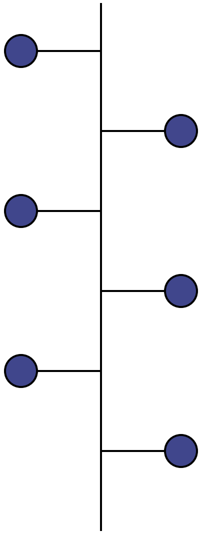
Introduction to Communication Networks and Distributed Systems



Unit 5: From WWW to Web Services

Prof. Dr.-Ing. Adam Wolisz

TKN Telecommunication Networks Group

Acknowledgements:

- We acknowledge the use of slides from: Prof. Holger Karl, Paderborn; Prof. Ion Stoica, Berkeley, Prof. Ashay Parekh; Berkeley; Prof Lauer WPI; Prof. Baker, ACET, as well as slides form books by Tannenbaum, Kurose and Ross, Colouris at al.

WEB SERVICES

Web services

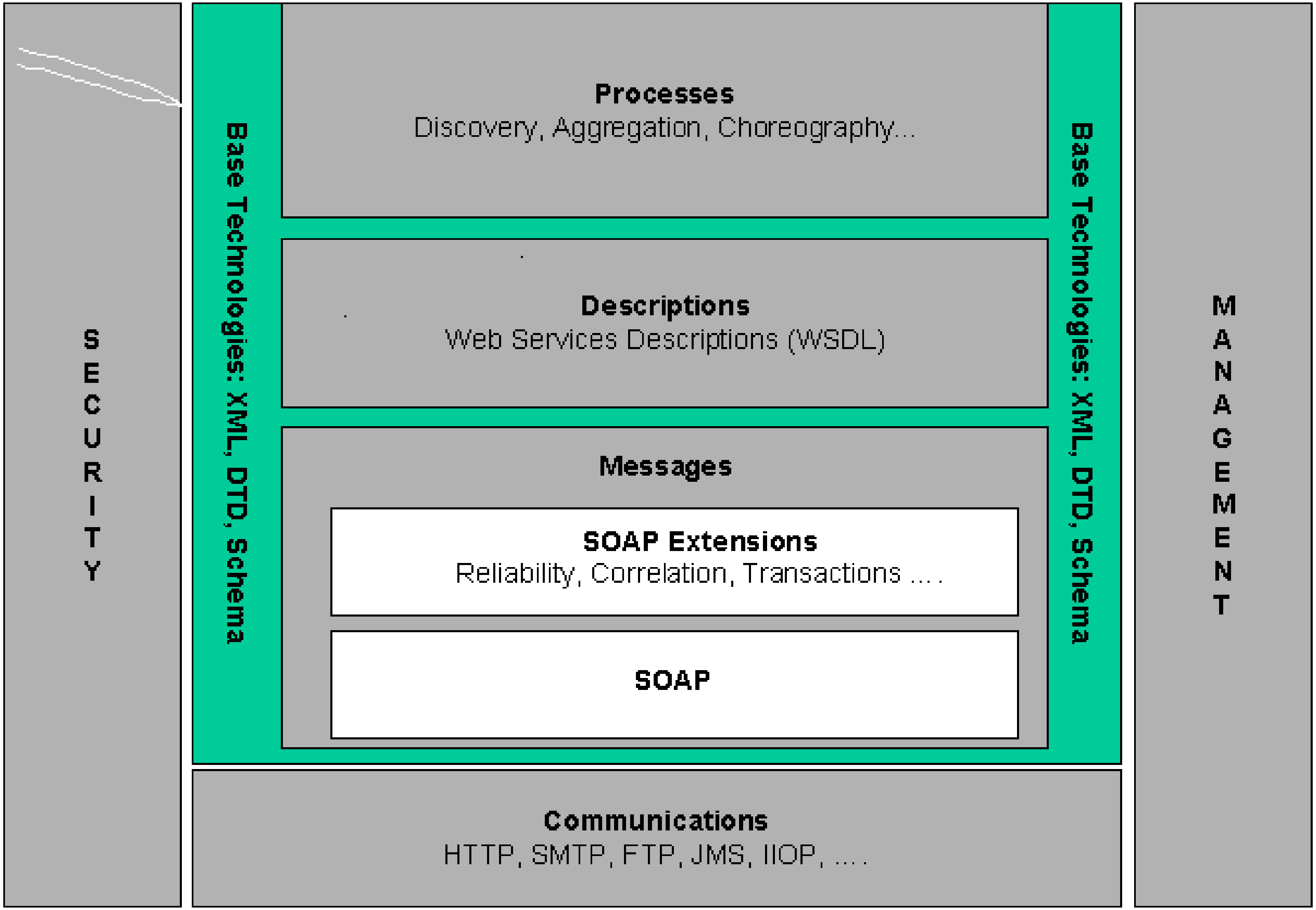
- A Web service is a collection of functions
  - Packaged as a single entity and
  - Published to the network for use by other programs
- Web services
  - Building blocks for creating open distributed systems
  - Allow companies and individuals to quickly and cheaply make their digital assets available worldwide
- A Web service can aggregate other Web services to provide a higher-level set of features

===

- Several popular sites provide Web services
  - Yahoo, Google, eBay, Amazon, …
- Example: Access to Amazon from within a program
  - See http://aws.amazon.com for details
  - Wrappers for several programming languages available

Web Services Architecture Stacks

- www.w3c.org


SECURITY
Base Technologies: XML, DTD, Schema
Processes
Discovery, Aggregation, Choreography…
Descriptions
Web Services Descriptions (WSDL)
Messages
SOAP Extensions
Reliability, Correlation, Transactions … .
SOAP
Communications
HTTP, SMTP, FTP, JMS, IIOP, … .
Base Technologies: XML, DTD, Schema
MANAGEMENT

Basic blocks of Web Services…

- **UDDI (universal description, discovery and integration)**

  Services have to be discovered - http://uddi.xml.org/

- **WSDL (web services description language)**

  Interfaces have to be described - http://www.w3.org/TR/wsdl

- **SOAP (simple object access protocol)**

  (remote) objects access - http://www.w3.org/TR/soap/

- **XML (Extensible Markup Language)**

  data description format - http://www.w3.org/XML/

- **HTTP (Hyper Text Transfer Protocol)**

  communication layer - http://www.w3.org/Protocols/

see also: http://www.w3schools.com/default.asp

What is SOAP?

