Ascenda

Real world application of Ractor

HIEU NGUYEN (hieuk09) – December 2023

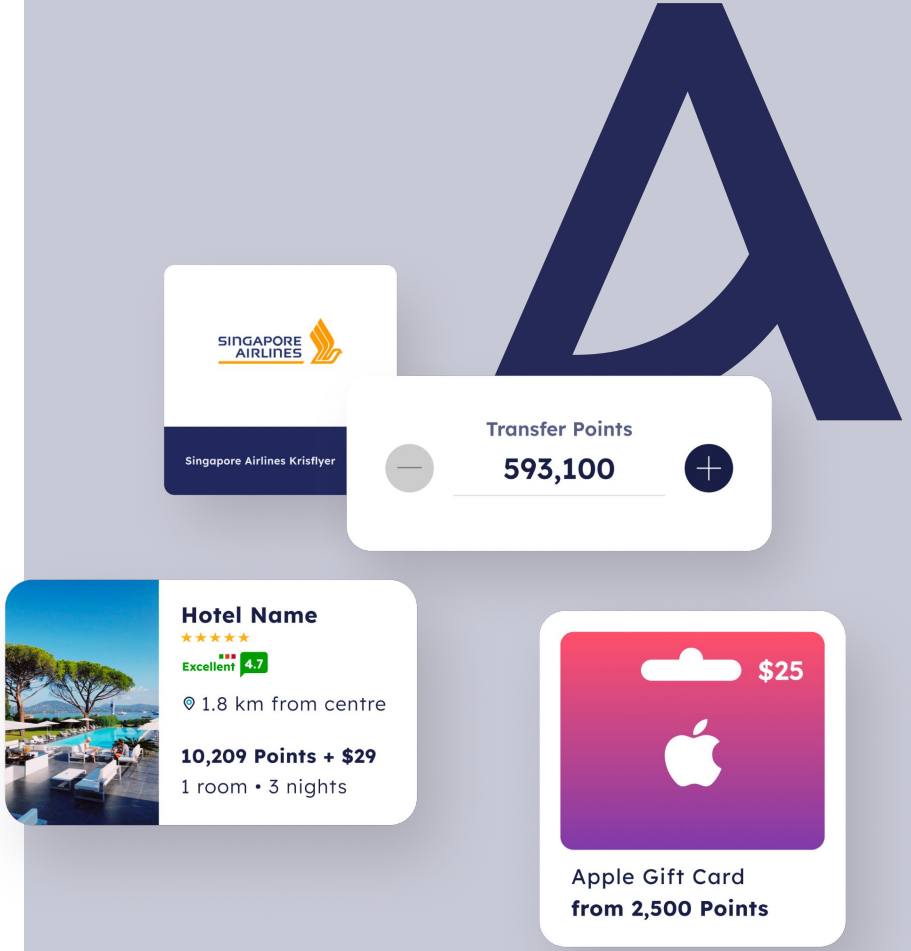
WHO ARE WE?

About Ascenda

B2B2C

Provide loyalty solutions for financial institutions

Built using Rails and Hanami



Agenda

01 Exploring Ractor

02 Use case in Ascenda: Encrypting PII Data

03 Application and benchmarks

04 Lessons learned

Ruby 3x3

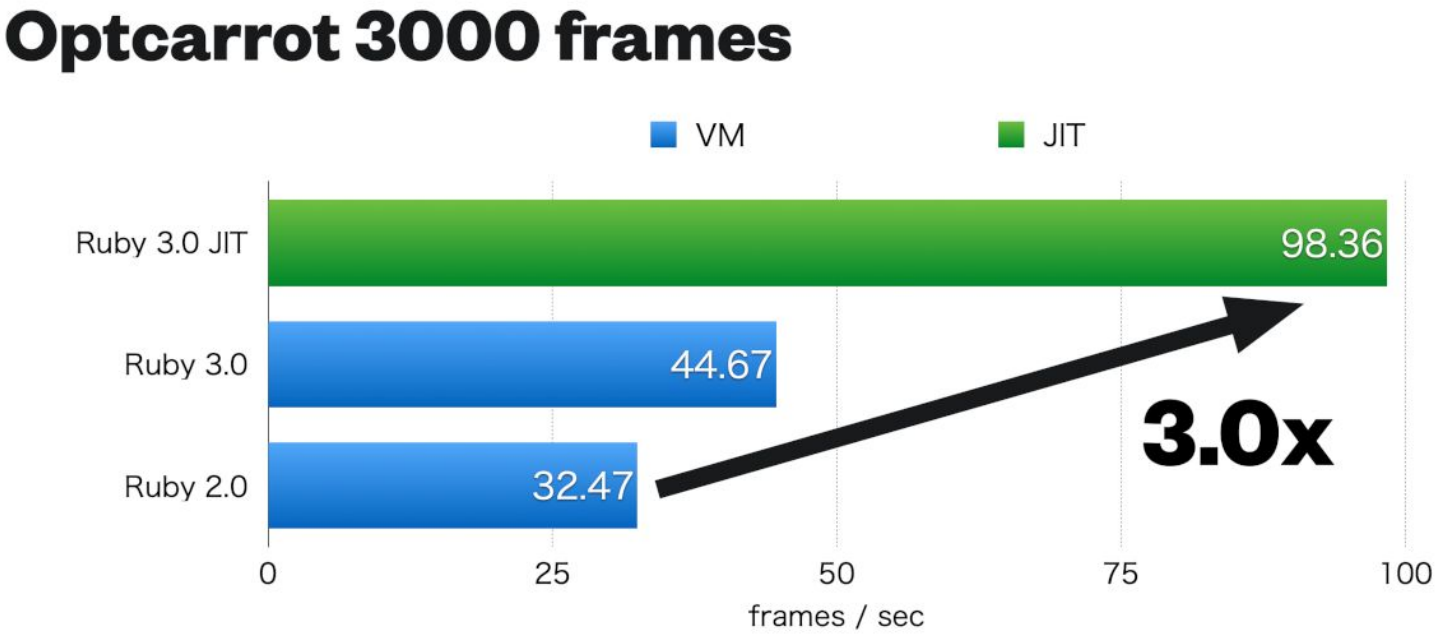


Image from https://goldenowl.asia/blog/ruby-3x3-is-it-actually-three-times-faster

