

# Advanced Techniques with UIKit Dynamics

Session 221

Olivier Gutknecht

Bruce D. Nilo

These are confidential sessions—please refrain from streaming, blogging, or taking pictures

# Agenda

What we will cover

# Agenda

## What we will cover

- Core concepts

# Agenda

## What we will cover

- Core concepts
- Combining behaviors

# Agenda

## What we will cover

- Core concepts
- Combining behaviors
- Using custom dynamic items

# Agenda

## What we will cover

- Core concepts
- Combining behaviors
- Using custom dynamic items
- Collection view and dynamics

# Agenda

## What we will cover

- Core concepts
- Combining behaviors
- Using custom dynamic items
- Collection view and dynamics
- Using dynamics for view controller transitions

# UIKit Dynamics



# UIKit Dynamics

- Real-world inspired animation and interaction system

# UIKit Dynamics

- Real-world inspired animation and interaction system
- Combinable, reusable, declarative

# UIKit Dynamics

- Real-world inspired animation and interaction system
- Combinable, reusable, declarative
- Does not replace for Core Animation, UIView animations or motion effects

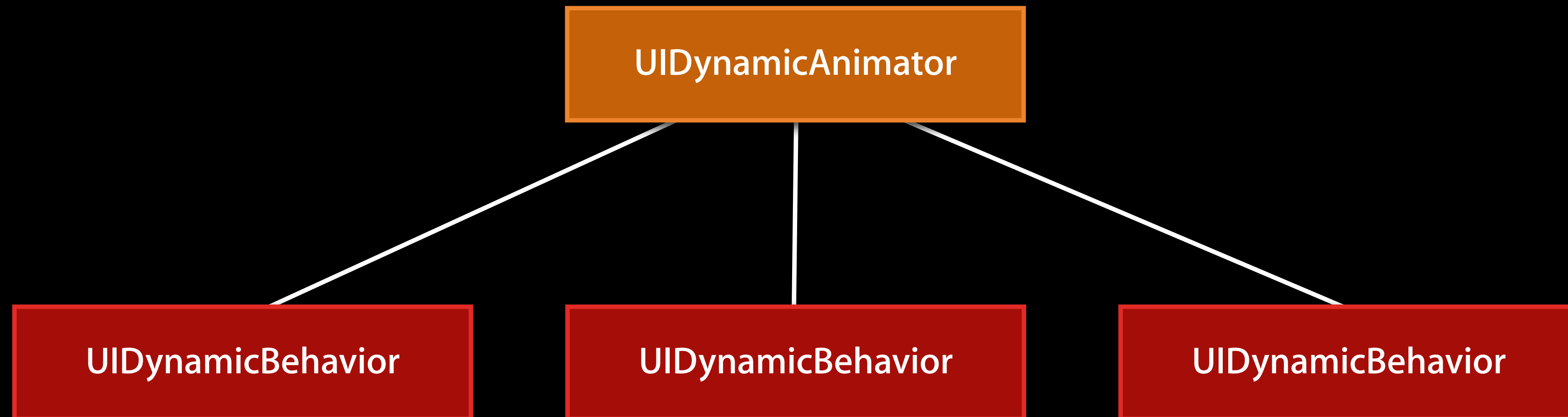
# UIKit Dynamics

- Real-world inspired animation and interaction system
- Combinable, reusable, declarative
- Does not replace for Core Animation, UIView animations or motion effects
- Think interactions

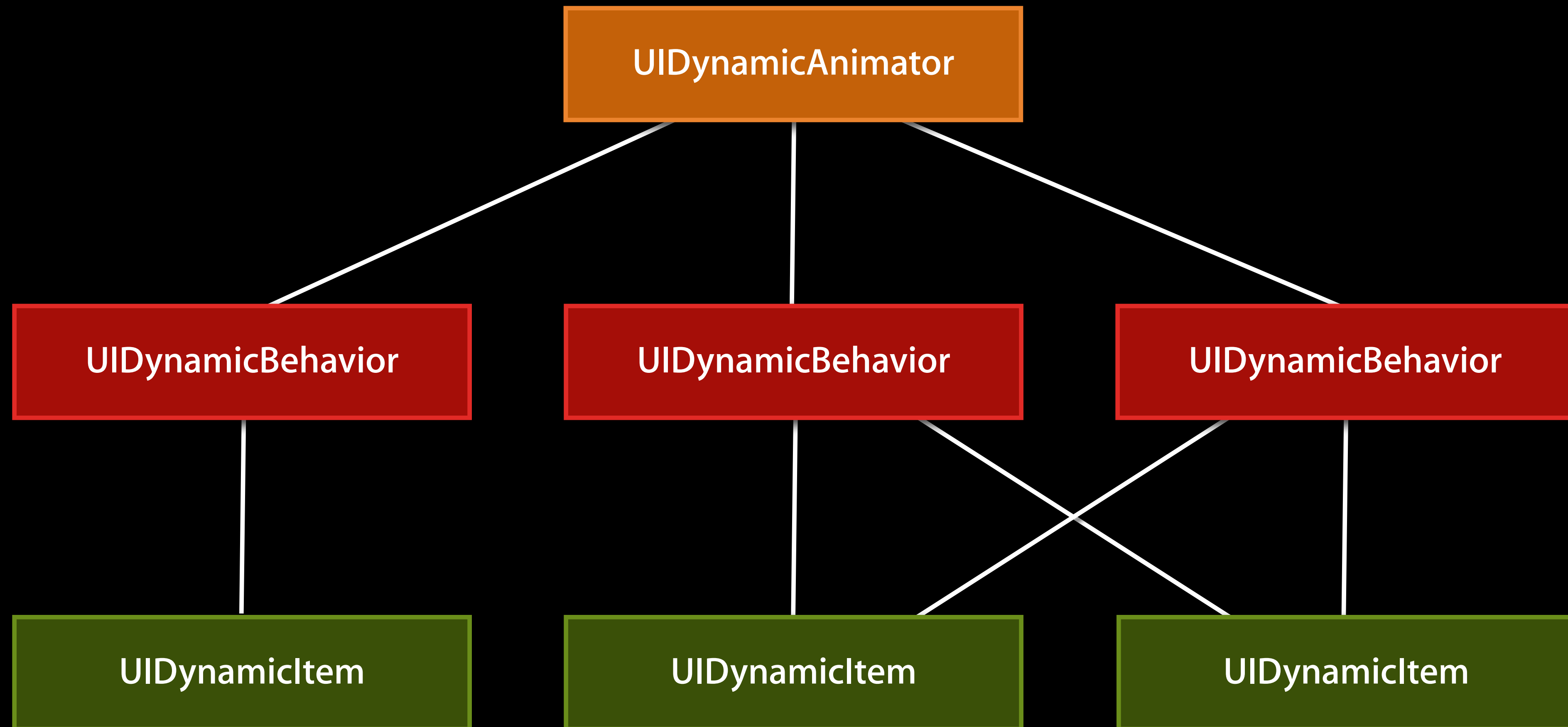
# Architecture

UIDynamicAnimator

# Architecture



# Architecture



# UIDynamicAnimator



# UIDynamicAnimator

- Track behaviors and animated items

# UIDynamicAnimator

- Track behaviors and animated items
- Wrap the underlying physics engine

# UIDynamicAnimator

- Track behaviors and animated items
- Wrap the underlying physics engine
- Run and optimize the animation

# UIDynamicAnimator

- Track behaviors and animated items
- Wrap the underlying physics engine
- Run and optimize the animation
  - To respond to pausing and resuming, use `UIDynamicAnimatorDelegate`

# UIDynamicAnimator

- Track behaviors and animated items
- Wrap the underlying physics engine
- Run and optimize the animation
  - To respond to pausing and resuming, use `UIDynamicAnimatorDelegate`
- Three different modes: Views, collection view layouts, and raw items

# Combining Behaviors

# Combining Behaviors

# Combining Behaviors

- Physics lets you combine elements



# Combining Behaviors

- Physics lets you combine elements
- Base behavior class supports grouping and subclassing

# Combining Behaviors

- Physics lets you combine elements
- Base behavior class supports grouping and subclassing
- Sub-behaviors and top-level behaviors are similar for the animator

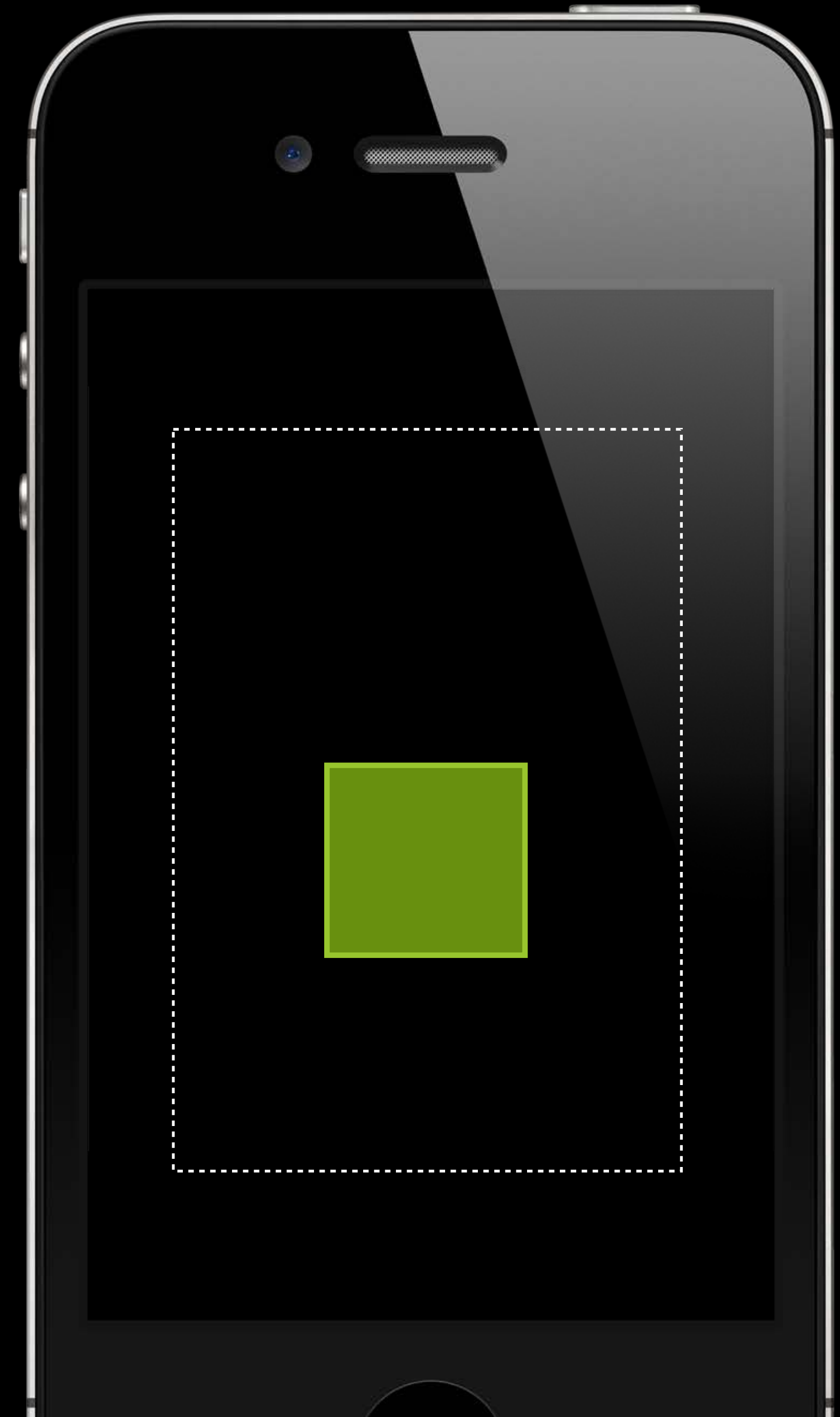
# Combining Behaviors

- Physics lets you combine elements
- Base behavior class supports grouping and subclassing
- Sub-behaviors and top-level behaviors are similar for the animator
- Compose your behaviors statically or dynamically

# Updating Active Behaviors

## Example: Drag and fall

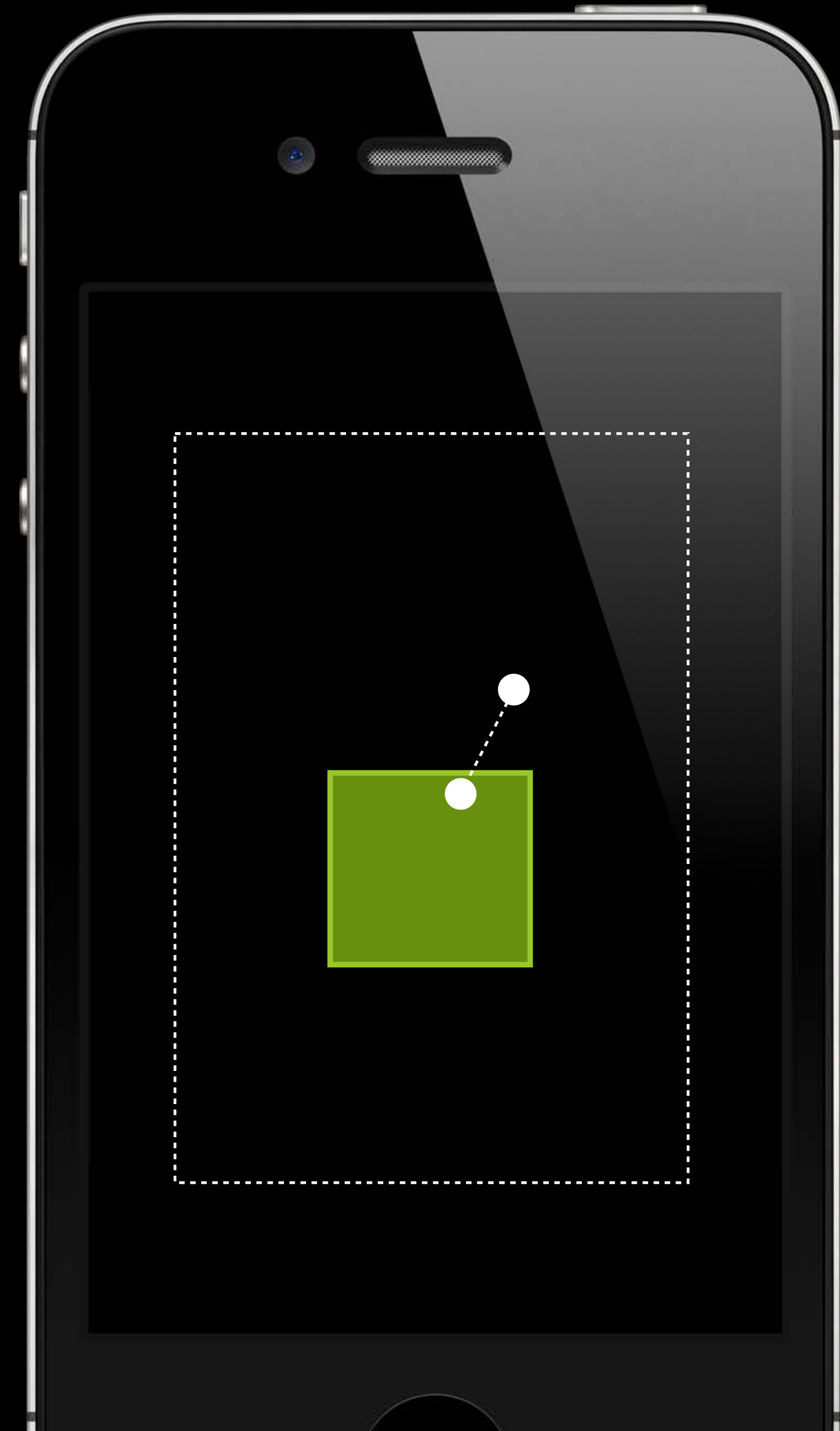
- Initial setup
  - Set up a collision behavior and add the view



# Updating Active Behaviors

## Example: Drag and fall

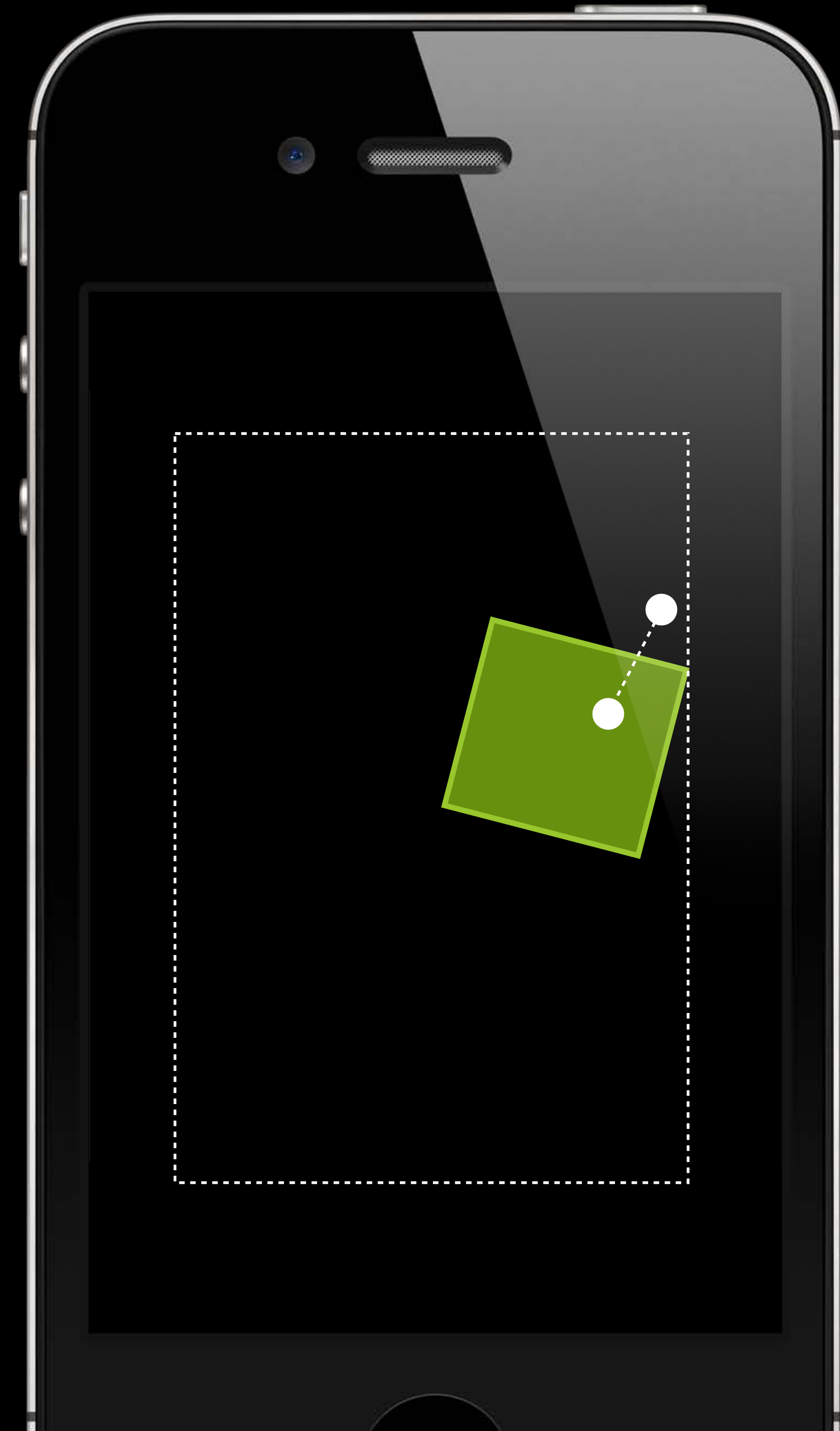
- Initial setup
  - Set up a collision behavior and add the view
- Gesture began
  - Add the view to an attachment behavior



# Updating Active Behaviors

## Example: Drag and fall

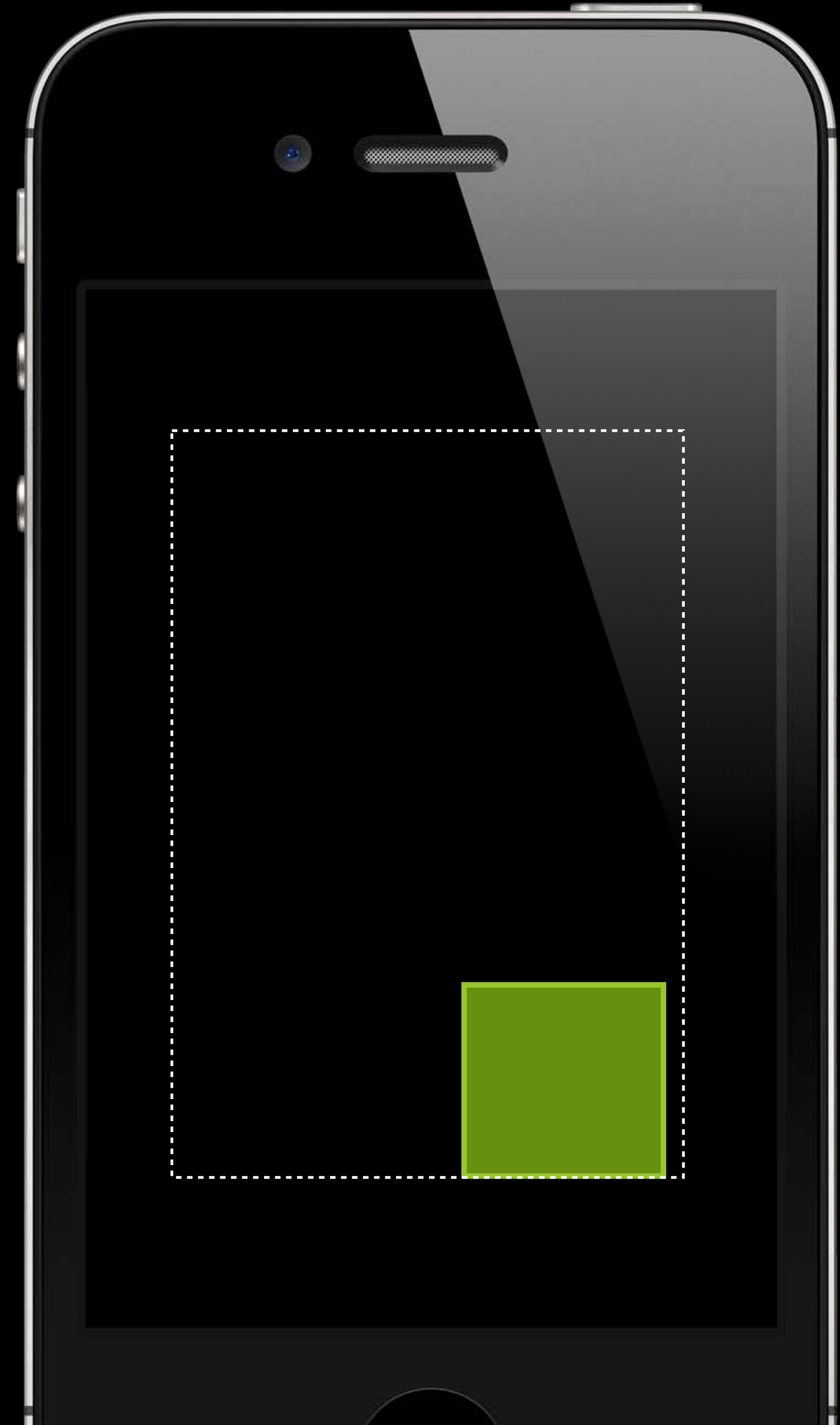
- Initial setup
  - Set up a collision behavior and add the view
- Gesture began
  - Add the view to an attachment behavior
- Gesture changed
  - Update its anchor point to the gesture position



# Updating Active Behaviors

## Example: Drag and fall

- Initial setup
  - Set up a collision behavior and add the view
- Gesture began
  - Add the view to an attachment behavior
- Gesture changed
  - Update its anchor point to the gesture position
- Gesture ended
  - Remove the attachment behavior
  - Add gravity



