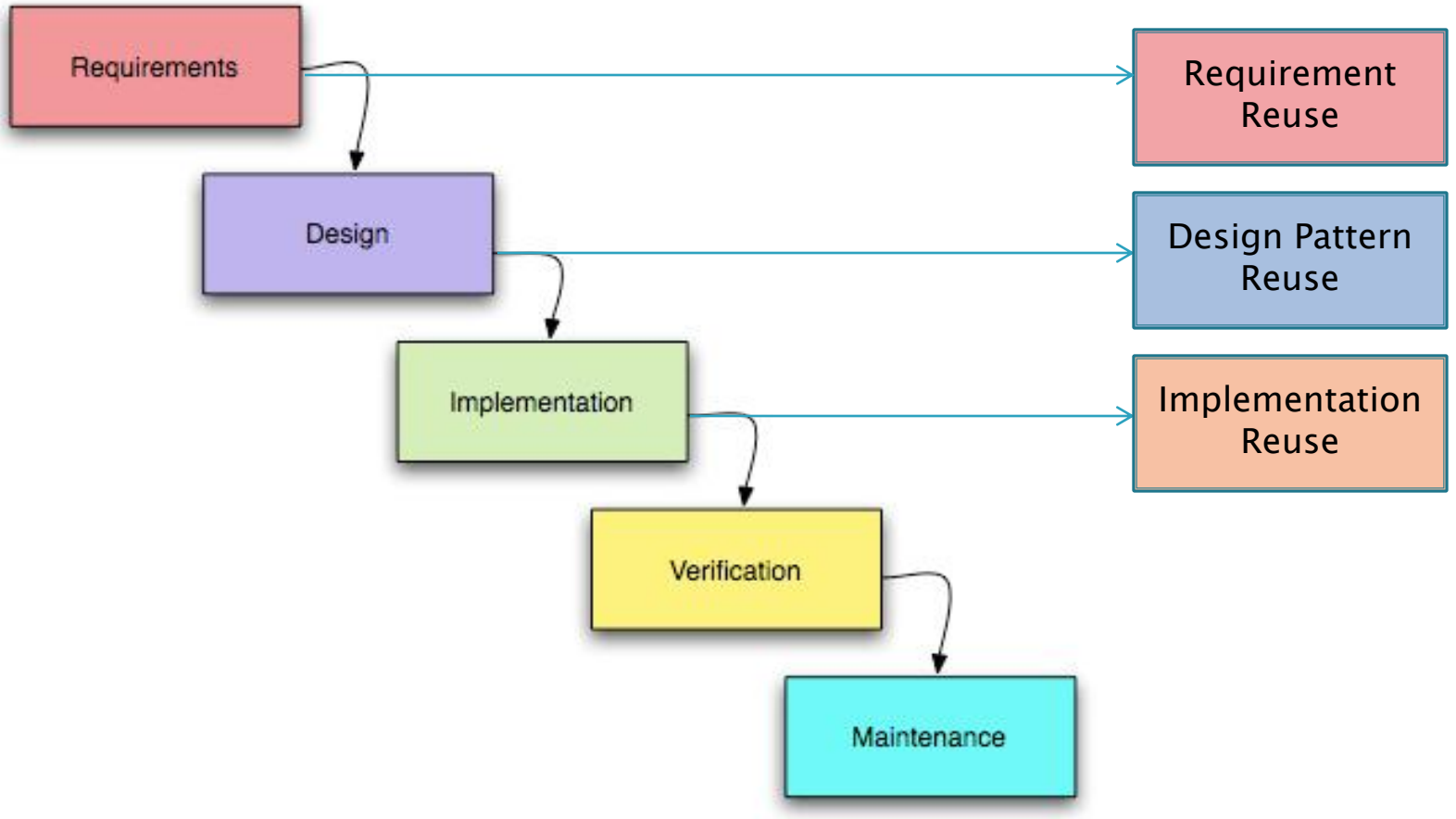
Module 4
Implementation Reuse (Part 3)

Dr. Shiping Chen
Course Email: chen.shiping@yahoo.com

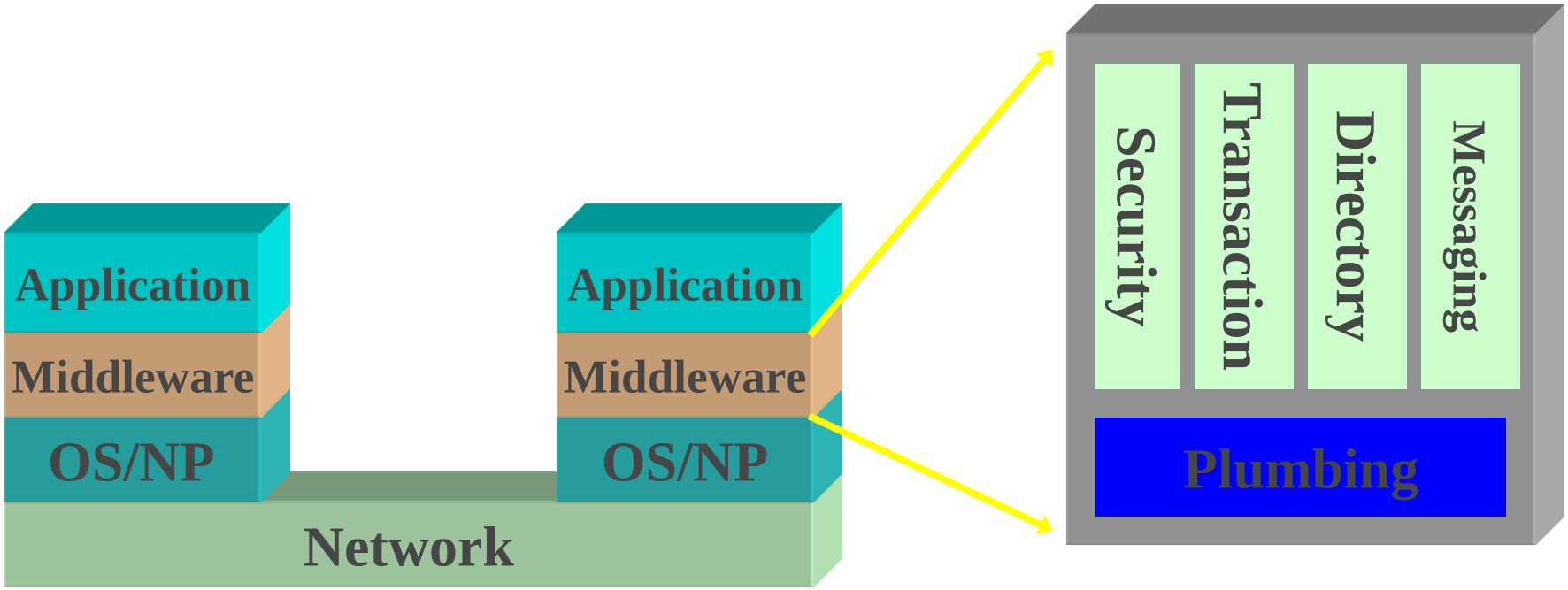
Where Are We?



Agenda

- **Overview of Web/App Servers**
- Web Services Technology
  - Overview
  - SOAP
  - WSDL
  - UDDI
  - Advices for programming

What is Middleware?



Middleware = Software Plumbing + Associated Services

Such as: CORBA, J2EE, .NET, Web Services, etc.

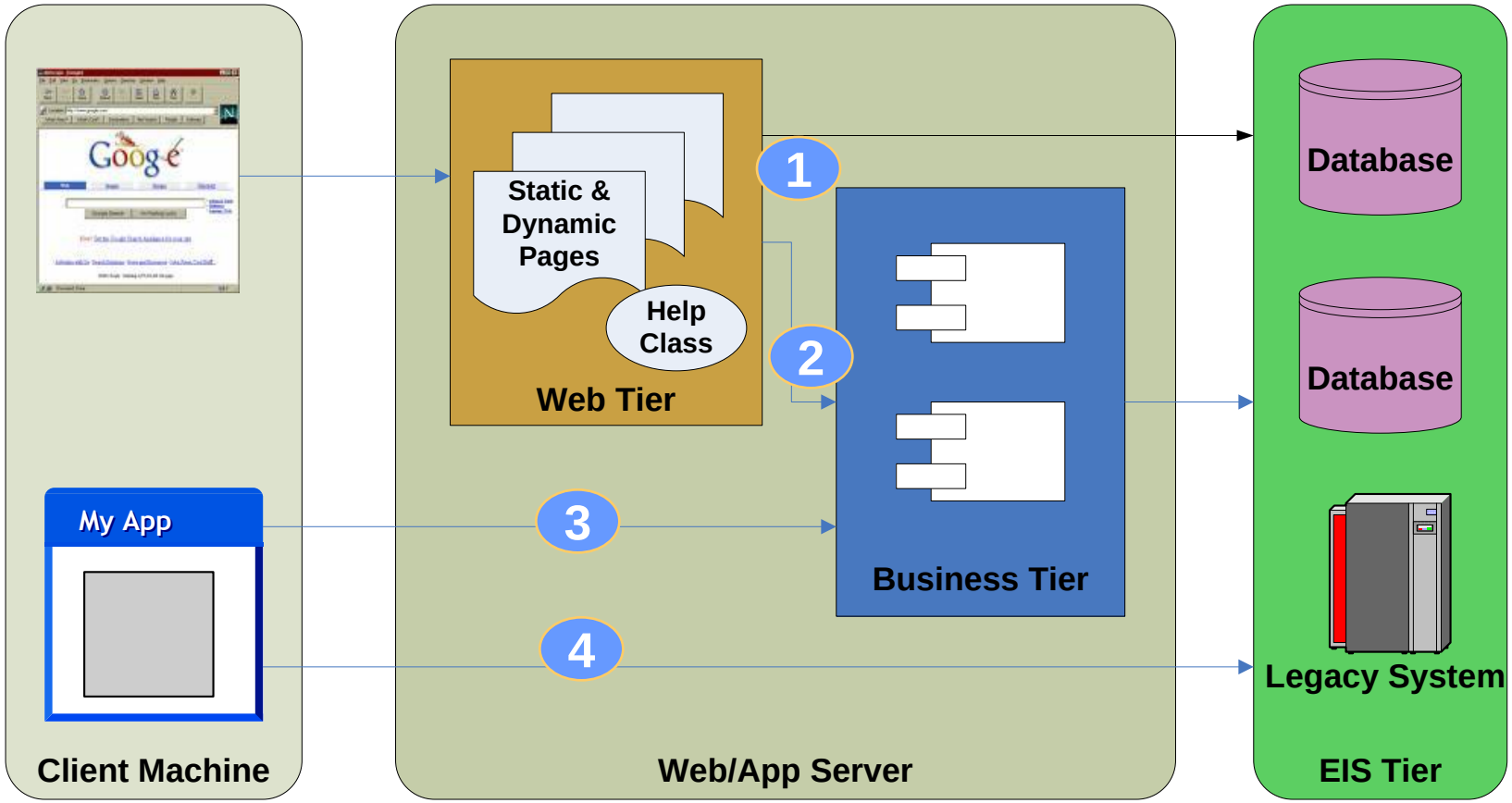
Why Middleware?

- To build standard distributed applications easily !
- Localized Programming/distribution-transparent
- High level protocols: IIOP, RMI, DCOM, SOAP
- Common services: Directory, Transaction, Messaging, Security and Concurrency Management.
- Standard Interoperation, esp. web services!

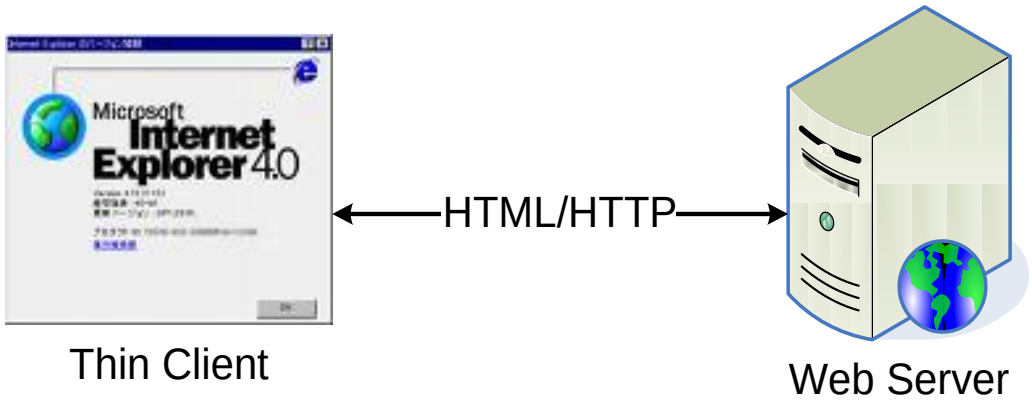
Middleware = 'Everyware'

- Banking and Finance
- Transportation
- Healthcare
- Insurance
- Telecommunication
- Internet Applications
- And other real-time systems
- ……..

Overview of Web Application Systems

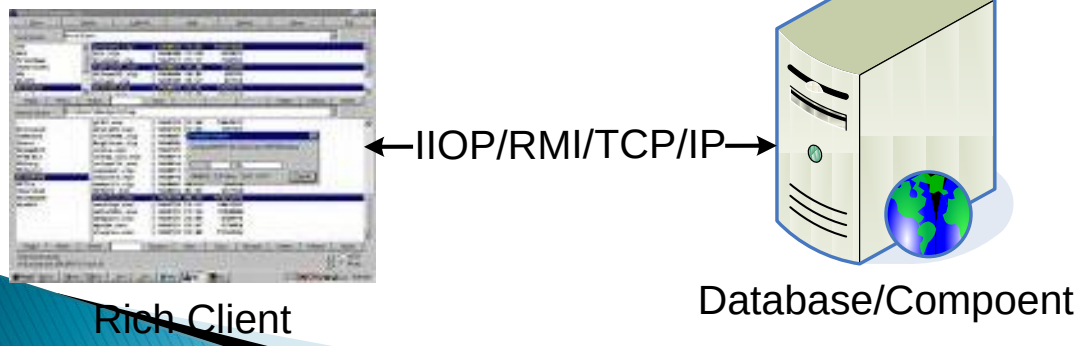


Thin Client vs. Rich Client



Also called web client:

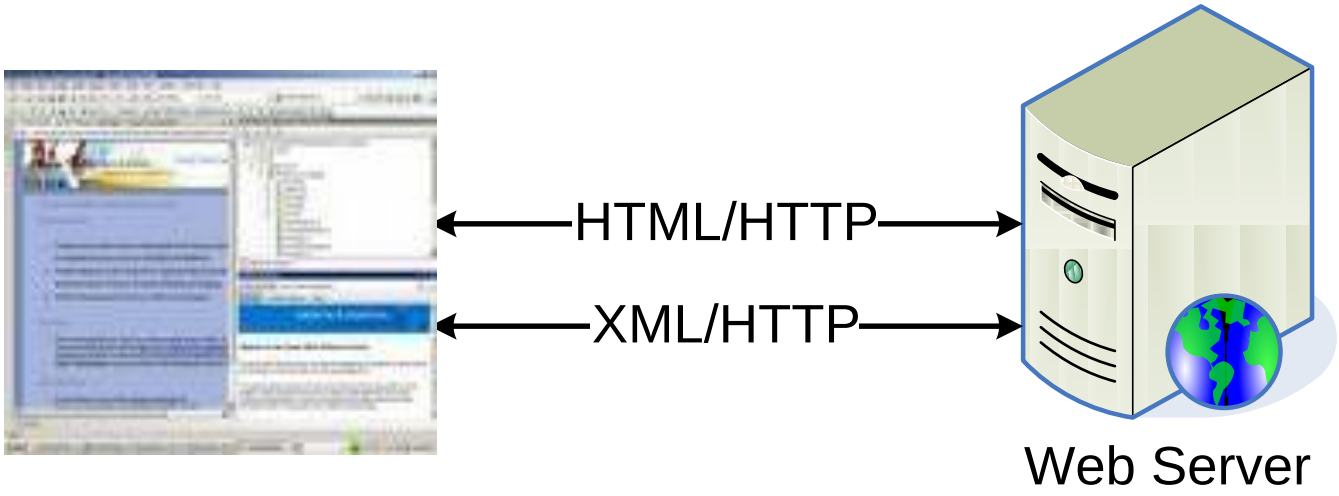
- Limited functionalities & access to local resources
- Poor performance, due to blocking
- little user experience expected
- Easy to deploy and maintain



Also called fat/thick client:

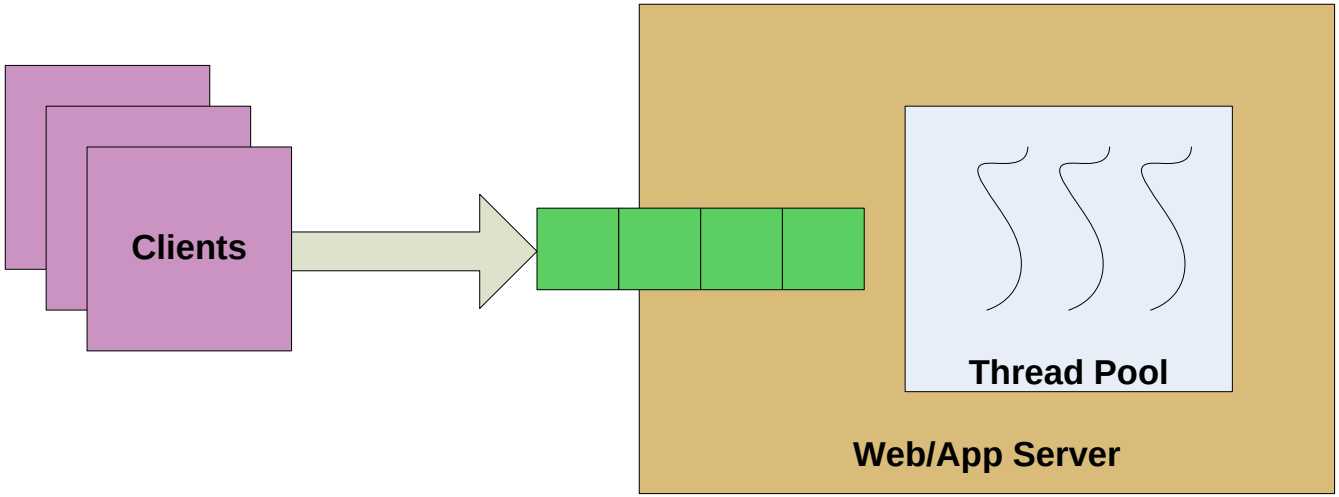
- Rich functionalities and ability to access local resources
- Fast
- Rich user experience expected
- Hard to deploy and maintain

Smart Client



- Making thin client rich and smart with:
  - JavaScript
  - DHTML/XHTML
  - Ajax/XML

Concurrency in Servers

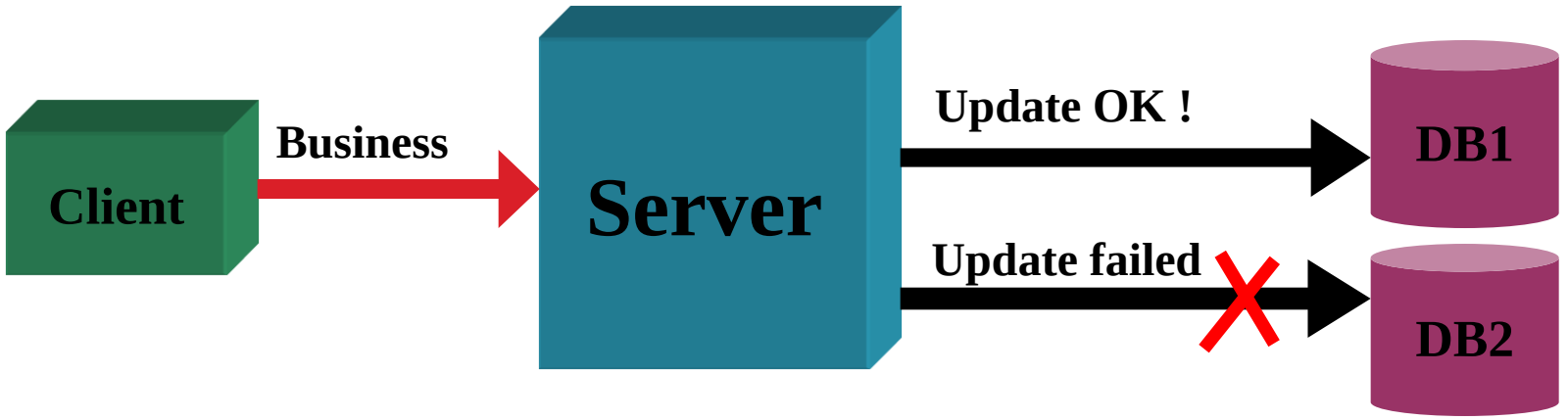


- Most web/app servers are multi-threaded so that multiple clients' requests can be handled in parallel, i.e. the more threads, the more concurrency/parallelisms
- But the more threads/concurrency, the more contentions on server-sides resources, such as CPU, memory, database etc, which can overweight the benefits from concurrency
- As a result, a thread pool is usually used to control the degree of concurrency;

Transactions

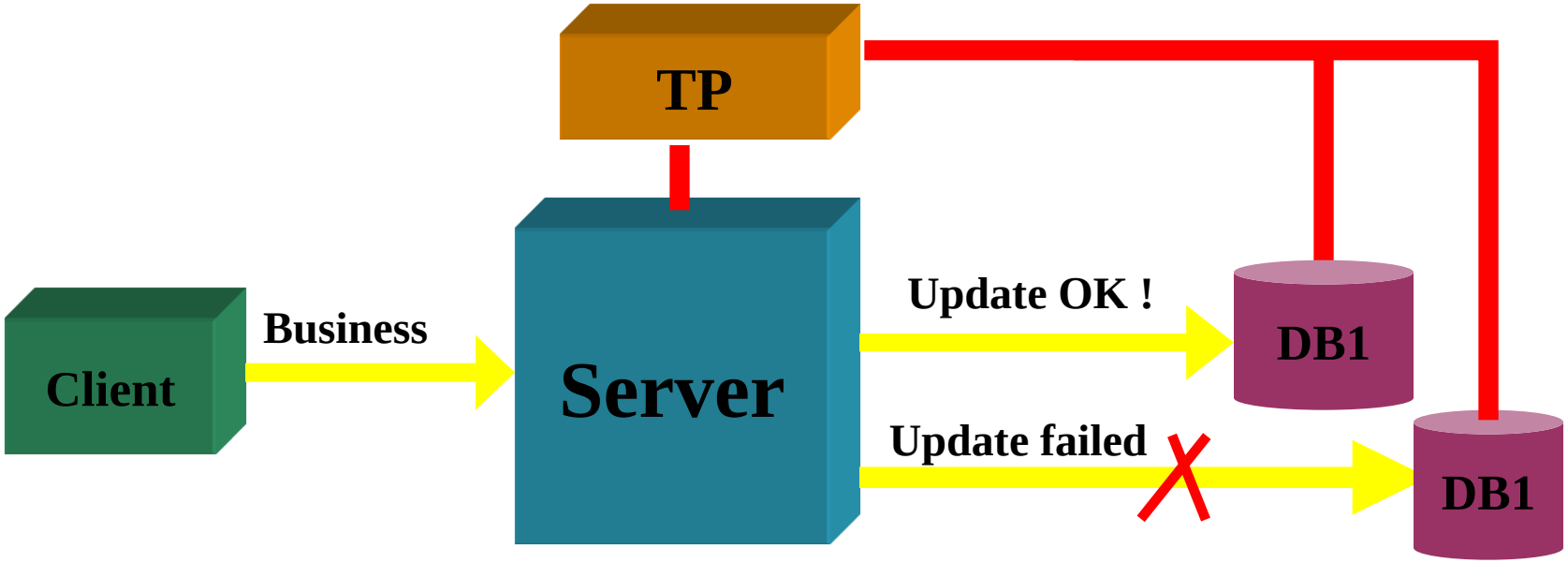
- What is Transaction?
  - A mechanism to maintain the integrity of business
- ACID properties of a transaction:
  - **Atomic**: either all success or all fail.
  - **Consistent**: from one consistent state to another.
  - **Isolated**: partial results hidden from others in a systems.
  - **Durable**: The results of a transaction are persistent.

An Classic Transaction Example



A system without transaction context

An Classic Transaction Example (Cont'd)



A system within transaction context

With Transaction, the server is able to ensure the multiple operations: either all successful, or nothing happened.

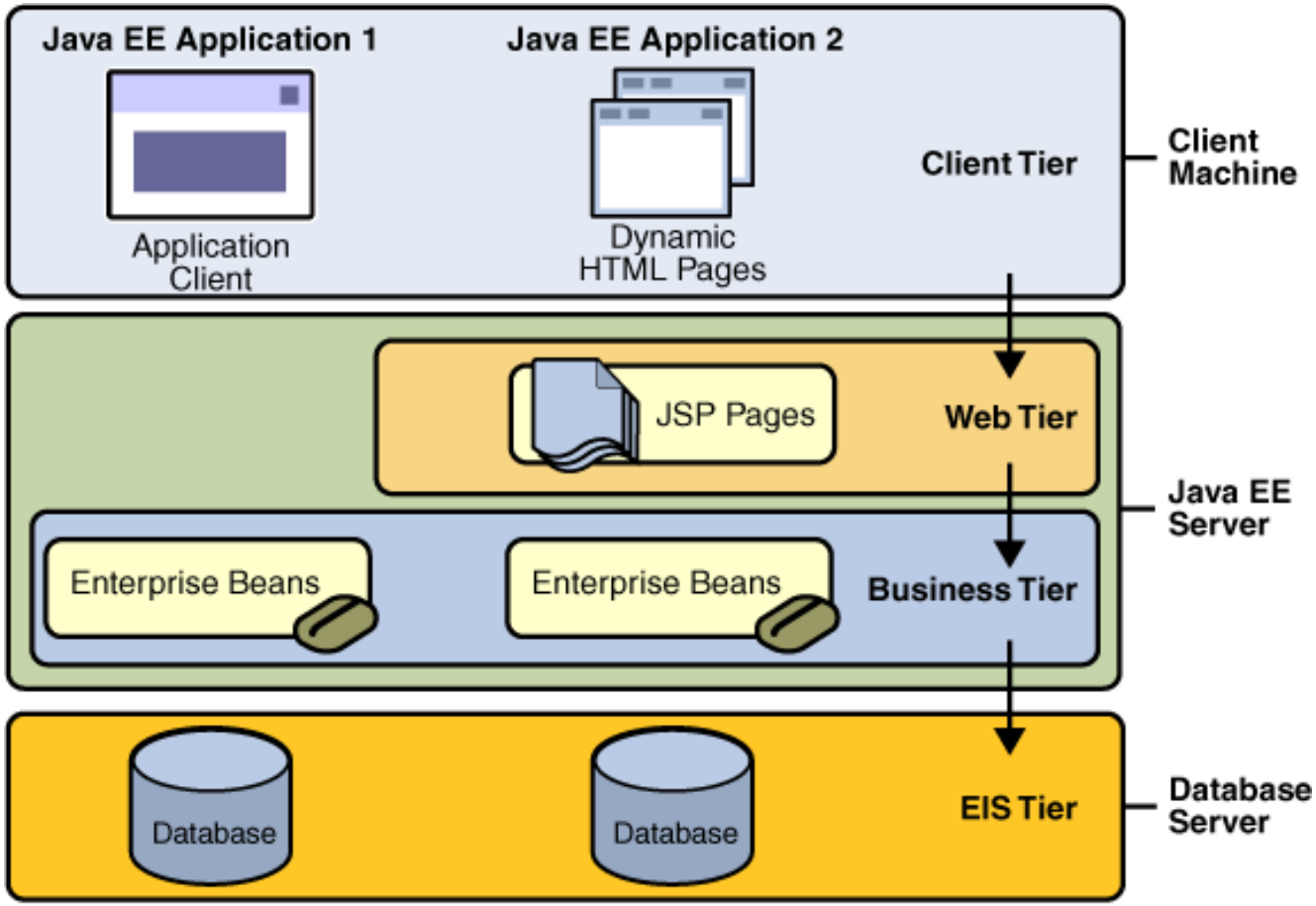
Server-Side Technologies: Specifications and Vendors

- Leaders
  - J2EE (Java 2 Enterprise Edition)
    - Based on a public-domain specification
    - See J2EE Material by SUN: http://java.sun.com/javaee/5/docs/tutorial/doc/
    - Across Multiple Platforms, but must be in Java
  - Microsoft Enterprise Technologies
    - Based on Microsoft Window technologies .NET Framework, COM+, MSMQ
    - See http://www.microsoft.com/net/TechnicalResources.aspx
    - Support multiple programming languages (C#, VB), but must be on Windows