- Lightweight protocol used for exchange of messages in a decentralized, distributed environment
- Platform-independent
- Used for Remote Procedure Calls
- W3C note defines the use of SOAP with XML as payload and HTTP as transport

Simple Object Access Protocol (SOAP)

- High-level communication protocol
  - Mostly: request/reply semantics (“RPC-style”), also documents…
  - SOAP defines message formats, not the protocol as such
  - Relies on the HTTP POST message for actually delivery
- Between applications
  - So far, we discussed RPC to achieve this
  - RPC “disadvantage”: compatibility problem, security (firewalls)
- Idea: Use RPC principles, but
  - Define a common representation of data
  - Use a generally available transport protocol: mostly HTTP
    - E.g., to traverse firewalls
    - Implementations using other protocols (e.g. SMTP) exist!
  - Use XML to represent data (plain text data representation)

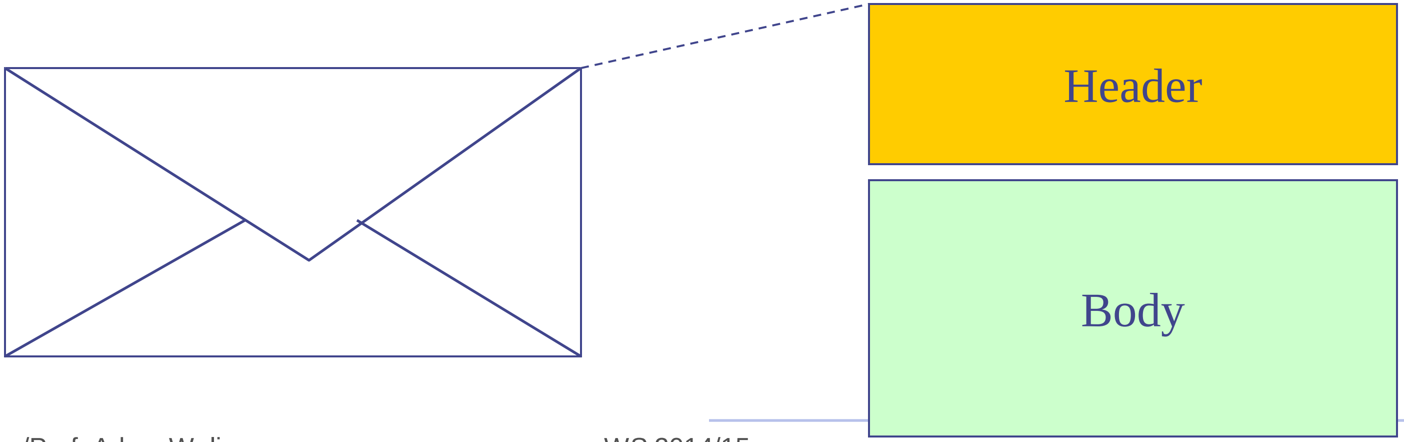
SOAP (continued)

[Karl, Paderborn]

- Main point: The interface of the service to which the address is sent need not be known!
  - Restrictions can be expressed with attributes like mustUnderstand
- How the service is implemented is irrelevant – it only needs to be able to process HTTP and XML

