

Managing Complex Screens with Plug-in Architecture

By Darius Sabaliauskas, Senior iOS Engineer @Vinted

Agenda

- History lesson
- Why plug-in architecture?
- What is plug-in architecture?
- Open/Closed principle
- Examples (UIKit)
- Summary

History

- Plug-in architecture == Microkernel architecture
- Unix (1970s) had small modular utilities that could be piped together
- 1980s-1990s component-based software engineering
- Internet Explorer first major browser to support extensions in 1997
- Eclipse IDE - plug-in based system (released in 2001)



Why?

- Some screens are very complex and doing different things. For example, product page, checkout, home page, etc.
- MVC, MVP, MVVM, VIP, VIPER, etc. are not enough for complex screens
- Plug-in/Microkernel architecture is one of solutions



Open/Closed principle

"software entities (classes, modules, functions, etc.) should be open for extension, but closed for modification"



Plug-in Architecture

Parts

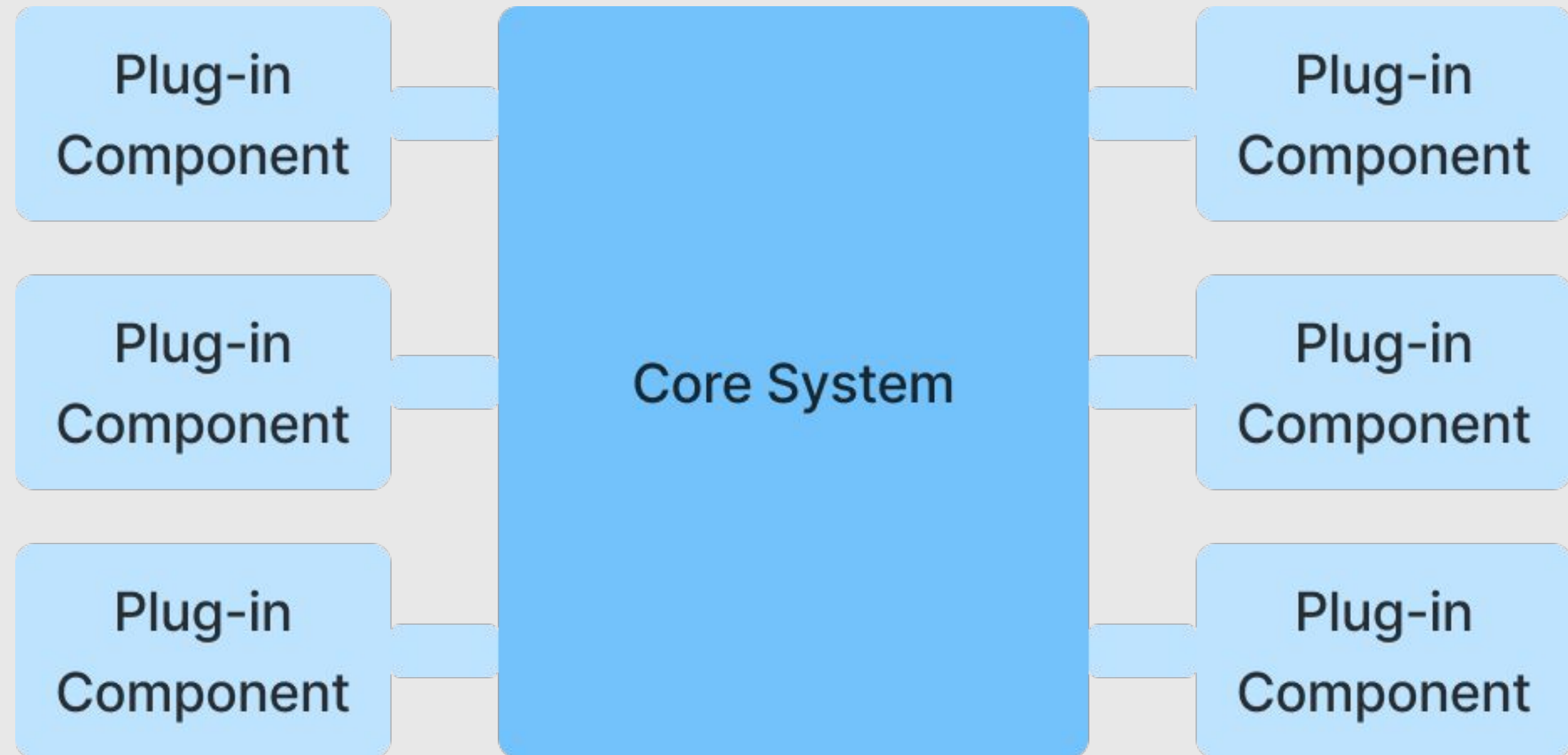
- Core system
- Plug-ins

PROS

- Makes it easy to add new functionality aka plug-in
- Plug-in can be developed and tested independently

CONS

- Core system needs to be well designed upfront
- Changing Core system is expensive



