JOSHUA ALVARADO

CUSTOM VIEWCONTROLLER TRANSITIONS

SUMMARY

- Demo
- UIKit custom transitions
- Implementation
- How to animate
- Architecture

DEMO

Image provided by: https://pixabay.com/photo-458151/

CUSTOM TRANSITIONS

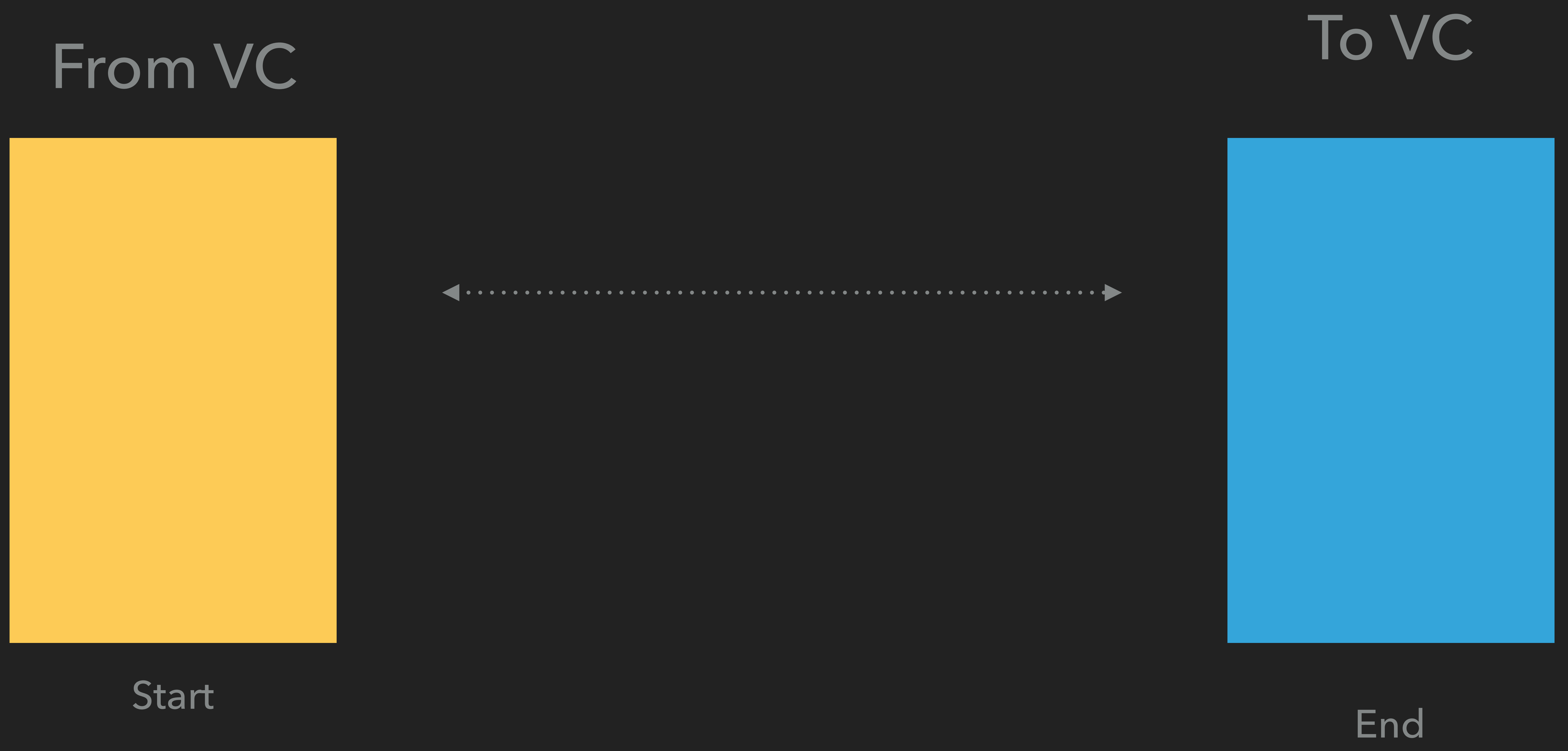
- UIViewControllers are pushed, popped and presented onto the navigation stack in a standard way
- You can provide custom animations during the transition of view controllers
- Custom transitions can be implemented on
    - Presentation, dismissal (modal)
    - UITabBarController
    - UINavigationController (push, pop)
    - UICollectionViewController layouts

