STRUCTURE OF ASP.NET PAGE

www.vinsys.in

INTRODUCTION

- An ASP.NET page is a server-side text file saved with the *.aspx* extension.
- The internal structure of the page comprises of page directives, code and page layout.
- Page directives set up the environment in which the page will run, specify how the HTTP runtime should process the page.

INTRODUCTION

- Directives also import namespaces to simplify coding, load assemblies not currently in the global assembly cache (GAC), and register new controls with custom tag names and namespace prefixes.
- The code section contains handlers for page and control events, plus optional helper routines.
- The page layout represents the skeleton of the page. It includes server controls, literal text, and HTML tags.

PAGE DIRECTIVES

```
<%@ Page Language="VB" AutoEventWireup="false"
        CodeFile="Default.aspx.vb"Inherits="_Default" %>
```

- A directive is opened with a <%@ and closed with a %>.
- Syntax

<%@ [*Directive*] [*Attribute=Value*] [*Attribute=Value*] %>

- The @Page directive enables you to specify attributes and values for an ASP.NET page.

PAGE DIRECTIVE ATTRIBUTES

- CodeFile
  - References the code-behind file with which the page is associated.
- Inherits
  - Defines a code-behind class for the page to inherit. This can be any class derived from the *Page* class.

PAGE DIRECTIVE ATTRIBUTES

- Language
  - Defines the language being used for any inline rendering and script blocks.
- EnableTheming
  - Page is enabled to use themes when set to True.
- MaintainScrollPositionOnPostback
  - Scroll position are maintained on postback of same page.

PAGE DIRECTIVE ATTRIBUTES

- ErrorPage
  - Specifies a URL to post to for all unhandled page exceptions.
- Theme
  - Applies the specified theme to the page using the ASP.NET 2.0 themes feature.
- ContentType
  - Defines the HTTP content type of the response as a standard MIME type.

PAGE DIRECTIVE ATTRIBUTES

- Title
  - Applies a page's title.
- MasterPageFile
  - Takes a String value that points to the location of the master page used with the page. This attribute is used with content pages.
- AutoEventWireUp
  - Specifies whether the page events are autowired when set to True. The default setting for this attribute is True.

IMPORT DIRECTIVE

- The @*Import* directive allows you to specify a namespace to be imported into the ASP.NET page or user control.

- The *Namespace* attribute takes a String value that specifies the namespace to be imported.

-
```
<%@ Import Namespace=”System.Data” %>
<%@ Import Namespace=”System.Data.SqlClient” %>
```

IMPLEMENTS DIRECTIVE

- The @*Implements* directive gets the ASP.NET page to implement a specified .NET Framework interface.

- Supports only single attribute *Interface*.

```
<%@ Implements Interface=”System.Web.UI.IValidator” %>
```

ASSEMBLY DIRECTIVE

- The @Assembly directive attaches assemblies, the building blocks of .NET applications, to an ASP.NET page or user control as it compiles.
- *Name*: Enables one to specify the name of an assembly (without extension) used to attach to the page files.
-

```
<%@ Assembly Name=”OurAssembly” %>
<%@ Assembly Src=”OurAssembly.cs” %>
```

PAGE CLASS

- Every web page is a custom class that inherits from System.Web.UI.Page.
- The ASP.NET HTTP runtime processes the page object and causes it to generate the markup to insert in the response.
- The generation of the response is marked by several events handled by user code and collectively known as the *page life cycle*.

