Ruby Concurrency

Me

Mike Barry
Software Developer
Refworks-COS, a division of ProQuest
mike.barry@refworks-cos.com

Concurrency

Rob Pike best described it at golang 2012:

- Concurrency is about dealing with lots of things at once.
- Parallelism is about doing lots of things at once.
- Concurrency is about structure, parallelism is about execution.
- Concurrency provides a way to structure a solution to solve a problem that may (but not necessarily) be parallelizable.

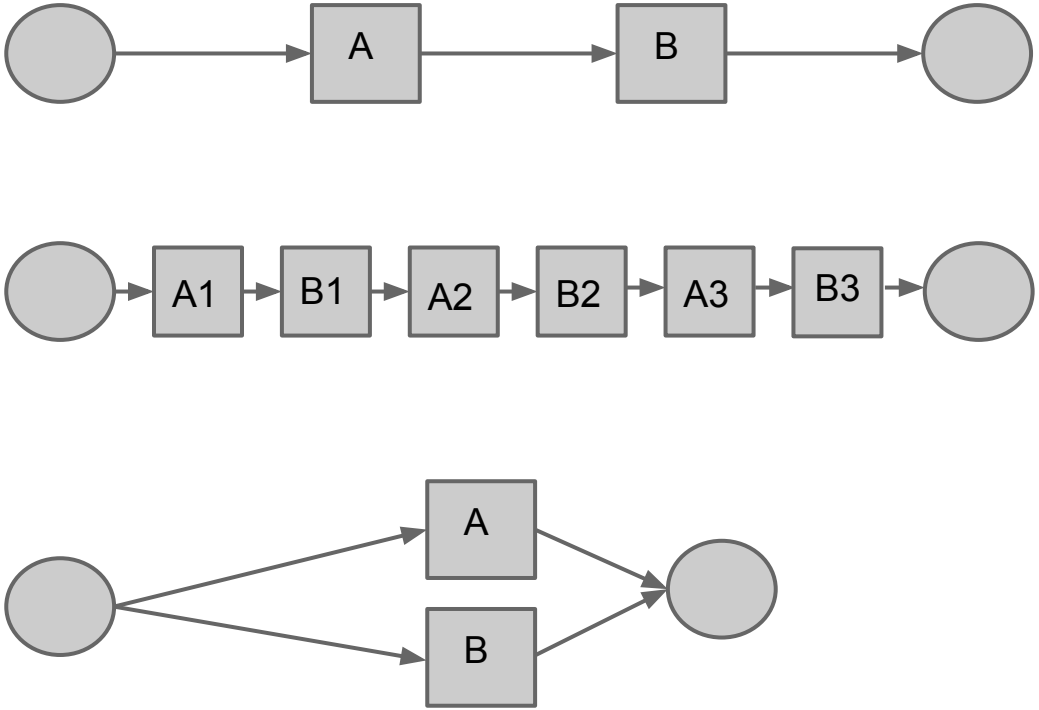
Concurrency

You have two tasks (A and B) that take a full day to complete. You could:

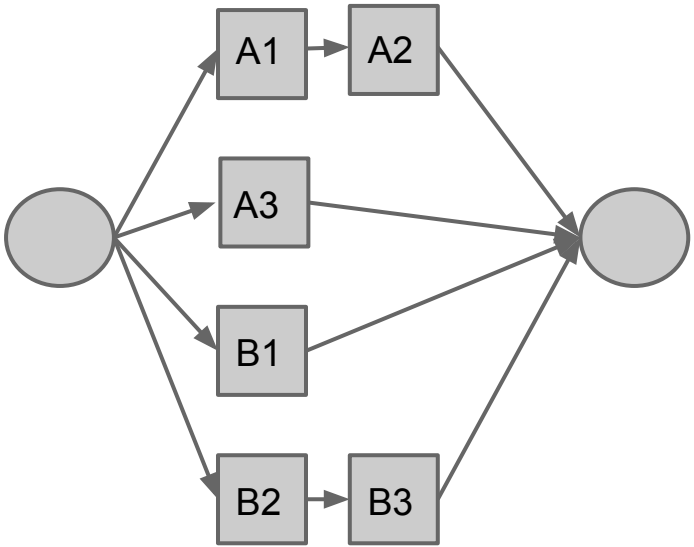
1. Work on A then on B
2. Work a bit on A, a bit on B, a bit on A …
3. You can hire another programmer to work on B while you work on A

-- stolen from Jesse Storimer's "Working with Ruby Threads

Concurrency



Concurrent or Parallel?



