meu acerto.

Q&A
Clean Architecture

Ivan Paulovich

www.meuacerto.com.br

September 3th, 2020

@ivanpaulovich

https://paulovich.net

@ivanpaulovich

Paulovich.NET

Microsoft® Most Valuable Professional MVP Reconnect

#CleanArchitectureManga 🍄

Ivan Paulovich

ivanpaulovich

♡ Sponsor

Tech Lead, Software Engineer, 20+ GitHub projects about Clean Architecture, SOLID, DDD and TDD. Speaker/Streamer. Microsoft rMVP.

- Nordax Bank
- Stockholm, Sweden
- Sign in to view email
- https://paulovich.net
- @ivanpaulovich

CleanArchitectureVSCodeSnippets

Clean Architecture C# Snippets for Visual Studio Code

TypeScript ☆ 6

FluentMediator

FluentMediator is an unobtrusive library that allows developers to build custom pipelines for Commands, Queries and Events.

C# ☆ 83 11

clean-architecture-manga

Template

🌀 Clean Architecture with .NET Core 3.1, C# 8 and React+Redux. Use cases as central organizing structure, completely testable, decoupled from frameworks

C# ☆ 1.6k 304

todo

✅ Commad-Line Task management with storage on your GitHub 🔥

C# ☆ 94 16

dotnet-new-caju

Template

Learn Clean Architecture with .NET Core 3.0 🔥

C# ☆ 202 30

event-sourcing-jambo

An Hexagonal Architecture with DDD + Aggregates + Event Sourcing using .NET Core, Kafka e MongoDB (Blog Engine)

