CERN
School of Computing

Is your web API truly RESTful (and does it matter)?

Josef Hammer

CERN

Inverted CERN School of Computing, 24-25 February 2014

The Programmable Web

- **The *“human web”* is a great success story**
  - Highly scalable
  - Easy to change
  - With only the knowledge of a base URL (e.g. `www.cern.ch`) you can explore and interact with any web site
- **But APIs for machines are more difficult**
  - Hard to discover / explore: Machines do not understand the meaning of names
  - Most APIs are difficult to change once deployed
- **RESTful architectures provide a solution**

Outline

- **History**
- **Introduction to REST**
- **RESTful API Design**
  - URIs
  - HTTP
  - Hypermedia
- **Conclusion**

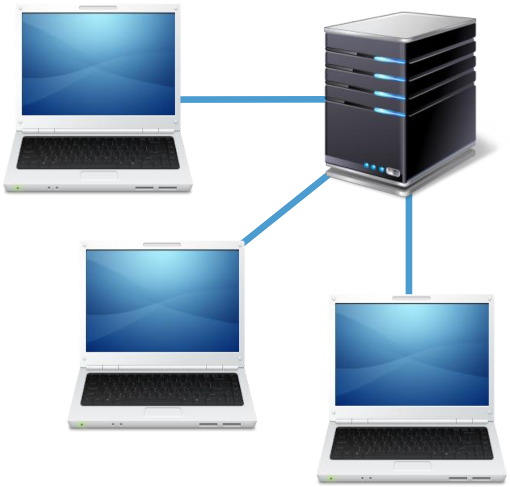
Where do we come from?

- **COM**
  - *Component Object Model*
- **CORBA**
  - *Common Object Request Broker Architecture*
- **XML-RPC**
  - *Extensible Markup Language Remote Procedure Call*
- **SOAP**
  - *Simple Object Access Protocol*
  - WSDL (Web Services Description Language)
  - Big “service document” → **tight coupling, hard to change**

Representational State Transfer (REST)

- **Term defined in Roy Fielding's dissertation in 2000** [fielding]
  - A technical description of how the World Wide Web works
- **Architectural style, not a protocol like SOAP**
  - 6 architectural constraints (“Fielding constraints”)
- **Resources + representations**
  - *“The server sends a representation describing the state of a resource. The client sends a representation describing the state it would like the resource to have. That’s representational state transfer.”* [rwa]
- **Not limited to HTTP**

Fielding Constraints (1) [fielding, rwa]



- **Client-server**
  - All communication on the web is one-to-one (vs. peer-to-peer w/ multiple sources)
- **Stateless**
  - When a client is not currently making a request, the server doesn’t know it exists.
- **Cacheable**
  - A client can save trips over the network by reusing previous responses from a cache.

Fielding Constraints (2) [fielding, rwa]

- **Layered system**
  - Intermediaries such as proxies can be invisibly inserted between client and server.
- **Code on demand** (optional)
  - The server can send executable code in addition to data. This code is automatically deployed when the client requests it, and will be automatically redeployed if it changes.
  - E.g. Javascript code in the browser

Fielding Constraints (3) [fielding, rwa]

- **The uniform interface**
  - *Identification of resources*
    - Each resource is identified by a stable URI.
  - *Manipulation of resources through representations*
    - The server describes resource state by sending representations to the client. The client manipulates resource state by sending representations to the server.
  - *Self-descriptive messages*
    - All the information necessary to understand a request or response message is contained in (or at least linked to from) the message itself.
  - *The hypermedia constraint ("HATEOAS")*
    - The server manipulates the client's state by sending a hypermedia "menu" containing options from which the client is free to choose.

HATEOAS (1)

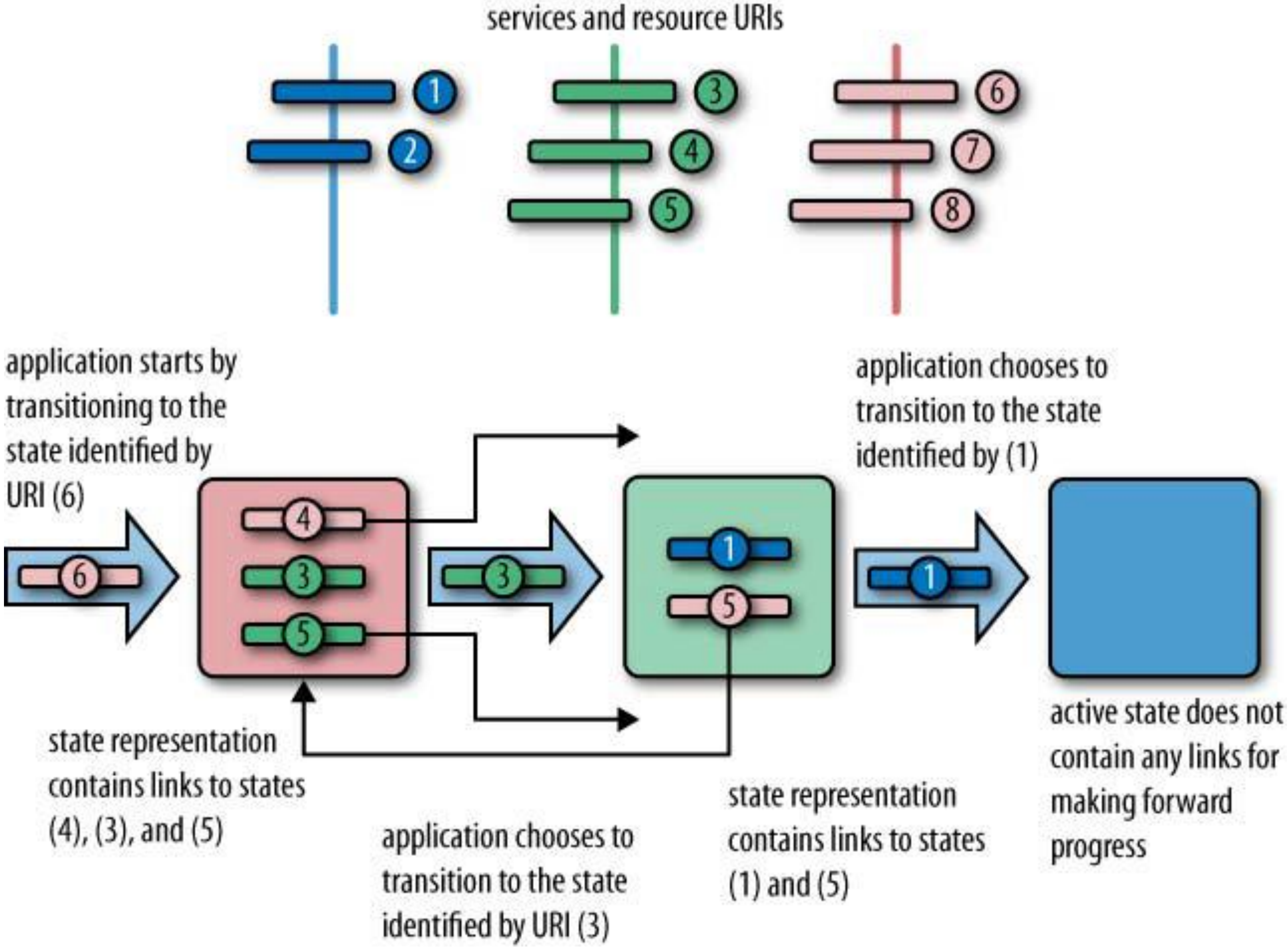
- **Hypermedia As The Engine Of Application State**
- **“Hypermedia”: Links, basically**
- *“Clients make state transitions only through actions that are dynamically identified within hypermedia by the server (e.g., by hyperlinks within hypertext). Except for simple fixed entry points to the application, a client does not assume that any particular action is available for any particular resources beyond those described in representations previously received from the server.”* [wiki-rest]

