Modularisation Strategies 🏗️

G Abhisek

- Modules 🧱
- Modularisation 🏗️
  - Benefits
- Common coding strategies of Modularisation 📋
- Challenges of Modularisation at Scale 📈

Modules

- Distinct and self contained units
- Independent functionality
- Focused on specific feature or functionality
- Logically connected units


Jio WiFi
7:27 PM
27%
Find services, food, or places
gopay
Rp20
Tap to top up
Top Up
Pay
Explore
2
GoRide
GoCar
GoFood
GoSend
<10k
GoMart
GoTagihan
GoTransit
More
tokopedia
Belanja makin hemat
Dapetin diskon dan harga spesialnya di Tokopedia sekarang sebelum kehabisan!
KETINGGALAN TV Rp200rb?
Cek WAR Diskon di Tokopedia!
Tiap 14:00 WIB
Home
Promo
Orders
Chat

```swift
class GoPayHomeViewController: UIViewController {
    init(config: GoPayHomeConfig) {•••}

    required init?(coder: NSCoder) {•••}

    func moveP2PFlow() {•••}
}
```

Payments Home

```swift
class GoPayP2PViewController: UIViewController {
    init(config: GoPayP2PConfig) {•••}

    required init?(coder: NSCoder) {•••}

    func moveToP2PFlow() {•••}

    func moveToTransactionHistoryPage() {•••}
}
```

P2P Page

```swift
class GoFoodCartViewController: UIViewController {
    init(config: GoFoodCartConfig) {
        super.init(nibName: nil, bundle: nil)
    }

    required init?(coder: NSCoder) {
        fatalError("init(coder:) has not been implemented")
    }

    func makePaymentForOrders() {
        let p2pViewController = GoPayP2PViewController(config: GoPayP2PConfig())
        navigationController?.pushViewController(p2pViewController, animated: true)
    }
}
```

```swift
class GoFoodHomeViewController: UIViewController {
    init(config: GoFoodHomeConfig) {•••}

    required init?(coder: NSCoder) {•••}

    func selectProduct(productId: String) {•••}
}
```

GoFood Home

🍔 ➡️ 💰

```swift
class GoFoodProductDetailViewController: UIViewController {
    init(config: GoFoodProductConfig) {•••}

    required init?(coder: NSCoder) {•••}

    func showProduct() {•••}
}
```

GoFood Product Detail

Issues

- Reduced Reusability
- Complexity and Monoliths
- Testing Challenges
- Scalability Issues
- Higher Risk of Bugs and Errors

🚫

Strategies 📋

Choosing Architecture

- Separation of concerns
- Works to segregate your codebase
- Addresses navigation, business logic separation, and scalability.
- MVVM, VIPER, TCA, Redux, etc
- Modules represented by interface and communicate via them.

Separating modules

- Independent business verticals
  - GoPay, GoFood, GoMart, GoRide
- Reusable logically connected units
  - Utilities, Networking, Analytics manager

```
class GoPayHomeViewController: UIViewController {
    init(config: GoPayHomeConfig) {•••}

    required init?(coder: NSCoder) {•••}

    func moveP2PFlow() {•••}
}
```

Payments Home

GoPay Module

```
class GoPayP2PViewController: UIViewController {
    init(config: GoPayP2PConfig) {•••}

    required init?(coder: NSCoder) {•••}

    func moveToP2PFlow() {•••}

    func moveToTransactionHistoryPage() {•••}
}
```

P2P Page

```
class GoFoodHomeViewController: UIViewController {
    init(config: GoFoodHomeConfig) {•••}

    required init?(coder: NSCoder) {•••}

    func selectProduct(productId: String) {•••}
}
```

GoFood Home

GoFood Module

```
class GoFoodProductDetailViewController: UIViewController {
    init(config: GoFoodProductConfig) {•••}

    required init?(coder: NSCoder) {•••}

    func showProduct() {•••}
}
```

