Fiber in the 10th year

Koichi Sasada
ko1@cookpad.com

cookpad

About this talk

- Behavior of Fiber
- History of Fiber
- Implementation of Fiber
- Auto Fiber proposal

Koichi Sasada

http://atdot.net/~ko1/

- A programmer
  - 2006-2012 Faculty
  - 2012-2017 Heroku, Inc.
  - 2017- Cookpad Inc.
- Job: MRI development
  - Core parts
    - VM, Threads, GC, etc

cookpad

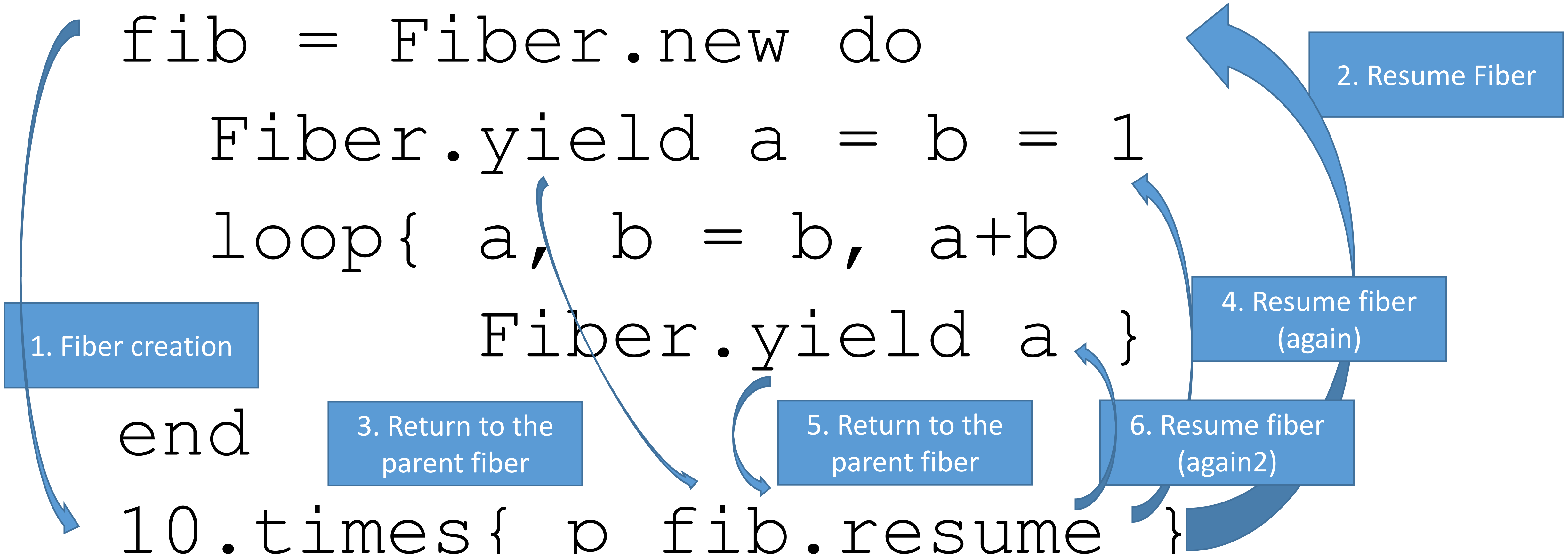
Fiber
User-defined context switching

Fiber example
Infinite generator

```
fib = Fiber.new do
  Fiber.yield a = b = 1
  loop{ a, b = b, a+b
        Fiber.yield a }
end
10.times{ p fib.resume }
```

Fiber example
Infinite generator

```
fib = Fiber.new do
  Fiber.yield a = b = 1
  loop{ a, b = b, a+b
        Fiber.yield a }
end
10.times{ p fib.resume }
```



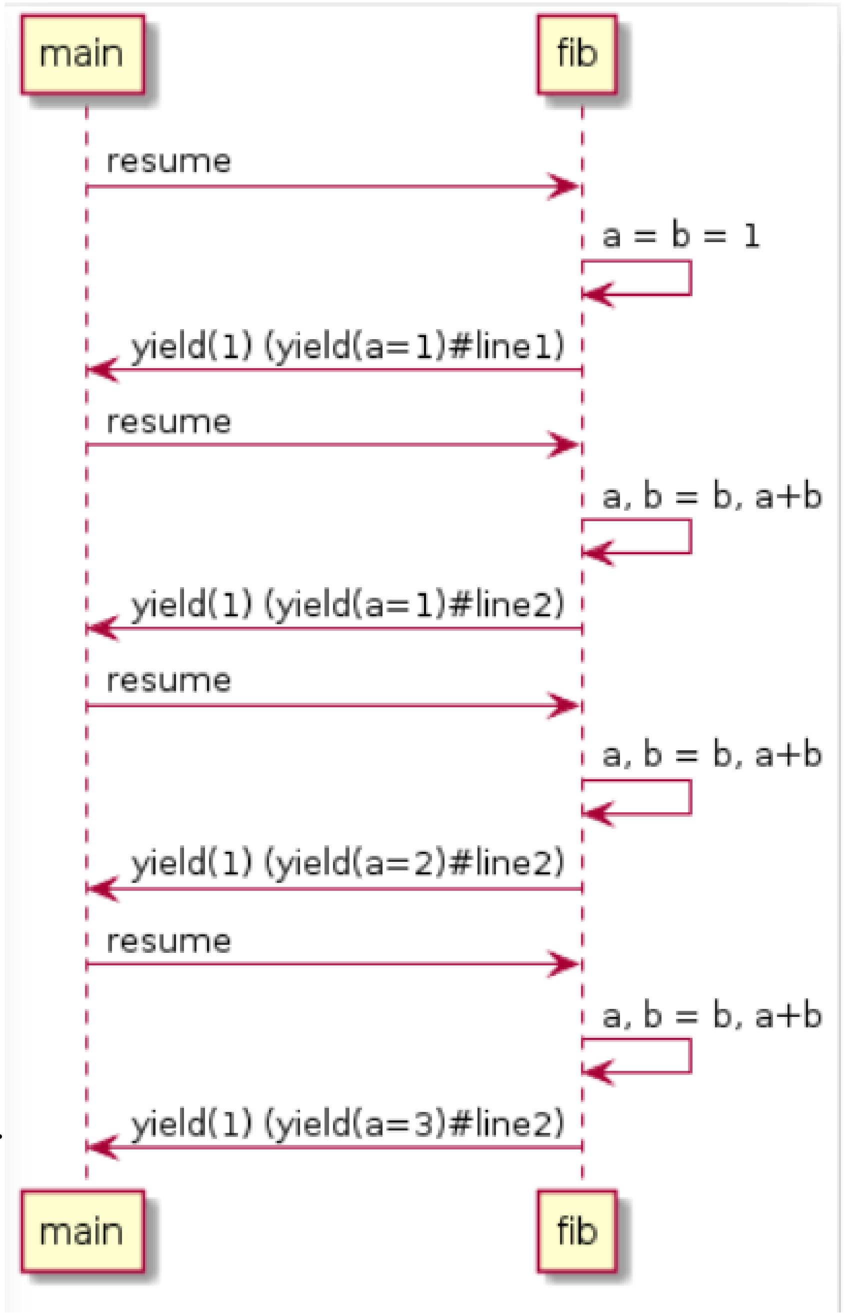
Fiber example
Infinite generator

```
fib = Fiber.new do
  Fiber.yield a = b =
  loop{ a, b = b, a+b
        Fiber.yield a
end
10.times{ p fib.resum
```

