App Frameworks

#WWDC17

# Advanced Animations with UIKit

Session 230

Joe Cerra, UIKit Engineer

Basics

Interactive and Interruptible Animations

New Property Animator Behaviors

Coordinating Animations

Tips and Tricks

Basics

Interactive and Interruptible Animations

New Property Animator Behaviors

Coordinating Animations

Tips and Tricks

Basics

Interactive and Interruptible Animations

New Property Animator Behaviors

Coordinating Animations

Tips and Tricks

Basics

Interactive and Interruptible Animations

New Property Animator Behaviors

Coordinating Animations

Tips and Tricks

Basics

Interactive and Interruptible Animations

New Property Animator Behaviors

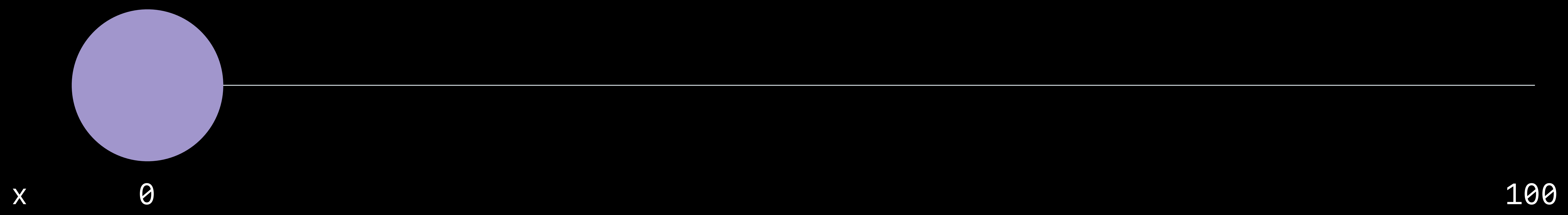
Coordinating Animations

Tips and Tricks

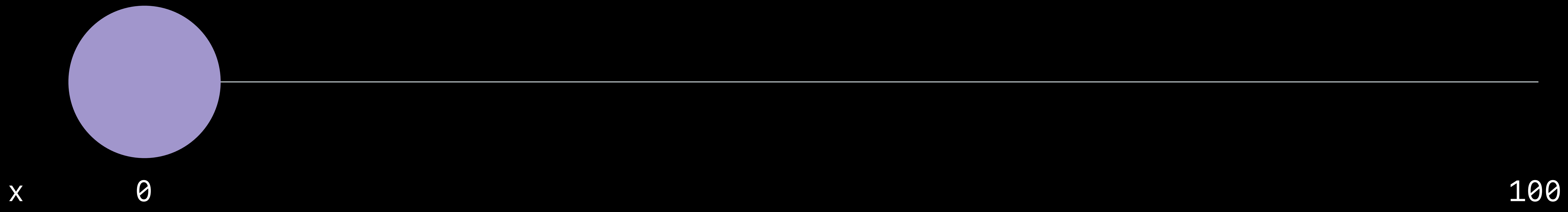
## Basics

## UIView-based Animations

## UIView-based Animations

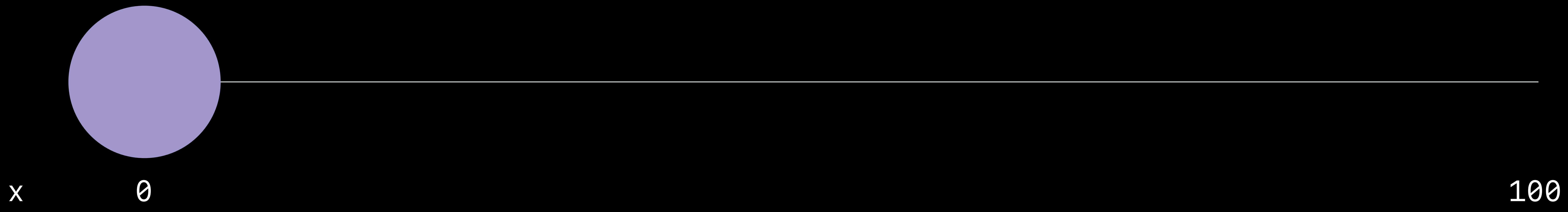


## UIView-based Animations



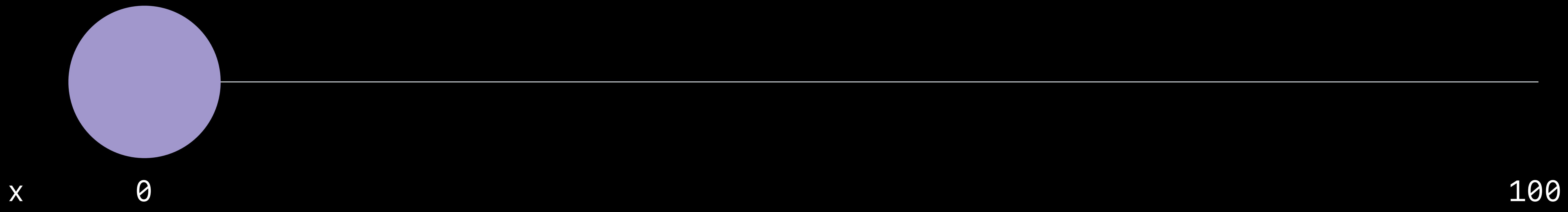
```
UIView.animate(withDuration: 5) {
    circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
}, completion: nil)
```

# UIView-based Animations



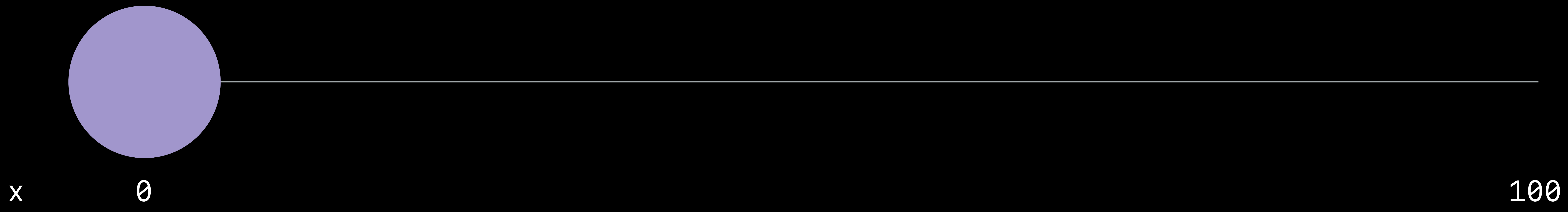
```
UIView.animate(withDuration: 5) {
    circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
}, completion: nil)
```

## UIView-based Animations



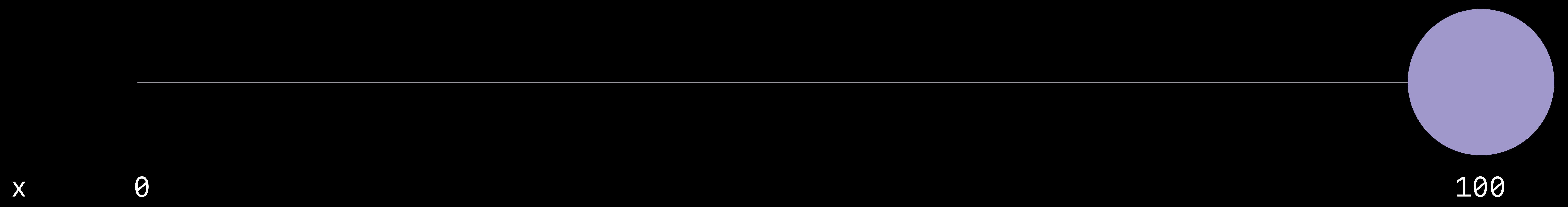
```
UIView.animate(withDuration: 5) {
    circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
}, completion: nil)
```

# UIView-based Animations



```
UIView.animate(withDuration: 5) {
    circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
}, completion: nil)
```

# UIView-based Animations



```
UIView.animate(withDuration: 5) {
    circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
}, completion: nil)
```

## UIViewPropertyAnimator

## UIViewPropertyAnimator

Features

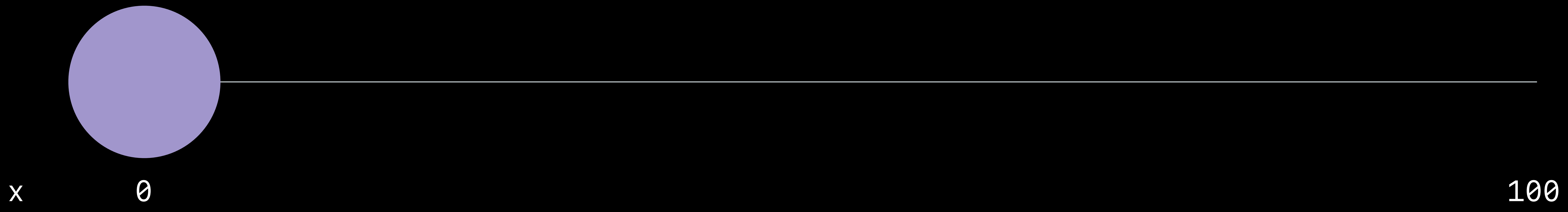
Custom timing

Interactive

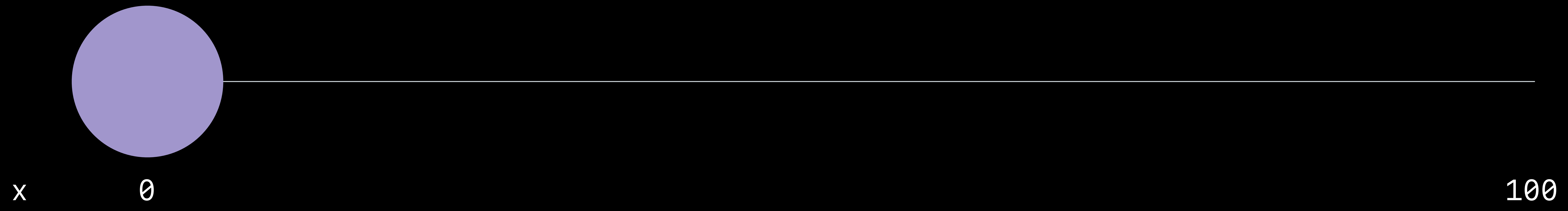
Interruptible

Responsive

## UIViewPropertyAnimator

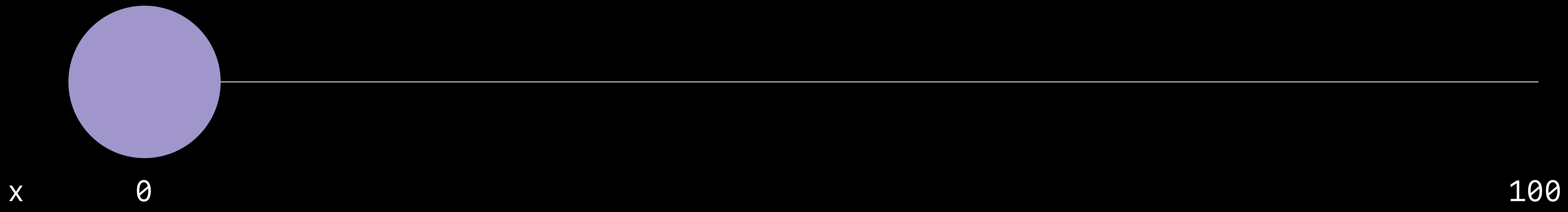


# UIViewPropertyAnimator



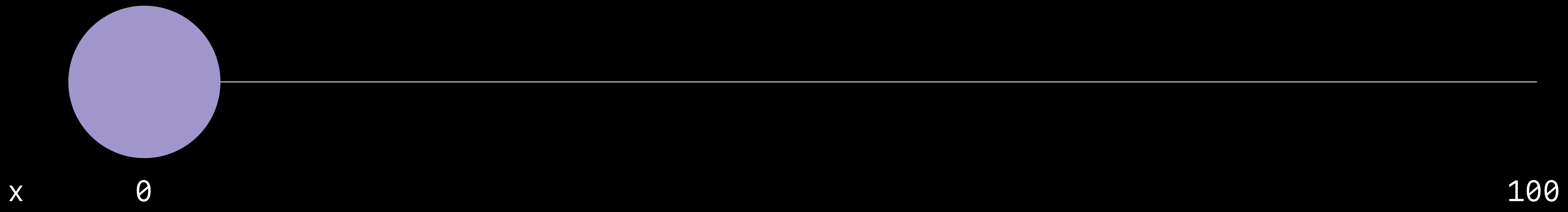
```
let animator = UIViewPropertyAnimator(duration: 2.5, curve: .linear) {
    circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
}
animator.startAnimation()
```

# UIViewPropertyAnimator



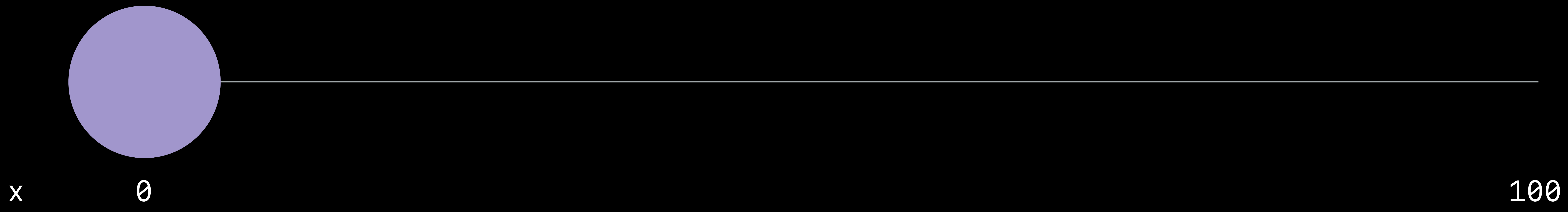
```
let animator = UIViewPropertyAnimator(duration: 2.5, curve: .linear) {
    circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
}
animator.startAnimation()
```

# UIViewPropertyAnimator



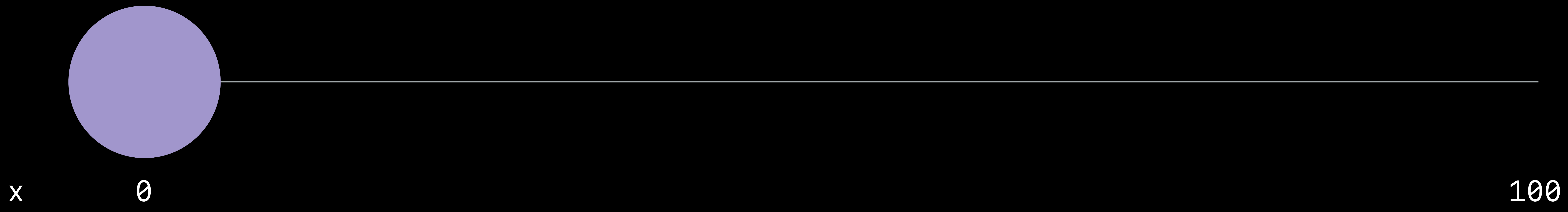
```
let animator = UIViewPropertyAnimator(duration: 2.5, curve: .linear) {
    circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
}
animator.startAnimation()
```

## UIViewPropertyAnimator



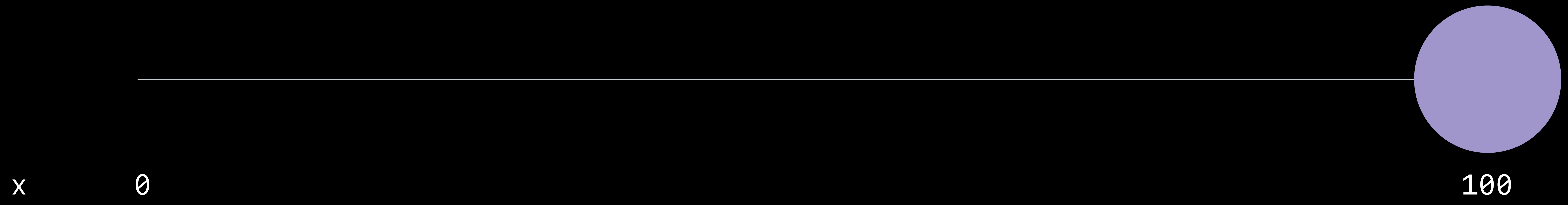
```
let animator = UIViewPropertyAnimator(duration: 2.5, curve: .linear) {
    circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
}
animator.startAnimation()
```

## UIViewPropertyAnimator



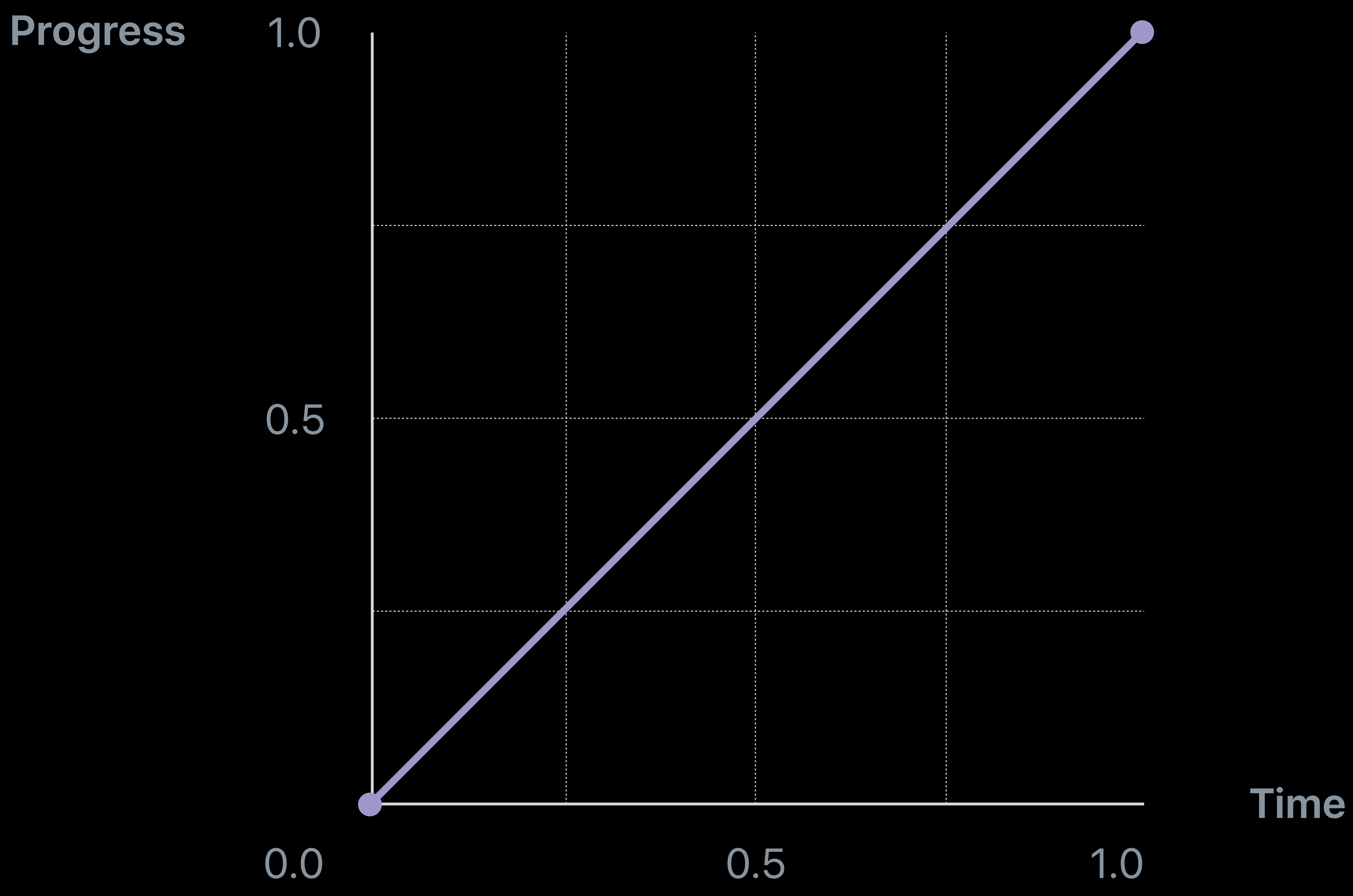
```
let animator = UIViewPropertyAnimator(duration: 2.5, curve: .linear) {
    circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
}
animator.startAnimation()
```

# UIViewPropertyAnimator



```
let animator = UIViewPropertyAnimator(duration: 2.5, curve: .linear) {
    circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
}
animator.startAnimation()
```

## Timing Curves



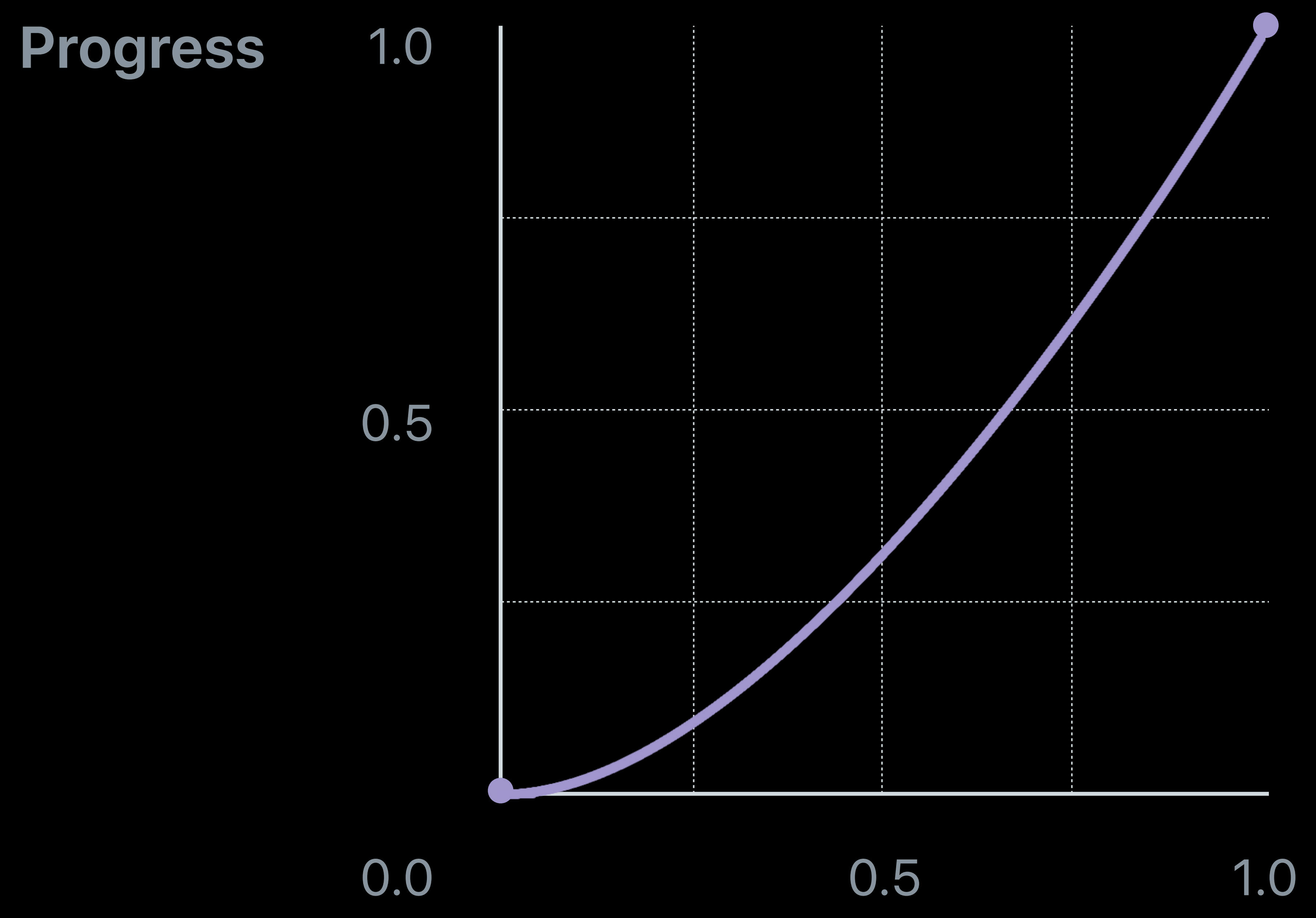
.linear

## Linear Curves

% Progress = % Time

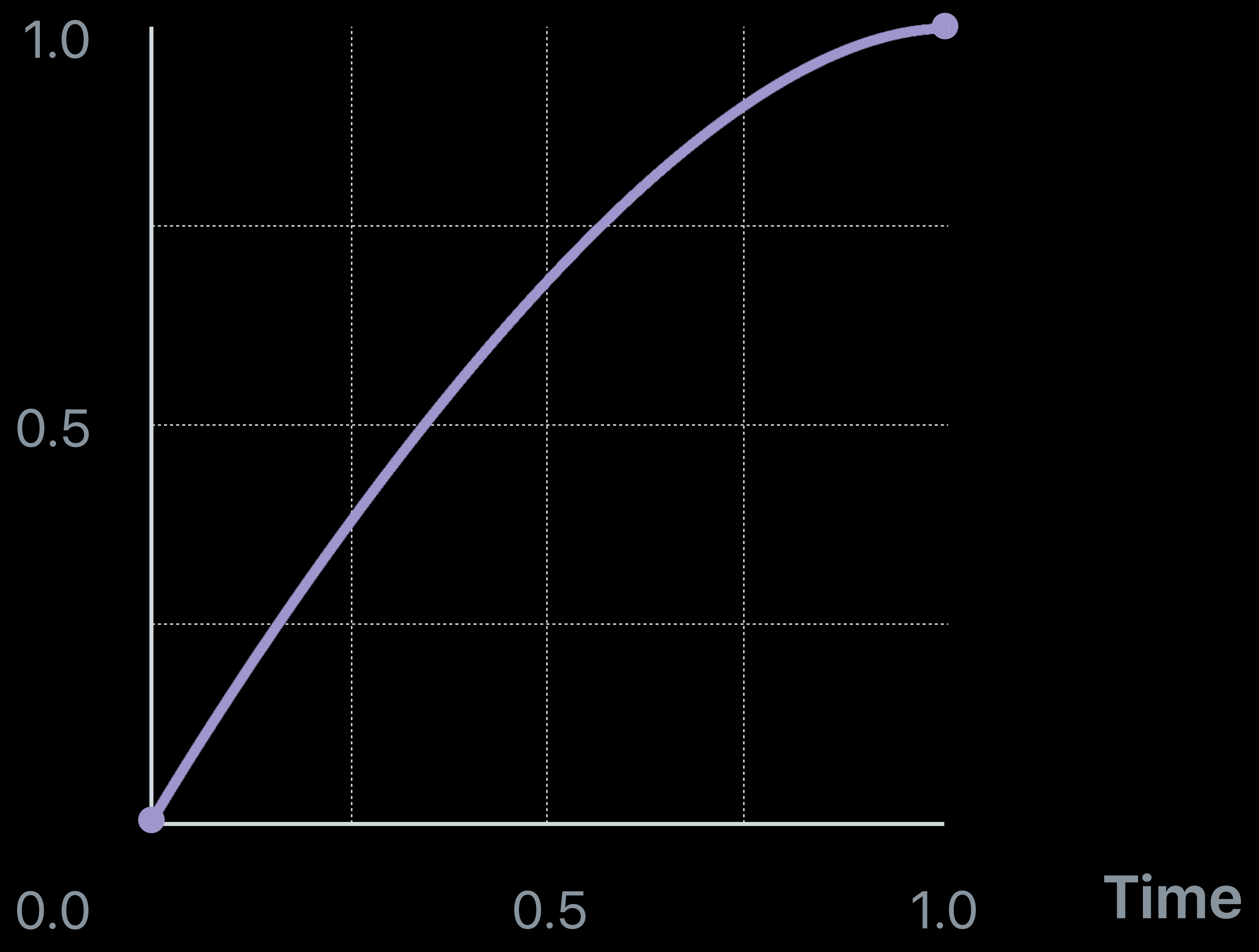
# Timing Curves

## Ease In



.easeIn

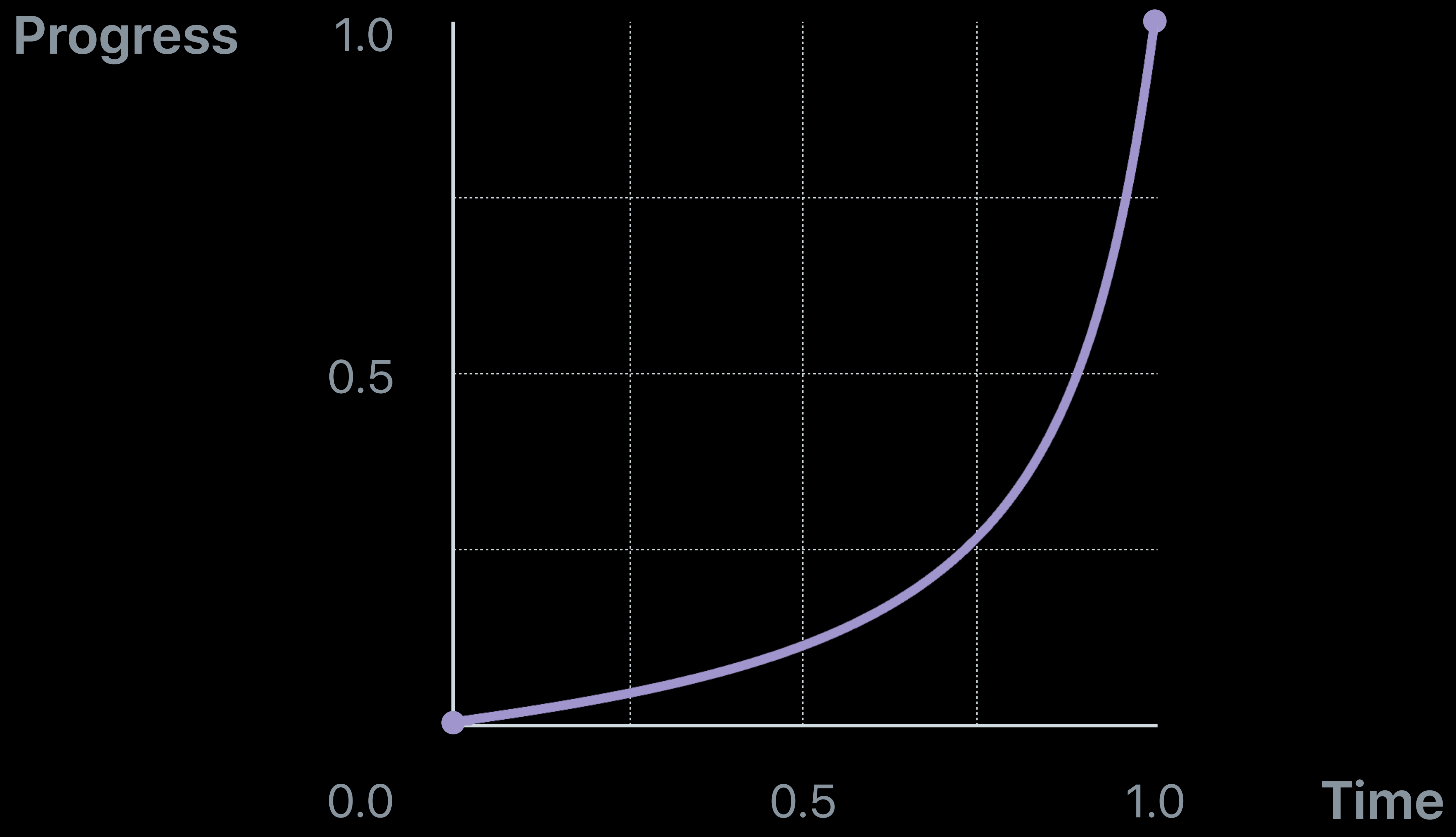
## Ease Out



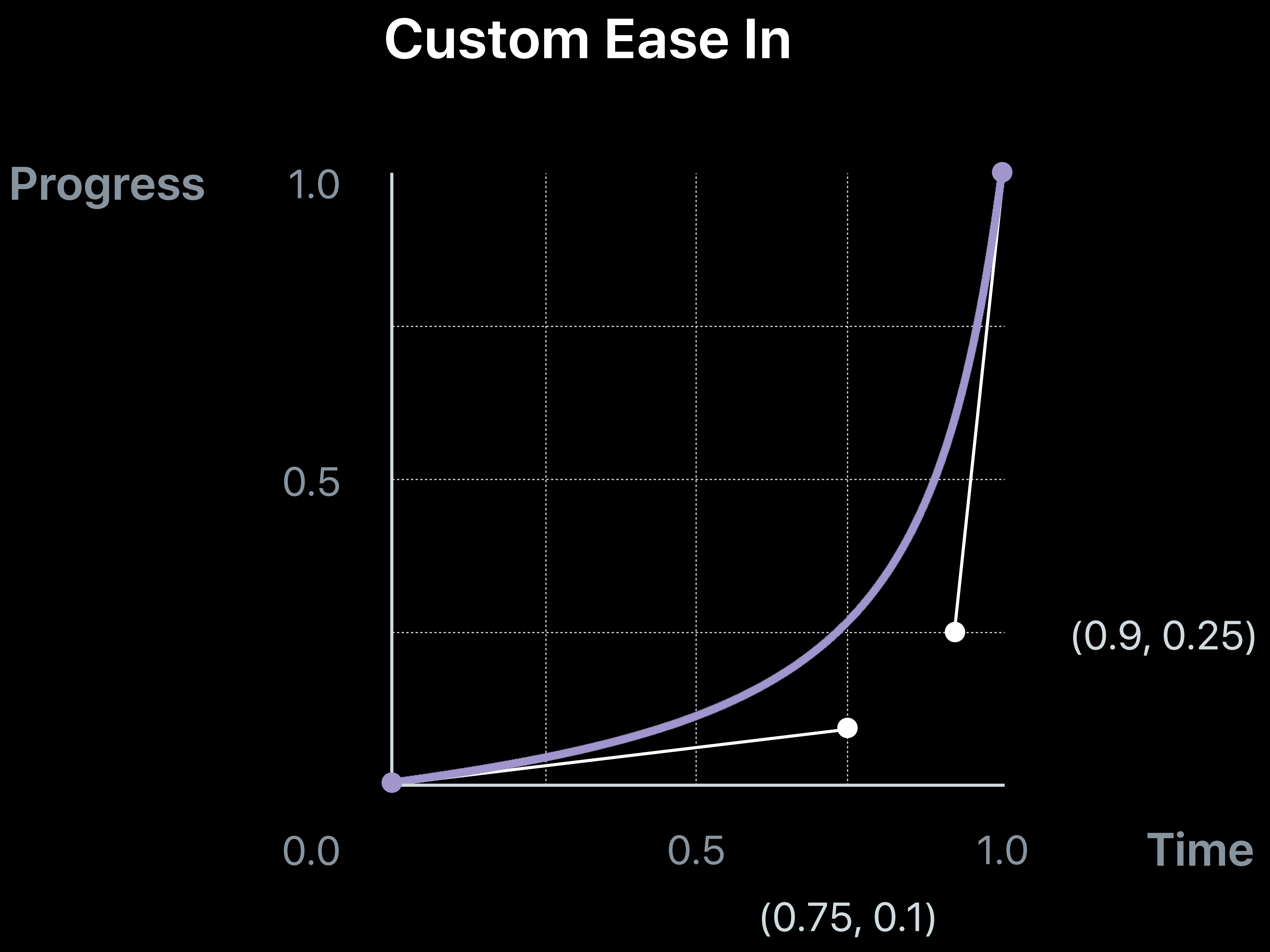
.easeOut

# Custom Curves

Custom Ease In



# Custom Curves



```
UICubicTimingParameters(controlPoint1: CGPoint(x: 0.75, y: 0.1),
                        controlPoint2: CGPoint(x: 0.9, y: 0.25))
```

## Interactively Animating

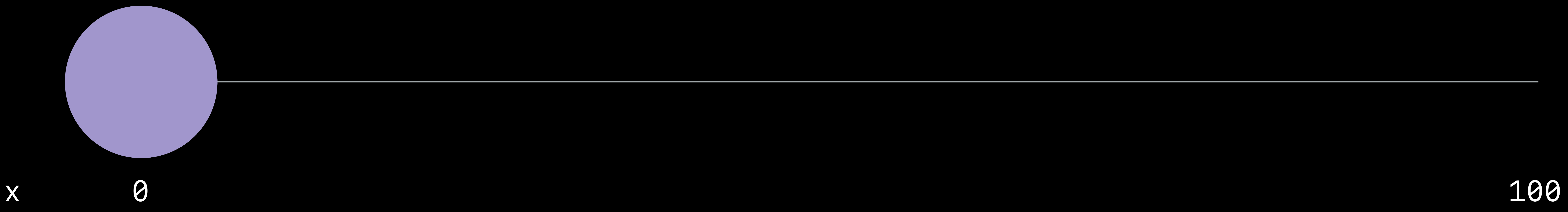
LTE
9:41 AM
9:41
Monday, June 5
Press home to unlock

LTE
9:41 AM
9:41
Monday, June 5
Press home to unlock

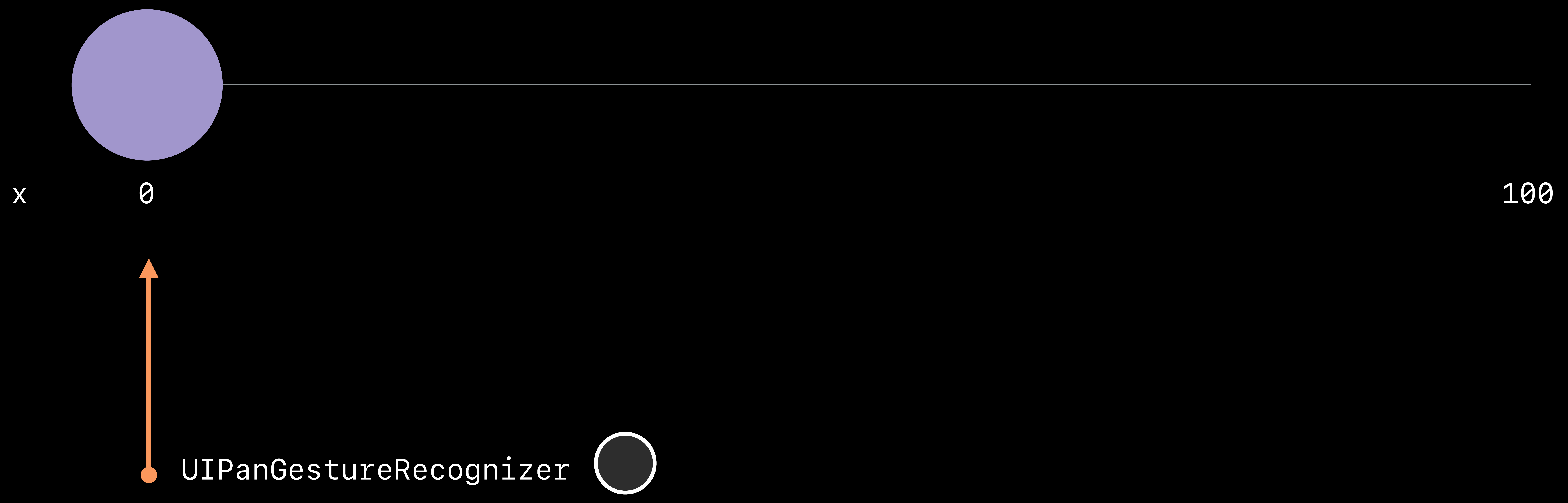
LTE
9:41 AM
9:41
Monday, June 5
Press home to unlock
Confidential & Proprietary, Call +1 877-595-1125

LTE
9:41 AM
9:41
Monday, June 5
Press home to unlock

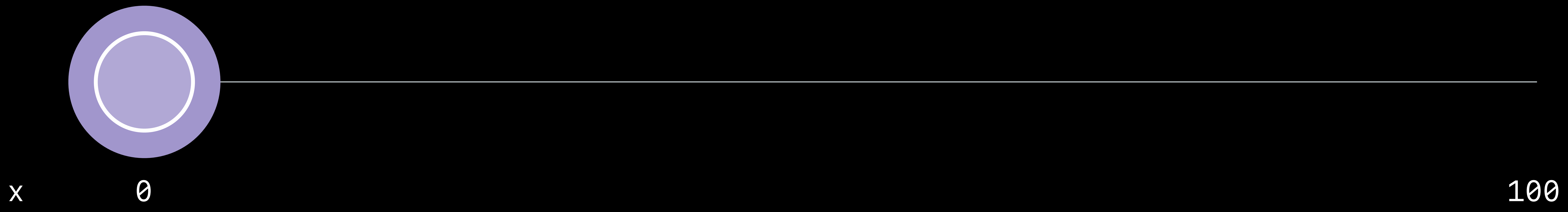
## Interactively animating



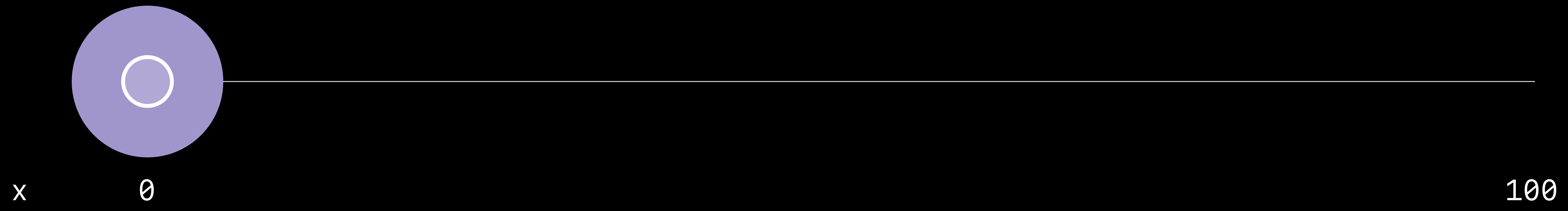
# Interactively animating



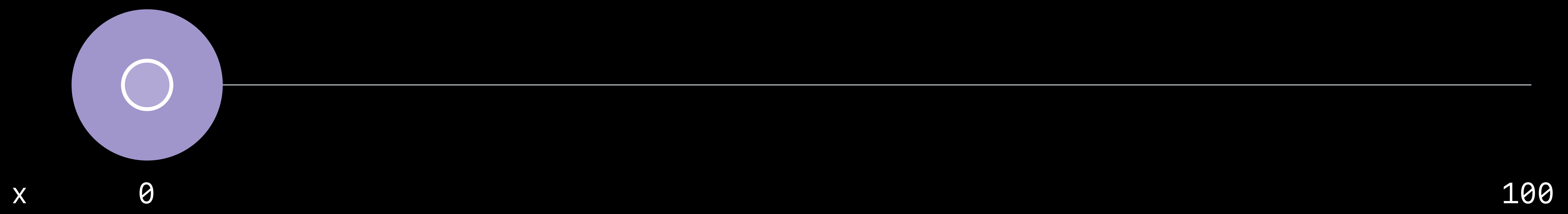
# Interactively animating



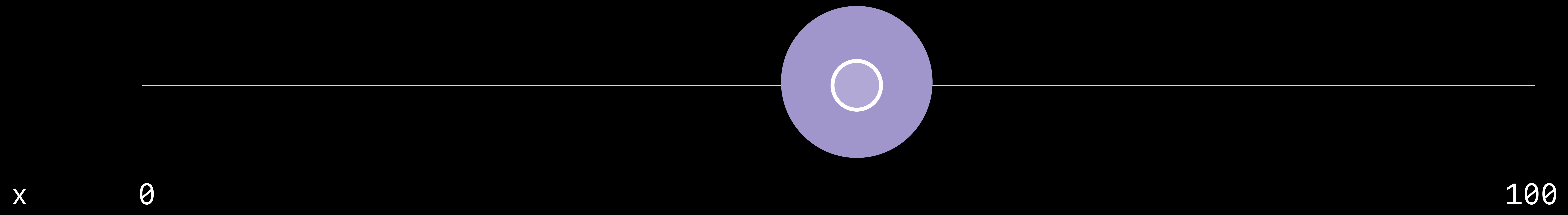
# Interactively animating



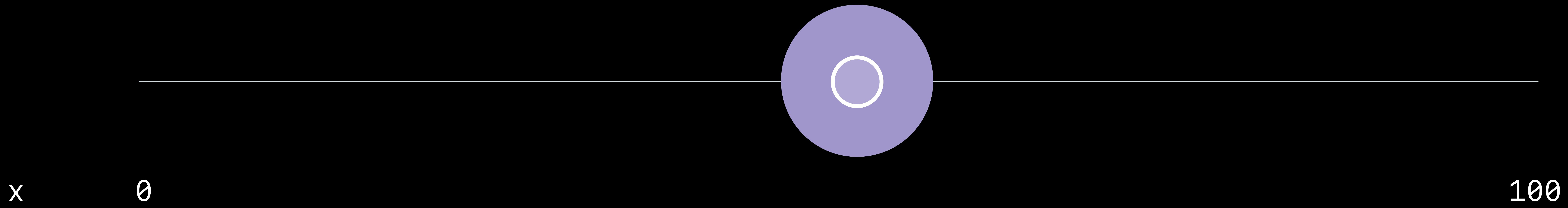
# Interactively animating



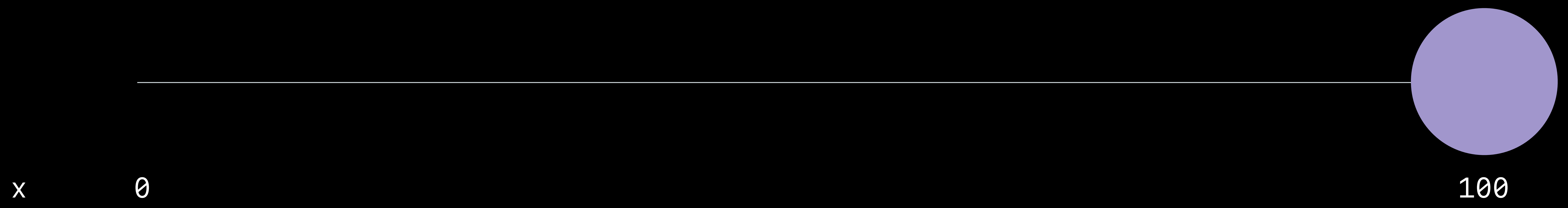
# Interactively animating



## Interactively animating



# Interactively animating



```swift
var animator: UIViewPropertyAnimator!

func handlePan(recognizer: UIPanGestureRecognizer) {
    switch recognizer.state {
    case .began:
        animator = UIViewPropertyAnimator(duration: 1, curve: .easeOut, animations: {
            circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
        })
        animator.pauseAnimation()
    case .changed:
        let translation = recognizer.translation(in: circle)
        animator.fractionComplete = translation.x / 100
    case .ended:
        animator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
    }
}
```

```swift
var animator: UIViewPropertyAnimator!
```

```swift
func handlePan(recognizer: UIPanGestureRecognizer) {
    switch recognizer.state {
    case .began:
        animator = UIViewPropertyAnimator(duration: 1, curve: .easeOut, animations: {
            circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
        })
        animator.pauseAnimation()
    case .changed:
        let translation = recognizer.translation(in: circle)
        animator.fractionComplete = translation.x / 100
    case .ended:
        animator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
    }
}
```

```
var animator: UIViewPropertyAnimator!

func handlePan(recognizer: UIPanGestureRecognizer) {
    switch recognizer.state {
    case .began:
        animator = UIViewPropertyAnimator(duration: 1, curve: .easeOut, animations: {
            circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
        })
        animator.pauseAnimation()
    case .changed:
        let translation = recognizer.translation(in: circle)
        animator.fractionComplete = translation.x / 100
    case .ended:
        animator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
    }
}
```

```swift
var animator: UIViewPropertyAnimator!

func handlePan(recognizer: UIPanGestureRecognizer) {
    switch recognizer.state {
    case .began:
        animator = UIViewPropertyAnimator(duration: 1, curve: .easeOut, animations: {
            circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
        })
        animator.pauseAnimation()
    case .changed:
        let translation = recognizer.translation(in: circle)
        animator.fractionComplete = translation.x / 100
    case .ended:
        animator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
    }
}
```

```swift
var animator: UIViewPropertyAnimator!

func handlePan(recognizer: UIPanGestureRecognizer) {
    switch recognizer.state {
    case .began:
        animator = UIViewPropertyAnimator(duration: 1, curve: .easeOut, animations: {
            circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
        })
        animator.pauseAnimation()
    case .changed:
        let translation = recognizer.translation(in: circle)
        animator.fractionComplete = translation.x / 100
    case .ended:
        animator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
    }
}
```

```swift
var animator: UIViewPropertyAnimator!

func handlePan(recognizer: UIPanGestureRecognizer) {
    switch recognizer.state {
    case .began:
        animator = UIViewPropertyAnimator(duration: 1, curve: .easeOut, animations: {
            circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
        })
        animator.pauseAnimation()
    case .changed:
        let translation = recognizer.translation(in: circle)
        animator.fractionComplete = translation.x / 100
    case .ended:
        animator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
    }
}
```

## Time Conversion

Pausing

Continuing

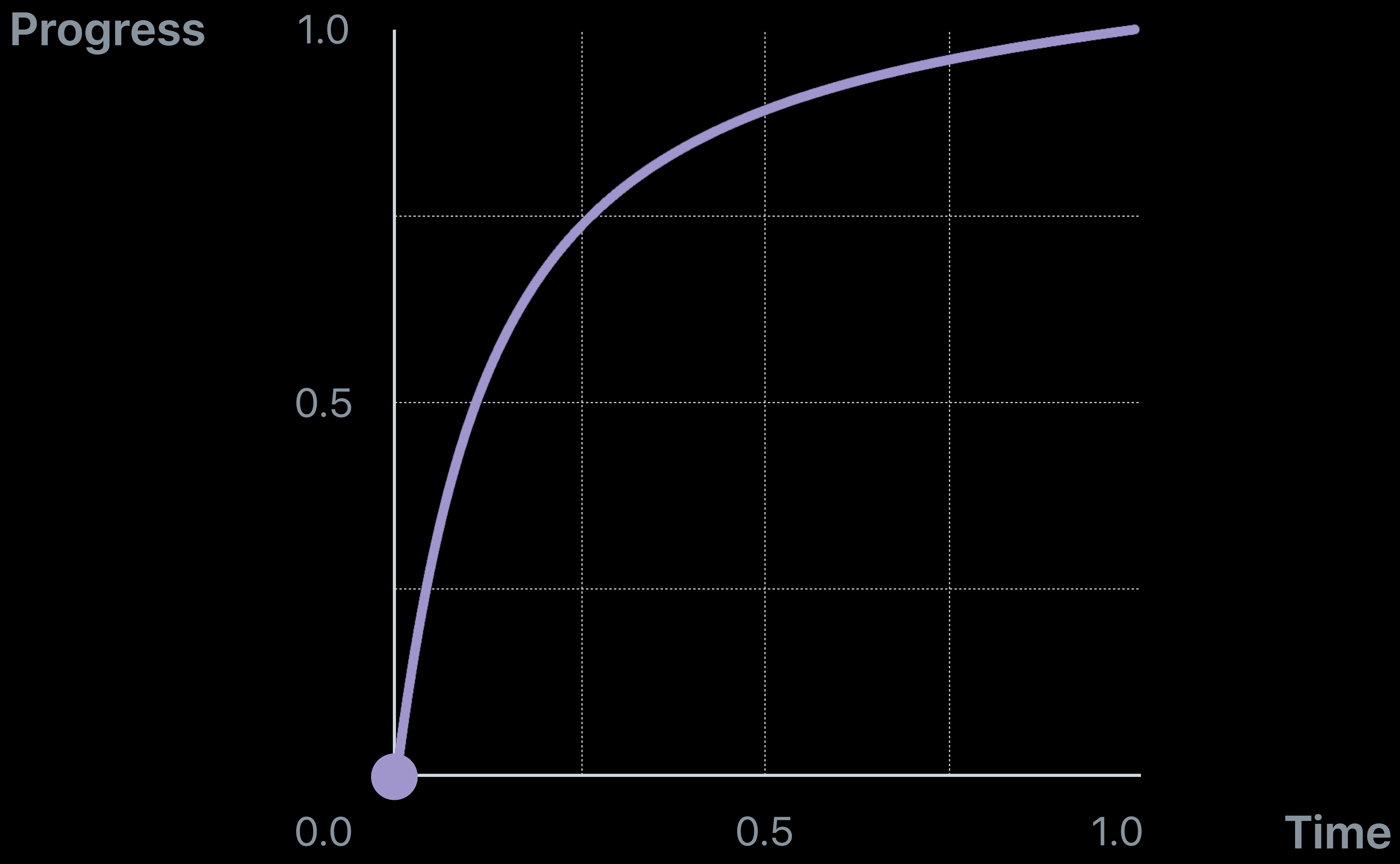
## Time Conversion

Pausing

Continuing

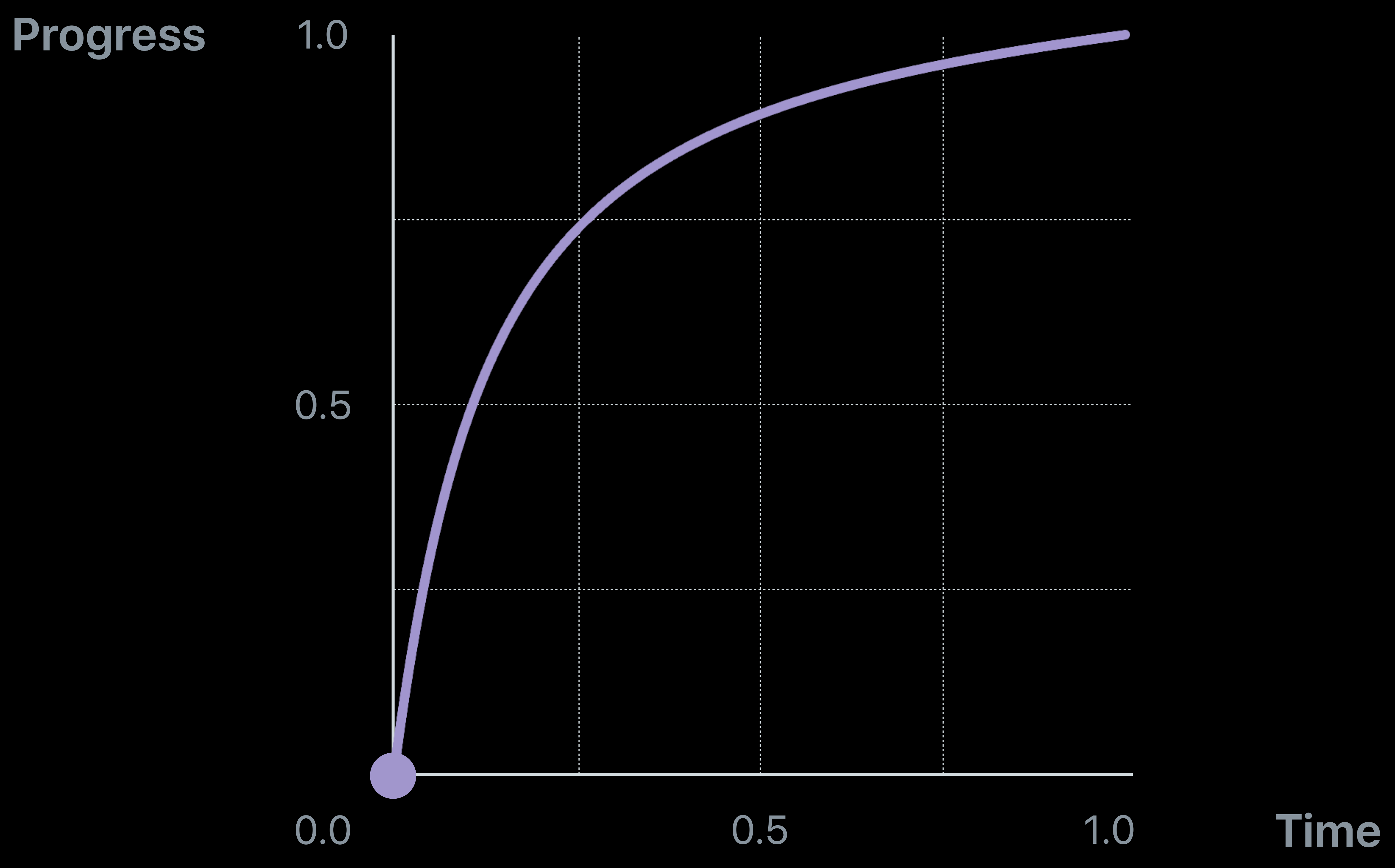
```
UIViewPropertyAnimator(duration: 1, curve: .easeOut)
```

| animationState | running | fractionComplete |
|---|---|---|
| .inactive | false | 0% |



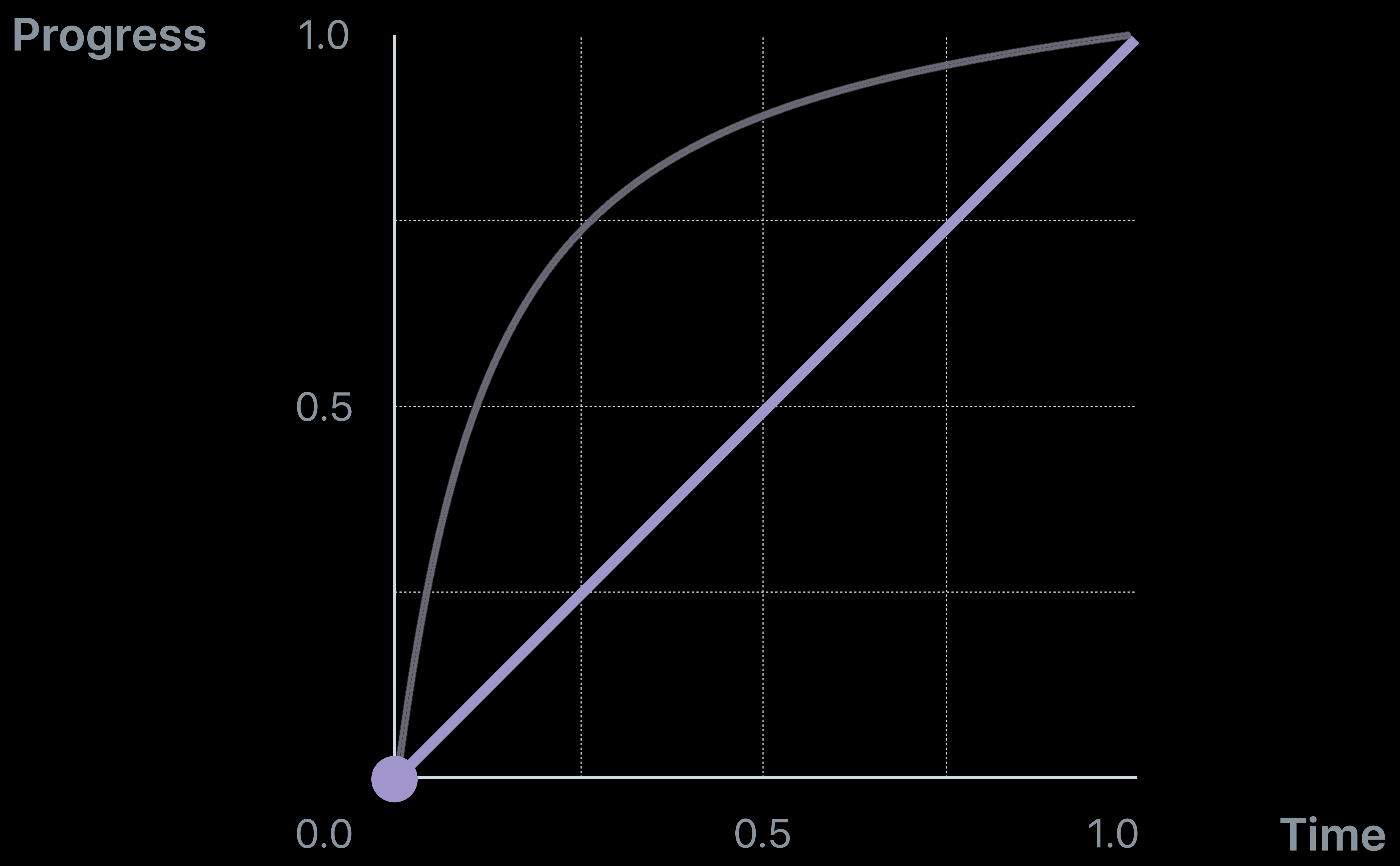
```
animator.pauseAnimation()
```

| animationState | running | fractionComplete |
|---|---|---|
| .active | false | 0% |



```
animator.pauseAnimation()
```

| animationState | running | fractionComplete |
|---|---|---|
| .active | false | 0% |

