Developer Tools #WWDC16

Protocol and Value Oriented Programming in UIKit Apps

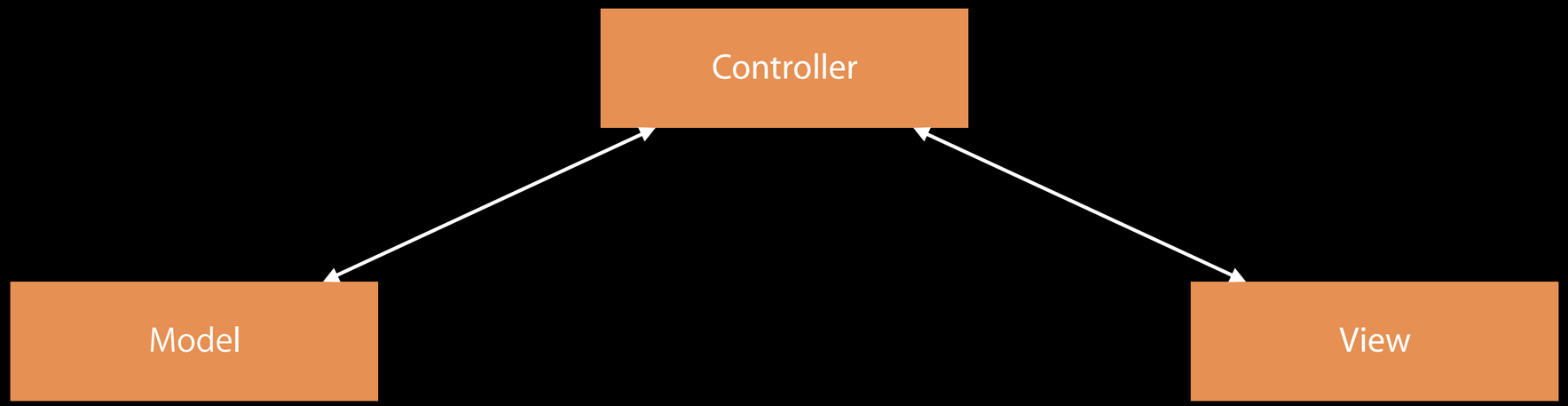
Swift in practice

Session 419

Jacob Xiao Protocol Oriented Programmer
Alex Migicovsky Swift Compiler Typo Engineer

Local Reasoning

Model View Controller

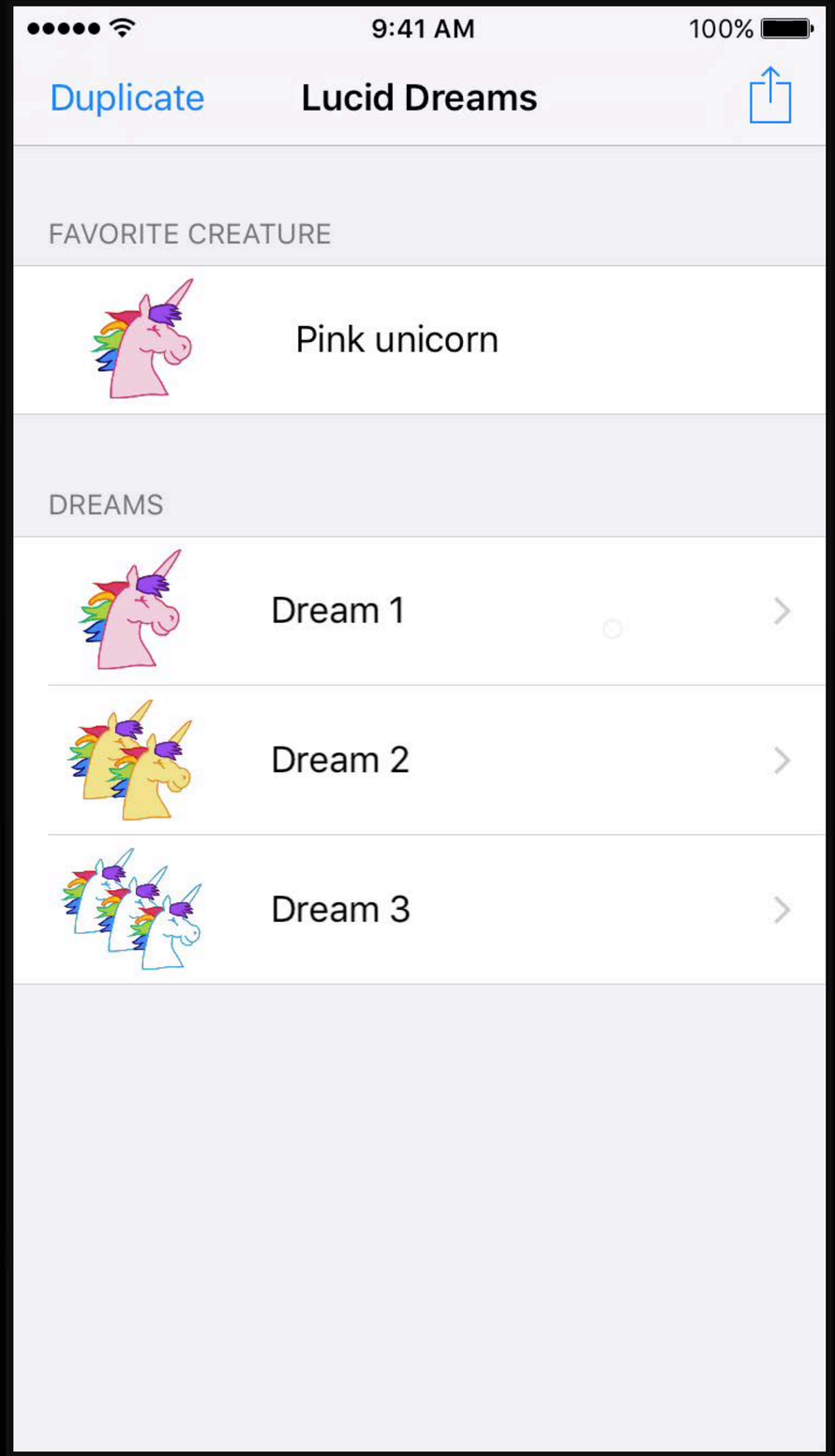


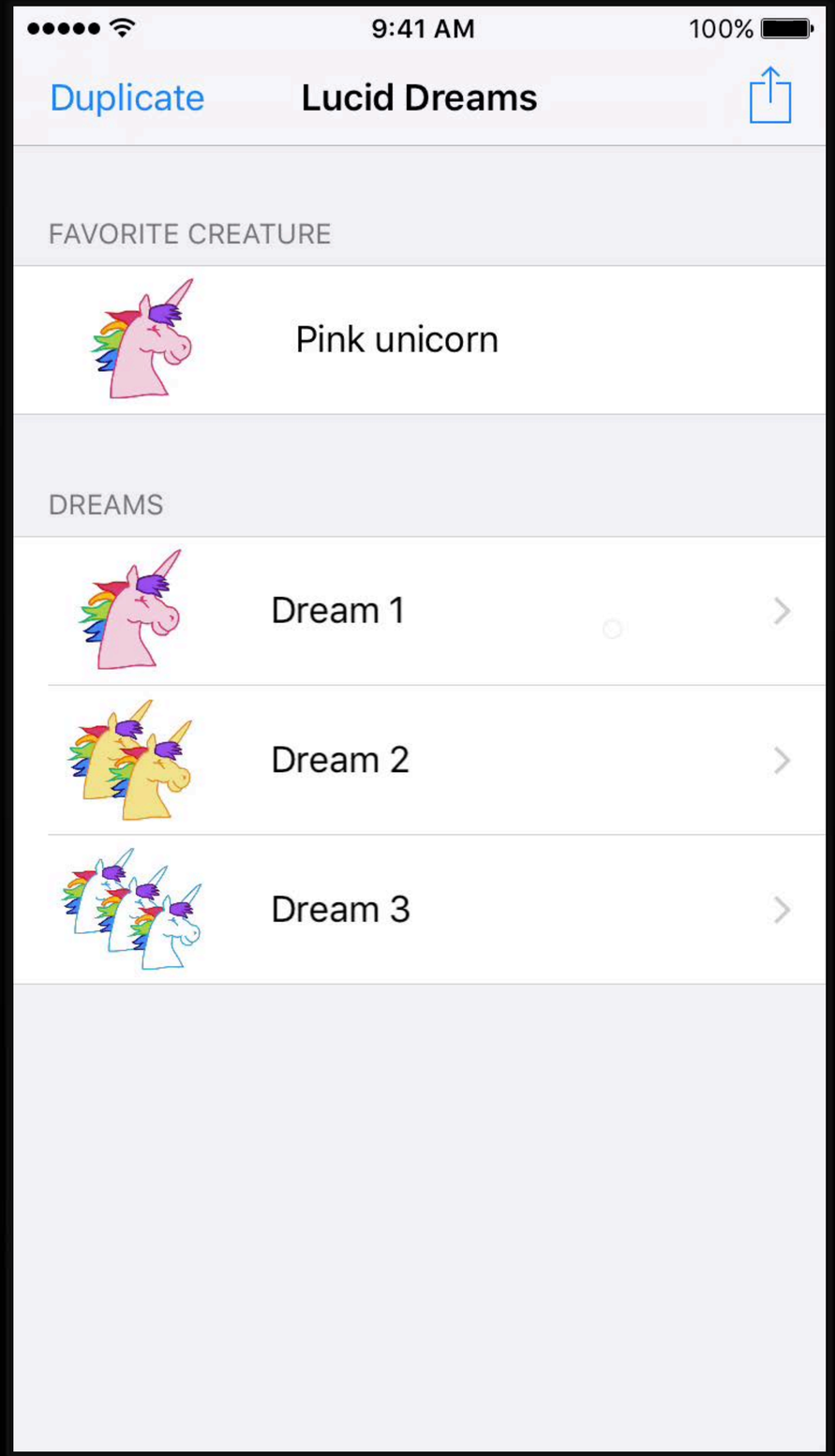
Lucid Dreams


UNICORN


9:41 AM
100%
LucidDreams
Phone
Safari
Mail
Music


9:41 AM
100%
LucidDreams
Phone
Safari
Mail
Music


9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3


9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

Think Different

Protocol-Oriented Programming in Swift	WWDC 2015
Building Better Apps with Value Types in Swift	WWDC 2015

Overview

Value types and protocols

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

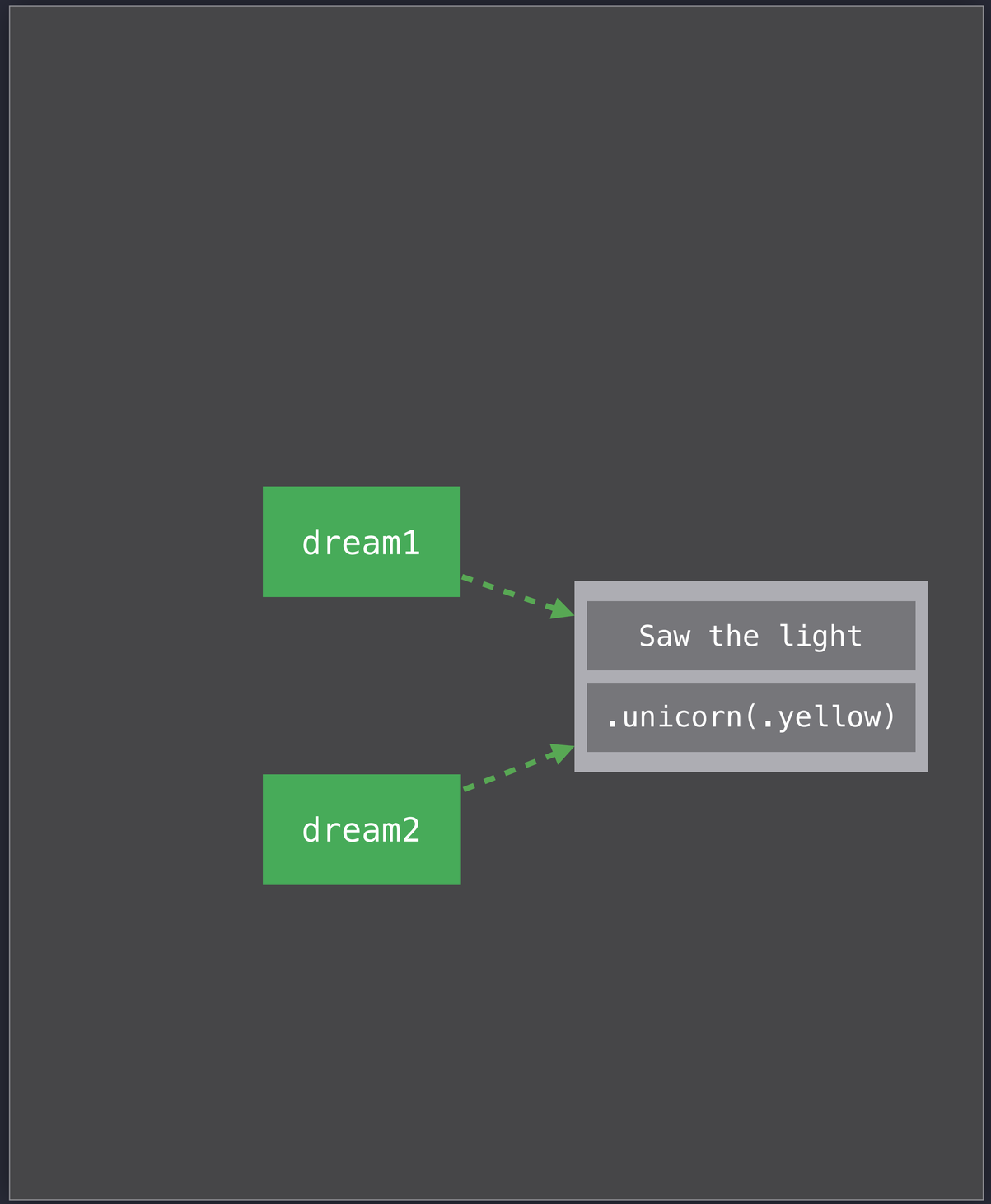
Sample code: https://developer.apple.com/go/?id=lucid-dreams

Model

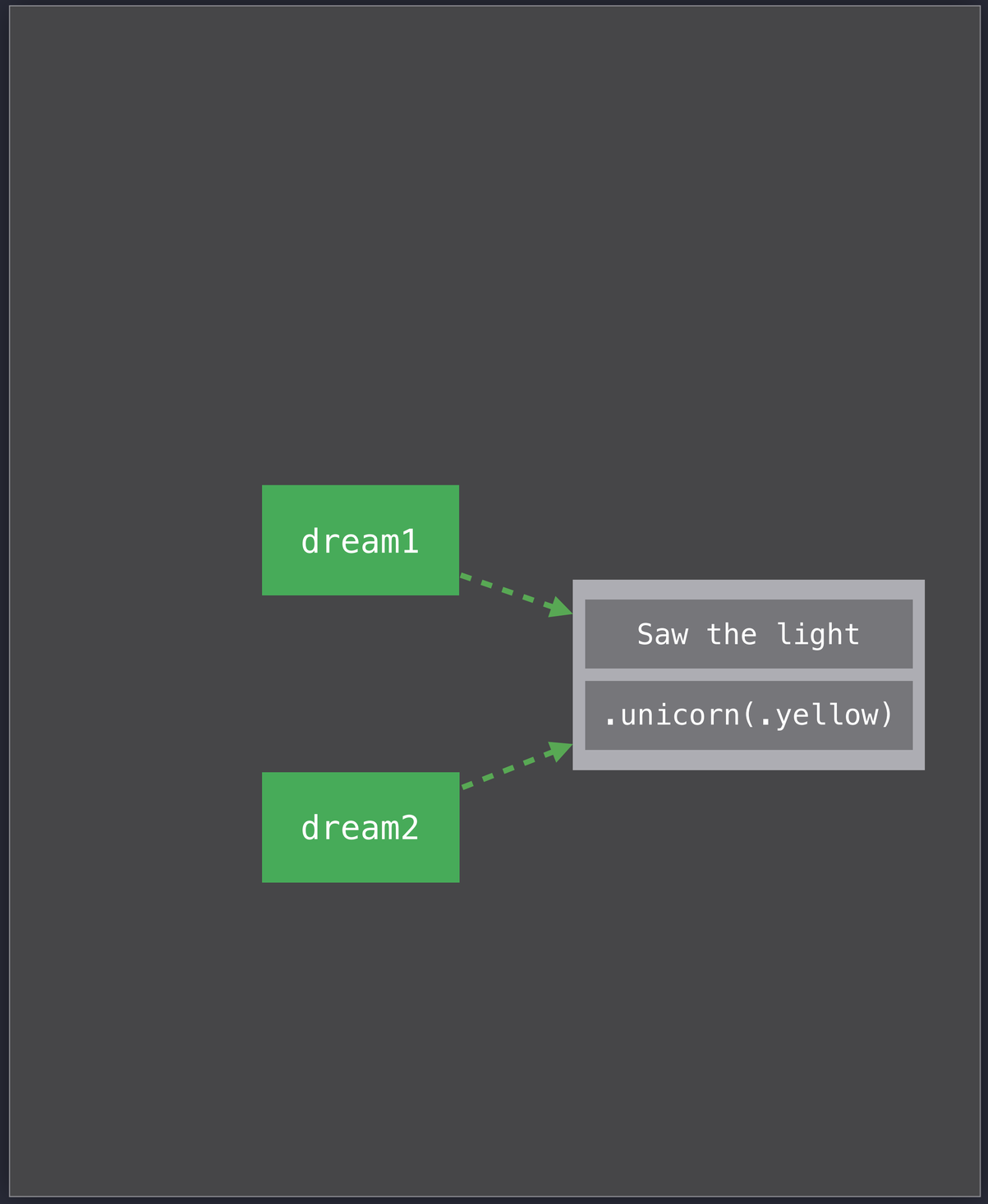
What's a dream?

```
// Reference Semantics

class Dream {
    var description: String
    var creature: Creature
    var effects: Set<Effect>

    ...
}
```

```
// Reference Semantics

class Dream {
    var description: String
    var creature: Creature
    var effects: Set<Effect>

    ...
}

var dream1 = Dream(...)
var dream2 = dream1
```

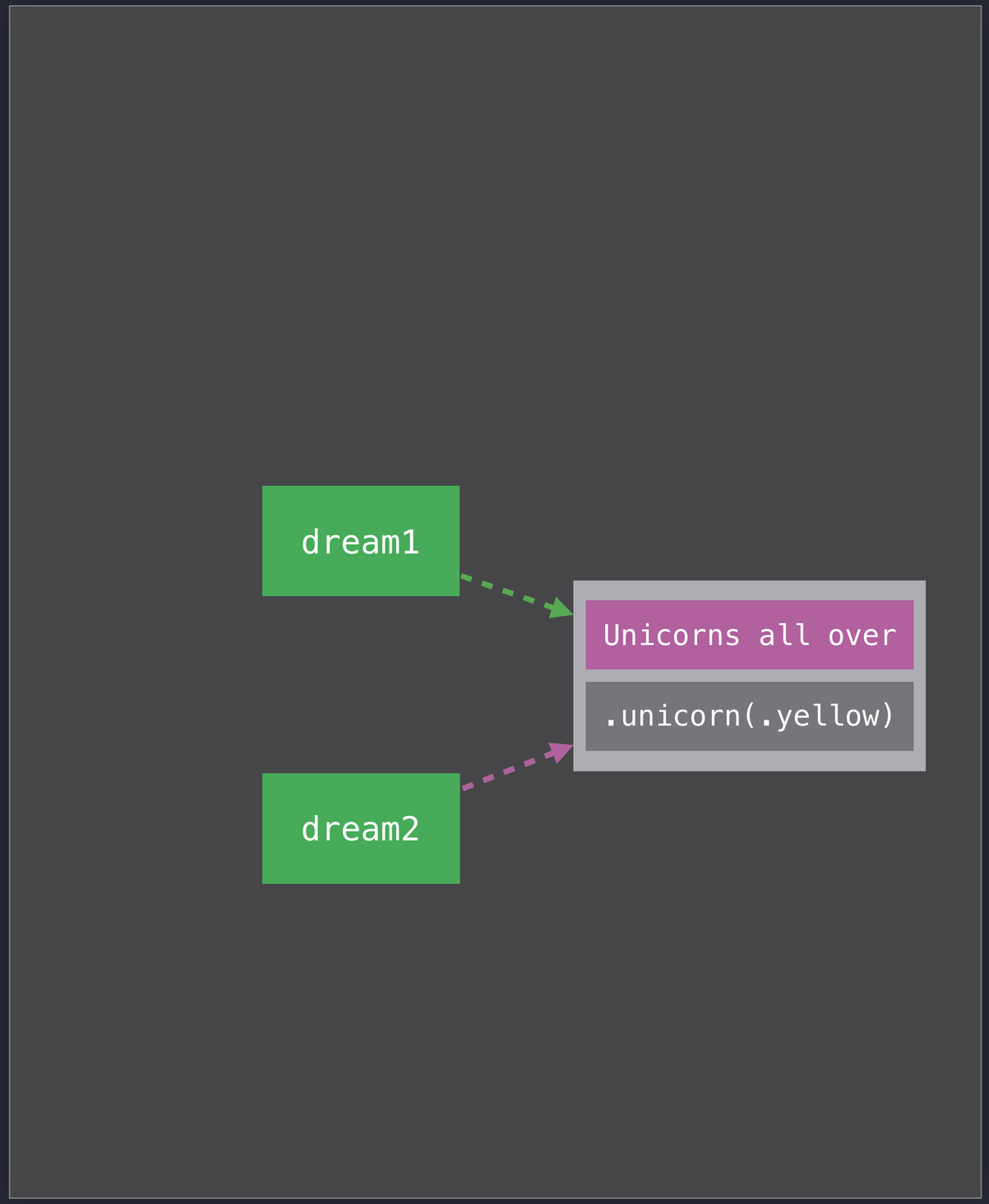


```
// Reference Semantics

class Dream {
    var description: String
    var creature: Creature
    var effects: Set<Effect>

    ...
}

var dream1 = Dream(...)
var dream2 = dream1
dream2.description = "Unicorns all over"
```



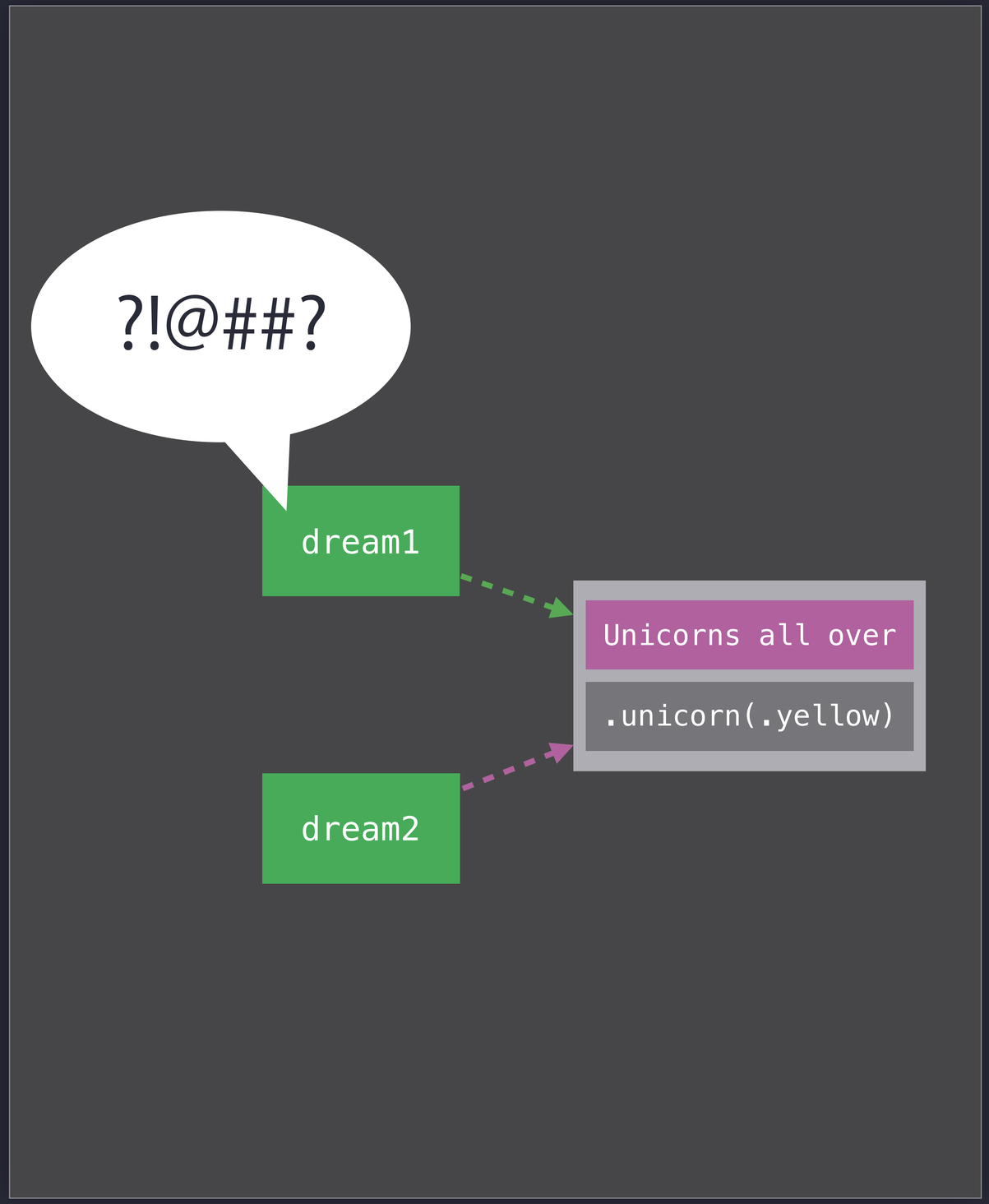
```
// Reference Semantics

class Dream {
    var description: String
    var creature: Creature
    var effects: Set<Effect>

    ...
}
```

```
var dream1 = Dream(...)
var dream2 = dream1
dream2.description = "Unicorns all over"
```



```
// Reference Semantics

class Dream {
    var description: String
    var creature: Creature
    var effects: Set<Effect>

    ...
}
```

```
var dream1 = Dream(...)
var dream2 = dream1
dream2.description = "Unicorns all over"
```



