
AU
RUBY
CONF
-2023-

PG::NumericValueOutOfRange

We ran out of numbers!?

$2^{31} - 1$

```
ALTER TABLE things
CHANGE COLUMN id TYPE BIGINT;
```

15,000 rows / second / core

2 billion records

2b / 15k = ~40 hours

100%	98%	100%	96%	97%	100%	98%	98%
72%	100%	96%	96%	95%	98%	99%	82%
93%	100%	94%	100%	99%	92%	94%	96%
91%	99%	100%	79%	97%	98%	96%	100%

100%	0%	0%	0%	0%	0%	0%	0%
0%	0%	0%	0%	0%	0%	0%	0%
0%	0%	0%	0%	0%	0%	0%	0%
0%	0%	0%	0%	0%	0%	0%	0%

What properties do we care about?

Efficient

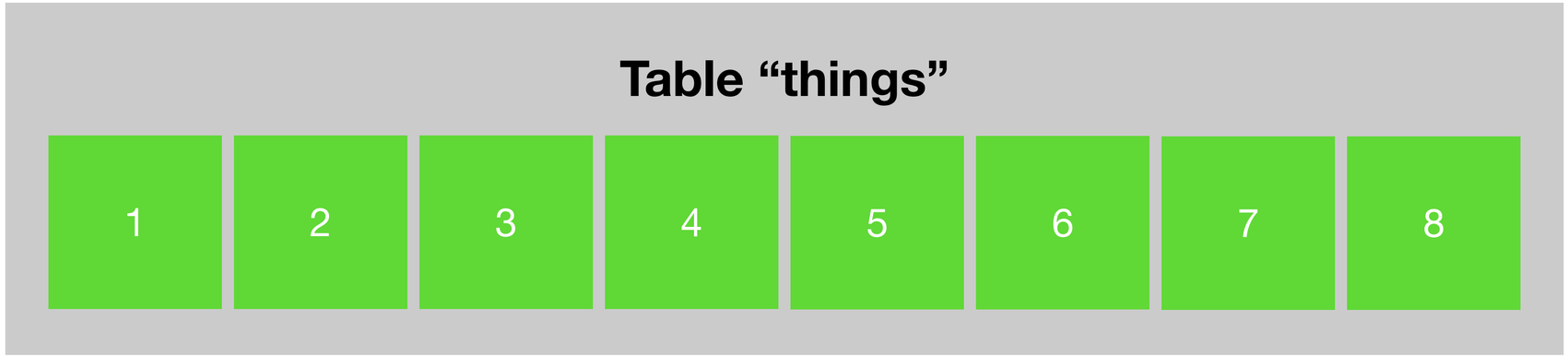
Responsive

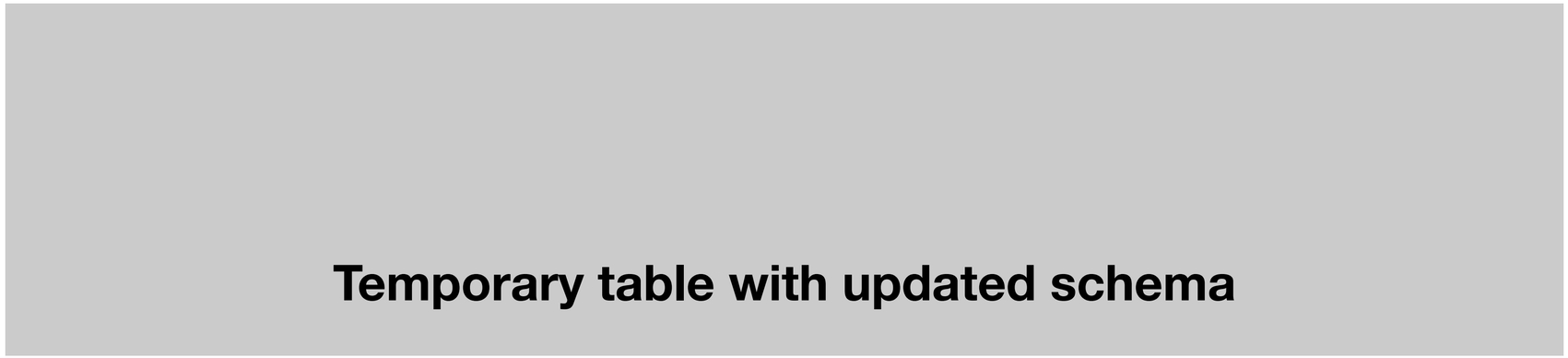
Robust

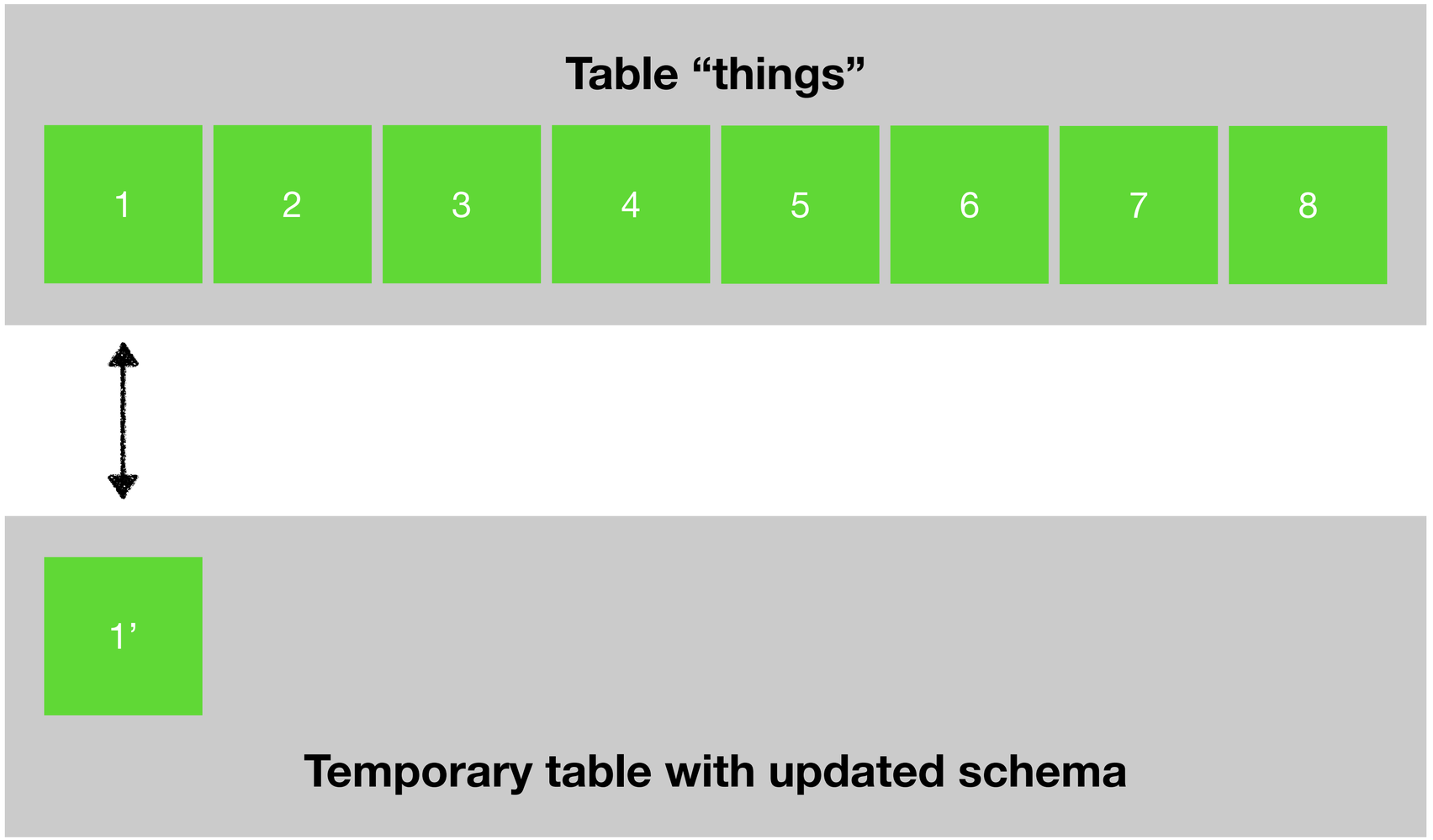
Asynchronous Rails

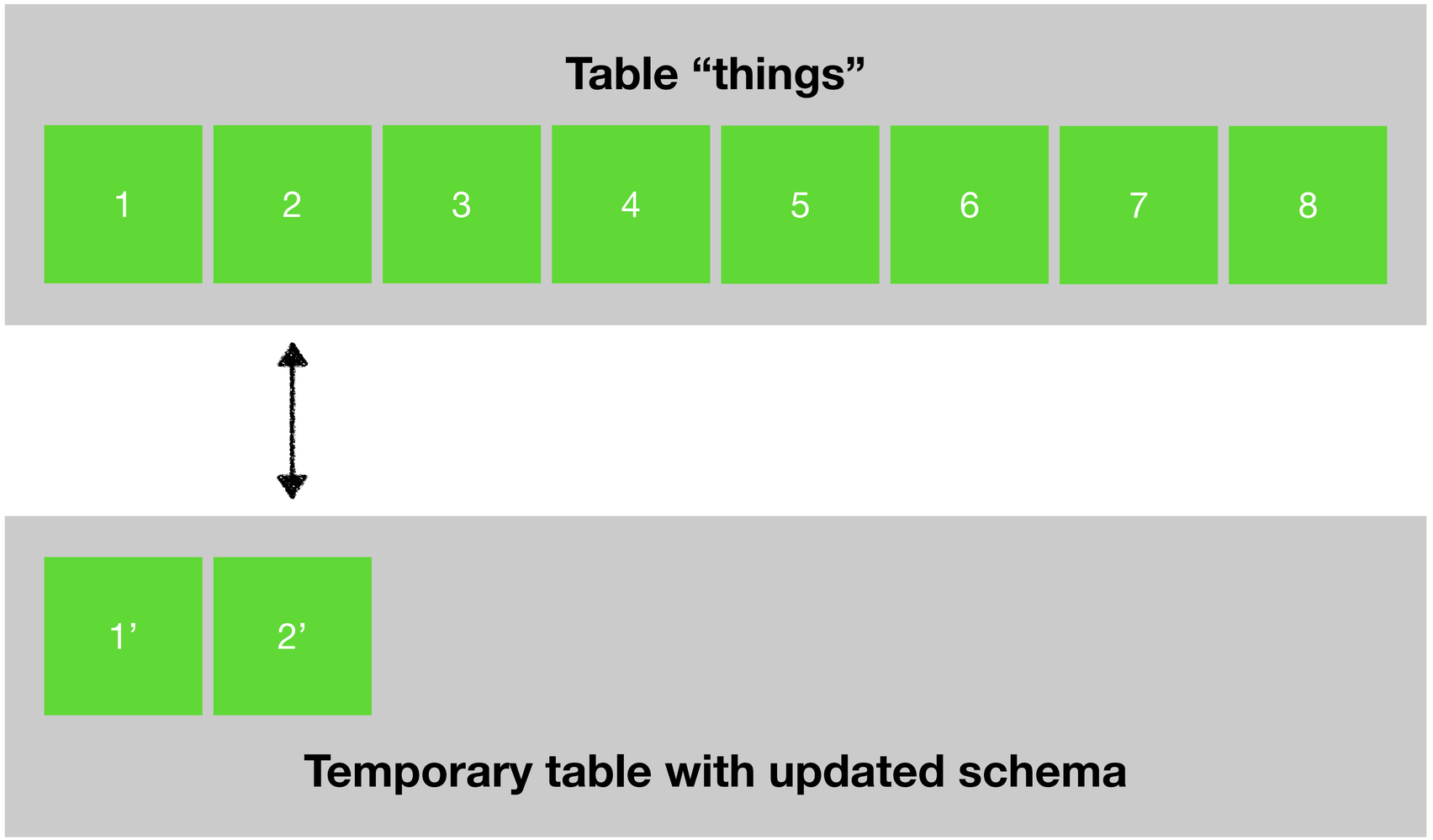
Samuel Williams
ruby.social/@ioquatix

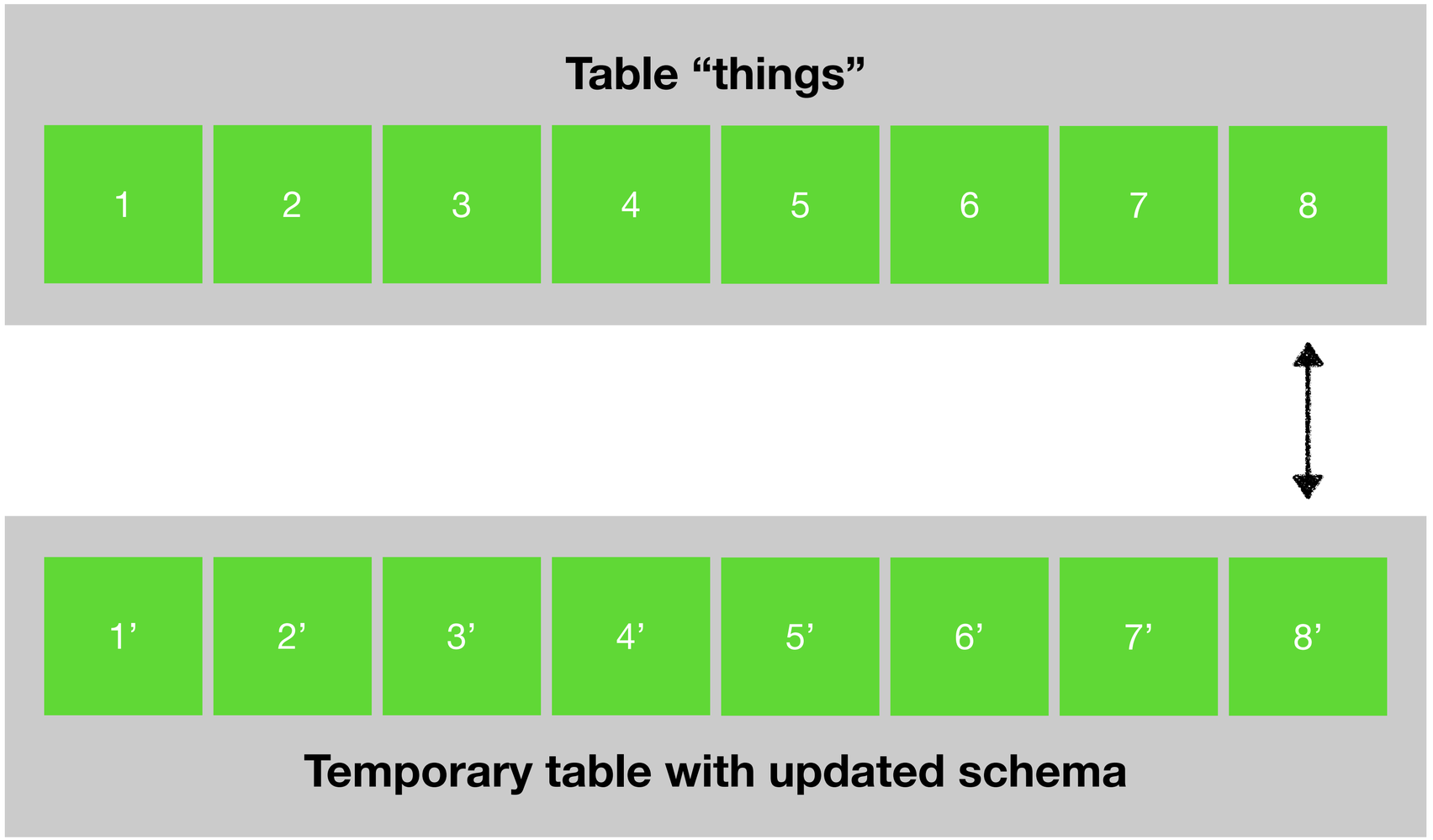
What is “Synchronous”?