GoFood Product Detail

```swift
class GoPayInterface {

    func goToP2PFlow(config:
GoPayP2PConfig) {
        // Navigate to P2P flow
    }

}
```

```swift
class GoFoodInterface {
    let goPayInterface: GoPayInterface

    func proceedToPayment() {
        goPayInterface.goToP2PFlow(config: GoPayP2PConfig())
    }
}
```

```swift
class GoFoodCartViewController:
UIViewController {
    let goFoodInterface: GoFoodInterface

    func makePaymentForOrders() {
        goFoodInterface.proceedToPayment()
    }
}
```

Issues

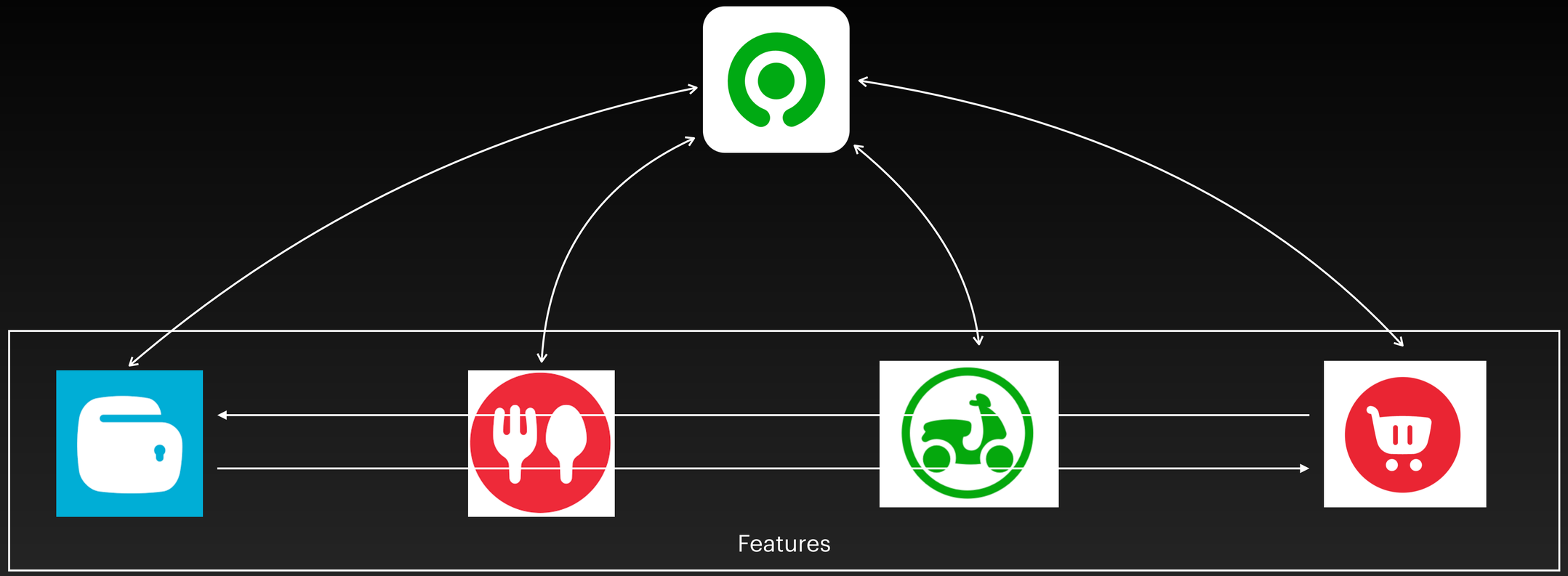
- Limited Reusability
- Testing isolation
- Code ownership and Collaboration
- Interface segregation
- Dependency management

