Developer Tools #WWDC16

Protocol and Value Oriented Programming in UIKit Apps

Swift in practice

Session 419

Jacob Xiao Protocol Oriented Programmer
Alex Migicovsky Swift Compiler Typo Engineer

Overview

Value types and protocols

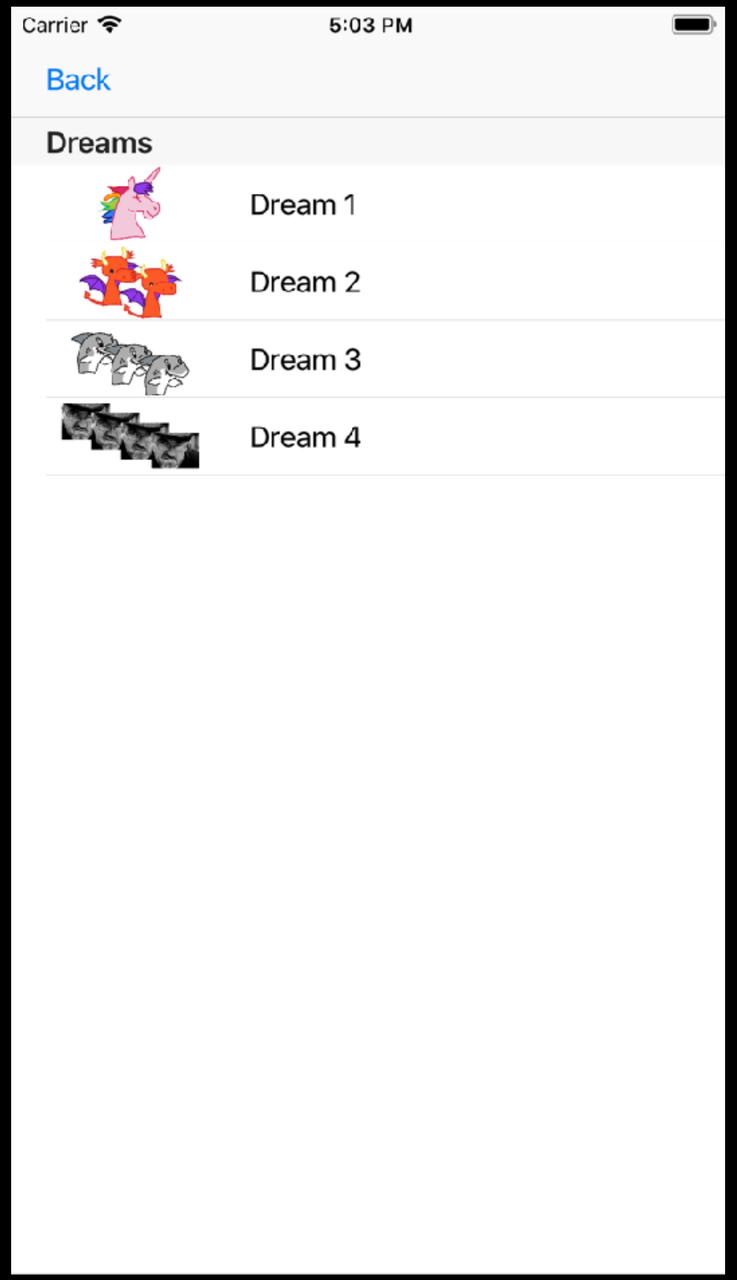
- Recap—Model
- Focus—View and controller
- Testing