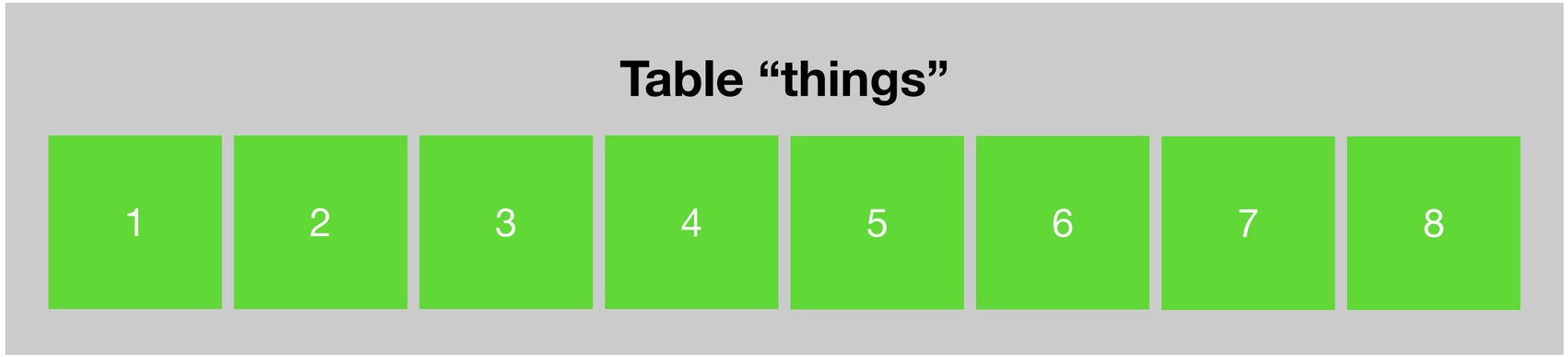


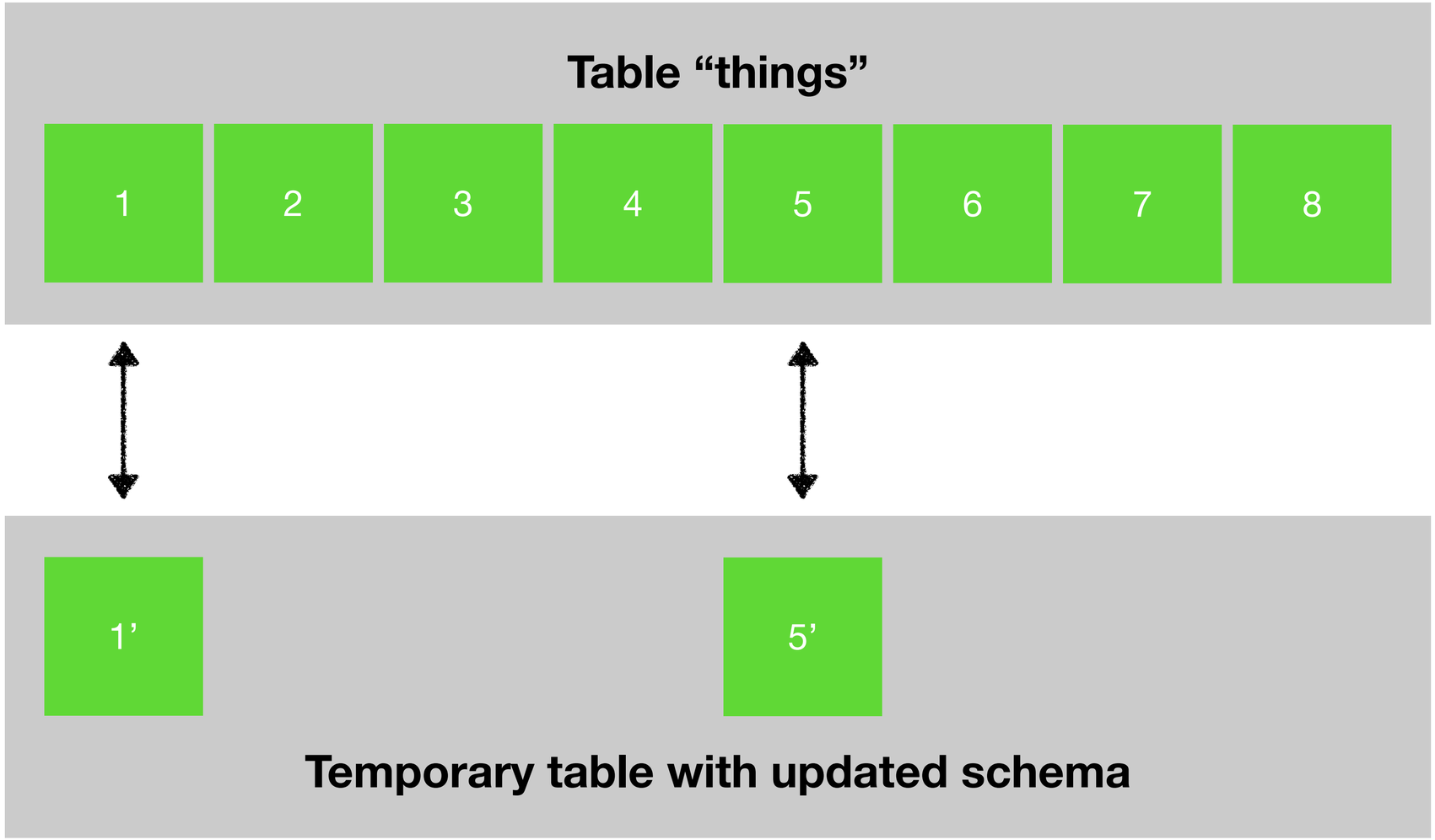


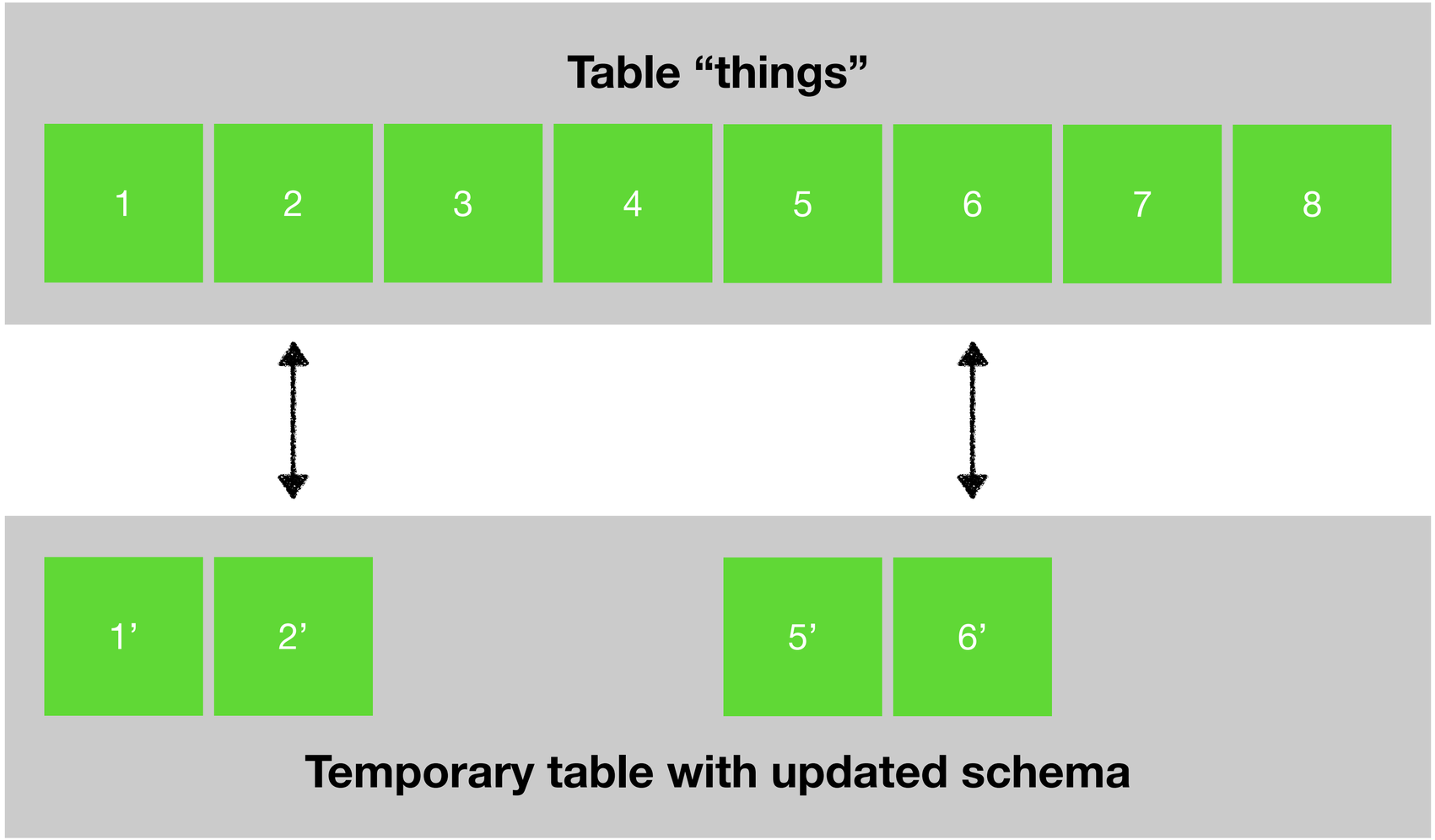


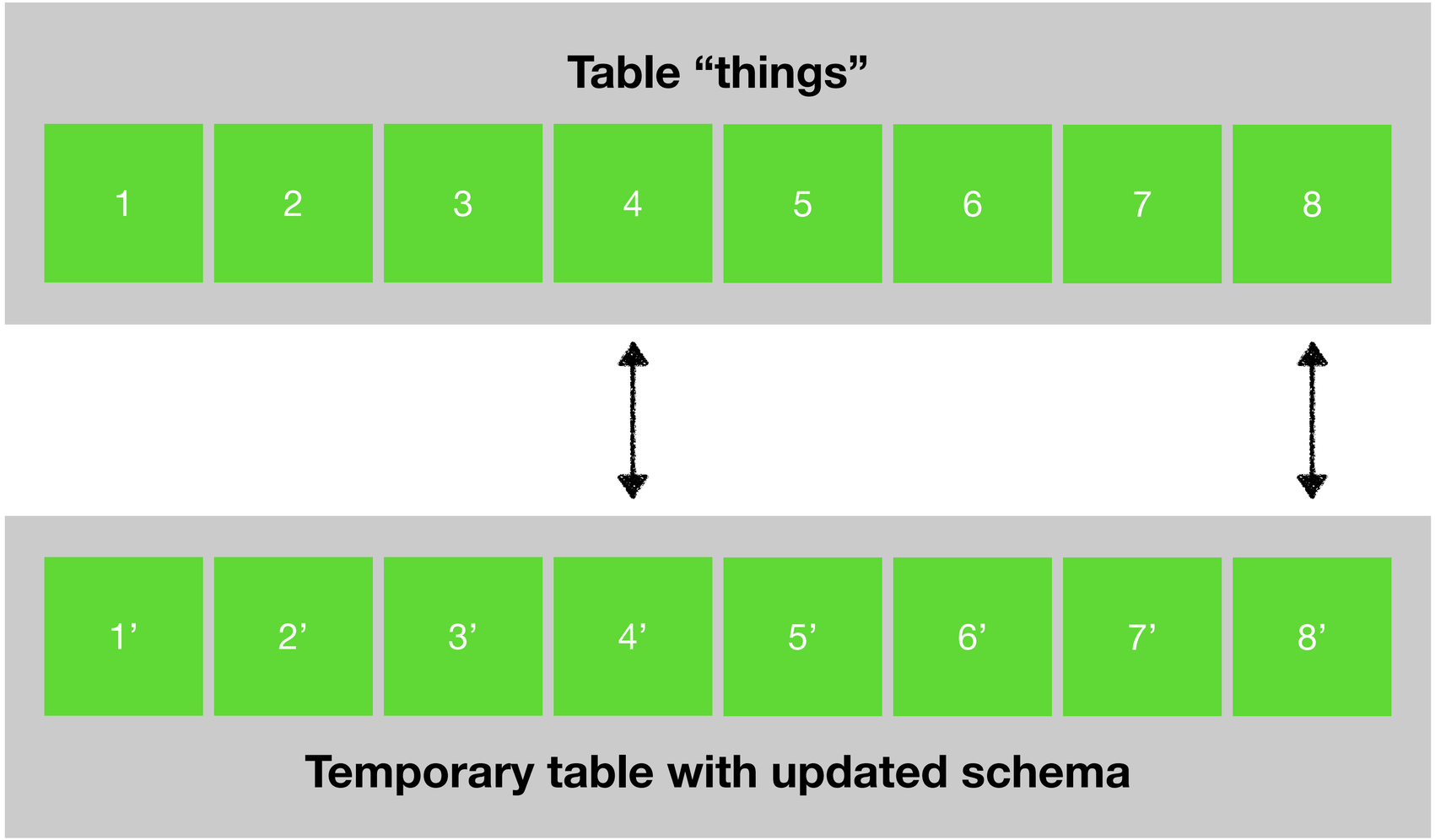
AU RUBY CONF -2023-

What is “Asynchronous”?