HATEOAS (2)

- *“A distributed application makes forward progress by transitioning from one state to another, just like a state machine. The difference from traditional state machines, however, is that the possible states and the transitions between them are not known in advance. Instead, as the application reaches a new state, the next possible transitions are discovered.“* [rip]

- **Clients only need to know the entry point (base URI)**
- **Clients shall not be required to construct URIs**
- **Loose coupling → easy to maintain**

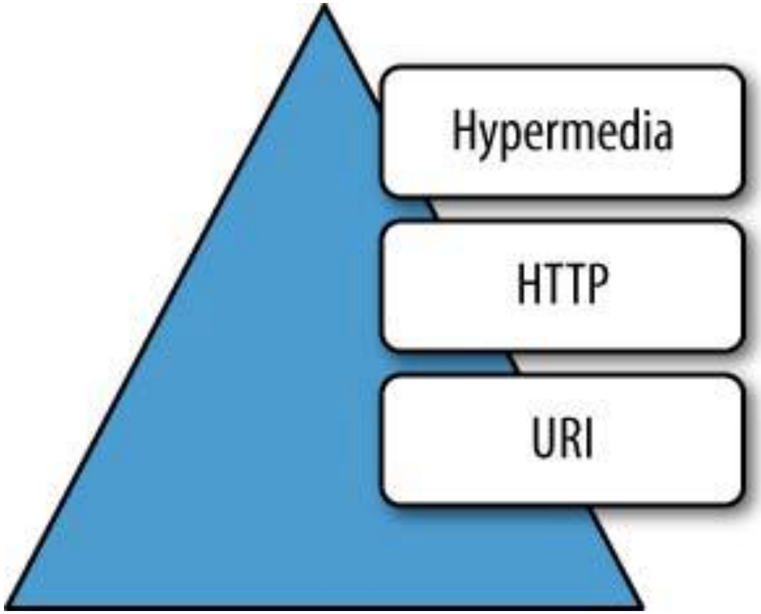
HATEOAS (3)


services and resource URIs
1
2
3
4
5
6
7
8
application starts by transitioning to the state identified by URI (6)
6
4
3
5
3
1
5
1
application chooses to transition to the state identified by (1)
state representation contains links to states (4), (3), and (5)
application chooses to transition to the state identified by URI (3)
state representation contains links to states (1) and (5)
active state does not contain any links for making forward progress

[rip]

REST Maturity Model (RMM) (1)

- **by Leonard Richardson** [rip; fowler-rmm]
  - a.k.a. Richardson Maturity Model
- **how “RESTful” is a web API?**



[rip]

REST Maturity Model (RMM) (2)

Level **3: Hypermedia controls**

- Level 2 + uses hypermedia for navigation
- *`<a href=“/slides/43” rel=“last”>`*

Level **2: HTTP methods**

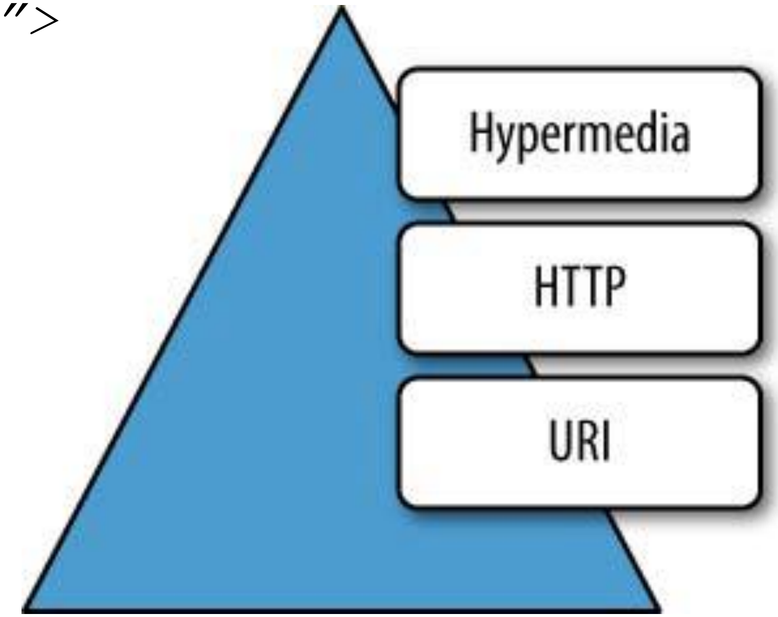
- multiple URIs, multiple HTTP methods
- *`PUT|DELETE /slides/1`*

Level **1: URIs (‘*Resources*’)**

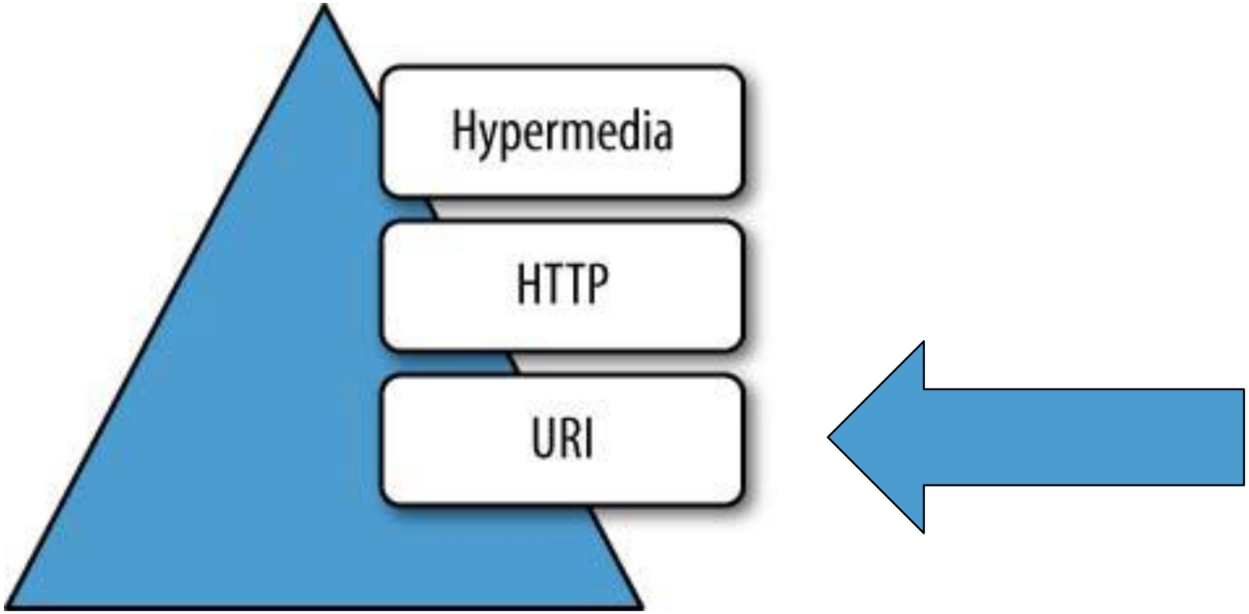
- multiple URIs, single HTTP method
- *`POST /slides/1`*

