Object management on Ruby 2.1

Koichi Sasada

Heroku, Inc.

ko1@heroku.com

Summary of this talk

- Ruby 2.1.0 will be released soon!
  - 2013/12/25
  - Some new features and internal performance improvements
- Rewrite object management to improve performance
  - "gc.c" → 3414 insertions, 1121 deletions
  - Allocation/Deletion trace mechanism
  - Introduce generational mechanism
  - Tuning GC parameters
  - Optimize object allocation path
  - Refactoring (terminology, method path, etc)

Who am I ?

- Koichi Sasada a.k.a. ko1
- From Japan
- 笹田 (family) 耕一 (given) in Kanji character
  - “Ichi” (Kanji character “一”) means “1” or first
  - This naming rule represents I’m the first son of my parents
  - Ko”ichi” → ko1

Who am I ?

- Koichi Sasada a.k.a. ko1
  - Matz team at Heroku, Inc.
    - Full-time CRuby developer
    - Working in Japan
  - CRuby/MRI committer
    - Virtual machine (YARV) from Ruby 1.9
    - YARV development since 2004/1/1
  - Director of Ruby Association

Advertisement

Ruby Association

- Foundation to encourage Ruby developments and communities
  - Chairman is Matz
  - Located at Matsue-city, Japan
- Activities
  - Maintenance of Ruby (Cruby) interpreter
    - Now, it is for Ruby 1.9.3
    - Ruby 2.0.0 in the future?
  - Events, especially RubyWorld Conference
  - Ruby Prize
    - 3 nominates
  - Grant. We have selected **<u>3 proposals</u>** in 2013
    - Win32Utils Support, Conductor, Smalruby - smalruby-editor
  - **<u>Making an appeal for contribution</u>**

Advertisement

heroku

- Heroku, Inc. <http://www.heroku.com>
  - You should know about Heroku!
- Heroku supports Ruby development
  - Many talents for Ruby (and also other languages)
  - Especially Heroku employs three **Ruby interpreter core developers**
    - Matz
    - Nobu
    - Ko1 (me)
  - We name our group "Matz team"

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

Current target is “Ruby 2.1”

Ruby 2.1
Next version

Ruby 2.1 release plan announcement

“I, Naruse, take over the release manager of Ruby 2.1.0 from mame. ***<u>Ruby 2.1.0 is planed to release in 2013-12-25.</u>*** *I‘m planning to call for feature proposals soon like 2.0.0 [ruby-core:45474], so if you have a suggestion you should begin preparing the proposal.”*

- [ruby-core:54726] Announce take over the release manager of Ruby 2.1.0

by NARUSE, Yui

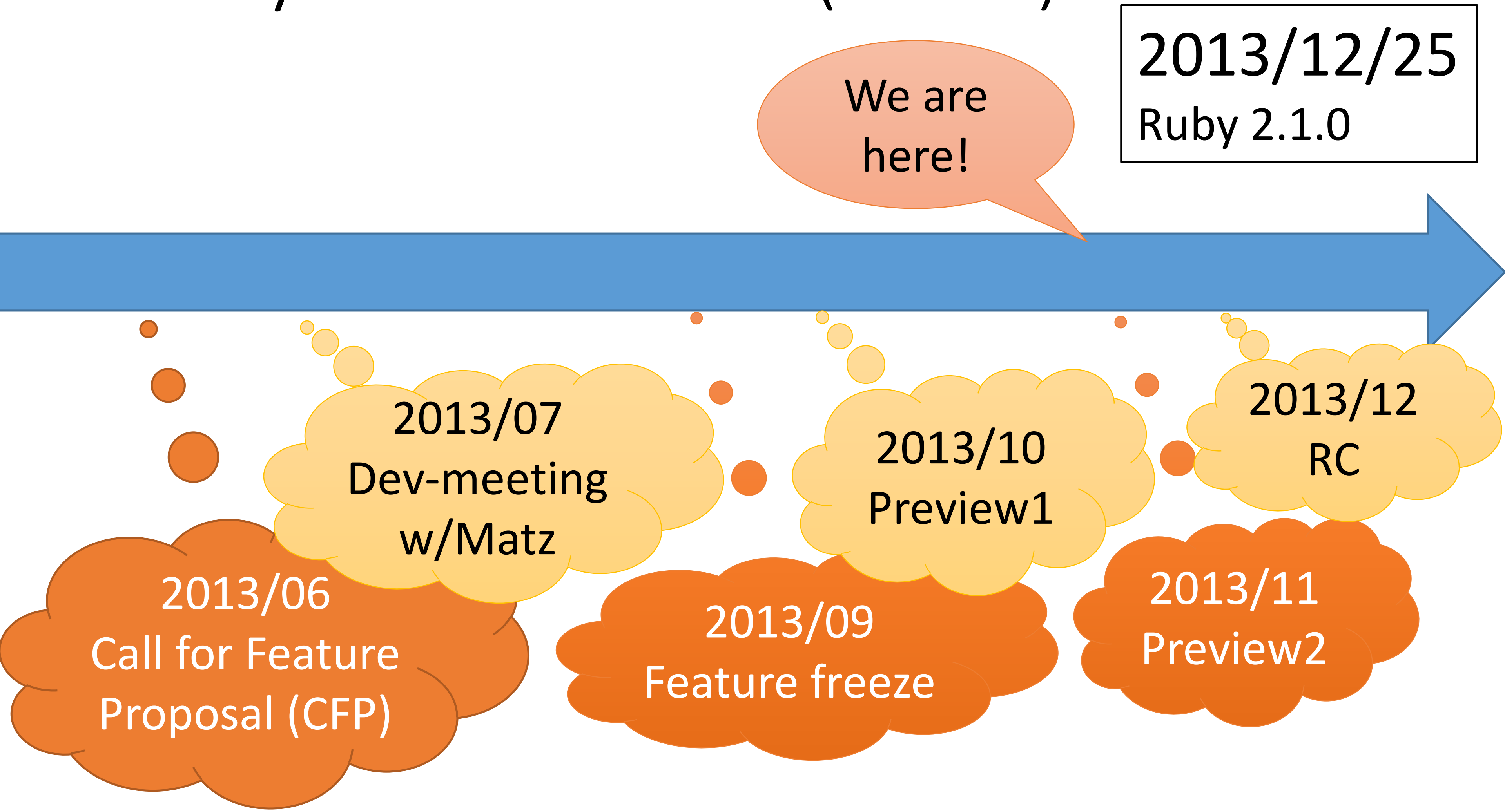
2013/12/25!

http://www.flickr.com/photos/htakashi/5285103341/ by Takashi Hososhima

Ruby 2.1 schedule

Events are important for
EDD (Event Driven Development) Developers

Ruby 2.1 schedule (more)



https://bugs.ruby-lang.org/projects/ruby-trunk/wiki/ReleaseEngineering210

Ruby 2.1
New features

```
# -*- rdoc -*-

= NEWS for Ruby 2.1.0

This document is a list of user visible feature changes made between
releases except for bug fixes.

Note that each entry is kept so brief that no reason behind or
reference information is supplied with.  For a full list of changes
with all sufficient information, see the ChangeLog file.

== Changes since the 2.0.0 release

=== Language changes

* Now the default values of keyword arguments can be omitted.  Those
  "required keyword arguments" need giving explicitly at the call time.

* Added suffixes for integer and float literals: 'r', 'i', and 'ri'.
  * "42r" and "3.14r" are evaluated as Rational(42, 1) and 3.14.rationalize,
    respectively.  But exponential form with 'r' suffix like "6.022e+23r" is
    not accepted because it is misleading.
  * "42i" and "3.14i" are evaluated as Complex(0, 42) and Complex(0, 3.14),
    respectively.
  * "42ri" and "3.14ri" are evaluated as Complex(0, 42r) and Complex(0, 3.14r),
    respectively.

* def-expr now returns the symbol of its name instead of nil.

* Added 'f' suffix for string literals that returns a frozen String object.

=== Core classes updates (outstanding ones only)

* Array
  * New methods
    * Array#to_h converts an array of key-value pairs into a Hash.

* Binding
  * New methods
    * Binding#local_variable_get(symbol)
    * Binding#local_variable_set(symbol, obj)
    * Binding#local_variable_defined?(symbol)

* Enumerable
  * New methods
    * Enumerable#to_h converts a list of key-value pairs into a Hash.

* GC
  * added environment variable:
    * RUBY_HEAP_SLOTS_GROWTH_FACTOR: growth rate of the heap.

* Integer
  * New methods
    * Fixnum#bit_length
    * Bignum#bit_length
  * Bignum performance improvement
    * Use GMP if available.
      GMP is used only for several operations:
      multiplication, division, radix conversion, GCD

* IO
  * extended methods:
    * IO#seek supports SEEK_DATA and SEEK_HOLE as whence.
    * IO#seek accepts symbols (:CUR, :END, :SET, :DATA, :HOLE) for 2nd argument.
    * IO#read_nonblock accepts optional `exception: false` to return symbols
    * IO#write_nonblock accepts optional `exception: false` to return symbols

* Kernel
  * New methods:
    * Kernel#singleton_method

* Module
  * New methods:
    * Module#using, which activates refinements of the specified module only
      in the current class or module definition.
    * Module#singleton_class? returns true if the receiver is a singleton class
      or false if it is an ordinary class or module.
  * extended methods:
    * Module#refine is no longer experimental.
    * Module#include and Module#prepend are now public methods.

* Mutex
  * misc
    * Mutex#owned? is no longer experimental.

* Numeric
  * extended methods:
    * Numeric#step allows the limit argument to be omitted, in which
      case an infinite sequence of numbers is generated.  Keyword
      arguments `to` and `by` are introduced for ease of use.

* Process
  * New methods:
    * alternative methods to $0/$0=:
      * Process.argv0() returns the original value of $0.
      * Process.setproctitle() sets the process title without affecting $0.
    * Process.clock_gettime
    * Process.clock_getres

* RbConfig
  * New constants:
    * RbConfig::SIZEOF is added to provide the size of C types.

* String
  * New methods:
    * String#scrub and String#scrub! verify and fix invalid byte sequence.
  * extended methods:
    * If invalid: :replace is specified for String#encode, replace
      invalid byte sequence even if the destination encoding equals to
      the source encoding.

* Symbol
  * All symbols are now frozen.

* pack/unpack (Array/String)
  * Q! and q! directives for long long type if platform has the type.

* toplevel
  * extended methods:
    * main.using is no longer experimental. The method activates refinements
      in the ancestors of the argument module to support refinement
      inheritance by Module#include.

=== Core classes compatibility issues (excluding feature bug fixes)

* IO
  * incompatible changes:
    * open ignore internal encoding if external encoding is ASCII-8BIT.

* Kernel#eval, Kernel#instance_eval, and Module#module_eval.
  * Copies the scope information of the original environment, which means
    that private, protected, public, and module_function without arguments
    do not affect the environment outside the eval string.
    For example, `class Foo; eval "private"; def foo; end; end' doesn't make
    Foo#foo private.

* Kernel#untrusted?, untrust, and trust
  * These methods are deprecated and their behavior is same as tainted?,
    taint, and untaint, respectively.  If $VERBOSE is true, they show warnings.

* Module#ancestors
  * The ancestors of a singleton class now include singleton classes,
    in particular itself.

* Module#define_method and Object#define_singleton_method
  * Now they return the symbols of the defined methods, not the methods/procs
    themselves.

* Numeric#quo
  * Raises TypeError instead of ArgumentError if the receiver doesn't have
    to_r method.

* Proc
  * Returning from lambda proc now always exits from the Proc, not from the
    method where the lambda is created.  Returing from non-lambda proc exits
    from the method, same as the former behavior.

=== Stdlib updates (outstanding ones only)

* CGI::Util
  * All class methods modulized.

* Digest
  * extended methods:
    * Digest::Class.file takes optional arguments for its constructor

* Matrix
  * Added Vector#cross_product.

* Net::SMTP
  * Added Net::SMTP#rset to implement the RSET command

* Pathname
  * New methods:
    * Pathname#write
    * Pathname#binwrite

* OpenSSL::BN
  * extended methods:
    * OpenSSL::BN.new allows Fixnum/Bignum argument.

* open-uri
  * Support multiple fields with same field name (like Set-Cookie).

* rake
  * Updated to 10.1.0.  Major changes include removal of the class namespace,
    Rake::DSL to hold the rake DSL methods and removal of support for legacy
    rake features.

    For a complete list of changes since rake 0.9.6 see:

    http://rake.rubyforge.org/doc/release_notes/rake-10_1_0_rdoc.html

    http://rake.rubyforge.org/doc/release_notes/rake-10_0_3_rdoc.html

* RDoc
  * Updated to 4.1.0.preview.1.  Major enhancements include a modified default
    template and accessibility enhancements.

    For a list of minor enhancements and bug fixes see:
    https://github.com/rdoc/rdoc/blob/v4.1.0.preview.1/History.rdoc

* Resolv
  * New methods:
    * Resolv::DNS.fetch_resource
  * One-shot multicast DNS support
  * Support LOC resources

* REXML::Parsers::SAX2Parser
  * Fixes wrong number of arguments of entitydecl event. Document of the event
    says "an array of the entity declaration" but implemenation passes two
    or more arguments. It is an implementation bug but it breaks backword
    compatibility.

* REXML::Parsers::StreamParser
  * Supports "entity" event.

* REXML::Text
  * REXML::Text#<< supports method chain like 'text << "XXX" << "YYY"'.
  * REXML::Text#<< supports not "raw" mode.

* Rinda::RingServer, Rinda::RingFinger
  * Rinda now supports multicast sockets.  See Rinda::RingServer and
    Rinda::RingFinger for details.

* RubyGems
  * Updated to 2.2.0.preview.1  For a list of enhancements and bug fixes see:
    https://github.com/rubygems/rubygems/blob/v2.2.0.preview.1/History.txt

* Set
  * New methods:
    * Set#intersect?
    * Set#disjoint?

* Socket
  * New methods:
    * Socket.getifaddrs

* StringScanner
  * extended methods:
    * StringScanner#[] supports named captures.

* Syslog::Logger
  * Added facility.

* Tempfile
  * New methods:
    * Tempfile.create

* Timeout
  * No longer an exception to terminate the given block can be rescued
    inside the block, by default, unless the exception class is given
    explicitly.

* TSort
  * New methods:
    * TSort.tsort
    * TSort.tsort_each
    * TSort.strongly_connected_components
    * TSort.each_strongly_connected_component
    * TSort.each_strongly_connected_component_from

* WEBrick
  * The body of a response may now be a StringIO or other IO-like that responds
    to #readpartial and #read.

* XMLRPC::Client
  * New methods:
    * XMLRPC::Client#http. It returns Net::HTTP for the client. Normally,
      it is not needed.  It is useful when you want to change minor HTTP client
      options. You can change major HTTP client options by XMLRPC::Client
      methods. You should use XMLRPC::Client methods for changing major
      HTTP client options instead of XMLRPC::Client#http.

=== Stdlib compatibility issues (excluding feature bug fixes)

* objspace
  * new method:
    * ObjectSpace.trace_object_allocations
    * ObjectSpace.trace_object_allocations_start
    * ObjectSpace.trace_object_allocations_stop
    * ObjectSpace.trace_object_allocations_clear
    * ObjectSpace.allocation_sourcefile
    * ObjectSpace.allocation_sourceline
    * ObjectSpace.allocation_class_path
    * ObjectSpace.allocation_method_id
    * ObjectSpace.allocation_generation
    * ObjectSpace.reachable_objects_from_root

* Set
  * incompatible changes:
    * Set#to_set now returns self instead of generating a copy.

* URI
  * incompatible changes:
    * URI.decode_www_form follows current WHATWG URL Standard.
      It gets encoding argument to specify the character encoding.
      It now allows loose percent encoded strings, but denies ;-separator.
    * URI.encode_www_form follows current WHATWG URL Standard.
      It gets encoding argument to convert before percent encode.
      UTF-16 strings aren't converted to UTF-8 before percent encode by default.

=== Built-in global variables compatibility issues

* $SAFE
  * $SAFE=4 is obsolete.  If $SAFE is set to 4 or larger, an ArgumentError
    is raised.

=== C API updates
```

