INF5750

RESTful Web Services

Recording

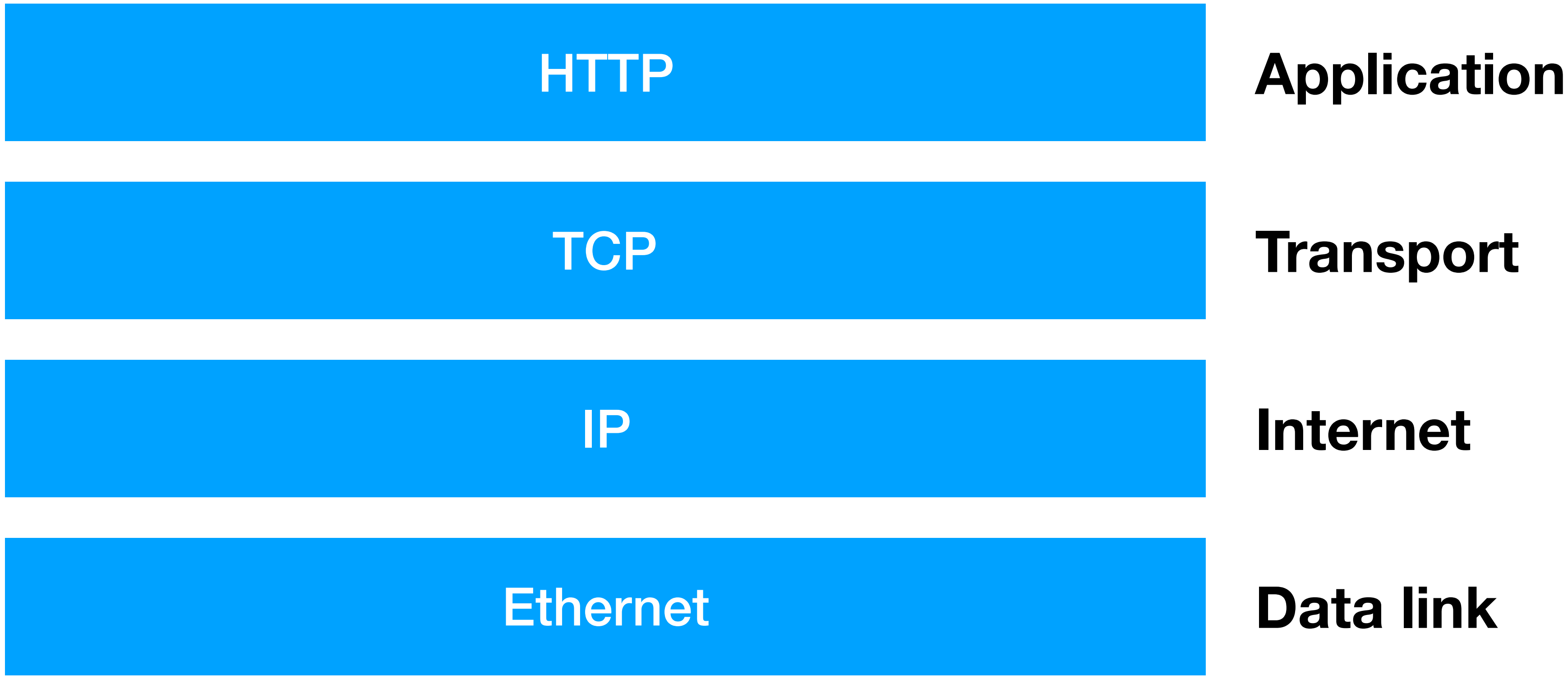
- Audio from the lecture will be recorded!
- Will be put online *if* quality turns out OK

Outline

- REST
- HTTP
- RESTful web services

HTTP

- Hypertext Transfer Protocol
- Application layer protocol - foundation for data communication on the Web



HTTP **Application**

TCP **Transport**

IP **Internet**

Ethernet **Data link**

HTTP development

- Work on HTTP protocol started in 1989
- HTTP/1.1 first released in 1997 - updated since
- Protocol for the Web, which meant it needed:
  - Low entry-barrier to enable adoption => simple
  - Preparedness for change over time => extensible
  - Usability of hypermedia (links) => minimal network interactions
  - Deployed on internet scale => built for unexpected load and network changes

