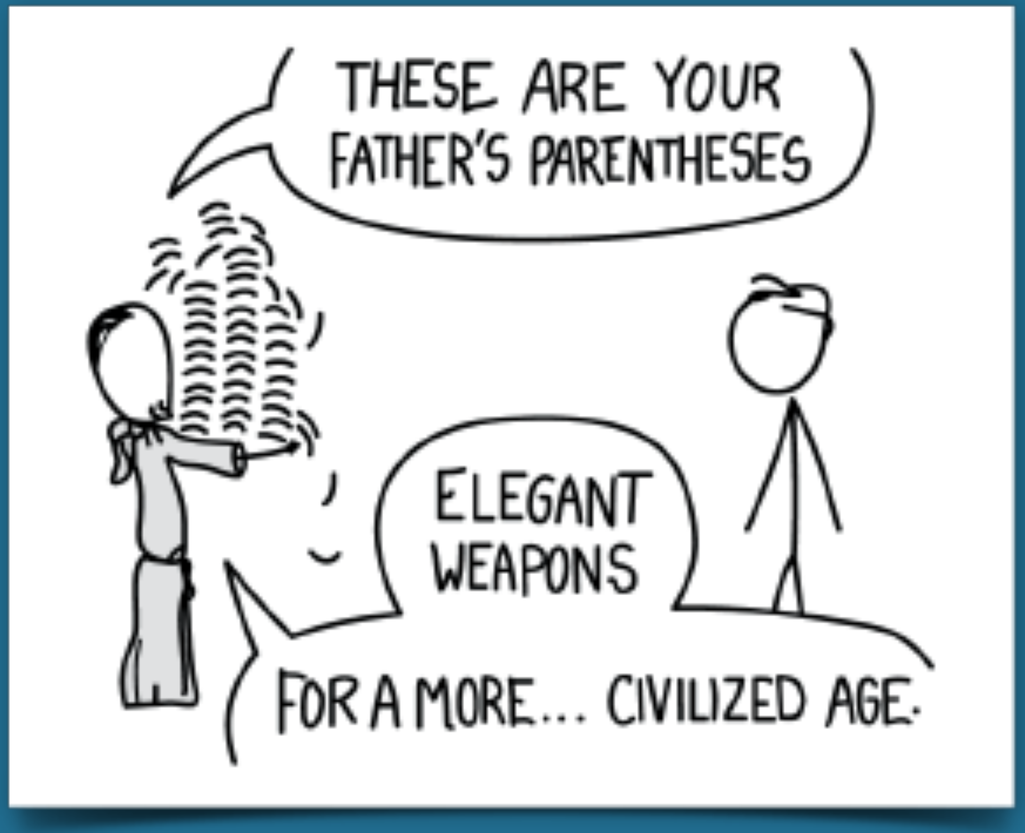
Interceptors: Reifying the Call Stack

Norbert Wójtowicz · @pithyless

Ruby · Clojure



Rack

```
class HelloWorld
  def call(env)
    [200,
     {"Content-Type" => “text/plain”},
     ["Hello world!"]]
  end
end
```

Rack Middleware - Enter

```
class AddGUID
  def call(env)
    env["tx_id"] = SecureRandom.uuid

    status, headers, body = @app.call(env)

    [status, headers, body]
  end
end
```

Rack Middleware - Leave

```
class NoCache
  def call(env)
    status, headers, body = @app.call(env)

    headers["Cache-Control"] = "no-cache"

    [status, headers, body]
  end
end
```

Rack Middleware - Around

```
class ProfileRequest
  def call(env)
    before = Time.now.to_f

    status, headers, body = @app.call(env)

    after = Time.now.to_f
    diff = after - before
    log_result(diff)

    [status, headers, body]
  end
end
```

