Introducing Rack

Christian Neukirchen
Editor in Chief of Anarchaia

rack

powers web applications

A Google Image Search for *"ruby rack"* reveals:

Interested in **Ruby** on Rails?
1180 x 1494 - 1364k - jpg
www.sitepoint.com

Overview

- What is Rack?
- Why do we need Rack?
- The design of Rack
- The Rack distribution
- Coset: a RESTful Rack framework
- Real World Rack

What is Rack?

- Rack is a specification (and implementation) of a minimal abstract Ruby API that models HTTP

Why do we need Rack?

- Developing a Ruby web framework is *not* hard…
- …but it's lots of *repetitive, boring* work:
  - *Again, write* interfaces to all the servers!
  - *Again, write* decoding code or copy cgi.rb

The big picture

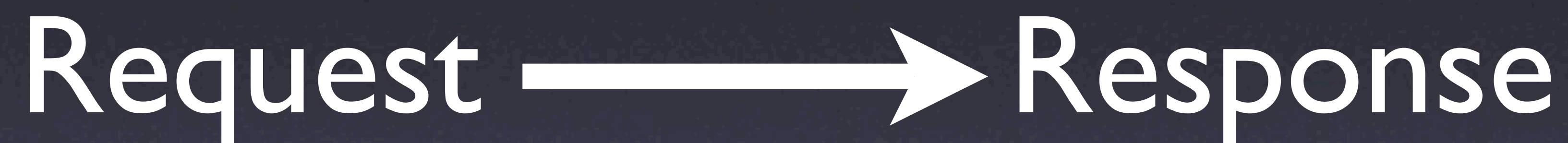
- Let’s make the simplest possible API that represents a generic web application
- Write the HTTP interfacing code OAOO*

* once and only once

Designing Rack

- How to do that?
- “type-based design” *(*shock*, *shudder*)*

HTTP from a Bird's-eye view



What's a request?

```
{"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 something like it, most developer know the fields
- Let's keep that

And what's a response?

- CGI uses stdout. Ugh!
- Let's look at one

Status

```
HTTP/1.1 302 Found
```

Headers

```
Date: Sat, 27 Oct 2007 10:07:53 GMT
Server: Apache/2.0.54 (Debian GNU/Linux)
  mod_ssl/2.0.54 OpenSS0.9.7e
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>
```

The Response in Ruby

- Duck-types to the rescue, an Array of:
  - Status: `to_i`
  - Headers: `each { |key, value| }`
  - Body: `each { |chunk| }`

How to *call* it

- Obviously, `#call`
- This also allows using *lambdas* as web apps!

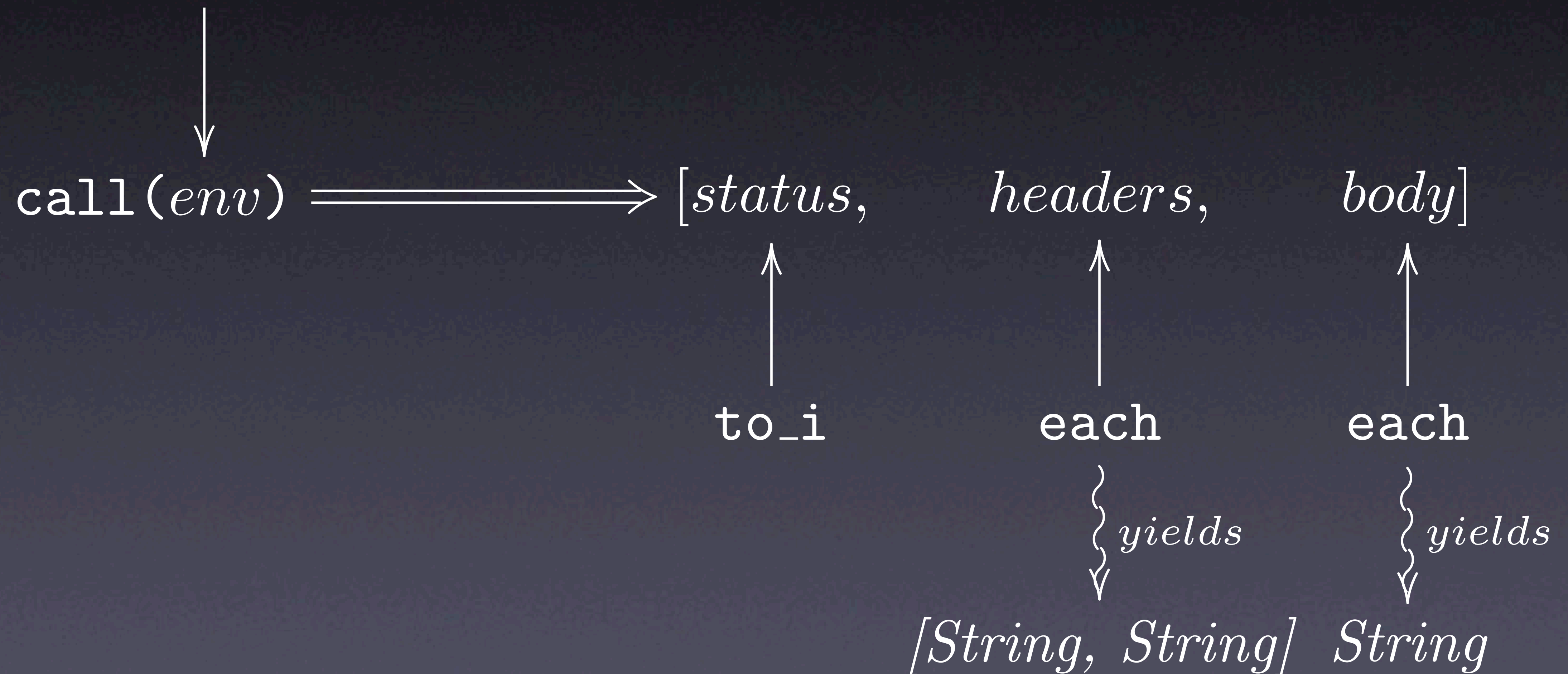
To summarize:

- The Rack app gets `call`ed with the CGI `env`ironment…
- …and returns an Array of `status`, `headers` and `body`.

Hello, world!

