.Net Core Fundamentals

Do Dang Tuan – Rookies Program for MS

Agenda

- Introduction
- .NET Core Overview
- .NET Core Components
- .NET Standard
- ASP.NET Core Fundamentals
- Demo
- Dependency Injection

Problems of .NET

- Windows only
- Closed
- All or nothing monolithic framework
- 15 years old

.NET Core - The Future

- A general purpose development platform
- Cross-platform, supporting Windows, macOS and Linux
- Can be used in device, cloud, and embedded/IoT.

.NET Core Components

- A .NET Runtime – CoreCLR
- A set of Framework Libraries – CoreFX
- .NET Core SDK
- The ‘dotnet’ app host

ASP.NET vs ASP.NET Core – Main Differences

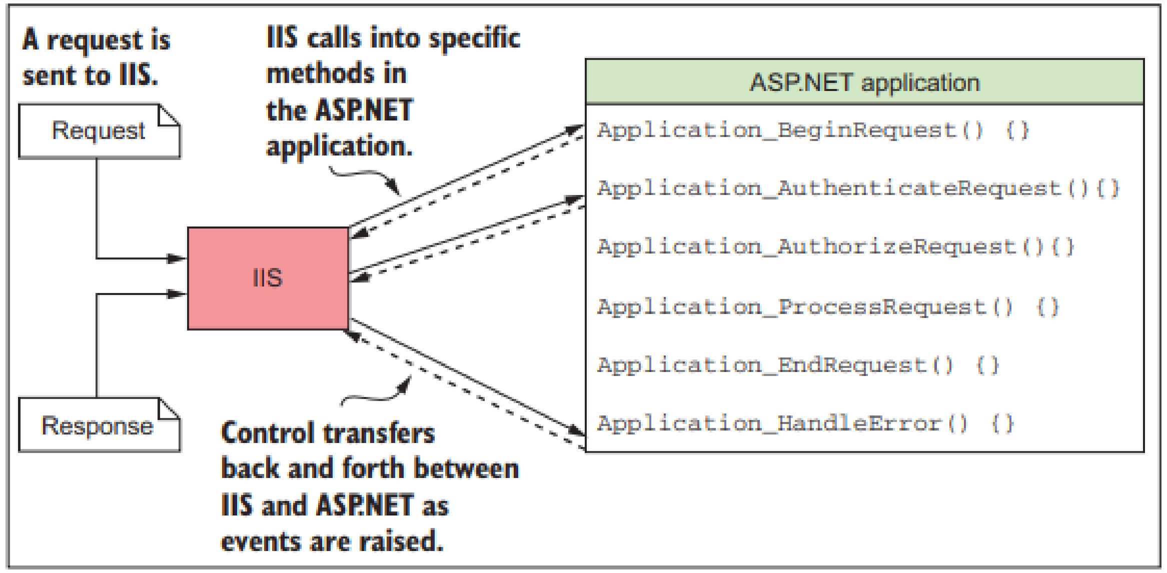
ASP.NET

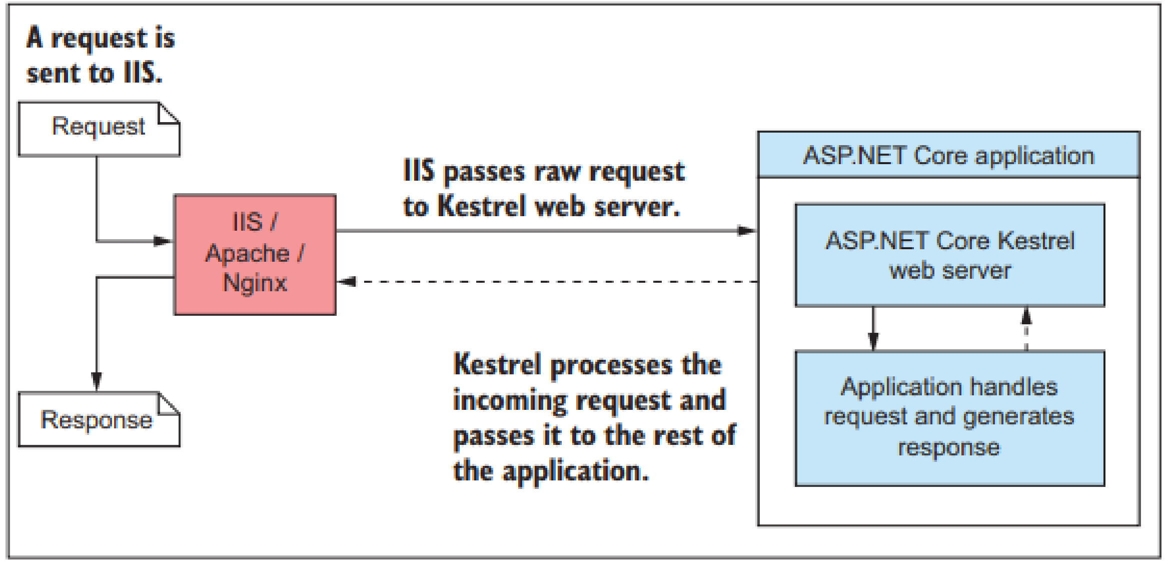
- IIS, Windows only
- System.Web, Included all by default
- HTTP Modules, HTTP Handlers
- MVC + Web API + Web Pages
- Web.config

ASP.NET CORE

- Kestrel, Cross-platform
- No System.Web, Everything is Nuget packages. There is no dll by default
- Middlewares
- ASP.NET Core MVC
- .json, .ini, environment variables, .etc

ASP.NET vs ASP.NET Core


A request is sent to IIS.
Request
IIS calls into specific methods in the ASP.NET application.
IIS
Response
Control transfers back and forth between IIS and ASP.NET as events are raised.
ASP.NET application
Application_BeginRequest() {}
Application_AuthenticateRequest(){}
Application_AuthorizeRequest(){}
Application_ProcessRequest() {}
Application_EndRequest() {}
Application_HandleError() {}


A request is sent to IIS.
Request
IIS / Apache / Nginx
Response
IIS passes raw request to Kestrel web server.
Kestrel processes the incoming request and passes it to the rest of the application.
ASP.NET Core application
ASP.NET Core Kestrel web server
Application handles request and generates response