C# ☆ 146 62

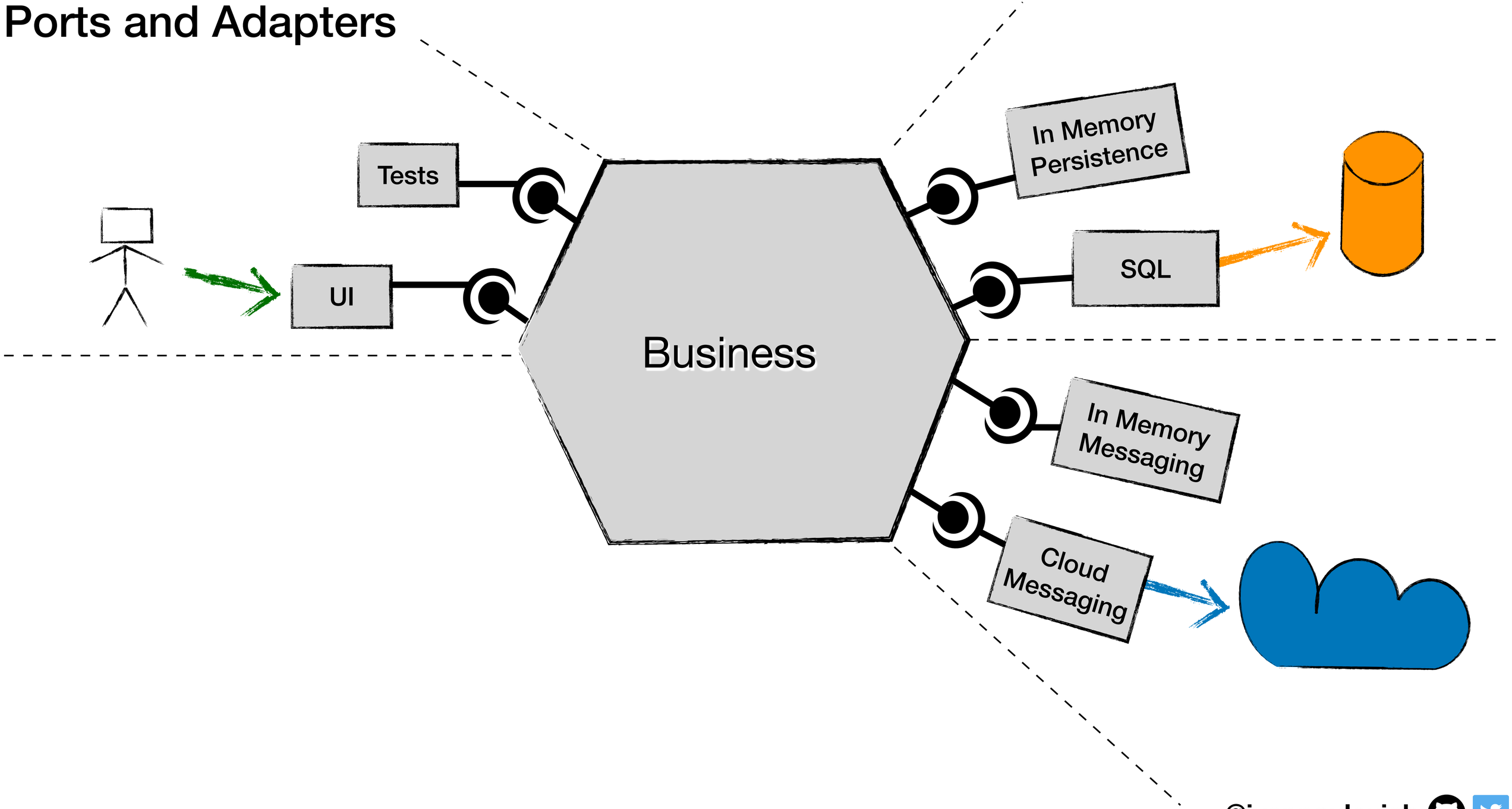
Agenda

- Hexagonal Architecture Style
  - Dependency Inversion Principle
  - Ports and Adapters Pattern
  - Test-Driven Design Development (Practices)
