What's happening in your Rails app?

Introduction to

Introspection features of Ruby

Koichi Sasada

ko1@heroku.com

Short summary of this presentation

What's happening in your Rails app?

heroku

GOOGLE IT

- You can use many existing tools to inspect your Rails application.

MAKE IT

- Ruby/MRI provides many low-level features to make tools.

heroku

Koichi Sasada is a Programmer

- MRI committer since 2007/01
  - Original YARV developer since 2004/01
    - YARV: Yet Another RubyVM
    - Introduced into Ruby (MRI) 1.9.0 and later
  - Introduce generational/incremental GC
- Not a Rails programmer ☹
  - C language programmer
  - My wife is a professional Rails dev ☺

heroku

Koichi is an Employee

Heroku: A Year in Review

Sponsor session

2:40PM-3:20PM@204H

heroku

Koichi is a member of Heroku Matz team

Mission

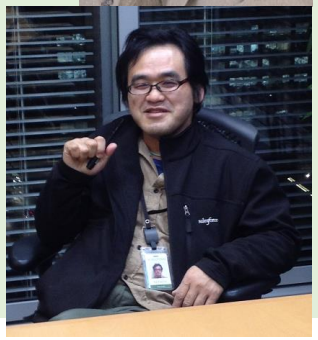
Design Ruby language
and improve quality of MRI

Heroku employs three full time Ruby core developers in Japan named “Matz team”

