Object lifetime analysis with Ruby 2.1

Koichi Sasada

<ko1@heroku.inc>

Who am I ?

- Koichi Sasada a.k.a. ko1
- From Japan
- 笹田 (family name) 耕一 (given name) in Kanji character

Who am I ?

- 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

Advertisement

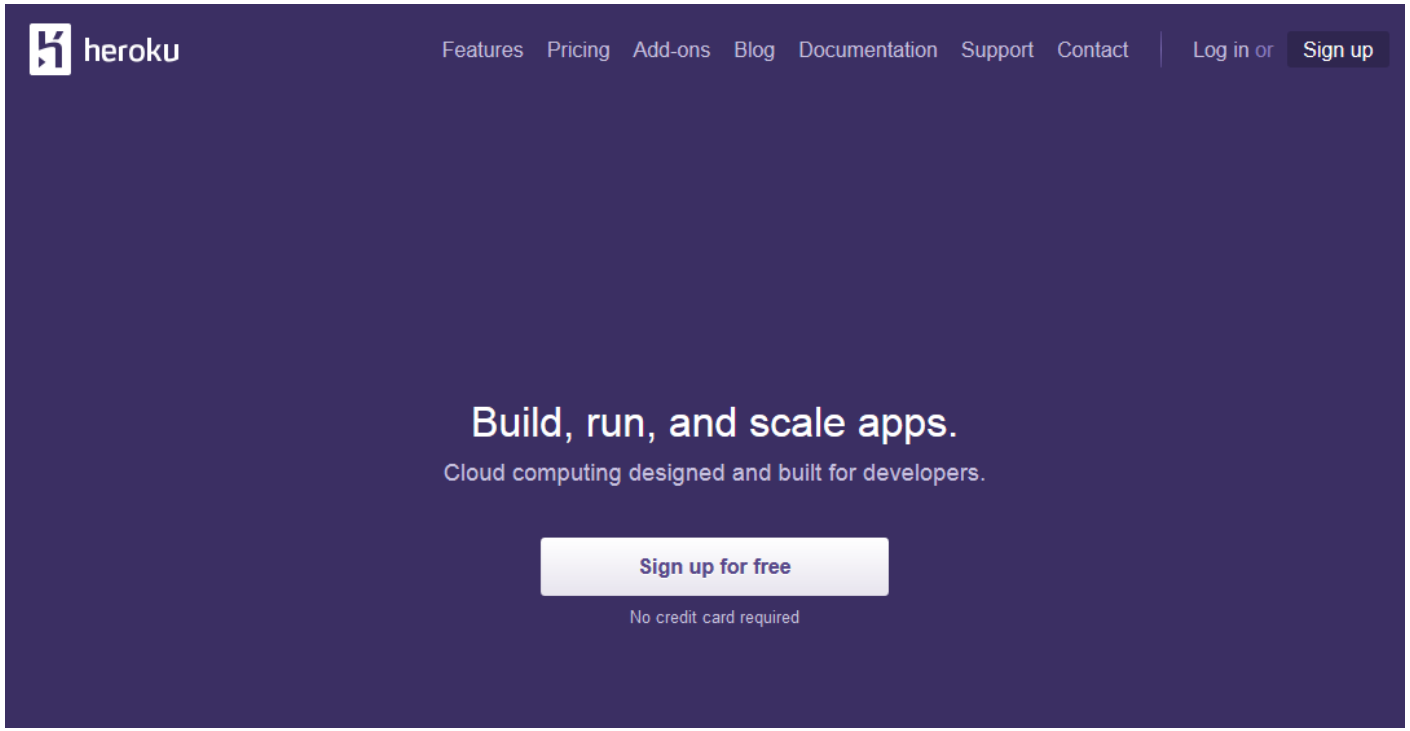
Ruby Association

- Foundation to encourage Ruby dev. and communities
  - Chairman is Matz, Located at Matsue-city, Shimane, Japan
- Activities
  - Ruby programmer certification program
    - http://www.ruby.or.jp/en/certification/examination/ in English
  - 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
  - Grant project. We have selected **3 proposals** in 2013
    - Win32Utils Support, Conductor, Smalruby - smalruby-editor
    - We will make this grant 2014!!
  - **Donation** for Ruby developments and communities

Advertisement

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

You should know about Heroku!!



Advertisement

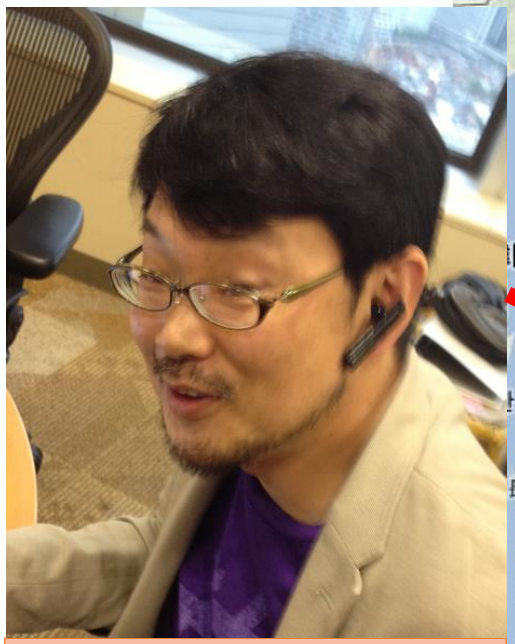
This presentation is supported by

Advertisement

- 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 in Heroku in Japan

Nobu @ Tochigi
Patch monster



Matz @ Shimane
Title collector

ko1 @ Tokyo
EDD developer

Object lifetime analysis with Ruby 2.1,
RubyConf.tw 2014, K.Sasada from Heroku, Inc.

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 2.2!!

Now, Ruby 2.1 is old version for us.

Ruby 2.1
Current stable

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

Ruby 2.1

- **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 the biggest change Version policy

- Change the versioning policy
  - Drop “patch level” in the version
  - Teeny represents patch level
    - Release new teeny versions about every 3 month
    - Teeny upgrades keep compatibility
  - Minor upgrades can break backward compatibility
    - We make an effort to keep compatibility

      (recently. Remember Ruby 1.9 ☺)

