Gradual Write-Barrier Insertion into a Ruby Interpreter

Koichi Sasada
Cookpad Inc.

ISMM 2019, June 23, 2019

Summary

- Now Ruby interpreter (2.6, 2018) employed **advanced GCs**.
  - **Generational GC** from Ruby 2.1 (2013)
  - **Incremental GC** from Ruby 2.2 (2014)
  - Ruby 2.0 and before used naïve “M&S GC” algorithm
- **Write barriers** (WBs) were issue to introduce these GCs.
  - To keep compatibility, we are not able to introduce WBs for 3rd party C-extension libraries.
- Proposal: New concept: “WB unprotected object”
  - Giving up WB insertion completely, but mark “WB unprotected”
  - Invented at 2013 for Ruby 2.1.
  - We can introduced advanced GCs with **keeping compatibility**.
- Our approach allows **Gradual WB development**.

Background
Ruby language

- Ruby is Object-Oriented programming language
  - Developed by Yukihiro Matsumoto (1993~)
  - Developed actively.
    - Koichi is one of the Ruby committers working on VM, GC, Concurrency management and so on.
- Ruby on Rails web-application framework is used widely, in world-wide.
- Several Ruby interpreters are available.
  - **“ruby”** command written in C (target of this research)
  - JRuby, Truffle Ruby written in Java
  - mruby written in C, for embedded systems

Background
Ruby (Ruby on Rails) is used seriously.

- One of our service
  - 72 countries, 29 languages
  - Around **96 million** monthly unique users (2019/03/31)
- is written in **Ruby** language

→ Performance of Ruby has huge impact, at least on our business

Background
GC before Ruby 2.1 (~2013)

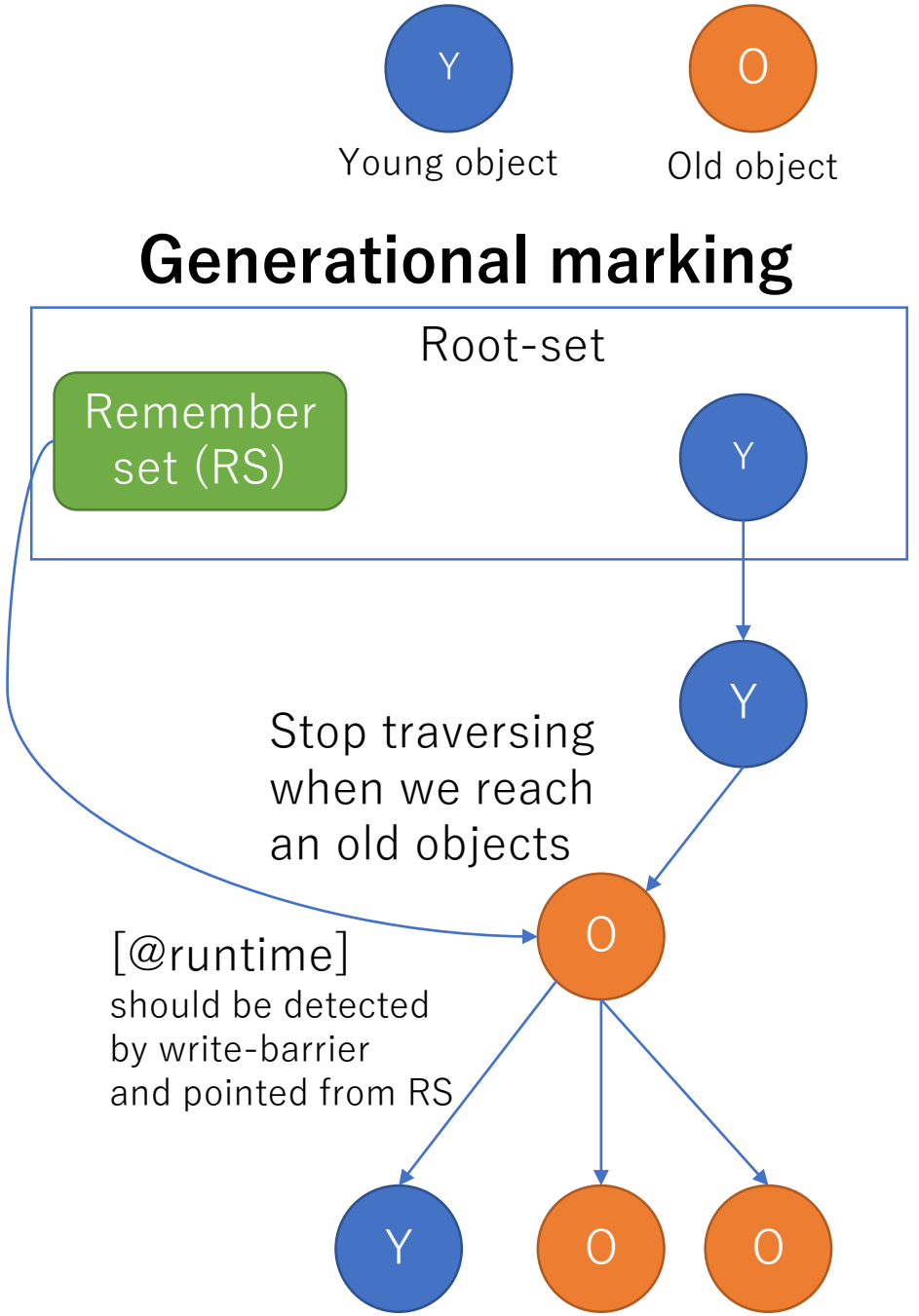
- Mark and Sweep GC
  - M&S GC stops application long time.
  - This was one of reason why **“Ruby is SLOW”**.
- Conservative marking
  - Allows to write C implementation **without special macros.**
    - ex) Free to update references in C assignments.

```
// New reference from an Array object to obj
RARRAY_PTR(ary)[10] = obj;
```

