MONASH University

Information Technology

FIT5183: Mobile and Distributed Computing Systems (MDCS)

Lecture 2B
Web Services Technology

References

1. Gustavo Alonso, Fabio Casati, Harumi Kuno, Vijay Machiraju, Web Services Concepts, Architectures and Applications, 2004, Springer.

2. https://www.vs.inf.ethz.ch/edu/WS0405/VS/VS-050124.pdf

3. http://www.w3schools.com/schema/

4. http://docs.oracle.com/cd/E19651-01/817-2151-10/wsgoverview.html

Outline

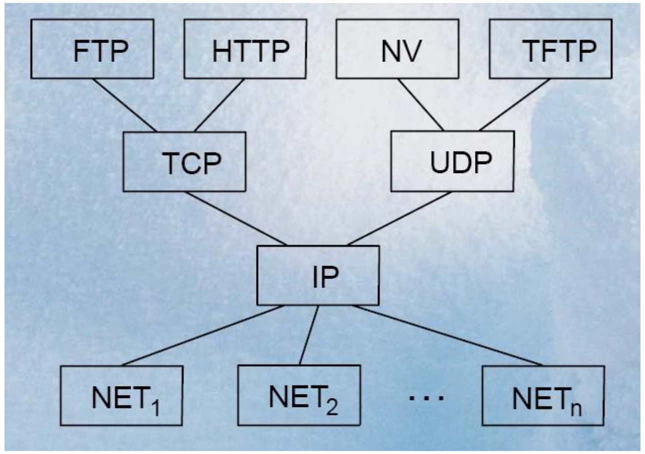
- ❑ The Internet and World Wide Web, revisited
- ❑ Distributed Systems Before the Web
- ❑ Overview of Web Services Technology
- ❑ Web services and SOA
- ❑ SOAP Web Service Definitions

Adapted from
http://archive.systems.ethz.ch/www.systems.ethz.ch/education/past-courses/fs10/web-services-and-soa.html

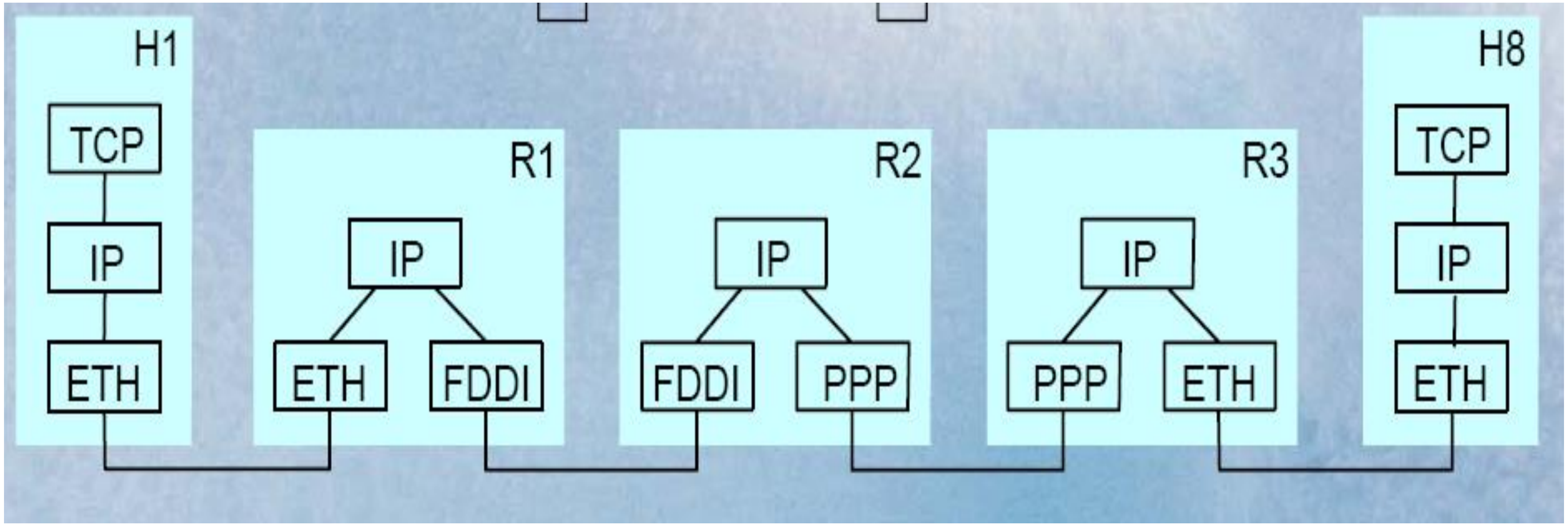
By Prof. Gustavo Alonso, ETH Zurich

TCP/IP Protocol and Hourglass Design

- IP protocol, at the network layer, for networking, addressing and routing
- Applications Data is split into *packets* and handled independently (routed through different paths)
- Supports both connectionless, less reliable **UDP** (User Datagram Protocol) and connection-oriented **TCP** (Transmission Control Protocol).



Host-to-host communications over TCP/IP



Network 2: ETH (Ethernet)
Network 3: FDDI (Fibre Distributed Data Interface)
Network 4:PPP (Point-to-Point Protocol)
Network 1: ETH (Ethernet)

URIs and URLs

```
scheme:[//[user:password@]host[:port]][/]path[?query][#fragment]
```

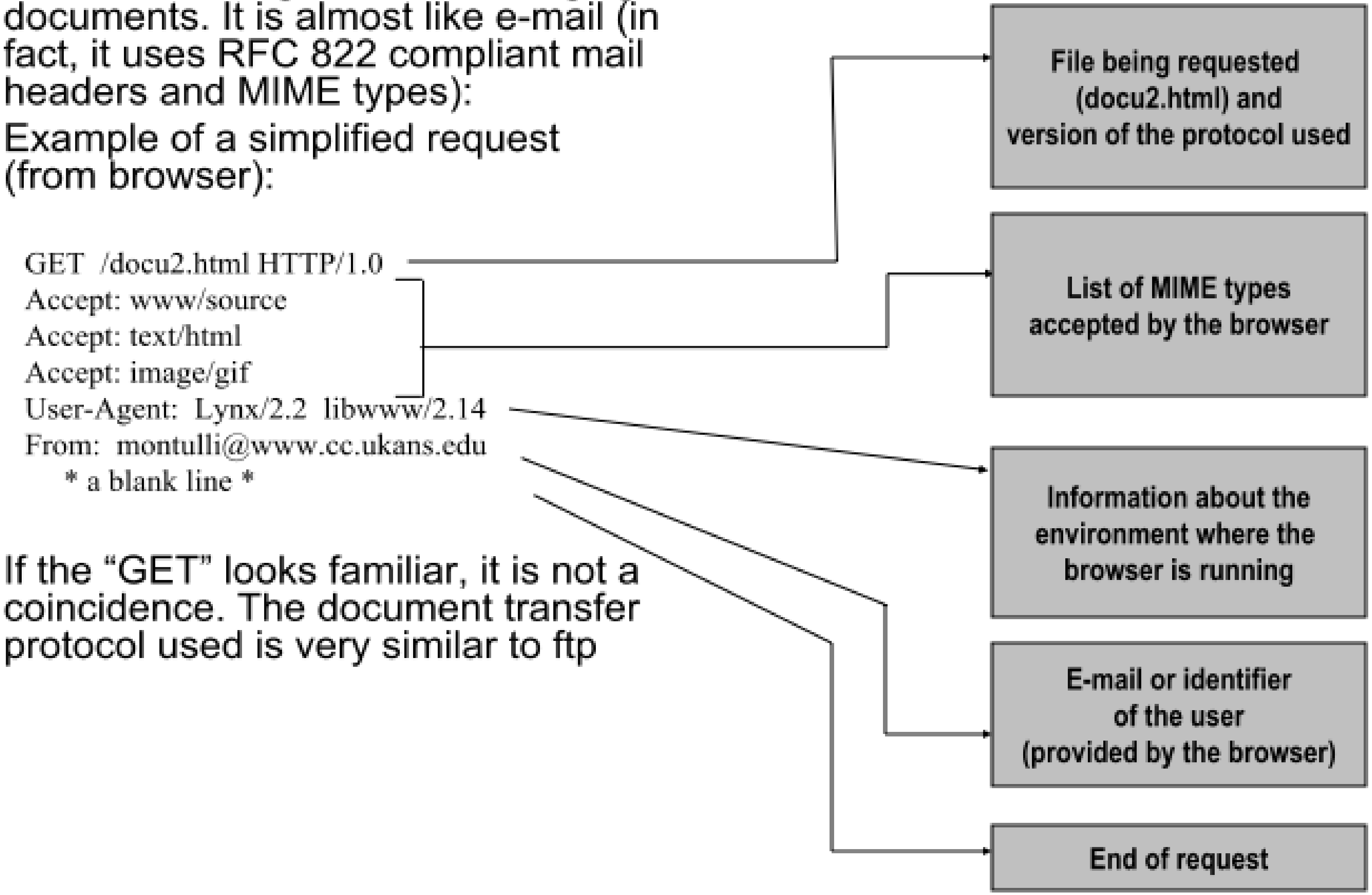
In information technology, a **Uniform Resource Identifier (URI)** is a string of characters used to identify a resource. Such identification enables interaction with representations of the resource over a network, typically the World Wide Web, using specific protocols. Schemes specifying a concrete syntax and associated protocols define each URI. The most common form of URI is the **Uniform Resource Locator (URL)**, frequently referred to informally as a web address. More rarely seen in usage is the **Uniform Resource Name (URN)**, which was designed to complement URLs by providing a mechanism for the identification of resources in particular namespaces.