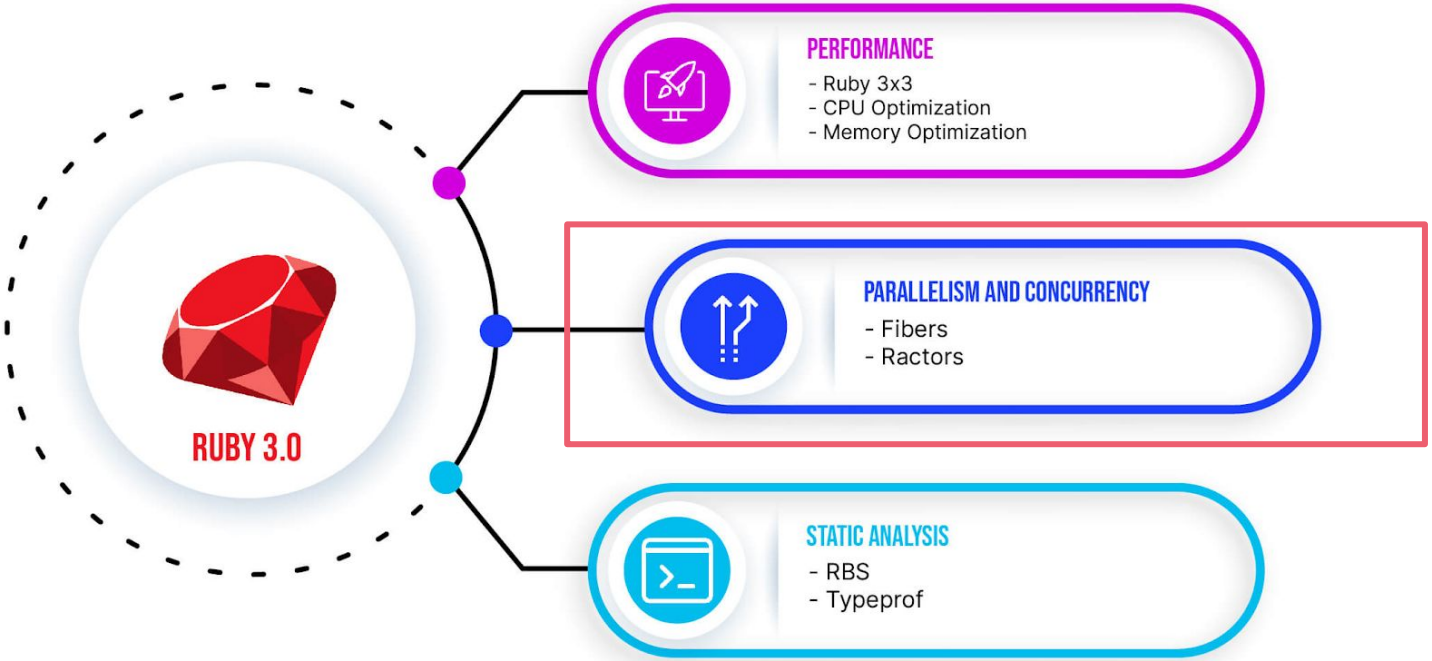
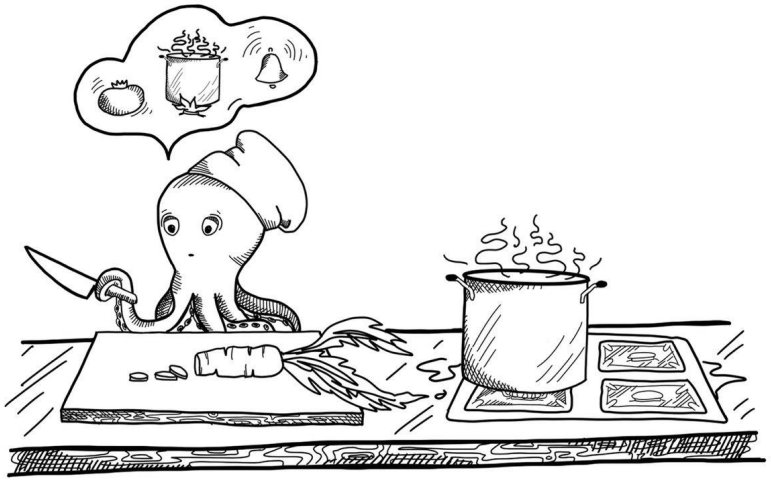
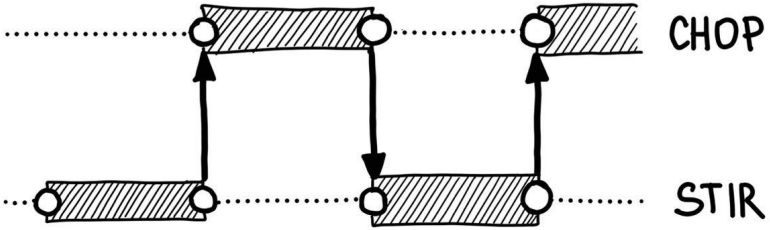