  - Ruby supports **C-extension libraries** with this technique.
    - 3rd party can extend Ruby with C-extensions.
    - There are many C-extensions to support Ruby’s eco-system.
  - Moving is not allowed (mostly is acceptable)

Background
Generational GC

- GenGC is well-known technique.
  - **Faster** than full GC because collecting only young objects.
- GenGC requires write-barriers
  - To detect “Old” to “Young” reference, **write-barriers (WBs)** should be introduced.
  - **“Completeness”** is required.
    - 1 oversight cause fatal error.



Problem: Inserting WBs

- Issue: Development cost
  - Practically, it is **very difficult** task to introduce WBs into Ruby code (250K lines in C) at once
- Issue: Compatibility
  - We need to re-write C code with WBs if needed.
  - We **can not modify** 3rd party C-extension libraries.
  - Drop old libs? vs. Give up GenGC?
    - If we need to rewrite all C-extensions, the update should be very difficult for existing Ruby users.
    - Make a new interpreter natively support good GC?

Background and Problem

- Ruby 2.0 (2013) needed Generational GC for speed.
- However, inserting write-barriers into C source code *completely is*

  difficult for huge ruby's source code

  impossible for 3rd party C-extensions

Trade off between Speed and Compatibility

Proposal: WB unprotected objects

- Introduce **WB protected and unprotected attribute** for **all objects**
  - WB protected objects (**WBp**) can detect new reference creation from them. Unprotected objects (**WBunp**) can not.
  - GC algorithm need to care about WB unprotected objects.
- Increase WB protected objects **gradually.**
  - When we insert WBs into class **K** data structure, then all instance of class **K** are WB protected objects.
  - We can priorities WB insertion development

    → Flexible development
    - **Frequently used** data types (**Array, Hash,** …) have high priority.
    - Scalar data types (String, …) also have high priority because it is easy.