*Level **0: XML-RPC, SOAP, ...***

- single URI, single HTTP method
- *`POST /slides`*

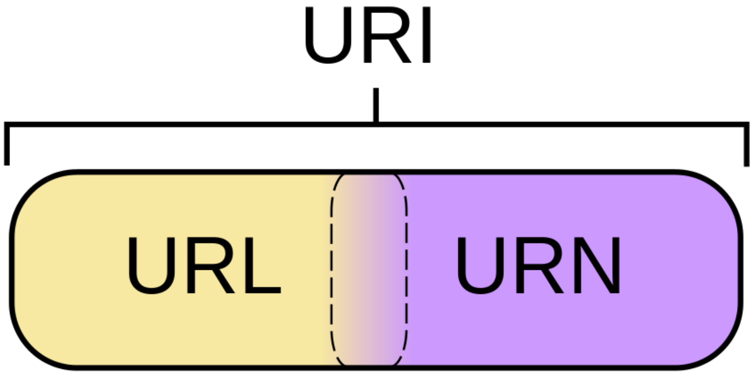


[rip]


Hypermedia
HTTP
URI

URI vs URL vs URN

- **URI: Uniform Resource Identifier**
  - A short string to identify a resource
  - Might have no representation



[uri]

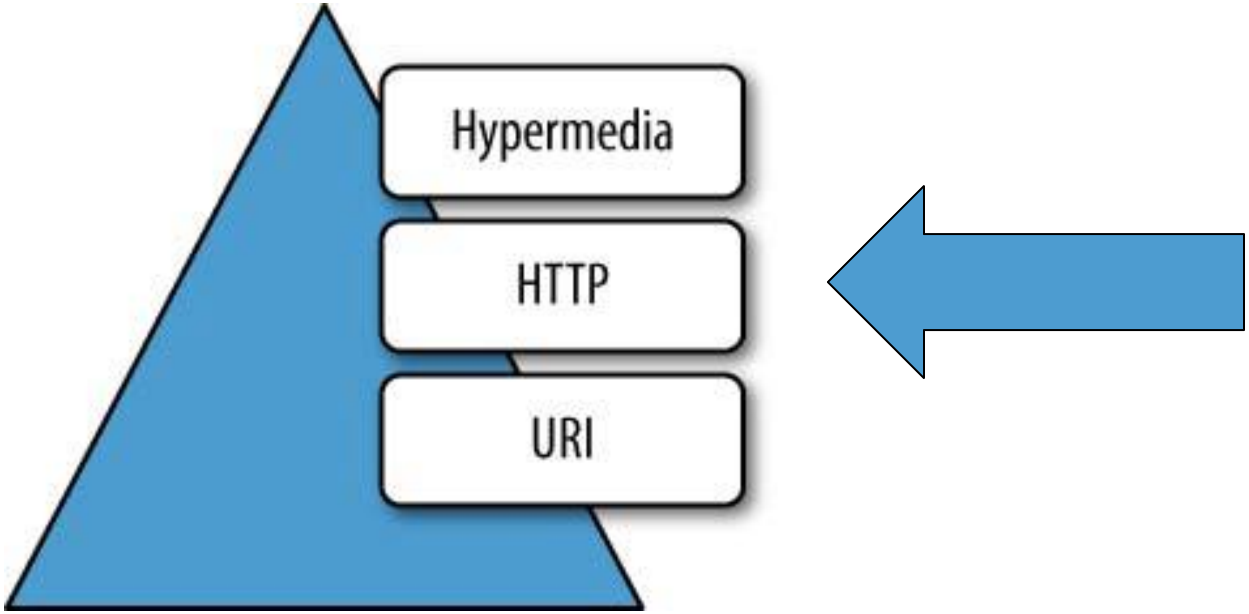
- **URL: Uniform Resource Locator**
  - A URI that can be dereferenced (= has a representation)
  - E.g. `http://www.cern.ch`
- **URN: Uniform Resource Name**
  - no protocol to dereference
  - E.g. `urn:isbn:9781449358063`

URI Design

- *"The only thing you can use an identifier for is to refer to an object. When you are not dereferencing,* ***you should not look at the contents of the URI string to gain other information."***
[Tim Berners-Lee, w3-axioms]
  - Client **code's** view: `http://cern.ch/8812ca6fa190e57b0730ea`
- *"That said, REST API designers should create URIs that* ***convey a REST API's resource model to its potential client developers."*** [rad]
  - Client **developer's** view: `http://cern.ch/events/2014/02/24/iCSC`
- *"A REST API's clients must* ***consider URIs to be the only meaningful resource identifiers****. Although other backend system identifiers (such as database IDs) may appear in a URI's path, they are meaningless to client code."* [rad]

Resource Archetypes [rad]

- **4 basic types** *(+ naming rules)*
  - **Document**
    - Single item (*noun, sg – e.g.* `/outline`)
  - **Collection**
    - Collection of items; server decides on URI (*noun, pl – e.g.* `/slides`)
    - ***Store***
      - Special kind of collection: item URIs are user-defined
  - **Controller**
    - Transactions etc. (*verb – e.g.* `/move`)
    - Try to avoid


Hypermedia
HTTP
URI

HTTP Methods („Verbs“) (1)

- The HTTP standard (RFC 2616) defines 8 methods a client can apply to a resource

- **GET**
  - Get a representation of this resource
  - **Safe + idempotent:** no side effects / state changes allowed!
  - Caching allowed

- **DELETE**
  - Destroy this resource
  - **Idempotent** (i.e. repeating the request leads to the same result / state)

HTTP Methods („Verbs“) (2)

- **PUT**
  - Replace the state of (*or create!*) this resource with the given representation
  - **Idempotent**
- **POST**
  - *POST-to-append*: Create a new resource underneath this one, based on the given representation
  - *Overloaded POST*: Trigger any state transition. Run queries with large inputs. *Do anything.*
  - **Neither safe nor idempotent** (the most generic method)

HTTP Methods („Verbs“) (3)

- **HEAD**
  - Get the headers that would be sent along with a representation of this resource, but not the representation itself. **Safe!**

- **OPTIONS**
  - Discover which HTTP methods this resource responds to

- **CONNECT**, **TRACE**
  - Used only with HTTP proxies

HTTP Methods („Verbs“) (4)

- **PATCH**
  - Extension defined in RFC 5789
  - Modify *part* of the state of this resource
- ***LINK*** *(draft)*
  - Connect some other resource to this one
- ***UNLINK*** *(draft)*
  - Destroy the connection between some other resource and this one

CRUD

- **Create, Read, Update, Delete**
  - everything you need for collections ☺
- **Maps perfectly well to HTTP verbs**
  - Create → POST (collection), PUT (store)
  - Read → GET
  - Update → PUT
  - Delete → DELETE
- **Rest Maturity Model Level 2**
  - does not fit everything (limited vocabulary)
  - shared, tightly coupled understanding of resource life

Requests: Good, Bad, or Evil? (1)

- `GET /deleteUser?id=1234`

  Evil! GET *must not* modify the resource state!

