Intro to tvOS

CS193W - Spring 2016 - Lecture 7

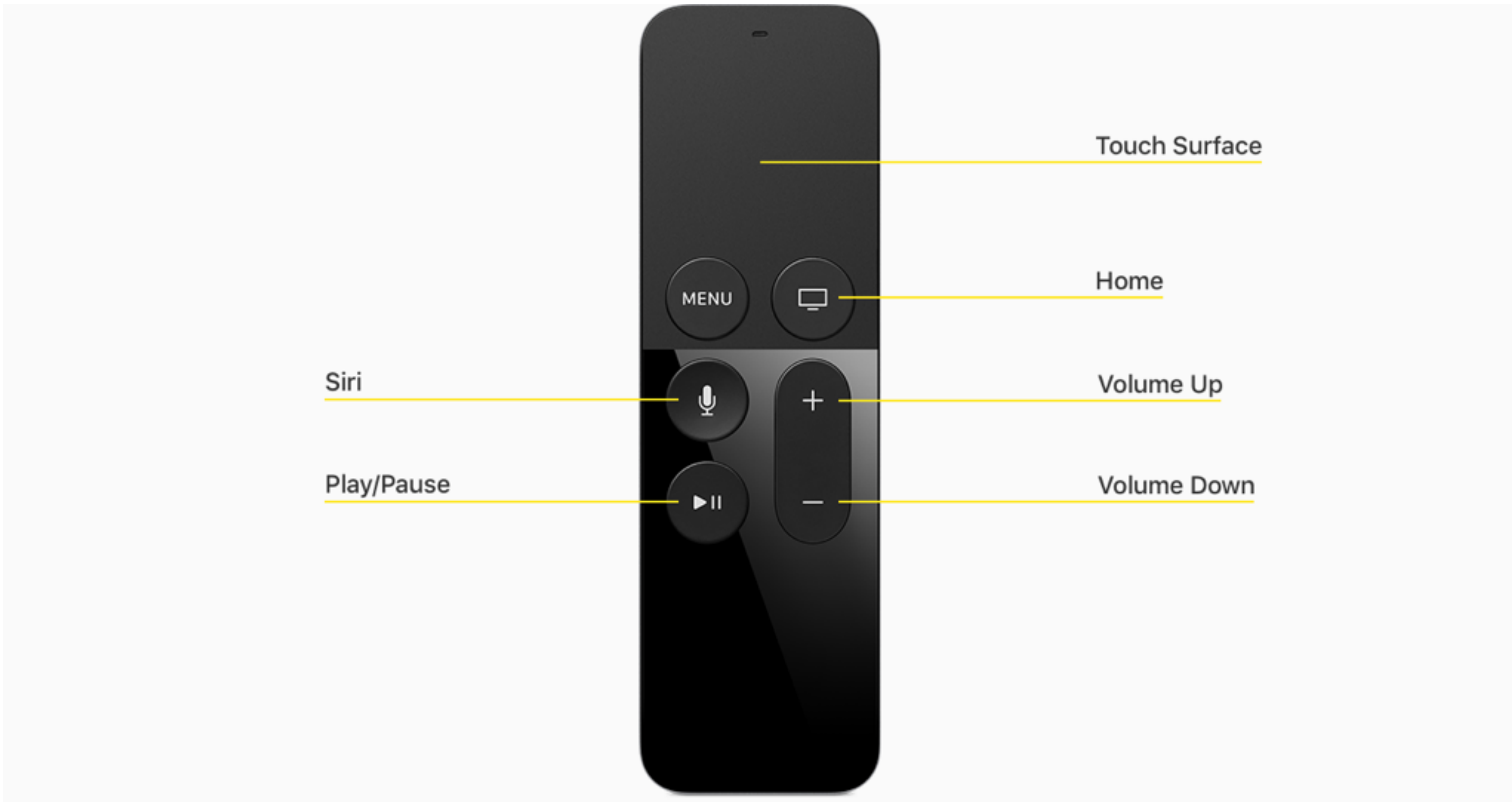
Apple TV

- Apple’s “most communal” device
- Always connected to (fast) Internet
- Limited local storage

Apple Watch vs Apple TV

Apple Watch	Apple TV
Apple's "most personal device"	Apple's “most communal device”
Tied to a particular user	Can be used by different users
Worn on the body	Used from across the room
Moves with the user	Stationary
Tiny screen	Large screen
Often used with no connectivity	Has persistent fast conection
Limited persistent storage	Limited persistent storage
Touch screen	Remote control

The Siri Remote



Interacting Via the Touch Surface

swipe
Used to scroll with inertia

tap
Used to navigate through a collection of items one at a time

click
Used to make a selection

Plan for inadvertent taps.

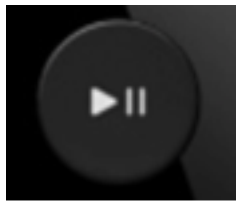
Home Button

- Works the same as the iOS Home Button
- Tapping once goes back to the home screen
- Doubling tapping brings up the list of recently used apps

Menu Button

- Works as a back button
- No need for back button UI on screen (e.g. like iOS / WatchOS have for Navigation Controllers)

Play/Pause Button



- This is, primarily, still a TV
- Use this as a queue to start playing content immediately

Sensors

- The Siri remote is equipped with an **accelerometer** and a **gyroscope**
- These are used mostly for games (like the Nintendo Wii)