J2EE Case Study

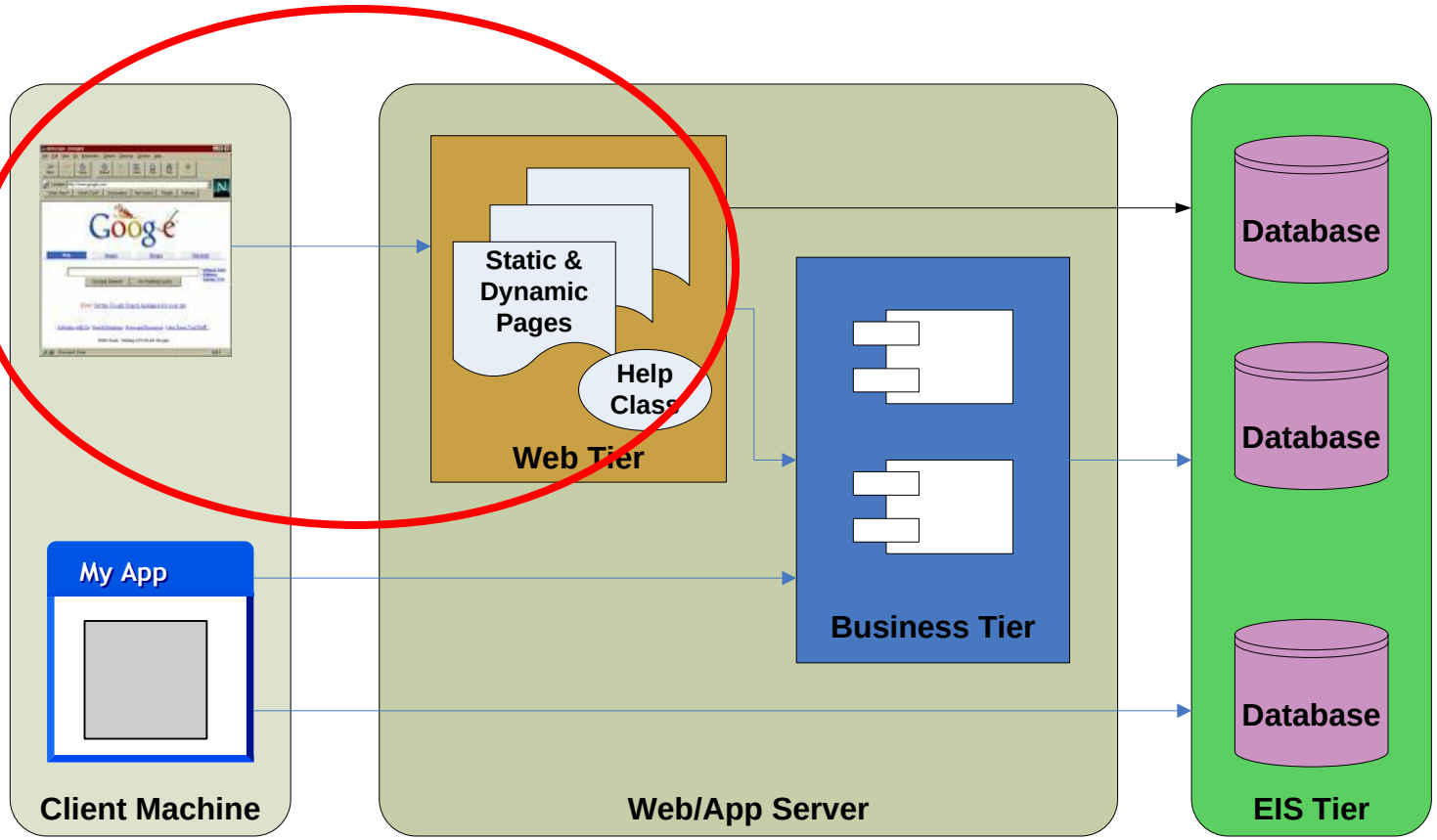
- J2EE Overview
- Basic Components/Technologies
  - Servlets: the fundamental J2EE web component
  - JSP: JavaServer Pages
  - JDBC: Java DataBase Connectivity
- Advanced Topics
  - EJB: Enterprise Java Beans (already learnt)
  - JMS: Java Messaging Services

J2EE Overview 3/3

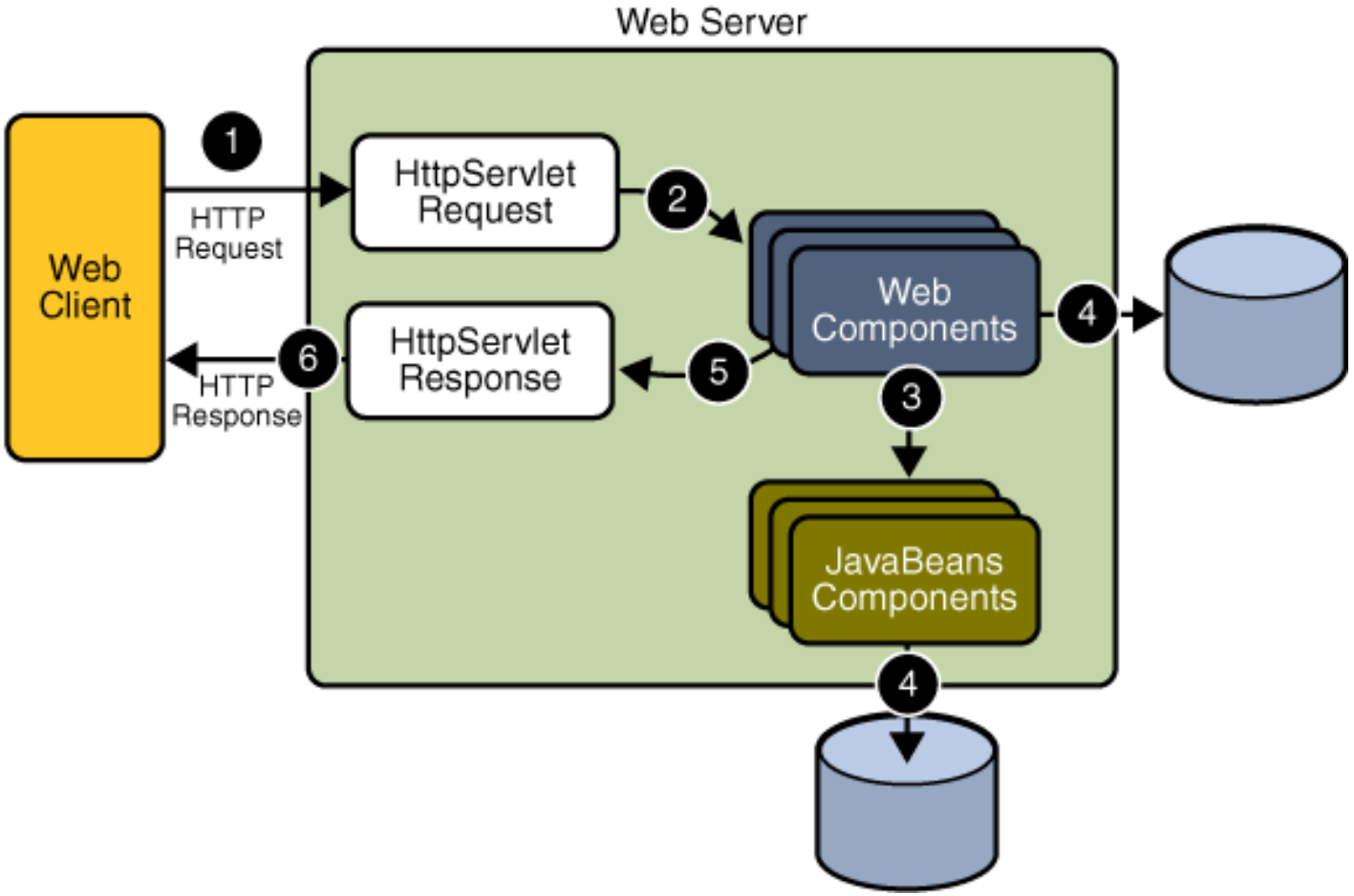


Source: http://java.sun.com/javaee/5/docs/tutorial/doc/

Let us Talk about Web Tier

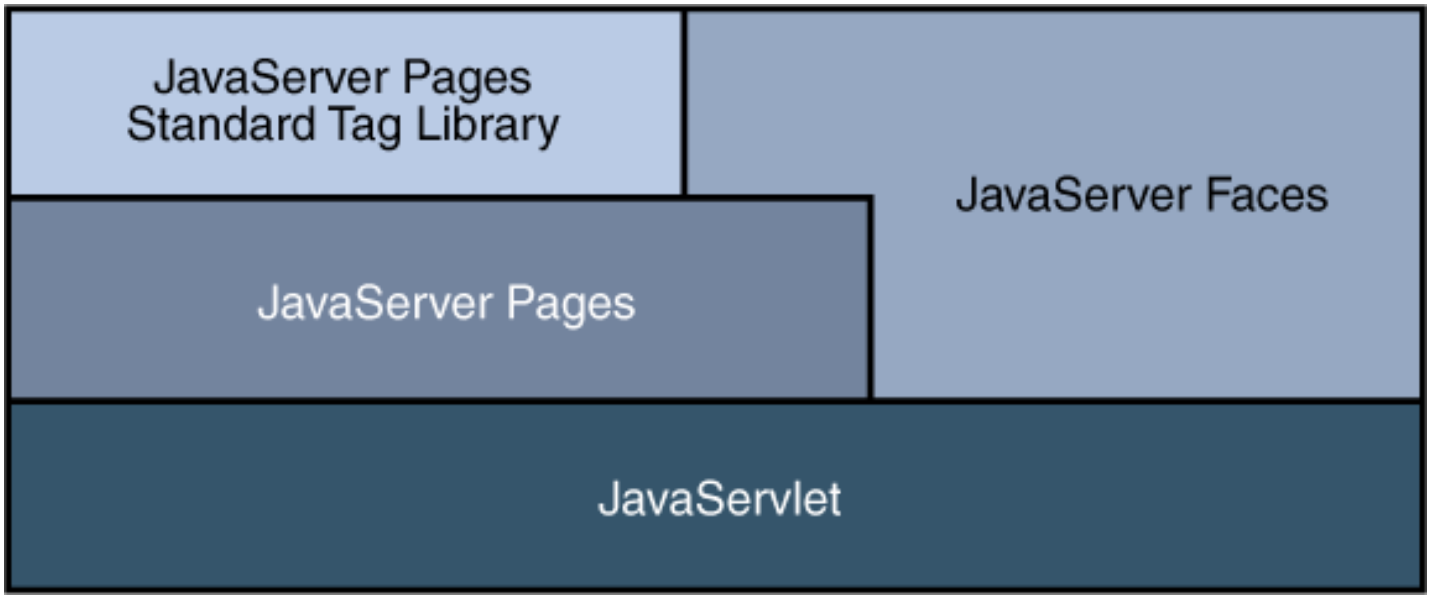


Web Tier Application Architecture



Source: http://java.sun.com/javaee/5/docs/tutorial/doc/

Web Tier Technologies



Source: http://java.sun.com/javaee/5/docs/tutorial/doc/

A Servlet's Job:

- Read explicit data sent by client (from data)
- Read implicit data sent by client (attributes in request header)
- Generate the results (business logics)
- Send the explicit data back to client (HTML)
- Send the implicit data to client (attributes in response headers)

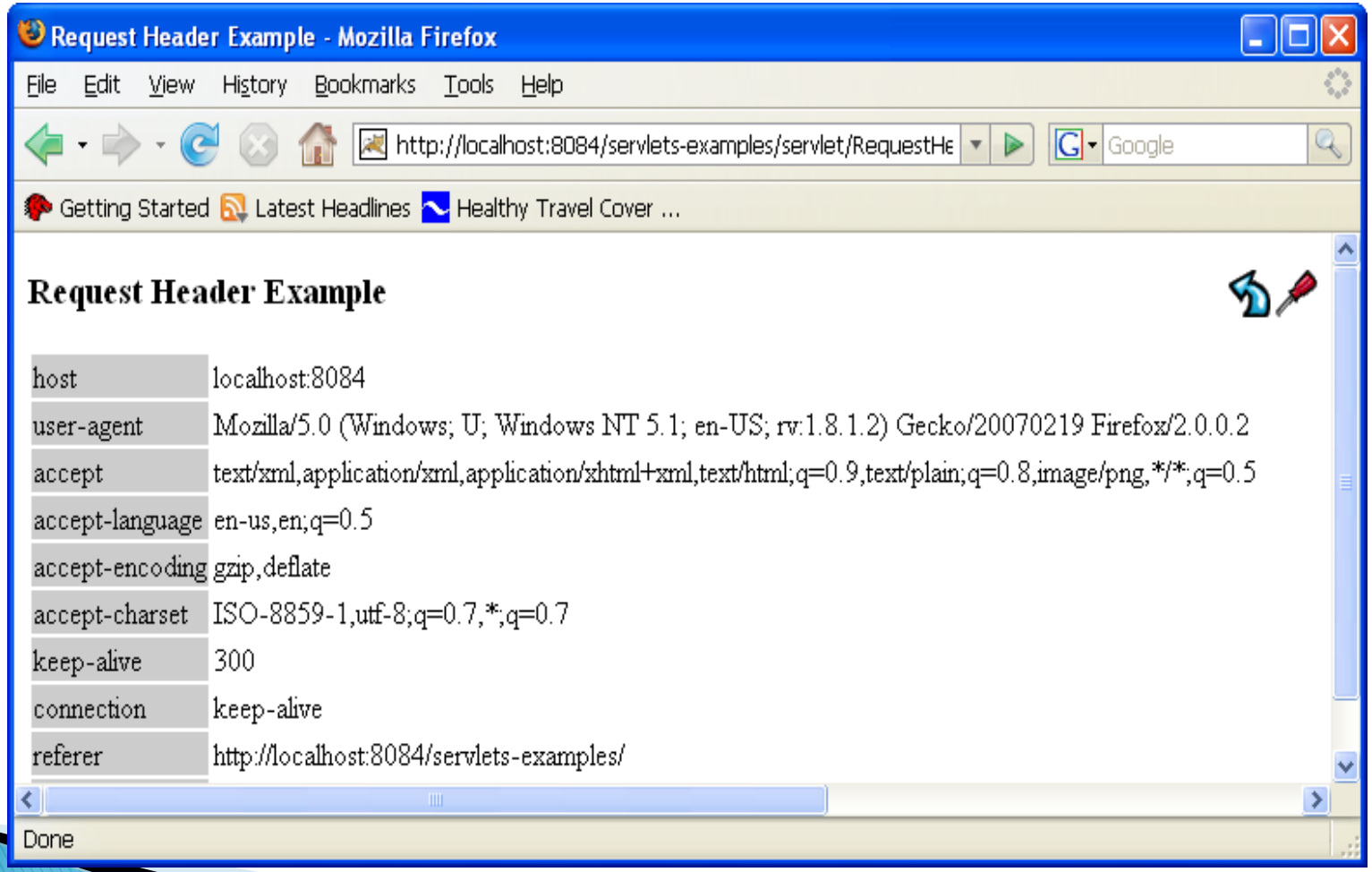
Source: http://courses.coreservlets.com/Course-Materials/csajsp2.html

A Servlet Example

```
public class RequestHeaderExample extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse
    response) throws IOException, ServletException
    {
        Enumeration e = request.getHeaderNames(); //  get data from
    requets

        //  do whatever you like (Business logics)

        response.setContentType("text/html");                    //  put data to
    response
        PrintWriter out = response.getWriter();

        while (e.hasMoreElements())
        {
            String name = (String) e.nextElement();
            out.println ( name + " = " + request.getHeader(name) );
        }
    }
}
```

Source: http://tomcat.apache.org

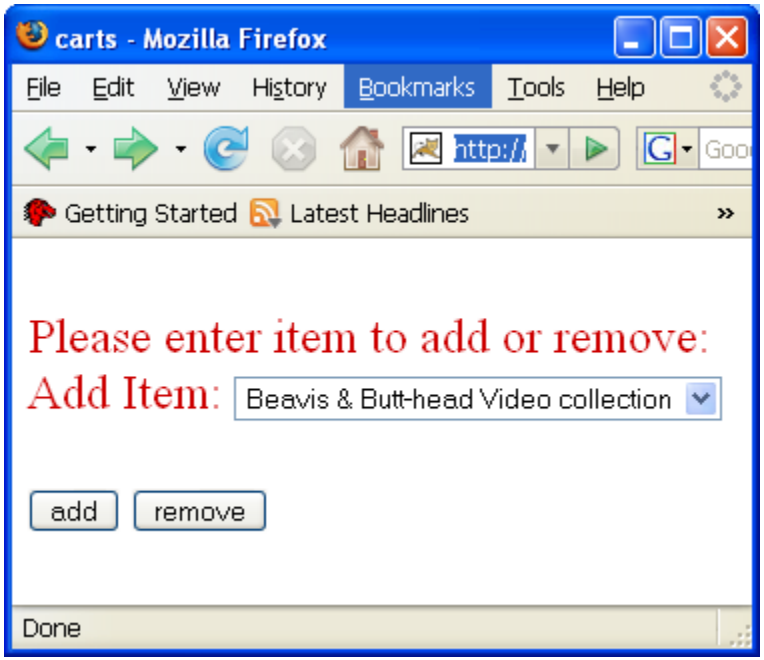
The results of The Example



Why JSP (JavaServer Pages)?