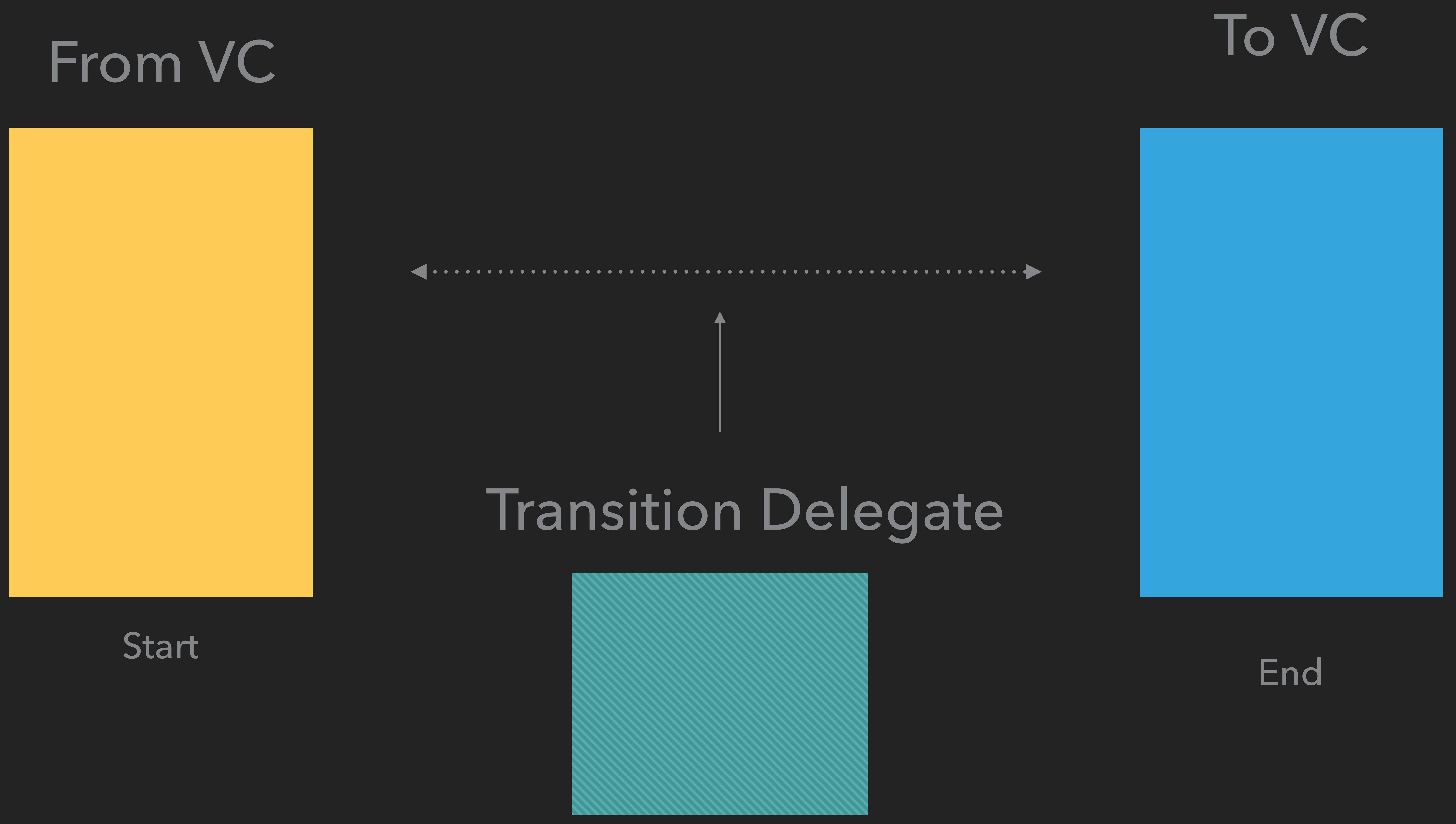
CUSTOM TRANSITIONS

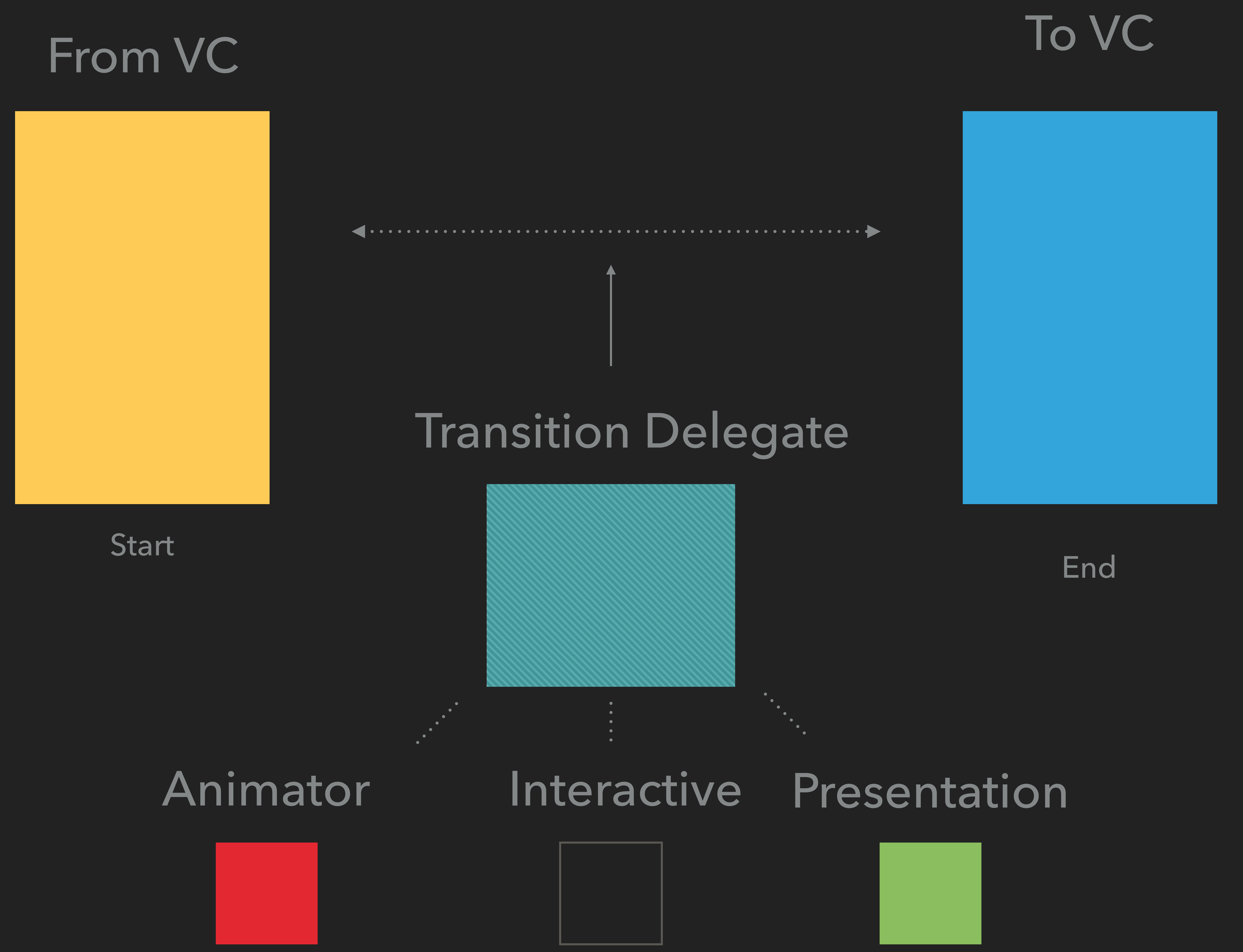
- UIViewControllers are objects iOS developers and users interact with a lot
- Providing interactive navigation to users creates an immersive experience
- Simplify Navigation
- Easy with UIKit
- Why not?

WHERE TO START

- Create and set the transition coordinator (delegate)
- Implement the UIKit protocols for the transition
- Invoke the transition


