DESIGNING HTTP INTERFACES AND RESTFUL WEB SERVICES

David Zuelke

David Zülke

http://en.wikipedia.org/wiki/File:München_Panorama.JPG

Founder

Bitextender

Lead Developer

Agavi

@dzuelke

THE OLDEN DAYS

Before REST was *En Vogue*

http://www.acme.com/index.php?action=zomg&page=lol

along came

dis is srs SEO bsns

and said

NEIN NEIN NEIN NEIN DAS IST VERBOTEN

at least if they were

so we had to make URLs "SEO friendly"

http://www.acme.com/zomg/lol

and then things got out of control

because nobody really had a clue

http://acme.com/videos/latest/hamburgers

http://acme.com/search/lolcats/pictures/yes/1/200

oh dear…

THE RISE OF WEB SERVICES

Ohai, I'm ur CEO, I canhaz SOAP API plz, today, kthx?

```
POST /soapendpoint.php HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <ns1:getProduct xmlns:ns1="http://agavi.org/sampleapp">
      <id>123456</id>
    </ns1:getProduct>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <ns1:getProductResponse xmlns:ns1="http://agavi.org/sampleapp">
      <product>
        <id>123456</id>
        <name>Red Stapler</name>
        <price>3.14</price>
      </product>
    </ns1:getProductResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
POST /soapendpoint.php HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <ns1:getProduct xmlns:ns1="http://agavi.org/sampleapp">
      <id>987654</id>
    </ns1:getProduct>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

```
HTTP/1.1 500 Internal Service Error
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Server</faultcode>
      <faultstring>Unknown Product </faultstring>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP sucks, said everyone

let's build APIs without the clutter, they said

example: the http://joind.in/ API

```
POST /api/talk HTTP/1.1
Host: joind.in
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" encoding="UTF-8"?>
<request>
        <auth>
                <user>Chuck Norris</user>
                <pass>roundhousekick</pass>
        </auth>
        <action type="getdetail">
                <talk_id>42</talk_id>
        </action>
</request>
```

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8

<?xml version="1.0" encoding="UTF-8"?>
<response>
    <item>
        <talk_title>My Test Talk</talk_title>
        <talk_desc>This is a sample talk description</talk_desc>
        <ID>42</ID>
    </item>
</response>
```

PROBLEMS WITH THIS API

- Always a POST
- Doesn't use HTTP Authentication
- Operation information is enclosed in the request ("getdetail")
- Nothing there is cacheable
- Everything through one endpoint (/api/talks for talks)

Level 0 in the **Richardson Maturity Model**:
Plain old XML over the wire in an RPC fashion

Room for "improvement:" use one URI for each resource

That would be **Level 1** in Richardson's Maturity Model

Level 0 and Level 1 are a bag of hurt.
Do not use them.
Ever.

ALONG CAME ROY FIELDING

And Gave Us REST

that was awesome

because everyone could say

I haz REST nao

when in fact

they bloody didn’t

REST

What Does That Even Mean?

REpresentational State Transfer

Roy Thomas Fielding: *Architectural styles and the design of network based software architectures.*

REST CONSTRAINTS

- Client-Server
- Stateless
- Cacheable
- Layered System
- Code on Demand (optional)
- Uniform Interface

UNIFORM INTERFACE

- A *URL* identifies a *Resource*
- *Methods* perform *operations* on resources
- The operation is implicit and **not** part of the URL
- A *hypermedia format* is used to represent the data
- *Link relations* are used to navigate a service

a web page is *not* a resource

it is a (complete) *representation* of a resource

GETTING JSON BACK

```
GET /products/ HTTP/1.1
Host: acme.com
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Allow: GET, POST

[
  {
    id: 1234,
    name: "Red Stapler",
    price: 3.14,
    location: "http://acme.com/products/1234"
  }
]
```

GETTING XML BACK

```
GET /products/ HTTP/1.1
Host: acme.com
Accept: application/xml
```

```
HTTP/1.1 200 OK
Content-Type: application/xml; charset=utf-8
Allow: GET, POST

<?xml version="1.0" encoding="utf-8"?>
<products xmlns="urn:com.acme.products" xmlns:xl="http://www.w3.org/1999/xlink">
  <product id="1234" xl:type="simple" xl:href="http://acme.com/products/1234">
    <name>Red Stapler</name>
    <price currency="EUR">3.14</price>
  </product>
</products>
```

but those are not hypermedia formats!

(more on that a bit later)

AND FINALLY, HTML

```
GET /products/ HTTP/1.1
Host: acme.com
Accept: application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,*/*;q=0.5
User-Agent: Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_5_8; en-us) AppleWebKit…
```

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=utf-8
Allow: GET, POST

<html lang="en">
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8"></meta>
    <title>ACME Inc. Products</title>
  </head>
  <body>
    <h1>Our Incredible Products</h1>
    <ul id="products">
      <li><a href="http://acme.com/products/1234">Red Stapler</a> (€3.14)</li>
    </ul>
  </body>
</html>
```

VOLUME ONE

Designing an HTTP Interface

FIRST: DEFINE RESOURCES

A Good Approach: Structure Your URLs

BAD URLS

- http://www.acme.com/product/
- http://www.acme.com/product/filter/cats/desc
- http://www.acme.com/product/1234
- http://www.acme.com/photos/product/1234
- http://www.acme.com/photos/product/1234/new
- http://www.acme.com/photos/product/1234/5678

WTF?

new what?

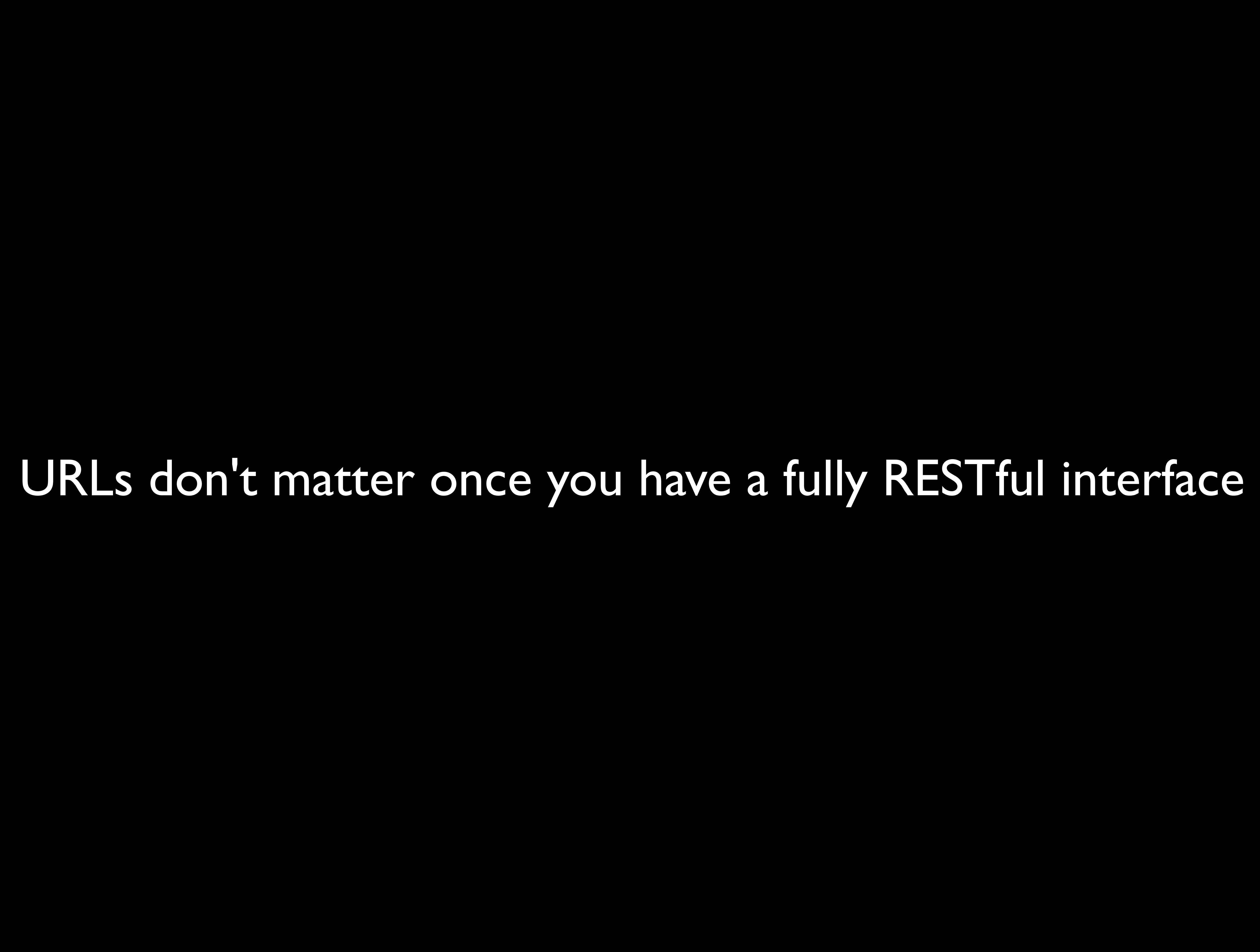
photo or product ID?