# Grouping Behaviors



# Grouping Behaviors

Effects

Combination

---

# Grouping Behaviors

Effects

Combination

Bounce

Gravity + Collision

# Grouping Behaviors

## Effects

Bounce

Drag and Snap in place

## Combination

Gravity + Collision

Attachment then Snap

# Grouping Behaviors

Effects

Combination

Bounce

Gravity + Collision

Drag and Snap in place

Attachment then Snap

Lock Screen

Gravity + Collision + Attachment + Push

# Grouping Behaviors

Effects

Combination

Bounce

Gravity + Collision

Drag and Snap in place

Attachment then Snap

Lock Screen

Gravity + Collision + Attachment + Push

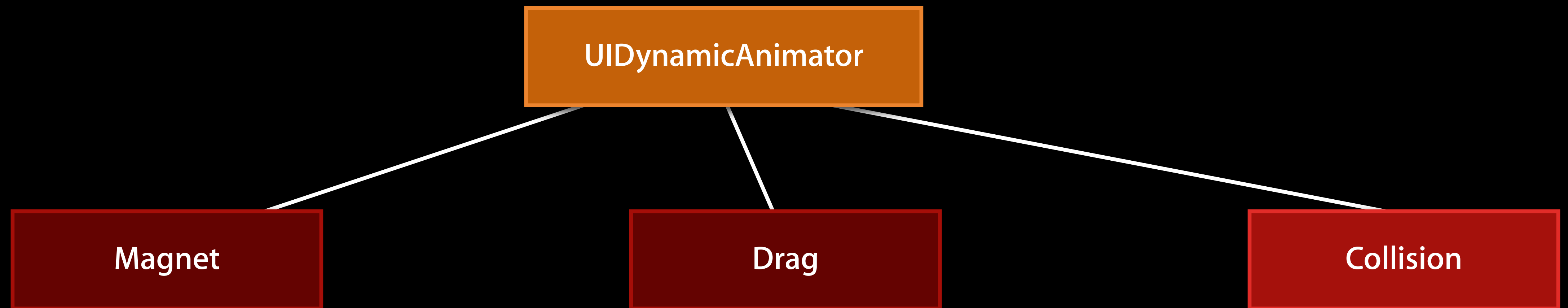
Attraction Field

Multiple Push

*Demo*

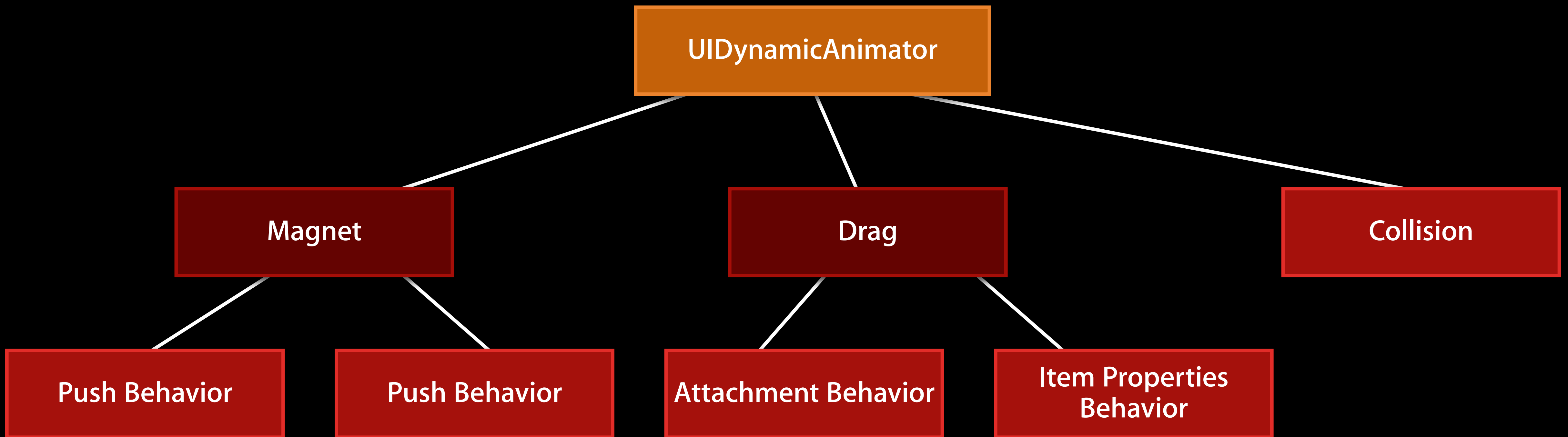
# Composing Behaviors

## The behavior tree



# Composing Behaviors

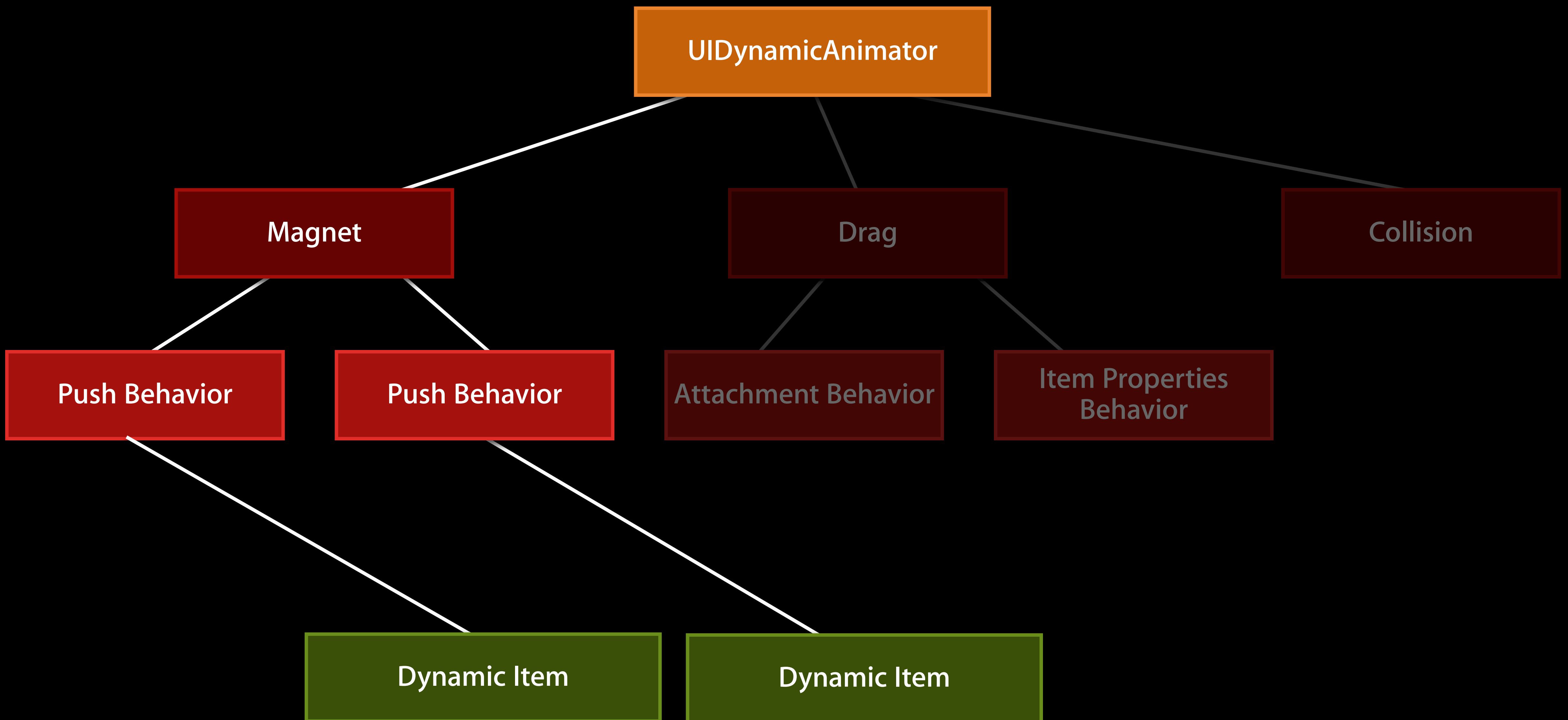
## The behavior tree





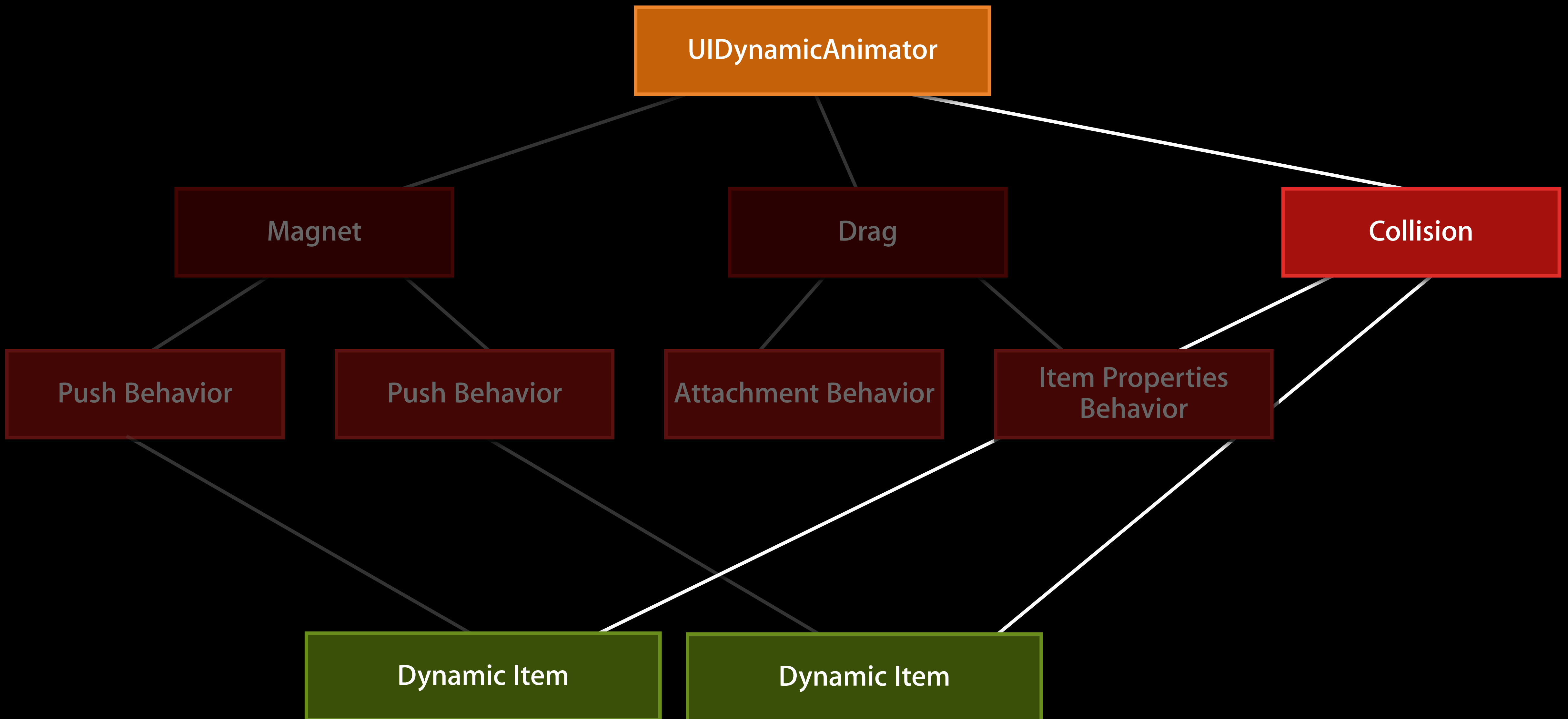
# Composing Behaviors

## The behavior tree



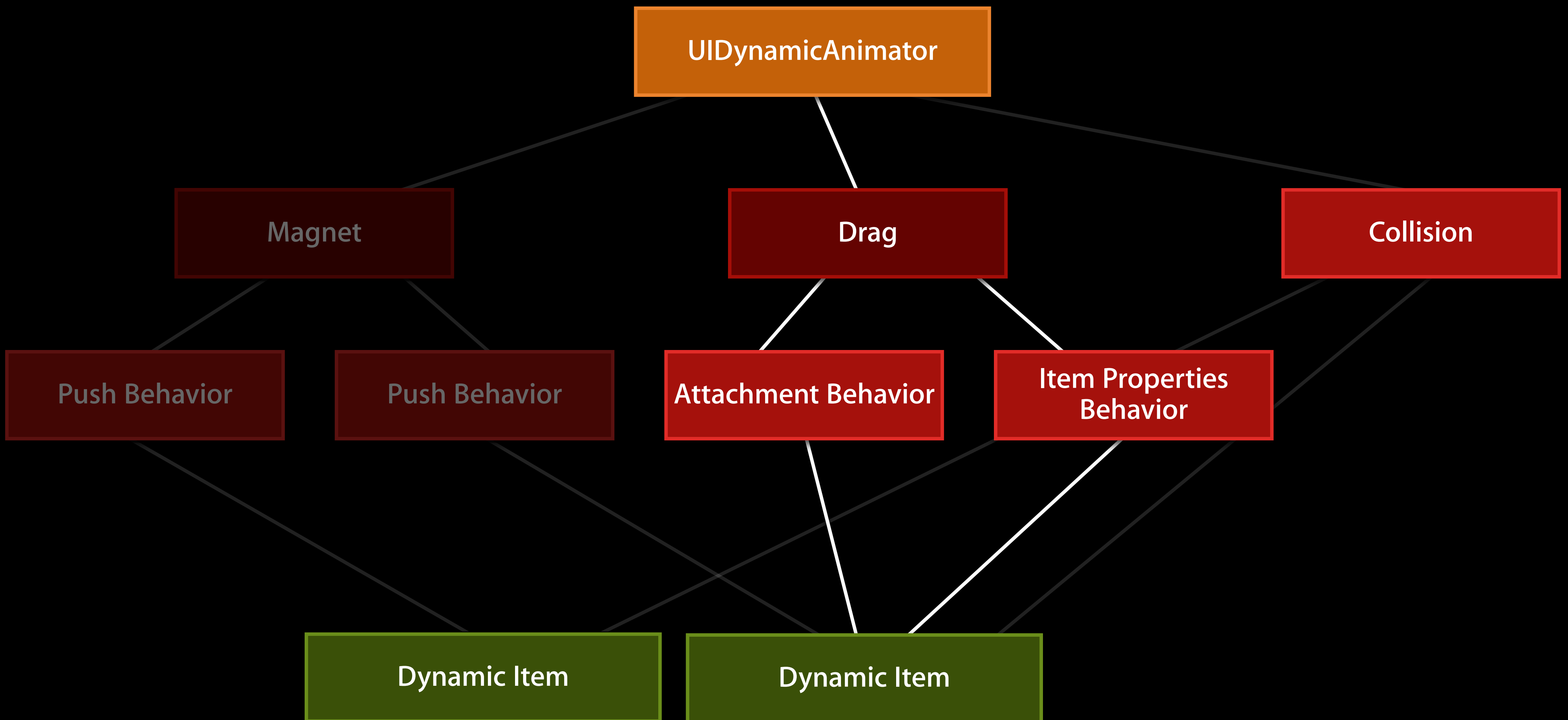
# Composing Behaviors

## The behavior tree



# Composing Behaviors

## The behavior tree



# How to Define Your High-Level Behavior

## Subclass UIDynamicBehavior

```
@interface BouncyFallBehavior : UIDynamicBehavior

-(instancetype)initWithItems:(NSArray*)items;

@end
```

# How to Define Your High-Level Behavior

## Define sub-behaviors

```
-(instancetype)initWithItems:(NSArray*)items {  
    if (self=[super init]) {  
        UIGravityBehavior* g = [UIGravityBehavior alloc] initWithItems:items;  
        UICollisionBehavior* c = [UICollisionBehavior alloc] initWithItems:items;  
        c.translatesReferenceBoundsIntoBoundary = TRUE;  
        [self addChildBehavior:g];  
        [self addChildBehavior:c];  
    }  
}
```

# How to Define Your High-Level Behavior

## Configure sub-behaviors

```
-(instancetype)initWithItems:(NSArray*)items {
    if (self=[super init]) {
        UIGravityBehavior* g = [UIGravityBehavior alloc] initWithItems:items;
        UICollisionBehavior* c = [UICollisionBehavior alloc] initWithItems:items;
        c.translatesReferenceBoundsIntoBoundary = TRUE;
        [self addChildBehavior:g];
        [self addChildBehavior:c];
    }
}
```

# How to Define Your High-Level Behavior

## Add child behaviors

```
-(instancetype)initWithItems:(NSArray*)items {  
    if (self=[super init]) {  
        UIGravityBehavior* g = [UIGravityBehavior alloc] initWithItems:items;  
        UICollisionBehavior* c = [UICollisionBehavior alloc] initWithItems:items;  
        c.translatesReferenceBoundsIntoBoundary = TRUE;  
        [self addChildBehavior:g];  
        [self addChildBehavior:c];  
    }  
}
```

# How to Define Your High-Level Behavior

Create an animator and add your own behavior

```
UIDynamicAnimator* animator;  
BouncyFallBehavior* behavior;
```

```
animator = [[UIDynamicAnimator alloc] initWithReferenceView:referenceView];  
behavior = [BouncyFallBehavior alloc] initWithItems:@[myView]];
```

```
[animator addBehavior:b];
```



# Best Practices

# Best Practices

- Encapsulate and define your own API

# Best Practices

- Encapsulate and define your own API
- Integrate with your existing interactions

# Best Practices

- Encapsulate and define your own API
- Integrate with your existing interactions
  - i.e. use the ending gesture velocity as an initial velocity

# Best Practices

- Encapsulate and define your own API
- Integrate with your existing interactions
  - i.e. use the ending gesture velocity as an initial velocity
- **If needed** you can define per-step actions

# Best Practices

- Encapsulate and define your own API
  - Integrate with your existing interactions
    - i.e. use the ending gesture velocity as an initial velocity
  - **If needed** you can define per-step actions
- ```
@property (nonatomic, copy) void (^action)(void);
```

# Best Practices

- Encapsulate and define your own API
- Integrate with your existing interactions
  - i.e. use the ending gesture velocity as an initial velocity
- **If needed** you can define per-step actions

```
@property (nonatomic, copy) void (^action)(void);
```

- i.e. adjust a force based on an item position

# Best Practices

- Encapsulate and define your own API
- Integrate with your existing interactions
  - i.e. use the ending gesture velocity as an initial velocity
- **If needed** you can define per-step actions

```
@property (nonatomic, copy) void (^action)(void);
```

- i.e. adjust a force based on an item position
- Performance is crucial



# Using Multiple UIDynamicItemBehaviors

- Multiple UIDynamicItemBehavior changing **distinct** properties is fine
- Multiple UIDynamicItemBehavior changing the **same** property?
  - Last one wins
  - Last one: Pre-order depth first walk of the behavior tree

# Using Multiple UIDynamicItemBehaviors

- Use UIDynamicItemBehavior to change base item properties
  - damping
  - friction
  - elasticity
  - rotation blocking
  - ...
- Multiple UIDynamicItemBehavior changing **distinct** properties is fine
- Multiple UIDynamicItemBehavior changing the **same** property?
  - Last one wins
  - Last one: Pre-order depth first walk of the behavior tree

# Using Multiple UIDynamicItemBehaviors

- Use UIDynamicItemBehavior to change base item properties
  - damping
  - friction
  - elasticity
  - rotation blocking
  - ...
- Multiple UIDynamicItemBehavior changing **distinct** properties is fine

# Using Multiple UIDynamicItemBehaviors

- Use UIDynamicItemBehavior to change base item properties
  - damping
  - friction
  - elasticity
  - rotation blocking
  - ...
- Multiple UIDynamicItemBehavior changing **distinct** properties is fine
- Multiple UIDynamicItemBehavior changing the **same** property?

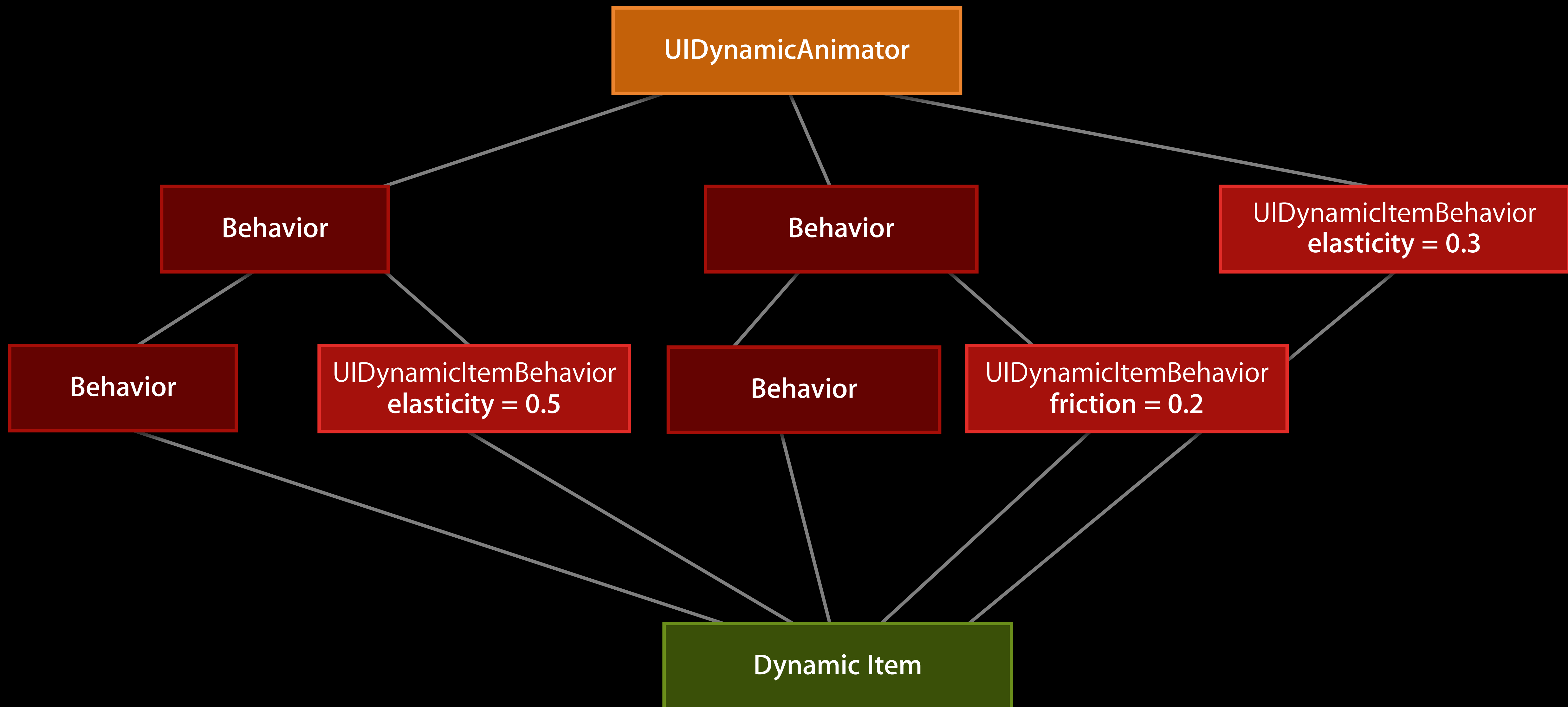
# Using Multiple UIDynamicItemBehaviors

- Use UIDynamicItemBehavior to change base item properties
  - damping
  - friction
  - elasticity
  - rotation blocking
  - ...
- Multiple UIDynamicItemBehavior changing **distinct** properties is fine
- Multiple UIDynamicItemBehavior changing the **same** property?
  - Last one wins

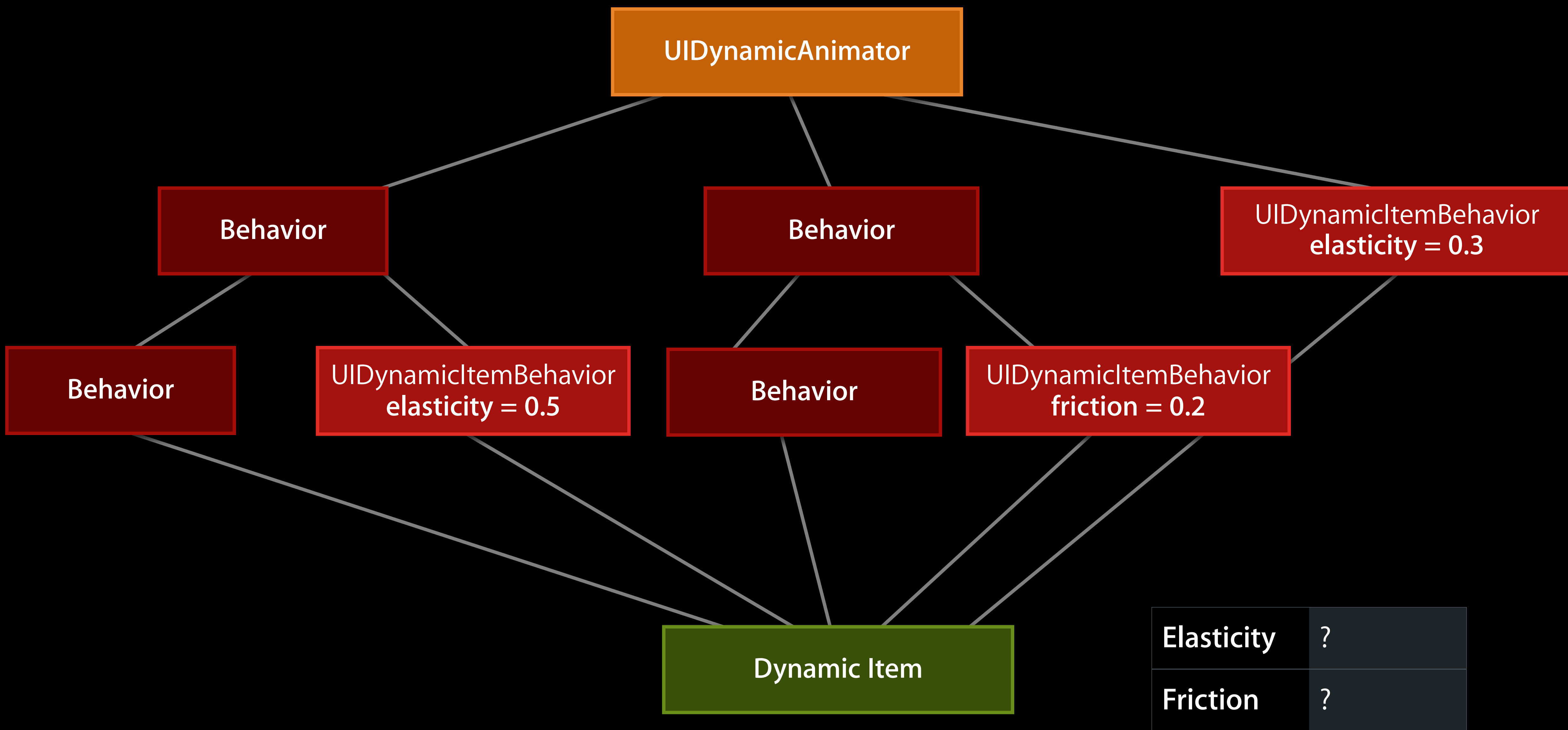
# Using Multiple UIDynamicItemBehaviors

- Use UIDynamicItemBehavior to change base item properties
  - damping
  - friction
  - elasticity
  - rotation blocking
  - ...
- Multiple UIDynamicItemBehavior changing **distinct** properties is fine
- Multiple UIDynamicItemBehavior changing the **same** property?
  - Last one wins
  - Last one: Pre-order depth first walk of the behavior tree