From VC
To VC
Start
End


From VC
To VC
Start
Transition Delegate
End


From VC
To VC
Start
End
Transition Delegate
Animator
Interactive
Presentation

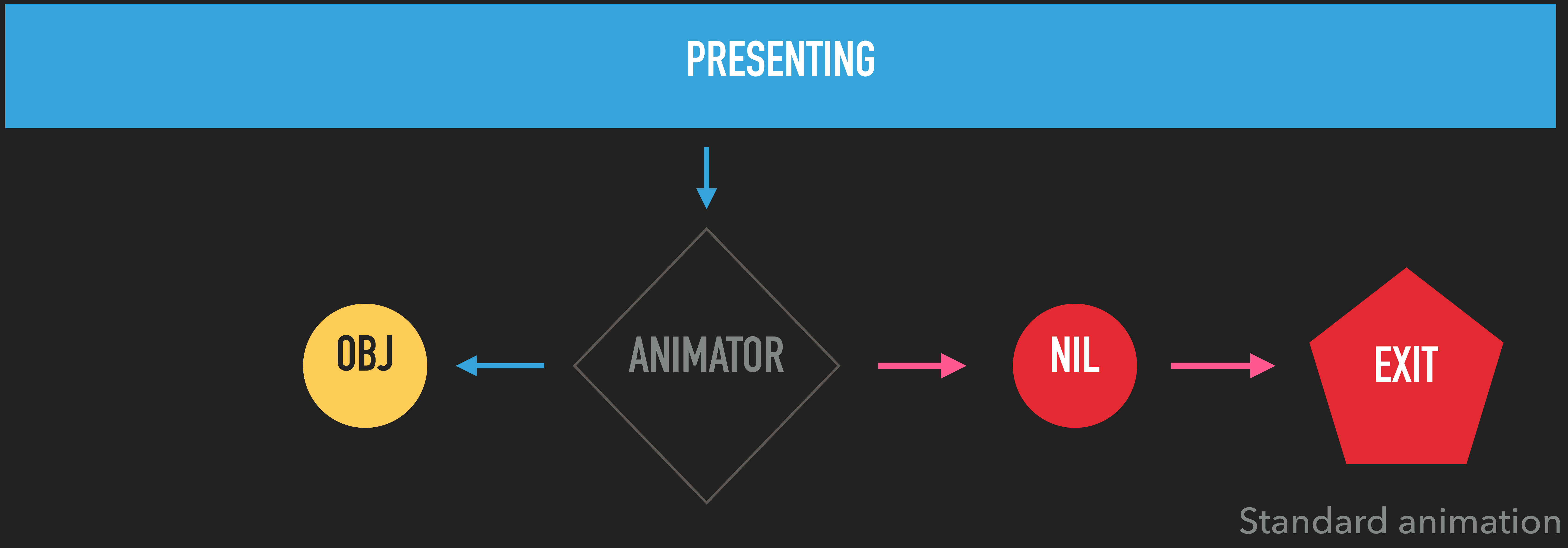
UIViewcontrollerTransitioningDelegate

- Starting point of the animation
- Conforms to UIViewControllerTransitioningDelegate
- Coordinates the animator objects for use in the animations
- Provides UIKit with:
    - Animator to use for the presenting/dismissing view controller
    - Interactive animator to use for the animation
    - Presentation controller (custom presentation style)

