Then Jesus said, "Come to me, all of you who are weary and carry heavy burdens, and I will give you **REST**."

Matthew 11:28

Approaching Pure **REST** in Java: HATEOAS and HTTP Tuning

Eder Ignatowicz
Senior Architect, Dextra

@ederign

Software Craftsman @ Dextra
(Architecture, NoSQL, Devops, QA)

PhD Candidate @ Unicamp

Lead Editor @ InfoQ

Board Member @ QConSP

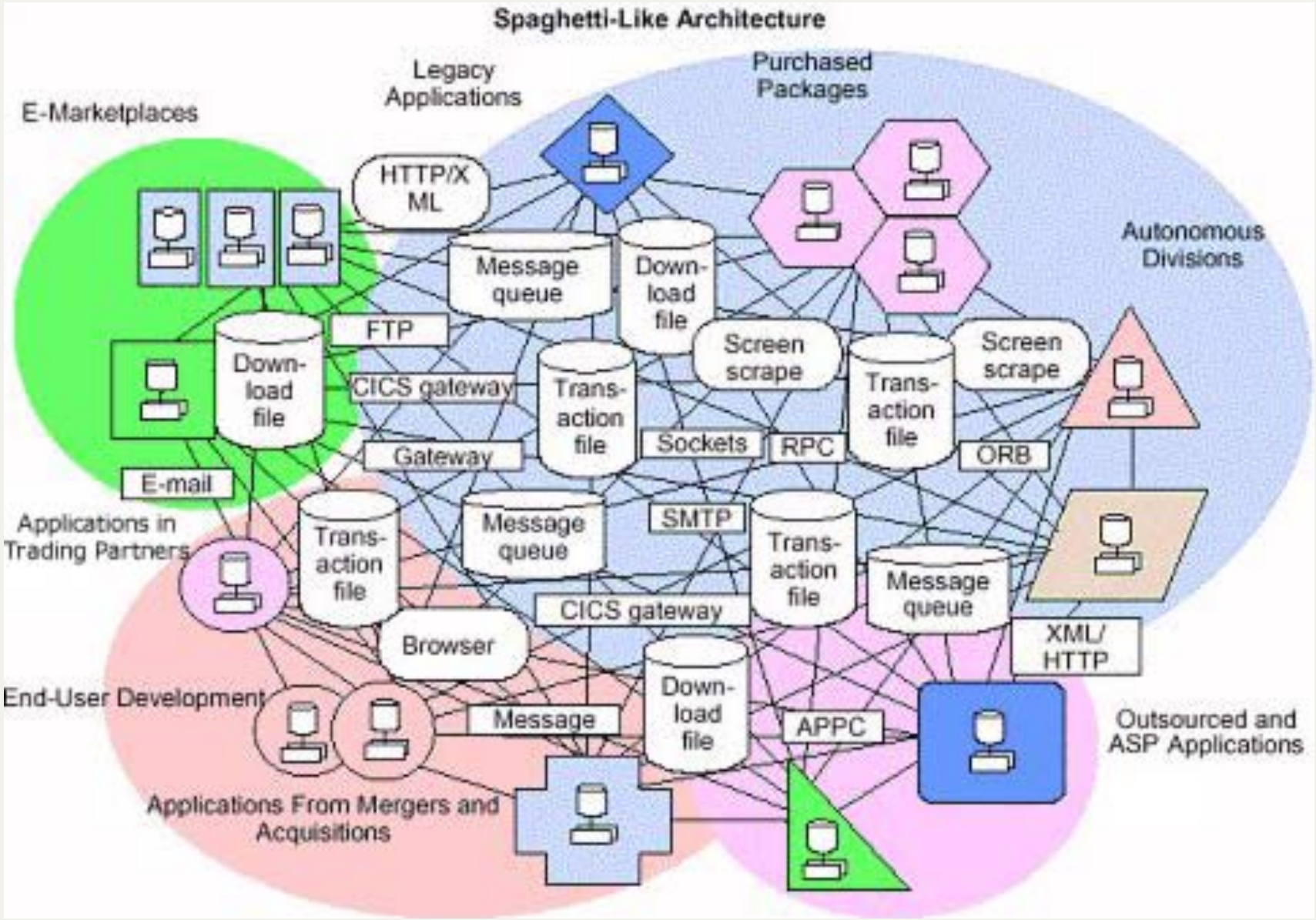
Challenging Projects

"Ponta Firme" Team

Dextra

Brazilian Software Company


The world, before REST



http://geekswithblogs.net/images/geekswithblogs_net/ugandadotnet/eai-spaghetti.jpg

TOUGH TIMES...

LOT OF "PATTERNS"

