Using and scaling Rack and Rack-based middleware

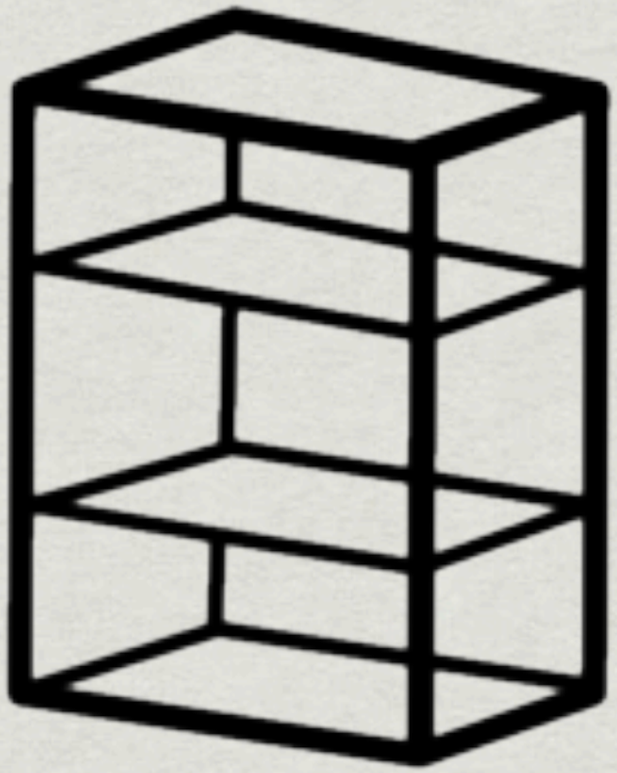
Alona Mekhovova

- Ruby/Rails developer & co-founder @ RubyGarage
- Organizer & coach @ RailsGirls
- Ruby on Rails courses leading
- Involved in growing Ruby Community in Dnipro
- Monthly Rails MeetUps organizer

skype: alony_
alony@rubygarage.org

Overview

- Why do we need Rack?
- Simple Rack app & own middleware
- Available middleware overview
- Plugging middleware into Rails
- What’s next?



rack

powers web applications

Lots of Web-servers

...and frameworks

the http protocol

- request => response
- stateless

request:

- Request method, URI, protocol version
- Request headers
- Request body

response:

- Protocol version, status code, its desc
- Response headers
- Response body

http
from a bird's eye view

Request structure

```
{"HTTP_USER_AGENT"=>"curl/7.12.2 ..."
 "REMOTE_HOST"=>"127.0.0.1",
 "PATH_INFO"=>"/",
 "HTTP_HOST"=>"ruby-lang.org",
 "SERVER_PROTOCOL"=>"HTTP/1.1",
 "SCRIPT_NAME"=>"",
 "REQUEST_PATH"=>"/",
"REMOTE_ADDR"=>"127.0.0.1",
"HTTP_VERSION"=>"HTTP/1.1",
"REQUEST_URI"=>"http://ruby-lang.org/",
"SERVER_PORT"=>"80",
"HTTP_PRAGMA"=>"no-cache",
"QUERY_STRING"=>"",
"GATEWAY_INTERFACE"=>"CGI/1.1",
"HTTP_ACCEPT"=>"*/*",
"REQUEST_METHOD"=>"GET"}
```

- Classically, a CGI environment
- Most frameworks already use smth like that, most developers know the fields
- Let’s keep it

Response structure

Status

```
HTTP/1.1 302 Found
Date: Sat, 27 Oct 2007 10:07:53 GMT
Server: Apache/2.0.54 (Debian GNU/Linux)
mod_ssl/2.0.54 OpenSSL0.9.7e
```

Headers

```
Location: http://www.ruby-lang.org/
Content-Length: 209
Content-Type: text/html; charset=iso-8859-1
```

Body

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a
href="http://www.ruby-lang.org/">here</a>
</p>
</body></html>
```

Response in Ruby

Status
Integer