- With servlets, you can:
  - Have a fully control of a request and its response
- But, it is a pain to:
  - Use those 'println' to 'manually' generate HTML
  - Maintain that HTML
- JSP: idea
  - Use regular HTML for static (most) parts of a page
  - Mark dynamic content with special tags and/or Java code

A JSP Example of Shopping Cart




```
<html>
    <head><title>carts</title></head>
    <body bgcolor="white">
    <font size = 5 color="#CC0000">
    <form type=POST action=carts.jsp>

    <br> Please enter item to add or remove:
    <br> Add Item:

    <SELECT NAME="item">
    <OPTION>Beavis & Butt-head Video collection
    <OPTION>X-files movie
    <OPTION>Twin peaks tapes
    <OPTION>NIN CD
    <OPTION>JSP Book
    <OPTION>Concert tickets
    <OPTION>Love life
    <OPTION>Switch blade
    <OPTION>Rex, Rugs & Rock n' Roll
    </SELECT>
    <br> <br>

    <INPUT TYPE=submit name="submit“ value="add">
    <INPUT TYPE=submit name="submit" value="remove">
    </form>
</FONT>
</body>
</html>
```

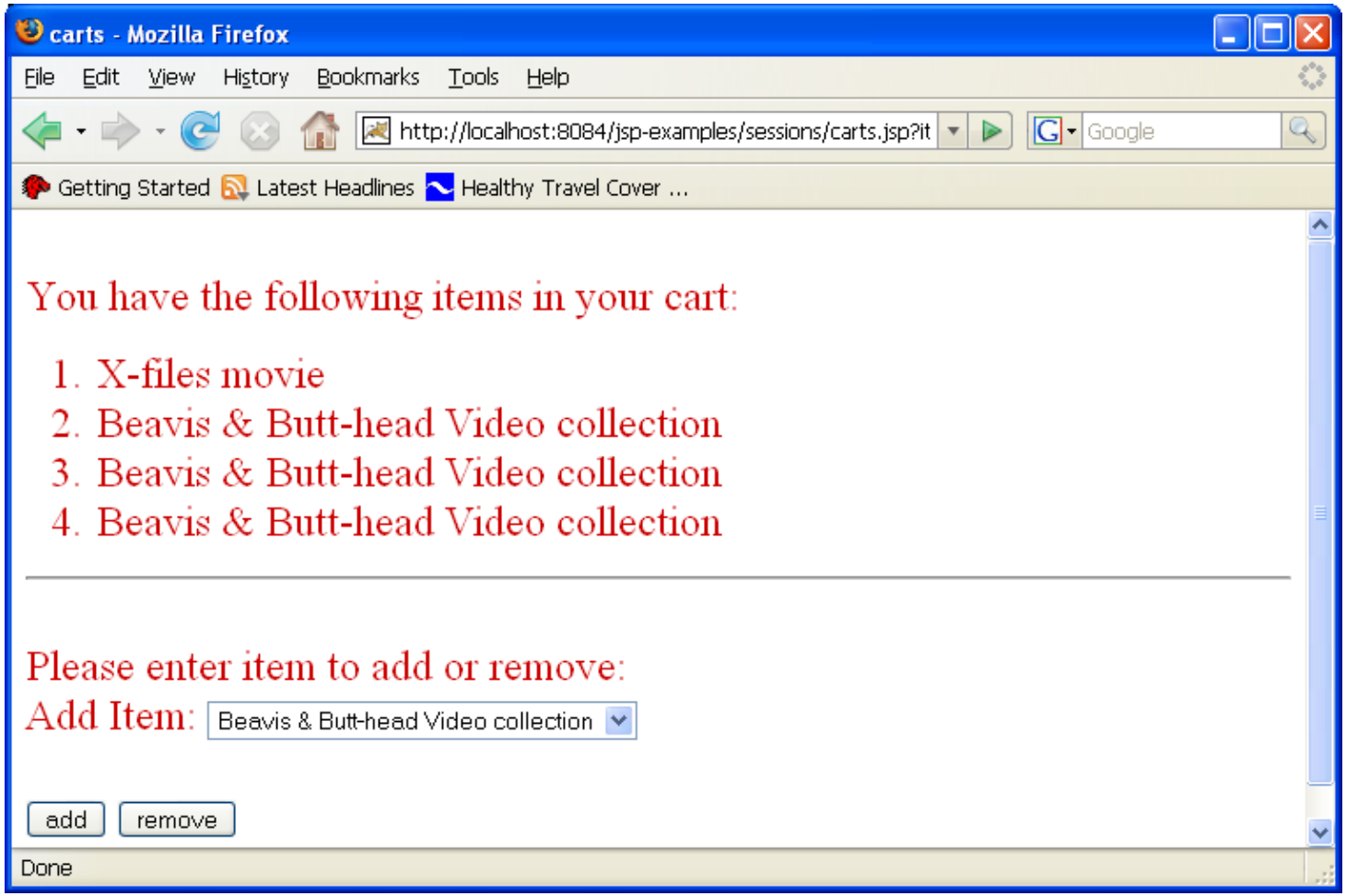
Carts.jsp

```
<html>
   <jsp:useBean id="cart" scope="session" class="sessions.DummyCart" />
   <jsp:setProperty name="cart" property="*" />
   <%
         cart.processRequest(request);
   %>
   <FONT size = 5 COLOR="#CC0000">
   <br> You have the following items in your cart:
   <ol>
<%
         String[] items = cart.getItems();
         for (int i=0; i<items.length; i++)
         {
%>
         <li> <% out.print(util.HTMLFilter.filter(items[i])); %>
<%
         }
%>
   </ol>
   </FONT>
   <hr> <%@ include file ="/sessions/carts.html" %>
</html>
```

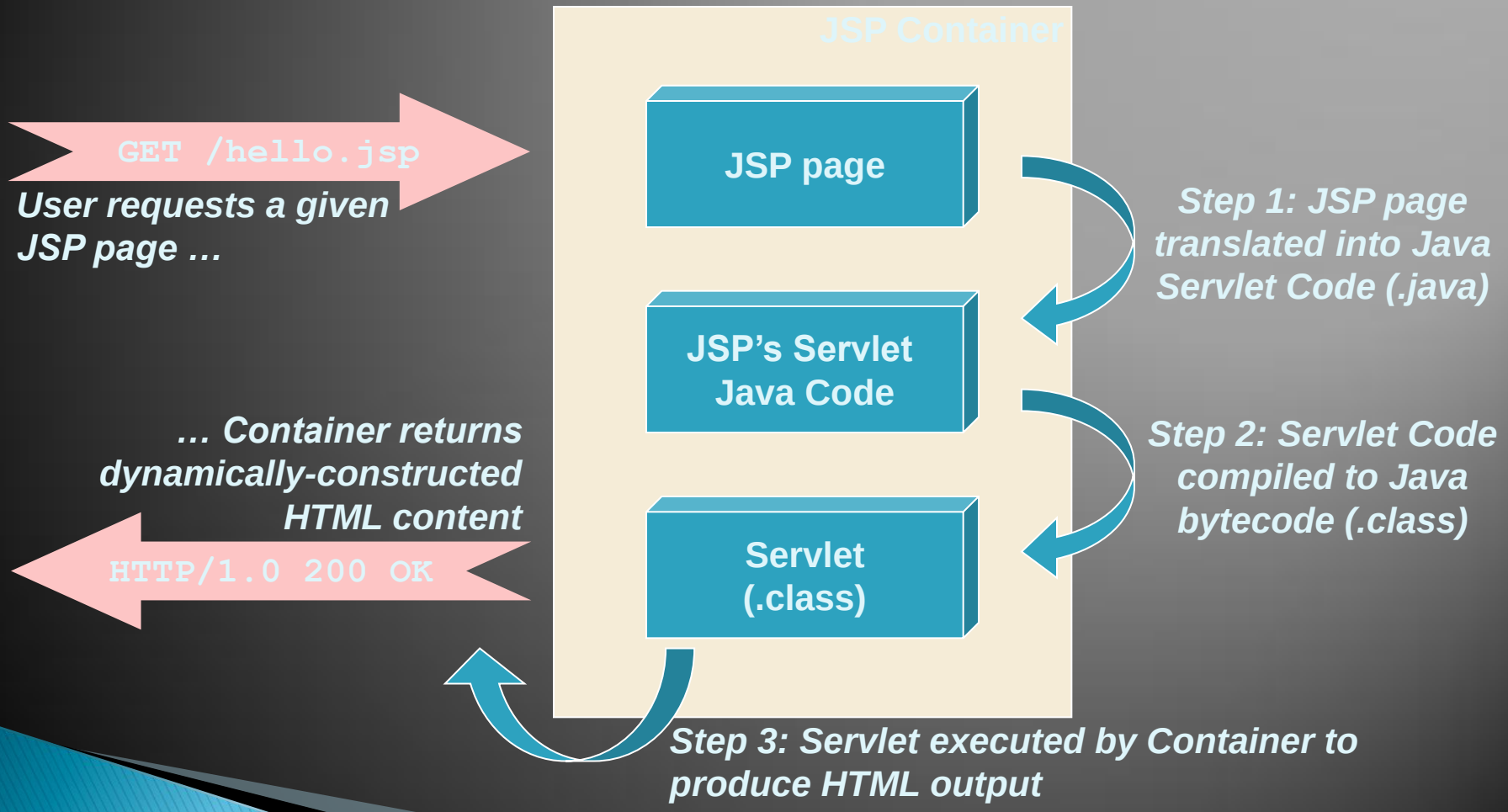
DummyCart.java

```
public class DummyCart
{
        Vector v = new Vector();
        String submit = null;
        String item = null;
    public void setSubmit(String s) { submit = s; }
    public void setItem(String i) { item = i); }
    public String[] getItems()
    {
        String[] s = new String[v.size()];
        v.copyInto(s);
        return s;
    }
    public void processRequest(HttpServletRequest request)
    {
        if (submit.equals("add"))
            addItem(item);
        else if (submit.equals("remove"))
            removeItem(item);
    }
}
```

