PROMPT 2011
Módulo 3 – Aplicações Web
ASP.NET HTTP Pipeline

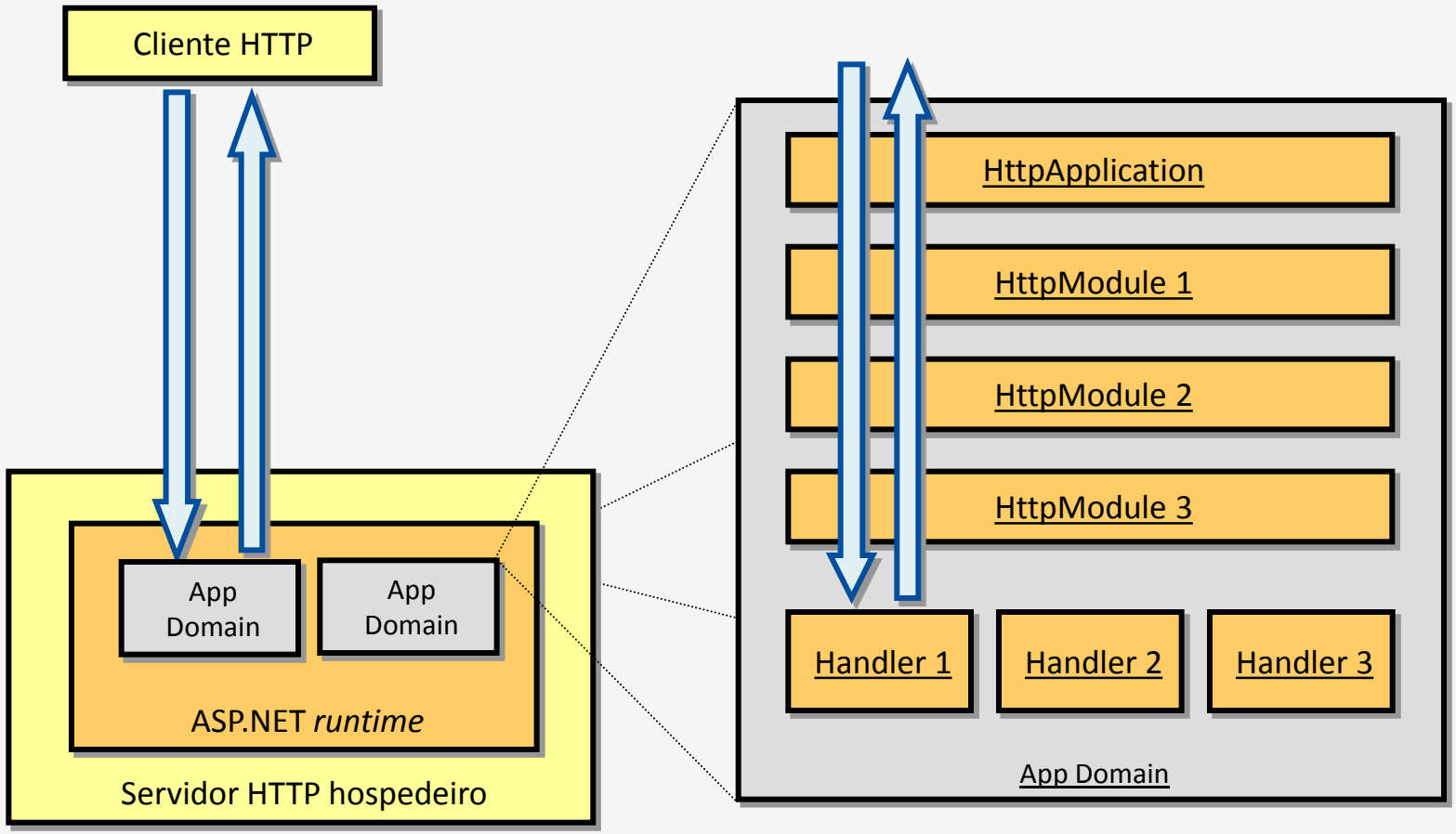
Centro de Cálculo
Instituto Superior de Engenharia de Lisboa

Luis Falcão (lfalcao@cc.isel.ipl.pt)
João Trindade (jtrindade@cc.isel.ipl.pt)

Agenda

- *HTTP Pipeline*
  - Pontos de extensibilidade
    - HttpHandler
    - HttpApplication
    - HttpModule
  - Gestão de estado
  - Aspectos de implementação

HTTP Pipeline – Pontos de extensibilidade



Custom Handlers

- Para definição de novos *endpoints* no atendimento de pedidos HTTP
- Definidos com classes que implementam a interface **IHttpHandler**

```
public interface System.Web.IHttpHandler
{
    void ProcessRequest(HttpContext context);
    bool IsReusable { get; }
}
```

- Associação URL → *handler* especificada via ficheiros de configuração
  - Implica a configuração do hospedeiro da infra-estrutura ASP.NET

