Request Management

Alex Wolf

www.alexwolfthoughts.com

To-Do List

The Shared ASP.NET Platform

Demo – Exploring the Application Life Cycle

Handling Requests with Web Forms

Demo – Touring the Legacy Application

Handling Requests with MVC

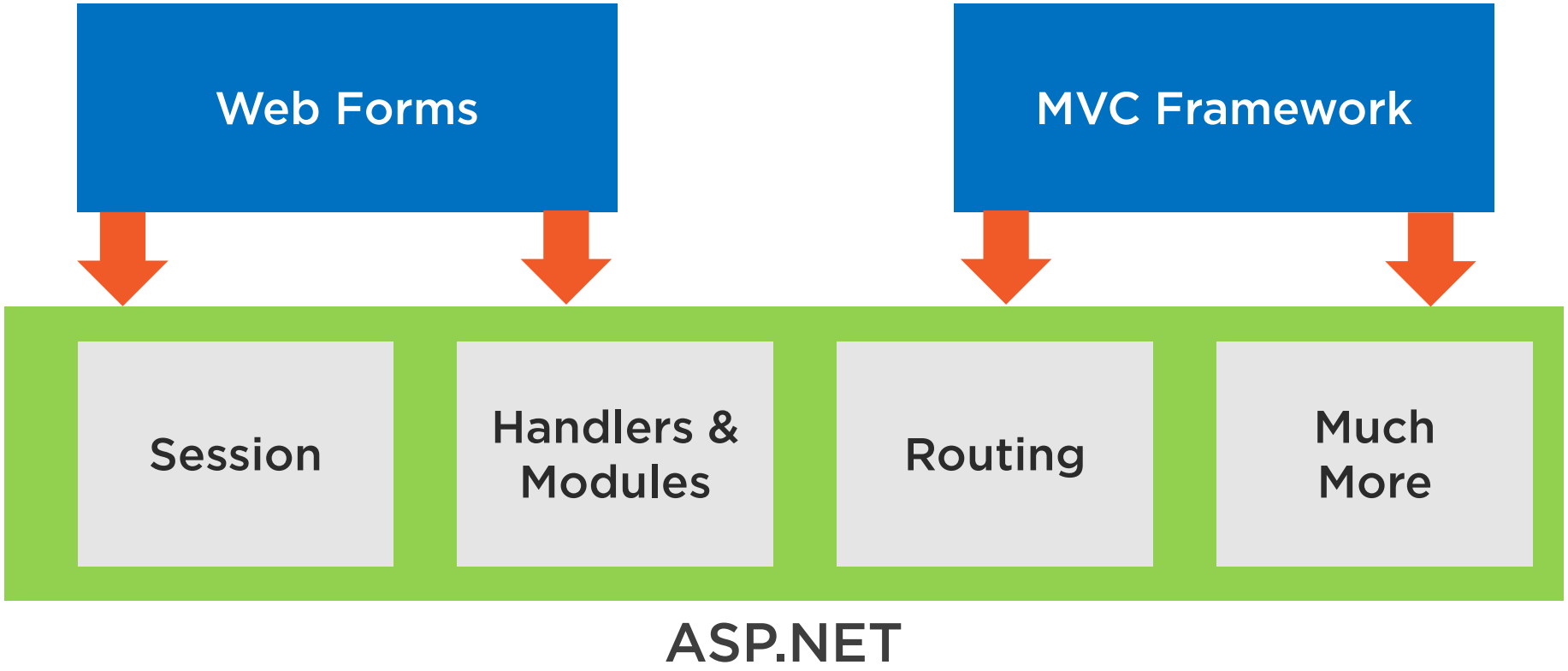
Understanding Routing

Demo – Routing and Requests in MVC

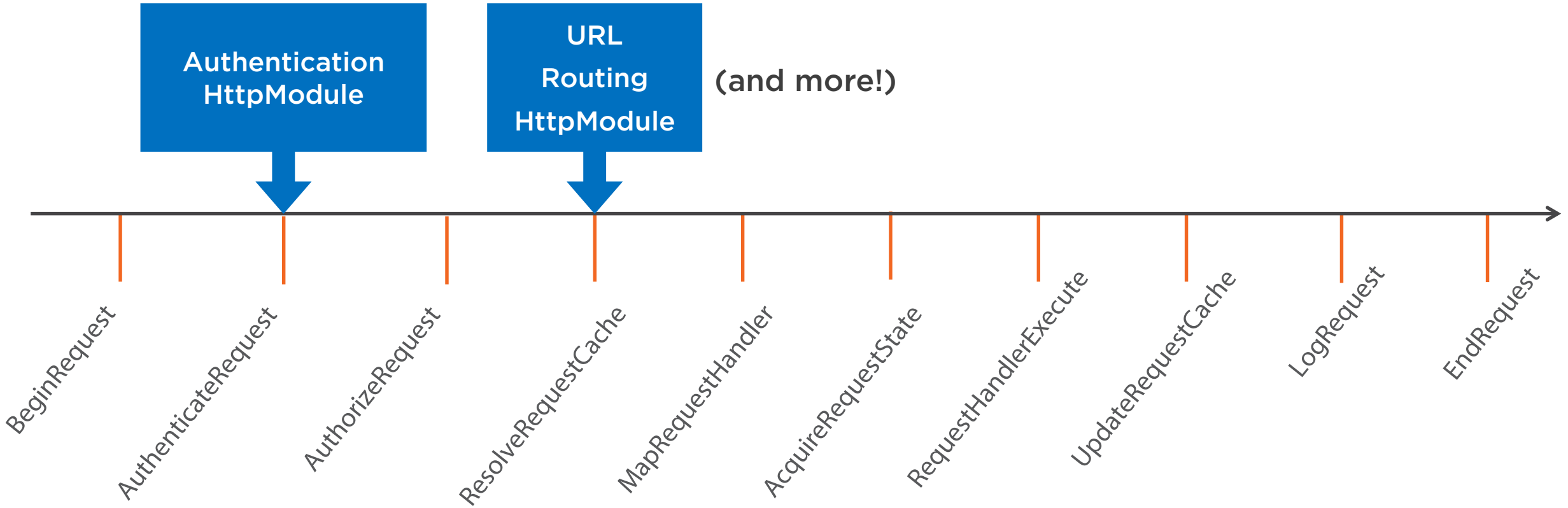
Demo – Building the Controller Structure