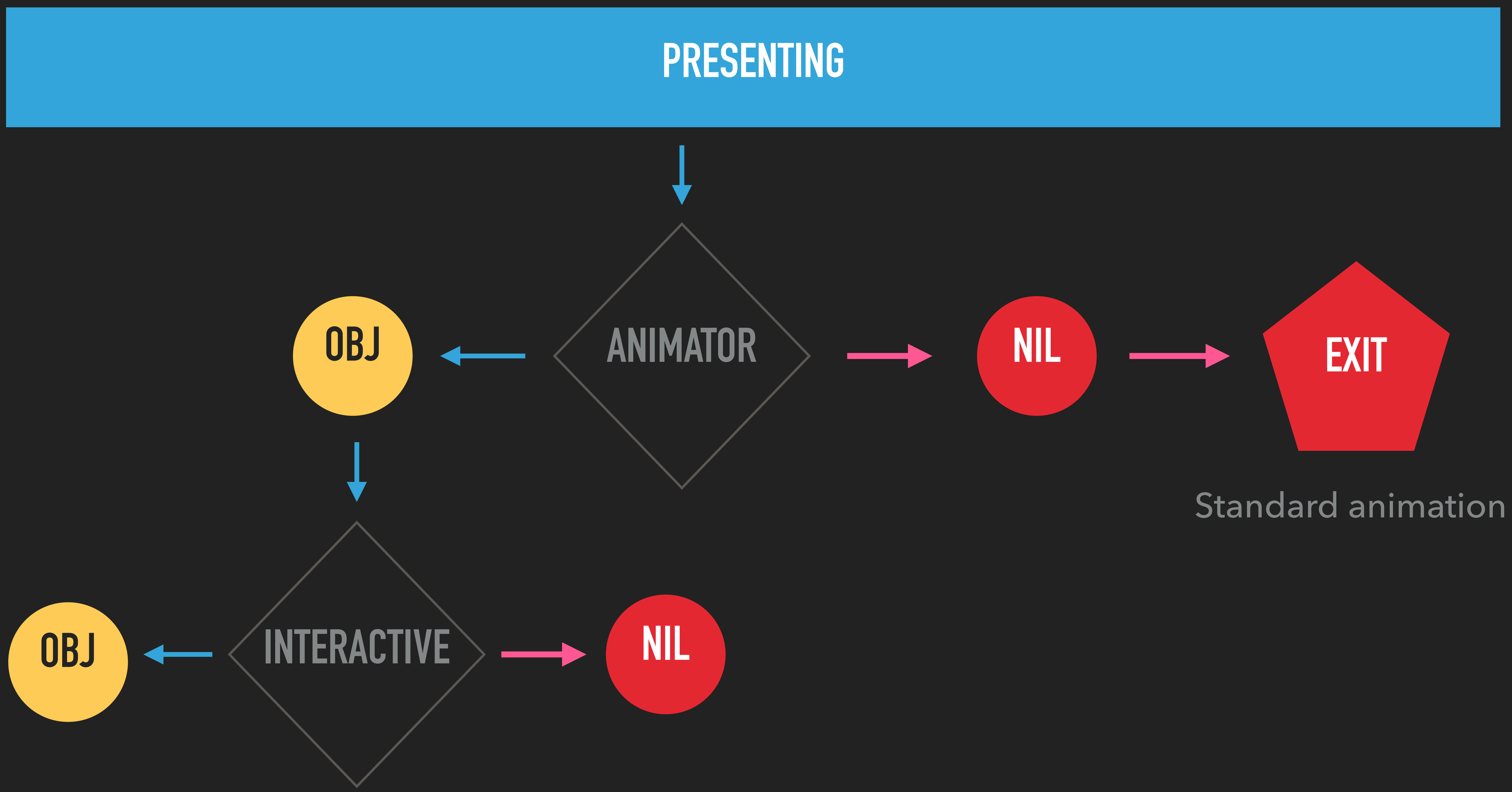
TRANSITIONING DELEGATE

PRESENTING

TRANSITIONING DELEGATE

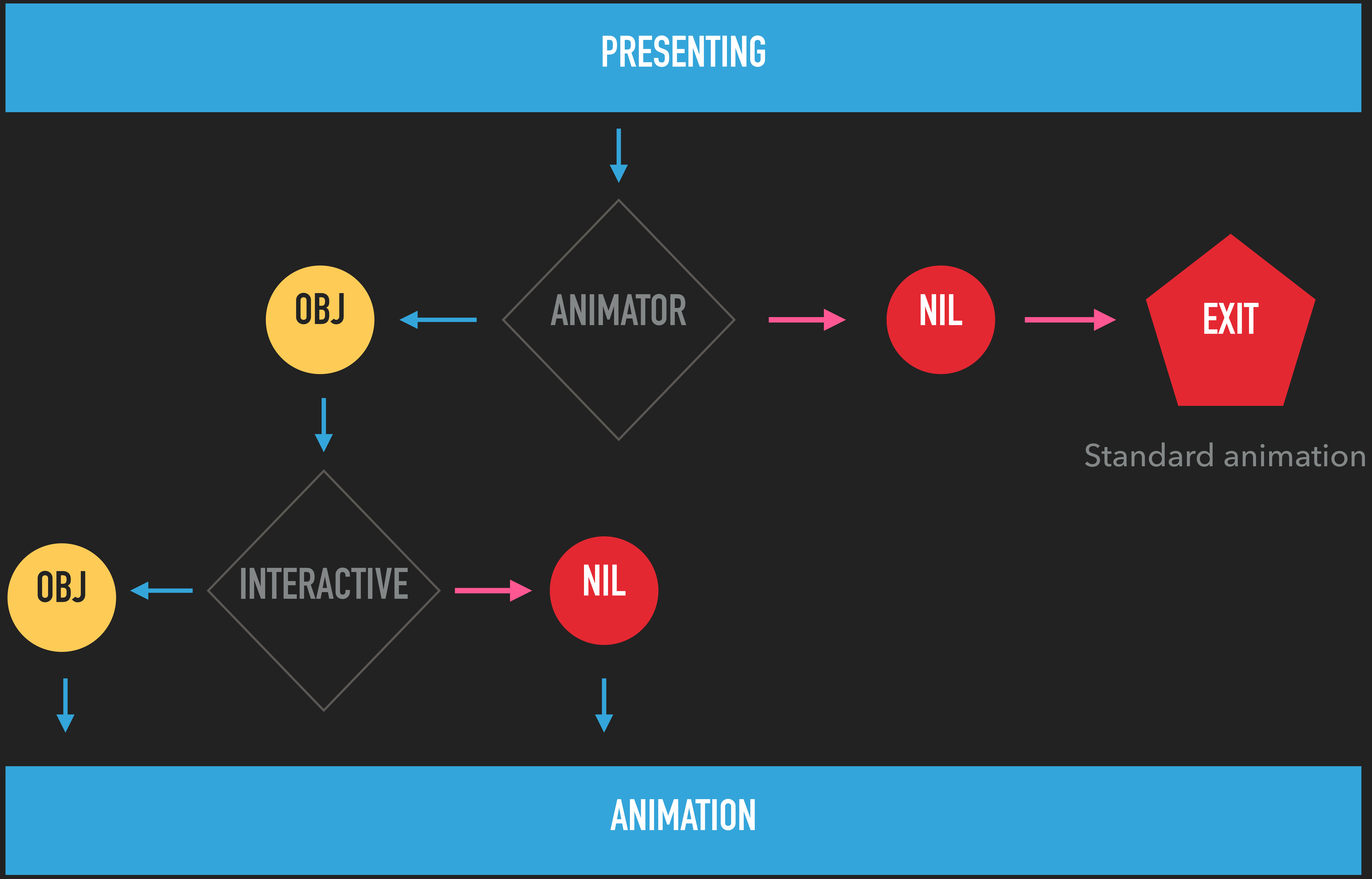


TRANSITIONING DELEGATE



Standard animation

TRANSITIONING DELEGATE



API

UIViewControllerAnimatedTransitioning

- Conforms to UIViewControllerAnimatedTransitioning
- Creates the animations for the view controllers to and from
- Animation is done with UIKit animation APIs (CoreAnimation, Keyframe)
- Is not an interactive animation
- Handles the stack of the views
- Sets the timing and calls the completion of the animation (very important)