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

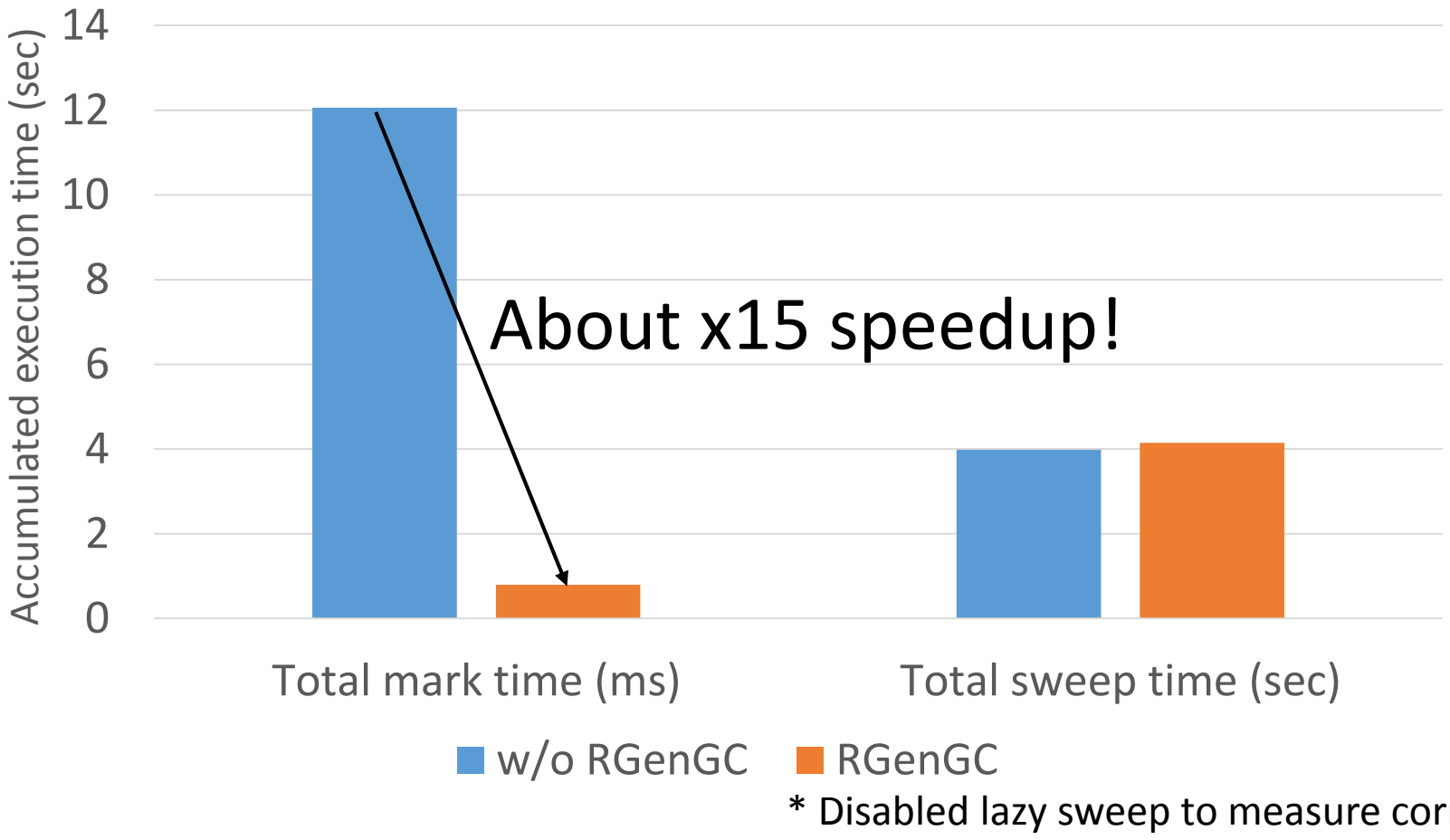
Ruby 2.1 Runtime new features

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

Performance improvements

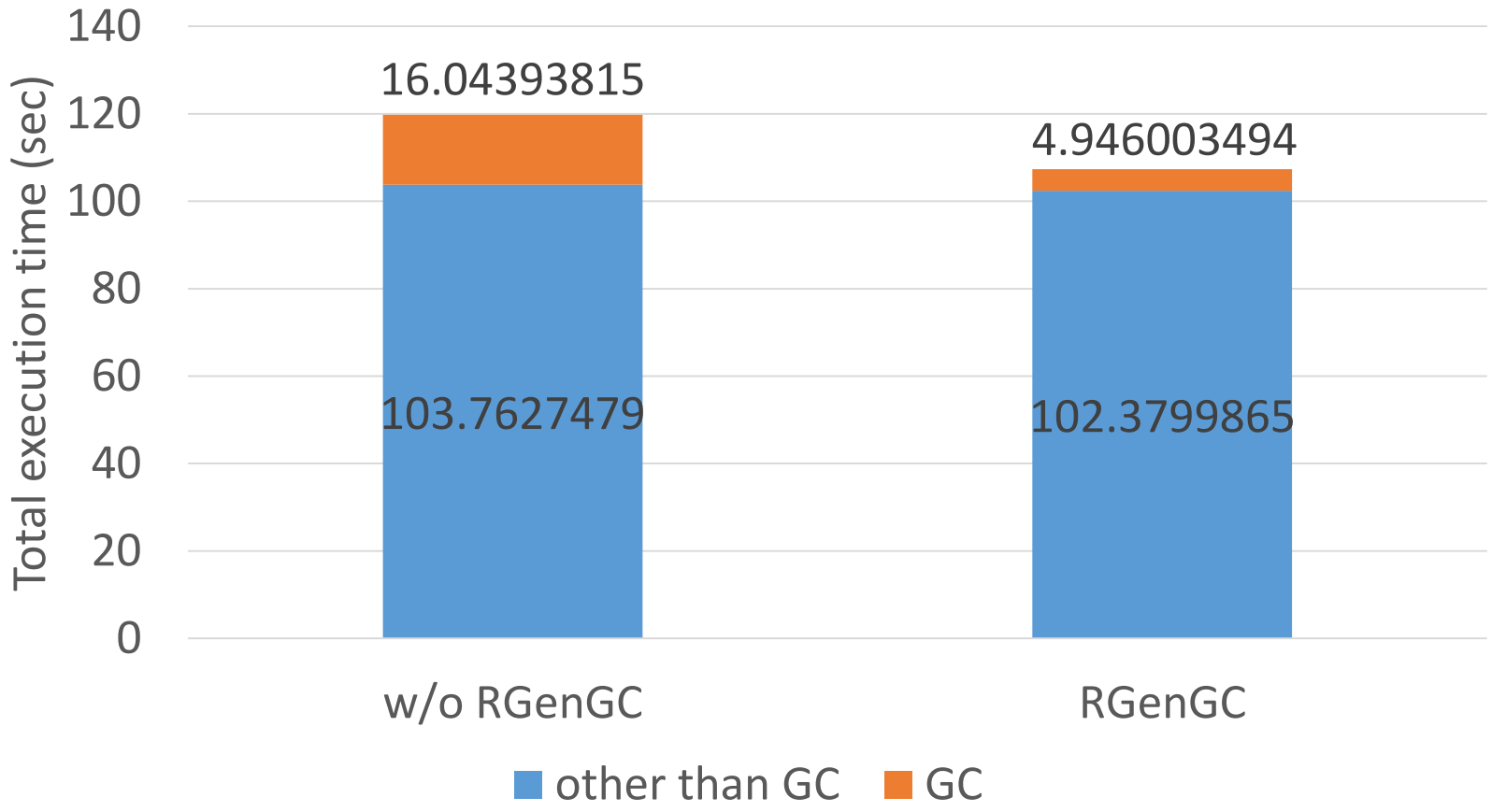
- Optimize “string literal”.freeze
- Sophisticated inline method cache
- Introducing Generational GC: RGenGC

RGenGC
Performance evaluation (RDoc)



* Disabled lazy sweep to measure correctly.

RGenGC
Performance evaluation (RDoc)



* 12% improvements compare with w/ and w/o RGenGC
* Disabled lazy sweep to measure correctly.