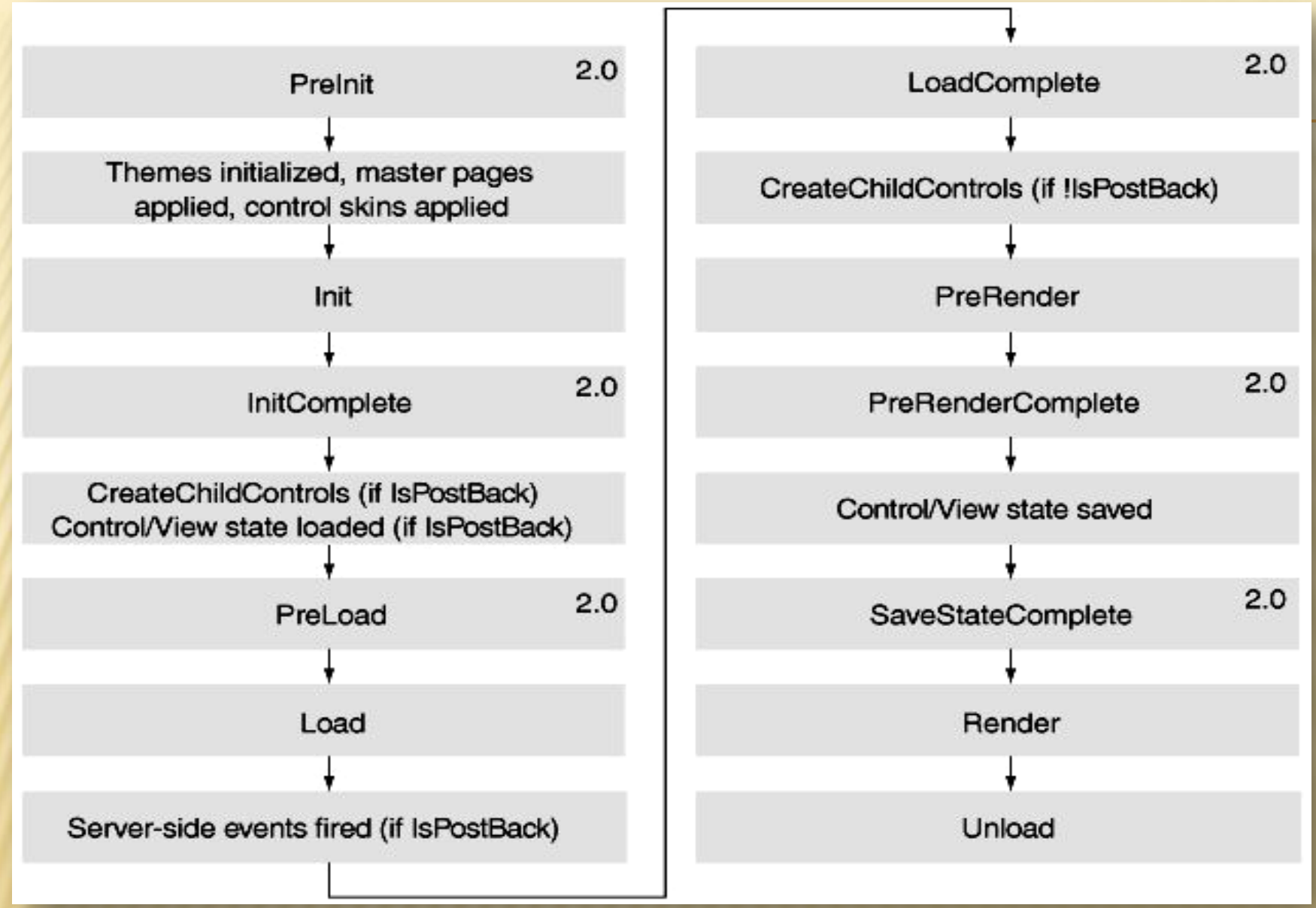
PAGE CLASS

public class Page:TemplateControl, IHttpHandler

- *TemplateControl* is the abstract class that provides both ASP.NET pages and user controls with a base set of functionality.
- *IHttpHandler* qualifying the object as the handler of a particular type of HTTP requests—those for *.aspx* files

PAGE EVENTS

- PreInit
  - Occurs just before the initialization phase of the page begins.
- Init
  - Occurs when the page is initialized, which is the first step in the page life cycle.
- InitComplete
  - Occurs when all child controls and the page have been initialized .



PAGE EVENTS

- PreLoad
  - Occurs just before the loading phase of the page begins.
- Load
  - Occurs when the page loads up, after being initialized.
- LoadComplete
  - Occurs when the loading of the page is completed and server events have been raised.