```
lambda { |env|
  [200,                                    # Status
   {"Content-Type"=>"text/plain"},  # Headers
   ["Hello, world!"]]                      # Body
}
```

Rack at a glance



The Rack Distribution, I

- The Rack specification
  - including Rack::Lint, which checks the apps

```
## * The <tt>CONTENT_LENGTH</tt>,
##   if given, must consist of digits only.
assert("Invalid CONTENT_LENGTH: " +
       "#{env["CONTENT_LENGTH"]}") {
  !env.include?("CONTENT_LENGTH") ||
    env["CONTENT_LENGTH"] =~ /\A\d+\z/
}
```

The Rack Distribution, II

- Handlers
  - CGI
  - FastCGI
  - WEBrick
  - Mongrel (also Swiftcore's *evented_mongrel*)
  - LiteSpeed (trunk only)

```
Rack::Handler::Mongrel.run app, :Port => 80
```

The Rack Distribution, III

- Included adapters
  - Camping
- 3rd party adapters
  - Ramaze
  - Merb
  - Rails (via Fuzed)

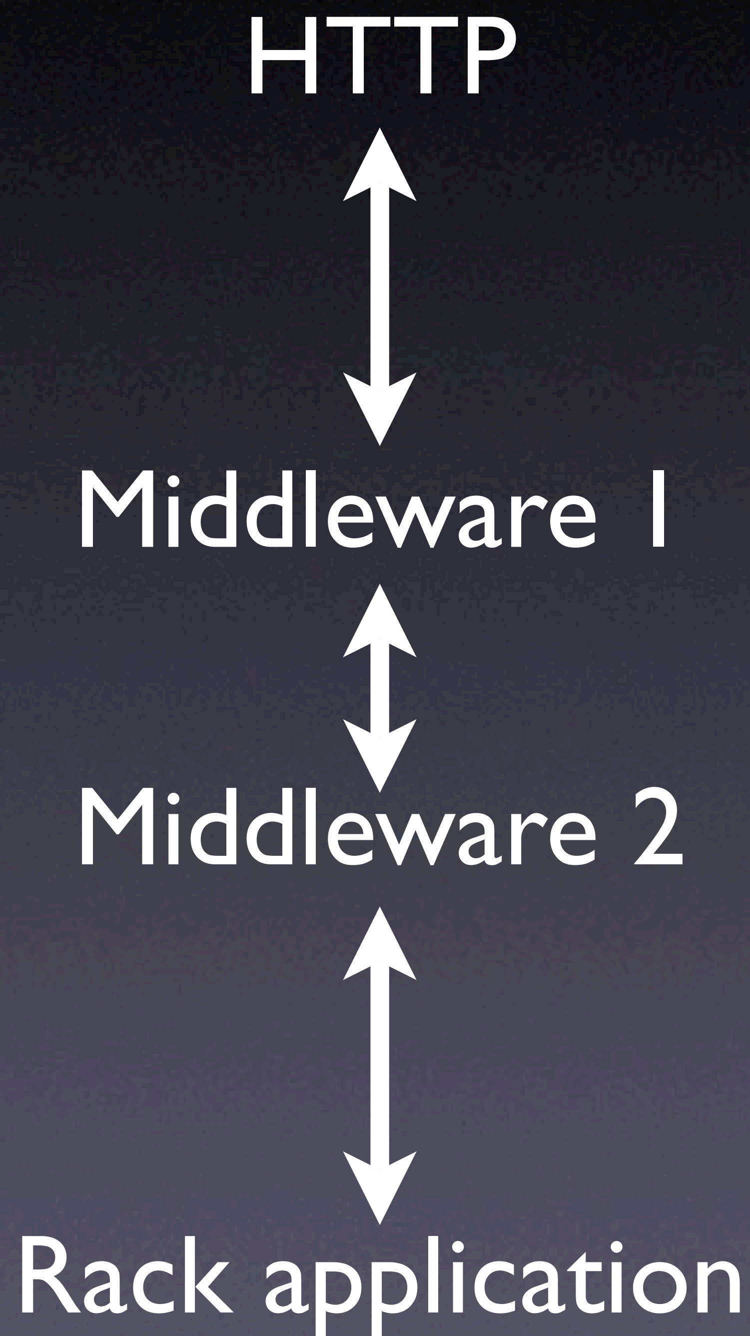
and more: Maveric, Sinatra, …

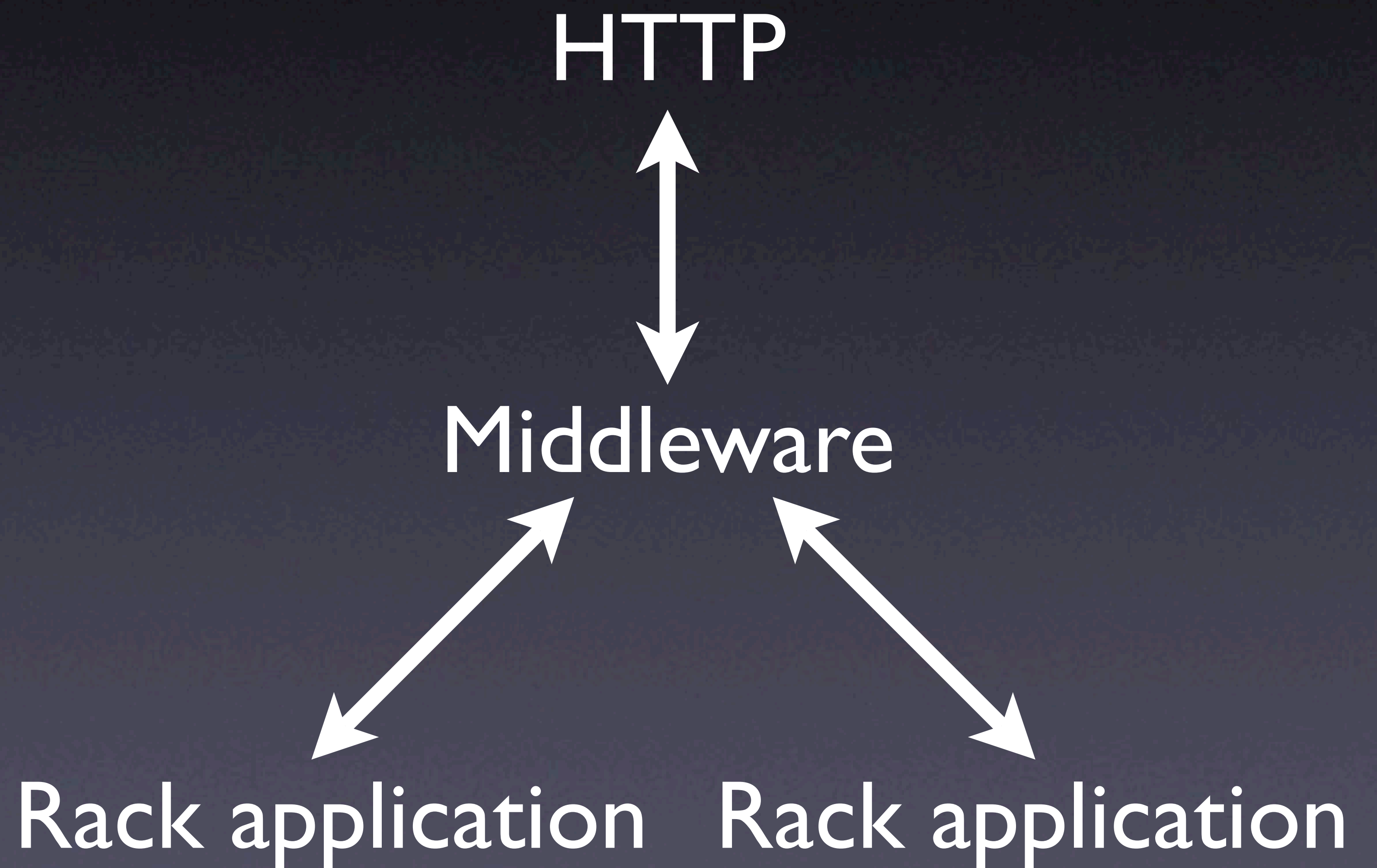
- All adapters are almost trivial

The Rack Distribution, IV

- Lots of *middleware*
  - Utilities that combine, compose, aggregate or modify Rack applications
  - Middleware is stackable, it’s just a Rack application itself
  - Since Rack applications are just Ruby objects, they are easy to write

Middleware





A few selected modules

- Rack::Cascade: Try a request with several apps, and return the first non-404 result
- Rack::CommonLogger: Make an Apache-like logfile
- Rack::Lint: Ensure the app obeys the specification

Rack::Request & Rack::Response

- They are your friends if you want to write Rack applications directly

