Growing the Ruby Interpreter

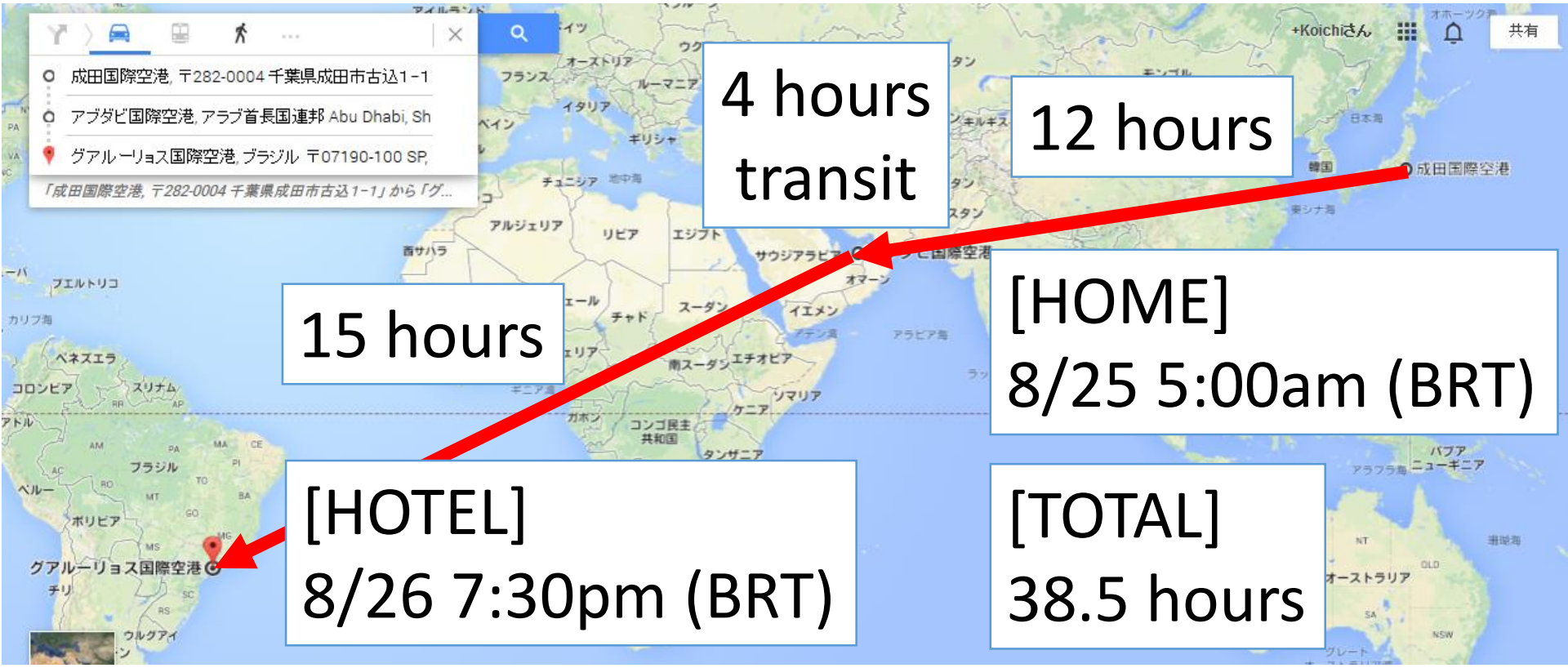
Koichi Sasada

<ko1@heroku.com>

Today's talk

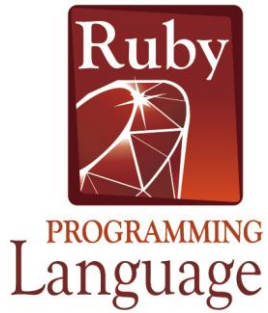
- Ruby 2.1 and Ruby 2.2
- How to grow up the Ruby interpreter?
  - Evaluator
  - Threading
  - Object management / Garbage collection

Koichi Sasada as a Japanese



Koichi Sasada as a Programmer

- CRuby/MRI committer
  - Core components developer
  - Virtual machine (YARV) since 2004/1/1
  - Rewrote Threads, GC, and so on
- Matz team at Heroku, Inc.
- Director of Ruby Association



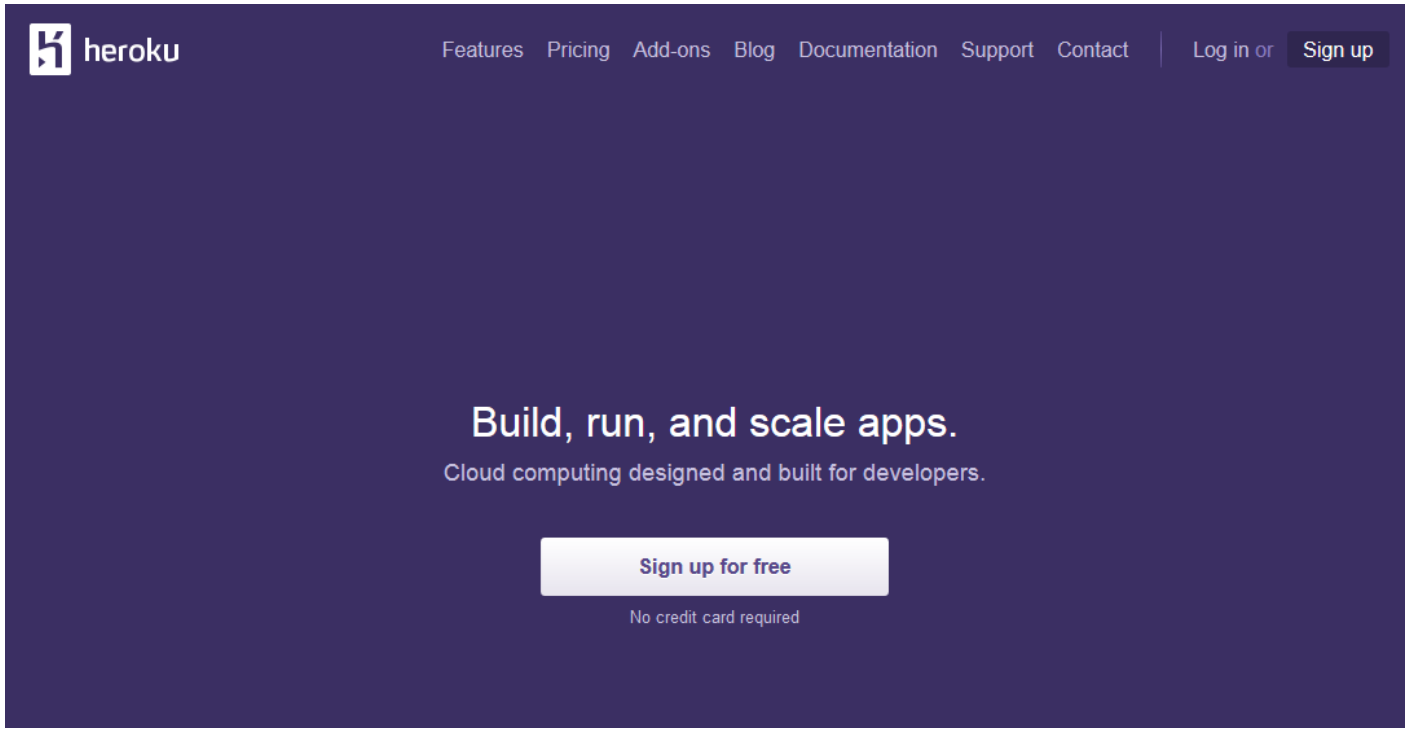
Advertisement

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. Submit your proposal now!
    - 3 projects. About 5,000 USD per project (deadline: 3rd ,Oct)
    - http://www.ruby.or.jp/en/news/20140805.html
  - Ruby Prize
  - Maintenance of Ruby (Cruby) interpreter
  - Events, especially RubyWorld Conference
  - **Donation** for Ruby developments and communities

Advertisement

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



Advertisement

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

Ask Nando Vieira for more details

- Heroku supports OSSs
  - 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”

heroku

Matz

Famous 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

heroku

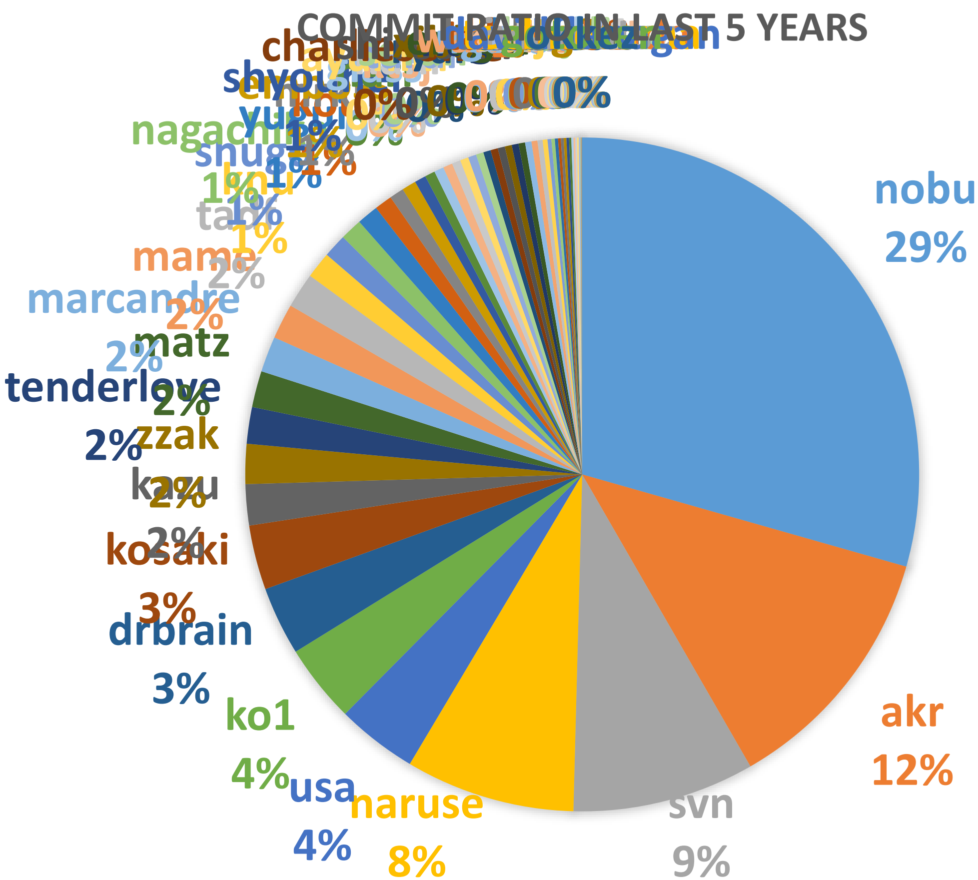
Nobu

Patch monster

- Great patch creator

heroku

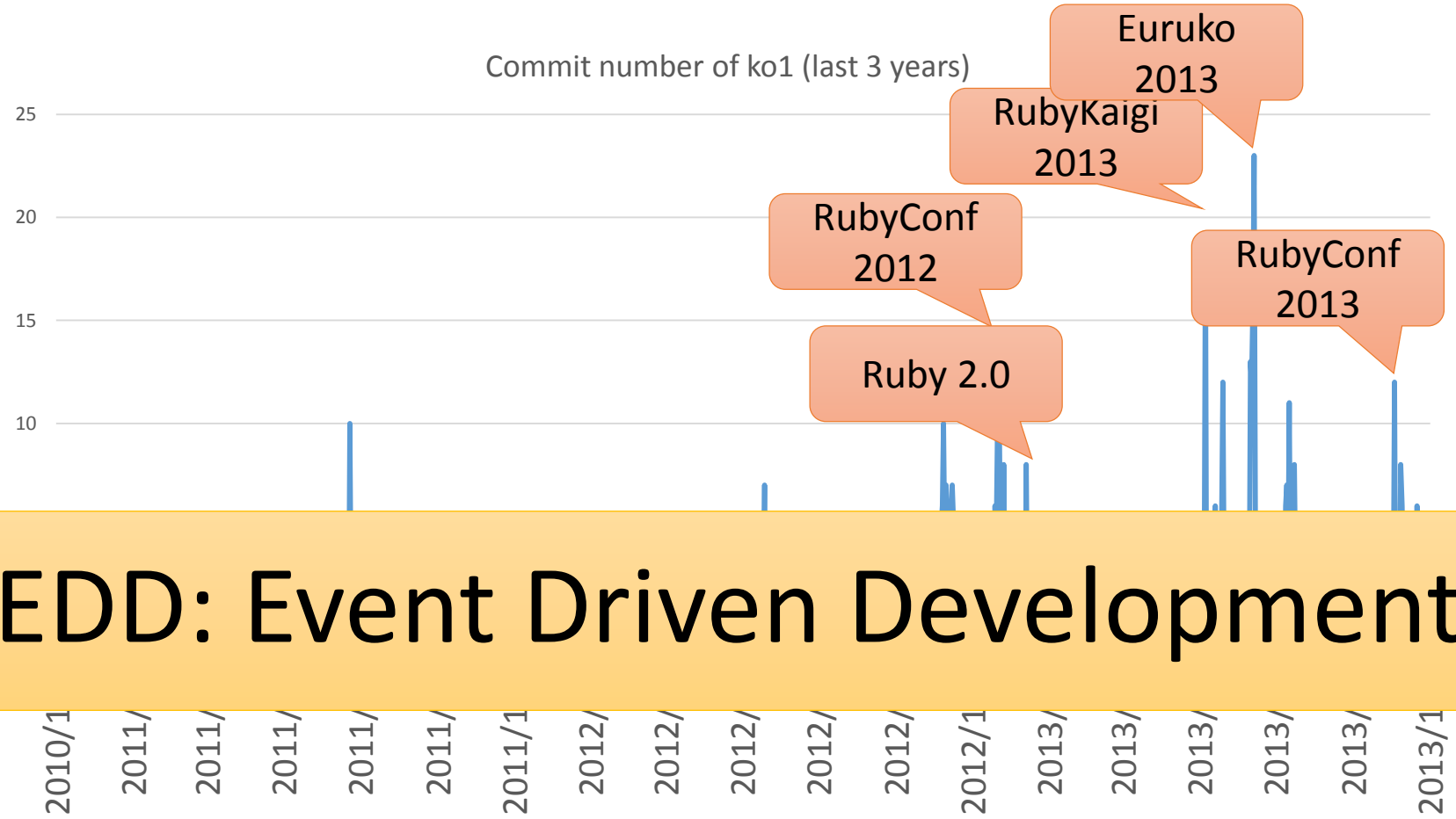
Nobu is
Great Patch Monster



heroku

Ko1

EDD developer



EDD: Event Driven Development

heroku

“Mission of Matz team”

