



Information and  
Communication Systems  
Research Group



**ETH**

Eidgenössische Technische Hochschule Zürich  
Swiss Federal Institute of Technology Zurich

1

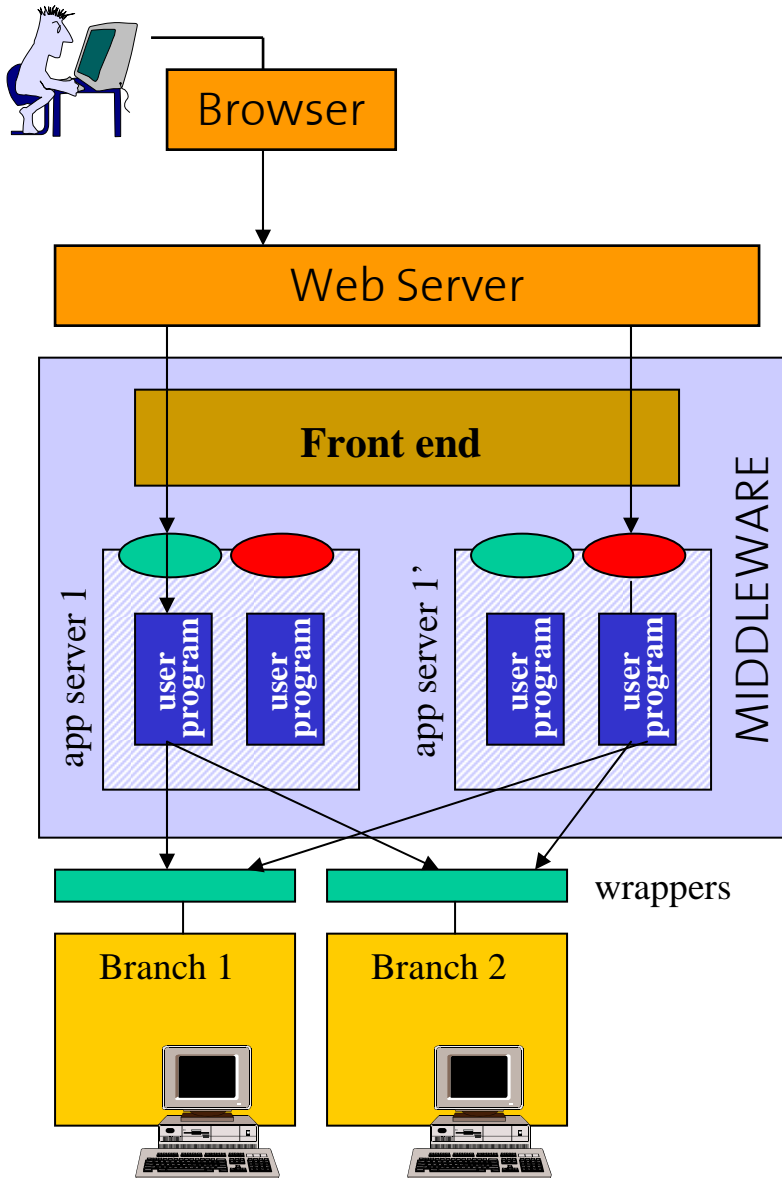
# Web Services and Service Oriented Architectures (SOA)

Gustavo Alonso  
Computer Science Department  
Swiss Federal Institute of Technology (ETHZ)  
alonso@inf.ethz.ch  
<http://www.iks.inf.ethz.ch/>

# Basic info

- Web pages:
  - [http://www.iks.inf.ethz.ch/education/sso8/ws\\_soa](http://www.iks.inf.ethz.ch/education/sso8/ws_soa)
  - This is where all the lecture materials and additional pointers can be found
  
- Exercises:
  - Exercises are mandatory and a requirement to do the exam
  - Please register as soon as possible in e-Doz
  
- Course (tentative):
  - Web service basics
    - SOAP
    - WSDL
    - UDDI
    - Composition & BPEL
    - Messaging
  - Web 2.0
    - REST
    - Mashups
  - SOA and integration architectures
  - Presentations from industry

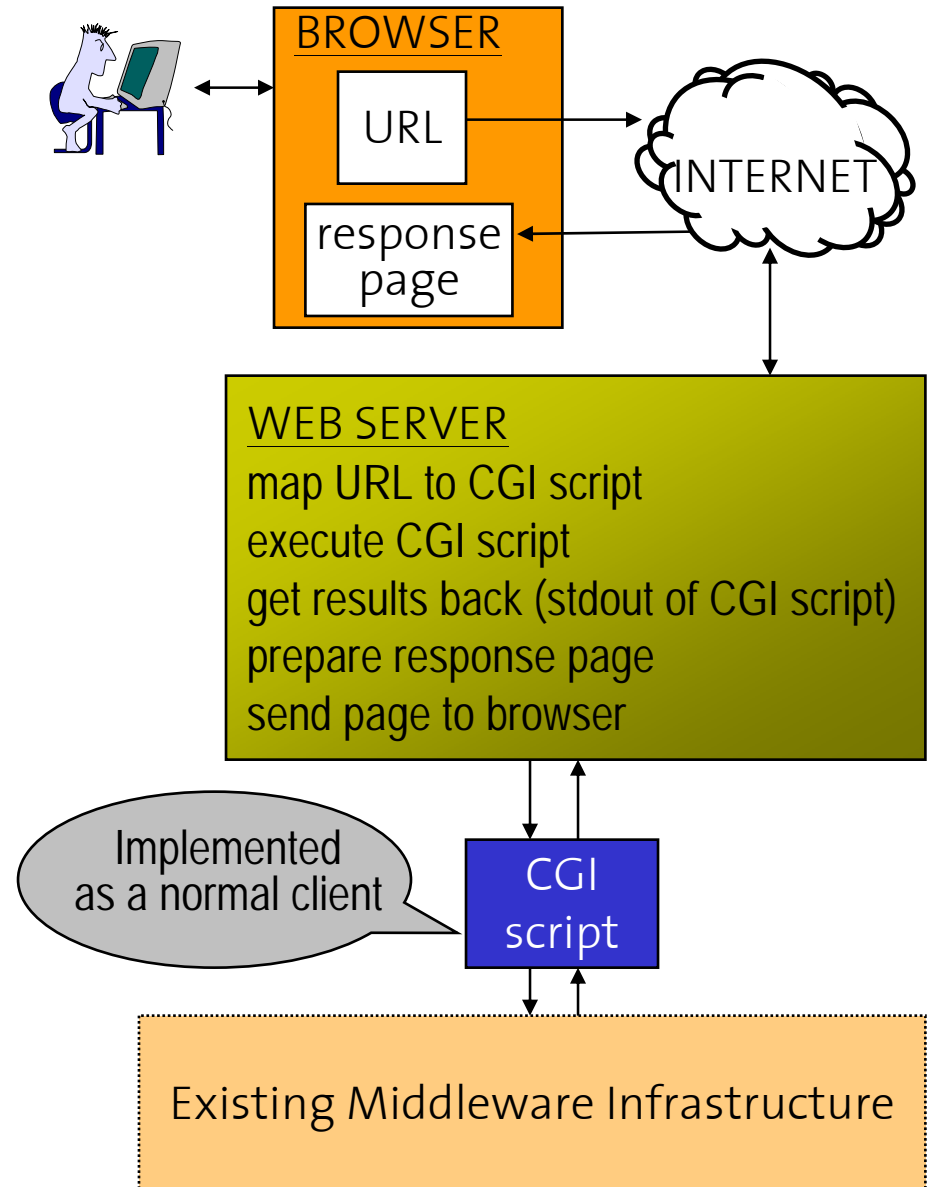
# The Web as software layer (N-tier)



