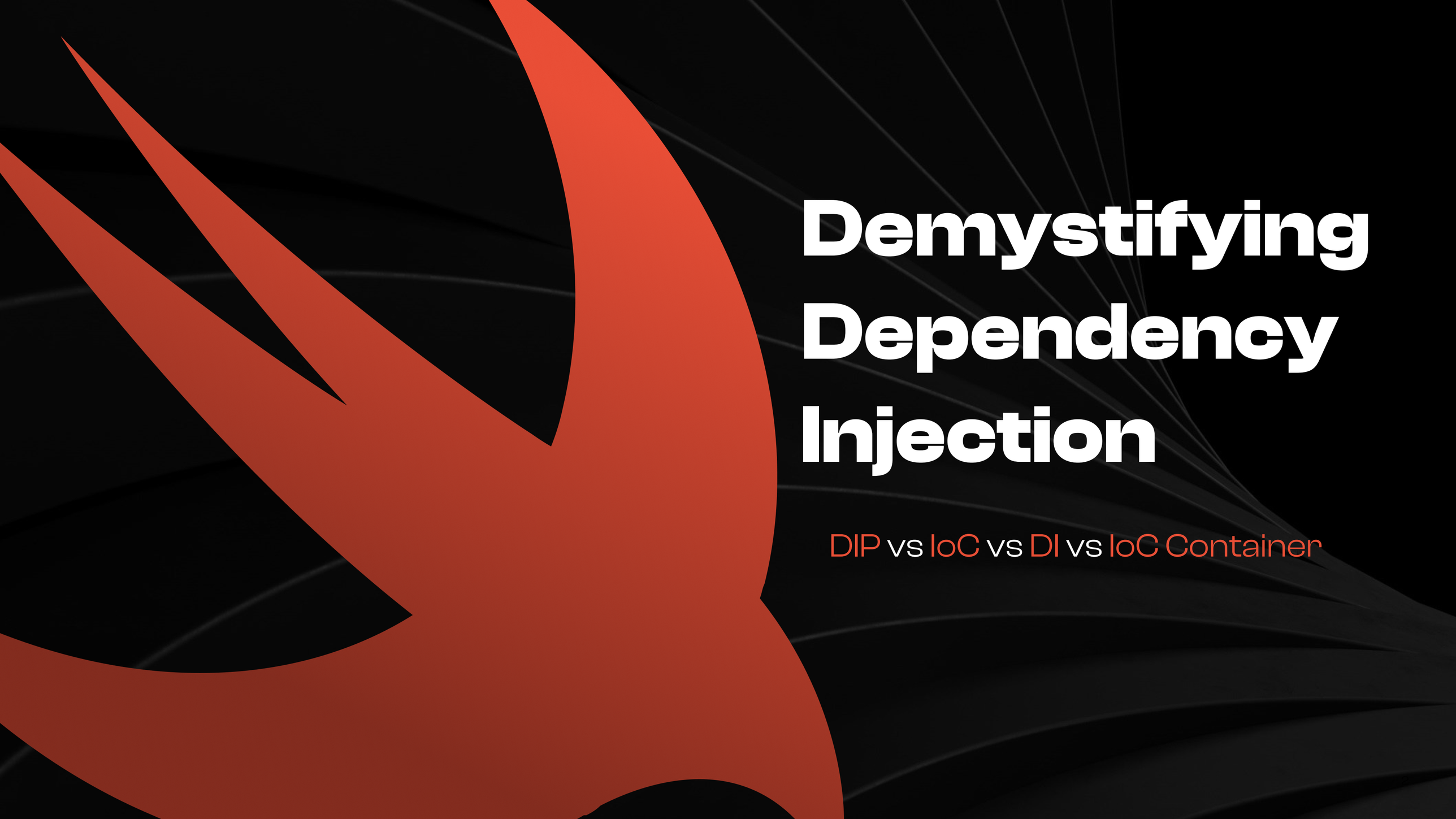
Mostafa Nafie

Senior Software Engineer in iOS

nafie.dev


Demystifying
Dependency
Injection
DIP vs IoC vs DI vs IoC Container

Introduction

- Whatever your level.
- Interviews and everyday tasks.
- There's a base idea and tangents.
- How to approach a new topic in general.

Table of contents

Prerequisites

Principle

The theoretical concept.

The What?

Principle

The theoretical concept.
The What?

Pattern

How this concept is applied practically.
The How?

Principle

The theoretical concept.
The What?

Pattern

How this concept is applied practically.
The How?

What if we do not follow these principles?

