Building Web App's with

ASP.NET MVC FRAMEWORK

By: Venkat Shiva Reddy

MVC – Objectives

- Gain a thorough understanding of the philosophy and architecture of Web applications using ASP.NET MVC
- Acquire a working knowledge of Web application development using ASP.NET MVC and Visual Studio 201X
- Access databases using EF Code First Approach
- Understanding and Implementing Dependency Injection Resolution using nInject/Unity Framework
- Implement security in ASP.NET MVC applications
- Working with Async model

MVC – Pre-requisites

- Exposure to HTML, CSS, JS
- Good knowledge of .Net, C# 3.0/4.0
- Exposure to EF would be an advantage
- Knowledge of Design Principles and Patterns would be an advantage
- Desire to learn

MVC – Target Audience

- .Net Web Developers
- Designers
- Architects

What is MVC?

- Model – View – Controller
- Old pattern invented in the 70's
- Successful in many frameworks
  - Smalltalk, Java, Python, Ruby on Rails
- Gives us a clean interaction model for web based development
- Separation of Concerns

What is ASP.NET MVC?

- A new Web Application Project type
- Simply an option
  - Not a replacement for WebForms
  - Builds on top of ASP.NET
- Open Source Web Application Framework
- Implements Model-View-Controller Pattern
- A powerful, patterns-based way to build Scalable and Secure dynamic websites.
- Enables a clean separation of concerns
- Supports TDD-friendly development

ASP.NET Webforms Pros

- Robust foundation
- Can scale well
- Drag and Drop for speed
- Easy to get started
- Very large user base

ASP.NET Webforms Cons

- Viewstate makes pages heavy
- Very complex page life cycle
- Business logic tends to end up in codebehind
- No Applications structure guidance
- View and Controller are mushed together

Why ASP.NET MVC?

- Easier to test without IIS
- Page life cycle is greatly simplified
- Builds on top of ASP.NET
  - Caching
  - Authentication
  - Master Pages
- Viewstate is gone
- Cleaner urls by default
- It helps to reduce the complexity by dividing an application into
  - Model-View-Controller
- This separation (loose coupling) helps in
  - Isolation of components while development
- ASP.Net MVC Web sites are
  - Good in performance and also easy to maintain.

What MVC is not ?

- **Not** the new Web Forms 5.0
- **Not** replacing Web Forms, but Adds to it
- It can **not** use Web Controls
- **Not** a whole new engine but sits on ASP.NET engine

What MVC is?

- Maintain **Clean Separation** of Concerns
- **Extensible** and **Pluggable**
- Enable **clean URLs and HTML**
- Great **integration within ASP.NET**
- **Tooling Support**

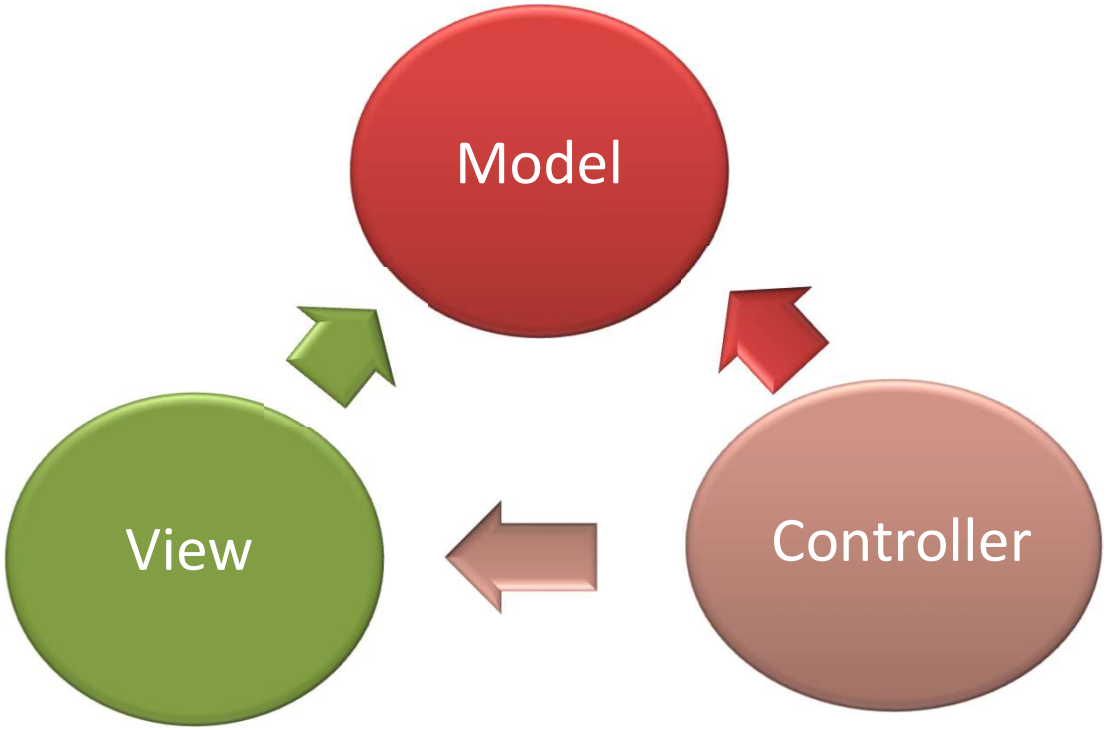
MVC v/s WebForms

- Both MVC and WebForms are built on
  - ASP.Net Framework
- MVC is full of scripting technologies
  - Requires good knowledge on scripting languages
- In MVC there are no server controls support
  - Its fully depend on HTML controls
- Web Forms Web app development is very easy compared to MVC apps
- Web Forms doesn't require a knowledge of HTML and JS
- Web Forms is very rich with server controls

To MVC or not To MVC, That's the <?/>

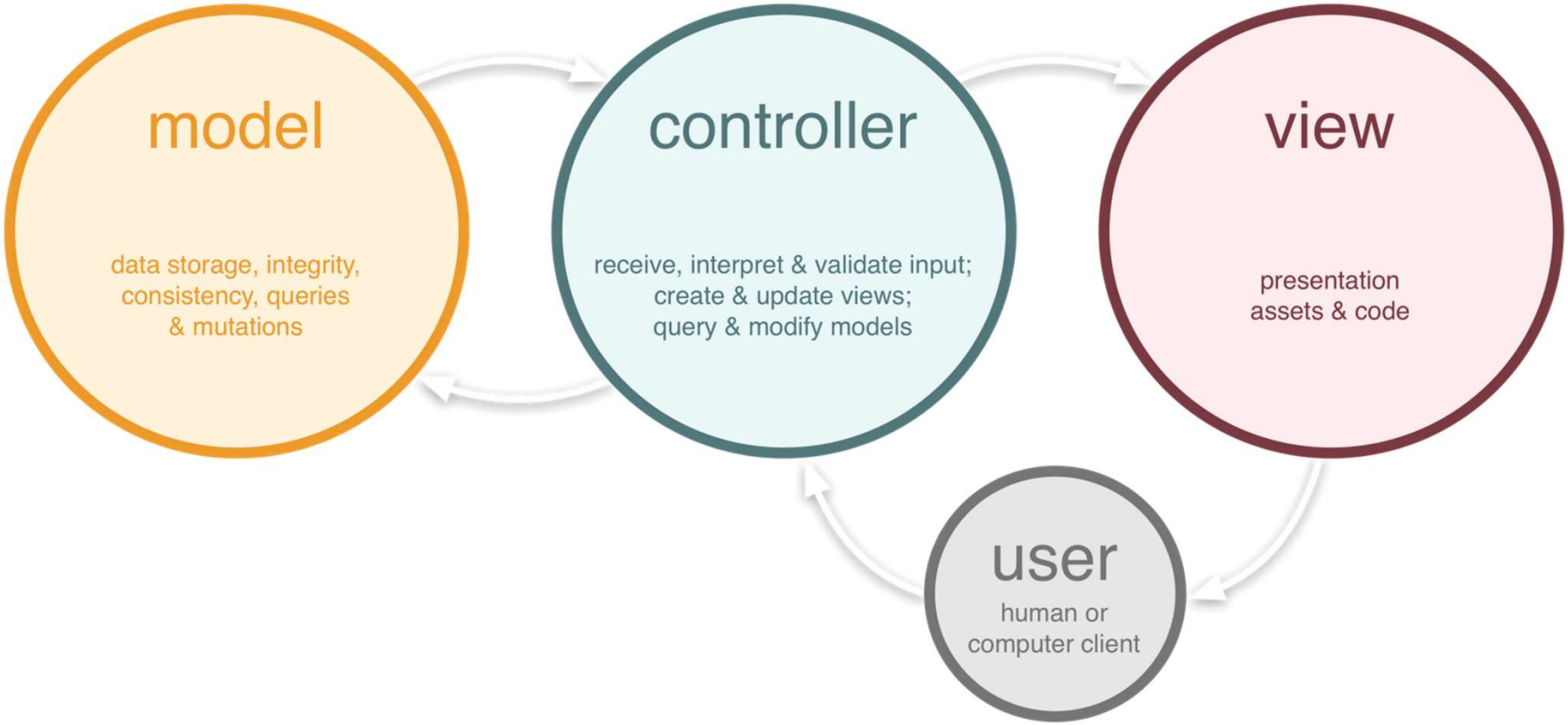
MVC	Web Forms
You want full control over markup	You like programming against the reusable control abstraction that encapsulate UI and logic
You want a framework that *enforces* separation of concerns	You like using the WYSWIG designer and would rather avoid angle brackets
TDD/Unit Testing is a priority for you	You like keeping logic on the server rather than hand writing JavaScript
You like writing Javascript	Unit testing with the MVP pattern is sufficient for your needs