Improve quality of next version of CRuby

heroku

“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 New syntax

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

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

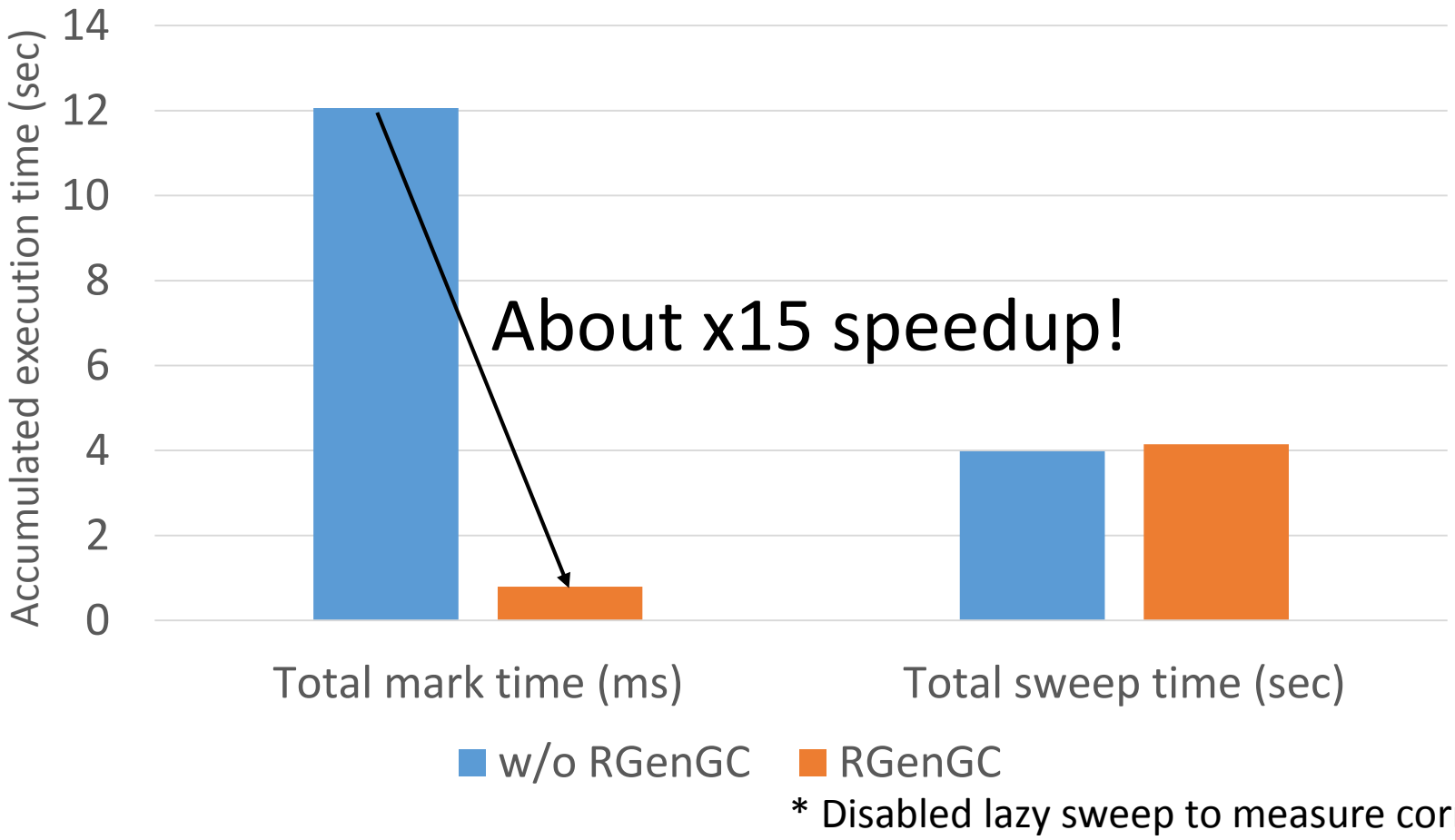
Ruby 2.1 Runtime new features

- 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
Performance evaluation (RDoc)

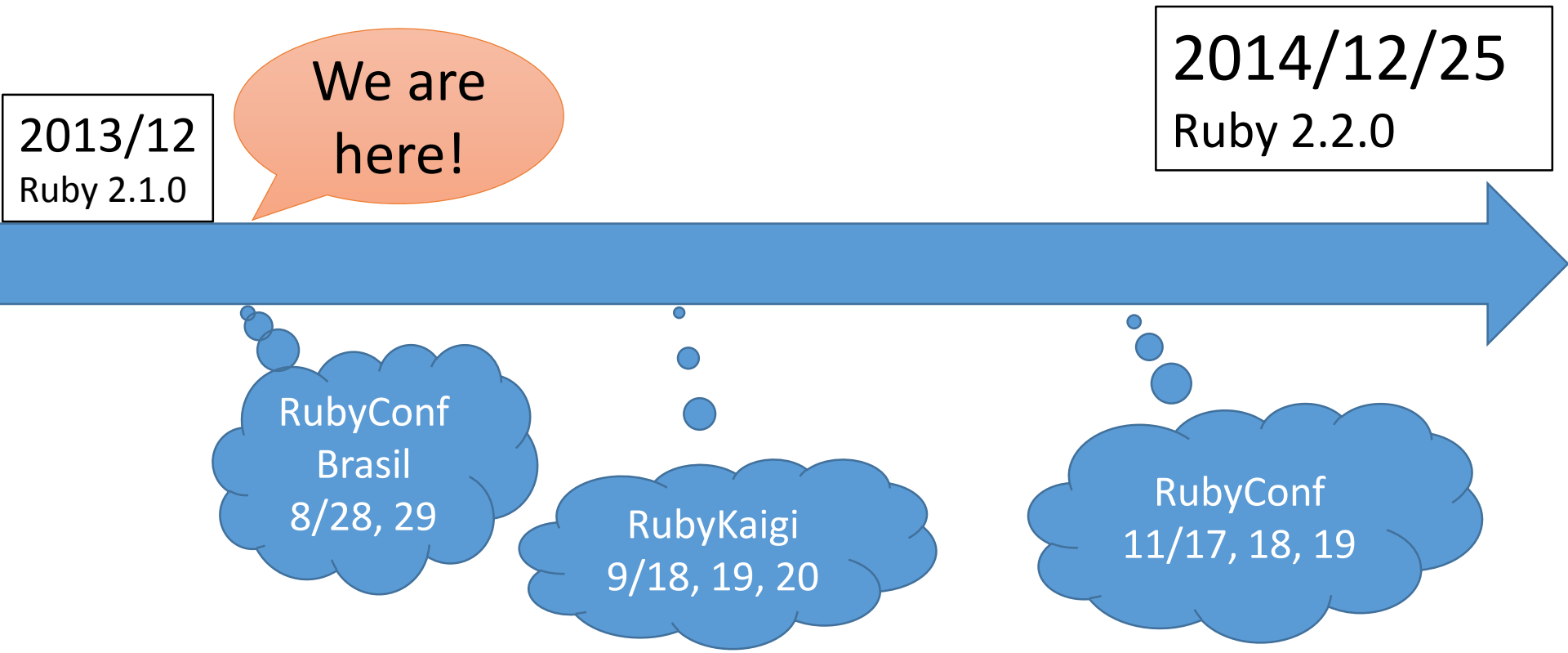


* Disabled lazy sweep to measure correctly.

