Introducción

1 Lo que aprenderás sobre Ruby

2 ¿Qué es Ruby? Ventajas y desventajas

3 Instalación de Ruby

Bases del lenguaje

4 Tipos de datos I: Integer, float

5 Tipos de datos II: Strings

6 Tipos de datos III: Symbols

7 Tipos de datos IV: Arrays

8 Tipos de datos V: Hashes

9 Uso de condicionales

10 Uso de ciclos

11 Rangos

Concurrencia VS Paralelismo

Los términos concurrencia y paralelismo pueden ser fácilmente confundidos. Por un lado 2 tareas se ejecutan en paralelo cuando ambas se ejecutan en unidades de procesamiento independientes al mismo tiempo, es decir, ambas tareas pueden comenzar exactamente al mismo tiempo pues su ejecución es manejada por dos unidades de procesamiento diferente. Por otro lado, dos tareas se ejecutan concurrentemente cuando se pueden ejecutar en la misma unidad de procesamiento intercalando subtareas de ambas tareas.

Por ejemplo:

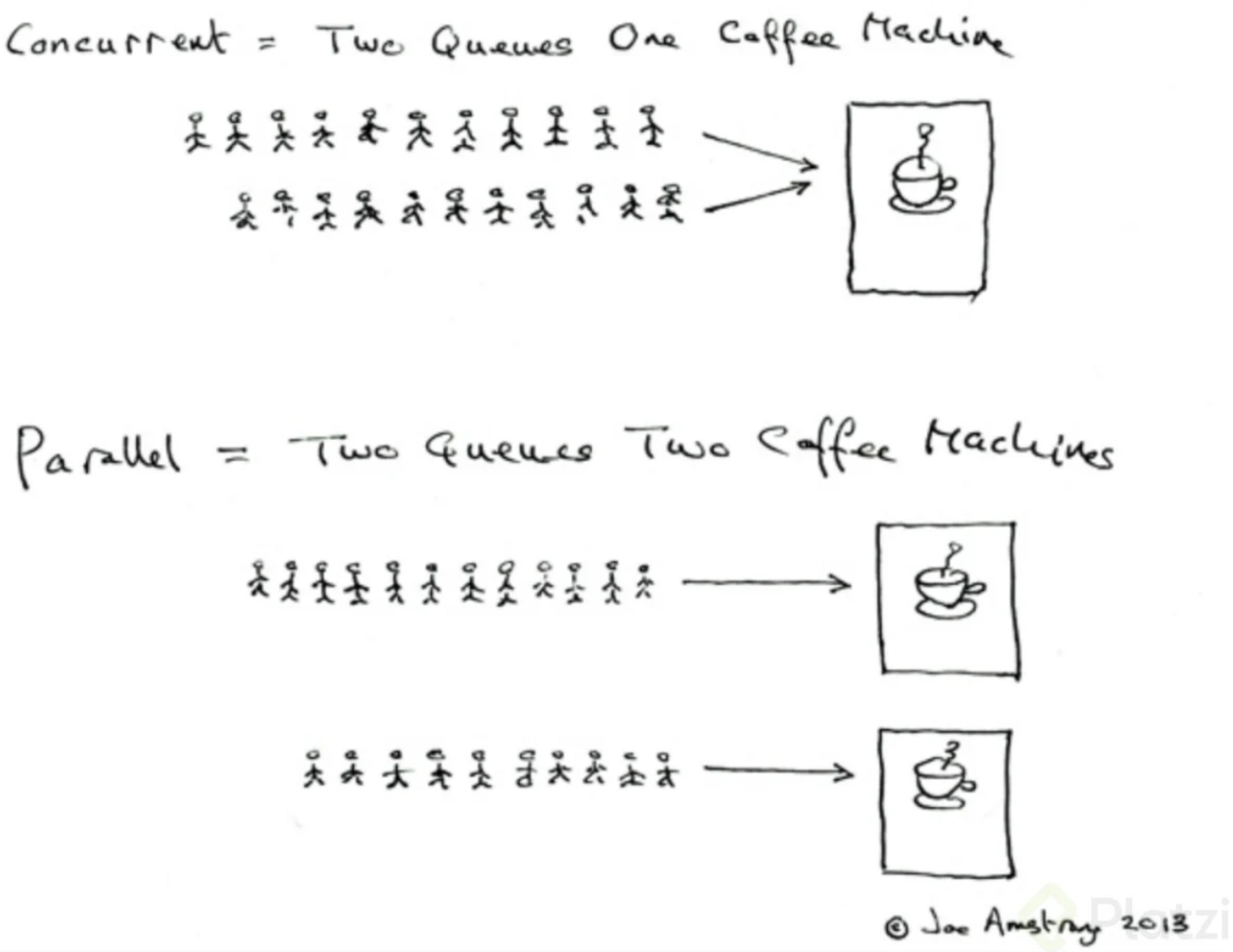
Hay una empresa que está desarrollando un blog como una aplicación web. Para esto necesita desarrollar el backend y el frontend para lo cual contrata a un desarrollador fullstack llamado Pepe que se puede encargar de ambas tareas. Pepe entonces tiene 2 tareas, hacer el backend y hacer el frontend, y a su vez estas 2 tareas se pueden dividir en pequeñas subtareas así:

- Backend
  - API para la autenticación
  - CRUD de posts
- Frontend
  - Vista de autenticación
  - Vista de los posts

Pepe puede trabajar en ambas tareas concurrentemente pues puede primero hacer una versión inicial de la implementación del API para la autenticación. Mientras su líder técnico revisa el código de esta versión inicial, Pepe puede comenzar a implementar la vista de autenticación y una vez su líder técnico termine la revisión de la versión inicial del API de

autenticación, puede continuar con esta tarea y repetir el mismo proceso hasta terminar con todas las subtareas y tareas.