See NEWS file

Now, much smaller than Ruby 2.0

Ruby 2.1 new features

- New Numeric syntax (1/2r => Rational(1, 2), etc.)
- “def” returns a symbol of method name
- Refine features introduced from Ruby 2.0
  - Keyword arguments
  - Refinements
  - Module#prepend
- New methods
  - Refine m17n introduced from Ruby 1.9
    - String#scrub, String#scrub!
    - Verify and fix invalid byte sequence.
  - Enumerable#to_h
- Frozen objects
  - All symbols
  - Frozen string is discussed now

Ruby 2.1 Internal improvements

- Profiling supports
  - Additional internal hooks for object allocation and deallocation
  - Profiling API
- More sophisticated garbage collection
  - RGenGC: Introduce generational GC into CRuby
  - GC Parameter tuning
  - Other tuning
- More sophisticated method caching
- Bignum/Integer improvements
- …

Today's topic
Object management

New Profiling support

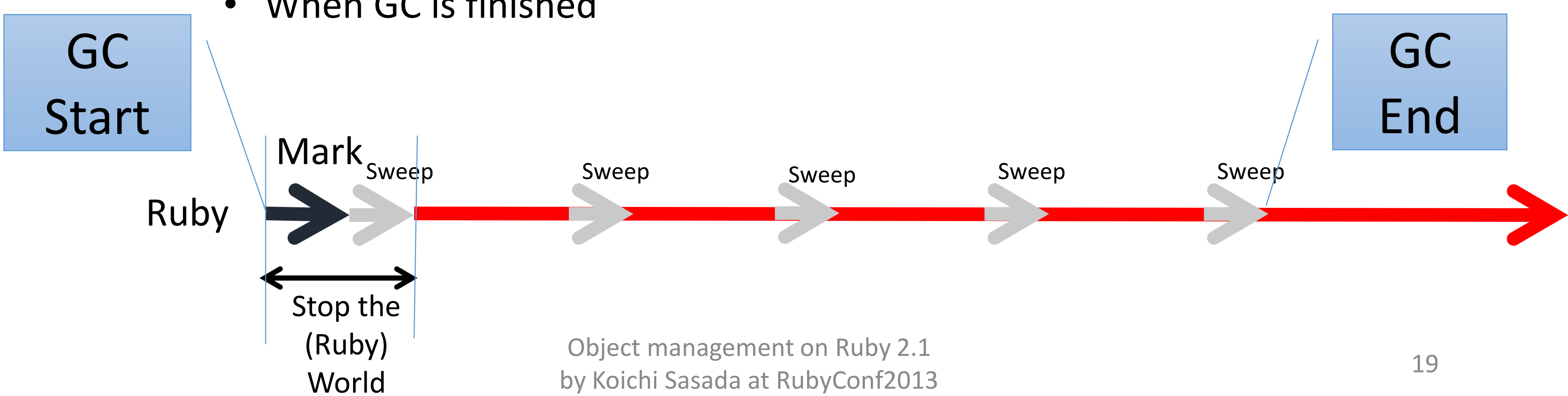
- Internal hooks for object management
- Profiling API to get backtrace information without huge overhead

Internal hooks for object management What's nice?

- You can collect more detailed analysis
- Examples
  - Collect object allocation site information
  - Collect usage of allocated objects
  - Measure GC performance from outside

Internal hooks for object management

- 4 added events
  - RUBY_INTERNAL_EVENT_NEWOBJ
    - When object is created
  - RUBY_INTERNAL_EVENT_FREEOBJ
    - When object is freed
  - RUBY_INTERNAL_EVENT_GC_START
    - When GC is started
  - RUBY_INTERNAL_EVENT_GC_END
    - When GC is finished



Internal hooks for object management *Caution*

- You can *NOT* trace these events using TracePoint (introduced from 2.0)
- You need to write C-ext to use them, because events are invoked during GC, etc
- Use new “postponed job” API

Internal hooks for object management Sample feature using new hooks

- ObjectSpace. trace_object_allocations
  - Trace object allocation and record allocation-site
    - Record filename, line number, creator method’s id and class
  - Usage:

```
ObjectSpace.trace_object_allocations{ # record only in the block
  o = Object.new
  file = ObjectSpace.allocation_sourcefile(o)   #=> __FILE__
  line = ObjectSpace.allocation_sourceline(o) #=> __LINE__ -2
}
```

  - Demonstration

Ruby 2.1

To be more sophisticated

object management

Better Object management

- Refactoring object management code
  - Object management code is in “gc.c” in CRuby
  - I have rewritten (am rewriting) gc.c many parts
  - “gc.c” → 3414 insertions, 1121 deletions
- GC parameter tuning
- New GC algorithm called “**RGenGC**”
  - Generational garbage collection
  - Keep compatibility and performance

GC parameter tuning
Introduce only one case

GC parameter tuning

- When GC occur?
  - (1) There are no slot to allocate object
  - (2) Exceed threshold of memory allocation
- (1) is easy to understand
- Introduce (2) more details

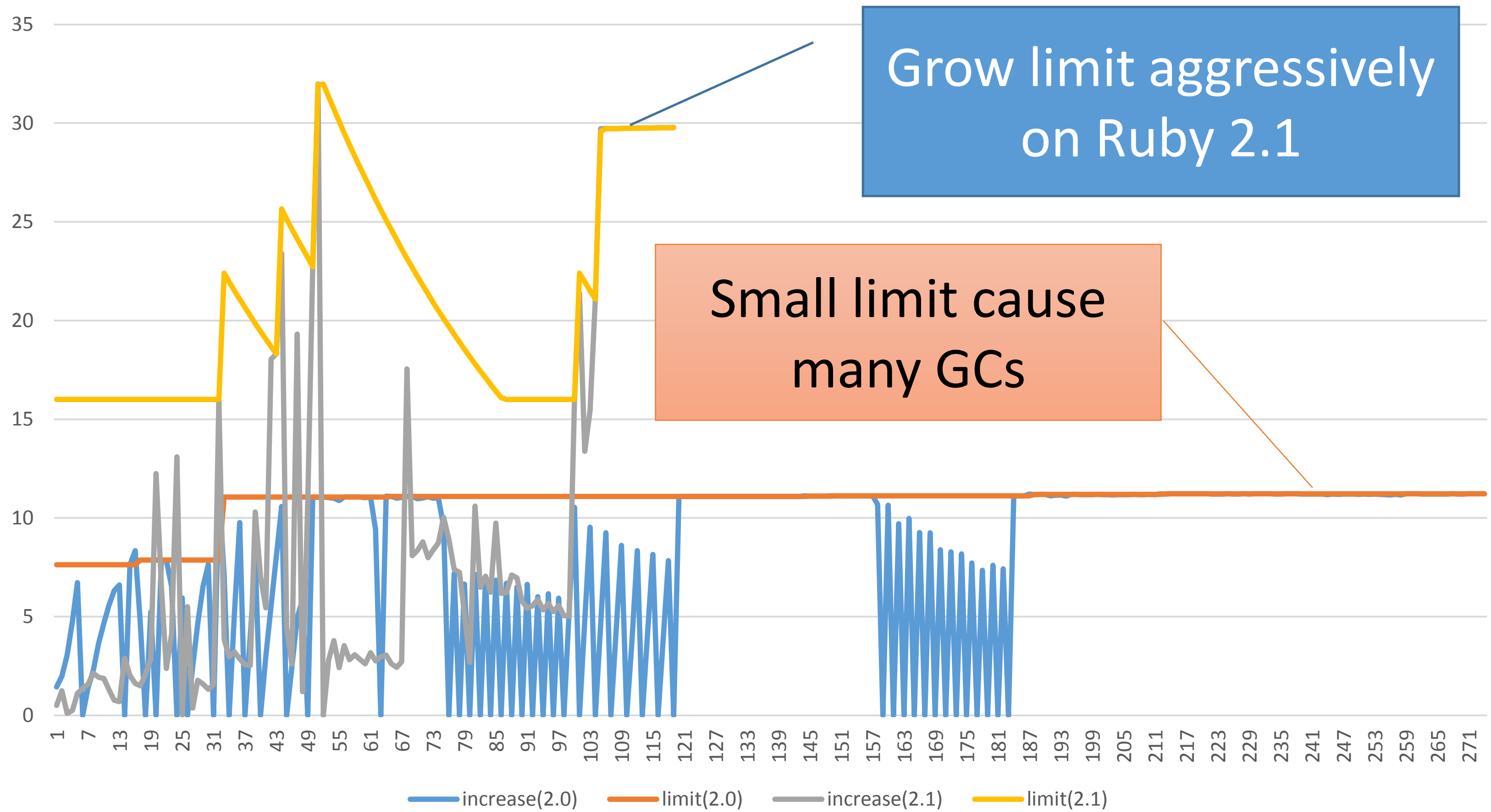
GC parameter tuning malloc_increase and malloc_limit

- Memory allocation → GC
  - Every time allocate “n” size memory (call malloc(n)) increase “malloc_increase” with “n”
  - If malloc_increase > malloc_limit, then cause GC
- Default parameter of “malloc_limit” is “8MB”!!
  - Too small!!
    - String read from 8MB file
    - An array which has 1M entry on 64bit CPU
  - I ask Matz “why such small value?”
  - His reply is “I used 10MB machine at 20 years old”

GC parameter tuning Dynamic tuning of “malloc_limit”

- Default: 8MB → 16MB
- Adaptive tuning
  - If “malloc_increase” exceeds “malloc_limit”, then increase “malloc_limit”
    - Increase “malloc_limit” by a factor of environment variable “GC_MALLOC_LIMIT_GROWTH_FACTOR” (default is 1.4)
    - Maximum value of “malloc_limit” can be set with environment variable “GC_MALLOC_LIMIT_MAX” (default is 32MB)
  - If “malloc_increase” doesn’t exceed limit, then decrease “malloc_limit”

GC parameter tuning
Dynamic tuning of “malloc_limit”



Introduce Generational GC into CRuby

RGenGC: Summary

- RGenGC: Restricted Generational GC
  - 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, **Array, String, Hash, Object, Numeric** objects 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: Previous talk

- Algorithm is introduced at
  - RubyKaigi2013
  - Euruko2013
- See also these slides/movie for details

RGenGC: Agenda

- Background
  - Generational GC
  - Ruby’s GC strategy
- Proposal: RGenGC
  - Separating into normal objects and shady objects
  - Shady objects at marking
  - Shade operation
- Implementation

RGenGC: Background
Current CRuby's GC

- Mark & Sweep
  - Conservative
  - Lazy sweep
  - Bitmap marking
  - Non-recursive marking
- C-friendly strategy
  - Don't need magical macros in C source codes
  - Many many C-extensions under this strategy