- Clean Architecture Style
  - Principles, Patterns and Practices.
- Walkthrough an Use Case
- Q&A

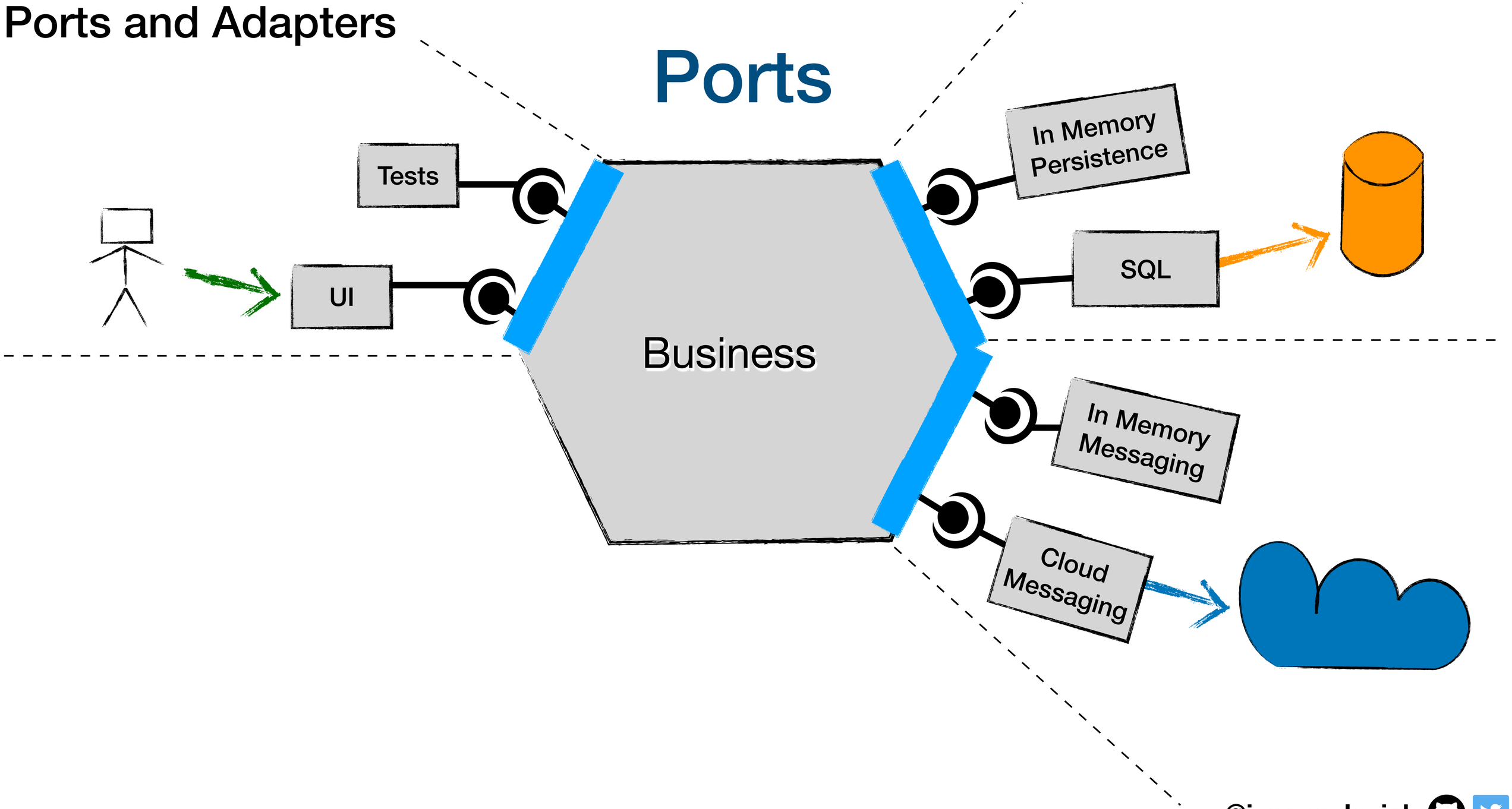
Dependency Inversion Principle (DIP)