REST

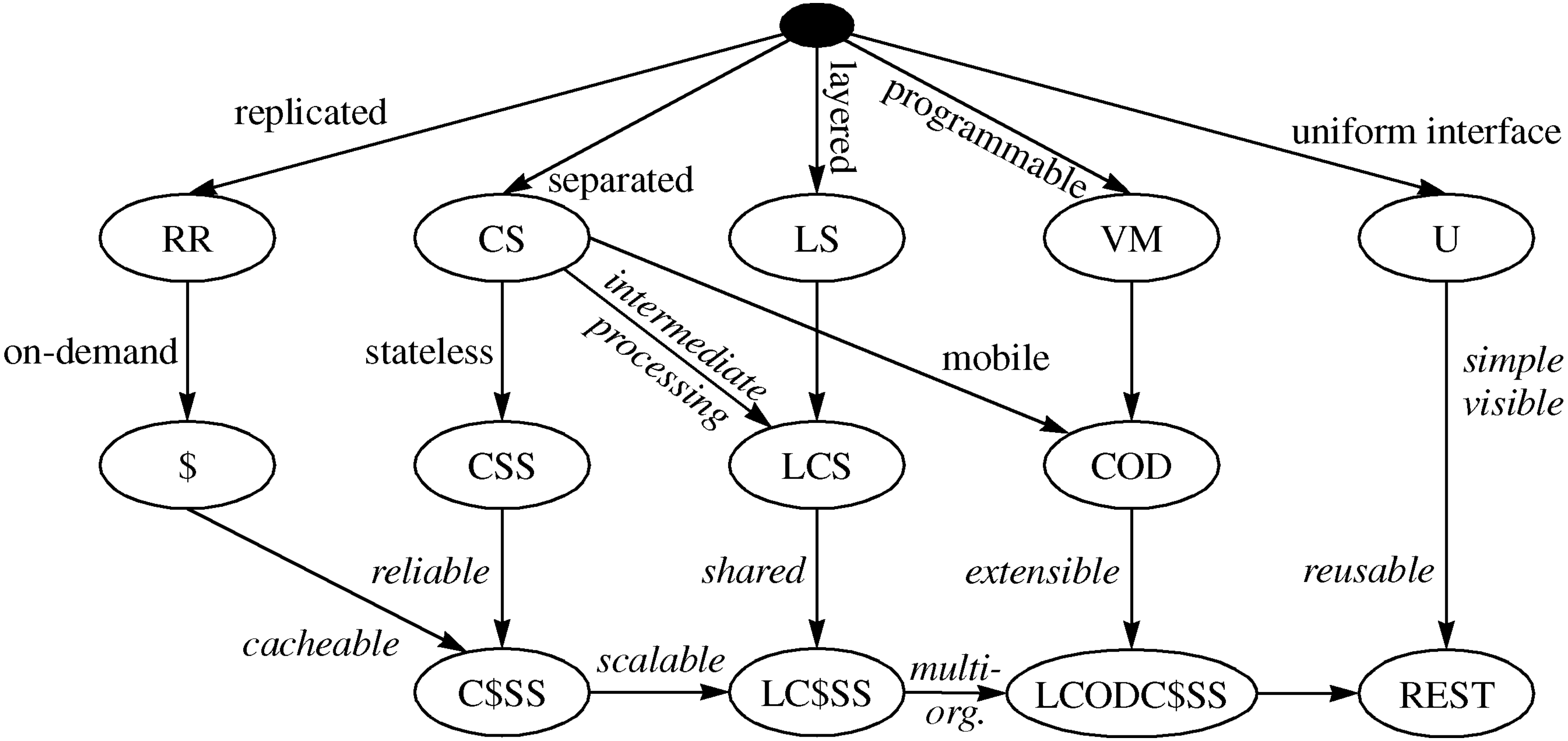
"We needed a model for how [the Web] *should* work. This idealised model of the interactions within an overall Web application - [the REST] architectural style - became the foundation for the modern Web architecture[…]"

- Roy Fielding

REpresentational **S**tate **T**ransfer

- REST is an *architectural style*
- Defined by a set of *architectural constraints*
- These guided the development of HTTP
- HTTP is a standard, REST is not

REST Architectural constraints



Source: Fielding and Taylor (2002)

Client-Server

- Client-server architecture
- Separation of concerns - interface from data storage

+ Simplifies the server component

+ Components can evolve separately

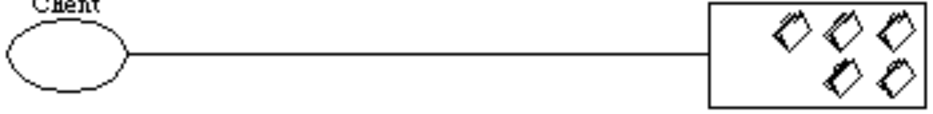


Figure 5-2. Client-Server

Statelessness

- Communication must be stateless:
  - Each request must be self-descriptive
  - Session state is kept by client