The Shared ASP.NET Platform

A Common Foundation



The ASP.NET Application Life Cycle



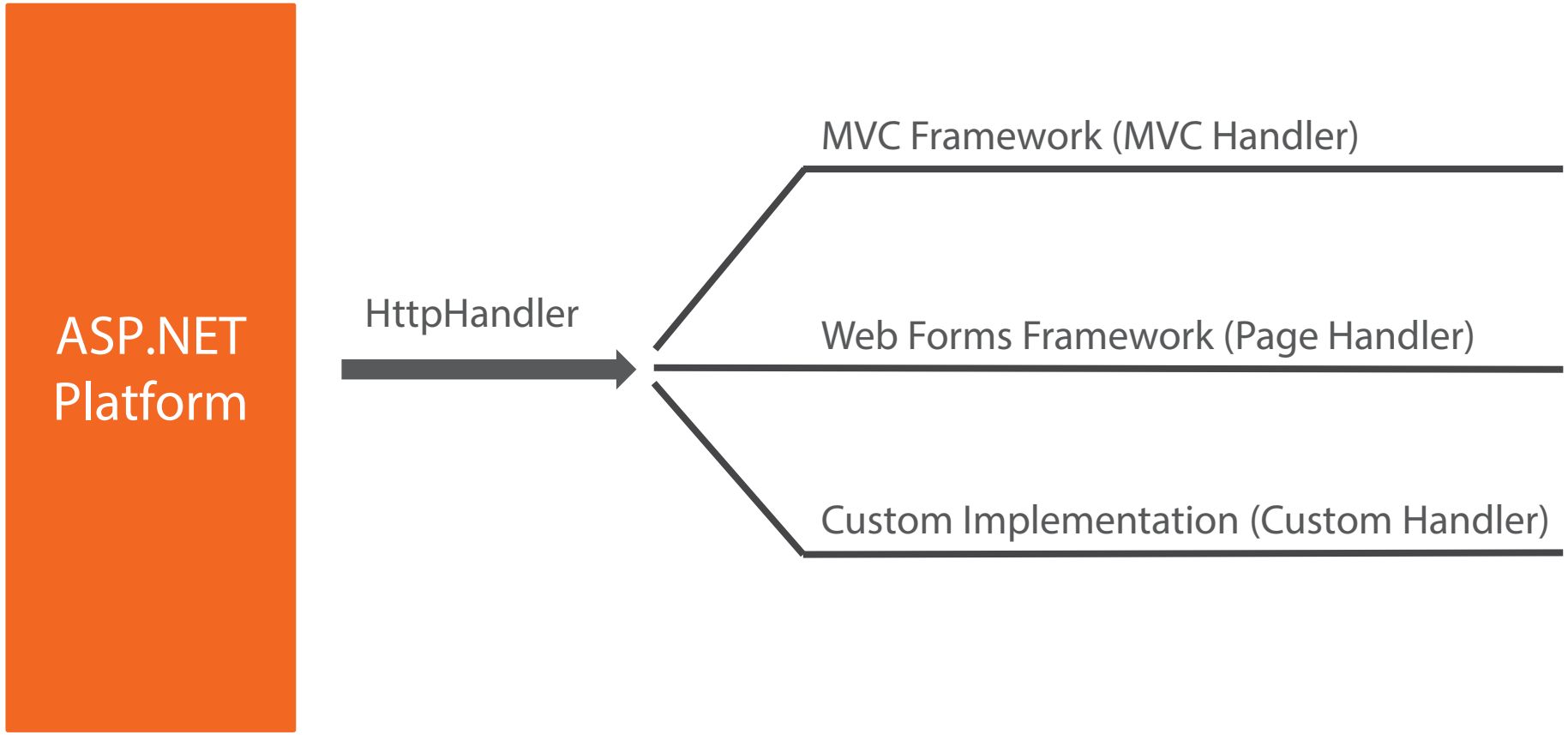
* Event names shown represent both pre and post events

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

Implementing an HttpModule

Not an overly common task, but just in case!

