

# Aplicações Internet

# Agenda

- Conceitos Internet
- Formatos de dados web
  - HTML, XML, DTDs
- Arquitetura de sistemas
- Introdução à arquitetura de três camadas
- A camada de apresentação
  - Formulários HTML; Get e Post HTTP, Codificação URL; Javascript; Stylesheets; XSLT
- A camada do médio
  - CGI, servidores de aplicações, Servlets, JavaServerPages, passagem de argumentos, manutenção do estado (cookies).

# Uniform Resource Identifiers (URI)

- Esquema de nomeação uniforme para identificar recursos na Internet.
- Um recurso pode ser:
  - Index.html
  - mysong.mp3
  - Picture.jpg
- Exemplo de URIs:  
<http://www.cs.wisc.edu/~dbbook/index.html>  
<mailto:webmaster@bookstore.com>

# Estrutura de URIs

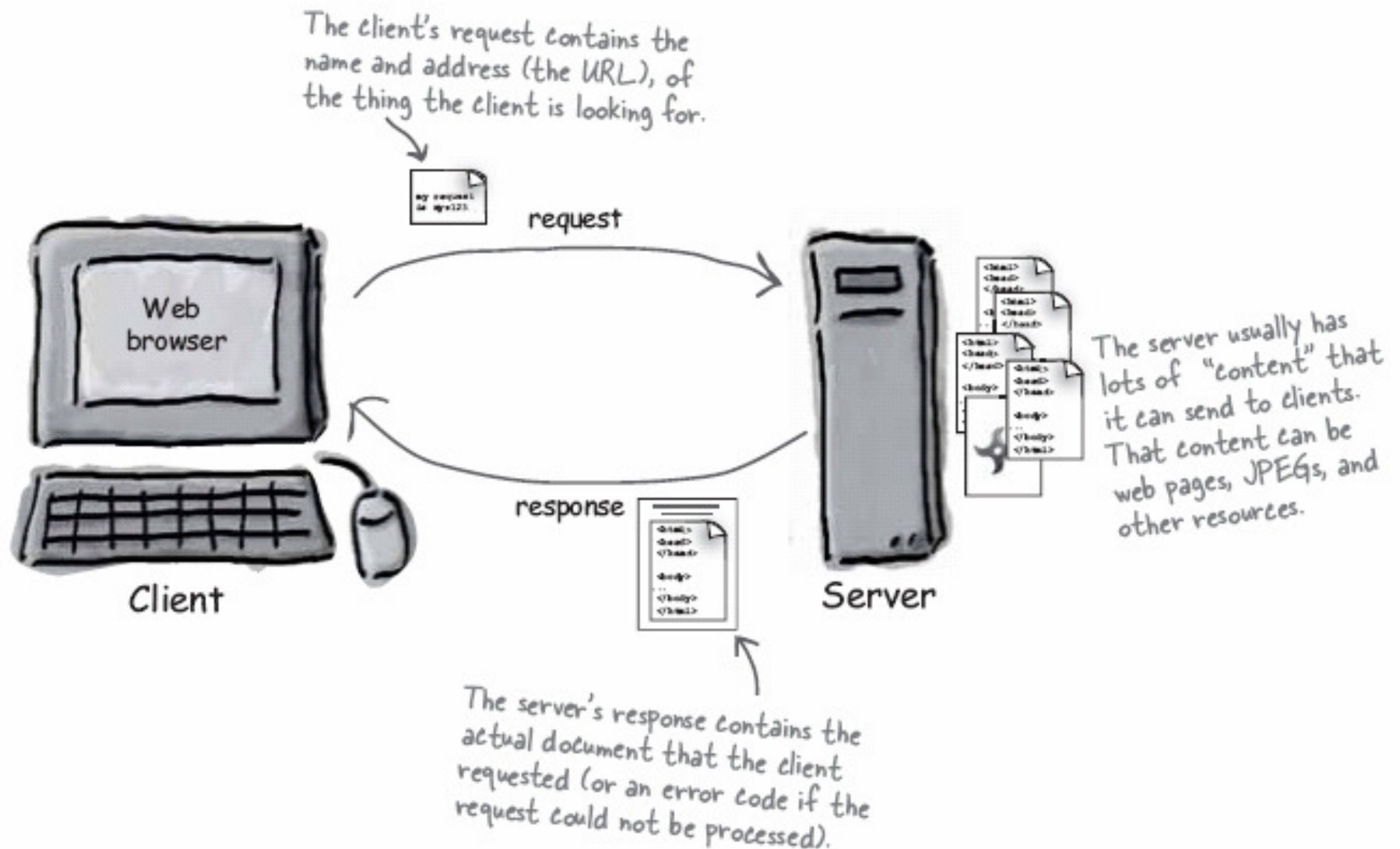
<http://www.cs.wisc.edu/~dbbook/index.html>

- URI tem três partes:
  - Name of the protocol used to access the resource (http)
  - Name of the host computer (www.cs.wisc.edu)
  - Nome do recurso (~dbbook/index.html)
- URLs são um subconjunto de URIs

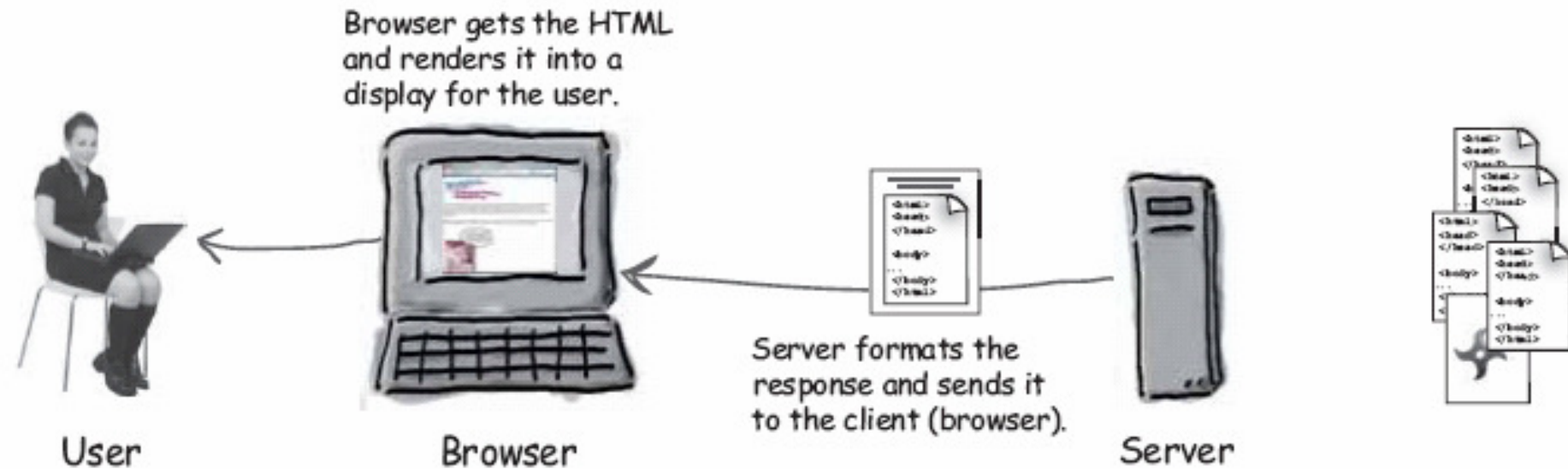
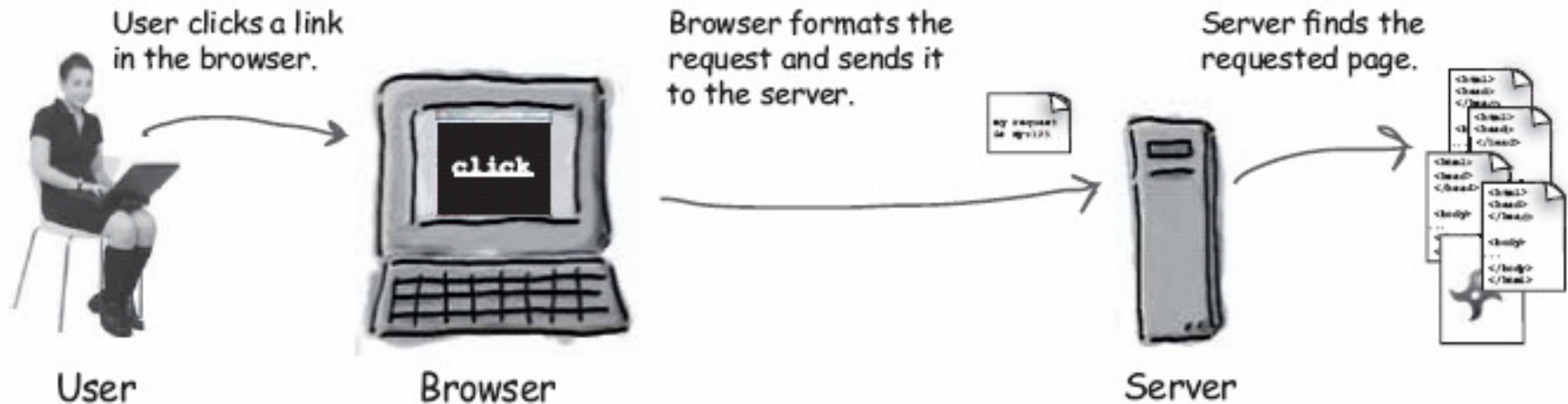
# Hypertext Transfer Protocol

- O que é um protocolo de comunicação?
  - Conjunto de padrões que definem a estrutura das mensagens
  - Exemplos: TCP, IP, HTTP
- O que acontece se você “clica” no [www.cs.wisc.edu/~dbbbook/index.html](http://www.cs.wisc.edu/~dbbbook/index.html)?
  - O cliente (browser) envia um pedido HTTP ao servidor
  - O servidor recebe o pedido e responde
  - O cliente recebe a resposta; faz novos pedidos

# O que faz o servidor.



# O que faz o cliente



# HTTP (Cont.)

## Cliente para servidor:

GET ~/index.html HTTP/1.1

User-agent: Mozilla/4.0

Accept: text/html, image/gif,  
image/jpeg

## Servidor responde:

HTTP/1.1 200 OK

Date: Mon, 04 Mar 2002 12:00:00 GMT

Server: Apache/1.3.0 (Linux)

Last-Modified: Mon, 01 Mar 2002 09:23:24  
GMT

Content-Length: 1024

Content-Type: text/html

<HTML><HEAD></HEAD>

<BODY>

<h1>Barns and Nobble Internet  
Bookstore</h1>

Our inventory

...