A URL is a URI that, in addition to identifying a web resource, specifies the means of acting upon or obtaining the representation of it, i.e. specifying both its primary access mechanism and network location. For example, the URL `http://example.org/wiki/Main_Page` refers to a resource identified as `/wiki/Main_Page` whose representation, in the form of HTML and related code, is obtainable via Hypertext Transfer Protocol (`http`) from a network host whose domain name is `example.org`

Source: Wikipedia

HTTP as a communication protocol

- □ HTTP was designed for exchanging documents. It is almost like e-mail (in fact, it uses RFC 822 compliant mail headers and MIME types):
- □ Example of a simplified request (from browser):



```
GET  /docu2.html HTTP/1.0
Accept: www/source
Accept: text/html
Accept: image/gif
User-Agent:  Lynx/2.2  libwww/2.14
From:  montulli@www.cc.ukans.edu
    * a blank line *
```

File being requested (docu2.html) and version of the protocol used

List of MIME types accepted by the browser

Information about the environment where the browser is running

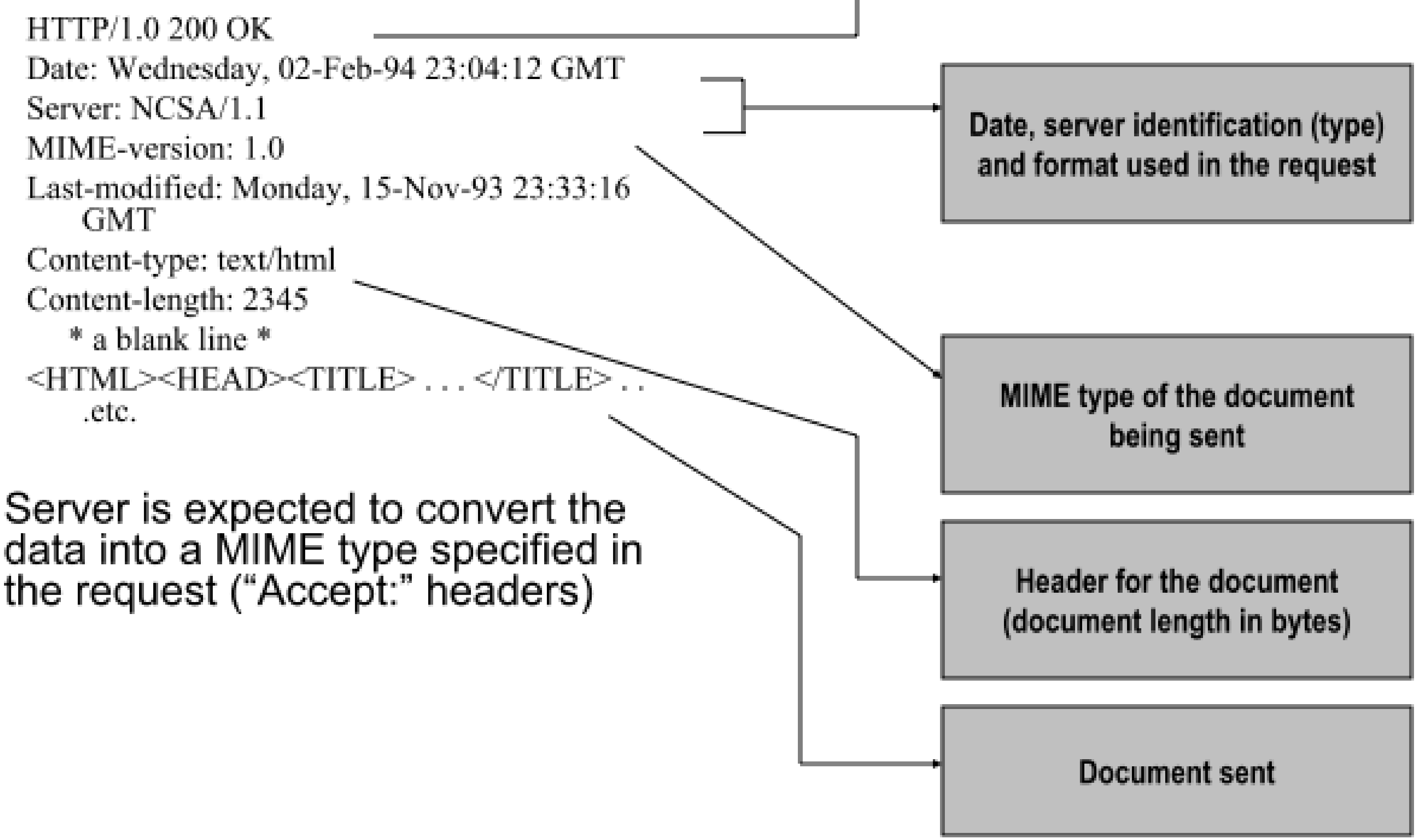
E-mail or identifier of the user (provided by the browser)

End of request

- □ If the "GET" looks familiar, it is not a coincidence. The document transfer protocol used is very similar to ftp

HTTP server side

- □ Example of a response from the server (to the request by the browser):



```
HTTP/1.0 200 OK
Date: Wednesday, 02-Feb-94 23:04:12 GMT
Server: NCSA/1.1
MIME-version: 1.0
Last-modified: Monday, 15-Nov-93 23:33:16
      GMT
Content-type: text/html
Content-length: 2345
    * a blank line *
<HTML><HEAD><TITLE> . . . </TITLE> . .
      .etc.
```

Protocol version, code indicating request status (200=ok)

Date, server identification (type) and format used in the request

MIME type of the document being sent

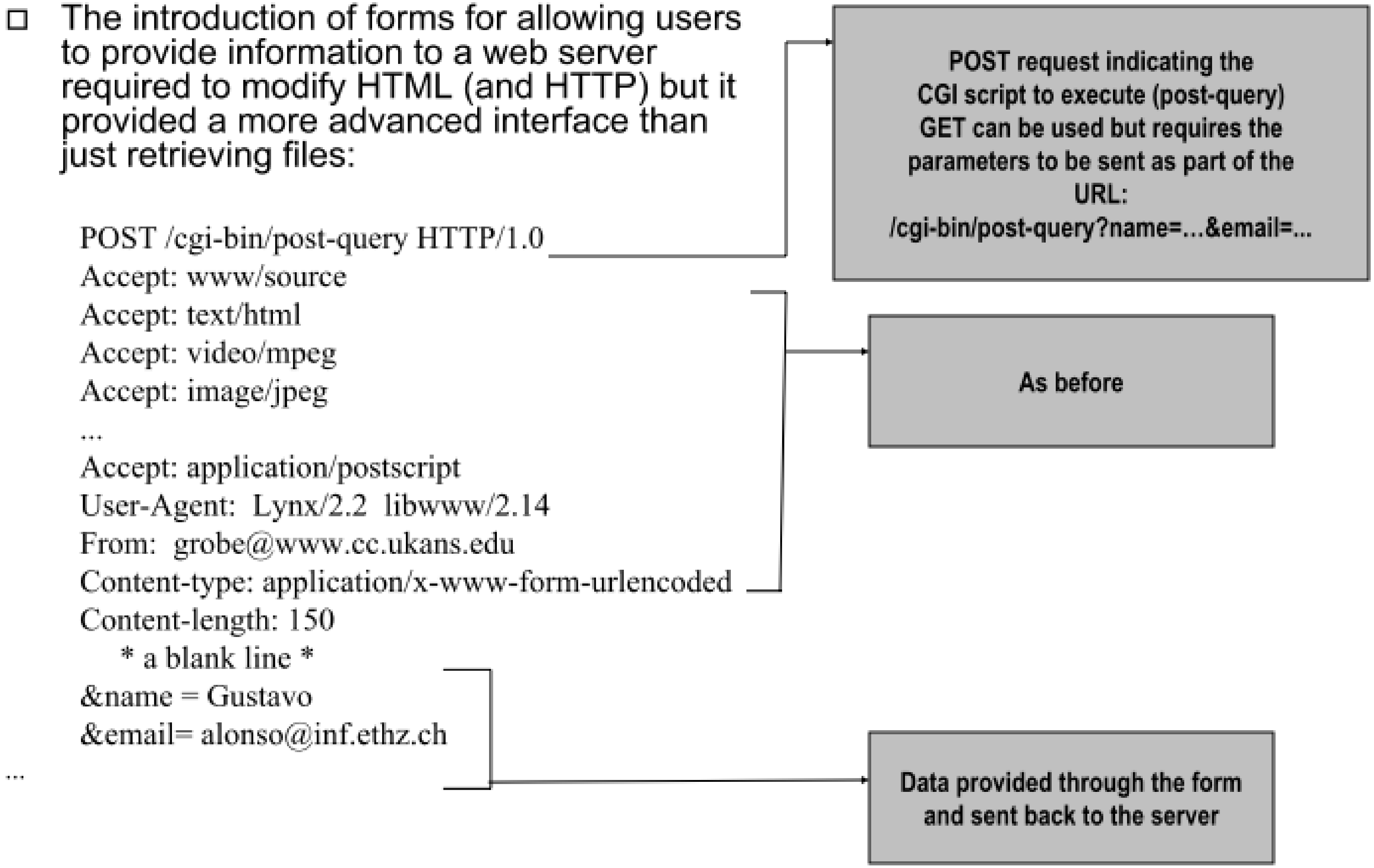
Header for the document (document length in bytes)

Document sent

- □ Server is expected to convert the data into a MIME type specified in the request (“Accept:” headers)

Parameter passing

- □ The introduction of forms for allowing users to provide information to a web server required to modify HTML (and HTTP) but it provided a more advanced interface than just retrieving files:



```
POST /cgi-bin/post-query HTTP/1.0
Accept: www/source
Accept: text/html
Accept: video/mpeg
Accept: image/jpeg
...
Accept: application/postscript
User-Agent:  Lynx/2.2  libwww/2.14
From:  grobe@www.cc.ukans.edu
Content-type: application/x-www-form-urlencoded
Content-length: 150
    * a blank line *
&name = Gustavo
&email= alonso@inf.ethz.ch
...
```