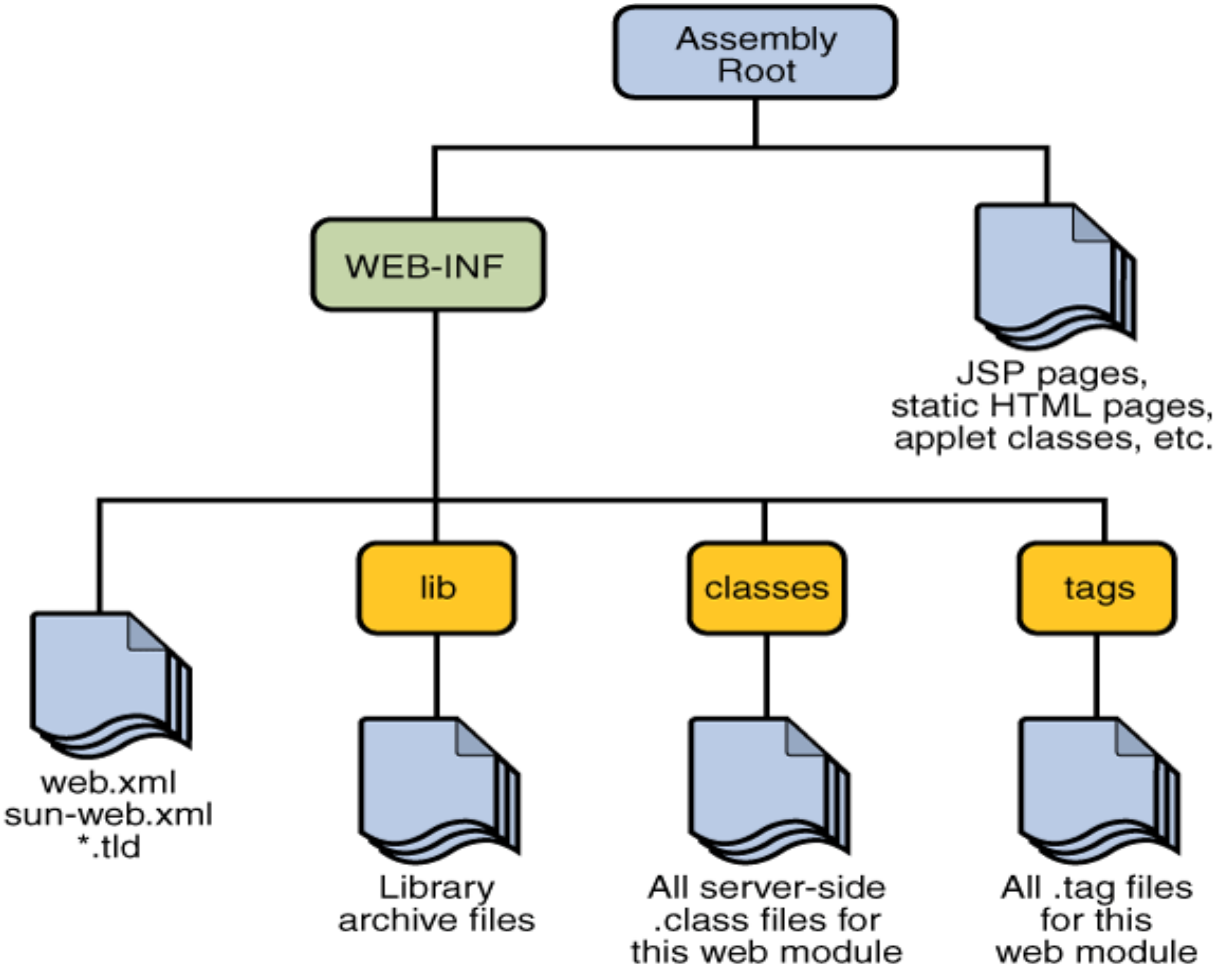
The Outcome of The JSP



JSP Request / Response Lifecycle

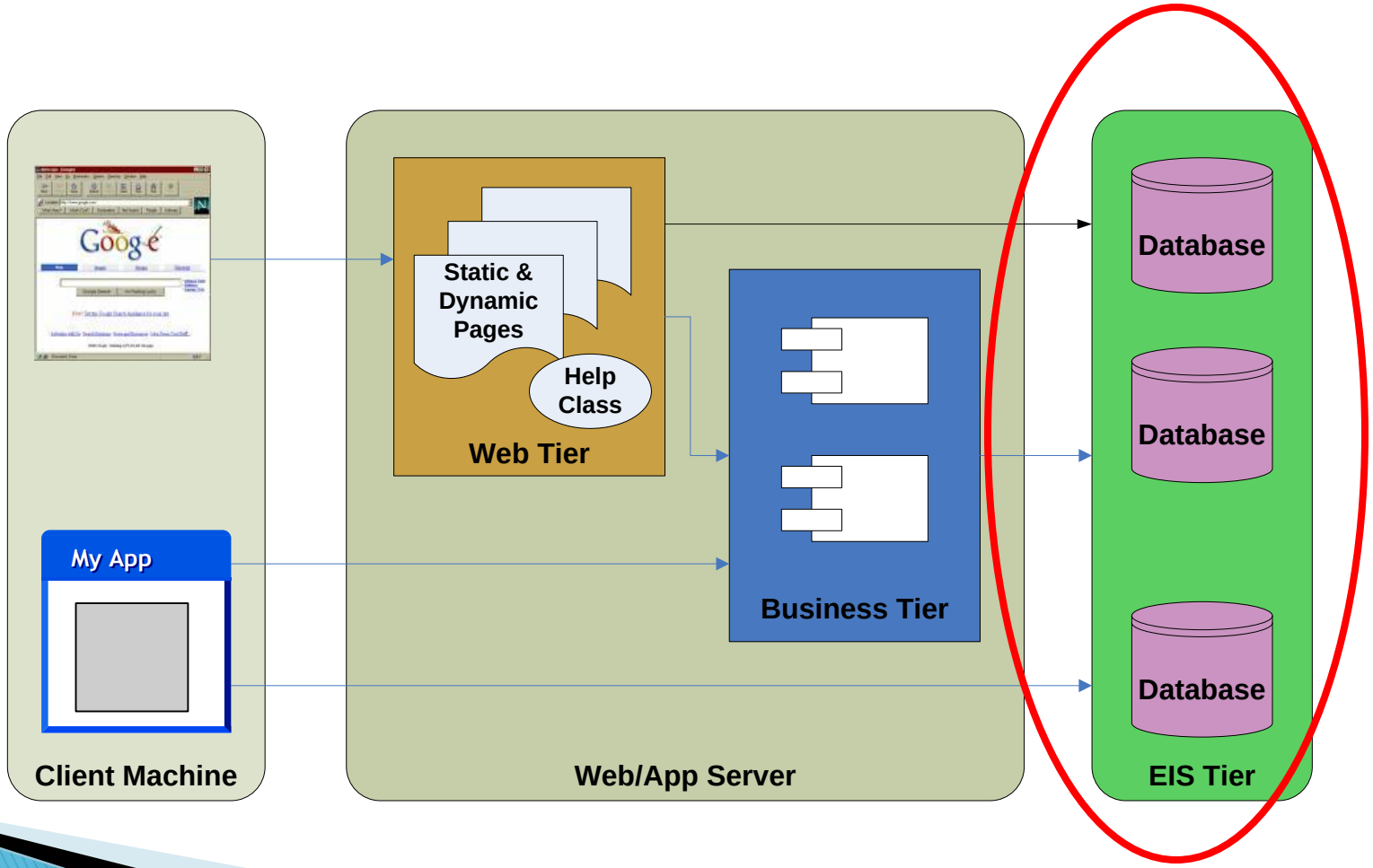


J2EE Web Application Deployment



Source: http://java.sun.com/javaee/5/docs/tutorial/doc/

Let us Talk about EIS Tier

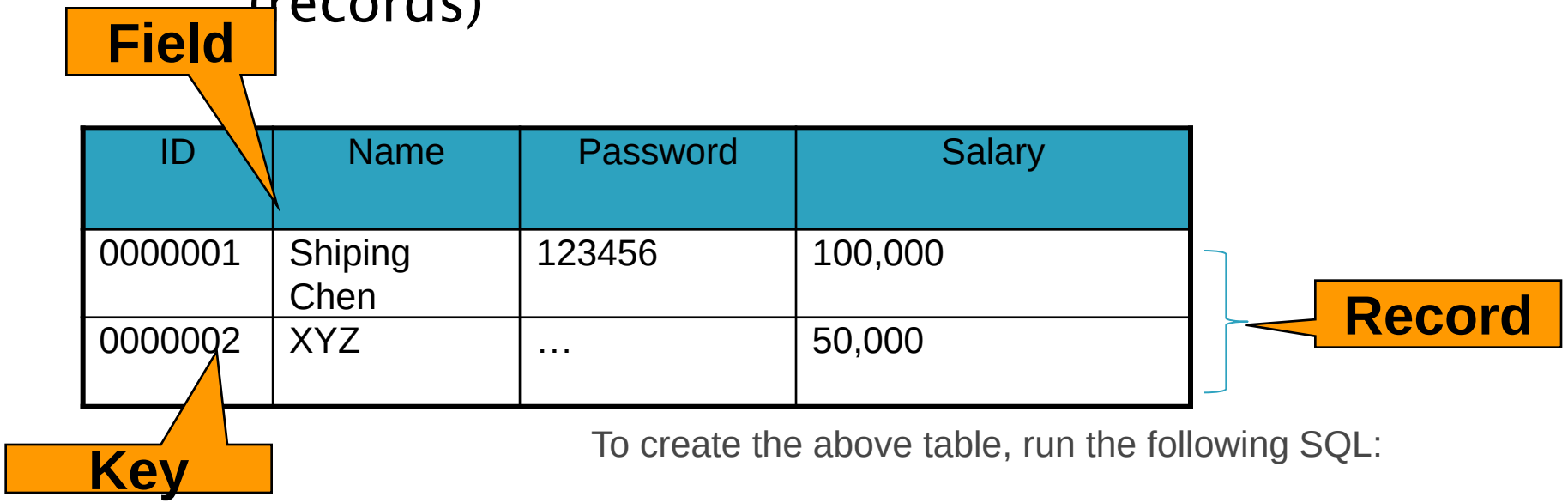


Database Basic

- What is Database (Database Management System)?
  - A **Database** is an integrated collection of logically related records or files that is stored in a computer system which consolidates records previously stored in separate files into a common pool of data records that provides data for many applications.
  - A **Database** is a collection of information that is organized so that it can easily be accessed, managed, and updated.

Database Basic

- Table: is the key component to host data (records)

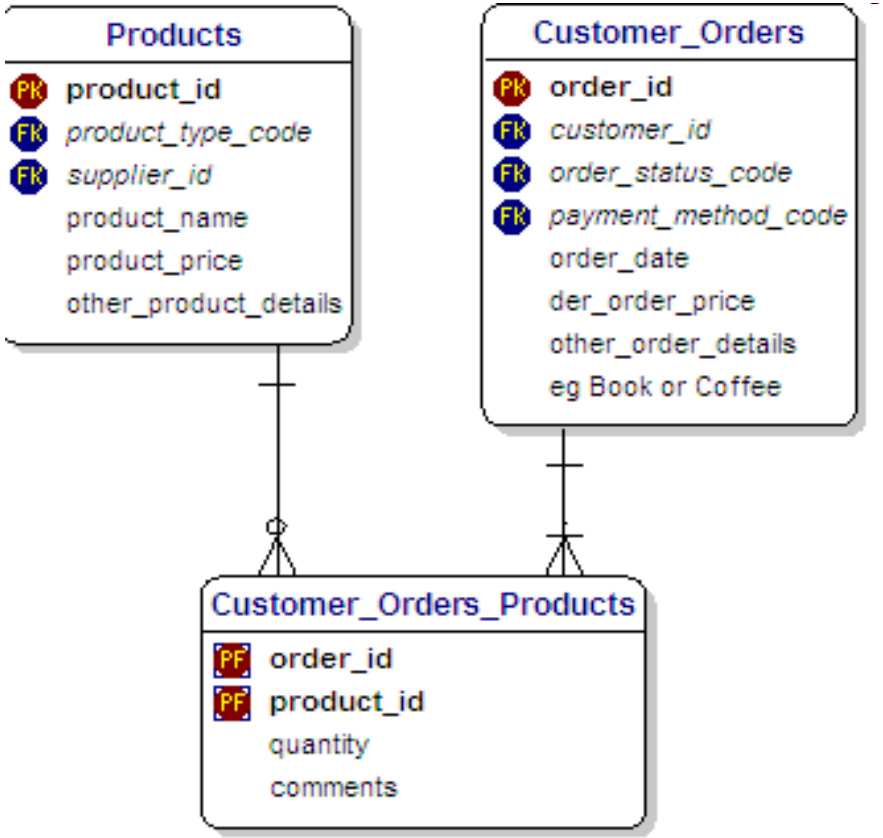


To create the above table, run the following SQL:

```
CREATE TABLE Employee (
      ID              INTEGER PRIMARY KEY,
      Name            VARCHAR(20) NOT NULL,
      Password        VARCHAR (10) NOT NULL,
      Salary          NUMBER(12,2)
);
```

Databse Basic

- One table may have relations/link to other tables via keys:



- Primary Key (PK)
- Foreign Key (FK)

Database Basic

- Usually, we talk to database in SQL to ask questions, and/or update the data, such as:
  - Select * from Employee where salary > 80,000;
- The common operations in SQL:
  - Create table …
  - Insert into …
  - Select … from …where
  - Update …. Where …
- Learn more: http://www.w3schools.com/SQl/default.asp

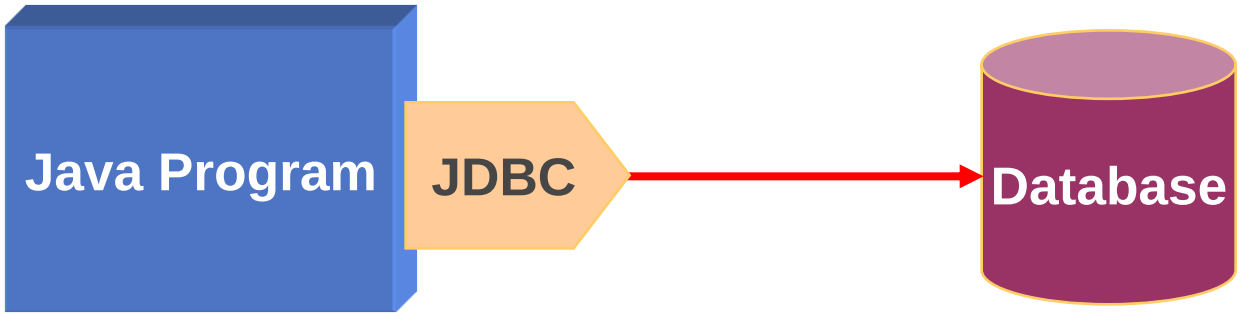
JDBC: Java Database Connectivity

- JDBC Basics
  - What is JDBC?
  - Preparations for accessing database via JDBC
  - Basic steps of JDBC programming
  - An example with demo

- Advanced Topics
  - Types of JDBC
  - Prepare statements
  - Connection pool
  - Transaction support

What is JDBC?

- **JDBC** is Java application programming interface that allows the Java programmers to access database from Java code to conduct SQL-related operations.



Which can be: JavaBean, EJB, Servlet, or plain Java applications

Which can be: Oracle, DB2, SQL Server etc.

Preparation for Accessing Database via JDBC

- Install and/or make it sure that your database is running and you have an database account to access the database
  - Usually check with admin and/or query tools
  - Usually create tables and add a few records to the table with the tool
- Install and/or make it sure that your JDBC driver jar file is accessible for your Java applications/JavaBean/EJB.
  - Meaning 'unzip the zip file' for Type 4 JDBC driver
  - Add the jar file to your classpath, or bound the database/JDBC to your EJBs as a datasource (How?)

Basic Steps of JDBC Programming 1/3

- Establish Connection to Database
  - Either explicitly/directly 'instance' a new connection with DriverManager
    - Class.forName("com.mysql.jdbc.Driver");
    - conn = DriverManager.getConnection(url, user, password);
  - Or indirectly 'get' a connection to a datasource from a database connection pool:
    - InitialContext ic = new InitialContext();
    - DataSource ds = ic.lookup("java:comp/env/jdbc/myDB");
    - Connection con = ds.getConnection();

Basic Steps of JDBC Programming 2/3

- SQL Operations:
  - Instance a sql statement object:
    - *String sql = "INSERT INTO customer(ID, NAME) VALUES (?, ?)";*
    - *PreparedStatement pstm = con.prepareStatement(sql);*
  - Bind variables to '?' in 'sql'
    - *pstm.setString(1,"000001");*
    - *pstm.setString(2,"Shiping");*
  - Execute the sql
    - *pstm.executeUpdate();*
- Release resources:
  - *pstm.close();*
  - *con.close(); // for pooled con, it is not closed, but back to the pool*