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

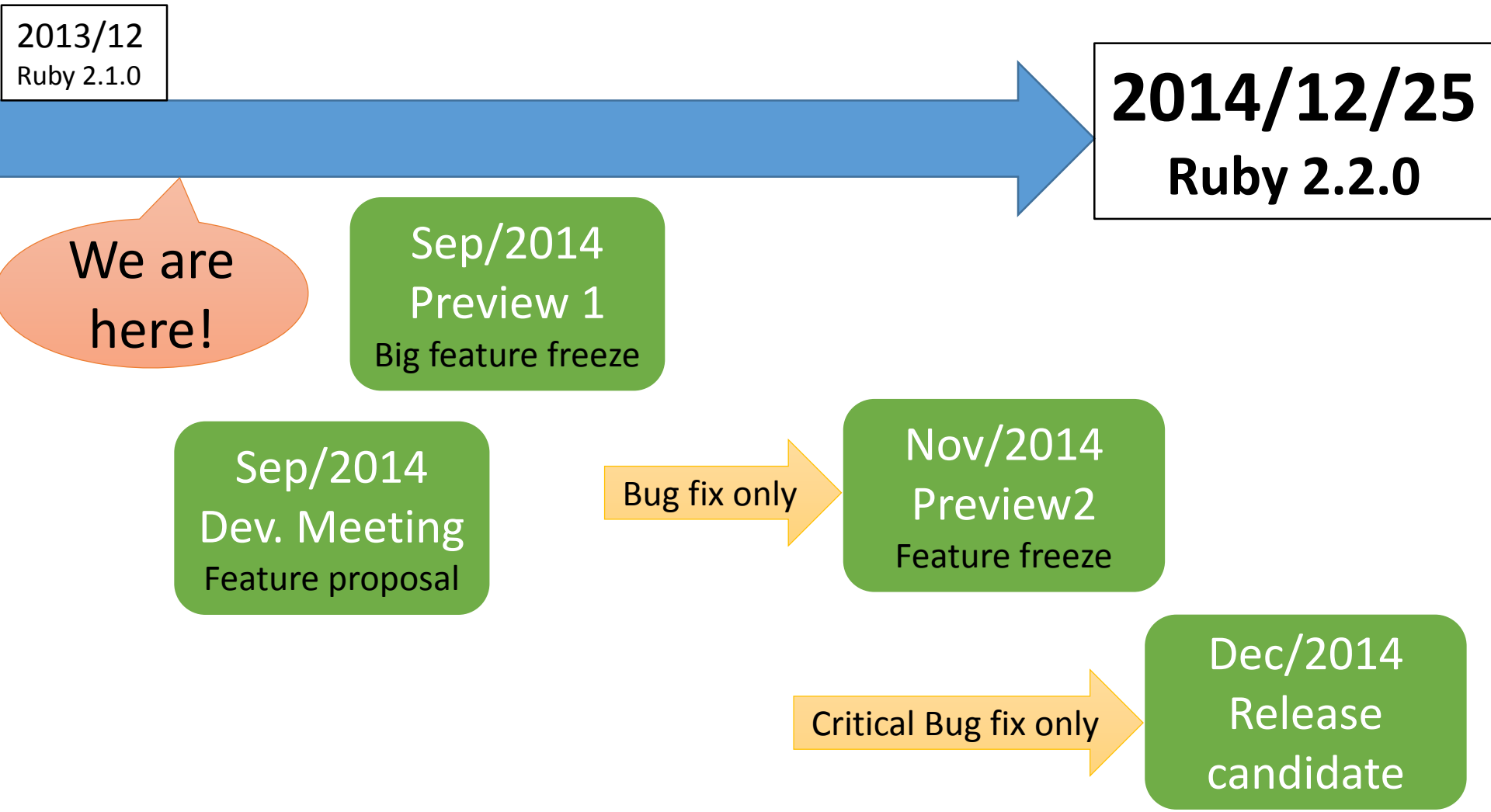
Ruby 2.2
Next version

Ruby 2.2 schedule



Events are important for EDD (Event Driven Development) Developers

Ruby 2.2 (rough) schedule



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)
    - More ages strategy to reduce too-fast-promotion
    - 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
```

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

NOTE: Drink a drop of water

Growing up the Ruby Interpreter

How do we grow up the Ruby interpreter?

Software consists of many components

Ruby's components for users

Ruby (Rails) app

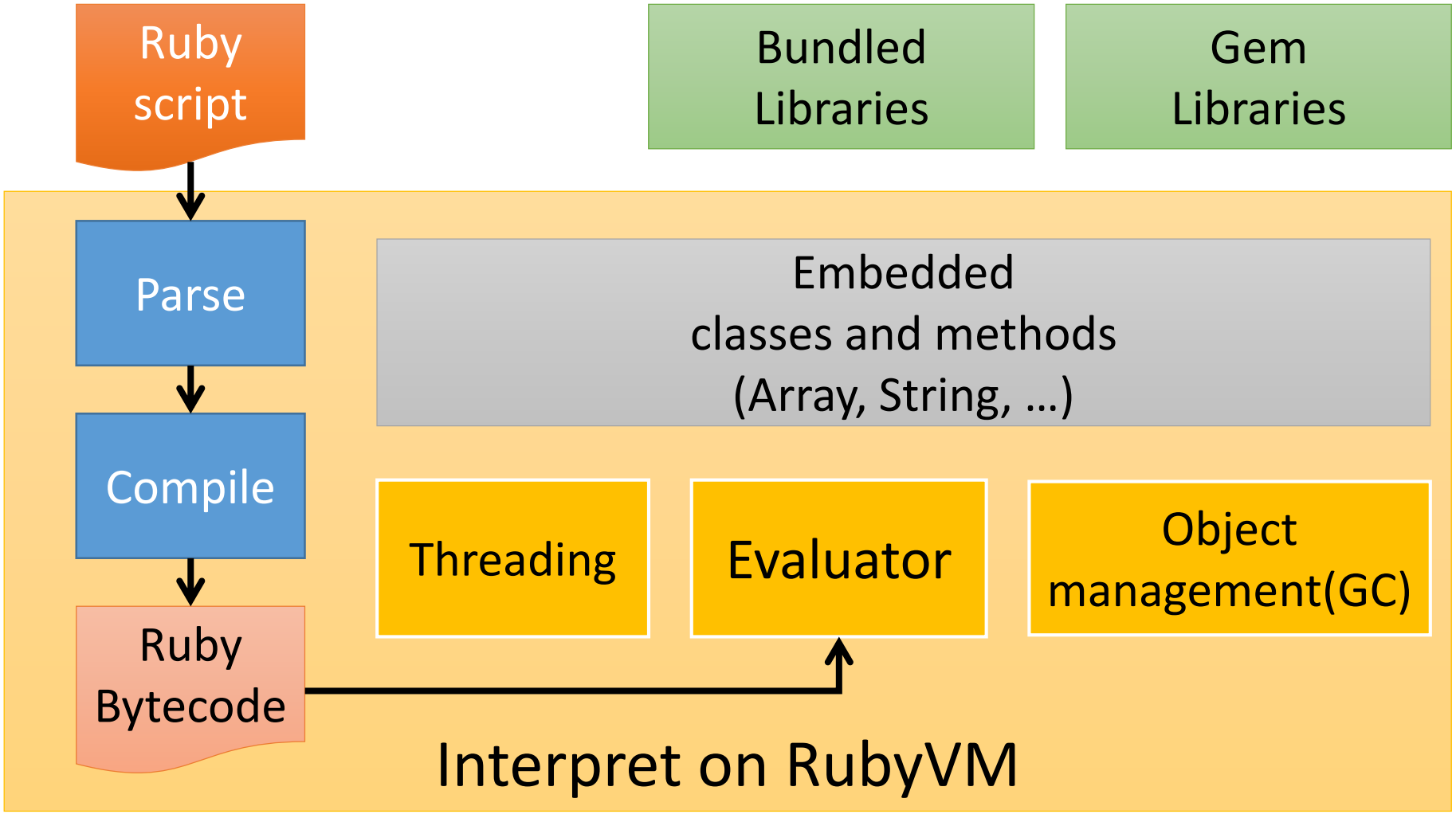
i gigantum umeris insidentes
Standing on the shoulders of giants

So many gems
such as Ruby on Rails (ActiveSupport, …) and so on.

RubyGems/Bundler

Ruby interpreter

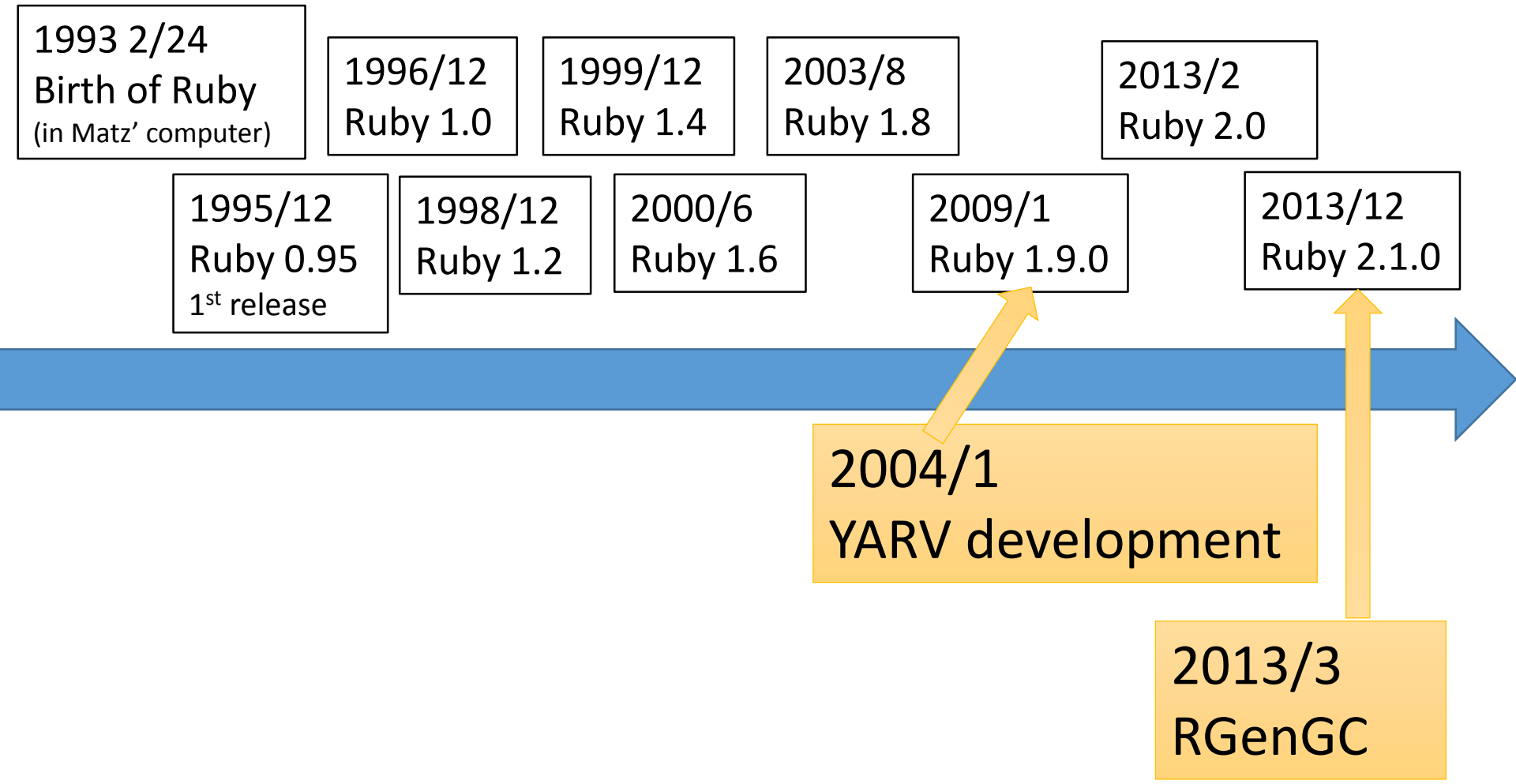
Ruby's components from core developer's perspective



My contributions

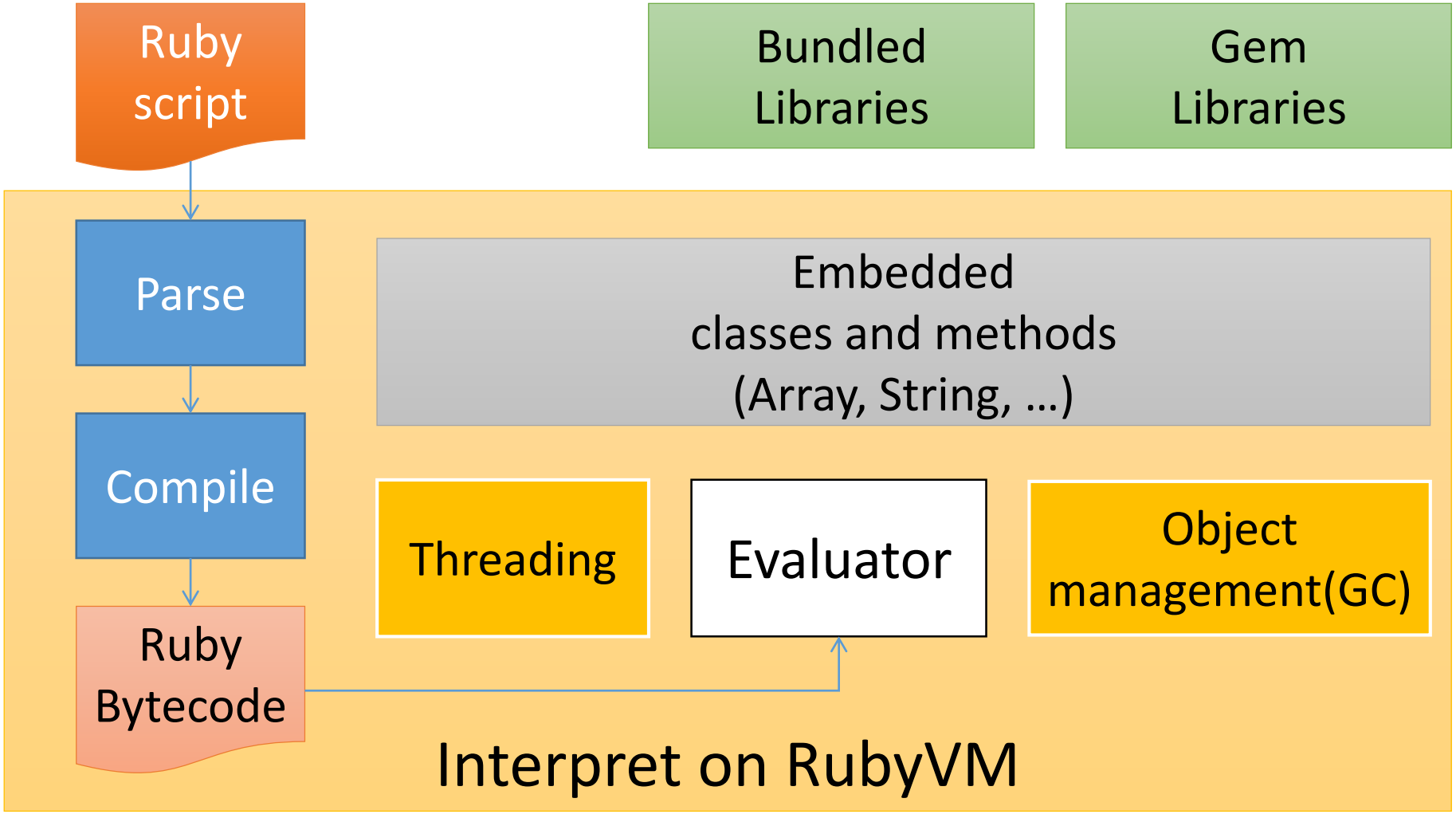
- Improve the performance for
  - Evaluator (10 years)
  - Thread management (10 years)
  - Memory management (recent years)

History of Ruby interpreter



Grow up Ruby interpreter by modification of core components

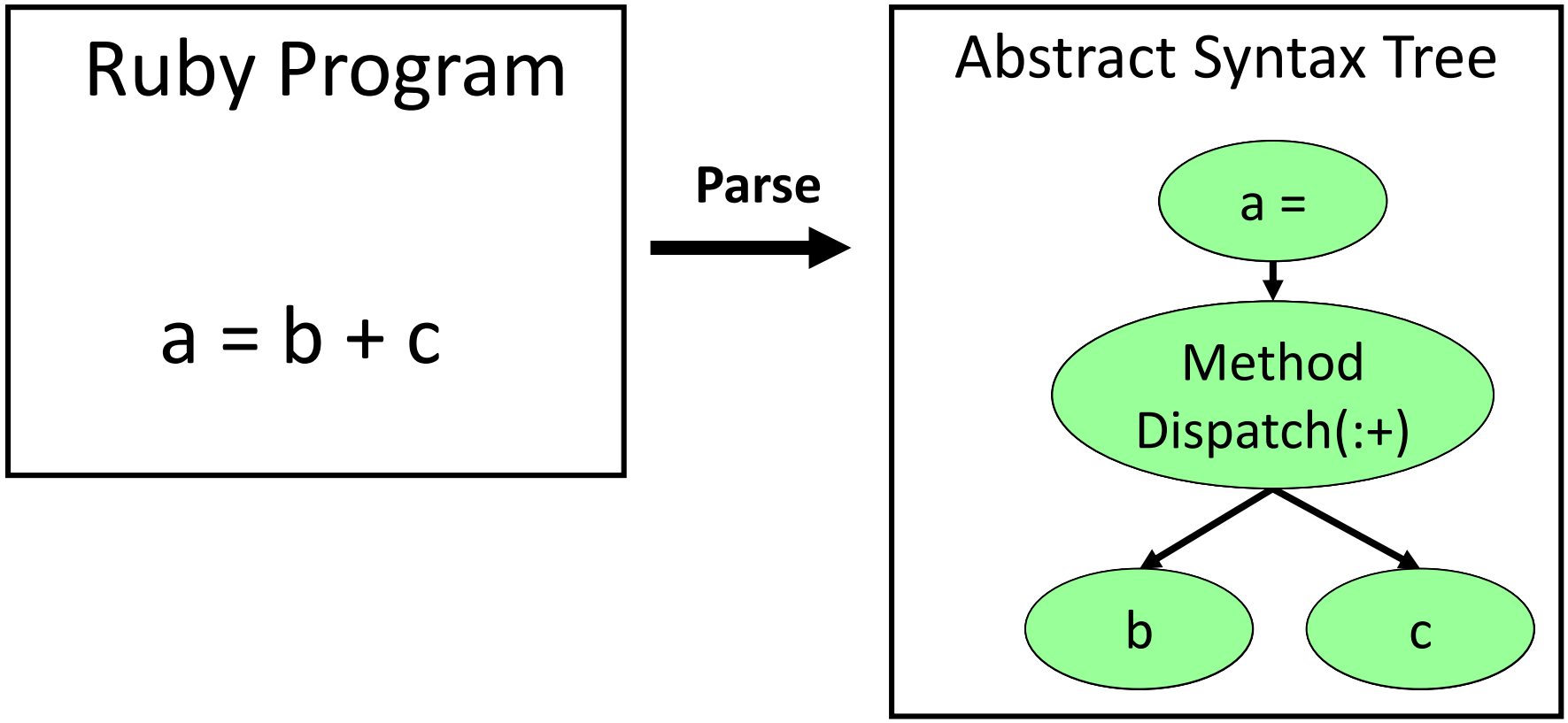
Evaluator


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

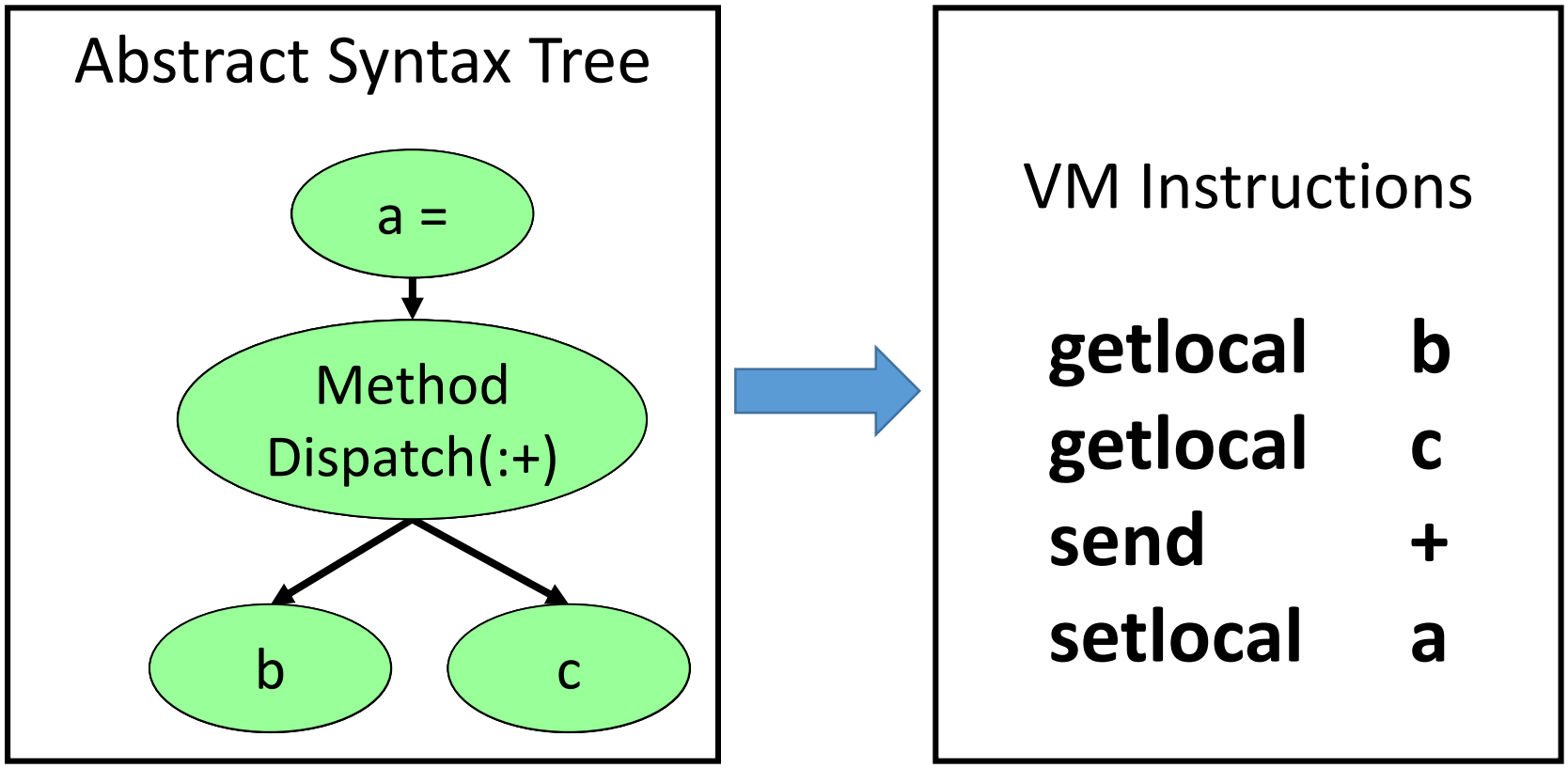
Evaluator

- 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

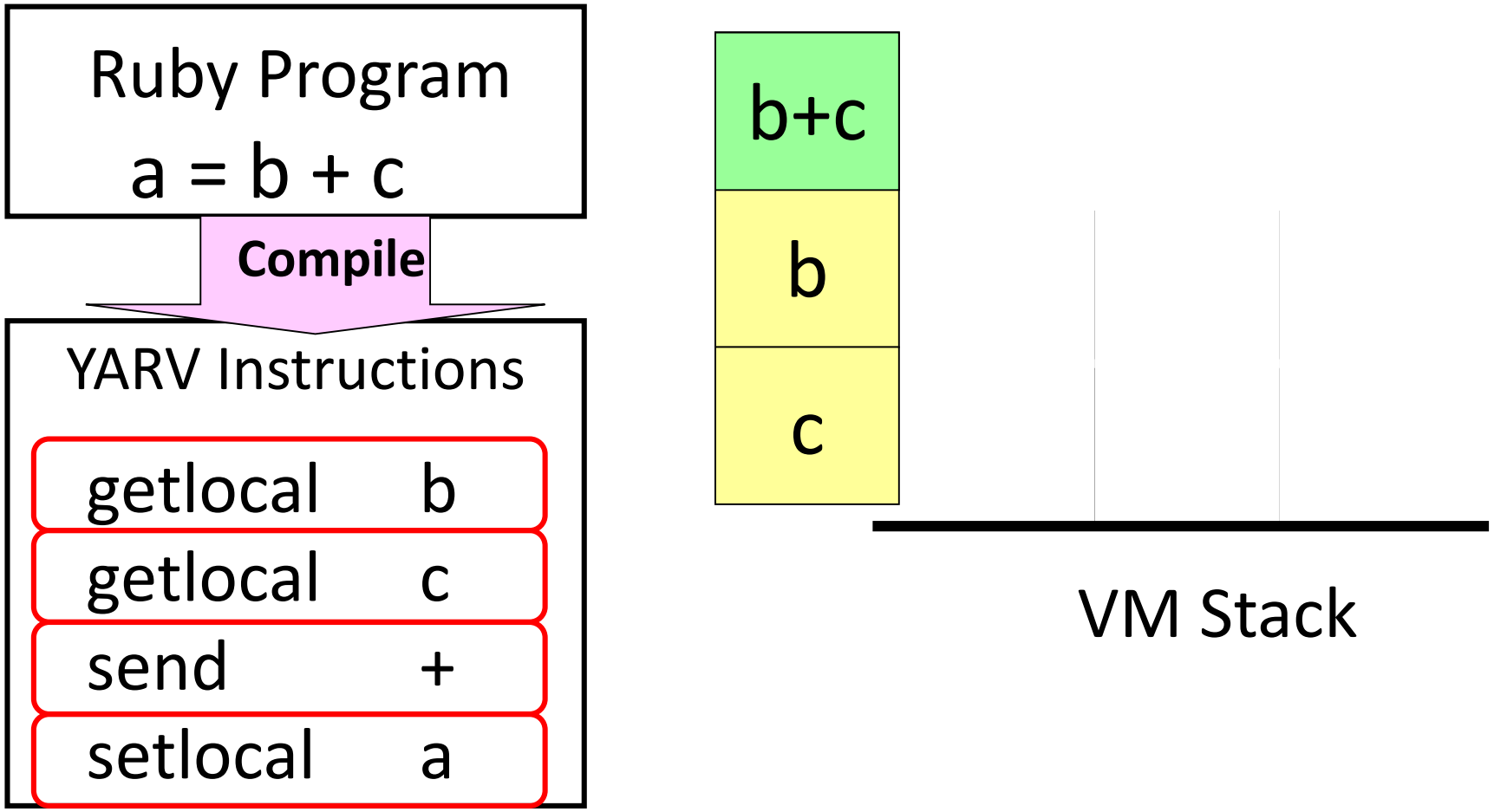