- `GET /deleteUser/1234`

  Certainly looks better ;) … nevertheless just as **evil!**

- `DELETE /deleteUser/1234`

  Method name in URI … **bad.**

- `POST /users/1234/delete`

  Why use a controller when there is a standard method? **Bad.**

- `DELETE /users/1234`

  ☺

Requests: Good, Bad, or Evil? (2)

- **`GET /users/register`**

  Assuming "register" means creating a new user:
  Might make sense for a human client (web site).
  In an API: **Bad.** Retrieve a template with `GET /users` if necessary.

- **`POST /users/register`**

  No need to use a controller for creating a resource … **bad.**

- **`POST /users`**

  ☺

- **`PUT /users`**

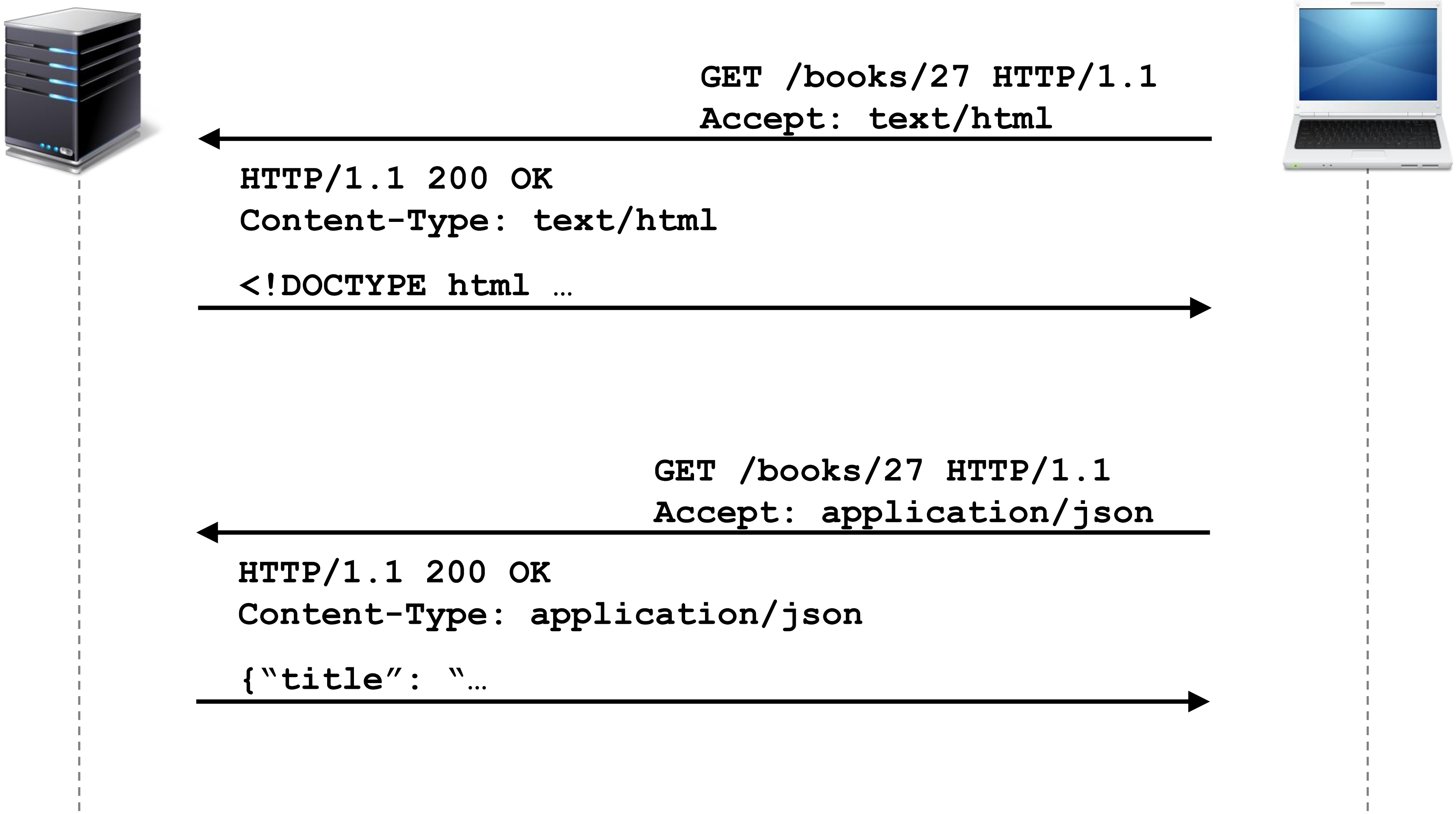
  If you really want to replace/update your entire user database ;)

- **`PUT /users/jhammer`**

Content Negotiation (1)

- **A single resource may have many representations**
  - Clients can request a specific one with the `Accept*` headers
- ***Media Type***
  - `Accept: application/json`
  - Syntax: `type "/" subtype *( ";" parameter )`
  - Type::= **`application`**`|audio|image|message|model|multipart|text|video`
- ***Language***
  - `Accept-Language: en, de; q=0.5, fr; q=0.1`

Content Negotiation (2)



```
GET /books/27 HTTP/1.1
Accept: text/html
```

```
HTTP/1.1 200 OK
Content-Type: text/html

<!DOCTYPE html …
```

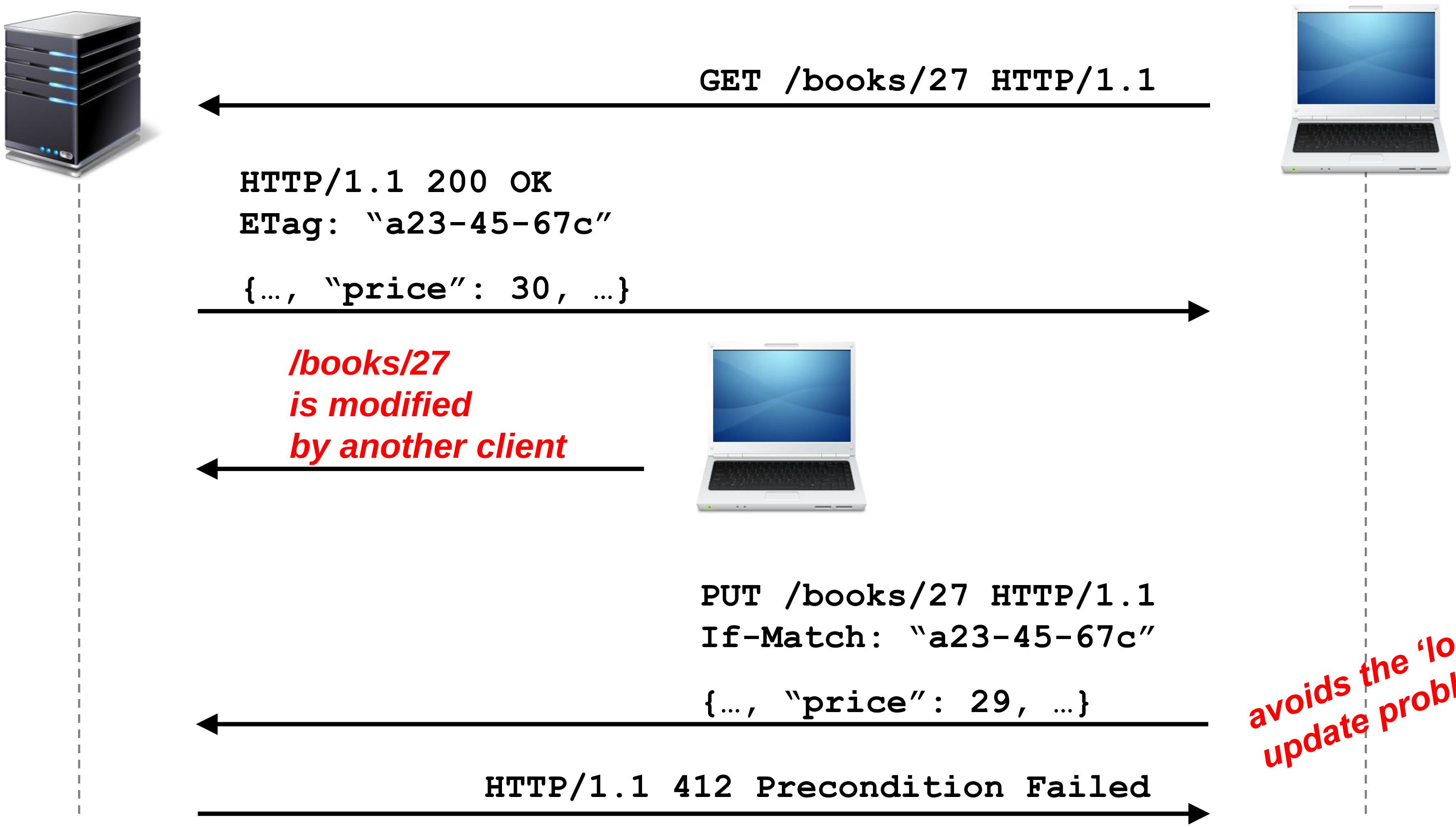
```
GET /books/27 HTTP/1.1
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{“title”: “…
```