MVC Architecture



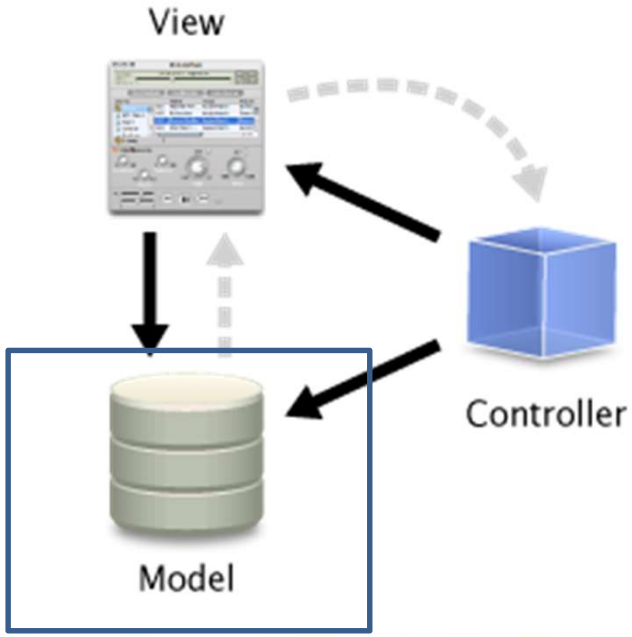
- The Model-View-Controller (MVC) architectural pattern separates an application into three main components: the model, the view, and the controller.

Interaction of MVC Approach



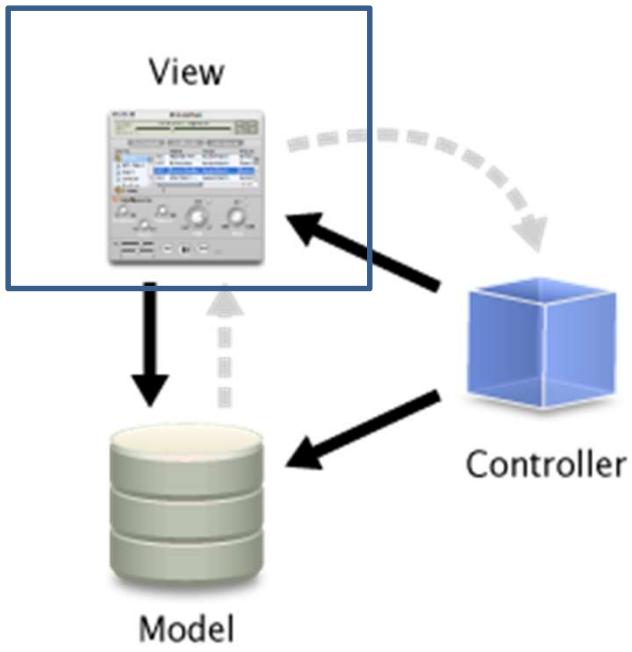
Model

- Should be “fat”
- Has business/domain logic
- ASP.NET MVC isn’t prescriptive here
  - LINQ to SQL
  - nHiberate
  - Entity Framework
  - Custom/POCO object



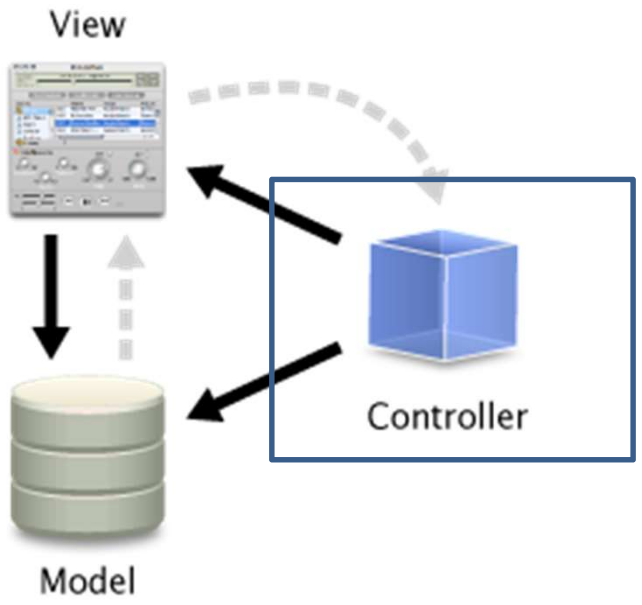
View

- Should be relatively “dumb”
- No business logic
- Only Display logic / Transformation
- JavaScript is valid for client side - jQuery

