Fundamentos de arquitetura Web

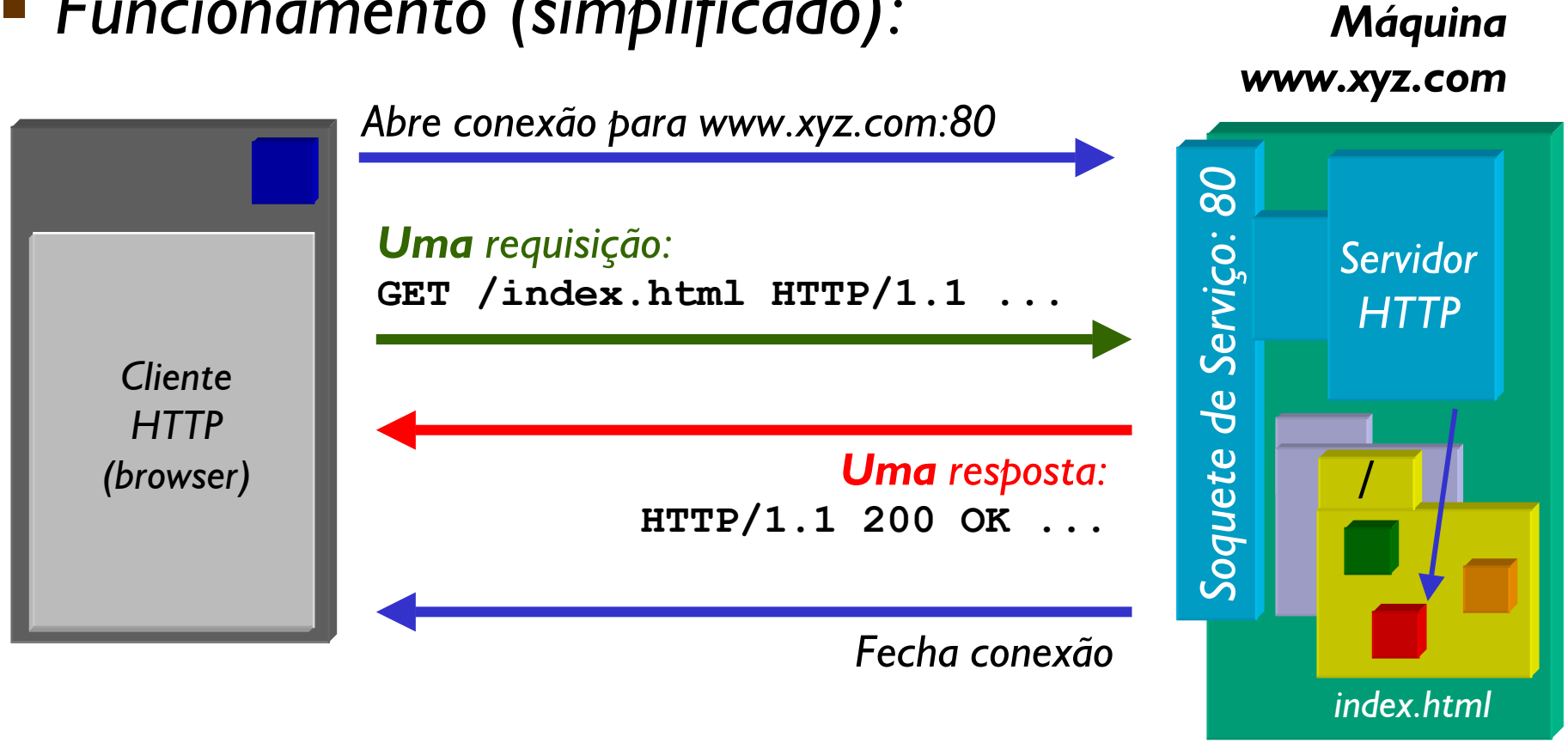
Helder da Rocha (helder@acm.org)
www.argonavis.com.br

Objetivos

- *Este módulo apresenta uma visão geral da plataforma Web*
  - *Lado-cliente*
  - *Lado-servidor*
  - *Protocolo de comunicação HTTP*
- *Descreve o funcionamento de HTTP e as principais tecnologias utilizadas na Web*
- *Apresenta tecnologias Java para a Web: servlets e JSP*
- *Introduz o ambiente de desenvolvimento: Tomcat*

A plataforma Web

- *Baseada em HTTP (RFC 2068)*
  - *Protocolo simples de transferência de arquivos*
  - *Sem estado (não mantém sessão aberta)*
- *Funcionamento (simplificado):*



Cliente e servidor HTTP

- Servidor HTTP
  - Gerencia sistema virtual de arquivos e diretórios
  - Mapeia pastas do sistema de arquivos local (ex: c:\htdocs) a diretórios virtuais (ex: /) acessíveis remotamente (notação de URI)
- Papel do servidor HTTP
  - Interpretar requisições HTTP do cliente (métodos GET, POST, ...)
  - Devolver resposta HTTP à saída padrão (código de resposta 200, 404, etc., cabeçalho RFC 822* e dados
- Papel do cliente HTTP
  - Enviar requisições HTTP (GET, POST, HEAD, ...) a um servidor. Requisições contém URI do recurso remoto, cabeçalhos RFC 822 e opcionalmente, dados (se método HTTP for POST)
  - Processar respostas HTTP recebidas (interpretar cabeçalhos, identificar tipo de dados, interpretar dados ou repassá-los.

* Padrão Internet para construção de cabeçalhos de e-mail

Principais métodos HTTP (requisição)

- ***GET*** *- pede ao servidor um arquivo (informado sua URI) absoluta (relativa à raiz do servidor)*

```
GET <uri> <protocolo>/<versão>
<Cabeçalhos HTTP>: <valores> (RFC 822)
<linha em branco>
```

  - *GET pode enviar dados através da URI (tamanho limitado)*

```
<uri>?dados
```

  - *Método HEAD é idêntico ao GET mas servidor não devolve página (devolve apenas o cabeçalho)*

- ***POST*** *- envia dados ao servidor (como fluxo de bytes)*

```
POST <uri> <protocolo>/<versão>
<Cabeçalhos HTTP>: <valores>
<linha em branco>
<dados>
```

Cabeçalhos HTTP

- *Na **requisição**, passam informações do cliente ao servidor*
  - *Fabricante e nome do browser, data da cópia em cache, cookies válidos para o domínio e caminho da URL da requisição, etc.*
- *Exemplos:*

```
User-Agent: Mozilla 5.5 (Compatible; MSIE 6.0; MacOS X)
If-Modified-Since: Thu, 23-Jun-1999 00:34:25 GMT
Cookies: id=344; user=Jack; flv=yes; mis=no
```

- *Na **resposta**: passam informações do servidor ao cliente*
  - *Tipo de dados do conteúdo (text/xml, image/gif) e tamanho, cookies que devem ser criados. endereço para redirecionamento, etc.*
- *Exemplos:*

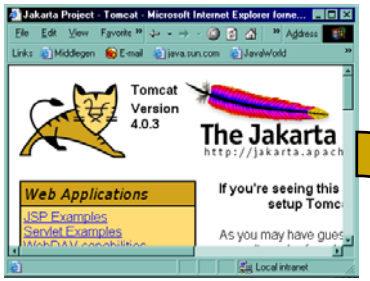
```
Content-type: text/html; charset-iso-8859-1
Refresh: 15; url=/pags/novaPag.html
Content-length: 246
Set-Cookie: nome=valor; expires=Mon, 12-03-2001 13:03:00 GMT
```