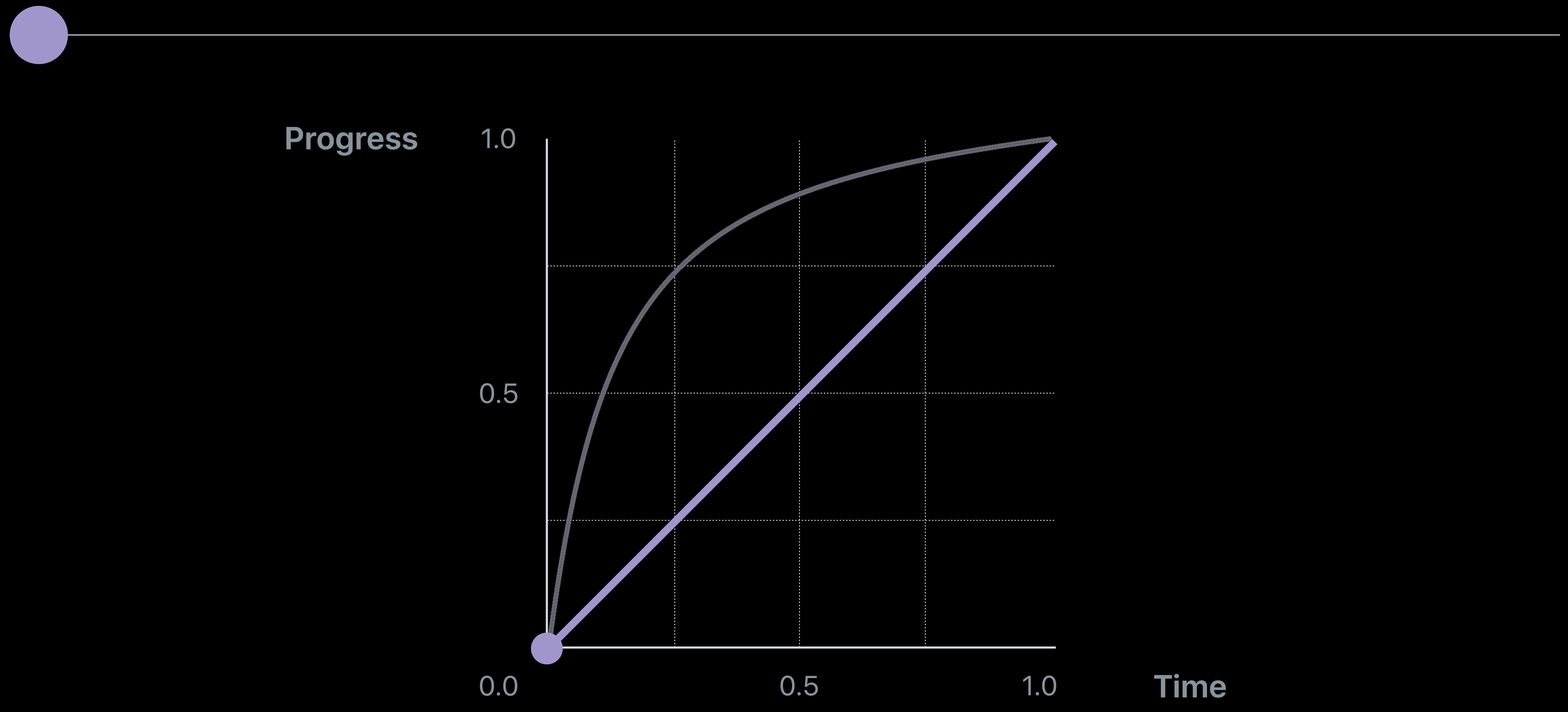


## Time Conversion

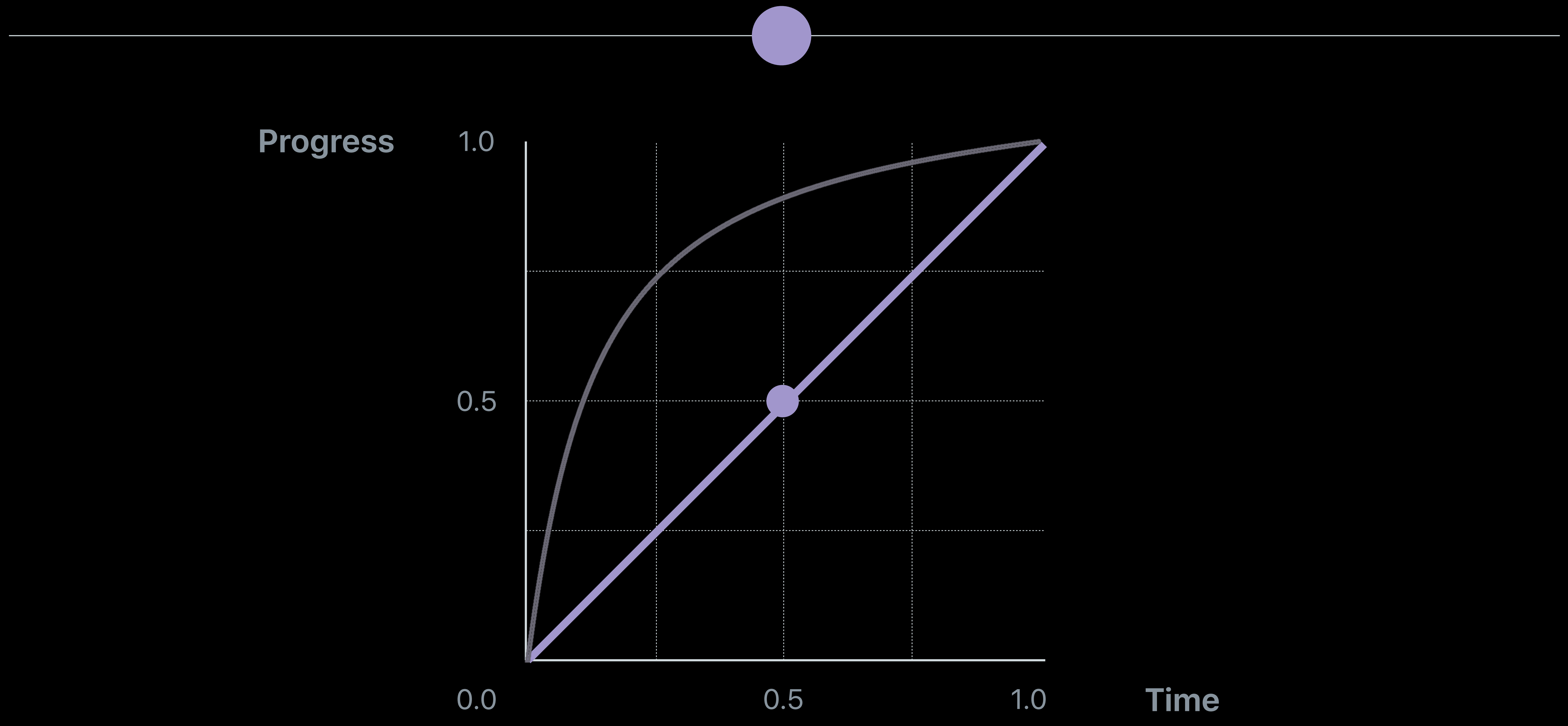
Pausing

Continuing

```
animator.fractionComplete = translation.x / distance
```

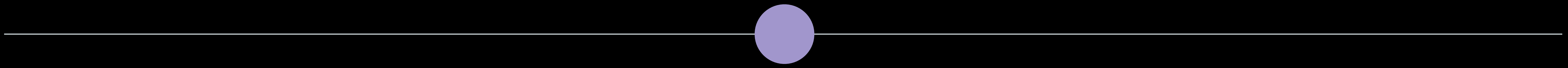


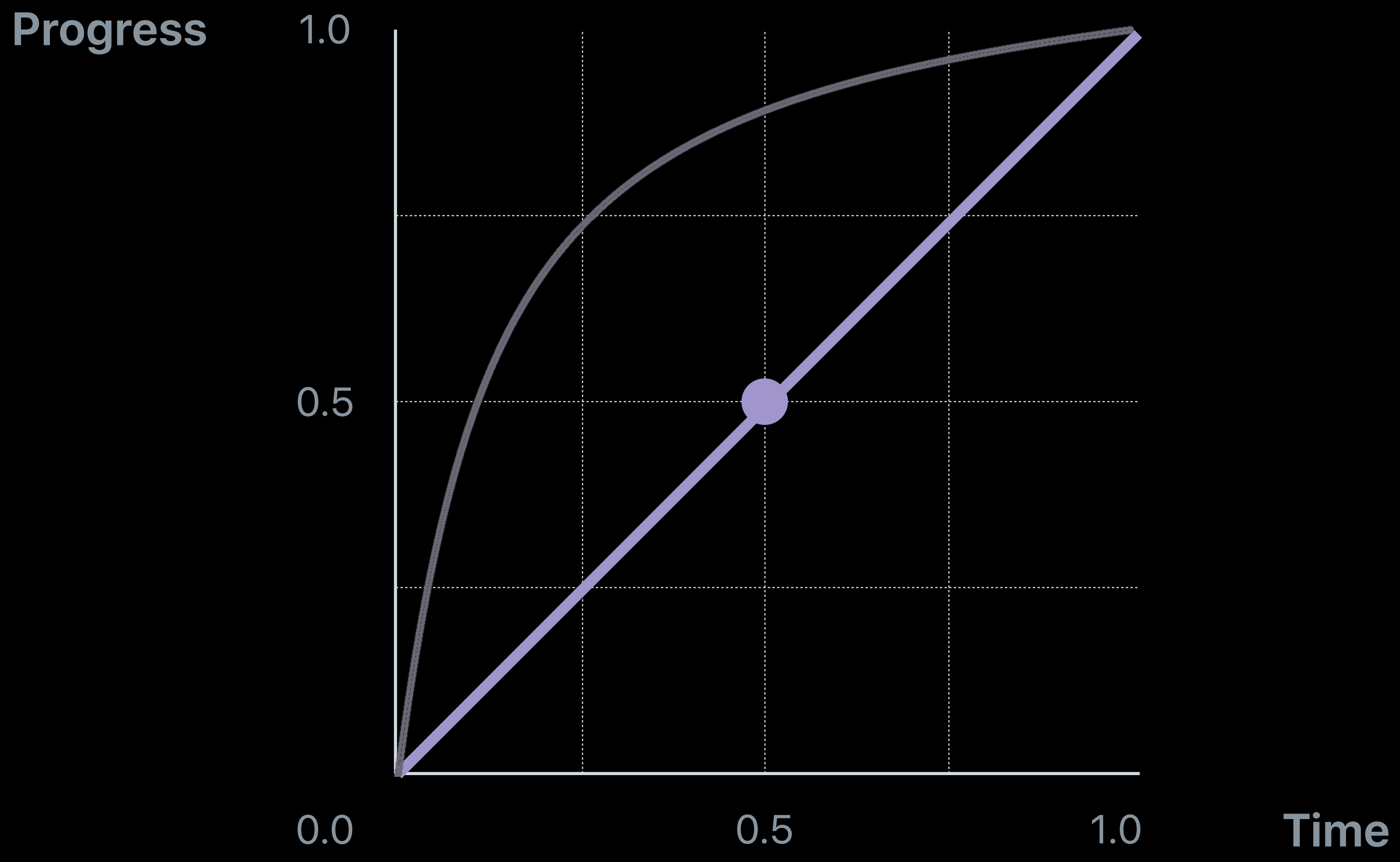
```
animator.fractionComplete = translation.x / distance
```



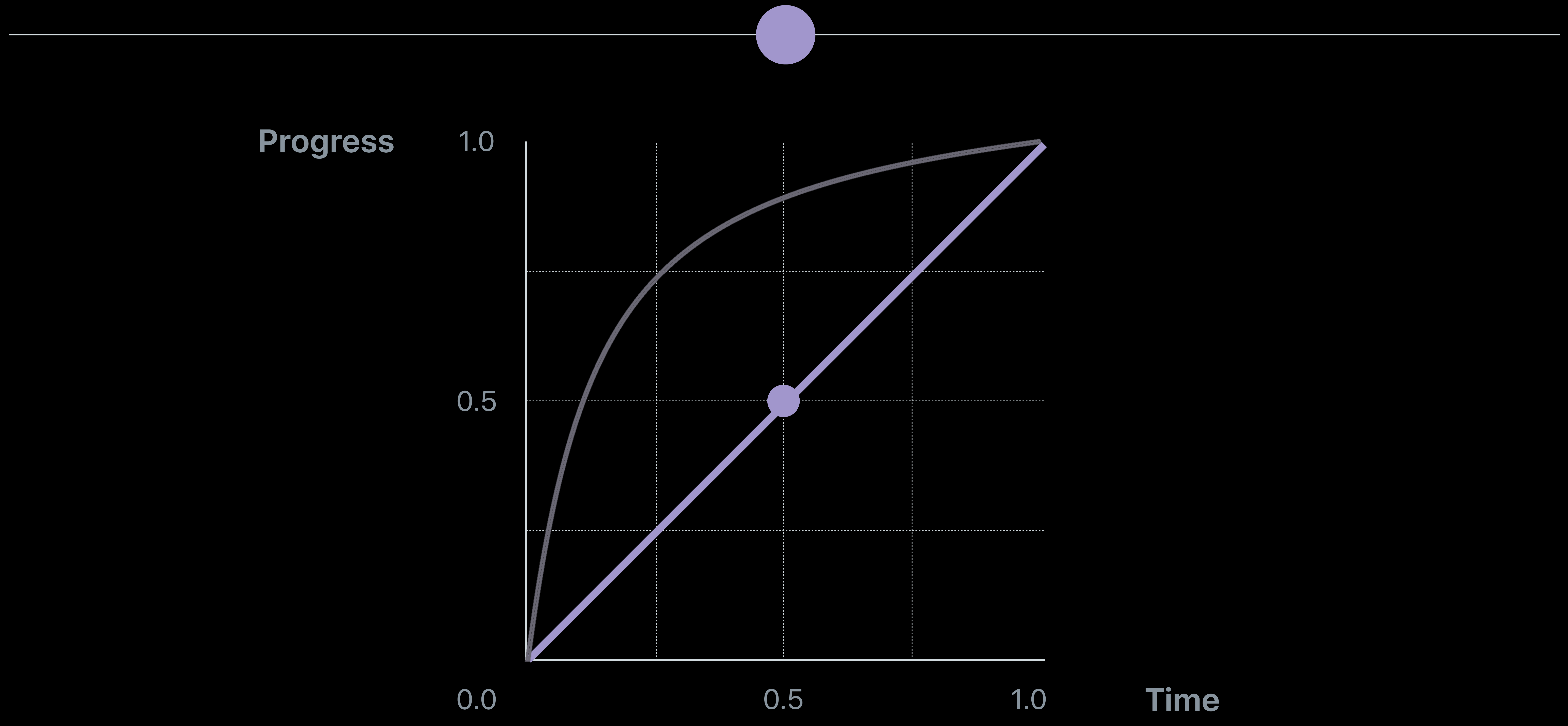
```
animator.fractionComplete = translation.x / distance
```

| **animationState** | **running** | **fractionComplete** |
|---|---|---|
| .active | false | 50% |



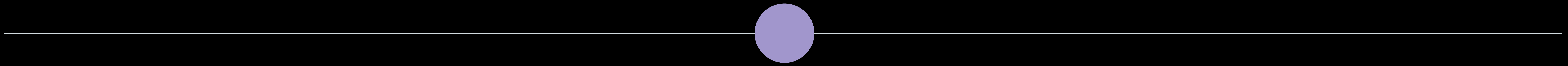


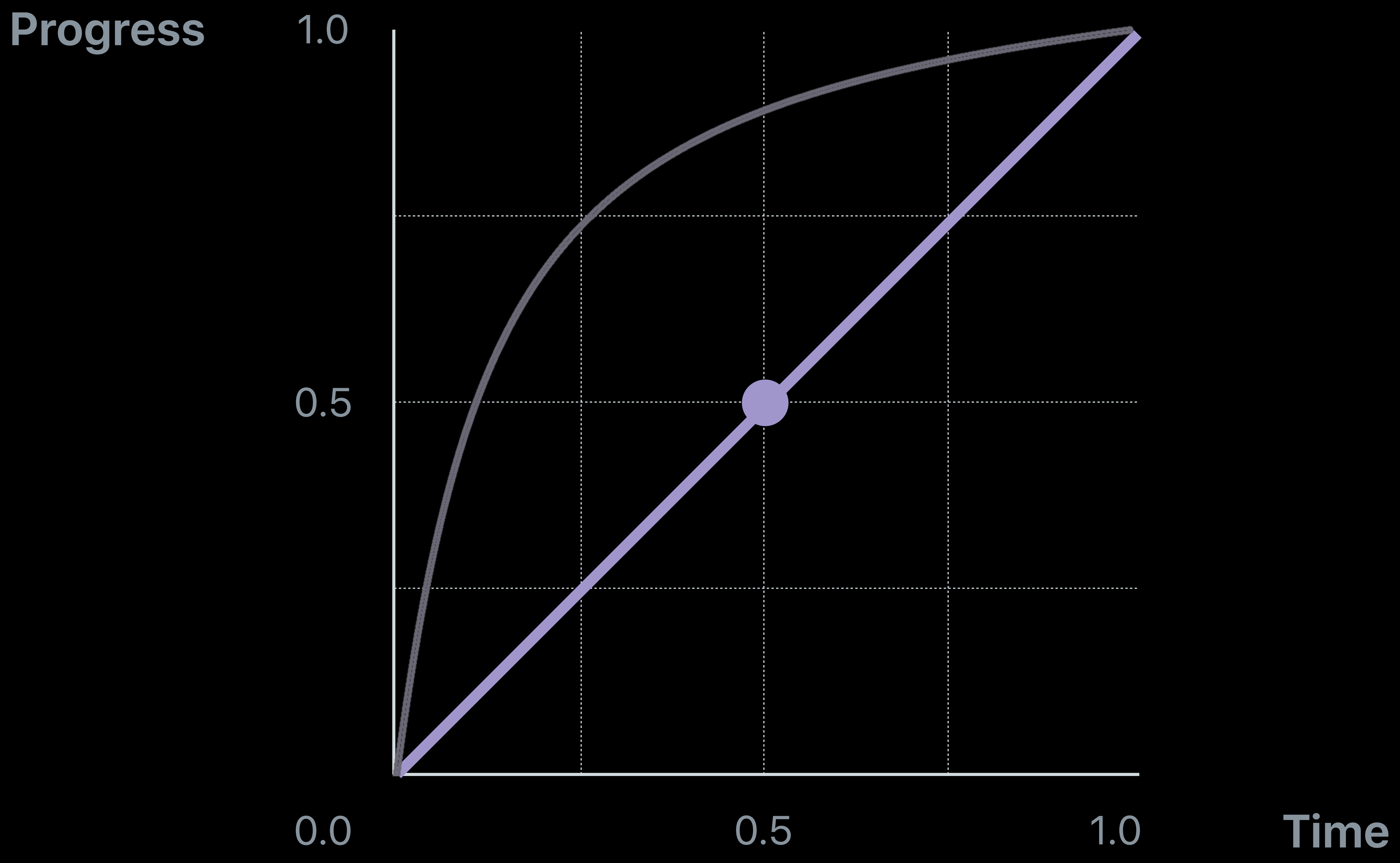
```
animator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
```



```
animator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
```

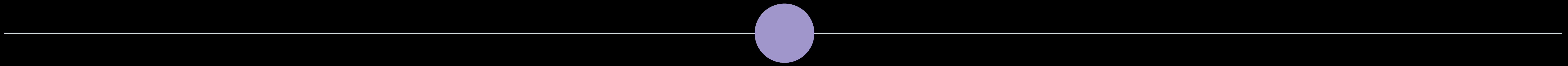
| animationState | running | fractionComplete |
|---|---|---|
| .active | true | 50% |

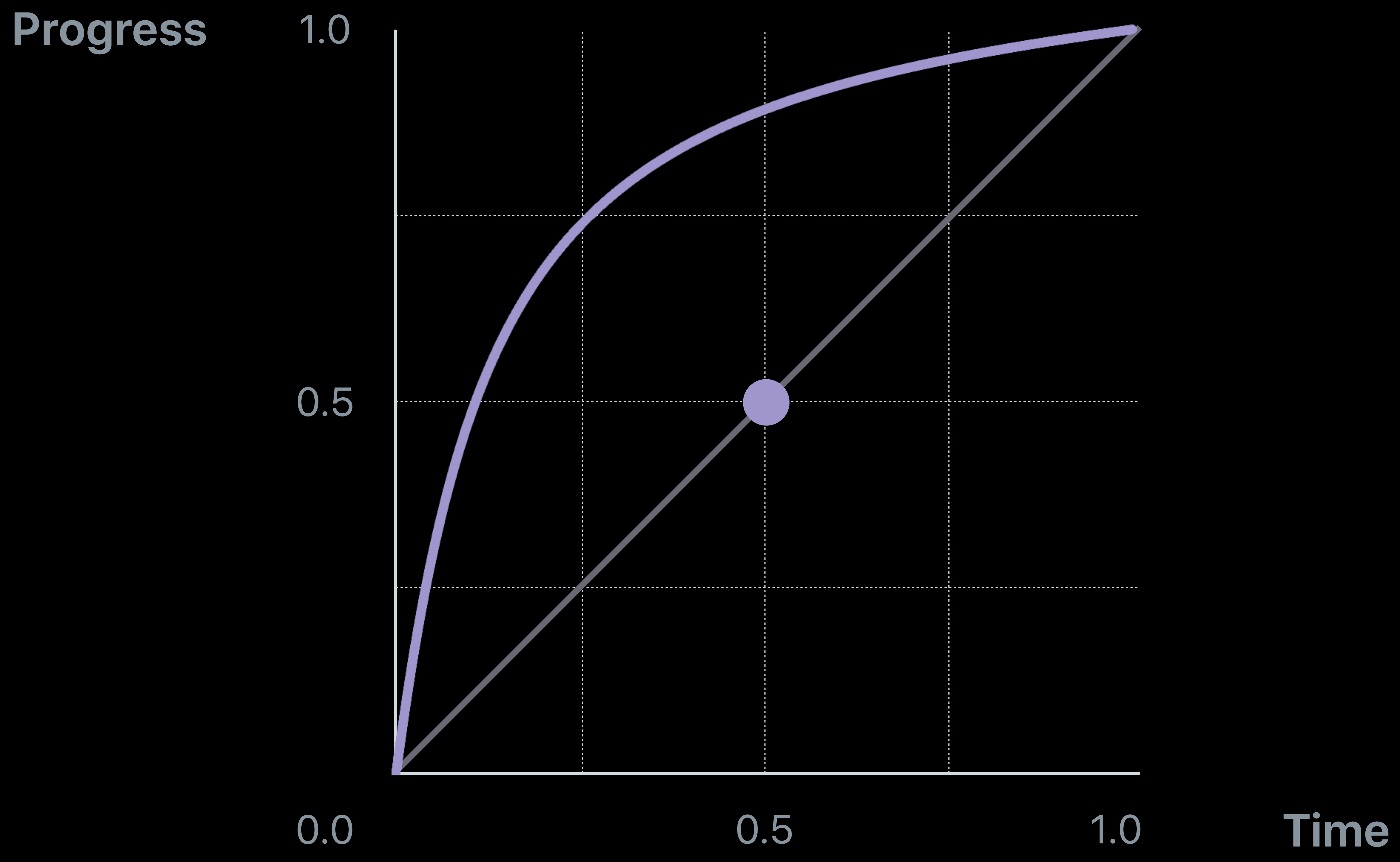




```
animator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
```

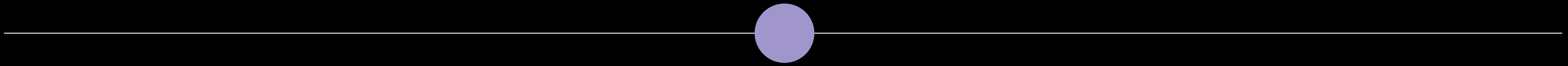
| animationState | running | fractionComplete |
|---|---|---|
| .active | true | 10% |

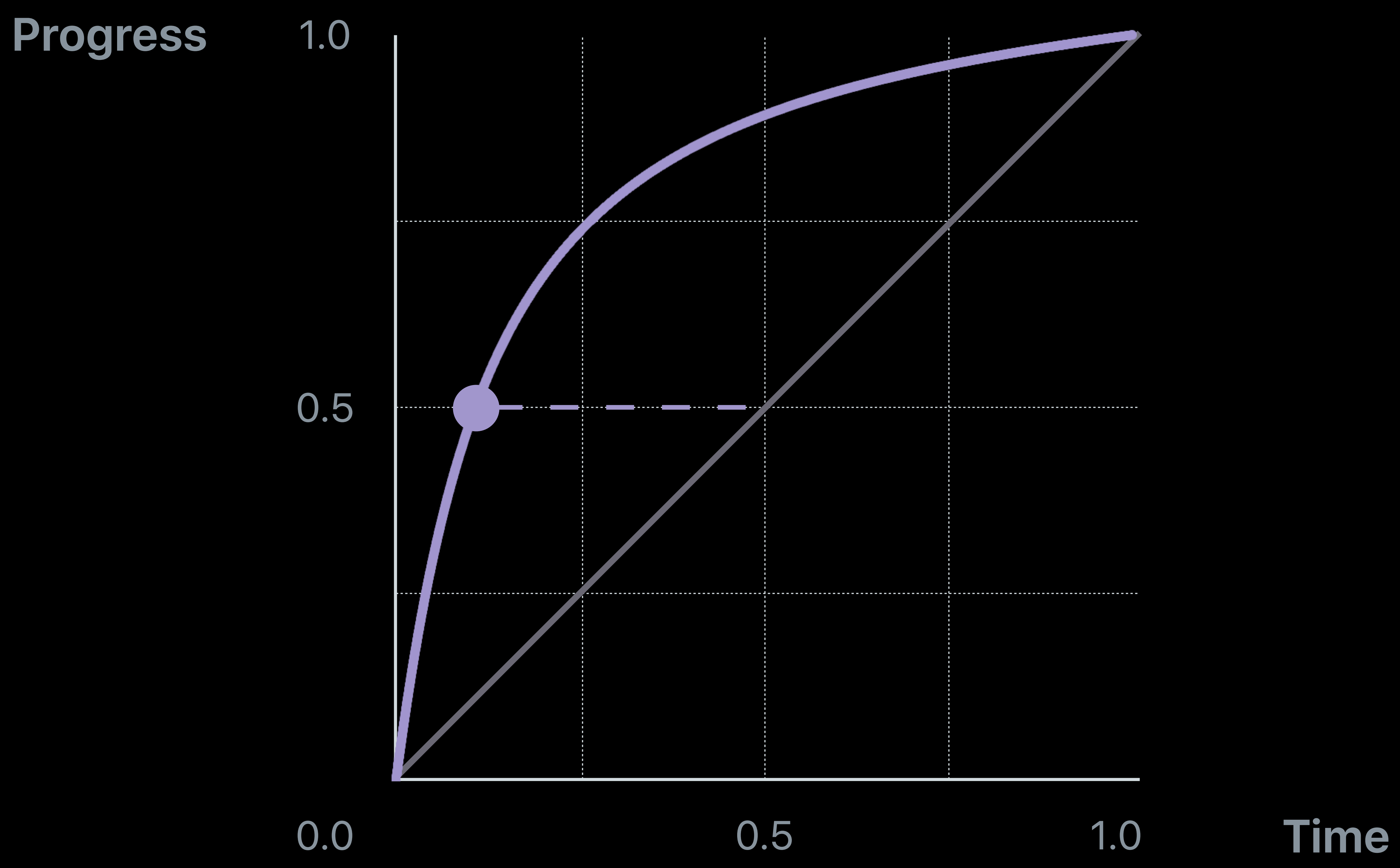




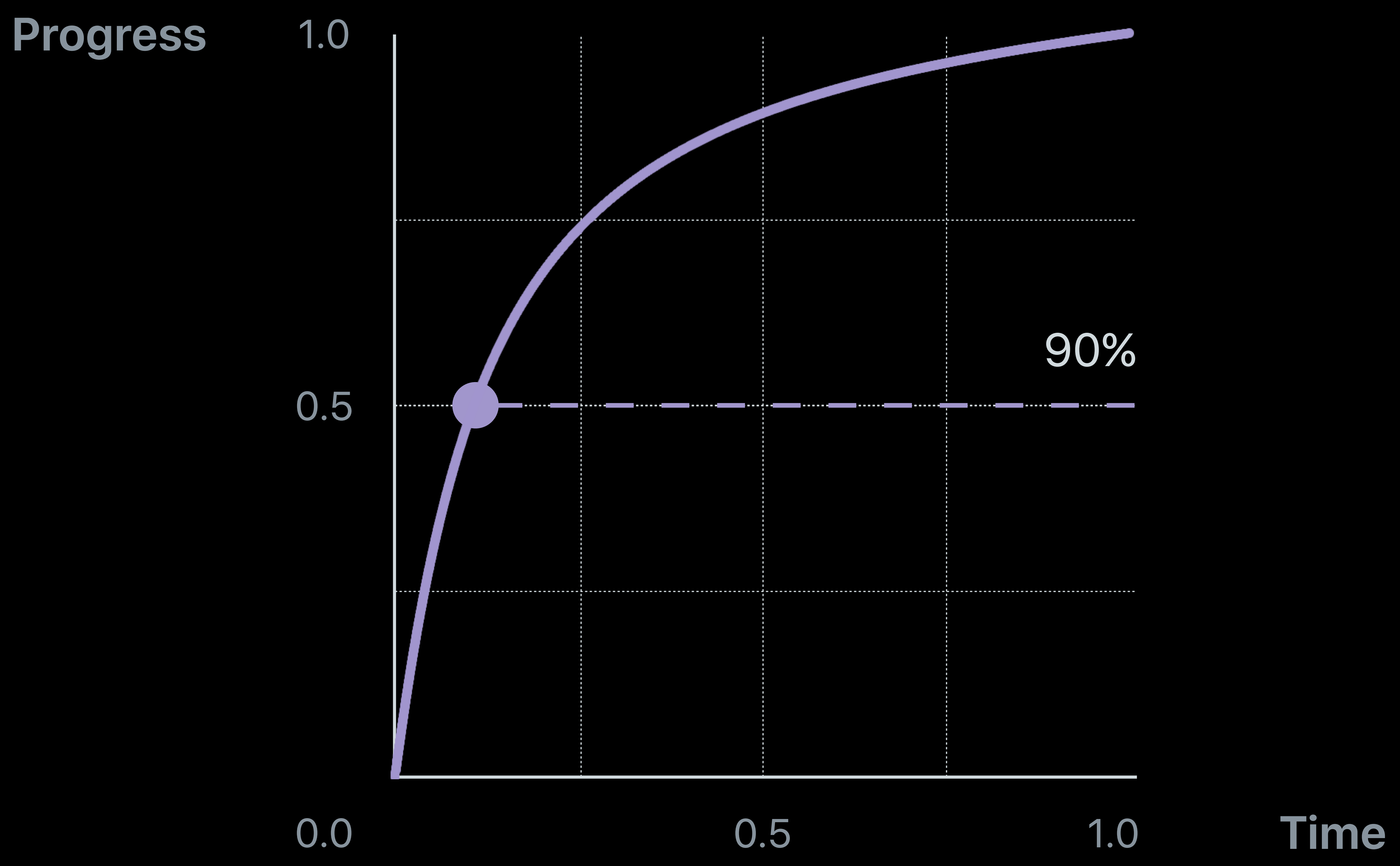
```
animator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
```

| animationState | running | fractionComplete |
|---|---|---|
| .active | true | 10% |

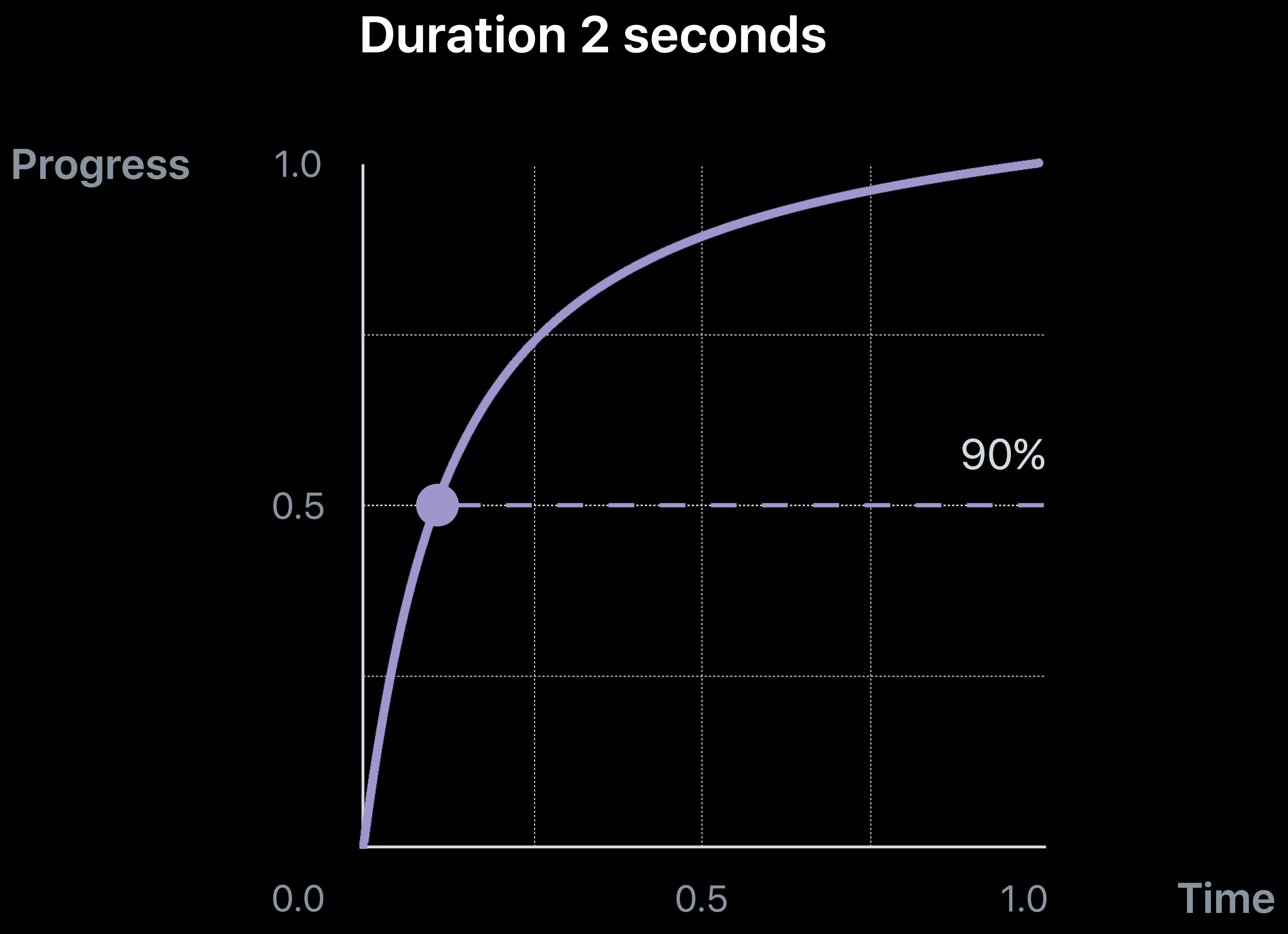




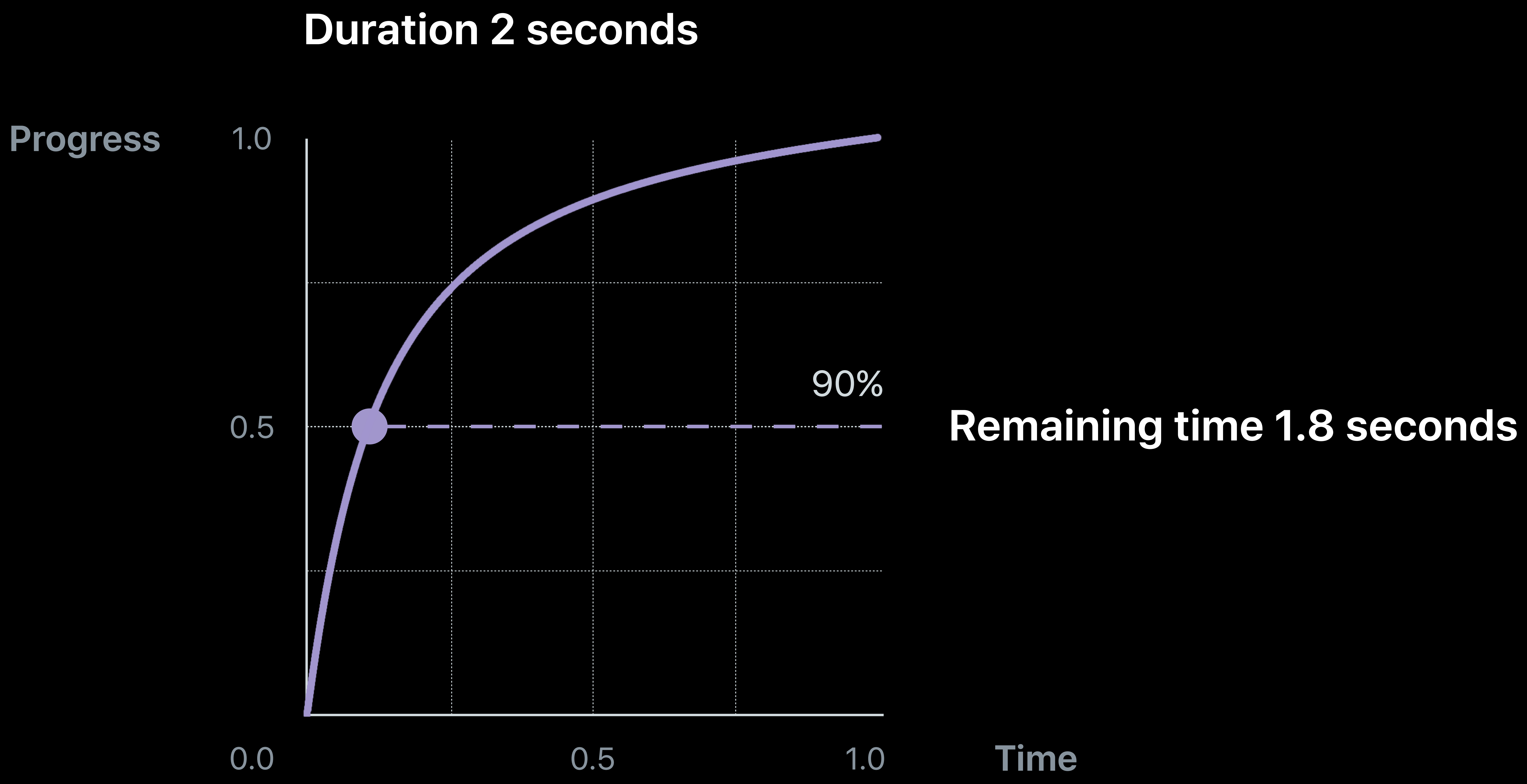
```
animator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
```



```
animator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
```



```
animator.continueAnimation(withTimingParameters: nil, durationFactor: 0)
```



## Interruptible Animations

LTE
9:41 AM
Apple Inc.
iPhone
This is 7.

LTE
9:41 AM
Apple Inc.
iPhone
This is 7.

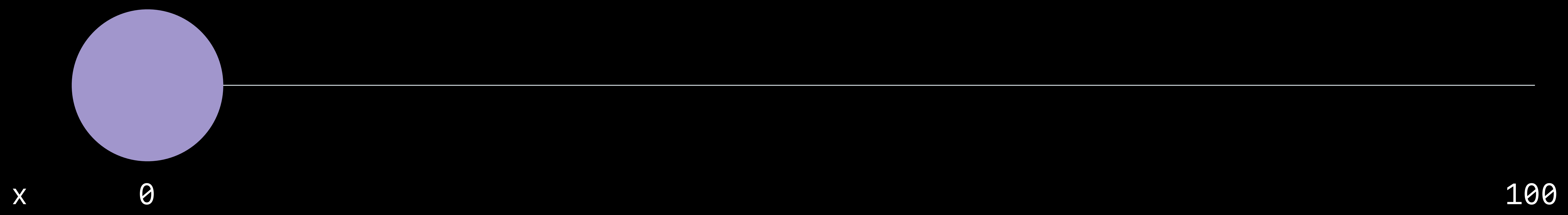
LTE
9:41 AM
Apple Inc.
iPhone
This is 7.

9:41 AM
LTE
Apple Inc.
iPhone
This is 7.

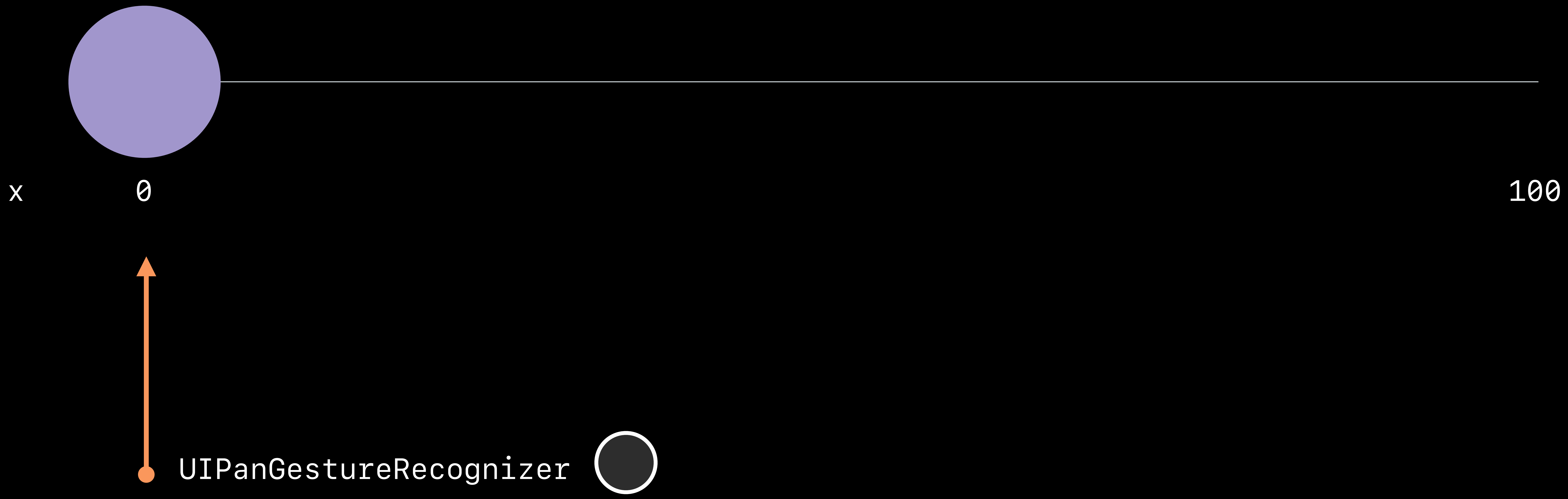
LTE
9:41 AM
Apple Inc.
iPhone
This is 7.

LTE
9:41 AM
Apple Inc.
iPhone
This is 7.

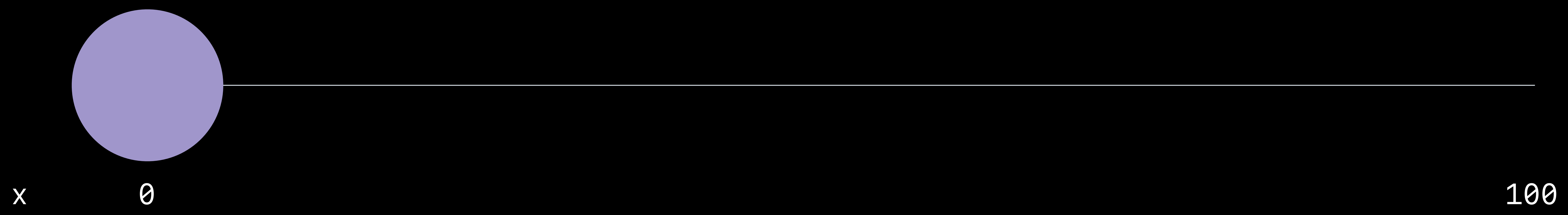
# Interrupting an Animation



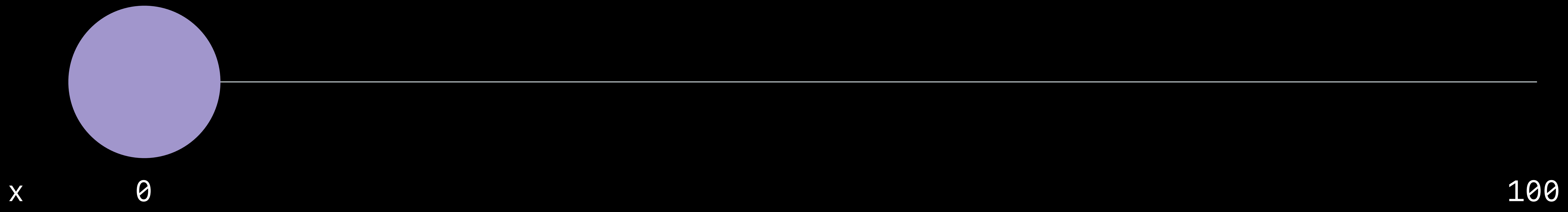
## Interrupting an Animation



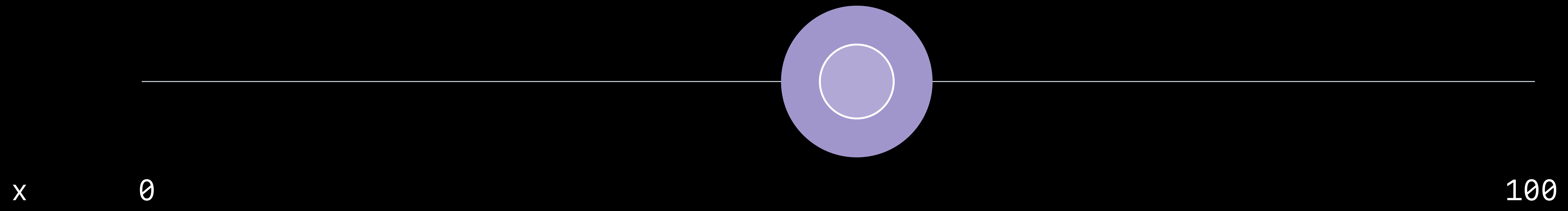
# Interrupting an Animation



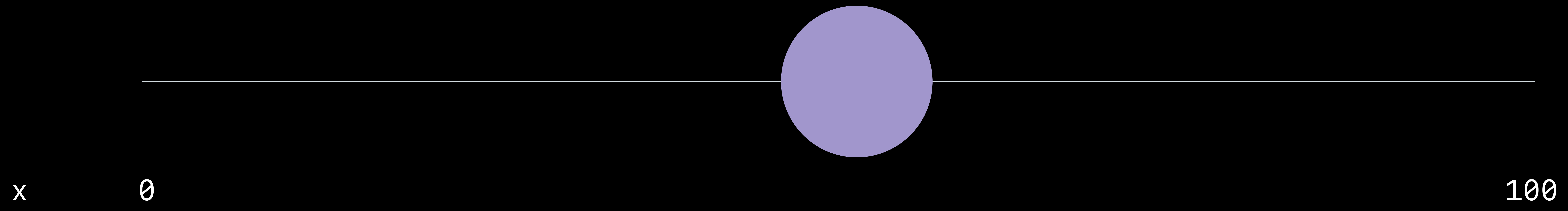
# Interrupting an Animation



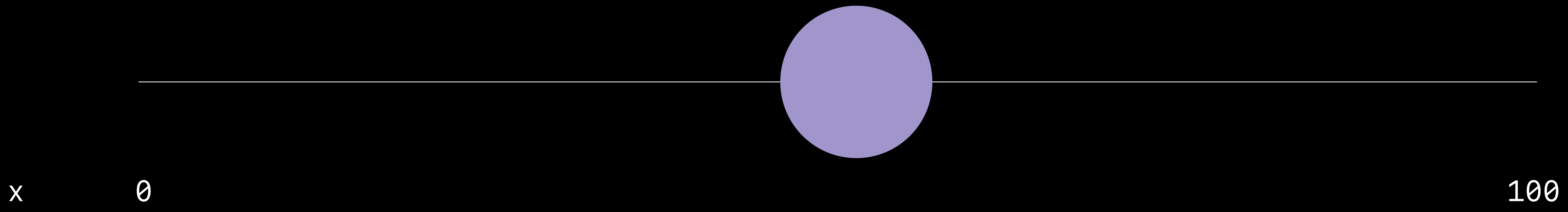
# Interrupting an Animation



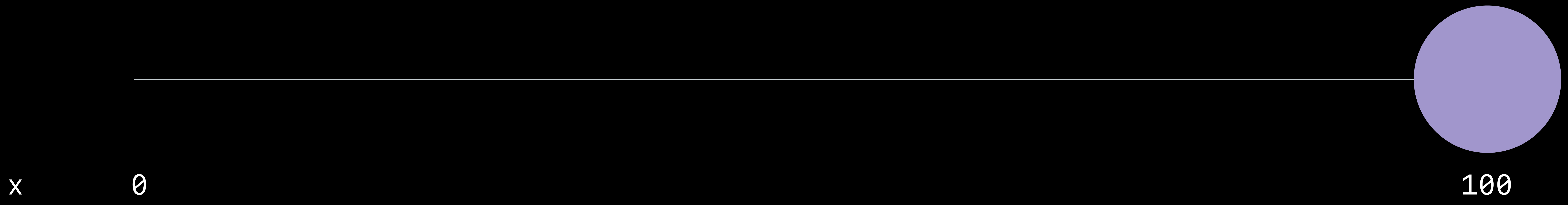
# Interrupting an Animation



## Interrupting an Animation



## Interrupting an Animation



```swift
func animateTransitionIfNeeded(duration: TimeInterval) {...}

var progressWhenInterrupted: CGFloat = 0

func handlePan(recognizer: UIPanGestureRecognizer) {
    switch recognizer.state {
    case .began:
        animateTransitionIfNeeded(duration: 1)
        animator.pauseAnimation()
        progressWhenInterrupted = animator.fractionComplete
    case .changed:
        let translation = recognizer.translation(in: circle)
        animator.fractionComplete = (translation.x / 100) + progressWhenInterrupted
    case .ended:
        let timing = UICubicTimingParameters(animationCurve: .easeOut)
        animator.continueAnimation(withTimingParameters: timing, durationFactor: 0)
    }
}
```

```swift
func animateTransitionIfNeeded(duration: TimeInterval) {...}
```

```swift
var progressWhenInterrupted: CGFloat = 0

func handlePan(recognizer: UIPanGestureRecognizer) {
    switch recognizer.state {
    case .began:
        animateTransitionIfNeeded(duration: 1)
        animator.pauseAnimation()
        progressWhenInterrupted = animator.fractionComplete
    case .changed:
        let translation = recognizer.translation(in: circle)
        animator.fractionComplete = (translation.x / 100) + progressWhenInterrupted
    case .ended:
        let timing = UICubicTimingParameters(animationCurve: .easeOut)
        animator.continueAnimation(withTimingParameters: timing, durationFactor: 0)
    }
}
```

```swift
func animateTransitionIfNeeded(duration: TimeInterval) {...}

var progressWhenInterrupted: CGFloat = 0

func handlePan(recognizer: UIPanGestureRecognizer) {
    switch recognizer.state {
    case .began:
        animateTransitionIfNeeded(duration: 1)
        animator.pauseAnimation()
        progressWhenInterrupted = animator.fractionComplete
    case .changed:
        let translation = recognizer.translation(in: circle)
        animator.fractionComplete = (translation.x / 100) + progressWhenInterrupted
    case .ended:
        let timing = UICubicTimingParameters(animationCurve: .easeOut)
        animator.continueAnimation(withTimingParameters: timing, durationFactor: 0)
    }
}
```

```swift
func animateTransitionIfNeeded(duration: TimeInterval) {...}

var progressWhenInterrupted: CGFloat = 0

func handlePan(recognizer: UIPanGestureRecognizer) {
    switch recognizer.state {
    case .began:
        animateTransitionIfNeeded(duration: 1)
        animator.pauseAnimation()
        progressWhenInterrupted = animator.fractionComplete
    case .changed:
        let translation = recognizer.translation(in: circle)
        animator.fractionComplete = (translation.x / 100) + progressWhenInterrupted
    case .ended:
        let timing = UICubicTimingParameters(animationCurve: .easeOut)
        animator.continueAnimation(withTimingParameters: timing, durationFactor: 0)
    }
}
```

```swift
func animateTransitionIfNeeded(duration: TimeInterval) {...}

var progressWhenInterrupted: CGFloat = 0

func handlePan(recognizer: UIPanGestureRecognizer) {
    switch recognizer.state {
    case .began:
        animateTransitionIfNeeded(duration: 1)
        animator.pauseAnimation()
        progressWhenInterrupted = animator.fractionComplete
    case .changed:
        let translation = recognizer.translation(in: circle)
        animator.fractionComplete = (translation.x / 100) + progressWhenInterrupted
    case .ended:
        let timing = UICubicTimingParameters(animationCurve: .easeOut)
        animator.continueAnimation(withTimingParameters: timing, durationFactor: 0)
    }
}
```

```swift
func animateTransitionIfNeeded(duration: TimeInterval) {...}

var progressWhenInterrupted: CGFloat = 0

func handlePan(recognizer: UIPanGestureRecognizer) {
    switch recognizer.state {
    case .began:
        animateTransitionIfNeeded(duration: 1)
        animator.pauseAnimation()
        progressWhenInterrupted = animator.fractionComplete
    case .changed:
        let translation = recognizer.translation(in: circle)
        animator.fractionComplete = (translation.x / 100) + progressWhenInterrupted
    case .ended:
        let timing = UICubicTimingParameters(animationCurve: .easeOut)
        animator.continueAnimation(withTimingParameters: timing, durationFactor: 0)
    }
}
```

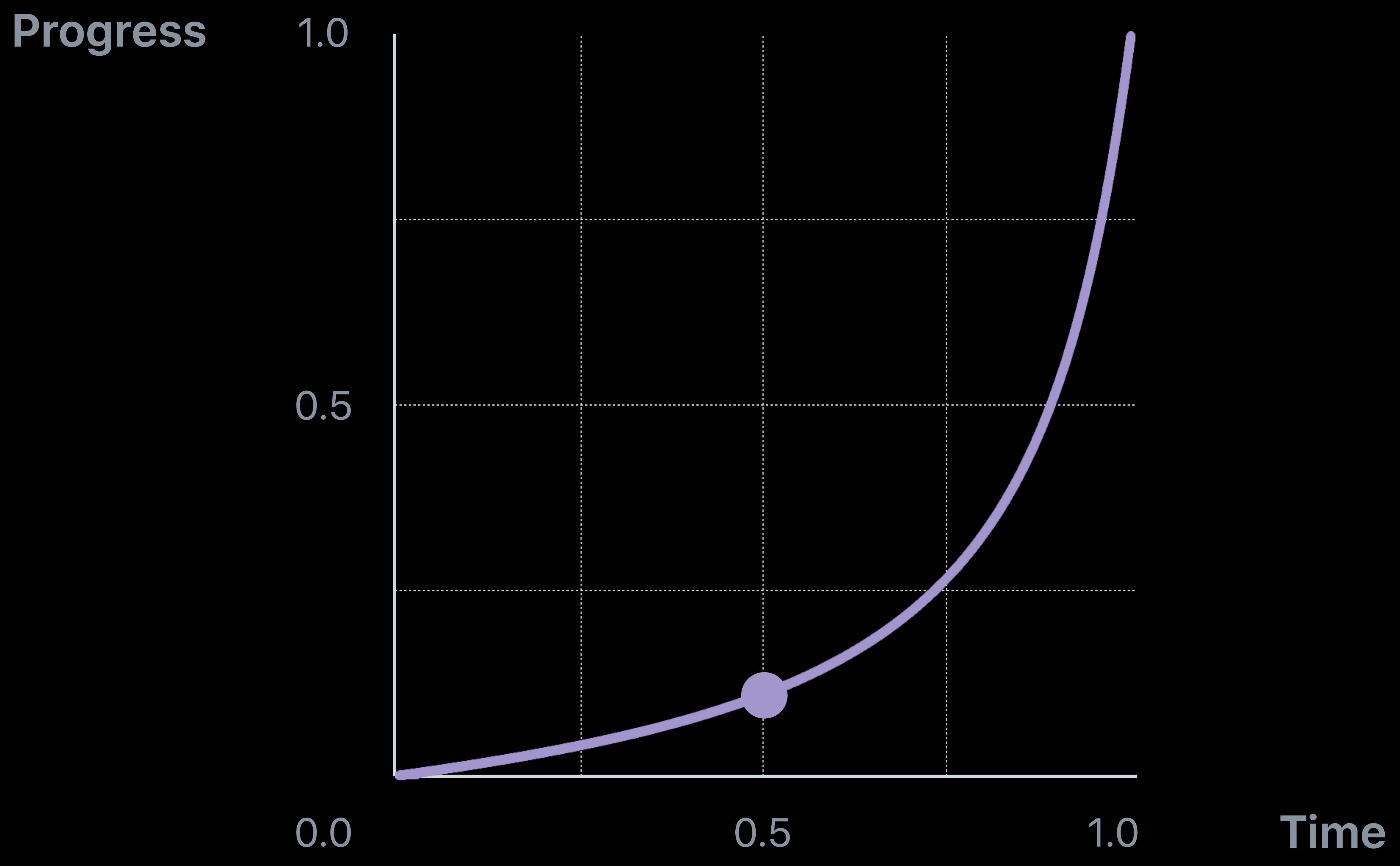
## Time Conversion

Pausing

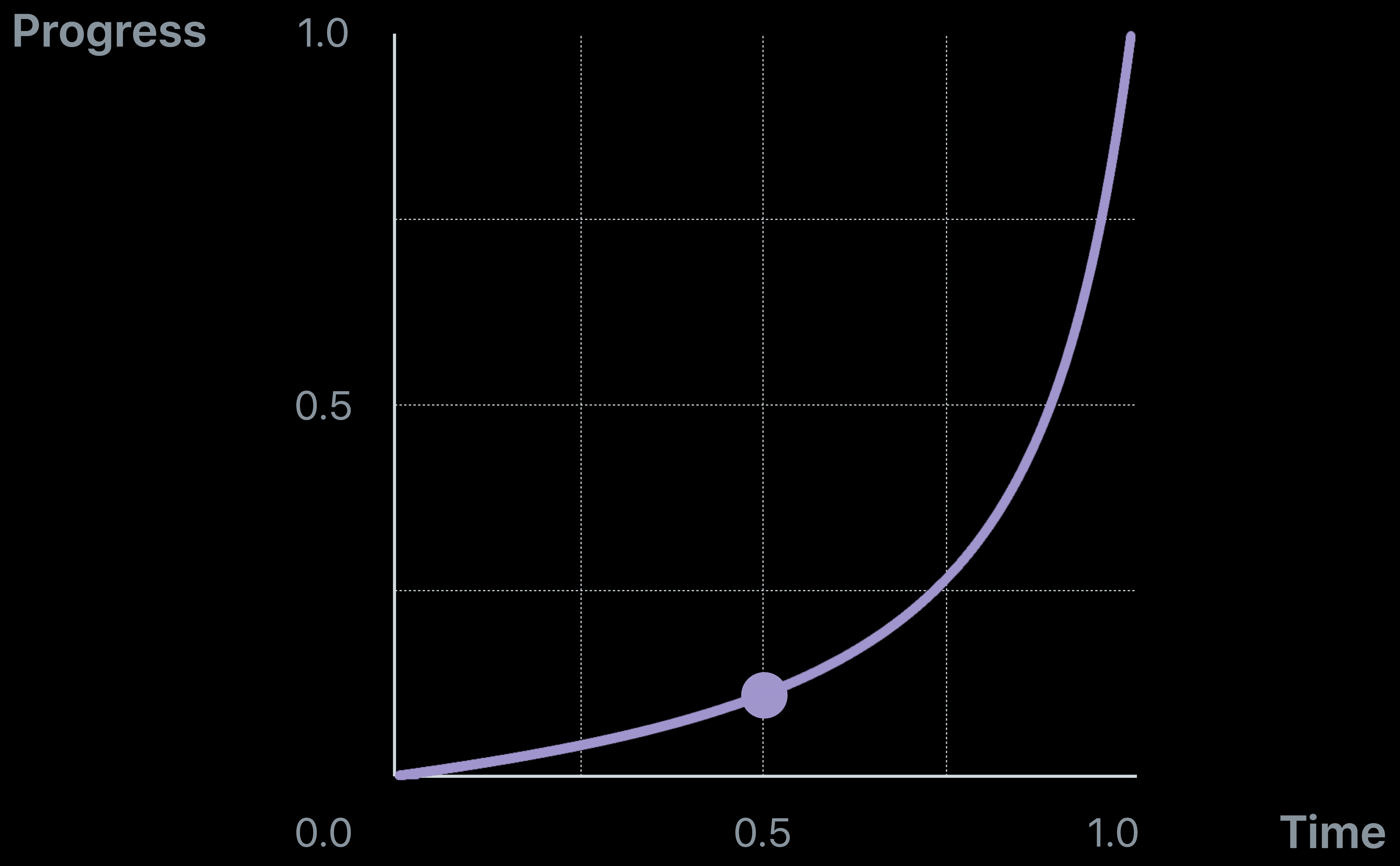
Continuing

## animator.isRunning

| animationState | running | fractionComplete |
|---|---|---|
| .active | false | 50% |



```
animator.pauseAnimation()
```