- N-tier architectures result from connecting several three tier systems to each other and/or by adding an additional layer to allow clients to access the system through a Web server
- The Web layer was initially external to the system (a true additional layer); today, it is slowly being incorporated into a presentation layer that resides on the server side (part of the middleware infrastructure in a three tier system, or part of the server directly in a two tier system)
- The addition of the Web layer led to the notion of “application servers”, which was used to refer to middleware platforms supporting access through the Web

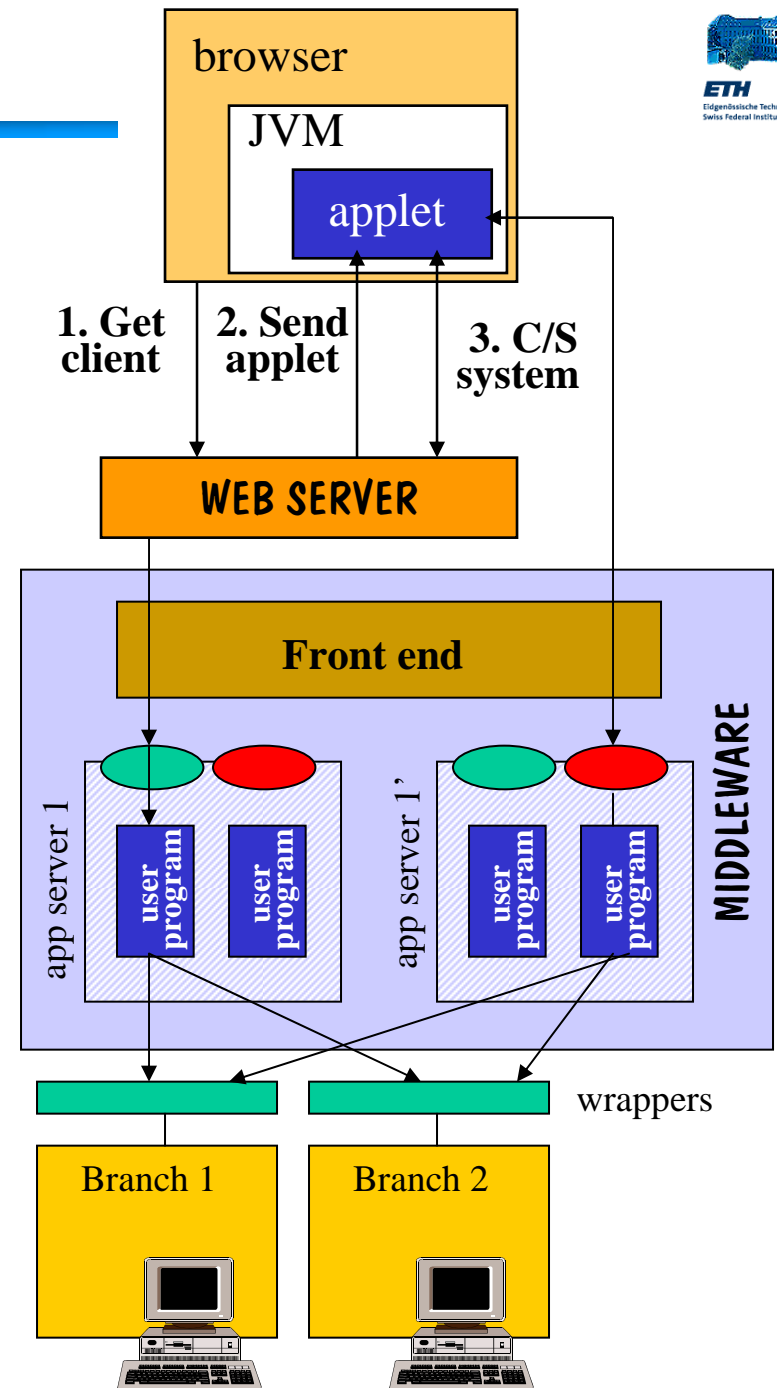
# WWW basics

- The earliest implementations were very simple and built directly upon the existing systems (client/server based on RPC, TP-Monitors, or any other form of middleware which allowed interaction through a programmable client)
  - 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



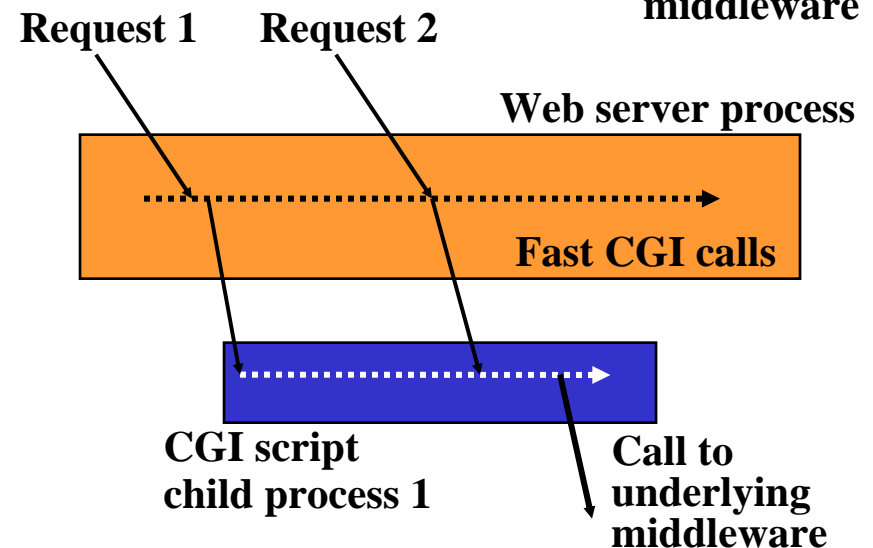
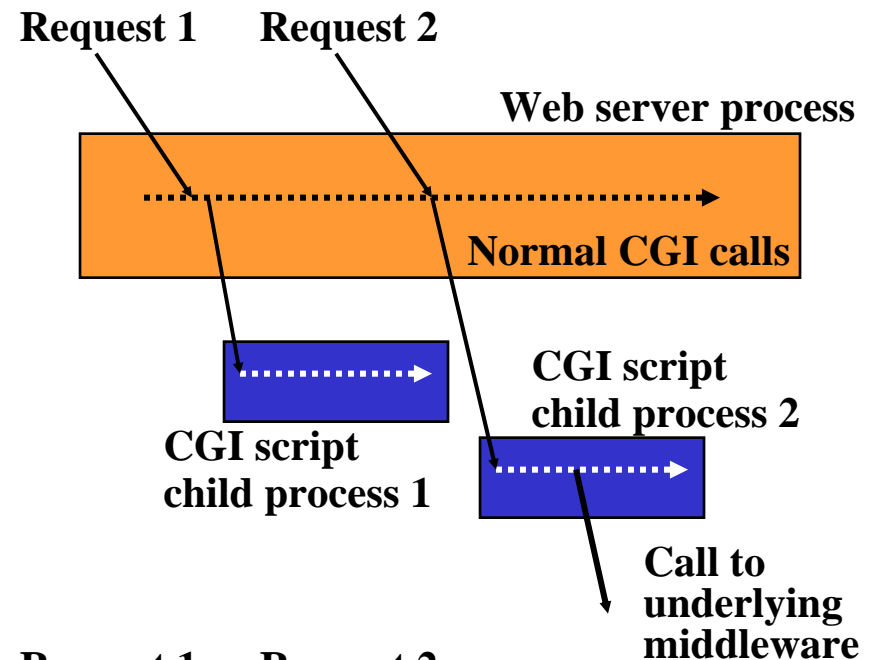
# Applets and clients