Plug-in Architecture

Parts

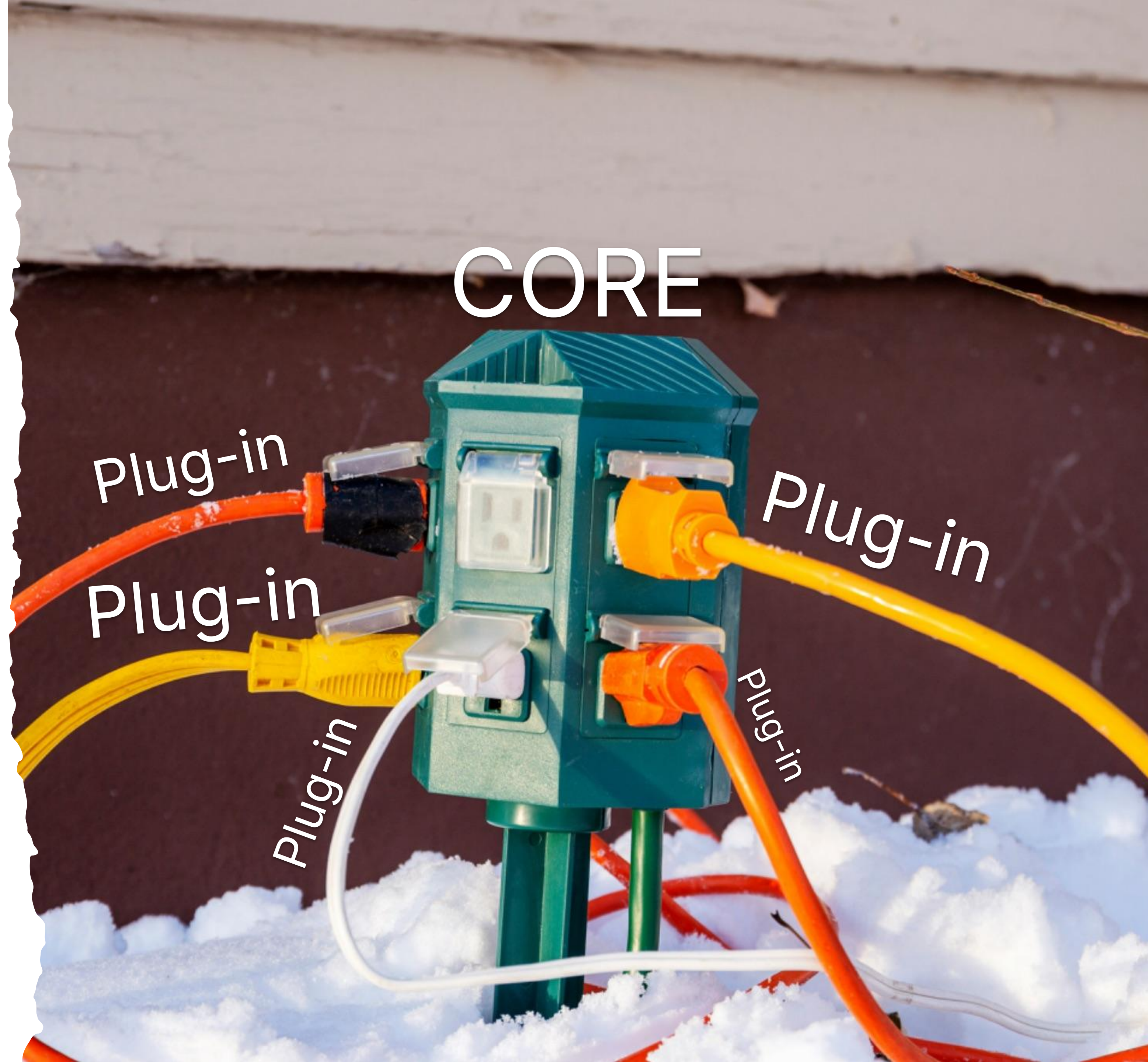
- Core system
- Plug-in

PROS

- Makes it easy to add new functionality aka plug-in
- Plug-in can be developed and tested independently






CONS

- Core system needs to be well designed upfront
- Changing Core system is expensive



