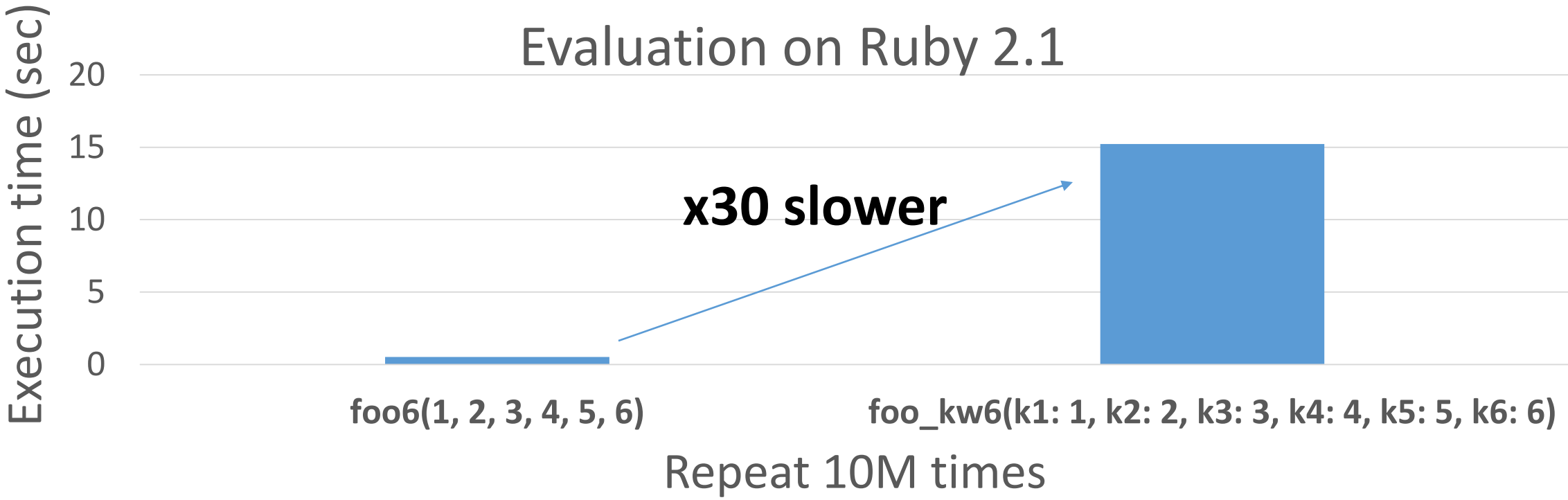
キーワードパラメータを支える技術

笹田耕一
<ko1@heroku.com>
Heroku, Inc.

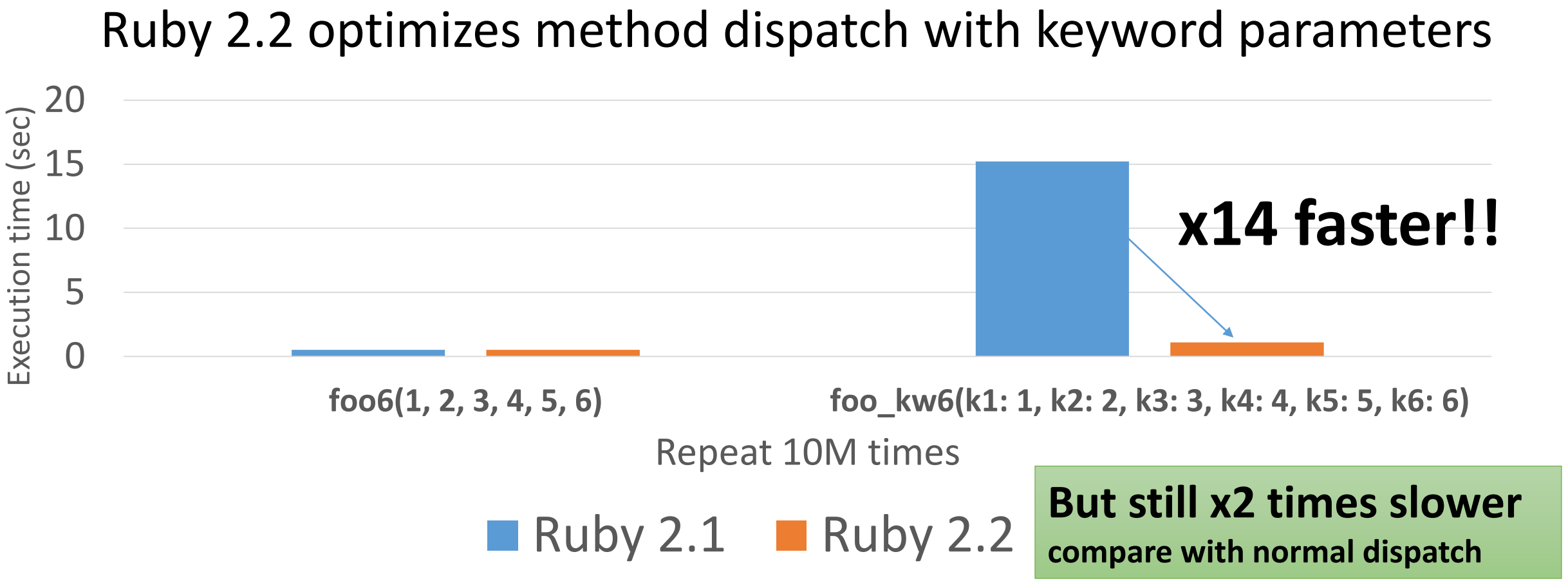
関西Ruby会議06

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

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

Koichi is an Employee

Koichi is a member of Heroku Matz team

Mission

Design Ruby language and improve quality of MRI

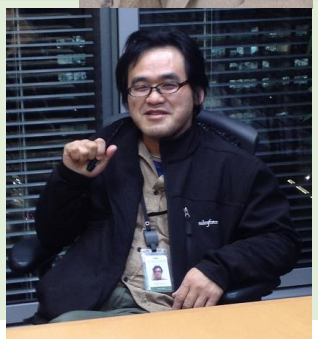
Heroku employs three full time Ruby core developers in Japan named “Matz team”