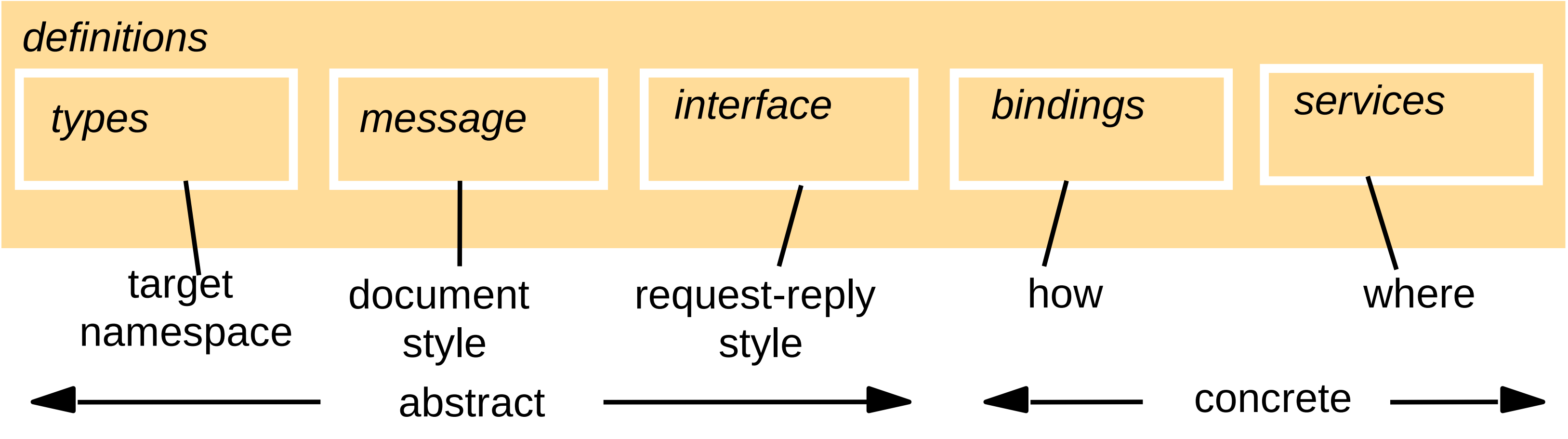
SOAP Elements

- Envelope (mandatory)
  - Top element of the XML document representing the message
- Header (optional)
  - Determines how a recipient of a SOAP message should process the message
  - Adds features to the SOAP message such as authentication, transaction management, payment, message routes, etc…
- Body (mandatory)
  - Exchanges information intended for the recipient of the message
  - Typical use is for RPC calls and error reporting



Web Service Description Language (WSDL)

- Interface specification for web services
  - Akin to interface definition languages for RPC, RMI
  - Written in XML to be programming-language-agnostic
  - Also includes how and where (URI) a service can be invoked
- Main elements of WSDL description
  - Abstract: which compound types are used, combined into which messages
  - Concrete: How and where is the service to be contacted?



WSDL Definitions (2)

[Karl, Paderborn]

- *Types*: First define which data types are going to be exchanged between participants
  - Use existing XML-based type system
- *Message*: Define which kinds of messages can be sent between different entities
  - Which data types are included in which message
  - These are abstract messages, no reference to how these messages are represented on the wire

WSDL Definitions (3)

[Karl, Paderborn]

- *Port types*: A set of supported operations form the type of a port
  - *Operation*: An operation is a specification which abstract message type is sent and which one is received
  - Four kinds of operations exist:
    - One-way: entity only receives a message
    - Request/response: entity receives a messages and answers with a message
    - Solicit/response: entity sends a message and receives an answer
    - Notification: entity only sends a message
- *Binding*: As a port type is still an abstract concept, a mapping to a single, real protocol has to be specified
  - Message format, protocol details
  - Typical bindings: SOAP, HTTP GET/POST
  - Bindings must not include address information

WSDL Definitions (4)

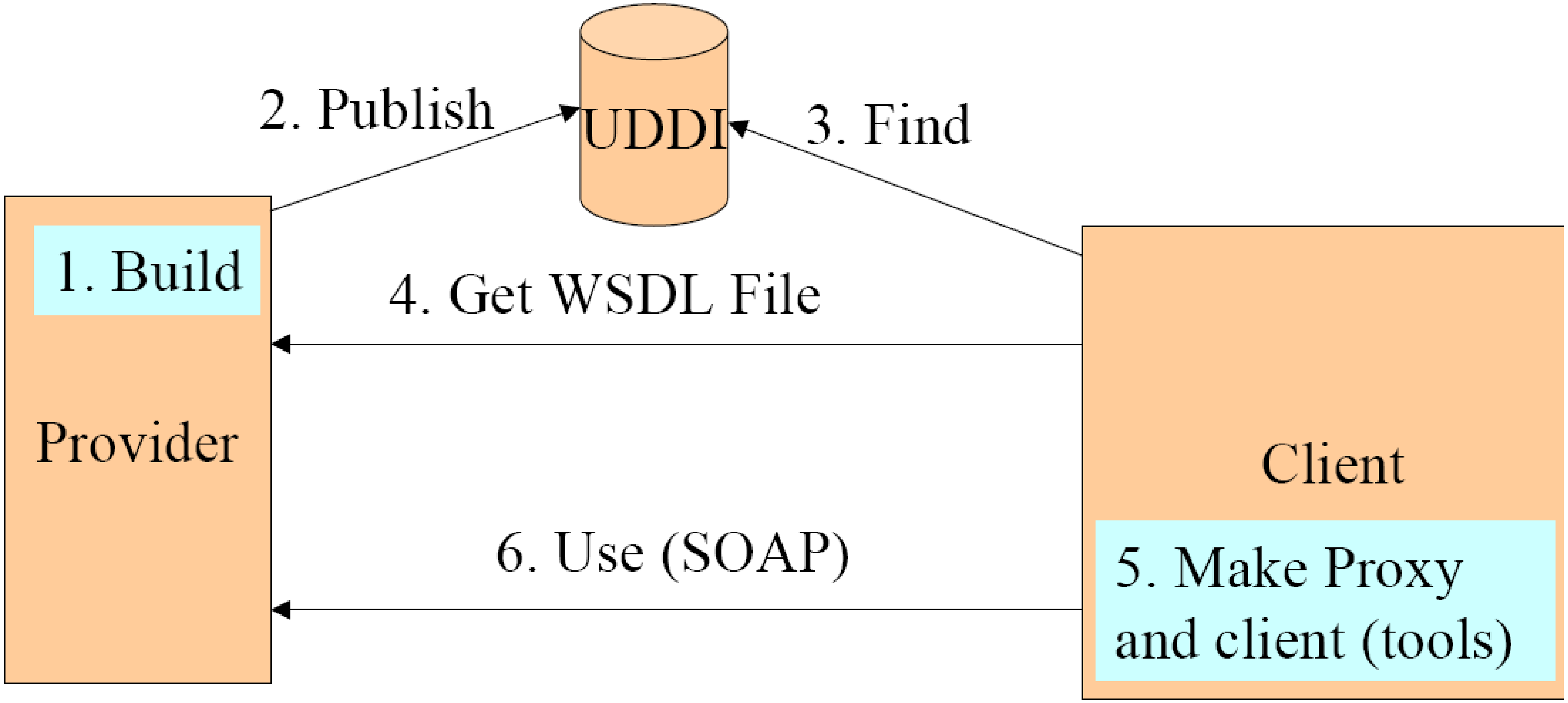
[Karl, Paderborn]