- High-level modules should not depend on low-level modules. Both should depend on abstractions.
- Abstractions should not depend on details.
- Details should depend on abstractions.

Ports and Adapters



Ports and Adapters



Ports

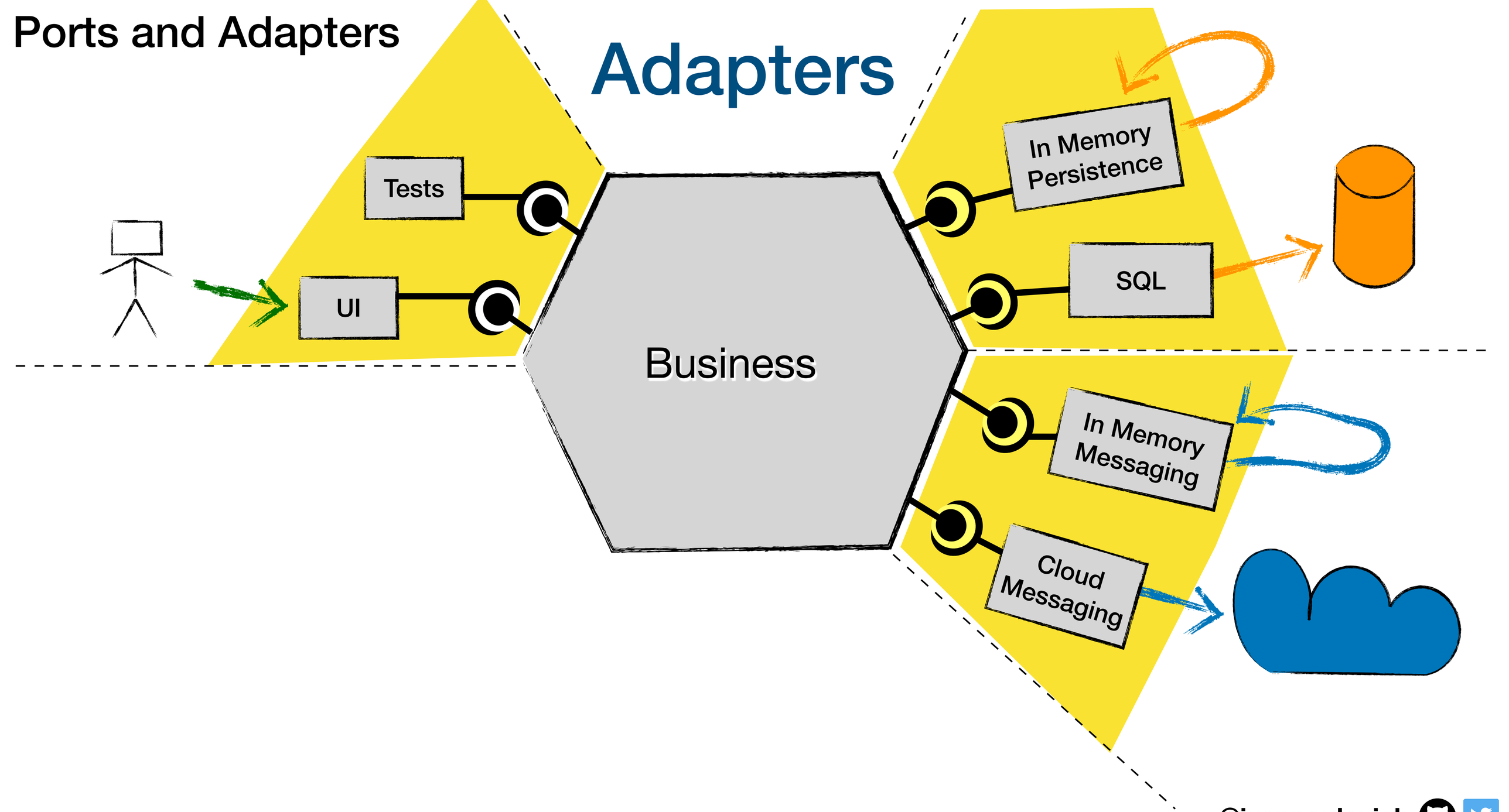
```
3 references | You, a month ago | 1 author (You)
public interface IDepositUseCase
{
    /// <summary>
    ///     Executes the Use Case.
    /// </summary>
    /// <param name="accountId">AccountId.</param>
    /// <param name="amount">Positive amount to deposit.</param>
    /// <param name="currency">Currency from amount.</param>
    /// <returns>Task.</returns>
    3 references
    Task Execute(Guid accountId, decimal amount, string currency);

    /// <summary>
    ///     Sets the Output Port.
    /// </summary>
    /// <param name="outputPort">Output Port</param>
    3 references
    void SetOutputPort(IOutputPort outputPort);
}
```