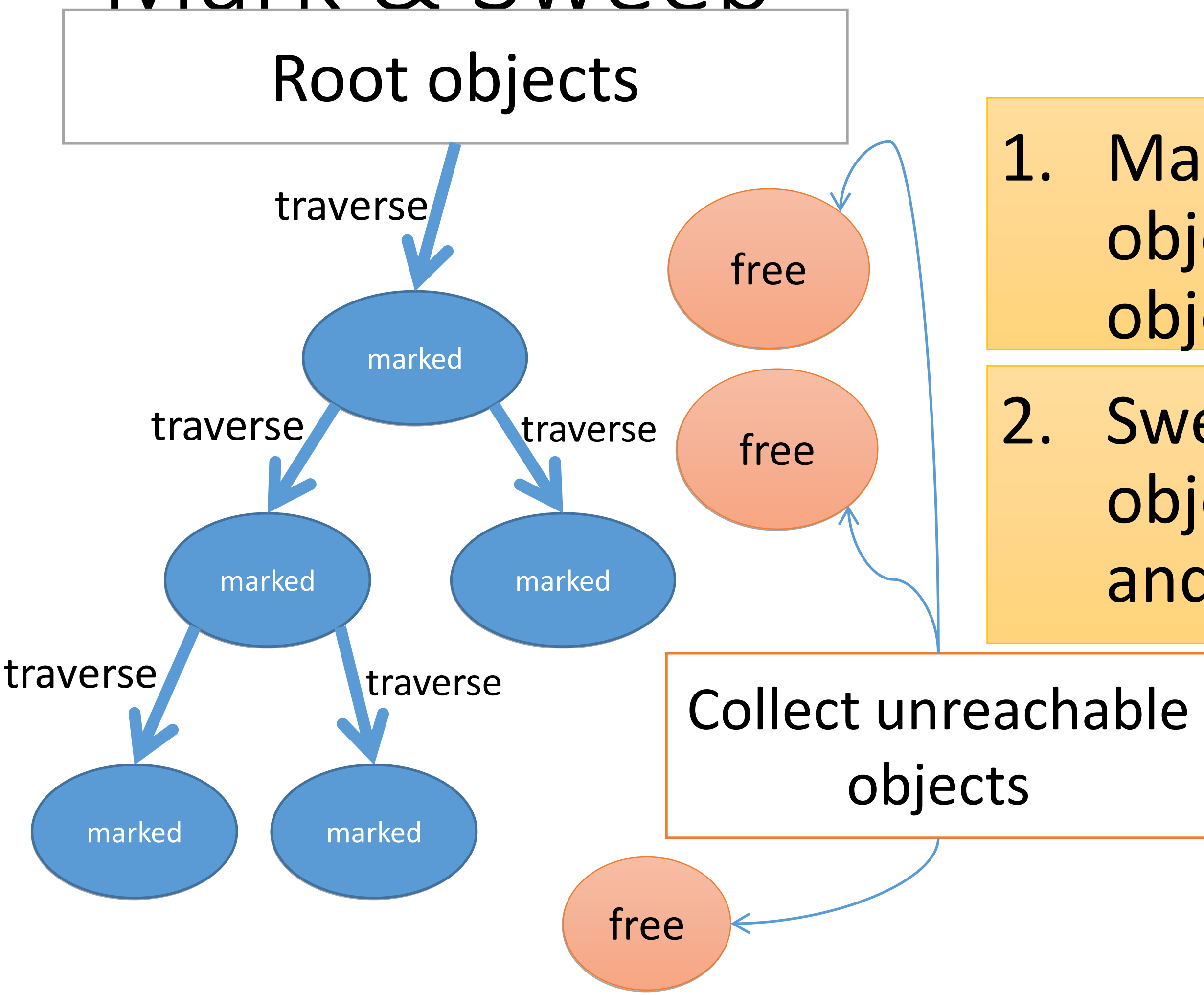
RGenGC
Restriction of CRuby's GC

1. Because of "C-friendly" strategy:
   - We can't know object relation changing timing
   - We can't use "Moving GC algorithm" (such as copying/compacting)

2. Because of "Object data structure":
   - We can't measure exact memory consumption
   - Based on assumption: "malloc" library may be smarter than our hack
     - We rely on "malloc" library for memory allocations
     - GC only manage "object" allocation/deallocation

GC Lecture

RGenGC: Background
Mark & Sweep



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

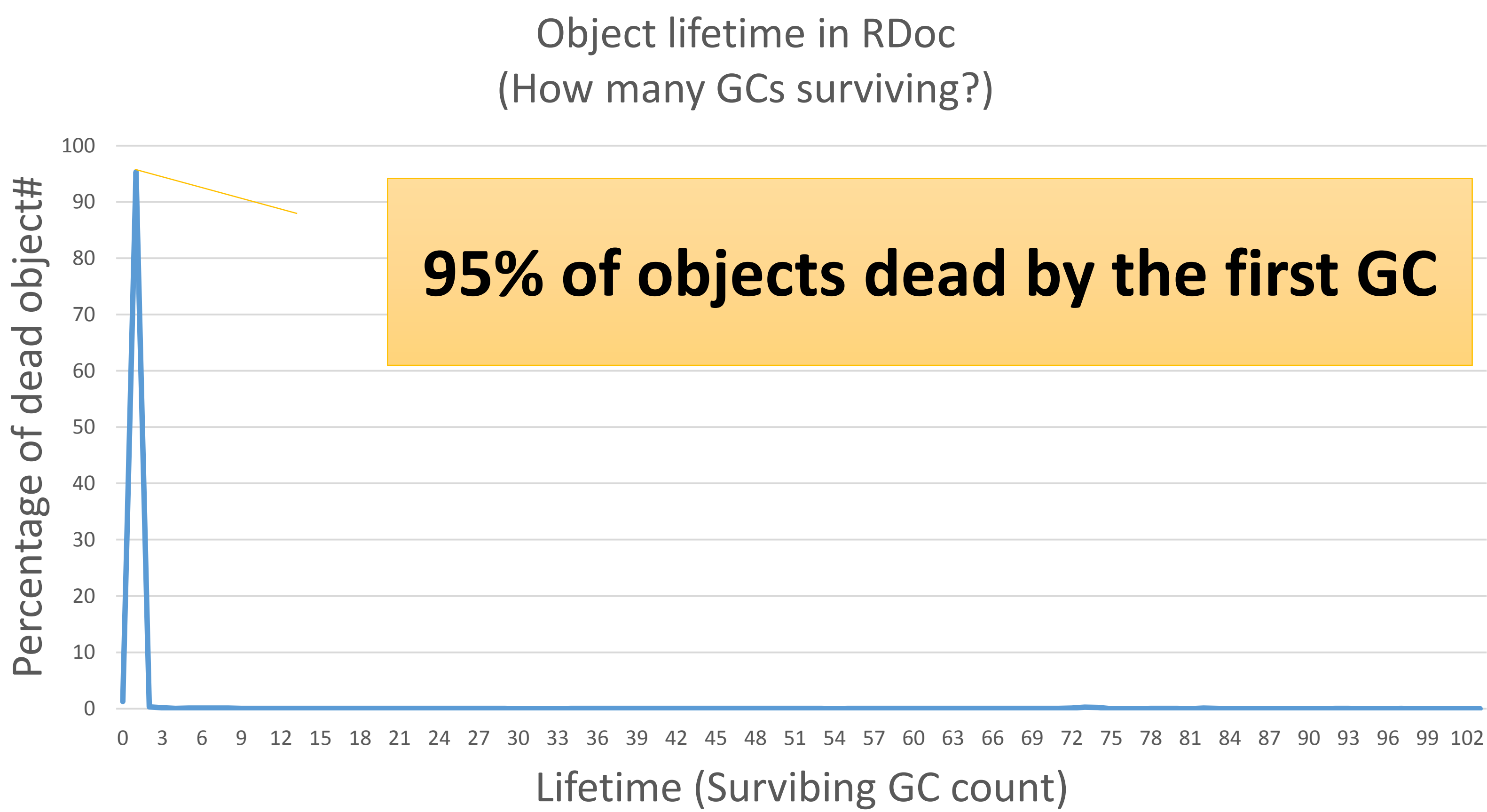
RGenGC: Background
Generational GC (GenGC)

- Weak generational hypothesis:

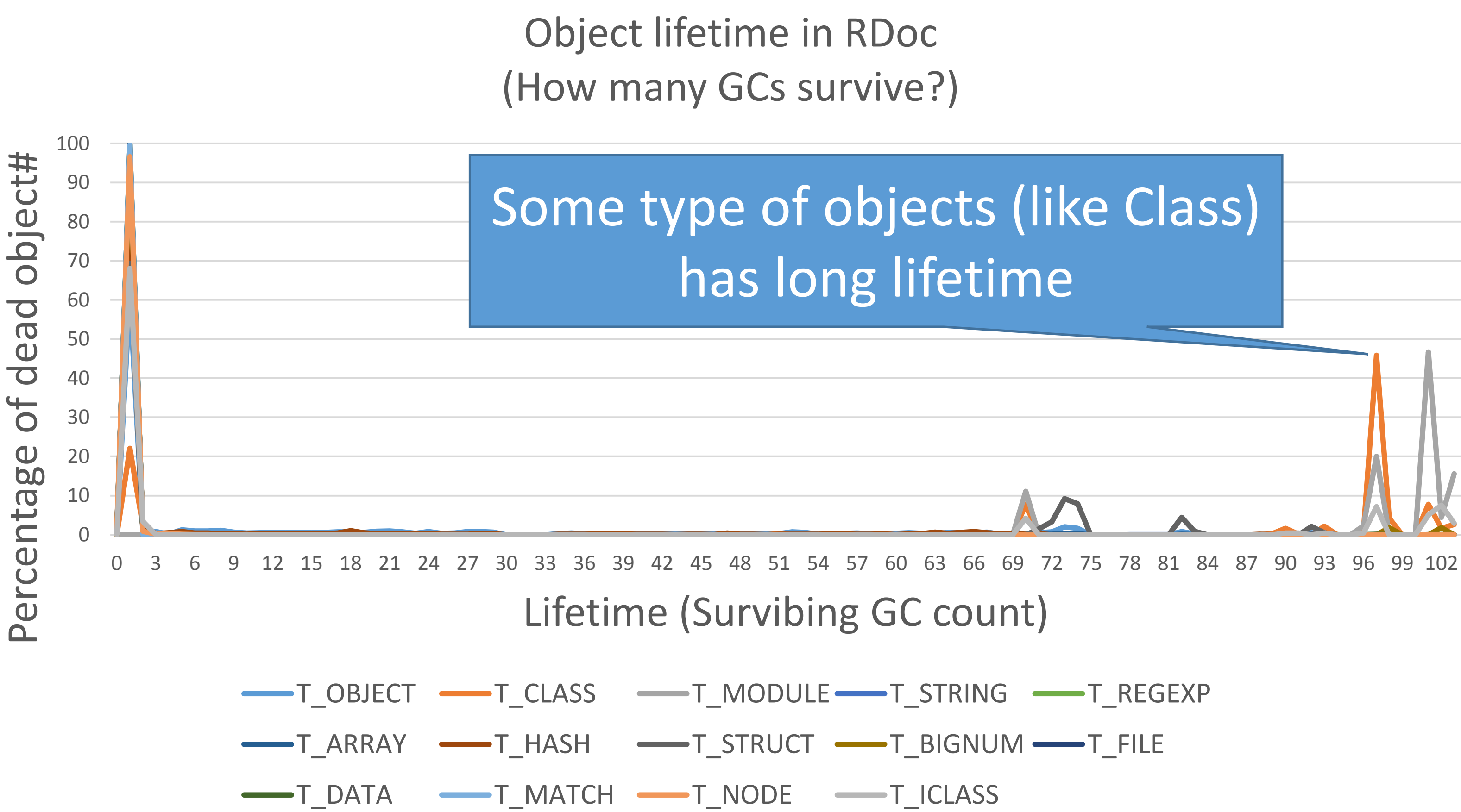
  “Most objects die young”

→ Concentrate reclamation effort only on the young objects

RGenGC: Background Generational hypothesis



RGenGC: Background Generational hypothesis



RGenGC: Background 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)
- Usually, GC on young space (minor GC)
- GC on both spaces if no memory (major/full GC)

RGenGC: Background Generational GC (GenGC)

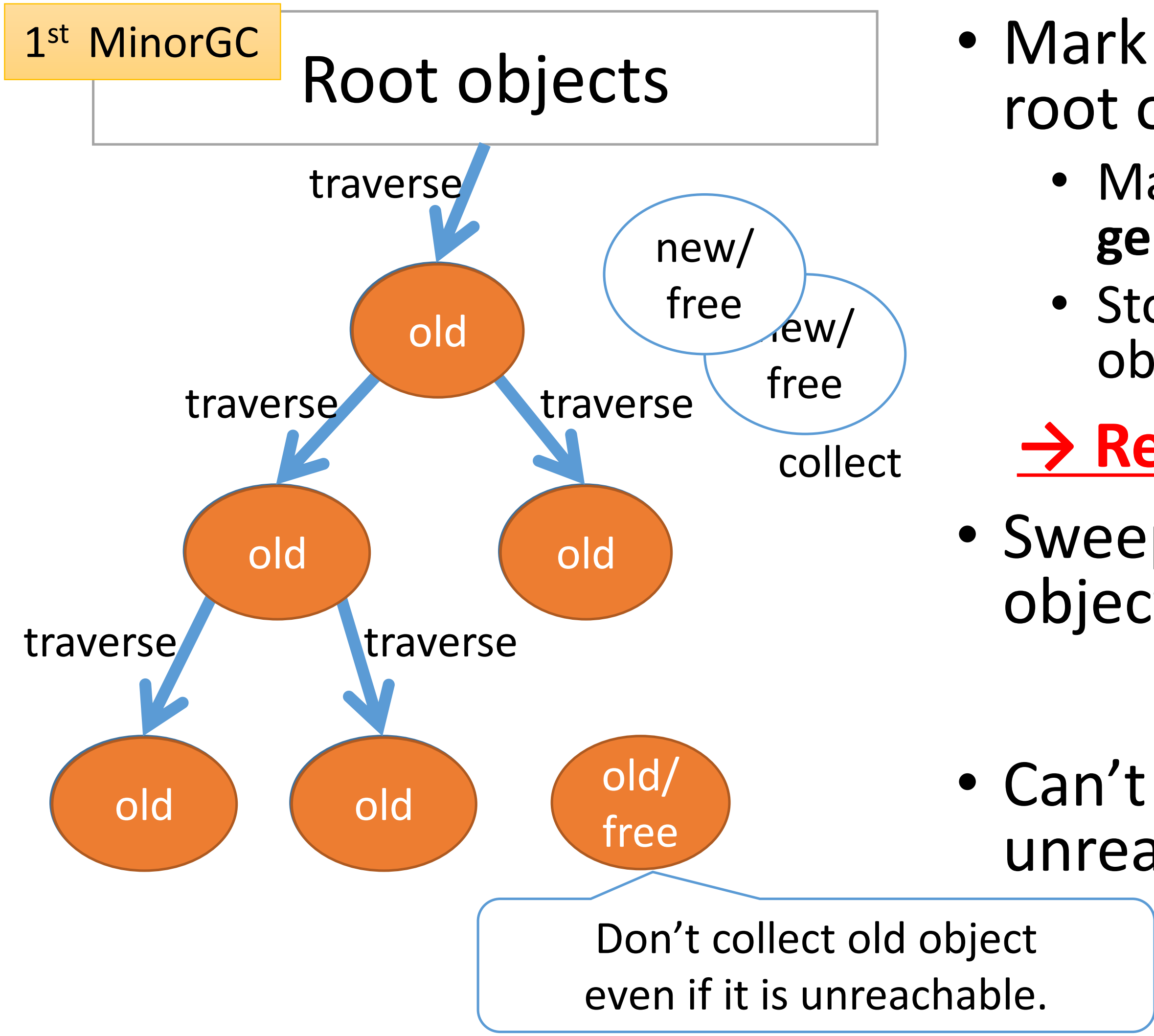
- Minor GC and Major GC can use different GC algorithm
  - Popular combination is:

    Minor GC: Copy GC, Major GC: M&S
  - **On the CRuby, we choose:**

    Minor GC: M&S, Major GC: M&S
  - Because of CRuby's restriction (we can't use moving algorithm)

GC Lecture

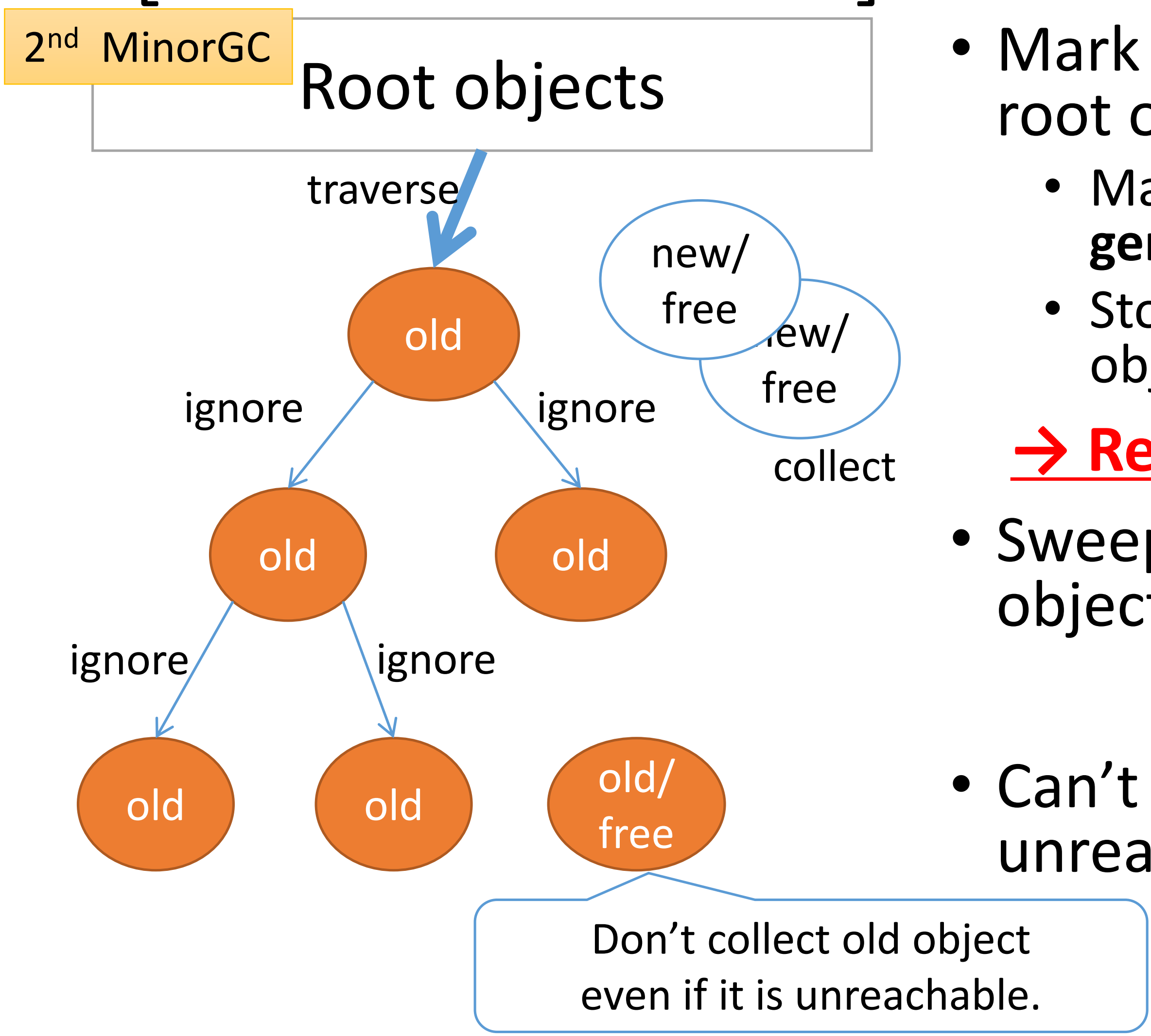
RGenGC: Background: GenGC [Minor M&S GC]



- 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

RGenGC: Background: GenGC [Minor M&S GC]

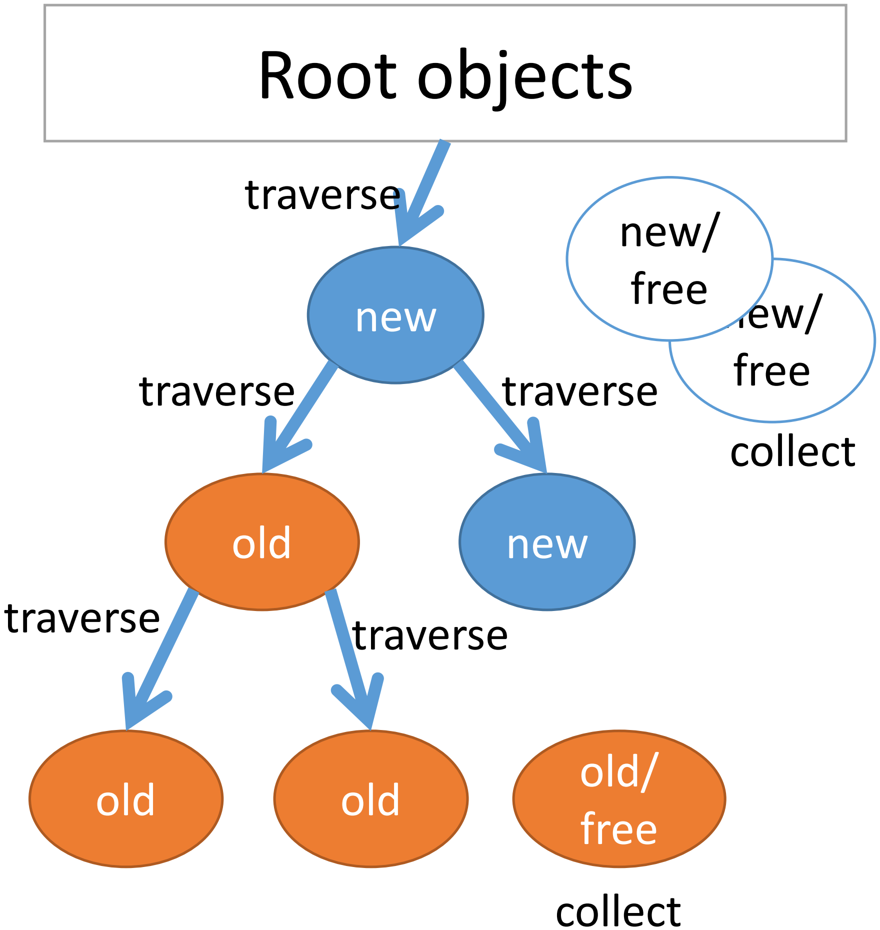


- Mark reachable objects from root objects.
  - Mark and **promote to old generation**
  - Stop traversing after old objects

→ Reduce mark overhead

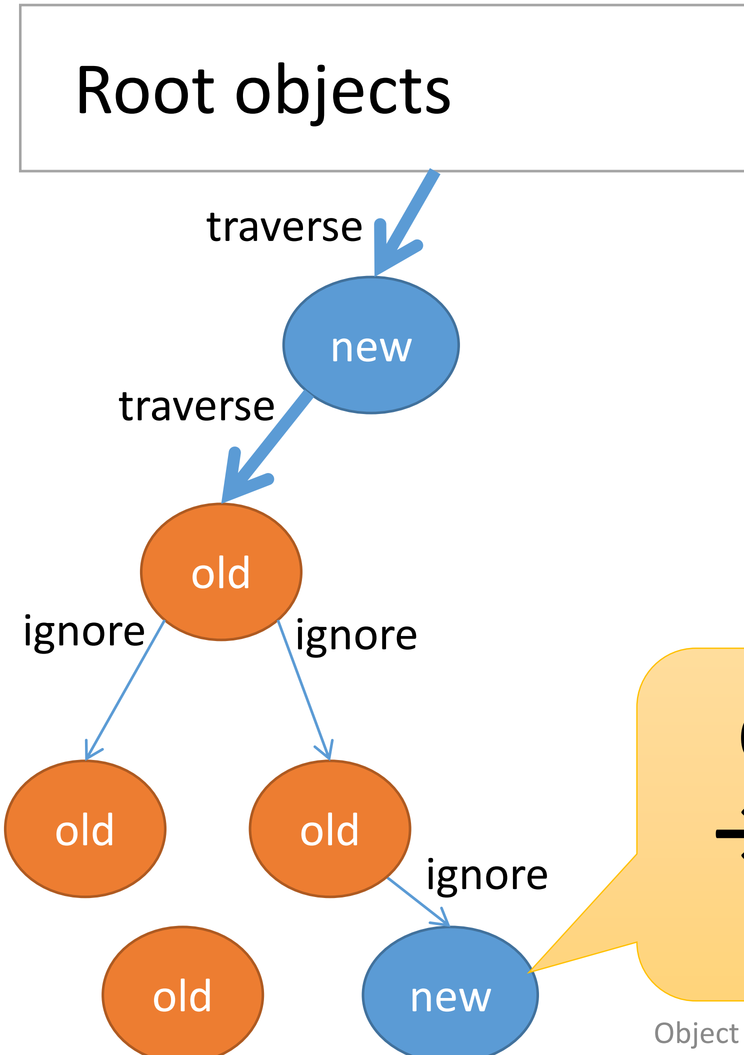
- Sweep not (marked or old) objects
- Can't collect Some unreachable objects

RGenGC: Background: GenGC [Major M&S GC]



- Normal M&S
- Mark reachable objects from root objects
  - Mark and **promote to old gen**
- Sweep unmarked objects
- *<u>Sweep all unreachable (unused) objects</u>*

RGenGC: Background: GenGC Problem: mark miss



- Old objects refer young objects

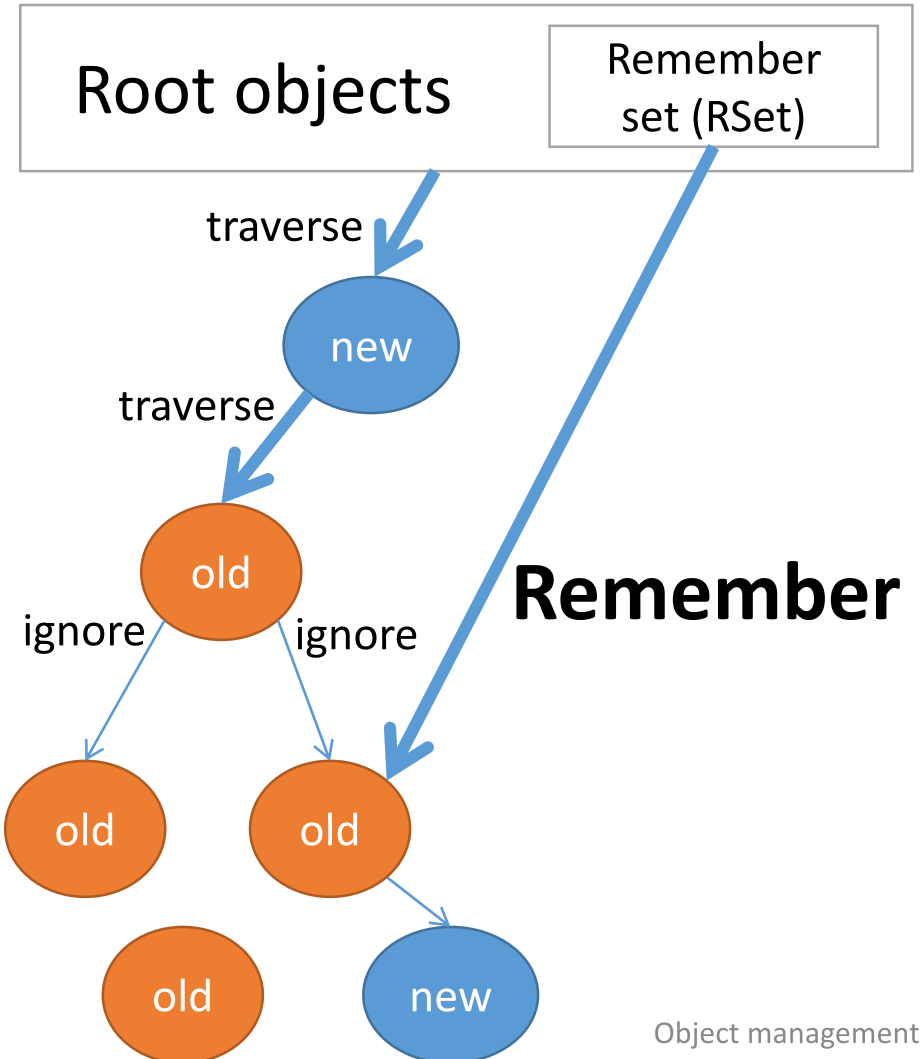
→ Ignore traversal of old object

<u>→ Minor GC causes marking leak!!</u>

- Because minor GC ignores referenced objects by old objects

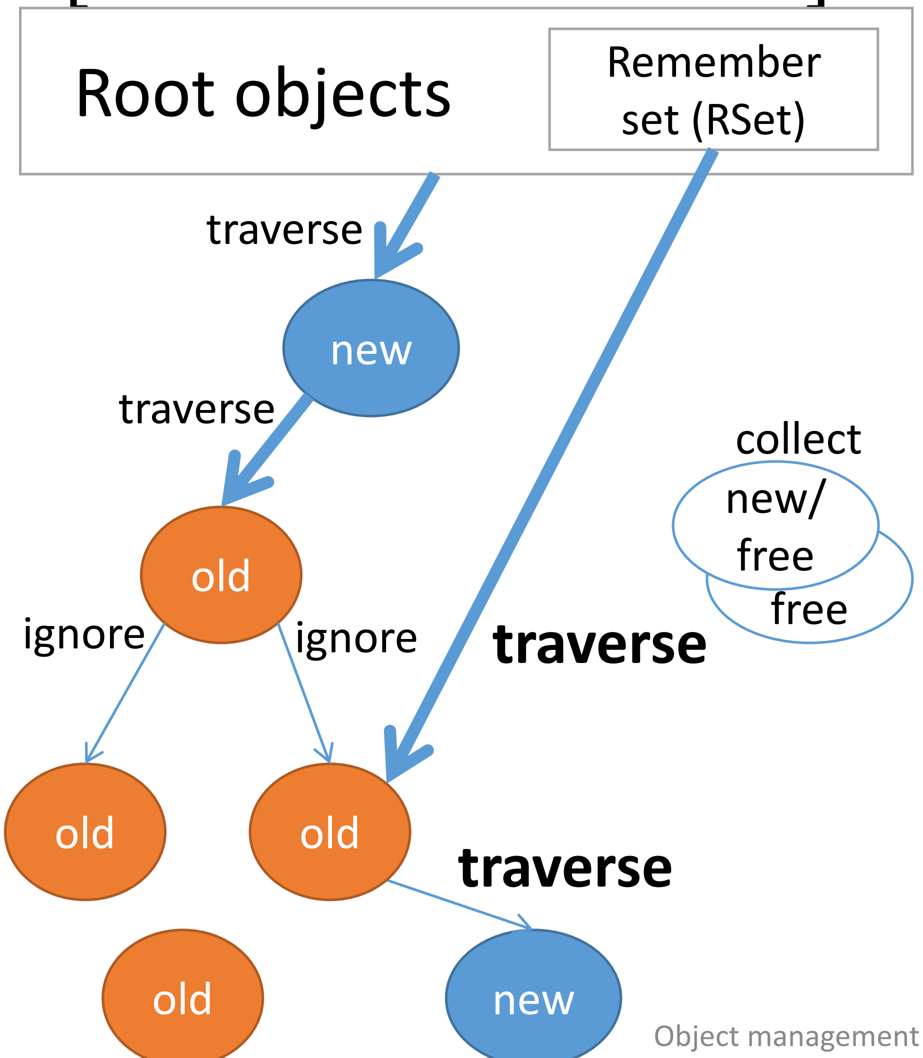
Can't mark new object!
→ Sweeping living object!
(Critical BUG)