Modularisation at Scale 📈

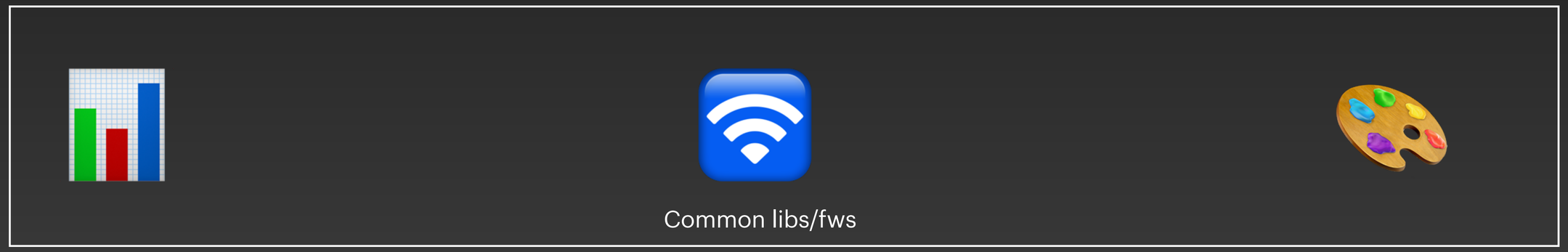
Modules as Frameworks 💼

```swift
public class GoPayInterface {
    func goToP2PFlow(config: GoPayP2PConfig) {
        // Navigate to P2P flow
    }
}

public class GoFoodInterface {
    let goPayInterface: GoPayInterface

    func proceedToPayment() {
        goPayInterface.goToP2PFlow(config: GoPayP2PConfig())
    }
}
```

```swift
import GoPayModule

import GoFood
```