- *Service*: Ports can be grouped into services
  - *Port*: A real port is then a binding with an address
    - Hence: an address where a number of operations can be invoked, along with the protocol information how to do so
  - Ports within a service do not communicate with each other
  - Service can contain several ports with the same port type, but different bindings -> alternative ways to access the same functionality using different protocols

UDDI

- UDDI is used to register and look up services with a central registry
  - Service Providers advertise their business services
  - Service consumers can look up UDDI-entries
  - UDDI Parts:
    - White pages: Business information (Name, contact, description,...)
    - Yellow pages: Service information
    - Green pages: Technical information (Access point, WSDL reference)
- UDDI-registry:
  - Distributed system (!) of individual UDDI-Servers
  - XML-based; Stores descriptions, provides WSDL
- Initial vision: "[…] help companies conduct business with each other in an automated fashion [...]" [sys-con.com]
- Reality today: Human element stays important ->UDDI not very widespread
- *Followed by: Business Process Execution Language (BPEL)*
  - *Specification of business processes based on Web Services*

Interaction



1. Build

2. Publish

3. Find

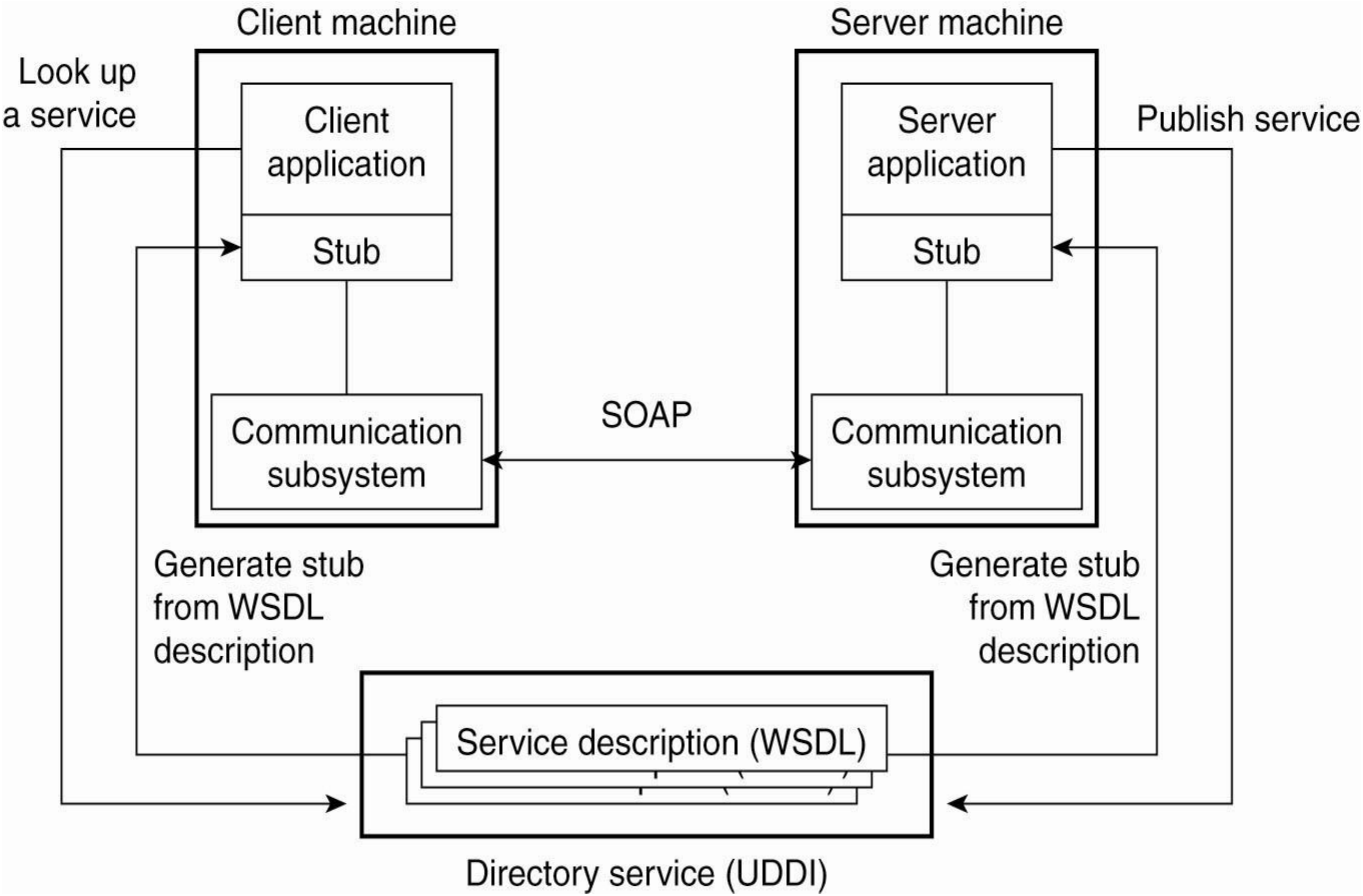
4. Get WSDL File

5. Make Proxy and client (tools)

6. Use (SOAP)

Web Services Fundamentals

[Tanenbaum]