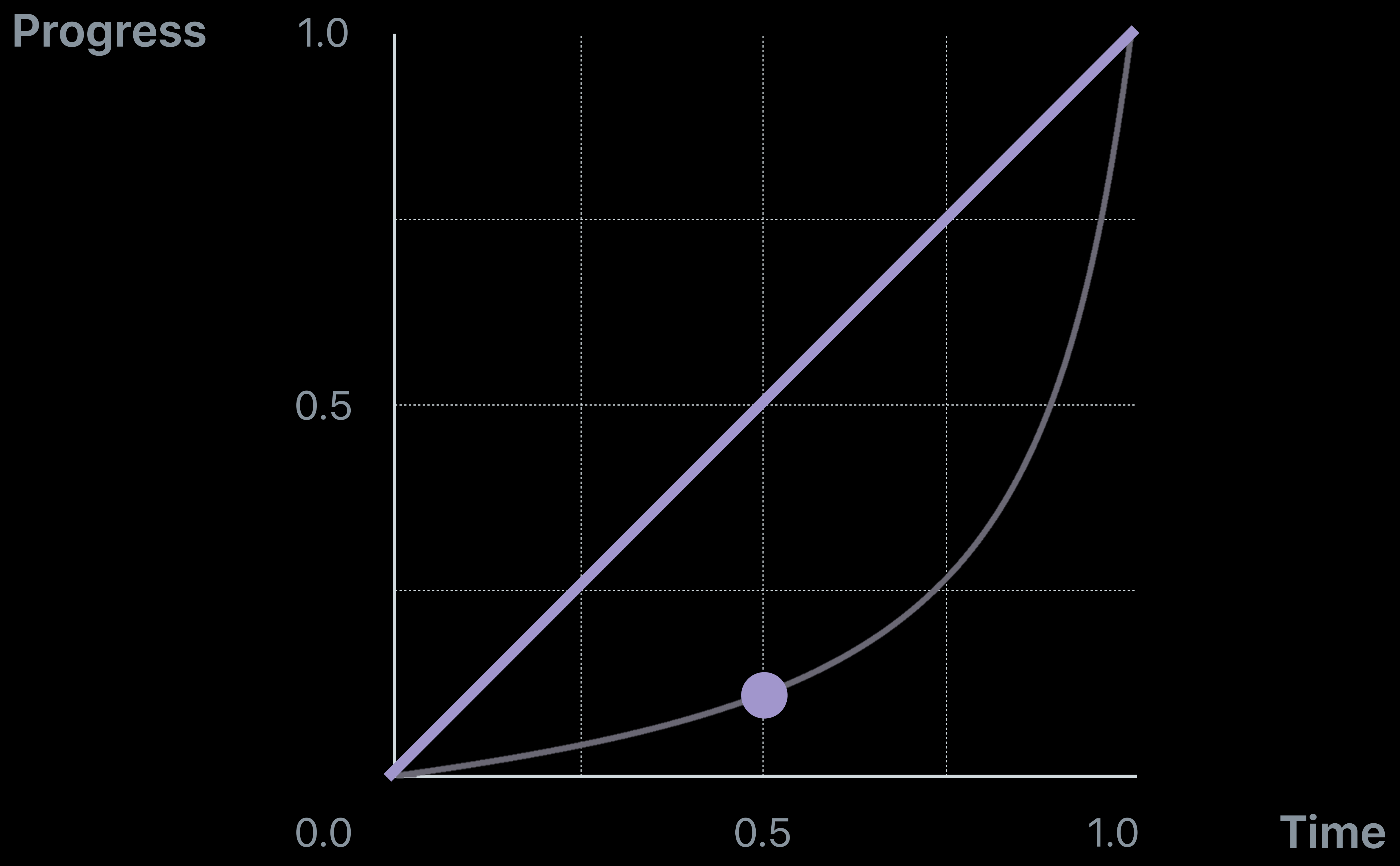


```
animator.pauseAnimation()
```

**animationState**
.active

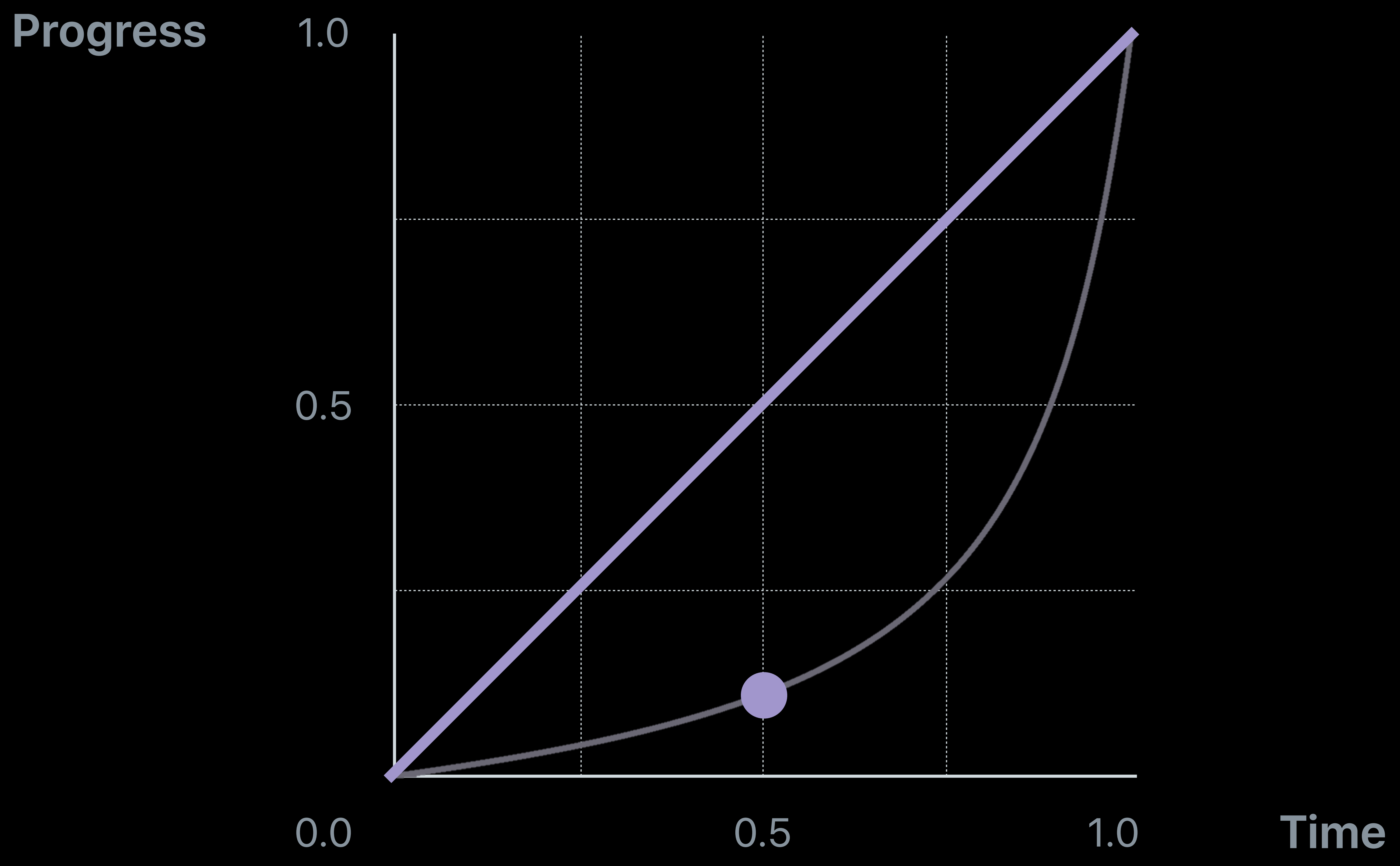
**running**
**false**

**fractionComplete**
50%



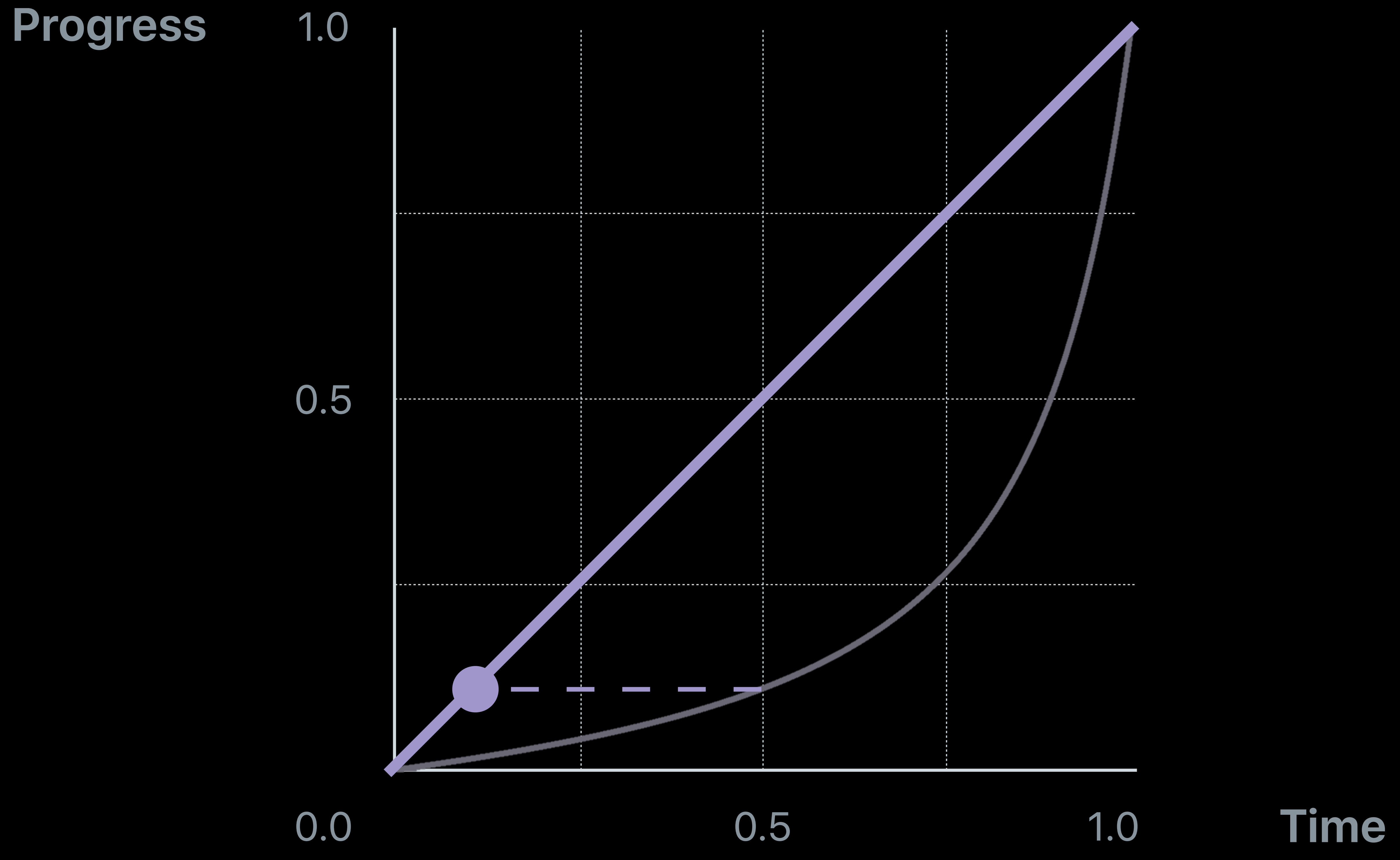
```
animator.pauseAnimation()
```

| animationState | running | fractionComplete |
|---|---|---|
| .active | false | 10% |



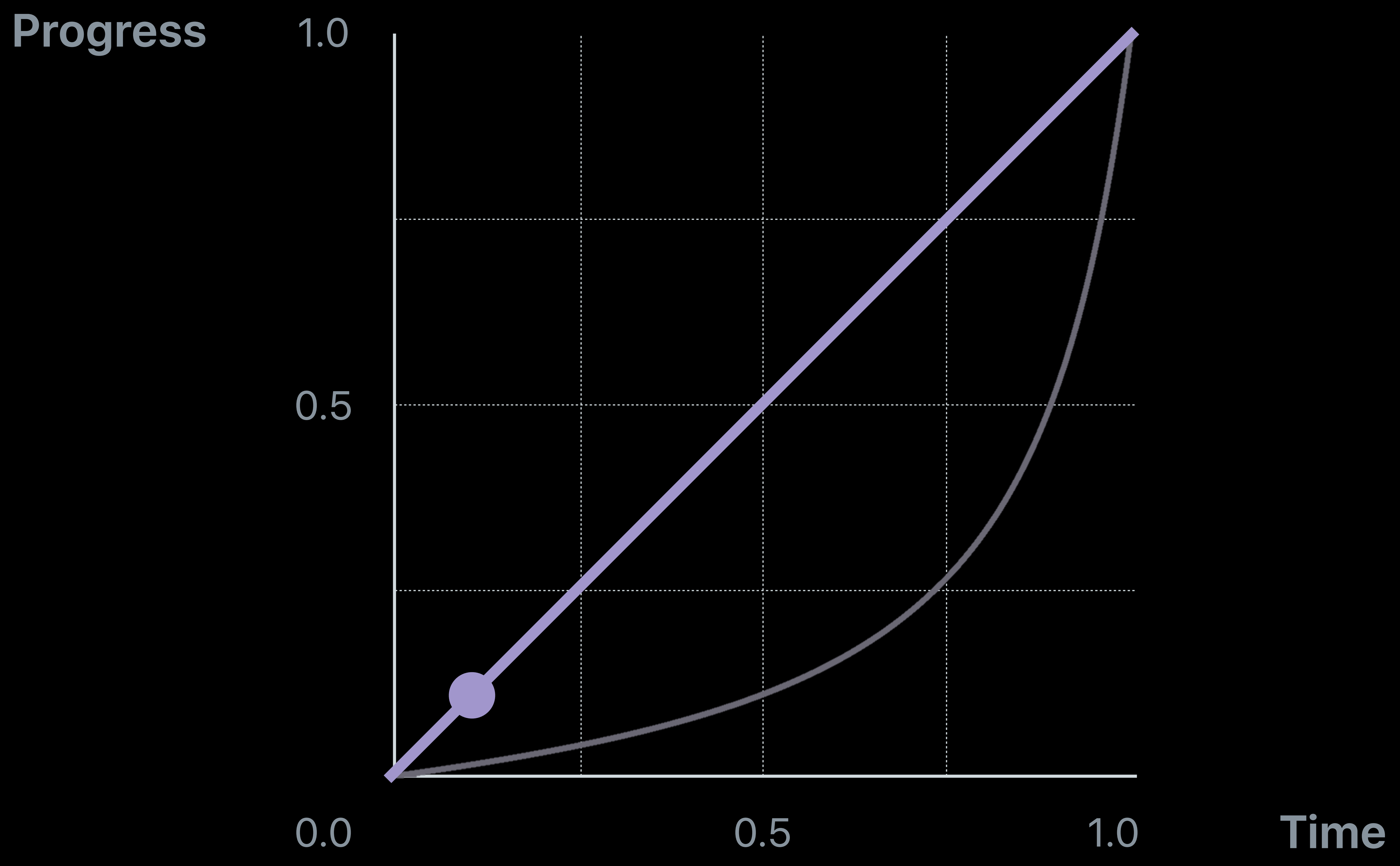
```
animator.pauseAnimation()
```

| animationState | running | fractionComplete |
|---|---|---|
| .active | false | 10% |

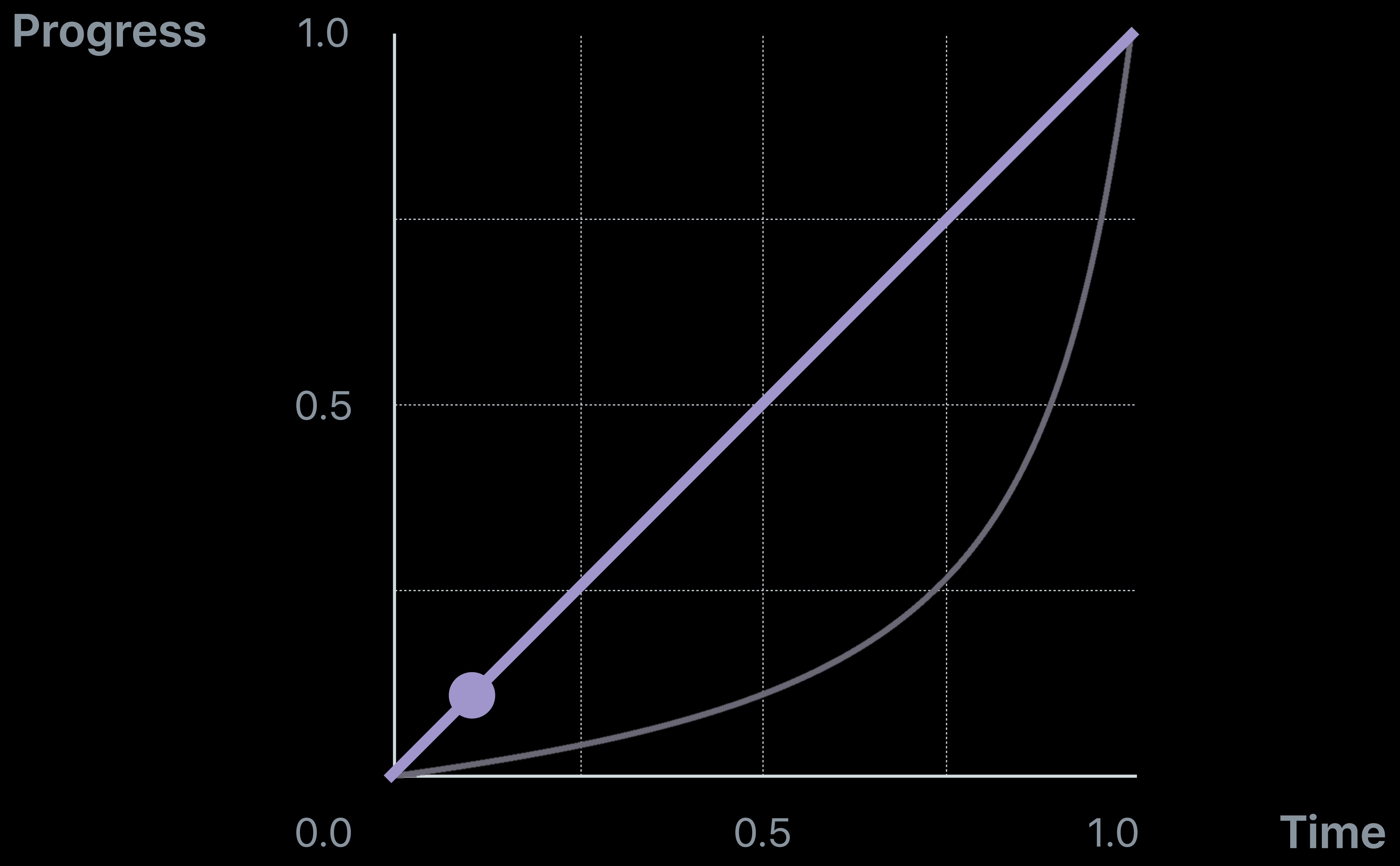


```
animator.fractionComplete = 0.1
```

| **animationState** | **running** | **fractionComplete** |
|---|---|---|
| .active | false | 10% |

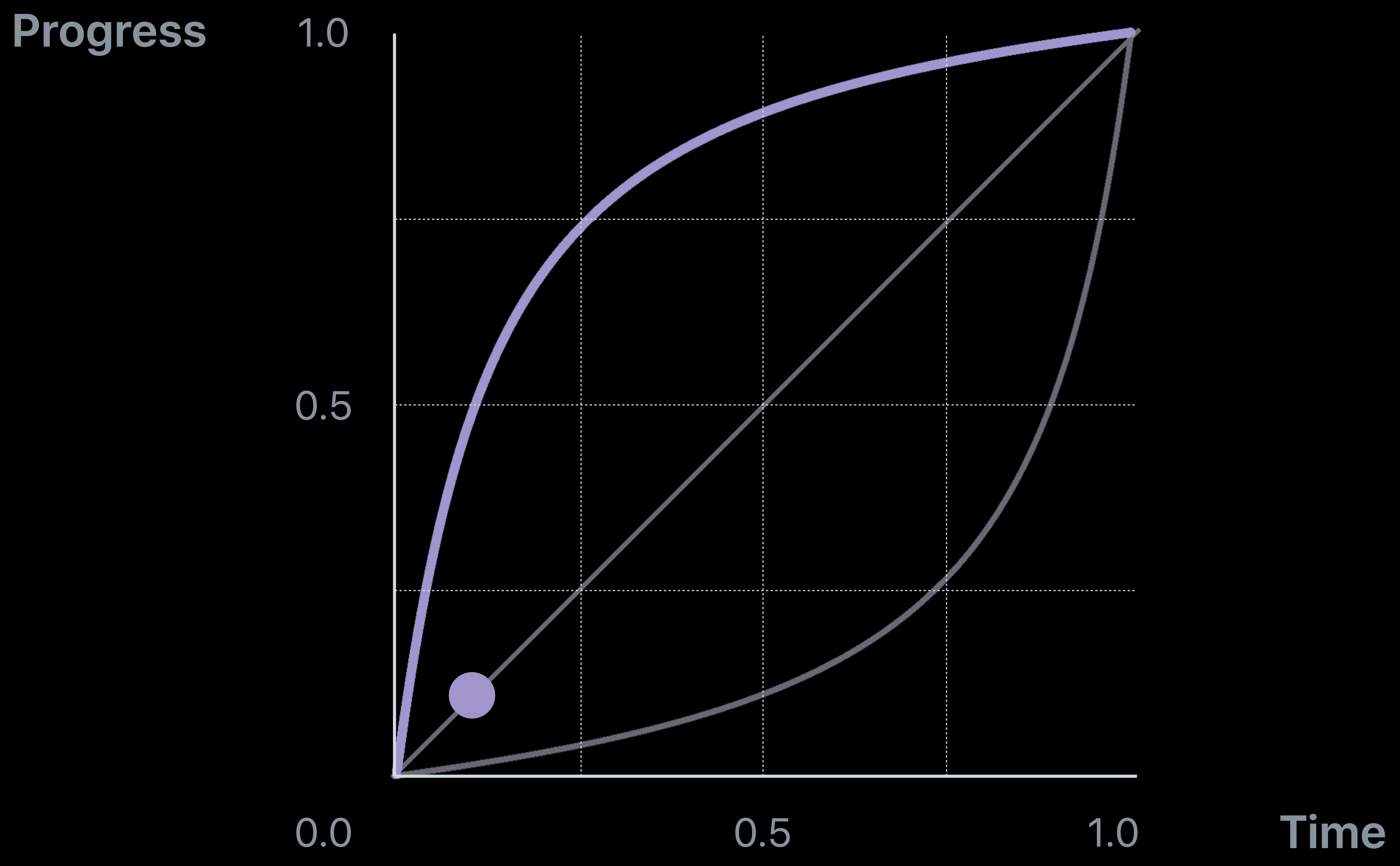


```
animator.continueAnimation(... animationCurve: .easeOut ...)
```



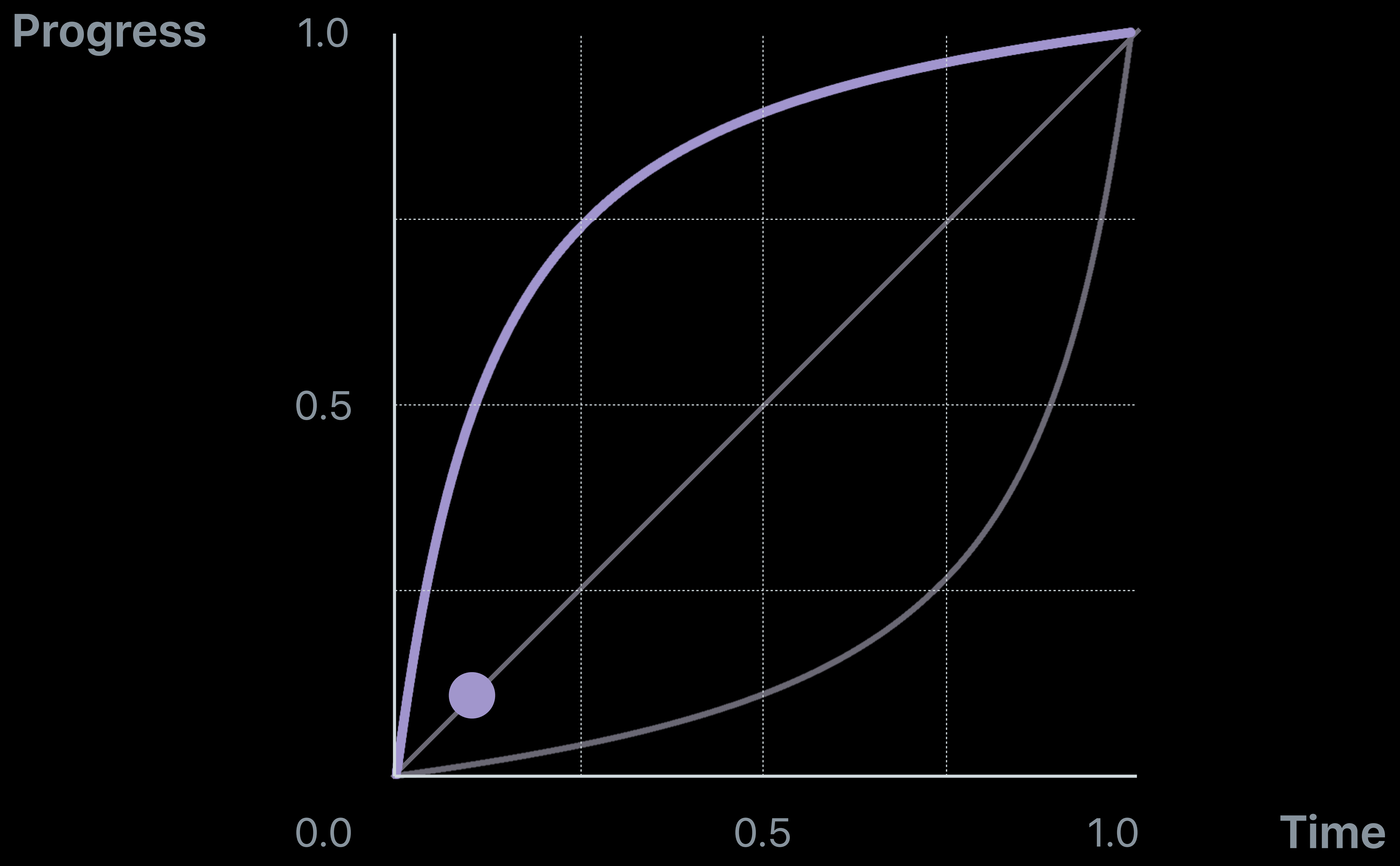
```
animator.continueAnimation(... animationCurve: .easeOut ...)
```

| animationState | running | fractionComplete |
|---|---|---|
| .active | true | 10% |



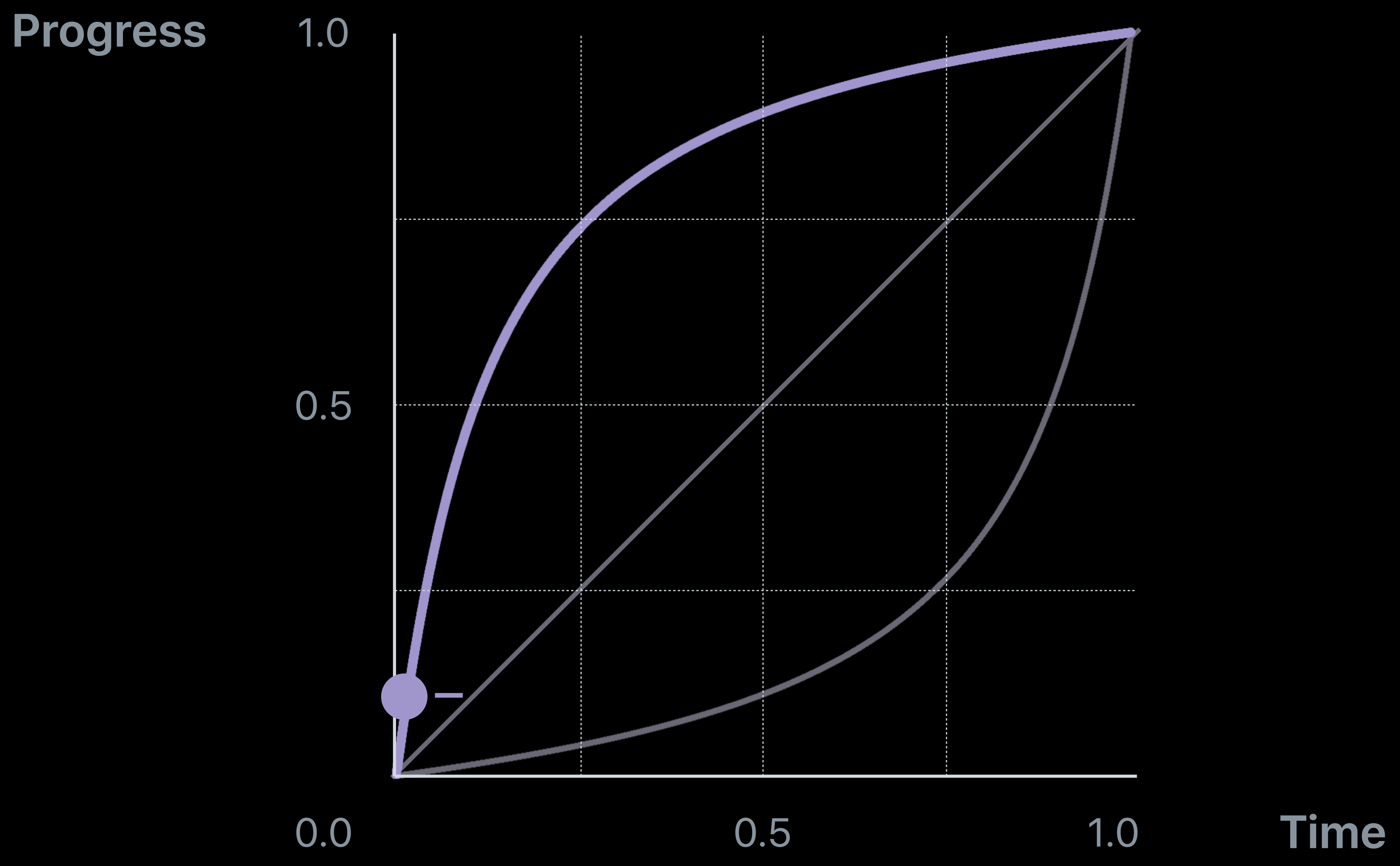
```
animator.continueAnimation(... animationCurve: .easeOut ...)
```

| animationState | running | fractionComplete |
|---|---|---|
| .active | true | 5% |



```
animator.continueAnimation(... animationCurve: .easeOut ...)
```

| animationState | running | fractionComplete |
|---|---|---|
| .active | true | 5% |



## New Animator Behaviors

# UIViewPropertyAnimator

New in iOS 11

NEW

## UIViewPropertyAnimator

New in iOS 11

NEW

```
var scrubsLinearly: Bool
var pausesOnCompletion: Bool
```

# UIViewPropertyAnimator

New in iOS 11

NEW

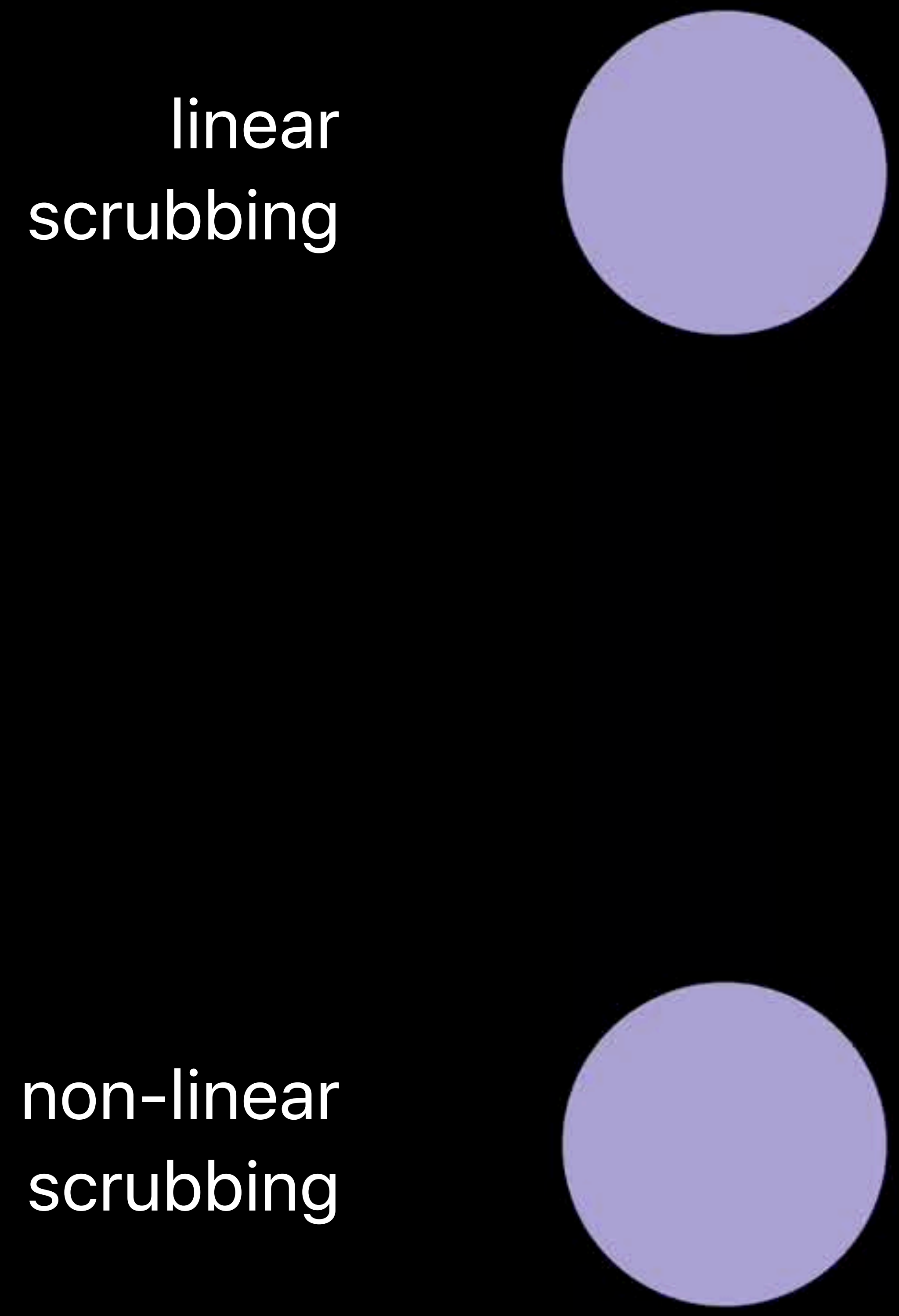
```
var scrubsLinearly: Bool
var pausesOnCompletion: Bool
```

Starting as Paused

# .scrubsLinearly

## Non-linear scrubbing

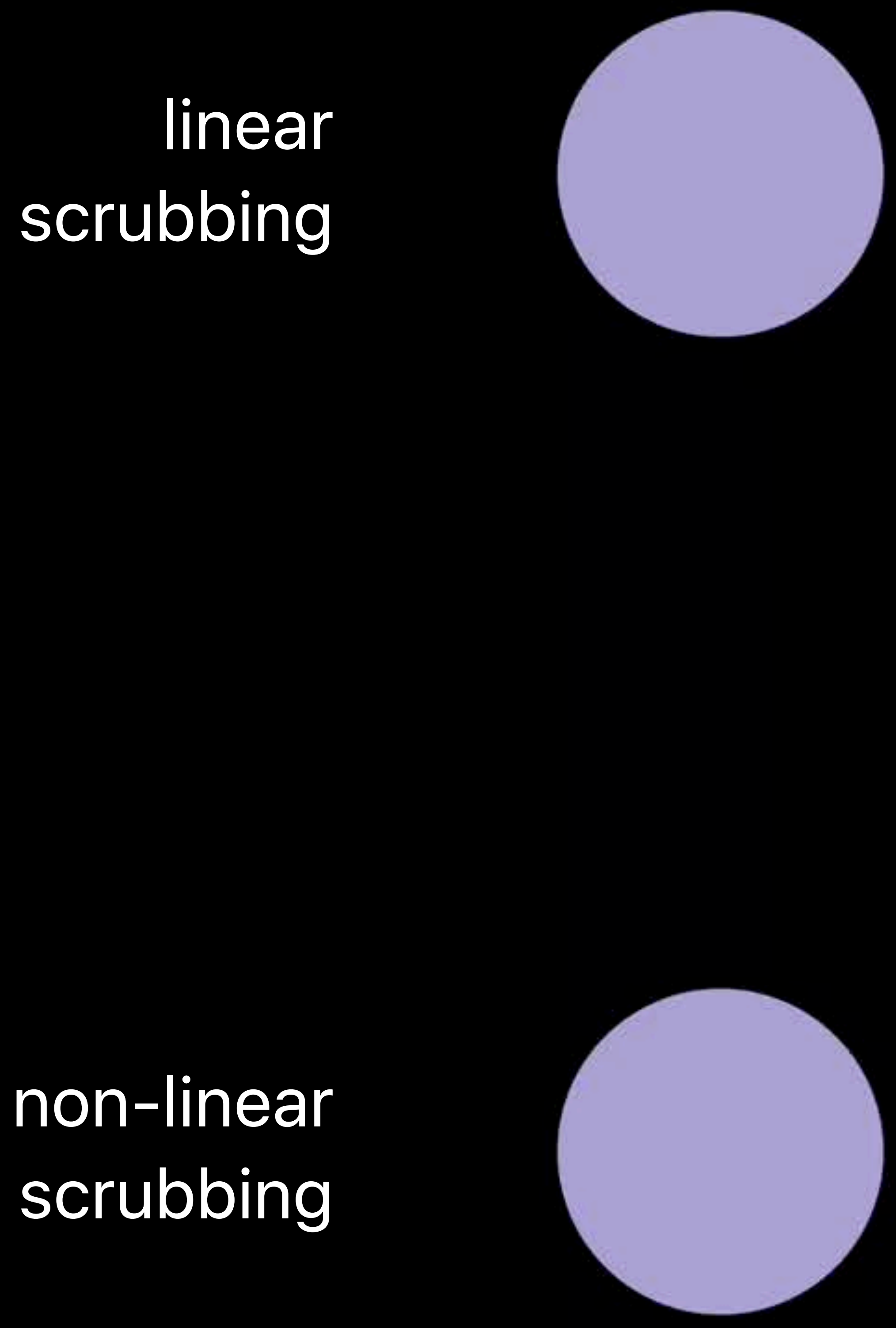
NEW



## .scrubsLinearly

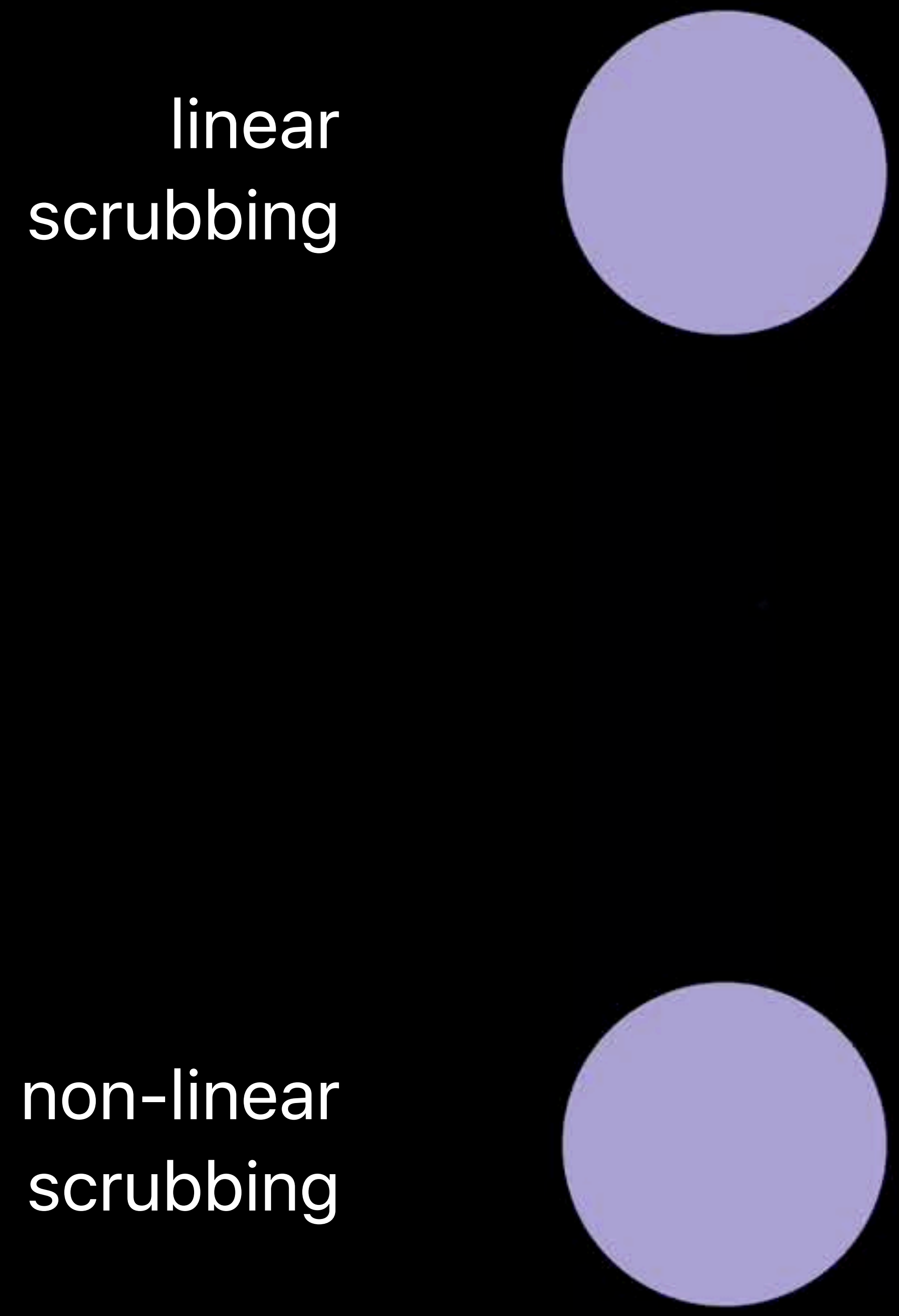
Non-linear scrubbing

NEW



## .scrubsLinearly
### Non-linear scrubbing

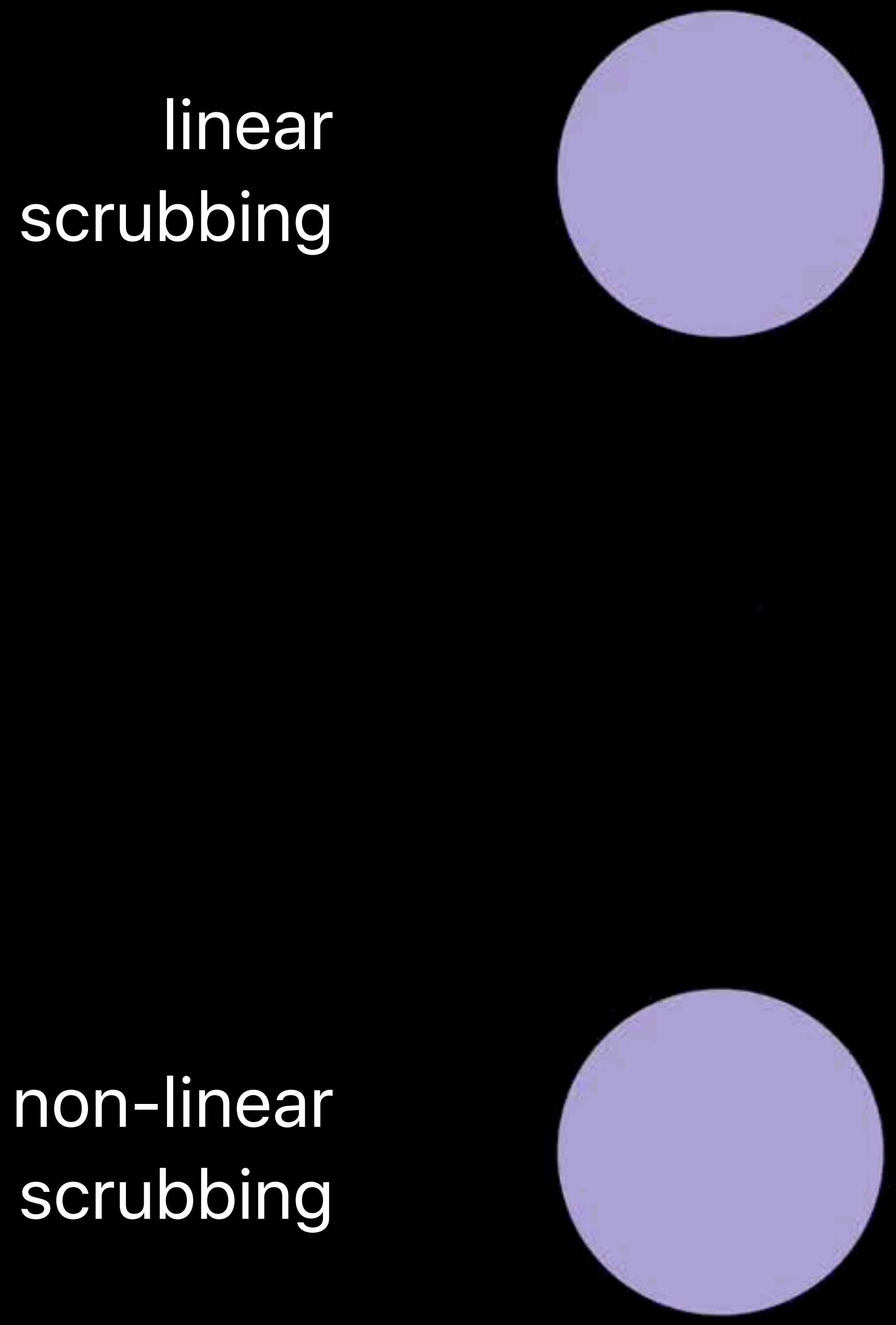
NEW



## .scrubsLinearly

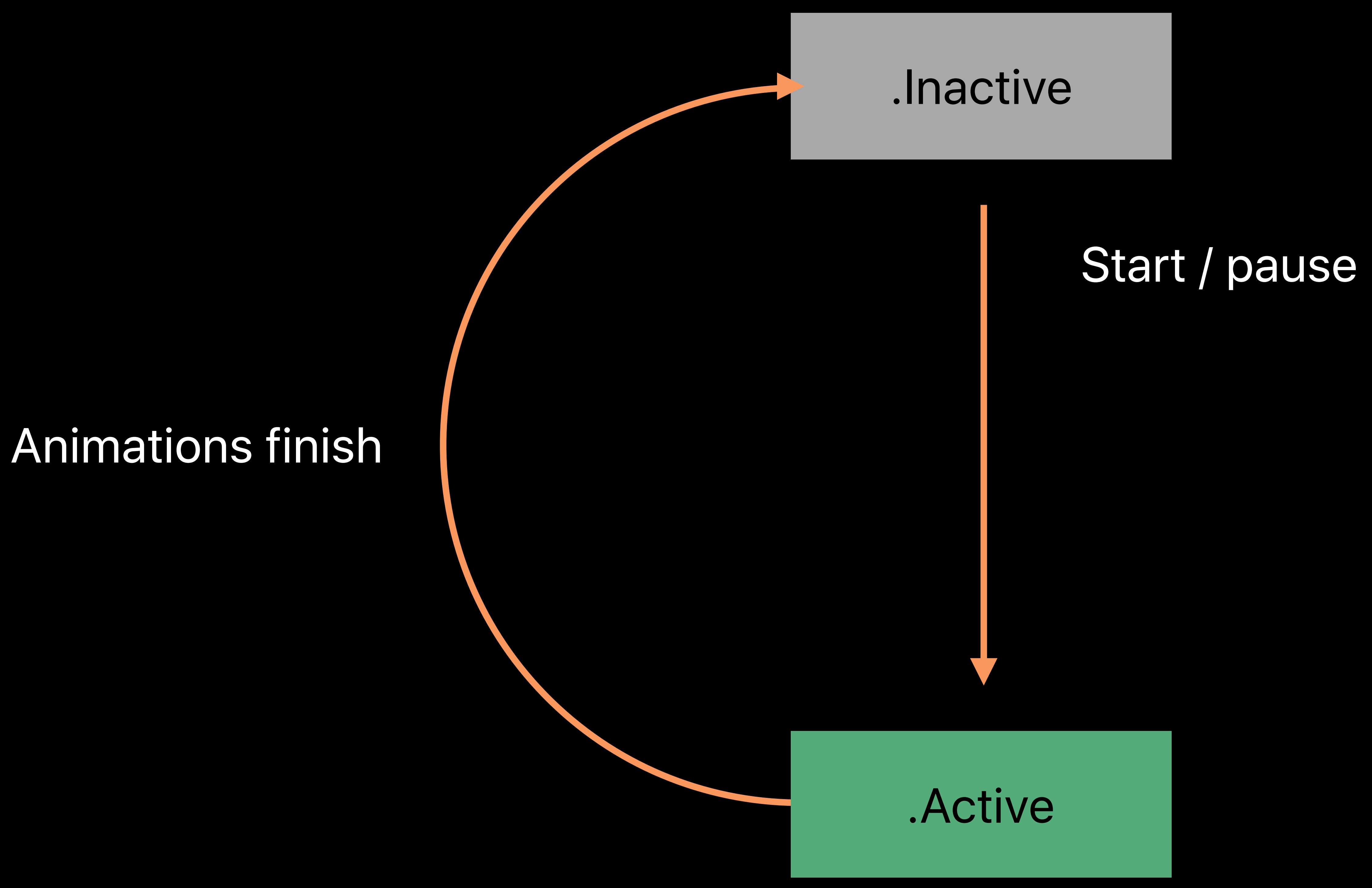
Non-linear scrubbing

NEW



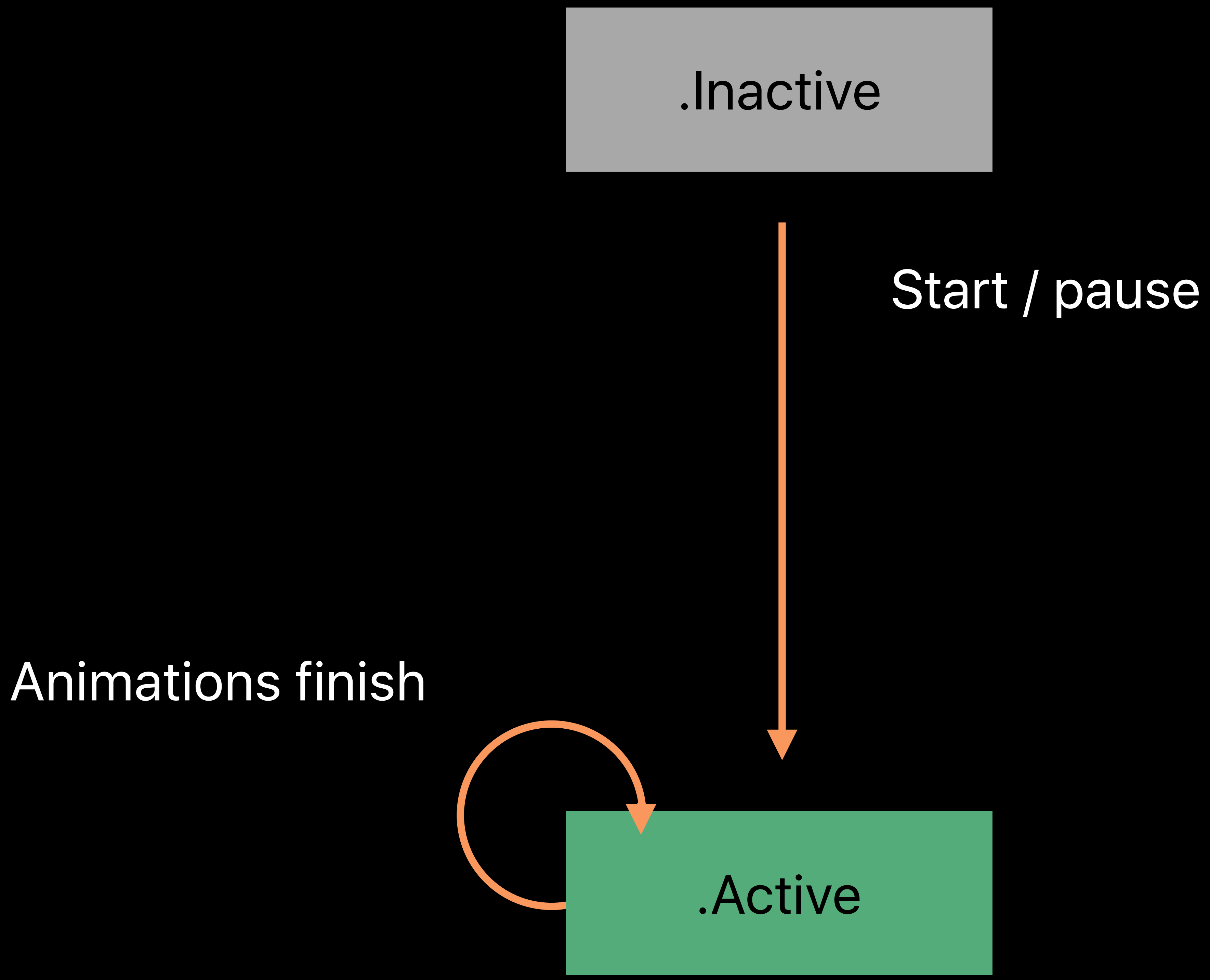
## .pausesOnCompletion

NEW

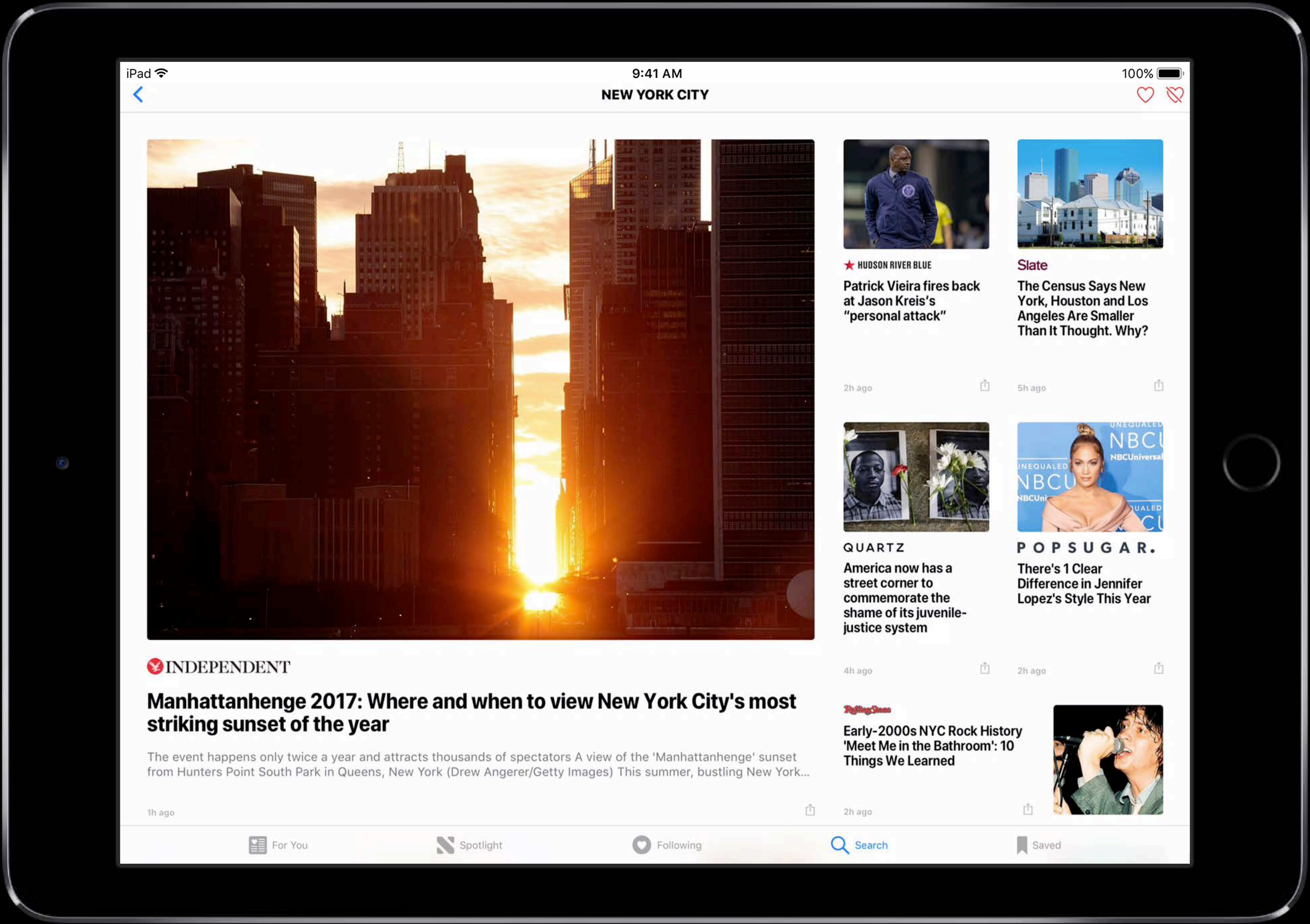


## .pausesOnCompletion

NEW



```
animator.pausesOnCompletion = true
```



iPad 9:41 AM 100%

NEW YORK CITY

INDEPENDENT

Manhattanhenge 2017: Where and when to view New York City's most striking sunset of the year

The event happens only twice a year and attracts thousands of spectators A view of the 'Manhattanhenge' sunset from Hunters Point South Park in Queens, New York (Drew Angerer/Getty Images) This summer, bustling New York...

1h ago

HUDSON RIVER BLUE

Patrick Vieira fires back at Jason Kreis's "personal attack"

2h ago

Slate

The Census Says New York, Houston and Los Angeles Are Smaller Than It Thought. Why?

5h ago

QUARTZ

America now has a street corner to commemorate the shame of its juvenile-justice system

4h ago

POPSUGAR.

