MRI Internals

Koichi Sasada
ko1@heroku.com

MRI Internals towards Ruby 3

Koichi Sasada
ko1@heroku.com

Today’s talk

- Koichi is working on improving Ruby internals
- Introduce my ideas toward **Ruby 3**

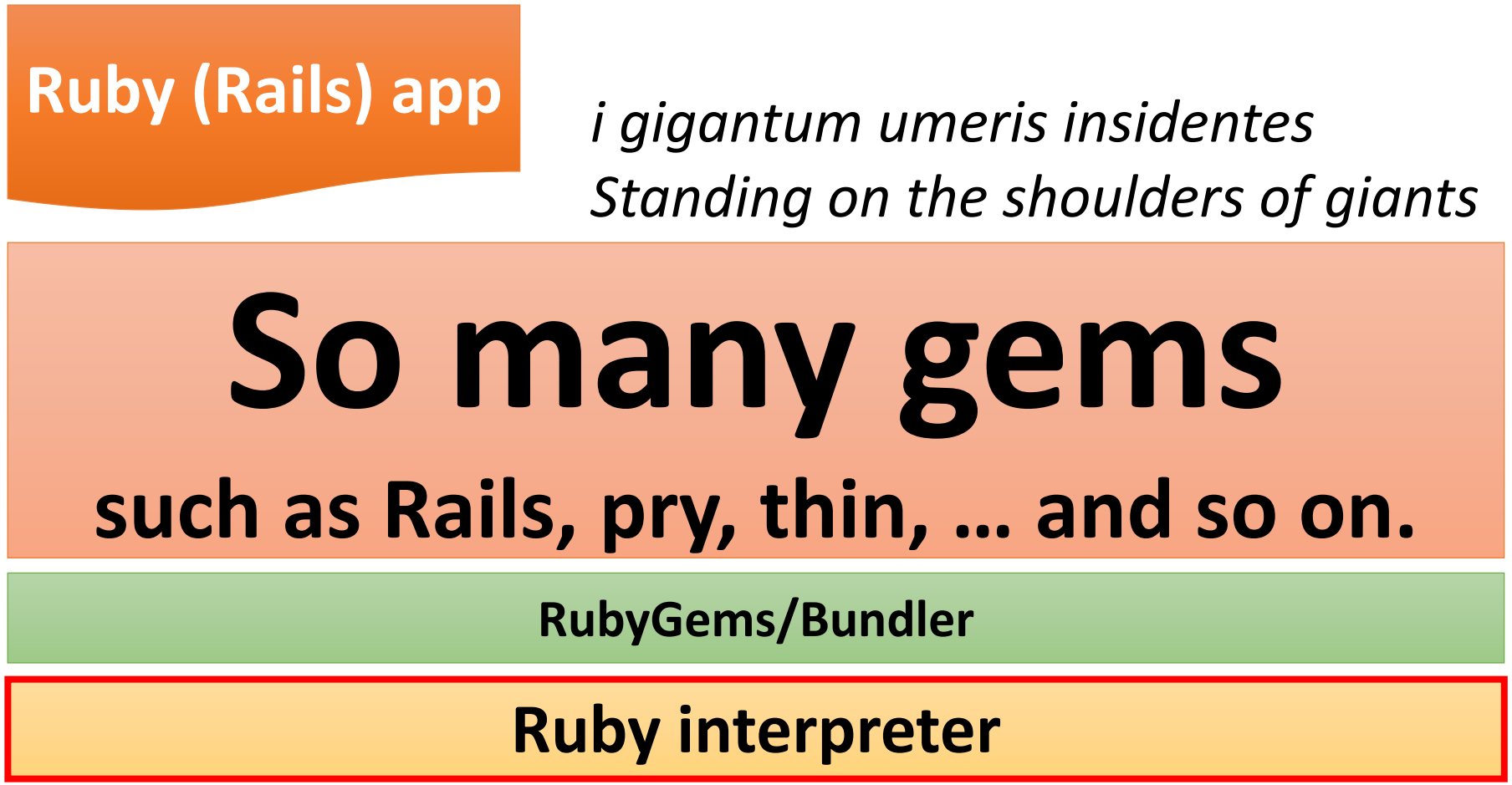
Koichi Sasada

A programmer living in Japan

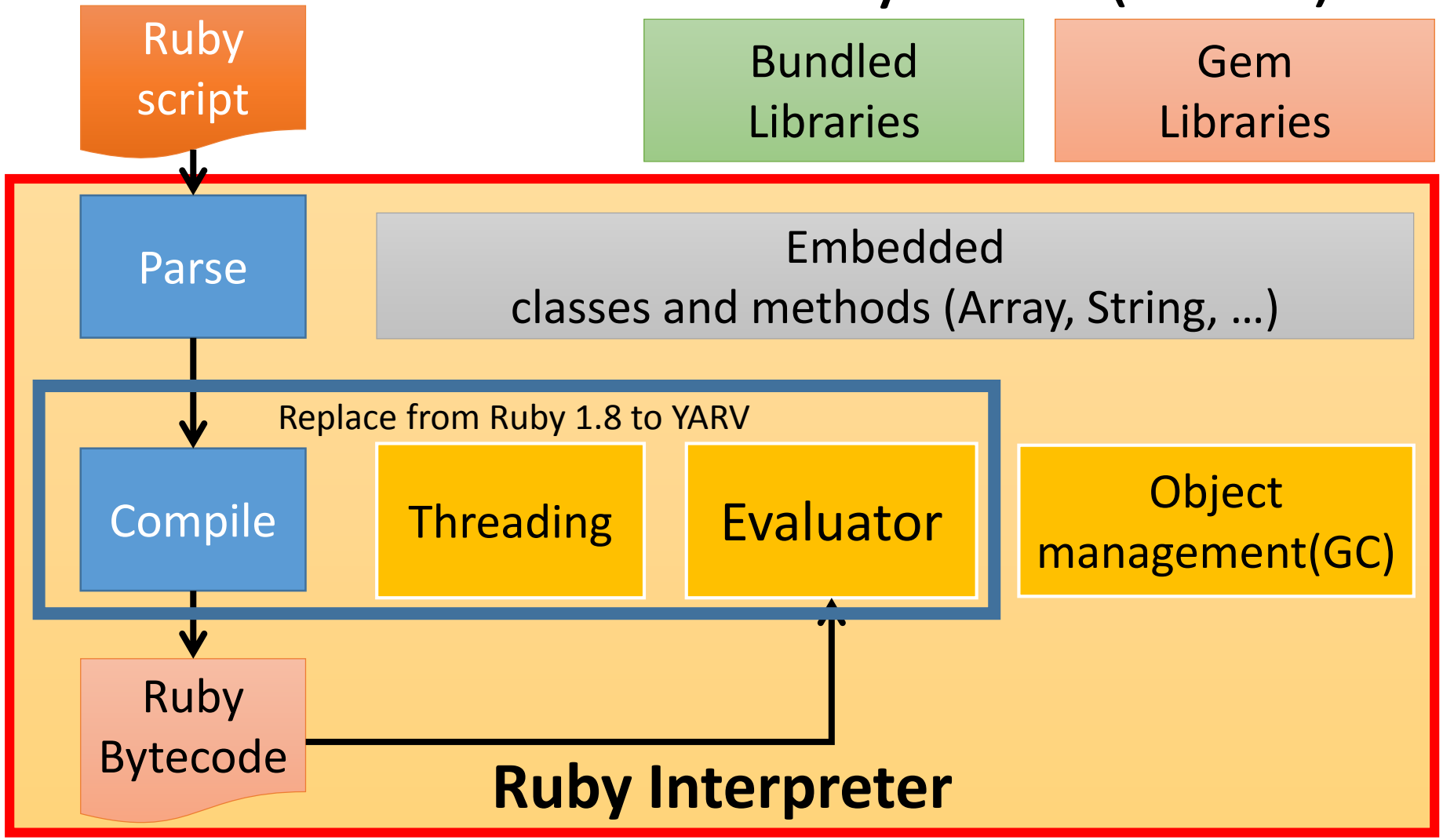
