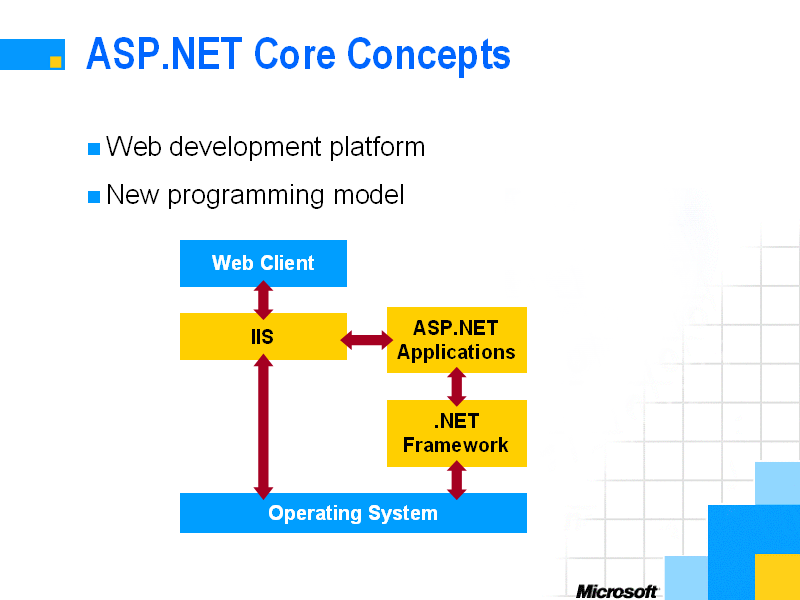
#### SYSTEM Architecture

In this section you will get an overview of the .NET Framework architecture, the Web application model, and the configuration system.



#### The .NET Framework Architecture

To really understand what makes ASP.NET more than just “ASP 4.0” we need to have a look at the underlying .NET Framework architecture.

The .NET Framework architecture is built on top of the operating system services. This framework contains different layers.

In the .NET Framework common language runtime, this resides on top of the operating system services. The common language runtime loads and executes code that targets the runtime. This code is therefore called *managed code*. The runtime gives you, for example, the ability for cross-language integration. For that matter it makes use of the common type system, which defines a standard set of types and rules to create new types.

As mentioned earlier, the .NET Framework provides a rich set of class libraries. These include base classes, like networking and input/output classes, a data class library for data access, and classes for use by programming tools, such as debugging services. All of them are brought together by the Services Framework, which sits on top of the common language runtime.

The top layer of the .NET framework consists of the Windows application model and, in parallel, the Web application model.

Use the Windows application model—Windows Forms—to develop more traditional Windows applications and take advantage of new Windows 2000 features. You can include existing COM components, but also make use of Web Services (which are explained later).

The Web application model—in the slide presented as ASP.NET—includes Web Forms and Web Services.

ASP.NET comes with built-in Web Forms controls, which are responsible for generating the user interface. They mirror typical HTML widgets like text boxes or buttons. If these controls do not fit your needs, you are free to create your own user controls.

Web Services brings you a model to bind different applications over the Internet. This model is based on existing infrastructure and applications and is therefore standard-based, simple, and adaptable.

A Web Service can be understood as a *contract* between server and client. Those contracts are described by the XML-based Web Services Description Language (WSDL). Doing so, Web Services are defined as endpoints that communicate via XML-based messages.

#### Web Application Model

But now we will show you in more detail how ASP.NET requests are handled.

As with earlier versions of ASP, a client can access your Web application using URLs. So, a Web application is a set of URLs related to one or more virtual directories on the Web server.

Each request is processed by the HTTP runtime, which is the core of the ASP.NET Web application model. Processing consists of resolving the URL of the request to the corresponding application, and dispatching the request to the application for further processing.

Requests are led through a pipeline of HTTP *modules*. With each module a developer can catch and modify requests. One of those modules could be, for example, a security module.

At the end of the module pipeline, there are *request handlers*. They enable the processing of individual URLs within an application.

From the developer’s point of view there is easy access to a clean and well-structured object model. Beside those aspects mentioned above, there is an object encapsulating all information about an individual HTTP request within ASP.NET. This object is called **HttpContext**. HTTP modules and request handlers access ASP.NET intrinsic via **HttpContext**.

#### HTTP Runtime

Now in more detail:

##### Managed code

While the HTTP runtime is managed code—because it targets the common language runtime—it runs within an unmanaged host process that could be, for example, Microsoft IIS.

So, for ASP.NET the .NET Framework uses IIS as a more or less “dump” gateway for its own HTTP infrastructure.

##### Aims for 100% availability

The HTTP runtime processes all requests asynchronously. Because of this and because it’s multithreaded, the HTTP runtime is fail-safe to a high level. For example, badly written code cannot block the HTTP runtime.

##### Replaces ISAPI

The HTTP runtime replaces the Internet Server Application Programming Interface (ISAPI).