- The problem of the using a web browser as universal client is that it does not do much beyond displaying data (it is a thin client):
  - multiple interactions are needed to complete complex operations
  - the same operations must be done over and over again for all clients
  - the processing power at the client is not used
- By adding a JVM (Java Virtual Machine) to the browser, now it becomes possible to dynamically download the client functionality (an applet) every time it is needed
- The client becomes truly independent of the operating system and is always under the control of the server



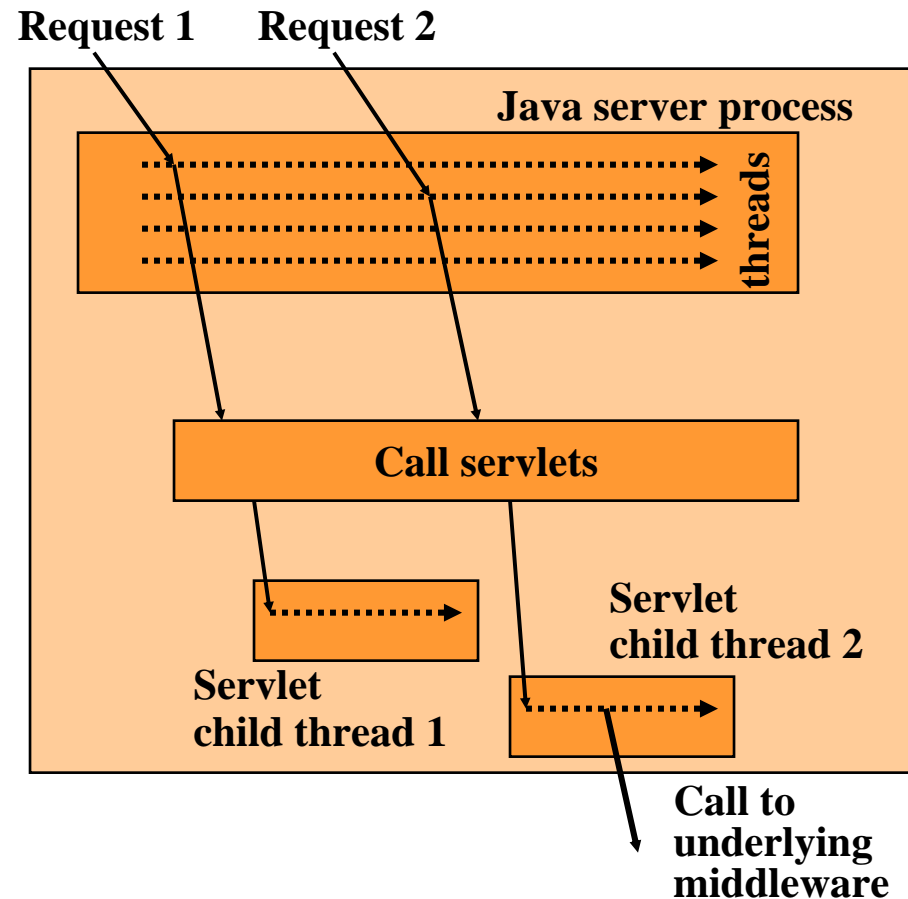
# Web server as a client of a EAI system

- CGI scripts were initially widely used as there was no other way of connecting the web server with the IT system so that it could do something beyond sending static documents
- However, CGI scripts have several problems that are not easy to solve:
  - CGI scripts are separate processes, requiring additional context switches when a call is made (and thereby adding to the overall delay)
  - Fast-CGI allows calls to be made to a single running process but it still requires two context switches
  - CGI is really a quick hack not designed for performance, security, scalability, etc.



# Servlets

- Servlets fulfill the same role as CGI scripts: they provide a way to invoke a program in response to an http request.
- However:
  - Servlets run as threads of the Java server process (not necessarily the web server) not as separate OS processes
  - unlike CGI scripts, that can be written in any language, Servlets are always written in Java (and are, therefore, portable)
  - can use all the mechanisms provided by the JVM for security purposes



# Servlets and HTML

## HTML request includes

```
< SERVLET NAME=MyServlet>  
  < PARAM NAME=param1 VALUE=val1>  
  < PARAM NAME=param2 VALUE=val2>  
  ...  
< /SERVLET>
```



