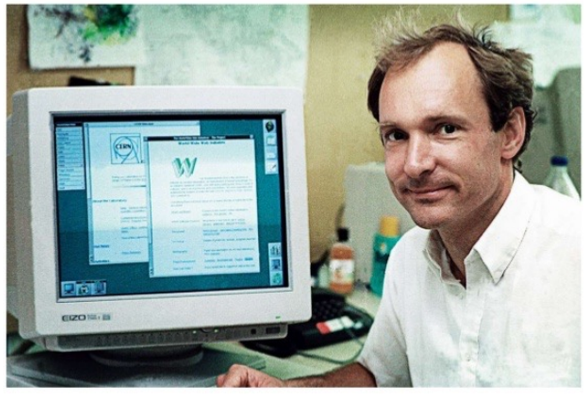
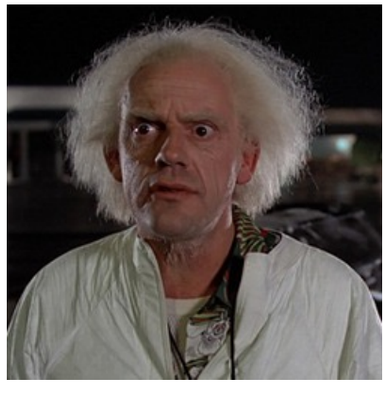
CERN

Tim Berners Lee

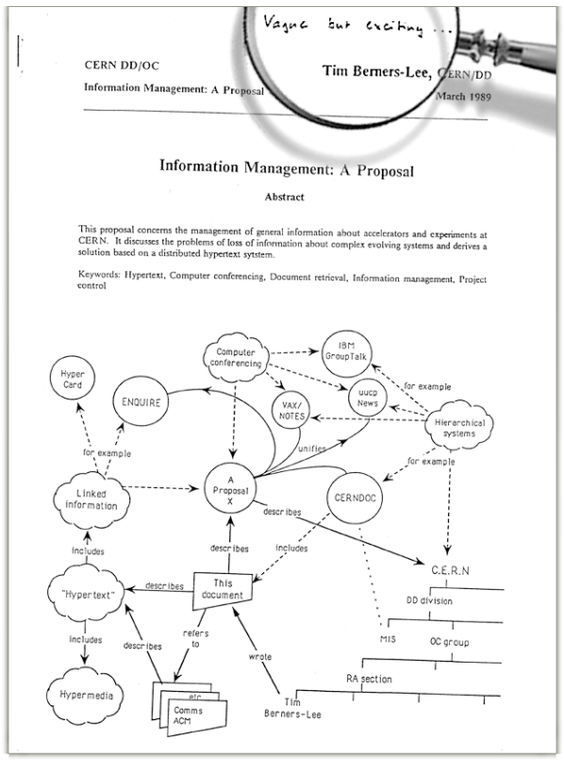


Genius 1



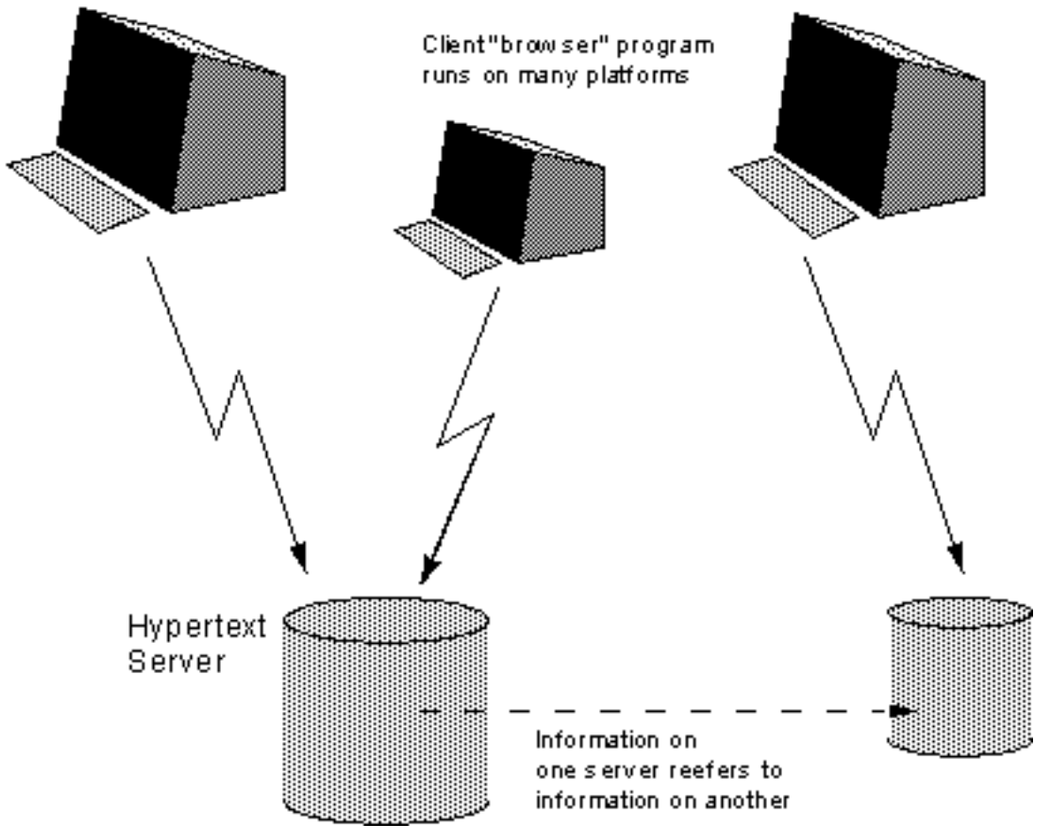
Genius 2

Draft for Information System in CERN

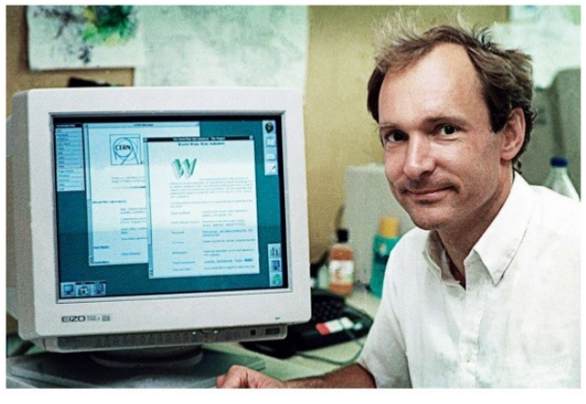


https://www.w3.org/History/1989/proposal.html

Boss (Mike Sendal) replied: “Vague but Exciting”



Tim Berners Lee



Needs something over
TCP to send information (**HTTP**)

How to structure
information (**HTML**)

Web Services —— Part 1

Syed Gillani
Laboratoire Hubert Curien St-Etienne, France

[HTTP: HyperText Transfer Protocol]

What happens when you click/enter a URL?

- Client opens a **TCP/IP connection** to host and sends request
  - ✓ GET /filename HTTP/1.0
  - ✓ GET URL
- Server returns **header info** and **HTML**

- URL Format:
  - ✓ *service://hostname/filename?otherstuff*

[HTTP: HyperText Transfer Protocol]

- Request:

  Request line: method (GET, POST) object(/filename) protocol (HTTP)

  Headers: many options, most optional

  empty line

  message body (optional)
- Example methods:

  GET: retrieval

  POST: submitting data to be processed (in body)
- Mandatory header:

  HOST: URL sending request to

[Example from Wikipedia entry for HTTP]

- **Request:**
  GET /index.html HTTP/1.1

  Host: www.example.com

- **Response:**
  HTTP/1.1 200 OK
  Date: Mon, 23 May 2005 22:38:34 GMT
  Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
  Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT

  Etag: “3f80f-1b6-3e1cb03b"
  Accept-Ranges: bytes
  Content-Length: 438
  Connection: close
  Content-Type: text/html; charset=UTF-8

  text of page

[HTTP: HyperText Transfer Protocol]

- Original design of HTTP just returns text to be displayed
  - ✓ now includes pictures, sounds, video, etc.
  - ✓ need helper plug-ins to display non-text content e.g. GIF, JPEG, sounds

- Form filled in by users
  - ✓ needs a program on the server to interpret the information (cgi-bin)

- HTTP is stateless
  - ✓ server does not remember anything from one request to another
  - ✓ need a way to remember information on the client : **cookies**

- Active content for client:
  - ✓ Javascript and other interpreters
  - ✓ Java applets
  - ✓ Plug-ins
  - ✓ Active X

Recall from Previous Lecture (Human vs Applications)

Can application understand HTML from the HTTP requests?

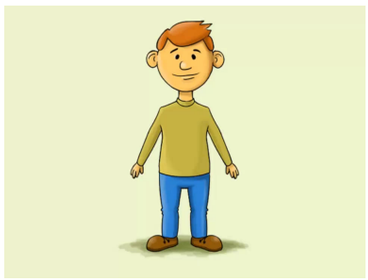
[Web Services]

A layer on top of the existing Web

- A Web Service (WS) is:
  - ✓ a software system
  - ✓ designed to support interoperable machine-to-machine interaction over a network
  - ✓ it interfaces (public) are described in XML (e.g. WSDL)
  - ✓ other systems can interact with the WS as prescribed by the interface description
  - ✓ uses XML-based (e.g. SOAP) messages

[Why Web Services? Use Case?]

A lot of manual work!!