Complex screen

Carrier 10:56

 Bicycle	12.00 €
 House	5.00 €
 Car	5.00 €
 Star	20.00 €
 Envelope	0.42 €


Total Price: 42.42 €

→


Complex screen


1

Carrier




10:56






Bicycle

12.00 €




House

5.00 €




Car

5.00 €



Star

20.00 €



Envelope

0.42 €

2

Total Price: 42.42 €

3

Enter Promo Code Use

4

Choose payment method →

5

PAY

Complex screen



Complex screen

1

UIViewController

UIViewController

2

3

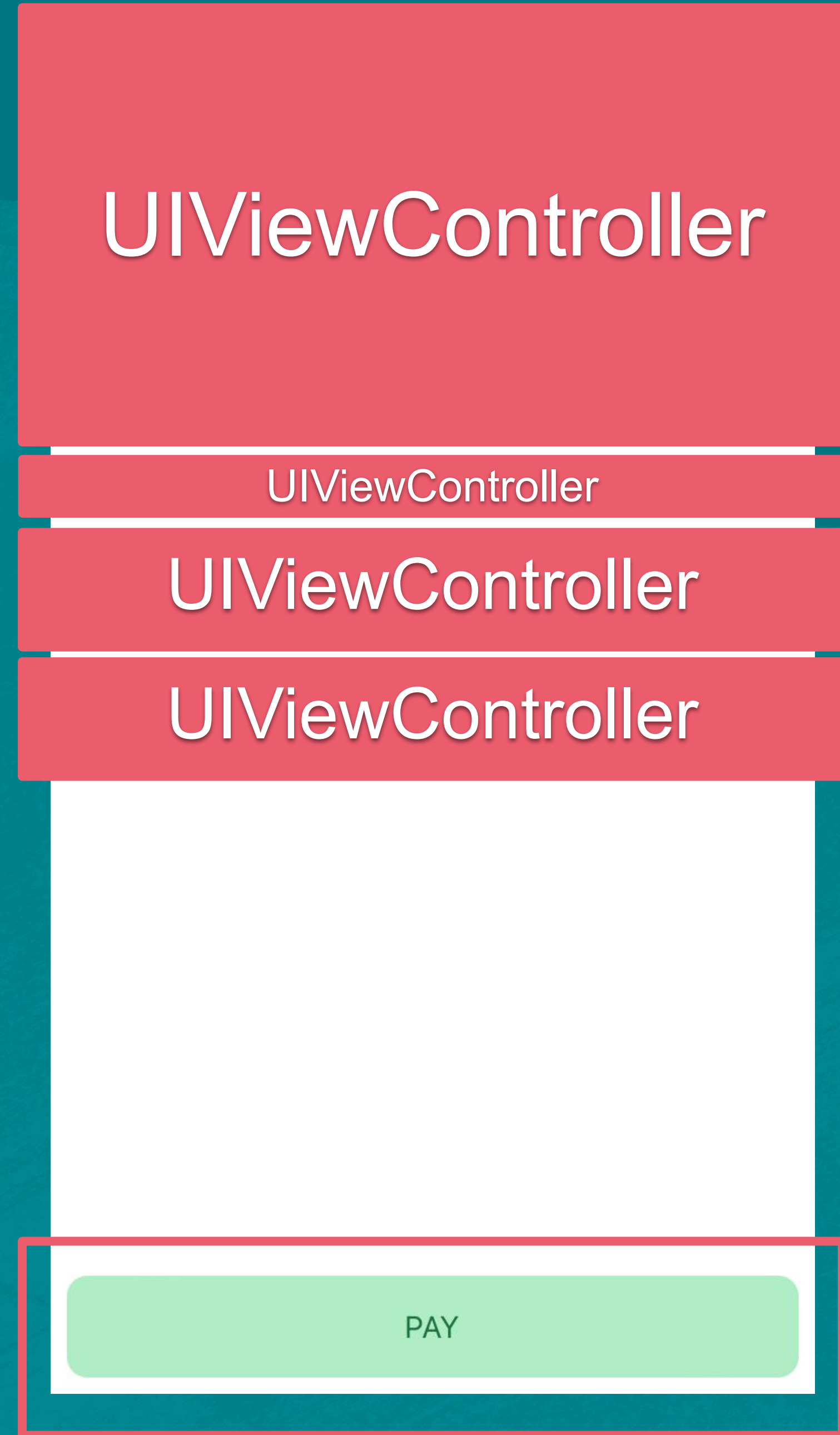
UIViewController

4

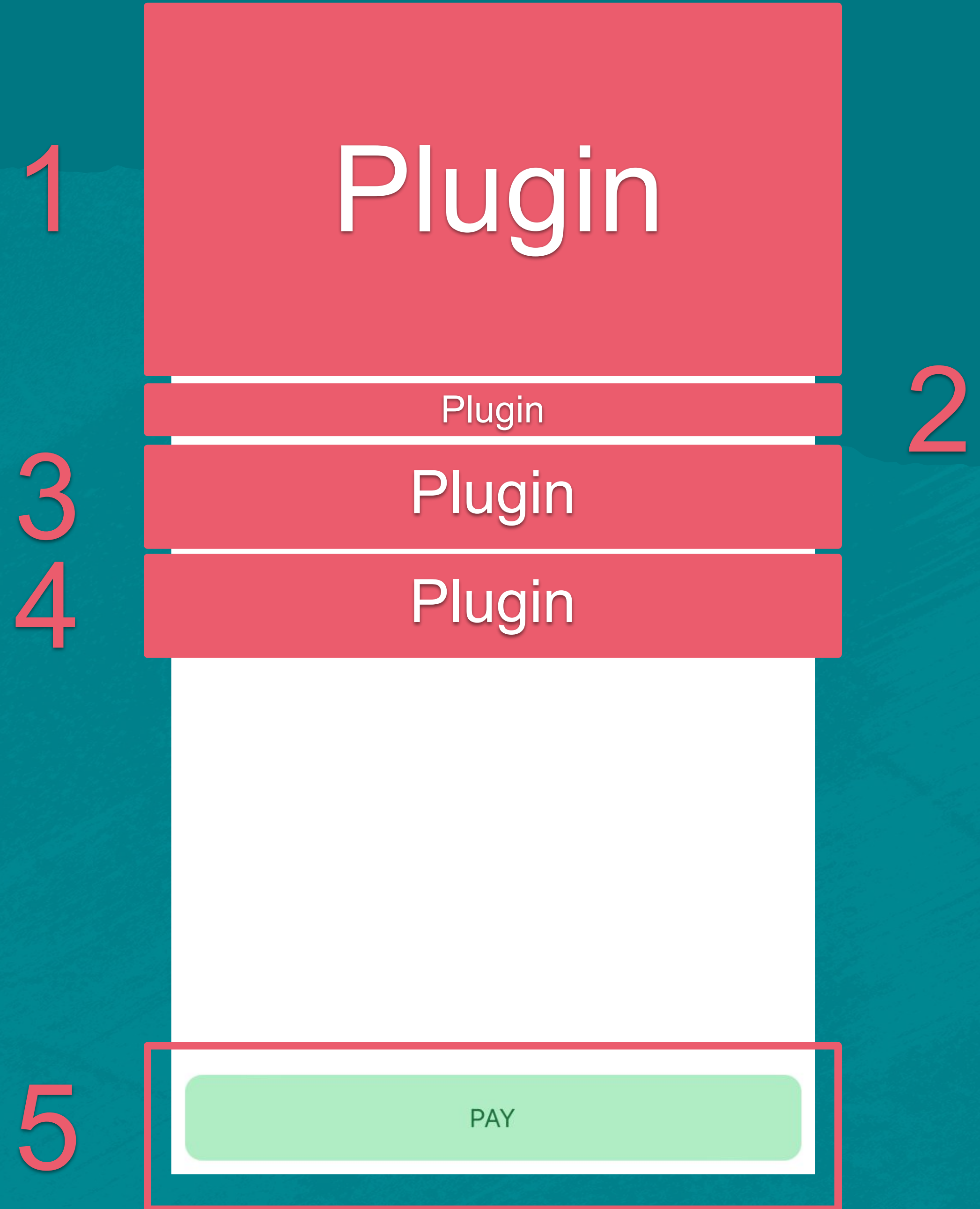
UIViewController

5

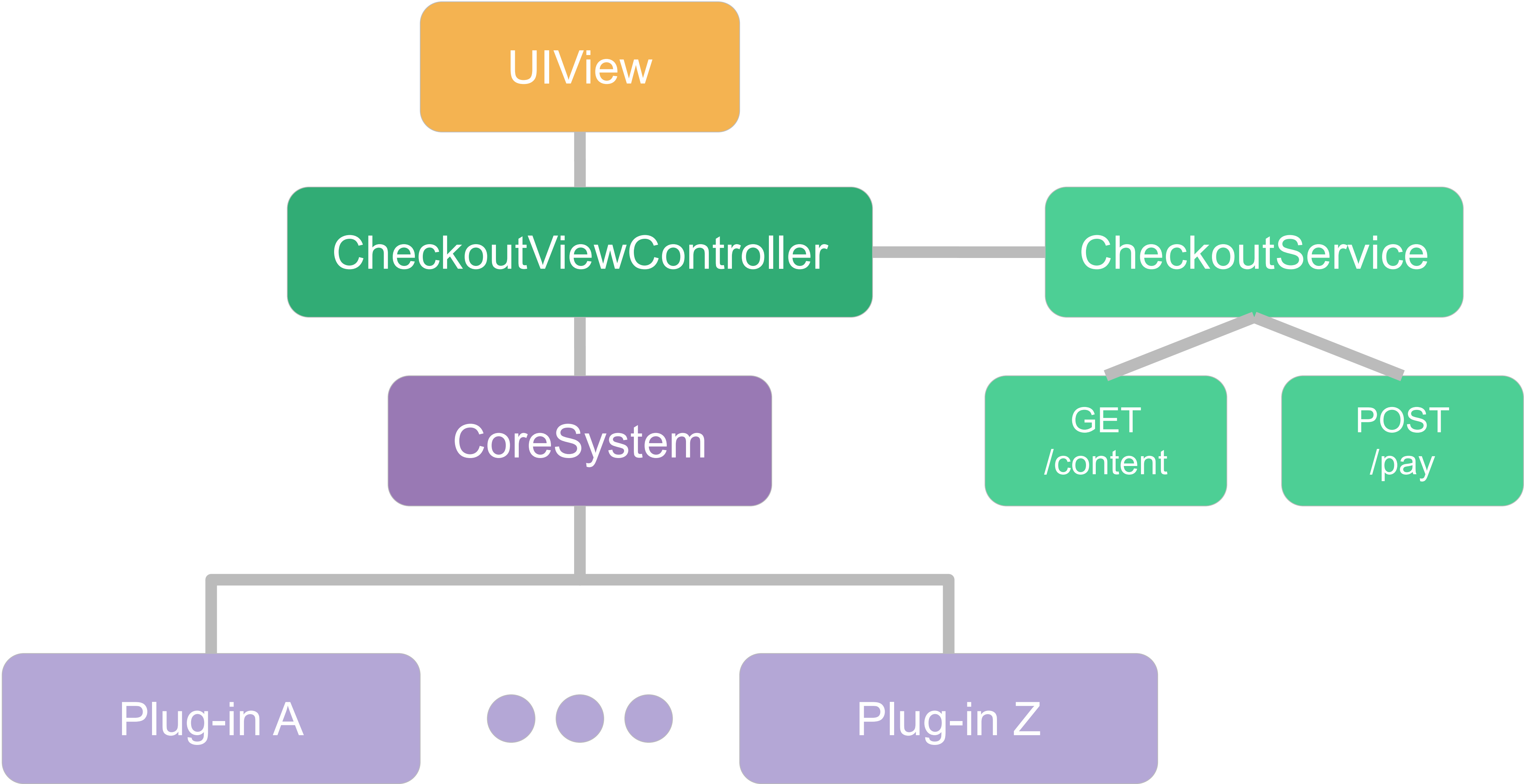
PAY



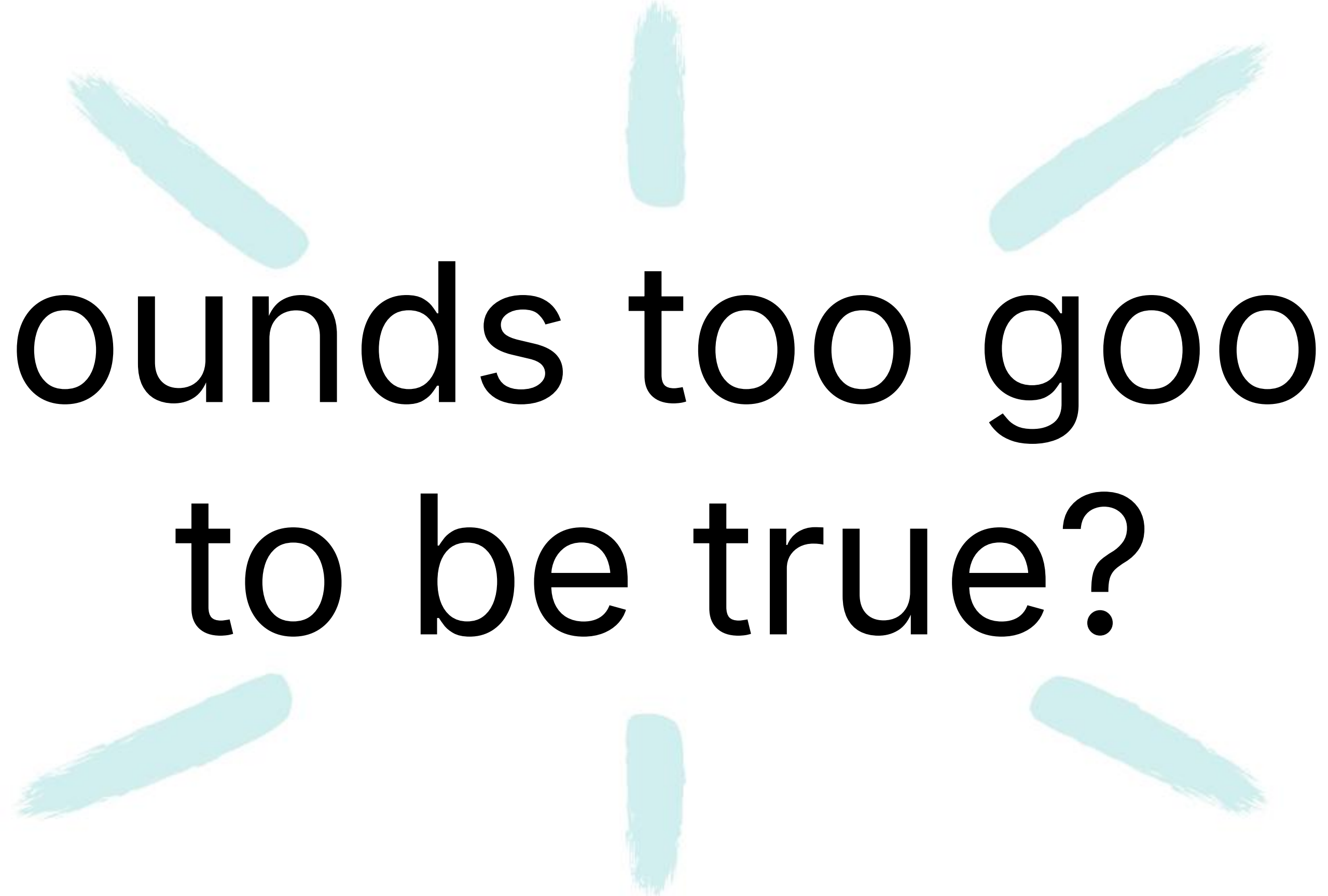
Complex screen



Structure




```
enum PluginRegistry {  
    static let entries: [String: PluginFactory.Type] = [  