# Using Multiple UIDynamicItemBehaviors

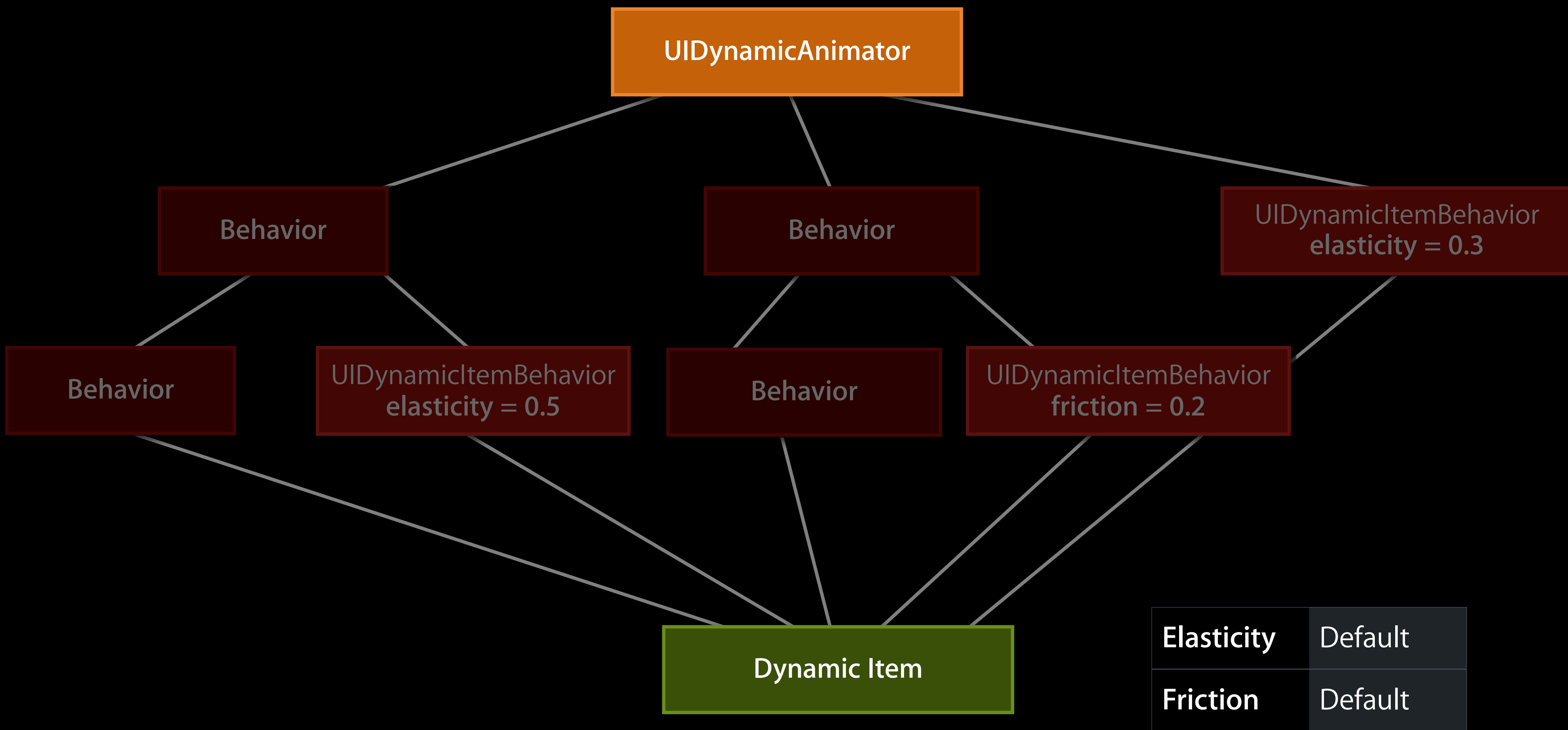


# Using Multiple UIDynamicItemBehaviors

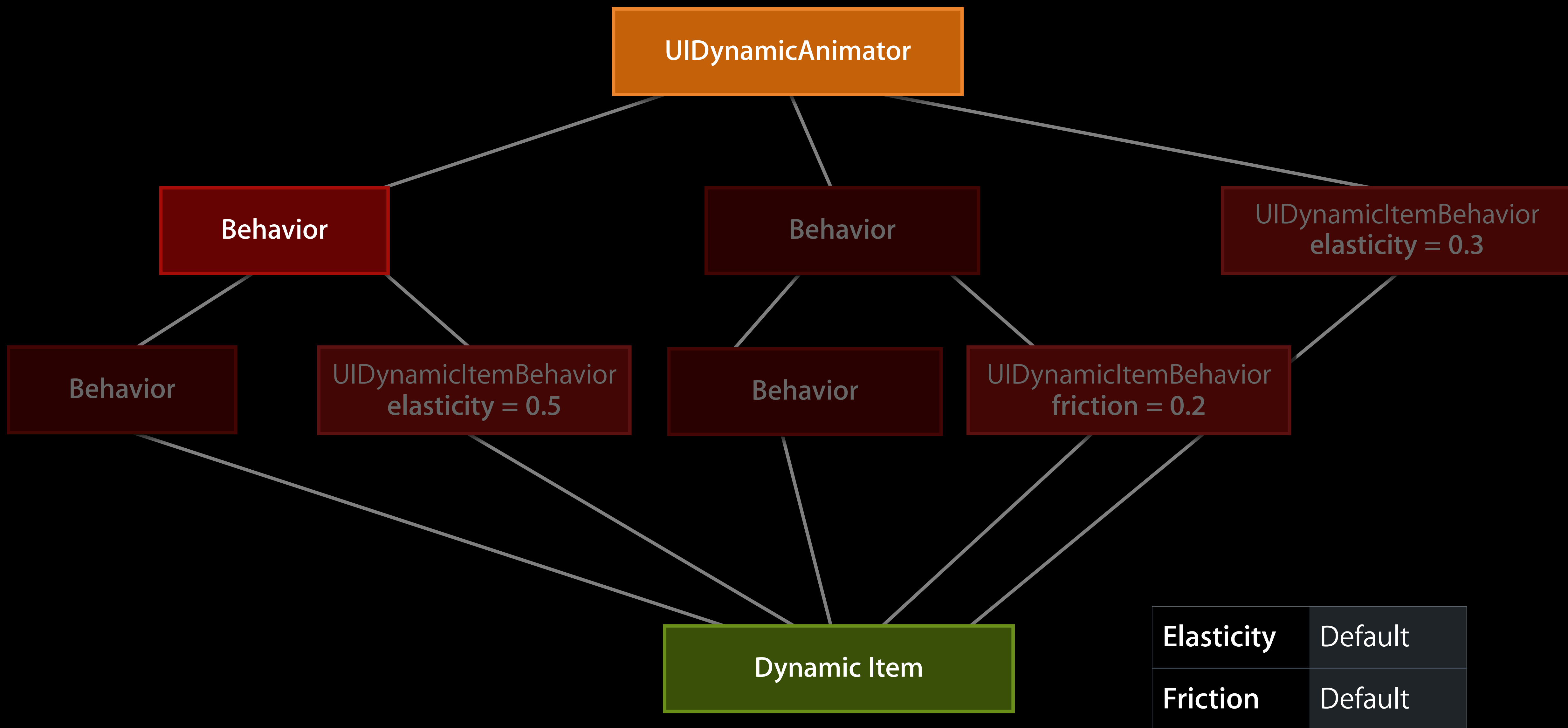




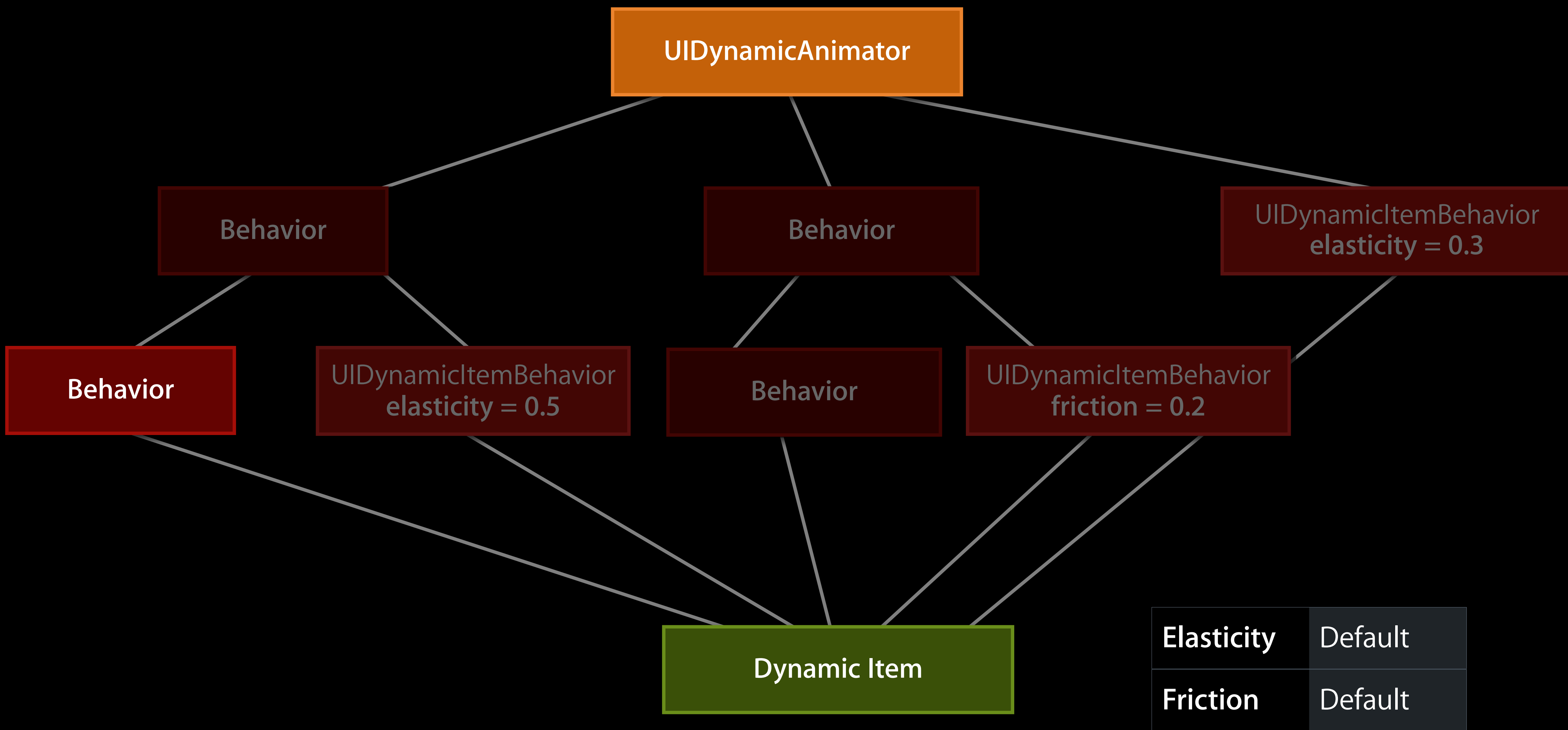
# Using Multiple UIDynamicItemBehaviors



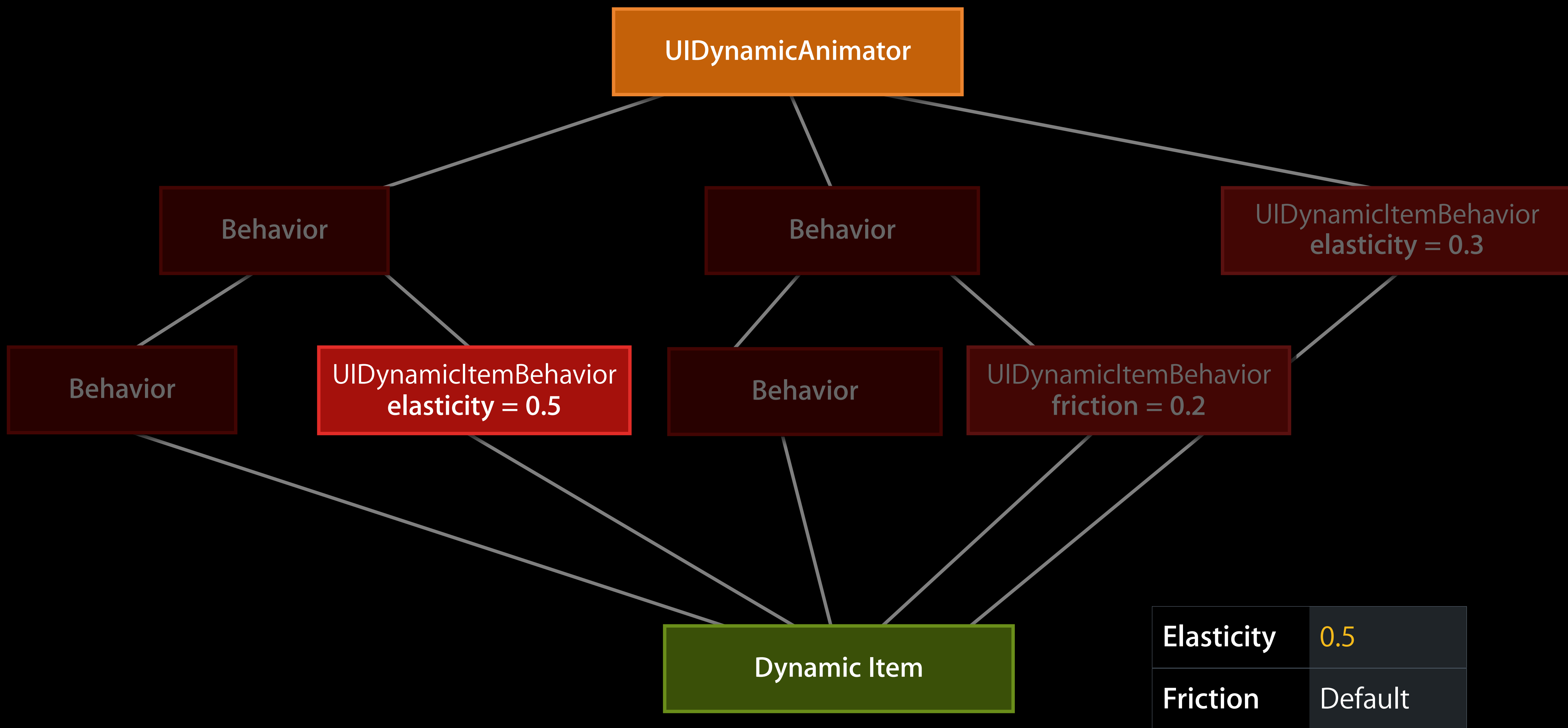
# Using Multiple UIDynamicItemBehaviors



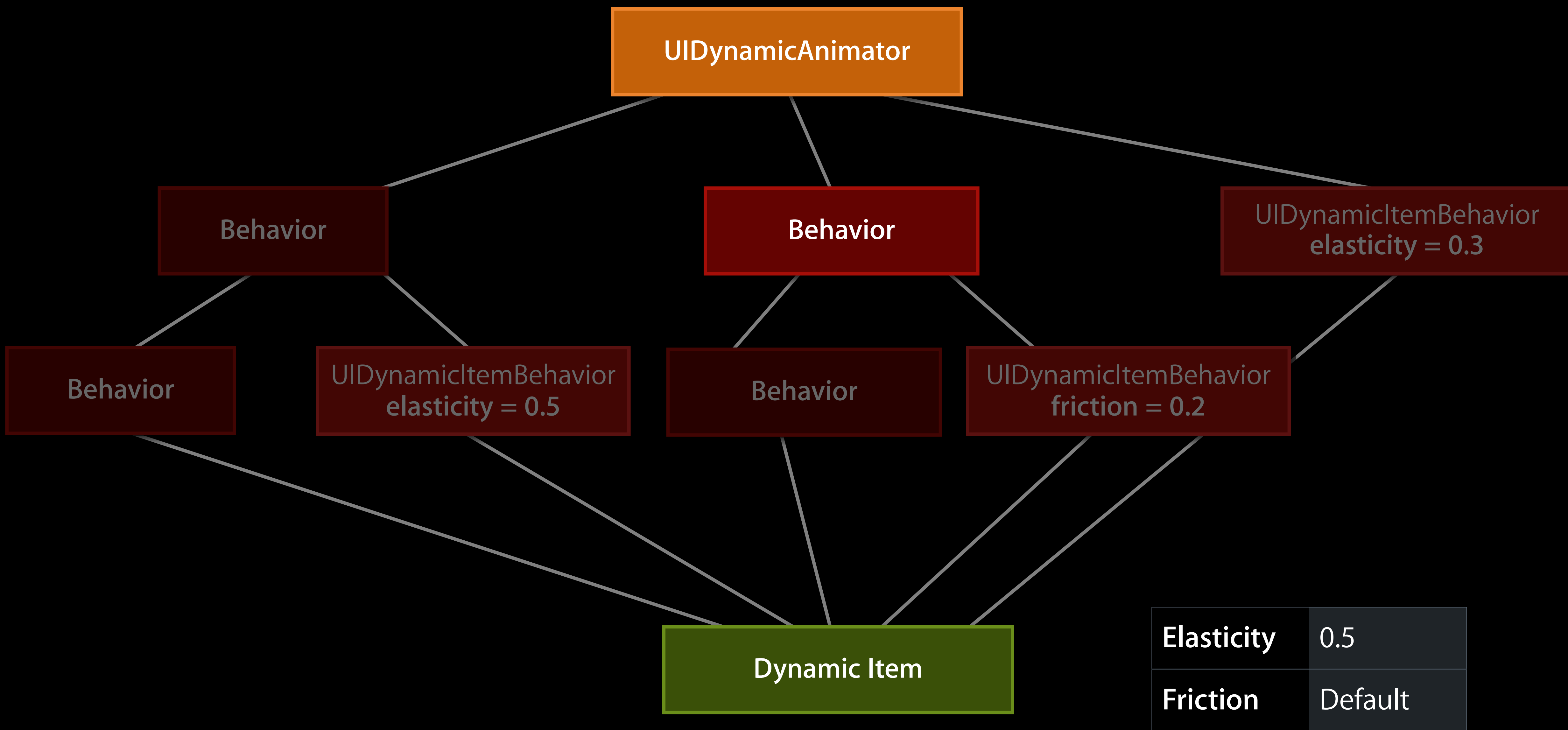
# Using Multiple UIDynamicItemBehaviors



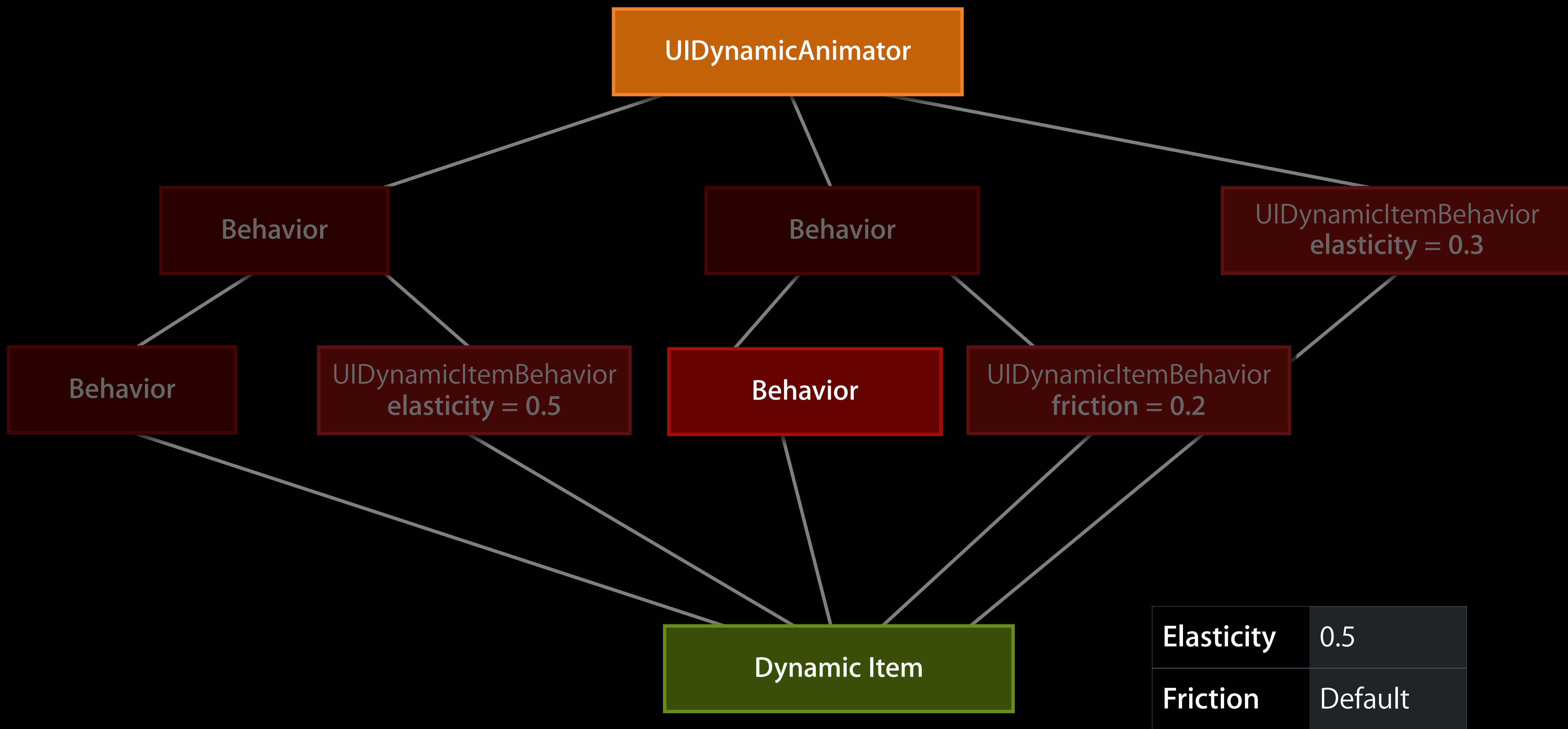
# Using Multiple UIDynamicItemBehaviors



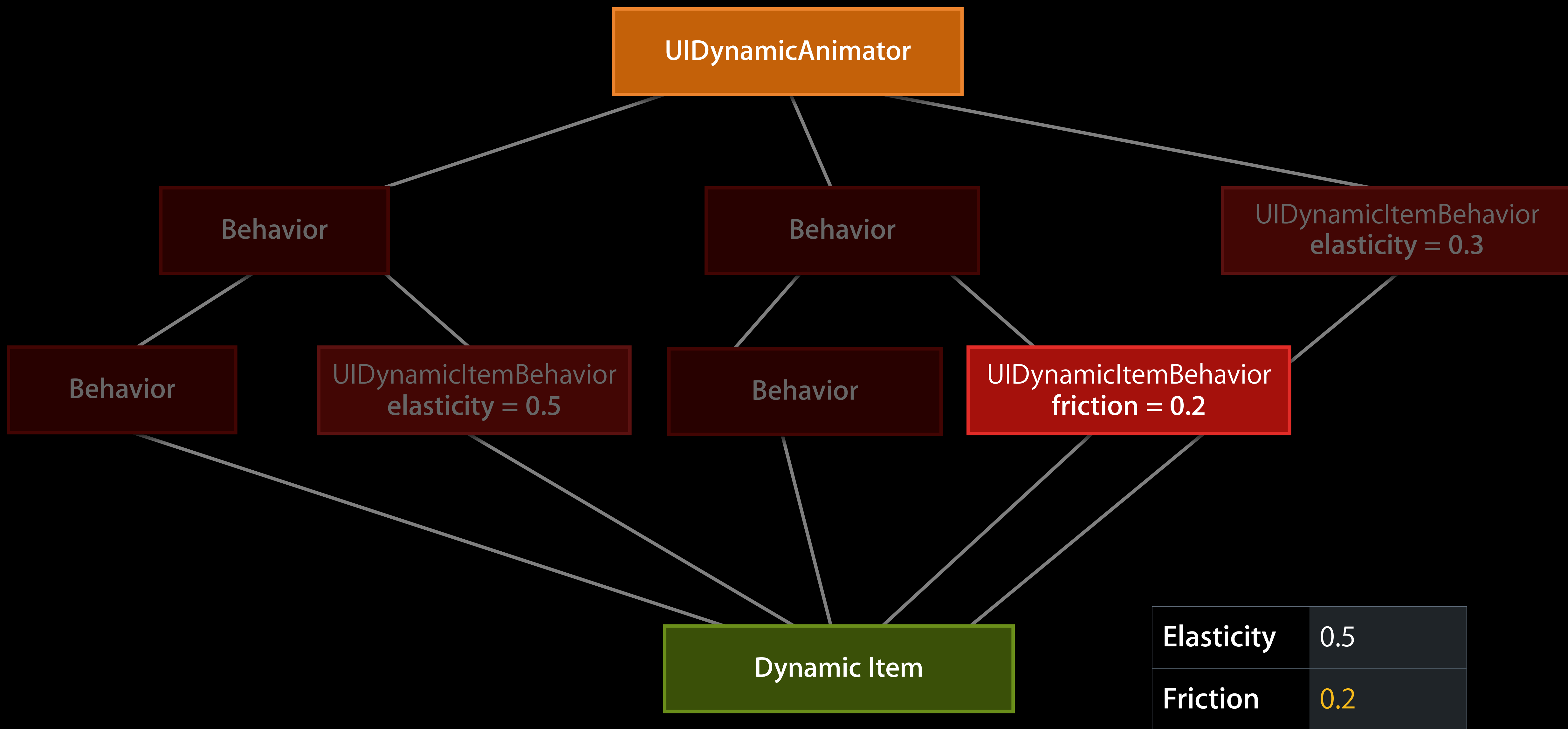
# Using Multiple UIDynamicItemBehaviors



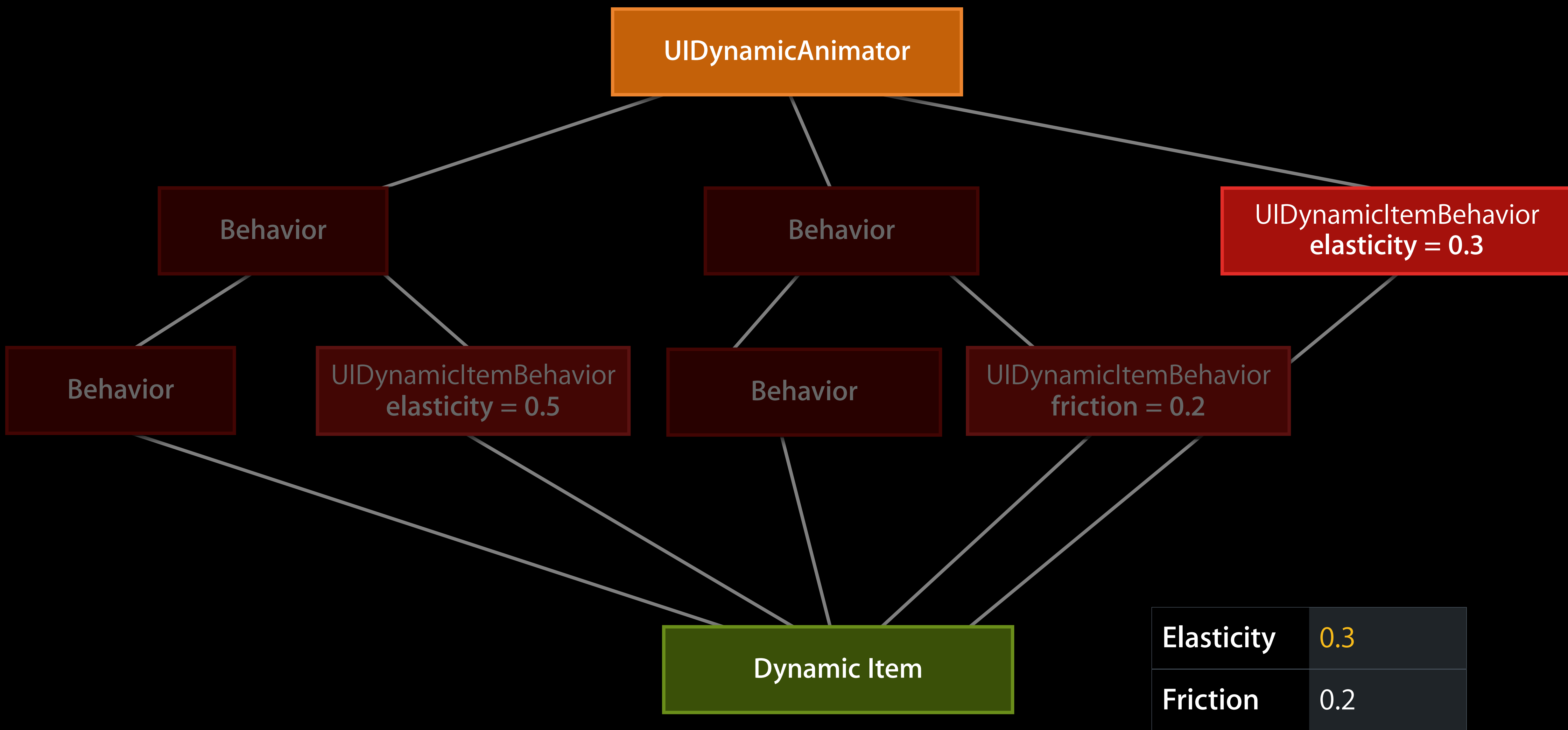
# Using Multiple UIDynamicItemBehaviors



# Using Multiple UIDynamicItemBehaviors

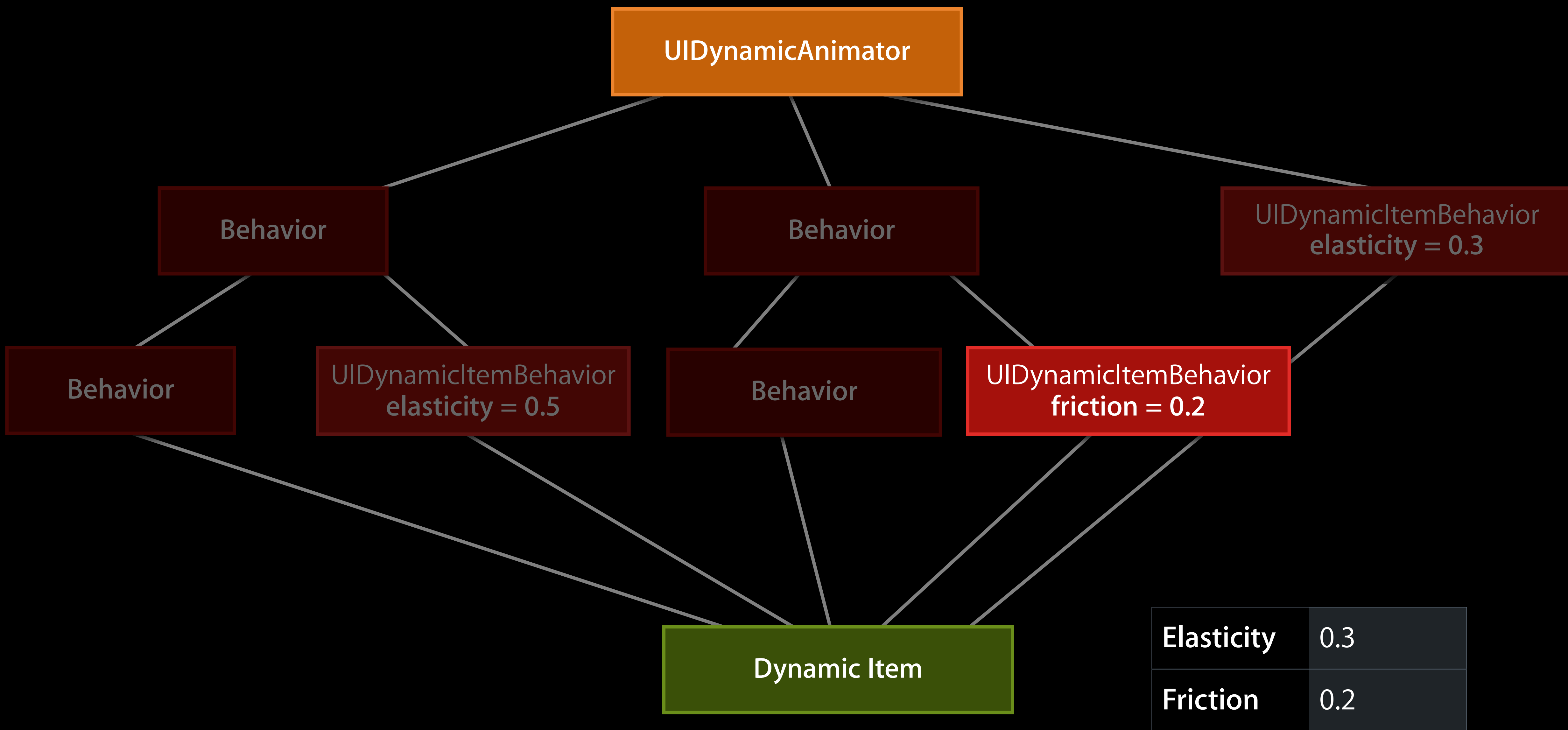


# Using Multiple UIDynamicItemBehaviors

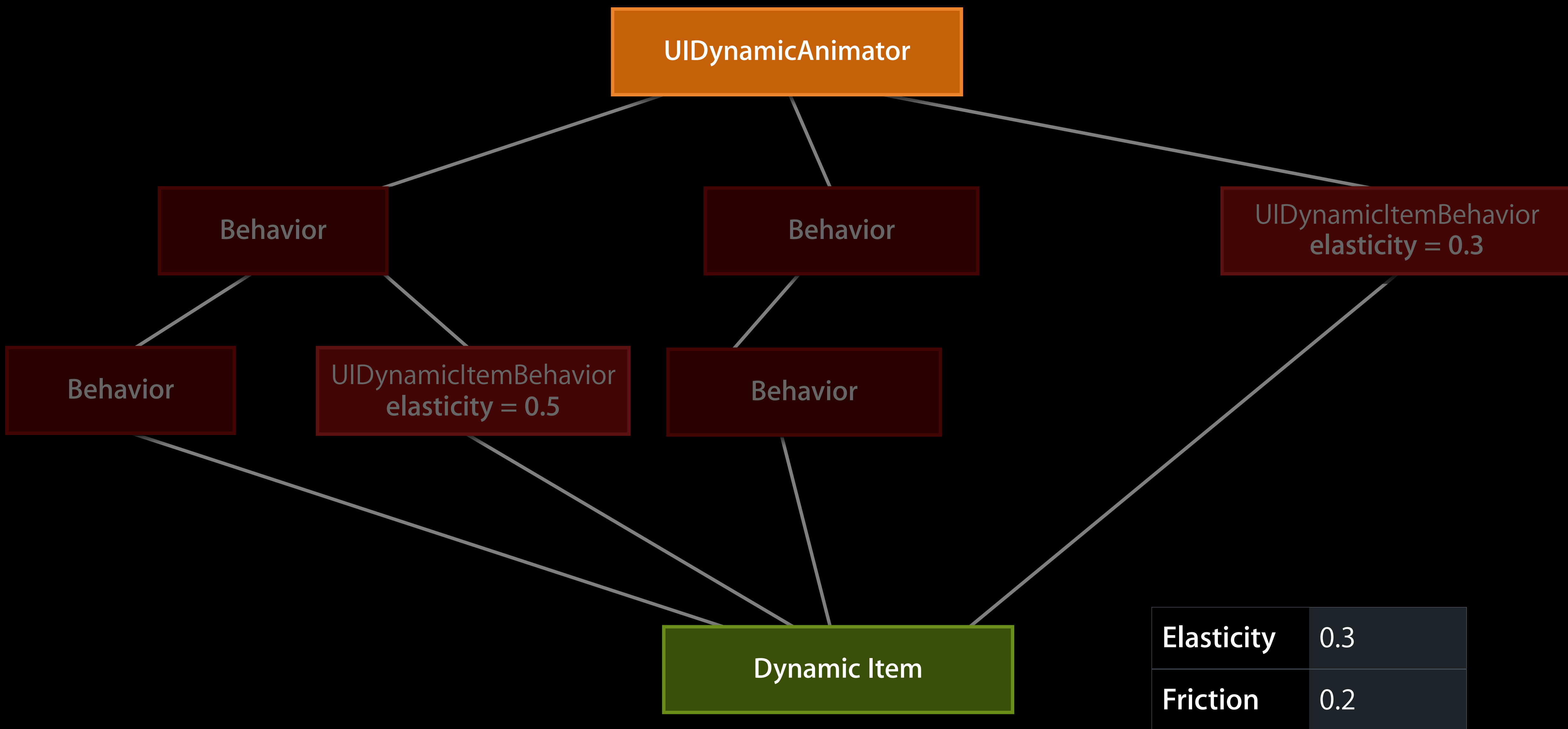




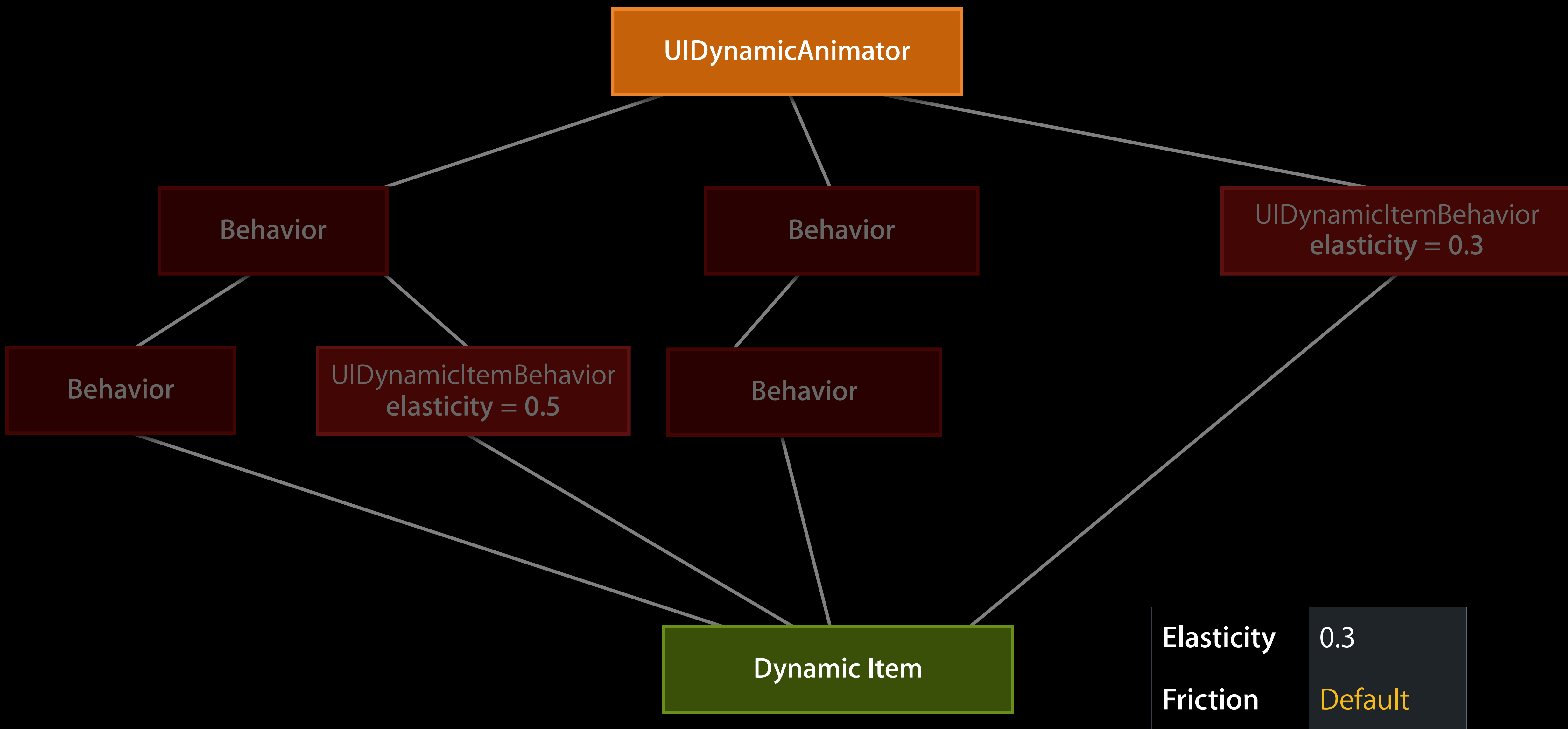
# Using Multiple UIDynamicItemBehaviors



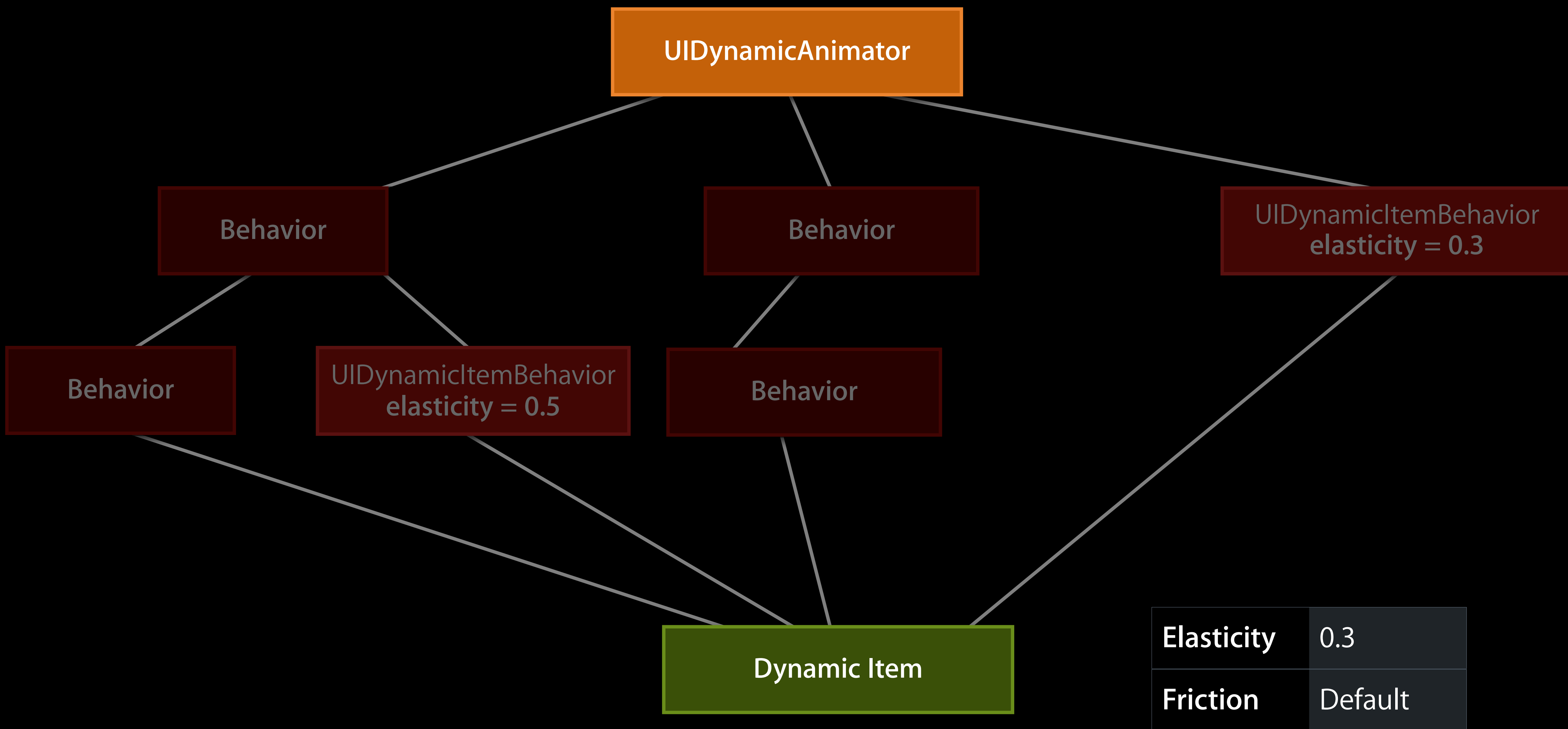
# Using Multiple UIDynamicItemBehaviors



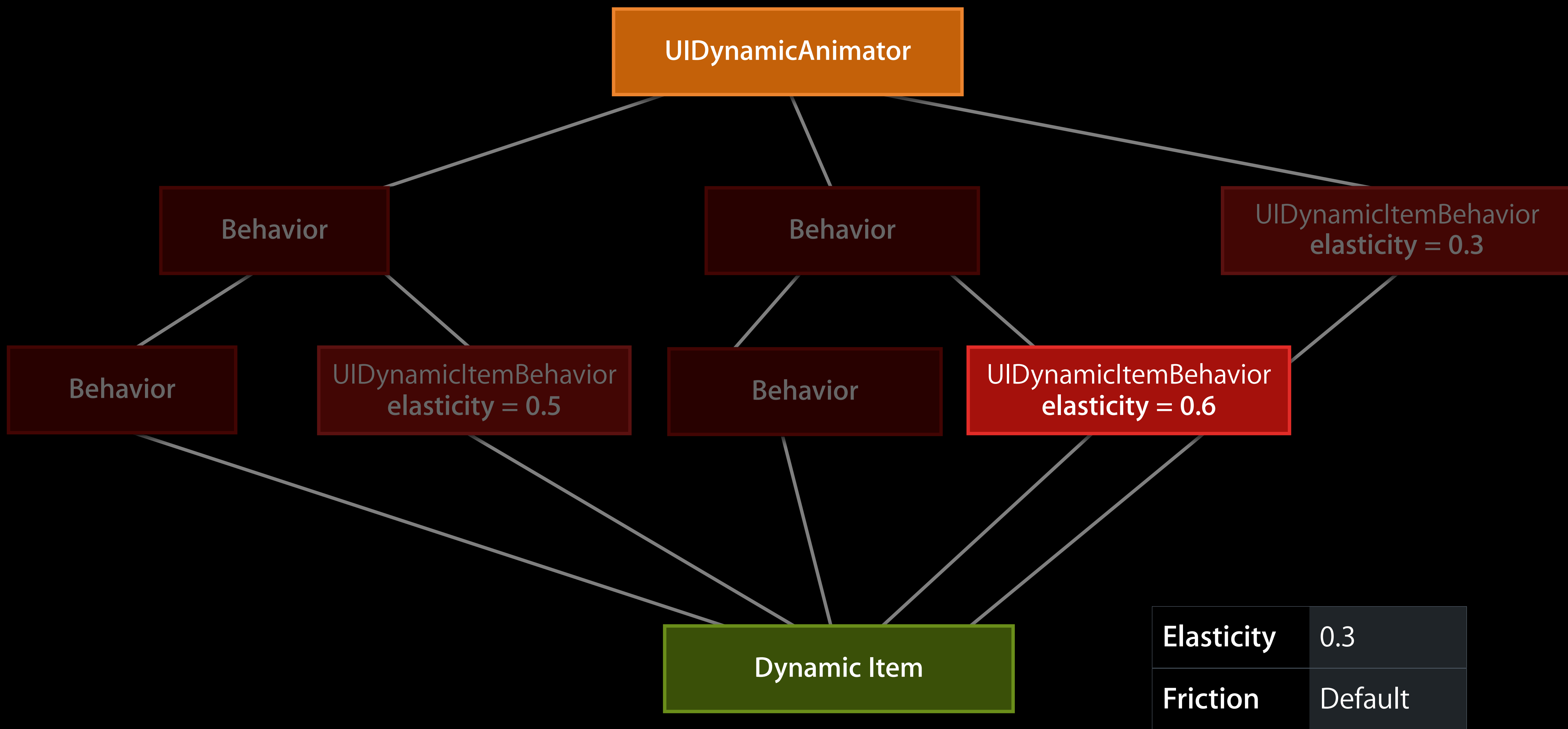
# Using Multiple UIDynamicItemBehaviors



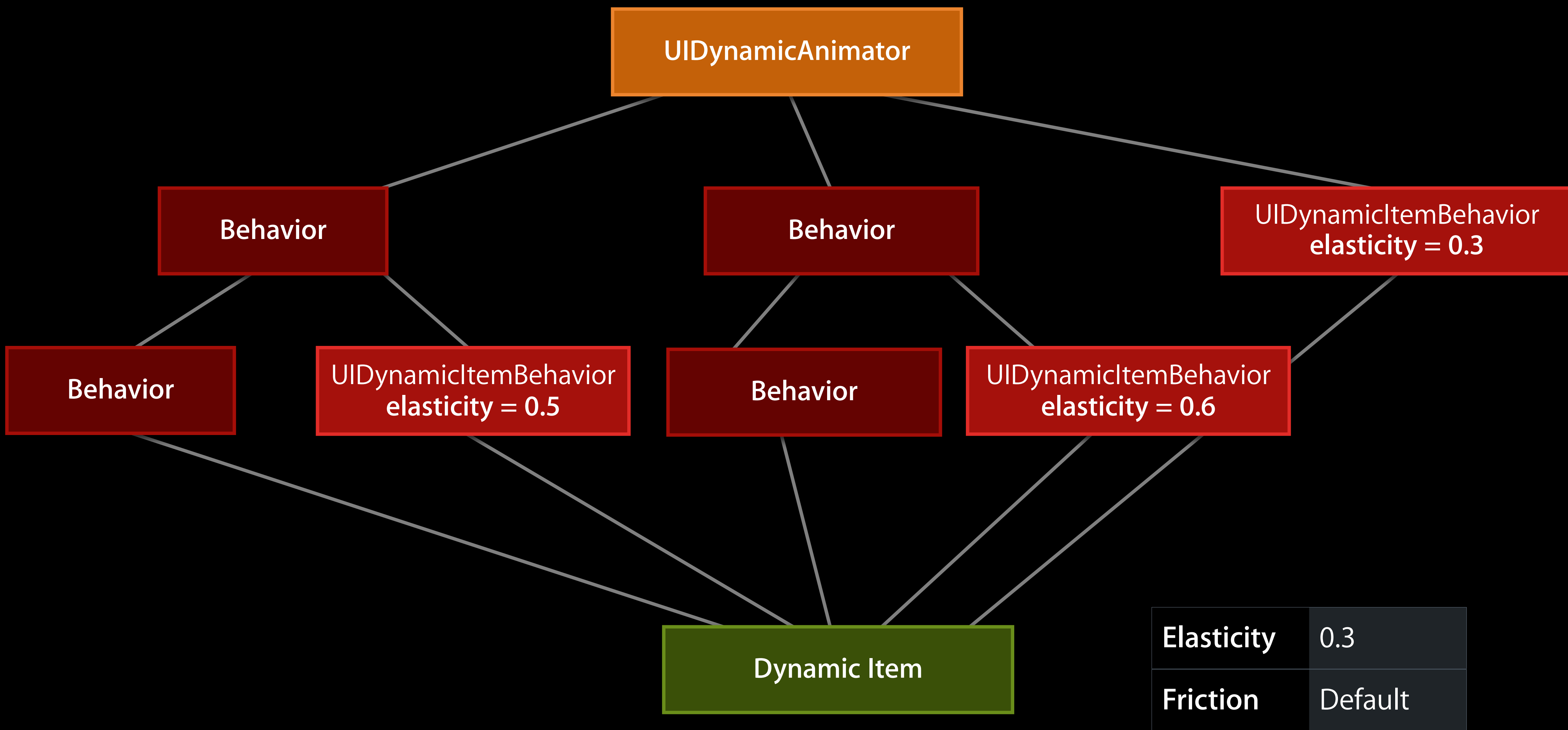
# Using Multiple UIDynamicItemBehaviors



# Using Multiple UIDynamicItemBehaviors



# Using Multiple UIDynamicItemBehaviors



# Creating Custom Dynamic Items

UIDynamicItem

# UIDynamicItem Protocol



# UIDynamicItem Protocol

- A way to integrate non-views items in behaviors

# UIDynamicItem Protocol

- A way to integrate non-views items in behaviors
- A protocol that all items animated by UIKit Dynamics must implement

# UIDynamicItem Protocol

- A way to integrate non-views items in behaviors
- A protocol that all items animated by UIKit Dynamics must implement
  - Center

# UIDynamicItem Protocol

- A way to integrate non-views items in behaviors
- A protocol that all items animated by UIKit Dynamics must implement
  - Center
  - Bounds

# UIDynamicItem Protocol

- A way to integrate non-views items in behaviors
- A protocol that all items animated by UIKit Dynamics must implement
  - Center
  - Bounds
  - Transform

# UIDynamicItem Protocol

- A way to integrate non-views items in behaviors
- A protocol that all items animated by UIKit Dynamics must implement
  - Center
  - Bounds
  - Transform
- Implemented by UIView and UICollectionViewLayoutAttributes

# UIDynamicItem Protocol

- A way to integrate non-views items in behaviors
- A protocol that all items animated by UIKit Dynamics must implement
  - Center
  - Bounds
  - Transform
- Implemented by UIView and UICollectionViewLayoutAttributes
- Only 2D-rotation transforms are supported

# UIDynamicItem Protocol



# UIDynamicItem Protocol

- Center, Bounds, and Transform are read only once by UIKit

# UIDynamicItem Protocol

- Center, Bounds, and Transform are read only once by UIKit
  - When adding the item to an animator for the first time

# UIDynamicItem Protocol

- Center, Bounds, and Transform are read only once by UIKit
  - When adding the item to an animator for the first time
- Center and Transform are set on every animation tick

# UIDynamicItem Protocol

- Center, Bounds, and Transform are read only once by UIKit
  - When adding the item to an animator for the first time
- Center and Transform are set on every animation tick
  - Performance is critical

# UIDynamicItem Protocol

# UIDynamicItem Protocol

- Will ignore any external change to Center, Transform, and Bounds

# UIDynamicItem Protocol

- Will ignore any external change to Center, Transform, and Bounds
- How to change the size of an item?

# UIDynamicItem Protocol

- Will ignore any external change to Center, Transform, and Bounds
- How to change the size of an item?
  - Remove it from its behaviors (and add it later), or



# UIDynamicItem Protocol

- Will ignore any external change to Center, Transform, and Bounds
- How to change the size of an item?
  - Remove it from its behaviors (and add it later), or
  - Change a subview, or

# UIDynamicItem Protocol

- Will ignore any external change to Center, Transform, and Bounds
- How to change the size of an item?
  - Remove it from its behaviors (and add it later), or
  - Change a subview, or
  - ...

# UIDynamicItem Protocol

- Will ignore any external change to Center, Transform, and Bounds
- How to change the size of an item?
  - Remove it from its behaviors (and add it later), or
  - Change a subview, or
  - ...
- A dynamic item should always have a valid default state

# UIDynamicItem Protocol

- Will ignore any external change to Center, Transform, and Bounds
- How to change the size of an item?
  - Remove it from its behaviors (and add it later), or
  - Change a subview, or
  - ...
- A dynamic item should always have a valid default state
  - Non zero size

# UIDynamicItem Protocol

- Will ignore any external change to Center, Transform, and Bounds
- How to change the size of an item?
  - Remove it from its behaviors (and add it later), or
  - Change a subview, or
  - ...
- A dynamic item should always have a valid default state
  - Non zero size
  - Reasonable position

# UIDynamicItem Use Cases

# UIDynamicItem Use Cases

- Sanitize or clamp values before applying changes to your views

# UIDynamicItem Use Cases

- Sanitize or clamp values before applying changes to your views
- Update multiple views from a single dynamic item



# UIDynamicItem Use Cases

- Sanitize or clamp values before applying changes to your views
- Update multiple views from a single dynamic item
- Map a position or angle to different properties

# UIDynamicItem Use Cases

- Sanitize or clamp values before applying changes to your views
- Update multiple views from a single dynamic item
- Map a position or angle to different properties
- Do not define a phantom view hierarchy

# UIDynamicItem Use Cases

- Sanitize or clamp values before applying changes to your views
- Update multiple views from a single dynamic item
- Map a position or angle to different properties
- Do not define a phantom view hierarchy
  - Use a dynamic item instead!

# Example

```
@interface ASCIIIDynamicItem : NSObject <UIDynamicItem>

@property (nonatomic, readonly) CGRect bounds;
@property (nonatomic, readwrite) CGPoint center;
@property (nonatomic, readwrite) CGAffineTransform transform;

@end
```

# Example

```
@implementation ASCIIIDynamicItem
-(CGRect)bounds {
    return CGRectMake(0.0, 0.0, 100.0, 100.0);
}
-(CGPoint)center {
    return CGPointMake(50.0, 50.0);
}
-(CGAffineTransform)transform {
    return CGAffineTransformIdentity;
}
-(void)setCenter:(CGPoint)center {
    NSLog(@"Center: %@", NSStringFromCGPoint(center));
}
-(void)setTransform:(CGAffineTransform)transform {
    NSLog(@"Transform: %@", NSStringFromCGAffineTransform(transform));
}
@end
```

# Collection View Meets Dynamics

# UIKit Dynamics and Collection Views

# UIKit Dynamics and Collection Views

- Use dynamics for specific animations



# UIKit Dynamics and Collection Views

- Use dynamics for specific animations
  - Create an animator as needed and discard it later

# UIKit Dynamics and Collection Views

- Use dynamics for specific animations
  - Create an animator as needed and discard it later
- Animate a subset of a layout

# UIKit Dynamics and Collection Views

- Use dynamics for specific animations
  - Create an animator as needed and discard it later
- Animate a subset of a layout
- Build an entire layout with UIKit Dynamics

# UIKit Dynamics and Collection Views

- Use dynamics for specific animations
  - Create an animator as needed and discard it later
- Animate a subset of a layout
- Build an entire layout with UIKit Dynamics
  - Only for small data sources!

# UIKit Dynamics and Collection Views

# UIKit Dynamics and Collection Views

- You must provide the initial state for your items

# UIKit Dynamics and Collection Views

- You must provide the initial state for your items
  - Prepare layout attributes

# UIKit Dynamics and Collection Views

- You must provide the initial state for your items
  - Prepare layout attributes
  - ...or subclass and existing layout



# UIKit Dynamics and Collection Views

- You must provide the initial state for your items
  - Prepare layout attributes
  - ...or subclass and existing layout
  - ...or add new items on the fly

# UIKit Dynamics and Collection Views

- You must provide the initial state for your items
  - Prepare layout attributes
  - ...or subclass and existing layout
  - ...or add new items on the fly
- Initialize an animator with your collection view layout