One Platform, Many Implementations

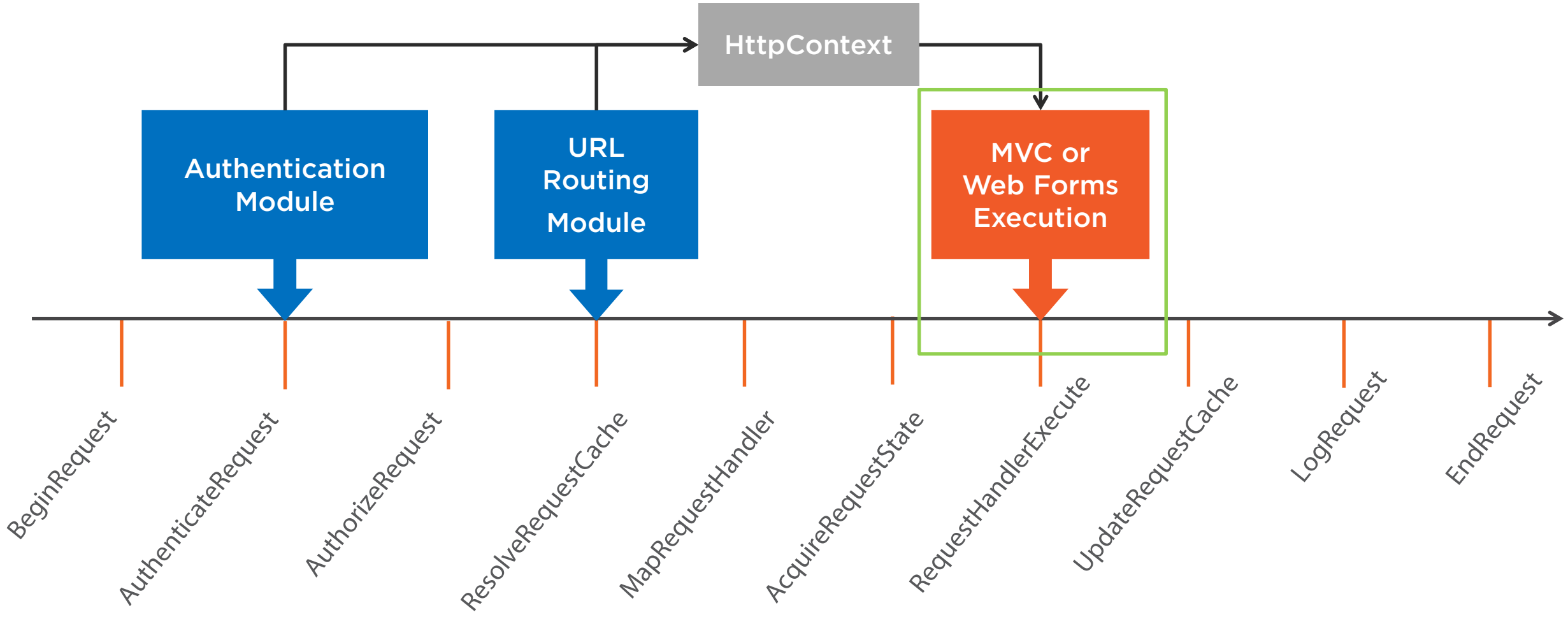


```
public interface IHttpHandler {
        bool IsReusable();
        void ProcessRequest(HttpContext context);
}
```

Implementing an HttpHandler

Remember, ProcessRequest ultimately generates the response

HttpHandlers and the Application Life Cycle



* Event names shown represent both pre and post events

Summarizing Modules and Handlers

HttpModules

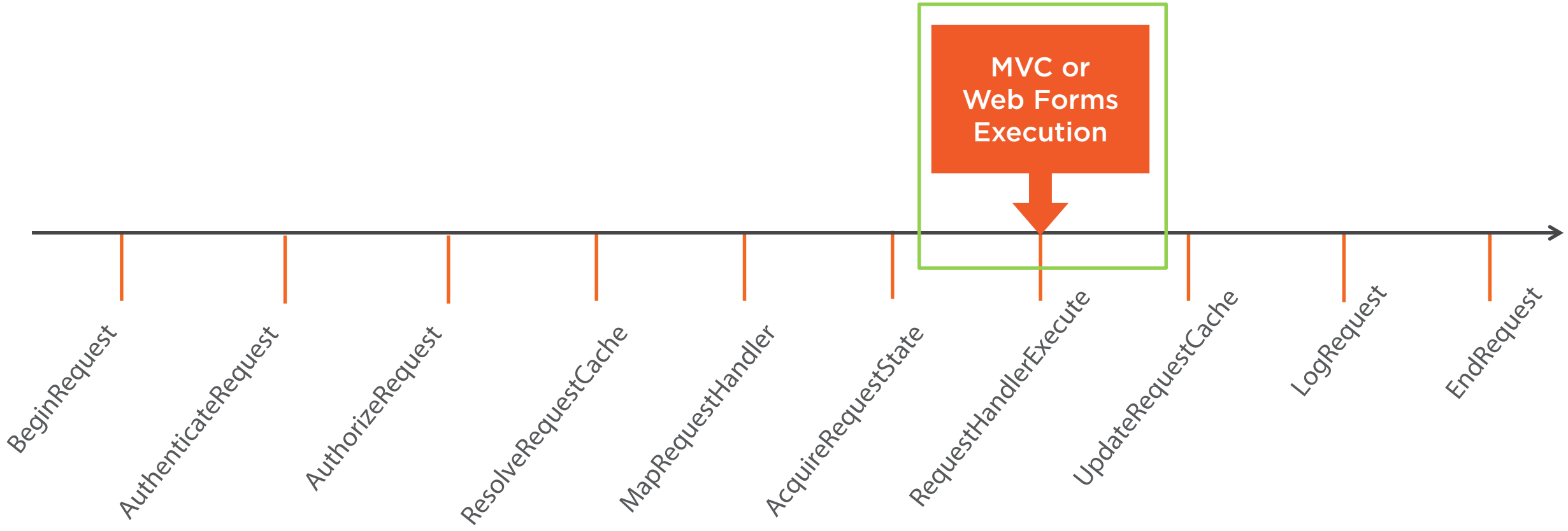
Hook into Application Level Events to provide supporting features