## Servlet code

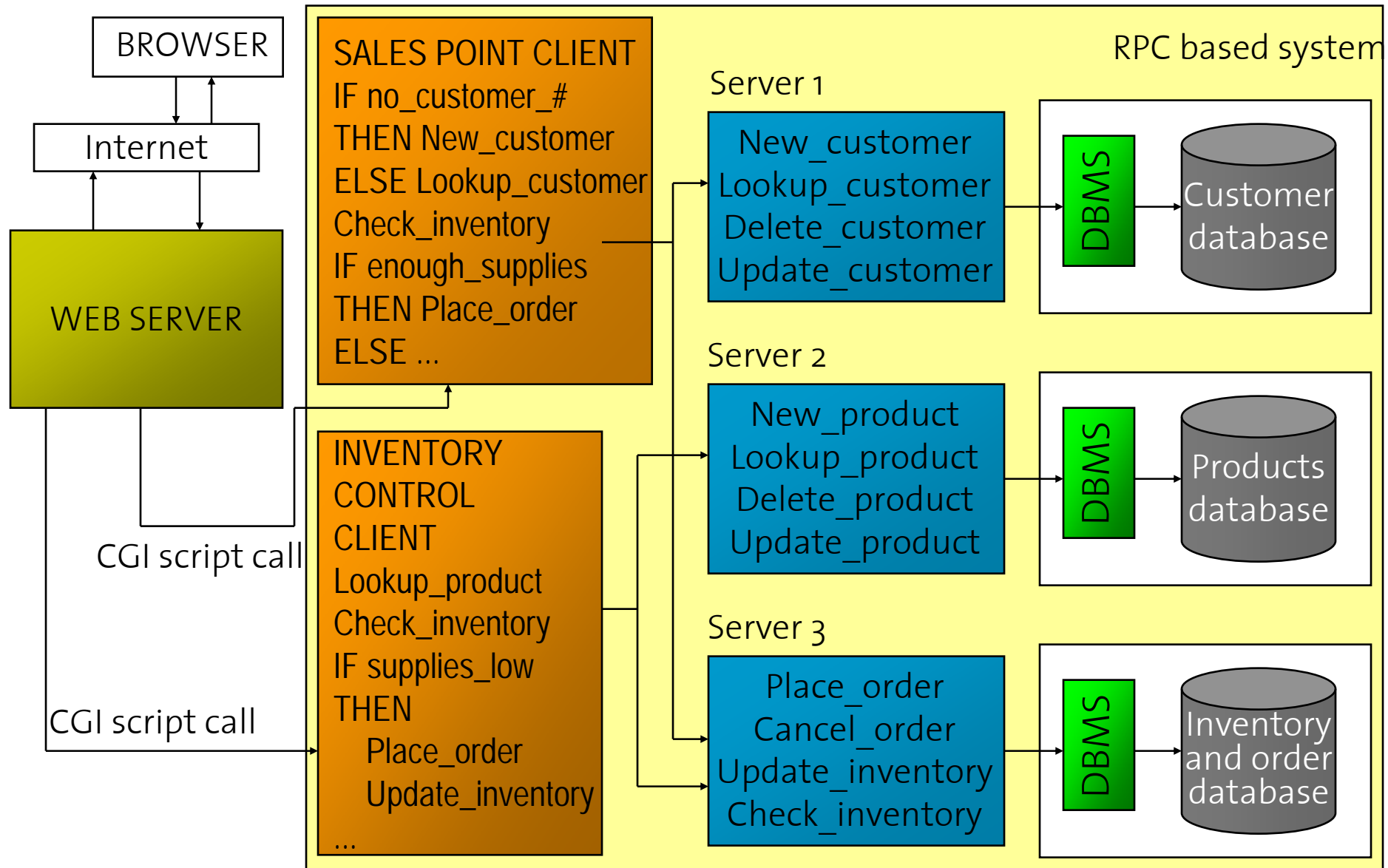
```
import java.servlet.*;  
public class MyServlet extends GenericServlet {  
  public void service (  
    ServletRequest request,  
    ServletResponse response  
  ) throws ServletException, IOException  
  {  
    ...  
  }  
  ...  
}
```