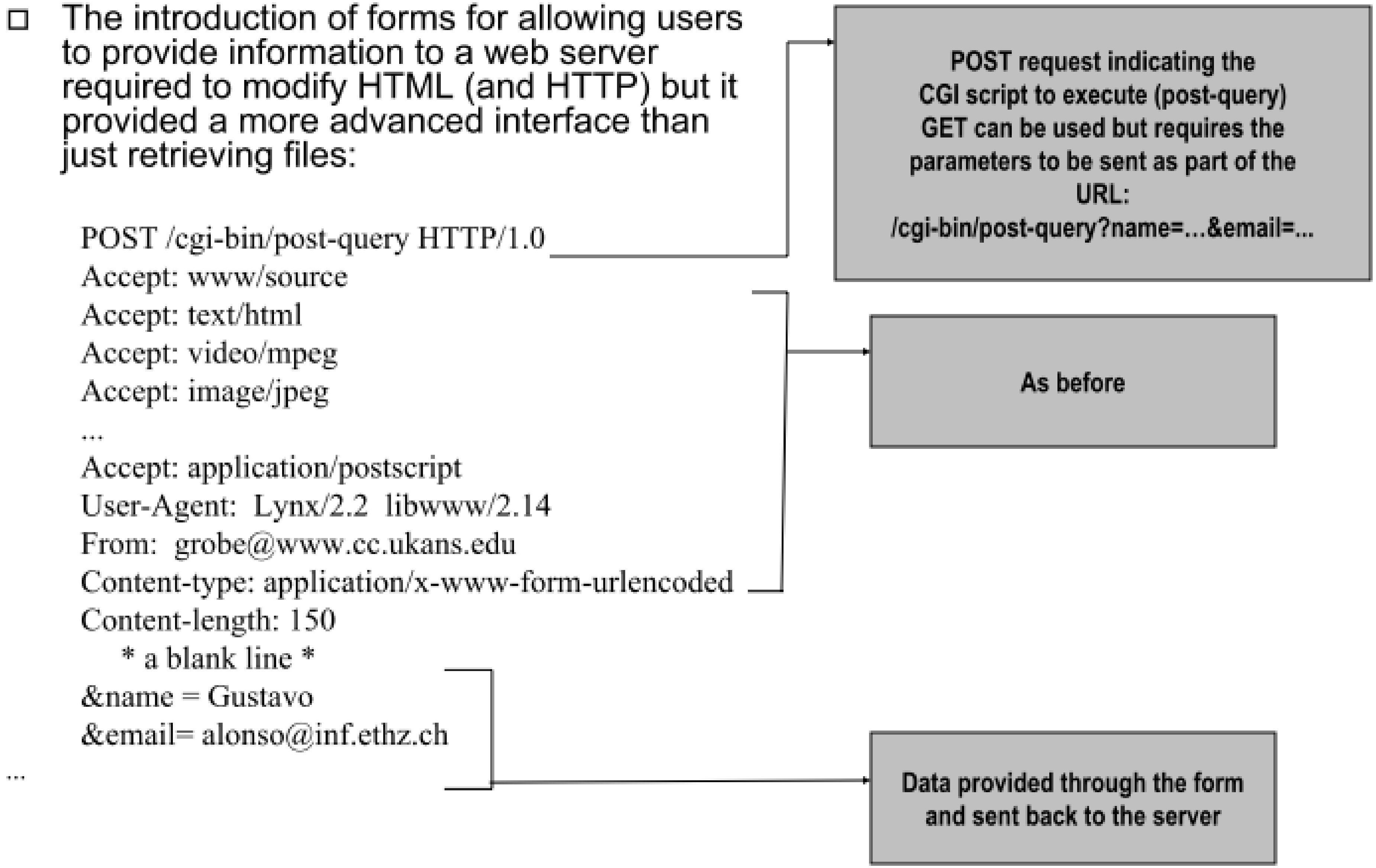
POST request indicating the CGI script to execute (post-query) GET can be used but requires the parameters to be sent as part of the URL: /cgi-bin/post-query?name=...&email=...

As before

Data provided through the form and sent back to the server

Parameter passing

- □ The introduction of forms for allowing users to provide information to a web server required to modify HTML (and HTTP) but it provided a more advanced interface than just retrieving files:



```
POST /cgi-bin/post-query HTTP/1.0
Accept: www/source
Accept: text/html
Accept: video/mpeg
Accept: image/jpeg
...
Accept: application/postscript
User-Agent:  Lynx/2.2  libwww/2.14
From:  grobe@www.cc.ukans.edu
Content-type: application/x-www-form-urlencoded
Content-length: 150
     * a blank line *
&name = Gustavo
&email= alonso@inf.ethz.ch
...
```

POST request indicating the CGI script to execute (post-query) GET can be used but requires the parameters to be sent as part of the URL: /cgi-bin/post-query?name=...&email=...

As before

Data provided through the form and sent back to the server

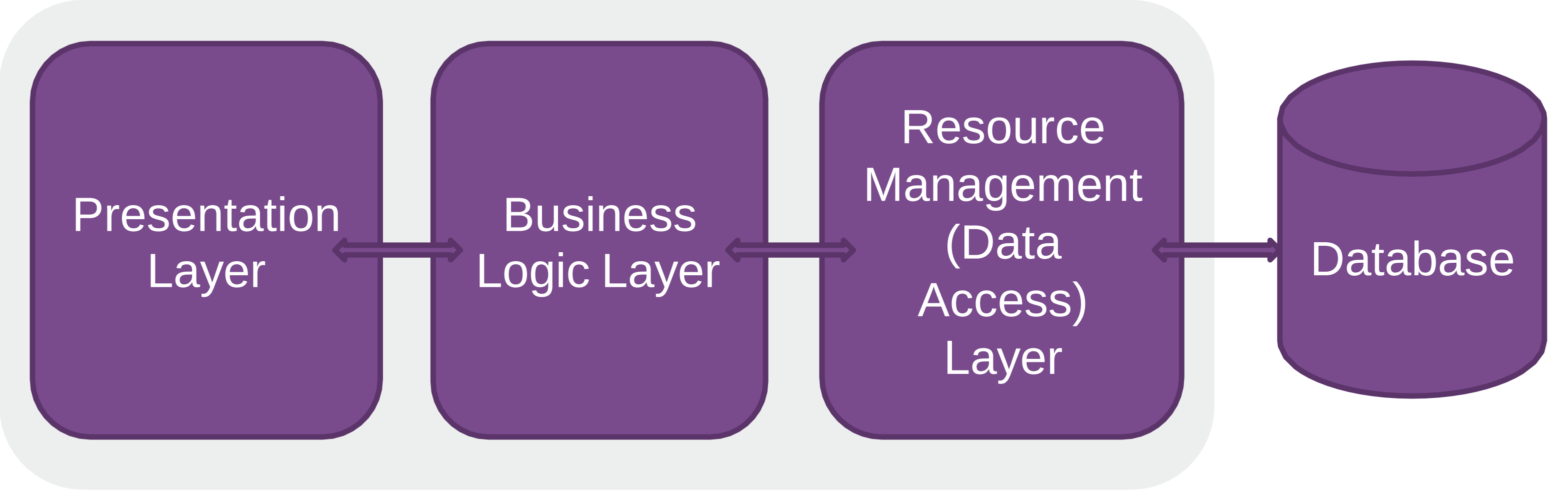
Distributed Systems Before the Web

- ❑ Information Systems consist of 3 layers (logical separation):
  - – **Presentation** (user interface) Layer
  - – **Application** Logic/Business Layer
  - – **Resource Management** (database) Layer

- ❑ Physical separation of these layers as Tiers
  - – Depending on where these components run

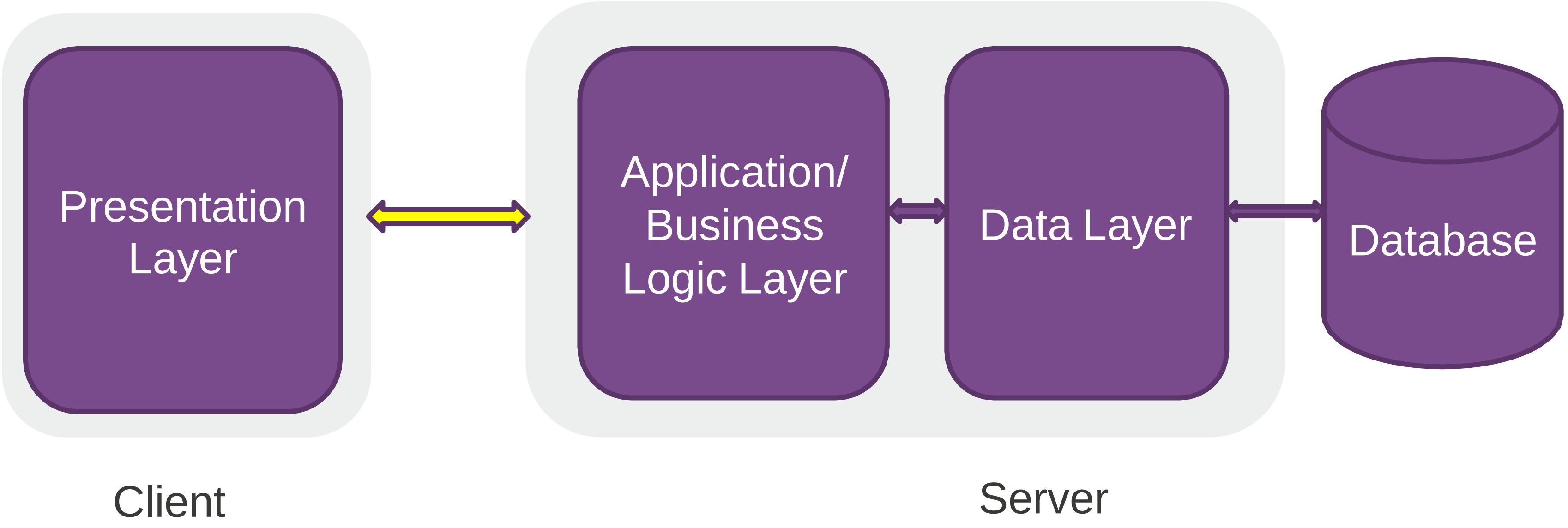
1-Tier

- ❑ 1 Tier: 3 layers running on one machine
  - – Dumb terminals and mainframes
  - – Layers tightly connected
  - – Difficulty with scalability and portability



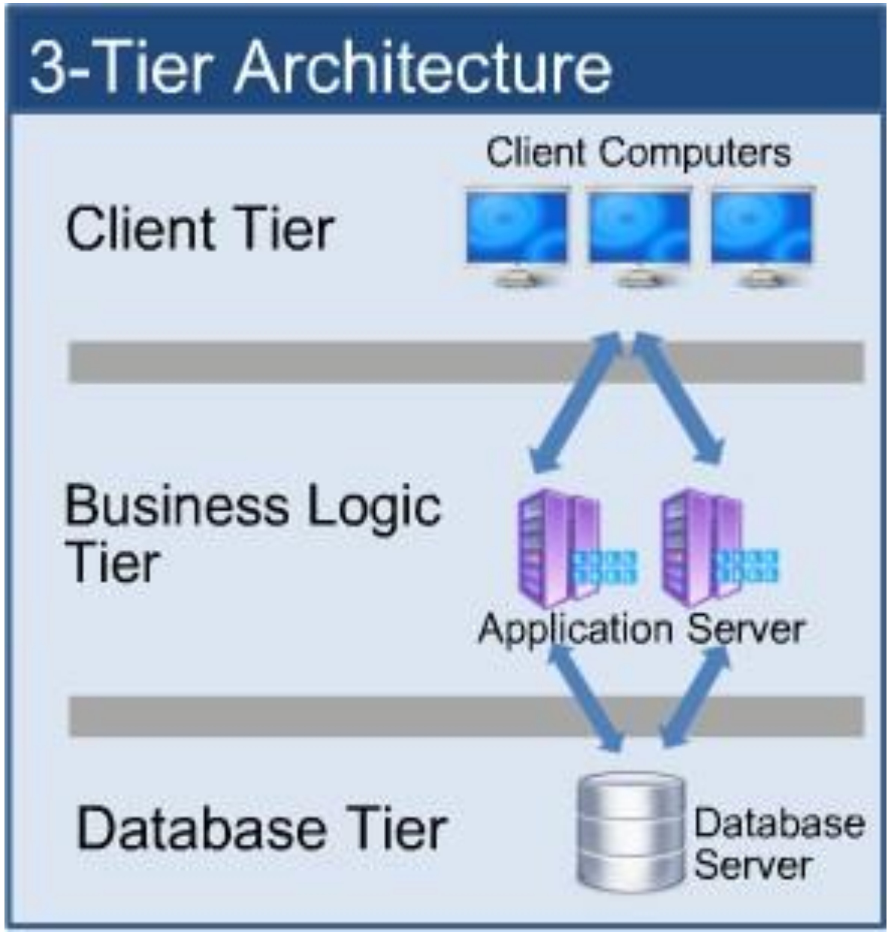
2-Tier

- ❑ The client/server architecture
  - – typically the presentation layer runs on the client machine and the data layer on the server side