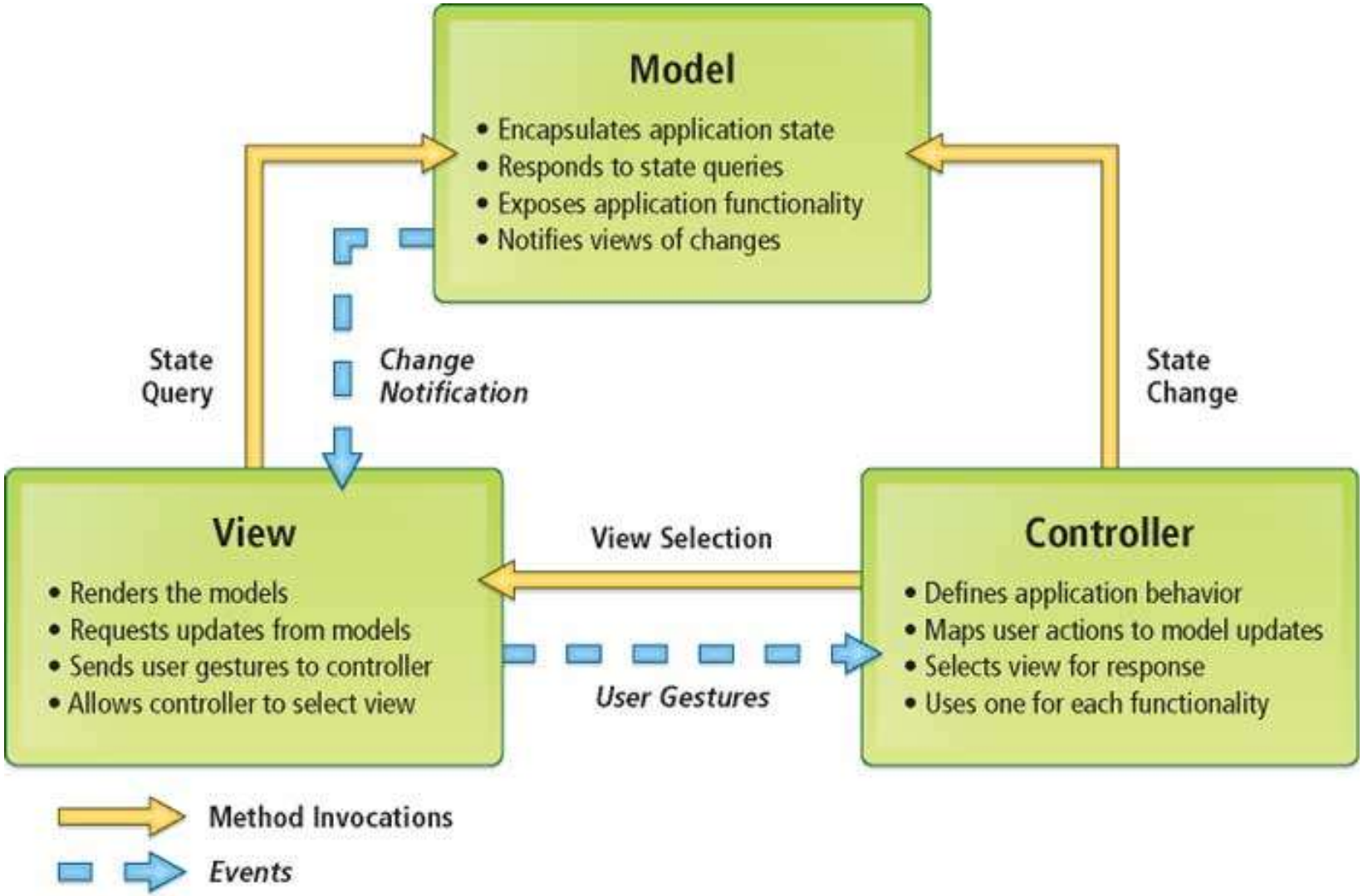


Controller

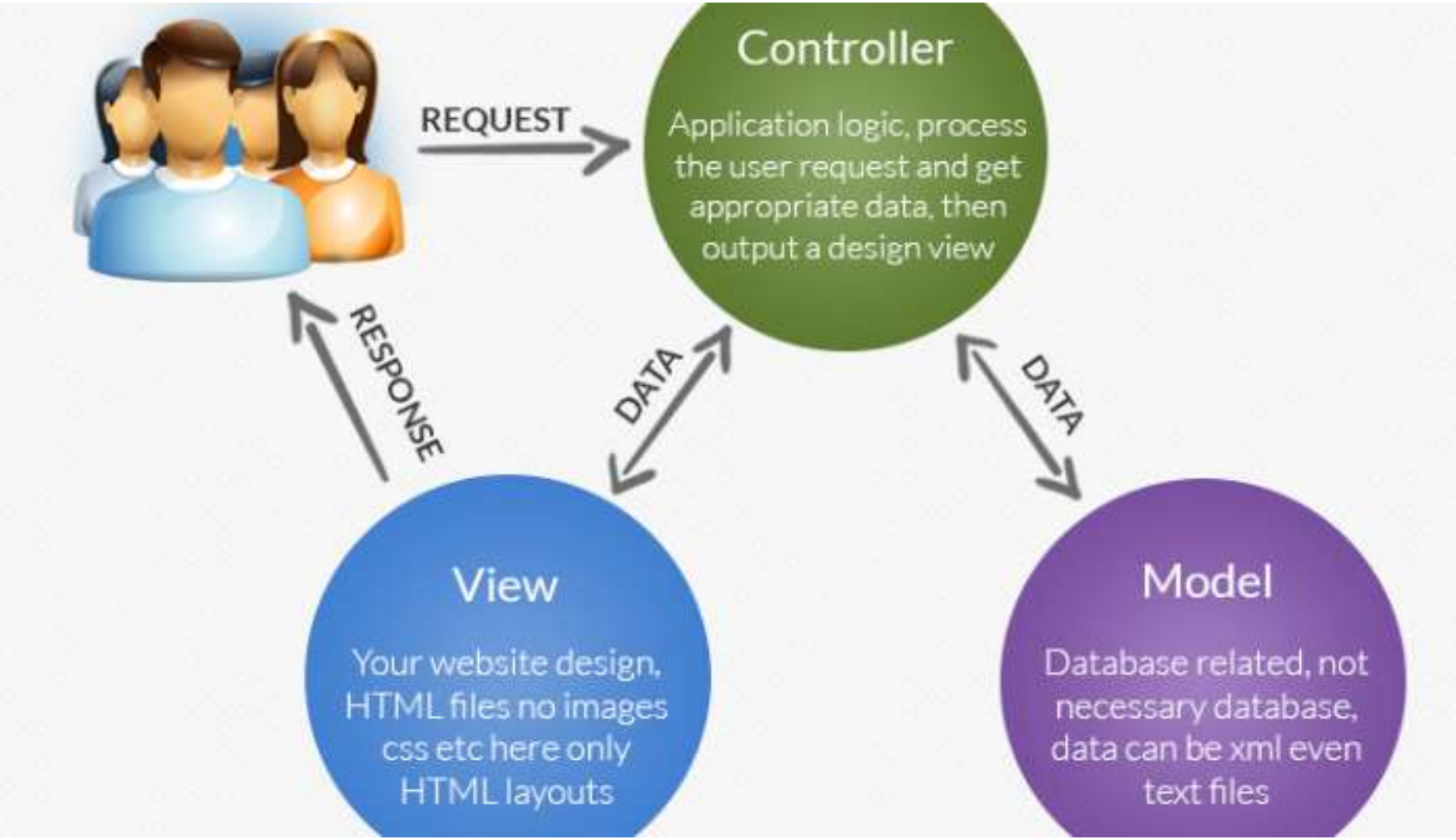
- Should be “skinny”
- Controller has ‘Actions’
- Requests always come through the controller
- Decides what data is needed
- Tells which View to render
- Tells the View what to render



MVC – Separation of Concerns



MVC Request Processing



MVC Flow

Step 1

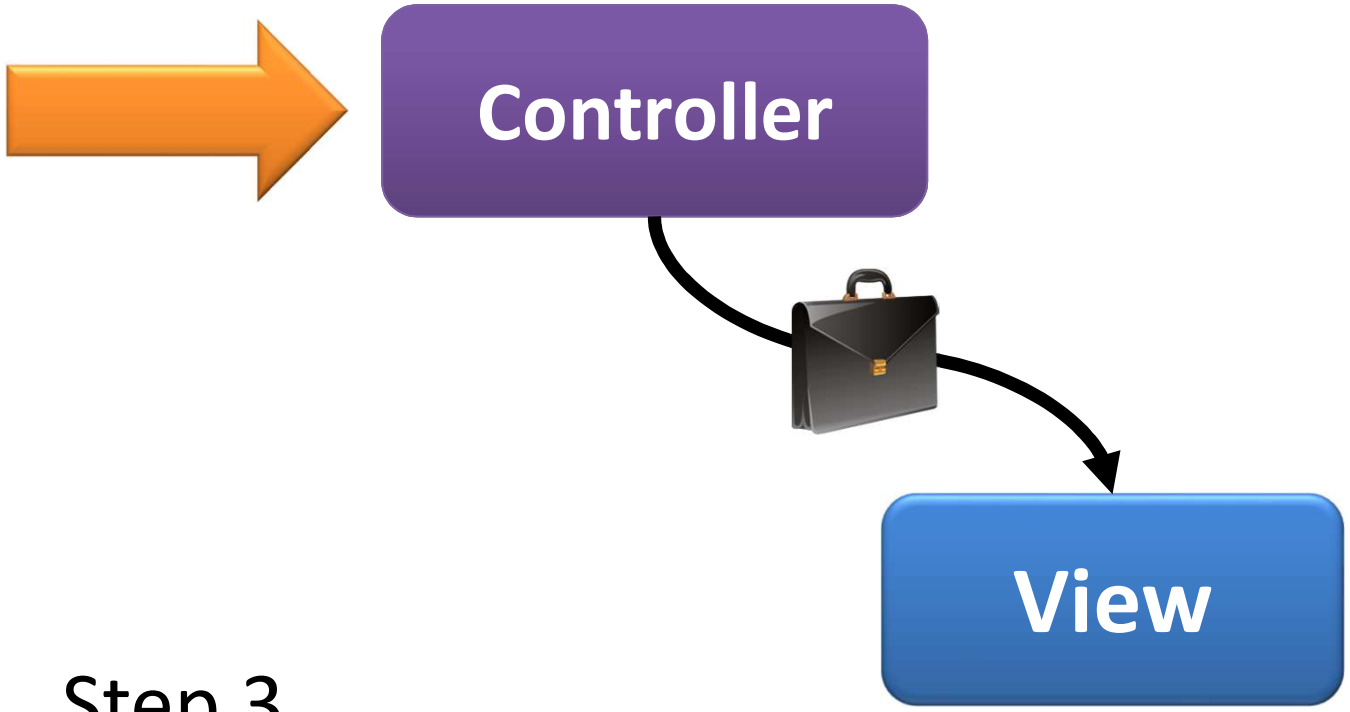
Incoming request directed to **Controller**