Image from https://goldenowl.asia/blog/ruby-3x3-is-it-actually-three-times-faster

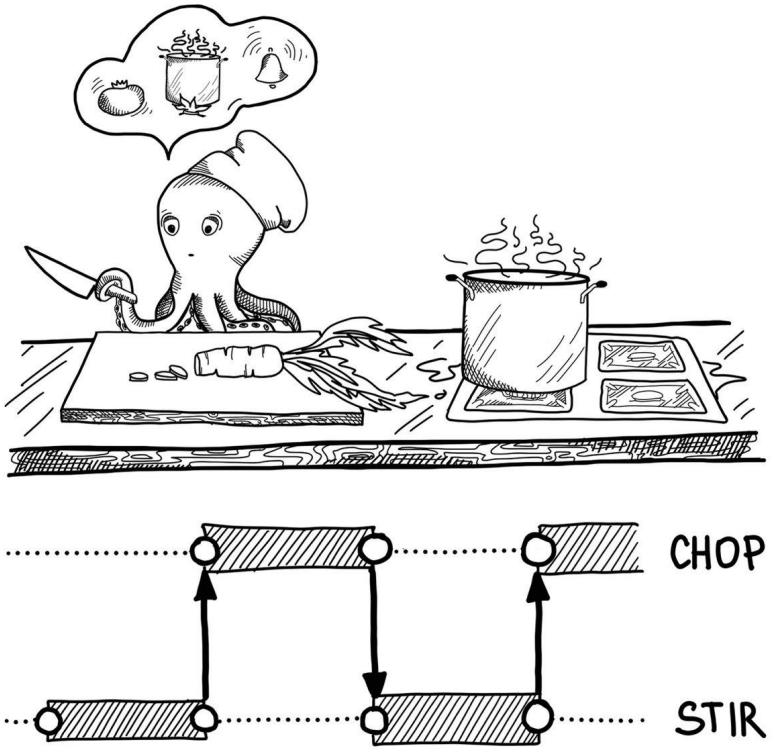
Concurrency

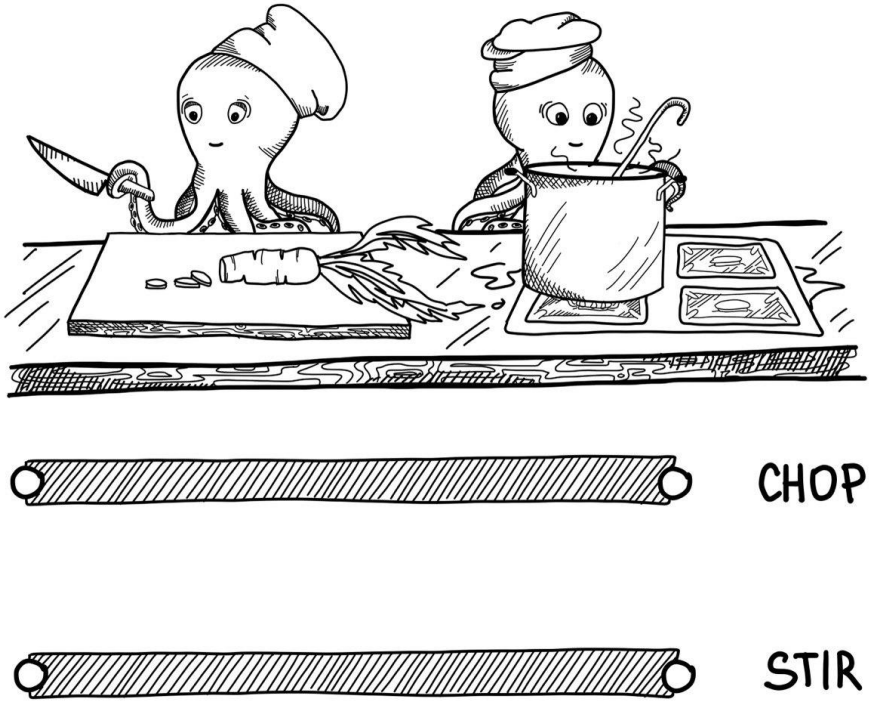




https://freecontent.manning.com/concurrency-vs-parallelism/

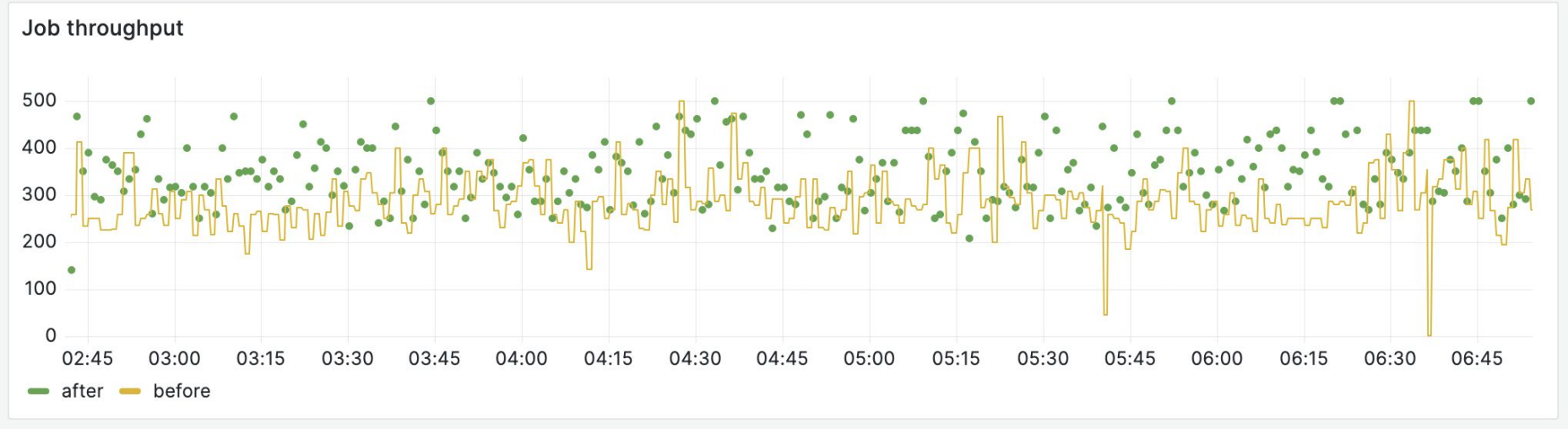
Parallelism



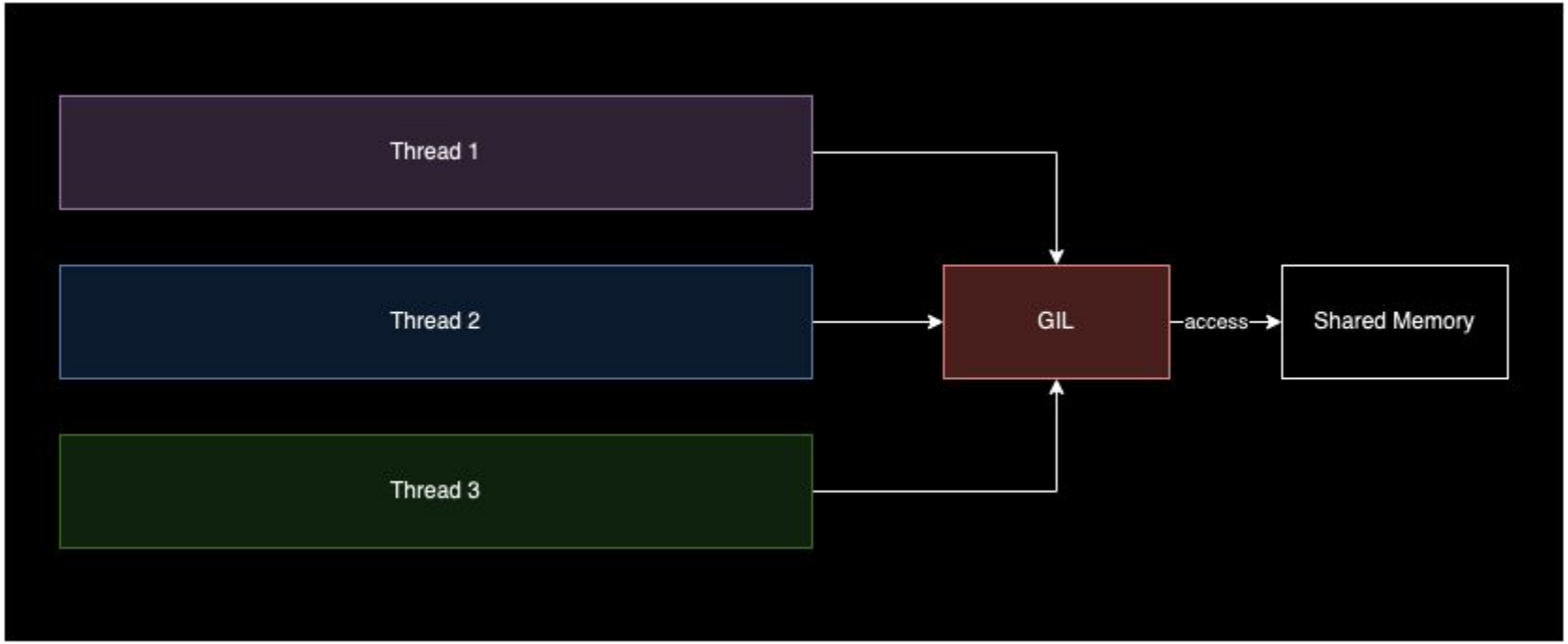


https://freecontent.manning.com/concurrency-vs-parallelism/