Sample code: https://developer.apple.com/go/?id=lucid-dreams

View

Cell layout

Jacob Xiao Protocol Oriented Programmer


Carrier
5:03 PM
Back
Dreams
Dream 1
Dream 2
Dream 3
Dream 4


Carrier
5:03 PM
Back
Detail
Dream 1
Pink unicorn

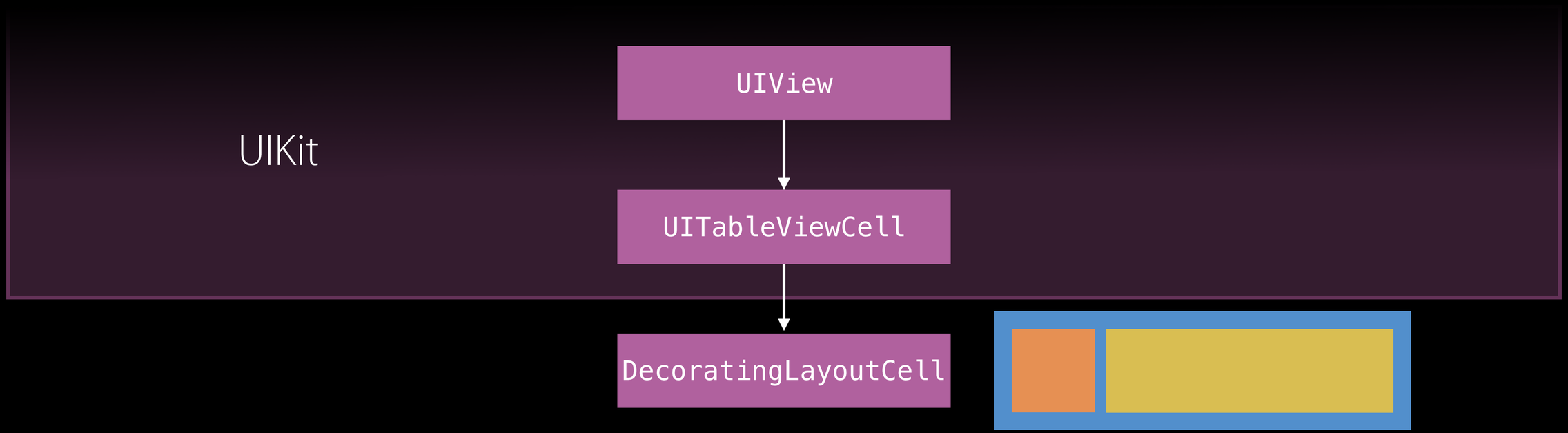
围绕以下问题展开：

DecoratingLayout

首先实现DecoratingLayout cell

- 问题1：UIView如何复用DecoratingLayout
- 问题2：如何让DecoratingLayout既可布局UIView元素，又能布局SKNode元素
- 问题3：如何控制DecoratingLayout中的元素为同种类型
- 问题4：对于左侧为多元素层叠展示的布局，如何重用DecoratingLayout

Cell Layout

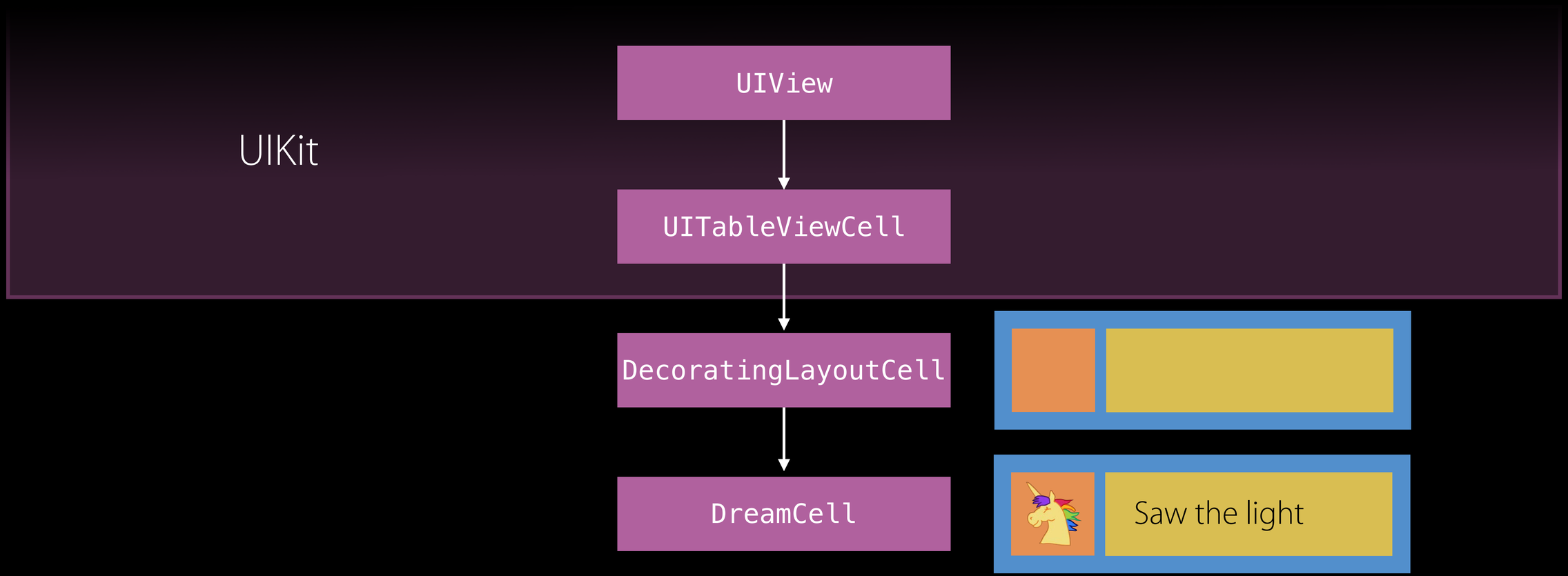


```
// Cell Layout

class DecoratingLayoutCell : UITableViewCell {
   var content: UIView
   var decoration: UIView

   // Perform layout...
}
```

Cell Layout

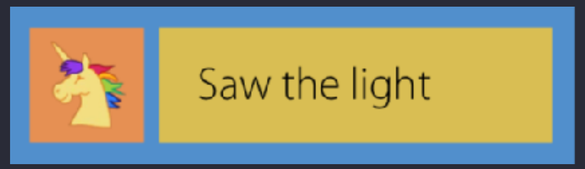

UIKit
UIView
UITableViewCell
DecoratingLayoutCell
DreamCell
Saw the light

```
// Cell Layout

class DreamCell : DecoratingLayoutCell {
    var title: UILabel
    var creature: UIImageView

    // Add title to content view
    // Add creature to decoration view
     // ……

}
```