Koichi is a Programmer

- Ruby interpreter (MRI) developer over 10 years
  - YARV: Yet Another RubyVM from 2004/Jan
  - MRI committer since 2007/Jan

Koichi's contributions
YARV: Yet Another RubyVM (1.9-)

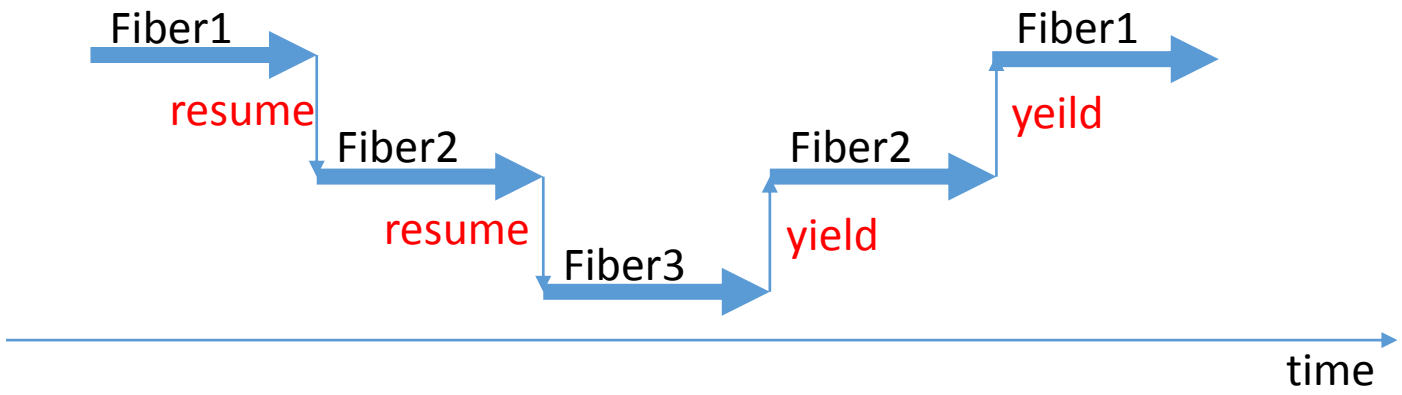


Koichi's contributions
YARV: Yet Another RubyVM (1.9-)