RGenGC: Background: GenGC
Introduce Remember set (Rset)



1. **<u>Detect</u>** creation of an [old->new] type reference
2. Add an [old object] into **<u>Remember set (RSet)</u>** if an old object refer new objects

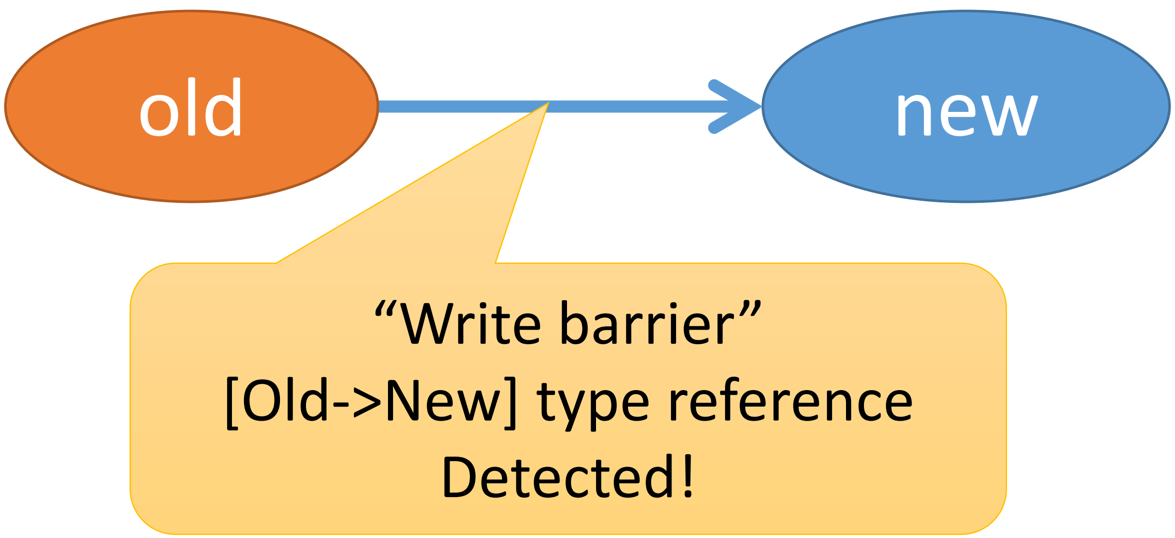
RGenGC: Background: GenGC [Minor M&S GC] w/ RSet



1. Mark reachable objects from root objects
   - <u>**Remembered objects are also root objects**</u>
2. Sweep not (marked or old) objects

RGenGC: Background: GenGC Write barrier

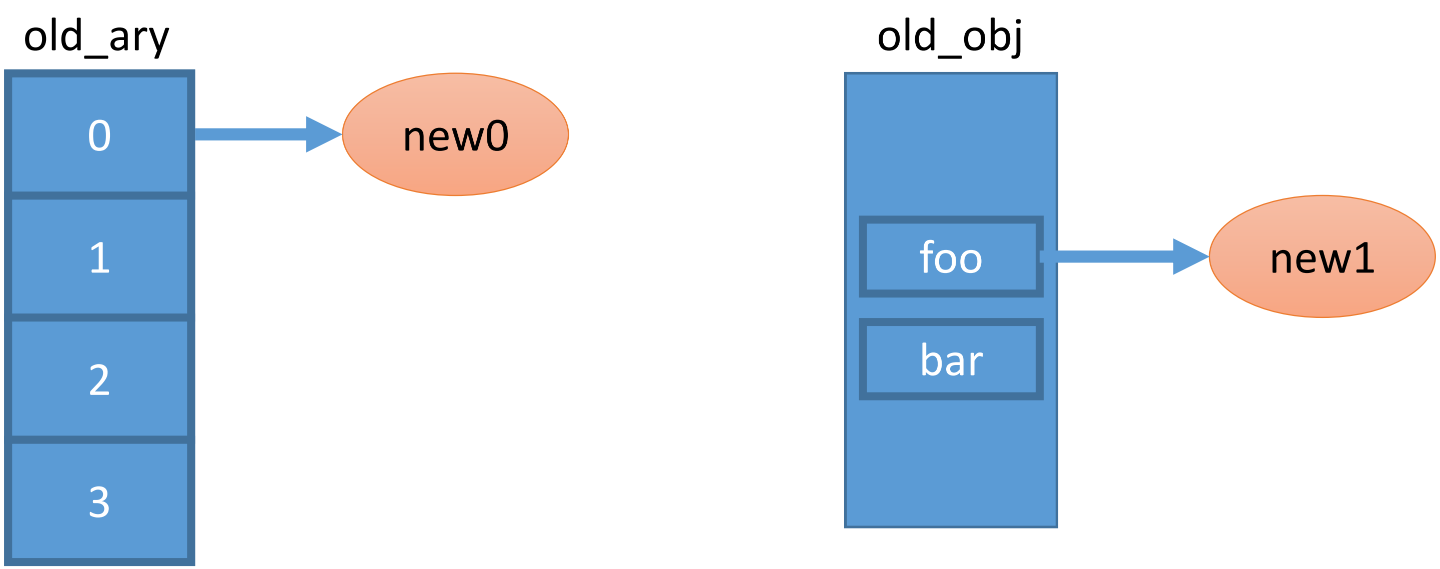
- To detect [old→new] type references, we need to insert <u>**“Write-barrier”**</u> into interpreter for all “Write” operation



RGenGC
Back to Ruby's specific issue

RGenGC: CRuby's case
Write barriers in Ruby

- Write barrier (WB) example in Ruby world
  - (Ruby) old_ary[0] = new0 # [old_ary → new0]
  - (Ruby) old_obj.foo = new1 # [old_obj → new1]



RGenGC: CRuby's case
Difficulty of inserting write barriers

- To introduce generational garbage collector, WBs are necessary to detect [old→new] type reference
- “Write-barrier miss” causes terrible failure
  1. WB miss
  2. Remember-set registration miss
  3. (minor GC) marking-miss
  4. **Collect live object → Terrible GC BUG!!**

RGenGC: Problem
Inserting WBs into C-extensions (C-exts)

- All of C-extensions need perfect Write-barriers
  - C-exts manipulate objects with Ruby's C API
  - C-level WBs are needed
- Problem: How to insert WBs into C-exts?
  - There are many WB required programs in C-exts
    - Example (C): RARRAY_PTR(old0)[0] = new1
    - Ruby C-API doesn't require WB before
  - CRuby interpreter itself also uses C-APIs
- How to deal with?
  - We can rewrite all of source code of CRuby interpreter to add WB, **with huge debugging effort!!**
  - We can't rewrite all of C-exts which are written by 3^{rd} party

RGenGC: Problem
Inserting WBs into C-extensions (C-exts)

Two options

		Performance	Compatibility
1	Give up GenGC	Poor	Good (No problem)
2	GenGC with re-writing all of C exts	Good	Most of C-exts doesn't work

2.0 and earlier conservative choice

Trade-off of Speed and Compatibility

RGenGC: Challenge

- Trade-off of Speed and Compatibility
  - Can we achieve both **speed-up w/ GenGC** and **keeping compatibility?**
- Several possible approaches
  - Separate heaps into the WB world and non-WB world
    - Need to re-write whole of Ruby interpreter
    - Need huge development effort
  - WB auto-insertion
    - Modify C-compiler
    - Need huge development effort

RGenGC: Our approach

- Create **new generational GC algorithm** permits WB protected objects **AND** WB un-protected object in the same heap

RGenGC: Restricted Generational Garbage Collection

RGenGC: Invent 3rd option

		Performance	Compatibility
1	Give up GenGC	Poor	Good (No problem)
2	GenGC with re-writing all of C codes	Good	Most of C-exts doesn't work
3	Use new RGenGC	Good	Most of C-exts works!!

Ruby 2.1 choice

Breaking the trade off. You can praise us!!

RGenGC: Key idea

- Introduce **Shady object**
  - I use the word “Shady” as questionable, doubtful, …
  - Something feeling dark
  - 日陰者, in Japanese

RGenGC:
Key Idea

- Separate objects into two types
    - Normal Object: WB Protected
    - Shady Object: WB Unprotected

Shady: doubtful, questionable, ...

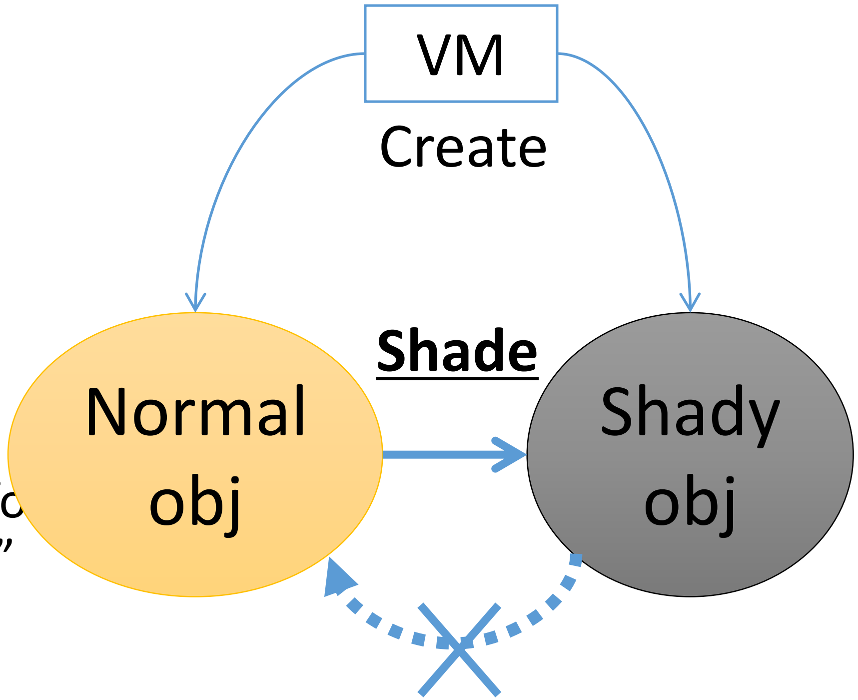
Normal
＼(^o^)／

Shady
(´･ω･`)

- We are not sure that a shady object points new objects or not
- Decide this type at creation time
    - A class care about WB → Normal object
    - A class don’t care about WB → Shady object

RGenGC:
Key Idea

- Normal objects can be changed to Shady objects
  - "Shade operation"
  - C-exts don't care about WB, objects will be shady objects
  - Example
    - ptr = RARRAY_PTR(ary)
    - In this case, we can't insert WB fo ptr operation, so VM shade "ary"

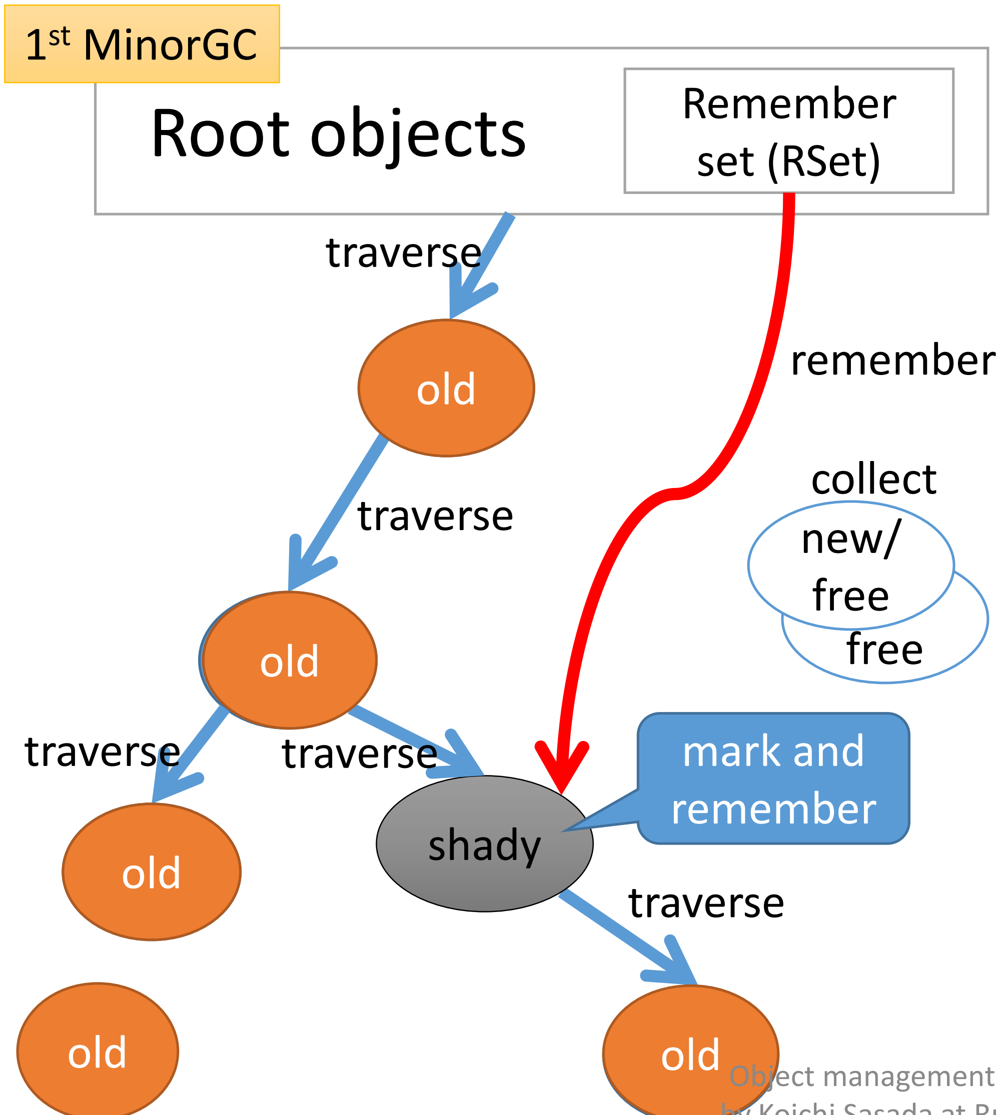


Now, Shady object **can't** change into normal object

RGenGC
Key Idea: Rule

- Treat “Shady objects” correctly
  - At Marking
  1. Don’t promote shady objects to old objects
  2. Remember shady objects pointed from old objects
  - At Shade operation for old normal objects
  1. Demote objects
  2. Remember shaded shady objects

