THE UNIVERSITY OF CHICAGO

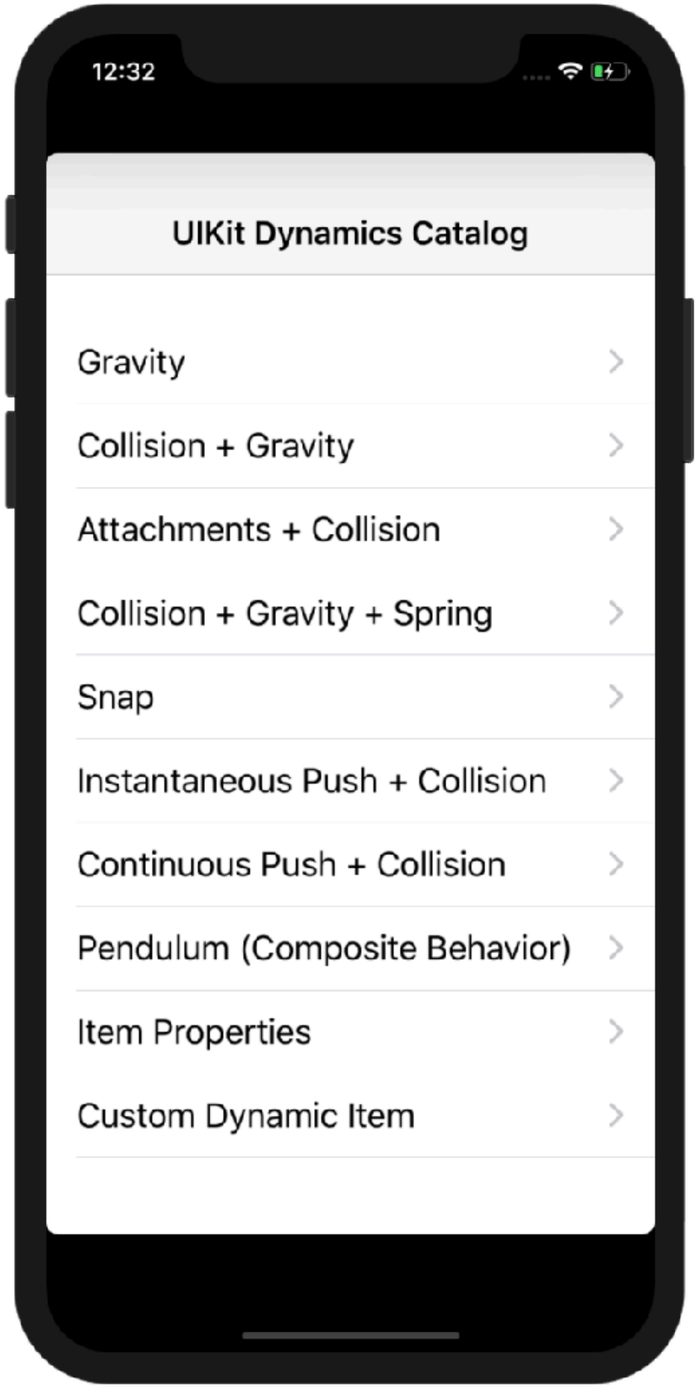
ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 2C

UIKIT DYNAMICS

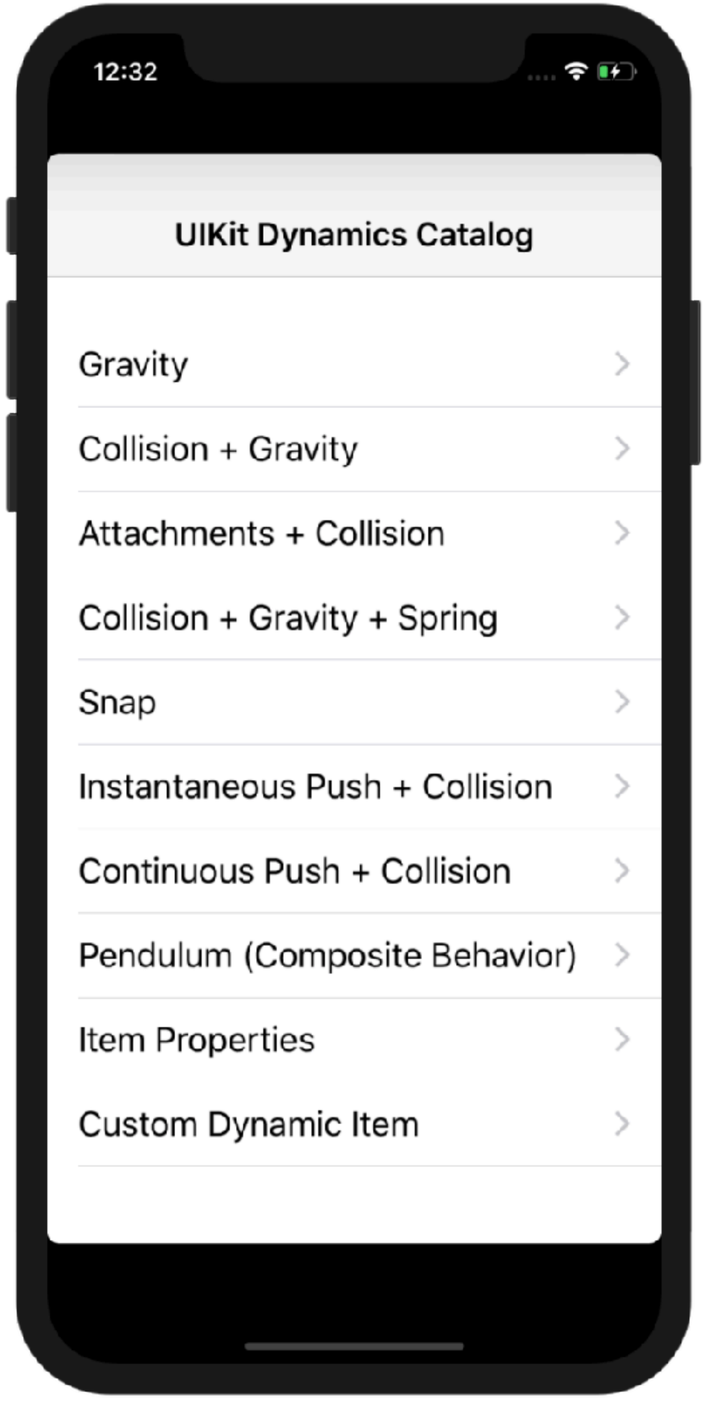
ANIMATIONS AND INTERACTIONS ON IOS

- Core Animation
- UIView animations
- Motion effects
- Gesture driven interactions



WHAT IS UIKIT DYNAMICS?

