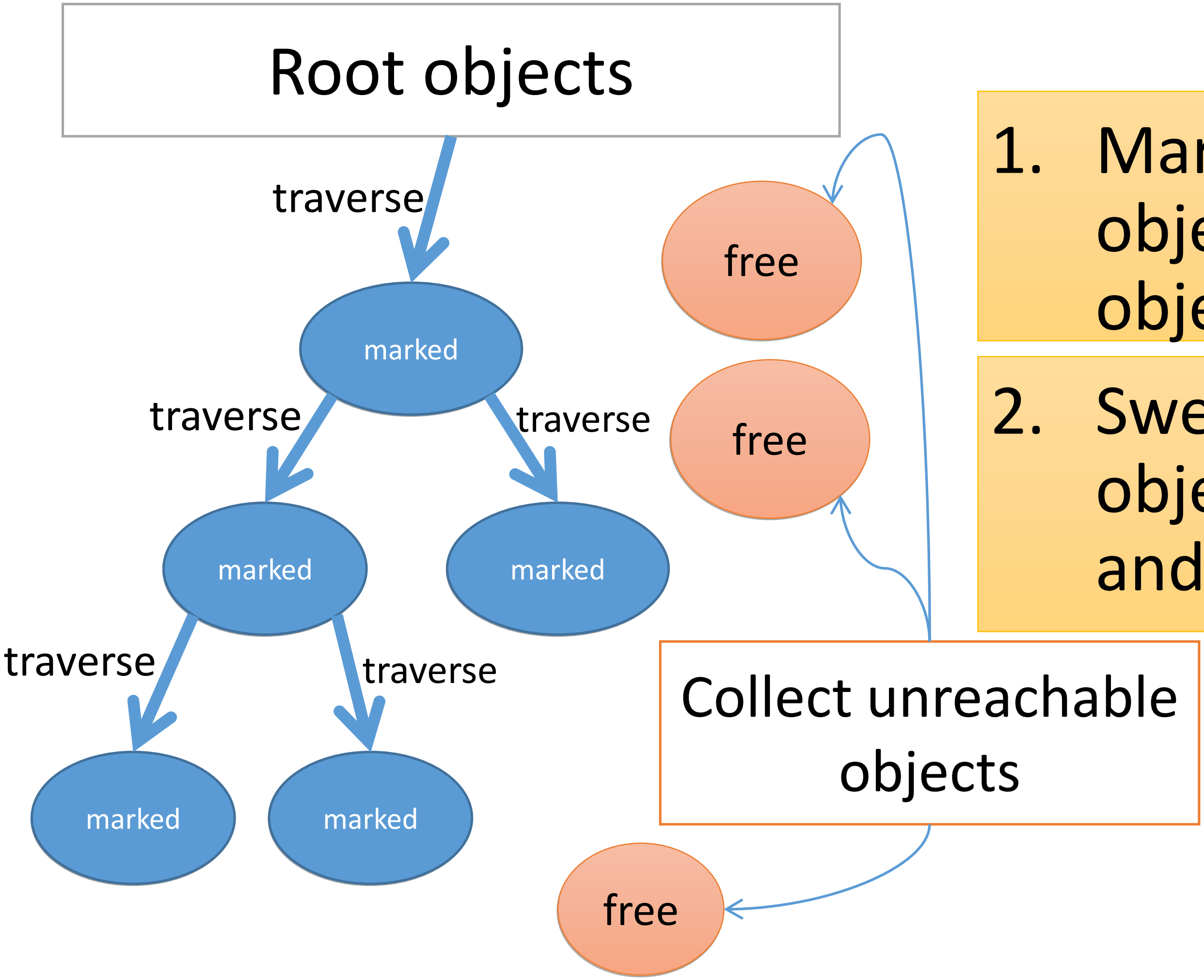
FIG. 109. — A GARBAGE COLLECTOR.