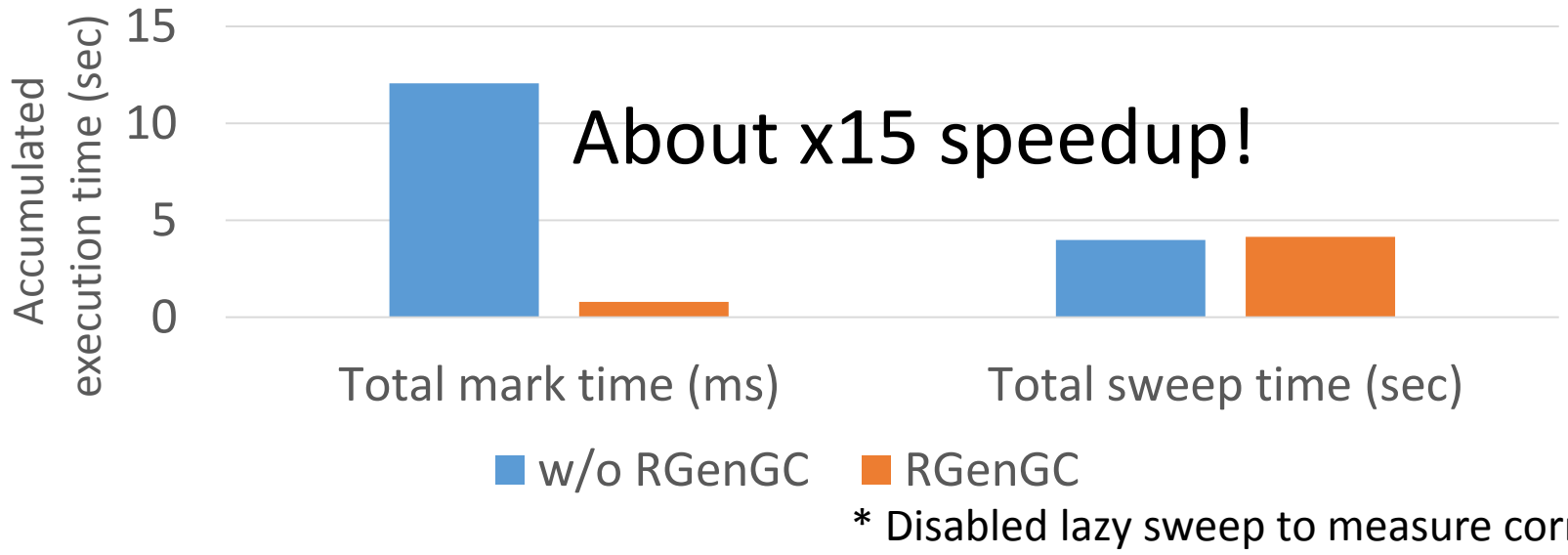
Koichi's contributions
Fiber (Ruby 1.9-)

- Abstraction objects of execution contexts
  - Fiber is from Windows API
  - Cooperative thread, Coroutine (or Semi-Coroutine)
- Fast fiber context switch with non-portable methods