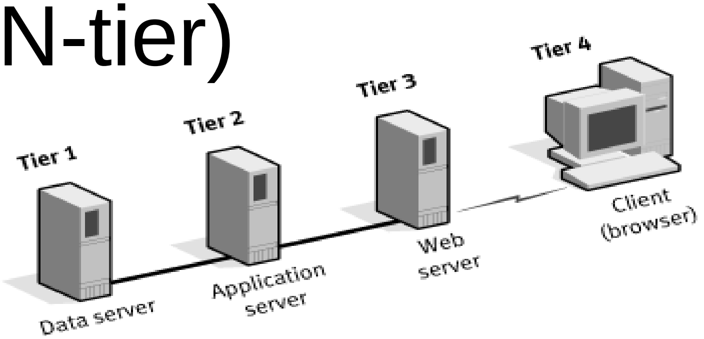
3-Tier

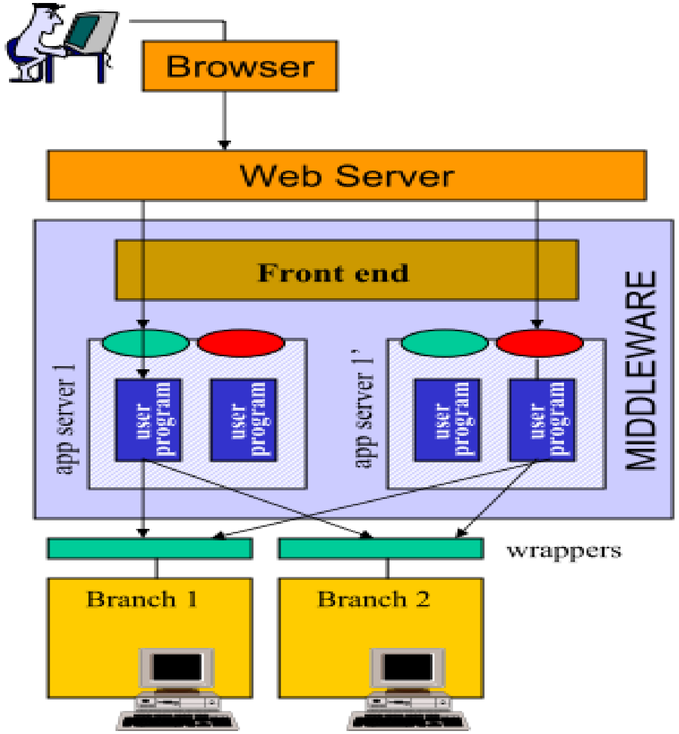
- ❑ Adding an extra tier between client and server called middleware, where Application Logic Layer resides
  - – Middleware also responsible for integration of underlying systems/ servers (other 2- tier or 3-tier systems)
- ❑ Each layer almost independent and running on a separate tier
  - – Three Layers sometimes called front-end (or GUI), middleware and back-end



The Web as software layer (N-tier)

- ❑ N-tier architectures: incorporating web servers
- ❑ Middleware platforms supporting access through the Web also known as “application servers”
- ❑ On receiving requests, Web Server can return the *static* content (e.g. html pages or images)
- ❑ To generate a *dynamic* responses, the web server sends or redirects the request to some other programs (e.g. Common Gateway Interface (CGI), JSP, PHP, ASP, Perl.)

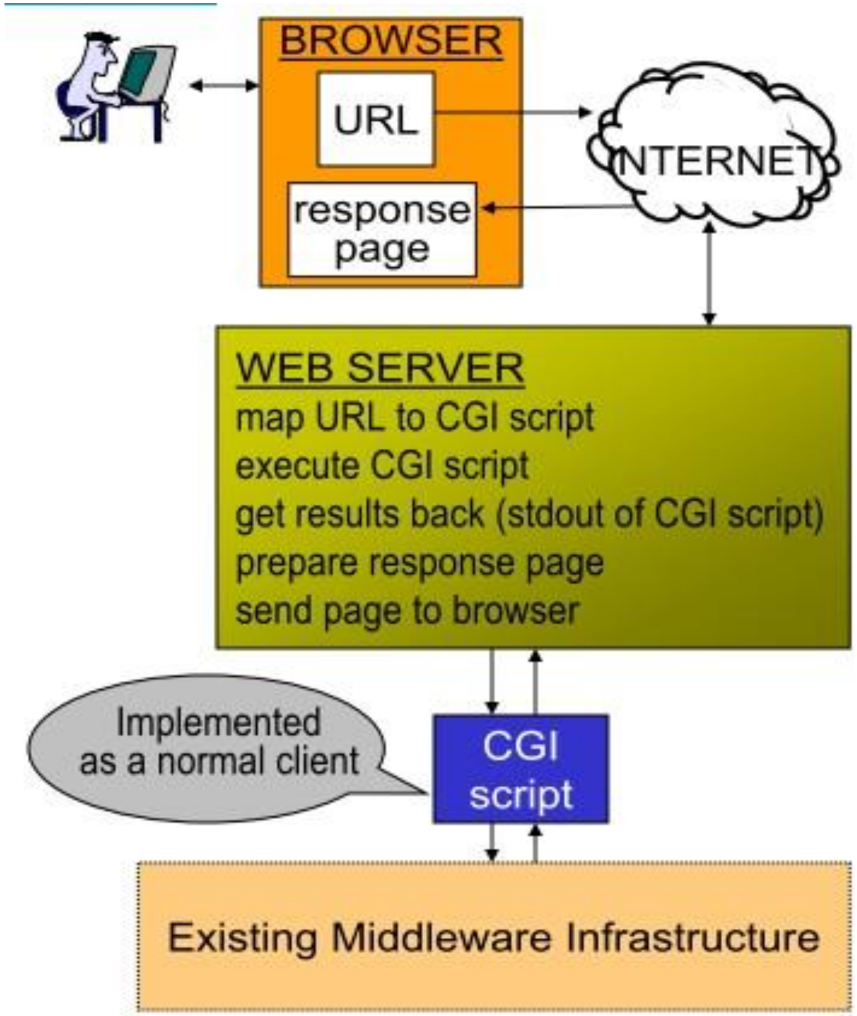




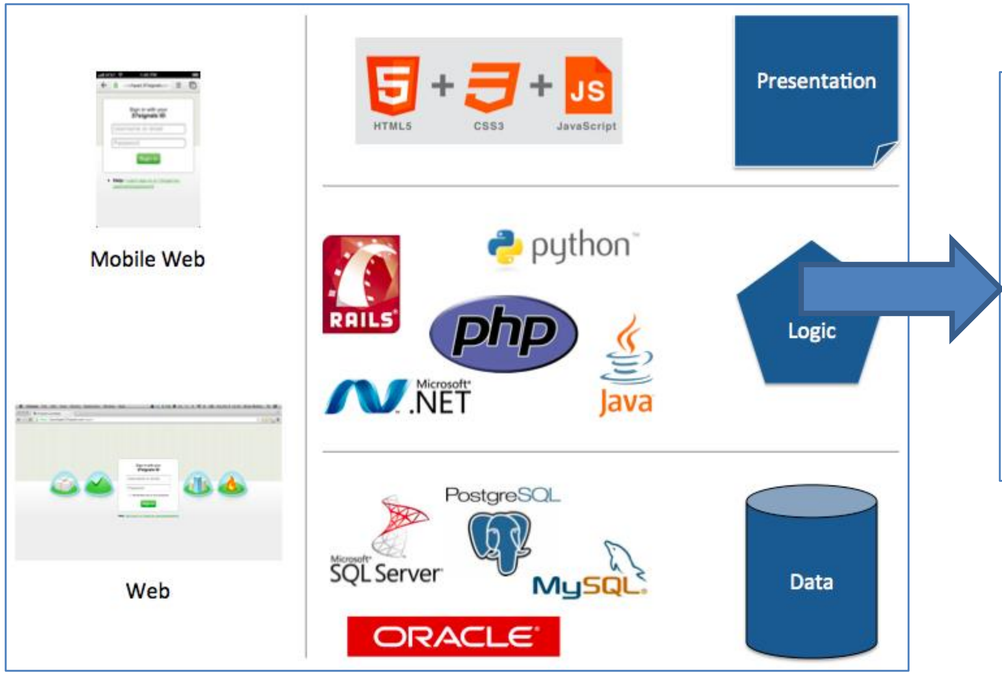
First Generation Web Technologies

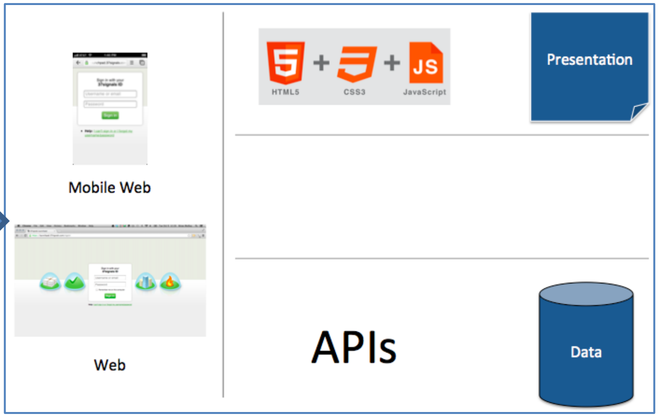
❑ The earliest implementations were very simple and built directly upon the existing client/server systems

- the CGI script (or program) acted as client in the traditional sense (for instance using RPC)
- the user clicked in a given URL and the server invoked the corresponding script
- the script executed, produced the results and passed them back to the server (usually as the address of a web page)
- the server retrieved the page and send it to the browser



New Technologies





Simplification of Development Methodology with Service Oriented Architectures (SOA)

- ❑ **Client-side technologies and scripting**

  Embedded within HTML (e.g. JavaScript)