1. Fiber creation

3. Return to the parent fiber

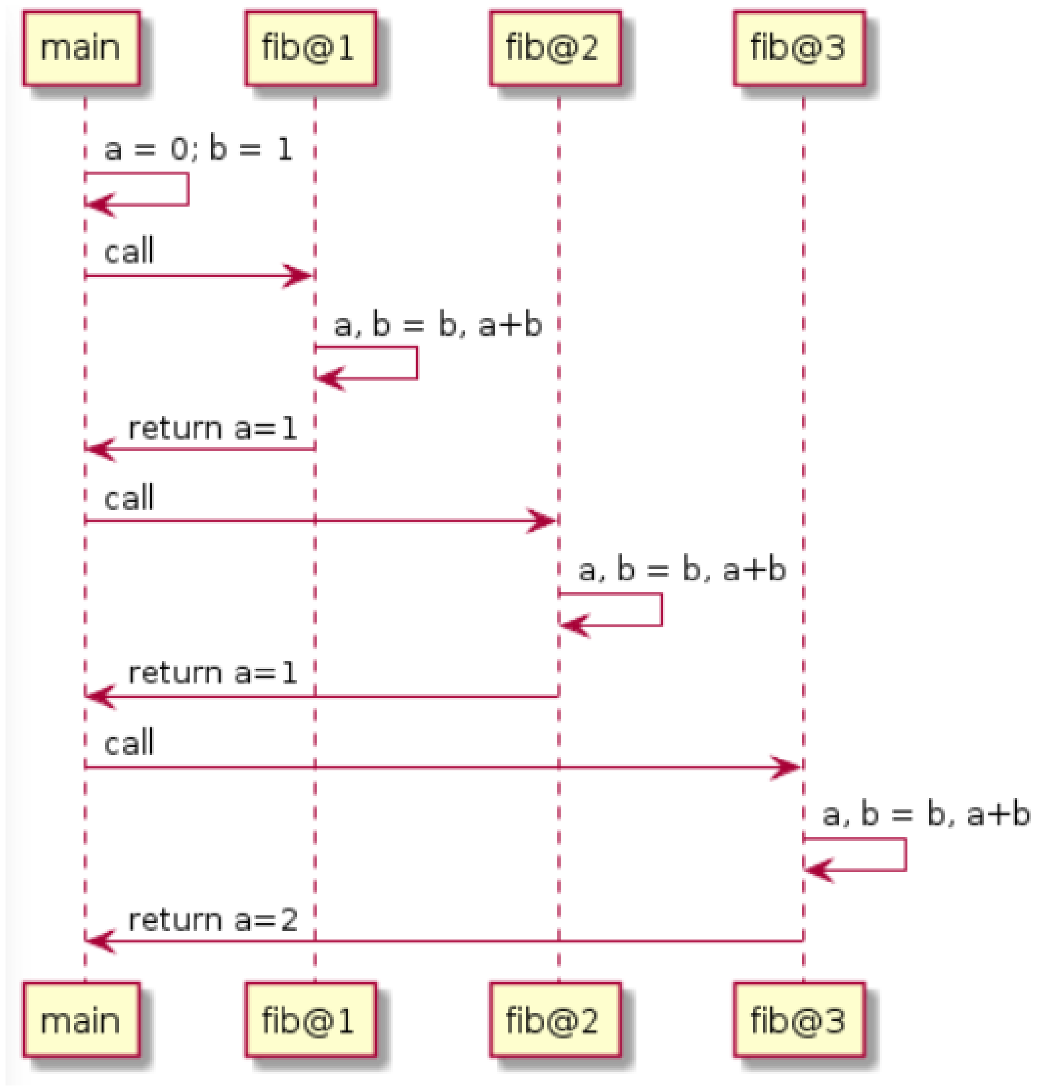
5. Return to the parent fiber


main
fib
resume
a = b = 1
yield(1) (yield(a=1)#line1)
resume
a, b = b, a+b
yield(1) (yield(a=1)#line2)
resume
a, b = b, a+b
yield(1) (yield(a=2)#line2)
resume
a, b = b, a+b
yield(1) (yield(a=3)#line2)
main
fib

Not a Proc?

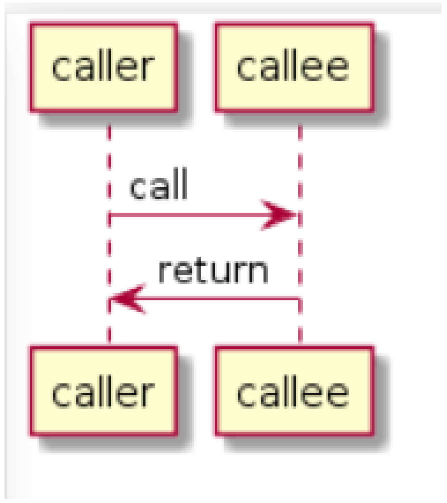
```
a = 0; b = 1
fib = Proc.new{
  a, b = b, a+b
  a
}
p fib.call #=> 1
p fib.call #=> 1
p fib.call #=> 2
p fib.call #=> 3
p fib.call #=> 5
```

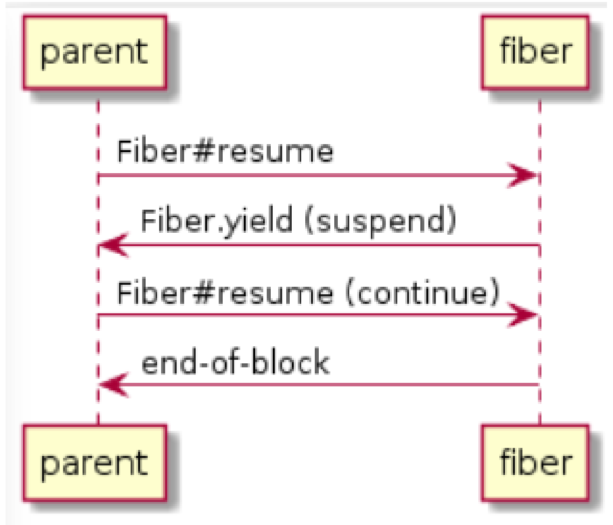
Proc can’t restart from the middle of block



Proc (method) v.s. Fiber

	Proc (method)	Fiber
Start	OK: call	OK: Fiber#resume
Parameters	OK: block (method) parameters	OK: block parameters
Return	OK: exit Proc/method	OK: exit Proc/method
Suspend	NG: N/A	**OK: Fiber.yield**
Continue	NG: N/A	**OK: Fiber#resume**





Fiber example
Inner iterator to external iterator

```
f1 = Fiber.new do
  2.times{|i| Fiber.yield i}
end

p f1.resume #=> 0
p f1.resume #=> 1
p f1.resume #=> 2 # return value of #times
p f1.resume #=> dead fiber called
                (FiberError)
```

Fiber example
Inner iterator to external iterator

```
etc_passwd_ex_iter = Fiber.new do
  open('/etc/passwd').each_line{|line|
    Fiber.yield line
  }
end
p etc_passwd_ex_iter.resume #=> 1st line
p etc_passwd_ex_iter.resume #=> 2nd line
…
```

Fiber example
Inner iterator to external iterator

```
# make Enumerator
iter = open('/etc/passwd').each_line

# Enumerator#next use Fiber implicitly
p iter.next #=> 1st line
p iter.next #=> 2nd line
…
```

Fiber example
Agent simulation

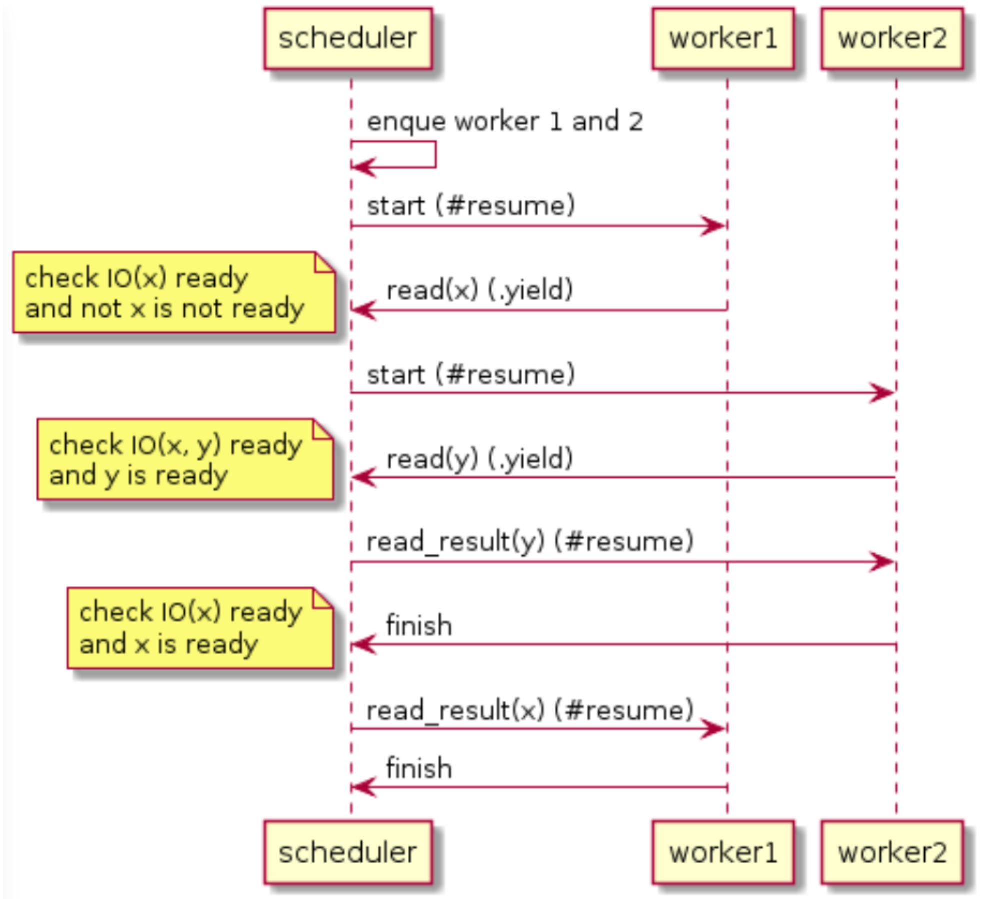
```
characters << Fiber.new{
  loop{cat.move_up; Fiber.yield}}
characters << Fiber.new{
  loop{dog.move_left; Fiber.yield}}
…
loop{cs.each{|e| e.resume}; redraw}
```

Fiber example
Agent simulation

```
characters << Fiber.new{
  # you can specify complex rule for chars
  loop{
    cow.move_up;    Fiber.yield
    cow.move_right; Fiber.yield
    cow.move_down;  Fiber.yield
    cow.move_left;  Fiber.yield
  }
}
```

Fiber example
Non-blocking IO scheduler

Wait multiple IO ops with traditional “select” or modern “poll”, “epoll” interface



Not a Thread?