Comunicação HTTP: detalhes

- *1. Página HTML*

```
<img src="tomcat.gif" />
```

Interpreta HTML



Gera requisição GET

- *2. Requisição: browser solicita imagem*

```
GET tomcat.gif HTTP/1.1
User-Agent: Mozilla 6.0 [en] (Windows 95; I)
Cookies: querty=uiop; SessionID=D236S11943245
→
```

Linha em branco termina cabeçalhos

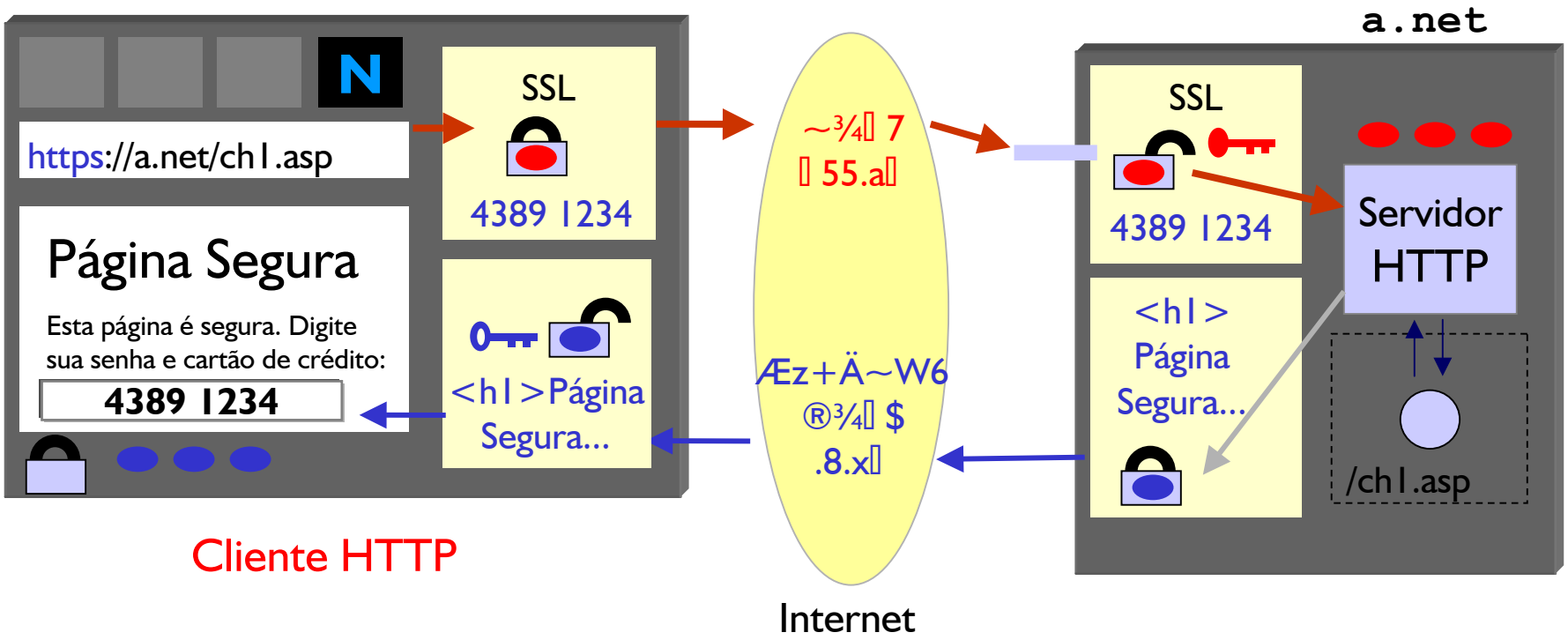
- *3. Resposta: servidor devolve cabeçalho + stream*

```
HTTP 1.1 200 OK
Server: Apache 1.32
Date: Friday, August 13, 2003 03:12:56 GMT-03
Content-type: image/gif
Content-length: 23779
→
!#GIF89~¾ 7
.55.a  6¤Ü4  ...
```

tomcat.gif

SSL - Secure Sockets Layer

- *Camada adicional na comunicação HTTP que introduz criptografia da comunicação*
- *Tanto o browser quanto o servidor têm que suportar o recurso para que uma transação seja segura*
- *Porta default : 443*



Serviço Web: funções

- Serviço de informações
  - finalidade: publicação de informações, multimídia
  - interatividade: limitada a hipertexto
  - tecnologias (passivas): HTML, folhas de estilo
- Serviço de aplicações locais (rodam no cliente)
  - finalidade: oferecer mais recursos interativos ao cliente
  - interatividade: limitada pelo cliente
  - tecnologias (ativas): JavaScript, applets Java, Flash, ActiveX
- Serviço de aplicações cliente/servidor
  - finalidade: oferecer interface para aplicações no servidor
  - interatividade: limitada pela aplicação e servidor Web
  - tecnologias (ativas): CGI, ASP, ISAPI, Servlets, JSP

Serviço de informações: Tecnologias de apresentação

- *HTML 4.0 (HyperText Markup Language)*
  - *Coleção de marcadores (SGML) usados para formatar texto:*
    - *<H2>Cabeçalho de Nível 2</H2>*
    - *<P>Primeiro parágrafo</P>*
  - *Nada diz sobre aparência (browser é quem decide). Define apenas estrutura e conteúdo.*
- *CSS 2.0 (Cascading Style Sheets)*
  - *Lista de regras de apresentação para uma página ou todo um site (linguagem declarativa)*
  - *Depende da estrutura do HTML. Define forma.*
- *Padrões W3C (http://www.w3.org)*

HTML - HyperText Markup Language

- Define a interface do usuário na Web
- Pode ser usada para
  - Definir a estrutura do texto de uma página (que o browser posteriormente formatará com uma folha de estilos)
  - Incluir imagens numa página
  - Incluir vínculos a outras páginas
  - Construir uma interface com formulários para envio de dados ao servidor
  - Servir de base para aplicações rodarem dentro do browser (applets Java, plug-ins, vídeos, etc.)

CSS - Cascading Style Sheets

- *Linguagem usada para definir folhas de estilo que podem ser aplicadas a todo o site.*
  - *Cuida exclusivamente da aparência (forma) da página*
  - *Permite posicionamento absoluto de textos e imagens, manipulação com fontes, cores, etc.*
- *Regras são colocadas em arquivo de texto .css:*

```
H1, H2     { color: rgb(91.5%, 13%, 19%);
                font-size: 72pt; margin-top: 0.6cm}
P.citacao    {font-family: Garamond, serif;
               font-weight:: 300}
```

Limitações

- *HTML e CSS são linguagens declarativas, interpretadas pelo browser, que definem apenas como a informação será organizada e apresentada.*
- *Não oferecem recursos de programação.*
- *Os formulários criados com HTML não fazem nada (eles precisam ser vinculados a uma aplicação)*
- *Não é possível construir aplicações Web interativas utilizando apenas CSS e HTML*
- *Dynamic HTML: solução para alguns problemas*
  - *apresentação + estrutura + interatividade*

Serviço de aplicações: Tecnologias interativas

- Lado-cliente
  - Linguagens de extensão: JavaScript, VBScript
  - Plug-ins e componentes (applets, activeX)
  - Soluções integradas: DHTML
- Persistência de sessão cliente-servidor
  - Cookies
- Lado-servidor
  - CGI, plug-ins do servidor e componentes
  - Linguagens de extensão: SAPIs, ASP, JSP, PHP