Heroku Matz team

Matz 

Designer/director of Ruby

Nobu 

Quite active committer

Ko1 

Internal Hacker

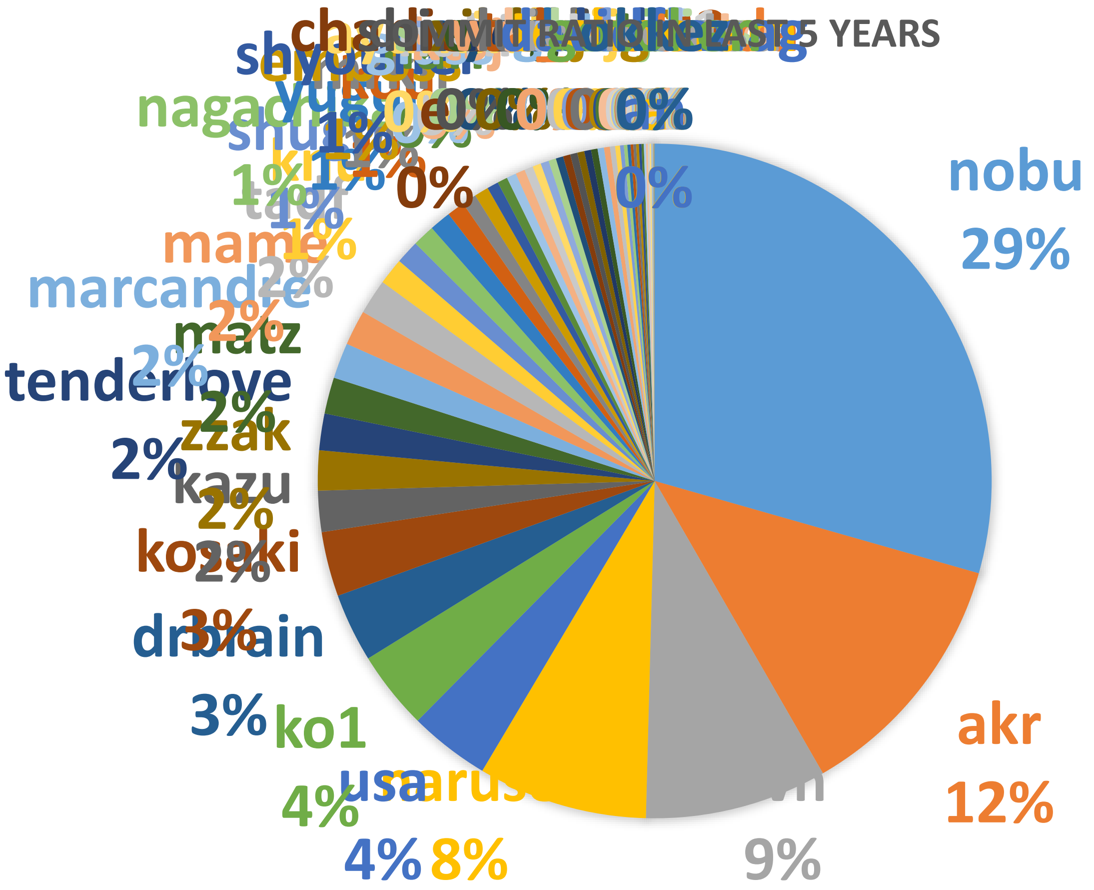
Matz
Title collector

- 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

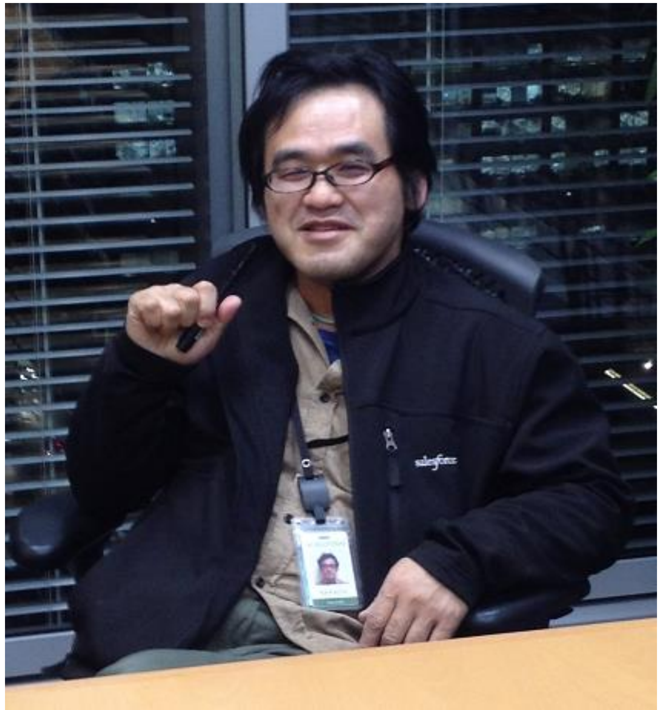
Nobu
Great Patch monster

Ruby's bug
|> Fix Ruby
|> Break Ruby
|> And Fix Ruby

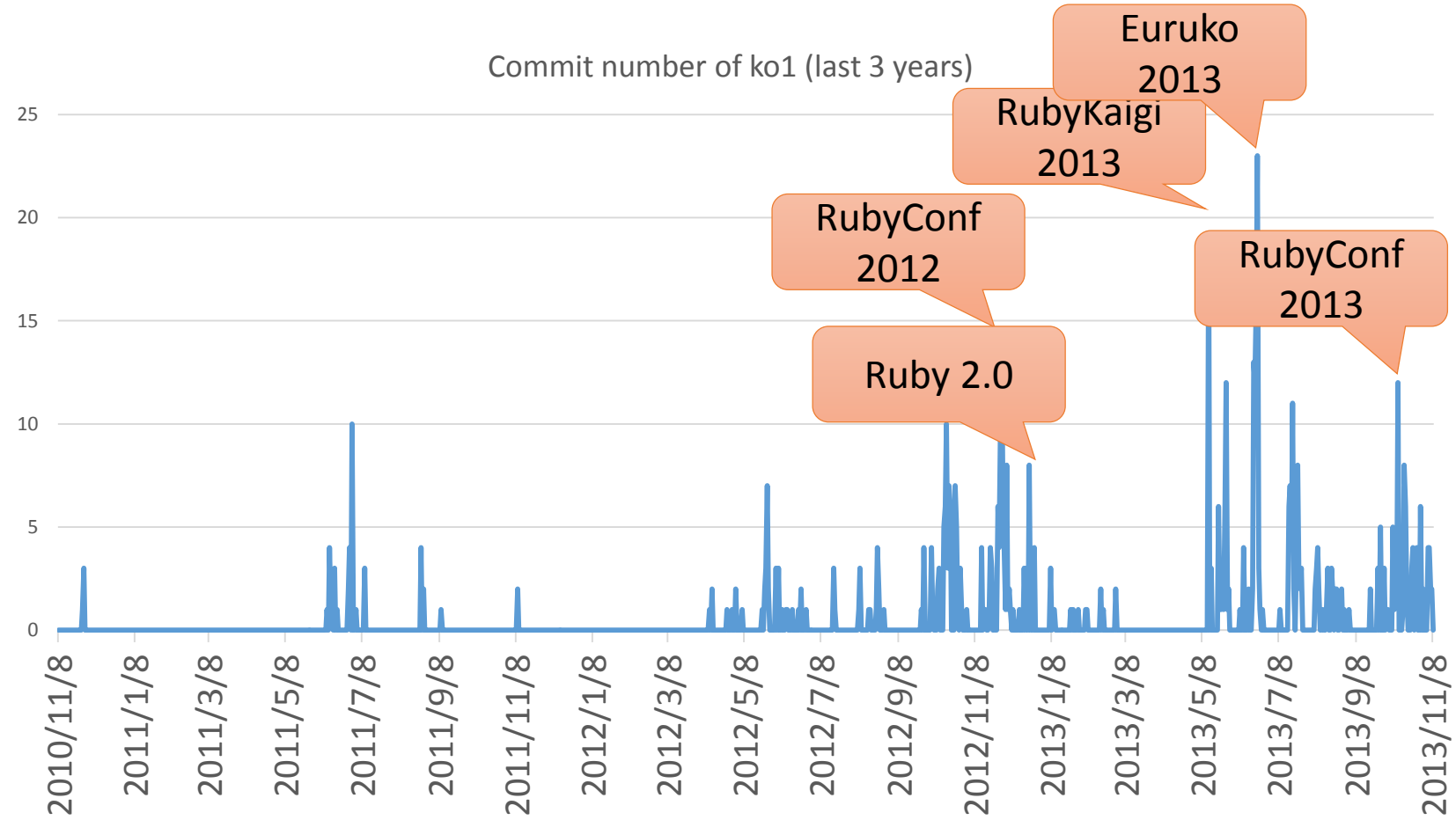
Nobu
Patch monster



Commit count of MRI



Ko1
EDD developer



EDD: Event Driven Development

Heroku Matz team and Ruby core team
Recent achievement

Ruby 2.2

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

Current stable

Ruby 2.2
Syntax

heroku

- Symbol key of Hash literal can be quoted

```
{“foo-bar”: baz}
  #=> {:“foo-bar” => baz}
  #=> not {“foo-bar” => baz} like JSON
```

TRAP!!

Easy to misunderstand

(I wrote a wrong code, already…)

Ruby 2.2
Classes and Methods

heroku

- Some methods are introduces
  - Kernel#itself
  - String#unicode_normalize
  - Method#curry
  - Binding#receiver
  - Enumerable#slice_after, slice_before
  - File.birthtime
  - Etc.nprocessors
  - …

Ruby 2.2 Improvements

heroku

- Improve GC
  - Symbol GC
  - Incremental GC
  - Improved promotion algorithm
    - Young objects promote after 4 GCs
- Fast keyword parameters
- Use frozen string literals if possible

Ruby 2.2
Symbol GC

heroku

```
before = Symbol.all_symbols.size
1_000_000.times{|i| i.to_s.to_sym} # Make 1M symbols
after = Symbol.all_symbols.size; p [before, after]

# Ruby 2.1
#=> [2_378, 1_002_378] # not GCed ☹

# Ruby 2.2
#=> [2_456, 2_456] # GCed! ☺
```

Ruby 2.2
Symbol GC (cont.)

heroku

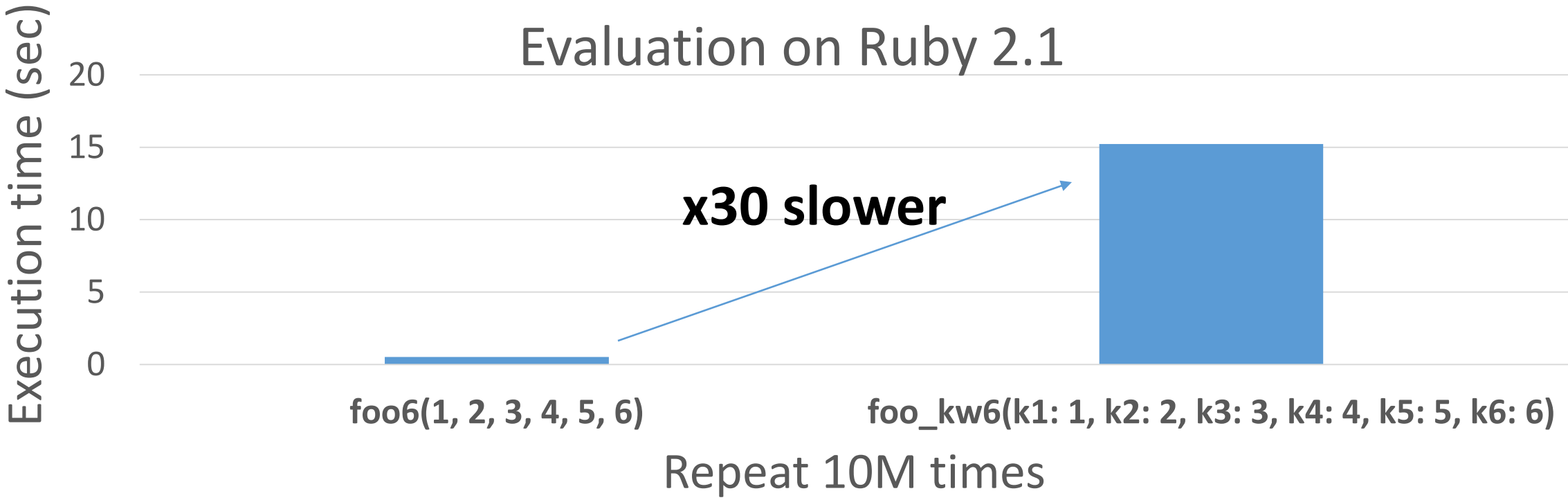
TRAP!!

Ruby 2.2.0 has memory leak error!

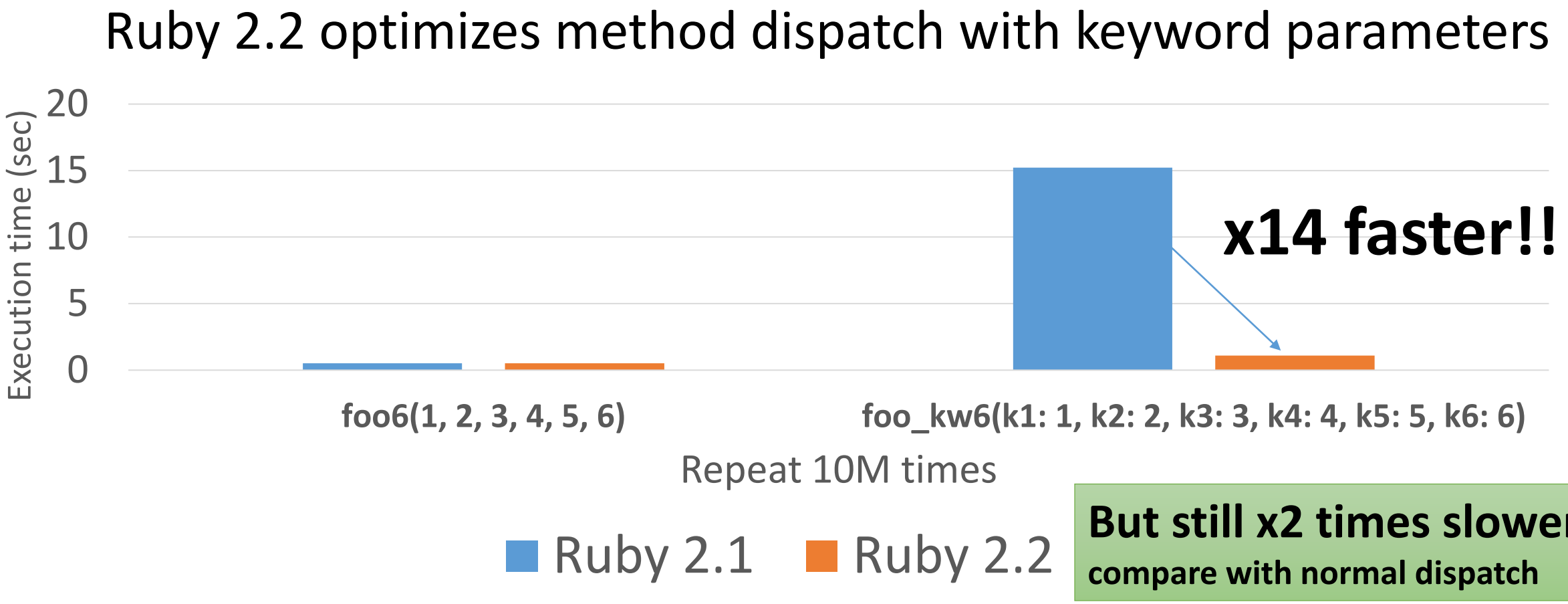
- Upgrade Ruby 2.2.2
  - Memory (object) leak problem
    - Symbols has corresponding String objects
    - Symbols are collected, but Strings are not collected! (leak)
  - Ruby 2.2.1 solved this problem!!
    - However, 2.2.1 also has problem (rarely you encounter BUG at **the end of process** **[Bug #10933]** ← not big issue, I want to believe)
  - Finally Ruby 2.2.2 had solved it!!

Ruby 2.2
Fast keyword parameters

“Keyword parameters” introduced in Ruby 2.0 is useful, but slow!!