UIViewControllerAnimatedTransitioning

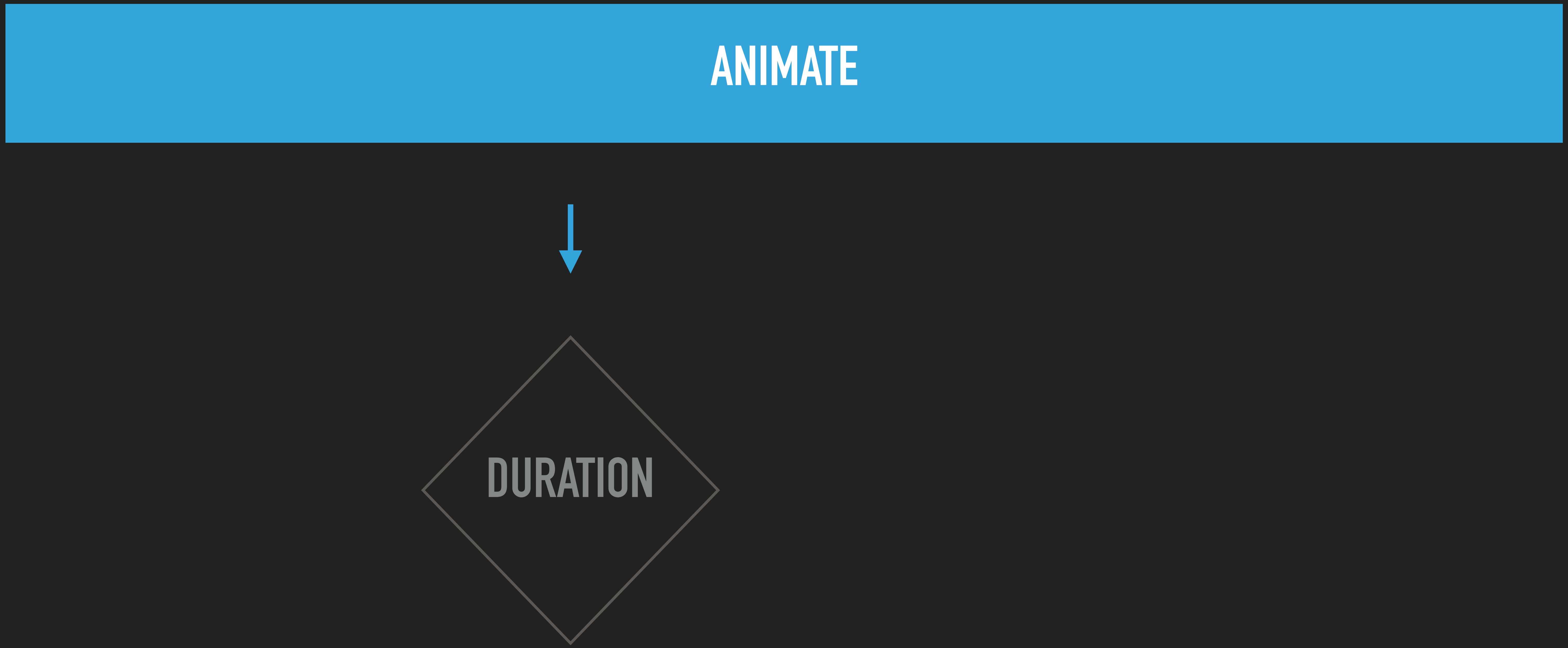
- Methods to implement

```
func transitionDuration(transitionContext: ctx) -> NSTimeInterval

func animateTransition(transitionContext: ctx) -> Void

func animationEnded(transitionCompleted: ctx)
```