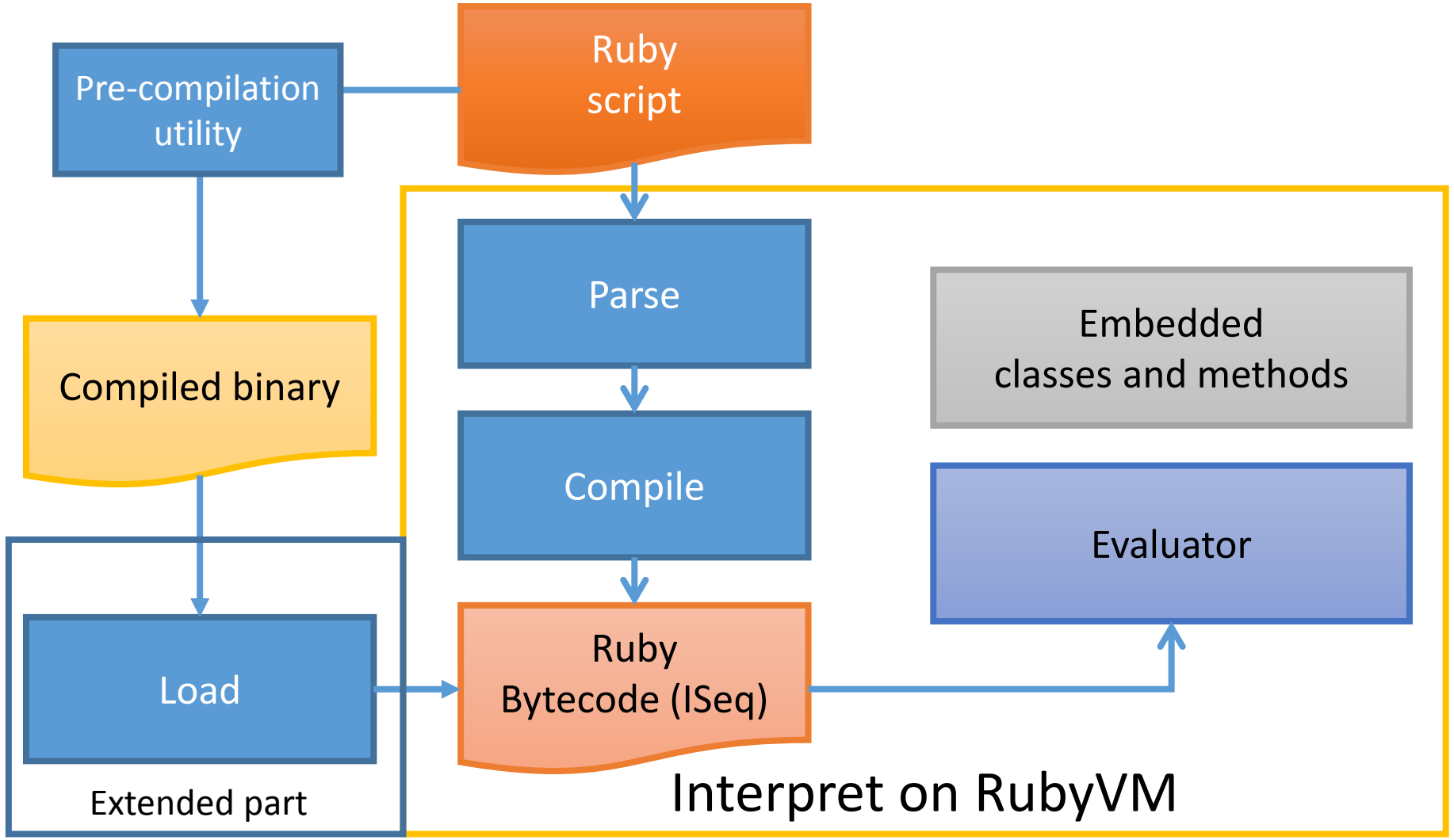
Koichi's contributions
GC Improvements (Ruby 2.1-)

- Generational GC (for Ruby 2.1)
  - Introduce RGenGC by inventing "WB-unprotected" objects technique and reduce marking time dramatically
- Incremental GC with same technique (for Ruby 2.2)



* Disabled lazy sweep to measure correctly.

Koichi's contributions
Pre-compilation (Ruby 2.3-)



Koichi is an Employee

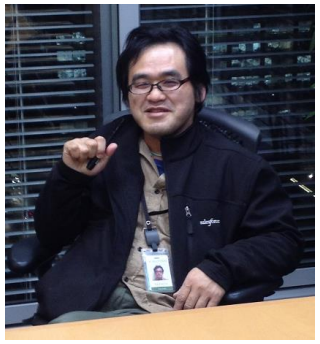
Koichi is a member of Heroku Matz team

Mission

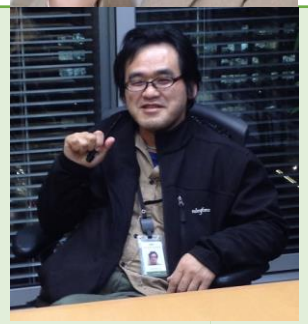
Design Ruby language

and improve quality of MRI

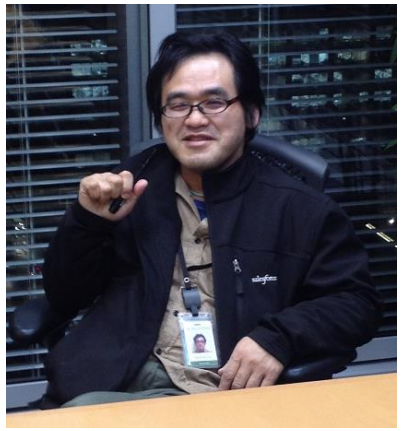
Heroku employs three full time Ruby core developers in Japan named “Matz team”



Heroku Matz team

Matz		**Designer/director of Ruby** **“Design”**
Nobu		**Quite active committer** **“Make and Fix”**
Ko1		**Internal Hacker** **“Optimize”**

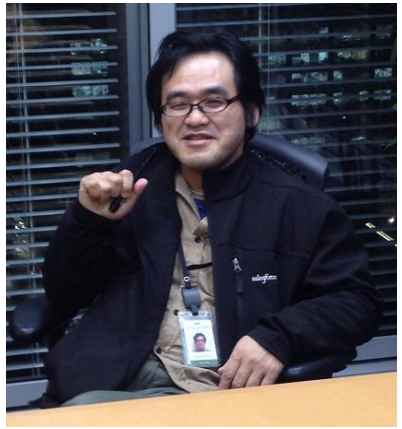
Heroku Matz team



Decide MRI → Make and fix MRI → Optimize MRI

Expected flow

Heroku Matz team



Decide MRI

Make, break and fix MRI

Optimize and break MRI

Matz team Eco system!

