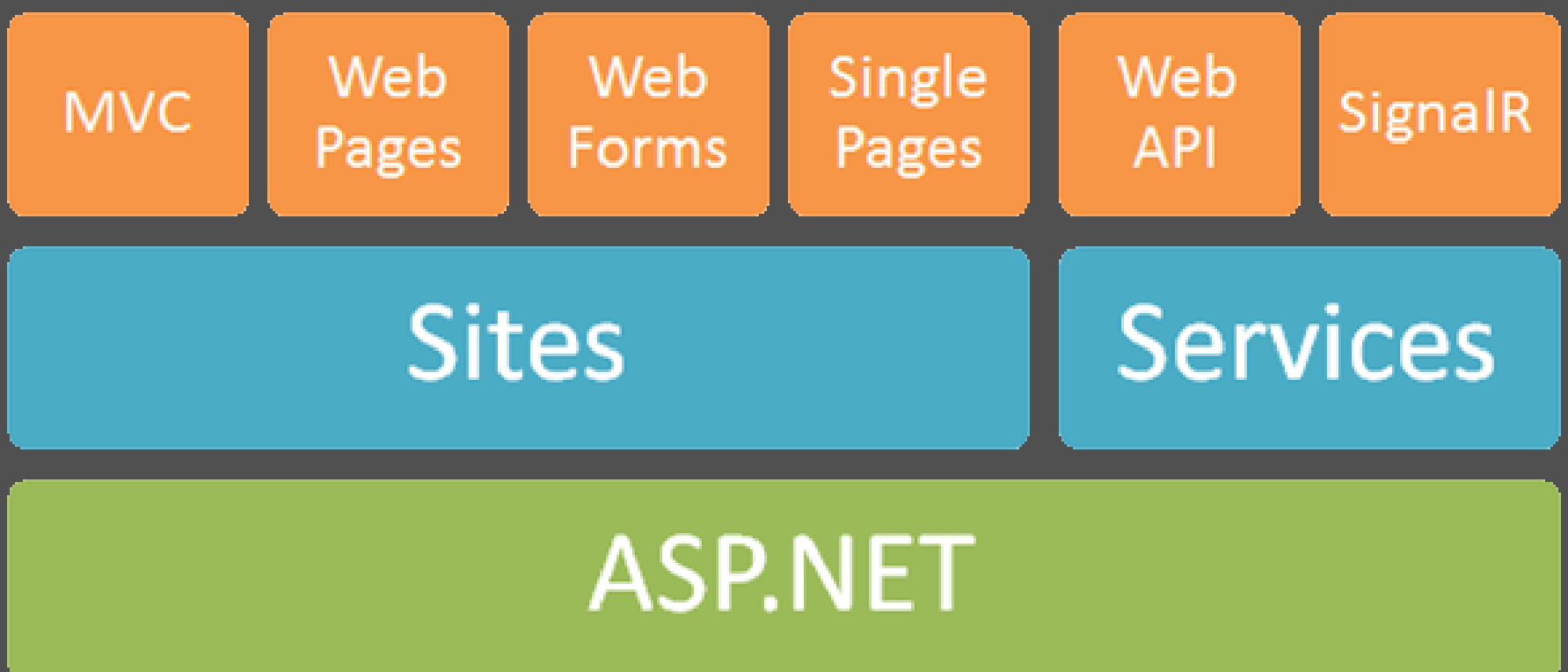
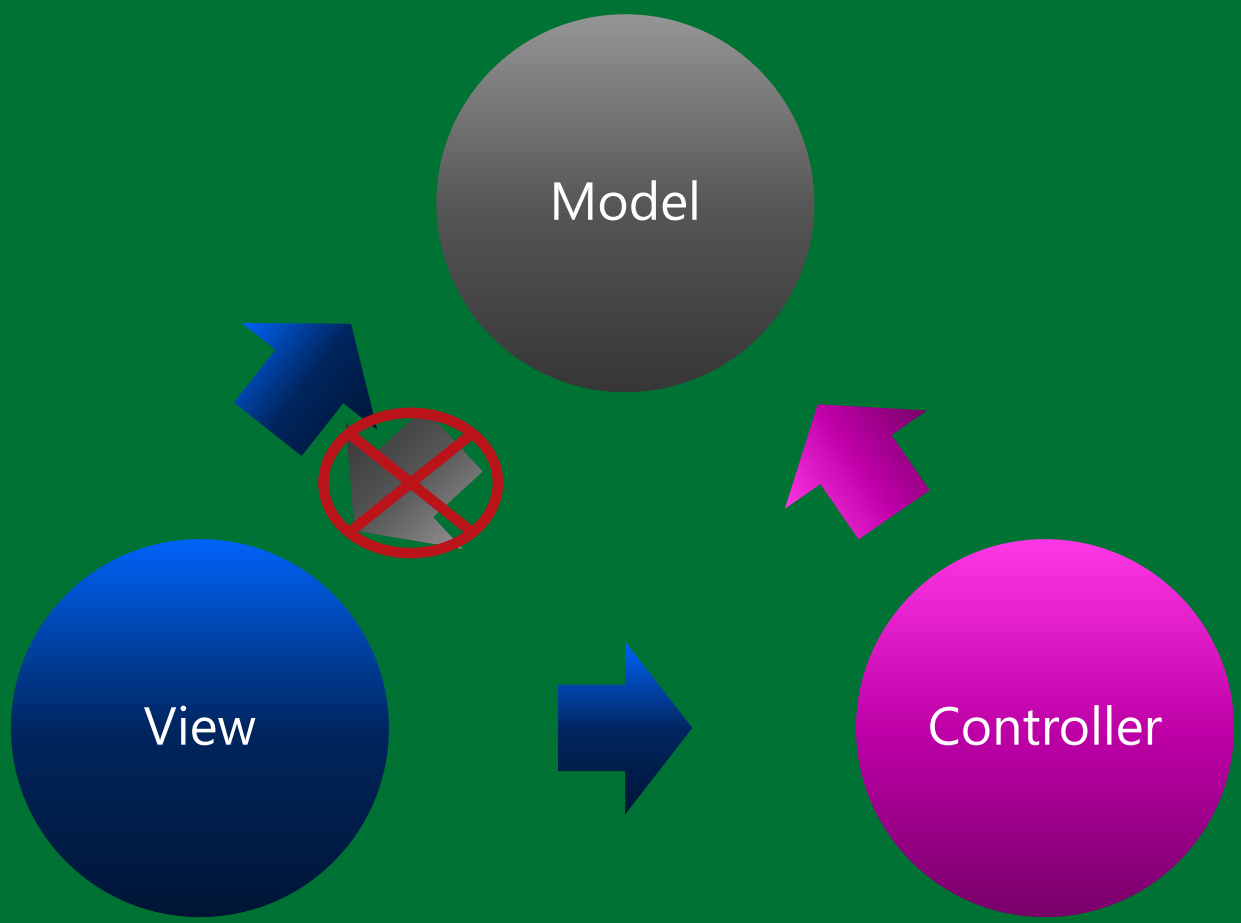

