**Web Services**

(Übung) What is a problem, when designing a RPC-style Web Service for a Use Case like this? What are the disadvantages of SOAP?

CRUD heavy Applications are a bad use case for RPC/SOAP

* We have to define a lot of (CRUD) operations
* Repeatedly, we have to define the same (CRUD) operations for different object types
* We have to define a lot of messages, that contain no parameters or only one primitive
* SOAP is depending on XML; large overhead for complex objects

(Übung) Comparing the REST API to the wsdl API of Exercise 3. What do you notice? Why is REST the better choice, especially for this Use Case?

* REST is data centric, therefore a nice Use Case for CRUD heavy applications
* Resources are identified by URL, unified schema
* Access always by the same operations POST, GET, PUT, DELETE
* Results in clear APIs; REST is independent of data representation

(Übung) Why do root and zone servers always operate in iterative mode?

* Root and zone servers are the entry point of the DNS hierarchy. Thus, they are the most frequently requested servers.

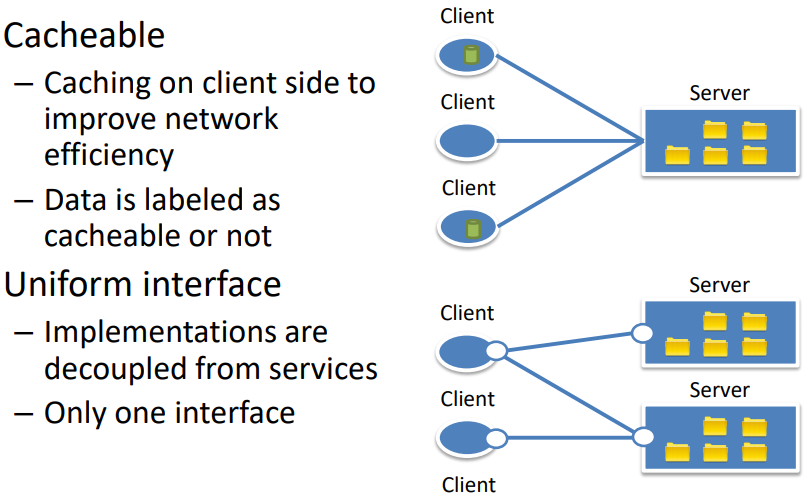
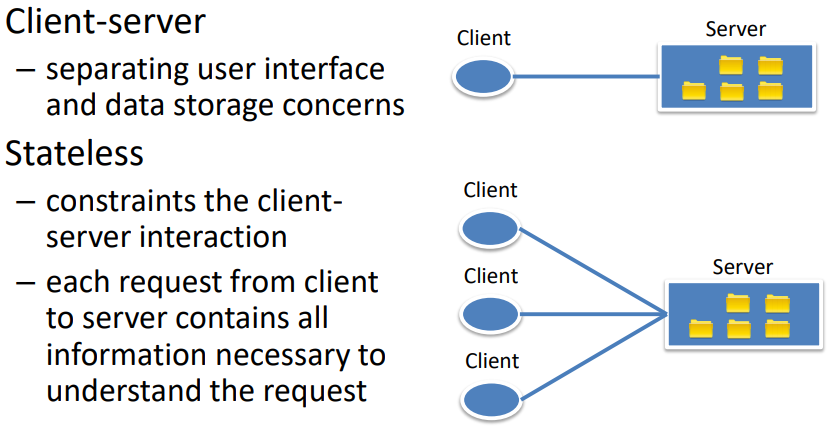
(Übung) Name two possible attacks on DNS?

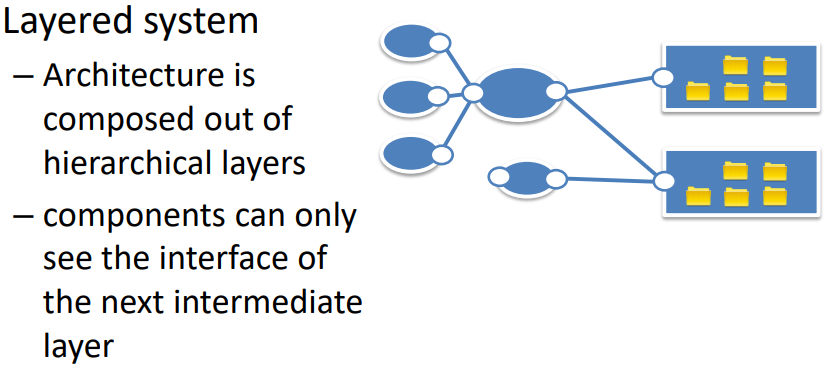
* DNS amplification attack
* DNS cache poisoning

**RESTful Webservices**

* Representational State Transfer
* REST is an architectural style for distributed systems
* Described by Roy Fielding, author of the HTTP Specification
* Abstraction of the structure and behaviour of the World Wide Web
* REST is very popular 🡪 Google, Facebook, Twitter, Flickr, eBay, Amazon support REST

Rest styles



**Rest operations**

Interface provides four HTTP methods to manipulate resources

* **GET** (@GET, @Path(“/shoppingcart/items/{item.id})🡪 /shoppingcart/Ball/15
  + Request a resource from the server
* **POST** 🡪Creates new (sub) – resource below the specified resource
* **PUT** 🡪Specified resources will be created
* **DELETE** 🡪Deletes specified resource

**REST advantages**

* Easy to implement, simple design
* Scalable, uses HTTP caching
* Global identification of resources
* REST is the architectural style of the web itself
* REST is protocol independent

