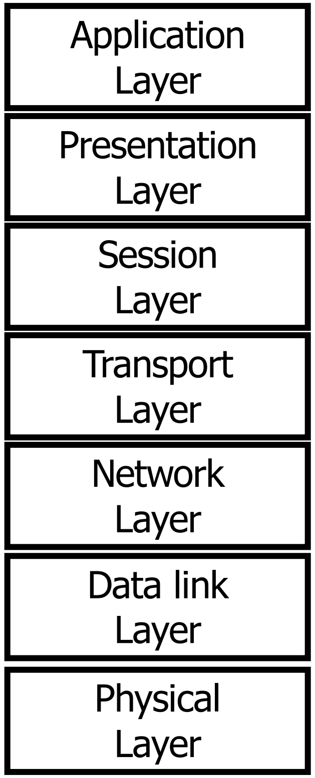
Computer Networks

Web Services

Chapter

1. Introduction
2. Protocols
3. Application layer
4. **Web services**
   - **Introduction**
   - **SOAP**
   - **WSDL**
   - **REST**
5. Distributed hash tables
6. Time synchronization
7. Transport layer
8. UDP and TCP
9. TCP performance
10. Network layer
11. Internet protocol
12. Data link layer

Top-Down-Approach



Web Services

Web Services & Web Technology

- We already discussed the client-server approach as one of the core building blocks of today's Internet
- We also discussed the web as one of the core **services** using the Internet
- So far, we hove not discussed:
  - Scalability (think of Amazon, Google, Ebay, etc.)
  - Other services offered on top of web technologies
  - Recent trends in web technology
- **Today's topics**:
  - Scalable web architectures
  - Resource- and service-oriented web services

What is a Web Service?

- 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, …

W3C Web Services

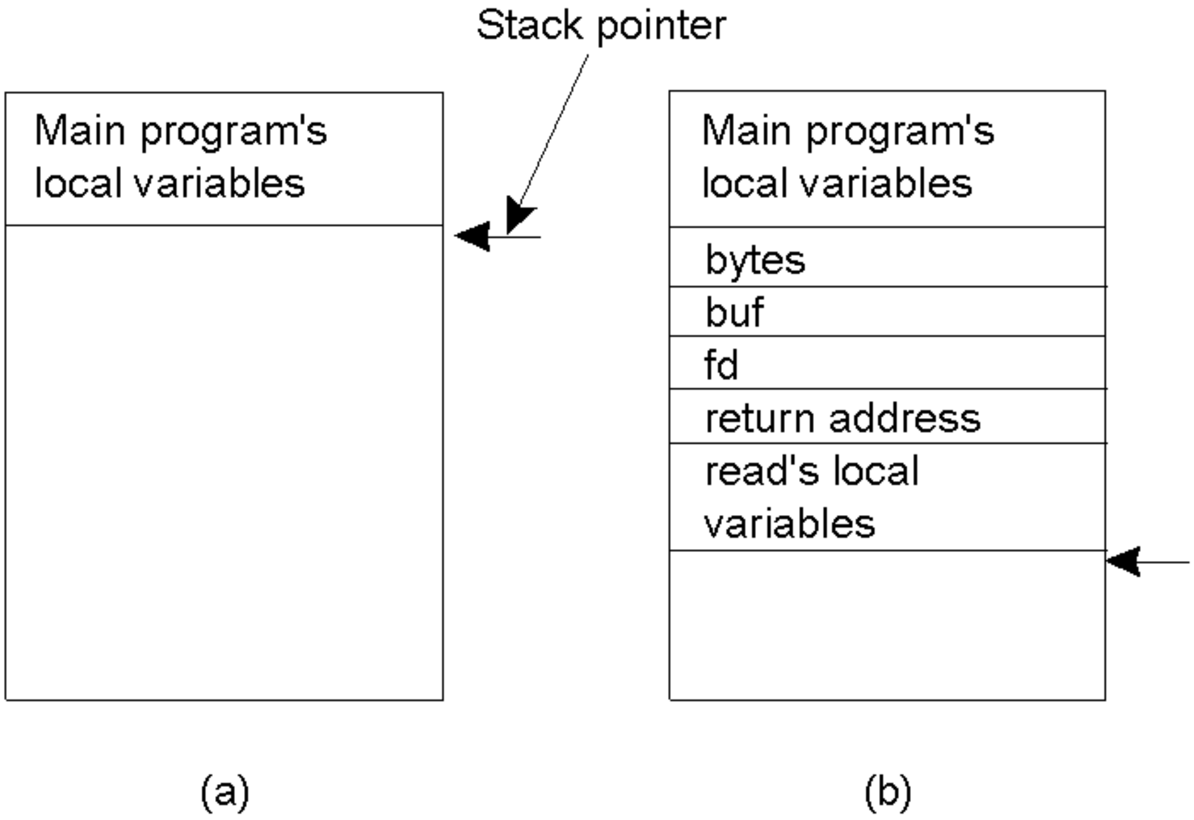
- A web service is a **software system** designed to support interoperable machine-to-machine interaction over a network. It has an **interface** described in a machine-processable format (specifically **WSDL**). Other systems interact with the web service in a manner prescribed by its description using **SOAP-messages**, typically conveyed using **HTTP** with an XML serialization in conjunction with other web-related standards.