Latest release Ruby 2.3

Today, no time to introduce new features…

Please ask me later.

Upcoming Ruby 2.4

Not big features yet.

We are discussing.

MRI Internals toward Ruby 3

Goals of Ruby 3 and current my ideas

Ruby3: Ruby3 has 3 goals

- Static type checking
- Just-in-Time (JIT) compilation
- Parallel execution w/ highly abstract concurrent model

Ruby3: Ruby3 has 3 goals

- For productivity
  - Static type checking
- For performance
  - Just-in-Time (JIT) compilation
  - Parallel execution w/ highly abstract concurrent model

Ruby3x3: Ruby 3 is 3 times faster

- Matz said

> “We will release Ruby 3
> when it is 3 times faster
> than **Ruby 2.0**”

- Proposed by AppFolio
- Good slogan to challenge

Ruby3: 3 goals
My ideas

Static type checking

Just-in-Time (JIT) compilation

Parallel execution w/ highly abstract concurrent model

Ruby3 Goal
Static type checking

Ruby3 Goal
Static type checking

- Please consider this scenario
  - (1) Write your code with Rails
  - (2) Run your rails server
  - (3) (5 hours later…)
  - (4) See RuntimeError with “**uncommon**” request

Ruby3 Goal
Static type checking

- Type checking: Pointing out “wrong” program before running
  - You can know “possible bugs” before running program
- Language types
  - Statically typed language
    - Need to note types for each elements (variables, functions)
    - “Type inference” helps to reduce typing
  - Dynamically typed language (Ruby is this type)
    - No need to write types explicitly
    - Objects (and so on) know each types

Ruby3 Goal
Static type checking

- Frequent proposals
  - Optional typing
    - Allow to write types as annotations
    - Specify classes
    - `def foo(n: Integer)`

Type annotation

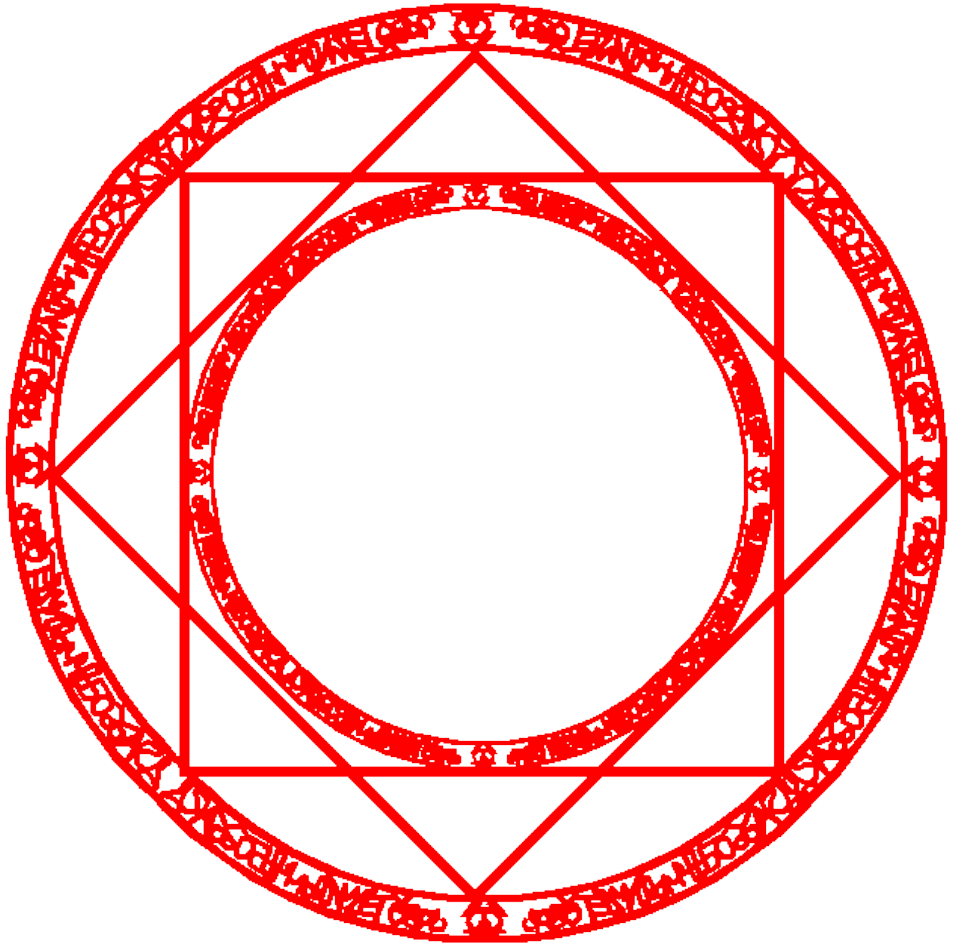
```
    …
    n.bar #=> Error because Integer
          #   does not have #bar
end
```