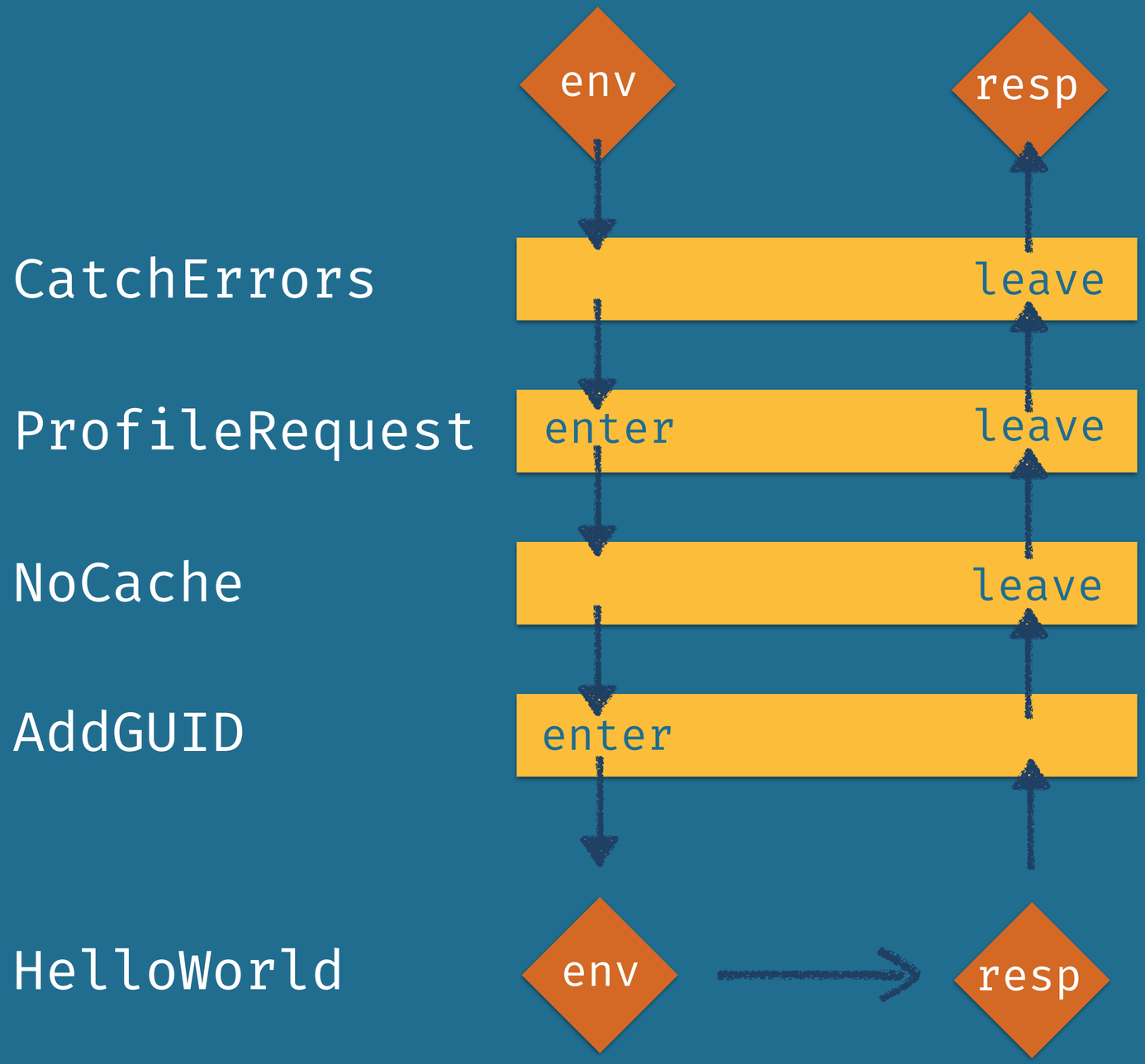
Rack Middleware - Error

```
class CatchErrors
  def call(env)
    @app.call(env)
  rescue StandardError => e
    log_error(e)
    [500, {...}, ["<html>Oh noes!</html>"]]
  end
end
```

Rack - Application

```
use CatchErrors
use ProfileRequest
use NoCache
use AddGUID

run HelloWorld
```

Call Stack


env
resp
CatchErrors
leave
ProfileRequest
enter
leave
NoCache
leave
AddGUID
enter
HelloWorld
env
resp