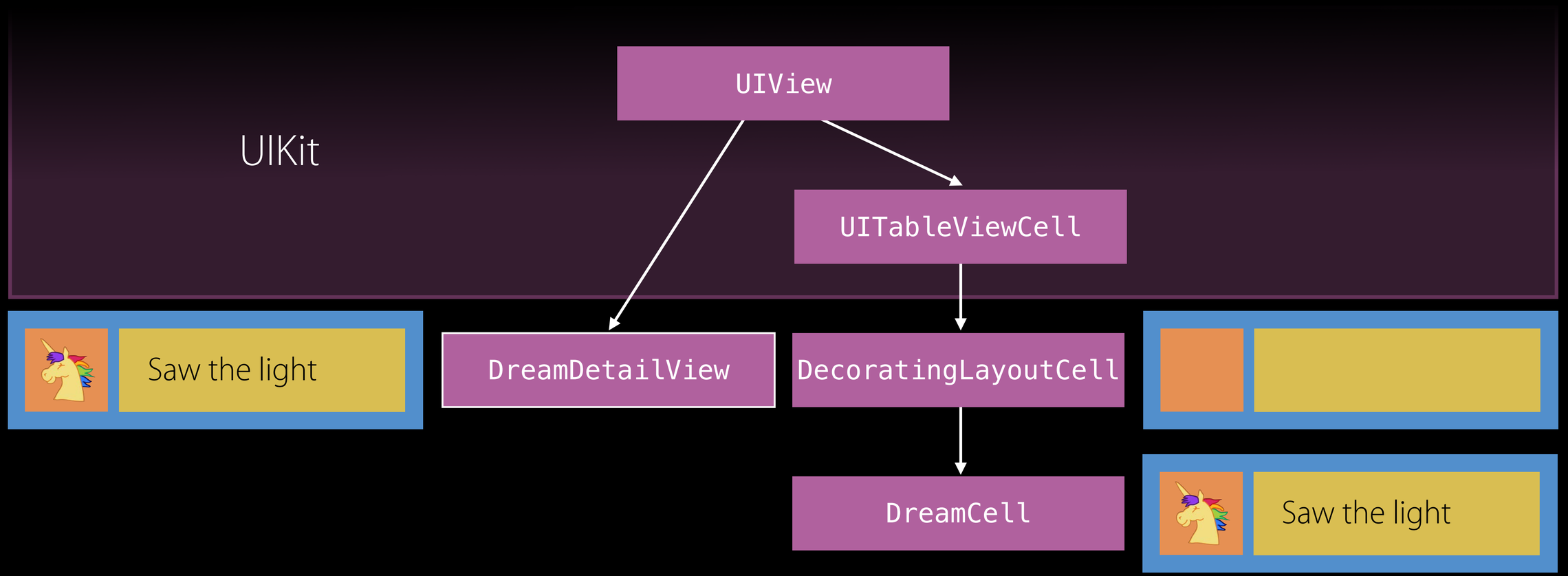
围绕以下问题展开：

DecoratingLayout

首先实现DecoratingLayout cell

- **问题1：UIView如何复用DecoratingLayout**
- 问题2：如何让DecoratingLayout既可布局UIView元素，又能布局SKNode元素
- 问题3：如何控制DecoratingLayout中的元素为同种类型
- 问题4：对于左侧为多元素层叠展示的布局，如何重用DecoratingLayout

Cell Layout


UIKit
UIView
UITableViewCell
Saw the light
DreamDetailView
DecoratingLayoutCell
DreamCell
Saw the light

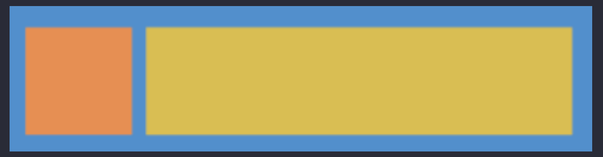
```
// View Layout

class DecoratingLayoutCell : UITableViewCell {
    var content: UIView
    var decoration: UIView

    // Perform layout...
}

struct DecoratingLayout {
    var content: UIView
    var decoration: UIView

    // Perform layout...
}
```



```
// View Layout

struct DecoratingLayout {
   var content: UIView
   var decoration: UIView

   mutating func layout(in rect: CGRect) {
      // Perform layout...
   }
}
```

```
// View Layout

class DreamCell : UITableViewCell {
    ...

    override func layoutSubviews() {
        var decoratingLayout = DecoratingLayout(content: content, decoration: decoration)
        decoratingLayout.layout(in: bounds)
    }
}


class DreamDetailView : UIView {
    ...

    override func layoutSubviews() {
        var decoratingLayout = DecoratingLayout(content: content, decoration: decoration)
        decoratingLayout.layout(in: bounds)
    }
}
```