Relationships

App Delegate

Navigation VC

Dreams VC

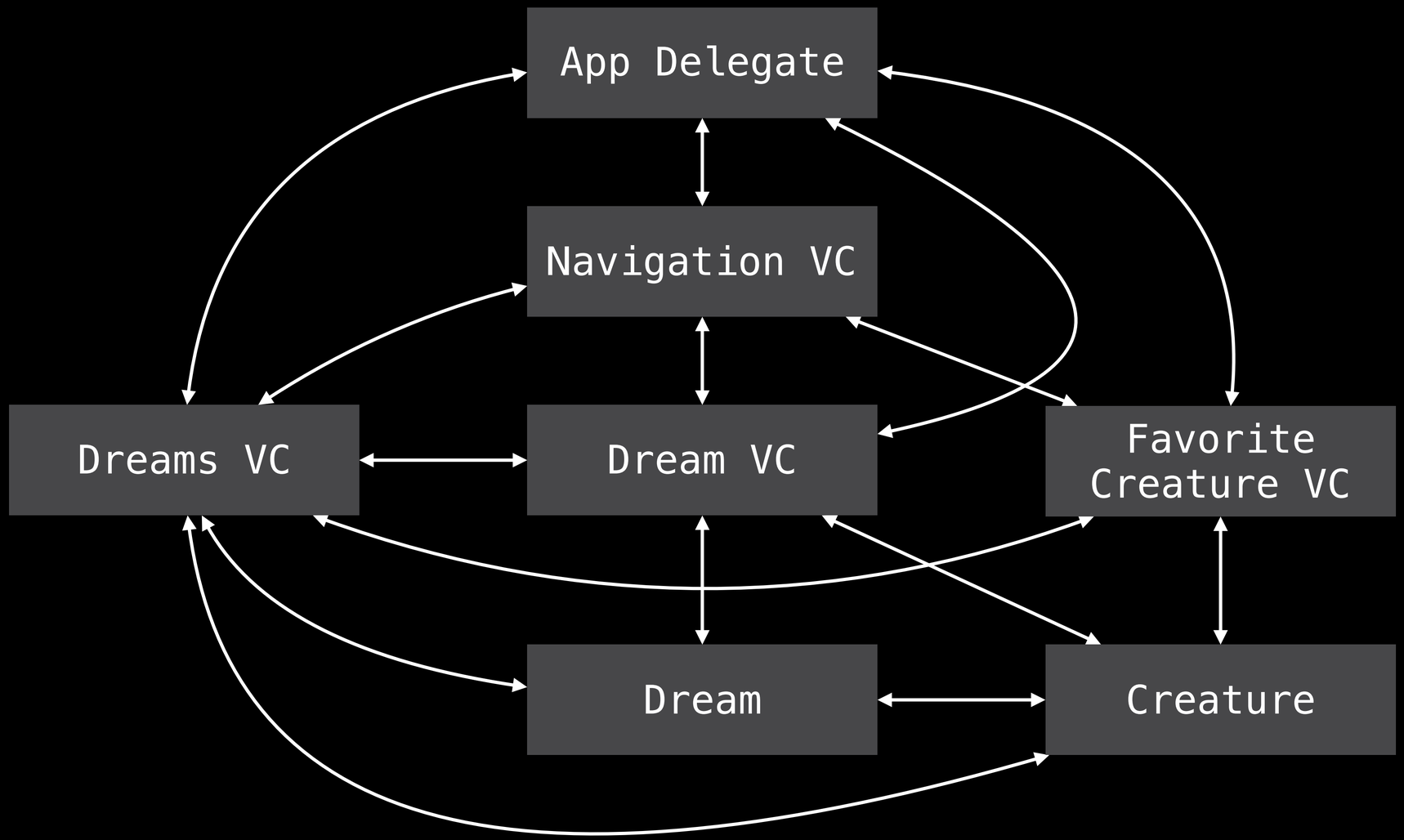
Dream VC

Favorite Creature VC

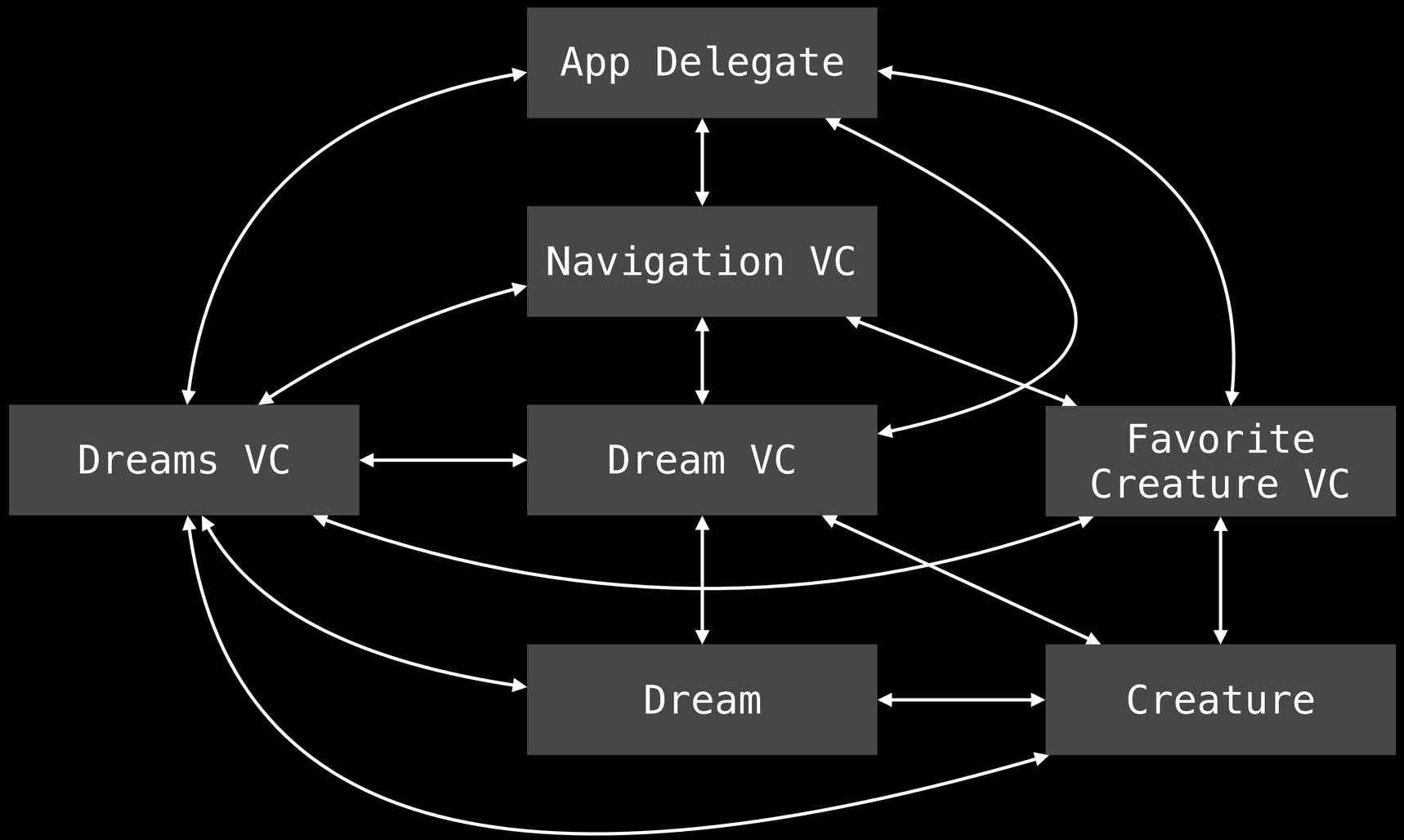
Dream

Creature

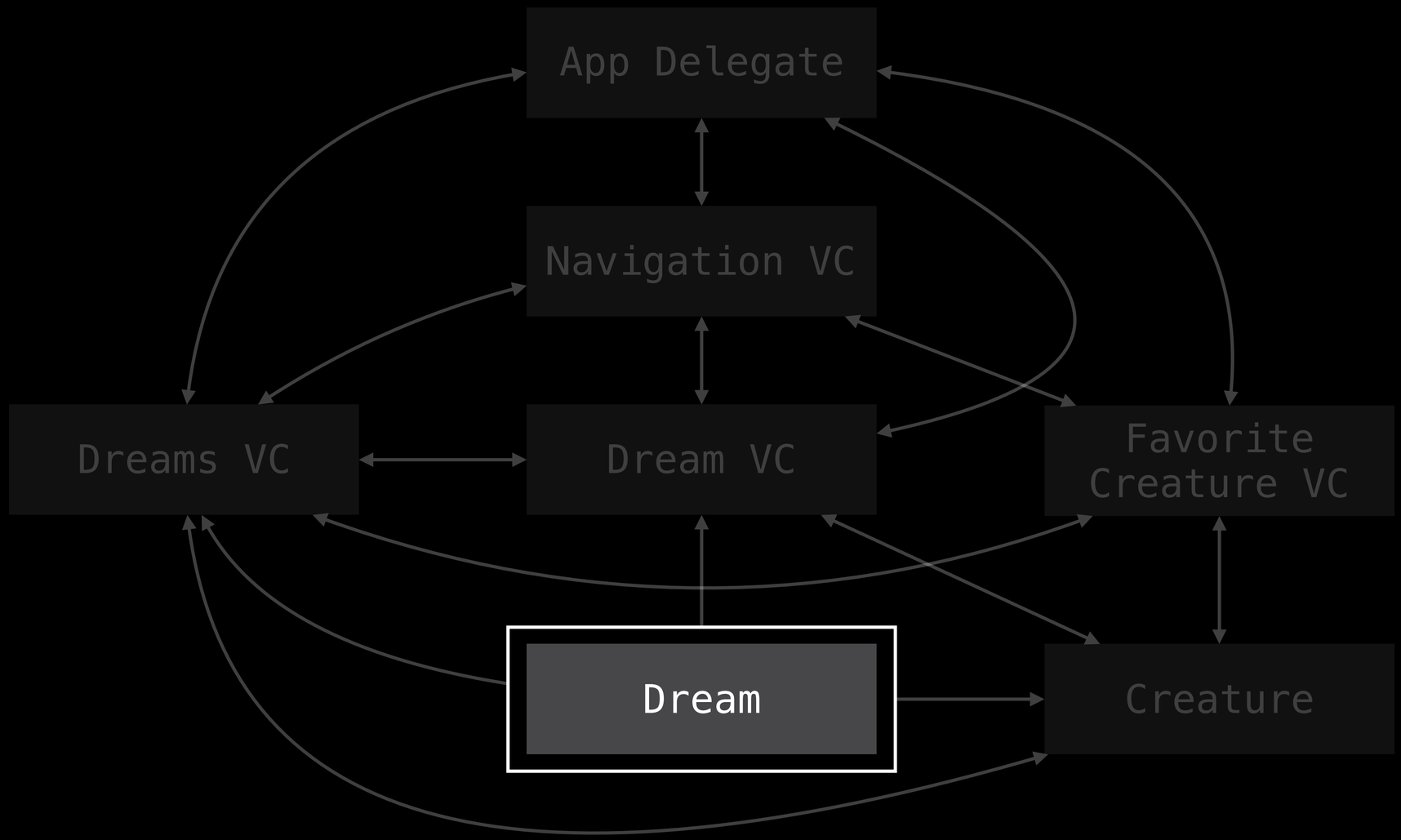
Relationships



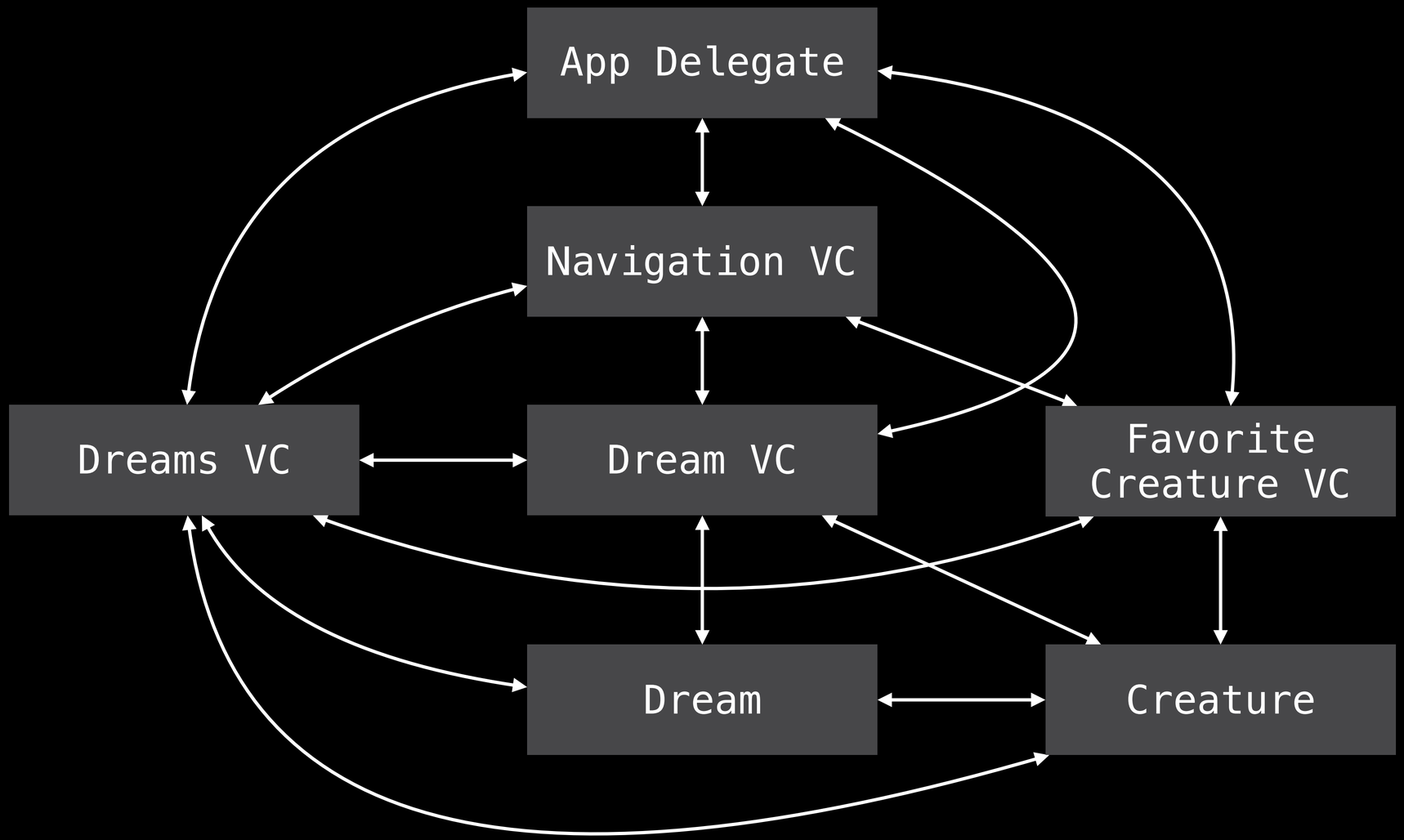
Relationships—It's Complicated…



Relationships—It's Complicated…

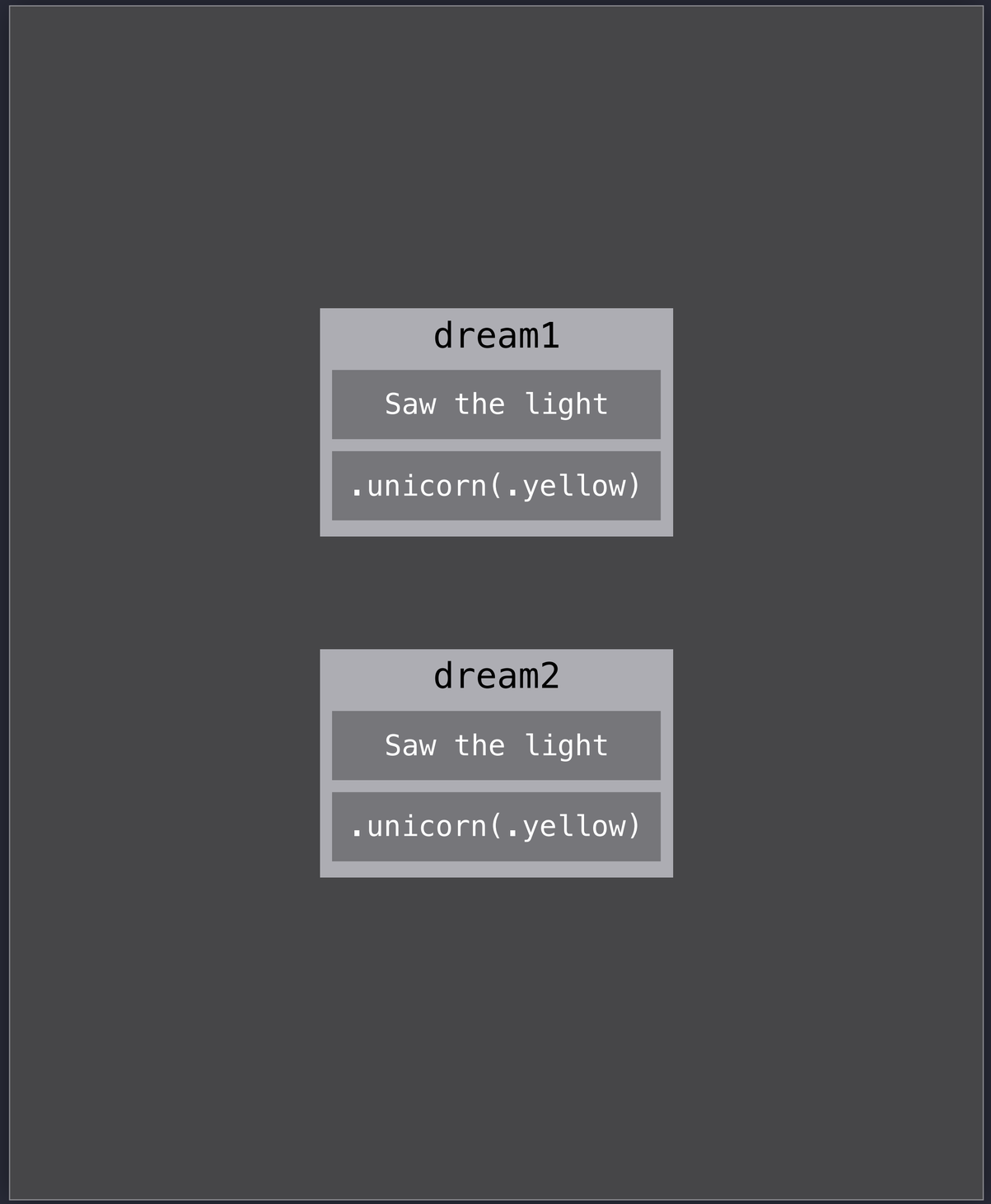


Relationships—It's Complicated…



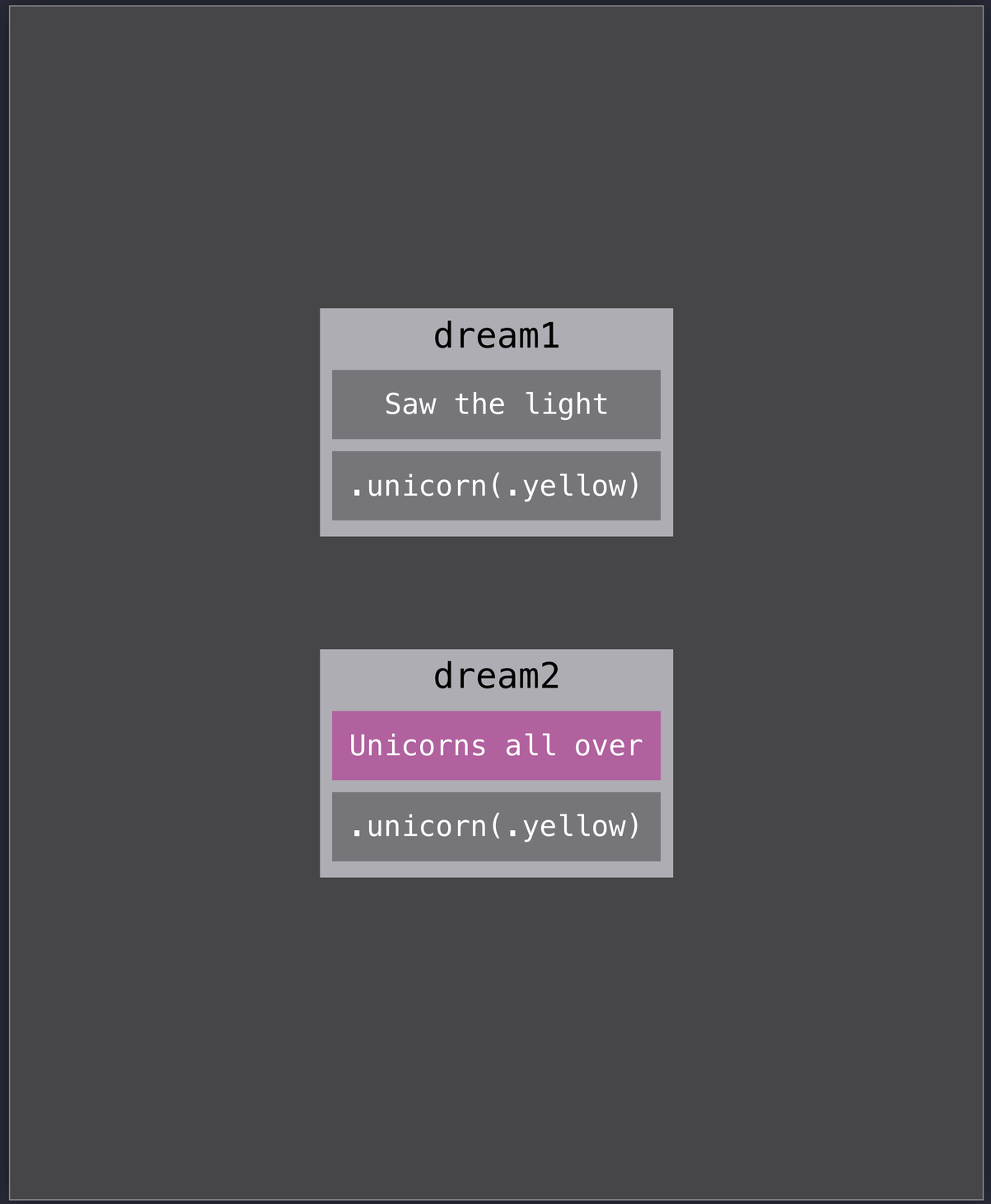
```
// Value Semantics

struct Dream {
    var description: String
    var creature: Creature
    var effects: Set<Effect>

    ...
}
```

```
var dream1 = Dream(...)
var dream2 = dream1
```



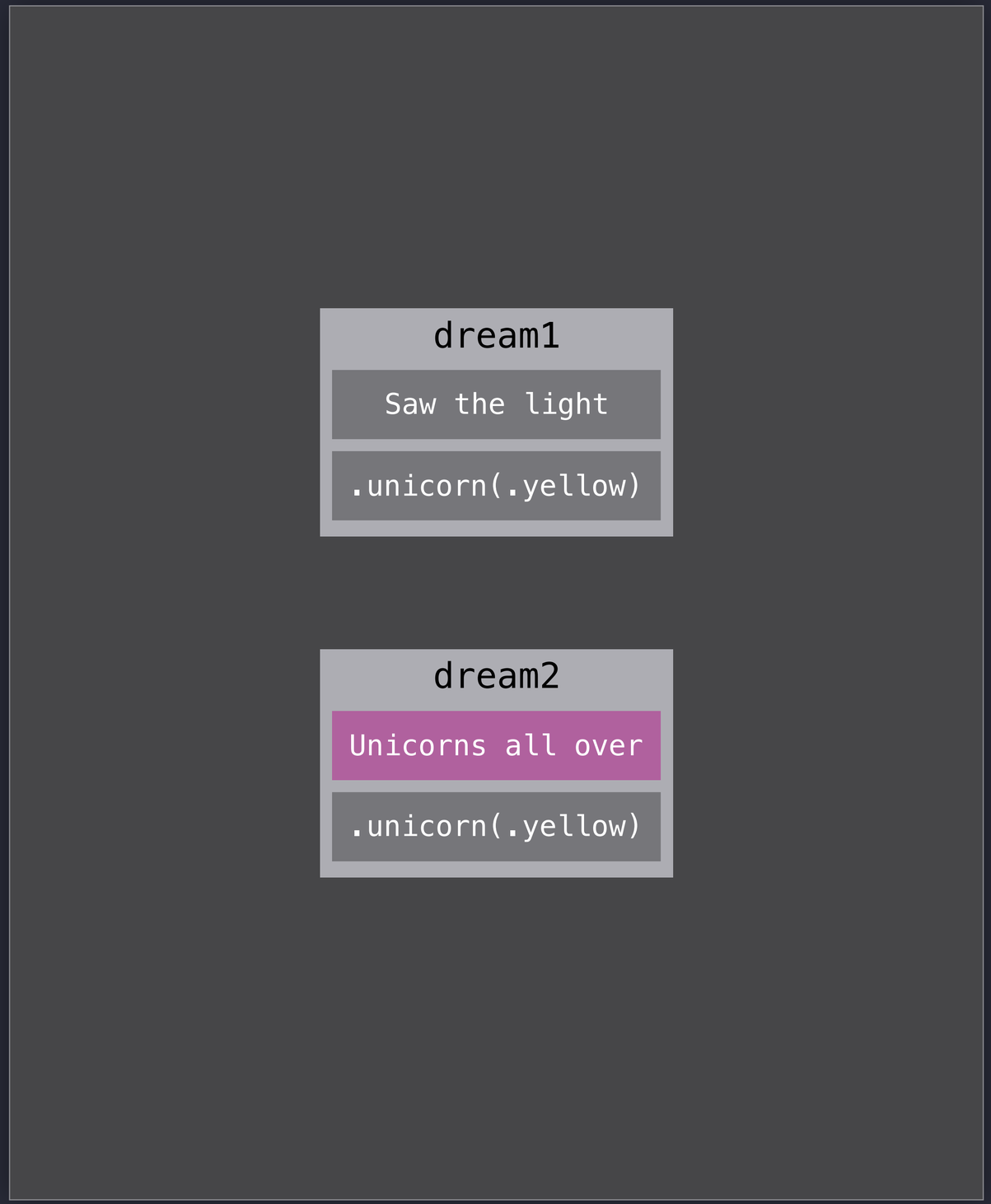
```
// Value Semantics

struct Dream {
    var description: String
    var creature: Creature
    var effects: Set<Effect>

    ...
}
```

```
var dream1 = Dream(...)
var dream2 = dream1
dream2.description = "Unicorns all over"
```



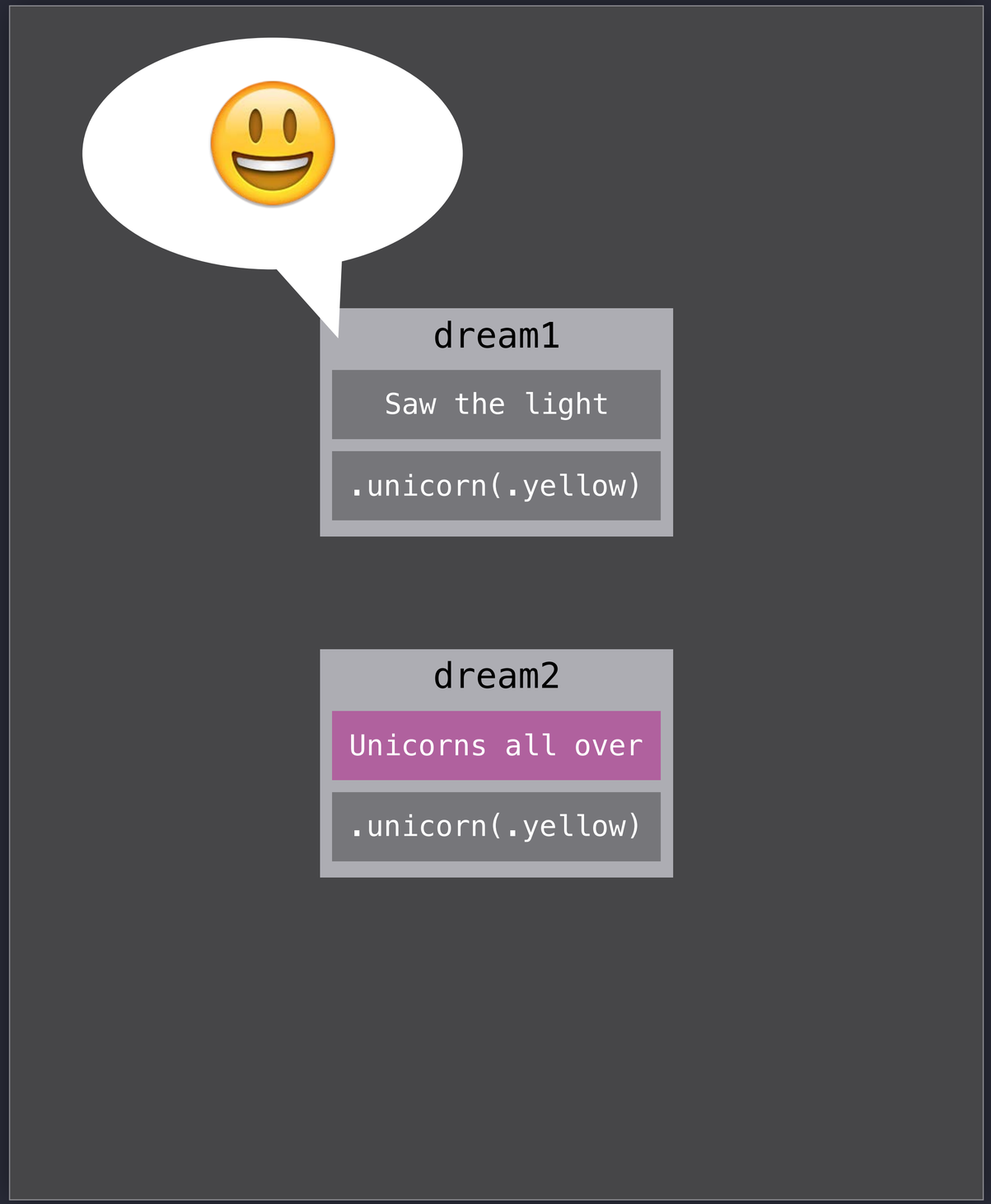
```
// Value Semantics

struct Dream {
    var description: String
    var creature: Creature
    var effects: Set<Effect>

    ...
}
```

```
var dream1 = Dream(...)
var dream2 = dream1
dream2.description = "Unicorns all over"
```



```
// Value Semantics

struct Dream {
    var description: String
    var creature: Creature
    var effects: Set<Effect>

    ...
}
```

```
var dream1 = Dream(...)
var dream2 = dream1
dream2.description = "Unicorns all over"
```



“Use values only for simple model types.”

The Internet™

~~“Use values only for simple model types.”~~

The Internet™

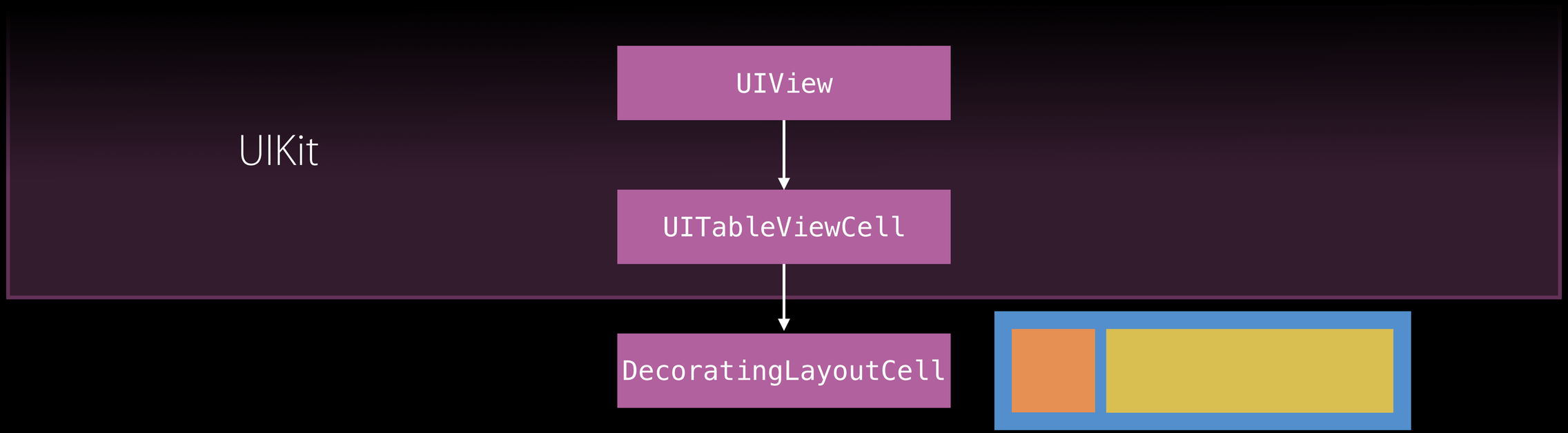
View

Cell layout

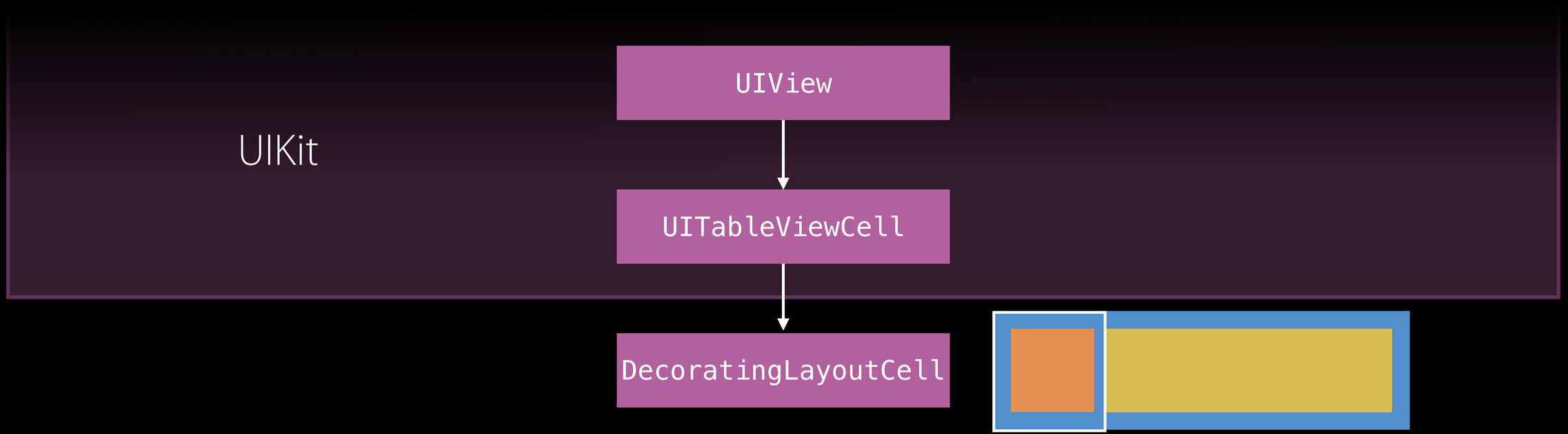
Jacob Xiao Protocol Oriented Programmer

9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

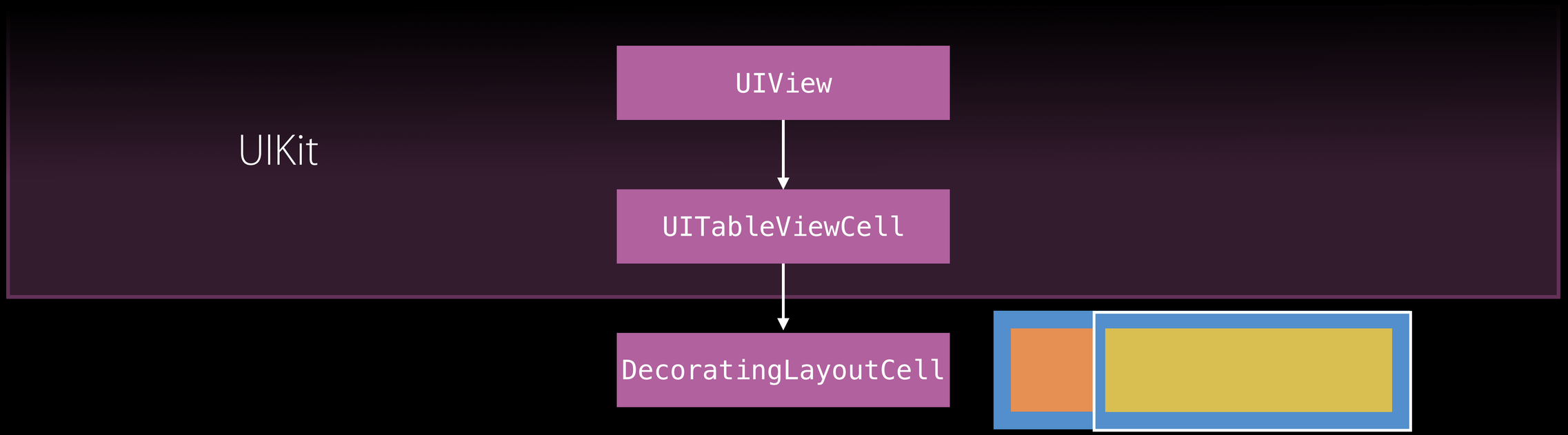
Cell Layout



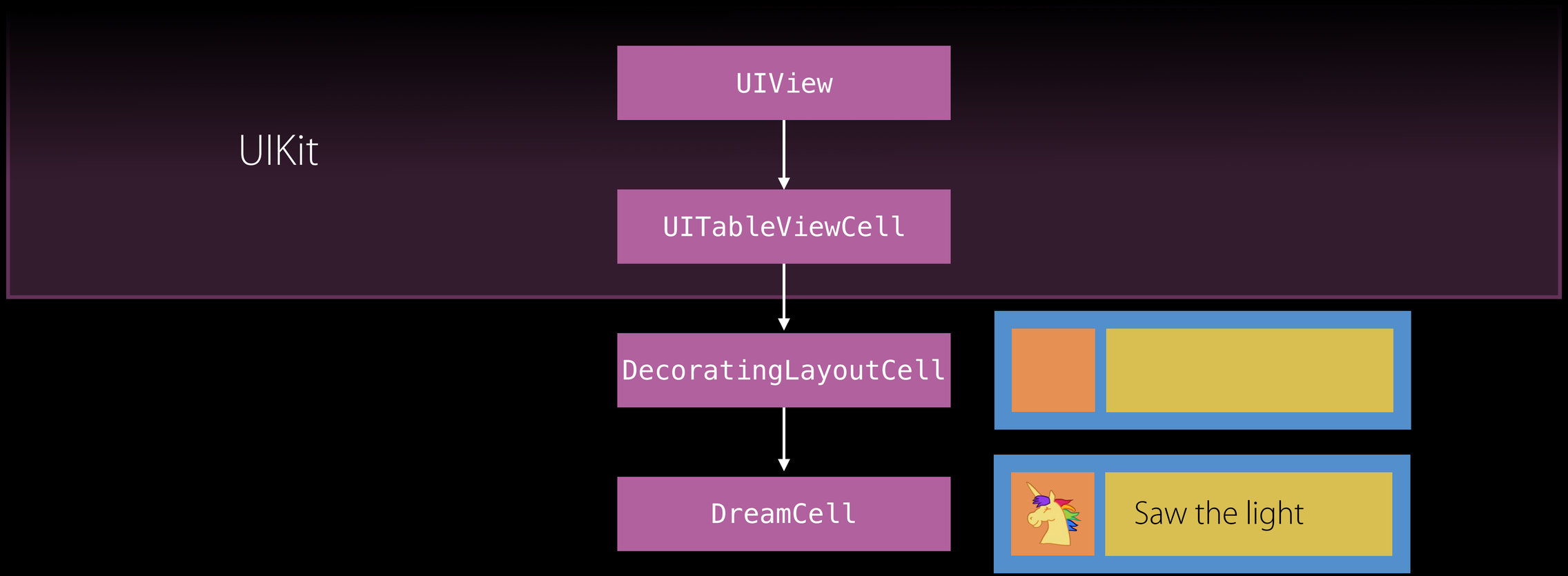
Cell Layout



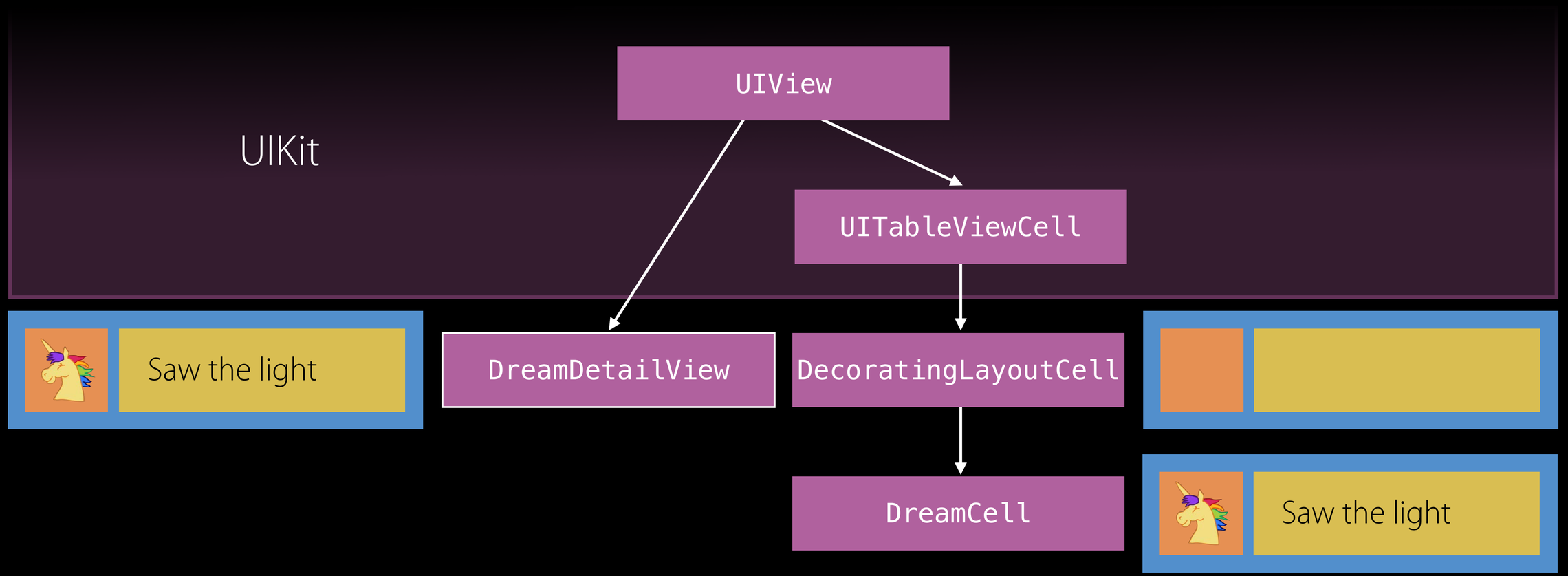
Cell Layout



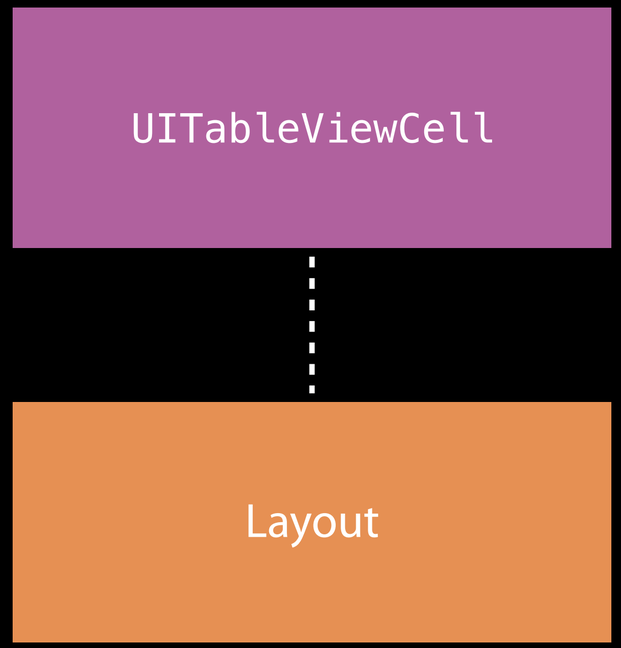
Cell Layout


UIKit
UIView
UITableViewCell
DecoratingLayoutCell
DreamCell
Saw the light

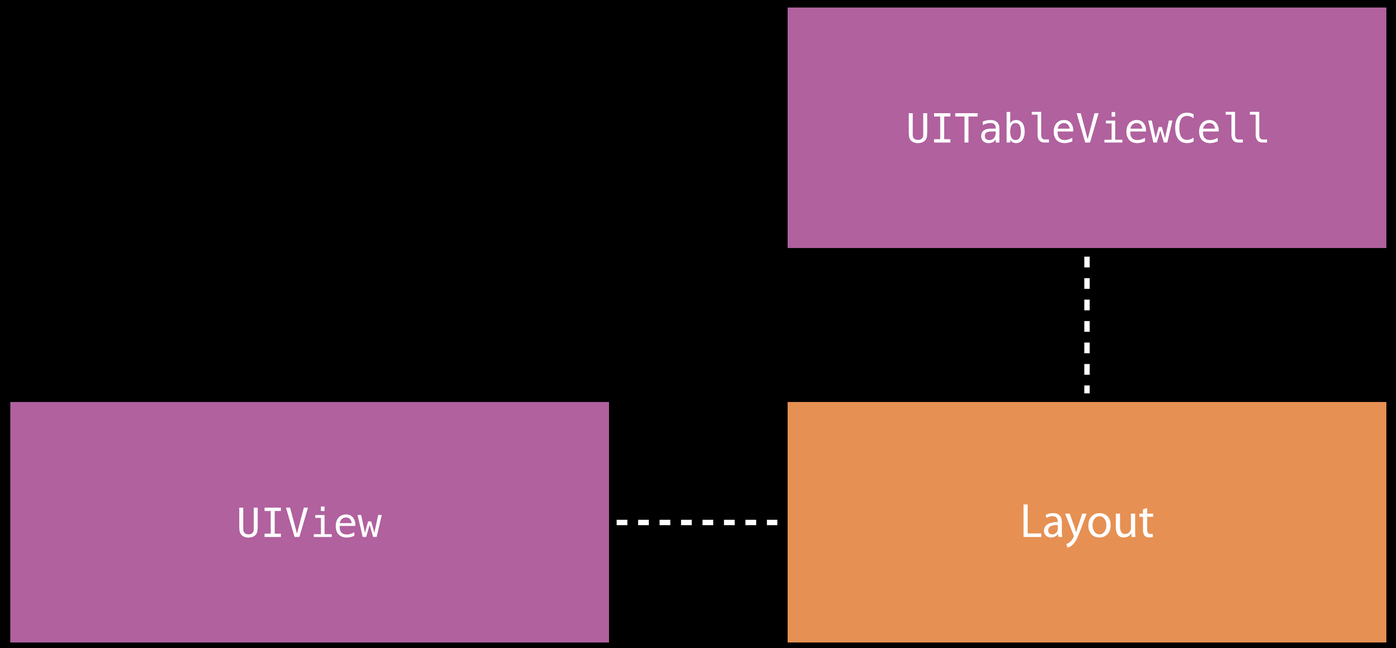
Cell Layout


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

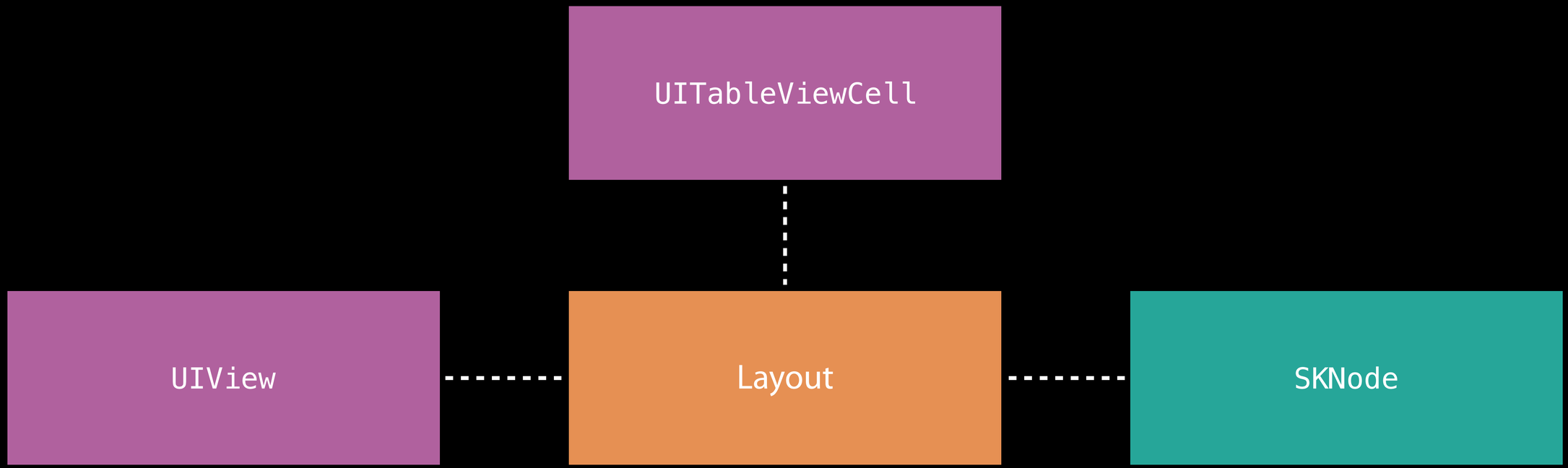
Cell Layout



Cell Layout



Cell Layout



```
// Cell Layout

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

   // Perform layout...
}
```

```
// View 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)
   }
}
```

```
// 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)
    }
}
```

```
// 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)
    }
}
```

```
// Testing
```

```
// Testing

func testLayout() {
    let child1 = UIView()
    let child2 = UIView()
```

```
// Testing

func testLayout() {
    let child1 = UIView()
    let child2 = UIView()

    var layout = DecoratingLayout(content: child1, decoration: child2)
    layout.layout(in: CGRect(x: 0, y: 0, width: 120, height: 40))
```

```
// Testing

func testLayout() {
    let child1 = UIView()
    let child2 = UIView()

    var layout = DecoratingLayout(content: child1, decoration: child2)
    layout.layout(in: CGRect(x: 0, y: 0, width: 120, height: 40))
```

```
// Testing

func testLayout() {
    let child1 = UIView()
    let child2 = UIView()

    var layout = DecoratingLayout(content: child1, decoration: child2)
    layout.layout(in: CGRect(x: 0, y: 0, width: 120, height: 40))

    XCTAssertEqual(child1.frame, CGRect(x: 0, y: 5, width: 35, height: 30))
    XCTAssertEqual(child2.frame, CGRect(x: 35, y: 5, width: 70, height: 30))
}
```

Local Reasoning

Easier to understand, easier to test

```
// View Layout

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

```
// 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 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 }
}
```

```
// 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 {}
```

```
// 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 {}
```

```
// 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 {}
```

```
// Layout
```

```
struct DecoratingLayout {
    var content: Layout      ←———————— UIView
    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 {}
```

```
// 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 = ...
    }
}

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 {}
```

Generic Types

More control over types

Can be optimized more at compile time

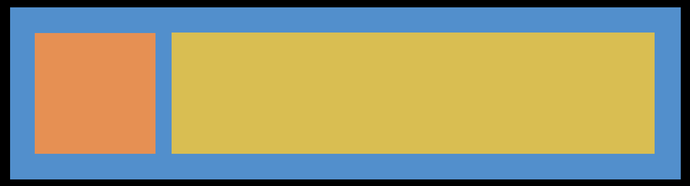
Generic Types

More control over types

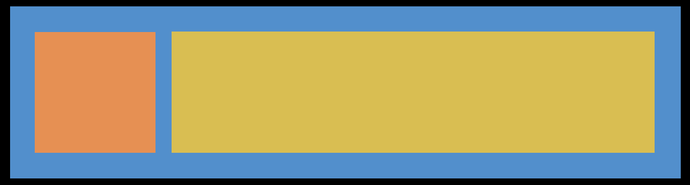
Can be optimized more at compile time

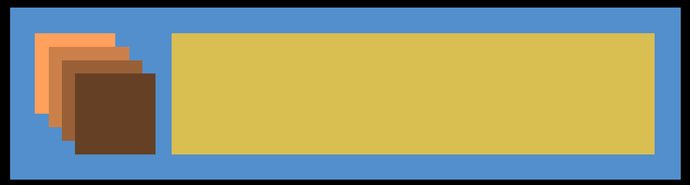
Understanding Swift Performance	Mission	Friday 11:00AM

Sharing Code

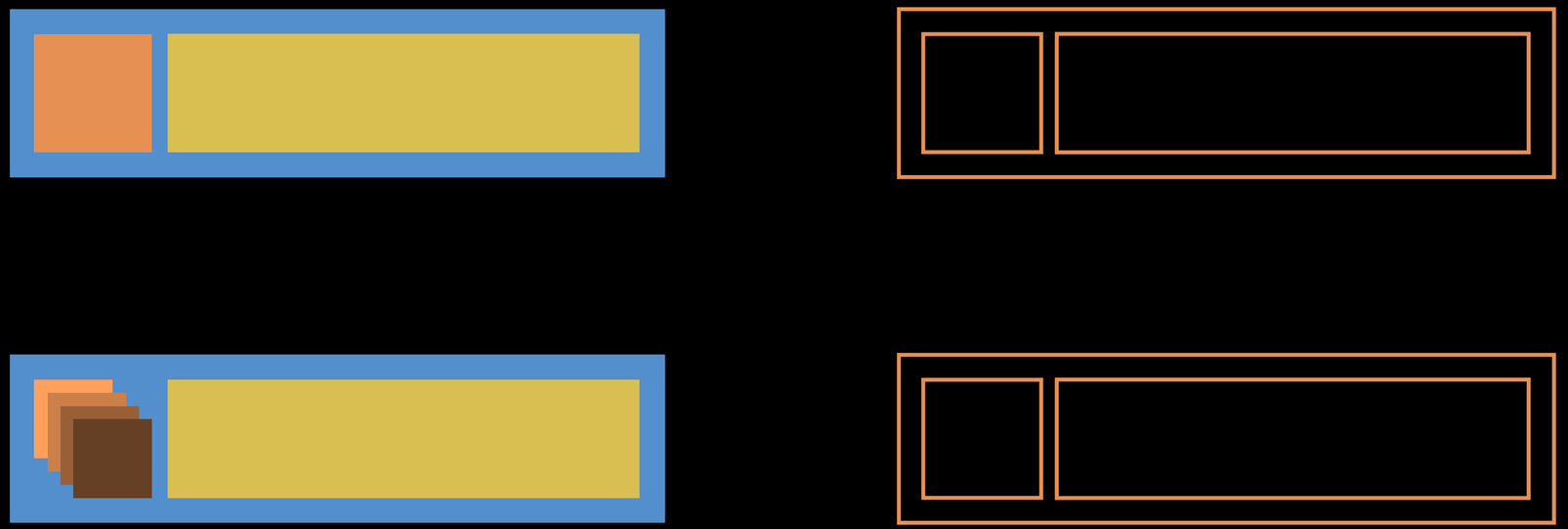


Sharing Code





Sharing Code



// Inheritance

```
class DreamCell: DecoratingLayoutCell {
    // MARK: Properties

    static let reuseIdentifier = "\(DreamCell.self)"

    var content = UILabel()
    var accessories = [UIImageView]()

    var dream: Dream! {
        didSet {
            // Update the UI when the `dream` changes.
            accessories = (0..<dream.numberOfCreatures).map { _ in
                let imageView = UIImageView(image: dream.creature.image)
                imageView.contentMode = .scaleAspectFit
                return imageView
            }
            content.text = dream.description
            for subview in contentView.subviews {
                subview.removeFromSuperview()
            }
            addSubviews()
            setNeedsLayout()
        }
    }

    // MARK: Initialization

    override init(style: UITableViewCellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)

        addSubviews()
    }

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

    // MARK: Layout

    private func addSubviews() {
        let multiPaneLayout = MultiPaneLayout(content: content, accessories: accessories)
        for view in multiPaneLayout.contents {
            contentView.addSubview(view)
        }
    }

    override func layoutSubviews() {
        super.layoutSubviews()

        /*
            This is the intersection between the UIKit view code and this sample's
            value based layout system.
        */
        var multiPaneLayout = MultiPaneLayout(content: content, accessories: accessories)
        multiPaneLayout.layout(in: contentView.bounds)
    }
}
```

// Inheritance

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

   override init(style: UITableViewCellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)

        decoration.contentMode = .scaleAspectFit

        contentView.addSubview(content)
        contentView.addSubview(decoration)
        setNeedsLayout()
    }

    override func layoutSubviews() {
        super.layoutSubviews()

        content.frame = contentLayoutFrame
        decoration.frame = decorationLayoutFrame
    }

    var contentLayoutFrame: CGRect {
        var dstRect = contentView.bounds
        dstRect.origin.x = rect.size.width / 3
        dstRect.size.width += 2/3
        return dstRect
    }

    var decorationLayoutFrame: CGRect {
        var dstRect = contentView.bounds
        dstRect.size.width /= 3
        return dstRect
    }

    required init?(coder aDecoder: NSCoder) {
        fatalError("\(#function) has not been implemented")
    }
}
```

```
class DreamCell: DecoratingLayoutCell {
    // MARK: Properties

    static let reuseIdentifier = "\(DreamCell.self)"

    var content = UILabel()
    var accessories = [UIImageView]()

    var dream: Dream! {
        didSet {
            // Update the UI when the `dream` changes.
            accessories = (0..<dream.numberOfCreatures).map { _ in
                let imageView = UIImageView(image: dream.creature.image)
                imageView.contentMode = .scaleAspectFit
                return imageView
            }
            content.text = dream.description
            for subview in contentView.subviews {
                subview.removeFromSuperview()
            }
            addSubviews()
            setNeedsLayout()
        }
    }

    // MARK: Initialization

    override init(style: UITableViewCellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)

        addSubviews()
    }

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

    // MARK: Layout

    private func addSubviews() {
        let multiPaneLayout = MultiPaneLayout(content: content, accessories: accessories)
        for view in multiPaneLayout.contents {
            contentView.addSubview(view)
        }
    }

    override func layoutSubviews() {
        super.layoutSubviews()

        /*
            This is the intersection between the UIKit view code and this sample's
            value based layout system.
        */
        var multiPaneLayout = MultiPaneLayout(content: content, accessories: accessories)
        multiPaneLayout.layout(in: contentView.bounds)
    }
}
```