**Web services with SOAP (? Woas net wos des genau isch)**

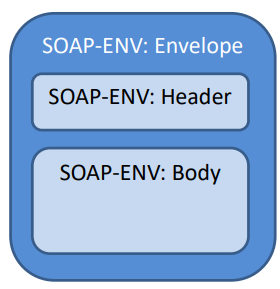
Web service protocol

* Simple Object Access Protocol (SOAP)
* XML messages (presentation layer)
* send over HTTP (session layer)

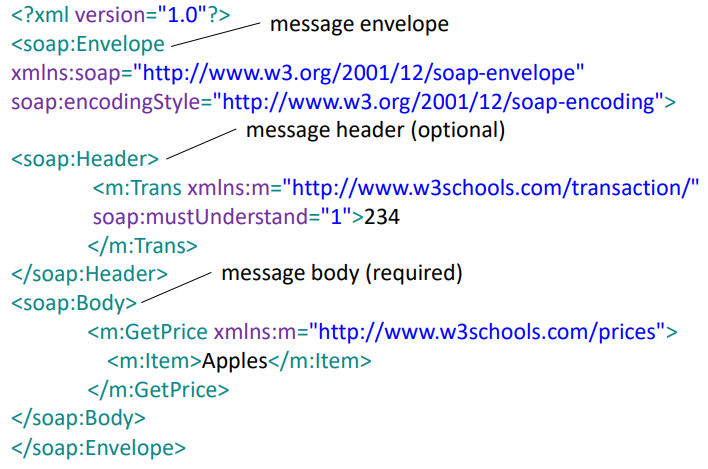
Web service interface description

* Web service definition language (wsdl file)
* XML representation
* provided over HTTP

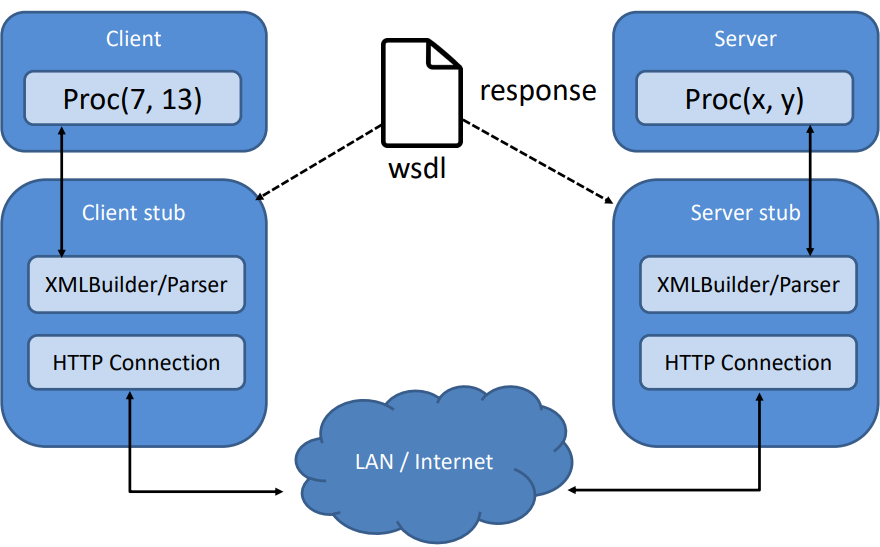
**Message format**

* Envelope
  + Enclosing entity of a message,
  + Defines namespace
* Header
  + Contains metadata for the body
  + Many WS-\* extensions add additional information here
* Body
  + Contains the payload
  + Further specifications define the body structure

**SOAP Request**

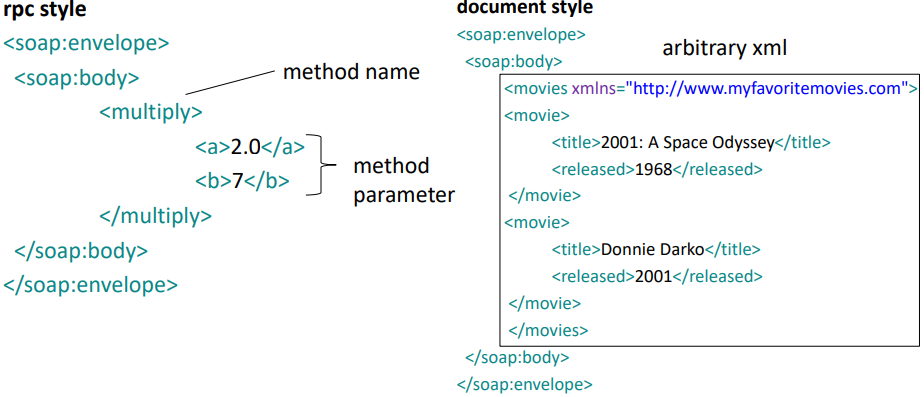


**RPC with SOAP (?)**



**RPC vs. document style**

* RPC-style
  + SOAP message is constructed in a specific way
  + call the Web service just like a normal function
  + Body of message contains parameters and method name as wrapper element – marshalling/unmarshalling is part of the standard
* Document-style – contains no restrictions
  + Message body is a XML document
  + client/server handles the marshalling/unmarshalling



Hier wäre noch wsdl aber fraglich ob man es braucht. Folieg 11 bis 16

**REST vs. SOAP**

* Data-centric vs. process-centric
* Define resources vs. define operations
* **SOAP** defines a contract between client and server 🡪 high coupling
* **REST** defines a generic client that knows how to use a protocol and standardized methods 🡪 less coupling