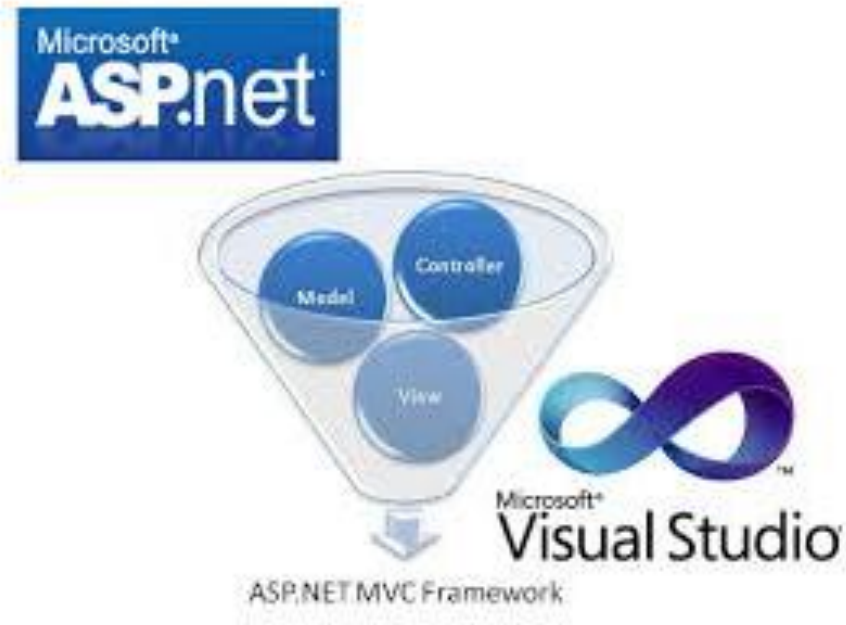


C#

1.INTRODUCTION

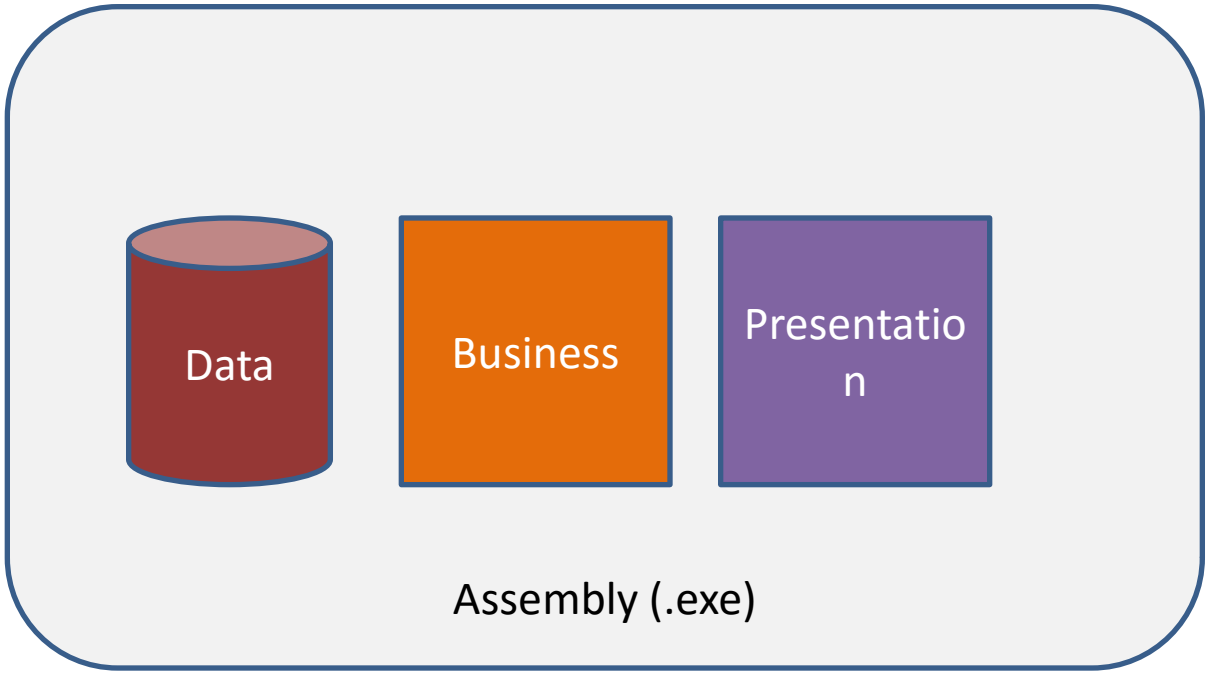
Agenda

- Architectures
- Where does ASP.NET MVC fit in
- MVC – Motivation
- Simple MVC Application
- MVC – Model View Controller
- Routing Basics
- Models
- Views
- Controllers
- Actions
- Bundling & Minification
- Validation
- Filters
- MVC Request Cycle

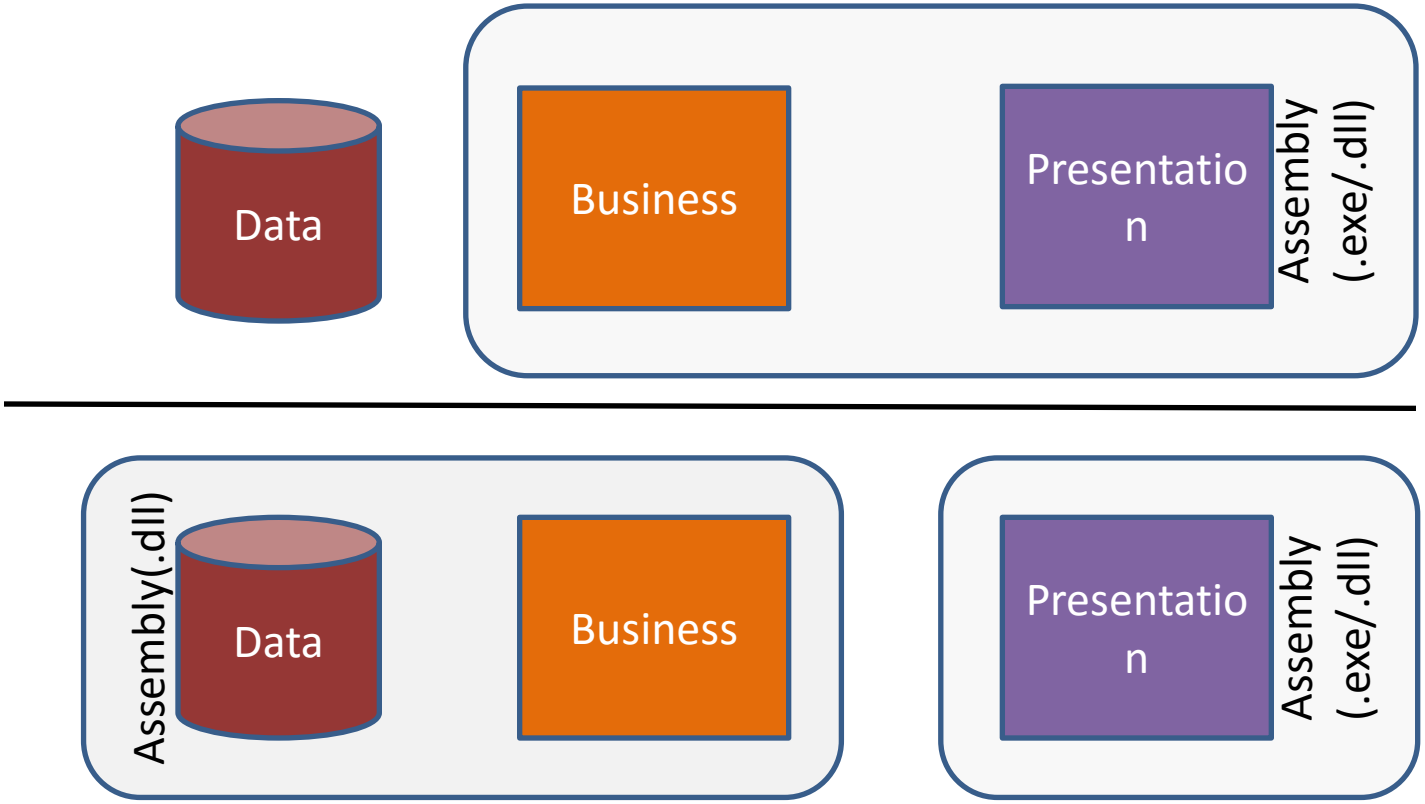
Architectures

- Three major concerns
  - Data / Storage (Files, Databases, External Storage)
  - Business Logic (Logic revolving adding/updating/deleting business objects)
  - Presentation
- One Tier
- Two Tier
- Three Tier
- n-Tier