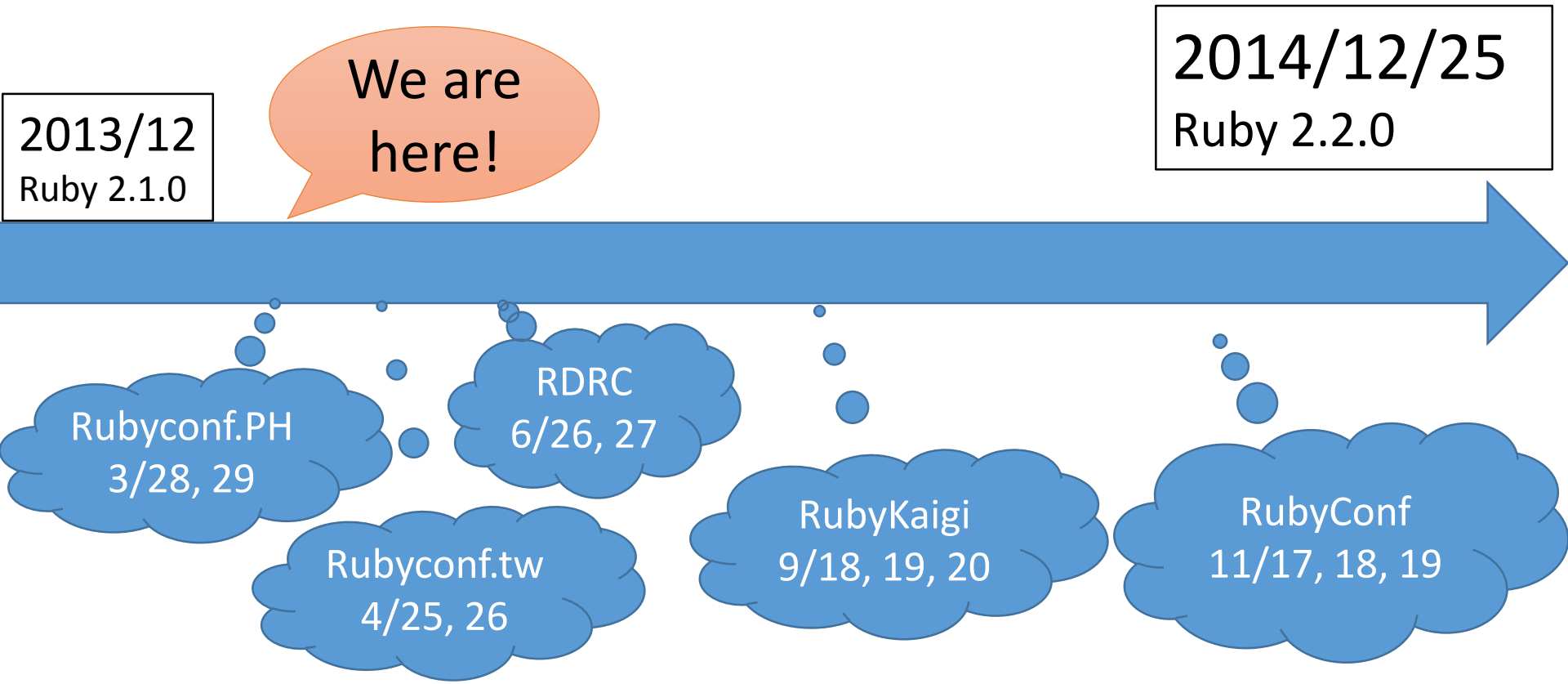
Ruby 2.2
Next version

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

Schedule of Ruby 2.2

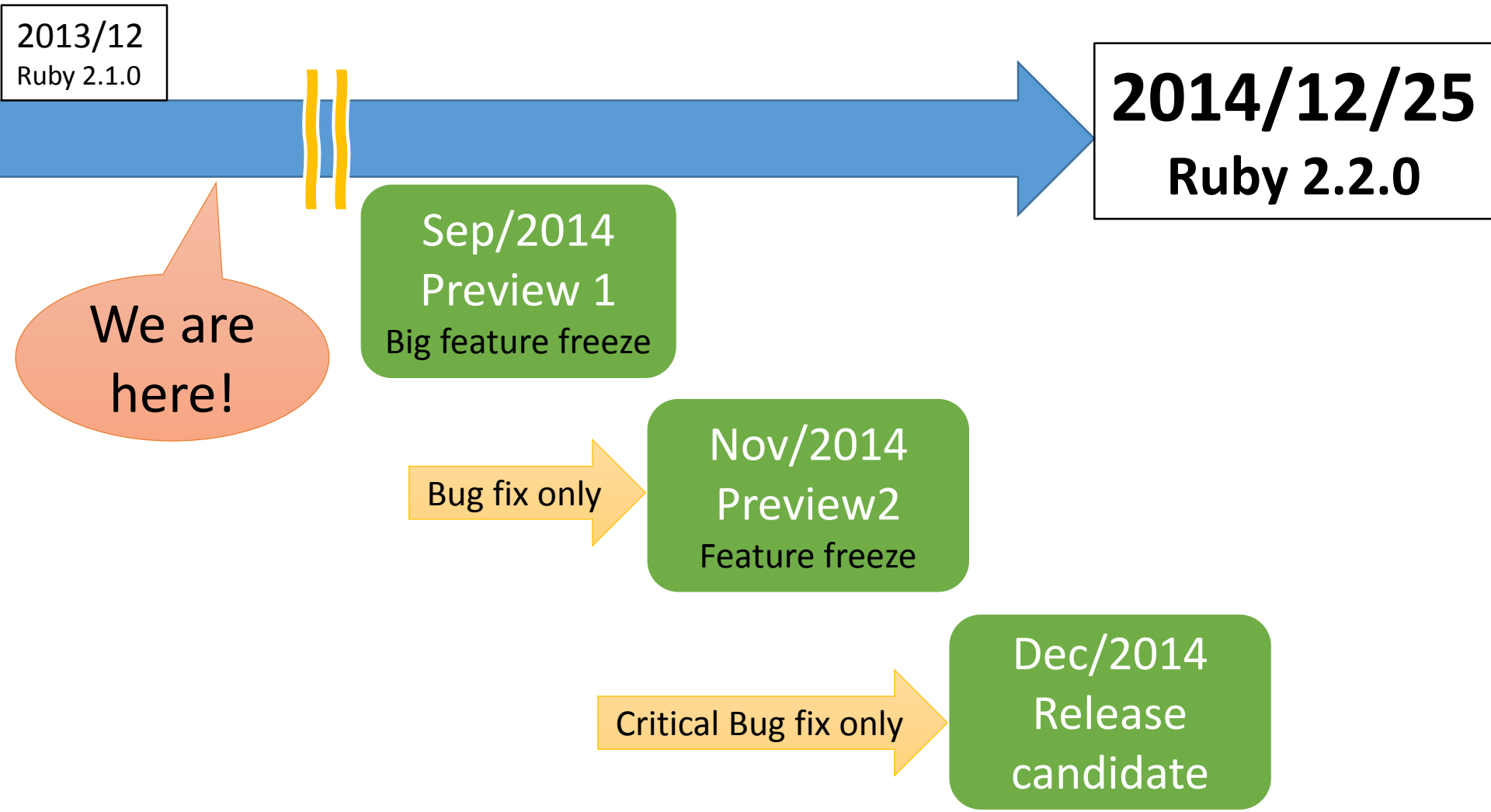
- Not published officially
- Schedule draft is available by Naruse-san
  - https://bugs.ruby-lang.org/projects/ruby-trunk/wiki/ReleaseEngineering22

Ruby 2.2 schedule



Events are important for
EDD (Event Driven Development) Developers

Ruby 2.2 (rough) schedule



2.2 big features (planned)

- New syntax: not available now
- New method: not available now
- Internal
  - GC
    - **Symbol GC (merged recently)**
    - **2age promotion strategy for RGenGC**
    - **Incremental GC** to reduce major GC pause time
  - VM
    - More sophisticated method cache

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

Object lifetime

Ruby's object/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

Why we don't need to clean up objects?

Garbage collection
The automatic memory management

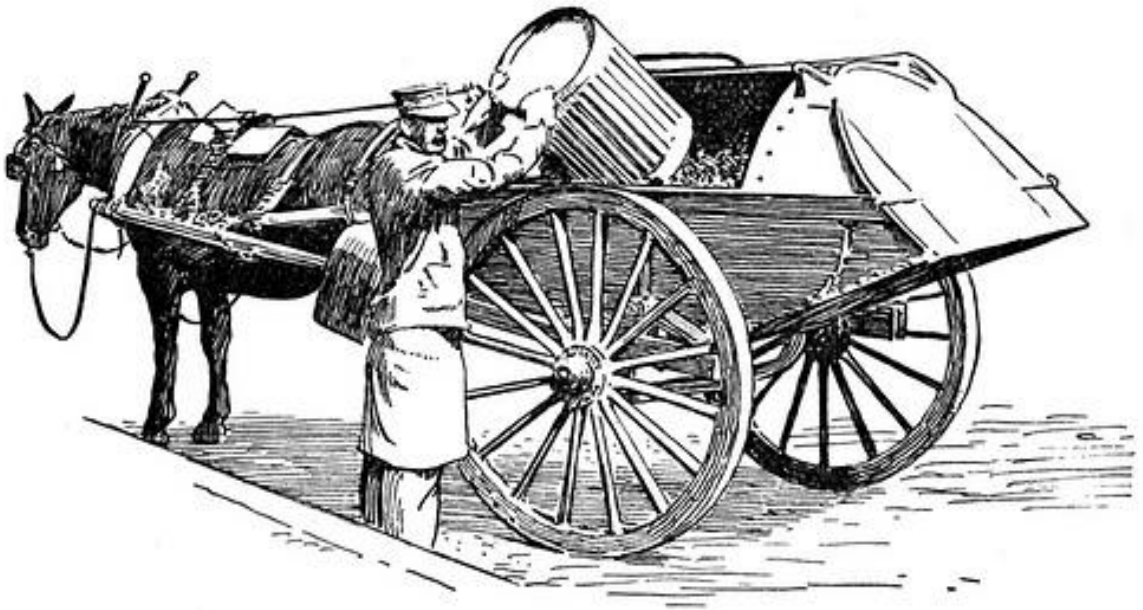


FIG. 109. — A GARBAGE COLLECTOR.

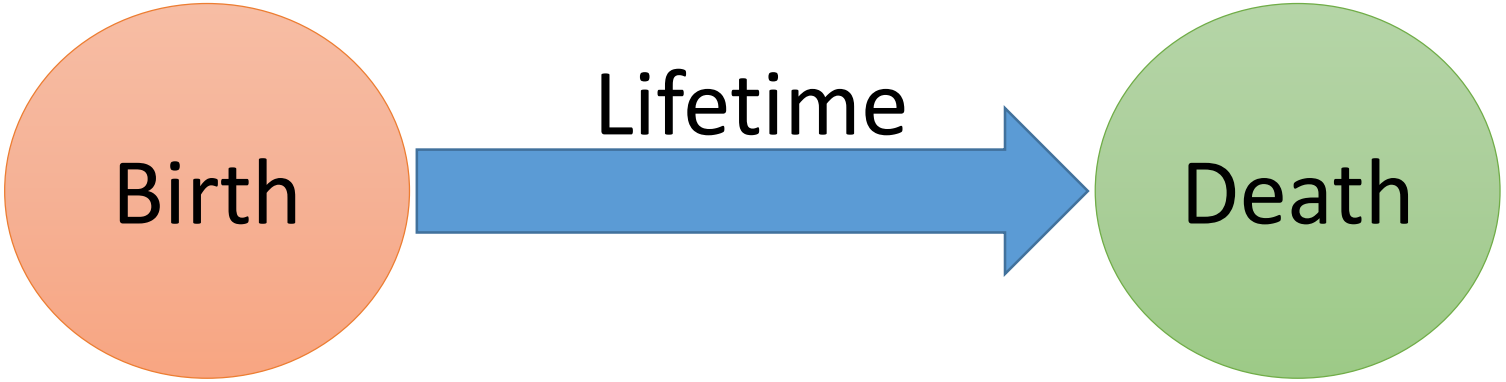
http://www.flickr.com/photos/circasassy/6817999189/