# UIKit Dynamics and Collection Views

- You must provide the initial state for your items
  - Prepare layout attributes
  - ...or subclass and existing layout
  - ...or add new items on the fly
- Initialize an animator with your collection view layout
- Add behaviors

# UIKit Dynamics and Collection Views

- You must provide the initial state for your items
  - Prepare layout attributes
  - ...or subclass and existing layout
  - ...or add new items on the fly
- Initialize an animator with your collection view layout
- Add behaviors
- Add `UICollectionViewLayoutAttributes` items to behaviors

# UIKit Dynamics Support for Collection Views

# UIKit Dynamics Support for Collection Views

- Will invalidate layout as needed

# UIKit Dynamics Support for Collection Views

- Will invalidate layout as needed
- Will pause the animator when a layout is no longer associated

# UIKit Dynamics Support for Collection Views

- Will invalidate layout as needed
- Will pause the animator when a layout is no longer associated
- Provide convenience methods



# UIKit Dynamics Support for Collection Views

- Will invalidate layout as needed
- Will pause the animator when a layout is no longer associated
- Provide convenience methods
  - (UICollectionViewLayoutAttributes\*)`layoutAttributesForCellAtIndex`Path:

# UIKit Dynamics Support for Collection Views

- Will invalidate layout as needed
- Will pause the animator when a layout is no longer associated
- Provide convenience methods
  - (UICollectionViewLayoutAttributes\*)layoutAttributesForCellAtIndexPath:
  - (UICollectionViewLayoutAttributes\*)layoutAttributesForSupplementaryViewOfKind:  
atIndexPath:

# UIKit Dynamics Support for Collection Views

- Will invalidate layout as needed
- Will pause the animator when a layout is no longer associated
- Provide convenience methods
  - (UICollectionViewLayoutAttributes\*)layoutAttributesForCellAtIndex:  
atIndexPath:
  - (UICollectionViewLayoutAttributes\*)layoutAttributesForSupplementaryViewOfKind:  
atIndexPath:
  - (UICollectionViewLayoutAttributes\*)layoutAttributesForDecorationViewOfKind:  
atIndexPath:

# Layout Updates

# Layout Updates

- Use standard collection view layout methods

# Layout Updates

- Use standard collection view layout methods  
`prepareLayout`

# Layout Updates

- Use standard collection view layout methods  
prepareLayout
  - Create initial setup

# Layout Updates

- Use standard collection view layout methods

prepareLayout

- Create initial setup

prepareForUpdate:



# Layout Updates

- Use standard collection view layout methods

`prepareLayout`

- Create initial setup

`prepareForUpdate:`

- Opportunity to add attributes to behaviors

# Layout Updates

- Use standard collection view layout methods

`prepareLayout`

- Create initial setup

`prepareForUpdate:`

- Opportunity to add attributes to behaviors

`layoutAttributesInRect:`

# Layout Updates

- Use standard collection view layout methods

`prepareLayout`

- Create initial setup

`prepareForUpdate:`

- Opportunity to add attributes to behaviors

`layoutAttributesInRect:`

- High performance `itemsInRect:` method on `UIDynamicAnimator`

# Layout Updates

- Use standard collection view layout methods

`prepareLayout`

- Create initial setup

`prepareForUpdate:`

- Opportunity to add attributes to behaviors

`layoutAttributesInRect:`

- High performance **`itemsInRect:`** method on `UIDynamicAnimator`
- Combine with your non-dynamics attributes

# Layout Updates

- Use standard collection view layout methods

`prepareLayout`

- Create initial setup

`prepareForUpdate:`

- Opportunity to add attributes to behaviors

`layoutAttributesInRect:`

- High performance **`itemsInRect:`** method on `UIDynamicAnimator`
- Combine with your non-dynamics attributes
- Off screen items might influence on screen items!

*Demo*

# How to Build This?

# How to Build This?

- Decompose

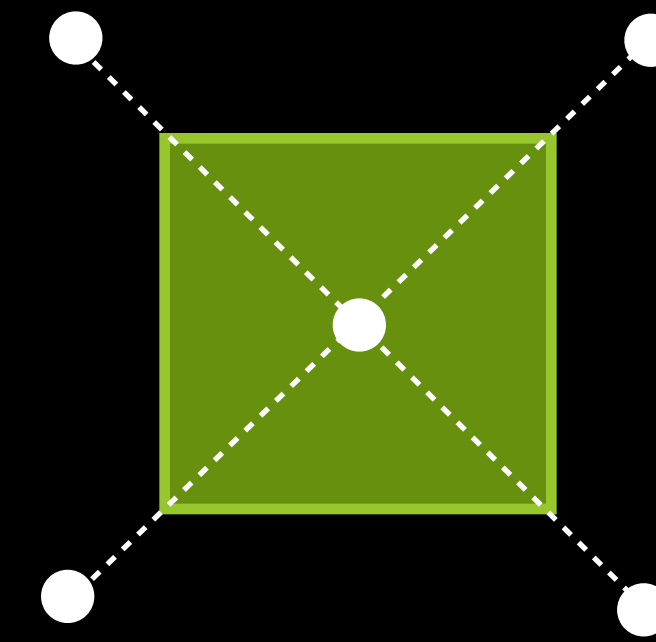


# How to Build This?

- Decompose
- “Attached to a rectangle” behavior

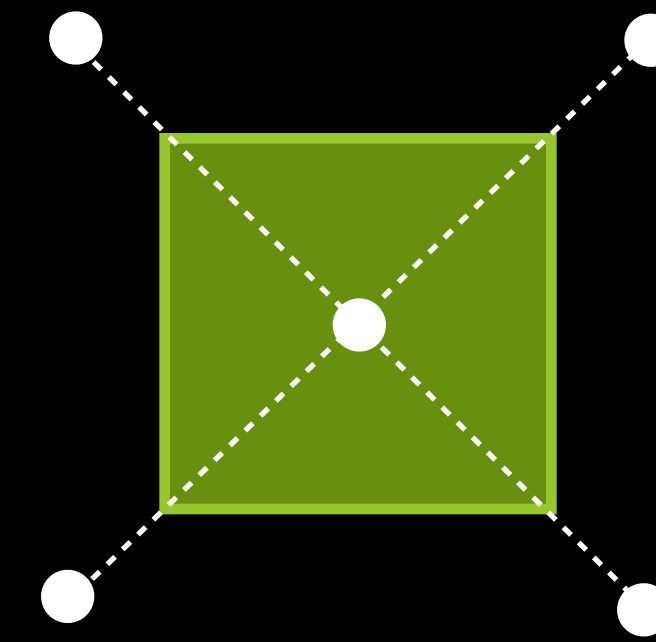
# How to Build This?

- Decompose
- “Attached to a rectangle” behavior



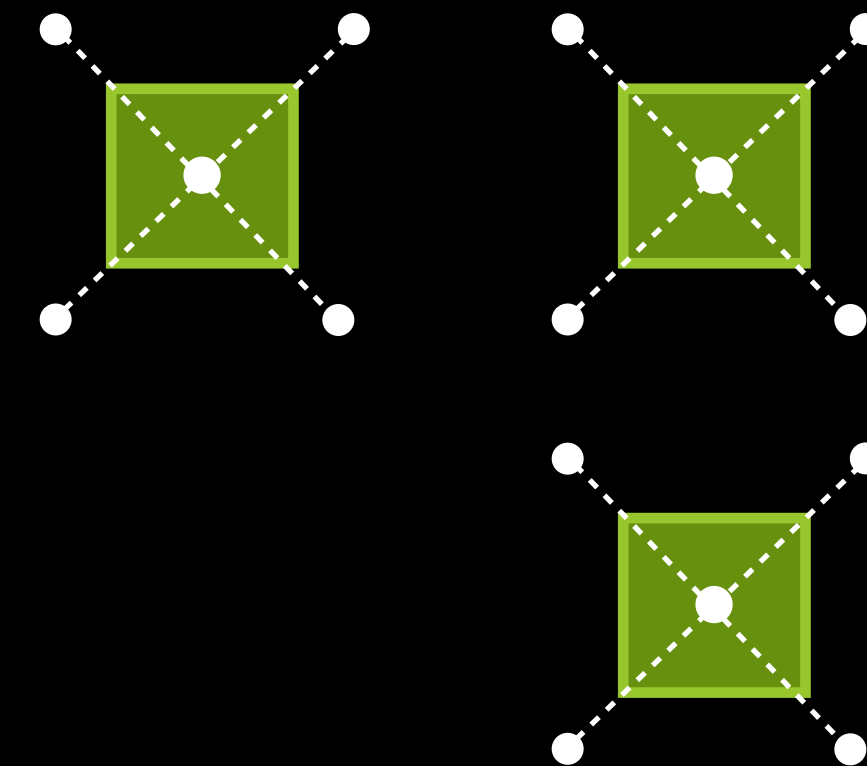
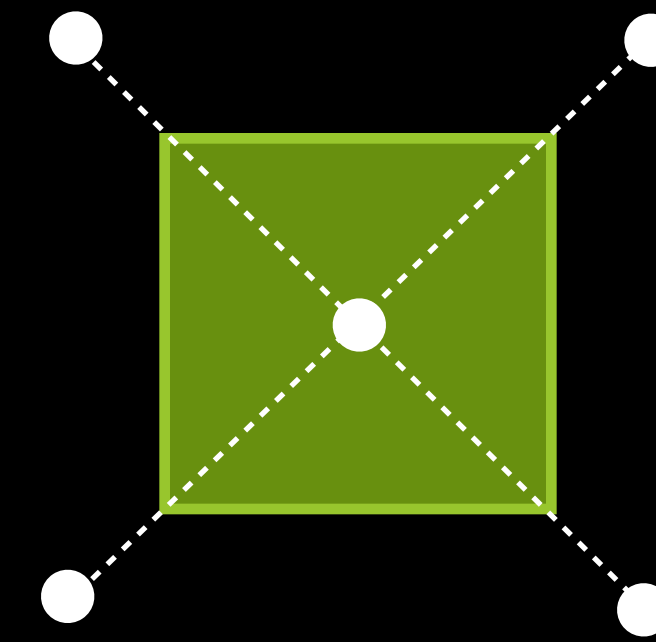
# How to Build This?

- Decompose
- “Attached to a rectangle” behavior
- “Drag many items” behavior



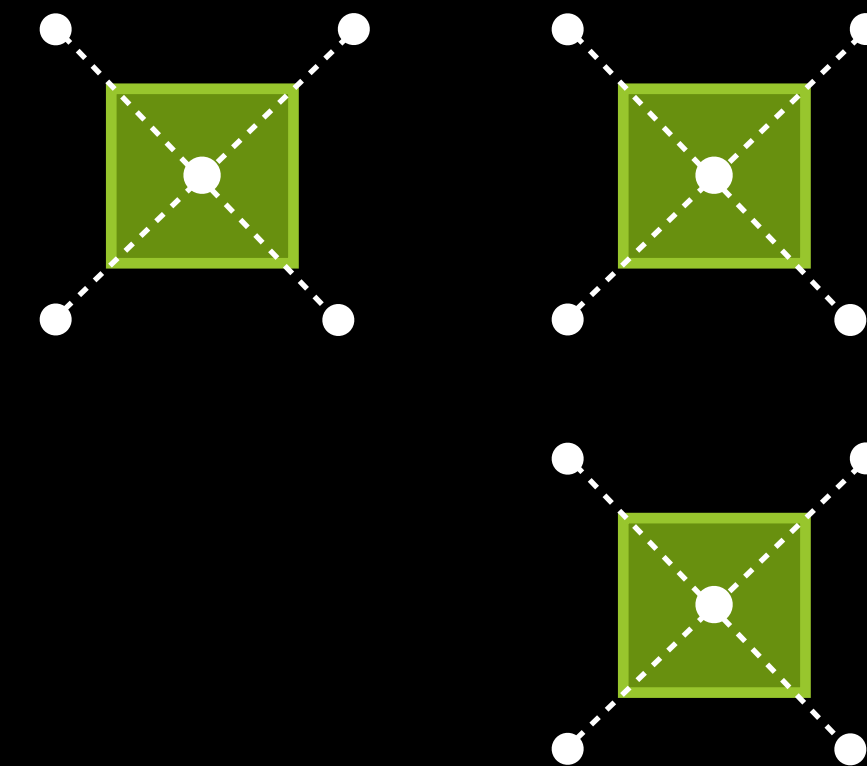
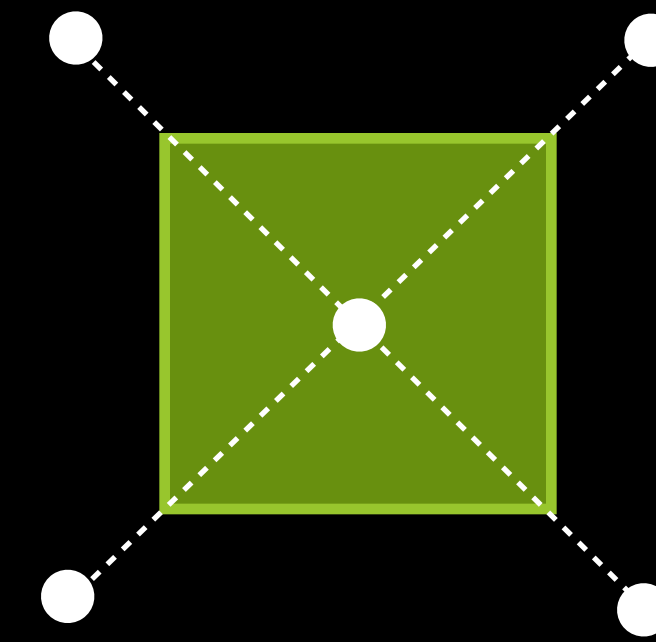
# How to Build This?

- Decompose
- “Attached to a rectangle” behavior
- “Drag many items” behavior



# How to Build This?

- Decompose
- “Attached to a rectangle” behavior
- “Drag many items” behavior
- Used in a flow layout subclass



# Interfaces

```
@interface DraggableLayout : UICollectionViewFlowLayout
```

- (void)setDraggedIndexPaths:(NSArray \*)selectedIndexPaths fromPoint:(CGPoint)p;
- (void)updateDragLocation:(CGPoint)p;
- (void)clearDraggedIndexPaths;

```
@end
```

```
@interface DragBehavior : UIDynamicBehavior
```

- (instancetype)initWithItems:(NSArray\*)items point:(CGPoint)p;
- (void)updateDragLocation:(CGPoint)p;

```
@end
```

```
@interface RectangleAttachmentBehavior : UIDynamicBehavior
```

- (instancetype)initWithItem:(id <UIDynamicItem>)item point:(CGPoint)p;
- (void)updateAttachmentLocation:(CGPoint)p;

```
@end
```

# Interfaces

```
@interface DraggableLayout : UICollectionViewFlowLayout
```

- (void)setDraggedIndexPaths:(NSArray \*)selectedIndexPaths fromPoint:(CGPoint)p;
- (void)updateDragLocation:(CGPoint)p;
- (void)clearDraggedIndexPaths;

```
@end
```

```
@interface DragBehavior : UIDynamicBehavior
```

- (instancetype)initWithItems:(NSArray\*)items point:(CGPoint)p;
- (void)updateDragLocation:(CGPoint)p;

```
@end
```

```
@interface RectangleAttachmentBehavior : UIDynamicBehavior
```

- (instancetype)initWithItem:(id <UIDynamicItem>)item point:(CGPoint)p;
- (void)updateAttachmentLocation:(CGPoint)p;

```
@end
```

# Interfaces

```
@interface DraggableLayout : UICollectionViewFlowLayout
- (void)setDraggedIndexPaths:(NSArray *)selectedIndexPaths fromPoint:(CGPoint)p;
- (void)updateDragLocation:(CGPoint)p;
- (void)clearDraggedIndexPaths;
@end
```

```
@interface DragBehavior : UIDynamicBehavior
- (instancetype)initWithItems:(NSArray*)items point:(CGPoint)p;
- (void)updateDragLocation:(CGPoint)p;
@end
```

```
@interface RectangleAttachmentBehavior : UIDynamicBehavior
- (instancetype)initWithItem:(id <UIDynamicItem>)item point:(CGPoint)p;
- (void)updateAttachmentLocation:(CGPoint)p;
@end
```



# Interfaces

```
@interface DraggableLayout : UICollectionViewFlowLayout
```

- (void)setDraggedIndexPaths:(NSArray \*)selectedIndexPaths fromPoint:(CGPoint)p;
- (void)updateDragLocation:(CGPoint)p;
- (void)clearDraggedIndexPaths;

```
@end
```

```
@interface DragBehavior : UIDynamicBehavior
```

- (instancetype)initWithItems:(NSArray\*)items point:(CGPoint)p;
- (void)updateDragLocation:(CGPoint)p;

```
@end
```

```
@interface RectangleAttachmentBehavior : UIDynamicBehavior
```

- (instancetype)initWithItem:(id <UIDynamicItem>)item point:(CGPoint)p;
- (void)updateAttachmentLocation:(CGPoint)p;

```
@end
```

# Interfaces

```
@interface DraggableLayout : UICollectionViewFlowLayout
```

- (void)setDraggedIndexPaths:(NSArray \*)selectedIndexPaths fromPoint:(CGPoint)p;
- (void)updateDragLocation:(CGPoint)p;
- (void)clearDraggedIndexPaths;

```
@end
```

```
@interface DragBehavior : UIDynamicBehavior
```

- (instancetype)initWithItems:(NSArray\*)items point:(CGPoint)p;
- (void)updateDragLocation:(CGPoint)p;

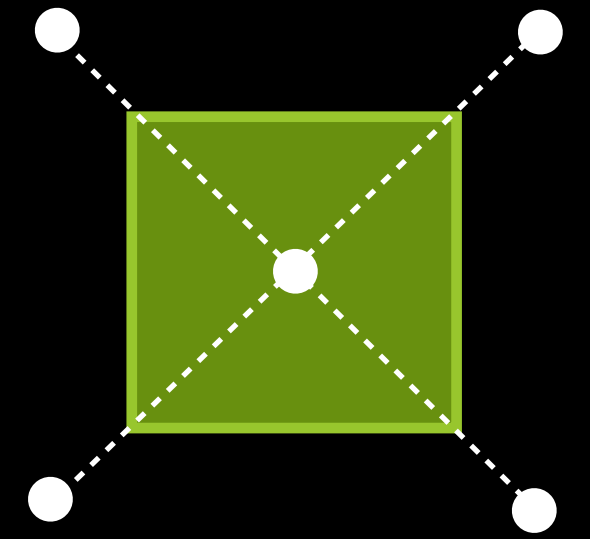
```
@end
```

```
@interface RectangleAttachmentBehavior : UIDynamicBehavior
```

- (instancetype)initWithItem:(id <UIDynamicItem>)item point:(CGPoint)p;
- (void)updateAttachmentLocation:(CGPoint)p;

```
@end
```

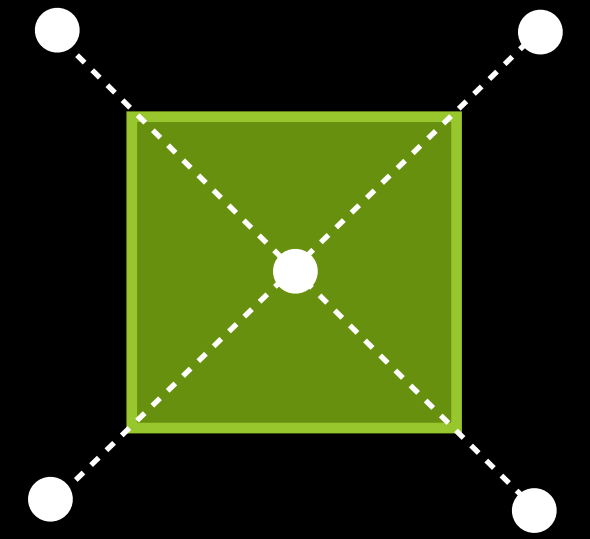
# RectangleAttachmentBehavior



```
@implementation RectangleAttachmentBehavior
```

```
-(instancetype)initWithItem:(id <UIDynamicItem>)item point:(CGPoint)p {  
    if (self = [super init]) {  
        CGPoint topLeft      = CGPointMake(p.x - WIDTH / 2.0, p.y - HEIGHT / 2.0);  
        CGPoint topRight     = ...  
  
        UIAttachmentBehavior* attachmentBehavior;  
        attachmentBehavior = [[UIAttachmentBehavior alloc] initWithItem:item  
  attachedToAnchor:topLeft];  
  
        [attachmentBehavior setFrequency:FREQUENCY];  
        [attachmentBehavior setDamping:DAMPING];  
        [self addChildBehavior:attachmentBehavior];  
  
        attachmentBehavior = [[UIAttachmentBehavior alloc] initWithItem:item  
  attachedToAnchor:topRight];  
  
        ...  
    }  
}
```