Tecnologias lado-cliente

- Estendem a funcionalidade básica do browser (que é apresentação de informação)
- Permitem criar uma interface do usuário dinâmica
  - Tratamento de eventos
  - Alteração dinâmica do conteúdo ou da apresentação
  - Realização de cálculos e computação
  - Disposição de recursos não disponíveis no browser
- Principais tecnologias
  - Extensões do HTML (scripts): JavaScript, VBScript, linguagens proprietárias
  - Extensões do browser (componentes): Applets, ActiveX, Plug-ins

Scripts : extensões do HTML

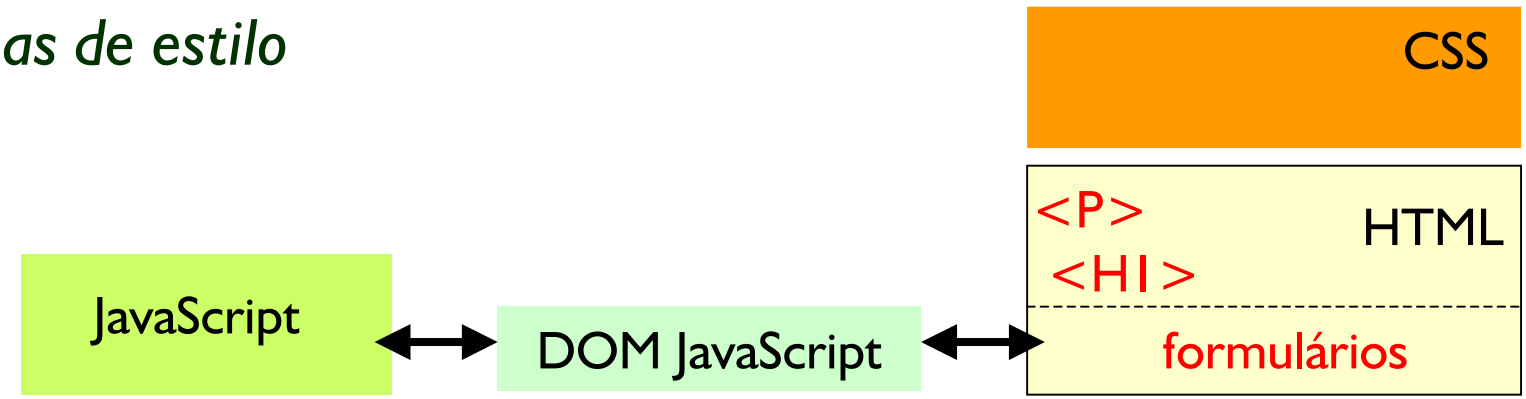
- Forma mais flexível de estender o HTML
- Código geralmente fica visível na página:
  - ```
    <head>
    <script language="JavaScript"><!--
      function soma(a, b) {
         return a + b;
      }
    //--></script></head>
    ```
- Linguagens de roteiro (script) mais populares
  - VBScript: baseado na sintaxe do Visual Basic. MSIE-only.
  - JavaScript/JScript: sintaxe semelhante a de Java
- Código é interpretado diretamente pelo browser (e não por uma máquina virtual, como ocorre com os applets)
    ```

JavaScript e ECMAScript

- JavaScript não é uma versão limitada de Java.
  - Possui sintaxe procedural semelhante, mas é interpretada, baseada em objetos e bem menor.
- JavaScript pode ser usada no browser ou no servidor. A linguagem possui duas partes
  - Núcleo (padrão ECMA chamado de ECMAScript)
  - Modelo de objetos do documento (quando usada no browser) ou do servidor (quando usada no servidor em tecnologias ASP, Livewire, etc.) - implementa padrão DOM
- O núcleo da linguagem define as estruturas de programação, sintaxe e objetos de propósito geral.
- Quando usada no browser, várias estruturas do HTML são acessíveis como 'objetos' JavaScript, permitindo que a linguagem os manipule.

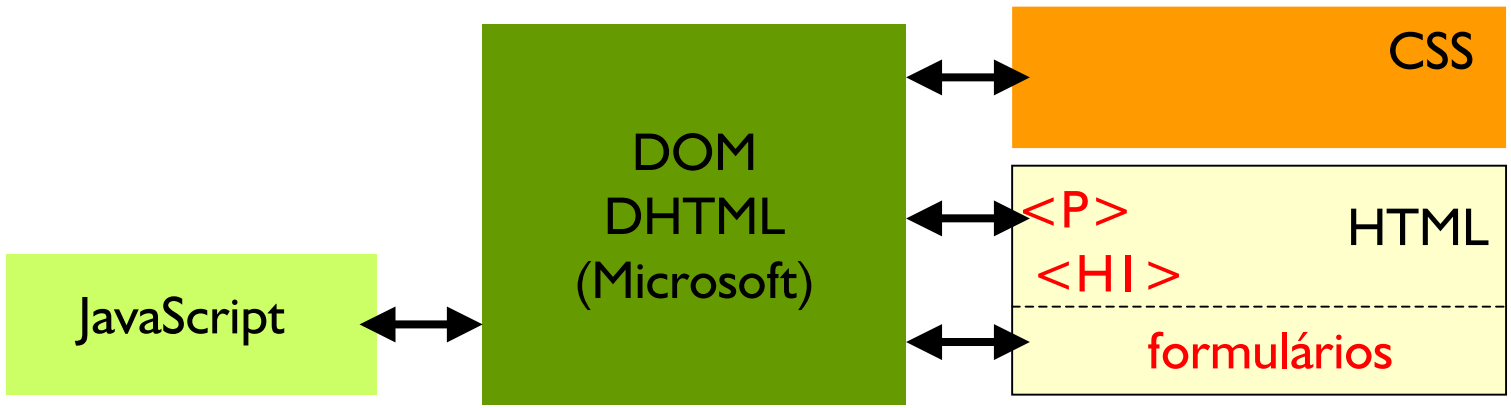
JavaScript e DOM

- O Document Object Model do JavaScript mapeia algumas estruturas do HTML a objetos (variáveis) da linguagem
  - Propriedades dos objetos (e conseqüentemente dos elementos da página) poderão ser alteradas em tempo de execução
  - Mapeamento restringe-se a elementos de formulário, vínculos, imagens e atributos da janela do browser.
  - Permite validação de campos dos formulários, cálculos locais, imagens dinâmicas, abertura de novas janelas, controle de frames, etc.
  - Não é completa. Não mapeia parágrafos, títulos ou folhas de estilo



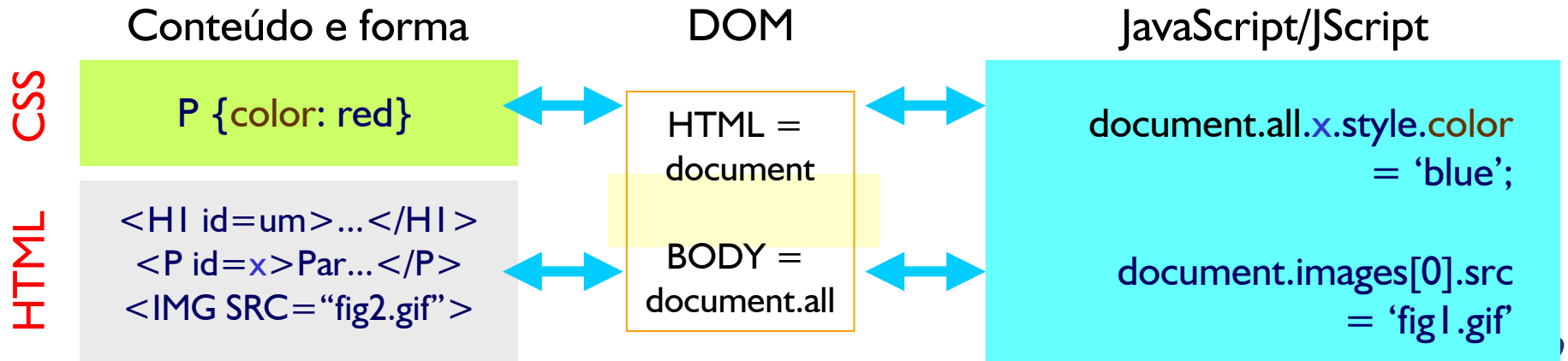
DOM do W3C

- *Document Object Model do W3C*
