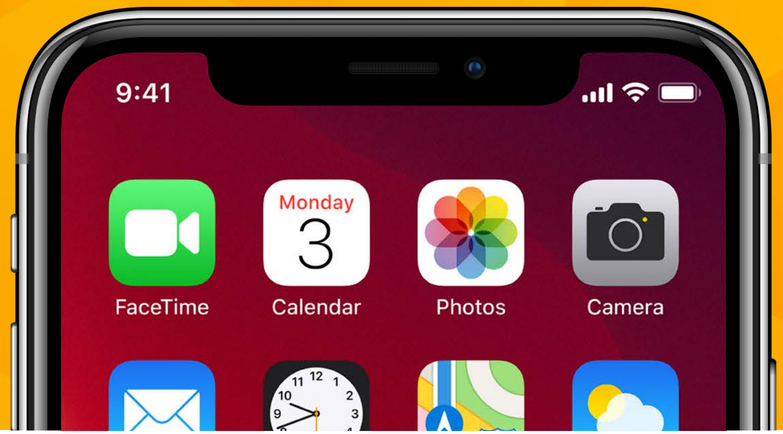


iOS Application Development

Lecture 5: Swift Protocols and Extensions

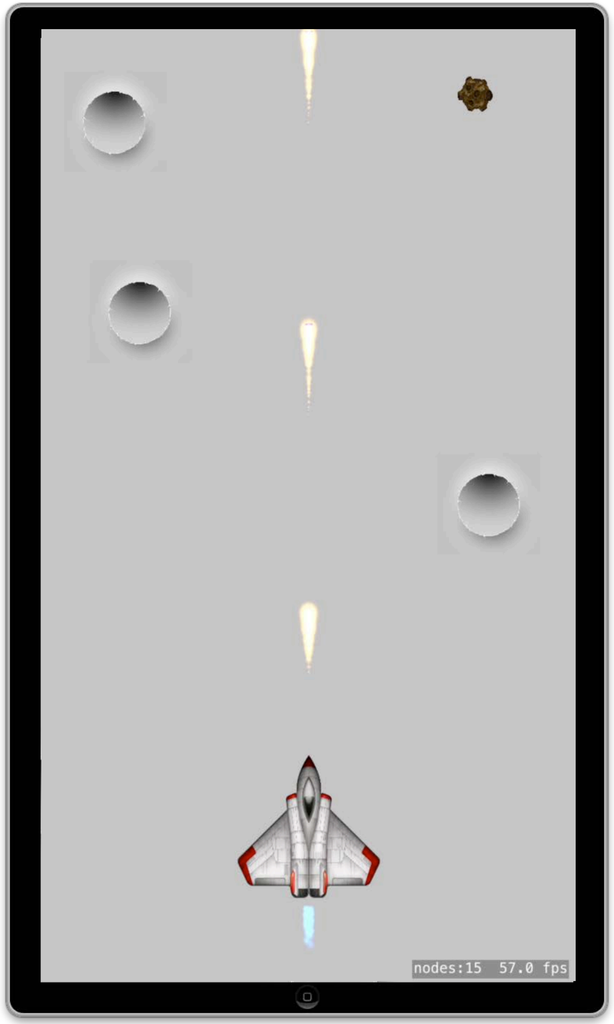
Simon Völker & Philipp Wacker
Media Computing Group
RWTH Aachen University

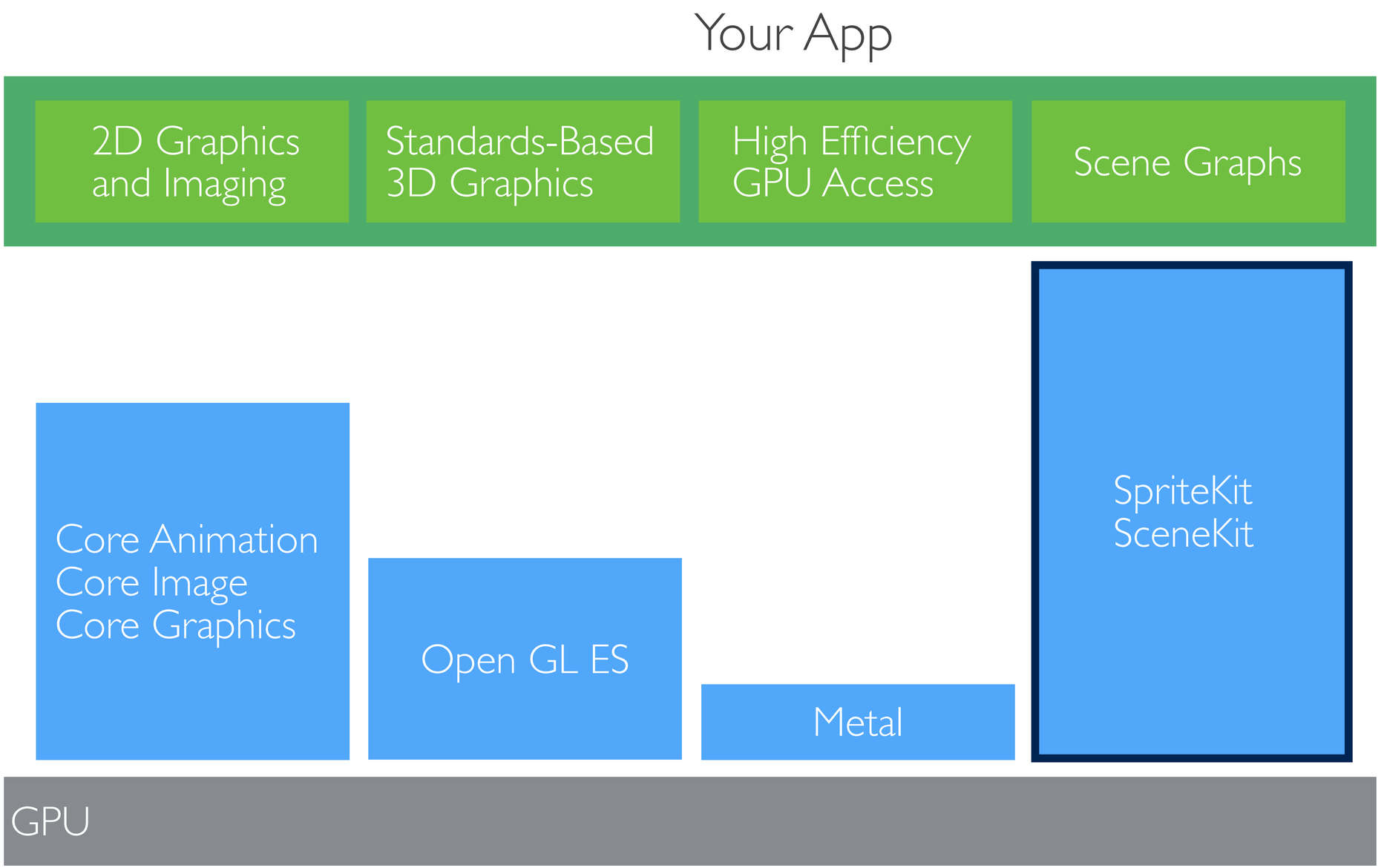
hci.rwth-aachen.de/ios

RWTH AACHEN UNIVERSITY

SpriteKit

What is Sprite Kit?