	Thread	**Fiber**
Suspend/continue	**Yes**	**Yes**
Switch on timer	**Yes**	No (explicit switch)
Switch on I/O blocking	**Yes**	No (explicit switch)
Synchronization	Required	**Not required**
Specify next context	No	**Yes**
Performance: Creation	Heavy	Lightweight
Performance: Switch	Lightweight	Heavy (initial version) Lightweight (now)

Brief History of Fibers

Fiber: Brief history

- 2007/05/23 cont.c (for callcc)
- 2007/05/25 Fiber impl. [ruby-dev:30827]
- 2007/05/28 Fiber introduced into cont.c
- 2007/08/25 Fix Fiber spec

Background: Callcc and Fiber on Ruby 1.9

- 2007/01 YARV was merged without “callcc”
- Biggest usage of “callcc” is for “Generator”
  - Convert an internal iterator to an external iterator
  - Usually one-shot continuation is required
    → Coroutine is enough for this purpose
  - Capturing continuation (callcc) is heavy operation
  - Implementation is easy because we can refer Ruby 1.8 user-level threads
- 2007/05/?? I was introduced one paper something like generator for (maybe) C# (so I began to consider about this feature)
  - And I have a spare time at academic conference

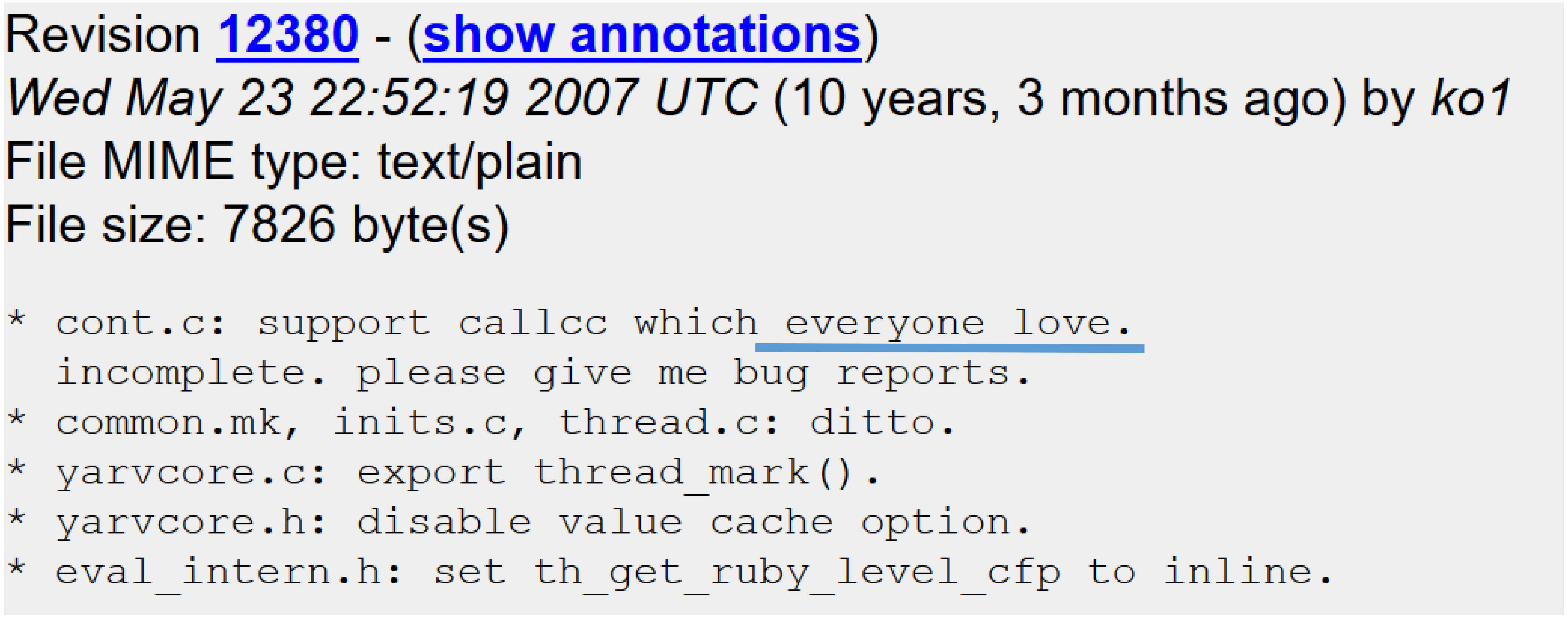
2007/05/22 IRC log

(seeing a blog post)

```
00:56:49 <ko1> うーむ，callcc 欲しいっすか
```

English: Umm, do you want “callcc”?

2007/05/23 cont.c



Revision 12380 - (show annotations)
Wed May 23 22:52:19 2007 UTC (10 years, 3 months ago) by *ko1*
File MIME type: text/plain
File size: 7826 byte(s)

```
* cont.c: support callcc which everyone love.
  incomplete. please give me bug reports.
* common.mk, inits.c, thread.c: ditto.
* yarvcore.c: export thread_mark().
* yarvcore.h: disable value cache option.
* eval_intern.h: set th_get_ruby_level_cfp to inline.
```

2007/05/23 IRC log

(nobu pointed out there are several bugs on callcc)

12:15:36 <ko1> callcc 禁止でいいよ

EN: callcc should be prohibited

12:15:52 <ko1> これ作りながら，Fiber作ったほうが 速いなーとか思って亜

EN: Building callcc, I’m thinking that making Fiber is more straightforward.

Fiber naming

- The name “Fiber” is from Windows API
  - “A *fiber* is a unit of execution that must be manually scheduled by the application. Fibers run in the context of the threads that schedule them. Each thread can schedule multiple fibers. In general, fibers do not provide advantages over a well-designed multithreaded application. However, using fibers can make it easier to port applications that were designed to schedule their own threads.” https://msdn.microsoft.com/ja-jp/library/windows/desktop/ms682661(v=vs.85).aspx

[ruby-dev:30828] Re: Supporting Fiber Naming of Fiber

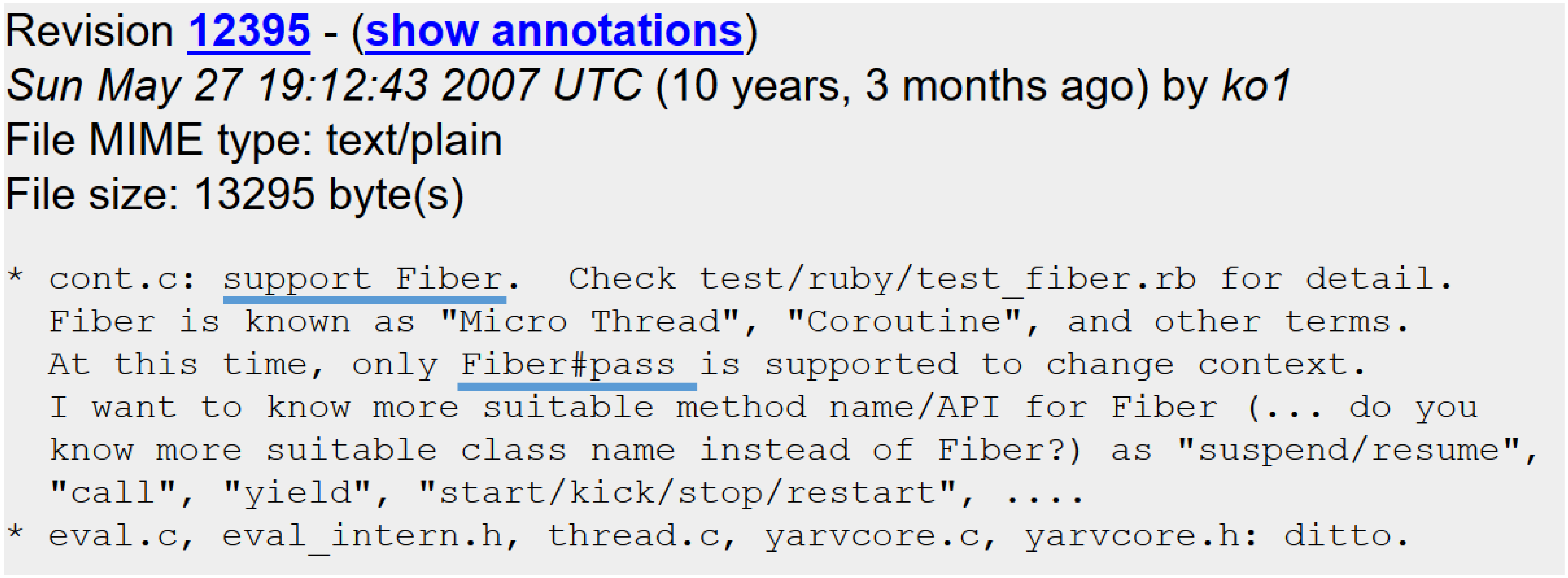
“Fiberでいいんじゃないでしょうか。

何かかっこいいですよね。” by shugo

EN: “I’m ok the name of “Fiber”.

Somewhat cool.” by shugo

2007/05/28 Introduction r13295, [ruby-dev:30827]



Revision 12395 - (show annotations)
Sun May 27 19:12:43 2007 UTC (10 years, 3 months ago) by *ko1*
File MIME type: text/plain
File size: 13295 byte(s)

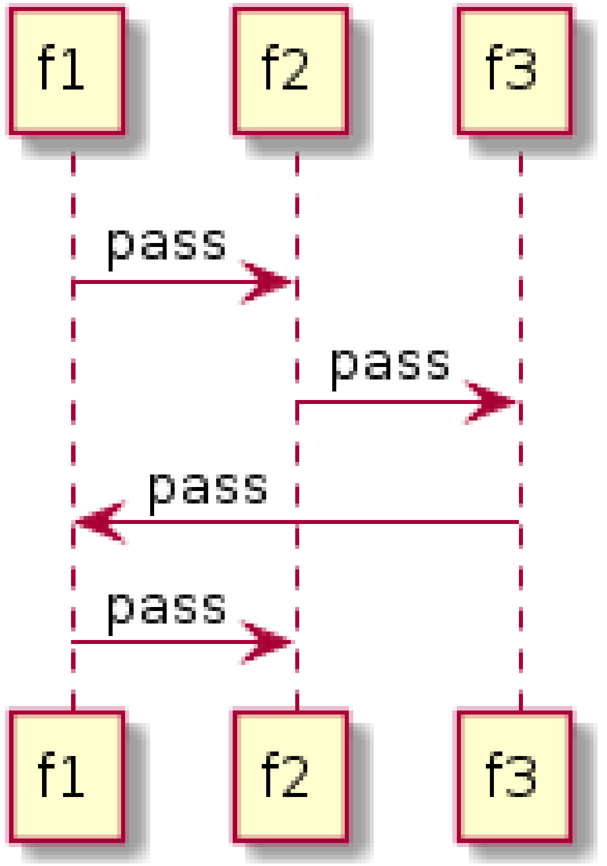
```
* cont.c: support Fiber.  Check test/ruby/test_fiber.rb for detail.
  Fiber is known as "Micro Thread", "Coroutine", and other terms.
  At this time, only Fiber#pass is supported to change context.
  I want to know more suitable method name/API for Fiber (... do you
  know more suitable class name instead of Fiber?) as "suspend/resume",
  "call", "yield", "start/kick/stop/restart", ....
* eval.c, eval_intern.h, thread.c, yarvcore.c, yarvcore.h: ditto.
```