Ruby3 Goal
Static type checking

- Problem
  - Matz does not like “Type annotation”
    - Matz said “I don’t want to write annotations any more”
    - Because he is lazy ☺
  - What is “Type?”
    - Class is not enough because Rubyist love “Duck typing”
  - How to care “meta-programming”?
    - Ruby can add methods easily while running

Ruby3 Goal
Static type checking

- Solution (?)
  - Invent a new magic
    - Precognition seems nice



Ruby3 Goal
Static type checking

- Solution
  - Soft typing, gradual typing, success typing … from academic research achievements

- Now, we are studying :p

Ruby3 Goal
Just-in-Time (JIT) compilation

Ruby3 Goal
Just-in-Time (JIT) compilation

- Compiling Ruby code while running (just-in-time)
  - Compile to something lower-level such as machine code
- Advantage compare with pre-compile
  - We can know program behaviors as hints
    - Example

```
def foo(n)
  n.times{…} # n seems “Integer”
             # with several traials
end
```

Ruby3 Goal
Just-in-Time (JIT) compilation

- Knowing behaviors (parameters) helps “optimization”
  - Example: Method/block inlining

```
def foo(n)
    n.times{…}
    #=> if n.kind_of?(Integer)
    #     i = 0; while(i<n); …; i+=1; end
    #   else
    #     n.times{…}
    #   end
end
```

Type guard

“while” is faster ☺

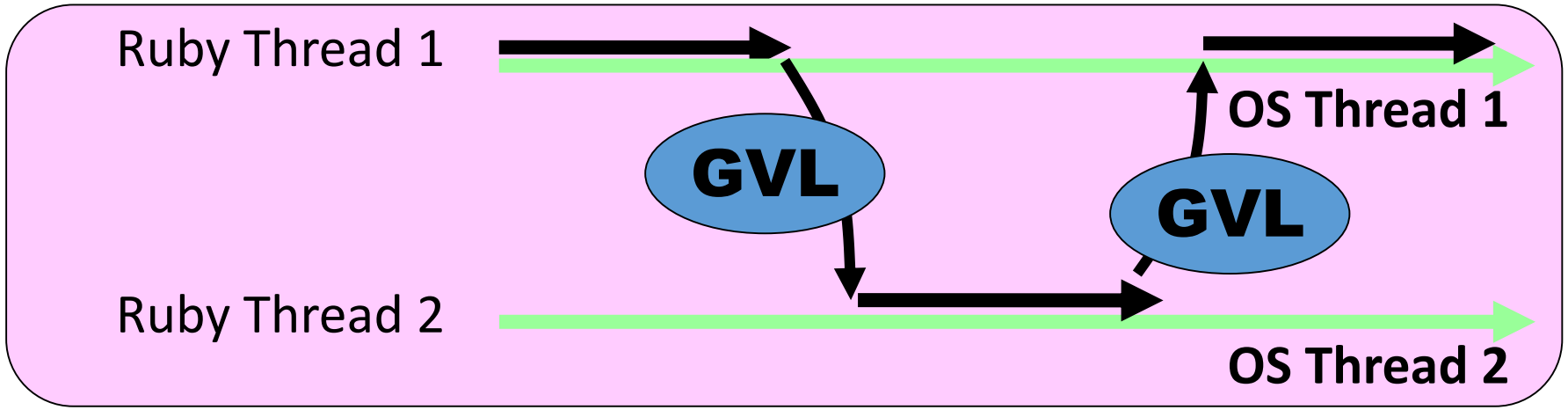
Ruby3 Goal
Just-in-Time (JIT) compilation

- How/who to implement it?
  - (1) Use IBM Ruby OMR project
    - https://github.com/rubyomr-preview/rubyomr-preview
  - (2) Use LLVM
  - (3) Implement own JIT compilers
    - RuJIT (translate Ruby to C with hints)
      - Memory consuming problems