// Inheritance

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

    override init(style: UITableViewCellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)

        decoration.contentMode = .scaleAspectFit

        contentView.addSubview(content)
        contentView.addSubview(decoration)
        setNeedsLayout()
    }

    override func layoutSubviews() {
        super.layoutSubviews()

        content.frame = contentLayoutFrame
        decoration.frame = decorationLayoutFrame
    }

    var contentLayoutFrame: CGRect {
        var distRect = contentView.bounds
        distRect.origin.x = rect.size.width / 3
        distRect.size.width *= 2/3
        return distRect
    }

    var decorationLayoutFrame: CGRect {
        var distRect = contentView.bounds
        distRect.size.width /= 3
        return distRect
    }

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


class DreamCell: DecoratingLayoutCell {
    // MARK: Properties

    static let reuseIdentifier = "\(DreamCell.self)"

    var content = UILabel()
    var accessories = [UIImageView]()

    var dream: Dream! {
        didSet {
            // Update the UI when the `dream` changes.
            accessories = (0..<dream.numberOfCreatures).map { _ in
                let imageView = UIImageView(image: dream.creature.image)
                imageView.contentMode = .scaleAspectFit
                return imageView
            }
            content.text = dream.description
            for subview in contentView.subviews {
                subview.removeFromSuperview()
            }
            addSubviews()
            setNeedsLayout()
        }
    }

    // MARK: Initialization

    override init(style: UITableViewCellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)

        addSubviews()
    }

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

    // MARK: Layout

    private func addSubviews() {
        let multiPaneLayout = MultiPaneLayout(content: content, accessories: accessories)
        for view in multiPaneLayout.contents {
            contentView.addSubview(view)
        }
    }

    override func layoutSubviews() {
        super.layoutSubviews()

        /*
            This is the intersection between the UIKit view code and this sample's
            value based layout system.
        */
        var multiPaneLayout = MultiPaneLayout(content: content, accessories: accessories)
        multiPaneLayout.layout(in: contentView.bounds)
    }
}


class CreatureCell: DreamCell {
    // MARK: Properties

    static let reuseIdentifier = "\(CreatureCell.self)"

    private var content = UILabel()
    private var decoration = UIImageView()

    var creature: Dream.Creature! {
        didSet {
            decoration.image = creature.image
        }
    }

    var title = "" {
        didSet {
            content.text = title
        }
    }

    // MARK: Initialization

    override init(style: UITableViewCellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)

        decoration.contentMode = .scaleAspectFit

        // Add our subviews based on the ordering of the decorating layout's contents.
        let decoratingLayout = DecoratingLayout(content: content, decoration: decoration)
        for view in decoratingLayout.contents {
            contentView.addSubview(view)
        }
    }

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

    // MARK: Layout

    override func layoutSubviews() {
        super.layoutSubviews()

        /*
            This is the intersection between the UIKit view code and this sample's
            value based layout system.
        */
        var decoratingLayout = DecoratingLayout(content: content, decoration: decoration)
        decoratingLayout.layout(in: contentView.bounds)
    }
}
```

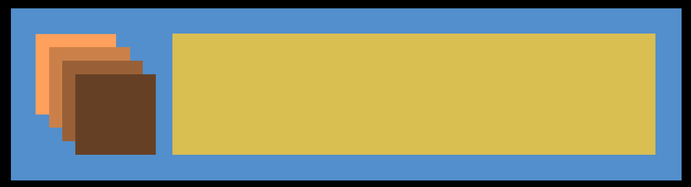
// Inheritance

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

    override init(style: UITableViewCellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)

        decoration.contentMode = .scaleAspectFit

        contentView.addSubview(content)
        contentView.addSubview(decoration)
        setNeedsLayout()
    }

    override func layoutSubviews() {
        super.layoutSubviews()

        content.frame = contentLayoutFrame
        decoration.frame = decoratio        Frame
    }

    var contentLayoutFrame: CGRect {
        var distRect = contentView.bounds
        distRect.origin.x = rect.size.width / 3
        distRect.size.width *= 2/3
        return distRect
    }

    var decorationLayoutFrame: CGRect {
        var distRect = contentView.bounds
        distRect.size.width /= 3
        return distRect
    }

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


class DreamCell: DecoratingLayoutCell {
    // MARK: Properties

    static let reuseIdentifier = "\(DreamCell.self)"

    var content = UILabel()
    var accessories = [UIImageView()]

    var dream: Dream! {
        didSet {
            // Update the UI when the `dream` changes.
            accessories = [0...dream.numberOfCreatures].map { _ in
                let imageView = UIImageView(image: dream.creature.image)
                imageView.contentMode = .scaleAspectFit
                return imageView
            }
            content.text = dream.description
            for subview in contentView.subviews {
                subview.removeFromSuperview()
            }
            addSubviews()
            setNeedsLayout()
        }
    }

    // MARK: Initialization

    override init(style: UITableViewCellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)

        addSubviews()
    }

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

    // MARK: Layout

    private func addSubviews() {
        let multiPaneLayout = MultiPaneLayout(content: content, accessories: accessories)
        for view in multiPaneLayout.contents {
            contentView.addSubview(view)
        }
    }

    override func layoutSubviews() {
        super.layoutSubviews()

        /*
            This is the intersection between the UIKit view code and this sample's
            value based layout system.
        */
        var multiPaneLayout = MultiPaneLayout(content: content, accessories: accessories)
        multiPaneLayout.layout(in: contentView.bounds)
    }
}


class CreatureCell: DreamCell {
    // MARK: Properties

    static let reuseIdentifier = "\(CreatureCell.self)"

    private var content = UILabel()
    private var decoration = UIImageView()

    var creature: Dream.Creature! {
        didSet {
            decoration.image = creature.image
        }
    }

    var title = "" {
        didSet {
            content.text = title
        }
    }

    // MARK: Initialization

    override init(style: UITableViewCellStyle, reuseIdentifier: String?) {
        super.init(style: style, reuseIdentifier: reuseIdentifier)

        decoration.contentMode = .scaleAspectFit

        // Add our subviews based on the ordering of the decorating layout's contents.
        let decoratingLayout = DecoratingLayout(content: content, decoration: decoration)
        for view in decoratingLayout.contents {
            contentView.addSubview(view)
        }
    }

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

    // MARK: Layout

    override func layoutSubviews() {
        super.layoutSubviews()

        /*
            This is the intersection between the UIKit view code and this sample's
            value based layout system.
        */
        var decoratingLayout = DecoratingLayout(content: content, decoration: decoration)
        decoratingLayout.layout(in: contentView.bounds)
    }
}
```

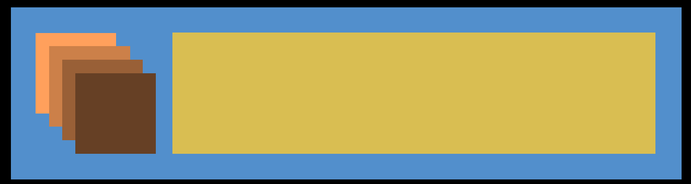
Composition

Share code without reducing local reasoning

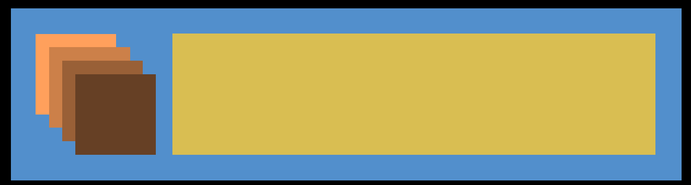
Composition of Views

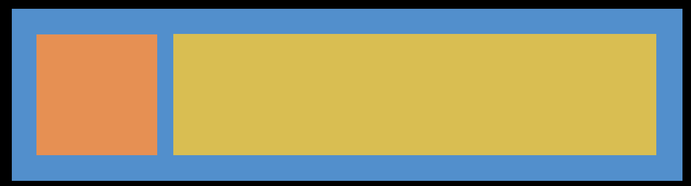


Composition of Views



Composition of Views





Composition of Views

Classes instances are expensive!

Composition of ~~Views~~

Classes instances are expensive!

Composition of ~~Views~~ Values!

Classes instances are expensive!

Composition of ~~Views~~ Values!

Classes instances are expensive!

Structs are cheap

Composition of ~~Views~~ Values!

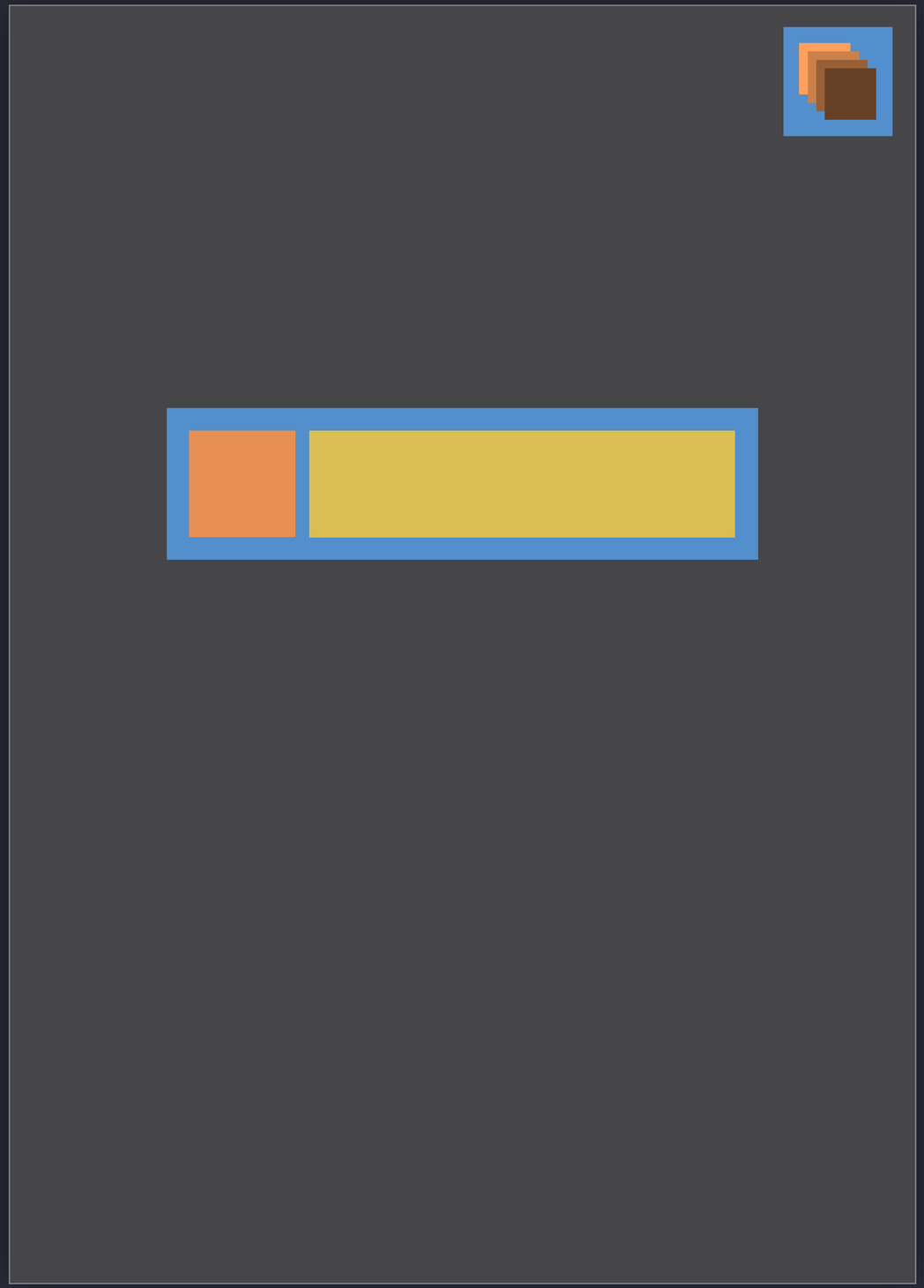
Classes instances are expensive!

Structs are cheap

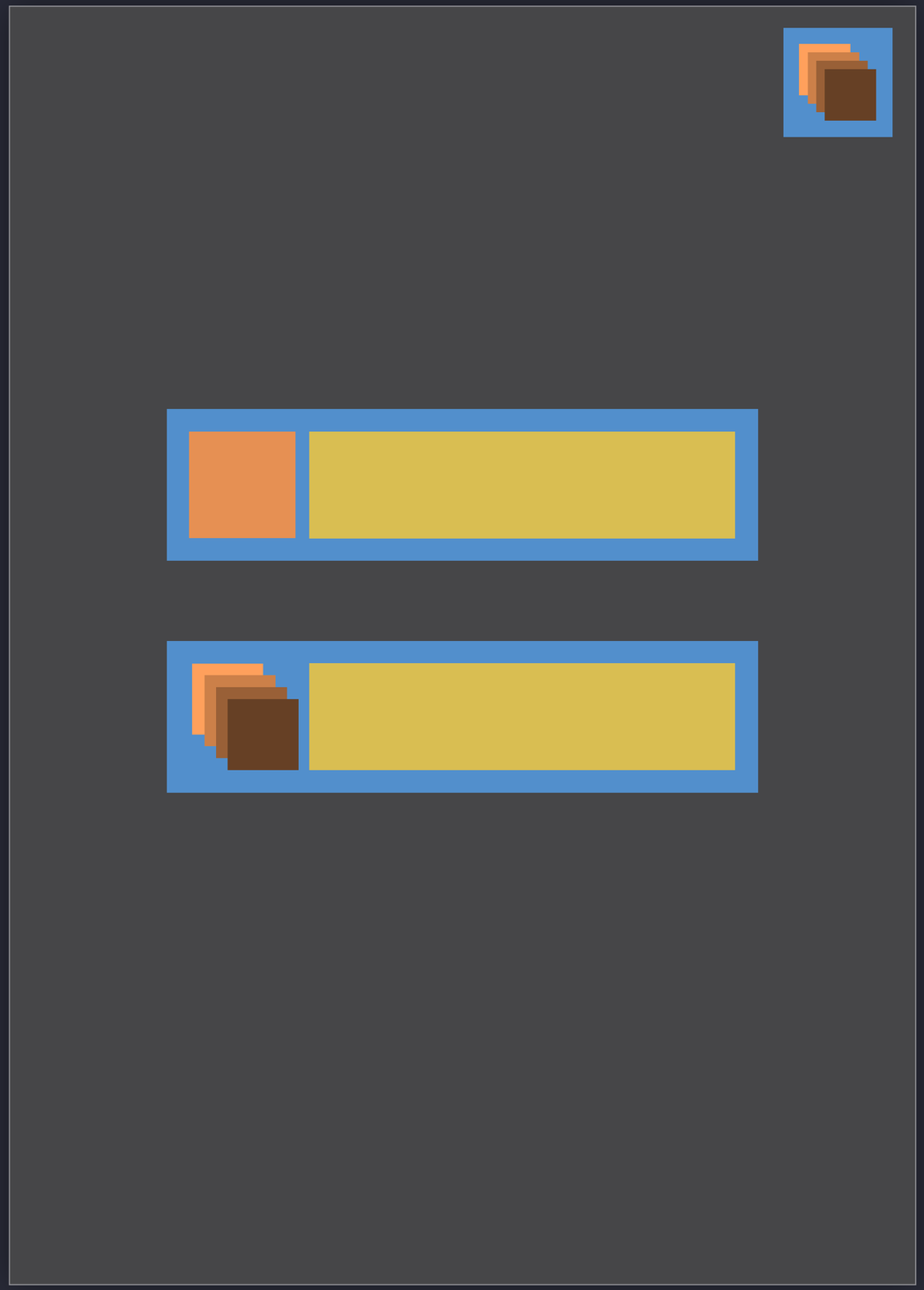
Composition is better with value semantics

```
// 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 = ...
   }
}
```



```
// Composition of Values

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

```
// Composition of Values

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 { ... }
```

```
// 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

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
```

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

```
// Composition of Values
```

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

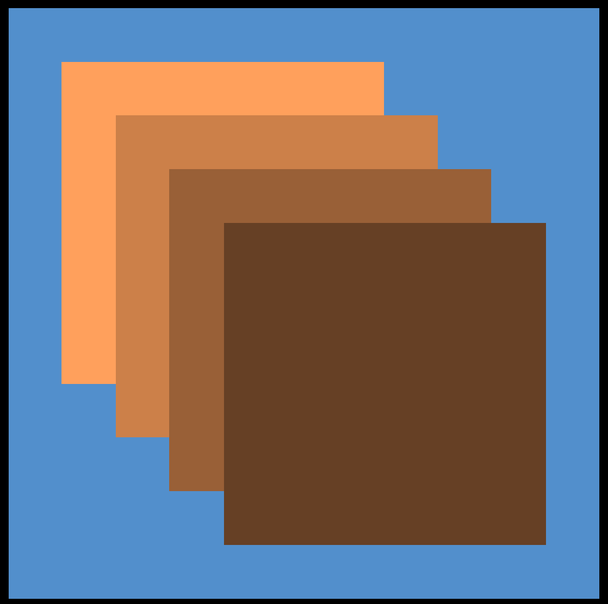
```
// Composition of Values
```

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

```
// Composition of Values
```

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

Contents



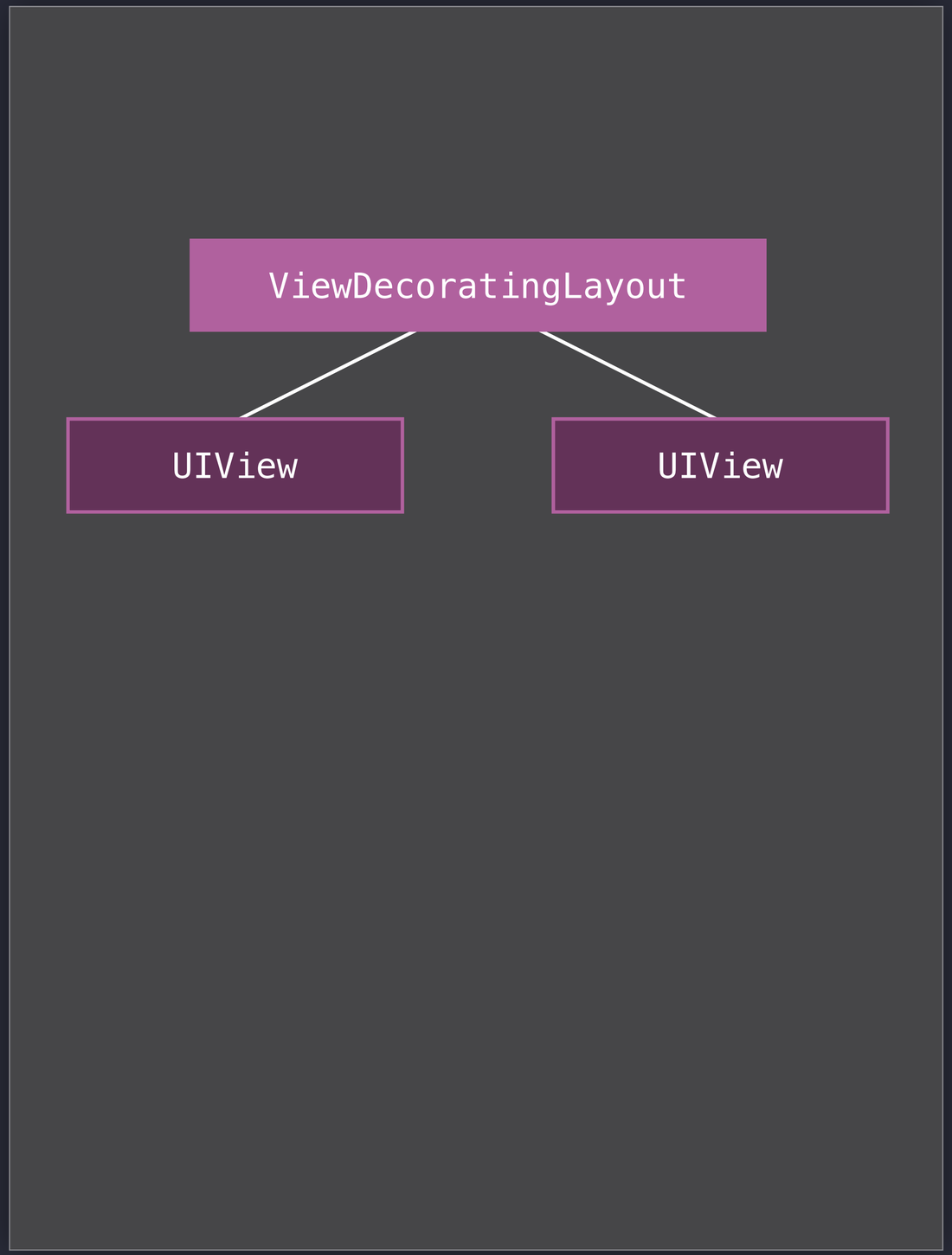
```
// Contents

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

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

```swift
// Contents

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

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

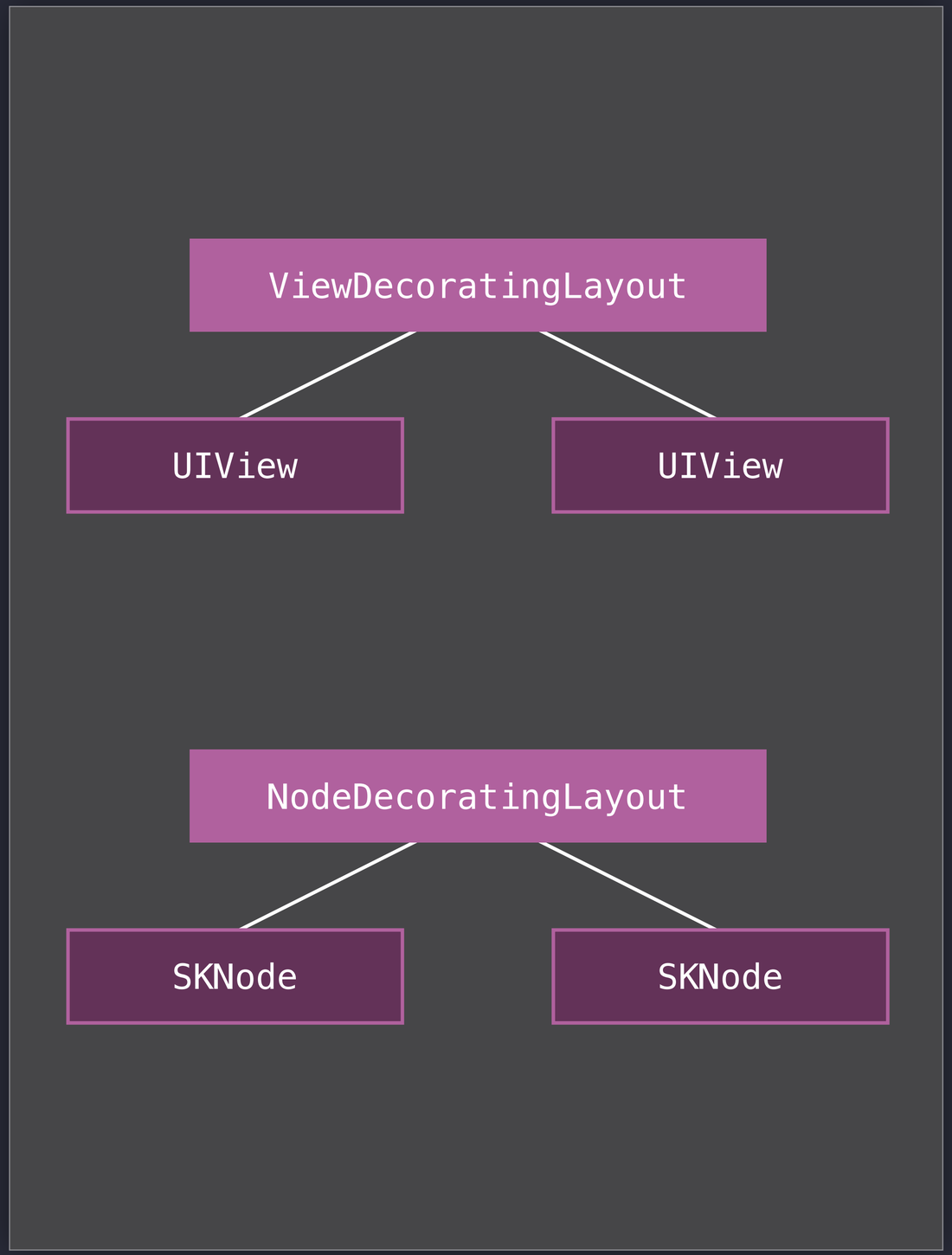
UIView and SKNode

```swift
// Associated Type

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

```swift
// Associated Type

struct ViewDecoratingLayout : Layout {
    ...

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

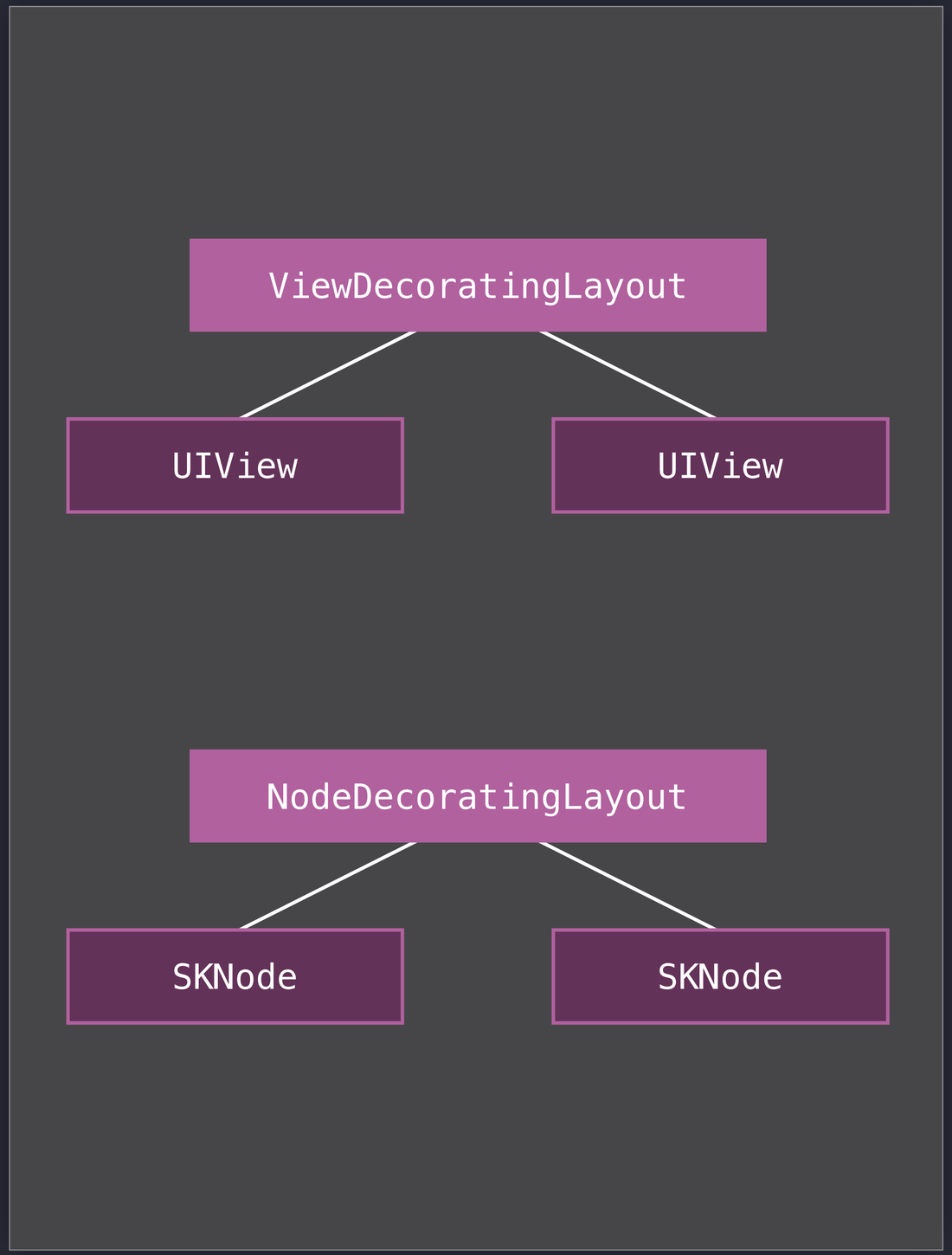


```swift
// Associated Type

struct ViewDecoratingLayout : Layout {
    ...

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

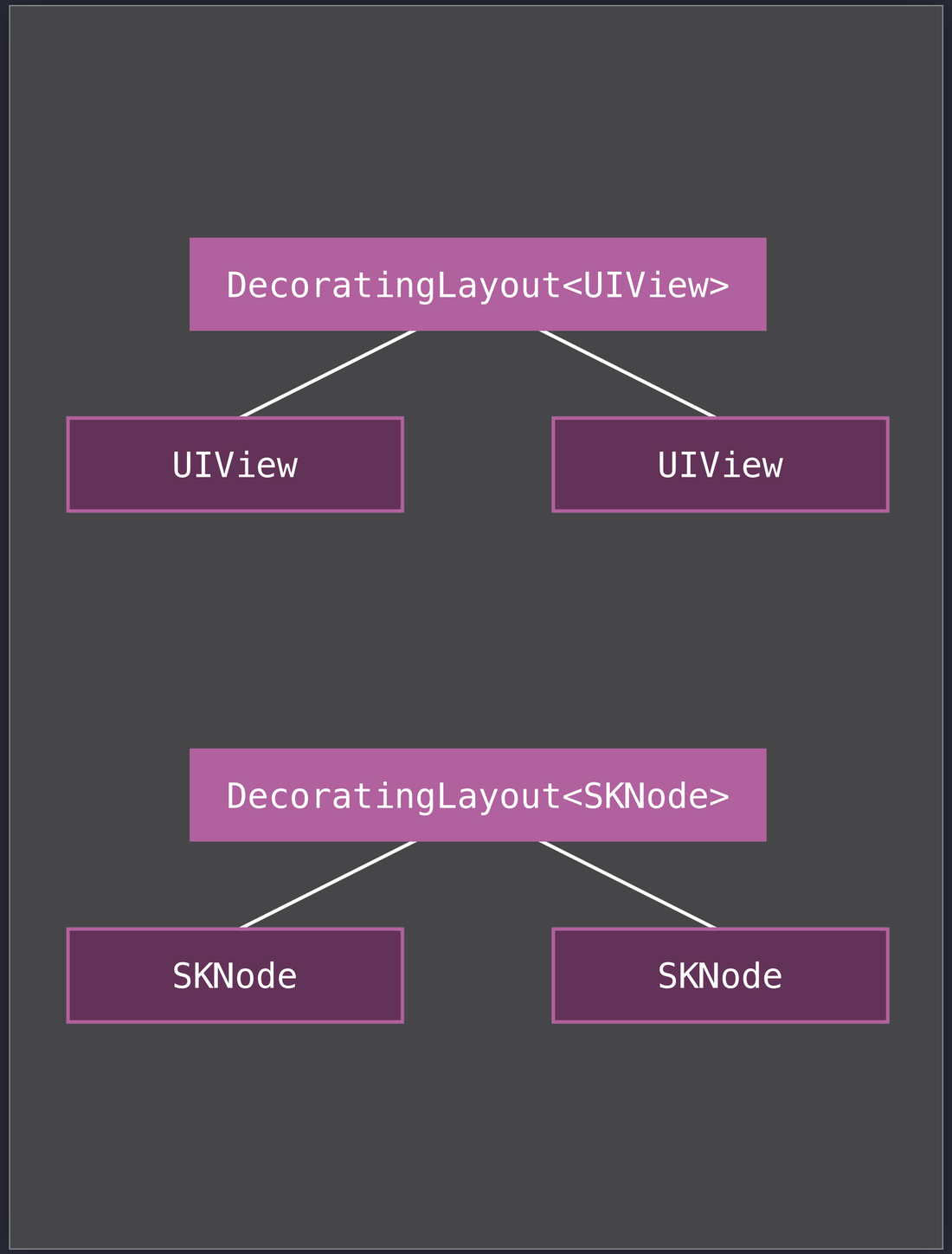



```swift
struct NodeDecoratingLayout : Layout {
    ...

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

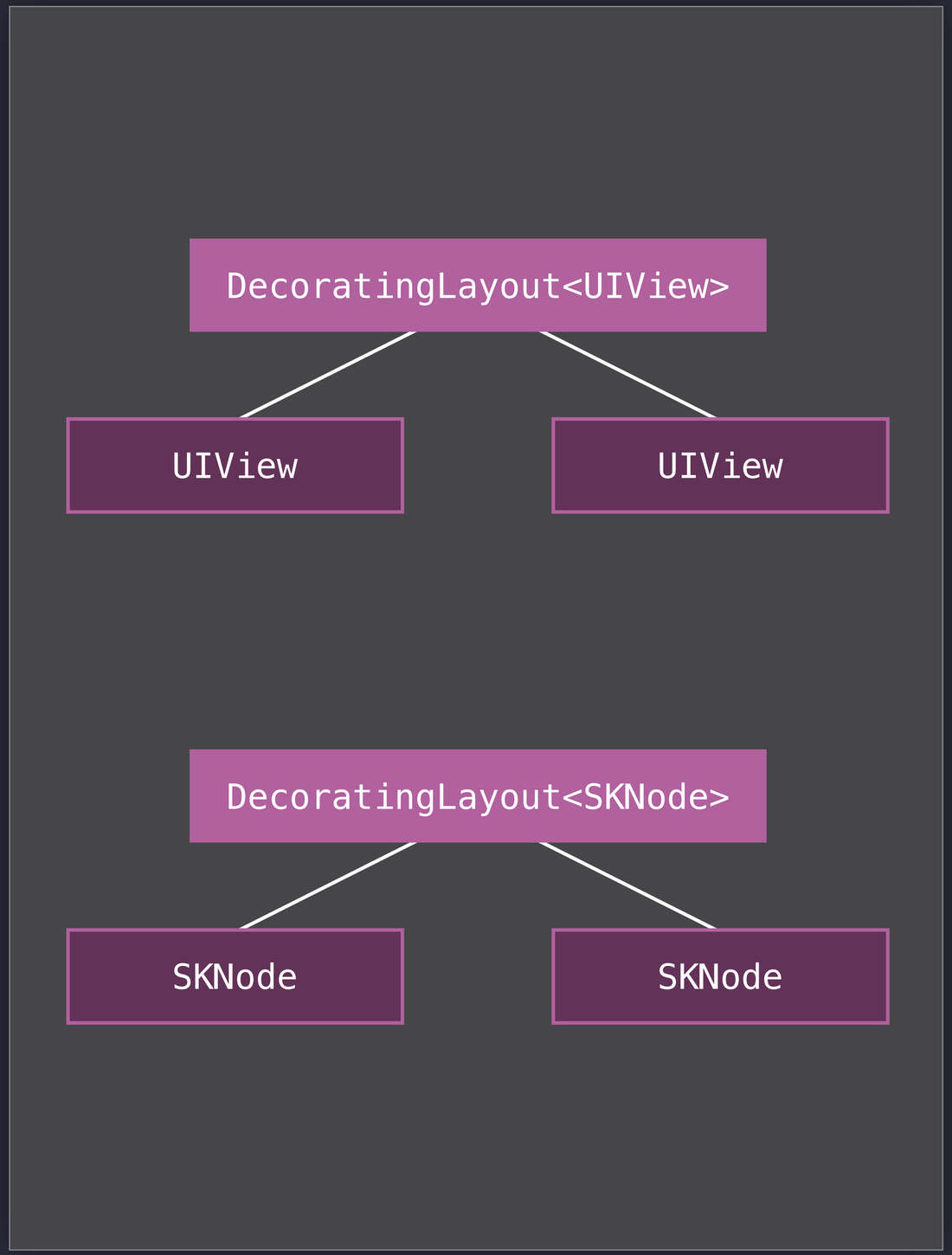
```swift
// Associated Type

struct NodeDecoratingLayout : Layout {
   ...

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



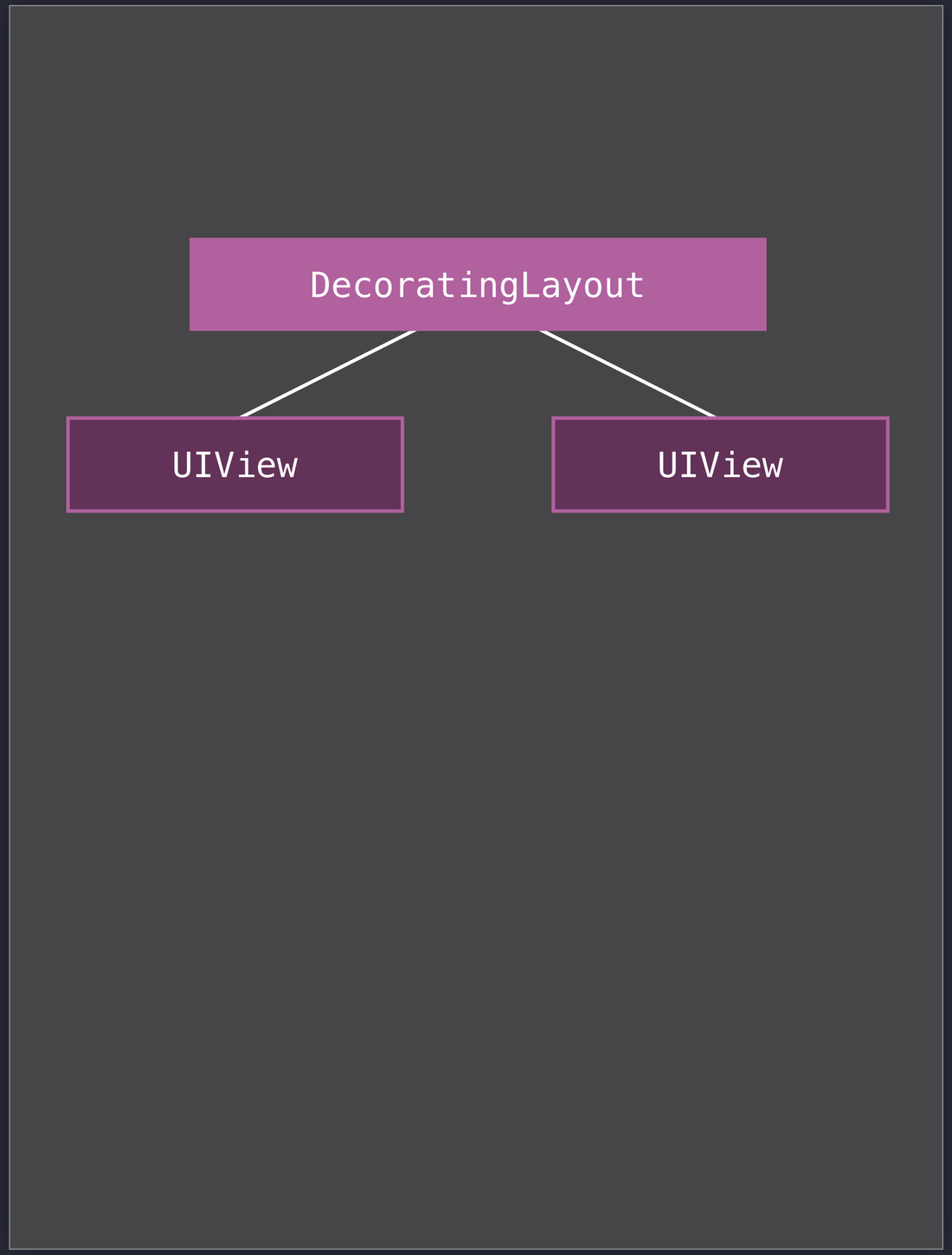
```
// Associated Type

struct DecoratingLayout<Child : Layout> : Layout {
   ...

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



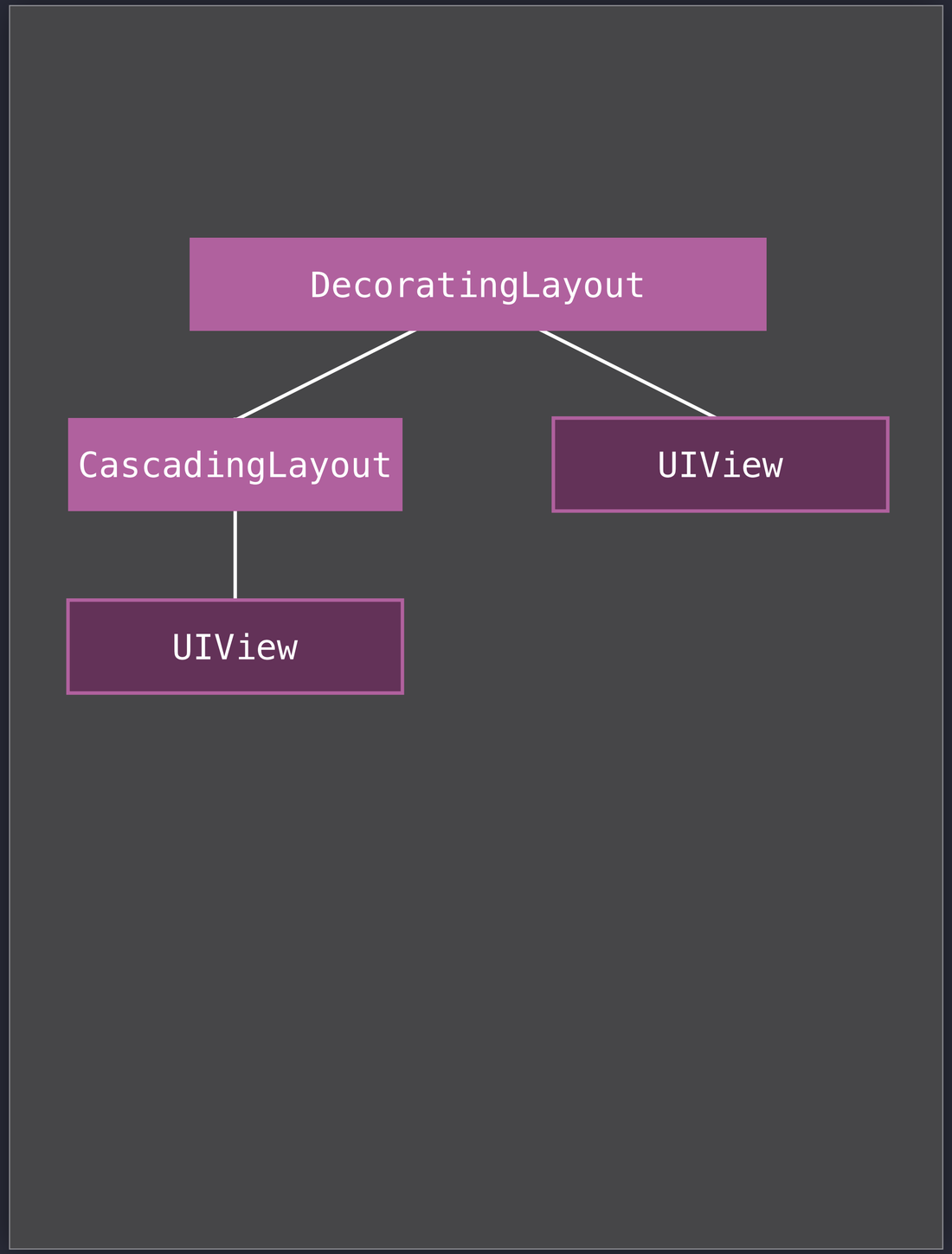
```swift
// Associated Type

struct DecoratingLayout<Child : Layout> : Layout {
    ...