Garbage collection
The automatic memory management

Garbage collection
The automatic memory management

Object lifetime

- Human's lifetime

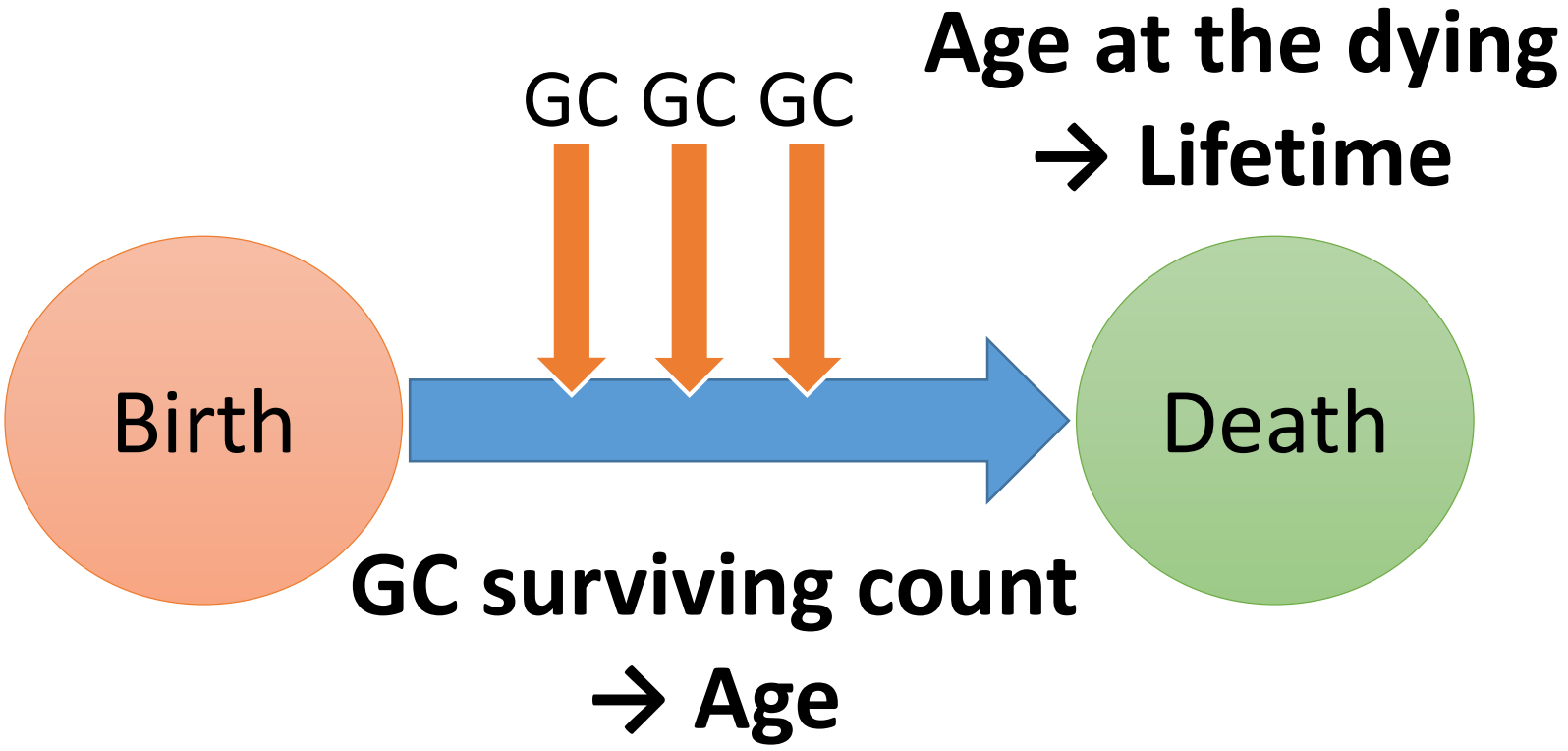


Object lifetime

# Object creation
obj = Object.new

Collected by GC
when nobody refers it

Object lifetime



Why lifetime is important?

- Old objects slow down major (full) GC
  - RGenGC have reduced GC overhead comes from old objects
  - However, major (full) GC needs a time
- Long life (old) objects can be unexpected memory leak

Reducing old objects makes
Ruby program faster ☺

Questions

- How to measure object lifetime?
- How to find old objects?

Allocation tracer gem

Allocation tracer

- Trace object allocations
  - Two features
    1. Show statistics data by line
    2. Show lifetime statistics table
  - Using newobj/freeobj internal hooks introduced from Ruby 2.1.0.
  - So that this gem only supports Ruby 2.1 and later

Allocation Tracer (1) Show statistics data by line

- Show statistics data by each lines
- Collect following data
  - Allocation count
  - Old object count
  - Total object lifetime (average lifetime by dividing allocation count)
  - Min/Max lifetime
  - Total amount of memory usage (average memory usage by dividing allocation count)

Allocation Tracer
(1) Show statistics data by line

Demo

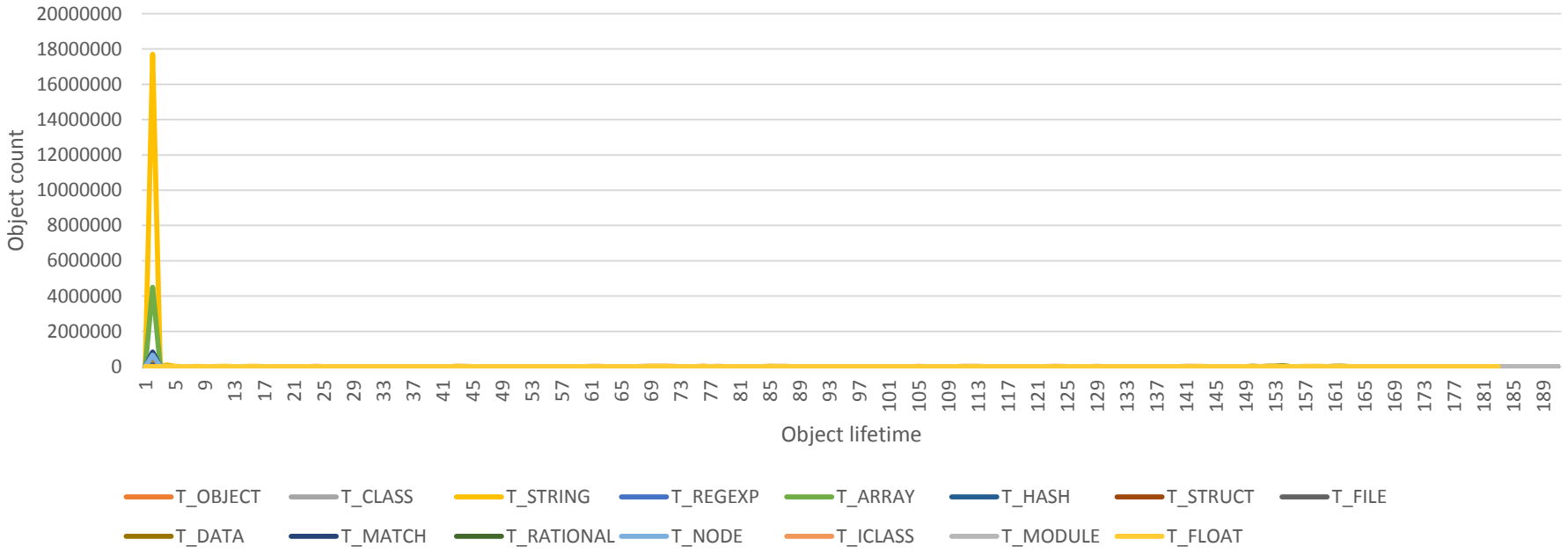
Allocation Tracer
(2) Object lifetime statistics

- Show object lifetime statistics table for each type

Age	0	1	2	3	4	5	6	7	8	9	10
T_OBJECT	7534	435532	6695	12916	2841	2298	2570	4222	3256	2659	4839
T_CLASS	0	90	1	0	0	0	0	0	4	5	1
T_STRING	116307	17706739	24244	97981	9461	5470	4825	7243	6533	5581	15826
T_REGEXP	0	41192	1	0	0	0	0	0	0	3	0
T_ARRAY	32874	4490228	5878	9262	4863	2567	2485	5697	3752	3033	5672
T_HASH	872	89158	113	323	90	85	93	138	161	101	183
T_STRUCT	1	8507	247	197	62	55	59	9	42	67	127
T_FILE	1	9330	5	2	1	1	0	0	0	0	0
T_DATA	8510	578913	146	69	96	53	77	54	65	90	62
T_MATCH	7	829992	52	0	0	0	0	0	0	0	0
T_RATIONAL	0	2	0	0	0	0	0	0	0	0	0
T_NODE	9333	649080	19438	4948	2665	7	17	5	9	26	135
T_ICLASS	0	90	1	0	0	0	0	0	0	1	1
T_MODULE	0	0	0	0	0	0	0	0	0	0	0
T_FLOAT	0	0	0	0	0	0	0	0	0	0	0