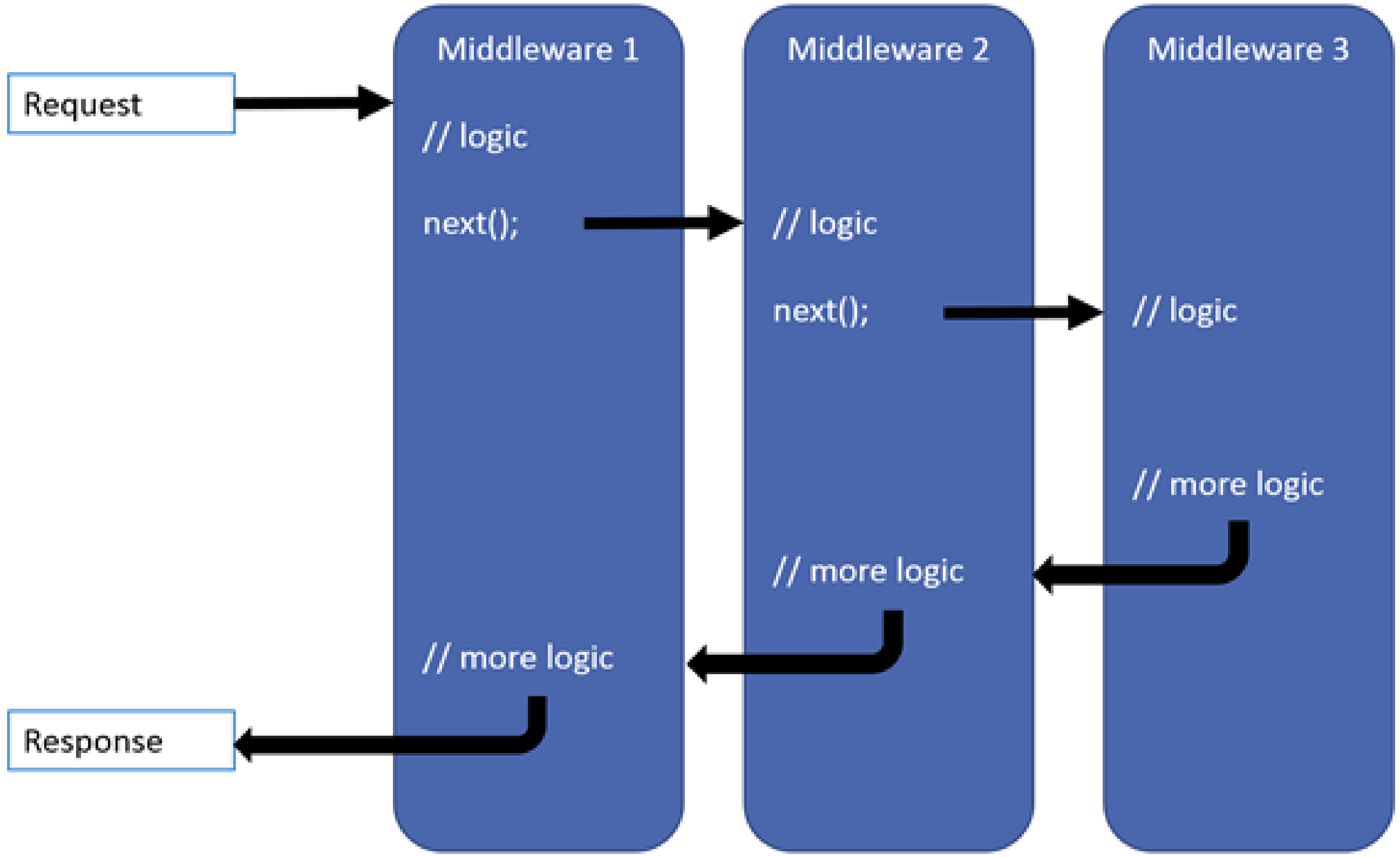
What is Middleware ?

- “Middleware are software components that are assembled into an application pipeline to handle requests and responses”

- Each component in the pipeline is a request delegate

- Each delegate can invoke the next component in the chain, or short-circuit, returning back up the call chain