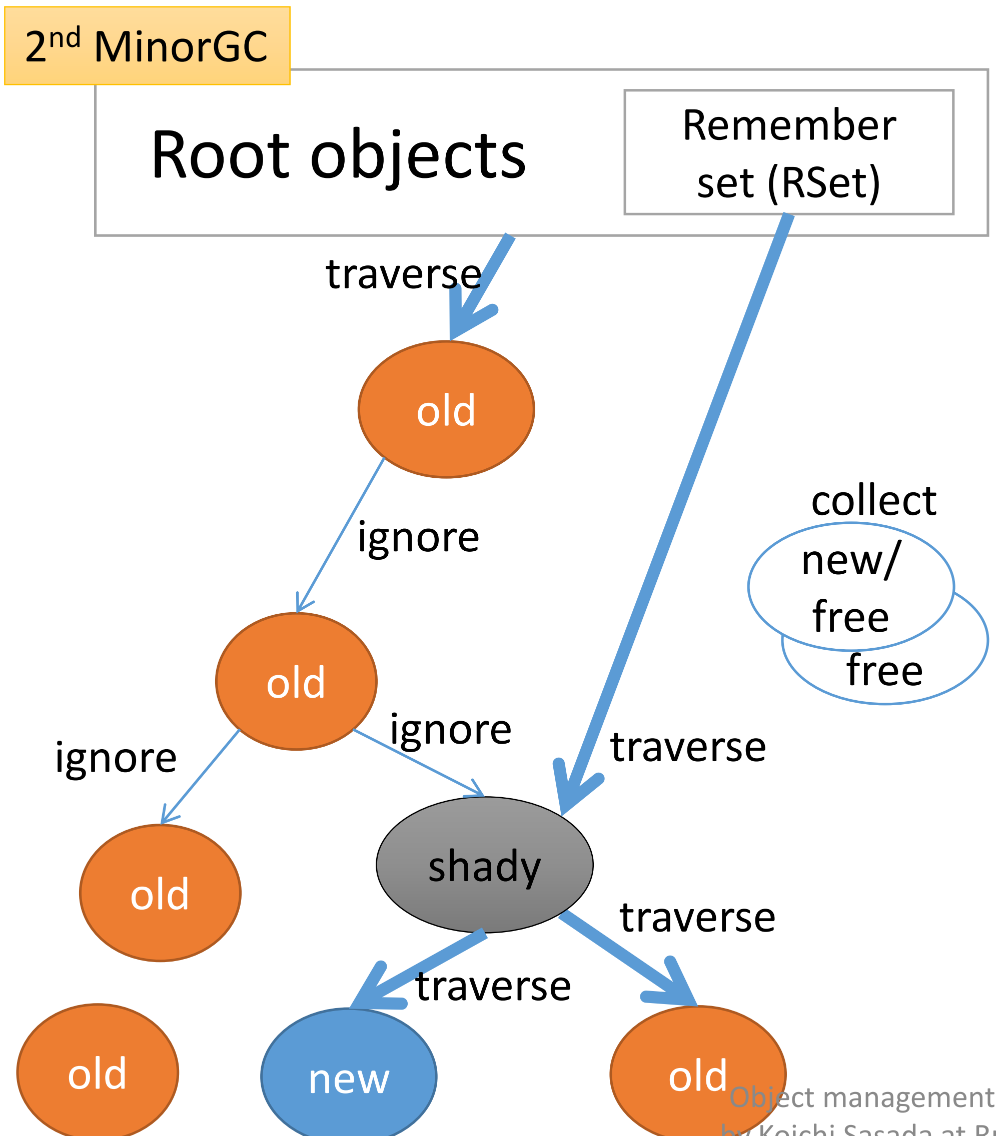
RGenGC [Minor M&S GC w/Shady object]



- Mark reachable objects from root objects
  - Mark shady objects, and ***don't promote*** to old gen objects
  - If shady objects **pointed from old objects**, then **remember shady objects** by RSet.

    → Mark shady objects every minor GC!!

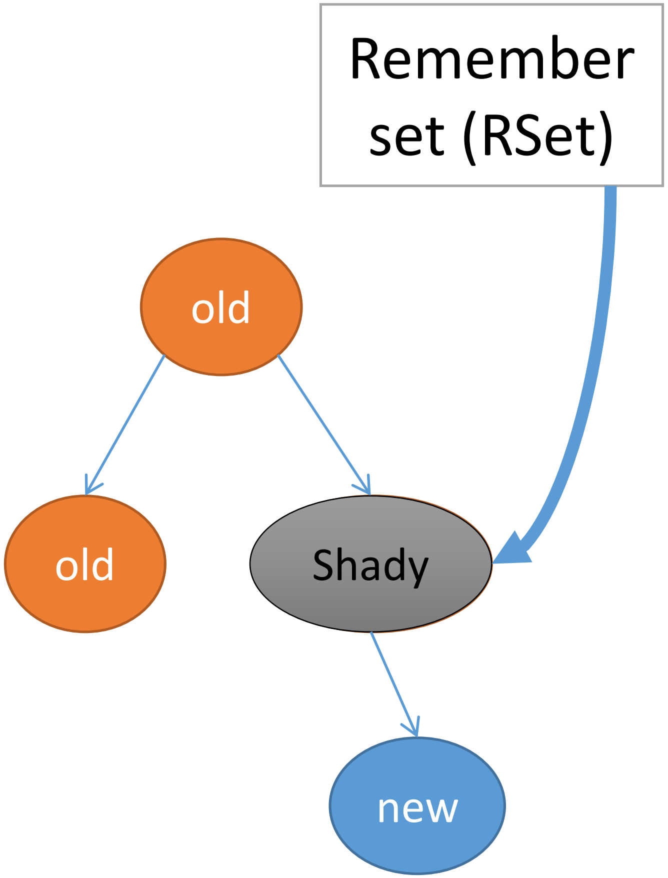
RGenGC [Minor M&S GC w/Shady object]



- Mark reachable objects from root objects
  - Mark shady objects, and **<u>*don't promote*</u>** to old gen objects
  - If shady objects pointed from old objects, then remember shady objects by RSet.

→ Mark shady objects every minor GC!!

RGenGC [Shade operation]



- Anytime Object can give up to keep write barriers

  → [Shade operation]

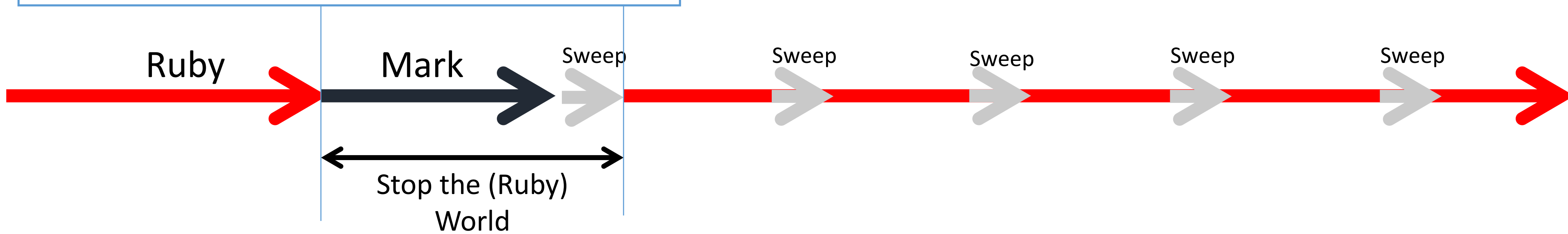
- Change old normal objects to shade objects
  - Example: RARRAY_PTR(ary)

  (1) Demote object (old → new)

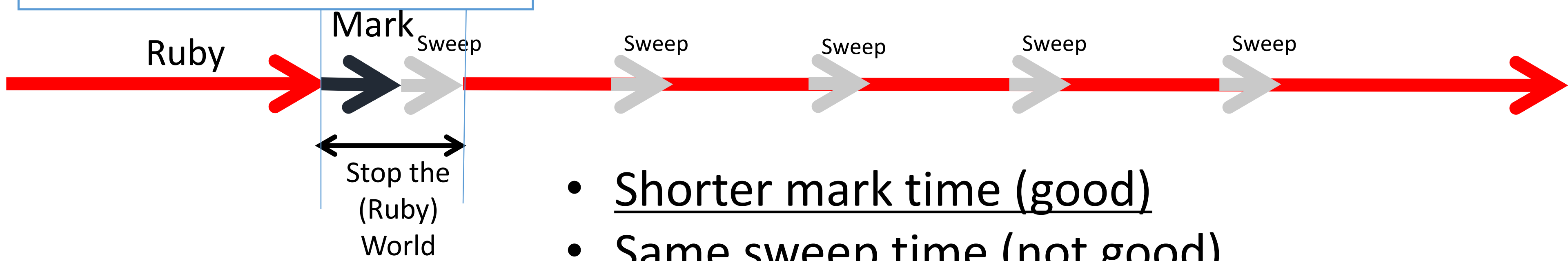
  (2) Register it to Remember Set

RGenGC
Timing chart

2.0.0 GC (M&S w/lazy sweep)

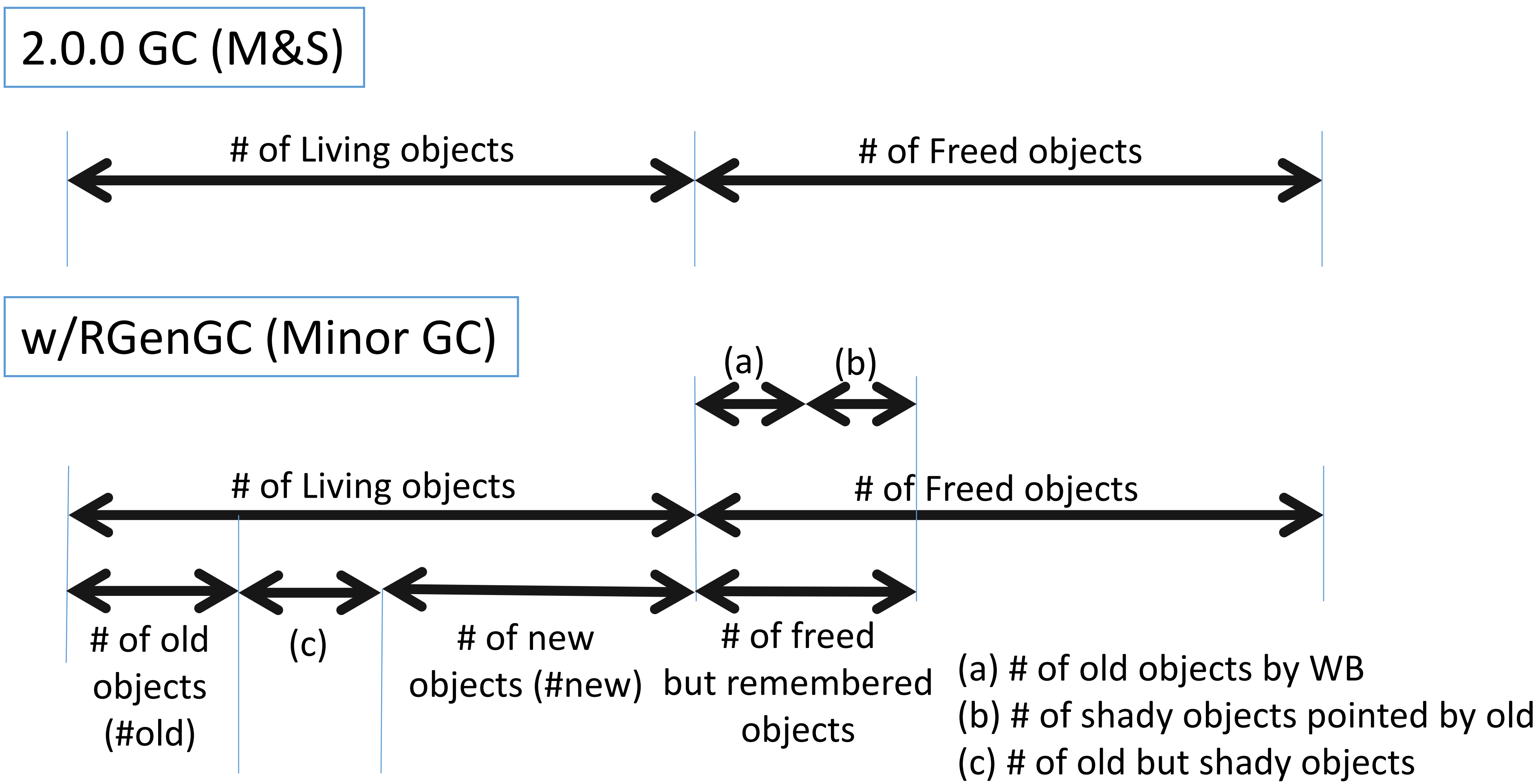


w/RGenGC (Minor GC)