Apple TV UI

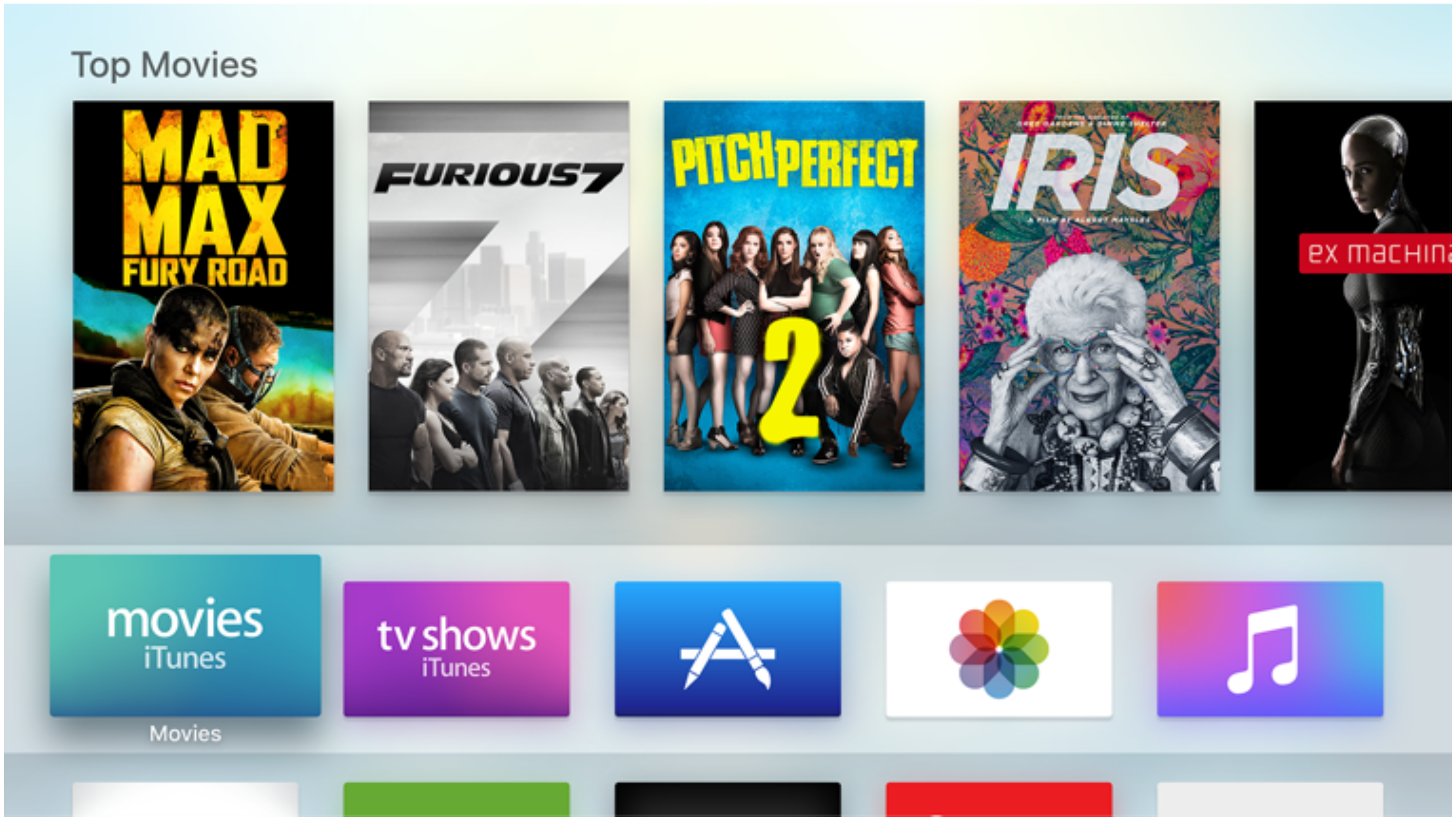
Design Principles

Connected
When the user interacts with the remote, the Apple TV should respond as if the user was directly manipulating the screen.

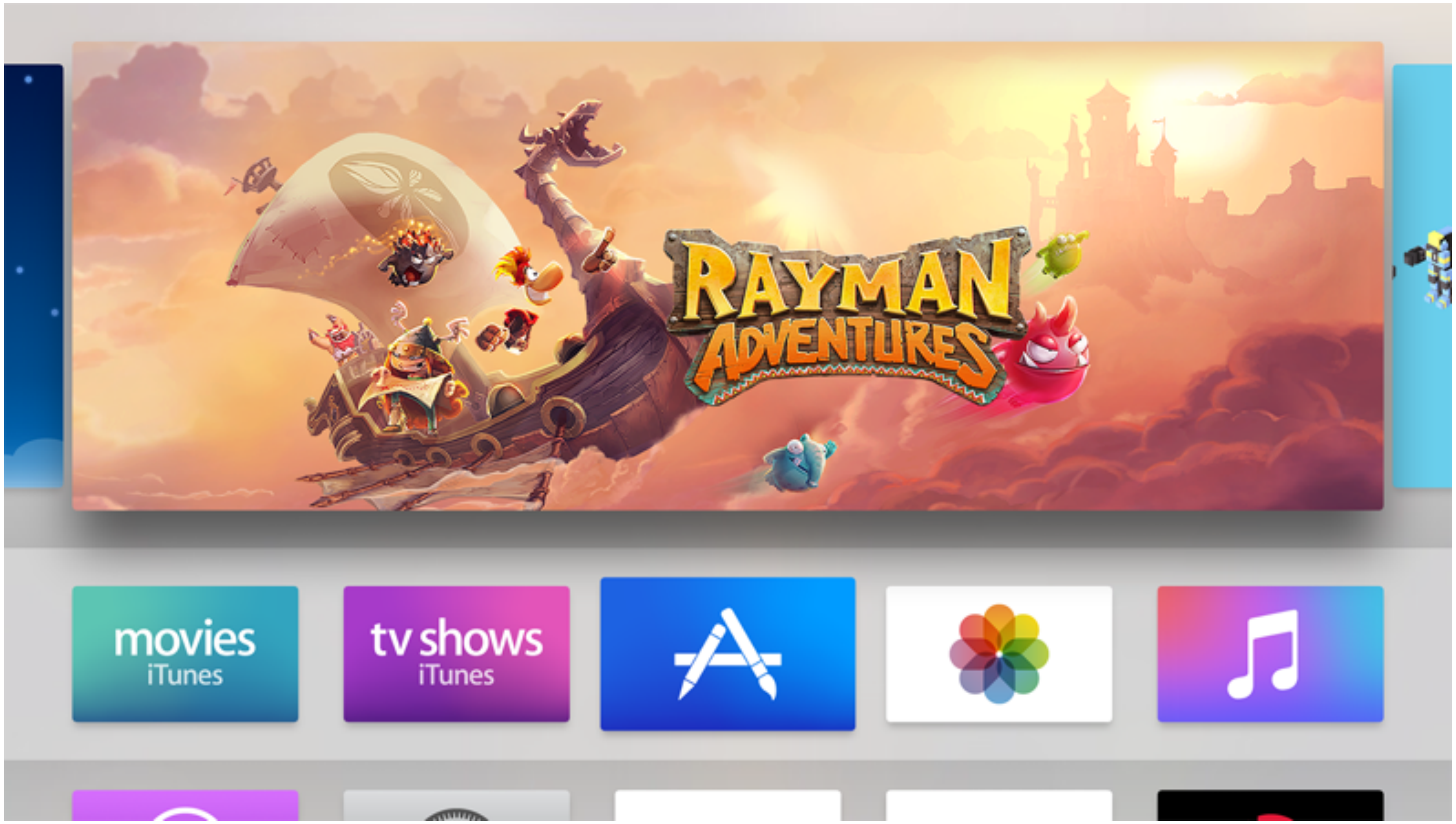
Clear
It should be obvious how things work, even from across the room.

Immersive
It's a TV! Use edge-to-edge media whenever possible.