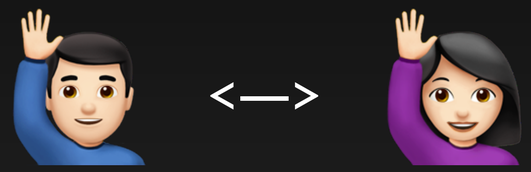


Cyclic Dependencies

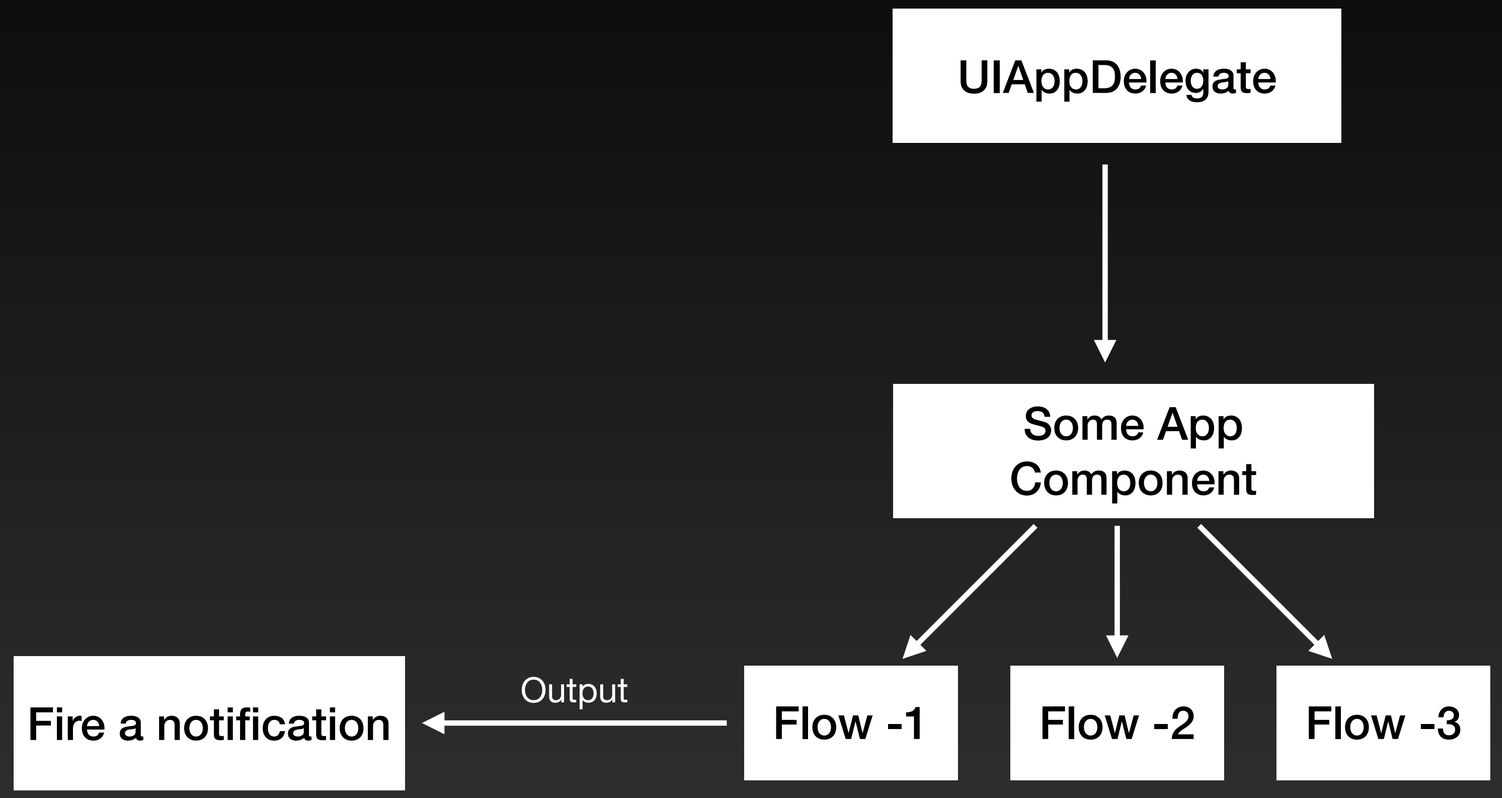


Communication