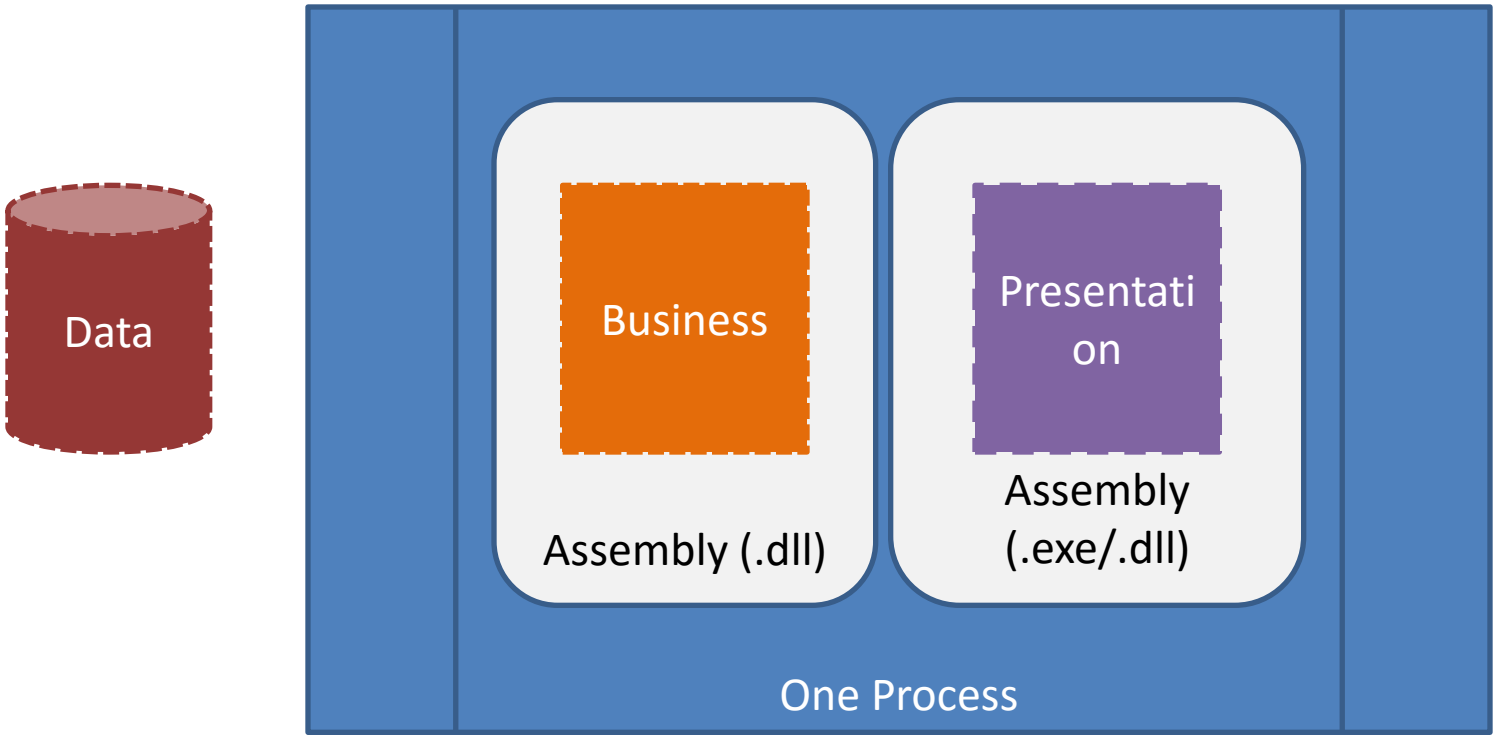
One Tier Architecture



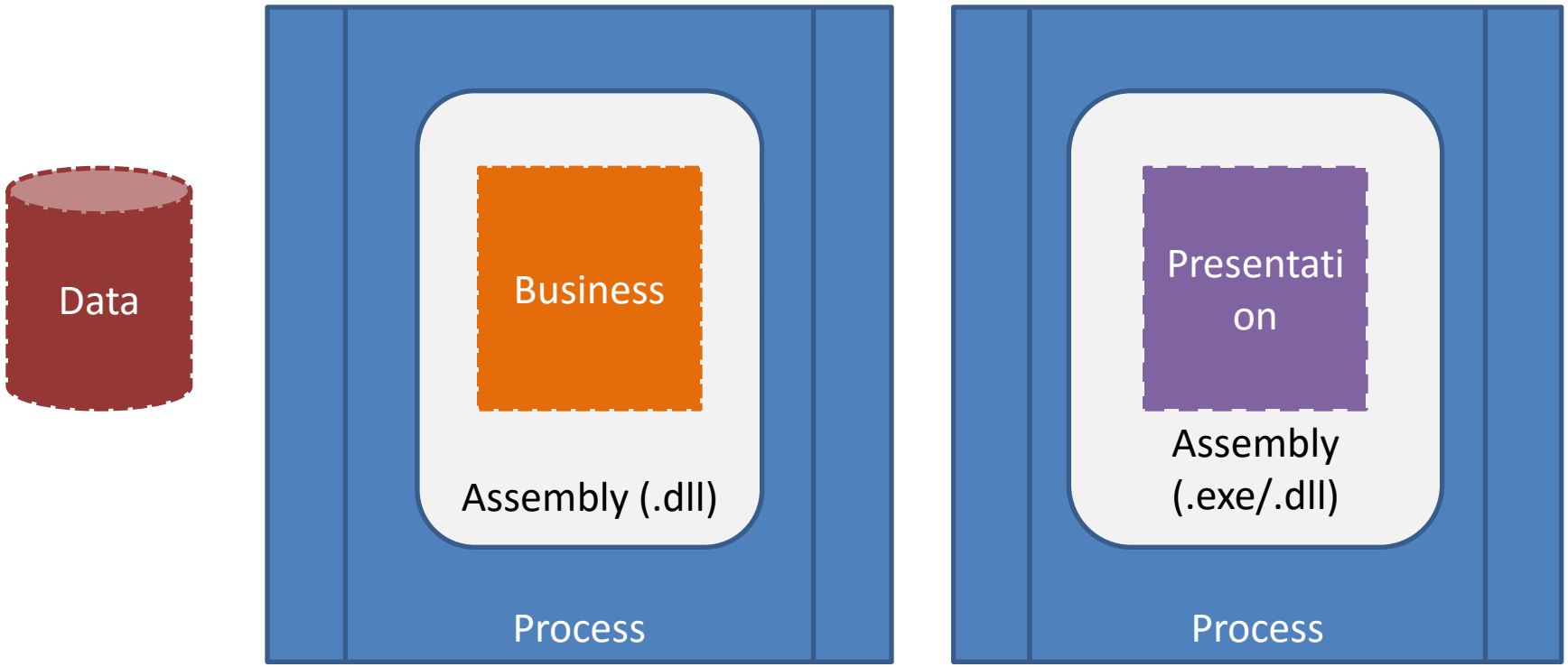
Two Tier Architecture



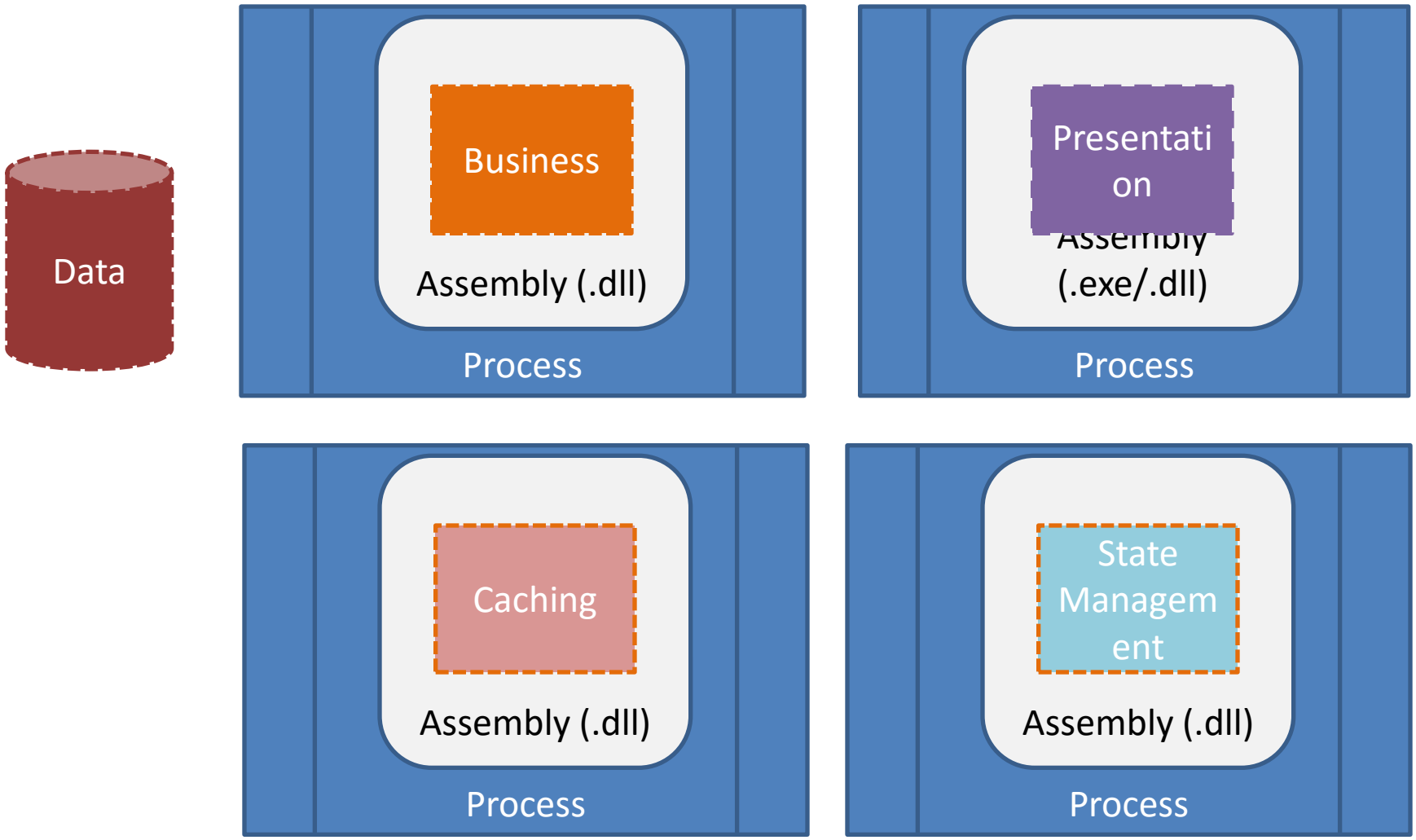
Three Tier Architecture (Logical)



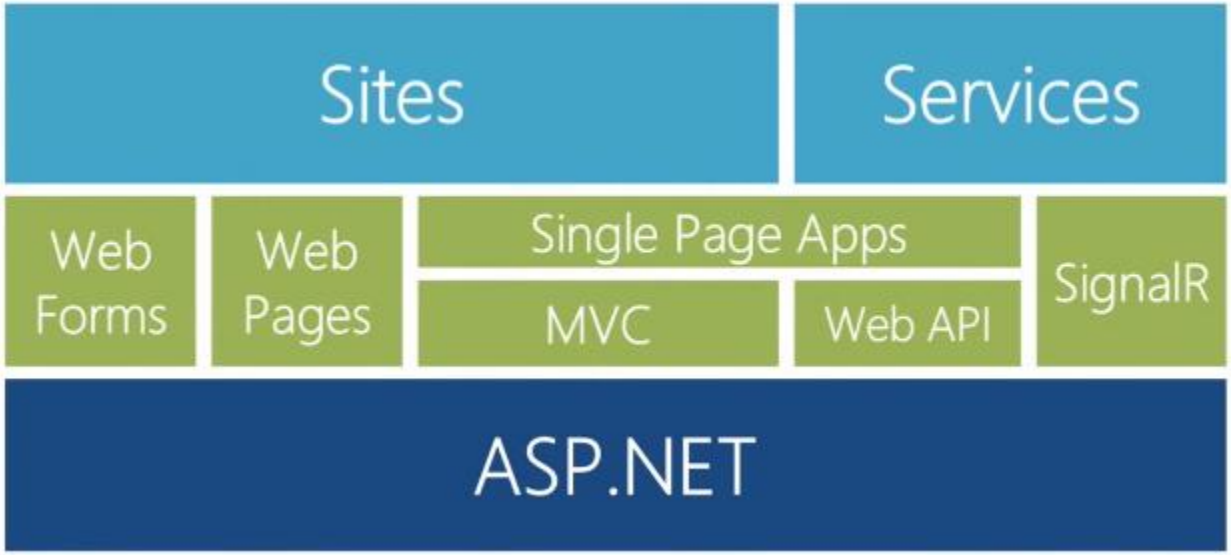
Three Tier Architecture (Physical)



n-Tier Architecture (Physical)