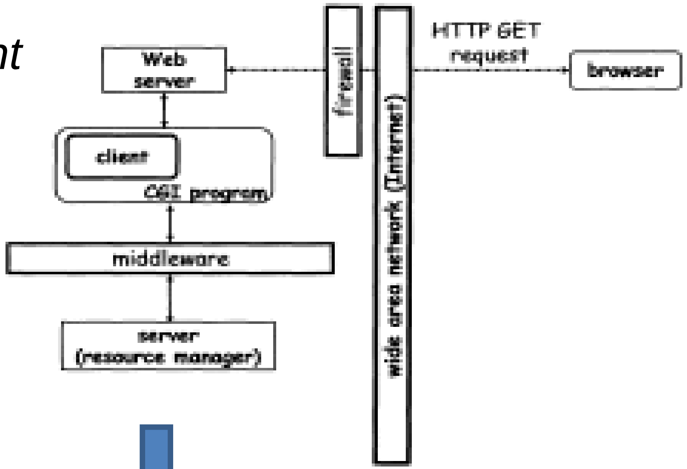
- ❑ **Server-side technologies and scripting**

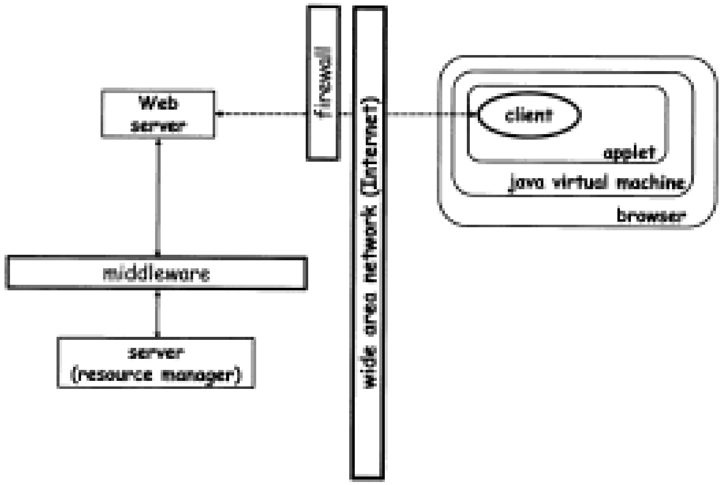
  JSP, PHP, ASP, Perl, Python, Ruby, JavaScript

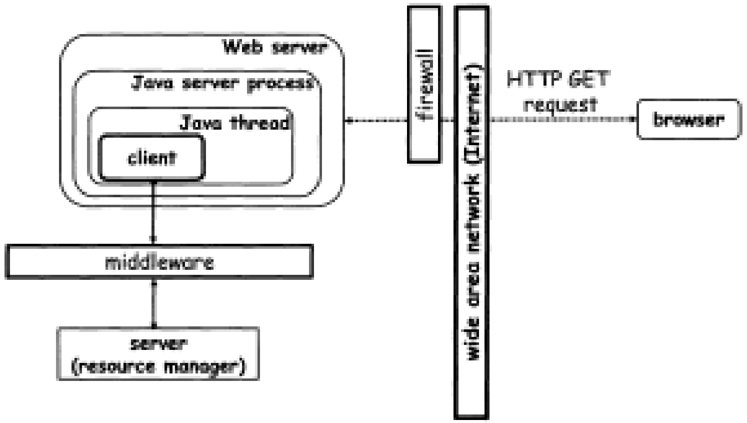
Java Web CS Development

- ❑ **Client-side**: Java Applets can implement *transient* clients via Web Browser. JVM increases security
- ❑ More permanent web-clients can be built using standardised HTTP protocol.

- ❑ **Server-side**: Java Server Process (JSP) coordinates servlets as worker threads. Invoked via HTTP
- ❑ Concurrent operation more efficient than CGI.

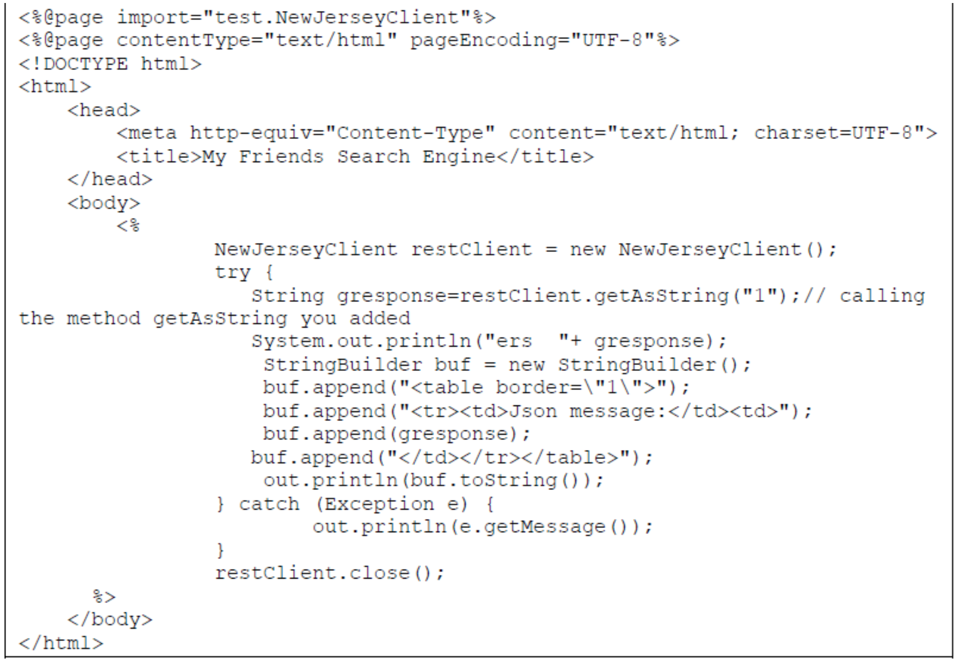






JavaServer Pages (JSP pages) and Servlets

- ❑ JSP is a server-side technology for creating dynamic web pages
- ❑ Servlets are Java programs to extend server capabilities
- ❑ **JSP is java in html** but **Servlet is html in java**
- ❑ Servlets and JSP pages can be used together to separate the presentation layer



```
<%@page import="test.NewJerseyClient"%>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>My Friends Search Engine</title>
    </head>
    <body>
        <%
                NewJerseyClient restClient = new NewJerseyClient();
                try {
                   String gresponse=restClient.getAsString("1");// calling
the method getAsString you added
                   System.out.println("ers  "+ gresponse);
                    StringBuilder buf = new StringBuilder();
                    buf.append("<table border=\"1\">");
                    buf.append("<tr><td>Json message:</td><td>");
                    buf.append(gresponse);
                   buf.append("</td></tr></table>");
                    out.println(buf.toString());
                } catch (Exception e) {
                        out.println(e.getMessage());
                }
                restClient.close();
      %>
    </body>
</html>
```

```
PrintWriter out = response.getWriter())
…
java.lang.String bodyText = TextArea1;
com.cdyne.ws.DocumentSummary doc = checkTextBodyV2(bodyText);
String allcontent = doc.getBody();
int no_of_mistakes = doc.getMisspelledWordCount();
List allwrongwords = doc.getMisspelledWord();
out.println("<html>");
out.println("<head>");
out.println("<title>Spell Checker Report</title>");
 out.println("</head>");
 out.println("<body>");
//Display the report's name as a header within the body of the report:
out.println("<h2><font color='red'>Spell Checker Report</font></h2>");
//Display all the content  between quotation marks:
out.println("<hr><b>Your text:</b> \"" + allcontent + "\"" + "<p>");


…
```

JavaScript

- ❑ A lightweight, object-oriented language to create applications to run over the internet
- ❑ Traditionally used just a client side language but for both client and server side programming
- ❑ Client side applications using JavaScript run in a browser, and server side applications run on a server to extend its capabilities

Client-side JavaScript example ([Turn on/off the light])

More about functions rather than classes

The script can be embedded within HTML or stored in a file (.js)

AJAX (Asynchronous JavaScript and XML)

Partial updating of a web page without reloading the entire page (Google Suggest)

jQuery

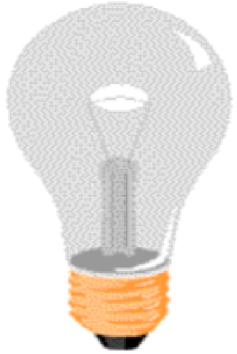
a set of JavaScript libraries that greatly simplifies JavaScript programming

JQuery example

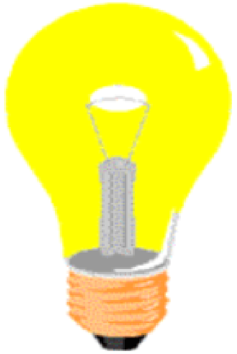
Javascript example

```
<!DOCTYPE html>
<html>
<title>Tutorial to turn on/of light</title>
<body>
<script>
function action() {
  var image = document.getElementById('bulb');
  if (image.src.match("bulbon")) {
  image.src = "bulboff_image.gif";
  } else {
  image.src = "bulbon_image.gif";
  }
}
</script>

<img id="bulb" onclick="action()“ src="bulboff_image.gif" width="150" height="230">
<p>Click the bulb to turn on or off the light</p>
</body>
</html>
```