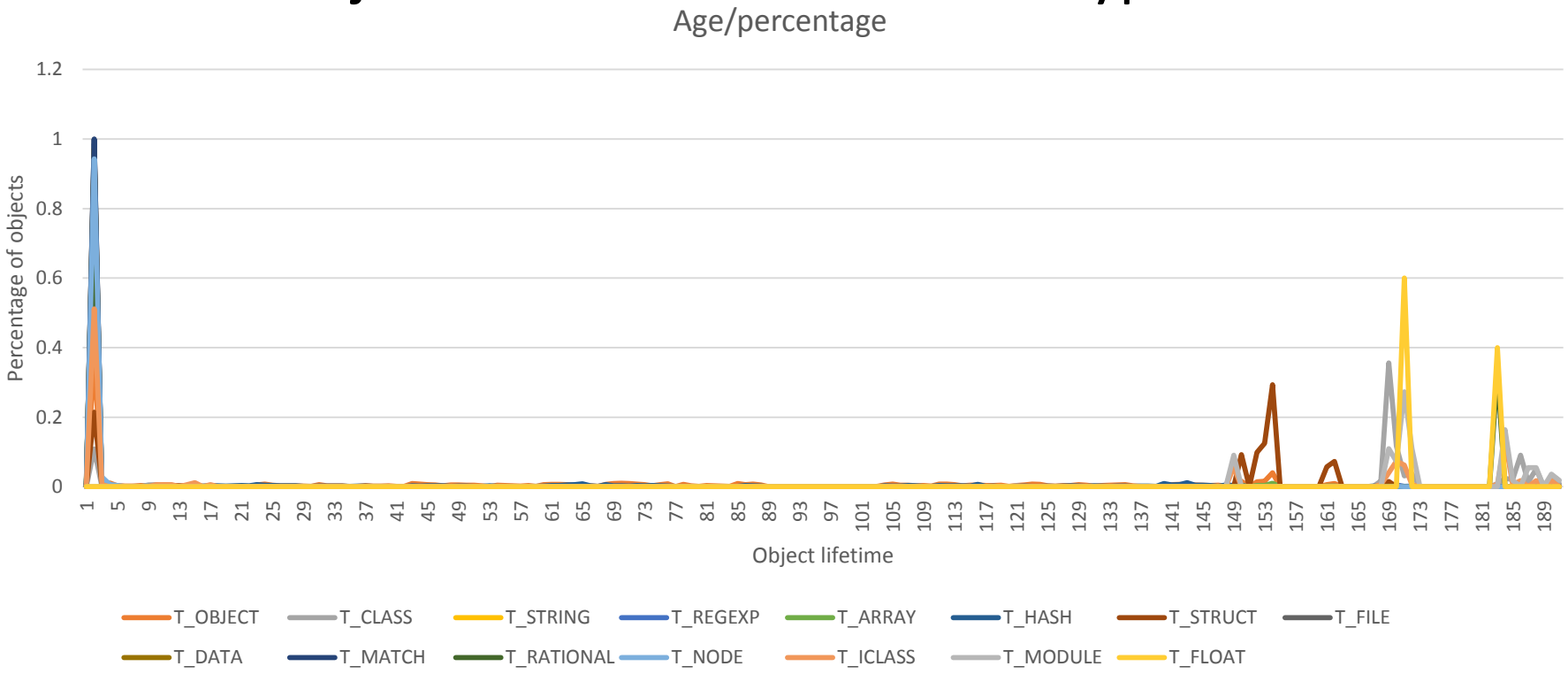
Allocation Tracer
(2) Object lifetime statistics

- Show object lifetime table for each type

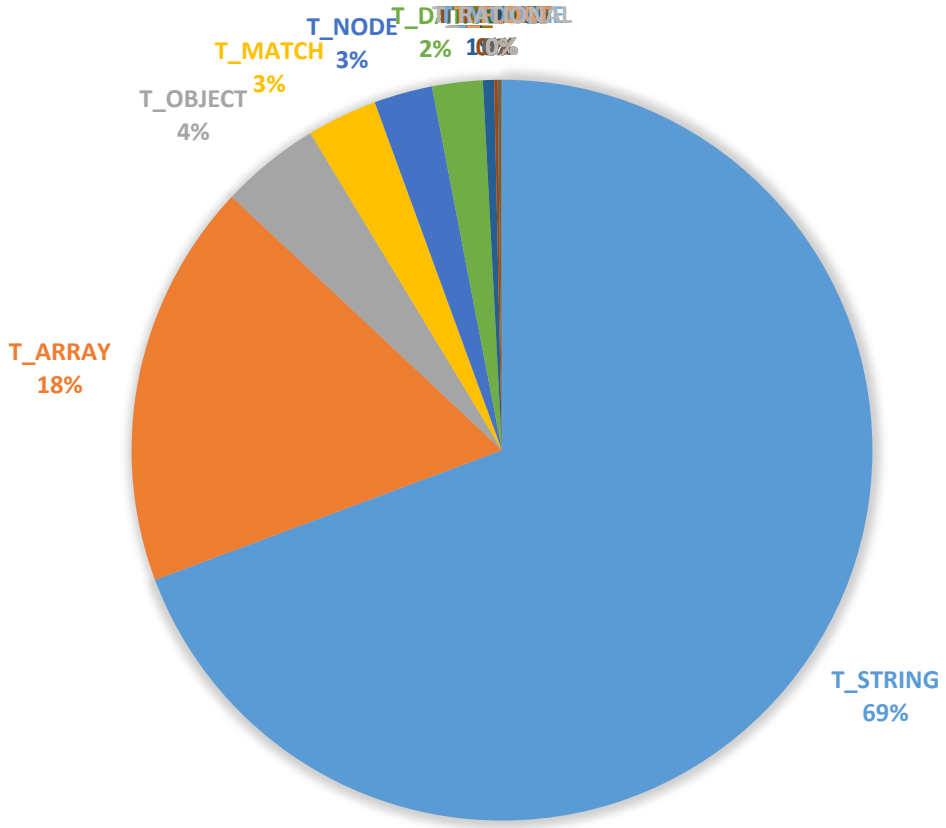


Allocation Tracer
(2) Object lifetime statistics

- Show object lifetime table for each type

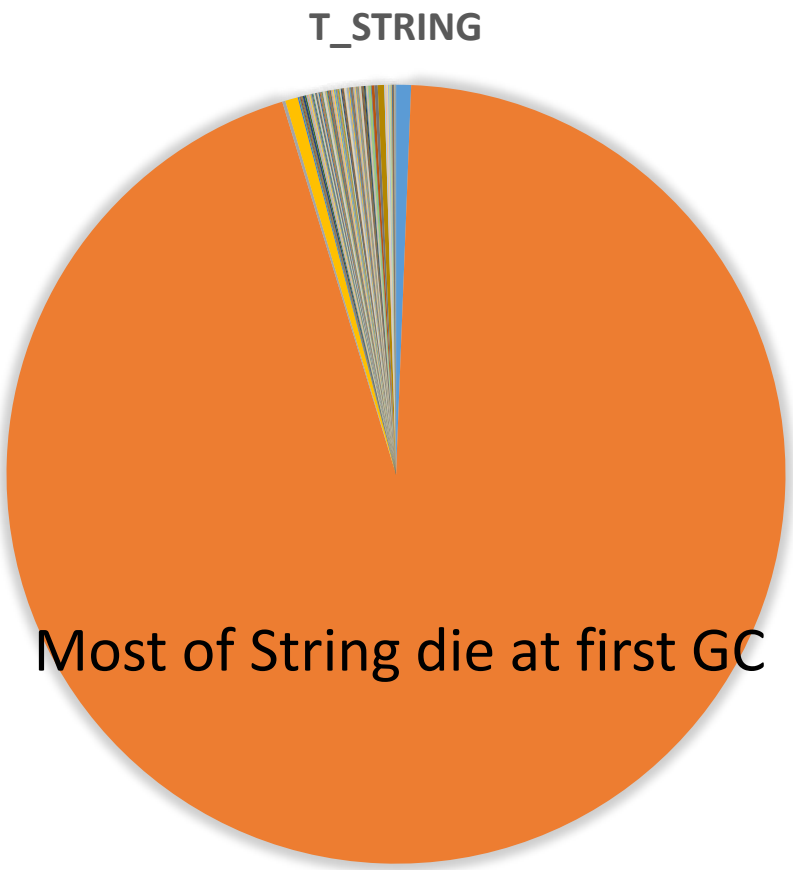


Allocation Tracer
(2) Object lifetime statistics



Most of allocated objects are **String**

Allocation Tracer demo (2) Object lifetime statistics



Allocation Tracer example

- Using “by line” statistics on RDoc program

Allocation Tracer example

path	line	count	total_memsize
/home/ko1/src/ruby/trunk/lib/rdoc/markup/parser.rb	327	128,097	498,086,366
/home/ko1/src/ruby/trunk/lib/rdoc/parser/c.rb	644	3,132	111,437,037
/home/ko1/src/ruby/trunk/lib/time.rb	325	1,146,900	85,714,539
/home/ko1/src/ruby/trunk/lib/rdoc/token_stream.rb	59	1,267,168	81,137,600
/home/ko1/src/ruby/trunk/lib/rdoc/store.rb	892	23,038	65,504,272

lib/rdoc/markup/parser.rb:327

@input.byteslice(0, byte_offset).length

- @input is String
- @input.byteslice(0, byte_offset) makes new String
- Return only length of sliced string