http://www.flickr.com/photos/circasassy/6817999189/

Automatic memory management Basic concept

- **Garbage collector recycled “unused” objects automatically**

Mark & Sweep algorithm

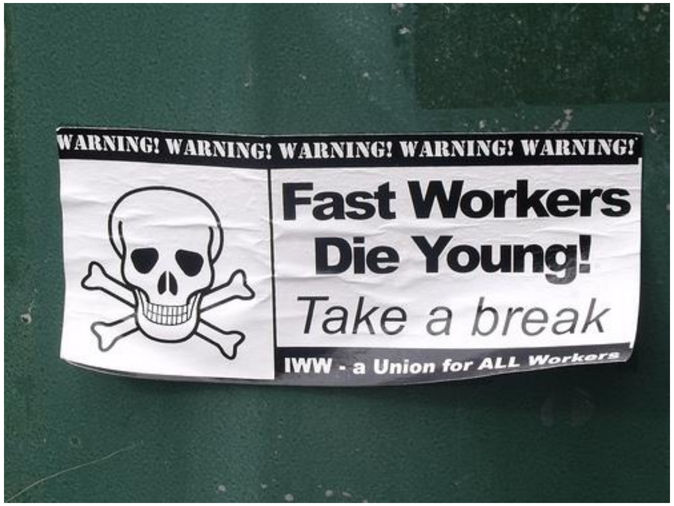


1. Mark reachable objects from root objects
2. Sweep ***unmarked*** objects (collection and de-allocation)

Generational GC (GenGC)

- Weak generational hypothesis:

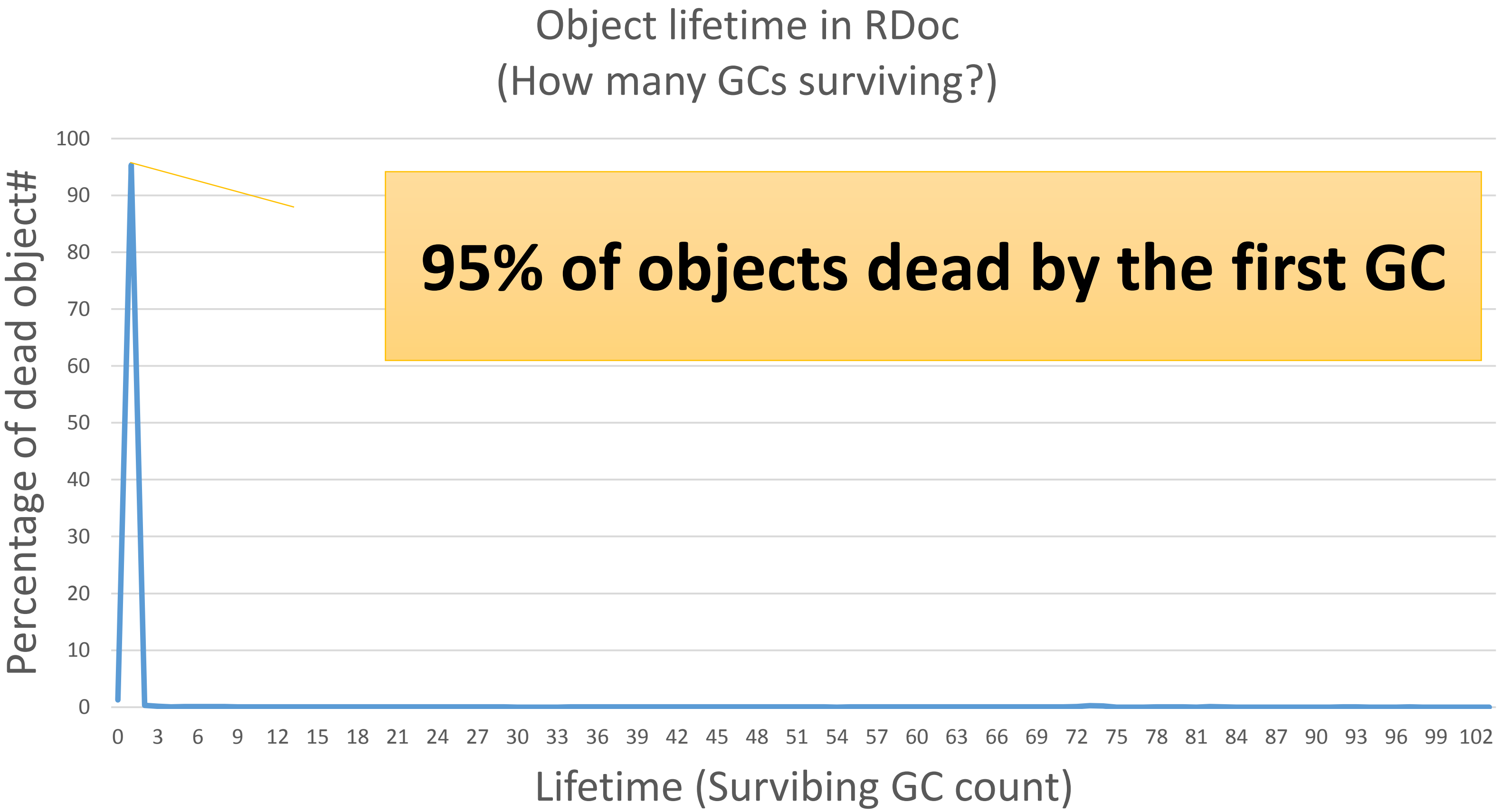
“Most objects die young”