Call Stack - Stack Overflow

```
def fib(num)
  num < 3 ? 1 : fib(num - 2) + fib(num - 1)
end

fib(10) #=> 55

fib(100_000) #=> Stack Level Too Deep
```

Call Stack - Asynchronous

```
class AsyncHelloWorld
  def call(env)
    Thread.new
      sleep(5)
      response = [200, {...}, ["Hello world!"]]
      env['async.callback'].call(response)
    end

    throw :async
  end
end
```

Call Stack - Asynchronous Middleware

```
def call(env)
  response = @app.call env
  throw :async if @throw_on.include? response.first
  response
end

def call(env)
  response = @async_response
  catch(:async) { response = @app.call env }
  response
end
```

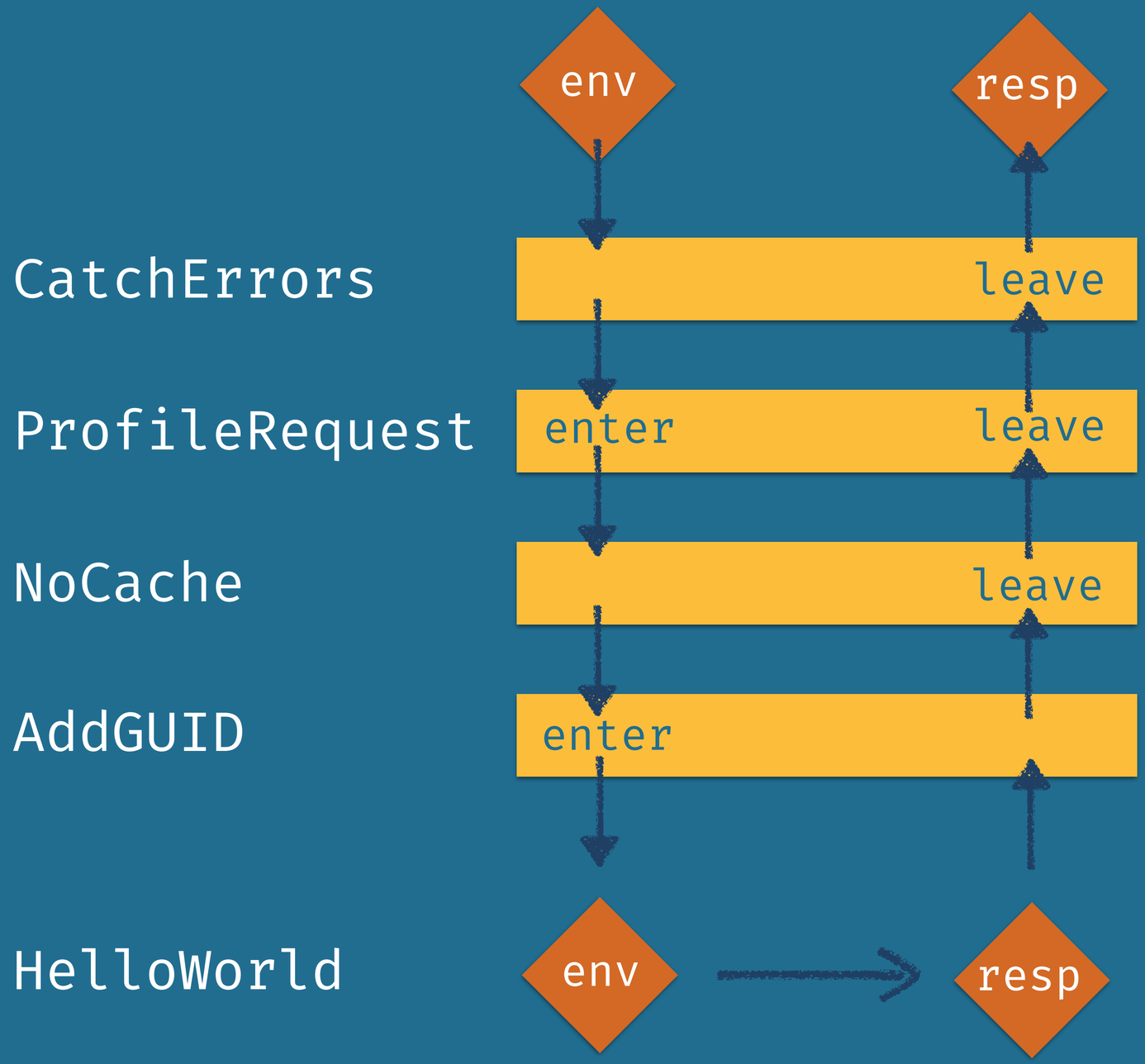
Call Stack - Streaming

```
def call(env)
  stream = env['rack.stream']
  stream.after_open do
    stream.chunk "Hello"
    stream.chunk "World"
    stream.close
  end
  [200, {'Content-Type' => 'text/plain'}, []]
end
```