GOOD URLS

a list of products
filtering is a query
a single product
all photos

- http://www.acme.com/products/
- http://www.acme.com/products/?filter=cats&sort=desc
- http://www.acme.com/products/1234
- http://www.acme.com/products/1234/photos/
- http://www.acme.com/products/1234/photos/?sort=latest
- http://www.acme.com/products/1234/photos/5678

now here's the ironic part


URLs don't matter once you have a fully RESTful interface

but it’s helpful to think in terms of *resources*

SECOND: USE RESOURCES

CRUD, but not really

COLLECTION OPERATIONS

- http://www.acme.com/products/
  - GET to *retrieve* a list of products
  - POST to *create* a new product
    - returns
      - 201 Created
      - Location: http://www.acme.com/products/1235

ITEM OPERATIONS

- http://www.acme.com/products/1234
  - GET to *retrieve*
  - PUT to *update*
  - DELETE to, you guessed it, *delete*

and remember

don't let the server maintain client state (e.g. cookies)

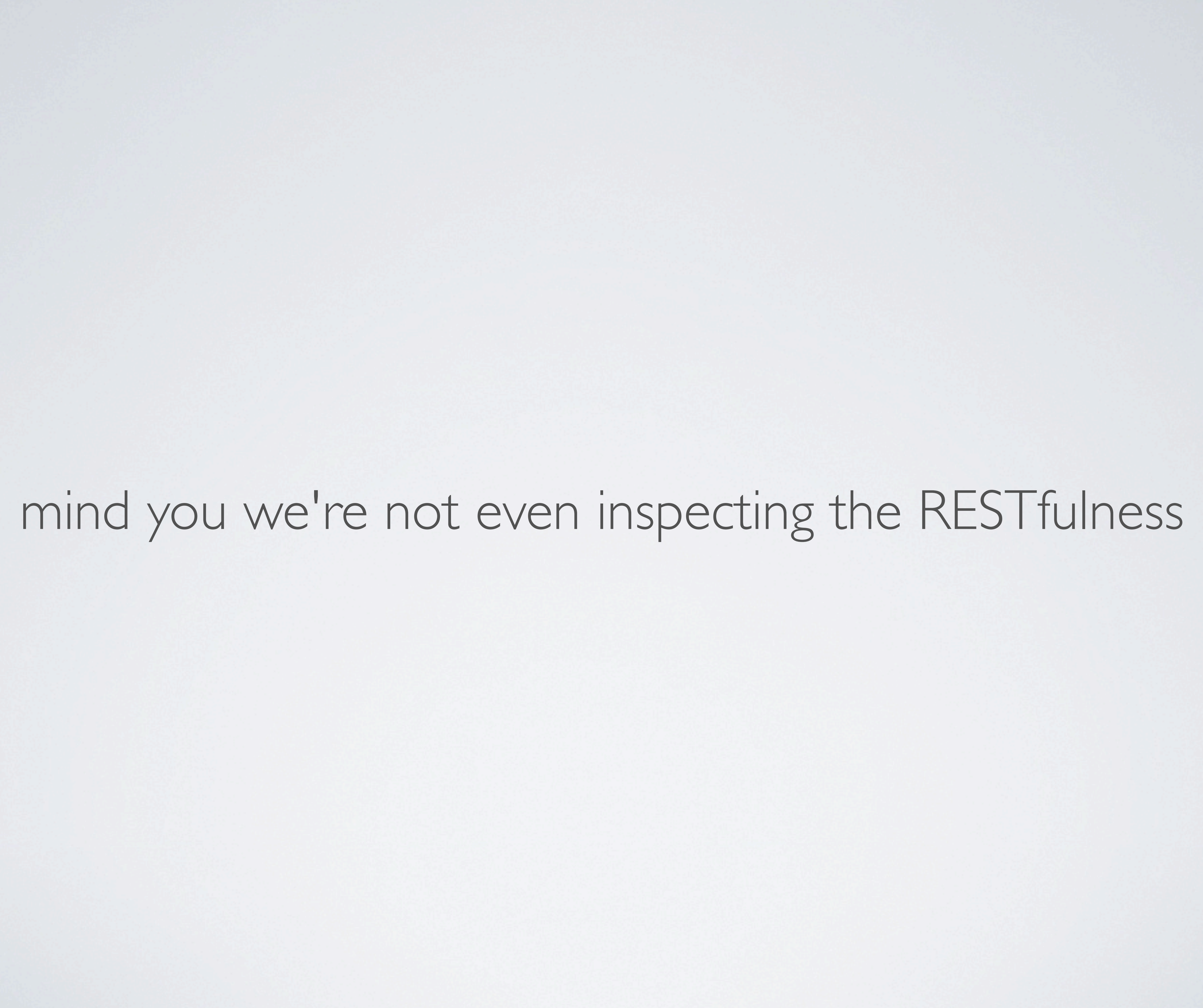
Now we are at **Level 2** in RMM

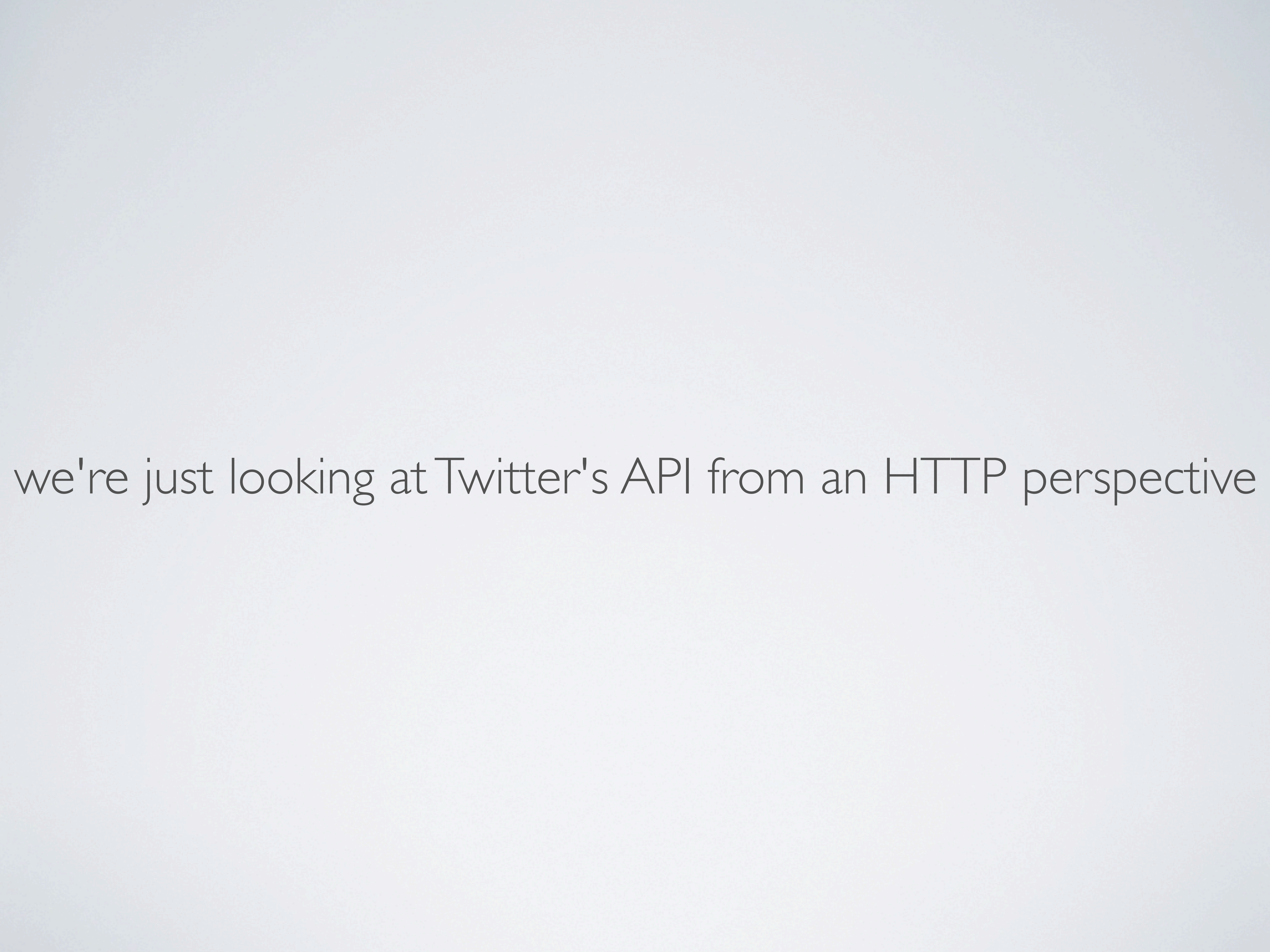
RMM LEVEL 2

- Use HTTP verbs
  - GET (safe and idempotent)
  - POST (unsafe, not idempotent)
  - PUT & DELETE (unsafe, idempotent)
- Use HTTP status codes to indicate result success
  - e.g. HTTP/1.1 409 Conflict

THE TWITTER API

Not RESTful, And Not Even Getting HTTP Right :(


mind you we're not even inspecting the RESTfulness


we're just looking at Twitter's API from an HTTP perspective

CURRENT STATE

Doesn't allow Accept header

- GET http://api.twitter.com/1/statuses/show/12345.json

Posts to auth'd user!

- POST http://api.twitter.com/1/statuses/update.json

"DELETE destroy", RPC much?

- DELETE http://api.twitter.com/1/statuses/destroy/12345.json
- GET http://api.twitter.com/1/statuses/retweets/12345.json

Why the difference?

- PUT http://api.twitter.com/1/statuses/retweet/12345.json

Why a PUT?

COULD BE SO MUCH SIMPLER

- http://twitter.com/*username*/statuses/
    - POST to create a new tweet
- http://twitter.com/*username*/statuses/12345
    - DELETE deletes (PUT could be used for updates)
- http://twitter.com/*username*/statuses/12345/retweets/