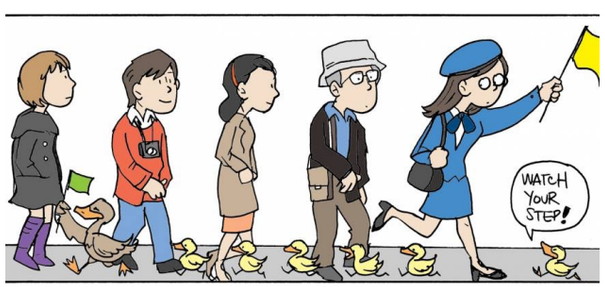
Get me out of here

Book Hotel

Book Flights

Book Taxis

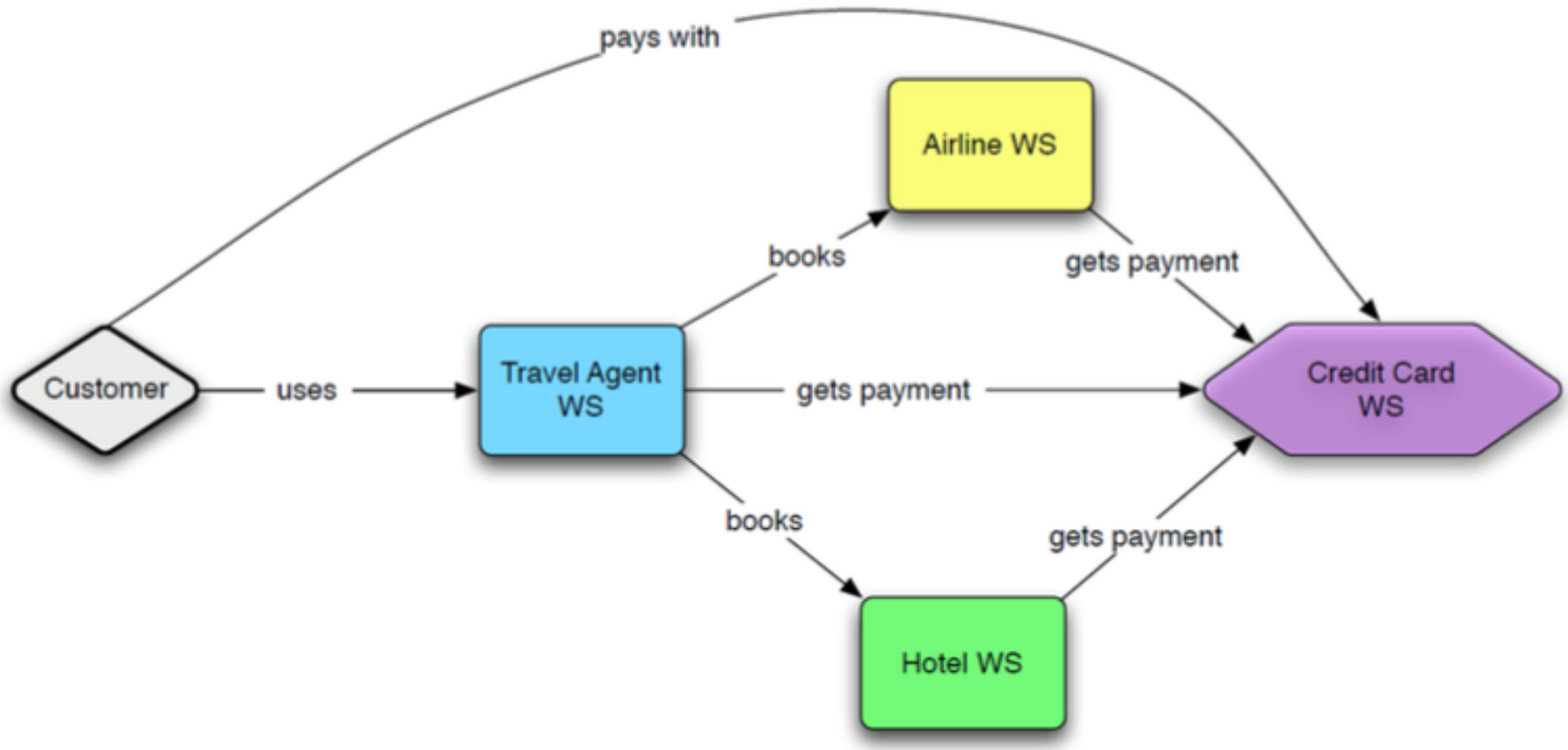
Book Tours



[Why Web Services? Use Case?]

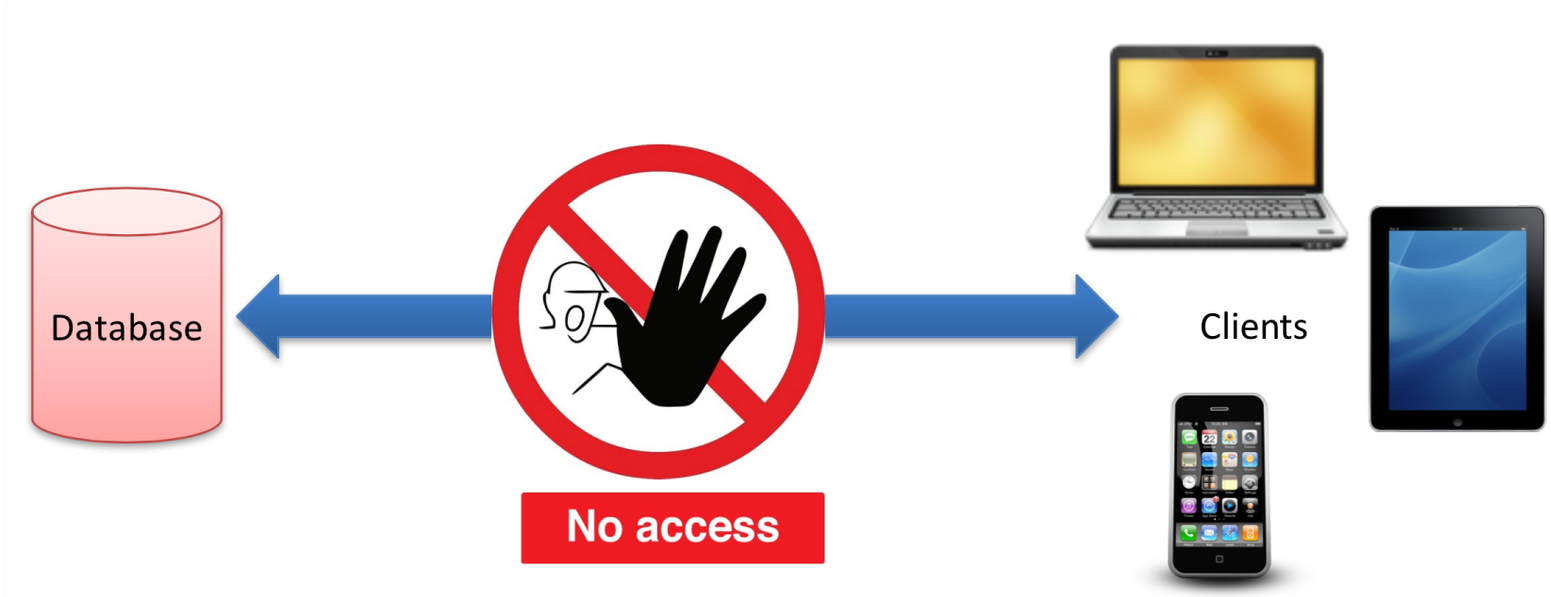
What if there is a WS for all the tasks?

- A WS for the travel agency to query and book what is required
- A Credit Card agency provides a WS to guarantee payment
- Airplane company, hotel and taxi provides their own WSs
- Only the vacationer is human; all other services are software agents



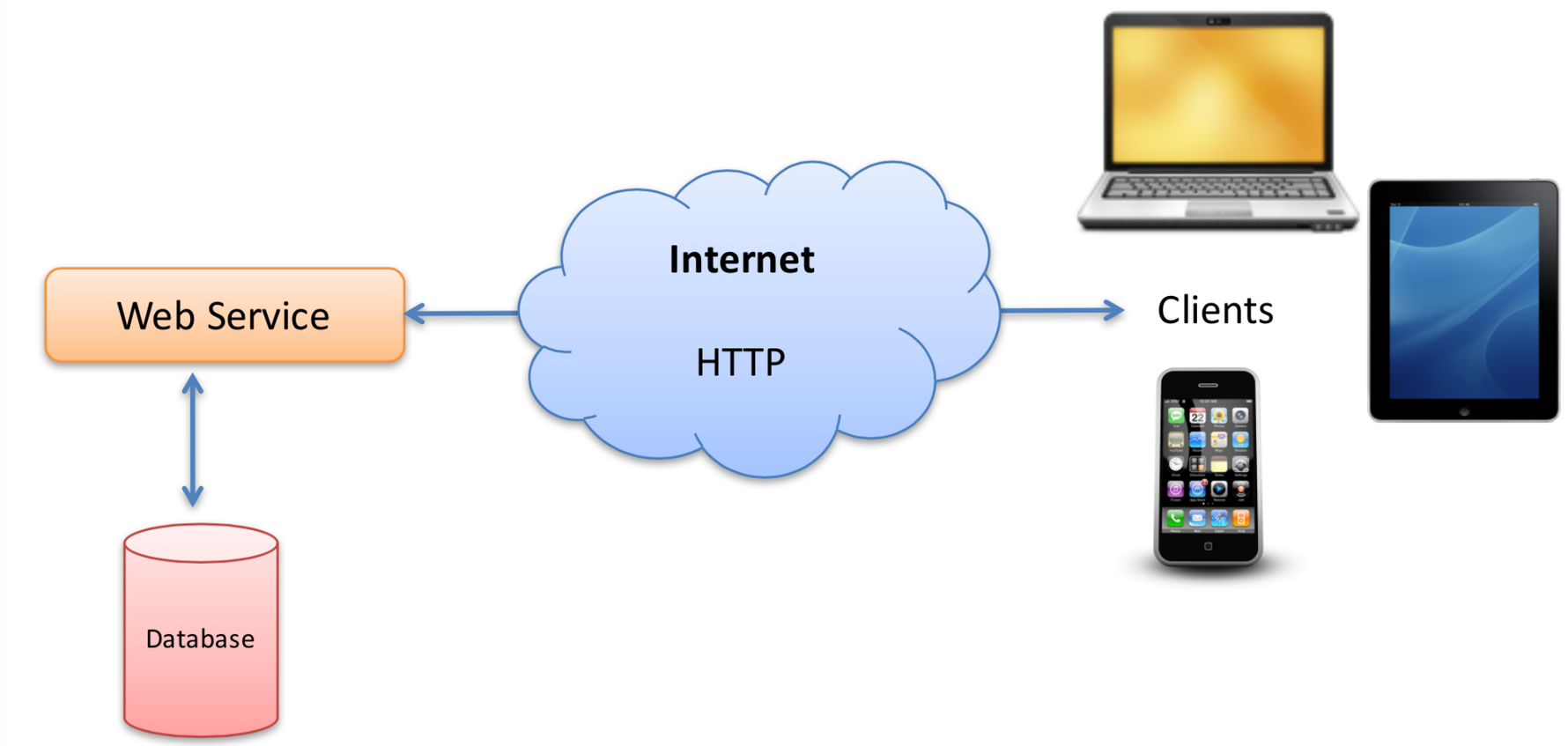
[Another View of Web Services]

How to share data between devices in a network?