+ Improves visibility, reliability and scalability

- Decrease network performance due to repetitive data

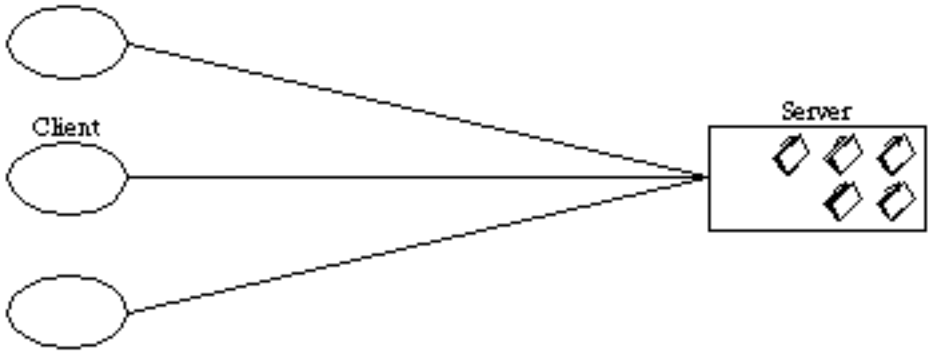


Figure 5-3. Client-Stateless-Server

Cacheable

- Clients and intermediaries can cache response
- Data within a response must be labeled cacheable (or not)

+ Improves network performance and reduces interaction

- Can decrease reliability

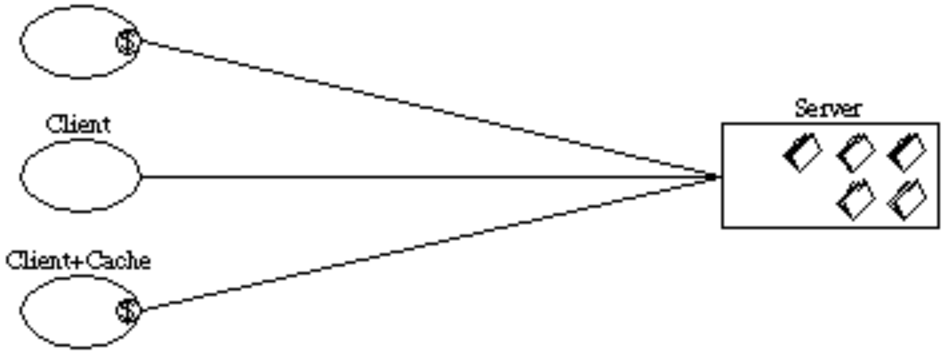


Figure 5-4. Client-Cache-Stateless-Server

Uniform interface

- There is a uniform interface for interacting with resources
- Four interface constraints:
  - identification of resources
  - manipulation of resources through representations
  - self-descriptive messages
  - hypermedia as the engine of application states

Uniform Interface

+ Decouples implementations from services that are provided

+ Can decrease efficiency - information is transferred in a standard format rather than optimised to the application

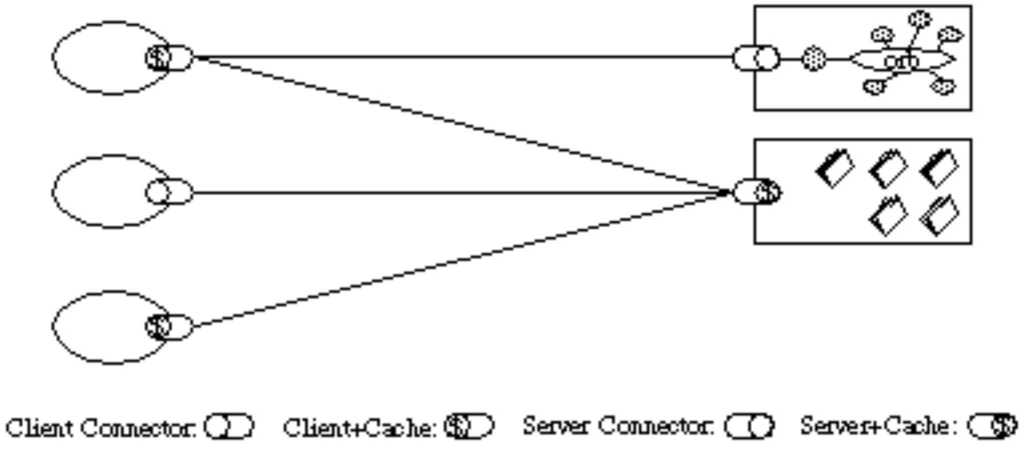


Figure 5-6. Uniform-Client-Cache-Stateless-Server

Layered system

- The architecture can consist of hierarchical levels
- Components only communicate with their "neighbours"