WARNING! WARNING! WARNING! WARNING! WARNING!
Fast Workers
Die Young!
Take a break
IWW - a Union for ALL Workers

http://www.flickr.com/photos/ell-r-brown/5026593710

→ Concentrate reclamation effort only on the young objects

Generational hypothesis

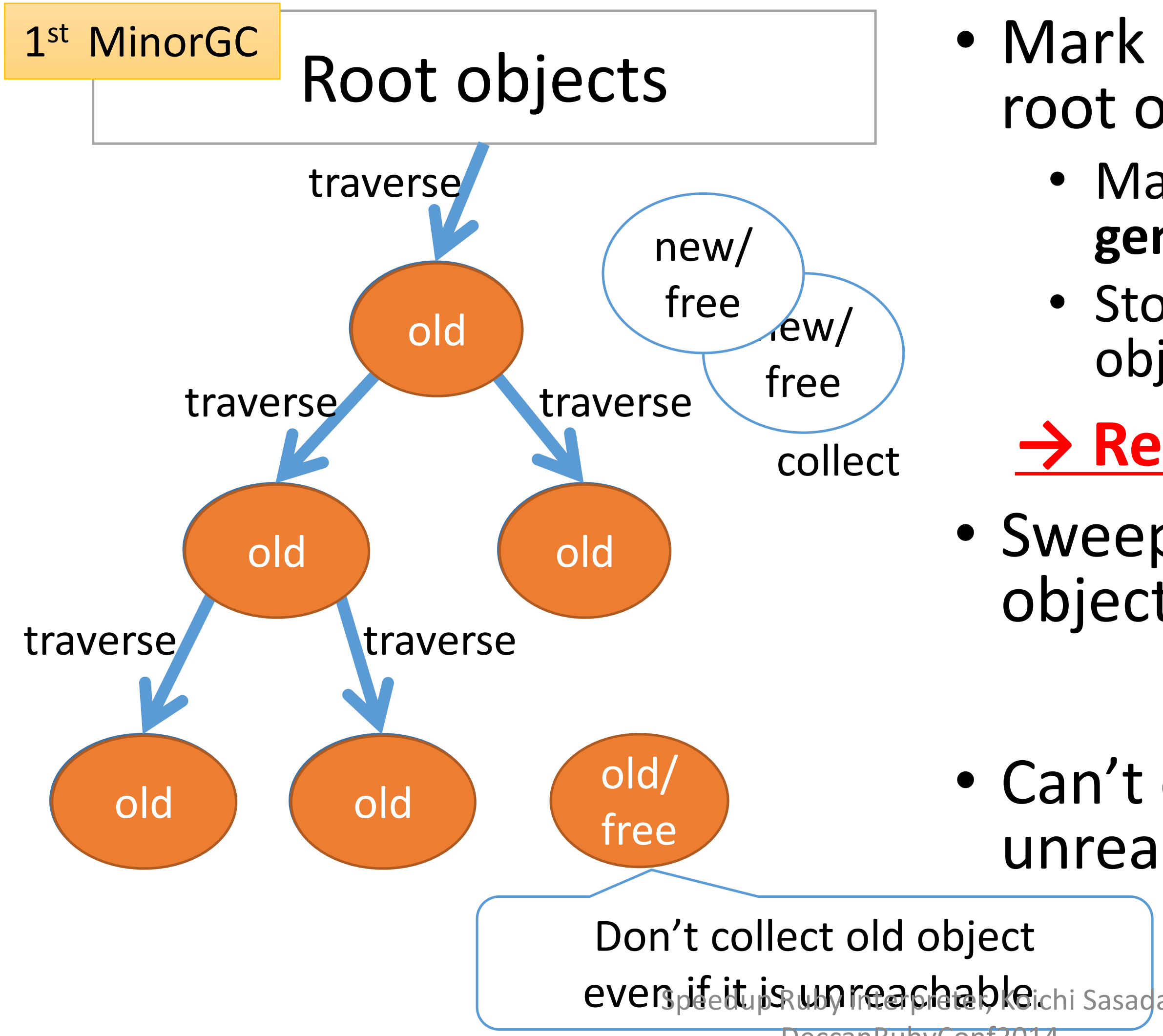


95% of objects dead by the first GC

Generational GC (GenGC)

- Separate young generation and old generation
    - Create objects as young generation