HttpHandlers

Selected and executed by ASP.NET to generate a response for a request

Handling Requests in Web Forms

HttpHandlers and the Application Life Cycle



* Event names shown represent both pre and post events

The Web Forms Page Life Cycle

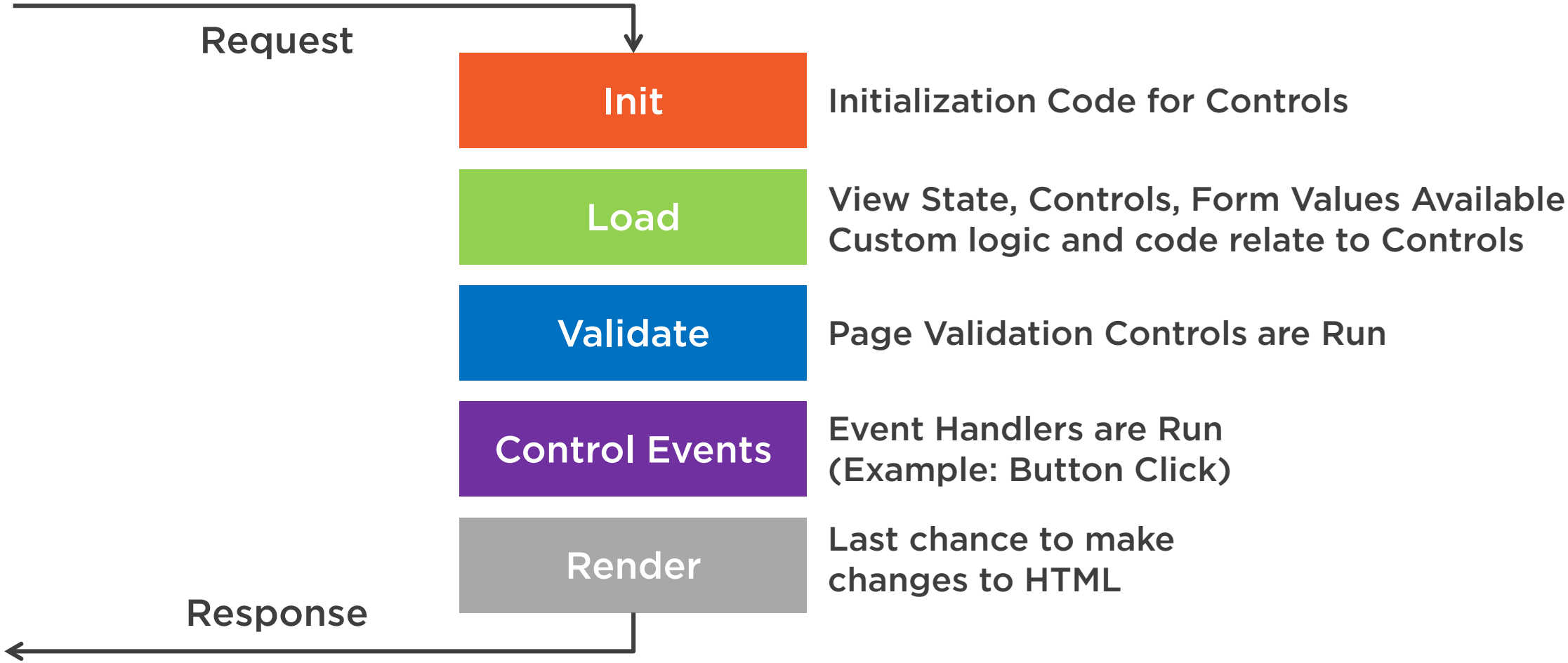
A series of page processing events

Abstraction over Http Requests

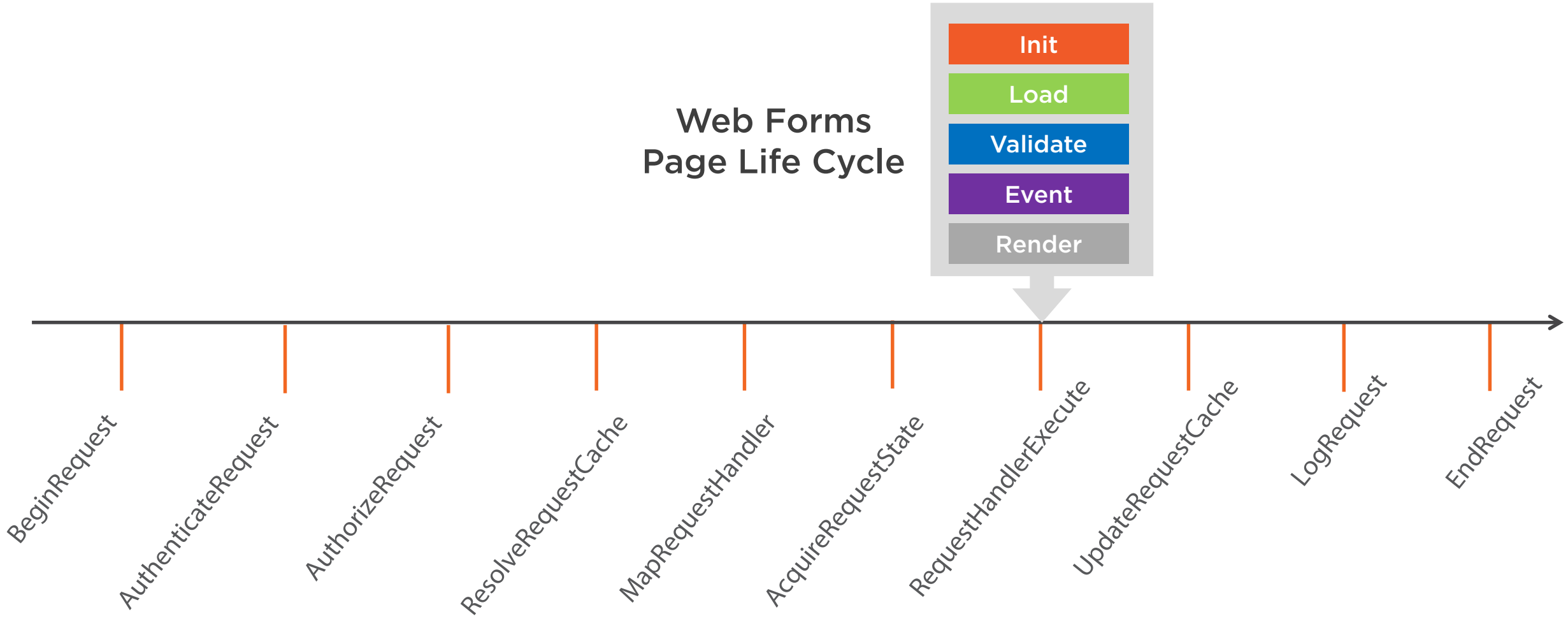
Ties together various features such as View State and Validation