        "items": ItemsPluginFactory.self,  
        "total_price": TotalPricePluginFactory.self,  
        "promo_code": PromoCodePluginFactory.self,  
        "payment_method": PaymentMethodPluginFactory.self,  
    ]  
}
```

Sounds too good
to be true?

Response problem

- V1
 - Do request in plug-in itself
- V2
 - Change decoding. `ResponseData` → `Dictionary` → `Array<[String: Data]>`


```
protocol PluginFactory {  
    init()  
    func make(from data: Data) throws -> Plugin  
}
```

```
protocol Plugin: AnyObject {  
    var controller: UIViewController { get }  
}
```

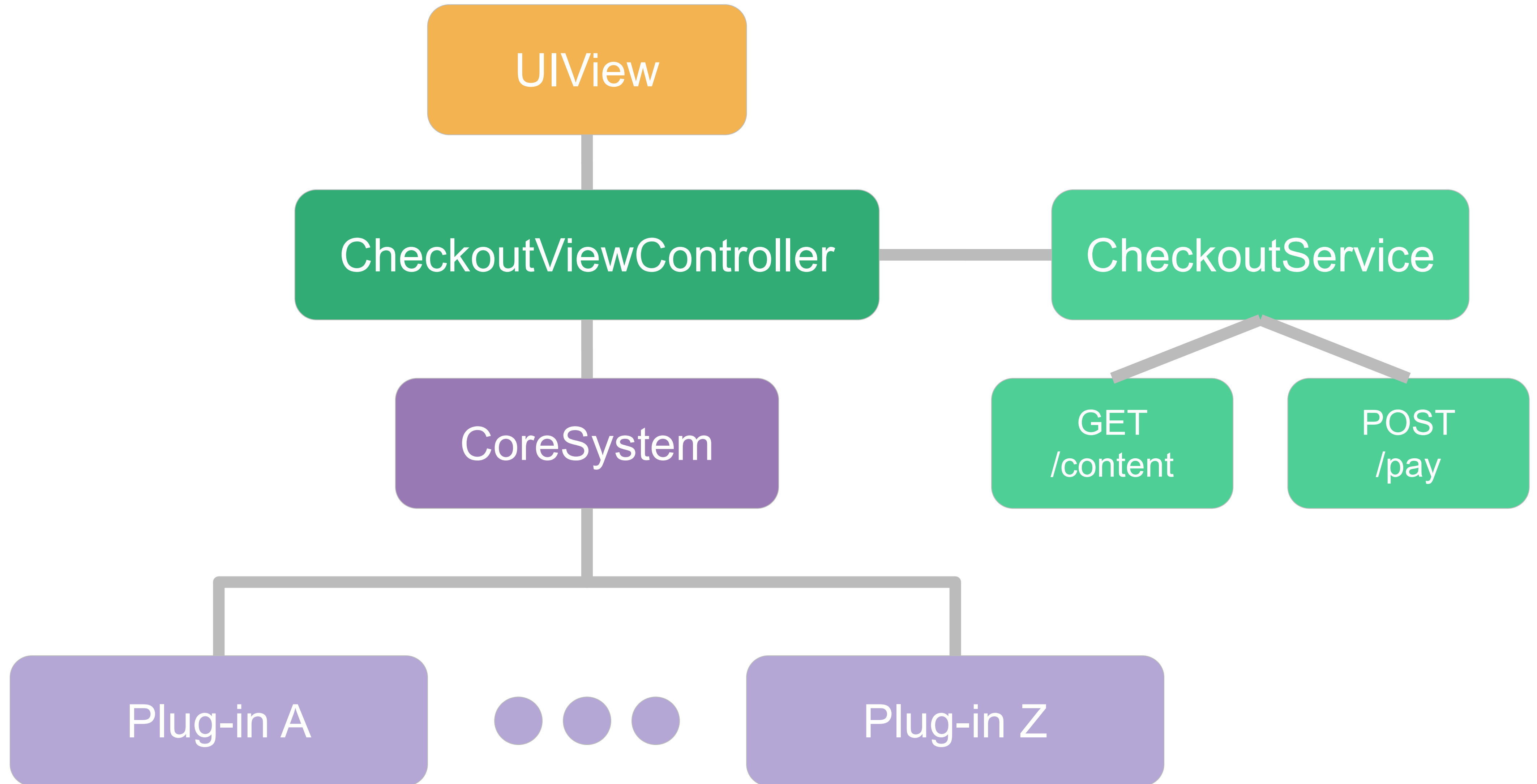


```
protocol PluginFactory {  
    init()  
    func make(from data: Data) throws -> Plugin  
}
```

```
{  
    "plugins": [  
        {  
            "type": "foo",  
            "content": { "a": "1", "...": "..." }  
        }  
    ]  
}
```

```
struct TotalPricePluginFactory: PluginFactory {  
    func make(from data: Data) throws -> Plugin {  
        let decoder = JSONDecoder()  
        let totalPrice = try decoder.decode(TotalPrice.self, from: data)  
        return TotalPricePlugin(totalPrice: totalPrice)  
    }  
}
```