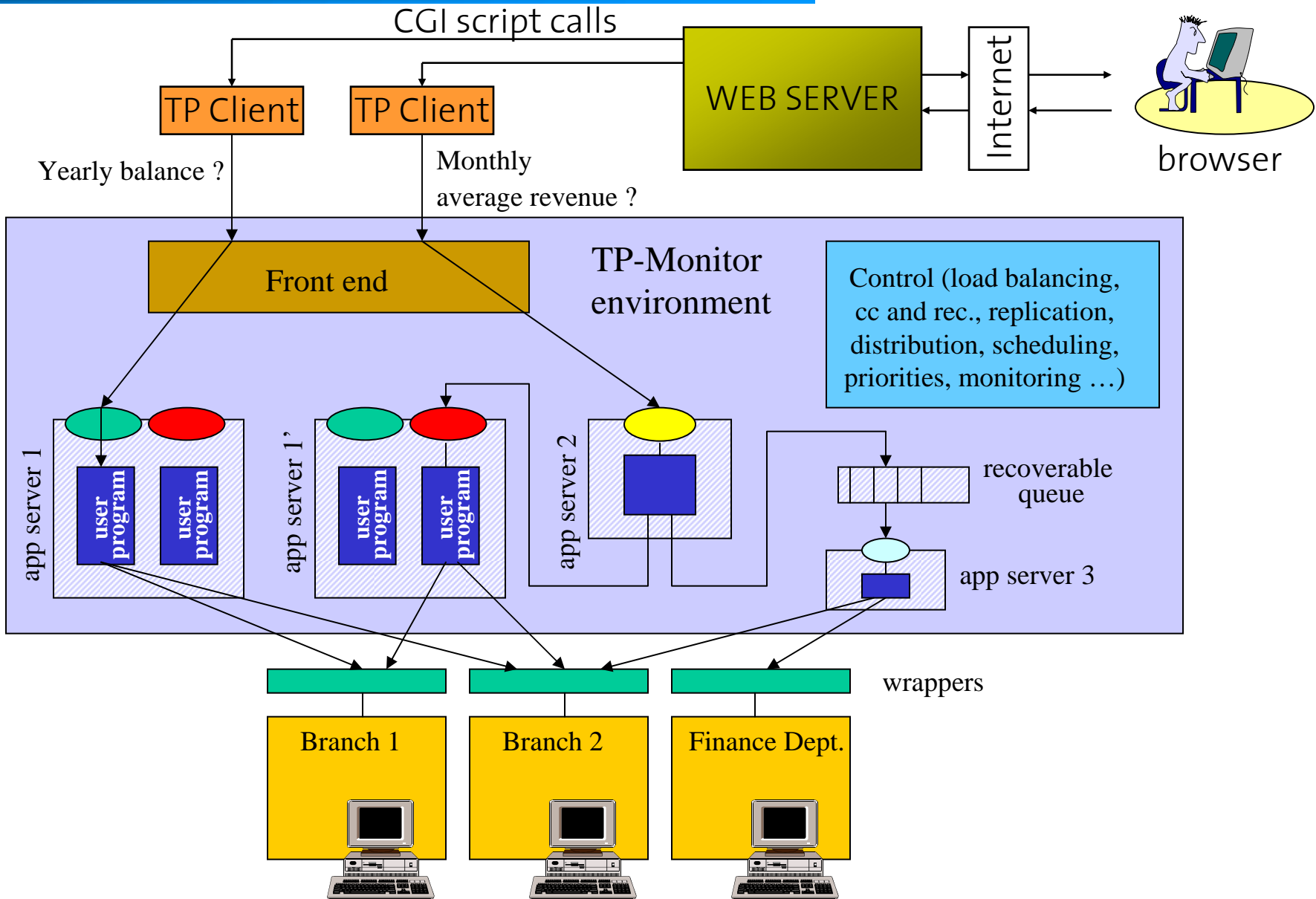
**HTML  
document**



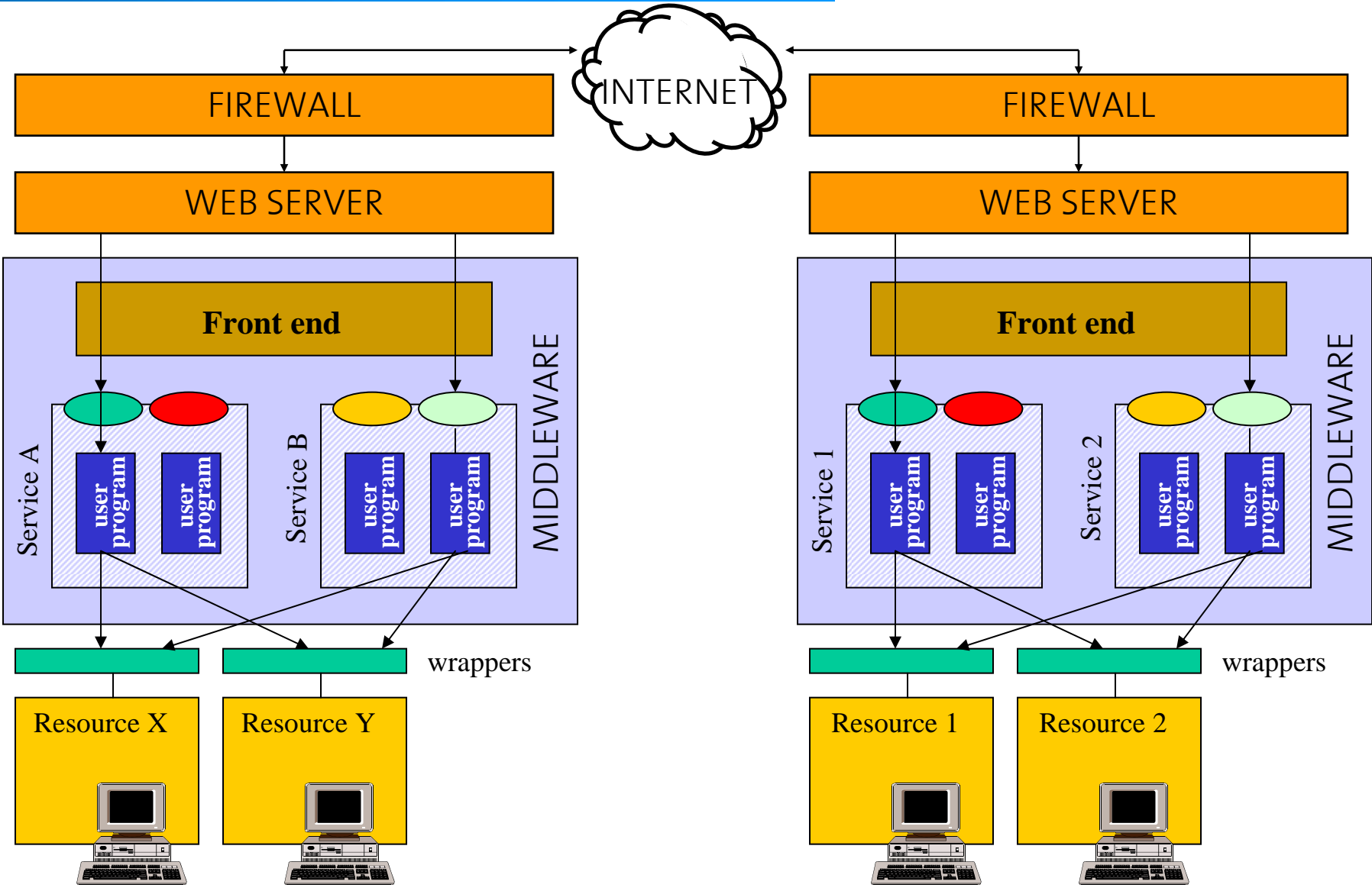
# Just one more layer ...



# ... on top of existing systems



# Business to Business (B2B)



# Limitations of the WWW

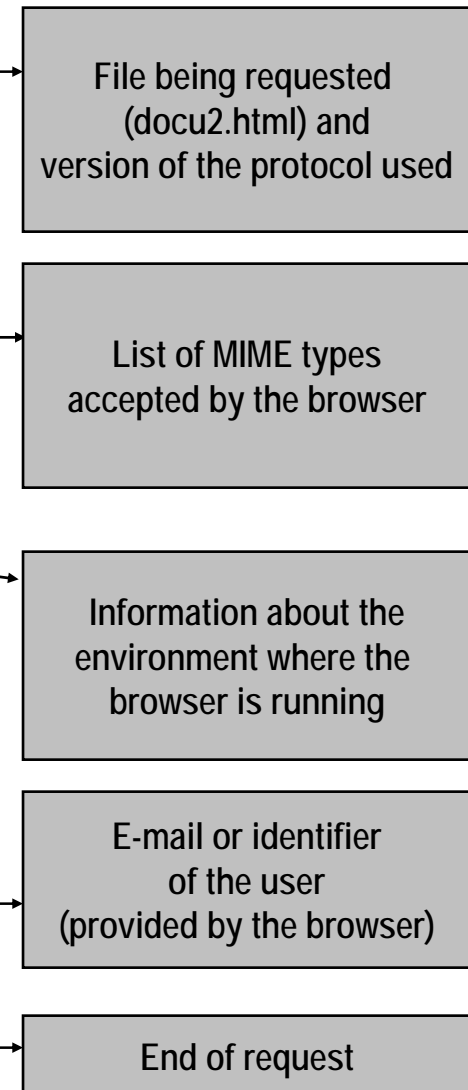
- HTTP was originally designed as a document exchange protocol (request a document, get the document, display the document). It lacked support for client side parameters
- Its architecture was originally designed with human users in mind. The document format (HTML) was designed to cope with GUI problems not with semantics. In EAI, the goal is almost always to remove humans from the business processes (mostly to reduce costs and to speed the process up). Strict formatting rules and tagging are key to exchanging messages across heterogeneous systems
- Interaction through document exchange can be very inefficient when the two sides of the interaction are programs (documents must be created, sent, parsed on arrival, information extracted, etc.). Unfortunately, http does not directly support any other form of interaction
- The initial WWW model was heavily biased towards the server side: the client (the browser) does not do much beyond displaying the document. For complex applications that meant
  - much more traffic between client and server
  - high loads at the server as the number of users increases

# 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 *
```

- 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.
    
```

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

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

# 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:  
URL:  
/cgi-bin/post-query?name=...&email=...

As before

Data provided through the form and sent back to the server

...