Many events, but only a handful used day to day

The (abridged) Web Forms Page Life Cycle



Web Forms and the Application Life Cycle

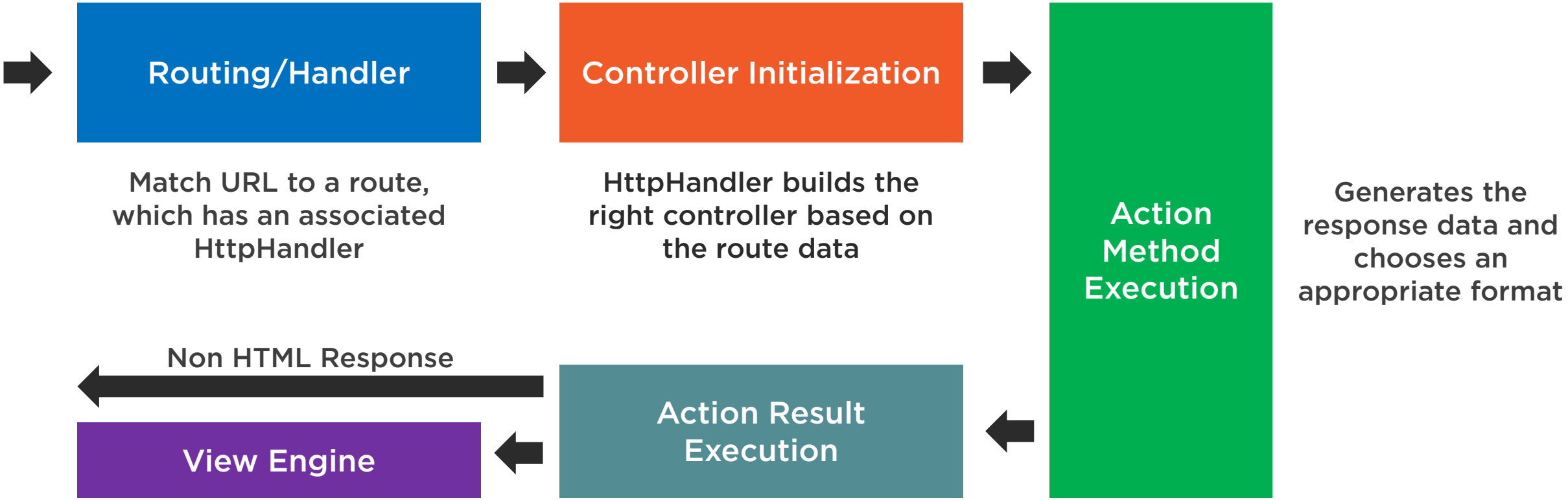


* Event names shown represent both pre and post events

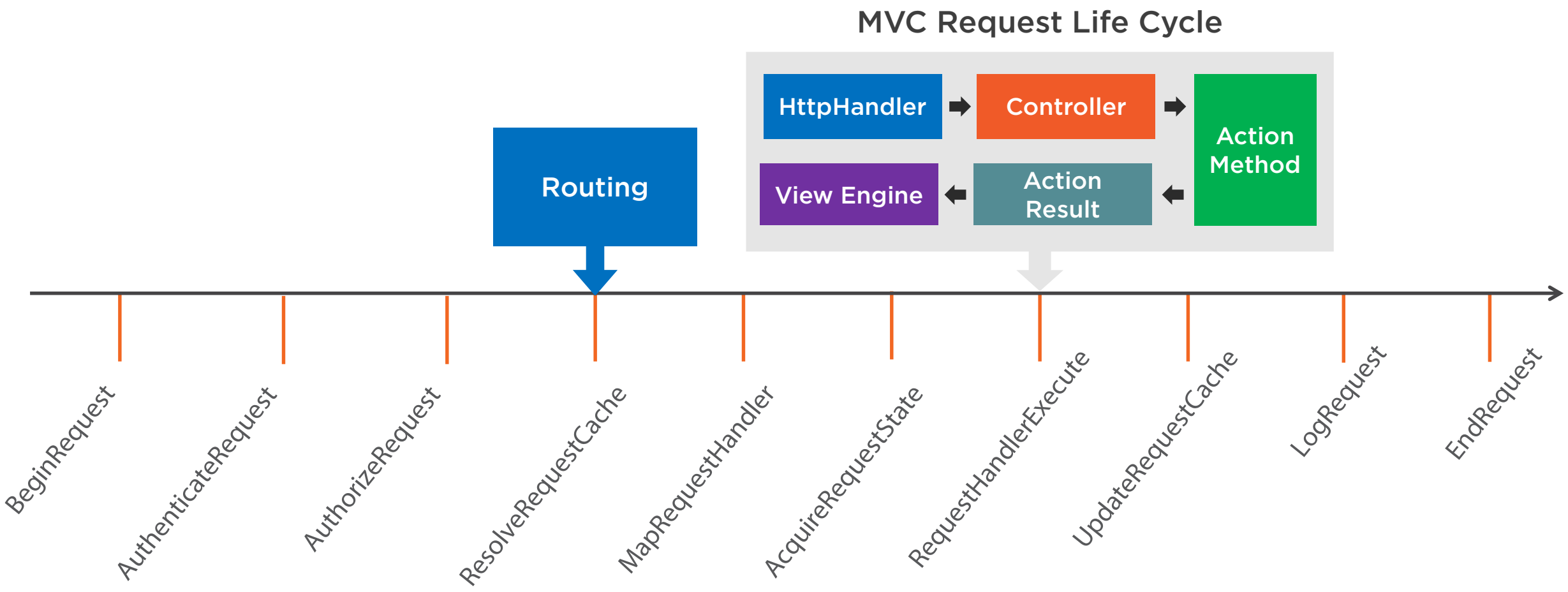
The (abridged) Web Forms Page Life Cycle

Handling Requests in MVC

The MVC Request Life Cycle



MVC and the Application Life Cycle



* Event names shown represent both pre and post events

```
public class MessagesController : IController{