WB unprotect operation

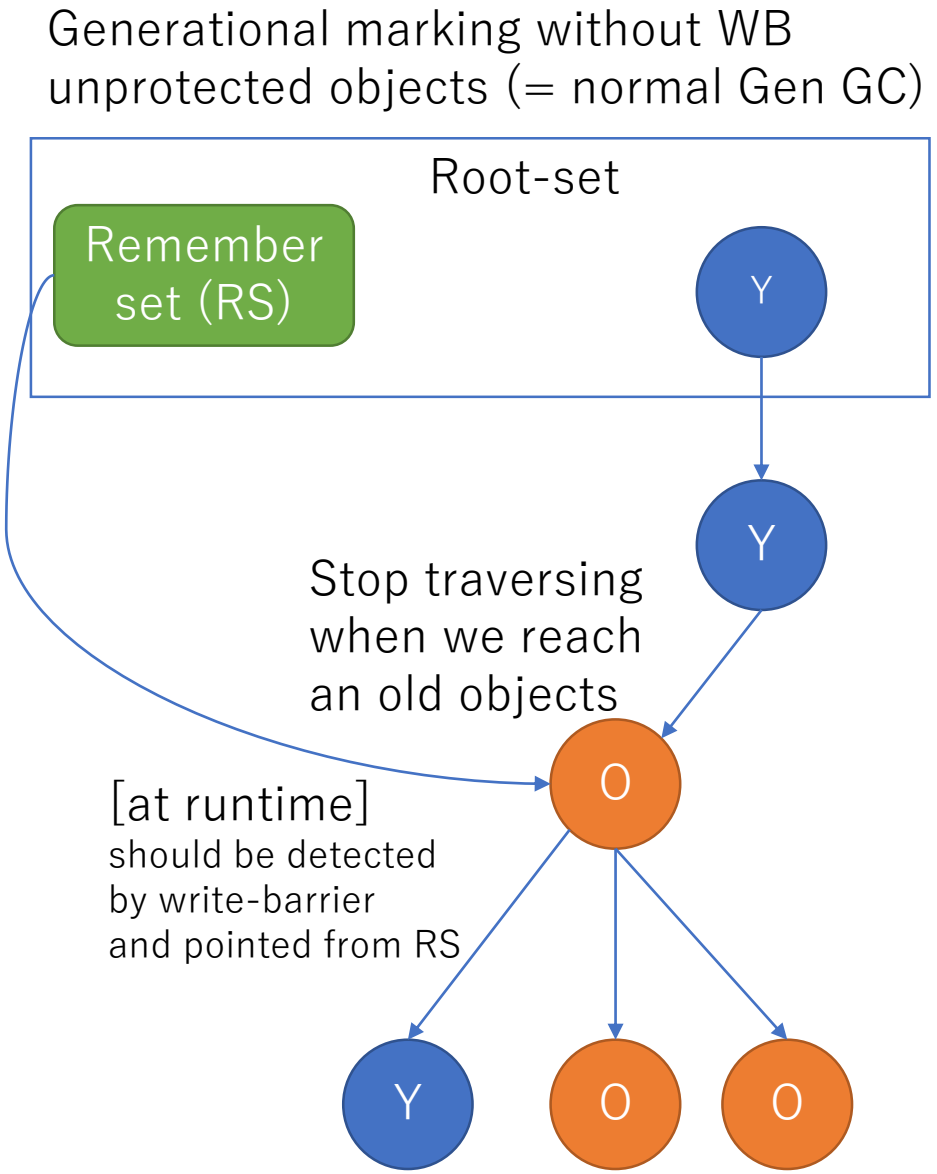
- *WBp* can become *WBunp* by **WB unprotect operation**
  - If C code acquire internal data structure such as Array memory block, the Array object becomes *WBunp* because unexpected reference can be created by C code.