MVC Flow

Step 2

Controller processes request and forms a data **Model**

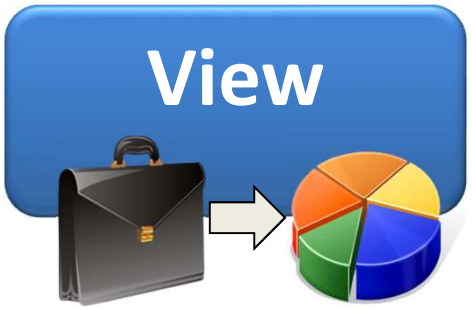
MVC Flow



Step 3

Model is passed to **View**

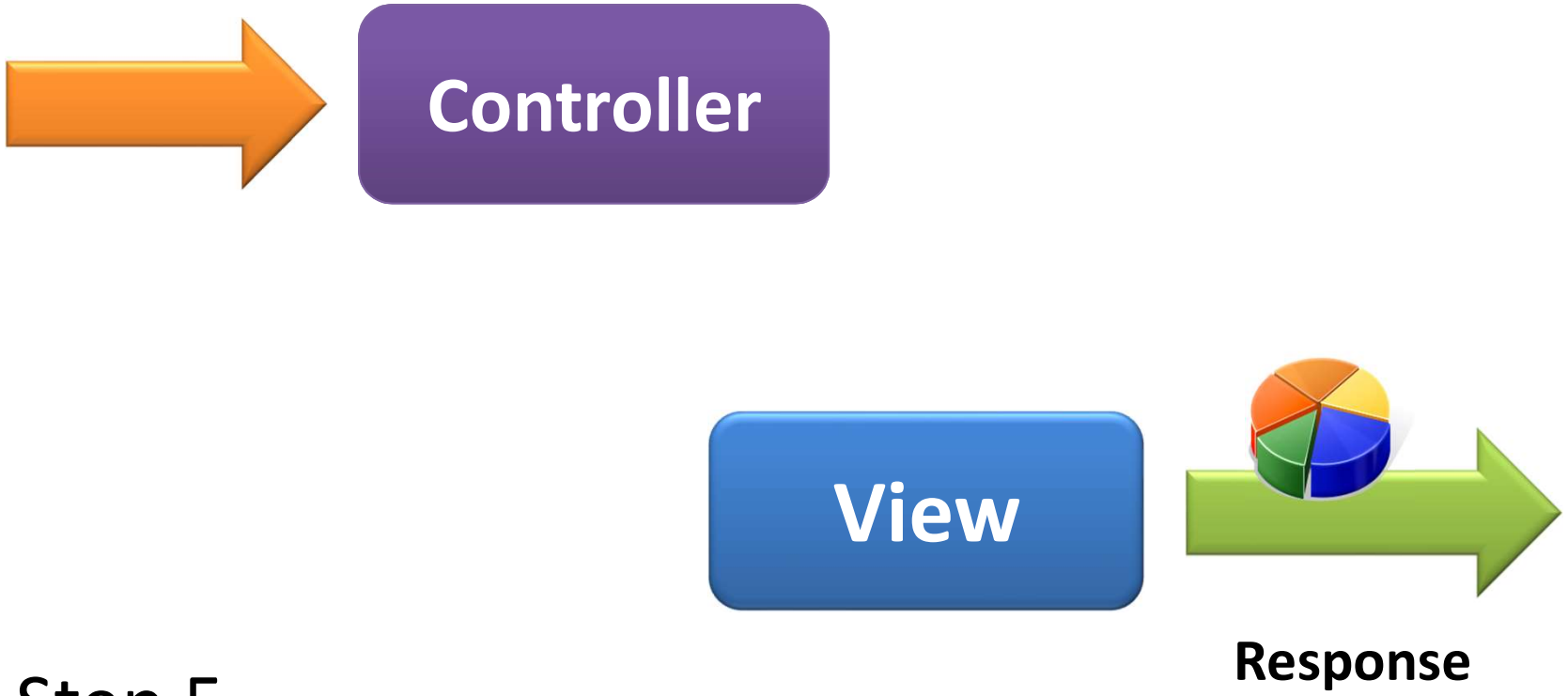
MVC Flow



Step 4

View transforms **Model** into appropriate output format

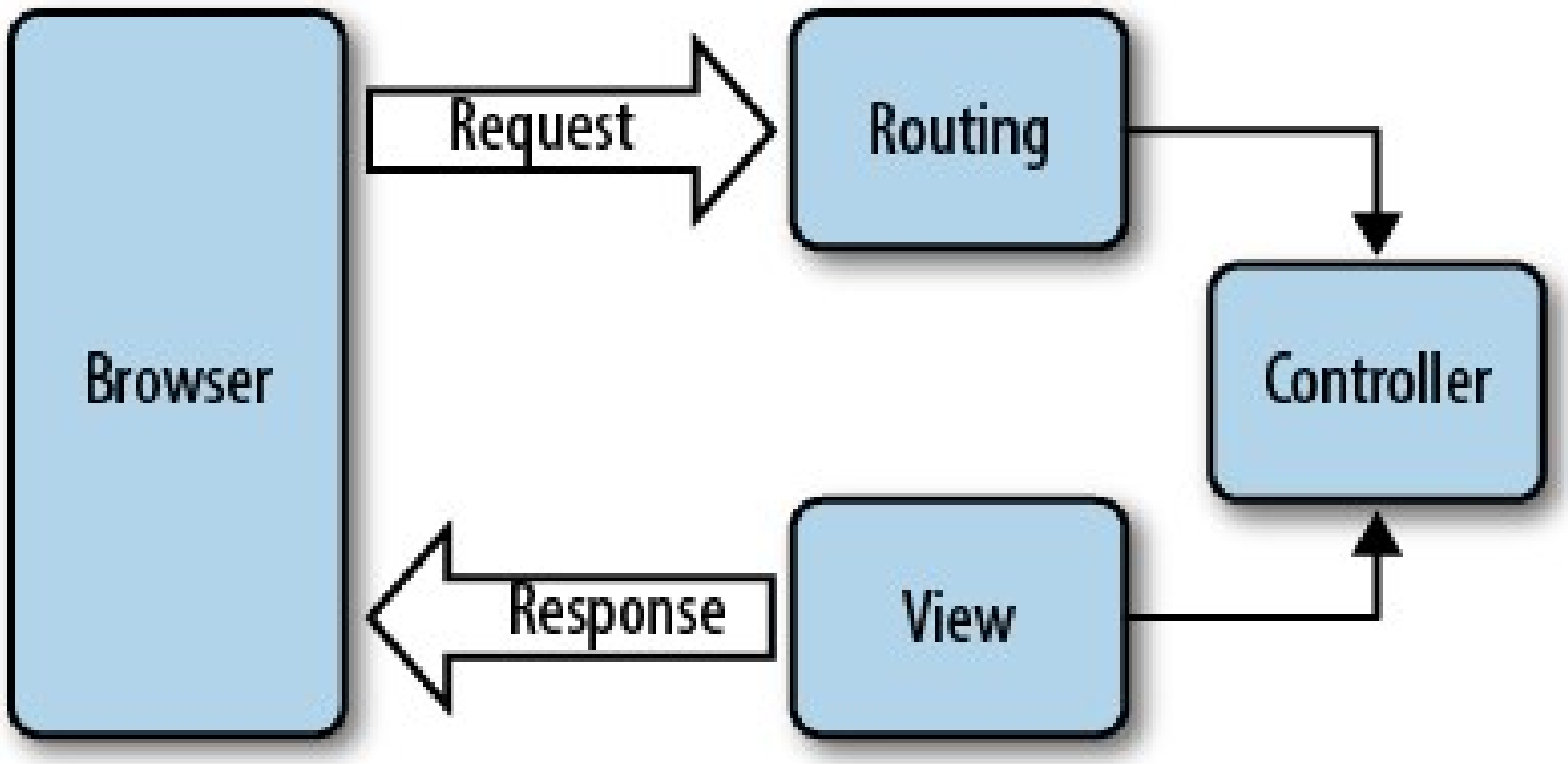