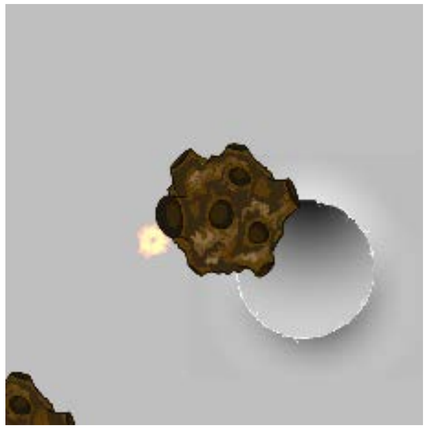




Sprite Kit

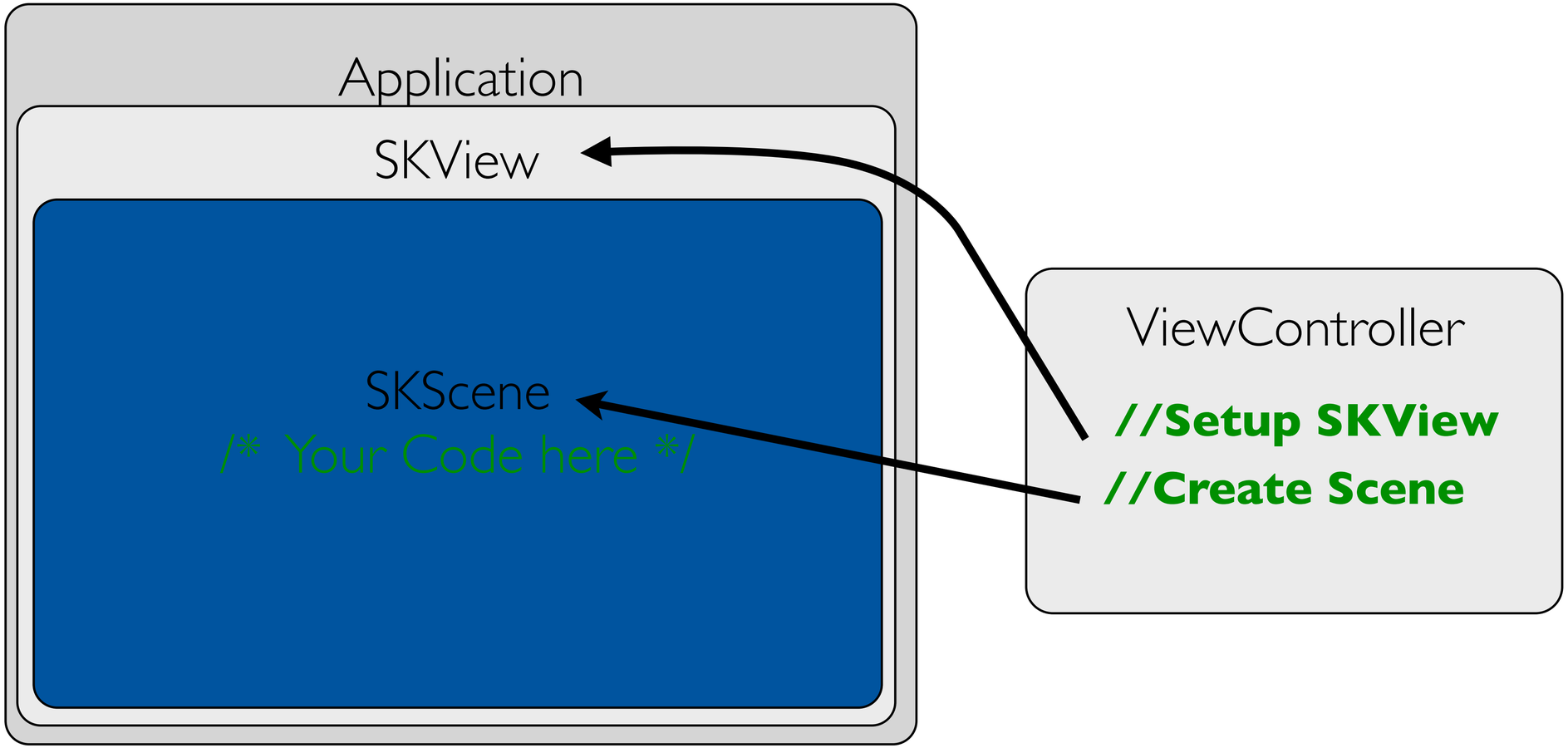
Objects

Actions

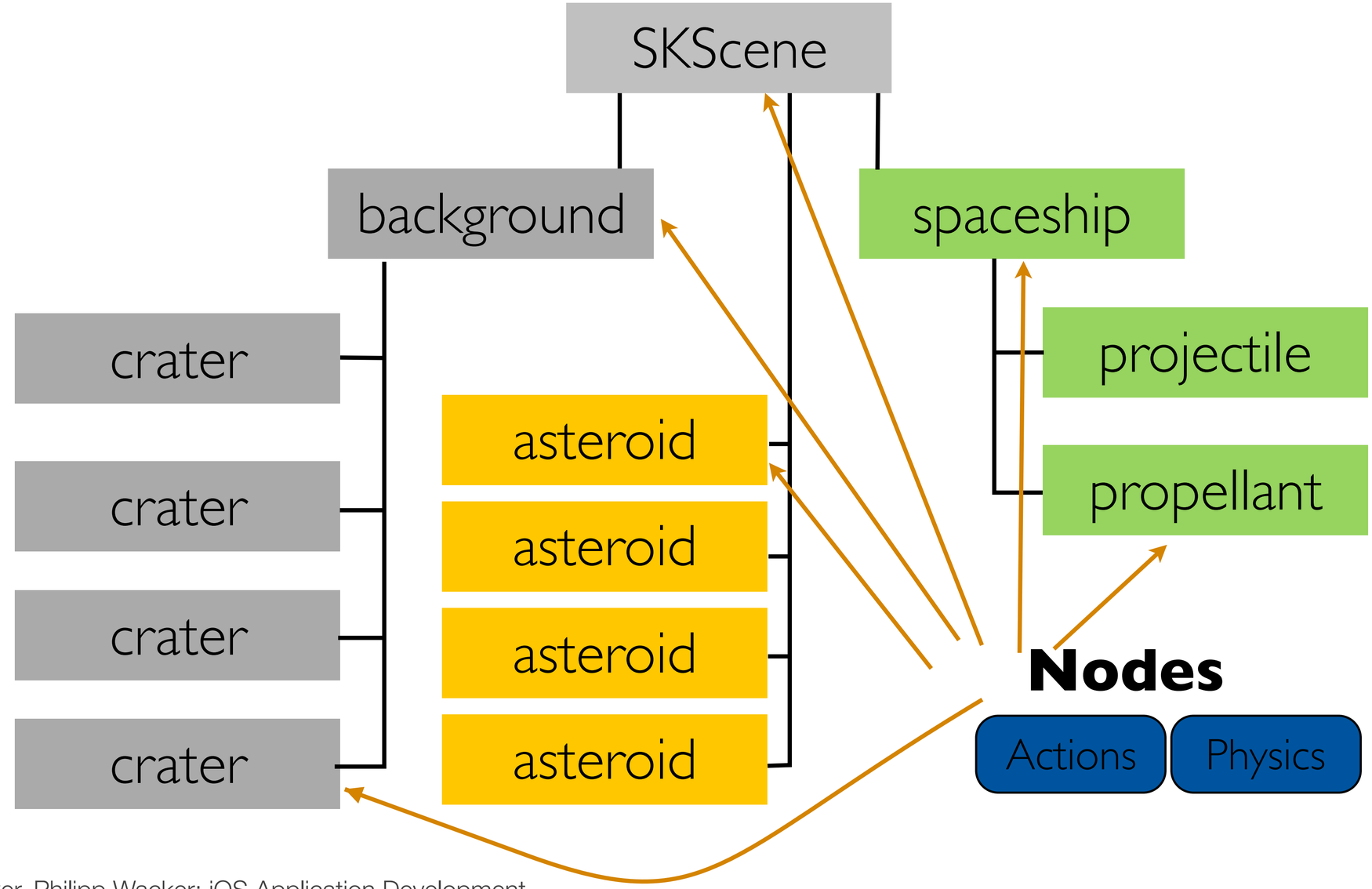


Physics

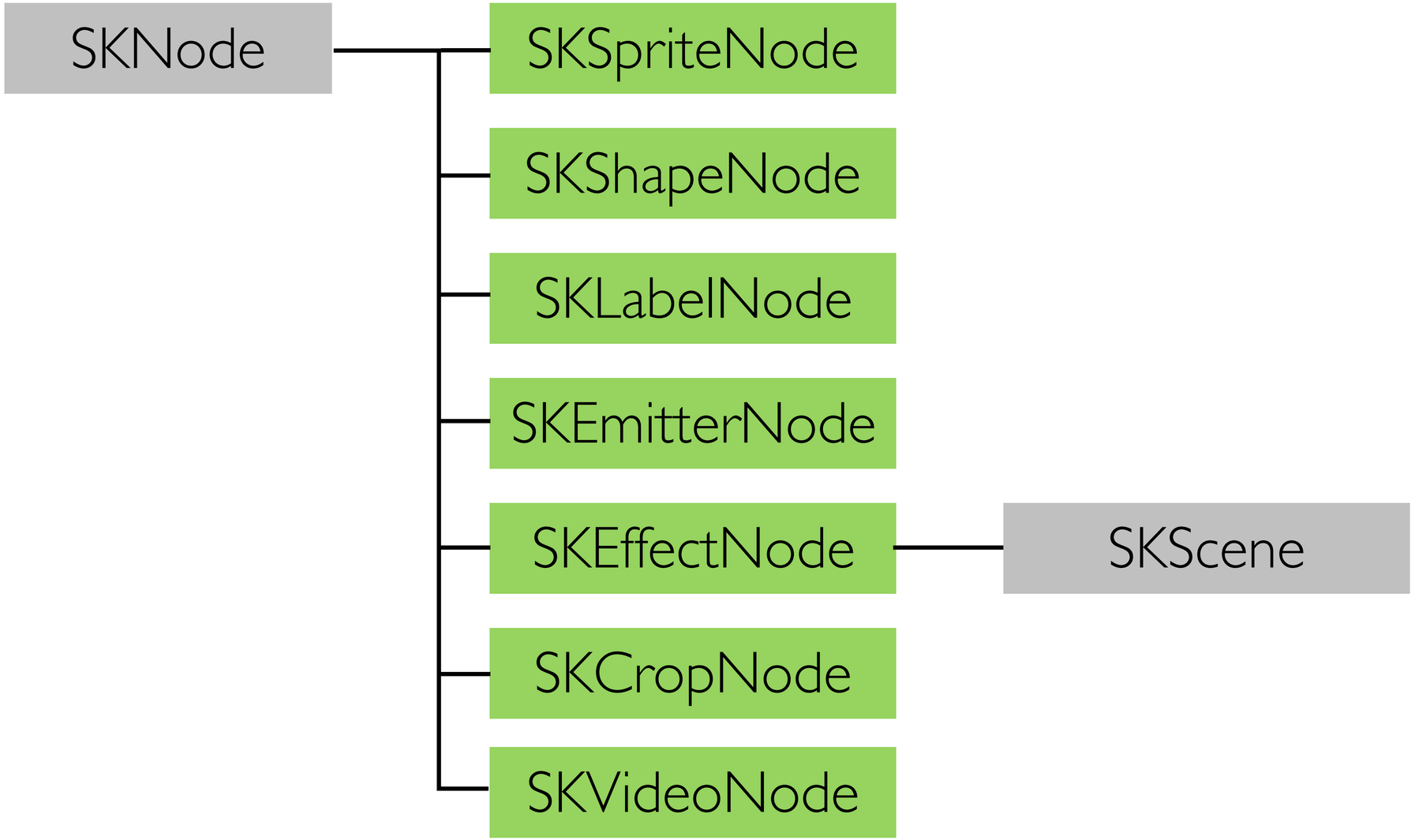
Root Object: SKScene



Scene Graph



Sprite Kit Nodes



Sprite Kit Demo

Swift Protocols and Extensions

Protocols

- A protocols defines a blueprint of methods, properties and other requirements
- Protocols can be adopted by another type that implements these requirements
- Swift utilizes many protocols such as:
  - `CustomStringConvertible`
  - `Equatable`
  - `Comparable`

CustomStringConvertible

```
let string = "Hello, world!"
print(string) // Output: Hello, world!

let number = 42
print(number) // Output: 42

let boolean = false
print(boolean) // Output: false
```

CustomStringConvertible

```
class Shoe {
    let color: String
    let size: Int
    let hasLaces: Bool

    init(color: String, size: Int,
         hasLaces: Bool) {
        self.color = color
        self.size = size
        self.hasLaces = hasLaces
    }
}
```

```
let myShoe = Shoe(color: "Black", size: 12,
hasLaces: true)
let yourShoe = Shoe(color: "Red", size: 8,
hasLaces: false)


print(myShoe)
// Output: Shoe
print(yourShoe)
// Output: Shoe
```

CustomStringConvertible

```
class Shoe : CustomStringConvertible {
    let color: String
    let size: Int
    let hasLaces: Bool

    init(color: String, size: Int,
         hasLaces: Bool) {
        self.color = color
        self.size = size
        self.hasLaces = hasLaces
    }
var description: String {
       return "Shoe(color: \(color),