        public void Execute(){  }
}
public class MessagesController : Controller {
        //Action Methods
}
```

Implementing a Controller

Use the interface yourself, or just inherit!

Handling Requests with Controllers

MySite.com/Task/Delete

MySite.com/Task/Create

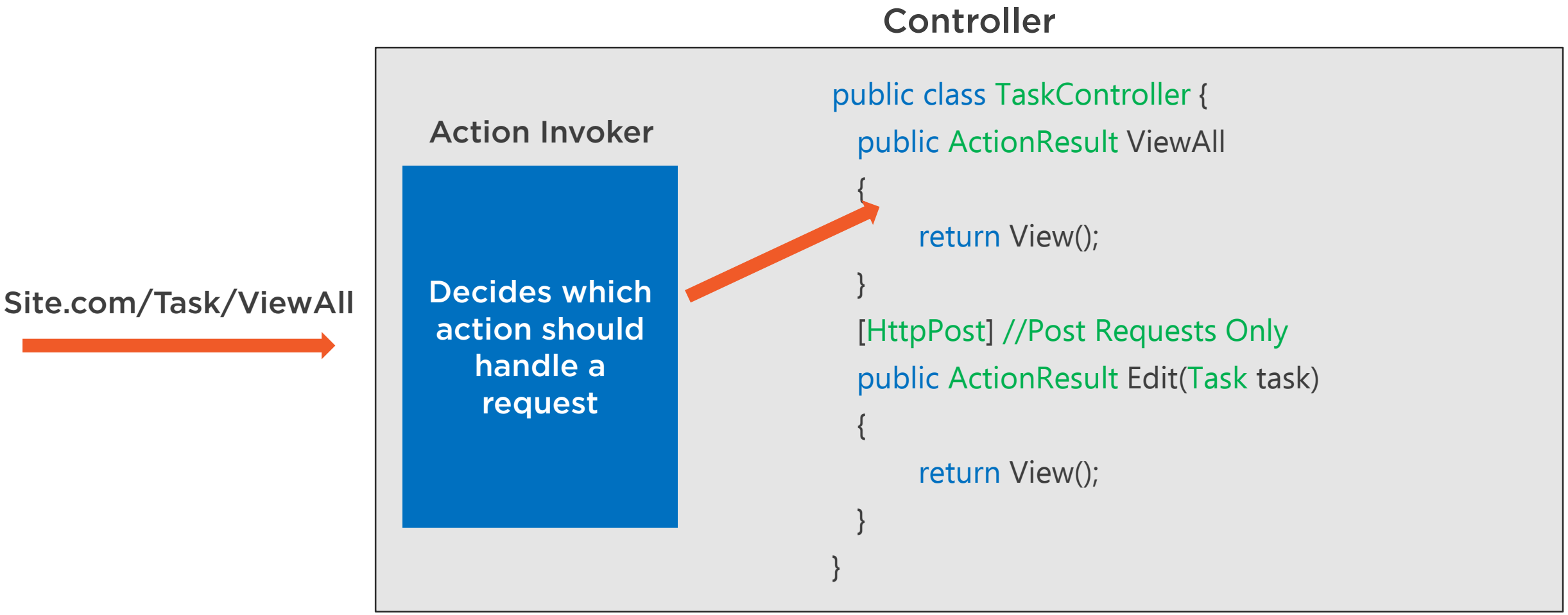
```
public class TaskController {

    public ActionResult Delete()
    {
            return View();
    }

    public ActionResult Create()
    {
            return View();
    }

}
```