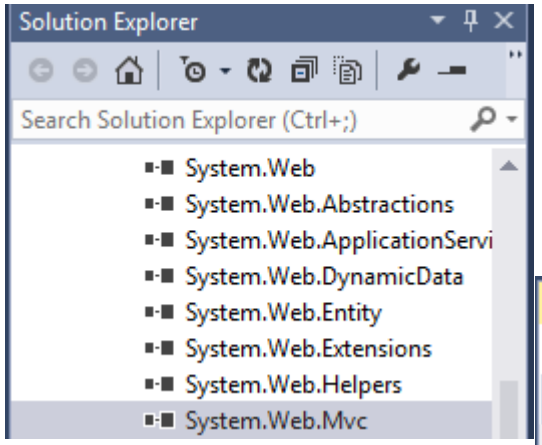


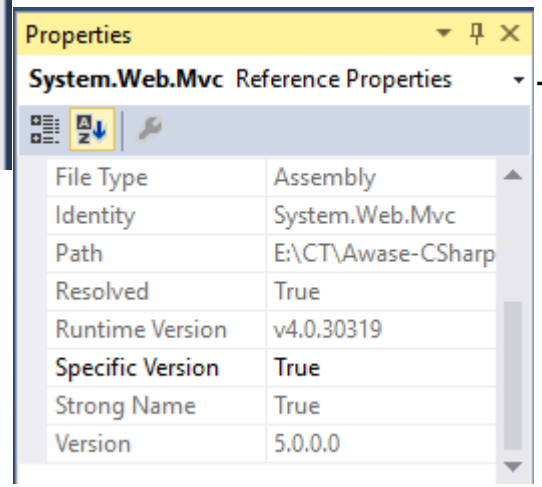
ASP.NET OVERVIEW



Installing ASP.NET MVC

- ASP.NET MVC
  - http://www.asp.net/mvc





What MVC version

- 2 ways to identify
  - At design time – Go to solution explorer -> expand “References” folder. Right click on “System.Web.MVC” assembly and select properties.
  - At run-time using the following code
    - Typeof(Controller).Assembly.GetName().Version.ToString();

Revisiting ASP.NET Web Forms

- First released in ASP .NET 1.0
- Replaced classic ASP (Active Server Pages)
  - Strongly typed code replace script
  - Abstract away the web
  - Click events replaced “POST” operations
- Original design from the late 90s
  - Web standards have strengthened
  - Client-side programming on the rise
- Web forms compete against other MVC frameworks
  - STURTS
  - RUBY ON RAILS (ROR)
  - DJANGO - PYTHON
  - ANGULARJS (VERY RECENTLY)

- Productive way to build web applications
- Control and event-based programming model
- Controls that abstract HTML, JS and CSS
- Rich UI Controls- datagrid, charts, AJAX
- Browser differences are handled for you

What's wrong with ASP.NET WEB FORM

- ViewState
- Page Life Cycle
- Limited Control over the rendered HTML
- Lack of Separation of Concerns (SoC)
- Untestable

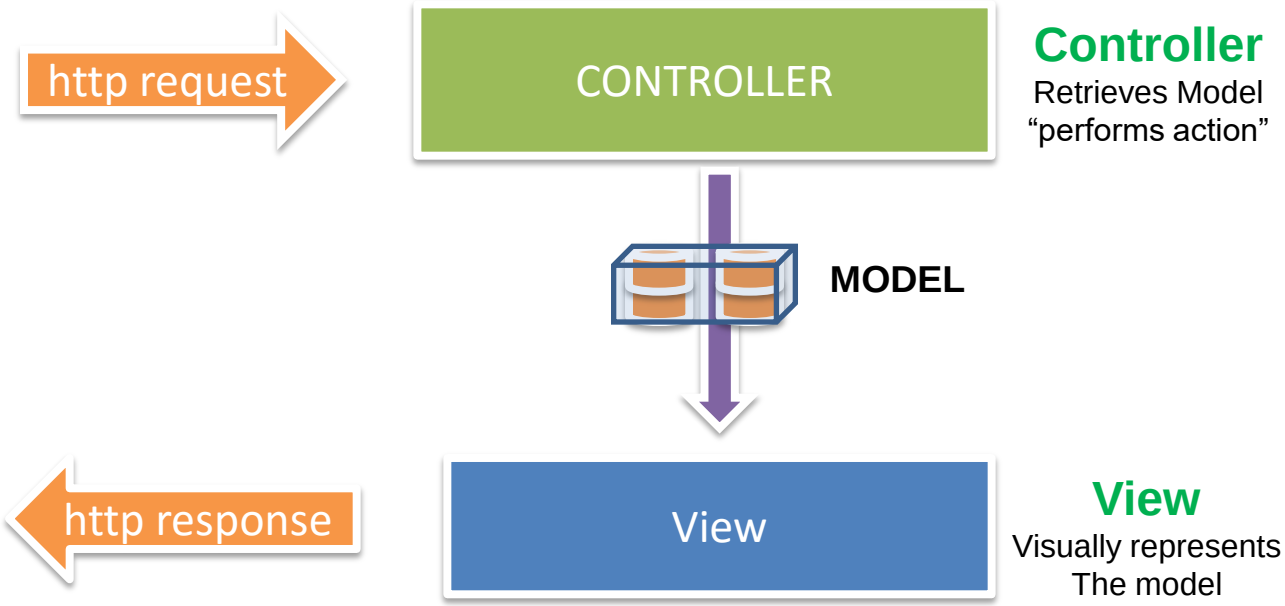
When to USE ASP.NET MVC

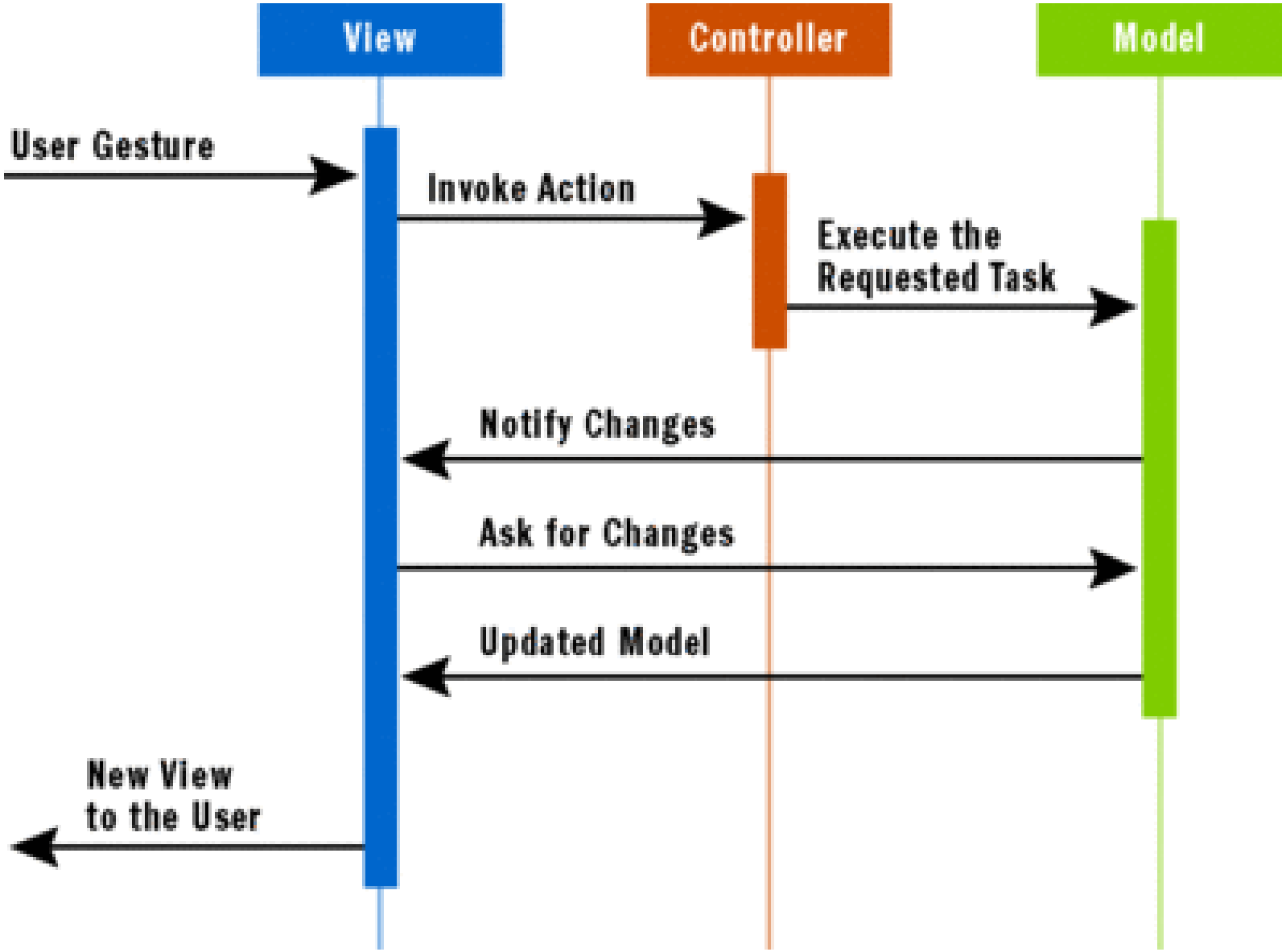
- ASP.NET MVC is **NOT** a replacement of ASP.NET web forms based applications
- The approach of application development must be decided based on the application requirements and features provided by ASP.NET MVC to suite them
- Application development with ASP.NET MVC is more complex as compared to web forms based applications as they lack readily available rich controls and less knowledge of the pattern in ASP.NET Web Developers
- Application maintainability will be higher with separation of application tasks

MVC DESIGN PATTERN

- View does not use Controller to update Model. Controller handles the events from View to manage user's interaction and data (via interaction with Model)
- Controller can be combined with View. Logical Separation of Model from the View
- The Controller does not contain the rendering logic.
- Controller uses view asking it to render new data.