     size: \(size), hasLaces: \(hasLaces))"
   }
}
```

```
let myShoe = Shoe(color: "Black", size: 12,
hasLaces: true)
let yourShoe = Shoe(color: "Red", size: 8,
hasLaces: false)


print(myShoe)
// Output:
Shoe(color: Black, size: 12, hasLaces: true)
print(yourShoe)
// Output:
Shoe(color: Red, size: 8, hasLaces: false)
```

Equatable

```
struct Employee {
    var firstName: String
    var lastName: String
    var jobTitle: String
    var phoneNumber: String
}
```

```
let employeeA = Employee(...)

let employeeB = Employee(...)

if employeeB == employeeA {
    // Do Something
}
else {
    // Do Something else

}
```

Equatable

```
struct Employee : Equatable {
    var firstName: String
    var lastName: String
    var jobTitle: String
    var phoneNumber: String

    static func ==(lhs: Employee,
                   rhs: Employee) -> Bool {
   return lhs.firstName == rhs.firstName &&
  lhs.lastName == rhs.lastName &&
  lhs.companyID == rhs.companyID
 }
}
```

```
let employeeA = Employee(...)

let employeeB = Employee(...)

if employeeB == employeeA {
    // Do Something
}
else {
    // Do Something else

}
```

Comparable

```
struct Employee {
    var firstName: String
    var lastName: String
    var jobTitle: String
    var phoneNumber: String

    static func ==(lhs: Employee,
                   rhs: Employee) -> Bool {
     return lhs.firstName == rhs.firstName &&
   lhs.lastName == rhs.lastName &&
   lhs.companyID == rhs.companyID
  }
}
```

```
let employees = [employee1, employee2,
employee3, employee4, employee5]

let sortedEmployees = employees.sorted(by: <)
```