Why these principals are here in the first place.
The Why?

What's our problem?

Table of contents

What?
Principles

Object Oriented Design (OOD)

Object Oriented Design (OOD)

↓

SOLID

Object Oriented Design (OOD)

↓

SOLID

↓

Dependency Inversion Principle (DIP)

Object Oriented Design (OOD)

↓

SOLID

↓

Dependency Inversion Principle (DIP)

Inversion of Control (IoC)

Table of contents

Why?

Maintainability

Easily substitute the dependencies; in case of changing a 3rd party dependency.

Maintainability

Easily substitute the dependencies; in case of changing a 3rd party dependency.

Testability

Substituting the dependency with a test double.

Maintainability

Easily substitute the dependencies; in case of changing a 3rd party dependency.

Testability

Substituting the dependency with a test double.

By achieving loose coupling

Table of contents

How?

Patterns

Applying the Patterns

Dependency Injection (DI)

- Separating the creation of objects from the classes that use them.

Dependency Injection (DI)

- Separating the creation of objects from the classes that use them.

Manual DI

- Constructor Injection
- Property Injection

Achieve Loose Coupling

1.

Apply DIP through Abstraction
Hide the dependency behind an abstraction.

2.

Apply IOC through DI
Extract the dependency and inject it.

3.

Switch the dependency as needed.

Dependency Injection (DI)

- Separating the creation of objects from the classes that use them.

Manual DI

- Constructor Injection
- Property Injection

Automatic DI

IoC Container

- Registering
- Resolving
- Managing dependency lifecycle

Write a simple IoC container in Swift

Write a simple IoC container in Swift

swift Copy code

```swift
import Foundation

// Protocol for the service
protocol Service {
    func performAction()
}

// Implementation of the service
class ConcreteService: Service {
    func performAction() {
        print("Action performed by ConcreteService")
    }
}

// IoC Container
class IoCContainer {
    private var services: [String: Any] = [:]

    // Register a service with a factory closure
    func register<T>(type: T.Type, factory: @escaping () -> T) {
        let key = String(describing: type)
        services[key] = factory
    }

    // Resolve a service
    func resolve<T>(type: T.Type) -> T? {
        let key = String(describing: type)
        if let factory = services[key] as? () -> T {
            return factory()
        }
        return nil
    }
}

// Example usage
let container = IoCContainer()
container.register(type: Service.self) { ConcreteService() }

if let service = container.resolve(type: Service.self) {
    service.performAction()  // Outputs: "Action performed by ConcreteService"
} else {
    print("Service not found")
}
```

Popular Dependency Injection Frameworks

Swinject

Testing no status | Carthage compatible | pod v2.9.1 | license MIT

platform iOS | macOS | tvOS | watchOS | Linux | Swift 4.2-5.4 | Reviewed by Hound

Swinject is a lightweight dependency injection framework for Swift.

Dependency injection (DI) is a software design pattern that implements Inversion of Control (IoC) for resolving dependencies. In the pattern, Swinject helps your app split into loosely-coupled components, which can be developed, tested and maintained more easily. Swinject is powered by the Swift generic type system and first class functions to define dependencies of your app simply and fluently.

Popular Dependency Injection Frameworks

Resolver

An ultralight Dependency Injection / Service Locator framework for Swift 5.x on iOS.

Note: Resolver is now officially deprecated and replaced by my new dependency injection system, Factory. Factory is a true container-based dependency injection system that's compile-time safe and is smaller, lighter, and faster than Resolver. As good as Resolver is, Factory is better.