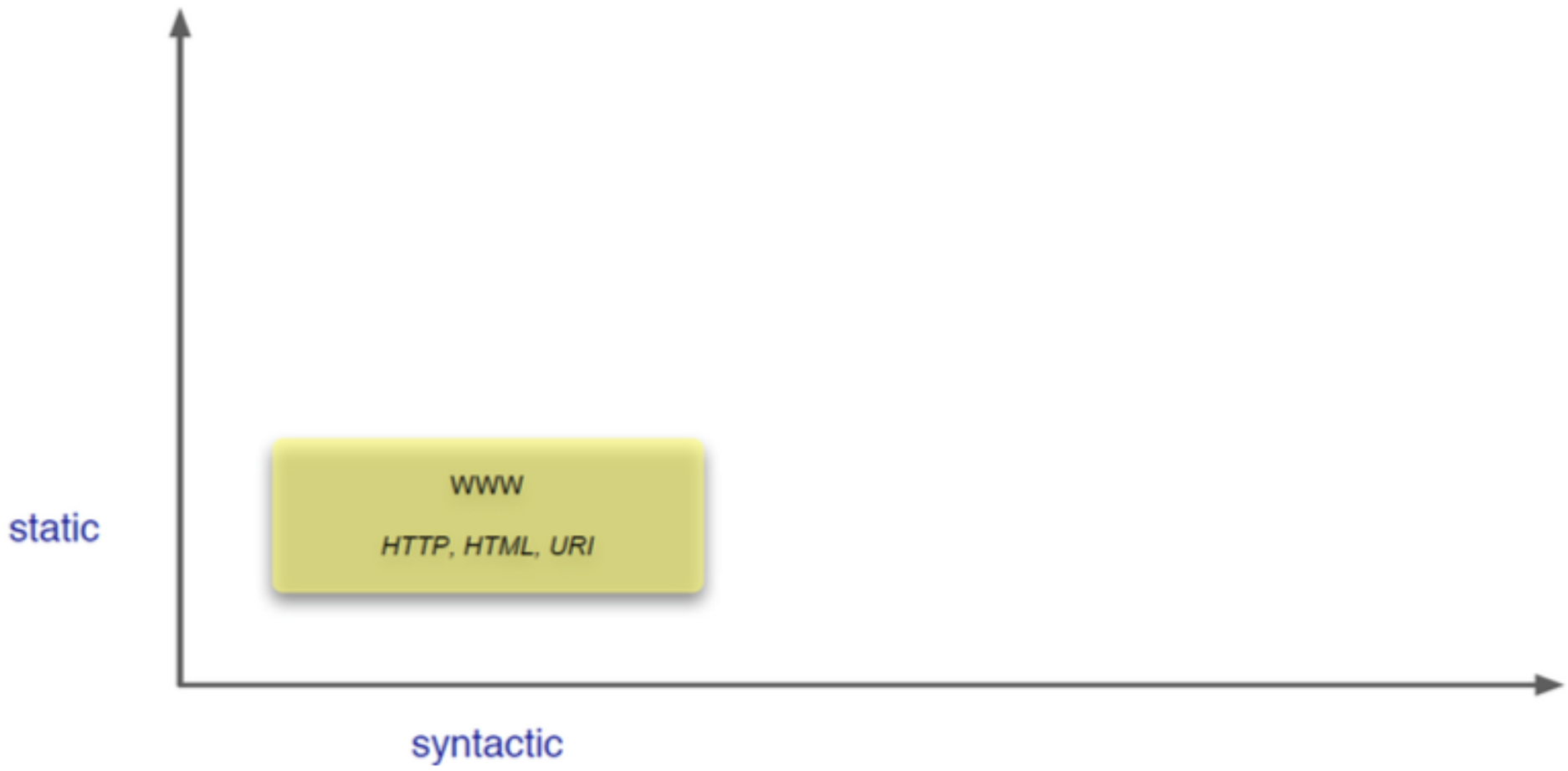
- Direct connection between the databased and the client is not suitable:
  - ✓Security, compatibility issues, Firewalls, hacking attacks…
- Direct Connection in a Local Network (behind the Firewall) is normally OK
  - – but not over the Internet

[Another View of Web Services]

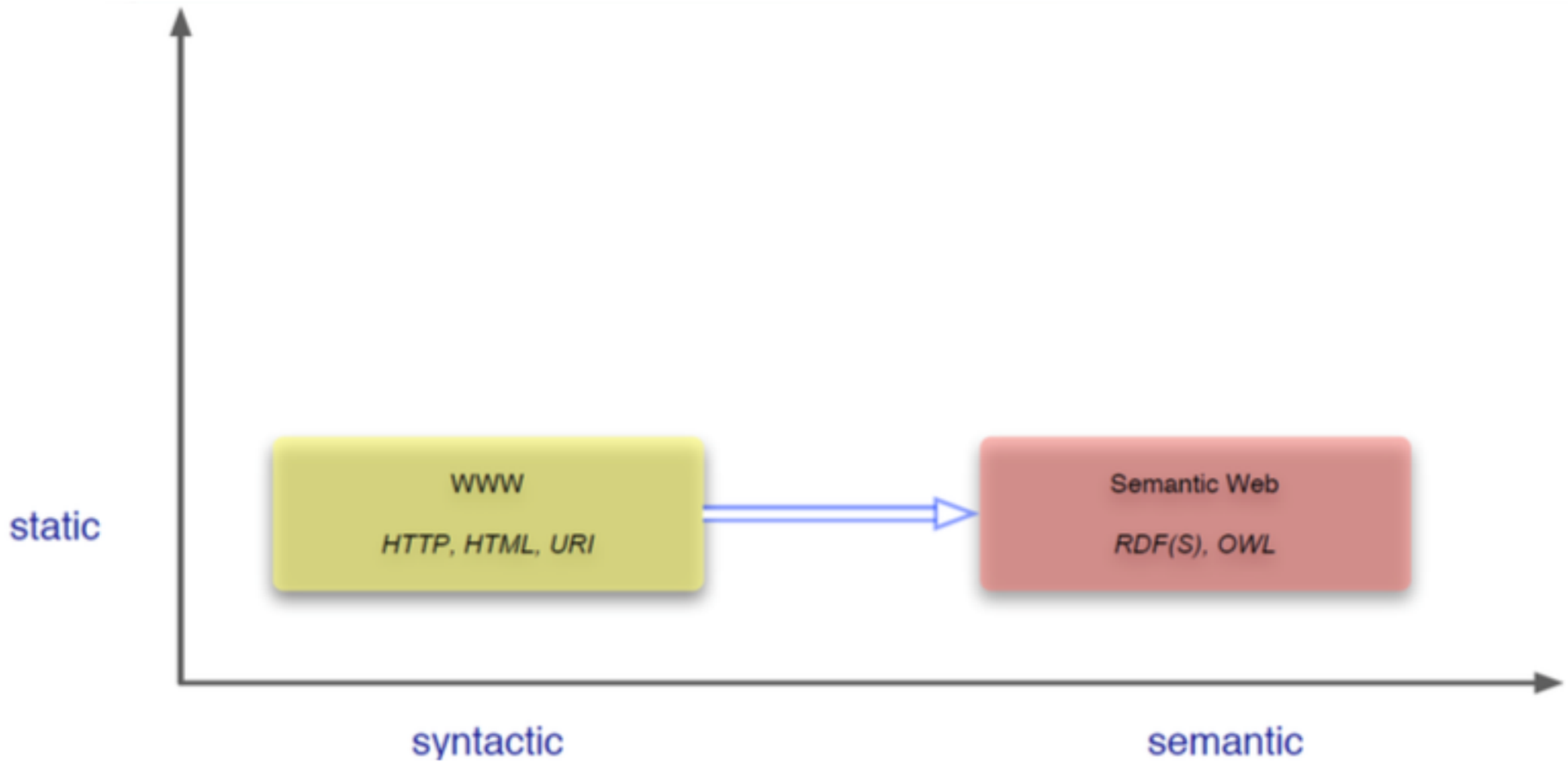


- Web Services uses standard web protocols like HTTP, etc.
- HTTP is supported by all Web Browser, Servers and many Programming Languages

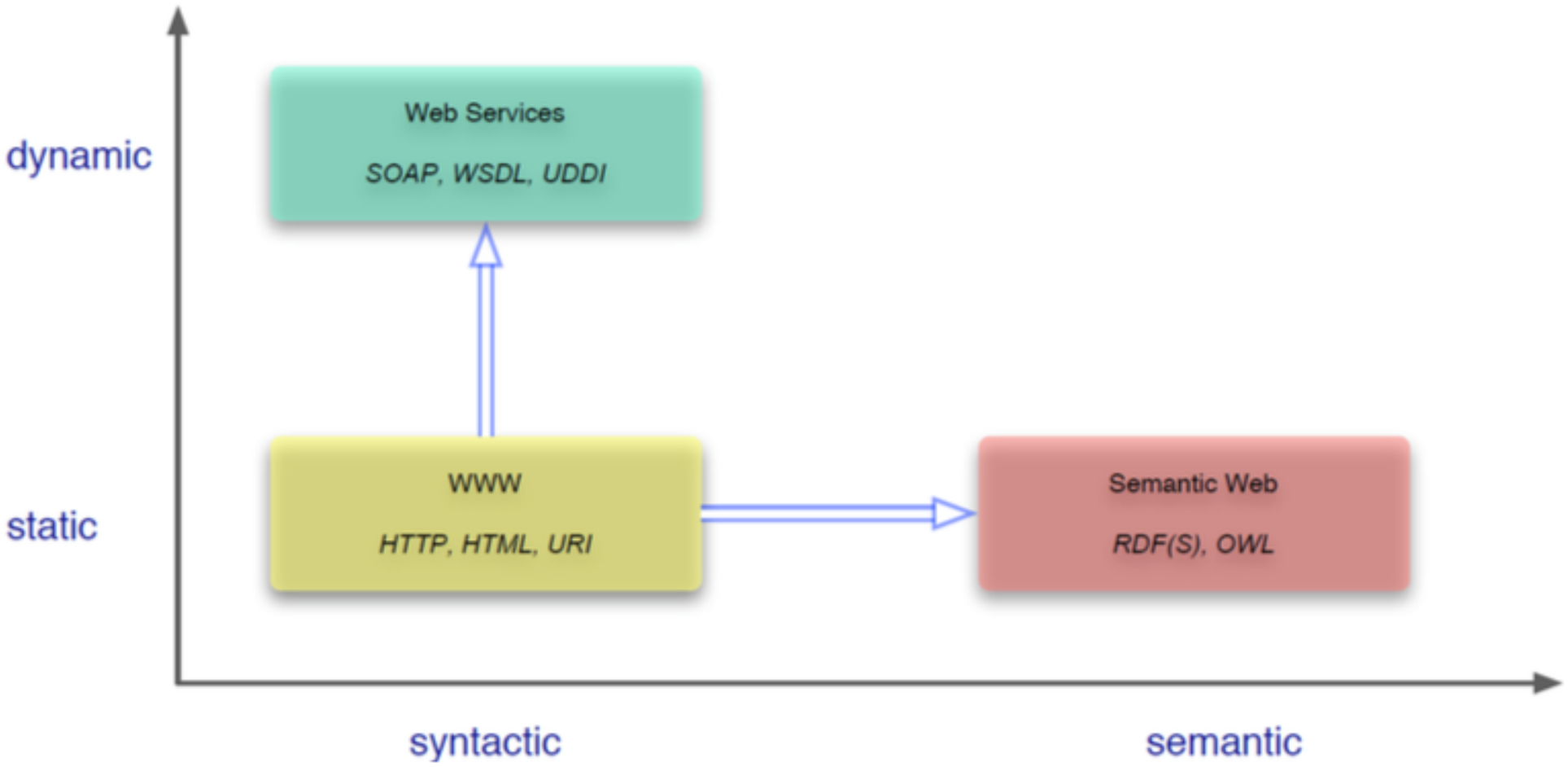
[Evolution of the WWW]