- São, em alternativa, definidos através de ficheiros ***.ashx**
  - Não implica a configuração do hospedeiro (caso seja o IIS)

demo

`CalcHandler`

Demo: CalcHandler

calcHandler.cs

```
using System.Web;
public class CalcHandler : IHttpHandler {
  public void ProcessRequest(HttpContext context) {
    context.ContentType = “text/plain”;
    int a = int.Parse(context.Request["a"]);
     int b = int.Parse(context.Request["b"]);
     switch(context.Request ["op"]) {
       case "add":      context.Response.Write(a + b); break;
       case "subtract": context.Response.Write(a - b); break;
       case "multiply": context.Response.Write(a * b); break;
       default:
         context.Response.Write("Unrecognized operation");
    break;
     }
  }
  // ...
}
```

web.config

```
<configuration><system.web>
  <httpHandlers >
    <add verb="GET" path="calculadora.calc" type="CalcHandler" />
  </httpHandlers>
</configuration></system.web>
```

Exemplo:

http://localhost/site1/**calculadora.calc?a=2&b=4&op=add**

Demo: Ficheiros *.ASHX

calculadora.ashx

```
<!– file: calc.ashx –>
<%@ WebHandler language="C#" class="CalcHandler" %>

using System.Web;
public class CalcHandler : IHttpHandler {
  public void ProcessRequest(HttpContext context) {
    int a = int.Parse(context.Request["a"]);
    int b = int.Parse(context.Request["b"]);
    switch(context.Request ["op"]) {
      case "add":      context.Response.Write(a + b); break;
      case "subtract": context.Response.Write(a - b); break;
      case "multiply": context.Response.Write(a * b); break;
      default:
        context.Response.Write("Unrecognized operation");
    break;
    }
  }
  // ...
}
```

NOTA: Não é preciso configurar este *handler* no ficheiro de configuração da aplicação **web.config**

Exemplo:
http://localhost/site1/**calculadora.ashx**?a=2&b=4&op=add

demo

Demo2

- `EchoHandler`
- `WaterMarkHandler`

CENTRO DE CÁLCULO
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

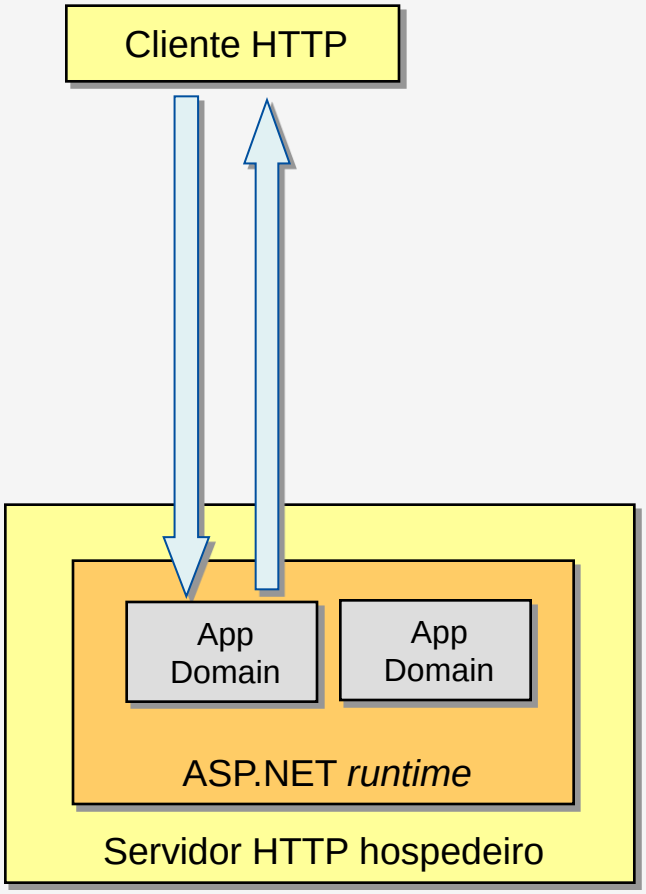
HttpHandler Factory

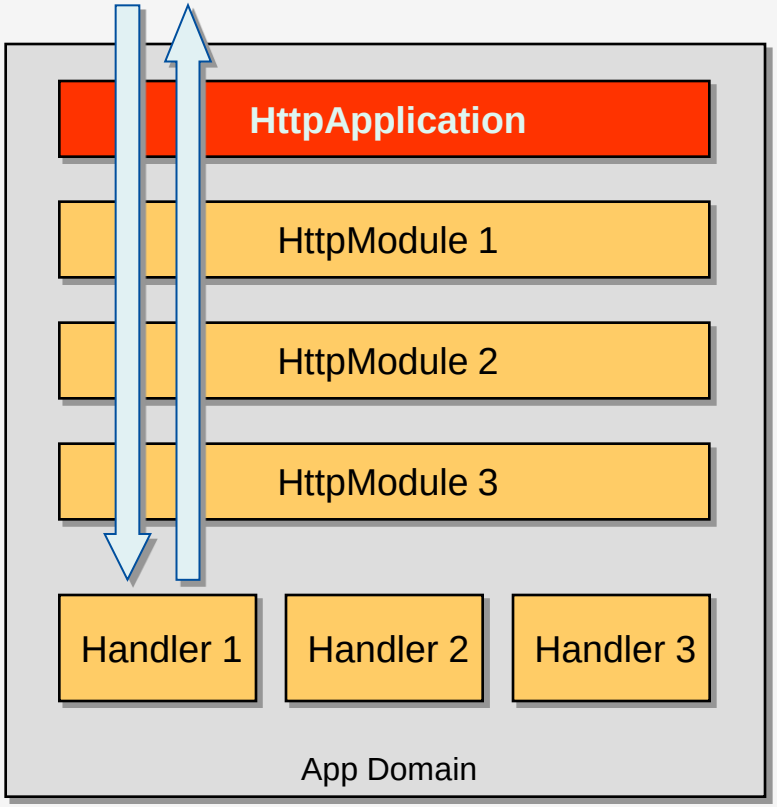
```
public interface IHttpHandlerFactory
{
  IHttpHandler GetHandler(HttpContext ctx, string requestType,
  string url, string translatedPath);
  void ReleaseHandler(IHttpHandler handler);
}
```