Structure





Solved?


```
final class CheckoutCoreSystem {  
    var layout: [UIViewController] {}  
    var onRefresh: ((Encodable) -> Void)? {}  
  
    init(  
        registry: [String: PluginFactory.Type] = PluginRegistry.entries,  
        items: [(identifier: String, data: Data)]  
    )  
  
    func validate() throws -> [PluginOutput] {}  
}
```


The word "Validation" is centered on a white background. It is surrounded by six teal-colored brushstroke accents. Three strokes are positioned above the word, and three are below it. The strokes are arranged in a circular pattern, with one stroke at the top, one at the bottom, and two on each side, all pointing towards the center.

Validation


```
protocol PluginValidatable {  
    func validate() throws -> PluginOutput  
}
```

```
enum PluginOutput {  
    case total(Double)  
    case paymentMethod(PaymentMethod)  
}
```

```
let validatablePlugins = plugins.compactMap {  
    $0 as? PluginValidatable  
}
```




Communication

```
protocol PluginEventPublishing: AnyObject {  
    var onPublish: ((PluginEvent) -> Void)? { get set }  
}
```

```
protocol PluginEventConsuming: AnyObject {  
    func onConsume(_ event: PluginEvent)  
}
```

```
enum PluginEvent {  
    case refresh([String: Encodable])  
}
```



```
private fun setupEventBindings() {  
    let publishers = plugins.compactMap { $0 as? PluginEventPublishing }  
    for publisher in publishers {  
        publisher.onPublish = { [weak self] event in  
            self?.handlePublishedEvent(event)  
        }  
    }  
}
```

```
private fun handlePublishedEvent(_ event: PluginEvent) {  
    notifyAllEventConsumingPlugins(on: event)  
    onConsume(event) // Notify self  
}
```

```
private fun notifyAllEventConsumingPlugins(on event: PluginEvent) {  
    let consumers = plugins.compactMap { $0 as? PluginEventConsuming }  
    consumers.forEach { $0.onConsume(event) }  
}
```

Summary

- Plug-in architecture could modularise complex screen
- Think well about core system and make it small (really small)
- Consider trade-offs
- Communication is a pain
- Remember Open/Closed principle
- Remember Interface Segregation principle

Thank you



@sabadarius



<https://www.linkedin.com/in/darius-sabaliauskas/>



<https://medium.com/@jamagas>