Real world application



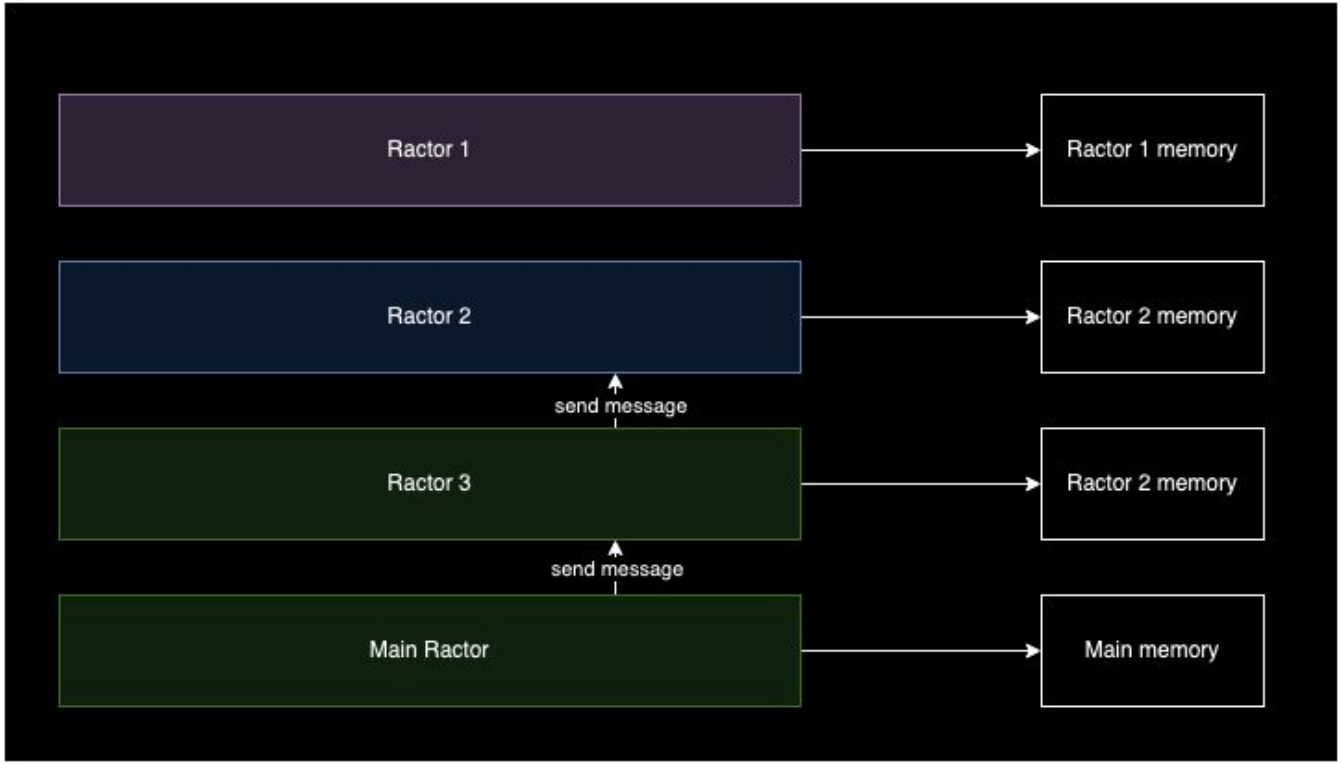
Thread



Thread

```
3.3.0+0 :001 > array = []
3.3.0+0 :002 > threads = 10.times.map do |i|
3.3.0+0 :003 >   Thread.new { array << i }
3.3.0+0 :004 > end.each(&:join)

3.3.0+0 :006 > puts array.inspect
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Ractor


Ractor 1
Ractor 1 memory