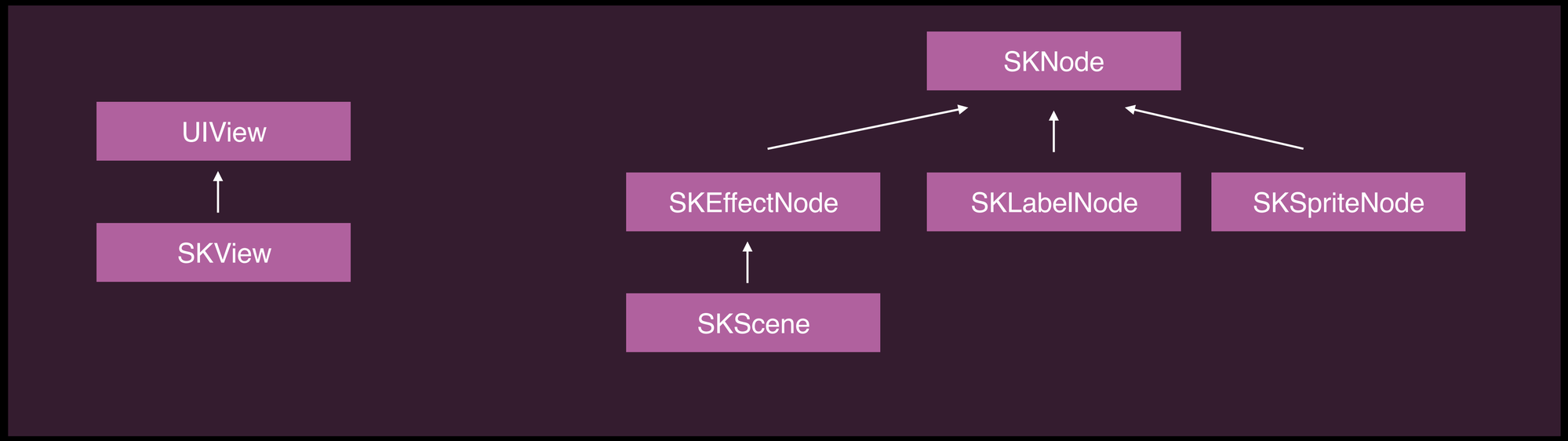
围绕以下问题展开：

DecoratingLayout

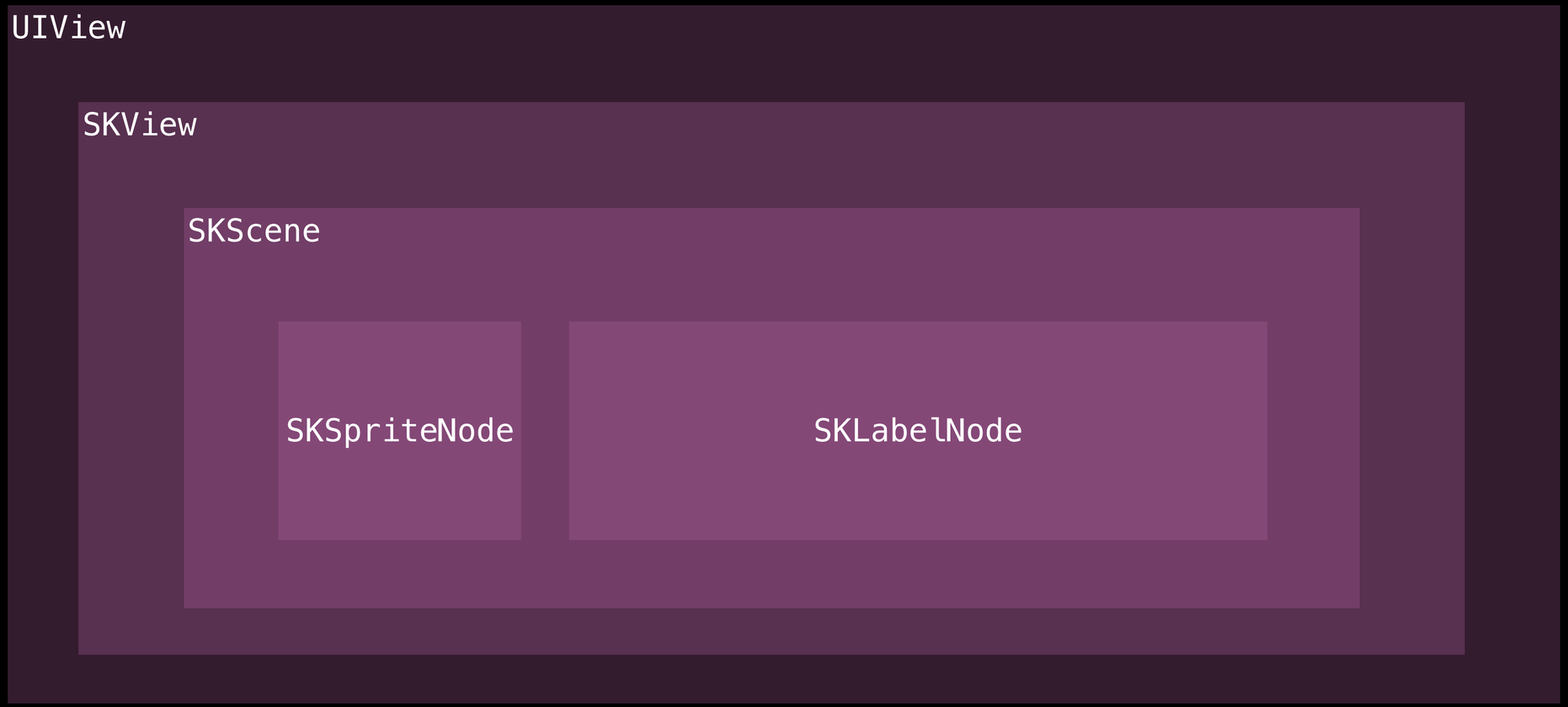
首先实现DecoratingLayout cell

- 问题1：UIView如何复用DecoratingLayout
- **问题2：如何让DecoratingLayout既可布局UIView元素，又能布局SKNode元素**
- 问题3：如何控制DecoratingLayout中的元素为同种类型
- 问题4：对于左侧为多元素层叠展示的布局，如何重用DecoratingLayout