    - POST creates a new retweet

INTERMISSION

What's the Biggest Reason for the Success of the Web?

WWW

first data exchange system

planetary scale

why is that possible?

Hyperlinks!

no tight coupling!

loosely coupled by design

no notification infrastructure

HTTP/1.1 404 Not Found

embraces failure

more information != more friction

no limits to scalability

WWW is **protocol-centric**

VOLUME TWO

RESTful Services with Hypermedia

THE UNIFORM INTERFACE

- Identification of Resources (e.g. through URIs)
  - Representations are conceptually separate!
- Manipulation Through Representations (i.e. they are complete)
- Self-Descriptive Messages (containing all information)
- Hypermedia As The Engine Of Application State ("HATEOAS")

magic awesomesauce essential to REST

HATEOAS

The Missing Piece in the Puzzle

ONE LAST PIECE IS MISSING

- How does a client know what to do with representations?
- How do you go to the “next” operation?
- What are the URLs for creating subordinate resources?
- Where is the *contract* for the service?

HYPERMEDIA AS THE ENGINE OF APPLICATION STATE

- Use links to allow clients to discover locations and operations
- Link relations are used to express the possible options
- Clients do not need to know URLs, so they can change
- The entire application workflow is abstracted, thus changeable
- The hypermedia type itself could be versioned if necessary
- No breaking of clients if the implementation is updated!

(X)HTML and Atom are Hypermedia formats

Or you roll your own...

A CUSTOM MEDIA TYPE

```
GET /products/1234 HTTP/1.1
Host: acme.com
Accept: application/vnd.com.acme.shop+xml
```

Remind clients of Uniform Interface :)

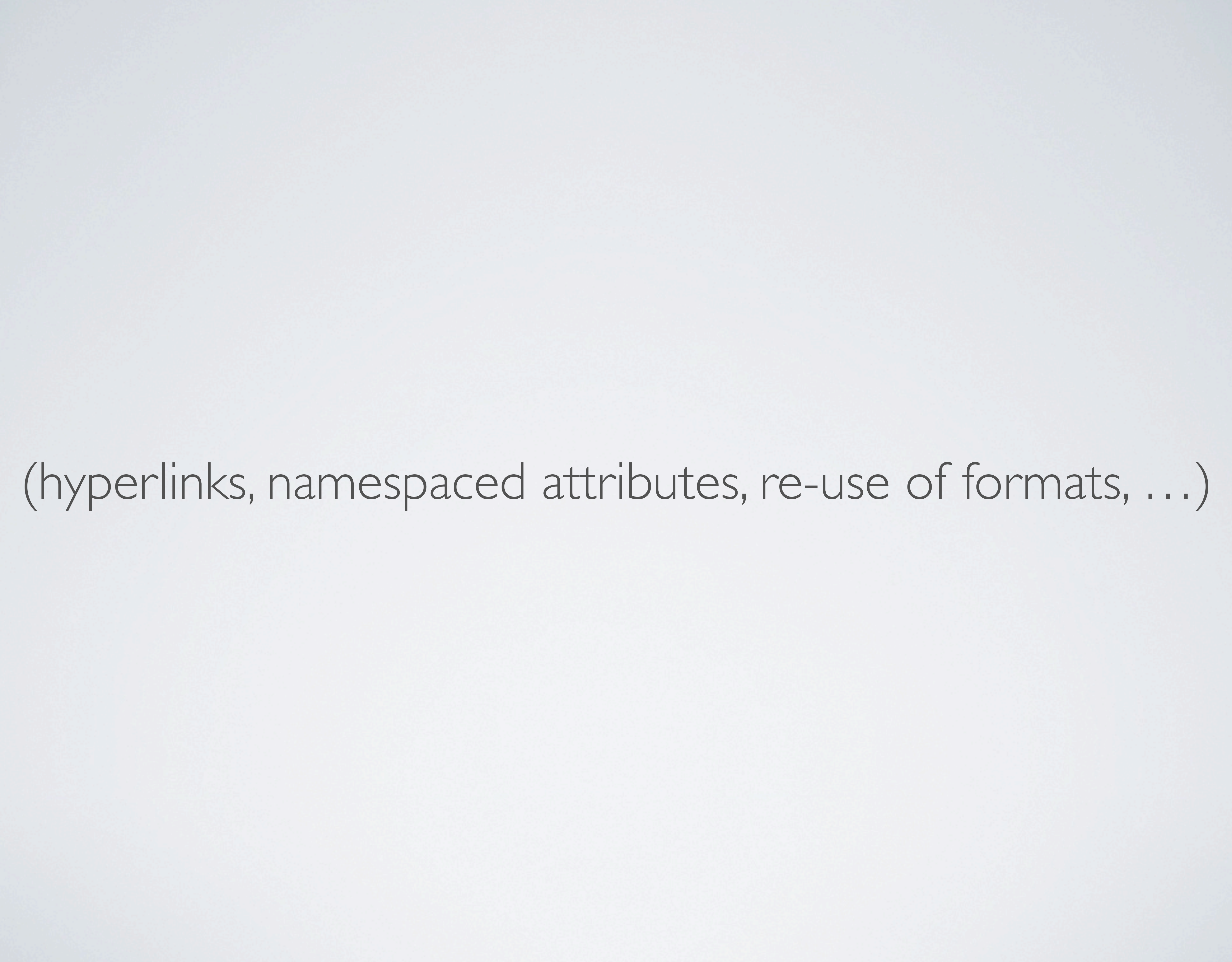
```
HTTP/1.1 200 OK
Content-Type: application/vnd.come.acme.shop+xml; charset=utf-8
Allow: GET, PUT, DELETE

<?xml version="1.0" encoding="utf-8"?>
<product xmlns="urn:com.acme.prods" xmlns:atom="http://www.w3.org/2005/Atom">
  <id>1234</id>
  <name>Red Stapler</name>
  <price currency="EUR">3.14</price>
  <atom:link rel="payment" type="application/vnd.com.acme.shop+xml"
             href="http://acme.com/products/1234/payment"/>
</product>
```