Ruby3 Goal
Parallel execution

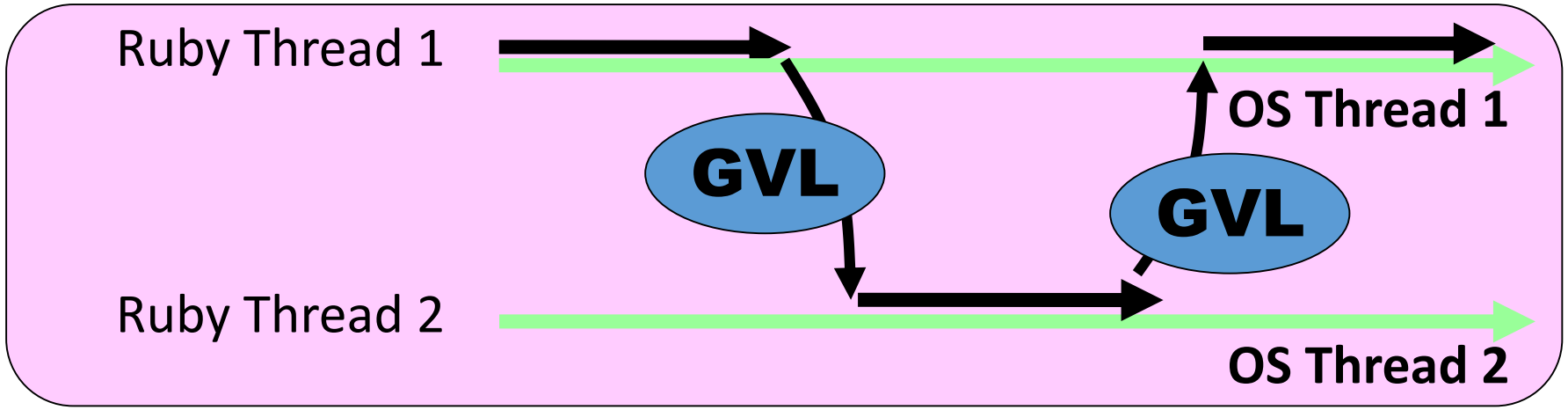
Ruby3 Goal
Parallel execution

- MRI supports **“Concurrent execution”**,
  but not support **“Parallel execution”**
  - MRI has GVL: Global (Giant) interpreter lock
  - GVL prepends to run Ruby threads in parallel



Ruby3 Goal
Parallel execution

- GVL: Advantage
  - It is enough to interleave blocking I/O operations
  - We don’t care about severe thread-safe error
  - We can continue to use existing C extensions
- GVL: Disadvantage → Can’t utilize multiple cores



Ruby3 Goal
Parallel execution

- One idea: Parallel threads
  - JRuby and Rubinius support it!
    - You can try them today!
  - So far as we write a correct thread-safe program, we can utilize multiple cores.

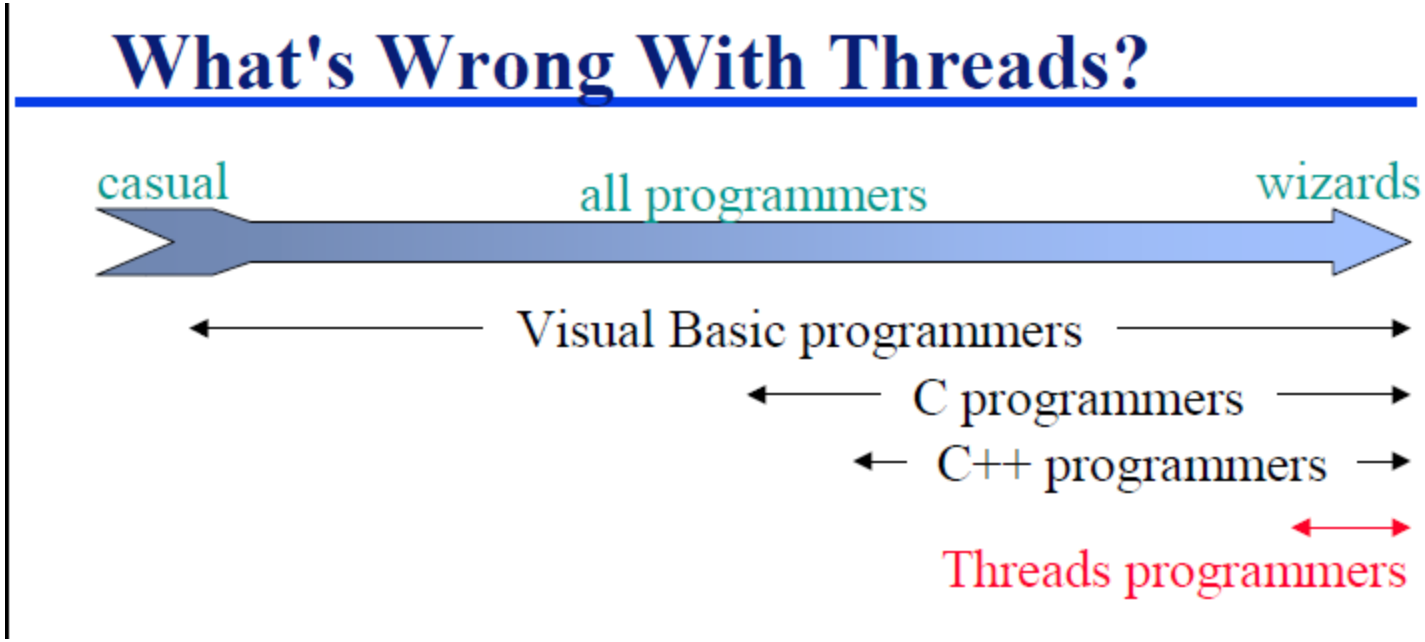
Ruby3 Goal
Parallel execution

- One idea: Parallel threads
  - JRuby and Rubinius support it!
    - You can try them today!
  - So far as we write a **correct thread-safe program**, we can utilize multiple cores.