```
def call(env)
  req = Rack::Request.new(env)
  res = Rack::Response.new

  if req.get?
    res.write "Hello #{req.GET["name"]}"
  elsif req.post?
    file = req.POST["file"]
    FileUtils.cp file[:tempfile].path,
      File.join(UPLOADS, file[:filename]
    res.write "Uploaded."
  end

  res.finish
end
```

Rack::ShowExceptions

RuntimeError at /

Lobster crashed

Ruby `./lib/rack/lobster.rb`: in `call`, line 40
Web `GET localhost/`

Jump to:

GET | POST | Cookies | ENV

Traceback (innermost first)

`./lib/rack/lobster.rb`: in `call`

```
33.       req = Request.new(env)
34.       if req.GET["flip"] == "left"
35.         lobster = LobsterString.split("\n").
36.           map { |line| line.ljust(42).reverse }.
37.           join("\n")
38.         href = "?flip=right"
39.       elsif req.GET["flip"] == "crash"
40.         raise "Lobster crashed"
41.       else
42.         lobster = LobsterString
43.         href = "?flip=left"
44.       end
45.
46.       Response.new.finish do |res|
47.         res.write "<title>Lobstericious!</title>"
```

`./lib/rack/showexceptions.rb`: in `call`

Rack::URLMap

```
Rack::URLMap.new "/one" => app1,
                 "/two" => app2,
                 "/one/foo" => app3
```

Also can do virtual hosts:

```
Rack::URLMap.new \
  "http://one.example.org/" => app1,
  "http://two.example.org/" => app2,
  "https://example.org/secure" => secureapp
```

Testing with Rack::MockRequest

```
require "rack"
require "test/spec"

describe "The sample application 3 slides ago" do
  it "should reply with a welcome on GET" do

    req = Rack::MockRequest.new(myapp)
    res = req.get("/?name=Euruko")

    res.should.be.ok
    res.should.match /Hello, Euruko/

  end
end
```

Rack configuration

- If you want to use many utilities, don't do

```
app = Rack::CommonLogger.new(
        Rack::ShowExceptions.new(
          Rack::ShowStatus.new(
            Rack::Lint.new(MyRackApp.new))))
```

- Instead, write

```
app = MyRackApp.new
app = Rack::Lint.new(app)
app = Rack::ShowStatus.new(app)
app = Rack::ShowExceptions.new(app)
app = Rack::CommonLogger.new(app)
```

Rack::Builder

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

rackup

```
#!/usr/bin/env rackup

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

Override basic configuration

```
$ myrackapp.ru -s mongrel -p 8080
```

Rackup autodetects CGI or FastCGI environments

Coset

- Finally, a small RESTful framework I wrote for my own
- Supports URI templates and direct HTTP method support
- As well as dealing with multiple content types