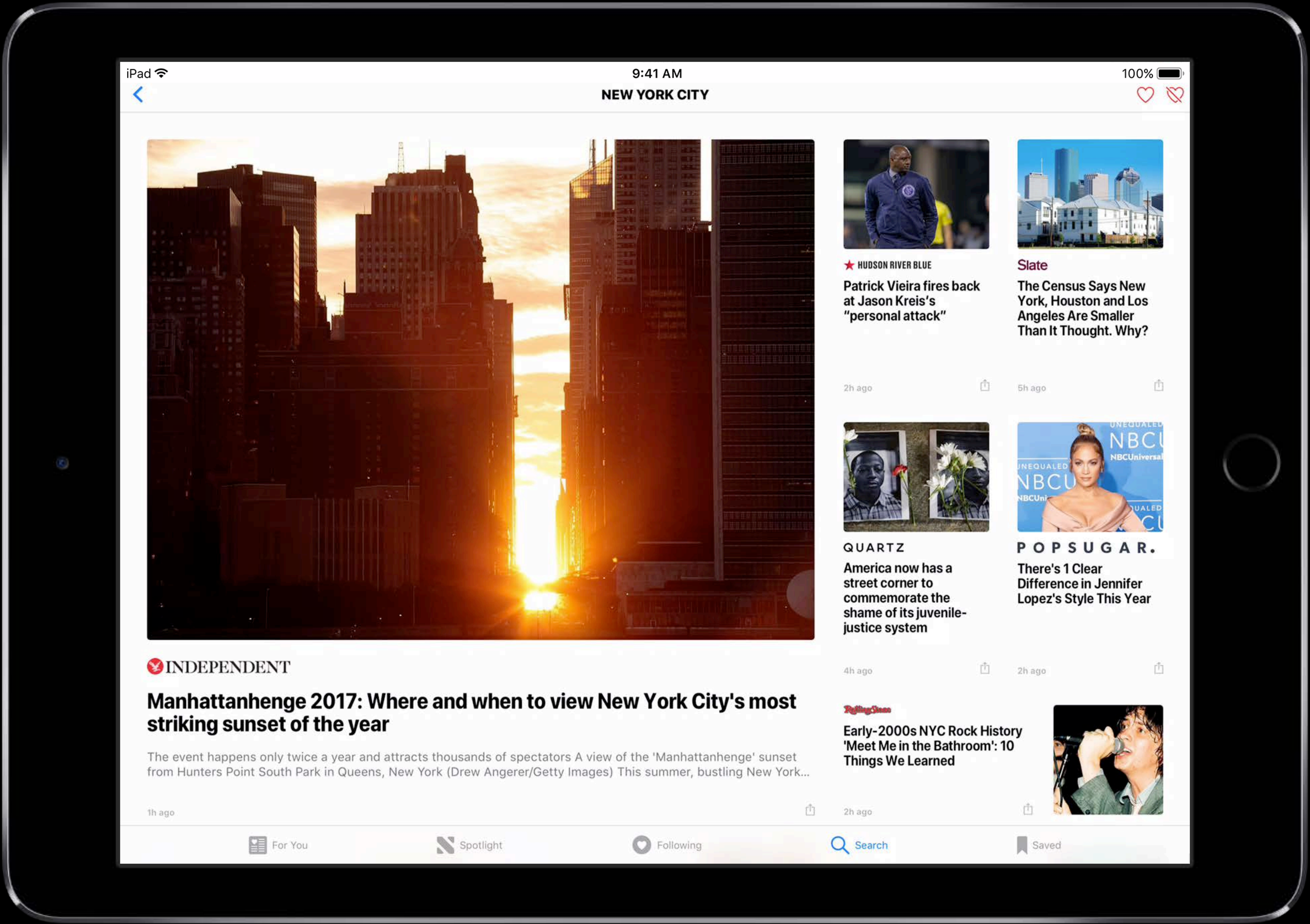
There's 1 Clear Difference in Jennifer Lopez's Style This Year

2h ago

Rolling Stone

Early-2000s NYC Rock History 'Meet Me in the Bathroom': 10 Things We Learned

2h ago



iPad 9:41 AM 100%

NEW YORK CITY

INDEPENDENT

Manhattanhenge 2017: Where and when to view New York City's most striking sunset of the year

The event happens only twice a year and attracts thousands of spectators A view of the 'Manhattanhenge' sunset from Hunters Point South Park in Queens, New York (Drew Angerer/Getty Images) This summer, bustling New York...

1h ago

HUDSON RIVER BLUE

Patrick Vieira fires back at Jason Kreis's "personal attack"

2h ago

Slate

The Census Says New York, Houston and Los Angeles Are Smaller Than It Thought. Why?

5h ago

QUARTZ

America now has a street corner to commemorate the shame of its juvenile-justice system

4h ago

POPSUGAR.

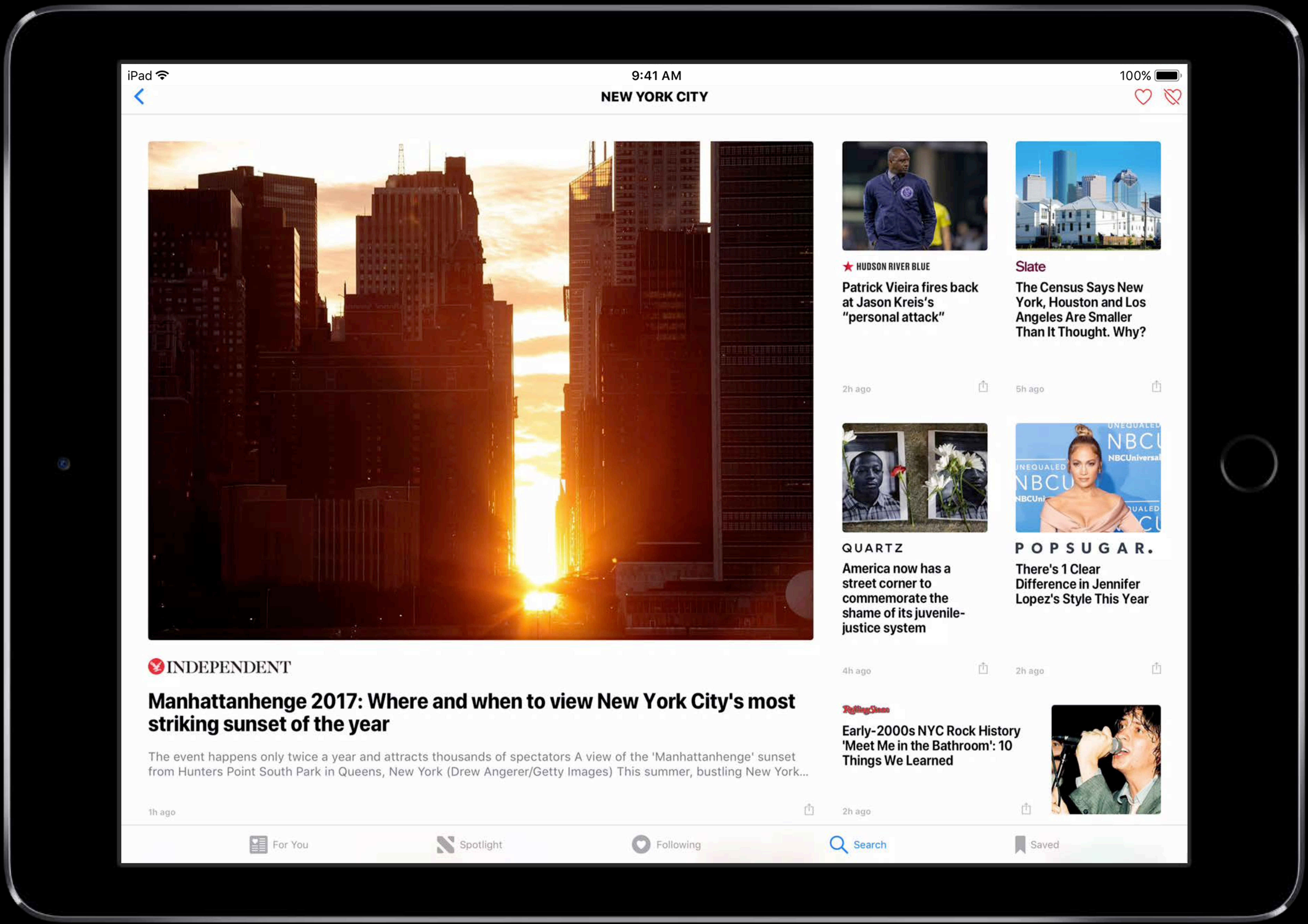
There's 1 Clear Difference in Jennifer Lopez's Style This Year

2h ago

RollingStone

Early-2000s NYC Rock History 'Meet Me in the Bathroom': 10 Things We Learned

2h ago



iPad 9:41 AM 100%

NEW YORK CITY

INDEPENDENT

**Manhattanhenge 2017: Where and when to view New York City's most striking sunset of the year**

The event happens only twice a year and attracts thousands of spectators A view of the 'Manhattanhenge' sunset from Hunters Point South Park in Queens, New York (Drew Angerer/Getty Images) This summer, bustling New York...

1h ago

HUDSON RIVER BLUE

**Patrick Vieira fires back at Jason Kreis's "personal attack"**

2h ago

Slate

**The Census Says New York, Houston and Los Angeles Are Smaller Than It Thought. Why?**

5h ago

QUARTZ

**America now has a street corner to commemorate the shame of its juvenile-justice system**

4h ago

POPSUGAR.

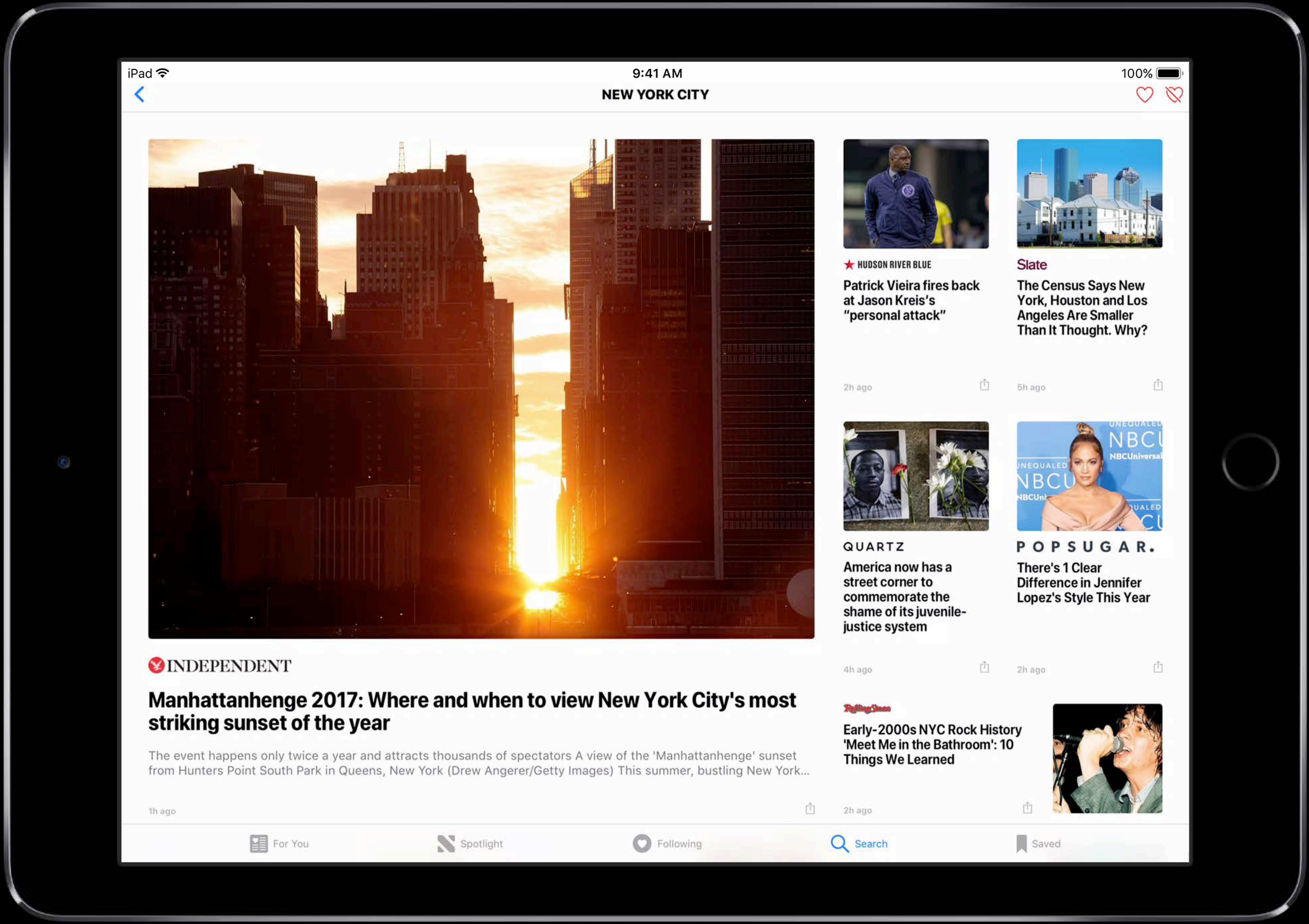
**There's 1 Clear Difference in Jennifer Lopez's Style This Year**

2h ago

RollingStone

**Early-2000s NYC Rock History 'Meet Me in the Bathroom': 10 Things We Learned**

2h ago



iPad 9:41 AM 100%

NEW YORK CITY

INDEPENDENT

**Manhattanhenge 2017: Where and when to view New York City's most striking sunset of the year**

The event happens only twice a year and attracts thousands of spectators A view of the 'Manhattanhenge' sunset from Hunters Point South Park in Queens, New York (Drew Angerer/Getty Images) This summer, bustling New York...

1h ago

HUDSON RIVER BLUE

**Patrick Vieira fires back at Jason Kreis's "personal attack"**

2h ago

Slate

**The Census Says New York, Houston and Los Angeles Are Smaller Than It Thought. Why?**

5h ago

QUARTZ

**America now has a street corner to commemorate the shame of its juvenile-justice system**

4h ago

POPSUGAR.

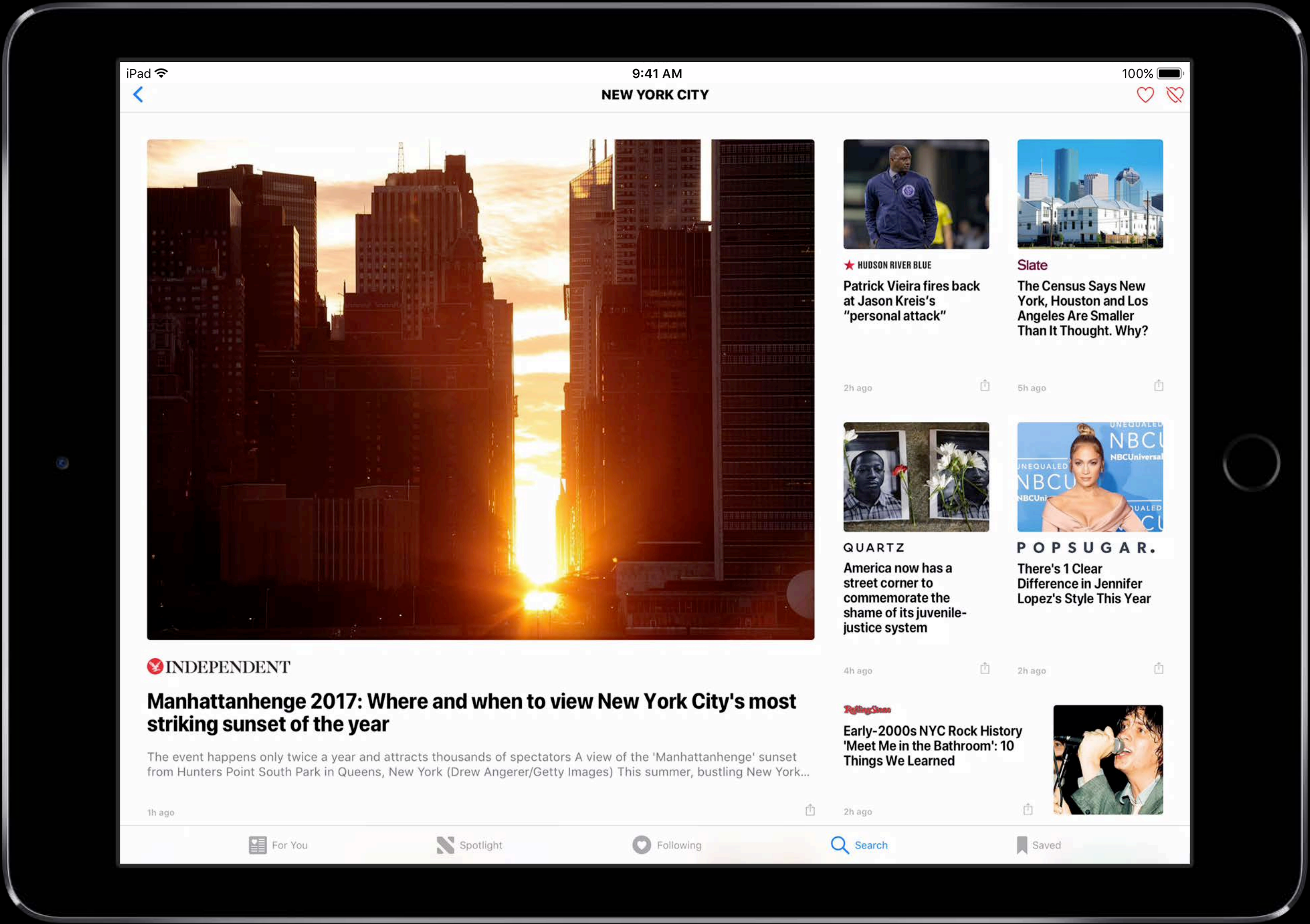
**There's 1 Clear Difference in Jennifer Lopez's Style This Year**

2h ago

Rolling Stone

**Early-2000s NYC Rock History 'Meet Me in the Bathroom': 10 Things We Learned**

2h ago



iPad 9:41 AM 100%

NEW YORK CITY

INDEPENDENT

Manhattanhenge 2017: Where and when to view New York City's most striking sunset of the year

The event happens only twice a year and attracts thousands of spectators A view of the 'Manhattanhenge' sunset from Hunters Point South Park in Queens, New York (Drew Angerer/Getty Images) This summer, bustling New York...

1h ago

HUDSON RIVER BLUE

Patrick Vieira fires back at Jason Kreis's "personal attack"

2h ago

Slate

The Census Says New York, Houston and Los Angeles Are Smaller Than It Thought. Why?

5h ago

QUARTZ

America now has a street corner to commemorate the shame of its juvenile-justice system

4h ago

POPSUGAR.

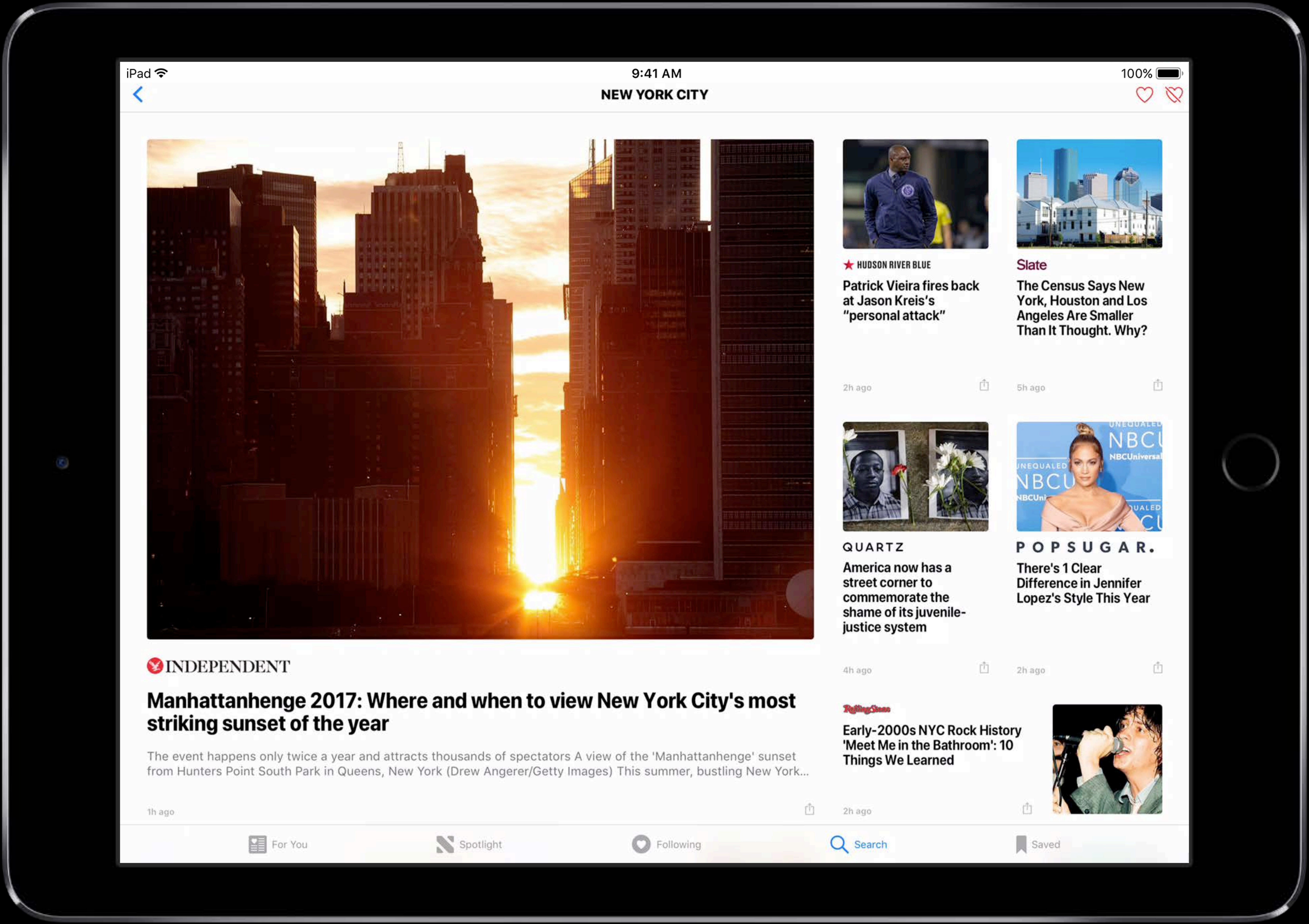
There's 1 Clear Difference in Jennifer Lopez's Style This Year

2h ago

Rolling Stone

Early-2000s NYC Rock History 'Meet Me in the Bathroom': 10 Things We Learned

2h ago



iPad 9:41 AM 100%

NEW YORK CITY

INDEPENDENT

Manhattanhenge 2017: Where and when to view New York City's most striking sunset of the year

The event happens only twice a year and attracts thousands of spectators A view of the 'Manhattanhenge' sunset from Hunters Point South Park in Queens, New York (Drew Angerer/Getty Images) This summer, bustling New York...

1h ago

HUDSON RIVER BLUE

Patrick Vieira fires back at Jason Kreis's "personal attack"

2h ago

Slate

The Census Says New York, Houston and Los Angeles Are Smaller Than It Thought. Why?

5h ago

QUARTZ

America now has a street corner to commemorate the shame of its juvenile-justice system

4h ago

POPSUGAR.

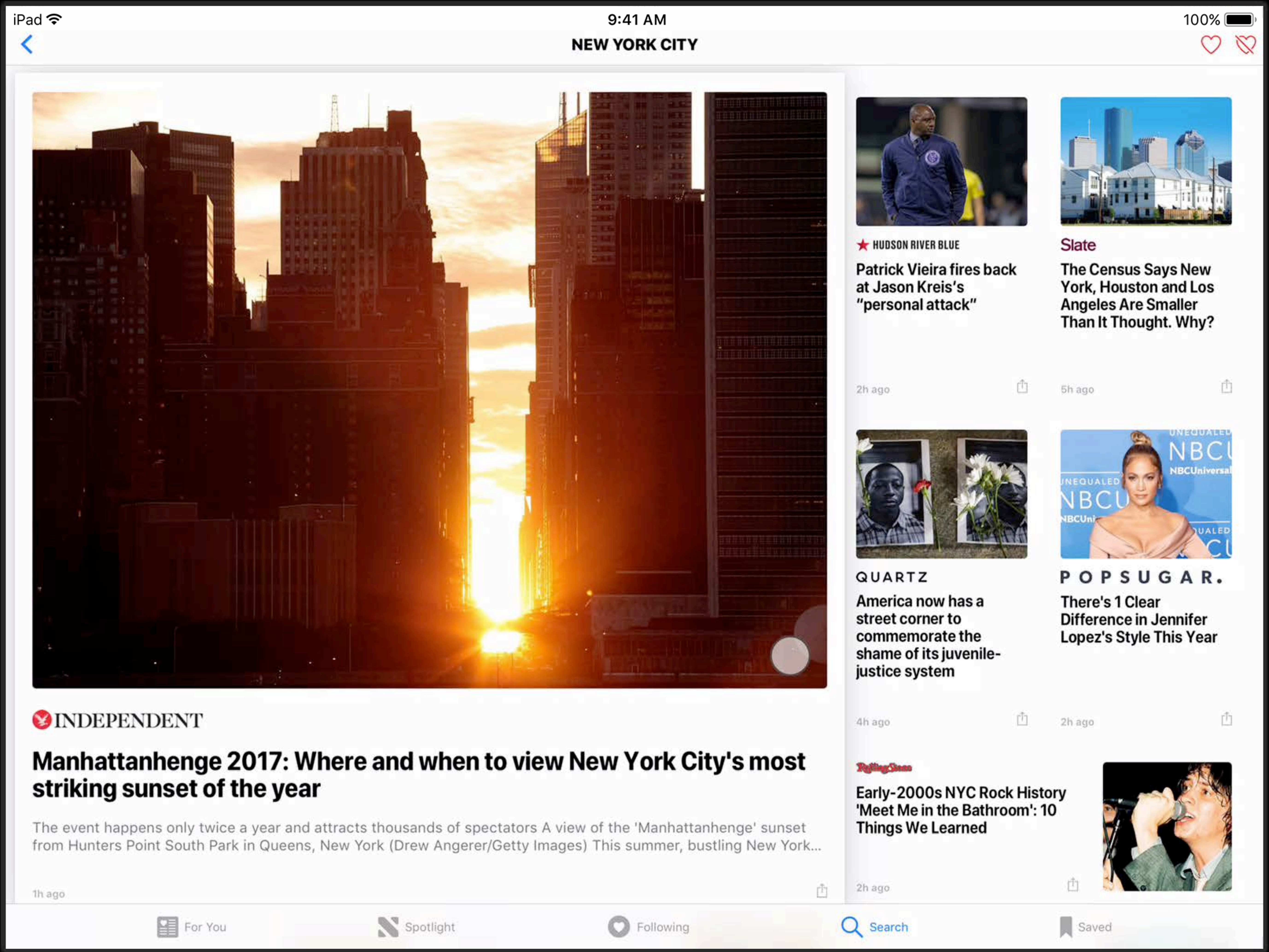
There's 1 Clear Difference in Jennifer Lopez's Style This Year

2h ago

RollingStone

Early-2000s NYC Rock History 'Meet Me in the Bathroom': 10 Things We Learned

2h ago



iPad 9:41 AM 100%

NEW YORK CITY

INDEPENDENT

Manhattanhenge 2017: Where and when to view New York City's most striking sunset of the year

The event happens only twice a year and attracts thousands of spectators A view of the 'Manhattanhenge' sunset from Hunters Point South Park in Queens, New York (Drew Angerer/Getty Images) This summer, bustling New York...

1h ago

HUDSON RIVER BLUE

Patrick Vieira fires back at Jason Kreis's "personal attack"

2h ago

Slate

The Census Says New York, Houston and Los Angeles Are Smaller Than It Thought. Why?

5h ago

QUARTZ

America now has a street corner to commemorate the shame of its juvenile-justice system

4h ago

POPSUGAR.

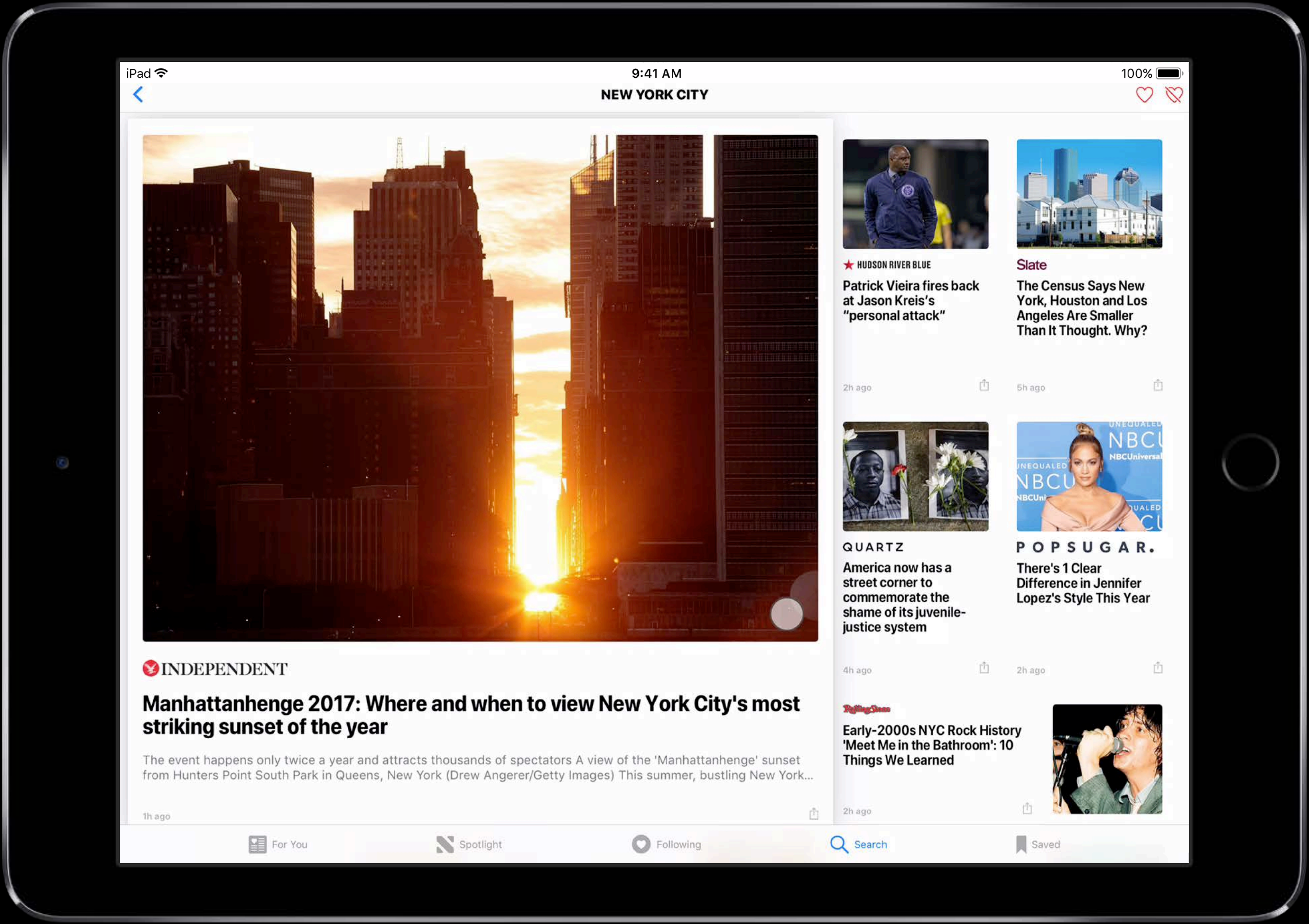
There's 1 Clear Difference in Jennifer Lopez's Style This Year

2h ago

Rolling Stone

Early-2000s NYC Rock History 'Meet Me in the Bathroom': 10 Things We Learned

2h ago



iPad 9:41 AM 100%

NEW YORK CITY

INDEPENDENT

**Manhattanhenge 2017: Where and when to view New York City's most striking sunset of the year**

The event happens only twice a year and attracts thousands of spectators A view of the 'Manhattanhenge' sunset from Hunters Point South Park in Queens, New York (Drew Angerer/Getty Images) This summer, bustling New York...

1h ago

HUDSON RIVER BLUE

**Patrick Vieira fires back at Jason Kreis's "personal attack"**

2h ago

Slate

**The Census Says New York, Houston and Los Angeles Are Smaller Than It Thought. Why?**

5h ago

QUARTZ

**America now has a street corner to commemorate the shame of its juvenile-justice system**

4h ago

POPSUGAR.

**There's 1 Clear Difference in Jennifer Lopez's Style This Year**

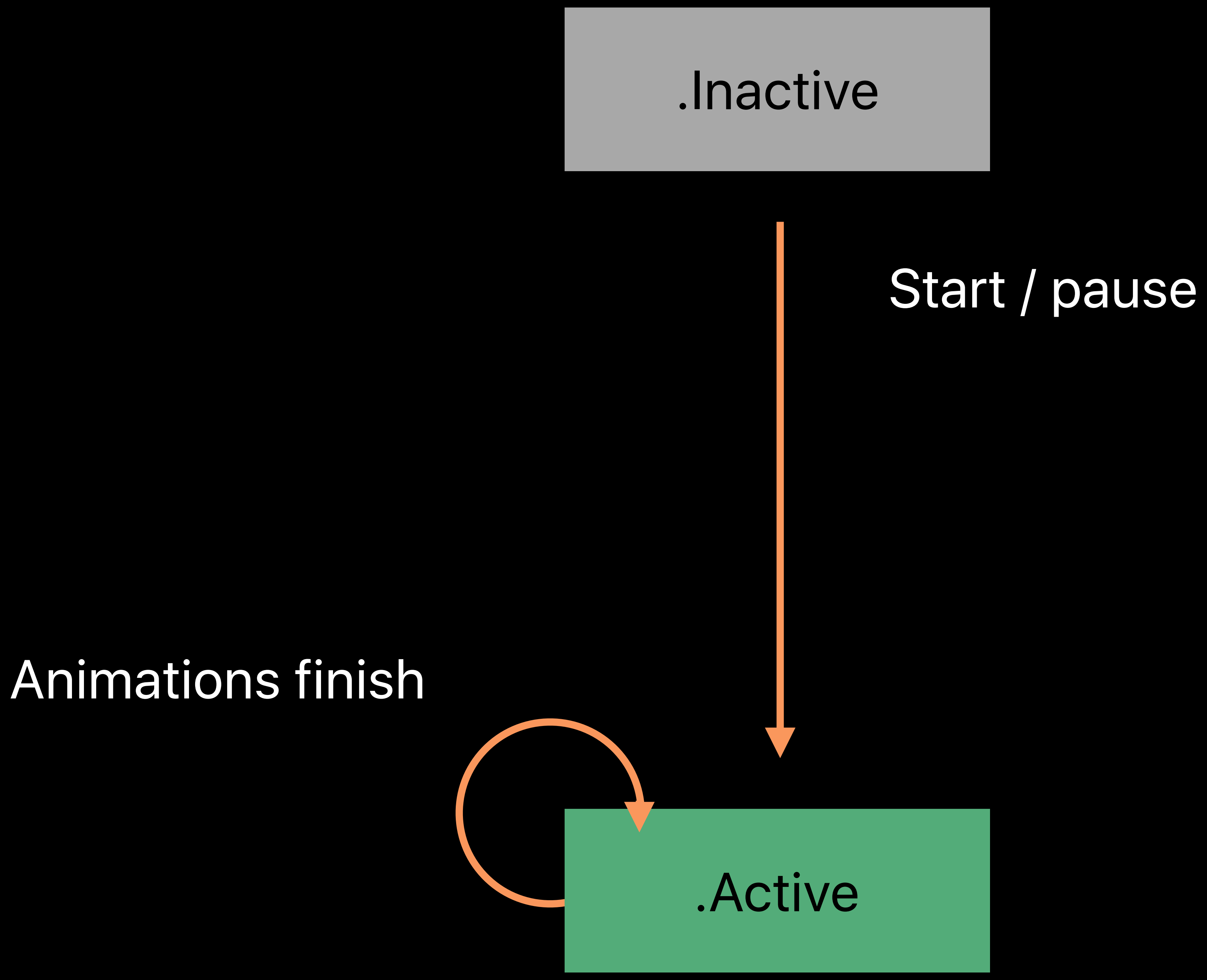
2h ago

Rolling Stone

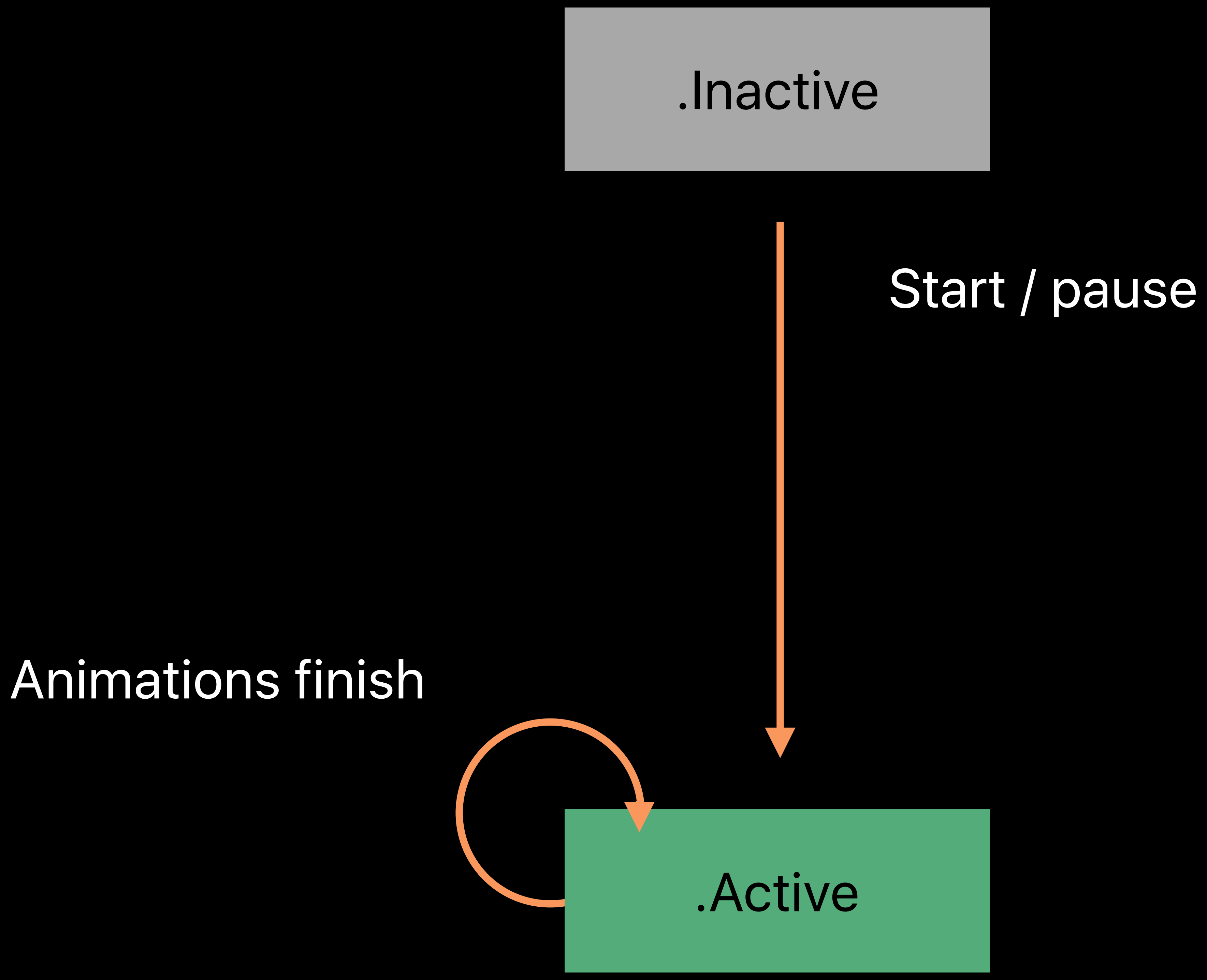
**Early-2000s NYC Rock History 'Meet Me in the Bathroom': 10 Things We Learned**

2h ago

## .pausesOnCompletion



## .pausesOnCompletion



```
animator.addObserver(self, forKeyPath: "running", options: [.new], context: nil)
```

## Starting as Paused

NEW

```swift
let animator = UIViewPropertyAnimator(duration: 1, curve: .easeIn)
animator.startAnimation()
// ...
animator.addAnimations {
    // will run immediately
    circle.frame = circle.frame.offsetBy(dx: 100, dy: 0)
}
```