  - *Mapeia todos os elementos do HTML e folha de estilos, tornando-os acessíveis como objetos JavaScript*
- *Desvantagem: compatibilidade*
  - *A Microsoft utiliza DOM diferente da Netscape*
  - *A W3C tenta padronizar outro*
  - *Na prática W3C DOM funciona bem com XML, mas é problemático com HTML*



DHTML - Dynamic HTML

- *Combinação de uma linguagem de programação (geralmente JavaScript) com HTML e CSS*
  - *Permite tratar eventos em qualquer lugar da página*
  - *Permite grande interatividade*
  - *Permite alteração dinâmica de conteúdo, estrutura e aparência*
- *DOM - Document Object Model é a ponte entre o HTML/CSS e a linguagem baseada em objetos*



Tecnologias lado-servidor

- ***Estendem*** *as funções básicas de servidor HTTP:*
  - *CGI - Common Gateway Interface*
  - *APIs: ISAPI, NSAPI, Apache API, Servlet API, ...*
  - *Scripts: ASP, JSP, LiveWire (SSJS), Cold Fusion, PHP, ...*
- *Rodam do lado do servidor, portanto, não dependem de suporte por parte dos browsers*
  - *Browsers fornecem apenas a interface do usuário*
- *Interceptam o curso normal da comunicação*
  - *Recebem dados via requisições HTTP (GET e POST)*
  - *Devolvem dados através de respostas HTTP*

CGI - Common Gateway Interface

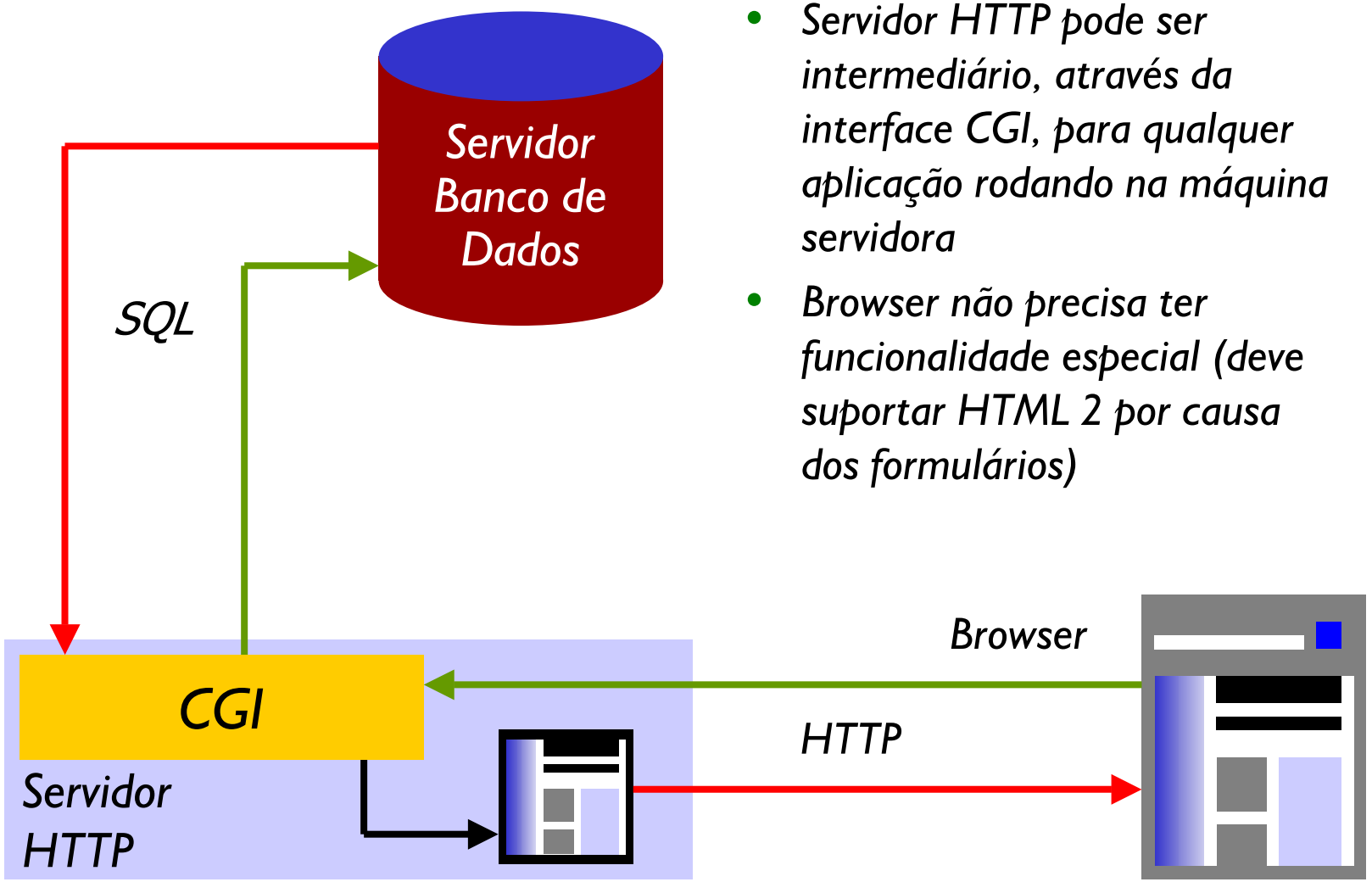
- **Especificação** que determina como construir uma aplicação que será executada pelo servidor Web
- Programas CGI podem ser escritos em qualquer linguagem de programação. A especificação limita-se a determinar os formatos de entrada e saída dos dados (HTTP).
- O que interessa é que o programa seja capaz de
  - Obter dados de entrada a partir de uma requisição HTTP
  - Gerar uma resposta HTTP incluindo os dados e parte do cabeçalho

- Escopo: camada do servidor
  - Não requer quaisquer funções adicionais do cliente ou do HTTP

Aplicações CGI

- Programas CGI podem ser escritos em qualquer linguagem. As linguagens mais populares são C e Perl.
- A linguagem usada deve ter facilidades para
  - Ler variáveis de ambiente (onde o servidor armazena informações passadas no cabeçalho da requisição).
  - Imprimir dados de 8 bits (e não apenas texto ASCII)
- Linguagens não recomendadas
  - Java: dificuldade de ler propriedades do sistema
  - MS-DOS: impossibilidade de gerar HTML
- Segurança depende do servidor e do código

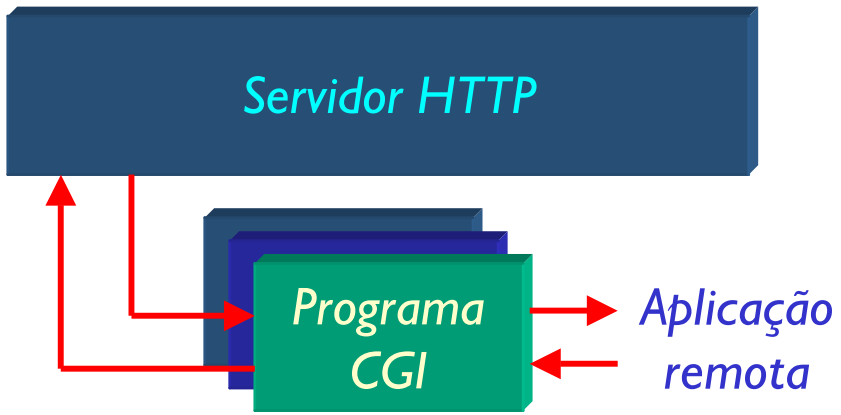
CGI: Gateway para aplicações



- Servidor HTTP pode ser intermediário, através da interface CGI, para qualquer aplicação rodando na máquina servidora