First Fiber is Coroutine Fiber#pass

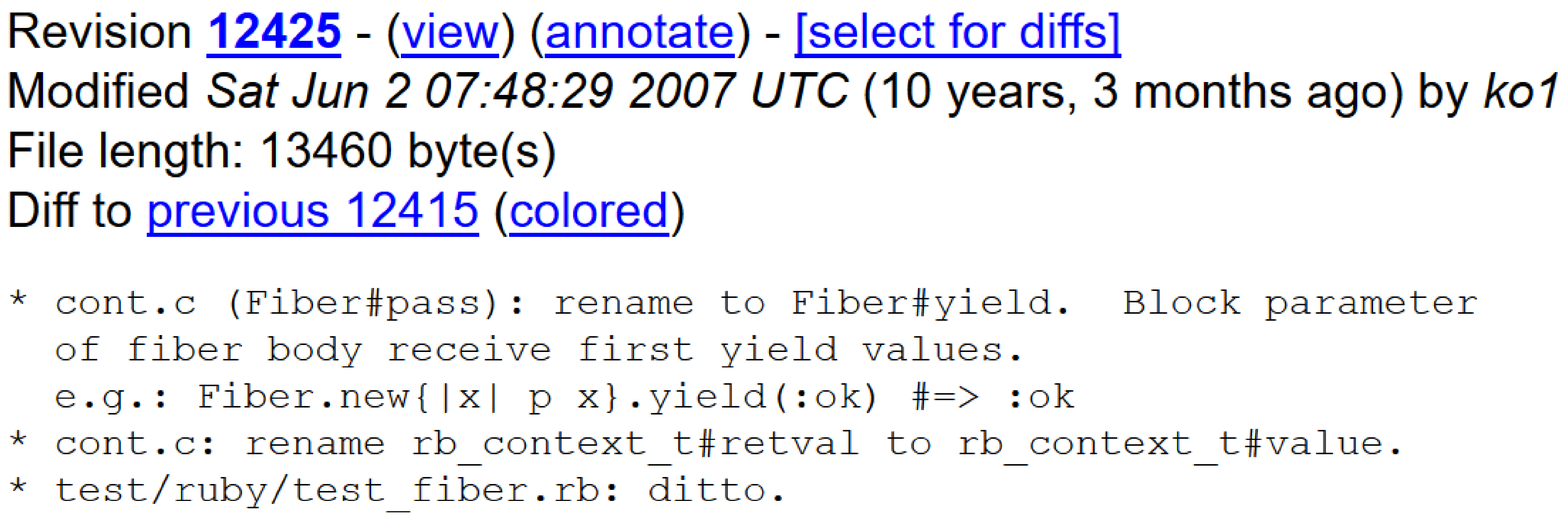
```
f1 = Fiber.new{
  f2.pass; f2.pass}
f2 = Fiber.new{
  f3.pass}
f3 = Fiber.new{
  f1.pass}
f1.pass
```

NOTE: renamed to “Fiber#transfer” now

No parents/children
All routines are equivalent
Co-operative routines
= Coroutine



Fiber#pass → Fiber#yield
[ruby-dev:30847] Re: Supporting Fiber



Revision **12425** - (view) (annotate) - [select for diffs]
Modified *Sat Jun 2 07:48:29 2007 UTC* (10 years, 3 months ago) by *ko1*
File length: 13460 byte(s)
Diff to previous 12415 (colored)

```
* cont.c (Fiber#pass): rename to Fiber#yield.  Block parameter
  of fiber body receive first yield values.
  e.g.: Fiber.new{|x| p x}.yield(:ok) #=> :ok
* cont.c: rename rb_context_t#retval to rb_context_t#value.
* test/ruby/test_fiber.rb: ditto.
```

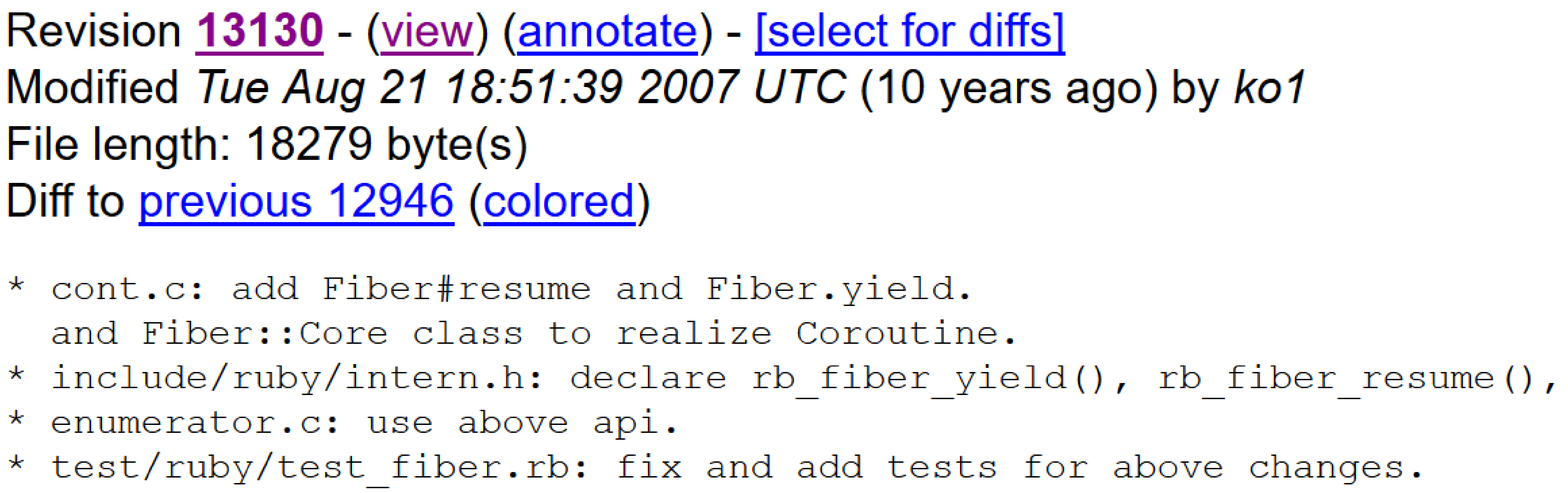
Matz's idea

Coroutine or Semi-coroutine

- Coroutine is difficult
  - You need to manage all transitions of Fibers
    - Remember that most of languages have only “routine” (not “co-”) and it is easy to use.
    - Most of case, semi-coroutine is easy and enough
  - Exception handling
    - On semi-croutine, exceptions are raised to the parent Fiber(s)
  - Maybe it has critical BUG issue.
- Coroutine is powerful
  - No limitation (a bit old-language constructs)

[ruby-dev:31583] Fiber reviesed

Semi-coroutine (Fiber) and Coroutine (Fiber::Core)



Revision 13130 - (view) (annotate) - [select for diffs]
Modified *Tue Aug 21 18:51:39 2007 UTC* (10 years ago) by *ko1*
File length: 18279 byte(s)
Diff to previous 12946 (colored)

```
* cont.c: add Fiber#resume and Fiber.yield.
  and Fiber::Core class to realize Coroutine.
* include/ruby/intern.h: declare rb_fiber_yield(), rb_fiber_resume(),
* enumerator.c: use above api.
* test/ruby/test_fiber.rb: fix and add tests for above changes.
```