- A composable, reusable, declarative, real-world inspired animation and interaction system
- Allow physics based interactions
  - Gravity
  - Collisions
  - Attachments
  - Snap
  - Forces
  - Item properties



UIKIT DYNAMICS

SUBTITLE

- Simplify animations that were complex
  - Math
  - Physics
  - Core Animation
- Underlying 2D physics engine
  - Based on Box2D
  - Not “accurate”

Same as SpriteKit

UIKit Dynamics

Apply physics-based animations to your views.

Framework
UIKit

Topics

Dynamic Animators

`class UIDynamicAnimator`
An object that provides physics-related capabilities and animations for its dynamic it
and provides the context for those animations.

Dynamic Items

`protocol UIDynamicItem`
A set of methods that can make a custom object eligible to participate in UIKit Dynar

`class UIDynamicItemBehavior`
A base dynamic animation configuration for one or more dynamic items.

`class UIDynamicItemGroup`
A dynamic item that comprises multiple other dynamic items.

Behaviors

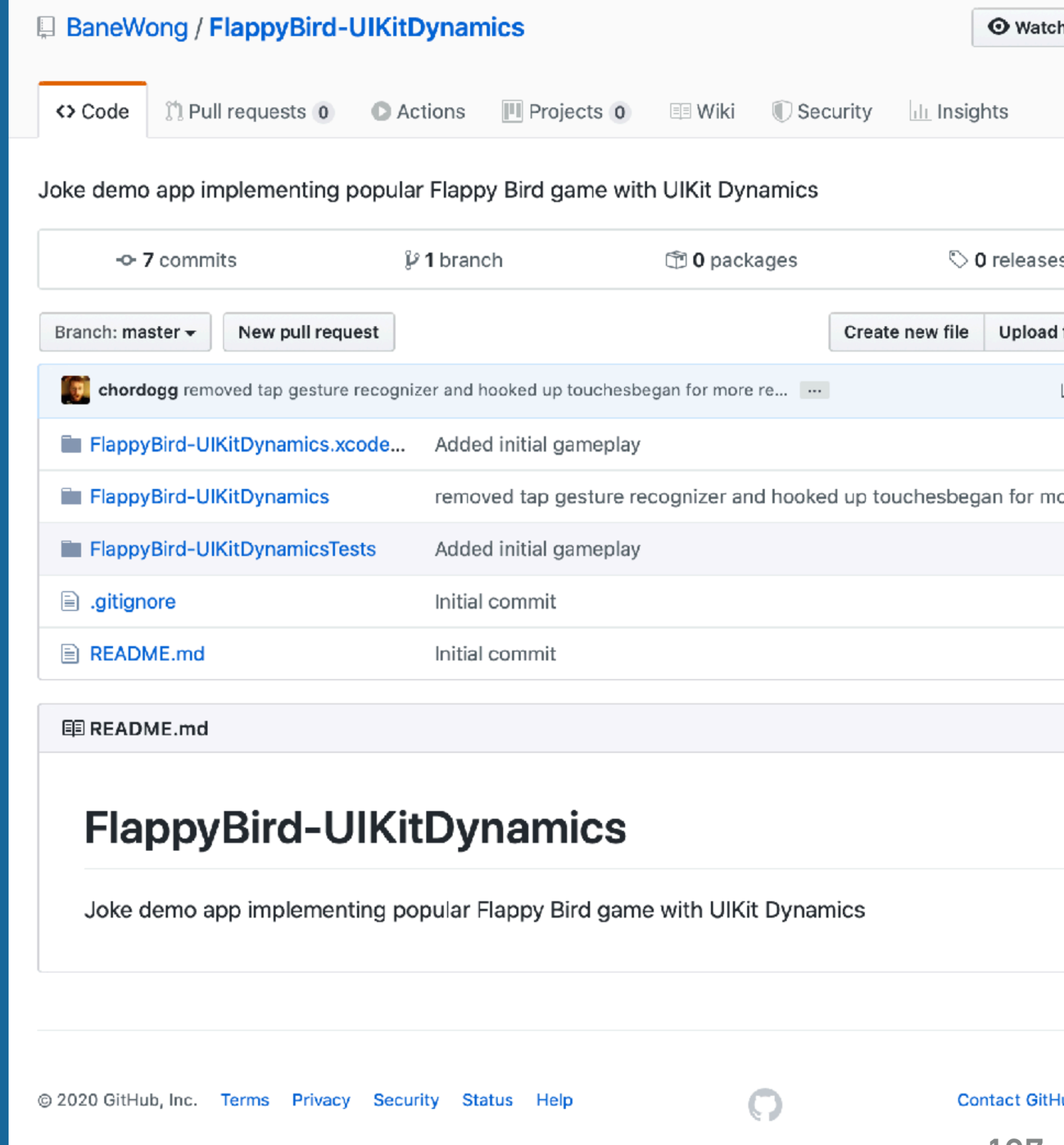
`class UIDynamicBehavior`
An object that confers a behavioral configuration on one or more dynamic items, for
participation in 2D animation.

`class UIAttachmentBehavior`
A relationship between two dynamic items, or between a dynamic item and an ancho
point.