W3C, Web Services Glossary

Quick Preview on Distributed Systems: Remote Procedure Calls

Conventional Procedure Call

- (a) Parameter passing in a local procedure call: the stack before the call to `read()`
- (b) The stack while the called procedure is active



Remote Procedure Calls (RPC)

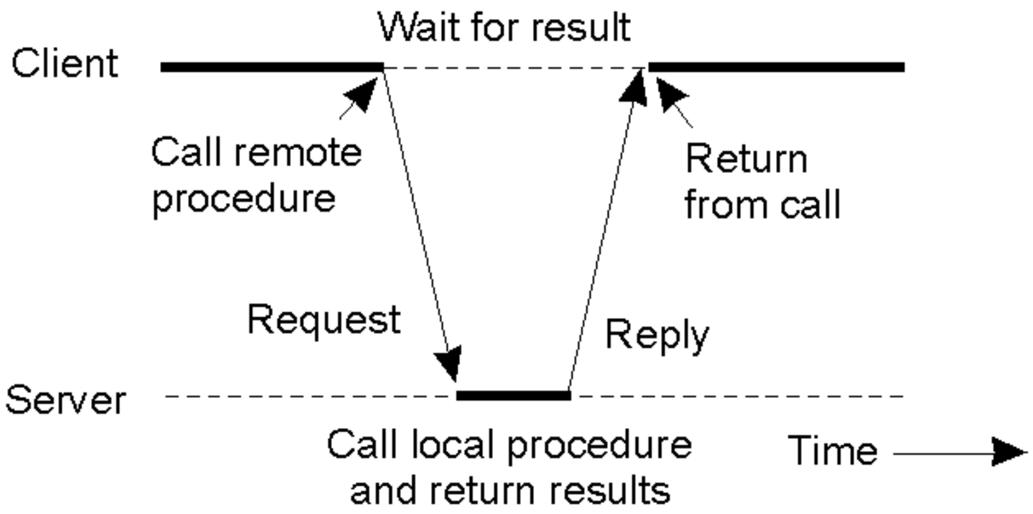
- Why we need Remote Procedures?
  - The client needs a easy way to call the procedures of the server to get some services.
  - RPC enables clients to communicate with servers by calling procedures in a similar way to the conventional use of procedure calls in high-level languages.
  - RPC is modeled on the local procedure call, but the called procedure is executed in a different process and usually a different computer.

RPC Principles

- Adaption of the `send()` & `receive()` IPC paradigm.
- Allows programs in distributed systems to interact with each other via function call/return semantics.
- Abstracts the network to the user and makes it look like the programs are on the same machine.
- Allows to specify a clearly documented interface for the remote server.
  - This also allows to automatically generate the code.
- Due to the clearly documented interface, it is easy to write applications that run on multiple operating systems.

RPC Model

- When a process on machine A calls a procedure on machine B, the calling process on A is suspended, and the execution of the called procedure takes place on B.
- Information can be transported from the caller to the callee in the parameters and can come back in the procedure result.
- No message passing or I/O at all is visible to the programmer.