re-use Atom for link relations

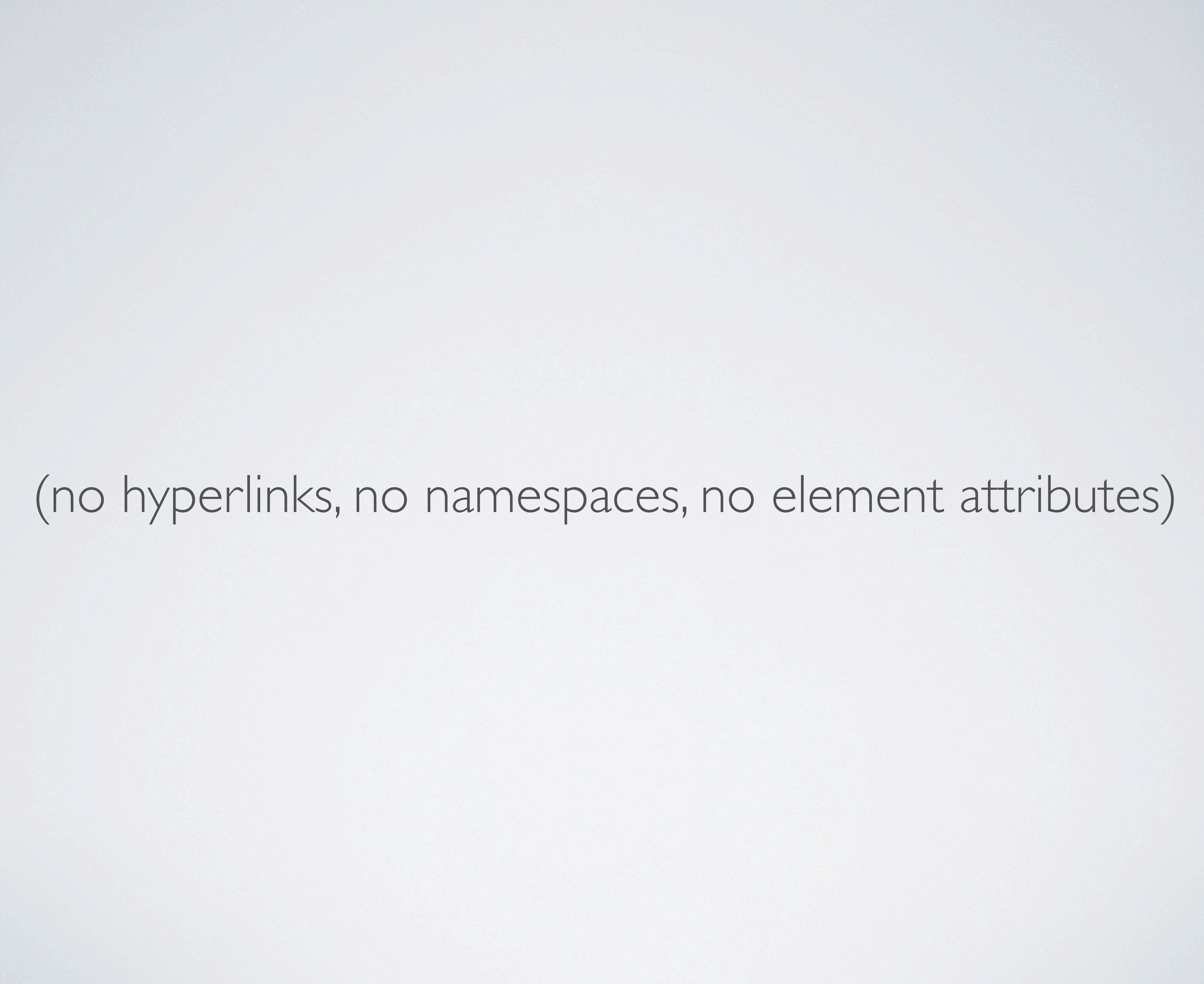
meaning defined in IANA Link Relations list

boom, RMM **Level 3**

XML is really good for hypermedia formats


(hyperlinks, namespaced attributes, re-use of formats, …)

JSON is more difficult


(no hyperlinks, no namespaces, no element attributes)

XML VERSUS JSON

```
<?xml version="1.0" encoding="utf-8"?>
<product xmlns="urn:com.acme.prods" xmlns:atom="http://www.w3.org/2005/xlink">
  <id>1234</id>
  <name>Red Stapler</name>
  <price currency="EUR">3.14</price>
  <atom:link rel="payment" type="application/com.acme.shop+xml"
             href="http://acme.com/products/1234/payment"/>
</product>
```

```
{
  id: 1234,
  name: "Red Stapler",
  price: {
    amount: 3.14,
    currency: "EUR"
  },
  links: [
    {
      rel: "payment",
      type: "application/vnd.com.acme.shop+json",
      href: "http://acme.com/products/1234/payment"
    }
  ]
}
```

also, JSON is hard to evolve without breaking clients

```xml
<?xml version="1.0" encoding="utf-8"?>
<products xmlns="http://acme.com/shop/products">
  <product id="123">
    <name>Bacon</name>
    <price>5.99</price>
  </product>
</products>
```