RMI, Corba, DCOM, Text Files

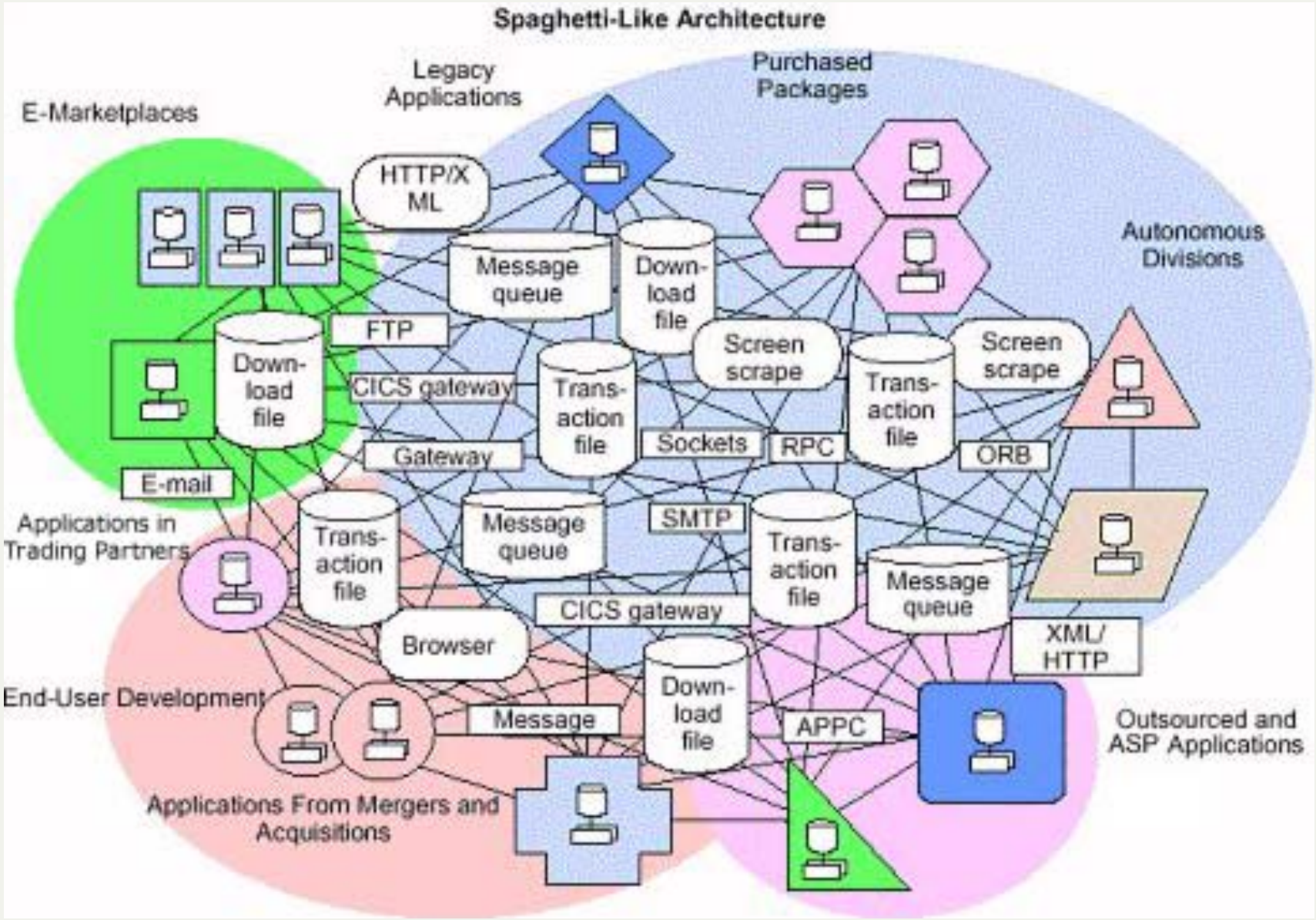
LOT OF SUPPLIERS

Sun, Microsoft, OASIS, OMG

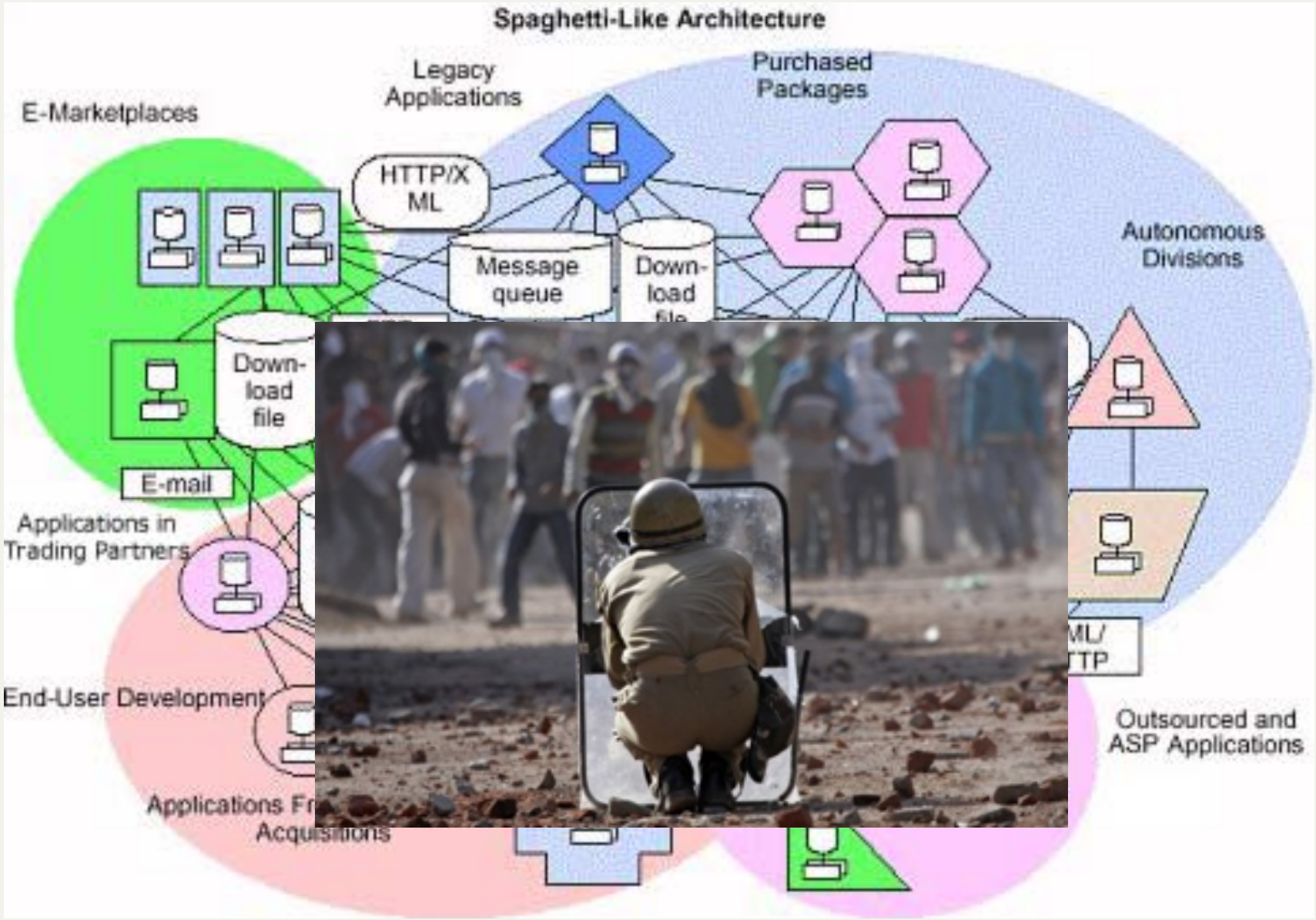
LOT OF TEARS

There is no Interoperability
Reinvent the Wheel
Vendor "lock-in"

SOAP WEB SERVICES



http://geekswithblogs.net/images/geekswithblogs_net/ugandadotnet/eai-spaghetti.jpg



http://geekswithblogs.net/images/geekswithblogs_net/ugandadotnet/eai-spaghetti.jpg


Then REST came to save sinners

Roy Fielding Dissertation

REpresentational State Transfer

REST

a style of software architecture for distributed systems such as the World Wide Web

Principles and Constraints

CLIENT-SERVER
STATELESS
CACHE
UNIFORM INTERFACE
Identification of resources
Manipulation of resources through these representations
Self-descriptive messages
Hypermedia as the engine of application state (aka HATEOAS)
LAYERED SYSTEM
CODE ON DEMAND
(OPTIONAL)

CACHE

HTTP tunning

HATEOAS

GET	Retrieve a representation of a resource
POST	Create a resource
PUT	Replace a resource
DELETE	Delete a resource
PATCH	Update part of a resource

"RESTful"

VERB	URI (Nouns/Verbs)	Action
POST	/bookmarks/create	Create
GET	/bookmarks/show/1	Visualize
POST	/bookmarks/update/1	Update
GET/POST	/bookmarks/delete/1	DELETE

RESTful

VERB	URI (Nouns)	Action
POST	/bookmarks/	Create
GET	/bookmarks/1	Visualize
PUT	/bookmarks/1	Create/Update
DELETE	/bookmarks/1	Delete

Security

Fault-tolerance

Our Dreams to Distributed Systems

Scalability

Low coupling

But extremely misunderstood ...

“