No escaping for animation blocks

## Springs

## Spring Animations

Critically damped spring

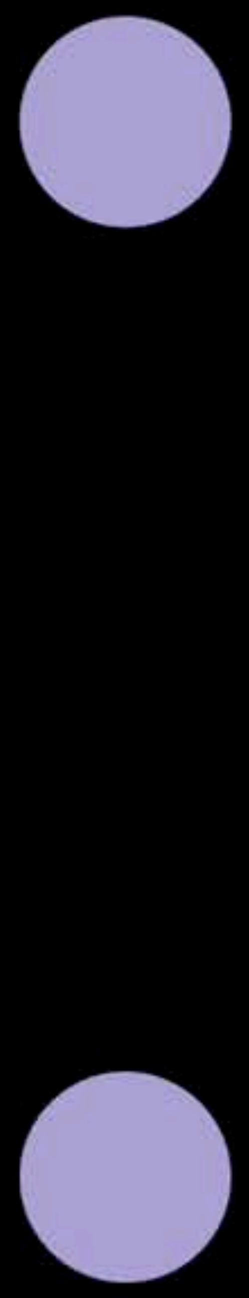
Under damped spring

## Spring Animations

## Spring Animations

Critically damped spring

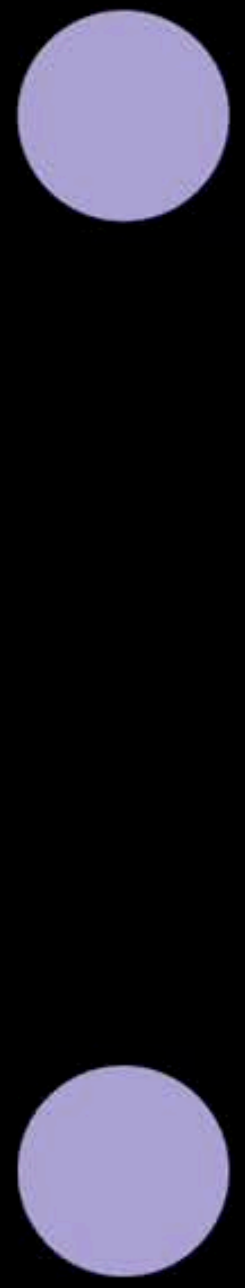
Under damped spring



## Spring Animations

Critically damped spring

Under damped spring



## Spring Animations

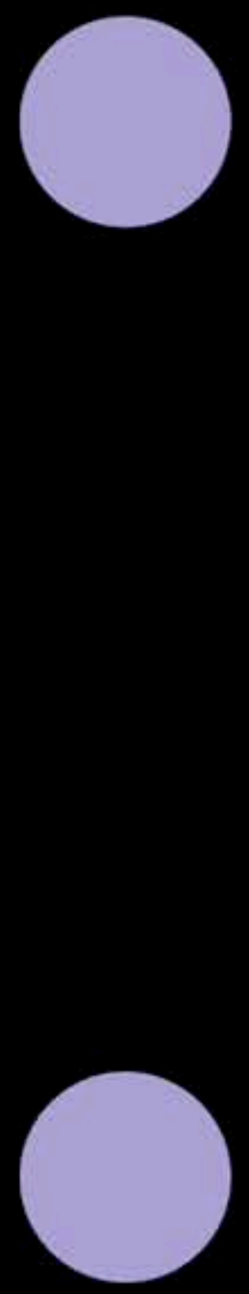
Critically damped spring

Under damped spring

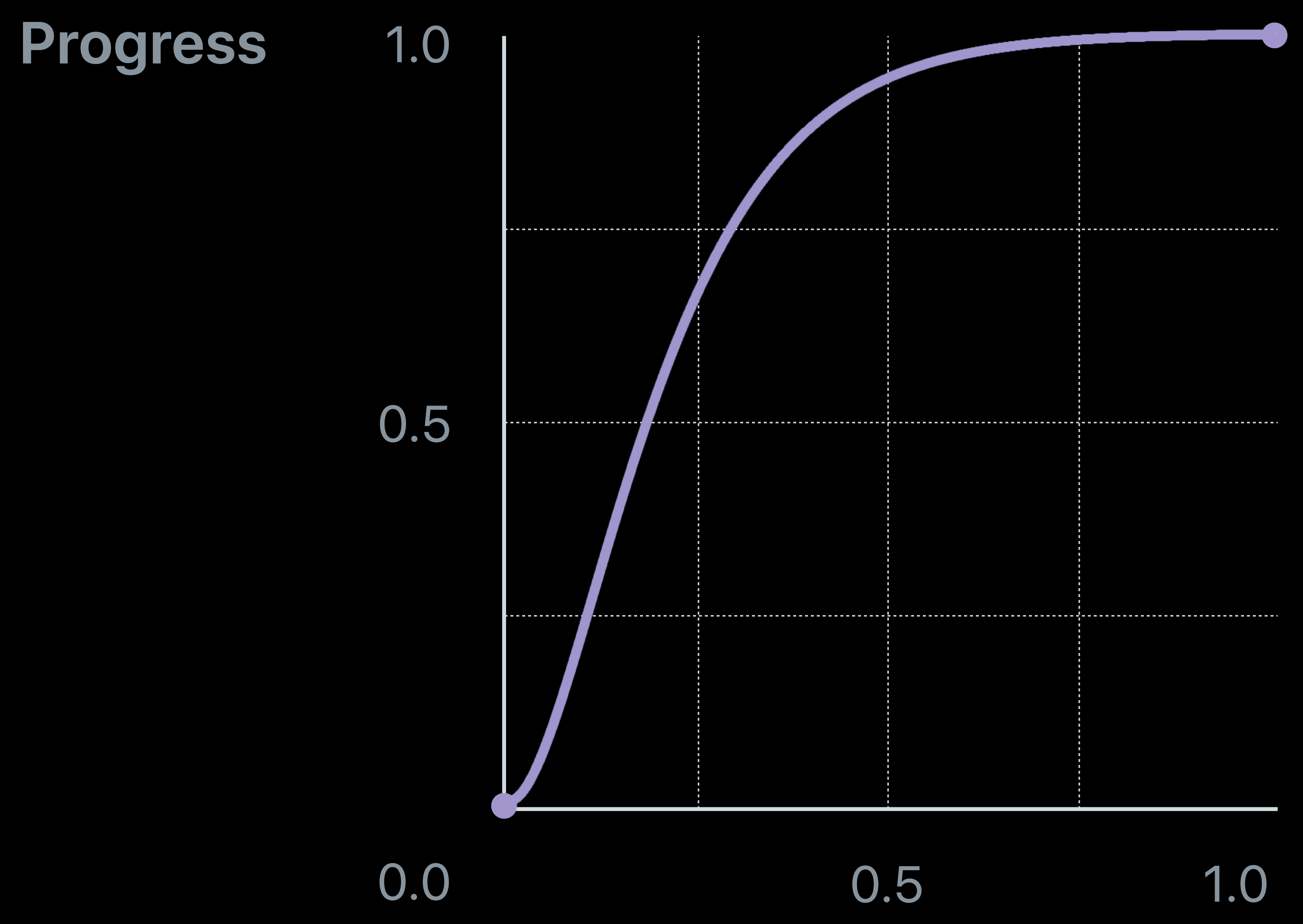
## Spring Animations

Critically damped spring

Under damped spring

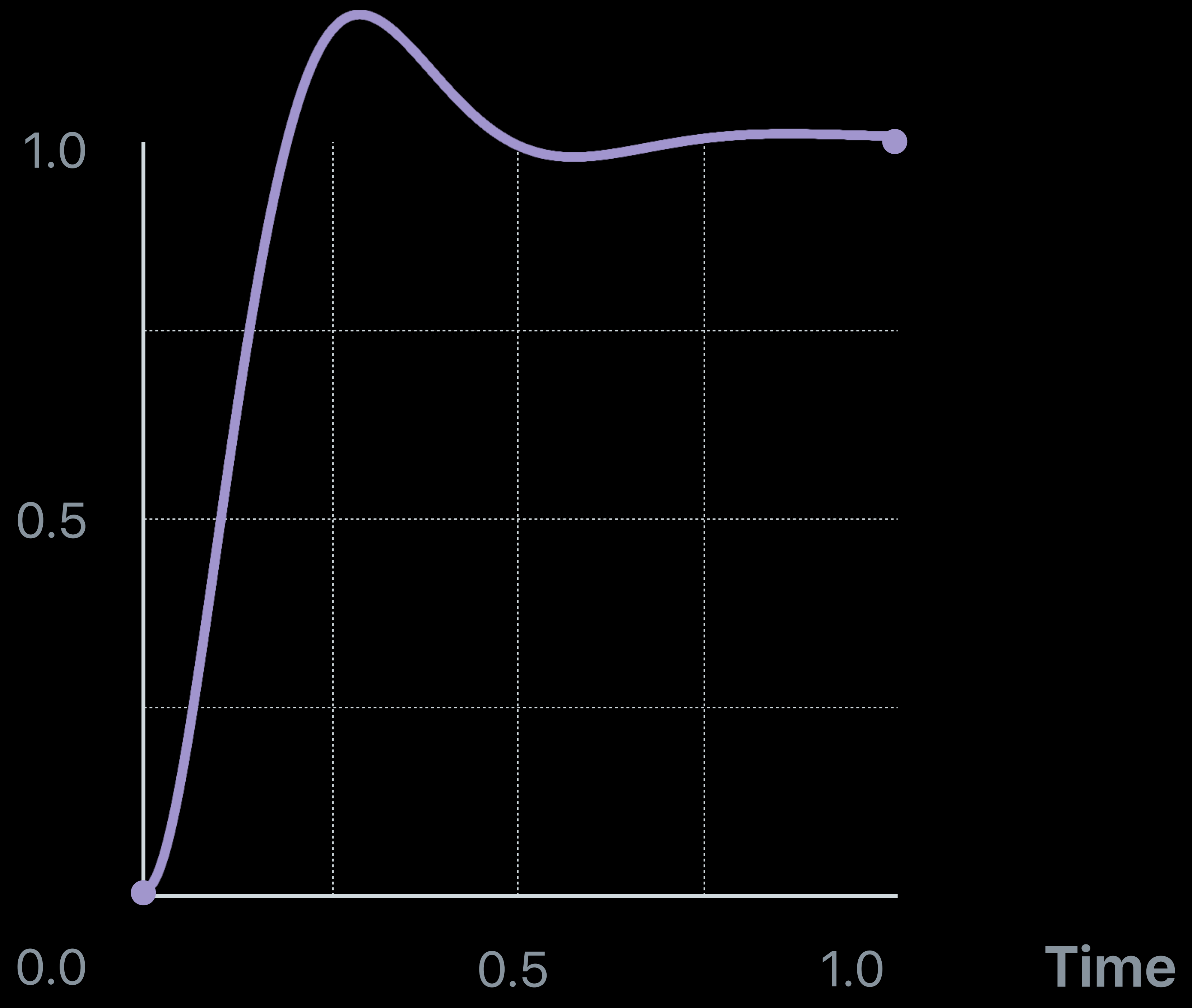


# Spring Animations



Critically damped spring

Damping ratio = 1.0



Under damped spring

Damping ratio < 1.0

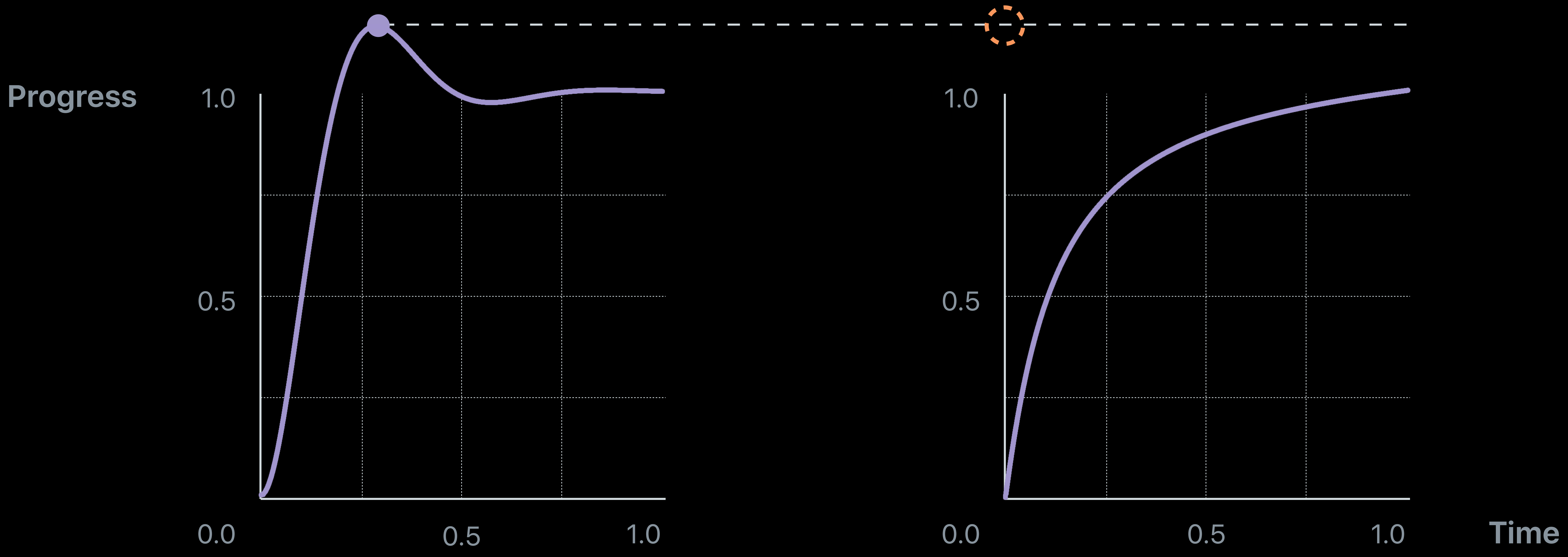
## Spring Animations

Why they always animate from current state

# Spring Animations

Why they always animate from current state

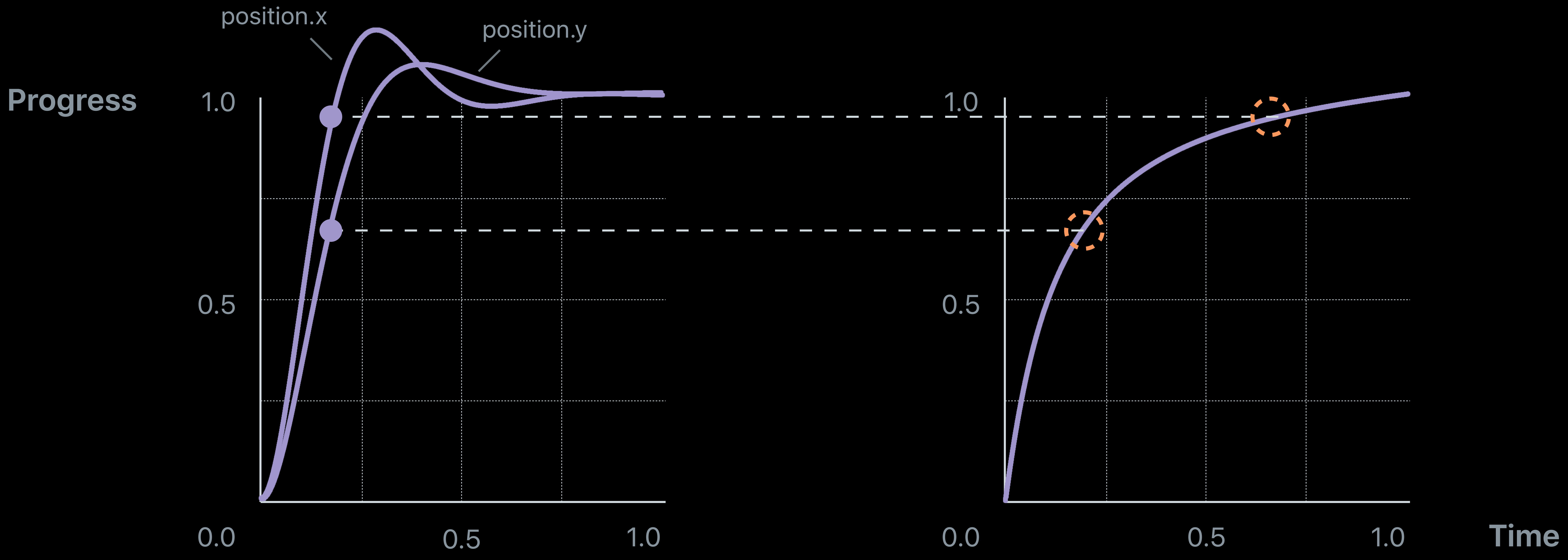
Remapping onto cubic may be undefined



# Spring Animations

Why they always animate from current state

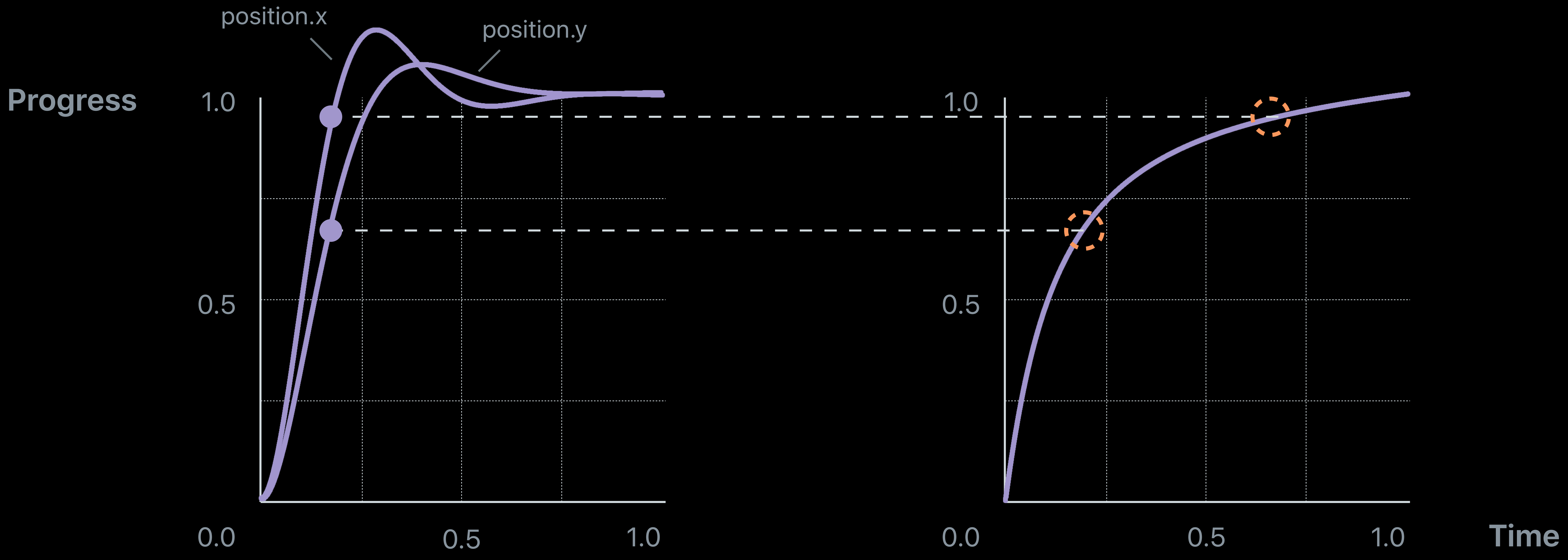
2D velocity desynchronization



# Spring Animations

Why they always animate from current state

## 2D velocity desynchronization



## Best Practices When Interrupting Springs

## Best Practices When Interrupting Springs

Stop and create a new property animator

## Best Practices When Interrupting Springs

Stop and create a new property animator

Use critically damped spring without velocity

## Best Practices When Interrupting Springs

Stop and create a new property animator

Use critically damped spring without velocity

Decompose component velocity with multiple animators

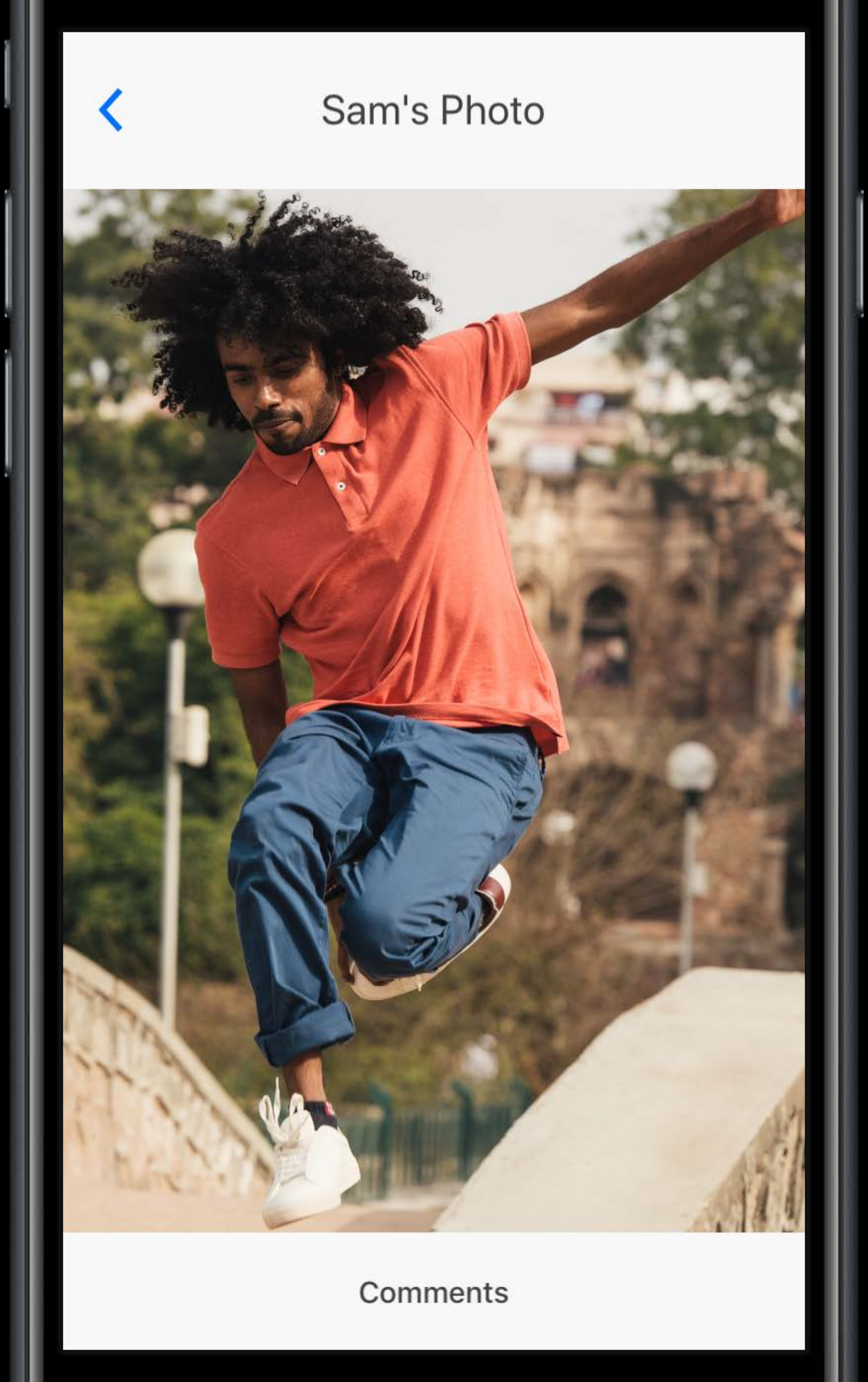
## Coordinating Animations

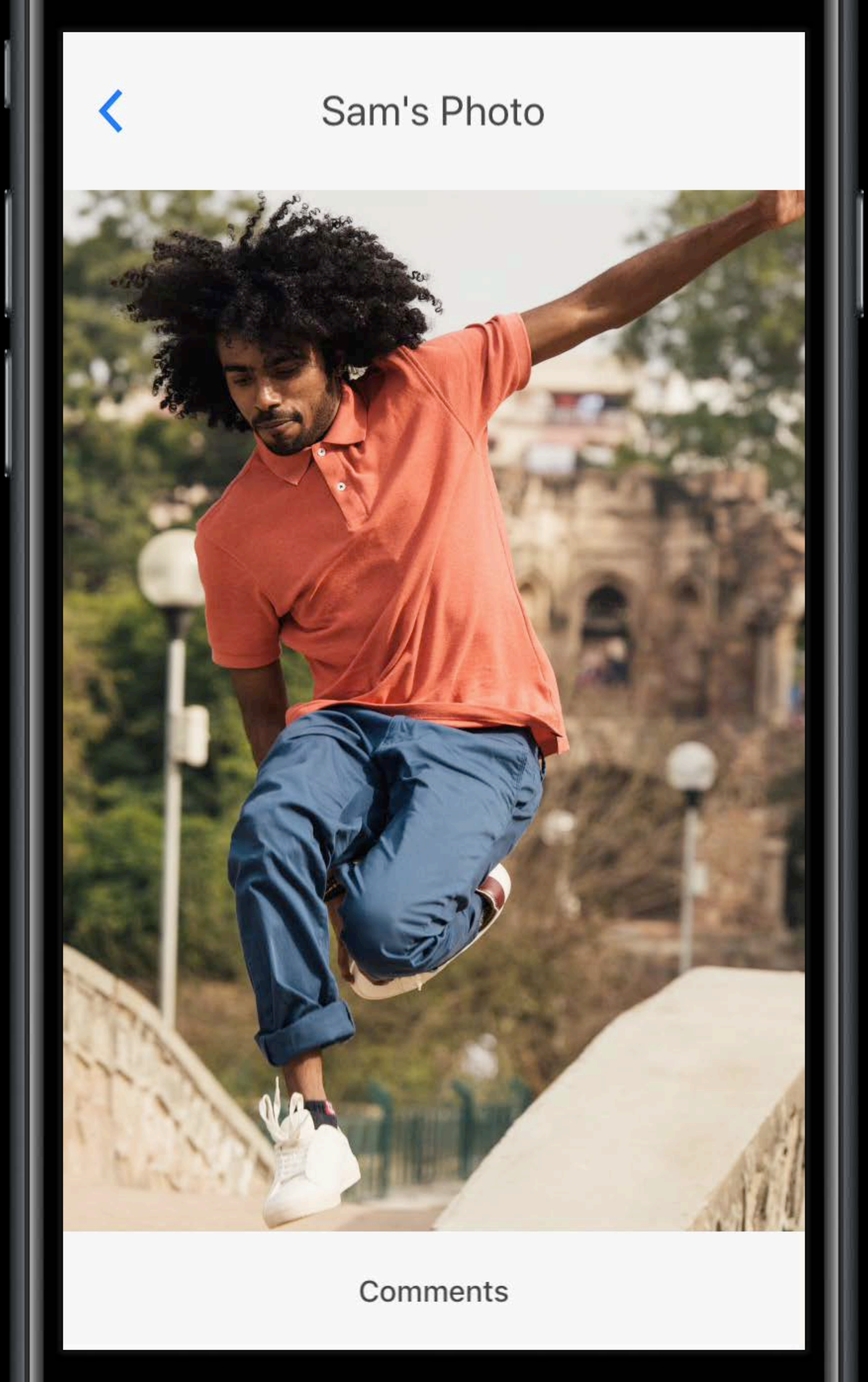
## Overview

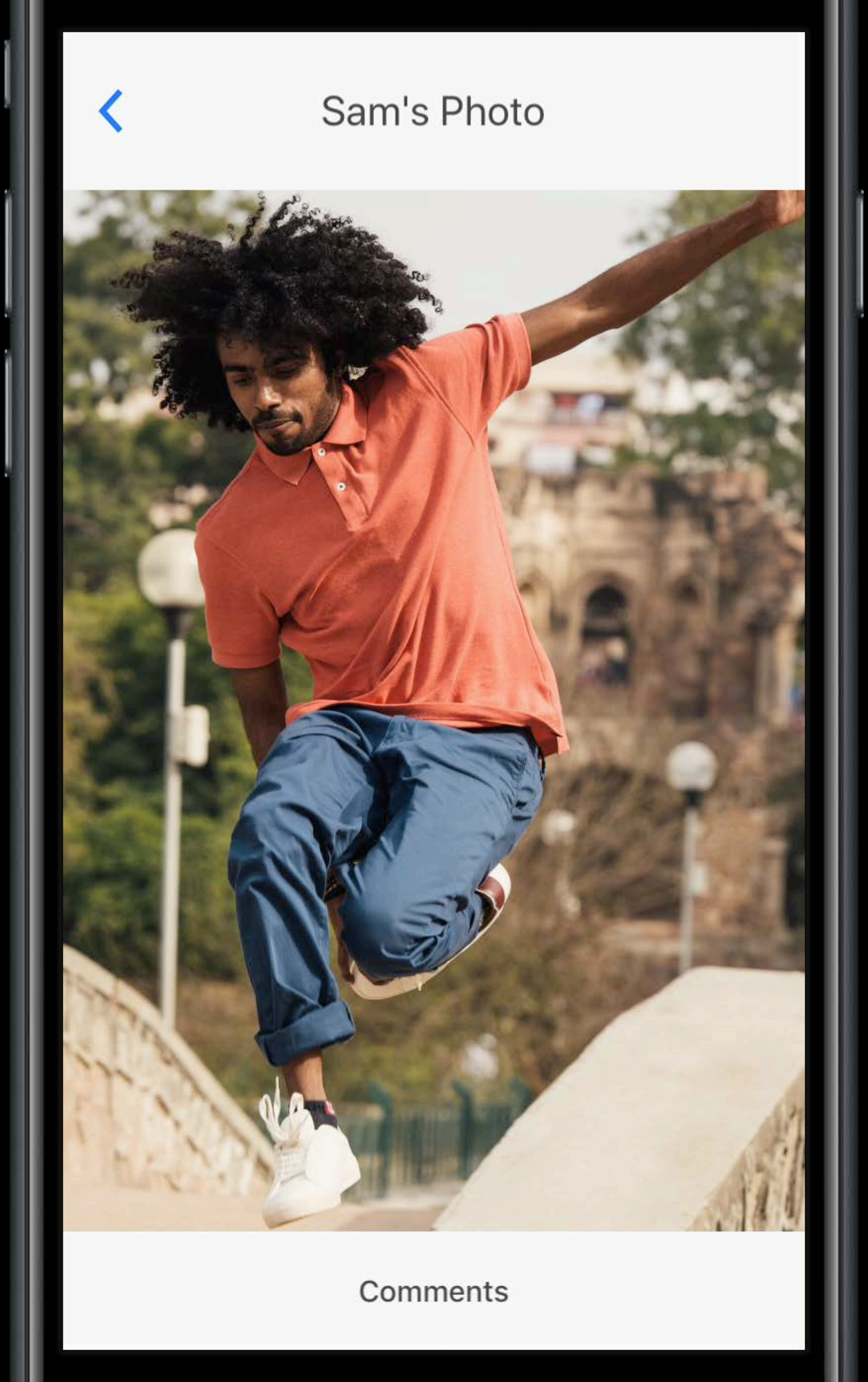
Build a fully interactive, interruptible animated transition

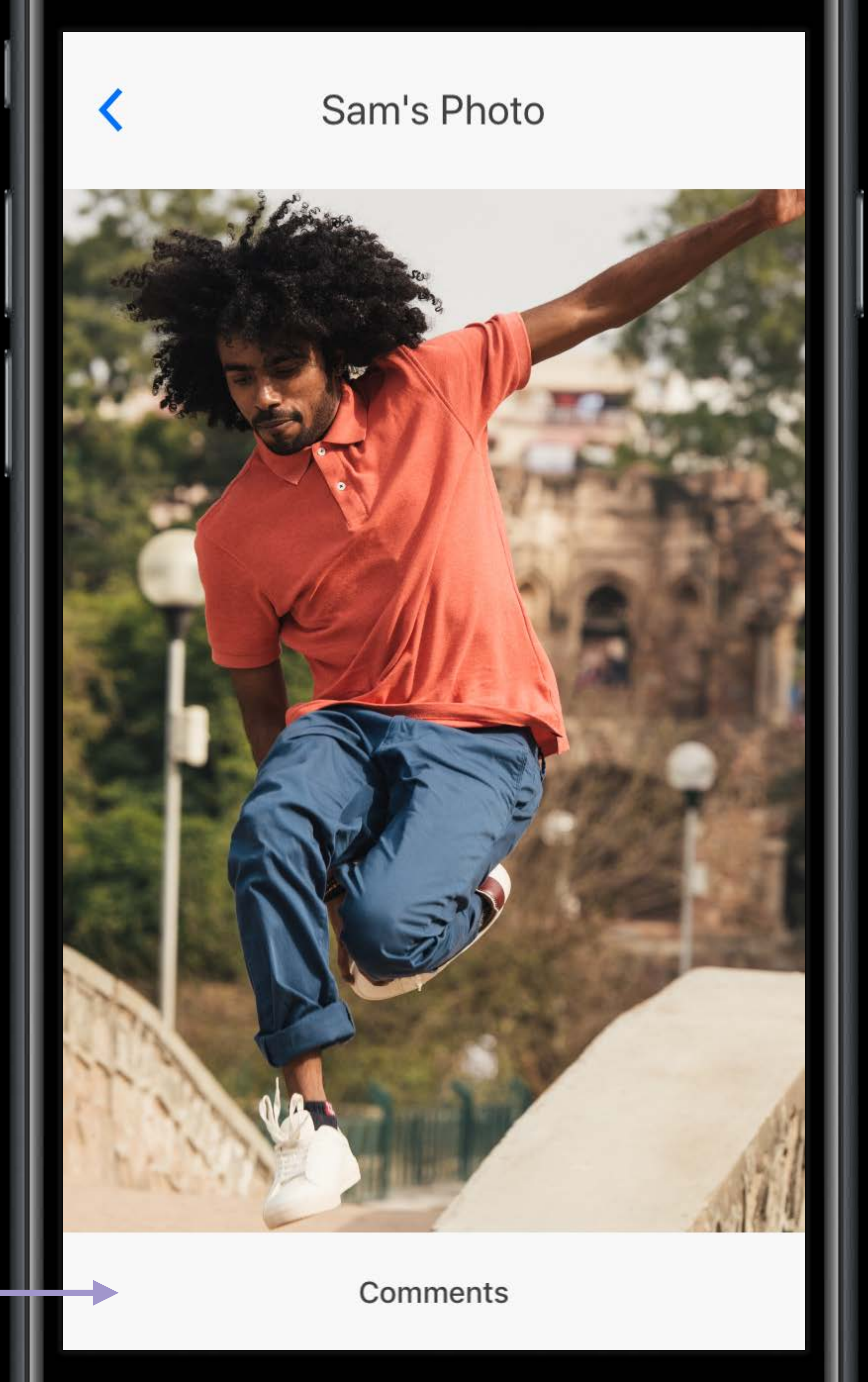
Coordinate across multiple uniquely timed animators

Sam's Photo


Sam's Photo
Comments


Sam's Photo
Comments


Sam's Photo
Comments



```
UITapGestureRecognizer
UIPanGestureRecognizer
```

```swift
// Tracks all running animators
var runningAnimators = [UIViewPropertyAnimator]()

// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary or reverses it on tap
func animateOrReverseRunningTransition(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary and pauses on pan .begin
func startInteractiveTransition(state: State, duration: TimeInterval) { ... }

// Scrubs transition on pan .changed
func updateInteractiveTransition(fractionComplete: CGFloat) { ... }

// Continues or reverse transition on pan .ended
func continueInteractiveTransition(cancel: Bool) { ... }
```

```swift
// Tracks all running animators
var runningAnimators = [UIViewPropertyAnimator]()

// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary or reverses it on tap
func animateOrReverseRunningTransition(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary and pauses on pan .begin
func startInteractiveTransition(state: State, duration: TimeInterval) { ... }

// Scrubs transition on pan .changed
func updateInteractiveTransition(fractionComplete: CGFloat) { ... }

// Continues or reverse transition on pan .ended
func continueInteractiveTransition(cancel: Bool) { ... }
```

```swift
// Tracks all running animators
var runningAnimators = [UIViewPropertyAnimator]()

// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary or reverses it on tap
func animateOrReverseRunningTransition(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary and pauses on pan .begin
func startInteractiveTransition(state: State, duration: TimeInterval) { ... }

// Scrubs transition on pan .changed
func updateInteractiveTransition(fractionComplete: CGFloat) { ... }

// Continues or reverse transition on pan .ended
func continueInteractiveTransition(cancel: Bool) { ... }
```

```swift
// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) {
```

```
// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) {
   if runningAnimators.isEmpty {
        let frameAnimator = UIViewPropertyAnimator(duration: duration, dampingRatio: 1) {
            switch state {
            case .Expanded:
                self.control.frame = CGRect(...)
            case .Collapsed:
                self.control.frame = CGRect(...)
            }
        }
        frameAnimator.startAnimation()
        runningAnimators.append(frameAnimator)
    }
}
```

```swift
// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) {
    if runningAnimators.isEmpty {
        let frameAnimator = UIViewPropertyAnimator(duration: duration, dampingRatio: 1) {
            switch state {
            case .Expanded:
                self.control.frame = CGRect(...)
            case .Collapsed:
                self.control.frame = CGRect(...)
            }
        }
        frameAnimator.startAnimation()
        runningAnimators.append(frameAnimator)
    }
}
```

```swift
// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) {
   if runningAnimators.isEmpty {
        let frameAnimator = UIViewPropertyAnimator(duration: duration, dampingRatio: 1) {
            switch state {
            case .Expanded:
                self.control.frame = CGRect(...)
            case .Collapsed:
                self.control.frame = CGRect(...)
            }
        }
        frameAnimator.startAnimation()
        runningAnimators.append(frameAnimator)
    }
}
```

```swift
// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) {
   if runningAnimators.isEmpty {
        let frameAnimator = UIViewPropertyAnimator(duration: duration, dampingRatio: 1) {
            switch state {
            case .Expanded:
                self.control.frame = CGRect(...)
            case .Collapsed:
                self.control.frame = CGRect(...)
            }
        }
        frameAnimator.startAnimation()
        runningAnimators.append(frameAnimator)
    }
}
```

```swift
// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) {
   if runningAnimators.isEmpty {
        let frameAnimator = UIViewPropertyAnimator(duration: duration, dampingRatio: 1) {
            switch state {
            case .Expanded:
                self.control.frame = CGRect(...)
            case .Collapsed:
                self.control.frame = CGRect(...)
            }
        }
        frameAnimator.startAnimation()
        runningAnimators.append(frameAnimator)
    }
}
```

```swift
// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) {
```

```swift
// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) {
```

```swift
// Tracks all running animators
var runningAnimators = [UIViewPropertyAnimator]()

// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary or reverses it on tap
func animateOrReverseRunningTransition(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary and pauses on pan .begin
func startInteractiveTransition(state: State, duration: TimeInterval) { ... }

// Scrubs transition on pan .changed
func updateInteractiveTransition(fractionComplete: CGFloat) { ... }

// Continues or reverse transition on pan .ended
func continueInteractiveTransition(cancel: Bool) { ... }
```

```swift
// Tracks all running animators
var runningAnimators = [UIViewPropertyAnimator]()

// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary or reverses it on tap
func animateOrReverseRunningTransition(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary and pauses on pan .begin
func startInteractiveTransition(state: State, duration: TimeInterval) { ... }

// Scrubs transition on pan .changed
func updateInteractiveTransition(fractionComplete: CGFloat) { ... }

// Continues or reverse transition on pan .ended
func continueInteractiveTransition(cancel: Bool) { ... }
```

```swift
// Starts transition if necessary or reverses it on tap
func animateOrReverseRunningTransition(state: State, duration: TimeInterval) {
```

```swift
// Starts transition if necessary or reverses it on tap
func animateOrReverseRunningTransition(state: State, duration: TimeInterval) {
   if runningAnimators.isEmpty {
        animateTransitionIfNeeded(state: state, duration: duration)
    } else {
        for animator in runningAnimators {
            animator.isReversed = !animator.isReversed
        }
    }
}
```

```swift
// Starts transition if necessary or reverses it on tap
func animateOrReverseRunningTransition(state: State, duration: TimeInterval) {
    if runningAnimators.isEmpty {
        animateTransitionIfNeeded(state: state, duration: duration)
    } else {
        for animator in runningAnimators {
            animator.isReversed = !animator.isReversed
        }
    }
}
```

```swift
// Starts transition if necessary or reverses it on tap
func animateOrReverseRunningTransition(state: State, duration: TimeInterval) {
   if runningAnimators.isEmpty {
        animateTransitionIfNeeded(state: state, duration: duration)
    } else {
        for animator in runningAnimators {
            animator.isReversed = !animator.isReversed
        }
    }
}
```

```swift
// Starts transition if necessary or reverses it on tap
func animateOrReverseRunningTransition(state: State, duration: TimeInterval) {
```

```
// Starts transition if necessary or reverses it on tap
func animateOrReverseRunningTransition(state: State, duration: TimeInterval) {
```

```swift
// Tracks all running animators
var runningAnimators = [UIViewPropertyAnimator]()

// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary or reverses it on tap
func animateOrReverseRunningTransition(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary and pauses on pan .begin
func startInteractiveTransition(state: State, duration: TimeInterval) { ... }

// Scrubs transition on pan .changed
func updateInteractiveTransition(fractionComplete: CGFloat) { ... }

// Continues or reverse transition on pan .ended
func continueInteractiveTransition(cancel: Bool) { ... }
```

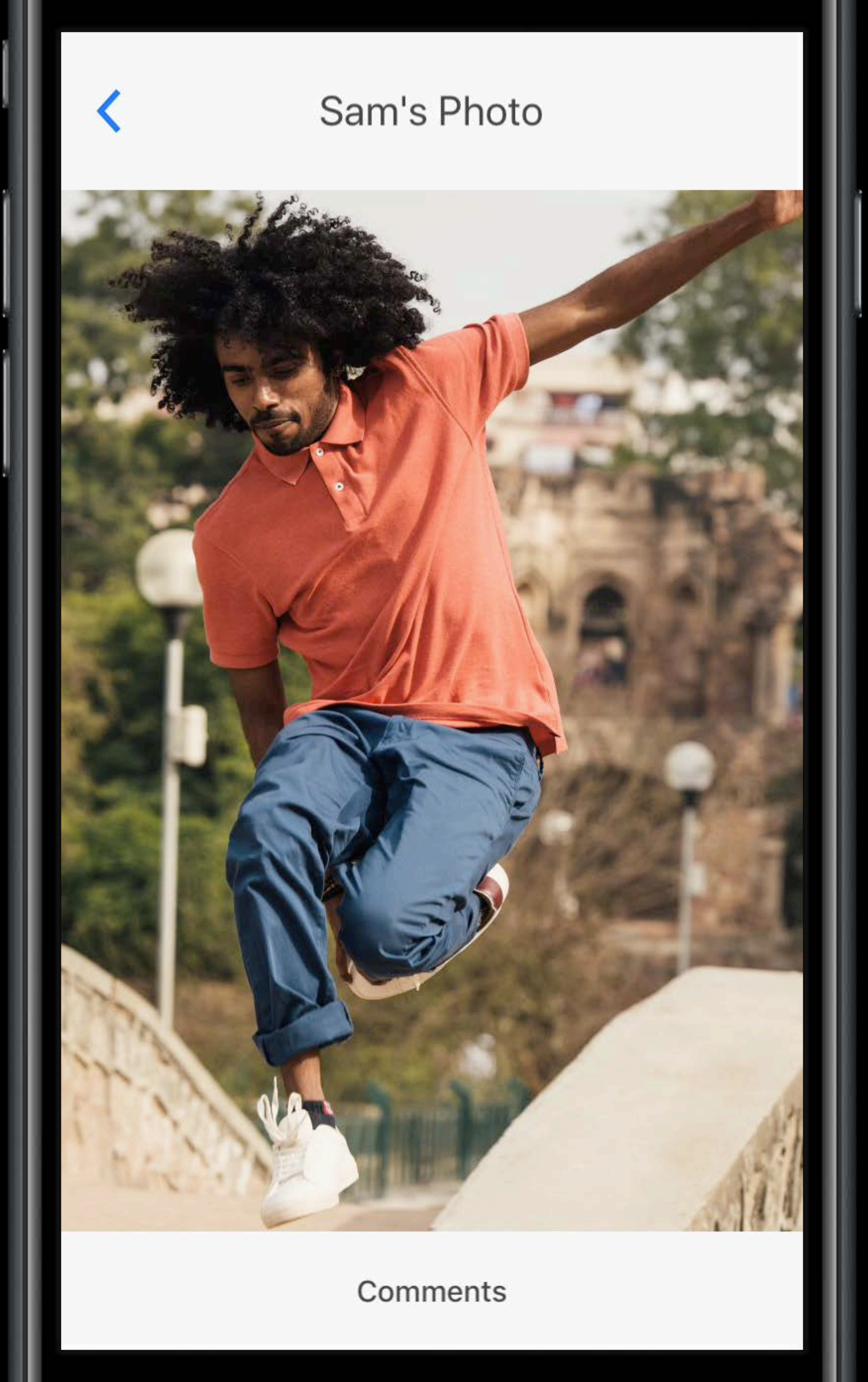
```swift
// Tracks all running animators
var runningAnimators = [UIViewPropertyAnimator]()

// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary or reverses it on tap
func animateOrReverseRunningTransition(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary and pauses on pan .begin
func startInteractiveTransition(state: State, duration: TimeInterval) { ... }

// Scrubs transition on pan .changed
func updateInteractiveTransition(fractionComplete: CGFloat) { ... }

// Continues or reverse transition on pan .ended
func continueInteractiveTransition(cancel: Bool) { ... }
```

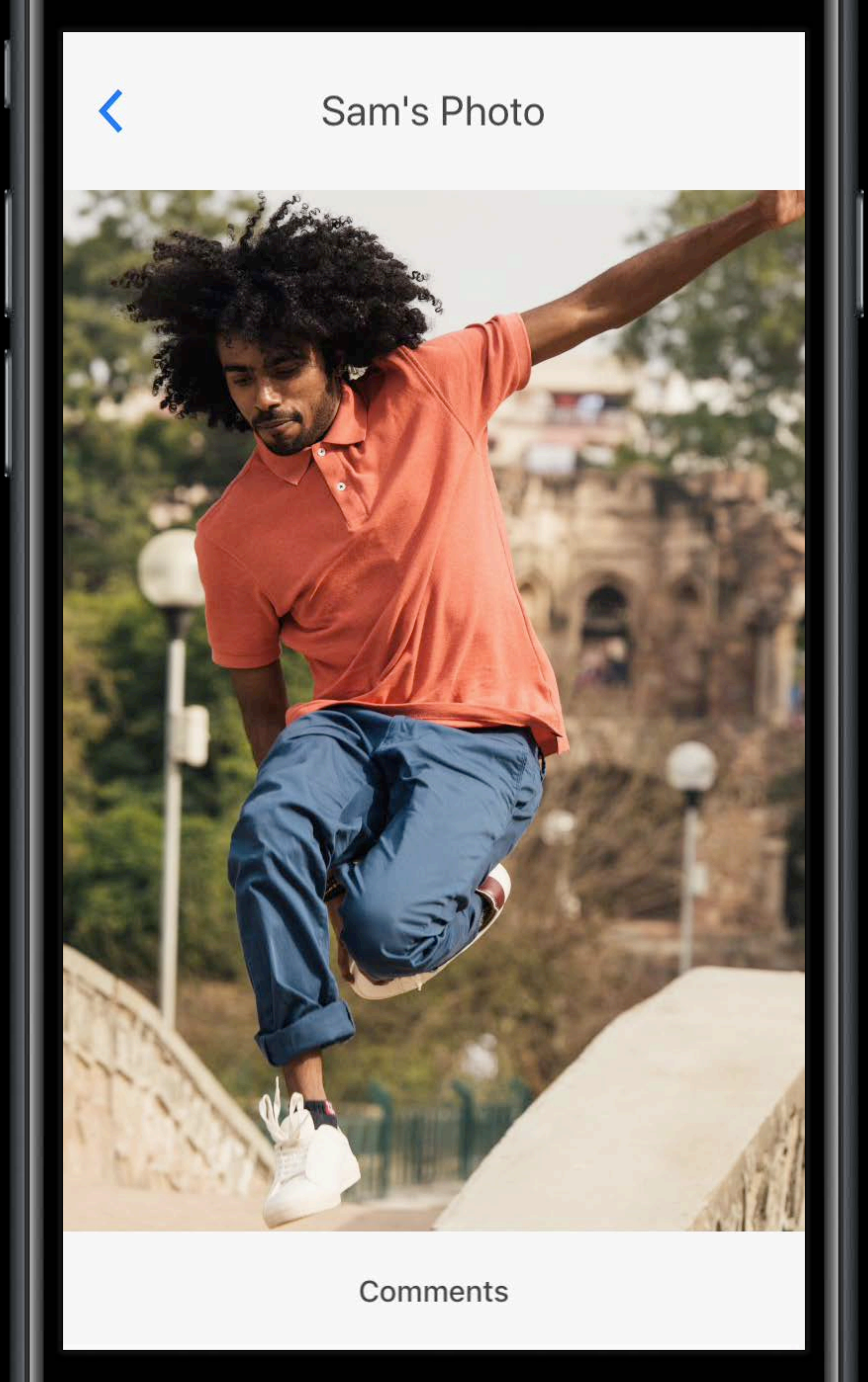
```swift
// Tracks all running animators
var runningAnimators = [UIViewPropertyAnimator]()

// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary or reverses it on tap
func animateOrReverseRunningTransition(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary and pauses on pan .begin
func startInteractiveTransition(state: State, duration: TimeInterval) { ... }

// Scrubs transition on pan .changed
func updateInteractiveTransition(fractionComplete: CGFloat) { ... }

// Continues or reverse transition on pan .ended
func continueInteractiveTransition(cancel: Bool) { ... }
```

```swift
// Tracks all running animators
var runningAnimators = [UIViewPropertyAnimator]()

// Perform all animations with animators if not already running
func animateTransitionIfNeeded(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary or reverses it on tap
func animateOrReverseRunningTransition(state: State, duration: TimeInterval) { ... }

// Starts transition if necessary and pauses on pan .begin
func startInteractiveTransition(state: State, duration: TimeInterval) { ... }

// Scrubs transition on pan .changed
func updateInteractiveTransition(fractionComplete: CGFloat) { ... }

// Continues or reverse transition on pan .ended
func continueInteractiveTransition(cancel: Bool) { ... }
```

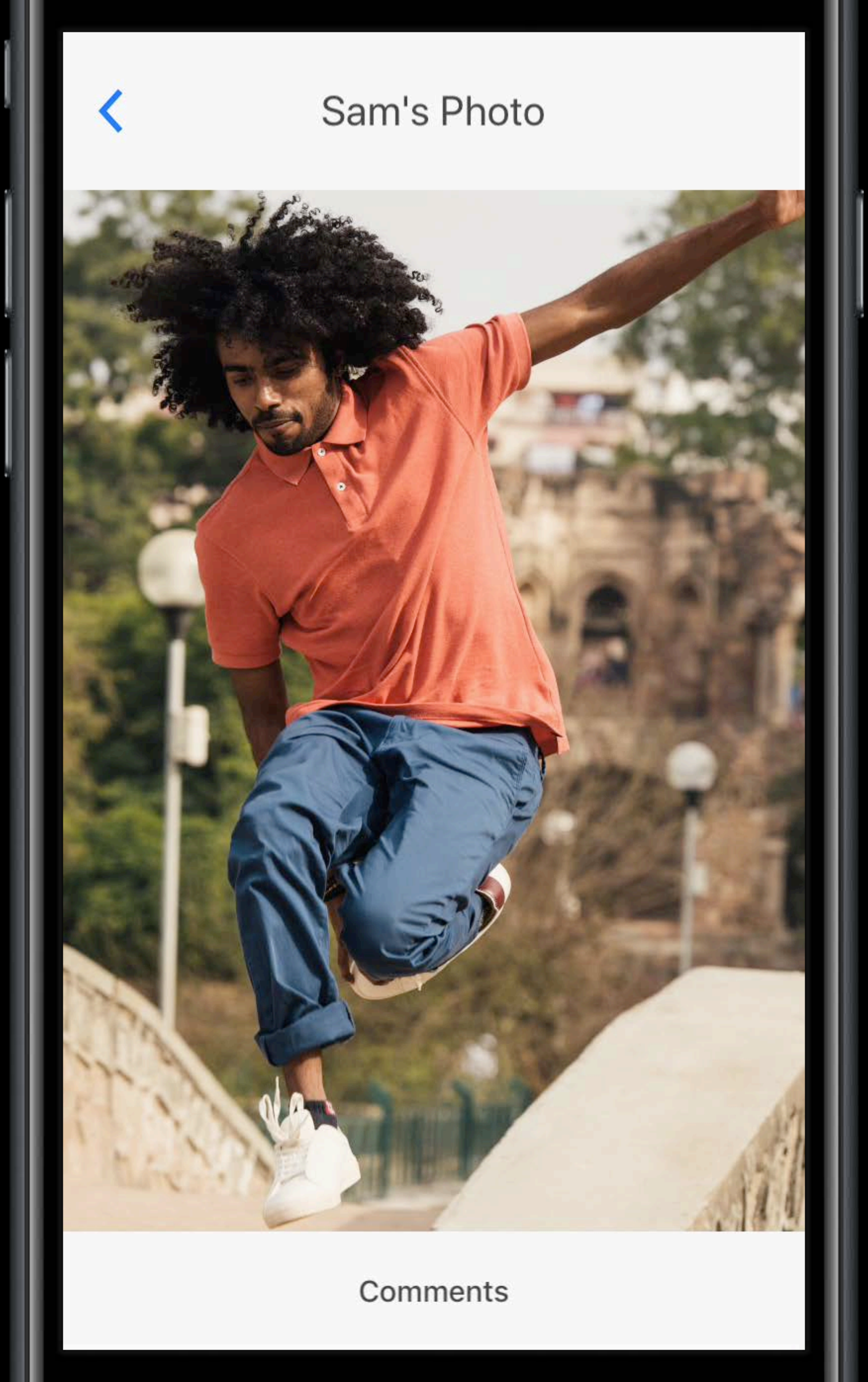
Sam's Photo
Comments

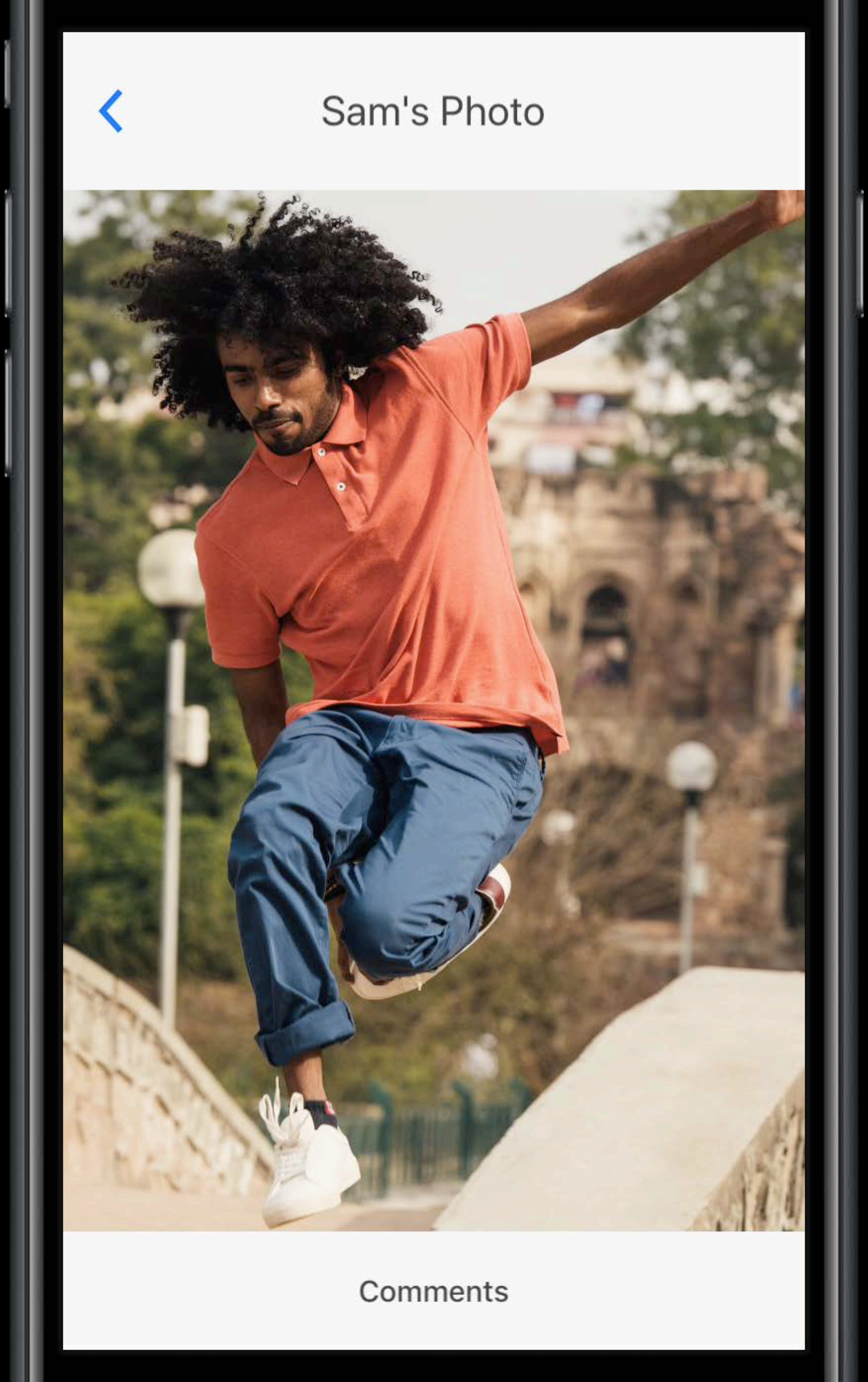
Sam's Photo
Comments


Sam's Photo
Comments


Sam's Photo
Comments

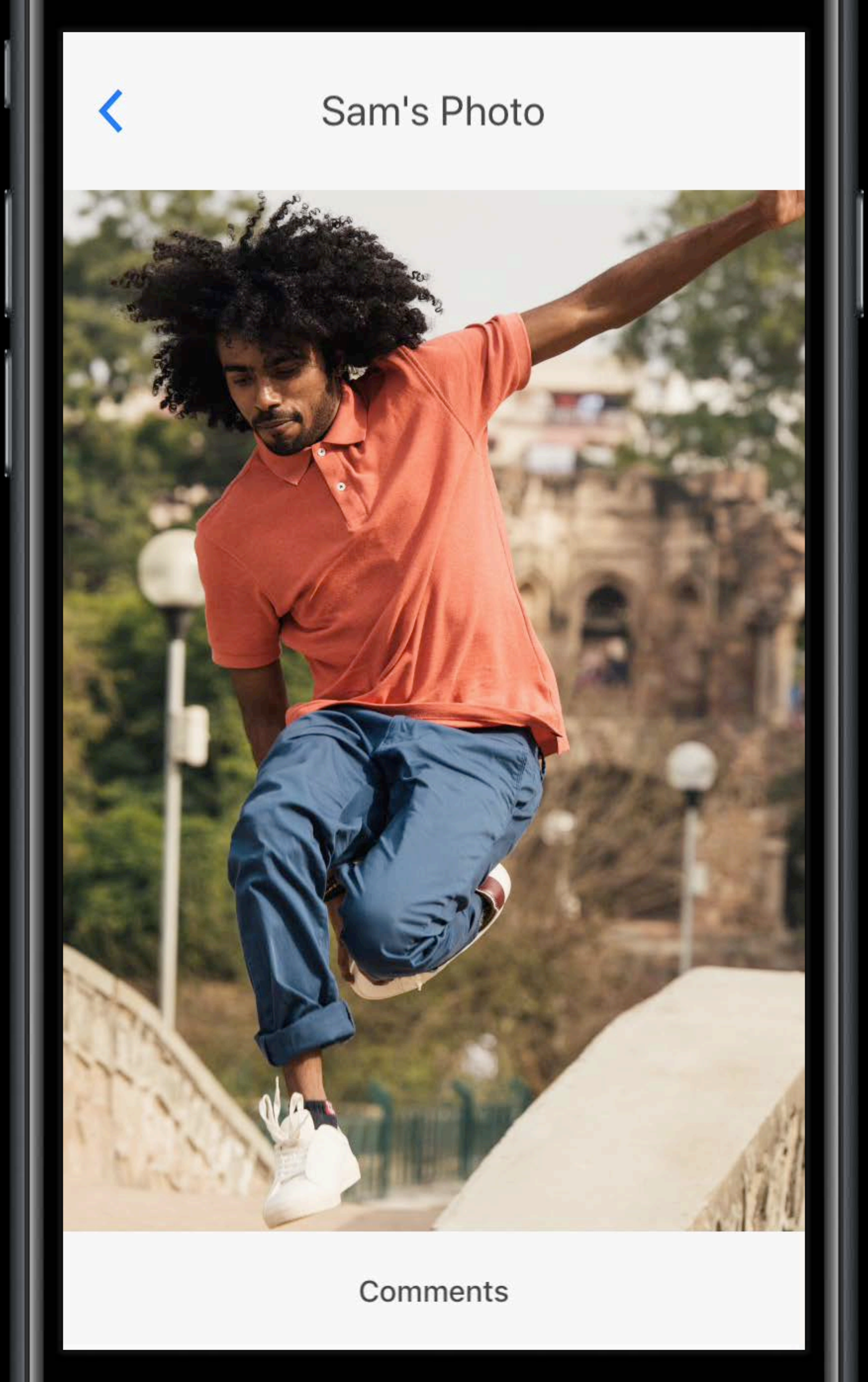
Sam's Photo
Comments


Sam's Photo
Comments

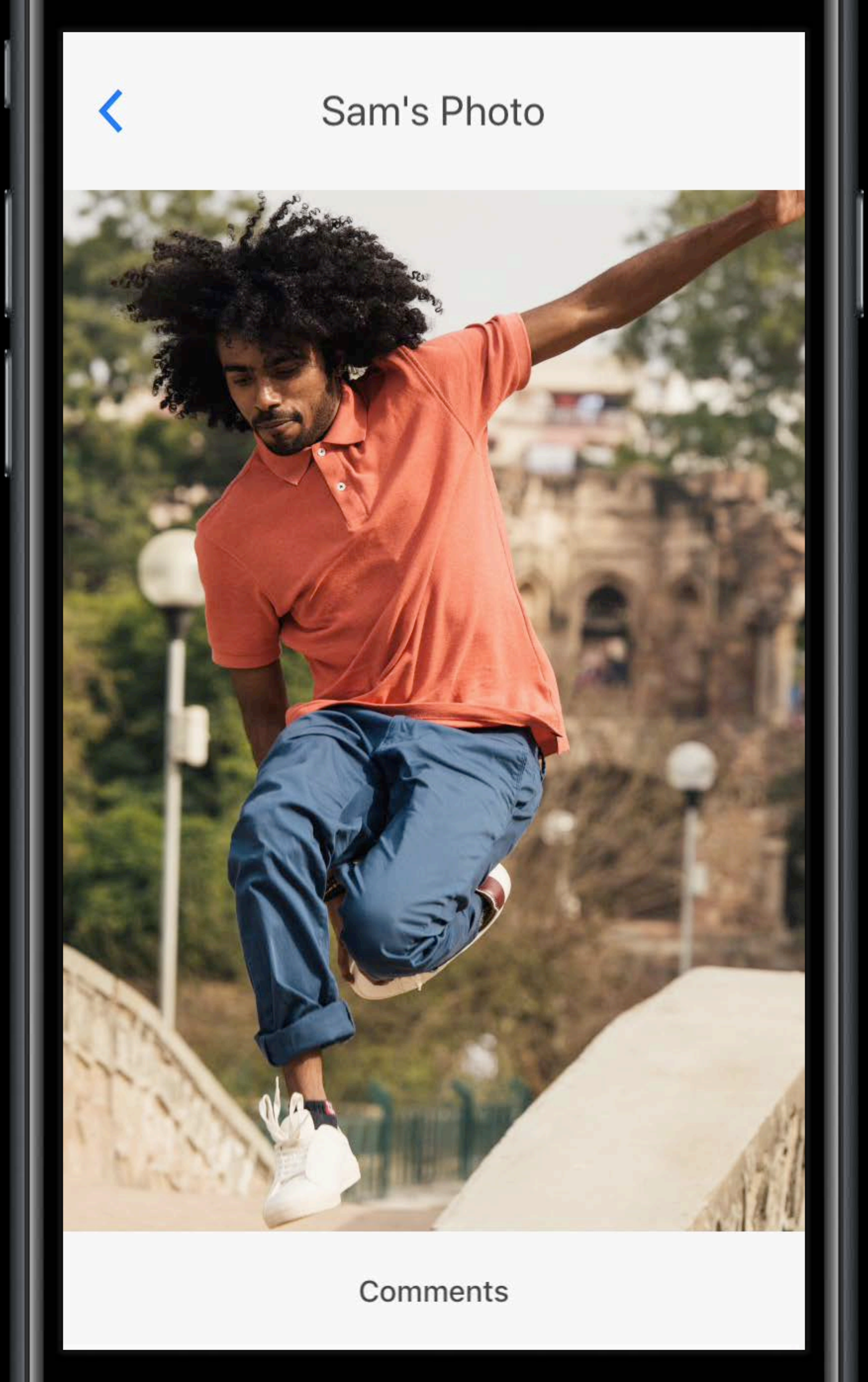


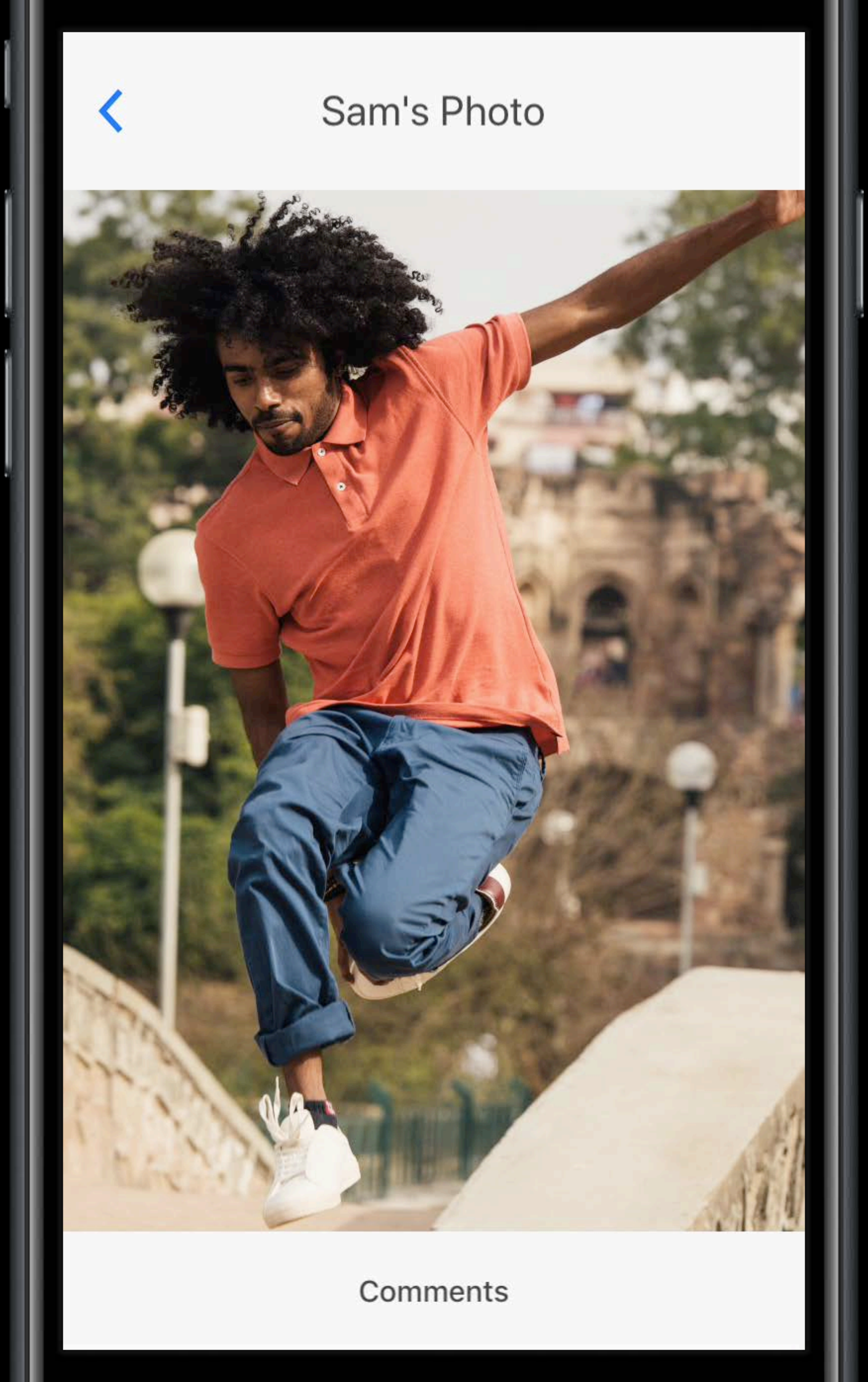
Sam's Photo

Comments



Sam's Photo
Comments


Sam's Photo
Comments


Sam's Photo
Comments

## Animating a Blur

## UIVisualEffectView

```swift
class UIVisualEffectView: UIView {
    var effect: UIVisualEffect    // animatable
}
class UIBlurEffect: UIVisualEffect {
    init(style: UIBlurEffectStyle)
}
class UIVibrancyEffect: UIVisualEffect {
    init(blurEffect: UIBlurEffect)
}
```

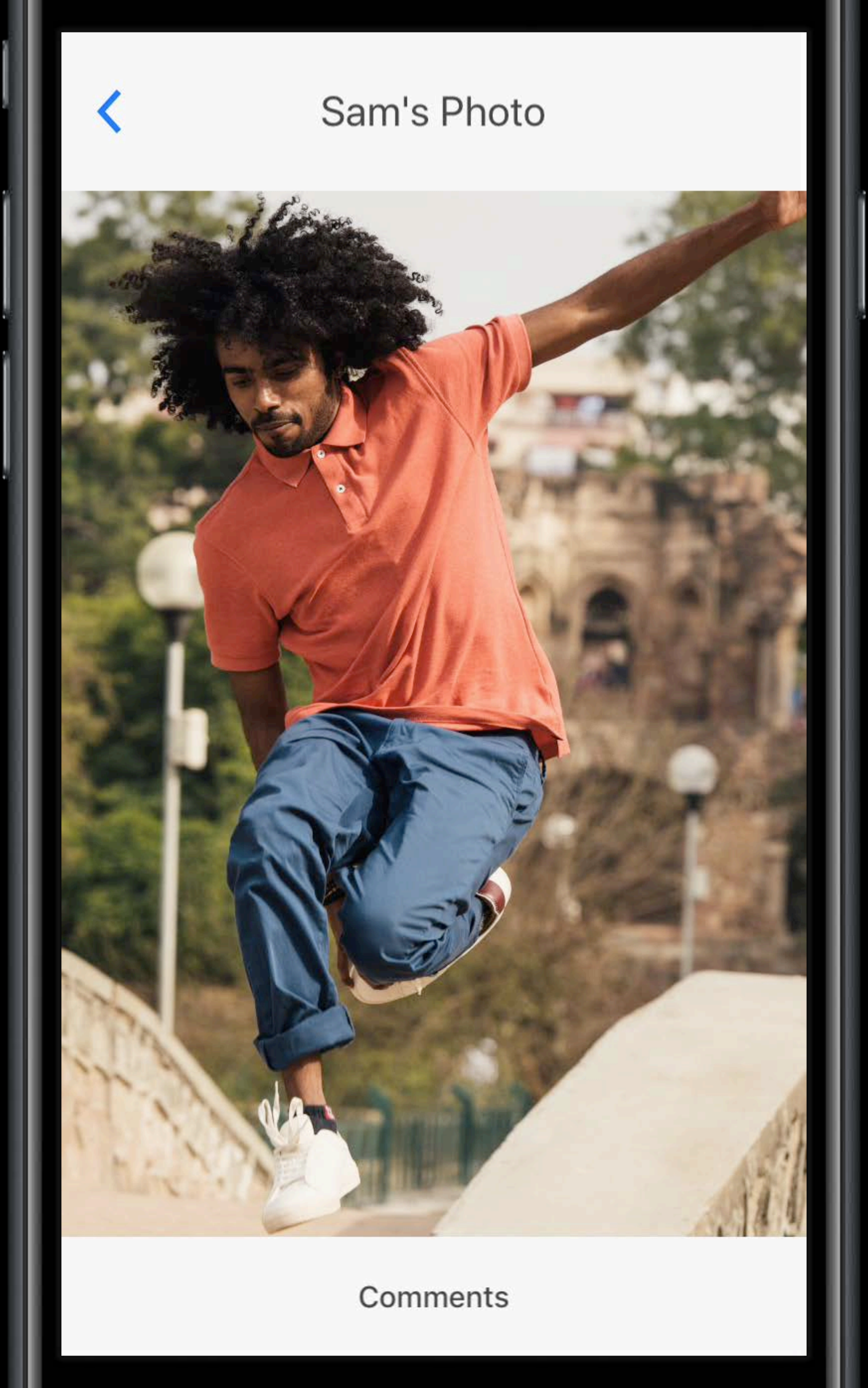
```swift
func animateTransitionIfNeeded(forState state: State, duration: TimeInterval) {
    // ...
    let blurAnimator = UIViewPropertyAnimator(duration: duration, dampingRatio: 1) {
        switch state {
        case .Expanded:
            self.blurEffectView.effect = UIBlurEffect(style: .dark)
        }
        case .Collapsed:
            self.blurEffectView.effect = nil
        }
    }
    blurAnimator.startAnimation()
    runningAnimators.append(blurAnimator)
    // ...
}
```

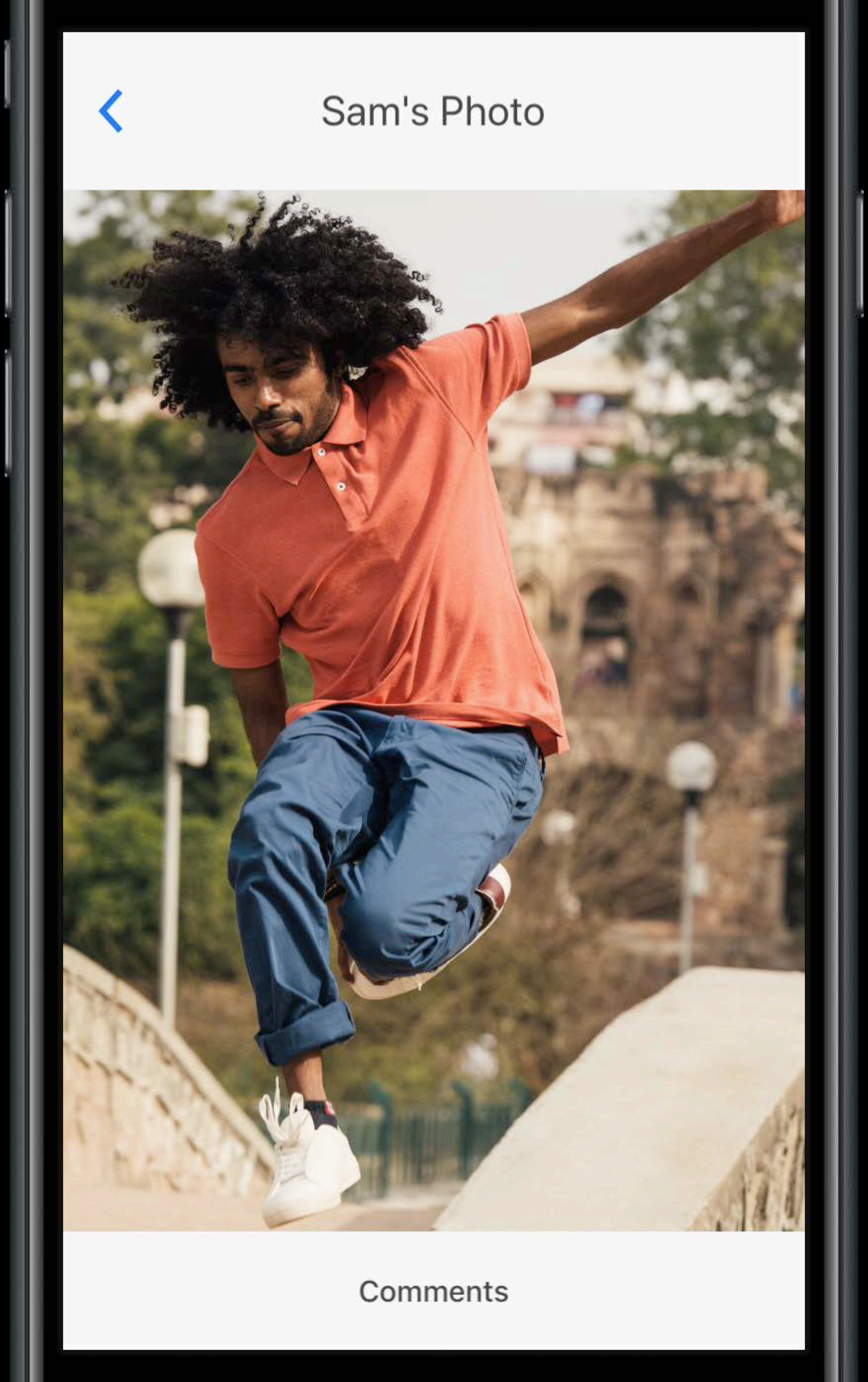
```swift
func animateTransitionIfNeeded(forState state: State, duration: TimeInterval) {
    // ...
    let blurAnimator = UIViewPropertyAnimator(duration: duration, dampingRatio: 1) {
        switch state {
        case .Expanded:
            self.blurEffectView.effect = UIBlurEffect(style: .dark)
        }
        case .Collapsed:
            self.blurEffectView.effect = nil
        }
    }
    blurAnimator.startAnimation()
    runningAnimators.append(blurAnimator)
    // ...
}
```