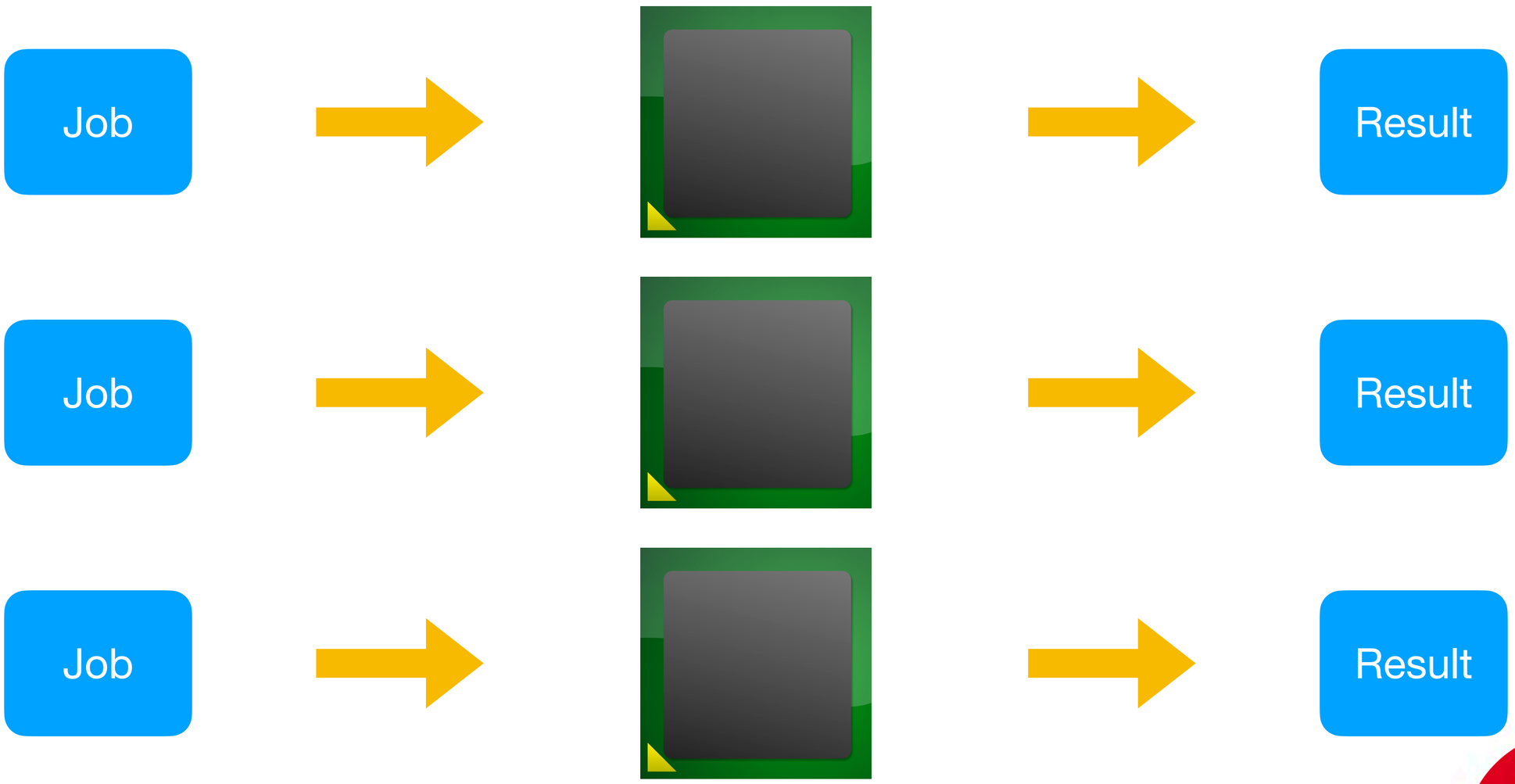


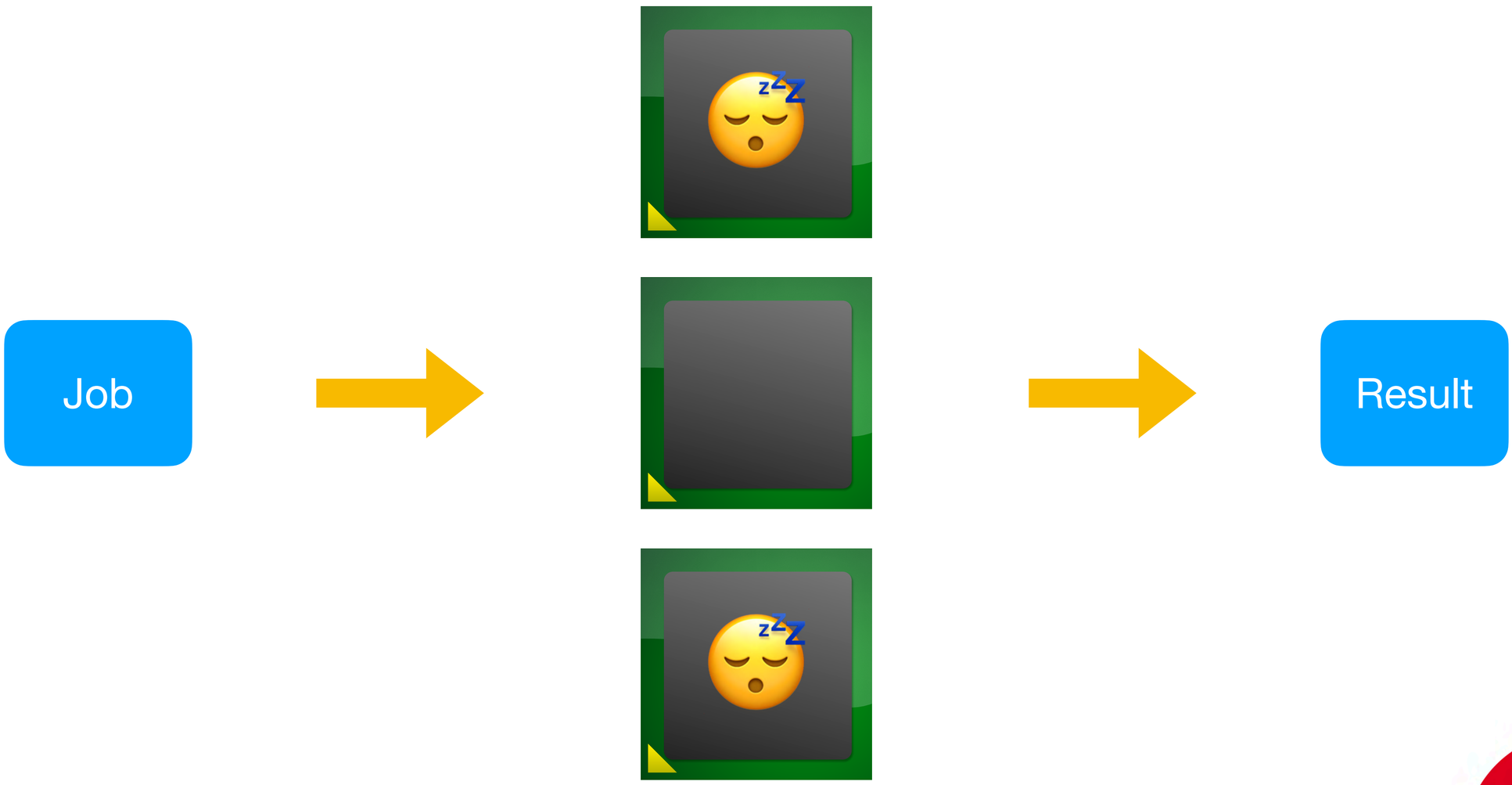






What is “Parallel Execution”?




Job
Result

What is “Concurrent Execution”?

Job

Job

Job

Job

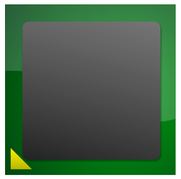
Job

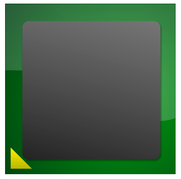
Job

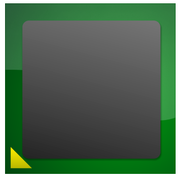
Result

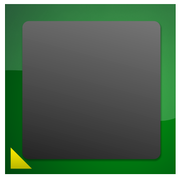
Result

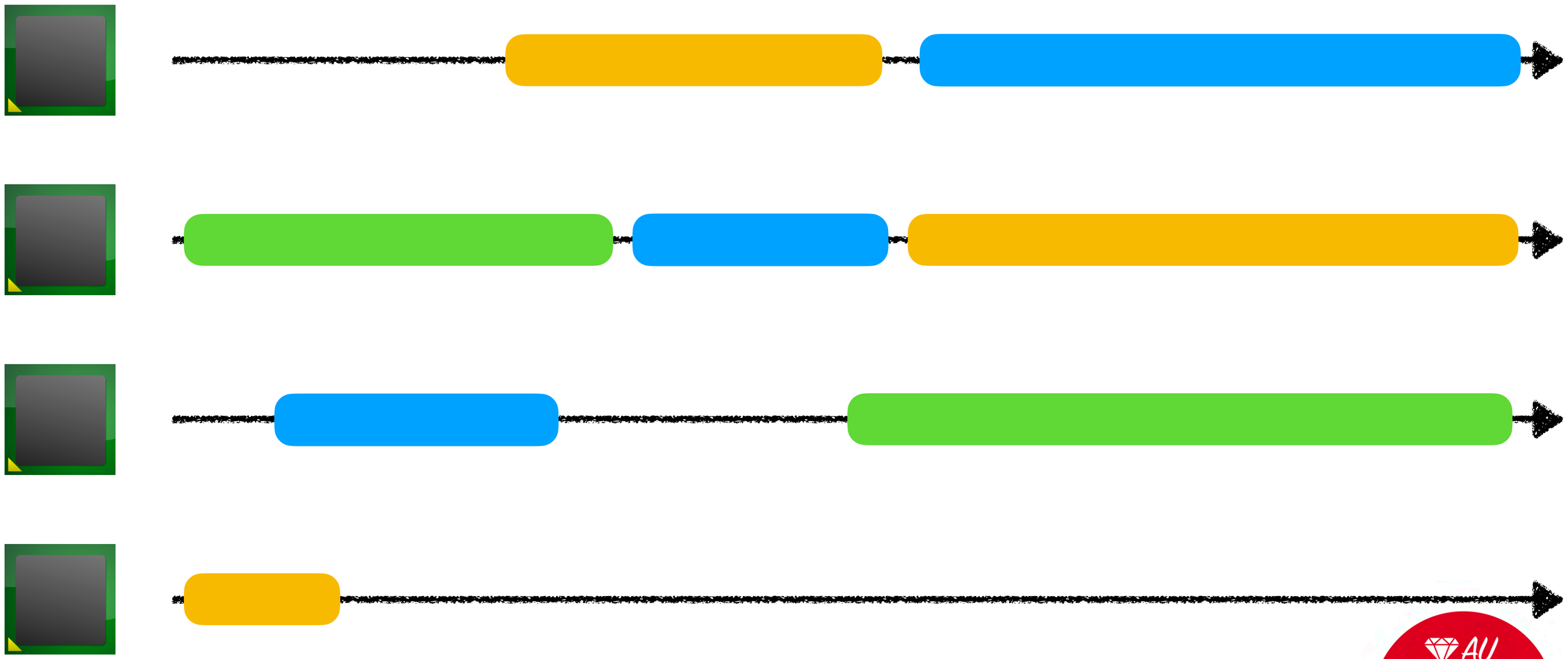
Result











What are the advantages of asynchronous execution?

How does it affect application performance?

Better scalability.

Lower latency.

How does it affect application design?

User Interactivity.

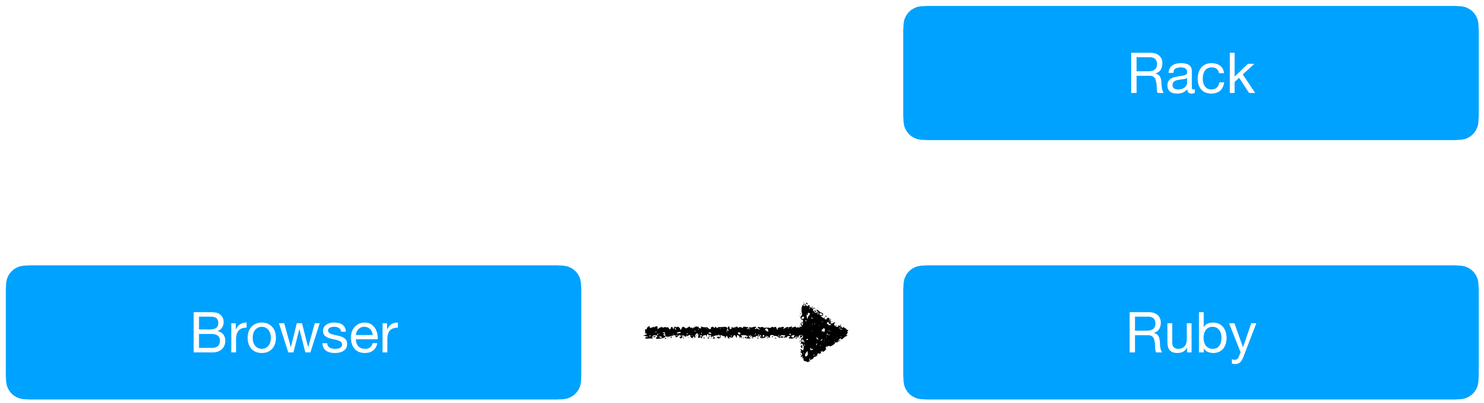
Real-time streaming.

