Visual Studio LIVE! EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS | Las Vegas

DI Why?
Getting a Grip on Dependency Injection

Jeremy Clark
Developer Betterer
jeremybytes.com

Level: Introductory / Intermediate

Next Generation Skills for Developers

#VSLIVE

Typical Introduction

```
private void BuildMainWindow()
{
    var builder = new ContainerBuilder();

    builder.RegisterType<SQLReader>().As<IPersonReader>()
        .SingleInstance();

    builder.RegisterSource(
        new AnyConcreteTypeNotAlreadyRegisteredSource());

    IContainer Container = builder.Build();

    Application.Current.MainWindow =
        Container.Resolve<PeopleViewerWindow>();
}
```

Dependency Injection

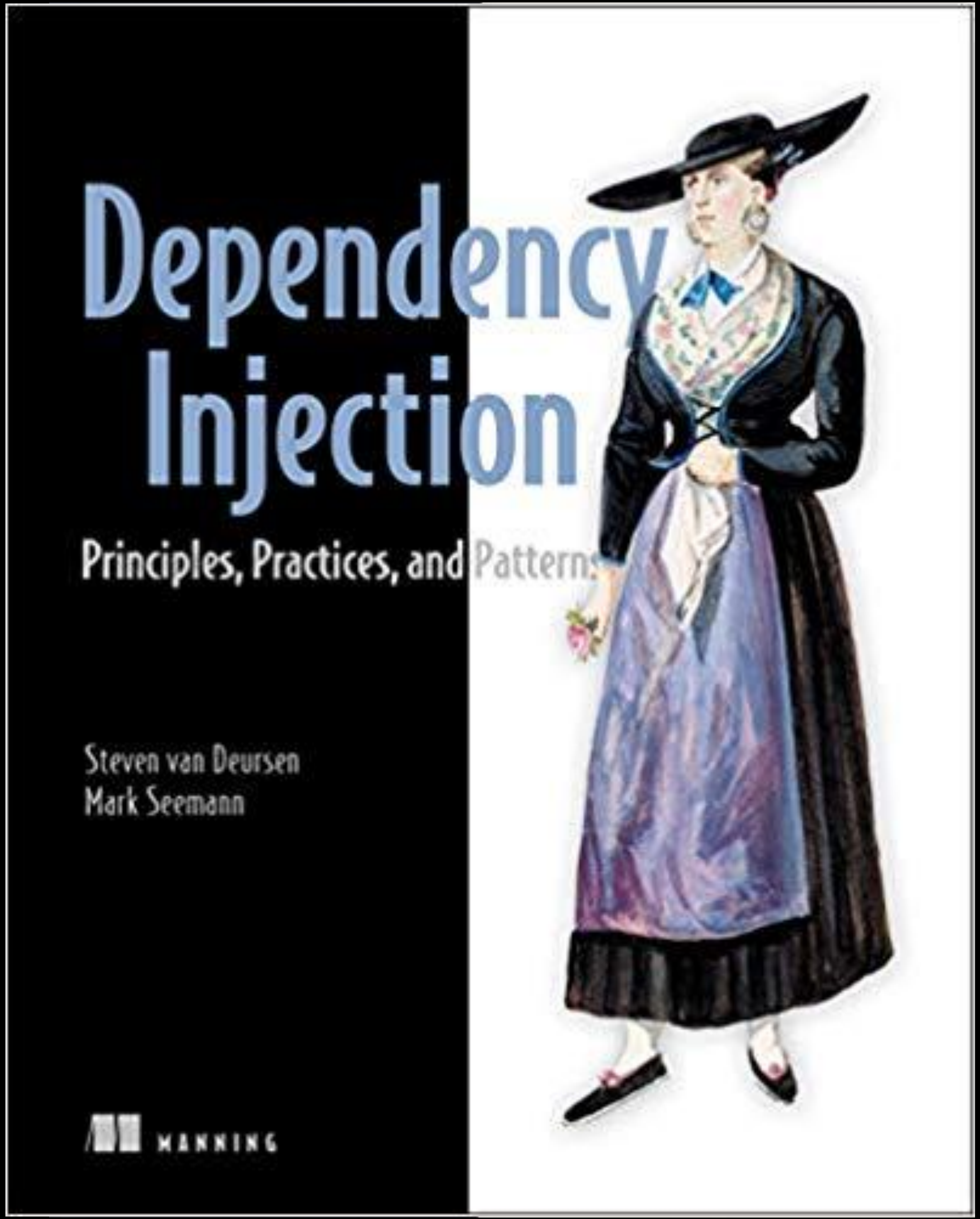
The fine art of making things someone else’s problem.