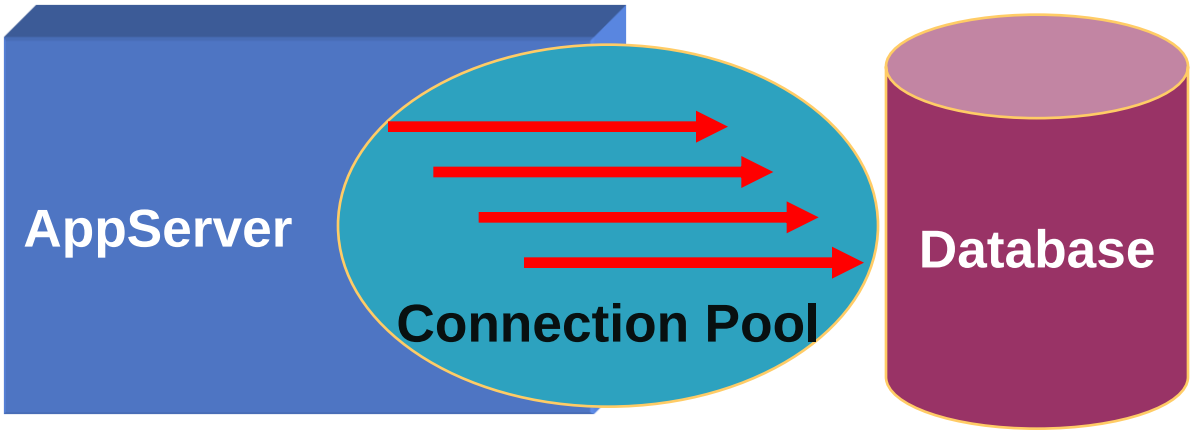
Basic Steps of JDBC Programming 3/3

- Another example for SQL Query in JDBC

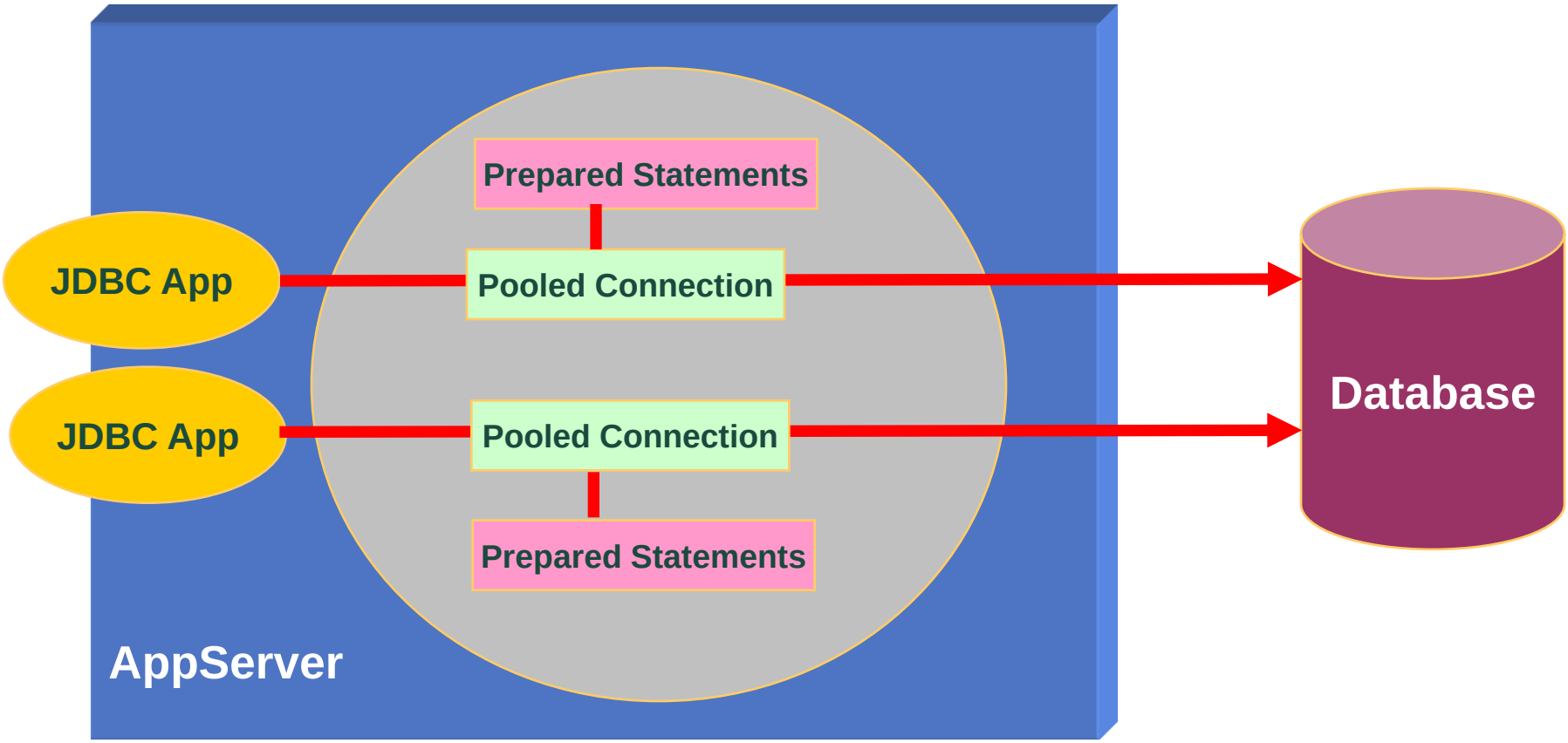
```
import java.sql.*;

public void findCustomer(String id)
{
        try
       {
              String sql =  "SELECT Name FROM Customer WHERE ID = '" + id  + "'";
              Statement stm = con.createStatement(); // suppose con is instanced !
              ResultSet rs = stm.executeQuery(sql);
              while (rs.next())
              {
                                  System.out.println("Name = " + rs.getString(1))
              }
              stm.close();
              con.close();
         }
        catch (SQLException ex) { ex.printStackTrace(); }
}
```

Connection Pool

- It is very expensive to establish/close a physical connection to database comparing to normal SQL operations.
- To improve performance, instead of dropping connections, a set of connections can be cached in a pool for reuse at runtime.



Pooled Connection vs. Prepared Statement



Source: http://java.sun.com/products/jdbc/download.html

JDBC Advices on Performance

- Try to get a connection from a datasource that supports connection pool
- Call *'con = ds.getConnection()'* only when you are about to use it
- Try to use '*PreparedStatement pst = '*
- Don't forget to call: '*pst.close()*' & '*con.close()*' as soon as possible

Transactions with JDBC

- Auto-commit Model
  - Commit as soon as each SQL execution
  - Default setting for most database systems
  - Can be disenabled by programmers or application servers
- Explicitly commit or rollback
  - Disenable auto-commit before a transaction
    - con.setAutoCommit(false);
  - If everything OK, con.commit();
  - If something wrong, con.rollback();
- Some JDBC also support distributed transactions via 2PC.

Details refer to JDBC Specification v4.0 & v3.0 http://java.sun.com/products/jdbc/download.html

Web Services

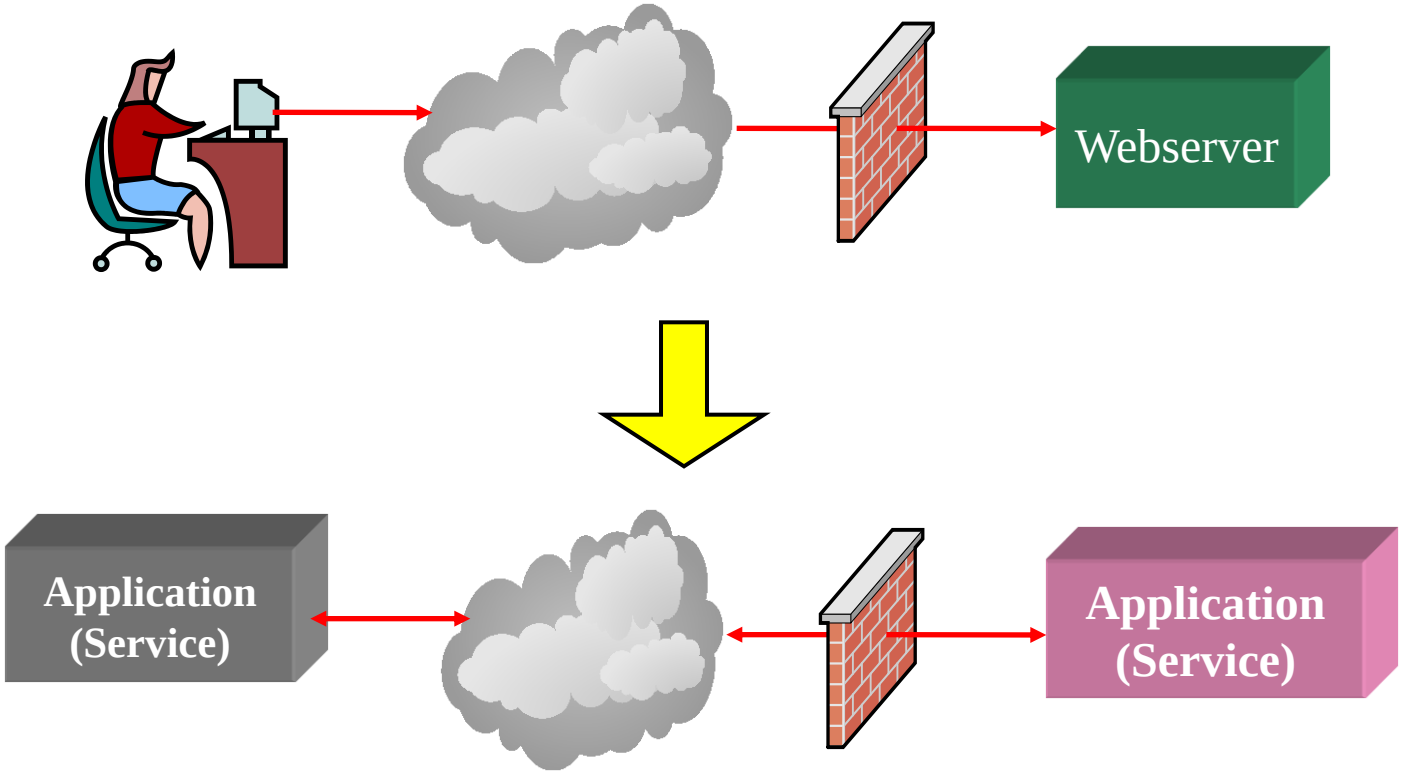
Motivation: Why Web Services? 1/2

- Let us have a look at the history of distributed computing/middleware:
  - Socket Programming
  - RPC: Remote Procedure Call
  - CORBA: Common Object Request Broker Architecture
  - EJB & COM+: Component-Based
- As making progress, but…
  - They all use binary-based protocols, such as IIOP, RMI, IIOP-over-RMI, DCOM
  - So they are product/languages/OS-dependent

Motivation: Why Web Services? 2/2

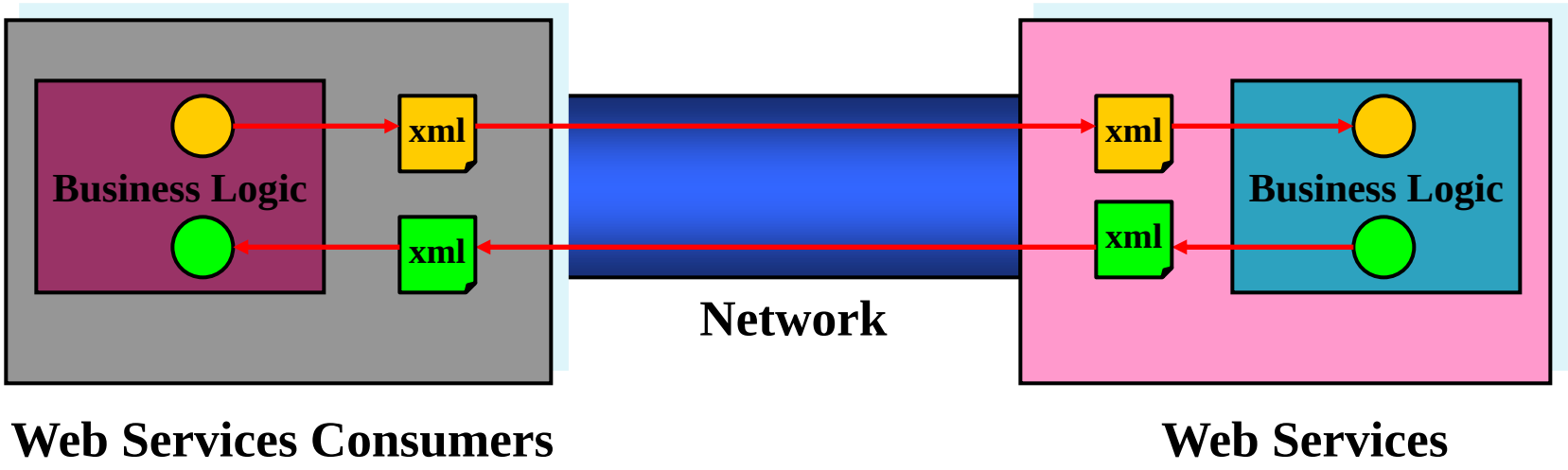
- EAI/EII Requirements:
  - Within enterprise: CRM/ERP, Just-in-Manufacture, remote-design/manufacture, etc
  - Across enterprises: supply-chains, retail brokers, banking, etc.
- XDI – Enterprise Data Interchange
  - EDI – Electronic Data Interchange http://www.1edisource.com/
  - FIX – Financial Information eXchange http://www.fixprotocol.org/cgi-bin/Welcome.cgi
  - More things are coming: XML/EDI, ebXML, etc.
- So we need a universal integration technology - Web Services!

What is Web Services?



Short Answer : A set of *technologies/standards* that enables *any* applications/services talk each other via *Web*.

How Web Services works?

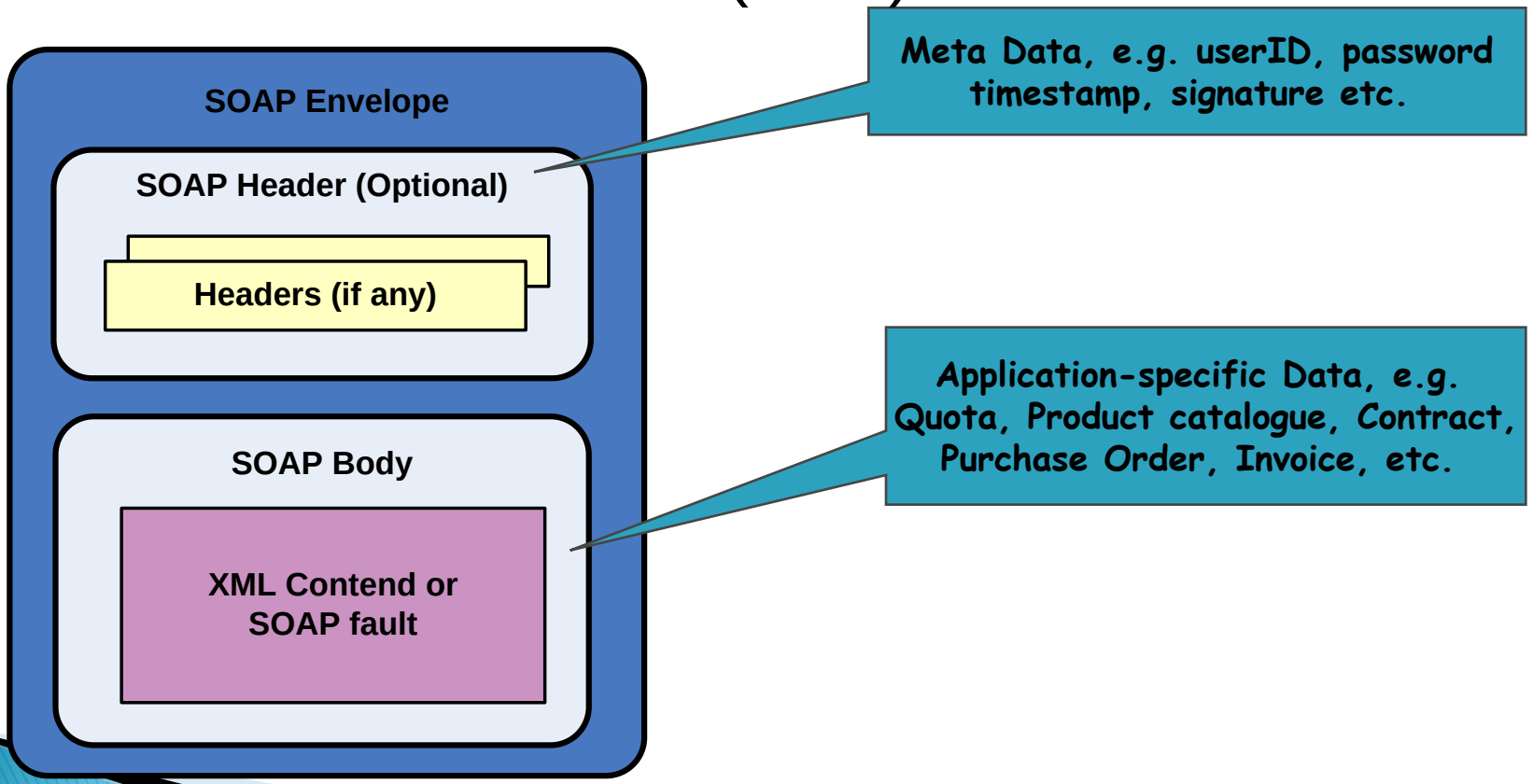


Core Technologies of Web Services

- SOAP – Simple Object Access Protocol
  - How to format the XML messages
- WSDL – Web Services Description Language
  - How to describe web services
- UDDI – Universal Description Discovery Integration
  - How to register and find web services