PAGE EVENTS

- PreRender
  - Occurs when the page is about to render.
- DataBinding
  - Occurs when the *DataBind* method is called on the page to bind all the child controls to their respective data sources.
- PreRenderComplete
  - Occurs just before the rendering phase begins.

PAGE EVENTS

- SaveStateComplete
  - Occurs when the view state of the page has been saved to the persistence medium.
- Unload
  - Occurs when the page is unloaded from memory but not yet disposed.
- Error
  - Occurs when an unhandled exception is thrown.

PAGE EVENTS

- Disposed
  - Occurs when the page is released from memory, which is the last stage of the page life cycle.
- AbortTransaction
  - Occurs for ASP.NET pages marked to participate in an automatic transaction when a transaction aborts.
- CommitTransaction
  - Occurs for ASP.NET pages marked to participate in an automatic transaction when a transaction commits.

PAGE INTRINSIC OBJECTS

- Application
  - Instance of the *HttpApplicationState* class; represents the state of the application.
- Request
  - Instance of the *HttpRequest* class; represents the current HTTP request.
- Response
  - Instance of the *HttpResponse* class; sends HTTP response data to the client.

PAGE INTRINSIC OBJECTS

- Server
  - Instance of the *HttpServerUtility* class; provides helper methods for processing Web requests.
- Trace
  - Instance of the *TraceContext* class; performs tracing on the page.
- Session
  - Instance of the *HttpSessionState* class; manages user-specific data.

PAGE INTRINSIC OBJECTS

- User
  - If the user has been authenticated, this property will be initialized with user information.
- Cache
  - Instance of the *Cache* class; implements the cache for an ASP.NET application.

PAGE PROPERTIES

- Form
  - Returns the current *HtmlForm* object for the page.
- ErrorPage
  - Gets or sets the error page to which the requesting browser is redirected in case of an unhandled page exception.
- IsPostBack
  - Indicates whether the page is being loaded in response to a client postback or whether it is being loaded for the first time .

PAGE PROPERTIES

- IsCrossPagePostBack
  - Indicates whether the page is being loaded in response to a postback made from within another page.
- IsValid
  - Indicates whether page validation succeeded.
- Master
  - Instance of the *MasterPage* class; represents the master page that determines the appearance of the current page.

PAGE PROPERTIES

- Page
  - Returns the current *Page* object.
- PreviousPage
  - Returns the reference to the caller page in case of a cross-page postback..
- IsCallback
  - Indicates whether the page is being loaded in response to a client script callback.

PAGE PROPERTIES

- Controls
  - Returns the collection of all the child controls contained in the current page.
- ClientScript
  - Gets a *ClientScriptManager* object that contains the client script used on the page.
- Title
  - Gets or sets the title for the page.
- TraceEnabled
  - Toggles page tracing on and off.

PAGE PROPERTIES

- EnableViewState
  - Indicates whether the page has to manage view-state data.
- StyleSheetTheme
  - Gets or sets the name of the style sheet applied to this page.
- ClientQueryString
  - Gets the query string portion of the requested URL.

PAGE METHODS

- FindControl
  - Takes a control's ID and searches for it in the page.
- HasControls
  - Determines whether the page contains any child controls.
- MapPath
  - Retrieves the physical, fully qualified path that an absolute or relative virtual path maps to.

PAGE METHODS

- LoadControl
  - Compiles and loads a user control from an .ascx file, and returns a *Control* object. If the user control supports caching, the object returned is *PartialCachingControl*.
- GetCallbackEventReference
  - Obtains a reference to a client-side function that, when invoked, initiates a client call back to server-side events.

PAGE METHODS

- GetPostBackEventReference
  - Returns the prototype of the client-side script function that causes, when invoked, a postback. It takes a *Control* and an argument, and it returns a string like this:

    __doPostBack('CtlID','')
- SetFocus
  - Sets the browser focus to the specified control.
- GetPostBackClientHyperlink
  - Appends *javascript:* to the beginning of the return string received from *GetPostBackEventReference*.

    javascript:__doPostBack('CtlID','')

CODE SECTION

- Any source code relevant to the page can be inserted inline or attached to the page through a separate file.
- If inserted inline, the code goes into a tag <script>.
- Server-side <*script*> tags are distinguished from client-side <*script*> tags by the use of the *runat=server* attribute.