# Estrutura do protocolo HTTP

## Pedido (request) HTTP

- Linha de pedido: GET~/index.html/1.1
  - GET: Campo do método Http (valores possíveis são GET e POST, mais tarde)
  - ~/index.html: Campo URI
  - HTTP/1.1: campo de versão HTTP
- Tipo de cliente: User-agent: Mozilla/4.0
- O que tipo de arquivos o cliente aceitará:  
Accept: text/html, image/gif, image/jpeg

# Estrutura do protocolo HTTP (Cont.)

## Resposta HTTP

- Linha de status: HTTP/1.1 200 OK
  - HTTP version: HTTP/1.1
  - Status code: 200
  - Mensagem do servidor: OK
  - Combinações de mensagens do servidor/código do status comuns
    - 200 OK: Pedido com sucesso
    - 400 Bad Request: Pedido não pode ser preenchido pelo servidor
    - 404 Not Found: Objeto pedido não existe no servidor
    - 505 HTTP Versão não suportada
  - Data quando o objeto foi criado:  
Last-Modified: Mon, 01 Mar 2002 09:23:24 GMT
  - Número de bytes sendo enviado: Content-Length: 1024
  - O que tipo de objeto sendo enviado: Content-Type: text/html
  - Outras informações tais como tipo de servidor, tempo do servidor, etc.

# Alguns comentários sobre HTTP

- HTTP é stateless
  - Sem sessões
  - Toda mensagem é completamente auto-contida
  - Nenhuma interação anterior é lembrada pelo protocolo
  - Compromisso entre facilidade de implementação e facilidade de desenvolvimento da aplicação: Outras funcionalidades tem que ser desenvolvidas no topo.
- Implicações para aplicações:
  - Qualquer informação de estado (carro de compras, informação sobre login do usuário) precisa ser codificada em todo pedido e resposta HTTP !
  - Métodos populares de como manter o estado: “cookies” e URLs únicos dinamicamente gerados no nível do servidor.

# Formatos de dados na web

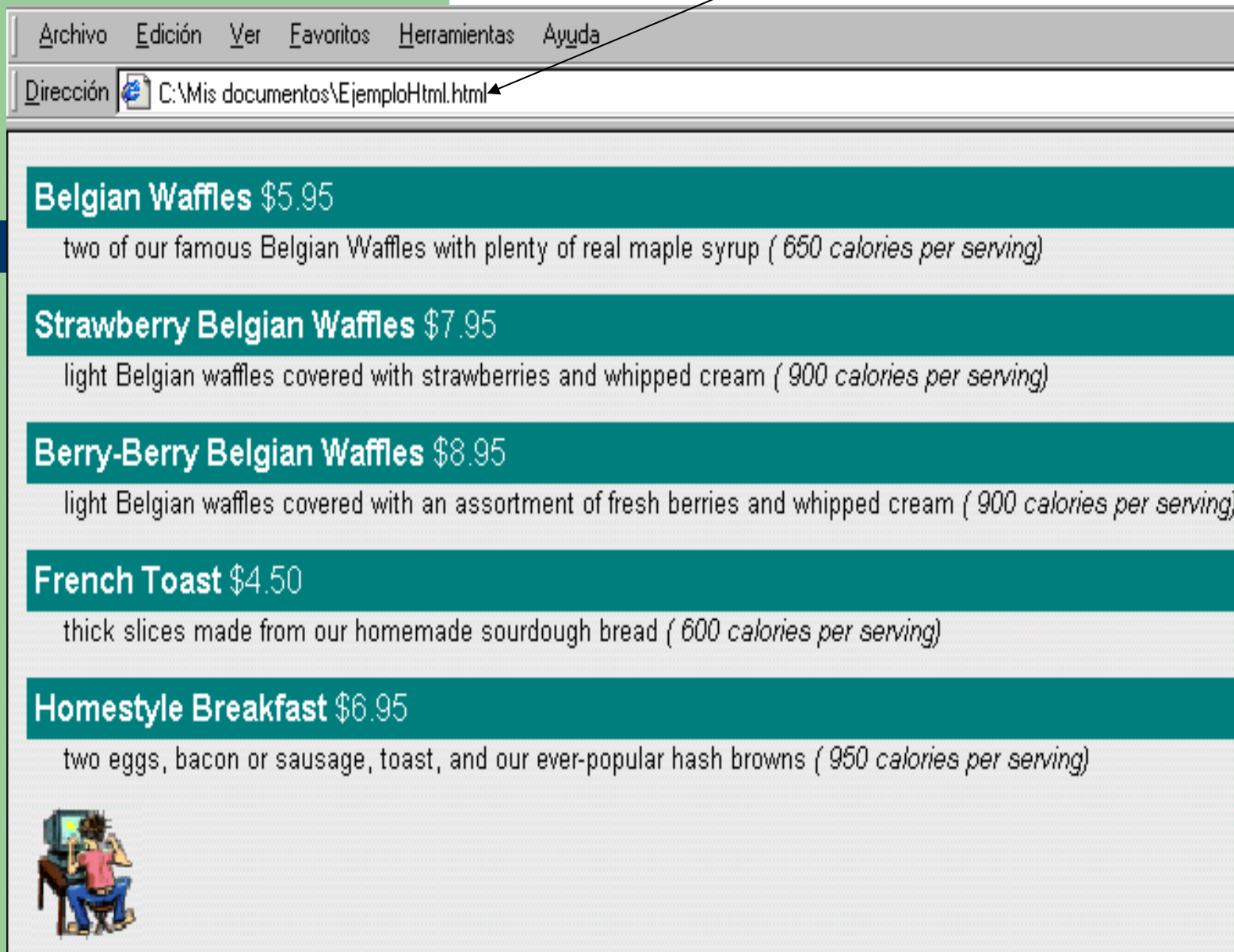
- HTML
  - A linguagem de apresentação para a Internet
- XML
  - Modelo de dados auto-descritivo e hierárquico
- DTD
  - Padronização de esquemas para XML
- XSLT (não tratado)

# HTML (Hypertext Markup Language)

---

- HTML é uma linguagem baseado em SGML.
- SGML é complexo e difícil de processar por ser muito flexível.
- O conjunto de marcas em HTML é fixo.
- O sucesso da Web deve-se, em parte, à simplicidade de HTML.
- Mais na medida que a Web cresce, aparecem problemas no projeto de páginas complexas, busca, “links” inválidos, etc.

## VISÃO DO DOCUMENTO HTML



# ESTRUTURA DOCUMENTO HTML

```
<html> ───────────────────────────────────▶ INICIO
<head>
  <title>Pagina de muestra HTML</title>
</head>
  <BODY STYLE="font-family:Arial, helvetica, sans-serif; font-size:12pt; background-
color:#EEEEEE">
    <DIV STYLE="background-color:teal; color:white; padding:4px">
      <SPAN STYLE="font-weight:bold; color:white">Belgian Waffles</SPAN> $5.95</DIV>
      <DIV STYLE="margin-left:20px; margin-bottom:1em; font-size:10pt">two of our famous
Belgian Waffles with plenty of real maple syrup<SPAN STYLE="font-style:italic"> ( 650
calories per serving) </SPAN> </DIV>

    <DIV STYLE="background-color:teal; color:white; padding:4px">
      <SPAN STYLE="font-weight:bold; color:white">Strawberry Belgian Waffles</SPAN>
$7.95</DIV>
      <DIV STYLE="margin-left:20px; margin-bottom:1em; font-size:10pt">light Belgian waffles
covered with strawberries and whipped cream<SPAN STYLE="font-style:italic"> ( 900
calories per serving) </SPAN> </DIV>
    
  </body>
</html> ───────────────────────────────────▶ FIM
```

# HTML

---

- HTML é orientado à apresentação e não fixa a estrutura, misturando a semântica de ambos os elementos.
- Solução: XML. Linguagem extensível  $\Rightarrow$  não tem um conjunto fixo de marcas. É uma metalinguagem.