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



```
// Associated Type

struct DecoratingLayout<Child : Layout> : Layout {

    var content: Child
    var decoration: Child

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



```
// Associated Type

struct DecoratingLayout<Child : Layout> : Layout {

    var content: Child
    var decoration: Child

    mutating func layout(in rect: CGRect)
    typealias Content = Child.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 }
}
```

```
// 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 }
}
```

```
// 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 }
}
```

```swift
// Layout

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

   associatedtype Content
   var contents: [Content] { get }
}
```

```swift
// Testing

func testLayout() {
    let child1 = UIView()
    let child2 = UIView()

    var layout = DecoratingLayout(content: child1, decoration: child2)
    layout.layout(in: CGRect(x: 0, y: 0, width: 120, height: 40))

    XCTAssertEqual(layout.contents[0].frame, CGRect(x: 0, y: 5, width: 35, height: 30))
    XCTAssertEqual(layout.contents[1].frame, CGRect(x: 35, y: 5, width: 70, height: 30))
}
```

```
// Testing

func testLayout() {
    let child1 = TestLayout()
    let child2 = TestLayout()

    var layout = DecoratingLayout(content: child1, decoration: child2)
    layout.layout(in: CGRect(x: 0, y: 0, width: 120, height: 40))

    XCTAssertEqual(layout.contents[0].frame, CGRect(x: 0, y: 5, width: 35, height: 30))
    XCTAssertEqual(layout.contents[1].frame, CGRect(x: 35, y: 5, width: 70, height: 30))
}

    struct TestLayout : Layout {
        var frame: CGRect
        ...
    }
```

```
// Testing

func testLayout() {
    let child1 = TestLayout()
    let child2 = TestLayout()

    var layout = DecoratingLayout(content: child1, decoration: child2)
    layout.layout(in: CGRect(x: 0, y: 0, width: 120, height: 40))

    XCTAssertEqual(layout.contents[0].frame, CGRect(x: 0, y: 5, width: 35, height: 30))
    XCTAssertEqual(layout.contents[1].frame, CGRect(x: 35, y: 5, width: 70, height: 30))
}

struct TestLayout : Layout {
    var frame: CGRect
    ...
}
```

Techniques

Techniques

Local reasoning with value types

Techniques

Local reasoning with value types

Generic types for fast, safe polymorphism

Techniques

Local reasoning with value types

Generic types for fast, safe polymorphism

Composition of values

Controller

Undo

Alex Migicovsky Swift Compiler Typo Engineer

Controller

Undo

Alex Migicovsky Swift Compiler Typo Engineer

9:41 AM 100%

Duplicate **Lucid Dreams**

FAVORITE CREATURE

Pink unicorn

DREAMS

Dream 1

Dream 2

Dream 3

9:41 AM 100%

Duplicate **Lucid Dreams**

FAVORITE CREATURE

Pink unicorn

DREAMS

Dream 1

Dream 2

Dream 3

9:41 AM 100%

Cancel **Favorite Creature** Done

Yellow unicorn

Pink unicorn ✓

White unicorn

Crusty

Shark

Dragon

9:41 AM 100%

Cancel **Favorite Creature** Done

Yellow unicorn

Pink unicorn ✓

White unicorn

Crusty

Shark

Dragon

9:41 AM 100%

Cancel **Favorite Creature** Done

Yellow unicorn

Pink unicorn

White unicorn

Crusty

Shark

Dragon

9:41 AM 100%

Cancel **Favorite Creature** Done

Yellow unicorn

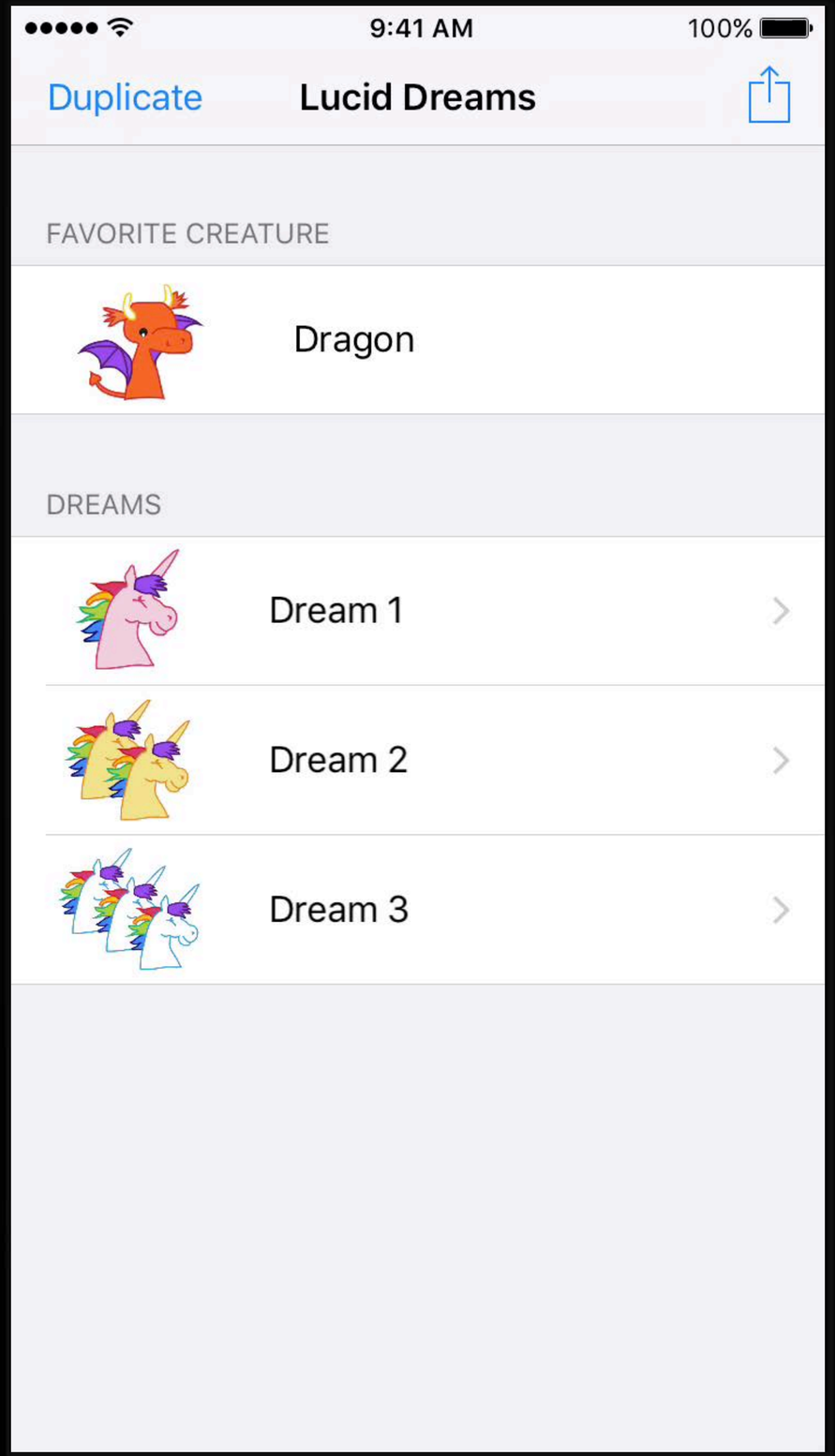
Pink unicorn

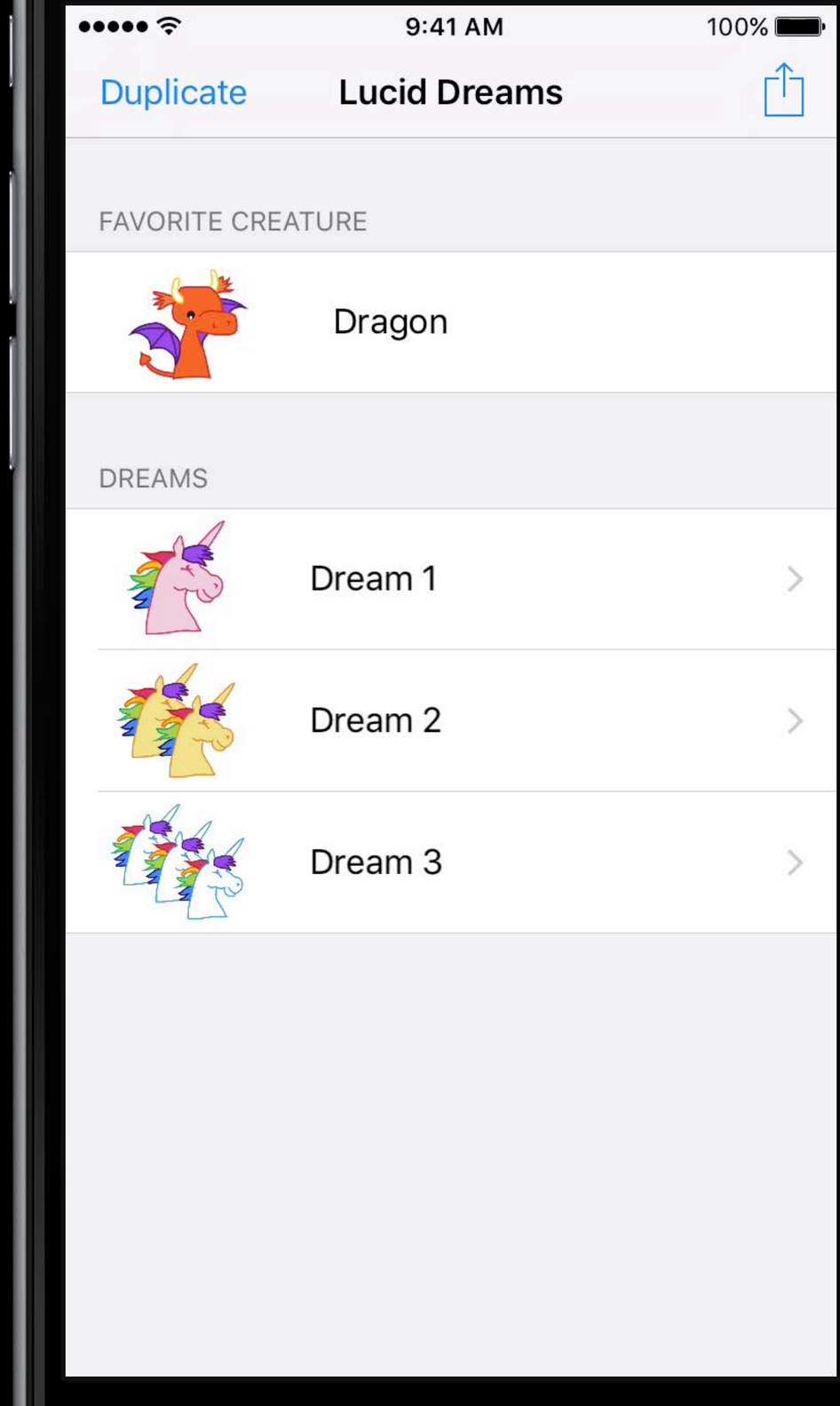
White unicorn

Crusty

Shark

Dragon

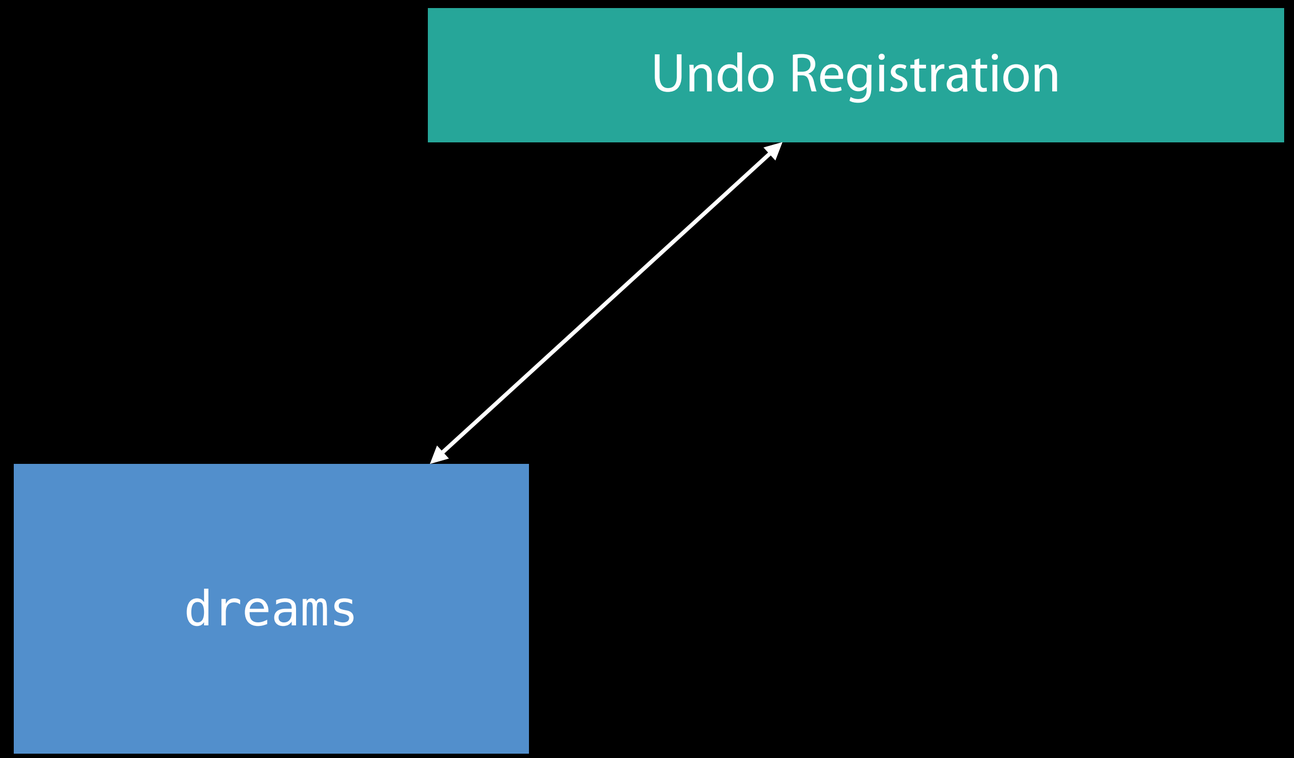

9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Dragon
DREAMS
Dream 1
Dream 2
Dream 3

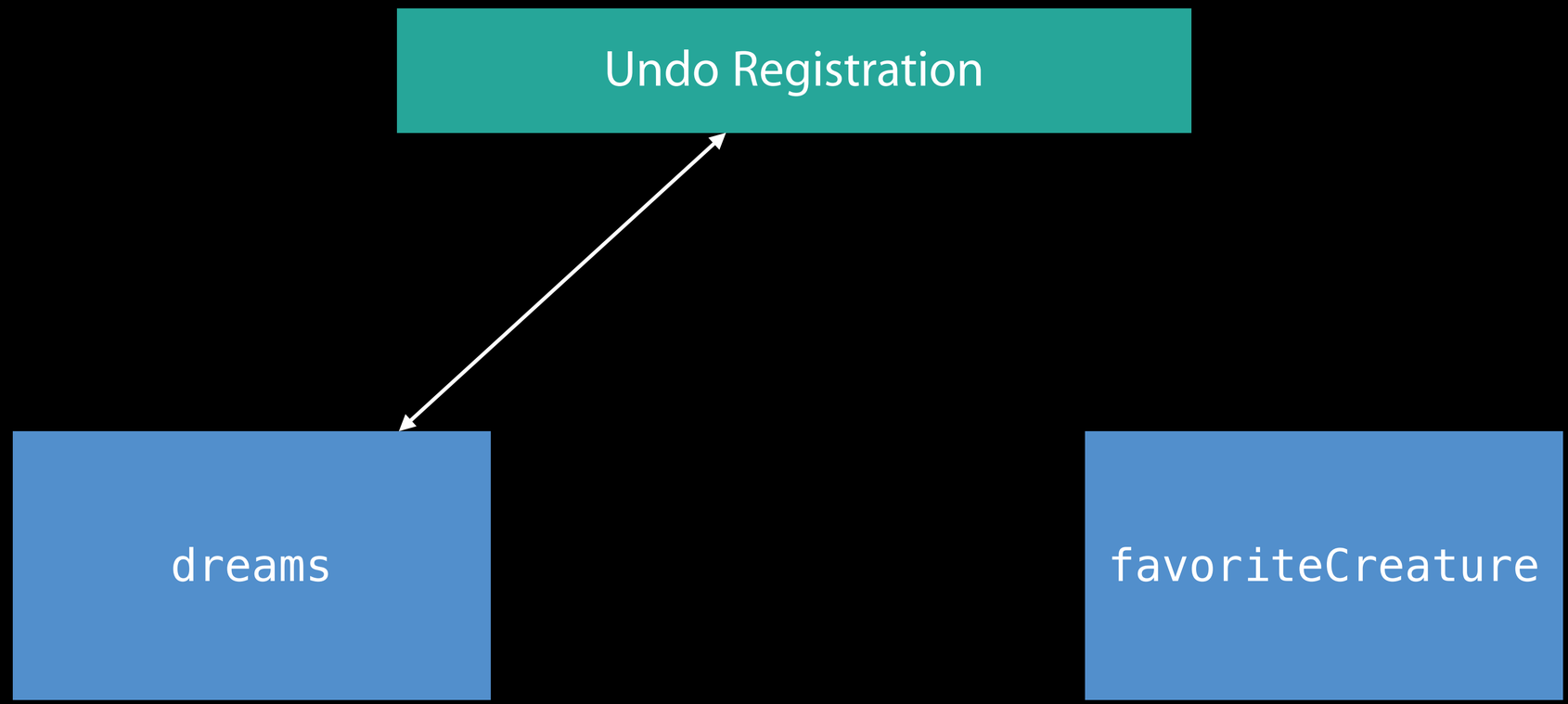

9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Dragon
DREAMS
Dream 1
Dream 2
Dream 3

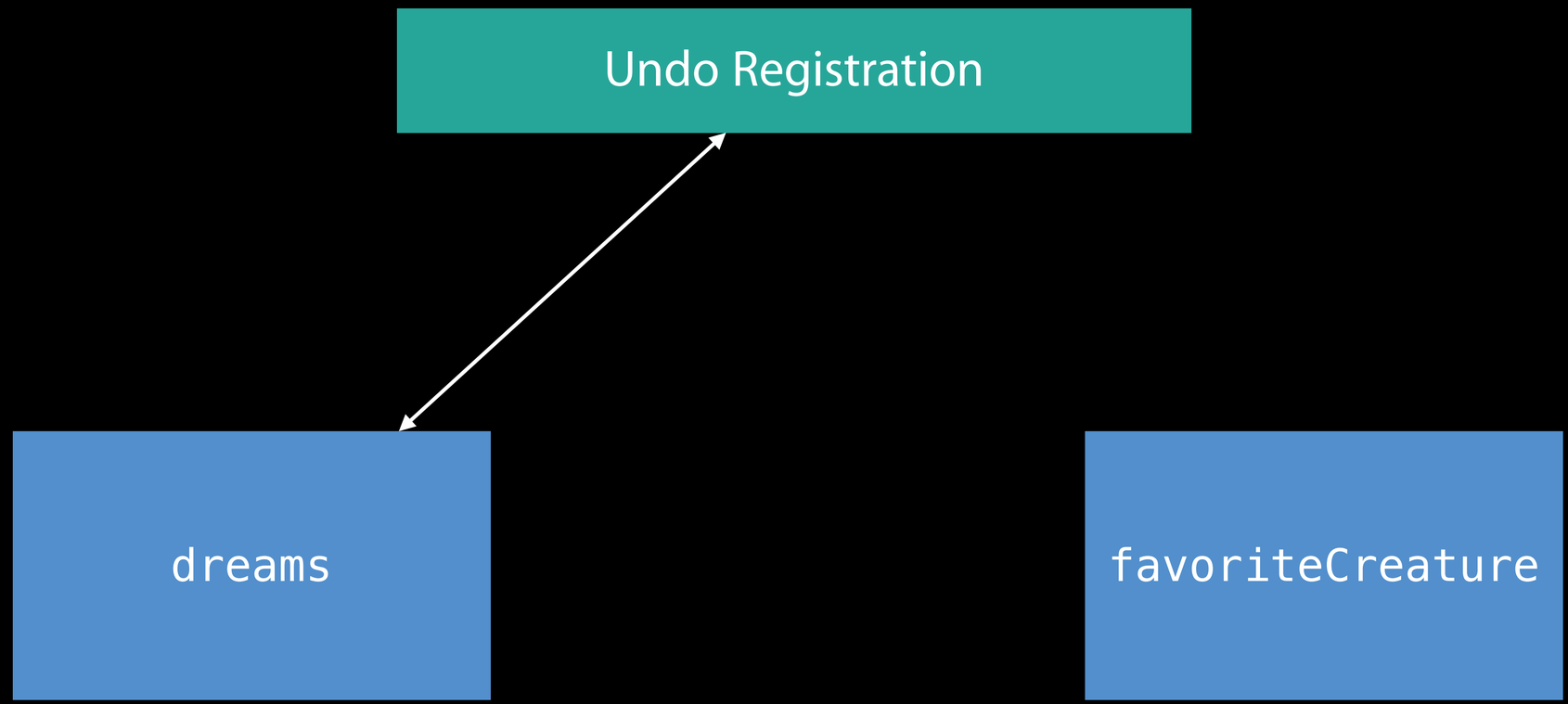
?!@##?

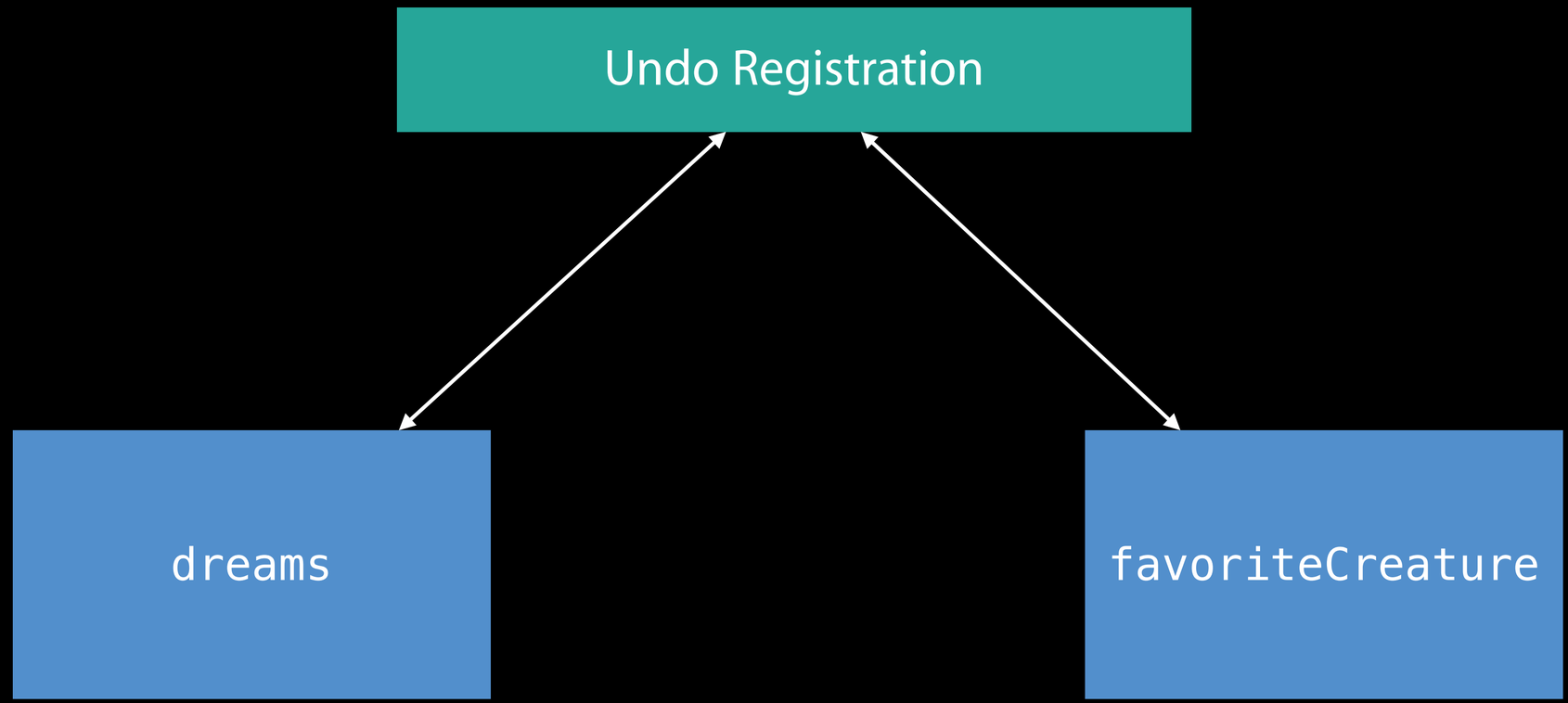
```
// DreamListViewController – Undo Bug

class DreamListViewController : UITableViewController {
    var dreams: [Dream]
    var favoriteCreature: Creature
    ...
}
```

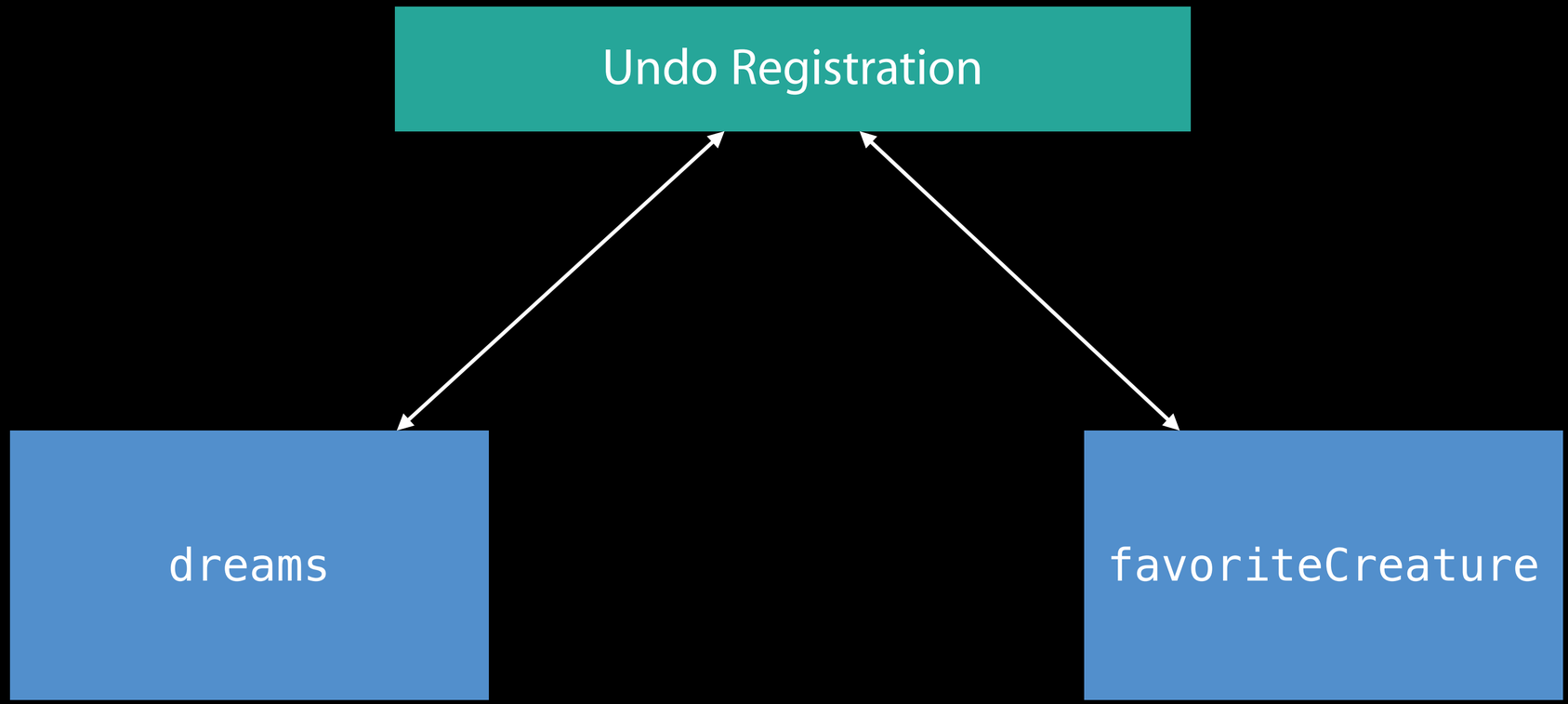
Model properties


Undo Registration
dreams

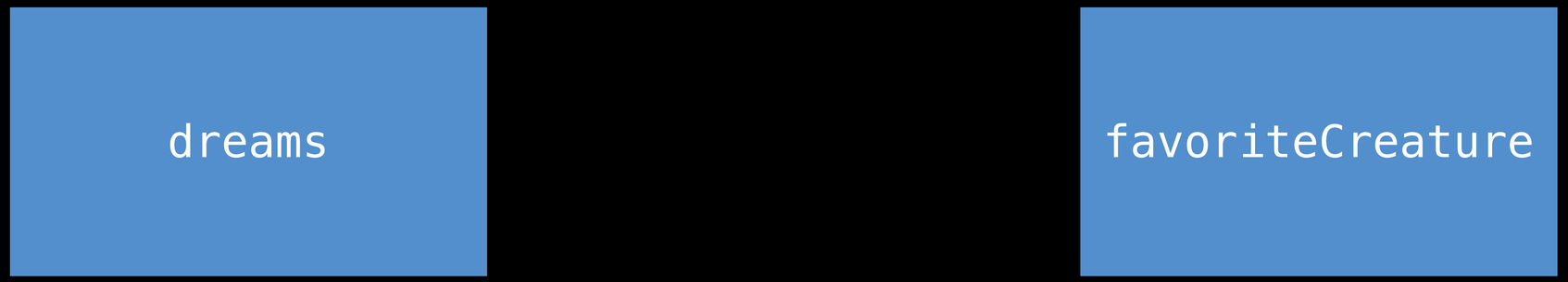

Undo Registration
dreams
favoriteCreature

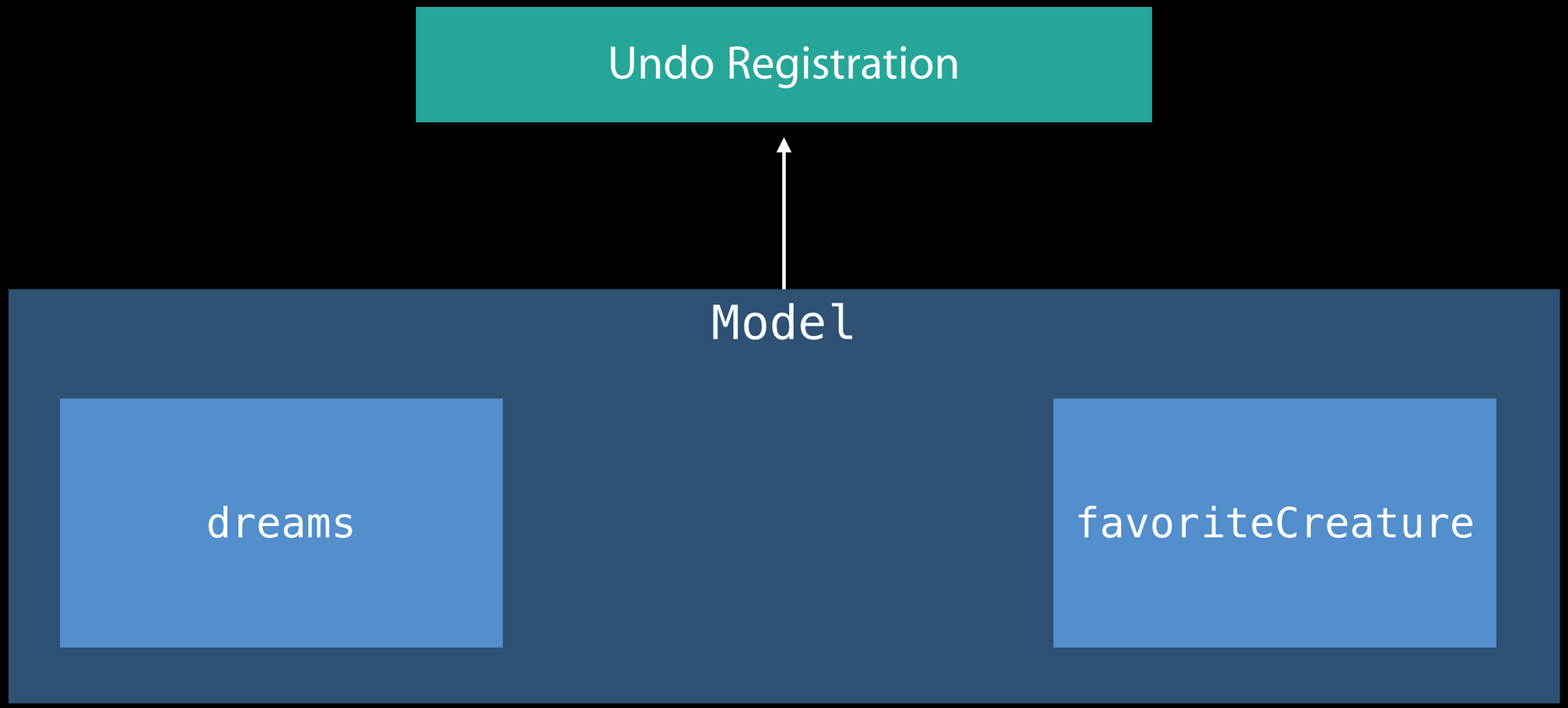

Undo Registration
dreams
favoriteCreature

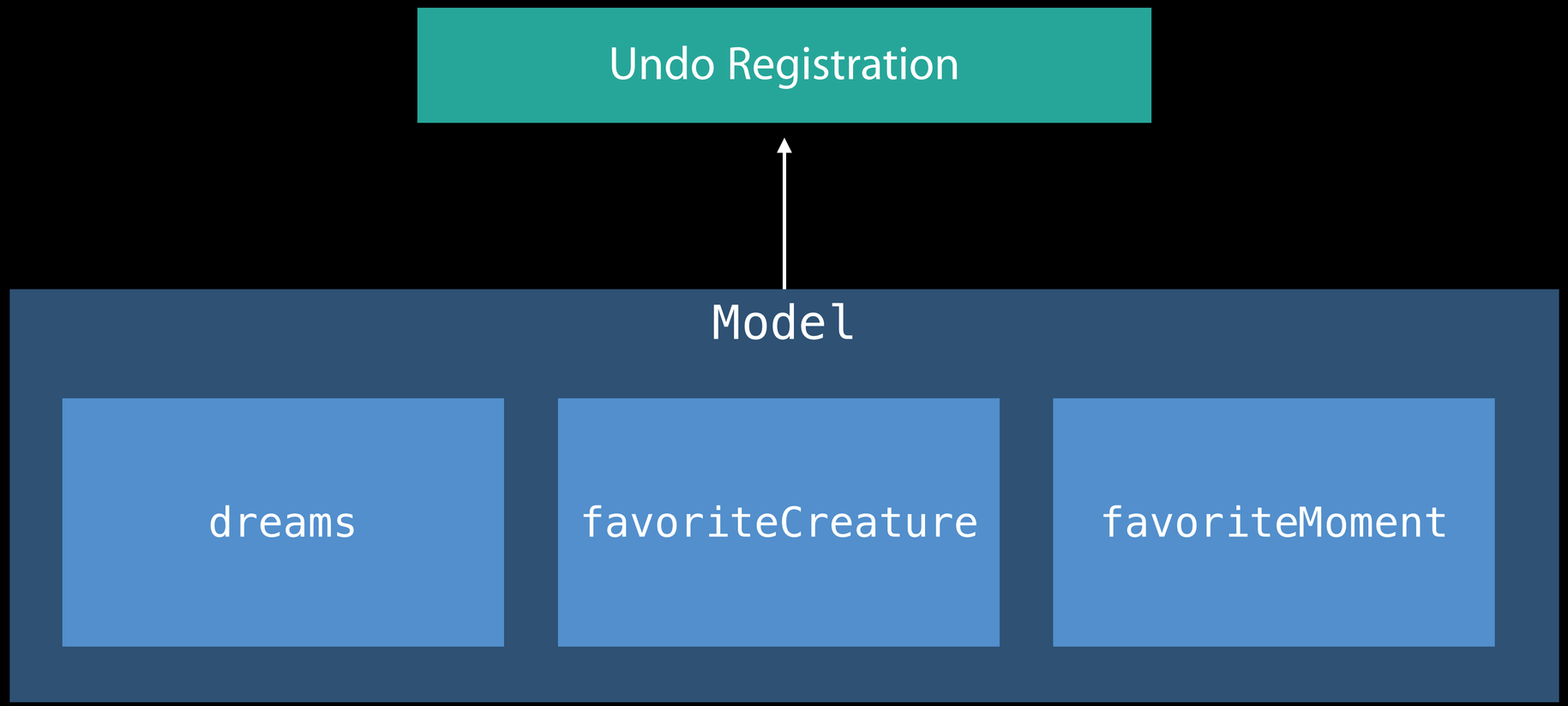

Undo Registration
dreams
favoriteCreature


Undo Registration
dreams
favoriteCreature

Undo Registration




Undo Registration
Model
dreams
favoriteCreature


Undo Registration
Model
dreams
favoriteCreature
favoriteMoment

Undo Registration

Model

dreams

favoriteCreature

favoriteMoment

```
// DreamListViewController – Isolating the Model

class DreamListViewController : UITableViewController {
    var dreams: [Dream]
    var favoriteCreature: Creature
    ...
}

    struct Model : Equatable {


    }
```

```
// DreamListViewController – Isolating the Model

class DreamListViewController : UITableViewController {



    ...
}

    struct Model : Equatable {
        var dreams: [Dream]
        var favoriteCreature: Creature
    }
```

```
// DreamListViewController – Isolating the Model

class DreamListViewController : UITableViewController {
    var model: Model

    ...
}

struct Model : Equatable {
    var dreams: [Dream]
    var favoriteCreature: Creature
}
```

Model

.unicorn(.yellow)

dreams[0]

dreams[1]

UndoManager Stack

dreams.removeLast()
tableView.deleteRows(at: ...)

favoriteCreature = .unicorn(.pink)
tableView.reloadRows(...)

dreams.append(Dream(...))
tableView.insertRows(at: ...)

Model

.unicorn(.yellow)

dreams[0]

dreams[1]

UndoManager Stack

```
dreams.removeLast()
tableView.deleteRows(at: ...)
```

```
favoriteCreature = .unicorn(.pink)
tableView.reloadRows(...)
```

```
dreams.append(Dream(...))
tableView.insertRows(at: ...)
```

Model

.unicorn(.yellow)

dreams[0]

UndoManager Stack

dreams.removeLast()
tableView.deleteRows(at: ...)

favoriteCreature = .unicorn(.pink)
tableView.reloadRows(...)

dreams.append(Dream(...))
tableView.insertRows(at: ...)

Model

.unicorn(.yellow)

dreams[0]

UndoManager Stack

favoriteCreature = .unicorn(.pink)
tableView.reloadRows(...)

dreams.append(Dream(...))
tableView.insertRows(at: ...)

Model

.unicorn(.yellow)

dreams[0]

UndoManager Stack

favoriteCreature = .unicorn(.pink)
tableView.reloadRows(...)

dreams.append(Dream(...))
tableView.insertRows(at: ...)

Model

.unicorn(.pink)

dreams[0]

UndoManager Stack

favoriteCreature = .unicorn(.pink)
tableView.reloadRows(...)

dreams.append(Dream(...))
tableView.insertRows(at: ...)

Model

UndoManager Stack

.unicorn(.pink)

dreams[0]

```
dreams.append(Dream(...))
tableView.insertRows(at: ...)
```

```
Terminating app due to uncaught exception
'NSInternalInconsistencyException', reason: 'Invalid
update: invalid number of rows in section 1. The number
of rows contained in an existing section after the update
(2) must be equal to the number of rows contained in that
section before the update (2), plus or minus the number
of rows inserted or deleted from that section (1
inserted, 0 deleted) and plus or minus the number of
rows moved into or out of that section (0 moved in, 0
moved out).'
```

Model

.unicorn(.yellow)

dreams[0]

dreams[1]

UndoManager Stack

dreams.removeLast()
tableView.deleteRows(at: ...)

favoriteCreature = .unicorn(.pink)
tableView.reloadRows(...)

dreams.append(Dream(...))
tableView.insertRows(at: ...)