CODE SECTION

- *CodeFile* attribute refers the separate code file. Must appear with *Inherits* attribute.

```
<script runat="server">
private void Page_Load(Object sender,EventArgs e)
{
        Response.Write("Hello, World!!!");
}
</script>
```

POSTBACKS

- A page instance is created on every request from the client, and its execution causes itself and its contained controls to iterate through their life-cycle stages.

- Postback consists of posting form data to the same page using the view state to restore the same state of controls existing when the posting page was last generated on the server.

POSTBACKS

- The __*VIEWSTATE* hidden field is where the view state of all controls is persisted at the end of a request.
- PreInit sets the *IsCallback*, *IsCrossPagePostback*, and *IsPostback*.
- *IsPostBack* tells that if the request is first or post back.

ISPOSTBACK

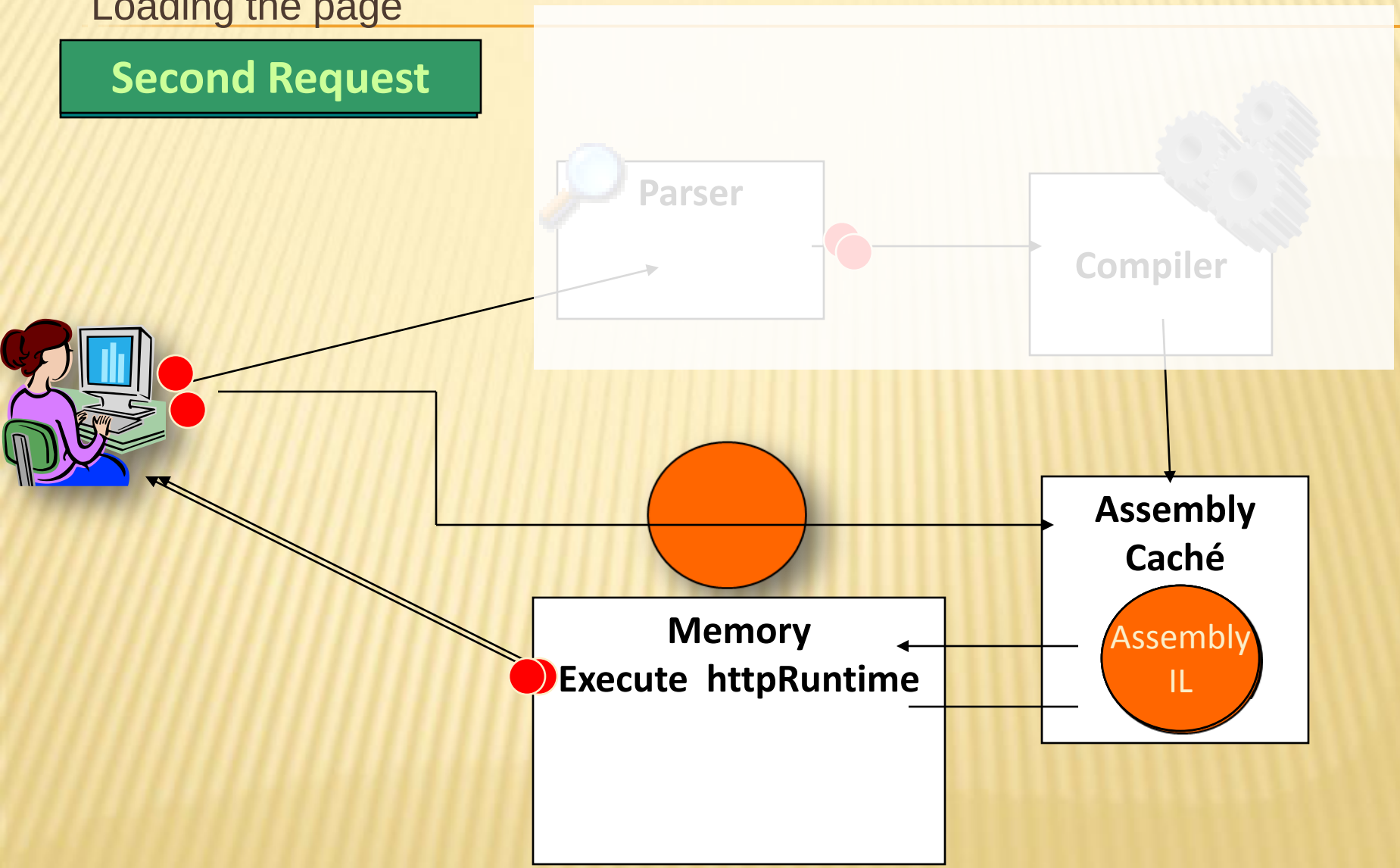
```
if(Page.IsPostBack)
{
        //Statements
}
```

```
If Page.IsPostBack Then
        ‘statements
End If
```