What needs to be done to make the REST architectural style clear on the notion that hypertext is a constraint? In other words, if the engine of application state (and hence the API) is not being driven by hypertext, then it cannot be RESTful and cannot be a REST API. Period. Is there some broken manual somewhere that needs to be fixed?

”

Roy Thomas Fielding

HYPERMEDIA

Richardson's Maturity Model

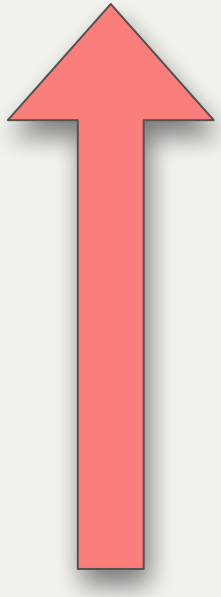
Holy REST

Level 3: Hypermedia Controls

Level 2: HTTP Verbs

Level 1: Resources

Level 0: The swamp of POX



Level 0: Swamp of POX

One **URI**, one **HTTP method**
XML-RPC / **SOAP** / POX
HTTP on **Transport Layer**

```
POST /scheduleService HTTP/1.1
[headers...]

<appointmentRequest>
  <slot doctor = "rcmito" start = "1400" end = "1450"/>
  <patient id = "ederi"/>
</appointmentRequest>
```

Level 1: Resources

Each resource as a URI
URI tunneling
One HTTP verb (POST ou GET)
HTTP on **TRANSPORT LAYER**

```
POST /slots/1234 HTTP/1.1
[headers...]

<appointmentRequest>
    <patient id = "ederi" />
</appointmentRequest>
```

Level 2: HTTP Verbs

LOT of URIs, right use of HTTP verbs
and response codes
Exposes state and not behaviour
CRUD

```
GET /doctors/rcmito/slots?date=20121010?status=open
HTTP/1.1
[headers...]
HTTP/1.1 200 OK

<openSlotList>
    <slot id = “1234” start=”1400” end=”1450”       />
     <slot id = “1234” start=”1600” end=”1650”/>
</openSlotList>
```

0,1 and 2 levels aren't RESTful

(and neither REST)

Level 3: Hypermedia Controls

Hypermedia As The Engine of Application State (HATEOAS)