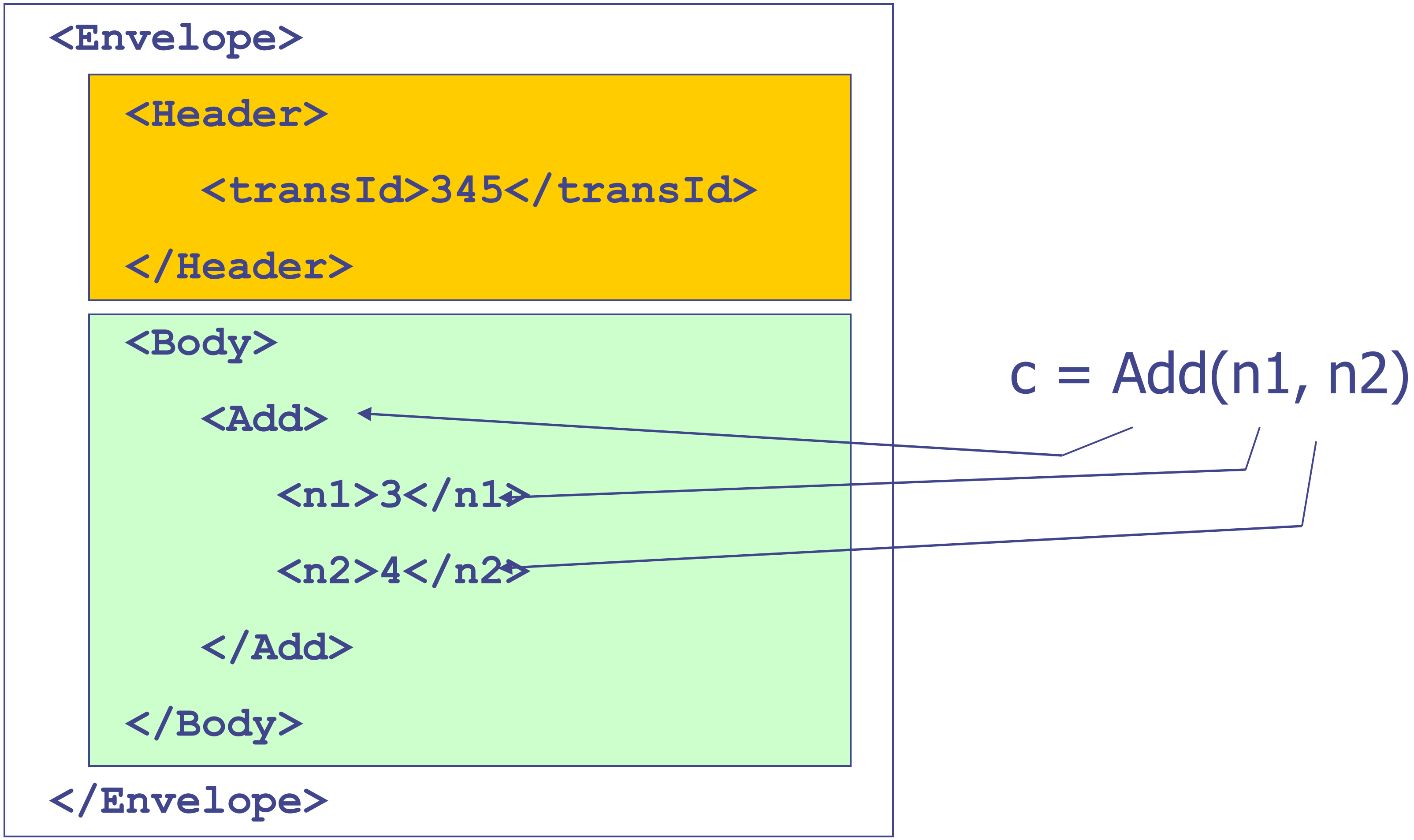


Addendum

Next few slides provide simple expamples for SOAP..

support for lab preparation..

Simple Example



```
<Envelope>
    <Header>
        <transId>345</transId>
    </Header>
    <Body>
        <Add>
            <n1>3</n1>
            <n2>4</n2>
        </Add>
    </Body>
</Envelope>
```

c = Add(n1, n2)

SOAP Request

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=“http://schemas.xmlsoap.org/soap/envelope/”
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/”>
   <SOAP-ENV:Header>
      <t:transId xmlns:t=“http://a.com/trans”>345</t:transId>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <m:Add xmlns:m=“http://a.com/Calculator”>
         <n1>3</n1>
         <n2>4</n2>
      </m:Add>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP Request

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=“http://schemas.xmlsoap.org/soap/envelope/”
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/”>
   <SOAP-ENV:Header>
      <t:transId xmlns:t=“http://a
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <m:Add xmlns:m=“http://a
         <n1>3</n1>
         <n2>4</n2>
      </m:Add>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Scopes the message to the SOAP namespace describing the SOAP envelope

Establishes the type of encoding that is used within the message (different data types supported)

SOAP Request

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV=“http://schemas.xmlsoap.org/soap/envelope/”
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/”>
   <SOAP-ENV:Header>
      <t:transId xmlns:t=“http://a.com/trans”>345</t:transId>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <m:Add xmlns:m=“http://a.com/Calculator”>
         <n1>3</n1>
         <n2>4</n2>
      </m:Add>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Qualifies transaction Id