+ Reduce system complexity
+ Intermediaries can improve efficiency

- Add overhead and latency

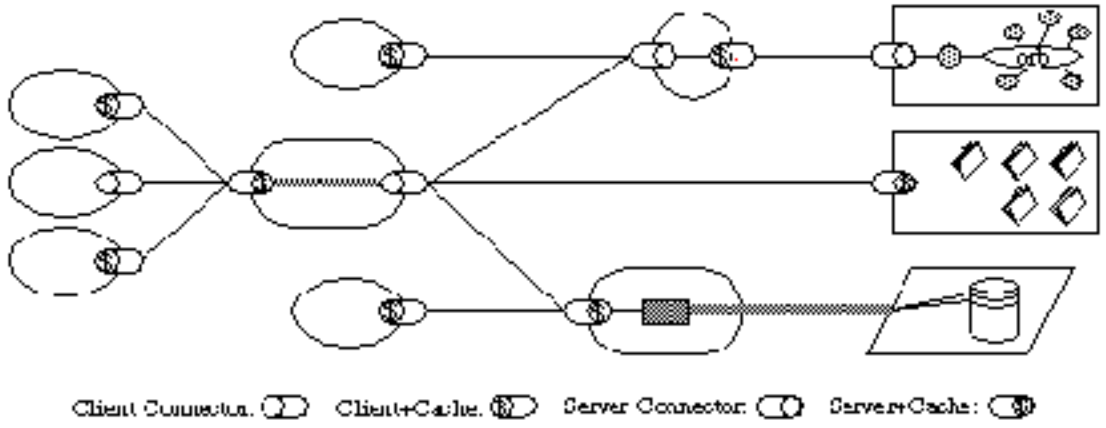


Figure 5-7. Uniform-Layered-Client-Cache-Stateless-Server

Code-on-demand

- Clients can download and execute code to extend functionality

+ Simplifies clients and improves extensibility

- Reduces visibility

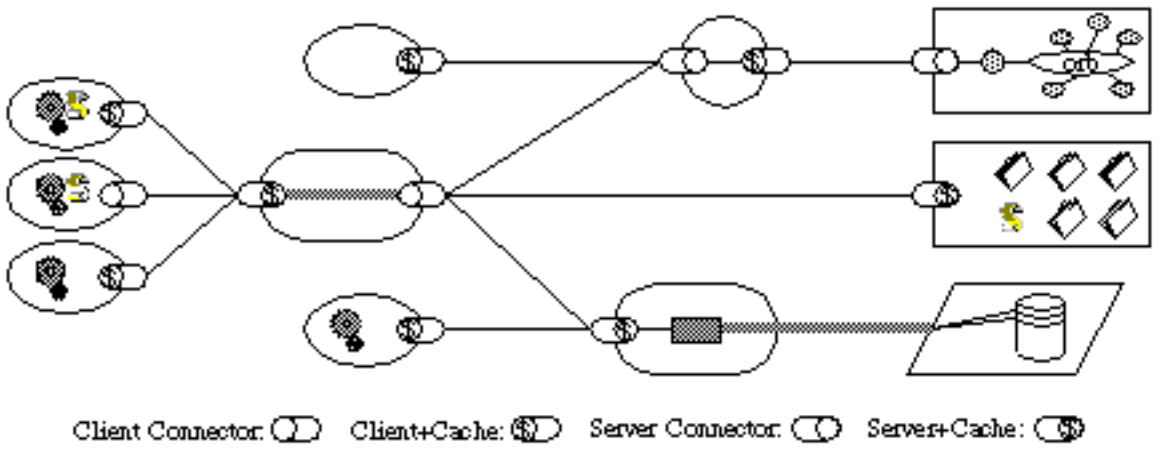


Figure 5-8. REST

REST constraints

- *Addressability* - all resources have a unique and stable identifier
- *Uniform interface -* a uniform interface with a small set of standard methods support all interactions
- *Stateless interactions* - each session is for a single interaction, and session state is not stored by server
- *Self-describing messages* - interaction happens though requests and response message that contain both data and metadata
- *Hypermedia* - resources include links to related resources, enabling decentralised discovery

REST architectural elements

- Data elements
- Components
- Connectors

REST data elements

Data element	Example
resource	link to Web service
resource identifier	URL
representation	HTML document, XML document, image file
representation metadata	media type, last-modified
resource metadata	source link, alternates
control data	cache-control