```
dreams.removeLast()
tableView.deleteRows(at: ...)
```

```
favoriteCreature = .unicorn(.pink)
tableView.reloadRows(...)
```

```
dreams.append(Dream(...))
tableView.insertRows(at: ...)
```

```
dreams.removeLast()
tableView.deleteRows(at: ...)
```

```
favoriteCreature = .unicorn(.yellow)
tableView.reloadRows(...)
```

```
favoriteCreature = .unicorn(.white)
tableView.reloadRows(...)
```

```
favoriteCreature = .unicorn(.pink)
tableView.reloadRows(...)
```

```
dreams.insert(Dream(...), at: 5)
tableView.insertRows(at: ...)
```

```
dreams.insert(Dream(...), at: 5)
tableView.insertRows(at: ...)
```

```
dreams.append(Dream(...))
tableView.insertRows(at: ...)
```

```
dreams.removeLast()
tableView.deleteRows(at: ...)
```

```
favoriteCreature = .unicorn(.yellow)
tableView.reloadRows(...)
```

```
favoriteCreature = .unicorn(.white)
tableView.reloadRows(...)
```

```
favoriteCreature = .unicorn(.pink)
tableView.reloadRows(...)
```

```
dreams.insert(Dream(...), at: 5)
tableView.insertRows(at: ...)
```

```
dreams.insert(Dream(...), at: 5)
tableView.insertRows(at: ...)
```

```
dreams.append(Dream(...))
tableView.insertRows(at: ...)
```

Model

.unicorn(.yellow)

dreams[0]

dreams[1]

UndoManager Stack

.unicorn(.yellow)

dreams[0]

.unicorn(.pink)

Model

.unicorn(.yellow)

dreams[0]

dreams[1]

UndoManager Stack

.unicorn(.yellow)

dreams[0]

.unicorn(.pink)

Model

.unicorn(.yellow)

dreams[0]

UndoManager Stack

.unicorn(.pink)

```
// DreamListViewController – Isolating the Model

class DreamListViewController : UITableViewController {
    ...
    func modelDidChange(old: Model, new: Model) {
    }
}
```

```
// DreamListViewController – Isolating the Model

class DreamListViewController : UITableViewController {
    ...
    func modelDidChange(old: Model, new: Model) {
        if old.favoriteCreature != new.favoriteCreature {
            // Reload table view section for favorite creature.
            tableView.reloadSections(...)
        }
    }
}
```

```
// DreamListViewController – Isolating the Model

class DreamListViewController : UITableViewController {
    ...
    func modelDidChange(old: Model, new: Model) {
        if old.favoriteCreature != new.favoriteCreature {
            // Reload table view section for favorite creature.
            tableView.reloadSections(...)
        }
        ...
    }
}
```

```
// DreamListViewController – Isolating the Model

class DreamListViewController : UITableViewController {
    ...
    func modelDidChange(old: Model, new: Model) {
        if old.favoriteCreature != new.favoriteCreature {
            // Reload table view section for favorite creature.
            tableView.reloadSections(...)
        }

        ...