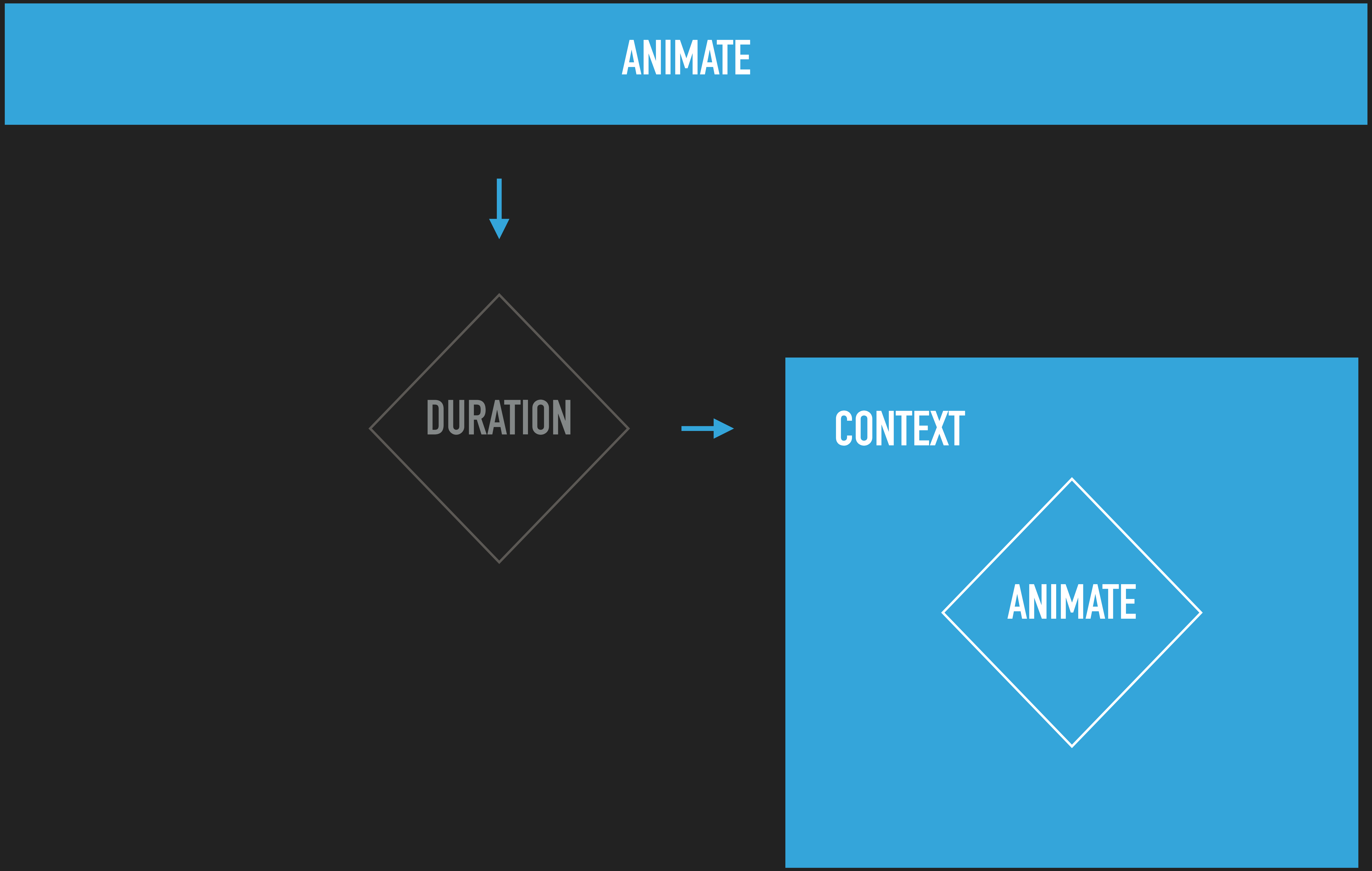
ANIMATOR OBJECT

ANIMATE

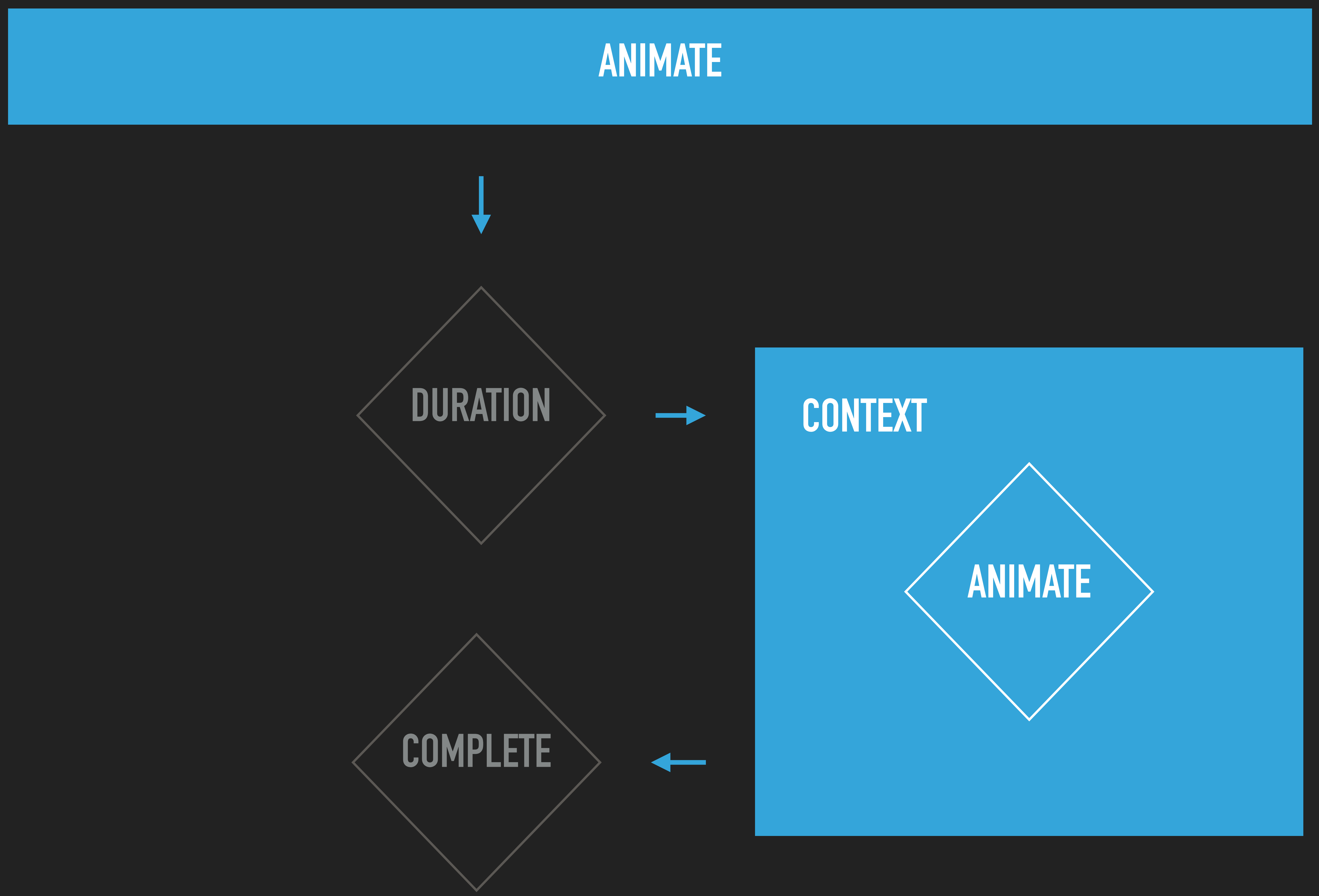
ANIMATOR OBJECT



ANIMATOR OBJECT



ANIMATOR OBJECT



API

UIViewControllerContextTransitioning

- Provides context information for the transition, passed to the animator from UIKit
- Provides the container view where all the animation takes place in

```
viewControllerForKey(key: String) -> UIViewController?

viewForKey(key: String) -> UIView?

UITransitionContextFromViewControllerKey
```

PSEUDO CODE

```
// view controller presenting

func presentVC() {

    var  vcToPresent = // get vc to present

    vcToPresent.transitionDelegate = customTransitionDelegate

    presentViewController(vcToPresent, animated: true, completion: nil)

 }
```

PSEUDO CODE

```
func animateTransition(ctx: UIViewControllerContextTransitioning) {
    let container = ctx.containerView()
    let toView = ctx.viewForKey(toViewKey)
    let fromView = ctx.viewForKey(fromViewKey)

    if presenting {
      container.addSubview(toView)
      // animation when presenting
    } else {
      container.insertSubview(toView, below: fromView)
      // animation when dismissing
    }
}
```

API

Interactive View Controller Transitions

- Easy to implement with UIPercentDrivenInteractiveTransition
- Uses percentage value to drive
- Done “magically” using the animator animation
- Runs forward and in reverse
- Provides cancelling
- Can be based on gestures or not

UIPercentDrivenInteractiveTransition

- Interactive animation is asked for only if animator object is provided
- interactive transition is based on a range from 0.0 - 1.0

```
func updateInteractiveTransition(percentComplete: CGFloat)