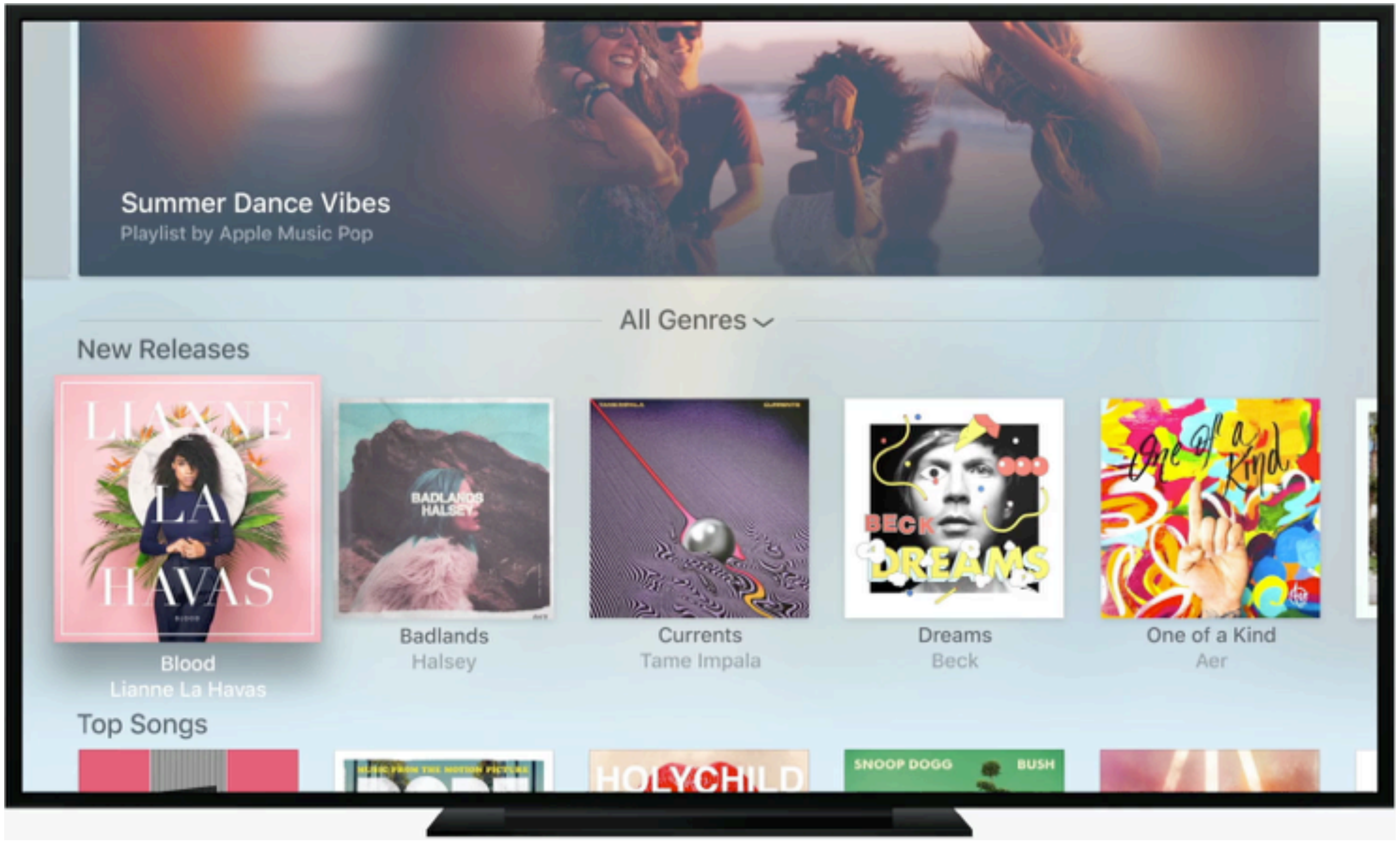
The Home Screen



The Top Shelf

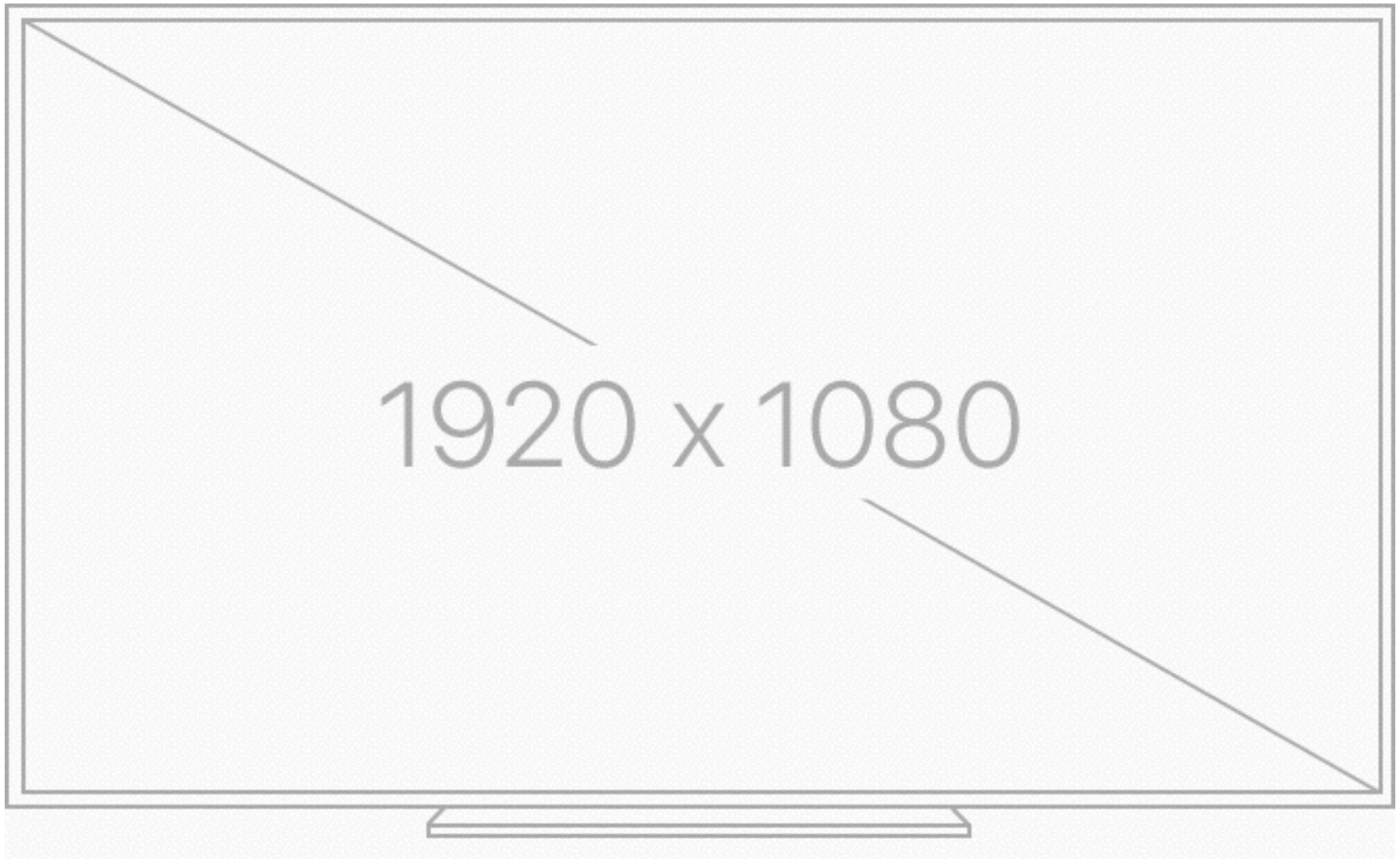


Focus

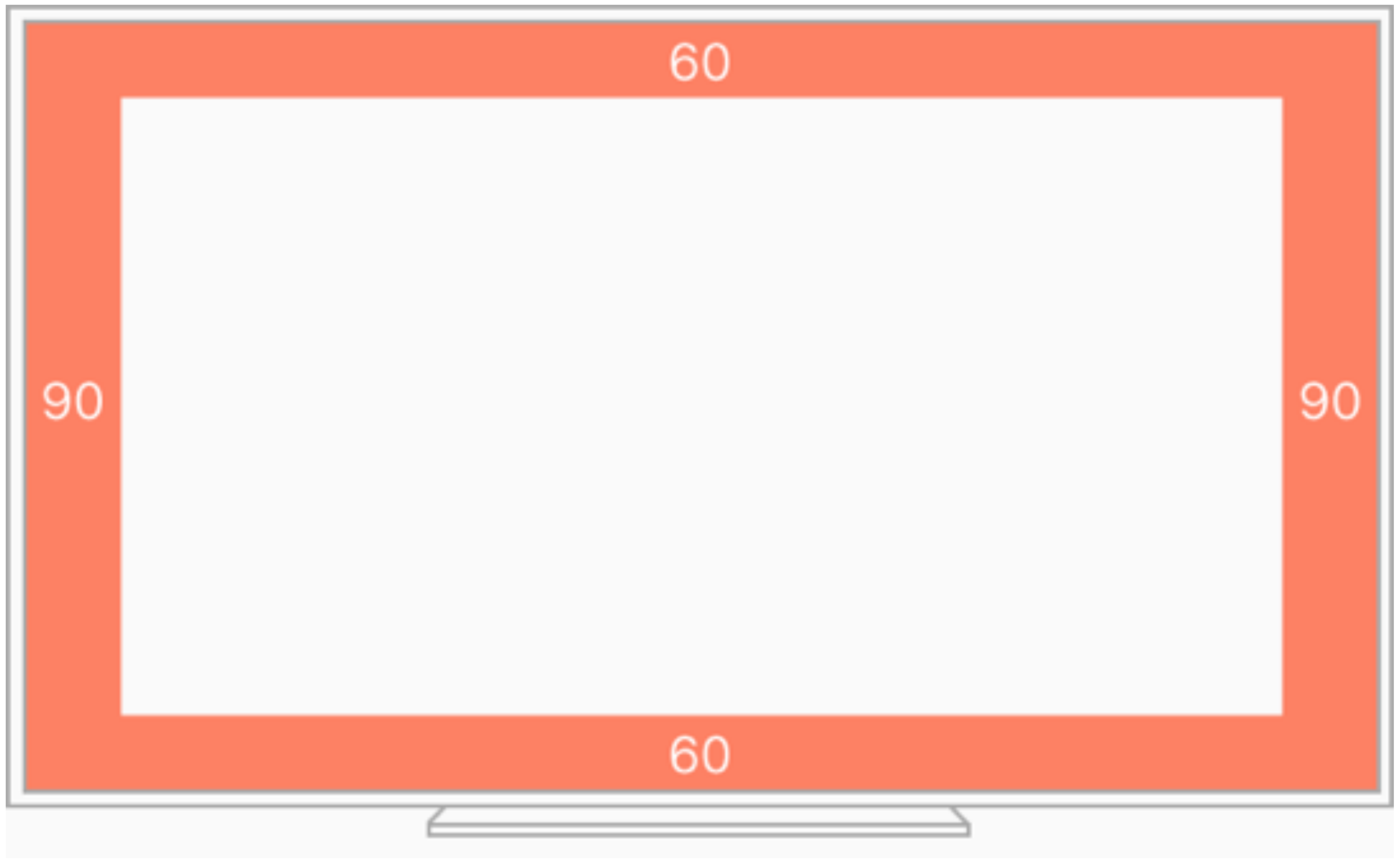

Summer Dance Vibes
Playlist by Apple Music Pop
All Genres
New Releases
Blood
Lianne La Havas
Badlands
Halsey
Currents
Tame Impala
Dreams
Beck
One of a Kind
Aer
Top Songs

Parallax

One screen size



But, cropping can occur on older TVs



- Use `UIScreen().overscanCompensationInsets` to determine the appropriate insets

UIKit / tvML

- Unlike WatchOS, tvOS supports a large portion of UIKit
- It also supports a markup language called TVML (analogous to HTML), with scripting via JavaScript
- We'll talk about UIKit today.

UIKit Interface Elements

- `UINavigationBar`
- `UITabBar`
- `UITableView`
- `UICollectionView`
- `UIAlertController`
- `UISearchController`
- `UILabel`
- `UITextField`
- `UITextView`

and many more…

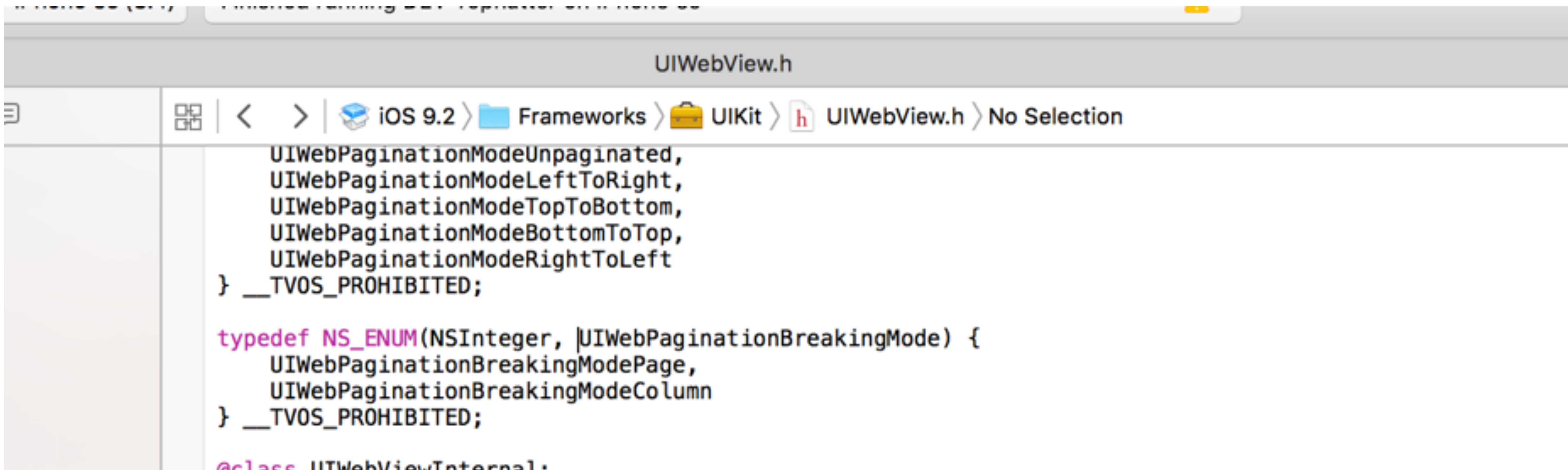
Differences in iOS and tvOS UIKit

Notable UIKit Interface Elements not in tvOS

- `UIDatePicker`