2007/08/25 IRC log

10:26:49 <ko1> 大クラス主義ならFiber に Semi も Coroutine も機能いっしょくたにするべきかなあ

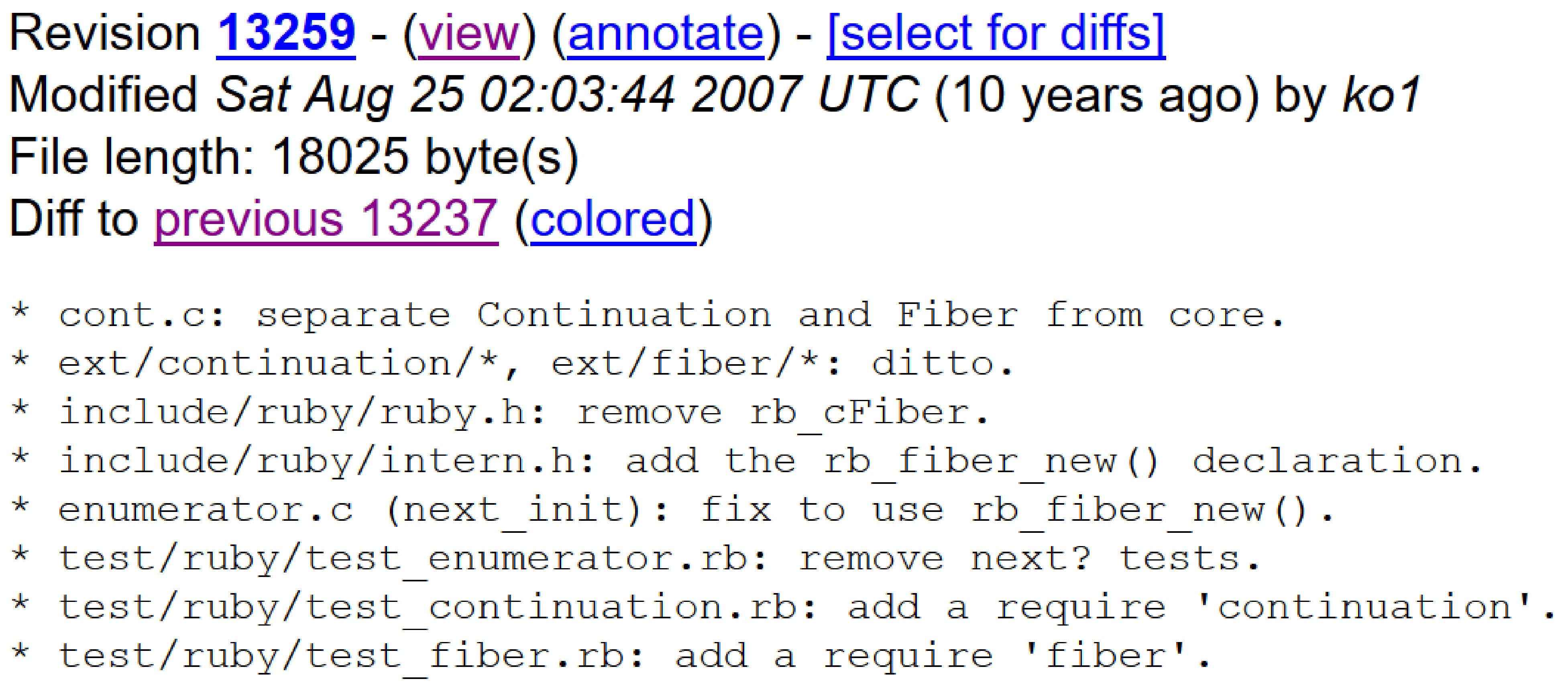
EN: Semi- and non-semi Coroutine may be
in one class undr big class principle

10:32:15 <ko1> というわけで，いっしょくたにしてみる

EN: So that I merged it.

* It was just idea in two lines…

Fiber::Core was removed



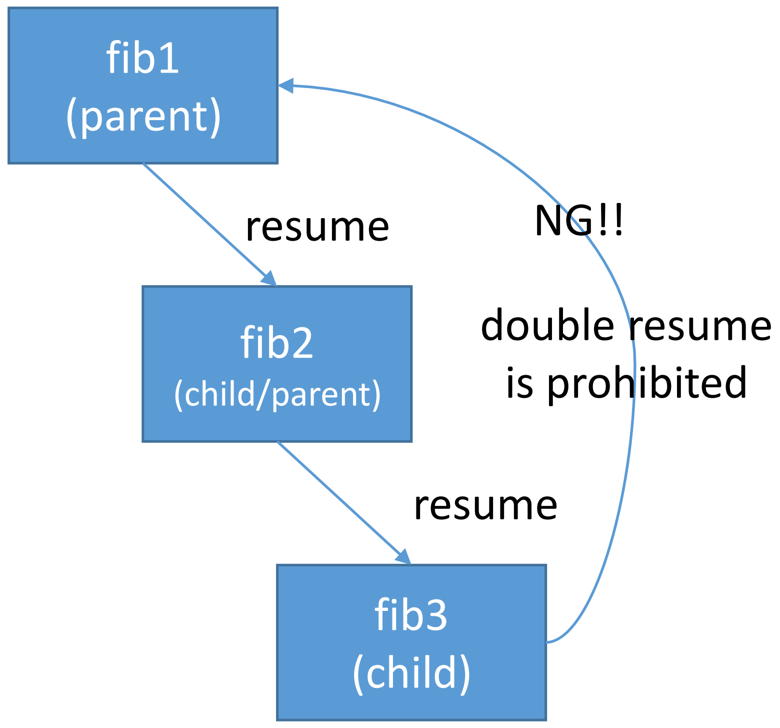
Revision 13259 - (view) (annotate) - [select for diffs]
Modified *Sat Aug 25 02:03:44 2007 UTC* (10 years ago) by *ko1*
File length: 18025 byte(s)
Diff to previous 13237 (colored)

```
* cont.c: separate Continuation and Fiber from core.
* ext/continuation/*, ext/fiber/*: ditto.
* include/ruby/ruby.h: remove rb_cFiber.
* include/ruby/intern.h: add the rb_fiber_new() declaration.
* enumerator.c (next_init): fix to use rb_fiber_new().
* test/ruby/test_enumerator.rb: remove next? tests.
* test/ruby/test_continuation.rb: add a require 'continuation'.
* test/ruby/test_fiber.rb: add a require 'fiber'.
```

Commit message does not work well…

Final specification of Fiber

- Semi-coroutine
  - Fiber#resume and Fiber.yield
  - Make parent and child relationship (tree)
  - Prohibit double resume
- Coroutine
  - Fiber#transfer
  - Prohibt to call semi-coroutine methods on “transfer”ed fiber (coroutine)



Implementation of Fibers

Implementation history

(1) 2007/05 Copy all machine stack

(2) 2010/05 FIBER_USE_NATIVE

(3) 2017/09 Switch only pointer

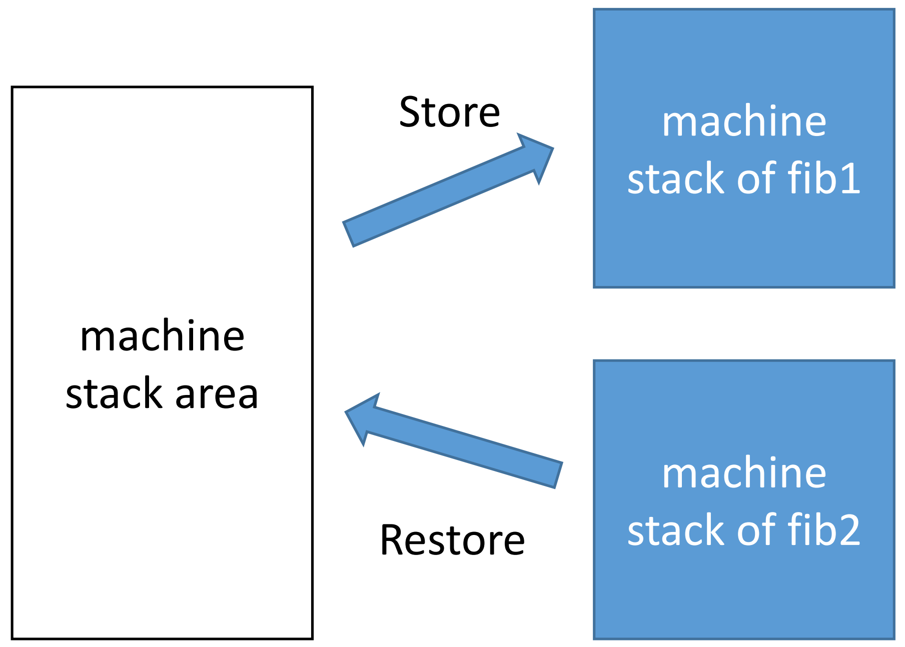
Fiber context representation

- Context:
  - Thread states (current program counter, etc)
  - VM stack
  - Machine stack
- “Context switching” means exchange contexts

Fiber implementation 2007 (1) Copy machine stack

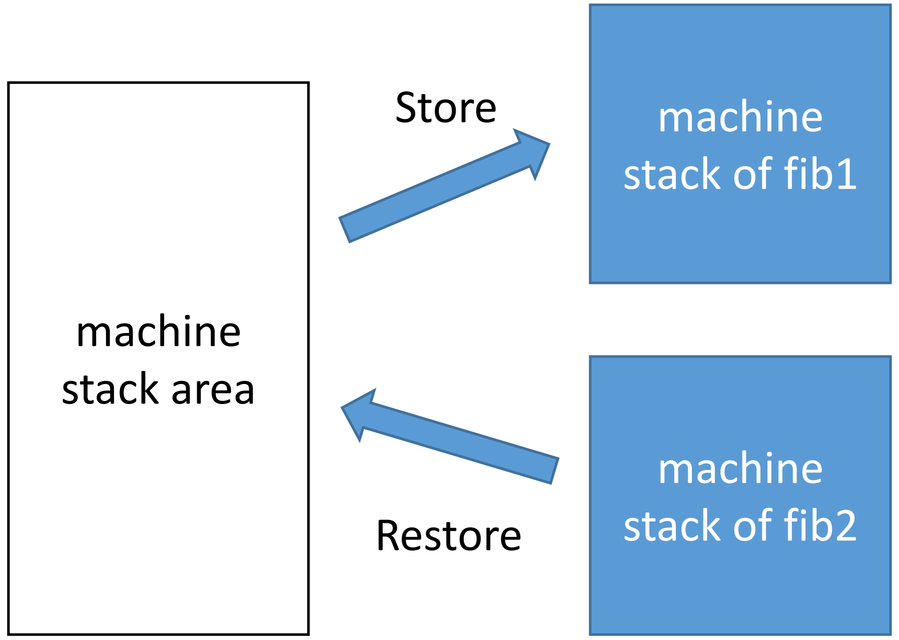
- Store and restore “Context” by copying machine stack

Switch from running fib1 to suspended fib2



Fiber implementation
2007 (1) Copy machine stack

- Good
  - Same idea of a Ruby 1.8 user-level thread code
  - Not so many memory usage
  - Almost portable
- Bad
  - Copy time is relative to stack-depth (O(N))



Fiber implementation
2010 (2) Use Native support

- Switch machine stack by system APIs
    - Supported APIs
        - POSIX makecontext/setcontext
        - Win32 Fiber API
    - Machine stack exchange is only pointer exchange (O(1))
- Implemented by Mr. Shiba (with me)