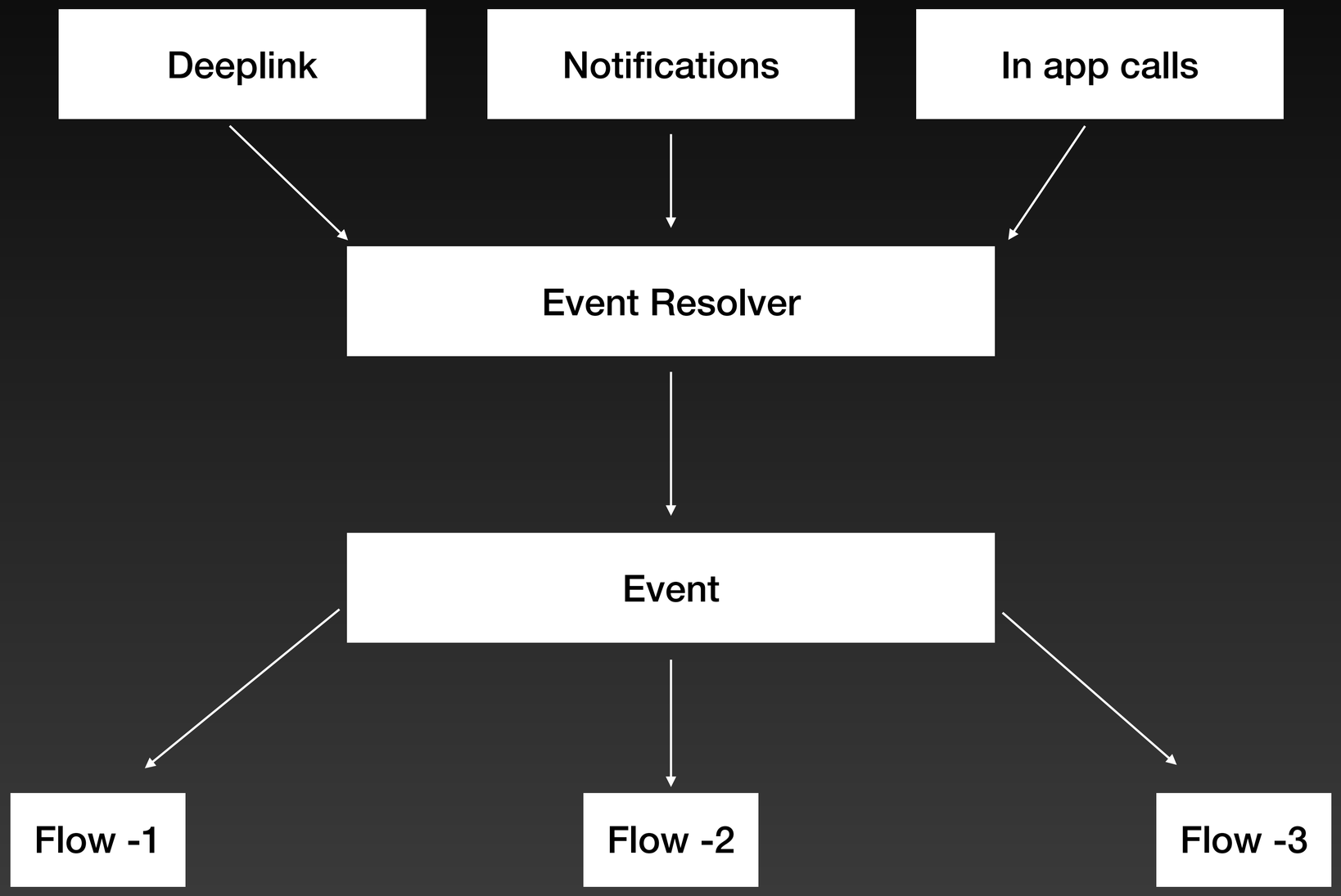
- Input <—> Output
- Deeplink
- Notifications
- In app navigation