- Fábrica de instâncias de HTTPHandler
  - Maior controlo sobre a criação e destruição das instâncias dos *handlers*
- É utilizada quando definida no web.config, em vez do *handler*

```
<configuration><system.web>
  <httpHandlers >
    <add verb="GET" path="calculadora.calc" type="CalcHandler" />
    <add verb="GET" path="calculadora.calc" type="CalcHandlerFactory" />
  </httpHandlers>
</configuration></system.web>
```

HTTP Pipeline – Pontos de extensibilidade





Custom HttpApplication

- Para definição do primeiro nó da cadeia
- Definido através do ficheiro **Global.asax**

```
<%@ Application Language="C#" %>
<script runat="server">
    void Application_Start(object sender, EventArgs e) { ... }
    void Application_End(object sender, EventArgs e) { ... }
    void Application_Error(object sender, EventArgs e) { ... }
    // Outros campos, propriedades ou métodos
    public override void Init() { ... }
</script>
```

- A classe produzida deriva de **System.Web.HttpApplication**
- Oportunidade de captura de **eventos globais**
  - De ciclo de vida
    - Início, terminação ordeira, excepção não tratada, etc...
  - De atendimento de pedido

Atendimento de pedido

sequência de acções e eventos

1. Recepção de pedido
2. Autenticação (durante e após) e Autorização (durante e após)
3. Verificação de existência de resposta em *cache* (durante e após)
4. Resolução URL → handler (após)
5. Aquisição de estado de conversação (durante e após)
6. Execução de *handler* para produção de resposta (antes e após)
7. Actualização de estado de conversação (durante e após)
8. Actualização da *cache* com resposta, caso se aplique (durante e após)
9. Fim de atendimento
10. Envio de resposta para cliente (antes de cabeçalho e de conteúdo)

Atendimento de pedido (2)

1. BeginRequest
2. AuthenticateRequest
3. PostAuthenticateRequest
4. AuthorizeRequest
5. PostAuthorizeRequest
6. ResolveRequestCache
7. PostResolveRequestCache

   After the **PostResolveRequestCache** event and before the **PostMapRequestHandler** event, an event handler (a page corresponding to the request URL) is created.
8. PostMapRequestHandler
9. AcquireRequestState
10. PostAcquireRequestState
11. PreRequestHandlerExecute

    The event handler is executed.
12. PostRequestHandlerExecute
13. ReleaseRequestState
14. PostReleaseRequestState

    After the **PostReleaseRequestState** event, response filters, if any, filter the output.
15. UpdateRequestCache
16. PostUpdateRequestCache
17. EndRequest

Fonte: http://msdn2.microsoft.com/library/system.web.httpapplication.aspx

demo

Demo3

"Calcular o tempo de processamento de um pedido em HttpApplication"

CENTRO DE CÁLCULO
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Eventos implícitos de *HttpApplication*

- A maioria dos eventos de aplicação ocorrem em cada pedido, excepto Error e Disposed
- Para adicionar um *handler* de evento
  - Registar *delegate* no evento
  - Definir método com a assinatura

```
void Application_event(object src, EventArgs args)
```

- Existem acontecimentos em HttpApplication que não são disponibilizados na forma de eventos da classe. Para adicionar *handler*:
  - Definir método cujos nomes correspondem a eventos, uma vez que estes são chamados por outras classes do *pipeline*

Event	Reason for Firing
Application_Start	Application starting
Application_End	Application ending
Session_Start	User session begins
Session_End	User session ends

