1 0 0 1 0 1 0 1 0 1 0 1 0 0

¿Qué tan óptima es tu Rails app?

1 0 1 0 0 1 0 1 0 1 0 0 1 0 1 0

Gabriel Coronado

- 7 años de experiencia.
- Dizque charlista.
- Mamador de gallo.

1001010100101010

1010010010110111

novo/scale koombea MUDFLAP

1
0
0
1
0
1
0
1
0
0
1
0
1
0
1
0

¡Ruby es lento!

1
0
1
0
0
1
0
0
1
0
1
1
0
1
1
1

1
0
0
1
0
1
0
1
0
0
1
0
1
0
1
0

¡Python también!

1
0
1
0
0
1
0
0
1
0
1
1
0
1
1
1

1
0
0
1
0
1
0
1
0
0
1
0
1
0
1
0

¡JS también!

1
0
1
0
0
1
0
0
1
0
1
1
0
1
1
1

1
0
0
1
0
1
0
1
0
0
1
0
1
0
1
0

¡Muchos otros lenguajes lo son!

1
0
1
0
0
1
0
0
1
0
1
1
0
1
1
1

Un poco de historia...

1001010100101010

- Tipado **estático**
- Manejo manual de la **memoria**
- Prioriza el **performance** y no la **productividad**

- Tipado **dinámico**
- Manejo **automático** de la memoria
- Prioriza la **productividad** y no el **performance** (el hardware avanzará)

1010010010110111

Un poco de historia...

3 DORITOS DESPUES

Un poco de historia...

1001010100101010

- Tipado **dinámico**
- Manejo **automático** de la memoria
- Prioriza la **productividad** y no el **performance** (el hardware avanzará)

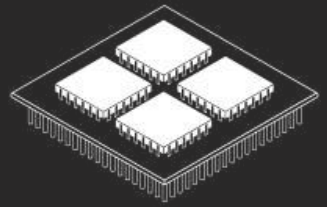
- Tipado **estático**
- Manejo manual de la **memoria**
- Prioriza el **performance** y no la **productividad**

1010010010110111

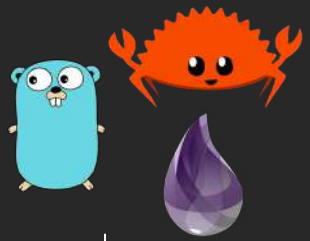
Un poco de historia...

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

Los 90’s



Multi-core CPUs (2000s)



2000 - Now

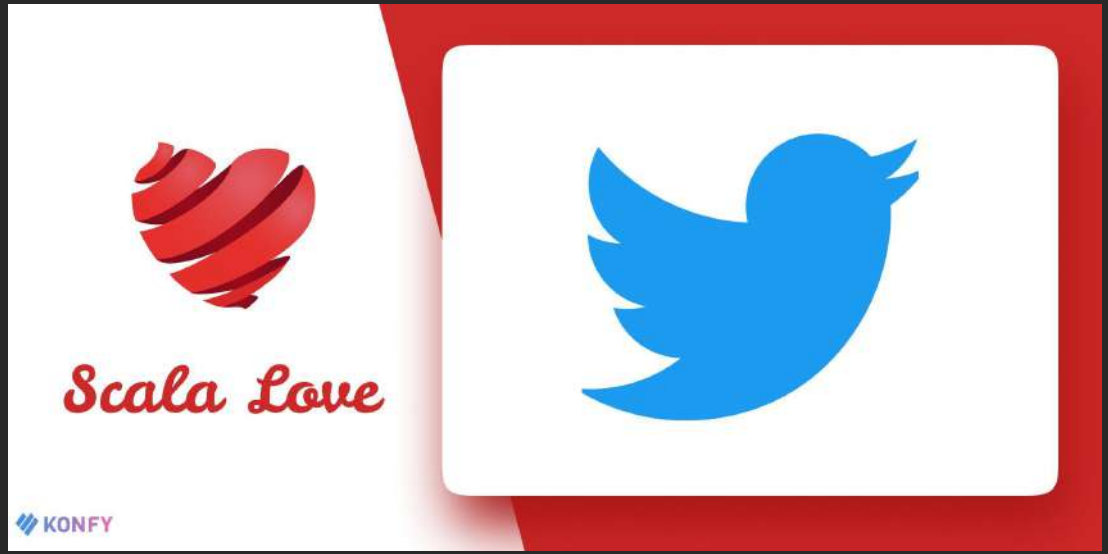
1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

Solución: ¿cambiar de lenguaje?

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

Solución: ¿cambiar de



1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

Ruby y los threads.