- `UIImagePickerController`
- `UIRefreshControl`
- `UISlider`
- `UISwitch`
- `UIToolbar`
- `UIWebView`

__TVOS_PROHIBITED

- tvOS uses the same UIKit API as iOS, but marks unsupported APIs with `__TVOS_PROHIBITED`
- e.g. `UIWebView` is unsupported



```
    UIWebPaginationModeUnpaginated,
    UIWebPaginationModeLeftToRight,
    UIWebPaginationModeTopToBottom,
    UIWebPaginationModeBottomToTop,
    UIWebPaginationModeRightToLeft
} __TVOS_PROHIBITED;

typedef NS_ENUM(NSInteger, UIWebPaginationBreakingMode) {
    UIWebPaginationBreakingModePage,
    UIWebPaginationBreakingModeColumn
} __TVOS_PROHIBITED;

@class UIWebViewInternal;
```

UIButton in tvOS

Use the `.PrimaryActionTriggered` event (as opposed to, say, `.TouchUpInside`):

```
button5.addTarget(self, action: "tappedButton",
forControlEvents: .PrimaryActionTriggered)
```

Tap Gesture Recognizers

- `UITapGestureRecognizer` works as expected. You can set `allowedTouchTypes` to a `UIPressType`:

`.Select` - the default (pressing the touch surface)