Ractor 2
Ractor 2 memory
send message
Ractor 3
Ractor 2 memory
send message
Main Ractor
Main memory

Ractor

Isolated

Message-based communication

Lightweight

Simplify concurrency

Improve performance

Drawbacks

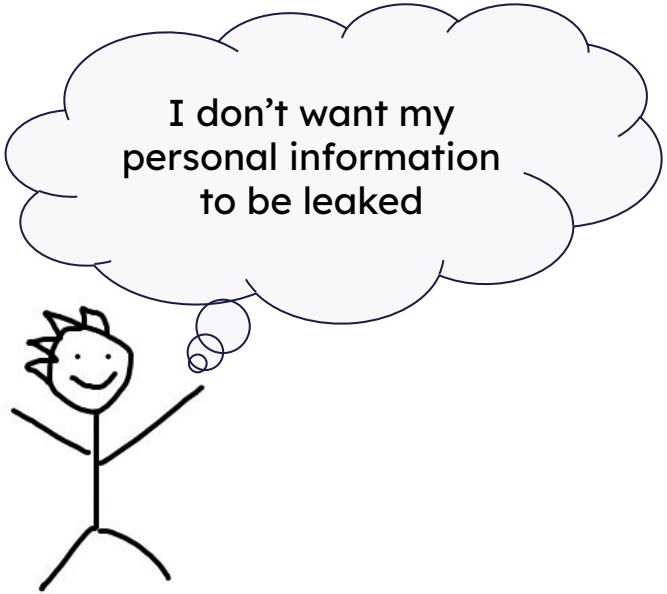
```
if __builtin_cexpr!("RBOOL(ruby_single_main_ractor)")
  warn("Ractor is experimental, and the behavior may change in future versions of Ruby! " \