Ruby 2.2
Fast keyword parameters



But still x2 times slower
compare with normal dispatch

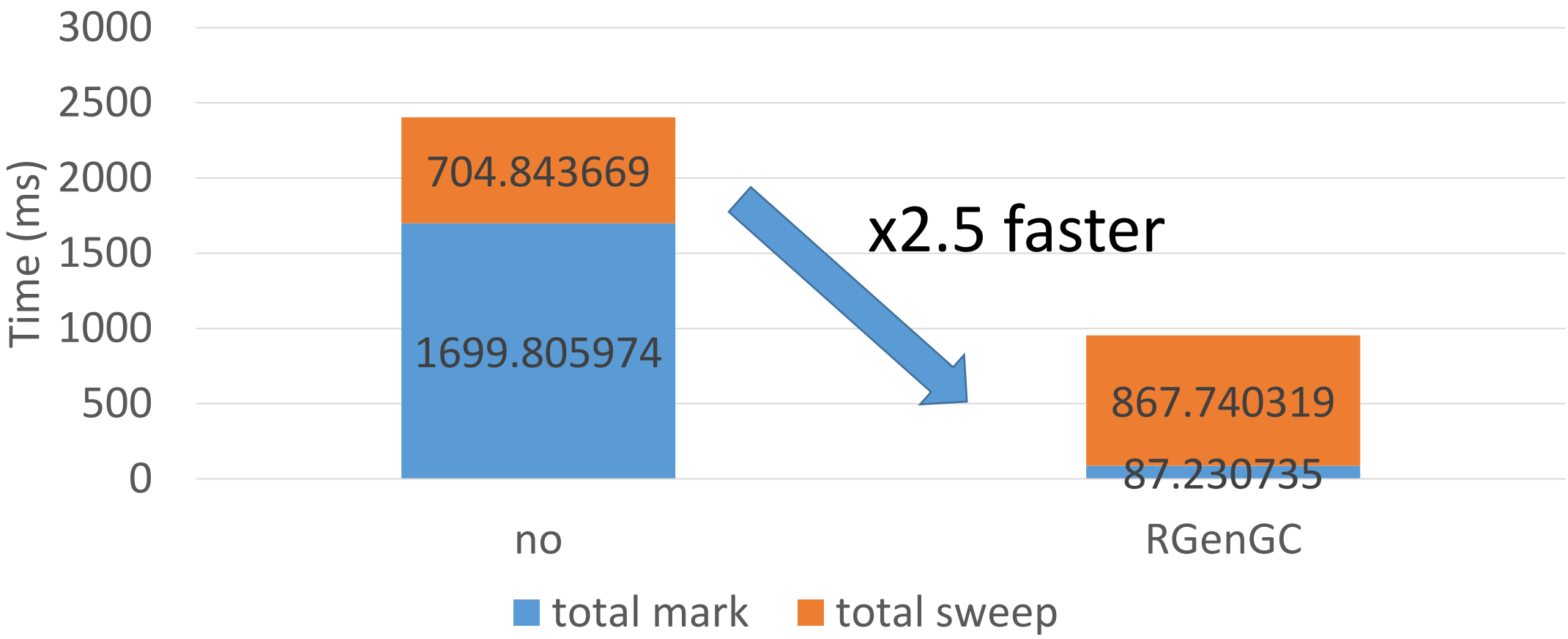
Ruby 2.2 Incremental GC

heroku

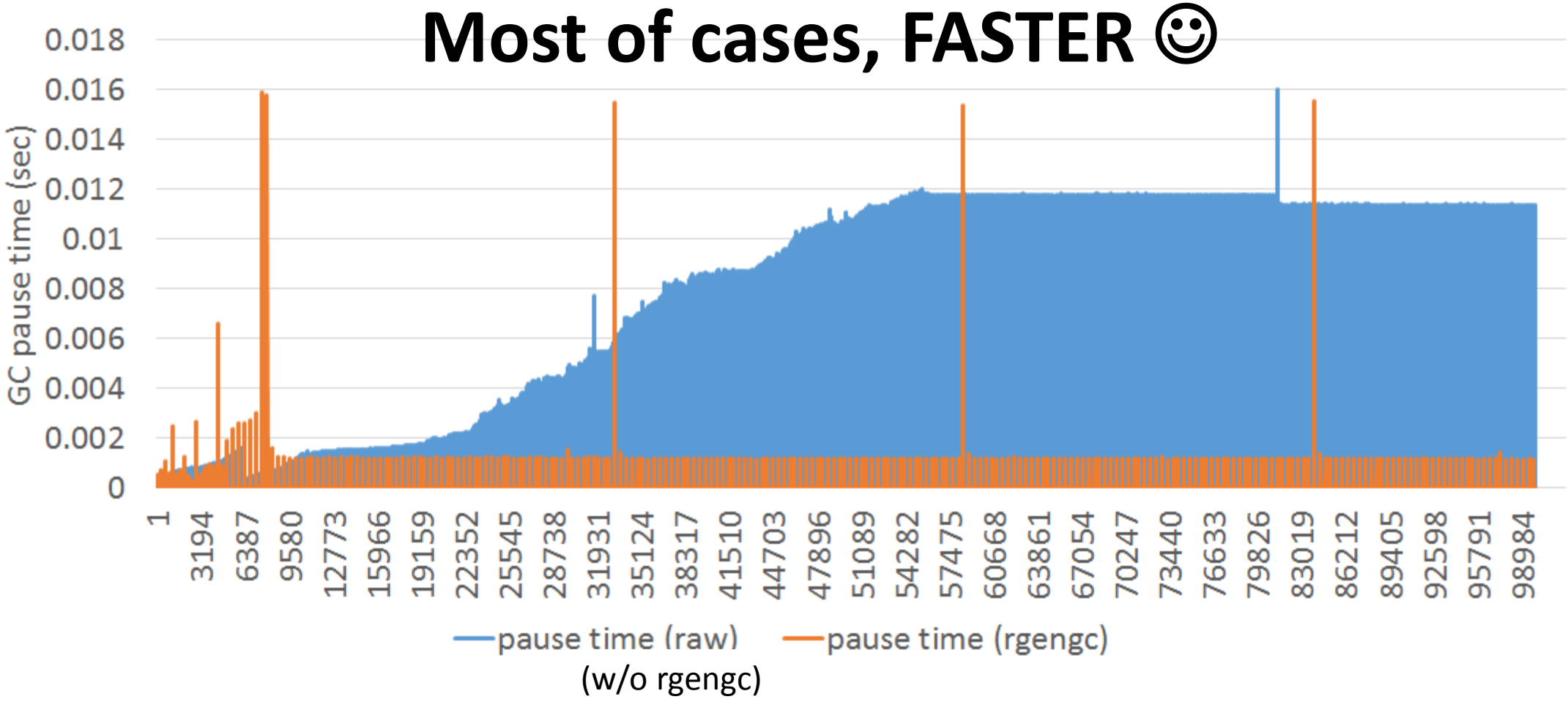
Goal

	Before Ruby 2.1	Ruby 2.1 RGenGC	Incremental GC	Ruby 2.2 Gen+IncGC
Throughput	Low	High	Low	High
Pause time	Long	Long	Short	Short