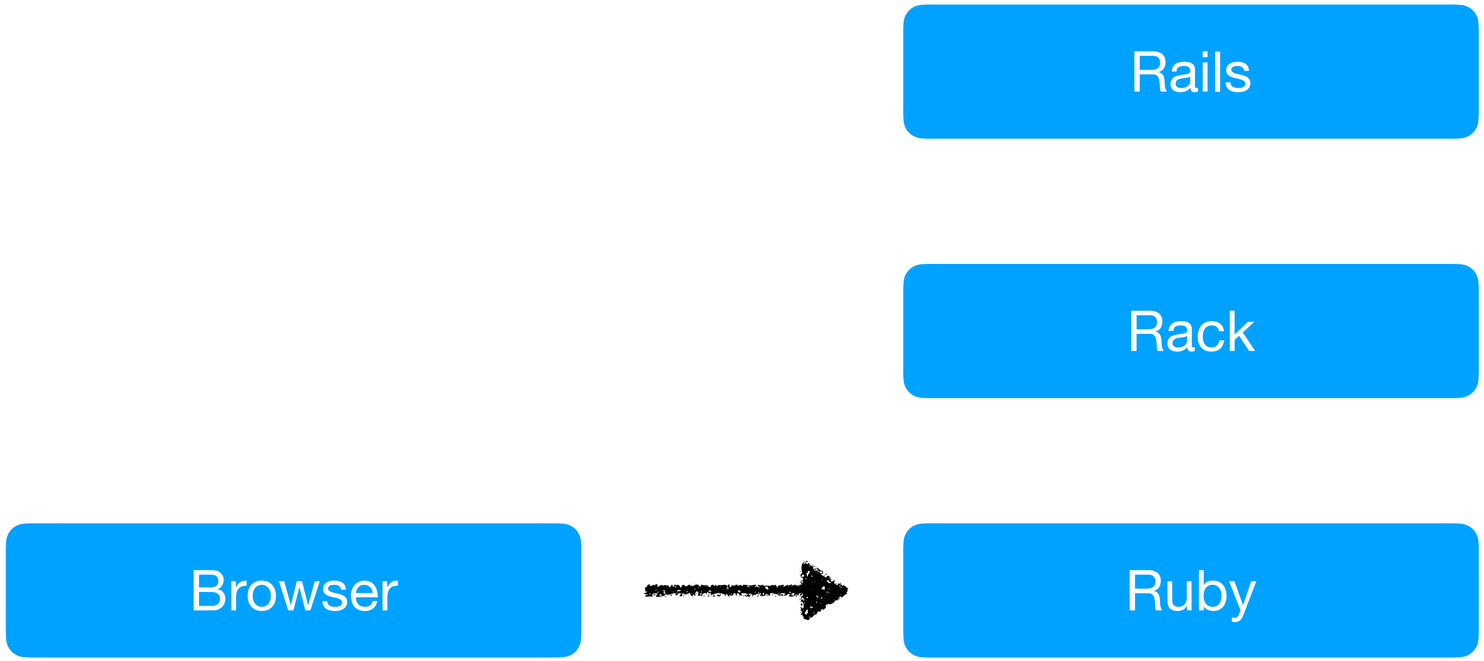
Big problems.

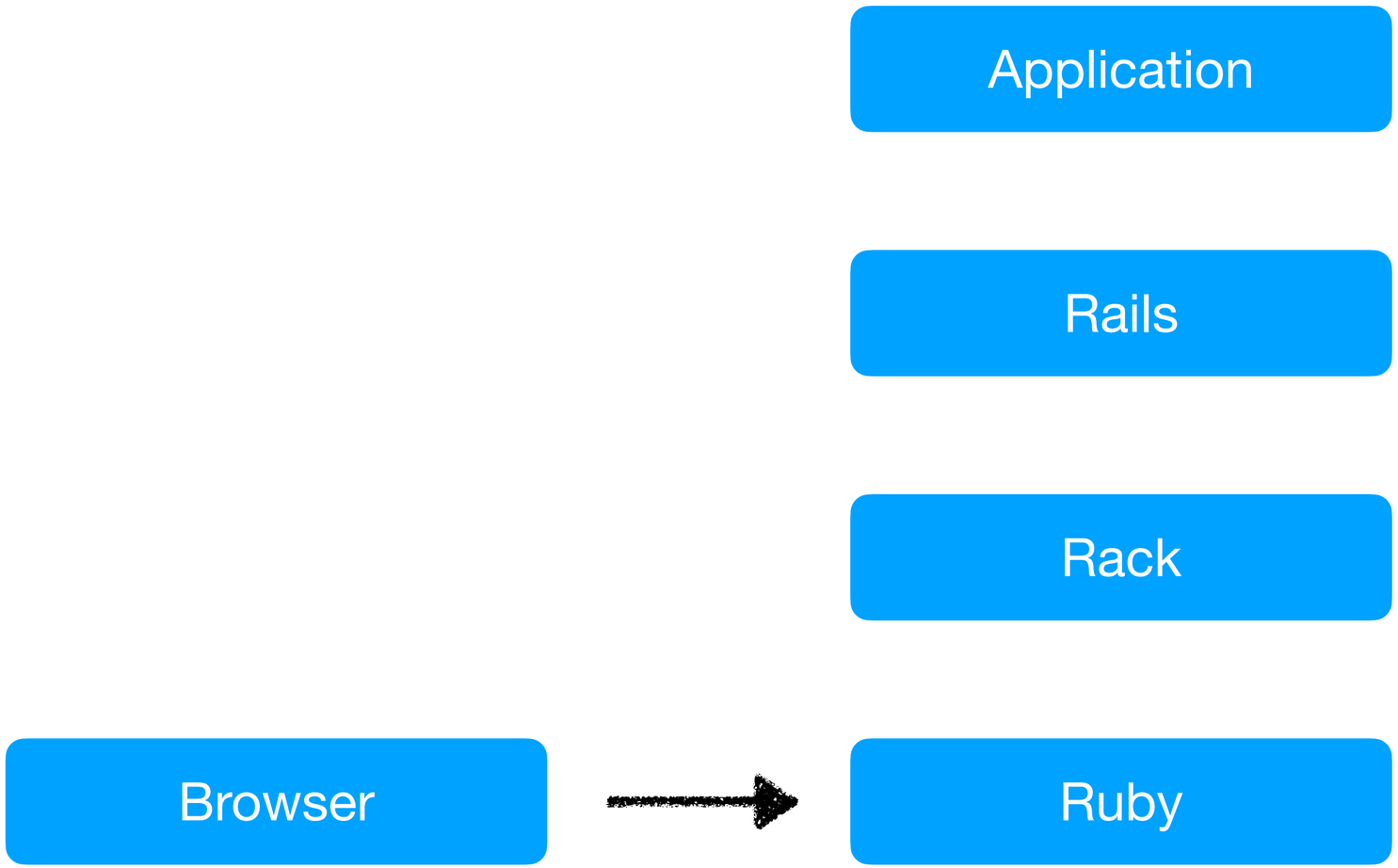
What about Rails?

Browser


Browser
Ruby







How did we change Ruby?

What are we waiting for?