Sprite Kit


UIView
SKView
SKNode
SKEffectNode
SKLabelNode
SKSpriteNode
SKScene

Sprite Kit



```swift
// View Layout

struct DecoratingLayout {
    var content: UIView
    var decoration: UIView
    mutating func layout(in rect: CGRect) {
        content.frame = ...
        decoration.frame = ...
    }
}
```

```
// SpriteKit Layout

struct ViewDecoratingLayout {
    var content: UIView
    var decoration: UIView
    mutating func layout(in rect: CGRect) {
        content.frame = ...
        decoration.frame = ...
    }
}
```

```
struct NodeDecoratingLayout {
    var content: SKNode
    var decoration: SKNode
    mutating func layout(in rect: CGRect) {
        content.frame = ...
        decoration.frame = ...
    }
}
```

```
// Layout

struct DecoratingLayout {
   var content:
   var decoration:
   mutating func layout(in rect: CGRect) {
      content.frame = ...
      decoration.frame = ...
   }
}
```

```
// Layout

struct DecoratingLayout {
   var content:
   var decoration:
   mutating func layout(in rect: CGRect) {
      content.frame = ...
      decoration.frame = ...
   }
}
```

```
// Layout
```

```
struct DecoratingLayout {
    var content: Layout
    var decoration: Layout
    mutating func layout(in rect: CGRect) {
        content.frame = ...
        decoration.frame = ...
    }
}
```

```
protocol Layout {
    var frame: CGRect { get set }
}
```

```
// Layout
```

```
struct DecoratingLayout {
   var content: Layout
   var decoration: Layout
   mutating func layout(in rect: CGRect) {
      content.frame = ...
      decoration.frame = ...
   }
}

protocol Layout {
   var frame: CGRect { get set }
}

extension UIView : Layout {}
extension SKNode : Layout {}
```

围绕以下问题展开：

DecoratingLayout

首先实现DecoratingLayout cell

- 问题1：UIView如何复用DecoratingLayout
- 问题2：如何让DecoratingLayout既可布局UIView元素，又能布局SKNode元素
- **问题3：如何控制DecoratingLayout中的元素为同种类型**
- 问题4：对于左侧为多元素层叠展示的布局，如何重用DecoratingLayout

```
// Layout
```

```
struct DecoratingLayout {
    var content: Layout      ←————————  UIView
    var decoration: Layout   ←————————  SKNode
    mutating func layout(in rect: CGRect) {
        content.frame = ...
        decoration.frame = ...
    }
}

protocol Layout {
    var frame: CGRect { get set }
}

extension UIView : Layout {}
extension SKNode : Layout {}
```

```
// Layout
```

```
struct DecoratingLayout<Child : Layout> {
   var content: Child
   var decoration: Child
   mutating func layout(in rect: CGRect) {
      content.frame = ...
      decoration.frame = ...
   }
}

protocol Layout {
   var frame: CGRect { get set }
}

extension UIView : Layout {}
extension SKNode : Layout {}
```

```
// Layout
```

```
struct DecoratingLayout<Child : Layout> {
    var content: Child
    var decoration: Child
    mutating func layout(in rect: CGRect) {
        content.frame = ...
        decoration.frame = ...
    }
}
```

Must be the same

```
protocol Layout {
    var frame: CGRect { get set }
}

extension UIView : Layout {}
extension SKNode : Layout {}
```

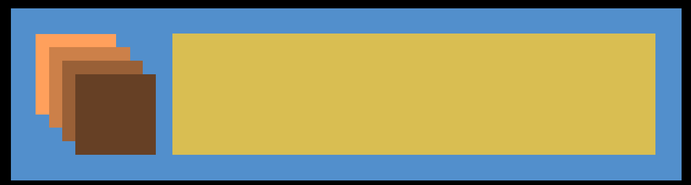
围绕以下问题展开：

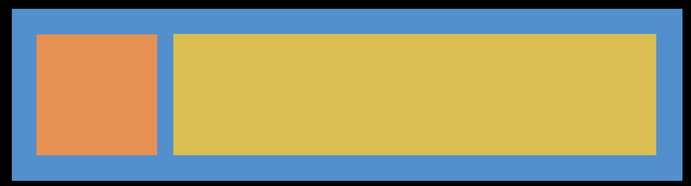
DecoratingLayout

首先实现DecoratingLayout cell

- 问题1：UIView如何复用DecoratingLayout
- 问题2：如何让DecoratingLayout既可布局UIView元素，又能布局SKNode元素
- 问题3：如何控制DecoratingLayout中的元素为同种类型
- **问题4：对于左侧为多元素层叠展示的布局，如何重用DecoratingLayout**