`.Menu` - the menu button

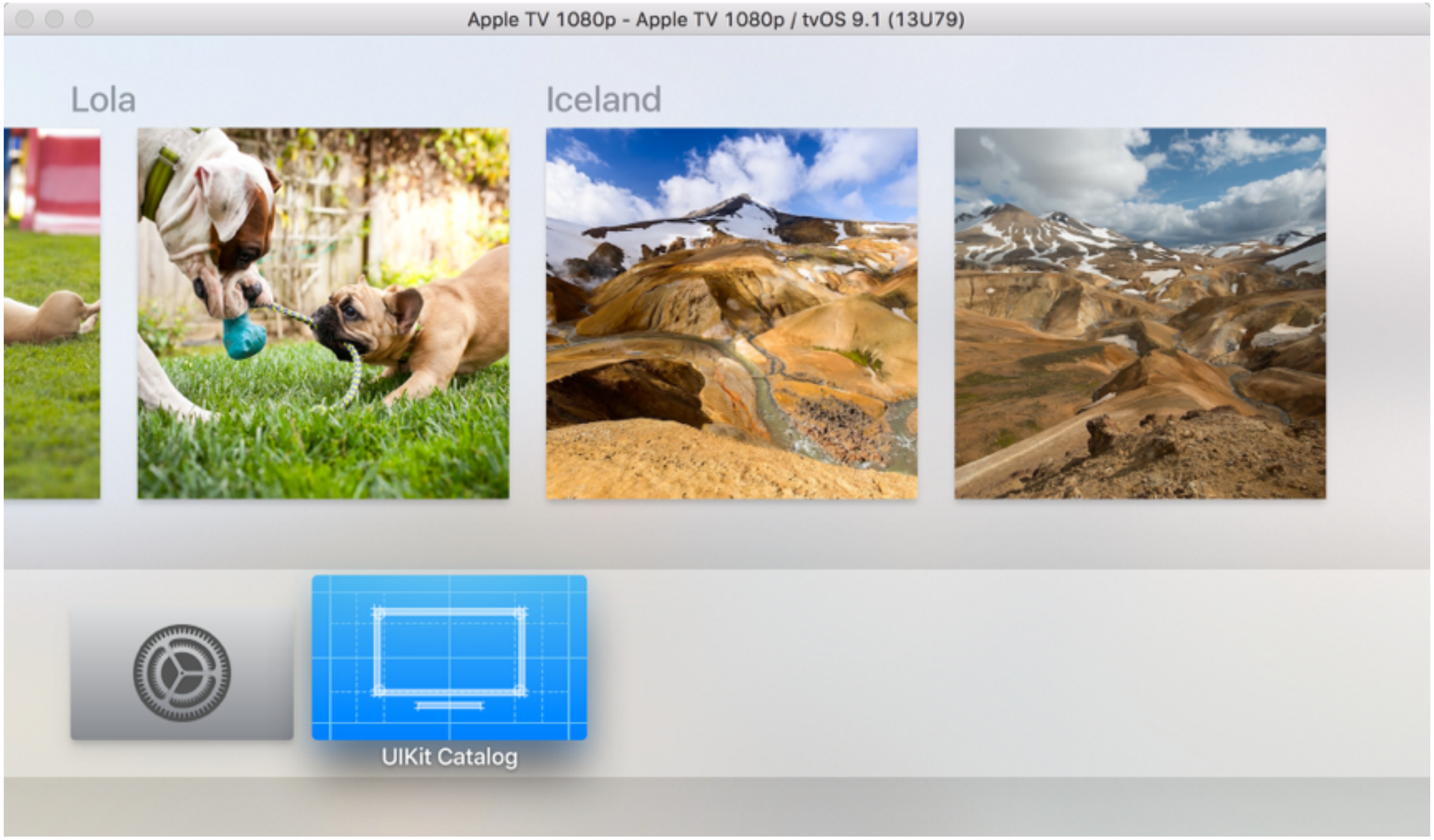
`.PlayPause` - the play/pause button

```
tapRecognizer.allowedPressTypes
 = [NSNumber(integer: UIPressType.PlayPause.rawValue)];
```

Low Level Press Event Handling

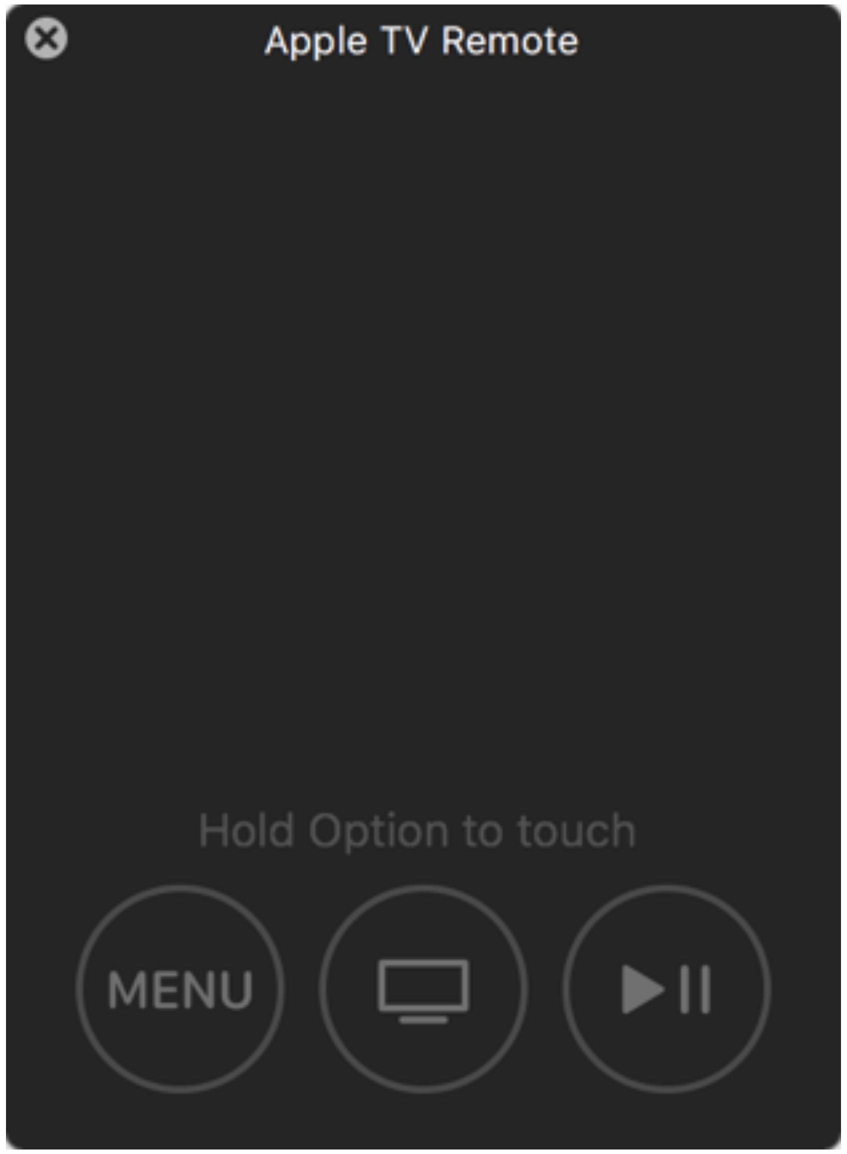
- `UIPress` is analogous to `UITouch`
- You can use the `UIPressesBegan` / `UIPressesEnded` / `UIPressesChanged` / `UIPressesCanceled` event handlers analogous to `UITouchesBegan` / `Ended` / `Changed` / `Canceled`
- Use `pressType` to get the button pressed

Debugging tvOS Apps in the Simulator



Apple TV Remote Simulator

- In Simulator, go to Hardware -> Show Apple TV Remote
- Hold Option and move your finger around on your trackpad to simulate touches/swipes
- Click the trackpad to click



Navigation

- There are two modes of navigation on tvOS
  1. The Focus Engine
  2. The Game Controller Framework

Navigating Using the Focus Engine

- Each view has an initially focused subview called the *preferred focus view*
- The user can then navigate to other subviews by swiping the touch surface
  - Horizontally, vertically, diagonally all work

canBecomeFocused

- `UIView` has a method `canBecomeFocused()` which is used to determine if a view can become focused
- In addition a view is not focusable if is *non-interactible*:
  - It is hidden
  - It has `alpha = 0`
  - `userInteractionEnabled = false`
  - It is not in the current view hierarchy