- Browser não precisa ter funcionalidade especial (deve suportar HTML 2 por causa dos formulários)

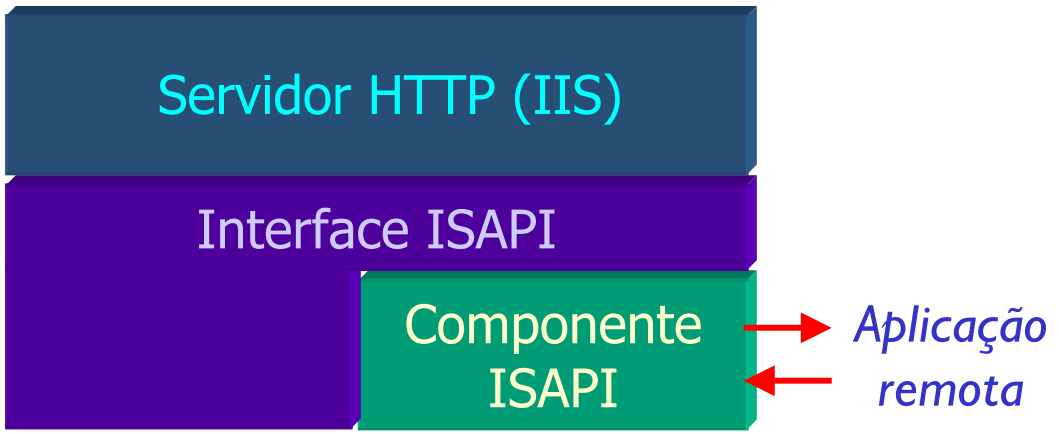
CGI é prático... Mas ineficiente!

- A interface CGI requer que o servidor sempre execute um programa
  - Um novo processo do S.O. rodando o programa CGI é criado para cada cliente remoto que o requisita.
  - Novos processos consomem muitos recursos, portanto, o desempenho do servidor diminui por cliente conectado.
- CGI roda como um processo externo, logo, não tem acesso a recursos do servidor
  - A comunicação com o servidor resume-se à entrada e saída.
  - É difícil o compartilhamento de dados entre processos

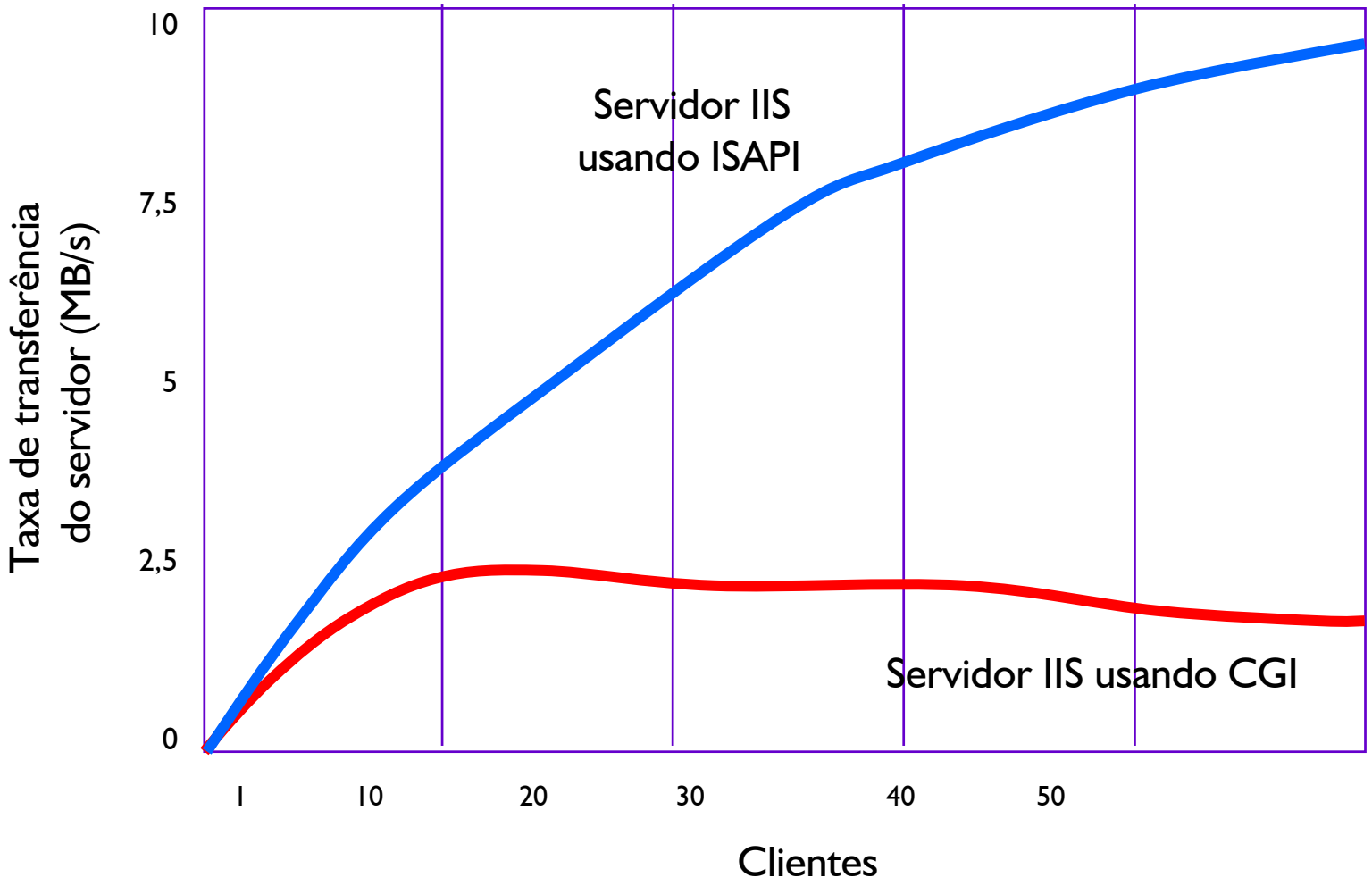


APIs do servidor

- *Podem substituir totalmente o CGI, com vantagens:*
  - *Toda a funcionalidade do servidor pode ser usada*
  - *Múltiplos clientes em processos internos (threads)*
  - *Muito mais rápidas e eficientes (menos overhead)*
- *Desvantagens:*
  - *Em geral dependem de plataforma, fabricante e linguagem*
  - *Soluções proprietárias*
- *Exemplos*
  - *ISAPI (Microsoft)*
  - *NSAPI (Netscape)*
  - *Apache Server API*
  - *??SAPI*



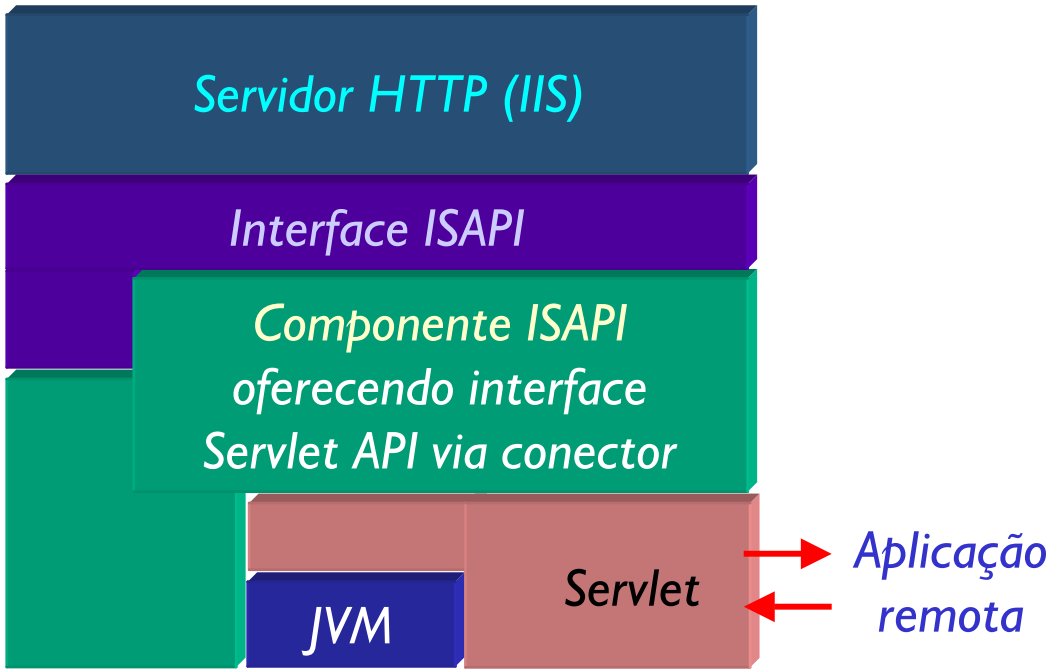
ISAPI vs. CGI



Teste realizado pela PCMagazine Labs com aplicações similares

Servlet API

- *API independente de plataforma e praticamente independente de fabricante*
- *Componentes são escritos em Java e se chamam servlets*
- *Como os componentes SAPI proprietários, rodam dentro do servidor, mas através de uma Máquina Virtual Java*
- *Disponível como 'plug-in' ou conector para servidores que não o suportam diretamente*
  - *Desenho ao lado mostra solução antiga de conexão com IIS*
- *Nativo em servidores Sun, IBM, ...*



Vantagens dos servlets...

- ... sobre CGI
  - Rodam como parte do servidor (cada nova requisição inicia um novo thread mas não um novo processo)
  - Mais integrados ao servidor: mais facilidade para compartilhar informações, recuperar e decodificar dados enviados pelo cliente, etc.
- ... sobre APIs proprietárias
  - Não dependem de único servidor ou sistema operacional
  - Têm toda a API Java à disposição (JDBC, RMI, etc.)
  - Não comprometem a estabilidade do servidor em caso de falha (na pior hipótese, um erro poderia derrubar o JVM)

Problemas dos servlets, CGI e APIs

- *Para gerar páginas dinâmicas (99% das aplicações), é preciso embutir o HTML ou XML dentro de instruções de uma linguagem de programação:*