Comparable

```
struct Employee : Comparable {
    var firstName: String
    var lastName: String
    var jobTitle: String
    var phoneNumber: String

    static func ==(lhs: Employee,
                   rhs: Employee) -> Bool {
     return lhs.firstName == rhs.firstName &&
      lhs.lastName == rhs.lastName &&
      lhs.companyID == rhs.companyID
   }

    static func < (lhs: Employee,
                   rhs: Employee) -> Bool {
     return lhs.lastName < rhs.lastName

}
```

```
let employees = [employee1, employee2,
employee3, employee4, employee5]

let sortedEmployees = employees.sorted(by: <)
```

Creating a Protocol

```
protocol FullyNamed {
    var fullName: String { get }

    func sayFullName()
}
```

Creating a Protocol

```
protocol FullyNamed {
    var fullName: String { get }

    func sayFullName()
}

struct Person: FullyNamed {
    var firstName: String
    var lastName: String

    var fullName: String {
        return "\(firstName) \(lastName)"
    }

    func sayFullName() {
        print(fullName)
    }
}
```

Optional Protocol Requirements

- Functions or parameters with `optional` modifier don't have to be implemented

- `@objc` Protocols can only be adapted by Objective-C classes or subclasses of E. g. UIKit classes

```
@objc protocol CounterDataSource {
      @objc optional func increment(forCount count: Int) -> Int
      @objc optional var fixedIncrement: Int { get }
}
```

Optional Protocol Requirements

- The SKPhysicsContactDelegate from SpriteKit

```
public protocol SKPhysicsContactDelegate : NSObjectProtocol {


    optional public func didBegin(_ contact: SKPhysicsContact)

    optional public func didEnd(_ contact: SKPhysicsContact)
}
```

Delegation

Delegation

- Delegation is a design pattern to hand of responsibilities to an instance of another type

- It is extensively used in UIKit and other iOS Frameworks

- SpriteKit example:

```
class GameScene: SKScene, SKPhysicsContactDelegate {
    …
    self.physicsWorld.contactDelegate = self
}
```

- TableView example:

```
class MyViewController:  UITableViewDataSource,   UITableViewDelegate {
…
self.myTableView.dataSource = self
self.myTableView.delegate = self
}
```

Extensions

Extensions

- Extensions add functionality to types that are already defined.
- You can add:
  - computed properties
  - define methods
  - provide new initializers
  - conform an existing type to a protocol
- You cannot add properties!

```
extension SomeType {
    // new functionality to add to
    // SomeType goes here
}


extension UIColor {
    static var favoriteColor: UIColor {
                return UIColor(red: 0.5,
                               green: 0.1,
                               blue: 0.5,
                               alpha: 1.0)
    }
}
```

Extensions

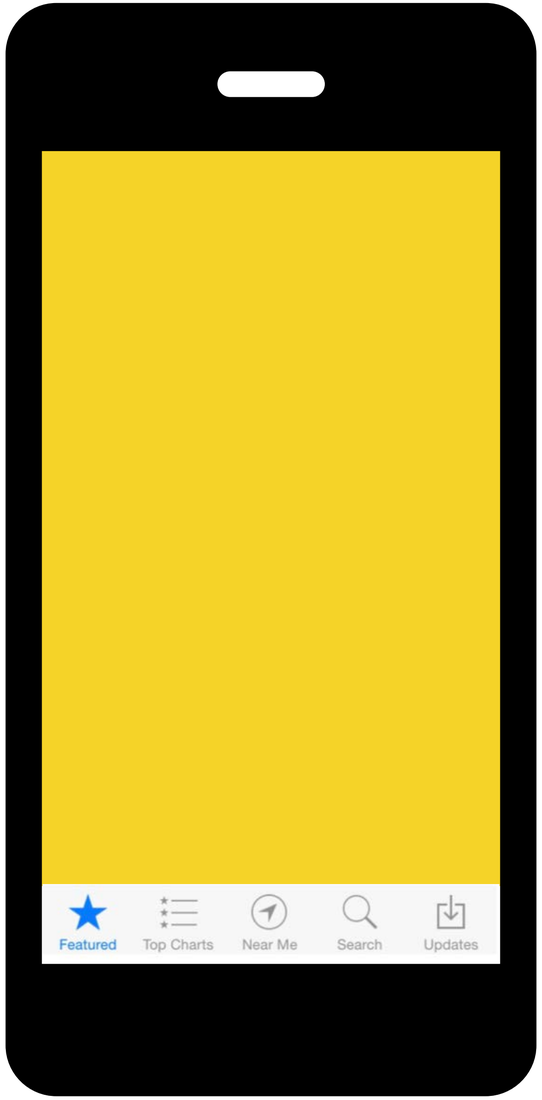
```
struct Employee {
    var firstName: String
    var lastName: String
    var jobTitle: String
    var phoneNumber: String

    static func ==(lhs: Employee,
                   rhs: Employee) -> Bool {
     return lhs.firstName == rhs.firstName &&
   lhs.lastName == rhs.lastName &&
   lhs.companyID == rhs.companyID
 }
}
```

```
struct Employee {
    var firstName: String
    var lastName: String
    var jobTitle: String
    var phoneNumber: String
}

extension Employee : Equatable {
    static func ==(lhs: Employee,
                   rhs: Employee) -> Bool {
     return lhs.firstName == rhs.firstName &&
   lhs.lastName == rhs.lastName &&
   lhs.companyID == rhs.companyID
  }
 }
```