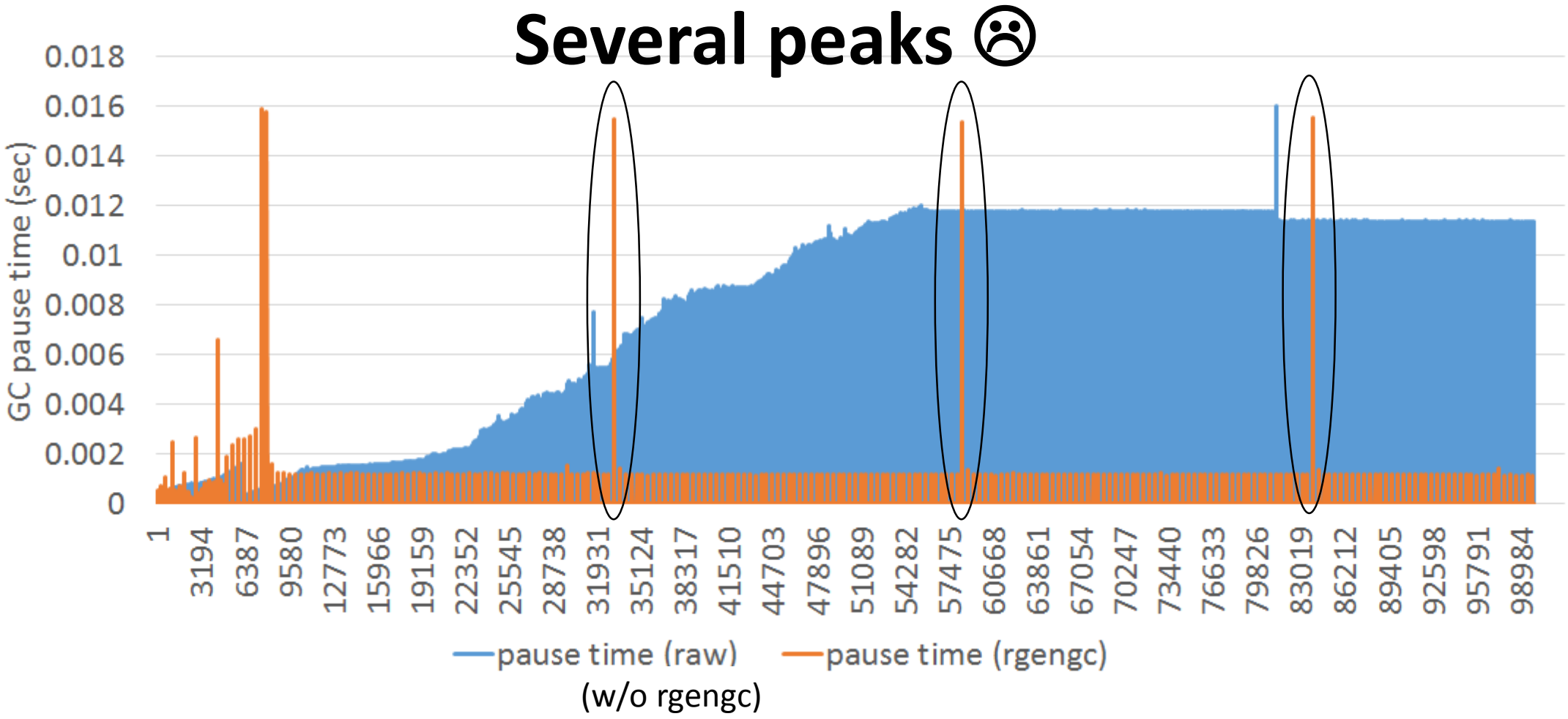
RGenGC from Ruby 2.1: Micro-benchmark



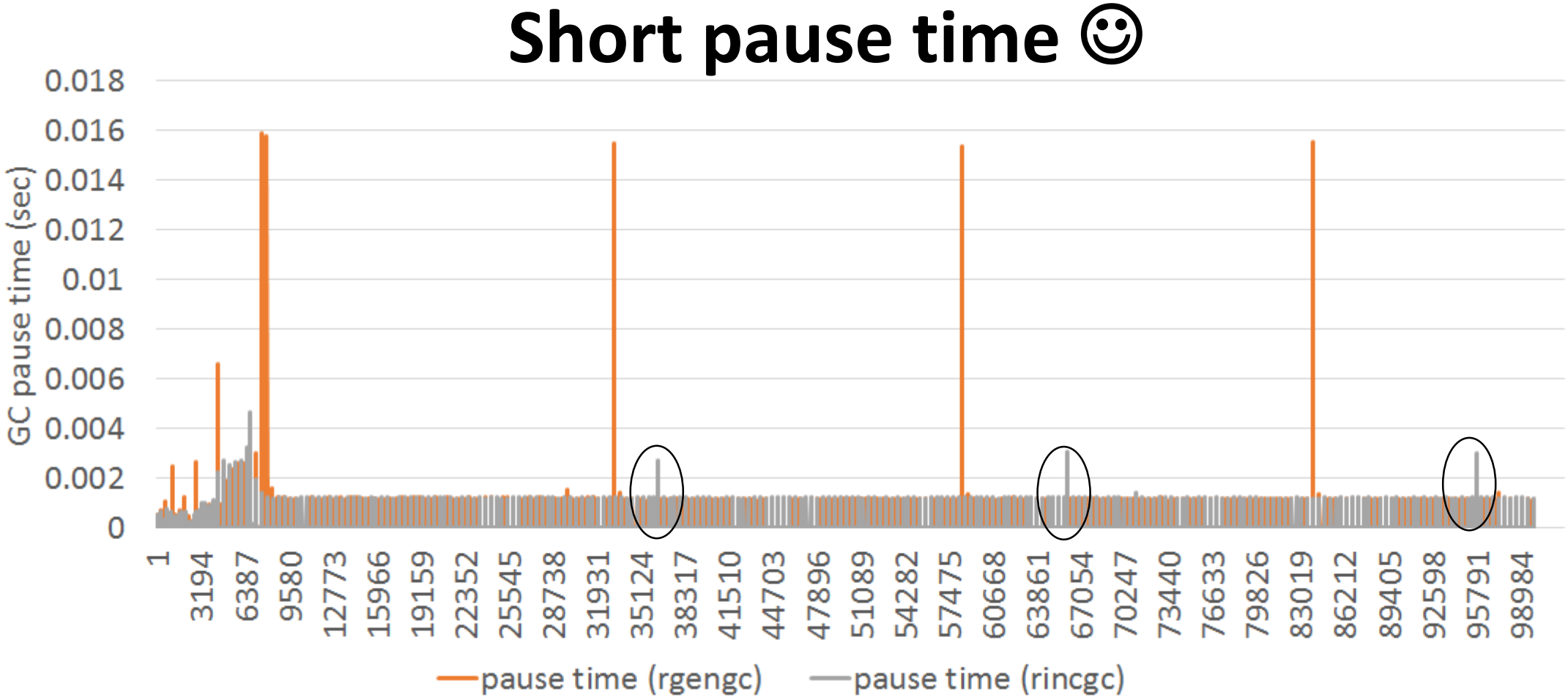
RGenGC from Ruby 2.1: Pause time



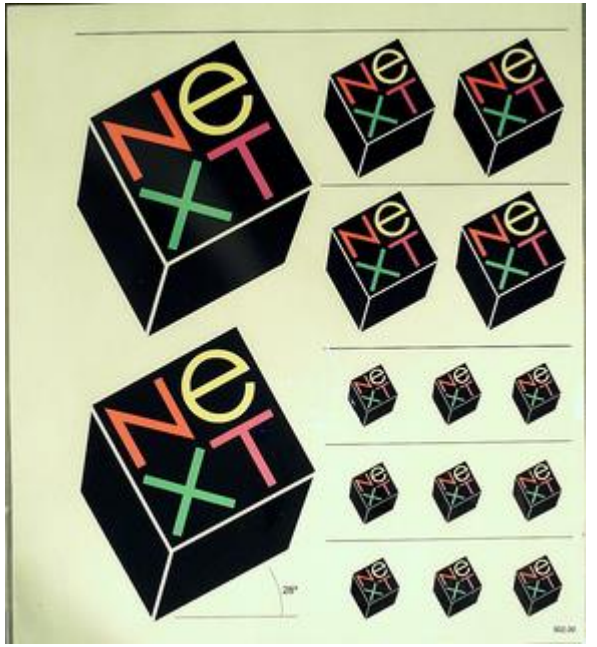
RGenGC from Ruby 2.1: Pause time



Ruby 2.2 Incremental GC



Ruby 2.3



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

Next version will be released the end of this year

Catch me and give me your feedback later

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

Break

What's happening in your Rails app?

Introduction to

Introspection features of Ruby

Today's topic
#駄洒落クラブ

#駄洒落クラブ (DAJARE Club)

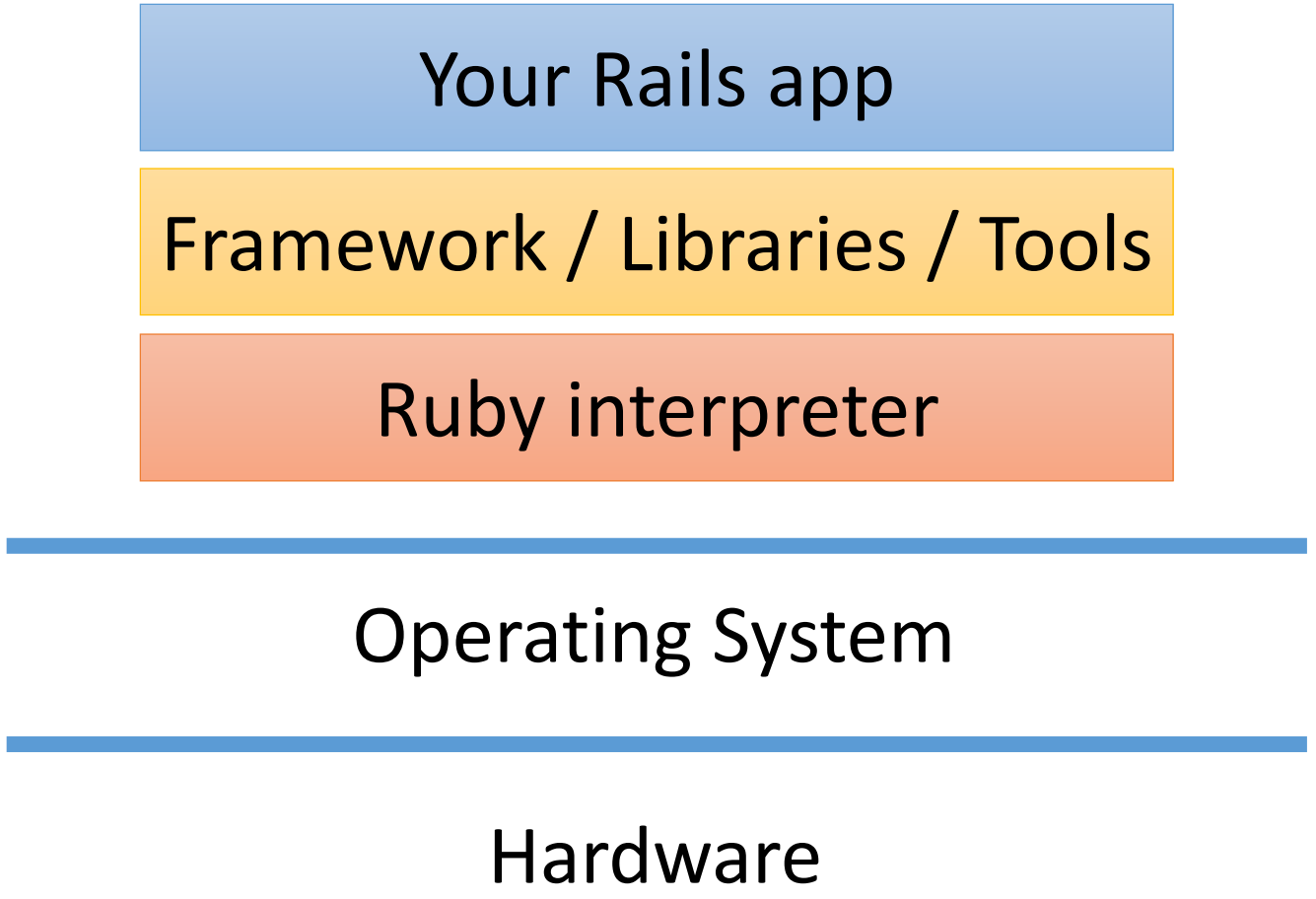
- DAJARE is something like PUN in Japanese
- #駄洒落クラブ is associated by our HERO Aaron Patterson @tenderlove
- Many (Japanese) Rubyists have joined
- Recommendation: Join our club if you are learning Japanese