Database Queries

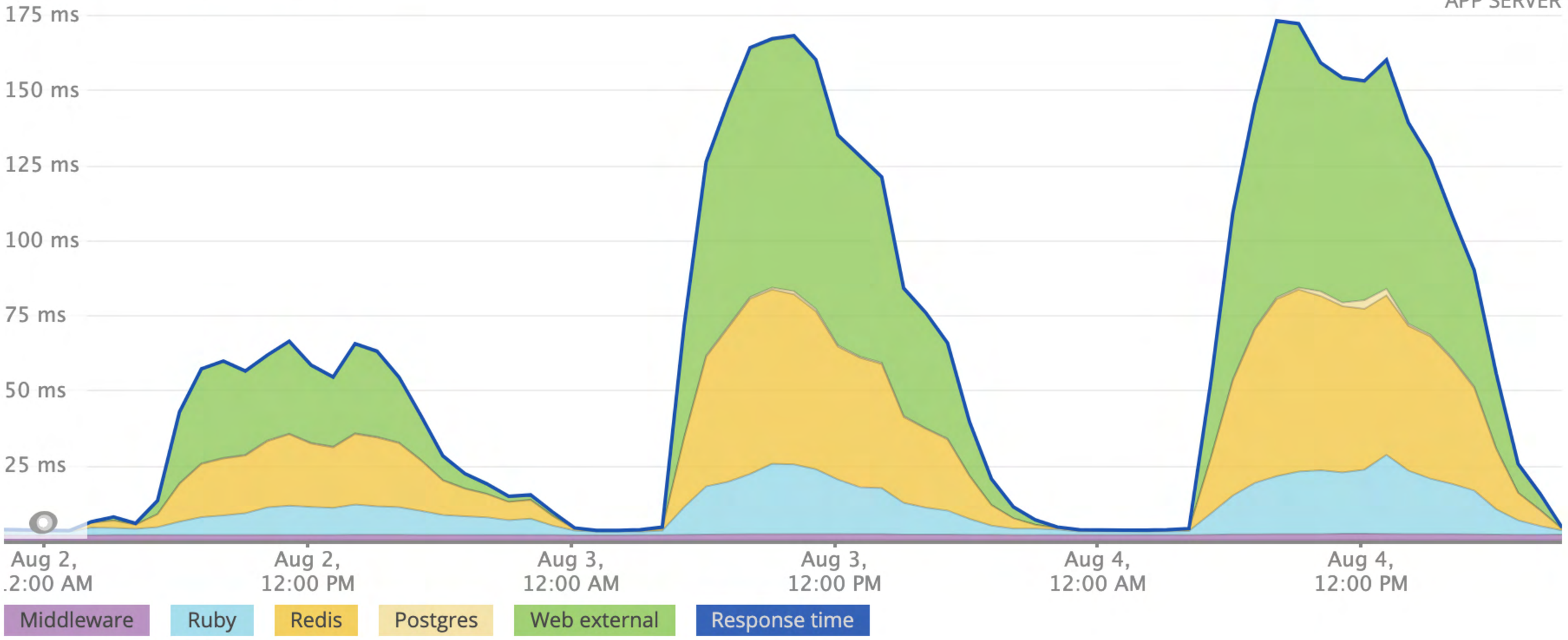
HTTP Requests

Redis Operations

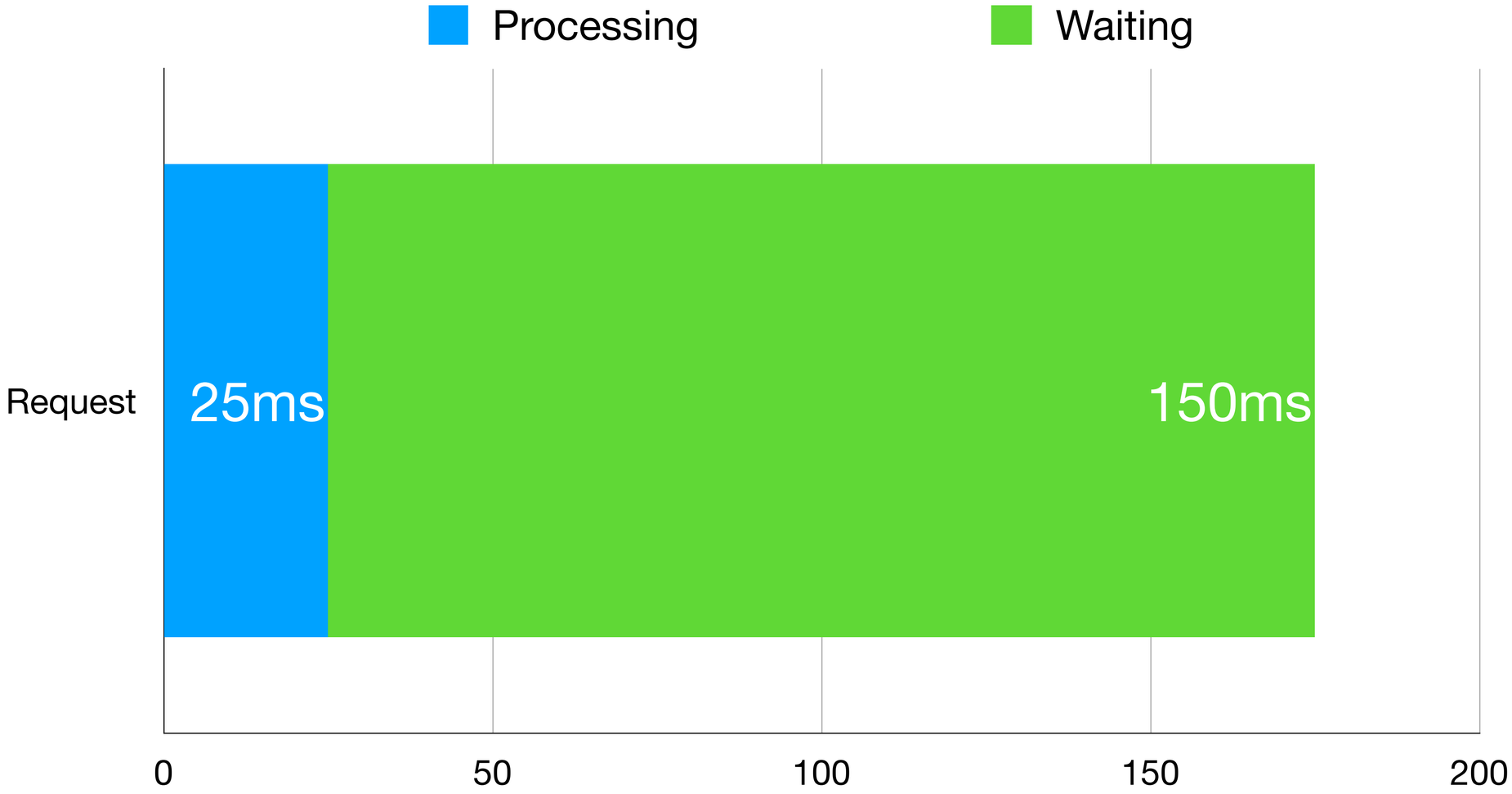
SERVERS
All servers

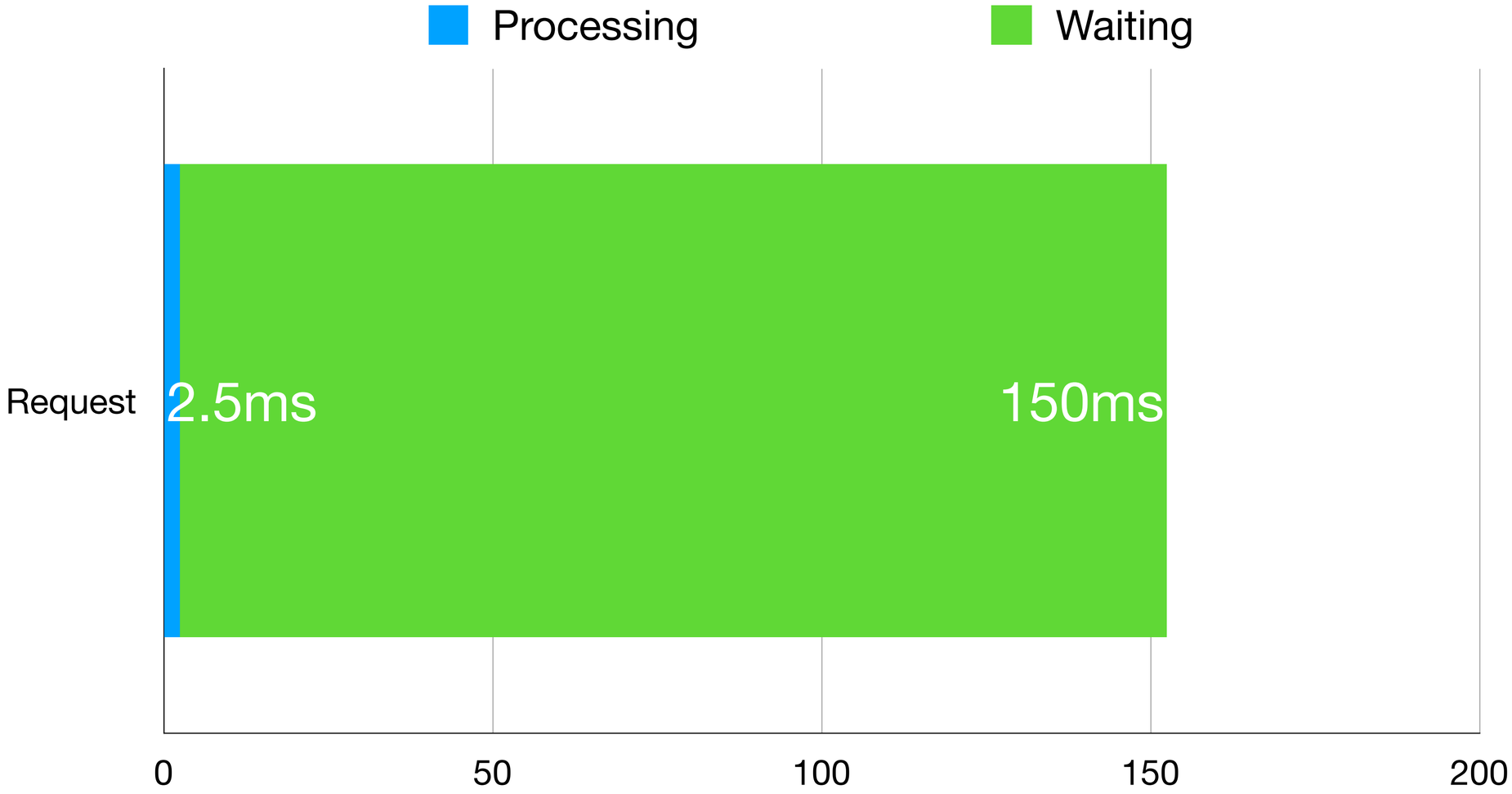
Web transactions time

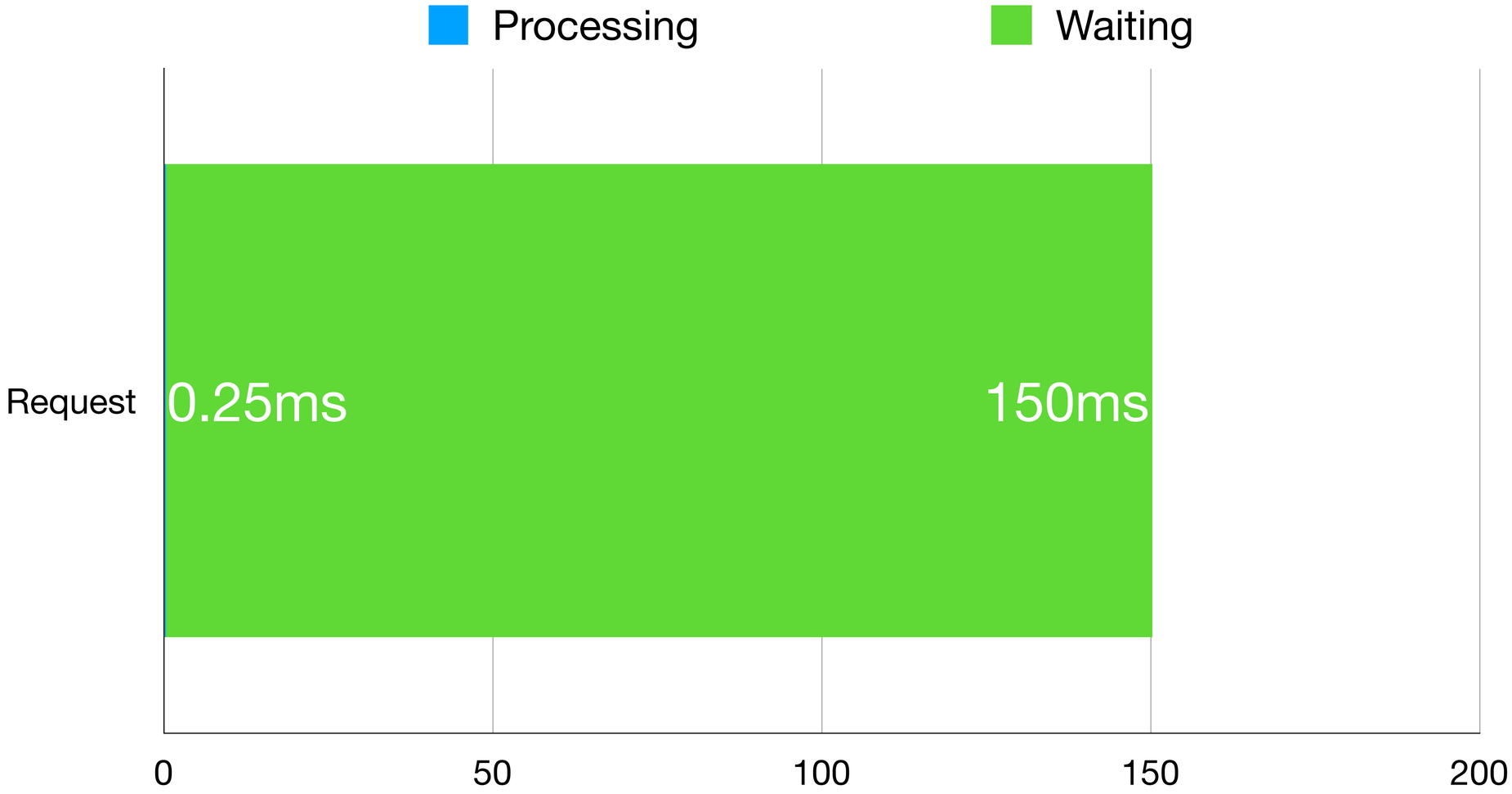
64.7 ms
APP SERVER



Can’t we just make Ruby faster?