    - Promote to old generation after surviving *n-th* GC
    - In CRuby, *n == 1* (after 1 GC, objects become old)
        - n == 2 from Ruby 2.2 (plan)
- Usually, GC on young space (minor GC)
- GC on both spaces if no memory (major/full GC)

GenGC [Minor M&S GC] (1/2)



Don’t collect old object even if it is unreachable.

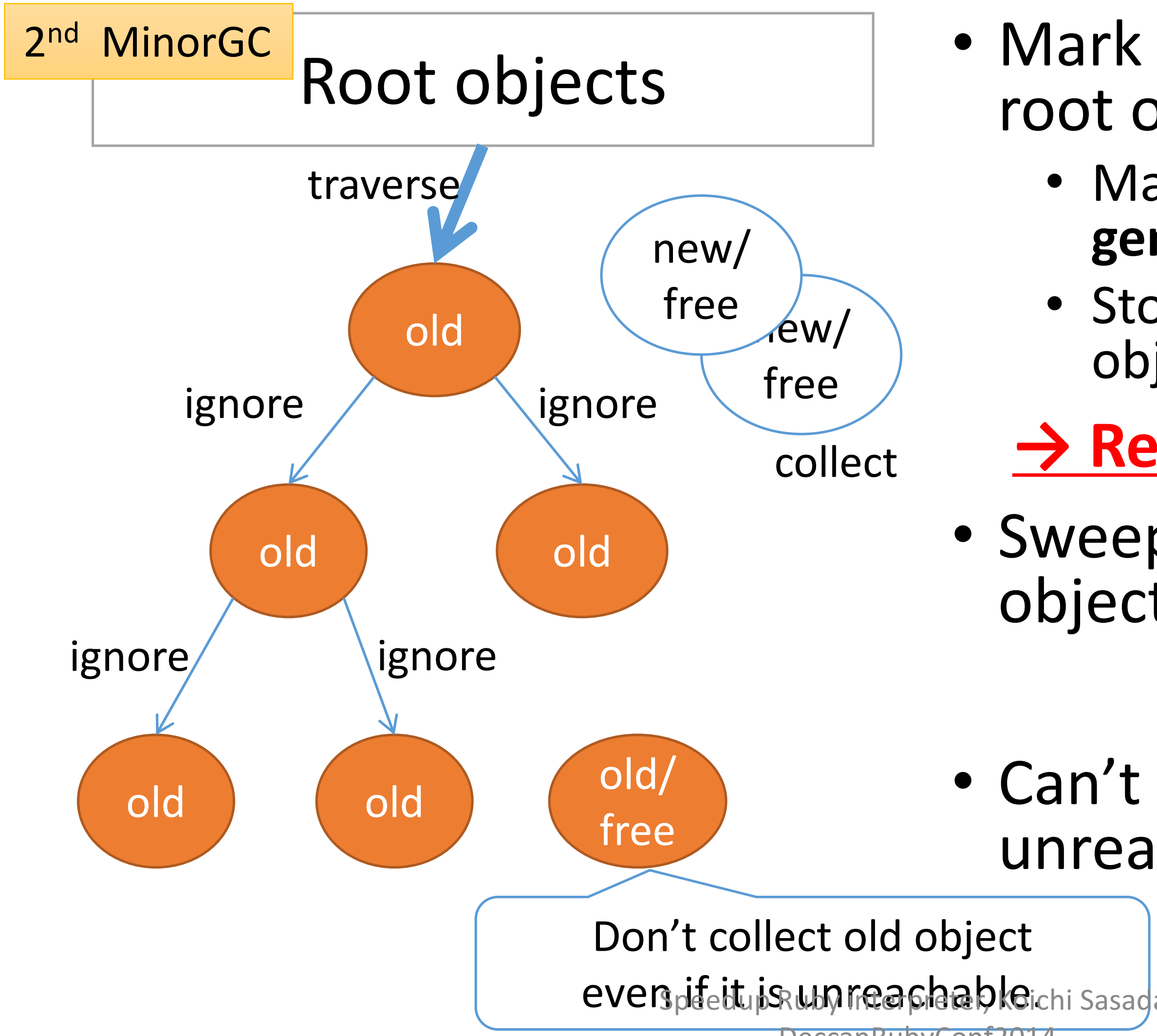
- Mark reachable objects from root objects.
  - Mark and **promote to old generation**
  - Stop traversing after old objects