# 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.
- There are many examples of B2B interactions. The most basic one is a “purchase order” whereby a company directly places an order with another company. If done correctly, even this basic interaction can become a very powerful advantage for a company.
- The problem is how to implement such a system:
  - the client is 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 existing systems, it is not possible to homogenize the supporting platforms
  - the Internet is cheap but open to everybody (unlike leased lines that are expensive but private)
- Existing systems/protocols are not really designed for such type of interactions

# Contents and presentation

- HTML is a tag language designed to describe how a document should be displayed (the visual format of the document).
- HTML is one of the many tag languages that exist, some of them having being in use before HTML even existed
- Tag languages have been developed and are used in many industries (aircraft manufacturing, semiconductors, computer manuals). 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.

```

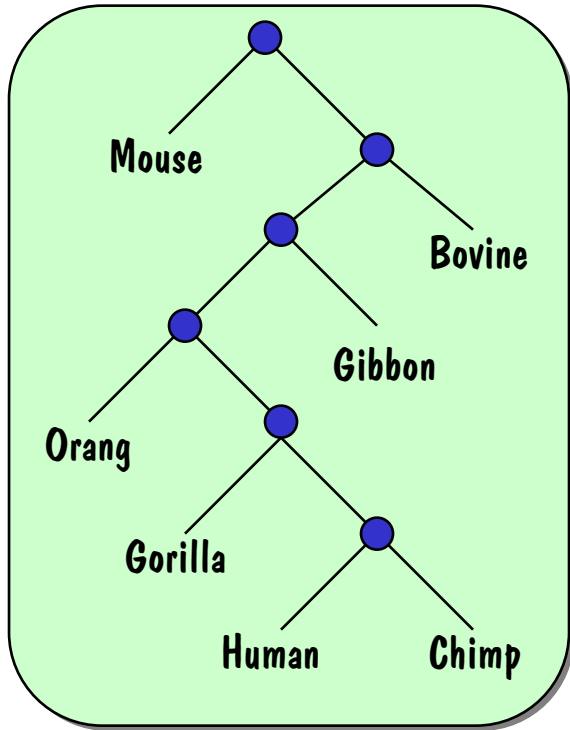
<h2>Table of contents</h2><a name=TOC></a>
<ul>
<li><a href="SG.htm">1 A Gentle Introduction to
SGML</a></li>
<li><a href="SG11.htm">2 What's Special about
SGML? </a></li>
<ul>
<li><a href="SG11.htm#SG111">2.1 Descriptive
Markup</a></li>
<li><a href="SG11.htm#SG112">2.2 Types of
Document</a></li>
<li><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>

```

# HTML and XML

- HTML only provides primitives for formatting a document with a human user in mind
- Using HTML there is no way to indicate what are the contents of a document (its semantics)
- For instance, a query to Amazon.com returns a book and its price as an HTML document
  - a human has no problem interpreting this information once the browser displays it
  - to parse the document to automatically identify the price of the book is much more complicated and an ad-hoc procedure (different for every bookstore)
- B2B applications require documents that are much more structured so that they can be easily parsed and the information they contain extracted
- To cope with this requirement, the XML standard was proposed
- Important aspects of XML:
  - XML is not an extension to HTML
  - XML is a version of SGML that can be implemented in a Web browser
  - XML is not a language but a “meta-language” used to define markup languages
  - XML tags have no standard meaning that can be interpreted by the browser. The meaning must be supplied as an addition in the form of a style sheet or program

# Data structures in XML



```
(('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;
```

Data to send

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

DTD File

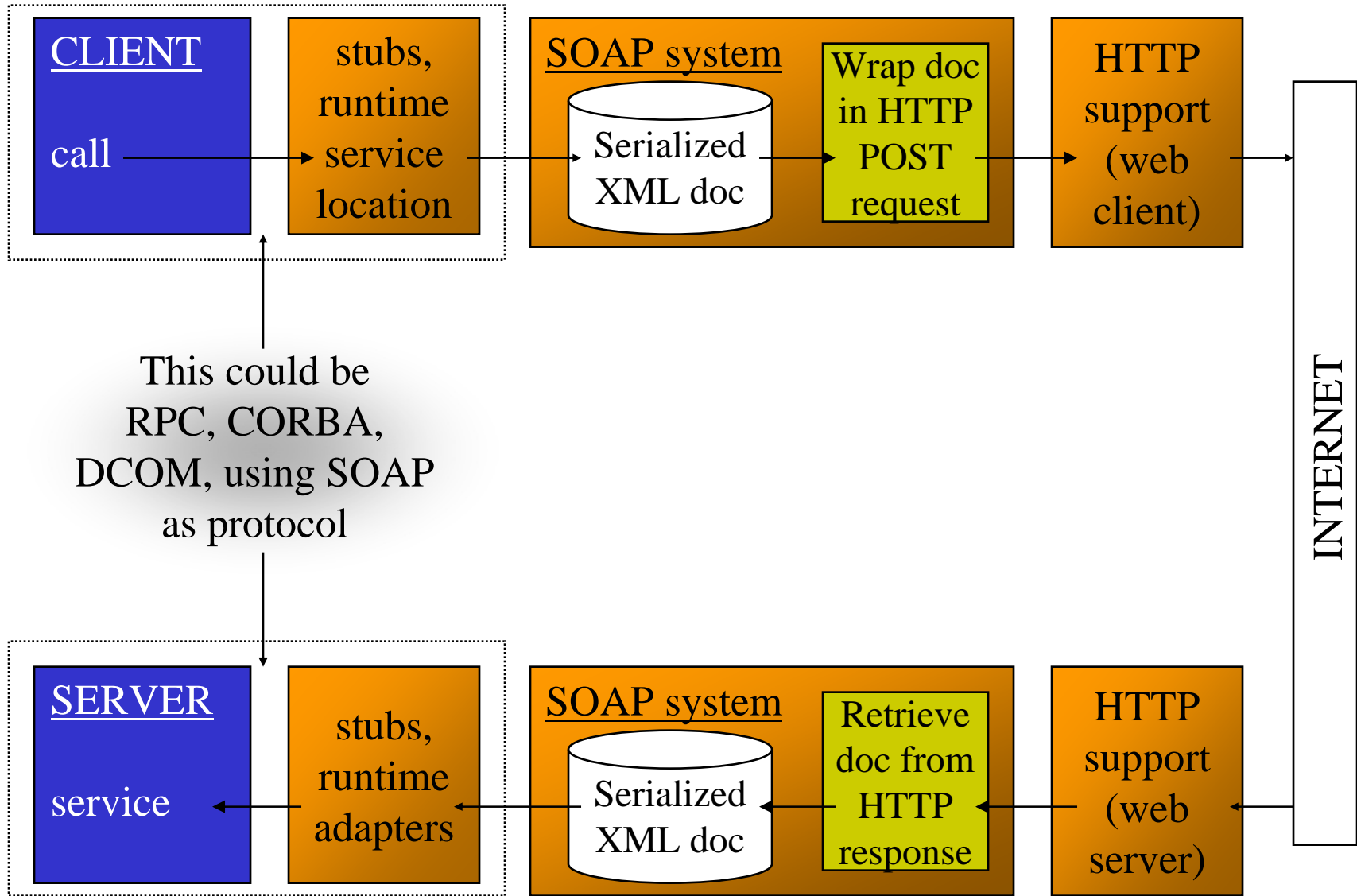
```
<?xml version="1.0" ?>
<!DOCTYPE trees SYSTEM "treefile.dtd">
<trees>
<tree>
<branch>
<node>
<specie>
'Mouse'
</specie>
</node>
<length>
0.792449
</length>
</branch>
<branch>
<node>
<branch>
<node>
<branch>
<node>
<branch>
<node>
<specie>
'Human'
</specie>
</node>
...
</tree>
</trees>
```

