#WWDC19

Implementing Dark Mode in iOS

Kurt Revis, UIKit Engineer
Tyler Fox, UIKit Engineer

© 2019 Apple Inc. All rights reserved. Redistribution or public display not permitted without written permission from Apple.

Dark Mode is a new look

It’s easy to implement

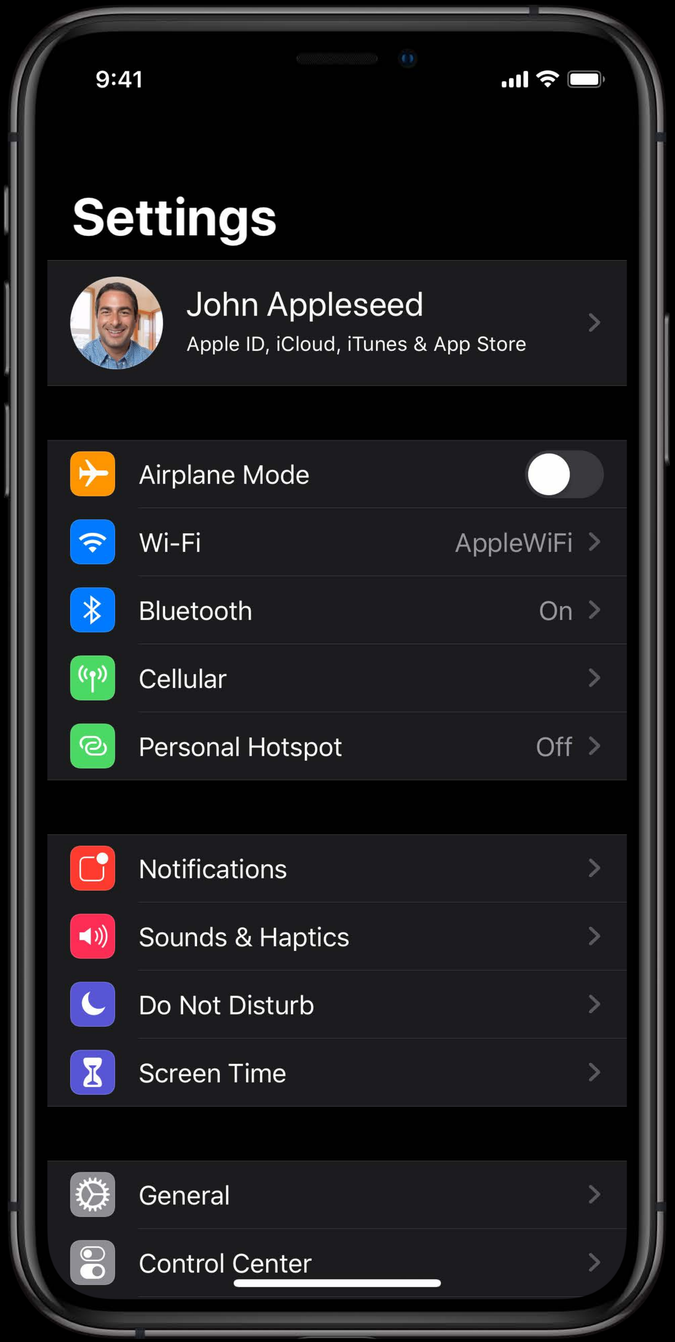
Flexible and powerful

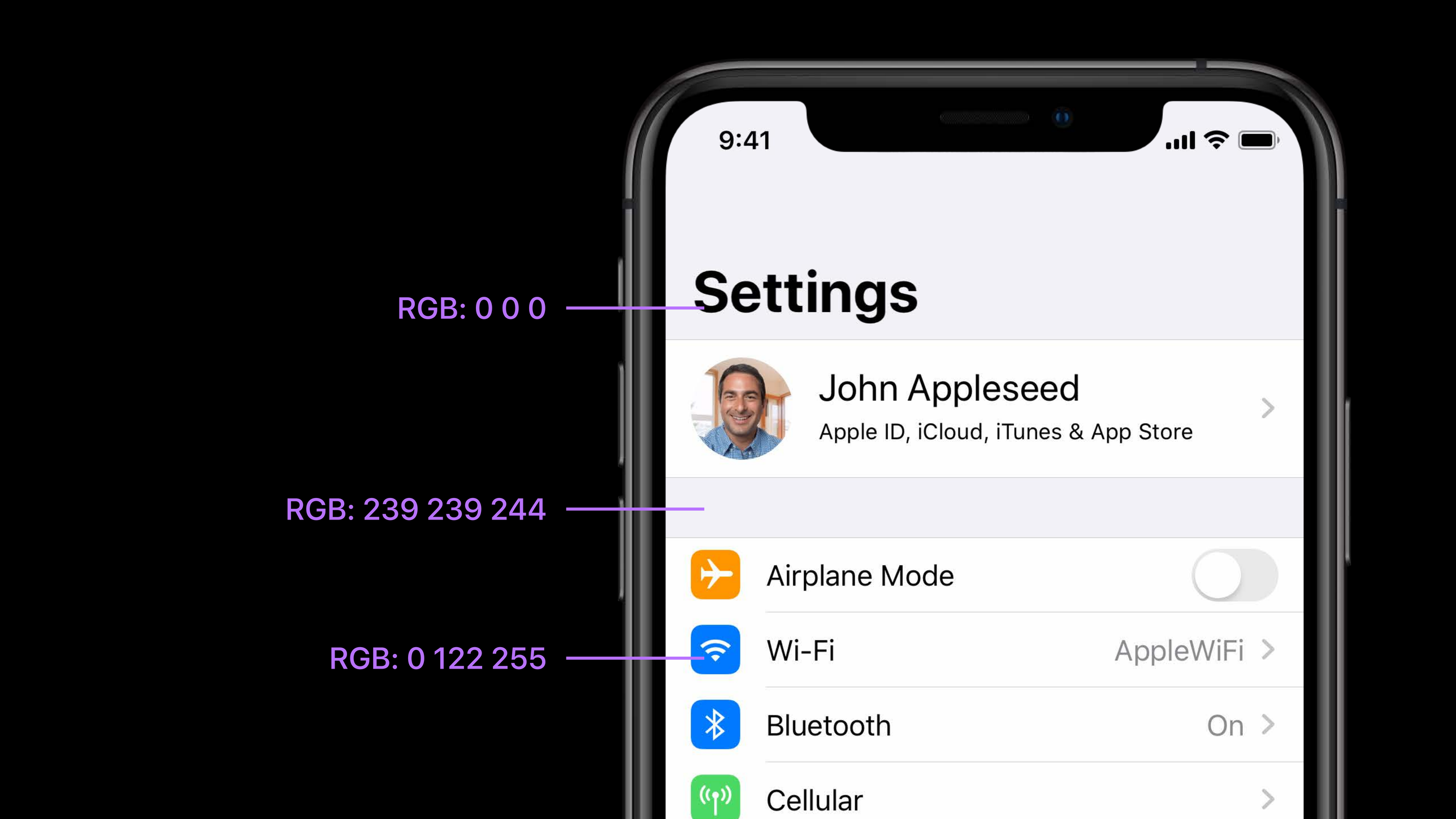
Dark Mode is a new look

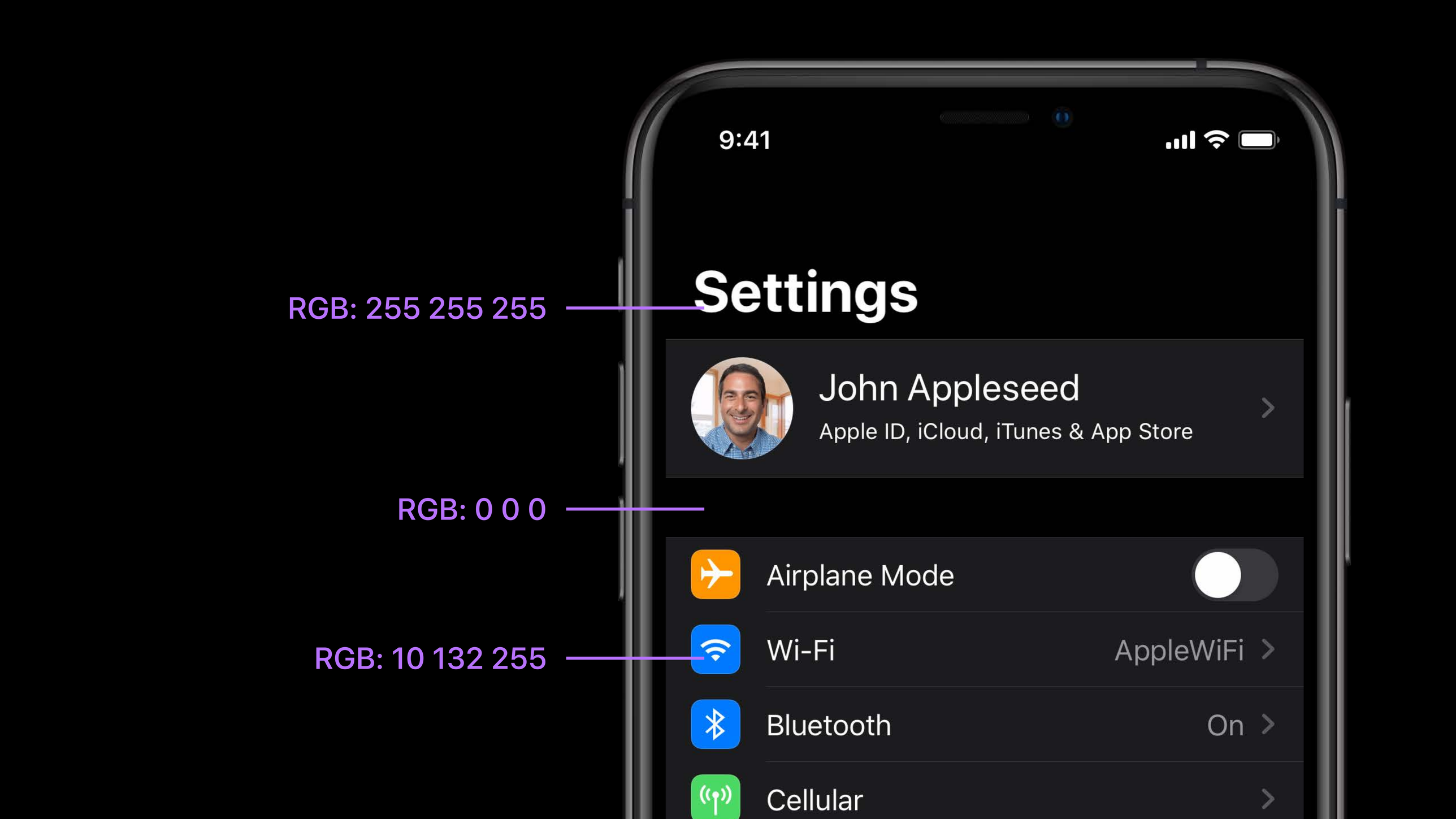
It’s easy to implement

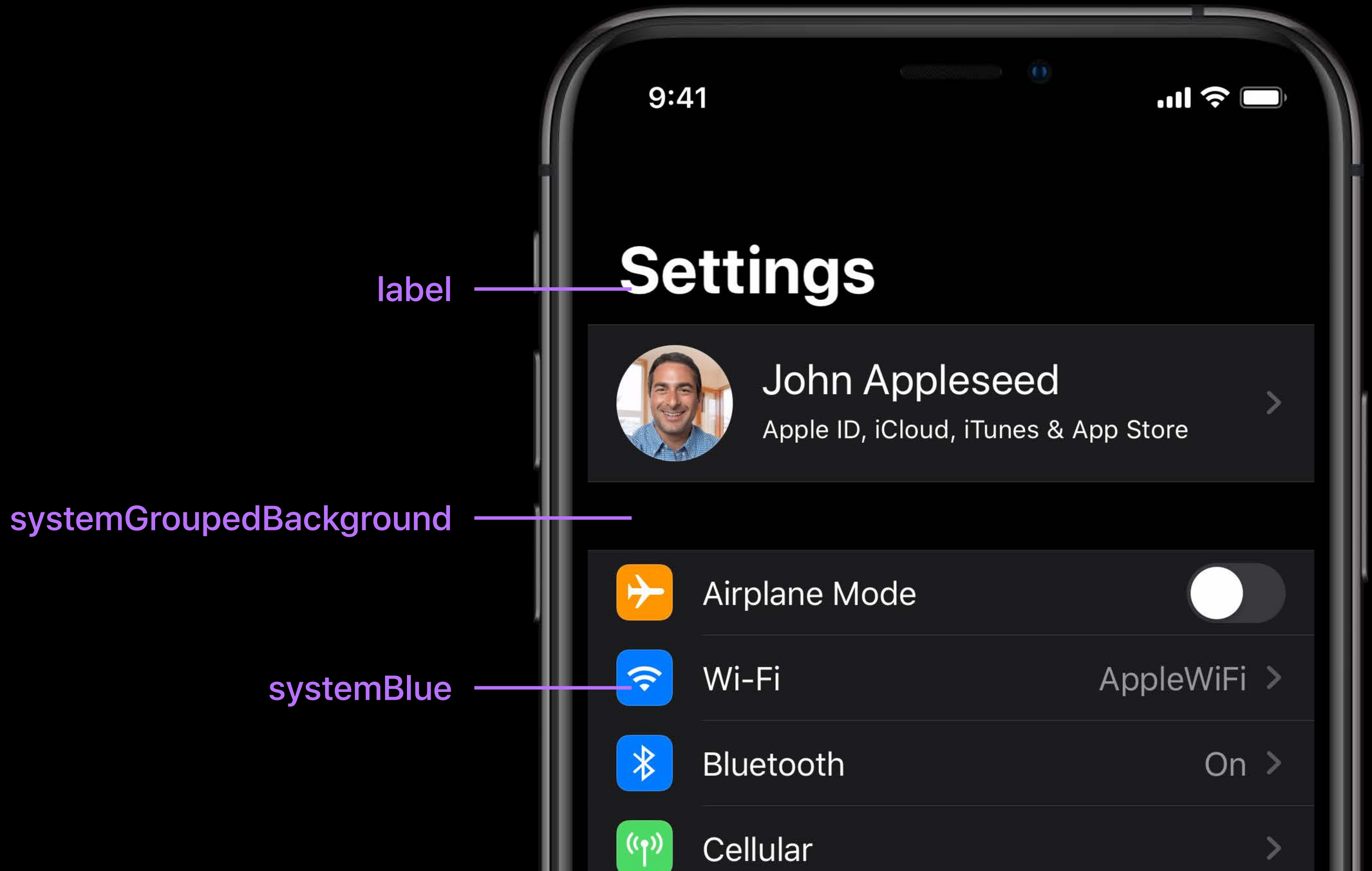
Flexible and powerful

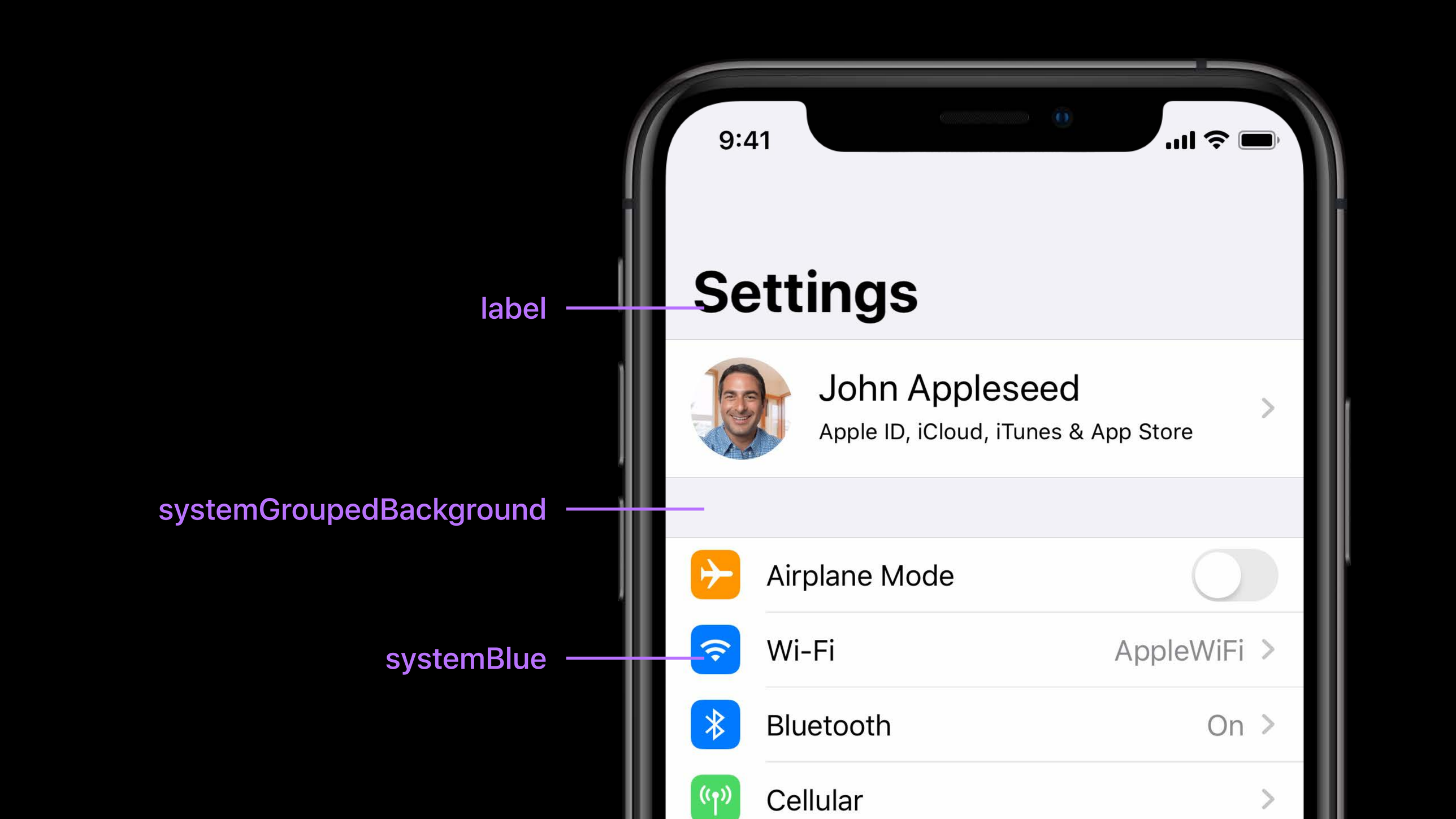

9:41
Settings
John Appleseed
Apple ID, iCloud, iTunes & App Store
Airplane Mode
Wi-Fi
AppleWiFi
Bluetooth
On
Cellular
Personal Hotspot
Off
Notifications
Sounds & Haptics
Do Not Disturb
Screen Time
General
Control Center


9:41
Settings
John Appleseed
Apple ID, iCloud, iTunes & App Store
Airplane Mode
Wi-Fi
AppleWiFi
Bluetooth
On
Cellular
Personal Hotspot
Off
Notifications
Sounds & Haptics
Do Not Disturb
Screen Time
General
Control Center

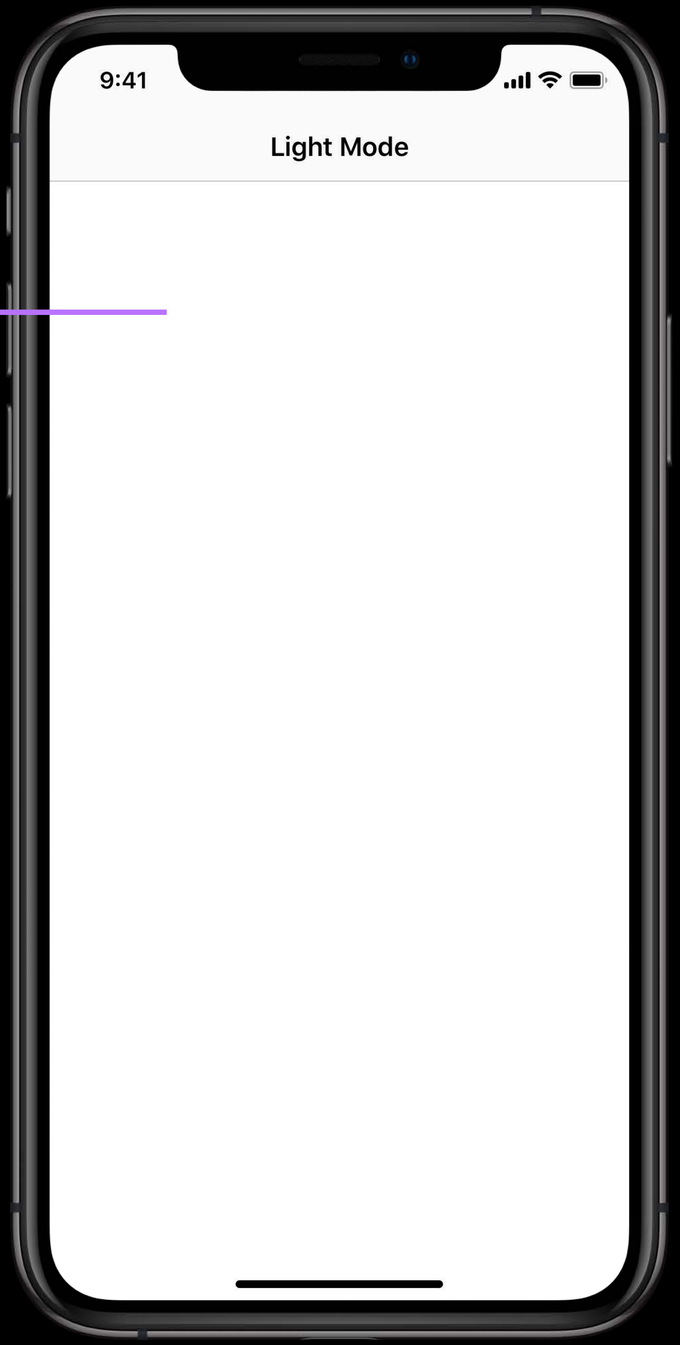

RGB: 0 0 0
RGB: 239 239 244
RGB: 0 122 255
9:41
Settings
John Appleseed
Apple ID, iCloud, iTunes & App Store
Airplane Mode
Wi-Fi
AppleWiFi
Bluetooth
On
Cellular

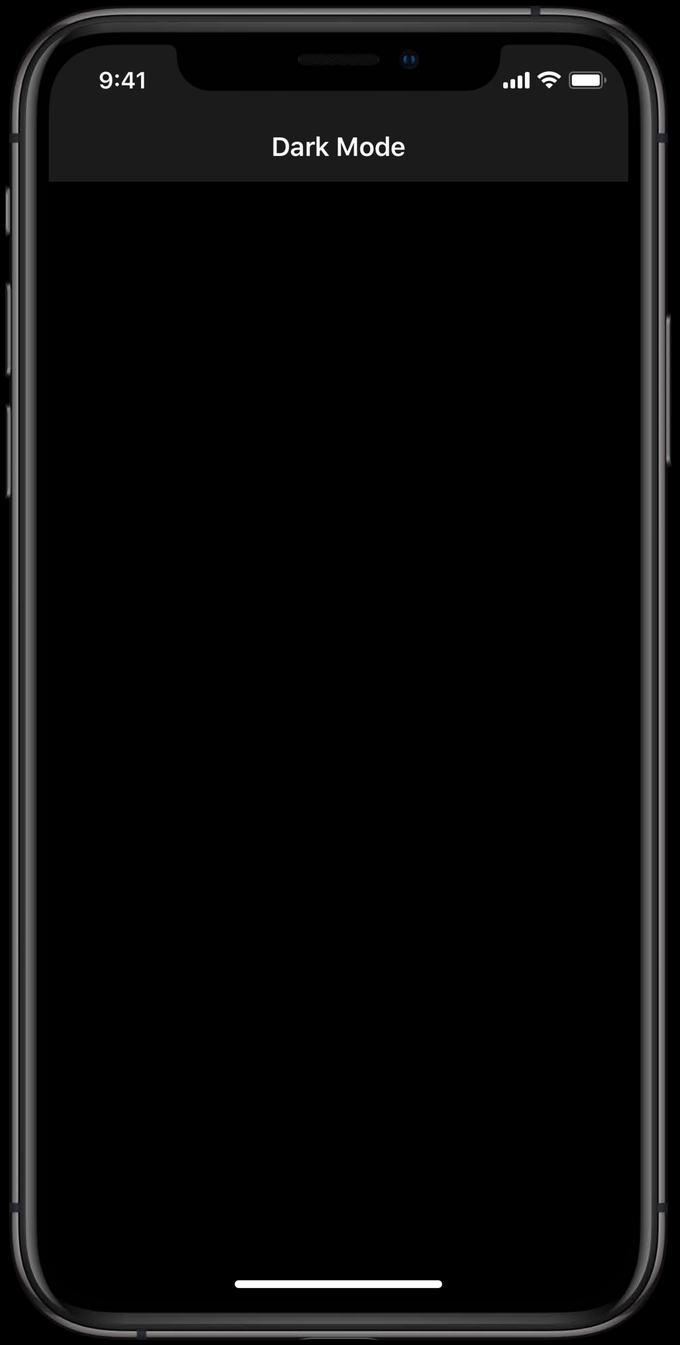

RGB: 255 255 255
RGB: 0 0 0
RGB: 10 132 255
9:41
Settings
John Appleseed
Apple ID, iCloud, iTunes & App Store
Airplane Mode
Wi-Fi
AppleWiFi
Bluetooth
On
Cellular