Evaluator
Compile Ruby to AST



Evaluator
Compile AST to Bytecode



Evaluator
Execution as stack machine



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
  - …

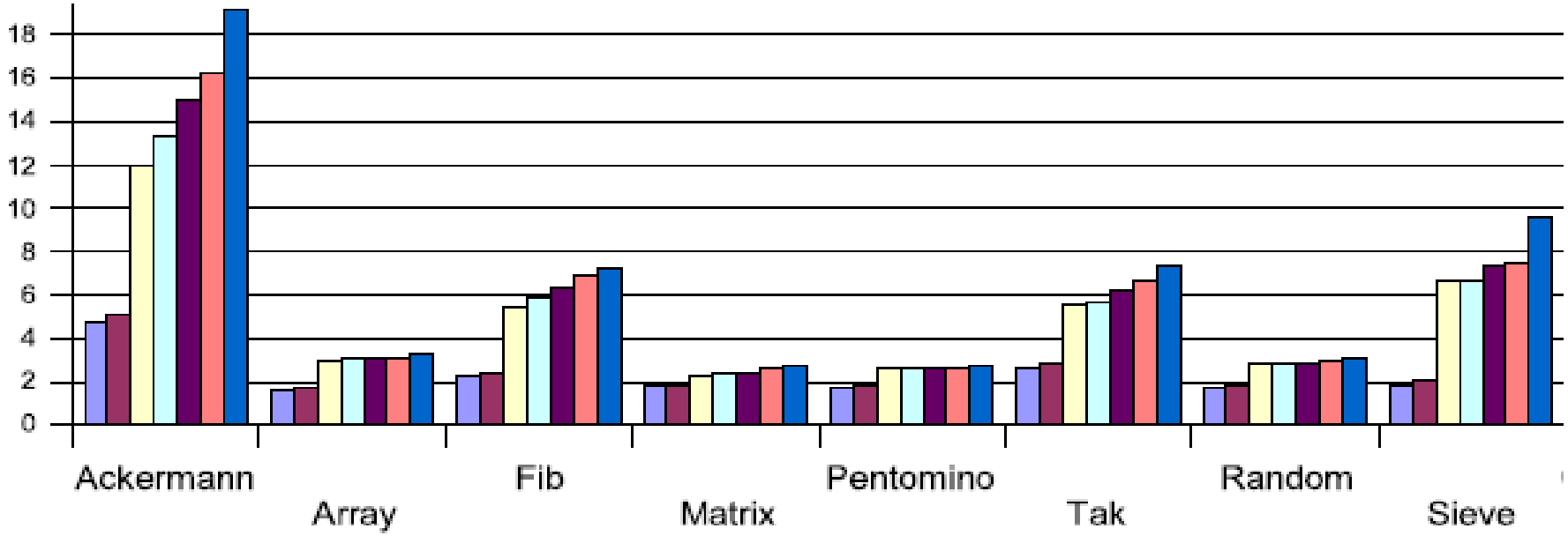
Evaluator Optimizations: Basic concept

- 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

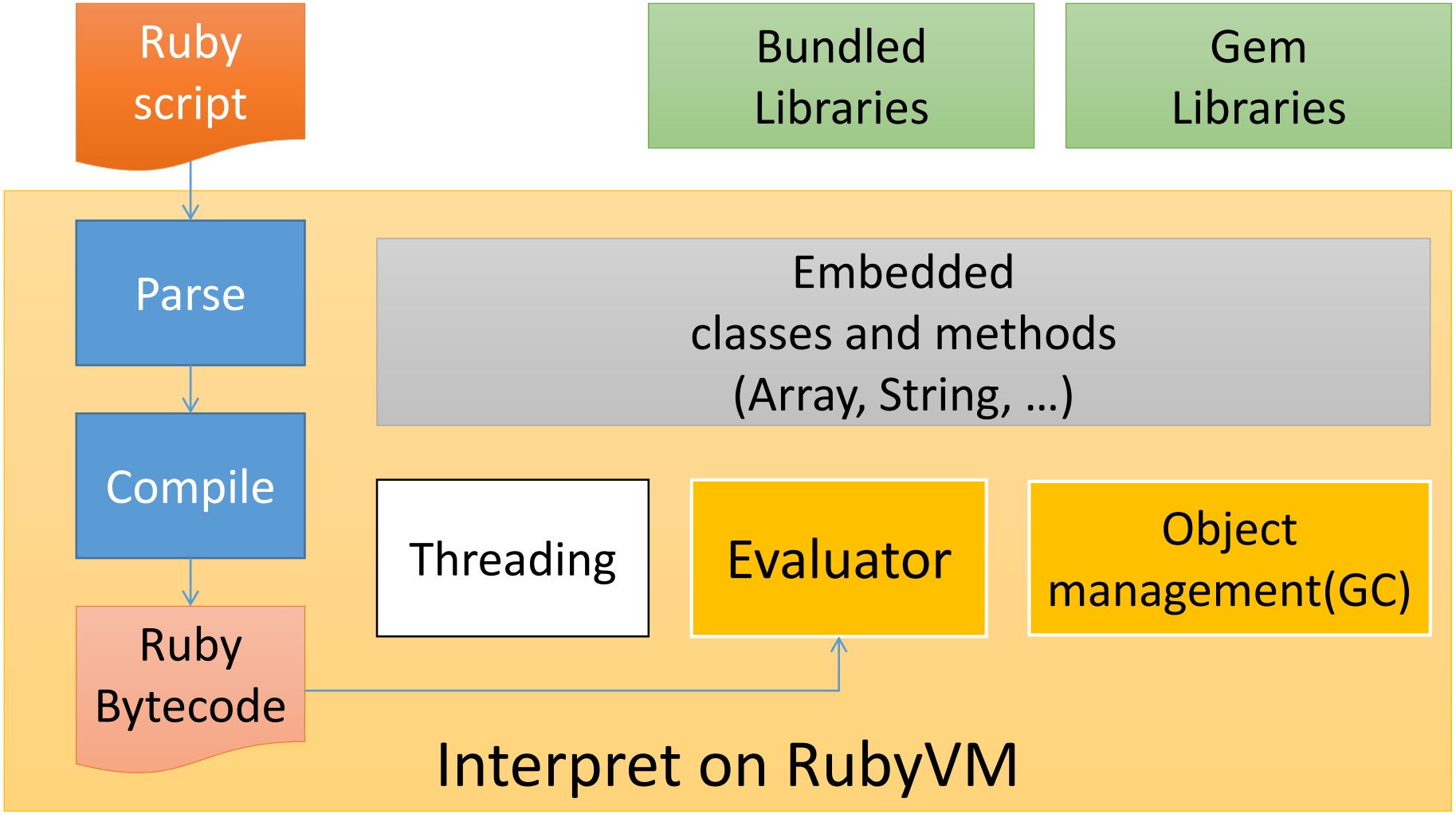
Higher is good



Main components

- Evaluator
- Thread management
- Memory management

Threading

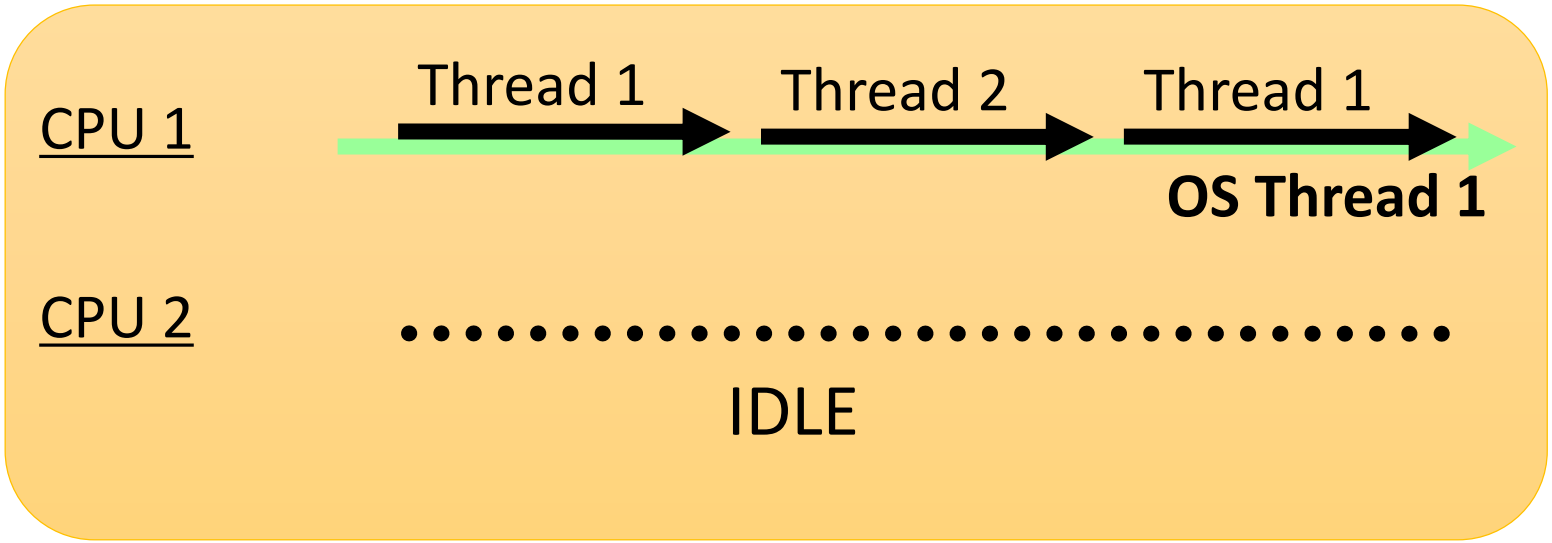

Ruby script
Bundled Libraries
Gem Libraries
Parse
Compile
Ruby Bytecode
Embedded classes and methods (Array, String, …)
Threading
Evaluator
Object management(GC)
Interpret on RubyVM

Threading

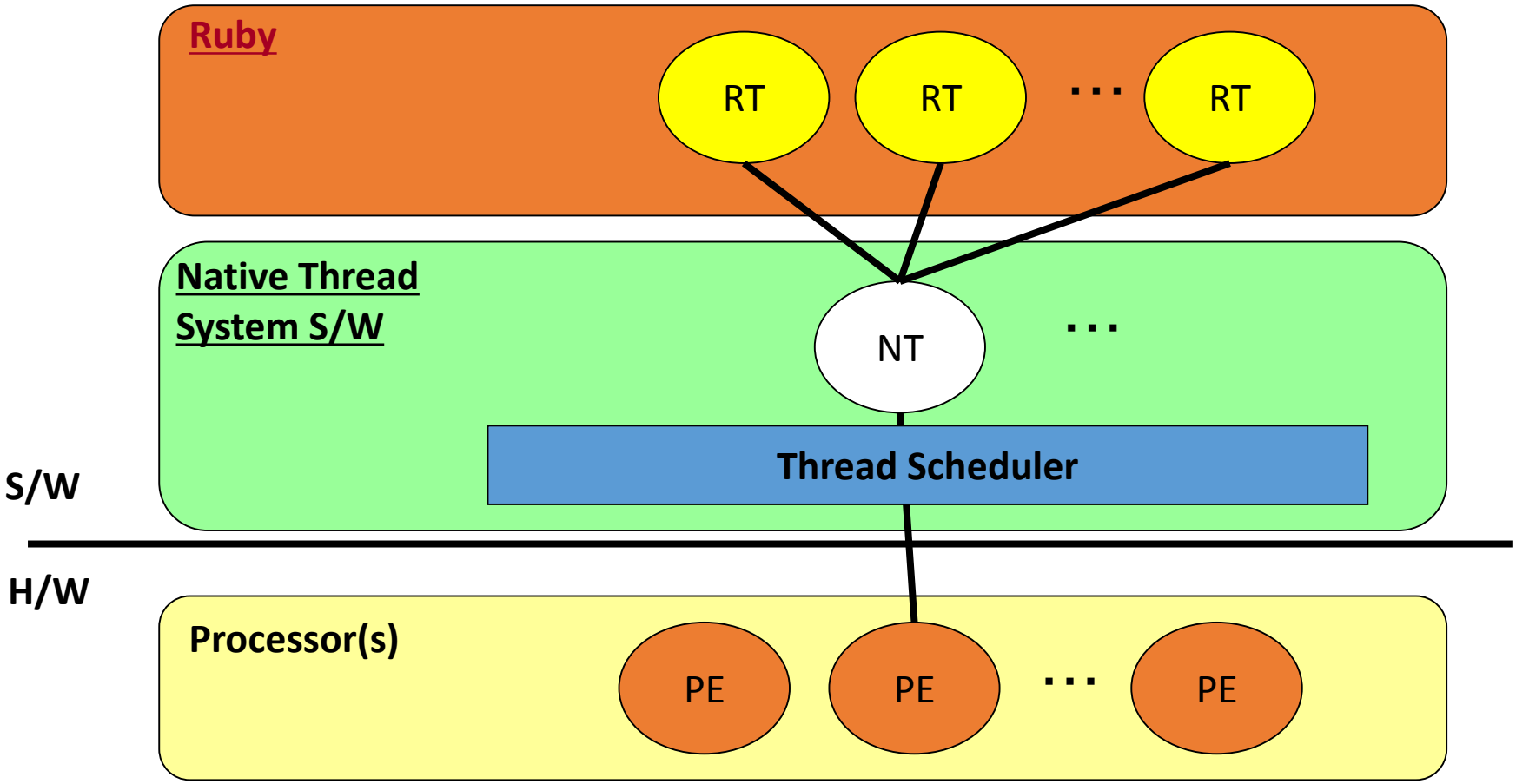
- Using native threads for each Ruby threads
- Parallel ruby execution is prohibited by GVL
  - You can free GVL if you write a code carefully in C level and run it in parallel

Threading
Ruby 1.8 and before

One OS (native) thread manages all Ruby threads
This technique is a.k.a. Green Thread

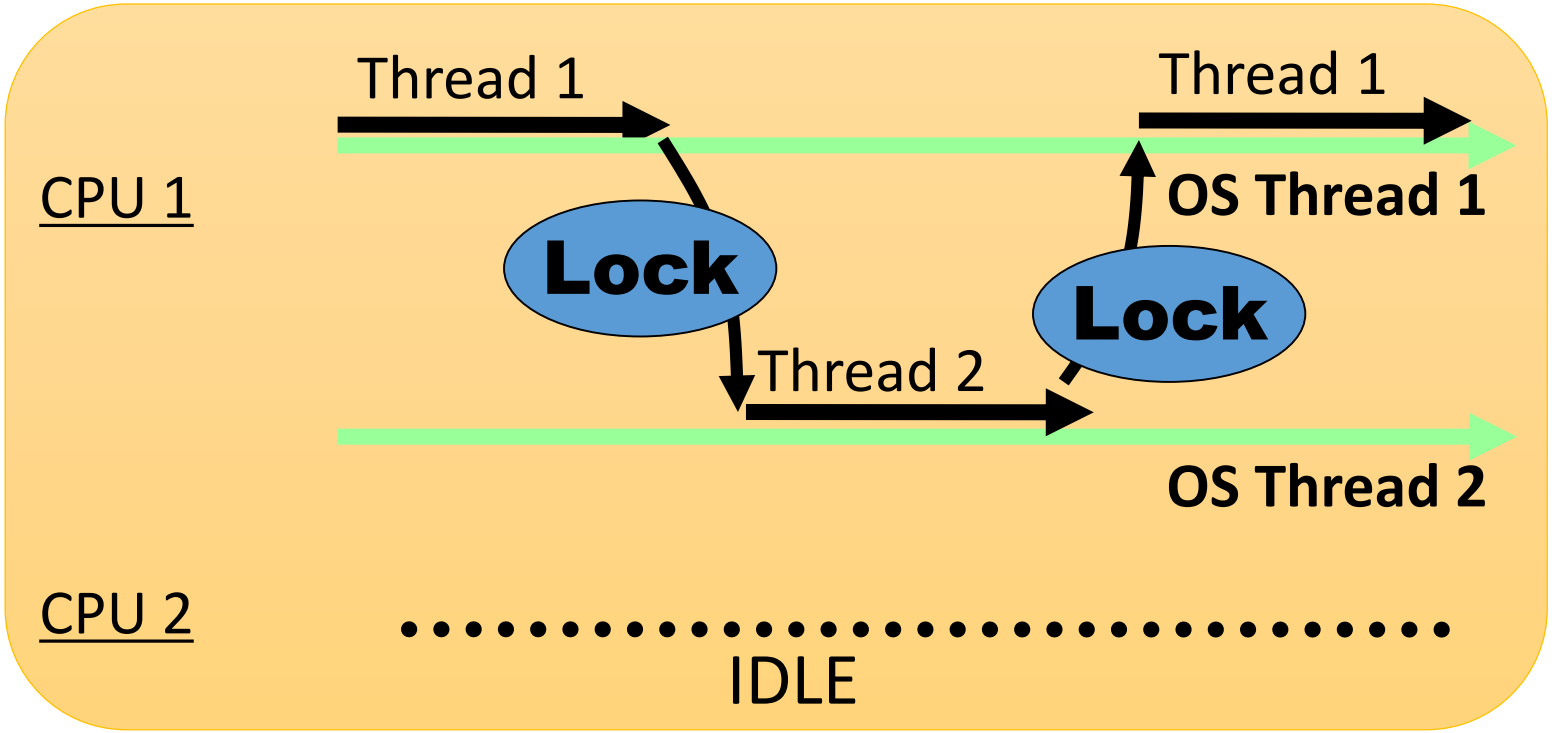


Threading
Layered view

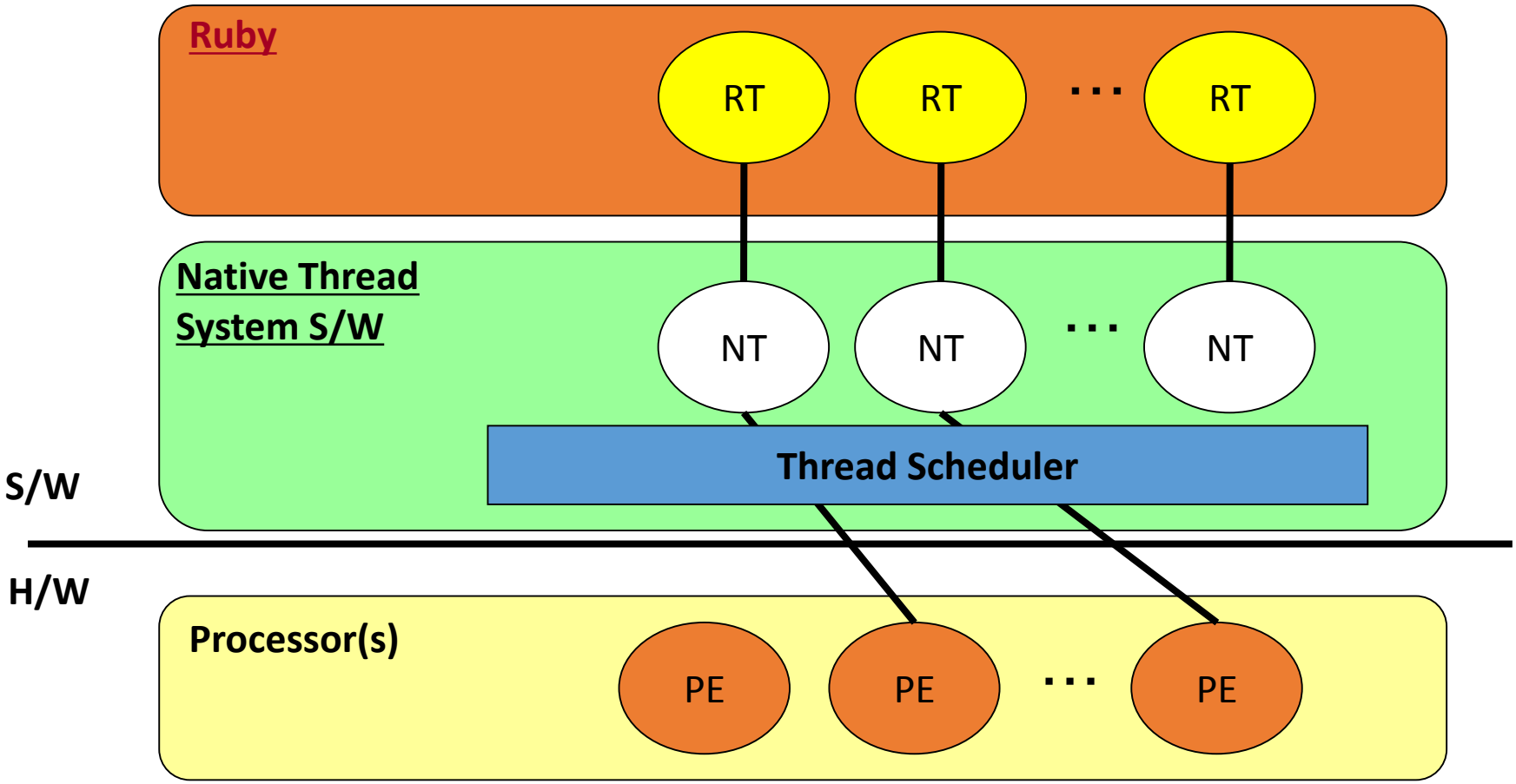


Threading
Ruby 1.9 and later

Native threads with Giant VM Lock



Threading
Layered view



Threading
Why not green threads?

- Advantage of green threads
  - Lightweight creation
- Disadvantage of green threads
  - Slow context switching (under portable way)
  - Need to take care for blocking methods
    - Such as network read/write