`class UICollisionBehavior`

UIKIT DYNAMICS

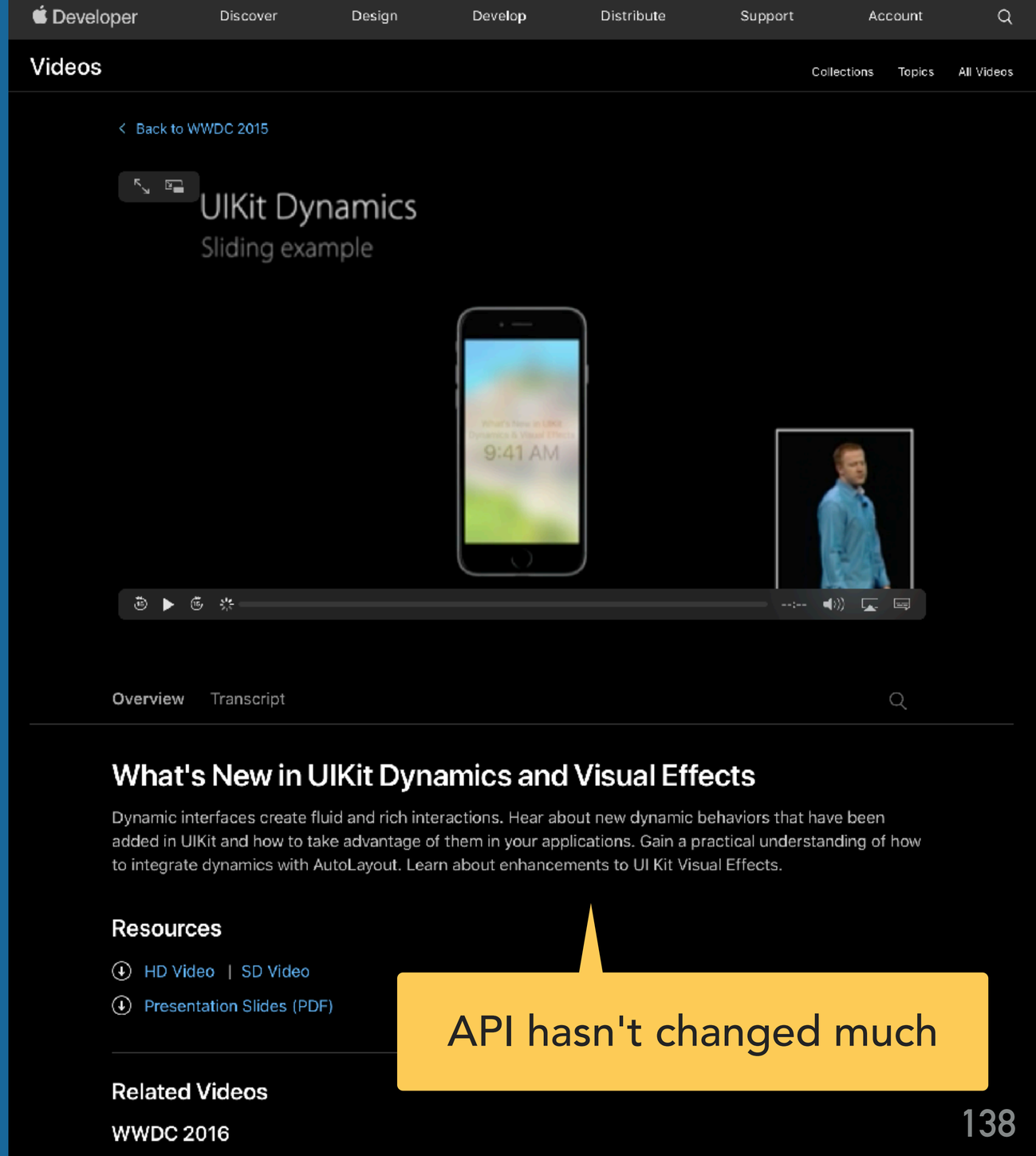
- Not for games
  - Flappy bird using UIKit Dynamics on github
  - Use Sprite Kit instead



WHY?

SUBTITLE

- Real world inspired interactions
  - Combining predefined and interactive animations
- Designed for increased user interface interactions
- Full featured demonstration
  - UIKit Dynamics Catalog from Apple WWDC 2013 (in Objectice-C)
  - WWDC 2015, 2016