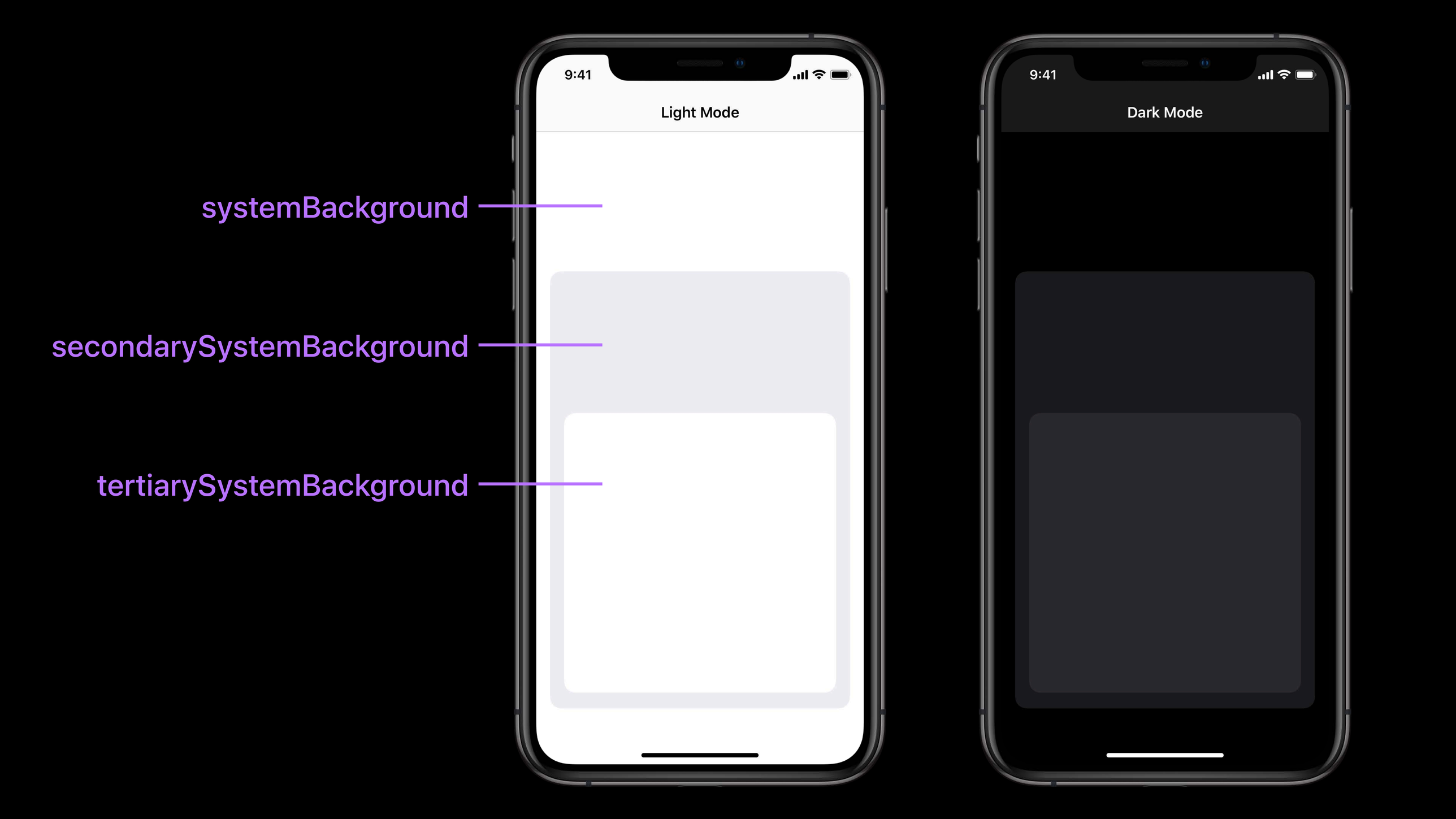

label
systemGroupedBackground
systemBlue
9:41
Settings
John Appleseed
Apple ID, iCloud, iTunes & App Store
Airplane Mode
Wi-Fi
AppleWiFi
Bluetooth
On
Cellular


label
systemGroupedBackground
systemBlue
9:41
Settings
John Appleseed
Apple ID, iCloud, iTunes & App Store
Airplane Mode
Wi-Fi
AppleWiFi
Bluetooth
On
Cellular

systemBackground






systemBackground
secondarySystemBackground
tertiarySystemBackground
9:41
Light Mode
9:41
Dark Mode

Primary

Secondary

Tertiary

Quaternary

Secondary

Tertiary

Quaternary

Title

Subtitle

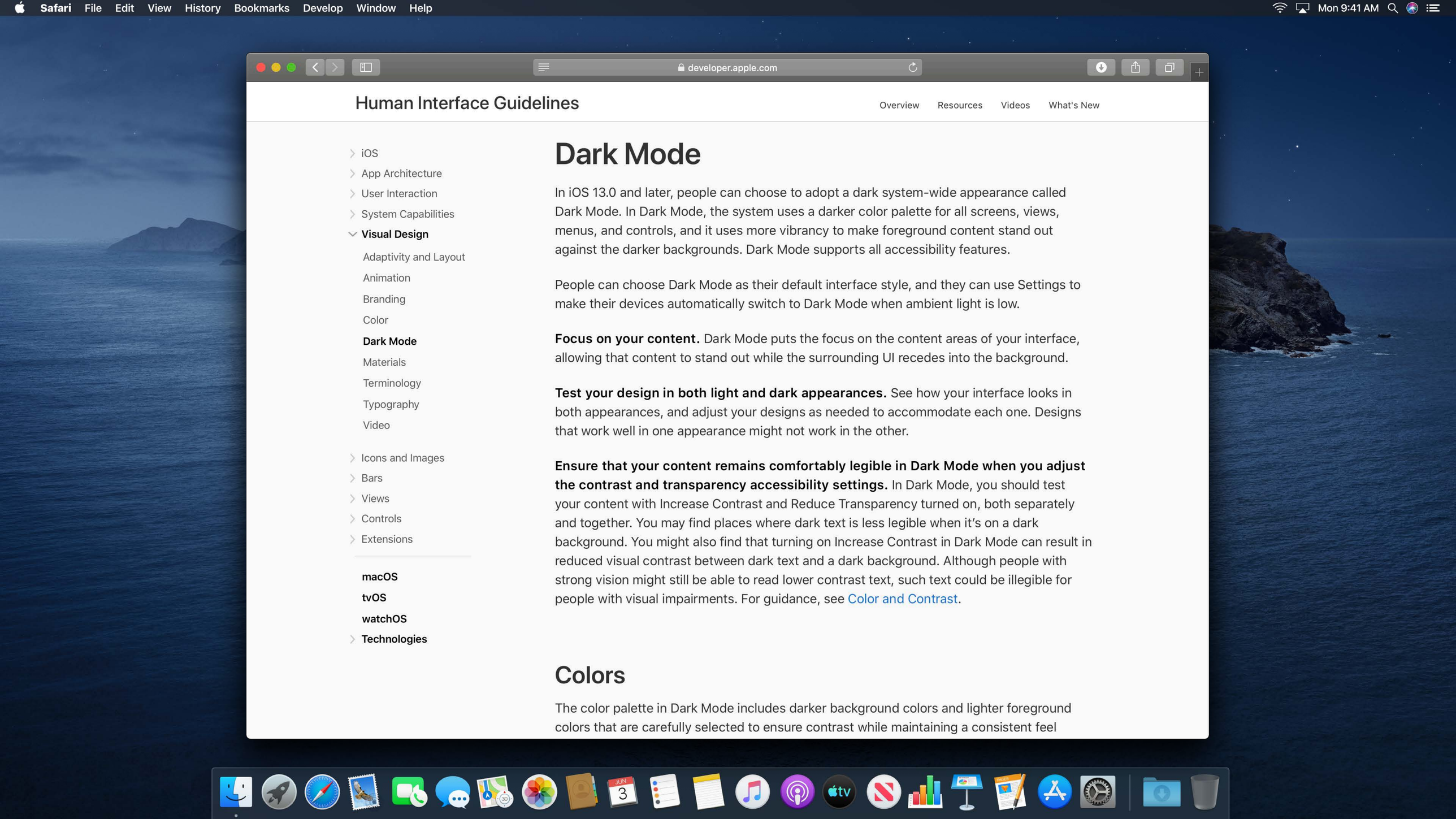
Placeholder

Disabled

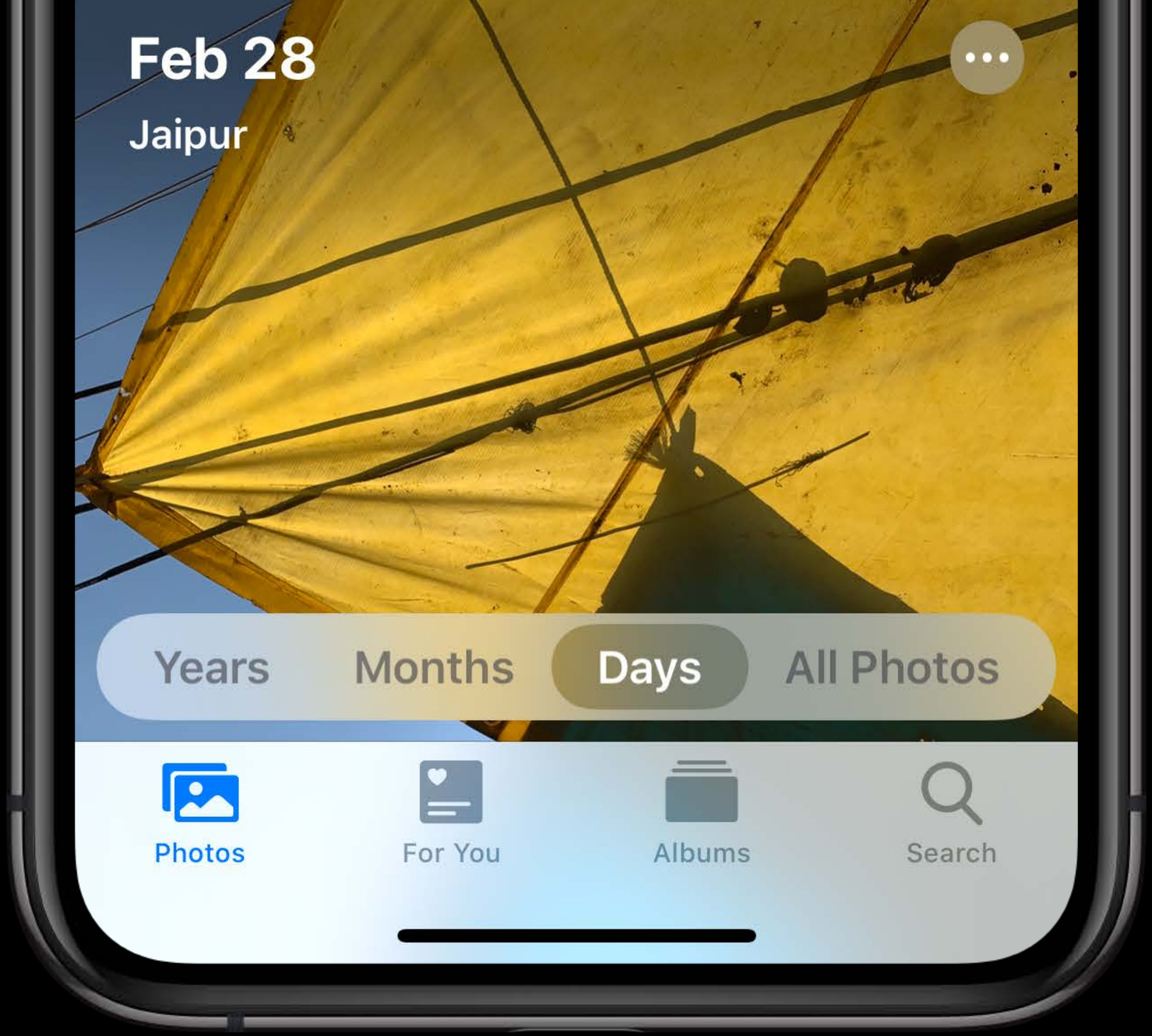
Subtitle

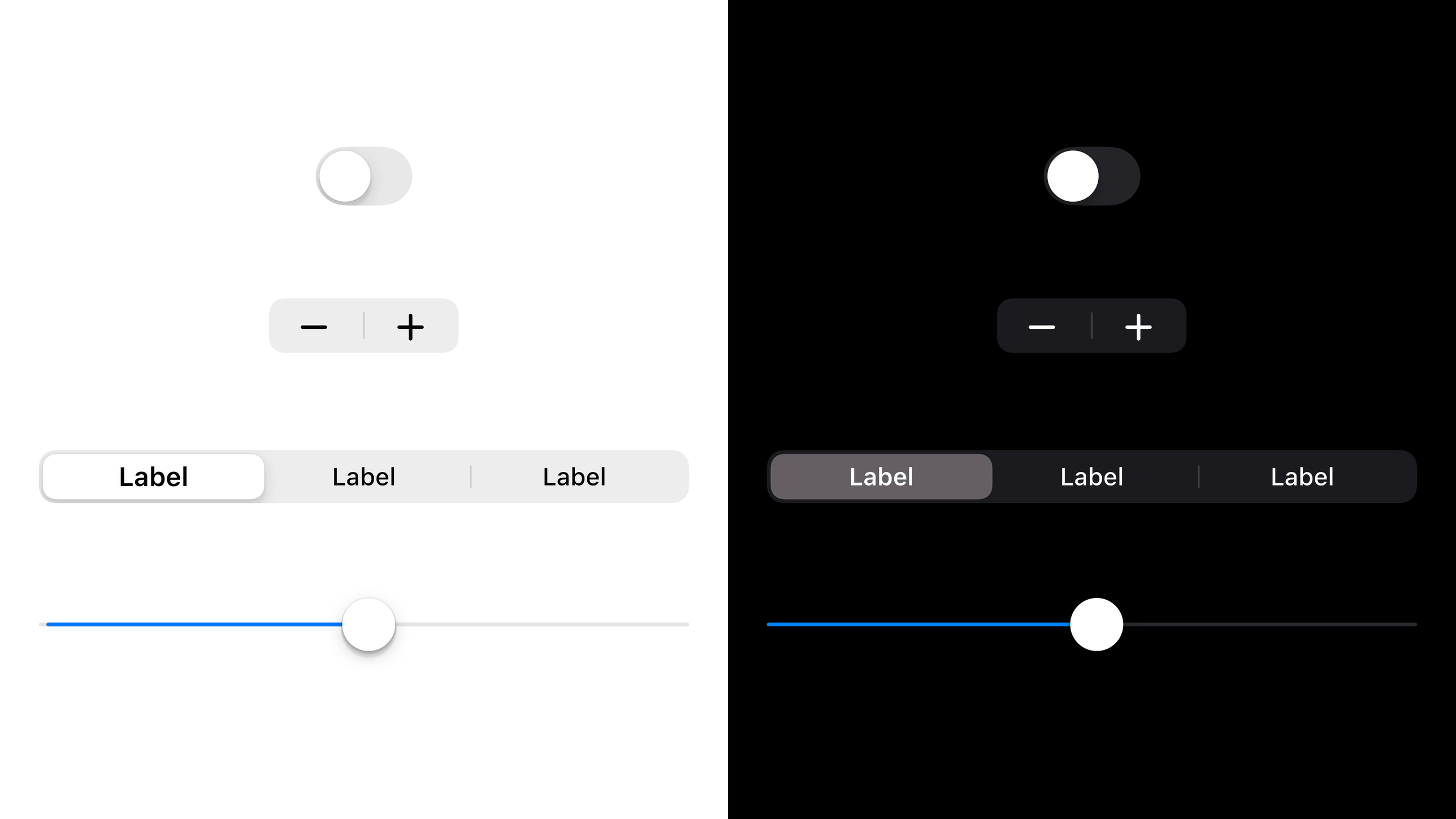
Placeholder

Disabled


Safari File Edit View History Bookmarks Develop Window Help
Mon 9:41 AM
developer.apple.com
Human Interface Guidelines
Overview Resources Videos What's New
iOS
App Architecture
User Interaction
System Capabilities
Visual Design
Adaptivity and Layout
Animation
Branding
Color
Dark Mode
Materials
Terminology
Typography
Video
Icons and Images
Bars
Views
Controls
Extensions
macOS
tvOS
watchOS
Technologies
Dark Mode
In iOS 13.0 and later, people can choose to adopt a dark system-wide appearance called Dark Mode. In Dark Mode, the system uses a darker color palette for all screens, views, menus, and controls, and it uses more vibrancy to make foreground content stand out against the darker backgrounds. Dark Mode supports all accessibility features.
People can choose Dark Mode as their default interface style, and they can use Settings to make their devices automatically switch to Dark Mode when ambient light is low.
Focus on your content. Dark Mode puts the focus on the content areas of your interface, allowing that content to stand out while the surrounding UI recedes into the background.
Test your design in both light and dark appearances. See how your interface looks in both appearances, and adjust your designs as needed to accommodate each one. Designs that work well in one appearance might not work in the other.
Ensure that your content remains comfortably legible in Dark Mode when you adjust the contrast and transparency accessibility settings. In Dark Mode, you should test your content with Increase Contrast and Reduce Transparency turned on, both separately and together. You may find places where dark text is less legible when it's on a dark background. You might also find that turning on Increase Contrast in Dark Mode can result in reduced visual contrast between dark text and a dark background. Although people with strong vision might still be able to read lower contrast text, such text could be illegible for people with visual impairments. For guidance, see Color and Contrast.
Colors
The color palette in Dark Mode includes darker background colors and lighter foreground colors that are carefully selected to ensure contrast while maintaining a consistent feel


Feb 28
Jaipur
Years
Months
Days
All Photos
Photos
For You
Albums
Search


Feb 28
Jaipur
Years
Months
Days
All Photos
Photos
For You
Albums
Search


Label
Label
Label
Label
Label
Label

Designing for Dark Mode

Use UIKit colors, materials, views, and controls

Customize colors and images when necessary

Dark Mode is a new look

It’s easy to implement

Flexible and powerful

Implementing Dark Mode

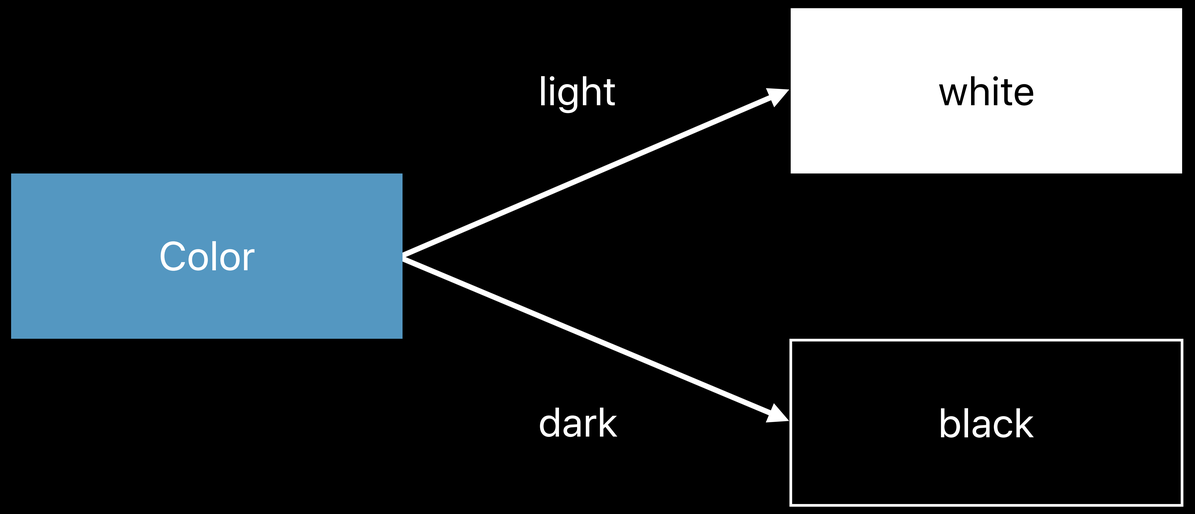
Using iOS 13 SDK implies Dark Mode support