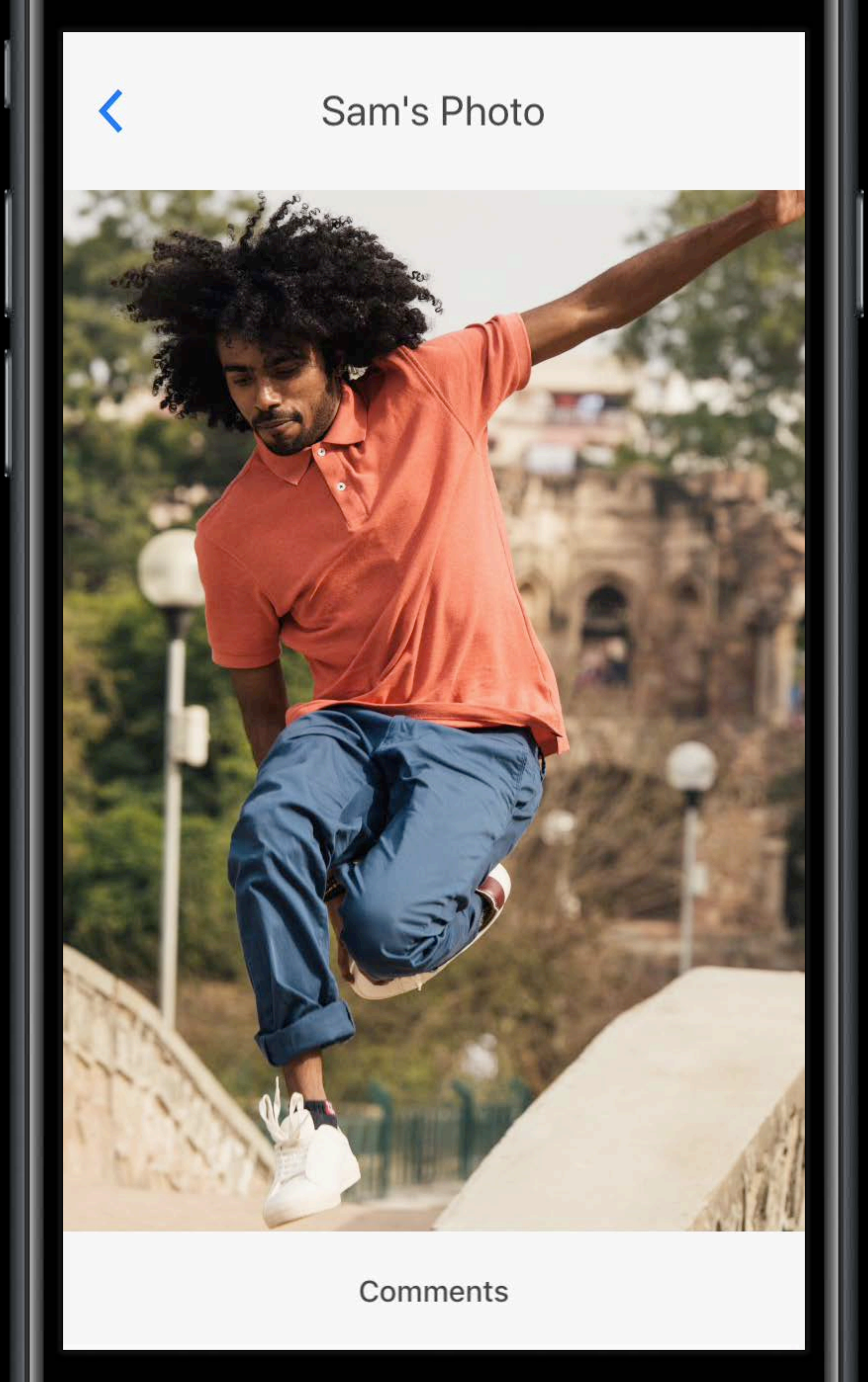
```swift
func animateTransitionIfNeeded(forState state: State, duration: TimeInterval) {
    // ...
    let blurAnimator = UIViewPropertyAnimator(duration: duration, dampingRatio: 1) {
        switch state {
        case .Expanded:
            self.blurEffectView.effect = UIBlurEffect(style: .dark)
        }
        case .Collapsed:
            self.blurEffectView.effect = nil
        }
    }
    blurAnimator.startAnimation()
    runningAnimators.append(blurAnimator)
    // ...
}
```

```swift
func animateTransitionIfNeeded(forState state: State, duration: TimeInterval) {
    // ...
    let blurAnimator = UIViewPropertyAnimator(duration: duration, dampingRatio: 1) {
        switch state {
        case .Expanded:
            self.blurEffectView.effect = UIBlurEffect(style: .dark)
        }
        case .Collapsed:
            self.blurEffectView.effect = nil
        }
    }
    blurAnimator.startAnimation()
    runningAnimators.append(blurAnimator)
    // ...
}
```

Sam's Photo
Comments


Sam's Photo
Comments

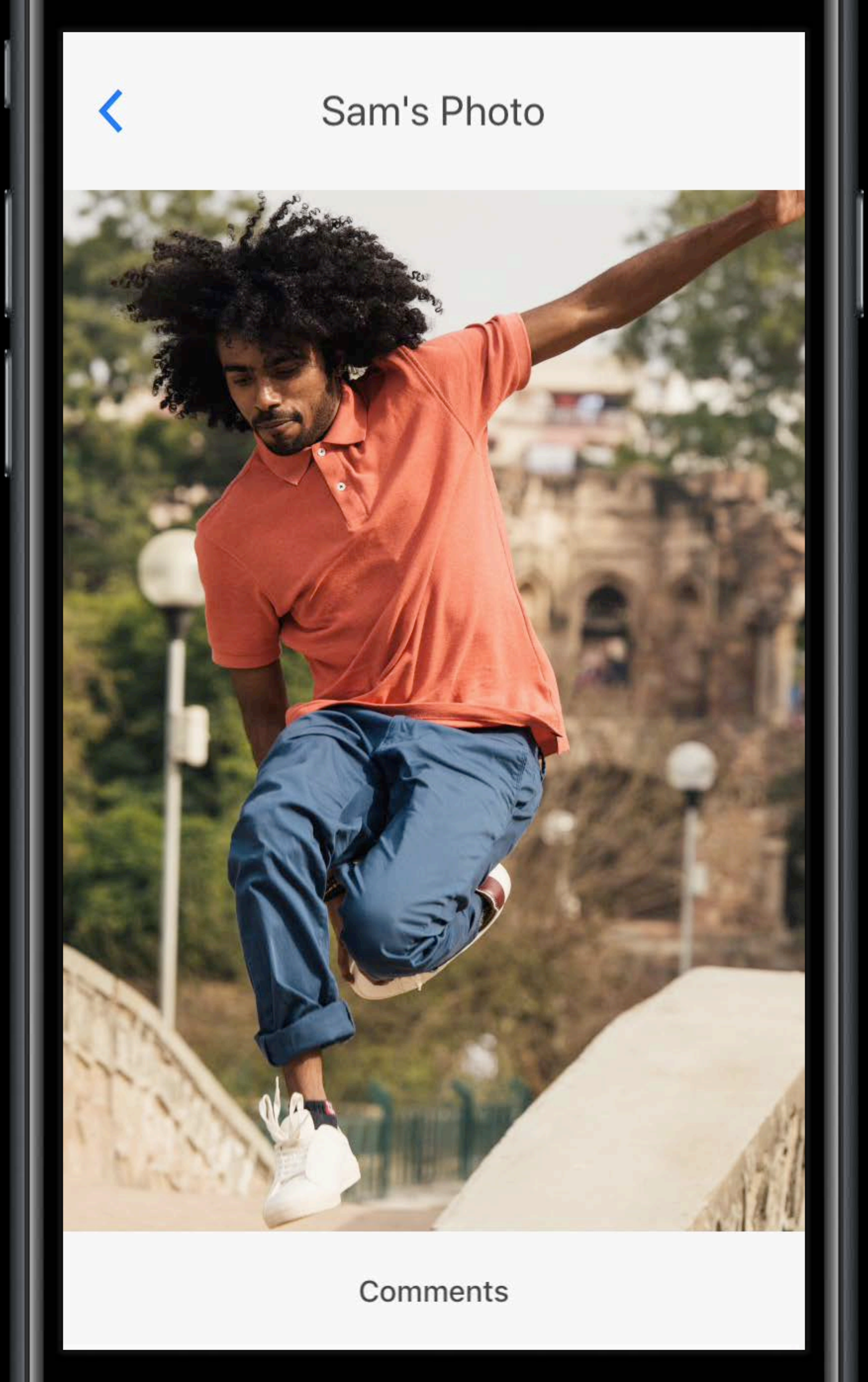

Sam's Photo
Comments


Sam's Photo
Comments

Sam's Photo
Comments

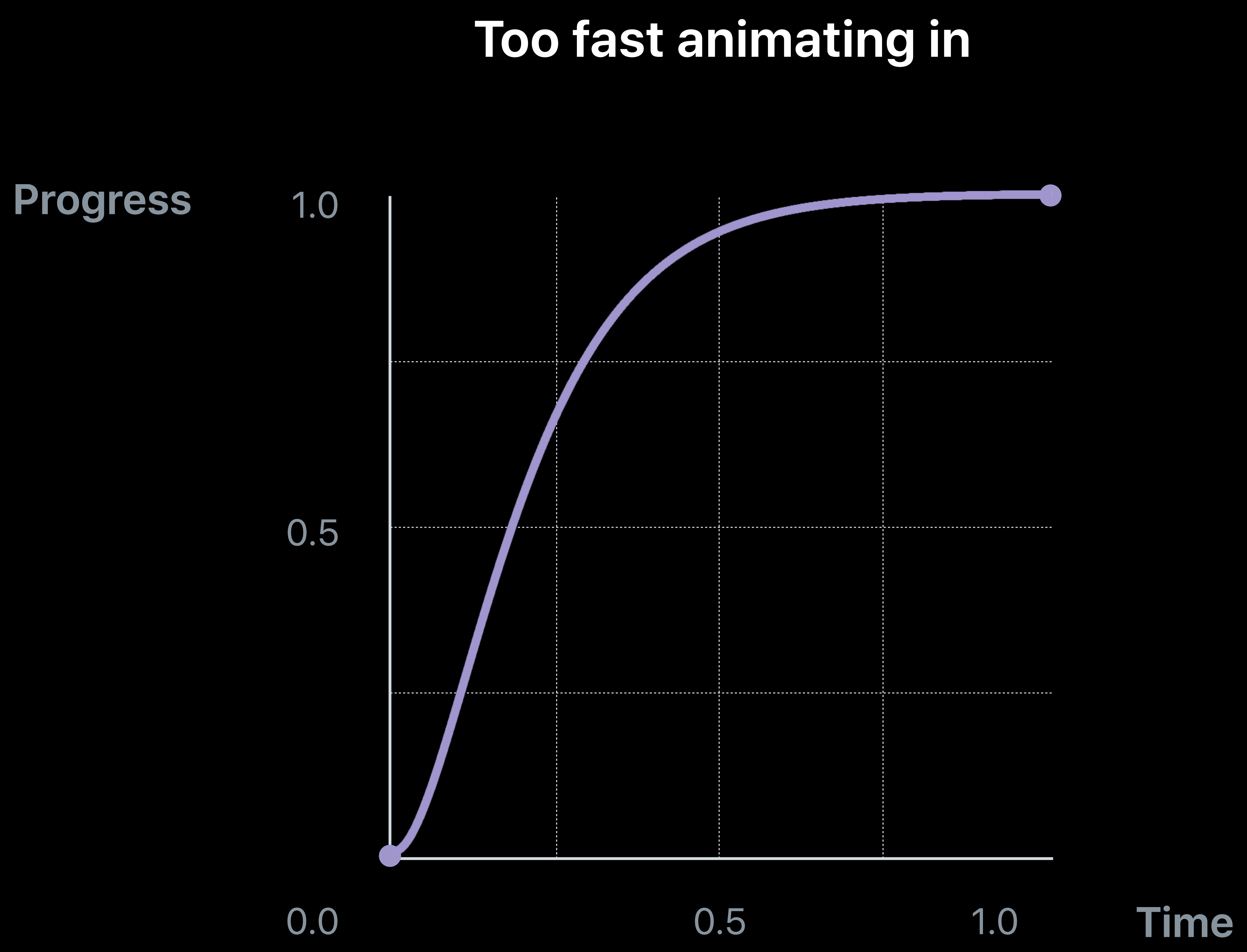
Sam's Photo
Comments

Sam's Photo
Comments

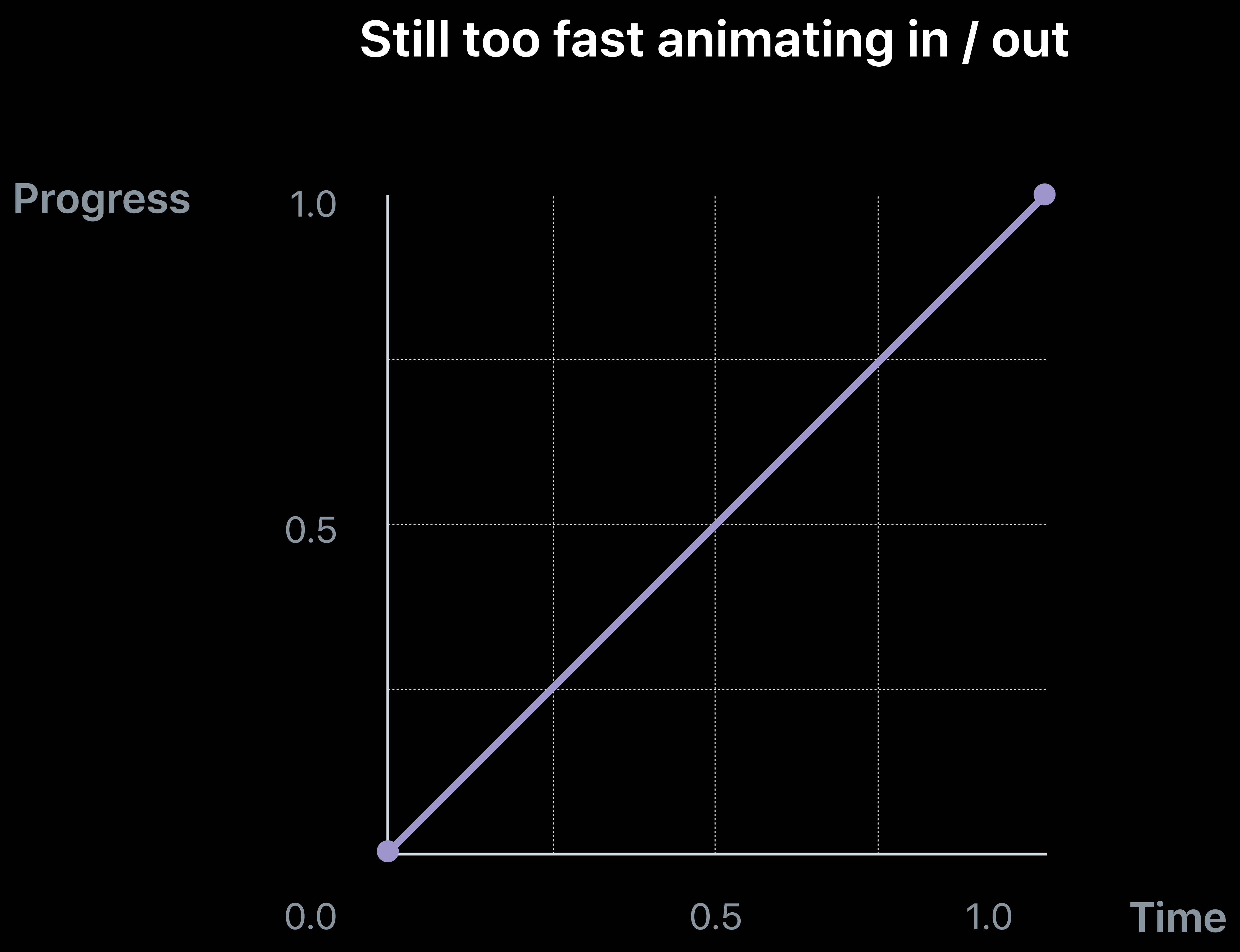

Sam's Photo
Comments

## Issues

# Issues



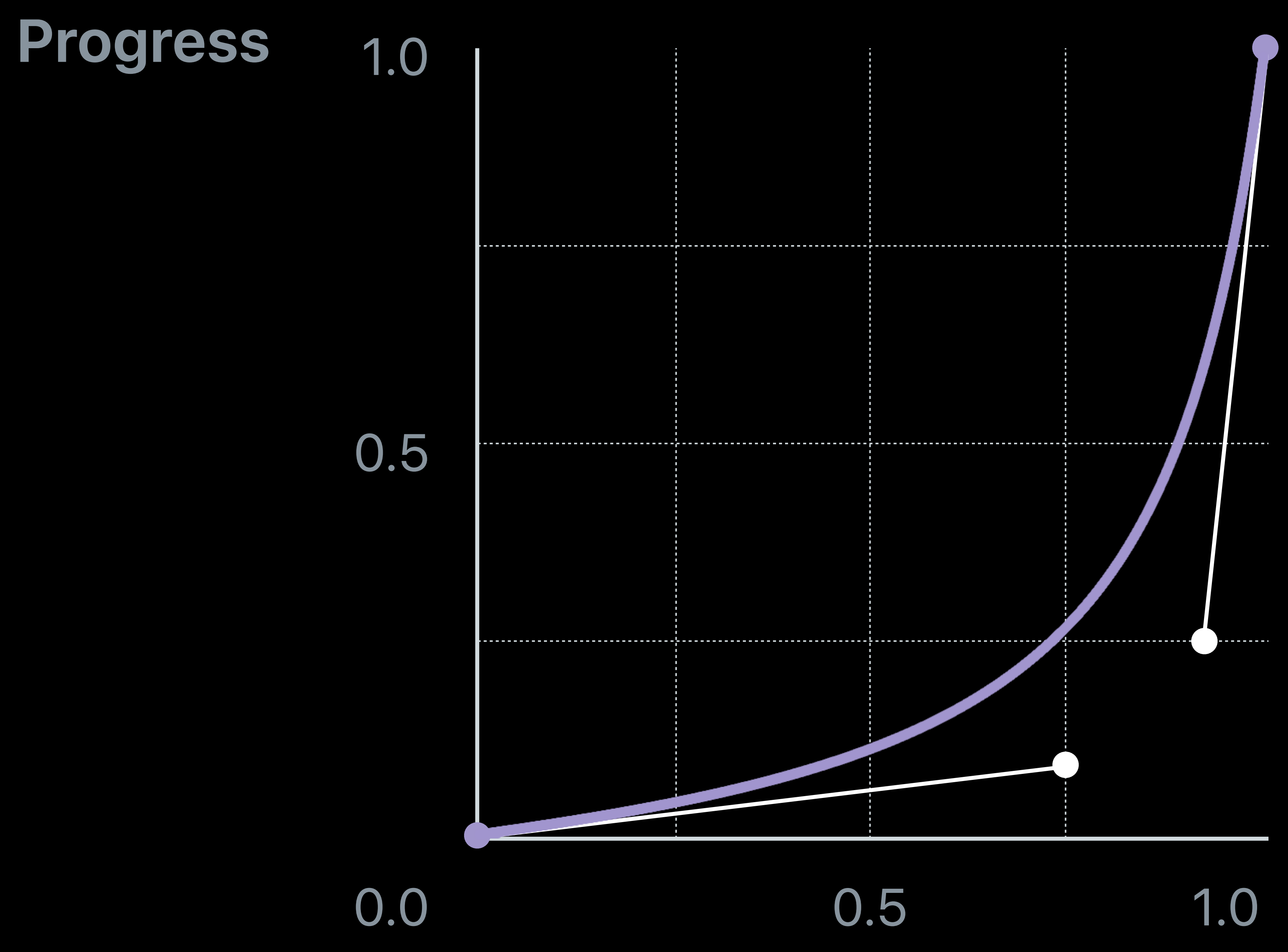
## Issues



# Custom Timing

## Symmetric pacing

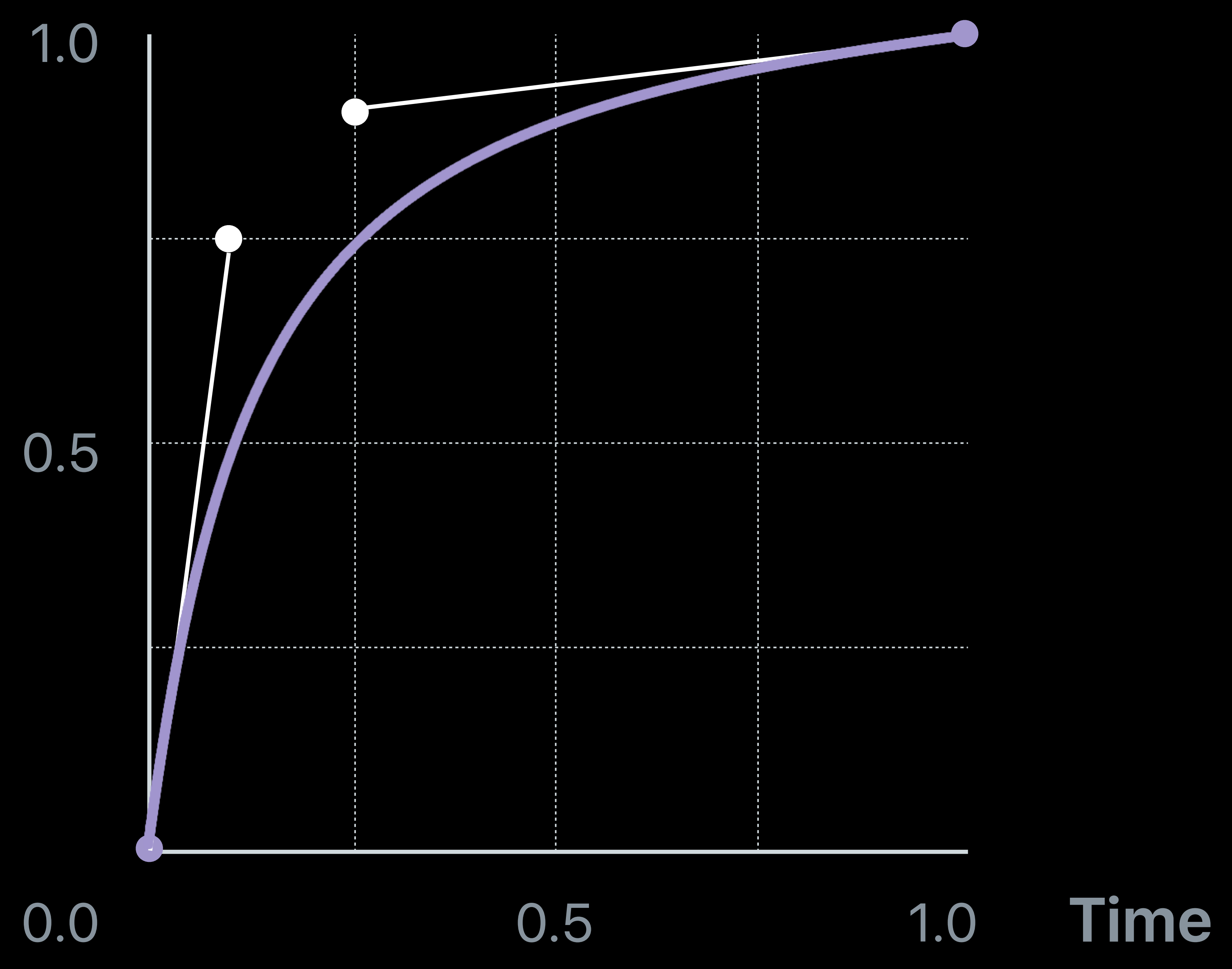
### Custom Ease In



**Animates our blur in slowly**

`(0.75, 0.1) (0.9, 0.25)`

### Custom Ease Out

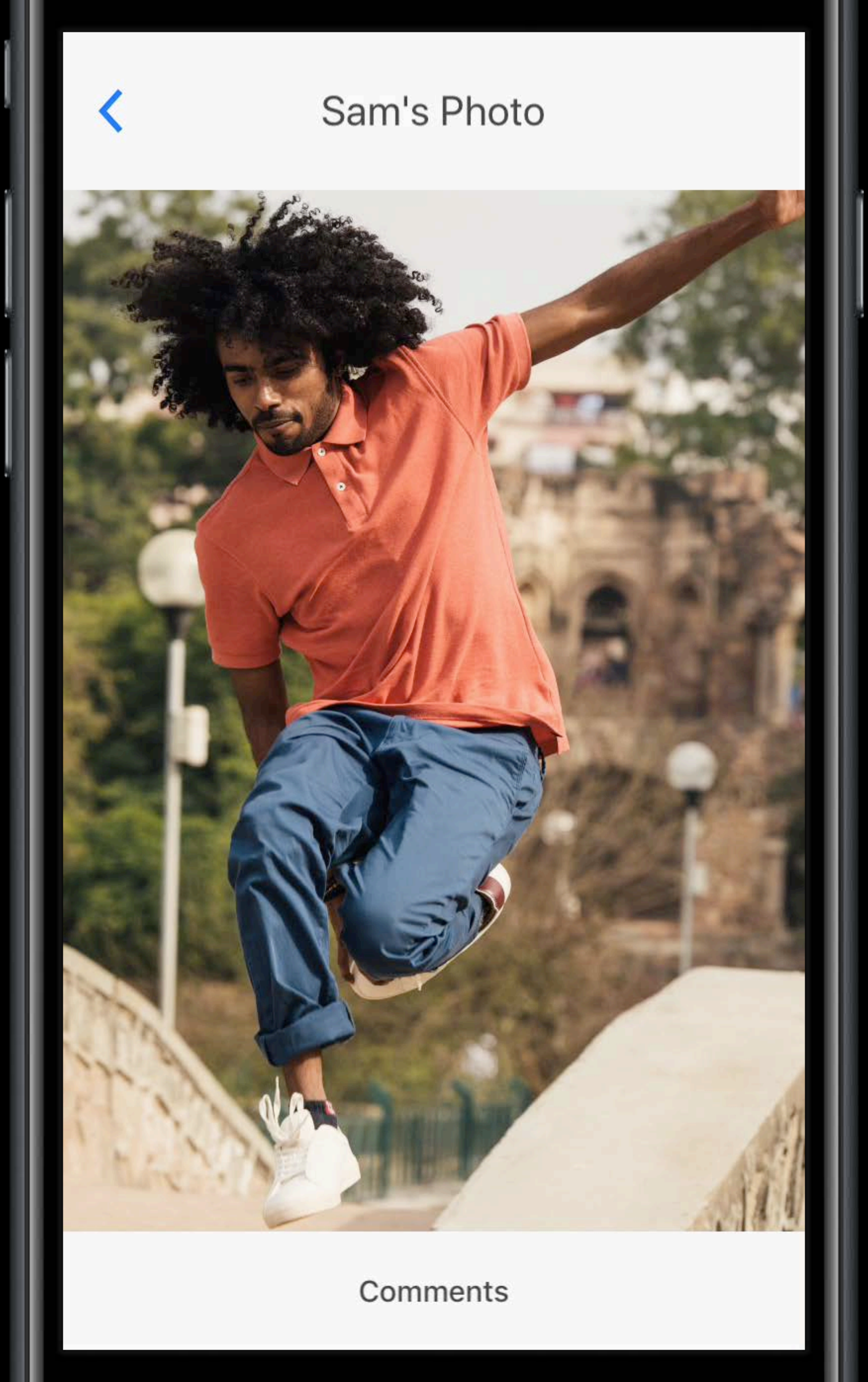


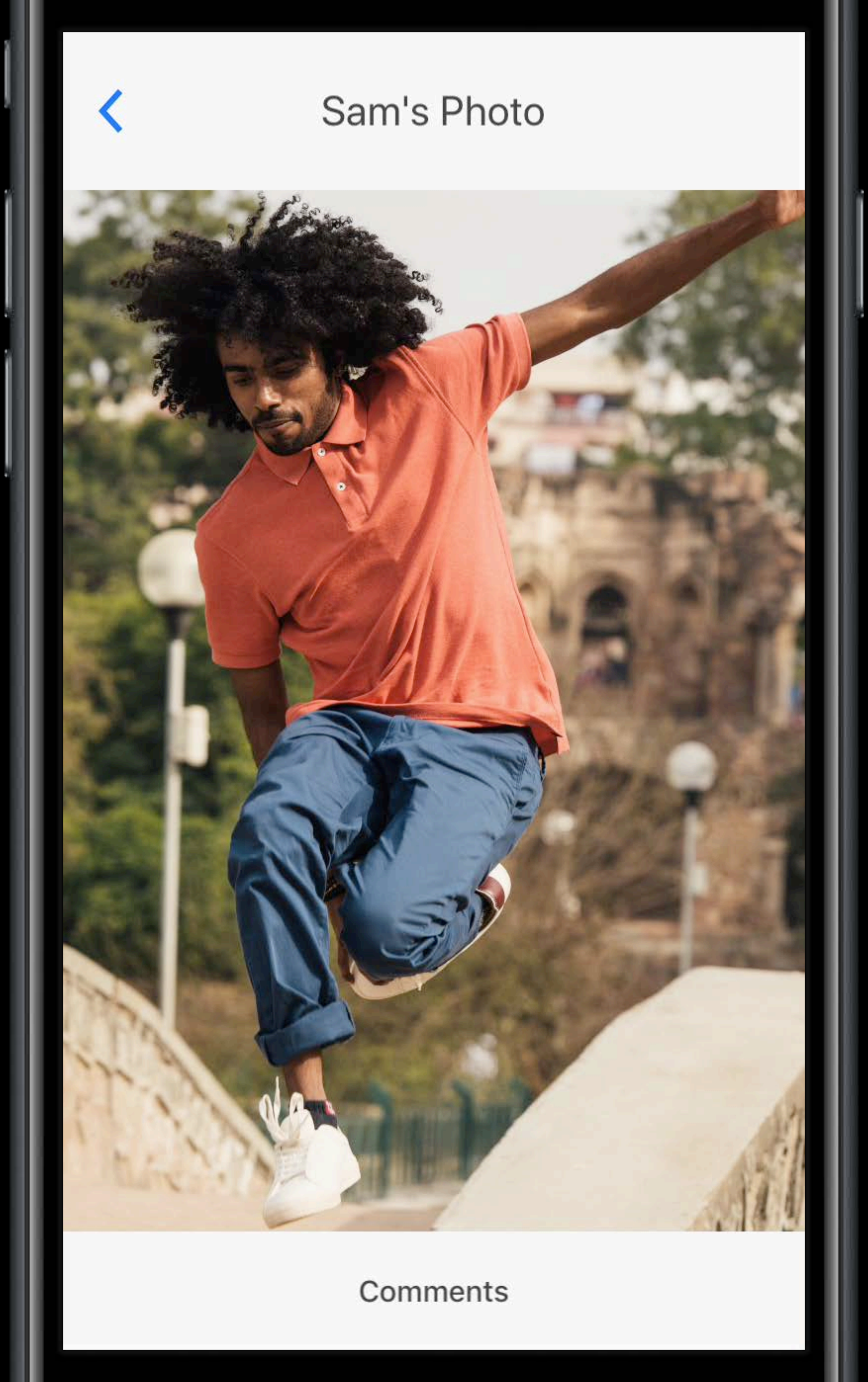
**Animates our blur out quickly**

`(0.1, 0.75) (0.25, 0.9)`

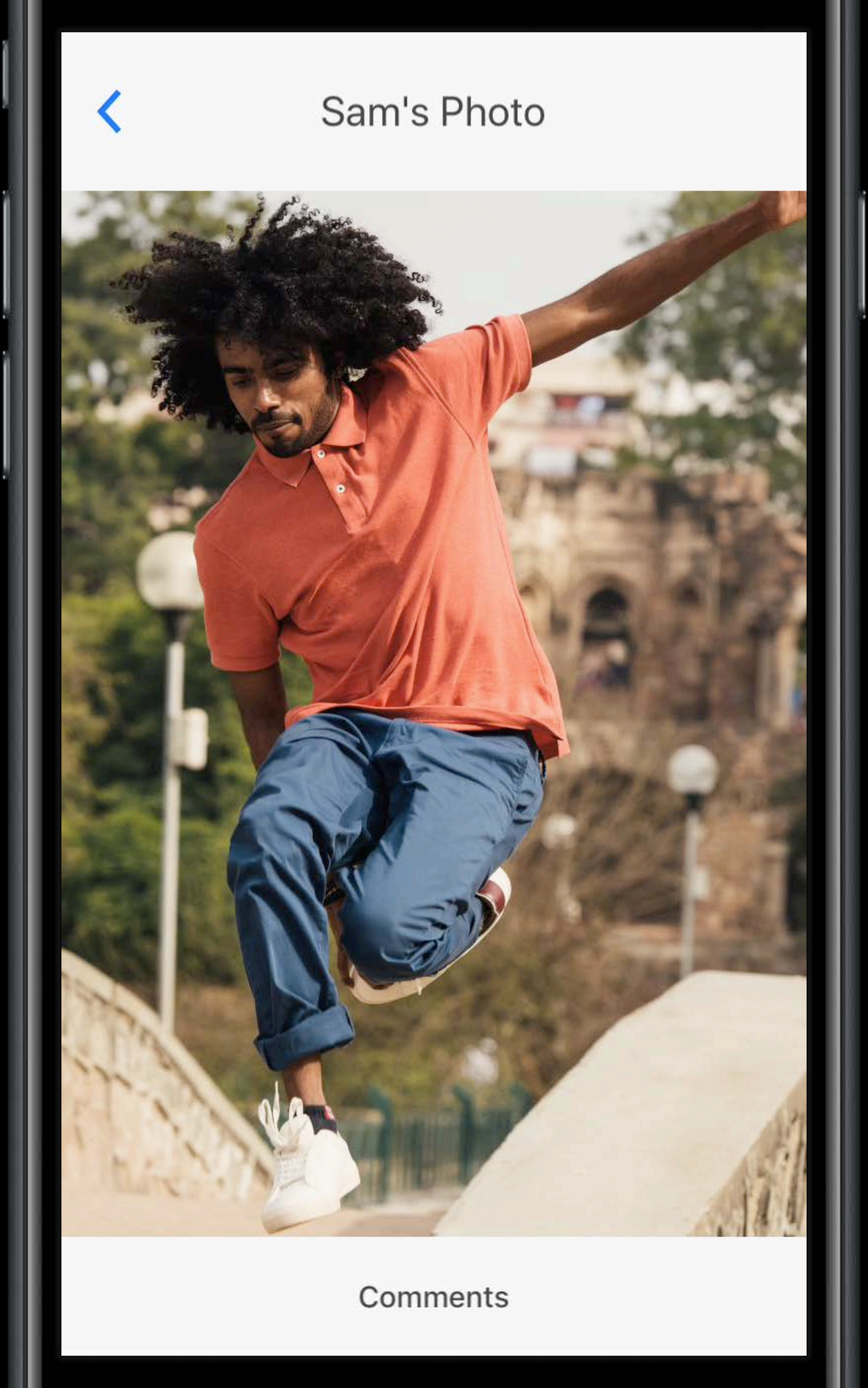
```swift
func animateTransitionIfNeeded(forState state: State, duration: TimeInterval) {
    // ...
    let timing: UITimingCurveProvider
    switch state {
    case .Expanded:
        timing = UICubicTimingParameters(controlPoint1: CGPoint(x: 0.75, y: 0.1),
                                         controlPoint2: CGPoint(x: 0.9, y: 0.25))
    case .Collapsed:
        timing = UICubicTimingParameters(controlPoint1: CGPoint(x: 0.1, y: 0.75),
                                         controlPoint2: CGPoint(x: 0.25, y: 0.9))
    }
    let blurAnimator = UIViewPropertyAnimator(duration: duration, timingParameters: timing)
    blurAnimator.scrubsLinearly = false
    // ...
}
```

Sam's Photo
Comments

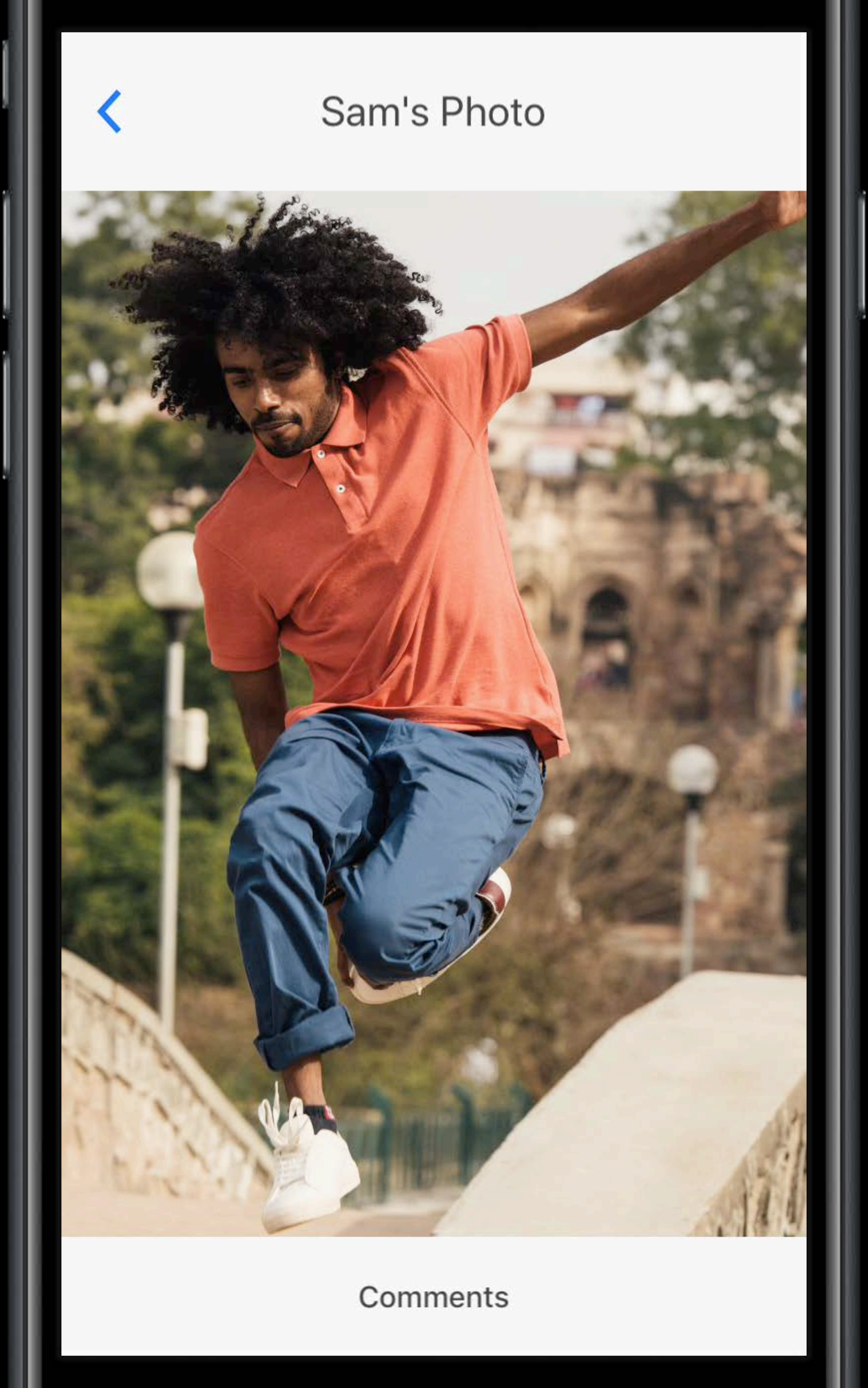

Sam's Photo
Comments


Sam's Photo
Comments

Sam's Photo
Comments


Sam's Photo
Comments

Sam's Photo
Comments



Sam's Photo

Comments

Sam's Photo
Comments

## View Morphing

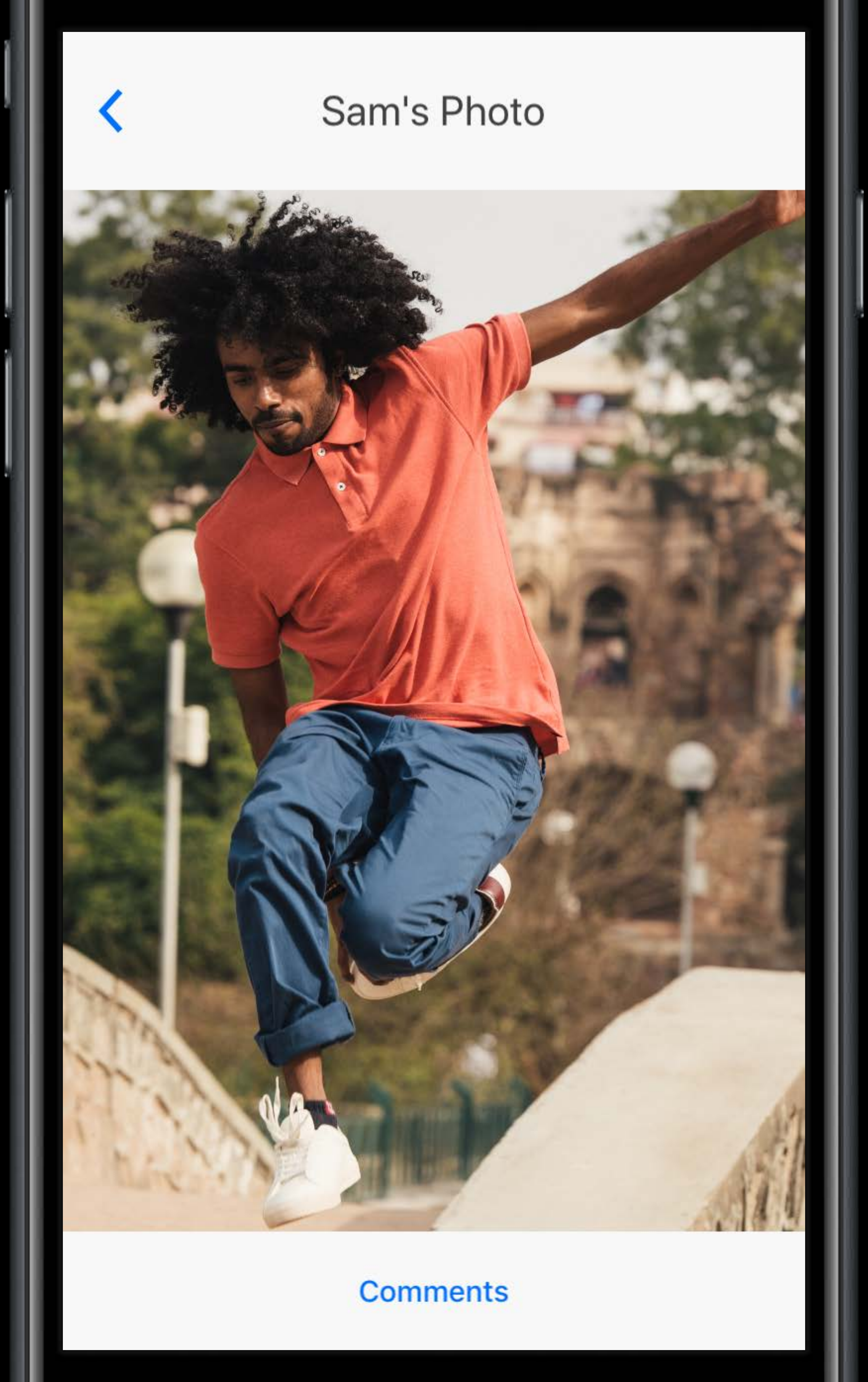
Hello

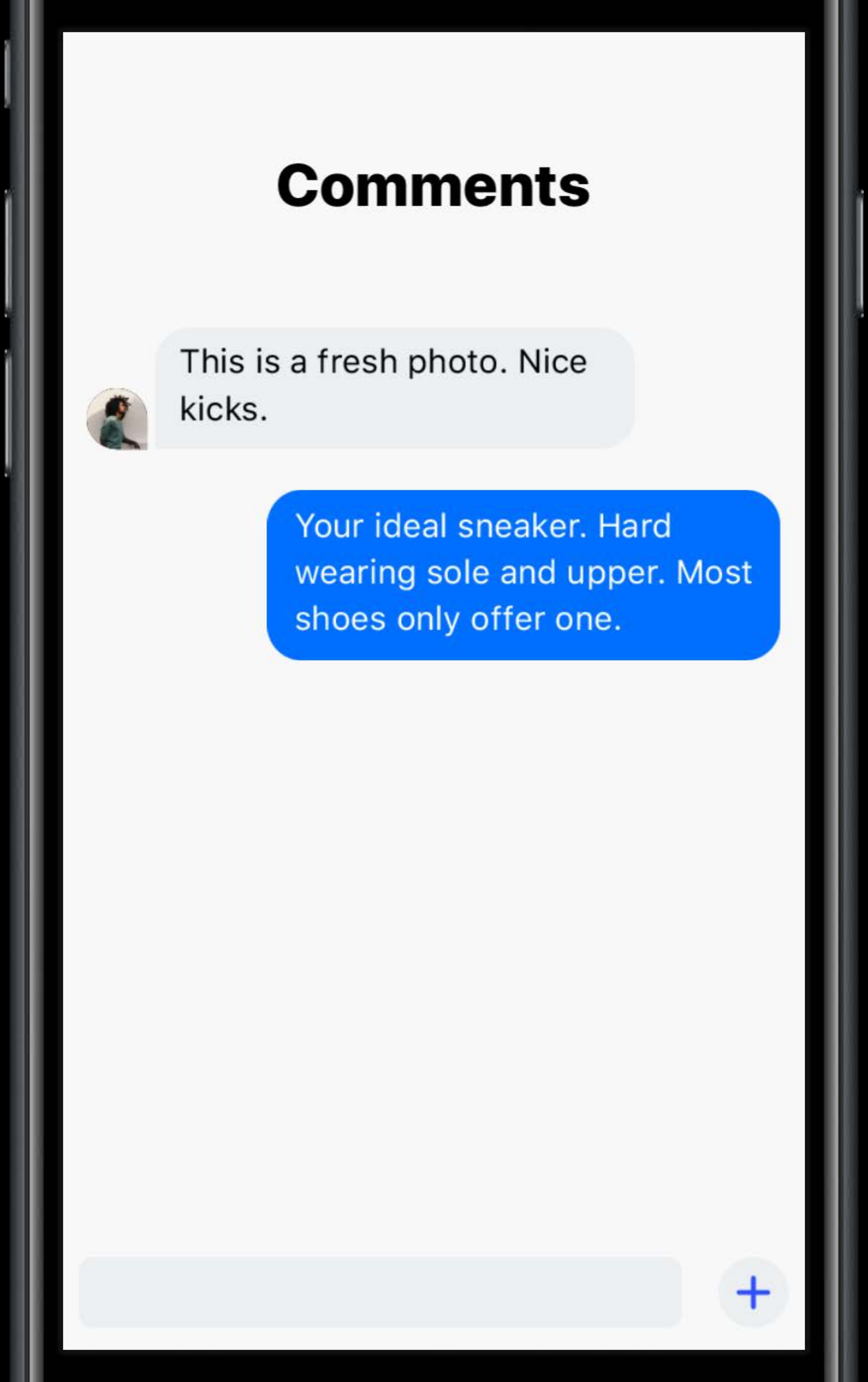
## View Morphing

Hello

## View Morphing

Scaling, translation, and opacity blending of two views

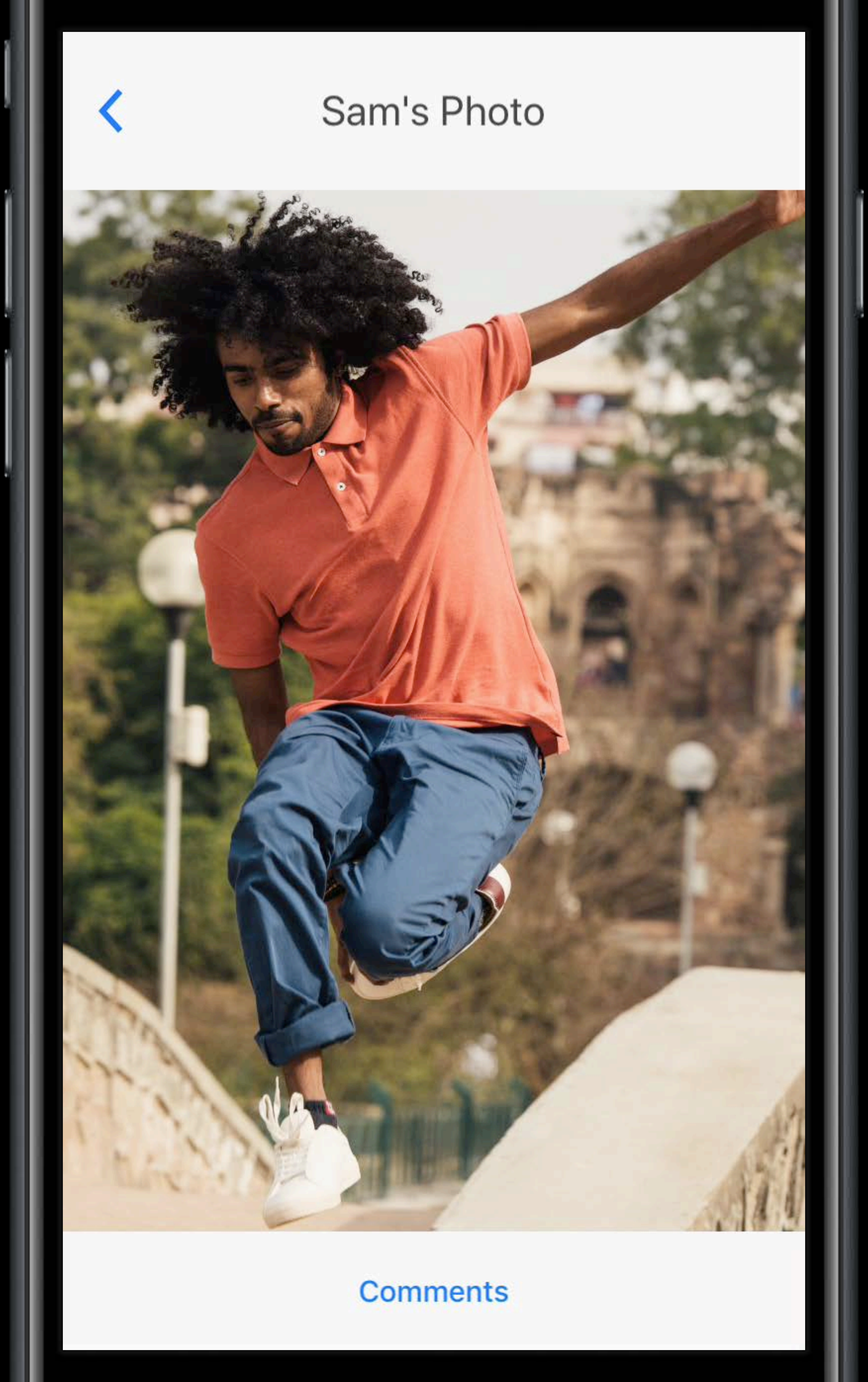

Sam's Photo
Comments

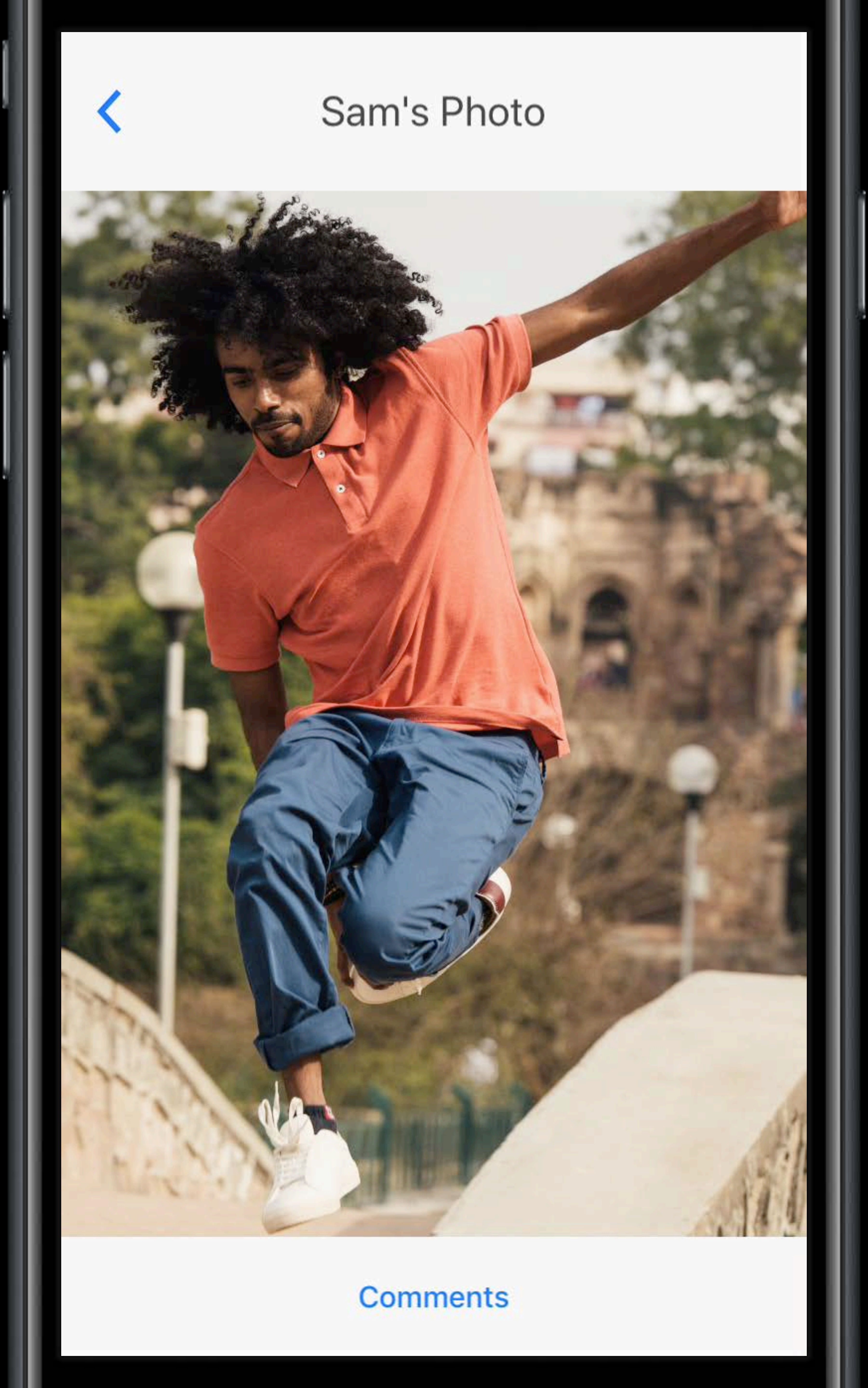


Comments

This is a fresh photo. Nice kicks.

Your ideal sneaker. Hard wearing sole and upper. Most shoes only offer one.