#駄洒落クラブ (DAJARE Club)

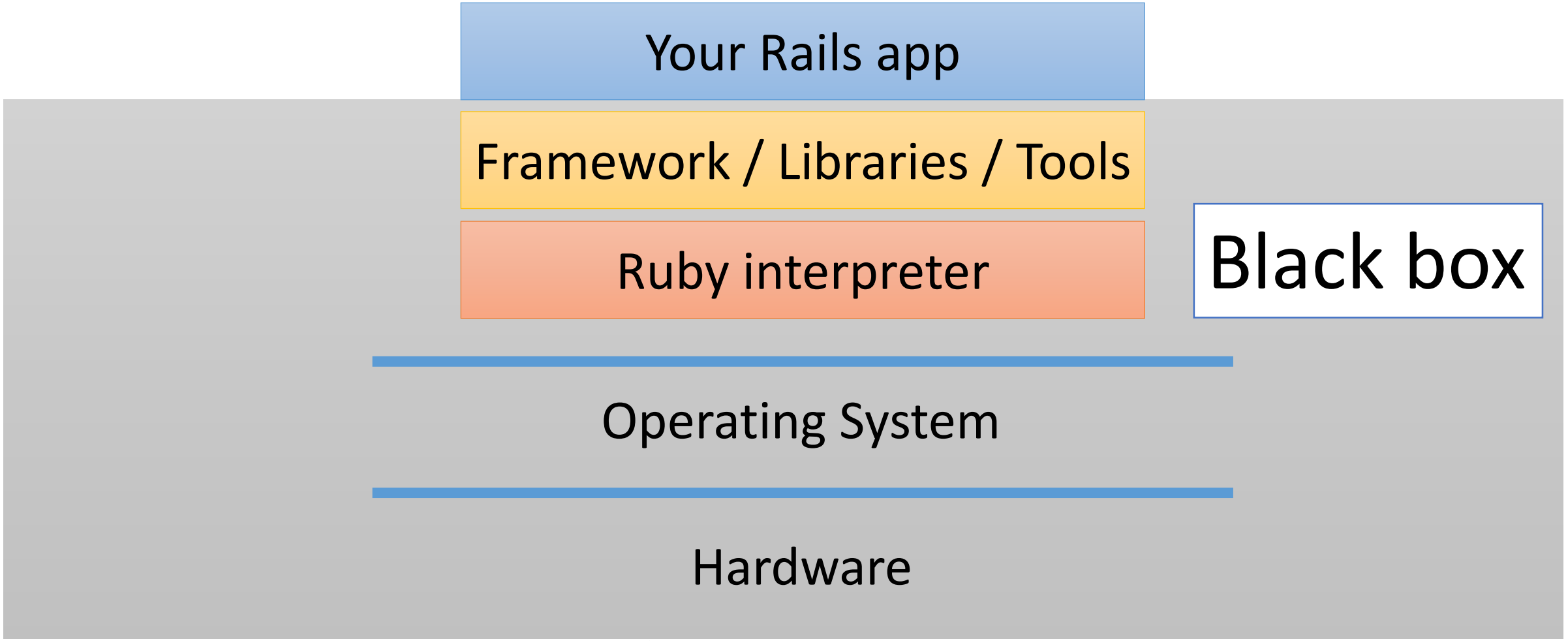
Introspection features of Ruby

for MRI

Launch your application on the Layered computer structure

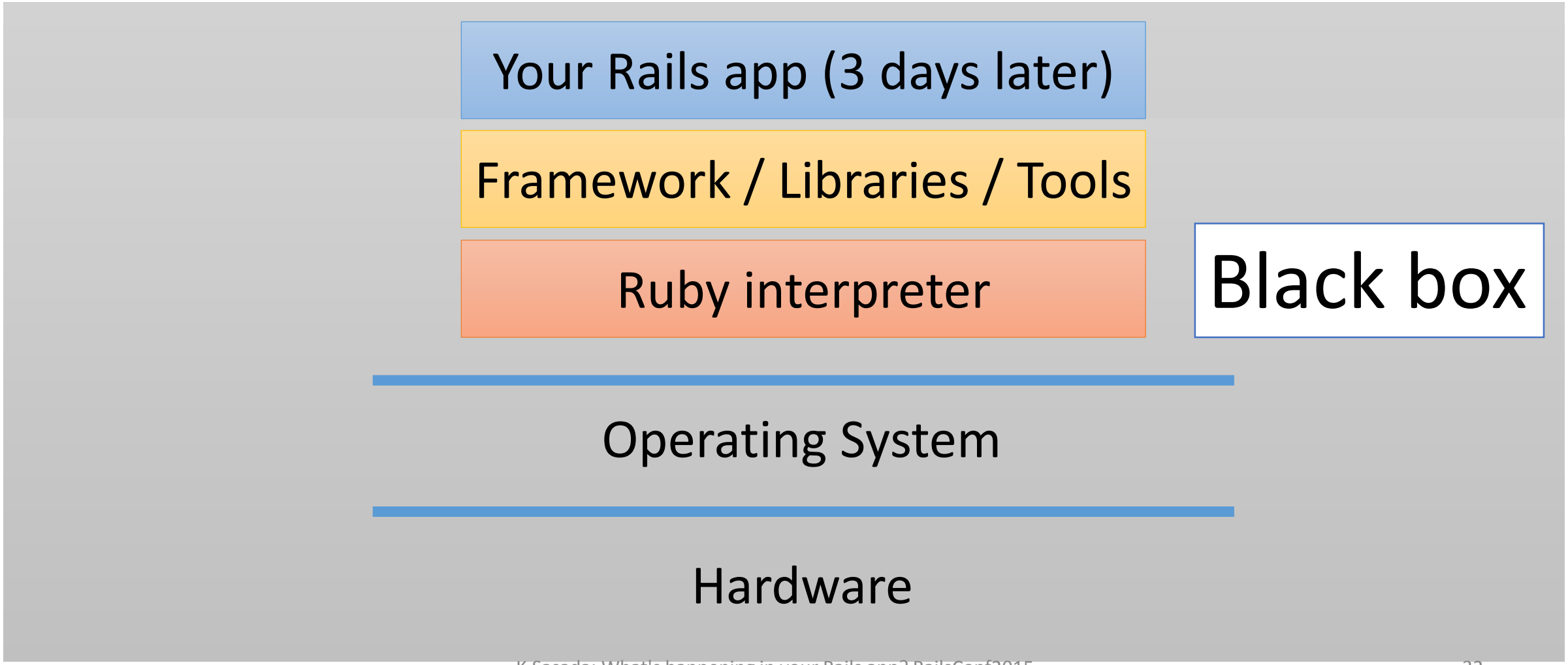


Launch your application on the Layered computer structure



Launch your application on the Layered computer structure

heroku



Trouble will come from a black box

https://www.flickr.com/photos/wingedwolf/5471047557/

Performance issue

- Slow requests…
- Memory consuming…

Unknown behavior

- Unexpected values…
- Difficult debugging…

Layered computer structure
How to inspect your Rails application?