You decide your app’s appearance

Use Dynamic Colors

Use Dynamic Colors

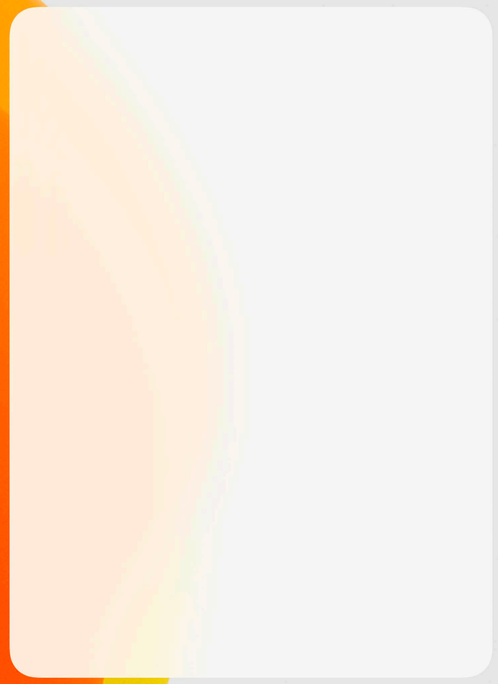
NEW



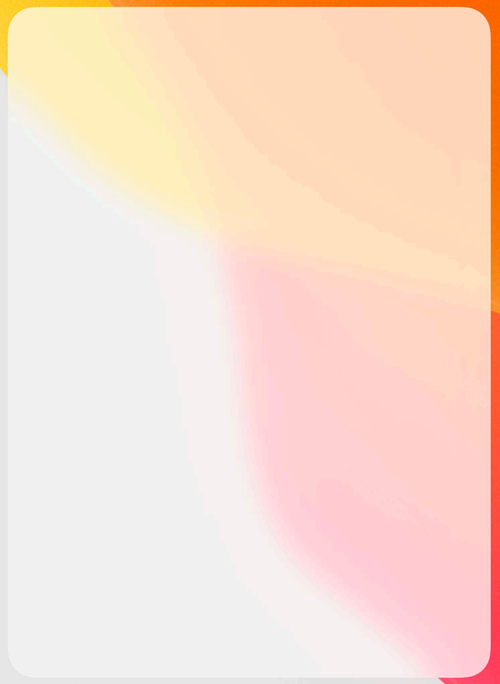
Demo

Materials

Thick



Regular



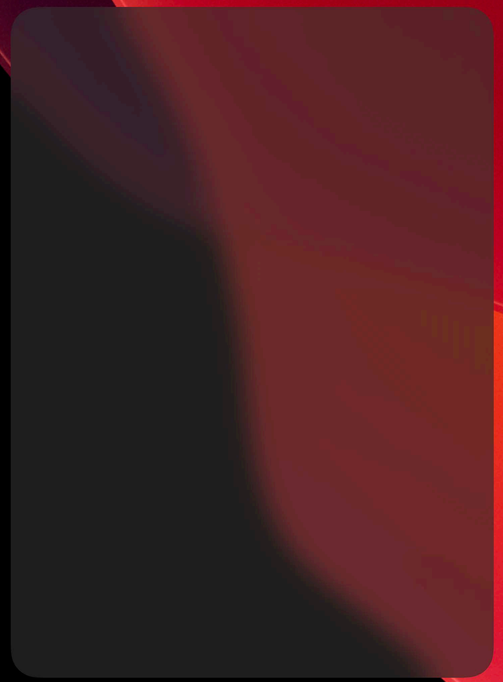
Thin

Ultrathin

Materials

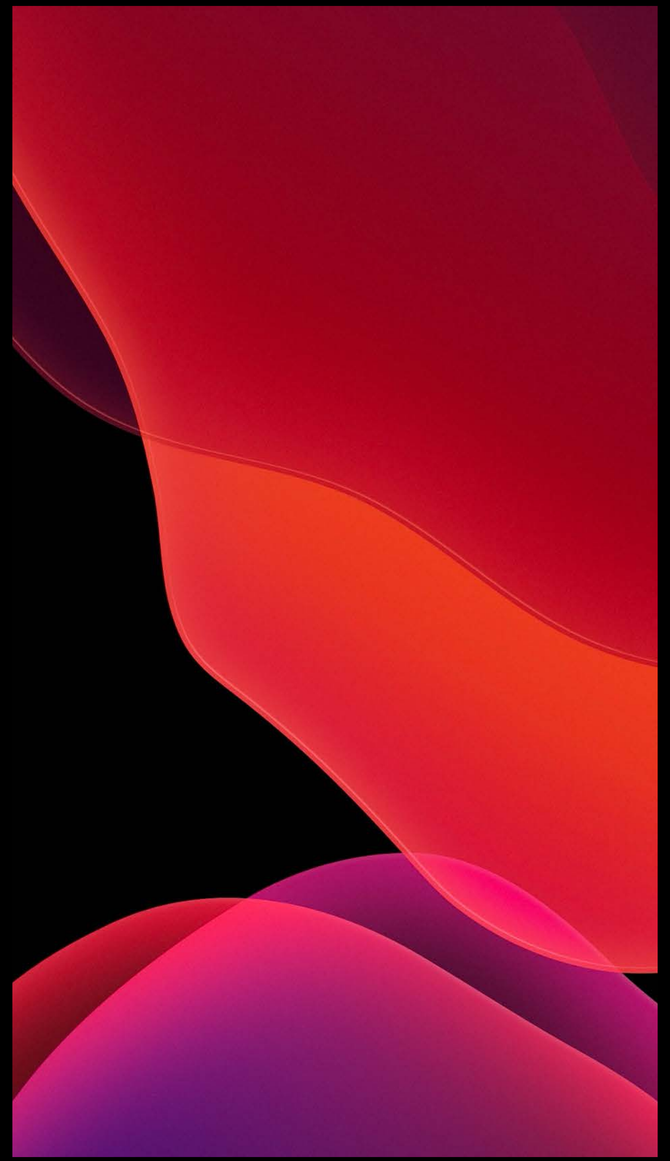
Thick

Regular

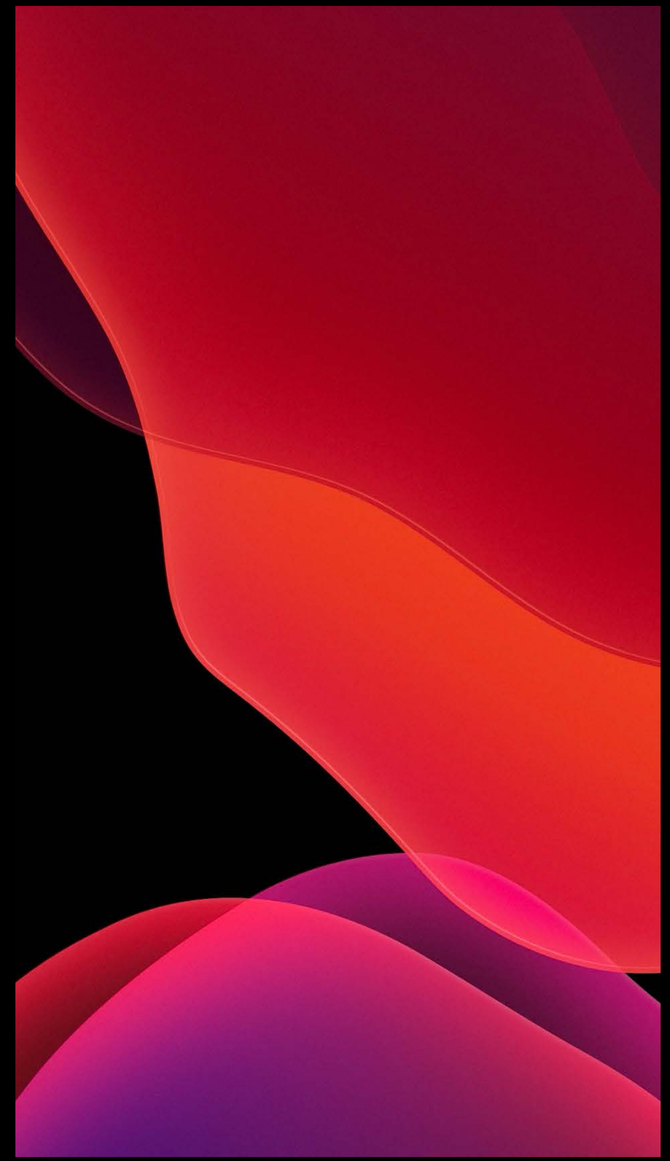


Thin

Ultrathin

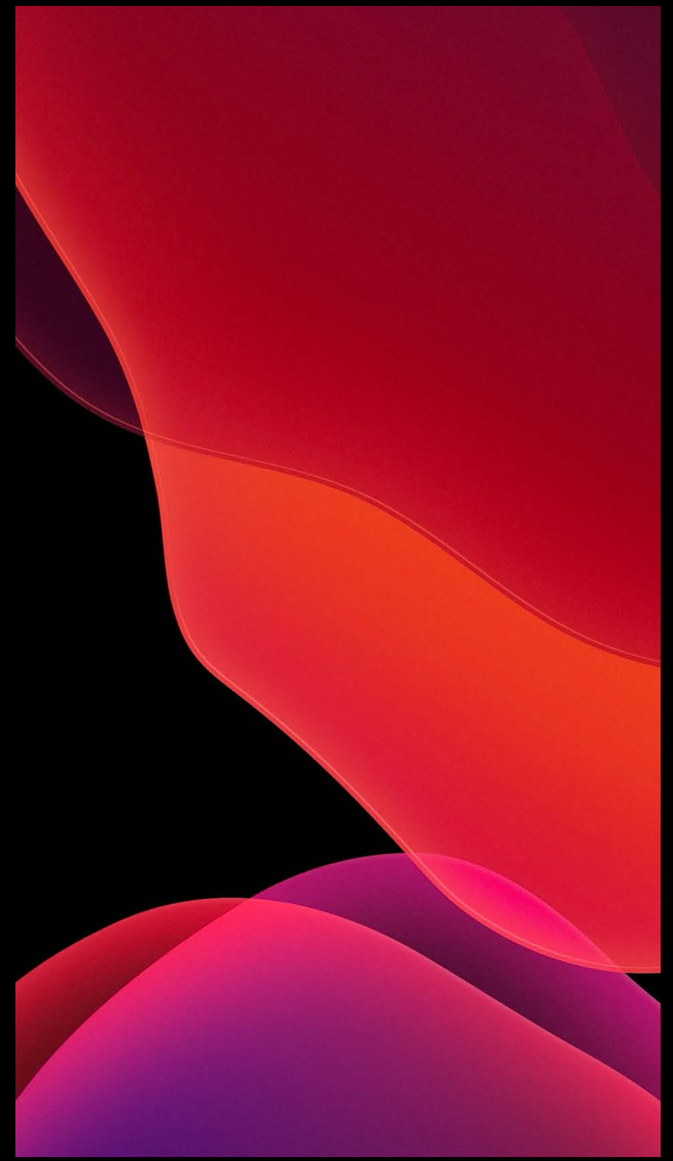


UIBlurEffect
style = .systemMaterial



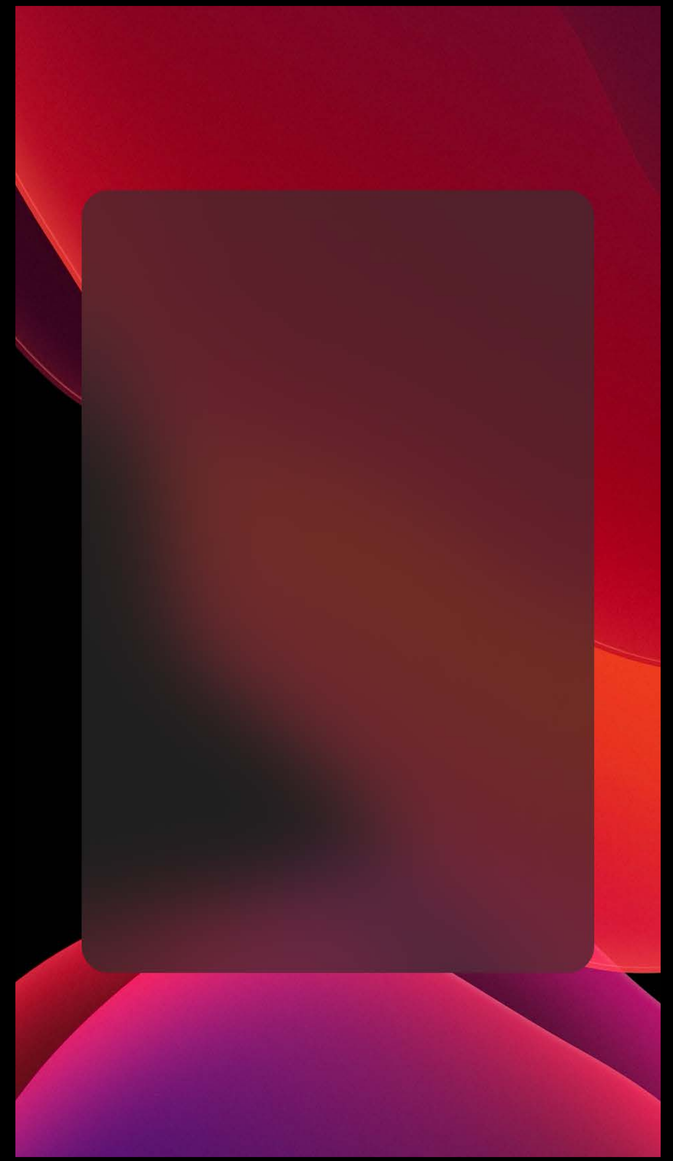
UIBlurEffect
style = .systemMaterial

UIVisualEffectView



UIBlurEffect
style = .systemMaterial

UIVisualEffectView



Labels

Vibrant Primary Text

Vibrant Secondary Text

Vibrant Tertiary Text

Vibrant Quarternary Text

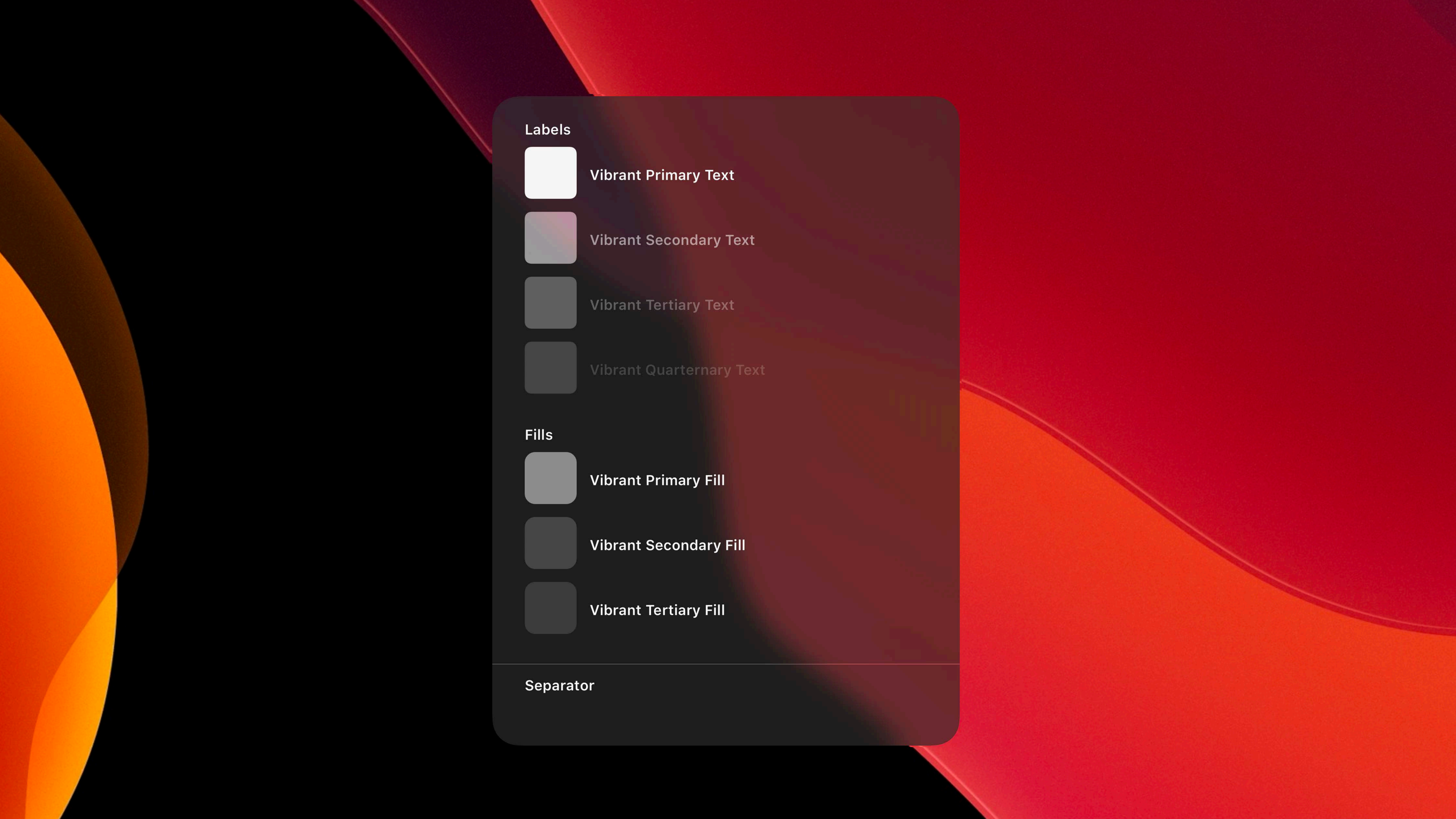
Fills

Vibrant Primary Fill

Vibrant Secondary Fill

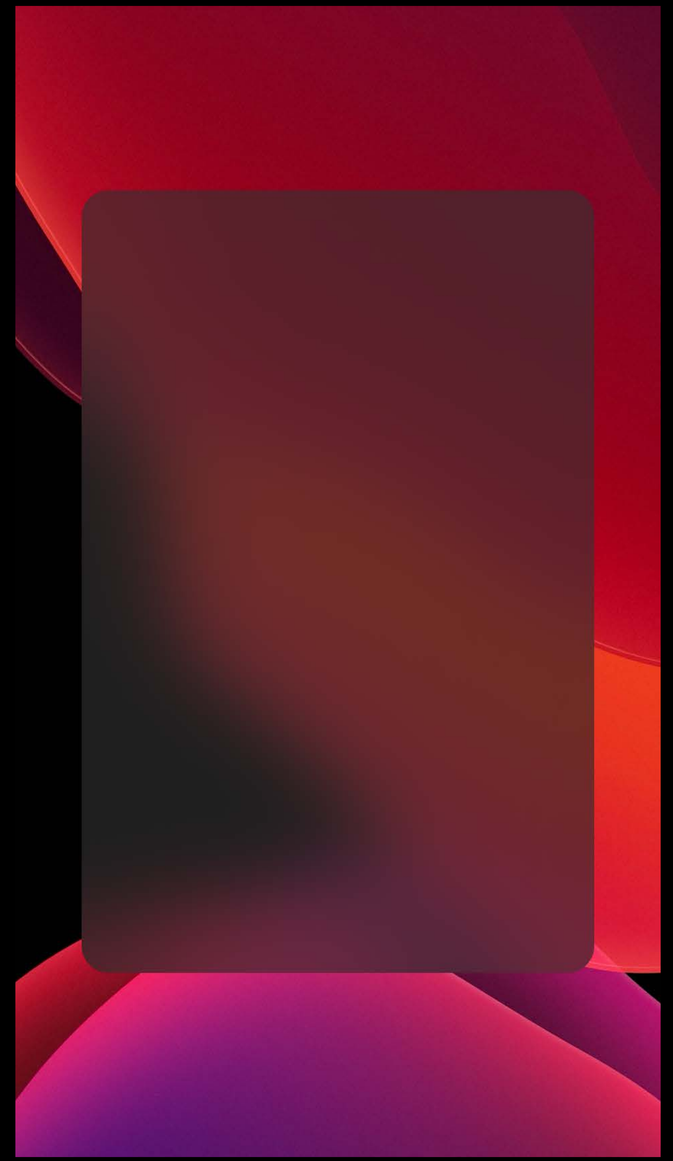
Vibrant Tertiary Fill

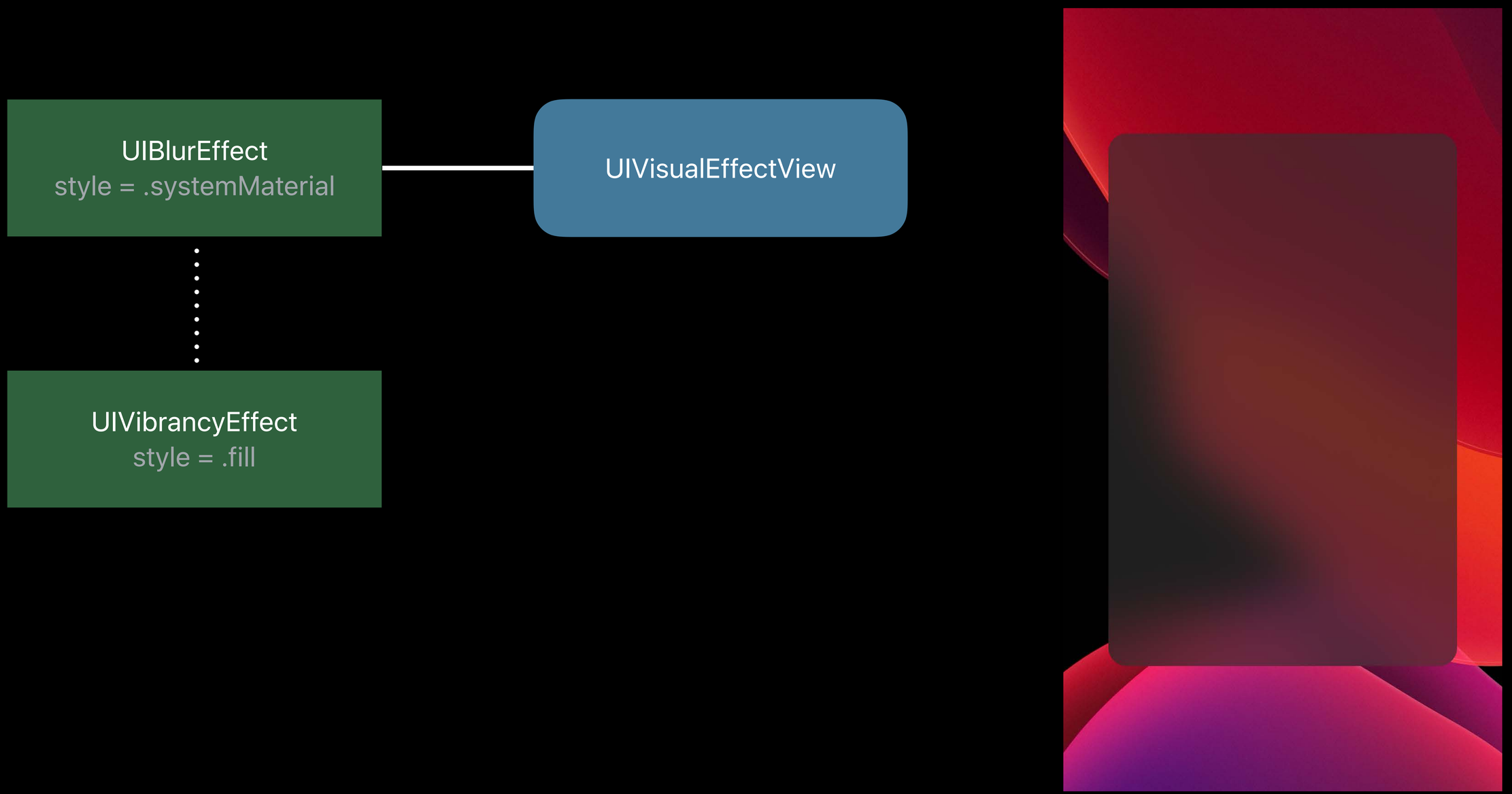
Separator

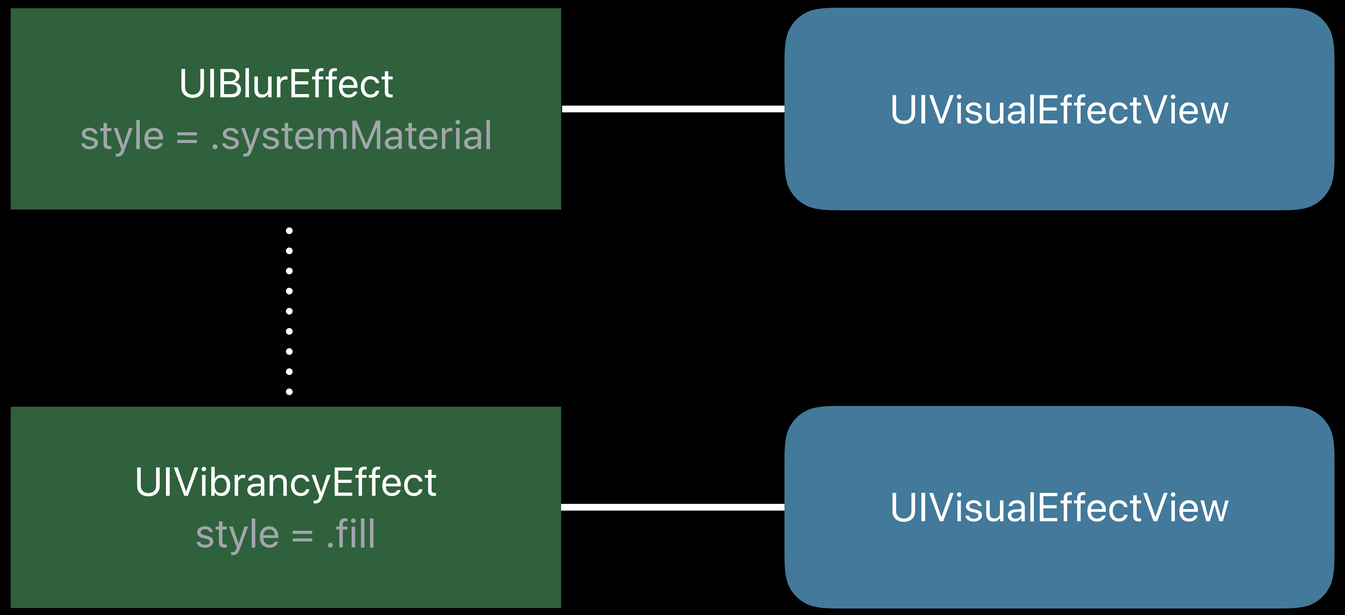

Labels
Vibrant Primary Text
Vibrant Secondary Text
Vibrant Tertiary Text
Vibrant Quarternary Text
Fills
Vibrant Primary Fill
Vibrant Secondary Fill
Vibrant Tertiary Fill
Separator