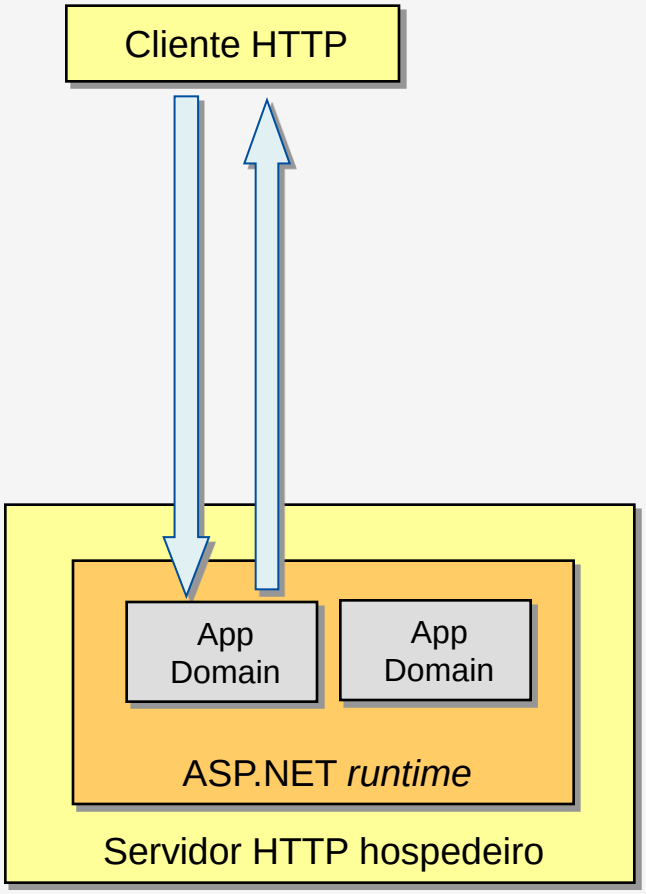
Eventos de HTTPApplication:
http://msdn2.microsoft.com/en-us/library/0dbhtdck.aspx

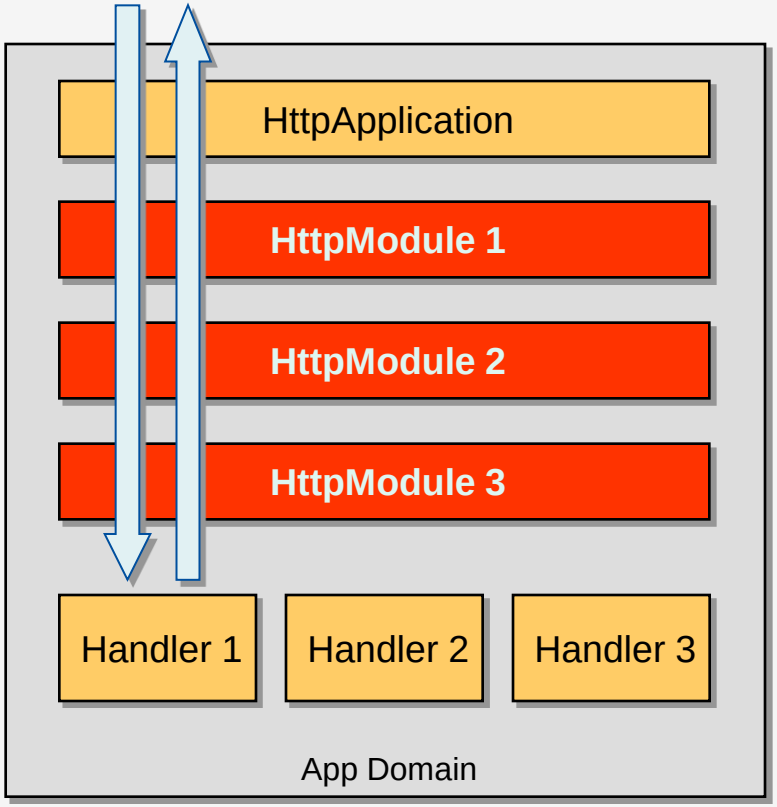
Outros Membros de *HttpApplication*

```
public class HttpApplication : IHttpAsyncHandler, IComponent
{
  // Properties
  public HttpApplicationState Application {get;}
  public HttpContext          Context     {get;}
  public HttpModuleCollection Modules     {get;}
  public HttpRequest          Request     {get;}
  public HttpResponse         Response    {get;}
  public HttpServerUtility    Server      {get;}
  public HttpSessionState     Session     {get;}
  public IPrincipal           User        {get;}

  // Methods
   // Called after modules initialization
  public virtual void Init();
   // To preemptively terminate the Request
  public void CompleteRequest();
    // ...
}
```

HTTP Pipeline – Pontos de extensibilidade





Custom Modules

- Para definição de novos nós da cadeia de atendimento de pedidos HTTP
- Definidos com classes que implementam a interface **IHttpModule**

```
public interface System.Web.IHttpModule
{
    void Init(HttpApplication application);
    void Dispose();
}
```

- Oportunidade para
  - Captura de eventos globais
    - Através do registo na instância de **HttpApplication**
  - Pré processamento de pedidos e pós processamento de respostas
- Cadeia especificada através de ficheiros de configuração

Custom Modules (2)

- A infra-estrutura ASP.NET utiliza módulos para:
  - Autenticação
  - Autorização
  - *Cache*
  - Manutenção de estado de sessão *out-of-process*

Módulos definidos pela plataforma ASP.NET

Module	Purpose
`OutputCacheModule`	Page-level output caching
`SessionStateModule`	Out-of-process session state management
`WindowsAuthenticationModule`	Client authentication using integrated Windows authentication
`FormsAuthenticationModule`	Client authentication using cookie-based forms authentication
`PassportAuthenticationModule`	Client authentication using MS Passport
`UrlAuthorizationModule`	Client authorization based on requested URL
`FileAuthorizationModule`	Client authorization based on requested file

Construção de um *Custom Module*

- Criar classe que implementa IHttpModule
  - Init() – Registar *delegates* em eventos de HttpApplication
  - Dispose() – Realizar processamento de terminação do módulo (caso exista)

```
public interface IHttpModule
{
  // Chamado quando o módulo é criado. Recebe como parâmetro uma
  // referência para a instância actual de HttpApplication
  void Init(HttpApplication context);
  // Chamado quando a aplicação está a ser terminada
  void Dispose();
}
```