Click the bulb to turn on or off the light

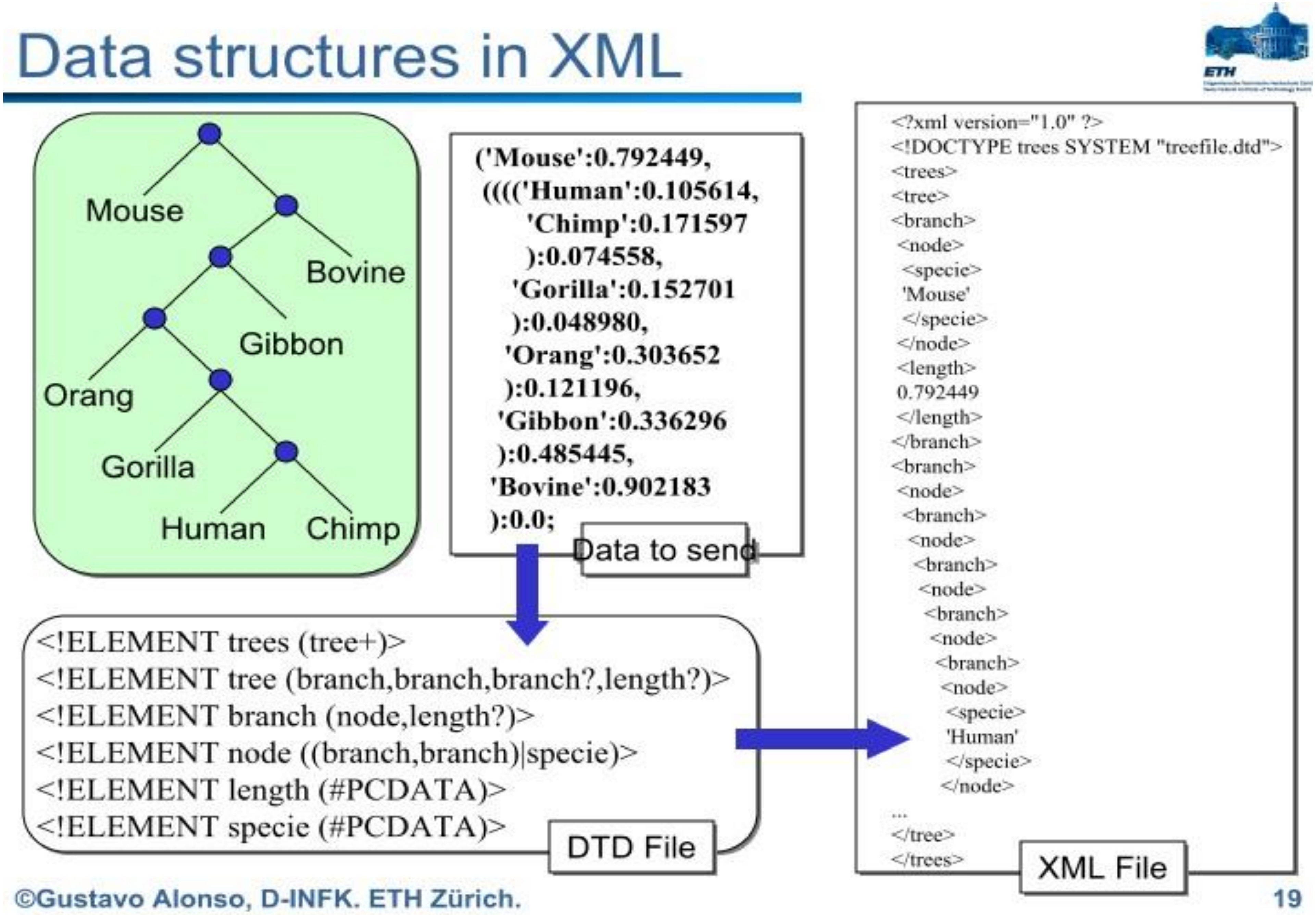


Click the bulb to turn on or off the light

XML

- ❑ XML tags not pre-defined like HTML
- ❑ The goal of XML is to provide a standardized way to specify data structures for data exchange and storage
- ❑ XML Schemas has support for data types
- ❑ Unlike HTML, XML is not intended for browsers
- ❑ XML can be automatically processed by other programs and machines
- ❑ XML can be used as the intermediate language for marshalling/serializing arguments when invoking services across the Internet

Data structures in XML



Data to send

```
('Mouse':0.792449,
 (((('Human':0.105614,
     'Chimp':0.171597
     ):0.074558,
    'Gorilla':0.152701
    ):0.048980,
   'Orang':0.303652
   ):0.121196,
  'Gibbon':0.336296
  ):0.485445,
 'Bovine':0.902183
 ):0.0;
```

DTD File

```
<!ELEMENT trees (tree+)>
<!ELEMENT tree (branch,branch,branch?,length?)>
<!ELEMENT branch (node,length?)>
<!ELEMENT node ((branch,branch)|specie)>
<!ELEMENT length (#PCDATA)>
<!ELEMENT specie (#PCDATA)>
```

XML File

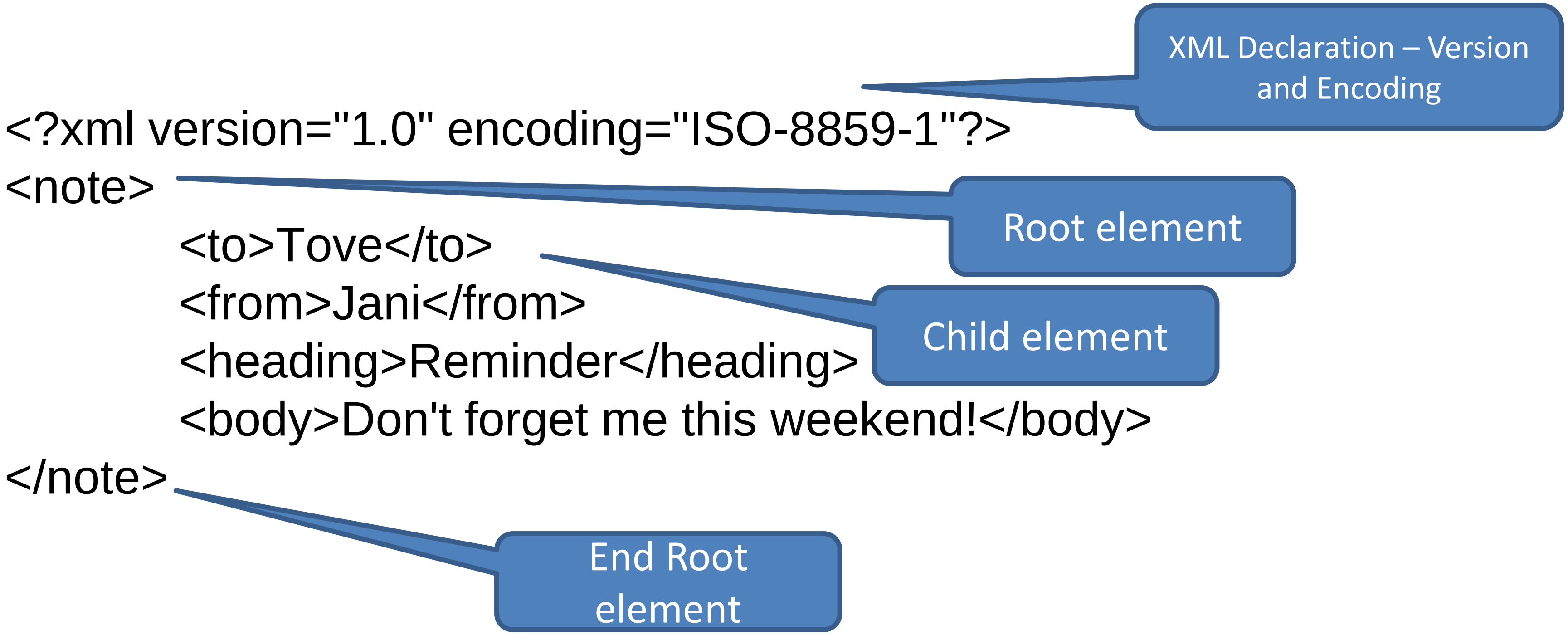
```
<?xml version="1.0" ?>
<!DOCTYPE trees SYSTEM "treefile.dtd">
<trees>
<tree>
<branch>
 <node>
  <specie>
  'Mouse'
  </specie>
 </node>
 <length>
 0.792449
 </length>
</branch>
<branch>
 <node>
  <branch>
   <node>
    <branch>
     <node>
      <branch>
       <node>
        <branch>
         <node>
          <specie>
          'Human'
          </specie>
         </node>
...
</tree>
</trees>
```

©Gustavo Alonso, D-INFK. ETH Zürich.

19

XML Document Structures

- ❑ XML documents form a tree structure that starts at "the root" and branches to "the leaves’’



```
<?xml version="1.0" encoding="ISO-8859-1"?>
<note>
        <to>Tove</to>
        <from>Jani</from>
        <heading>Reminder</heading>
        <body>Don't forget me this weekend!</body>
</note>
```

XML Declaration – Version and Encoding

Root element

Child element

End Root element

XML Schema and XML Documents

```
<xs:element name="shipto">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="name" type="xs:string"/>
   <xs:element name="address" type="xs:string"/>
   <xs:element name="city" type="xs:string"/>
   <xs:element name="country" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

```
<shipto>
   <name>OlaNordmann</name>
   <address>Langst 23</address>
   <city>4000 Stavanger</city>
   <country>Norway</country>
 </shipto>
```

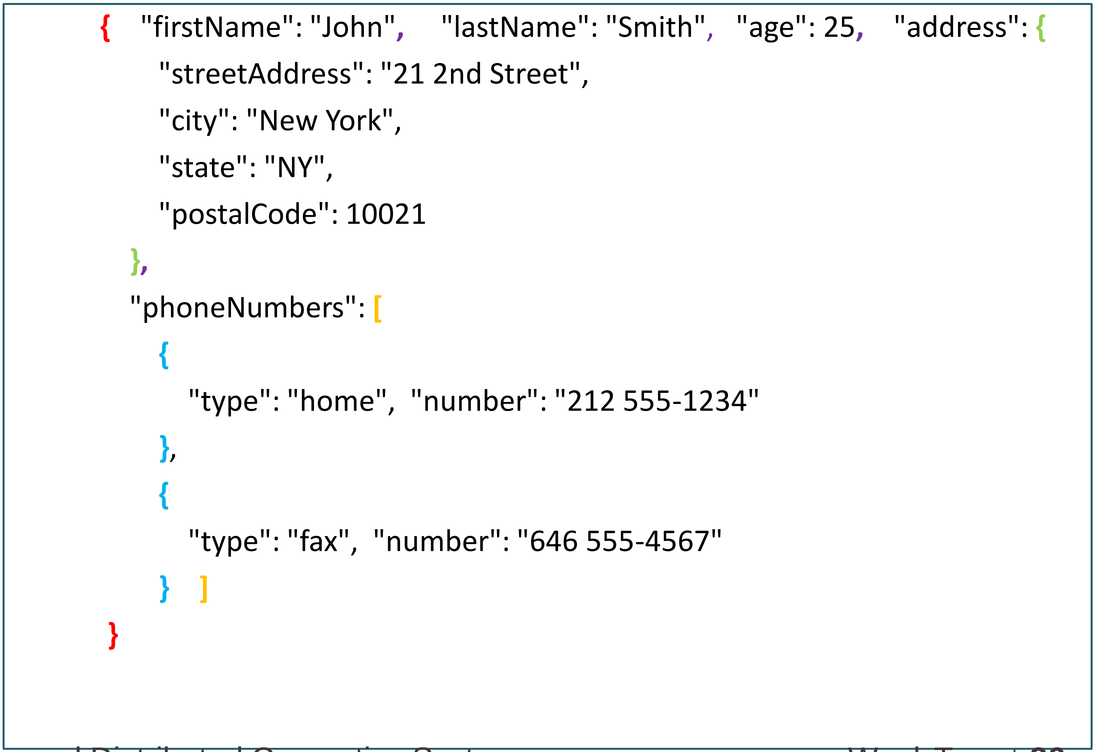
XML Namespaces – xmlns Attribute

- ❑ When a namespace is defined for an element, all child elements with the same prefix are associated with the same namespace
- ❑ Namespaces can be declared in the elements where they are used or in the XML root element