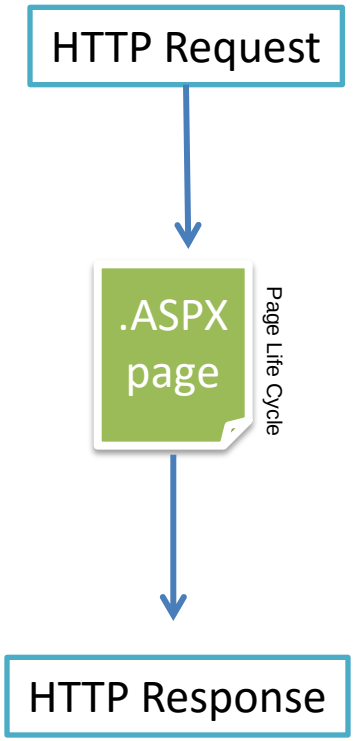
MVC flow



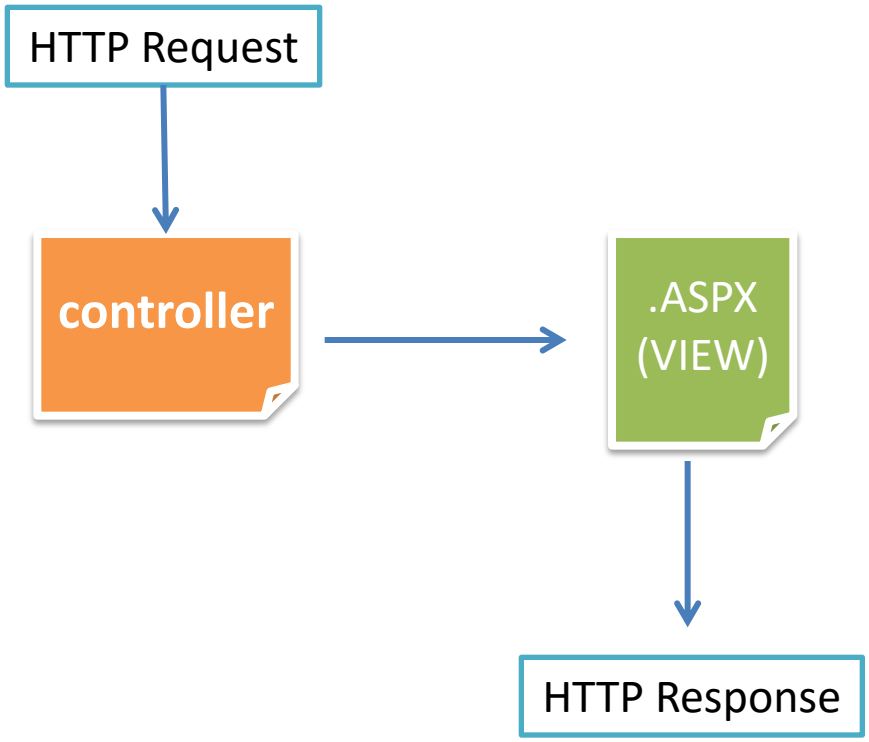


WEB FORMS vs MVC

ASP.NET WEB FORMS



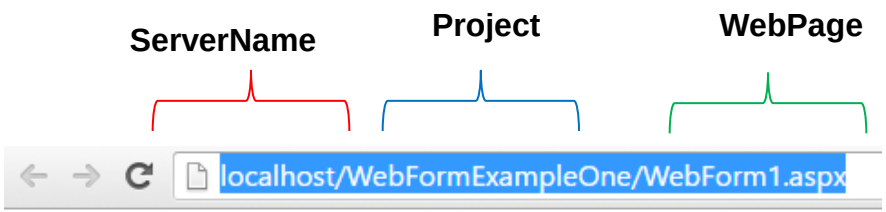
ASP.NET MVC



WEB FORMS vs MVC

Web Forms

- In a **webForms URL's** are mapped to **<u>Physical Files</u>**

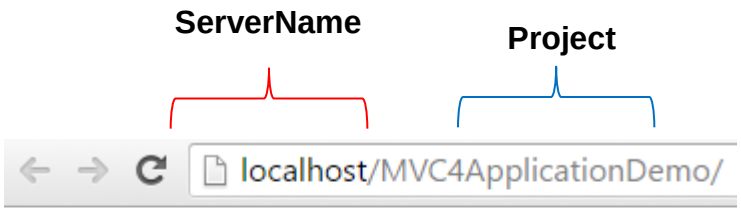


Hello from Webforms application

```
protected void Page_Load(object sender, EventArgs e)
{

    Response.Write("Hello from Webforms application");

}
```

MVC

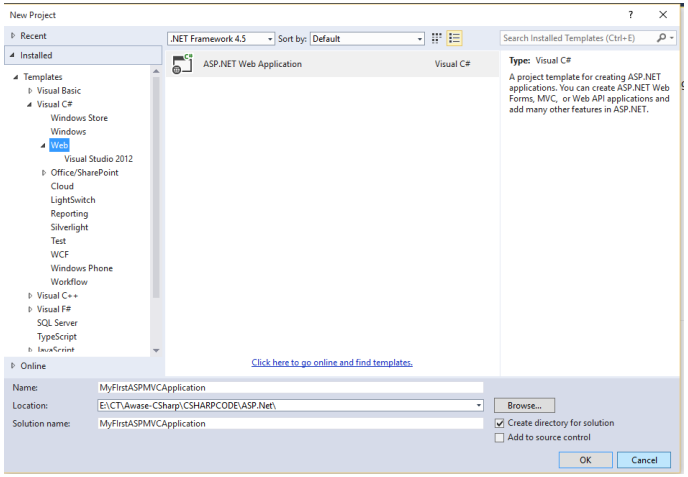
- MVC URL's are mapped to **<u>controller Action Methods</u>**
- Functions in a controller are generally called as **Controller Action Methods**

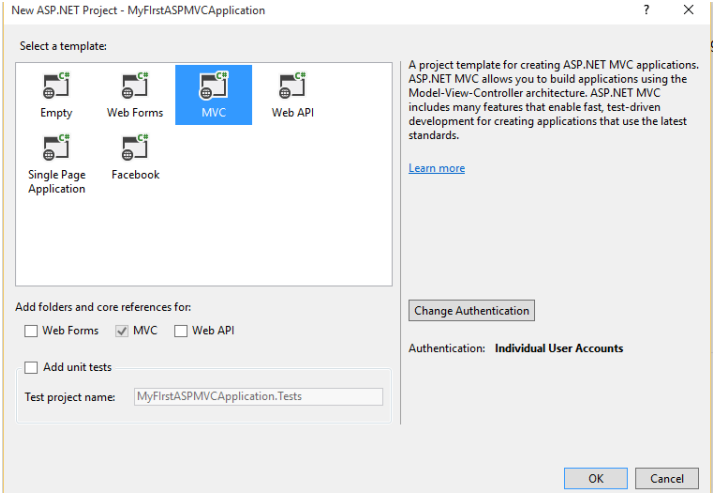


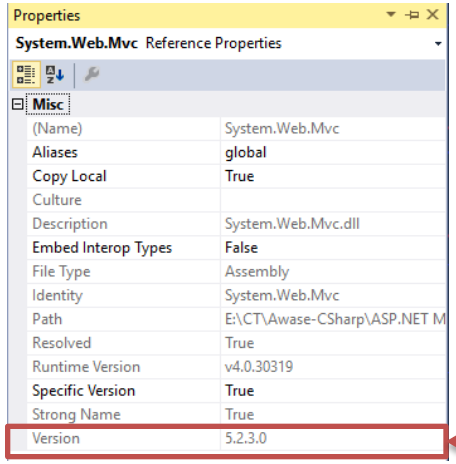
Hello from MVC4 Application using razor Engine

```
0 references
public String Index()
{
    //ViewBag.Message = "Modify this template to jump-start your ASP.NET MVC
      application.";

    //return View();
    return "Hello from MVC4 Application using razor Engine";
}
```

MVC Application with VS







@designtime

Identifying MVC Version @ runtime using reflection

```
public ActionResult Index()
{
    //1. Getting the version of the MVC libraries at runtime using reflection.
    //2. Using ViewBag.Dictionary
    //3. Basic Controllers defined in the scaffolding template provided by Visual
      Studio
    ViewBag.MVCVersionQuery = typeof(Controller).Assembly.GetName
      ().Version.ToString();
    //4.Conventional view called by the Controller action method - Index()
      (without any parameters passed).
    //5.Using ViewData to pass on MVC Version Property
    ViewData["MVCversion"] = typeof(Controller).Assembly.GetName
      ().Version.ToString();
    //6. using model to pass on MVC Version Property
```

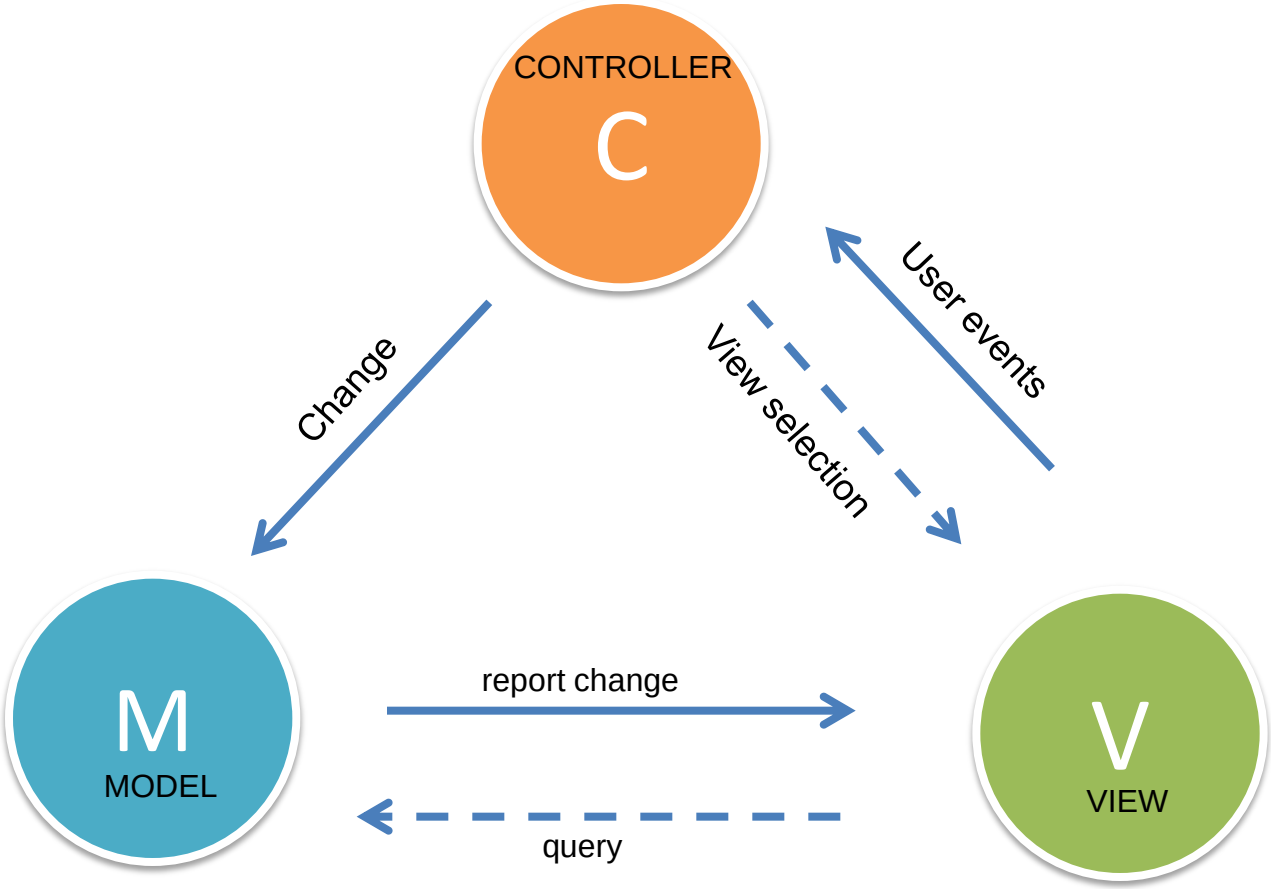
ASP.NET MVC DESIGN GOALS

- Does not replace web forms
  - An alternative project type
- Still runs on ASP.NET
  - Caching
  - Modules
  - Master pages
  - Providers
  - Handlers
  - Session state
- Embrace the web
  - No illusion of state – no page life cycle
  - Clean URLS and clean HTML
- Extensible
  - Pluggable view engines
  - Controller factories
- Testable
  - Maintains a strict separation of concerns