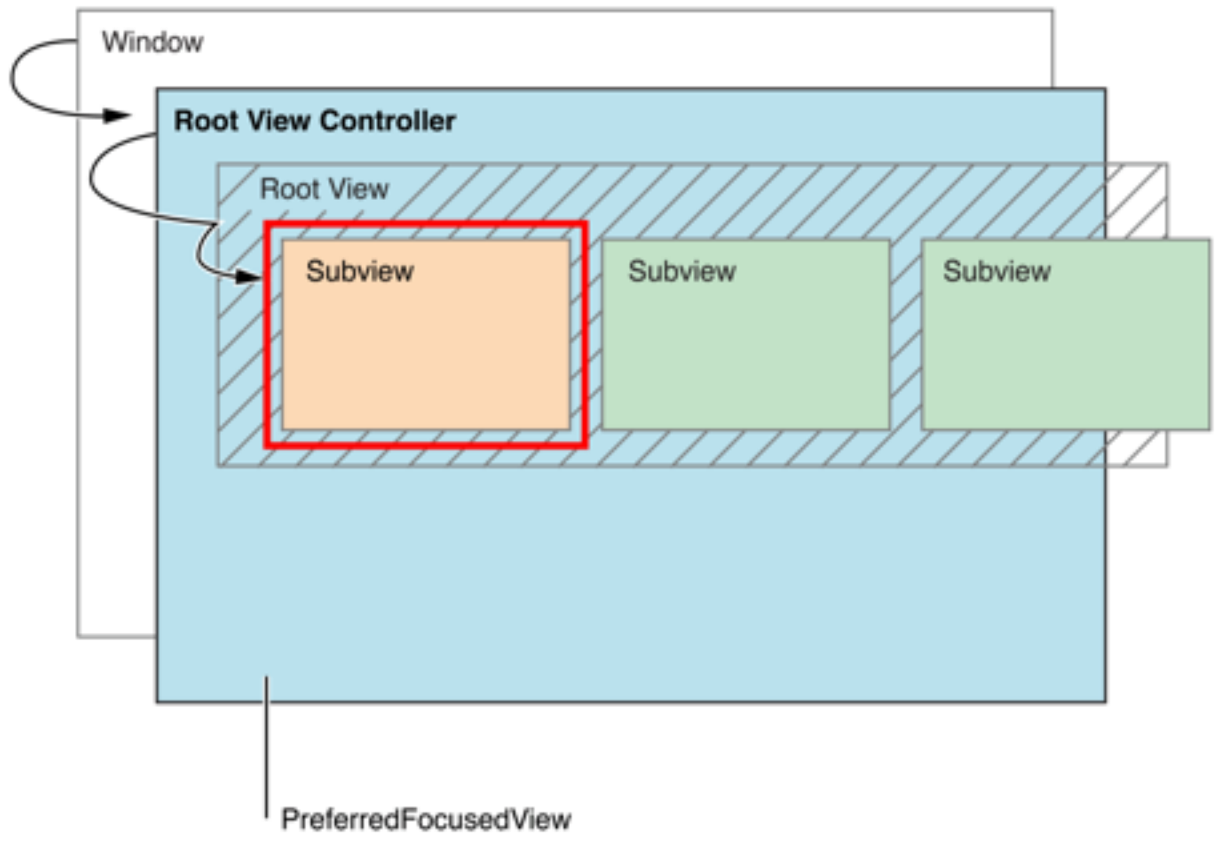
Focusable UIViews

- The following `UIKit` classes are focusable:
- `UIButton`
- `UIControl`
- `UISegmentedControl`
- `UITabBar`
- `UITextField`
- `UISearchBar` (or more specifically, its internal text field)
- And optionally, `UITableViewCell` / `UICollectionViewCell`

Getting the Current Focused View

- Use `UIScreen`'s `focusedView` to determine the current focused view (read only)
- You can also use UIView's `focused` to determine if a particular view is focused

Default Preferred Focused View



- By default, the closet focusable view to the top-left corner of the screen is focused

Overriding the Default Focused View

- `UIView`, `UIViewController`, `UIWindow`, and `UIPresentationController` all conform to the `UIFocusEnviroment` protocol
- `UIFocusEnvironment`'s `preferredFocusedView` (read only) is used to determine the preferred focused view

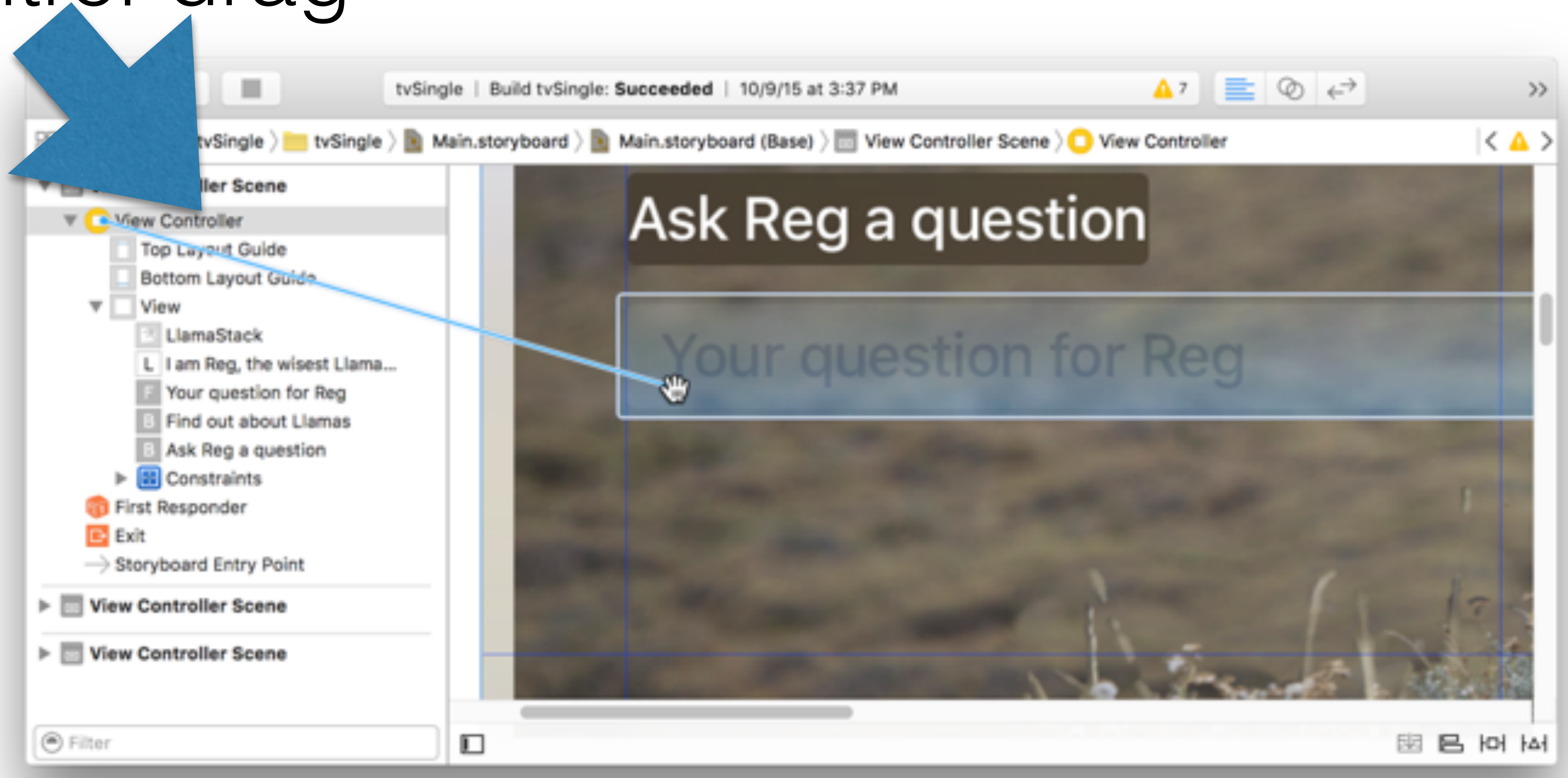
The Focus Chain

- For each view, get its `preferredFocusedView` and recurse, forming a list of views called the *Focus Chain*. The focus chain ends if a non-interactible view is encountered. The first focusable view encountered in the Focus Chain is focused.