Call Stack - Global Ordering

```
@use << proc { |app| middleware.new(app, *args, &block) }

...

app = @use.reverse.inject(app) { |a,e| e[a] }
```

Call Stack


env
resp
CatchErrors
leave
ProfileRequest
enter
leave
NoCache
leave
AddGUID
enter
HelloWorld
env
resp

Ring

```
(defn hello-world [request]
  {:status 200
   :headers {"Content-Type" "text/plain"}
   :body "Hello World!”})
```

Ring - Streaming

```
(defn hello-world [request]
  {:status 200
   :headers {"Content-Type" "text/plain"}
   :body "Hello World!”})

;; Body:
;; - String
;; - ISeq
;; - File
;; - InputStream
```

Ring - Asynchronous

```
(defn hello-world
  ;; Synchronous
  ([request]
    {:status 200
     :headers {"Content-Type" "text/plain"}
     :body "Hello World!"})

  ;; Asynchronous
  ([request respond raise]
    (respond (hello-world request))))
```

Ring - Application

```
(def app
  (-> handler
      (add-guid)
      (no-cache)
      (profile-request)
      (catch-errors)))
```

Reify

verb: make (something abstract)
more concrete or real

Pedestal - Interceptors

1. enter · leave · error

2. reified call stack

Interceptor - Handler

```
(defn hello-world [request]
  {:status 200
   :headers {"Content-Type" "text/plain"}
   :body "Hello World!”})

;; Body:
;; - String
;; - ISeq
;; - File
;; - InputStream
```

Interceptor - Enter

```
{:name ::attach-guid
 :enter (fn [context]
          (assoc context
                 ::guid
                 (java.util.UUID/randomUUID)))}
```

Interceptor - Leave

```
{:name ::disable-cache
 :leave (fn [context]
          (assoc-in context
                    [:response :headers "Cache-Control"]
                    "no-cache"))}
```