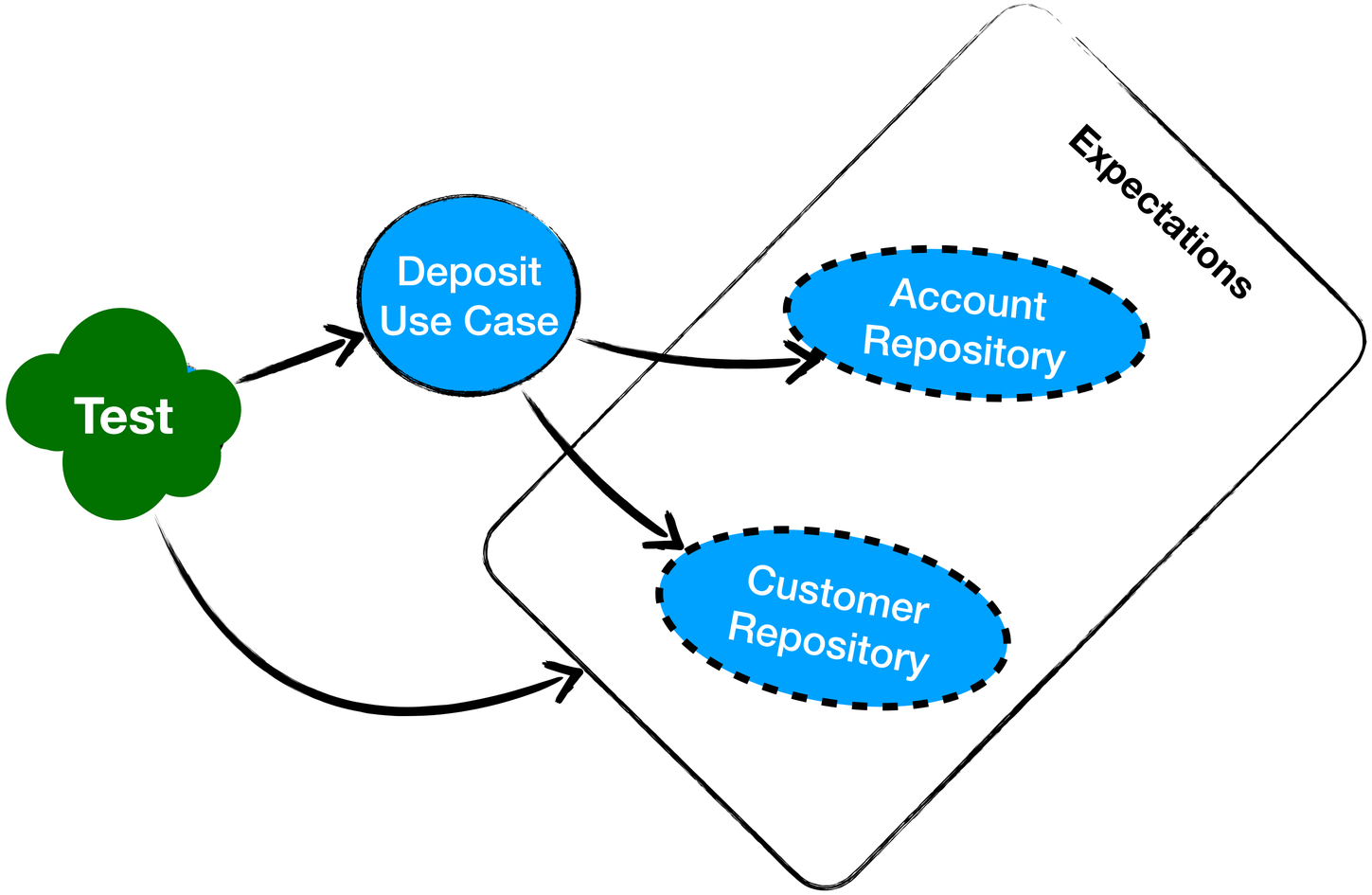
```
public interface IAccountRepository
{
    Task<IAccount> GetAccount(AccountId accountId);
    Task<IList<IAccount>> GetBy(CustomerId customerId)
    Task Add(IAccount account, ICredit credit);
    Task Update(IAccount account, ICredit credit);
    Task Update(IAccount account, IDebit debit);
    Task Delete(IAccount account);
}
```