CHAPTER 4

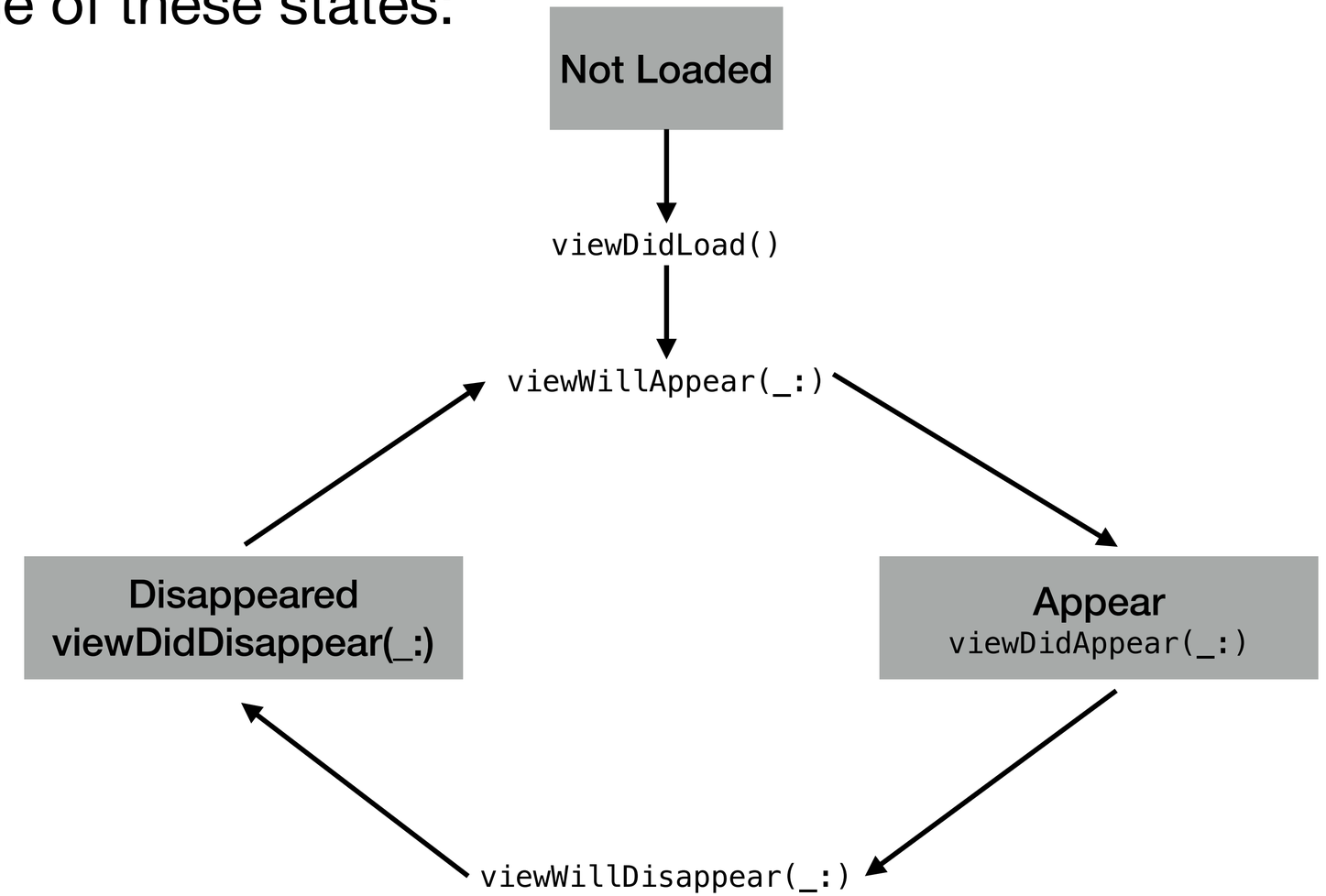
UIView Controllers

Tab Bar Controller

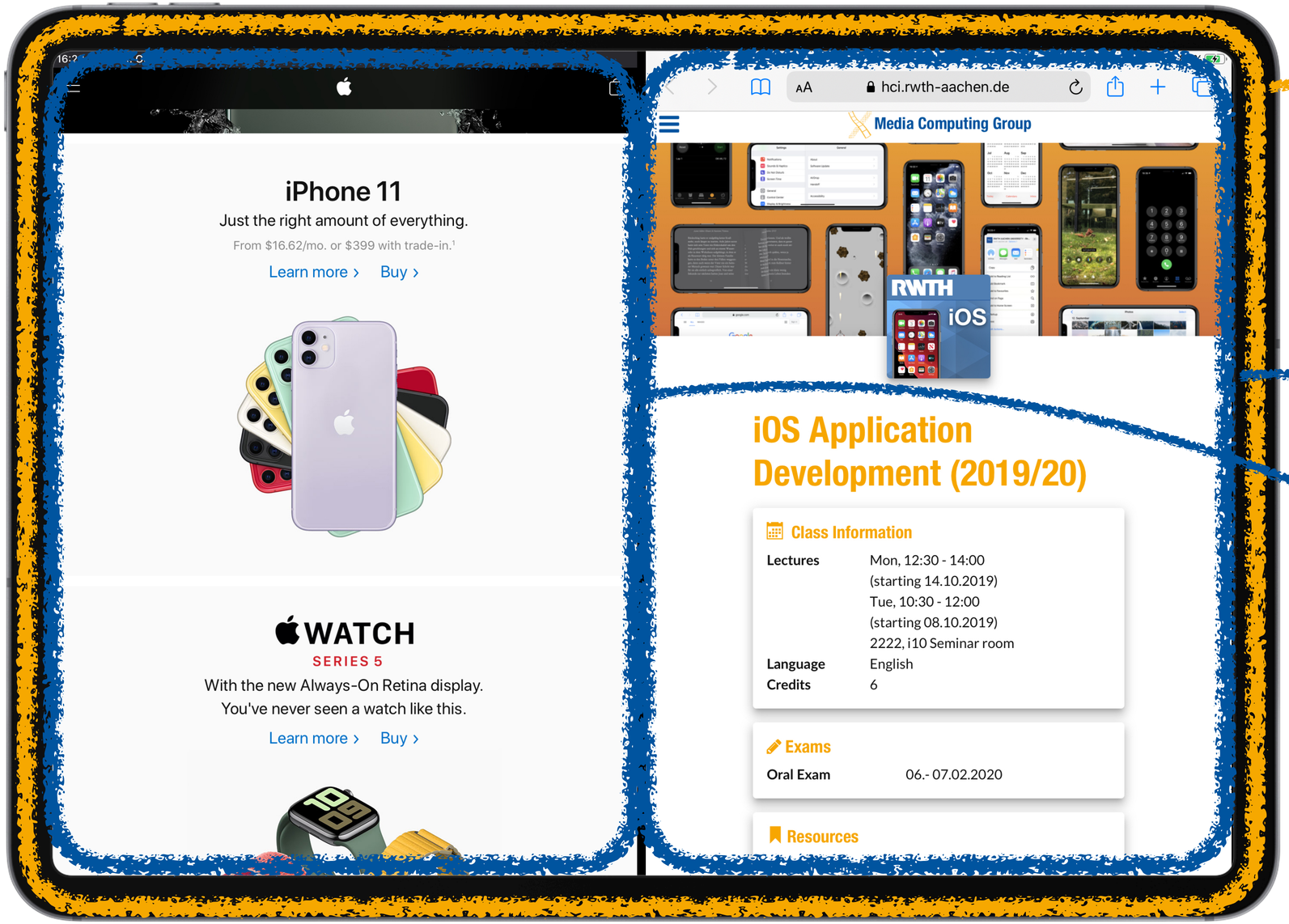


View Controller Life-Cycle

- View Controllers can be in one of these states:
  - View not loaded
  - View appearing
  - View appeared
  - View disappearing
  - View disappeared