UIKITDYNAMICS CLASSES

UIKIT DYNAMICS CLASSES

- UIDynamicAnimator
- UIDynamicBehavior
- UIDynamicItems

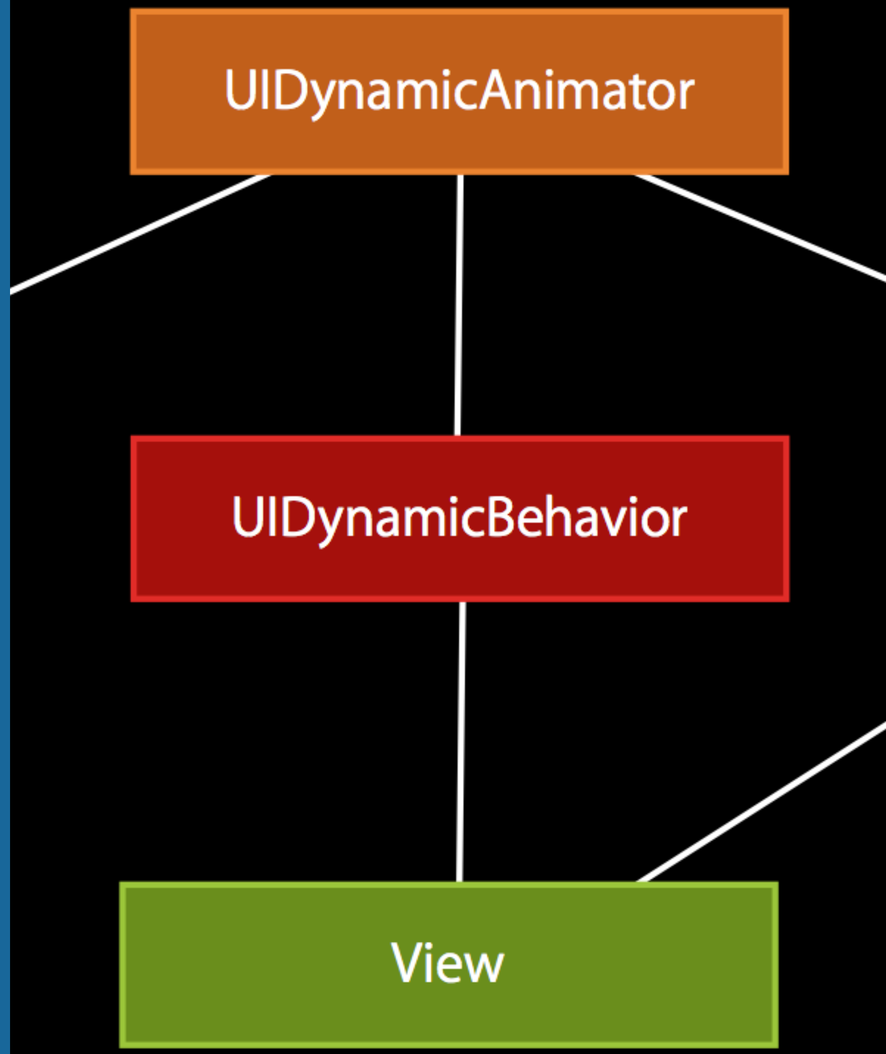


Image via Apple

UIKIT DYNAMICS CLASSES

- Reference View is the UIView to act on

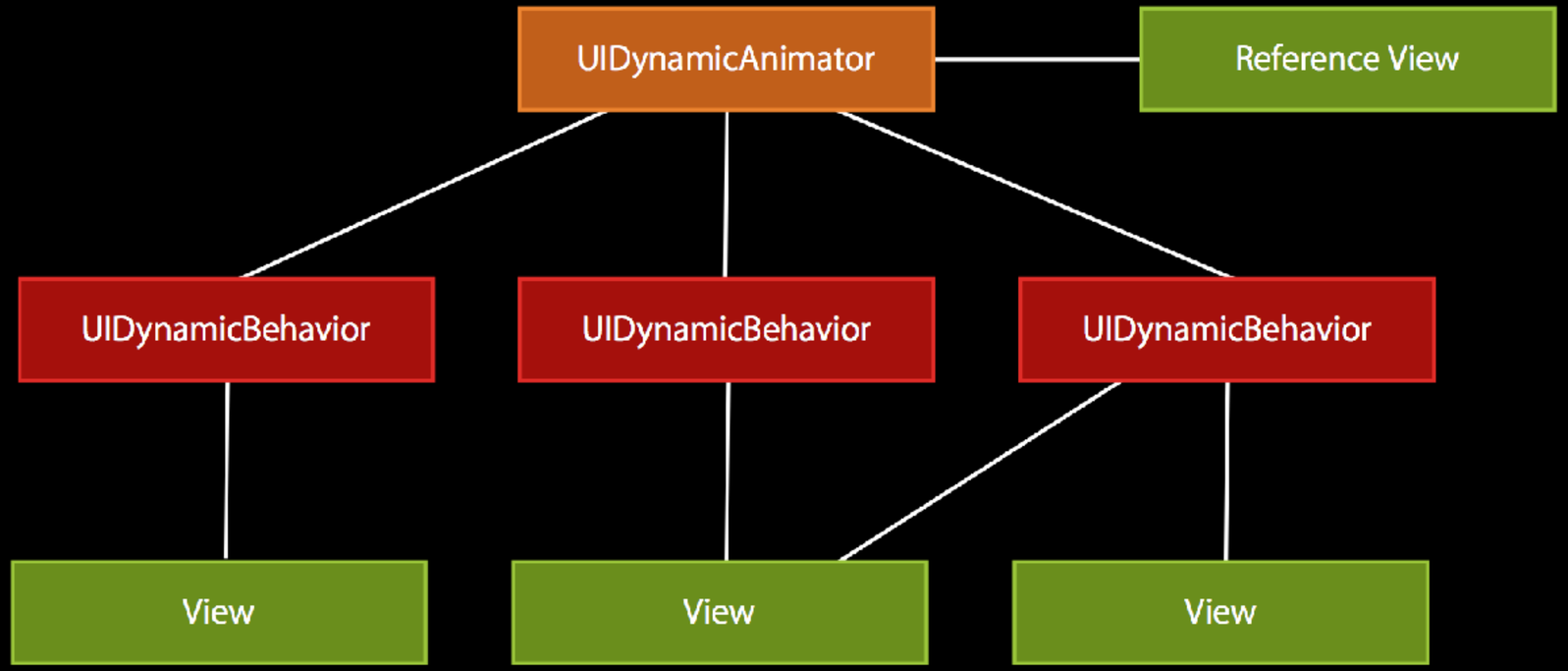


Image via Apple

UIKIT DYNAMICS CLASSES

- Reference view can act on multiple subviews

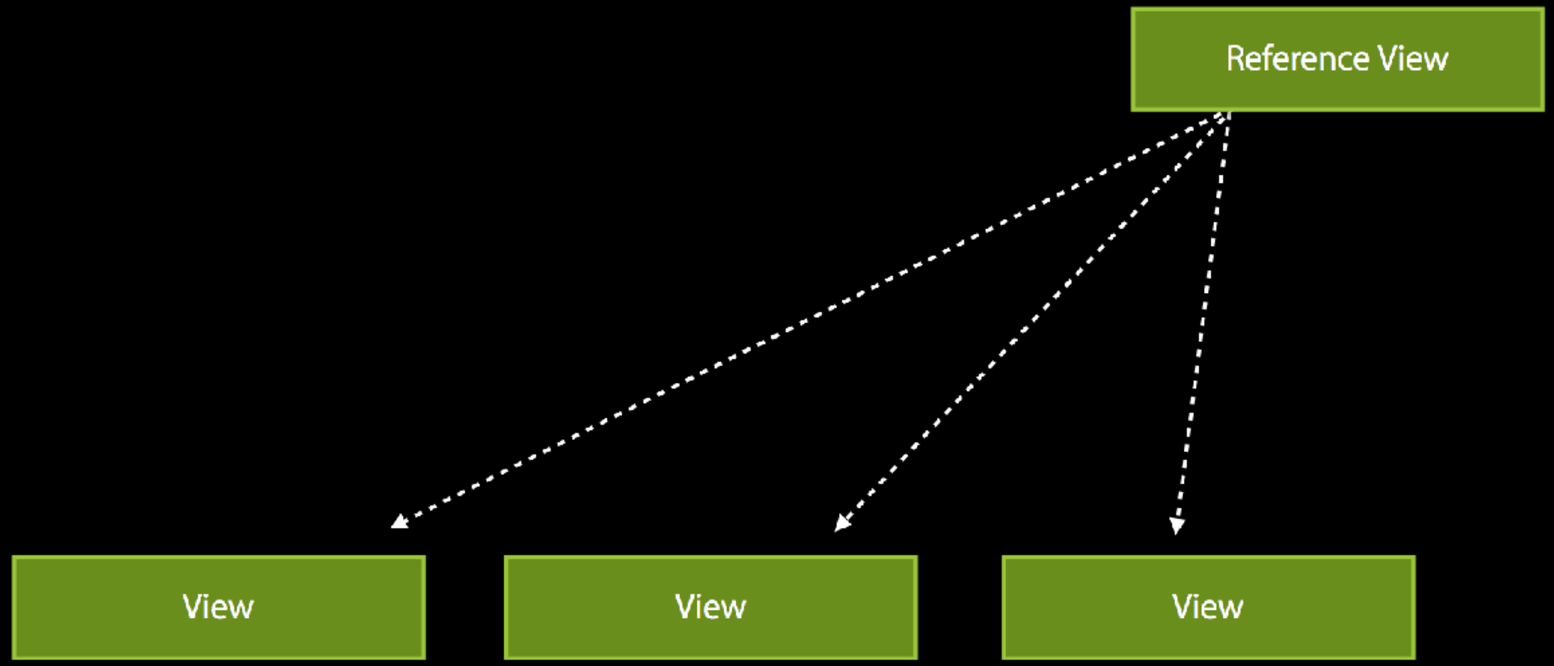
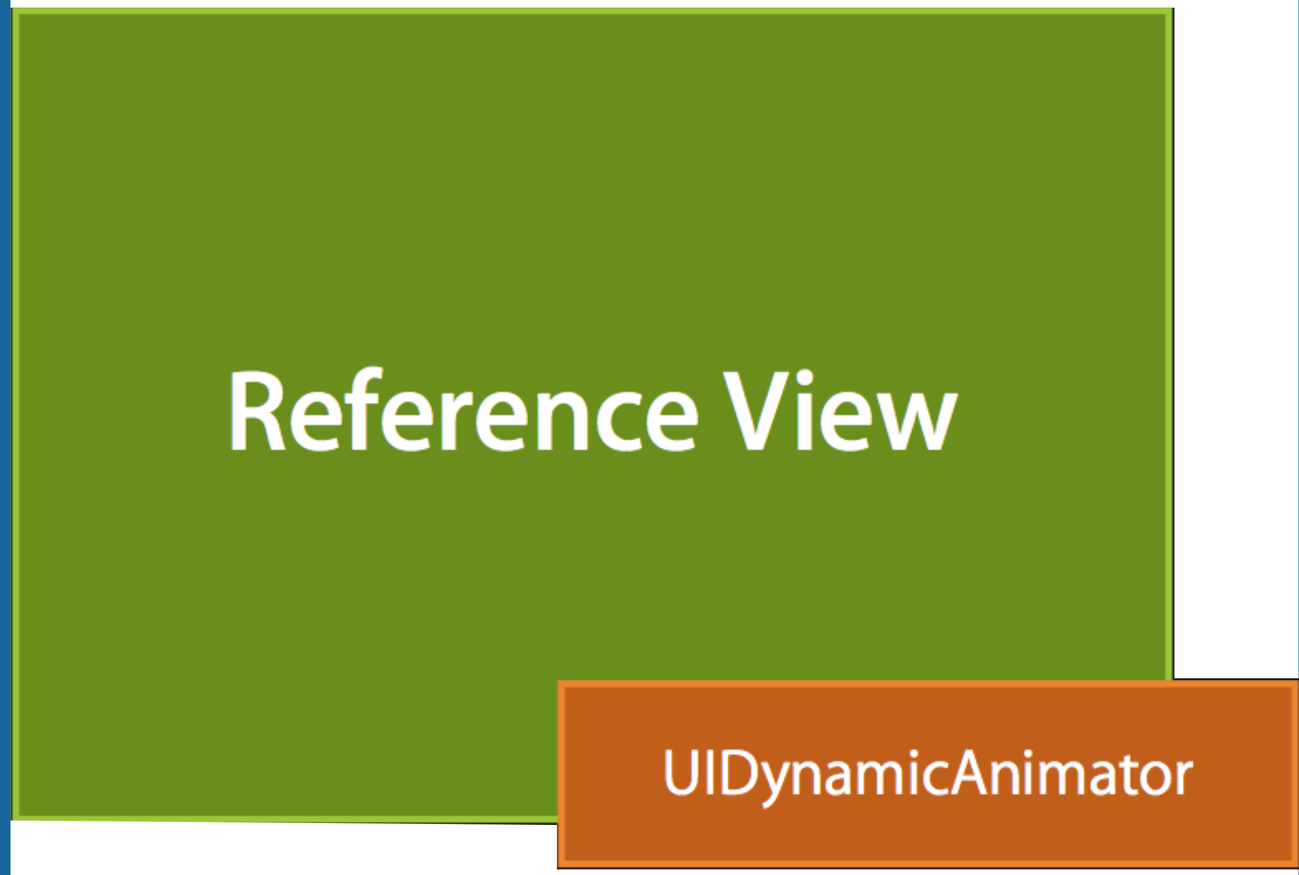