Resources

- Resources are the key information elements in REST
- Any information that can be named can be a resource - image, service, document
- Resources refer to conceptual mappings, not particular entities or values
- Abstract definition of resources enables:
    - generality - information is not divided by type, implementation
    - late binding to representation - representation (format) can be decided based on request
    - we can refer/link to (persistent) concepts rather than specific instances of a concept

Resource identifiers

- Each resource needs an identifier
- Identifier is defined by the "author" of the resource, not centralised

Representations

- *Resources* are not transferred between components in the architecture, but *representations* of resources

- Representations consists of both data and metadata describing the data

- Resource metadata provide information about the resource not specific to the representation

- Control data provides information about the message, such as for caching

REST components

Component	Example
origin server	apache, MS IIS
gateway/reverse proxy	squid, cgi, nginx
proxy	
user agent	Chrome, Firefox, curl

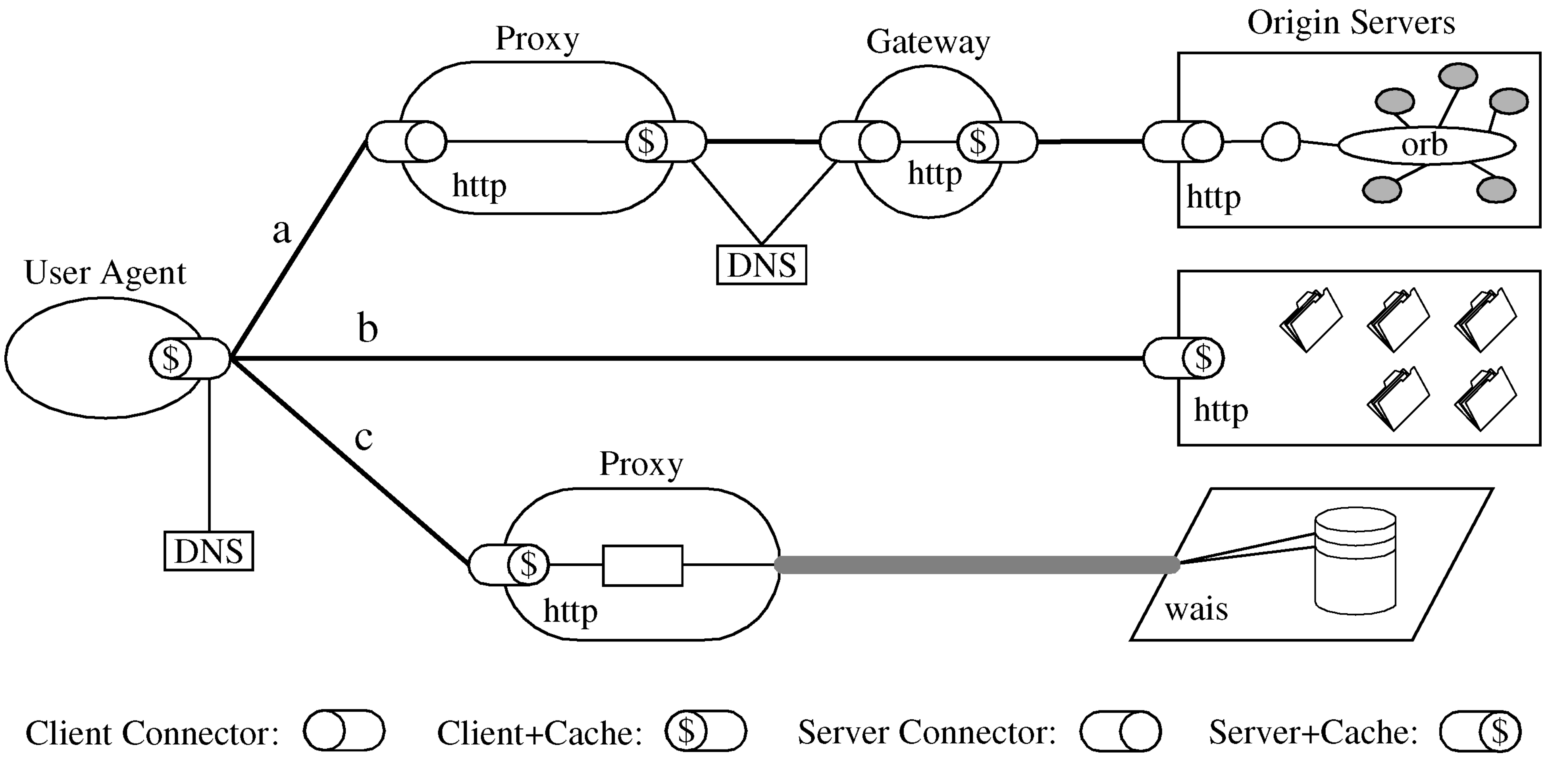
REST connectors

Connector	Example
client	libwww, libcurl
server	libwww, Apache API
cache	browser, cache networks
resolver	bind
tunnel	SOCKS