```
out.print("<h1>Servlet</h1>");
for (int num = 1; num <= 5; i++) {
    out.print("<p>Parágrafo " + num + "</p>");
}
out.print("<table><tr><td> ... </tr></table>");
```

- *Maior parte da informação da página é estática, no entanto, precisa ser embutida no código*
- *Afasta o Web designer do processo*
  - *Muito mais complicado programar que usar HTML e JavaScript*
  - *O design de páginas geradas dinamicamente acaba ficando nas mãos do programador (e não do Web designer)*

Solução: scripts de servidor

- *Coloca a linguagem de programação dentro do HTML (e não o contrário)*

```
<h1>Servlet</h1>
 <% for (int num = 1; num <= 5; i++) { %>
   <p>Parágrafo <%= num %></p>
 <%}%>
 <table><tr><td> ... </tr></table>
```

- *Permite o controle da aparência e estrutura da página em softwares de design (DreamWeaver, FrontPage)*
- *Página fica mais legível*
- *Quando houver muita programação, código pode ser escondido em servlets, JavaBeans, componentes (por exemplo: componentes ActiveX, no caso do ASP)*

Scripts de servidor

- Alguns dos mais populares:
  - Microsoft Active Server Pages (ASP)
  - Sun JavaServer Pages (JSP)
  - Macromedia Cold Fusion
  - PHP
- A página geralmente possui uma extensão de nome de arquivo diferente para que o servidor a identifique como um programa
- As página ASP, PHP, JSP, etc. são processadas e os roteiros são executados pelo servidor, que os consome
  - No browser, chega apenas a saída do programa: página HTML
  - Comandos <% .. %> ou similares nunca chegam no browser
  - Servidor envia cabeçalho Content-type: text/html (default) ou algum outro tipo texto (text/xml, text/plain)

Controle de sessão

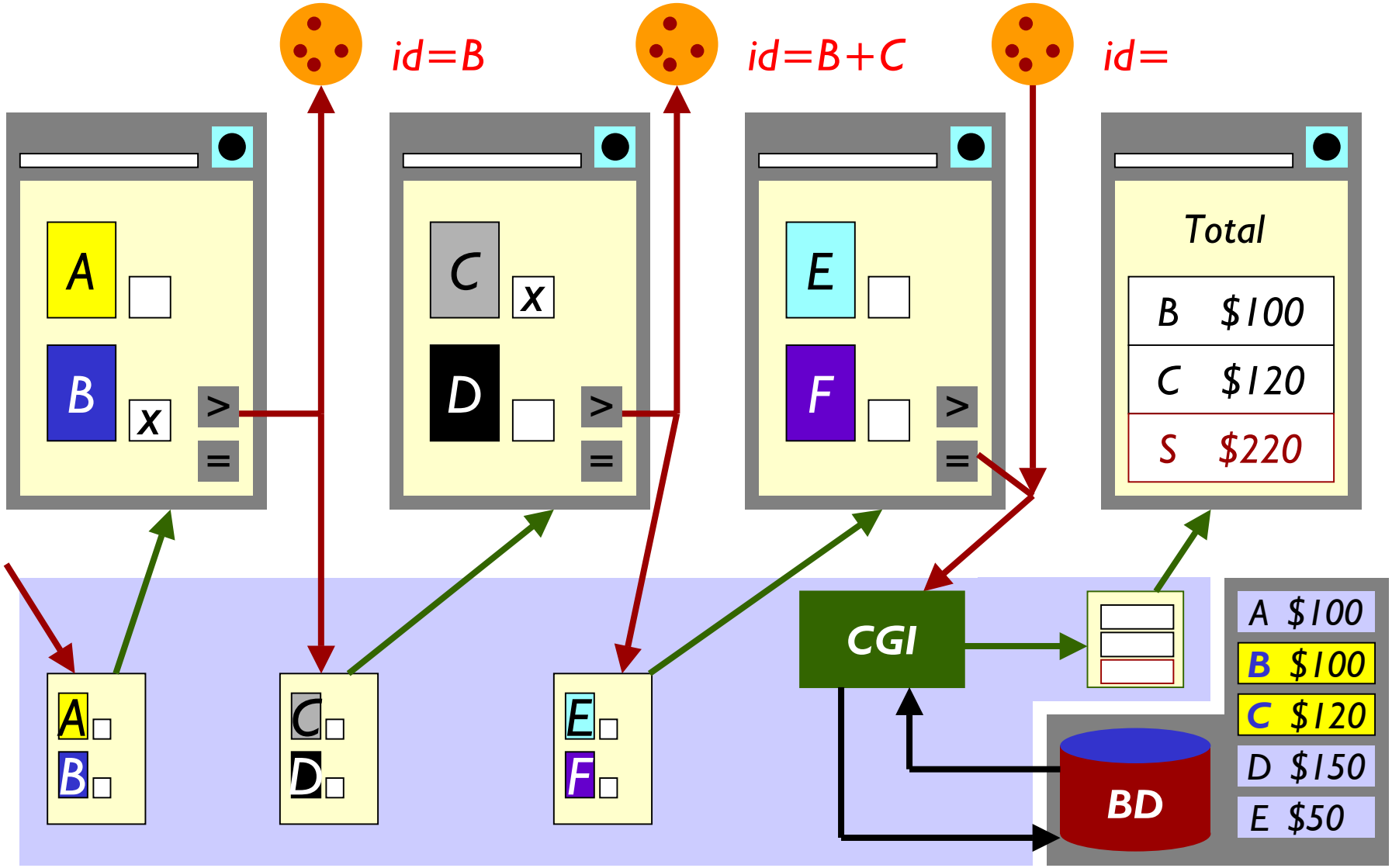
- *HTTP não preserva o estado de uma sessão. É preciso usar mecanismos artificiais com CGI (ou qualquer outra tecnologia Web)*
  - *Seqüência de páginas/aplicações: desvantagens: seqüência não pode ser quebrada; mesmo que página só contenha HTML simples, precisará ser gerada por aplicação*
  - *Inclusão de dados na URL: desvantagens: pouca flexibilidade e exposição de informações*
  - *Cookies (informação armazenada no cliente): desvantagens: espaço e quantidade de dados reduzidos; browser precisa suportar a tecnologia*

Cookies

- *Padrão Internet (RFC) para persistência de informações entre requisições HTTP*
- *Um cookie é uma pequena quantidade de informação que o servidor armazena no cliente*
  - *Par nome=valor. Exemplos: usuario=paulo, num=123*
  - *Escopo no servidor: domínio e caminho da página*
  - *Pode ser seguro*
  - *Escopo no cliente: browser (sessão)*
  - *Duração: uma sessão ou tempo determinado (cookies persistentes)*
- *Cookies são criados através de cabeçalhos HTTP*

