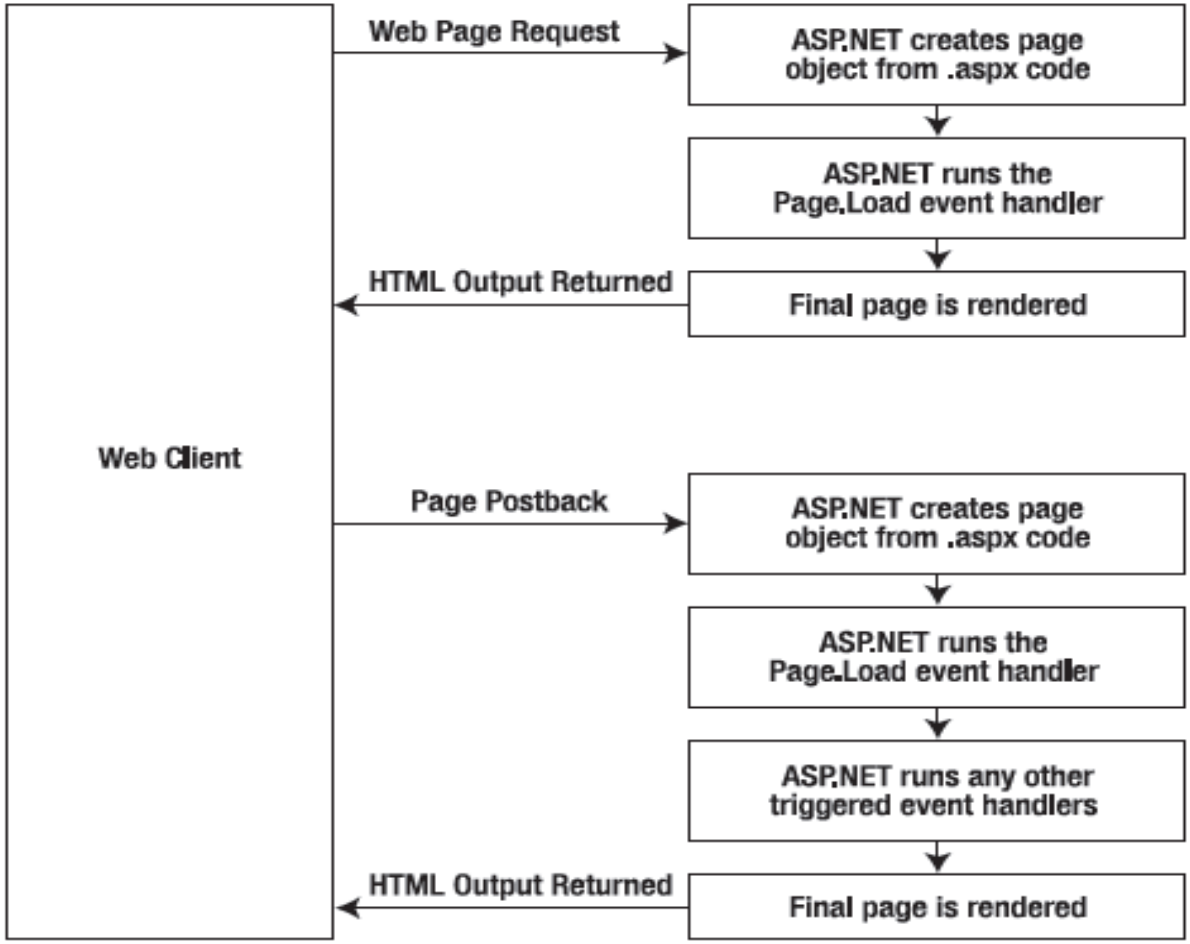
Web Controls

Today You Will Learn

- Web Control Events and AutoPostBack
- A simple example

Web Control Events and AutoPostBack

Page Processing Sequence



Web Control Events and AutoPostBack

- Some events, such as the **Click event** of a button, do occur immediately. That's because when clicked, the button **posts back the page**.
- However, other actions *do* cause events but *don't* trigger a postback. An example is when the user changes the text in a text box (which triggers the TextChanged event) or chooses a new item in a list (the SelectedIndexChanged event).
- You might want to respond to these events, but **without a postback** your code has no way to run.