[Evolution of the WWW]



[Evolution of the WWW]



[Appeal of Web Services]

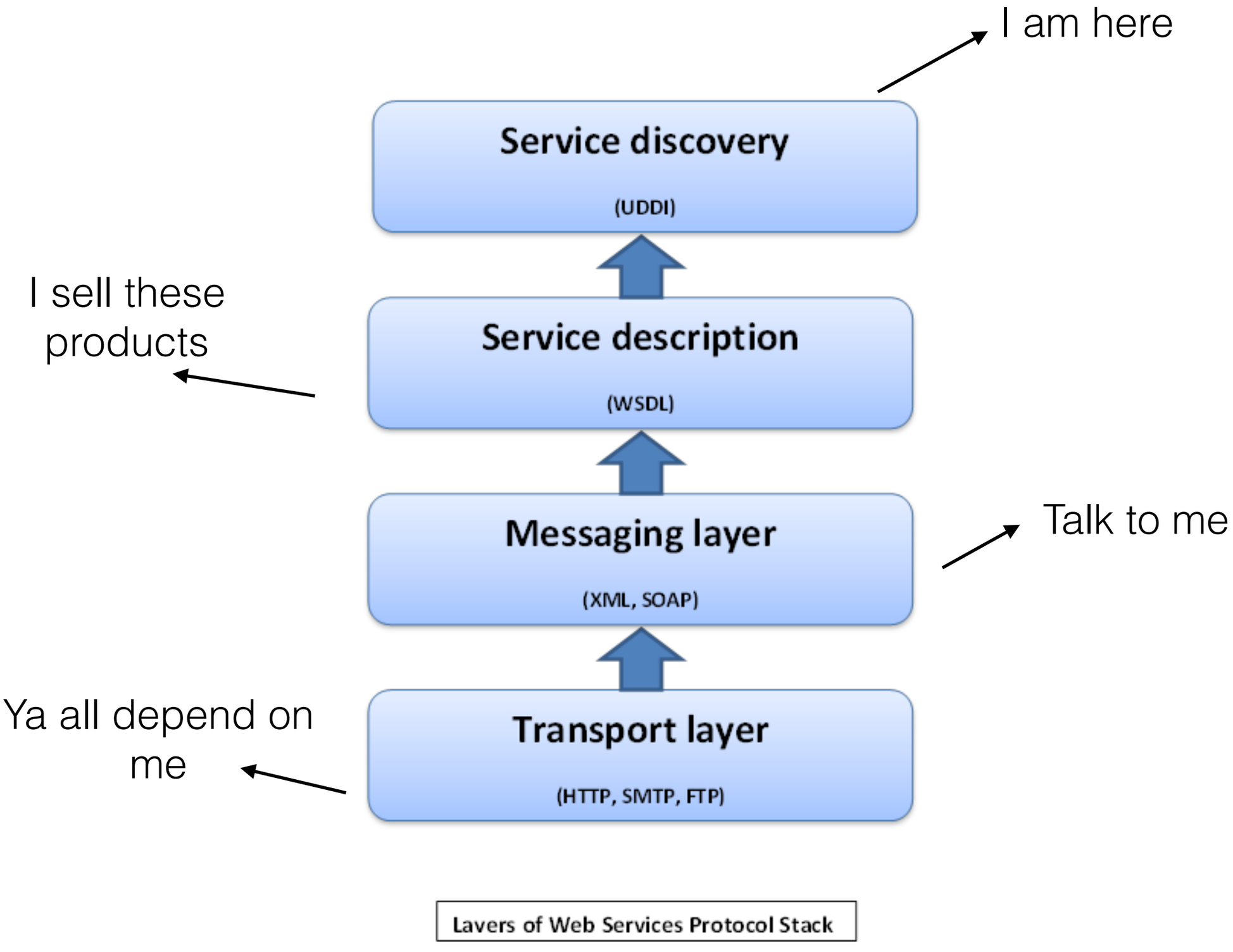
- A means of building distributed system across the internet
- Virtualisation: independent of programming language, OS, development environment
- Based on well-understood underlying transport mechanisms (e.g. HTTP)
- Components can be developed and upgraded independently
- Fairly decentralised (though issues about discovery, composition)

[Web Service (WS) Protocol Stack]

- How do WSs make them available (You can find me now)
- How to locate and access WS (Where are you?)
- How do web services interact among each other (Who are you, my name is?)
- How does a client identify a web services offerings (What are you selling?)

The answer to all these question lies in the WS protocol stack

[Web Service (WS) Protocol Stack]



Layers of Web Services Protocol Stack

[Web Service Protocol Stack]

- UDDI: Universal Description Discovery and Integration
- WSDL: Web Services Description Language
- SOAP: Simple Object Access Protocol
- SMTP: Simple Mail Transfer Protocol
- FTP: File Transfer Protocol

[Web Service (WS) Protocol Stack]

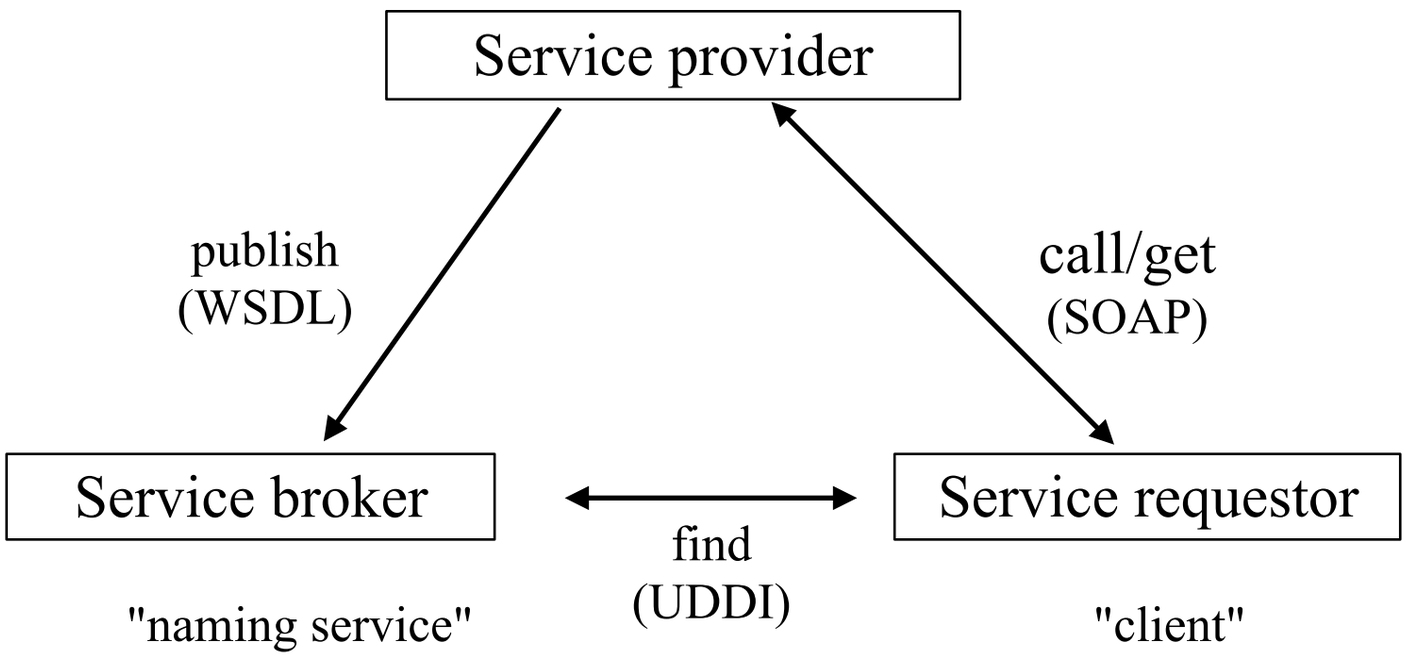
Discovery layer: centralises services with a common registry. Helps clients to look-up for available services in the network. UDDI is the standard used in this layer

Description layer: helps describe the public interface to this web service. WSDL format is commonly used for this purpose. Helps client identify the offerings of a web-service

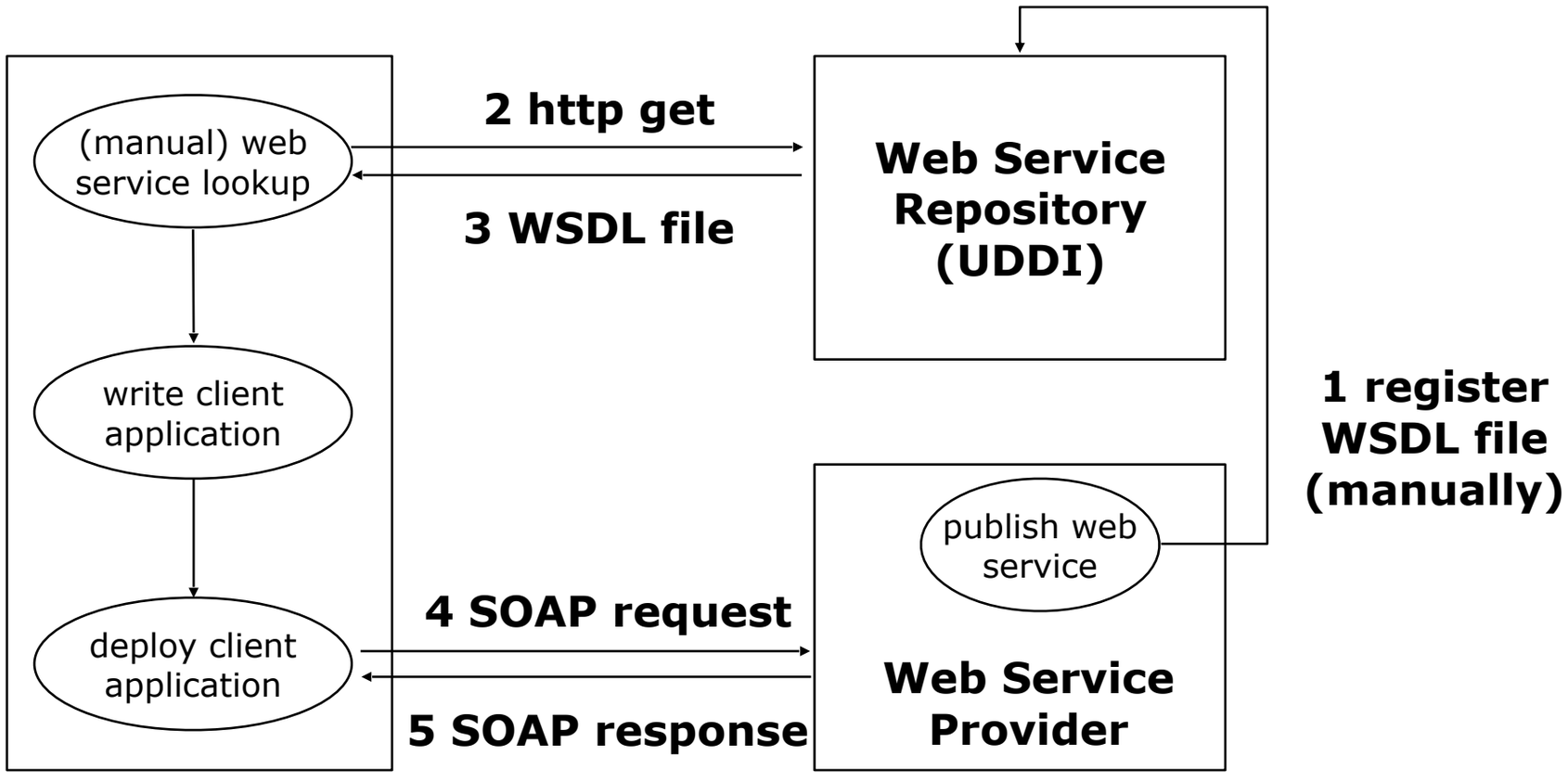
Messaging layer: responsible for encoding messages into a common format such as XML which is understood by either endpoints of a network. Common protocols used in this layer is XML-RPC, SOAP

Transport layer: transports messages between network applications and makes use of popular transport protocols such as HTTP, SMTP, FTP etc.