# RectangleAttachmentBehavior

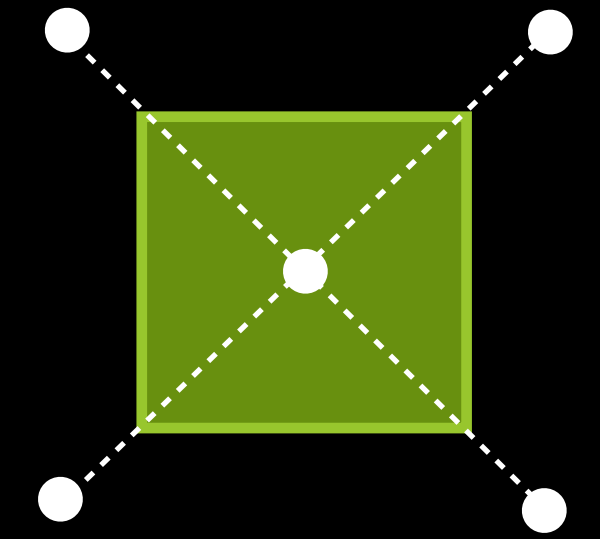


```
@implementation RectangleAttachmentBehavior
```

```
-(instancetype)initWithItem:(id <UIDynamicItem>)item point:(CGPoint)p {  
    if (self = [super init]) {  
        CGPoint topLeft      = CGPointMake(p.x - WIDTH / 2.0, p.y - HEIGHT / 2.0);  
        CGPoint topRight     = ...  
    }  
}
```

```
UIAttachmentBehavior* attachmentBehavior;  
attachmentBehavior = [[UIAttachmentBehavior alloc] initWithItem:item  
   attachedToAnchor:topLeft];  
[attachmentBehavior setFrequency:FREQUENCY];  
[attachmentBehavior setDamping:DAMPING];  
[self addChildBehavior:attachmentBehavior];  
  
attachmentBehavior = [[UIAttachmentBehavior alloc] initWithItem:item  
   attachedToAnchor:topRight];  
...  
...
```

# RectangleAttachmentBehavior



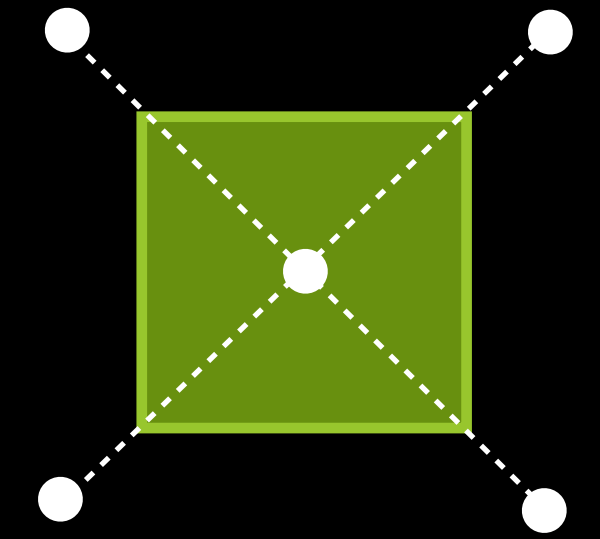
```
@implementation RectangleAttachmentBehavior
```

```
-(instancetype)initWithItem:(id <UIDynamicItem>)item point:(CGPoint)p {  
    if (self = [super init]) {  
        CGPoint topLeft      = CGPointMake(p.x - WIDTH / 2.0, p.y - HEIGHT / 2.0);  
        CGPoint topRight     = ...
```

```
        UIAttachmentBehavior* attachmentBehavior;  
        attachmentBehavior = [[UIAttachmentBehavior alloc] initWithItem:item  
  attachedToAnchor:topLeft];  
        [attachmentBehavior setFrequency:FREQUENCY];  
        [attachmentBehavior setDamping:DAMPING];  
        [self addChildBehavior:attachmentBehavior];
```

```
        attachmentBehavior = [[UIAttachmentBehavior alloc] initWithItem:item  
  attachedToAnchor:topRight];  
        ...
```

# RectangleAttachmentBehavior



```
@implementation RectangleAttachmentBehavior
```

```
-(instancetype)initWithItem:(id <UIDynamicItem>)item point:(CGPoint)p {  
    if (self = [super init]) {  
        CGPoint topLeft      = CGPointMake(p.x - WIDTH / 2.0, p.y - HEIGHT / 2.0);  
        CGPoint topRight     = ...
```

```
        UIAttachmentBehavior* attachmentBehavior;  
        attachmentBehavior = [[UIAttachmentBehavior alloc] initWithItem:item  
  attachedToAnchor:topLeft];  
        [attachmentBehavior setFrequency:FREQUENCY];  
        [attachmentBehavior setDamping:DAMPING];  
        [self addChildBehavior:attachmentBehavior];
```

```
        attachmentBehavior = [[UIAttachmentBehavior alloc] initWithItem:item  
  attachedToAnchor:topRight];  
        ...
```

# RectangleAttachmentBehavior

```
@implementation RectangleAttachmentBehavior
```

```
-(void)updateAttachmentLocation:(CGPoint)p {  
    CGPoint topLeft      = CGPointMake(p.x - WIDTH / 2.0, p.y - HEIGHT / 2.0);  
    CGPoint topRight     = ...  
    ....  
    UIAttachmentBehavior* attachmentBehavior;  
    attachmentBehavior = [[self childBehaviors] objectAtIndex:0];  
    attachmentBehavior.anchorPoint = topLeft;  
    attachmentBehavior = [[self childBehaviors] objectAtIndex:1];  
    attachmentBehavior.anchorPoint = topRight;  
    ....  
}
```



# RectangleAttachmentBehavior

@implementation RectangleAttachmentBehavior

```
-(void)updateAttachmentLocation:(CGPoint)p {  
    CGPoint topLeft      = CGPointMake(p.x - WIDTH / 2.0, p.y - HEIGHT / 2.0);  
    CGPoint topRight     = ...  
    ....  
    UIAttachmentBehavior* attachmentBehavior;  
    attachmentBehavior = [[self childBehaviors] objectAtIndex:0];  
    attachmentBehavior.anchorPoint = topLeft;  
    attachmentBehavior = [[self childBehaviors] objectAtIndex:1];  
    attachmentBehavior.anchorPoint = topRight;  
    ....  
}
```



# RectangleAttachmentBehavior

```
@implementation RectangleAttachmentBehavior
```

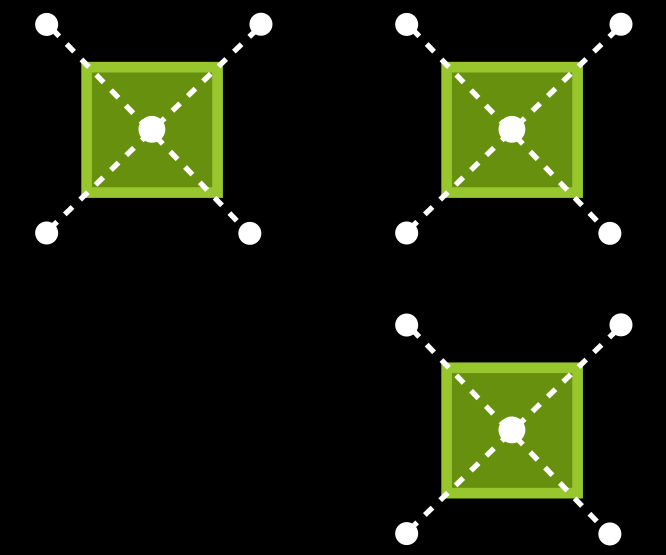
```
-(void)updateAttachmentLocation:(CGPoint)p {  
    CGPoint topLeft      = CGPointMake(p.x - WIDTH / 2.0, p.y - HEIGHT / 2.0);  
    CGPoint topRight     = ...  
    ....  
    UIAttachmentBehavior* attachmentBehavior;  
    attachmentBehavior = [[self childBehaviors] objectAtIndex:0];  
    attachmentBehavior.anchorPoint = topLeft;  
    attachmentBehavior = [[self childBehaviors] objectAtIndex:1];  
    attachmentBehavior.anchorPoint = topRight;  
    ....  
}
```

# RectangleAttachmentBehavior

```
@implementation RectangleAttachmentBehavior
```

```
-(void)updateAttachmentLocation:(CGPoint)p {  
    CGPoint topLeft      = CGPointMake(p.x - WIDTH / 2.0, p.y - HEIGHT / 2.0);  
    CGPoint topRight     = ...  
    ....  
    UIAttachmentBehavior* attachmentBehavior;  
    attachmentBehavior = [[self childBehaviors] objectAtIndex:0];  
    attachmentBehavior.anchorPoint = topLeft;  
    attachmentBehavior = [[self childBehaviors] objectAtIndex:1];  
    attachmentBehavior.anchorPoint = topRight;  
    ....  
}
```

# DragBehavior

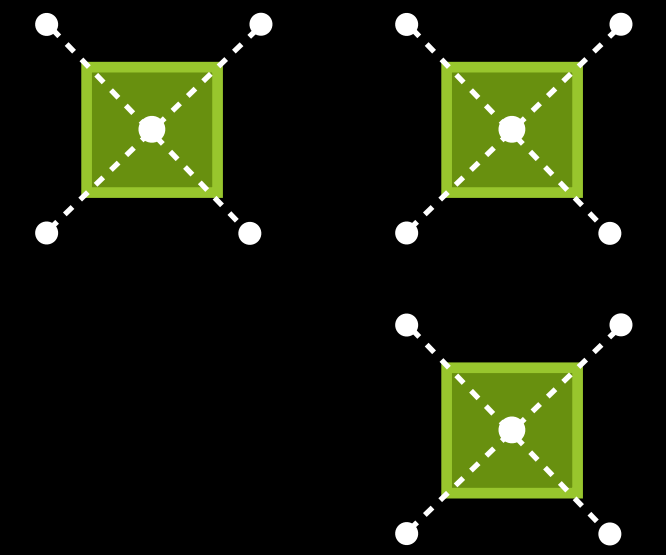


```
@implementation DragBehavior
```

```
-(instancetype)initWithItems:(NSArray*)items point:(CGPoint)p {
    if (self = [super init]) {
        for (id <UIDynamicItem> item in items) {
            RectangleAttachmentBehavior* rectangleAttachment =
                [[RectangleAttachmentBehavior alloc] initWithItem:item point:p];
            [self addChildBehavior:rectangleAttachment];
        }
    }
    return self;
}

- (void)updateDragLocation:(CGPoint)p {
    for (RectangleAttachmentBehavior* behavior in [self childBehaviors]) {
        [behavior updateAttachmentLocation:p];
    }
}
```

# DragBehavior



```
@implementation DragBehavior
```

```
-(instancetype)initWithItems:(NSArray*)items point:(CGPoint)p {
```

```
    if (self = [super init]) {
```

```
        for (id <UIDynamicItem> item in items) {
```

```
            RectangleAttachmentBehavior* rectangleAttachment =
```

```
                [[RectangleAttachmentBehavior alloc] initWithItem:item point:p];
```

```
            [self addChildBehavior:rectangleAttachment];
```

```
        }
```

```
    }
```

```
    return self;
```

```
}
```

```
-(void)updateDragLocation:(CGPoint)p {
```

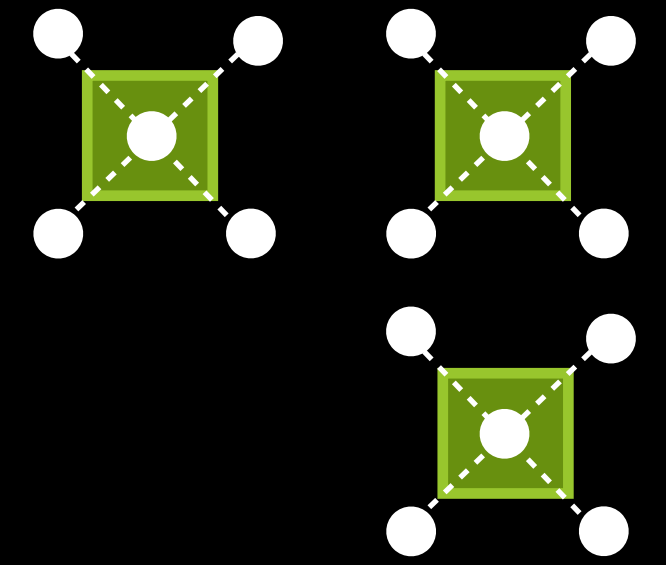
```
    for (RectangleAttachmentBehavior* behavior in [self childBehaviors]) {
```

```
        [behavior updateAttachmentLocation:p];
```

```
    }
```

```
}
```

# DragBehavior



```
@implementation DragBehavior
```

```
-(instancetype)initWithItems:(NSArray*)items point:(CGPoint)p {  
    if (self = [super init]) {  
        for (id <UIDynamicItem> item in items) {  
            RectangleAttachmentBehavior* rectangleAttachment =  
                [[RectangleAttachmentBehavior alloc] initWithItem:item point:p];  
            [self addChildBehavior:rectangleAttachment];  
        }  
    }  
    return self;  
}  
  
- (void)updateDragLocation:(CGPoint)p {  
    for (RectangleAttachmentBehavior* behavior in [self childBehaviors]) {  
        [behavior updateAttachmentLocation:p];  
    }  
}
```

# DraggableLayout

## Interaction

```
- (void)startDraggingIndexPaths:(NSArray *)selectedIndexPaths fromPoint:(CGPoint)p {  
  
    _indexPathsForDraggedElements = selectedIndexPaths;  
    _animator = [[UIDynamicAnimator alloc] initWithCollectionViewLayout:self];  
  
    NSMutableArray* draggableAttributes = [NSMutableArray array];  
    for (NSIndexPath* path in _indexPathsForDraggedElements) {  
        UICollectionViewLayoutAttributes* attributes =  
            [super layoutAttributesForItemAtIndexPath:path];  
        attributes.zIndex = 1;  
        [draggableAttributes addObject:attributes];  
    }  
  
    _dragBehavior = [[DragBehavior alloc] initWithItems:draggableAttributes point:p];  
    [_animator addBehavior:_dragBehavior];  
}
```

# DraggableLayout

## Interaction

– (void)startDraggingIndexPaths:(NSArray \*)selectedIndexPaths fromPoint:(CGPoint)p {

```
_indexPathsForDraggedElements = selectedIndexPaths;  
_animator = [[UIDynamicAnimator alloc] initWithCollectionViewLayout:self];
```

```
NSMutableArray* draggableAttributes = [NSMutableArray array];  
for (NSIndexPath* path in _indexPathsForDraggedElements) {  
    UICollectionViewLayoutAttributes* attributes =  
        [super layoutAttributesForItemAtIndexPath:path];  
    attributes.zIndex = 1;  
    [draggableAttributes addObject:attributes];  
}
```

```
_dragBehavior = [[DragBehavior alloc] initWithItems:draggableAttributes point:p];  
[_animator addBehavior:_dragBehavior];  
}
```



# DraggableLayout

## Interaction

```
- (void)startDraggingIndexPaths:(NSArray *)selectedIndexPaths fromPoint:(CGPoint)p {  
    _indexPathsForDraggedElements = selectedIndexPaths;  
    _animator = [[UIDynamicAnimator alloc] initWithCollectionViewLayout:self];  
  
    NSMutableArray* draggableAttributes = [NSMutableArray array];  
    for (NSIndexPath* path in _indexPathsForDraggedElements) {  
        UICollectionViewLayoutAttributes* attributes =  
            [super layoutAttributesForItemAtIndexPath:path];  
        attributes.zIndex = 1;  
        [draggableAttributes addObject:attributes];  
    }  
  
    _dragBehavior = [[DragBehavior alloc] initWithItems:draggableAttributes point:p];  
    [_animator addBehavior:_dragBehavior];  
}
```



# DraggableLayout

## Interaction

```
- (void)startDraggingIndexPaths:(NSArray *)selectedIndexPaths fromPoint:(CGPoint)p {  
    _indexPathsForDraggedElements = selectedIndexPaths;  
    _animator = [[UIDynamicAnimator alloc] initWithCollectionViewLayout:self];  
  
    NSMutableArray* draggableAttributes = [NSMutableArray array];  
    for (NSIndexPath* path in _indexPathsForDraggedElements) {  
        UICollectionViewLayoutAttributes* attributes =  
            [super layoutAttributesForItemAtIndexPath:path];  
        attributes.zIndex = 1;  
        [draggableAttributes addObject:attributes];  
    }  
  
    _dragBehavior = [[DragBehavior alloc] initWithItems:draggableAttributes point:p];  
    [_animator addBehavior:_dragBehavior];  
}
```

# DraggableLayout

## Interaction

```
- (void)updateDragLocation:(CGPoint)p {  
    [_dragBehavior updateDragLocation:p];  
}  
  
- (void)clearDraggedIndexPaths {  
    _animator = nil;  
    _indexPathsForDraggedElements = nil;  
}
```

# DraggableLayout

## Interaction

```
- (void)updateDragLocation:(CGPoint)p {  
    [_dragBehavior updateDragLocation:p];  
}  
  
- (void)clearDraggedIndexPaths {  
    _animator = nil;  
    _indexPathsForDraggedElements = nil;  
}
```

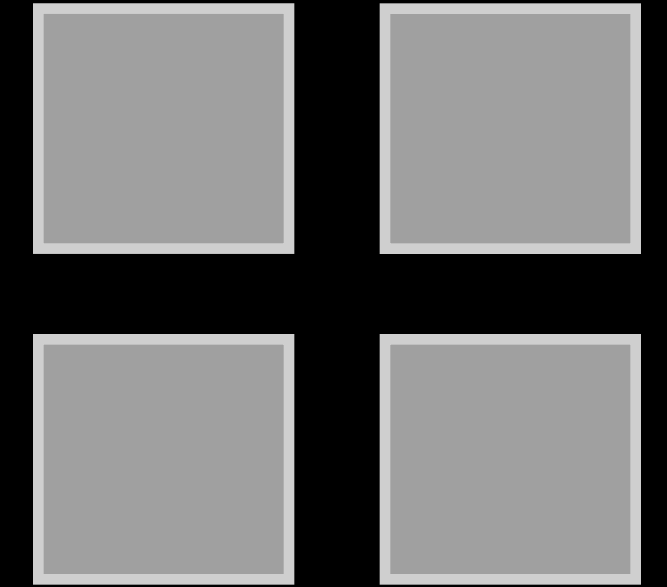
# DraggableLayout

## Interaction

```
- (void)updateDragLocation:(CGPoint)p {  
    [_dragBehavior updateDragLocation:p];  
}  
  
- (void)clearDraggedIndexPaths {  
    _animator = nil;  
    _indexPathsForDraggedElements = nil;  
}
```

# DraggableLayout

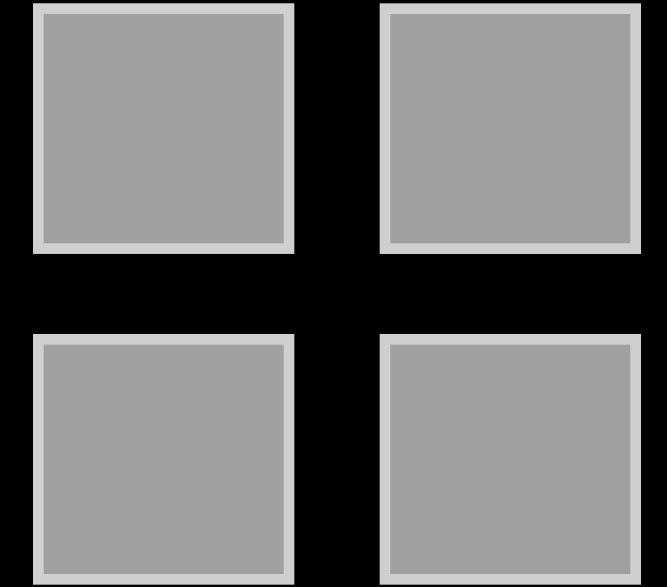
## Layout part



```
-(NSArray*) layoutAttributesForElementsInRect:(CGRect) rect
{
    NSArray* existingAttributes = [super layoutAttributesForElementsInRect:rect];
    NSMutableArray *allAttributes = [NSMutableArray array];
    for (UICollectionViewLayoutAttributes* attributes in existingAttributes) {
        if (![_indexPathsForDraggedElements containsObject:attributes.indexPath]) {
            [allAttributes addObject:attributes];
        }
    }
    [allAttributes addObjectsFromArray:[_animator itemsInRect:rect]];
    return allAttributes;
}
```

# DraggableLayout

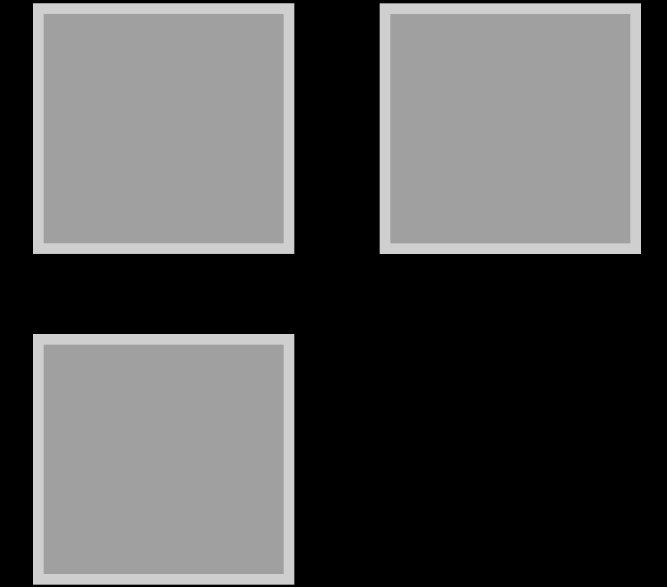
## Layout part



```
-(NSArray*) layoutAttributesForElementsInRect:(CGRect) rect
{
    NSArray* existingAttributes = [super layoutAttributesForElementsInRect:rect];
    NSMutableArray *allAttributes = [NSMutableArray array];
    for (UICollectionViewLayoutAttributes* attributes in existingAttributes) {
        if (![_indexPathsForDraggedElements containsObject:attributes.indexPath]) {
            [allAttributes addObject:attributes];
        }
    }
    [allAttributes addObjectsFromArray:[_animator itemsInRect:rect]];
    return allAttributes;
}
```

# DraggableLayout

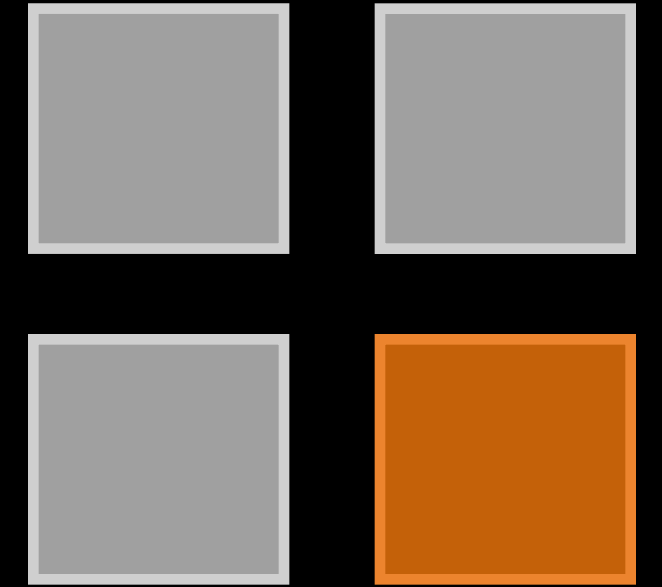
## Layout part



```
-(NSArray*) layoutAttributesForElementsInRect:(CGRect) rect
{
    NSArray* existingAttributes = [super layoutAttributesForElementsInRect:rect];
    NSMutableArray *allAttributes = [NSMutableArray array];
    for (UICollectionViewLayoutAttributes* attributes in existingAttributes) {
        if (![_indexPathsForDraggedElements containsObject:attributes.indexPath]) {
            [allAttributes addObject:attributes];
        }
    }
    [allAttributes addObjectsFromArray:[_animator itemsInRect:rect]];
    return allAttributes;
}
```

# DraggableLayout

## Layout part



```
-(NSArray*) layoutAttributesForElementsInRect:(CGRect) rect
{
    NSArray* existingAttributes = [super layoutAttributesForElementsInRect:rect];
    NSMutableArray *allAttributes = [NSMutableArray array];
    for (UICollectionViewLayoutAttributes* attributes in existingAttributes) {
        if (![_indexPathsForDraggedElements containsObject:attributes.indexPath]) {
            [allAttributes addObject:attributes];
        }
    }
    [allAttributes addObjectsFromArray:[_animator itemsInRect:rect]];
    return allAttributes;
}
```



# UIKit Dynamics and UIViewController Transitions

# New UIViewController Transition APIs

A quick review

# New UIViewController Transition APIs

## A quick review

- Your applications vends objects that conform to two protocols
  - <UIViewControllerAnimatedTransitioning>
    - (void)**animateTransition**:(id <UIViewControllerContextTransitioning>)
  - <UIViewControllerInteractiveTransitioning>
    - (void)**startInteractiveTransition**: (id <UIViewControllerContextTransitioning>)

# New UIViewController Transition APIs

## A quick review

- Your applications vends objects that conform to two protocols
  - <UIViewControllerAnimatedTransitioning>
    - (void)**animateTransition**:(id <UIViewControllerContextTransitioning>)
  - <UIViewControllerInteractiveTransitioning>
    - (void)**startInteractiveTransition**: (id <UIViewControllerContextTransitioning>)

# New UIViewController Transition APIs

## A quick review

- Your applications vends objects that conform to two protocols
  - <UIViewControllerAnimatedTransitioning>
    - (void)**animateTransition**:(id <UIViewControllerContextTransitioning>)
  - <UIViewControllerInteractiveTransitioning>
    - (void)**startInteractiveTransition**: (id <UIViewControllerContextTransitioning>)

# New UIViewController Transition APIs

## A quick review

- Your applications vends objects that conform to two protocols
  - `<UIViewControllerAnimatedTransitioning>`
    - `(void)animateTransition:(id <UIViewControllerContextTransitioning>)`
  - `<UIViewControllerInteractiveTransitioning>`
    - `(void)startInteractiveTransition: (id <UIViewControllerContextTransitioning>)`
- The system will call these objects with a system created object
  - `<UIViewControllerContextTransitioning>`
    - This object defines the transition in many important ways

# New UIViewController Transition APIs

## A quick review

- Your applications vends objects that conform to two protocols
  - `<UIViewControllerAnimatedTransitioning>`
    - `(void)animateTransition:(id <UIViewControllerContextTransitioning>)`
  - `<UIViewControllerInteractiveTransitioning>`
    - `(void)startInteractiveTransition: (id <UIViewControllerContextTransitioning>)`
- The system will call these objects with a system created object
  - `<UIViewControllerContextTransitioning>`
    - This object defines the transition in many important ways

# UIViewController Transition APIs

## <UIViewControllerContextTransitioning>



```
@protocol UIViewControllerContextTransitioning <NSObject>
```

```
// The view in which the animated transition should take place.
```

```
- (UIView *)containerView;
```

```
- (UIViewController *) viewControllerForKey:(NSString *)key;
```

```
- (CGRect) initialFrameForViewController:(UIViewController *)vc;
```

```
- (CGRect) finalFrameForViewController:(UIViewController *)vc;
```

```
...
```

```
@end
```



# UIViewController Transition APIs

## <UIViewControllerContextTransitioning>



```
@protocol UIViewControllerContextTransitioning <NSObject>
```

```
// The view in which the animated transition should take place.
```

```
- (UIView *)containerView;
```

```
- (UIViewController *) viewControllerForKey:(NSString *)key;
```

```
- (CGRect) initialFrameForViewController:(UIViewController *)vc;
```

```
- (CGRect) finalFrameForViewController:(UIViewController *)vc;
```

```
...
```

```
@end
```

# UIViewController Transition APIs

## <UIViewControllerContextTransitioning>



```
@protocol UIViewControllerContextTransitioning <NSObject>
```

```
// The view in which the animated transition should take place.
```

```
- (UIView *)containerView;
```

```
- (UIViewController *) viewControllerForKey:(NSString *)key;
```

```
- (CGRect) initialFrameForViewController:(UIViewController *)vc;
```

```
- (CGRect) finalFrameForViewController:(UIViewController *)vc;
```

```
...
```

```
@end
```

# UIViewController Transition APIs

## <UIViewControllerContextTransitioning>



```
@protocol UIViewControllerContextTransitioning <NSObject>
```

```
...
```

- (void) **updateInteractiveTransition:**(CGFloat)percent;
- (void) **finishInteractiveTransition:**
- (void) **cancelInteractiveTransition:**

```
// This MUST be called whenever a transition completes (or is cancelled.)
```

- (void)**completeTransition:**(BOOL)didComplete;

```
@end
```

# UIViewController Transition APIs

## <UIViewControllerContextTransitioning>



```
@protocol UIViewControllerContextTransitioning <NSObject>
```

```
...
```

```
- (void) updateInteractiveTransition:(CGFloat)percent;  
- (void) finishInteractiveTransition:  
- (void) cancelInteractiveTransition:
```

```
// This MUST be called whenever a transition completes (or is cancelled.)
```

```
- (void)completeTransition:(BOOL)didComplete;
```

```
@end
```

# UIViewController Transition APIs

## <UIViewControllerContextTransitioning>



```
@protocol UIViewControllerContextTransitioning <NSObject>
```

```
...
```

- (void) **updateInteractiveTransition:**(CGFloat)percent;
- (void) **finishInteractiveTransition:**
- (void) **cancelInteractiveTransition:**

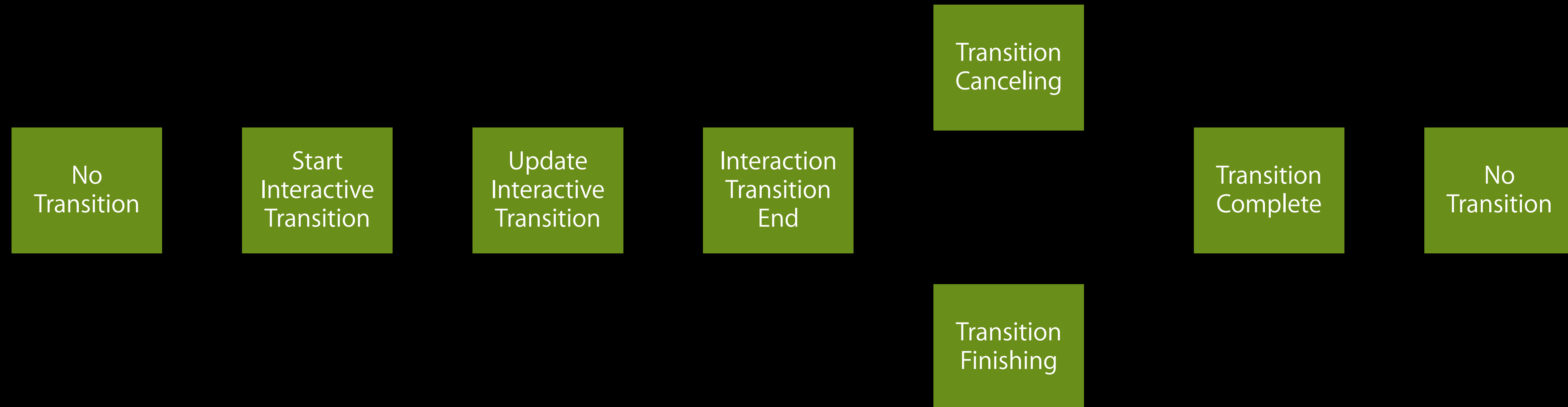
```
// This MUST be called whenever a transition completes (or is cancelled.)
```

```
– (void)completeTransition:(BOOL)didComplete;
```

```
@end
```