Web Control Events and AutoPostBack

- **ASP.NET handles this by giving you two options:**

1. You can wait until the next postback to react to the event. For example, imagine you want to react to the SelectedIndexChanged event in a list. If the user selects an item in a list, nothing happens immediately.

- However, if the user then clicks a button to post back the page, *two* events fire: Button.Click followed by ListBox.SelectedIndexChanged.
- And if you have several controls, it's quite possible for a single postback to result in several change events, which fire one after the other, in an **undetermined order**.

Web Control Events and AutoPostBack

2. You can use the ***automatic postback*** feature to force a control to post back the page immediately when it detects a specific user action.

- In this scenario, when the user clicks a new item in the list, the page is posted back, your code executes, and a new version of the page is returned.
- The option you choose depends on the result you want. If you need to react immediately you need to use automatic postbacks.
- On the other hand, **automatic postbacks** can sometimes make the **page less responsive**, because each postback and page refresh adds a short, but noticeable, delay and page refresh.

Web Control Events and AutoPostBack

- All input web controls support automatic postbacks. Table provides a basic list of web controls and their events.

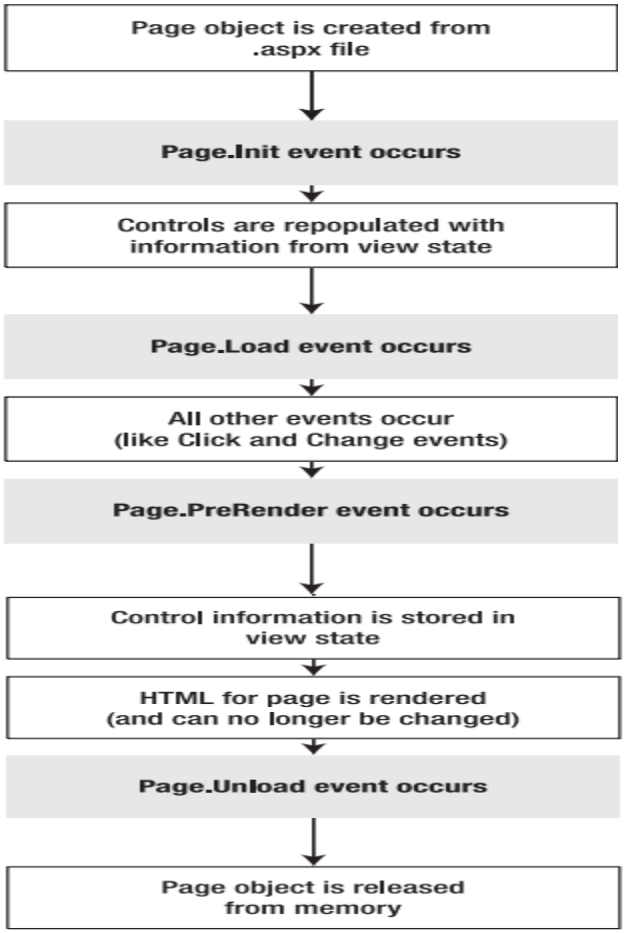
Event	Web Controls That Provide It	Always Posts Back
Click	Button, ImageButton	True
TextChanged	TextBox (fires only after the user changes the focus to another control)	False
CheckedChanged	CheckBox, RadioButton	False
SelectedIndexChanged	DropDownList, ListBox, CheckBoxList, RadioButtonList	False

Web Control Events and AutoPostBack

- If you want to capture a change event such as TextChanged immediately, you need to set the control's **AutoPostBack** property to **True**.
- This way, the page will be submitted automatically when the user interacts with the control.
- When the **page is posted back**, ASP.NET will examine the page, load all the current information, and then allow your code to perform some extra processing before returning the page back to the user.

Web Control Events and AutoPostBack

The postback processing sequence



Init - Use this event to read or initialize control properties.

Load - Use this event method to set properties in controls and to establish database connections.

PreRender - Use the event to make final changes to the contents of the page or its controls before the rendering stage begins.

Unload - use this event to do final cleanup work, such as closing open files and database connections, or finishing up logging or other request-specific tasks.