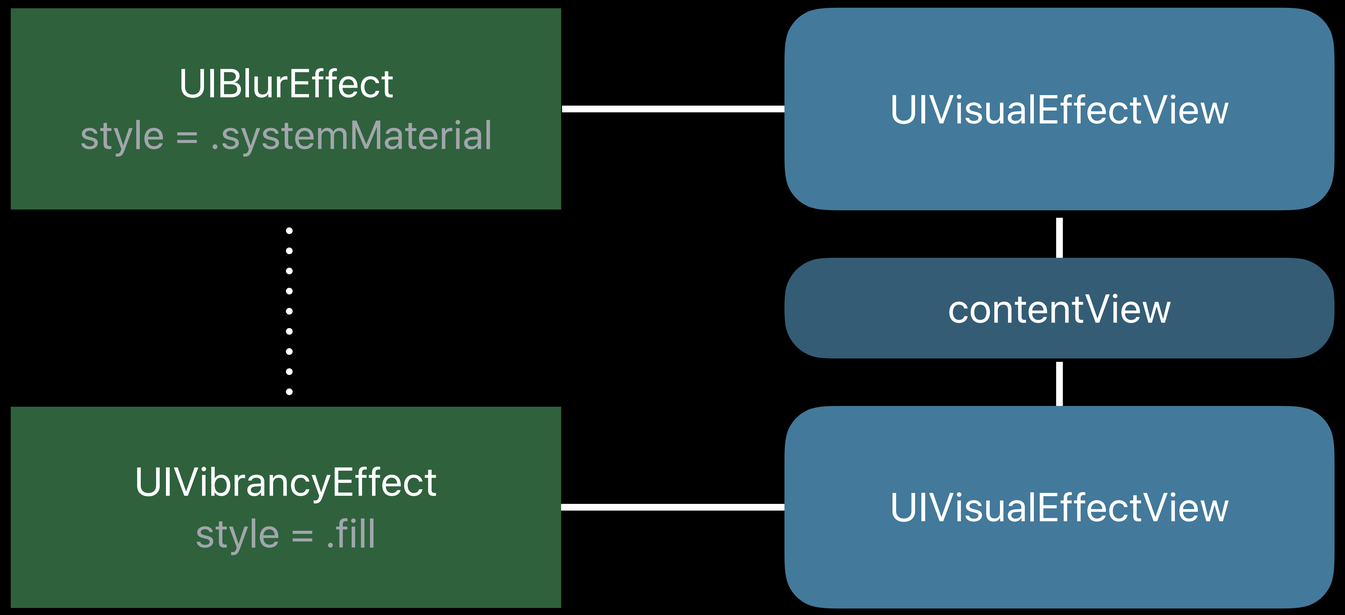
UIBlurEffect
style = .systemMaterial

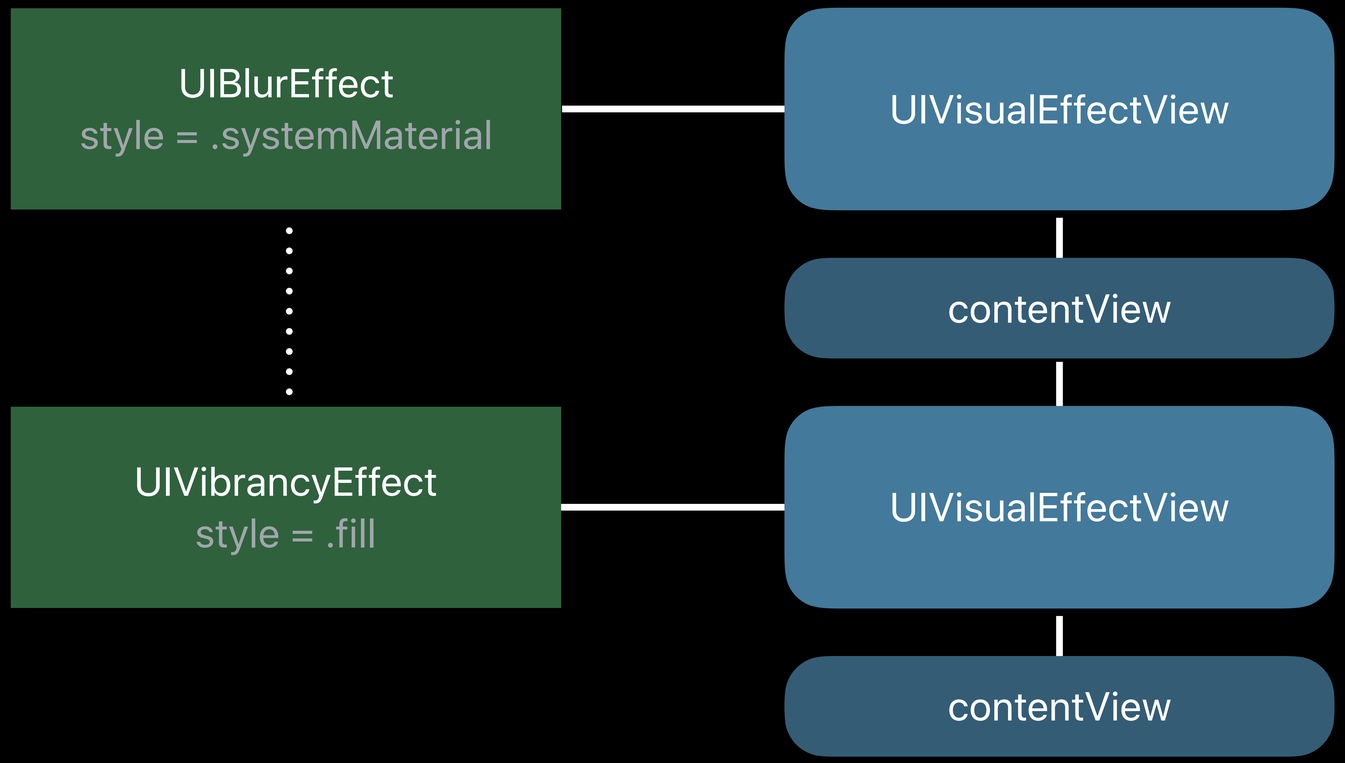
UIVisualEffectView

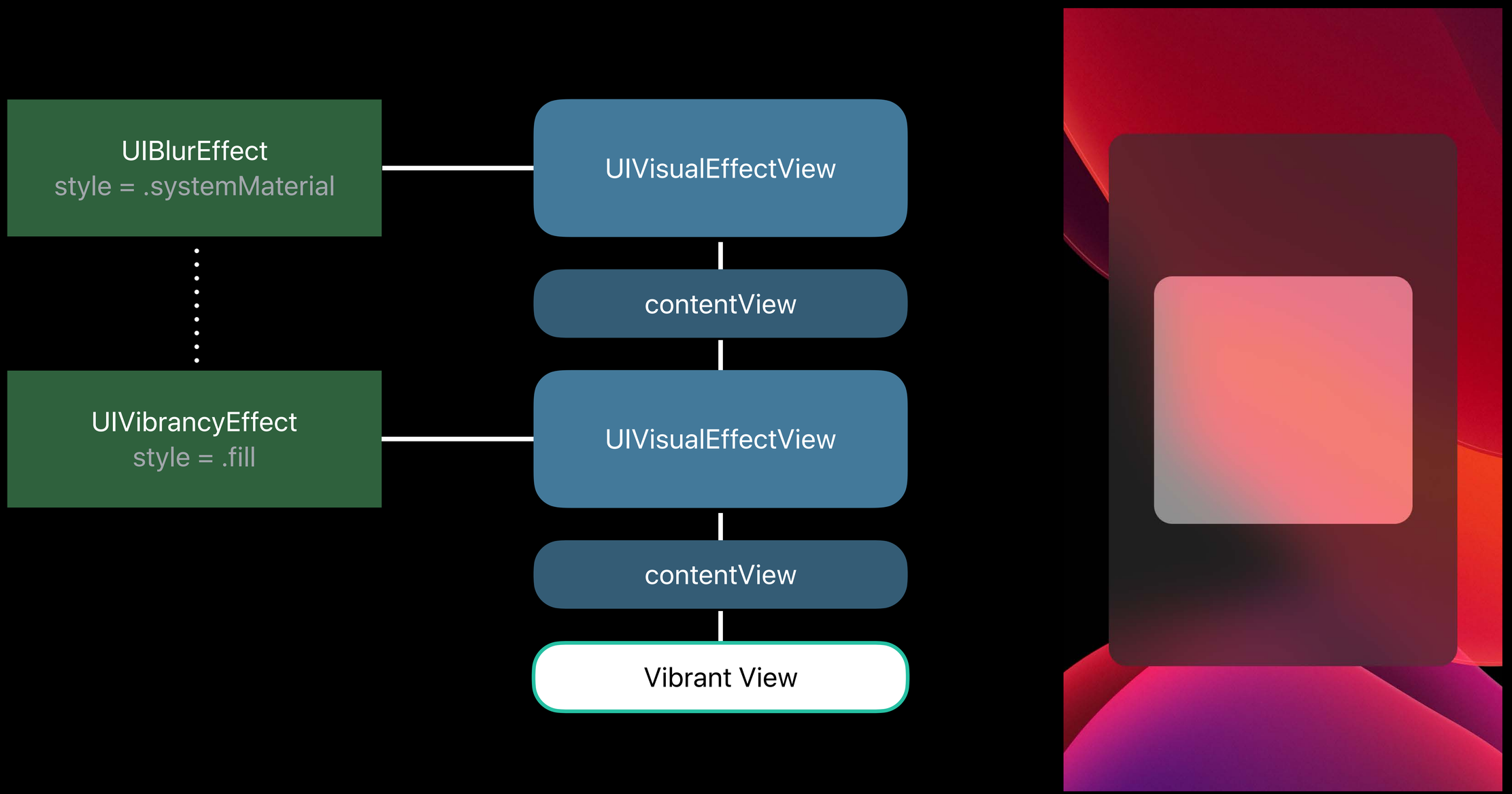



UIBlurEffect
style = .systemMaterial
UIVisualEffectView
UIVibrancyEffect
style = .fill

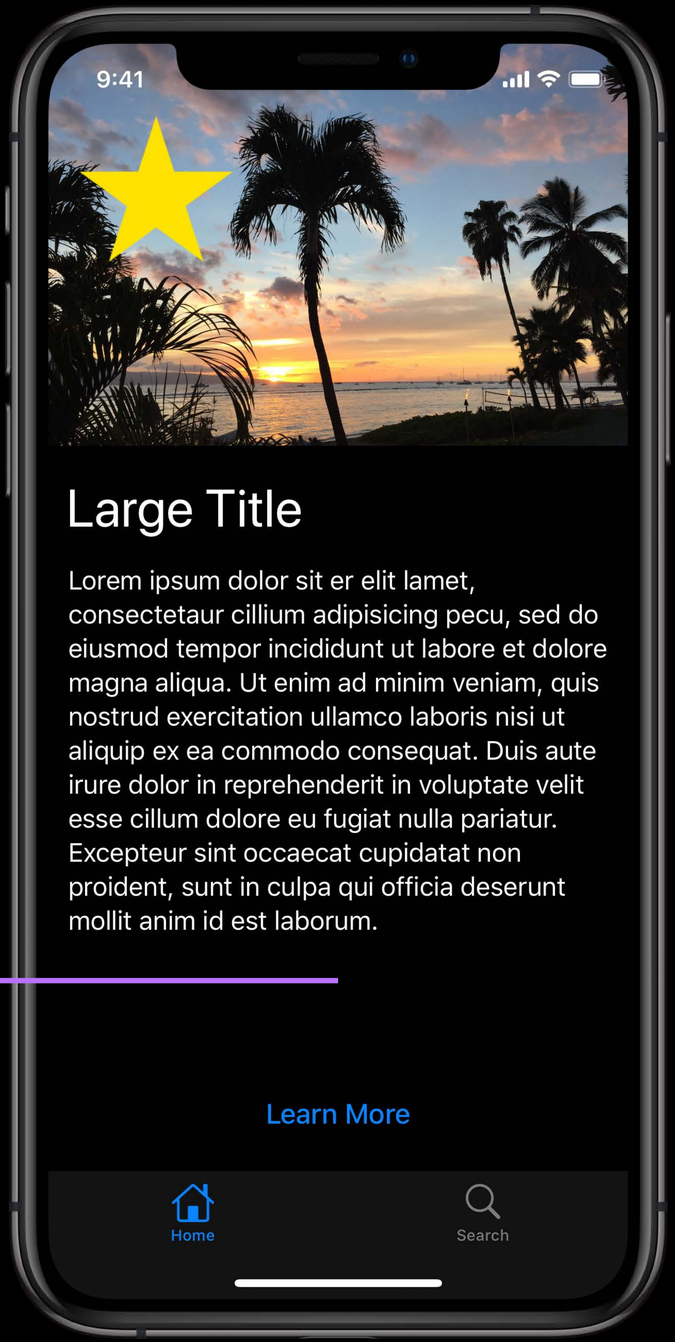

UIBlurEffect
style = .systemMaterial
UIVisualEffectView
UIVibrancyEffect
style = .fill
UIVisualEffectView


UIBlurEffect
style = .systemMaterial
UIVisualEffectView
contentView
UIVibrancyEffect
style = .fill
UIVisualEffectView

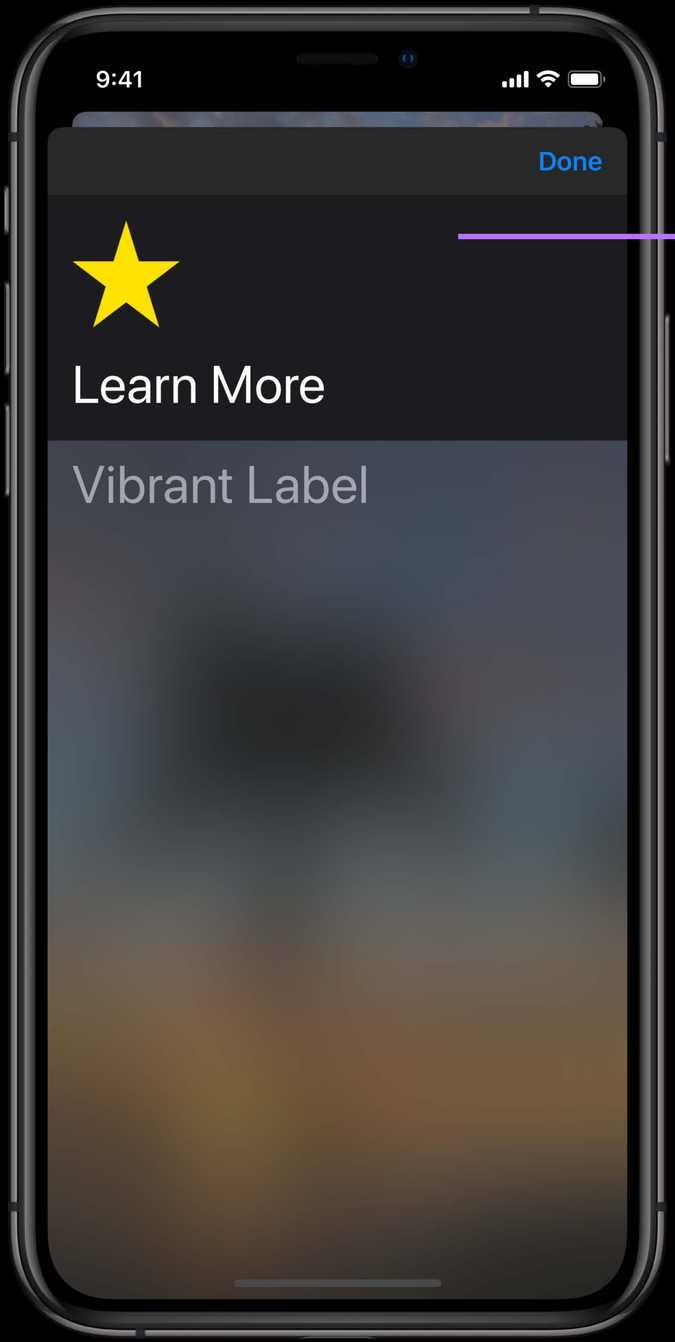

UIBlurEffect
style = .systemMaterial
UIVisualEffectView
contentView
UIVibrancyEffect
style = .fill
UIVisualEffectView
contentView


UIBlurEffect
style = .systemMaterial
UIVisualEffectView
contentView
UIVibrancyEffect
style = .fill
UIVisualEffectView
contentView
Vibrant View

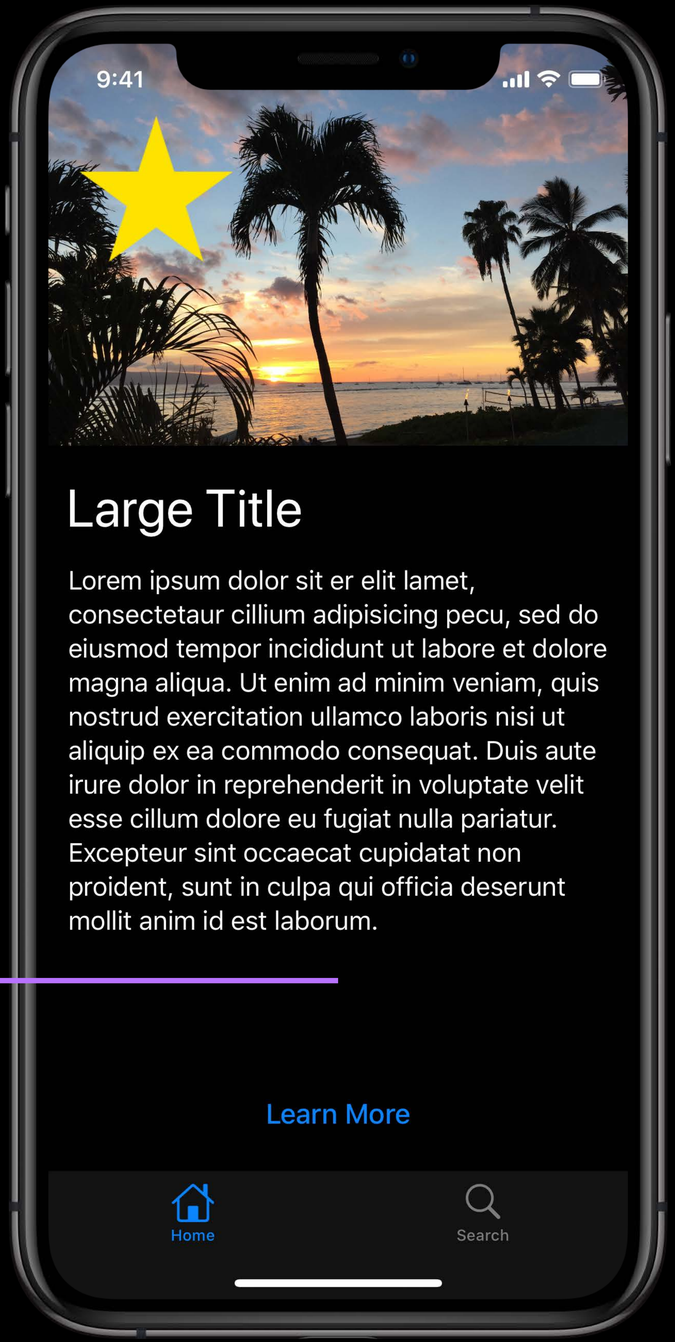
Demo



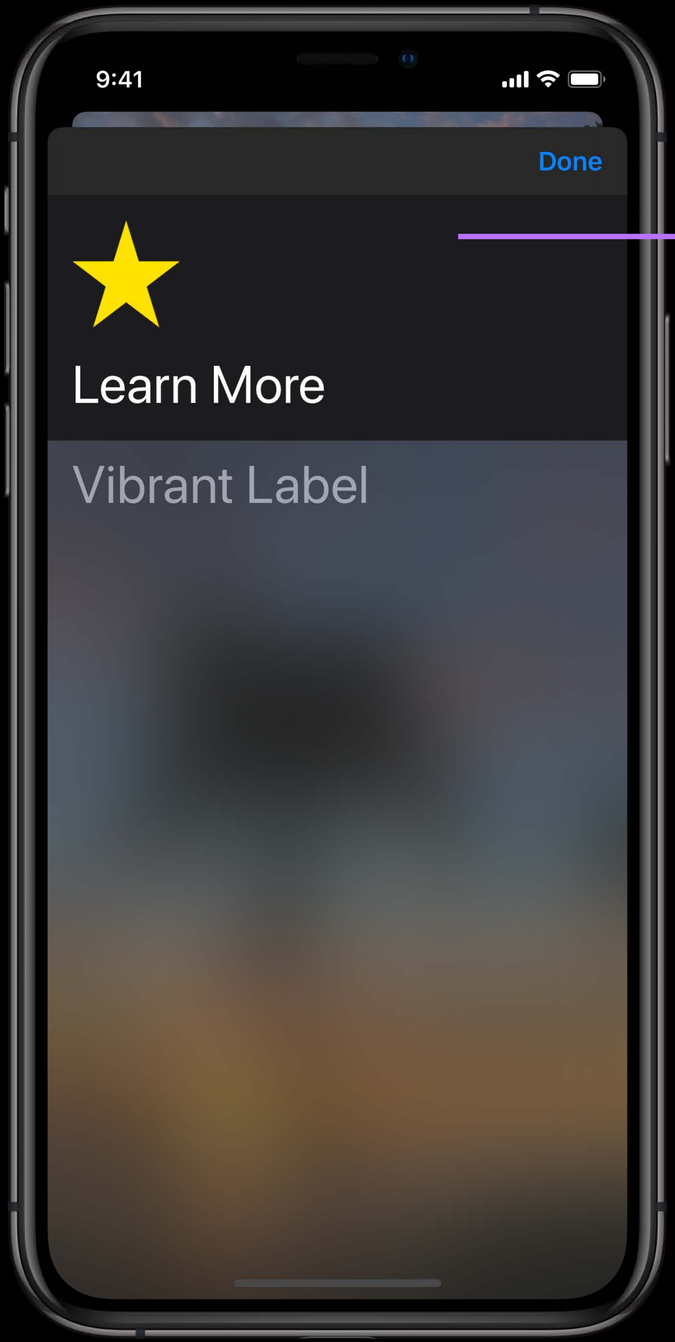
systemBackground



systemBackground



base level



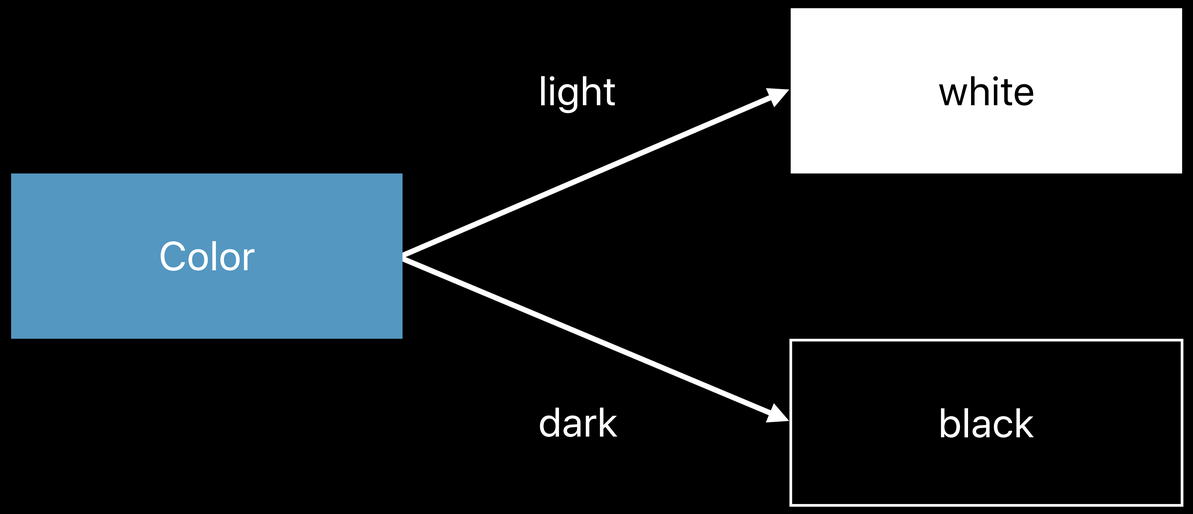
elevated level

Dark Mode is a new look

It’s easy to implement

Flexible and powerful

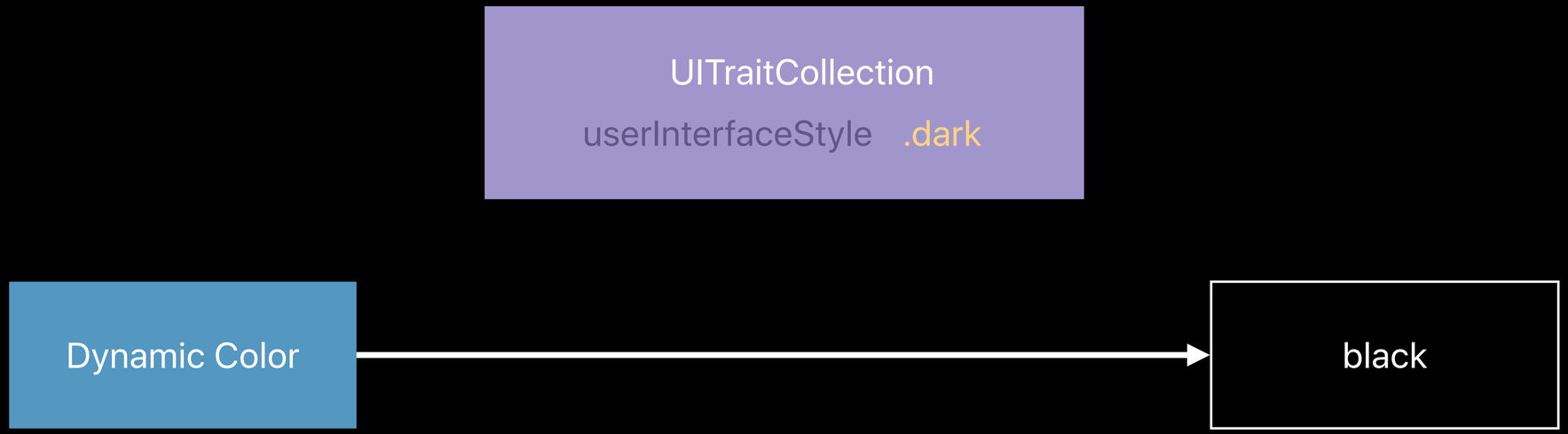
Dynamic Colors



UITraitCollection

userInterfaceIdiom	.phone
userInterfaceStyle	.dark
userInterfaceLevel	.base

Resolving Dynamic Colors



Resolving Dynamic Colors

NEW

```
let dynamicColor = UIColor.systemBackground
let traitCollection = view.traitCollection
let resolvedColor = dynamicColor.resolvedColor(with: traitCollection)
```

Resolving Dynamic Colors

NEW

```
let dynamicColor = UIColor.systemBackground
let traitCollection = view.traitCollection
let resolvedColor = dynamicColor.resolvedColor(with: traitCollection)
```

Resolving Dynamic Colors

NEW

```
let dynamicColor = UIColor.systemBackground
let traitCollection = view.traitCollection
let resolvedColor = dynamicColor.resolvedColor(with: traitCollection)
```

Resolving Dynamic Colors

NEW

```
let dynamicColor = UIColor.systemBackground
let traitCollection = view.traitCollection
let resolvedColor = dynamicColor.resolvedColor(with: traitCollection)
```

Custom Dynamic Colors

NEW

```
let dynamicColor = UIColor { (traitCollection: UITraitCollection) -> UIColor in
    if traitCollection.userInterfaceStyle == .dark {
        return .black
    } else {
        return .white
    }
}
```

Custom Dynamic Colors

NEW

```
let dynamicColor = UIColor { (traitCollection: UITraitCollection) -> UIColor in
    if traitCollection.userInterfaceStyle == .dark {
        return .black
    } else {
        return .white
    }
}
```

Custom Dynamic Colors

NEW

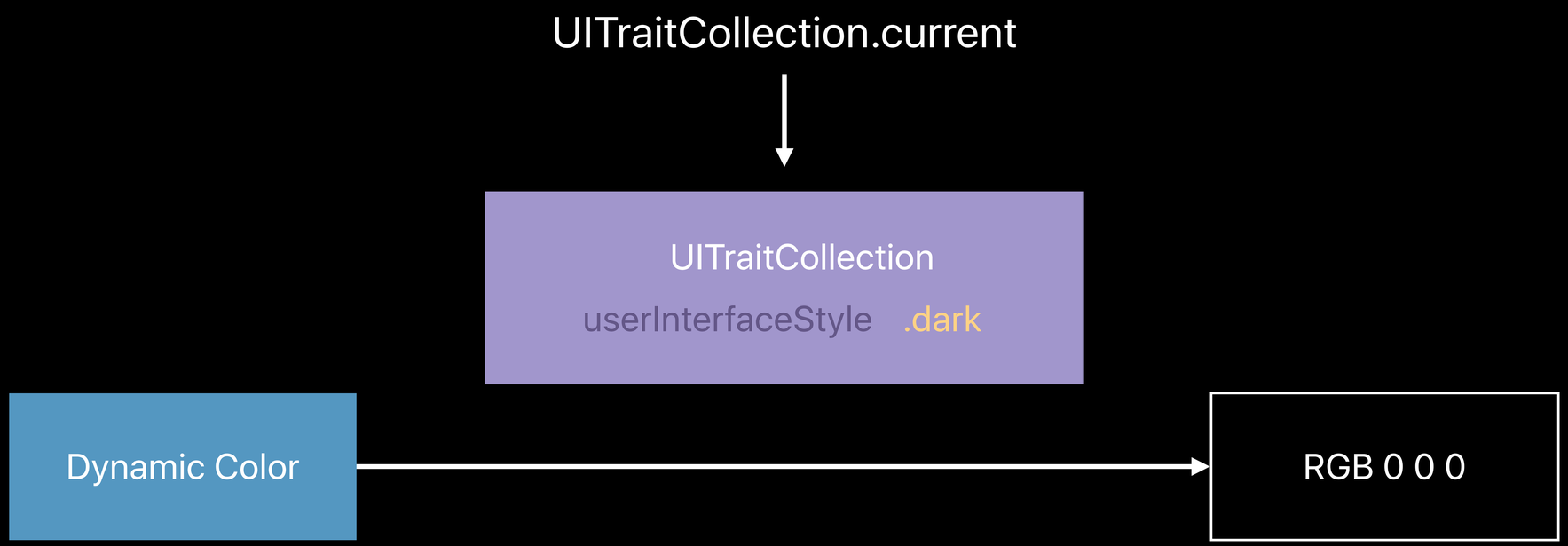
```
let dynamicColor = UIColor { (traitCollection: UITraitCollection) -> UIColor in
    if traitCollection.userInterfaceStyle == .dark {
        return .black
    } else {
        return .white
    }
}
```