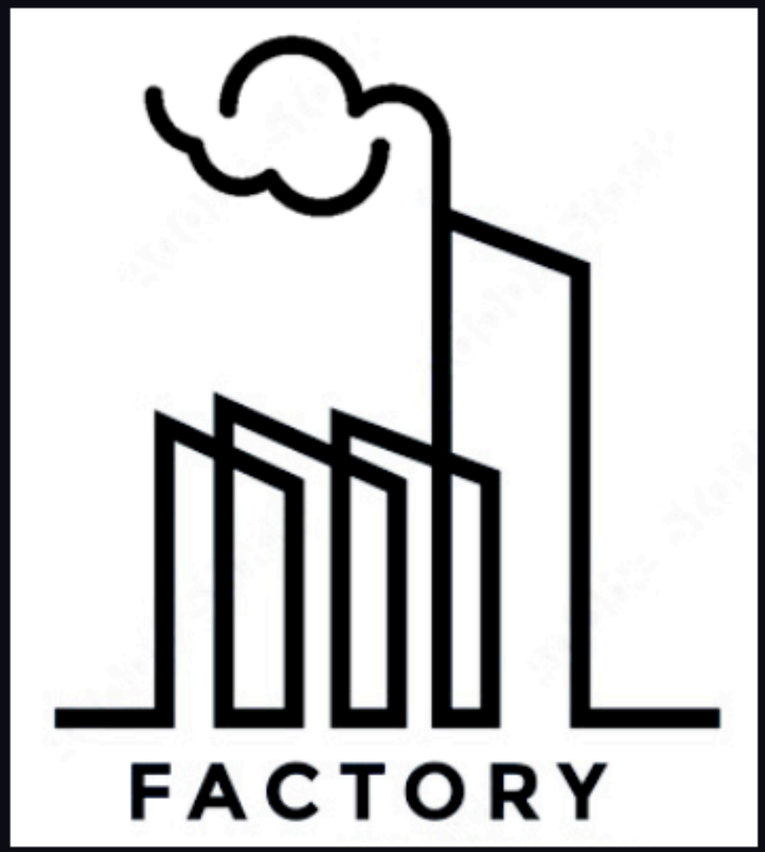
Introduction

Dependency Injection frameworks support the Inversion of Control design pattern. Technical definitions aside, dependency injection pretty much boils down to:

| Giving an object the things it needs to do its job.

That's it. Dependency injection allows us to write code that's loosely coupled, and as such, easier to reuse, to mock, and to test.

Popular Dependency Injection Frameworks



A new approach to Container-Based Dependency Injection for Swift and SwiftUI.