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

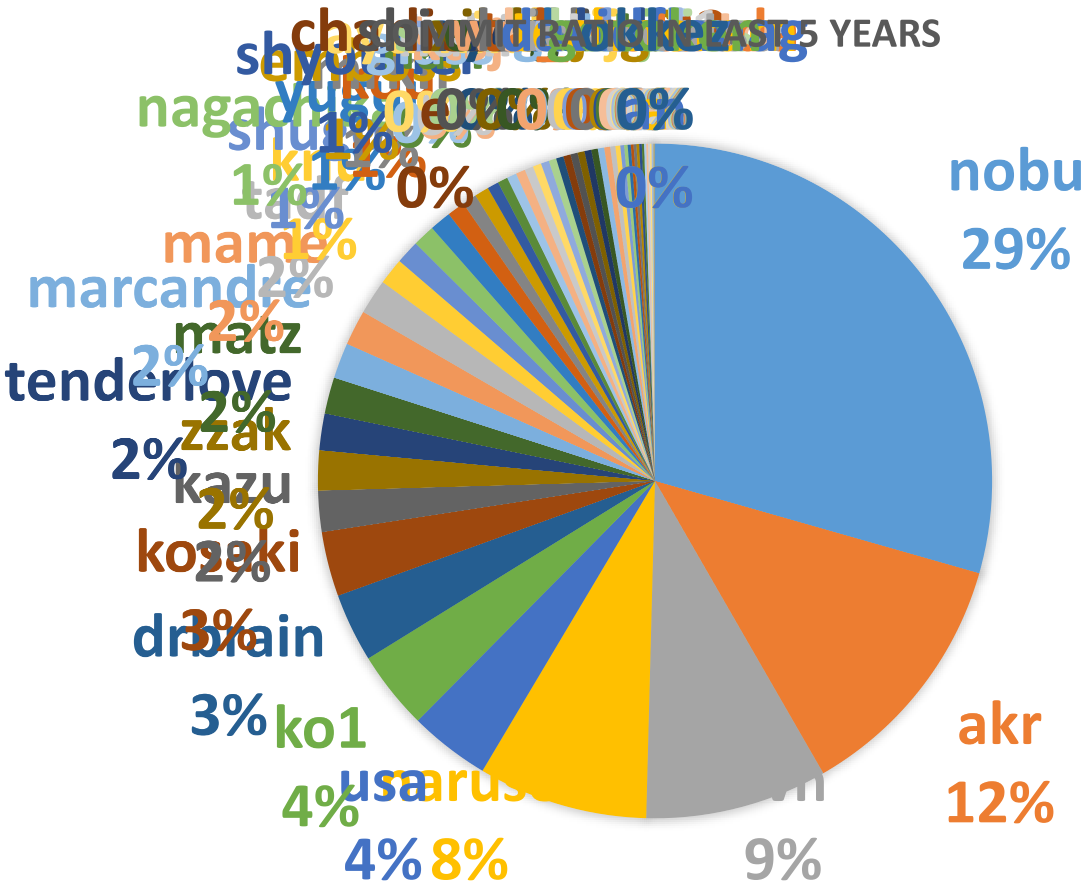
Ruby's bug

|> Fix Ruby

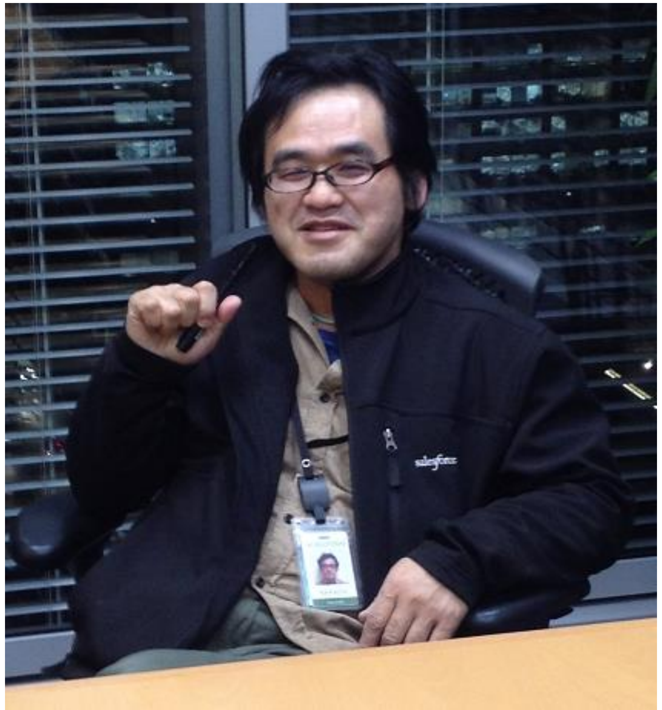
|> Break Ruby

|> And Fix Ruby

Nobu
Patch monster

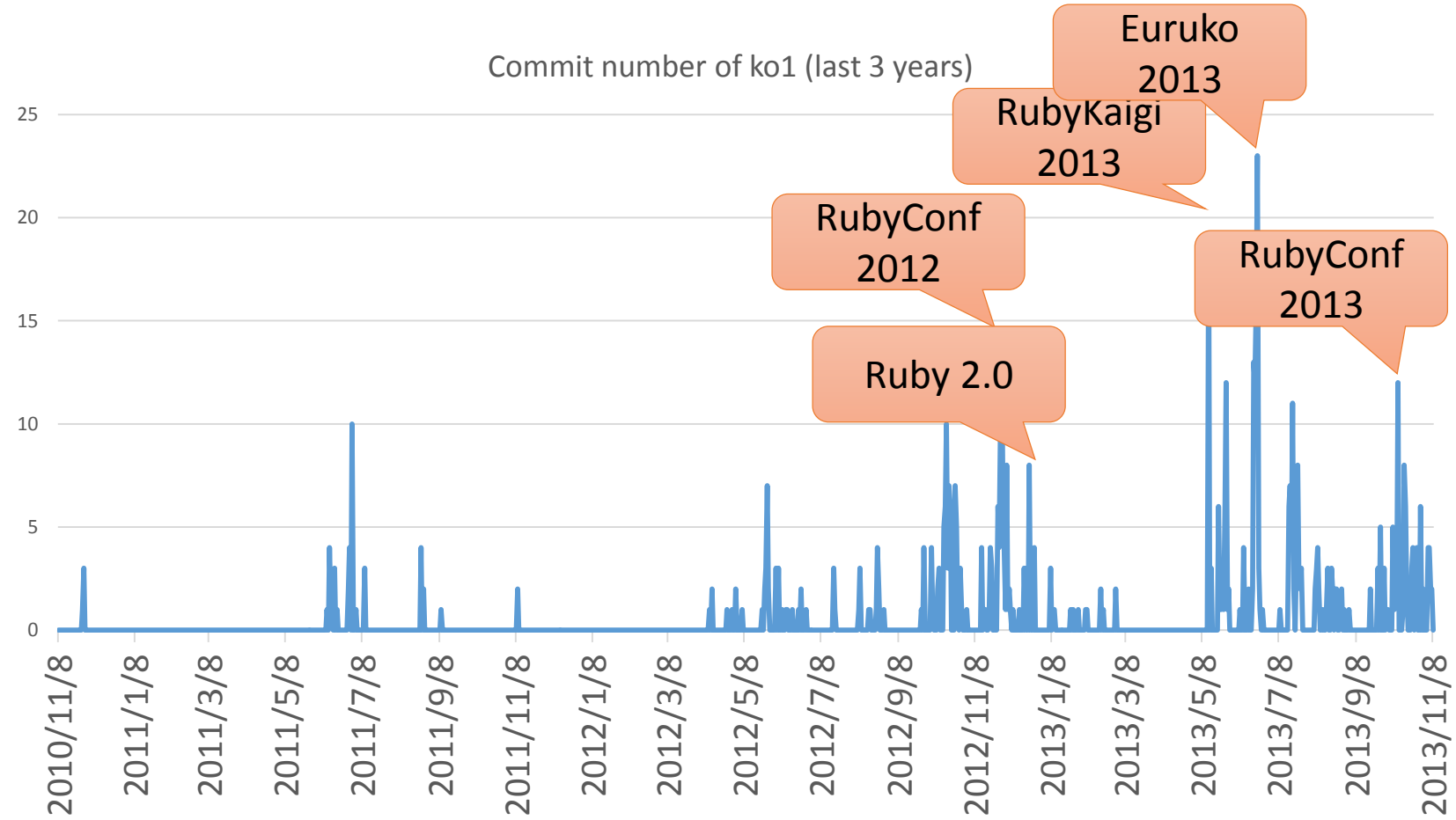


Commit count of MRI



Nobu
The Ruby Hero

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

- 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

- 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