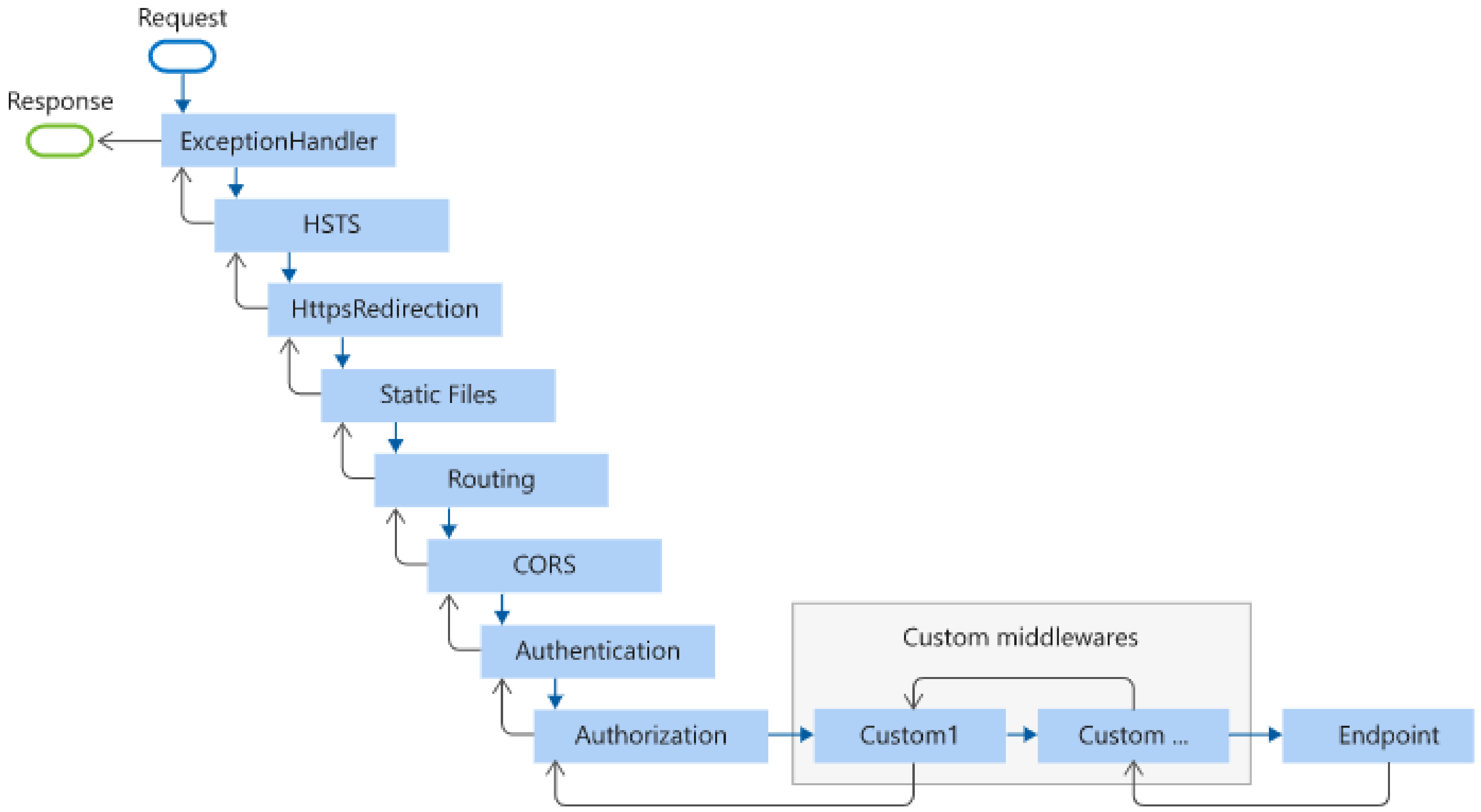
Middleware


Request
Middleware 1
// logic
next();
// more logic
Middleware 2
// logic
next();
// more logic
Middleware 3
// logic
// more logic
Response

Build-in Middlewares

- Routing
- Authentication
- Static files
- Diagnostics
- Error handling
- Session
- CORS
- Localization
- Custom