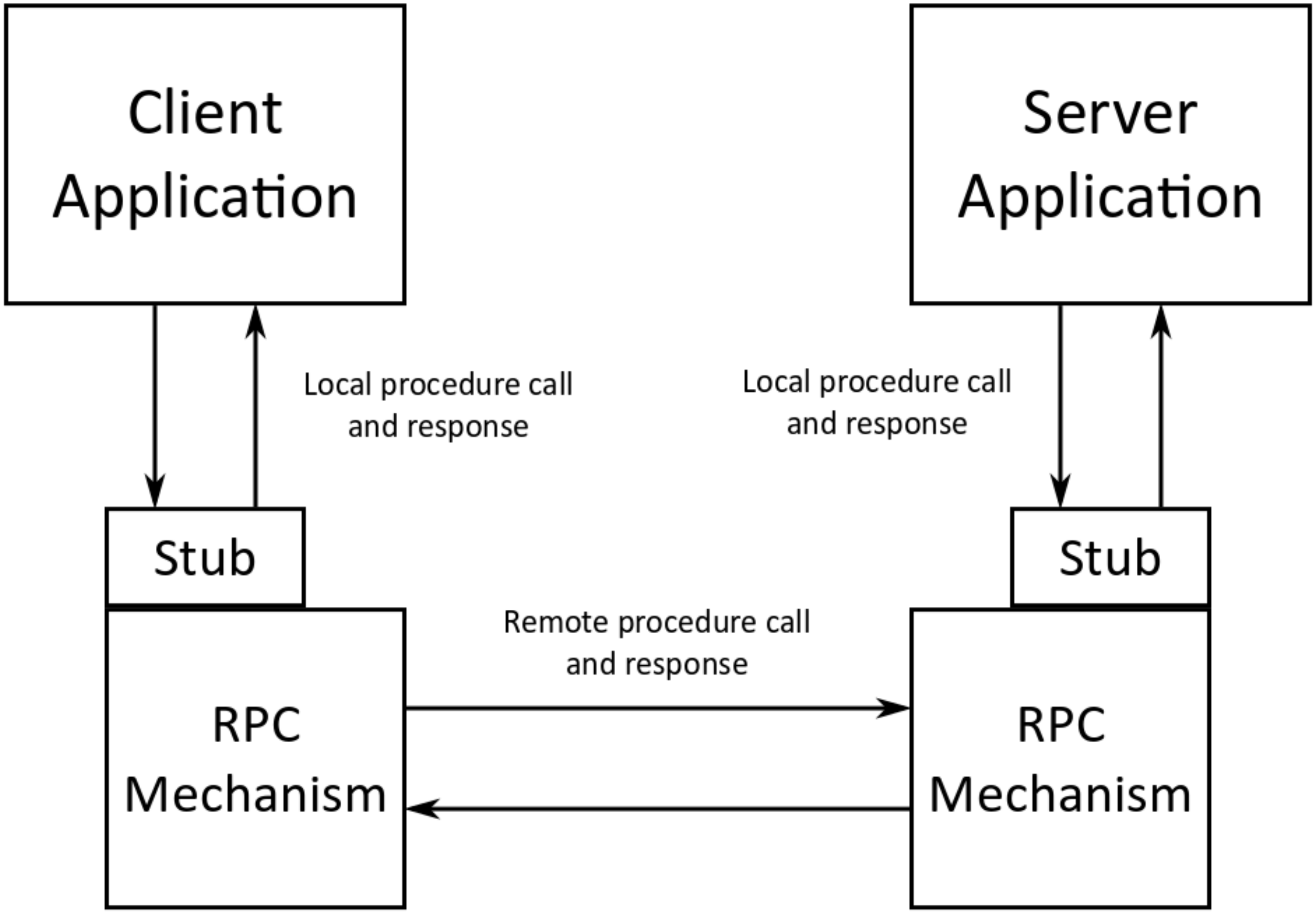


RPC Overview

- The user just calls a function `f(x)` which returns `y`.
- **Abstraction:** It may be hidden that actually a remote call is used

1. The RPC mechanism creates a message out of the function call which includes at least `f` and `x` and sends this message to a remote system.
2. The remote system unpacks the message and a call to the local function `f` with parameters `x` is issued which then leads to the return value `y`.
3. The result `y` is sent back to the initiator and the caller does not see the difference to a local procedure call.