Conditional Requests (1)

- **Server sends `ETag` header** (“entity tag”; MD5 or Seq# or …)
  - `ETag: “a23-45-67c”`
- **Client uses this value to send a conditional request**
  - GET only if modified:
    - `If-None-Match: “a23-45-67c”`
    - Result: `304 (Not Modified)`
  - PUT only if NOT modified (since last GET):
    - `If-Match: “a23-45-67c”`
    - Result: `412 (Precondition Failed)`
- **Less reliable: `Last-Modified` (timestamp; 1s resolution)**
  - Client: `If-Modified-Since`, `If-Unmodified-Since`

Conditional Requests (2)



```
GET /books/27 HTTP/1.1
```

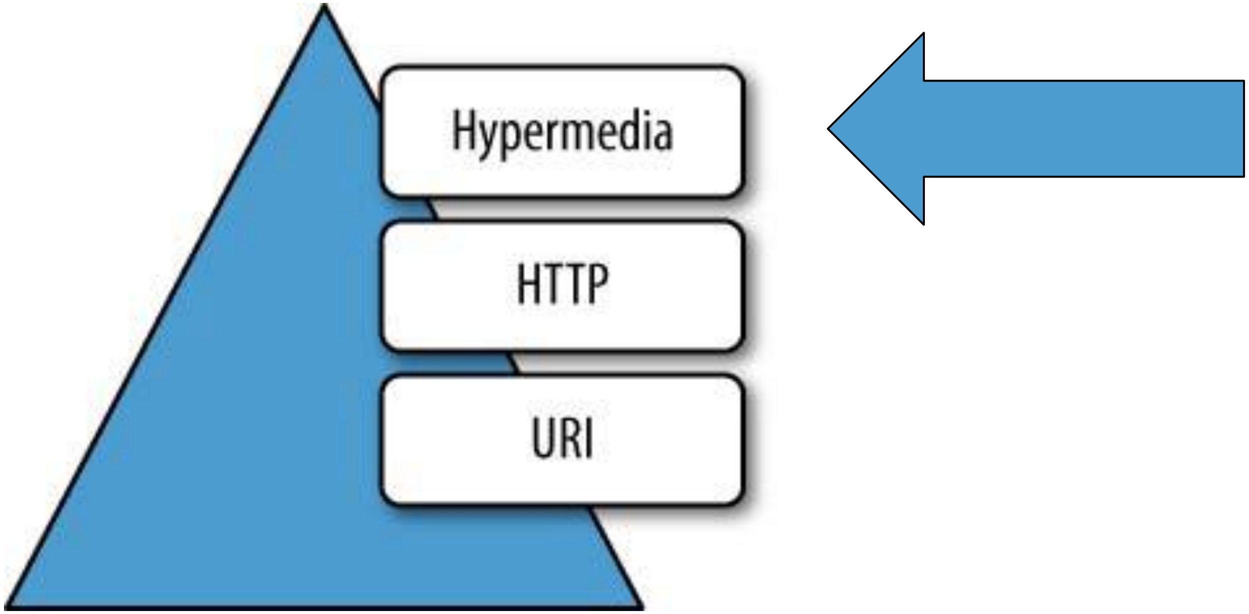
```
HTTP/1.1 200 OK
ETag: "a23-45-67c"

{…, "price": 30, …}
```

/books/27
is modified
by another client

```
PUT /books/27 HTTP/1.1
If-Match: "a23-45-67c"

{…, "price": 29, …}
```

```
HTTP/1.1 412 Precondition Failed
```

avoids the 'lost update problem'