```
Content-type: text/html
Content-length: 34432
Set-Cookie: usuario=ax343
Set-Cookie: lastlogin=12%2610%2699
```

Exemplo com cookies: Loja virtual



> Guarda cookie e chama próxima página

= Lê cookie, apaga-o e envia dados para CGI

Exercícios

- *1. Conecte-se via Telnet na porta HTTP de um servidor conhecido. Ex: telnet servidor 80*
- *2. Envie o comando GET abaixo, digite <ENTER> duas vezes e veja o resultado*

```
GET / HTTP/1.0
```

- *3. Envie o comando POST abaixo. Observe o momento do envio*

```
POST /servlet/TestPost HTTP/1.0
Content-type: text/x-www-form-urlencoded
Content-length: 10

abcde01234
```

- *4. Rode o programa SendGet.class*
- *3. Execute o programa Listen.class (veja cap01/) em uma janela. Ele escuta a porta 8088. Conecte-se via browser à porta 8088 usando http://localhost:8088*
  - *O programa imprime a requisição recebida pelo browser*

Aplicaçoes Web e Java

- *Servlets e JavaServer Pages (JSP) são as soluções Java para estender o servidor HTTP*
  - *Suportam os métodos de requisição padrão HTTP (GET, POST, HEAD,PUT, DELETE, OPTIONS, TRACE)*
  - *Geram respostas compatíveis com HTTP (códigos de status, cabeçalhos RFC 822)*
  - *Interagem com Cookies*
- *Além dessas tarefas básicas, também*
  - *Suportam filtros, que podem ser chamados em cascata para tratamento de dados durante a transferência*
  - *Suportam controle de sessão transparentemente através de cookies ou rescrita de URLs (automática)*
- *É preciso usar um servidor que suporte as especificações de servlets e JSP*

Exemplo de um servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class SimpleServlet extends HttpServlet {
    public void doGet (HttpServletRequest  request,
                       HttpServletResponse response)
                          throws ServletException, IOException {
       PrintWriter  out;
       response.setContentType("text/html");
       out = response.getWriter();
       String user = request.getParameter("usuario");
       if (user == null)
           user = "World";

       out.println("<HTML><HEAD><TITLE>");
       out.println("Simple Servlet Output");
       out.println("</TITLE></HEAD><BODY>");
       out.println("<H1>Simple Servlet Output</H1>");
       out.println("<P>Hello, " + user);
       out.println("</BODY></HTML>");
       out.close();
    }
}
```

Exemplo de um JSP equivalente

```
<HTML><HEAD>
<TITLE>Simple Servlet Output</TITLE>
</HEAD><BODY>
<%
      String user =
         request.getParameter("usuario");
      if (user == null)
          user = "World";
%>
<H1>Simple Servlet Output</H1>
<P>Hello, <%= user %>
</BODY></HTML>
```

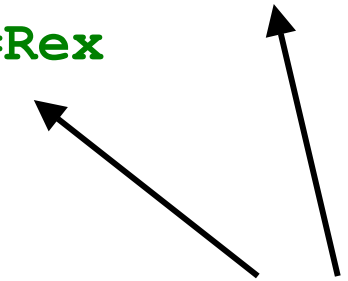
Página recebida no browser

- *Url da requisição*

```
http://servidor/servlet/SimpleServlet?usuario=Rex
http://servidor/hello.jsp?usuario=Rex
```

- *Código fonte visto no cliente*

```
<HTML><HEAD>
<TITLE>
Simple Servlet Output
</TITLE>
</HEAD><BODY>
<H1>Simple Servlet Output</H1>
<P>Hello, Rex
</BODY></HTML>
```



Usando contexto default ROOT no TOMCAT

Um simples JavaBean

```
package beans;

public class HelloBean implements
                        java.io.Serializable {

      private String msg;

      public HelloBean() {
            this.msg = "World";
      }

      public String getMensagem() {
            return msg;
      }

      public void setMensagem(String msg) {
            this.msg = msg;
      }
}
```

JSP usando JavaBeans

- *Página JSP que usa HelloBean.class*

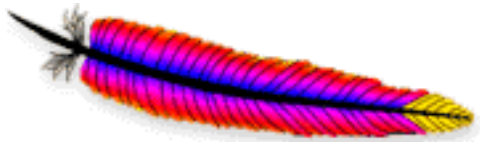
```
<HTML><HEAD>
<jsp:useBean id="hello" class="beans.HelloBean" />
<jsp:setProperty name="hello" property="mensagem"
                 param="usuario" />
<TITLE>
Simple Servlet Output
</TITLE>
</HEAD><BODY>
<H1>Simple Servlet Output</H1>
<P>Hello, <jsp:getProperty name="hello"
                           property="mensagem" />
</BODY></HTML>
```

Como executar servlets e JSP

- Para executar servlets e JSP é preciso implantá-los em um Web Container
- Um Web Container pode estar executando como parte de um servidor HTTP que o repassa as requisições destinadas a servlets e JSP
- Neste curso, usaremos o Tomcat Web Container, que pode tanto funcionar conectado a outro servidor como usar seu próprio servidor Web
- O Tomcat ocupará a porta 8080