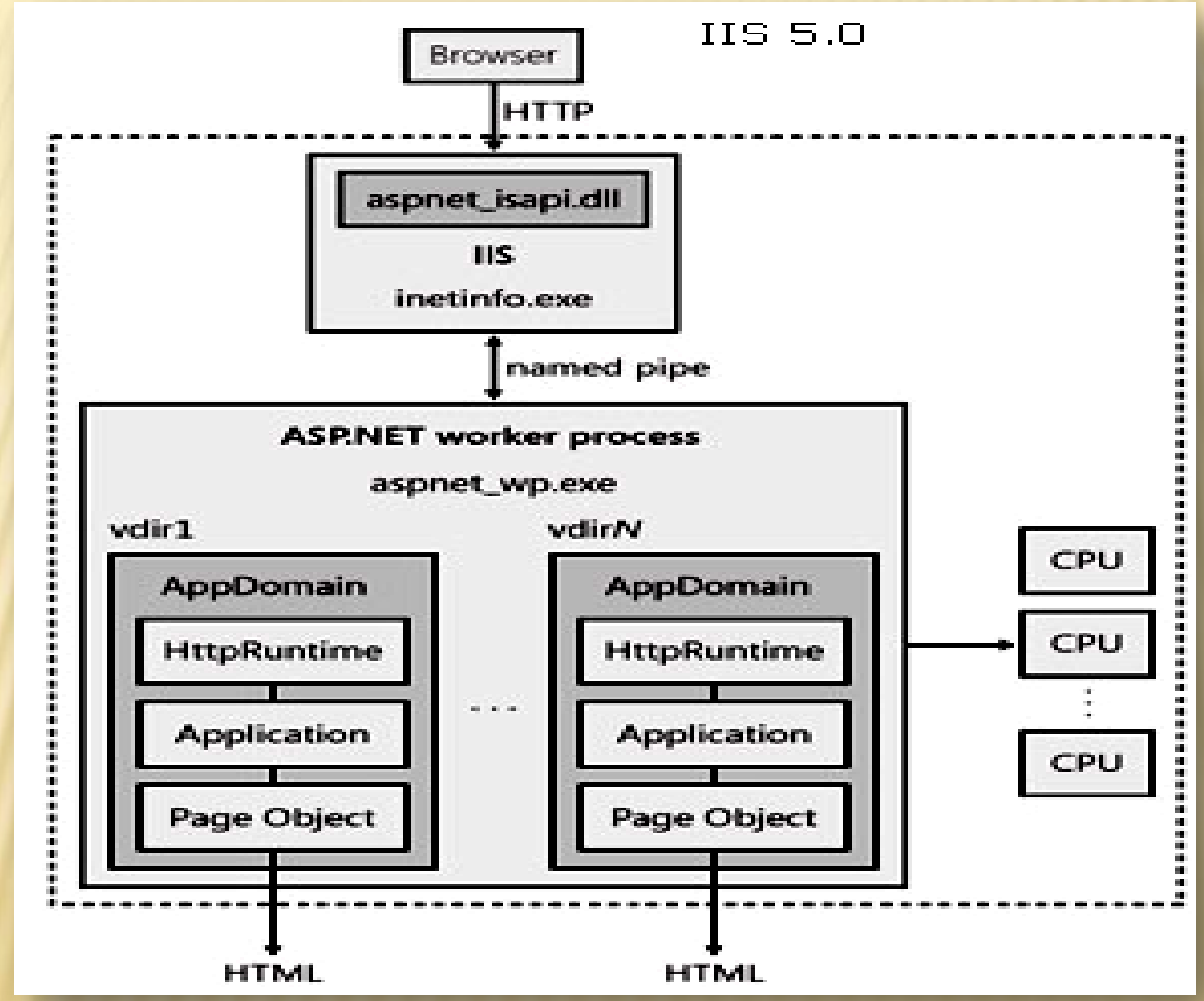
The .aspx page as a dynamic page

Loading the page

Second Request



IIS PROCESS MODEL



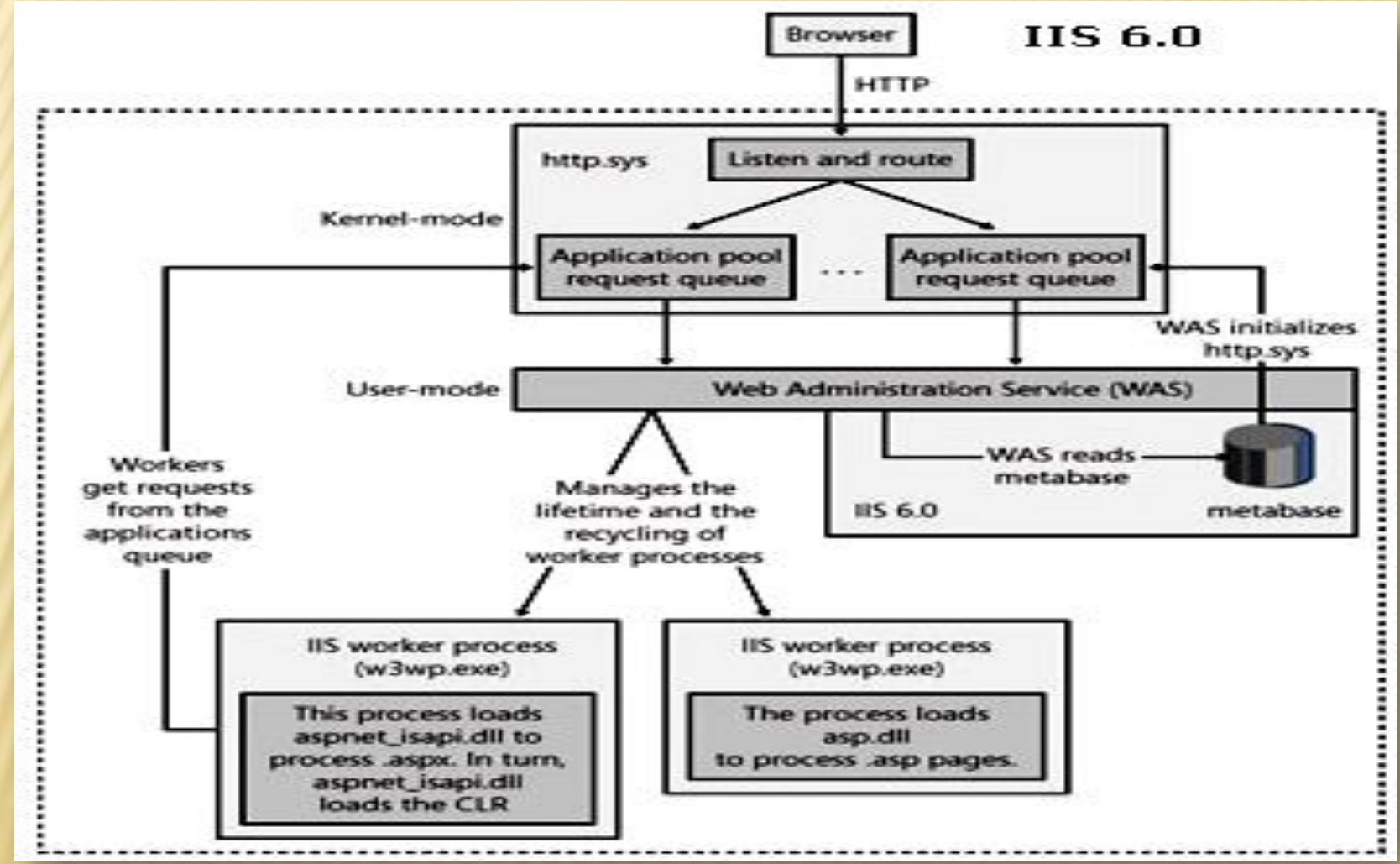
IIS PROCESS MODEL

- When the request for a resource arrives, IIS first verifies the type of the resource. Static resources such as images, text files, and HTML pages are resolved directly by IIS.
- ASP.NET pages are handled by aspnet_isapi.dll
- aspnet_isapi.dll collects all the information available about the invoked URL and the underlying resource, and then it routes the request ASP.NET worker process (aspnet_wp.exe).

IIS PROCESS MODEL

- Each web application identified with its virtual directory and belongs to a distinct *application domain*.
- A new AppDomain is created within the ASP.NET worker process whenever a client addresses a virtual directory for the first time.
- In new AppDomain, ASP.NET runtime loads all the needed assemblies and passes control to the hosted HTTP pipeline to serve the request.

IIS PROCESS MODEL



IIS PROCESS MODEL

- IIS 6.0 implements its HTTP listener as a kernel-level module.
- All incoming requests are first managed by *http.sys* driver in kernel.