Web Control Events and AutoPostBack

How Postback Events Work

- If you create a web page that includes one or more web controls that are configured to use **AutoPostBack**, ASP.NET adds a **special JavaScript function** to the rendered HTML page.
- This function is named **__doPostBack().** When called, it triggers a **postback**, sending data back to the web server.
- ASP.NET also adds **two additional hidden input fields** that are used to pass information back to the server.
- This information consists of the
  - **ID of the control that raised the event**
  - **Any additional information that might be relevant**.
- These fields are initially empty, as shown here:

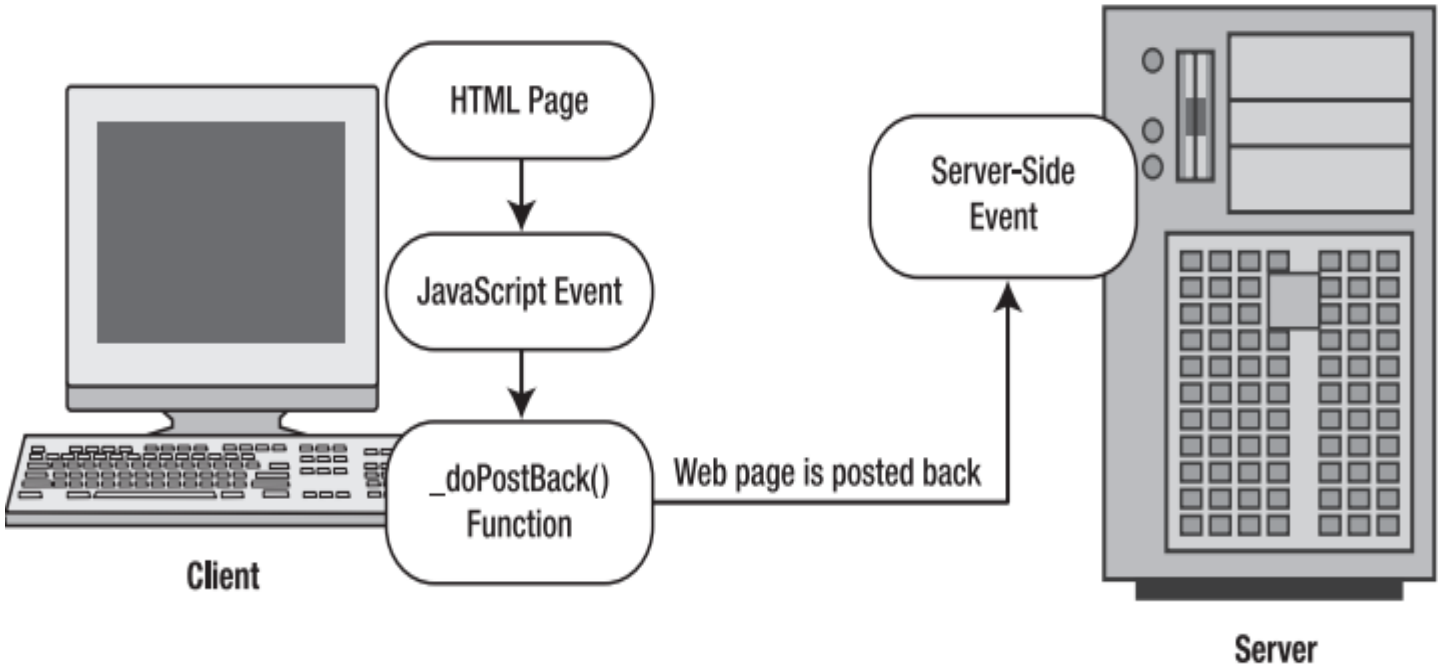
```
<input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
<input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT" value="" />
```

Web Control Events and AutoPostBack

- The **__doPostBack()** function has the responsibility for setting these values with the appropriate information about the event and then submitting the form.
- Remember, ASP.NET generates the **__doPostBack() function automatically**, provided at least one control on the page uses automatic postbacks.
- Finally, any control that has its **AutoPostBack** property set to **True** is connected to the __doPostBack() function using the **onclick or onchange** attributes.
- These attributes indicate what action the browser should take in response to the client-side JavaScript events onclick and onchange.