Hypermedia
HTTP
URI

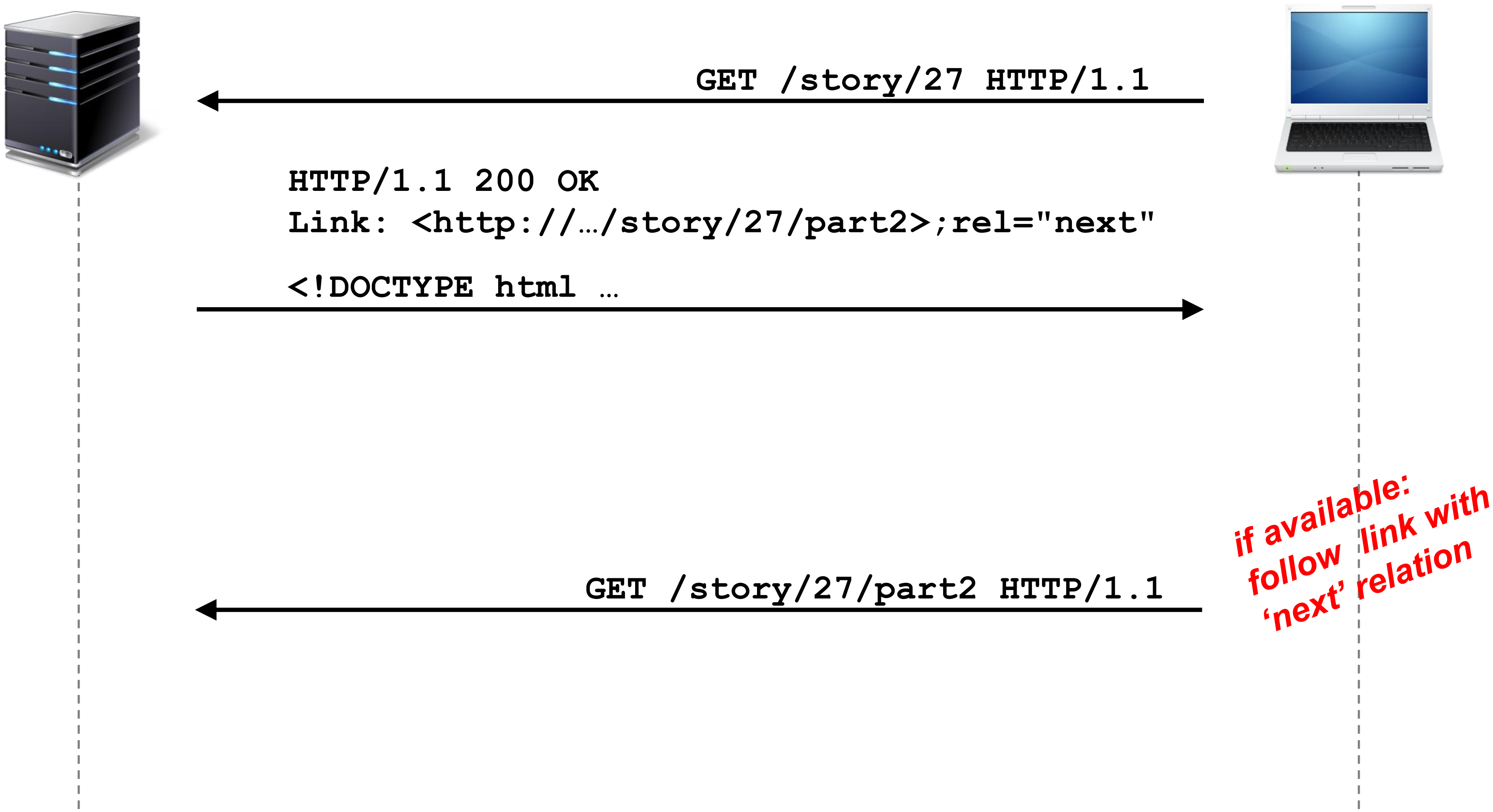
Hypermedia

- *“Hypermedia is the general term for things like HTML links and forms: the techniques a server uses to explain to a client what it can do next.”* [rwa]
  - E.g. the **`<a>`** tag is a simple *hypermedia control*
- **Works well for human clients**
  - We simply follow links labelled “Add to Cart”, “Sign In”, …
- **… but how can we tell machines the semantic meaning of these links?**

Link Relations (1)

- **Links in many data formats allow the `rel` attribute**
  - Relation between the linked resource and the current one
- **E.g. in HTML**
  - `<link rel="stylesheet" type="text/css" href="/style.css"/>`
  - Tells browsers to automatically retrieve `/style.css` and use it to style the current page
- **Communicate the “meaning” of a link to the client**
  - Clients can interpret the relation and choose the right link

Link Relations (2)



```
GET /story/27 HTTP/1.1
```

```
HTTP/1.1 200 OK
Link: <http://…/story/27/part2>;rel="next"

<!DOCTYPE html …
```

```
GET /story/27/part2 HTTP/1.1
```

if available: follow link with 'next' relation

Link Relations (3)

- **Link relations mean nothing without a formal definition**

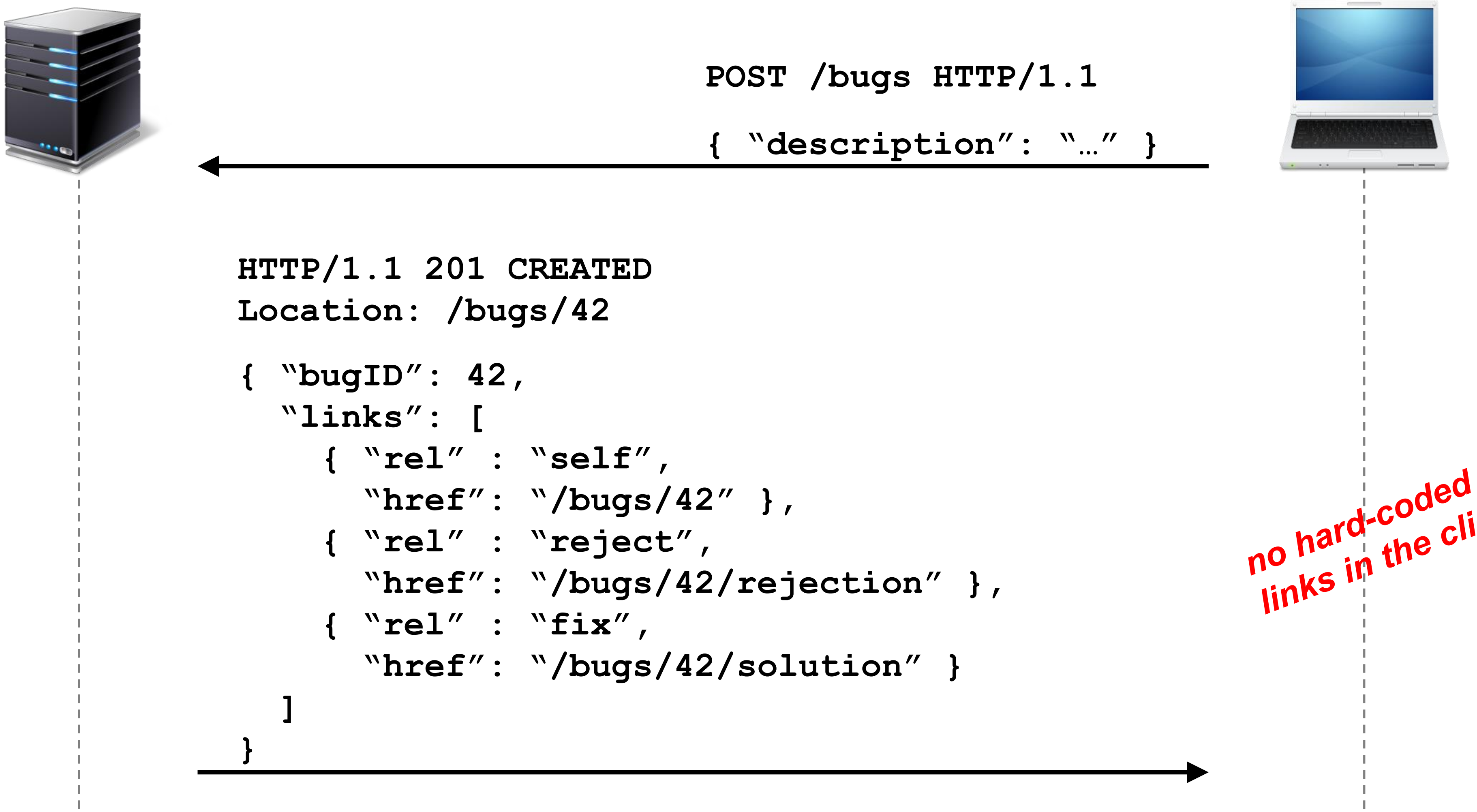
- **RFC 5988 defines 2 types**
  - *Registered link relations*
    - E.g. IANA (Internet Assigned Numbers Authority) manages a registry
    - E.g. `self`, `next`, `previous`
  - *Extension relations*
    - Like URLs – you are allowed to define anything within your domain
    - E.g. `http://josefhammer.com/toc`

Evolvable APIs (1)

- **Decoupling the client from the server**
  - Use link relations instead of hard-coded / constructed links
  - Choose from the set of provided links only

- **… allows APIs to evolve**
  - URIs can be changed
    → only the relation is hard-coded
  - Features can be added
    → old versions of the client will ignore unknown links
  - Features can be removed
    → clients gracefully ignore missing links