Custom Dynamic Colors

NEW

```
let dynamicColor = UIColor { (traitCollection: UITraitCollection) -> UIColor in
    if traitCollection.userInterfaceStyle == .dark {
        return .black
    } else {
        return .white
    }
}
```


Color
RGB 0 0 0


UITraitCollection.current
UITraitCollection
userInterfaceStyle .dark
Dynamic Color
RGB 0 0 0

Current Trait Collection

```
class BackgroundView: UIView {
    override func draw(_ rect: CGRect) {

        UIColor.systemBackground.setFill()
        UIRectFill(rect)
    }
}
```

Current Trait Collection

```
class BackgroundView: UIView {
    override func draw(_ rect: CGRect) {

        UIColor.systemBackground.setFill()
        UIRectFill(rect)
    }
}
```

Current Trait Collection

```
class BackgroundView: UIView {
    override func draw(_ rect: CGRect) {
        // UIKit sets UITraitCollection.current to self.traitCollection
        UIColor.systemBackground.setFill()
        UIRectFill(rect)
    }
}
```

Current Trait Collection

```
class BackgroundView: UIView {
    override func draw(_ rect: CGRect) {
        // UIKit sets UITraitCollection.current to self.traitCollection
        UIColor.systemBackground.setFill()
        UIRectFill(rect)
    }
}
```

Current Trait Collection

```
class BackgroundView: UIView {
    override func draw(_ rect: CGRect) {
        // UIKit sets UITraitCollection.current to self.traitCollection
        UIColor.systemBackground.setFill()
        UIRectFill(rect)
    }
}
```

Current Trait Collection

```
class BackgroundView: UIView {
    override func draw(_ rect: CGRect) {
        // UIKit sets UITraitCollection.current to self.traitCollection
        UIColor.systemBackground.setFill()
        UIRectFill(rect)
    }
}
```

Current Trait Collection

UIView

UIViewController

UIPresentationController

```
draw()
```

Current Trait Collection

UIView	UIViewController	UIPresentationController
`draw()`		
`layoutSubviews()`	`viewWillLayoutSubviews()` `viewDidLayoutSubviews()`	`containerViewWillLayoutSubviews()` `containerViewDidLayoutSubviews()`

Current Trait Collection

UIView	UIViewController	UIPresentationController
`draw()`		
`layoutSubviews()`	`viewWillLayoutSubviews()` `viewDidLayoutSubviews()`	`containerViewWillLayoutSubviews()` `containerViewDidLayoutSubviews()`
`traitCollectionDidChange()` `tintColorDidChange()`	`traitCollectionDidChange()`	`traitCollectionDidChange()`

```
let layer = CALayer()



layer.borderColor = UIColor.label.cgColor
```

```
let layer = CALayer()
let traitCollection = view.traitCollection



layer.borderColor = UIColor.label.cgColor
```

```
let layer = CALayer()
let traitCollection = view.traitCollection



layer.borderColor = UIColor.label.cgColor
```

```swift
let layer = CALayer()
let traitCollection = view.traitCollection

// Option 1

let resolvedColor = UIColor.label.resolvedColor(with: traitCollection)
layer.borderColor = resolvedColor.cgColor
```

```
let layer = CALayer()
let traitCollection = view.traitCollection

// Option 1

let resolvedColor = UIColor.label.resolvedColor(with: traitCollection)
layer.borderColor = resolvedColor.cgColor
```

```
let layer = CALayer()
let traitCollection = view.traitCollection

// Option 2

traitCollection.performAsCurrent {
   layer.borderColor = UIColor.label.cgColor
}
```

```
let layer = CALayer()
let traitCollection = view.traitCollection

// Option 2

traitCollection.performAsCurrent {
    layer.borderColor = UIColor.label.cgColor
}
```

```
let layer = CALayer()
let traitCollection = view.traitCollection

// Option 3



UITraitCollection.current = traitCollection
layer.borderColor = UIColor.label.cgColor
```

```
let layer = CALayer()
let traitCollection = view.traitCollection

// Option 3


UITraitCollection.current = traitCollection
layer.borderColor = UIColor.label.cgColor
```

```
let layer = CALayer()
let traitCollection = view.traitCollection

// Option 3

let savedTraitCollection = UITraitCollection.current

UITraitCollection.current = traitCollection
layer.borderColor = UIColor.label.cgColor

UITraitCollection.current = savedTraitCollection
```

```
let layer = CALayer()
let traitCollection = view.traitCollection

// Option 3

let savedTraitCollection = UITraitCollection.current

UITraitCollection.current = traitCollection
layer.borderColor = UIColor.label.cgColor

UITraitCollection.current = savedTraitCollection
```

When Dynamic Color Might Change

```
override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
    super.traitCollectionDidChange(previousTraitCollection)




}
```

When Dynamic Color Might Change

```
override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
    super.traitCollectionDidChange(previousTraitCollection)




}
```

When Dynamic Color Might Change

NEW

```
override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
    super.traitCollectionDidChange(previousTraitCollection)

    if traitCollection.hasDifferentColorAppearance(comparedTo: previousTraitCollection) {
        // Resolve dynamic colors again
    }
}
```

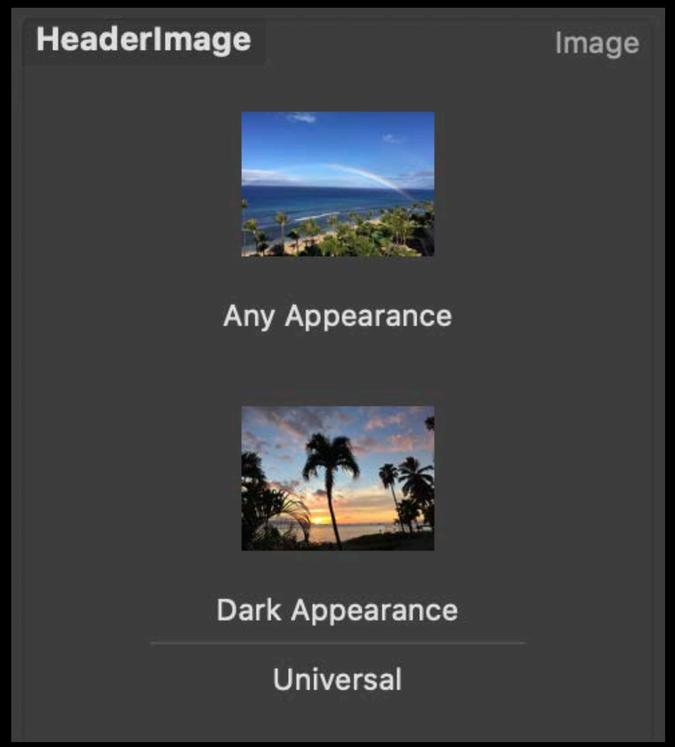
When Dynamic Color Might Change

NEW

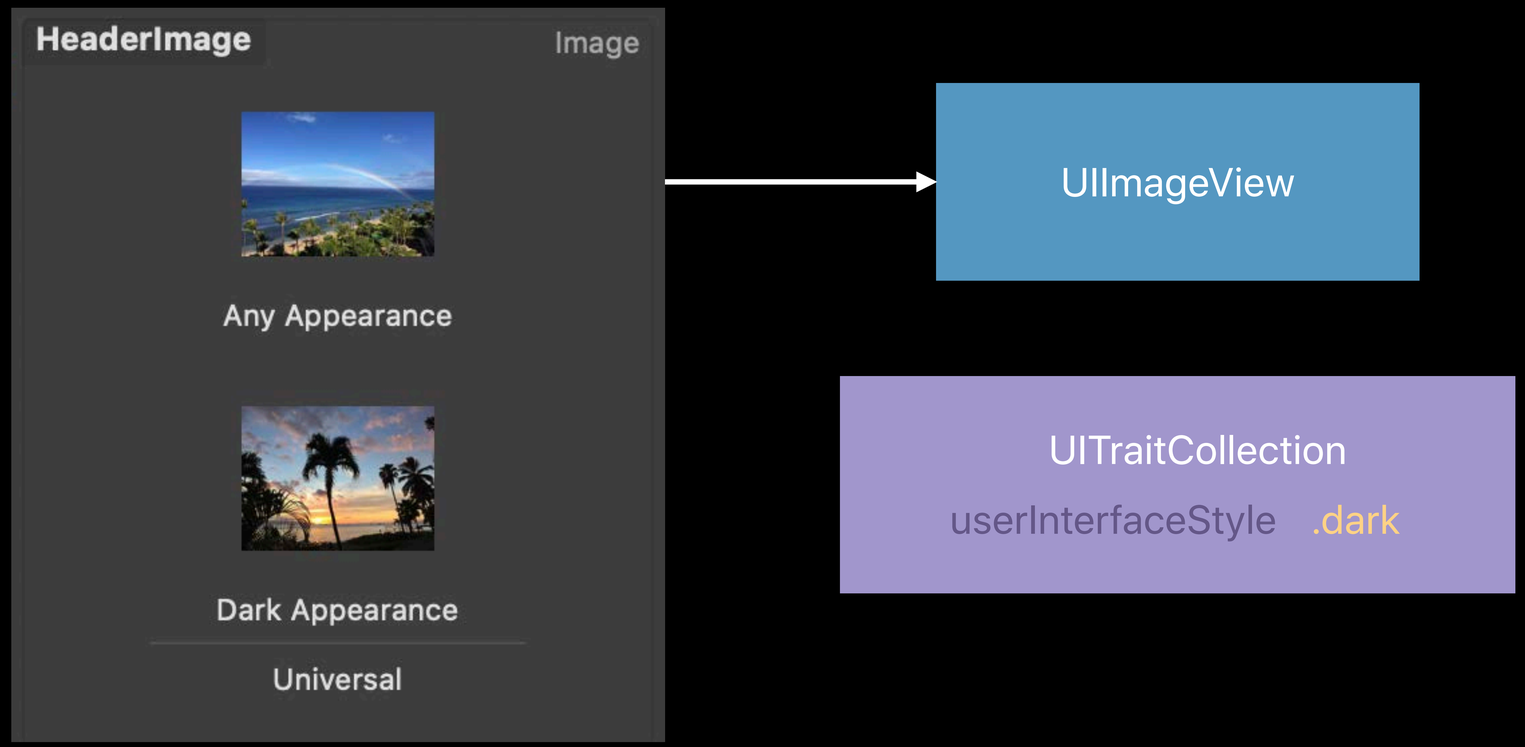
```
override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
    super.traitCollectionDidChange(previousTraitCollection)

    if traitCollection.hasDifferentColorAppearance(comparedTo: previousTraitCollection) {
        // Resolve dynamic colors again
    }
}
```

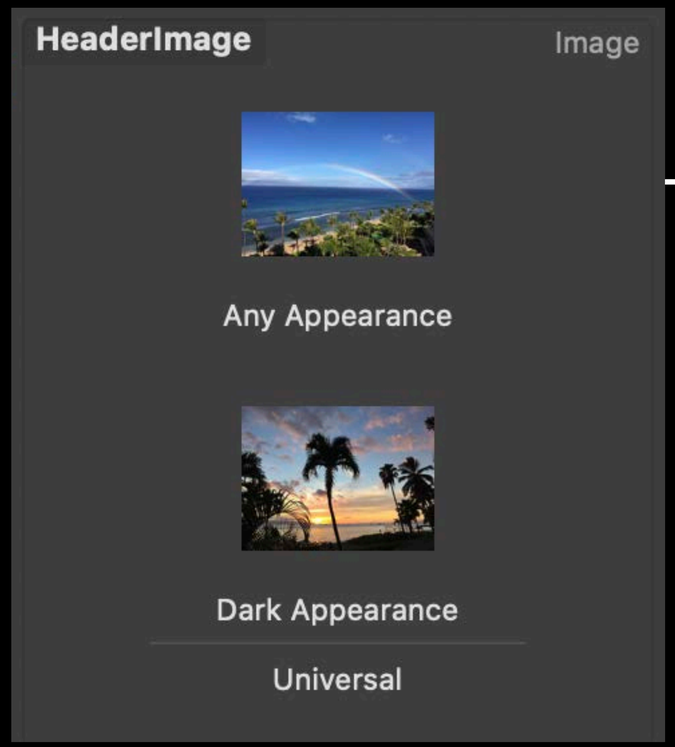
Dynamic Images



Dynamic Images



Dynamic Images



UIImageView

UITraitCollection

userInterfaceStyle .dark

Resolving Dynamic Images

```
let image = UIImage(named: "HeaderImage")
let asset = image?.imageAsset
let resolvedImage = asset?.image(with: traitCollection)
```

Resolving Dynamic Images

```
let image = UIImage(named: "HeaderImage")
let asset = image?.imageAsset
let resolvedImage = asset?.image(with: traitCollection)
```

Resolving Dynamic Images

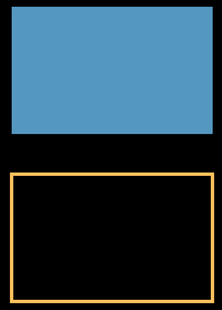
```
let image = UIImage(named: "HeaderImage")
let asset = image?.imageAsset
let resolvedImage = asset?.image(with: traitCollection)
```

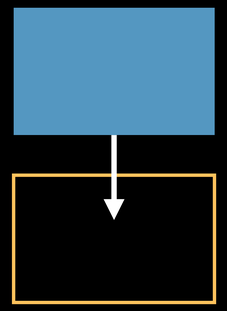
Resolving Dynamic Images

```
let image = UIImage(named: "HeaderImage")
let asset = image?.imageAsset
let resolvedImage = asset?.image(with: traitCollection)
```