# HTML: um exemplo

```
<HTML>
  <HEAD></HEAD>
  <BODY>
    <h1>Barns and Nobble Internet
      Bookstore</h1>
    Our inventory:

    <h3>Science</h3>
    <b>The Character of Physical
      Law</b>
    <UL>
      <LI>Author: Richard
        Feynman</LI>
      <LI>Published 1980</LI>
      <LI>Hardcover</LI>
    </UL>
```

```
    <h3>Fiction</h3>
    <b>Waiting for the Mahatma</b>
    <UL>
      <LI>Author: R.K. Narayan</LI>
      <LI>Published 1981</LI>
    </UL>
    <b>The English Teacher</b>
    <UL>
      <LI>Author: R.K. Narayan</LI>
      <LI>Published 1980</LI>
      <LI>Paperback</LI>
    </UL>
  </BODY>
</HTML>
```

# HTML: Uma pequena introdução

- HTML é uma linguagem de marcação
- Comandos são rótulos (“tags”)>
  - “tag” de início e “tag” de fim
  - Exemplos:
    - <HTML>...</HTML>
    - <UL>...</UL>
- Vários editores geram HTML diretamente do seu documento (ex: Microsoft word)

# HTML: Exemplos de Comandos

- `<HTML>`:
- `<UL>`: lista não ordenada
- `<LI>`: entrada de lista
- `<h1>`: maior cabeçalho
- `<h2>`: cabeçalho de segundo nível, `<h3>`, `<h4>` análogo
- `<B>Título</B>`: Negrito

# XML: A linguagem de marcação extensível

- Linguagem
  - Uma forma de comunicação de informação
- Marcação
  - Notas ou metadados que descrevem seus dados ou linguagens
- Extensível
  - Habilidade sem limite para definir novas linguagens ou conjuntos de dados

# XML – Qual é a vantagem?

- Você pode incluir seus dados e uma descrição do que esses dados representam
  - Isto é útil para definir sua própria linguagem ou protocolo
- Exemplo: Linguagem de Marcação Química

```
<molécula>
  <peso>234.5</peso>
  <Espectro>...</Espectro>
  <Figuras>...</Figuras>
</molécula>
```
- Metas de projeto de XML:
  - XML deve ser compatível com SGML
  - Deveria ser fácil escrever processadores XML
  - O projeto deveria ser formal e preciso

# XML

- Vantagens:

- Separação do conteúdo do estilo.

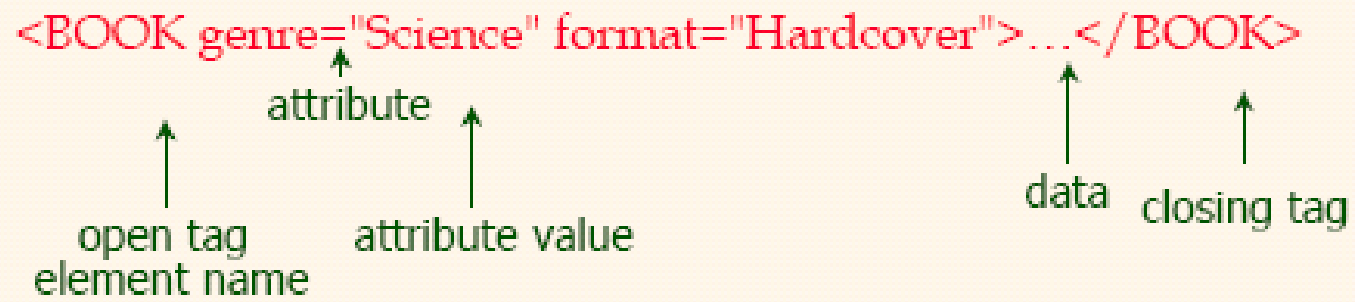
- As páginas Web ficam mais consistentes e fáceis de manter.
    - Facilita a edição de documentos.
    - Busca: Usando XML vai permitir que nossas marcas forneçam um contexto. São necessárias marcas comuns dentro de um domínio.
    - Manter a consistência dos “links”  $\Rightarrow$  Xlink (XML Linking Language)

# XML: Um exemplo

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<BOOKLIST>
  <BOOK genre="Science" format="Hardcover">
    <AUTHOR>
      <FIRSTNAME>Richard</FIRSTNAME><LASTNAME>Feynman</LASTNAME>
    </AUTHOR>
    <TITLE>The Character of Physical Law</TITLE>
    <PUBLISHED>1980</PUBLISHED>
  </BOOK>
  <BOOK genre="Fiction">
    <AUTHOR>
      <FIRSTNAME>R.K.</FIRSTNAME><LASTNAME>Narayan</LASTNAME>
    </AUTHOR>
    <TITLE>Waiting for the Mahatma</TITLE>
    <PUBLISHED>1981</PUBLISHED>
  </BOOK>
  <BOOK genre="Fiction">
    <AUTHOR>
      <FIRSTNAME>R.K.</FIRSTNAME><LASTNAME>Narayan</LASTNAME>
    </AUTHOR>
    <TITLE>The English Teacher</TITLE>
    <PUBLISHED>1980</PUBLISHED>
  </BOOK>
</BOOKLIST>
```

# XML - Estrutura

- XML: Confluência de SGML e HTML
- Xml parece HTML
- Xml é uma hierarquia de tags definido pelo usuário chamados elementos com atributos e dados.
- Dados são descritos por elementos, elementos são descritos por atributos



The diagram shows an XML tag structure with annotations. The tag is `<BOOK genre="Science" format="Hardcover">...</BOOK>`. Annotations with arrows point to specific parts: 'open tag' and 'element name' point to `<BOOK`; 'attribute' points to `genre="Science"`; 'attribute value' points to `"Science"`; 'data' points to `...`; and 'closing tag' points to `</BOOK>`.

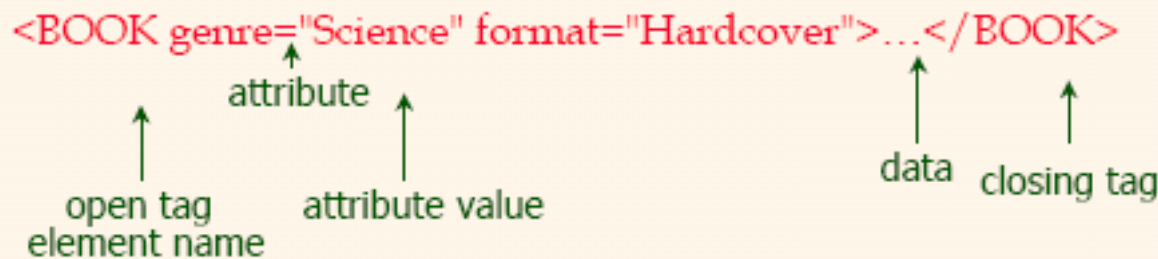
```
<BOOK genre="Science" format="Hardcover">...</BOOK>
```

Annotations:

- open tag (points to `<BOOK`)
- element name (points to `BOOK`)
- attribute (points to `genre="Science"`)
- attribute value (points to `"Science"`)
- data (points to `...`)
- closing tag (points to `</BOOK>`)

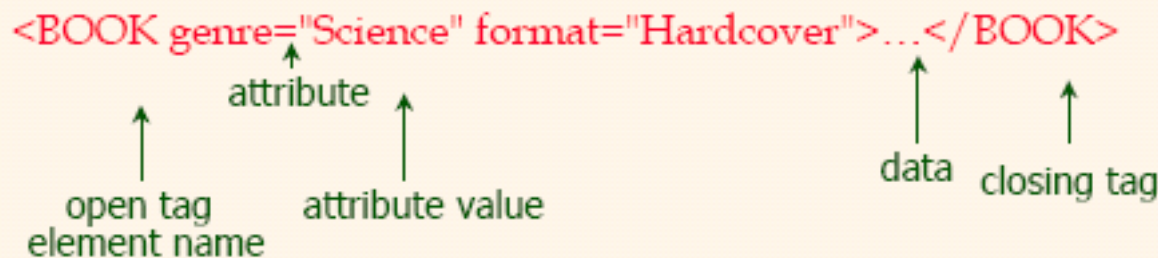


# XML - Elementos



- Os nomes dos tags que iniciam o fecham os elementos são idênticos.
- Tag de abertura: “<” + nome de elemento + “>”
- Tag de fechamento: “</” + nome de elemento + “>”
- Elementos vazios não tem dados nem tags de fechamento:
  - Eles começam com um “<” e finalizam com um “/>”  
`<BOOK/>`

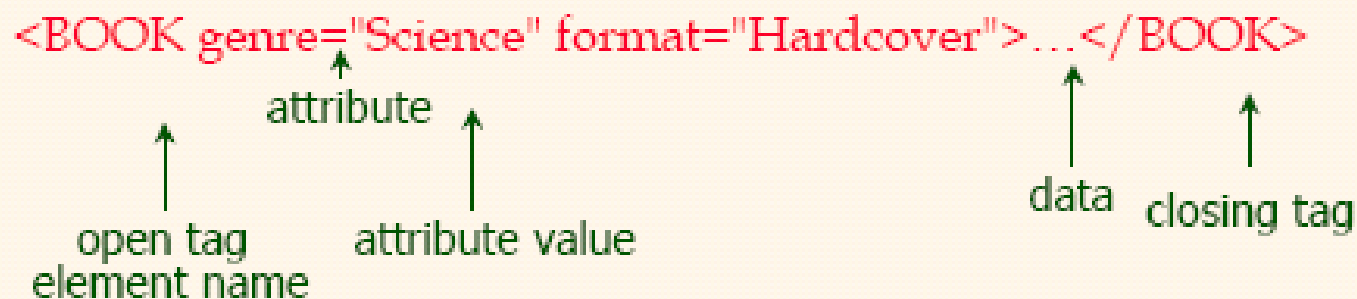
# XML - Atributos



The diagram shows an XML tag: `<BOOK genre="Science" format="Hardcover">...</BOOK>`. Green arrows point from labels to parts of the tag: 'open tag' and 'element name' point to '<BOOK'; 'attribute' points to 'genre'; 'attribute value' points to '"Science"'; 'data' points to '...'; and 'closing tag' points to '</BOOK>'.

- Atributos fornecem informação adicional para os tags de elementos.
- Podem existir zero ou mais atributos em todo elemento; cada um tem a forma:  
Attribute\_name='attribute\_value'
  - Não há espaço entre o nome e o “=”
  - Valores dos atributos devem ser cercados por caracteres “ ou ‘
- Atributos múltiplos são separados por espaços em branco (um ou mais espaços ou tabs).

# XML – Dados e comentários



The diagram shows an XML tag: `<BOOK genre="Science" format="Hardcover">...</BOOK>`. Green arrows point from labels below to parts of the tag: 'open tag' and 'element name' point to '<BOOK'; 'attribute' points to 'genre="Science"'; 'attribute value' points to '"Science"'; 'data' points to '...'; and 'closing tag' points to '</BOOK>'.

- Dados XML é qualquer informação entre um tag de início e outro de fechamento
- Dados XML não devem conter os caracteres “<” ou “>”
- Comentários:  
<!-- comentário -->

# XML – Aninhamento e Hierarquia

- Tags XML podem ser aninhados numa hierarquia em árvore
- Documentos XML podem ter somente um tag raiz.
- Entre um tag de abertura e fechamento você pode inserir:
  1. Dados
  2. Mais elementos
  3. Uma combinação de dados e elementos

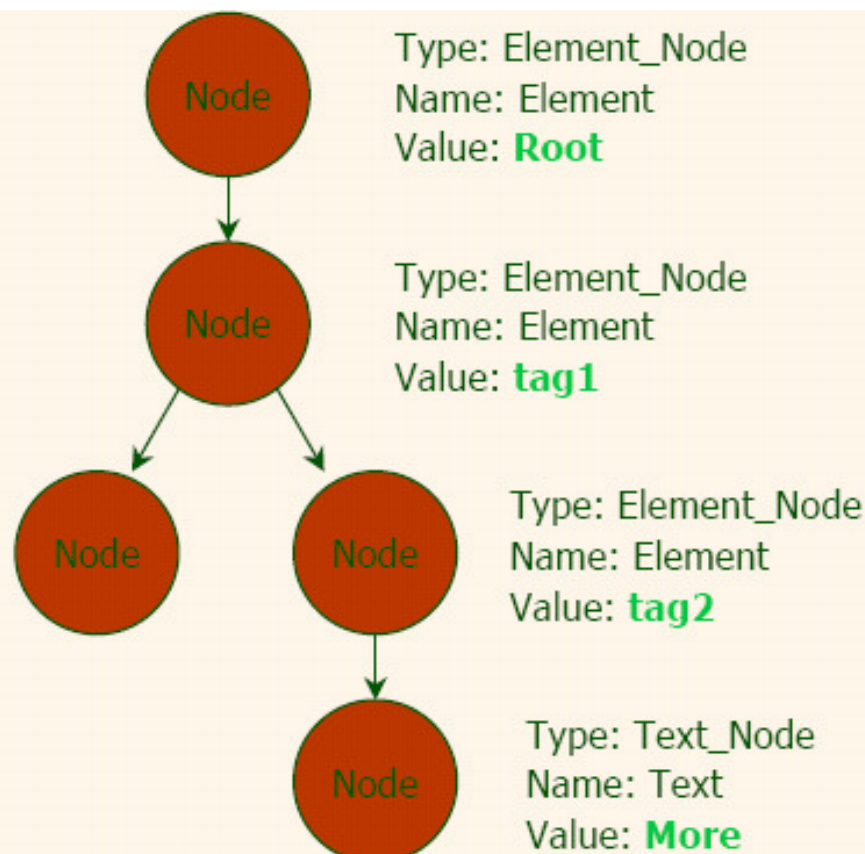
```
<root>
  <tag1>
    Some Text
    <tag2>More</tag2>
  </tag1>
</root>
```

# XML - Armazenamento

- O armazenamento é feito como uma árvore n-ária