Image via Apple

UIKIT DYNAMICS CLASSES

SUBTITLE

- UIDynamicAnimator
  - Provides the overall context
  - Control engine
  - Define the coordinate system
  - Monitor behaviors



UIKIT DYNAMICS

```
self.animator = UIDynamicAnimator(referenceView: self.view)

let gravity = UIGravityBehavior(items: [square])
gravity.gravityDirection = CGVector(dx: 0, dy: 0.8)
animator?.addBehavior(gravity)
```

- Instantiate a UIDynamicAnimator
- Add behaviors

View that provides the context for the dynamic animator

UIKIT DYNAMICS

```
UIDynamicAnimator *animator = [[UIDynamicAnimator alloc] initWithReferenceView:self.view];
[animator addBehavior:…];
[animator addBehavior:…];
```

- Instantiate a UIDynamicAnimator
- Add behaviors

View that provides the context for the dynamic animator

UIKIT DYNAMICS

Running 2018-session2-uikitdynamics

2018-session2-uikitdynamics 〉 Untitled Page

```
1  //: A UIKit based Playground for presenting user interface
2
3  import UIKit
4  import PlaygroundSupport
5
6  class MyViewController : UIViewController {
7
8    var animator:UIDynamicAnimator? = nil;
9    let gravity = UIGravityBehavior()
10
11
12   override func loadView() {
13     let view = UIView()                                          UIView
14     view.backgroundColor = .white                                UIView
15     self.view = view                                             .MyViewController: 0x7...
16     play()                                                       .MyViewController: 0x7...
17   }
18
19
20   func play() {
21
22     let square = UIView(frame: CGRect(x: 100, y: 100, width: 40, height: 40))   UIView
23     square.backgroundColor = UIColor.gray                        UIView
24     self.view.addSubview(square)                                 UIView
25
26     self.animator = UIDynamicAnimator(referenceView: self.view)  .MyViewController: 0x7...
27
28     let gravity = UIGravityBehavior(items: [square])             <UIGravityBehavior: 0x...
29     gravity.gravityDirection = CGVector(dx: 0, dy: 0.8)          <UIGravityBehavior: 0x...
```

Li...iew 〉 Untitled Page.xcplaygroundpage (Live View)

DYNAMIC BEHAVIORS

DYNAMIC BEHAVIOR

SUBTITLE

- Core unit of composition for a dynamic animation
  - Define the way items interact with animations
- Behaviors are
  - Declarative, describe “influences” on views
  - Added and removed at any time
  - Composable
  - Subclassable

Class

UIDynamicBehavior

An object that confers a behavioral configuration on one or more dynamic i
for their participation in 2D animation.

Overview

A *dynamic item* is any iOS or custom object that conforms to the `UIDynamicItem` protoc
The `UIView` and `UICollectionViewLayoutAttributes` classes implement this prot
starting in iOS 7.0. You can implement this protocol to use a dynamic animator with custo
objects for such purposes as reacting to rotation or position changes computed by an ani
—an instance of the `UIDynamicAnimator` class.

This parent class, `UIDynamicBehavior`, is inherited by the primitive dynamic behavior
classes `UIAttachmentBehavior`, `UICollisionBehavior`, `UIGravityBehavior`,
`UIDynamicItemBehavior`, `UIPushBehavior`, and `UISnapBehavior`.

You can subclass `UIDynamicBehavior`. By using the `addChildBehavior(_:)` metho
an instance of this class or in a custom subclass, you can create composite behaviors of y
own design.

When you subclass `UIDynamicBehavior`, you typically need to provide one or more
initializers, along with other housekeeping methods such as those implemented in the iOS
primitive dynamic behaviors.