Si la empresa contrata a un desarrollador frontend y a un desarrollador backend, ambos desarrolladores pueden trabajar en ambas tareas en paralelo. Pues ambos pueden comenzar con ambas tareas al tiempo. En este ejemplo los desarrolladores son las unidades de procesamiento y el desarrollo del backend y frontend son las tareas que se pueden ejecutar en diferentes threads o unidades de procesamiento.
Ahora un ejemplo gráfico:



Tomado de: https://joearms.github.io/published/2013-04-05-concurrent-and-parallel-programming.html

Limitaciones de concurrencia en Ruby

En Ruby tenemos la posibilidad de crear Threads, sin embargo, su comportamiento depende del intérprete que utilicemos. El interprete que usamos en el curso que adicionalmente es el interprete más popular (cruby o MRI) no permite paralelismo asi el computador en el que ejecutemos nuestro programa tenga múltiples cores en su procesador. MRI utiliza un mecanismo llamado Global Interpreter Lock (GIL) que hace que el interprete solo pueda ejecutar un Thread a la vez. Esto es una decisión que tomaron quienes diseñaron el lenguaje pues es una manera relativamente sencilla de evitar race conditions, deadlocks y otros problemas comunes que surgen cuando se está haciendo programación concurrente o en paralelo.

Aunque el GIL no permite que multiples threads se ejecuten, sí permite cambiar de contexto cuando se esta realizando una operación por fuera del interprete como operaciones de lectura o escritura. Como estas operaciones suceden por fuera del interprete, cruby permite cambiar de contexto para ejecutar otro thread mientras estas operaciones terminan y de esta manera se puede hacer programacion concurrente.

Otros interpretes como JRuby y Rubinius no tienen un GIL así que permiten ejecución en paralelo.

Threads en Ruby

```ruby
# threads.rb

def without_threads
 puts "Without threads"
 start = Time.now
 3.times { |i| http_call(i) }
 puts "Total time: #{Time.now - start} seconds"
 puts "---------------------------------------"
end

def with_threads
 puts "With threads"
 start = Time.now
 threads = 3.times.map { |i| Thread.new { http_call(i) } }
 threads.map(&:join)
 puts "Total time: #{Time.now - start} seconds"
 puts "---------------------------------------"
end

# Ejecución sin threads


def http_call(n)
 puts "Simulating HTTP call #{n}"
 sleep(1)
end
```

```
without_threads
with_threads
```

```
$ ruby threads.rb
Without threads
Simulating HTTP call 0
Simulating HTTP call 1
Simulating HTTP call 2
Total time: 3.009568 seconds
---------------------------------------
With threads
Simulating HTTP call 2
Simulating HTTP call 1
Simulating HTTP call 0
Total time: 1.003631 seconds
---------------------------------------
```

Platzi

En este ejemplo estamos simulando hacer 3 llamados HTTP. Cada llamado toma 3 segundos y estamos ejecutando estos llamados con y sin threads. Aunque el GIL no permite la ejecución en paralelo de múltiples threads, como la operación HTTP debe esperar un segundo por la respuesta (este comportamiento es simulado usando “sleep(1)”), el interprete puede cambiar de contexto y ejecutar los demás threads que necesiten ser ejecutados. De esta manera podemos ver que la ejecución sin threads toma 3 segundos pues ejecuta los 3 llamados en serie, sin embargo la ejecución que utiliza threads solo toma 1 segundo pues tan pronto un thread llama al método sleep, el interprete detecta que debe esperar así que puede cambiar de contexto y permitir la ejecución de los otros 2 threads.

A tener en cuenta

- Para inicializar un Thread se utiliza debe crear un objeto Thread con Thread.new y pasarle un bloque en donde definimos lo que se debe ejecutar.

- El punto de entrada de un programa Ruby se ejecuta en un thread principal o “main thread”. Tan pronto este thread termina, la ejecución de todo el programa es terminado, así que si creamos varios threads pero el “main thread” finaliza primero los otros threads van a ser terminados. Para evitar esto debemos hacer “join” de los threads adicionales. Join es un mecanismo que hace que el thread principal espere a la finalizacion del thread al que se le hace join lo que podemos ver en la linea " threads.map(&:join)".

Referencias:

---

https://github.com/simon0191/platzi-curso-ruby/commit/8497e9ace3011767355fb814a473c39f896dfe67
https://www.toptal.com/ruby/ruby-concurrency-and-parallelism-a-practical-primer
https://joearms.github.io/published/2013-04-05-concurrent-and-parallel-programming.html
https://medium.com/@franzejr/ruby-3-mri-and-gil-a302577c6634
https://robots.thoughtbot.com/untangling-ruby-threads