Real World Problem

thousands …. possibly millions of URLs in a database and we need to check if they're still valid or not

(uggh)

Caution!

(yeah … ruby, not rails)

```
con = Mysql.new( 'localhost', '', '', 'test' )
rs  = con.query( 'select * from urls' )
cnt = 0
rs.each_hash do |h|
  uri = URI.parse( h['url'] )
  response = Net::HTTP.get_response( uri )
  puts "#{response.code} -- #{h['url']}"
end
con.close
```

Real World Problem

- sequential
- could run fine one day
- could run horrible the next
- since it’s sequential, you have to wait

Real World Problem, Old School Solution

```
max_proc = 10
pm  = Parallel::ForkManager.new( max_proc )
con = Mysql.new( 'localhost', '', '', 'test' )
rs  = con.query( 'select * from urls' )
rs.each_hash do |h|
 pm.start(h) and next
   uri = URI.parse( h['url'] )
   response = Net::HTTP.get_response( uri )
   puts "#{response.code} -- #{h['url']}"
   pm.finish(0)
end
con.close
pm.wait_all_children
```

Processes

- Pros
  - well understood
  - fast for certain applications
  - copy-on-write means process are not as heavy as they used to be (for certain applications)
- Cons
  - access to shared resources can be painful
  - communication between processes can be painful
  - copy-on-write penalty still present

Real World Problem,
Kinda old but not really that old solution

```
con = Mysql.new( 'localhost', '', '', 'test' )
rs  = con.query( 'select * from urls' )
ths = []
rs.each_hash do |h|
 ths << Thread.new do
   uri = URI.parse( h['url'] )
   response = Net::HTTP.get_response( uri )
   puts "#{response.code} -- #{h['url']}"
 end
end

ths.each(&:join)
con.close
```

Threads

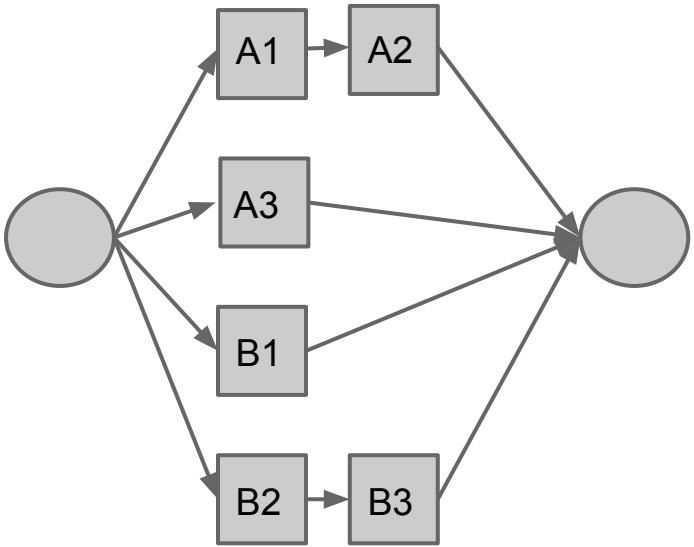
- Pros
  - easier access to shared resources
  - no copy-on-write penalty
- Cons
  - easier access to shared resources
  - no matter how good your concurrency is, with MRI, you can never be parallel