To perform per-step logic in a dynamic animation, provide a block object using the `actio`
property.

DYNAMIC BEHAVIOR

SUBTITLE

- Predefined behaviors
  - Gravity
  - Collisions
  - Attachments
  - Snap
  - Forces

Class

UIDynamicBehavior

An object that confers a behavioral configuration on one or more dynamic
for their participation in 2D animation.

Overview

A *dynamic item* is any iOS or custom object that conforms to the `UIDynamicItem` proto
The `UIView` and `UICollectionViewLayoutAttributes` classes implement this pro
starting in iOS 7.0. You can implement this protocol to use a dynamic animator with custo
objects for such purposes as reacting to rotation or position changes computed by an an
—an instance of the `UIDynamicAnimator` class.

This parent class, `UIDynamicBehavior`, is inherited by the primitive dynamic behavior
classes `UIAttachmentBehavior`, `UICollisionBehavior`, `UIGravityBehavior`,
`UIDynamicItemBehavior`, `UIPushBehavior`, and `UISnapBehavior`.

You can subclass `UIDynamicBehavior`. By using the `addChildBehavior(_:)` meth
an instance of this class or in a custom subclass, you can create composite behaviors of
own design.

When you subclass `UIDynamicBehavior`, you typically need to provide one or more
initializers, along with other housekeeping methods such as those implemented in the iO
primitive dynamic behaviors.

To perform per-step logic in a dynamic animation, provide a block object using the `acti`
property.

DYNAMIC BEHAVIOR

```swift
self.animator = UIDynamicAnimator(referenceView: self.view)

let gravity = UIGravityBehavior(items: [square])
gravity.gravityDirection = CGVector(dx: 0, dy: 0.8)
animator?.addBehavior(gravity)
```

Add gravity to an Animator

- Instantiate a UIDynamicBehavior
  - Initialize with view(s)
- Add behaviors to animator

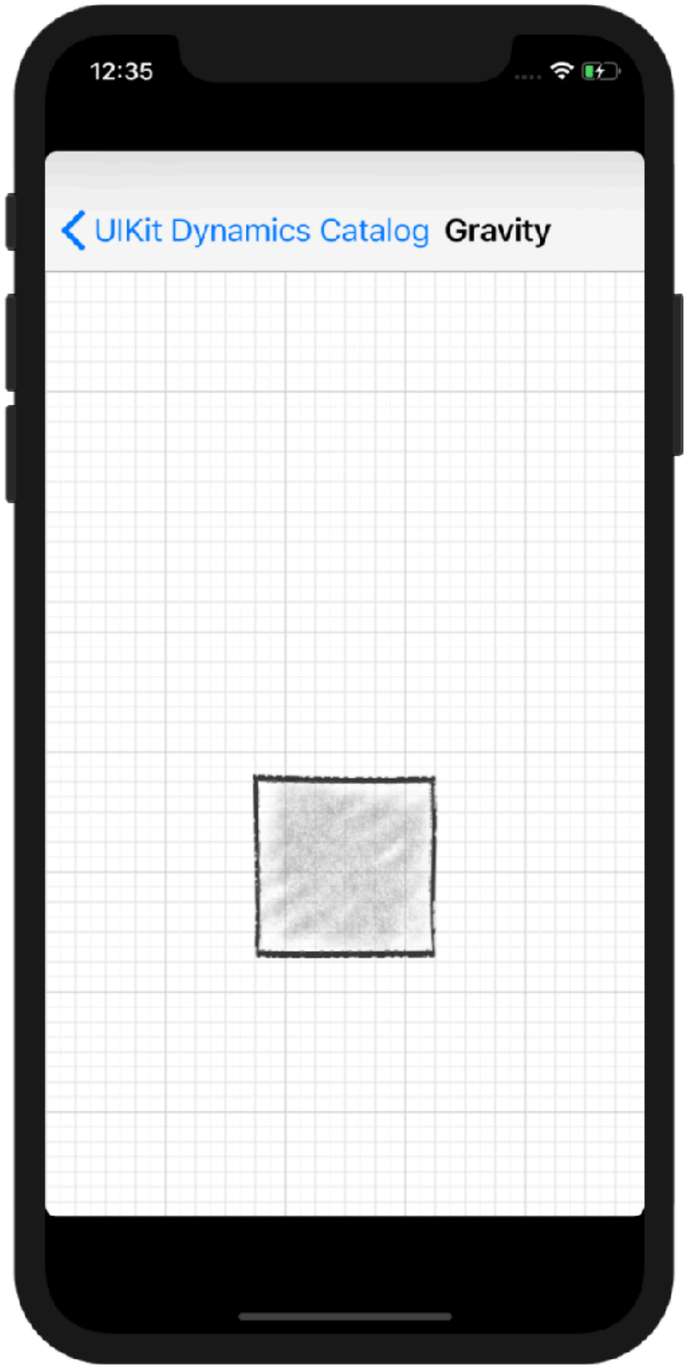
```objc
myBehavior = [[MyBehavior alloc] initWith...];
[animator addBehavior:myBehavior];
```

DYNAMIC BEHAVIOR

- Add views to behaviors
- Remove views from behaviors
  - The influence stops when the behavior is removed
- Caveats
  - Possible to create impossible scenarios
  - Not physics-accurate

DYNAMIC BEHAVIORS

- UIGravityBehavior
  - A simple gravity vector
  - UIKit coordinate system
    - (0,1) by default
    - Items can be added and removed at any time
  - UIKit gravity
    - 1000 p/s^2