Self-described resources

A REST client enters a REST application through a simple fixed URL

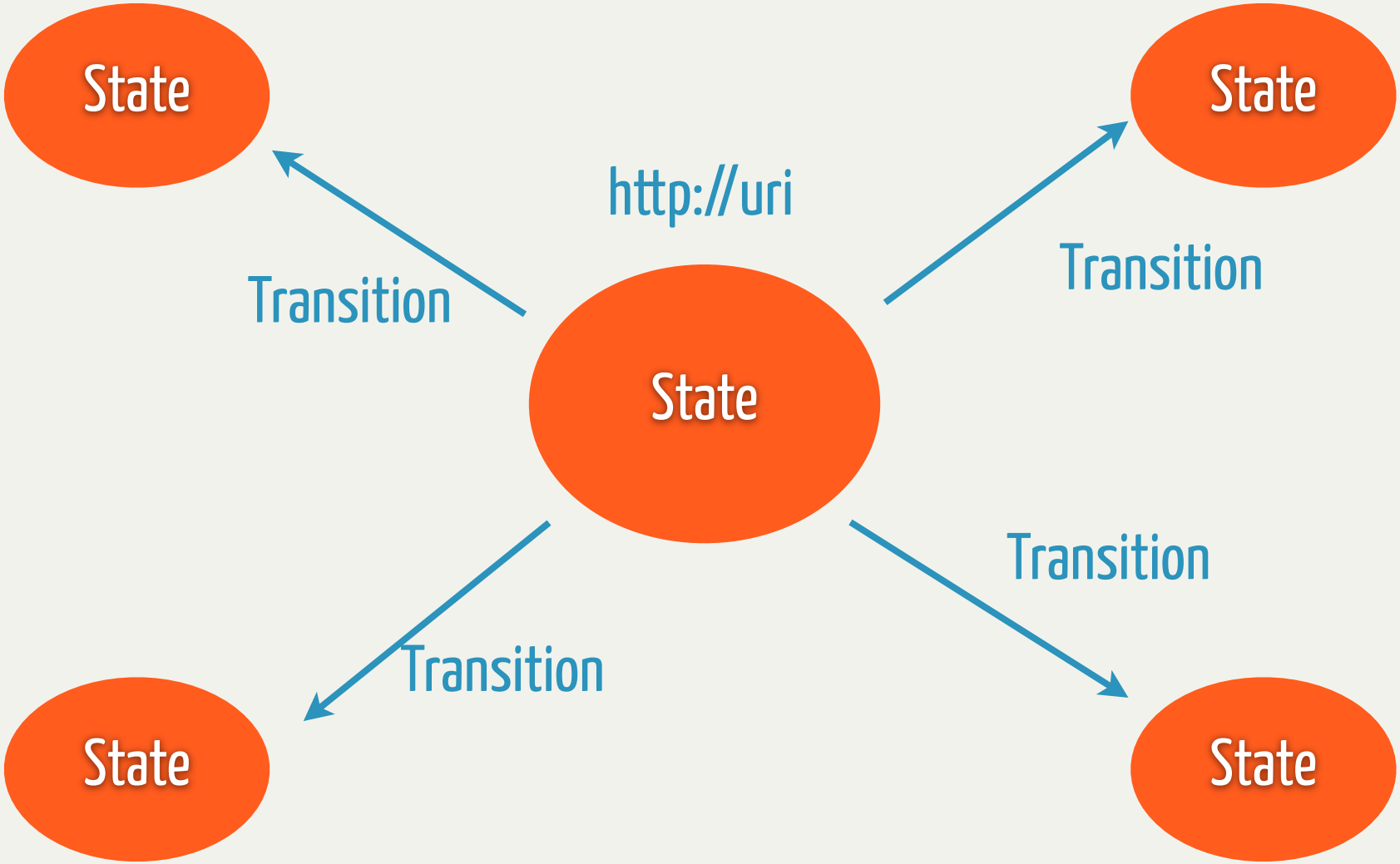
All future actions, the client may take are discovered within resource representations returned from the server.

“

The name **Representational State Transfer** was chosen with the intention to create an image of how a well-designed Web application behaves: a network of web pages (state machine), where the user navigates through selecting links (state transitions), resulting in the next page (next state of the application).

”

Roy Thomas Fielding

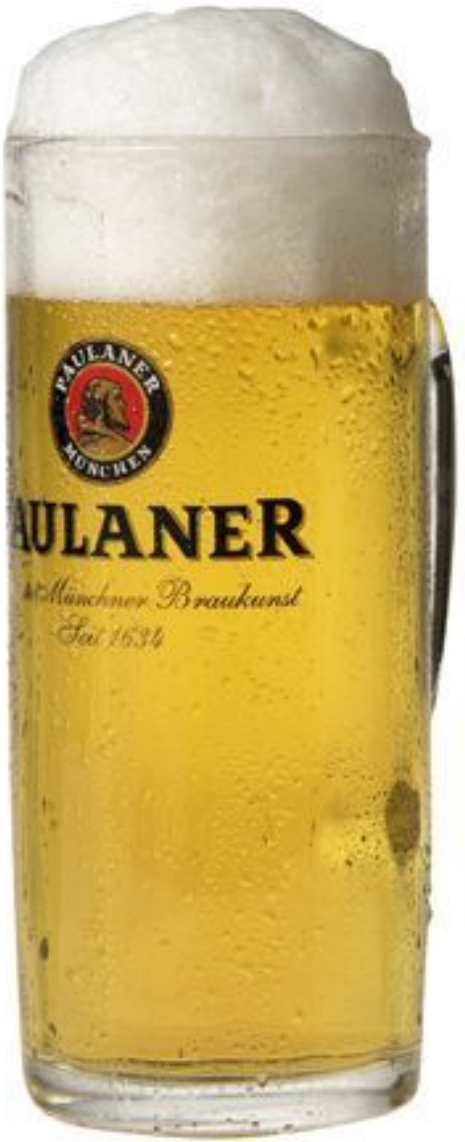

State
State
http://uri
Transition
Transition
State
Transition
Transition
State
State

```
HTTP/1.1 201 Created
Location: http://jogano10.com/slots/1234/appointment
[various headers]

<appointment>
  <slot id = "1234" doctor = "rcmito" start = "1400"
end = "1450"/>
  <patient id = "ederi"/>
  <link rel = "/linkrels/appointment/cancel"
        uri = "/slots/1234/appointment"/>
  <link rel = "/linkrels/appointment/addTest"
        uri = "/slots/1234/appointment/tests"/>
  <link rel = "self"
        uri = "/slots/1234/appointment"/>
  <link rel = "/linkrels/appointment/updateContactInfo"
        uri = "/patients/ederi/contactInfo"/>
</appointment>
```