Sam's Photo
Comments


Sam's Photo
Comments

## Strategy

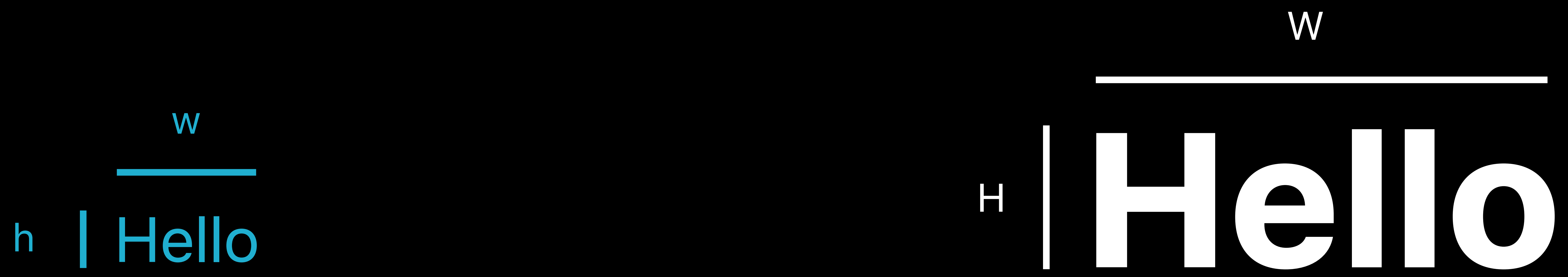
## Strategy

.transform: CGAffineTransform

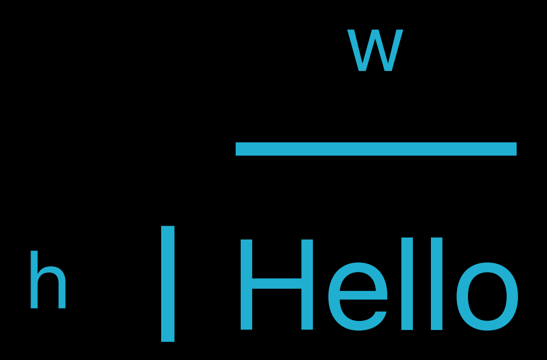
Compute transform.scale and transform.translation

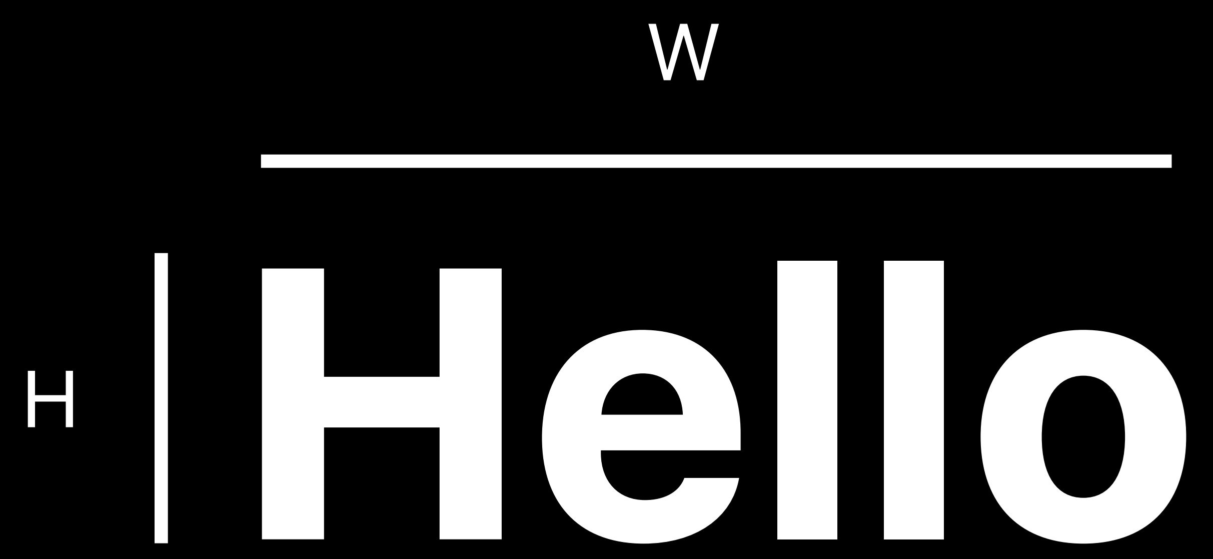
Prepare views and animate .transform and .alpha

## Computing Scale



## Computing Scale





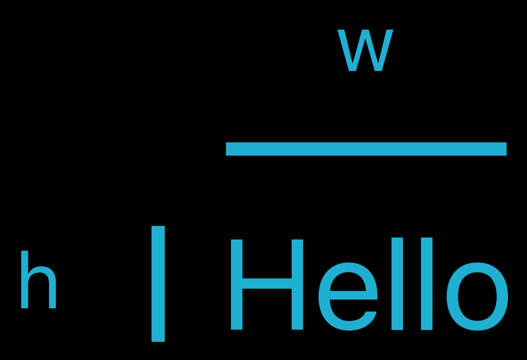
$.scale.width = \frac{W}{w}$

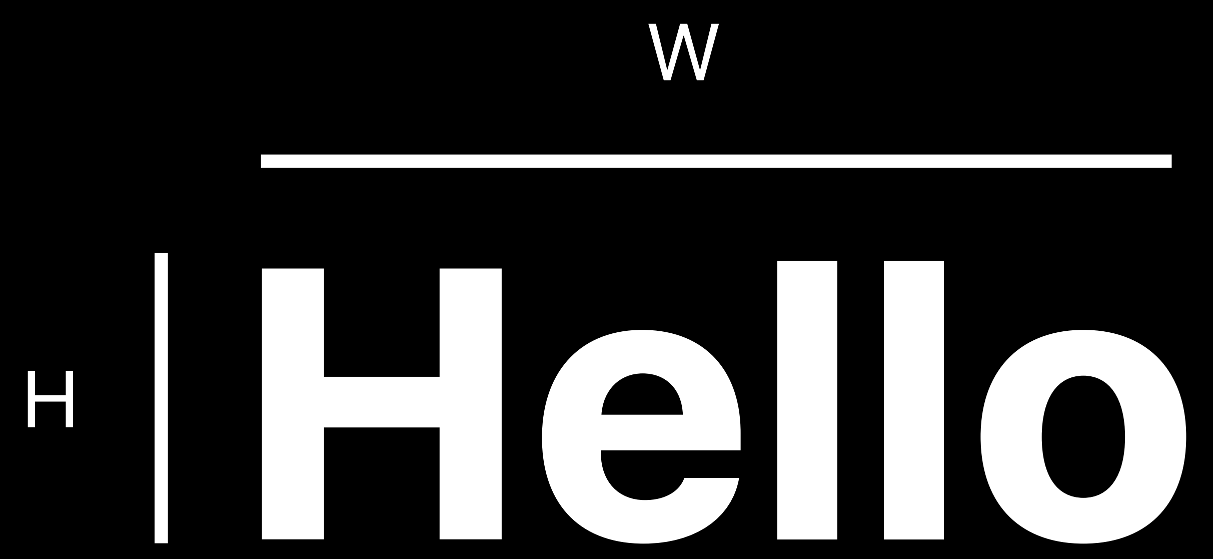
$.scale.height = \frac{H}{h}$

$.scale.width = \frac{w}{W}$

$.scale.height = \frac{h}{H}$

# Computing Scale





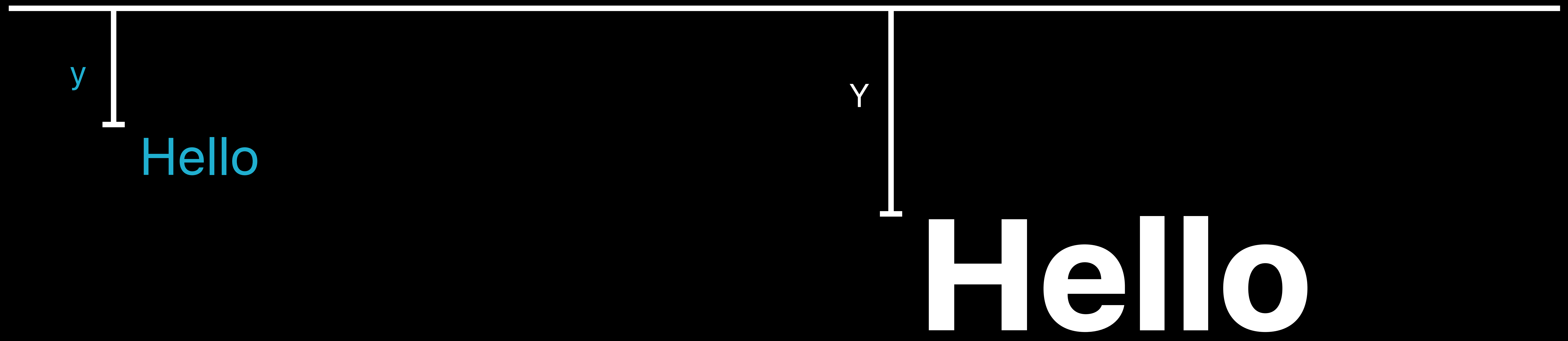
$$.scale.width = \frac{W}{w}$$

$$.scale.height = \frac{H}{h}$$

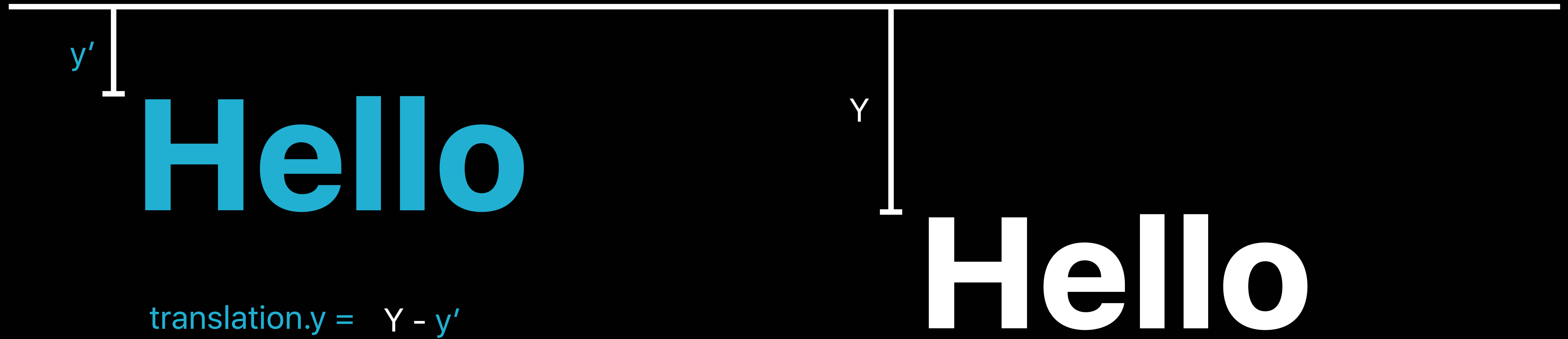
$$.scale.width = \frac{w}{W} = \frac{1}{.scale.width}$$

$$.scale.height = \frac{h}{H} = \frac{1}{.scale.height}$$

## Computing Translation

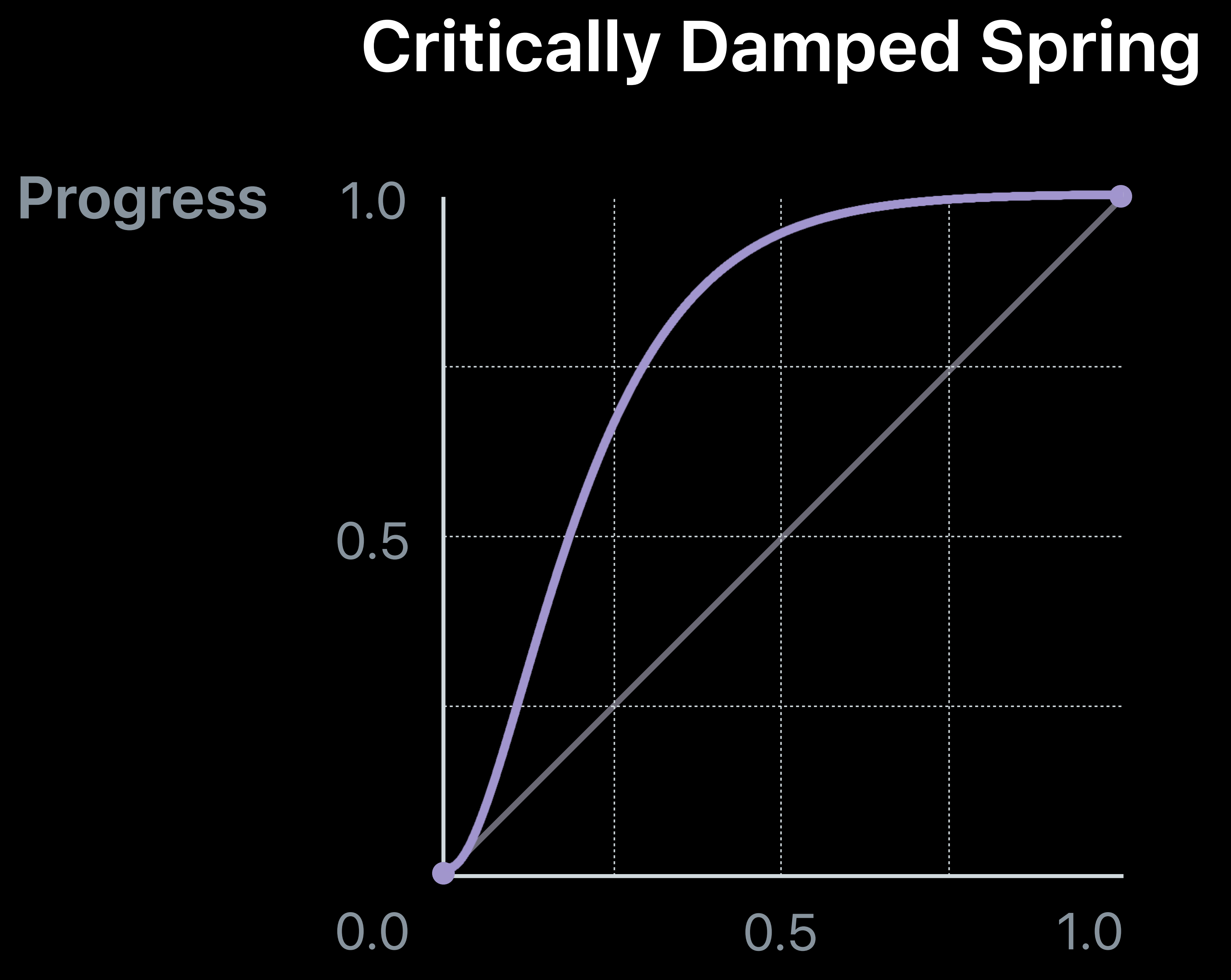


## Computing Translation

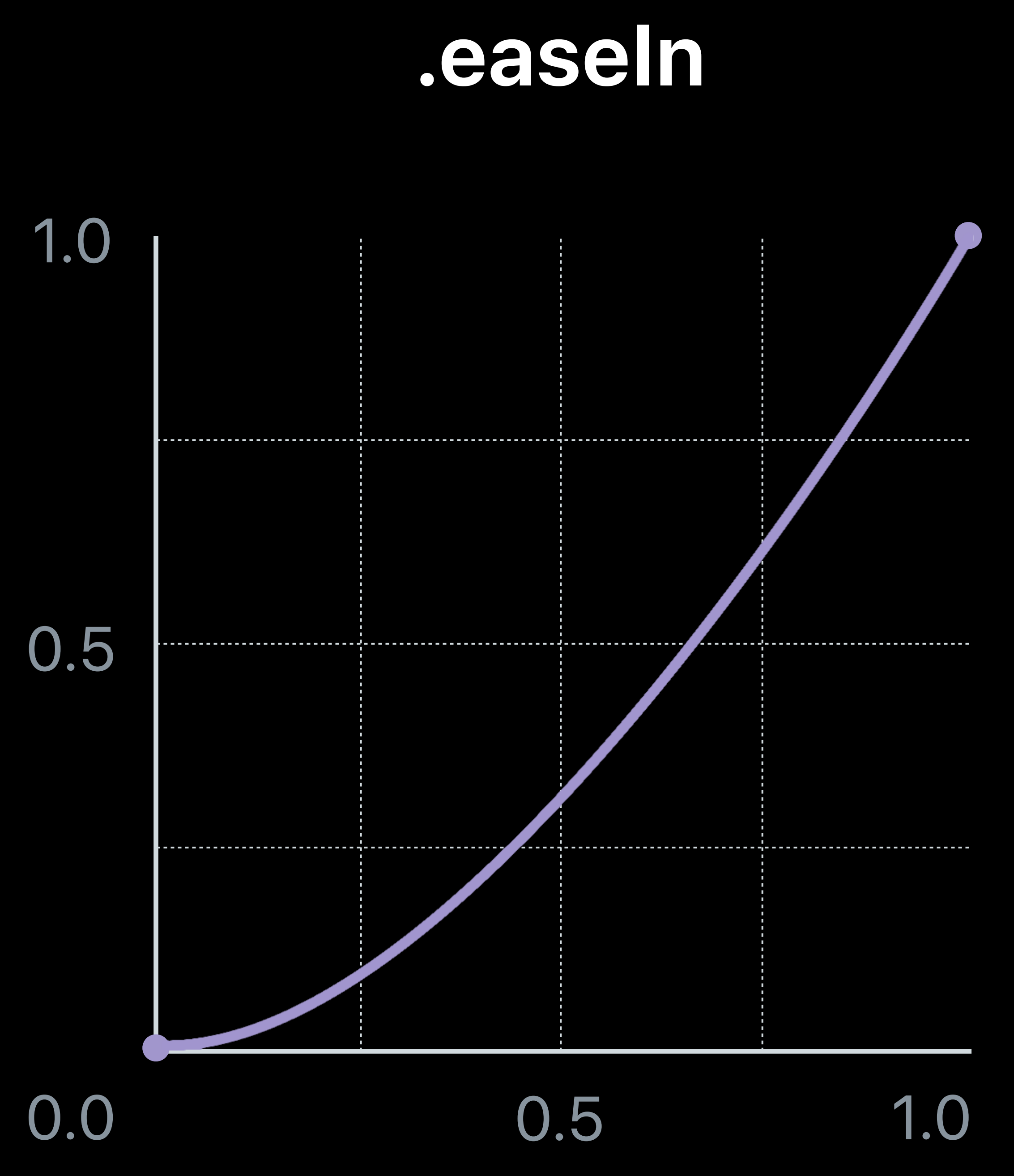


translation.y = Y - y′

# Animations

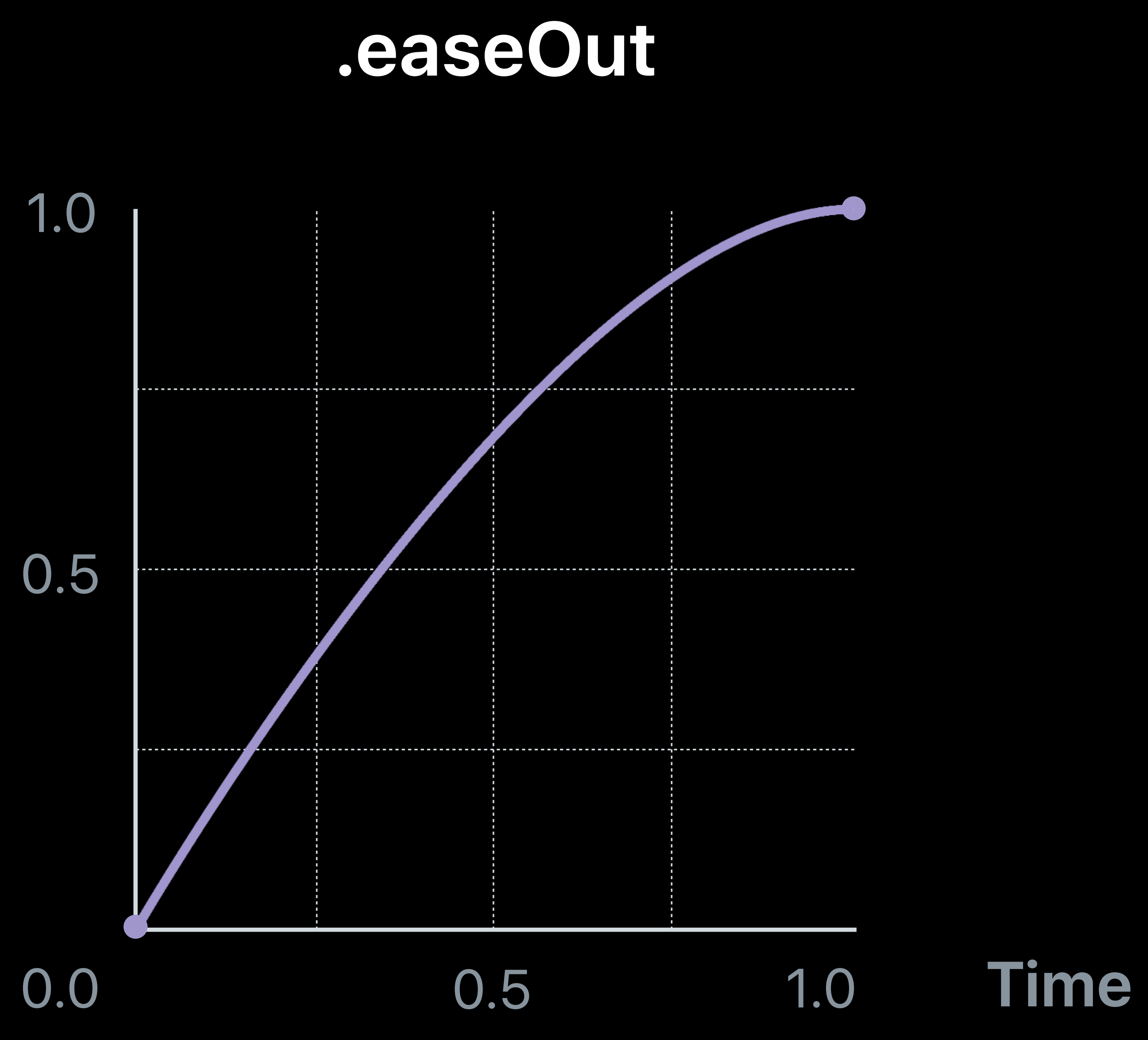


.transform



Incoming view .alpha

Non linear scrubbing



Outgoing view .alpha

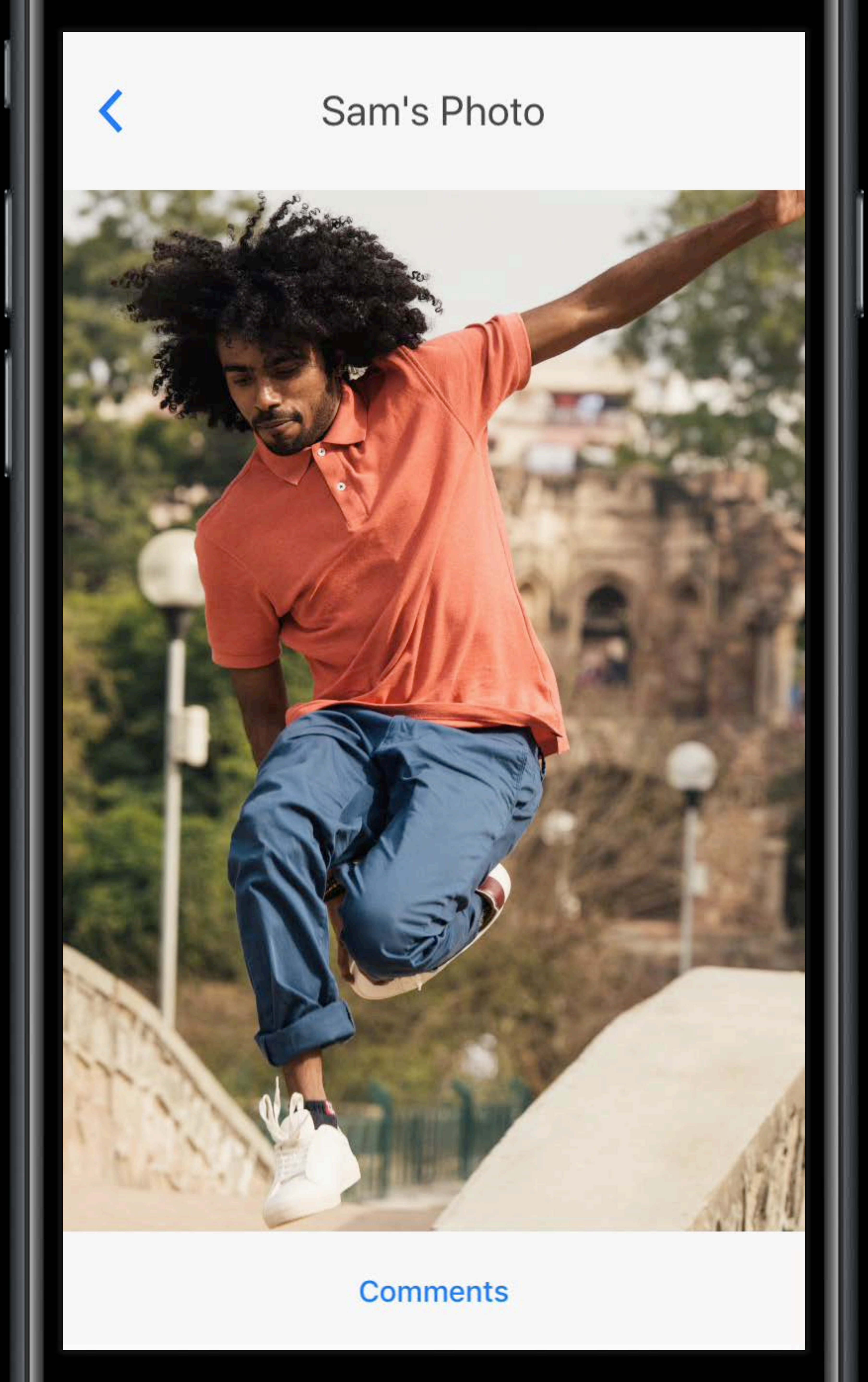
Non linear scrubbing

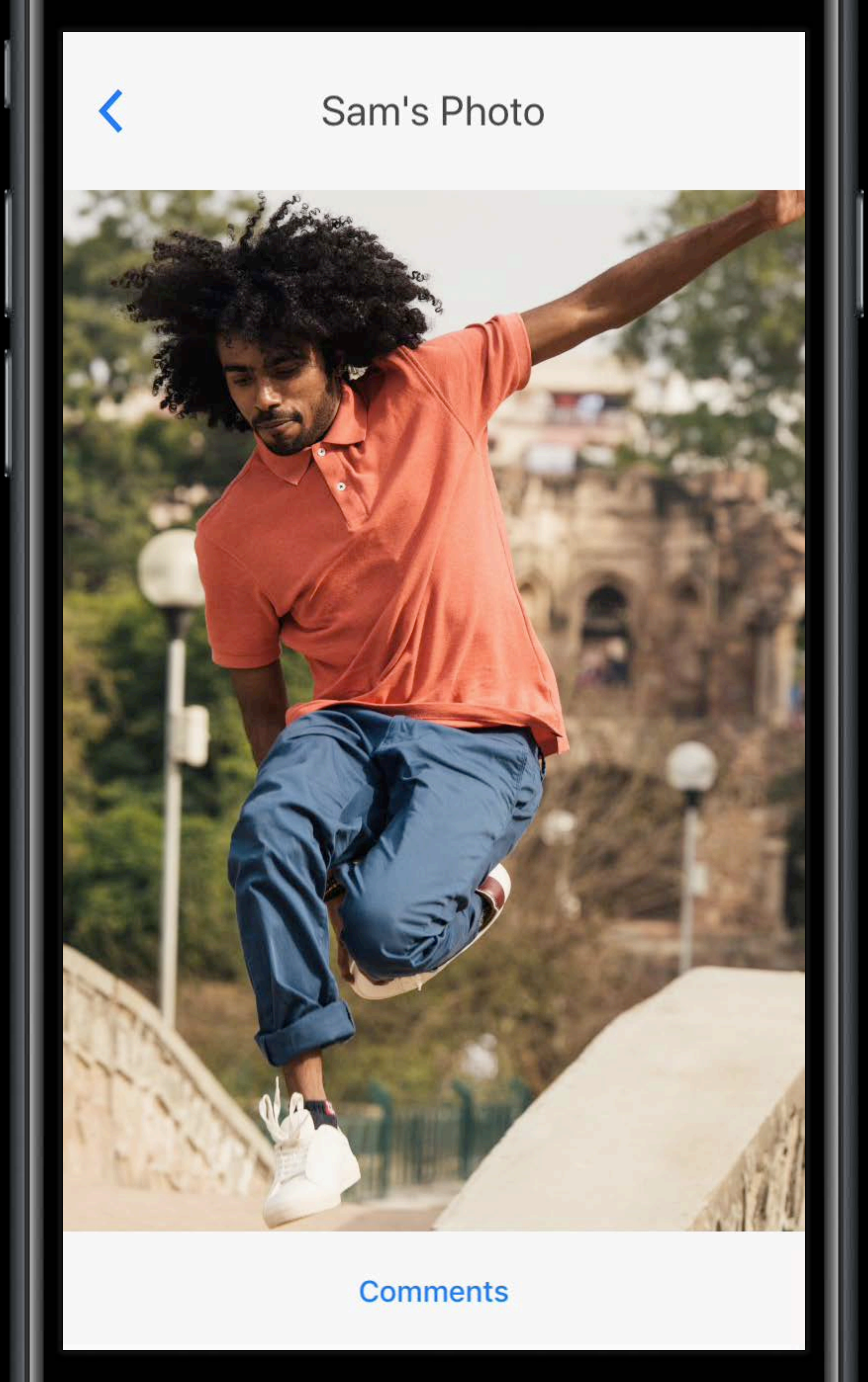
```swift
func animateTransitionIfNeeded(forState state: State, duration: TimeInterval) {
    // ...
    let transformAnimator = UIViewPropertyAnimator(duration: duration, dampingRatio: 1) {
        inLabel.transform = CGAffineTransform.identity
        outLabel.transform = inLabelScale.concatenating(inLabelTranslation)
    }
    // ...
    let inLabelAnimator = UIViewPropertyAnimator(duration: duration, curve: .easeIn) {
        inLabel.alpha = 1
    }
    inLabelAnimator.scrubsLinearly = false
    // ...
    let outLabelAnimator = UIViewPropertyAnimator(duration: duration, curve: .easeOut) {
        outLabel.alpha = 0
    }
    outLabelAnimator.scrubsLinearly = false
    // ...
}
```

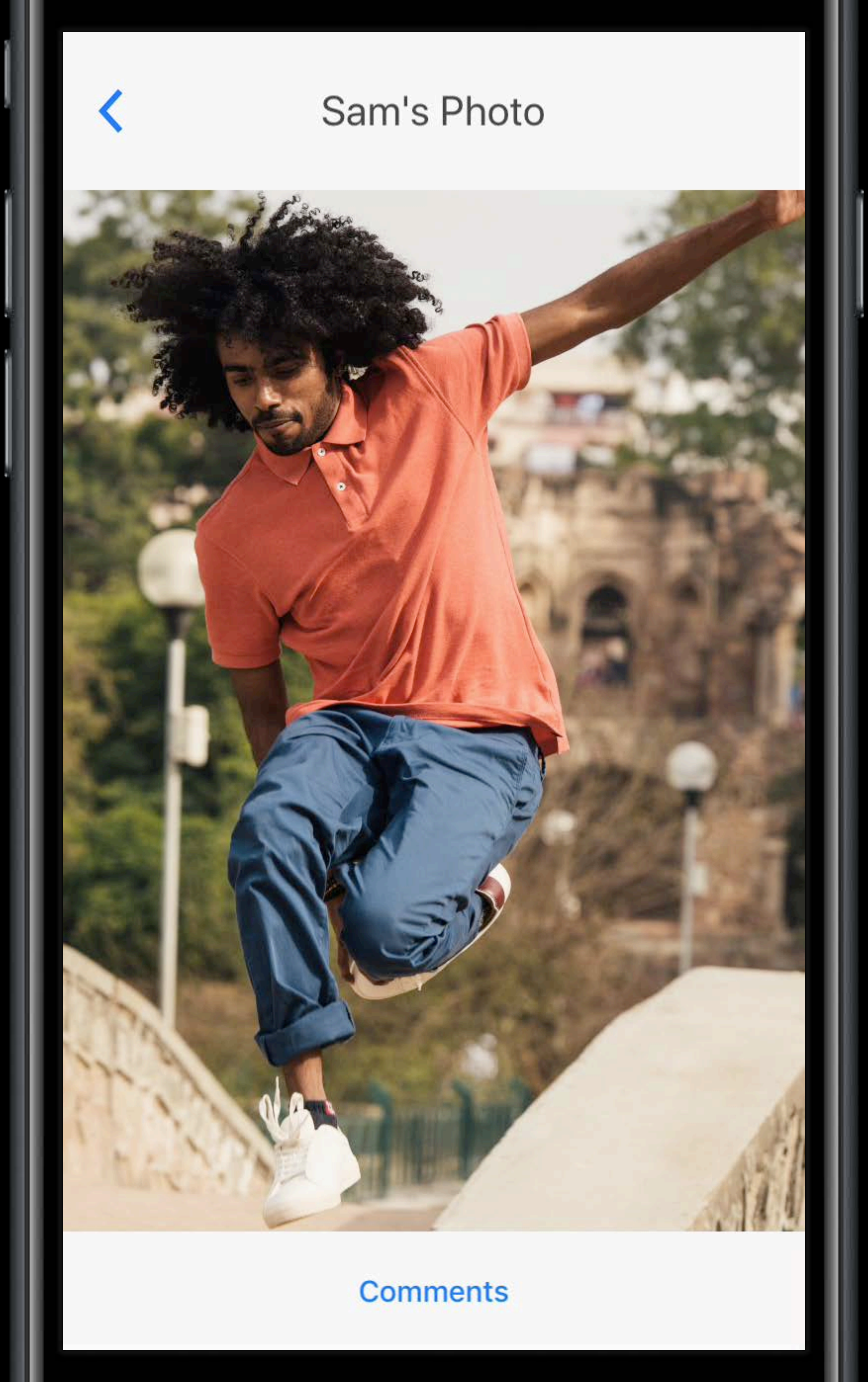
```swift
func animateTransitionIfNeeded(forState state: State, duration: TimeInterval) {
    // ...
    let transformAnimator = UIViewPropertyAnimator(duration: duration, dampingRatio: 1) {
        inLabel.transform = CGAffineTransform.identity
        outLabel.transform = inLabelScale.concatenating(inLabelTranslation)
    }
    // ...
    let inLabelAnimator = UIViewPropertyAnimator(duration: duration, curve: .easeIn) {
        inLabel.alpha = 1
    }
    inLabelAnimator.scrubsLinearly = false
    // ...
    let outLabelAnimator = UIViewPropertyAnimator(duration: duration, curve: .easeOut) {
        outLabel.alpha = 0
    }
    outLabelAnimator.scrubsLinearly = false
    // ...
}
```

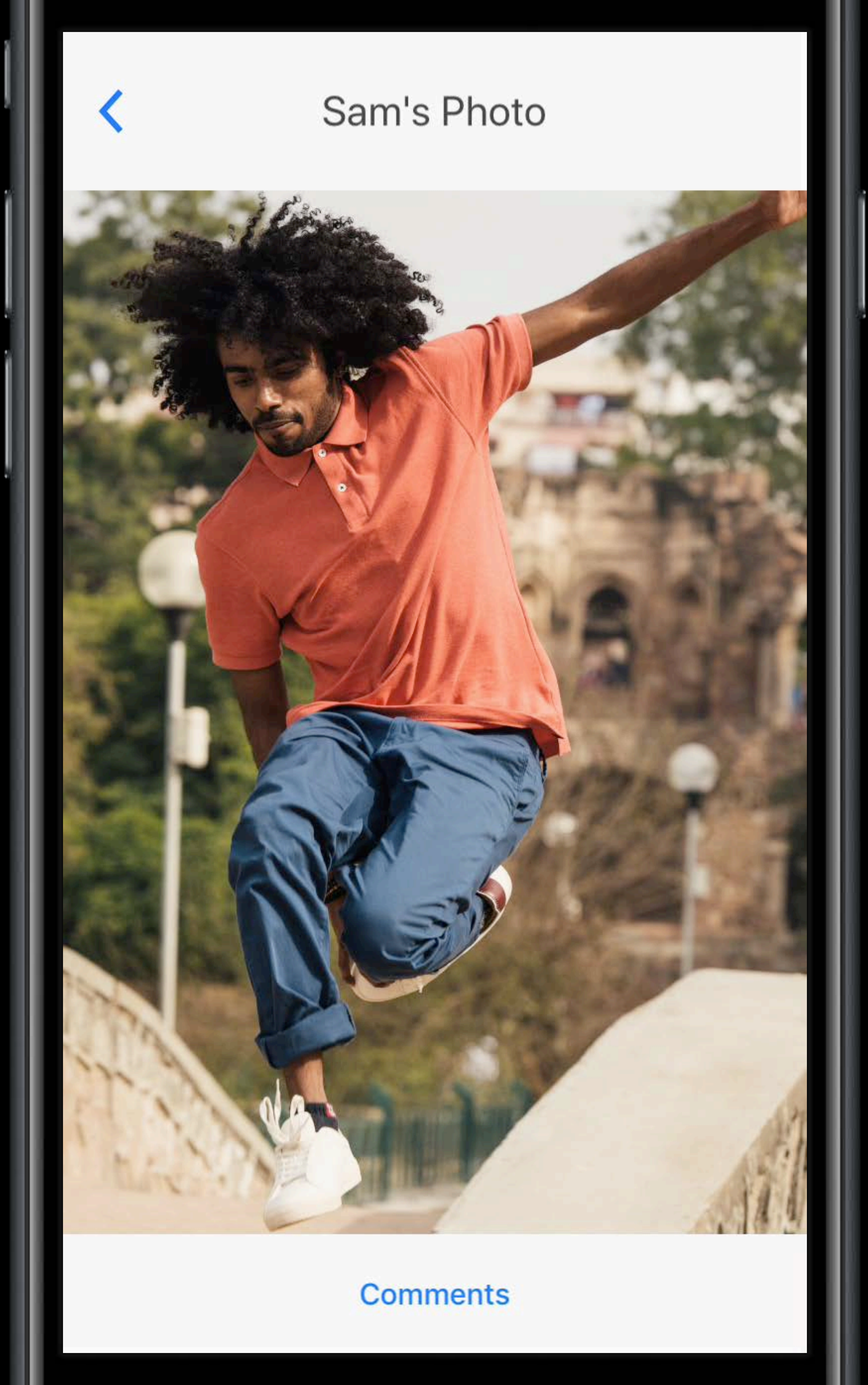
```swift
func animateTransitionIfNeeded(forState state: State, duration: TimeInterval) {
    // ...
    let transformAnimator = UIViewPropertyAnimator(duration: duration, dampingRatio: 1) {
        inLabel.transform = CGAffineTransform.identity
        outLabel.transform = inLabelScale.concatenating(inLabelTranslation)
    }
    // ...
    let inLabelAnimator = UIViewPropertyAnimator(duration: duration, curve: .easeIn) {
        inLabel.alpha = 1
    }
    inLabelAnimator.scrubsLinearly = false
    // ...
    let outLabelAnimator = UIViewPropertyAnimator(duration: duration, curve: .easeOut) {
        outLabel.alpha = 0
    }
    outLabelAnimator.scrubsLinearly = false
    // ...
}
```

```swift
func animateTransitionIfNeeded(forState state: State, duration: TimeInterval) {
    // ...
    let transformAnimator = UIViewPropertyAnimator(duration: duration, dampingRatio: 1) {
        inLabel.transform = CGAffineTransform.identity
        outLabel.transform = inLabelScale.concatenating(inLabelTranslation)
    }
    // ...
    let inLabelAnimator = UIViewPropertyAnimator(duration: duration, curve: .easeIn) {
        inLabel.alpha = 1
    }
    inLabelAnimator.scrubsLinearly = false
    // ...
    let outLabelAnimator = UIViewPropertyAnimator(duration: duration, curve: .easeOut) {
        outLabel.alpha = 0
    }
    outLabelAnimator.scrubsLinearly = false
    // ...
}
```


Sam's Photo
Comments


Sam's Photo
Comments

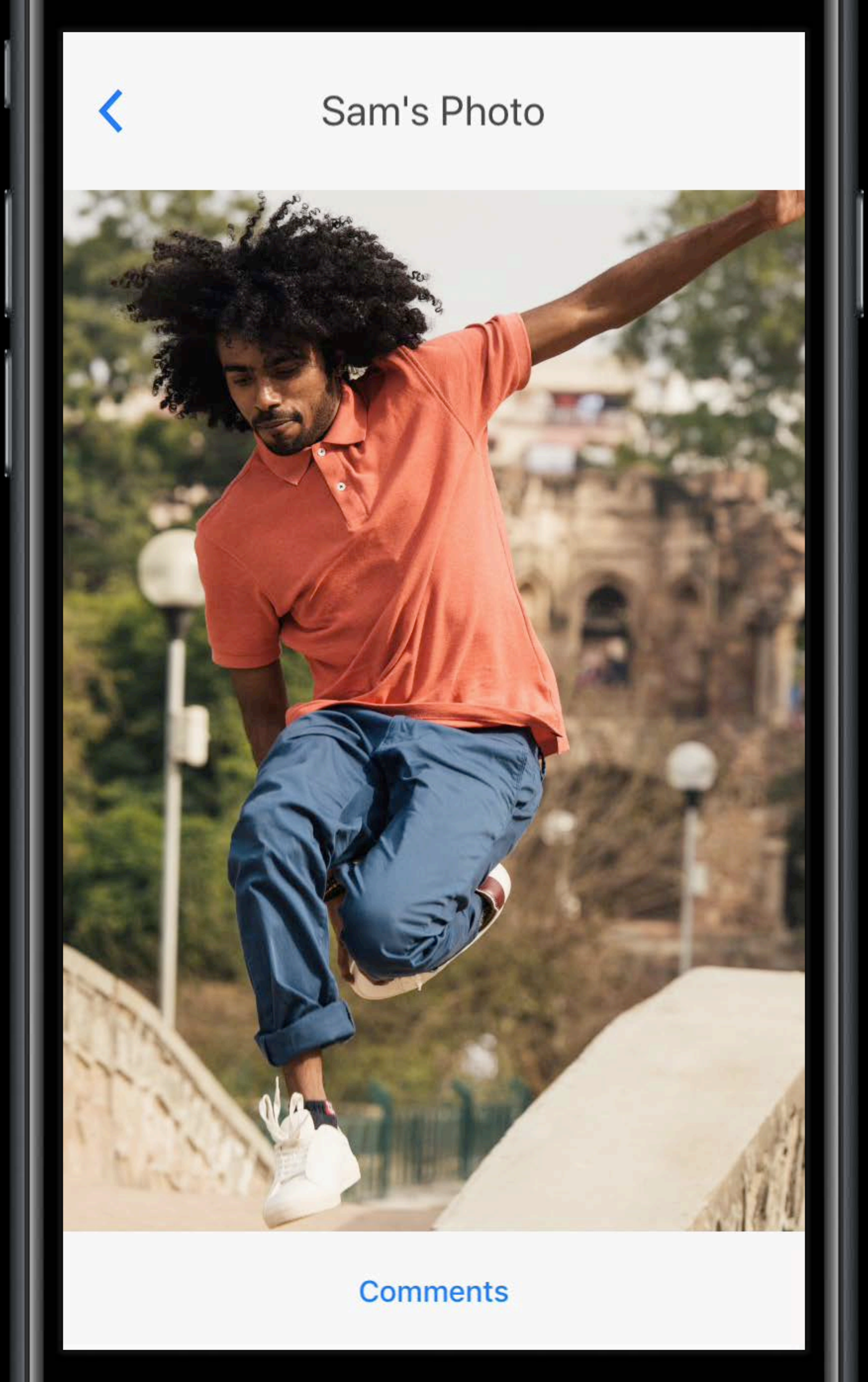

Sam's Photo
Comments



Sam's Photo

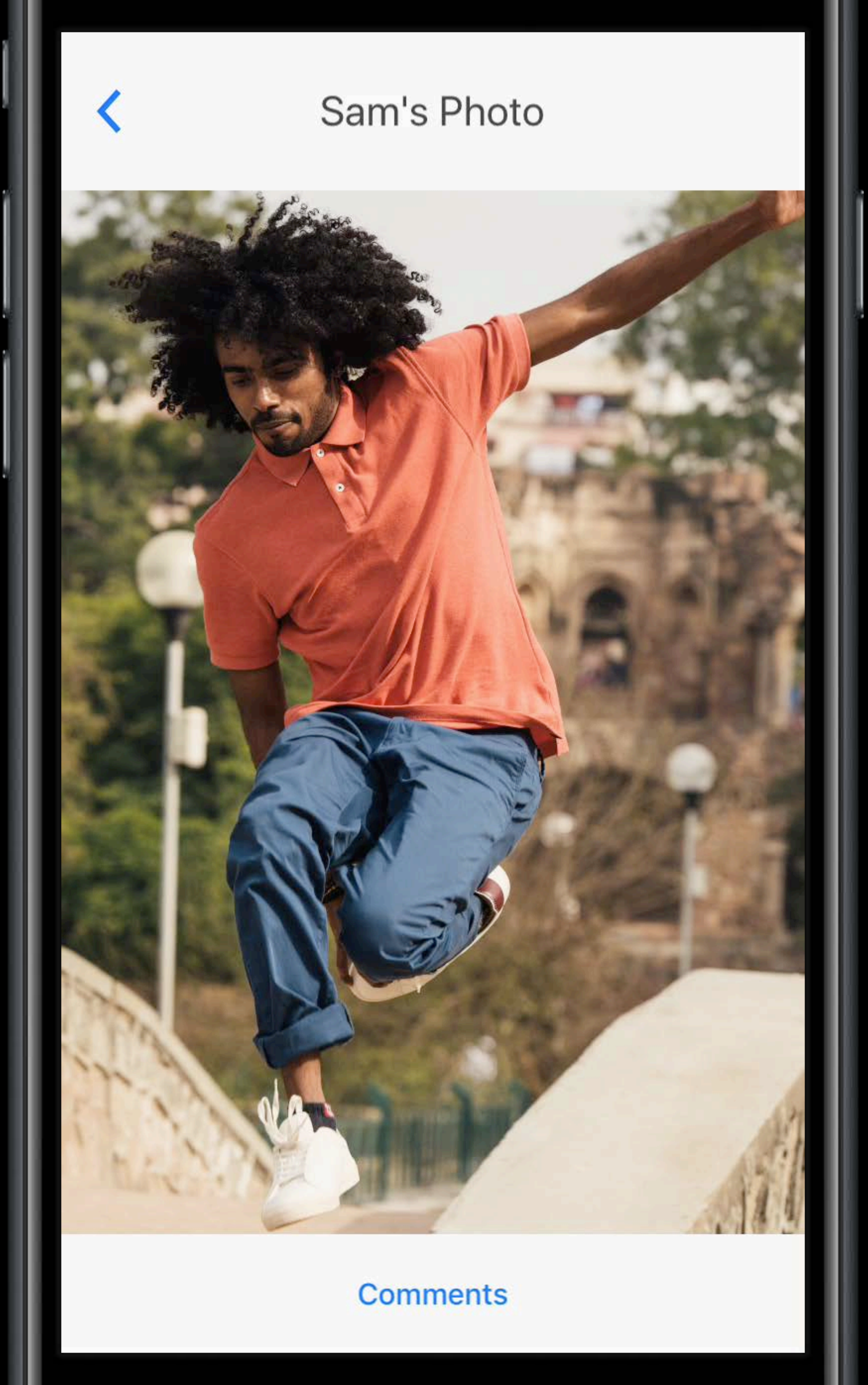
Comments

Sam's Photo
Comments



Sam's Photo

Comments

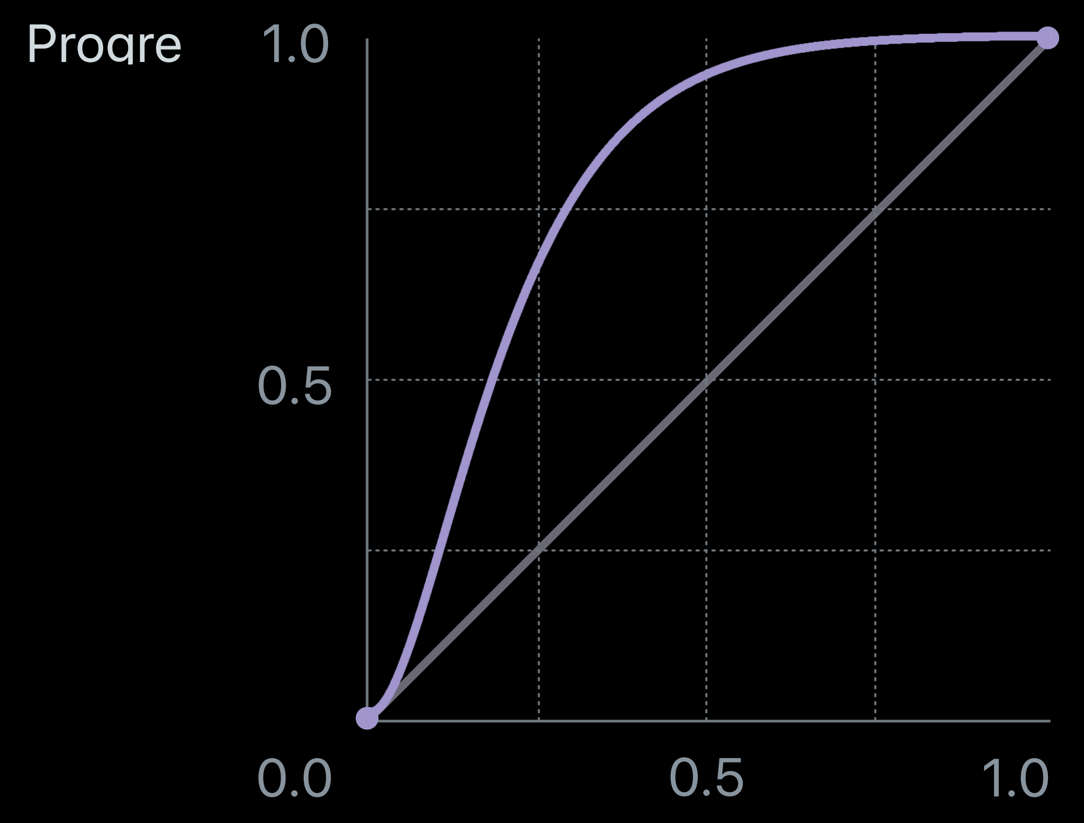


Sam's Photo

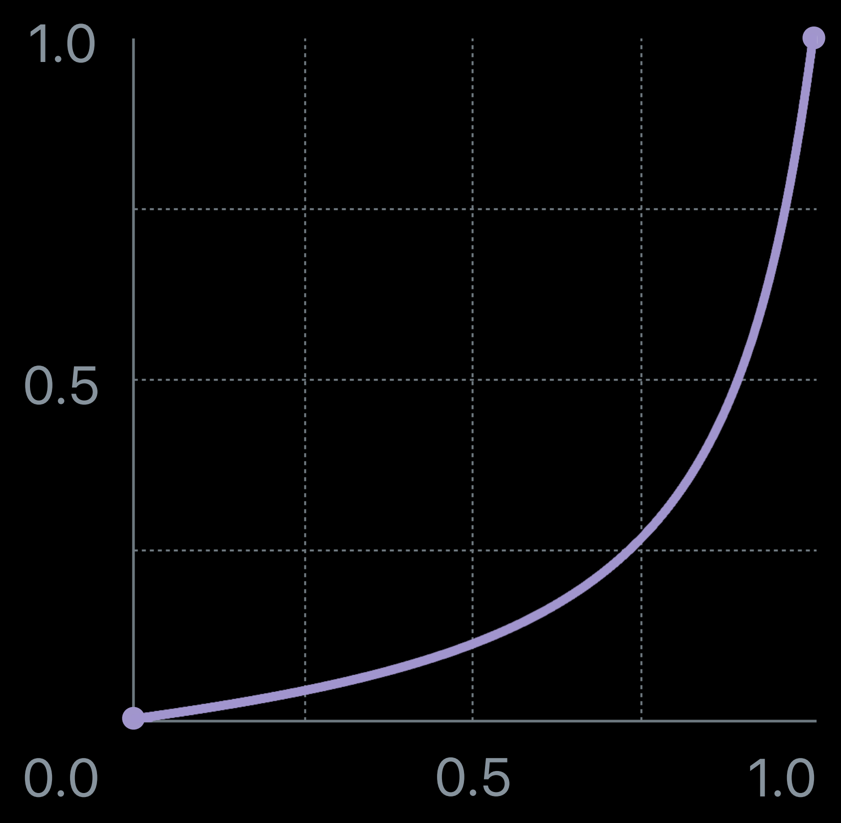
Comments

Sam's Photo
Comments

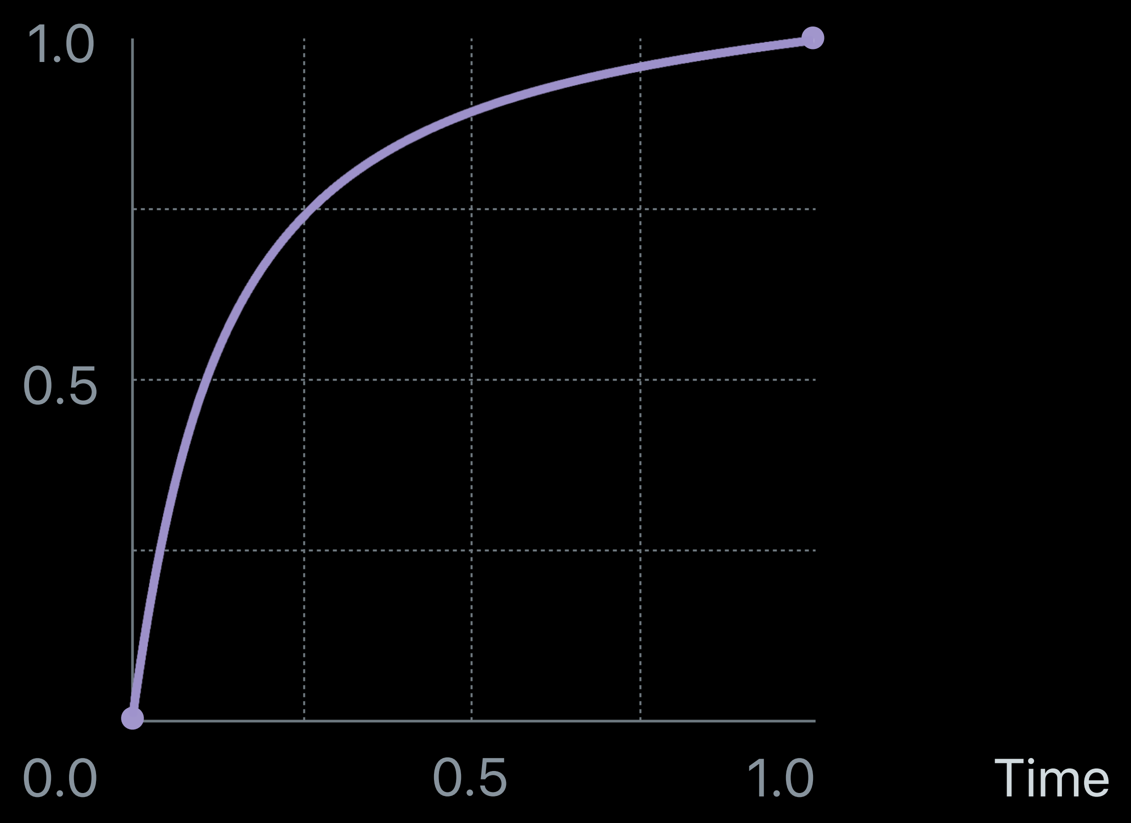
Comments View



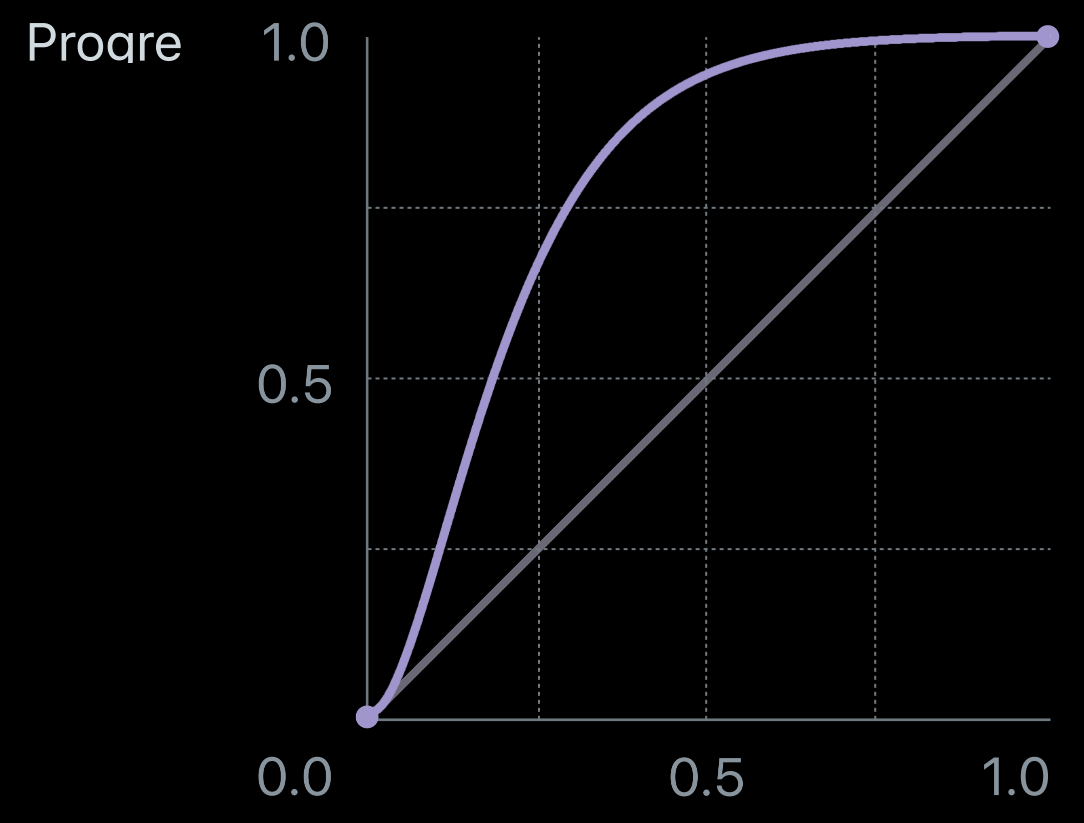
Blur In



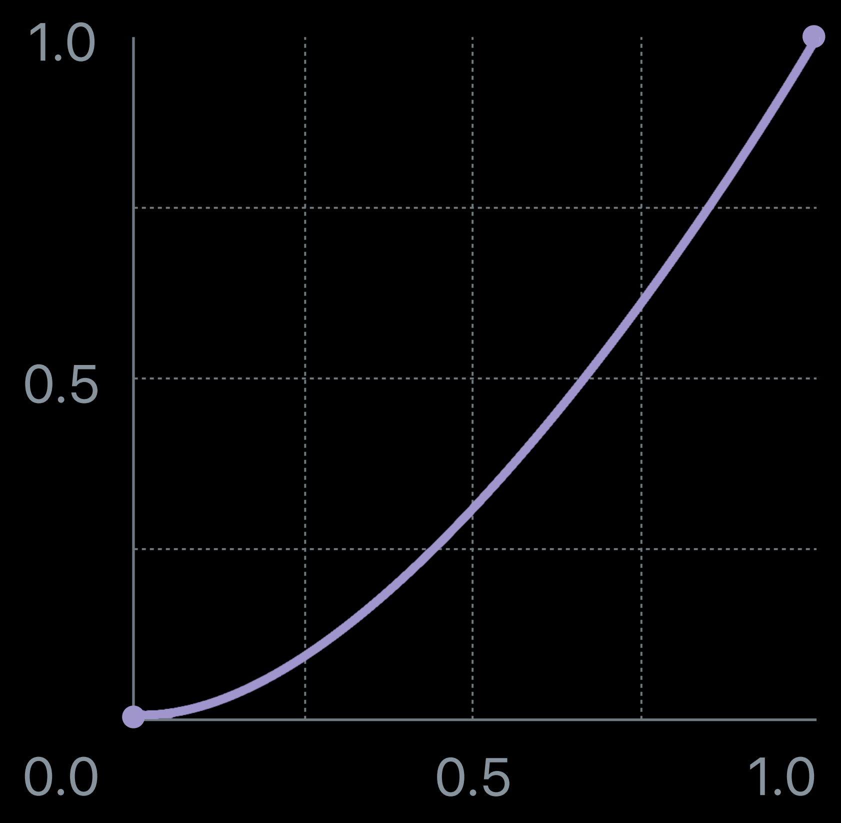
Blur Out



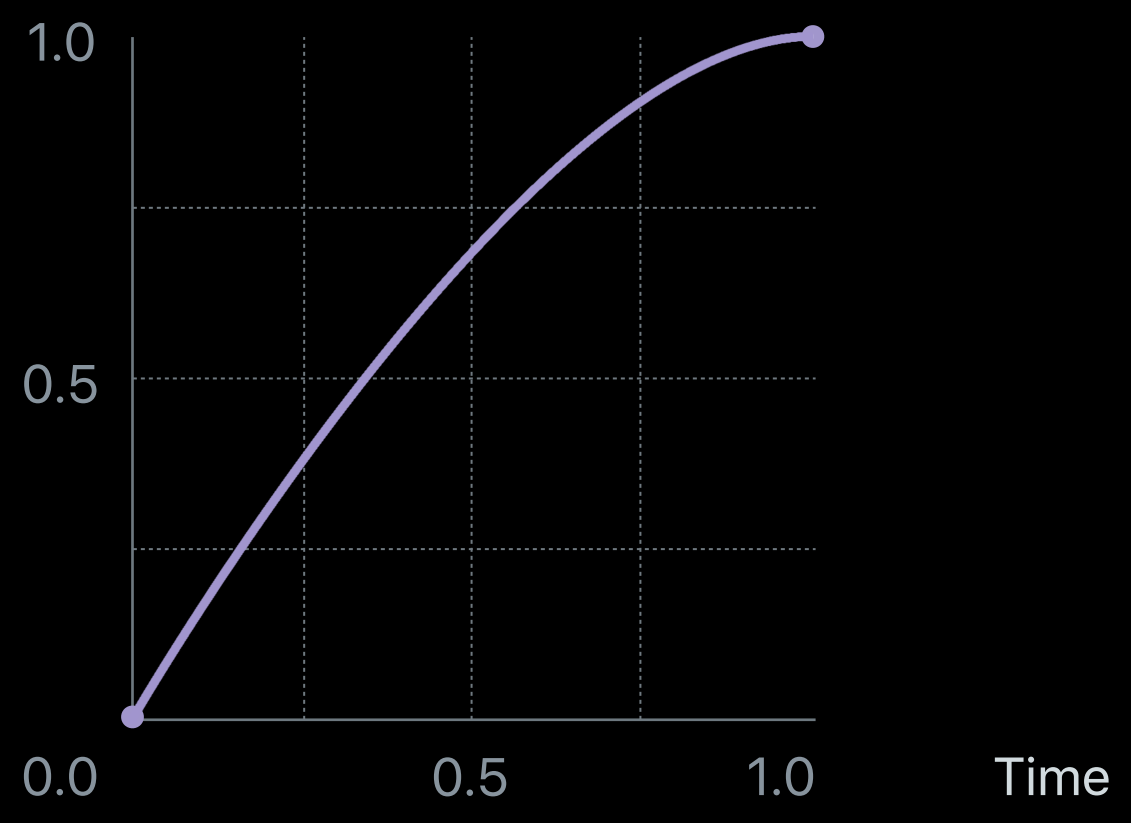
Label Transform



Label Alpha In

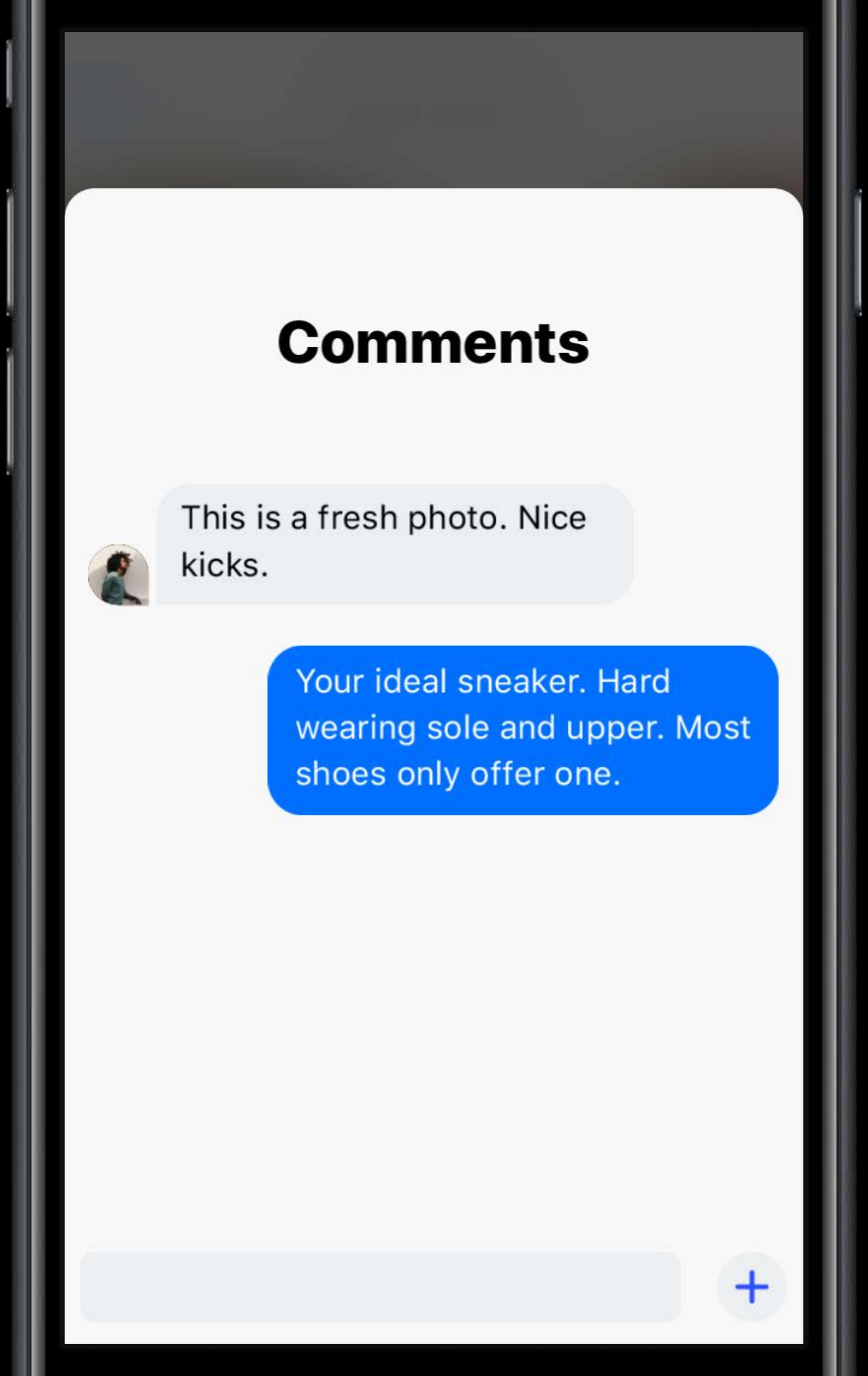


Label Alpha Out



## Tips and Tricks

## Animating Corner Radius



Comments

This is a fresh photo. Nice kicks.