MRI Threads and concurrency

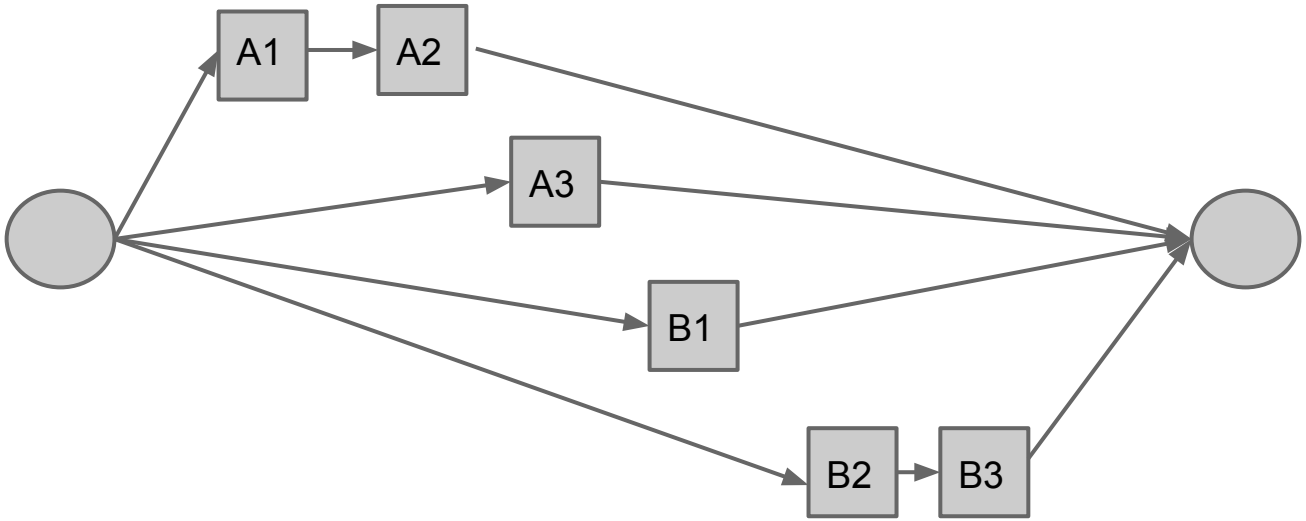
MRI 1.9.x uses native threads so threads can run on multiple cores

However since the interpreter is single threaded - access to it is protected by a mutex - the global interpreter lock - or GIL, you can never achieve parallelism.

MRI Threads and concurrency



MRI Threads and concurrency



MRI Threads and concurrency

Except … for blocking I/O which makes our example primed for parallelism.

So if you want 100% parallelism

JRuby or Rubinius

Jruby and Rubinius are alternate ruby interpreters.

JRuby is a JVM based interpreter.

Rubinius is a C++ based interpreter.

Beyond Threads - Actor Model

- an actor is a primitive that runs in it’s own thread
- in response to a message an actor recieves, it can
  - send messages to other actors
  - create additional actors
  - designate a new behavior to process subsequent messages

Celluloid

- Celluloid is a framework that implements the Actor Model in Ruby
- works in MRI 1.9.3, JRuby 1.6+ and Rubinius 2.0
- JRuby or Rubinius are preferred since there’s no GIL

Real World Problem -- Celluloid

```
class Fetcher
  include Celluloid

  def fetch( url )
    uri = URI.parse( url )
    response = Net::HTTP.get_response( uri )
  end
end

pool = Fetcher.pool( size: 10 )
fetchers = URLS.each do |url|
  pool.future(:fetch, url)
end
```

Wrap It Up

- MRI Ruby comes with a set of primitives that allow concurrent programming.
- MRI's GIL prevents 100% parallelism
- There are alternate implementations that either do away with the GIL or minimize it
- There are higher-level frameworks that build atop the primitives that make your life easy.

References

Rob Pike, “Concurrency is not Parellilism” -- http://talks.golang.org/2012/waza.slide#1

Jesse Storimer -- “Working with Ruby Threads” -- http://www.jstorimer.com/products/working-with-ruby-threads

Celluloid -- https://github.com/celluloid/celluloid

Questions?