XML File

# DTDs and documents

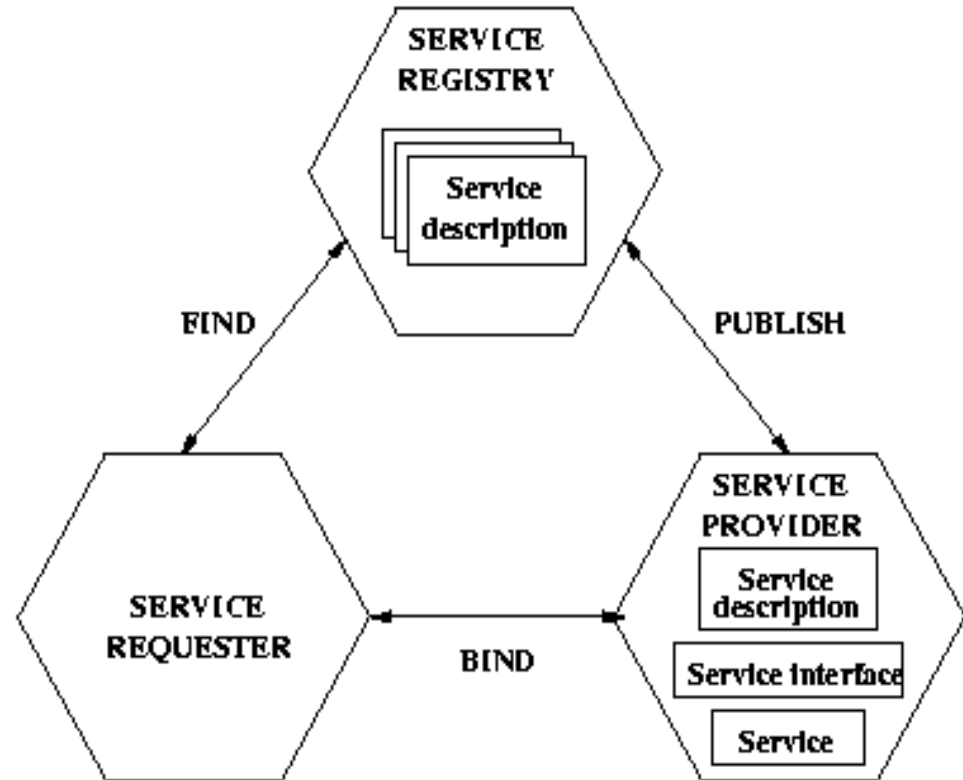
- The goal of XML is to provide a standardized way to specify data structures so that when data is exchanged, it is possible to understand what has been sent
- The Document Type Definition (DTD) specifies how the data structure is described: processing instructions, declarations, comments, and elements
- Using the DTD, the XML document can be correctly interpreted by a program by simply parsing the document using the grammar provided by the DTD
- The idea is similar to IDL except that instead of defining parameters as combinations of standard types, a DTD describes arbitrary documents as semi-structured data
- Using XML is possible to exchange data through HTTP and Web servers and process the data automatically
- Note that the use of XML reduces the universality of the browser since now a browser needs additional programs to deal with specific markup languages developed using XML (somewhat similar to plug-ins but more encompassing in terms of functionality)
- However, this is not much of a problem since the browser is for humans while XML is for automated processing
- XML can be used as the intermediate language for marshalling/serializing arguments when invoking services across the Internet

# Web services



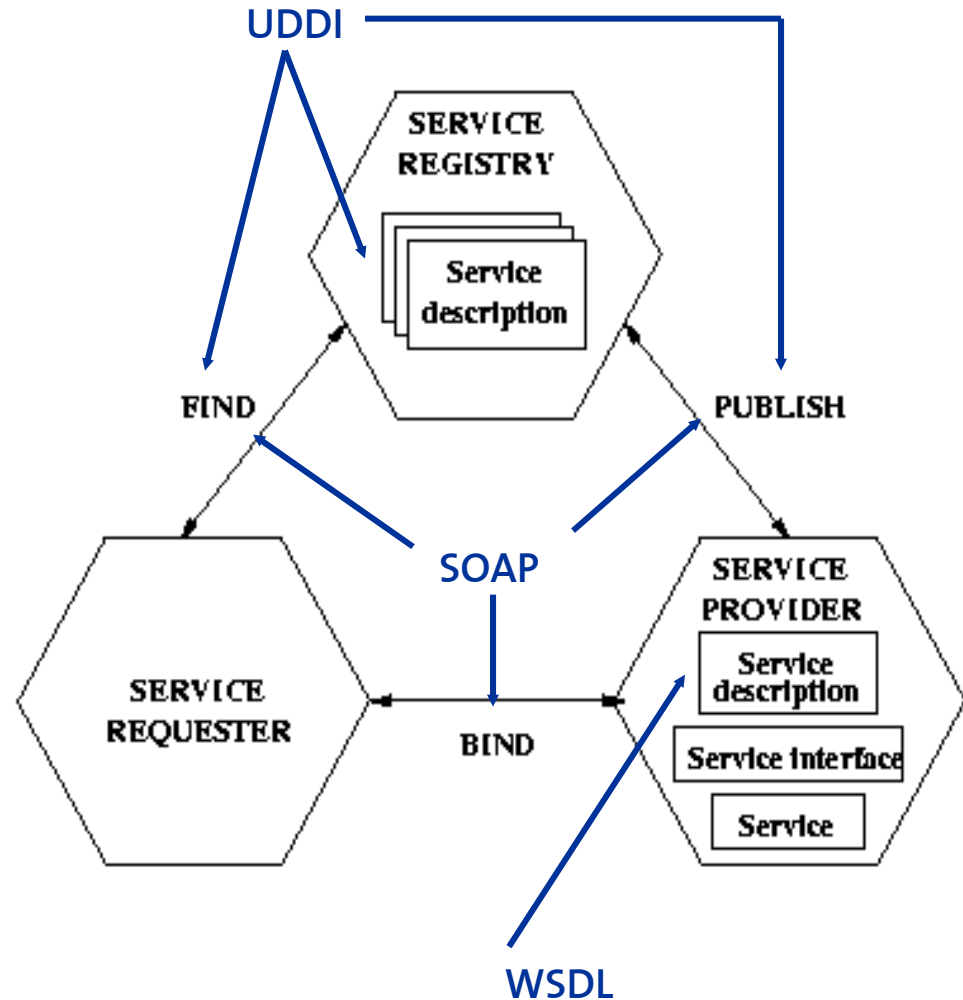
# Web Services Architecture

- A popular interpretation of Web services is based on IBM's *Web service architecture* based on three elements:
  1. Service **requester**: The potential user of a service (the client)
  2. Service **provider**: The entity that implements the service and offers to carry it out on behalf of the requester (the server)
  3. Service **registry**: A place where available services are listed and that allows providers to advertise their services and requesters to lookup and query for services