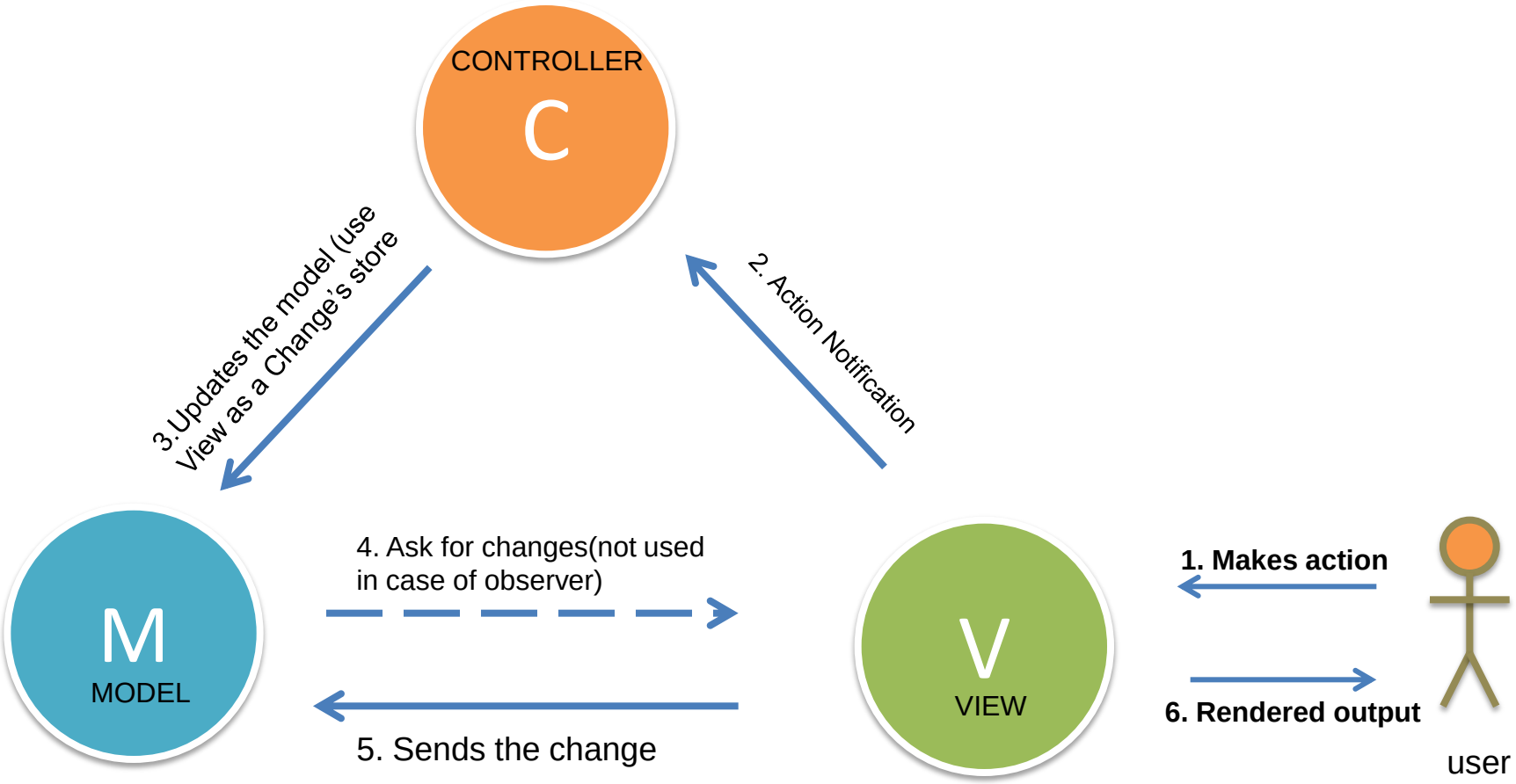
Features of MVC

- Separation of application tasks viz.business logic, UI logic and input logic
- Supports Test Driven Development
- Highly Testable framework
- Powerful URL-mapping component for comprehensible and searchable URLs
- Support existing ASP.net features viz.authentication, authorization, memberships and roles, caching, state management, configuration, health monitoring etc..

MVC DESIGN PATTERN



MVC DESIGN PATTERN



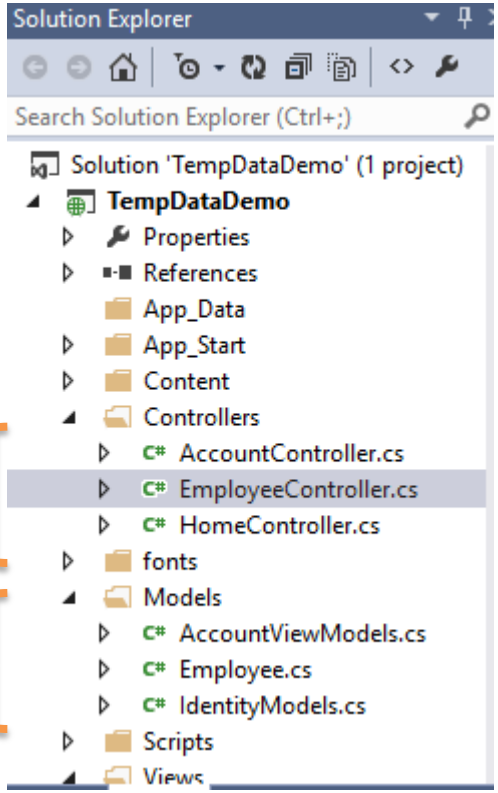
MVC DESIGN PATTERN

- Controller initializes the events of View interface to interact with model and controller.
- The user interacts with the View (UI)
- Controller handles user's events (can be the "observer" pattern) and asks Model to update.
- Model raises events, informing subscribers (View) about changes
- View (UI) (subscribes to model events) handles Model's events and shows new Model's data.
- The UI user interface waits for further user actions

MODELS in ASP.NET MVC

```
2 references
public class Employee
{
    1 reference
    public int EmployeeId { get; set; }
    1 reference
    public string Name { get; set; }
    1 reference
    public string Gender { get; set; }
    1 reference
    public string city { get; set; }
}
```

```
0 references
public class EmployeeController : Controller
{
    //
    // GET: /Employee/
    0 references
    public ActionResult Index()
    {
        Employee employee = new Employee
        {
            EmployeeId = 101,
            Name = "John",
            Gender = "male",
            city = "London"
        };
        return View(employee);
    }
}
```

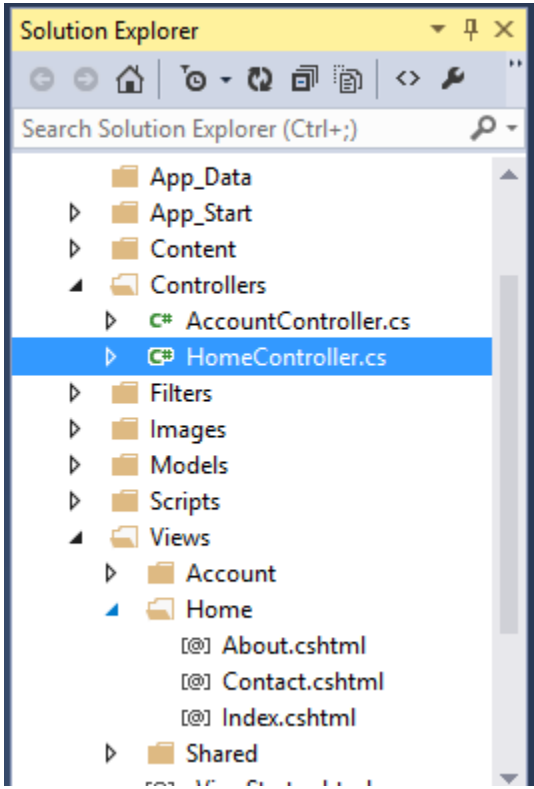


Controllers in an MVC Application

Controller action method

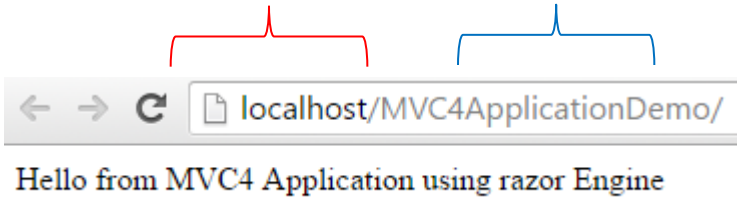
```
namespace MVC4ApplicationDemoTwo.Controllers
{
    0 references
    public class HomeController : Controller
    {
        0 references
        public ActionResult Index()
        {
            ViewBag.Message = "Modify this template to jump-start your ASP.NET MVC
              application.";

            return View();
        }
```

Controller



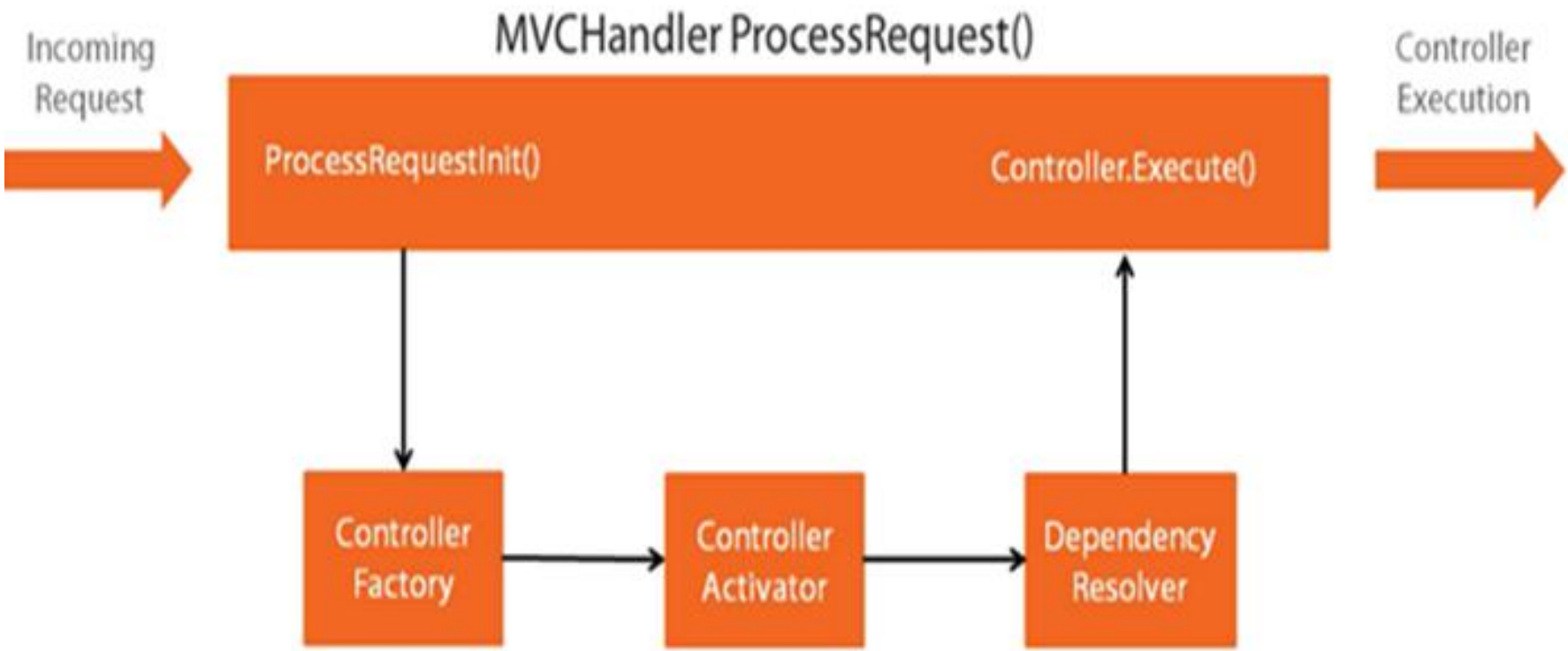
Views

ServerName **Project**



How are the URL's mapped to Controller action Method? (ActionResult Index())?