What Is Dependency Injection?

- Dependency Injection is a set of software design principles and patterns that enable us to develop loosely coupled code.
  - Mark Seemann

Dependency Injection
Principles, Practices, and Patterns

- Mark Seemann
- Steven van Deursen



Primary Benefits

- Extensibility
- Parallel Development
- Maintainability
- Testability
- Late Binding

- Adherence to S.O.L.I.D. Design Principles.

Benefits – Extensibility

Code can be extended in ways not explicitly planned for.

Benefits – Parallel Development

Code can be developed in parallel with less chance of merge conflicts.

Benefits – Maintainability

Classes with clearly defined responsibilities are easier to maintain.

Benefits – Testability

Classes can be unit tested,
i.e., easily isolated from other classes
and components for testing.

Benefits – Late Binding

Services can be swapped with other services without recompiling code.

Benefits – SOLID Principles

- Single Responsibility Principle (SRP)
- Open/Closed Principle (OCP)
- Liskov Substitution Principle (LSP)
- Interface Segregation Principle (ISP)
- Dependency Inversion Principle (DIP)

Dependency Injection Concepts

- DI Design Patterns
  - Constructor Injection
  - Property Injection
  - Method Injection
  - Ambient Context
  - Service Locator
- Dimensions of DI
  - Object Composition
  - Interception
  - Lifetime Management

Dependency Injection Containers

- C# Containers
  - Autofac
  - Ninject

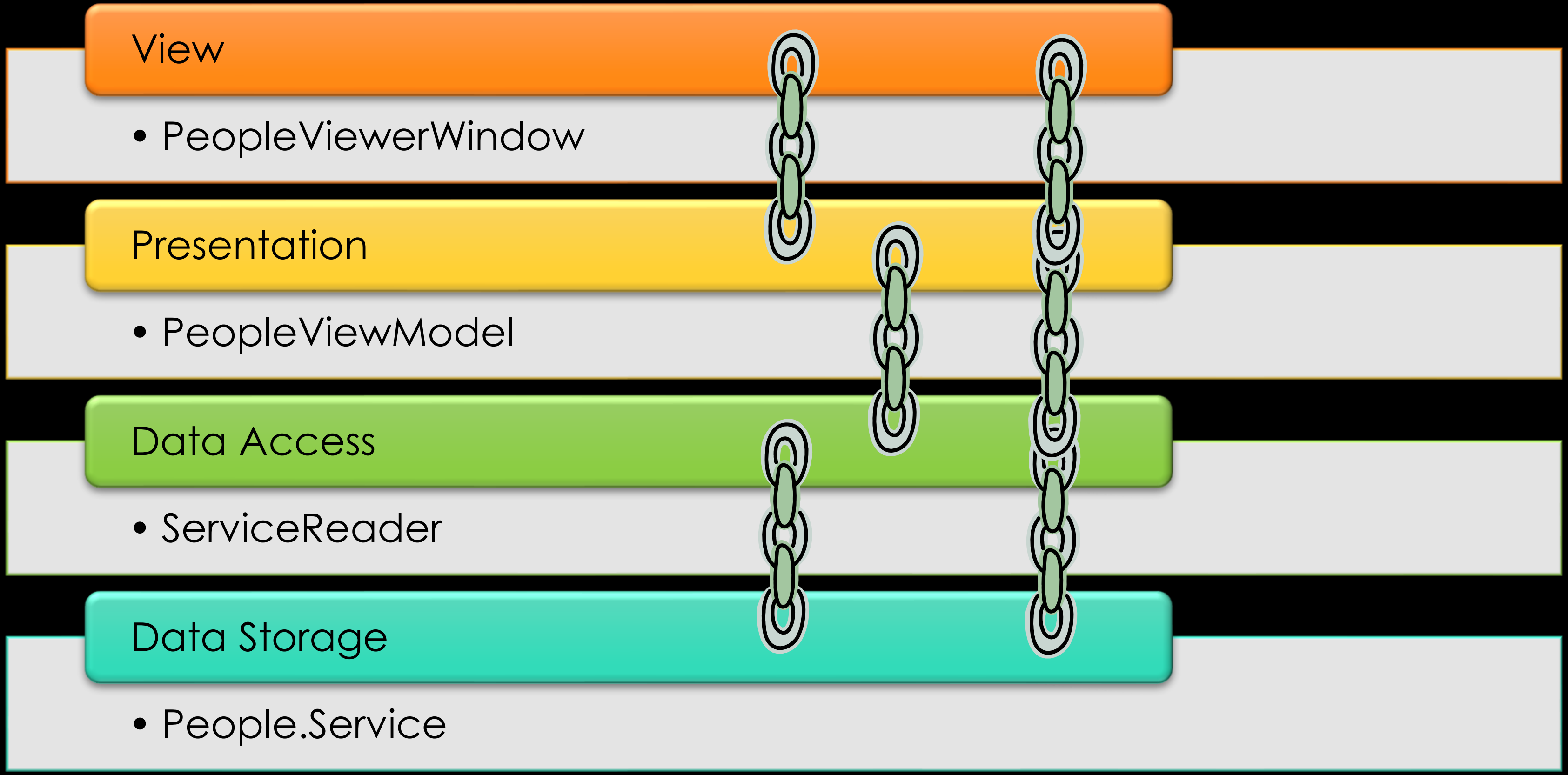
- Frameworks w/ Containers
  - ASP.NET Core
  - Angular
  - Prism