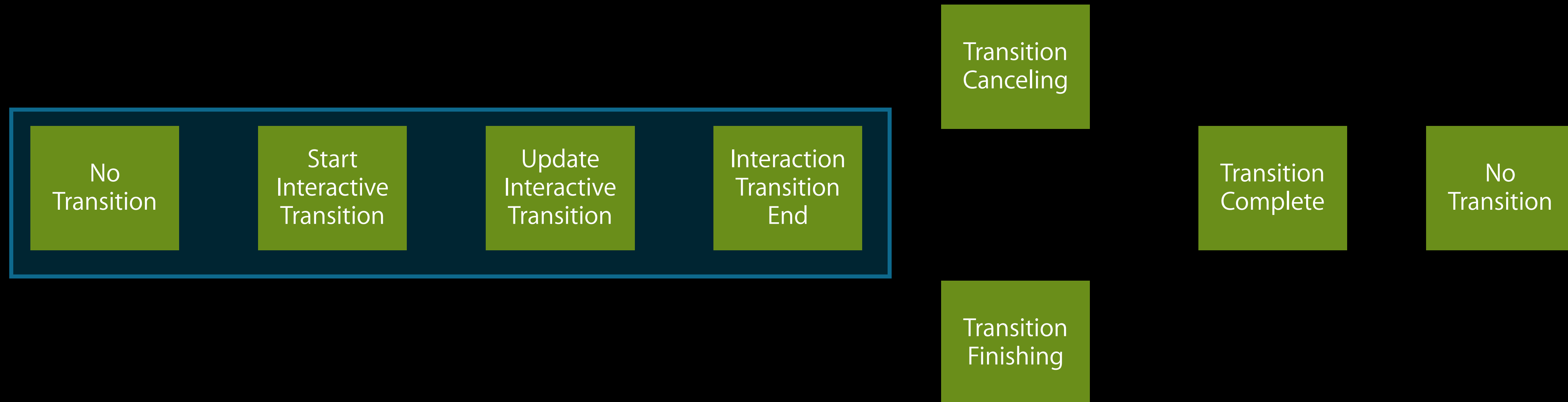
# UIViewController Transition APIs

## Interaction transition states



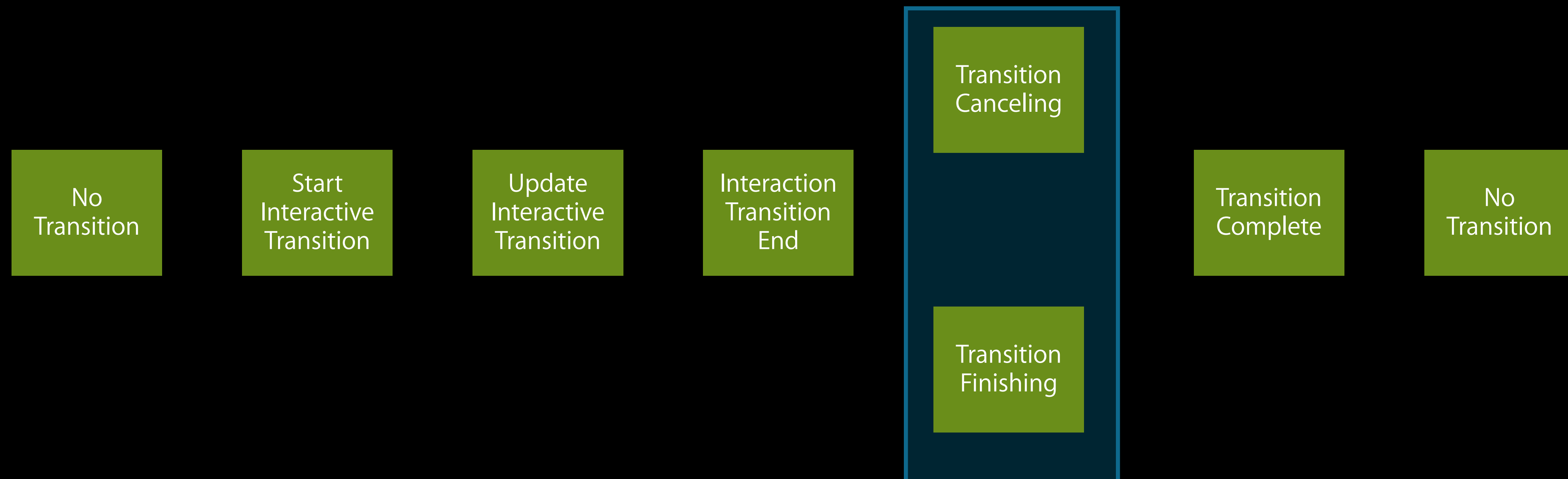
# UIViewController Transition APIs

## Interaction transition states



# UIViewController Transition APIs

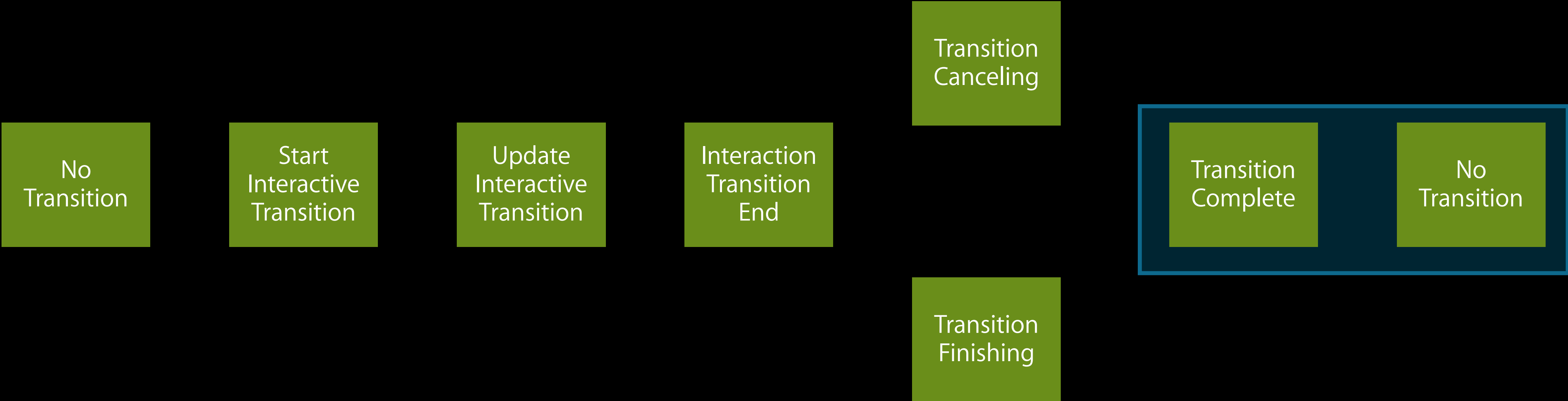
## Interaction transition states





# UIViewController Transition APIs

## Interaction transition states



# UIKit Dynamic Transitions

Two examples

# UIKit Dynamic Transitions

## Two examples

- A drop in and out dialog
  - Not interactive
  - A two stage dynamics simulation

# UIKit Dynamic Transitions

## Two examples

- A drop in and out dialog
  - Not interactive
  - A two stage dynamics simulation
- A drop shade transition
  - Is interactive
  - Can be used for navigation and present dismiss transitions

# UIKit Dynamic Transitions

A drop in and out dialog

# UIKit Dynamic Transitions

A drop in and out dialog

- A custom non-interactive present or dismiss transition

# UIKit Dynamic Transitions

## A drop in and out dialog

- A custom non-interactive present or dismiss transition
- Implemented as a compound `UIDynamicBehavior` that conforms to `<UIViewControllerAnimatedTransitioning>`

# UIKit Dynamic Transitions

## A drop in and out dialog

- A custom non-interactive present or dismiss transition
- Implemented as a compound `UIDynamicBehavior` that conforms to `<UIViewControllerAnimatedTransitioning>`
- Demonstrates



# UIKit Dynamic Transitions

## A drop in and out dialog

- A custom non-interactive present or dismiss transition
- Implemented as a compound `UIDynamicBehavior` that conforms to `<UIViewControllerAnimatedTransitioning>`
- Demonstrates
  - `UIDynamicBehavior`'s `action block` property

# UIKit Dynamic Transitions

## A drop in and out dialog

- A custom non-interactive present or dismiss transition
- Implemented as a compound `UIDynamicBehavior` that conforms to `<UIViewControllerAnimatedTransitioning>`
- Demonstrates
  - `UIDynamicBehavior`'s `action block` property
  - `UICollisionBehavior` and `UICollisionBehaviorDelegate`

# UIKit Dynamic Transitions

## A drop in and out dialog

- A custom non-interactive present or dismiss transition
- Implemented as a compound `UIDynamicBehavior` that conforms to `<UIViewControllerAnimatedTransitioning>`
- Demonstrates
  - `UIDynamicBehavior`'s `action` block property
  - `UICollisionBehavior` and `UICollisionBehaviorDelegate`
  - `– (void)[id <UIDynamicAnimatorDelegate> dynamicAnimatorDidPause:]`

# UIKit Dynamic Transitions

## A drop in and out dialog

- A custom non-interactive present or dismiss transition
- Implemented as a compound `UIDynamicBehavior` that conforms to `<UIViewControllerAnimatedTransitioning>`
- Demonstrates
  - `UIDynamicBehavior`'s `action` block property
  - `UICollisionBehavior` and `UICollisionBehaviorDelegate`
  - `– (void)[id <UIDynamicAnimatorDelegate> dynamicAnimatorDidPause:]`
    - `– (NSTimeInterval)[UIDynamicAnimator's elapsedTime]`

*Demo*

Drop in and out dialog

# Drop In and Out Dialog

Deconstruction

# UIKit Dynamic Transitions

## Drop dialog–YYDropOutAnimator

```
@interface YYDropOutAnimator : UIDynamicBehavior <UIViewControllerAnimatedTransitioning,  
  UIDynamicAnimatorDelegate,  
  UICollisionBehaviorDelegate>  
  
@property (nonatomic,assign) NSTimeInterval duration;  
@property (nonatomic, strong) id <UIViewControllerContextTransitioning> transitionContext;  
  
@property (nonatomic,assign,getter = isAppearing) BOOL appearing;  
  
@property (nonatomic,assign) NSTimeInterval finishTime;  
  
@property (nonatomic, strong) UIDynamicAnimator *dynamicAnimator;  
@property (nonatomic, strong) UIDynamicItemBehavior *bodyBehavior;  
@property (nonatomic, strong) UICollisionBehavior *collisionBehavior;  
@property (nonatomic, strong) UIGravityBehavior *gravityBehavior;  
@property (nonatomic, strong) UIAttachmentBehavior *attachBehavior;  
  
@end
```

# UIKit Dynamic Transitions

## Drop dialog–YYDropOutAnimator

```
@interface YYDropOutAnimator : UIDynamicBehavior <UIViewControllerAnimatedTransitioning,  
  UIDynamicAnimatorDelegate,  
  UICollisionBehaviorDelegate>  
  
@property (nonatomic,assign) NSTimeInterval duration;  
@property (nonatomic, strong) id <UIViewControllerContextTransitioning> transitionContext;  
  
@property (nonatomic,assign,getter = isAppearing) BOOL appearing;  
  
@property (nonatomic,assign) NSTimeInterval finishTime;  
  
@property (nonatomic, strong) UIDynamicAnimator *dynamicAnimator;  
@property (nonatomic, strong) UIDynamicItemBehavior *bodyBehavior;  
@property (nonatomic, strong) UICollisionBehavior *collisionBehavior;  
@property (nonatomic, strong) UIGravityBehavior *gravityBehavior;  
@property (nonatomic, strong) UIAttachmentBehavior *attachBehavior;  
  
@end
```



# UIKit Dynamic Transitions

## Drop dialog–YYDropOutAnimator

```
@interface YYDropOutAnimator : UIDynamicBehavior <UIViewControllerAnimatedTransitioning,  
  UIDynamicAnimatorDelegate,  
  UICollisionBehaviorDelegate>
```

```
@property (nonatomic,assign) NSTimeInterval duration;  
@property (nonatomic, strong) id <UIViewControllerContextTransitioning> transitionContext;
```

```
@property (nonatomic,assign,getter = isAppearing) BOOL appearing;
```

```
@property (nonatomic,assign) NSTimeInterval finishTime;
```

```
@property (nonatomic, strong) UIDynamicAnimator *dynamicAnimator;  
@property (nonatomic, strong) UIDynamicItemBehavior *bodyBehavior;  
@property (nonatomic, strong) UICollisionBehavior *collisionBehavior;  
@property (nonatomic, strong) UIGravityBehavior *gravityBehavior;  
@property (nonatomic, strong) UIAttachmentBehavior *attachBehavior;
```

```
@end
```

# UIKit Dynamic Transitions

## Drop dialog–YYDropOutAnimator

```
@interface YYDropOutAnimator : UIDynamicBehavior <UIViewControllerAnimatedTransitioning,  
  UIDynamicAnimatorDelegate,  
  UICollisionBehaviorDelegate>  
  
@property (nonatomic,assign) NSTimeInterval duration;  
@property (nonatomic, strong) id <UIViewControllerContextTransitioning> transitionContext;  
  
@property (nonatomic,assign,getter = isAppearing) BOOL appearing;  
  
@property (nonatomic,assign) NSTimeInterval finishTime;  
  
@property (nonatomic, strong) UIDynamicAnimator *dynamicAnimator;  
@property (nonatomic, strong) UIDynamicItemBehavior *bodyBehavior;  
@property (nonatomic, strong) UICollisionBehavior *collisionBehavior;  
@property (nonatomic, strong) UIGravityBehavior *gravityBehavior;  
@property (nonatomic, strong) UIAttachmentBehavior *attachBehavior;  
  
@end
```

# UIKit Dynamic Transitions

## Drop dialog–YYDropOutAnimator

```
@interface YYDropOutAnimator : UIDynamicBehavior <UIViewControllerAnimatedTransitioning,  
  UIDynamicAnimatorDelegate,  
  UICollisionBehaviorDelegate>  
  
@property (nonatomic,assign) NSTimeInterval duration;  
@property (nonatomic, strong) id <UIViewControllerContextTransitioning> transitionContext;  
  
@property (nonatomic,assign,getter = isAppearing) BOOL appearing;  
  
@property (nonatomic,assign) NSTimeInterval finishTime;  
  
@property (nonatomic, strong) UIDynamicAnimator *dynamicAnimator;  
@property (nonatomic, strong) UIDynamicItemBehavior *bodyBehavior;  
@property (nonatomic, strong) UICollisionBehavior *collisionBehavior;  
@property (nonatomic, strong) UIGravityBehavior *gravityBehavior;  
@property (nonatomic, strong) UIAttachmentBehavior *attachBehavior;  
  
@end
```

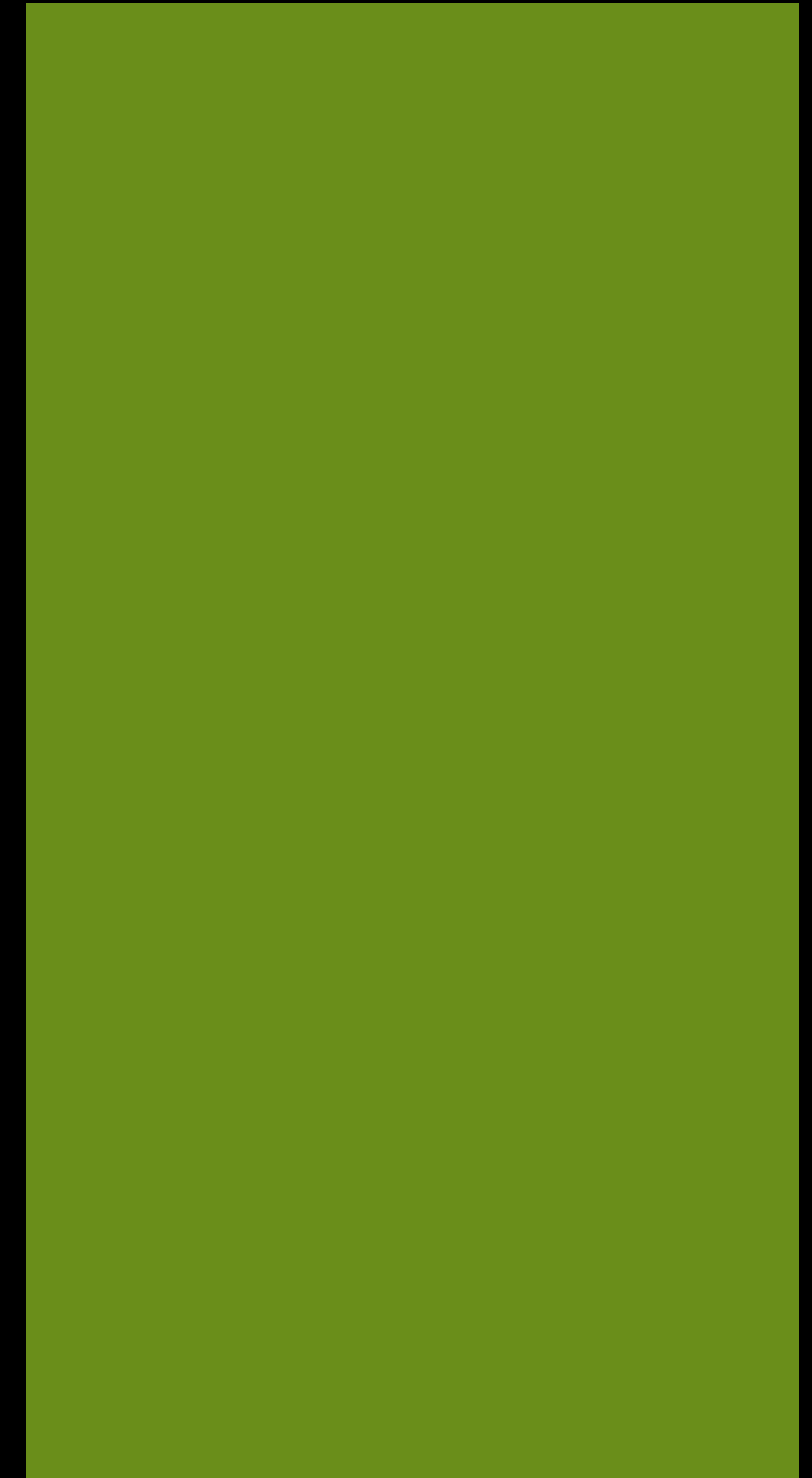
# UIKit Dynamic Transitions

## Drop dialog–YYDropOutAnimator

```
@interface YYDropOutAnimator : UIDynamicBehavior <UIViewControllerAnimatedTransitioning,  
  UIDynamicAnimatorDelegate,  
  UICollisionBehaviorDelegate>  
  
@property (nonatomic,assign) NSTimeInterval duration;  
@property (nonatomic, strong) id <UIViewControllerContextTransitioning> transitionContext;  
  
@property (nonatomic,assign,getter = isAppearing) BOOL appearing;  
  
@property (nonatomic,assign) NSTimeInterval finishTime;  
  
@property (nonatomic, strong) UIDynamicAnimator *dynamicAnimator;  
@property (nonatomic, strong) UIDynamicItemBehavior *bodyBehavior;  
@property (nonatomic, strong) UICollisionBehavior *collisionBehavior;  
@property (nonatomic, strong) UIGravityBehavior *gravityBehavior;  
@property (nonatomic, strong) UIAttachmentBehavior *attachBehavior;  
  
@end
```

# UIKit Dynamic Transitions

Drop dialog: Presenting

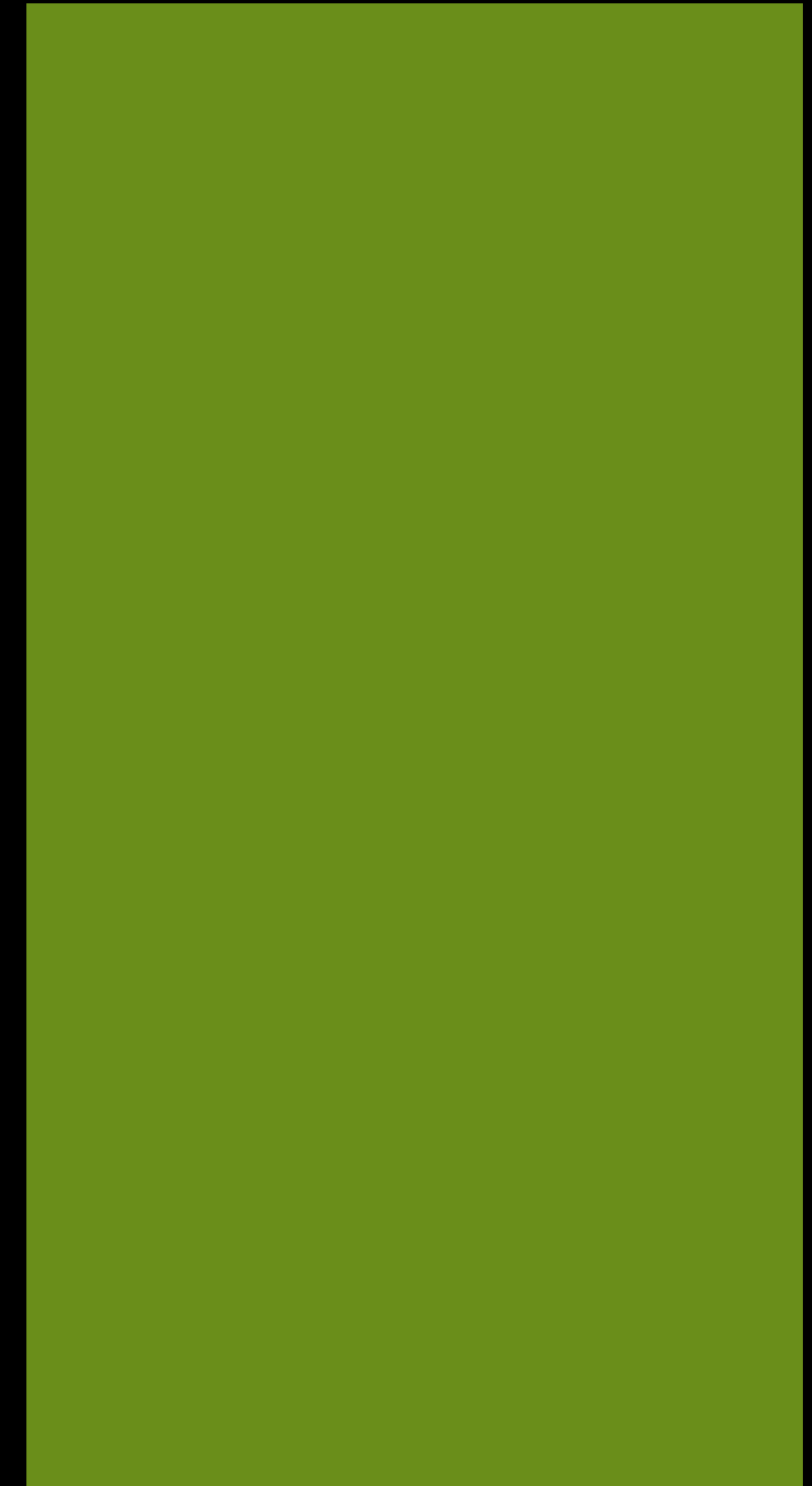


# UIKit Dynamic Transitions

## Drop dialog: Presenting

```
- (void) animateTransition:(id <UIViewControllerAnimatedTransitioning:>)context {
```

```
}
```



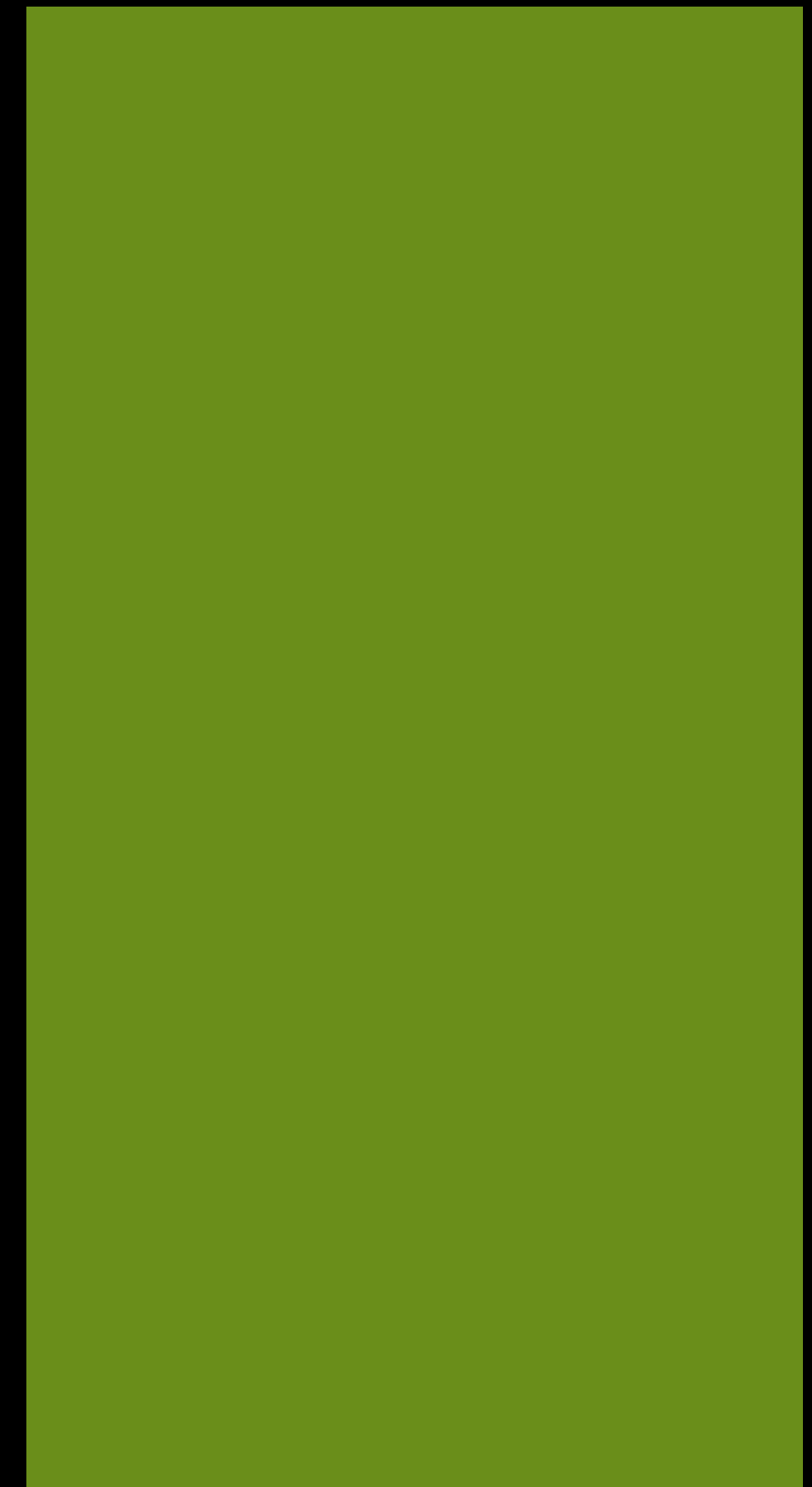
# UIKit Dynamic Transitions

## Drop dialog: Presenting

– (void) animateTransition: ... context {

```
UIView *inView = [context containerView];  
CGFloat height =InView.frame.size.height;  
NSTimeInterval duration = [self  
transitionDuration:context];  
UIView *dynamicView = ...
```

}



# UIKit Dynamic Transitions

## Drop dialog: Presenting

– (void) animateTransition: ... context {

```
UIView *inView = [context containerView];
CGFloat height =InView.frame.size.height;
NSTimeInterval duration = [self
transitionDuration:context];
UIView *dynamicView = ...
```

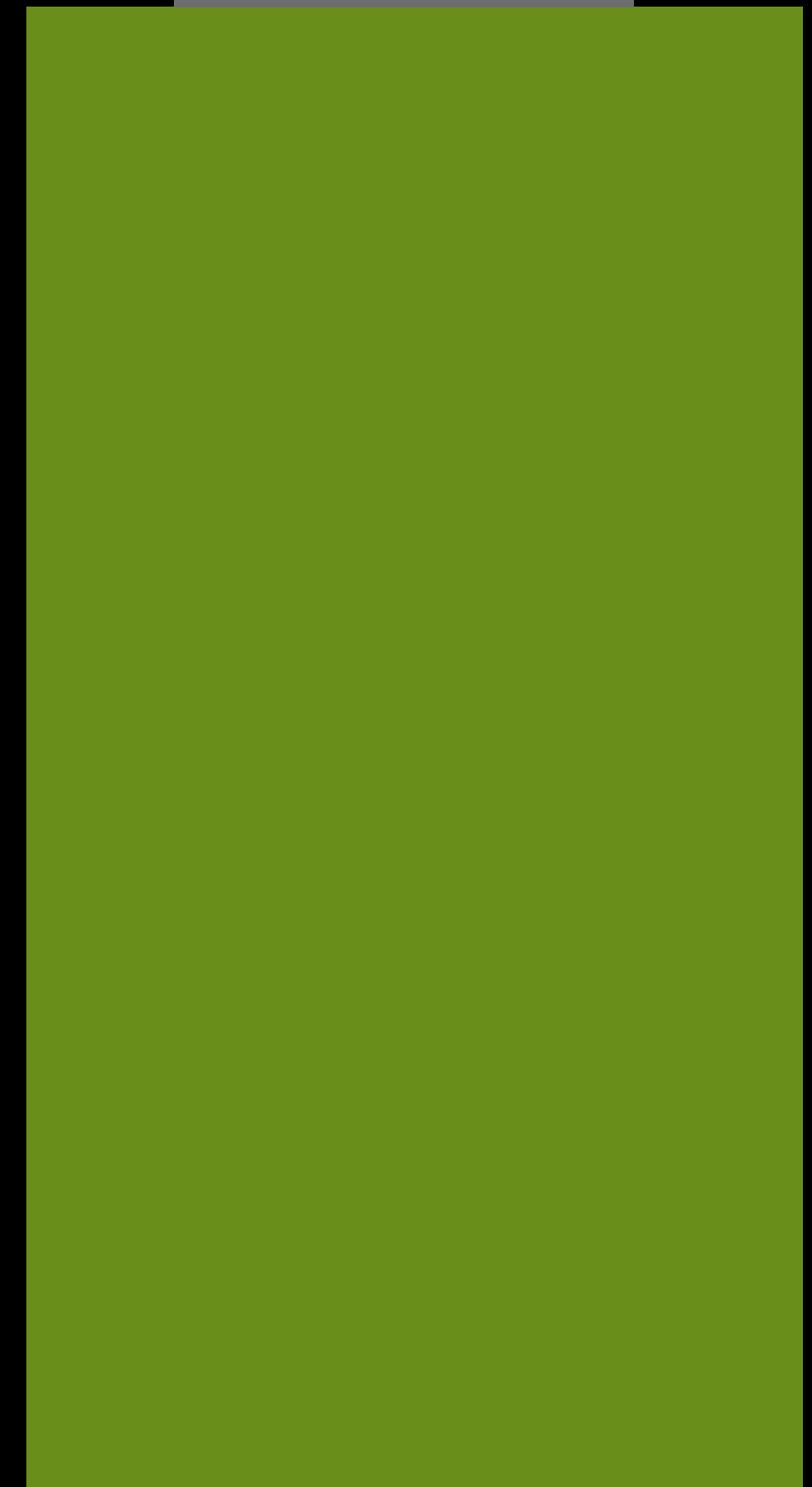
```
[inview addSubview: dynamicView];
```

```
bodyBehavior = [[UIDynamicItemBehavior alloc] init];
bodyBehavior.elasticity = .3;
[bodyBehavior addItem:dynamicView];
bodyBehavior.allowsRotation = NO;
```

```
}
```

Where ever you  
go there you are

Good to Know





# UIKit Dynamic Transitions

## Drop dialog: Presenting

– (void) animateTransition: ... context {

```
UIView *inView = [context containerView];  
CGFloat height =InView.frame.size.height;  
NSTimeInterval duration = [self  
transitionDuration:context];  
UIView *dynamicView = ...
```

```
NSArray items = @[dynamicView];  
gravityBehavior = [[UIGravityBehavior alloc]  
initWithItems:items];  
[gravityBehavior setXComponent:0.0 yComponent:3.0];
```