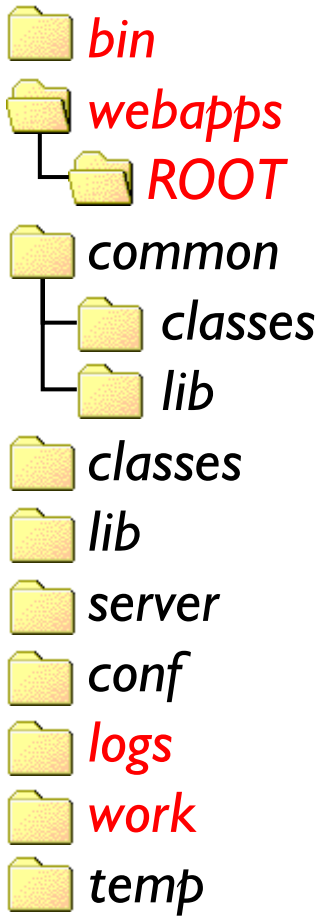
Jakarta Tomcat

- O Apache Jakarta Tomcat é a implementação de referência para aplicações Web
  - Tomcat 3.x - I.R. para servlets 2.2 e JSP 1.1
  - Tomcat 4.x - I.R. para servlets 2.3 e JSP 1.2
- Em produção, geralmente é acoplado a um servidor de páginas estáticas eficiente (Apache, ou outro)
- Em desenvolvimento, pode-se usar o servidor distribuído com o Tomcat
- Instale o Tomcat
  - Use c:\tomcat-4.0 ou /usr/jakarta/tomcat e defina as variáveis CATALINA_HOME e TOMCAT_HOME apontando para o mesmo lugar



The Jakarta Project
http://jakarta.apache.org

Estrutura do Tomcat



bin	Executáveis. Para iniciar o Tomcat, rode startup.bat
webapps	Contém pastas de contextos (aplicações Web)
ROOT	Contexto raiz (coloque suas páginas Web aqui)
common	Classpaths do Tomcat (valem para todas as aplicações)
classes	Classpath (coloque classes aqui)
lib	Classpath de JARs (o servlet.jar está aqui)
classes	Classpath do servidor (não use)
lib	Classpath do servidor para JARs (não use)
server	Executáveis do Tomcat
conf	Arquivos de configuração (server.xml e outros)
logs	Logs para todas as aplicações
work	Contém servlets gerados a partir de JSPs
temp	Diretório temporário

- $TOMCAT_HOME/conf/server.xml: configuração do servidor (onde se pode configurar novos contextos)
- $TOMCAT_HOME/common/lib/*.jar: Classpath para todas as aplicações que rodam no container (use com cuidado para evitar conflitos)

Como iniciar e parar o Tomcat

- No diretório bin/ há vários arquivos executáveis
- Para iniciar o Tomcat, use, a partir do bin/
  - ./startup.sh ou
  - startup.bat
- Para encerrar o Tomcat, use
  - ./shutdown.sh ou
  - shutdown.bat
- Crie atalhos na sua área de trabalho para esses arquivos ou para o diretório bin (se eles já não existirem). Eles serão usados freqüentemente
- Refira-se aos arquivos no diretório logs/ para realizar depuração de suas aplicações. Crie um atalho para ele também

Como implantar uma aplicação no Tomcat

- Há três maneiras
  - Transferir os arquivos da aplicação (JSP, servlets) para contextos já reconhecidos pelo servidor
  - Configurar o servidor para que reconheça um novo contexto onde os arquivos da aplicação residem (server.xml)
  - Implantar a aplicação como um WebArchive (WAR)
- Contextos são diretórios devidamente configurados que o Tomcat reconhece como aplicações Web
- O contexto raiz chama-se ROOT.
  - Arquivos copiados para `$TOMCAT_HOME/webapps/ROOT/` podem ser acessados via `http://servidor:8080/`
  - Servlets em `webapps/ROOT/WEB-INF/classes` podem ser acessados via `http://servidor:8080/servlet/`

Outros contextos existentes

- Os exemplos do Tomcat rodam em um contexto diferente de ROOT: no contexto /examples/
- Para usar /examples:
  - Coloque páginas Web, JSPs, imagens, etc. em `$TOMCAT_HOME/webapps/examples/`
  - Coloque beans, classes e servlets em `$TOMCAT_HOME/webapps/examples/WEB-INF/classes/`
- Acesse as páginas e JSP usando:
  - http://servidor/examples/pagina.html
- Acesse os servlets usando
  - http://servidor/examples/servlet/pacote.Classe
- Não precisa reiniciar o servidor

Exercícios

- 1. Copie o arquivo hello.jsp para o contexto ROOT
  - a) Copie para `$CATALINA_HOME/webapps/ROOT` (`%CATALINA_HOME%` aponta para `c:\tomcat-4.0` na nossa instalação Windows)
  - b) Acesse via `http://localhost:8080/hello.jsp`
- 2. Implante o SimpleServlet no servidor
  - a) Compile usando o servlet.jar encontrável em common/lib e copie para webapps/ROOT/WEB-INF/classes
  - b) Se classes não existir, crie o diretório e reinicie o servidor

```
javac -d $CATALINA_HOME/webapps/ROOT/WEB-INF/classes
      -classpath  $CATALINA_HOME/common/lib/servlet.jar
      SimpleServlet.java
```

  - c) Acesse via `http://localhost:8080/servlet/SimpleServlet`

Exercícios (2)

- 3. Implante a aplicação hellobean.jsp. Ela tem duas partes: uma página JSP e uma classe Java (JavaBean)
  - a) Compile o JavaBean (observe que ele está em um pacote) e copie-o para o `webapps/ROOT/WEB-INF/classes`
    `src/` contém diretório `beans/` (pacote), que contém `HelloBean.java`

```
> cd src
> javac -d $CATALINA_HOME/webapps/ROOT/WEB-INF/classes
        beans/HelloBean.java
```

  - b) Copie `hellobean.jsp` para `webapps/ROOT`
  - c) Acesse via `http://localhost:8080/hellobean.jsp`

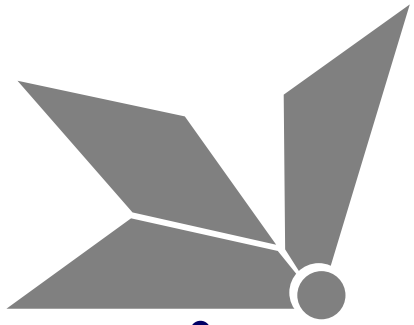
Arquiteturas de aplicações Web

- Grandes aplicações Web geralmente consistem de várias páginas JSP, HTML, imagens misturadas com classes Java comuns e servlets
- Procura-se separar responsabilidades
  - Controle de requisição, resposta, repasse de dados
  - Processamento de lógica de negócio
  - Processamento de resposta e geração de páginas
- Aplicações que dividem-se em camadas de acordo com as responsabilidades acima são aplicações MVC (Model-View-Controller)
  - Mais fáceis de manter e de reutilizar por equipes heterogêneas (web designers, programadores, etc.)

J2EE

- Java 2 Enterprise Edition é uma especificação que inclui JSP e servlets
- J2EE define uma arquitetura em camadas independentes formadas por componentes reutilizáveis
  - Páginas HTML ou outras tecnologias no cliente
  - Servlets e JSP na camada do servidor Web
  - Enterprise JavaBeans na camada de negócios
  - Conectores na camada de integração
  - Sistemas de informação na camada de dados
- Servlets e JSP podem ser usados em servidores J2EE
  - Precisam aderir a novas restrições do ambiente

helder@acm.org



argonavis.com.br