ASP.NET
Rasmus Lystrøm
External Associate Professor
ITU

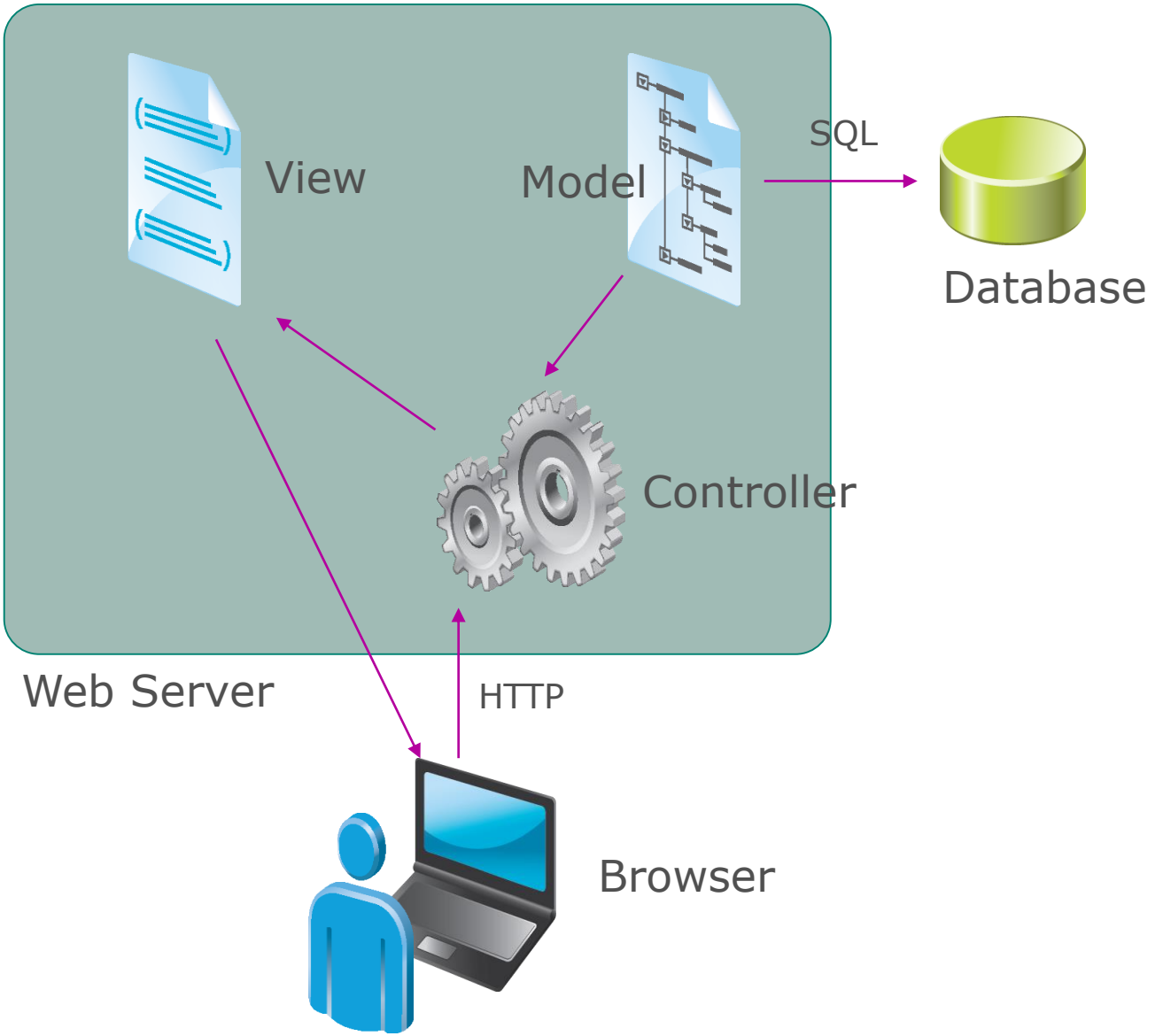
http://www.asp.net


MVC
Web Pages
Web Forms
Single Pages
Web API
SignalR
Sites
Services
ASP.NET

ASP.NET MVC Architecture



Models, Views, and Controllers



MVC Architecture

The Model is the part of the application that handles the logic for the application data. Often model objects retrieve data (and store data) from a database

The View is the parts of the application that handles the display of the data. Most often the views are created from the model data

The Controller is the part of the application that handles user interaction

Model

Domain-specific representation of data

Business logic

Storage layer is an implementation detail

View

Presents data to the user

Read-only views as well as forms

Minimal display-only logic

Controller

Responds to requests

Connects models to view

Invokes model code as appropriate

What is Razor?

Razor is a markup syntax that lets you embed server-based code into web pages

Server-based code can create dynamic web content on the fly, while a web page is written to the browser

Razor is based on ASP.NET, and designed for creating web applications

Razor Syntax

Razor code blocks are enclosed in @{ ... }

Inline expressions (variables and functions) start with @

Code statements end with semicolon

Variables are declared with the var keyword

Strings are enclosed with quotation marks

C# code is case sensitive

C# files have the extension .cshtml

Differentiating Server Side Code from HTML with @

Use **@** to identify server-side C# code

Use **@@** to render an @ symbol in an HTML page.

Use **@:** to explicitly declare a line of text as content and not code.

Use **<text>**to explicitly declare several lines of text as content and not code.

To render text without HTML encoding, you can use the **Html.Raw()** helper.

