Ruby Fibers

Farrel Lifson
farrel.lifson@aimred.com
http://www.aimred.com

Concurrent Programming in Ruby

- Processes
- Threads
- Fibers

Processes

- Seperate Ruby processes
- Managed by OS
- Robust
- Heavy – interpreter for every process
- External communication (sockets, files, db)

Aimred

Threads

- Managed by VM
- Limit (+- 3000)
- Usual Concurrent Programming Pitfalls
  - Deadlocks
  - WTHIH!

Fibers?

- Lightweight threads
- Low memory (4KB/thread)
- User scheduled

Simple Fiber

```
f = Fiber.new do
      yield 1
      yield 2
    end

f.resume
f.resume
```

Aimred

Parameters

```
f = Fiber.new do |number|
      Fiber.yield number + 5
      Fiber.yield number + 10
    end

f.resume 5
f.resume 5
f.resume 5 # FiberError!
```

Fibonacci

```
fib = Fiber.new do
        f1 = f2 = 1
        loop do
          Fiber.yield f1
          f1, f2 = f2, f1 + f2
        end
      end

10.times { puts fib.resume }
```

Aimred

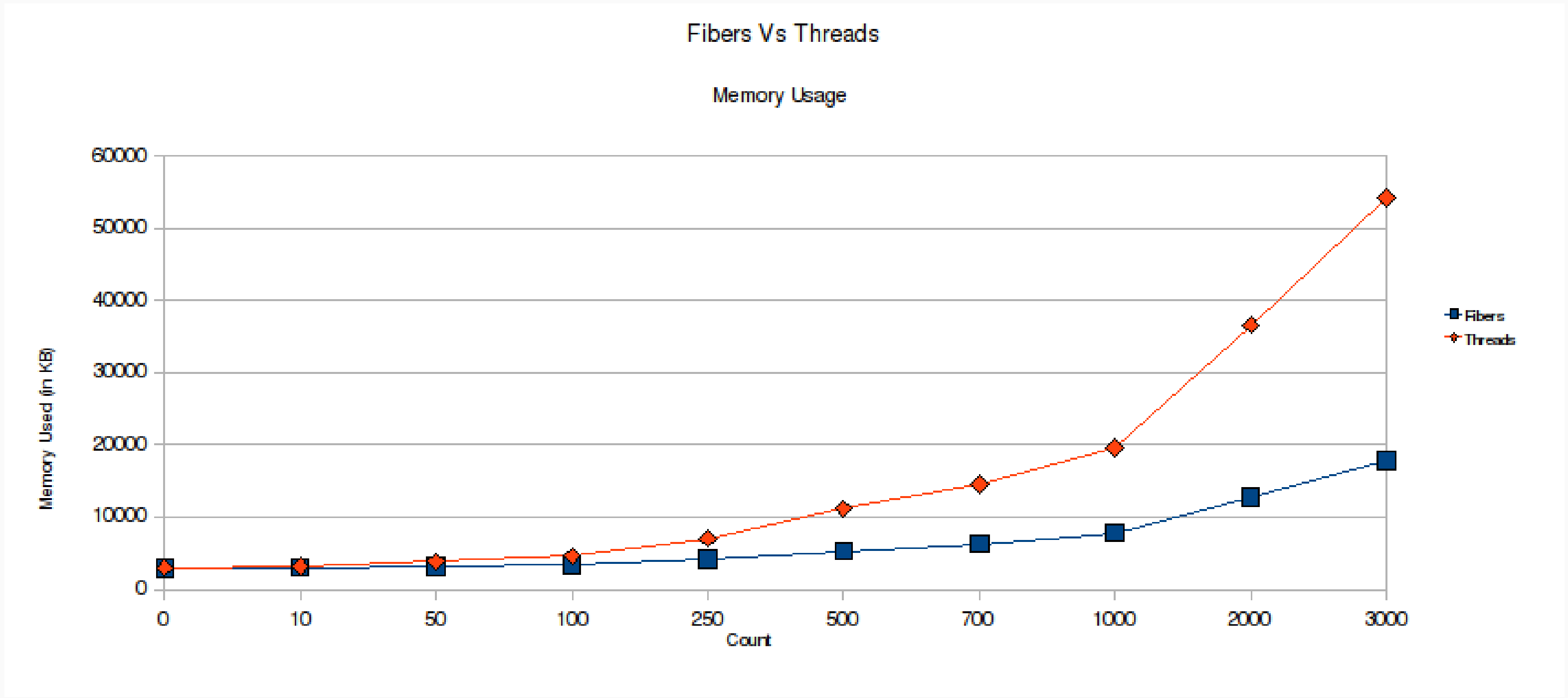
API

- Fiber.current
- Fiber.yield
- Fiber#alive?
- Fiber#resume
- Fiber#transfer

Aimred

So why use them?

- Fits in neatly with event driven programming
- Memory Usage



Aimred

Event Driven Programming

```
EventMachine.run {
  page = EventMachine::HttpRequest.new('http://google.ca/').get
  page.errback { p "Google is down! terminate?" }
  page.callback {
    about = EM::HR.new('http://.../searchq=eventmachine').get
    about.callback { # callback nesting, ad infinitum }
    about.errback  { # error-handling code }
  }
}
```

EDP With Fibers

```
def http_get(url)
  f = Fiber.current
  http = EventMachine::HttpRequest.new(url).get
  http.callback { f.resume(http) }
  http.errback  { f.resume(http) }
  return Fiber.yield
end

EventMachine.run do
  f = Fiber.new do
    page = http_get('http://www.google.com/')
    about = http_get('http://www.google.com/search?q=eventmachine') if page
  end
  f.resume
end
```

Aimred

Pitfalls

- Still have the Ruby GIL
- Written for 1.9
- Backported to 1.8 (green threads in disguise)