- 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

```
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.)

TRAP!!

Ruby 2.2.0 has memory leak error!

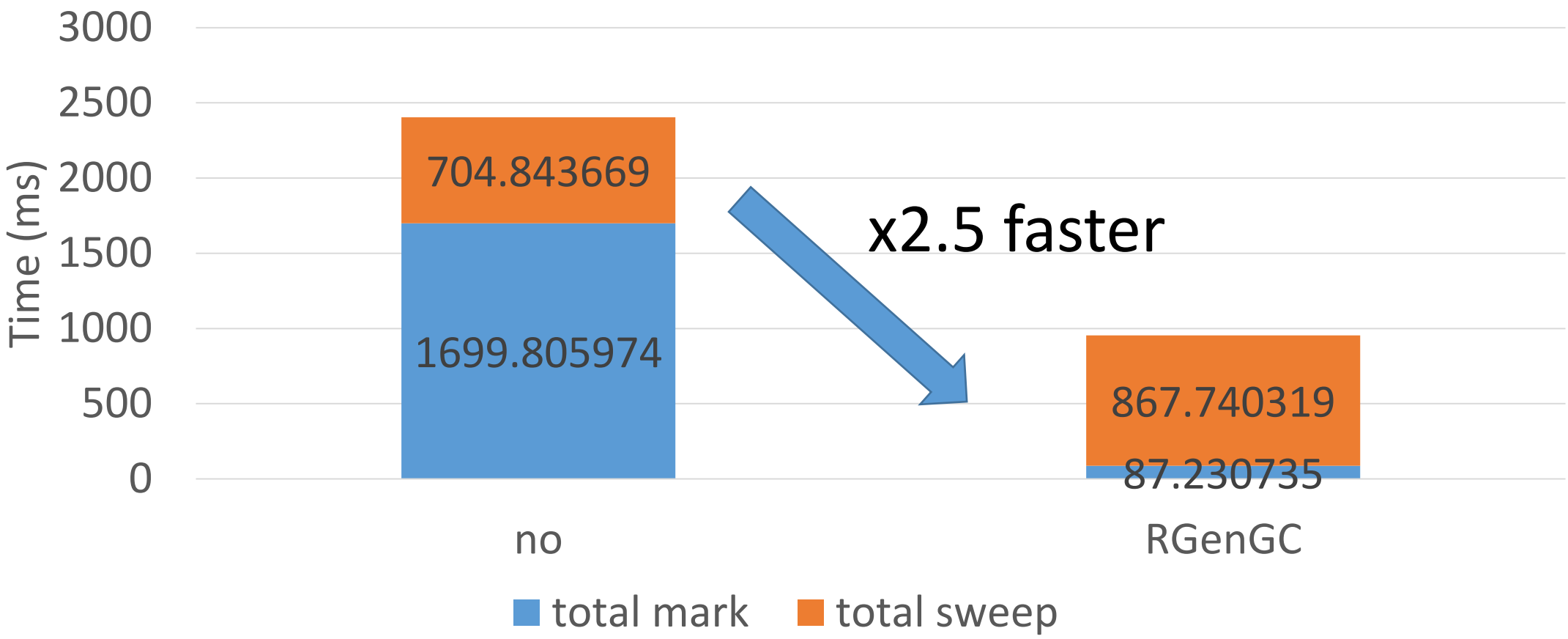
- Upgrade Ruby 2.2.2
  - Memory (object) leak problem
    - Symbols has corresponding String objects
    - Symbols are collected, but Strings are not collected! (leak)
  - Ruby 2.2.1 solved this problem!!