```
HTTP/1.1 302 Found
```

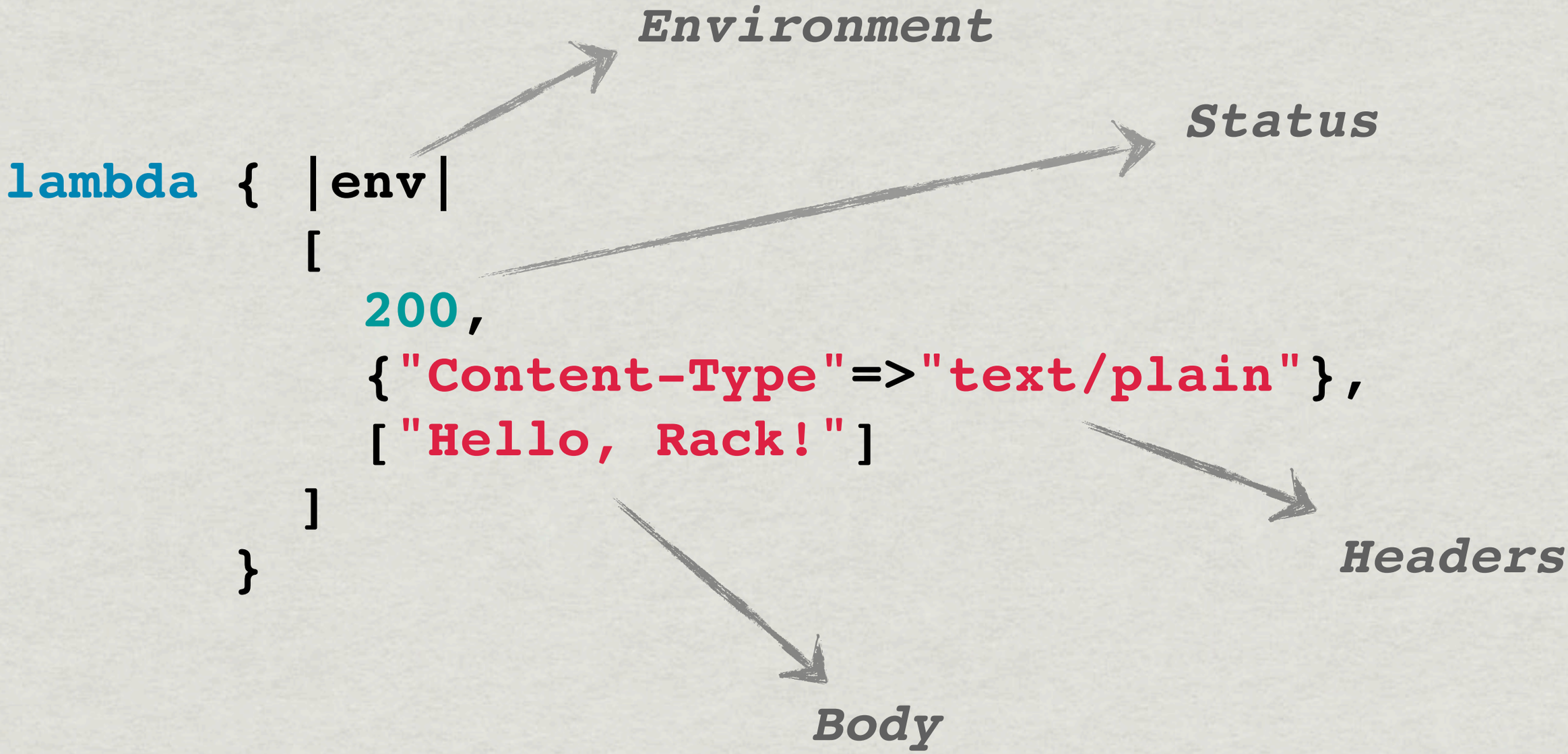
Headers
Hash

```
Date: Sat, 27 Oct 2007 10:07:53 GMT
Server: Apache/2.0.54 (Debian GNU/Linux)
mod_ssl/2.0.54 OpenSSL0.9.7e