```
<root>  
  <tag1>  
    Some Text  
    <tag2>More</tag2>  
  </tag1>  
</root>
```

Type: Text\_Node  
Name: Text  
Value: **Some Text**



# DTD – Document Type Definition

---

- Um DTD é um esquema para dados XML
- Protocolos e linguagens XML podem ser padronizados com arquivos DTD
- Um DTD disse que elementos e atributos são obrigatórios ou opcionais
  - Define a estrutura formal da linguagem

# DTD – Um exemplo

```
<?xml version='1.0'?>
<!ELEMENT Basket (Cherry+, (Apple | Orange)*) >
  <!ELEMENT Cherry EMPTY>
    <!ATTLIST Cherry flavor CDATA #REQUIRED>
  <!ELEMENT Apple EMPTY>
    <!ATTLIST Apple color CDATA #REQUIRED>
  <!ELEMENT Orange EMPTY>
    <!ATTLIST Orange location 'Florida'>
```

---



```
<Basket>
  <Cherry flavor='good'/>
  <Apple color='red'/>
  <Apple color='green'/>
</Basket>
```



```
<Basket>
  <Apple/>
  <Cherry flavor='good'/>
  <Orange/>
</Basket>
```

# DTD - !ELEMENT

`<!ELEMENT Basket (Cherry+, (Apple | Orange)*) >`

Name                      Children

- !ELEMENT declara um nome de elemento, e que elementos filho deveria ter.
- Tipos de conteúdo:
  - Outros elementos
  - #PCDATA (parsed character data)
  - EMPTY (nenhum conteúdo)
  - ANY (nenhuma verificação dentro desta estrutura)
  - Uma expressão regular



## DTD - !ELEMENT (Cont.)

- Uma expressão regular tem a seguinte estrutura:
  - exp1, exp2, exp3, ..., expk: Uma lista de expressões regulares
  - exp\*: Uma expressão opcional com zero ou mais ocorrências
  - exp+: Uma expressão opcional com uma ou mais ocorrências
  - exp1 | exp2 | exp3 | ... | expk: Uma disjunção de expressões

# DTD - !ATTLIST

```
<!ATTLIST Cherry flavor CDATA #REQUIRED>
```

Diagram illustrating the structure of the DTD declaration:

- Cherry: Element
- flavor: Attribute
- CDATA: Type
- #REQUIRED: Flag

```
<!ATTLIST Orange location CDATA #REQUIRED  
color 'orange'>
```

- !ATTLIST define uma lista de atributos para um elemento
- Os atributos podem ser de diferentes tipos, podem ser obrigatórios ou não, e eles podem ter valores default.

## DTD – Bem formada e válida

```
<?xml version='1.0'?>
<!ELEMENT Basket (Cherry+)>
  <!ELEMENT Cherry EMPTY>
    <!ATTLIST Cherry flavor CDATA #REQUIRED>
```

## Not Well-Formed

```
<basket>
  <Cherry flavor=good>
</Basket>
```

## Well-Formed but Invalid

```
<Job>
  <Location>Home</Location>
</Job>
```

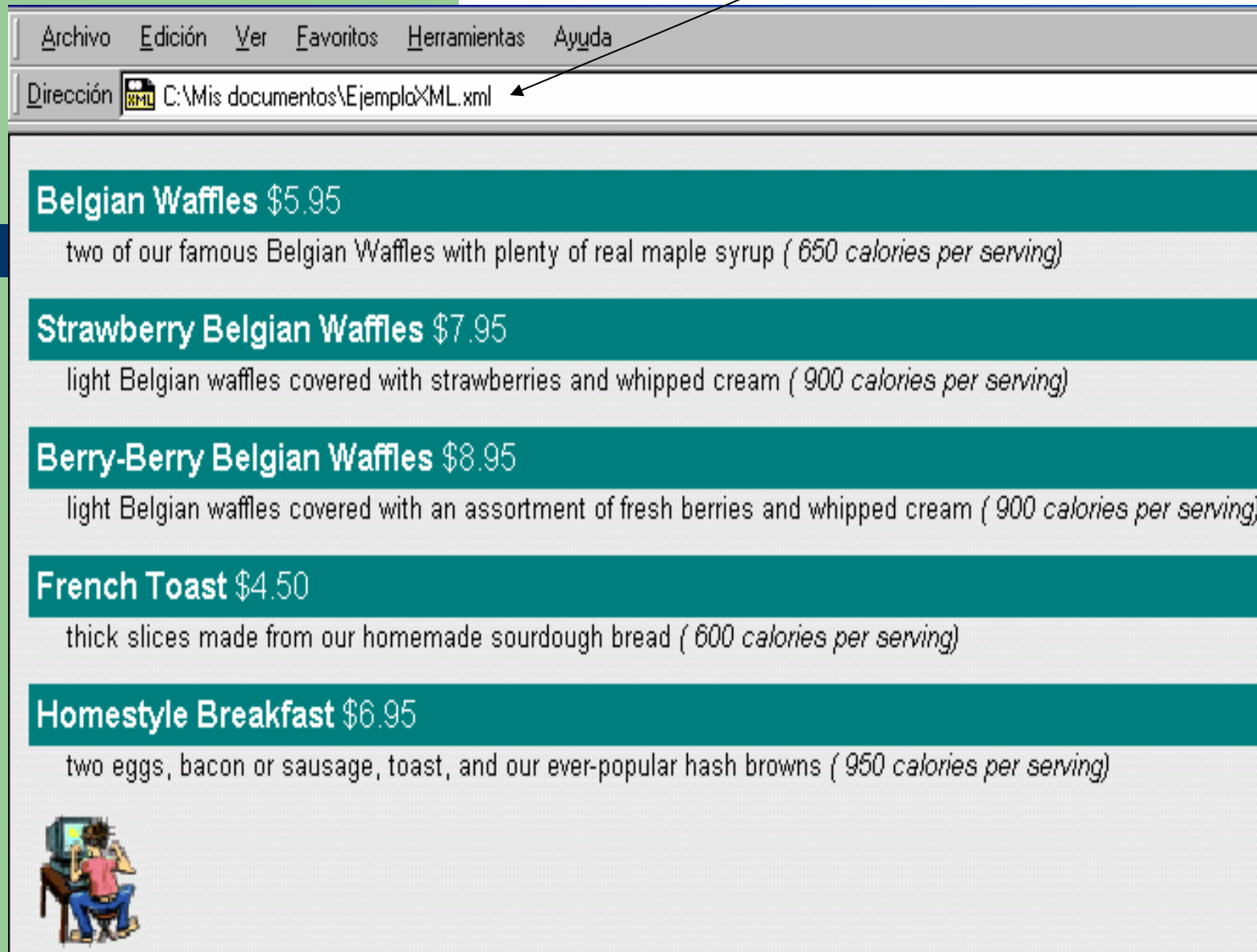
## Well-Formed and Valid