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

Ruby web applications usually have a limited number of web workers to serve requests, and each worker is only capable of processing one request at a time. A worker is backed by a Ruby thread or a system process. Due to the **Global Virtual Machine Lock** (**GVL**), adding more threads doesn't help us to increase the throughput. Usually, the number of threads is as low as three to five.

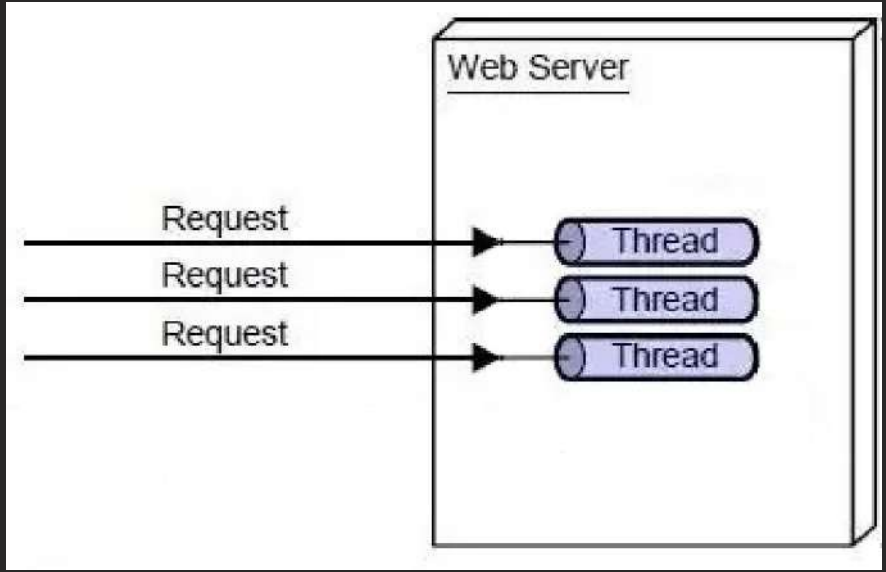
1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

Layered Design for Ruby on Rails Applications

VLADIMIR DEMENTYEV

Ruby y los threads.

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0



1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

¿Solución? Fibers.

1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

Fiber

Class

Fibers are primitives for implementing light weight cooperative concurrency in Ruby. Basically they are a means of creating code blocks that can be paused and resumed, much like threads. The main difference is that they are never preempted and that the scheduling must be done by the programmer and not the VM.

Upon yielding or termination the `Fiber` returns the value of the last executed expression

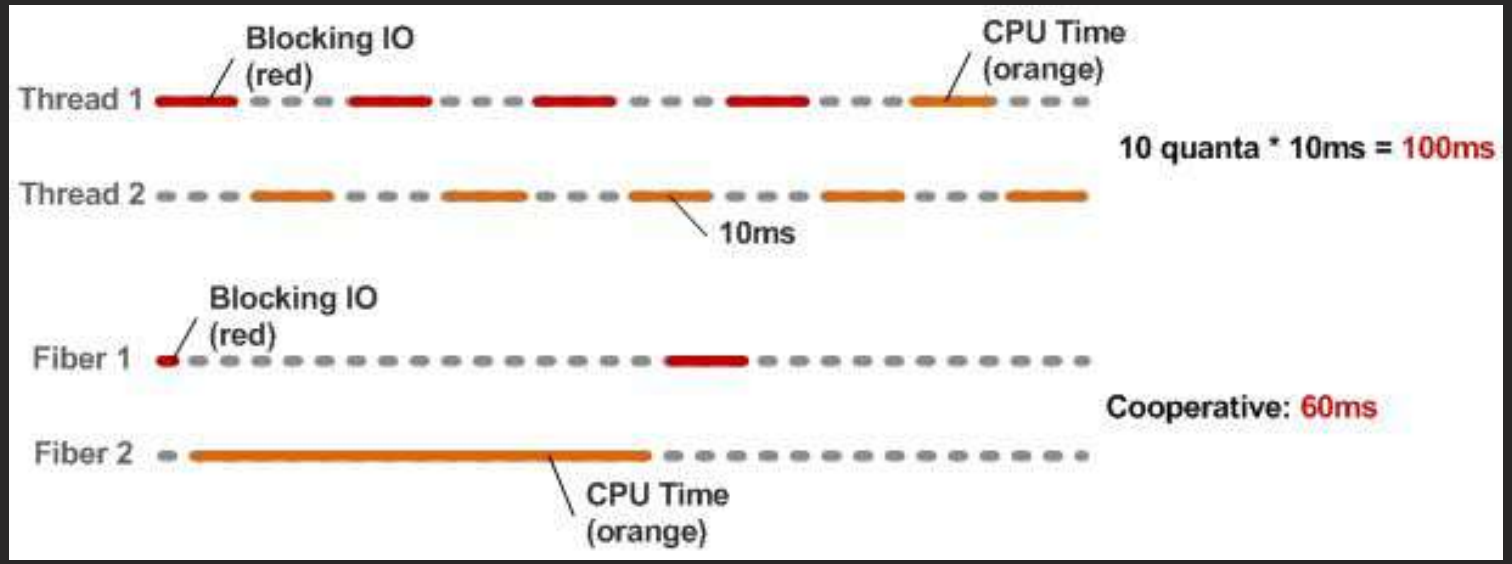
For instance:

Example

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

puts fiber.resume
puts fiber.resume
puts fiber.resume
```

https://rubyapi.org/3.2/o/fiber


Blocking IO
(red)
CPU Time
(orange)
Thread 1
10 quanta * 10ms = 100ms
Thread 2
10ms
Blocking IO
(red)
Fiber 1
Cooperative: 60ms
Fiber 2
CPU Time
(orange)

Ractors

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

“It’s multi-core age today. Concurrency is very important. With Ractor, along with Async Fiber, Ruby will be a real concurrent language.”

- Matz, 2020

1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

Ractors

- Parallel execution features.
- No thread-safety concerns.
- Actor-model based.

1
0
0
1
0
1
0
1
0
0
1
0
1
0
1
0

Pero... hay un pequeño problema.

1
0
1
0
0
1
0
0
1
0
1
1
0
1
1
1

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

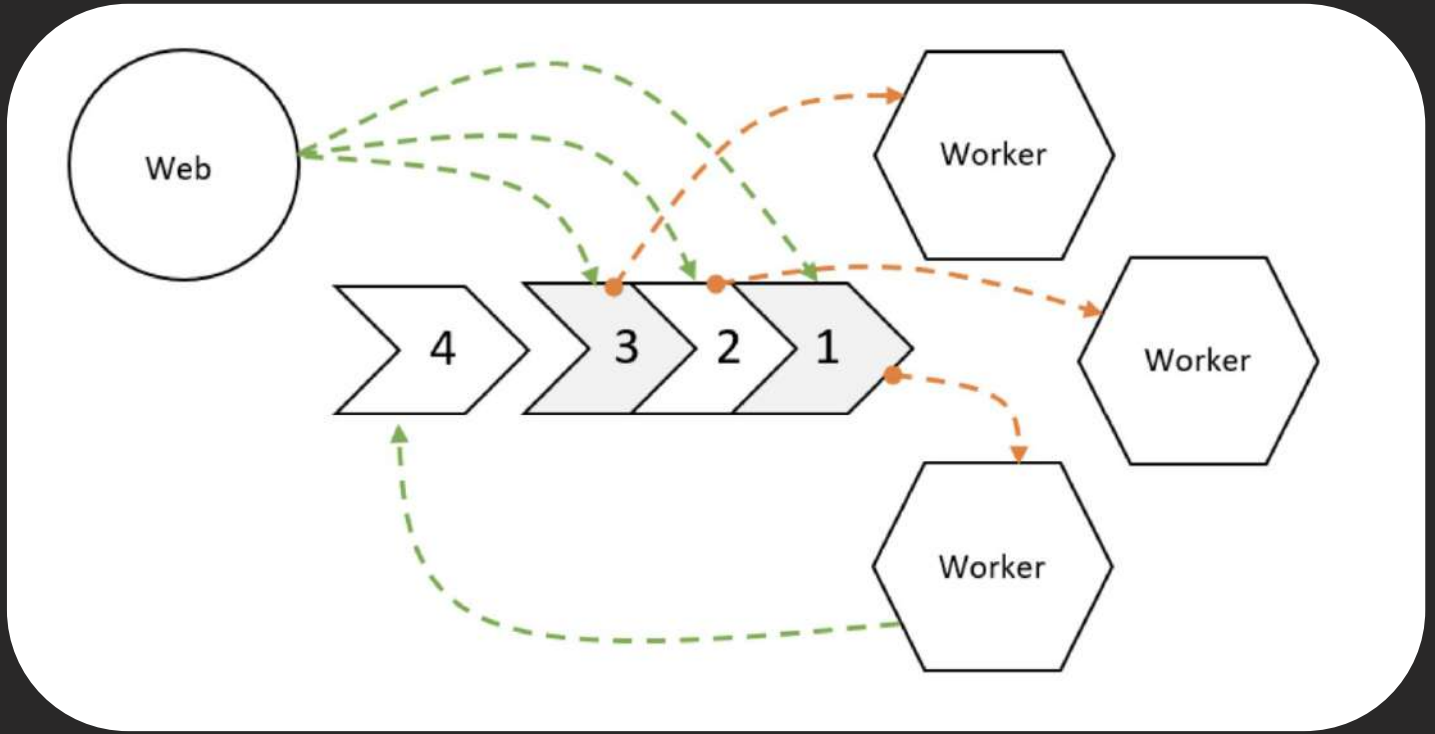
Rails aún no está listo para Fibers o Ractors.

1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

1
0
0
1
0
1
0
1
0
0
1
0
1
0
1
0

Sidekiq **al rescate**

1
0
1
0
0
1
0
0
1
0
1
1
0
1
1
1


Web
Worker
Worker
Worker
4
3
2
1

Sidekiq NO es magia. No lo maluses.

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

For really heavy processes, look at Batch, Iterable, etc...

No todo es código! Por ejemplo: meses o incluso años de data? Why not a limit?

1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

Aprende Sidekiq!

- Latency queues.
- Idempotency.
- Best practices.
- Etc…

Exceptions as Flow Control

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

“Don't use exceptions as control flow.”

1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

Exceptions as Flow Control