correct thread-safe program

Ruby3 Goal
Parallel execution

- “Why Threads Are A Bad Idea (for most purposes)”
  - Quoted from John Ousterhout, 1995 (about 20 years ago ☺)



Ruby3 Goal
Parallel execution

- Writing correct thread-safe program is very hard
  - Shared everything
  - Need suitable synchronization
  - Hard to reproduce timing problems
- Many techniques are invented in 20 years
  - Synchronous queue, compare and swap primitive, …
  - Helper libraries
    - java.util.concurrent
    - ruby-concurrent
- Such techniques require “remember to use them”

Ruby3 Goal
Parallel execution

- Writing correct thread-safe program is very hard
  - Shared everything
  - Need suitable synchronization
  - Hard to reproduce timing problems
- Many techniques are invented in 20 years
  - Synchronous queue, compare and swap primitive, …
  - Helper libraries
    - java.util.concurrent
    - ruby-concurrent
- Such techniques require **“remember to use them”**

“remember to use them”

Ruby3 Goal
Parallel execution

- Two ways to avoid such difficulty

(1) Making smart thread-safe debugger

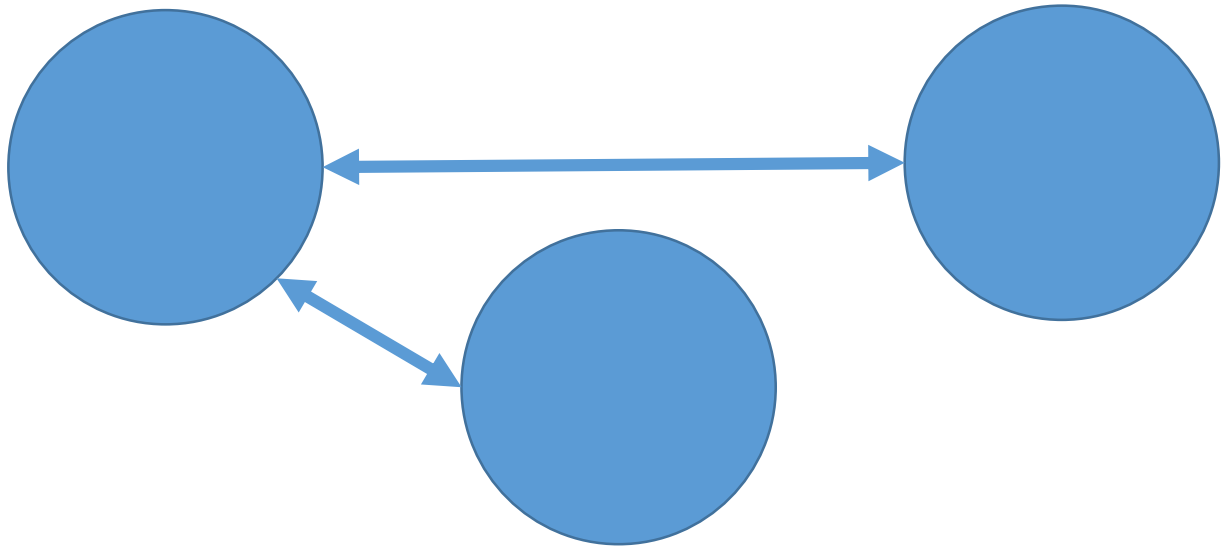
(2) Introduce higher abstraction for concurrency

Ruby3 Goal
Parallel execution

- (1) Making smart thread-safe debugger
  - Example) Pointing out lack of locks
  - Well researched by academic area and used in production
- Continue to use threads, shared everything model
  - Good
    - Well-known model
    - No overhead
  - Bad
    - We need to debug it
    - We can’t guarantee 100% coverage

Ruby3 Goal
Parallel execution

- (2) Introduce higher abstraction for concurrency
- Introduce new isolated concurrent execution entities
  - Run entities in parallel and communicate with each other
  - Do not mutate anything simultaneously



Ruby3 Goal
Parallel execution

- (2) Introduce higher abstraction for concurrency
- Introduce new isolated concurrent execution entities
  - Run entities in parallel and communicate with each other
  - Like “Processes” by “fork”
  - Good
    - Shared-nothing → we don’t need to care about thread-safety (problems of concurrent programming such as dead-lock are remain)
  - Bad
    - Introduce overhead
    - Learning cost for new abstraction

Ruby 3: Ruby 3 has 3 goals

- For productivity
  - Static type checking
- For performance
  - Just-in-Time (JIT) compilation
  - Parallel execution w/ highly abstract concurrent model

Message

- We are discussing about Ruby3 just now
- Any suggestions and implementations are welcome!

It may be your turn!

Time to make our future of Ruby

Thank you for your attention

Koichi Sasada
<ko1@heroku.com>

salesforce heroku