RPC Overview



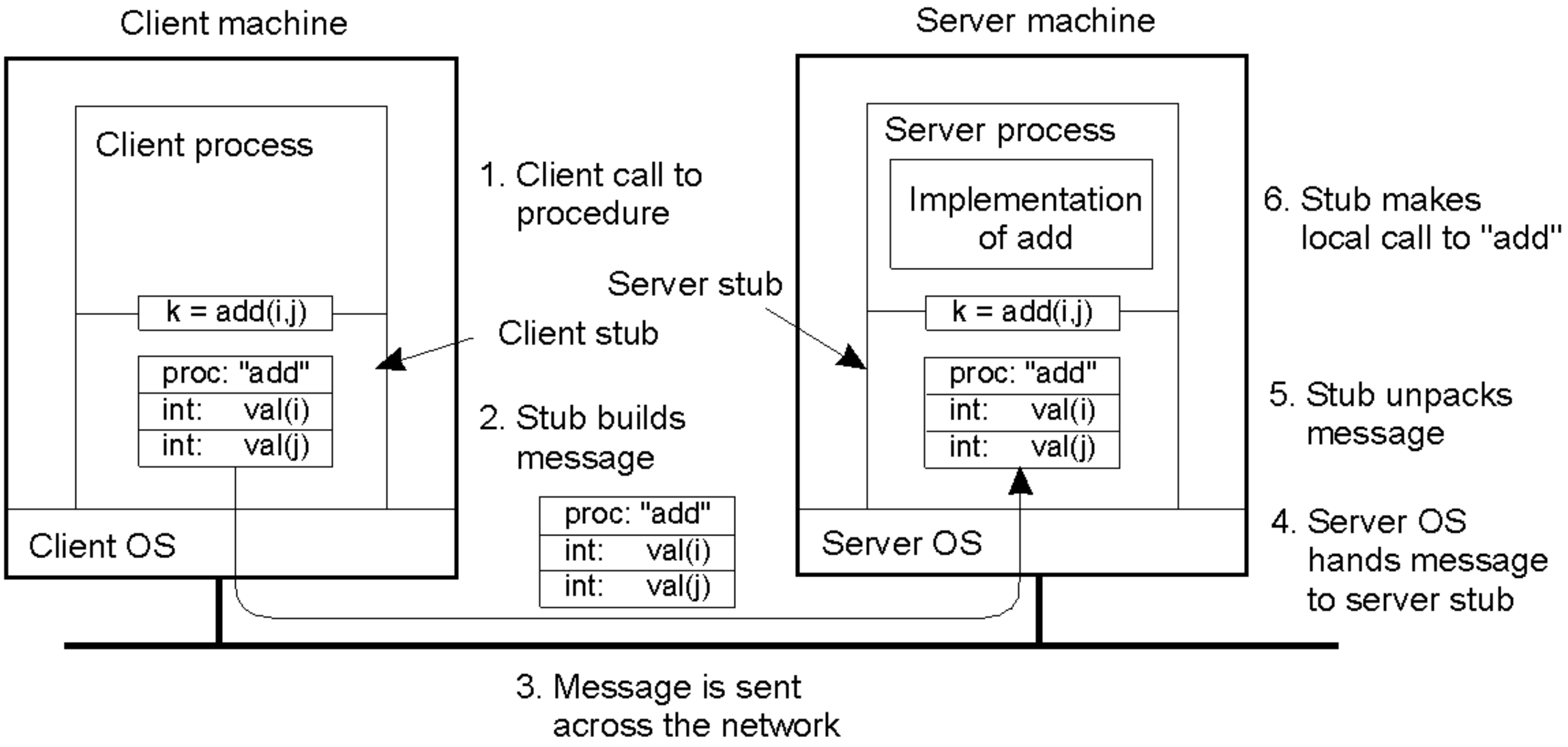
Marshalling

- How does the client transfer its call request (the procedure name) and the arguments to the server via network?

- **Marshalling** and communication with server:
  - For each remote procedure call, a (client) stub procedure is generated and attached to the (client) program.
  - Replace the remote procedure call to a (local) call to the stub procedure.
  - The (codes in the) stub procedure marshals (the input) arguments and places them into a message together with the procedure identifier (of the remote procedure).

Passing Value Parameters

- Steps involved in doing remote computation through RPC



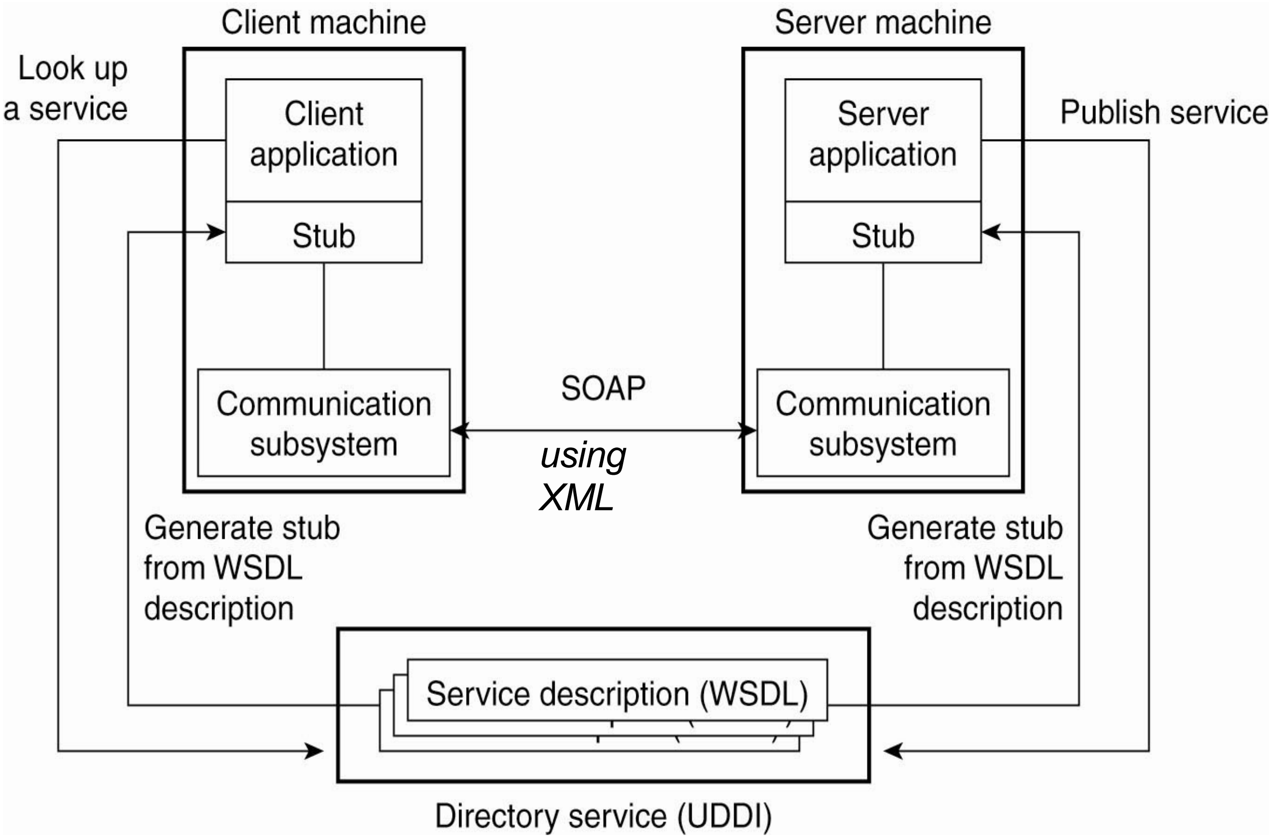
RPC Issues

- Most RPC systems use **call by value** as it is easy to realize.
- To implement **call by reference** a lot has to be done – might not be worth the effort.
- If both systems use the same language parameter, representation is easy!
  - If not, there exist special RPC facilities that handle the problem of data representation.
- RPC is more vulnerable to failure (since it involves a communication system, another machine, and another process).
  - The programmer should be aware of the call semantics, i.e., programs that make use of RPC must have the capability of handling errors that cannot occur in local procedure calls.