  - Difficult to collaborate with other C libraries using threads

We don't need to make threads frequently. (and we also have Fiber)

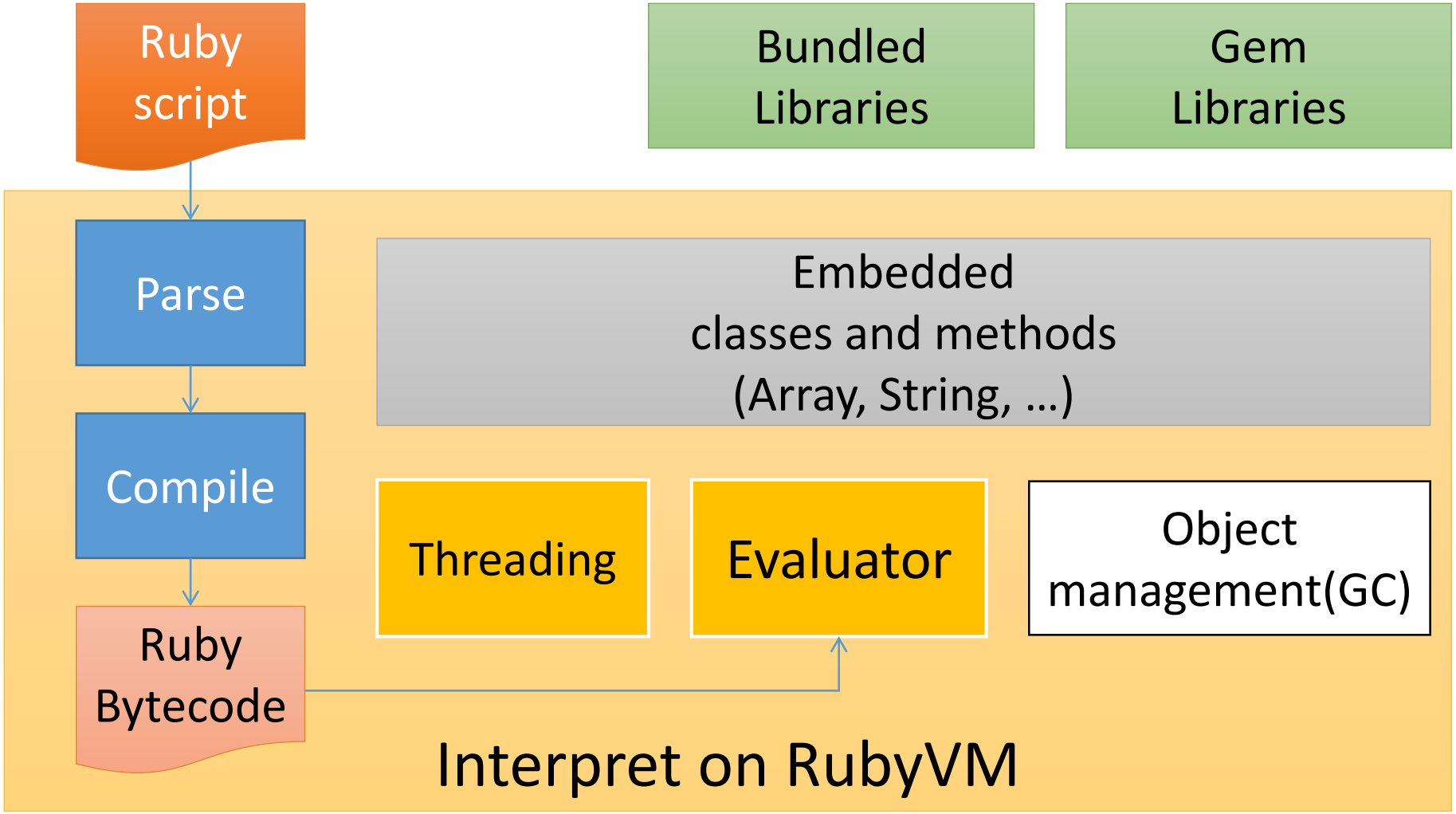
Threading
Why GVL?

- To protect Ruby users from nightmare debugging
  - Shared parallel threading can make **non-deterministic** bugs which is too hard to debug
  - “Thread programming is too difficult for human being”
- 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

Threading
How to make parallel ruby program?

- Now:
  - Use parallel threads provided by JRuby/Rubinius
    - If you think you can make correct thread programs
  - Use process (for example, w/ parallel gem, w/ dRuby)
- Future:
  - Introduce smart conventions to avoid threading bugs
    - Matz likes Actor model (Erlang)
  - Introduce limited shared memory model
  - Introduce smart debugging feature
    - Detecting bugs, avoid nondeterministic behaviors, …

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

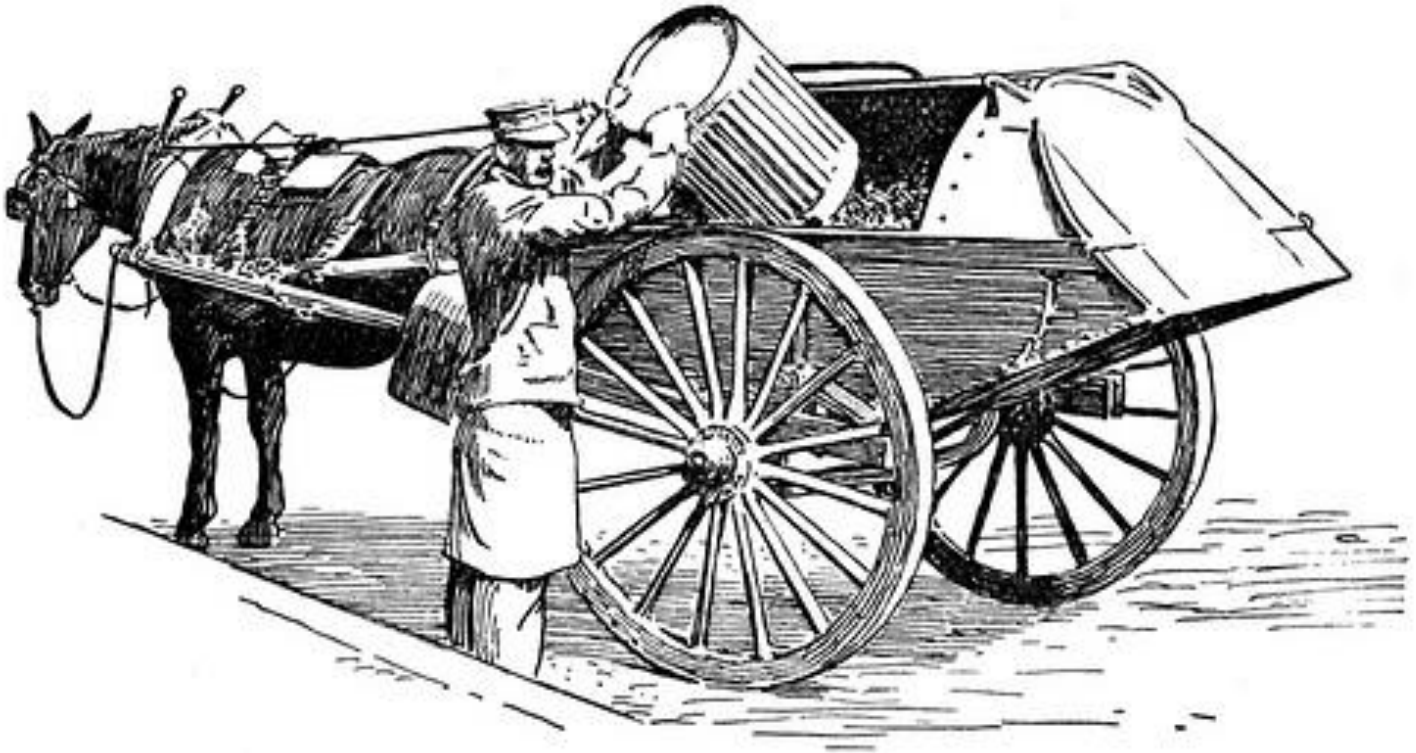


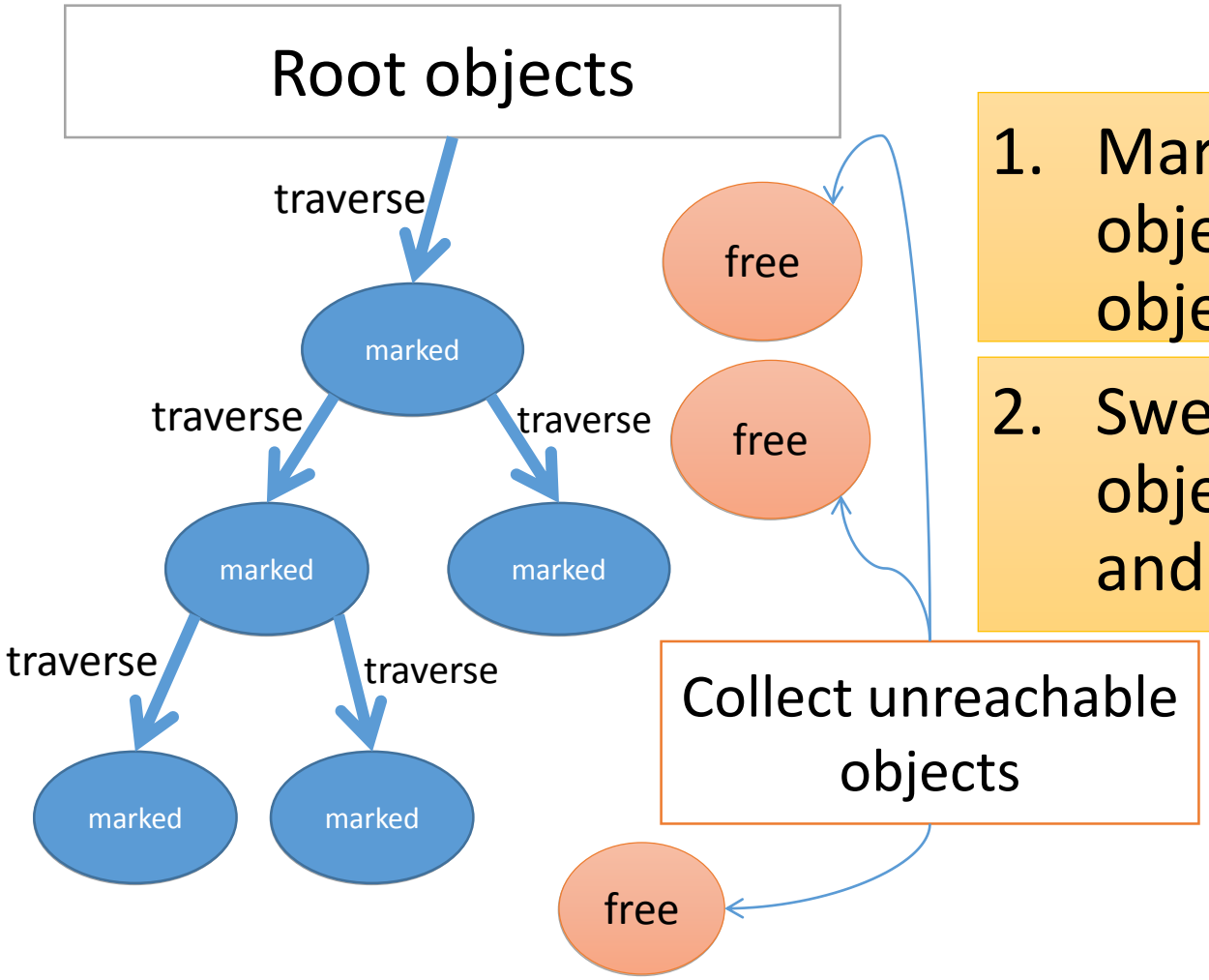
FIG. 109.—A GARBAGE COLLECTOR.

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

Automatic memory management Basic concept

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

Mark & Sweep algorithm



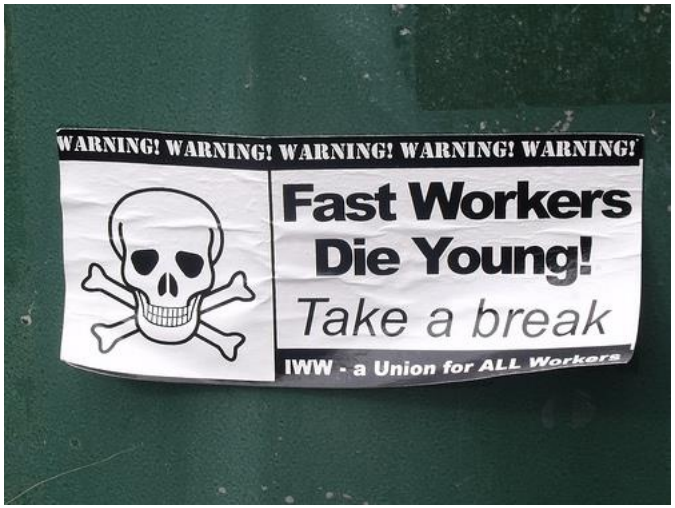
1. Mark reachable objects from root objects
2. Sweep ***unmarked*** objects (collection and de-allocation)

RGenGC: Restricted Generational GC

Generational GC (GenGC) from Ruby 2.1