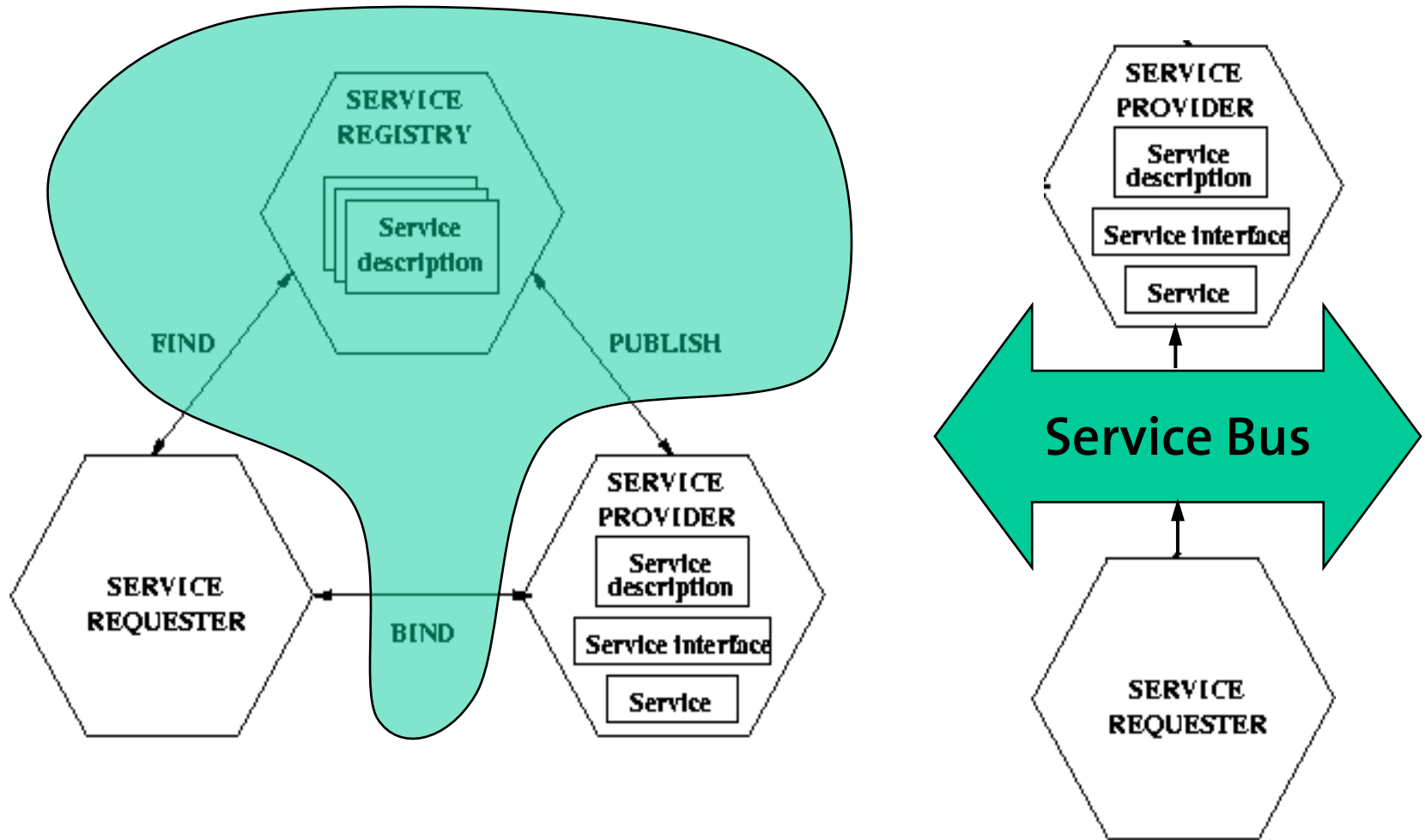
# Main Web Services Standards

- The Web service architecture proposed by IBM is based on two key concepts:
  - architecture of existing synchronous middleware platforms
  - current specifications of SOAP, UDDI and WSDL
- The architecture has a remarkable client/server flavor
- It reflects only what can be done with
  - SOAP (Simple Object Access Protocol)
  - UDDI (Universal Description and Discovery Protocol)
  - WSDL (Web Services Description Language)



# The Service Bus

- The service bus can be seen as a refactoring of the basic Web service architecture, where a higher degree of loose coupling has been added.



# 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

# WS Standards and Specifications

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

- 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 applications and follows on previous ideas such as software bus, IT backbone, or enterprise bus
  
- 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 (unlike Web services, CORBA, RPC, etc.)

# The novelty behind SOA

- The concept of SOA is not new:
  - Message oriented middleware
  - Message brokers
  - Event based architectures
  
- The current context is different
  - Emergence of standard interfaces (Web services)
  - Emphasis on simplifying development (automatic)
  - Use of complex underlying infrastructure (containers, middleware stacks, etc.)
  
- Interest in SOA arises from a number of reasons:
  - Basic technology in place
  - More clear understanding of distributed applications
  - The key problem is integration not programming

# The need for SOA

- Most companies today have a large, heterogeneous IT infrastructure that:
  - Keeps changing
  - Needs to evolve to adopt new technology
  - Needs to be connected of that of commercial partners
  - Needs to support an increasing amount of purposes and goals
  
- This was the field of Enterprise Application Integration using systems like CORBA or DCOM. However, solutions until now suffered from:
  - Tightly integrated systems
  - Vendor lock-in (e.g., vendor stacks)
  - Technology lock-in (e.g., CORBA)
  - Lack of flexibility and limitations when new technology arises (e.g., Internet)
  
- SOA is an attempt to build on standards (web services) to reduce the cost of integration
  
- It introduces very interesting possibilities:
  - Development by composition
  - Large scale reuse
  - Frees developers from “lock-in” effects of various kinds

# 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)
  - Event based interaction (no longer synchronous models)
  - Message based exchanges (no RPC)
  - Composition and orchestration

# SOA and web services

There is no problem in system **design** that cannot be solved by adding a level of indirection.

There is no **performance** problem that cannot be solved by removing a level of indirection.



Take advantage of  
**Middleware** but let the  
system decide what  
to use

- WS Invocation Framework
  - Use WSDL to describe a service
  - Use WSIF to let the system decide what to do when the service is invoked:
    - If the call is to a local EJB then do nothing
    - If the call is to a remote EJB then use RMI
    - If the call is to a queue then use JMS
    - If the call is to a remote Web service then use SOAP and XML
  - There is a single interface description, the system decides on the binding
  - This type of functionality is at the core of the notion of Service Oriented Architecture