```
<Basket>
  <Cherry flavor='good' />
</Basket>
```

# XML e DTDs

- Cada vez mais DTDs padronizados serão desenvolvidos
  - MathML
  - Chemical Markup Language
- Permite a troca de dados com a mesma semântica.
- São disponíveis linguagens sofisticadas de consulta para XML:
  - Xquery
  - XPath

## VISTA DEL DOCUMENTO XML



# ESTRUCTURA DOCUMENTO XML

```
<?xml version="1.0"?> _____ PROLOGO DEL DOCUMENTO  
<?xml-stylesheet type="text/xsl" href="EjemploXSL.xsl" ?>
```

```
<!DOCTYPE breakfast_menu [ _____  
<!ELEMENT breakfast_menu (food+)>  
<!ELEMENT food (name, price, description, calories)>  
<!ELEMENT calories (#PCDATA)> DTD  
<!ELEMENT description (#PCDATA)>  
<!ELEMENT name (#PCDATA)>  
<!ELEMENT price (#PCDATA)>  
>_____
```

La DTD también se puede escribir de la siguiente forma

```
<!DOCTYPE breakfast_menu SYSTEM archivo.dtd ?>
```

↑  
Para indicar que la DTD esta fuera del documento XML

# ESTRUCTURA DOCUMENTO XML

```
<breakfast_menu>
  <food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>two of our famous
    Belgian Waffles with plenty of
    real maple syrup</description>
    <calories>650</calories>
  </food>
  <food>
    <name>Strawberry Belgian Waffles</name>
    <price>$7.95</price>
    <description>light Belgian waffles covered with
    strawberries and whipped cream</description>
    <calories>900</calories>
  </food>
```

Diagram illustrating the XML structure for a breakfast menu:

- <breakfast\_menu>** is the **ELEMENTO RAIZ** (Root Element).
- <food>** is the **ELEMENTO NODO** (Node Element).
- The elements **<name>**, **<price>**, **<description>**, and **<calories>** are the **ELEMENTOS HOJA** (Leaf Elements).

# ESTRUCTURA DOCUMENTO XSL

```
<?xml version="1.0" ?>
<HTML xmlns:xsl="http://www.w3.org/TR/WD-xsl">
<BODY STYLE="font-family:Arial, helvetica,
  sans-serif; font-size:12pt;
  background-color:#EEEEEE">

<xsl:for-each select="breakfast_menu/food">
<DIV STYLE="background-color:teal; color:white;
  padding:4px">
<SPAN STYLE="font-weight:bold; color:white">
  <xsl:value-of select="name" />
</SPAN>
  <xsl:value-of select="price" />
</DIV>
<DIV STYLE="margin-left:20px; margin-bottom:1em; font-size:10pt">
  <xsl:value-of select="description" />
<SPAN STYLE="font-style:italic">
  (
  <xsl:value-of select="calories" />
  calories per serving)
</SPAN>
</DIV>
</xsl:for-each>