- Weak generational hypothesis:

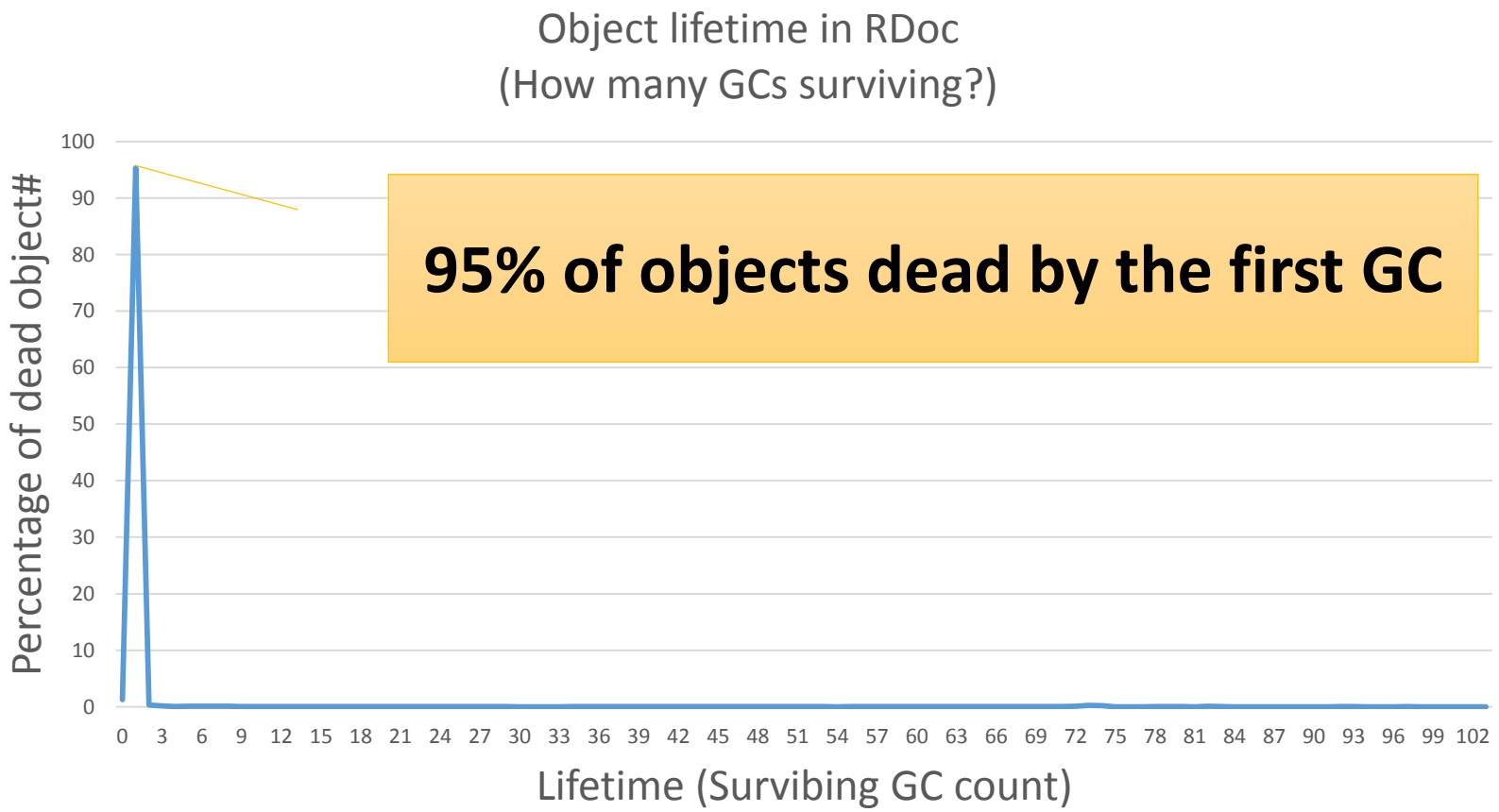
“Most objects die young”



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

→ Concentrate reclamation effort only on the young objects

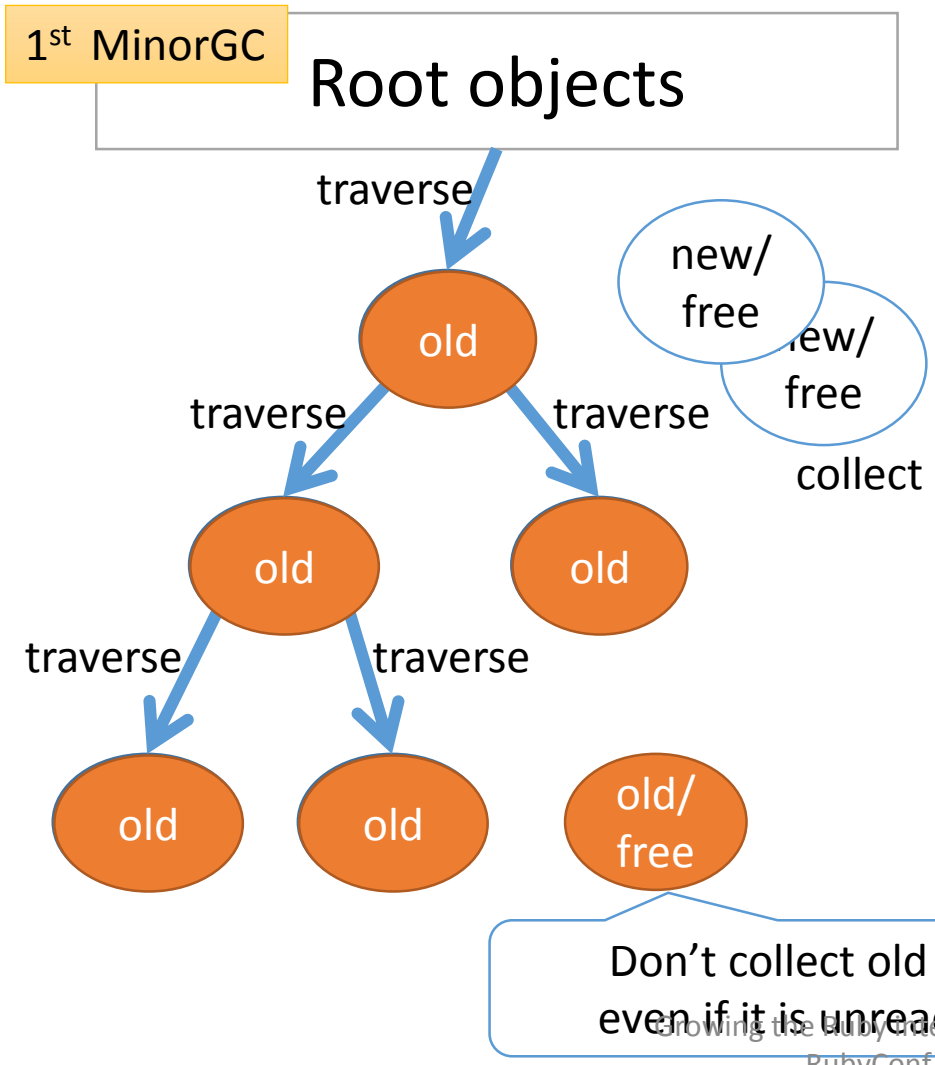
Generational hypothesis



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 or 3 from Ruby 2.2
- Usually, GC on young space (minor GC)
- GC on both spaces if no memory (major/full GC)

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



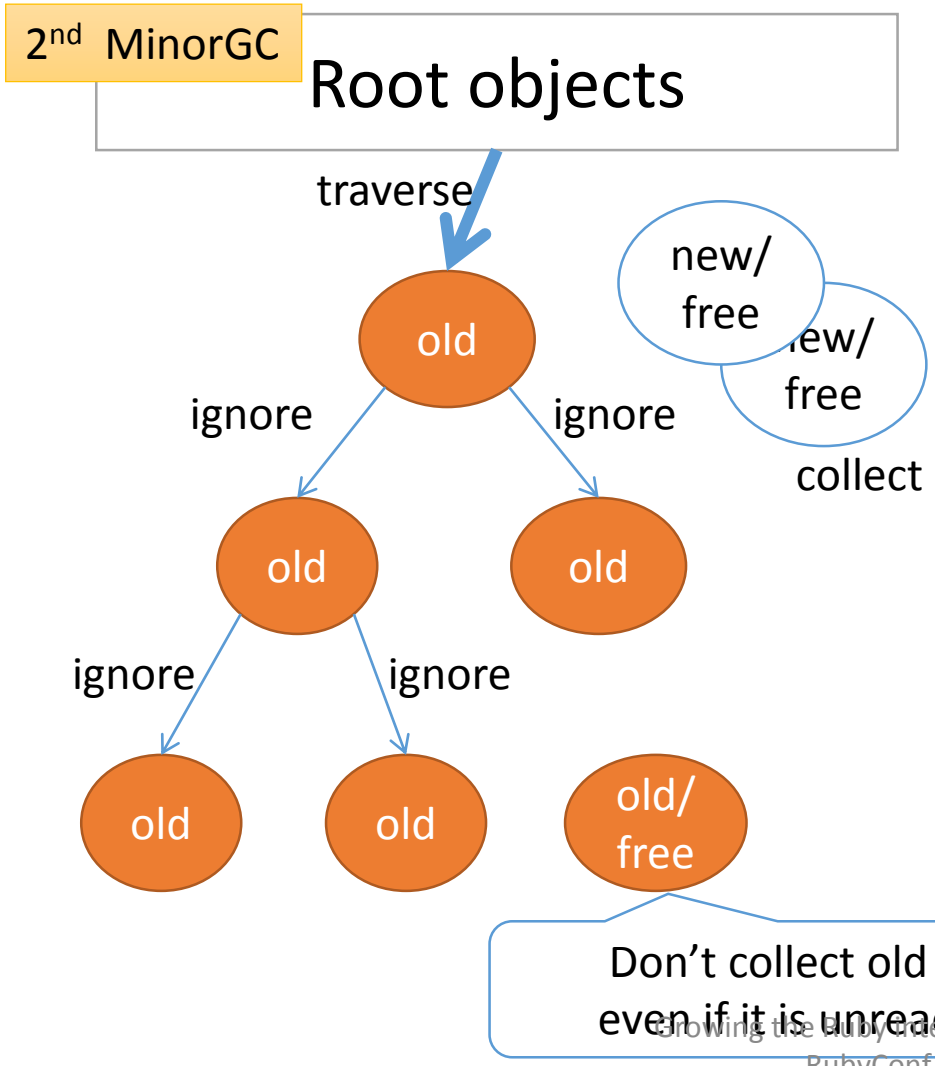
- Mark reachable objects from root objects.
  - Mark and **promote to old generation**
  - Stop traversing after old objects

→ Reduce mark overhead

- Sweep not (marked or old) objects

- Can't collect Some unreachable objects

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

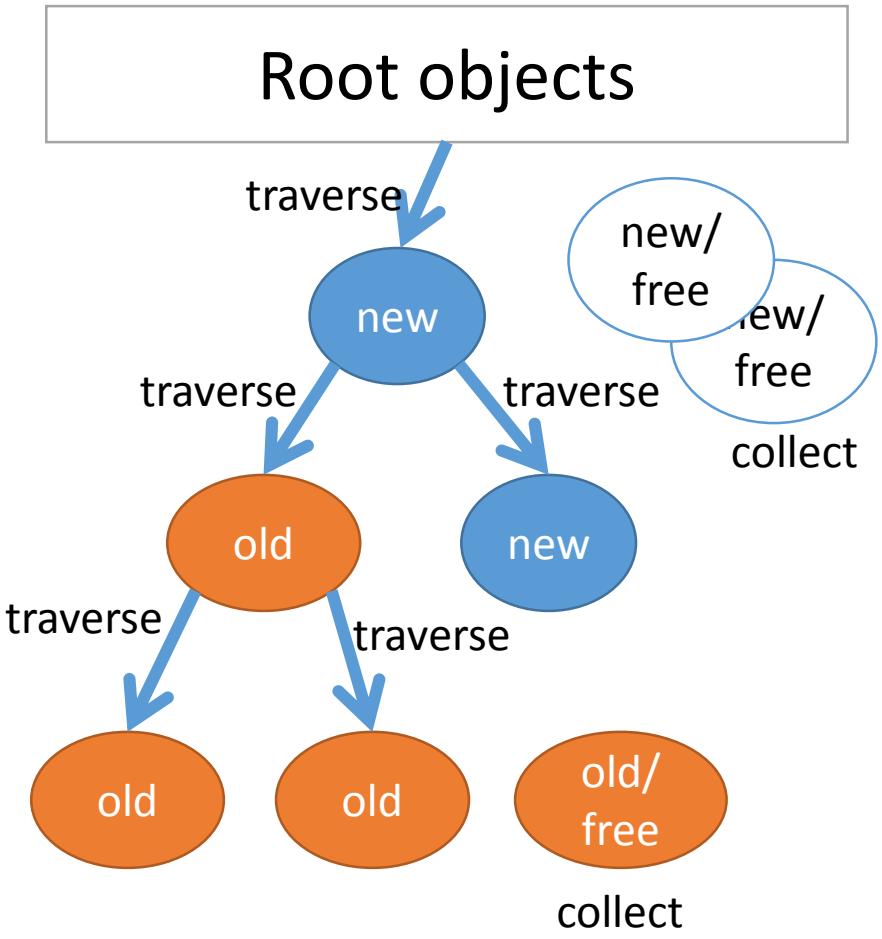


- Mark reachable objects from root objects.
  - Mark and **promote to old generation**
  - Stop traversing after old objects

→ Reduce mark overhead

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

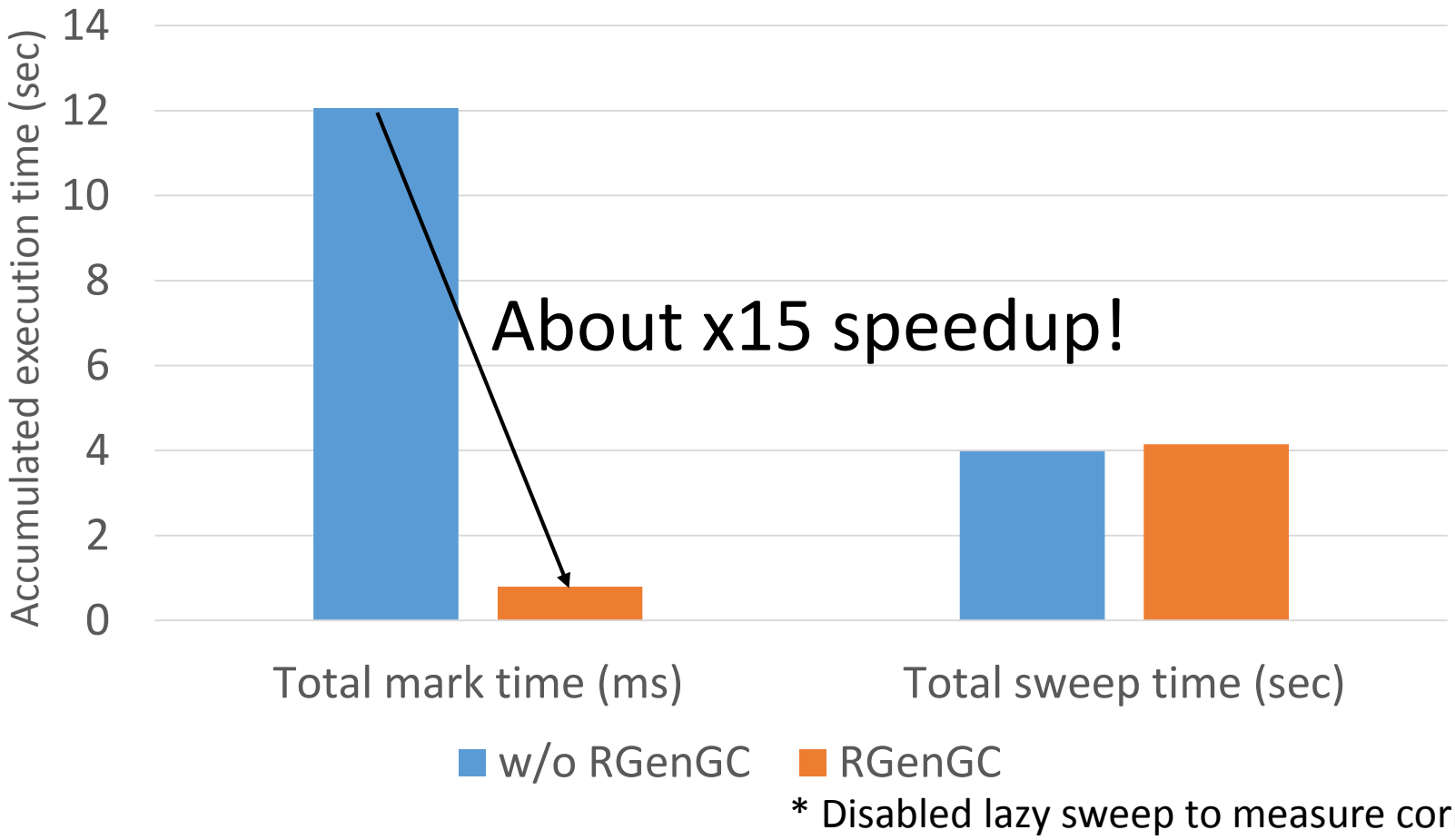
GenGC [Major M&S GC]



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

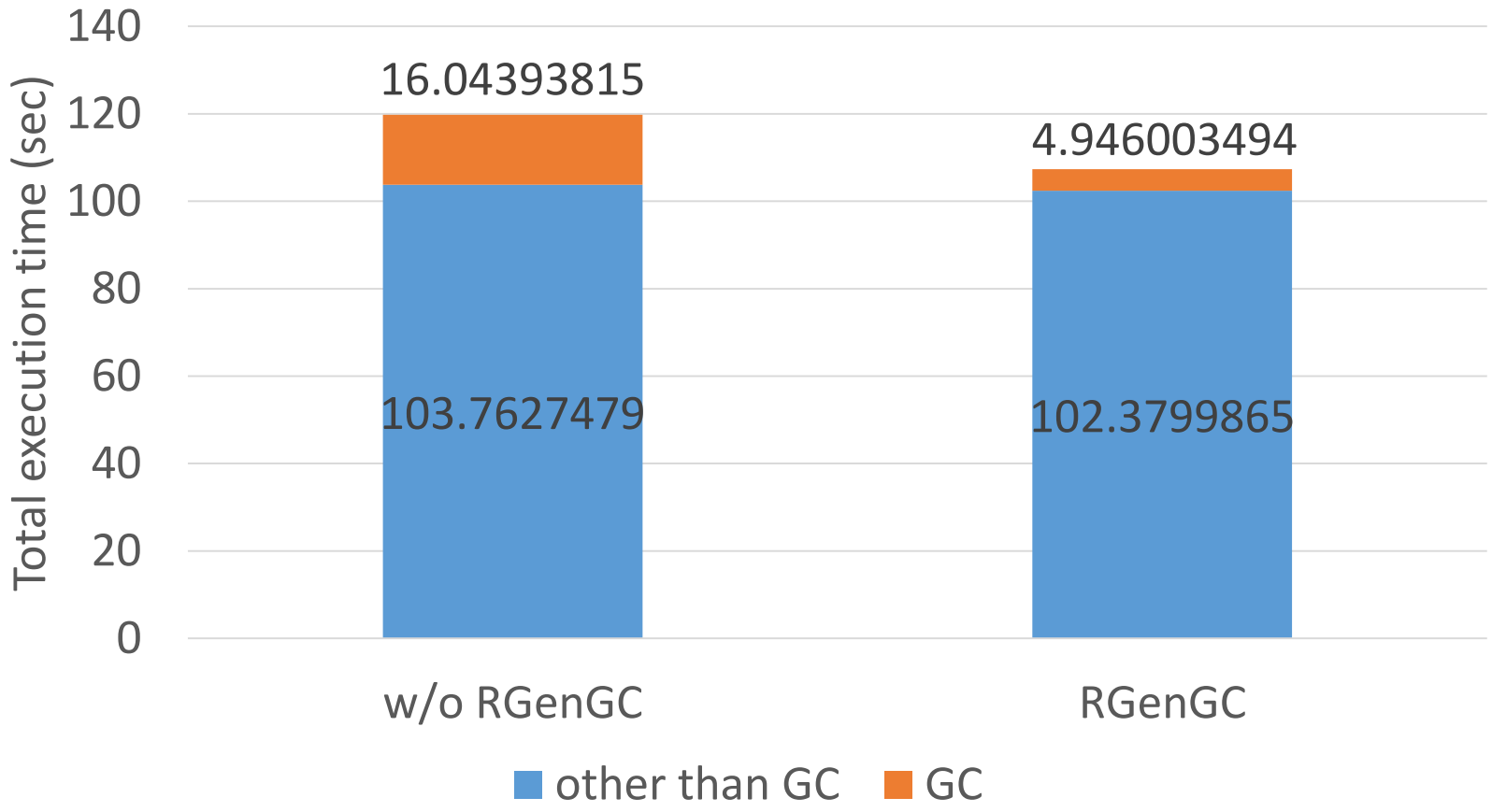
- *Sweep all unreachable (unused) objects*

RGenGC
Performance evaluation (RDoc)



* 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.

RincGC:
Restricted incremental GC

RincGC
Background and motivation