    - However, 2.2.1 also has problem (rarely you encounter BUG at **the end of process** **[Bug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

Ruby 2.2
Incremental GC

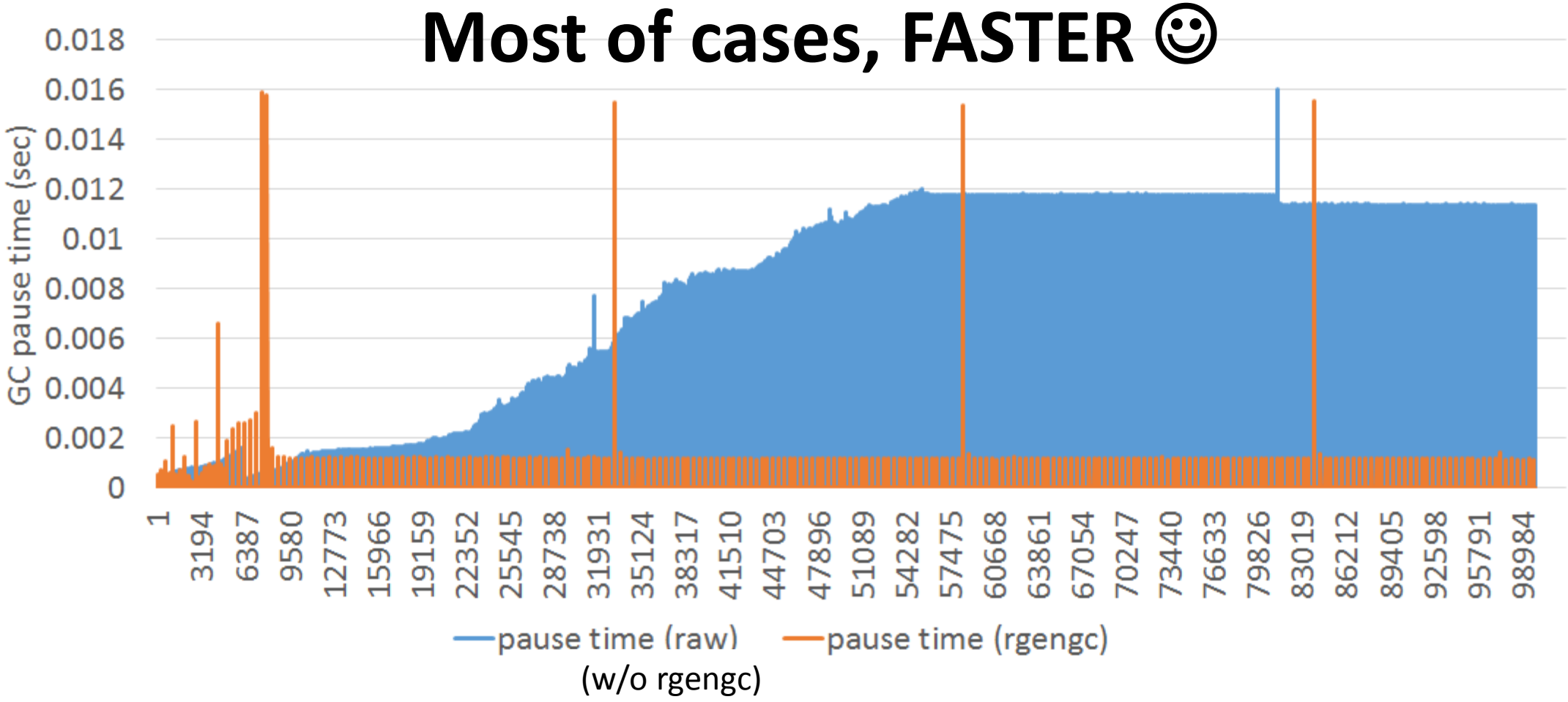
Goal

	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

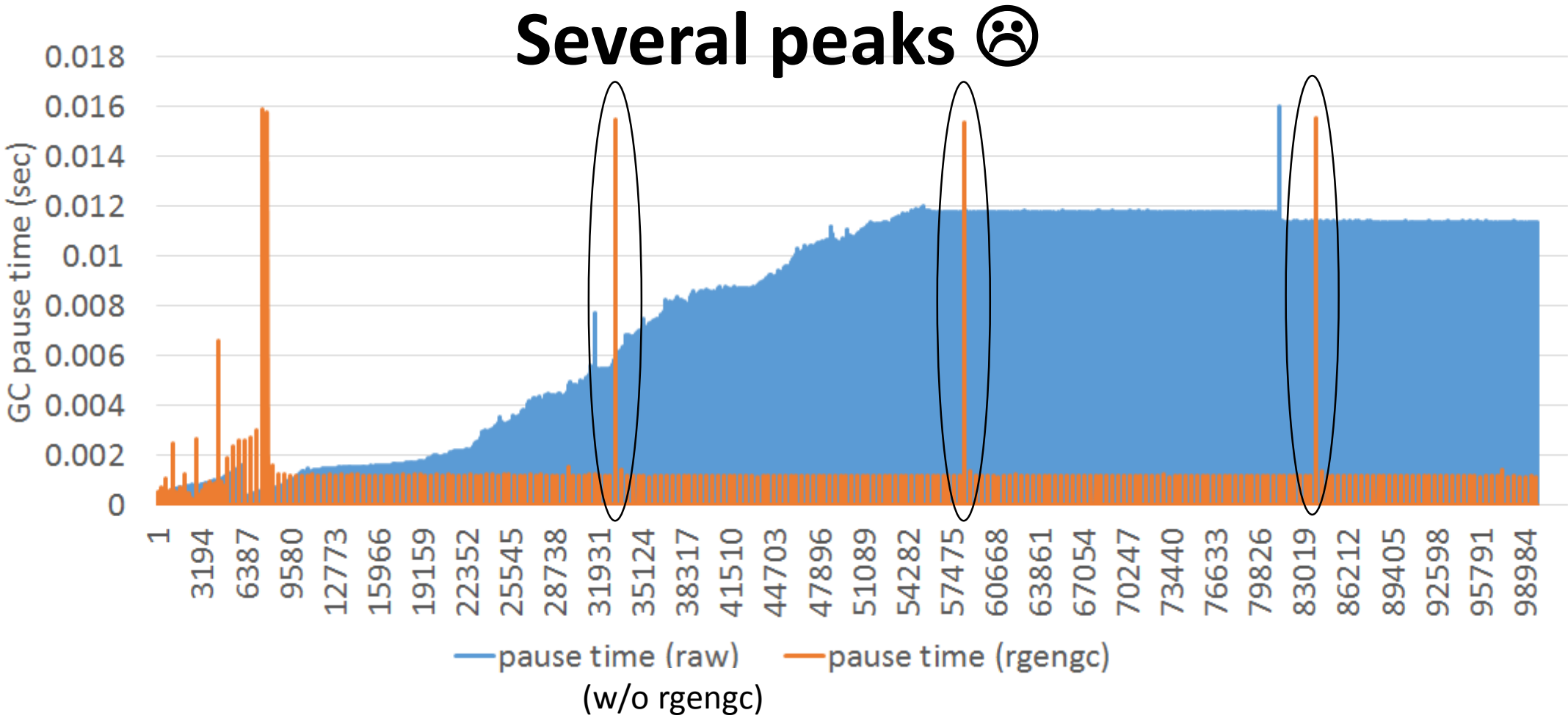
RGenGC from Ruby 2.1: Micro-benchmark



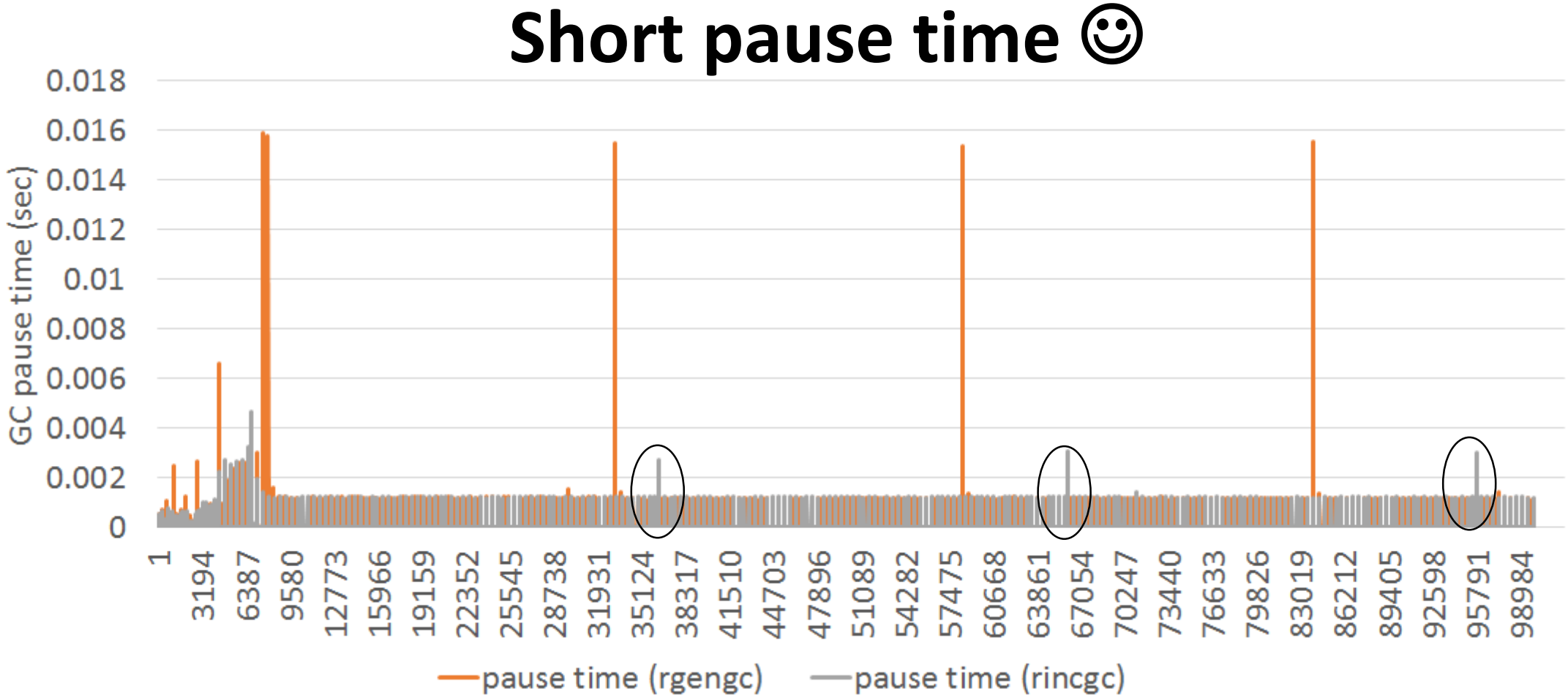
RGenGC from Ruby 2.1: Pause time



RGenGC from Ruby 2.1: Pause time