More details → Lecture Disributed Systems

Back to Web Services…

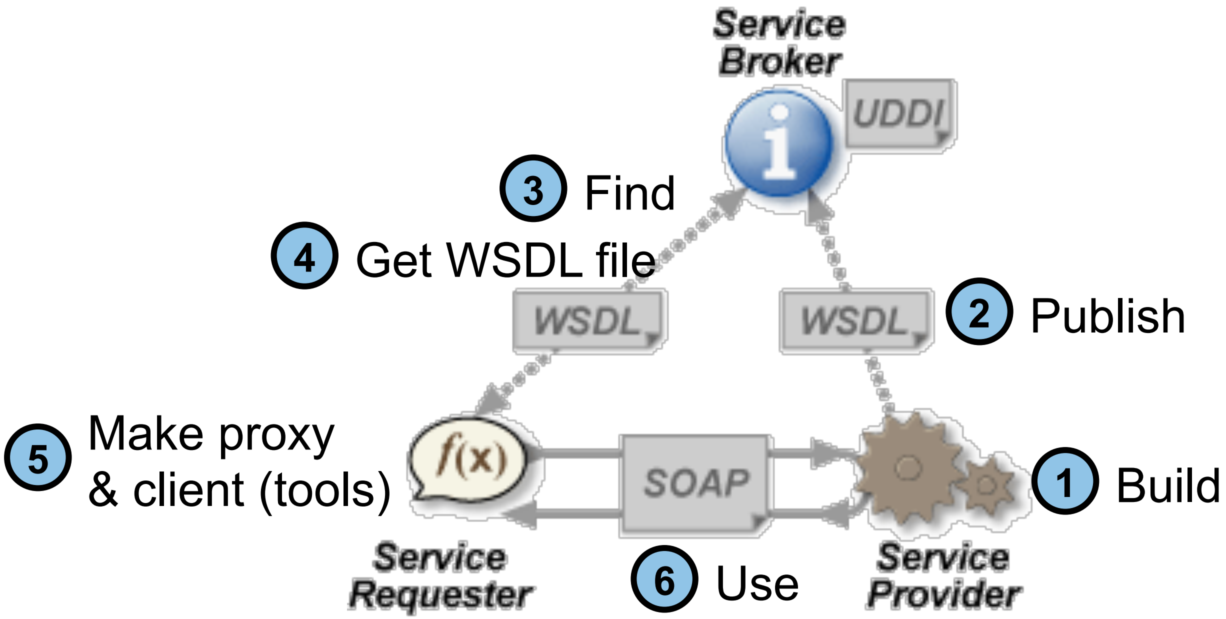
Web Services Architecture



Basic building 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/

Web Service Interaction



1 Build

2 Publish

3 Find

4 Get WSDL file

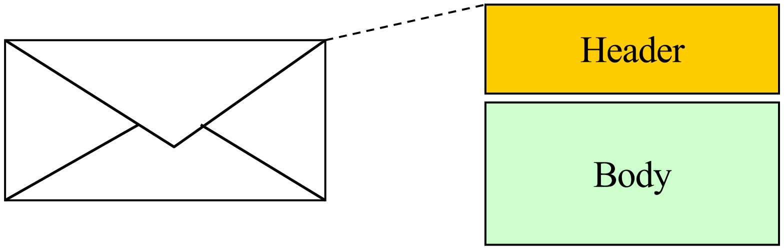
5 Make proxy & client (tools)

6 Use

Simple Object Access Protocol (SOAP)

- High-level communication protocol
  - Mostly: request/reply semantics (RPC style), also documents (message passing) …
  - SOAP **defines message formats**, not the protocol as such
  - Relies on the **HTTP for actual delivery**
- Details
  - Using a common representation of data (XML)
  - Use a generally available transport protocol: mostly HTTP
    - E.g., to traverse firewalls
    - Implementations using other protocols (e.g., SMTP) exist!
    - Comment: W3C defines the use of SOAP with XML as payload and HTTP as transport

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

SOAP Example

- Usage of SOAP to query a database
  - Search for books with „SOAP“ in title
- Request sent by client:

```
<?xml version="1.0"?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
        <s:Body>
                <m:TitleInDatabase xmlns:m="http://www.lecture-db.de/soap">
                        SOAP
                </m:TitleInDatabase>
        </s:Body>
</s:Envelope>
```

SOAP Example (II)

- Response sent by server:

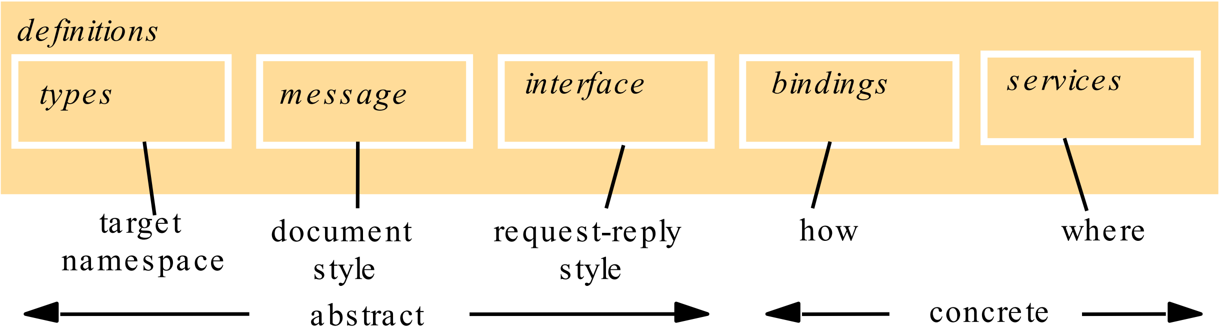
```xml
<?xml version="1.0"?>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope">
        <s:Header>
                <m:RequestID xmlns:m="http://www.lecture-db.de/soap">a3f5c109b</m:RequestID>
        </s:Header>
        <s:Body>
                <m:DbResponse xmlns:m="http://www.lecture-db.de/soap">
                        <m:title value="SOAP">
                                <m:Choice value="1">Arbeitsbericht Informatik</m:Choice>
                                <m:Choice value="2">Seminar XML und Datenbanken</m:Choice>
                        </m:title>
                </m:DbResponse>
        </s:Body>
</s:Envelope>
```

Web Service Description Language (WSDL)

- 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?
- Interface specification for web services
  - Written in XML to be programming-language-agnostic
  - Also includes how and where (URI) a service can be invoked
  - Define which kinds of messages can be sent between different entities (which data types are included in which message)

WSDL Definitions (1)

- **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



[Karl, op. cit.]

WSDL Definitions (2)

- **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 (3)

- **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

Usage of WSDL

- Client
  - reads WSDL to find out the functionality provided by Web Service → all used data types are described in WSDL file
  - creates stub from WSDL file
  - uses SOAP to call a particular function listed in WSDL file

WSDL Example – Stock Quote Service

```
<definitions name="StockQuote" targetNamespace="http://example.com/stockquote.wsdl"
xmlns:tns="http://example.com/stockquote.wsdl"
xmlns:xsd1="http://example.com/stockquote.xsd"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns="http://schemas.xmlsoap.org/wsdl/">
<types>
        <schema targetNamespace="http://example.com/stockquote.xsd"
xmlns="http://www.w3.org/2001/XMLSchema">
        <element name="TradePriceRequest">
                <complexType>
                        <all> <element name="tickerSymbol" type="string"/> </all>
                </complexType>
        </element>
        <element name="TradePrice">
                <complexType>
                        <all> <element name="price" type="float"/> </all>
                </complexType>
        </element>
        </schema>
</types>
```

WSDL Example – Stock Quote Service (II)

```
<message name="GetLastTradePriceInput">
      <part name="body" element="xsd1:TradePriceRequest"/>
</message>
<message name="GetLastTradePriceOutput">
      <part name="body" element="xsd1:TradePrice"/>
</message>

<portType name="StockQuotePortType">
      <operation name="GetLastTradePrice">
            <input message="tns:GetLastTradePriceInput"/>
            <output message="tns:GetLastTradePriceOutput"/>
      </operation>
</portType>
```

WSDL Example – Stock Quote Service (III)

```
<binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
                <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
                <input>
                        <soap:body use="literal"/>
                </input>
                <output>
                        <soap:body use="literal"/>
                </output>
        </operation>
</binding>
<service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteSoapBinding">
                <soap:address location="http://example.com/stockquote"/>
        </port>
</service>
</definitions>
```

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)

- And today?
  - 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

Representational State Transfer (REST)

REST for Web Services

- An alternative to SOAP/WSDL-based Web services
- Set of architectural principles by which one can design Web services that focus on a system's **resources**, including how **resource states** are addressed and transferred over **HTTP**
- Service access by **direct use of HTTP methods** in accordance to the semantics defined in RFC 2616.
- One-to-one mapping between create, read, update, and delete (CRUD) operations and HTTP methods
  - To create a resource on the server (append), use HTTP POST method
  - To retrieve a resource, use HTTP GET method
  - To change the state of a resource or to replace, use HTTP PUT method
  - To remove or delete a resource, use HTTP DELETE method

REST

- REST
  - 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**!

[Baker, op. cit.]

Understanding REST

- Representational State Transfer:
  - The **Resource**:
    - A resource is any information that can be named: documents, images, services, people, and 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.