[Web Service Architecture]



[Basic Web Service Usage Scenario]



[Protocols and Endpoints]

Protocol: Convention that govern syntax, semantics and synchronisation of communication between computing '**endpoints**'. Enables/controls connection, communication, data transfer.

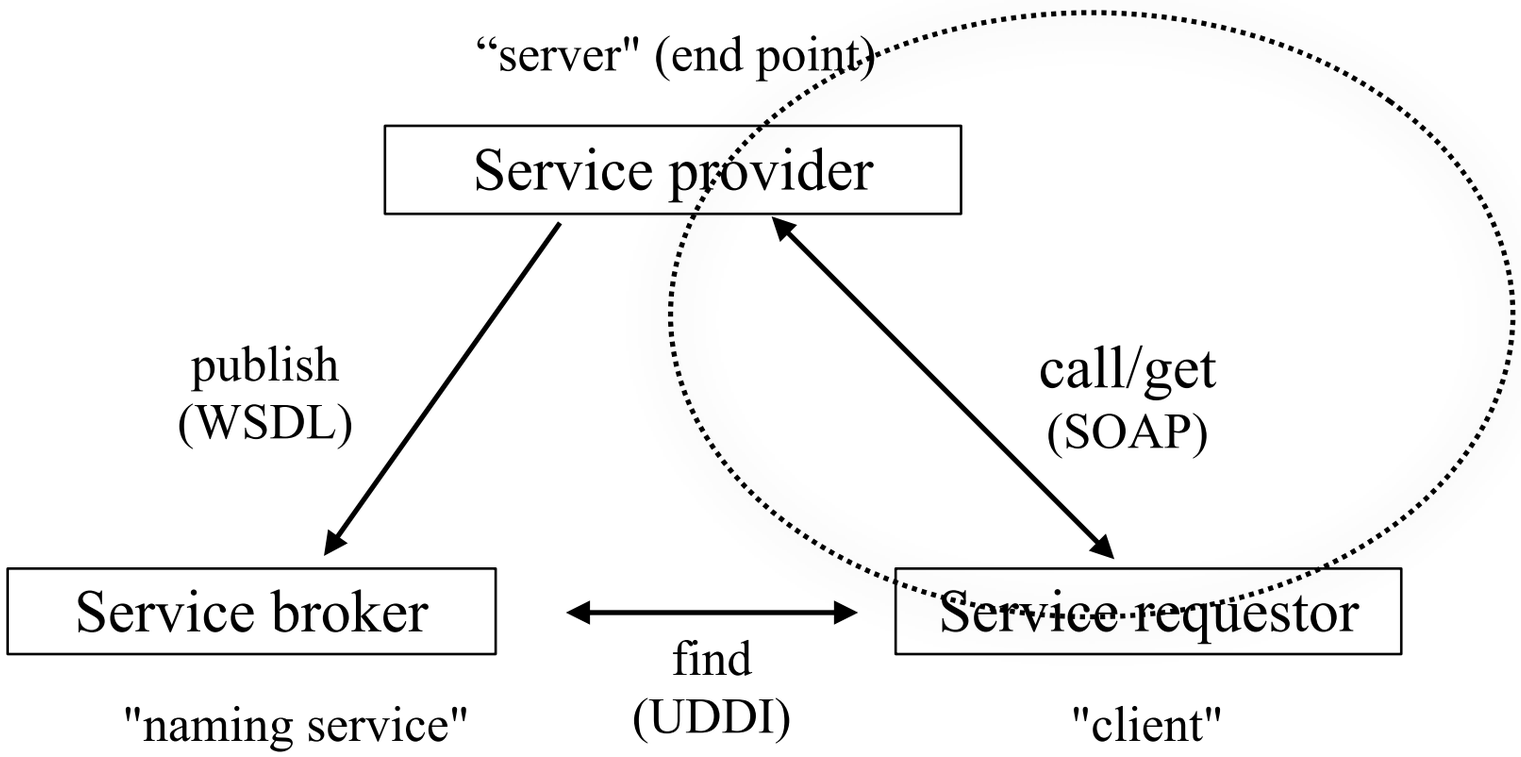
Endpoint: Endpoint is "an entity, processor or resource to which...messages can be addressed".

Endpoint Reference: Conveys the information needed to address an endpoint. Interactions may create new service instances, hence a need to dynamically create new endpoint references.
(cf. http://www.w3.org/TR/ws-addr-core/)

[Intermission]

- Server (endpoint) can describe what it can do (through WSDL)
- But how a server executes the actions a client asked for?

A variety of different things happens at the server but mainly it employs **Remote Procedure Call**



[Remote Procedure Call (RPC)]

- RPC is a protocol to allow agent on one host to cause execution of code on remote host
- Uses cline-server model of distributed computation:
  - ✓client sends message to the server
  - ✓[Execute] procedure/function P with arguments a1, a2,…an
  - ✓server executes P, and sends back the message to client
  - ✓Result [of executing P(a1,a2,…an)]

[RPC Example]

- Assume a Hotel is making reservations available as a web service
- It should expose a function/procedure **checkAvailability** which
  - Takes
    - the check-in and check-out dates
    - room type as input parameter, and
  - Returns price in Euro (€) as a floating point number

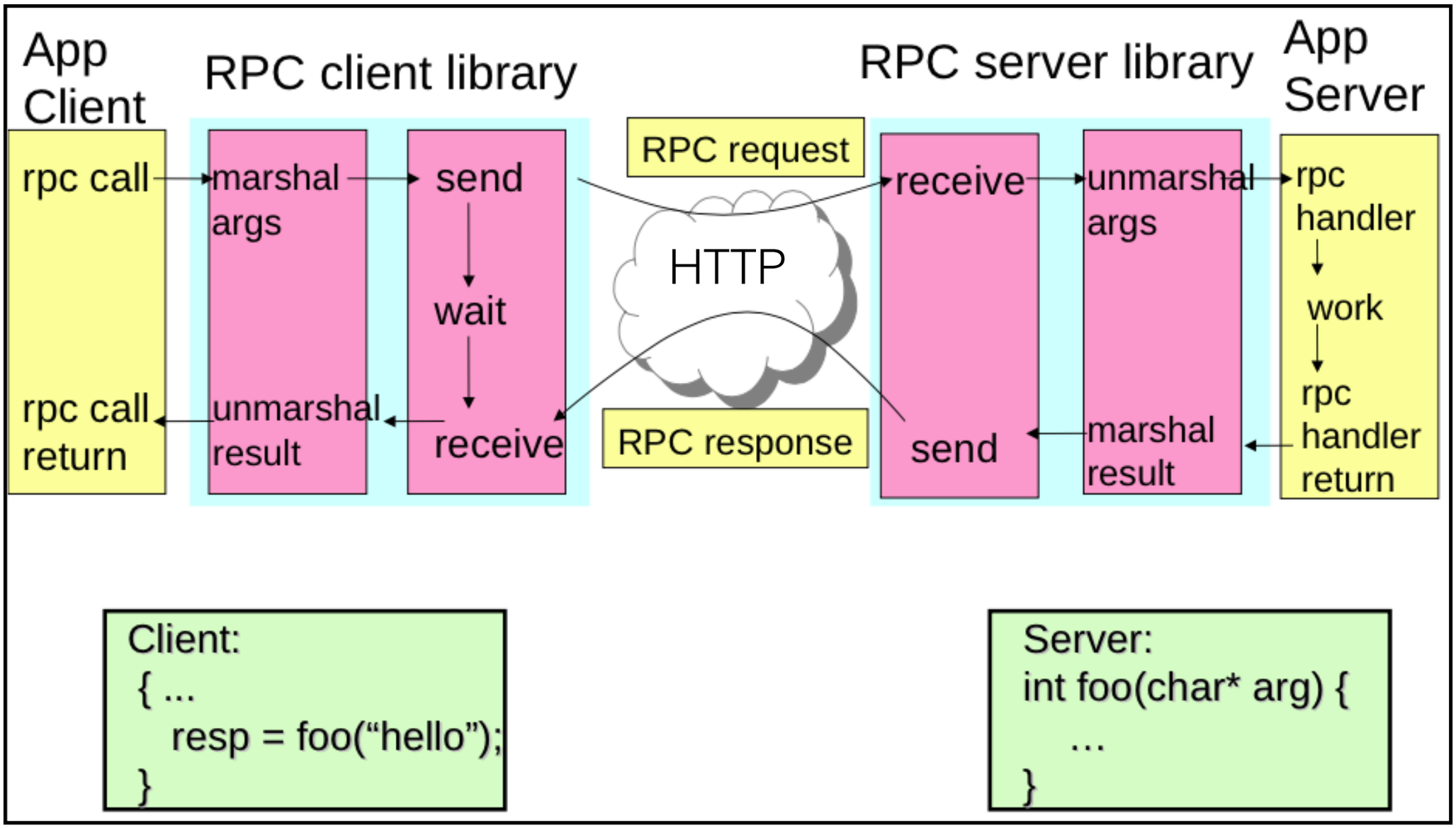
Example Function:

```
Public float checkAvailability(Date checkinDate, Date checkoutDate, String roomType)
{
if (roomAvailable(roomType)) {
return roomRateInUSD;
} else {
      return 0.0;
}
  }
```

[Why RPC?]