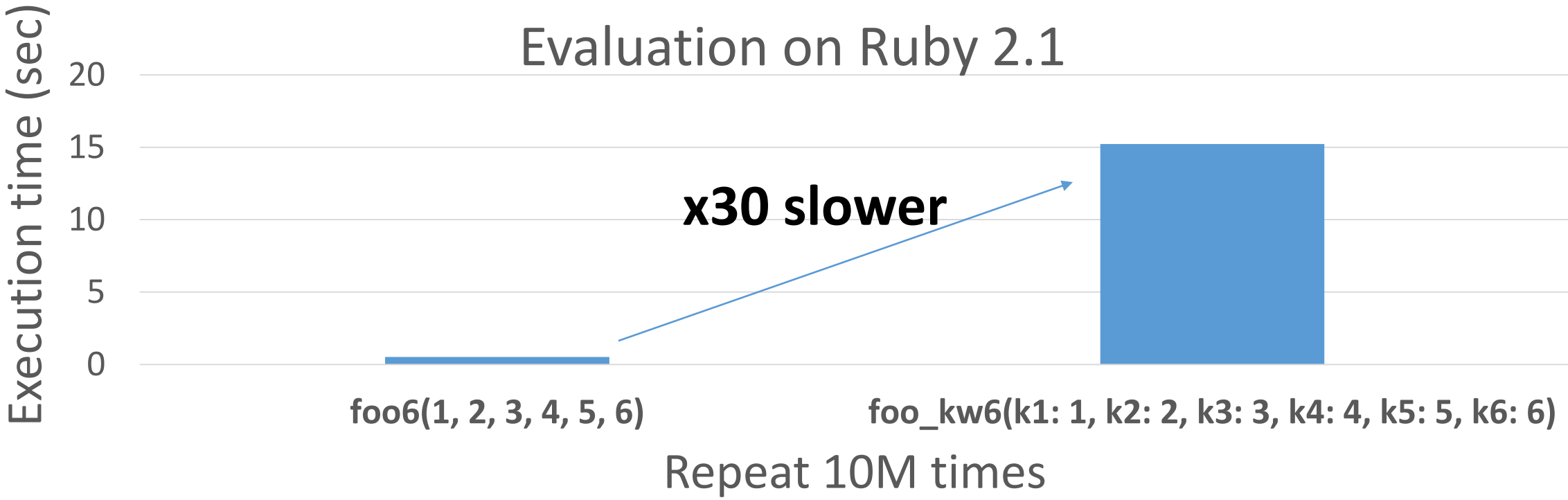


Ruby 2.2 Incremental GC

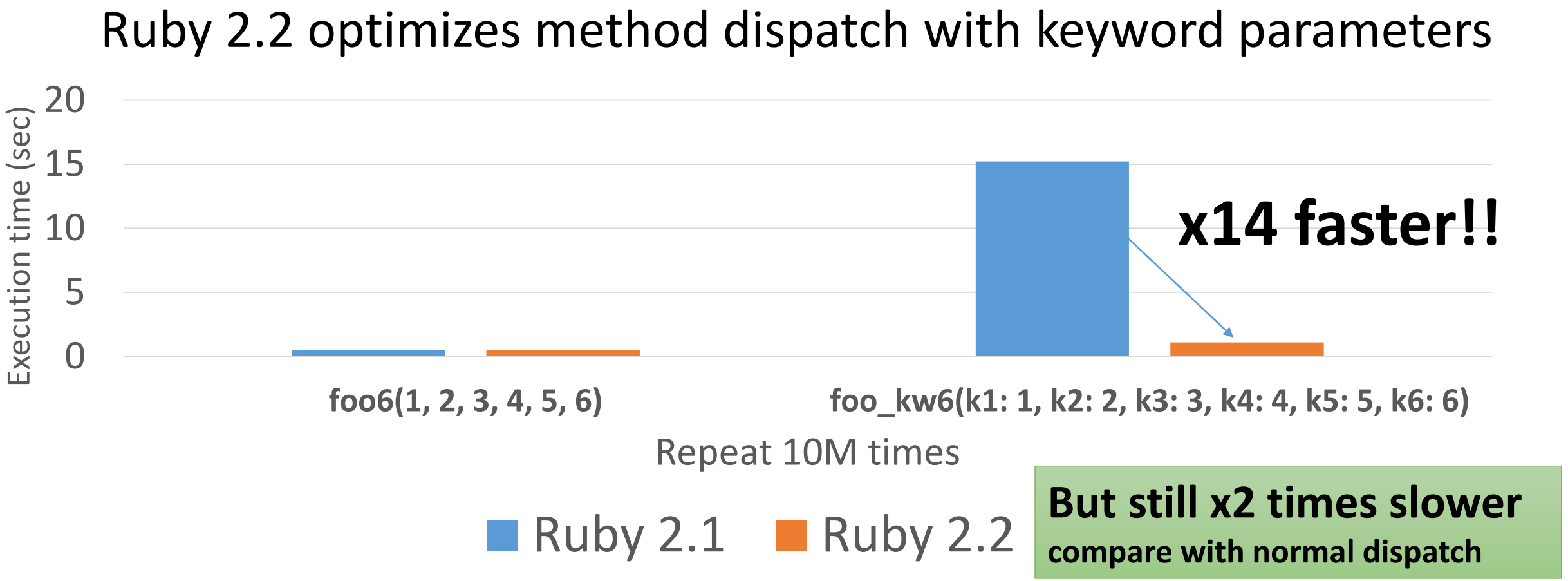


Ruby 2.2
Fast keyword parameters

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



Ruby 2.2
Fast keyword parameters



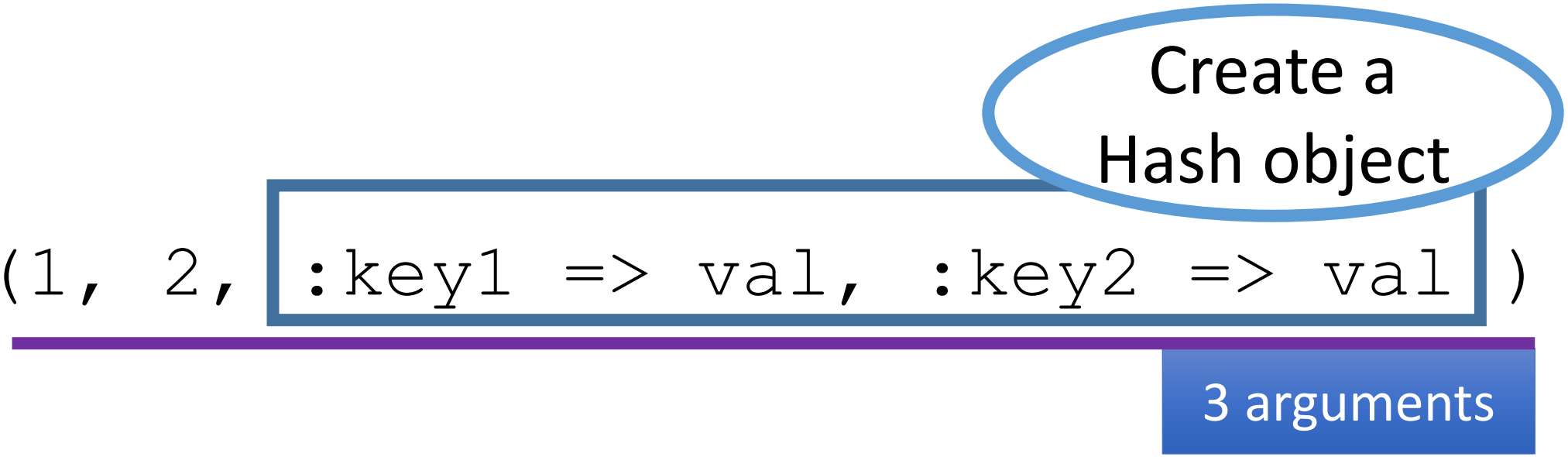
But still x2 times slower
compare with normal dispatch

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

Break

The History of Keyword parameter

Hash notation at the last argument