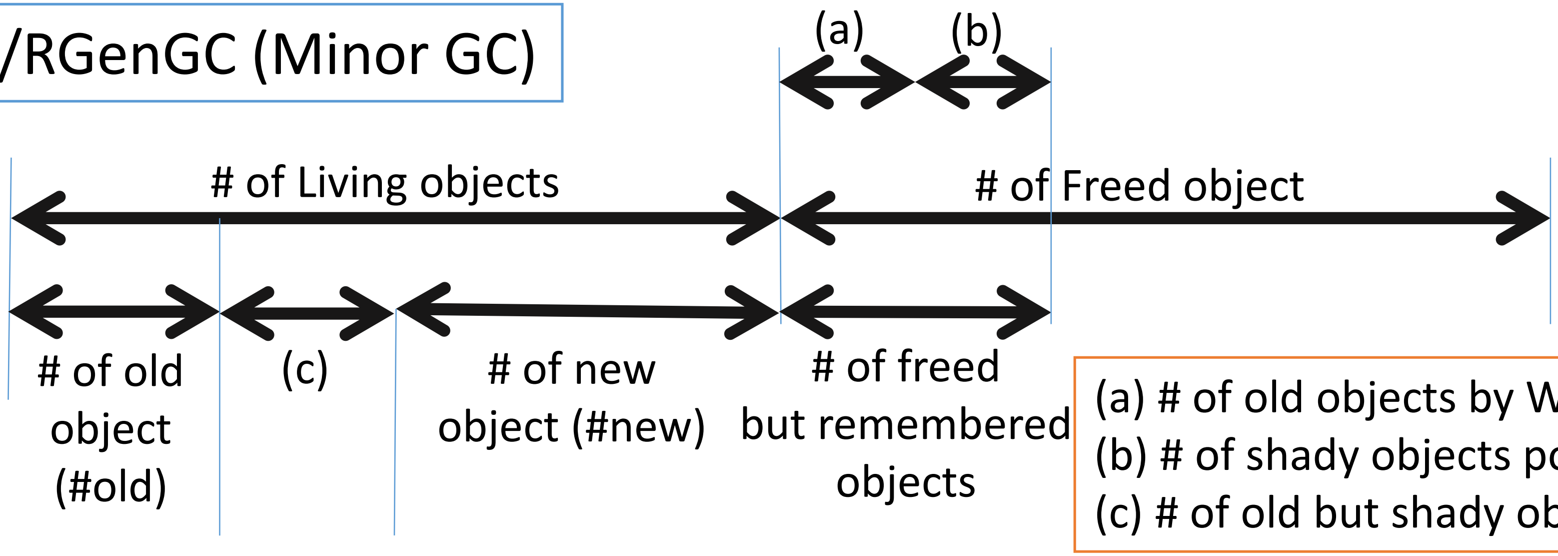
- <u>Shorter mark time (good)</u>
- <u>Same sweep time (not good)</u>
- (little) Longer execution time b/c WB (bad)

RGenGC Number of objects



RGenGC
Number of objects

w/RGenGC (Minor GC)



(a) # of old objects by WB
(b) # of shady objects pointed by old
(c) # of old but shady objects

	Marking space	Number of unused, uncollected objs	Sweeping space
Mark&Swep GC	# of Living objects	0	Full heap
Traditional GenGC	#new + (a)	(a)	#new
RGenGC	#new + (a) + (b) + (c)	(a) + (b)	Full heap

RGenGC Discussion: Pros. and Cons.

- Pros.
  - Allow WB unprotected objects (shady objects)
    - **100% compatible** w/ existing extensions which don't care about WB
    - A part of CRuby interpreter which doesn't care about WB
  - **Inserting WBs step by step, and increase performance gradually**
    - We don't need to insert all WBs into interpreter core at a time
    - We can concentrate into popular (effective) classes/methods.
    - We can ignore minor classes/methods.
  - Simple algorithm, easy to develop (already done!)

RGenGC Discussion: Pros. and Cons.

- Cons.
  - Increasing “unused, but not collected objects until full/major GC
    - Remembered normal objects (caused by traditional GenGC algorithm)
    - Remembered shady objects (caused by RGenGC algorithm)
  - WB insertion bugs (GC development issue)
    - RGenGC permit shady objects, but sunny objects need correct/perfect WBs. But inserting correct/perfect WBs is difficult.
    - This issue is out of scope. We have another idea against this problem (out of scope).
  - Can’t reduce Sweeping time
    - But many (and easy) well-known techniques to reduce sweeping time (out of scope).
  - Increase complexity
    - Additional tuning parameters

RGenGC Implementation: WB support status

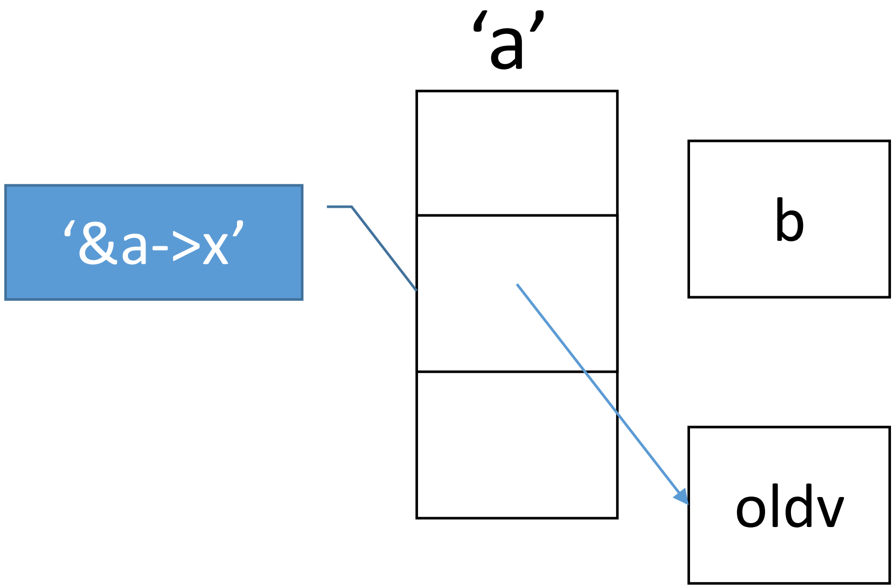
Type name	Status	Comment
T_OBJECT	Supported	
T_CLASS	Supported	Possible to change into shady
T_ICLASS	Supported	Possible to change into shady
T_MODULE	Supported	Possible to change into shady
T_FLOAT	Supported	
T_STRING	Supported	
T_REGEXP	Supported	
T_ARRAY	Supported	Possible to change into shady / more efforts are needed
T_HASH	Supported	Possible to change into shady
T_STRUCT	Supported	
T_BIGNUM	Supported	
T_FILE	Unsupported	Not yet
T_DATA	Supported	Only InstructionSequence objects are supported
T_MATCH	Unsupported	Most of MatchData objects are short-lived
T_RATIONAL	Supported	
T_COMPLEX	Supported	
T_NODE	Unsupported	Most of Node objects are short-lived

RGenGC Implementation

- Introduce two flags into RBasic
  - FL_KEEP_WB: WB protected or not protected
    - 0 → unprotected → Shady object
    - 1 → protected → Sunny object
    - Usage: NEWOBJ_OF(ary, struct RArray, klass, T_ARRAY | **FL_KEEP_WB**);
  - FL_PROMOTED: Promoted or not
    - 0 → Young gen
    - 1 → Old gen
    - Don't need to touch by user program
- Remember set is represented by bitmaps
  - Same as marking bitmap
  - heap_slot::rememberset_bits
  - Traverse all object area with this bitmap at first

RGenGC
Implementation: WB operation API

- OBJ_WRITE(a, &a->x, b)
  - Declare ‘a’ aggregates ‘b’
  - **Write:** *&a->x = b
  - Write barrier
  - OBJ_WRITE(a, b) returns “a”



- OBJ_WRITTEN(a, oldv, b)
  - Declare ‘a’ aggregates ‘b’ and old value is ‘oldv’
  - Non-write operation
  - Write barrier

RGenGC Implementation: WB operation API

- T_ARRAY
  - **<u>RARRAY_PTR(ary) causes shade operation</u>**
    - Can't get RGenGC performance improvement
    - But works well ☺ (Do not need to modify codes)

Important!!

- Instead of RARRAY_PTR(ary), use alternatives
  - **RARRAY_AREF(ary, n)** → RARRAY_PTR(ary)[n]
  - **RARRAY_ASET(ary, n, obj)** → RARRAY_PTR(ary)[n] = obj w/ Write-barrier
  - **RARRAY_PTR_USE(ary, ptrname, {...block...})**
    - Only in block, pointers can be accessed by `ptrname' variable (VALUE*).
    - **<u>Programmers need to insert collect WBs (miss causes BUG)</u>**.

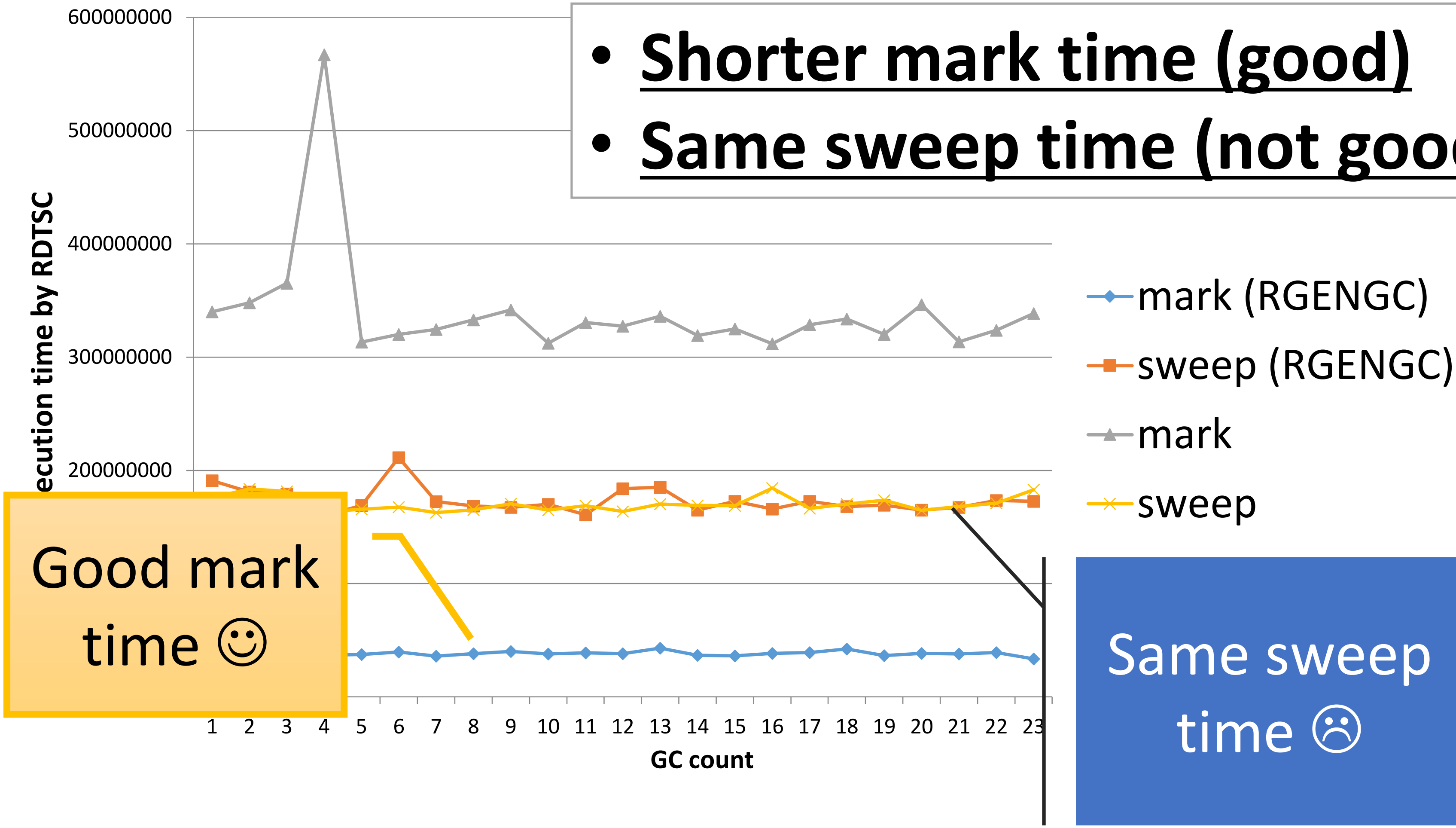
RGenGC Incompatibility

- Make RBasic::klass “const”
  - Need WBs for a reference from an object to a klass.
  - Only few cases (zero-clear and restore it)
  - Provide alternative APIs
    - Now, RBASIC_SET_CLASS(obj, klass) and RBASIC_CLEAR_CLASS(obj) is added. But they should be internal APIs (removed soon).
    - rb_obj_hide() and rb_obj_reveal() is provided.