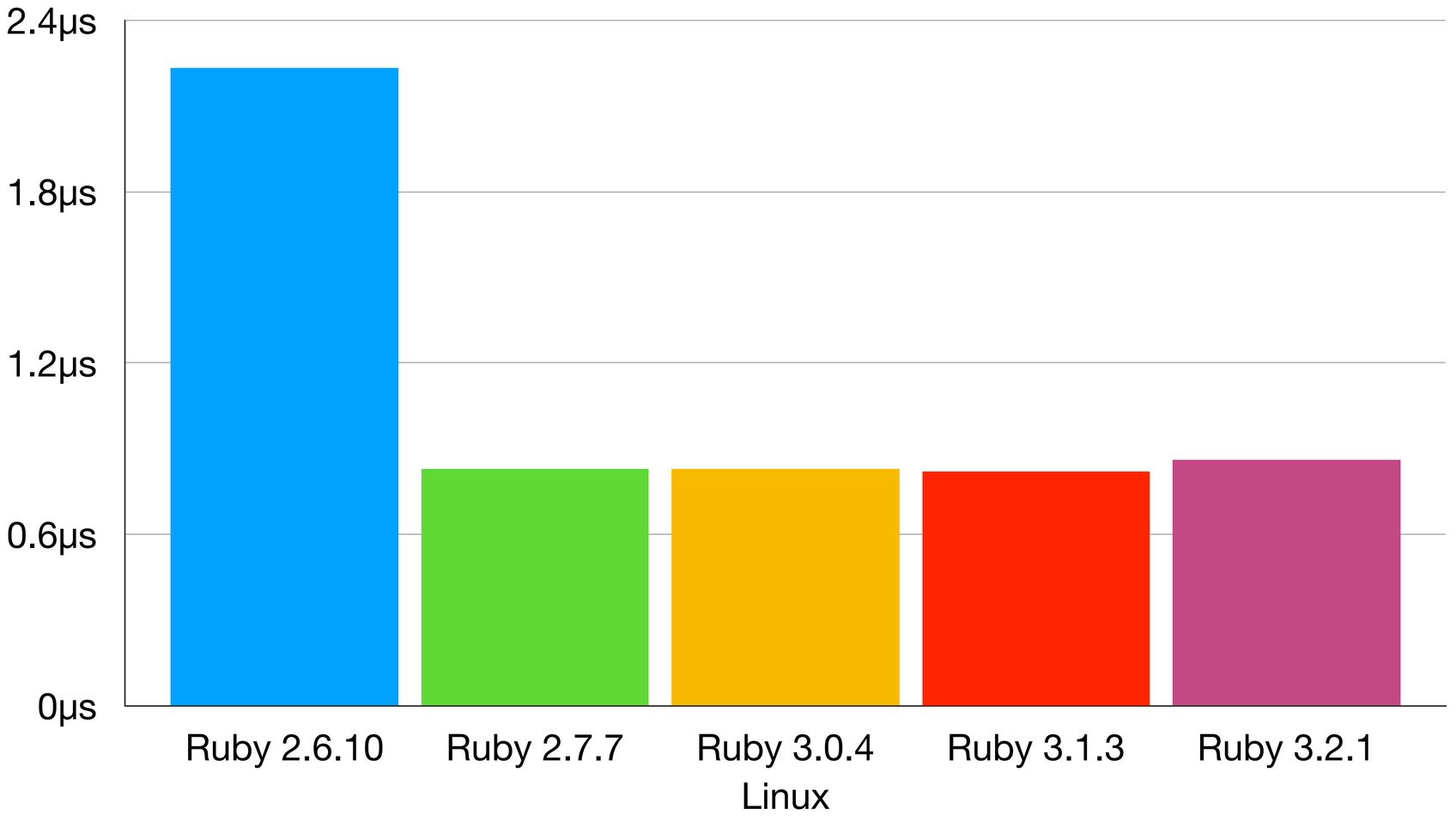




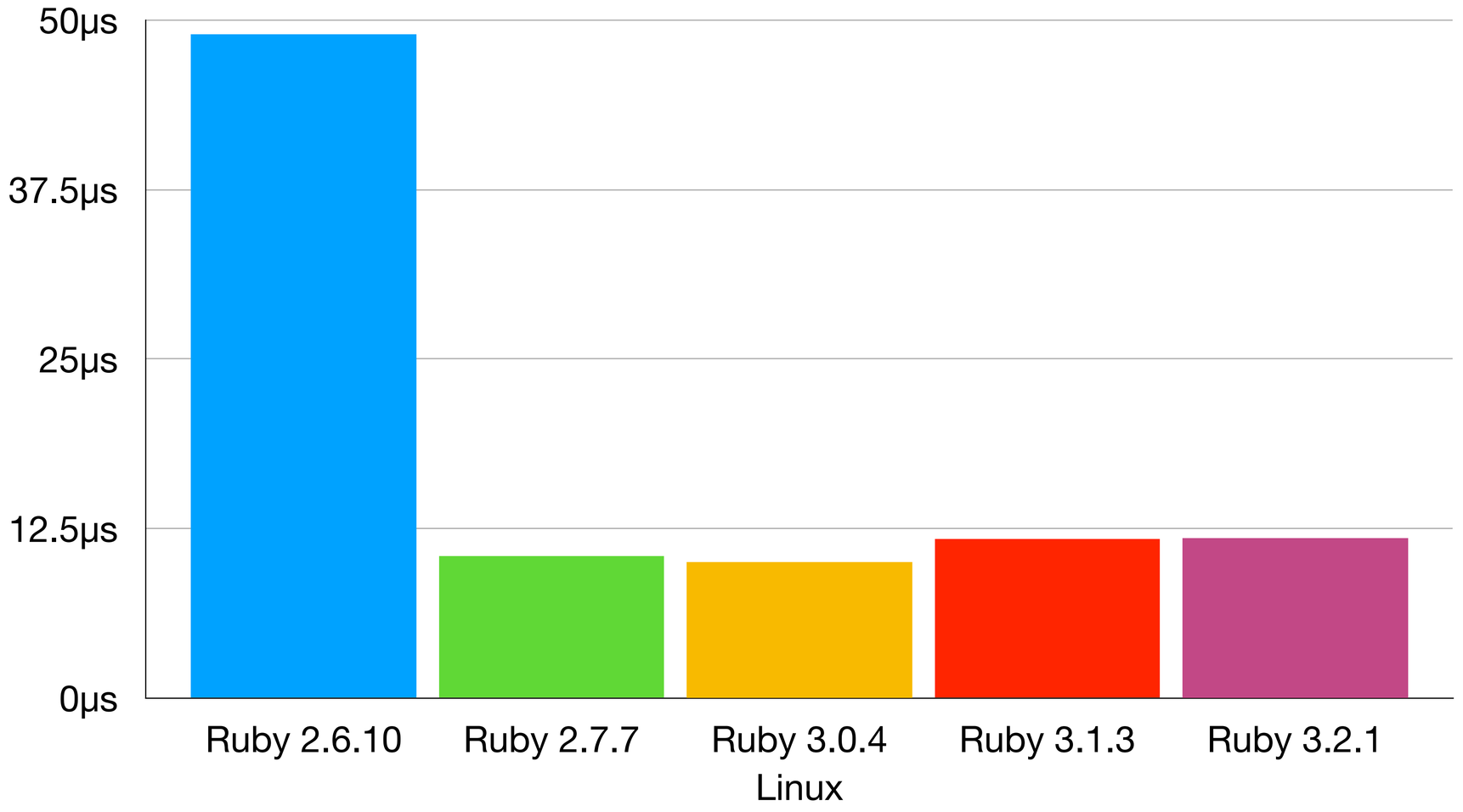


AU RUBY CONF -2023-

Fiber Create / Resume Time



Thread Create / Join Time

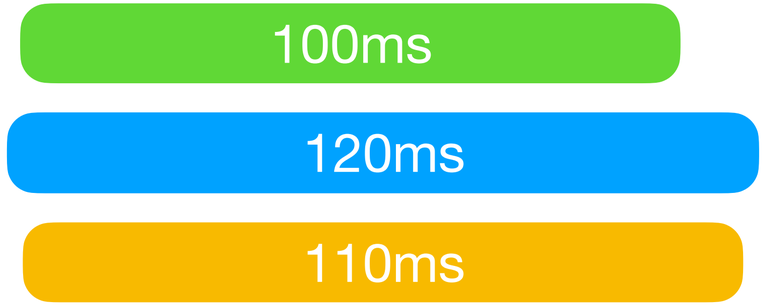


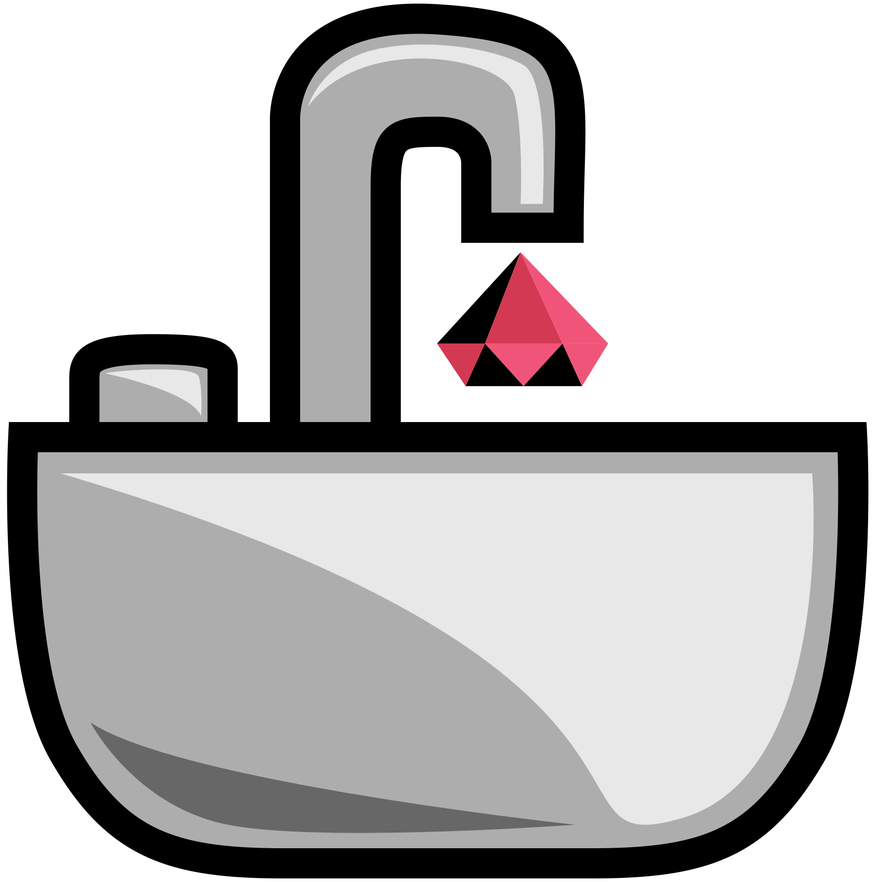
How do we reduce waiting time?

What are fibers?

Total Execution Time: 330ms

Total Execution Time: 120ms





ASYNC

AU RUBY CONF -2023-

Feature #14736 CLOSED

Edit Unwatch Copy ...

Thread selector for flexible cooperative fiber based concurrency

Added by ioquatix (Samuel Williams) about 3 years ago. Updated over 1 year ago.

Status:	Closed
Priority:	Normal
Assignee:	-
Target version:	-

[ruby-core:86873]

Description

Quote

Ruby concurrency can be greatly enhanced by concurrent, deterministic IO.

Fibers have been shown many times to be a great abstraction for this purpose. The retain normal code flow and don't require any kind of Thread synchronisation. They are enjoyable to write code with because you don't have to concern yourself with thread synchronisation or other complicated issues.

The basic idea is that if some operation would block, it yields the Fiber, and other Fibers within the thread can continue to execute.

There are a number of ways to implement this. Here is a proof of concept to amend the existing `rb_io_wait_readable`/`rb_io_wait_writable`.

https://github.com/ruby/ruby/pull/1870

This design minimally affects the Ruby implementation and allows flexibility for selector implementation. With a small amount of work, we can support EventMachine (65 million downloads), NIO4r (21 million downloads). It would be trivial to back port.

This PR isn't complete but I am seeking feedback. If it's a good idea, I will do my best to see it through to completion, including support for EventMachine and NIO4r.

Scheduler

The per-thread fiber scheduler interface is used to intercept blocking operations. A typical implementation would be a wrapper for a gem like EventMachine or Async. This design provides separation of concerns between the event loop implementation and application code. It also allows for layered schedulers which can perform instrumentation, enforce constraints (e.g. during testing) and provide additional logging. You can see a sample implementation here.

```ruby
class Scheduler
  # Wait for the given file descriptor to become readable.
  def wait_readable(io)
  end

  # Wait for the given file descriptor to become writable.
  def wait_writable(io)
  end

  # Wait for the given file descriptor to match the specified events within
  # the specified timeout.
  # @param event [Integer] a bit mask of +IO::WAIT_READABLE+,
  #   `IO::WAIT_WRITABLE` and `IO::WAIT_PRIORITY`.
  # @param timeout [#to_f] the amount of time to wait for the event.
  def wait_any(io, events, timeout)
  end

  # Sleep the current task for the specified duration, or forever if not
  # specified.
  # @param duration [#to_f] the amount of time to sleep.
  def wait_sleep(duration = nil)
  end

  # The Ruby virtual machine is going to enter a system level blocking
  # operation.
  def enter_blocking_region
  end

  # The Ruby virtual machine has completed the system level blocking
```

What operations are hooked?

IO & Sockets.

```
Async do
  socket = Socket.tcp('www.google.com', 80)
  socket.write("GET / HTTP/1.1\r\nHost: www.google.com\r\n\r\n")
  while line = socket.gets
    puts line
  end
end
```

```ruby
Async do
  socket = Socket.tcp('www.google.com', 80)
  socket.write("GET / HTTP/1.1\r\nHost: www.google.com\r\n\r\n")
  while line = socket.gets
    puts line
  end
end
```

```
Async do
  socket = Socket.tcp('www.google.com', 80)
  socket.write("GET / HTTP/1.1\r\nHost: www.google.com\r\n\r\n")
  while line = socket.gets
    puts line
  end
end
```

Thread#join

```ruby
Async do
  threads = 3.times.map do
    Thread.new do
      # Legacy code
      sleep 1
    end
  end

  # Non-blocking join - won't block the reactor.
  threads.each(&:join)
end
```

```ruby
Async do
  threads = 3.times.map do
    Thread.new do
      # Legacy code
      sleep 1
    end
  end

  # Non-blocking join - won't block the reactor.
  threads.each(&:join)
end
```

```ruby
Async do
  threads = 3.times.map do
    Thread.new do
      # Legacy code
      sleep 1
    end
  end

  # Non-blocking join - won't block the reactor.
  threads.each(&:join)
end
```

Thread::Mutex#lock

```ruby
items = [1, 1]
guard = Thread::Mutex.new

thread = Thread.new do
  5.times do |i|
    guard.synchronize {items << items[-1] + items[-2]}
  end
end

Async do
  5.times do |i|
    guard.synchronize {items << items[-1] + items[-2]}
  end
end

thread.join
p items # [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

```ruby
items = [1, 1]
guard = Thread::Mutex.new

thread = Thread.new do
  5.times do |i|
    guard.synchronize {items << items[-1] + items[-2]}
  end
end

Async do
  5.times do |i|
    guard.synchronize {items << items[-1] + items[-2]}
  end
end

thread.join
p items # [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

```ruby
items = [1, 1]
guard = Thread::Mutex.new

thread = Thread.new do
  5.times do |i|
    guard.synchronize {items << items[-1] + items[-2]}
  end
end

Async do
  5.times do |i|
    guard.synchronize {items << items[-1] + items[-2]}
  end
end

thread.join
p items # [1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144]
```

`Thread::Queue#pop`

```ruby
queue = Thread::Queue.new

Thread.new do
  3.times do |i|
    queue.push(i)
    sleep 1
  end

  queue.close
end

Async do
  while i = queue.pop
    puts i # 1, 2, 3
  end
end
```

```ruby
queue = Thread::Queue.new

Thread.new do
  3.times do |i|
    queue.push(i)
    sleep 1
  end

  queue.close
end

Async do
  while i = queue.pop
    puts i # 1, 2, 3
  end
end
```

```ruby
queue = Thread::Queue.new

Thread.new do
  3.times do |i|
    queue.push(i)
    sleep 1
  end

  queue.close
end

Async do
  while i = queue.pop
    puts i # 1, 2, 3
  end
end
```

DNS Resolution

```ruby
Async do |task|
  5.times do
    # This operation uses non-blocking DNS resolution:
    Addrinfo.getaddrinfo("www.google.com", 80, :INET, :STREAM)
  end
end
```

```ruby
Async do |task|
  5.times do
    # This operation uses non-blocking DNS resolution:
    Addrinfo.getaddrinfo("www.google.com", 80, :INET, :STREAM)
  end
end
```

Process#wait

```ruby
require 'async'

Async do |task|
  5.times do
    task.async {system("sleep 1")}
  end
end

# Total execution time ~1 second.
```

```ruby
require 'async'

Async do |task|
  5.times do
    task.async {system("sleep 1")}
  end
end

# Total execution time ~1 second.
```

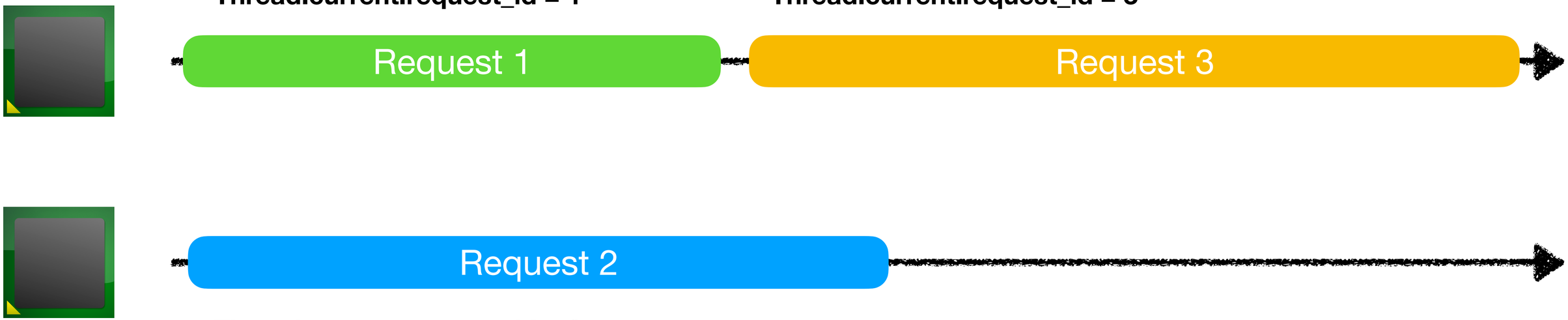
Safer Timeout

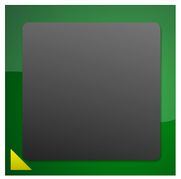
```ruby
require 'async'
require 'timeout'

Async do
  Timeout.timeout(1) do
    sleep(10)
  end
rescue ⇒ error
  p error # <Timeout::Error: execution expired>
end

# Total execution time ~1 second.
```

```ruby
require 'async'
require 'timeout'

Async do
  Timeout.timeout(1) do
    sleep(10)
  end
rescue ⇒ error
  p error # <Timeout::Error: execution expired>
end

# Total execution time ~1 second.
```

Transparently concurrent.

How do we handle asynchronous state?





Thread.current.request_id = 1

Request 1 → Request 2 → Request 1

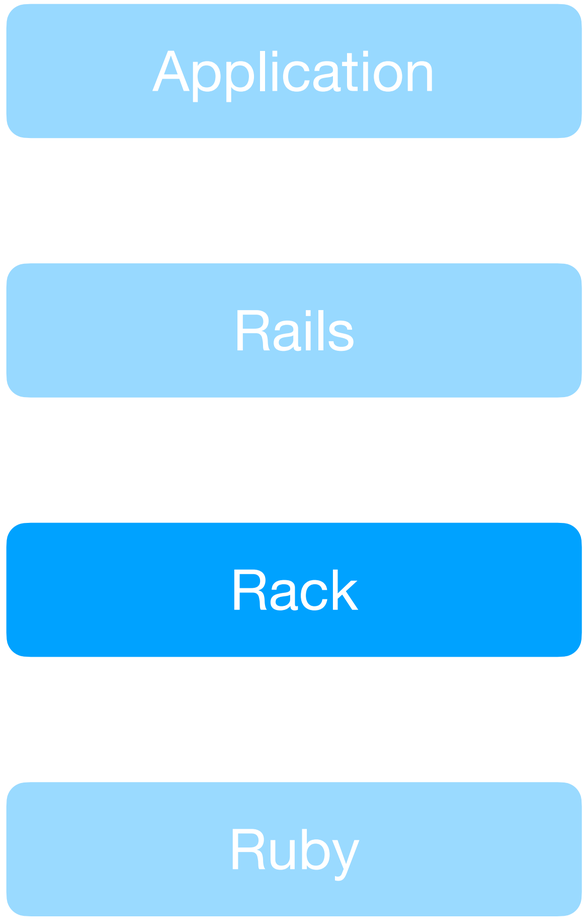
Thread.current.request_id = 2

```ruby
def handle_request
  Fiber[:request_id] ||= SecureRandom.uuid

  jobs = 3.times.map do
    Thread.new do
      puts "Starting job #{Fiber[:request_id]}"
      sleep 1
    end
  end

  jobs.each(&:join)
end
```

```ruby
def handle_request
  Fiber[:request_id] ||= SecureRandom.uuid

  jobs = 3.times.map do
    Thread.new do
      puts "Starting job #{Fiber[:request_id]}"
      sleep 1
    end
  end

  jobs.each(&:join)
end
```

```ruby
def handle_request
  Fiber[:request_id] ||= SecureRandom.uuid

  jobs = 3.times.map do
    Thread.new do
      puts "Starting job #{Fiber[:request_id]}"
      sleep 1
    end
  end

  jobs.each(&:join)
end
```

How did we change Rack?