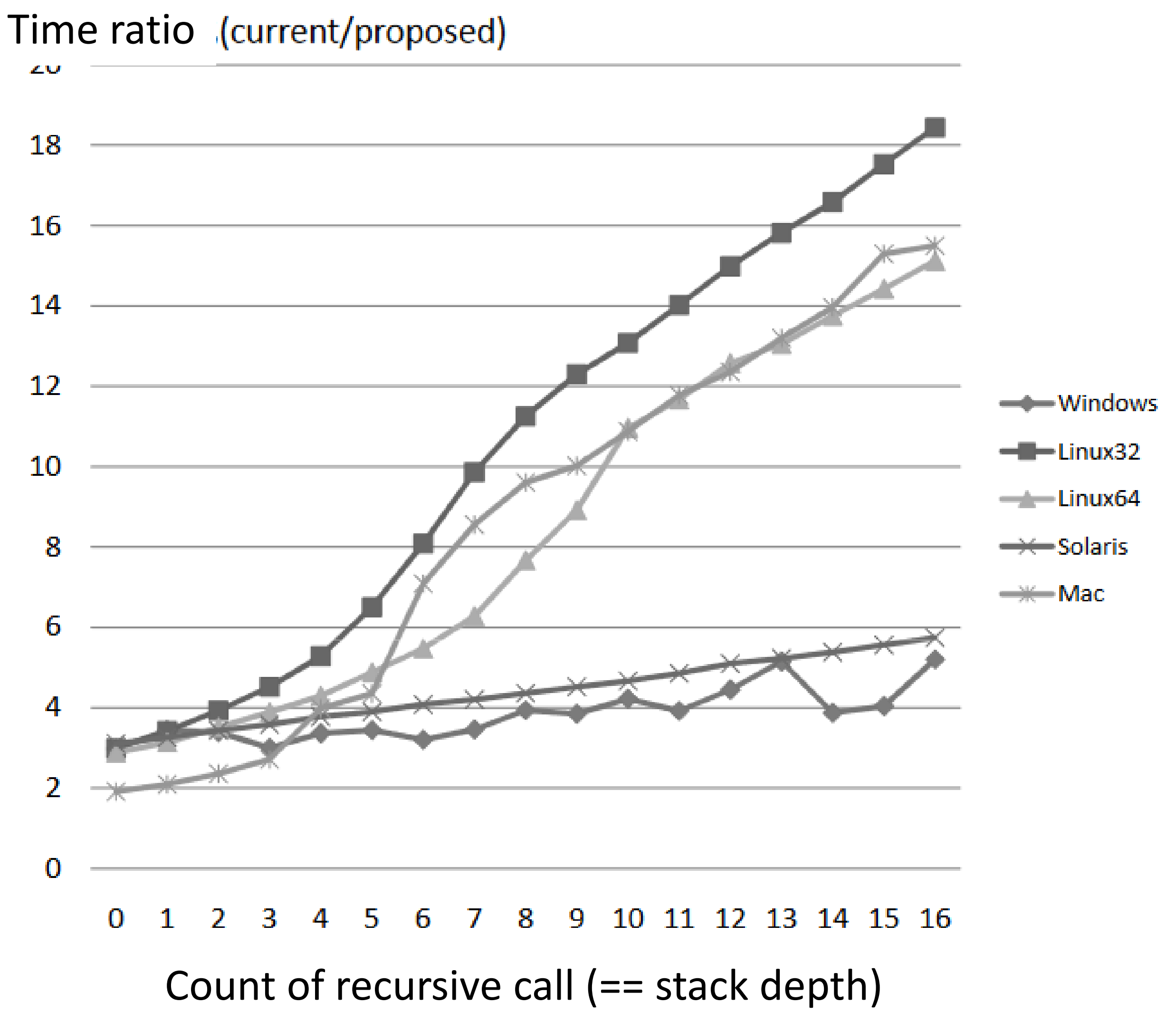


図 5　Fiber 間コンテキストスイッチのベンチマーク結果

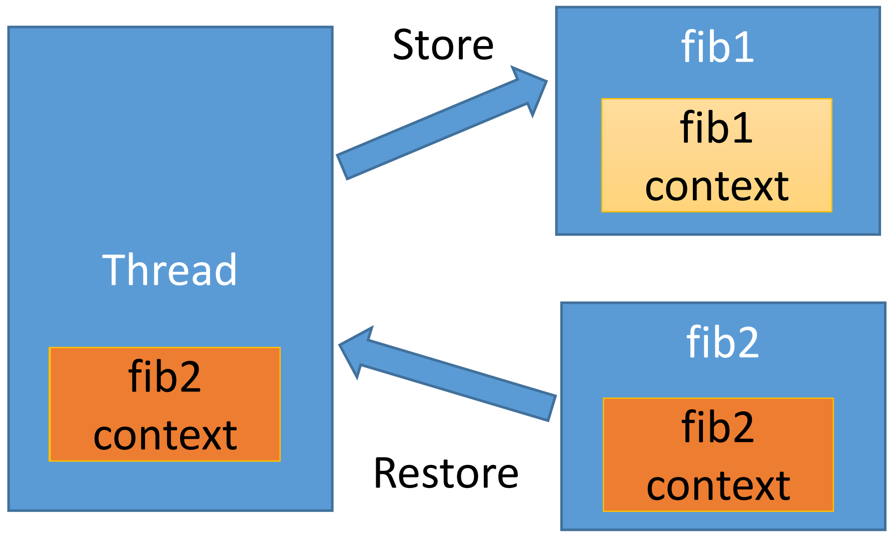
"A Fast Fiber Implementation for Ruby 1.9"
"Ruby1.9での高速なFiberの実装",
第51回プログラミング・シンポジウム予稿集, pp.21--28 (2010).

Fiber implementation 2017 (3) More lightweight switching

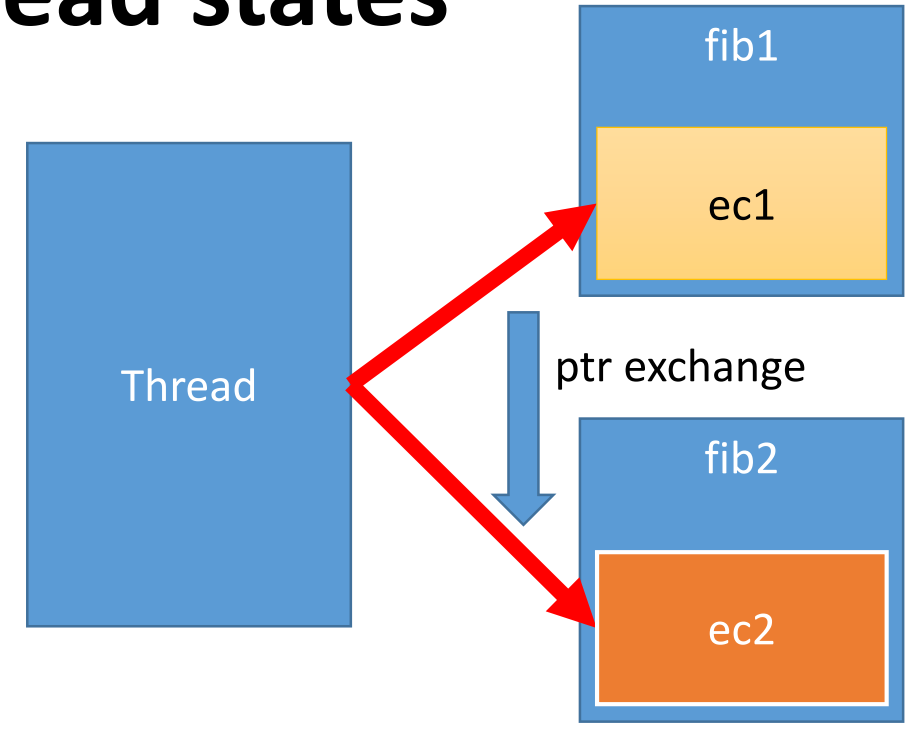
- Context exchange
  - **[copy] Thread states**
  - [ptr exchange] VM stack
  - [ptr exchange] Machine stack
- “setcontext” calls sigprocmask
  - Ruby threads/fibers use same signal mask
    → Useless system call