Composition of Views





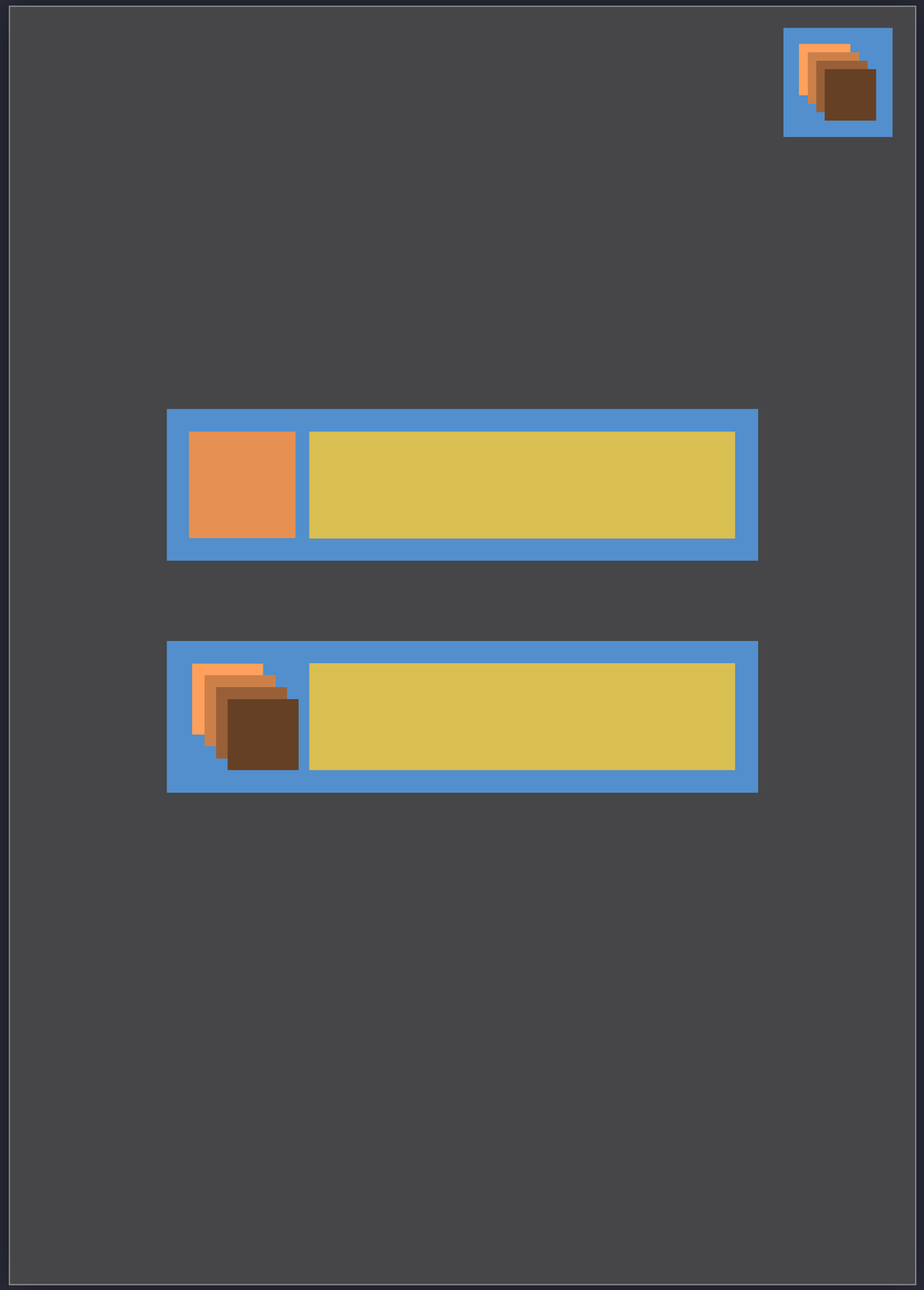
```
// Composition of Values


struct CascadingLayout<Child : Layout> {
   var children: [Child]
   mutating func layout(in rect: CGRect) {
      ...
   }
}
```

```
// Composition of Values

struct CascadingLayout<Child : Layout> {
   var children: [Child]
   mutating func layout(in rect: CGRect) {
      ...
   }
}

struct DecoratingLayout<Child : Layout> {
   var content: Child
   var decoration: Child
   mutating func layout(in rect: CGRect) {
      content.frame = ...
      decoration.frame = ...
   }
}
```