→ Reduce mark overhead

- Sweep not (marked or old) objects

- Can’t collect Some unreachable objects

GenGC [Minor M&S GC] (2/2)



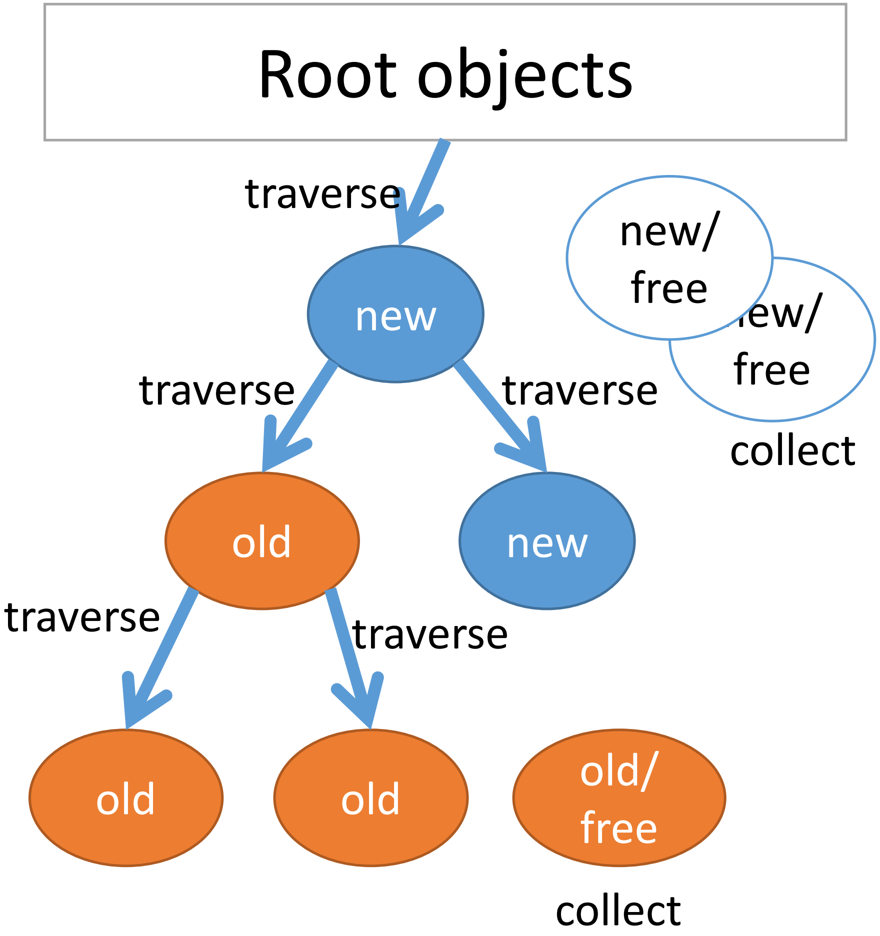
Don't collect old object even if it is unreachable.