```
# Same as
# foo(1, 2, {:key1 => val, :key2 => val})
```

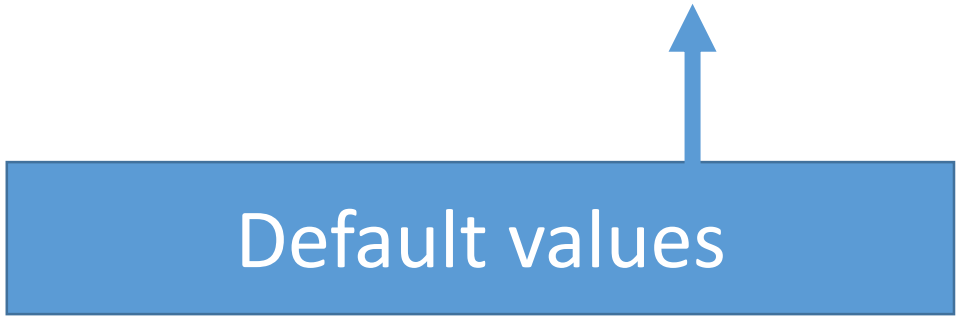
Symbol hash notation from Ruby 1.9.3

```
foo(1, 2, key1: val, key2: val)

# Same as
# foo(1, 2, :key1 => val, :key2 => val)
# foo(1, 2, {:key1 => val, :key2 => val})
```

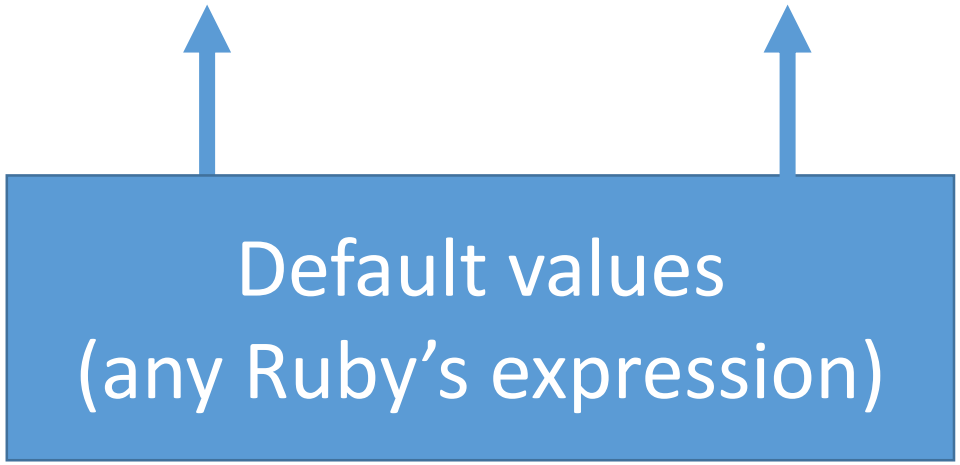
Keyword parameters processing before 2.0

```
def foo(a, b, kw) # kw is Hash
  key1 = kw.fetch(:key1, 1)
  key2 = kw.fetch(:key2, 2)
  …
end
```



Keyword parameters from Ruby 2.0

```
def foo(a, b, key1: 1, key2: 2)
  …
end
```



Keyword parameters from Ruby 2.0 (2)

- Raise an exception when unknown keywords are passed
- Rest keyword parameter (**kw) can receive non-specified keyword parameters

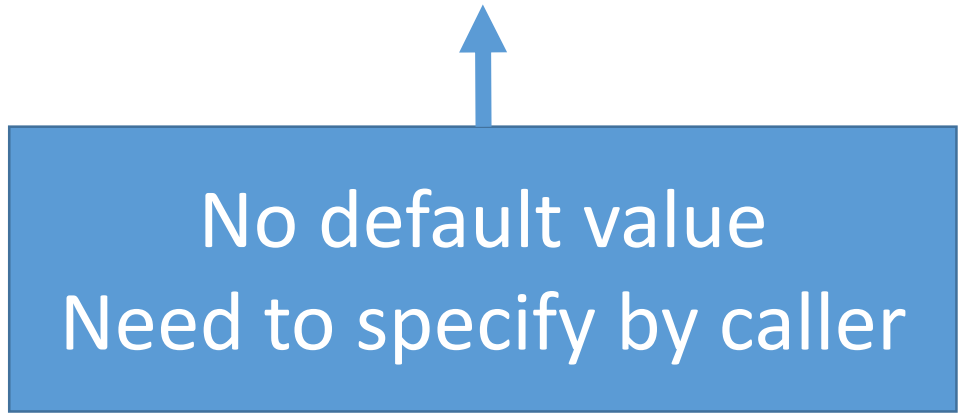
```
def foo(k1: v1, **kw)
  p kw #=> {k2: 2, k3: 3}
end
foo(k1: 1, k2: 2, k3: 3)
```

- Also blocks can accept keyword parameters

```
foo{|k1: 1, k2: 2| …}
```

Required keyword parameter from Ruby 2.1

```
def foo(a, b, key1: 1, key2:)
  …
end
```




```
def foo(a, b, key1: 1, key2: raise(“err”))
  …
end
```

The Implementation of Keyword parameter

Implementation of keyword parameter Ruby 2.0 and Ruby 2.1

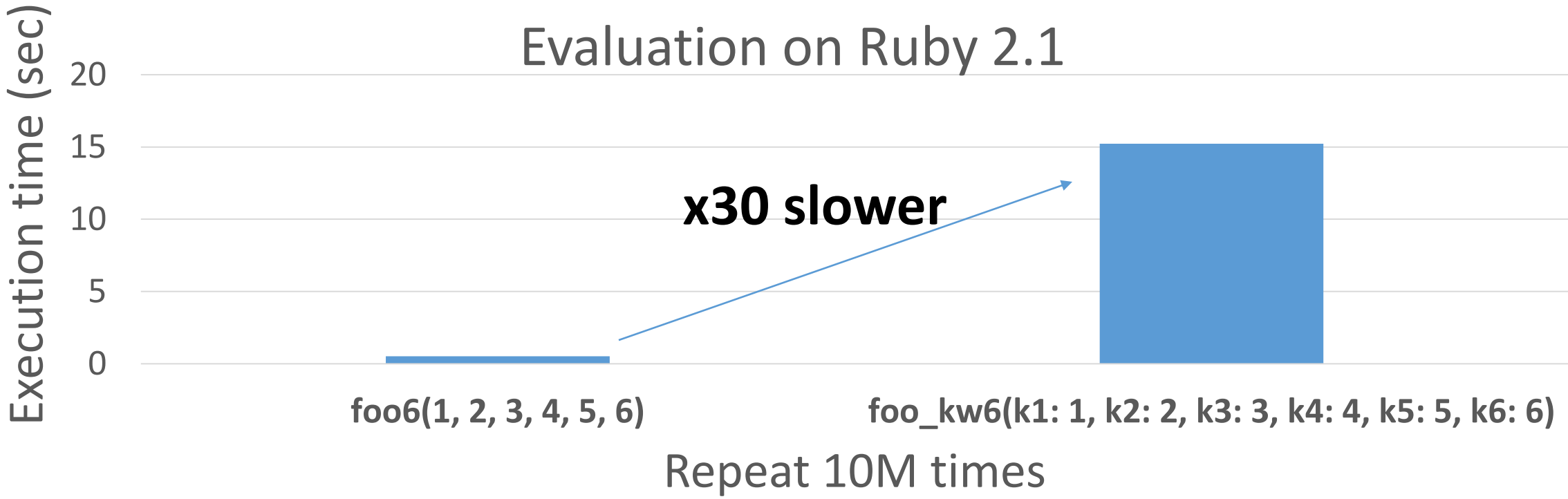
- Caller: make a Hash object and pass it normally
  - Same as Ruby 2.0
- Callee: decompose a Hash object and assign correctly
  - Mostly same code of decomposing code in Ruby
  - Need some more error checking