This is RESTFUL

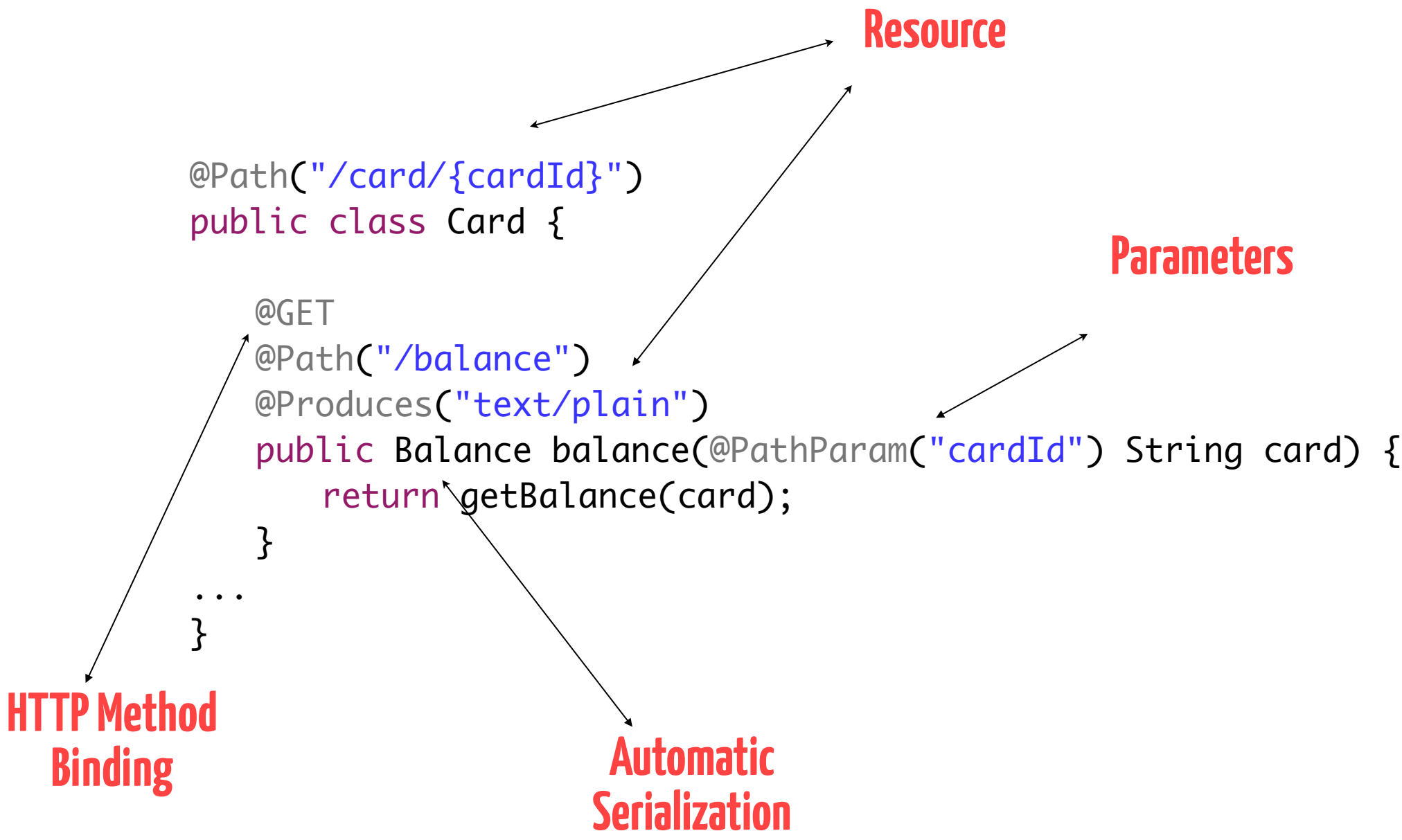
The three great inventions in human history are:


PAULANER
MÜNCHEN
AULANER
Münchner Braukunst
Seit 1634

HYPERLINK !!!

JAX-RS API

JAX-RS API


Resource
@Path("/card/{cardId}")
public class Card {
Parameters
@GET
@Path("/balance")
@Produces("text/plain")
public Balance balance(@PathParam("cardId") String card) {
return getBalance(card);
}
...
}
HTTP Method Binding
Automatic Serialization