Evolvable APIs (2)

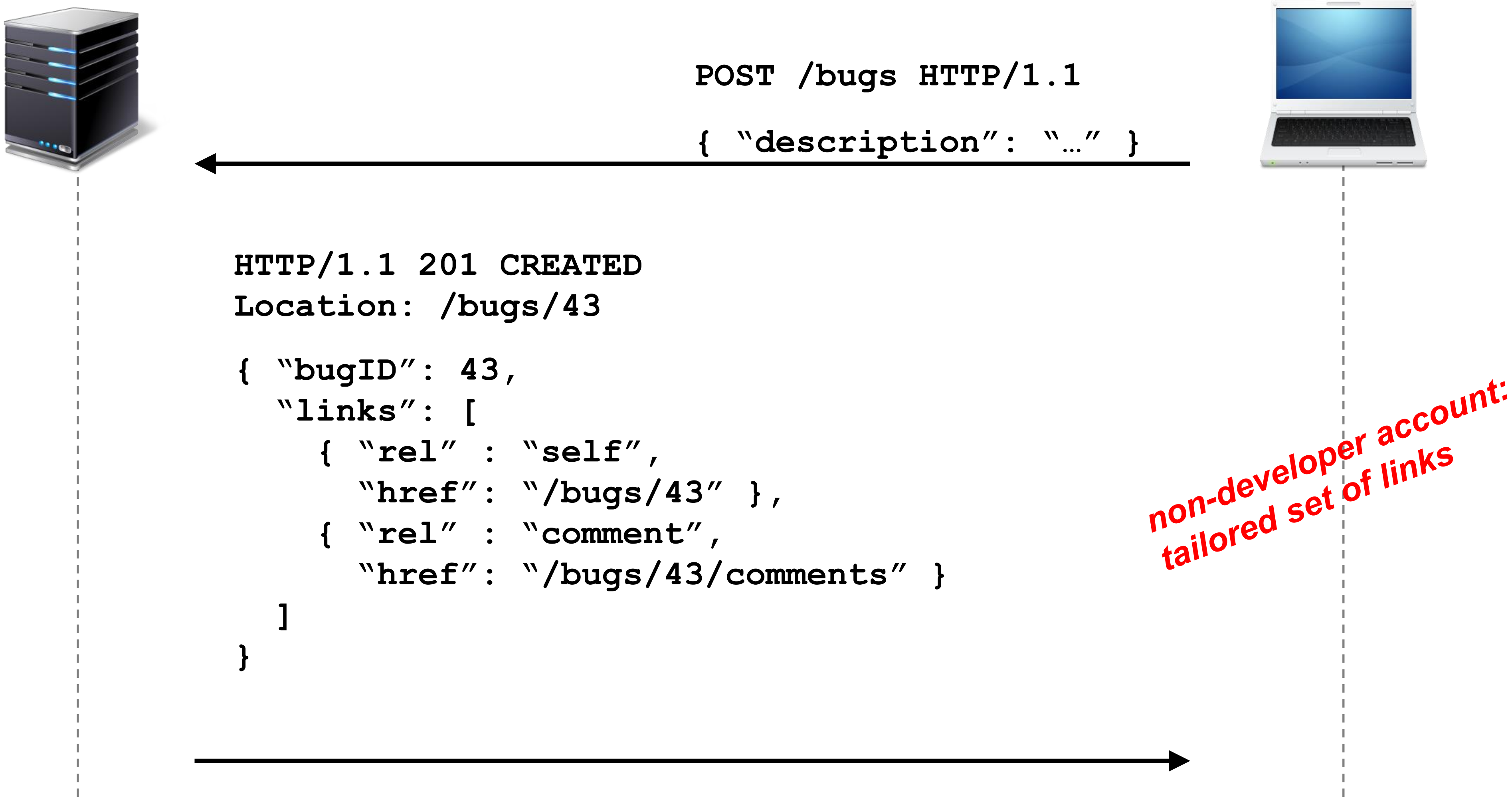


```
POST /bugs HTTP/1.1

{ “description”: “…” }
```

```
HTTP/1.1 201 CREATED
Location: /bugs/42

{ “bugID”: 42,
  “links”: [
    { “rel” : “self”,
      “href”: “/bugs/42” },
    { “rel” : “reject”,
      “href”: “/bugs/42/rejection” },
    { “rel” : “fix”,
      “href”: “/bugs/42/solution” }
  ]
}
```

no hard-coded links in the client

Evolvable APIs (3)

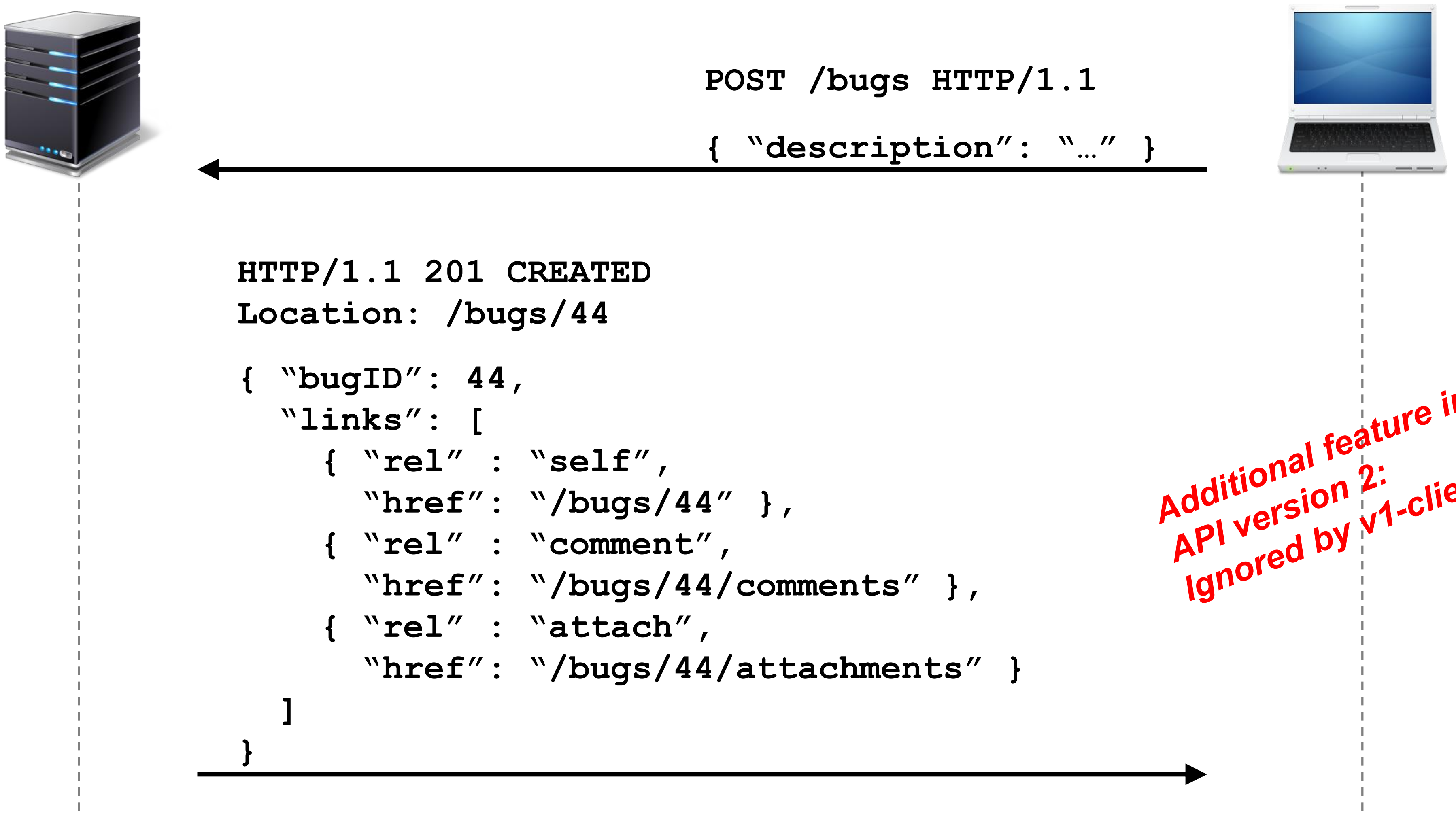


```
POST /bugs HTTP/1.1

{ “description”: “…” }
```

```
HTTP/1.1 201 CREATED
Location: /bugs/43

{ “bugID”: 43,
  “links”: [
    { “rel” : “self”,
      “href”: “/bugs/43” },
    { “rel” : “comment”,
      “href”: “/bugs/43/comments” }
  ]
}
```