  → Created string object is completely temporary

lib/rdoc/markup/parser.rb:327

Introduce String#byteslice_length

@input.byteslice_length(0, byte_offset)

- No temporary String object

After introducing new method…

path	line	count	total_memsize
/home/ko1/src/ruby/trunk/lib/rdoc/parser/c.rb	644	4,369	93,019,334
/home/ko1/src/ruby/trunk/lib/rdoc/token_stream.rb	59	1,386,966	88,834,304
/home/ko1/src/ruby/trunk/lib/time.rb	325	1,137,612	84,959,225
/home/ko1/src/ruby/trunk/lib/rdoc/ruby_lex.rb	162	944,833	60,724,300

And total consuming memory reduced from about 140MB to 120MB

Another GC tip
GC Tracer

gc_tracer gem

- Helper gem to analyze GC behavior for tuning

http://www.flickr.com/photos/nasa_goddard/5188180370

How to use tuning parameters?

1. Profile your application
2. Try GC parameters (environment variables)

Profile memory management GC.stat (MRI specific)

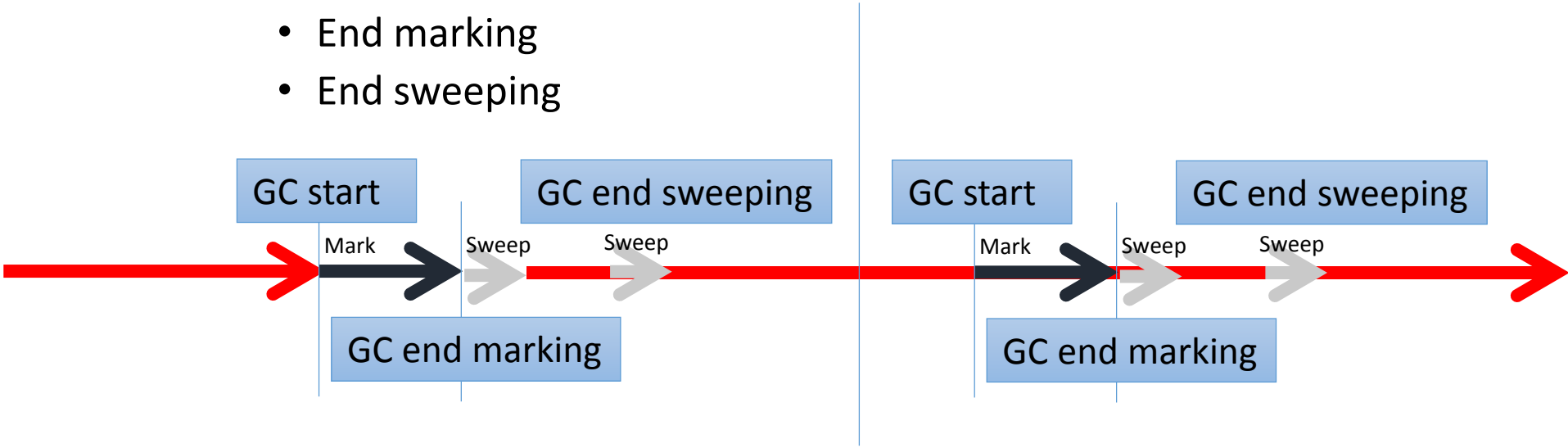
- “GC.stat” returns statistics information about GC
  - Counts
    - :count=>2, # GC count
    - :minor_gc_count=>2, # minor GC count
    - :major_gc_count=>0, # major GC count
  - Current slot information
    - :heap_live_slot=>6836, #=> # of live objects
    - :heap_free_slot=>519, #=> # of freed objects
    - :heap_final_slot=>0, #=> # of waiting finalizer objects
    - total_slots = heap_live_slot + heap_free_slot + heap_final_slot
  - Statistics
    - :total_allocated_object=>7674, # total allocated objects
    - :total_freed_object=>838, # total freed objects
    - Current living objects = total_allocated_object - total_freed_object

Profile memory management GC.latest_gc_info (MRI specific)

- “GC.latest_gc_info” returns details of latest GC
  - **:gc_by=>:newobj** **# why GC invoked?**
    - newobj: no slots available
    - malloc: malloc_increase > malloc_limit
  - :major_by=>nil # why major GC invoked?
  - :have_finalizer=>false # have finalizer?
  - :immediate_sweep=>false # immediate sweep?

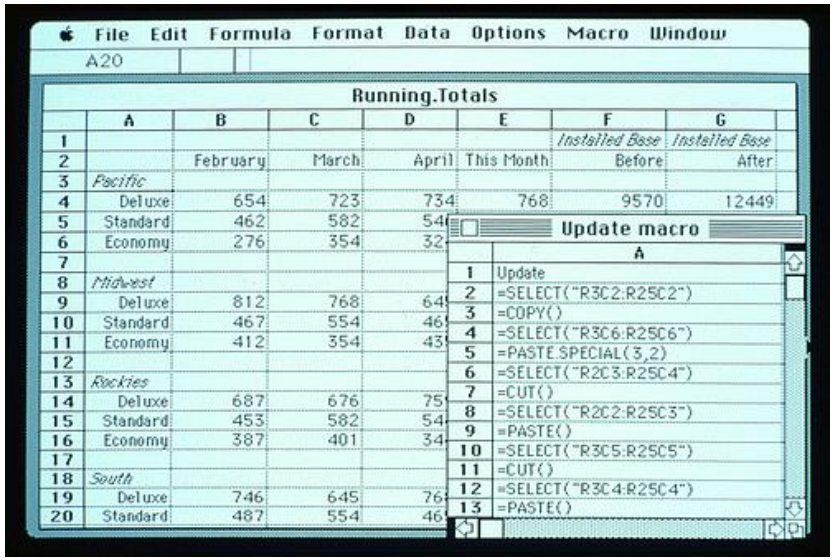
Profile memory management “gc_tracer” gem (MRI 2.1.0 later!!)

- GC::Tracer.start_logging(filename)
  - Save all GC.stat/GC.latest_gc_info results at every GC events into specified file
  - GC events:
    - Start
    - End marking
    - End sweeping



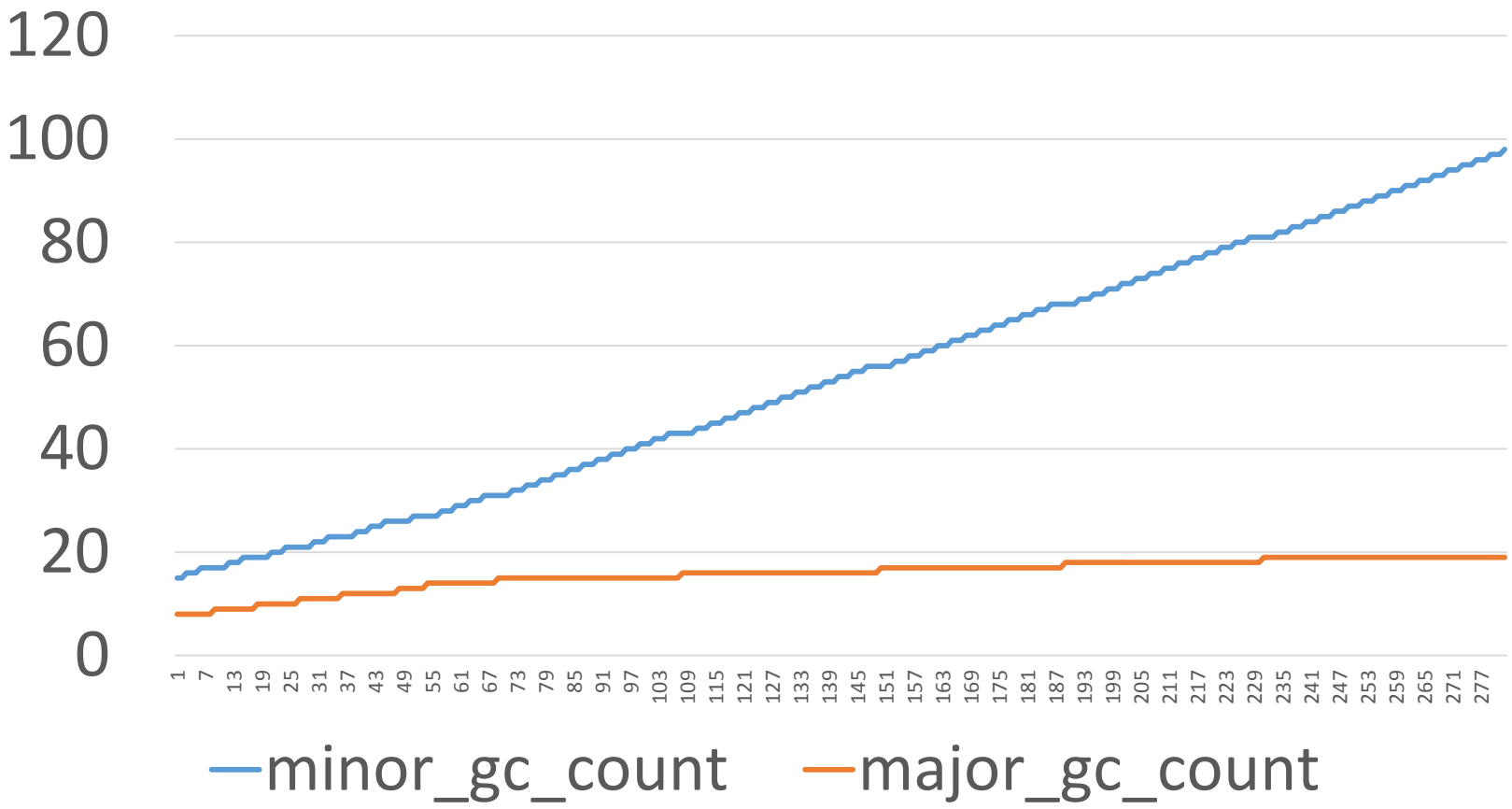
Profile memory management “gc_tracer” gem