DYNAMIC BEHAVIORS

Add gravity to the animator

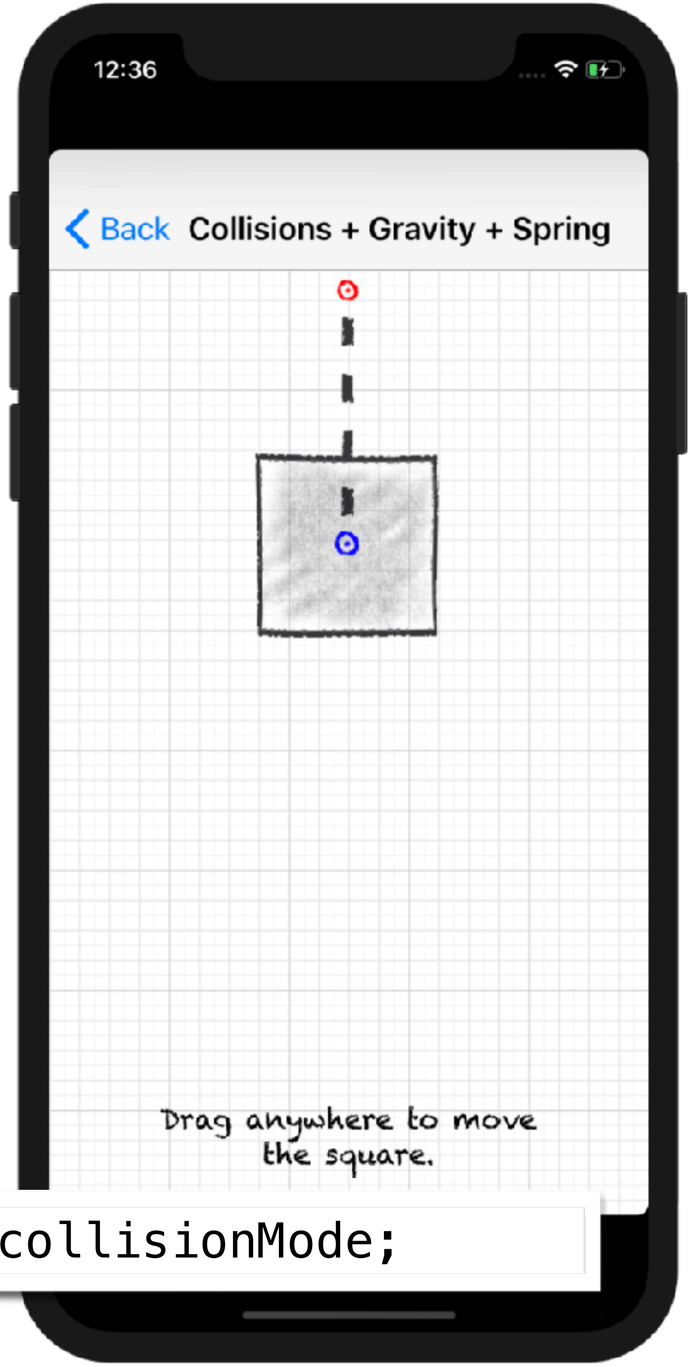
```
UIDynamicAnimator *animator = [[UIDynamicAnimator alloc]
                                    initWithReferenceView:self.view];

UIGravityBehavior *gravityBehavior = [[UIGravityBehavior alloc]
                                    initWithItems:@[self.square1]];

[animator addBehavior:gravityBehavior];
```

- UIGravityBehavior

```
self.animator = UIDynamicAnimator(referenceView: self.view)

let gravity = UIGravityBehavior(items: [square])
gravity.gravityDirection = CGVector(dx: 0, dy: 0.8)
animator?.addBehavior(gravity)
```

DYNAMIC BEHAVIORS

- UICollisionBehavior
  - Between a view and a boundary
  - Or between views associated to the same behavior
  - Or both, by default




```
@property (nonatomic, readwrite) UICollisionBehaviorMode collisionMode;
```

DYNAMIC BEHAVIORS

```
UIDynamicAnimator *animator = [[UIDynamicAnimator alloc] initWithReferenceView:self.view];




UICollisionBehavior *collisionBehavior = [[UICollisionBehavior alloc]
                                                    initWithItems:@[self.square1]];

// Creates collision boundaries from the bounds of the dynamic animator's
// reference view (self.view).
collisionBehavior.translatesReferenceBoundsIntoBoundary = YES;
collisionBehavior.collisionDelegate = self;
[animator addBehavior:collisionBehavior];
```

Delegate provides information about the collision

Set boundary based on reference view

- UICollisionBehavior

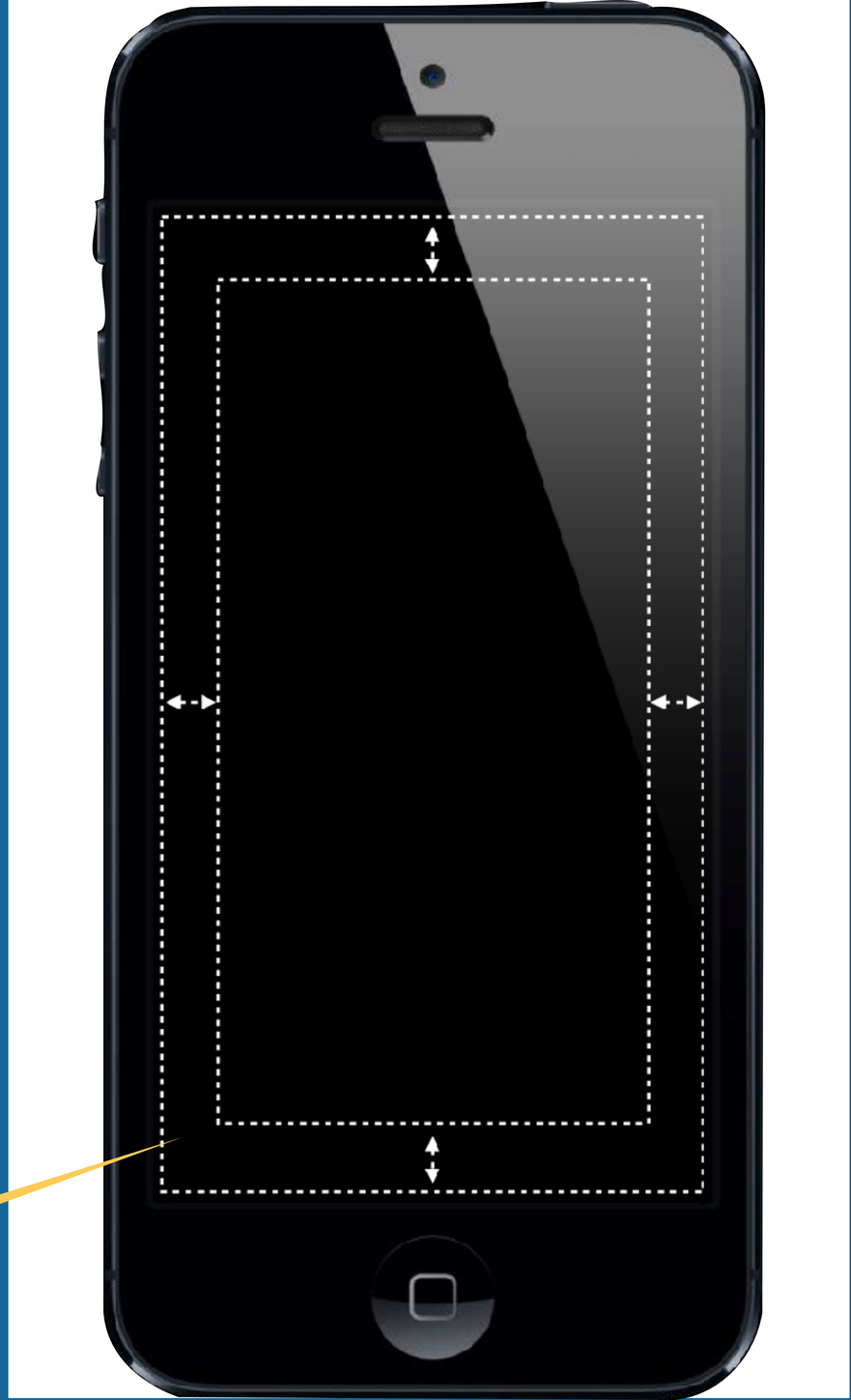
DYNAMIC BEHAVIORS

- UICollisionBehavior boundaries
  - Set automatically from frame

```
@property (nonatomic, readwrite) BOOL translatesReferenceBoundsIntoBoundary;
```