MVC Flow



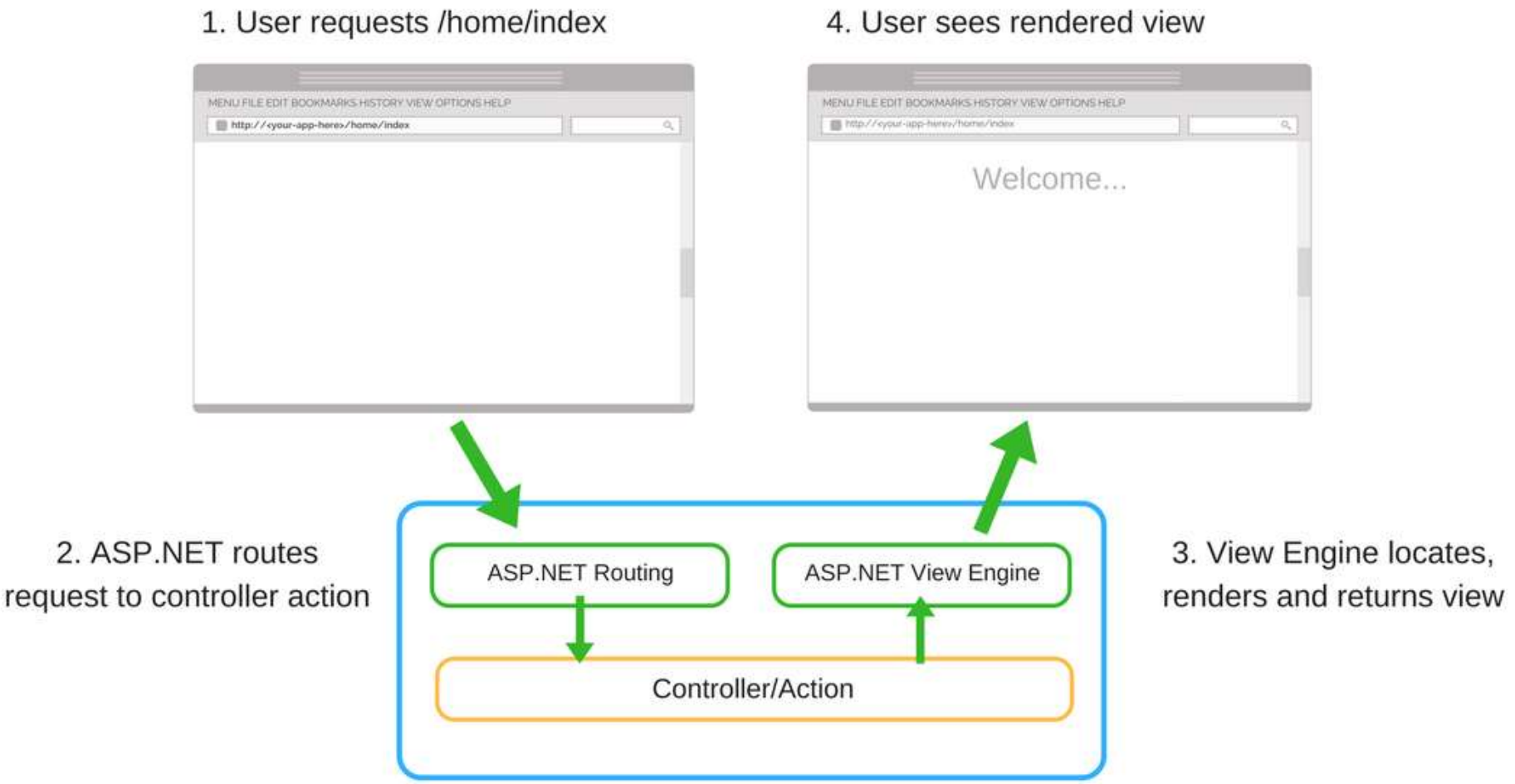
Step 5

Response is rendered

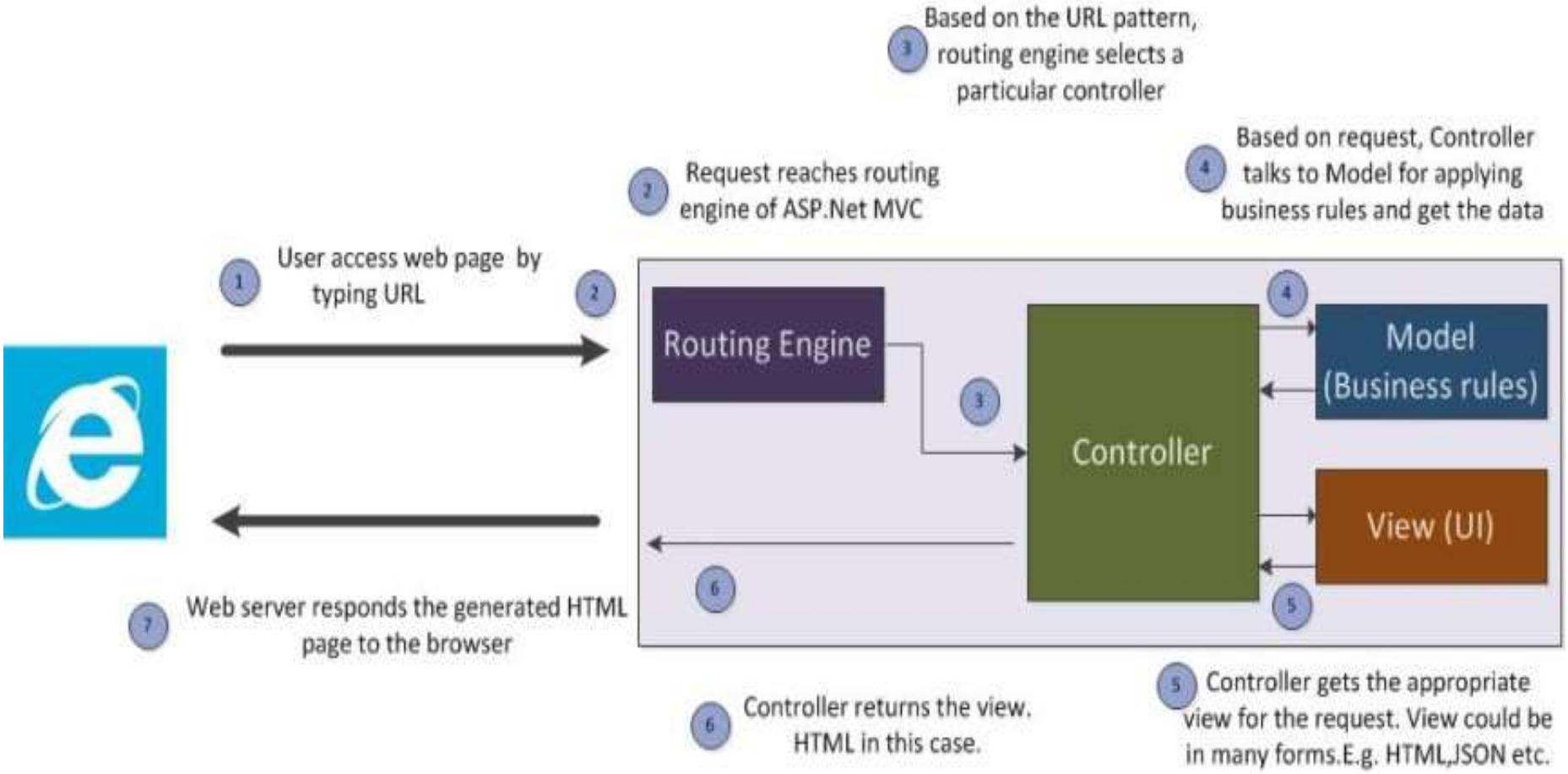
MVC Request Life Cycle



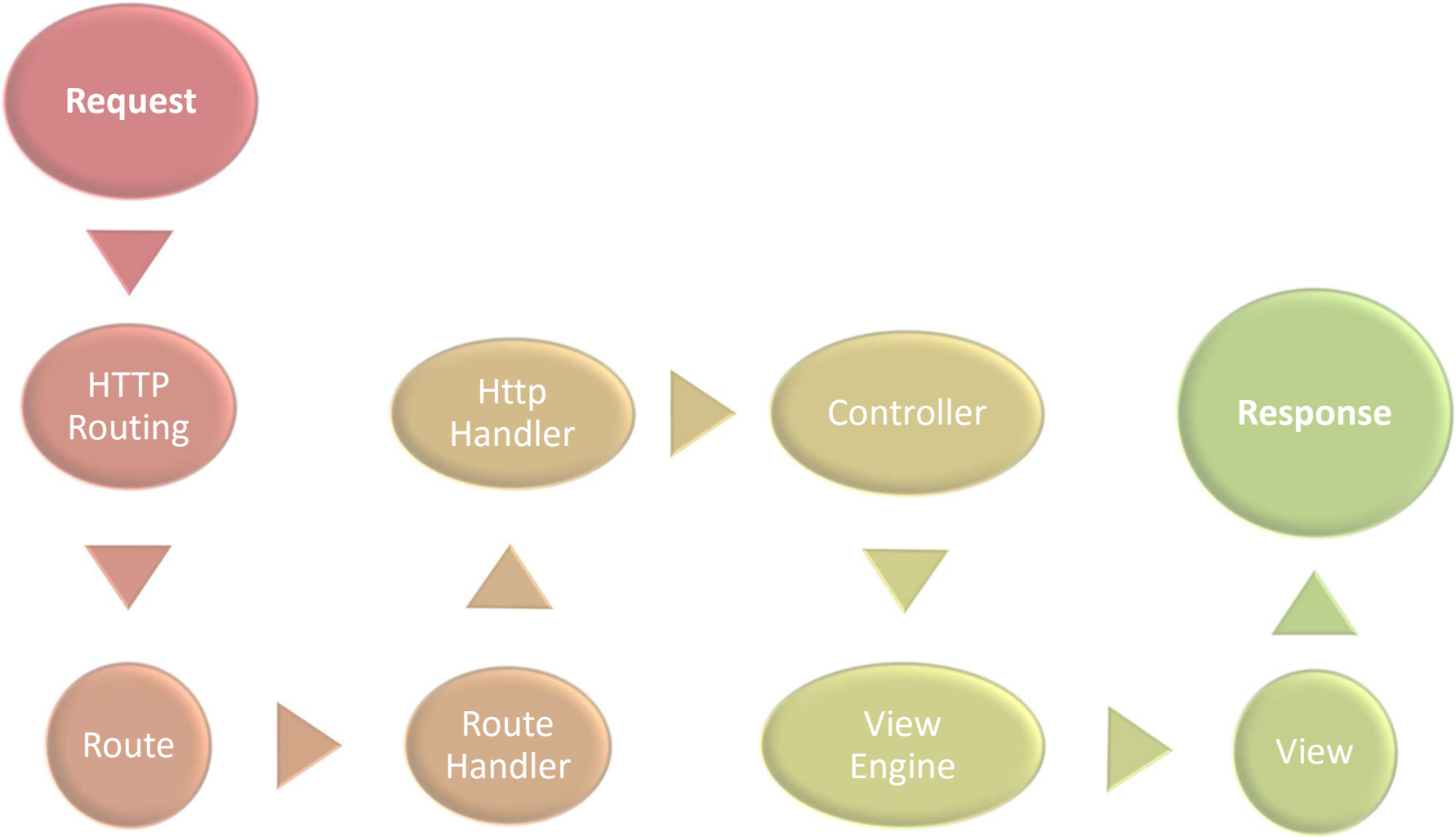
MVC Request Life Cycle



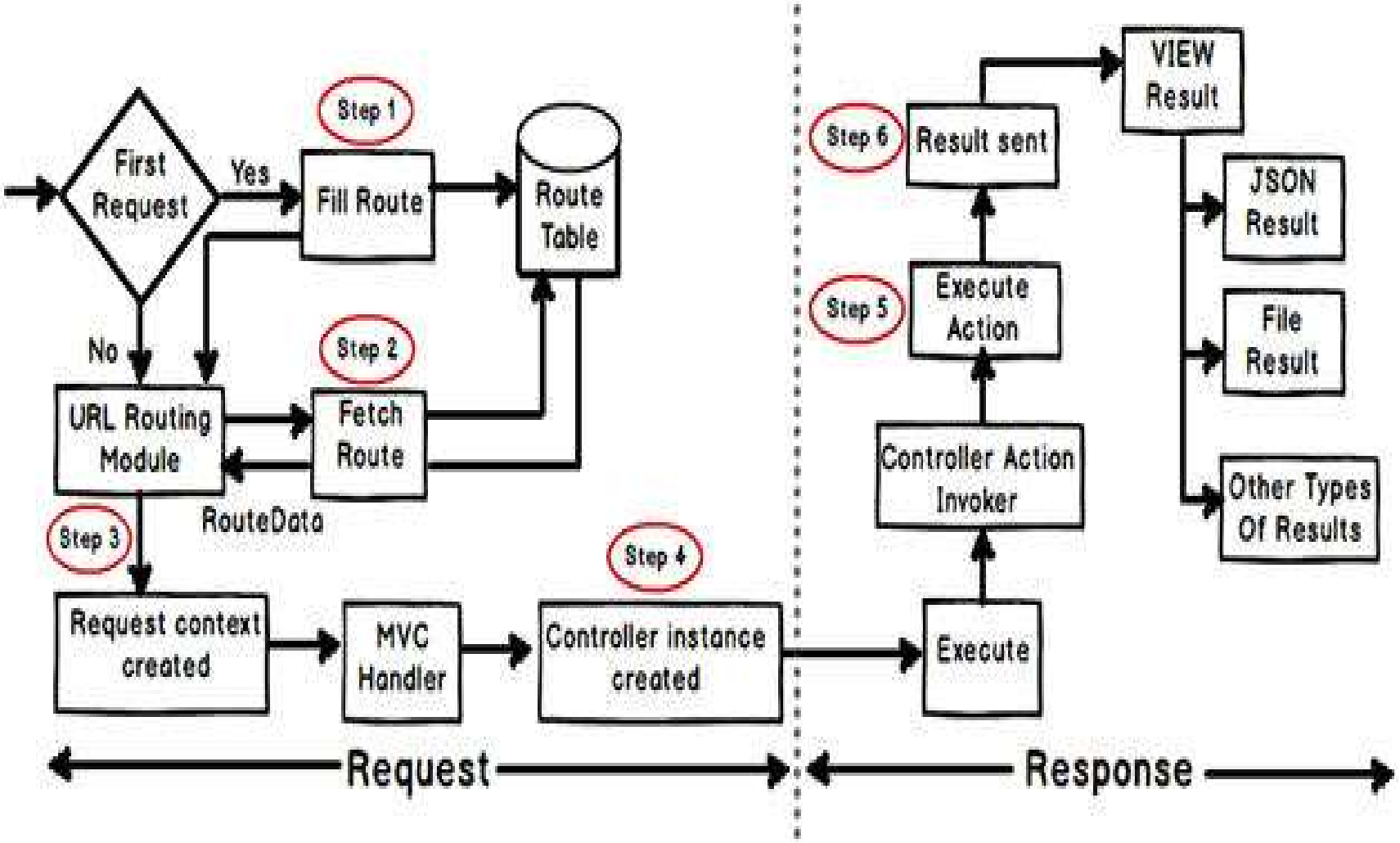
MVC Request Life Cycle



Request Flow – in more detail