- Registar o módulo no *Web.config*

```
<configuration><system.web>
  <httpModules>
    <add name="someKey" type="TimerModule, TimerModuleAssembly" />
  </httpModules>
</configuration></system.web>
```

demo

Demo4

"Calcular o tempo de processamento de um pedido num módulo"

"ApplicationOffline"

CENTRO DE CÁLCULO
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Características dos módulos

- Pooling
  - Os módulos são sempre reutilizados (contrariamente aos *handlers* cuja reutilização é controlada pela propriedade IsReusable). Por cada instância de HttpApplication criada, são criados os respectivos módulos dessa aplicação
  - **Não guardar estado em campos de instância dos módulos entre pedidos**
- Módulos *vs* Global.asax

Feature	Module	global.asax
Can receive event notifications for all HttpApplication-generated events	Yes	Yes
Can receive event notifications for Session_Start/_End, Application_Start/_End	No	Yes
Can be deployed at the machine level	Yes	No

A "cola do *pipeline*": *HTTPContext*

- Uma instância de HTTPContext flúi por todo o *pipeline*, contendo toda a informação sobre cada pedido

- É parâmetro de vários métodos, incluindo o método `IHttpHandler.ProcessRequest()`

- Acessível em todo o pipeline e através das propriedades `Page.Context, HttpApplication.Context` ou `HttpContext.Current`

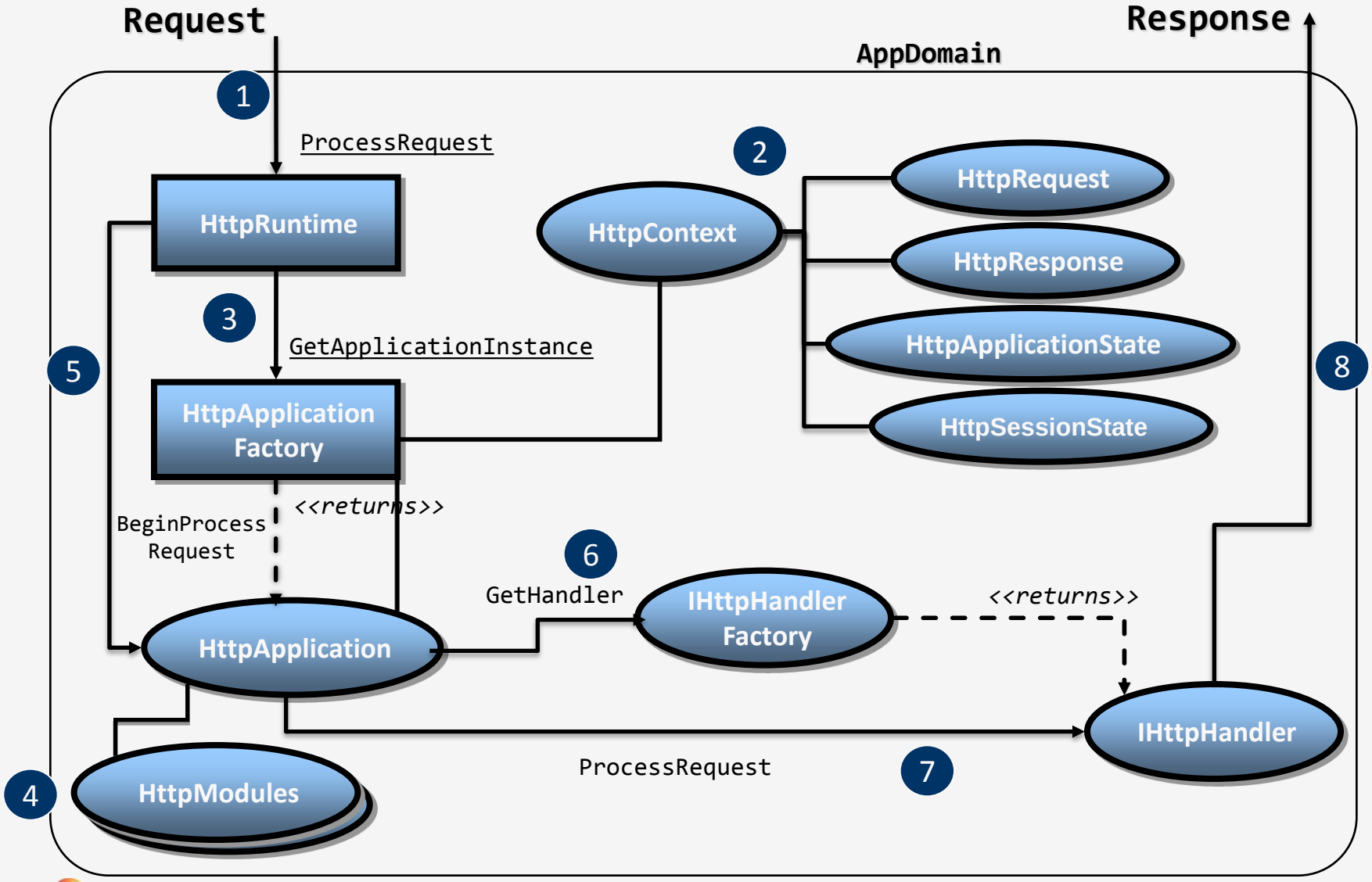
- Deve ser utilizado como o repositório por excelência de dados relativos aos pedidos, pois é acessível aos intervenientes no HTTP Pipeline