Web Control Events and AutoPostBack

An automatic postback



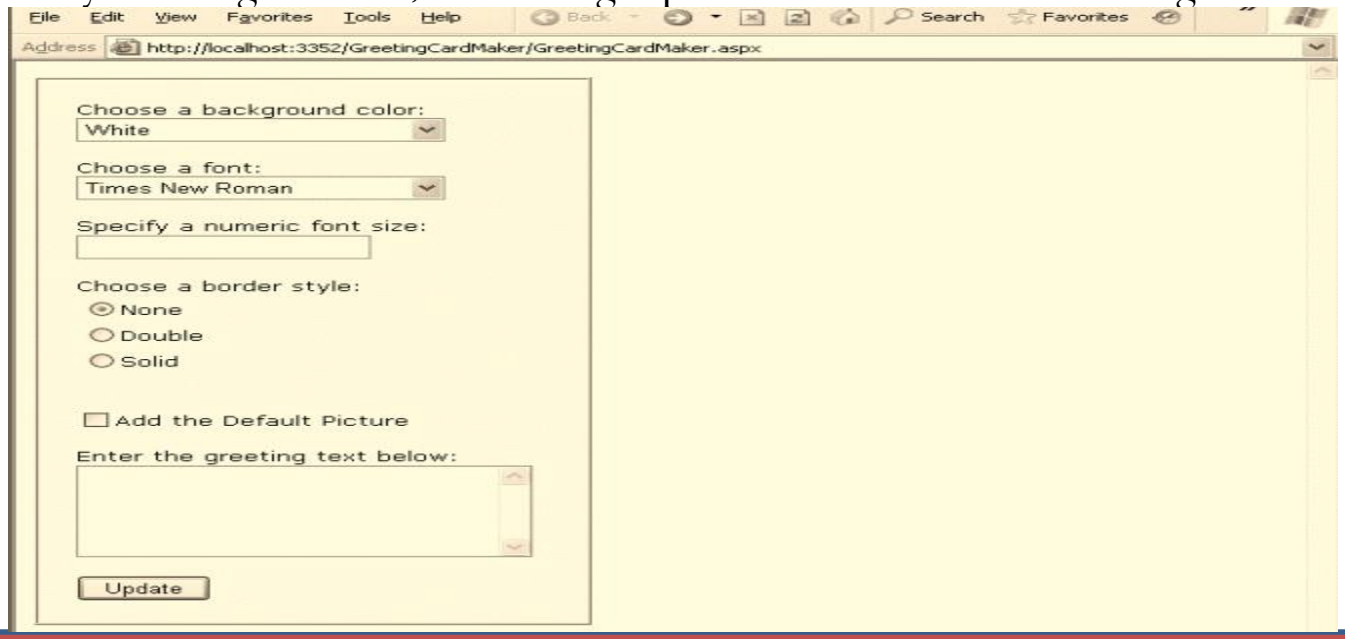
Web Control Events and AutoPostBack

The Page Life Cycle

- Consider what happens when a user changes a control that has the **AutoPostBack** property set to **True**:
  1. On the client side, the JavaScript __doPostBack function is invoked, and the page is resubmitted to the server.
  2. ASP.NET re-creates the Page object using the .aspx file.
  3. ASP.NET retrieves state information from the hidden view state field and updates the controls accordingly.
  4. The Page.Load event is fired.
  5. The appropriate change event is fired for the control. (If more than one control has been changed, the order of change events is undetermined.)
  6. The Page.PreRender event fires, and the page is rendered (transformed from a set of objects to an HTML page).
  7. Finally, the Page.Unload event is fired.
  8. The new page is sent to the client.