</BODY>
</HTML>
```

→ PROLOGO

→ ESTANDAR QUE IDENTIFICA EL DOCUMENTO COMO UNA HOJA DE ESTILO XSL

→ CICLO PARA RECORRER LOS ELEMENTOS DEL DOCUMENTO XML

→ ATRIBUTOS DE COLOR Y FORMA PARA LOS ELEMENTOS



# XML e Bancos de Dados

- Os dados na Web são semi-estruturados ou não estruturados (texto, imagens, etc.).
- A Web nos fornece com um novo formato, XML, para a troca de dados com estrutura.
- Bancos de Dados está trabalhando um novo modelo, dados semi-estruturados, que são menos restritos com a estrutura.
- A convergência de XML e dado semi-estruturado  $\Rightarrow$  tecnologia Web-Base de Datos.

# Agenda

- Conceitos Internet
- Formatos de dados web
  - HTML, XML, DTDs
- Introdução à arquitetura de três camadas
- A camada de apresentação
  - Formulários HTML; Get e Post HTTP, Codificação URL; Javascript; Stylesheets; XSLT
- A camada do médio
  - CGI, servidores de aplicações, Servlets, JavaServerPages, passagem de argumentos, manutenção do estado (cookies).

# Componentes de sistemas intensivos em dados

- Três tipos separados de funcionalidade:
  - Gestão de dados
  - Lógica da aplicação
  - Apresentação
- A arquitetura do sistema determina se estes três componentes residem sobre uma camada ou são distribuídos a través de várias camadas

# Arquiteturas de uma só camada

- Toda a funcionalidade combinada numa só camada, usualmente num mainframe
  - O usuário acessa a través de terminais burros
- Vantagens:
  - Facilidade de manutenção e administração
- Desvantagens:
  - Hoje, os usuários esperam interfaces gráficas
  - Computação centralizada é ineficiente.

# Arquiteturas Cliente-Servidor

- Divisão do trabalho: cliente leve (“thin”)
  - Cliente implementa somente a interface gráfica do usuário
  - Servidor implementa a lógica de negócios e a gestão de dados
- Divisão do trabalho: cliente pesado (“thick”)
  - Cliente implementa tanto a interface gráfica quanto a lógica de negócios
  - Servidor implementa a gestão de dados.

# Arquiteturas Cliente-Servidor (Cont.)

- Desvantagens dos clientes pesados
  - Não existe um lugar central para atualizar a lógica de negócios.
  - Segurança: O servidor precisa de clientes confiáveis
    - O controle de acesso e autenticação precisa ser gerenciado no servidor
    - Clientes precisam deixar o servidor de BD num estado consistente
    - Uma possibilidade: encapsular todos os acessos ao BD em procedimentos armazenados
  - Não escalar a mais de uns centos de clientes
    - Grande transferência de dados entre o servidor e cliente
    - Mais de um servidor cria um problema:  $x$  clientes,  $y$  servidores:  $x*y$  conexões

# A arquitetura de três camadas

Presentation tier

Client Program (Web Browser)

Middle tier

Application Server

Data management  
tier

Database System

# As três camadas

- Camada de apresentação
  - Principal interface do usuário
  - Precisa adaptar para diferentes tipos dispositivos (PC, PDA, celulares, acesso a voz?)
- Camada do médio
  - Implementa a lógica de negócios (implementa ações complexas, guarda o estado entre diferentes passos de um workflow)
  - Acessa diferentes SGBDs
- Camada de Gestão de dados
  - Um ou mais SGBDs



# Exemplo 1

- Construir um sistema para fazer reservas em linhas aéreas.
- O que é feito nas diferentes camadas\_
- Gestão de dados
  - Info da linha aérea, poltronas disponíveis, info cliente, etc.
- Servidor de aplicações
  - Programas para registrar reservas, cancelar reservas, adicionar novas linhas aéreas, etc.