```
def foo(k1: v1, k2: v2)
  …
end
```

Mostly same as

```
def foo(h)
  k1 = h.fetch(:k1, v1)
  k2 = h.fetch(:k2, v2)
  …
end
```

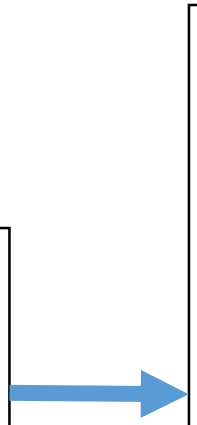
Slow keyword parameters



Why slow compare with normal parameters?

1. Hash creation
2. Hash access

```
def foo(k1: v1, k2: v2)
  …
end
foo(k1: 1, k2: 2)
```



```
def foo(h)
  k1 = h.fetch(:k1, v1)
  k2 = h.fetch(:k2, v2)
  …
end
foo( {k1: 1, k2: 2} )
```

(2) Hash access

(1) Hash creation

Optimization technique of keyword parameters from Ruby 2.2

- Key technique

  → Pass “a keyword list” instead of a Hash object

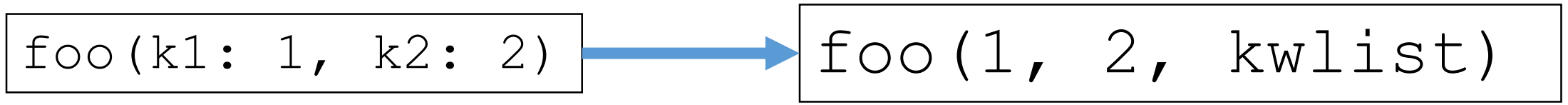
Preparation
Make “keyword list” and “default value list”

- We can see all source code at compile time
- Collect keywords in a list for each method dispatch
  - ex: “foo(k1: x, k2: y)” #=> kwlist is [:k1, :k2]
- Collect “Default values list” in each method definition
  - ex: “def foo(k1: 1, k2: 2)” #=> dvlist is [1, 2]
  - ex: “def foo(k1: 1, k2: f2()) #=> dvlist is [1, Qundef]

NOTE: Qundef is internal special value which should not expose Ruby world

Call with keyword parameter [Sender] Pass “kwlist” instead of making a Hash

- Pass values with “the keyword list”



NOTE: kwlist is not passed as an argument, but passed as calling information.

Call with keyword parameter [Receiver] Manipulate passed kwlist

- Assign local variables with passed keyword list

```
def foo(k1: 1, k2: 2, k3: 3)
```

Rkwlist = [:k1, :k2, :k3]
dvlist = [1, 2, 3]

Pseudo code

```
def foo(*vs, kwlist)
  Rkwlist.each.with_index{|k, i|
    ki = kwlist.index(k))
    assign(k, ki ? vs[ki] : dvlist[i])
  }
```

Call with keyword parameter [Receiver] Treat with default values as expressions

```
def foo(k1: 1, k2: f2(), k3: f3())
```

Rkwlist = [:k1, :k2, :k3]
dvlist = [1, Qundef, Qundef]

Pseudo code

```
def foo(*vs, kwlist)
  unset_bits = 0
  Rkwlist.each.with_index{|k, i|
    if ki = kwlist.index(k)
      v = vs[ki]
    else if (v = dvlist[i]) == Qundef
      v = nil
      unset_bits[i] = 1
    end
    assign(k, v)
  } # cont to right
    # continue
    # k1 is already initialized
    k2 = f2() unless unset_bits[1]
    k3 = f3() unless unset_bits[2]

    … # start of method body
  end
```

NOTE: Qundef is internal special value which should not expose Ruby world

Q. Why not assign Qundef directly?

```
def foo(k1: 1, k2: f2(), k3: f3())
```

Rkwlist = [:k1, :k2, :k3]
dvlist = [1, Qundef, Qundef]

Pseudo code

```
def foo(*vs, kwlist)
  unset_bits = 0
  Rkwlist.each.with_index{|k, i|
    ki = kwlist.index(k)
    v = ki ? vs[ki] : dvlist[i]
    assign(k, ki)
  }
  k2 = f2() unless k2 == Qundef
  k3 = f3() unless k3 == Qundef
  … # start of method body
end
```

A. We can access initializing keyword variables with eval()

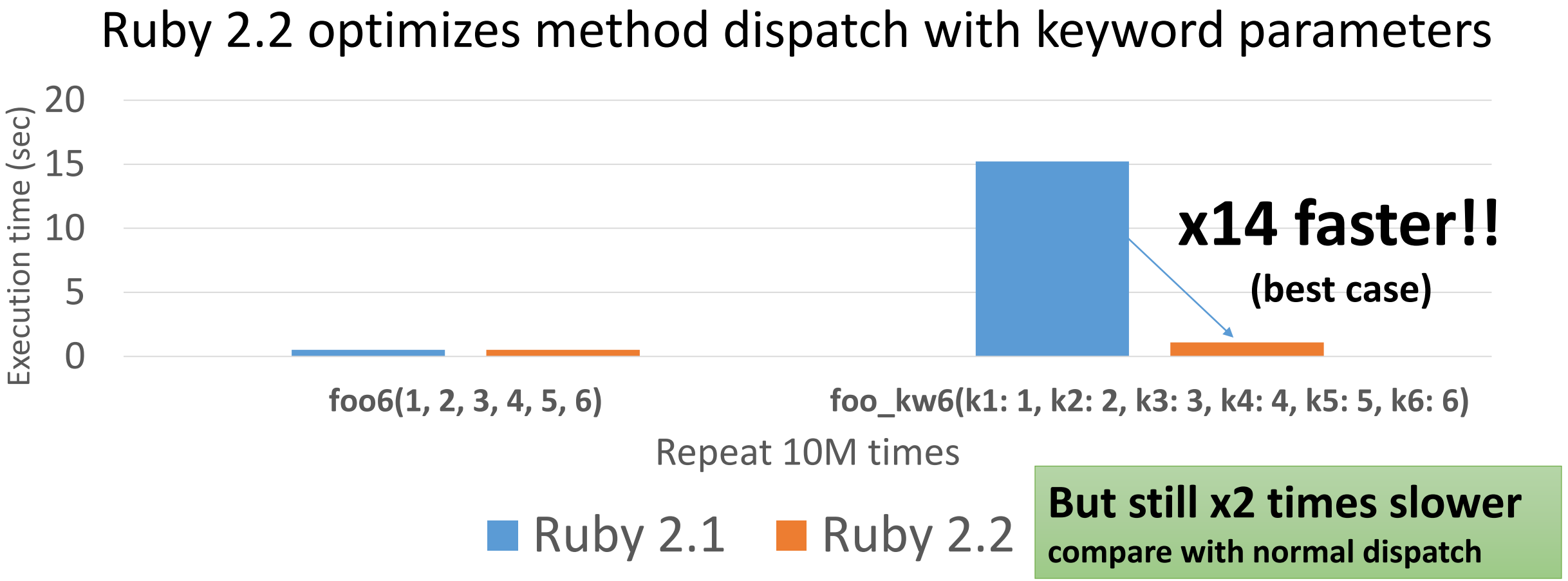
```
def foo(k1: 1, k2: eval(“k3”)), k3: f3())