Application Life-Cycle



AppDelegate.swift

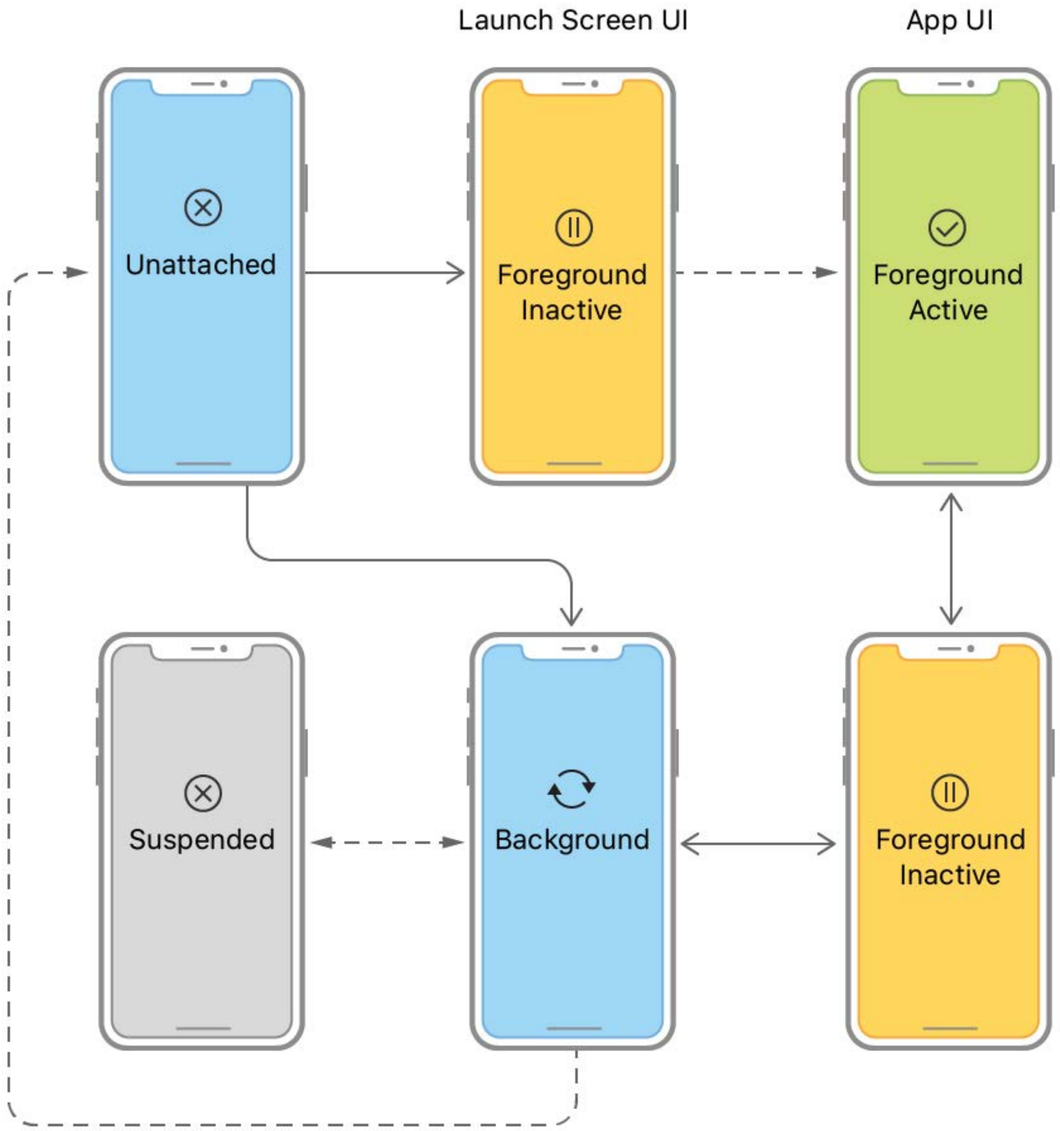
SceneDelegate.swift

AppDelegate.swift

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [UIApplication.LaunchOptionsKey: Any]?) -> Bool {
                return true
    }

func application(_ application: UIApplication, configurationForConnecting
connectingSceneSession: UISceneSession, options: UIScene.ConnectionOptions) ->
UISceneConfiguration {
    }

func application(_ application: UIApplication, didDiscardSceneSessions sceneSessions:
Set<UISceneSession>) {
    }
```

Source: https://developer.apple.com/documentation/uikit/app_and_environment/managing_your_app_s_life_cycle

Scene Life-Cycle



SceneDelegate.swift

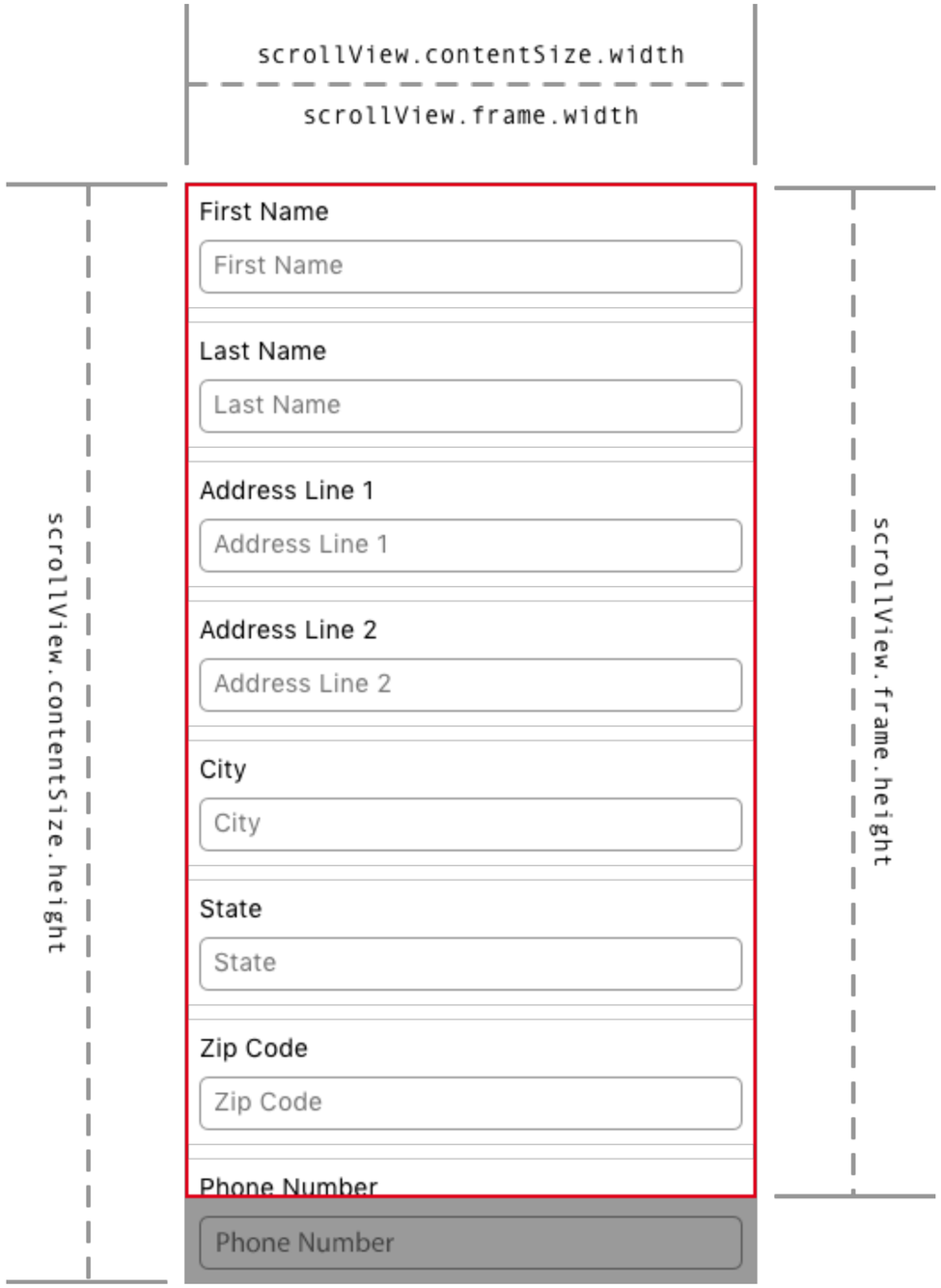
```
func scene(_ scene: UIScene, willConnectTo session: UISceneSession, options
connectionOptions: UIScene.ConnectionOptions) {
    }