ex)

```
// RARRAY_PTR() macro makes “ary” unprotect.
ptr = RARRAY_PTR(ary);
// This line creates new ref: ary→obj
// which GC can not detect.
ptr[10] = obj;
```

Generational marking

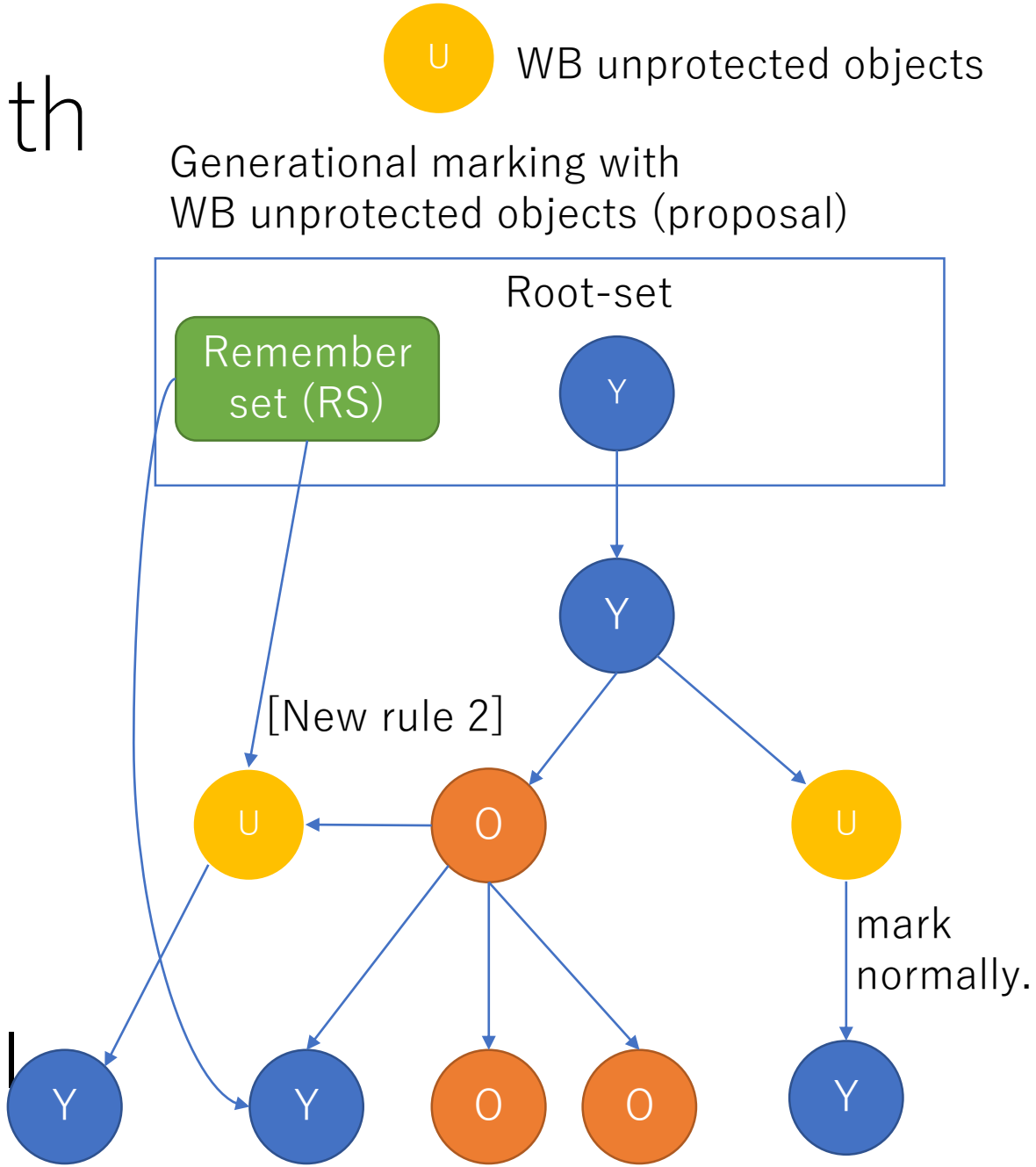
- Basic algorithm
  - Two generations: Young and Old
  - Objects have age 0~3 and age 3 is an old object.
  - Only generational marking (not generational sweeping)
  - Minor GC and Major (full) GC