JAX-RS API

HATEOAS SUPORT

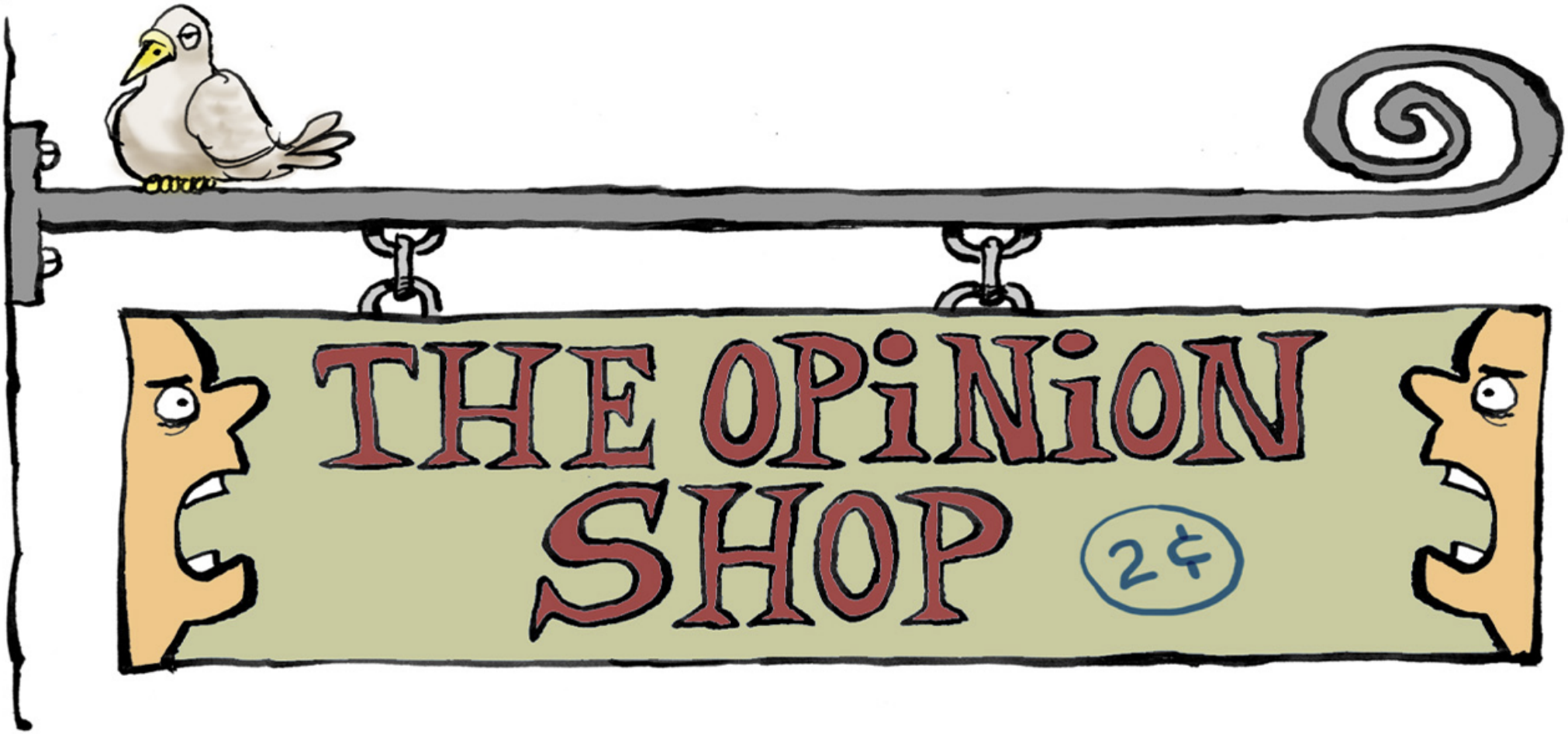
Hypermedia

HATEOAS SUPORT

```
// Server API
Response res = Response.ok(order)
    .link("http://.../orders/1/ship", "ship")
    .build();

// Client API
Response order = client.target(…)
    .request("application/xml").get();

if (order.getLink(“ship”) != null) {
    Response shippedOrder = client
        .target(order.getLink("ship"))
        .request("application/xml").post(null);
 }
```