```
<root>
      xmlns:h="http://www.w3.org/TR/html4/"
      xmlns:f="http://www.w3schools.com/furniture">
<h:table>
      <h:tr>
      <h:td>Apples</h:td>
      <h:td>Bananas</h:td>
      </h:tr>
</h:table>
<f:table>
      <f:name>African Coffee Table</f:name>
      <f:width>80</f:width>
      <f:length>120</f:length>
</f:table>
</root>
```

JSON

- ❑ JSON stands for JavaScript Object Notation
  - uses JavaScript syntax for describing data objects
- ❑ JSON is lightweight text-data interchange format
- ❑ JSON is "self-describing" and easy to understand
- ❑ Data is in name/value pairs, followed by a colon
- ❑ Data is separated by commas
- ❑ Curly braces hold objects
- ❑ Square brackets hold arrays



```
{   "firstName": "John",    "lastName": "Smith",   "age": 25,    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        {
            "type": "home",  "number": "212 555-1234"
        },
        {
            "type": "fax",  "number": "646 555-4567"
        }   ]
 }
```

Web Services

- ❑ “A Web service is a software system/application designed to support interoperable machine-to-machine interaction over a network” W3C
- ❑ Hosting services on a remote machine
- ❑ A standardized way of integrating web-based applications
- ❑ The request and the response encoded in a format easy for a program to decode
  - The most common encodings are XML (SOAP or POX) and JSON
- ❑ SOAP and RESTful web services (RESTful Web APIs)

A Web Service..

- ❑ has an interface describing a collection of operations
- ❑ enables access to business logic, data and processes or other services
- ❑ can be accessed by humans, other applications or other web services
- ❑ all communications in XML so not limited to any operating system or programming language (SOAP)
- ❑ easy and cheap to develop with so many supporting tools
- ❑ Motivations: Enterprise Application Integration (**EAI**), **Supply Chain** management and Business-to-business **B2B** Integration.

XML (Extensible Markup Language)
SOAP (Simple Object Access Protocol)
WSDL (Web Services Description Language

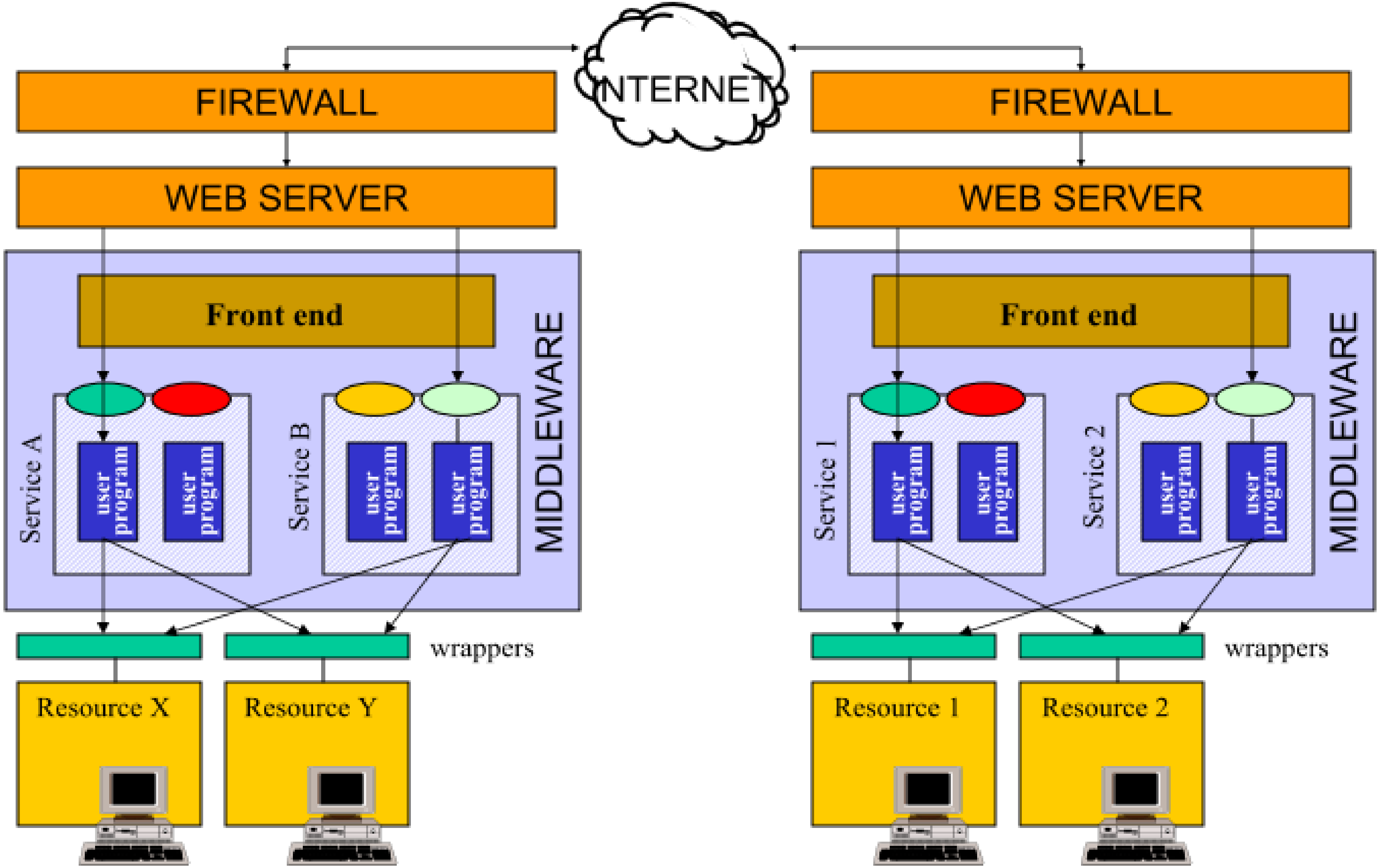
Benefits of Web Services

❑ One important difference with conventional middleware is related to the standardization efforts at the W3C that should guarantee:

- Platform independence (Hardware, Operating System)
- Reuse of existing networking infrastructure (HTTP has become ubiquitous)
- Programming language neutrality (.NET talks with Java, and vice versa)
- Portability across Middleware tools of different Vendors
- Web services are “loosely coupled” components that foster software reuse
- WS technologies should be composable so that they can be adopted incrementally

©Gustavo Alonso, D-INFK. ETH Zürich

Business to Business (B2B)



©Gustavo Alonso, D-INFK. ETH Zürich.

Challenges of B2B

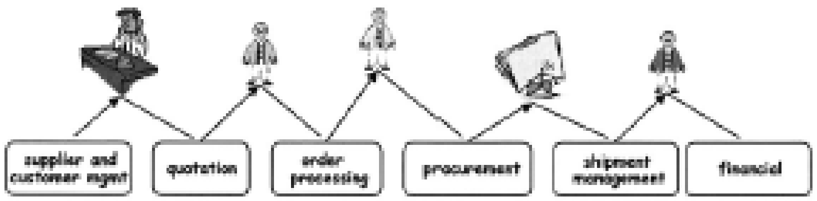
- ❑ The basic idea behind B2B is simple and follows the client/ server model.
- ❑ A service provided by one company can be directly invoked by a client running in another company.
- ❑ That way, the interactions between the companies are **automated** and their IT systems can directly interact with each other, thereby speeding up all transactions between both companies.

The problem is how to implement such a system?

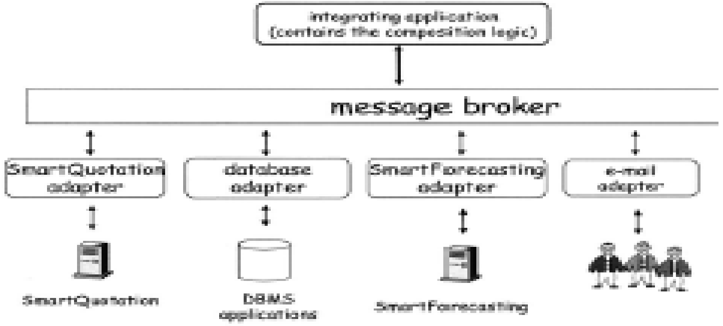
- the **client no longer near the server**
- joint development of client and server makes no sense
- the server and client are likely to be hidden **behind firewalls**
- the interaction takes place among **different systems**, it is not possible to homogenize supporting platforms
- the **Internet is cheap but open** to everybody (**security** problems)
- Existing systems/protocols are not really designed for such type of interactions

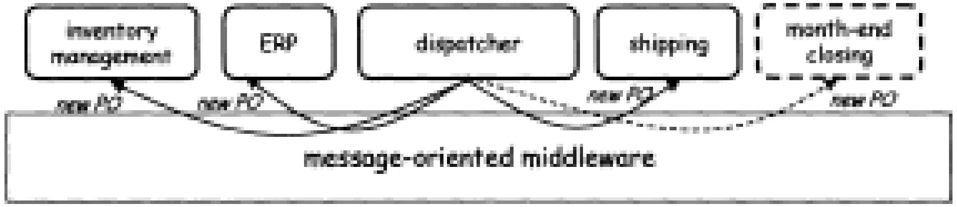
©Gustavo Alonso, D-INFK. ETH Zürich

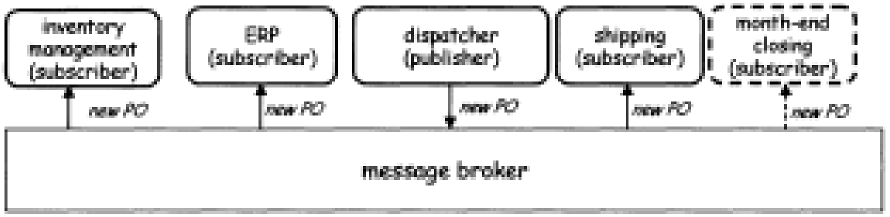
Challenges of EAI and (global) Supply Chains



- ❑ Manual implementation of supply chain. Human users extract, reformat and relay data
- ❑ Message brokers and Message Oriented Middleware based on RPC attempt to increase interoperability







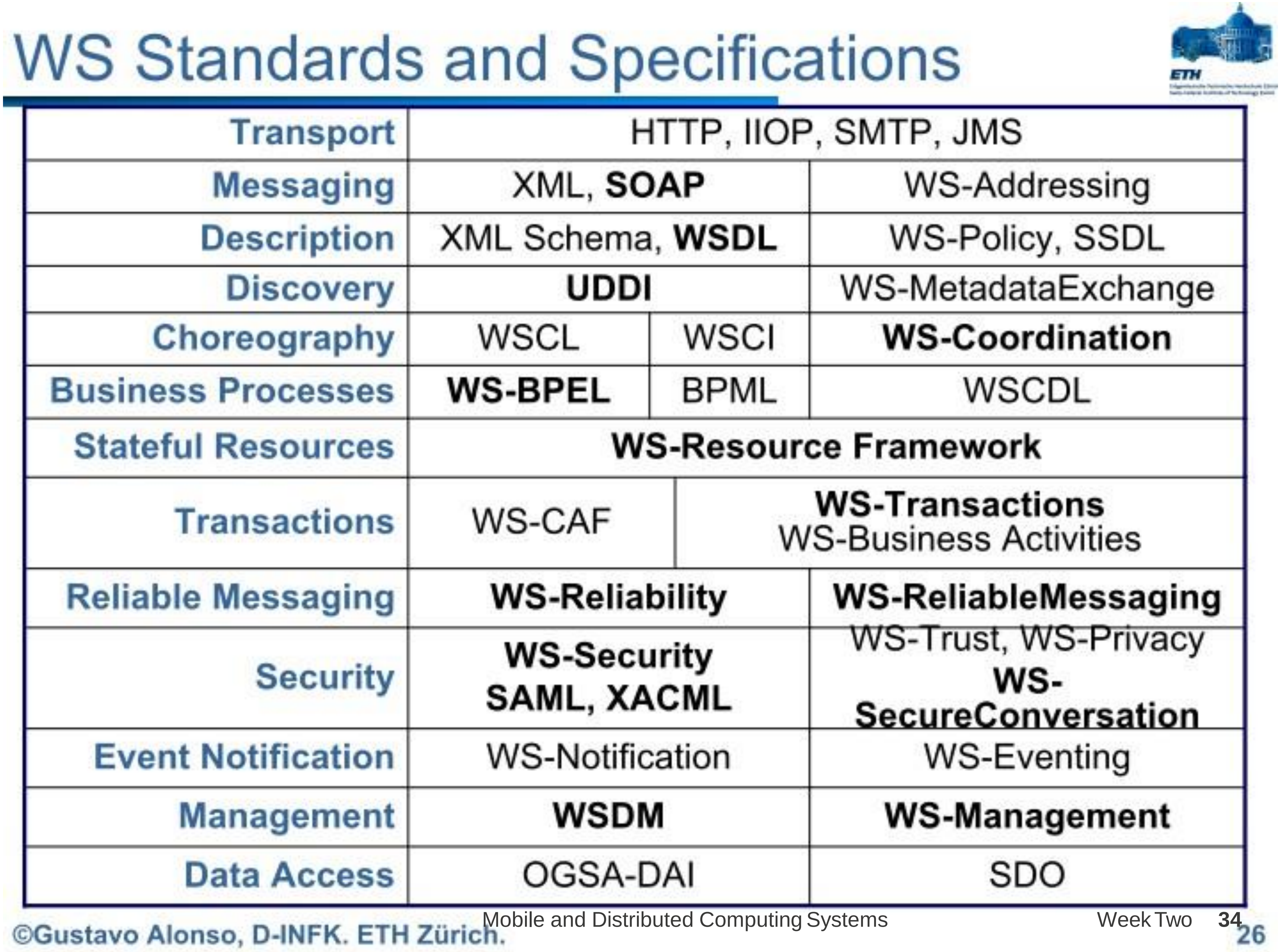
- ❑ Publish-subscribe models add level of indirection and increase flexibility
- ❑ Web-based SOA makes it simple and economical to use web standards to rapidly integrate heterogeneous and evolving Enterprise Applications

Open Standards

❑ **Web Services are built on open standards:**

- Simple Object Access Protocol (SOAP)
- Web Services Description Language (WSDL)
- Universal Description, Discovery and Integration (UDDI)*
- Hypertext Transfer Protocol (HTTP)
  - ○ Supported as a transport protocol – SOAP/HTTP for transporting messages across network applications
- REpresentational State Transfer (REST)
- We will look at SOAP today

* UDDI was intended as a core web service standard however original goals too ambitions and unrealistic



WS Standards and Specifications

Transport	HTTP, IIOP, SMTP, JMS		
Messaging	XML, **SOAP**		WS-Addressing
Description	XML Schema, **WSDL**		WS-Policy, SSDL
Discovery	**UDDI**		WS-MetadataExchange
Choreography	WSCL	WSCI	**WS-Coordination**
Business Processes	**WS-BPEL**	BPML	WSCDL
Stateful Resources	**WS-Resource Framework**		
Transactions	WS-CAF	**WS-Transactions** WS-Business Activities	
Reliable Messaging	**WS-Reliability**		**WS-ReliableMessaging**
Security	**WS-Security** **SAML, XACML**		WS-Trust, WS-Privacy **WS-SecureConversation**
Event Notification	WS-Notification		WS-Eventing
Management	**WSDM**		**WS-Management**
Data Access	OGSA-DAI		SDO

©Gustavo Alonso, D-INFK. ETH Zürich. 26

HTTP

- ❑ HTTP communicates over TCP/IP.
- ❑ An HTTP client connects to an HTTP server using TCP.
- ❑ After establishing a connection, the client can send an HTTP request message to the server:

```
POST /item HTTP/1.1
Host: 189.123.345.239
Content-Type: text/plain
Content-Length: 200
```

- ❑ The server then processes the request and sends an HTTP response back to the client.
- ❑ The response contains a status code that indicates the status of the request.

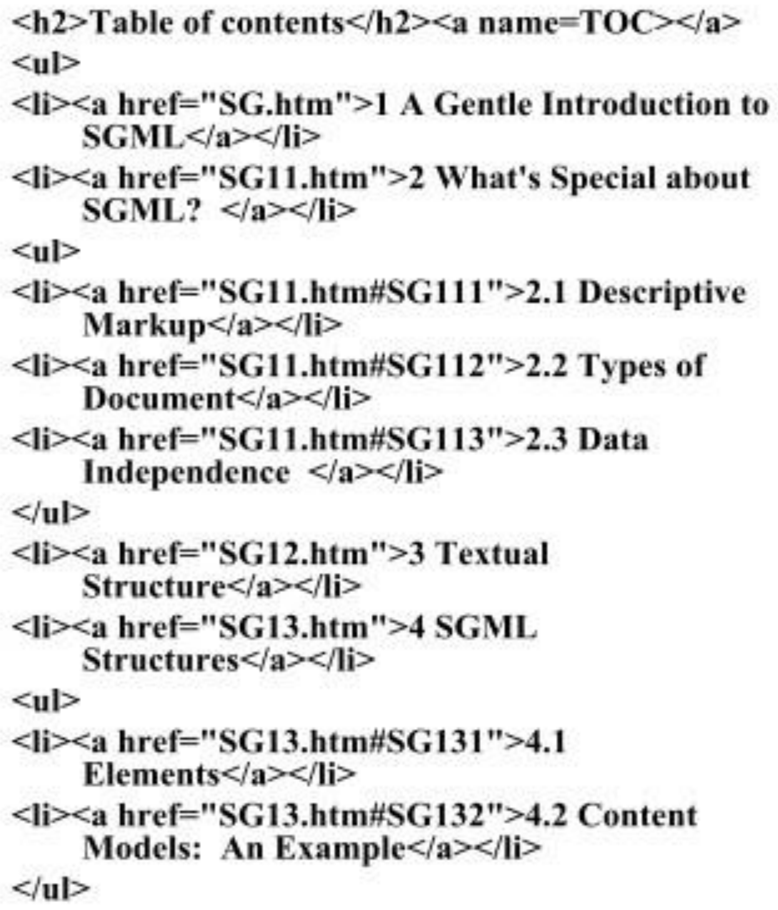
```
200 OK
Content-Type: text/plain
Content-Length: 200
```

- ❑ In the example above, the server returned a status code of 200.
- ❑ This is the standard success code for HTTP. If the server could not decode the request, it could return:

```
400 Bad Request Content-Length: 0
```

Contents and Presentation

- **HTML** is **a tag language** designed to describe how a document should be displayed (**the visual format** of the document).
- **Tag languages** provide a **standardized** grammar defining the meaning of tags and their use
- Tag languages use **SGML**, an international text processing standard from the 80’s, to define tag sets and grammars
- HTML is based on SGML, that is, the tags and the grammar used in HTML documents have been defined using SGML



```
<h2>Table of contents</h2><a name=TOC></a>
<ul>
<li><a href="SG.htm">1 A Gentle Introduction to
      SGML</a></li>
<li><a href="SG11.htm">2 What's Special about
      SGML?  </a></li>
<ul>
<li><a href="SG11.htm#SG111">2.1 Descriptive
      Markup</a></li>
<li><a href="SG11.htm#SG112">2.2 Types of
      Document</a></li>
<li><a href="SG11.htm#SG113">2.3 Data
      Independence  </a></li>
</ul>
<li><a href="SG12.htm">3 Textual
      Structure</a></li>
<li><a href="SG13.htm">4 SGML
      Structures</a></li>
<ul>
<li><a href="SG13.htm#SG131">4.1
      Elements</a></li>
<li><a href="SG13.htm#SG132">4.2 Content
      Models:  An Example</a></li>
</ul>
```

Web Services and SOA

- ❑ **SOA** = Services Oriented Architecture
  - **Services** = another name for large scale components wrapped behind a standard interface (Web services although not only)
  - **Architecture** = SOA is intended as **A WAY** to **build complex systems and applications**
- ❑ The part that it is not in the name
  - **Loosely-coupled** = the services are independent of each other, heterogeneous, distributed
  - **Message based** = interaction is through message exchanges rather than through direct calls

The Need for SOA

❑ Most companies today have large, heterogeneous IT infrastructure that:

- keeps changing
- needs to evolve to adopt new technology
- needs to be connected of that of commercial partners

❑ In the field of Enterprise Application Integration using systems like CORBA or DCOM. However, solutions until now suffered from:

- Tightly integrated systems
- Vendor lock-in
- Technology lock-in (e.g., CORBA)
- Lack of flexibility and limitations when new technology arises (e.g., Internet)
- Lack of standardization

©Gustavo Alonso, D-INFK. ETH Zürich

The Novelty behind SOA

❑ SOA is an attempt to build on standards (web services) to reduce the cost of integration

❑ It introduces very interesting possibilities:

- Development by composition
- Supports reuse
- Frees developers from “lock-in” effects of various kinds (such as from SaaS, PaaS and to a lesser extent IaaS services)
- Use of standard interfaces (Web services)
- Existing supporting infrastructure for easy development (automatic)

SOA vs. Web Services

- ❑ Web services are about
  - – Interoperability
  - – Standardization
  - – Integration across heterogeneous, distributed systems
- ❑ Service Oriented Architectures are about:
  - – Large scale software design
  - – Software Engineering
  - – Architecture of distributed systems
- ❑ SOA is possible but more difficult without Web services
- ❑ SOA introduces some radical changes to software:
  - – Language independence (what matters is the interface)
  - – Message based exchanges (no RPC)
  - – Composition and orchestration

SOAP Web Service Definitions

- ❑ Web services use the XML, SOAP, WSDL and UDDI open standards over an Internet protocol backbone
- ❑ XML provides an open standard for data exchange, HTTP an open transport protocol