- Offload computation from devices to servers (Cloud Computing)
  - ✓phones, raspberry pi, sensors, etc.
- Hide the implementation details
  - ✓proprietary Algorithms
- Functions that just can’t run locally
  - ✓tags = Facebook.MatchFacesInPhoto(photo)
  - ✓print tags [person1, person 2]

[RPC Architecture]



[RPC Architecture]

- Marshalling: Converting the data or objects into byte-stream
- Unmarshalling: Reverse processing of converting byte-stream back to the original data or object

[RPC Technologies]

- **XML-RCP**

  ✓over HTTP, huge XML parsing overheads

- **SOAP**

  ✓added feature for flexibility via HTTP, huge XML overheads

- **CORBA**

  ✓relatively comprehensive, but quite complex and heavy

- **COM**

  ✓mainly for Windows client software

- **Protocol Buffers**

  ✓lightweight developed by Google

- **Thrift**

  ✓lightweight, support services, developed by Facebook

Check your this week assignment

[XML-RPC]

- Provides an XML and HTTP-based mechanism for making method or function calls across a network
- Very simple & useful

- Is a very lightweight RPC system:
  - ✓ support for elementary data types (basically, the built-in C types together with a boolean and a datetime type)
  - ✓ few simple commands
  - ✓ follows the request/response pattern

- Two key features:
  - ✓ use XML marshalling/unmarshaling to achieve language neutrality
  - ✓ reliance on HTTP for transport

[XML-RPC]

- Recall from previous lecture:
  - ✓ XML is a standard for describing structured documents
  - ✓ Uses tags to define structure: <tag> … </tag> demarcates an element
  - ✓ Elements can have attributes, which are encoded as name-value pairs

- XML-RPC simply uses the mapping of objects into XML
- Mainly consists of data types, structs and arrays

[XML-RPC Data Types]

- Scalar values
  - ✓ Represented by a **<value><type> … </type></value>** block
- Integer
  - ✓ **<i4>12</i4>**
- Boolean
  - ✓ **<boolean>0</boolean>**
- String
  - ✓ **<string>Hello world</string>**
- Double
  - ✓ **<double>11.4368</double>**
- Also Base64 (binary), DateTime, etc.

These data types are used for the arguments of a function/method invoked by a client

[XML-RPC Struct]

- Structures:
  - ✓Represented as a set of <member>s
  - ✓Each member contains a <name> and a <value>

```
<struct>
 <member>
   <name>lowerBound</name>
   <value><i4>18</i4></value>
 </member>
 <member>
   <name>upperBound</name>
   <value><i4>139</i4></value>
 </member>
</struct>
```

[XML-RPC Arrays]

- Arrays:
  - ✓A single <data> element, which
  - ✓contains any number of <value> elements

```
<array>
  <data>
    <value><i4>12</i4></value>
    <value><string>UK</string></value>
    <value><boolean>0</boolean></value>
    <value><i4>-31</i4></value>
  </data>
</array>
```

[XML-RPC Request Structure]

- Requests are a combination of XML content and HTTP headers.
- The XML content
  - ✓ uses the data typing structure to pass parameters
  - ✓ contains information identifying the procedure being called
- HTTP headers provide a wrapper for passing the request over the Web
- A request contains a single XML document,
  - ✓ whose root element is a **methodCall/FunctionCall**
  - ✓ each **methodCall** element contains a methodName element and a params element

[XML-RPC Request Structure]

- XML-RPC call to add(17,13) results in this request

```
Post /xmlrpc HTTP 1.0
User-Agent: myXMLRPCClient/1.0
Host: 192.168.124.2
Content-Type:text/xml
Content-Length:169
```

HTTP Header

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <methodCall>
     <methodName>sample.add</methodName>
    <params>
            <param>
             <value><int>17</int></value>
              </param>
          <param>
              <value><int>13</int></value>
           </param>
          </params>
</methodCall>
```

XML-RPC Request

[XML-RPC Response Structure]

- Much like the requests, but we distinguish two possible cases:
  - ✓ If the response is successful :
    - the procedure was found, executed correctly, and returned results
  - ✓ If there was a problem in processing the XML-RPC request
    - The result will contain a fault message

[XML-RPC Response Structure]

- The response will look like the request
  - ✓ The methodCall element is replaced by a methodResponse element
  - ✓ There is no methodName element

```
<?xml version="1.0" encoding="ISO-8859-1"?>
 <methodResponse>
     <params>
          <param>
           <value><int>30</int></value>
           </param>
     </params>
</methodResponse>
```

Remark: XMLRPC response can only contain one parameter (this parameter can be an array or struct)

[XML-RPC Code Example]

Apache's XML/RPC Java Library Example

(http://www.tutorialspoint.com/xml-rpc/xml_rpc_examples.htm)

1.a) The client-side code:

```
import org.apache.xmlrpc.*;

public class JavaClient {
  public static void main(String [] args) {
    try {
      XmlRpcClient server = new XmlRpcClient("http://localhost/RPC2");

      Vector params = new Vector();
      params.addElement(new Integer(17));
      params.addElement(new Integer(13));

      Object result = server.execute("sample.add", params);
      int sum = ((Integer)result).intValue();
      System.out.println("The sum is: "+ sum);
    } catch (Exception exception) { // …
    }
  }
}
```

1.b) The server-side code:

```
import org.apache.xmlrpc.*;

public class RPCHandler {
  public Integer add(int x, int y) {
    return new Integer(x + y);
  }

  public static void main (String[] args) {
    try {
      WebServer server = new WebServer(80);
      server.addHandler("sample", new RPCHandler());  // register the handler class
      server.start();
    } catch (Exception exception) { // …
    }
  }
}
```

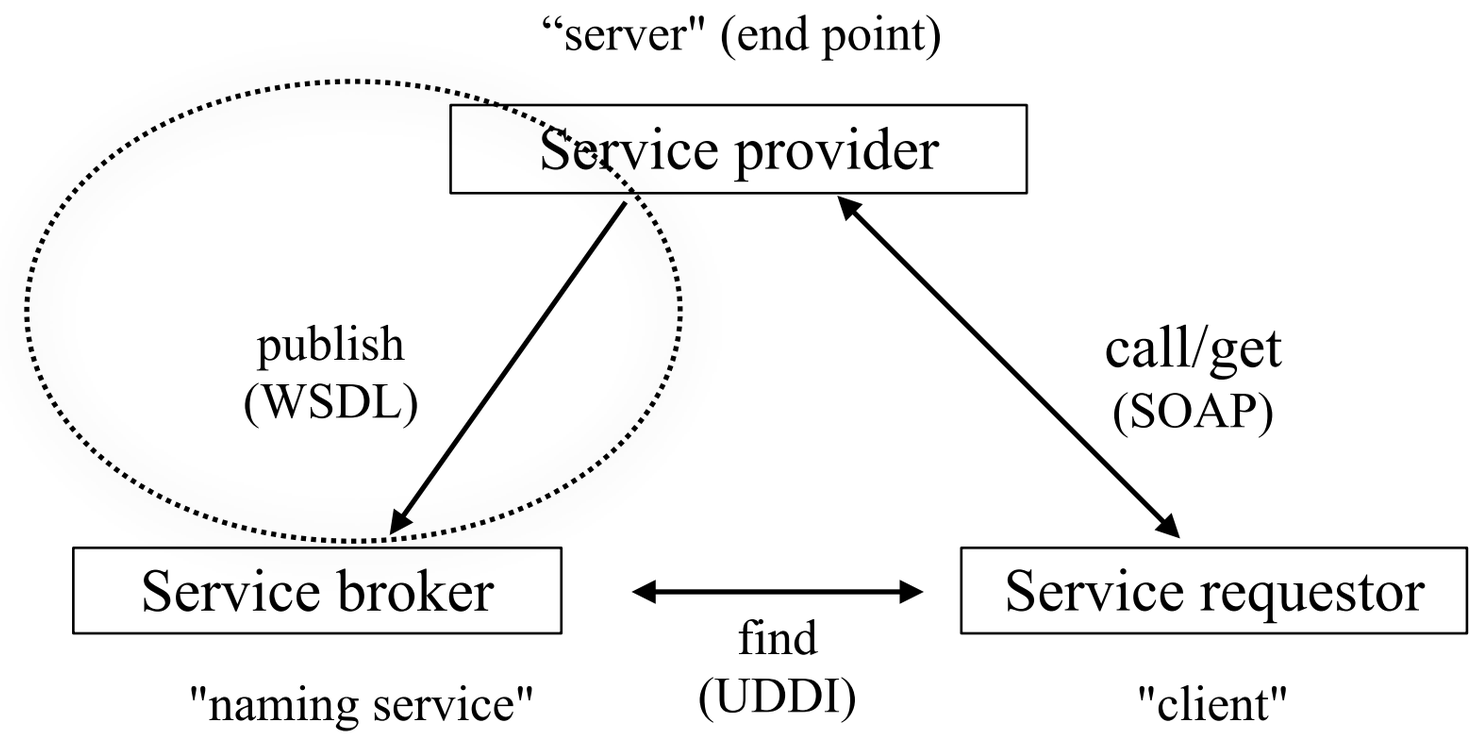
1.c) XML Marshaling:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<methodCall>
   <methodName>sample.add</methodName>
   <params>
       <param>
            <value><int>17</int></value>
       </param>
       <param>
             <value><int>13</int></value>
       </param>
   </params>
</methodCall>
```

[XML-RPC Limitations]

- XML is extremely verbose, which affects performance
- Complex data types are not supported
- Does not exploit HTTP well
  - ✓ all the XML-RPC responses uses **200 ok** response code, even if a fault is contained in the message
- Most of the XML-RPC libraries does not support protocol versioning (i.e. updates)
  - ✓ what if I want another param?
  - ✓ what happens if I reverse the order of x and y in method call?
  - ✓ what if I forgot to add parameters?
  - ✓ In general, lack of data types makes it difficult to build and maintain code

[Interlude]



How to publish methods for RPC calls?

[Web Service Metadata]

- Metadata is data/information about the data
  - ✓ e.g. **Bloomfield, Leonard. 1933. Language. New York: Holt, Rinehart & Winston**.
  - ✓ The reference above is information about a book — that is, data about data.

- Need two kinds of metadata about services

✓ **Operational**:
- service category (e.g. hotel room booking)
- informal description (e.g. this service books hotels)
- information about the provider entity (e.g. name, contact, etc.)

✓ **Non-Operational**:
- service interface (browsers compatibility)
- communication protocol (SOAP, XML-RPC)
- service endpoints (e.g. QoS, cost, etc.)