        undoManager?.registerUndo(withTarget: self, handler: { target in
            target.model = old
        })
    }
}
```

Benefits

Single code path

Benefits

Single code path

- Better local reasoning

Benefits

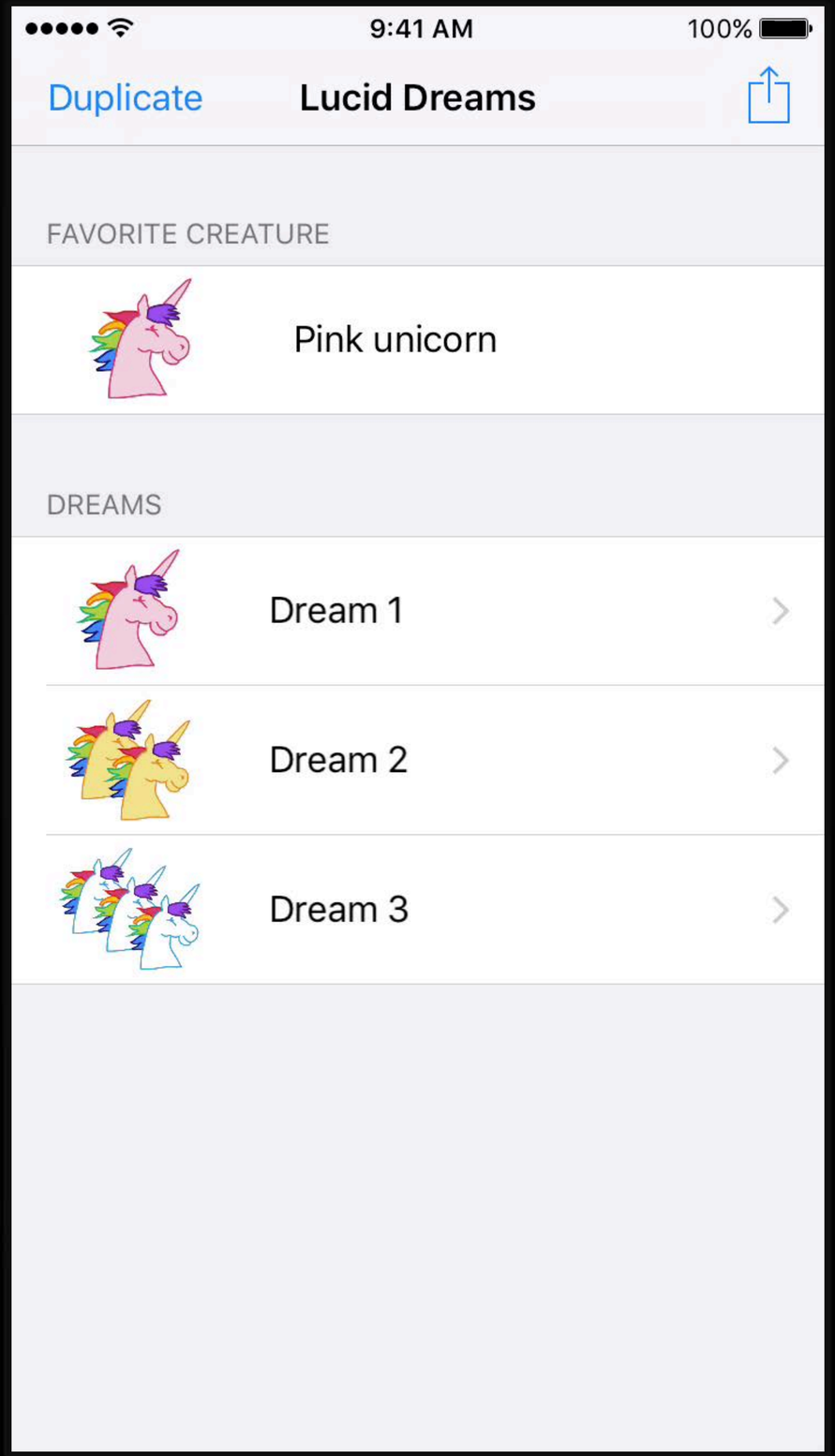
Single code path

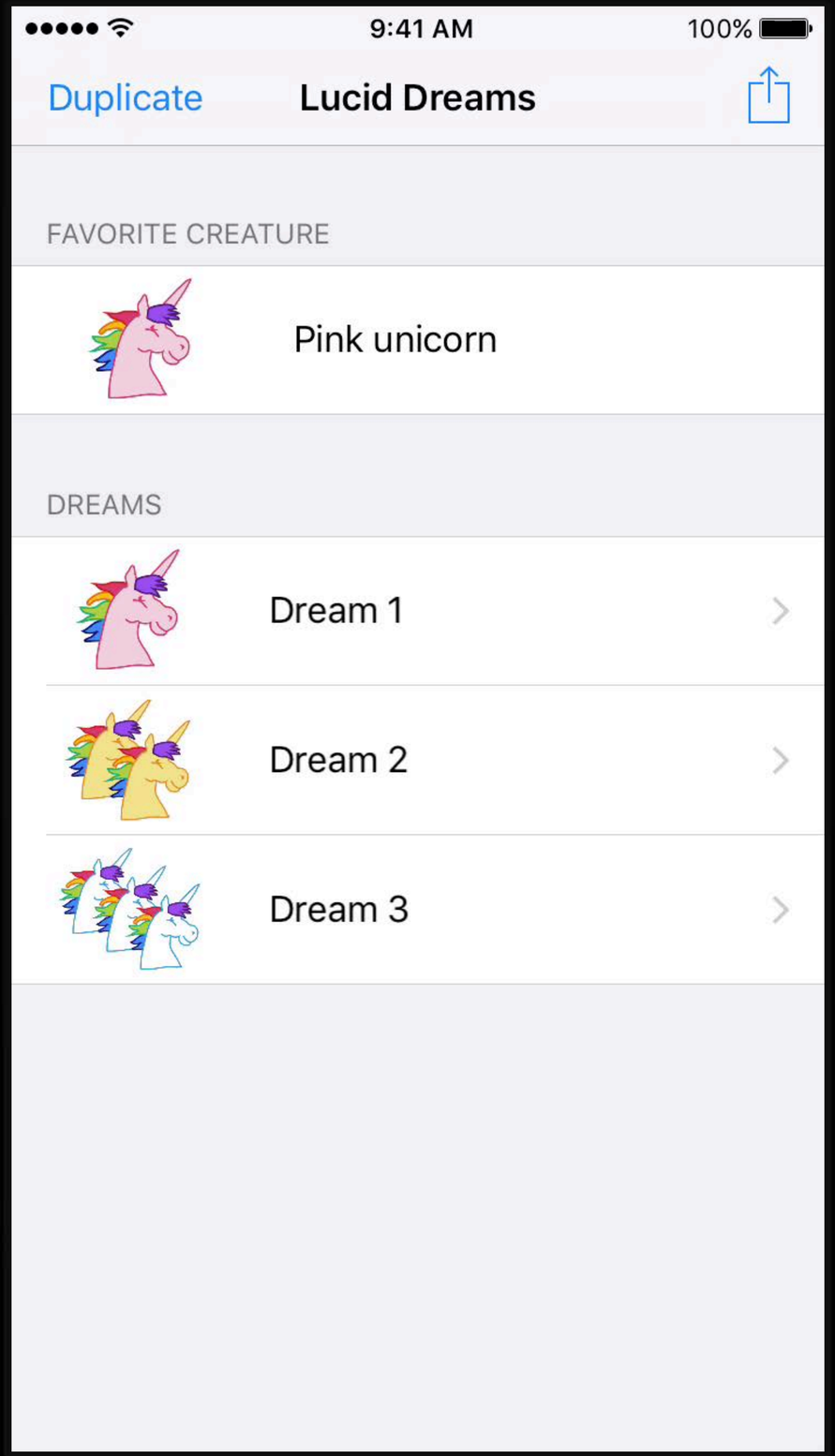
- Better local reasoning

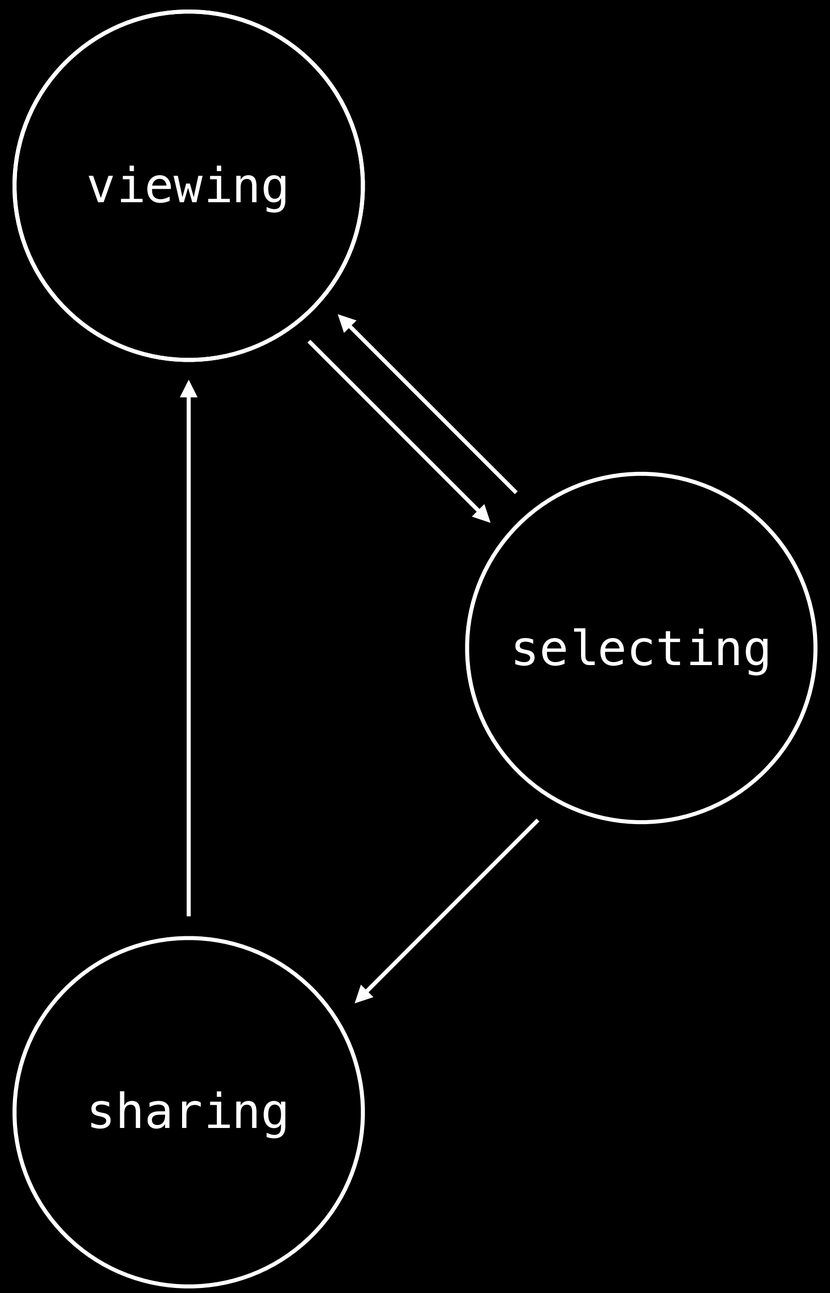
Values compose well with other values

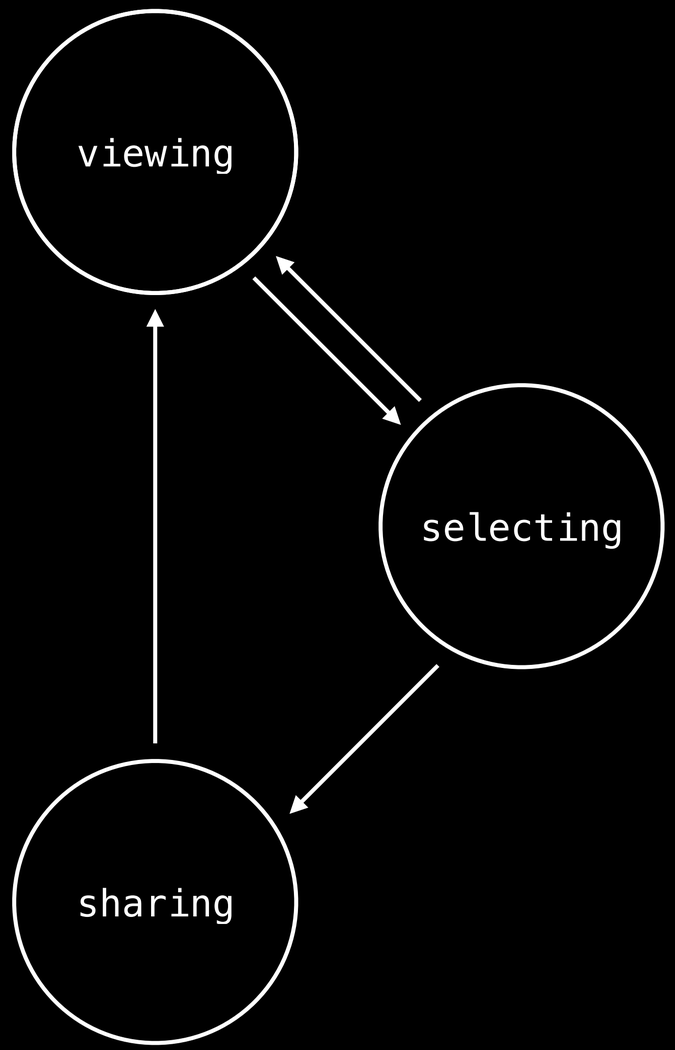
Controller

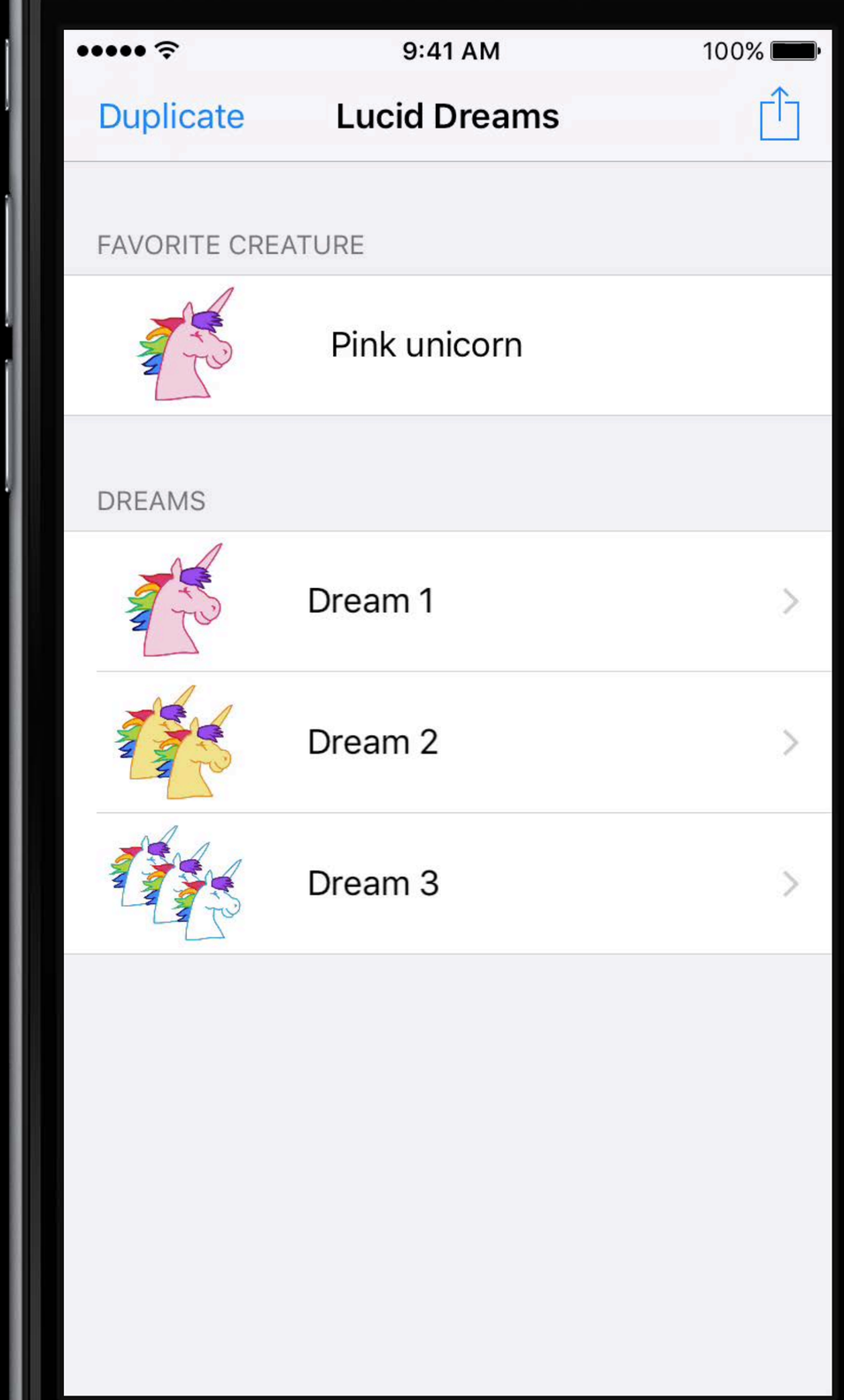
UI state

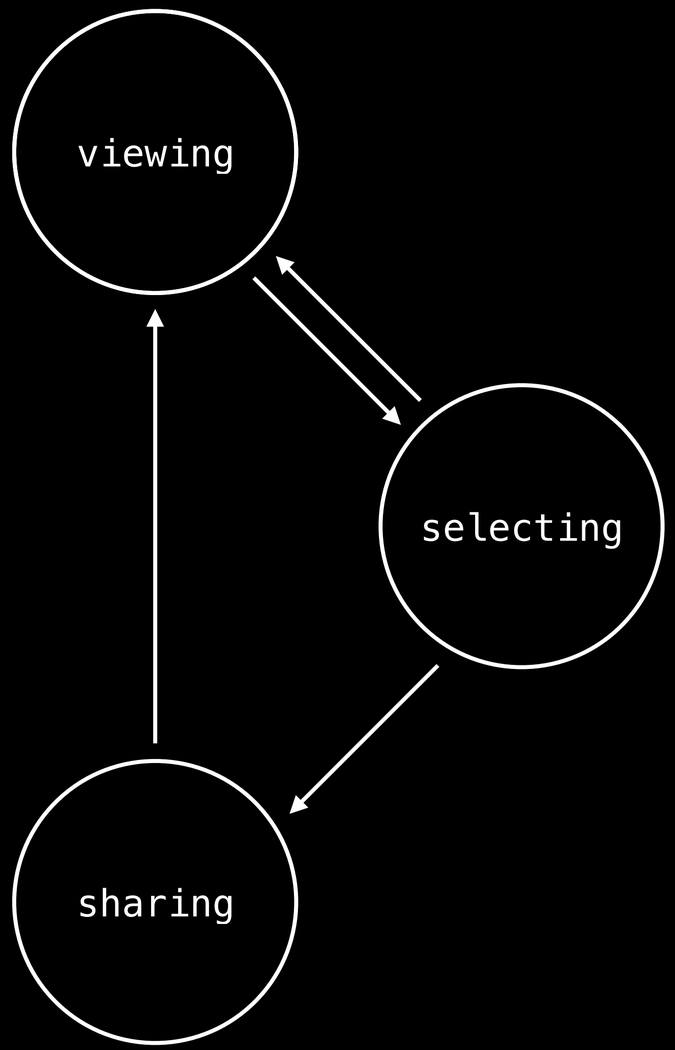

9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

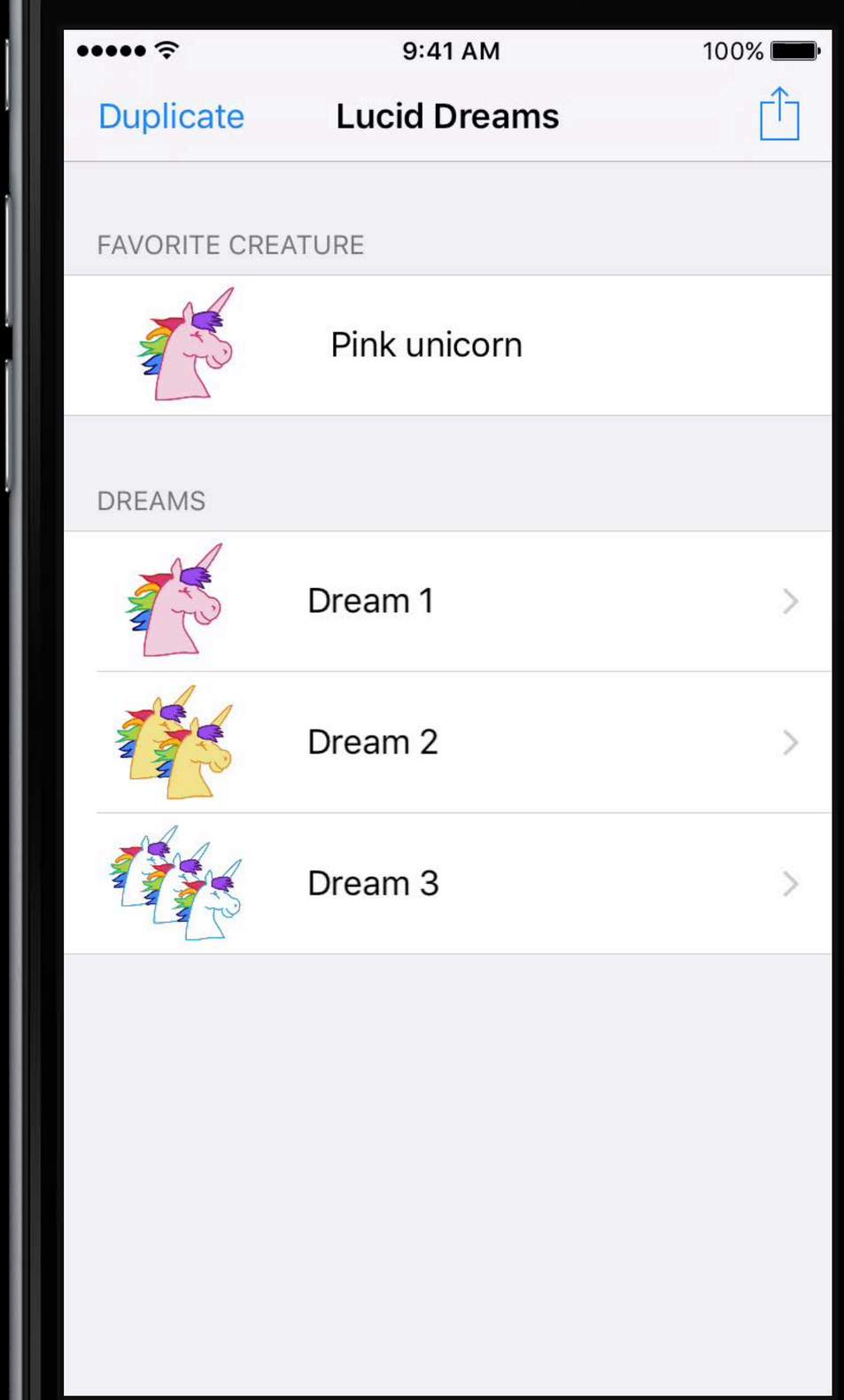

9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

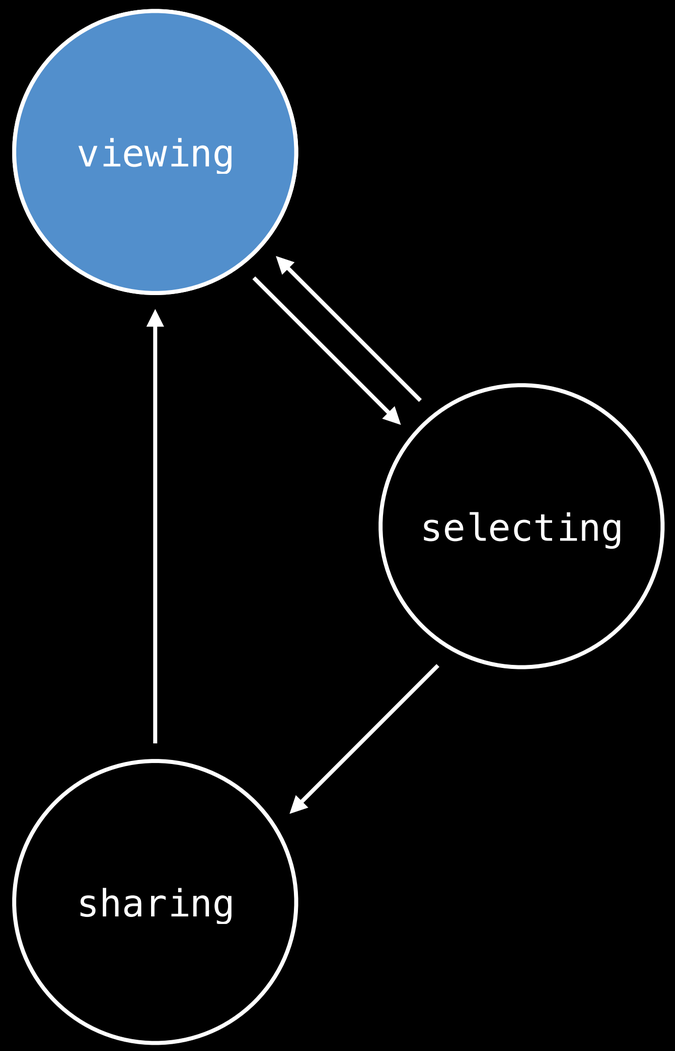

viewing
selecting
sharing

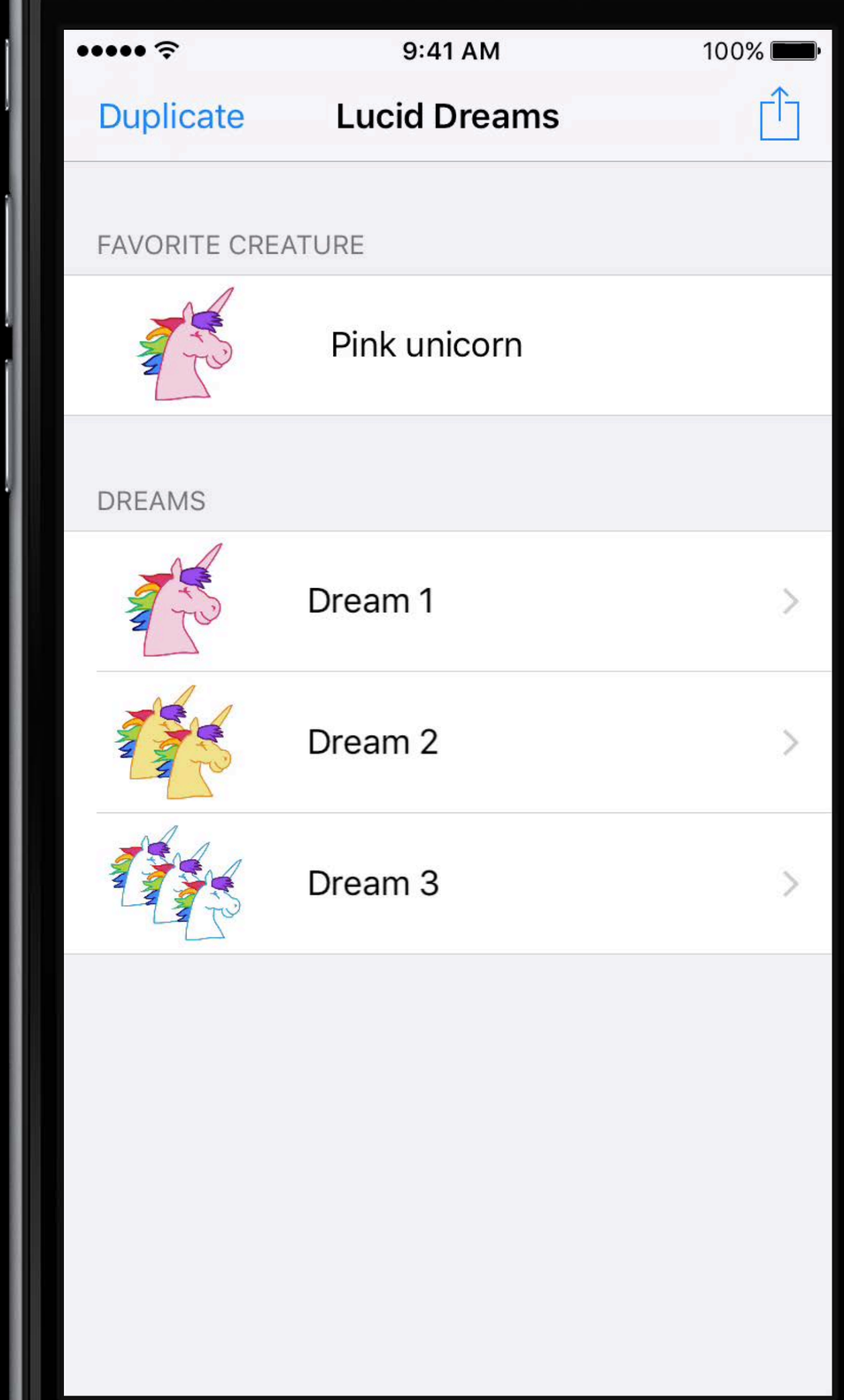

viewing
selecting
sharing

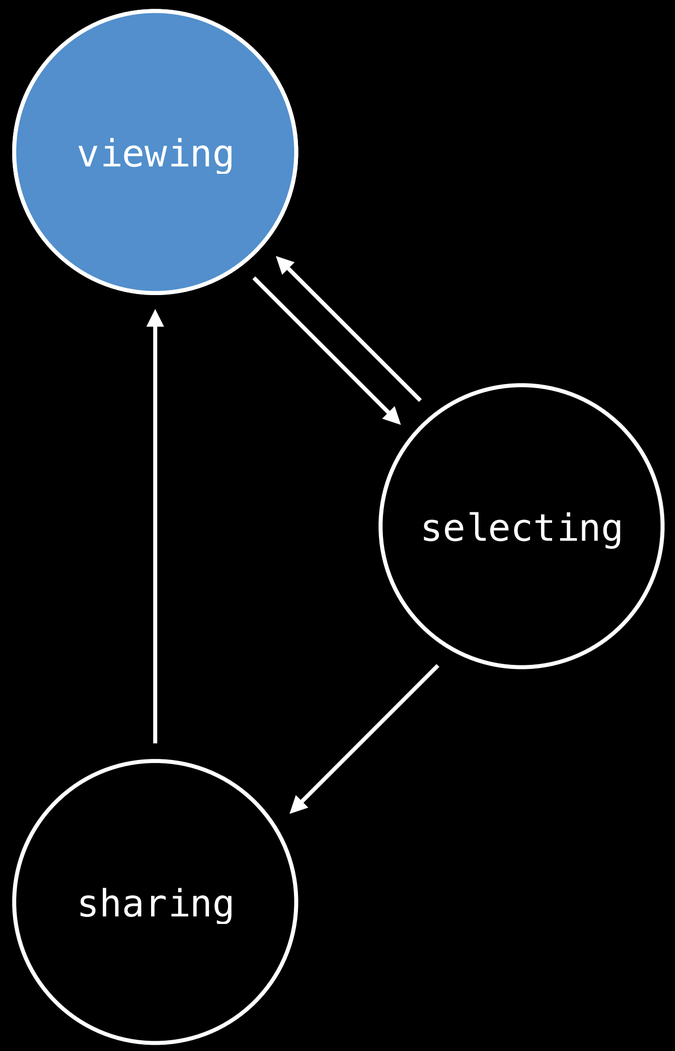

9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

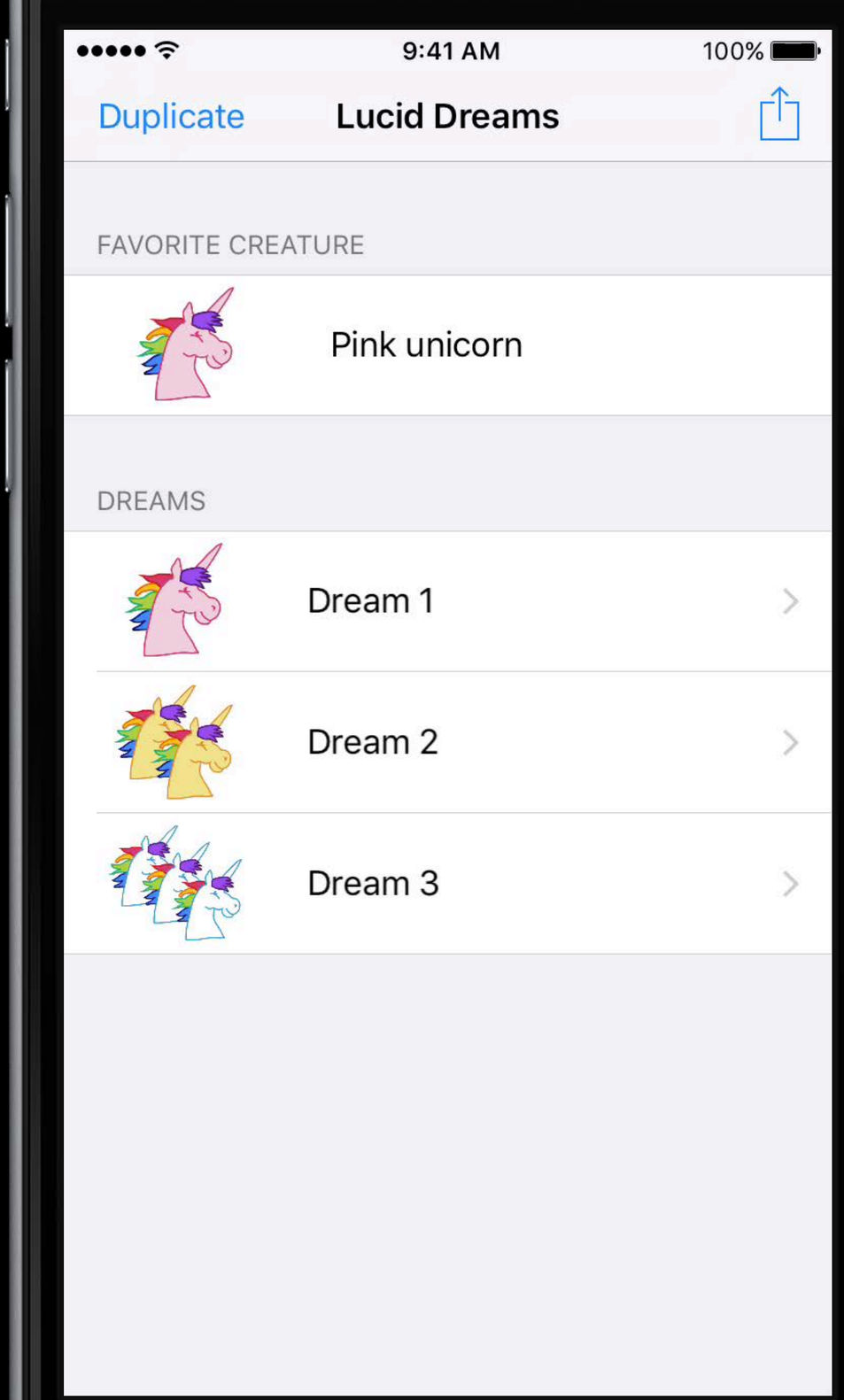

viewing
selecting
sharing

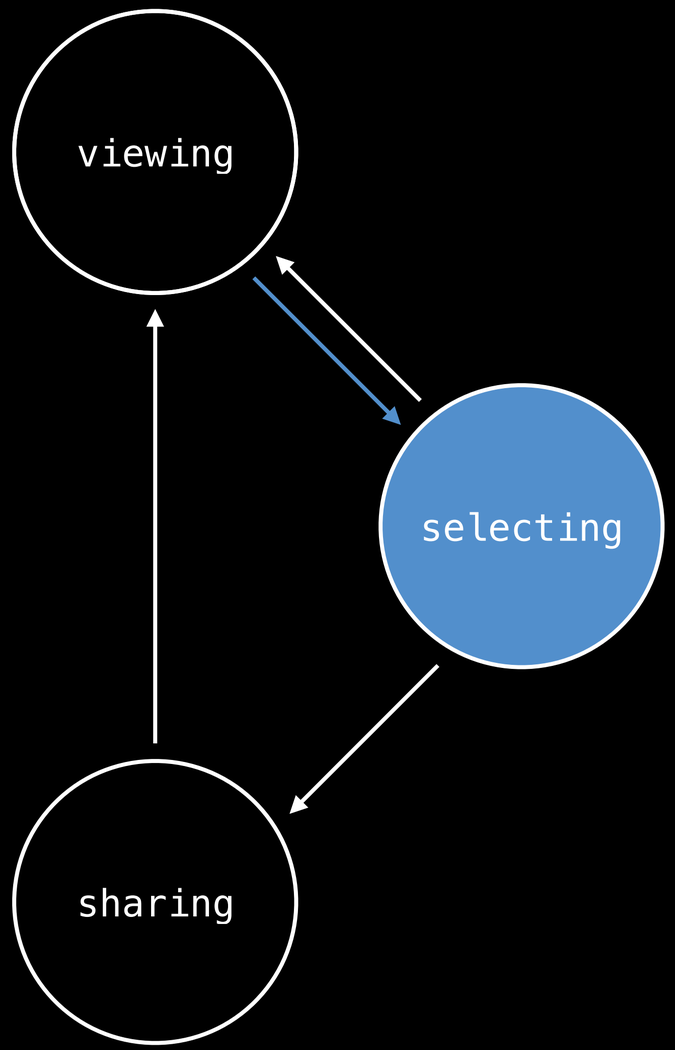

9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

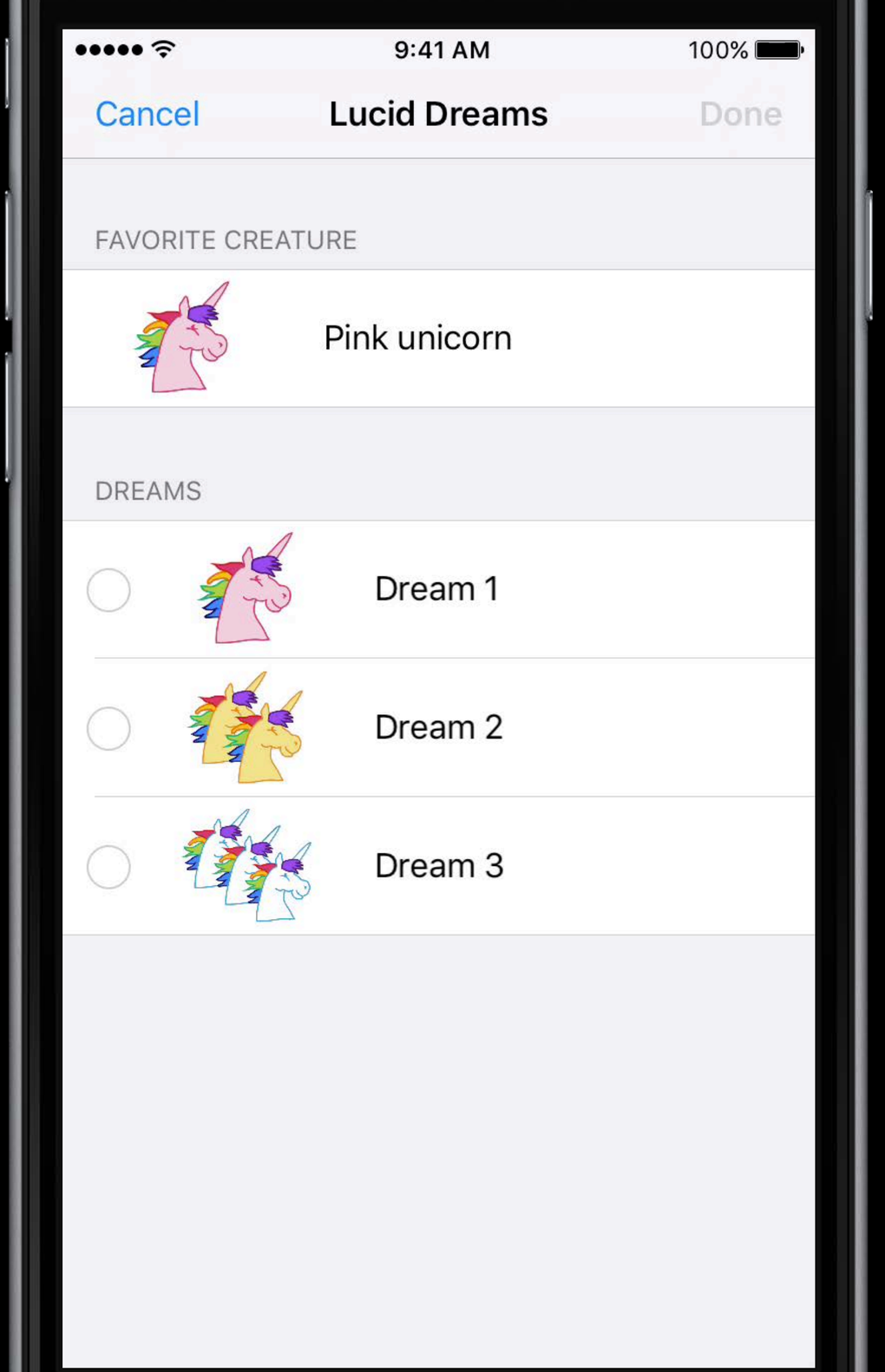

viewing
selecting
sharing

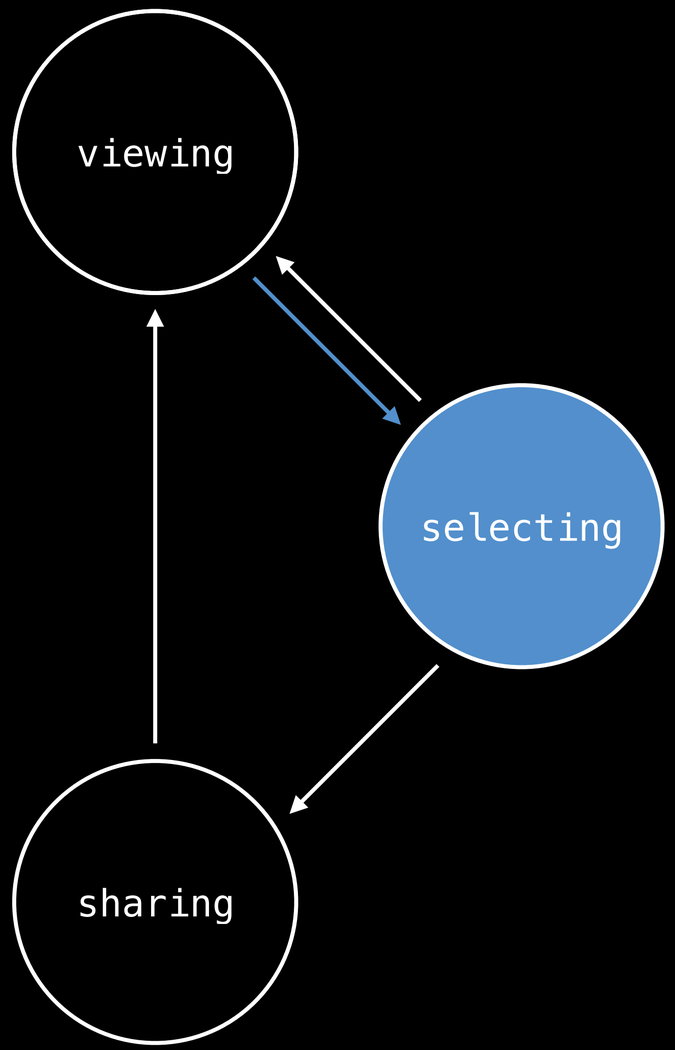

9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

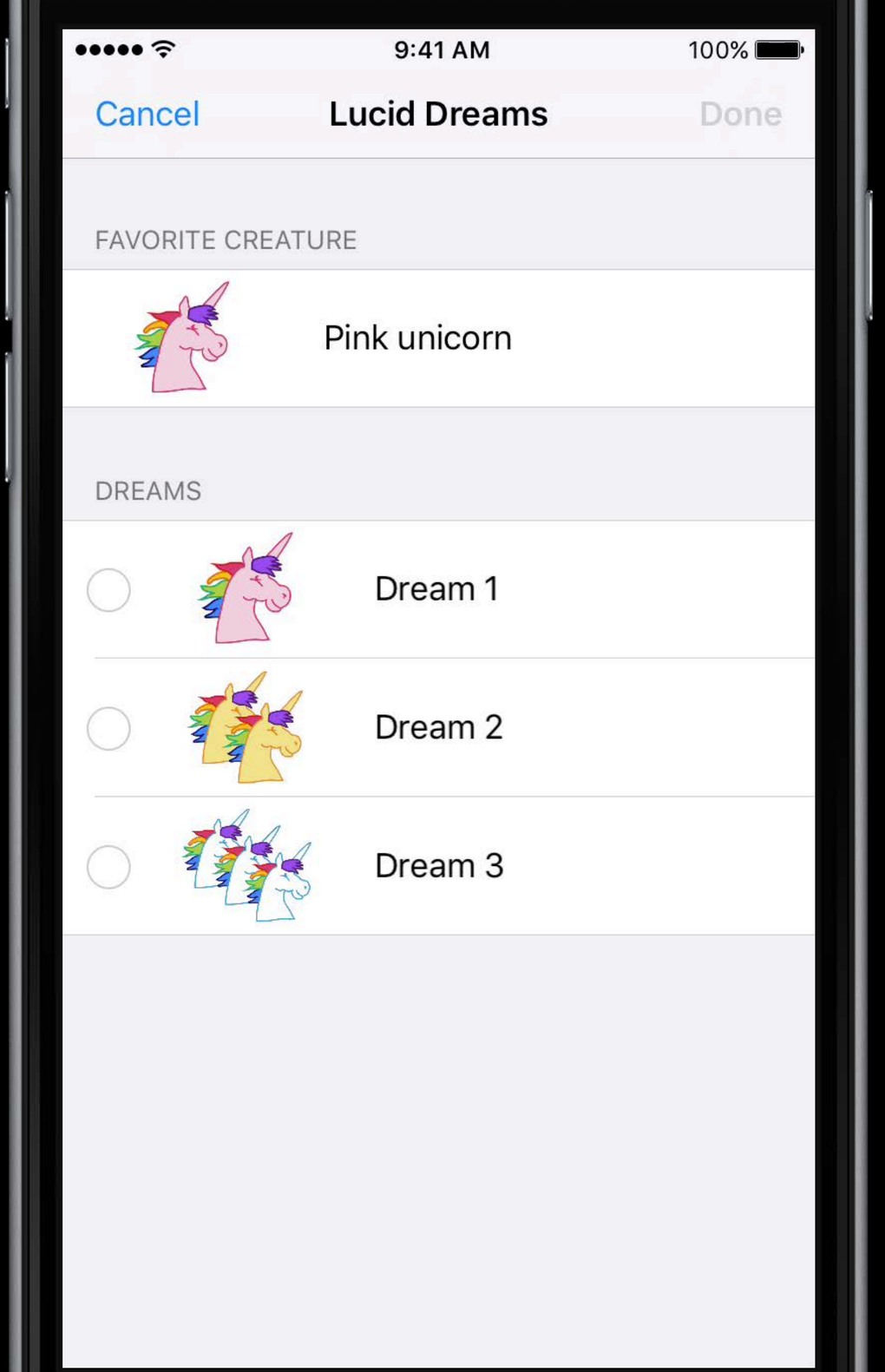

viewing
selecting
sharing

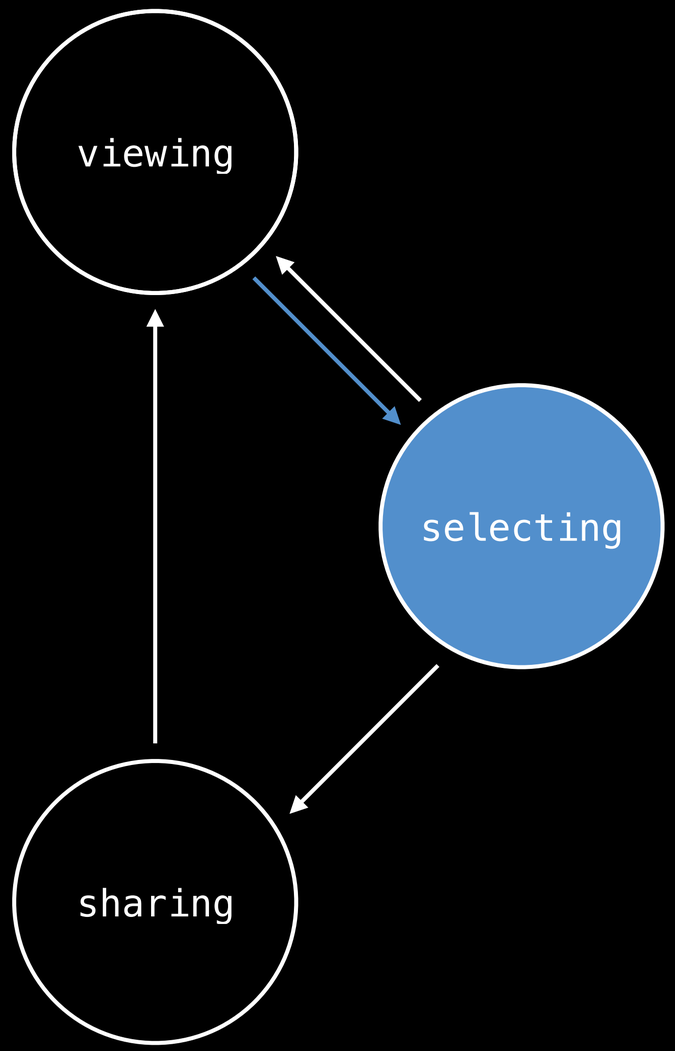

9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

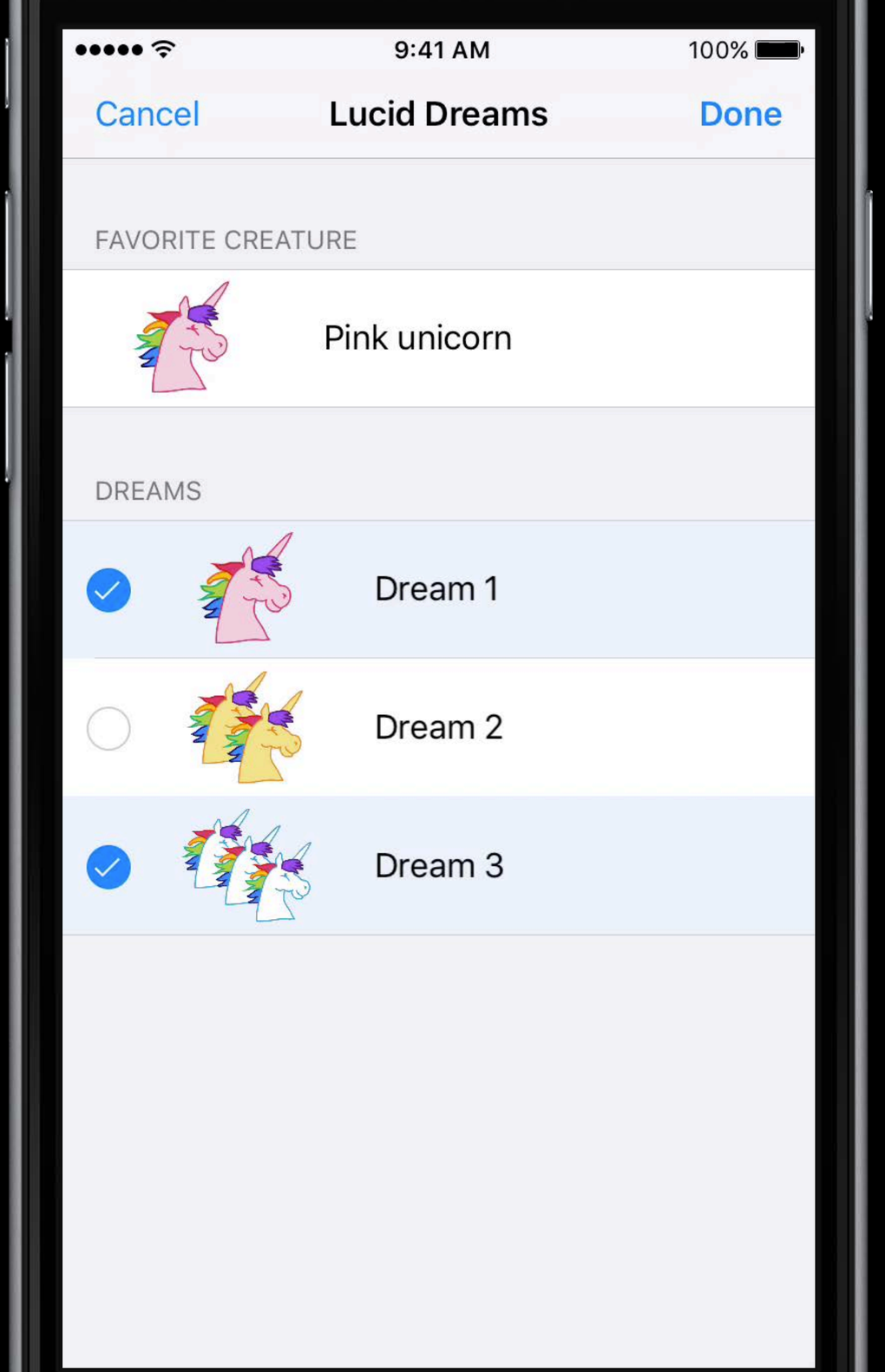

viewing
selecting
sharing

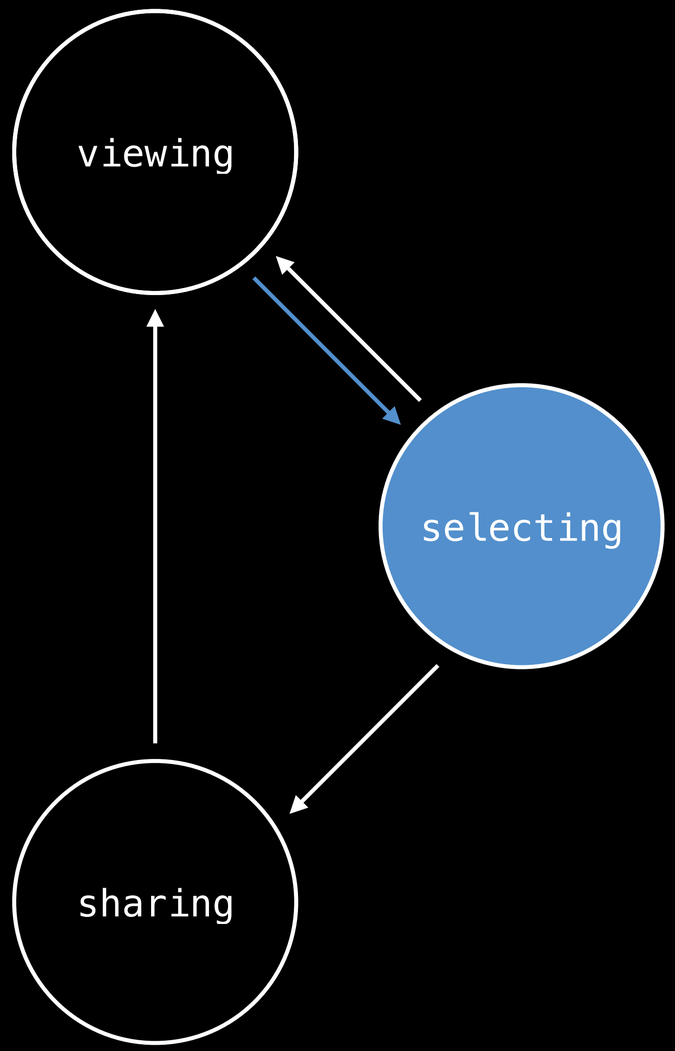

9:41 AM
100%
Cancel
Lucid Dreams
Done
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

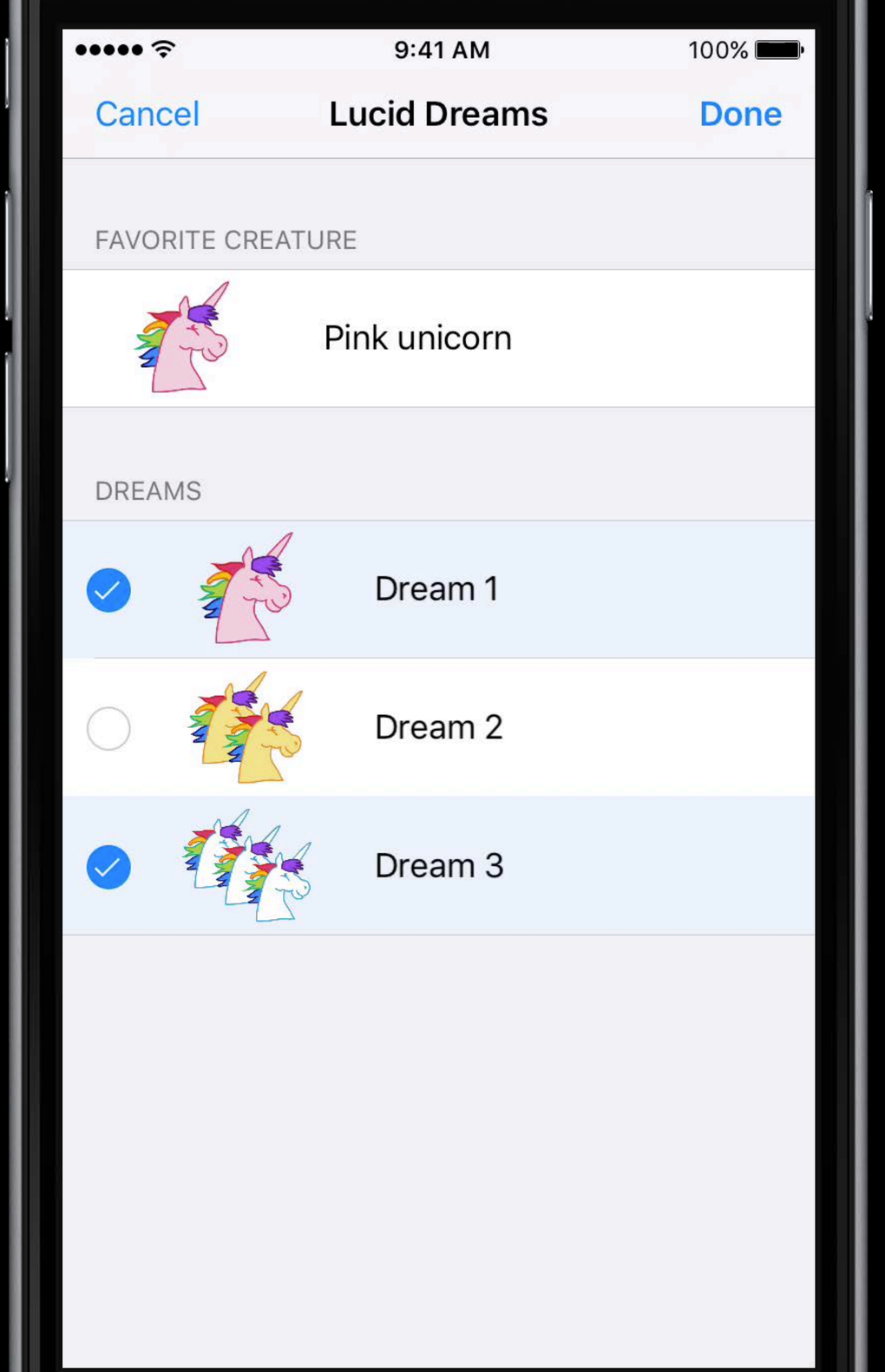

viewing
selecting
sharing

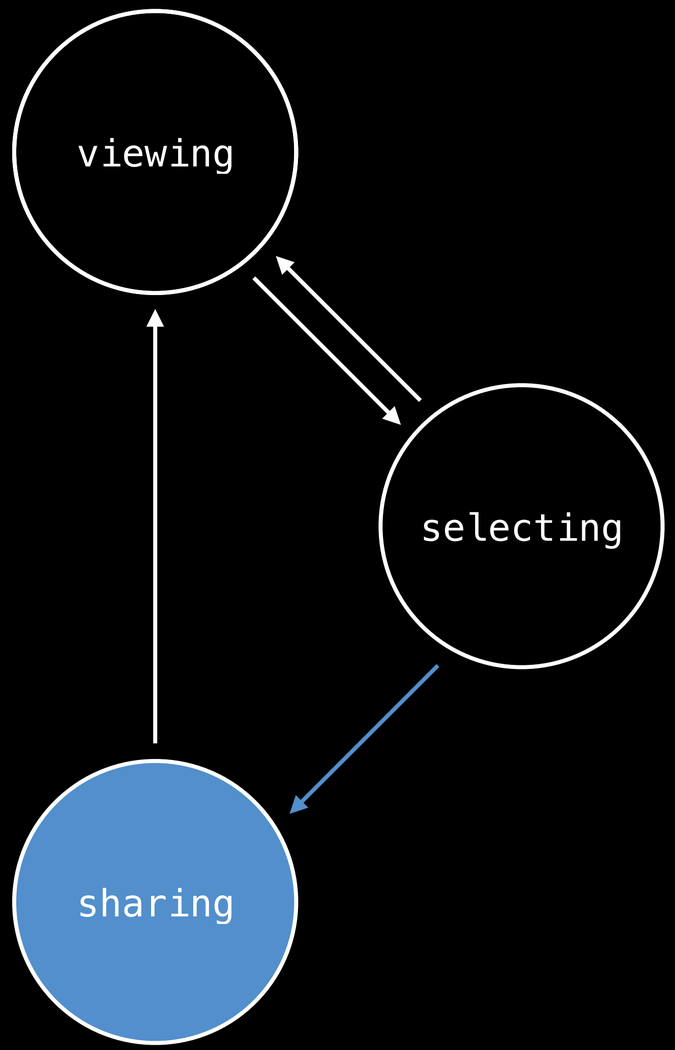

9:41 AM
100%
Cancel
Lucid Dreams
Done
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

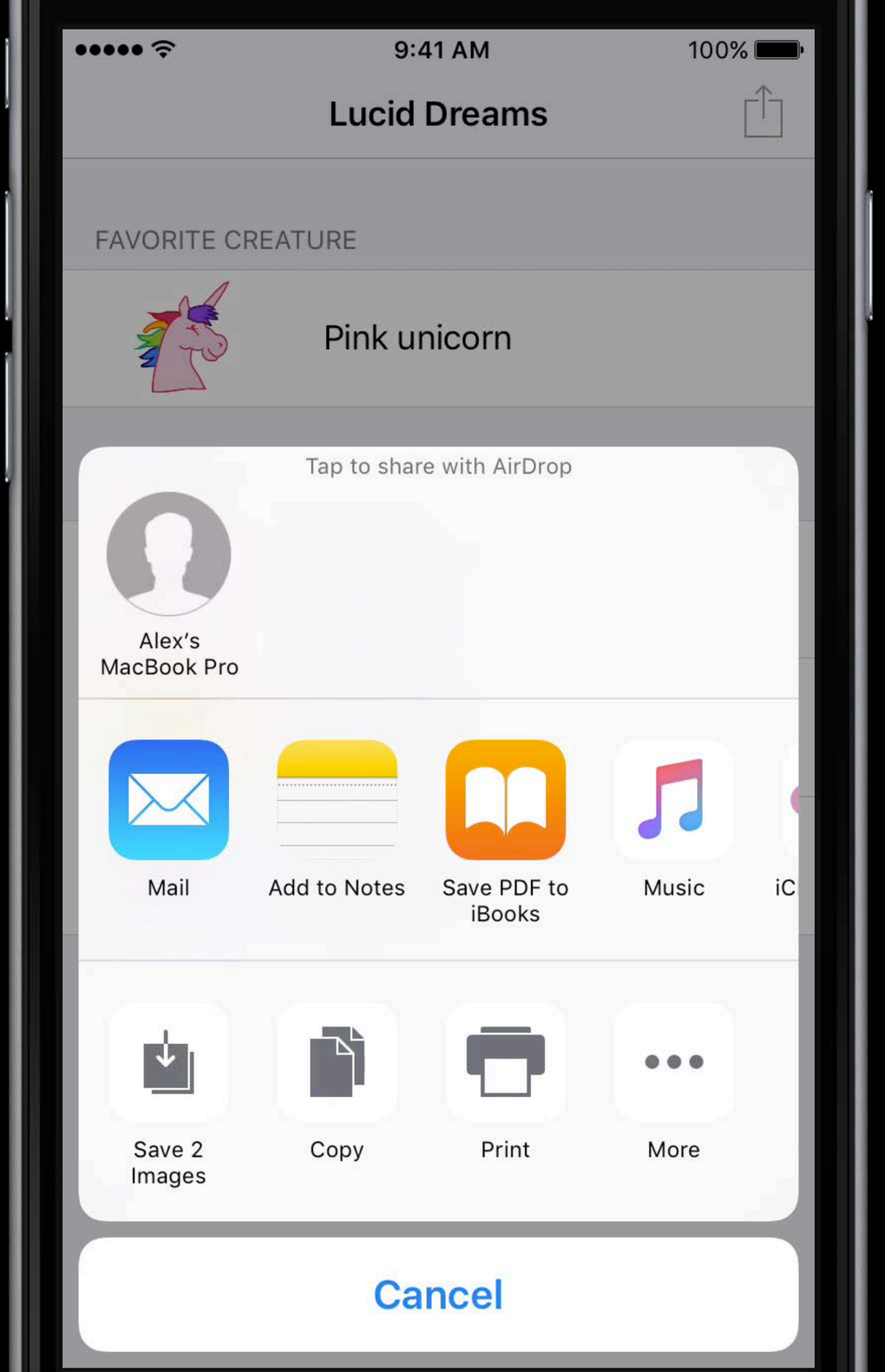

viewing
selecting
sharing

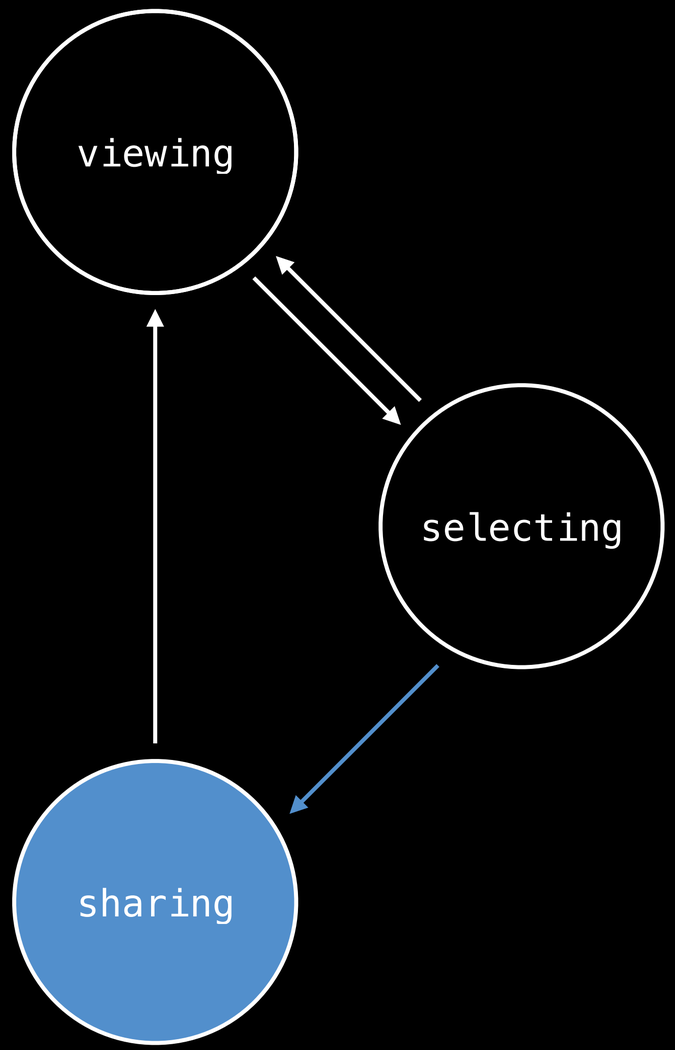

9:41 AM
100%
Cancel
Lucid Dreams
Done
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

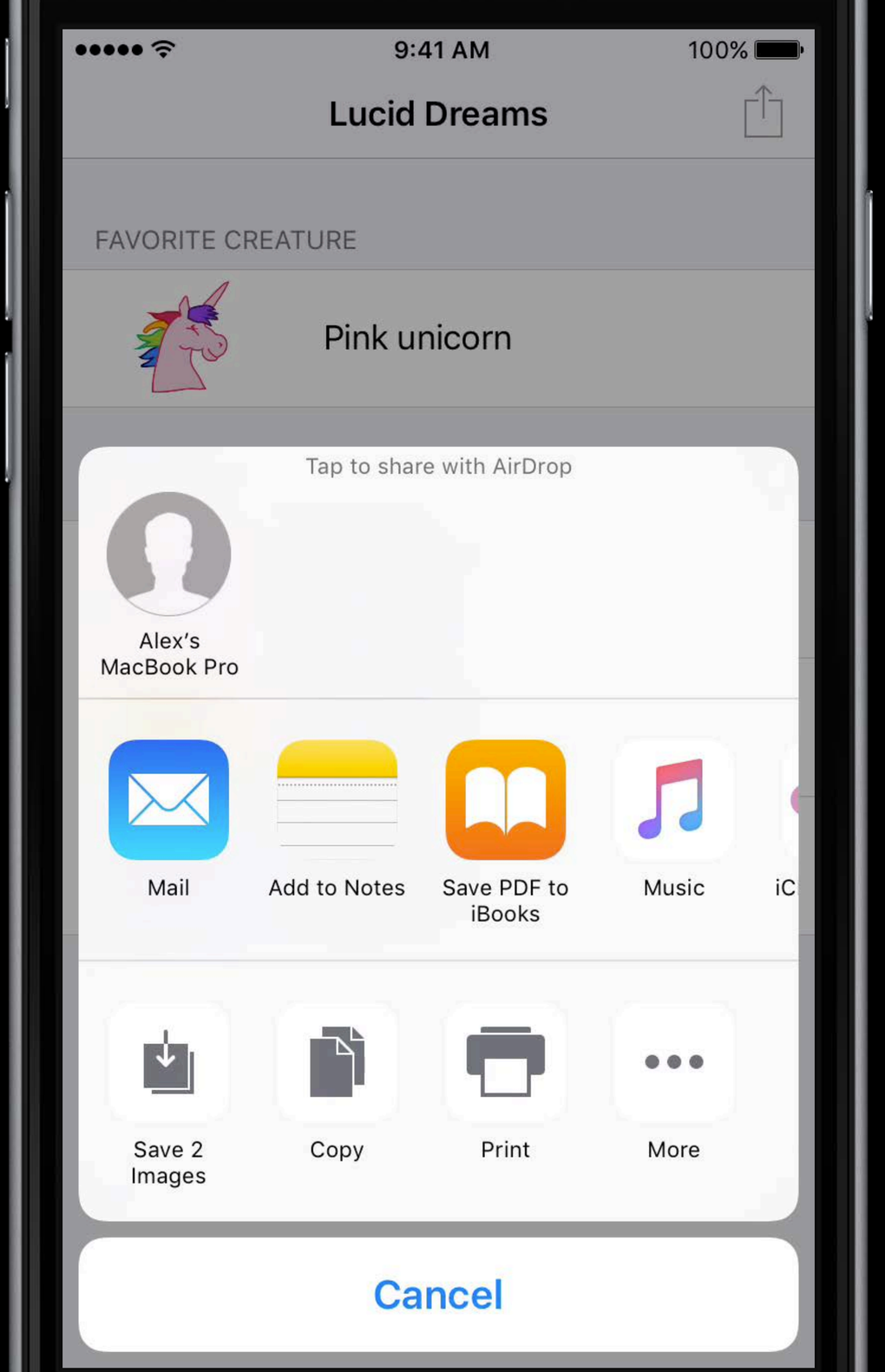

viewing
selecting
sharing

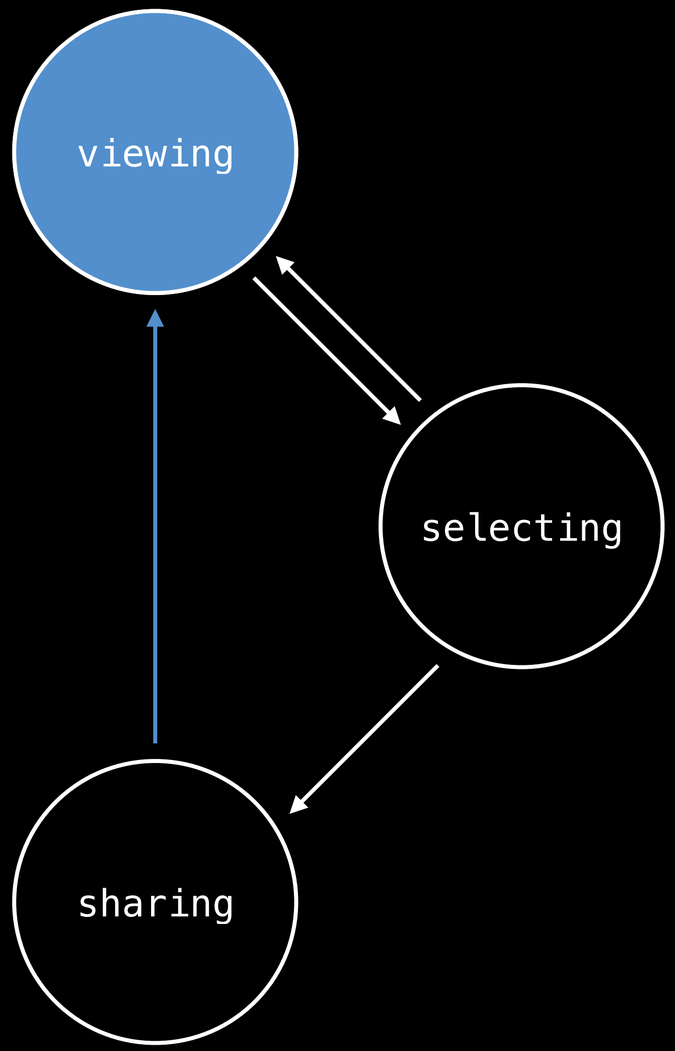

9:41 AM
100%
Cancel
Lucid Dreams
Done
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

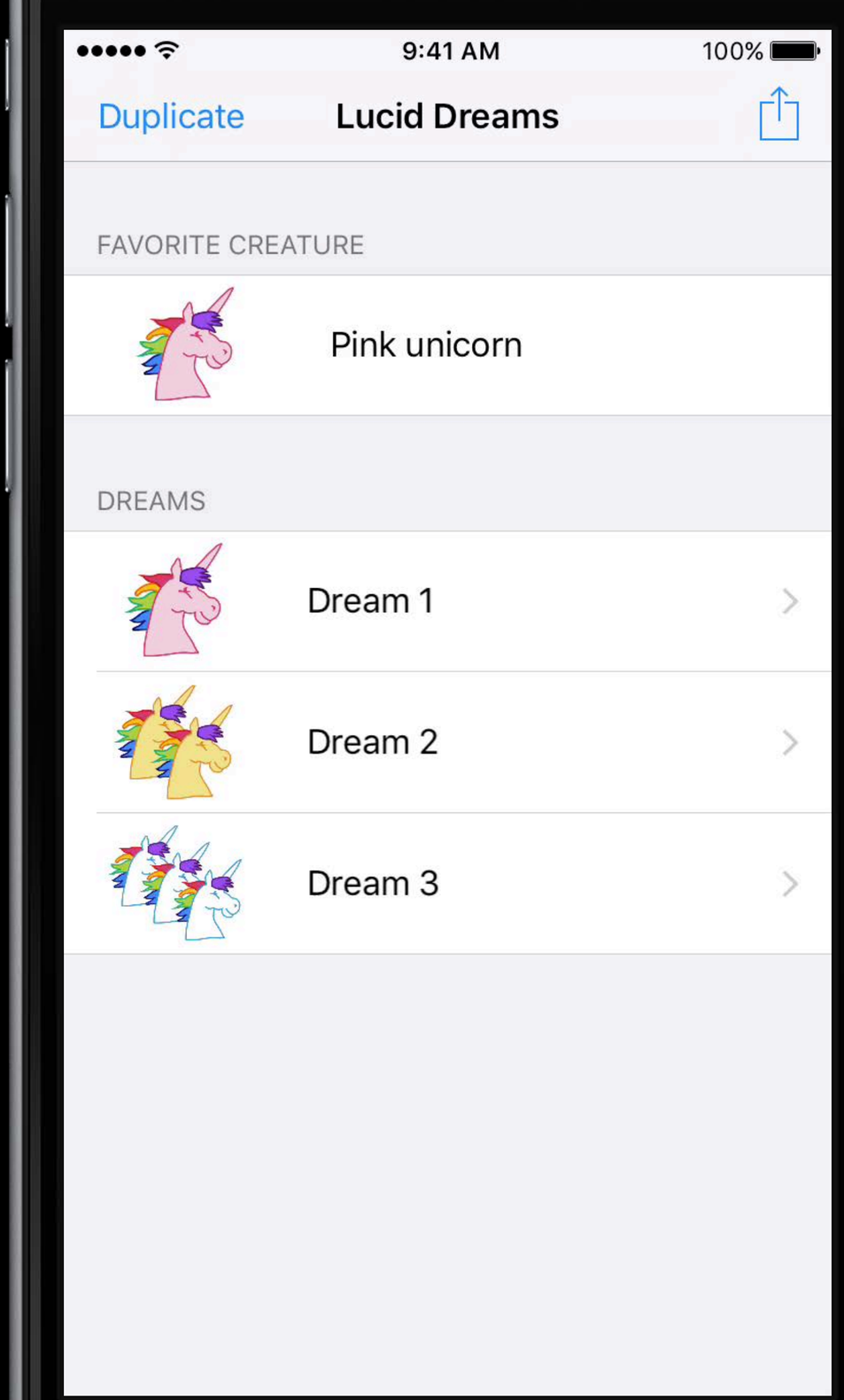

viewing
selecting
sharing