Defines the method

SOAP Response

```
<SOAP-ENV:Envelope
   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
   SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <SOAP-ENV:Header>
      <t:transId xmlns:t="http://a.com/trans">345</t:transId>
   </SOAP-ENV:Header>
   <SOAP-ENV:Body>
      <m:AddResponse xmlns:m="http://a.com/Calculator">
         <result>7</result>
      </m:AddResponse>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Response typically uses method name with "Response" appended

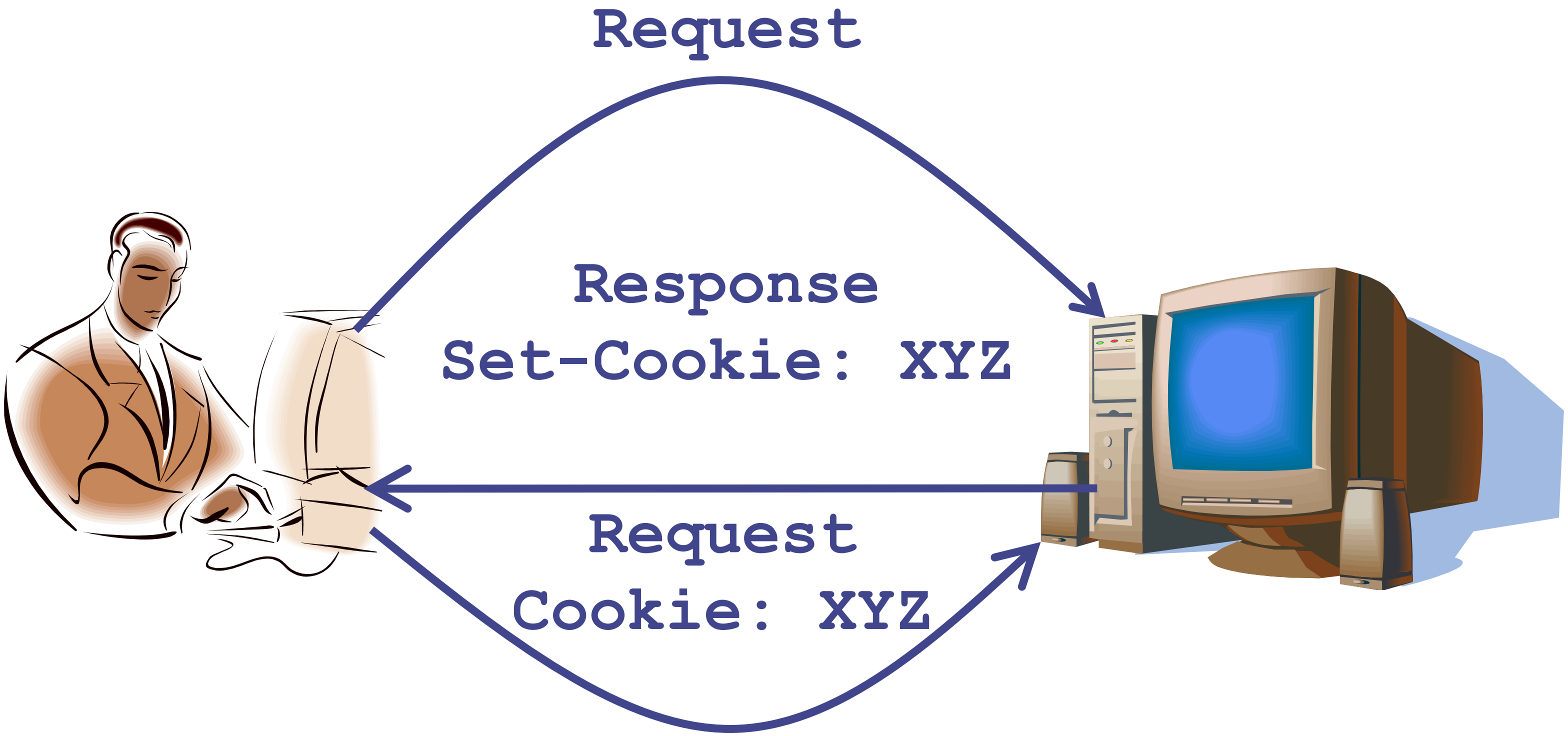
Being Stateless…

HTTP is Stateless

- **Stateless** Servers should *not be required* to retain state
- This is **good -** Improves scalability on the server-side
  - Don't have to retain info across requests
  - Can handle higher rate of requests
  - Order of requests doesn't matter
- This is also **bad -** Some applications need persistent state
  - Need to uniquely identify user or store temporary info
  - *e.g.*, Shopping cart, user preferences/profiles, usage tracking, …

State in a Stateless Protocol: Cookies

- *Client-side* state maintenance
  - Client stores small(?) state on behalf of server
  - Client sends state in future requests to the server
- Can provide authentication



Notion of *Fate-Sharing*

- Idea: when storing state in a distributed system, keep it co-located with the entities that ultimately rely on the state
- Fate-sharing is a technique for dealing with failure
  - Only way that failure can cause loss of the critical state is if the entity that cares about it also fails ...
  - … in which case it doesn't matter
- Often argues for keeping *network state* at end hosts rather than inside routers
  - In keeping with End-to-End principle
  - E.g., packet-switching rather than circuit-switching
  - **E.g., HTTP "cookies"**

RESTful [Baker]

- RESTful
  - REST is an architectural style for distributed systems.
- An architectural style is:
  - … an abstraction, a design pattern, a way of discussing an architecture without concern for its implementation.
- REST defines a series of constraints for distributed systems that together achieve the properties of:
  - Simplicity, Scalability, modifiable, performance, visibility (to monitoring), portability and reliability.
- A system that exhibits all defined constraints is RESTful!

Understanding REST [Baker]

- Representational State Transfer:
  - The Resource:
    - A resource is any information that can be named: documents, images, services, people, an collections.
  - Resources have state:
    - State may change over time.
  - Resources have identifiers:
    - A resource is anything important enough to be referenced.
  - Resources expose a uniform interface:
    - System architecture simplified, visibility improved,
    - Encourages independent evolution of implementations.

Understanding REST (2) [Baker]

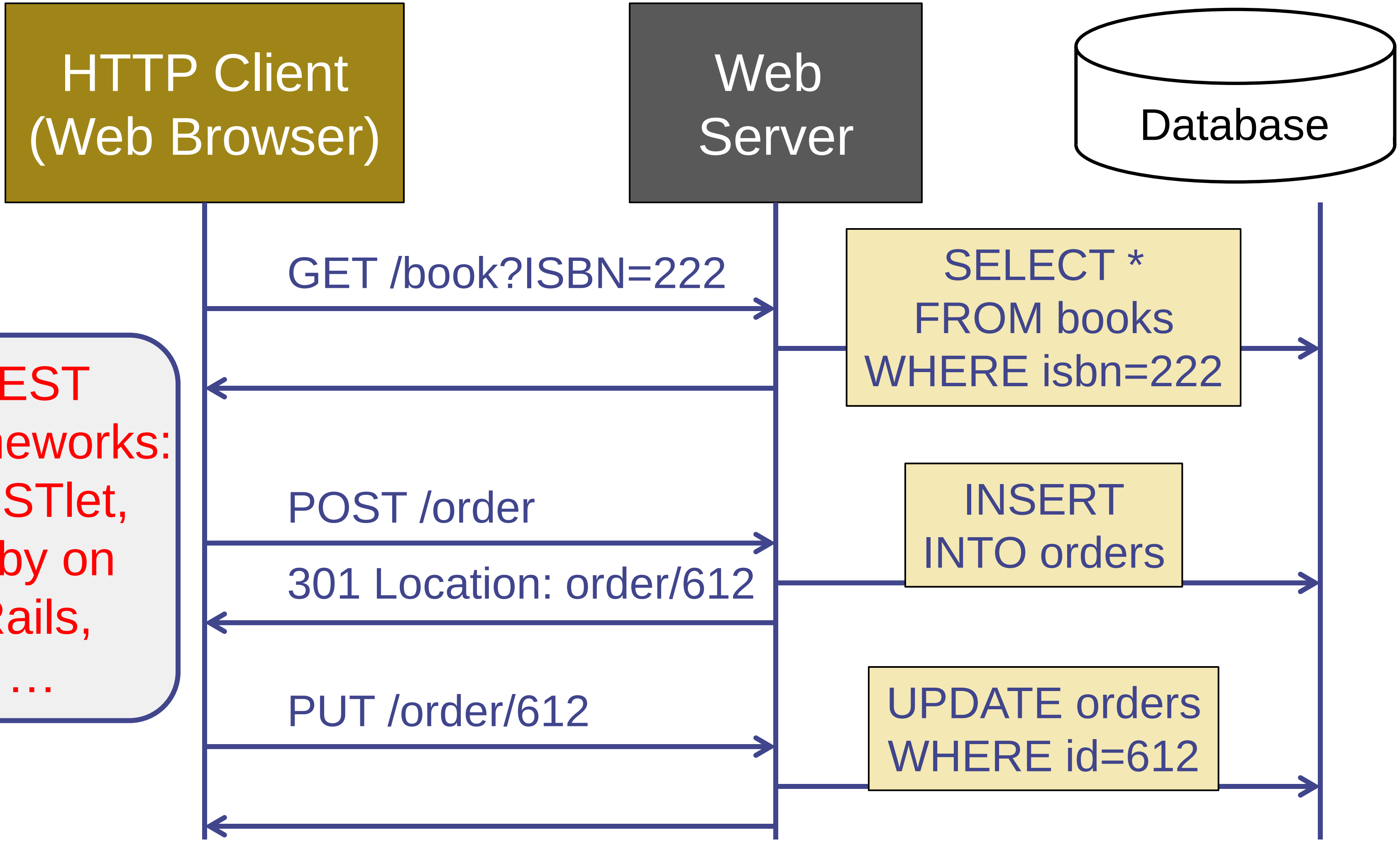
- Representational State Transfer:
  - On request, a resource may transfer a representation of its state to a client:
    - Necessitates a client-server architecture.
  - A client may transfer a proposed representation to a resource:
    - Manipulation of resources through representations.
  - Representations returned from the server should link to additional application state:
    - Clients may follow a proposed link and assume a new state Hypermedia as the engine of application state.

Understanding REST (3) [Baker]

- Representational State Transfer:
  - Stateless interactions:
    - Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server.
  - Statelessness necessitates self-descriptive messages:
    - Standard media types,
    - Meta-data and control-data.
  - Uniform interface + Stateless + Self-descriptive = Cacheable:
    - Cacheable necessitates a layered-system.

RESTful

- Example:



REST Frameworks: RESTlet, Ruby on Rails, …

Web Sockets

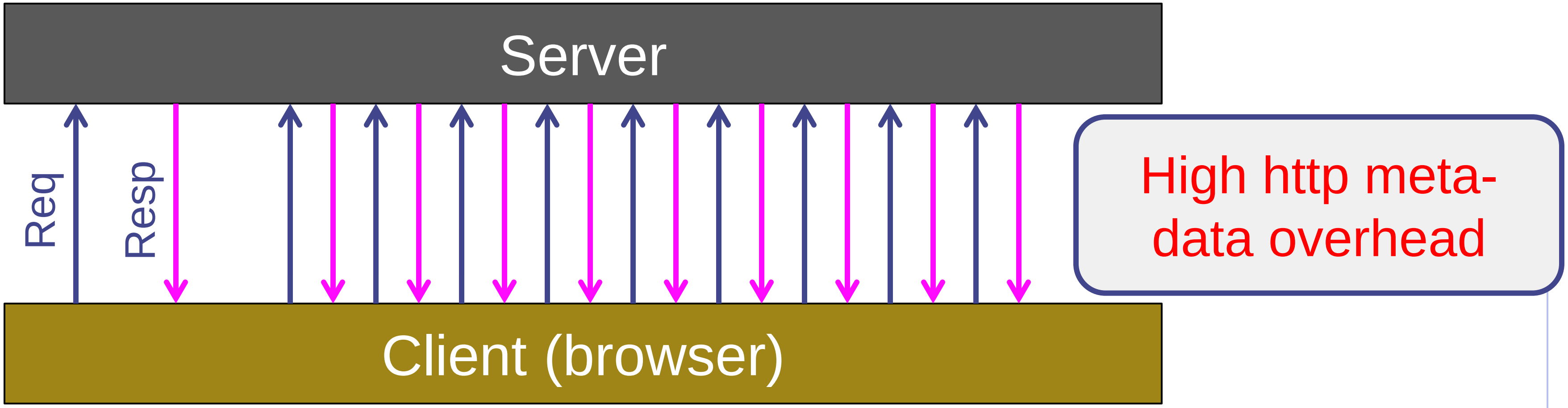
For more details see:

http://www.websocket.org/aboutwebsocket.html

https://developer.mozilla.org/en-US/docs/WebSockets/Writing_WebSocket_client_applications

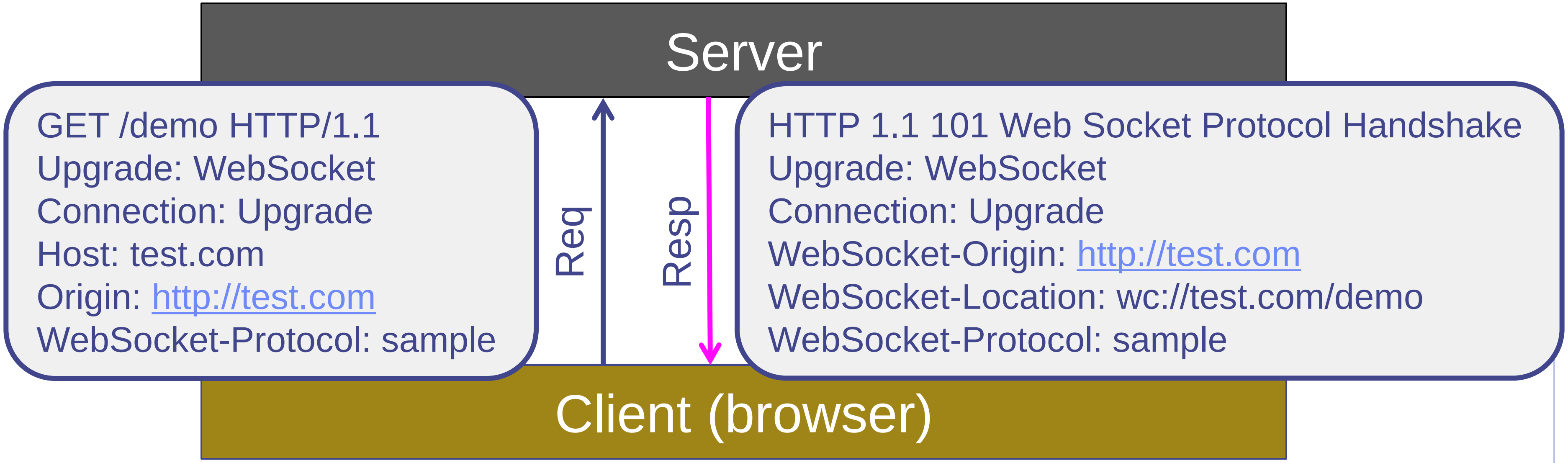
HTML 5 WebSocket API

- Why do we need WebSockets?
  - Web apps demand real-time, event-driven communication with minimal latency
    - E.g. financial applications, online games, …
  - Problems with HTTP
    - HTTP is half-duplex (traffic flow in only one direction at a time)
    - HTTP adds latency
- Typical use case: polling (AJAX)
  - Poll server for updates, wait at client



What is a WebSocket (WS)?

- W3C/IETF standard
- Uses Websocket protocol instead of HTTP
  - ws:// and wss://
- True full-duplex communication channel
  - Strings + binary frames sent in any direction at the same time
- Uses port 80/443 (-> proxy/firewall)
- Connection established by „**upgrading**“ from HTTP to Websocket protocol



```
GET /demo HTTP/1.1
Upgrade: WebSocket
Connection: Upgrade
Host: test.com
Origin: http://test.com
WebSocket-Protocol: sample
```

```
HTTP 1.1 101 Web Socket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
WebSocket-Origin: http://test.com
WebSocket-Location: wc://test.com/demo
WebSocket-Protocol: sample
```

WS - How Network Traffic is Reduced

- Each message frame has only 2 Bytes of overhead
- No latency from establishing new TCP connection for each HTTP message
- No polling overhead – only send messages when there is something to send
- Usage: e.g. Javascript client

```
connect: function() {
        try {
                this.ws = new WebSocket(‘ws://www.test.com ‘);
                this.ws.onopen = function (event) { /* … */ };
                this.ws.onclose = function (event) { /* … */ };
                this.ws.onmessage = messageListener;
        } catch (exception) {}
},
messageListener: function(event) {
        alert(‘New message: ‘ + event.data);
},
send: function(message) {
        if (this.ws) {  this.ws.send(message);  }
},
```