Operational metadata is standardly expressed using Web Service Description Language (WSDL)

[Web Services Description Language (WSDL)]

- It represents a contract between the service provider and the client
  - ✓ similar to a Java Interface but the difference is : WSDL is platform independent
- Using WSDL, a client can locate a web service and invoke any of its publicly available functions

It provides

- ✓ **Interface information:** describing all publicly available functions
- ✓ **Data type information:** for all message requests and message responses
- ✓ **Binding information:** about the transport protocol to be used
- ✓ **Address information:** for locating the specified service

[Web Services Description Language (WSDL)]

- Uses XML format for describing all the information needed to communicate and invoke a web service

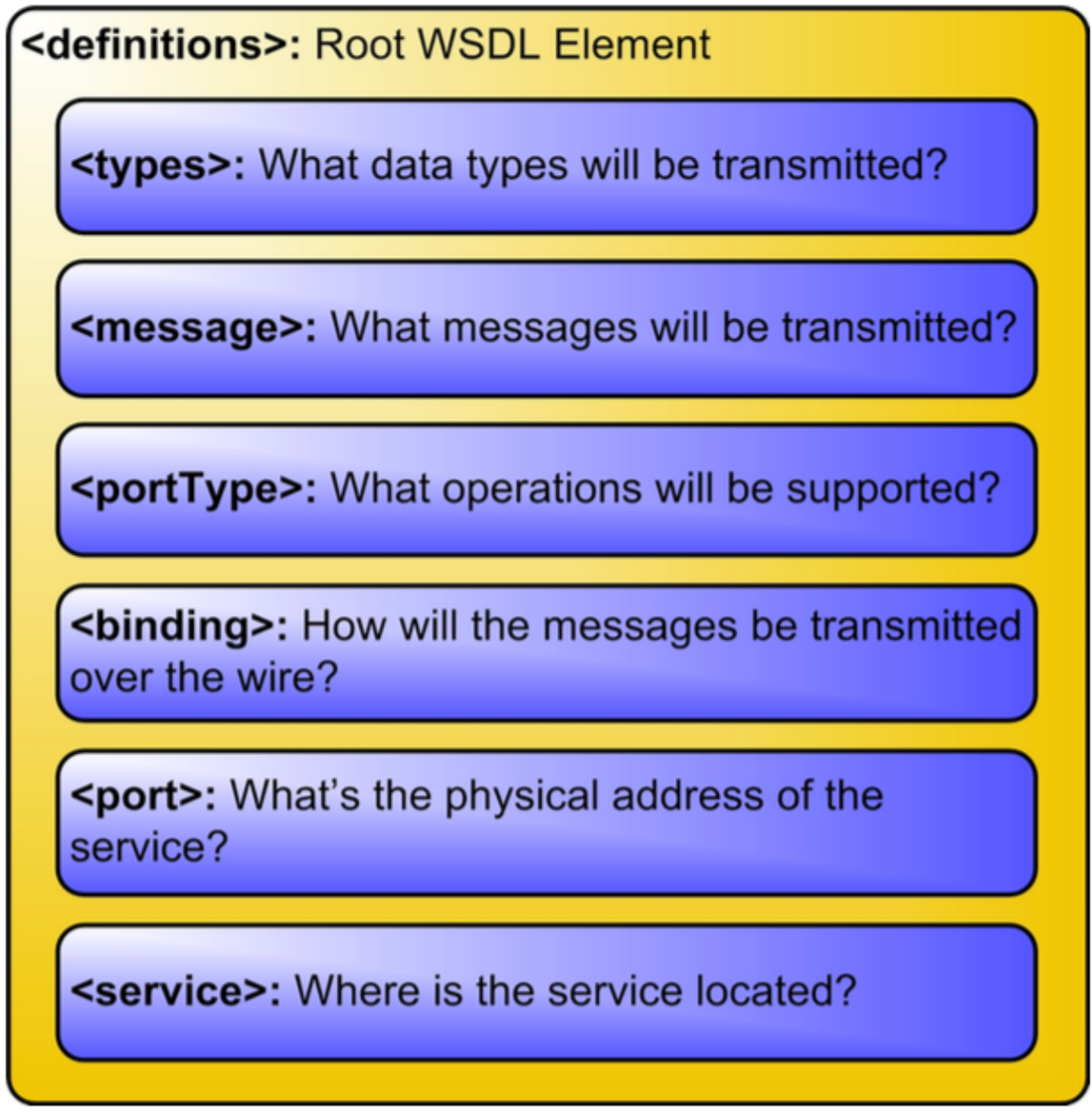
- **Functional/Procedural Description**:
  - ✓ how the web service is invoked, where it is invoked, syntax of the message, and configuration of protocol to deliver the message

- **Non-functional Description**:
  - ✓ other details, such as security policy, additional header information

[WSDL Document Structure]

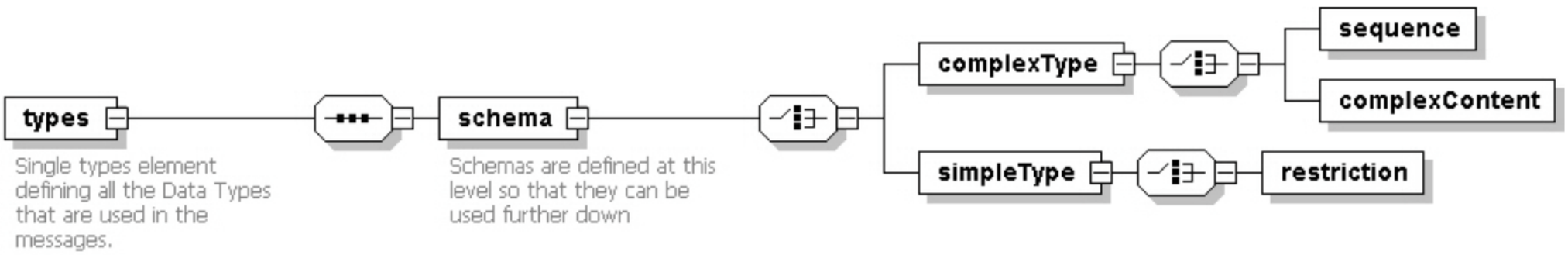
A WSDL Document is a set of definition with a single root element.
Services can be defined using the following XML elements

- **Types**: data types (string, integer, etc.)
- **Messages:** Methods (getPrice), arguments
- **PortType:** Interfaces (Java interface)
- **Binding:** Encoding schemes (Serialisations)
- **Port:** URL of the service
- **Service:** Many URLs



[Type Element]

- Data Types of the function/method call's argument (same as in XSD)
- A WSDL document can have at most one types element
- A type element can contains simple Type or complex Type

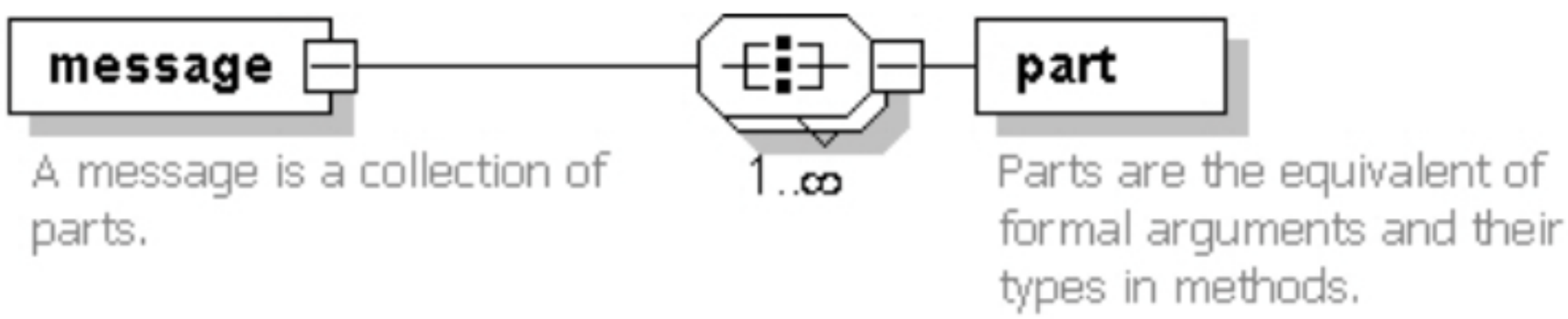


[Type Element:Example]

```
<!-- Type Definitions -->
<types>
  <xsd:schema targetNamespace="http://weather.com/ns"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:complexType name="HumidityType">
      <xsd:sequence>
        <xsd:element name="loc" type="xsd:string">
        <xsd:element name="humd" type="xsd:double">
        <xsd:element name="temp" type="xsd:double">
      </xsd:sequence>
    </xsd:complexType>
  </xsd:schema>
</types>
```

[Message Element]

- A message is a collection of parts; intuitively a part is a named argument with its type
- A WSDL document can contain zero or more message elements
- Each message element can be used as an input, output or fault message within an operation

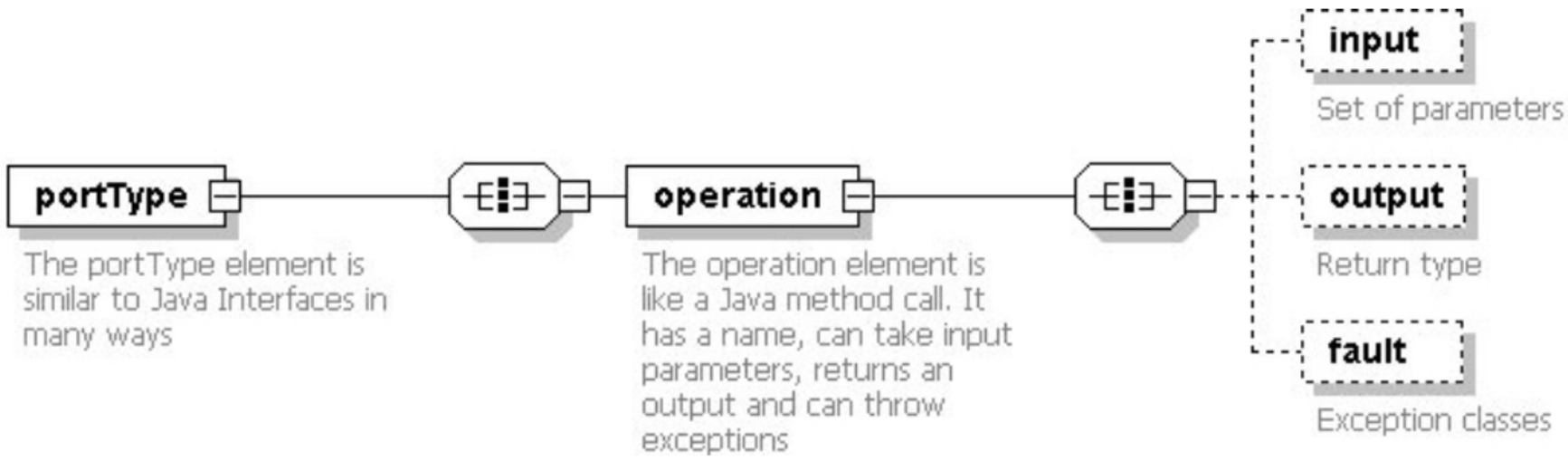


[Message Element: Example]

```
<!-- Message Definitions -->
<message name="checkTemperatureRequest">
    <part name="location" type="xsd:string">
</message>
<message name="checkTemperatureResponse">
    <part name="result" type="xsd:double">
</message>
<message name="checkHumidityRequest">
    <part name="location" type="xsd:string">
</message>
<message name="checkHumidityResponse">
    <part name="result" type="ns:HummidityType"
</message>
```