Trait Collections

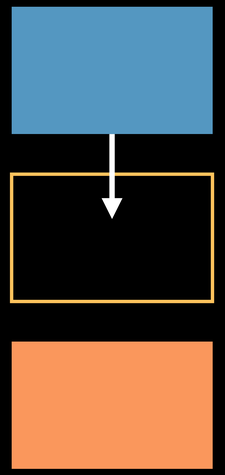
UIScreen


UIScreen
UIWindowScene



UIScreen

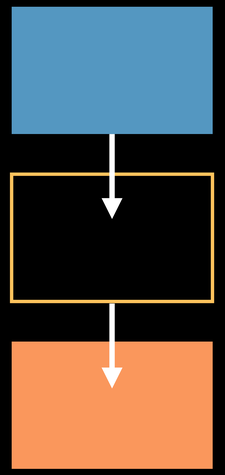
UIWindowScene



UIScreen

UIWindowScene

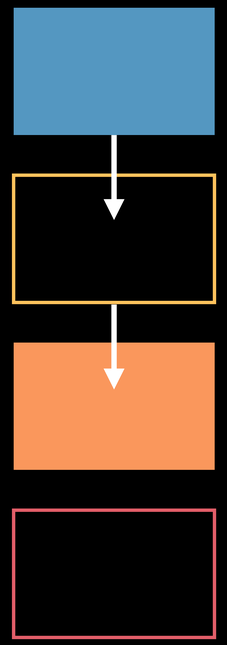
UIWindow



UIScreen

UIWindowScene

UIWindow

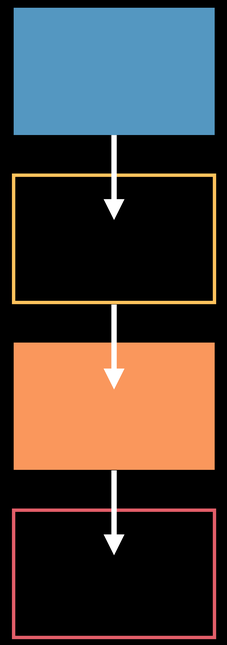


UIScreen

UIWindowScene

UIWindow

UIPresentationController

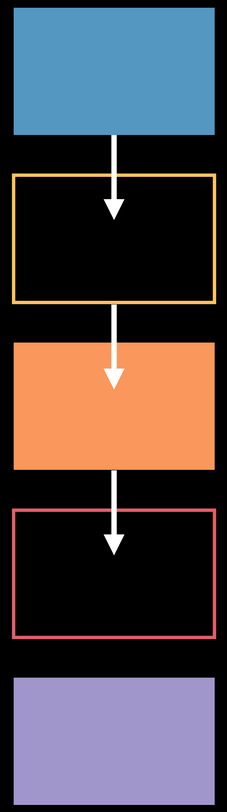


UIScreen

UIWindowScene

UIWindow

UIPresentationController



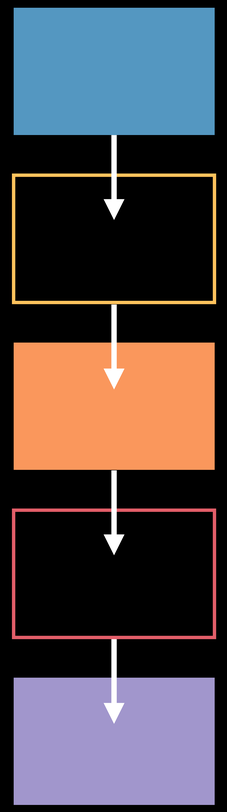
UIScreen

UIWindowScene

UIWindow

UIPresentationController

UIViewController



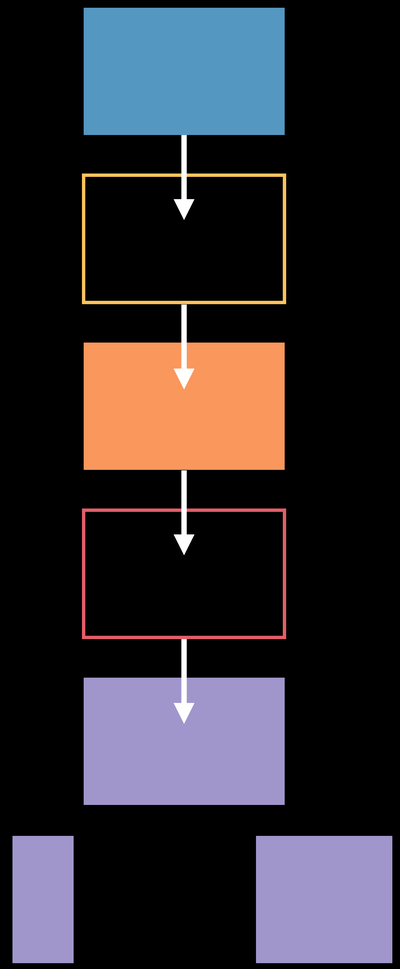
UIScreen

UIWindowScene

UIWindow

UIPresentationController

UIViewController



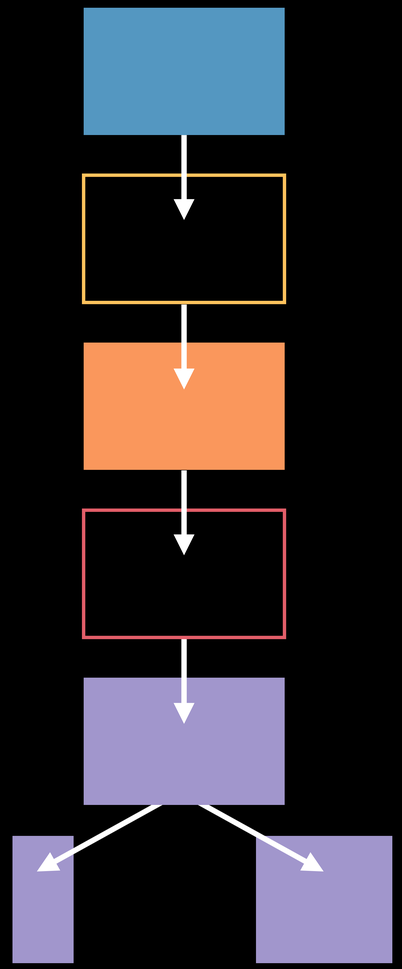
UIScreen

UIWindowScene

UIWindow

UIPresentationController

UIViewController



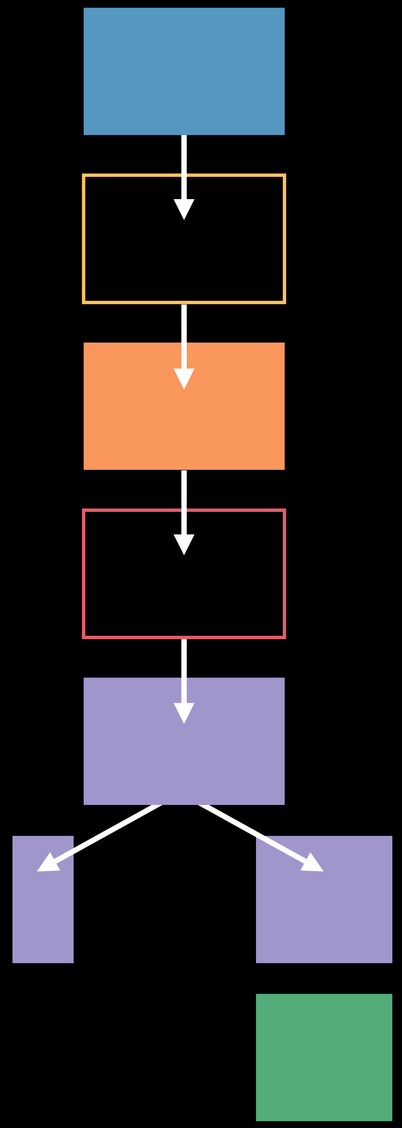
UIScreen

UIWindowScene

UIWindow

UIPresentationController

UIViewController



UIScreen

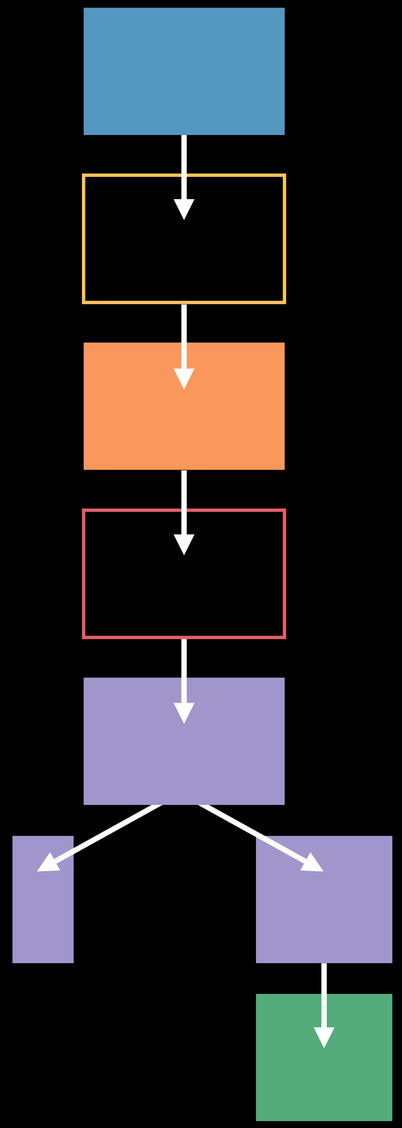
UIWindowScene

UIWindow

UIPresentationController

UIViewController

UIView



UIScreen

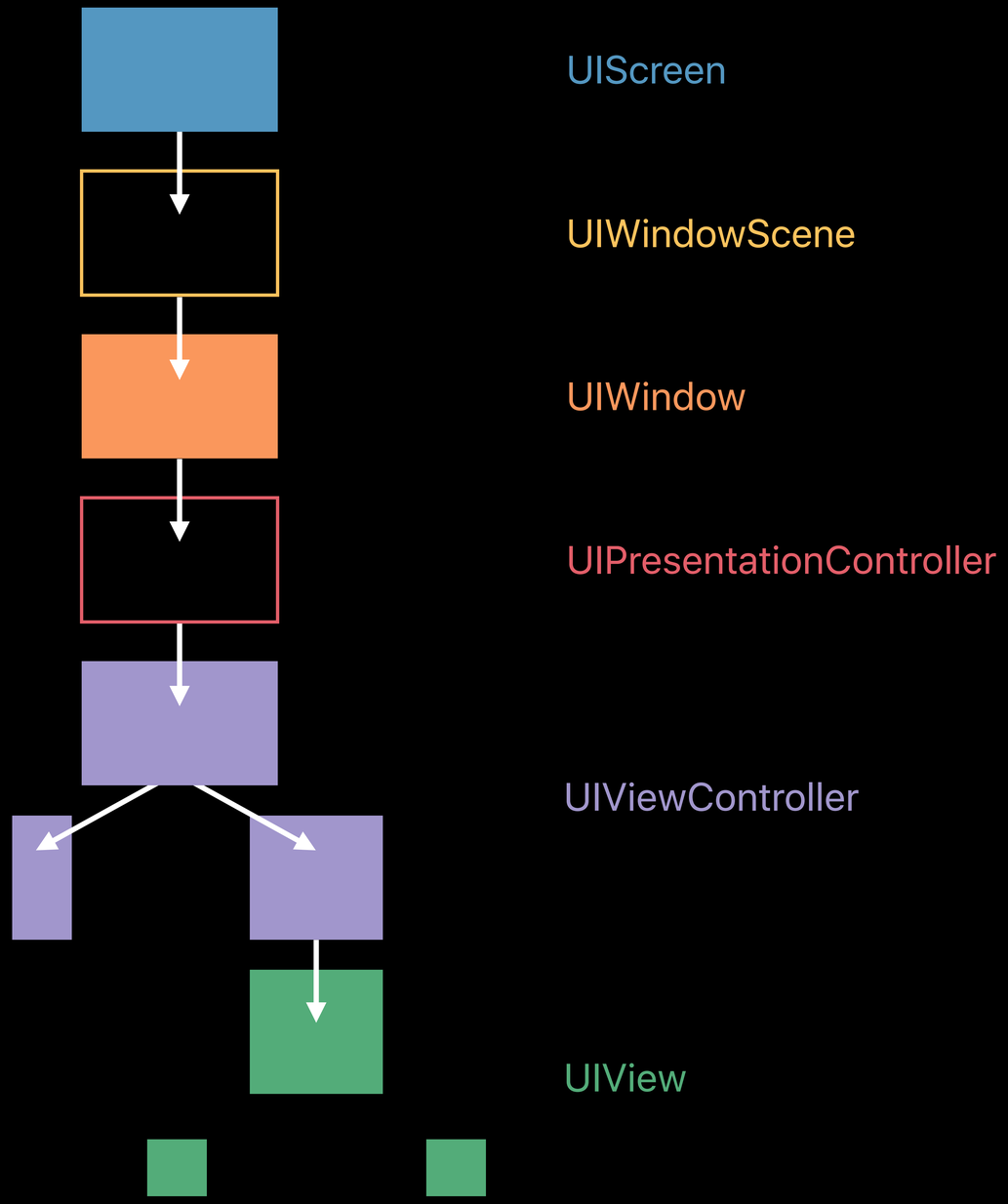