NOTE: See our paper to refer complete algorithm

Generational marking with WB unprotected objects

- Additional Rule for *WBunp*
  1. *WBunp* can not promote.
  2. If old objects refer to a *WBunp*, then the *WBunp* is remembered **until next major GC** because *WBunp* can refer young objects.
  3. If Old objects become *WBunp* by *WBunpOp*, it will be remembered.



Incremental marking with WB unprotected objects.

- At the end of normal incremental marking (3 color algorithm), mark all living (black) *WBunp* at once (not incremental).
  - This phase can introduce long pause time.
  - O(n), n is the number of living WBunp.

NOTE: See our paper to refer complete algorithm

Implementation technique
Bitmap

- We introduce bitmap to represent *WBunp* .
  - With this bitmap and marking bitmap, we can easy to list “living *WBunp*” for incremental GC.

***WBunp* bit**	0	1	1	0	0	0	0
Marking bit	1	1	0	0	1	1	0

Living WB unprotected object

Evaluation
Measurements

- Several measurements
  - Microbenchmark
  - Application benchmark
    - RDoc
    - Ruby on Rails web application
- Environment
  - Intel(R) Core(TM) i7-6700 CPU, 64GB of memory, Ubuntu 18.04.2, gcc 7.3.0
  - ruby 2.7.0dev (2019-03-08 trunk 67194) x86_64-linux

Evaluation
Microbenchmark

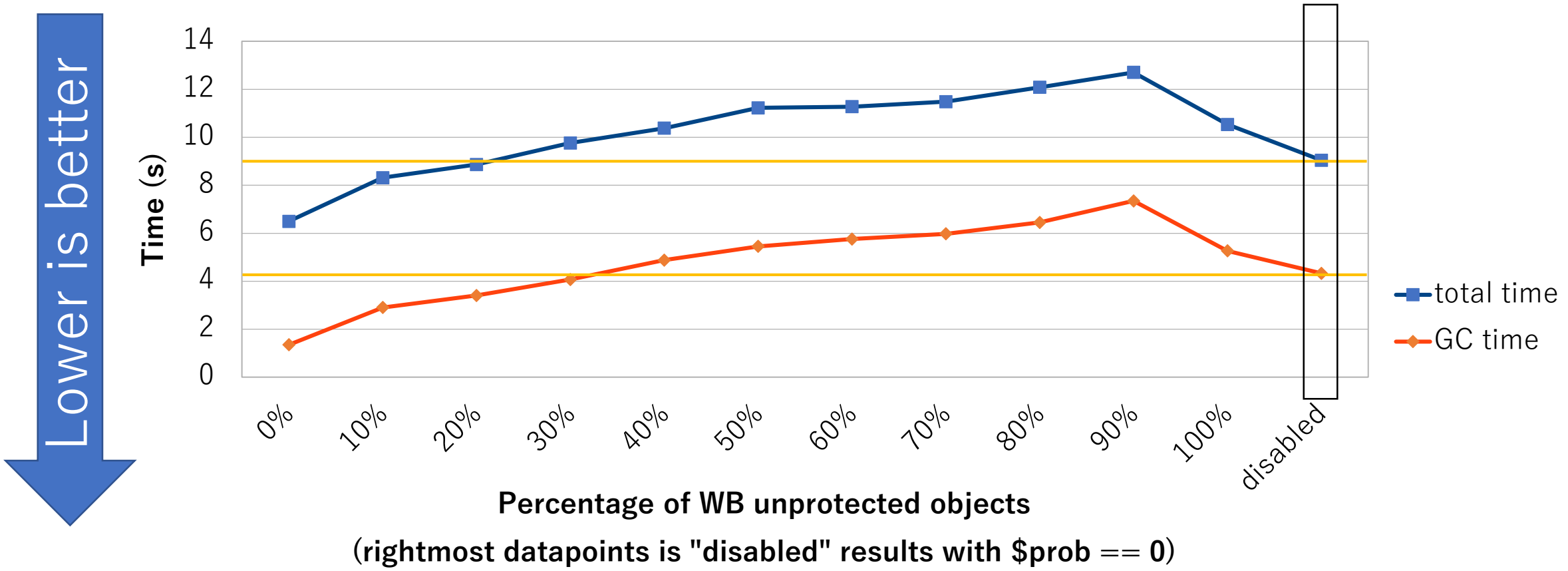
```
def make_linked_list n
  list = []
  n.times{
    list = [list]
    # $prob is percentage
    # of WB unprotected objs.
    if rand(100) < $prob
      list.wb_unprotect
    end
  }
  list
end
```

```
# Create a long linked list
huge_list = make_linked_list(
  10_000_000)

# Create 100 M empty arrays
# to invoke minor GC
100_000_000.times { [] }
```