func sceneDidDisconnect(_ scene: UIScene) {
    }

func sceneDidBecomeActive(_ scene: UIScene) {
    }

func sceneWillResignActive(_ scene: UIScene) {
    }

func sceneWillEnterForeground(_ scene: UIScene) {
    }

func sceneDidEnterBackground(_ scene: UIScene) {
    }
```

CHAPTER 4

Scroll Views

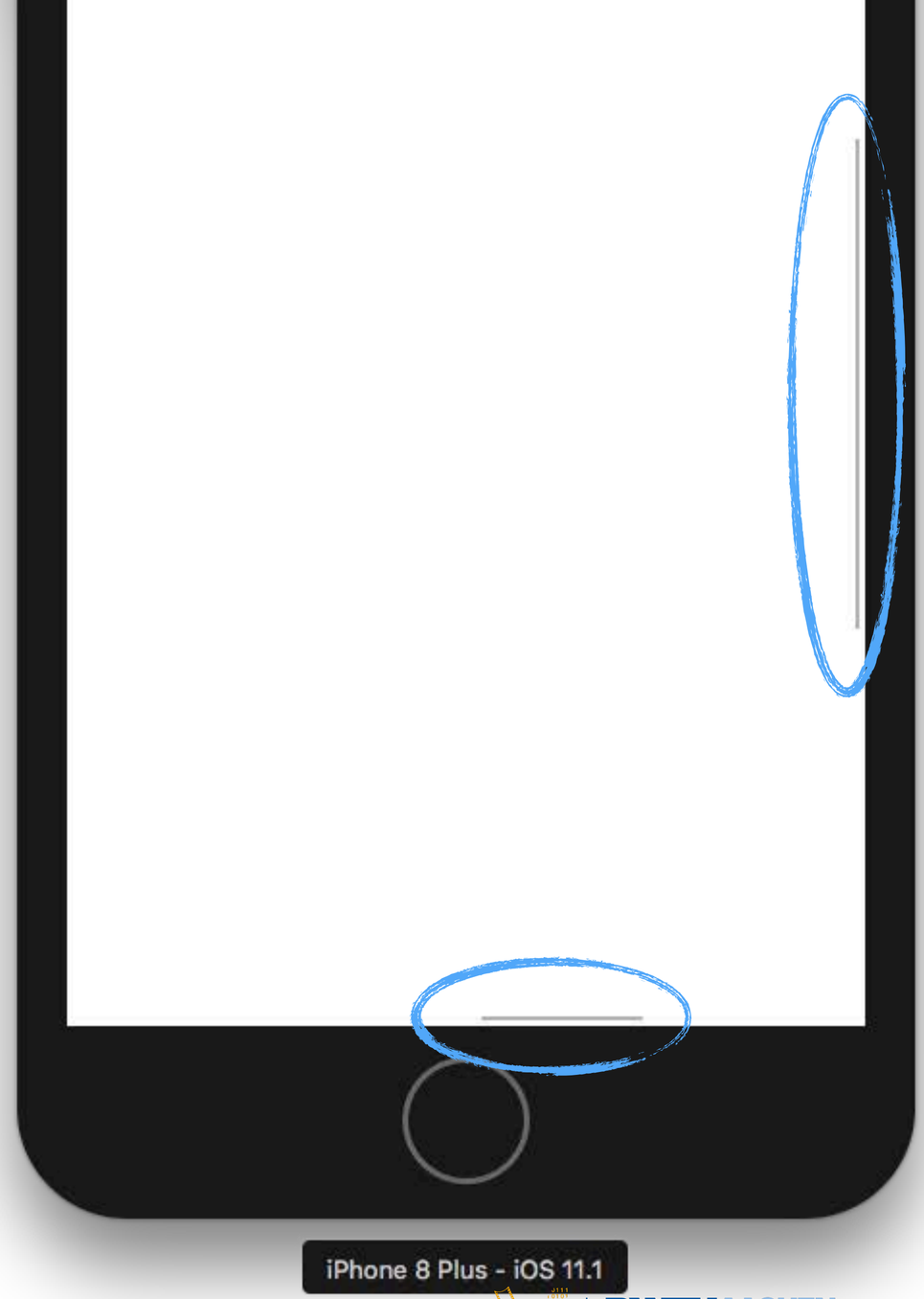
ScrollView

- UIScrollView
  - Parent of UITableView
- Show content that does not fit on one screen
- .frame property
  - Where on the screen is the ScrollView?
- .contentSize property
  - How large is the scrollable area?



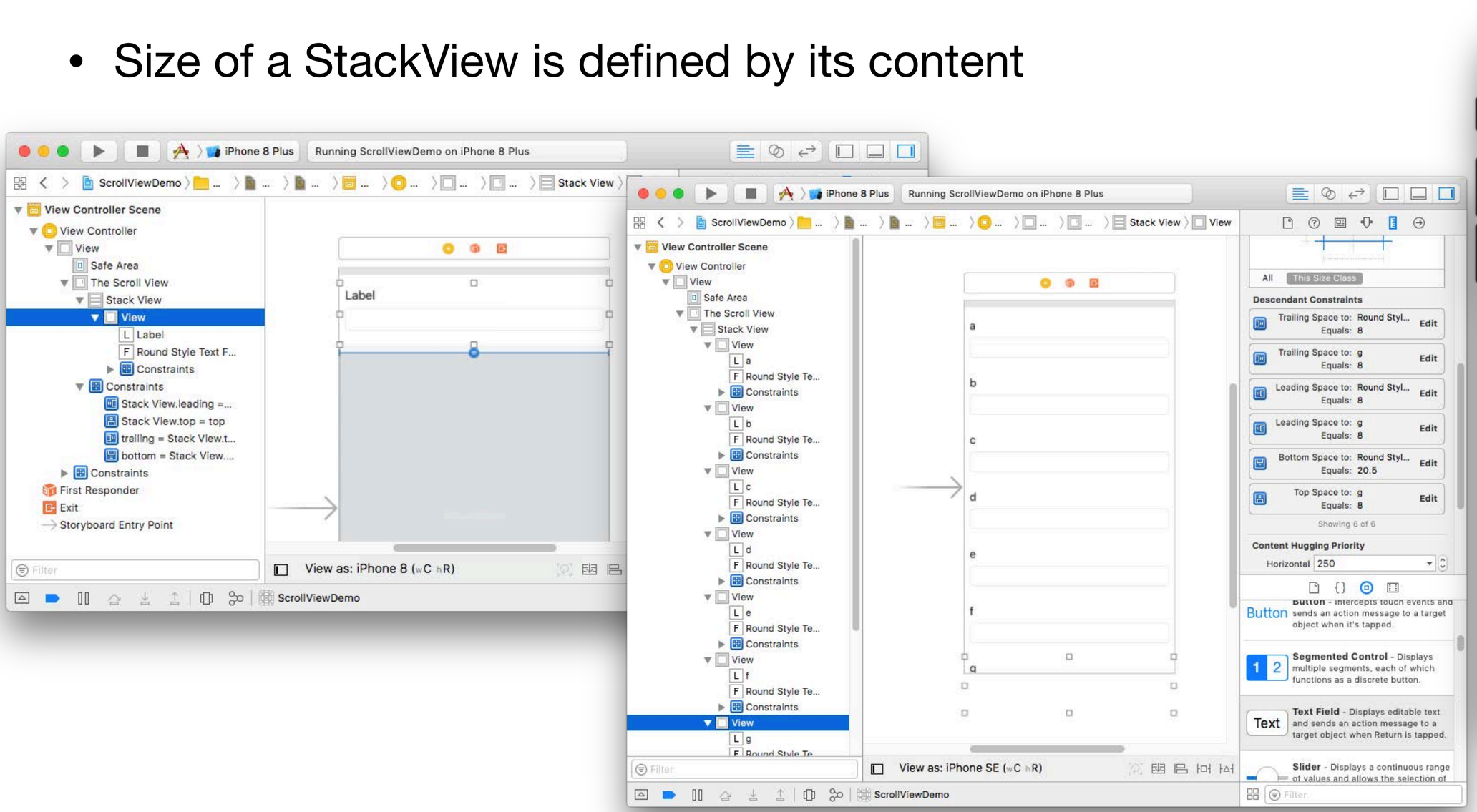
Setting the contentSize