```
def user_loggedin?
  User.find(session[:user_id])
rescue ActiveRecord::RecordNotFound
  false
end
```

Exceptions as Flow Control

```
def user_loggedin?
  User.find_by(id: session[:user_id])
end
```

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

¿Qué tienen que ver las excepciones con el performance?

1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

1 0 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0

¡Las excepciones en Ruby son lentas!

1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

```
class TestBench
  def fast
    customer = Customer.new

    if Charge.create(amount: 400)
      customer.status = :active
    else
      customer.status = :delinquent
    end
  end

  def slow
    customer = Customer.new
    Charge.create!(amount: 400)
    customer.status = :active
  rescue Charge::Declined
    customer.status = :delinquent
  end
end
```

```
irb(main):036:0> test = TestBench.new
=> #<TestBench:0x0000000100be4d20>
irb(main):037:0>
irb(main):038:1* Benchmark.ips do |x|
irb(main):039:1*   x.report("if/else") { test.fast }
irb(main):040:1*   x.report("exceptions") { test.slow }
irb(main):041:1*   x.compare!
irb(main):042:0> end
ruby 3.2.5 (2024-07-26 revision 31d0f1a2e7) [arm64-darwin23]
Warming up --------------------------------------
             if/else   386.607k i/100ms
          exceptions   115.633k i/100ms
Calculating -------------------------------------
             if/else      4.105M (± 4.1%) i/s -     20.490M in   5.002335s
          exceptions      1.115M (± 6.9%) i/s -      5.550M in   5.006719s

Comparison:
             if/else:  4104736.2 i/s
          exceptions:  1114649.9 i/s - 3.68x  slower
```

1
0
0
1
0
1
0
1
0
0
1
0
1
0
1
0

Ruby y la memoria

1
0
1
0
0
1
0
0
1
0
1
1
0
1
1
1

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

El garbage collector de Ruby tampoco es magia, y tampoco es el más rápido. Así que ojo!

1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

Memory Leak vs Memory Bloat

- Memory Leak: Memoria que nunca se libera, causando pérdida permanente de recursos.
- Memory Bloat: Uso excesivo de memoria que podría optimizarse, causando ineficiencia.

Memory Leak vs Memory Bloat

- Be aware of memory usage, queries, objects, variables, etc.
- Use oink or ps to look for large allocations in your app.
- Audit your gemfile using derailed_benchmarks, looking for anything that require more than ~10MB of memory
- Look to replace these bloated gems with lighter alternatives.

1
0
0
1
0
1
0
1
0
0
1
0
1
0
1
0

jemalloc al rescate

1
0
1
0
0
1
0
0
1
0
1
1
0
1
1
1

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

jemalloc es un “free win”

1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

As you can see, each pod pre-jemalloc was gobbling up an average of **1.6GB**. Post-jemalloc, this consumption plummeted to a mere **550 MB**.

Por favor, optimiza tus queries.

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

- N+1
- Heavy queries.
- Use the right Indexes.
- Use EXPLAIN, ANALYZE

1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

Tus mejores amigas, las tools

- Oink
- Detailed_benchmark
- AppSignal
- NewRelic

1001010100100101010

“The first step to a scalable web service is automatic autoscaling. Without autoscaling, you're just waiting for a flash sale, social media post, or other "white swan" event to take down your site at just the wrong moment. ”

Nate Berkopec / The Complete Guide to Rails Performance

1010010010110111

1 0 0 1 0 1 0 1 0 0 1 0 1 0 1 0

¡Muchas gracias!

1 0 1 0 0 1 0 0 1 0 1 1 0 1 1 1

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**