- Run your application with gc_tracer
- Plot with Excel!

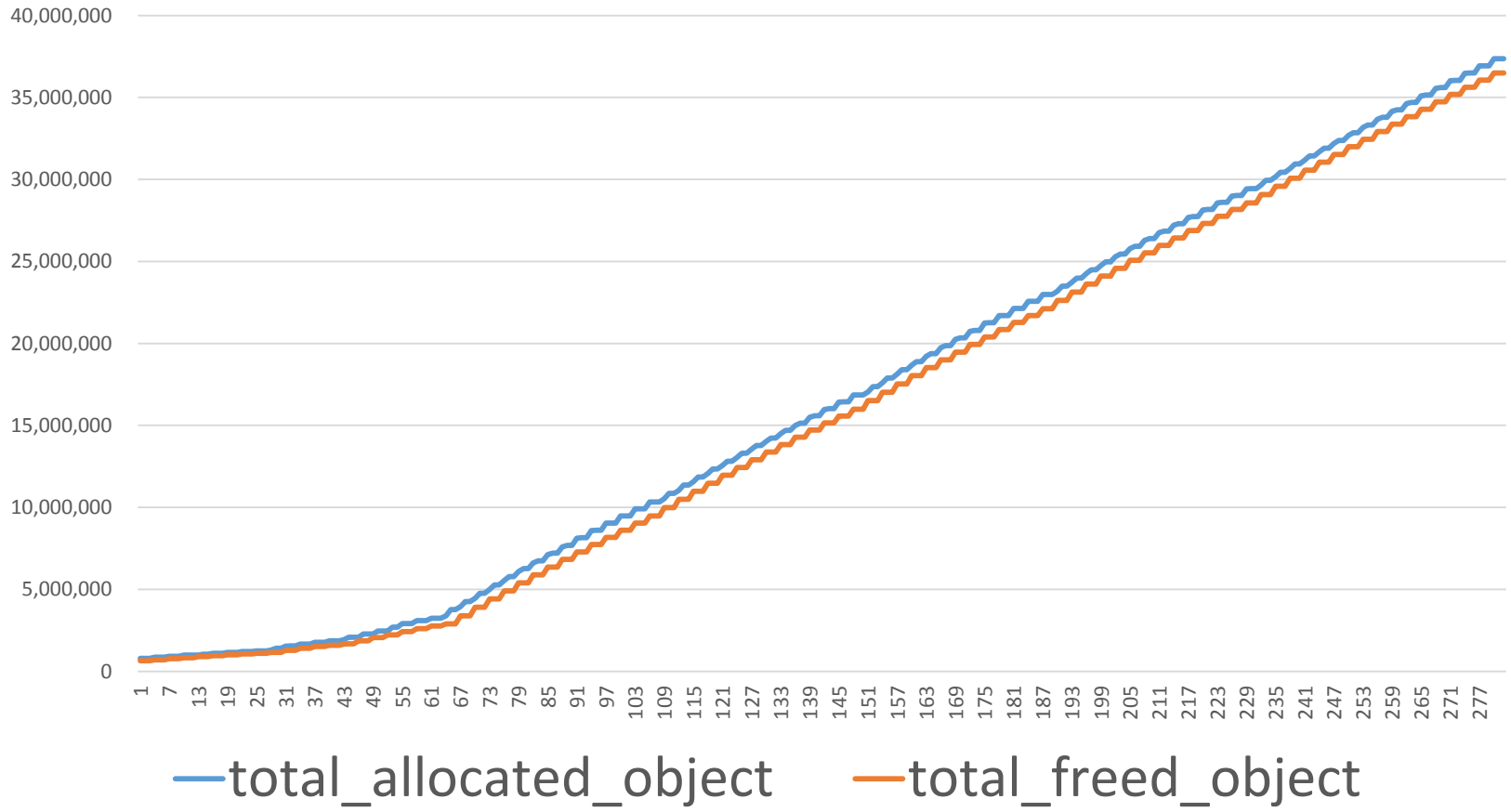


http://www.flickr.com/photos/microsoftsweden/5394685465

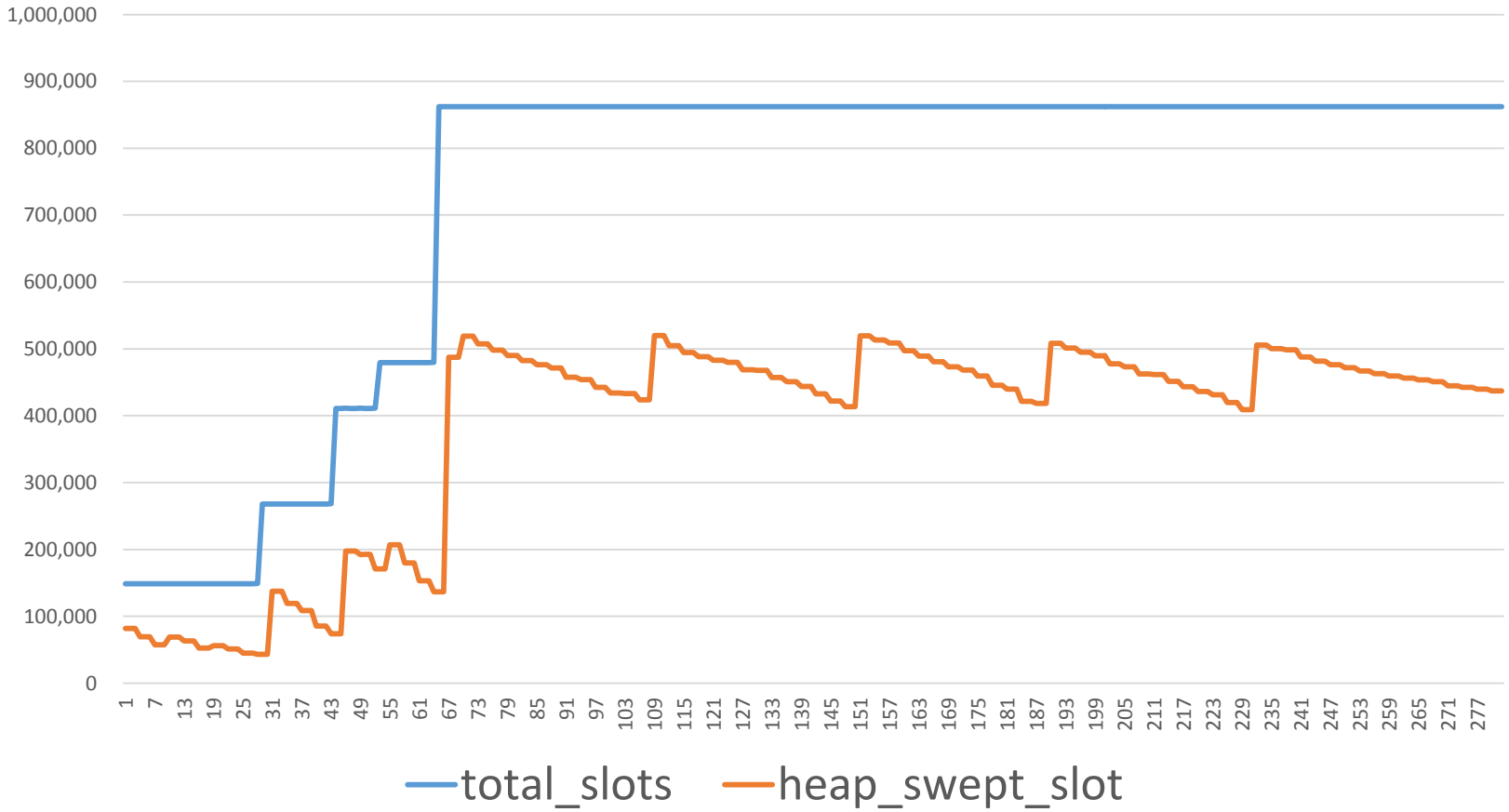
Profile memory management “gc_tracer” gem