HomeController

MessagesController

WidgetController

Mapping Requests to Action Methods



Understanding ActionResult Types

ViewResult

Parses Razor Syntax and returns HTML

JsonResult

Formats response as JSON data

ContentResult

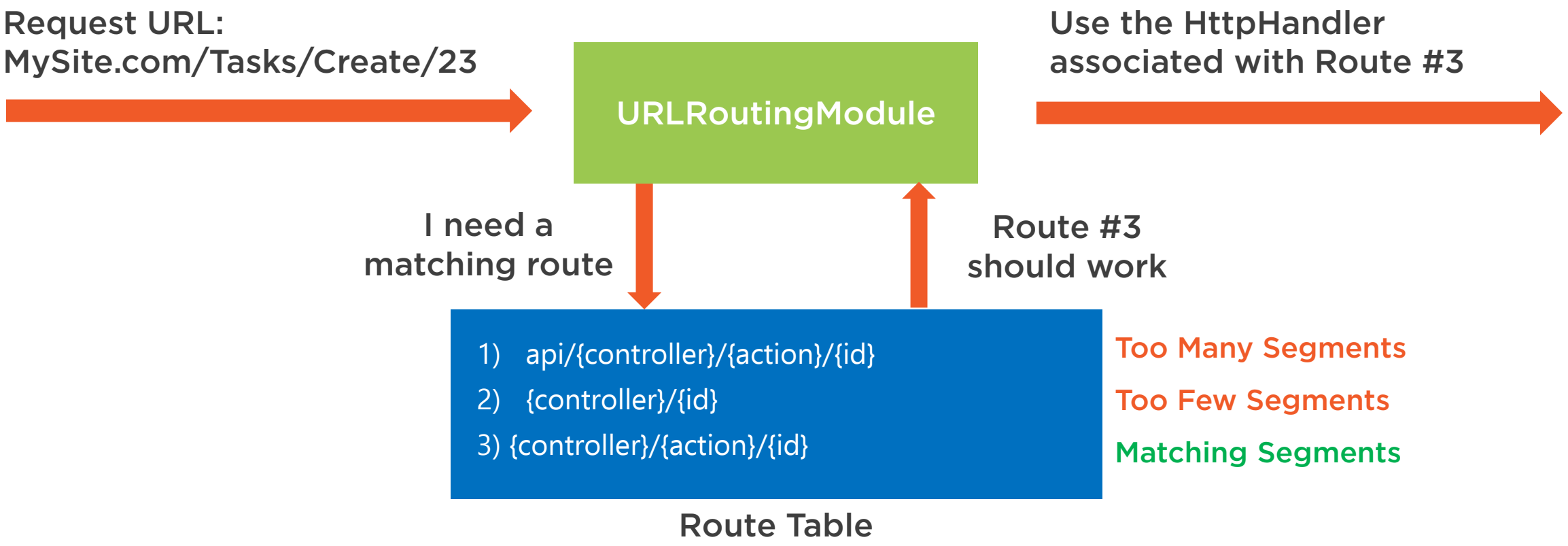
Writes out a string result

ActionResult

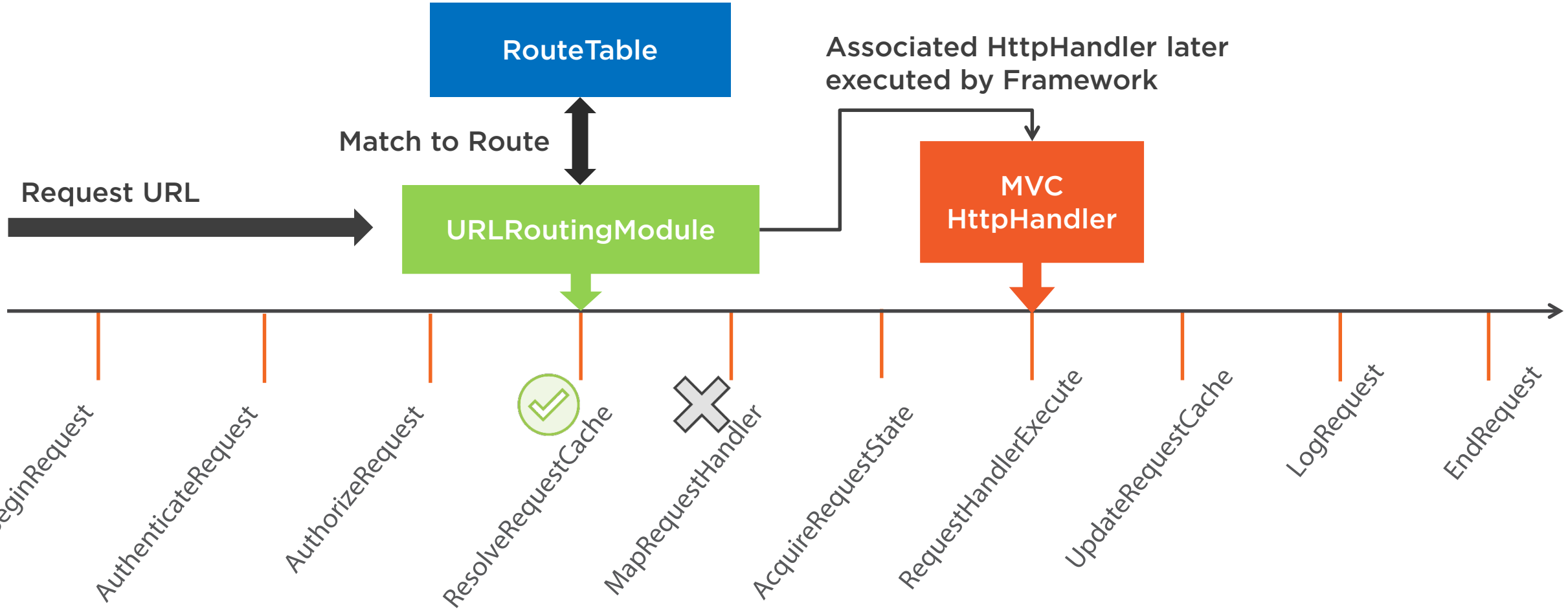
What About Routing?