Controller Initialization

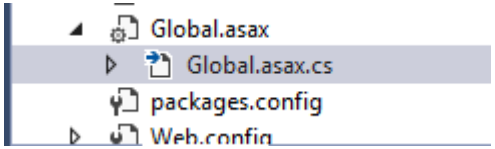


MVC Controllers

- Public methods are "**actions**"
  - Method invoked by ASP.NET once it determines the proper route
  - Controller can build the model and place in ViewData
  - Return value of ActionResult tells the framework where to go next

```
5 references
public class HomeController : Controller
{
    1 reference
    public ActionResult Index()
    {
        return View();
    }

    1 reference
    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";

        return View();
    }

    1 reference
    public ActionResult Contact()
    {
        ViewBag.Message = "Your contact page.";

        return View();
    }
}
```

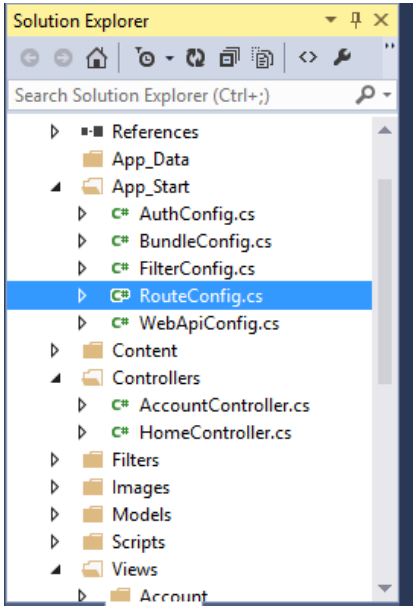
MVC Routing



- System.Web.Routing
  - Part of ASP.NET and released with .NET 3.5 SP1
- Directs incoming request to and MVC controller
  - Defines routes during application startup
  - Map URLs to controller action with parameters

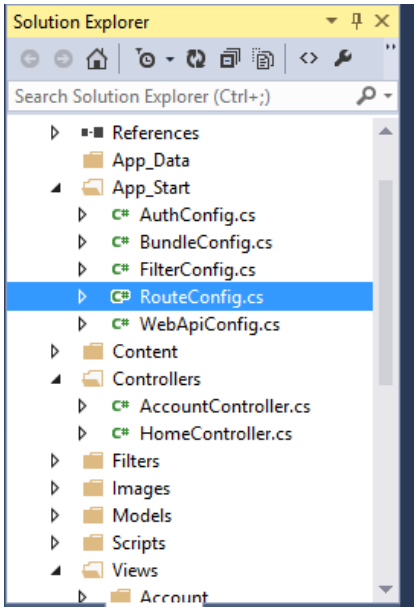
```
0 references
protected void Application_Start()
{
    AreaRegistration.RegisterAllAreas();

    WebApiConfig.Register(GlobalConfiguration.Configuration);
    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);
    AuthConfig.RegisterAuth();
}
```

```
routes.MapRoute(
        "Default",                          // Route name
        "{controller}/{action}/{id},        //URL with parameters
        new {
                controller = "Home",
                action = "Index",
                id = UrlParameter.Optional
        } ;             //Parameter defaults
```



MVC Routing

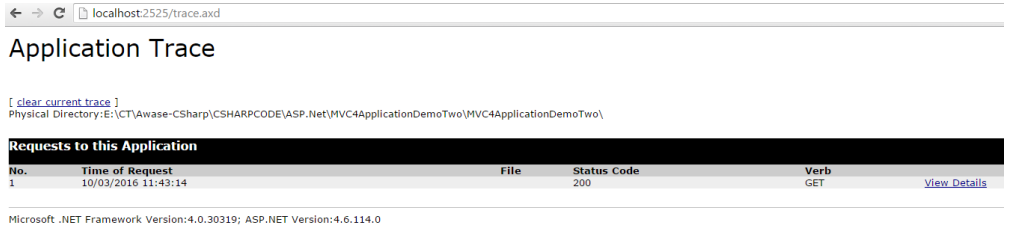
```
1 reference
public class RouteConfig
{
    1 reference
    public static void RegisterRoutes(RouteCollection routes)
    {
        routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

        routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "Home", action = "Index", id =
              UrlParameter.Optional }
        );
    }
```

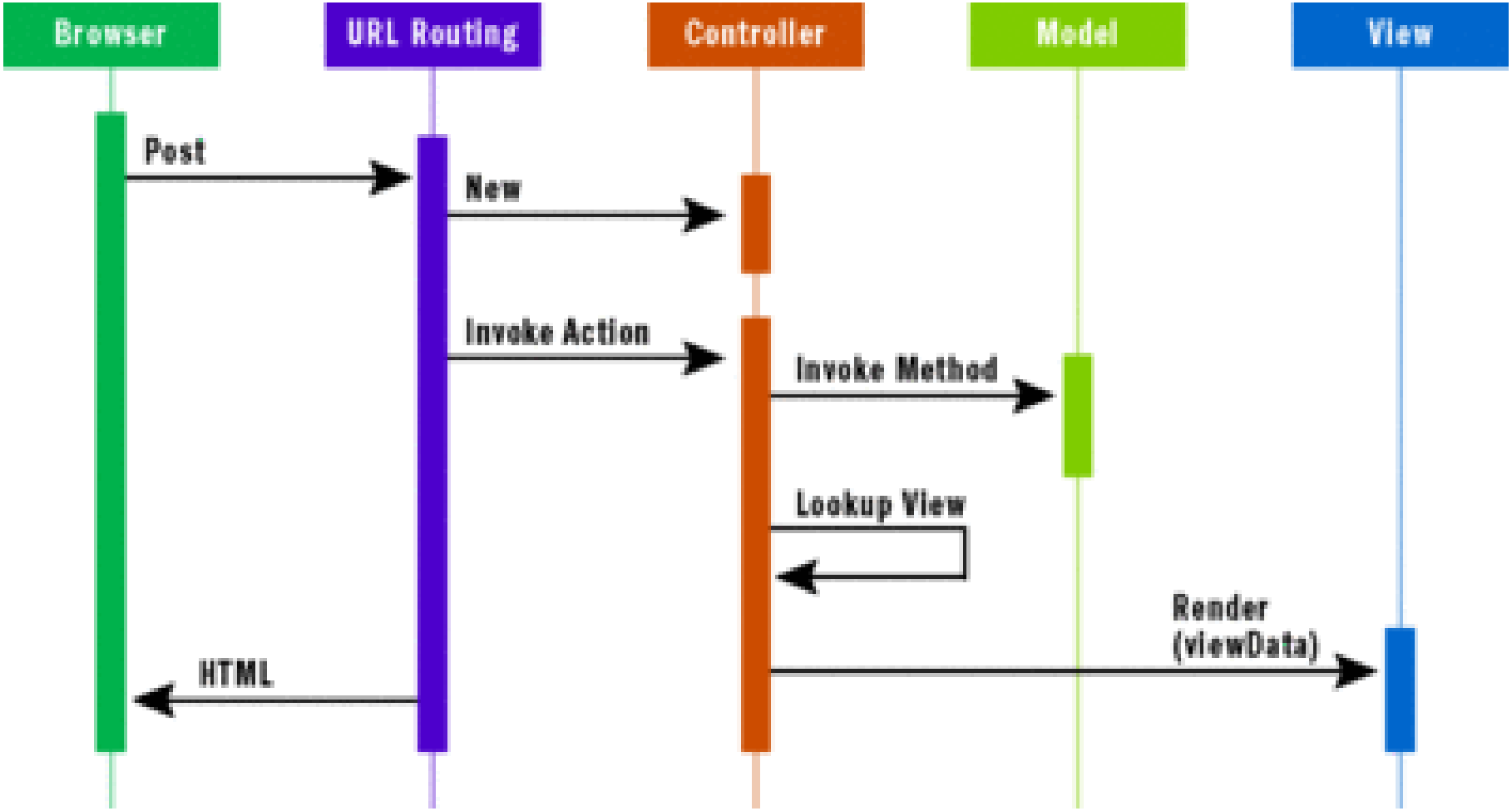


Enable trace in web.config

```
<system.web>
  <compilation debug="true" targetFramework="4.0" />
    <trace enabled="true" pageOutput="false"/>
```

http://localhost:2525/trace.axd

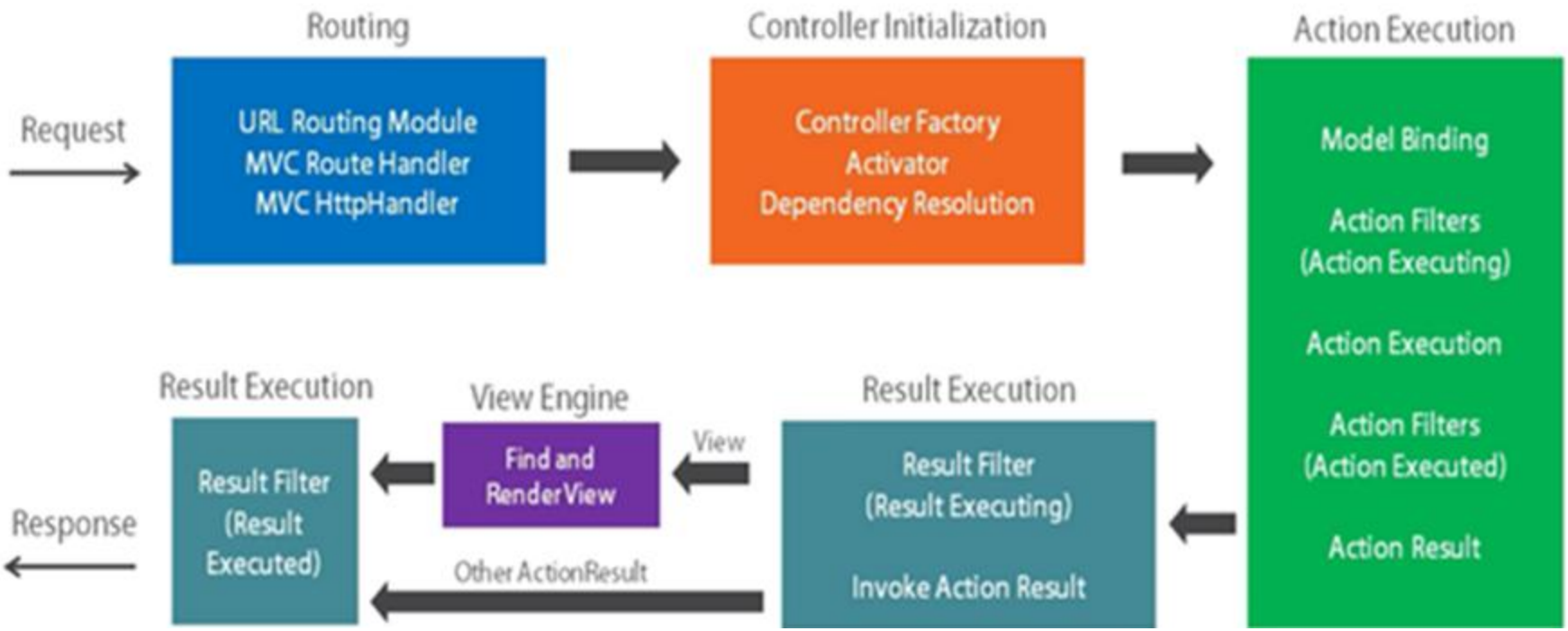




Stages of Request Execution

Stage	Description
httpRequest to the Application	**Global.asax** file, Route Objects are added to the **Route Table** Object
Perform routing	**UrlRoutingModule** uses the first matching Route Object in the **RouteTable** collection to create the **RouteData** Object, which it then uses to create a **RequestContext** Object
Create MVC request handler	**MvcRouteHandler** object creates an instance of the MvcHandler class and passes the RequestContext instance to the handler
Create Controller	**MvcRouteHandler** object uses the RequestContext instance to identify the IControllerFactory object (typically an instance of the DefaultControllerFactory class) to create the controller instance with.
Execute Controller	MvcHandler instance calls the controller's EXECUTE method
Invoke Action	For controllers that inherit from the ControllerBase class. The ControllerActionInvoker object that is associated with the controller determines which action method of the controller calss to call and then calls that method
Execute Result	Action method receives user input, prepares appropriate response data and then executes the result by returning a result type. The built-in result type that can be executed include the following: ViewResult(which renders a view and is most often used result type). RedirectToRouteResult, RedirectResult, ContentResult, JsonReseult, FileResult and EmptyResult.