}

Where ever you  
go there you are

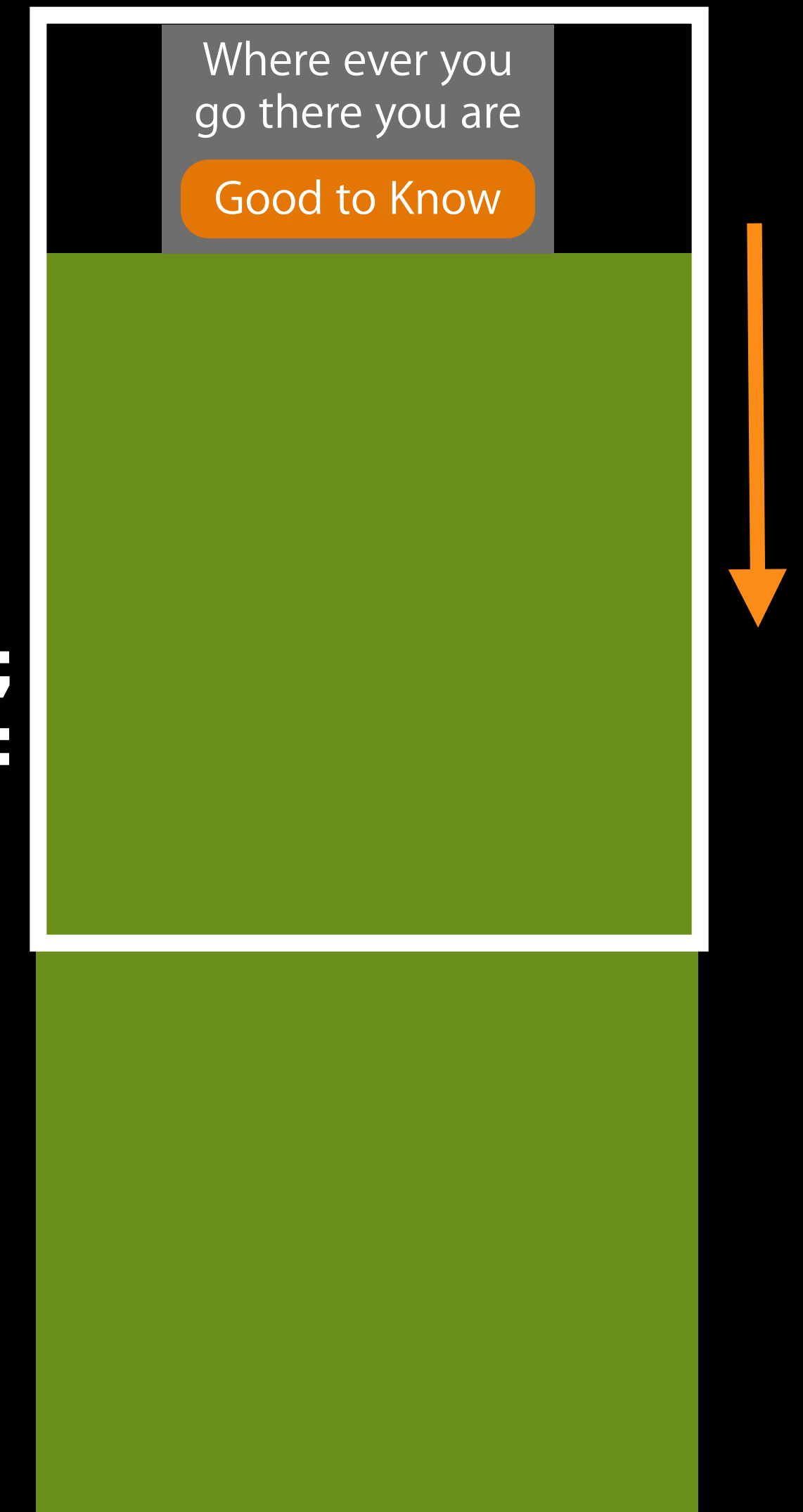
Good to Know



# UIKit Dynamic Transitions

## Drop dialog: Presenting

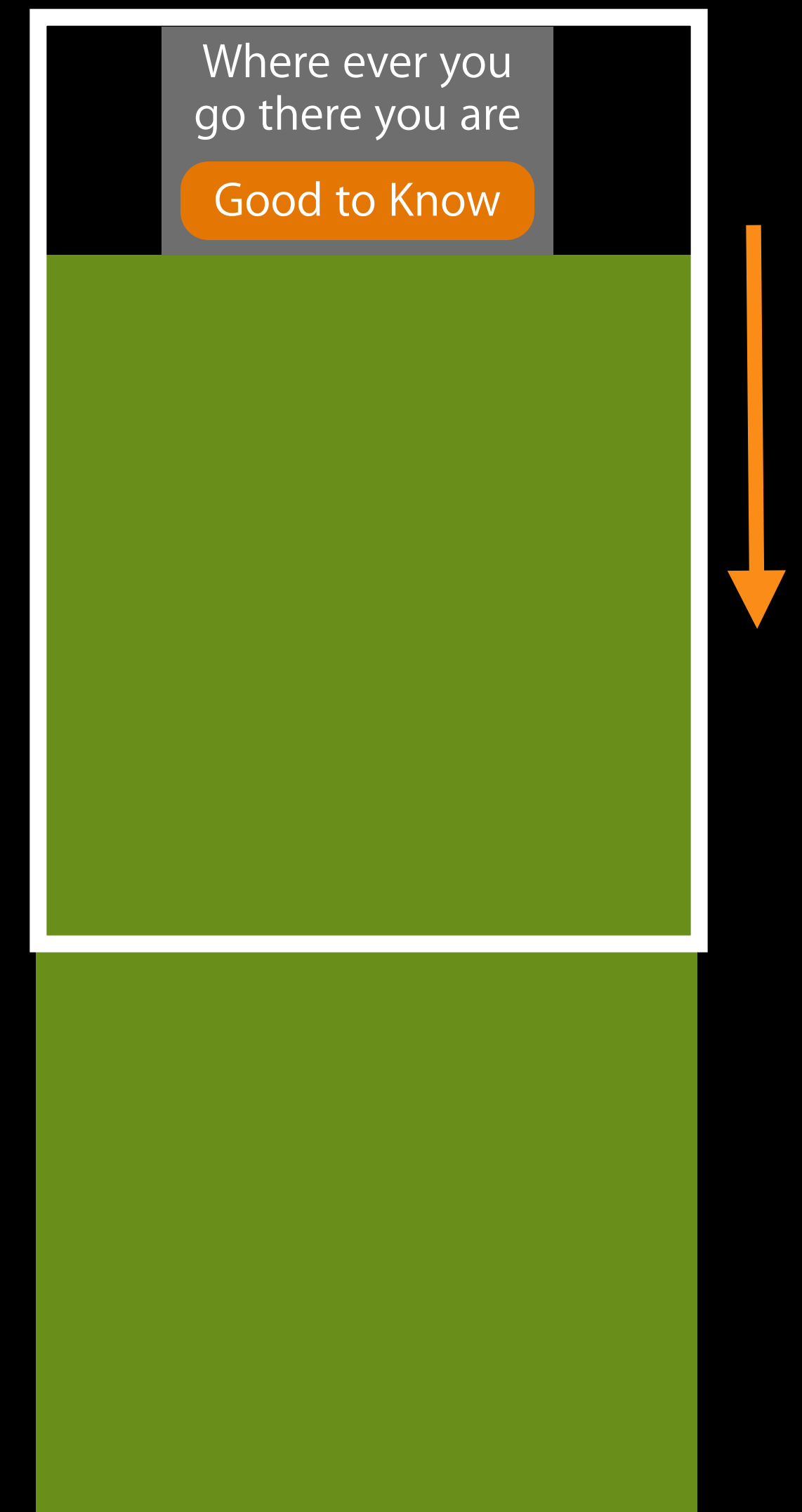
```
– (void) animateTransition: ... context {  
    UIView *inView = [context containerView];  
    CGFloat height =InView.frame.size.height;  
    NSTimeInterval duration = [self  
        transitionDuration:context];  
    UIView *dynamicView = ...  
  
    NSArray items = @[dynamicView];  
    cb = [[UICollisionBehavior alloc] initWithItems:items];  
    [cb setTranslatesReferenceBoundsIntoBoundaryWithInsets:  
  
    }  
}
```



# UIKit Dynamic Transitions

## Drop dialog: Presenting

```
- (void) animateTransition: ... context {  
    UIView *inView = [context containerView];  
    CGFloat height =InView.frame.size.height;  
    NSTimeInterval duration = [self  
        transitionDuration:context];  
    UIView *dynamicView = ...  
  
    self.finishTime = [self.dynamicAnimator elapsedTime]  
        + duration;  
    YYDropOutAnimator *weakSelf = self;  
    self.action = ^{  
        if([weakSelf.dynamicAnimator elapsedTime] >=  
            self.finishTime) {  
            [weakSelf.dynamicAnimator  
                removeBehavior:weakSelf];  
        }  
    };  
}
```



# UIKit Dynamic Transitions

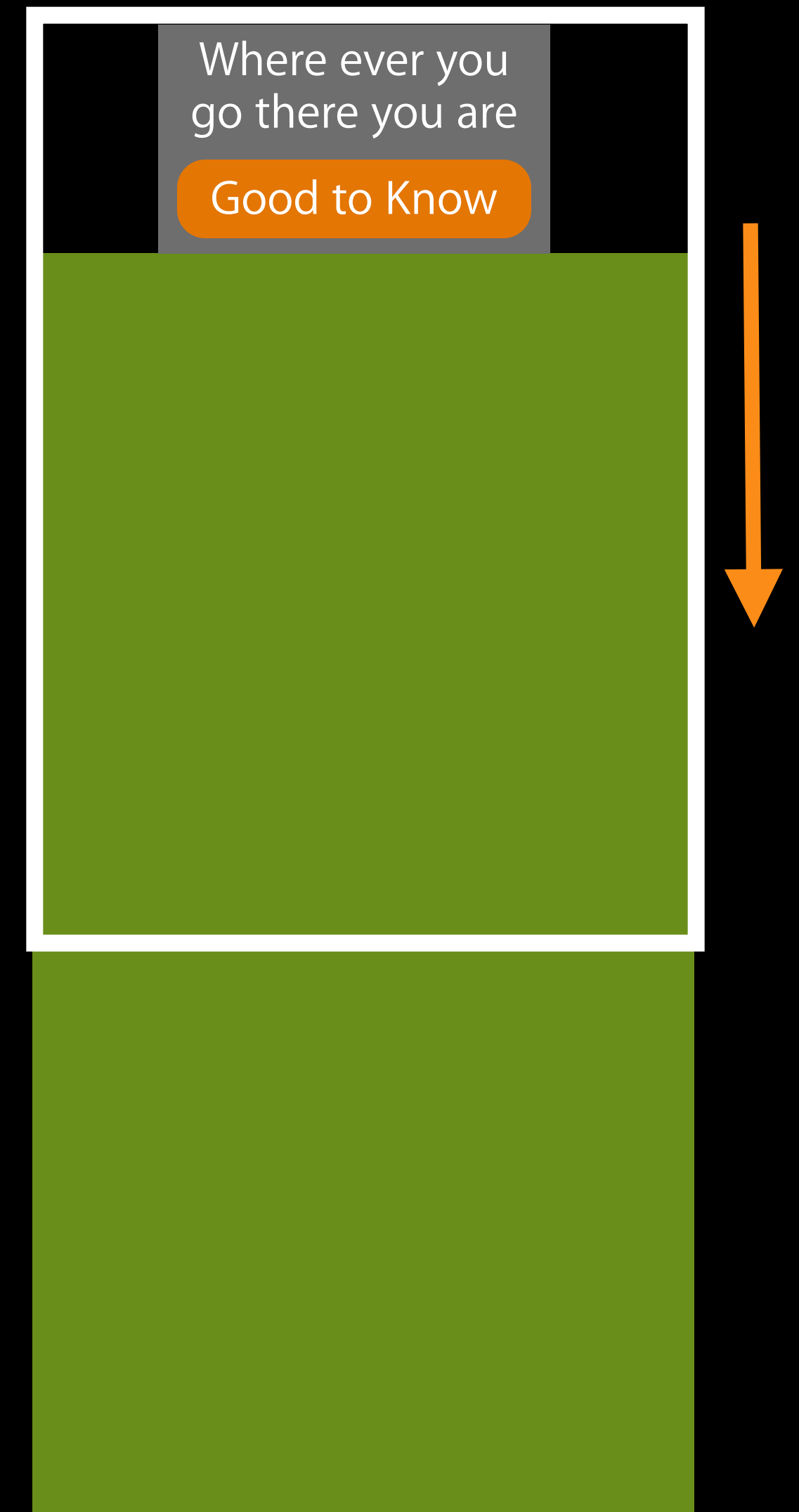
## Drop dialog: Presenting

– (void) animateTransition: ... context {

```
UIView *inView = [context containerView];  
CGFloat height =InView.frame.size.height;  
NSTimeInterval duration = [self  
transitionDuration:context];  
UIView *dynamicView = ...
```

```
[self addChildBehavior:self.collisionBehavior];  
[self addChildBehavior:self.bodyBehavior];  
[self addChildBehavior:self.gravityBehavior];  
[self.dynamicAnimator addBehavior:self];
```

}



# UIKit Dynamic Transitions

## Drop dialog: Presenting



# UIKit Dynamic Transitions

## Drop dialog: Dismissing

Where ever you  
go there you are

Good to Know

# UIKit Dynamic Transitions

## Drop dialog: Dismissing

– (void) animateTransition:... context {

```
UIView *inView = [context containerView];  
CGFloat height =InView.frame.size.height;  
NSTimeInterval duration = [self  
transitionDuration:context];  
UIView *dynamicView = ...
```

}

Where ever you  
go there you are

Good to Know

# UIKit Dynamic Transitions

## Drop dialog: Dismissing

```
– (void) animateTransition:... context {  
    UIView *inView = [context containerView];  
    CGFloat height =InView.frame.size.height;  
    NSTimeInterval duration = [self  
    transitionDuration:context];  
    UIView *dynamicView = ...  
  
    bodyBehavior = [[UIDynamicItemBehavior alloc] init];  
    bodyBehavior.elasticity = 0.8;  
    bodyBehavior.angularResistance = 5.0;  
    [bodyBehavior addItem:dynamicView];  
    bodyBehavior.allowsRotation = YES;  
  
}
```

Where ever you  
go there you are

Good to Know



# UIKit Dynamic Transitions

## Drop dialog: Dismissing

```
– (void) animateTransition:... context {  
    UIView *inView = [context containerView];  
    CGFloat height =InView.frame.size.height;  
    NSTimeInterval duration = [self  
    transitionDuration:context];  
    UIView *dynamicView = ...  
  
    NSArray items = @[dynamicView];  
    gravityBehavior = [[UIGravityBehavior alloc]  
                      initWithItems:items;  
    [gravityBehavior setXComponent:0.0 yComponent:3.0];  
  
}
```

Where ever you  
go there you are

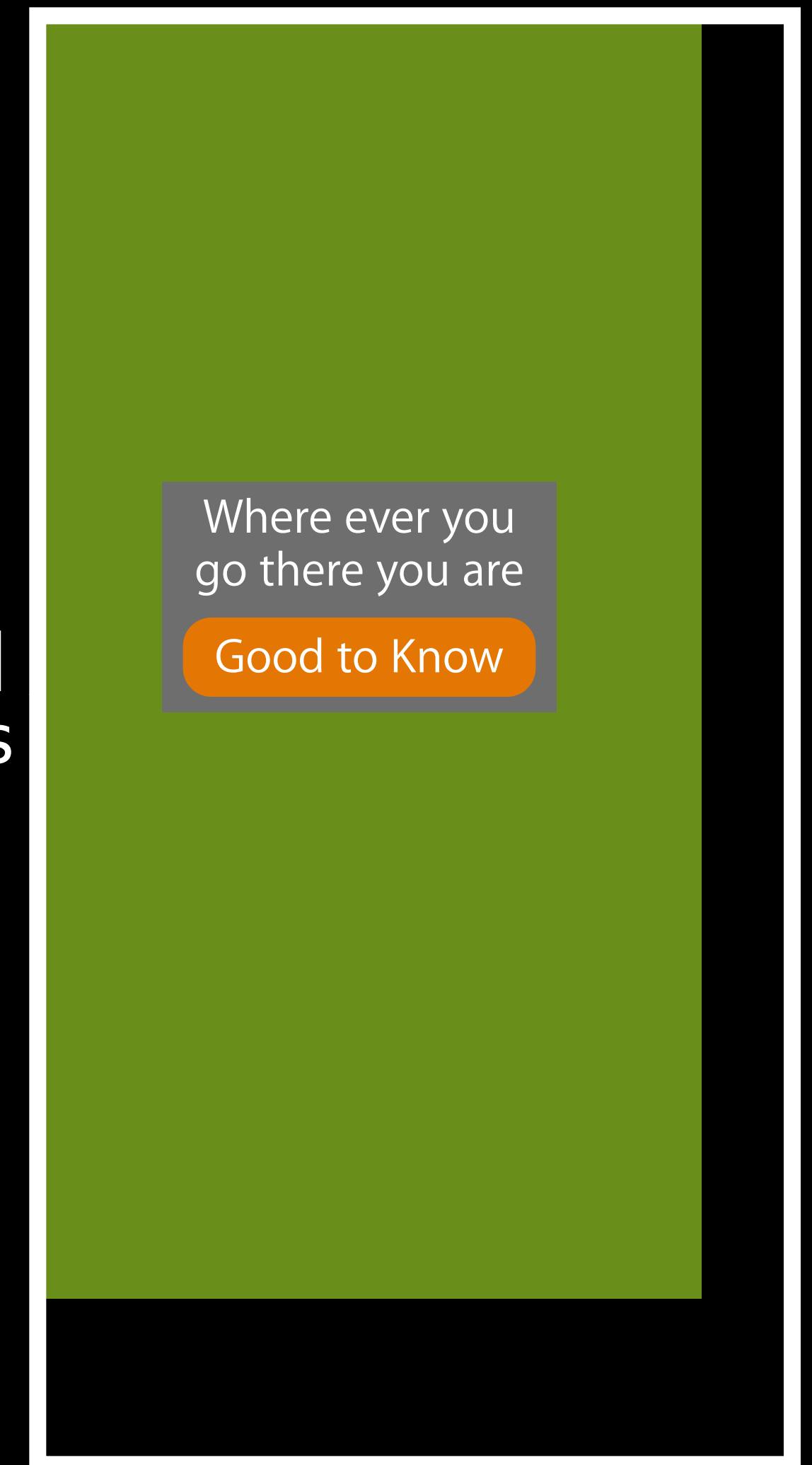
Good to Know



# UIKit Dynamic Transitions

## Drop dialog: Dismissing

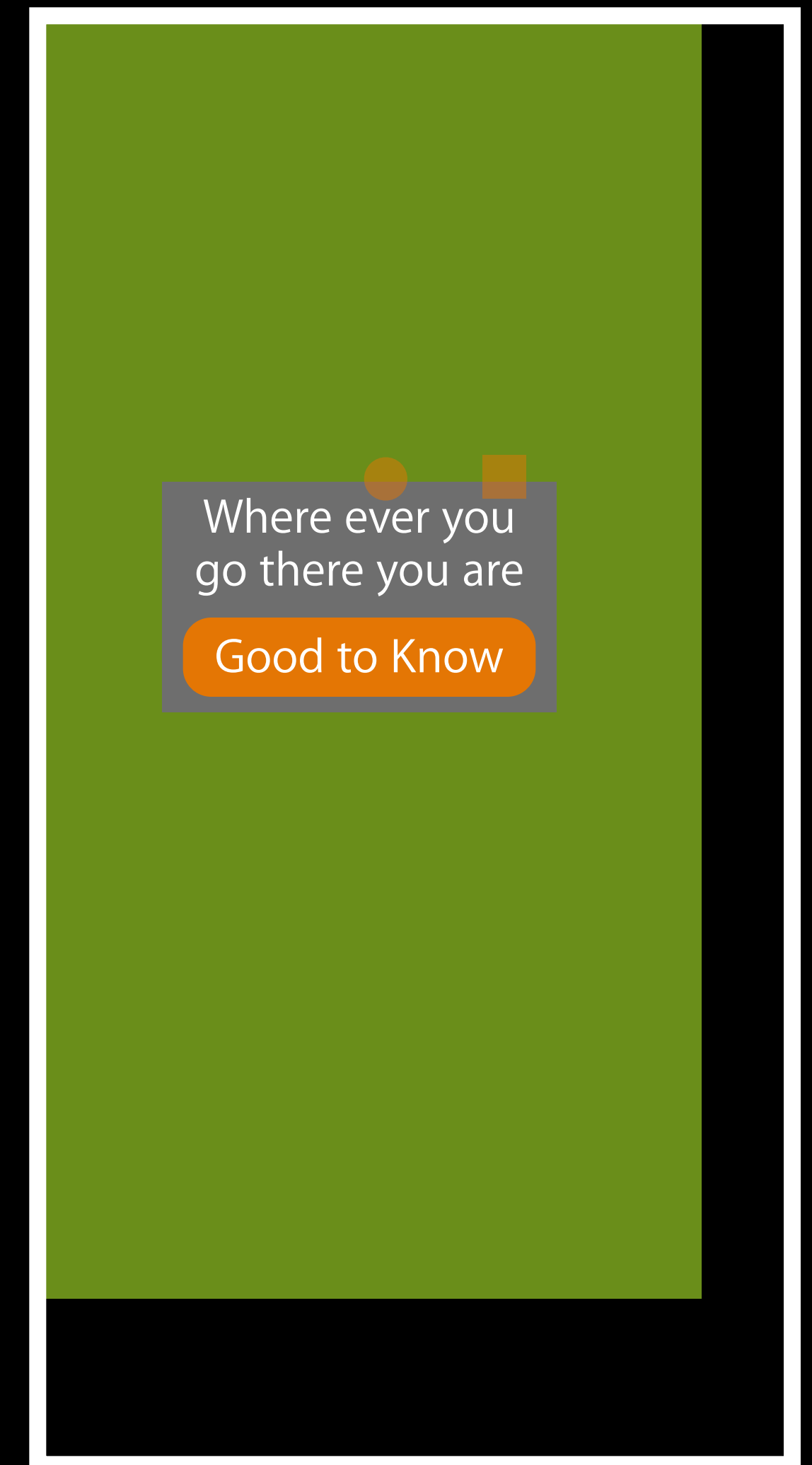
```
– (void) animateTransition:... context {  
    UIView *inView = [context containerView];  
    CGFloat height =InView.frame.size.height;  
    NSTimeInterval duration = [self  
    transitionDuration:context];  
    UIView *dynamicView = ...  
  
    NSArray items = @[dynamicView];  
    cb = [[UICollisionBehavior alloc] initWithItems:items]  
    [cb setTranslatesReferenceBoundsIntoBoundaryWithInsets  
    [cb setCollisionDelegate:self];  
  
}
```



# UIKit Dynamic Transitions

## Drop dialog: Dismissing

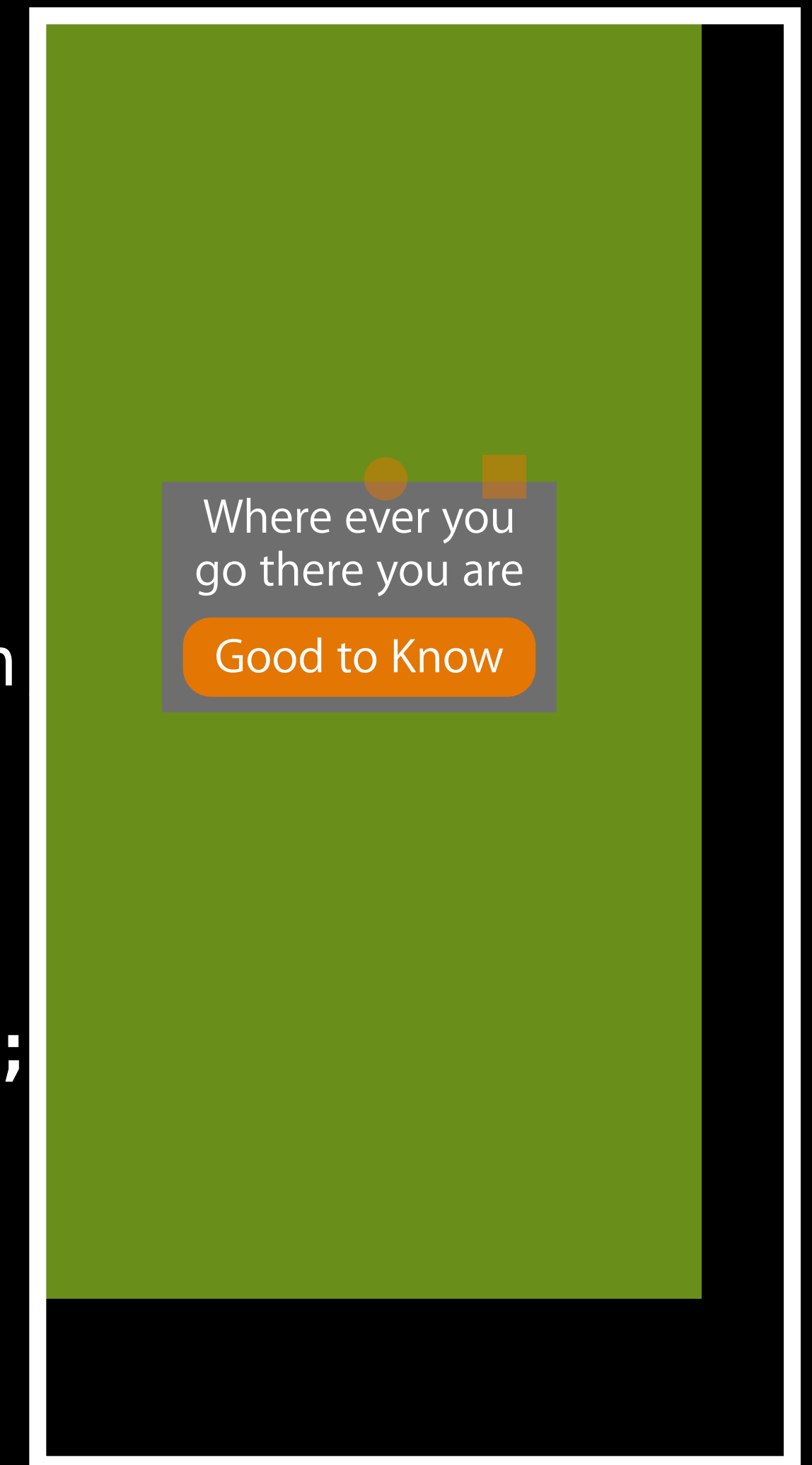
```
– (void) animateTransition:... context {  
    UIView *inView = [context containerView];  
    CGFloat height =InView.frame.size.height;  
    NSTimeInterval duration = [self  
    transitionDuration:context];  
    UIView *dynamicView = ...  
  
    self.attachBehavior = [[UIAttachmentBehavior alloc]  
                           initWithItem:dynamicView  
                           point:point  
                           attachedToAnchor:anchor];  
    [self.attachBehavior setFrequency: 2];  
    [self.attachBehavior setDamping:.8];  
  
}
```



# UIKit Dynamic Transitions

## Drop dialog: Dismissing

```
- (void) animateTransition:... context {  
    UIView *inView = [context containerView];  
    CGFloat height =InView.frame.size.height;  
    NSTimeInterval duration = [self  
    transitionDuration:context];  
    UIView *dynamicView = ...  
  
    self.finishTime = (2./3.) * [self.dynamicAnimator  
                                elapsedTime] + duration  
    YYDropOutAnimator *weakSelf = self;  
    self.action = ^{  
        if([weakSelf.dynamicAnimator elapsedTime] >=  
            self.finishTime) {  
            [weakSelf.dynamicAnimator removeBehavior:weakSelf];  
        }  
    };  
}
```



# UIKit Dynamic Transitions

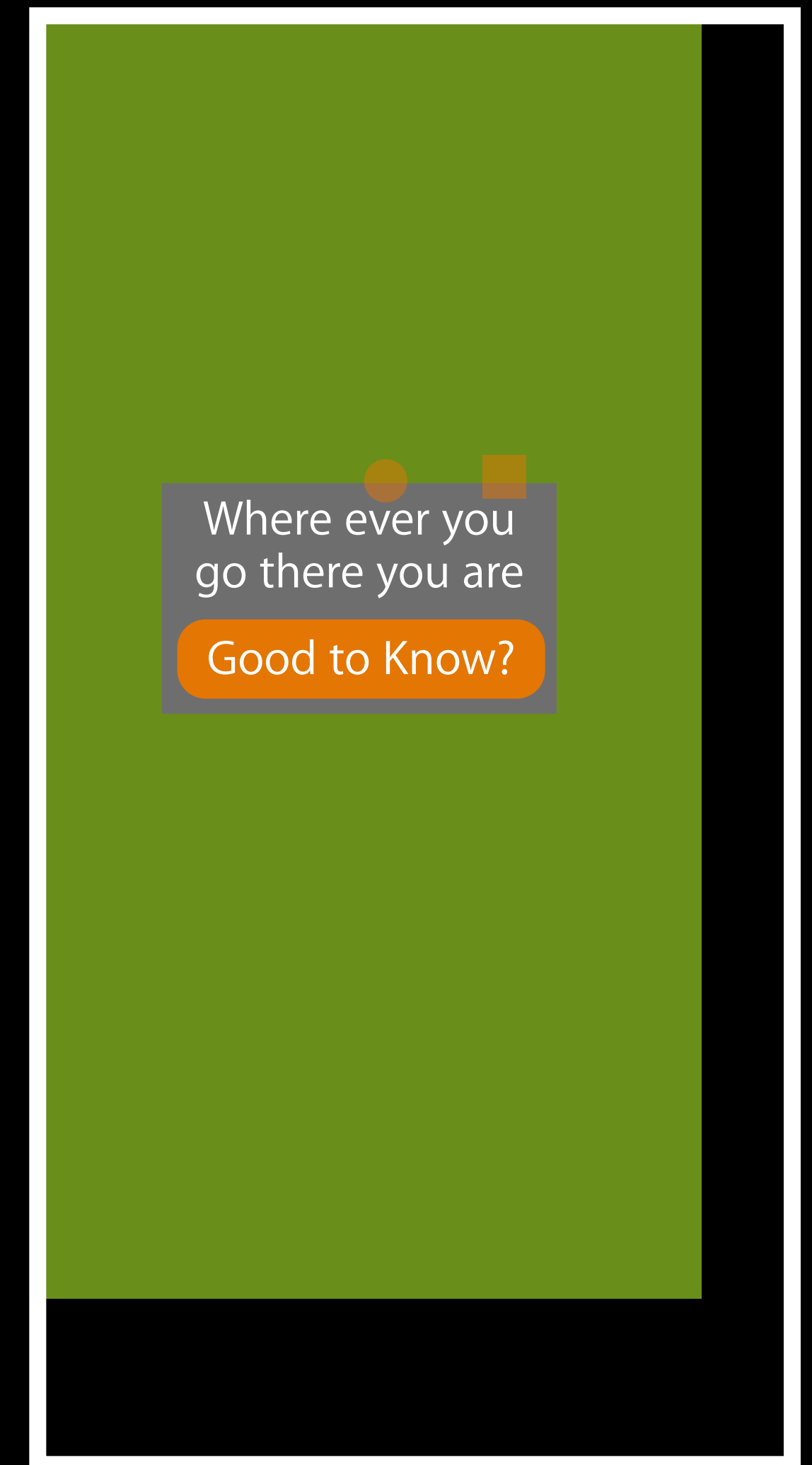
## Drop dialog: Dismissing

– (void) animateTransition:... context {

```
UIView *inView = [context containerView];  
CGFloat height =InView.frame.size.height;  
NSTimeInterval duration = [self  
transitionDuration:context];  
UIView *dynamicView = ...
```

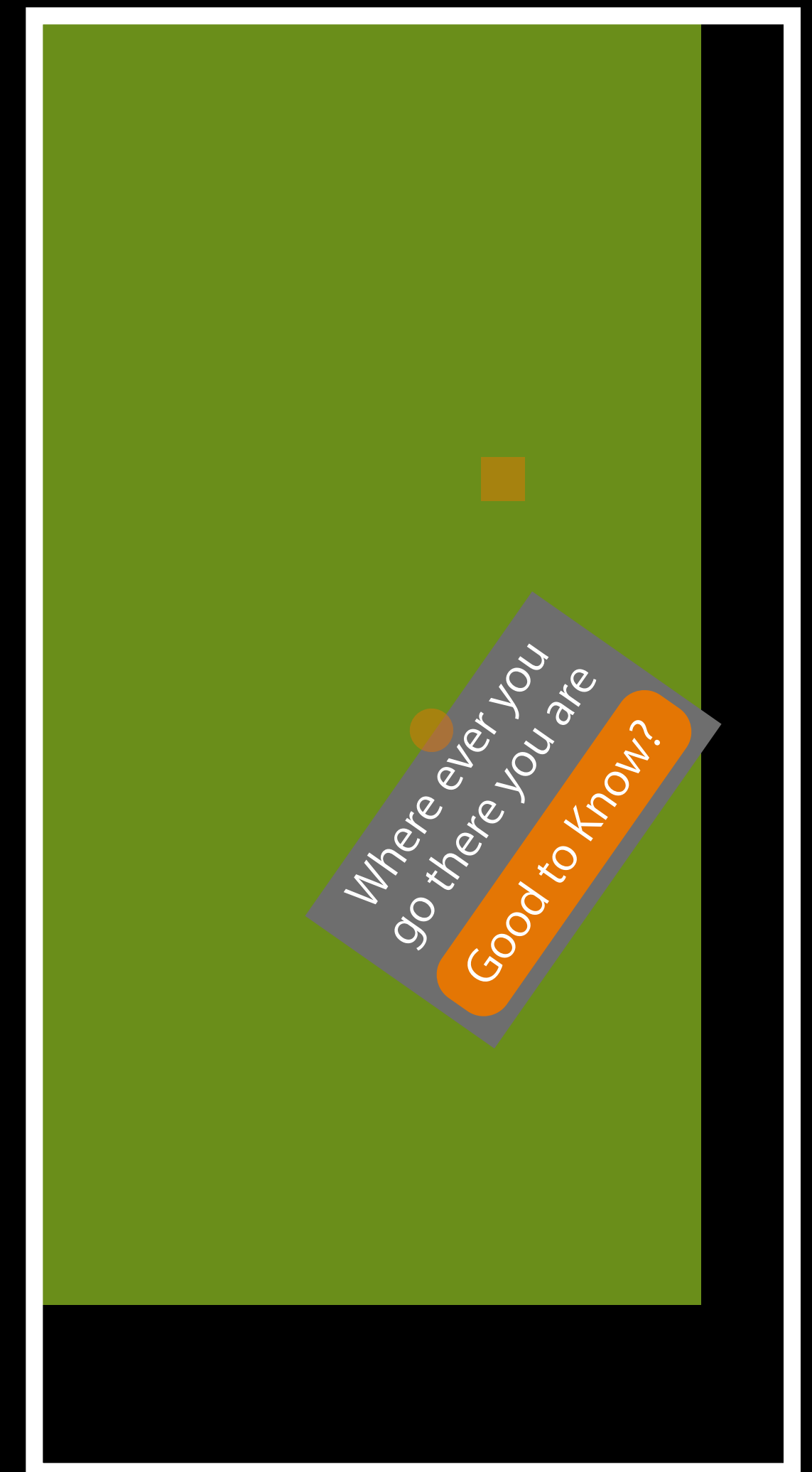
```
[self addChildBehavior:self.collisionBehavior];  
[self addChildBehavior:self.bodyBehavior];  
[self addChildBehavior:self.gravityBehavior];  
if(!self.isAppearing) {  
    [self addChildBehavior:self.attachBehavior];  
}  
[self.dynamicAnimator addBehavior:self];
```