Fiber implementation
2017 (3) More lightweight switching

- Context exchange
  - **[copy->ptr exchange] Thread states**



Before



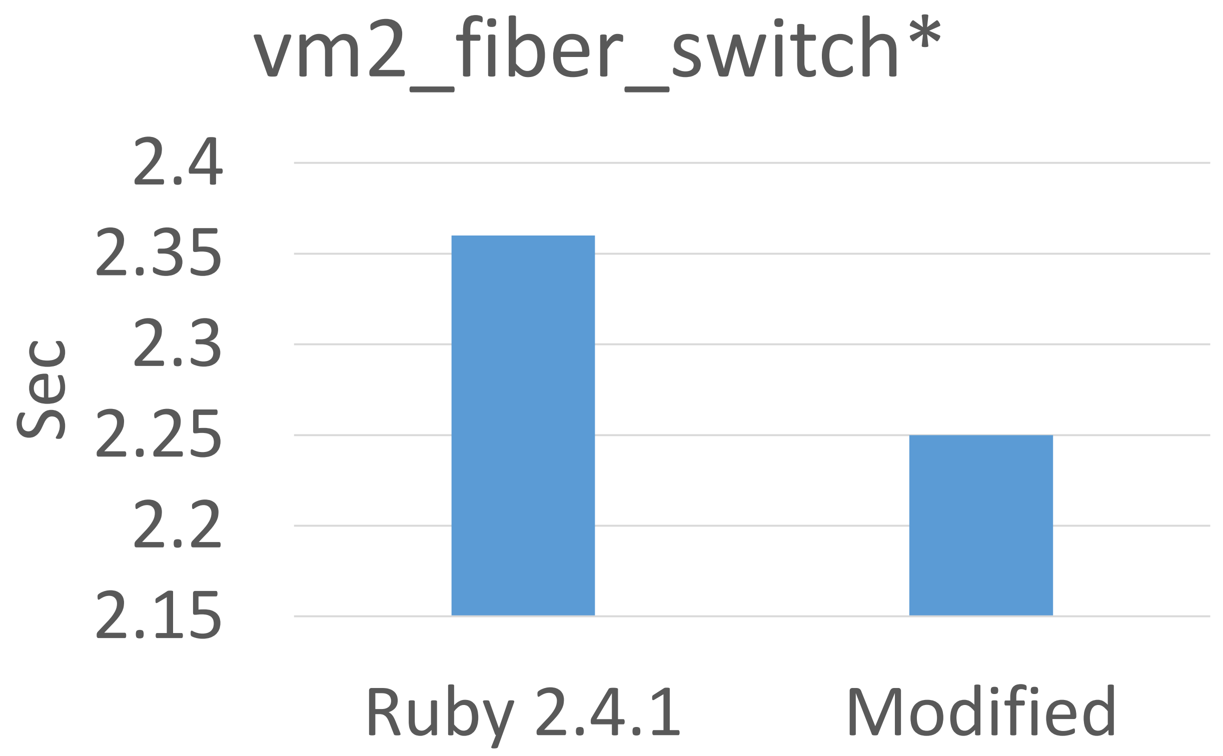
After

Fiber implementation 2017 (3) More lightweight switching

- [Futurework] Use custom “setcontext” excludes sigprocmask
  - setcontext issues “sigprocmask” system call to restore signal mask, but MRI doesn’t change signalmask so that it is completely useless.
  - This idea is also proposed at https://rethinkdb.com/blog/making-coroutines-fast/
  - License?

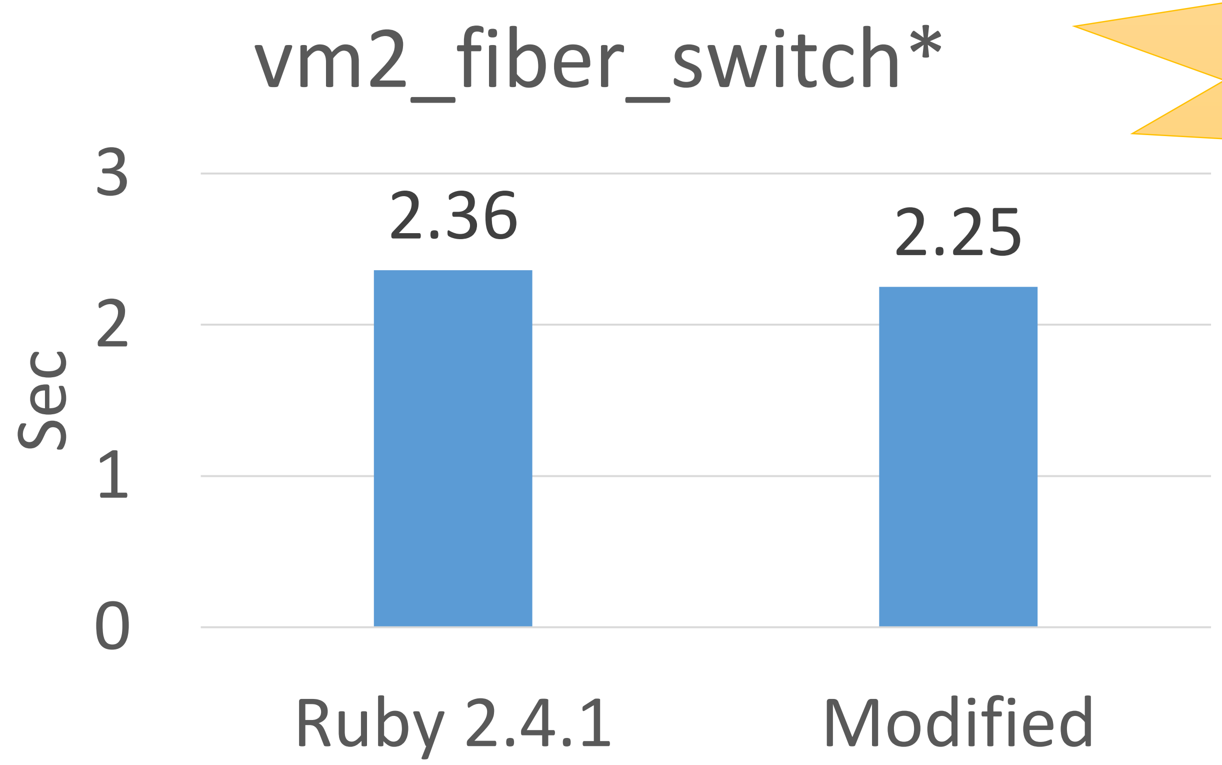
Fiber implementation
2017 (3) More lightweight switching

- Performance



Fiber implementation 2017 (3) More lightweight switching

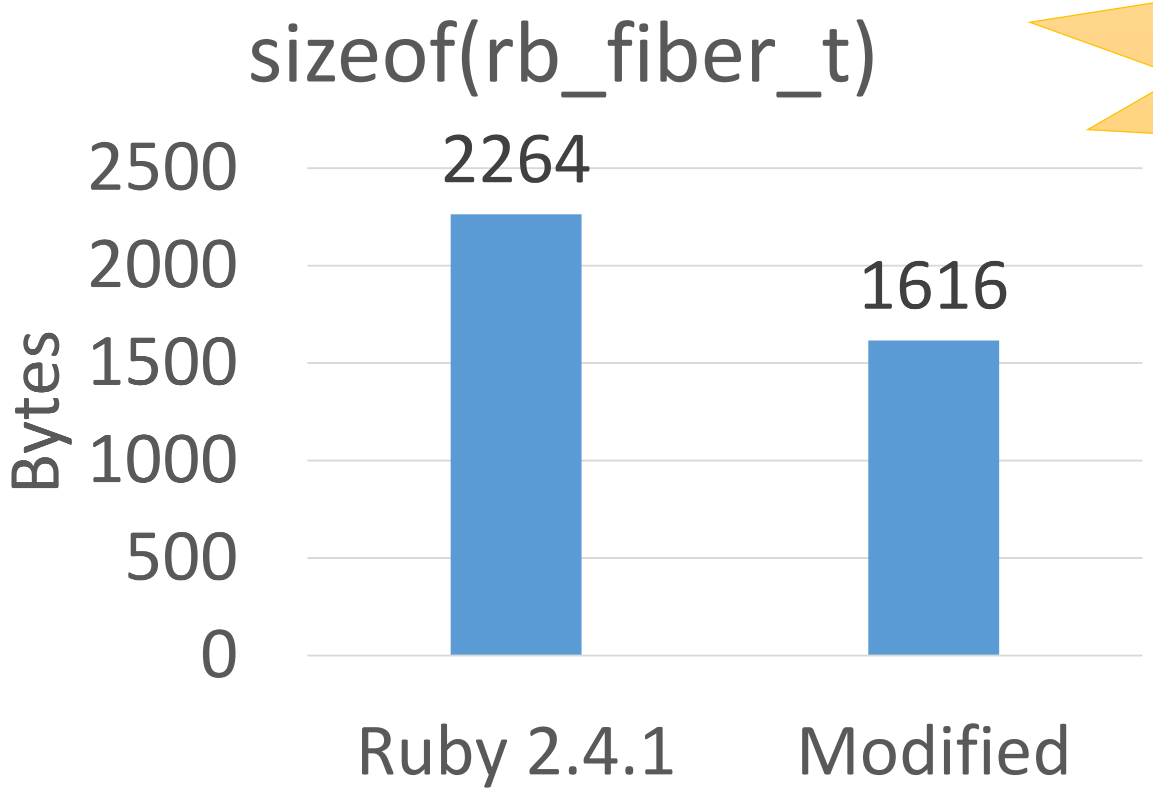
- Performance



5% improvement!