heroku

Your Rails app

Framework / Libraries / Tools

Ruby interpreter

Operating System

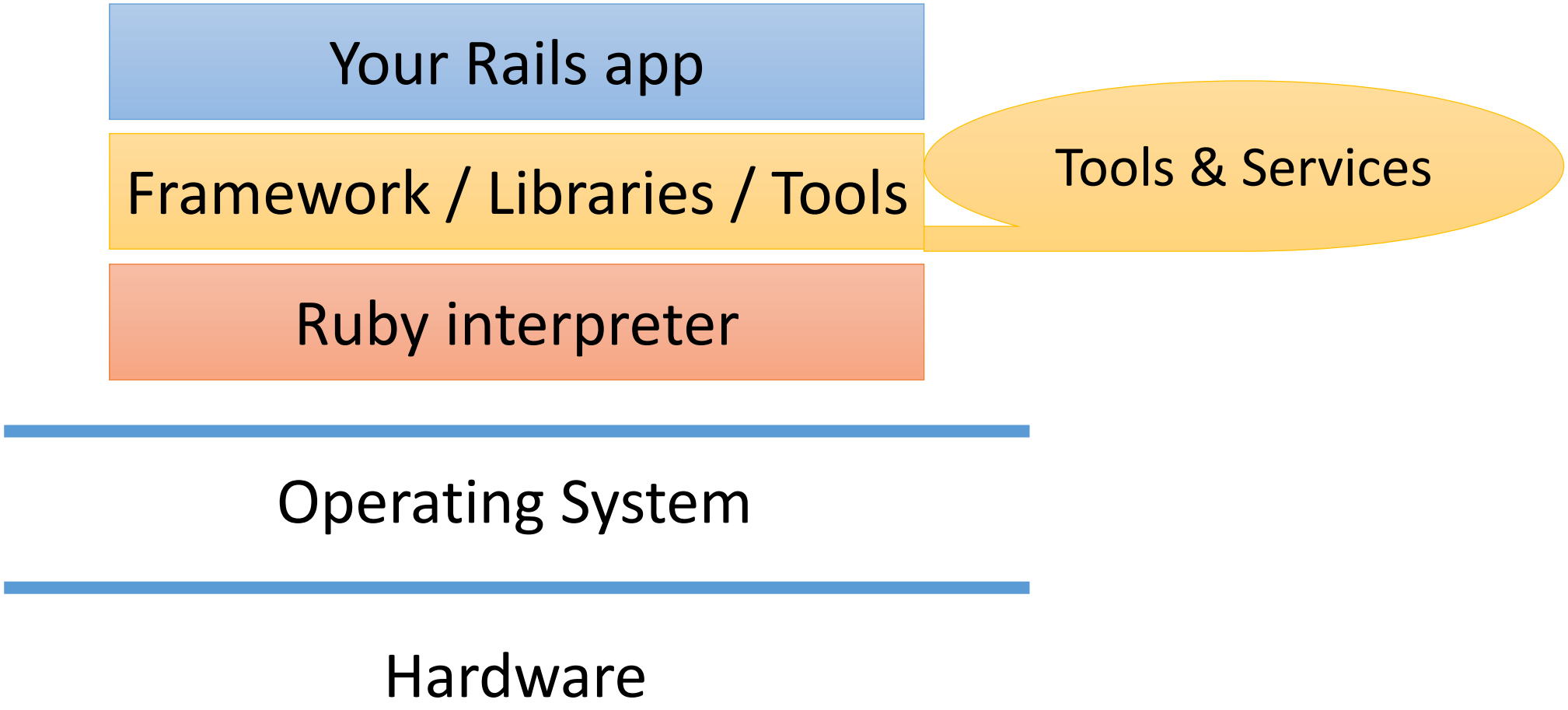
Hardware

heroku

1. Use existing tools
2. Make your own tools

heroku

1. Use existing tools



heroku

Great presentations at RailsConf2015

- “Prying Open The Black Box” by Godfrey Chan
- “Speed Science” by Richard Schneeman
- Keynote: Aaron Patterson
- …

Performance issue

Slow requests…

Memory consuming …

heroku

The easiest way to solve performance issue is...

Use PX: Performance dyno on Heroku

(or another high-performance machine)

Dyno Size	Memory (RAM)	CPU Share	Multitenant	Compute	Price/dyno-hour
1X	512MB	1x	yes	1x-4x	$0.05
2X	1024MB	2x	yes	4x-8x	$0.10
PX	6GB	100%	no	40x	$0.80

https://devcenter.heroku.com/articles/dyno-size

https://blog.heroku.com/archives/2014/2/3/heroku-xl

Performance issue

We need to know what happen in a black box

heroku

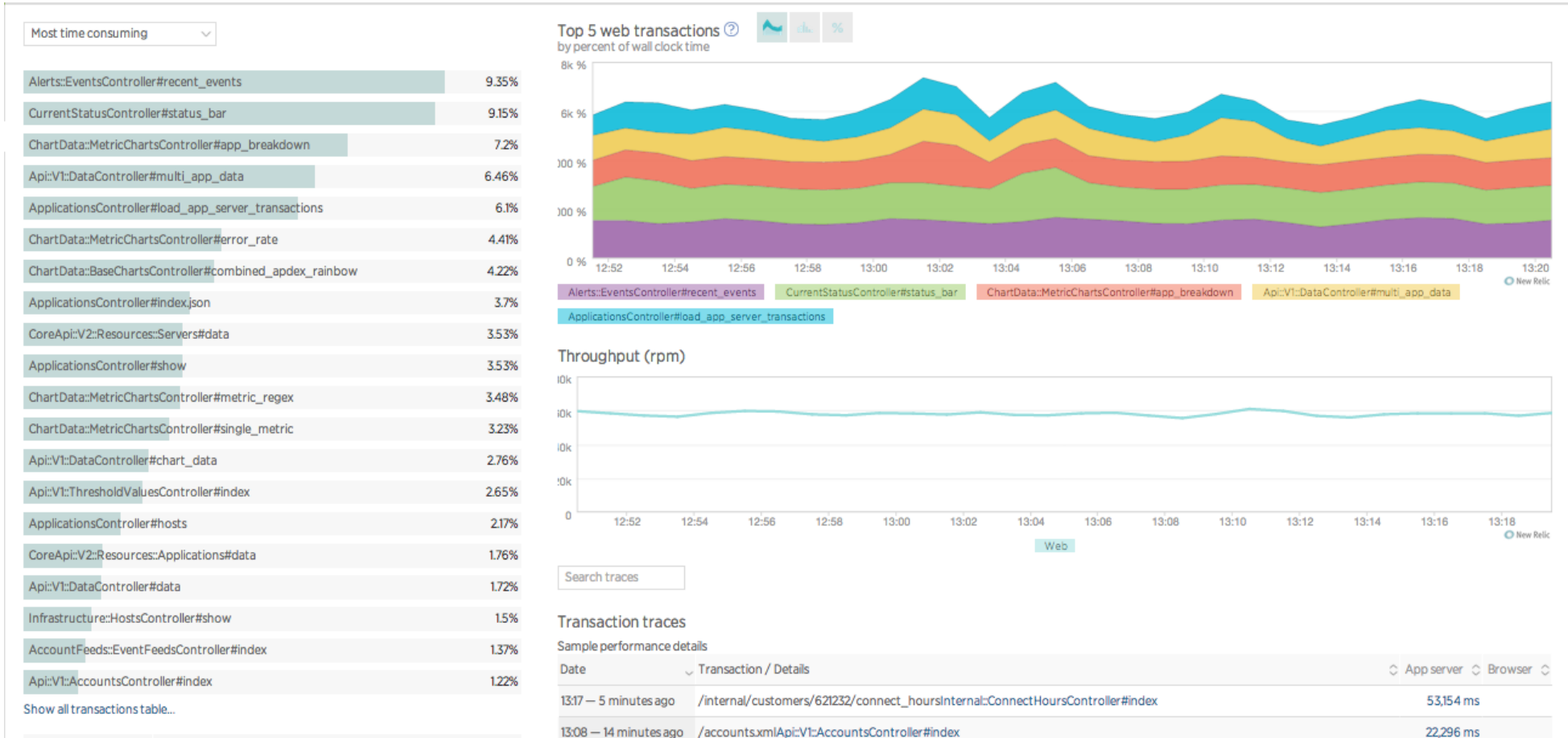
Slow requests

- Which part is slow?
  - DB?
  - External API access?
  - Your Ruby app?
    - Application?
    - GC?

Memory consuming

- Who is consuming memory?
- Correct GC Parameter?

Analyze slow requests
Use New Relic



http://newrelic.com/application-monitoring/features

Analyze slow requests
Use New Relic

heroku

- **“Dive into Ruby VM Stats with New Relic”**
  http://blog.newrelic.com/2014/04/23/ruby-vm-stats/
- **“Ruby VM measurements”**
  https://docs.newrelic.com/docs/ruby/ruby-vm-stats

IMPORTANT

You can use New Relic very easily on Heroku as an Add-on

Analyze slow requests
Performance profilers

heroku

“Debugging Ruby Performance” by Aman Gupta

- https://speakerdeck.com/tmm1/debugging-ruby-performance

“Ruby 2.1 in Production” by Aman Gupta

- http://rubykaigi.org/2014/presentation/S-AmanGupta

Analyze memory consumption issues Background

heroku

- Ruby has GC
  - Automatic object recycling and memory management
  - Ruby 2.2 has “Incremental and generational” GC
    - https://engineering.heroku.com/blogs/2015-02-04-incremental-gc
- Issues
  - Incorrect GC parameters (environment variables)
  - Object leaking
  - MRI bugs

Analyze memory consumption issues Background

heroku

- Generational GC (from Ruby 2.1)
  - Collect only newer objects
  - Old objects can remain long time unexpectedly