}



# UIKit Dynamic Transitions

## Drop dialog: Dismissing

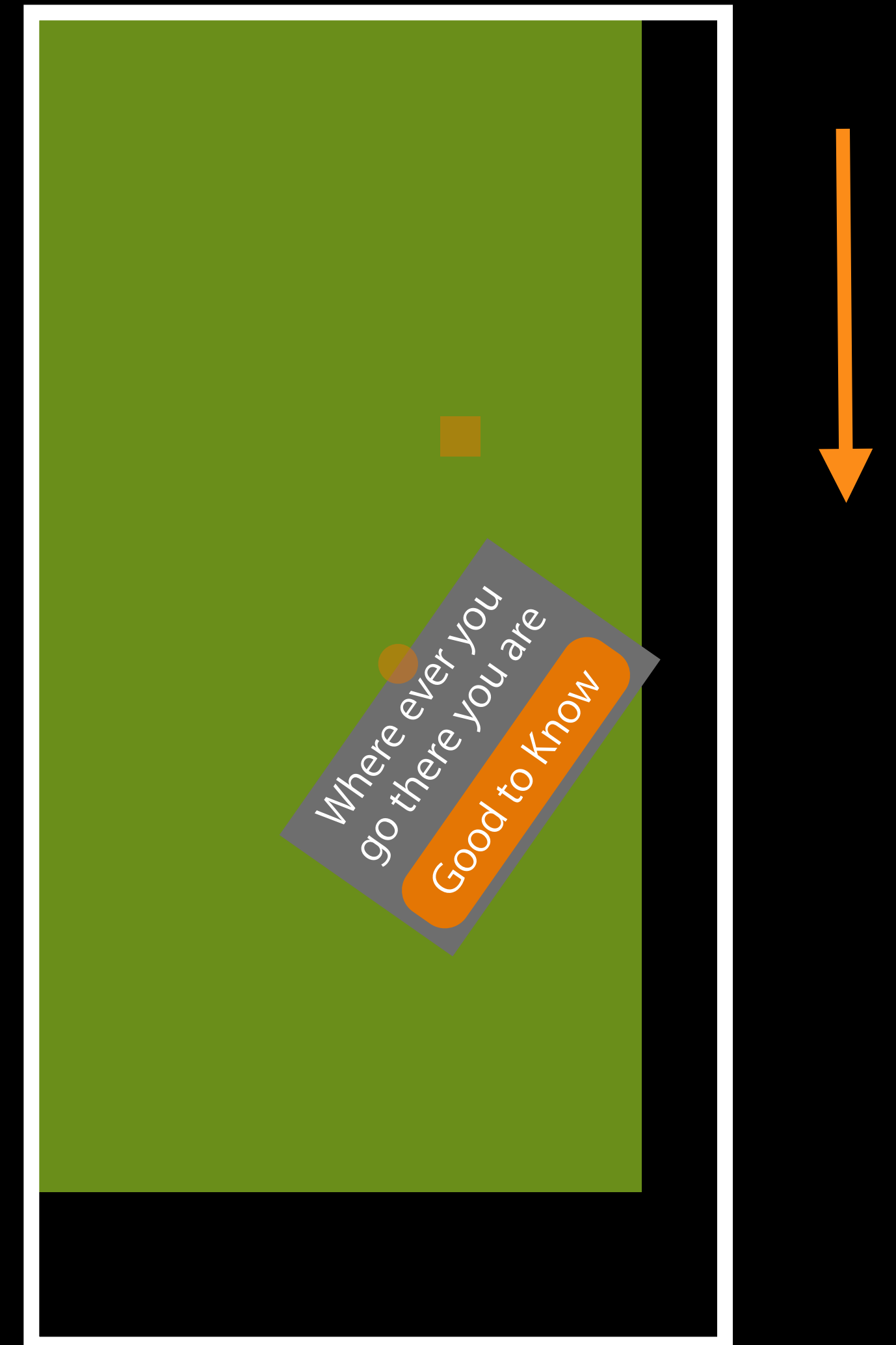


# UIKit Dynamic Transitions

## Drop dialog: Dismissing

```
– (void) dynamicAnimatorDidPause: ... {
```

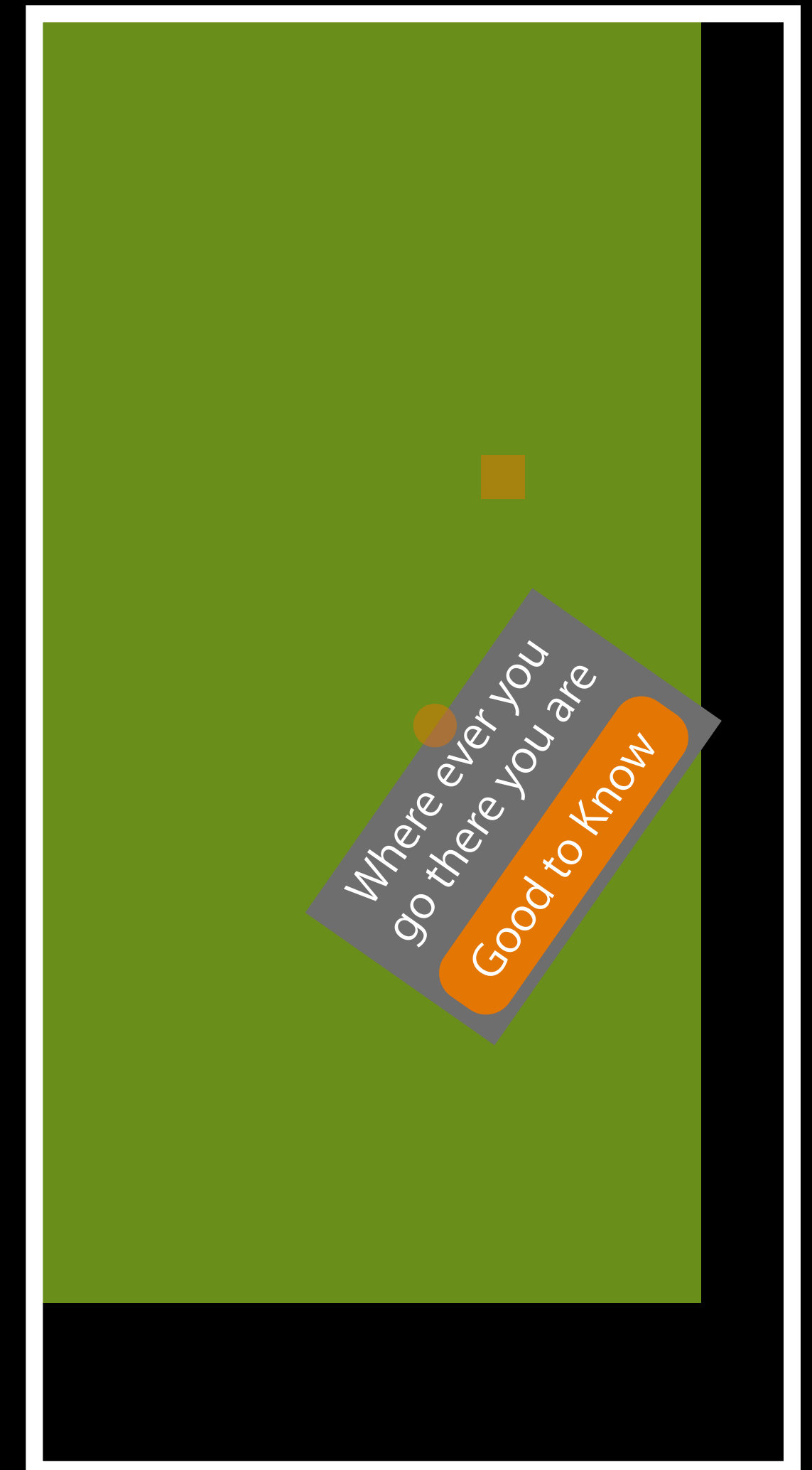
```
}
```



# UIKit Dynamic Transitions

## Drop dialog: Dismissing

```
- (void) dynamicAnimatorDidPause: ... {  
  
    if(self.attachBehavior) {  
        [self removeChildBehavior:self.attachBehavior];  
        self.attachBehavior = nil;  
        [self.dynamicAnimator addBehavior:self];  
        self.finishTime = 1./3. * self.duration +  
                           [animator elapsedTime];  
    }  
    else {  
        [self.transitionContext completeTransition: YES];  
        [self removeAllChildBehaviors];  
        [self.dynamicAnimator removeAllBehaviors];  
        self.transitionContext = nil;  
        ...  
    }  
}
```





# UIKit Dynamic Transitions

## Drop dialog: Dismissing

```
- (void) dynamicAnimatorDidPause: ... {  
  
    if(self.attachBehavior) {  
        [self removeChildBehavior:self.attachBehavior];  
        self.attachBehavior = nil;  
        [self.dynamicAnimator addBehavior:self];  
        self.finishTime = 1./3. * self.duration +  
                           [animator elapsedTime];  
    }  
    else {  
        [self.transitionContext completeTransition: YES];  
        [self removeAllChildBehaviors];  
        [self.dynamicAnimator removeAllBehaviors];  
        self.transitionContext = nil;  
        ...  
    }  
}
```



# UIKit Dynamic Transitions

## Drop dialog: Dismissing

```
– (void) (void)collisionBehavior: ... {
```

```
    CGFloat xContact = [[self.transitionContext  
                        containerView] bounds].size.width;  
    if(point.x < xContact) {  
        [self removeChildBehavior:behavior];  
        [self.dynamicAnimator removeBehavior:behavior];  
        self.collisionBehavior = nil;  
    }
```

```
}
```



# UIKit Dynamic Transitions

## Drop dialog: Dismissing

– (void) dynamicAnimatorDidPause: ... {

```
if(self.attachBehavior) {  
    [self removeChildBehavior:self.attachBehavior];  
    self.attachBehavior = nil;  
    [self.dynamicAnimator addBehavior:self];  
    self.finishTime = 1./3. * self.duration +  
                      [animator elapsedTime];  
}
```

```
else {
```

```
[self.transitionContext completeTransition: YES];  
[self removeAllChildBehaviors];  
[self.dynamicAnimator removeAllBehaviors];  
self.transitionContext = nil;  
...
```

```
}
```

```
}
```

# UIKit Dynamic Transitions

A drop shade transition

# UIKit Dynamic Transitions

A drop shade transition

- Implements an “interactive” transition
  - Can be used in navigation and modal transitions

# UIKit Dynamic Transitions

## A drop shade transition

- Implements an “interactive” transition
  - Can be used in navigation and modal transitions
- Implemented as a compound UIDynamicBehavior that conforms to `<UIViewControllerAnimatedTransitioning>` and `<UIViewControllerInteractiveTransitioning>`

# UIKit Dynamic Transitions

## A drop shade transition

- Implements an “interactive” transition
  - Can be used in navigation and modal transitions
- Implemented as a compound UIDynamicBehavior that conforms to `<UIViewControllerAnimatedTransitioning>` and `<UIViewControllerInteractiveTransitioning>`
- Demonstrates
  - The “interactive” portion of the transition does not use UIKit Dynamics
  - Use of UIDynamicBehavior’s action block property to drive an interactive transition
  - Use of `dynamicAnimatorDidPause:` to complete the transition

*Demo*

Drop shade transition

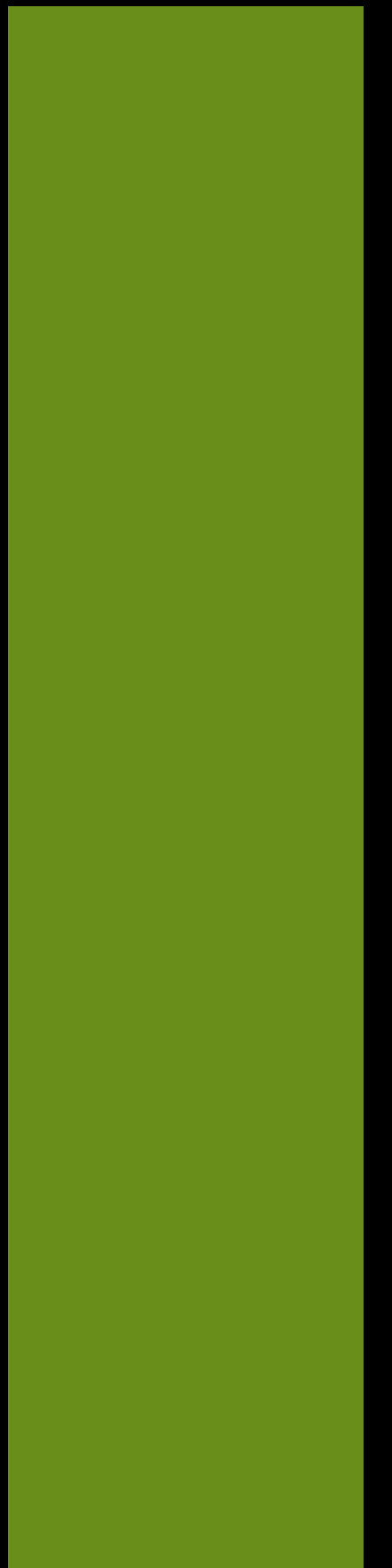


# A Drop Shade Transition

Deconstruction

# UIKit Dynamic Transitions

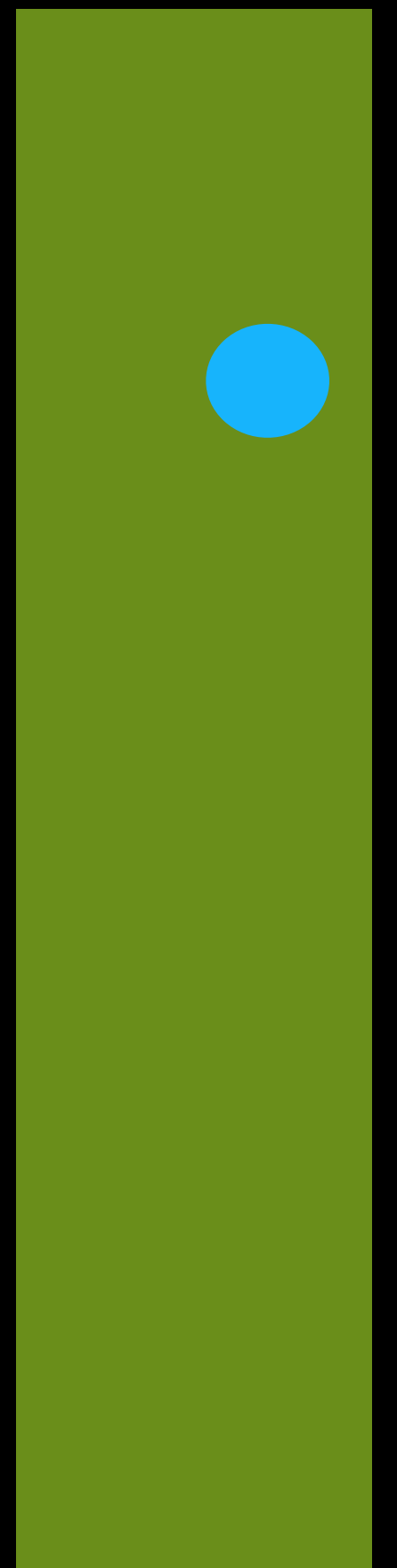
Drop shade transition



# UIKit Dynamic Transitions

## Drop shade transition

```
- (void) handleGesture:(UIPanGestureRecognizer *)gr {  
    UIViewController *fromVC = [self.transitionContext viewControllerForKey:...];  
    UIViewController *toVC = [self.transitionContext viewControllerForKey:...];  
    switch ([gr state]) {  
        case UIGestureRecognizerStateBegan: {  
            if(self.isAppearing) {  
                UIViewController *vc = [[YYImageVC alloc] initWithNibName:...];  
                [self.parent pushViewController:vc animated:YES];  
            }  
            else {  
                [self.parent popViewControllerAnimated:YES];  
            }  
        }  
    }  
}
```



# UIKit Dynamic Transitions

## Drop shade transition

```
- (void) startInteractiveTransition:(id <UIViewControllerContextTransitioning> ctx
{
    [self setupViewsForTransition:ctx];
}
```



# UIKit Dynamic Transitions

## Drop shade transition

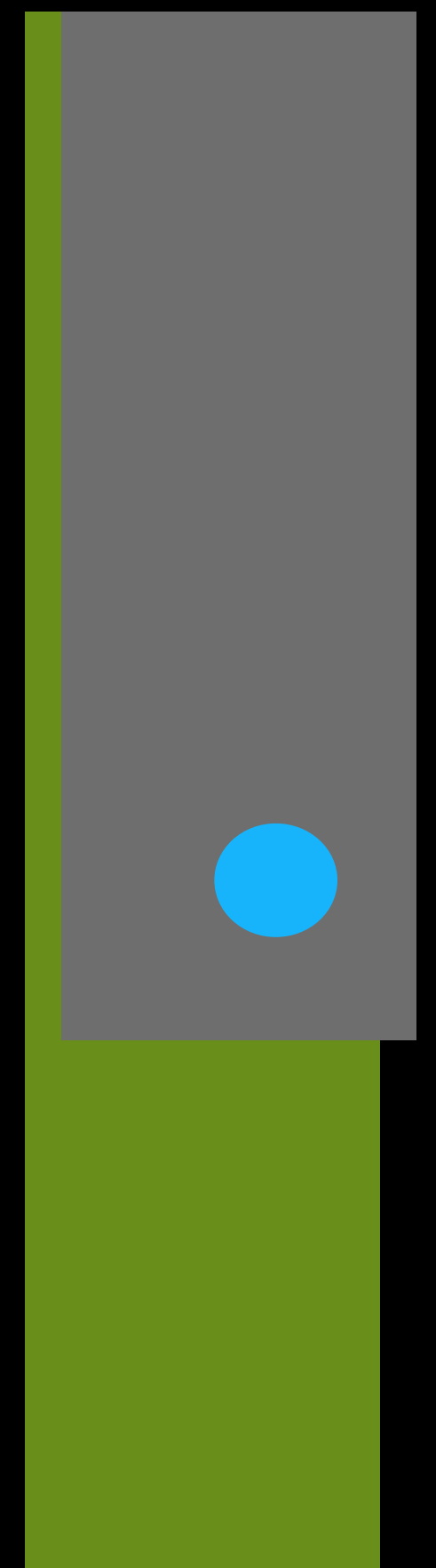
```
- (void) handleGesture:(UIPanGestureRecognizer *)gr {  
    UIViewController *fromVC = [self.transitionContext viewControllerForKey:...];  
    UIViewController *toVC = [self.transitionContext viewControllerForKey:...];  
    switch ([gr state]) {  
        case UIGestureRecognizerStateChanged: {  
            UIView *view = self.isAppearing ? [toVC view] : [fromVC view];  
            view.center = newBlockViewCenter;  
            self.percentComplete = (translation.y / self.toEndFrame.size.height);  
            [self.transitionContext updateInteractiveTransition:self.percentComplete];  
        }  
    }  
}
```



# UIKit Dynamic Transitions

## Drop shade transition

```
- (void) handleGesture:(UIPanGestureRecognizer *)gr {  
  
    UIViewController *fromVC = [self.transitionContext viewControllerForKey:...];  
    UIViewController *toVC = [self.transitionContext viewControllerForKey:...];  
    switch ([gr state]) {  
        case UIGestureRecognizerStateChanged: {  
            UIView *view = self.isAppearing ? [toVC view] : [fromVC view];  
            view.center = newBlockViewCenter;  
            self.percentComplete = (translation.y / self.toEndFrame.size.height);  
            [self.transitionContext updateInteractiveTransition:self.percentComplete];  
        }  
    }  
}
```



# UIKit Dynamic Transitions

## Drop shade transition

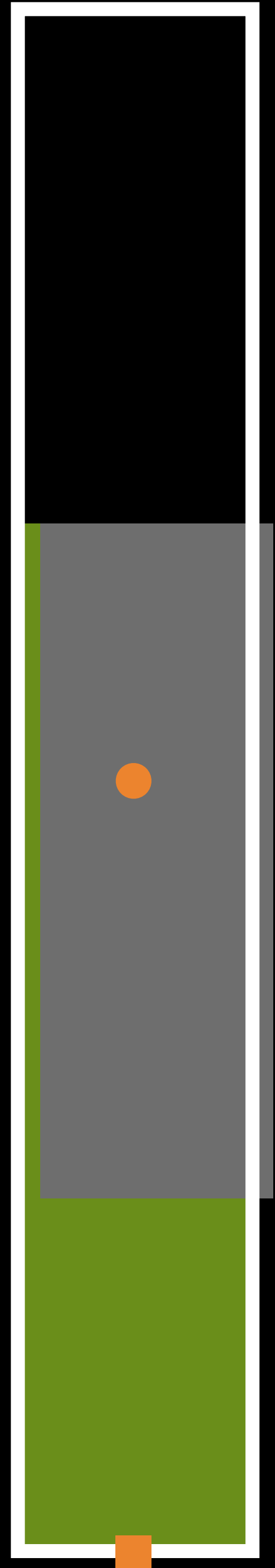
```
case UIGestureRecognizerStateEnded:
case UIGestureRecognizerStateCancelled:

    self.cancelled = [self shouldTransitionComplete:...];

    self.bodyBehavior.elasticity = .6
    [self.bodyBehavior addItem: dynamicView]
    [self.collisionBehavior setTranslatesReferenceBoundsIntoBoundaryWithInsets:];
    [self.collisionBehavior addItem: dynamicView];

    anchor =(self.isAppearing) ? CGPointMake(dynamicView.center.x,
  frame.size.height) :
                                CGPointMake(dynamicView.center.x, -1 * height)
    self.attachBehavior = [[UIAttachBehavior alloc] initWithItem:dynamicItem
  attachedToAnchor:anchor];

    self.attachBehavior.damping = .1;
    self.attachBehavior.frequency = 3.0;
    self.attachBehavior.length = .5 * frame.size.height;
```

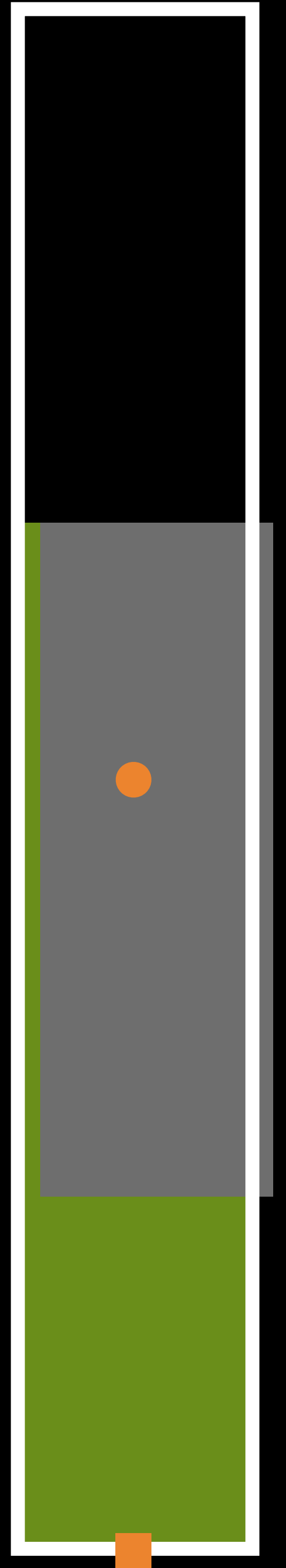


# UIKit Dynamic Transitions

## Drop shade transition

```
self.action = ^{
    if([weakSelf.dynamicAnimator elapsedTime] >
        weakSelf.finishTime) {
        [weakSelf.dynamicAnimator removeAllBehaviors];
    }
    else {
        [weakSelf.transitionContext updateInteractiveTransition:
            [weakSelf percentComplete]];
    }
};

[self.dynamicBehavior addBehavior:self];
```



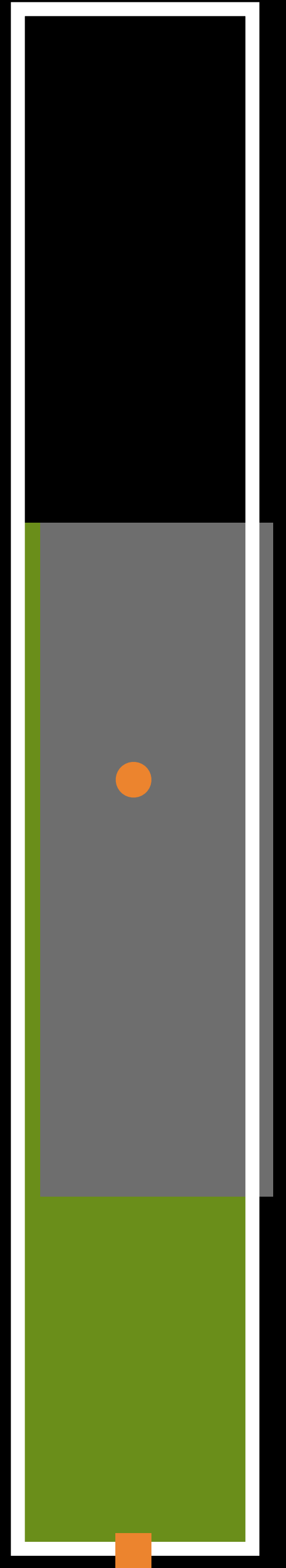


# UIKit Dynamic Transitions

## Drop shade transition

```
self.action = ^{
    if([weakSelf.dynamicAnimator elapsedTime] >
        weakSelf.finishTime) {
        [weakSelf.dynamicAnimator removeAllBehaviors];
    }
    else {
        [weakSelf.transitionContext updateInteractiveTransition:
            [weakSelf percentComplete]];
    }
};

[self.dynamicBehavior addBehavior:self];
```



# UIKit Dynamic Transitions

Drop shade transition



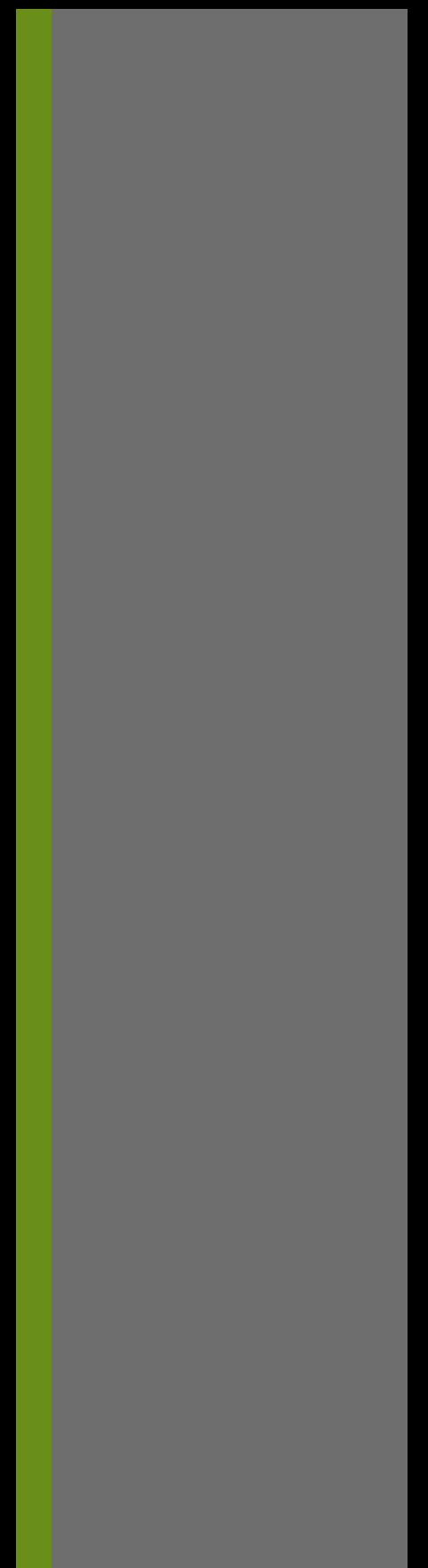
# UIKit Dynamic Transitions

## Drop shade transition

```
- (void)dynamicAnimatorDidPause: ... dynamicAnimator
{
    CGPoint velocity = [self.bodyBehavior linearVelocityForItem:self.dynamicView];
    if([[self.dynamicAnimator behaviors] count] == 0 ||
        velocity.y < .5) {
        if(self.isCancelled)
            [self.transitionContext cancelInteractiveTransition];
        else
            [self.transitionContext finishInteractiveTransition];

        [self.transitionContext completeTransition: !self.isCancelled];

        ...
    }
}
```



# UIKit Dynamic Transitions

What we learned

# UIKit Dynamic Transitions

## What we learned

- UIKit Dynamics and Custom Transitions can be used together!

# UIKit Dynamic Transitions

## What we learned

- UIKit Dynamics and Custom Transitions can be used together!
- Subclass UIDynamicBehavior to create composite behaviors

# UIKit Dynamic Transitions

## What we learned

- UIKit Dynamics and Custom Transitions can be used together!
- Subclass UIDynamicBehavior to create composite behaviors
- Transitions can be comprised of multiple dynamic steps:
  - The UIDynamicAnimator delegate
  - The UICollisionBehavior delegate
  - UIDynamicBehavior actions

# UIKit Dynamic Transitions

What we learned



# UIKit Dynamic Transitions

## What we learned

- A UIDynamicBehavior subclass can conform to one or both transitioning protocols

# UIKit Dynamic Transitions

## What we learned

- A UIDynamicBehavior subclass can conform to one or both transitioning protocols
- How long does it take?

# UIKit Dynamic Transitions

## What we learned

- A UIDynamicBehavior subclass can conform to one or both transitioning protocols
- How long does it take?
  - You can enforce a duration by using the UIDynamicAnimator's elapsedTime

# UIKit Dynamic Transitions

## What we learned

- A UIDynamicBehavior subclass can conform to one or both transitioning protocols
- How long does it take?
  - You can enforce a duration by using the UIDynamicAnimator's elapsedTime
- UIDynamicBehavior actions can call updateInteractiveTransition

# Wrap Up

# Wrap Up

- Focus on your intent
  - What needs animating
  - Constraints (duration, interactivity, etc.)

# Wrap Up

- Focus on your intent
  - What needs animating
  - Constraints (duration, interactivity, etc.)
- Other options may be more suitable

# Wrap Up

- Focus on your intent
  - What needs animating
  - Constraints (duration, interactivity, etc.)
- Other options may be more suitable
- Compose your behaviors, and iterate them to perfection



# Related Sessions

Building User Interfaces for iOS 7

Presidio  
Tuesday 10:15AM

Getting Started with UIKit Dynamics

Presidio  
Tuesday 4:30PM

Introduction to Sprite Kit

Presidio  
Wednesday 11:30AM

Exploring Scroll Views on iOS 7

Presidio  
Thursday 10:15AM

Best Practices for Great iOS UI Design

Presidio  
Friday 10:15AM

 WWDC2013