       "Also there are many implementation issues.", uplevel: 0, category: :experimental)
end
```

Drawbacks

Unfamiliar interface

Lack of libraries support

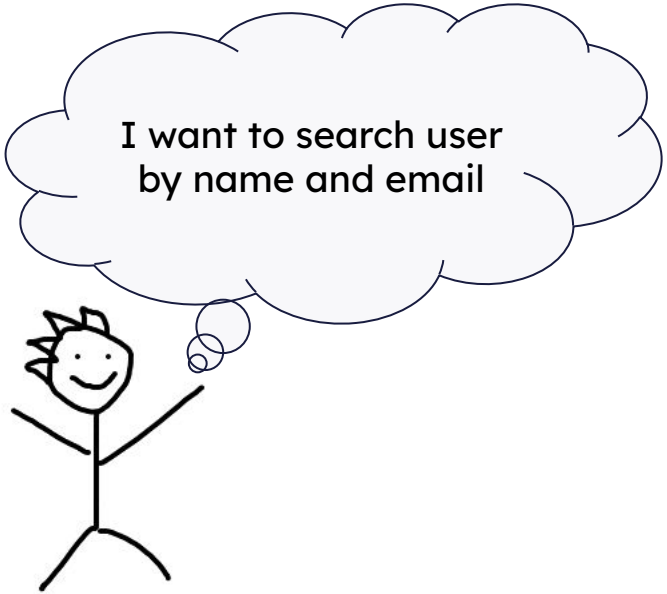
User Privacy



User

Engineer

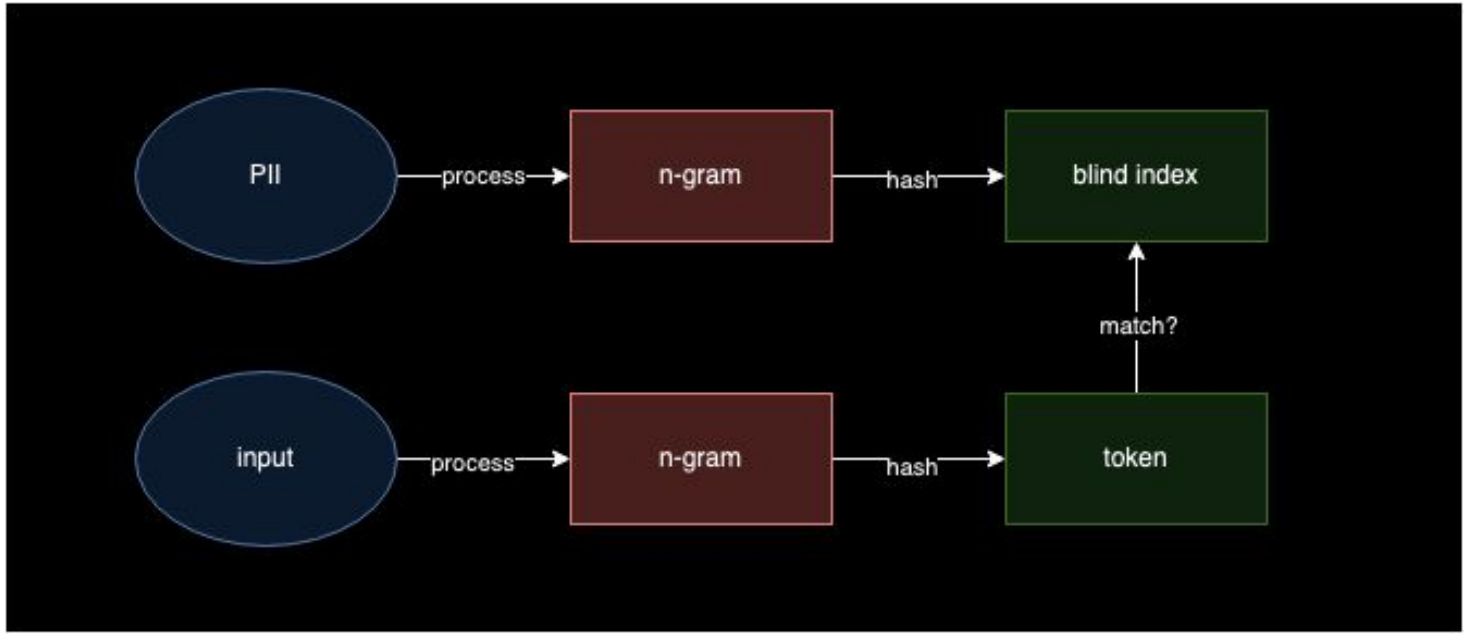
User Experience



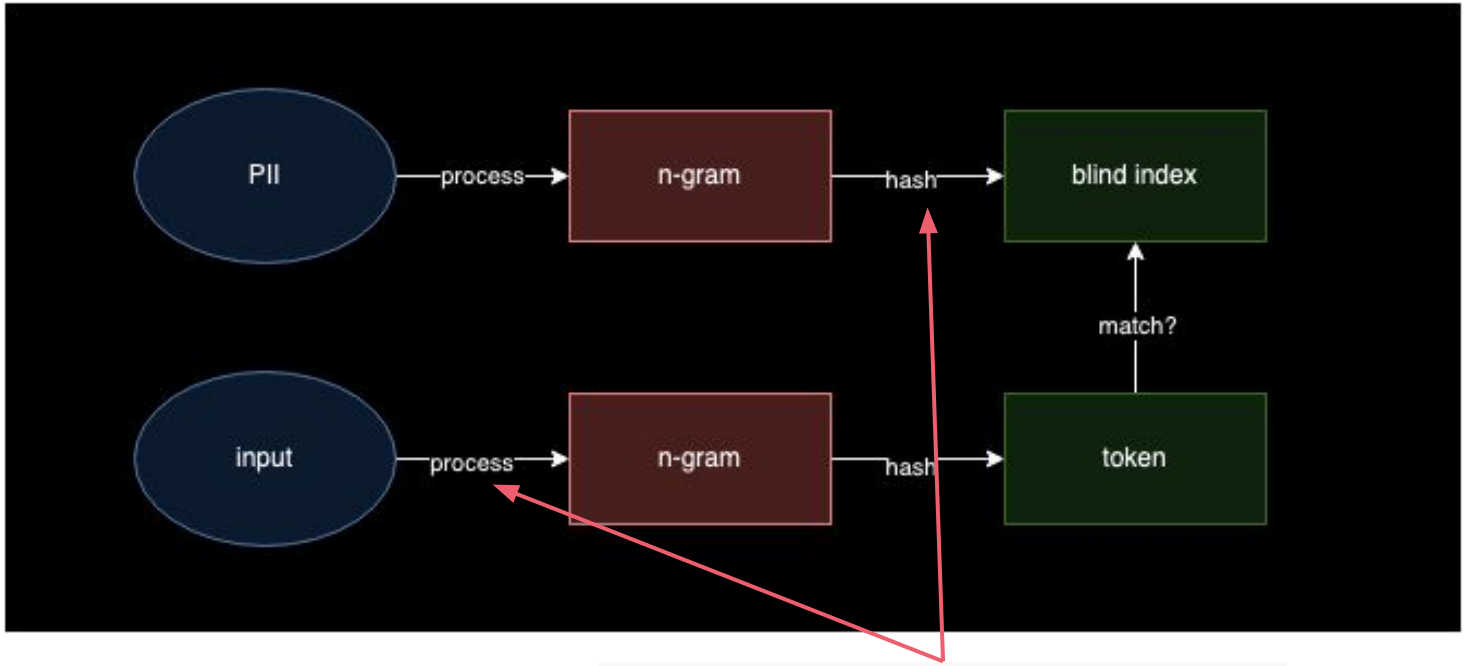
Admin

Engineer