Middleware Order



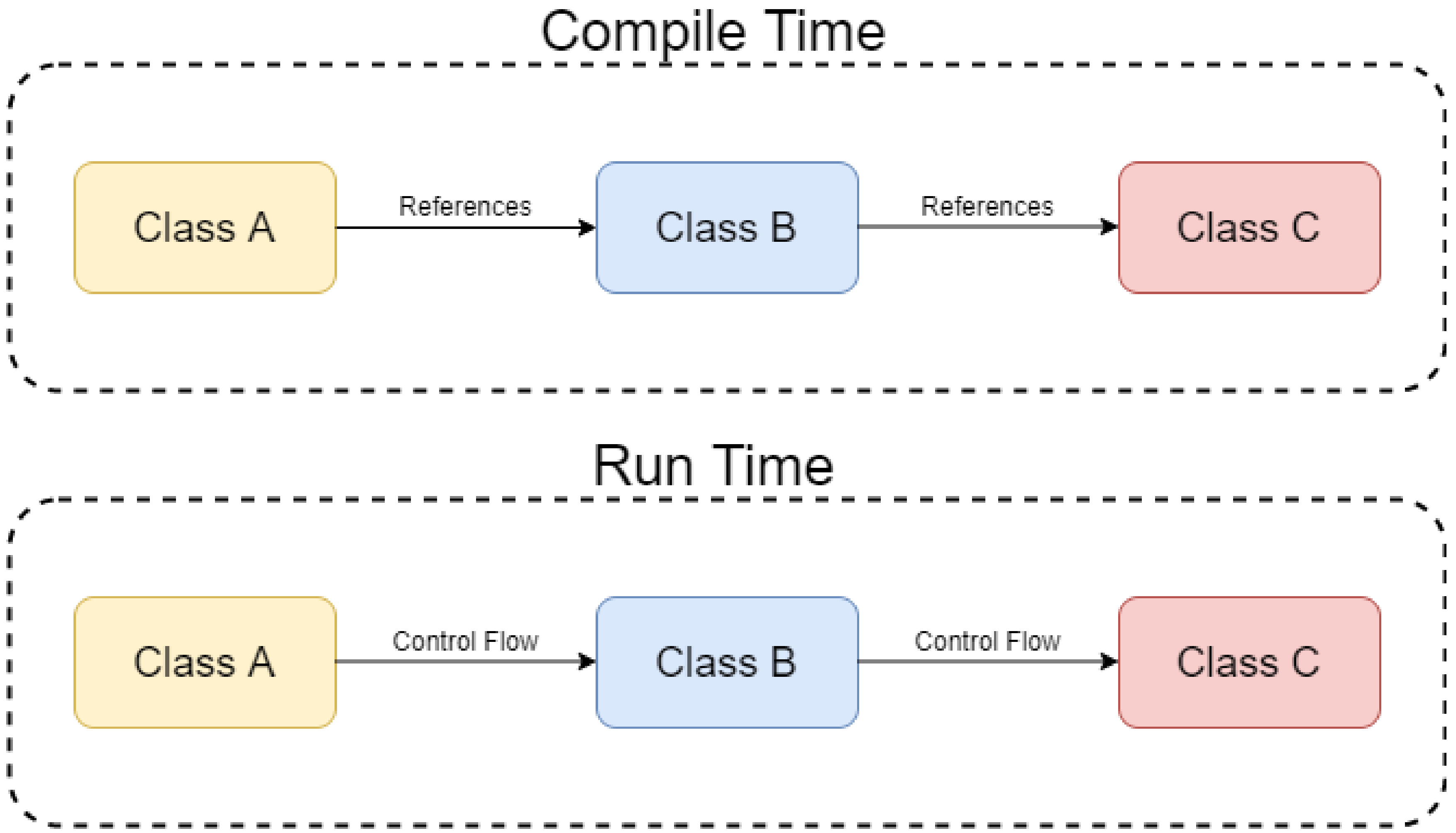
Demo

- Create an ASP.NET Core project
- Project structure
- Write simple middlewares
- Working with CLI