UIWindowScene

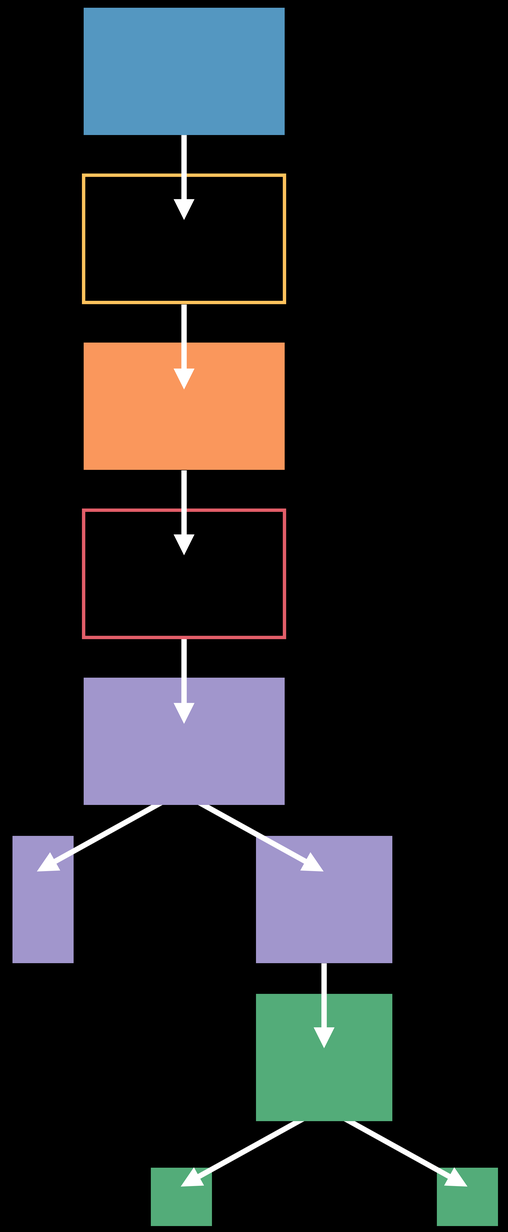
UIWindow

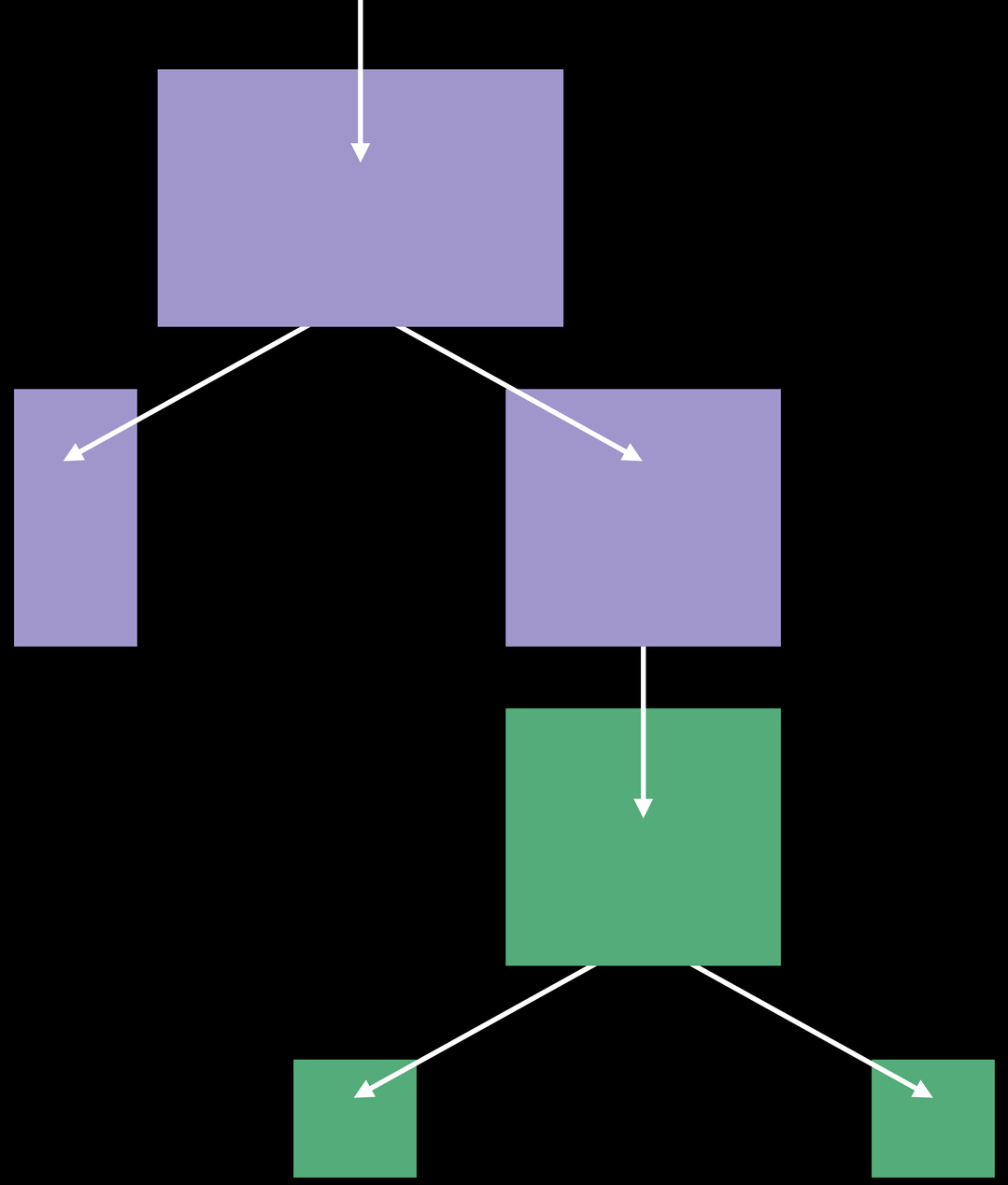
UIPresentationController

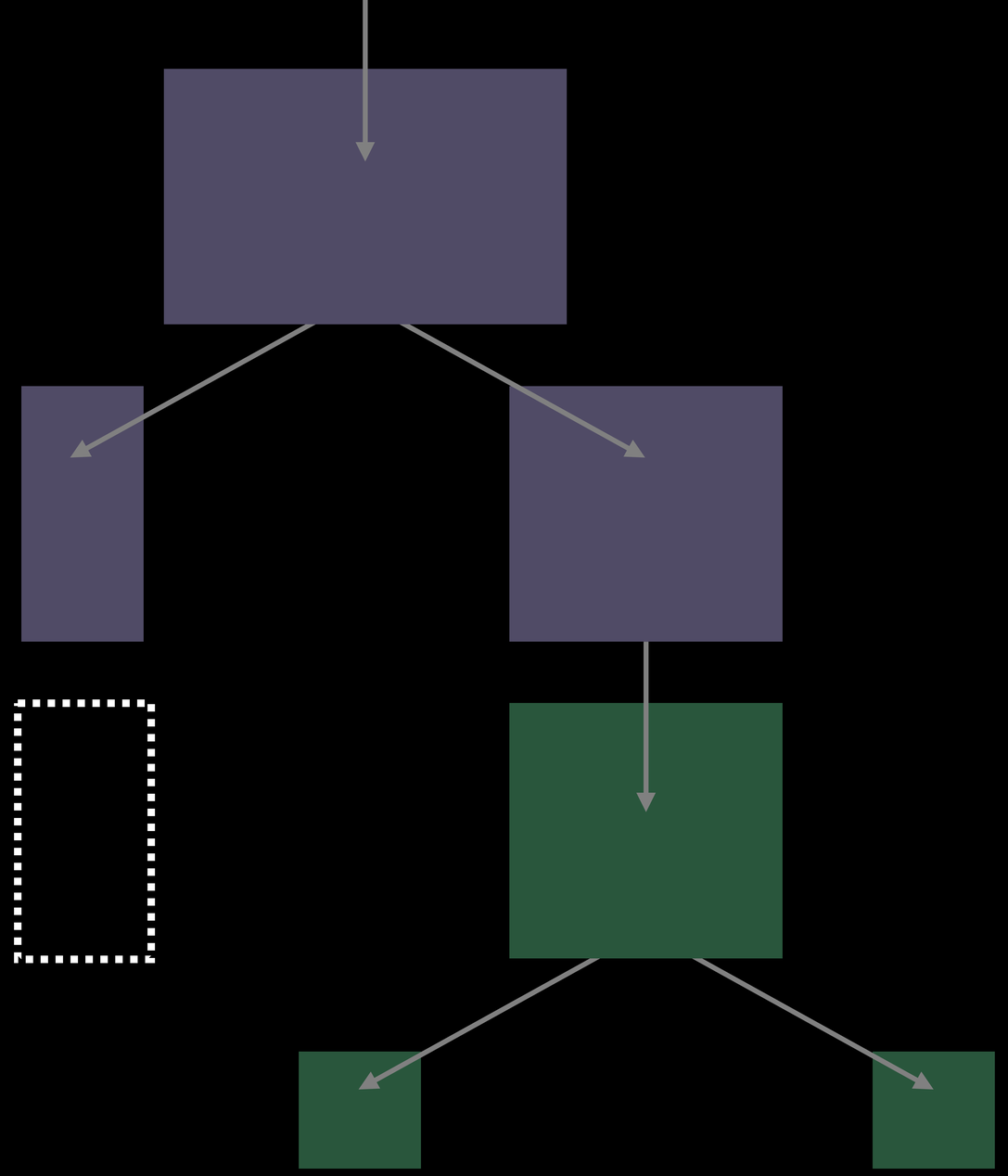
UIViewController

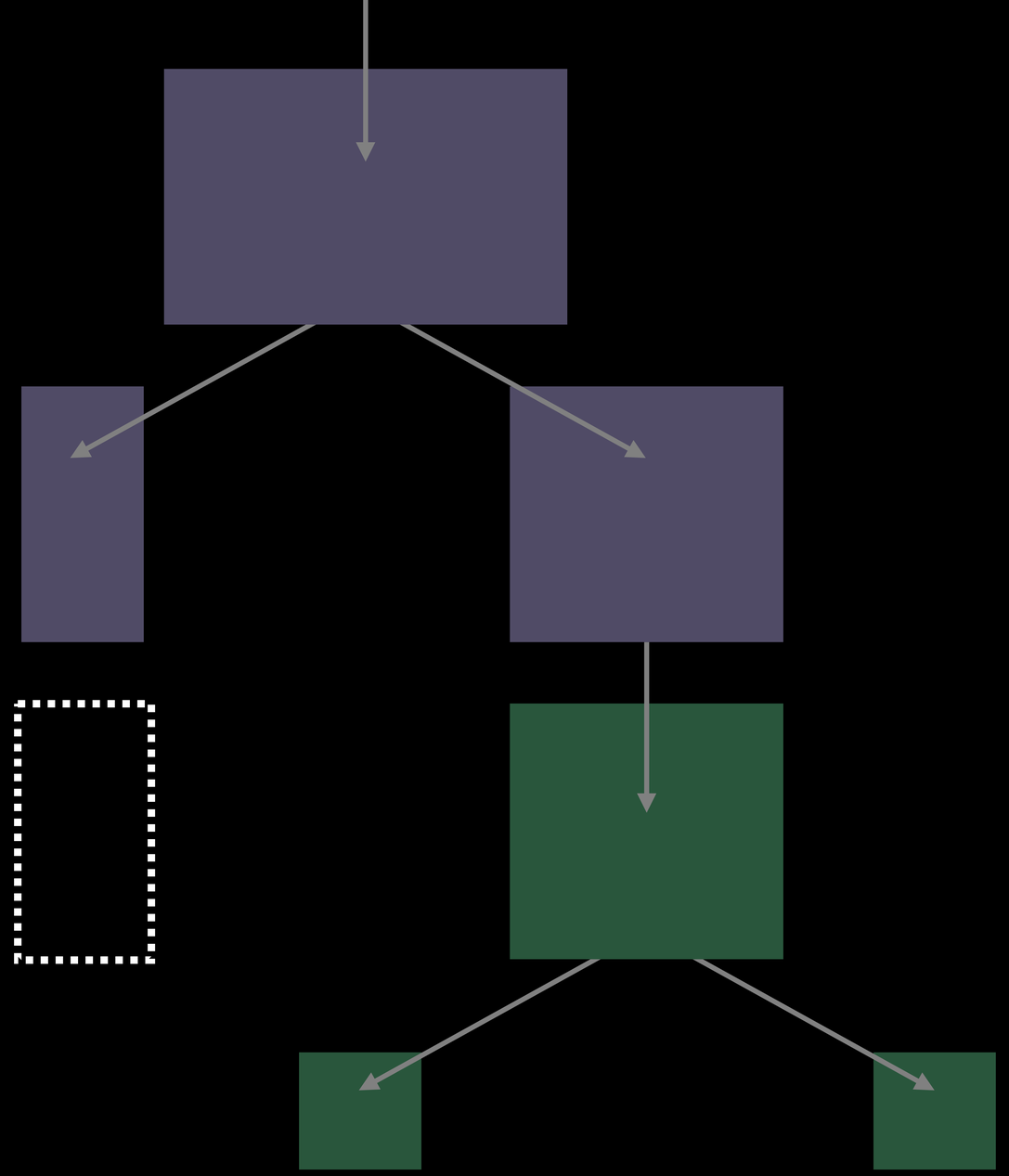
UIView

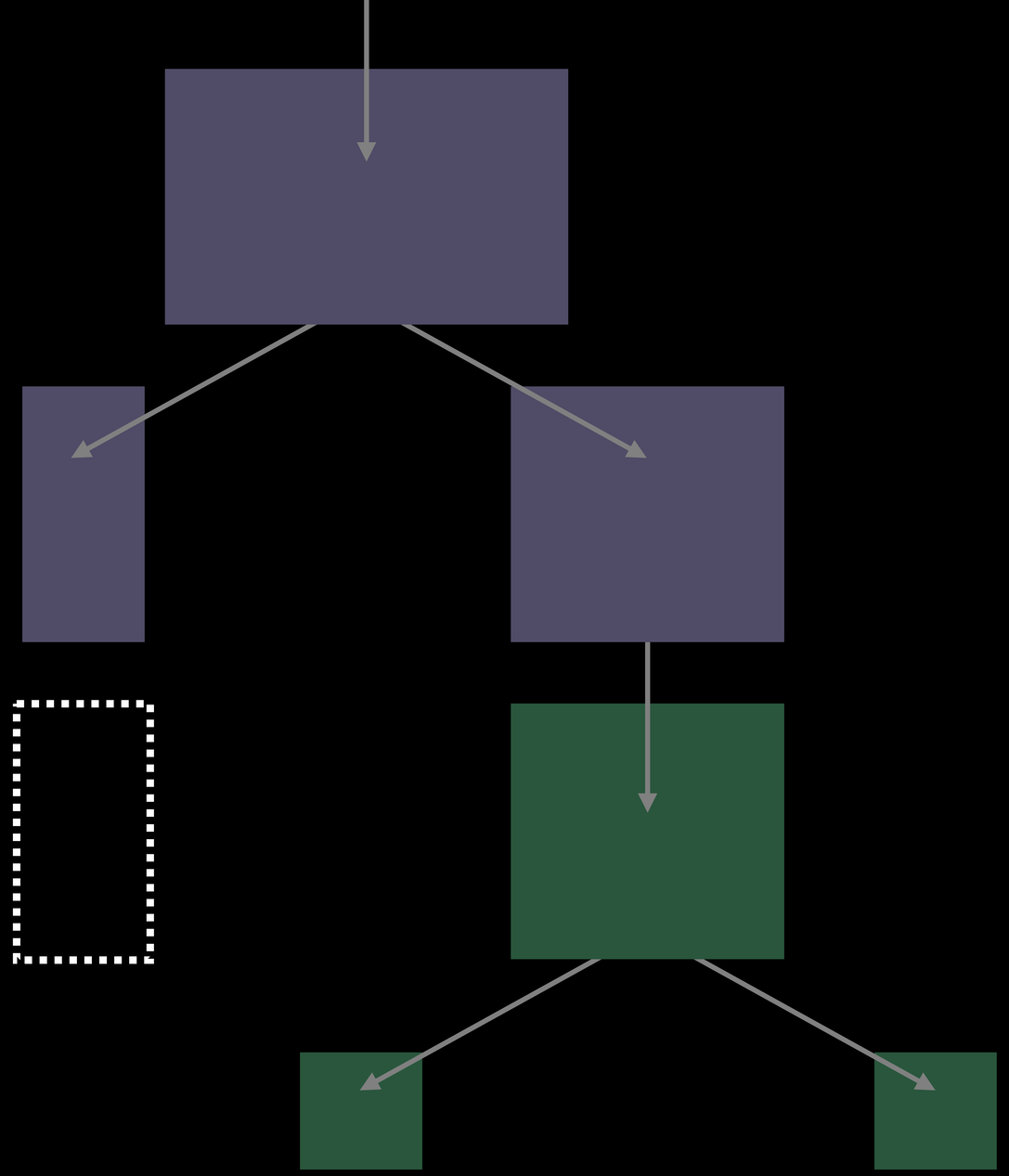

UIScreen
UIWindowScene
UIWindow
UIPresentationController
UIViewController
UIView

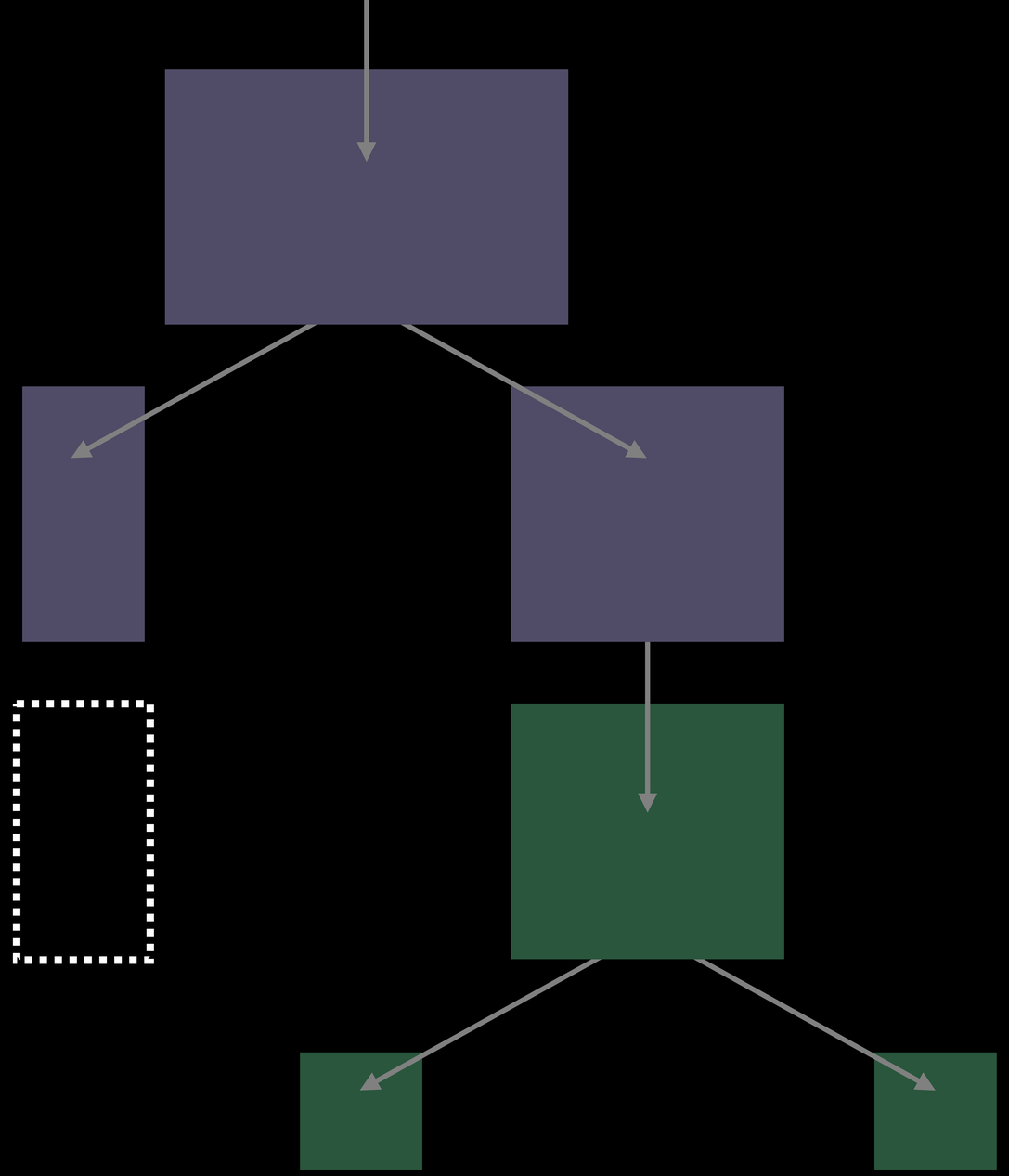

UIScreen
UIWindowScene
UIWindow
UIPresentationController
UIViewController
UIView


UIViewController
UIView

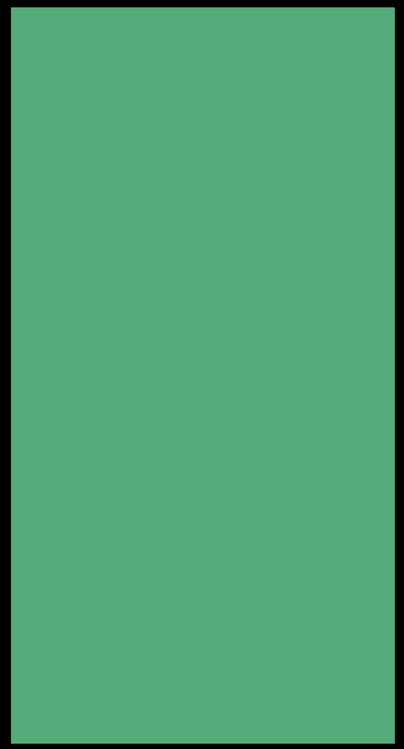

UIViewController
UIView

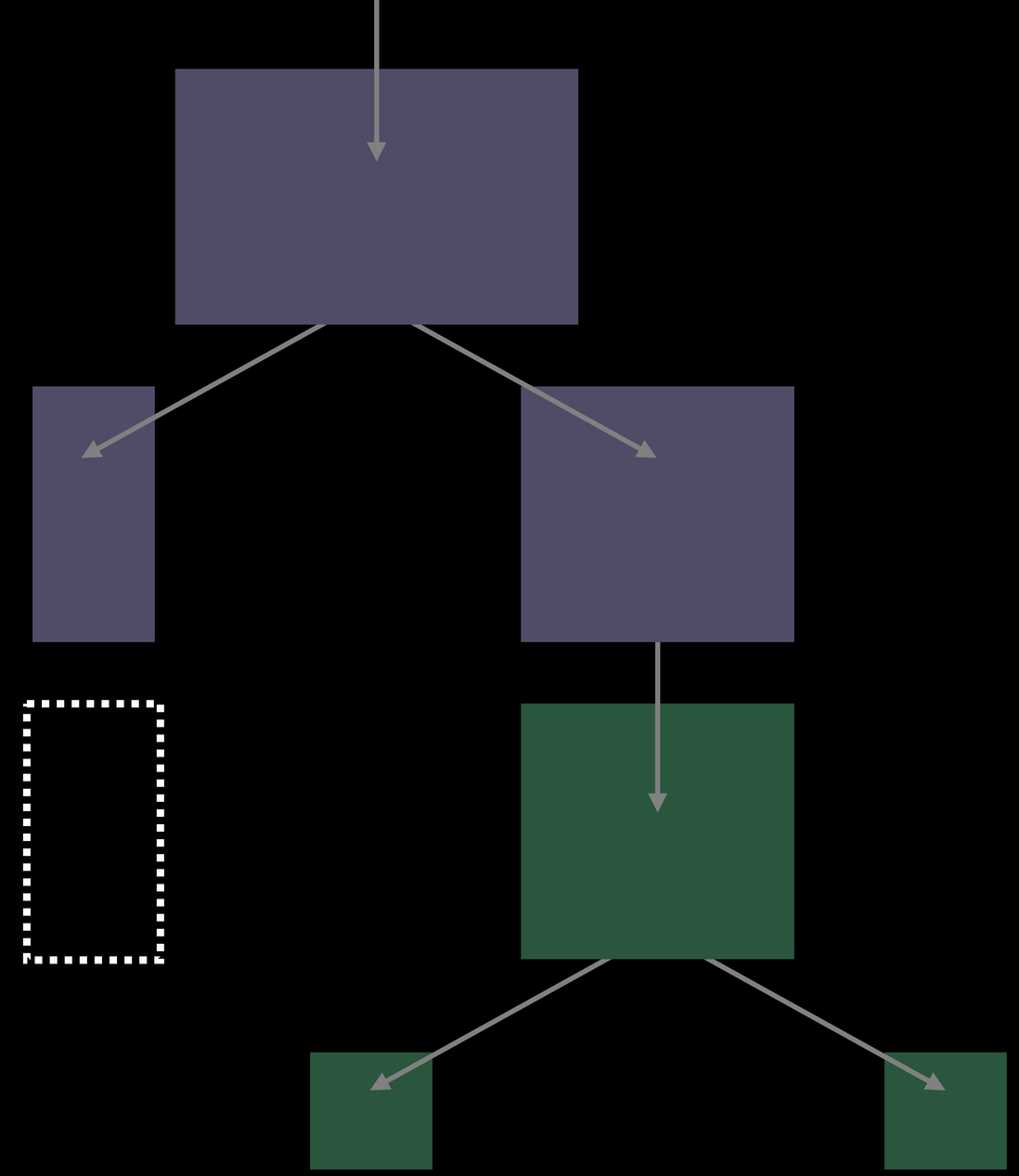

UIViewController
UIView



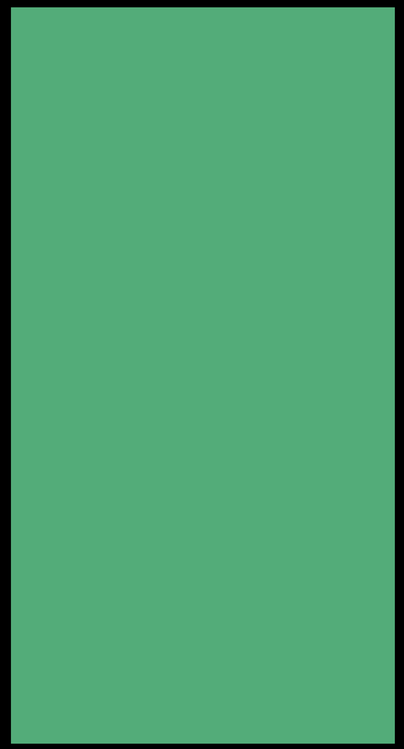


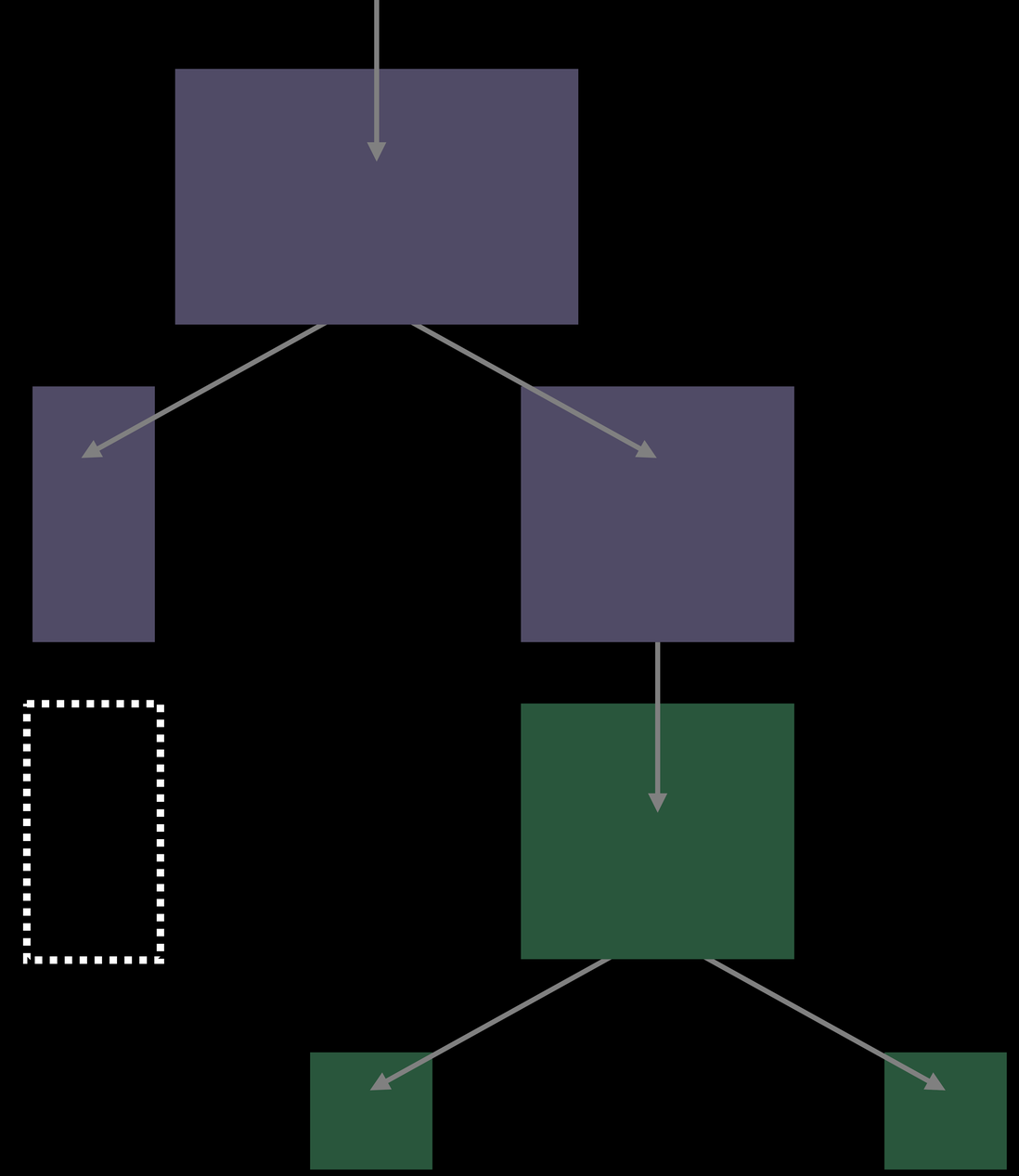
```
let view = UIView()
```



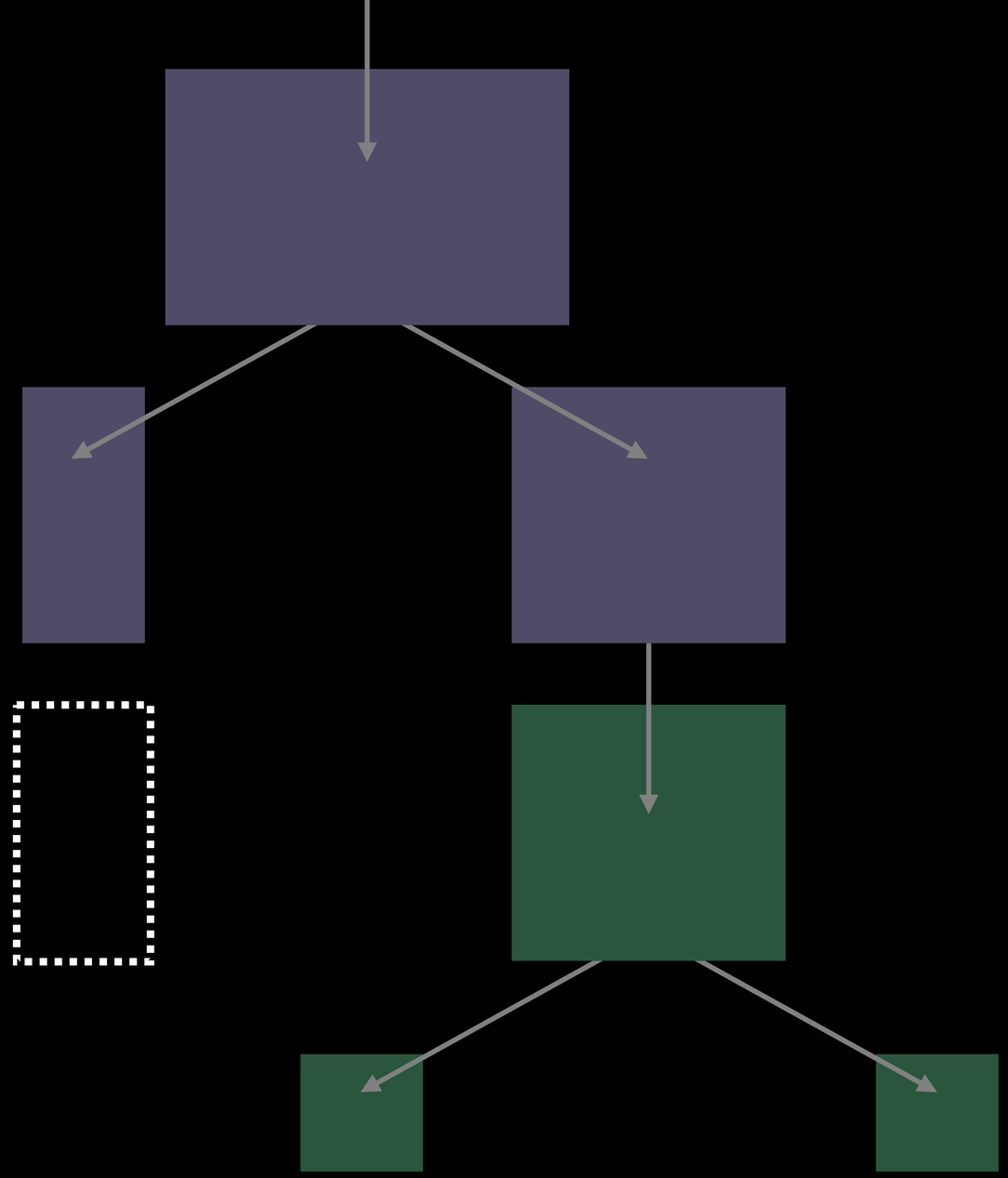


```
let view = UIView()
```

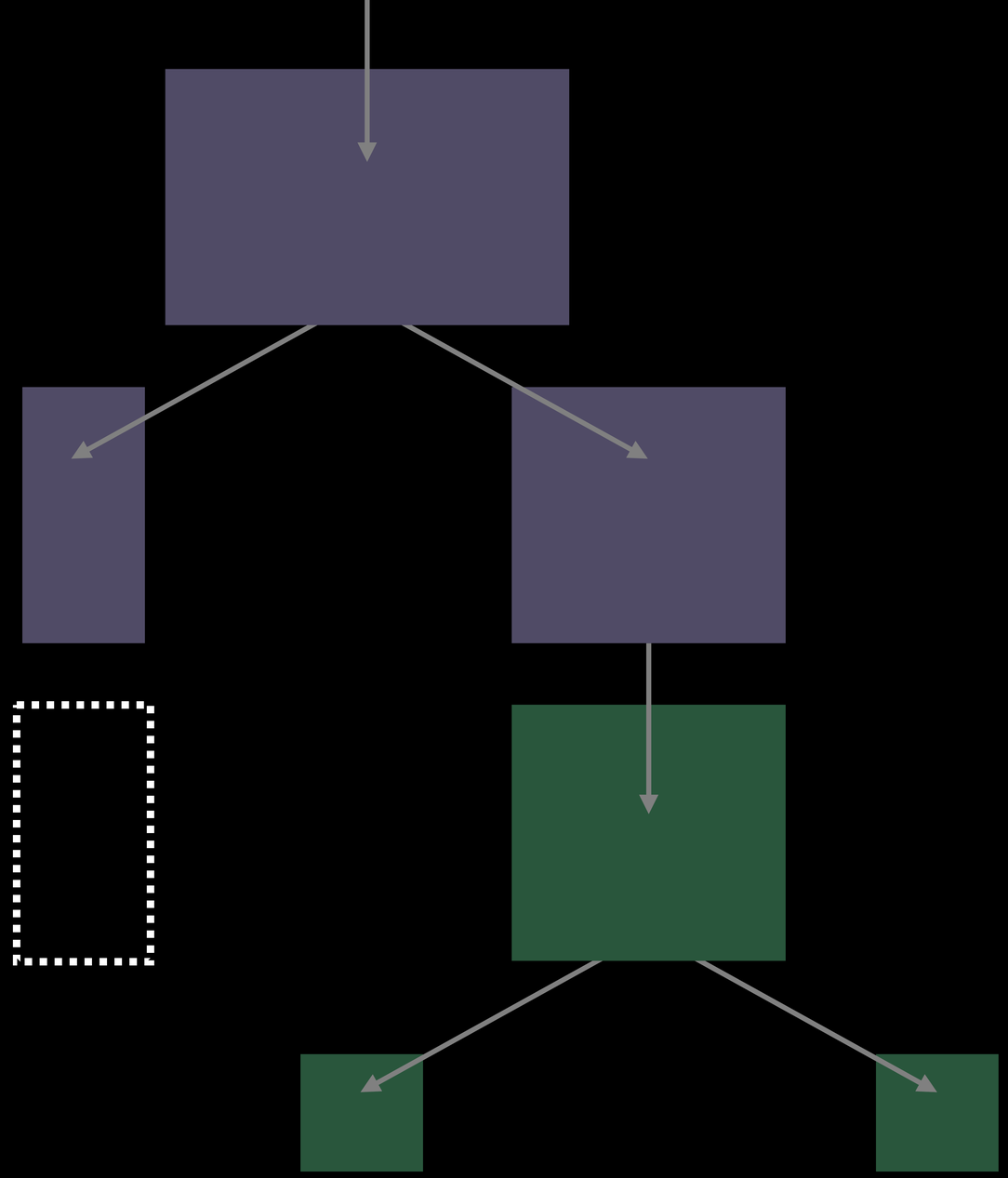


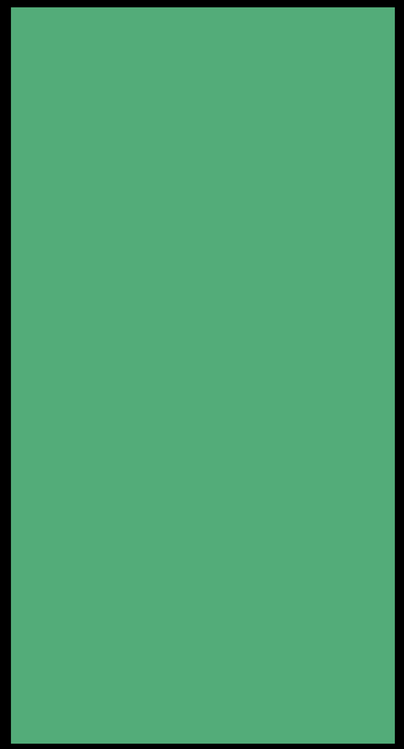


```
let view = UIView()
```

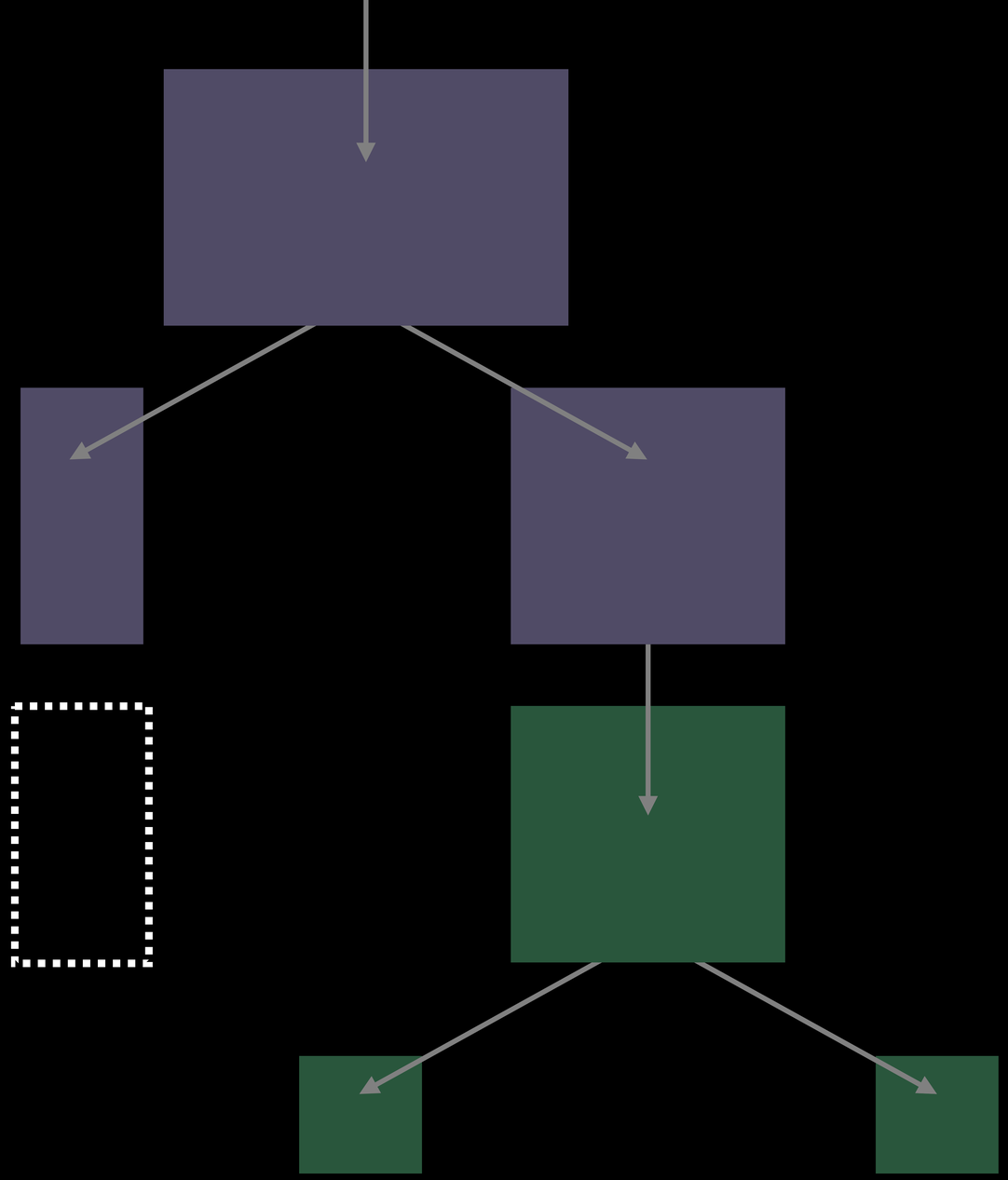


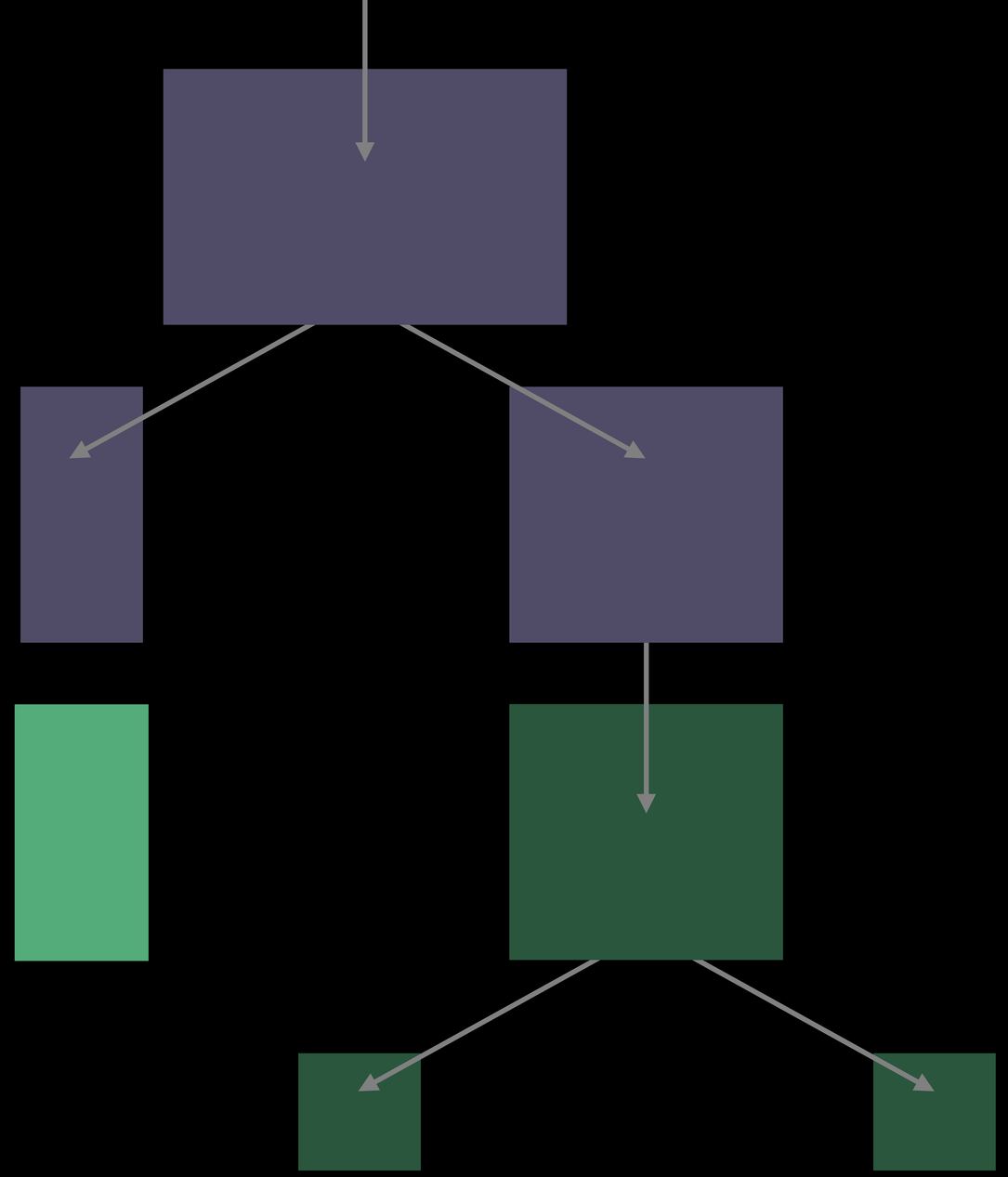
```
let view = UIView()
```



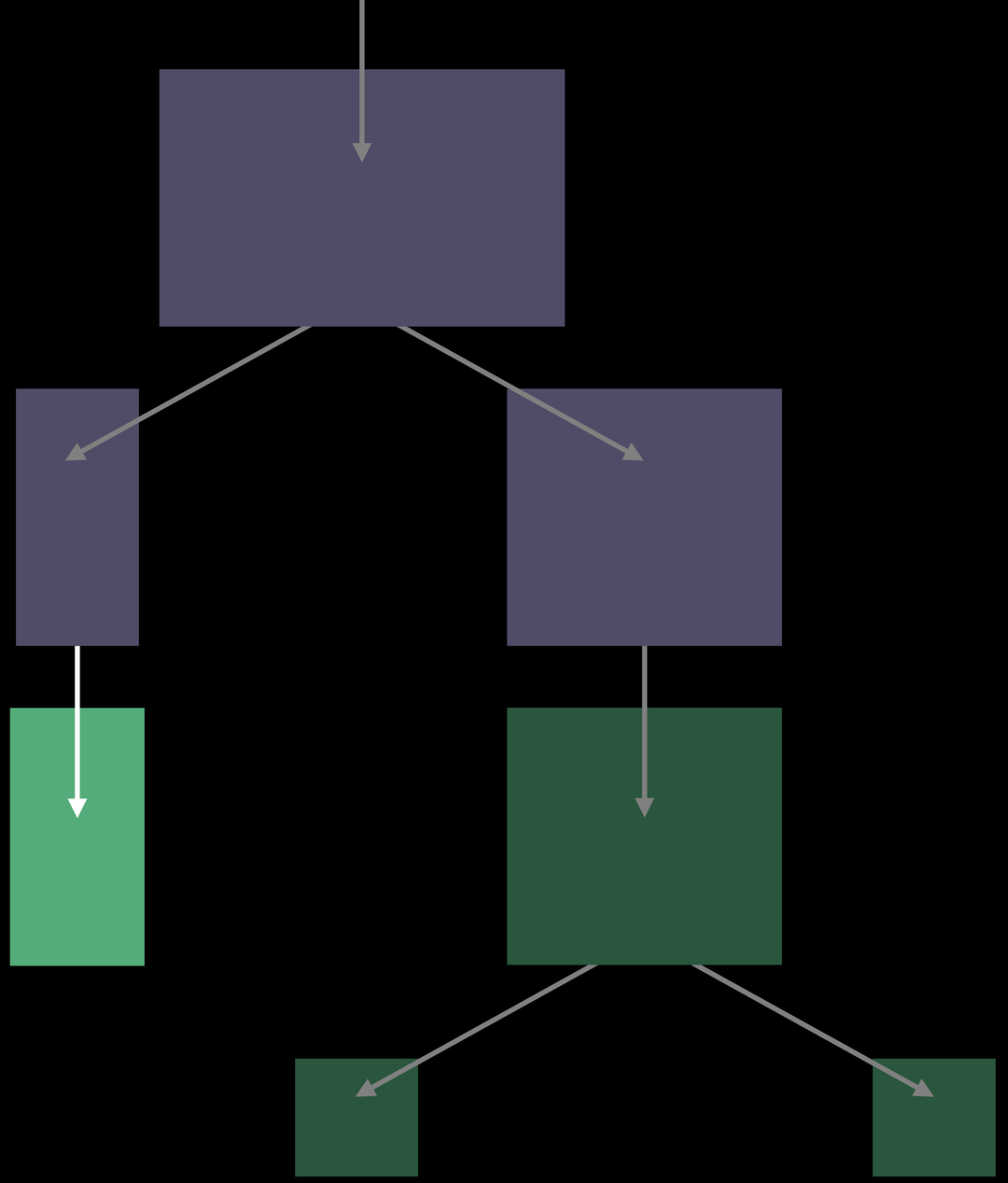


```
addSubview(view)
```





```
addSubview(view)
```