non-developer account:
tailored set of links

Evolvable APIs (4)



```
POST /bugs HTTP/1.1

{ “description”: “…” }
```

```
HTTP/1.1 201 CREATED
Location: /bugs/44

{ “bugID”: 44,
  “links”: [
    { “rel” : “self”,
      “href”: “/bugs/44” },
    { “rel” : “comment”,
      “href”: “/bugs/44/comments” },
    { “rel” : “attach”,
      “href”: “/bugs/44/attachments” }
  ]
}
```

Additional feature in API version 2:
Ignored by v1-clients

Domain specific data formats

- **Try to exploit existing domain specific data formats**
  - Atom, AtomPub
  - OData
  - Collection+JSON
  - OpenSearch
  - …
  - Microformats
  - HTML Microdata

→ Client tools may exist

→ Developers more likely to be familiar with the terms

Microformats

- **E.g. the hcard microformat** [hcard]

  -
    ```
    <div class="vcard">
       <span class="n">
          <span class="given-name">Josef</span>
          <span class="family-name">Hammer</span>
       </span>
    </div>
    ```

- **Well-defined and -understood terms**

- **Easy to embed in HTML**

- **microformats.org** provides a collection of schemata

Microdata

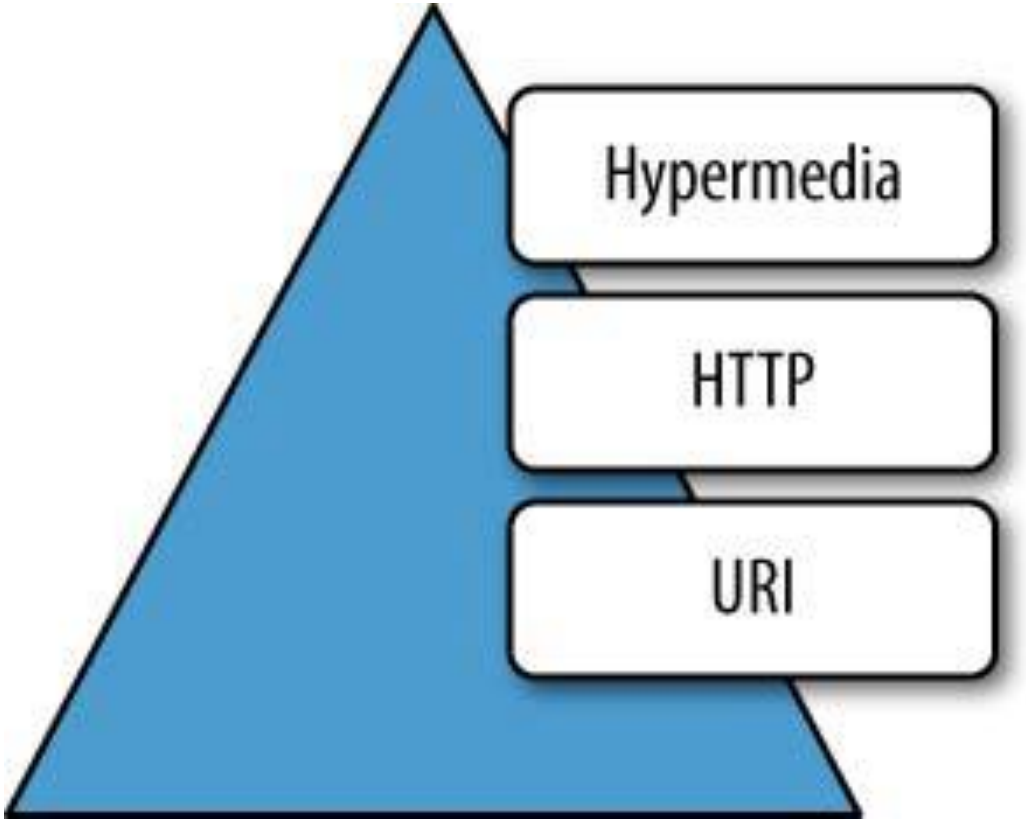
- **A refinement of the microformat concept for HTML 5**
- **5 new attributes** for *any* HTML tag

`itemscope`	Starts a new scope (boolean)
`itemprop`	Like **`class`** in HTML
`itemtype`	Where to find the type definition
`itemid`	Global identifier (valid URL)
`itemref`	List of itemIDs

- **schema.org** provides a collection of schemata

Conclusion

- **Yes, it does matter → strive for the highest level**



<a href="/slides/43" ***rel="last"****>*

Loose coupling – easier to change

GET | POST | PUT | DELETE | …

☹ GET /deleteUser/1234

☺ DELETE /users/1234

/slides/outline/move

Collection | Document | Controller

References

- **fielding**: Architectural Styles and the Design of Network-based Software Architectures. Roy Thomas Fielding; Doctoral dissertation, University of California, Irvine, 2000; http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
- **fowler-rmm**: http://martinfowler.com/articles/richardsonMaturityModel.html
- **hcard**: http://microformats.org/wiki/hcard
- **rad**: REST API Design Rulebook. Mark Masse; O'Reilly, October 2011
- **rip**: REST in Practice. Jim Webber, Savas Parastatidis, Ian Robinson; O'Reilly, September 2010
- **rwa**: RESTful Web APIs. Leonard Richardson, Mike Amundsen, Sam Ruby; O’Reilly, September 2013
- **rwc**: RESTful Web Services Cookbook. Subbu Allamaraju; O’Reilly, March 2010
- **rws**: RESTful Web Services. Leonard Richardson, Sam Ruby; O’Reilly, May 2007
- **uri**: http://en.wikipedia.org/wiki/File:URI_Euler_Diagram_no_lone_URIs.svg
- **w3-axioms**: http://www.w3.org/DesignIssues/Axioms.html
- **waa**: Designing Evolvable Web APIs with ASP.NET. Glenn Block, Pablo Cibraro, Pedro Felix, Howard Dierking, Darrel Miller; O’Reilly, March 2014 (est.; early release March 2013)
- **wiki-rest**: http://en.wikipedia.org/wiki/Representational_state_transfer