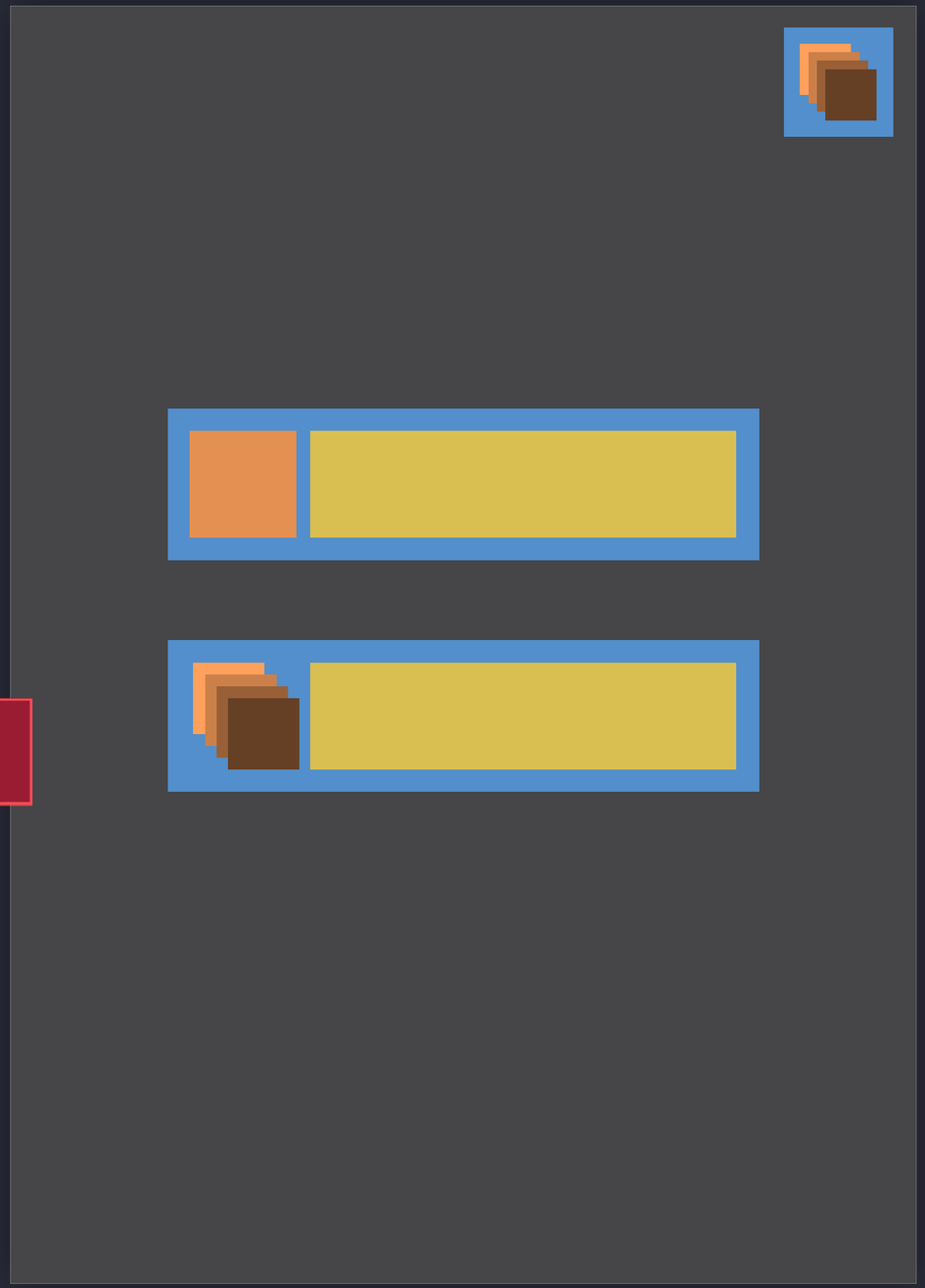
```
// Composition of Values


struct CascadingLayout<Child : Layout> {
   var children: [Child]
   mutating func layout(in rect: CGRect) {
      ...
   }
}


struct DecoratingLayout<Child : Layout> {
   var content: Child
   var decoration: Child

   mutating func layout(in rect: CGRect) {
      content.frame = ...
      decoration.frame = …
   }
}
```

Child must conform to Layout protocol



```
// Layout

struct DecoratingLayout {
    var content: Layout
    var decoration: Layout

    mutating func layout(in rect: CGRect) {
        content.frame = ...
        decoration.frame = ...
    }
}



protocol Layout {
    var frame: CGRect { get set }
}
```

```
// Composition of Values
```

```
protocol Layout {
    var frame: CGRect { get set }
}
```

```
protocol Layout {
    mutating func layout(in rect: CGRect)
}
```

```
// Composition of Values

protocol Layout {
   mutating func layout(in rect: CGRect)
}

extension UIView : Layout { ... }
extension SKNode : Layout { ... }

struct DecoratingLayout<Child : Layout> : Layout { ... }
struct CascadingLayout<Child : Layout> : Layout { ... }
```

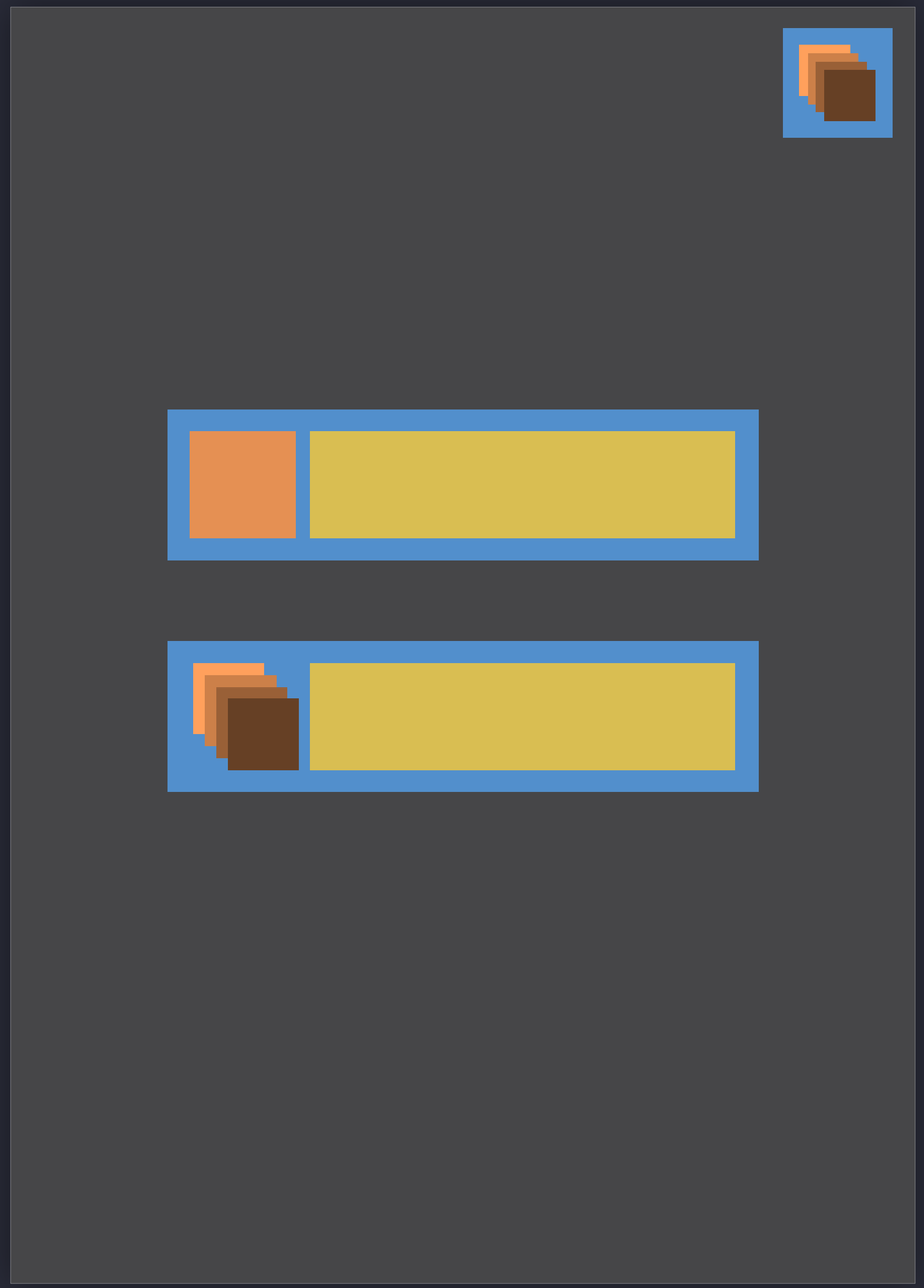
```
// Composition of Values


struct CascadingLayout<Child : Layout> : Layout {
   var children: [Child]
   mutating func layout(in rect: CGRect) {
      ...
   }
}


struct DecoratingLayout<Child : Layout> : Layout {
   var content: Child
   var decoration: Child

   mutating func layout(in rect: CGRect) {
      content.layout(rect)
      decoration.layout(rect)
   }
}
```