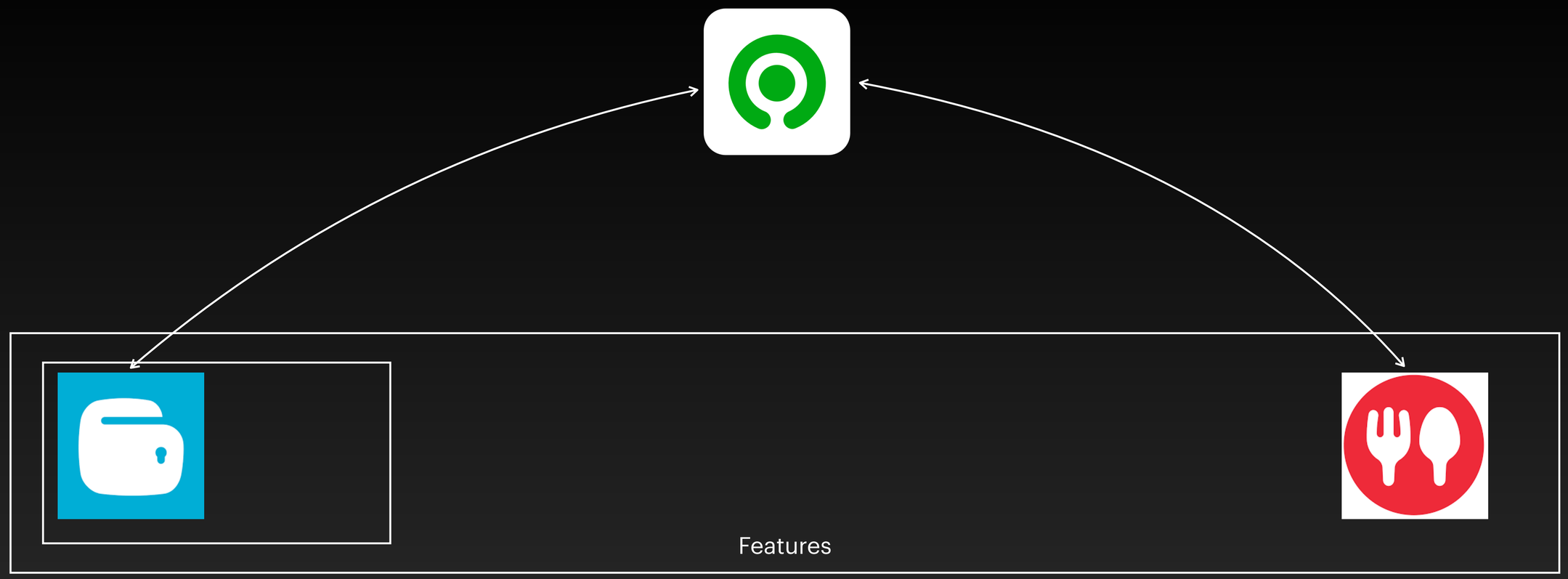
Deeplink


UIAppDelegate
Some App Component
Flow -1
Flow -2
Flow -3
Output
Fire a notification