Membros de *HTTPContext*

Nome	Tipo	Descrição
Current (static)	HttpContext	Context for the request currently in progress
Application	HttpApplicationState	Application-wide property bag
ApplicationInstance	HttpApplication	Active application instance
Session	HttpSessionState	Per-client session state
Request	HttpRequest	HTTP request object
Response	HttpResponse	HTTP response object
User	IPrincipal	Security ID of the caller
Handler	IHttpHandler	Handler for the request
Items	IDictionary	Per-request property bag
Server	HttpServerUtility	HTTP server object
Error	Exception	Unhandled exception object
Cache	Cache	Application-wide cache
Trace	TraceContext	Trace class for diagnostic output
TraceIsEnabled	Boolean	Whether tracing is currently enabled
WorkerRequest	HttpWorkerRequest	The current worker request object
IsCustomErrorEnabled	Boolean	Whether custom error pages are currently enabled
IsDebuggingEnabled	Boolean	Whether the current request is in debug mode
IsInCancellablePeriod	Boolean	Whether the current request can still be cancelled

HTTP Pipeline – Sequência de acções

(modelo)



Agenda

- *HTTP Pipeline*
  - Pontos de extensibilidade
    - `HttpHandler`
    - `HttpApplication`
    - `HttpModules`
  - Gestão de estado
  - Aspectos de implementação

Gestão de estado

- Estado de aplicação
  - Utilização recomendada: *read-only*
  - Consequências de utilização indevida: redução de escalabilidade
- Estado de conversação (sessão)
  - Suportado pelo módulo **SessionStateModule**
  - Realização condicional da aquisição e libertação de estado de conversação
    - **IRequiresSessionState**
    - **IReadOnlySessionState**
  - Alvo de reestruturação na versão 2.0
    - Arquitectura extensível (SPI)
    - Auto-detecção de existência de suporte para *cookies* no cliente