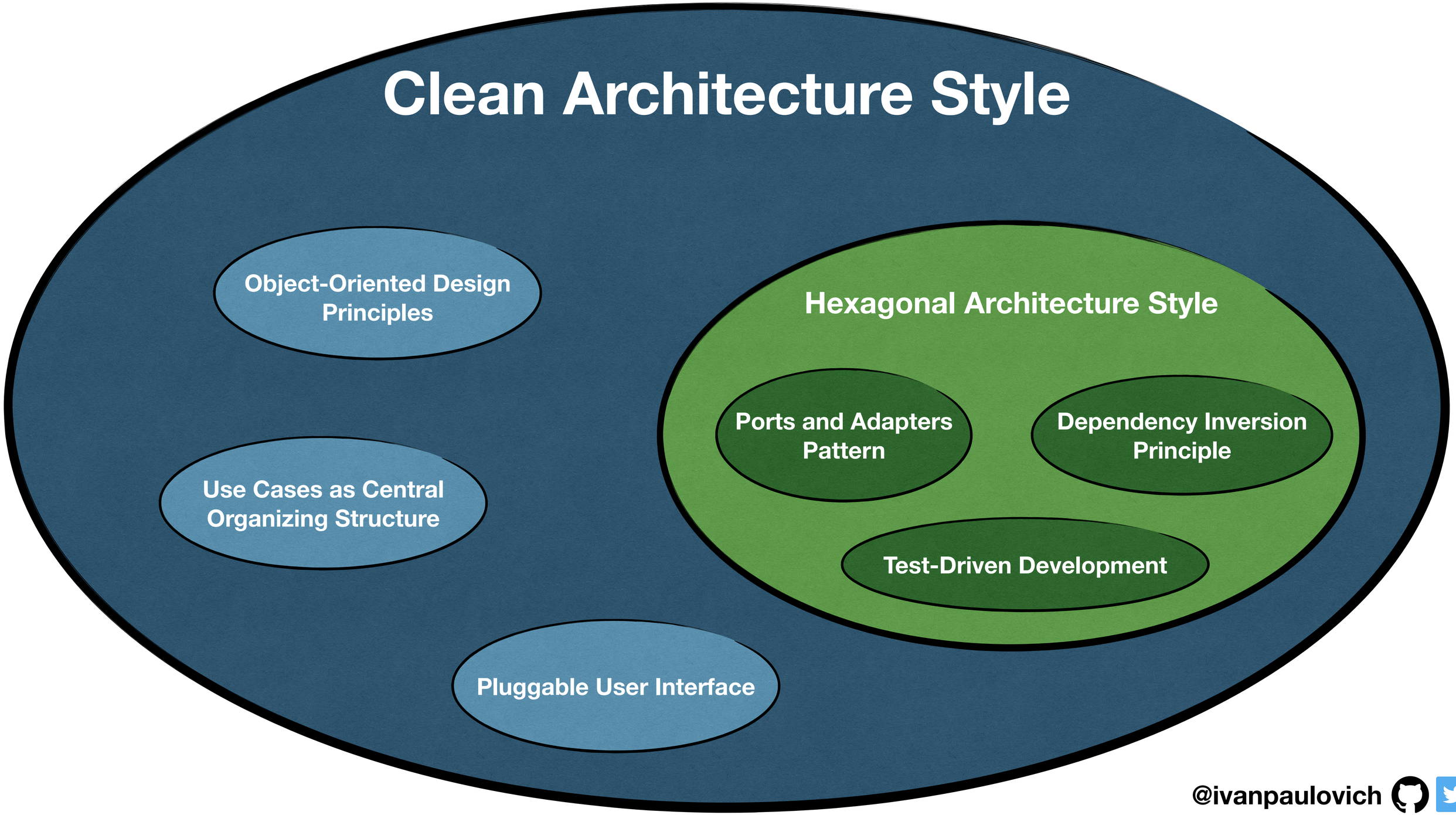
@ivanpaulovich

Ports and Adapters



Test-Driven Development (Practices)