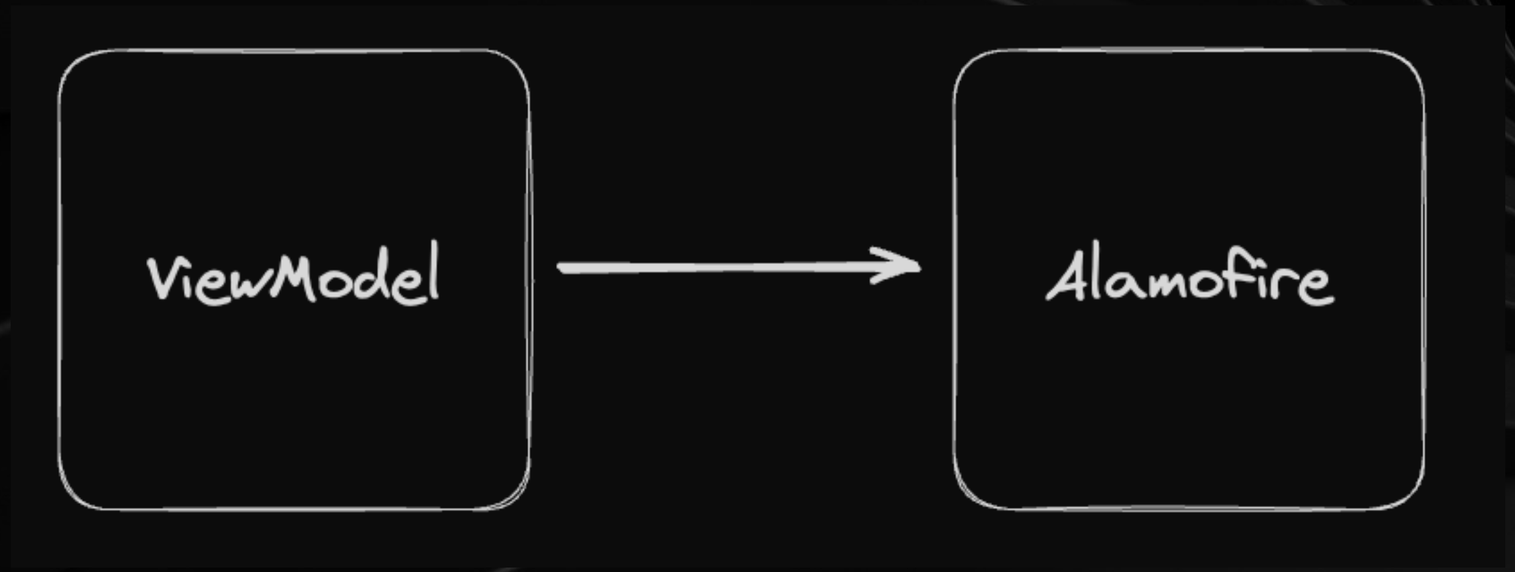
Outcomes

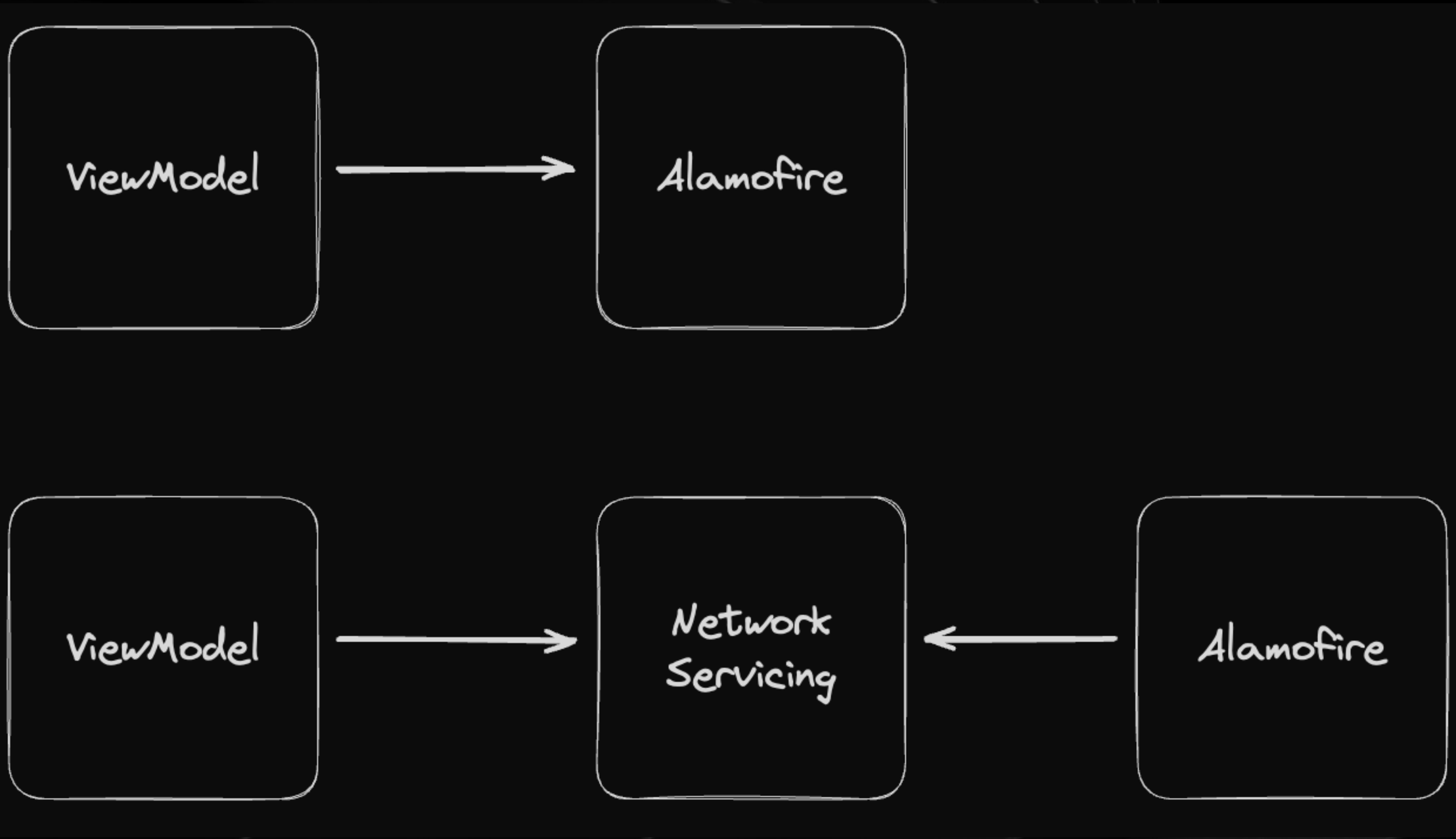
By now you should be able to understand the following