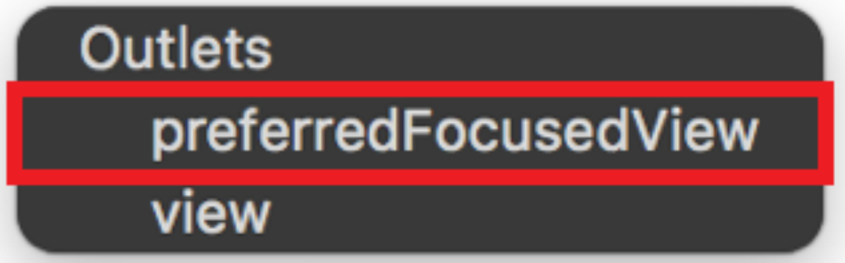
- By default, a `UIView` returns `self` (which results in the top-leftmost view being selected) and a `UIViewController` returns its root view

Setting preferredFocusedView for a View Controller in a Storyboard

Control-drag



In the popup that appears, choose preferredFocusedView



Making UITableViewCells / UICollectionViewCells focusable

```
optional func tableView(_ tableView: UITableView,
 canFocusRowAtIndexPath indexPath: NSIndexPath) -> Bool
```

```
optional func collectionView(_ collectionView: UICollectionView,
     canFocusItemAtIndexPath indexPath: NSIndexPath) -> Bool
```

By default, these return `true`.

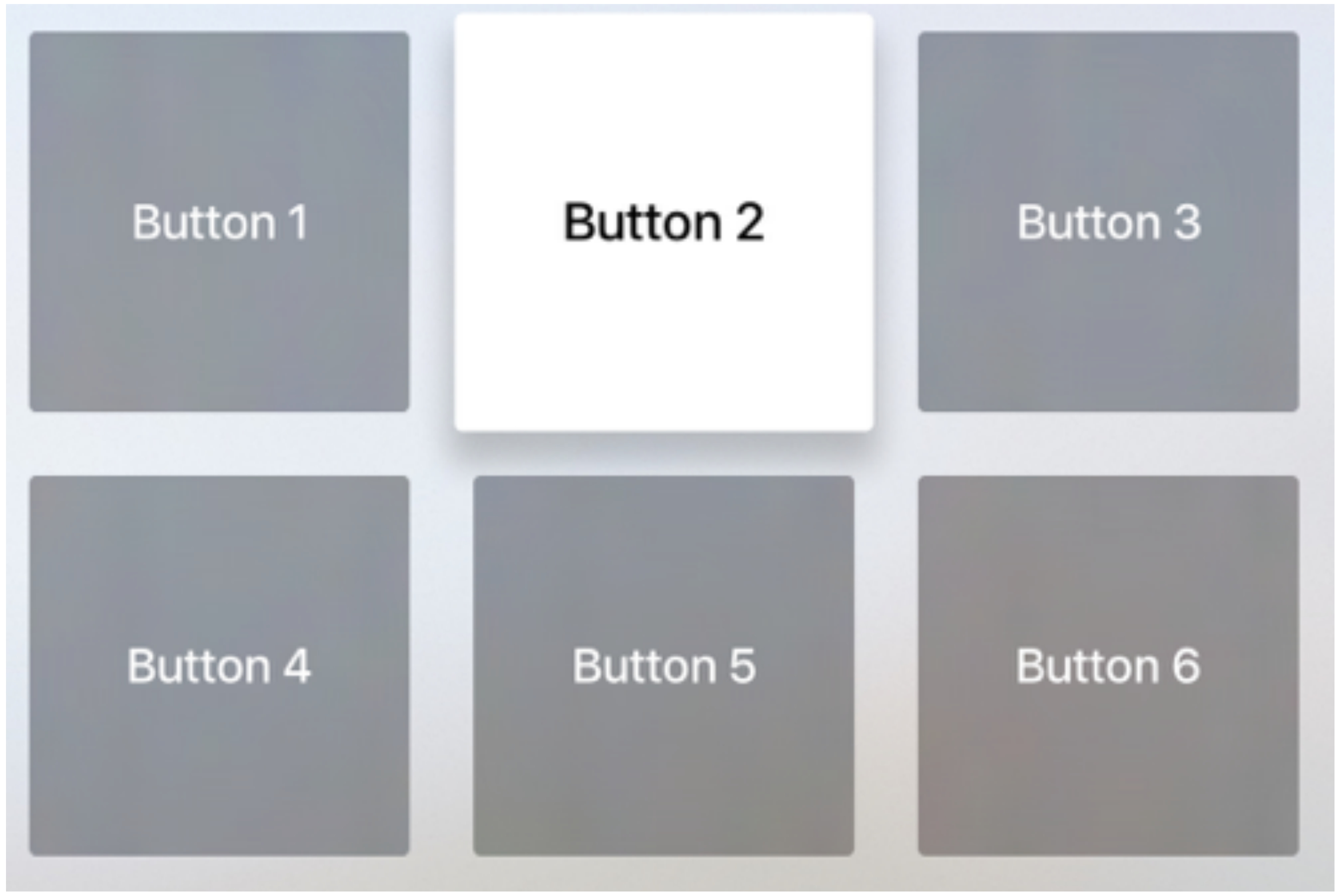
Debugging Focus Issues

- Apple recommends the use of an internal API, `_whyIsThisViewNotFocusable`

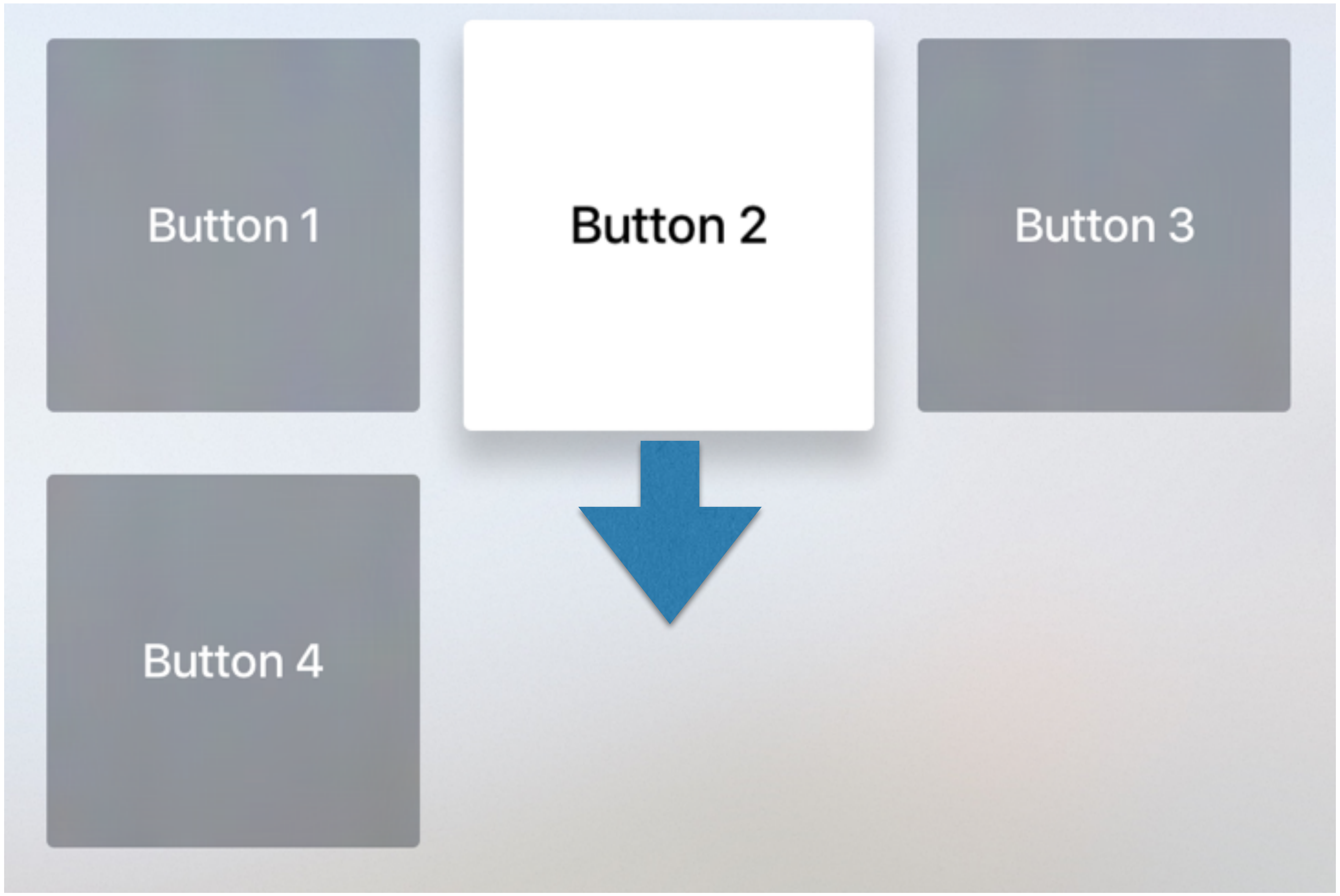
```
po self.customView.performSelector(Selector("_whyIsThisViewNotFocusable”))
```