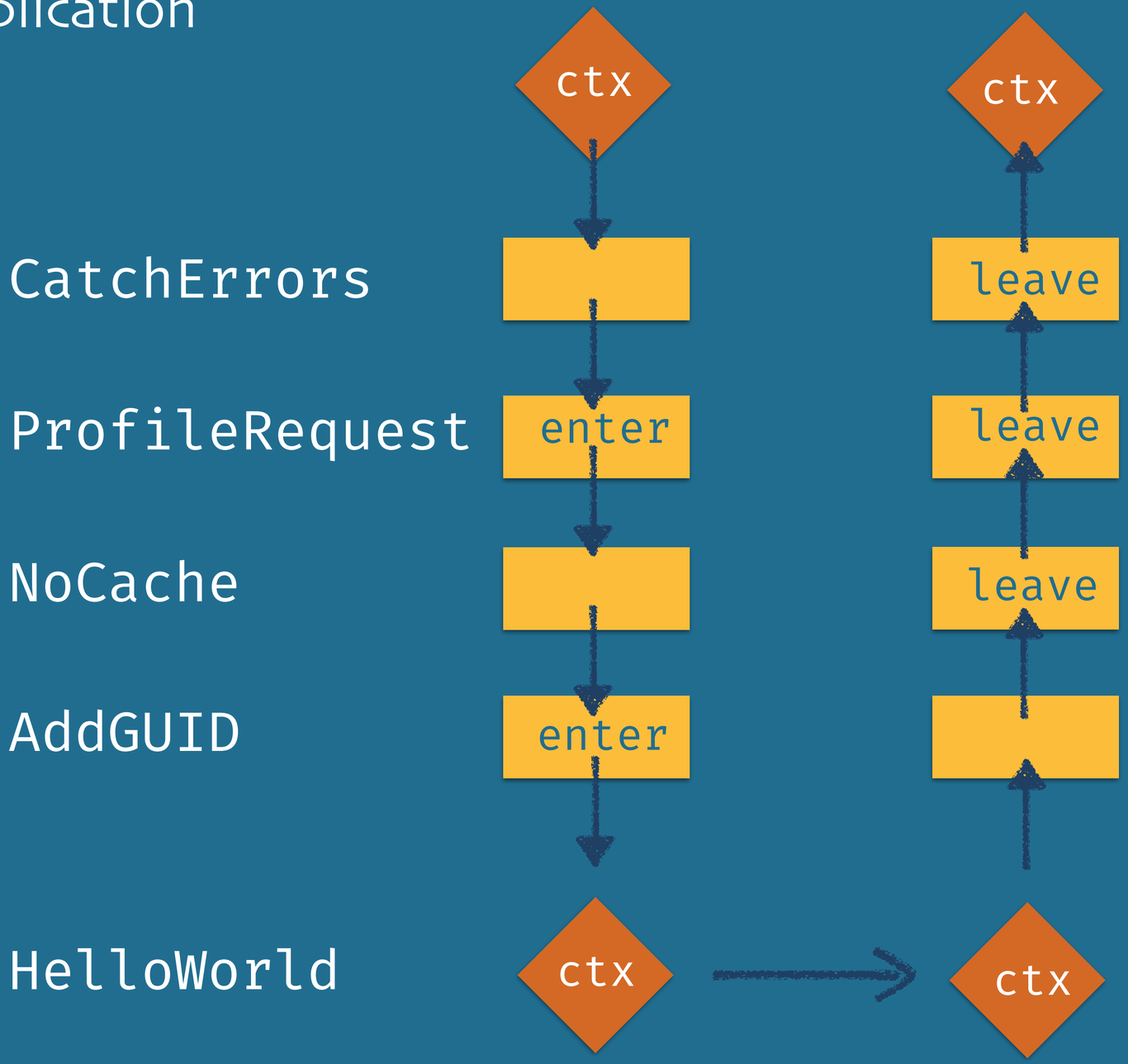
Interceptor - Around

```
{:name ::profile-request
 :enter (fn [context]
          (let [before (System/currentTimeMillis)]
            (assoc context ::start-time before)))
 :leave (fn [context]
          (let [before (get context ::start-time)
                after  (System/currentTimeMillis)
                diff   (- after before)]
            (log diff)
            context))}
```

Interceptor - Errors

```
{:name ::handle-error
 :enter (fn [context] ...)
 :leave (fn [context] ...)
 :error (fn [context ex]
          ;; 1. Handle error
          context

          ;; 2. Cannot handle error
          (assoc context ::interceptor/error ex)

          ;; 3. Should have handled error, but...
          (throw (ex-info "Oh noes!" {:extra :data}))))}
```

Interceptors - Application



Interceptor - Deferred Calls

```
{:name ::third-party-auth
 :enter (fn [context]
          (go (assoc context
                     :account
                     (call-auth-system context))))}
```

Interceptor - Parameterized

```
(defn auth-system [env]
  (if (dev? env)
    {:name ::fake-auth
     :enter (fn [context]
              (assoc context
                     :account
                     (fake-account context)))}
    {:name ::third-party-auth
     :enter (fn [context]
              (go (assoc context
                         :account
                         (call-auth-system context)))))}))
```

Interceptor - Stateful

```
(defn attach-database [uri]
  (let [db (db-pool uri)]
    {:name ::attach-database
     :enter (fn [context]
              (assoc context ::db db))}))
```

Interceptor - Custom Ordering

```
(defroutes routes
  ["/api" [(requires rate-limiter :account)]
    ["/slack" [(provides slack-auth :account)] ...]
    ["/hipchat" [(provides hipchat-auth :account)] ...]])
```