We can control the ratio of WB unprotected arrays.

Evaluation
Microbenchmark



Increasing *WBunp* slows down the application.

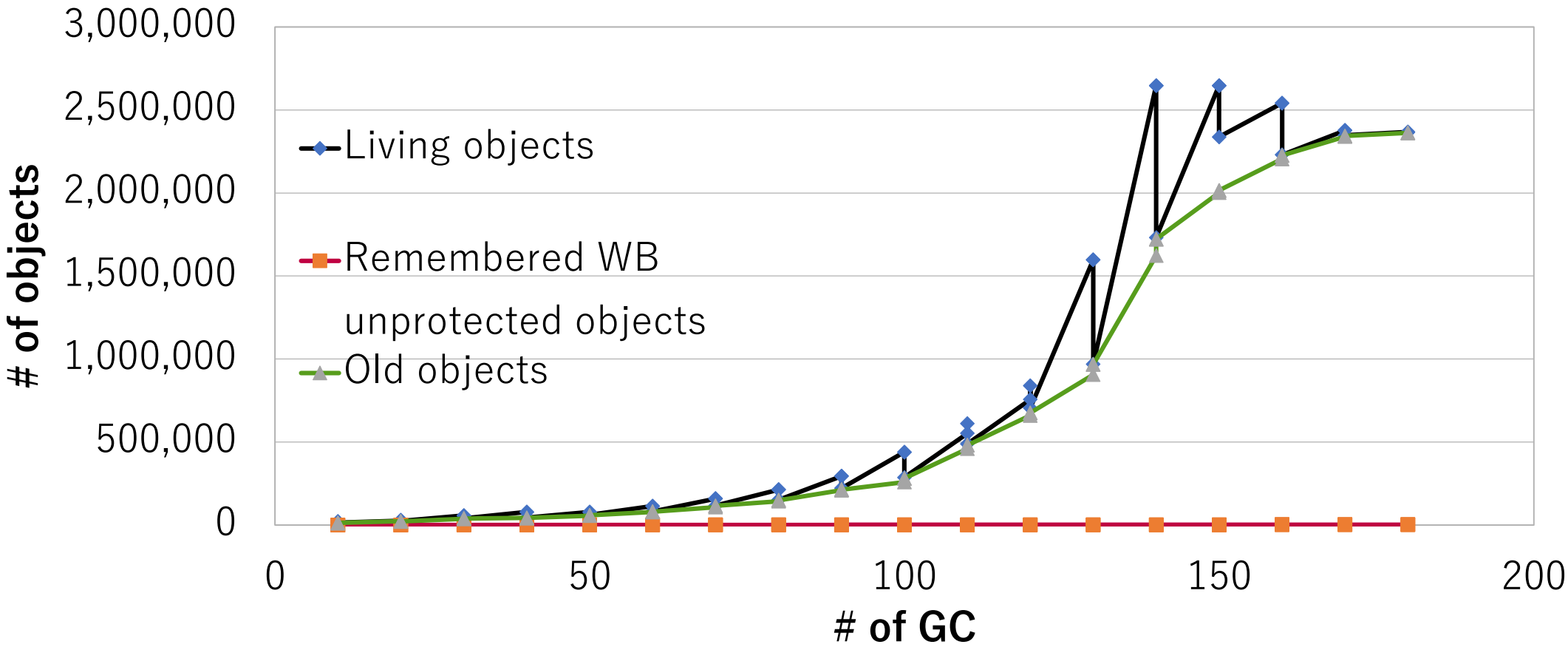
Application benchmark
RDoc

- RDoc is document generation system