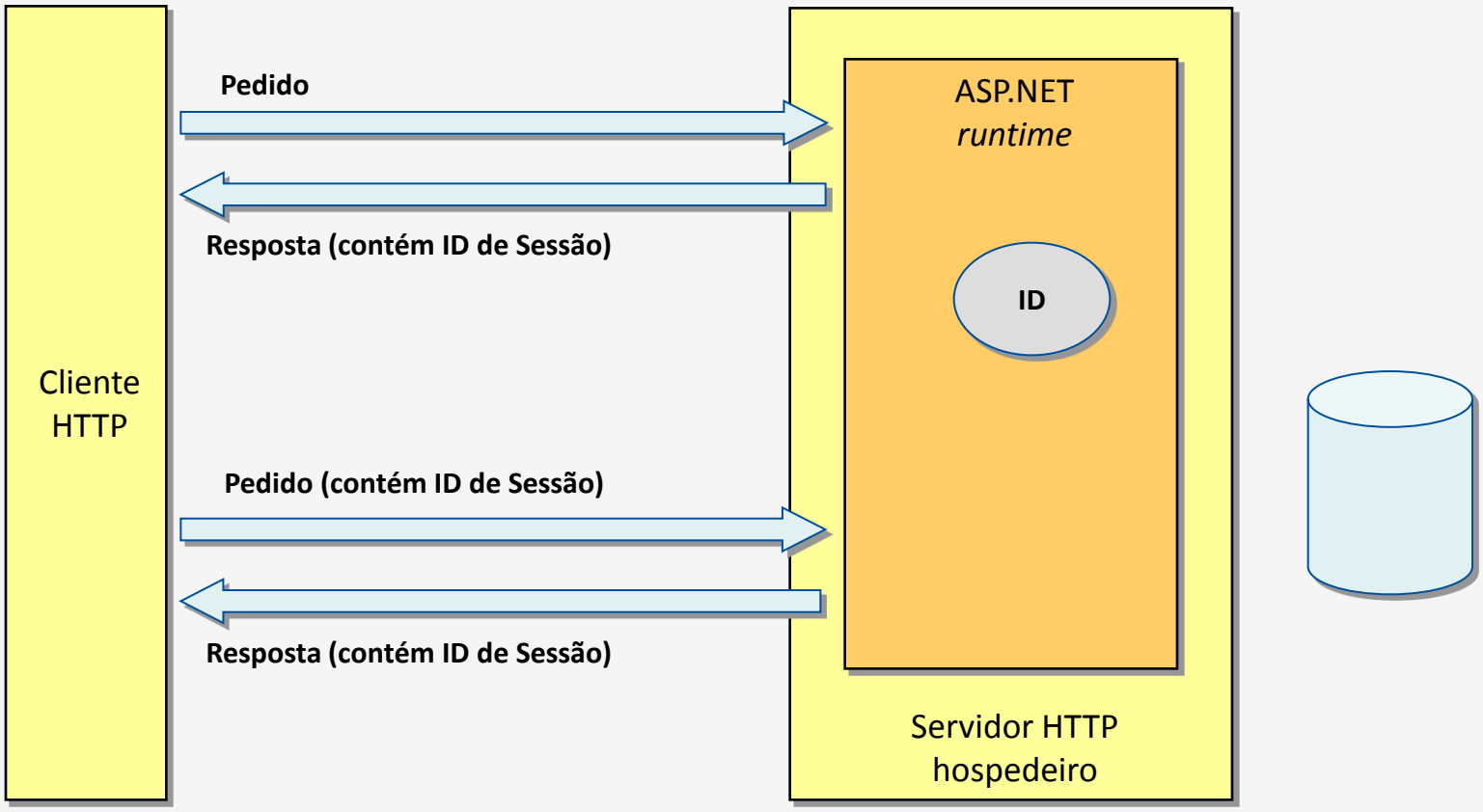
demo

Demo5

StateManagement

CENTRO DE CÁLCULO
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

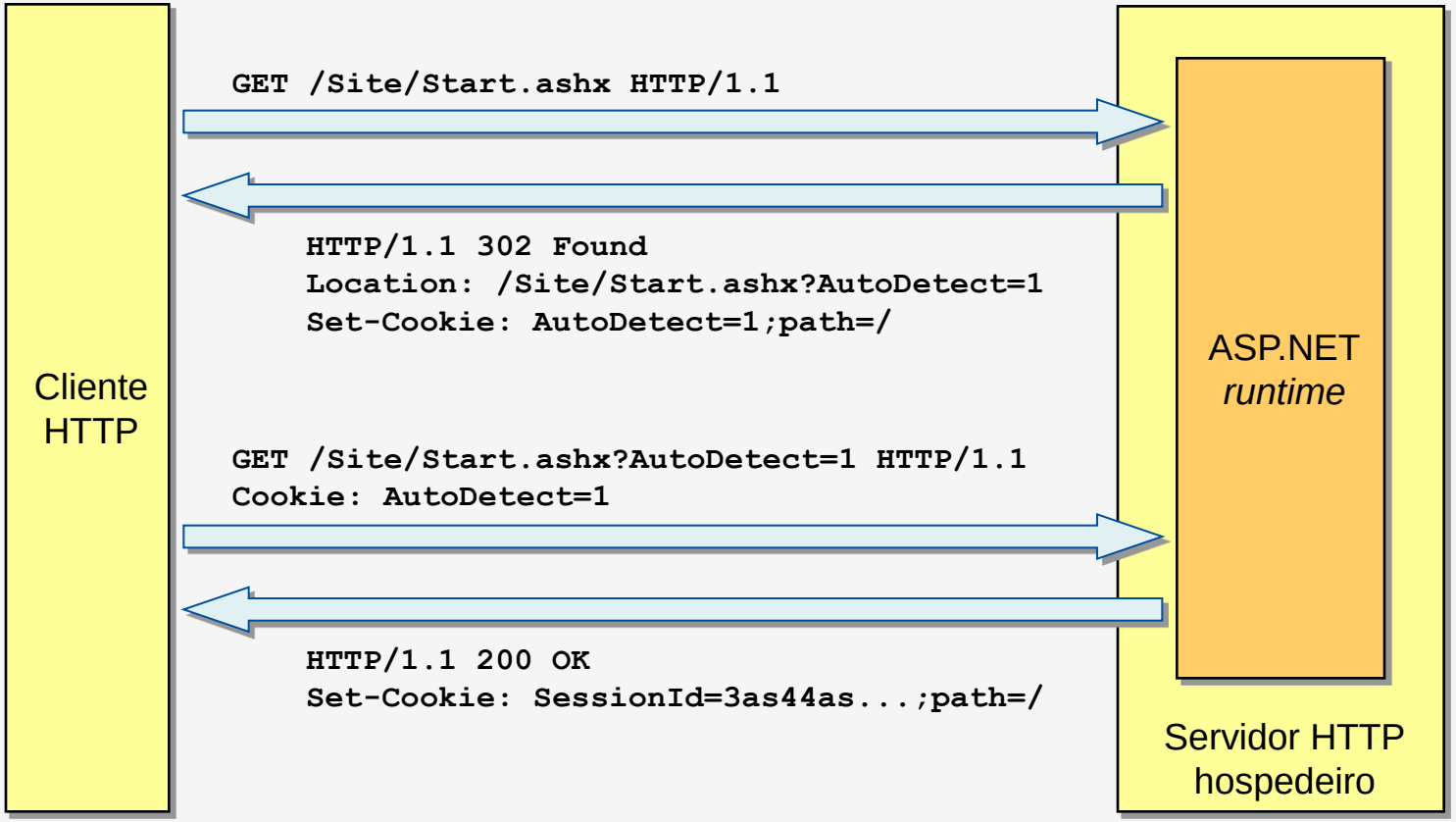
Estado de conversação sobre HTTP



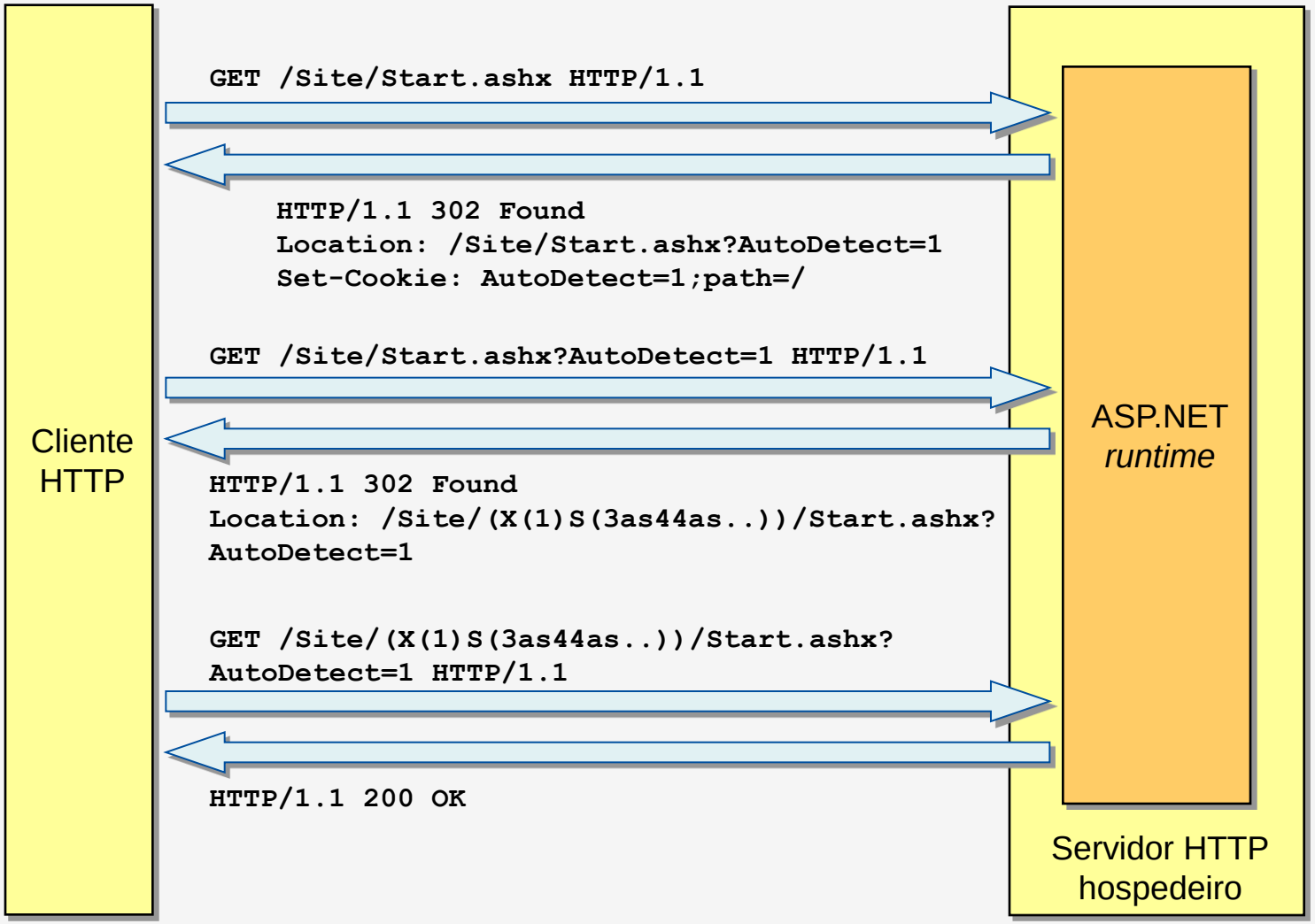
HTTP Pipeline – aspectos de implementação

- Maximização do paralelismo através da redução da partilha de instâncias
  - A cada pedido é atribuída uma nova cadeia de atendimento, com início em **HttpApplication**
- Técnicas usadas:
  - *Instance pooling*
    - Conjunto de instâncias equivalentes