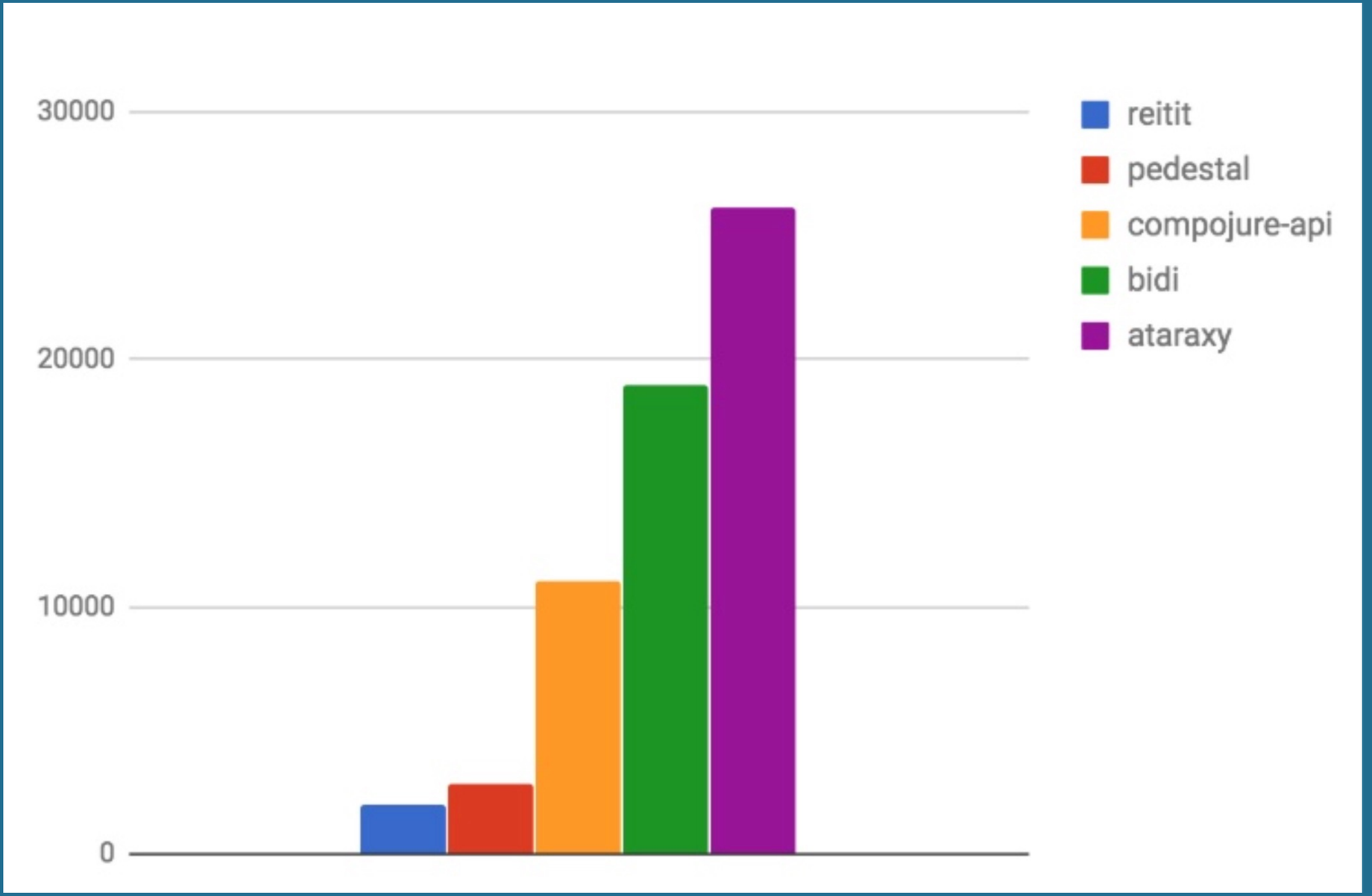
Interceptor - Dynamic Ordering

```
(keys context)

[...
 :io.pedestal.interceptor.chain/queue
 :io.pedestal.interceptor.chain/stack
 ...]
```