  - Reading ruby/c source code and generate formatted reference.
  - Source is ruby's source code.

	Total time (s)	GC time (s)
Disabled	30.46	10.20
Enabled	22.57	1.63

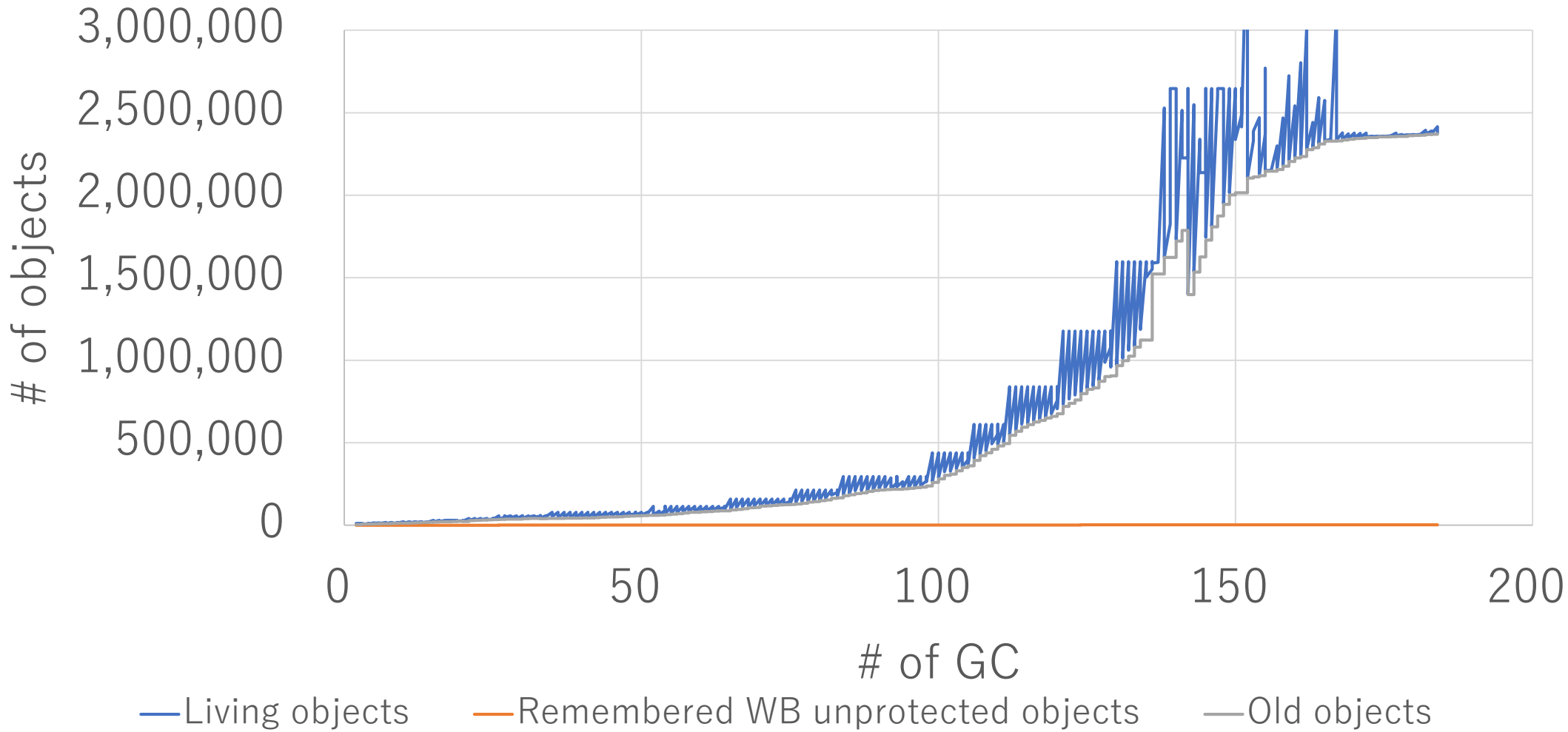
The ratio of *WBunp* objects is 2%.