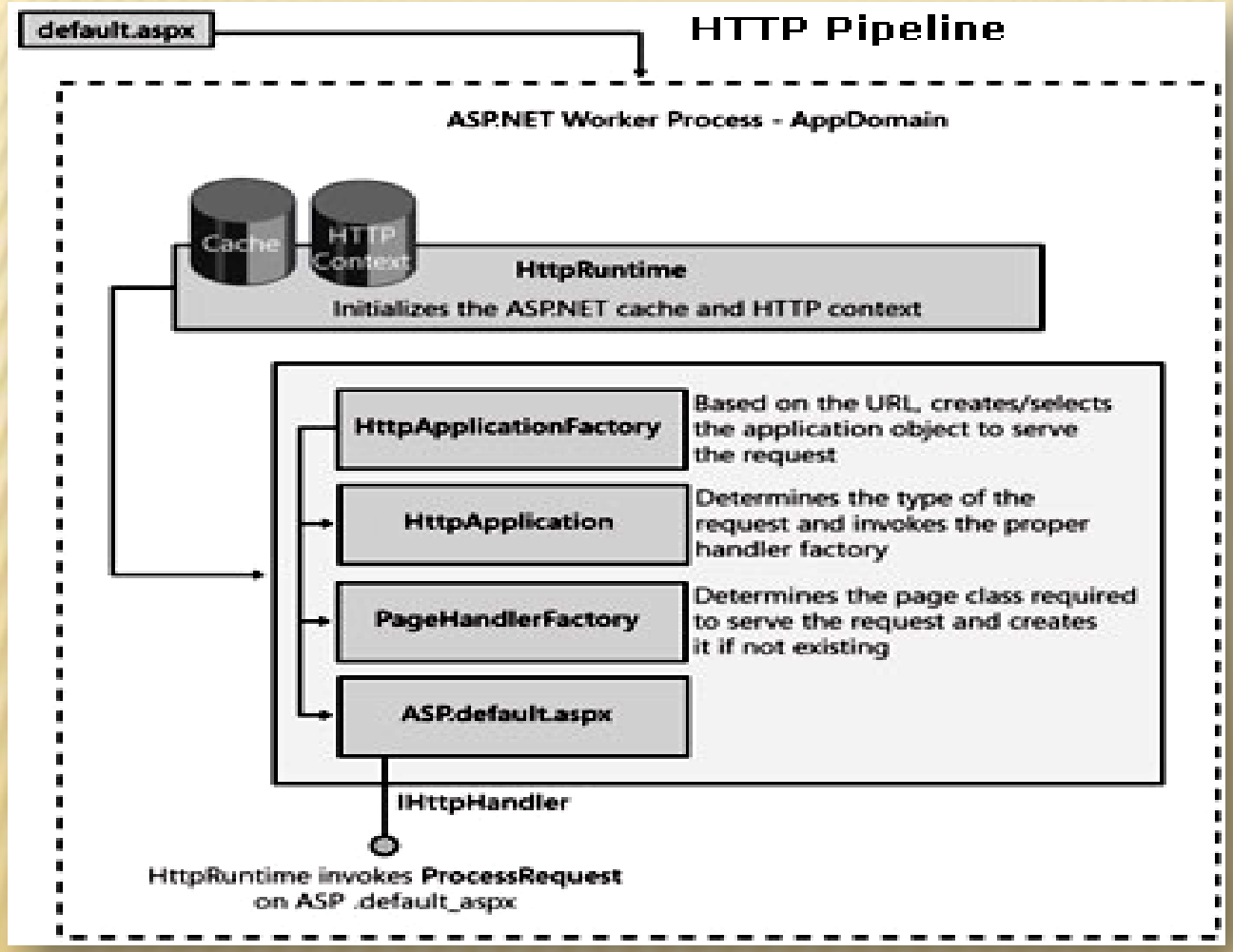
IIS PROCESS MODEL

- The *http.sys* driver listens for requests and posts them to the request queue of the appropriate application pool.
- Web Administration Service (WAS) creates the request queues.
- The HTTP request is delivered directly at the worker process that hosts the CLR.

HTTPRUNTIME OBJECT

- The ASP.NET worker process passes any incoming HTTP requests to HTTP pipeline.
- HTTP pipeline is activated by creating a new instance of HTTP Runtime, which calls *ProcessRequest* (not of IHttpHandler) method.
- *HttpRuntime* object creates a new context for the request and initializes a specialized text writer object in which the markup code will be accumulated.

HTTPRUNTIME OBJECT



HTTPAPPLICATIONFACTORY OBJECT

- *HttpApplicationFactory* object maintains a pool of *HttpApplication* objects to serve incoming HTTP requests.
- When invoked, the application factory object verifies that an AppDomain exists for the virtual folder the request targets.
- If AppDomain doesn't exist, it creates a new *HttpApplication* object in new AppDomain.

HTTPAPPLICATIONFACTORY OBJECT

- The creation of an *HttpApplication* object entails the compilation of the *global.asax* application file, if one is present, and the creation of the assembly that represents the actual page requested.

- An *HttpApplication* object is used to process a single page request at a time.

HTTPAPPLICATION OBJECT

- *HttpApplication* is the base class that represents a running ASP.NET application. A running ASP.NET application is represented by a dynamically created class that inherits from *HttpApplication*.

- If *global.asax* is available, the application class is built and named after it: *ASP.global_asax*. Otherwise, the base *HttpApplication* class is used.

HTTPAPPLICATION OBJECT

- *HttpApplication* object determines the type of object that represents the resource being requested (e.g.Web Page, Web Service, User Control etc.).

- It asks to proper handler factory to get object to handle the request.

- A handler factory object is a class that implements the *IHttpHandlerFactory* interface and is responsible for returning an instance of a managed class that can handle the HTTP request—an HTTP handler

PAGE FACTORY

- For a request that targets a page, the factory is a class named *PageHandlerFactory*.
- It creates the object representing the page requested. Object inherits from *Page* class which implements *IHttpHandler* interface.
- The page object is returned to the application factory, which passes that back to the *HttpRuntime* object.