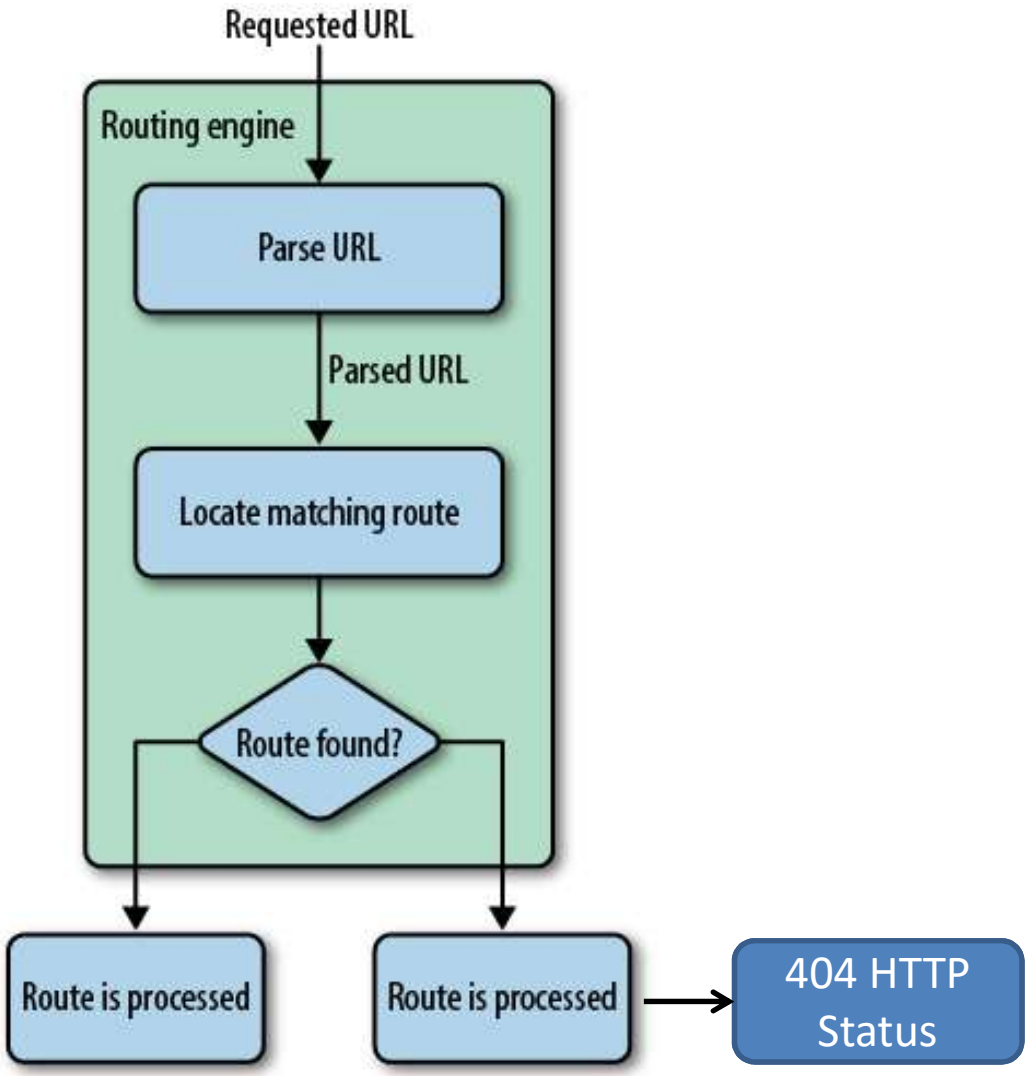
MVC Application Life Cycle



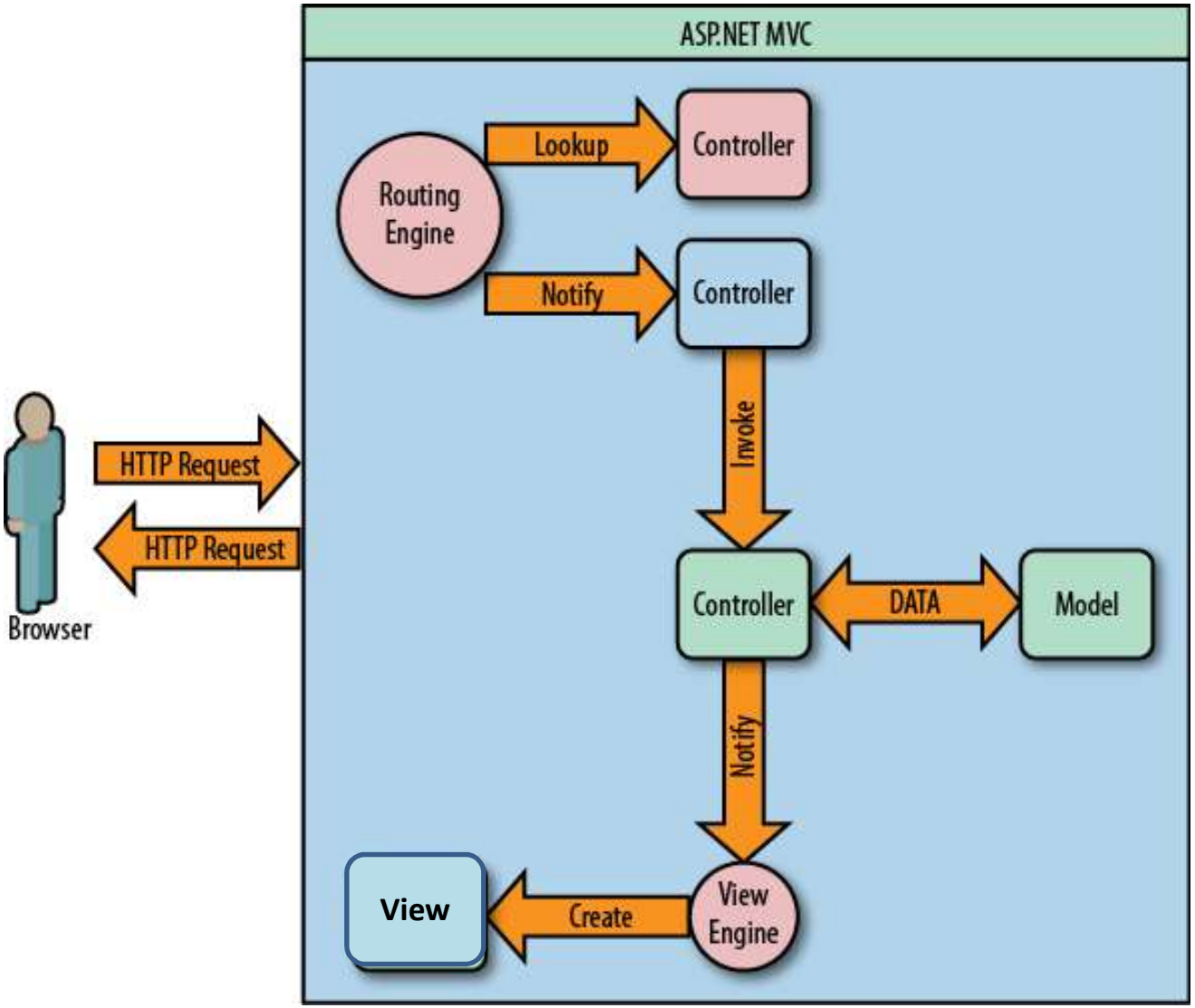
Routing Engine

- Pattern Matching System
- Parses Incoming Request
  - URLs -> application -> Controller -> Action
- Construct outgoing URLs
  - Constructed URLs can be used to call back to Controllers/Actions