Location: http://www.ruby-lang.org/
Content-Length: 209
Content-Type: text/html; charset=iso-8859-1
```

Body
Array

```
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>302 Found</title>
</head><body>
<h1>Found</h1>
<p>The document has moved <a
href="http://www.ruby-lang.org/">here</a>
</p>
</body></html>
```

Response in Ruby



Summarizing

- The Rack app gets called with the CGI environment...
- ...and returns an Array of status, header & body

...and again

HTTP SERVER

YOUR FANCY APP

Simplest Rack app

```
run Proc.new { |env|
          [
            200,
            {"Content-Type"=>"text/plain"},
            ["Hello, Rack!"]
          ]
        }
```

rackup

```
# config.ru

class Awesome
  def call(env)
    [200, {'Content-Type' => 'text/plain'}, 'AWESOME.']
  end
end

run Awesome.new


# console

rackup config.ru
```

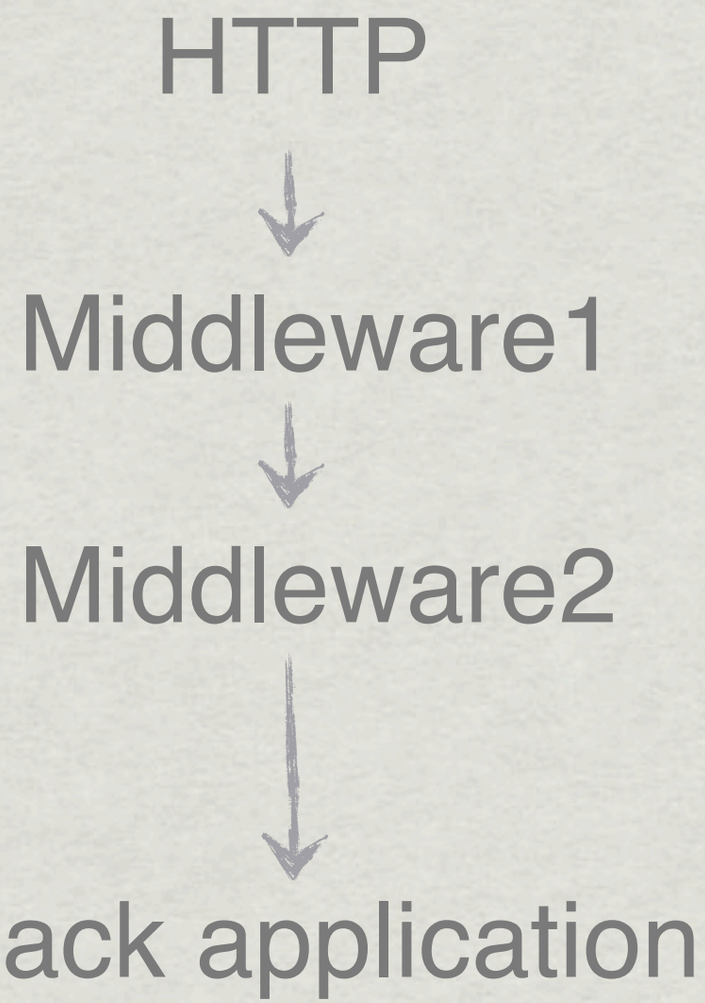
Rack::Builder

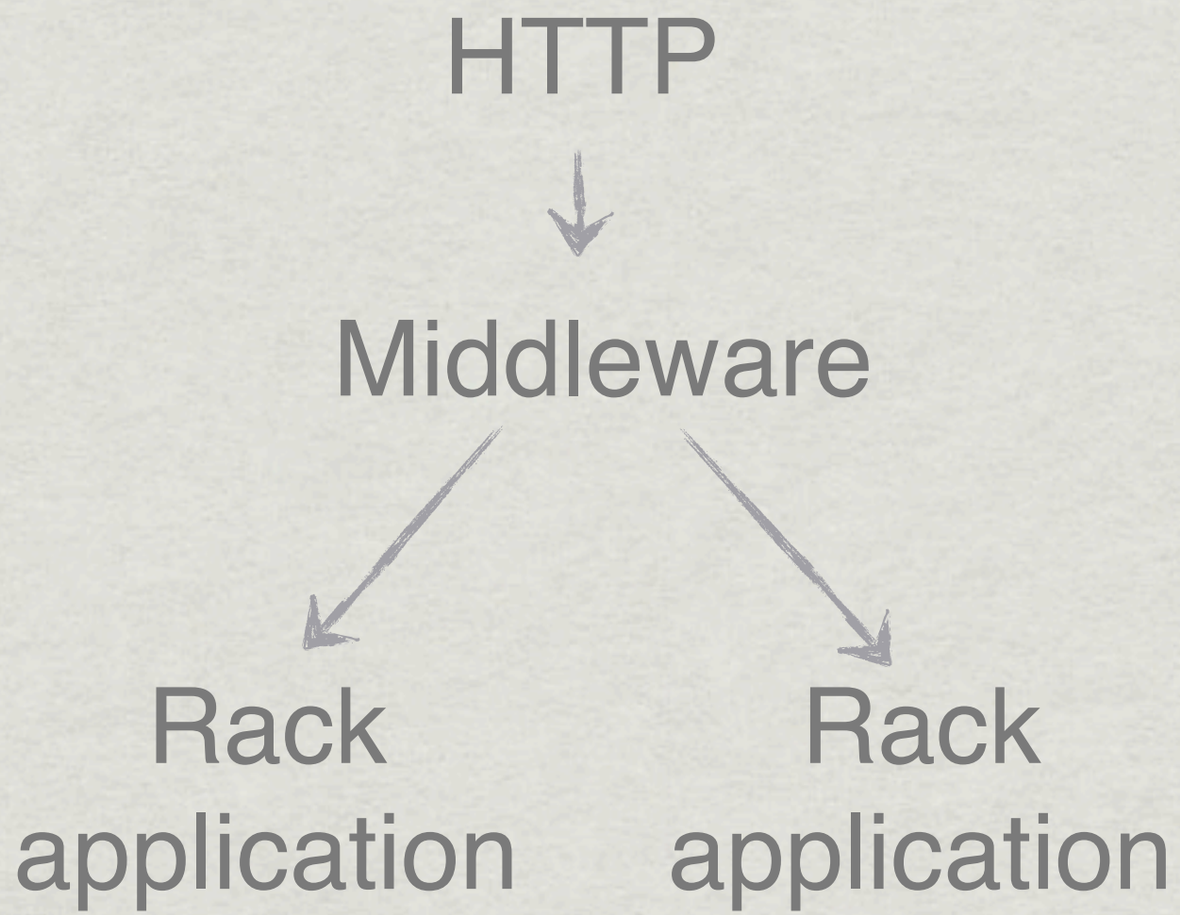
```
app = Rack::Builder.app do
  map '/awesome' do
    run Awesome
  end