Changing the Focus

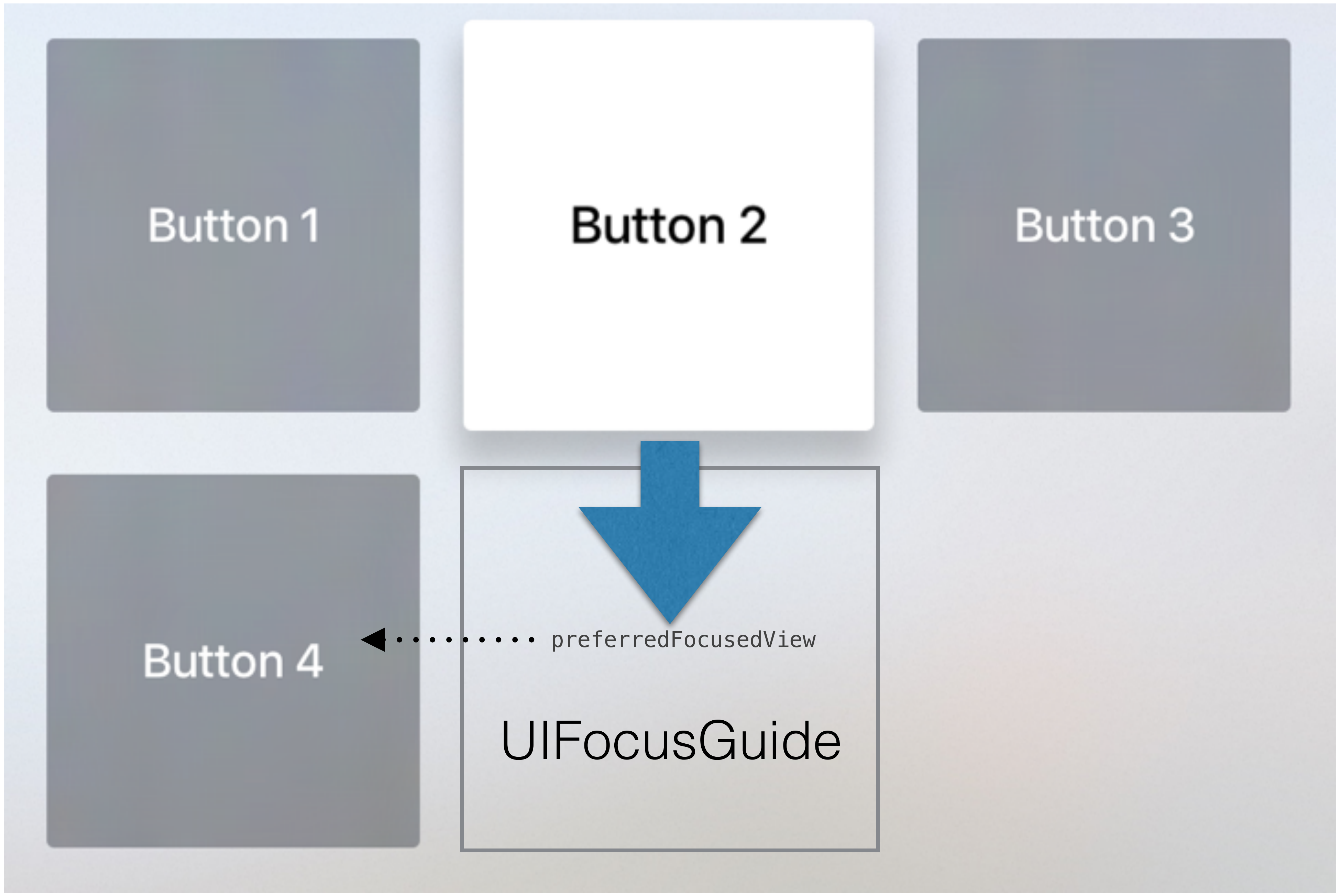
- Swiping the touch surface will cause the system to look for the next focusable view in the direction of the swipe starting from the current focused view



An interesting case



Focus Guides



Focus Guide Code

```
let focusGuide = UIFocusGuide()
self.view.addLayoutGuide(focusGuide)

button2.leftAnchor.constraintEqualToAnchor(focusGuide.leftAnchor).active = true
button2.rightAnchor.constraintEqualToAnchor(focusGuide.rightAnchor).active = true
button4.topAnchor.constraintEqualToAnchor(focusGuide.topAnchor).active = true
button4.bottomAnchor.constraintEqualToAnchor(focusGuide.bottomAnchor).active = true

focusGuide.preferredFocusedView = button4
```

Focus Update Callbacks on UIFocusEnvironment

Recall that `UIFocusEnvironment` implementors includes `UIView`, `UIViewController`, `UIWindow`, and `UIPresentationController`.

```
func shouldUpdateFocusInContext(_ context: UIFocusUpdateContext) ->
Bool

func didUpdateFocusInContext(_ context: UIFocusUpdateContext,
    withAnimationCoordinator coordinator: UIFocusAnimationCoordinator)
```

Called on all focus environments that contain the previously focused view and the newly focused view.

UIFocusUpdateContext

```swift
weak var previouslyFocusedView: UIView? { get }

weak var nextFocusedView: UIView? { get }

var focusHeading: UIFocusHeading { get }

struct UIFocusHeading : OptionSetType {
    init(rawValue rawValue: UInt)
    static var Up: UIFocusHeading { get }
    static var Down: UIFocusHeading { get }
    static var Left: UIFocusHeading { get }
    static var Right: UIFocusHeading { get }
    static var Next: UIFocusHeading { get }
    static var Previous: UIFocusHeading { get }
}
```

Coordinating Animations With Focus Change

- When focus changes, there are two system-generated animations:
  - The previous view becomes unfocused
  - The new view becomes focused

Depending on the speed of the swipe, the duration of the animations will differ. Generally, unfocusing animations run slower than focusing.

UIFocusAnimationCoordinator

```
override func didUpdateFocusInContext(context: UIFocusUpdateContext,
              withAnimationCoordinator coordinator: UIFocusAnimationCoordinator) {
  super.didUpdateFocusInContext(context, withAnimationCoordinator: coordinator)

  let button4Focused = (context.nextFocusedView == self.button4)

  coordinator.addCoordinatedAnimations({
      button4Title.alpha = button4Focused ? 1 : 0
    },
    completion: nil
  )
}
```

Coordinated animations are run at the same speed as the focus update animations.

To explicitly access the animation duration, call the UIView's class method while in an animation block:

`class func inheritedAnimationDuration() -> NSTimeInterval`

The completion block will be called after the focus update is called.