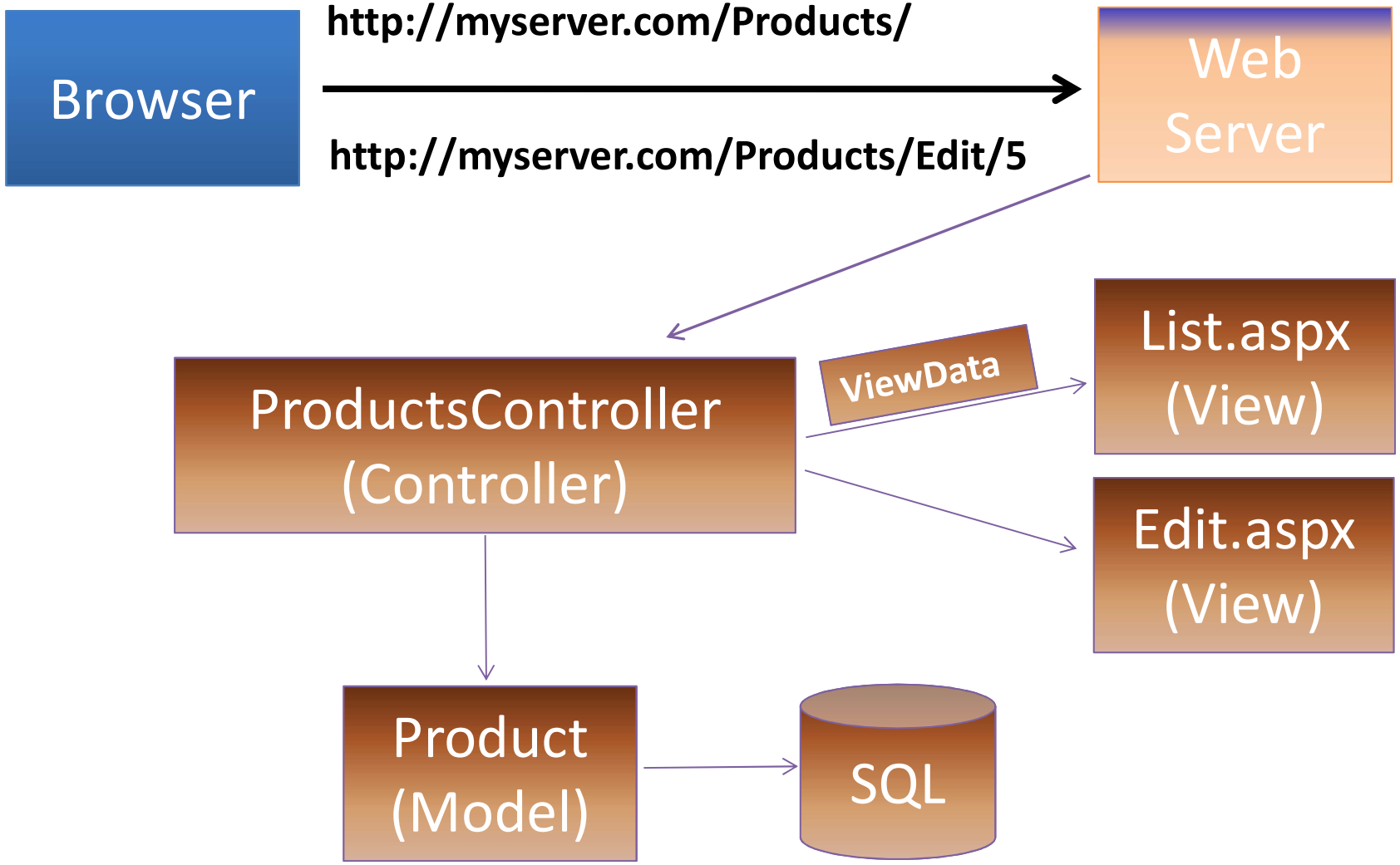
ASP.Net Routing



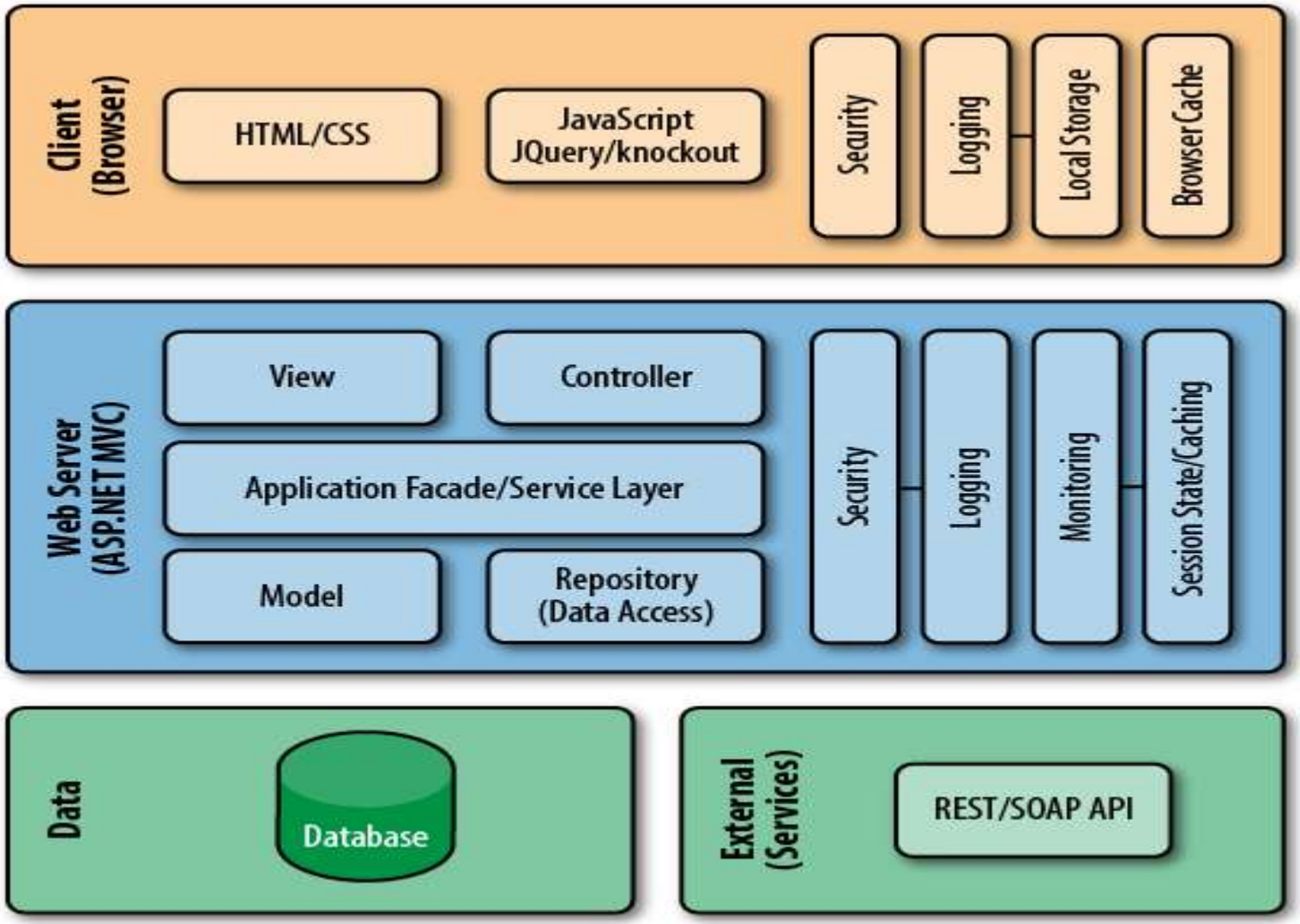
MVC Process Life Cycle



ASP.NET MVC Routing Model



MVC Web Application Logical Architecture

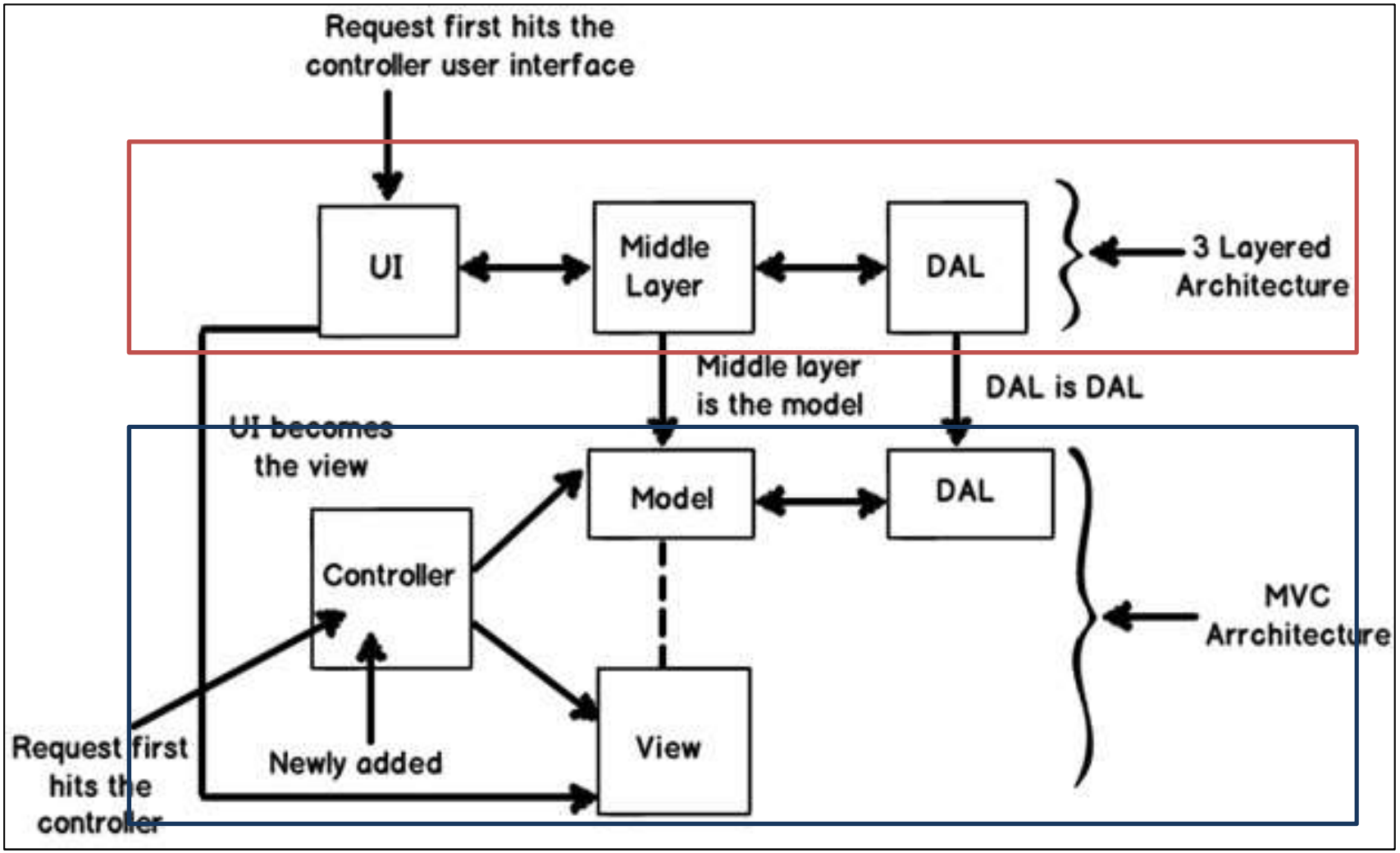


MVC and Three Layered Architecture

MVC is an evolution of a three layered traditional architecture. Many components of the three layered architecture are part of MVC. So below is how the mapping goes:

Functionality	**Three layered / tiered architecture**	**MVC Architecture**
Look and Feel	User interface	View
UI logic	User interface	Controller
Business logic /validations	Middle layer	Model
Request is first sent to	User interface	Controller
Accessing data	Data access layer	Data Access Layer

MVC and Three Layered Architecture



MVC version Comparison

MVC 1/2	MVC 3	MVC 4/5
Client-side validation	**Razor**	**ASP.NET Web API**
Templated Helpers Areas	Readymade project templates	Refreshed and modernized default project templates. New mobile project template.
Asynchronous Controllers	**HTML 5 enabled templates**	**Many new features to support mobile apps**
Html.ValidationSummary Helper Method	**Support for Multiple View Engines, JavaScript, and AJAX**	Bundling and Minification
DefaultValueAttribute in Action-Method	Model Validation Improvements	Enhanced support for asynchronous methods
Parameters binding		
Binary data with Model Binders		