Your ideal sneaker. Hard wearing sole and upper. Most shoes only offer one.

## .cornerRadius

Now animatable in UIKit

NEW

**CALayer**

```
var .cornerRadius: CGFloat
```

## .cornerRadius

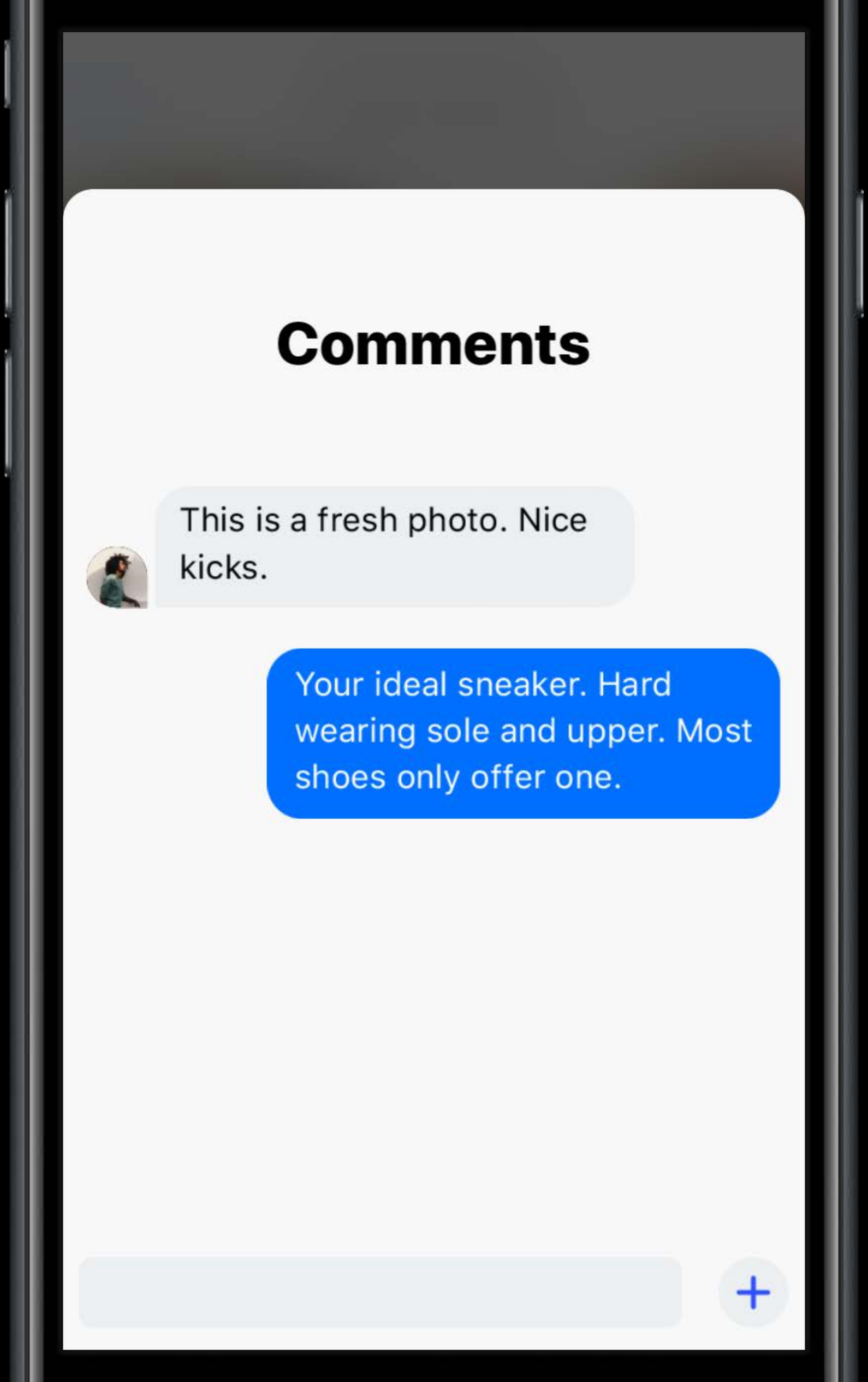
Now animatable in UIKit

NEW

**CALayer**

```
var .cornerRadius: CGFloat
```

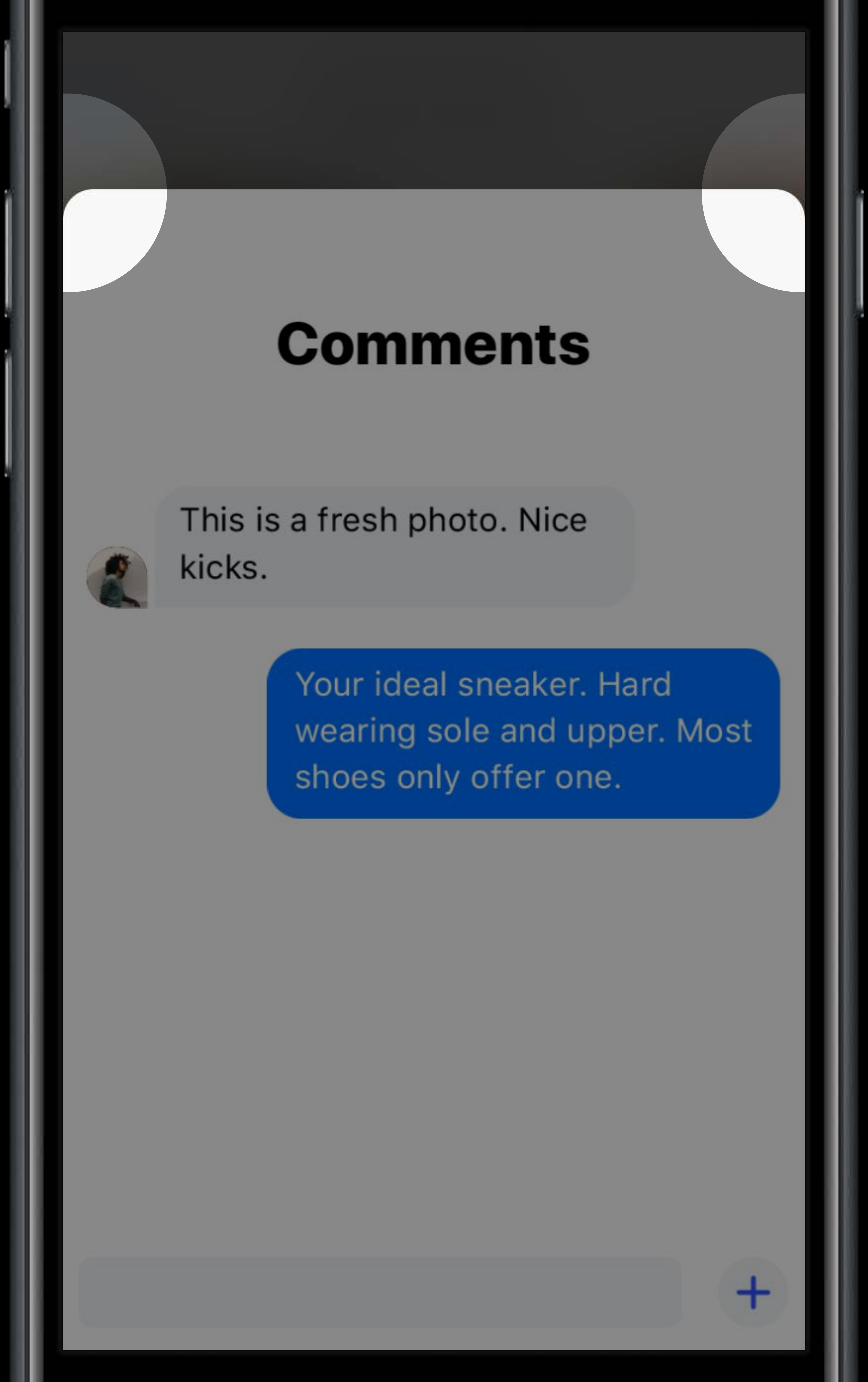
```
circle.clipsToBounds = true
UIViewPropertyAnimator(duration: 1, curve: .linear) {
    circle.layer.cornerRadius = 12
}.startAnimation()
```



Comments

This is a fresh photo. Nice kicks.

Your ideal sneaker. Hard wearing sole and upper. Most shoes only offer one.



Comments

This is a fresh photo. Nice kicks.

Your ideal sneaker. Hard wearing sole and upper. Most shoes only offer one.

# .maskedCorners

New in iOS 11

NEW

**CALayer**

```
var .maskedCorners: CACornerMask
```

## .maskedCorners

New in iOS 11

NEW

**CALayer**

```
var .maskedCorners: CACornerMask
```

# .maskedCorners

New in iOS 11

NEW

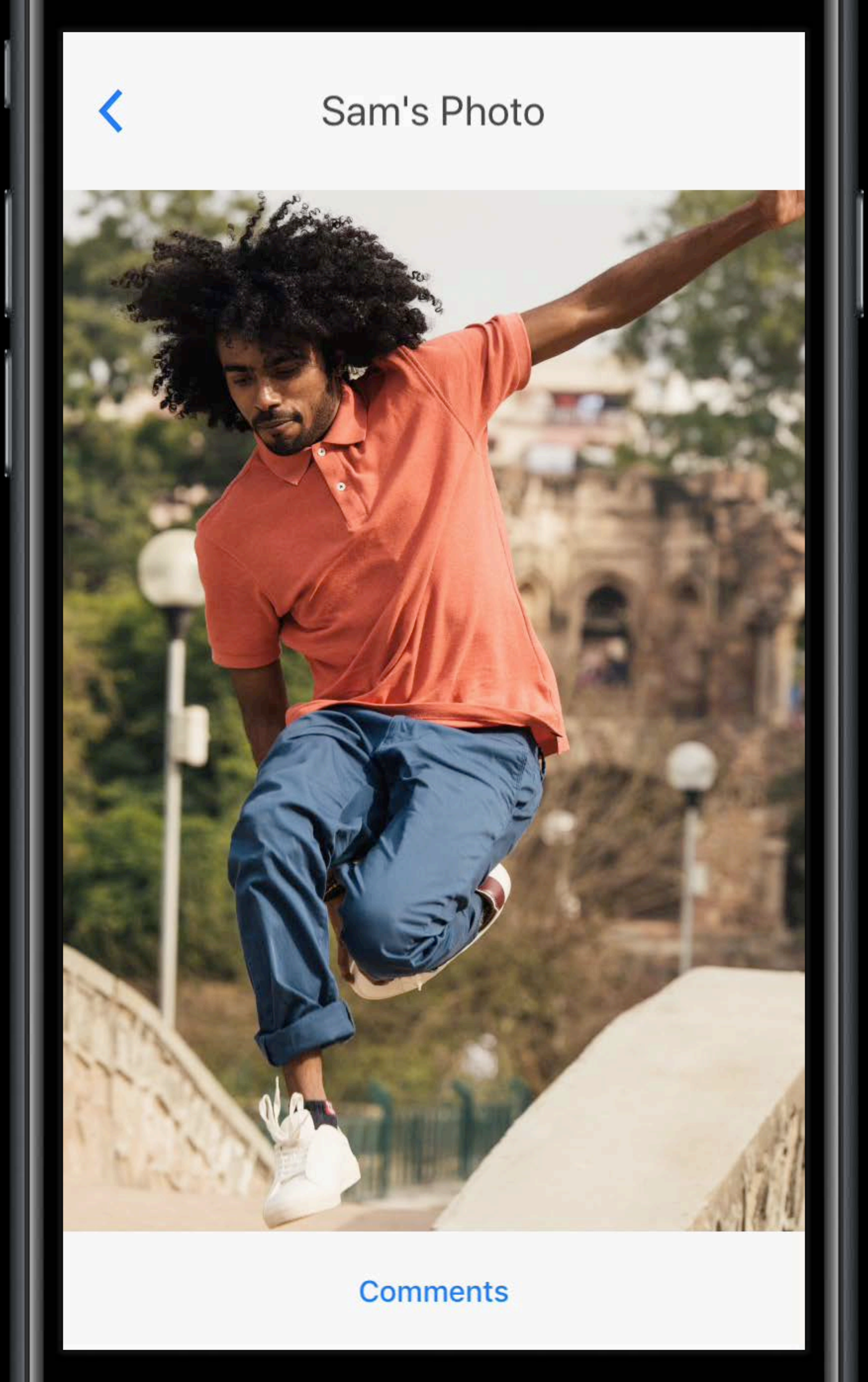
**CALayer**

```
var .maskedCorners: CACornerMask
```

```
circle.layer.maskedCorners = [.layerMinXMinYCorner, .layerMaxXMinYCorner]
```

```swift
func animateTransitionIfNeeded(forState state: State, duration: TimeInterval) {
    // ...
    let cornerAnimator = UIViewPropertyAnimator(duration: duration, curve: .linear) {
        switch state {
        case .Expanded:
            self.control.layer.cornerRadius = 12
        case .Collapsed:
            self.control.layer.cornerRadius = 0
        }
    }
    // ...
}
```

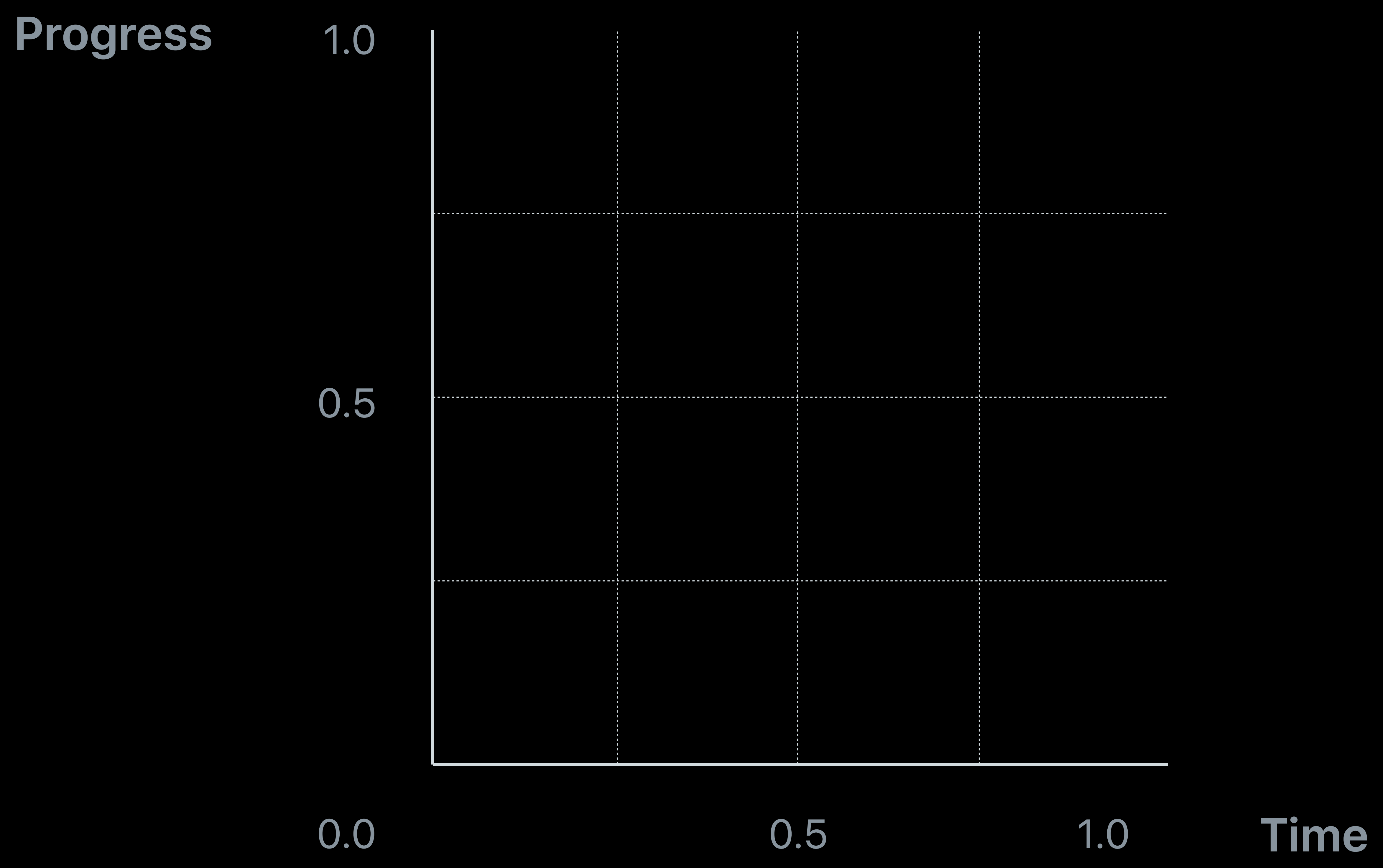
Sam's Photo
Comments


Sam's Photo
Comments

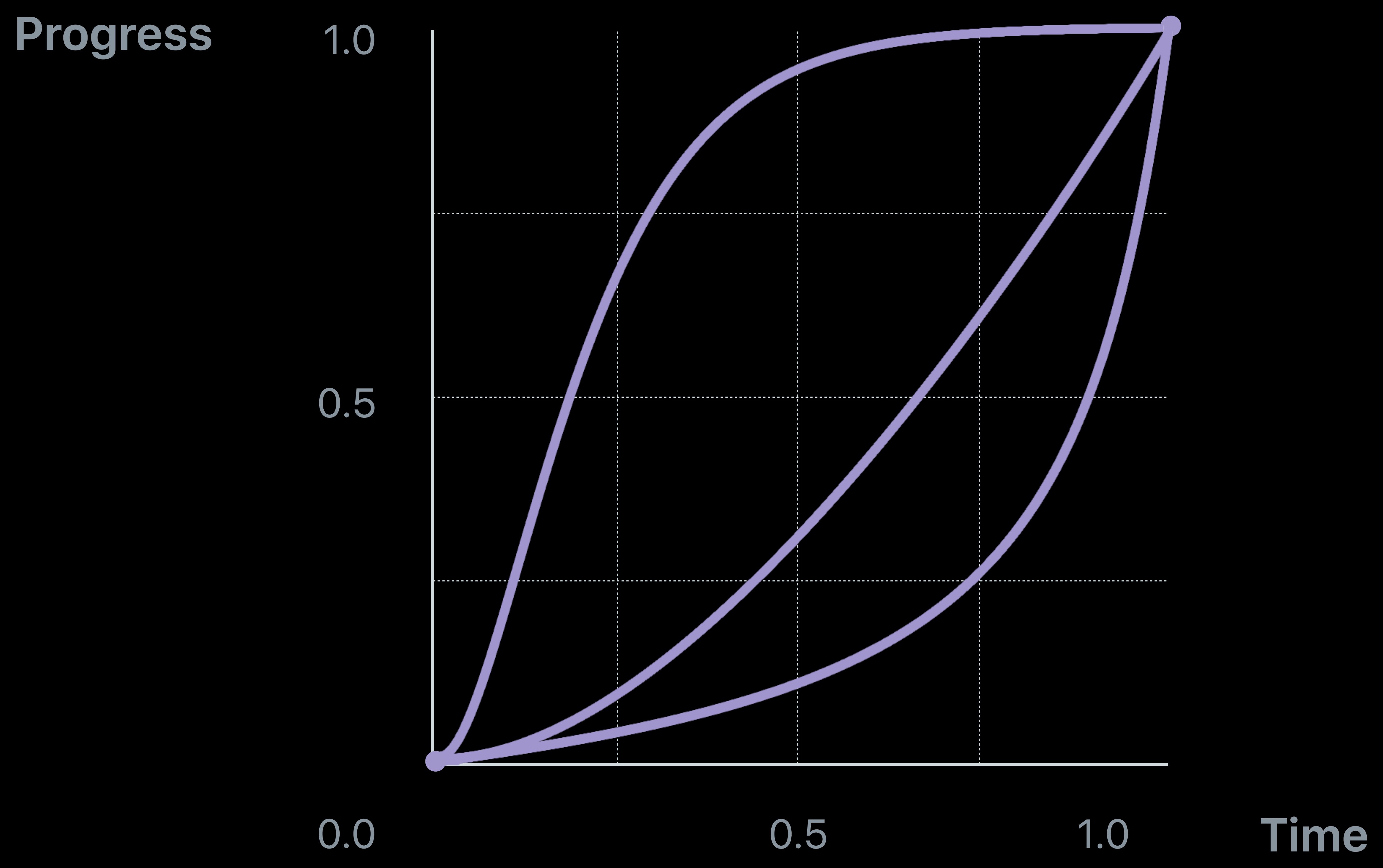
Sam's Photo
Comments

Sam's Photo
Comments

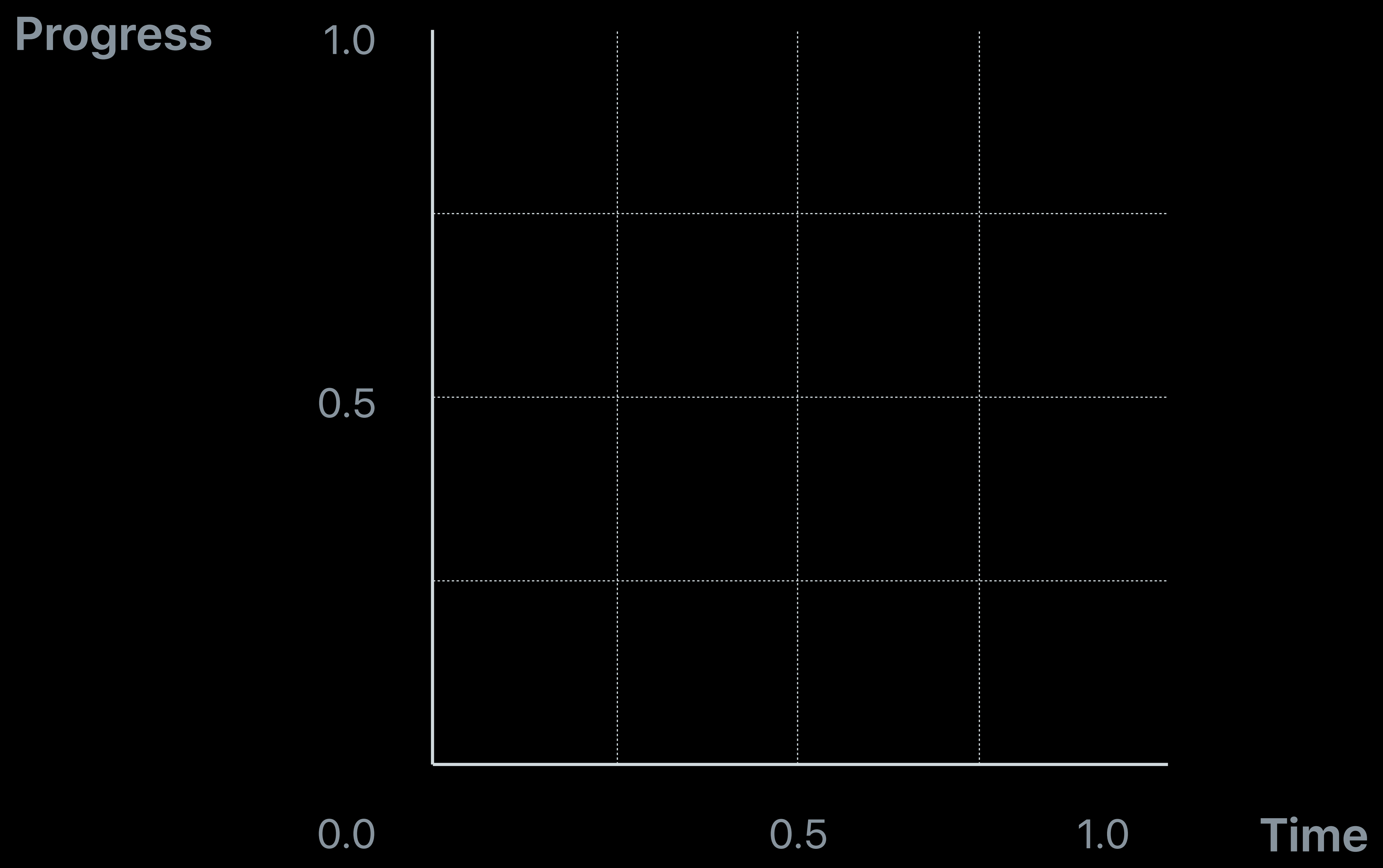
## Component Timing



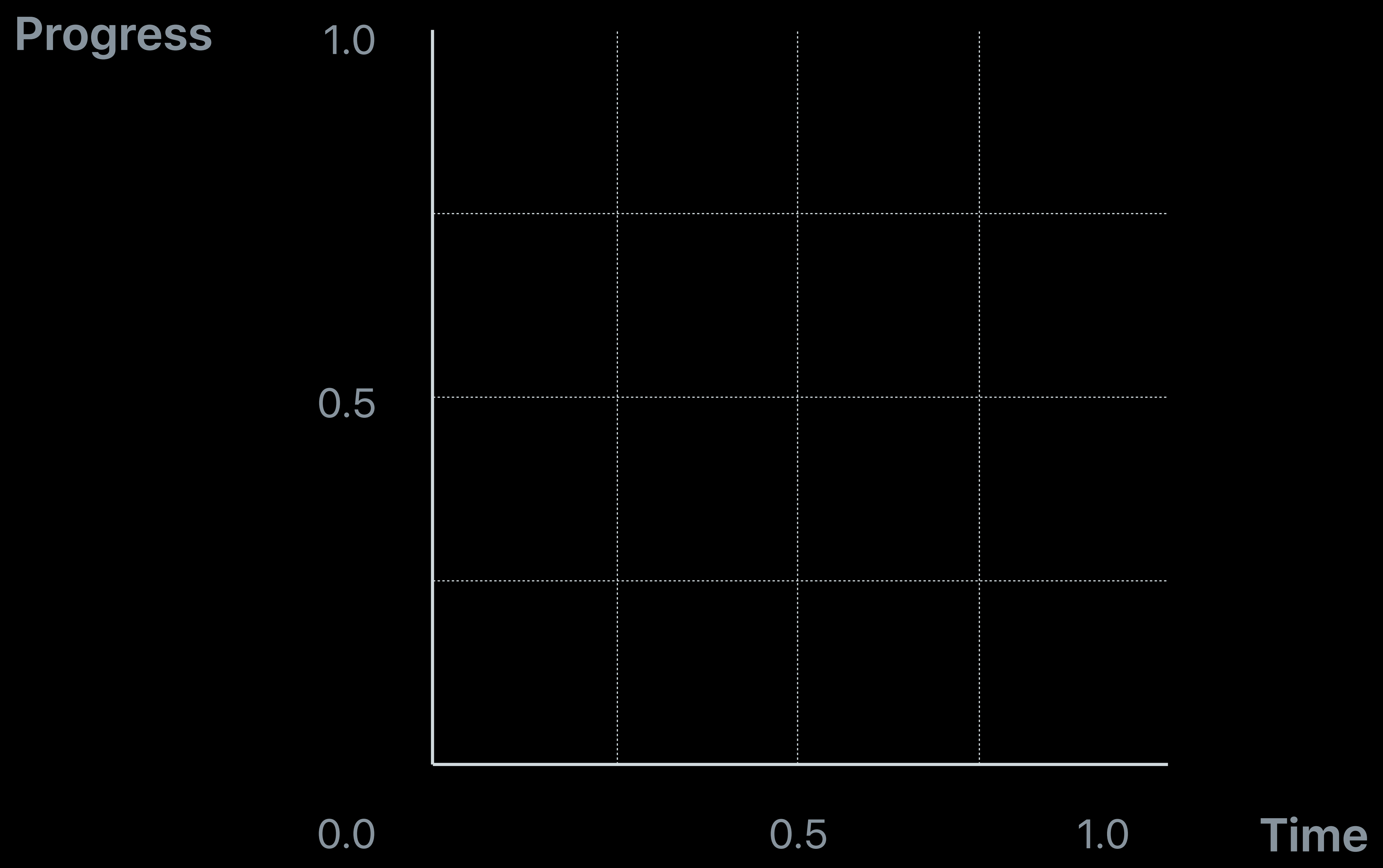
# Component Timing



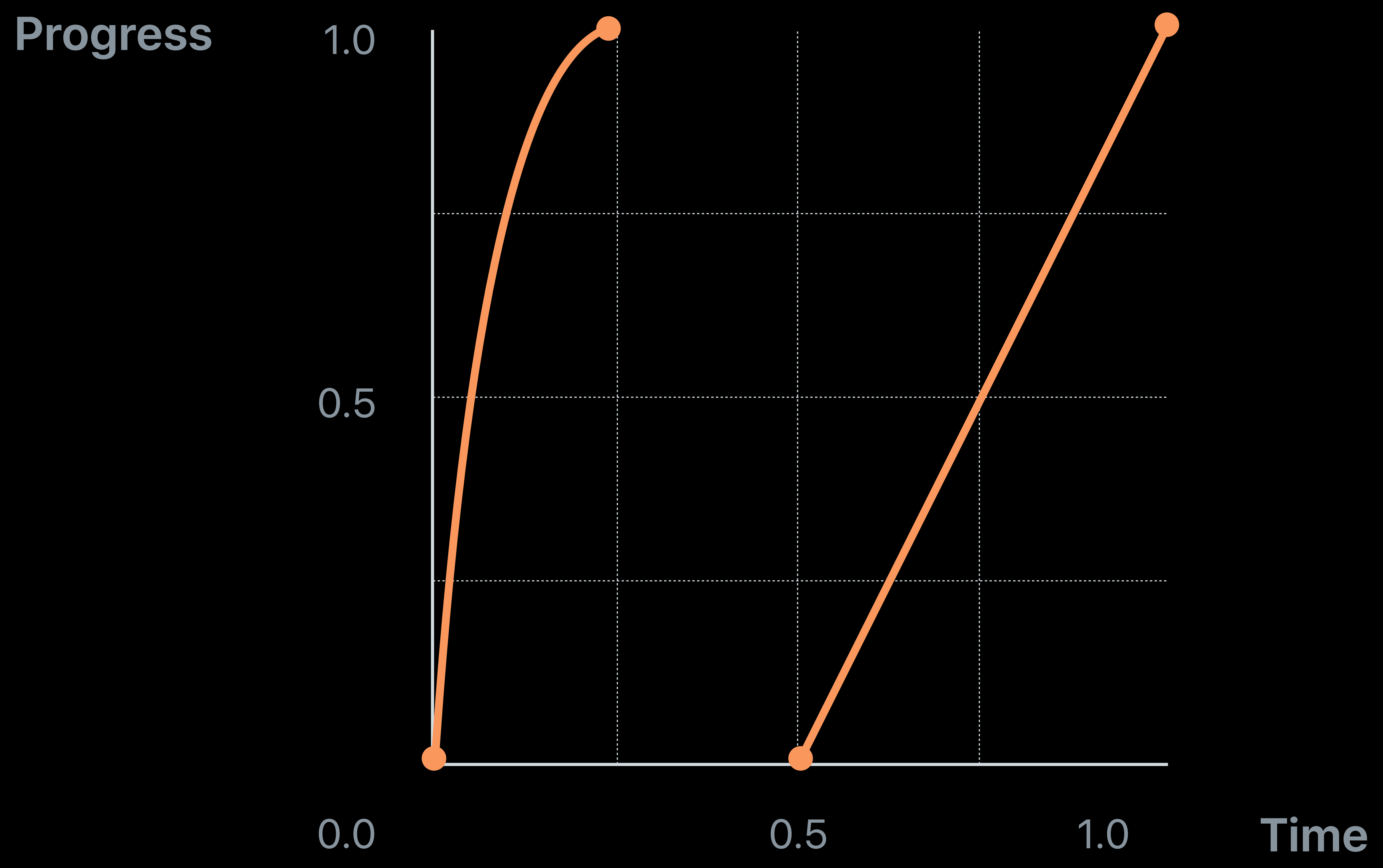
## Component Timing



## Component Timing

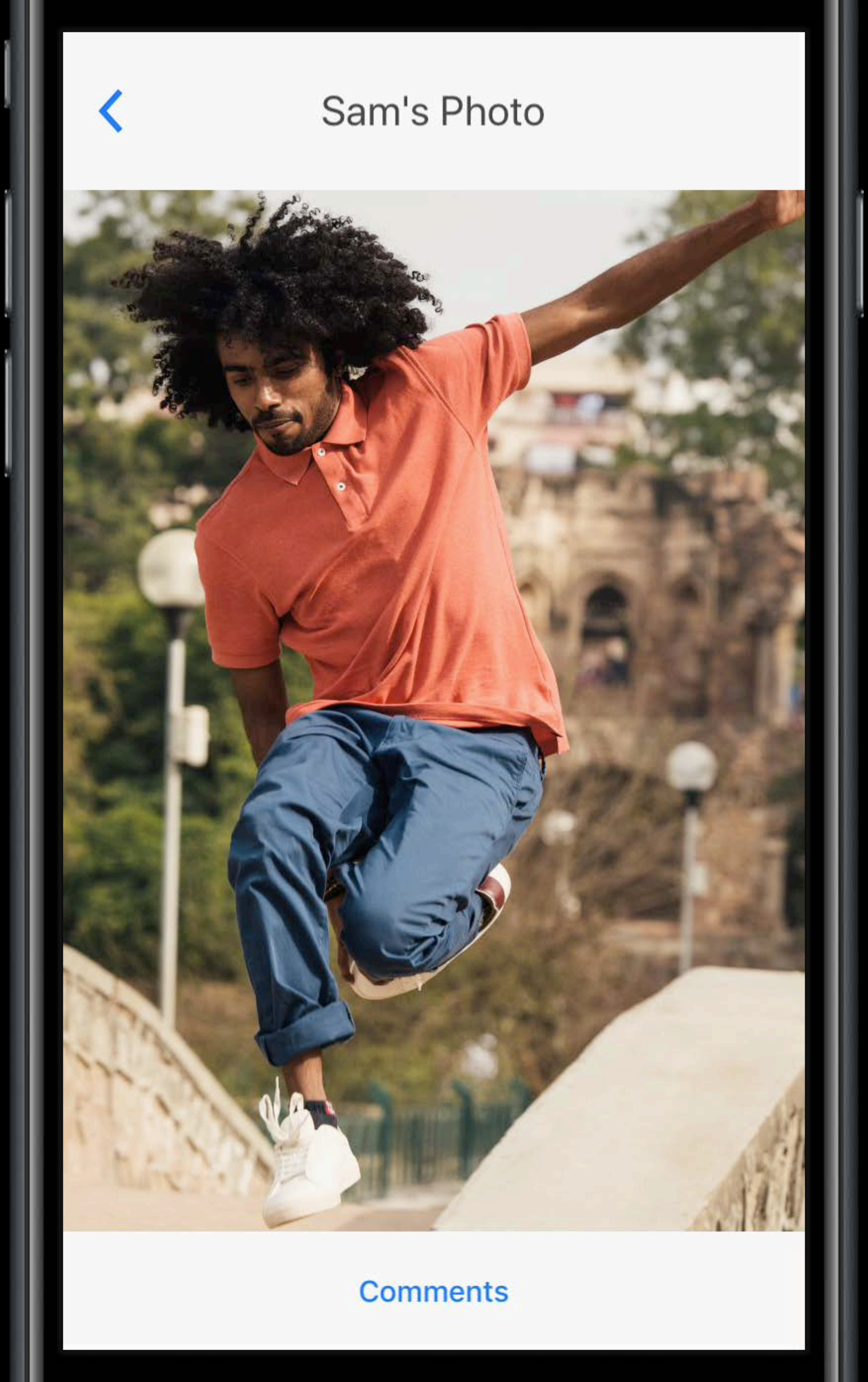


# Component Timing



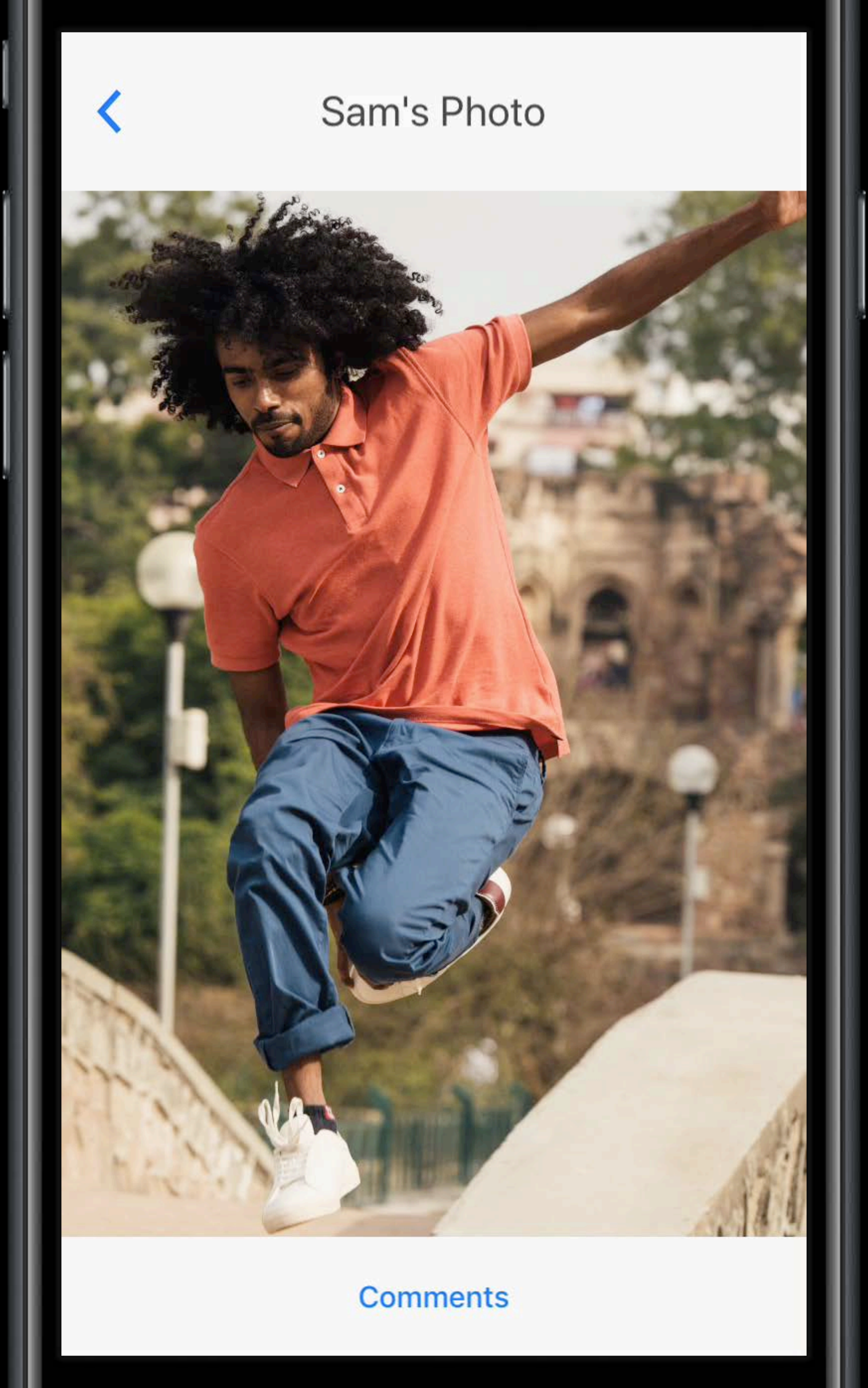
Sam's Photo

Comments


Sam's Photo
Comments

Sam's Photo
Comments

Sam's Photo
Comments


Sam's Photo
Comments

Sam's Photo
Comments

## Keyframe Animations

## Keyframe Animations

`UIView`

```
func animateKeyframes(withDuration duration: TimeInterval, delay: TimeInterval,
                      options: ..., animations: ..., completion: …)

func addKeyframe(withRelativeStartTime frameStartTime: Double,
                 relativeDuration frameDuration: Double,
                 animations: ...)
```

## Keyframe Animations

`UIView`

```
func animateKeyframes(withDuration duration: TimeInterval, delay: TimeInterval,
                      options: ..., animations: ..., completion: …)

func addKeyframe(withRelativeStartTime frameStartTime: Double,
                 relativeDuration frameDuration: Double,
                 animations: ...)
```

```swift
func animateTransitionIfNeeded(forState state: State, duration: TimeInterval) {
    // ...
    let buttonAnimator = UIViewPropertyAnimator(duration: duration, curve: .linear) {
        UIView.animateKeyframes(withDuration: 0.0, delay: 0.0, options: [], animations: {
            switch state {
            case .Expanded:
                UIView.addKeyframe(withRelativeStartTime: 0.5, relativeDuration: 0.5) {
                    // Start with delay and finish with rest of animations
                    detailsButton.alpha = 1
                })
            case .Collapsed:
                UIView.addKeyframe(withRelativeStartTime: 0.0, relativeDuration: 0.5) {
                    // Start immediately and finish in half the time
                    detailsButton.alpha = 0
                })
            }
        }, completion: nil)
    }
}
```

```swift
func animateTransitionIfNeeded(forState state: State, duration: TimeInterval) {
    // ...
    let buttonAnimator = UIViewPropertyAnimator(duration: duration, curve: .linear) {
        UIView.animateKeyframes(withDuration: 0.0, delay: 0.0, options: [], animations: {
            switch state {
            case .Expanded:
                UIView.addKeyframe(withRelativeStartTime: 0.5, relativeDuration: 0.5) {
                    // Start with delay and finish with rest of animations
                    detailsButton.alpha = 1
                })
            case .Collapsed:
                UIView.addKeyframe(withRelativeStartTime: 0.0, relativeDuration: 0.5) {
                    // Start immediately and finish in half the time
                    detailsButton.alpha = 0
                })
            }
        }, completion: nil)
    }
}
```

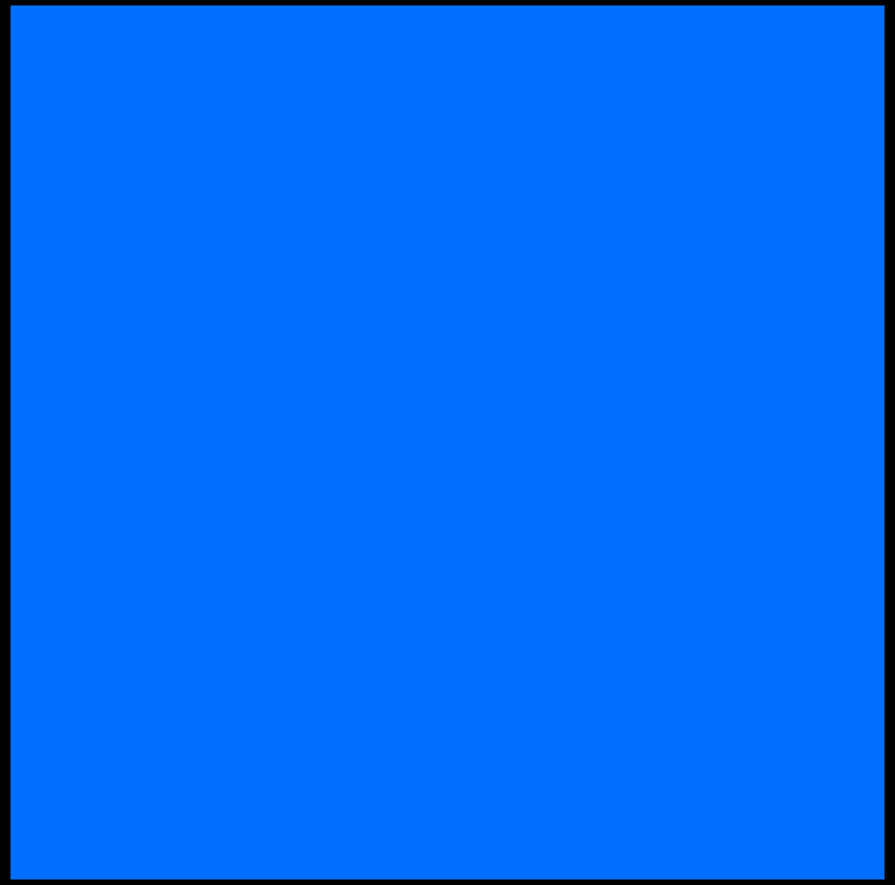
```swift
func animateTransitionIfNeeded(forState state: State, duration: TimeInterval) {
    // ...
    let buttonAnimator = UIViewPropertyAnimator(duration: duration, curve: .linear) {
        UIView.animateKeyframes(withDuration: 0.0, delay: 0.0, options: [], animations: {
            switch state {
            case .Expanded:
                UIView.addKeyframe(withRelativeStartTime: 0.5, relativeDuration: 0.5) {
                    // Start with delay and finish with rest of animations
                    detailsButton.alpha = 1
                })
            case .Collapsed:
                UIView.addKeyframe(withRelativeStartTime: 0.0, relativeDuration: 0.5) {
                    // Start immediately and finish in half the time
                    detailsButton.alpha = 0
                })
            }
        }, completion: nil)
    }
}
```

```swift
func animateTransitionIfNeeded(forState state: State, duration: TimeInterval) {
    // ...
    let buttonAnimator = UIViewPropertyAnimator(duration: duration, curve: .linear) {
        UIView.animateKeyframes(withDuration: 0.0, delay: 0.0, options: [], animations: {
            switch state {
            case .Expanded:
                UIView.addKeyframe(withRelativeStartTime: 0.5, relativeDuration: 0.5) {
                    // Start with delay and finish with rest of animations
                    detailsButton.alpha = 1
                })
            case .Collapsed:
                UIView.addKeyframe(withRelativeStartTime: 0.0, relativeDuration: 0.5) {
                    // Start immediately and finish in half the time
                    detailsButton.alpha = 0
                })
            }
        }, completion: nil)
    }
}
```

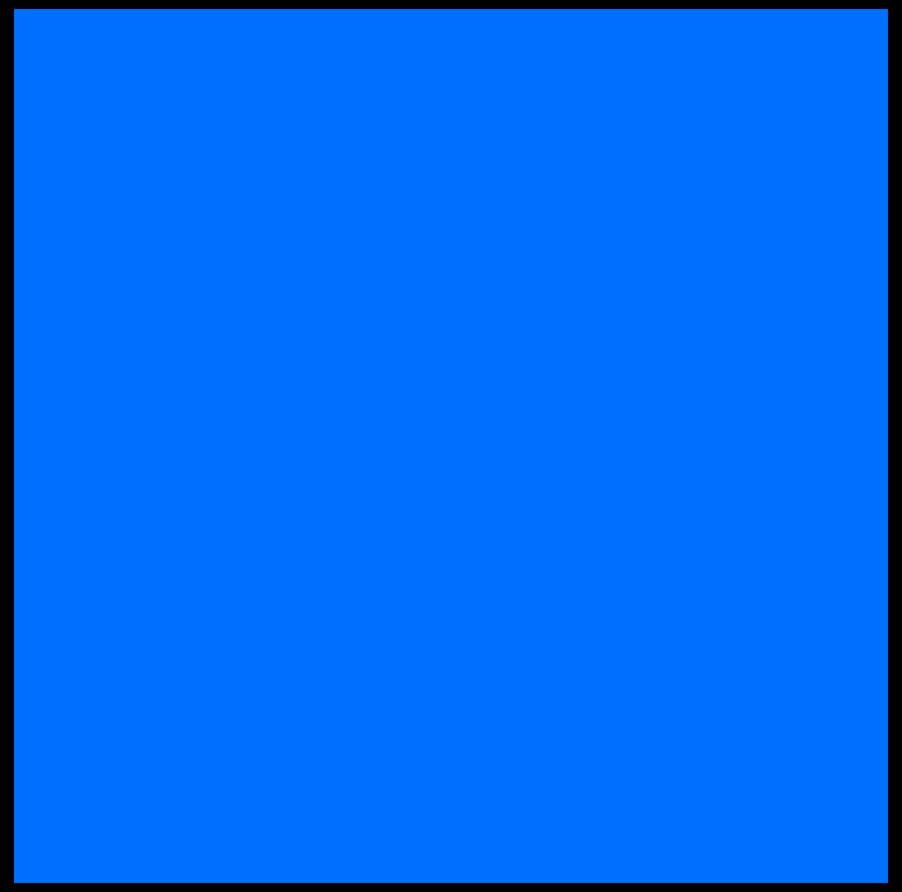
```swift
func animateTransitionIfNeeded(forState state: State, duration: TimeInterval) {
    // ...
    let buttonAnimator = UIViewPropertyAnimator(duration: duration, curve: .linear) {
        UIView.animateKeyframes(withDuration: 0.0, delay: 0.0, options: [], animations: {
            switch state {
            case .Expanded:
                UIView.addKeyframe(withRelativeStartTime: 0.5, relativeDuration: 0.5) {
                    // Start with delay and finish with rest of animations
                    detailsButton.alpha = 1
                })
            case .Collapsed:
                UIView.addKeyframe(withRelativeStartTime: 0.0, relativeDuration: 0.5) {
                    // Start immediately and finish in half the time
                    detailsButton.alpha = 0
                })
            }
        }, completion: nil)
    }
}
```

## Additive Animations

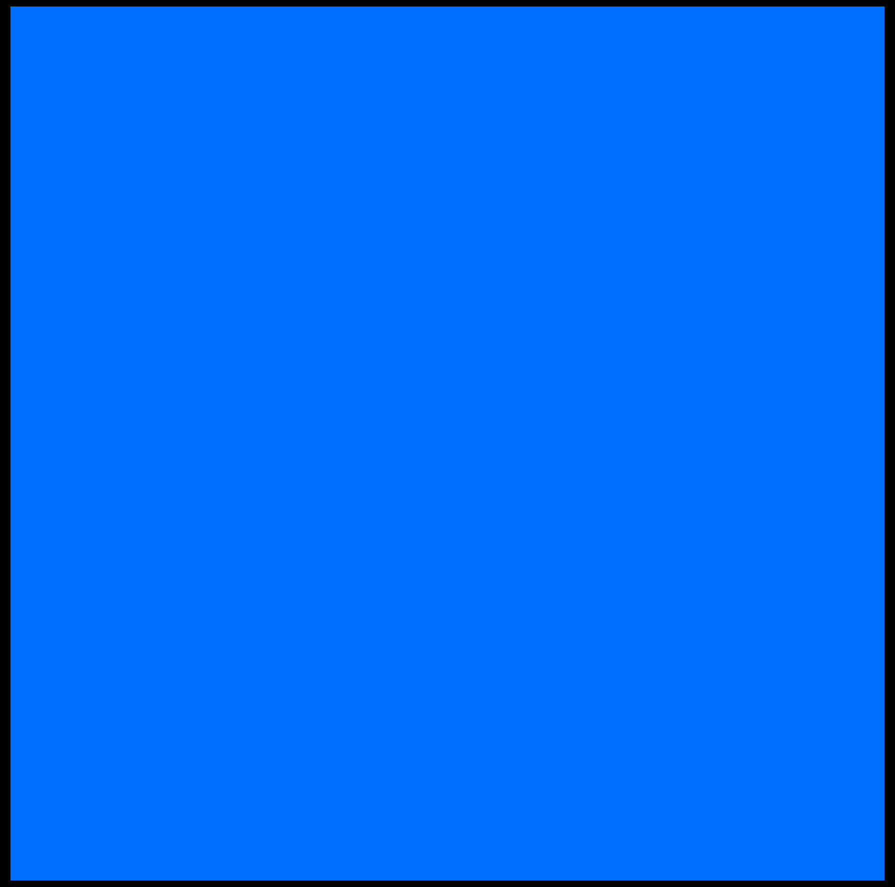
## Additive Animations



## Additive Animations



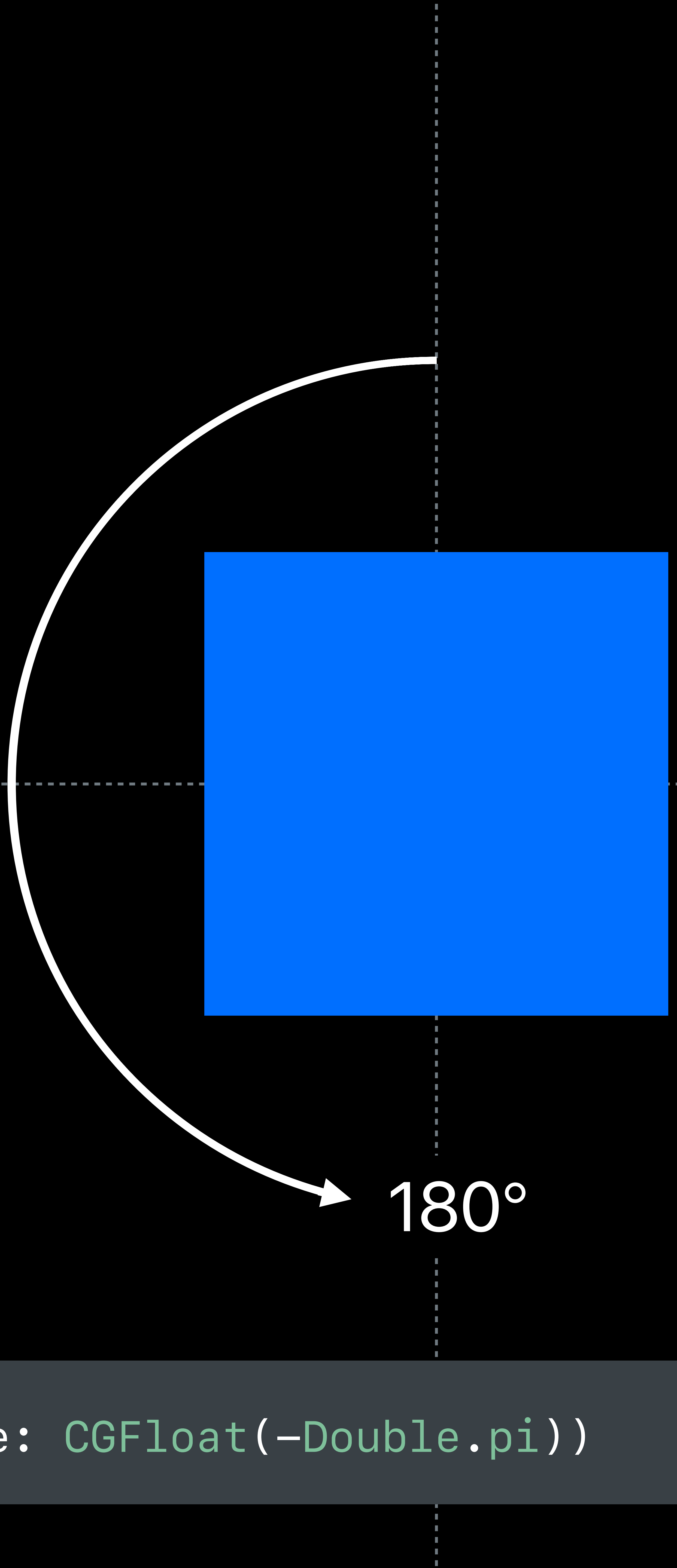
## Additive Animations



```
let animator = UIViewPropertyAnimator(duration: 5, curve: .easeInOut) {
    square.transform = CGAffineTransform(rotationAngle: CGFloat(Double.pi * 20))
}
animator.startAnimation()
```

```
CGAffineTransform(rotationAngle: CGFloat(Double.pi * 20))
```

```
CGAffineTransform(rotationAngle: CGFloat(-Double.pi))
```



```
CGAffineTransform(rotationAngle: CGFloat(-Double.pi))
```

```
CGAffineTransform(rotationAngle: CGFloat(-Double.pi))
```

```
CGAffineTransform(rotationAngle: CGFloat(-Double.pi))
```

## Options

Use Core Animation

- Low level
- No scrubbing

## Options

Use Core Animation

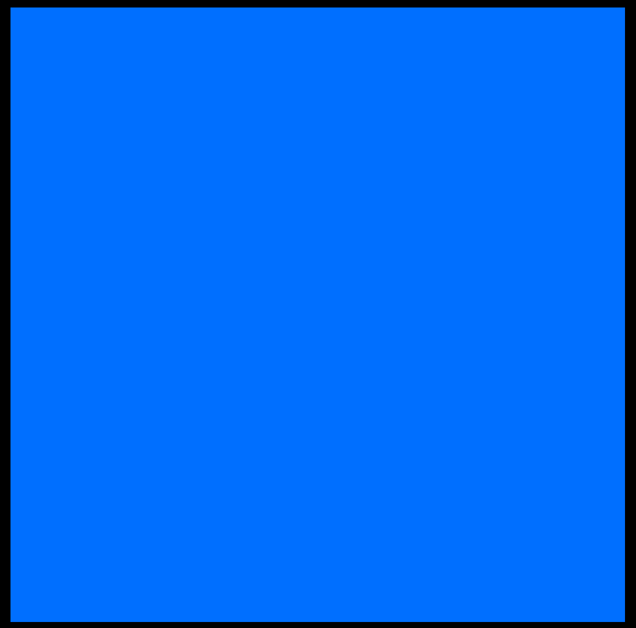
- Low level
- No scrubbing

Decompose into several smaller additive rotation animations

## Additively Animatable Properties

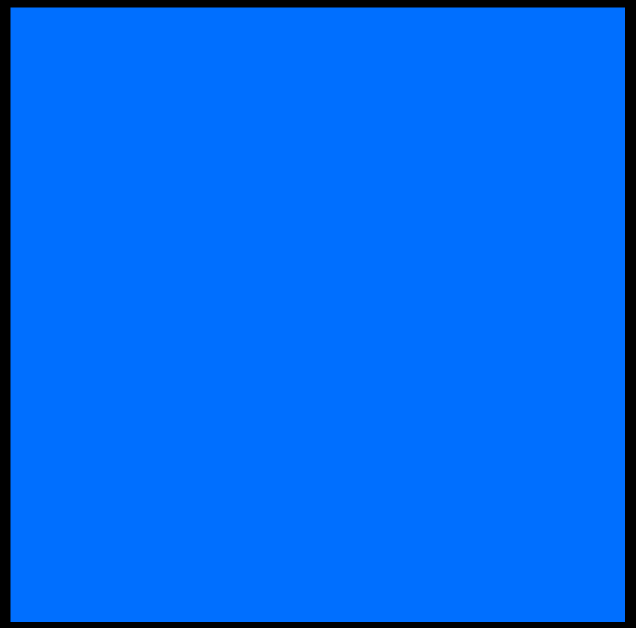
```
var transform: CGAffineTransform  // affine only

var frame: CGRect

var bounds: CGRect

var center: CGPoint

var position: CGPoint
```

## Decomposed Additive Animations



```
let animator = UIViewPropertyAnimator(duration: 5, curve: .easeInOut, animations: {
    for _ in 0..<20 {
        let rotation = CGAffineTransform(rotationAngle: CGFloat(Double.pi))
        square.transform = square.transform.concatenating(rotation)
    }
})
animator.startAnimation()
```

# Decomposed Additive Animations



```
let animator = UIViewPropertyAnimator(duration: 5, curve: .easeInOut, animations: {
    for _ in 0..<20 {
        let rotation = CGAffineTransform(rotationAngle: CGFloat(Double.pi))
        square.transform = square.transform.concatenating(rotation)
    }
})
animator.startAnimation()
```

## Summary

Modern methods for making animations interactive and interruptible

Coordinating several animations during interactive transition

## Related Sessions

| | | |
|---|---|---|
| What's New in Cocoa Touch | Hall 3 | Tuesday 10:20AM |
| Introducing Drag and Drop | Hall 3 | Tuesday 11:20AM |
| Mastering Drag and Drop | Executive Ballroom | Wednesday 11:00AM |
| Modern User Interaction on iOS | Grand Ballroom B | Wednesday 4:10PM |
| Drag and Drop with Collection and Table View | Hall 2 | Thursday 9:00AM |
| Data Delivery with Drag and Drop | Hall 2 | Thursday 10:00AM |

## Previous Sessions

| | |
|---|---|
| Core Animation Essentials | WWDC 2011 |
| Custom Transitions Using View Controllers | WWDC 2013 |
| Building Interruptible and Responsive Interactions | WWDC 2014 |
| Advanced Graphics and Animations for iOS Apps | WWDC 2014 |
| View Controller Advancements in iOS 8 | WWDC 2014 |
| Advances in UIKit Animations and Transitions | WWDC 2016 |

## Labs

| | | |
|---|---|---|
| Cocoa Touch and Haptics Lab | Technology Lab C | Friday 12:00PM |

## More Information

https://developer.apple.com/wwdc17/230

WWDC17