  map '/' do
    run Proc{ 404,
     {"Content-Type" => "text/html"},
     ['awesome requests only'] }
  end
end

run app
```

Rack::Builder

```
Rack::Builder.new do
  use Rack::CommonLogger
  use Rack::ShowExceptions
  use Rack::ShowStatus
  use Rack::Lint
  run MyRackApp.new
end
```

Middleware


HTTP
Middleware1
Middleware2
Rack application


HTTP
Middleware
Rack application
Rack application

Home-made middleware

```
class JSMinifier

  def initialize(app, path)
    @app, @root = app, File.expand_path(path)
  end

  def call(env)
    path = File.join(@root, Utils.unescape(env["PATH_INFO"]))
    return @app.call(env) unless path.match(/.*\/(\w+\.js)$/)

    if !File.readable?(path) or env["PATH_INFO"].include?("..")
      return [403, {"Content-Type" => "text/plain"}, ["Forbidden\n"]]
    end

    return [ 200,
             { "Content-Type" => "text/javascript" },
             [JSMin.minify( File.new( path, "r" ) )]
           ]
  end
end
```

Home-made middleware

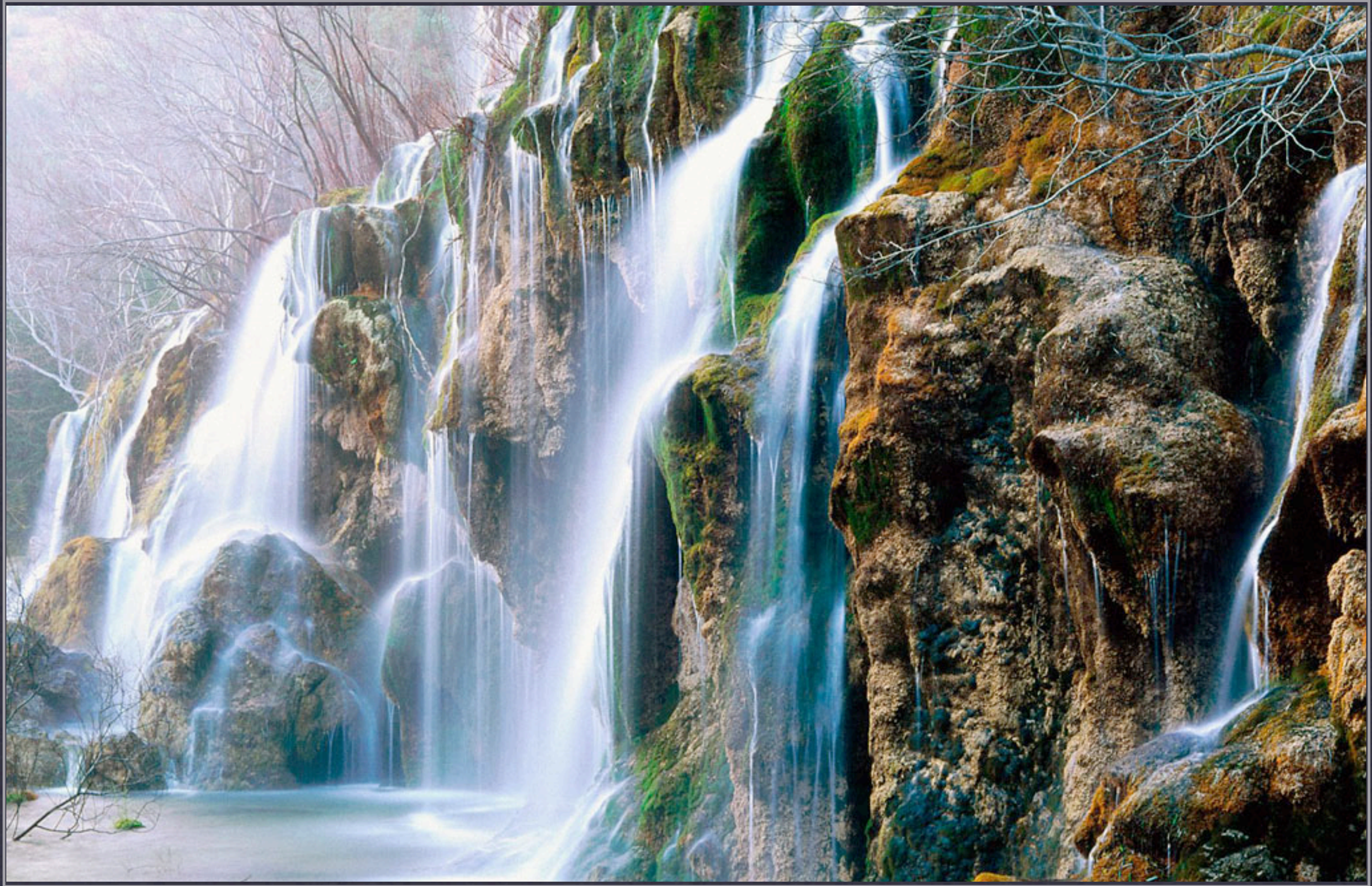
```
class JSMinifier