    - Dimensão do conjunto <= dimensão do *ThreadPool*
  - *Caching*
- Onde são usadas?
  - *Instance pooling* para instâncias de **HttpApplication** e opcionalmente para *handlers* (nas respectivas fábricas)
  - *Caching* para os restantes elementos do *pipeline*
- Quais as consequências?

Auto-detecção de suporte para *cookies* (1)



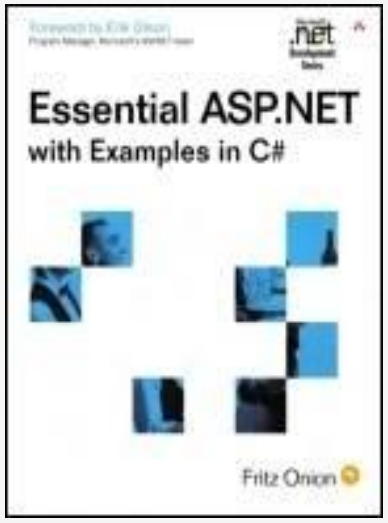
Auto-detecção de suporte para *cookies* (2)



Bibliografia

- Fritz Onion,

  "Essential ASP.NET with Examples in C#", Addison-Wesley, 2003 - Capítulo 4



- MSDN: http://msdn2.microsoft.com/library/system.web.aspx