Trait Collection Changes in iOS 13

NEW

Traits are predicted during initialization

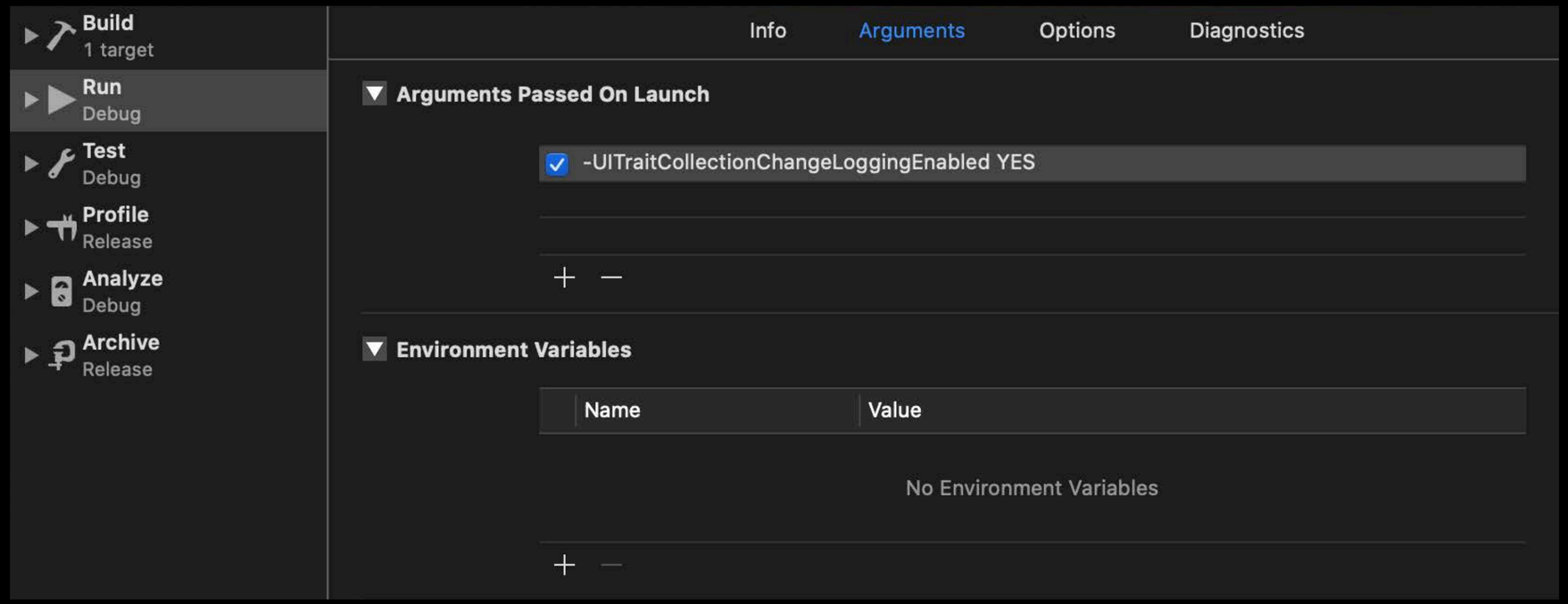
`traitCollectionDidChange(_:)` called only for changes

Trait Collection Changes in iOS 13

NEW

Enable debug logging with launch argument

```
-UITraitCollectionChangeLoggingEnabled YES
```



Using Trait Collections

Layout is the best time to use traits

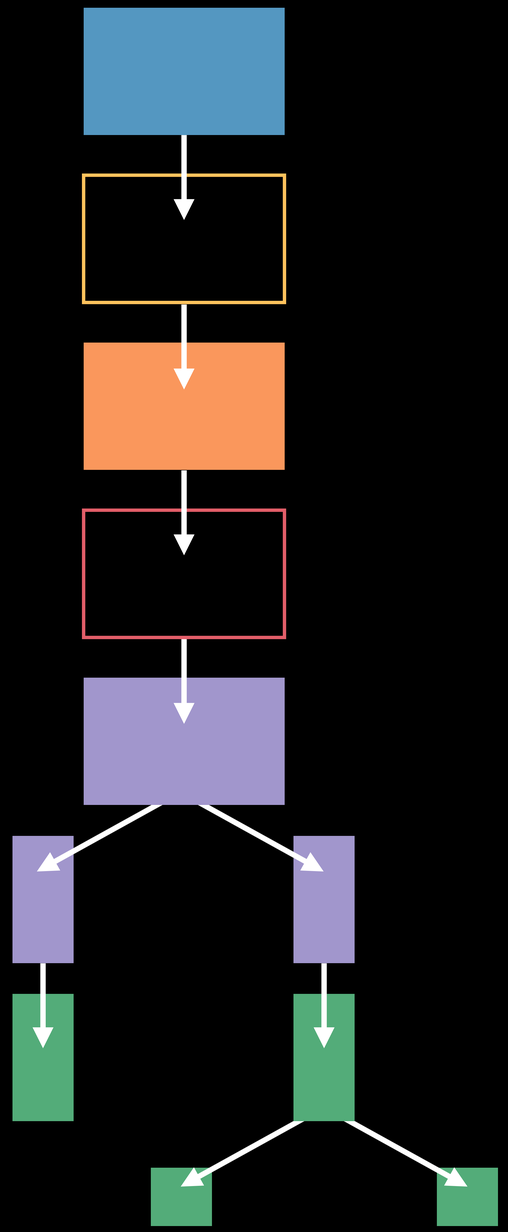
```
UIViewController.viewWillLayoutSubviews()
```

```
UIView.layoutSubviews()
```

```
UIViewController.viewDidLayoutSubviews()
```


9:41
Large Title
Lorem ipsum dolor sit er elit lamet, consectetaur cillium adipisicing pecu, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Learn More
Home
Search


9:41
Large Title
Lorem ipsum dolor sit er elit lamet, consectetaur cillium adipisicing pecu, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.
Learn More
Home
Search



UIScreen

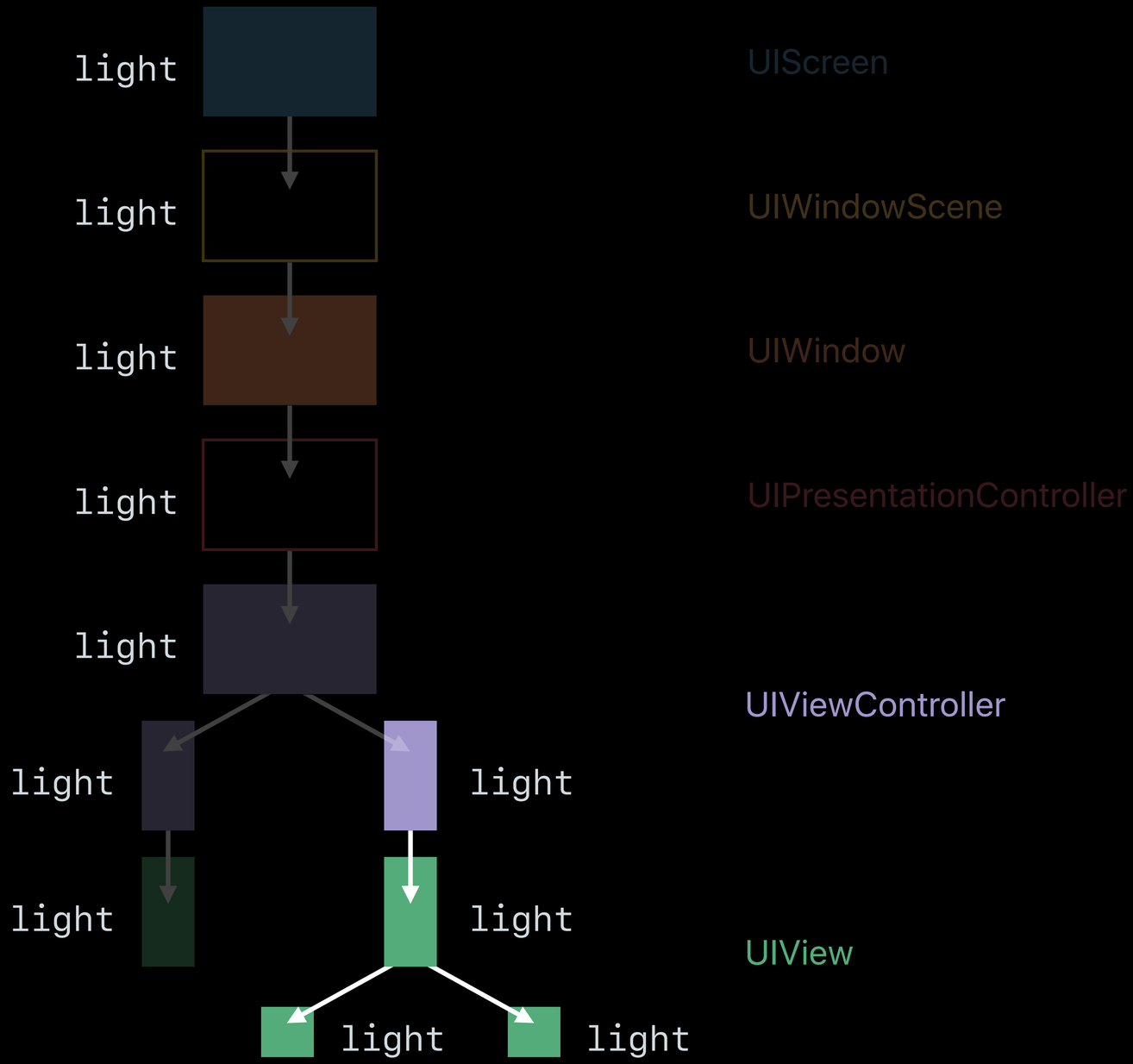
UIWindowScene

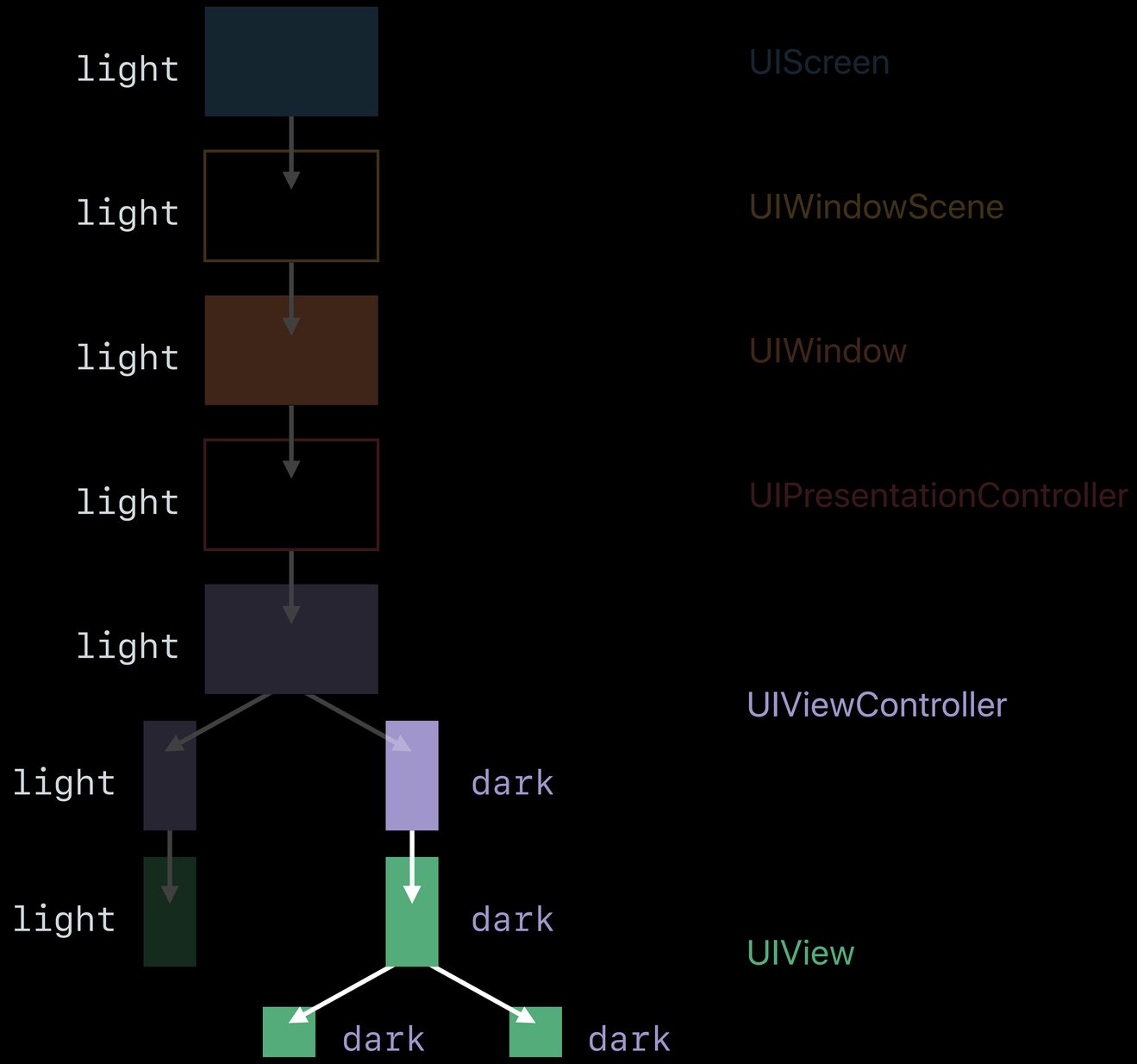
UIWindow

UIPresentationController

UIViewController

UIView


light
light
light
light
light
light
light
light
light
light
UIScreen
UIWindowScene
UIWindow
UIPresentationController
UIViewController
UIView


light
light
light
light
light
light
light
dark
dark
dark
dark
UIScreen
UIWindowScene
UIWindow
UIPresentationController
UIViewController
UIView

Overriding User Interface Style

NEW

```
class UIViewController {
    var overrideUserInterfaceStyle: UIUserInterfaceStyle
}
```

```
class UIView {
    var overrideUserInterfaceStyle: UIUserInterfaceStyle
}
```

Overriding User Interface Style

NEW

```
class UIViewController {
    var overrideUserInterfaceStyle: UIUserInterfaceStyle
}
```

```
class UIView {
    var overrideUserInterfaceStyle: UIUserInterfaceStyle
}
```

For entire app, set Info.plist key `UIUserInterfaceStyle` to `Light` or `Dark`

Overriding Traits

Existing API to override any traits

```
class UIPresentationController {
   var overrideTraitCollection: UITraitCollection?
}
```

```
class UIViewController {
    func setOverrideTraitCollection(_: UITraitCollection?, forChild: UIViewController)
}
```

Only specify values for traits you want to override

Dark Mode API Updates

Status Bar

Before iOS 13

Status Bar

iOS 13

NEW

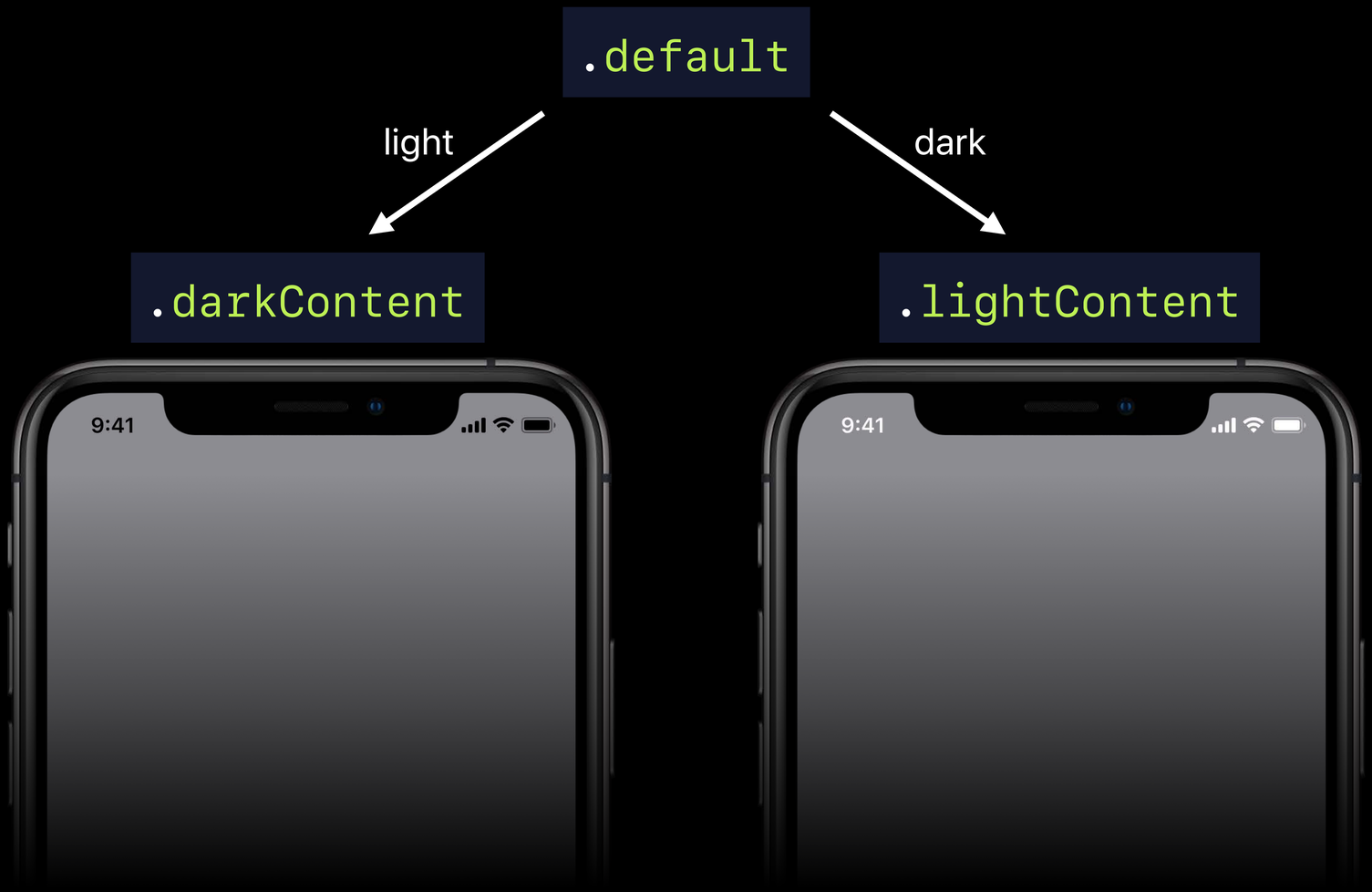
.default

.lightContent

Status Bar

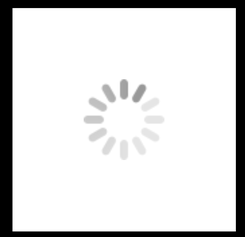
iOS 13

NEW

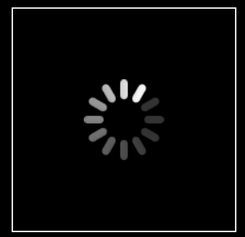


UIActivityIndicatorView

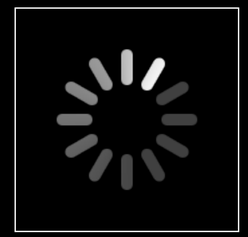
Deprecated styles



`.gray`



`.white`



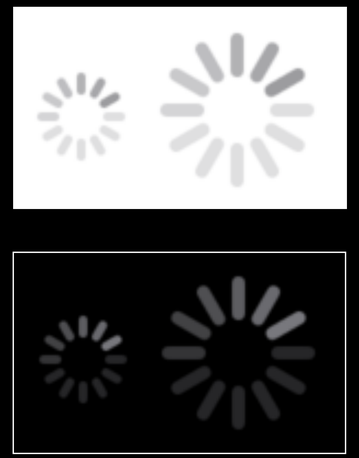
`.whiteLarge`

UIActivityIndicatorView

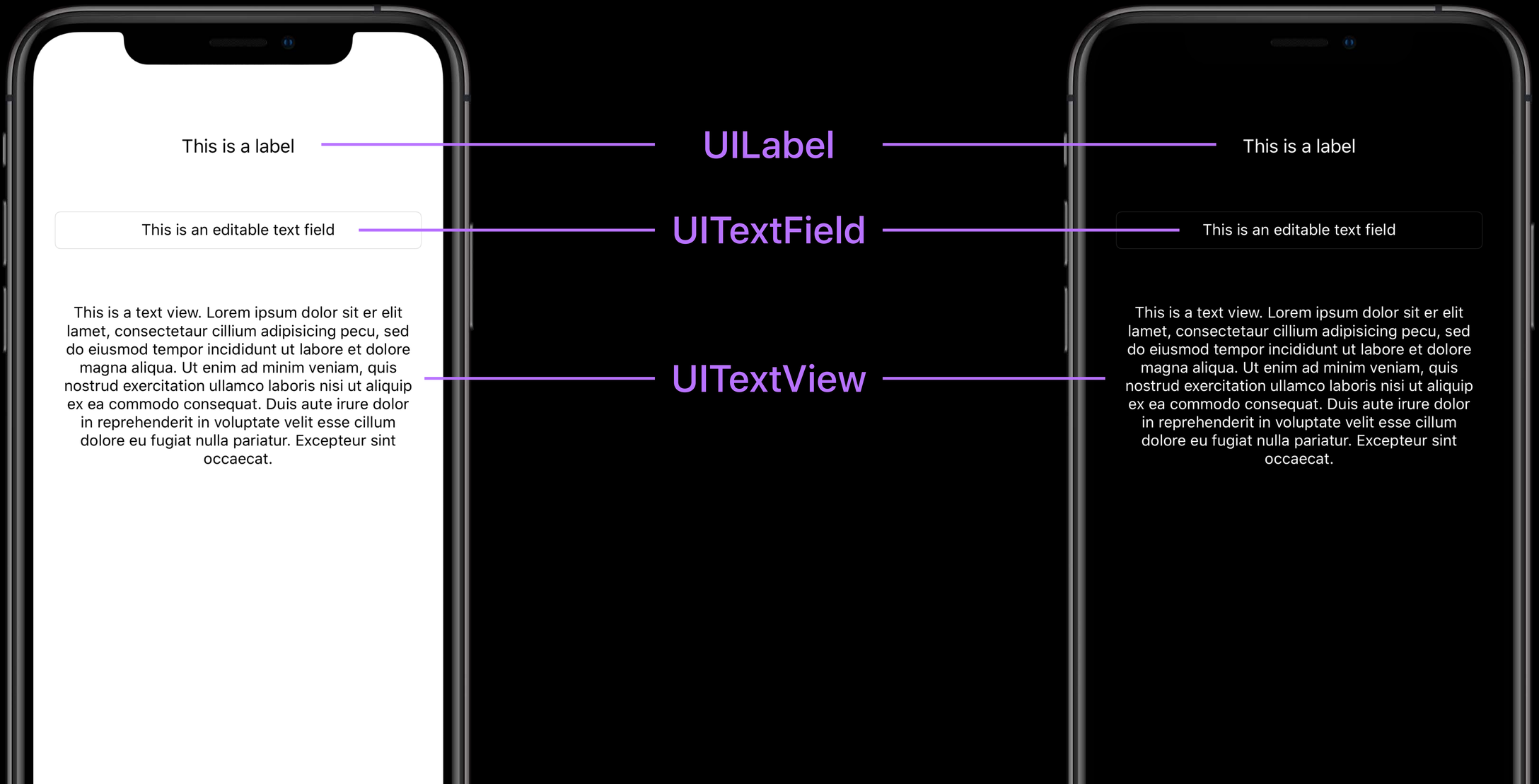
NEW

New dynamic styles `.medium` and `.large`

Use the `color` property to set your own



Drawing Text



Drawing Attributed Text

```
let attributes: [NSAttributedString.Key: Any] = [
    .font: UIFont.systemFont(ofSize: 36.0)
]
```

Drawing Attributed Text

```
let attributes: [NSAttributedString.Key: Any] = [
    .font: UIFont.systemFont(ofSize: 36.0)
    .foregroundColor: UIColor.label
]
```

Drawing Attributed Text

```
let attributes: [NSAttributedString.Key: Any] = [
    .font: UIFont.systemFont(ofSize: 36.0)
    .foregroundColor: UIColor.label
]
```

Web Content

Must opt in to dark mode

Declare supported color schemes with `color-scheme`

Use `prefers-color-scheme` media query for custom colors and images

tvOS

Apps built using tvOS 13 SDK are expected to support dark mode

Most new API is available

iPad Apps for Mac

Same API

Follows system setting

Matches AppKit colors and materials

Apps on iOS 13 are expected to support dark mode

Apps on iOS 13 are expected to support dark mode

Use system colors and materials

Apps on iOS 13 are expected to support dark mode

Use system colors and materials

Create your own dynamic colors and images

Apps on iOS 13 are expected to support dark mode

Use system colors and materials

Create your own dynamic colors and images

Leverage flexible infrastructure

More Information

developer.apple.com/wwdc19/214


WWDC19