REST connectors

- Connectors handles communication for the components
- Because interactions are stateless and requests self-descriptive:
    - Connectors can handle requests independently and in parallel
    - Intermediaries can understand requests in isolation
    - Information relevant for caching is part of each request

REST process



Source: Fielding and Taylor (2002)

HTTP in practice

- Anatomy of HTTP requests and responses
- HTTP methods
- Content negotiations
- Status codes

HTTP requests

- HTTP requests consists of header and body
- Body - the data/payload
- Header - different types:
    - General header that can apply to both request and response - Date, Cache-Control
    - Request header - Accept, User-Agent, Referer
    - Response header - Age, Location, Server
    - Entity header is metadata about the body (MIME, content length etc)

```
~>curl google.com -v
* Rebuilt URL to: google.com/
*   Trying 216.58.209.142...
* TCP_NODELAY set
* Connected to google.com (216.58.209.142) port 80 (#0)
> GET / HTTP/1.1
> Host: google.com
> User-Agent: curl/7.54.0
> Accept: */*
>
< HTTP/1.1 302 Found
< Cache-Control: private
< Content-Type: text/html; charset=UTF-8
< Referrer-Policy: no-referrer
< Location: http://www.google.no/?gfe_rd=cr&dcr=0&ei=mEu4WbXAL4ir8we1o4a4Dg
< Content-Length: 268
< Date: Tue, 12 Sep 2017 21:03:20 GMT
<
<HTML><HEAD><meta http-equiv="content-type" content="text/html;charset=utf-8">
<TITLE>302 Moved</TITLE></HEAD><BODY>
<H1>302 Moved</H1>
The document has moved
<A HREF="http://www.google.no/?
gfe_rd=cr&amp;dcr=0&amp;ei=mEu4WbXAL4ir8we1o4a4Dg">here</A>.
</BODY></HTML>
* Connection #0 to host google.com left intact
```

Request header

Response header

Response body

```
curl -X PATCH "https://play.dhis2.org/demo/api/dataElements/FTRrcoaog83" -u admin:district -H "Content-
type: application/json" -d '{"domainType": "BLABLA"}' -vv
*   Trying 52.30.174.183...
* TCP_NODELAY set
* Connected to play.dhis2.org (52.30.174.183) port 443 (#0)
* TLS 1.2 connection using TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
* Server certificate: play.dhis2.org
* Server certificate: RapidSSL SHA256 CA - G3
* Server certificate: GeoTrust Global CA
* Server auth using Basic with user 'admin'
> PATCH /demo/api/dataElements/FTRrcoaog83 HTTP/1.1
> Host: play.dhis2.org
> Authorization: Basic YWRtaW46ZGlzdHJpY3Q=
> User-Agent: curl/7.54.0
> Accept: */*
> Content-type: application/json
> Content-Length: 24
>
* upload completely sent off: 24 out of 24 bytes
< HTTP/1.1 500 Internal Server Error
< Server: nginx/1.4.6 (Ubuntu)
< Date: Tue, 12 Sep 2017 21:15:09 GMT
< Content-Type: application/json;charset=UTF-8
< Content-Length: 408
< Connection: keep-alive
< X-XSS-Protection: 1; mode=block
< X-Frame-Options: SAMEORIGIN
< X-Content-Type-Options: nosniff
< Set-Cookie: JSESSIONID=62886259EE13F8F9A3A9BFFAAA5E8077; Path=/demo/; HttpOnly
< Cache-Control: no-cache, private
<
* Connection #0 to host play.dhis2.org left intact
{"httpStatus":"Internal Server Error","htt[illegible]tus":"ERROR","message":"Can not construct
instance of org.hisp.dhis.dataelement.Data[illegible]ring value (\"BLABLA\"): value not one of
declared Enum instance names: [TRACKER, AGGREGATE]\n at [Source: {\"domainType\": \"BLABLA\"}; line: 1,
column: 16] (through reference chain: org.hisp.dhis.dataelement.DataElement[\"domainType\"])"}
```

Request header

Response header

Response body

HTTP methods

- GET - request representation of a resource
- POST - create an entity based on the payload (body)
- PUT - update an entity based on the payload
- PATCH - partially update an entity based on the payload
- DELETE - delete the resource
- HEAD, TRACE, OPTIONS, CONNECT