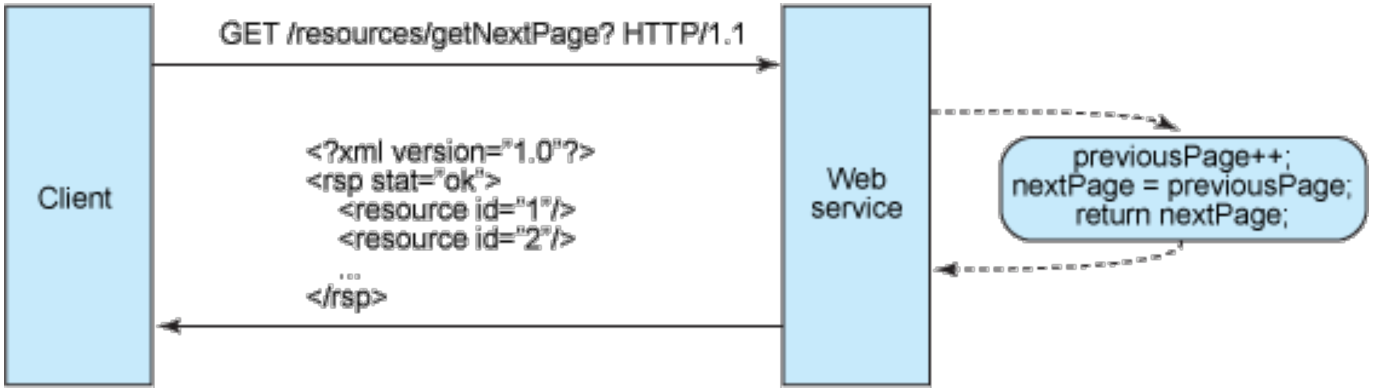
Understanding REST (2)

- 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)

- 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.

Stateful vs. Stateless Design

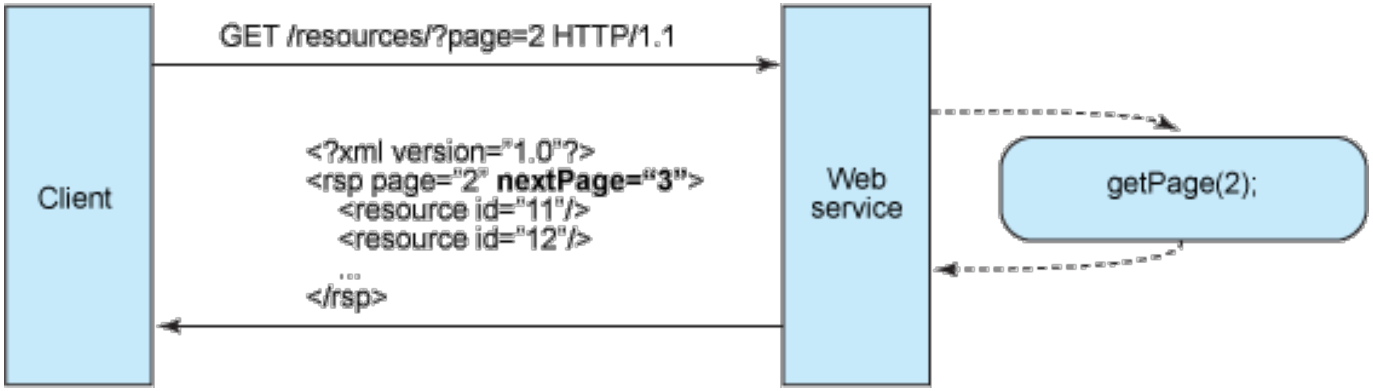


```
GET /resources/getNextPage? HTTP/1.1
```

```
<?xml version="1.0"?>
<rsp stat="ok">
   <resource id="1"/>
   <resource id="2"/>
   ...
</rsp>
```

```
previousPage++;
nextPage = previousPage;
return nextPage;
```

Stateful design: Server keeps Client-related state (previousPage)



```
GET /resources/?page=2 HTTP/1.1
```

```
<?xml version="1.0"?>
<rsp page="2" nextPage="3">
   <resource id="11"/>
   <resource id="12"/>
   ...
</rsp>
```

```
getPage(2);
```

Stateless design: the Client has to include explicit state information in the call, the Server does not keep any client-related state

Uniform Interface

- The semantics of the methods is constant and independent from the particular resource that is being addressed
- Evolution of services goes via new resources rather than calls
- Simplifies integration of services from different providers (service mashing)
- Simplifies error recovery by end systems and intermediate systems (caches) who can repeat idempotent calls automatically without having access / understanding of the “payload”
- Accompanied by a set of “Unified Status Codes”

Uniform Interface (2)

	GET	PUT	DELETE	POST
Resource URL e.g. *http://shop.oreilly.com/product/9780596529260.do*	**Retrieve**	**Create / Replace**	**Delete**	**Append / Modify**
	Safe Idempotent Cacheable	Idempotent	Idempotent	Not safe Not idempotent

Unified Status Codes

```
GET /product/9780596529260.do HTTP/1.1
Host: shop.oreilly.com
```