- Programa cliente
  - Entrada para diferentes usuários, apresenta formulários e respostas do sistema.

# Tecnologias

Client Program  
(Web Browser)

*HTML*  
*Javascript* *AJAX*  
*XSLT* *CSS*

Application Server  
(Tomcat, Apache)

*JSP* *JSF*  
*Servlets*  
*Cookies*  
*CGI*

↕ JDBC

Database System  
(DB2)

*XML*  
*Stored Procedures*

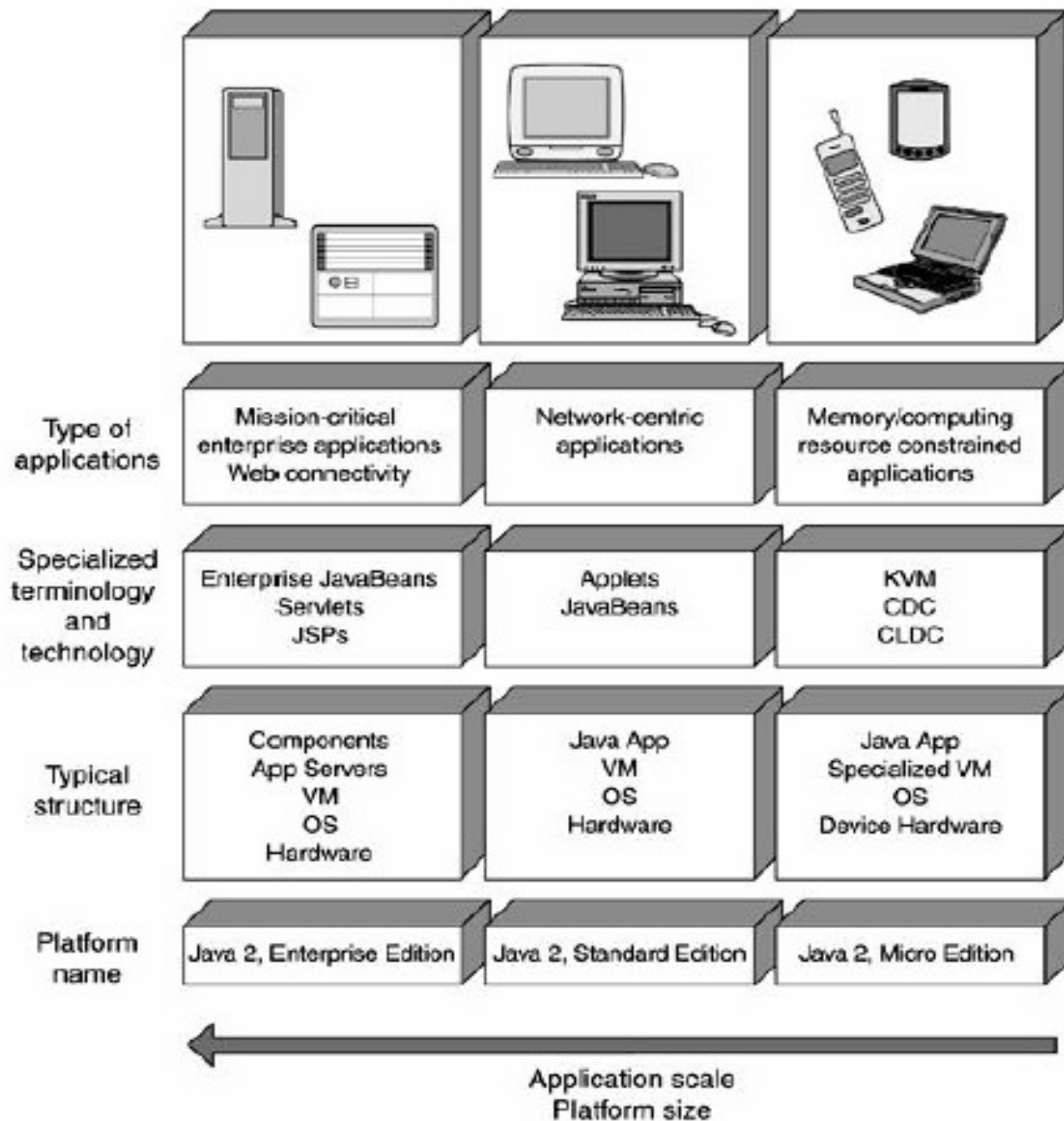
# Vantagens da Arquitetura de três camadas

- **Sistemas Heterogêneos**
  - Camadas podem ser mantidos, modificados, e trocados independentemente
- **Clientes leves**
  - Somente camadas de apresentação nos clientes (visualizadores web)
- **Acesso de dados integrados**
  - Vários sistemas de BDs podem ser manipulados transparentemente na camada do médio
  - Gestão central de Conexões
- **Escalabilidade**
  - Replicação na camada do médio permite escalabilidade da lógica de negócios
- **Desenvolvimento de software**
  - Código da lógica de negócios é centralizado
  - Interação entre camadas a través de APIs bem definidos. Podem reusar componentes padrão em cada camada.

# J2EE Java 2 Platform Enterprise Edition

- Sun Microsystems tem organizado a plataforma Java 2 em três áreas específicas:
  - Edição Micro (J2ME); edição padrão (J2SE); e edição empresa (J2EE).
- J2EE, mais importante para o desenvolvimento de aplicações empresariais.
- J2EE é o resultado de alinhar diferentes tecnologias Java e APIs juntos em plataformas de desenvolvimento java para a execução de tipos específicos de aplicações.

Figure 2-2. Overview of the Java 2 platforms



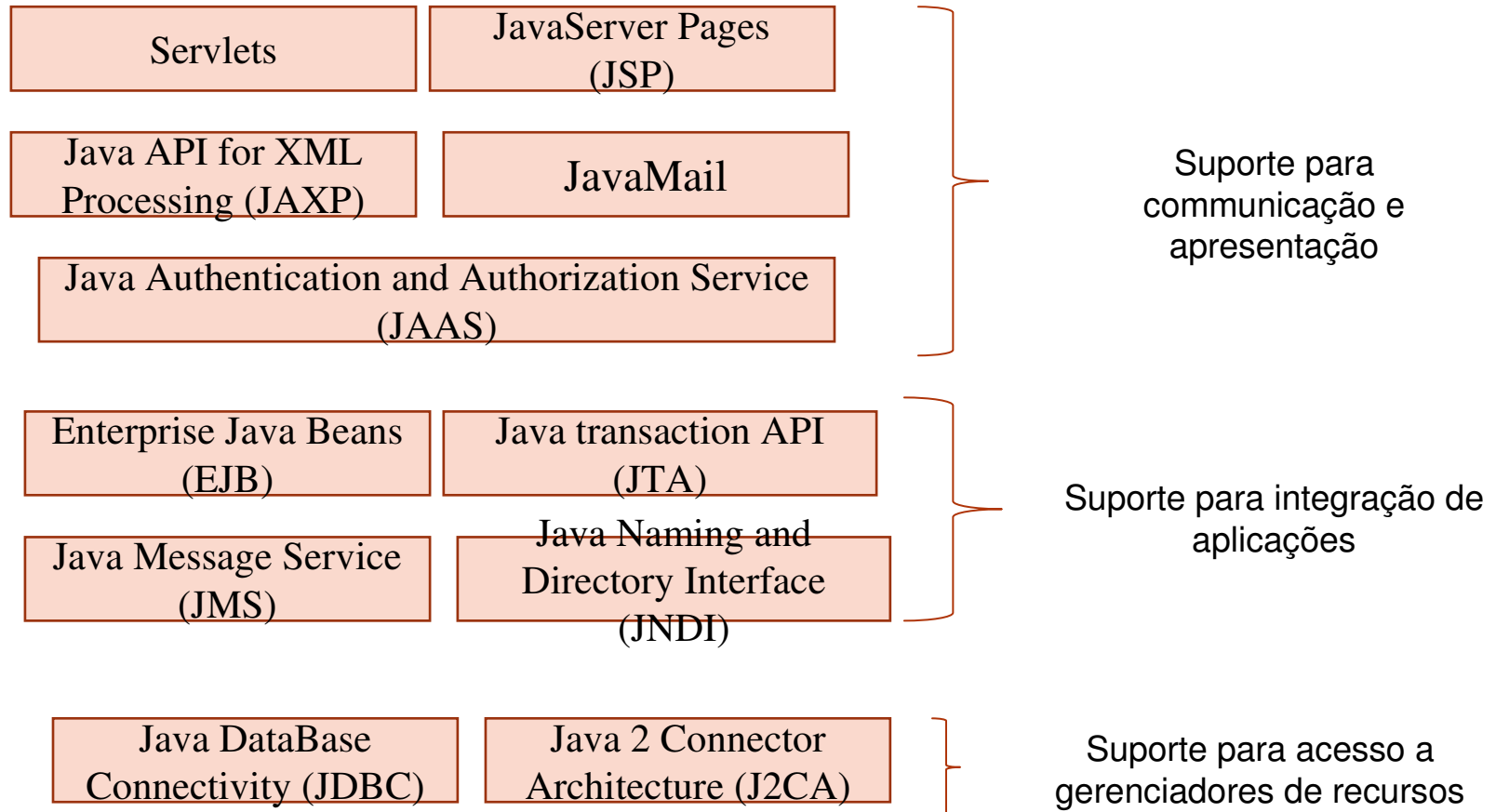
# Servidores de Aplicação

- Servidores de aplicações foram uma resposta ao desafio de integração
- Vendedores tentam fornecer a lógica do middleware
  - Por ex. para transações, segurança, persistência
  - Funcionalidade é similar a CORBA, monitores TP e brokers de mensagens
- A meta é fornecer um ambiente para hospedar todo tipo de lógica de aplicação:
  - Pode ser usado para EAI assim como para integração baseada na Web
- Duas plataformas dominantes: J2EE e .NET
- (nós usaremos J2EE como um exemplo nos próximos slides)

# Como o Java EE pode ajudar a enfrentar problemas

- As aplicações web possuem regras de negócio bastante complicadas. Codificar essas regras representam um grande trabalho. Além dessas regras funcionais, existem outros requisitos que precisam ser atingidos através da infra-estrutura: persistência, transação, acesso remoto, web services, gerenciamento de threads, gerenciamento de conexões http, cachê de objetos, gerenciamento da sessão web, balanceamento de carga, etc.
- Ter responsabilidade de tudo isso seria muito trabalhoso.
- Java EE consiste de uma série de especificações, dando uma receita de como deve ser implementado um software que faz cada um desses serviços.

# J2EE: Componentes Principais





# Algumas especificações do Java EE

- As APIs a seguir são as principais dentre as disponibilizadas pelo Java Enterprise:
  - JavaServer Pages (JSP), Java Servlets, Java Server Faces (JSF) (Trabalhar para a Web, onde é focado este curso).
  - Enterprise JavaBeans Components (EJB) e Java Persistence API (JPA) (objetos distribuídos, clusters, acesso remoto a objetos, etc.)
  - Java API for XML Web Services (JAX-WS), Java API for XML Binding (trabalhar com arquivos XML e webservices)
  - Java Authentication and Authorization Service (JAAS) (API padrão do Java para segurança)