Blind index

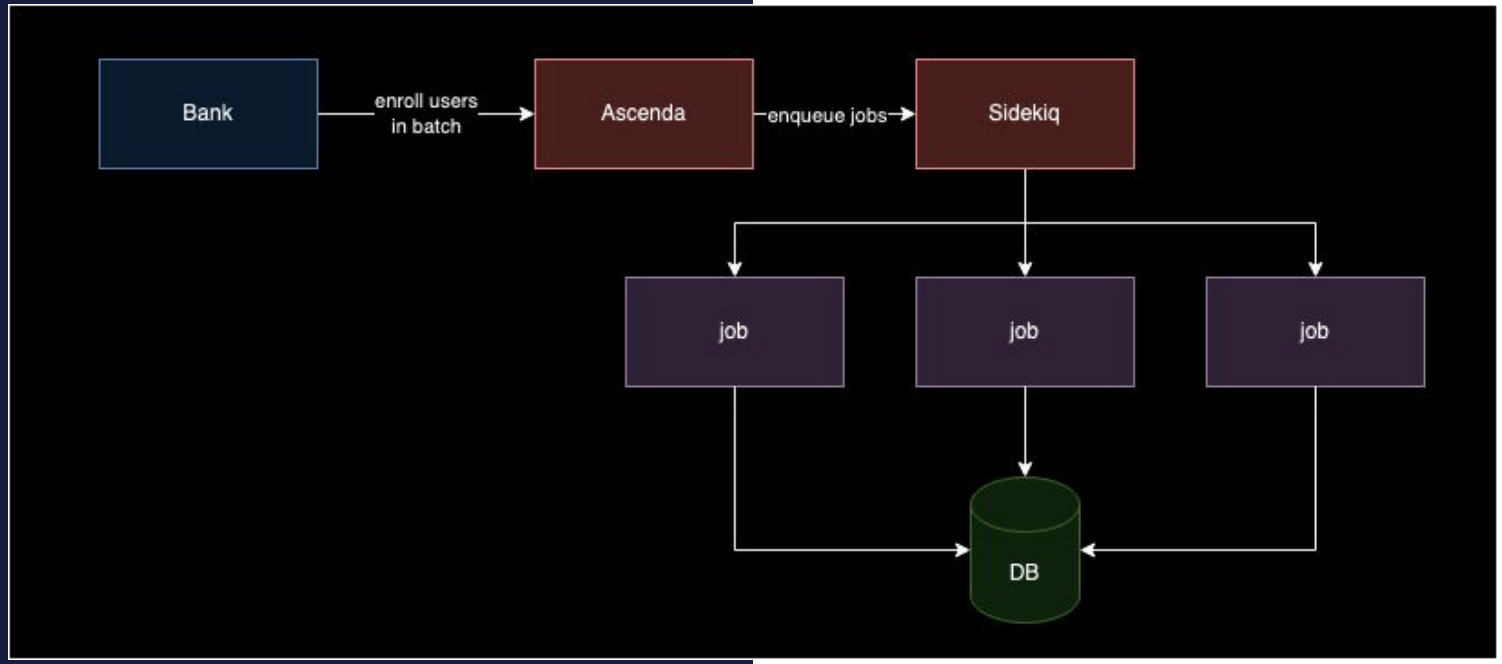

PII
process
n-gram
hash
blind index
match?
input
process
n-gram
hash
token

Blind index



CPU intensive actions!

Existing solution



Existing solution

Mature

Scalable (to certain extend)

Reliable

Under-utilize CPUs (sometimes)

Overhead

Existing solution

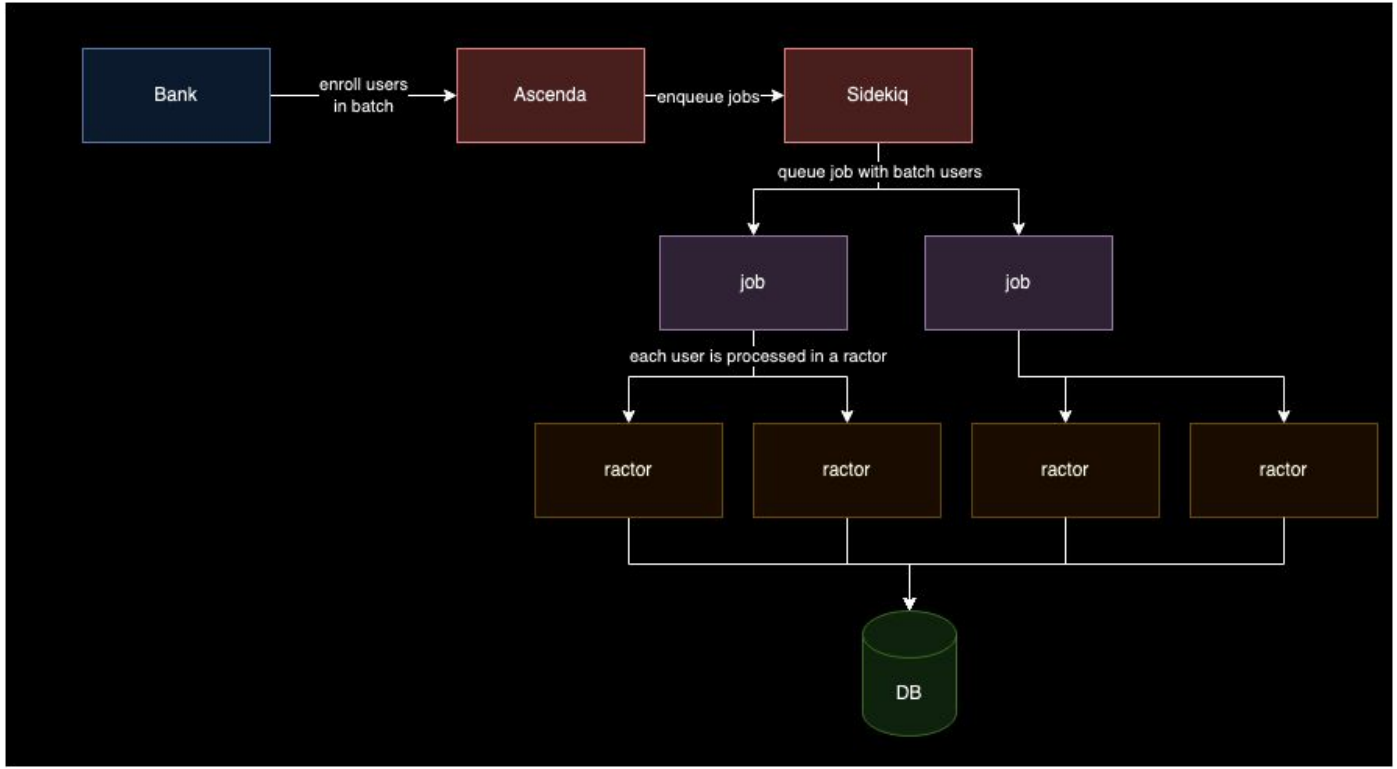
Generate token