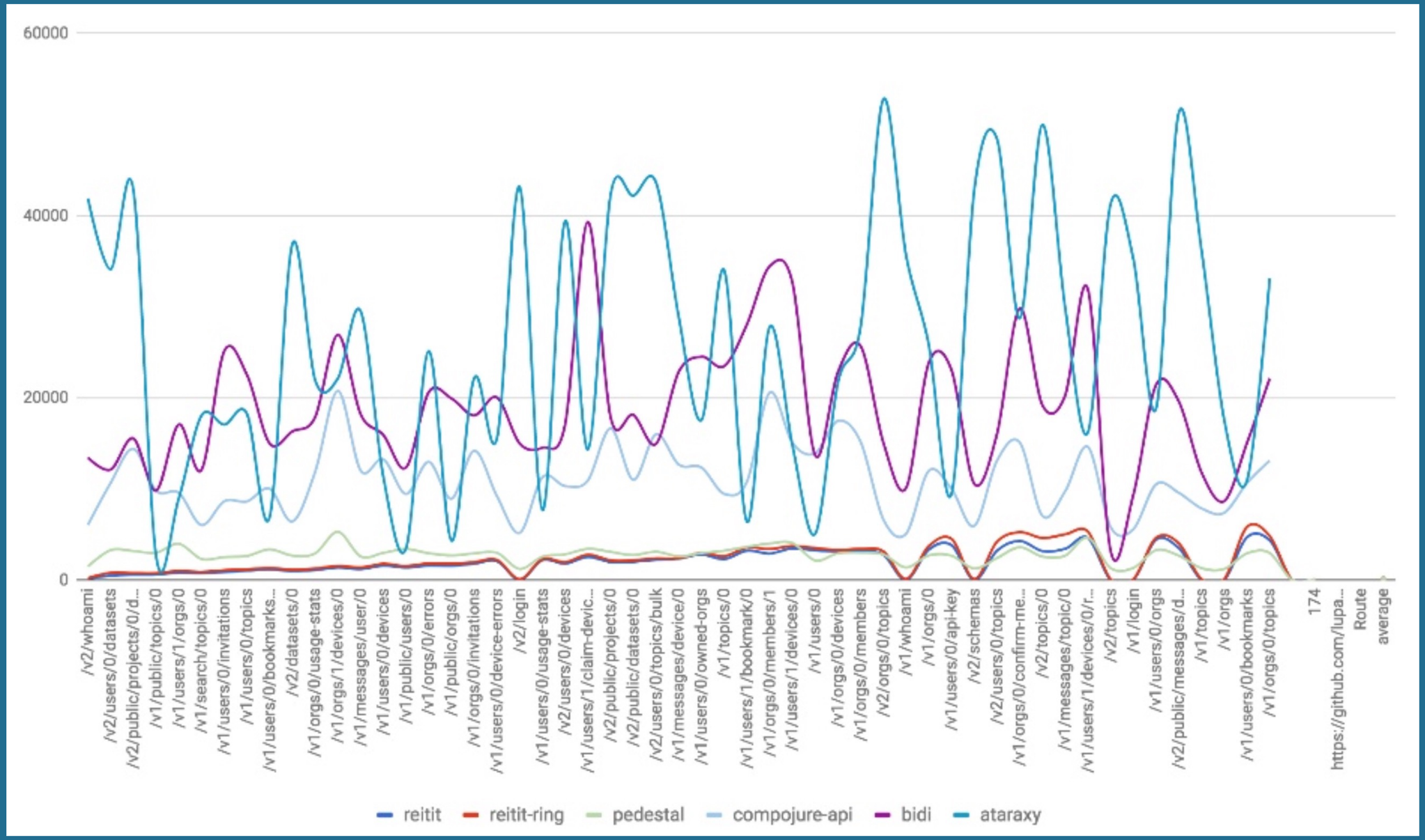
Interceptor - Dynamic Routing

```
GET  /docs
auth → params → read-db → handler → to-json

GET  /docs/secret.pdf
auth → File (streaming)

POST /docs
auth → params → upload → process file → write-db → SSE
```

Interceptor - Performance



Interceptor - Performance



Reify all the things!

@pithyless