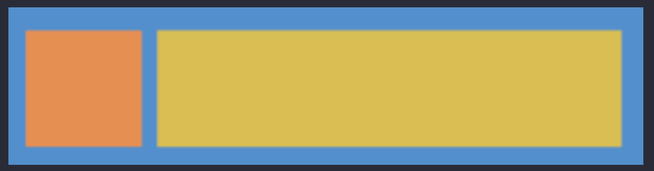
the type of content must consistent with decoration

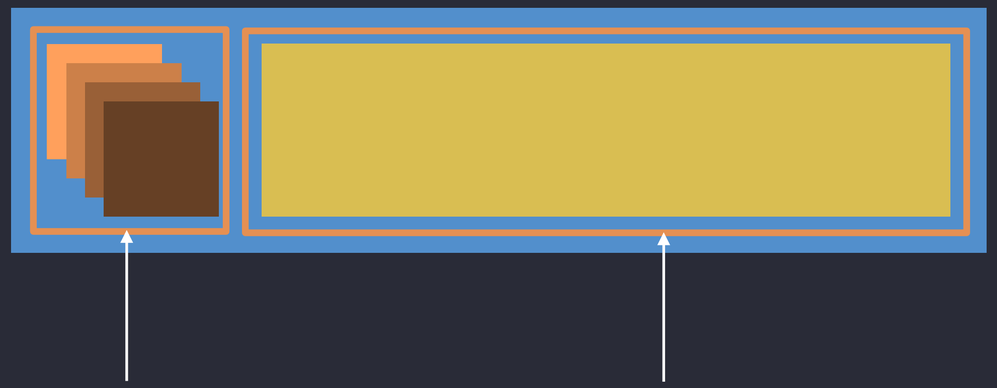


```
// Composition of Values

struct DecoratingLayout<Child : Layout> : Layout {
   var content: Child
   var decoration: Child

   mutating func layout(in rect: CGRect) {
      content.layout(rect)
      decoration.layout(rect)
   }
}
```





```
// Contents


protocol Layout {
   mutating func layout(in rect: CGRect)
   var contents: [????] { get }
}
```

```swift
// Contents

protocol Layout {
    mutating func layout(in rect: CGRect)

    var contents: [Layout] { get }
}
```

UIView and SKNode

```
// Contents


protocol Layout {
    mutating func layout(in rect: CGRect)
    var contents: [Layout] { get }  <-------- UIView and SKNode
}
```

```swift
// Associated Type

protocol Layout {
    mutating func layout(in rect: CGRect)
    associatedtype Content
    var contents: [Content] { get }
}
```

```
// Associated Type

struct DecoratingLayout<Child : Layout, Decoration : Layout
                        where Child.Content == Decoration.Content> : Layout {
   var content: Child
   var decoration: Decoration

   mutating func layout(in rect: CGRect)
   typealias Content = Child.Content
   var contents: [Content] { get }
}
```

```
// Composition

var content = UILabel()
var accessories = [UIImageView]()
let decoration = CascadingLayout(children: accessories)
var composedLayout = DecoratingLayout(content: content, decoration: decoration)
composedLayout.layout(in: rect)
```

Related Sessions

Understanding Swift Performance	Mission	Friday 11:00AM
Protocol-Oriented Programming in Swift		WWDC 2015
Building Better Apps with Value Types in Swift		WWDC 2015

More Information

https://developer.apple.com/wwdc16/419

WWDC16