Structure of SOAP Messages 1/2

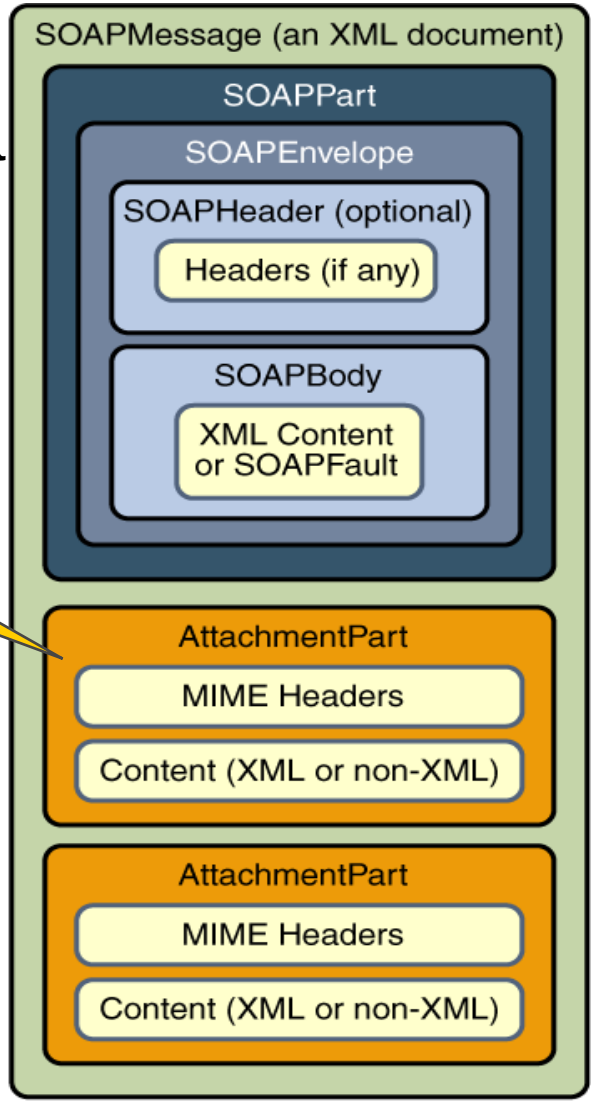
- A Plain (99%) SOAP



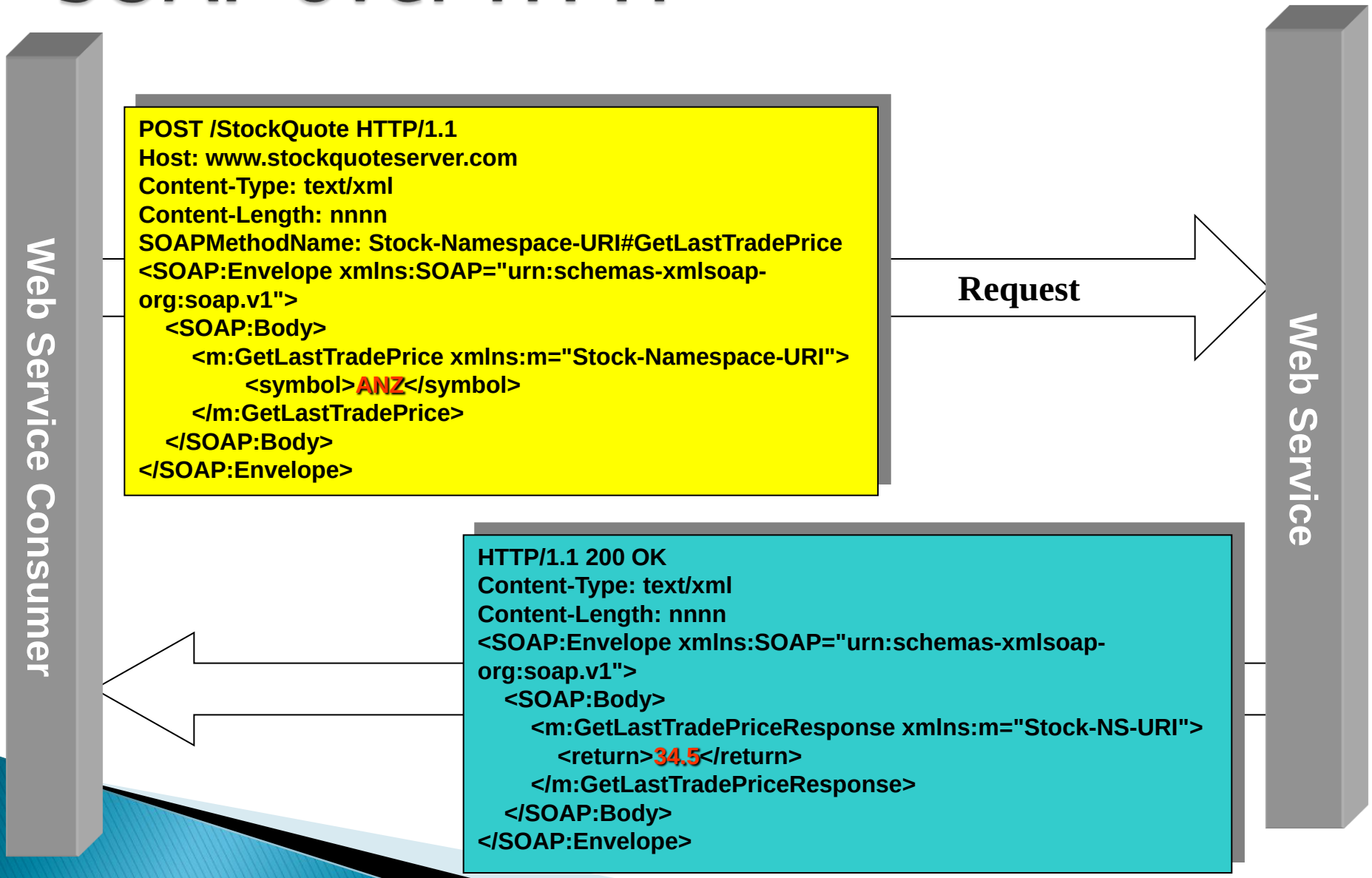
Structure of SOAP Messages 2/2

- SOAP with Atta

A SOAP message can contain multiple non-XML data blocks as attachments, such as image, music etc.



SOAP over HTTP



Summary of SOAP

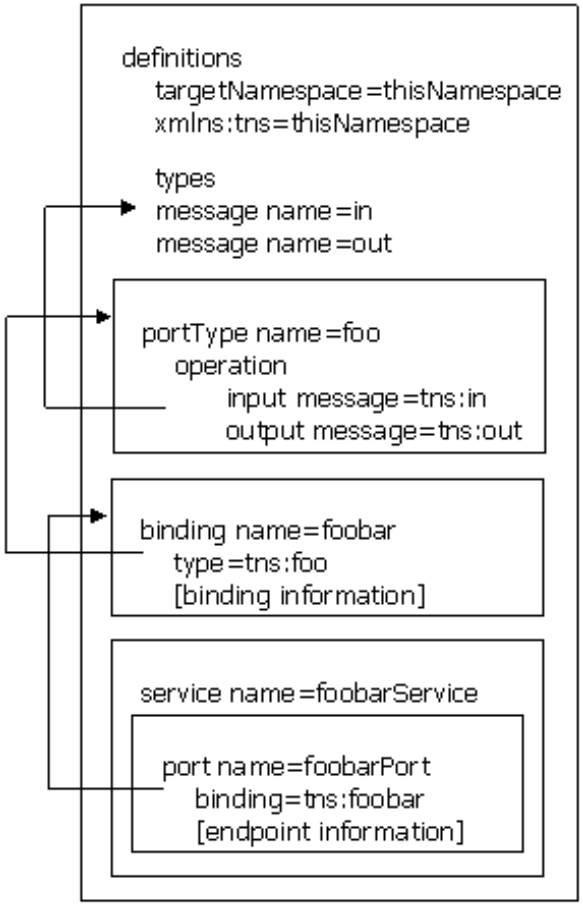
- SOAP define a overall format of XML message exchanged between applications/services
- SOAP is extendable via its Header, i.e. meta data
- SOAP is transport-independent, i.e. SOAP ≠ HTTP
- SOAP is changing (currently v1.2):
  - From v1.2, SOAP is no longer an acronym (the original acronym has lost its meaning; someone suggest 'Service Oriented Architecture Protocol')
  - SOAP v1.2 supports multiple bindings, multiple headers and multiple-reference values etc.
  - Moving from RPC-style to document-style and XML infoset
  - Details refer to:
- But SOAP itself has no mechanism for security!

Why WSDL?

– Web Services Description Language

- SOAP only define the generic format of SOAP Messages
- To access a specific web service, an application need to know:
  - What data (also called XML schema or data type) in the SOAP messages
  - What Services/operations provided
  - What transport protocol used
  - Where it is: How to address the end port
- WSDL is designed to answer the above questions

WSDL (2.0.2) Data Model



types contains data type definitions
messages consist of one or more parts

portType describes an abstract set
of operations

binding describes a concrete set of
formats and protocols for the foo
portType

port describes an implementation
of the foobar binding

Source: http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v202-20040631_files/image002.gif

Types

- XML structures that are exchanged are discribed using XML Schema
- Types in WSDL define ALL new ‘complexType’ for the WSDL description

```
<wsdl:types>
    <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema" targetNamespace="urn:GoogleSearch">
        <xsd:complexType name="GoogleSearchResult">
            <xsd:all>
                <xsd:element name="documentFiltering" type="xsd:boolean"/>
                <xsd:element name="searchComments" type="xsd:string"/>
                <xsd:element name="estimatedTotalResultsCount" type="xsd:int"/>
                <xsd:element name="estimateIsExact" type="xsd:boolean"/>
                <xsd:element name="resultElements" type="typens:ResultElementArray"/>
                <xsd:element name="searchQuery" type="xsd:string"/>
                <xsd:element name="startIndex" type="xsd:int"/>
                <xsd:element name="endIndex" type="xsd:int"/>
                <xsd:element name="searchTips" type="xsd:string"/>
                <xsd:element name="directoryCategories" type="typens:DirectoryCategoryArray"/>
                <xsd:element name="searchTime" type="xsd:double"/>
            </xsd:all>
        </xsd:complexType>
    </xsd:schema>
</wsdl:types>
```

The slices about WSDL sourcing: http://dret.net/lectures/services-fall06/webservices1#(15)

Messages

- Messages describe the SOAP body parts
- Messages can have one or multiple parts as different logic units

```
<message name="doGoogleSearch">
        <part name="key" type="xsd:string"/>
        <part name="q" type="xsd:string"/>
        <part name="start" type="xsd:int"/>
        <part name="maxResults" type="xsd:int"/>
        <part name="filter" type="xsd:boolean"/>
        <part name="restrict" type="xsd:string"/>
        <part name="safeSearch" type="xsd:boolean"/>
        <part name="lr" type="xsd:string"/>
        <part name="ie" type="xsd:string"/>
        <part name="oe" type="xsd:string"/>
</message>
```

Port Types

- Port Types are named sets of abstract operations
  - The operations are described by the messages they are based on
- WSDL supports 4 kinds of message exchange patterns:
  - ***One-way:*** *The endpoint receives a message*
  - ***Request-response:*** *the endpoint receives a message and sends a correlated message (response)*
  - ***Solicit-response:*** *the endpoint sends a message and receives a correlated message*
  - ***Notification:*** *the endpoint sends a message*

```
<portType name="GoogleSearchPort">
        <operation name="doGetCachedPage">
                <input message="typens:doGetCachedPage"/>
                <output message="typens:doGetCachedPageResponse"/>
        </operation>
        <operation name="doSpellingSuggestion">
                <input message="typens:doSpellingSuggestion"/>
                <output message="typens:doSpellingSuggestionResponse"/>
        </operation>
        <operation name="doGoogleSearch">
                <input message="typens:doGoogleSearch"/>
                <output message="typens:doGoogleSearchResponse"/>
        </operation>
</portType>
```

Bindings

- Bindings defines message format and protocol details for a port type
  - the type attribute identifies the associated port type
  - there can be any number of bindings for a port type

```
<binding name="GoogleSearchBinding" type="typens:GoogleSearchPort">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="doGetCachedPage">
        <soap:operation soapAction="urn:GoogleSearchAction"/>
        <input>
            <soap:body use="encoded" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="…" />
        </input>
        <output>
            <soap:body use="encoded" encodingStyle=" http://schemas.xmlsoap.org/soap/encoding/" namespace="…" />
        </output>
    </operation>
</binding>
```

Services & Ports

- Services group related ports together
  - These ports must not communicate with each other
- Ports are service endpoints
  - A port specifies an address for a (physical) binding

```
<service name="GoogleSearchService">
    <port name="GoogleSearchPort" binding="typens:GoogleSearchBinding">
        <soap:address location="http://api.google.com/search/beta2"/>
    </port>
</service>
```

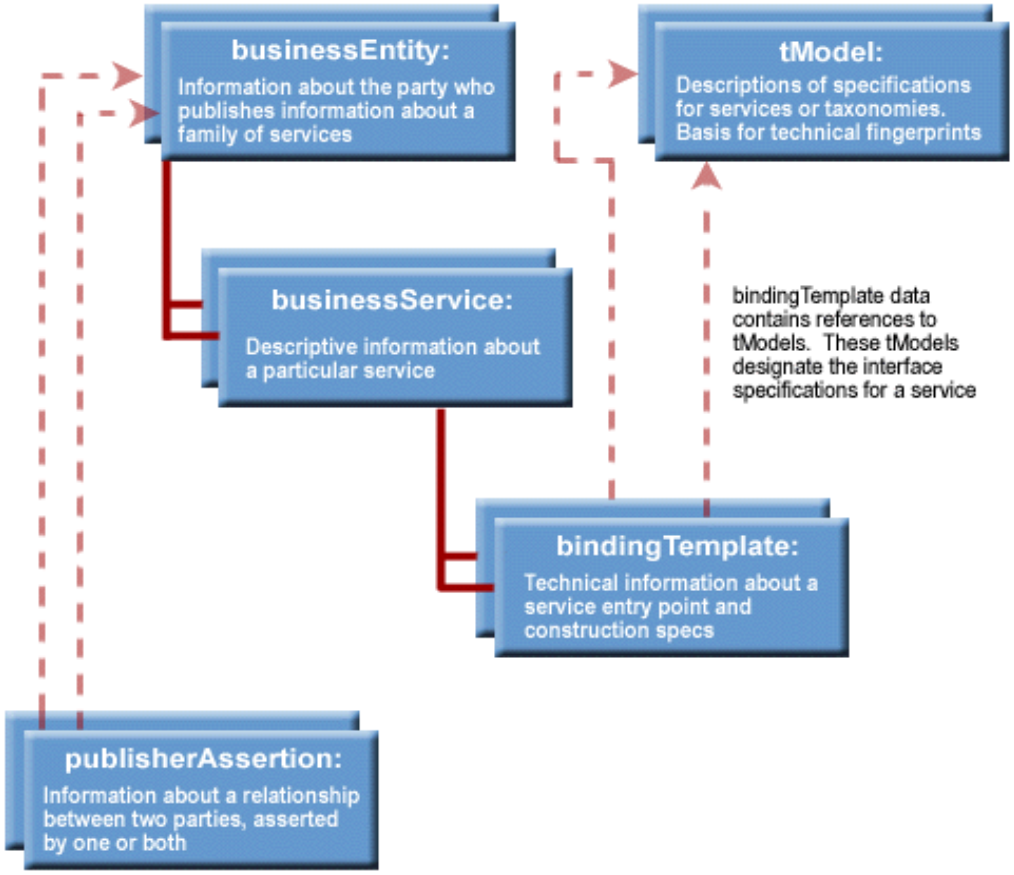
Summary of WSDL

- WSDL is XML-based language used to describe web services, including:
  - Types
  - PortTypes
  - Messages
  - Bindings
  - Services
  - Ports
- WSDL plays a very important role in developing web services applications (IDL) !
- Where/How to publish a WSDL?