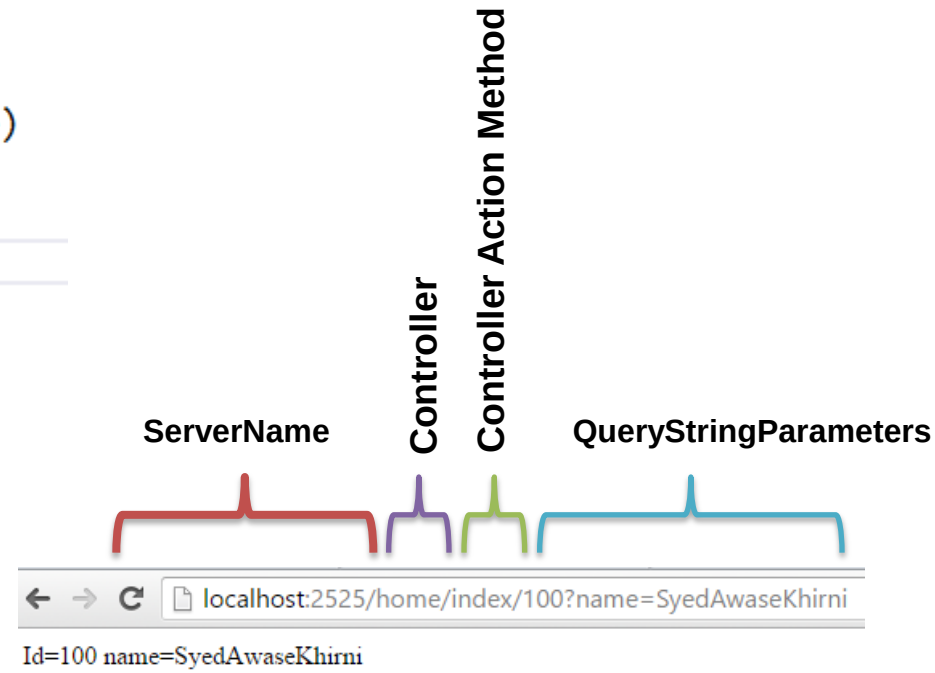
The MVC Request Life Cycle



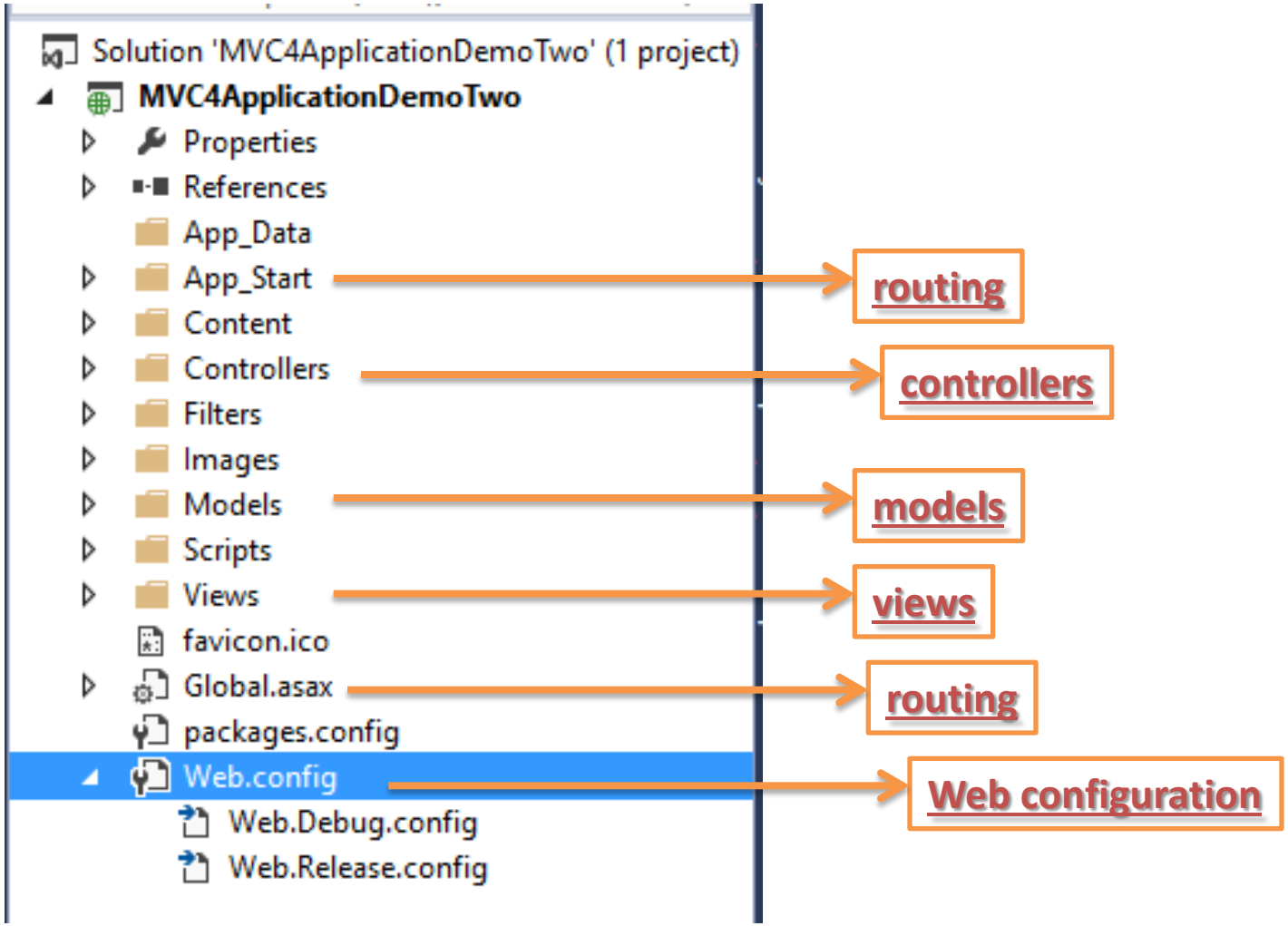
ASP.NET MVC FORM POST PARAMETERS

- ASP.NET MVC will automatically pass any query string or form post parameters named “name” to Index action method when it is invoked.

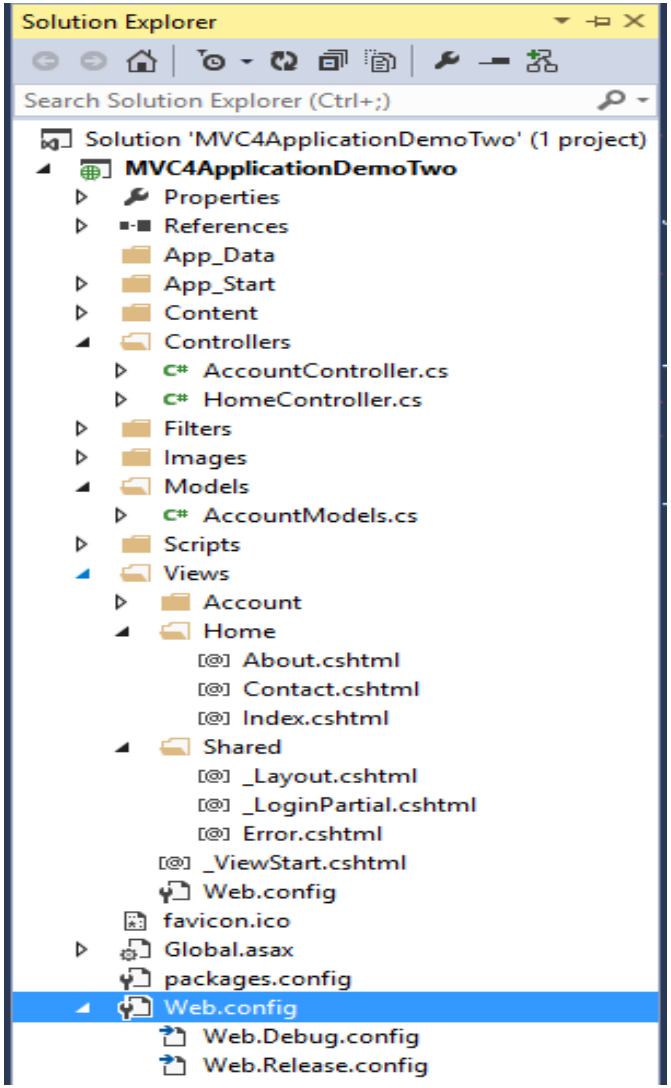
```
0 references
public class HomeController : Controller
{
    0 references
    public string Index(string id, string name)
    {
        return "Id=" + id +" name=" +name;
    }
```



Controller and View Conventions



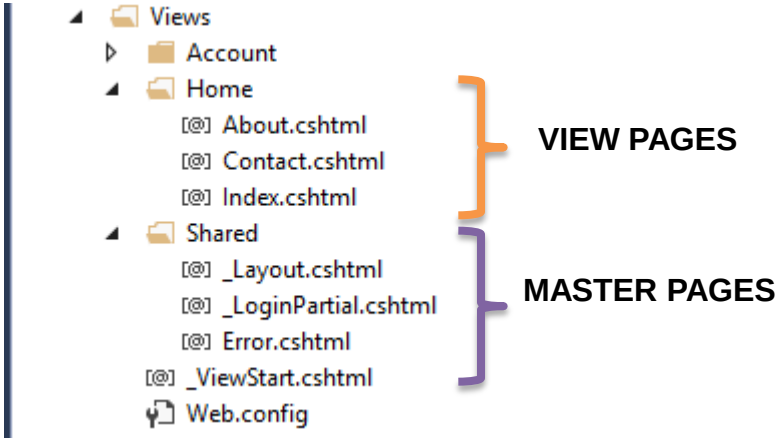
Controller and View Conventions



- **Controllers folder**
  - Recommended location for controllers
  - Controller type name must end with **Controller(thus omitted from the route)**
- **Views folder**
  - Recommended location for views
  - .aspx, .ascx, .master files
  - Subfolders for every controller
  - Shared folder contains views used by multiple controllers