What purpose does Rack serve?

What protocol does Rack implement?

```
def call(env)
  [200, {"Content-Type" ⇒ "text/plain"}, ["Hello World"]]
end
```

What were the limitations?

AU RUBY CONF -2023-

Bi-directional Streaming

```ruby
run do |env|
  body = proc do |stream|
    stream.write "Hello World"
    stream.close
  end

  [200, {}, body]
end
```

```ruby
run do |env|
  body = proc do |stream|
    stream.write "Hello World"
    stream.close
  end

  [200, {}, body]
end
```

```ruby
run do |env|
  body = proc do |stream|
    stream.write "Hello World"
    stream.close
  end

  [200, {}, body]
end
```

```
require 'trenni/template'
TEMPLATE = Trenni::Template.load(<<~'HTML')
<!DOCTYPE html><html><head><title>Beer Song</title></head><body>
  <?r 99.downto(1) do |i| ?>
  <p>#{i} bottles of beer on the wall</br>
     <?r sleep 1 ?>#{i} bottles of beer</br>
     <?r sleep 1 ?>take one down, and pass it around</br>
     <?r sleep 1 ?>#{i-1} bottles of beer on the wall</br></p>
  <?r end ?>
</body></html>
HTML

run do |env|
  [200, {"content-type" ⇒ "text/html"}, TEMPLATE.to_proc]
end
```

```
require 'trenni/template'
TEMPLATE = Trenni::Template.load(<<~'HTML')
<!DOCTYPE html><html><head><title>Beer Song</title></head><body>
  <?r 99.downto(1) do |i| ?>
  <p>#{i} bottles of beer on the wall</br>
     <?r sleep 1 ?>#{i} bottles of beer</br>
     <?r sleep 1 ?>take one down, and pass it around</br>
     <?r sleep 1 ?>#{i-1} bottles of beer on the wall</br></p>
  <?r end ?>
</body></html>
HTML

run do |env|
  [200, {"content-type" ⇒ "text/html"}, TEMPLATE.to_proc]
end
```

```
require 'trenni/template'
TEMPLATE = Trenni::Template.load(<<~'HTML')
<!DOCTYPE html><html><head><title>Beer Song</title></head><body>
  <?r 99.downto(1) do |i| ?>
  <p>#{i} bottles of beer on the wall</br>
    <?r sleep 1 ?>#{i} bottles of beer</br>
    <?r sleep 1 ?>take one down, and pass it around</br>
    <?r sleep 1 ?>#{i-1} bottles of beer on the wall</br></p>
  <?r end ?>
</body></html>
HTML

run do |env|
  [200, {"content-type" ⇒ "text/html"}, TEMPLATE.to_proc]
end
```

```
require 'trenni/template'
TEMPLATE = Trenni::Template.load(<<~'HTML')
<!DOCTYPE html><html><head><title>Beer Song</title></head><body>
  <?r 99.downto(1) do |i| ?>
  <p>#{i} bottles of beer on the wall</br>
     <?r sleep 1 ?>#{i} bottles of beer</br>
     <?r sleep 1 ?>take one down, and pass it around</br>
     <?r sleep 1 ?>#{i-1} bottles of beer on the wall</br></p>
  <?r end ?>
</body></html>
HTML

run do |env|
  [200, {"content-type" ⇒ "text/html"}, TEMPLATE.to_proc]
end
```

```
> nghttp -y https://localhost:9292
```

```
> nghttp -y https://localhost:9292
<!DOCTYPE html><html><head><title>99 bottles of beer</title></
head><body>
 <p>99 bottles of beer on the wall</br>
    99 bottles of beer</br>
    take one down, and pass it around</br>
    98 bottles of beer on the wall</br></p>
 <p>98 bottles of beer on the wall</br>
    98 bottles of beer</br>
    take one down, and pass it around</br>
    97 bottles of beer on the wall</br></p>
                          … and so on …
```

```ruby
require 'async/websocket/adapters/rack'

app = lambda do |env|
  Async::WebSocket::Adapters::Rack.open(env) do |connection|
    while message = connection.read
      connection.write message
    end
  end or [404, {}, []]
end

run app
```

```ruby
require 'async/websocket/adapters/rack'

app = lambda do |env|
  Async::WebSocket::Adapters::Rack.open(env) do |connection|
    while message = connection.read
      connection.write message
    end
  end or [404, {}, []]
end

run app
```

```ruby
require 'async/websocket/adapters/rack'

app = lambda do |env|
  Async::WebSocket::Adapters::Rack.open(env) do |connection|
    while message = connection.read
      connection.write message
    end
  end or [404, {}, []]
end

run app
```

Streaming large data sets.

```ruby
require 'csv'

run do |env|
  body = proc do |stream|
    csv = CSV.new(stream)

    while true
      csv << ["Hello", "World"]
    end
  end

  [200, {'content-type' => 'text/csv'}, body]
end
```

```ruby
require 'csv'

run do |env|
  body = proc do |stream|
    csv = CSV.new(stream)

    while true
      csv << ["Hello", "World"]
    end
  end

  [200, {'content-type' ⇒ 'text/csv'}, body]
end
```

```ruby
require 'csv'

run do |env|
  body = proc do |stream|
    csv = CSV.new(stream)

    while true
      csv << ["Hello", "World"]
    end
  end

  [200, {'content-type' ⇒ 'text/csv'}, body]
end
```

```
> nghttp -y https://localhost:9292
```

```
> nghttp -y https://localhost:9292
Hello,World
Hello,World
Hello,World
Hello,World
Hello,World
Hello,World
Hello,World
Hello,World
Hello,World
Hello,World
Hello,World
Hello,World
Hello,World
Hello,World
Hello,World
Hello,World
Hello,World
Hello,World
Hello,World
```

Server sent events.

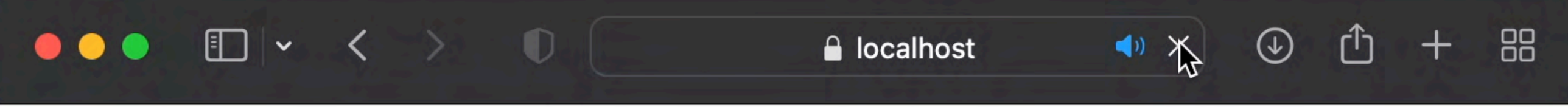
```ruby
run do |env|
  body = proc do |stream|
    while true
      stream.write("data: The time is #{Time.now}\n\n")
      sleep(1.0)
    end
  end

  [200, {'content-type' ⇒ 'text/event-stream',
    'cache-control' ⇒ 'no-cache'}, body]
end
```

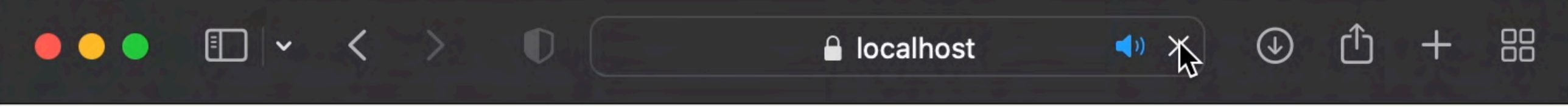
```ruby
run do |env|
  body = proc do |stream|
    while true
      stream.write("data: The time is #{Time.now}\n\n")
      sleep(1.0)
    end
  end

  [200, {'content-type' ⇒ 'text/event-stream',
    'cache-control' ⇒ 'no-cache'}, body]
end
```

```
<!doctype HTML>
<title>Time Service</title>
<div id="history"></div>
<script>
  var eventSource = new EventSource("/time");
  eventSource.addEventListener("message", function(event) {
    var container = document.createElement("p");
    container.innerText = event.data;
    var history = document.querySelector("#history");
    history.appendChild(container);
  });
</script>
```

```
<!doctype HTML>
<title>Time Service</title>
<div id="history"></div>
<script>
  var eventSource = new EventSource("/time");
  eventSource.addEventListener("message", function(event) {
    var container = document.createElement("p");
    container.innerText = event.data;
    var history = document.querySelector("#history");
    history.appendChild(container);
  });
</script>
```

```
<!doctype HTML>
<title>Time Service</title>
<div id="history"></div>
<script>
  var eventSource = new EventSource("/time");
  eventSource.addEventListener("message", function(event) {
    var container = document.createElement("p");
    container.innerText = event.data;
    var history = document.querySelector("#history");
    history.appendChild(container);
  });
</script>
```


localhost


localhost

How do we ensure servers all behave the right way?

Rack Conform

Rack::Conform::Application

$SERVER

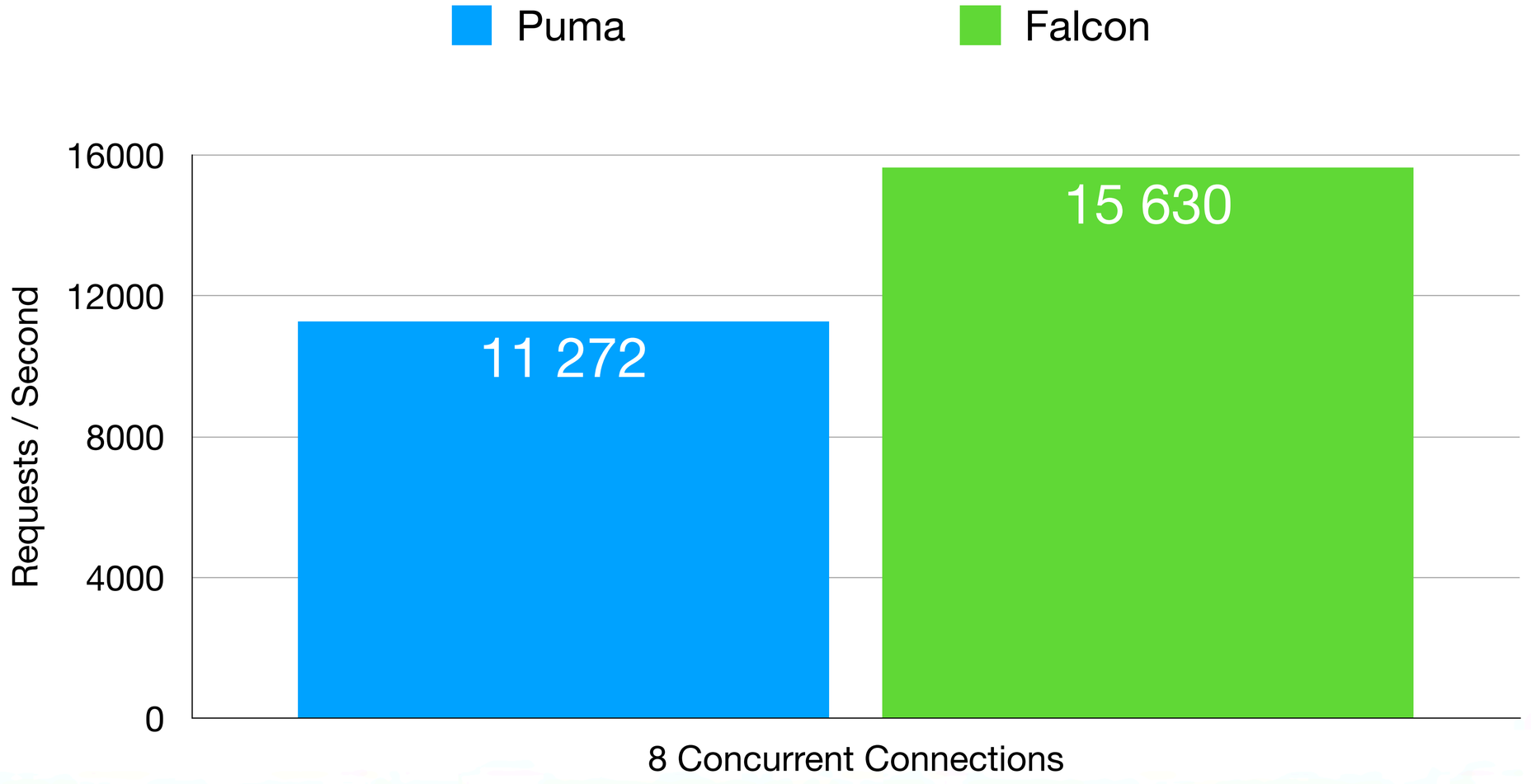
Rack Conform Test Suite

Server	Rack 2	Rack 3
Passenger	Pass	-
Unicorn	Pass	-
Puma v6.1	Pass	Pass
Falcon	Pass	Pass
Rackup / WEBRick	Pass	Pass
Thin	Pass	-