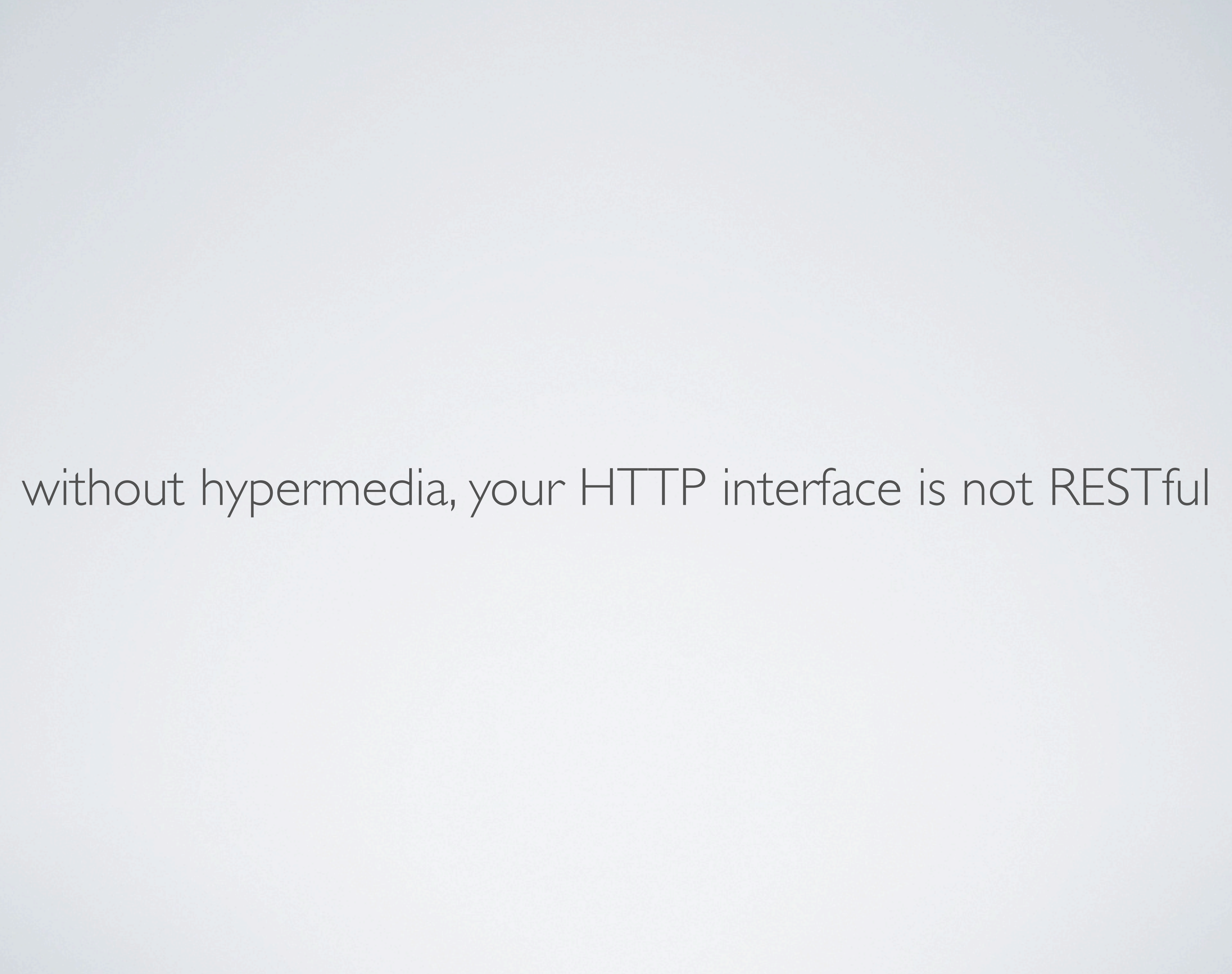
```xml
<?xml version="1.0" encoding="utf-8"?>
<products xmlns="http://acme.com/shop/products">
  <product id="123">
    <name>Bacon</name>
    <price>5.99</price>
    OMNOMNOM Bacon
  </product>
</products>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<products xmlns="http://acme.com/shop/products">
  <product id="123">
    <name>Bacon</name>
    <price>5.99</price>
    <price currency="EUR">4.49</price>
  </product>
</products>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<products xmlns="http://acme.com/shop/products">
  <product id="123">
    <name xml:lang="en">Bacon</name>
    <name xml:lang="de">Speck</name>
    <price>5.99</price>
  </product>
</products>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<products xmlns="http://acme.com/shop/products">
  <product id="123">
    <name xml:lang="en">Bacon</name>
    <name xml:lang="de">Speck</name>
    <price>5.99</price>
    <link rel="category" href="..." />
  </product>
</products>
```

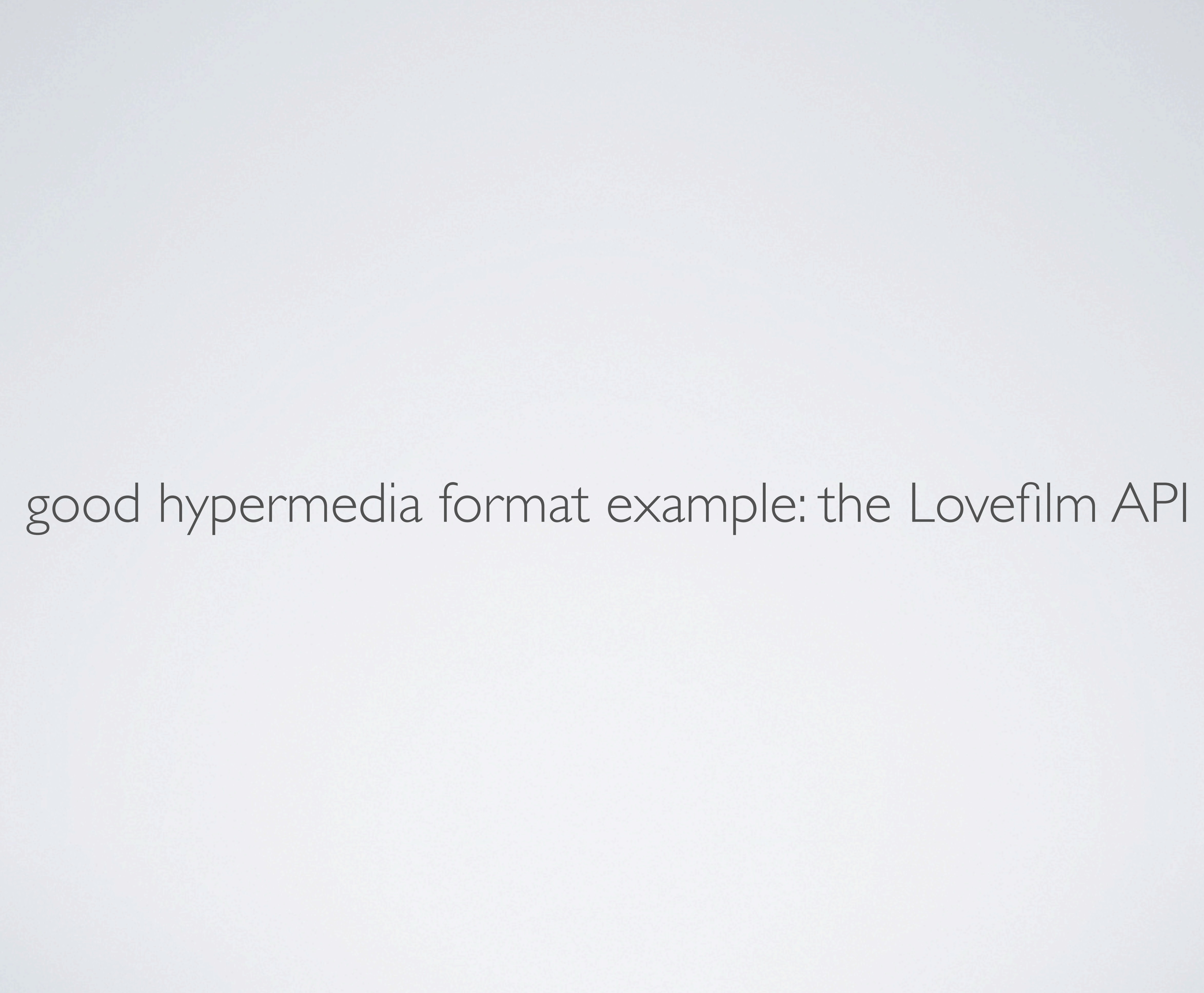
and hey


without hypermedia, your HTTP interface is not RESTful

that’s totally fine
and sometimes even the only way to do it

(e.g. CouchDB or S3 are never going to be RESTful)