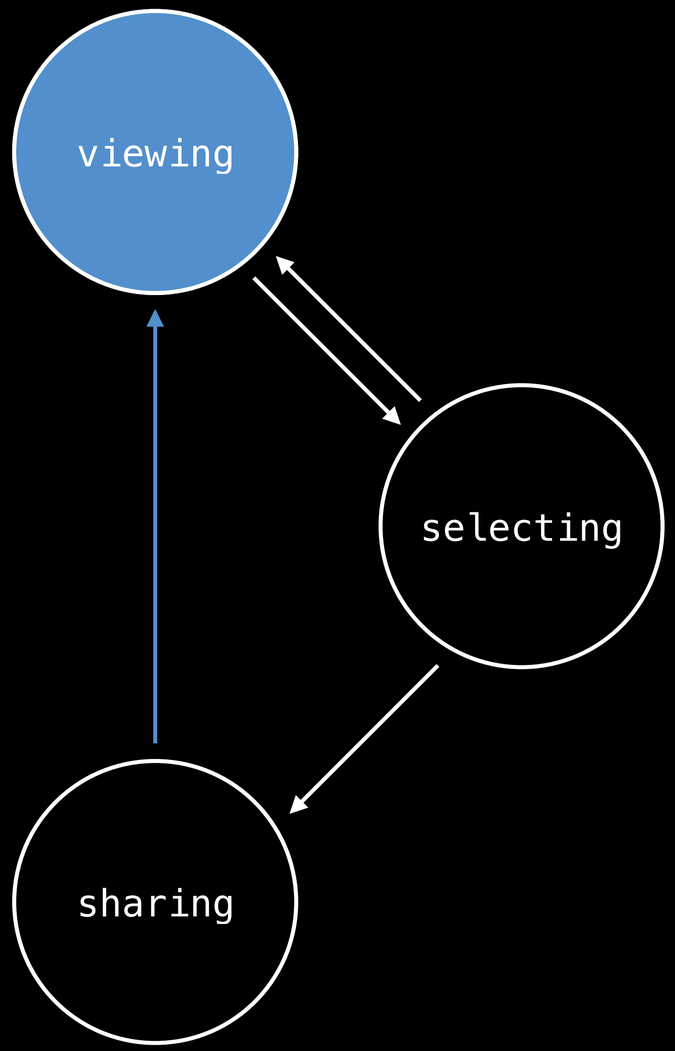

9:41 AM
100%
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
Tap to share with AirDrop
Alex's MacBook Pro
Mail
Add to Notes
Save PDF to iBooks
Music
Save 2 Images
Copy
Print
More
Cancel

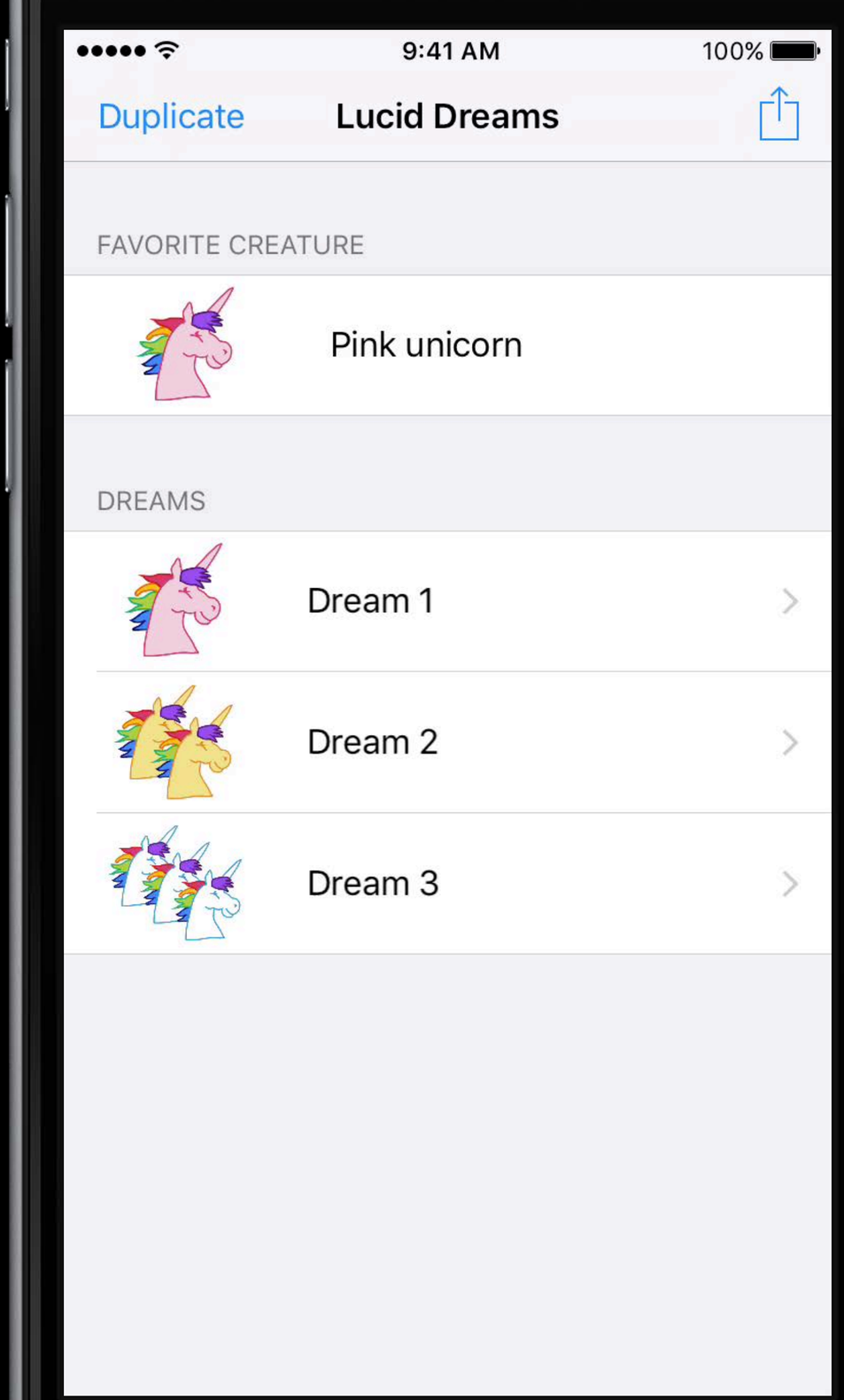

viewing
selecting
sharing

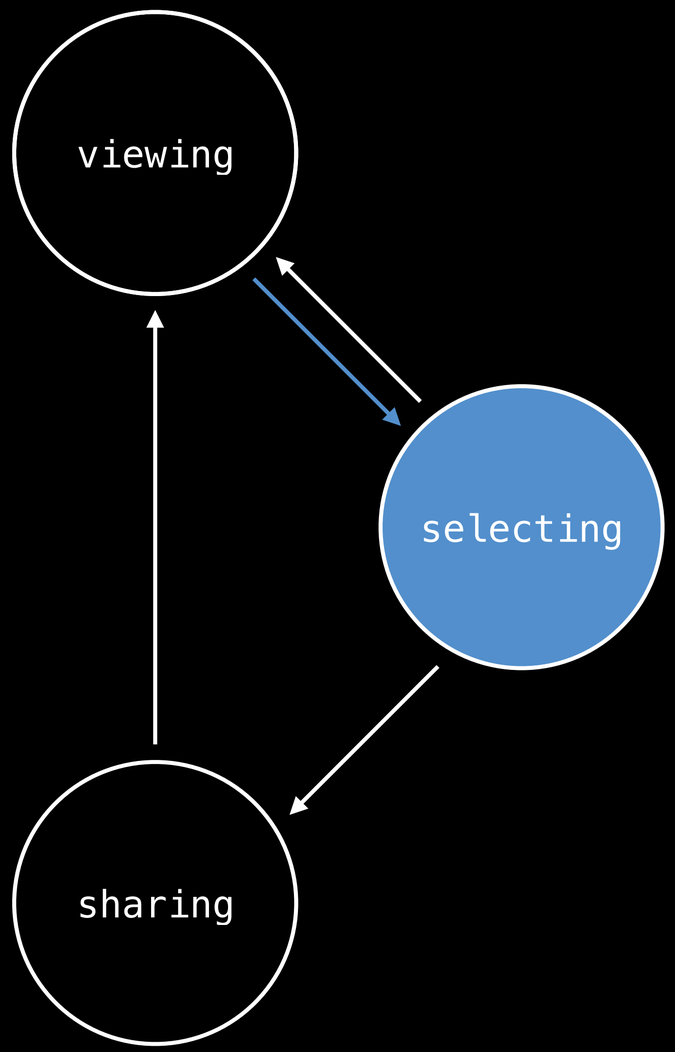

9:41 AM
100%
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
Tap to share with AirDrop
Alex's MacBook Pro
Mail
Add to Notes
Save PDF to iBooks
Music
Save 2 Images
Copy
Print
More
Cancel

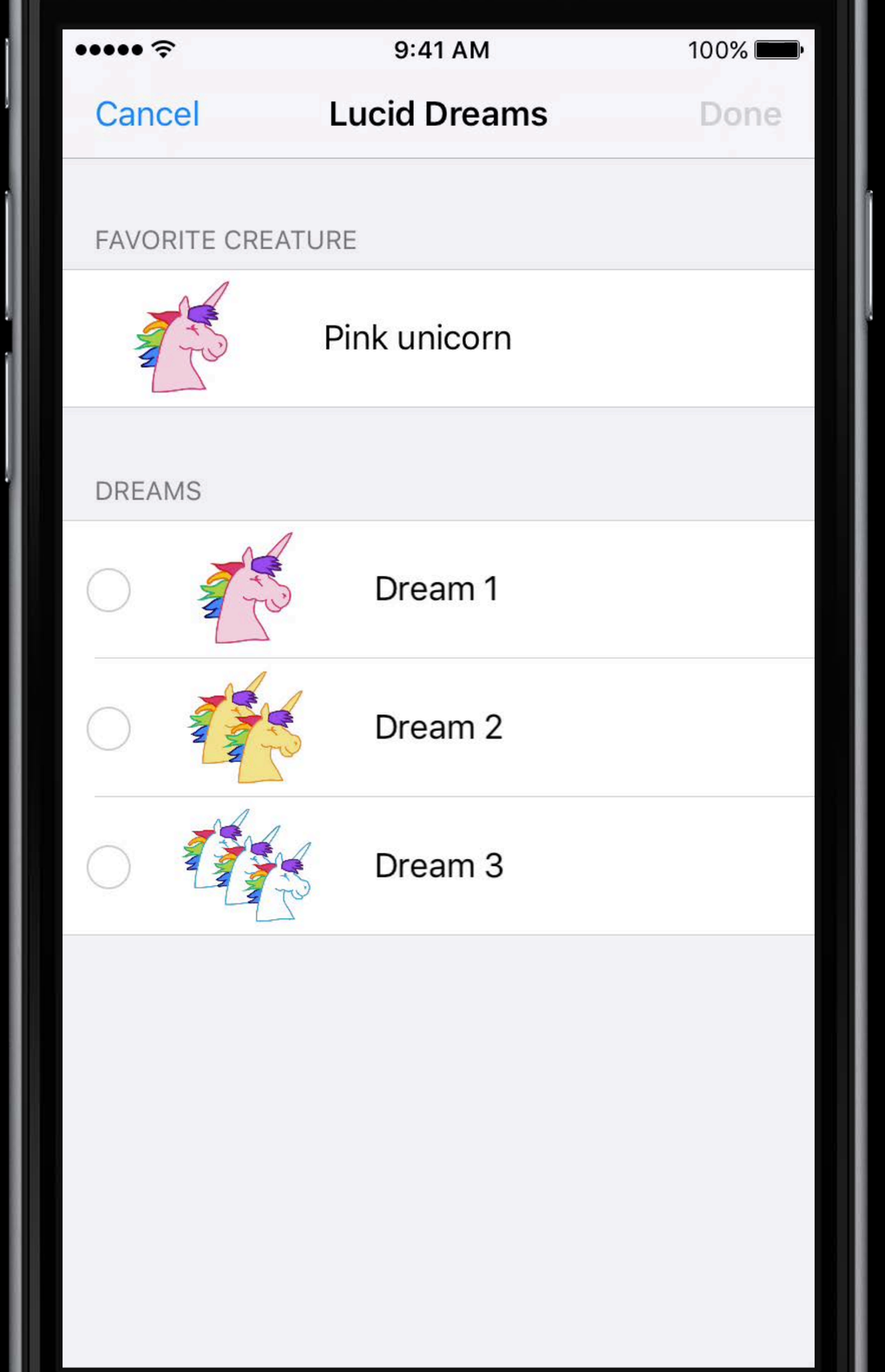

viewing
selecting
sharing

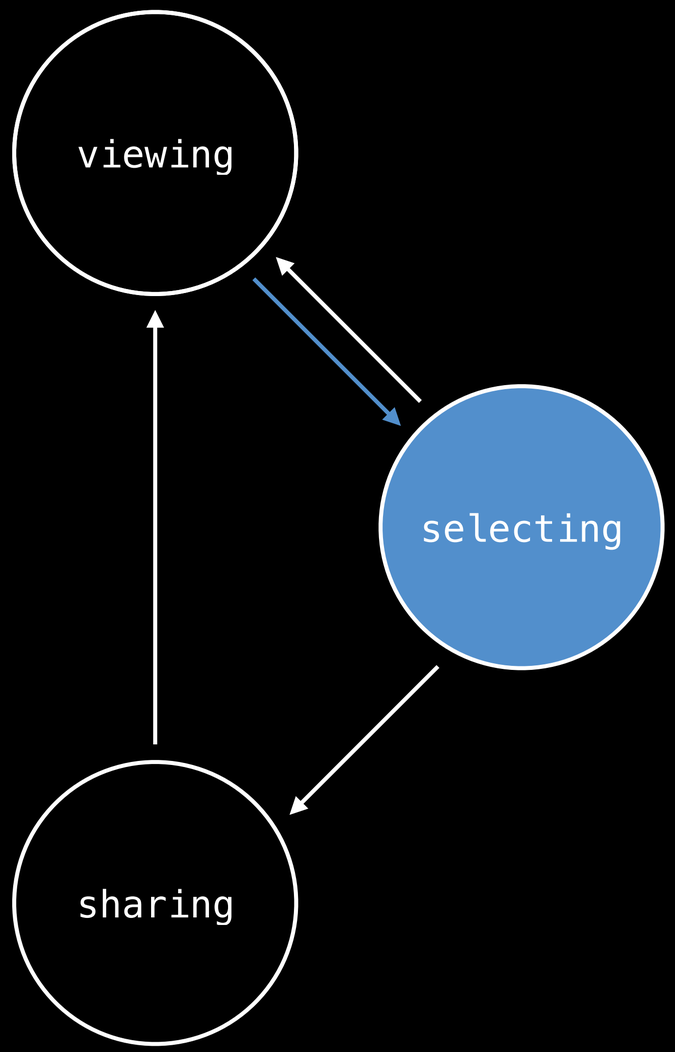

9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

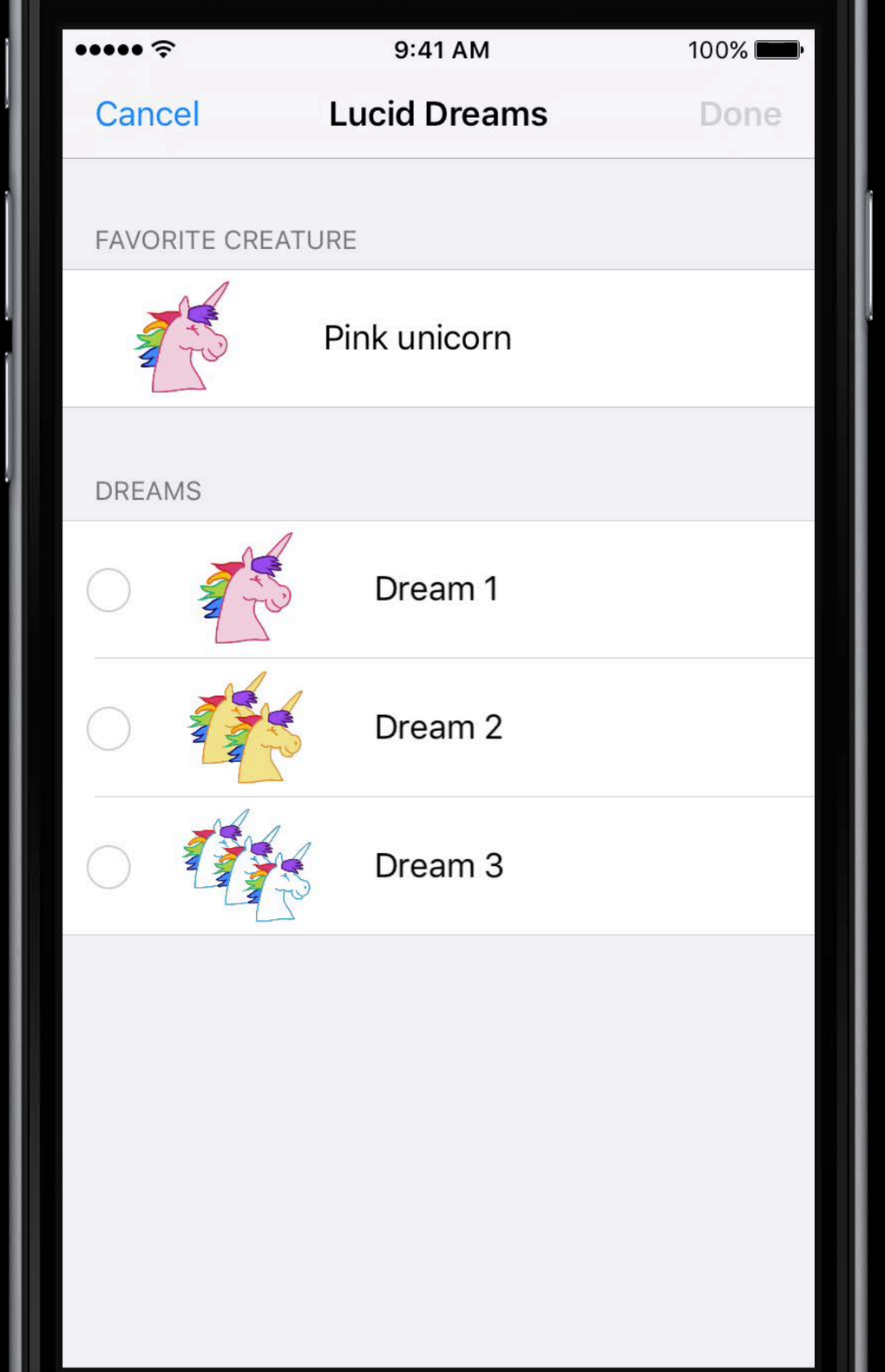

viewing
selecting
sharing

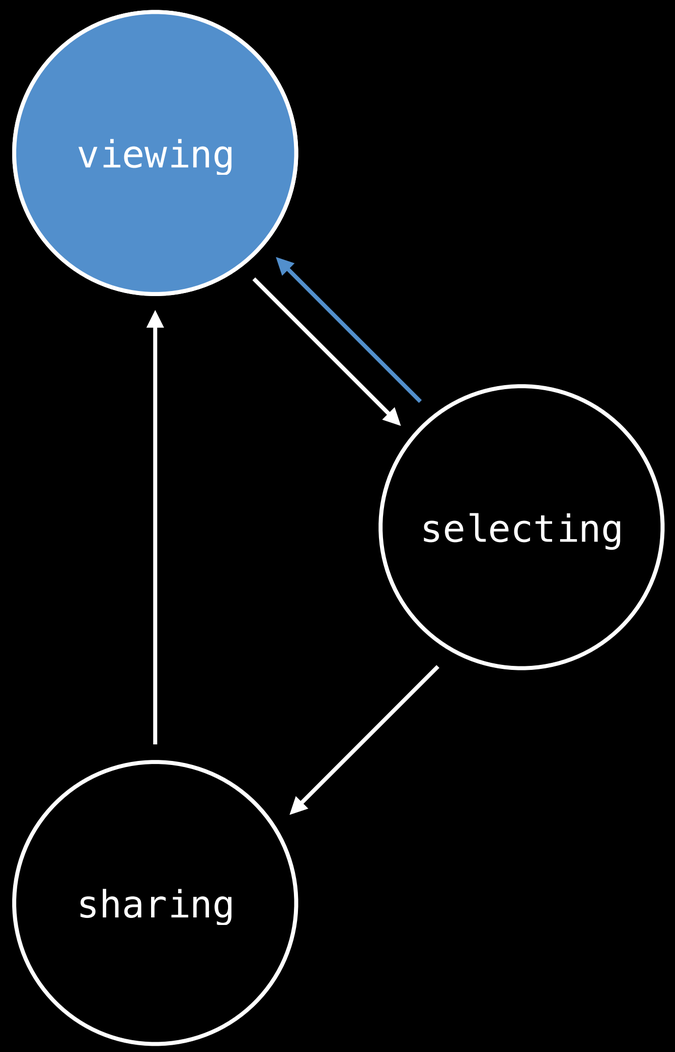

9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

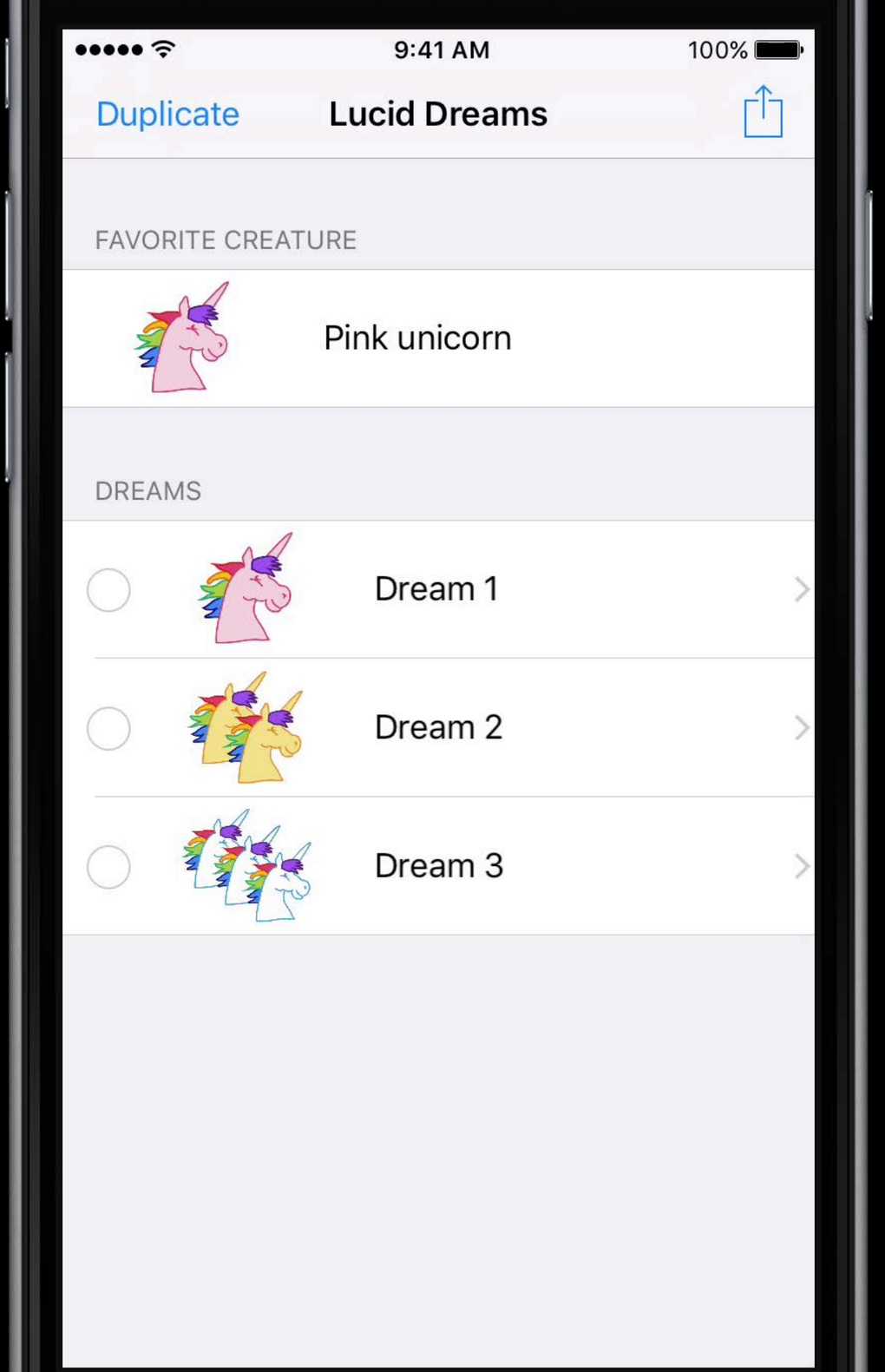

viewing
selecting
sharing

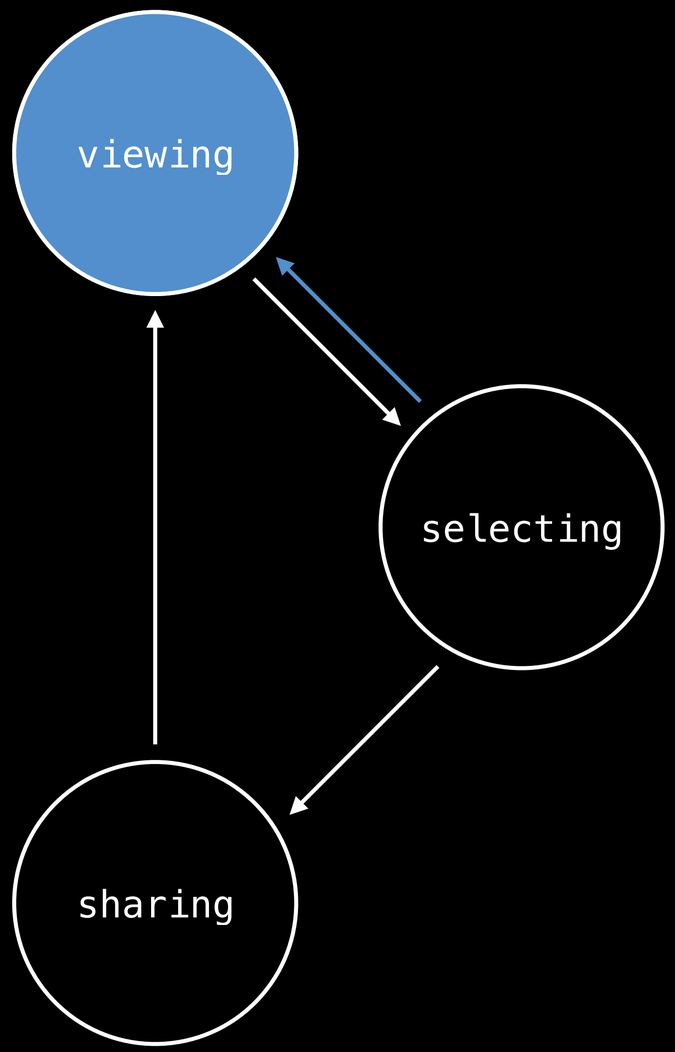

9:41 AM
100%
Cancel
Lucid Dreams
Done
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

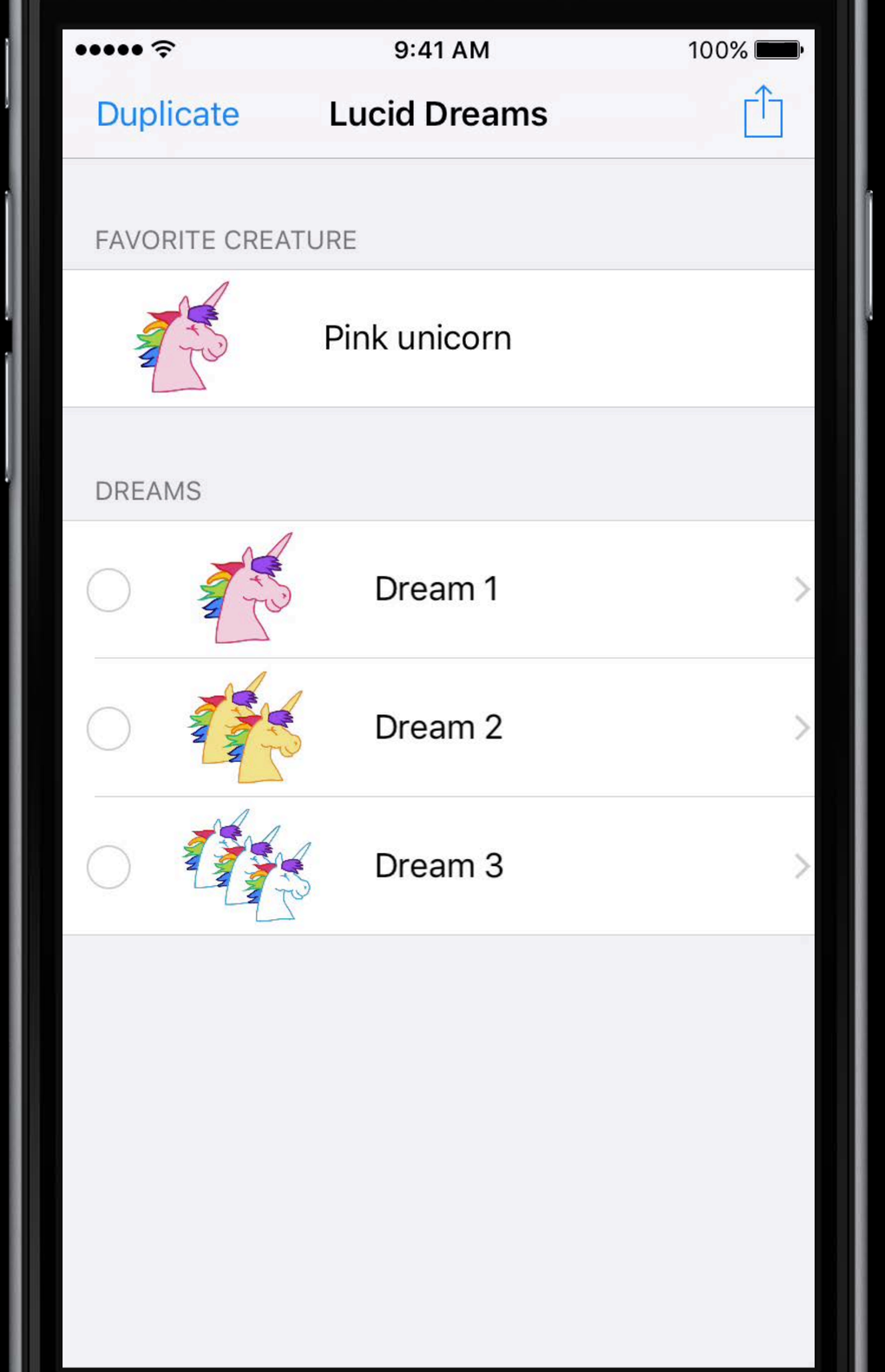

viewing
selecting
sharing

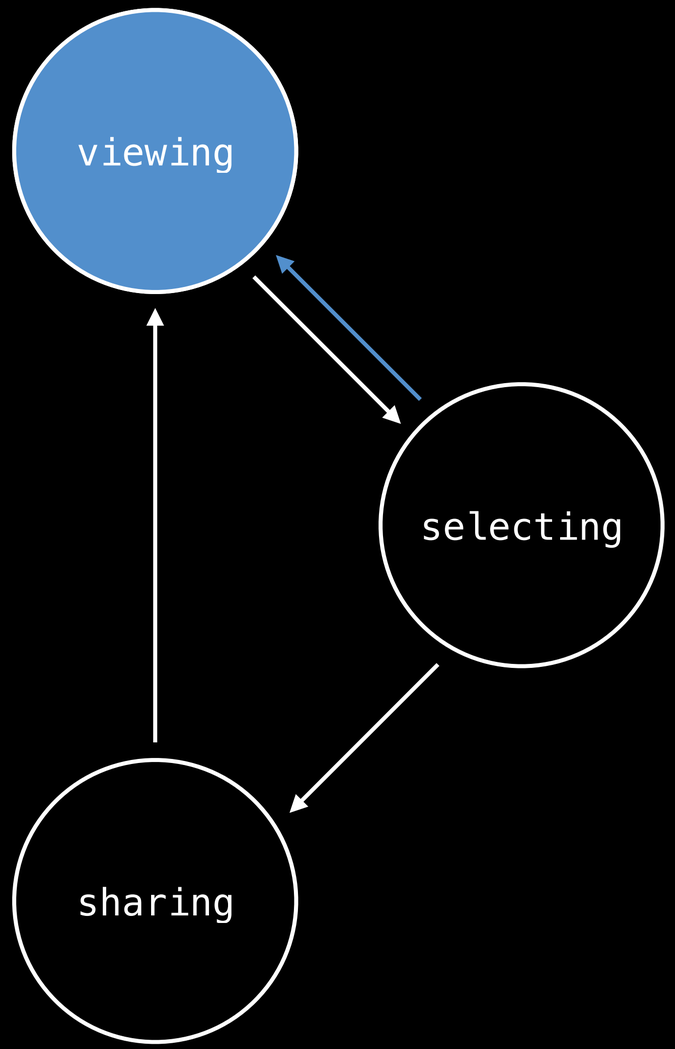

9:41 AM
100%
Cancel
Lucid Dreams
Done
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

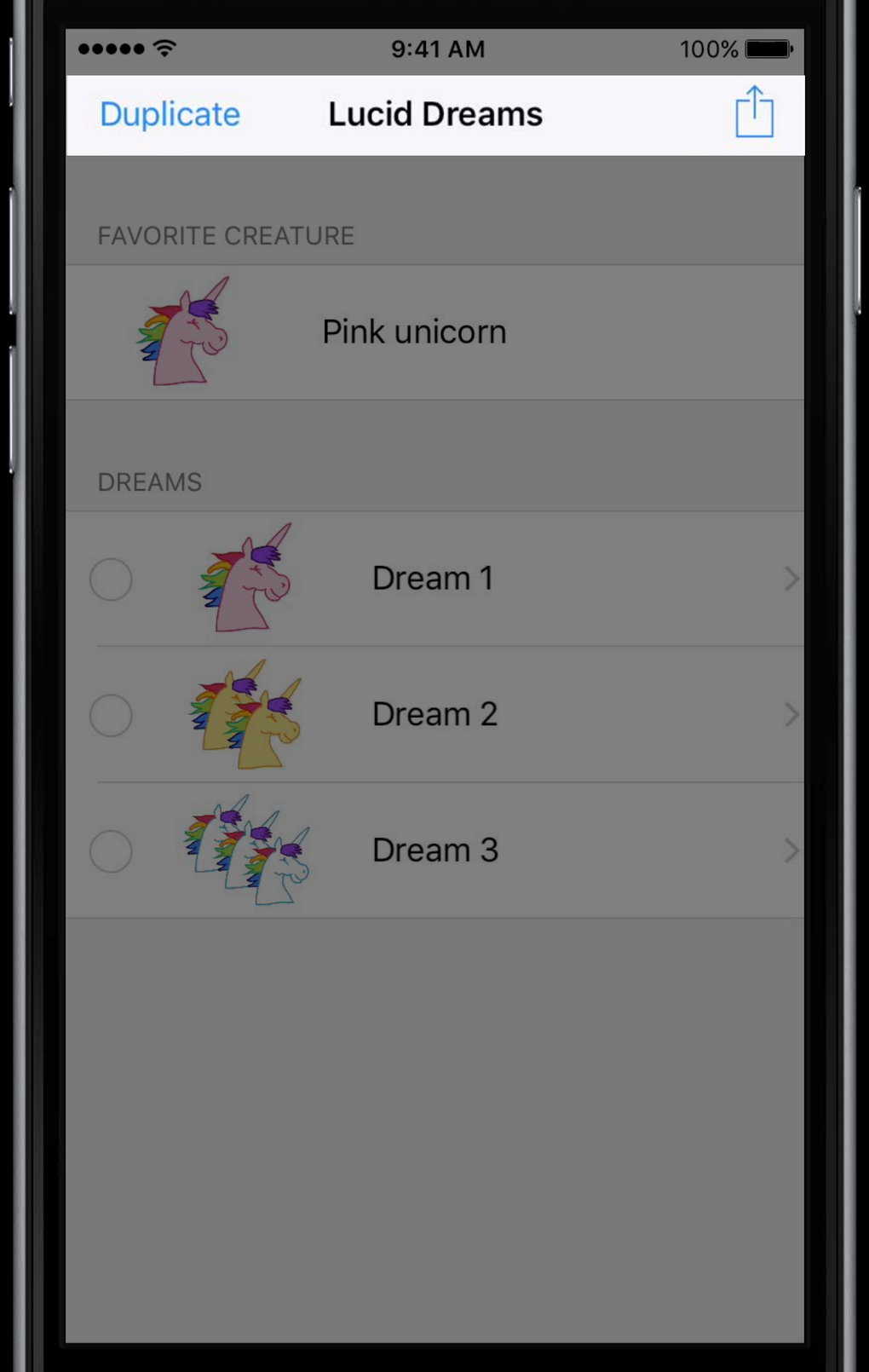

viewing
selecting
sharing

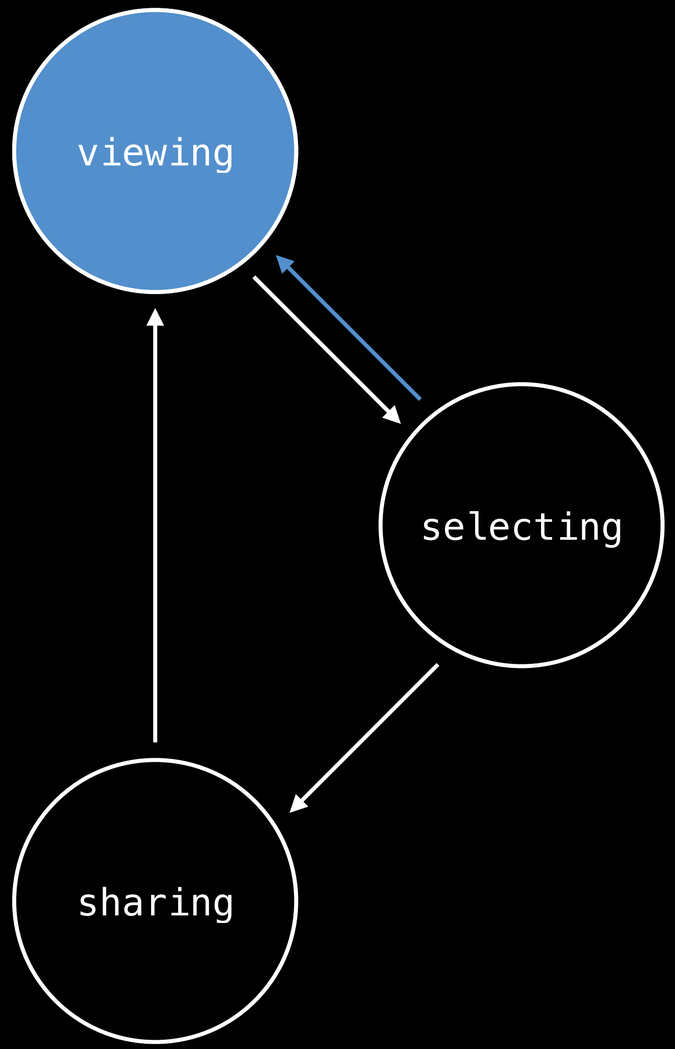

9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

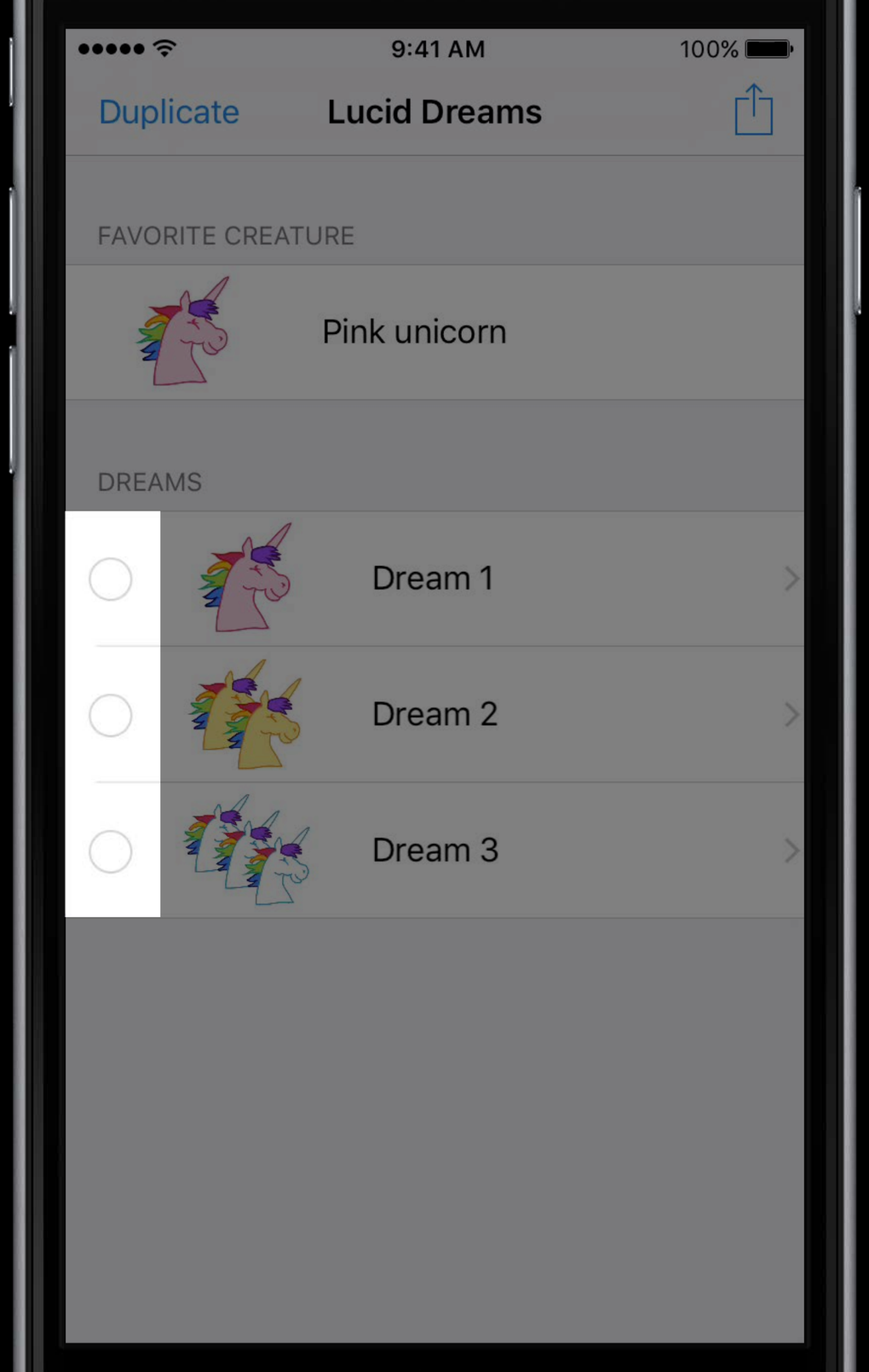

viewing
selecting
sharing

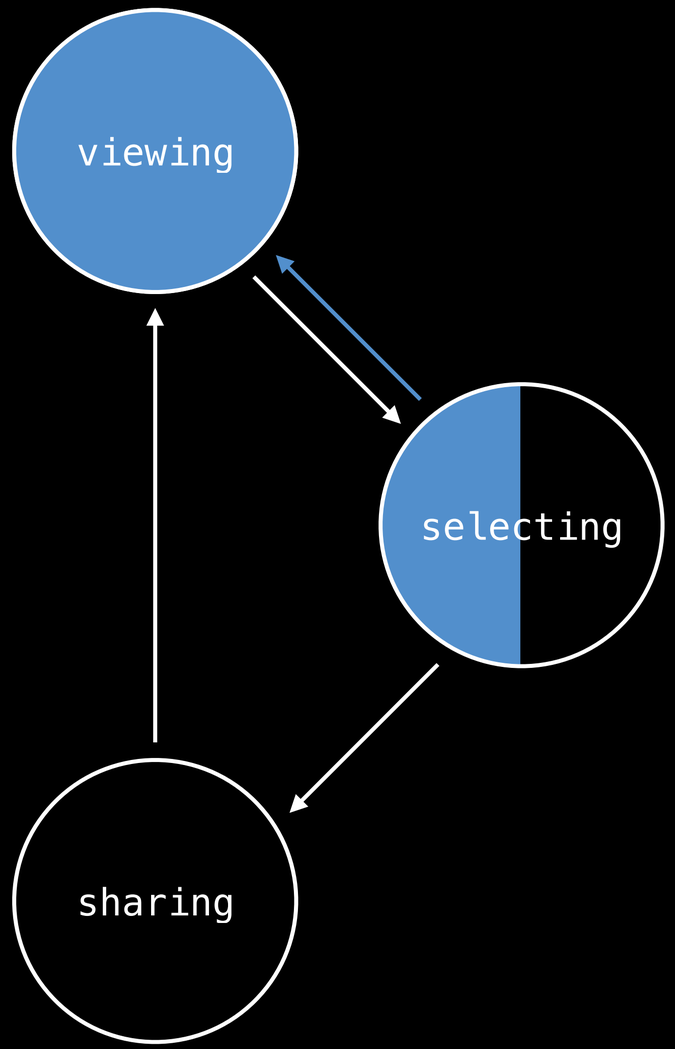

9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

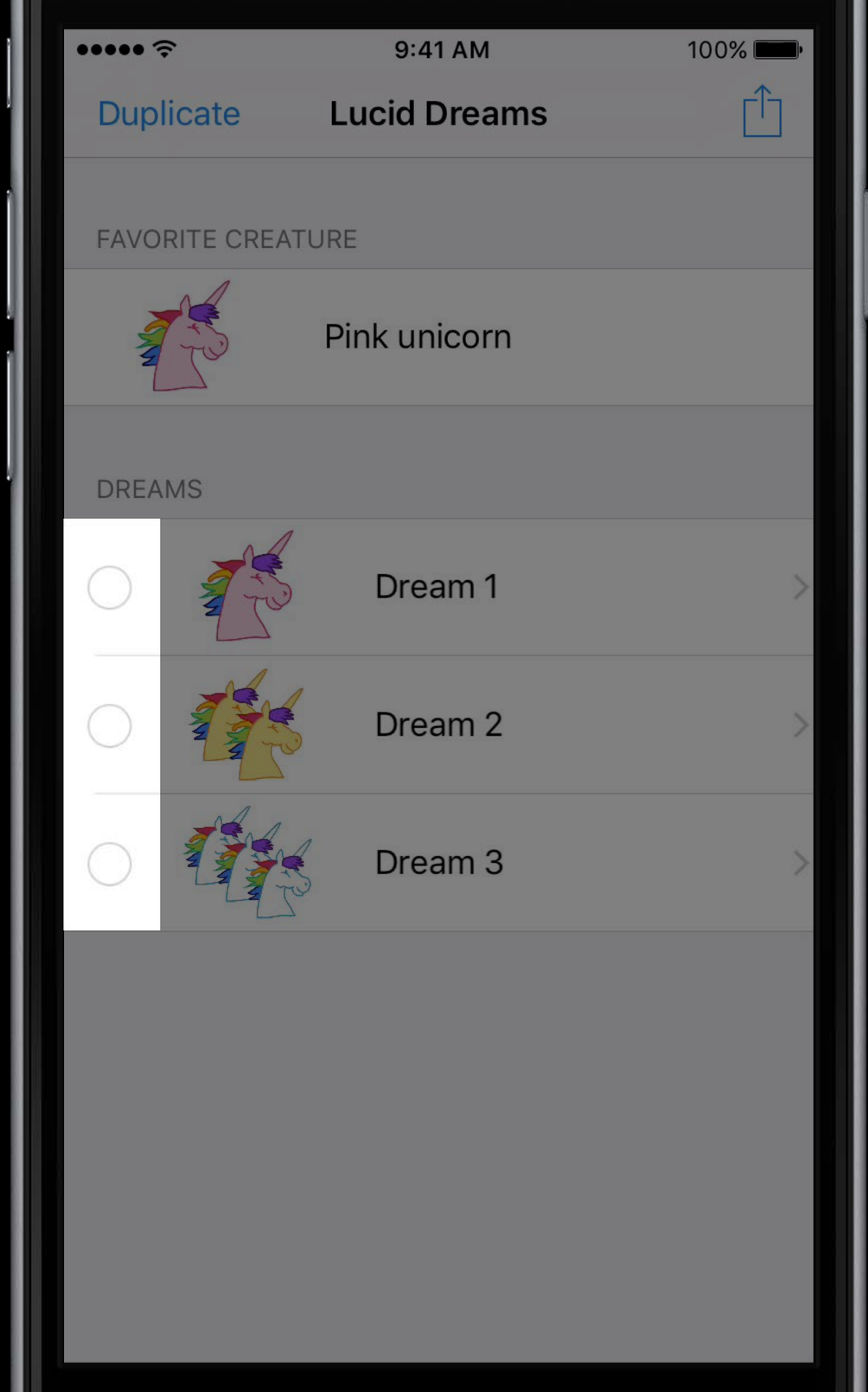

viewing
selecting
sharing

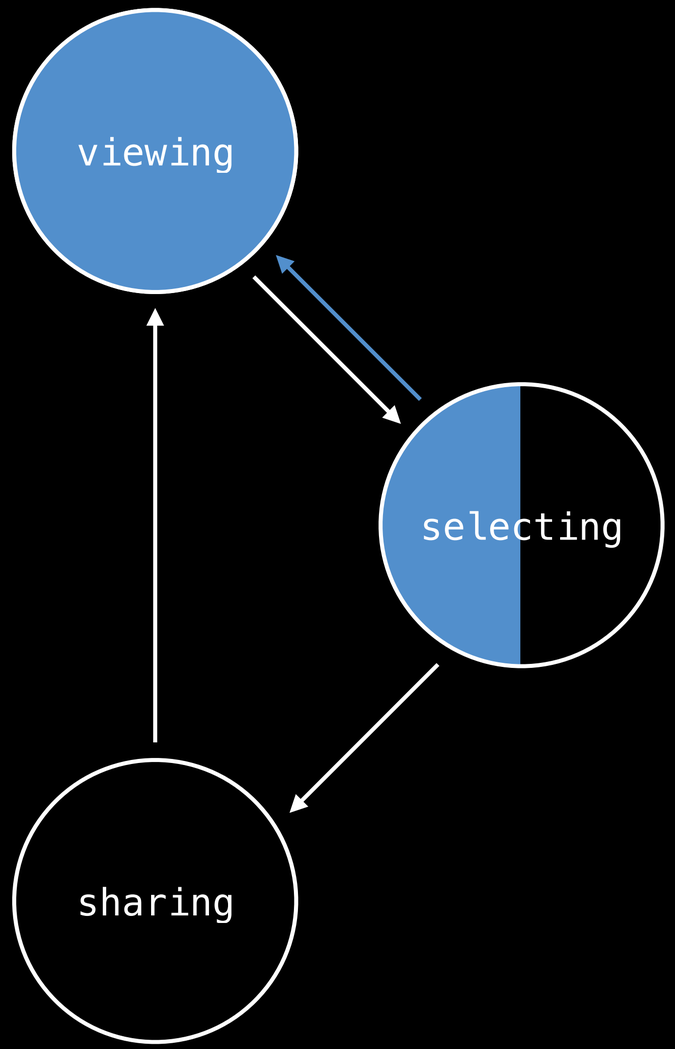

9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

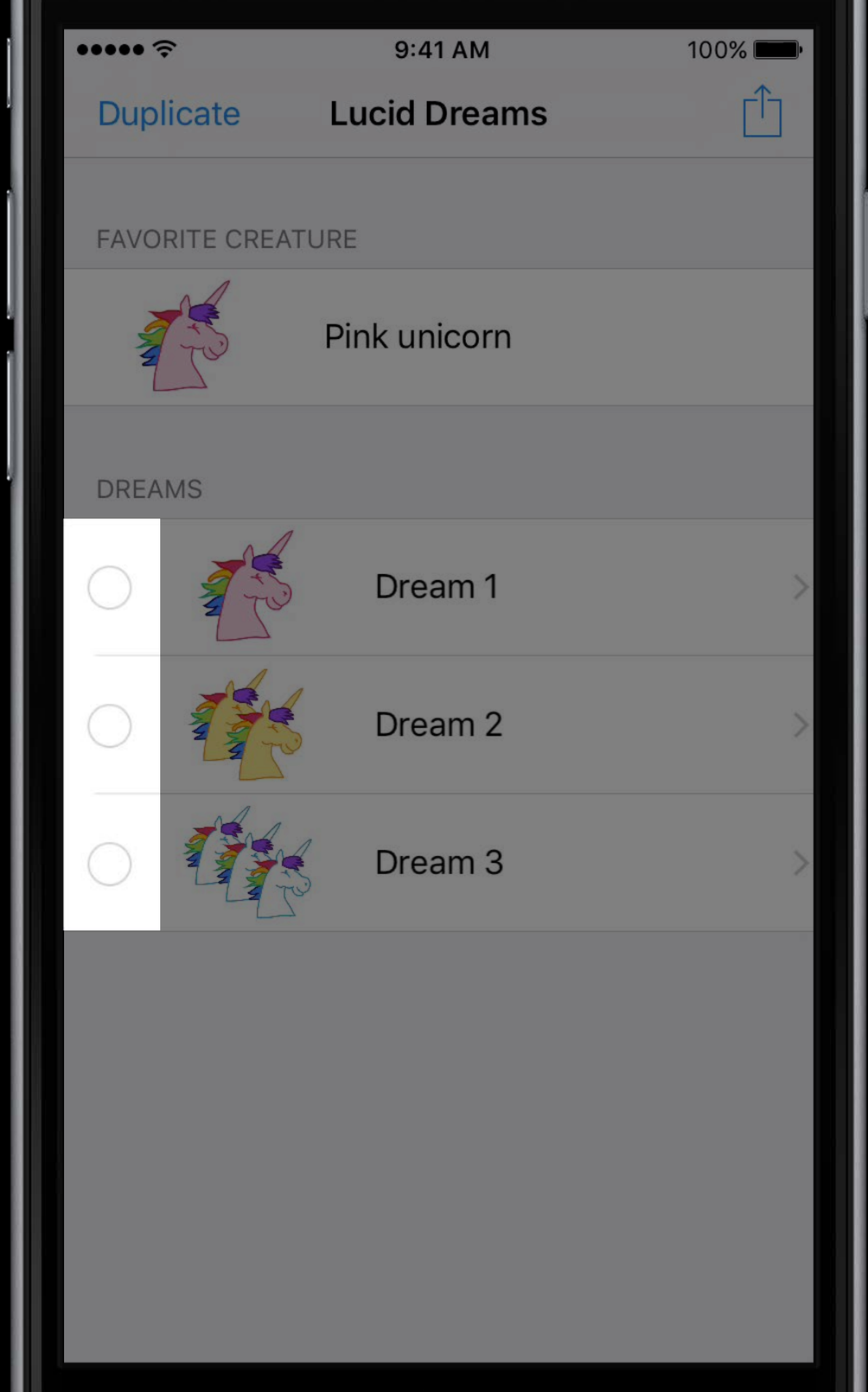

viewing
selecting
sharing

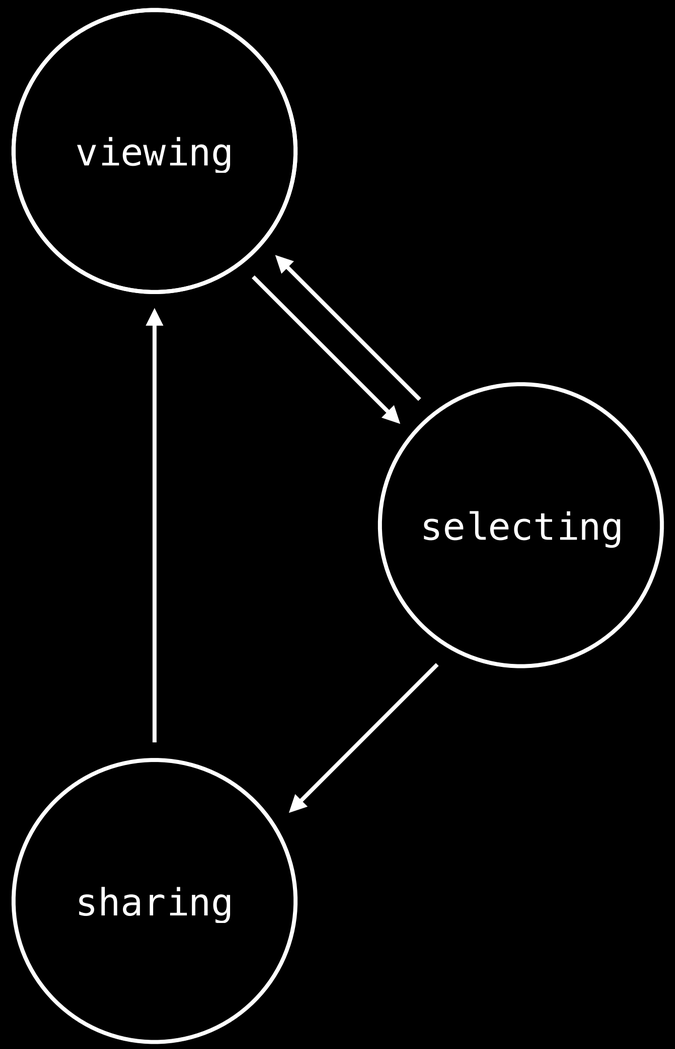

9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3

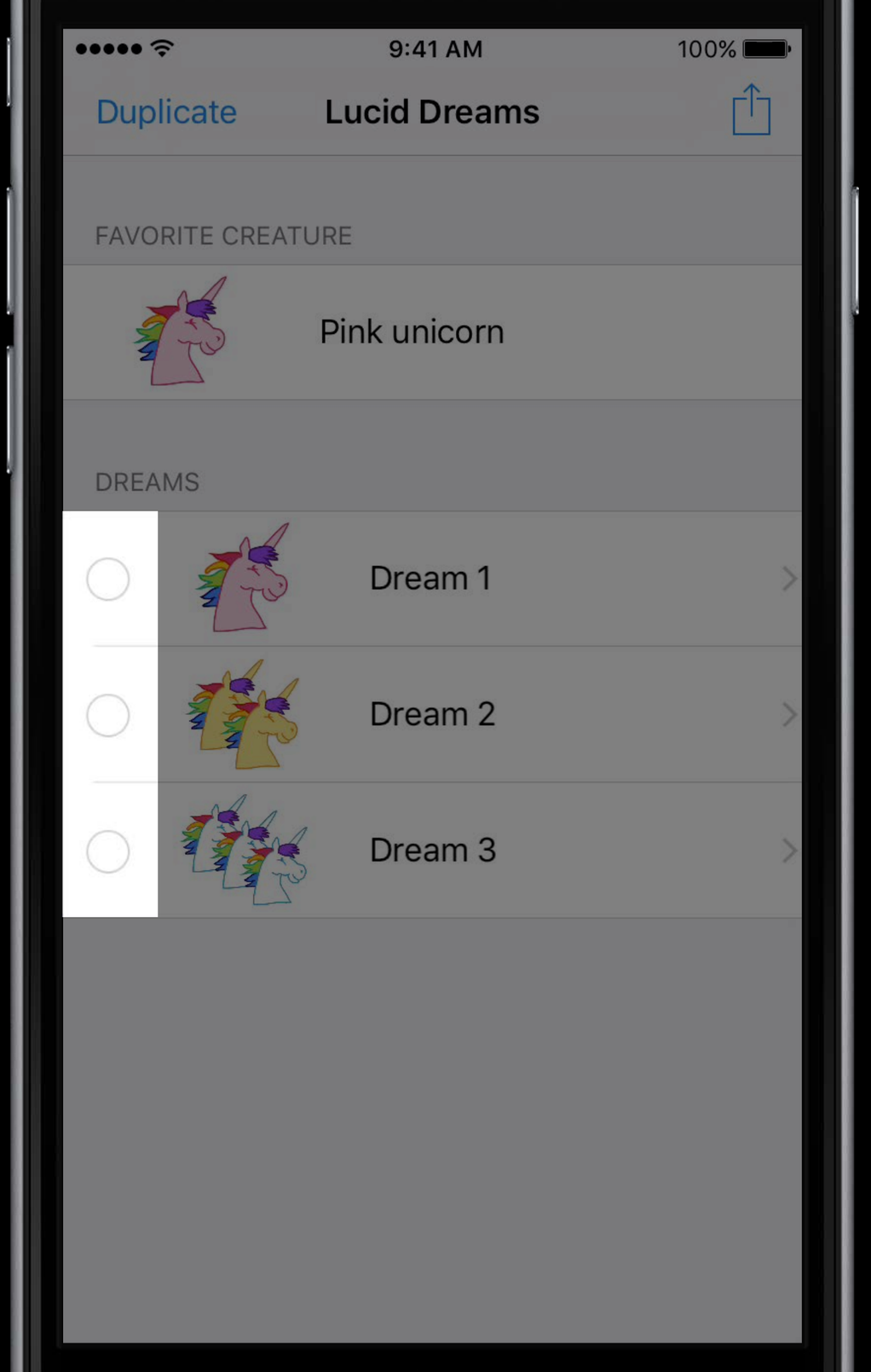

viewing
selecting
sharing

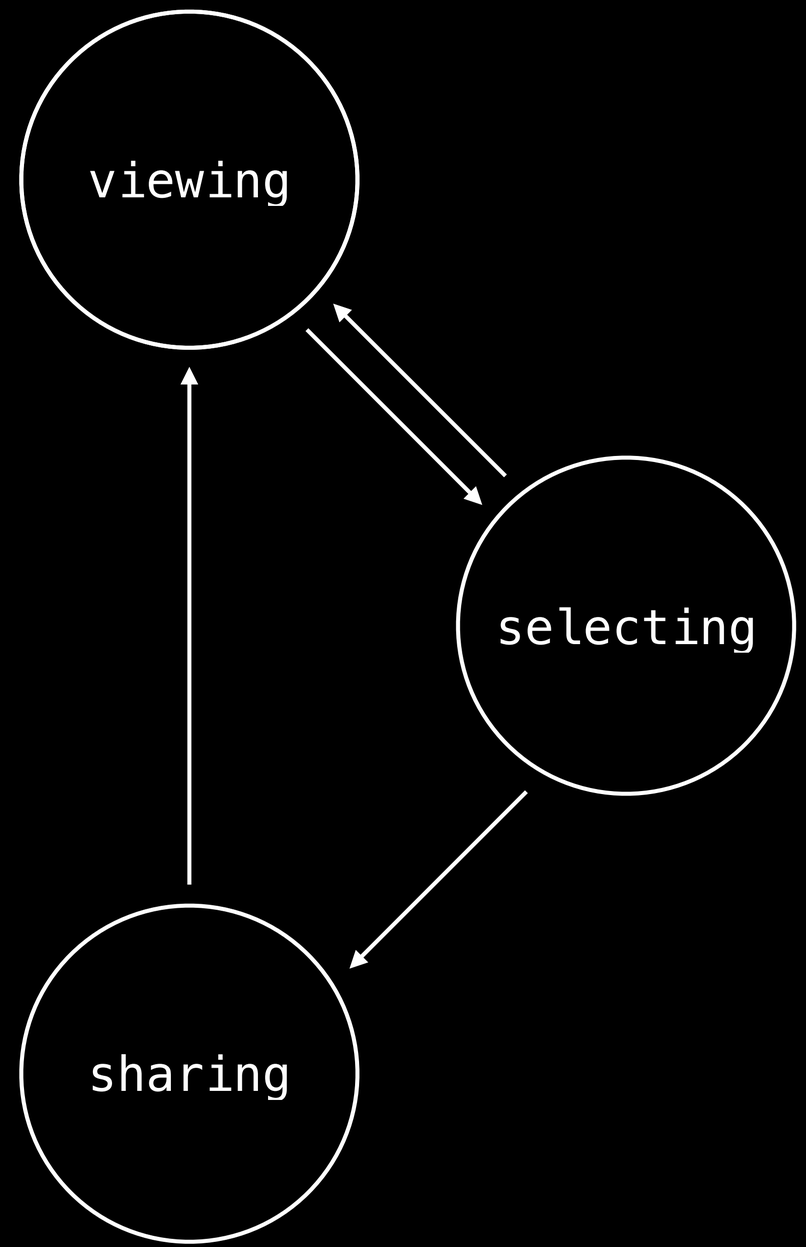

9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3


viewing
selecting
sharing


9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3


viewing
selecting
sharing


9:41 AM
100%
Duplicate
Lucid Dreams
FAVORITE CREATURE
Pink unicorn
DREAMS
Dream 1
Dream 2
Dream 3


viewing
selecting
sharing

viewing

selecting

sharing