[Port Type Element]

- Use XML format for describing all the information needed to communicate and invoke a web service
- A portType element contains a single name attribute
- A portType element also contains one or more operation elements, with a name attribute that contains input, output and fault elements



[Port Type Element: Example]

```
<!-- Port Type Definition Example -->
<portType name="weatherCheckPortType">
  <operation name="checkTemperature">
    <input message="checkTemperatureRequest"/>
    <output message="checkTemperatureResponse"/>
  </operation>
  <operation name="checkHumidity">
    <input message="checkHumidityRequest"/>
    <output message="checkHumidityResponse"/>
  </operation>
</portType>
```

[Binding Element]

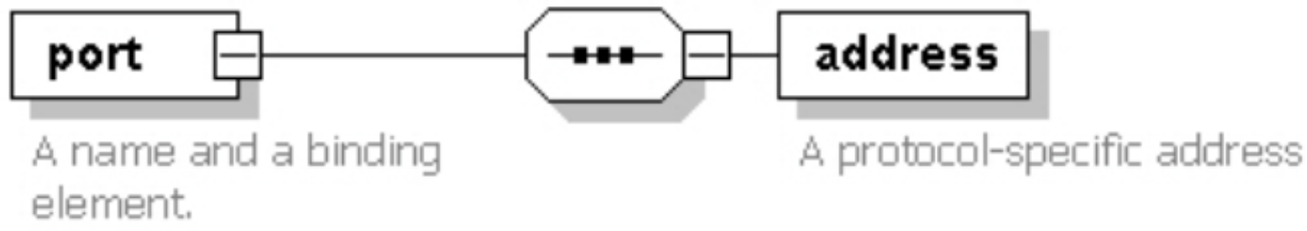
- The binding element specifies to the service requester how to format the message in a protocol-specific manner
- Each portType can have one or more binding elements associated with it
- For a given portType the binding element has to specify a messaging and transport part (HTTP/SMTP/ SOAP(will see this later))

[Binding Element: Example]

```
<binding name="WeatherBinding" type="weatherCheckPortType">
  <soap:binding style="rpc"
    transport="http://schemas.xmlsoap.org/soap/http" />
  <operation name="checkTemperature">
    <soap:operation soapAction="" />
    <input>
      <soap:body use="encoded" namespace="checkTemperature"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </input>
    <output>
      <soap:body use="encoded" namespace="checkTemperature"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
    </output>
  </operation>
</binding>
```

[Port Element]

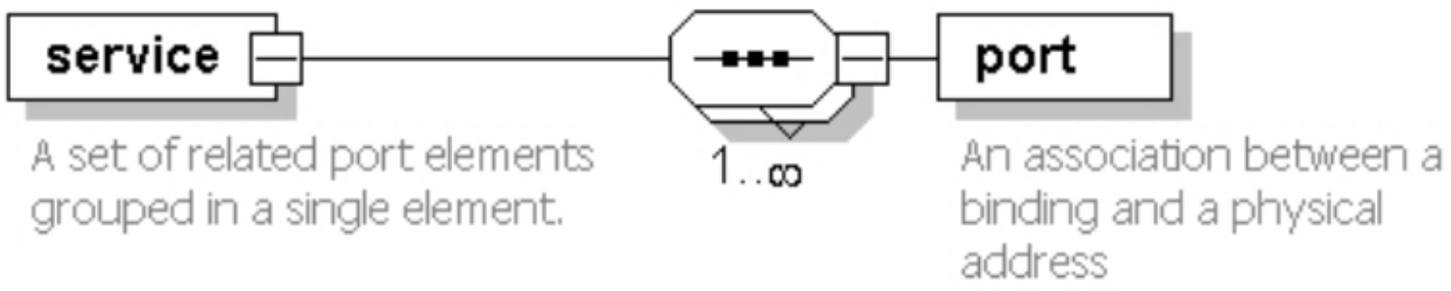
- The port element specifies the network address of the endpoint hosting the web service
- It associate a single protocol-specific address to an individual binding element
- Ports are named and must be unique within the document




```
<port name="WeatherCheck"
      binding="wc:WeatherCheckSOAPBinding">
  <soap:address location="http://host/WeatherCheck"/>
</port>
```

[Service Element]

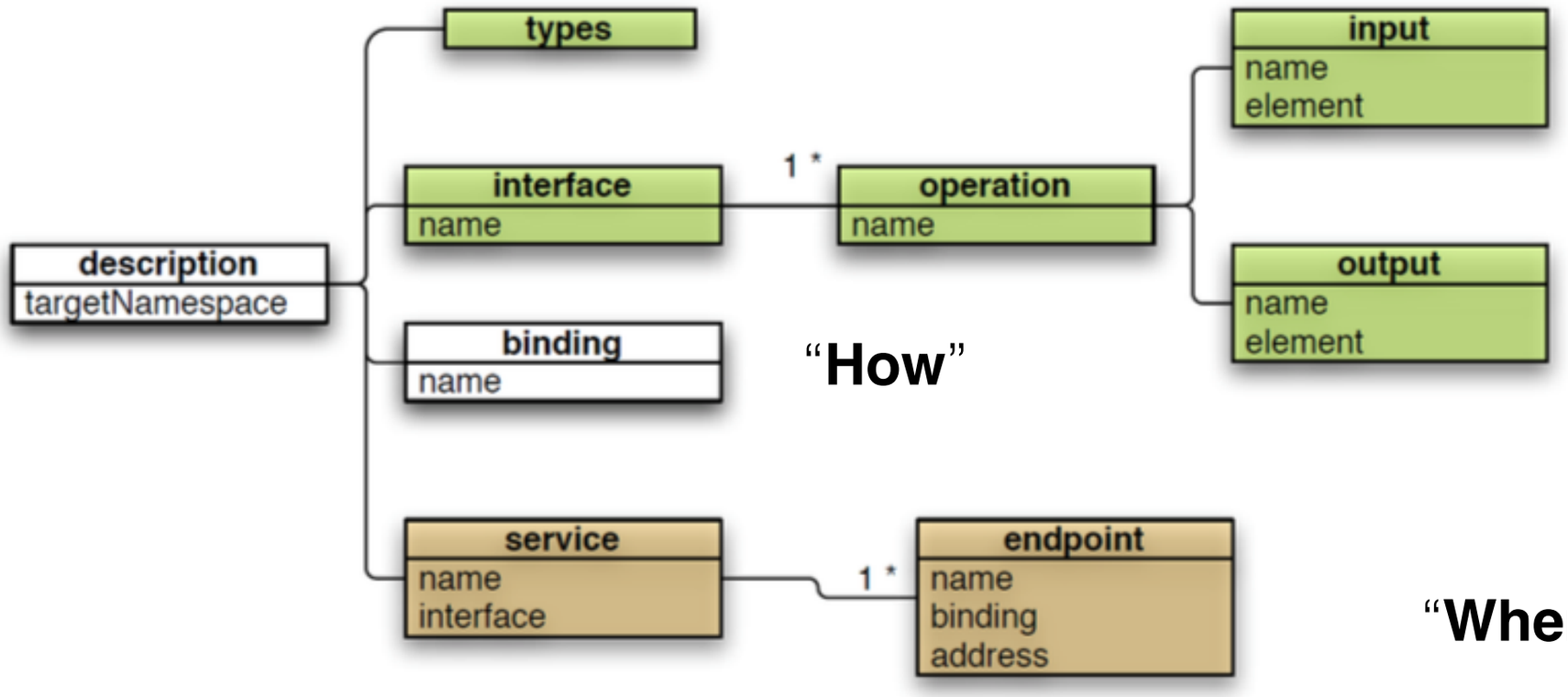
- The service element is a collection of related port elements identified by a single service name
- A WSDL document is allowed to contain multiple service elements, but conventionally contains a single one
- Each service must be uniquely named



[Service Element: Example]

```
<!-- Service definition -->
<service name="WeatherCheckService">
  <port name="WeatherCheckSOAP"
      binding="wc:WeatherCheckSOAPBinding">
  <soap:address location="http://host/WeatherCheck"/>
  </port>
  <port name="WeatherCheckSMTP"
      binding="wc:WeatherCheckSMTPBinding">
  <soap:address location="http://host/WeatherCheck"/>
</port>
</service>
```

[WSDL]

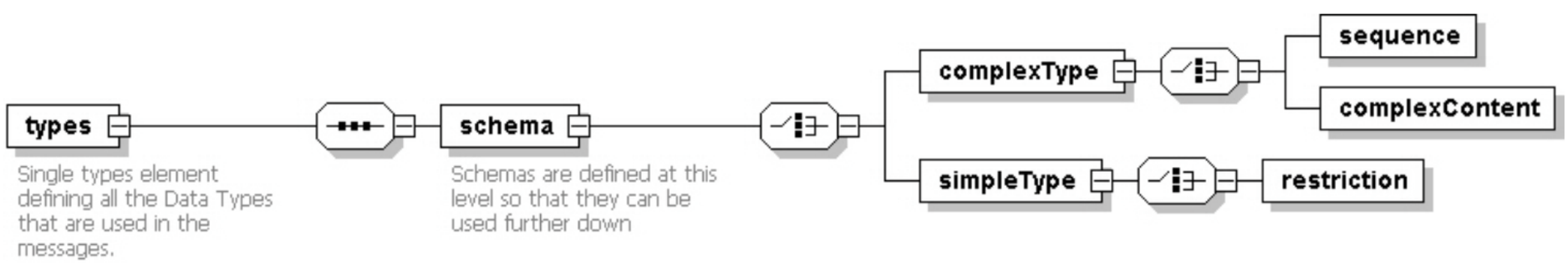


http://www.w3.org/TR/wsdl20-primer

DIY: Check whats new in WSDL 2.0

[WSDL Exercise]

- Create the **Types** Part of WSDL
- It should expose a function/procedure **checkAvailability** which
  - Takes
    - the check-in and check-out dates
    - room type as input parameter, and
  - Returns price in Euro (€) as a floating point number



[WSDL Exercise]

```
<types>
…
<xs:element name =“checkAvailability” type=“tCheckAvailability”/>
<xs:complexType name =“tCheckAvailability”>
  <xs:sequence>
    <xs:element name=“checkinDate” type=“xs:date”/>
    <xs:element name=“checkOutDate” type=“xs:date”/>
    <xs:element name=“roomType” type=“xs:string”/>
  </xs:sequence>
</xs:complexType>
<xs:element name =“checkAvailabilityResponse” type=“xs:double”/>
</types>
```

[Fin]

Acknowledgements: Jacques Fleuriot, University of Edinburgh; Ioannis Baltopoulos, University of Cambridge; Web Service Essentials by Ethan Cerami