Coset example

```
class TimeServer < Coset
  GET "/time{EXT}" do
    now = Time.now
    wants "text/html" do
      res.write "<title>Current time</title>\n"
      res.write "It's now #{now}.\n"
    end
    wants "text/plain" do
      res["Content-Type"] = "text/plain"
      res.write now.to_s + "\n"
    end
    wants "application/json" do
      res["Content-Type"] = "application/json"
      res.write "{\"current_time\": \"#{now}\"}\n"
    end
  end
end
```

```
$ curl -i localhost:3333/time
HTTP/1.1 200 OK
Content-Type: text/html

<title>Current time</title>
It's now Sun Nov 04 12:03:18 CET 2007.
```

```
$ curl -i localhost:3333/time.txt
HTTP/1.1 200 OK
Content-Type: text/plain

Sun Nov 04 12:03:56 CET 2007
```

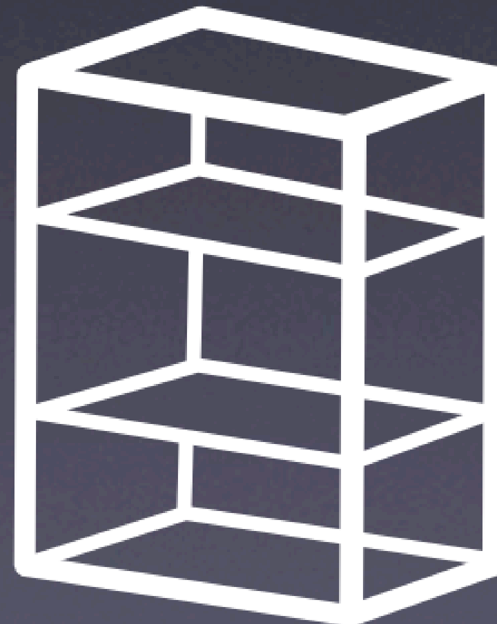
```
$ curl -i -H "Accept: application/json" \
                      localhost:3333/time
HTTP/1.1 200 OK
Content-Type: application/json

{"current_time": "Sun Nov 04 12:05:14 CET 2007"}
```

Real World Rack

- Personifi uses a custom Rack application to serve 30 billion(!) requests a month
- Rack allows fast development due to its lean interface and convenient APIs

Summary

- An abstract interface on top of HTTP allows for
  - code reuse
  - easier testing
  - new ways of combining code/applications
- Rack is easy to support
  - …and quickly pays off
- Support Rack!

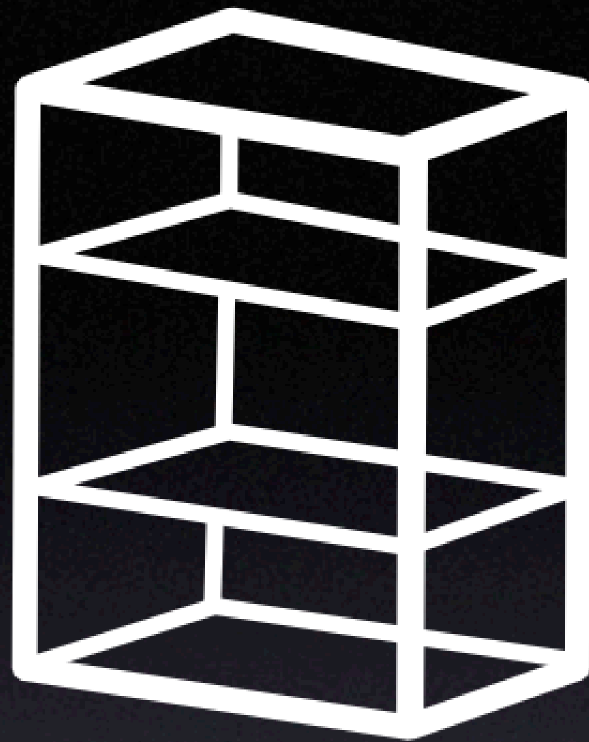


rack

powers web applications

Thanks for your attention!

- Special thanks to:
  - everyone that contributed to Rack (see AUTHORS and README)
  - Personifi for giving access to machines to do real world testing
  - the WSGI team for creating a superb specification I just needed to adapt



rack
powers web applications

- These slides: http://chneukirchen.org/talks
  - You’ll also find a paper on Rack there

- http://rack.rubyforge.org
- #rack @ freenode.net

Thanks to: Horacio López a.k.a. vruz, Johan Sørensen, and Aria Stewart for reviewing the slides.
Verbatim copying is allowed as long as this message is preserved. Duplication is encouraged.