  def initialize(app, path)
    @app, @root = app, File.expand_path(path)
  end

  def call(env)
    path = File.join(@root, Utils.unescape(env["PATH_INFO"]))
    return @app.call(env) unless path.match(/.*\/(\w+\.js)$/)

    if !File.readable?(path) or env["PATH_INFO"].include?("..")
      return [403, {"Content-Type" => "text/plain"}, ["Forbidden\n"]]
    end

    return [ 200,
             { "Content-Type" => "text/javascript" },
             [JSMin.minify( File.new( path, "r" ) )]
           ]
  end
end
```

Home-made middleware

```
class JSMinifier

  def initialize(app, path)
    @app, @root = app, File.expand_path(path)
  end

  def call(env)
    path = File.join(@root, Utils.unescape(env["PATH_INFO"]))
    return @app.call(env) unless path.match(/.*\/(\w+\.js)$/)

    if !File.readable?(path) or env["PATH_INFO"].include?("..")
      return [403, {"Content-Type" => "text/plain"}, ["Forbidden\n"]]
    end

    return [ 200,
             { "Content-Type" => "text/javascript" },
             [JSMin.minify( File.new( path, "r" ) )]
           ]
  end
end
```

What we already have

Rack::Bug



Rack::Cascade

Rack::Cache

Rack::GeoIP

Rack::Callback

Rack::MailExceptions

Rack::Honeypot

```
env['warden'].authenticated?
env['warden'].authenticate!(:password)
env['warden'].user
```

```
Warden::Strategies.add(:password) do

  def valid?
    params[:username] || params[:password]
  end

  def authenticate!
    u = User.authenticate(params[:username],
params[:password])
    u.nil? ? fail!("Could not log in") : success!
(u)
  end
end
```

Warden

... and lots of others

- Rack::Static
- Rack::CSSHTTPRequest
- Rack::noIE
- Rack::GoogleAnalytics
- Rack::Profiler
- Rack::Maintenance
- Rack::Lock
- Rack::Spellcheck
- Rack::Log
- Rack::Pack
- Rack::Validate

https://github.com/rack/rack/wiki/List-of-Middleware

http://coderack.org/middlewares

Plug it into Rails stack

```
$ rake middleware

use ActionDispatch::Static
use Rack::Lock
use ActiveSupport::Cache::Strategy::LocalCache
use Rack::Runtime
use Rails::Rack::Logger
use ActionDispatch::ShowExceptions
use ActionDispatch::DebugExceptions
use ActionDispatch::RemoteIp
use Rack::Sendfile
use ActionDispatch::Callbacks
use ActiveRecord::ConnectionAdapters::ConnectionManagement
use ActiveRecord::QueryCache
    ...
use Rack::MethodOverride
use ActionDispatch::Head
use ActionDispatch::BestStandardsSupport
run MyApp::Application.routes
```

Configure middleware stack

```
Rails::Initializer.run do |config|
  config.middleware.use Rack::MailExceptions
  config.middleware.delete Rack::Lock
  config.middleware.insert_after Rack::Sendfile, Rack::Static
  config.middleware.insert_before Rack::Head, Ben::Asplode
  config.middleware.swap ActionDispatch::Callbacks,
ActiveRecord::QueryCache
end
```

```
config.middleware.is_a? Array    # => true
```

...or replace it

```ruby
# config/initializers/stack.rb
ActionController::Dispatcher.middleware =
        ActionController::MiddlewareStack.new do |m|
          m.use ActionController::Failsafe
          m.use ActiveRecord::QueryCache
          m.use Rack::Head
end
```

```ruby
# console
$ rake middleware

use ActionController::Failsafe
use ActiveRecord::QueryCache
use Rack::Head
run ActionController::Dispatcher.new
```

The future

- Rails 4 will have an API only middleware stack configuration
- RocketPants is an interesting alternative right now (github.com/filtersquad/rocket_pants)

Many thanks!