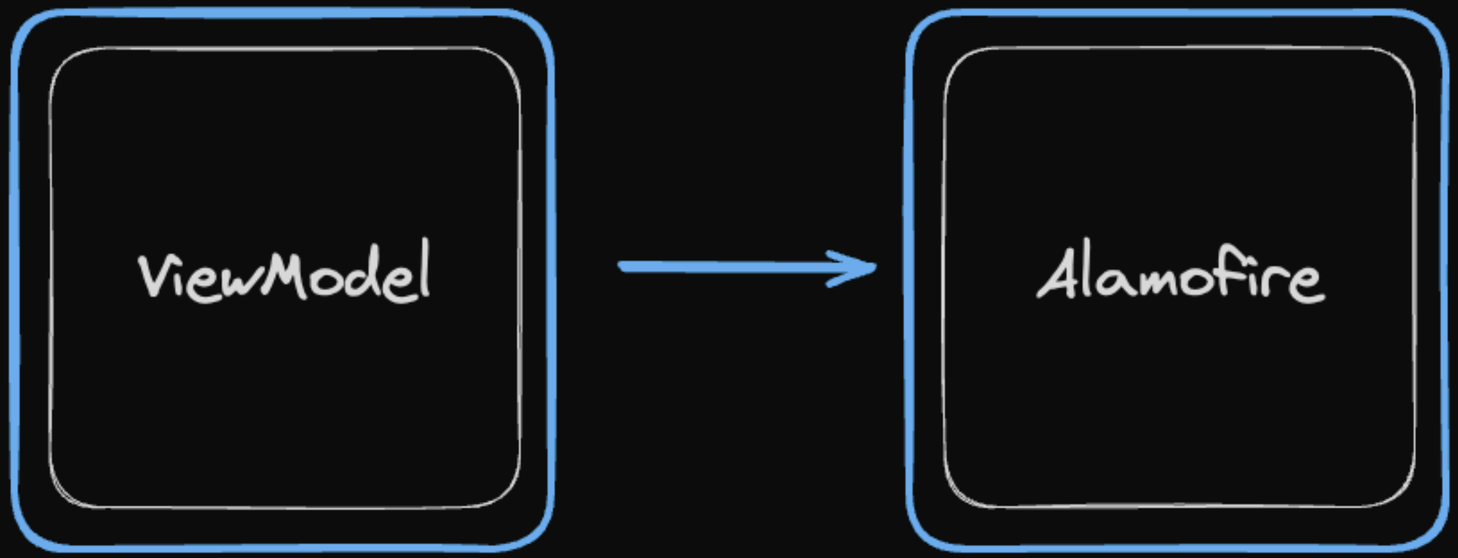
- The differences between:
  - Principles:
    - Dependency Inversion Principle (DIP)
    - Inversion of Control (IoC)
  - Patterns:
    - Dependency Injection (DI)
    - IoC Container
- The theoretical basis of these principles.
- The practical usage of these patterns.

One more thing...

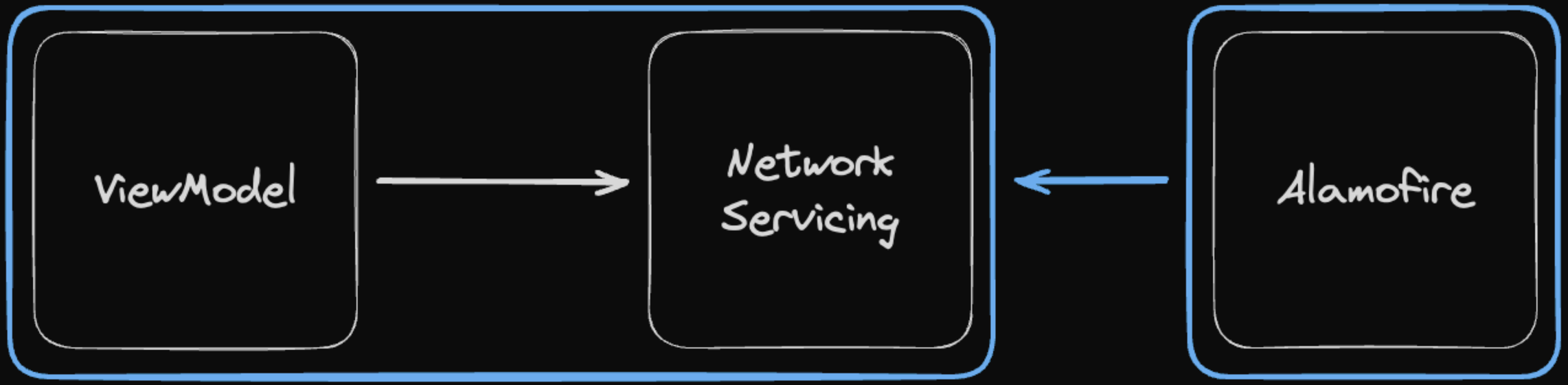
What's the inversion in DIP?


ViewModel
Alamofire


ViewModel
Alamofire
ViewModel
Network Servicing
Alamofire


ViewModel
Alamofire

Modularity


ViewModel
Network
Servicing
Alamofire

Thank You!

nafie.dev