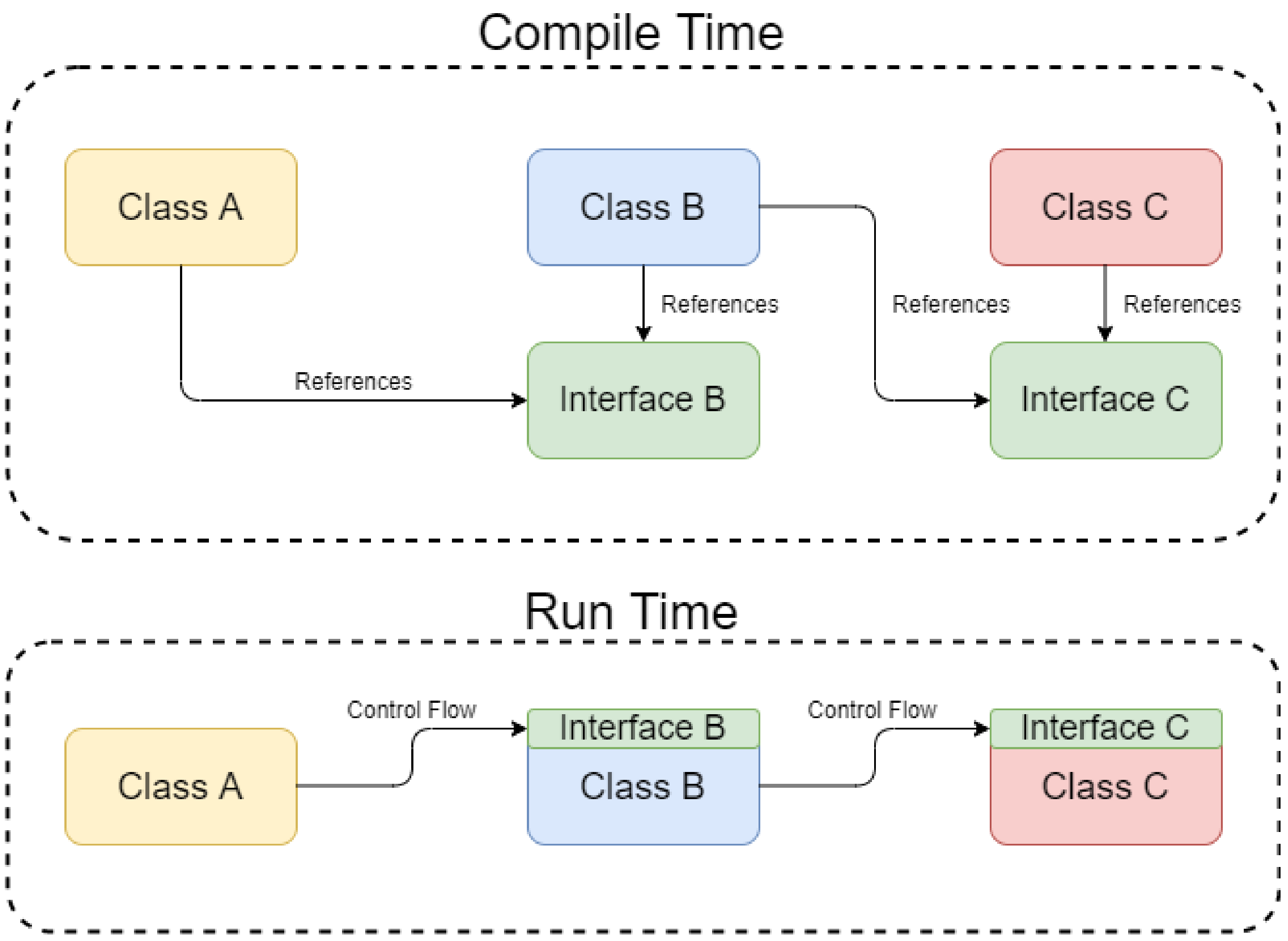
What is Dependency Injection ?

- Dependency Injection (often called just DI) is a software **design pattern that helps us create loosely coupled applications**. It is an implementation of the **Inversion of Control** (IoC) principle, and **Dependency Inversion Principle (D in SOLID)**.

Normal Flow vs DI Flow



Normal Flow vs DI Flow



Dependency Injection approaches

- Constructor Injection
- Method Injection
- Property Injection

Different Service Registration Lifetimes

- **Transient** – We can use this for lightweight, stateless services. Each time the service is called, the new instance is created
- **Scoped** – The instance of the service is created once per request and within that request (scope) it is reused
- **Singleton** – The instance is created only once

Benefits of Dependency injection

- Dependency injection facilitates loose coupling of software components
- The Code is clean and more readable
- Improves the testability and maintainability of the applications
- Allows you to change your implementations without having to change the classes

Thank you

Nash Tech.