THE OPiNiON
SHOP
2¢

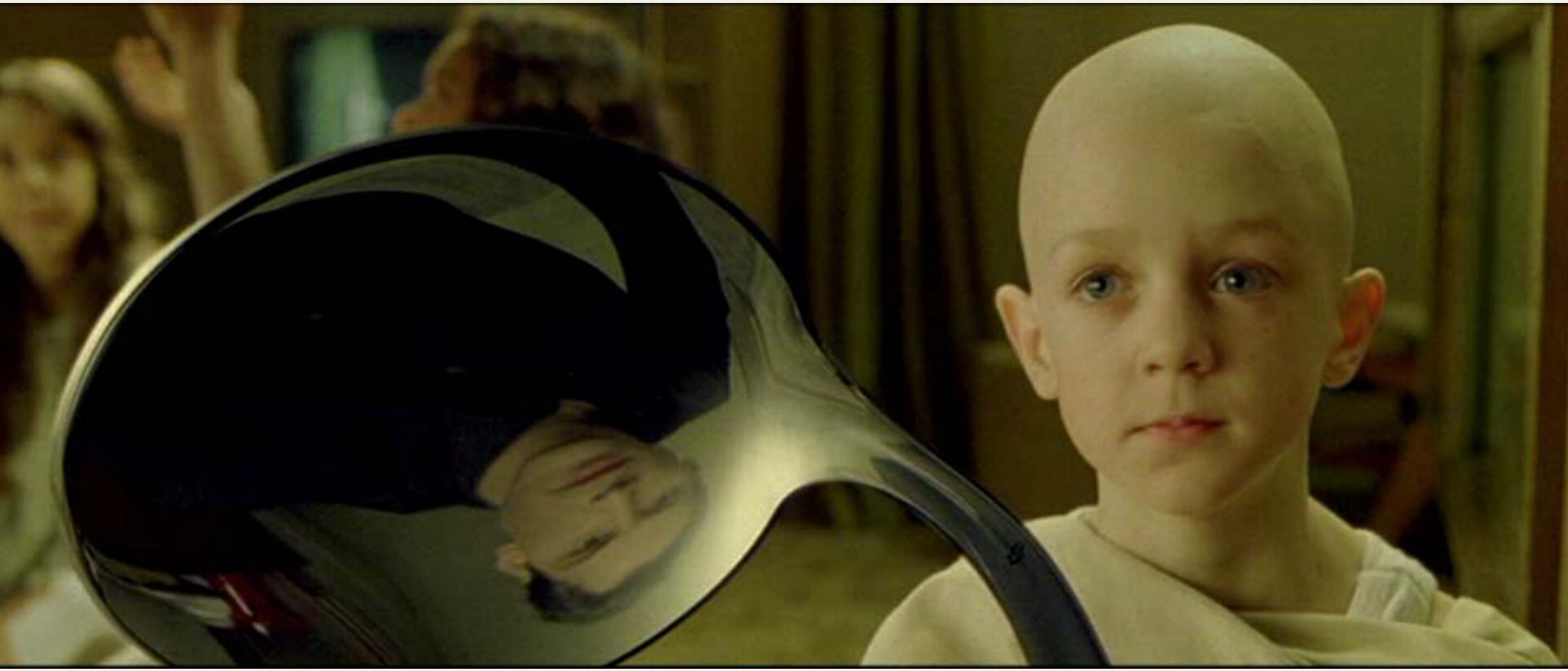
REST without HATEOAS are not REST

There should be no verbs in your URI

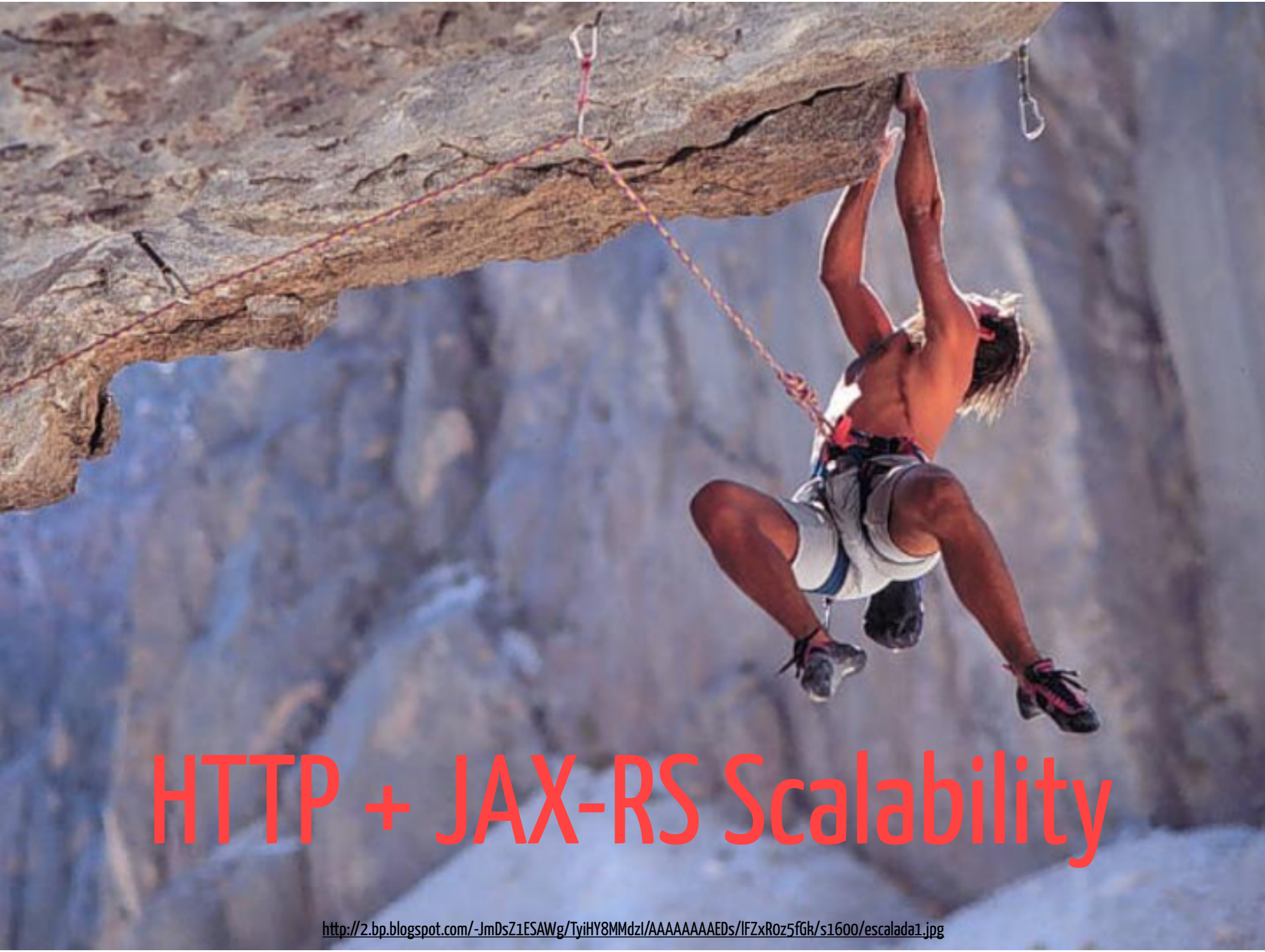
But I need more
then
GET / POST / PUT
DELETE / PATCH semantics

Concurrency

Transactions... ?!?