just avoid calling it a "REST API" :)


good hypermedia format example: the Lovefilm API

```xml
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<search>
  <total_results>6</total_results>
  <items_per_page>1</items_per_page>
  <start_index>1</start_index>
  <link href="http://openapi.lovefilm.com/catalog/games?start_index=1&amp;items_per_page=1&amp;term=old"
        rel="self" title="self"/>
  <link href="http://openapi.lovefilm.com/catalog/games?start_index=2&amp;items_per_page=1&amp;term=old"
        rel="next" title="next"/>
  <link href="http://openapi.lovefilm.com/catalog/games?start_index=6&amp;items_per_page=1&amp;term=old"
        rel="last" title="last"/>
  <catalog_title>
    <can_rent>true</can_rent>
    <release_date>2003-09-12</release_date>
    <title full="Star Wars: Knights of the Old Republic" clean="Star Wars: Knights of the Old Republic"/>
    <id>http://openapi.lovefilm.com/catalog/title/59643</id>
    <adult>false</adult>
    <number_of_ratings>574</number_of_ratings>
    <rating>4</rating>
    <category scheme="http://openapi.lovefilm.com/categories/catalog" term="games"/>
    <category scheme="http://openapi.lovefilm.com/categories/format" term="Xbox"/>
    <category scheme="http://openapi.lovefilm.com/categories/genres" term="Adventure"/>
    <category scheme="http://openapi.lovefilm.com/categories/genres" term="Role-playing"/>
    <category scheme="http://openapi.lovefilm.com/categories/certificates/bbfc" term="TBC"/>
    <link href="http://openapi.lovefilm.com/catalog/title/59643/synopsis"
          rel="http://schemas.lovefilm.com/synopsis" title="synopsis"/>
    <link href="http://openapi.lovefilm.com/catalog/title/59643/reviews"
          rel="http://schemas.lovefilm.com/reviews" title="reviews"/>
    <link href="http://www.lovefilm.com/product/59643-Star-Wars-Knights-of-the-Old-Republic.html?cid=LFAPI"
          rel="alternate" title="web page"/>
  </catalog_title>
</search>
```

ROOM FOR IMPROVEMENT IN THE LOVEFILM API

- Uses application/xml instead of a custom media type
  - Once that is fixed, all the link elements could also have a “type” attribute indicating the media type
- Should use XML namespaces on the root element, with one namespace per type (e.g. “urn:com.lovefilm.api.item”, “urn:com.lovefilm.api.searchresult” and so on)
  - That way, clients can determine the resource type easily

another great RESTful API: Huddle

```xml
<document
  xmlns="http://schema.huddle.net/2011/02/"
  title="TPS report May 2010"
  description="relentlessly mundane and enervating.">

  <link rel="self" href="..." />
  <link rel="parent" href="..." title="..."/>
  <link rel="edit" href="..." />
  <link rel="delete" href="..." />
  <link rel="content" href="..." title="..." type="..." />
  <link rel="thumb" href="..." />
  <link rel="version-history" href="..." />
  <link rel="create-version" href="..." />
  <link rel="comments" href="..." />

  <actor name="Peter Gibson" rel="owner">
    <link rel="self" href="..." />
    <link rel="avatar" href="..." type="image/jpg" />
    <link rel="alternate" href="..." type="text/html" />
  </actor>

  <actor name="Barry Potter" rel="updated-by">
    <link rel="self" href="..." />
    <link rel="avatar" href="..." type="image/jpg" />
    <link rel="alternate" href="..." type="text/html" />
  </actor>

  <size>19475</size>

  <version>98</version>
  <created>2007-10-10T09:02:17Z</created>
  <updated>2011-10-10T09:02:17Z</updated>
  <processingStatus>Complete</processingStatus>
  <views>9</views>
</document>
```

ROOM FOR IMPROVEMENT IN THE HUDDLE API

- Uses custom rels like "thumb" or "avatar" not defined in the IANA registry (http://www.iana.org/assignments/link-relations)
  - Risk of collisions and ambiguity; should use something like "http://rels.huddle.net/thumb" instead.
- Uses one global XML schema and namespace for all entities
  - Clients cannot detect entity type based on namespace
  - Difficult to evolve schema versions independently

API VERSIONING

Media Types To The Rescue!

why not api.myservice.com/v1/foo/bar?
and then api.myservice.com/v2/foo/bar?

different URLs means different resources!

also, keep bookmarks (by machines) in mind

API VERSION 1

```
GET /products HTTP/1.1
Host: acme.com
Accept: application/vnd.com.myservice+xml
```

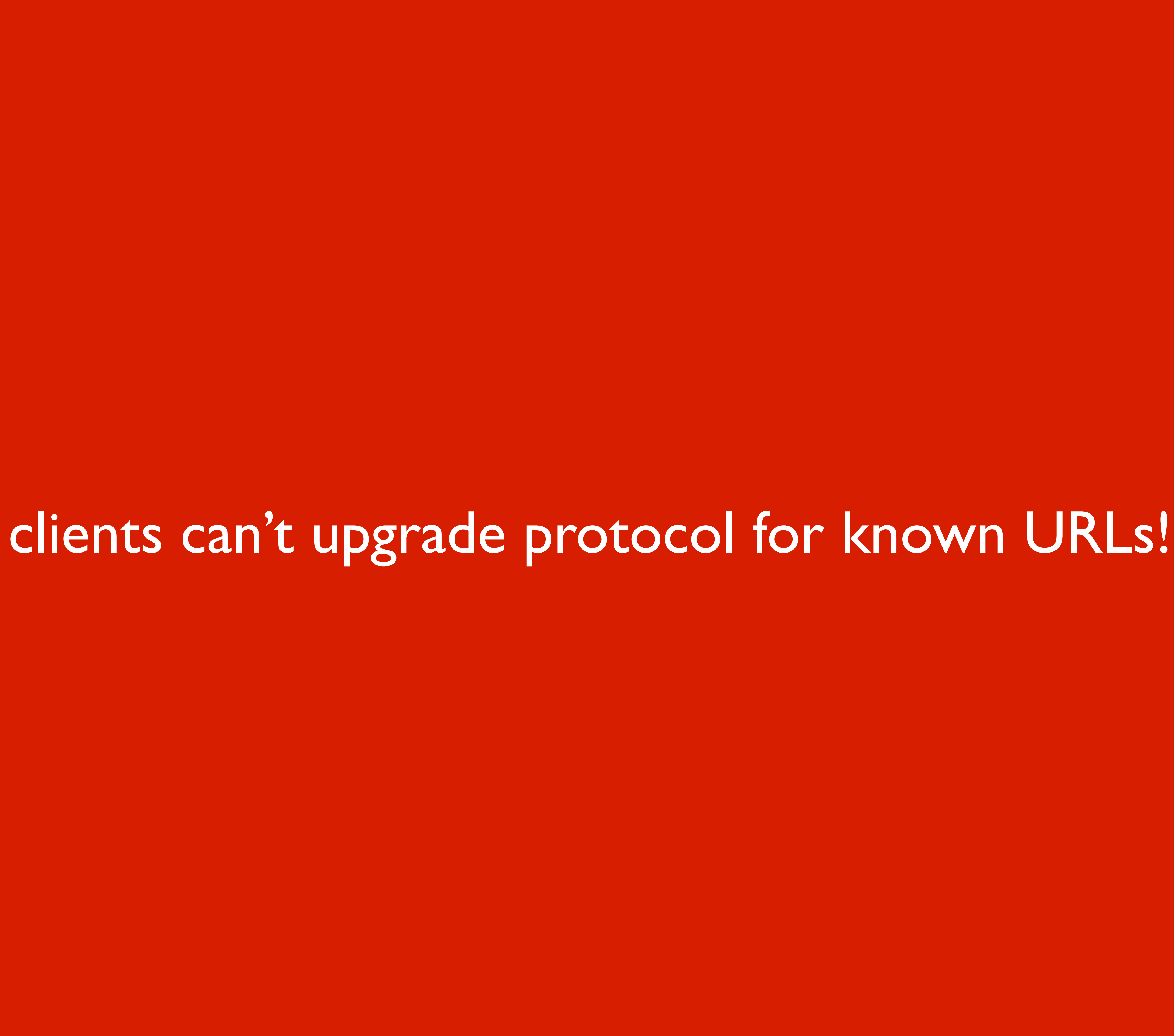
```
HTTP/1.1 200 OK
Content-Type: application/vnd.com.myservice+xml; charset=utf-8
Allow: GET, POST

<?xml version="1.0" encoding="utf-8"?>
<products xmlns="urn:com.acme.products" xmlns:xl="http://www.w3.org/1999/xlink">
  <product id="1234" xl:type="simple" xl:href="http://acme.com/products/1234">
    <name>Red Stapler</name>
    <price currency="EUR">3.14</price>
  </product>
</products>
```