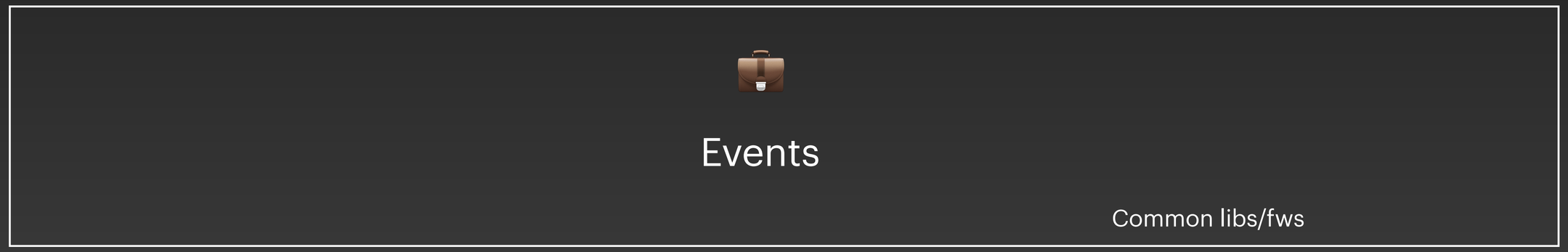
Input - Event, Output - Callback

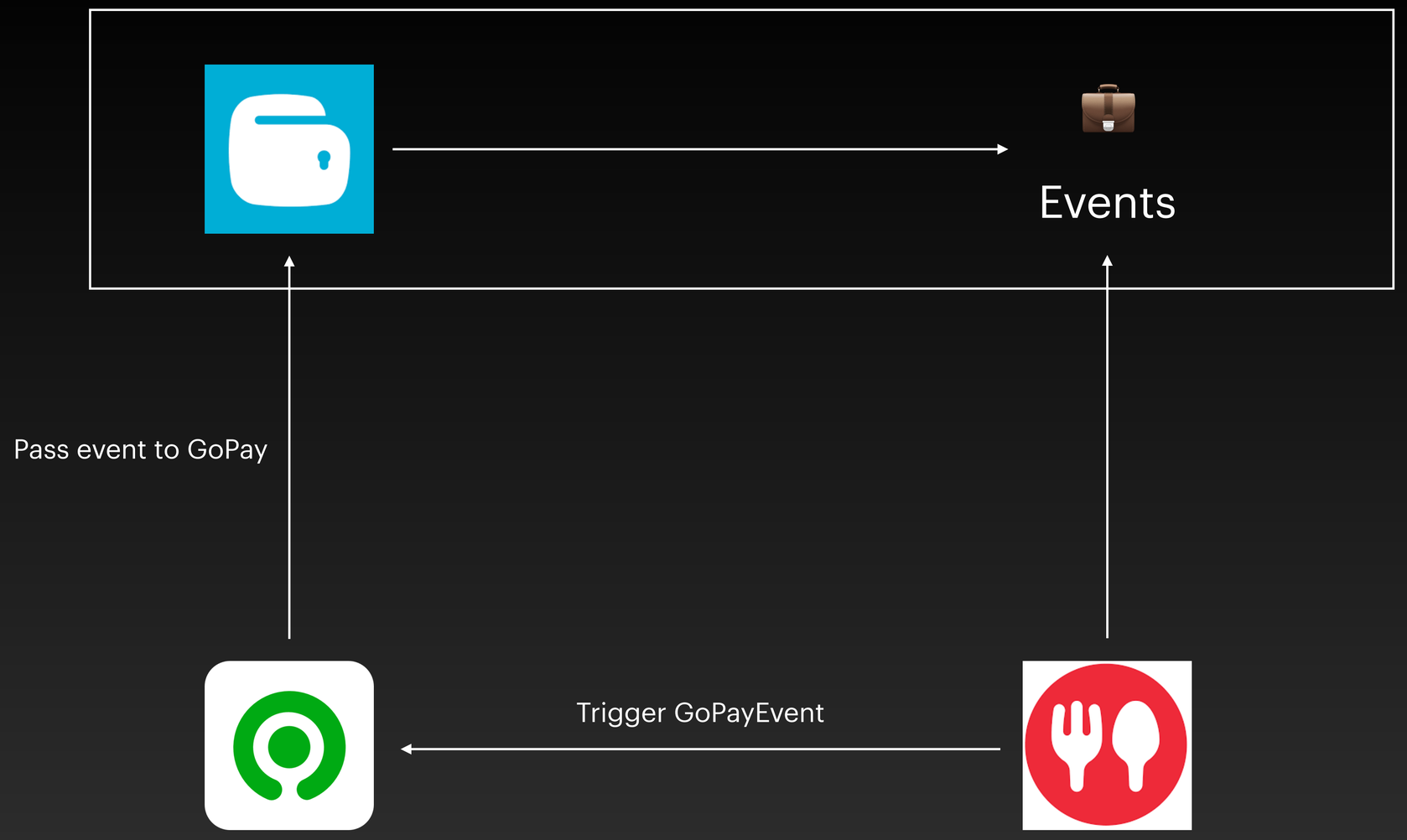


```
public class GoPayInterface: Module {
    let enum GoPayEvent: Event {
    let     case openP2P(GoPayP2PConfig, ()->Void)
                    var productId: String
    publ    var productId: { "GoPay" }RL) throws {
        }et event = eventResolver.eventFrom(deeplink: deeplinkURL)
    }           protocol Event {
        class GoPayEventResolver {: String
    publ    func eventFrom(deeplink: URL) -> GoPayEvent {

    }       }
            }
class AppDelegate: UIResponder, UIApplicationDelegate {
    private let productInstances = [GoPayInterface(), GoFoodInterface()]
    private  var products: [String: Module] = Dictionary(uniqueKeysWithValues: productInstances.map { product in
        return (product.productId, product)
    })

    func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
        // Fetch deeplink
        if deeplink.isOfGoPay {
            (products["GoPay"] as? GoPayInterface)?.handleDeeplink(url: url)
        }
        return true
    }

    func handleEvent(event: Event) {
        if event.isOfGoPay {
            (products["GoPay"] as? GoPayInterface)?.handleEvent(event: event)
        }
    }
}
```



Cyclic Dependencies




Events
Pass event to GoPay
Trigger GoPayEvent

Dependency Management

- CocoaPods
- Swift Package Manager

Collaboration and Optimisations

- Xcodegen
- Tuist

Thank you