Views

- **Views are .aspx files**
  - Derive from ViewPage, which derives from Page
  - Have a ViewData dictionary property populated from the controller
  - Still use markup, can still contain server-side script
  - No server-side form required
    - No _VIEWSTATE
    - No control state
- Strongly types views
  - Derive from ViewPage<T> instead of ViewPage
  - Generic type Parameter represents the type of the model



Views

- ViewData, ViewBag and TempData mechanisms to pass on data from the **Controller** to the **View.**
- **To pass data from Controller to a View.** ***<u>It's always a good practice to use strongly typed view models</u>***
- ***"@" symbol is used to switch between html and C# code***

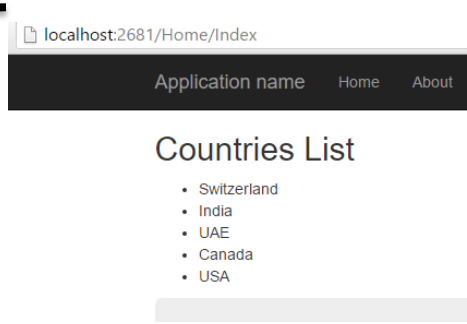
```
public class HomeController : Controller
{
    0 references
    public ActionResult Index( string id, string name)
    {
        ViewBag.Countries = new List<string>()
        {
            "Switzerland",
            "India",
            "UAE",
            "Canada",
            "USA"

        };

        return View();
    }
```

Controller Code

View Code (in html)

```
@{
    ViewBag.Title = "Countries.List";
}

<h2> Countries List</h2>
<ul>
    @foreach (string strCountry in ViewBag.Countries)
    {
        <li>@strCountry</li>
    }

</ul>
```

output



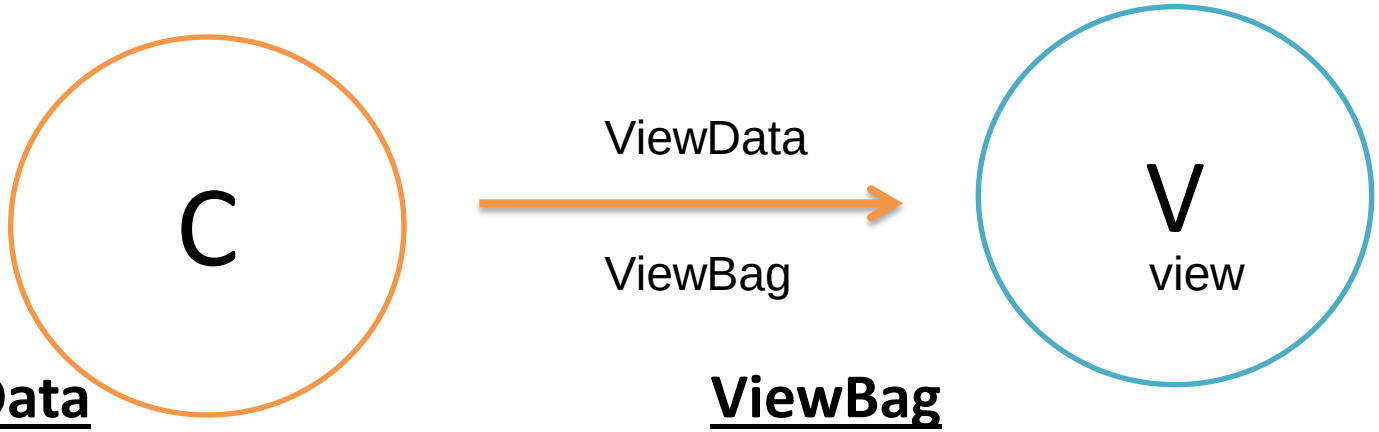
Stongly Typed View

- ***It's always a good practice to use strongly typed view models***
- ***ViewData and ViewBag are used to pass data from a controller to a view.***
- ***Strongly types view models provide compile time error checking.***

```
0 references
public class HomeController : Controller
{
    0 references
    public ActionResult Index()
    {
        return View();
    }

    0 references
    public ActionResult About()
    {
        ViewBag.Message = "Your application description page.";

        return View();
    }
```

Both ViewData and Viewbag do not Provide compile time error checking

ViewData vs ViewBag



ViewData

- ViewData is a dictionary of objects that is derived from ViewDataDictionary class and is accessible using strings as keys.
- ViewData requires typecasting for complex data type and check for null values to avoid error.

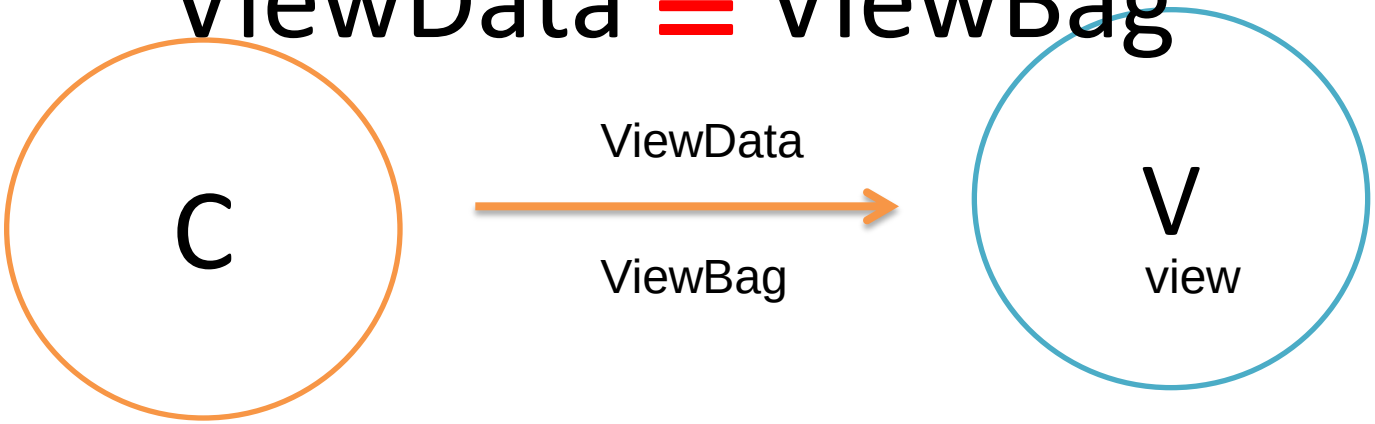
```
ViewData["YourKey] = "YourData";
```

ViewBag

- A dynamic feature introduced in C# 4.0
- Allows an object to have properties dynamically added to it.
- ViewBag doesn't require typecasting for complex data type.

```
ViewBag.YourProperty="YourData";
```

ViewData ≅ ViewBag



- Similarities between ViewBag & ViewData:
  - Helps to maintain data when you move from controller to view.
  - Used to pass data from controller to corresponding view.
  - **Short life** means <u>value becomes null when redirection occurs</u>. This is because their ***goal is to provide a way to communicate between controllers and views***. It's a communication mechanism within the server call.

TEMPDATA

- A dictonary derived from TempDataDictonary Class and stored in short lives Session and it is a string key and object value.
- The difference is the lifecycle of the object.
- Works with 302/303 redirection because it's in the same HTTP Request.
- **Used to move data from one controller to other controller or from one action to other action.**
- **On redirection, "Tempdata" helps to maintain data between those redirects.**

- **Internally uses session variables.**
- **Tempdata is used during the current and subsequent request only, means it is used when you are sure that next request will be redirecting to next view.**
- **Requires typecasting for complex data type and check for null values to avoid error.**
- **Used to store only one time messages like error messages, validation messages**

Scenarios when to use ViewBag, ViewData and TempData

- ViewBag and View Data object work well in the following scenarios:
  - Incorporating dropdown lists of lookup data into an entity
  - Components like a shopping cart
  - Widgets like a user profile widget
  - Small amounts of aggregate data
- TempData object works well in **one basic scenario**:
  - Passing data between the current and next HTTP requests

View Helpers

- Helpers available via properties of a ViewPage

Property	Class	Description
Ajax	AjaxHelper	Invoke controller actions asynchronously and update client content
Html	HtmlHelper	Create anchor tags, encode HTML
Url	UrlHelper	Create URLs to invoke controller actions

```
<div>
    @Html.ActionLink("About Us", "About", "Home");
</div>
```

→

About us

Preparing Models

- Attributes
  - Decorate properties
- Available Attributes
  - DataType Attribute
  - Display Attribute
  - Validation
    - RequiredAttribute
    - StringLength Attribute
    - RegularExpressionAttribute
    - CompareAttribute

ASP.NET WEBFORM vs MVC

Features	Web Forms	ASP.NET MVC
Separation of concerns	NO	YES
Familiar Event Driven Model	YES	NO
ViewState Issues	YES	NO
Server Controls	Yes	No
Control over HTML	No	Yes
Test Driven Development	No	Yes

ASP.NET WEBFORM vs MVC

ASP.NET Web Forms	ASP.NET MVC
ASP.NET Web Forms **uses Page controller pattern** approach for rendering layout. In this approach, every page has it's own controller i.e. code-behind file that processes the request.	ASP.NET MVC **uses Front Controller approach**. That approach means ,a common controller for all pages, processes the requests.
No separation of concerns. As we discussed that every page (.aspx) has it's own controller (code behind i.e. aspx.cs/.vb file), so both are tightly coupled.	Very clean separation of concerns. View and Controller are neatly separate.
Because of this coupled behavior, automated testing is really difficult.	Testability is key feature in ASP.NET MVC. Test driven development is quite simple using this approach.
In order to achieve stateful behavior, viewstate is used. Purpose was to give developers, the same experience of a typical WinForms application.	ASP.NET MVC approach is stateless as that of the web. So here no concept of viewstate.
Statefulness has a lots of problem for web environment in case of excessively large viewstate. Large viewstate means increase in page size.	As controller and view are not dependent and also no viewstate concept in ASP.NET MVC, so output is very clean.

ASP.NET WEBFORM vs MVC

ASP.NET WebForms model follows a Page Life cycle.	No Page Life cycle like WebForms. Request cycle is simple in ASP.NET MVC model.
Along with statefulness, microsoft tries to introduce server-side controls as in Windows applications. Purpose was to provide somehow an abstraction to the details of HTML. In ASP.NET Web Forms, minimal knowledge of HTML, JavaScript and CSS is required.	In MVC, detailed knowledge of HTML, JavaScript and CSS is required.
Above abstraction was good but provides limited control over HTML, JavaScript and CSS which is necessary in many cases.	Full control over HTML, JavaScript and CSS.
With a lots of control libraries availability and limited knowledge of other related technologies, ASP.NET WebForms is RAD(Rapid Application Development) approach.	It's a step back. For developers decrease in productivity.
It's good for small scale applications with limited team size.	It's better as well as recommended approach for large-scale applications where different teams are working together.

WebForms is an abstraction of web application programming; envisaged to ease the transition for the Visual Basic programmers who were the primary target for .NET when it was launched. Uses VB's event-based model, such as viewstate and postbacks	
Concurrency between multiple clients goes out the window.	
It is stored as a massive chunk of data in a hidden field leading to increased page load sizes.	
It is wholly dependent on POST methods, so it breaks navigation in HTML.	

Summary

- ASP.NET MVC is an alternative to WEB FORMS
  - Builds on ASP.NET, does not replace ASP.NET
- Strives for simplicity
  - Clean URLs
  - Clean HTML
- Separation of concerns
  - It's what Model, View, Controller is about.