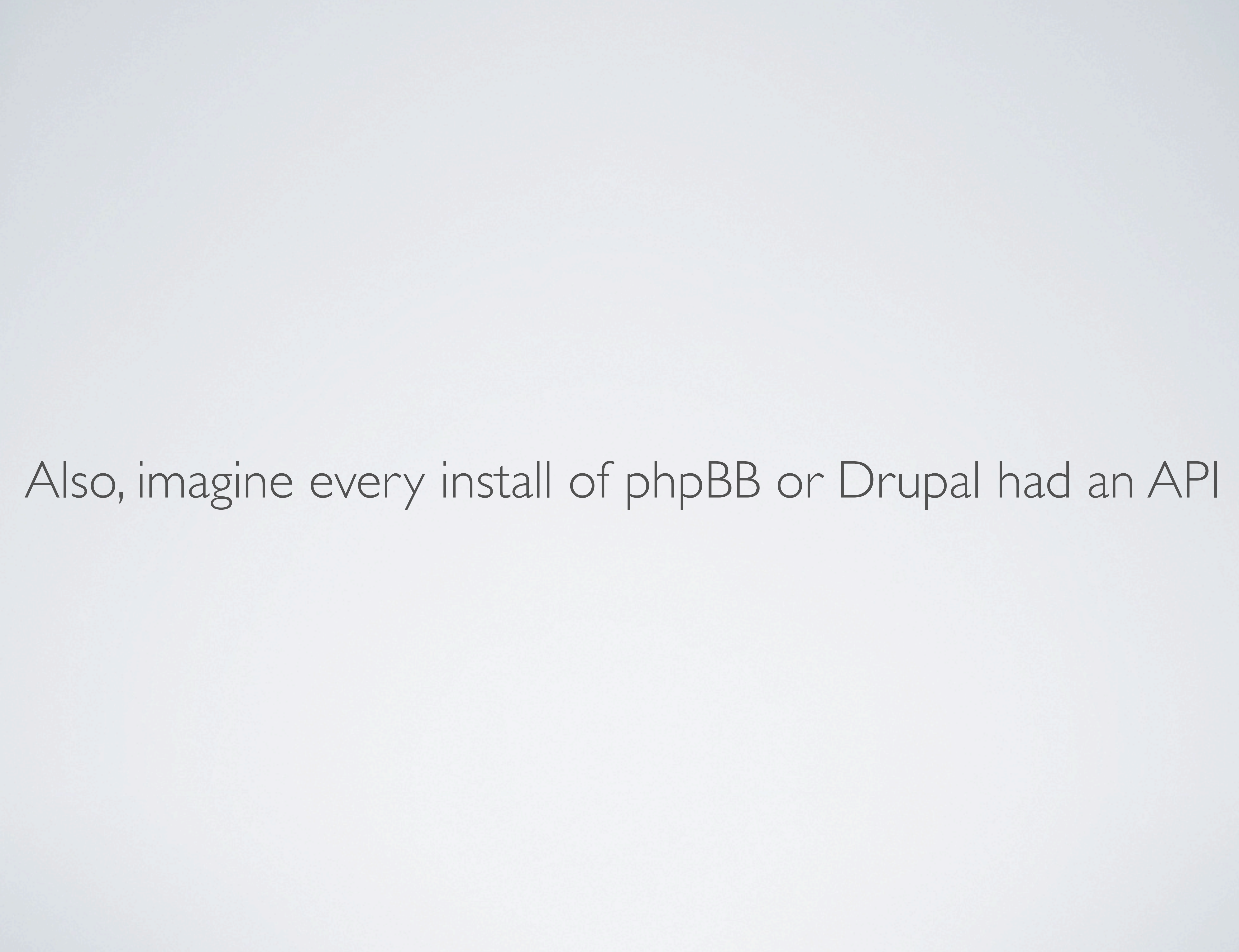
(some years pass...)

API VERSION 2

```
GET /products HTTP/1.1
Host: acme.com
Accept: application/vnd.com.myservice.v2+xml
```

```
HTTP/1.1 200 OK
Content-Type: application/vnd.com.myservice.v2+xml; charset=utf-8
Allow: GET, POST

<?xml version="1.0" encoding="utf-8"?>
<products xmlns="urn:com.acme.products" xmlns:xl="http://www.w3.org/1999/xlink">
  <product id="1234" xl:type="simple" xl:href="http://acme.com/products/1234">
    <name>Red Stapler</name>
    <price currency="EUR">3.14</price>
    <availability>false</availability>
  </product>
</products>
```


clients can’t upgrade protocol for known URLs!