DataAnnotations Attributes		
Model-Validator Providers		
New RequireHttpsAttribute Action Filter		
Templated Helpers		
Display Model-Level Errors		

Extensible

- Replace Any Part with one of your own
- As simple or complex as it needs to be to suit your tasks
- Plays well with others
  - Want to use NHibernate for models? OK!
  - Want to use Brail for views? OK!
  - Want to use VB for controllers? OK!

Who will benefit?

- Beginners – pros and cons
  - Less drag and drop
  - Fits the nature of the web better – no state, etc
- Existing developers
  - Need to unlearn some design habits
  - More manageable solutions

Webforms vs MVC

- ASP.Net Webforms
  - faster starting point
  - drag and drop
- ASP.NET MVC
  - Unnecessary abstractions are gone
  - Easier to Unit Test components
- Can write bad or good code in both
- Either can be complex or simple

Q & A

DEMO

MVC First Steps

Summary

- Not a replacement for WebForms
  - All about alternatives
- Fundamental
  - Part of the System.Web namespace
  - Same team that builds WebForms

Additional Resources

- Official sites
  - Central landing site: http://asp.net/mvc
  - Forums: http://forums.asp.net/1146.aspx
- Source available
  - Source drop: http://codeplex.com/aspnet
- Blogs
  - http://blogs.microsoft.co.il/blogs/noam
  - http://weblogs.asp.net/scottgu
  - http://hanselman.com/
  - http://haacked.com/