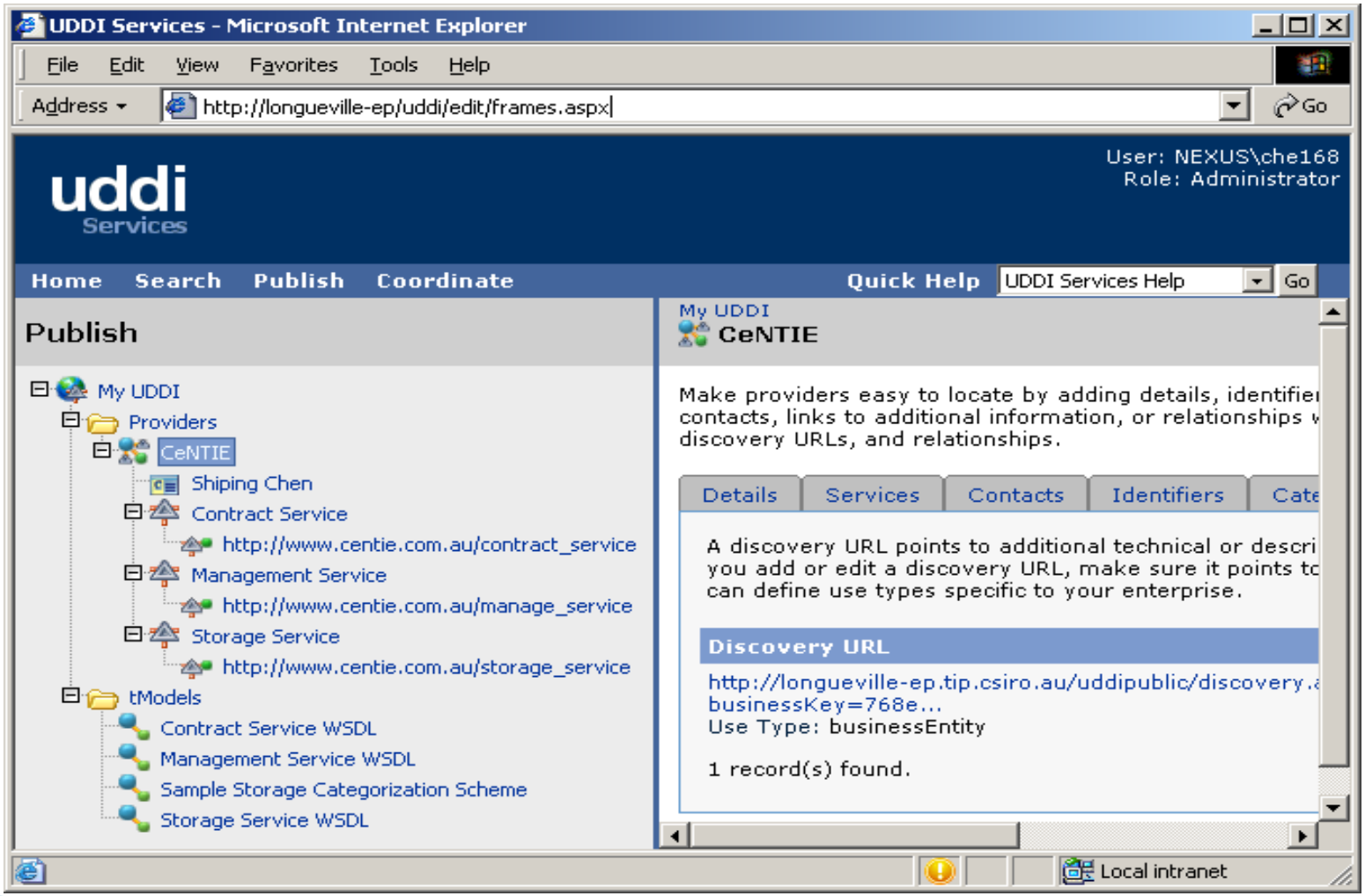
UDDI – Universal Description Discovery Integration

- Intended to be a repository for web services
- Modelled and operated as *yellow pages*
- Very generic and high-level data model
- Has not widely adopted in industry
  - Not as successful as SOAP and WSDL ☹

UDDI Data Model



Microsoft Windows Built-in UDDI



How UDDI Works?

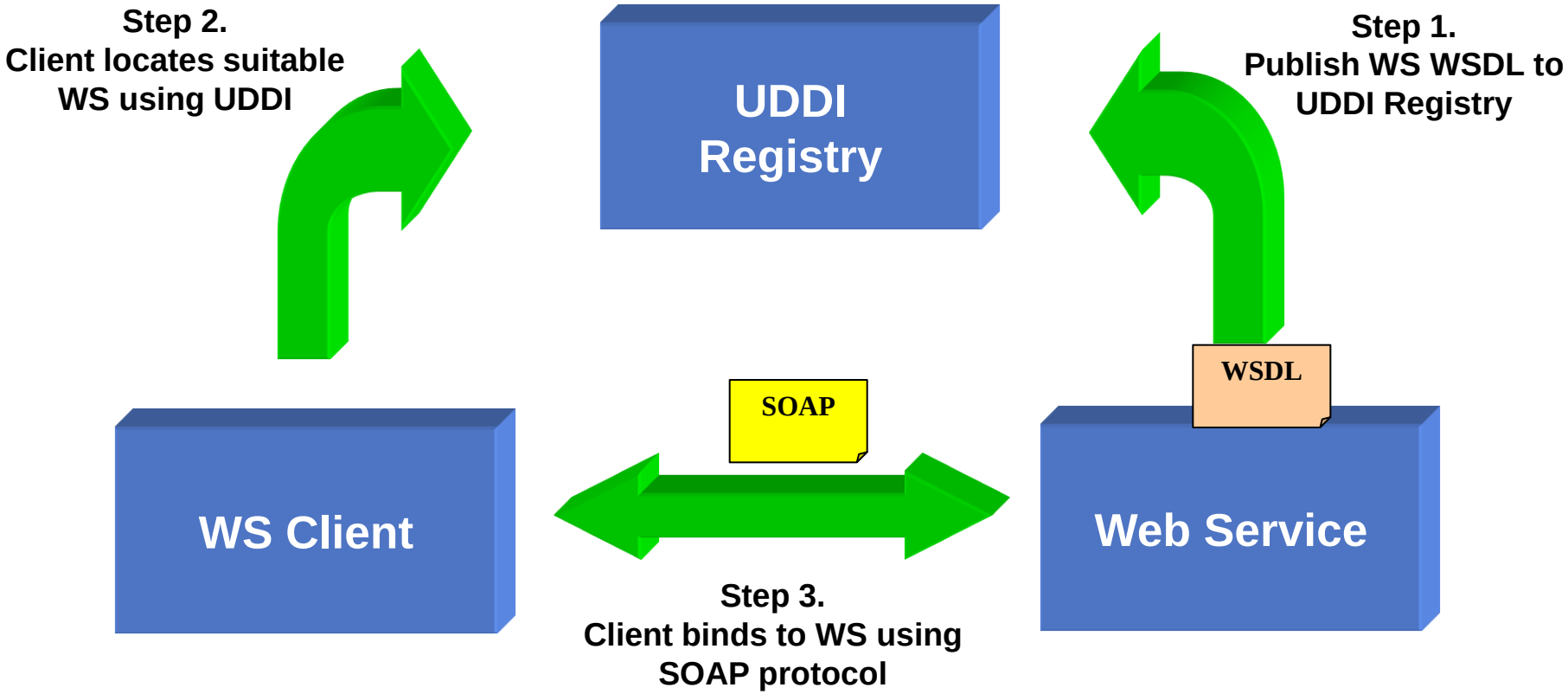


Figure 4.6 – Role of UDDI in Web Service Architecture

Major Web Services Players

- Vendors & Products:
  - MicroSoft ASP.NET
  - IBM WebSphere
  - BEA WebLogic (now is owned by HP)
  - Oracle/Sun Application Servers
- Open Sources:
  - Apache CFX
  - Apache Axis http://ws.apache.org/axis
  - JBoss, now owned by RedHat http://labs.jboss.com/

Tips/Steps for Developing Web Services

- Analysis information exchanged between applications/services
- Design interfaces/contracts between them, i.e. XML schema and operations
- Use primer data types as possible, such as string, int, rather platform-dependent types
- Use XML data binding tools to generate the class code
- Design and code the interface (web method)
- Try to get WSDL as early as possible
- Use WSDL tools to generate the client-side sub code
- Move the url from the code as a constructor's parameter
- Test your web services as early as possible

Tips/Steps for reusing web services

- Get the web service wsdl
- Generate the web service proxy class using the web service wsdl
- Call the web service using the generated client-sub clod

Demo/Exercise – build a client to call a real web service

- **Web Service:**
  - **China Stock Web Service**
- **To get its wsdl:**
  - **http://www.webxml.com.cn/WebServices/ChinaStockWebService.asmx?wsdl**
  - **Save as: StockWebService.wsdl**
- **To generate its proxy class using the downloaded wsdl, e.g. using the JavaEE6 utility:**
  - ***wsimport* StockWebService.wsdl *-Xnocompile -d src -p com.shiping.demo***

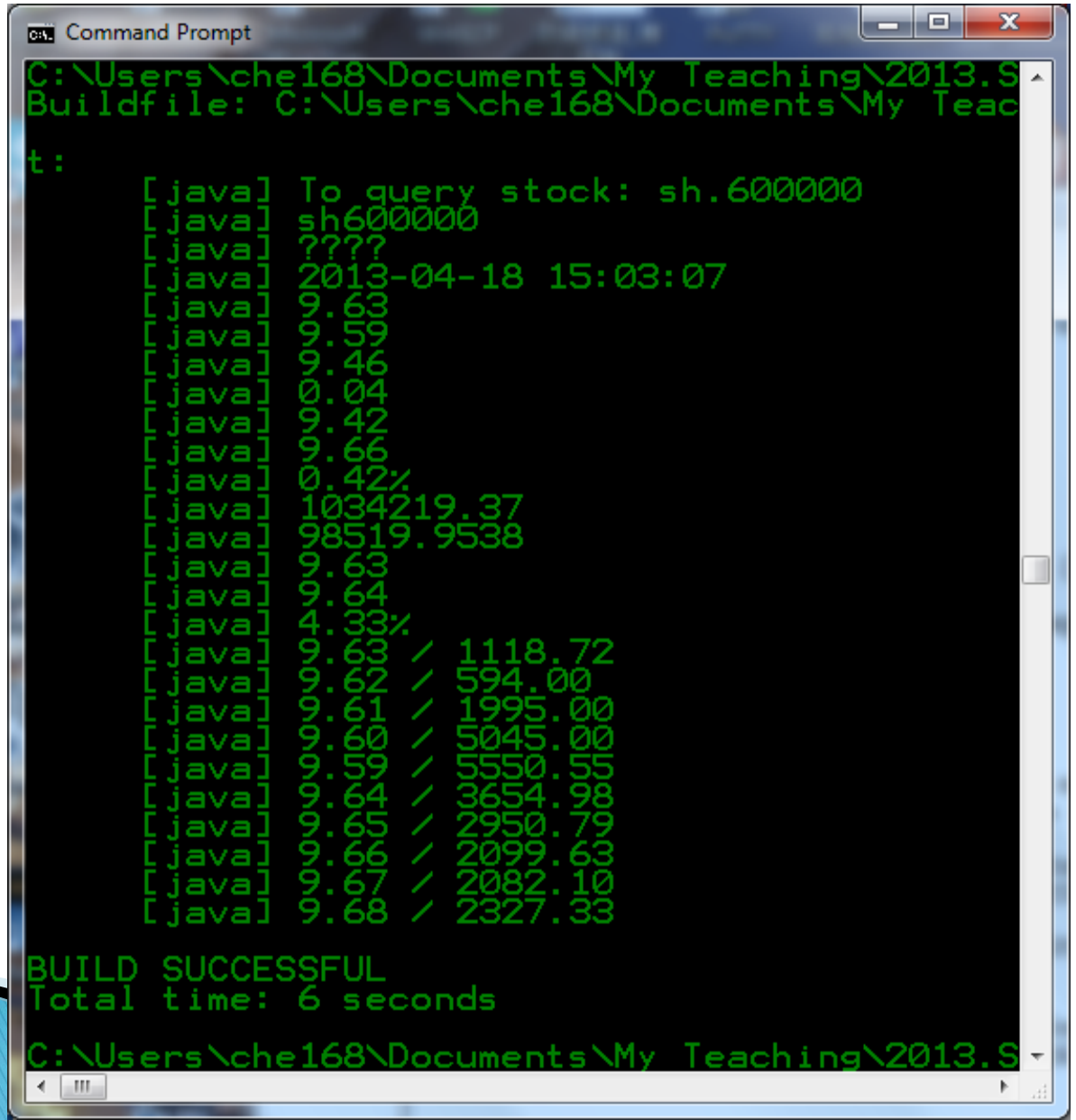
The Web Sevice Client in Java

Tester.java - Notepad

File Edit Format View Help

```
/*
 * @(#) Tester.java     1.0 17/04/2013
 *
 * Copyright (c) Dr. Shiping Chen. All rights reserved.
 *
 */

package com.shiping.demo;

import java.util.*;

class Tester {
        public static void main(String[] args) {
                if (args.length<1) {
                        System.out.println("Usage: java Tester stockCode");
                        System.out.println("       java Tester sh.600000");
                        System.exit(1);
                }
                String stockCode = args[0];

                ChinaStockWebService service = new ChinaStockWebService();
                ChinaStockWebServiceSoap port = service.getChinaStockWebServiceSoap();

                System.out.println("To query stock: " + stockCode);
                ArrayOfString result = port.getStockInfoByCode(stockCode);

                List<String> list = result.getString();
                Iterator it = list.iterator();
                while(it.hasNext()) {
                        String stockInf = (String) it.next();
                        System.out.println(stockInf);
                }
        }
}
```

Let us test it....



Task 3 of Assignment

- Now you can start to implement the stock broker system as designed in Task 2?

- Advices
  - Try to reuse the existing frameworks
  - Try to reuse the component technologies
  - Try to reuse the existing web services