2xx Success

```
HTTP/1.1 200 OK
Content-Type: text/html;charset=UTF-8
...
```

3xx Redirect

```
HTTP/1.1 304 Not Modified
ETag: “1234567890”
```

4xx Client Error

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Basic
...
```

5xx Server Error

```
HTTP/1.1 500 Internal Server Error
...
```

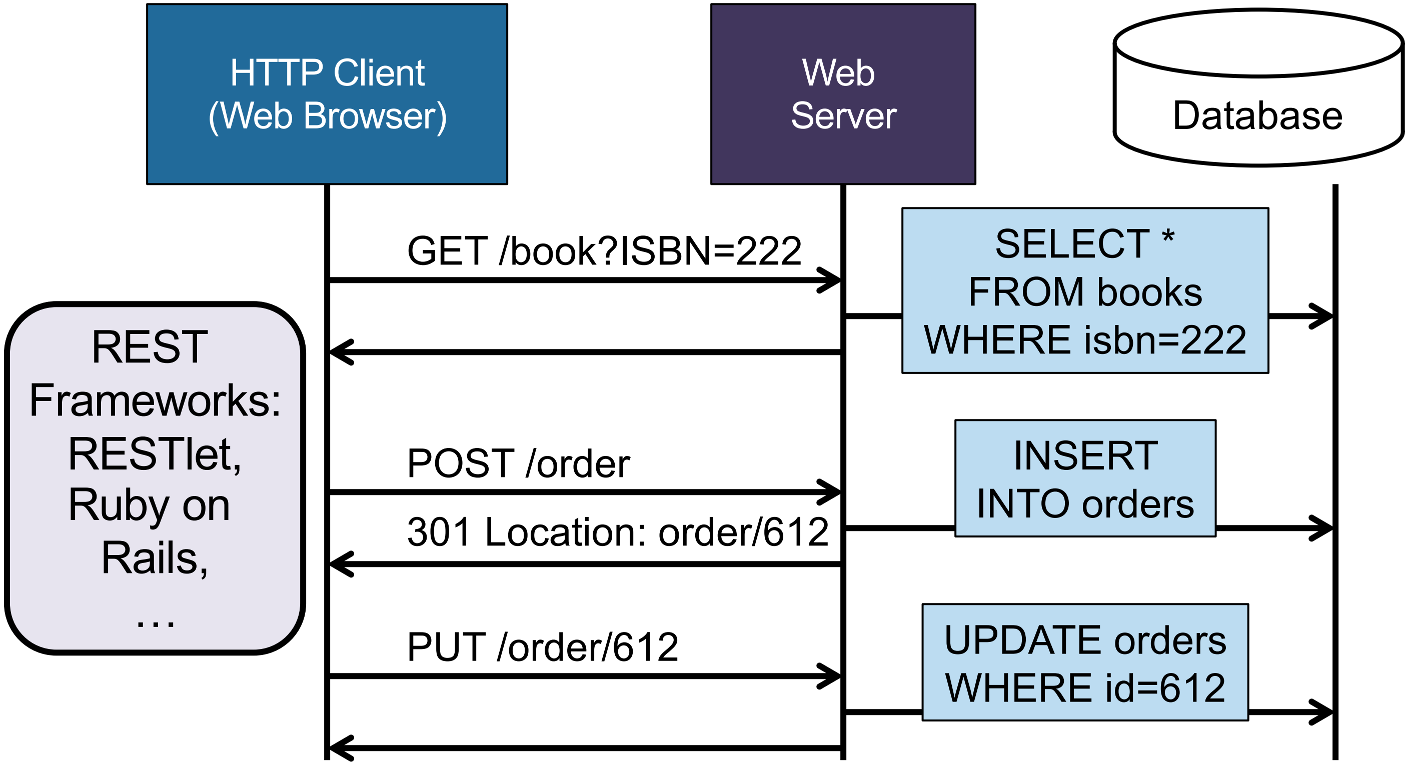
Narrow vs. Wide Interfaces

- The usage of just the HTTP methods in REST is an example of a more general class of service designs based on “narrow” interfaces
- Narrow interface service design
  - Limited set of common methods / procedures / functions
  - Services defined through the resources / arguments passed to the narrow method interface
  - Service design is focused on the resources, not on the functions

Narrow vs. Wide Interfaces (2)

- Examples:
  - UNIX: “Everything is a file descriptor”
    - Methods: create, open, read, write, sync, seek, close
    - Resource: a “file”, device, socket, etc. , addressed by a file descriptor
  - SNMP: Simple Network Management Protocol
    - Methods: snmpGet, snmpGetBulk, snmpGetList, snmpSet, etc.
    - Resource: managed “objects” identified by object identifiers (OIDs), organized in hierarchical Management Information Base (MIB)

REST - Example



Summary: Comparison of SOAP and REST

Features	SOAP-Style	REST-Style
Interaction	Stateful	Stateless
Interface	Specified by description language (WSDL, IDL)	Uniform interface
Data Format	Specified by description language (WSDL, IDL)	MIME Types (negotiation)
URI	Service	Resource
Payload	Opaque message	Self-described message
Performance	Hard to cache	Easy caching