Fiber implementation 2017 (3) More lightweight switching

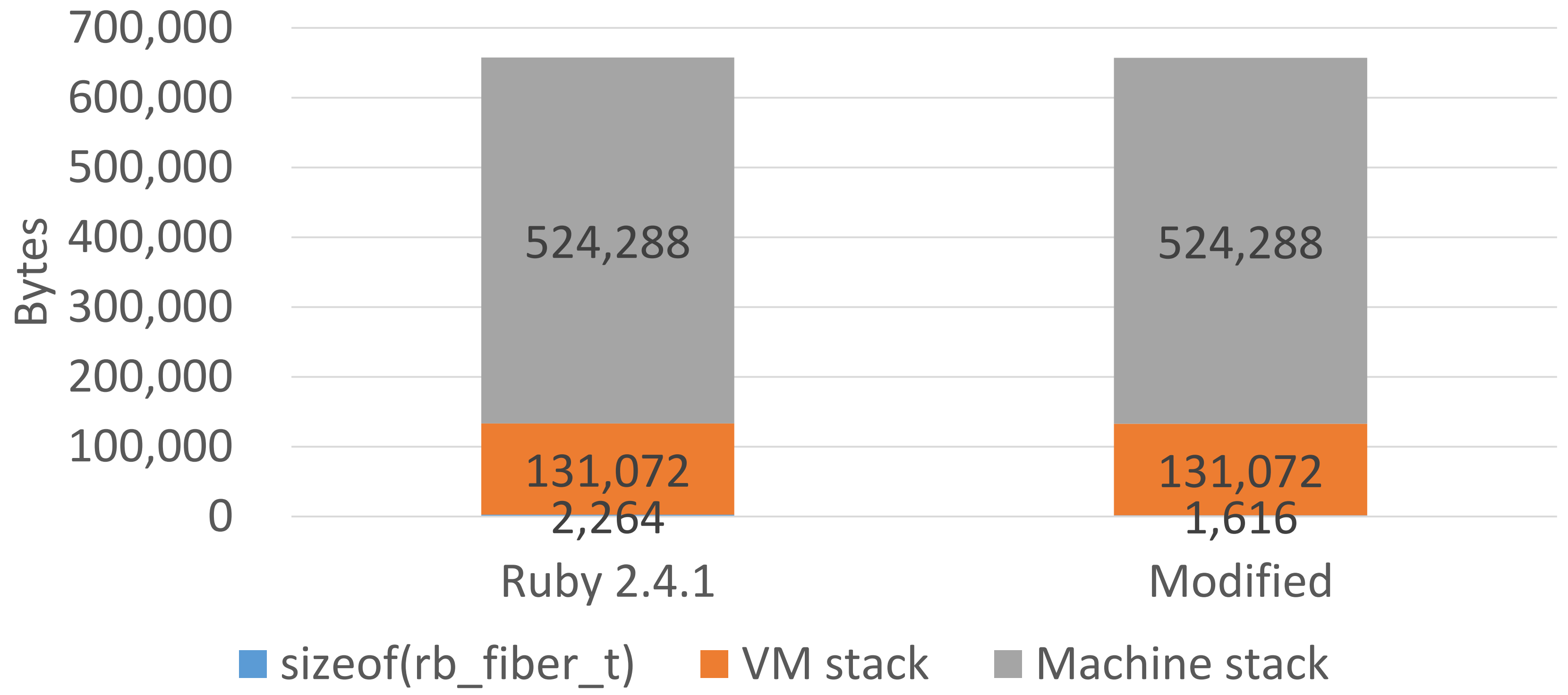
- Memory size / fiber



30% reduced!

Fiber implementation 2017 (3) More lightweight switching

- Memory size / fiber



2017 (3) More lightweight switching Not a valuable work?

- I spent this hack 2 or 3 months because of code complicity.
- This work (hopefully) will be a basis of Guild work (we need to pass context information for each APIs like mrb_state on mruby)

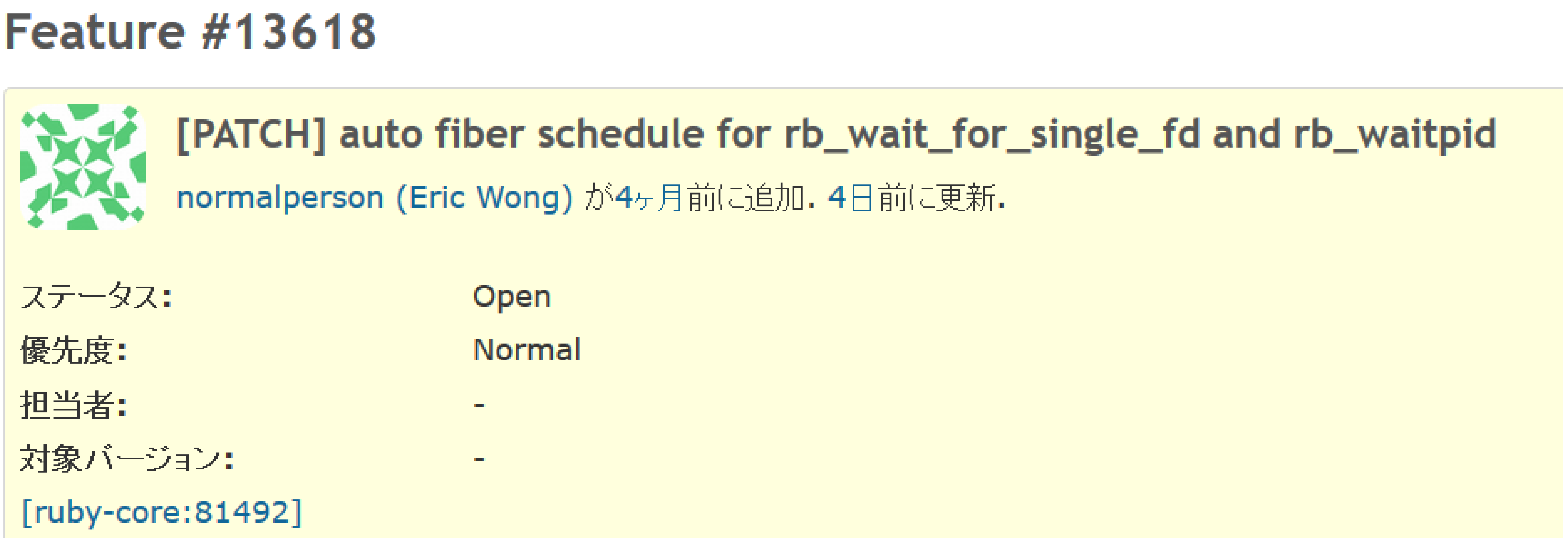
Auto-Fiber proposal

Auto Fiber proposal

- “Fiber” enables writing scheduler by Ruby programmer
    - Maybe Seki-san introduce one example
- Why doesn’t an interpreter support it natively? → Auto Fiber proposal

Auto Fiber proposal

https://bugs.ruby-lang.org/issues/13618



Feature #13618

[PATCH] auto fiber schedule for rb_wait_for_single_fd and rb_waitpid

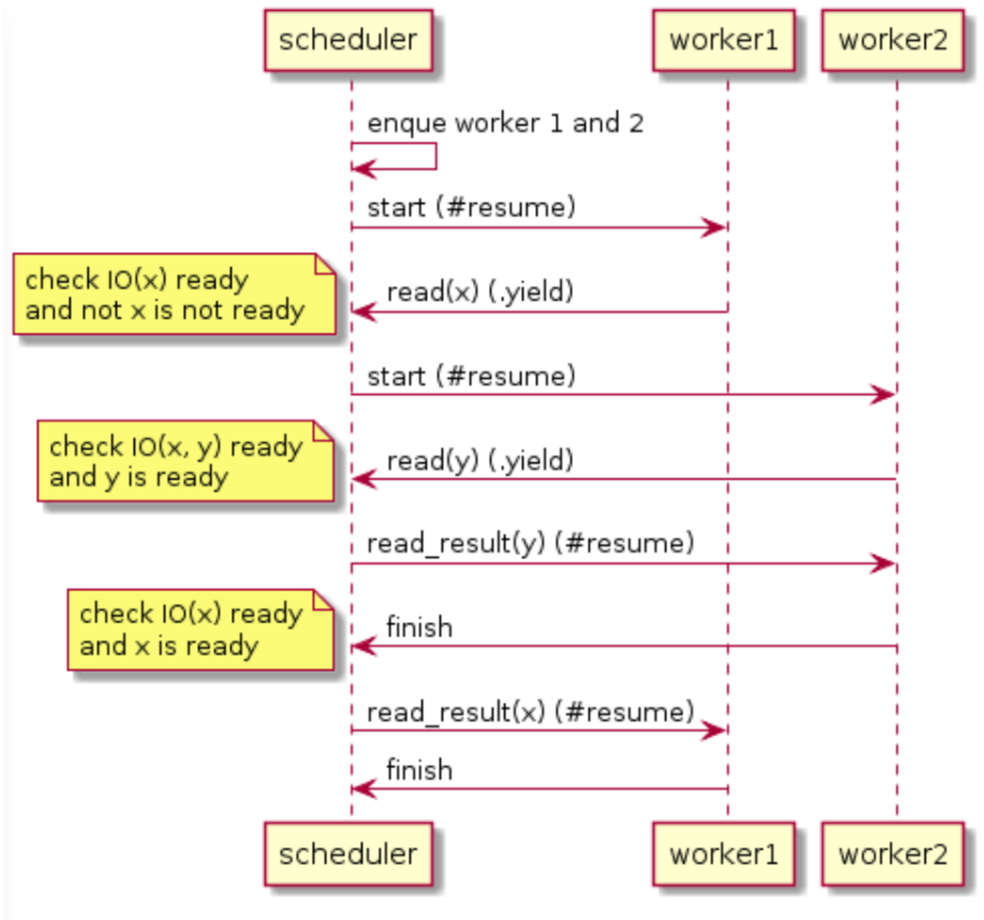
normalperson (Eric Wong) が4ヶ月前に追加. 4日前に更新.

ステータス:	Open
優先度:	Normal
担当者:	-
対象バージョン:	-

[ruby-core:81492]

Auto Fiber proposal
Automatic schedule on I/O blocking

- Support Fiber scheduler natively
  - Don’t need to return scheduler
- Switch Fibers on all blocking I/O (and other ops)
  - No need to change existing programs



Comparison

	Thread	Fiber	Auto Fiber
Suspend/continue	**Yes**	**Yes**	**Yes**
Switch on timer	**Yes**	No	No
Switch on I/O b.	**Yes**	No	**Yes**
Synchronization	Required	**Not required**	**Required**
Specify next	No	**Yes**	**No**
Performance: Creation	Heavy	Lightweight	Lightweight
Performance: Switch	Lightweight	Lightweight	Lightweight

Advantage and Disadvantage

- Advantage
  - Don’t need to modify existing programs
  - Lightweight as a Fiber
  - Safer than Threads (no preemption)
- Disadvantage
  - Introduce “non-deterministic” dangers same as Thread programs
    - Non atomic operations can intercept accidentally.

Change the name…?

About this talk

- Behavior of Fiber
- History of Fiber
- Implementation of Fiber
- Auto Fiber proposal

Thank you for your attention

Koichi Sasada

<ko1@cookpad.com>

cookpad