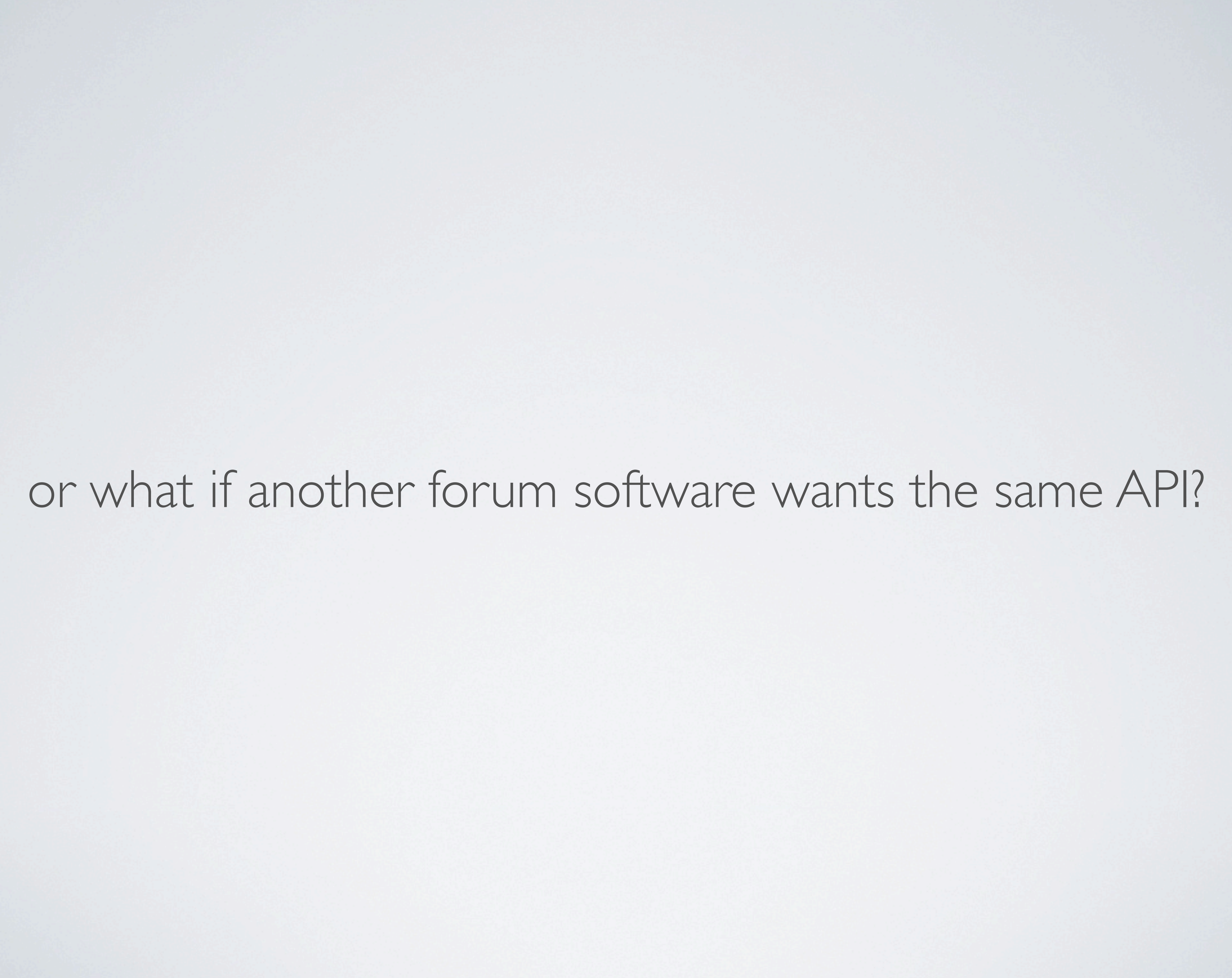

Also, imagine every install of phpBB or Drupal had an API

If the version is in the URL, clients need to regex those

http://sharksforum.org/community/api/v1/threads/102152

http://forum.sharksforum.org/api/v1/threads/102152

that would be fail


or what if another forum software wants the same API?

also would have to use “/v1/” in their URLs

URI based versioning kills interoperability

YOU MIGHT BE WONDERING

Why Exactly Is This Awesome?

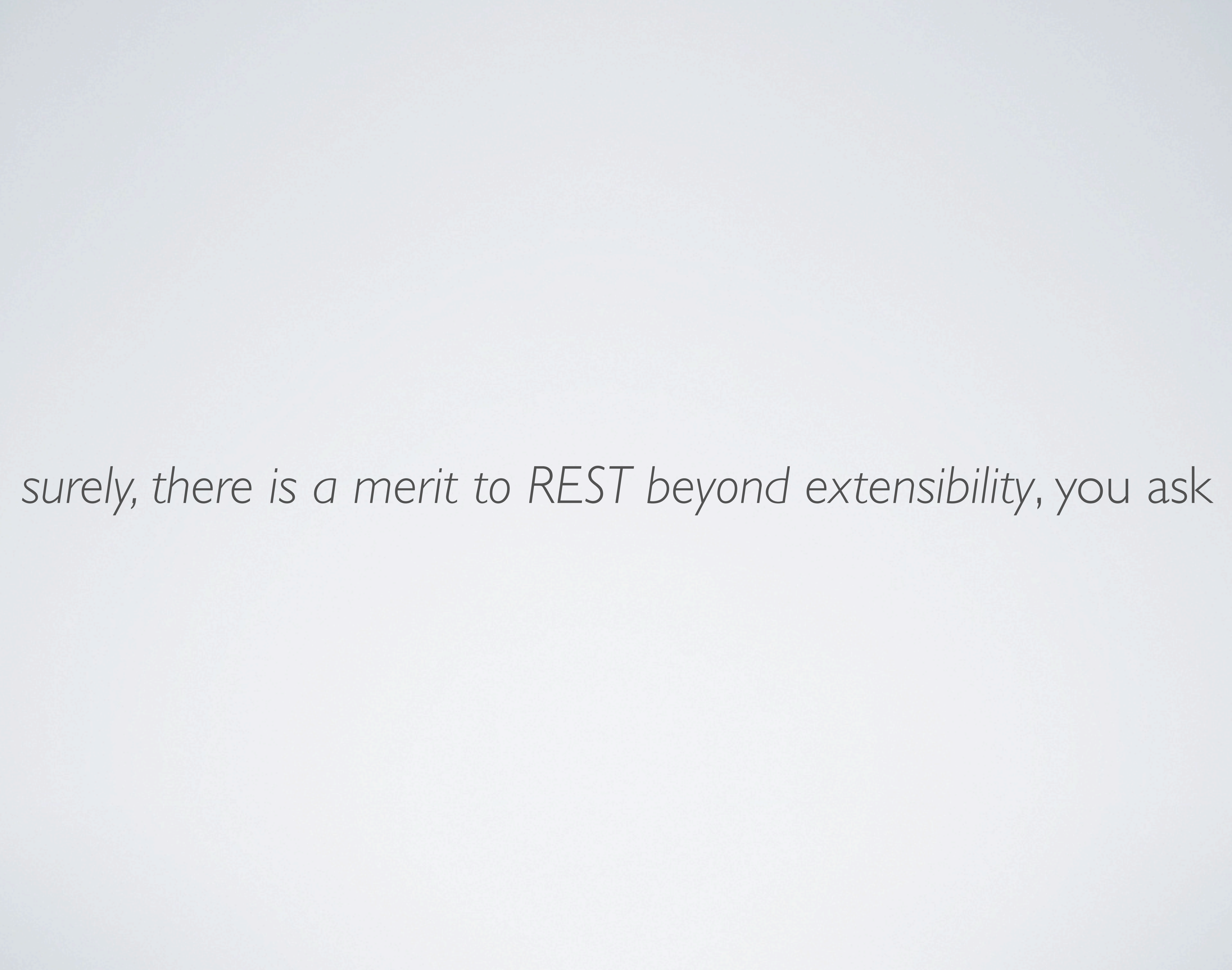
THE MERITS OF REST

- Easy to evolve: add new features or elements without breaking BC
- Easy to learn: developers can "browse" service via link rels
- Easy to scale up: grows well with number of features, users and servers
- Easy to implement: build it on top of HTTP, and profit!
  - Authentication & TLS
  - Caching & Load Balancing
  - Conditional Requests
  - Content Negotiation

but...

hold on, you say

a plain HTTP-loving service does the job, you say


surely, there is a merit to REST beyond extensibility, you ask

nope

"REST is software design on the scale of decades: every detail is intended to promote software longevity and independent evolution. Many of the constraints are directly opposed to short-term efficiency. Unfortunately, people are fairly good at short-term design, and usually awful at long-term design."

Roy Fielding

"Most of REST's constraints are focused on preserving independent evolvability over time, which is only measurable on the scale of years. Most developers simply don't care what happens to their product years after it is deployed, or at least they expect to be around to rewrite it when such change occurs."

Roy Fielding

FURTHER READING

- Ryan Tomayko
  How I Explained REST to my Wife
  http://tomayko.com/writings/rest-to-my-wife

- Jim Webber, Savas Parastatidis & Ian Robinson
  How to GET a Cup of Coffee
  http://www.infoq.com/articles/webber-rest-workflow

- Roy Thomas Fielding
  Architectural Styles and the Design of Network-based Software Architectures
  http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

BOOKS ON REST

- Jim Webber, Savas Parastatidis, Ian Robinson
  REST in Practice
  ISBN: 978-0596805821

- Subbu Allamaraju
  RESTful Web Services Cookbook
  ISBN: 978-0596801687

- Leonard Richardson, Sam Ruby
  RESTful Web Services
  ISBN: 978-0596529260

The End

Questions?

THANK YOU!

This was http://munich2012.drupal.org/node/2673

by @dzuelke

Send me questions or hire us:

david.zuelke@bitextender.com