RGenGC Performance evaluation

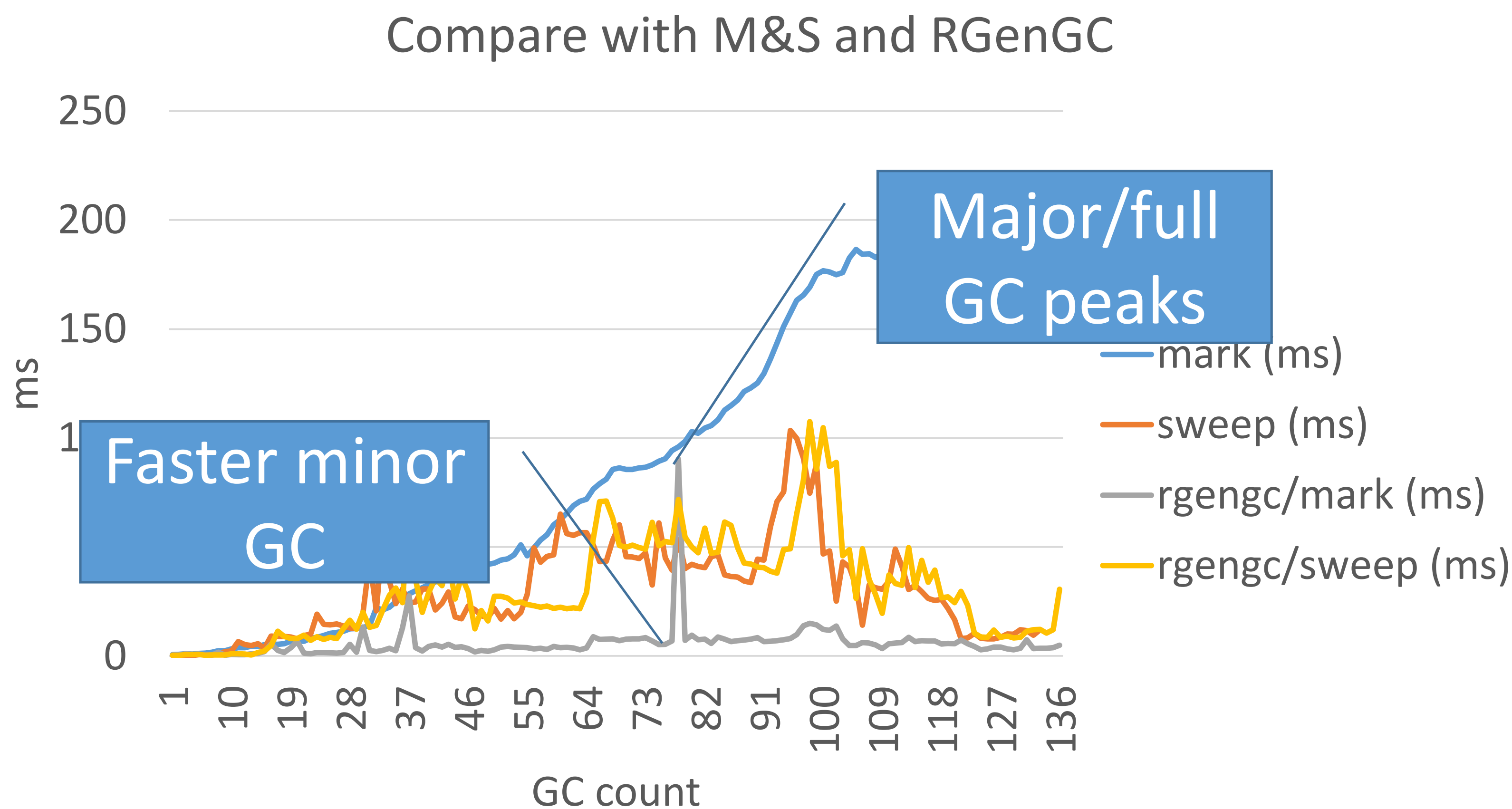
- Ideal micro-benchmark for RGenGC
  - Create many old objects at first
  - Many new objects (many minor GC, no major GC)
- RDoc
  - Same “make doc” task from trunk

RGenGC Performance evaluation (micro)



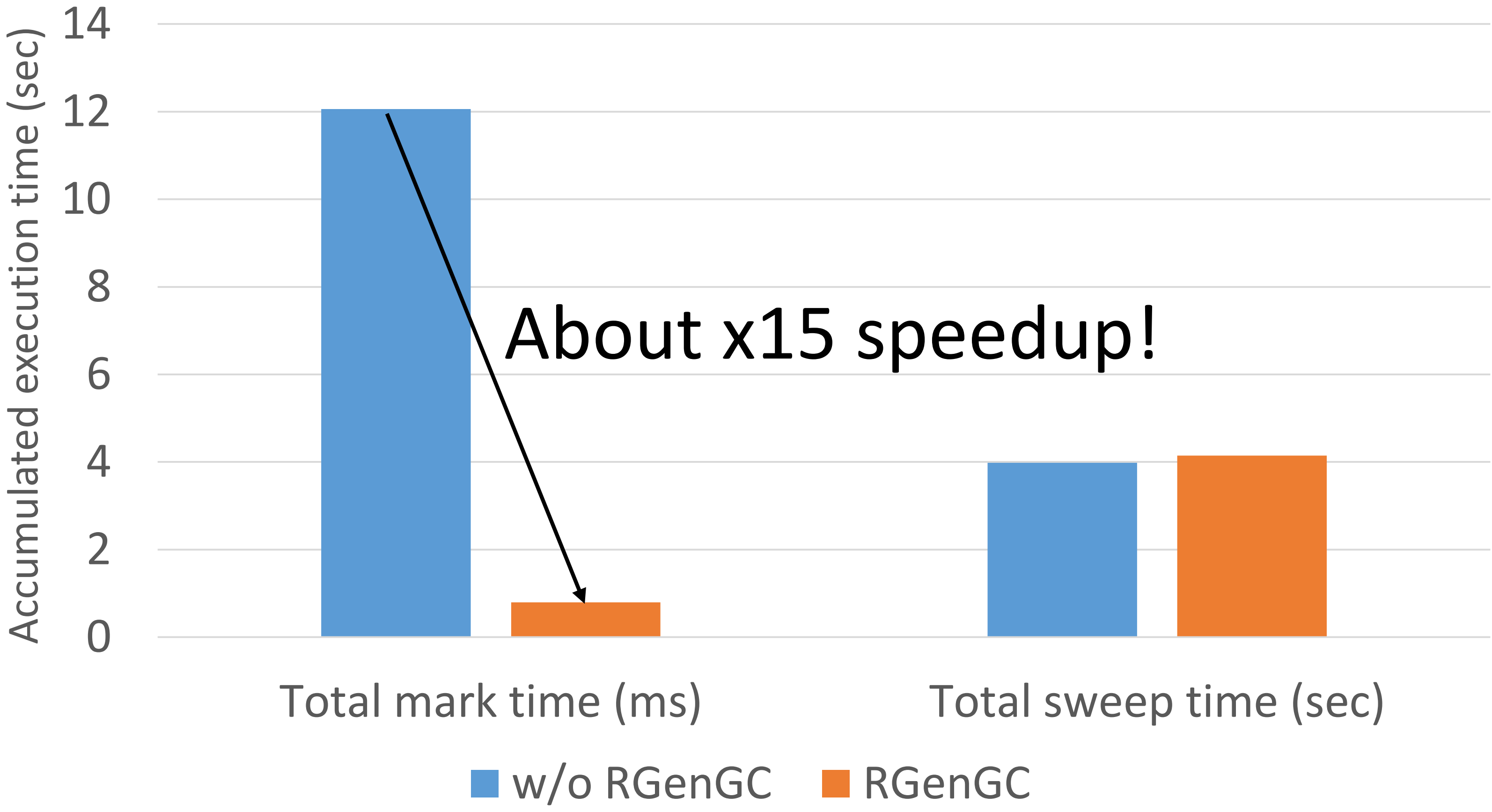
- **<u>Shorter mark time (good)</u>**
- **<u>Same sweep time (not good)</u>**

RGenGC
Performance evaluation (RDoc)



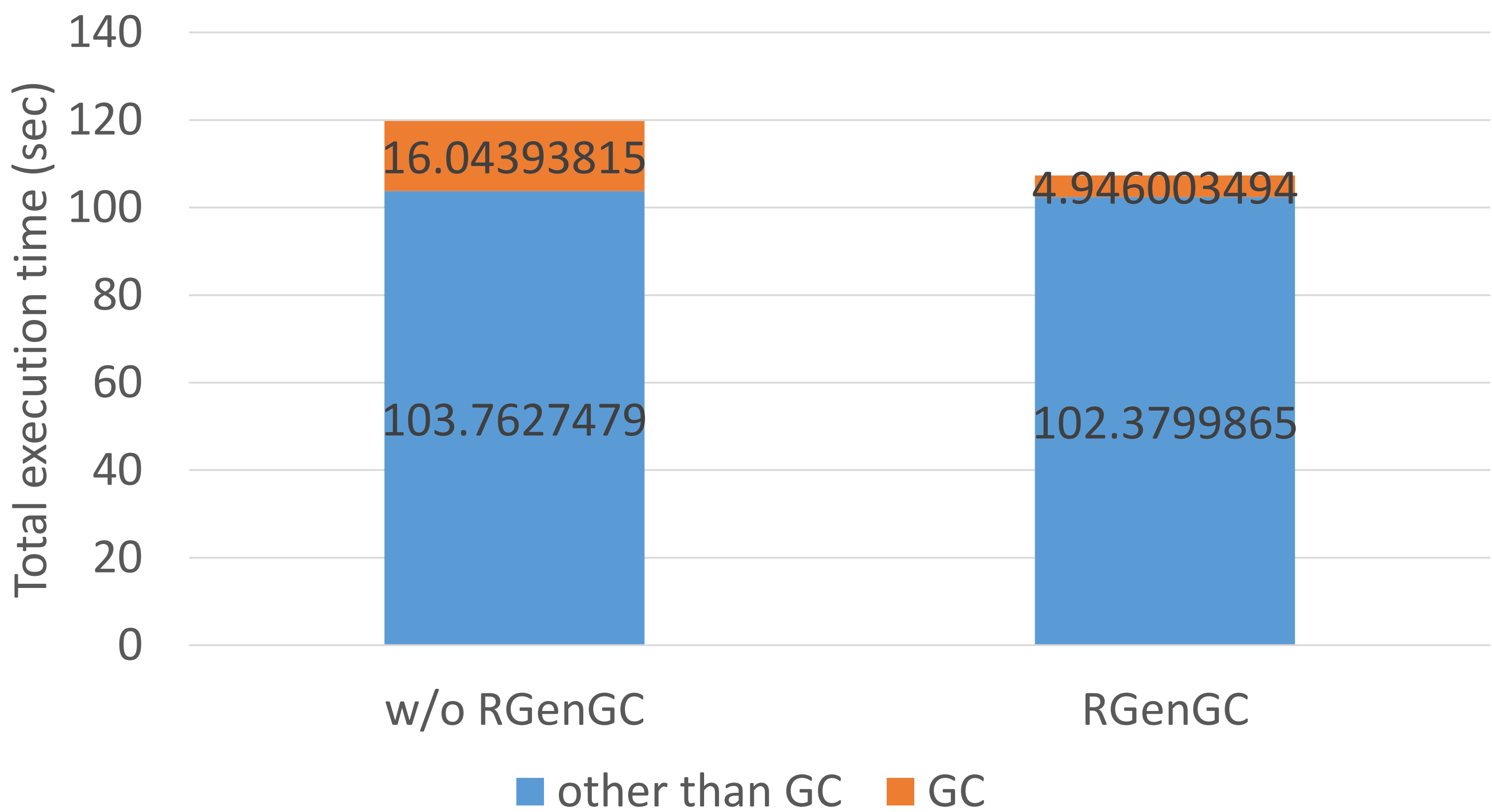
* Disabled lazy sweep to measure correctly.

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.

RGenGC: Summary

- RGenGC: Restricted Generational GC
  - New GC algorithm allow mixing “Write-barrier protected objects” and “WB unprotected objects”
  - (mostly) **No compatibility issue** with C-exts
- Inserting WBs gradually
  - We can concentrate WB insertion efforts for major objects and major methods

RGenGC
Remaining task

- Reduce old objects
  - Short lived objects promotion old-gen accidentally is harmful
- Minor GC / Major GC timing tuning
  - Too many major GC → slow down
  - Too few major GC → memory consumption issue
- Inserting WBs w/ application profiling
  - Profiling system
  - Benchmark programs
- Debugging/Detecting system for WBs bugs
- Improve sweeping performance

Remaining task

- To reduce old objects
  - → <u>3 Generations GC (3GenGC)</u>
- Minor GC / Major GC timing tuning
  - → Count oldgen objects count
  - → <u>Estimate oldgen space</u>

Three generational GC (3GenGC) Problem

- RGenGC introduces two generation “Young” and “Old”
- Some “short-lived” young object will be promoted as old-gen accidentally
  - Ex: loop{a = Object.new; b = Object.new}
- If such “short-lived” objects consumes huge memory, we need to free such objects
  - Ex: loop{Array.new(1_000_000)} # 1M entries

Three generational GC (3GenGC) Idea

- Add new generation “Infant” generation
  - Before: Young → Old
  - After 3gen GC: Infant → Young → Old
- Most of objects died in infant, and also young gen
  - Good: Avoid short-lived old-gen objects
  - Good: Reduce full-GC timing
  - Bad: Some overhead
- **We implemented this feature and evaluating it now**

Estimated oldgen space Problem

- We can not measure how oldgen objects consume memories collectly
  - A few oldgen objects can grab huge memory
  - Major GC takes long time

- Trade-off between time and memory usage

Estimate oldspace Idea

- Estimate how much memory old-gen objects consumes
- Invoke full GC when estimation exceed the threshold
  - Good: More correct major GC timing
  - Bad: Some overhead to measure memory size
  - Bad: More tuning parameters

- **We implemented this feature and evaluating it now**

Future work
For smarter object management

- Need more tuning
  - Some program slower than Ruby 2.0.0
  - We need practical benchmarks (other than RDoc)
- Parallel marking/sweeping
  - Parallel sweeping is already implemented <https://github.com/ko1/ruby/tree/parallel_sweep>, however, it does not improve performance…
- Concurrent marking to reduce full marking stop time

Summary of this talk

- Ruby 2.1.0 will be released soon!
  - 2013/12/25
  - Some new features and internal performance improvements
- Rewrite object management to improve performance
  - “gc.c” → 3414 insertions, 1121 deletions
  - <u>Allocation/Deletion trace mechanism</u>
  - <u>Introduce generational mechanism</u>
  - <u>Tuning GC parameters</u>
  - Optimize object allocation path
  - Refactoring (terminology, method path, etc)

Thank you

Koichi Sasada

Heroku, Inc.

<ko1@heroku.com>

Questions and answers

Questions and Answers RGenGC and CoW friendly

- No problem because only touch flags for oldgen and shady

Questions and Answers GC + Threads

- Parallel GC
  - Run GC process in parallel (simultaneously)
  - Parallel marking
  - Parallel sweeping (in today's talk)
- Concurrent GC / Incremental GC
  - Run ruby threads (mutator threads) and GC threads concurrently
  - Major GC consumes huge time (same as current GC) → Need concurrent GC to reduce pause time
  - New WB API is also designed for concurrent GC