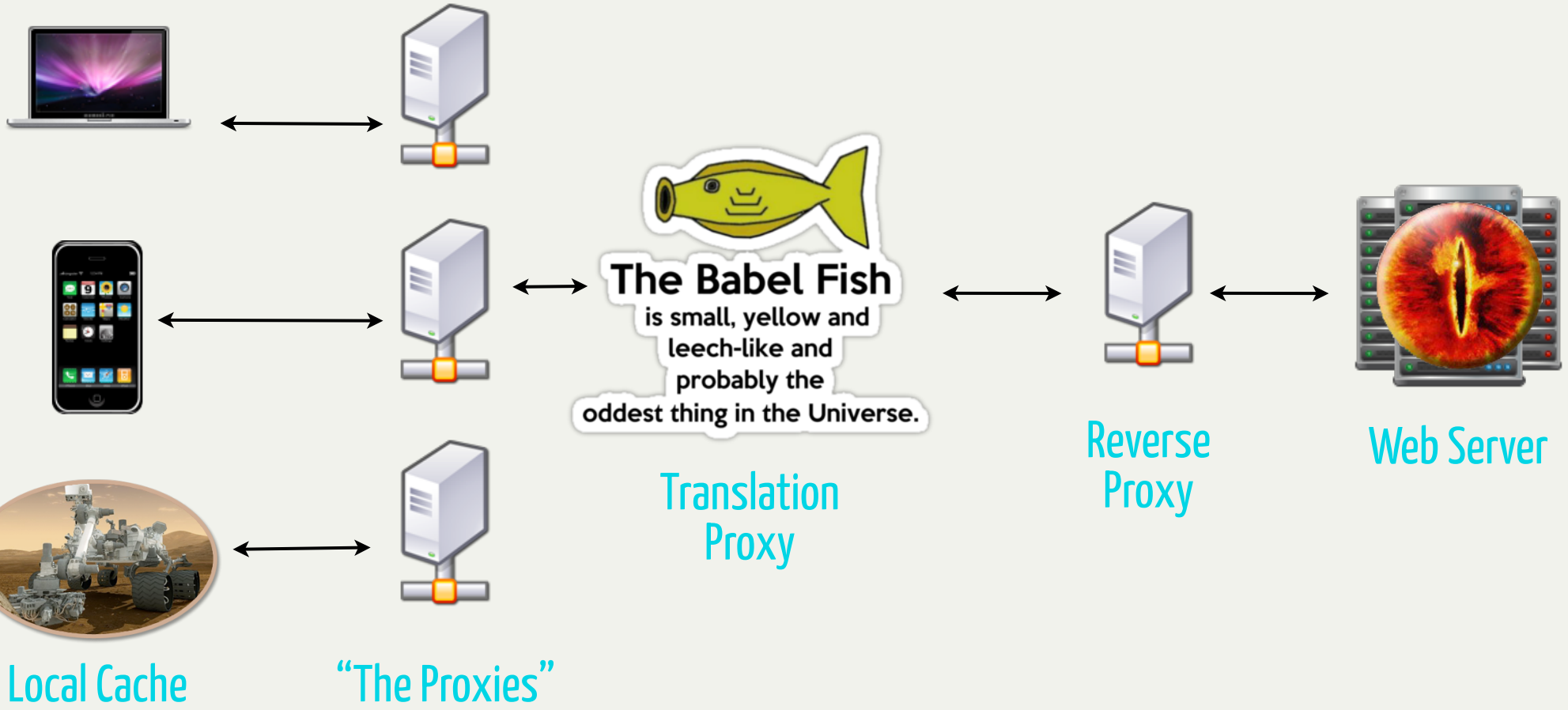


There is (almost)
no need for API or API
versioning!



http://2.bp.blogspot.com/-JmDsZ1ESAWg/TyiHY8MMdzI/AAAAAAAAEDs/lFZxR0z5fGk/s1600/escalada1.jpg

HTTP Caching



HTTP

Caching features

allowed (or not)
expiration
intermediary caches
allowed (or not)
validation
storable(or not)

HTTP

When caches?

Expires header

```
Expires: Sun, 04 Aug 2012 16:00 GMT
```

Cache-control header

```
Cache-Control: no-cache
Cache-Control: public, max-age=3000
```

Validation Header

```
Last-Modified: Mon, 29 Jun 2012 02:28:12 GMT
ETag: "3e86-410-3596fbbc"
```

JAX-RS and Cache Control

```
@Path("/doctors")
public class DoctorsService {
    @Path("/{id}")
    @GET
    @Produces("application/xml")
    public Response getDoctors(@PathParam("id") int id) {
        List<Doctor> doctors = //getDoctors
        CacheControl cc = new CacheControl();
        cc.setMaxAge(3000);
        return Response.ok(doctors).cacheControl(cc).build();
    }
}
```

JAX-RS and Conditional Gets

```
Last-Modified: Mon, 29 Jun 2012 02:28:12 GMT
ETag: "3e86-410-3596fbbc"
```

The cache is valid?

304 “Not Modified”

No?

200 “OK” + valid resource

```java
@Path("/doctors")
public class DoctorsService {
    @Path("/{id}")
    @GET
    @Produces("application/xml")
    public Response getDoctors(@PathParam("id") int id,
            @Context Request request) {
        EntityTag tag = // get fresh tag
        ResponseBuilder builder = null;

        builder = request.evaluatePreconditions(tag);
        if (builder != null){
            return builder.cacheControl(cc).build();
        }
        Object doctors = // getDoctors
        return Response.ok(doctors).cacheControl(cc).build();
    }
}
```

JAX-RS e Cache-Control

```
@Path("/doutores")
public class DoutoresService {
    @Path
    @GET
    @Prod
    publi                                                    id) {
        Li
        Co
        co
        re                                                   build();
    }
}
```

The same principle applies to conditional PUTs e POSTs (concurrency updates)

JSR 339: JAX-RS 2.0

Adopt a JSR

Client API

```
@Before
public void setUp() throws Exception {
     // start the server
     server = Main.startServer();
     // create the client
     Client c = ClientFactory.newClient();
     target = c.target(Main.BASE_URI);
}

@After
public void tearDown() throws Exception {
     server.stop();
}

@Test
public void testExtratoSemHATEOAS() {
     String responseMsg = target.path("cartao/1/saldo").request()
               .get(String.class);
     assertEquals(value, responseMsg);
}
```

Interceptors/Handlers

Extension Points: Logging, Compression, Security, etc.

```
@Provider
class LoggingFilter
    implements RequestFilter, ResponseFilter {

    @Override
    public FilterAction preFilter(FilterContext ctx)
      throws IOException {
        logRequest(ctx.getRequest());
        return FilterAction.NEXT;
    }
    @Override
    public FilterAction postFilter(FilterContext ctx)
      throws IOException {
        logResponse(ctx.getResponse());
        return FilterAction.NEXT;
    }
```

Async

API Client Suport

```
// Acesso URI
Target target = client.target("http://.../atm/balance")…

// Chamada async e callback
Future<?> handle = target.request().async().get(
    new InvocationCallback<String>() {
        public void complete(String balance) { … }
        public void failed(InvocationException e) { … }
      });
```

Java <3

dextra

Coding your Business

@ederign