HTTP methods

- GET - safe, idempotent, cacheable
- POST
- PUT - idempotent
- PATCH - *can* be idempotent
- DELETE - idempotent
- Idempotent methods can be called multiple times without changing the result/outcome

Content negotiation

- Content negotiation is the process of determining the *representation* of the resource
- Clients specify desired representation through:
  - HTTP header **Accept** field - Accept: application/json
  - URL extension - http://localhost/api/cars**.json**
- If the requested representation is not available the server should:
  - Respond with status code 406 not acceptable
  - Include a list of available representations

HTTP status codes

- HTTP status codes are divided into classes:
  - 1XX - informational
  - 2XX - success
  - 3XX - redirection
  - 4XX - client error
  - 5XX - server error

HTTP status codes

- Each class is extensible with additional codes
- Clients do not need to understand *all* codes
- Unknown codes default to the X00 code (100, 200 etc)

- https://tools.ietf.org/html/rfc7231#section-6

REST and RESTful

- REST is an architectural style

- RESTful web services are used to describe web services designed according to the REST style

- "RESTful Web services are software services which are published on the Web, taking full advantage and making correct use of the HTTP protocol"

Maturity of RESTful WS

- Whether a web service is RESTful is not either or
- Can be seen as a maturity model with levels of adherence to the REST architecture

Level 0 - HTTP as a tunnel

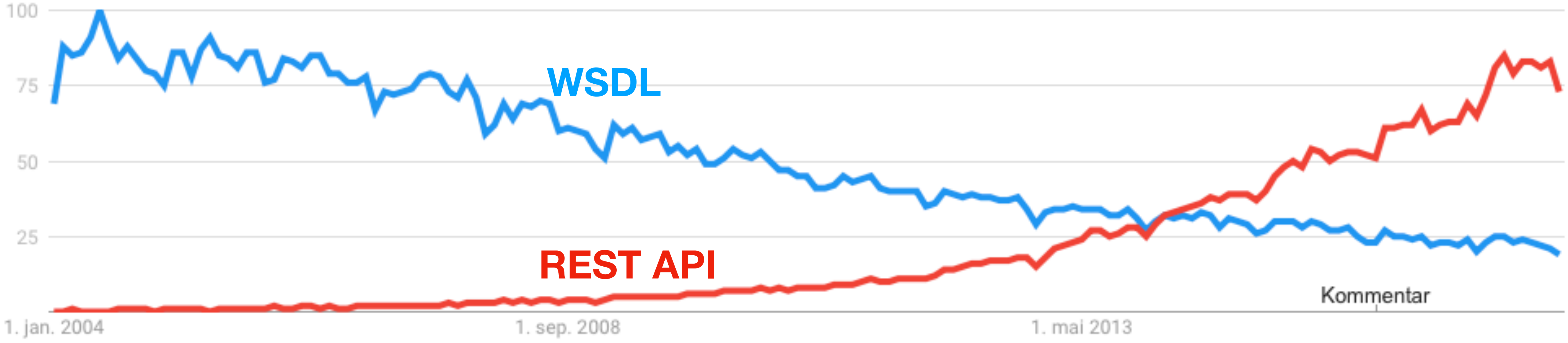
Level 1 - Use of multiple identifiers and resources

Level 2 - *Proper* use of uniform resource interface and verbs

Level 3 - Use of hypermedia to model relationships

RESTful vs other WS

- RESTful web services make full use of the HTTP protocol
- Alternatives like SOAP and XML-RPC use HTTP primarily for transport



"Big" Web Services

- Traditional (non-RESTful) web services are often called "big" web services
- Commonly based on using two standards:
  - WSDL (Web Services Description Language) - XML format for describing/defining the web service
  - SOAP - XML format for communication

"Big" web services

- Based on interacting with *services* e.g. through remote procedure calls (RPCs)
- All operations are typically POSTed to one/few endpoint(s)
- Operations to be performed is based on content of SOAP (or similar) message rather than an HTTP verb
- Extensions to SOAP for specific functionality - WS-Security, WS-Policy, WS-Addressing etc