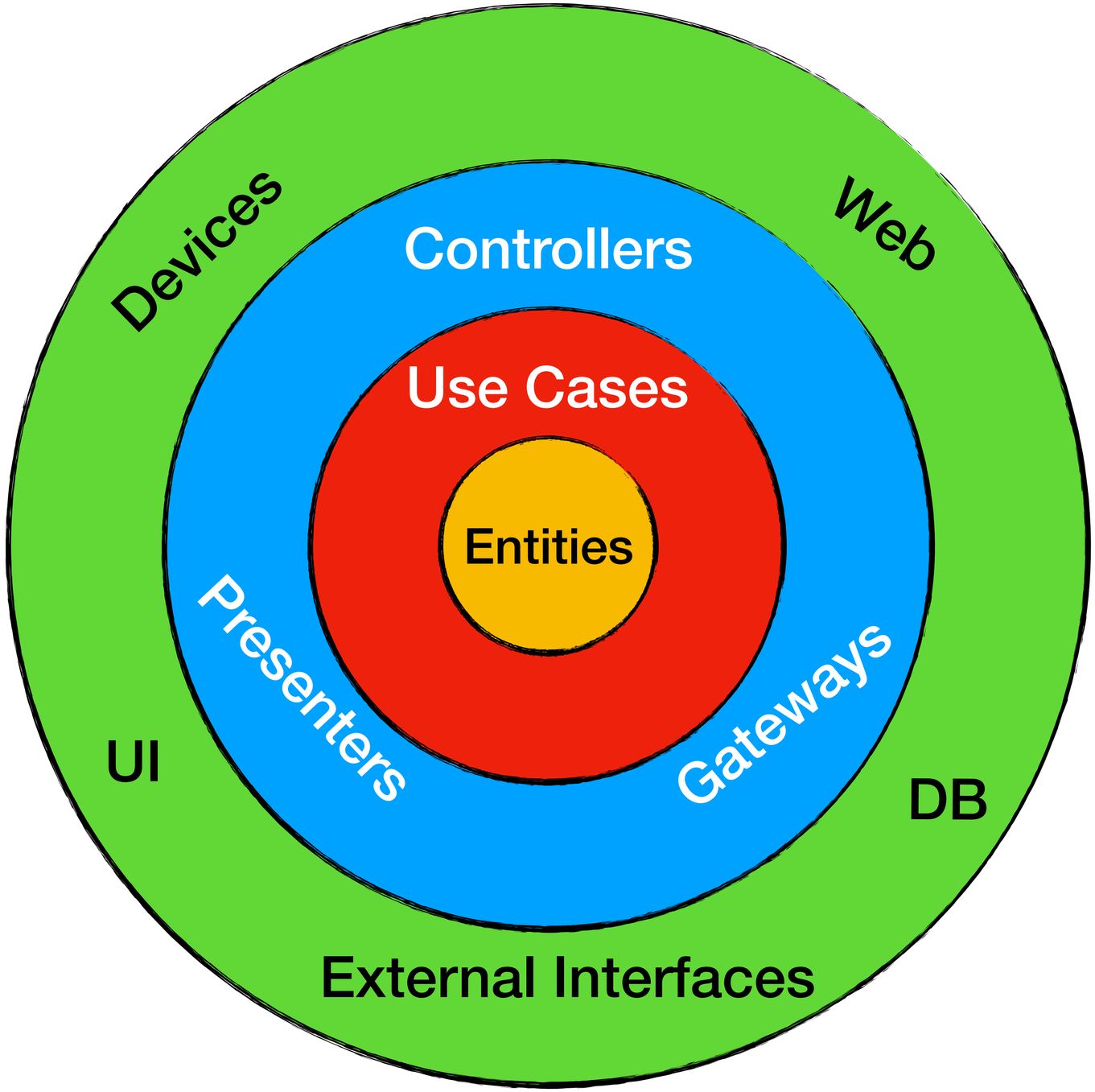



Clean Architecture Style
Object-Oriented Design Principles
Use Cases as Central Organizing Structure
Pluggable User Interface
Hexagonal Architecture Style
Ports and Adapters Pattern
Dependency Inversion Principle
Test-Driven Development