```
class BlindIndexWorker < ApplicationWorker
  def perform(user_id)
    user_repo = UserRepository.new
    user = user_repo.find(user_id)

    tokens = user.to_h.slice(*PARTIAL_MATCH_FIELDS).compact.flat_map do |column, value|
      BlindIndexToken.generate(column, value)
    end

    return if tokens.empty?

    user_repo.update(user_id, blind_index_tokens: tokens)
  end
end

# usage
BlindIndexWorker.perform_bulk(users.map { |user| [user.id] })
```

Save to DB

queue jobs

Apply Ractor



Approach

1

No mutable actions

2

Avoid blocking operations

3

Data is sent via message

Apply Ractor

Generate token

```
ractors = users.map do |user|
  Ractor.new(user) do |user|
    tokens = user.to_h.slice(*PARTIAL_MATCH_FIELDS).compact.flat_map do |column, value|
      BlindIndexToken.generate(column, value)
    end

    [user, tokens]
  end
end
```

Save to DB

```
ractors.map(&:take).each do |user, tokens|
  next if tokens.empty?

  user_repo.update(user.id, blind_index_tokens: tokens)
end
```

Apply Ractor

Ractor usage

```
ractors = users.map do |user|
  Ractor.new(user) do |user|
    tokens = user.to_h.slice(*PARTIAL_MATCH_FIELDS).compact.flat_map do |column, value|
      BlindIndexToken.generate(column, value)
    end

    [user, tokens]
  end
end

ractors.map(&:take).each do |user, tokens|
  next if tokens.empty?

  user_repo.update(user.id, blind_index_tokens: tokens)
end
```

Result

```
Ractor::IsolationError
  can not access non-shareable objects in constant PARTIAL_MATCH_FIELDS by non-main Ractor
```

```
RuntimeError:
  defined with an un-shareable Proc in a different Ractor
```

Learning

Mutable constant is not shareable

Method with global state cannot be called

Update

```
ractors = users.map do |user|
  ractor = Ractor.new(user.to_h, PARTIAL_MATCH_FIELDS) do |attributes, fields|
    attributes.slice(*fields).compact.flat_map do |column, value|
      BlindIndexToken.generate(column, value)
    end
  end

  [user.id, ractor]
end.to_h

ractors.each do |id, ractor|
  tokens = ractor.take
  next if tokens.empty?

  user_repo.update(id, blind_index_tokens: tokens)
end
```

Result

```
Warming up --------------------------------------
         synchronous     1.000  i/100ms
        using ractor     1.000  i/100ms
        using thread     1.000  i/100ms
Calculating -------------------------------------
         synchronous      0.619  (± 0.0%) i/s -      4.000  in   6.466065s
        using ractor      1.820  (± 0.0%) i/s -     10.000  in   5.501745s
        using thread      0.632  (± 0.0%) i/s -      4.000  in   6.325387s

Comparison:
        using ractor:        1.8 i/s
        using thread:        0.6 i/s - 2.88x  slower
         synchronous:        0.6 i/s - 2.94x  slower
```

Improvement

Handle error and retry

Get response faster with Ractor.select

Save progress

Take away

1

Ractor is a great tool to achieve parallelism in Ruby

2

Ractor interface is easy to use and can already replace Thread simple use case

3

Ractor is still unstable and more tooling is necessary to support complex use case

Thank you for listening

sincepast

hieuk09