Routing Requests

Understanding Route and HttpHandler Selection



The URLRoutingModule at Work



* Event names shown represent both pre and post events

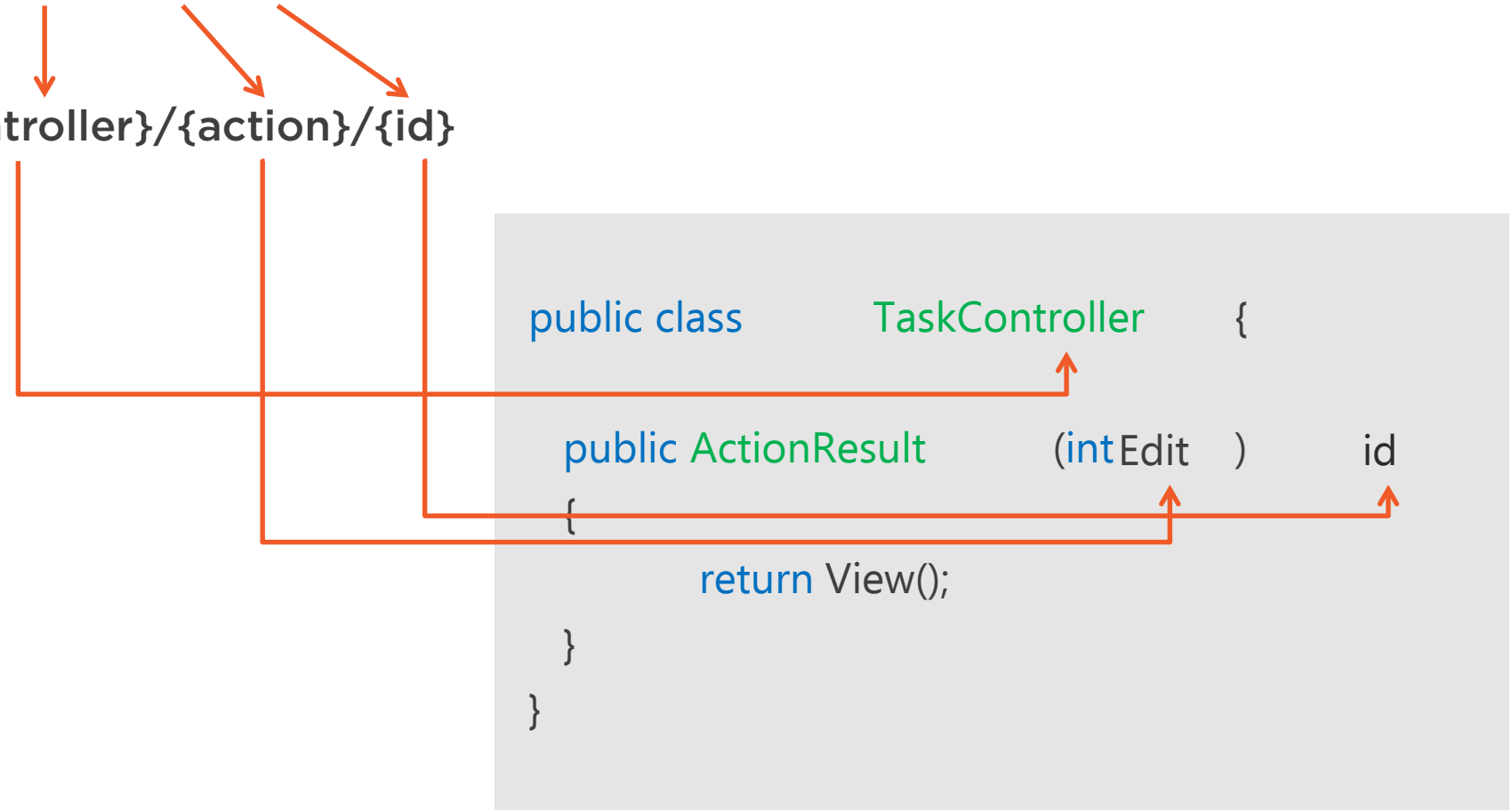
```
routes.MapRoute(
  name: "Default",

  url: "{controller}/{action}/{id}",

  defaults: new {
        controller = "Home",
        action = "Index",
        id = UrlParameter.Optional
        }
);
```

- Name of the route for easy referencing
- The URL segment pattern to match
- Default values can be provided as fall backs for missing segments
- Constraints that apply rules for whether a URL segment value is valid

Understanding Route Pattern Matching

URL: Mysite.com/Task/Edit/23

Matched Route: {controller}/{action}/{id}



The Route Ahead

Summary

Web Forms maps requests to physical pages in file directories

MVC dynamically handles requests by mapping them to action methods

Action Methods can return whatever data type and format is appropriate

MVC relies heavily on routing to process request information

ASP.NET provides considerable infrastructure for both frameworks