- Mark reachable objects from root objects.
  - Mark and **promote to old generation**
  - Stop traversing after old objects

→ Reduce mark overhead

- Sweep not (marked or old) objects
- Can't collect Some unreachable objects

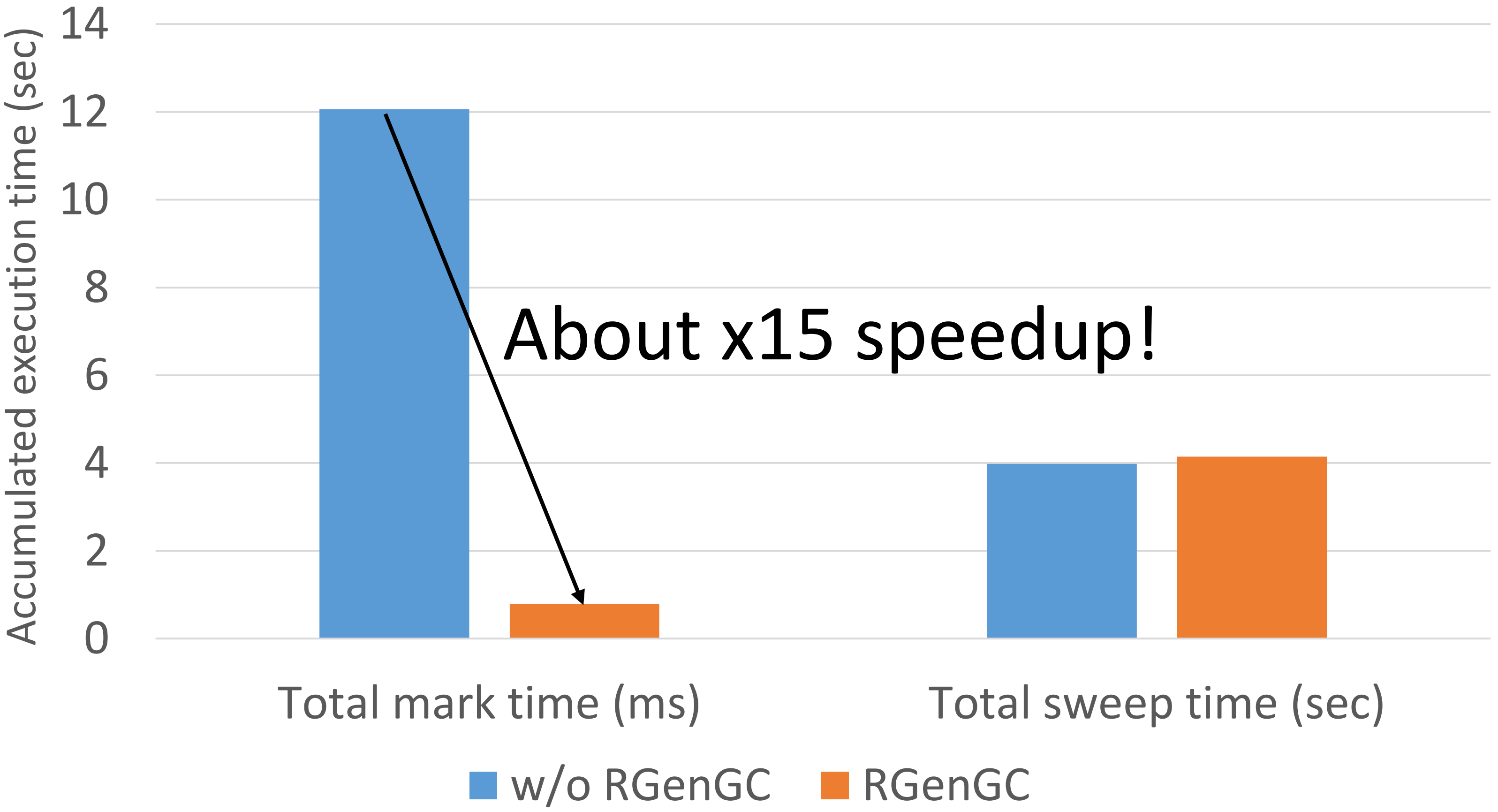
GenGC [Major M&S GC]



- Normal M&S
- Mark reachable objects from root objects
  - Mark and **promote to old gen**
- Sweep unmarked objects