→ Object leaking

Analyze memory consumption issues Tools (pickup my original two)

heroku

Measure GC statistics

- gc_tracer.gem

Find object creation locations

- allocation_tracer.gem

Analyze memory consumption issues GC Tracer in your app

heroku

- Require and call “GC::Tracer.start_logging(filename)”
  - You can specify logging filename by env val “GC_TRACER_LOGFILE”
- All GC related information are recorded into “filename” (or STDERR)

Analyze memory consumption issues Allocation Tracer in your app

 # Source code

```
require 'allocation_tracer'
require 'pp'

ObjectSpace::AllocationTracer.setup(%i{path line class})
pp ObjectSpace::AllocationTracer.trace{
  50_000.times{|i|
    i.to_s
    i.to_s
    i.to_s
  }}
```

 # result

```
{["t.rb", 7, String]=>[50000, 1, 40960, 0, 6, 1745520],
 ["t.rb", 8, String]=>[50000, 0, 40950, 0, 1, 1745560],
 ["t.rb", 9, String]=>[50000, 1, 40963, 0, 6, 1745480]}
```

GC/Allocation tracer
Use as Rack middleware

heroku

You only need to write the following lines

```
# Sample in config.ru

# GC tracer
require 'rack/gc_tracer'
use Rack::GCTracerMiddleware, view_page_path: '/gc_tracer', filename: '/tmp/rails-gc_tracer'

# Allocation Tracer
require 'rack/allocation_tracer'
use Rack::AllocationTracerMiddleware
```

Warning: You should not use Allocation Tracer in production because it has big overhead

arigatobook

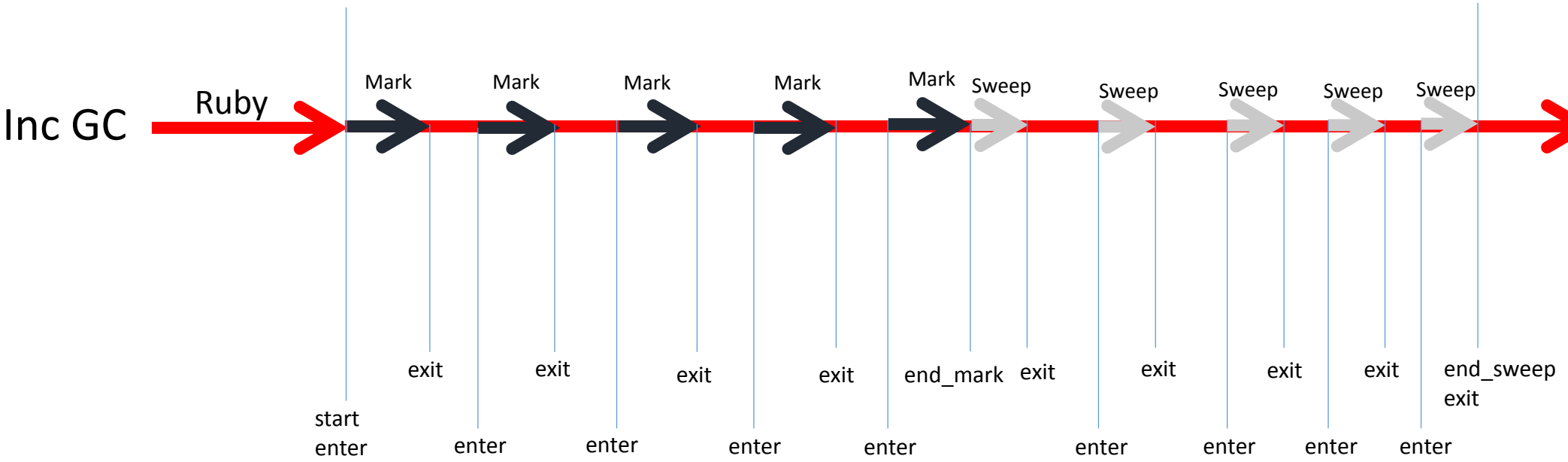
heroku

Demo App

http://atdot.net/ab/

GC Tracer Events

- Capture information on every GC related events
  - Start marking, start sweeping, end sweeping (default events)
  - Enter GC, exit GC, newobj, freeobj (optional events)



GC Tracer
Captured information

heroku

- GC.stat results (statistics for current GC)
  - https://docs.google.com/spreadsheets/d/11Ua4uBr6o0k-nORrZLEIIUkHJ9JRzRR0NyZfrhEEnc8/edit?usp=sharing
- GC. latest_gc_info
- Result of get_gusage (on supported OSs)
- Custom fields
  - You can add any numbers
  - “Access number” field on Rack mode

GC Tracer
How to see the log

- “Debugging memory leaks in Ruby” by Sam Saffron
  - http://samsaffron.com/archive/2015/03/31/debugging-memory-leaks-in-ruby
- Send me ko1@heroku.com your gc_tracer log

heroku

Allocation tracer

- Object allocation status for each line
  - # of created objects
  - # of old objects
  - Total ages (total ages / # of created objects = the average age)
  - Minimum / maximum ages
  - Consumed memory bytes
- # of created objects helps to find out newobj overheads
- # of old objects helps to find out unexpected long lifetime objects
- A bit slow (tracking all object creation and recycling)

Unknown behavior

heroku

Tools to reveal unknown behaviors

- Debugger
  - pry
  - Byebug
  - …
- Error messages
  - better_errors
  - did_you_mean
  - pretty_backtrrace
  - …

“Did you mean?” gem

https://github.com/yuki24/did_you_mean

```
class User
  attr_accessor :first_name, :last_name
  def to_s; "#{f1rst_name} #{last_name}"; end
end

user.to_s

# => NameError: undefined local variable or method `f1rst_name' for
#<User:0x0000000928fad8>
#
#     Did you mean? #first_name
#
```

“Pretty backtrace” gem

https://github.com/ko1/pretty_backtrace

 # Source code

```
require 'pretty_backtrace/enable'
def fib n
  n < 1 ? raise : fib(n-1)
End

fib(3)
```

 # Error message

```
test.rb:5:in `fib' (n = 0): unhandled exception
        from test.rb:5:in `fib' (n = 1)
        from test.rb:5:in `fib' (n = 2)
        from test.rb:5:in `fib' (n = 3)
        from test.rb:8:in `<main>'
```

heroku

No suitable tools.

How to do it?

heroku

2. Make your own tools

Your Rails app

Framework / Libraries / Tools

Ruby interpreter

Ruby provides primitives

Operating System

Hardware

heroku

Low level Ruby APIs

- TracePoint
  - Hook events
- Exception
  - Add your favorite information to error messages
- Reflection features
- debug_inspect gem
- And more…

Easy tracking with TracePoint

Example: Where is the “#index” method called?

```
# insert this part of code into boot code
TracePoint.new(:call) do |tp|
  if tp.method_id == :index
    pp [tp, caller_locations]
  end
end.enable
```

Modify exceptions

Example: Modify raised exception backtraces with ♥

 # src

```
TracePoint.new(:raise){|tp|
    e = tp.raised_exception
    e.set_backtrace(
      e.backtrace.map{|line|
       line + "<3<3<3"})
}.enable{
  raise
}
```

 # result

```
t.rb:5:in `block in <main>'<3<3<3:
unhandled exception
        from t.rb:4:in `enable'<3<3<3
        from t.rb:4:in `<main>'<3<3<3
```

Reflection API
Example: Getting local variable name

 # src

```
def foo of: STDOUT, if: STDIN
  binding.local_variables.each{|lv|
    p [lv, binding.local_variable_get(lv)] }
  # BTW: how to get the value of “if”?
end


foo
```

 # result

```
[:of, #<IO:<STDOUT>>]
[:if, #<IO:<STDOUT>>]
```

Debug Inspect gem
Example: Getting bindings for all frames

 # source code

```
require 'debug_inspector‘ # gem
def local_variables_for_frames
  RubyVM::DebugInspector.open{|dc|
    dc.backtrace_locations.size.times{|i|
      p dc.frame_binding(i).local_variables if dc.frame_binding(i)
  }}
end

def fib n; n < 1 ? local_variables_for_frames : fib(n-1); end
fib(3)
```

 # result

```
[]
[:n]
[:n]
[:n]
[:n]
[]
```

heroku

Combination of techniques

- Examples
  - Track “raise” events
  - Modify backtrace
  - Get local variables from binding
  - Get bindings for each method frames