- Ruby 2.1 had introduced generational GC
  - Short marking time on minor GC
  - Improve application throughput
- Still long pause time on major GC
  - Long pause time affects user response time

Proposal:
RincGC: Incremental GC for major GC

- Introducing incremental GC to reduce pause time
- Can combine with Generational GC

	Generational GC	Incremental GC	Gen+Inc GC
Throughput	High	Low (a bit slow)	High
Pause time	Long	Short	Short

RincGC: Base idea Incremental GC algorithm

- Move forward GC processes incrementally
  - Mark slots incrementally
  - Sweep slots incrementally
- Incremental marking in 3 phase
  - (1) Mark roots (pause)
  - (2) Mark objects reachable from roots (incremental)
  - (3) Mark roots again, and mark remembered objects (pause)
- Mark objects with three state (white/grey/black)
  - White: Untouched objects
  - Grey: Marked, and prepare to mark directly reachable objects
  - Black: Marked, and all directly reachable objects are marked
- Use write barriers to avoid marking miss from marked objects to live objects
  - Detect new reference from black objects to white objects
  - Remember such source black objects (marked at above (3))

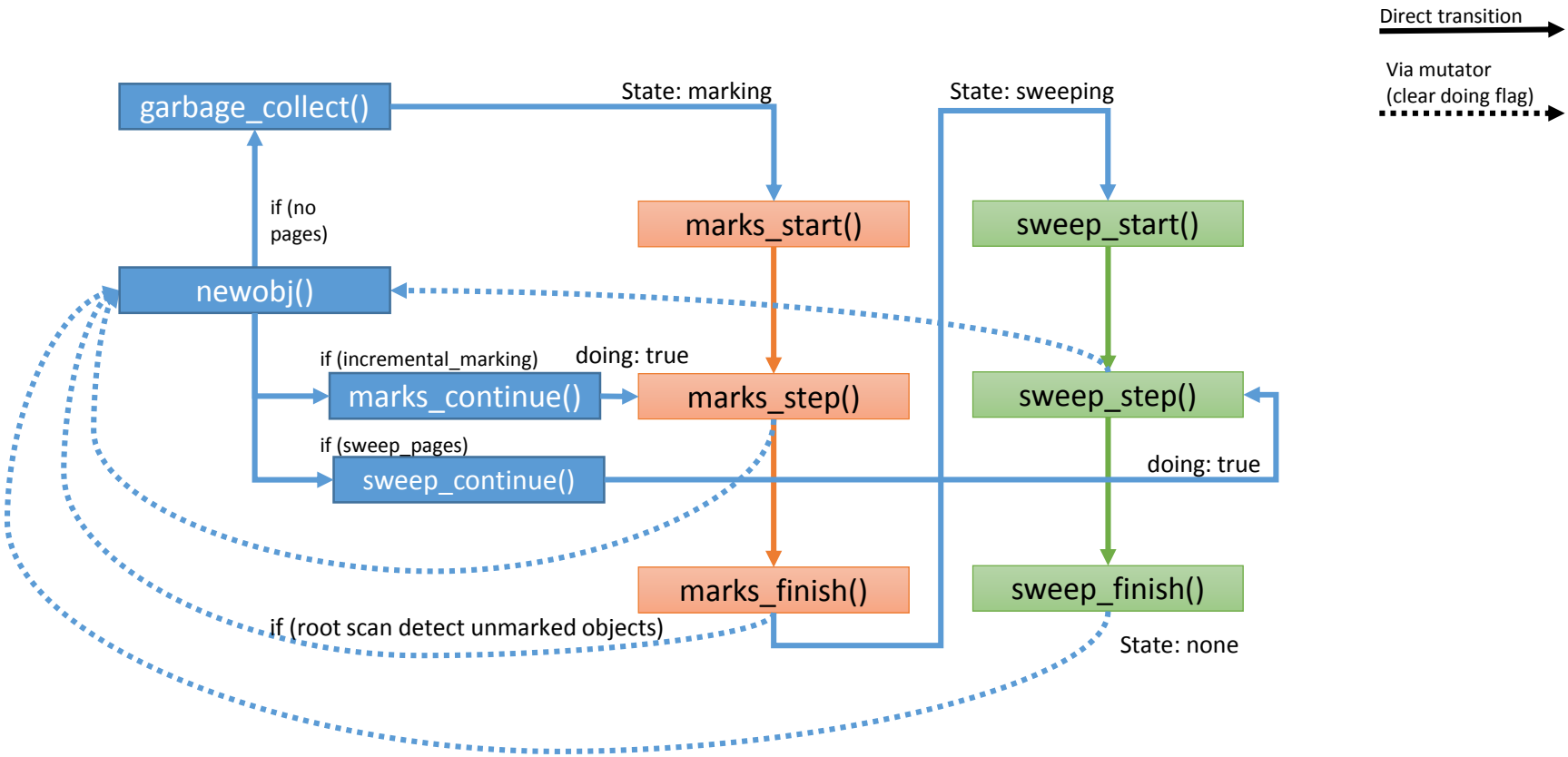
RincGC: Incremental GC for CRuby/MRI

- Incremental marking
  - (1) mark roots (gc_mark_roots())
  - (2) Do incremental mark at rb_newobj_of()
  - (3) Make sure write barrier with WB-protected objects
  - (4) Take care of **WB-unprotected objects** (MRI specific)
- Incremental sweeping