- *Sweep all unreachable (unused) objects*

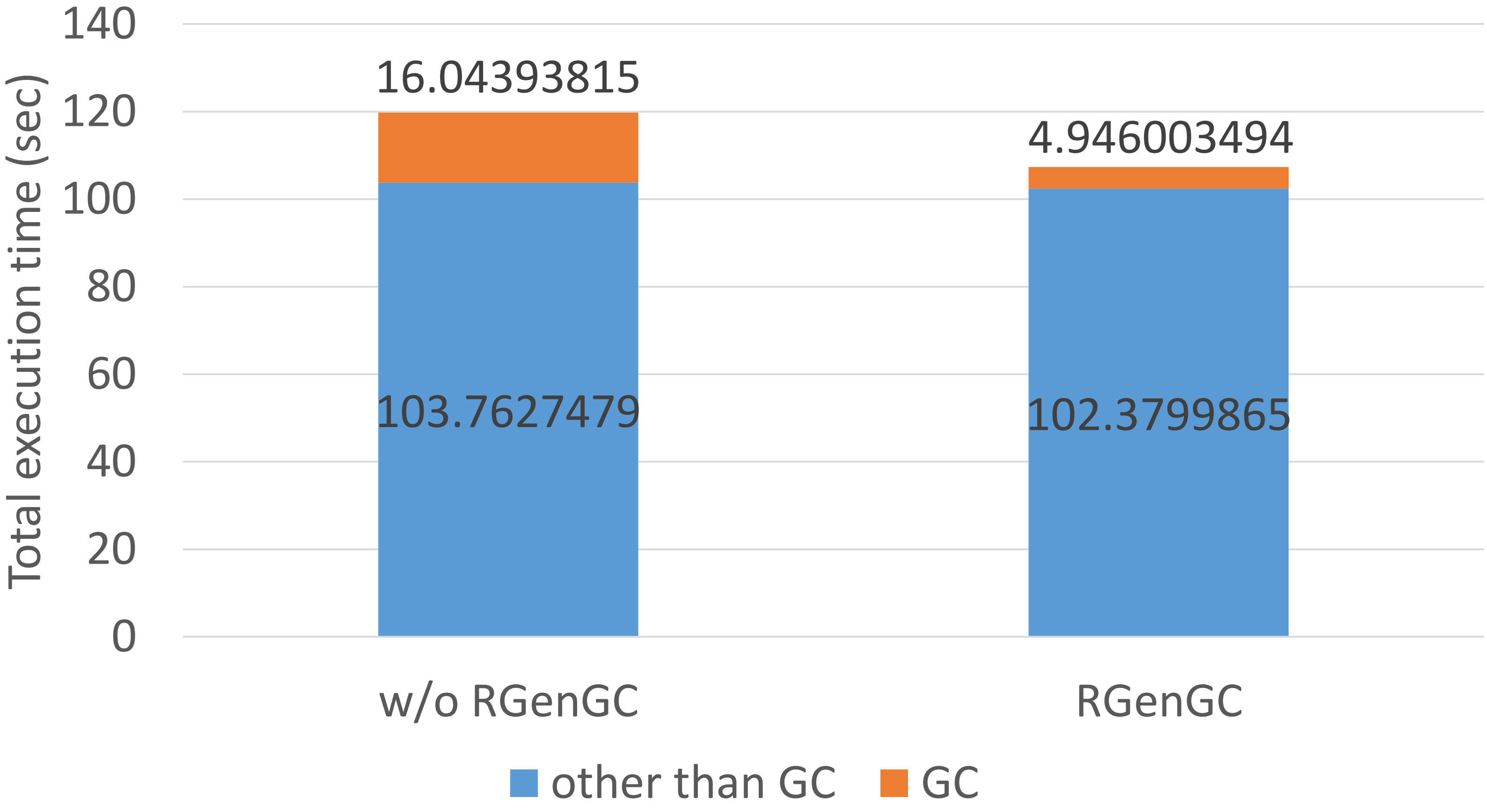
RGenGC
Performance evaluation (RDoc)



About x15 speedup!

* Disabled lazy sweep to measure correctly.

RGenGC
Performance evaluation (RDoc)