```
var isInViewingMode: Bool
```

```
var selectedRows: IndexSet?
```

```
var sharingDreams: [Dream]?
```

```
// DreamListViewController – Invalid UI State Bug

class DreamListViewController : UITableViewController {

                                    var isInViewingMode: Bool



    ...
}
                                                          var selectedRows: IndexSet?


var sharingDreams: [Dream]?
```

```
// DreamListViewController – Invalid UI State Bug

class DreamListViewController : UITableViewController {
    var isInViewingMode: Bool
    var sharingDreams: [Dream]?
    var selectedRows: IndexSet?

    ...
}
```

```
// DreamListViewController – Invalid UI State Bug

class DreamListViewController : UITableViewController {
    var isInViewingMode: Bool
    var sharingDreams: [Dream]?
    var selectedRows: IndexSet?

    ...
}
```

```
// DreamListViewController – Isolating UI State

class DreamListViewController : UITableViewController {
    var isInViewingMode: Bool
    var sharingDreams: [Dream]?
    var selectedRows: IndexSet?

    ...
}
```

```
    enum State {




    }
```

```
// DreamListViewController – Isolating UI State

class DreamListViewController : UITableViewController {
    var isInViewingMode: Bool
    var sharingDreams: [Dream]?
    var selectedRows: IndexSet?

    ...
}
```

```
    enum State {




    }
```

```
// DreamListViewController – Isolating UI State

class DreamListViewController : UITableViewController {




    ...
}
```

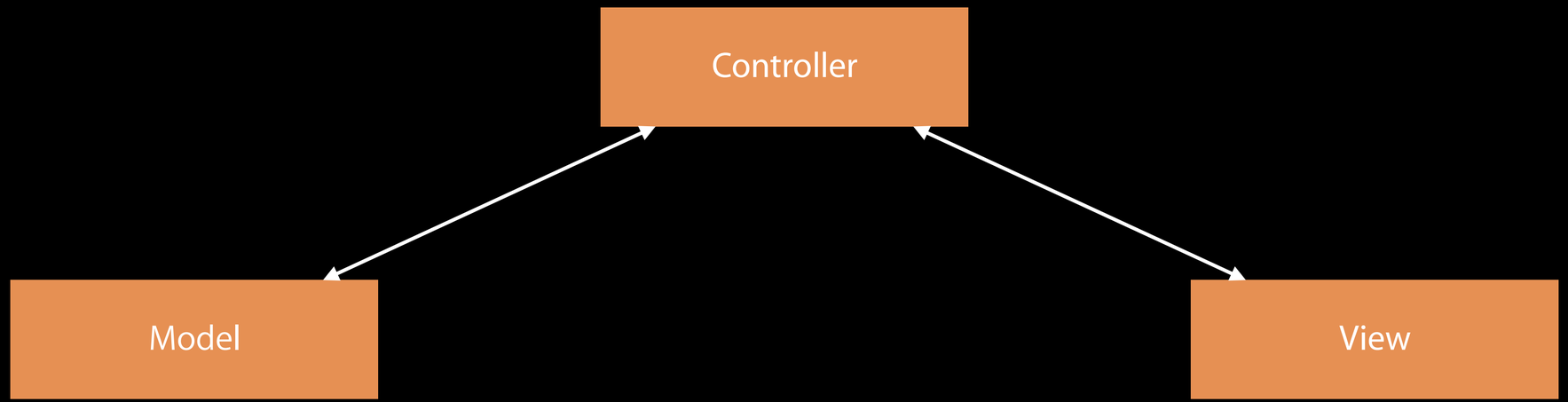
```
    enum State {
        case viewing
        case sharing(dreams: [Dream])
        case selecting(selectedRows: IndexSet)
    }
```

```
// DreamListViewController – Isolating UI State

class DreamListViewController : UITableViewController {
    var state: State



    ...
}
```

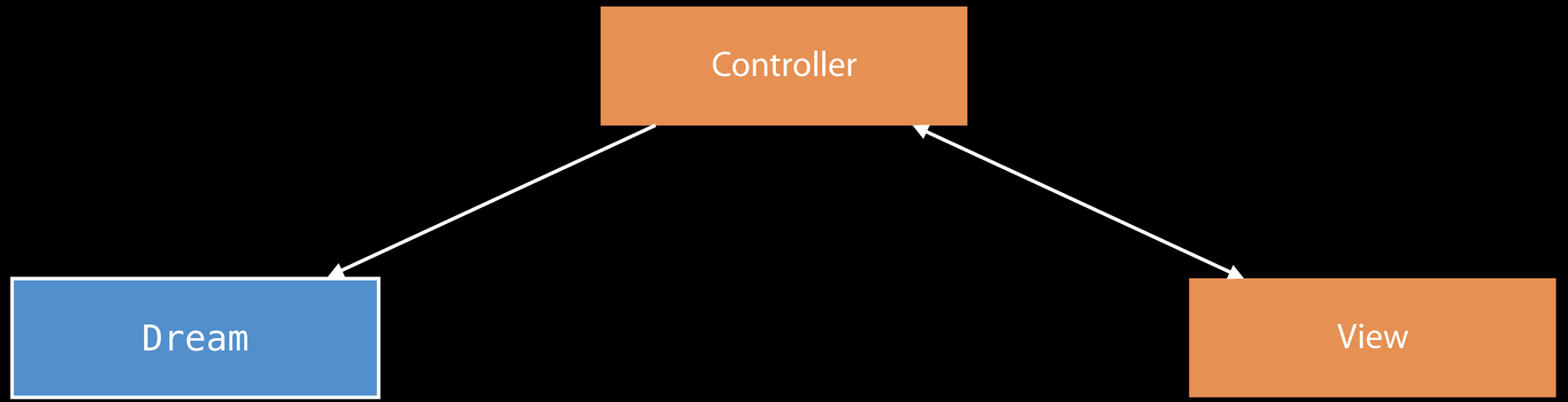
```
enum State {
    case viewing
    case sharing(dreams: [Dream])
    case selecting(selectedRows: IndexSet)
}
```

Recap

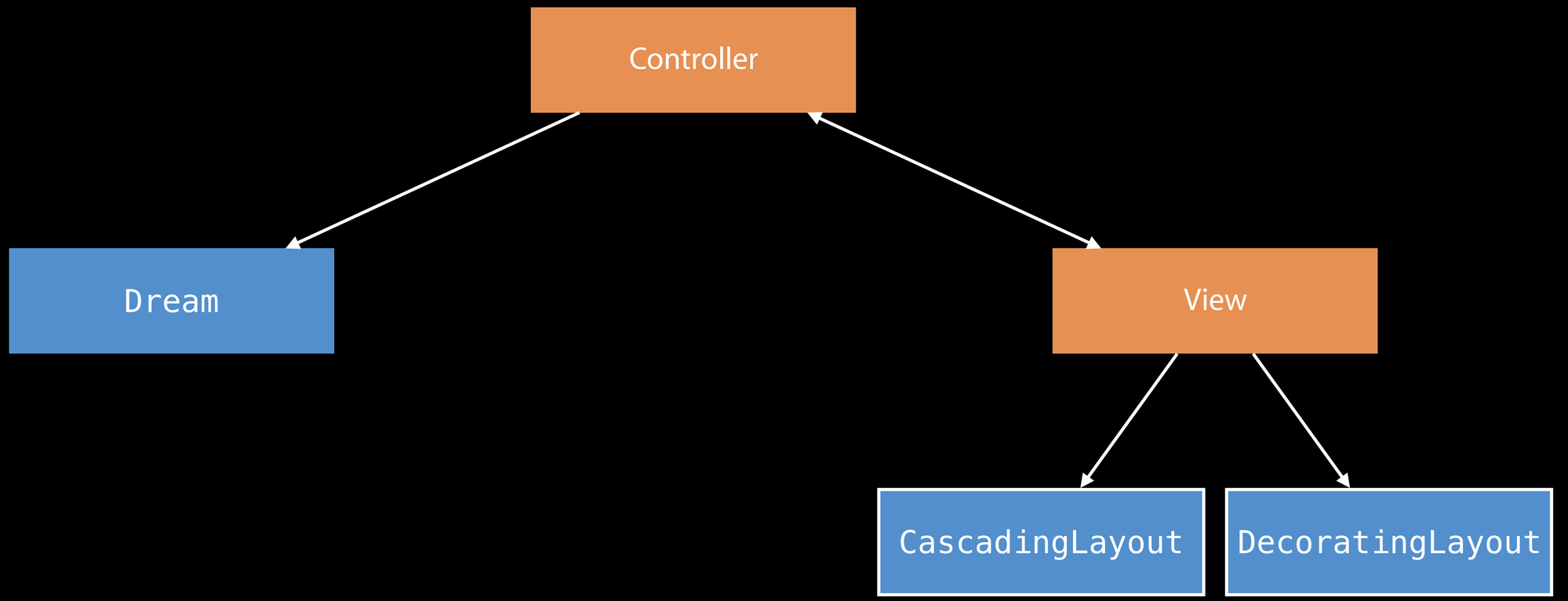
Model View Controller



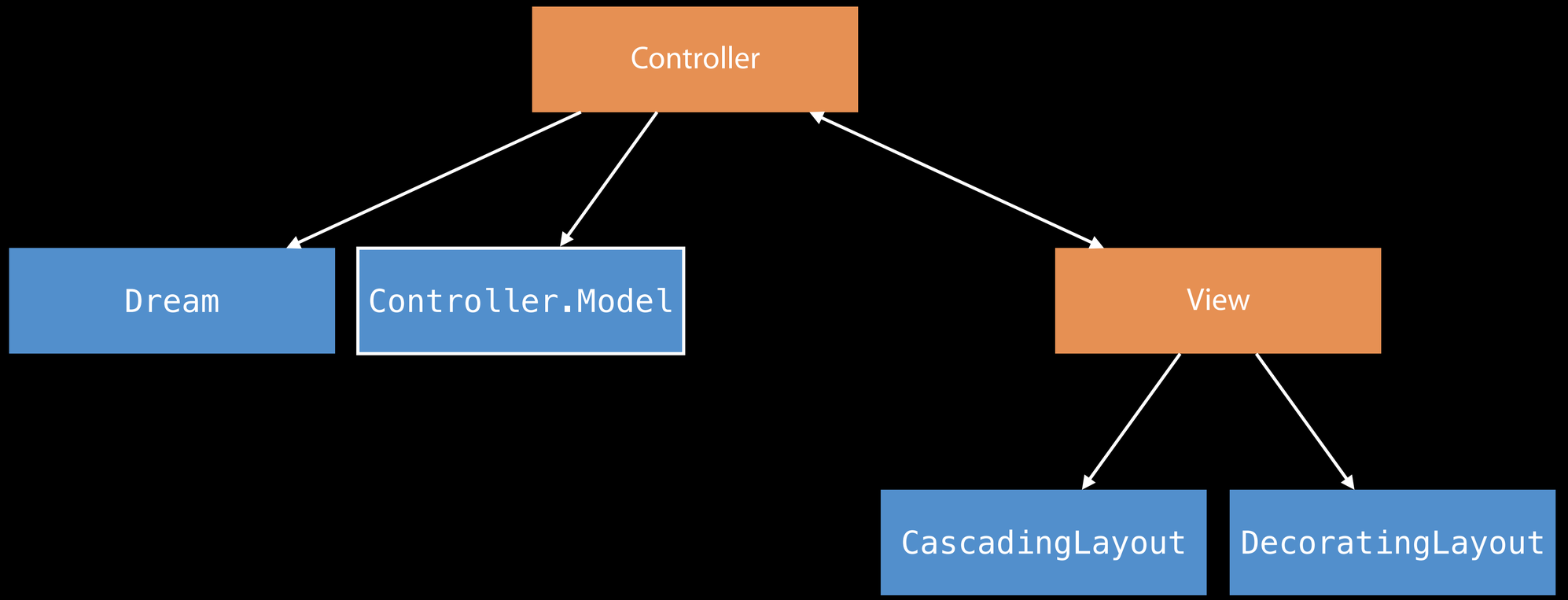
Model View Controller



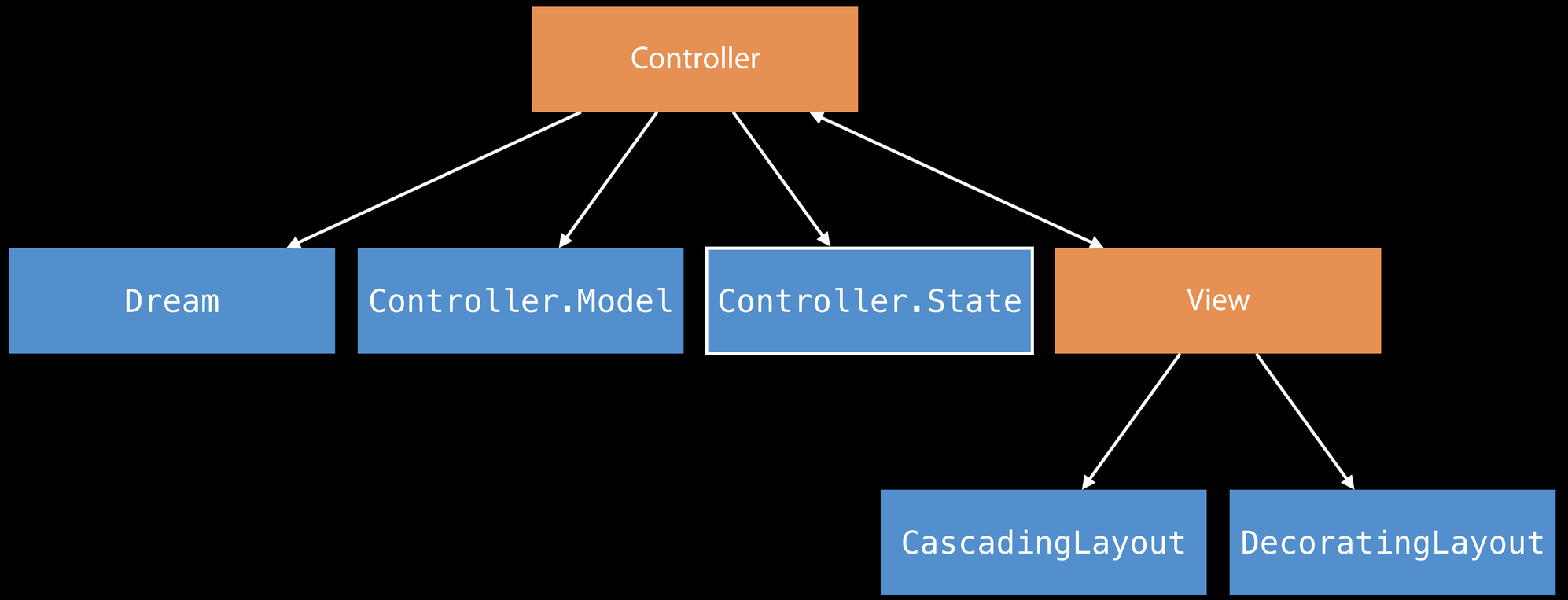
Model View Controller



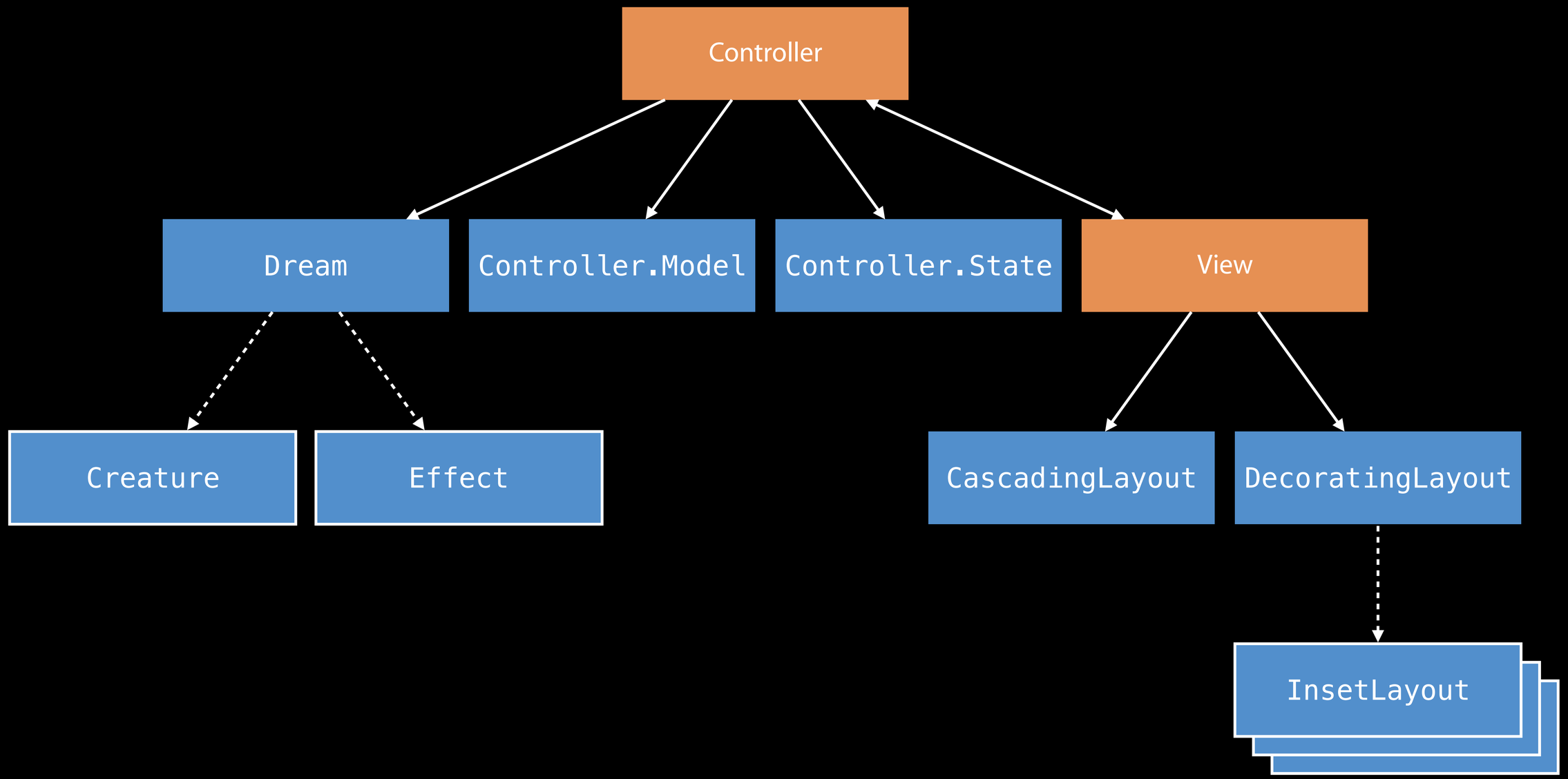
Model View Controller



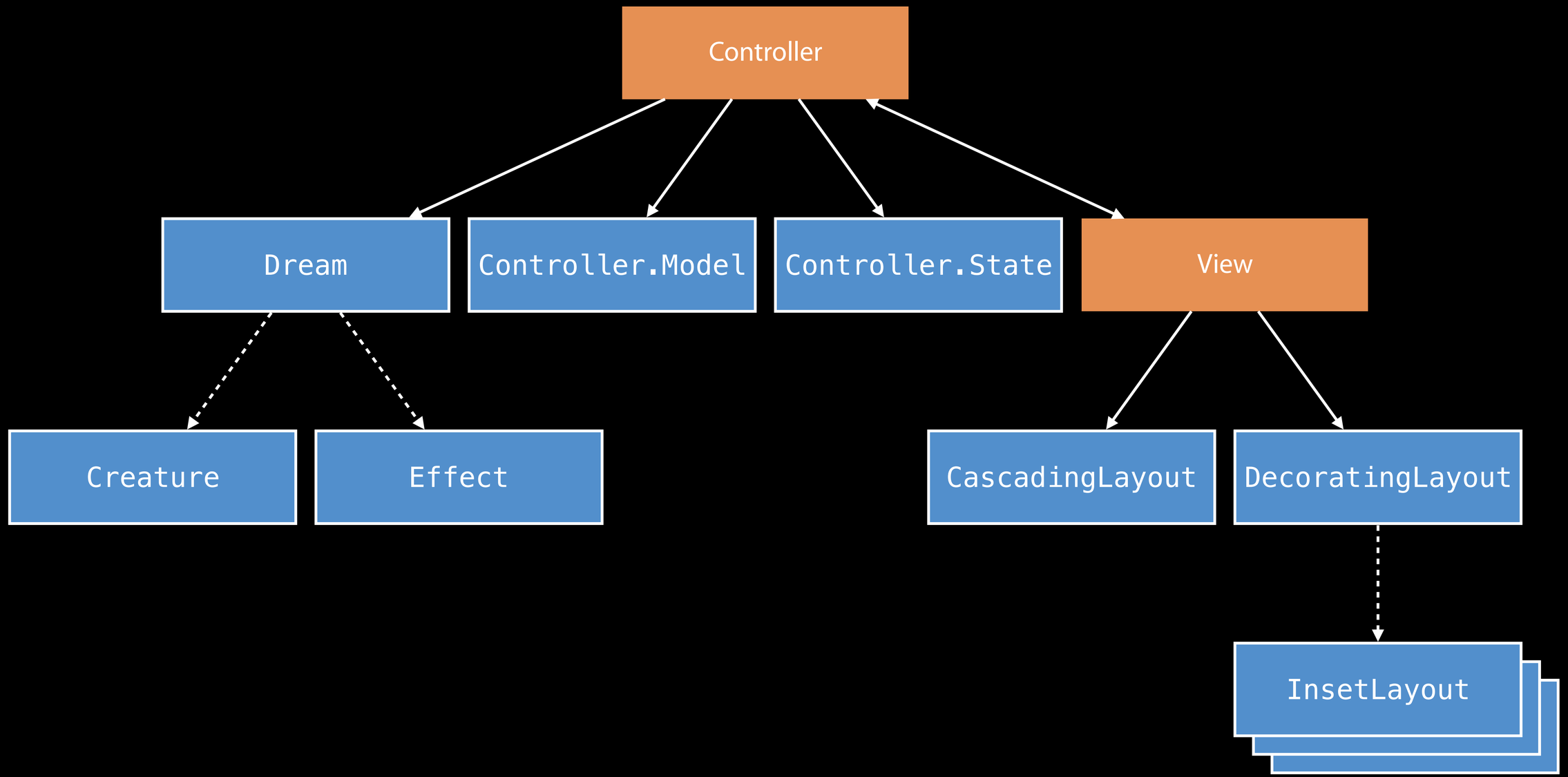
Model View Controller



Model View Controller



Model View Controller



Techniques and Tools

Techniques and Tools

Customization through composition

Techniques and Tools

Customization through composition

Protocols for generic, reusable code

Techniques and Tools

Customization through composition

Protocols for generic, reusable code

Taking advantage of value semantics

Techniques and Tools

Customization through composition

Protocols for generic, reusable code

Taking advantage of value semantics

Local reasoning

More Information

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

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

WWDC16