  - Modify current lazy sweep implementation

RincGC: Incremental marking

- (1) mark roots (gc_mark_roots())
  - Push all root objects onto “mark_stack”
- (2) Do incremental mark at rb_newobj_of()
  - Fall back incremental marking process periodically
  - Consume (pop) some objects from “mark_stack” and make forward incremental marking
- (3) Make sure write barrier with WB-protected objects
  - Mark and push pointed object onto “mark_stack”
- (4) Take care of **WB-unprotected objects** (MRI specific)
  - After incremental marking (“mark_stack” is empty), re-scan all roots and all living non-WB-protected objects
  - WB-unprotected objects are represented by bitmap (WB_UNPROTECTED_BITS)

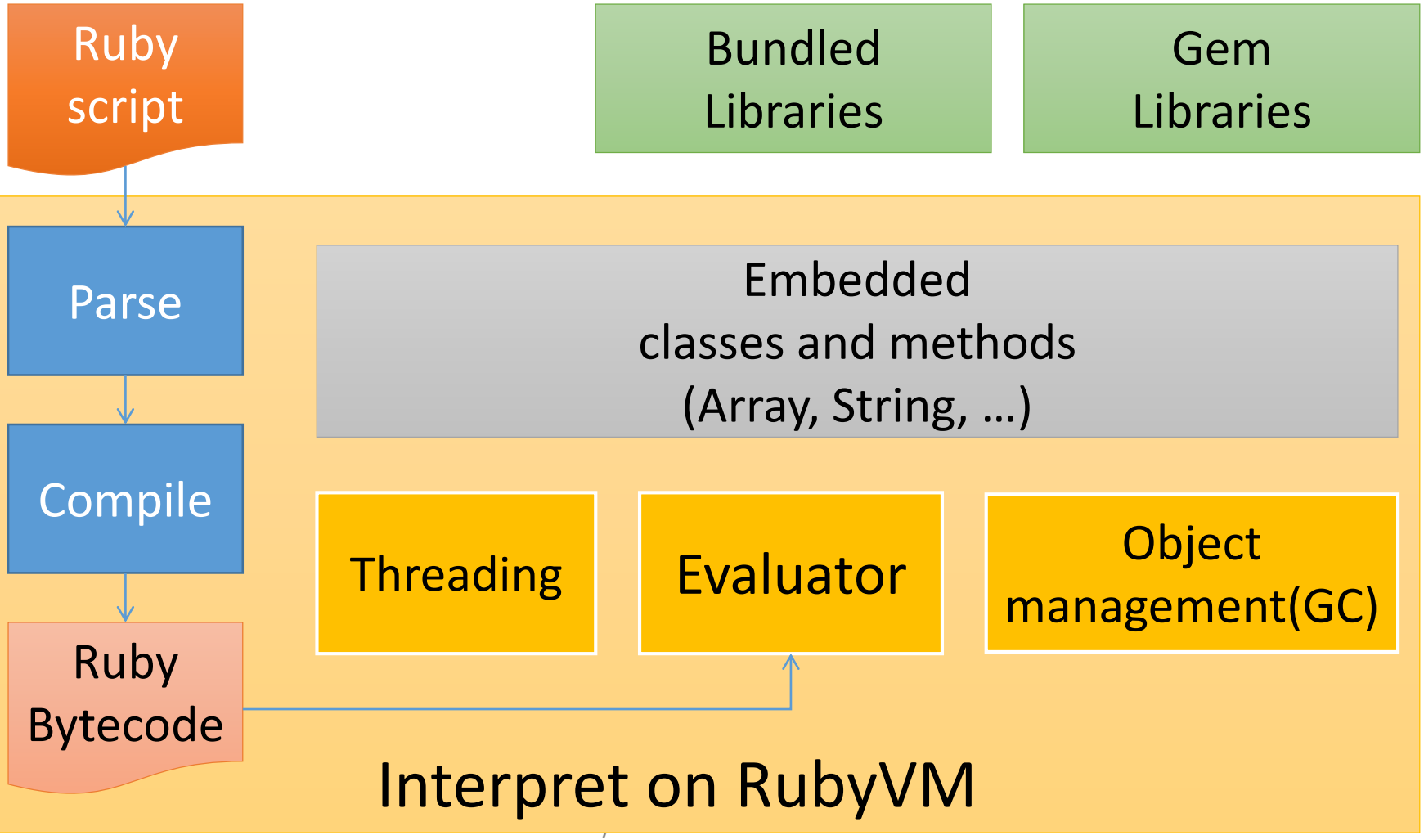
RincGC: Diagram



Growing up the Ruby Interpreter

How do we grow up the Ruby interpreter?

DO EVERYTHING!
NO SILVER BULLET!



DO EVERYTHING!
NO SILVER BULLET!

Loop do

- Survey techniques
- Implement techniques
- Invent new techniques
- Evaluate techniques

end # endless

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 threading?)
- Object management and GC
  - Compaction GC
  - Lightweight object allocation
  - CoW friendly memory management
- And more

Summary

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

Summary

- Ruby 2.1 and Ruby 2.2
- How to grow up the Ruby interpreter?

My answers is:

#=> Continue software development

(with love?)

Thank you for your attention

Koichi Sasada

<ko1@heroku.com>

heroku