Combining all examples makes “Pretty Backtrace” gem

Advanced hacking

Escape from Ruby world

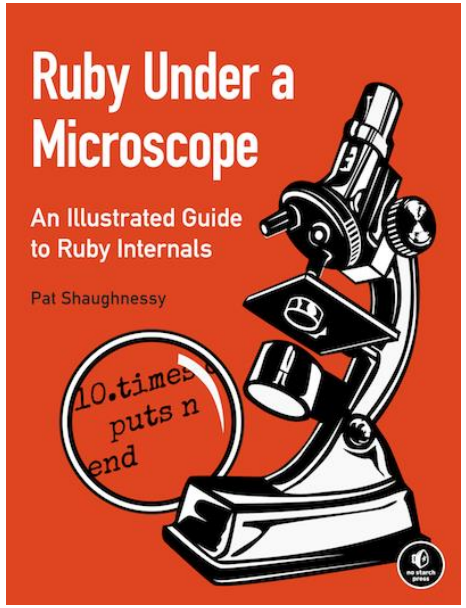
heroku

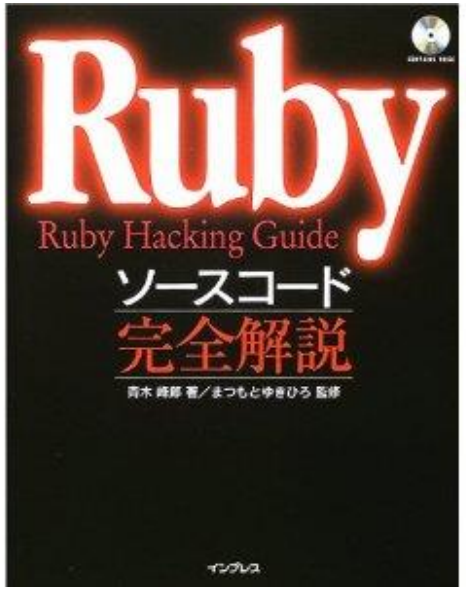
Make C extensions with low level C APIs

- See “ruby/debug.h”
  - TracePoint with internal events
  - rb_profile APIs
  - rb_debug_inspect APIs
  - GC inspection APIs

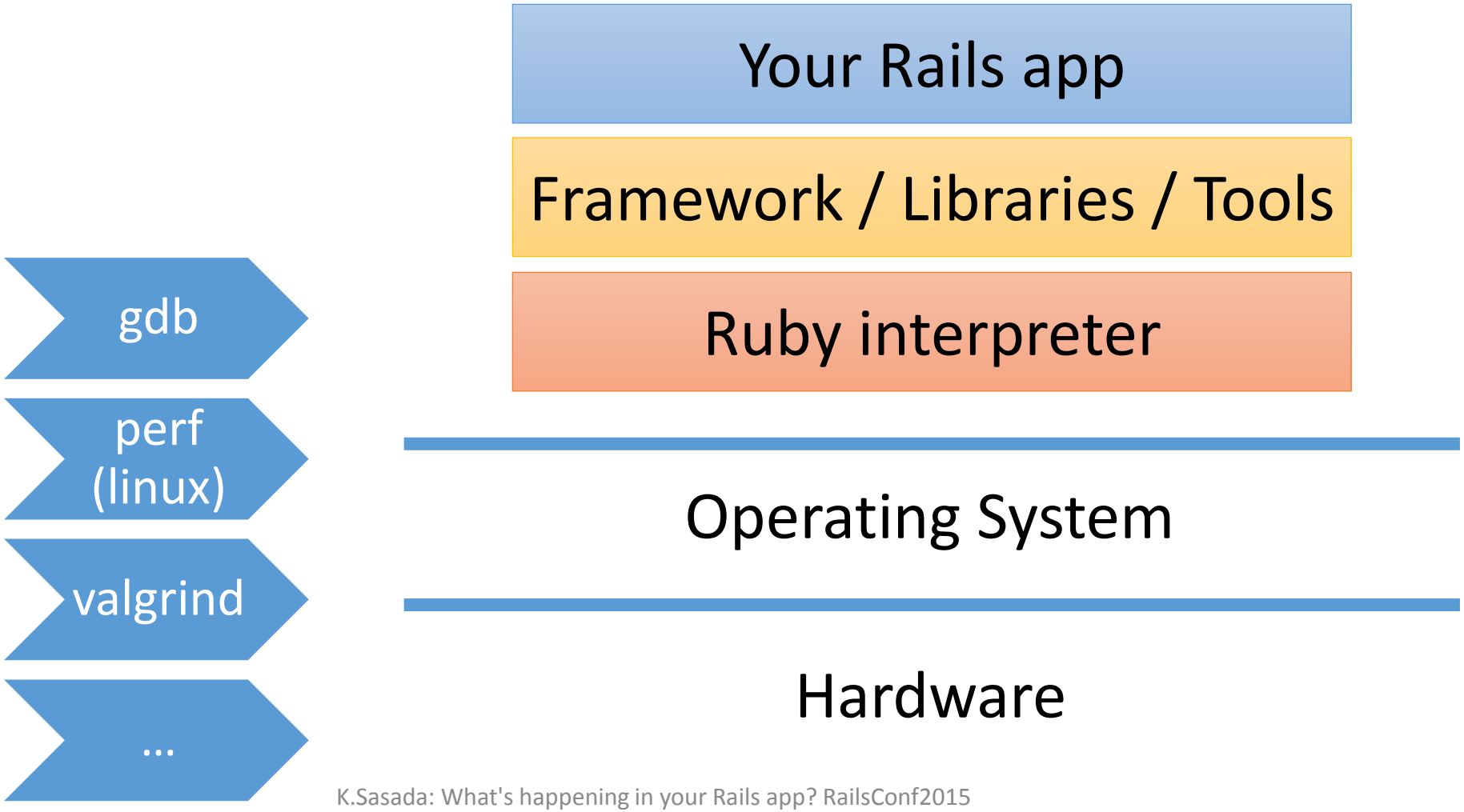
Hack Ruby

- “Ruby Under a Microscope”
  - By Pat Shaughnessy
  - http://patshaughnessy.net/ruby-under-a-microscope
- “Ruby Hacking Guide”
  - By Minero Aoki, written in Japanese
  - English translation: http://ruby-hacking-guide.github.io/

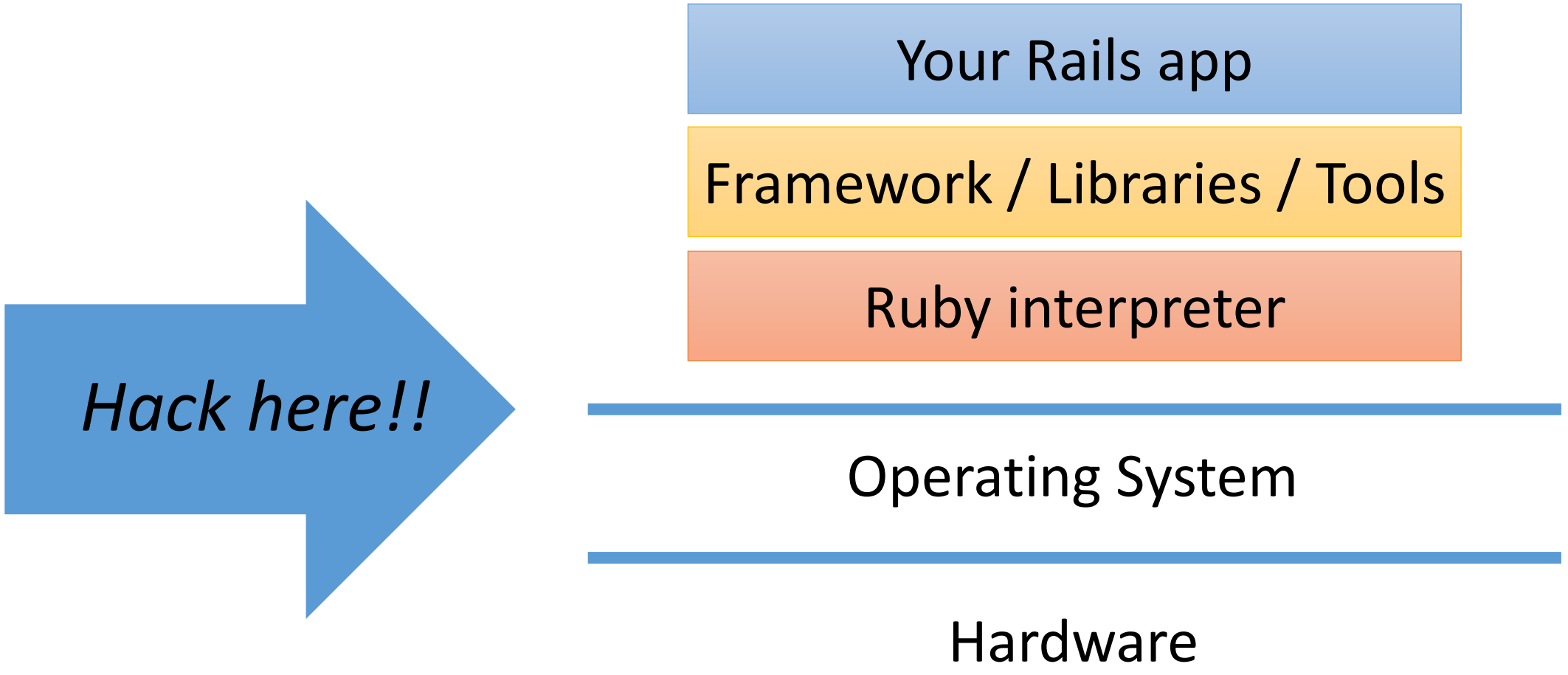




Combine with low level tools



Hacking low level systems



Rails programming is fun.

Low-level programming is also fun!

(sometimes)

heroku

Message of this talk

You can introspect your Rails application with existing tools

- So many efforts on this purpose
- Please find them

You can make your own inspection tools for your Rails application

- Recent Ruby/MRI will help you
- I can help you, too

Thank you for your attention

Koichi Sasada

<ko1@heroku.com>