Features of Razor Syntax

A sample code block displaying the features of Razor.

```
@* Some more Razor examples *@

<span>
    Price including Sale Tax: @Model.Price * 1.2
</span>
<span>
    Price including Sale Tax: @(Model.Price * 1.2)
</span>

@if (Model.Count > 5)
{
 <ol>
   @foreach(var item in Model)
   {
      <li>@item.Name</li>
   }
 </ol>
}
```

Binding Views to Model Classes and Displaying Properties

You can use strongly-typed views and include a declaration of the model class. Visual Studio helps you with additional IntelliSense feedback and error-checking as you write the code.

Binding to Enumerable Lists:

```
@model IEnumerable<MyWebSite.Models.Product>

<h1>Product Catalog</h1>
@foreach (var product in Model)
{
    <div>Name: @product.Name</div>
}
```

You can use dynamic views to create a view that can display more than one model class.

HTML Helpers

Action Helpers

- Html.ActionLink
- Url.Action

Editor Helpers

- Html.EditorFor

Validation Helpers

- Html.ValidationMessageFor
- Html.ValidationSummary

Display Helpers

- Html.DisplayFor
- Html.DisplayNameFor
- Html.LabelFor

Form

- Html.BeginForm

Display and Edit Data Annotations

```
public class Photo
{
    [DisplayName("Picture")]
    public byte[] File { get; set; }

    [DataType(DataType.MultilineText)]
    public string Description { get; set; }

    [DataType(DataType.DateTime)]
    [DisplayName("Created Date")]
    [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}")]
    public DateTime CreatedDate { get; set; }
}
```

Validating User Input with Data Annotations

```
public class Person
{
    public int Id { get; set; }

    [Required(ErrorMessage="Please enter a name.")]
    public string Name { get; set; }

    [Range(0, 250)]
    public int Height { get; set; }

    [Required]
    [DataType(DataType.EmailAddress)]
    public string EmailAddress { get; set; }
}
```