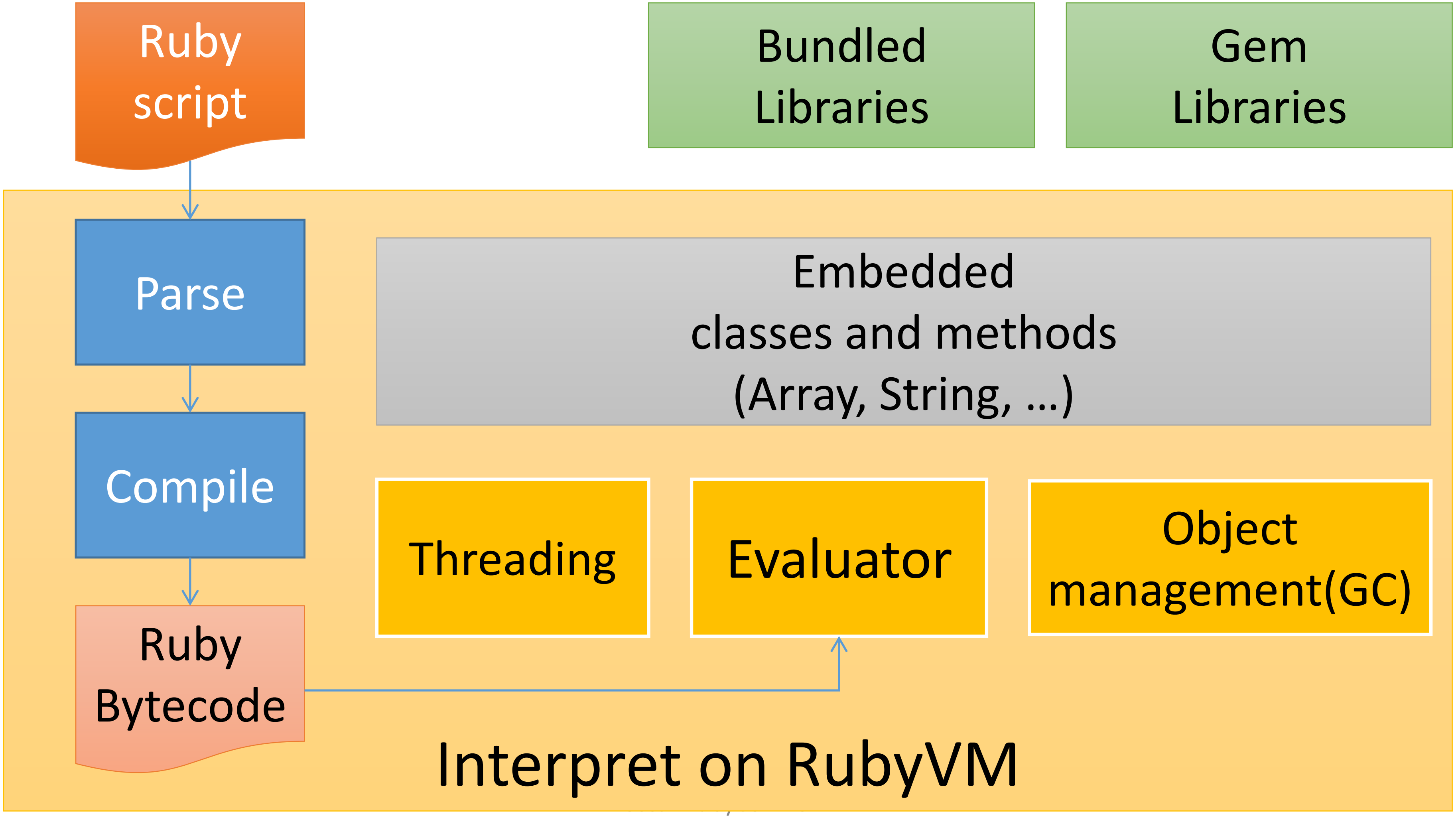
* 12% improvements compare with w/ and w/o RGenGC

* Disabled lazy sweep to measure correctly.

Speedup Ruby Interpreter

How to speed up Ruby interpreter?

DO EVERYTHING!
NO SILVER BULLET!



Ruby script

Bundled Libraries

Gem Libraries

Parse

Embedded classes and methods (Array, String, …)

Compile

Threading

Evaluator

Object management(GC)

Ruby Bytecode

Interpret on RubyVM

DO EVERYTHING!
NO SILVER BULLET!

We did.

We are doing.

We will do!!

Only continuous effort improves software quality.

Future work: Many many many!!

- Evaluator
  - JIT compilation
  - More drastic optimizations
- Threading
  - Parallel execution model (not a thread?)
- Object management and GC
  - Incremental GC
  - Compaction GC
  - Lightweight object allocation

Summary

- Ruby 2.1 and Ruby 2.2
- How to speed up Ruby interpreter?
  - Evaluator
  - Threading
  - Object management / Garbage collection

One answers is:

#=> Continue software development

Thank you for your attention
Q&A?

With slowly/clearly English, thank you.

Koichi Sasada

<ko1@heroku.com>

heroku