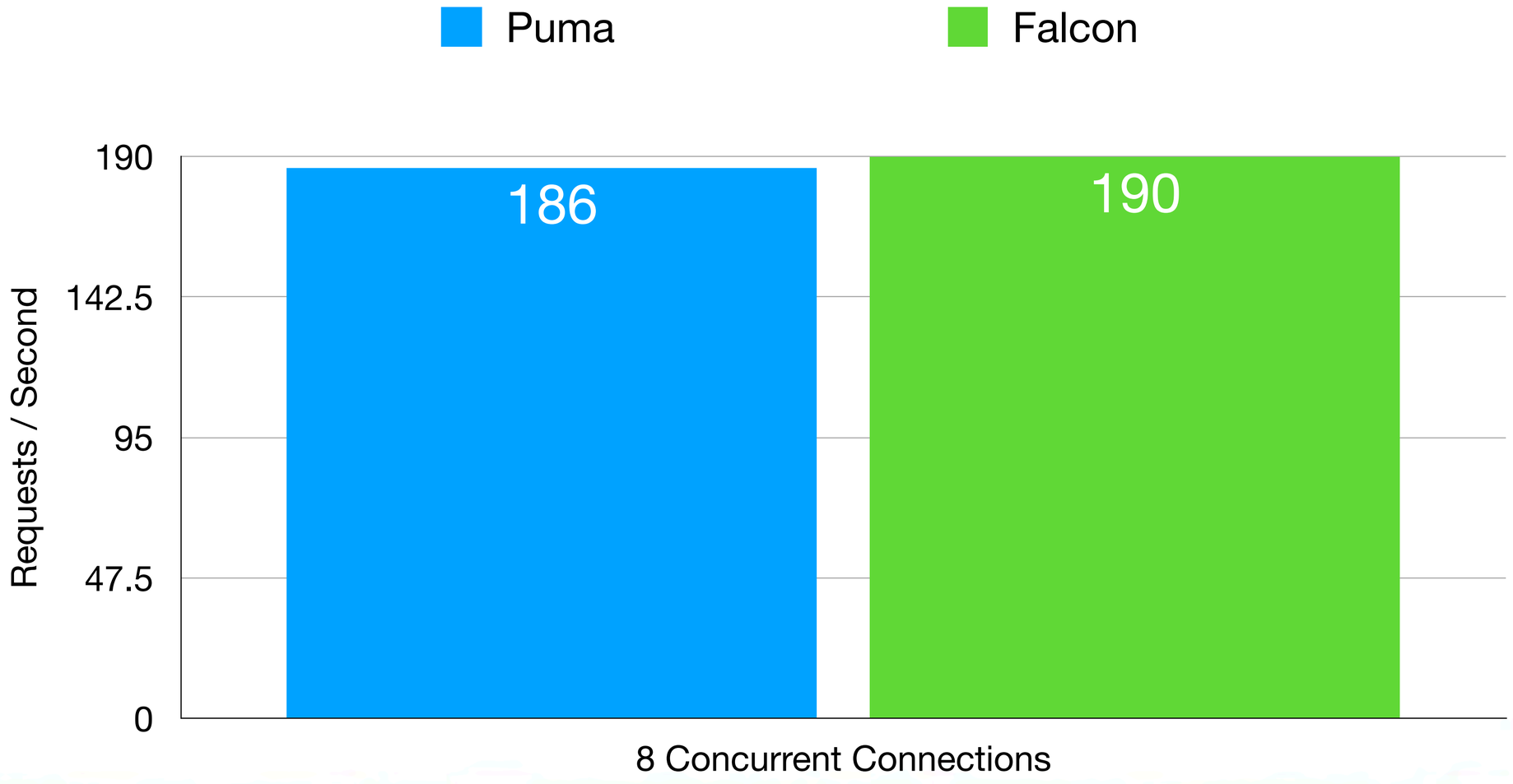
AU RUBY CONF -2023-

Fiber-based Concurrency

1.2KB Response



12MB Response



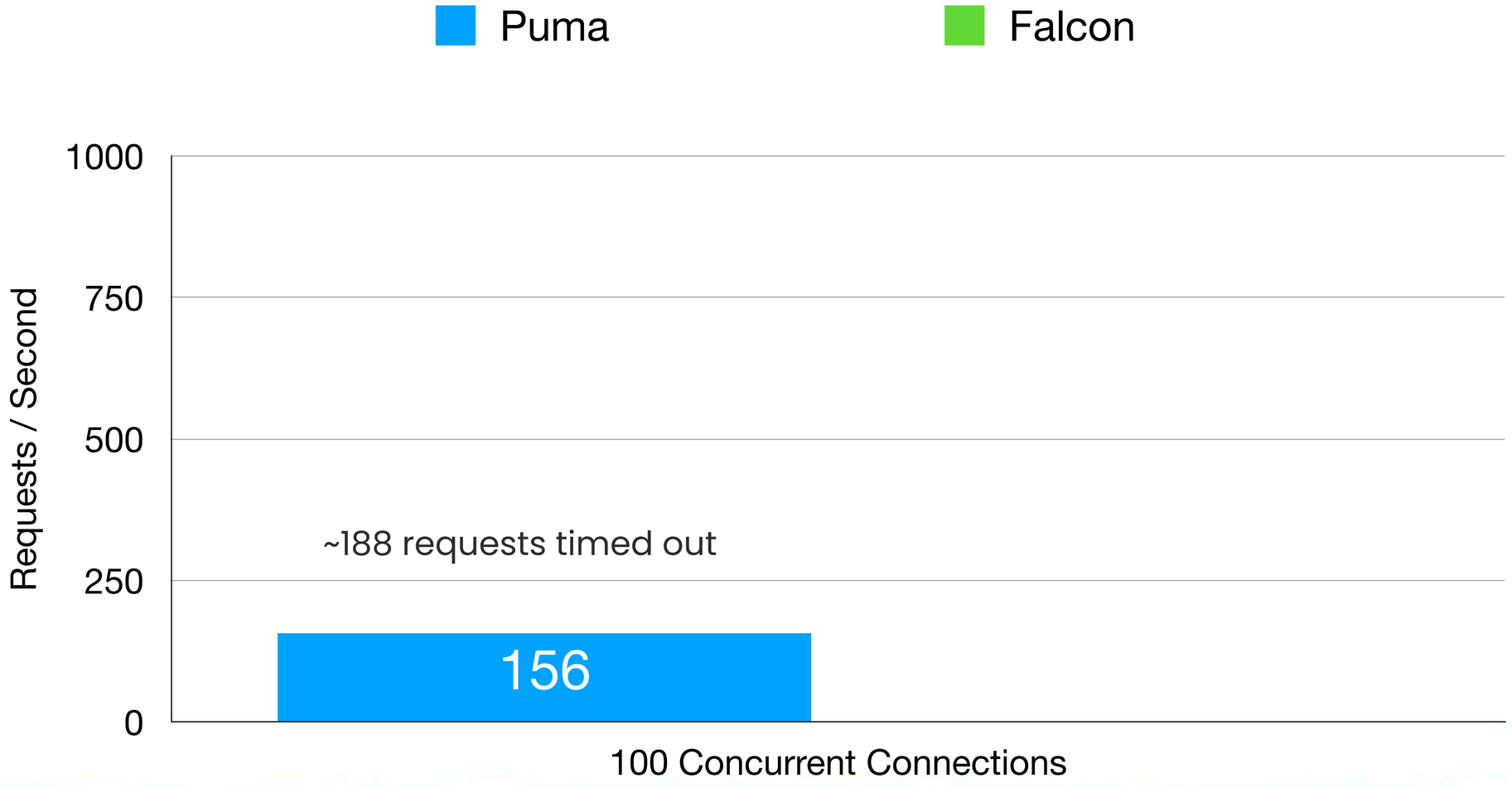
What about blocking operations?

```ruby
def call(env)
  sleep(0.1) # Simulate slow database query.
  [200, {}, []]
end
```

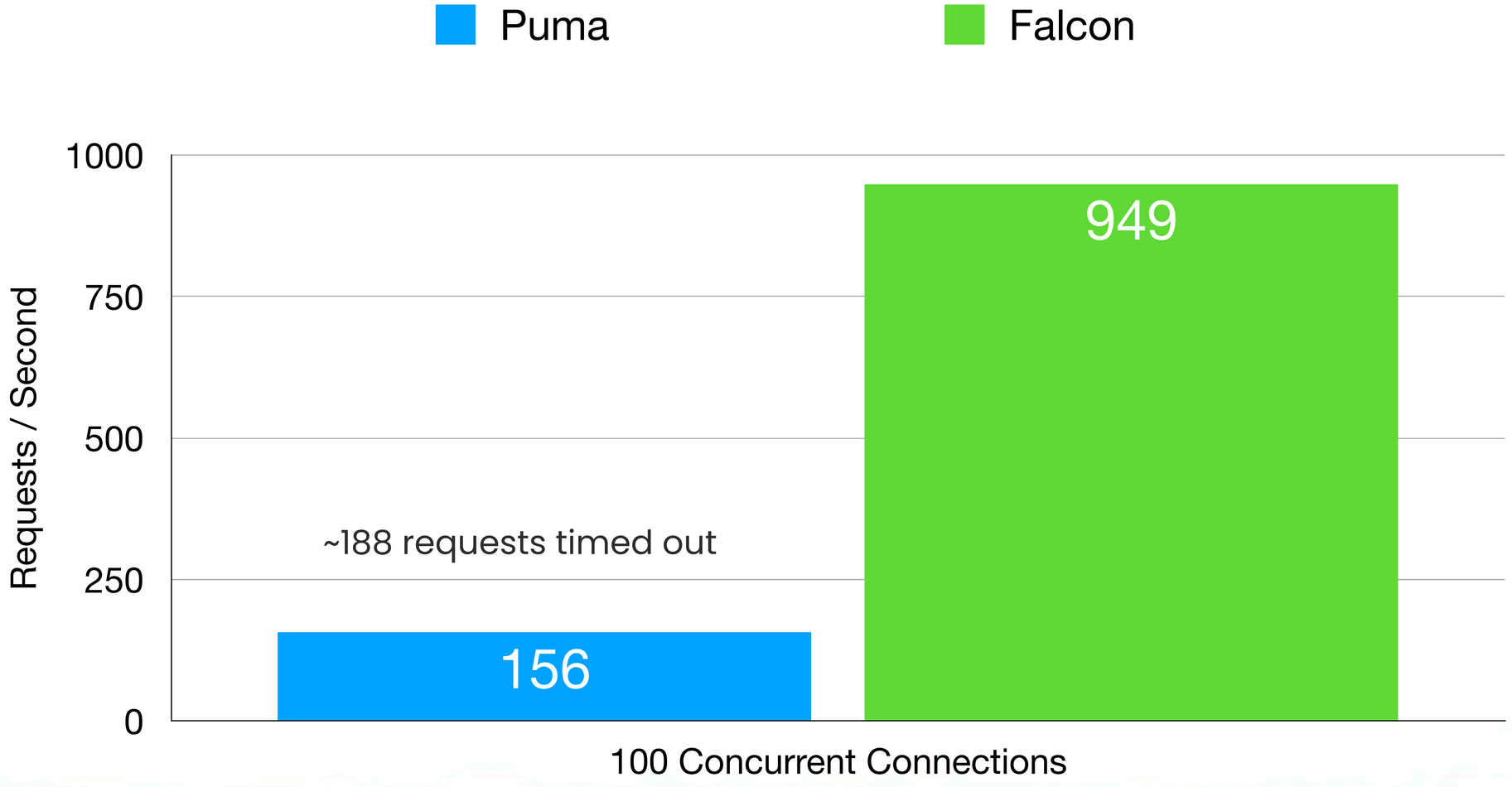
Sleep Response

~188 requests timed out

Sleep Response



Sleep Response

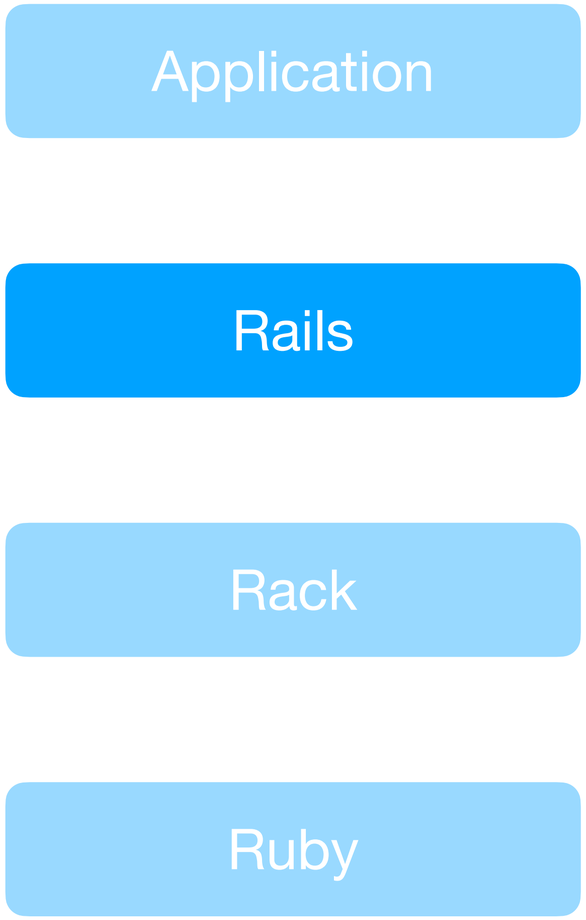


kinnalru 3 days ago

edited

After porting IO-bound (lot of aws-s3 uploads) project to falcon without any code modification it shows outstanding concurrency. S3 uploads take most of the request processing time.

How did we change Rails?



Introduce
`ActiveSupport::IsolatedExecutionState`
for internal use. #43596

Merged

Fiber-safe ConnectionPool. #44219

Merged

Don't assume Rack input supports rewindability. #47077

Improve matching of cookie assertions. #47066

Merged

Use Rack's own headers classes where appropriate. #47091

Add `rack-session` and `rackup` gems.

#47086

Merged

What database adapter can I use?

Just use Postgres.

If you use MySQL…

How do you change your application?

You don’t need to change anything.

Summary

Summary

- Ruby 3.2+ provides a great foundation for asynchronous programs and in particular concurrent execution.

Summary

- Ruby 3.2+ provides a great foundation for asynchronous programs and in particular concurrent execution.
- Rack 3 provides a standard (non-optional) interface for bidirectional streaming and enables things like WebSockets without escape hatches.

Summary

- Ruby 3.2+ provides a great foundation for asynchronous programs and in particular concurrent execution.
- Rack 3 provides a standard (non-optional) interface for bidirectional streaming and enables things like WebSockets without escape hatches.
- Rails 7.1+ is compatible with Rack 2 and Rack 3 and allows web applications to take advantage of all these new features.

Asynchronous Rails

Samuel Williams
ruby.social/@ioquatix