func cancelInteractiveTransition()

func finishInteractiveTransition()
```

PSEUDO CODE

```
func handleGesture(gesture: UIPinchGestureRecognizer) {
 switch gesture.state {
 case .Began:
      startScale = gesture.scale
 case .Changed:
      let progress = 1.0 - (gesture.scale / startScale)
      updateInteractiveTransition(progress < 0.0 ? 0.0 : progress)
 case .Cancelled:
      cancelInteractiveTransition()
 case .Ended:
      if gesture.velocty <= 0.0 {
         finishInteractiveTransition()
      }
   }
}
```

Notes

- UITabBarController and UINavigationController's delegate provides methods to set the UIViewControllerAnimatedTransitioning for the view controllers.

```
func tabBarController(tabBarController: UITabBarController, fromVC: UIViewController,
toVC: UIViewController) -> UIViewControllerAnimatedTransitioning? {

        // return the animator for the tab bar controller

        return ViewControllerAnimatedTransitioning()
    }
```

Review

- Custom transitions are achieved by being the “middleware” of the animation context
- Implement the protocols that UIKit uses to handle view controller transitions
- Use Core Animation or Keyframes for the animation
- Handle the navigation stack correctly and complete the animation
- Add UIPercentDrivenInteractiveTransition for interactive transitions

ADDITIONAL READING

- WWDC 2013 Custom Transitions using View Controllers
- Apple iOS Programming Guide Customizing Transitions
- objc.io view controller transitions
- Raywenderlich custom uiviewcontroller transitions

CONTACT

@ALVARADOJOSHUA0

LOSTATSEAJOSHUA

ALVARADOJOSHUA0@GMAIL.COM

Icons provided by https://icons8.com