Principles

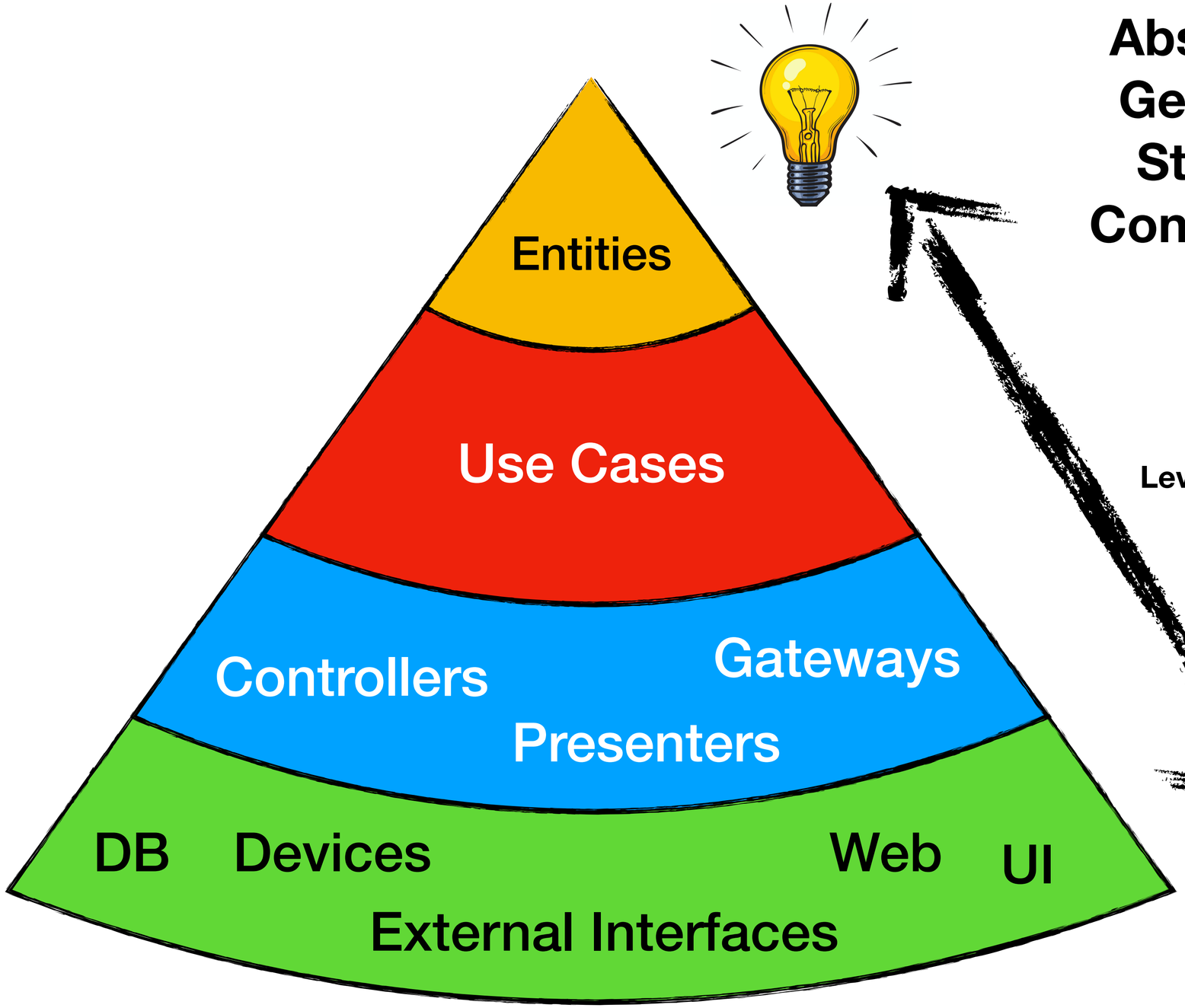
- SOLID Principles.
- Package Cohesion Principles:
  - The Release Reuse Equivalency Principle.
  - The Common Closure Principle.
  - The Common Reuse Principle.
- Coupling Packages Principles:
  - The Acyclic Dependencies Principle.
  - The Stable Dependencies Principle.
  - The Stable Abstractions Principle.

Clean Architecture



- Abstractness increases with stability.
- Modules depend in the direction of stability.
- Classes that change together are packaged together.

Clean Architecture



**Abstract,
General,
Stable,
Consistent**

Level

**Concrete,
Specific,
Unstable,
Inconsistent**

@ivanpaulovich

Patterns

- Use case as central organizing structure.
- Ports and Adapters Pattern from Hexagonal Architecture.

Practices

- The same TDD from Hexagonal Architecture Style.

Wrapping up Clean Architecture

- Clean Architecture is about usage and the use cases are the central organizing structure.
- Use cases implementation are guided by tests.
- The User Interface and secondary actors are designed to fulfil the core needs.
- Defer decisions by implementing the simplest component first.

Walkthrough an Use Case