SOAP example

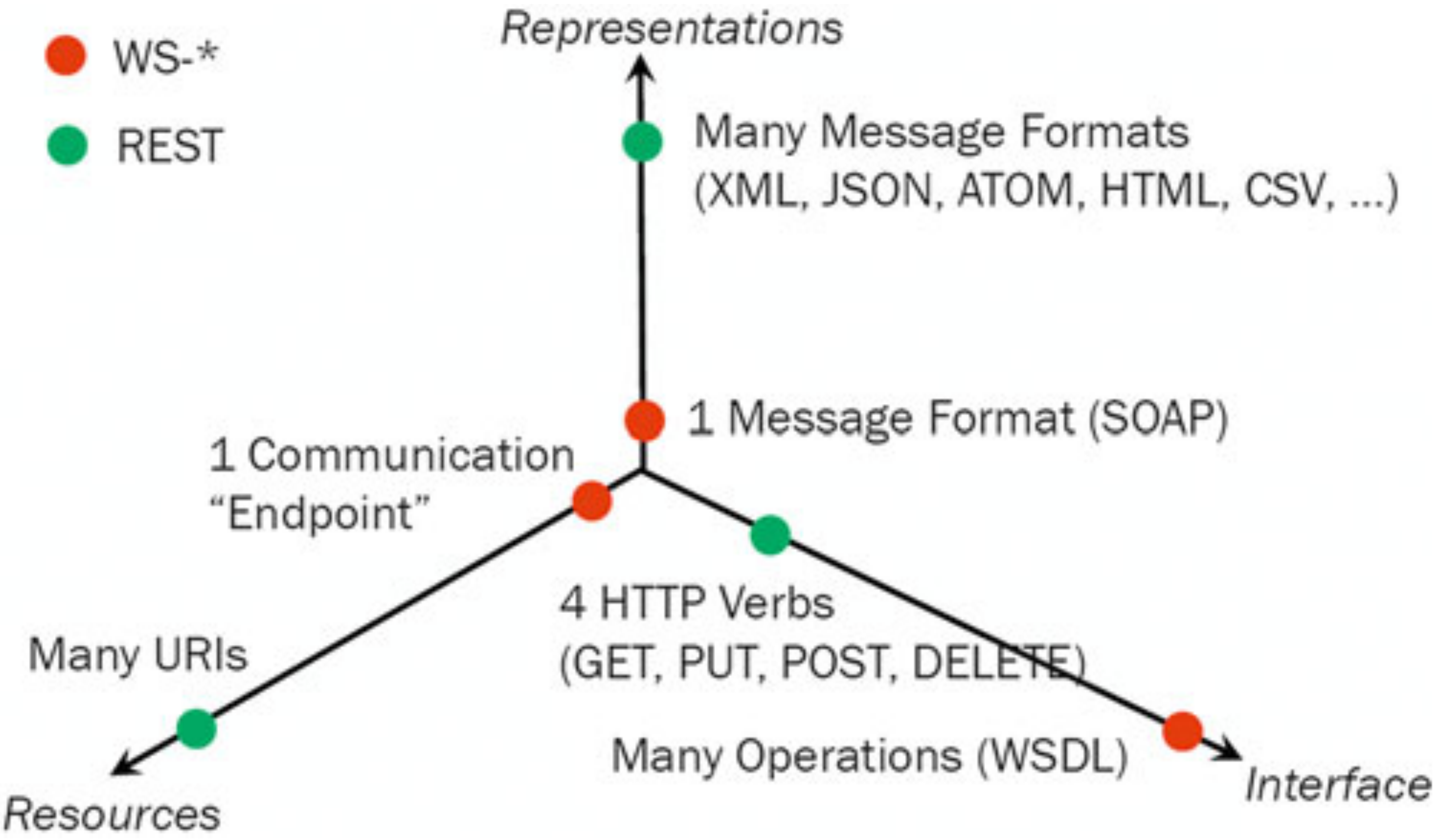
```
<?xml version="1.0" encoding="utf-8" ?>
<soap:Envelope namespaces defined here...>
  <soap:Body>
    <FindCustomerByNum xmlns="urn:OrderSvc:OrderInfo">
      <CustomerNumber>3</CustomerNumber>
    </FindCustomerByNum>
  </soap:Body>
</soap:Envelope>
```

```
<?xml version="1.0" encoding="UTF-8" ?>
<soap:Envelope namespaces defined here...>
  <soap:Body>
    <FindCustomerByNumResponse xmlns="urn:OrderSvc:OrderInfo">
      <CustomerName>Hoops</CustomerName>
    </FindCustomerByNumResponse>
  </soap:Body>
</soap:Envelope>
```

http://somedomain.com/api/customers/3

```
{
  id: 3,
  name: Hoops
}
```

RESTful vs other WS



Literature

- Fielding and Taylor. 2002.
- Pautusso. 2014.

- More on "Big" web services vs RESTful web services: http://www2008.org/papers/pdf/p805-pautassoA.pdf