Application benchmark
RDoc (sampling per 10 GCs)



There are only few remembered WB unprotected objects

Application benchmark
RDoc (sampling per 1 GC)



Application benchmark
Discourse (Ruby on Rails web app)

(response time percentile in milliseconds)

Page		50%	75%	90%	99%
categories	Disable	36	44	148	159
	Enabled	35	36	52	87
home	Disable	39	148	161	164
	Enabled	39	42	56	96
categories_admin	Disable	64	179	186	194
	Enabled	63	70	82	147
home_admin	Disable	71	180	186	194
	Enabled	67	80	86	157

Generational GC improve performance (~90%)
Incremental GC is effective, but not enough (99%).

Evaluation
Gradual WB development

WB implementation history	WB protected classes
Ruby 2.1 (2013)	**Container types: Array, Hash**, Struct, Object (User defined classes), Class **Scalar types: String**, Range, Regexp, RubyVM::ISeq (bytecode)
Ruby 2.4 (2016)	Proc (closure class), Env (local variables) (postponed to impl. them at 2013 **because it was difficult task**)
Ruby 2.5 (2017)	Dir, Binding, Thread::Queue, Thread::SizedQueue and Thread::ConditionVariable

- We can **give up** difficult WB insertions
  - Some kind of “Class” objects has complex relations and I can not remove a bug

  → Make them *WBunp* with WB unprotect operation

Related work

- TruffleRuby introduce special wrappers to support C-extension library[7]
  - Issue: We need two GCs
- Special C-preprocessor to auto-WB insertion [5]
  - Issue: False positive. Difficult to maintain.
- Using hardware memory protection to detect writing [3]
  - Issue: Portability problem (difficult to maintain)
- Scan all old spaces [1]
  - Issue: Scanning cost

Summary

- Now Ruby interpreter (2.6, 2018) employed **advanced GCs**.
  - **Generational GC** from Ruby 2.1 (2013)
  - **Incremental GC** from Ruby 2.2 (2014)
  - Ruby 2.0 and before used naïve “M&S GC” algorithm
- **Write barriers** (WBs) were issue to introduce these GCs.
  - To keep compatibility, we are not able to introduce WBs for 3rd party C-extension libraries.
- Proposal: New concept: “WB unprotected object”
  - Giving up WB insertion completely, but mark “WB unprotected”
  - Invented at 2013 for Ruby 2.1.
  - We can introduced advanced GCs with **keeping compatibility**.
- Our approach allows **Gradual WB development**.

Message to researchers

- Ruby interpreter is used by **many people** and the **performance is still issue**.
- We (other Ruby committers and Cookpad) can help your research on Ruby.
- Please contact us if you have interest: **ko1@atdot.net**

Thank you for your attension!