```
let aView = UIView.init(frame: CGRect.init(
x: 0, y: 0, width: 2000, height: 2000))

self.theScrollView.addSubview(aView)
self.theScrollView.contentSize = aView.frame.size
```



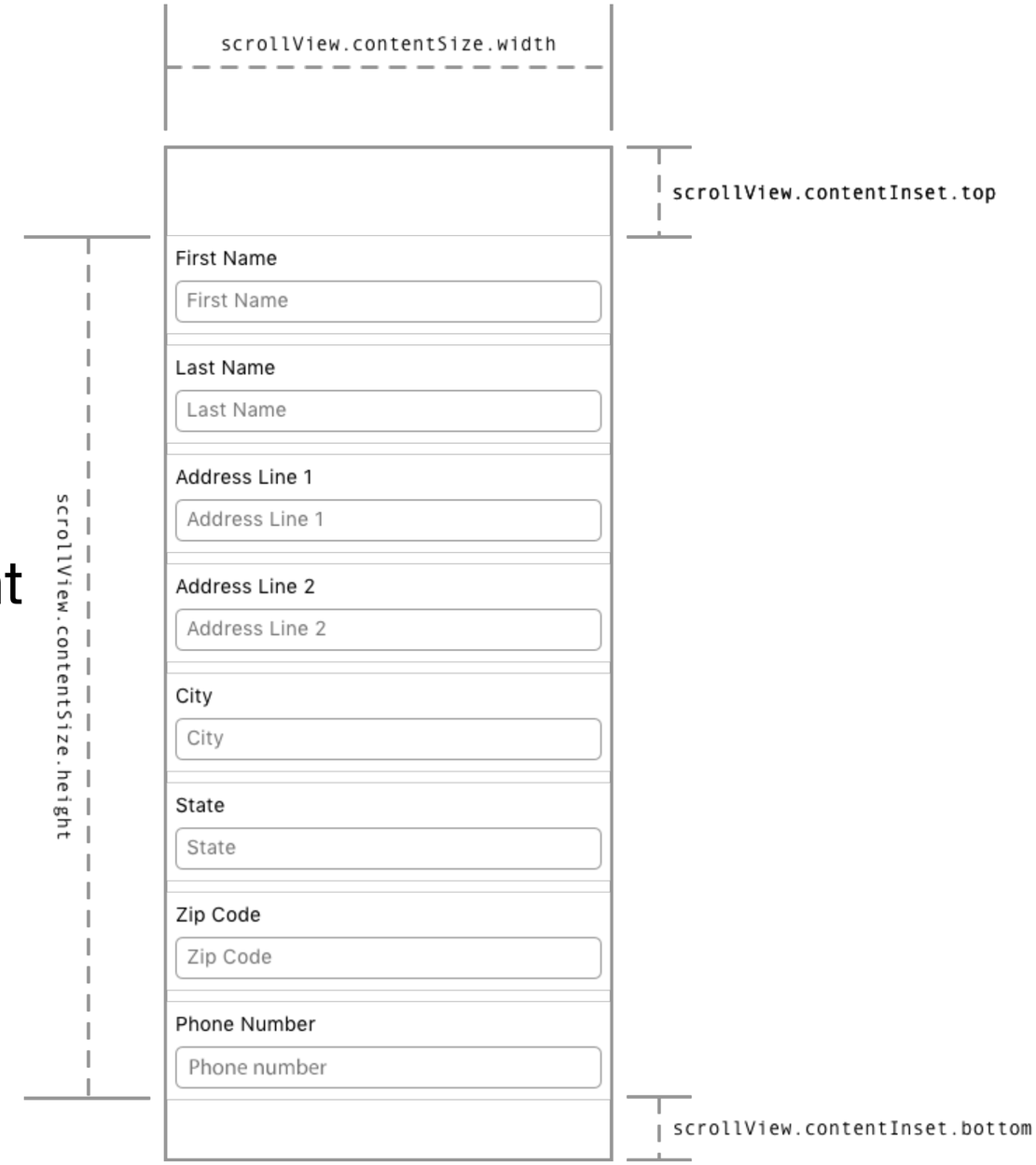
ScrollView with StackView

- Size of a StackView is defined by its content



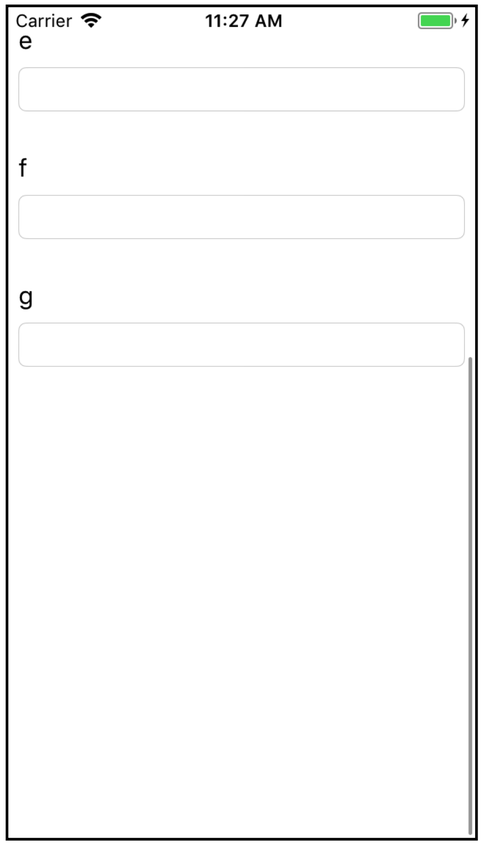
Content Insets

- Use insets to provide padding to your content
  - e.g., on the bottom when displaying the keyboard

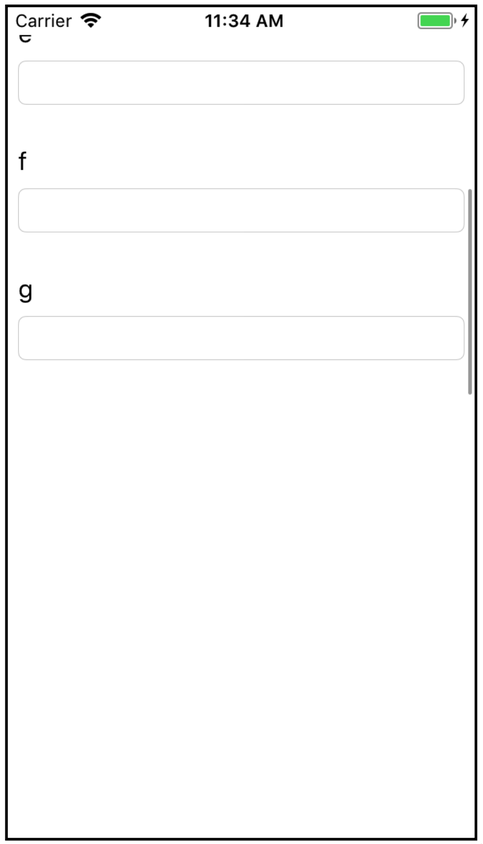


Content Insets

```
let contentInsets =
UIEdgeInsetsMake(0.0, 0.0, 300, 0.0)
self.theScrollView.contentInset =
contentInsets
```




```
self.theScrollView.scrollIndicatorInsets =
contentInsets
```



UIScrollViewDelegate

```
func scrollViewDidScroll(_ scrollView: UIScrollView) {
}

func viewForZooming(in scrollView: UIScrollView) -> UIView? {
}

func scrollViewDidEndScrollingAnimation(_ scrollView: UIScrollView) {
}

func scrollViewWillBeginDragging(_ scrollView: UIScrollView) {
}

func scrollViewWillBeginDecelerating(_ scrollView: UIScrollView) {
}

...
```

MVC in iOS