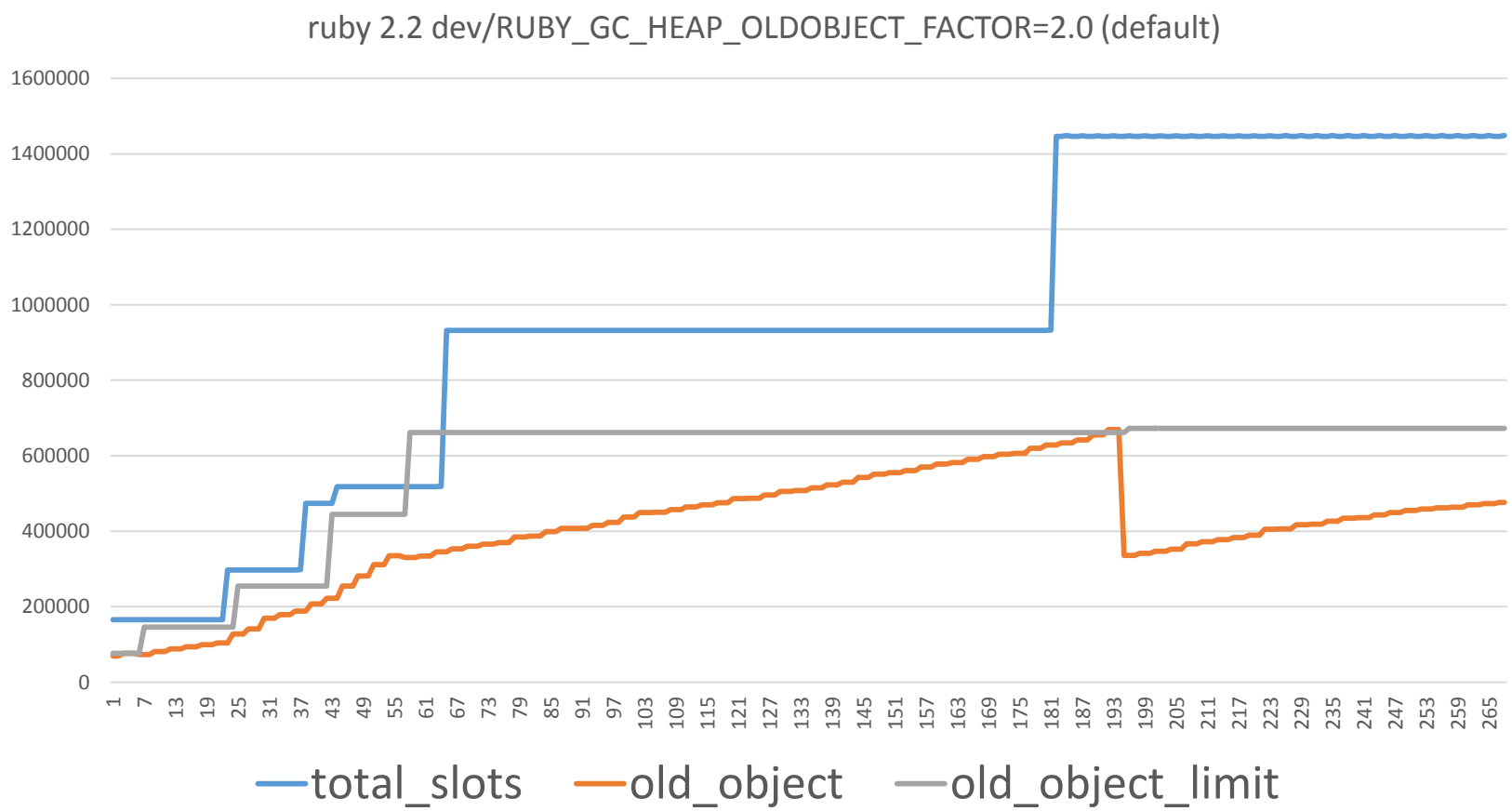
Profile memory management “gc_tracer” gem



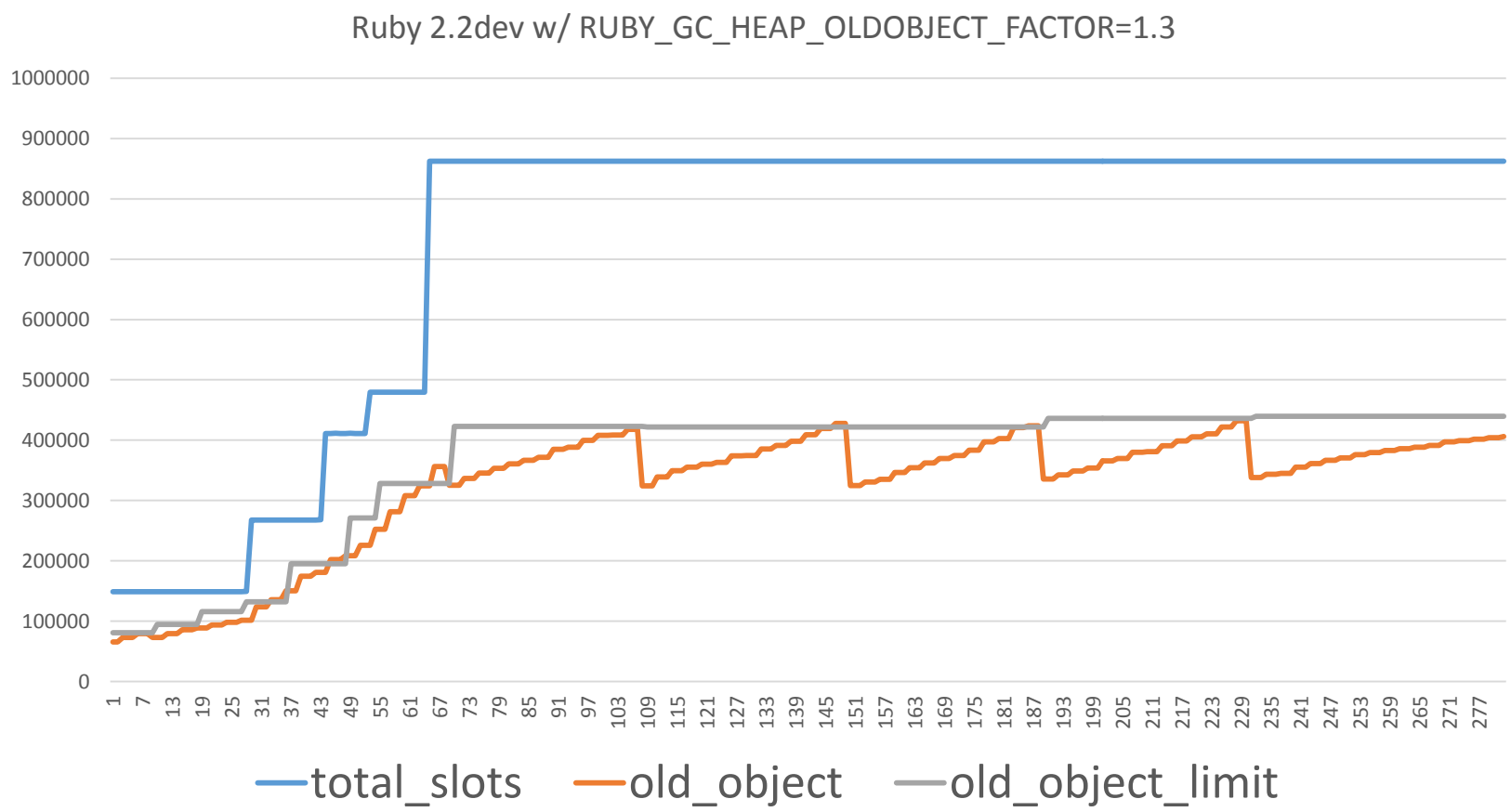
Profile memory management “gc_tracer” gem



Profile memory management “gc_tracer” gem



Profile memory management “gc_tracer” gem



For more information

Please refer my slide at RubyConf PH 2014

Summary of this talk

- Ruby 2.1 was released and 2.2 will be released this year
- Object lifetime analysis is important for program tuning
- Allocation Tracer gem helps how applications allocate objects
- Another GC Tips
  - GC Tracer gem helps GC behavior analysis for GC tuning

Thank you for your attention
Q&A?

Koichi Sasada

<ko1@heroku.com>

heroku