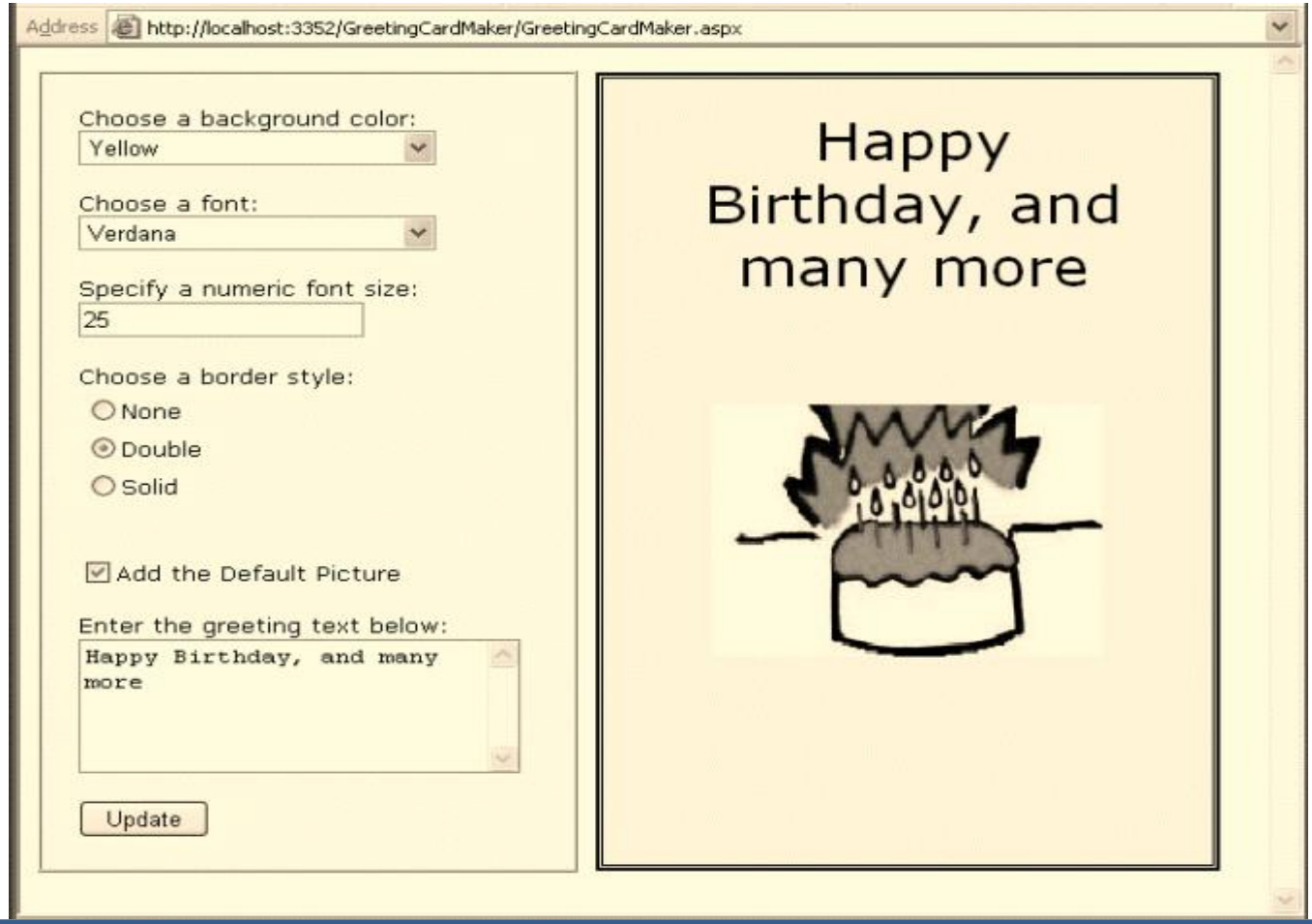
A Simple Web Page

- A simple example for a dynamic e-card generator. The web page is divided into two regions.
  - On the left is an ordinary tag containing a set of web controls for specifying card options.
  - On the right is a Panel control (named pnlCard), which contains
- Two other controls (lblGreeting and imgDefault) that are used to display user-configurable text and a picture. This text and picture represents the greeting card. When the page first loads, the card hasn't yet been generated, and the right portion is blank as shown in fig.



A Simple Web Page

- Whenever the user clicks the Update button, the page is posted back and the "card" is updated as shown in figure.



A Simple Web Page

Generating the Cards Automatically

- The last step is to use ASP.NET's automatic postback events to make the card **update dynamically** every time an option is changed.
- The Update button could now be used to submit the final, perfected greeting card, which might then be e-mailed to a recipient or stored in a database.
- To configure the controls so they automatically trigger a page postback, simply set the **AutoPostBack property of each input control to True**. An example is shown here:
- Choose a background color:

```
<br /> <asp:DropDownList ID="lstBackColor" AutoPostBack="True" runat="server“
Width="194px" Height="22px"/>
```

A Simple Web Page

- Next, you need to create an **event handler** that can handle the change events. To save a few steps, you can use the same event handler for all the input controls.
- All the event handler needs to do is call the update routine that regenerates the greeting card.

```
protected void ControlChanged(object sender, System.EventArgs e)
{
    // Refresh the greeting card (because a control was changed).
    UpdateCard();
}
protected void cmdUpdate_Click(object sender, EventArgs e)
{
    // Refresh the greeting card (because the button was clicked).
    UpdateCard();
}
private void UpdateCard()
{
    // (The code that draws the greeting card goes here.)
}
```

END OF LECTURE