  and many others

Application Layers

- View
  - PeopleViewerWindow
- Presentation
  - PeopleViewModel
- Data Access
  - ServiceReader
- Data Storage
  - People.Service

Tight Coupling



View
- PeopleViewerWindow

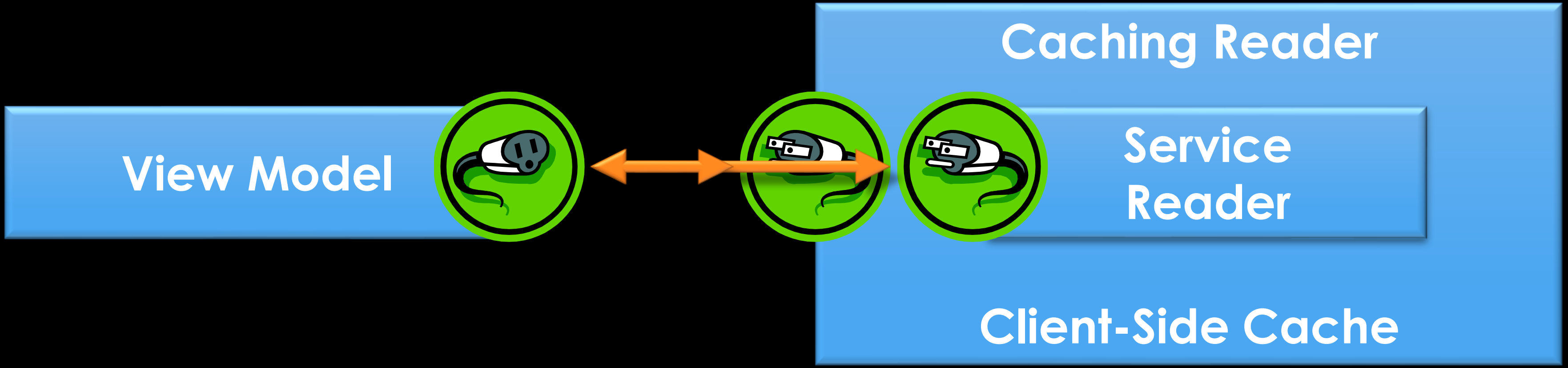
Presentation
- PeopleViewModel

Data Access
- ServiceReader

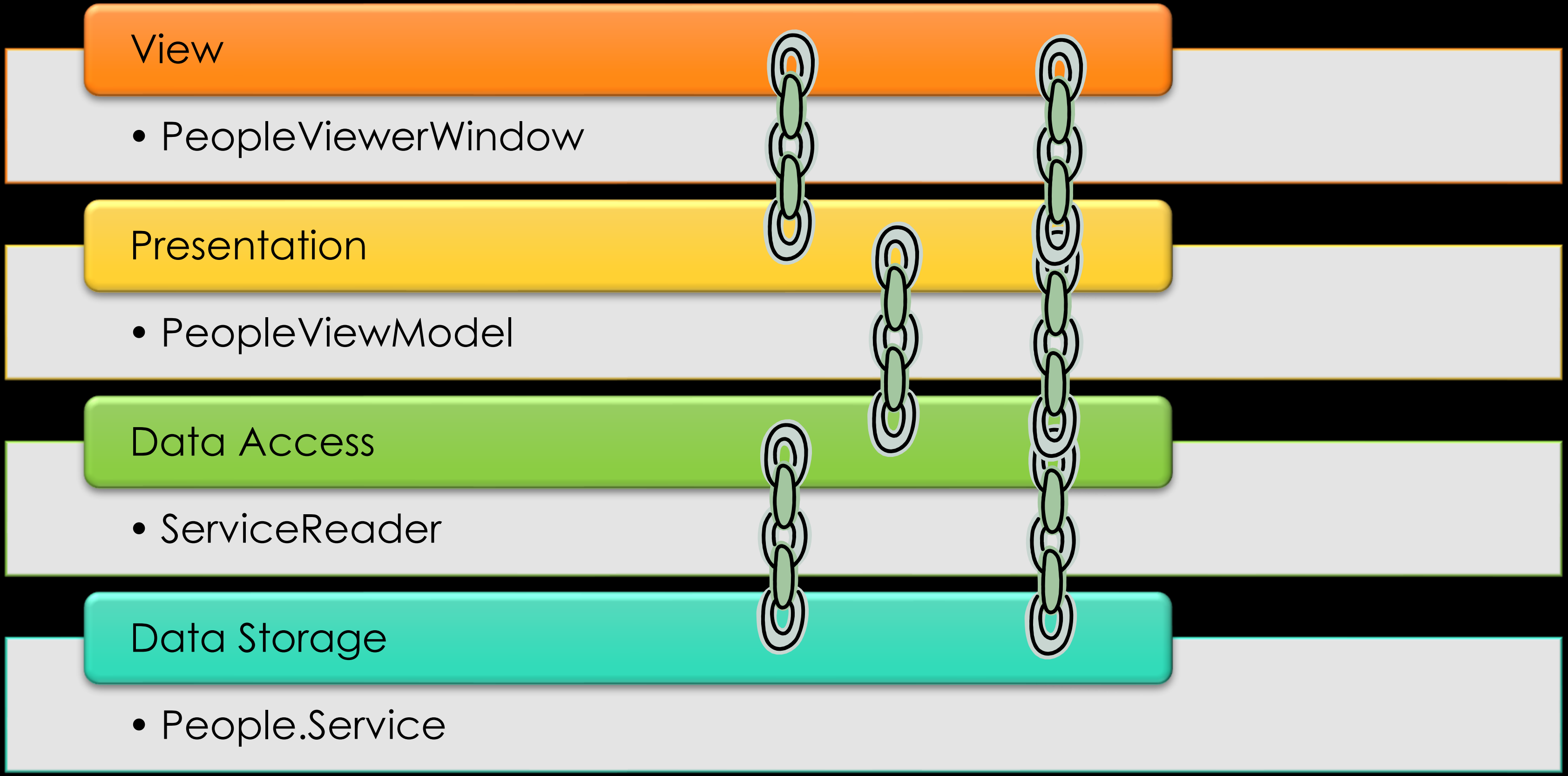
Data Storage
- People.Service

Creating a Caching Reader

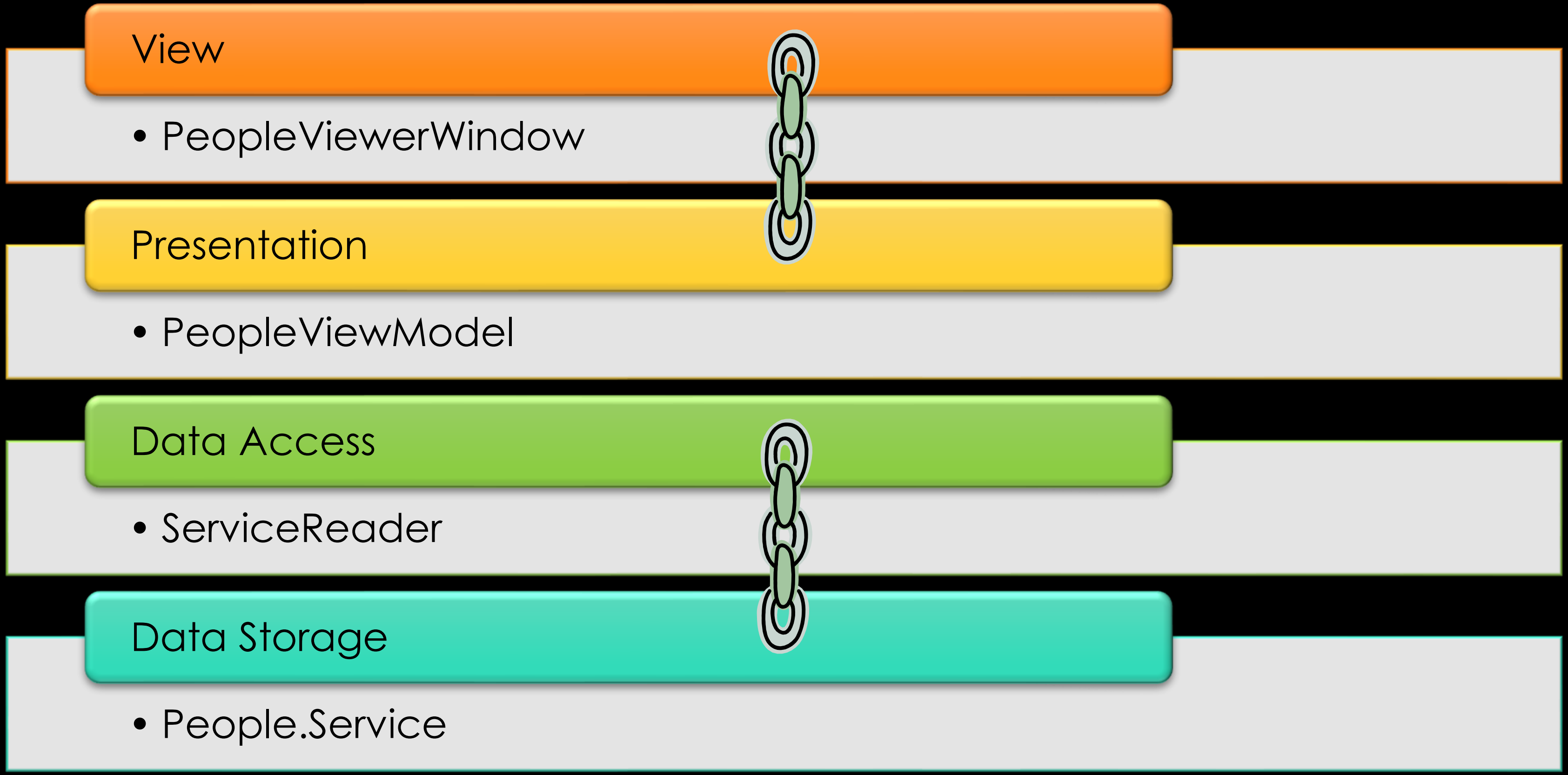
The Decorator Pattern



Loose(r) Coupling



Loose(r) Coupling



View
- PeopleViewerWindow

Presentation
- PeopleViewModel

Data Access
- ServiceReader

Data Storage
- People.Service

Primary Benefits

- Extensibility
- Parallel Development
- Maintainability
- Testability
- Late Binding

- Adherence to S.O.L.I.D. Design Principles.

Dependency Injection Concepts

- DI Design Patterns
  - Constructor Injection
  - Property Injection
  - Method Injection
  - Ambient Context
  - Service Locator

- Dimensions of DI
  - Object Composition
  - Interception
  - Lifetime Management

Session Survey

- Your feedback is very important to us
- Please take a moment to complete the session survey found in the mobile app
- Use the QR code or search for “Converge360 Events” in your app store
- Find this session on the Agenda tab
- Click “Session Evaluation”
- Thank you!

Visual Studio LIVE!
EXPERT SOLUTIONS FOR ENTERPRISE DEVELOPERS

Thank You!

Jeremy Clark

- jeremybytes.com
- jeremy@jeremybytes.com
- @jeremybytes

https://github.com/jeremybytes/vslive2025-lasvegas