                                # k2 should be nil
```

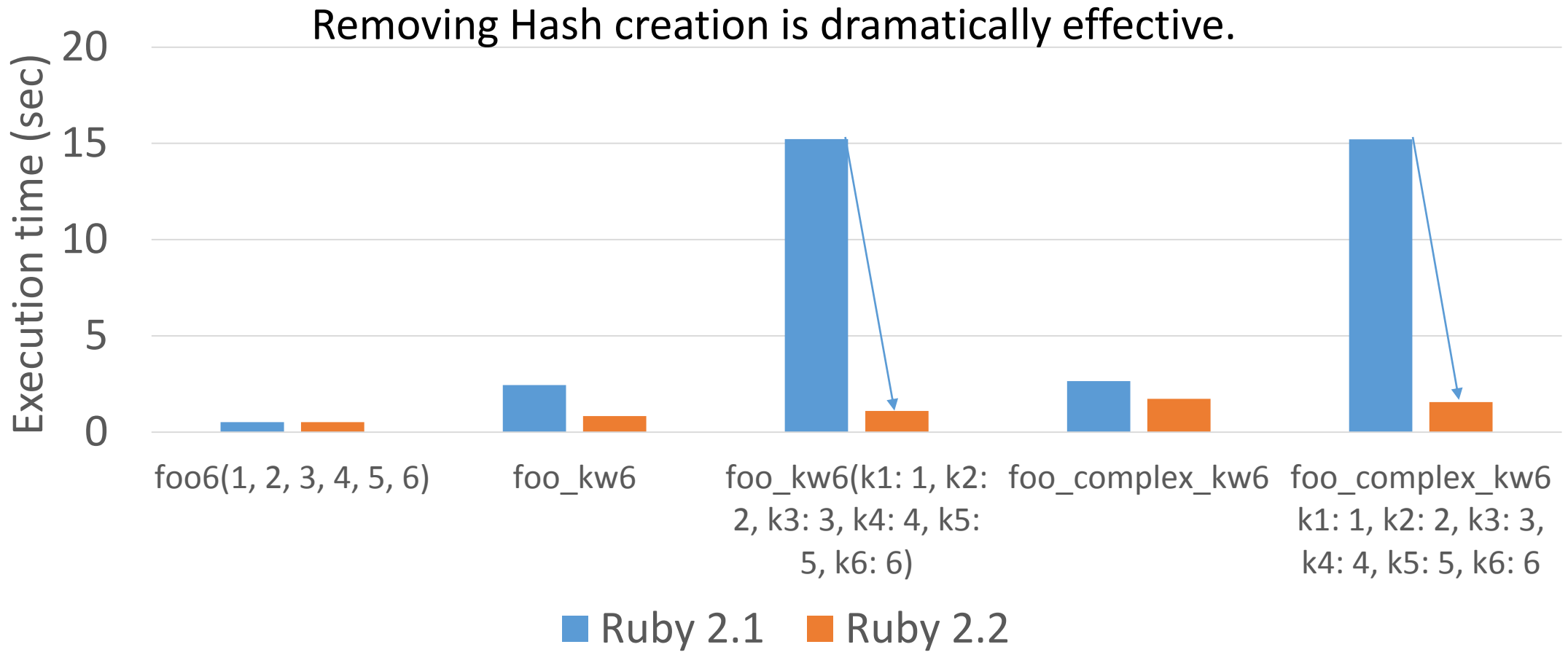
Result
Compare 3 types methods

1. def foo6(a, b, c, d, e, f); end
2. def foo_kw6(k1: 1, k2: 2, k3: 3, k4: 4, k5: 5, k6: 6); end
3. def foo_complex_kw6(k1: 1+1, k2: 2+1, k3: 3+1, k4: 4+1, k5: 5+1, k6: 6+1); end
   - Default values are expressions (not immediate values)

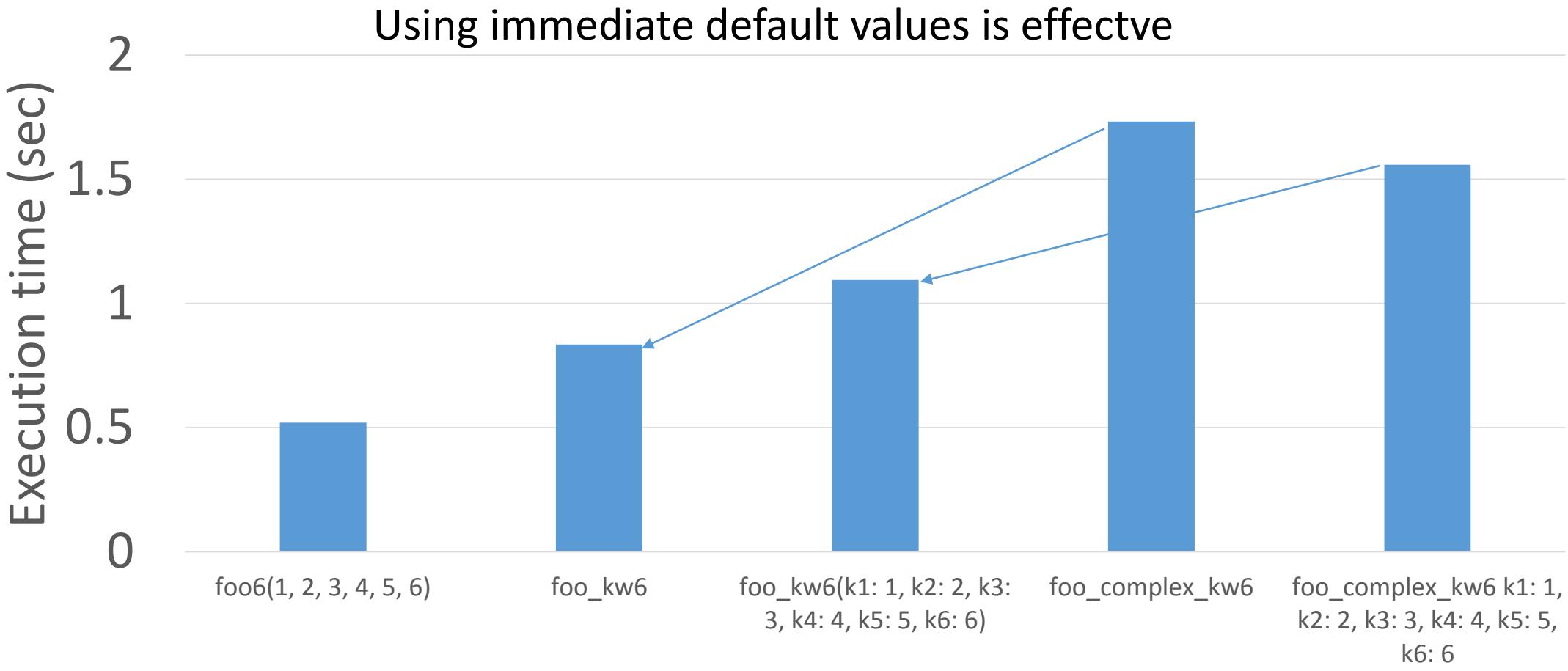
Result: Fast keyword parameters



Result: Ruby 2.1 vs. Ruby 2.2



Result: Ruby 2.2



Challenge: improve computational complexity

- Computational complexity is O(mn)
  - Now, m and n is enough small (only a few keywords), but…

n = kwlist.length
m = Rkwlist.length
Total computationan complexity: **O(mn)**

Pseudo code

```
def foo(v1, v2, kwlist)
  Rkwlist.each.with_index{|k, i| # m times
    ki = kwlist.index(k)
                                  O(n)
    …
```

Thank you for your attention

Koichi Sasada

<ko1@heroku.com>

heroku