  - Java Transaction API (JTA) (controle de transação no Contêiner)
  - Java Message Service (JMS) (troca de mensagens assíncronas)
  - Java Naming and Directory Interface (JNDI) (espaço de nomes e objetos)
  - Java Management Extensions (JMX) (administração da sua aplicação e estatísticas sobre a mesma)

# Suporte para a camada de Aplicação: EJB

- Suporte para a lógica de aplicação através de: EJB, JNDI e JMS
- EJB (Enterprise Java Beans) está no centro de J2EE
  - Componente do lado do Servidor: oferece aplicações de funcionalidade específica
  - Especifica três tipos diferentes de beans:
    - Beans de Sessão manipula uma sessão com um cliente. Pode ser stateful ou stateless. Ex.: Carrinho de shopping (stateful)
    - Entity beans vivem além da fronteira de uma sessão do cliente. Eles tem estado e estão armazenados persistentemente.
    - Message-driven beans atende a interação assíncrona com os clientes
  - O contenedor EJB fornece o ambiente no qual os beans executam
    - Toda as interações entre o EJB e outros objetos se dá através do contenedor
    - Fornece serviços como transações, persistência, segurança (desenvolvedores não precisam implementar esta funcionalidade)

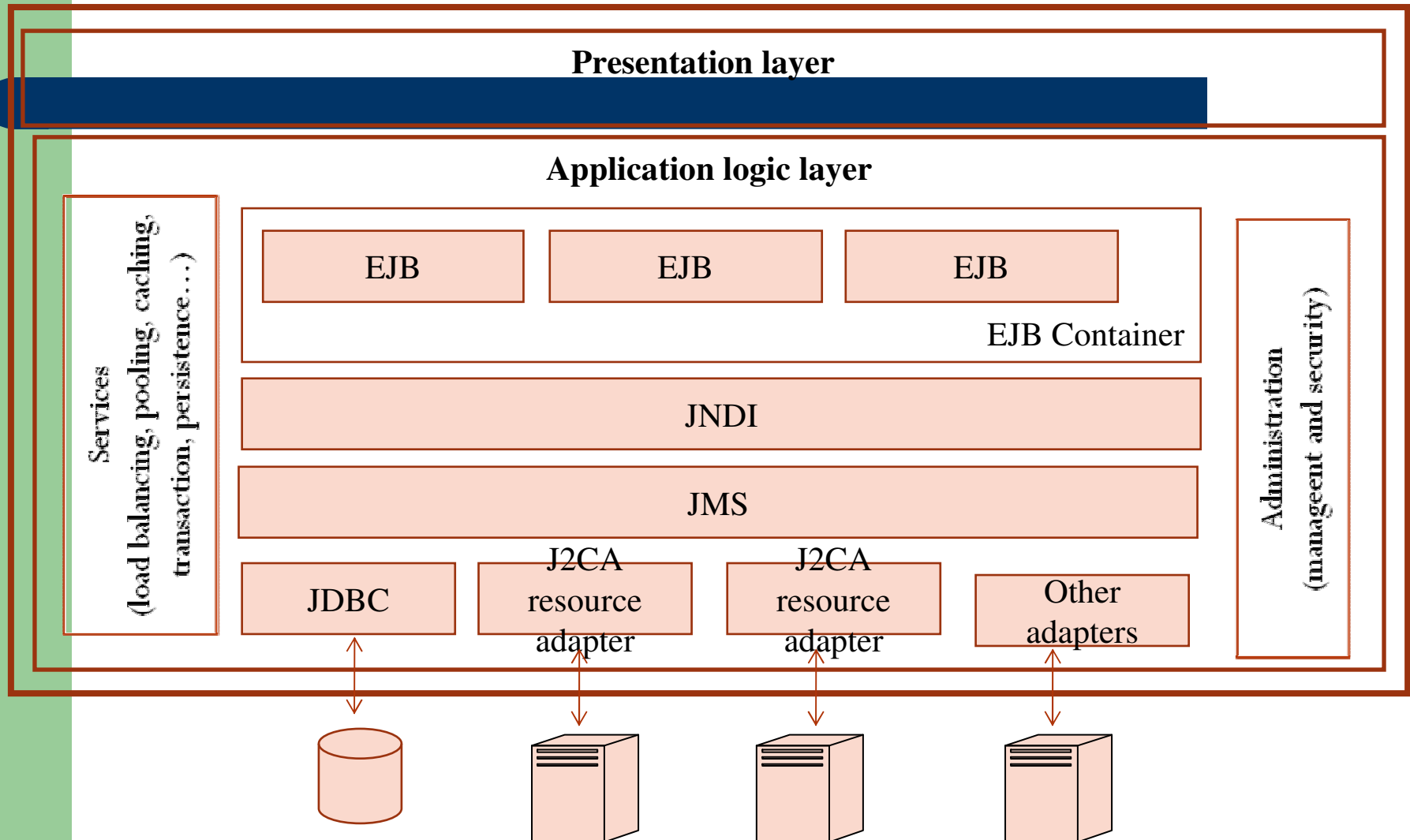
## Suporte para a camada de aplicação: JNDI

- Java Naming and Directory Interface define uma interface para serviços de diretório:
  - Não obriga uma implementação
  - Em geral, permite que os clientes se liguem a um servidor baseado no nome do objeto
  - Permite a ligação com EJBs, no contexto J2EE

# Conectando-se à camada de Recursos: JDBC & J2CA

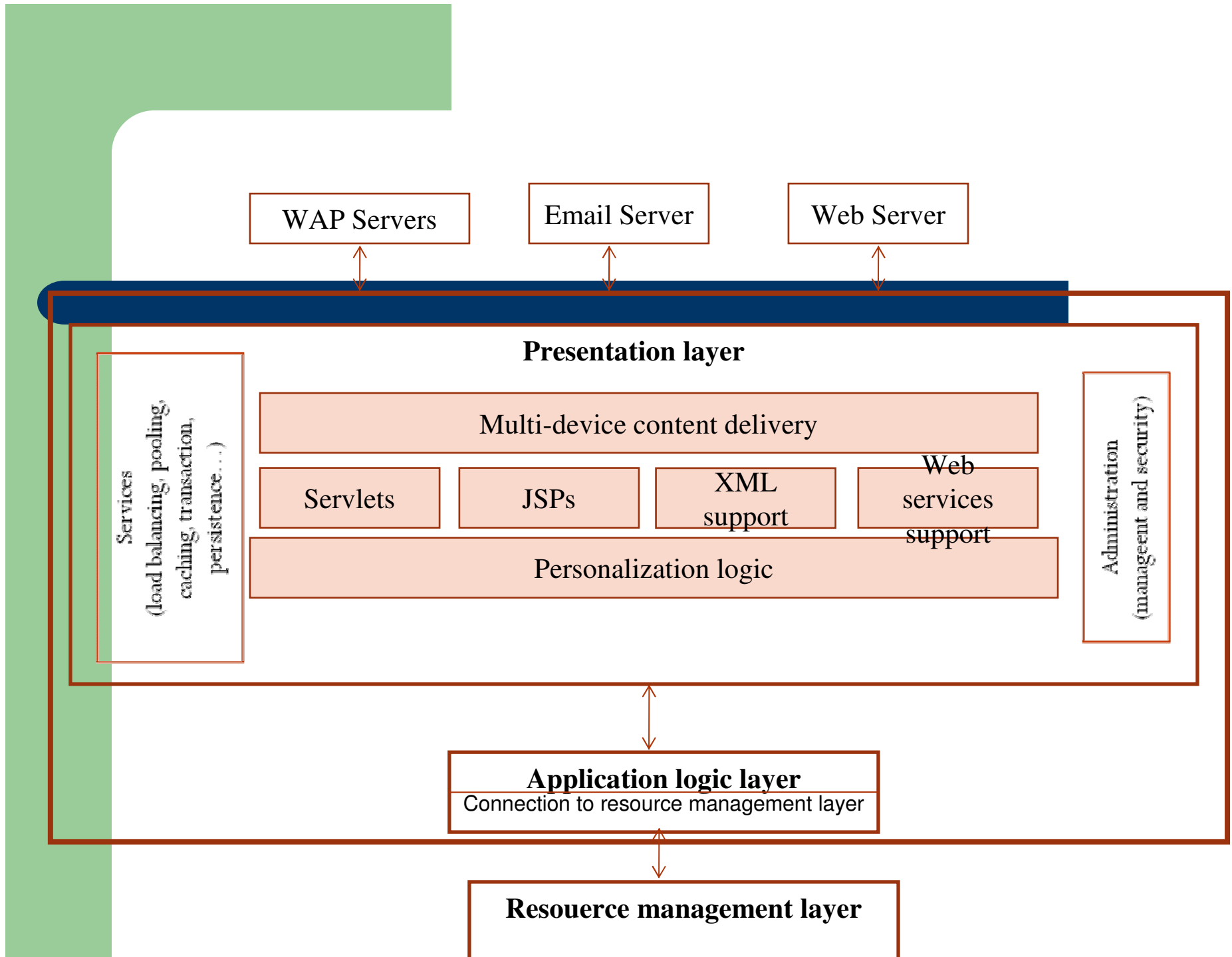
- JDBC:
  - API que permite aos desenvolvedores se conectar a um BD relacional executando comandos SQL desde um programa Java
  - Os métodos JDBC podem ser chamados desde um EJB ou diretamente desde um servlet (ignorando o acesso a uma camada de lógica da aplicação )
- J2CA:
  - Generalização da abordagem de JDBC
  - Define como construir adaptadores de recursos
    - Componentes java que podem ajudar EJB e outras aplicações Java na interação com gerenciadores de recursos
    - Caracterizado por contratos que definem a API que aplicações Java podem usar para acessar o gerenciador de recursos

# Suporte para a camada da lógica de aplicação



# Suporte para a camada de apresentação

- Este suporte diferencia servidores de aplicação do middleware convencional
- Servidores de aplicação fornecem características de apresentação e personalização para uma variedade de clientes:
  - Web browsers, aplicações, dispositivos, programas de correio, clientes de Web services



# Servidor de Aplicação

- Como vimos o Java EE é um grande conjunto de especificações. Essas especificações, quando implementadas, vão auxiliar bastante o desenvolvimento da sua aplicação.
- É necessário fazer download de implementações das especificações.
- Esses softwares tem o papel de servir sua aplicação para auxiliá-la com serviços de infraestrutura → servidor de aplicação.
- Exemplos: Glassfish da Sun, WebLogic Application Server da Oracle, IBM Websphere, Jboss da RedHat, Apache Geronimo, etc.



# Servlet Container

- Servidor que suporta funcionalidades para o desenvolvimento de uma aplicação web mas não necessariamente o Java EE completo.
- Exemplos: Apache Tomcat, Jetty, Cloud Google App Engine, etc.

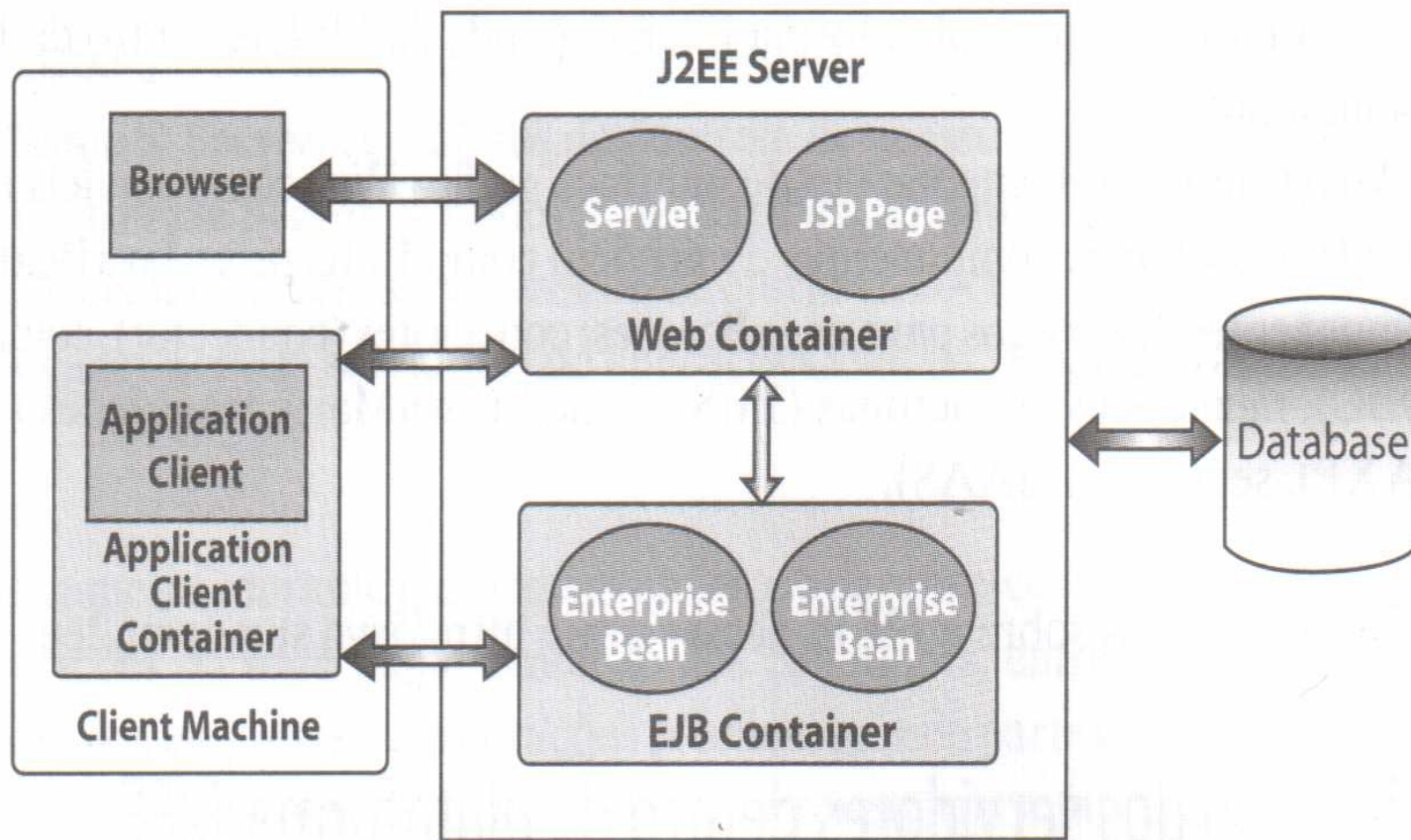


Figura 3.1.

# Plataformas J2SE e J2EE

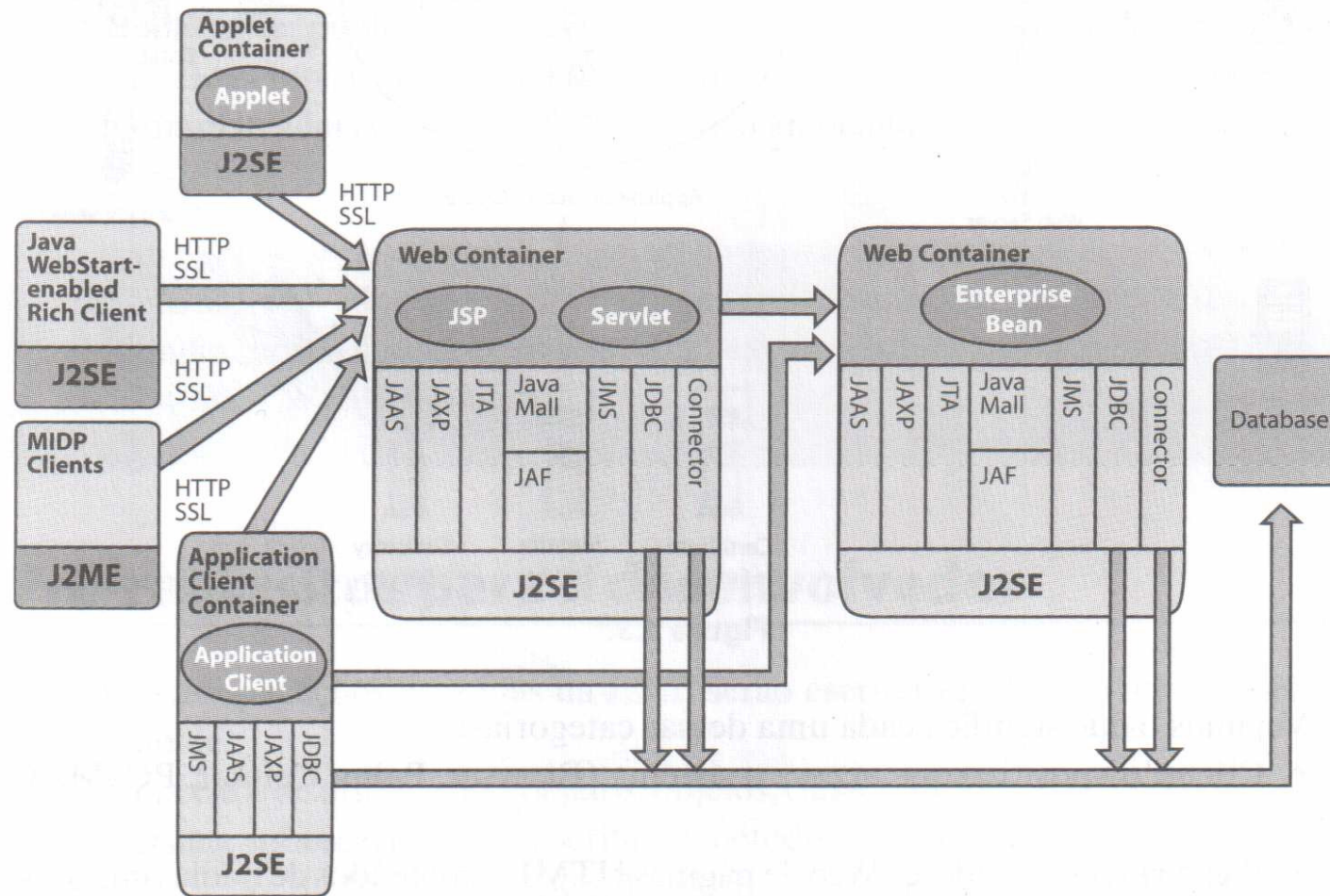


Figura 3.2.