  - Optional inset

```
-(void)setTranslatesReferenceBoundsIntoBoundaryWithInsets:(UIEdgeInsets)insets;
```



DYNAMIC BEHAVIORS

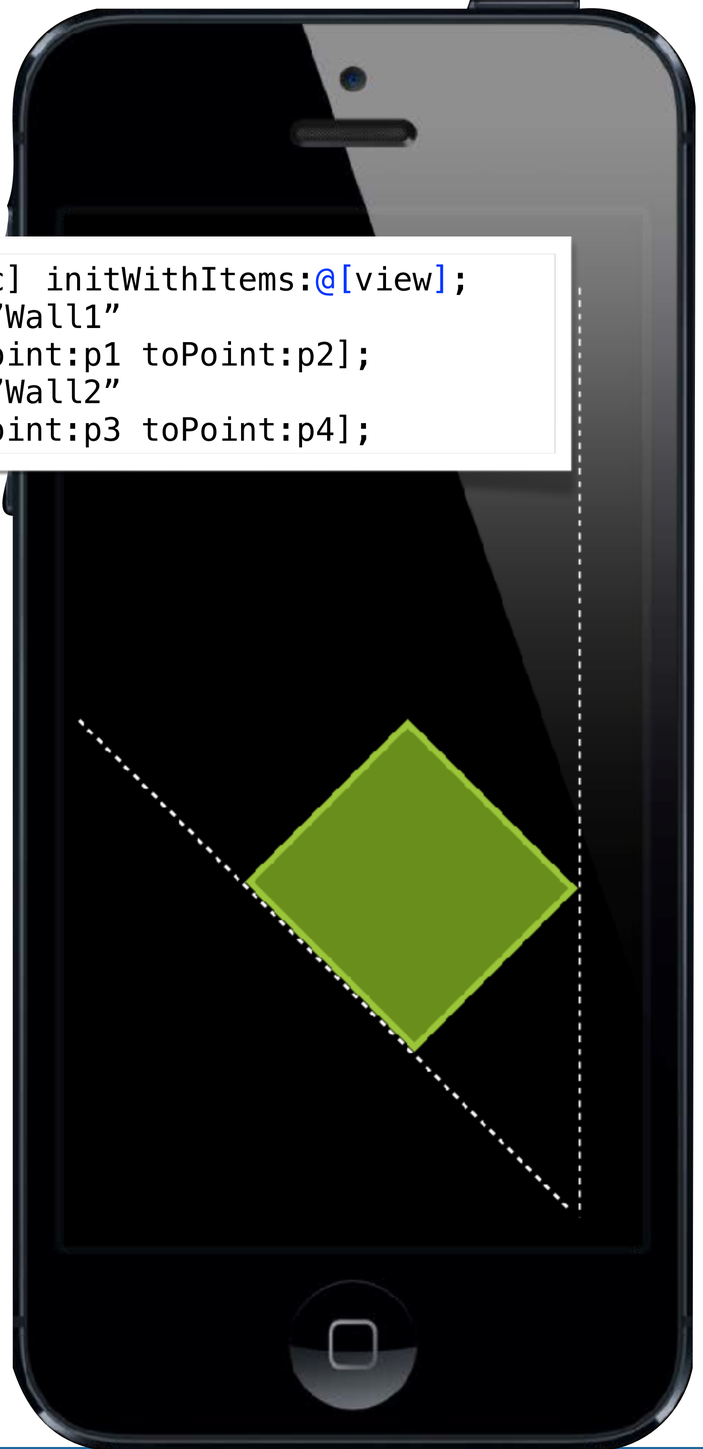
- Adding collision behavior

```
let square = UIView(frame: CGRect(x: 100, y: 100, width: 40, height: 40))
square.backgroundColor = UIColor.gray
self.view.addSubview(square)

// Instantiate the Animator
self.animator = UIDynamicAnimator(referenceView: self.view)

// Setup a behavior
let gravity = UIGravityBehavior(items: [square])
gravity.gravityDirection = CGVector(dx: 0, dy: 0.8)
// Add the behavior to the animator
animator?.addBehavior(gravity)

// Setup up another behavior
let collision = UICollisionBehavior(items: [square])
collision.translatesReferenceBoundsIntoBoundary = true
// Add to the animator
animator?.addBehavior(collision)

// Create a new view for the boundary
let barrier = UIView(frame: CGRect(x: 0, y: 300, width: 110, height: 20))
barrier.backgroundColor = UIColor.red
self.view.addSubview(barrier)

let rightEdge = CGPoint(x: barrier.frame.origin.x + barrier.frame.size.width,
                        y: barrier.frame.origin.y)
collision.addBoundary(withIdentifier: "barrier" as NSCopying,
                      from: barrier.frame.origin, to: rightEdge)
```

DYNAMIC BEHAVIORS

```
c = [[UICollisionBehavior alloc] initWithItems:@[view];
[c addBoundaryWithIdentifier:@”Wall1”
                                fromPoint:p1 toPoint:p2];
[c addBoundaryWithIdentifier:@”Wall2”
                                fromPoint:p3 toPoint:p4];
```

- Explicitly with segments
  - - (void)addBoundaryWithIdentifier: (id)identifier fromPoint:(CGPoint)p1 toPoint: (CGPoint)p2;
- • Or paths (approximated)
  - - (void)addBoundaryWithIdentifier: (id)identifierforPath:(UIBezierPath*)p;



DYNAMIC BEHAVIORS

- UICollisionBehaviorDelegate
  - Between views
    - - collisionBehavior: beganContactForItem: withItem: atPoint:
    - - collisionBehavior: endedContactForItem: withItem:
  - • Or boundaries
    - - collisionBehavior: beganContactForItem: withBoundaryIdentifier: atPoint:
    - - collisionBehavior: endedContactForItem: withBoundaryIdentifier:
  - The reference boundary identifier is always nil
  - Conform to protocol <UICollisionBehaviorDelegate>

DYNAMIC BEHAVIORS

- Handling contacts

begin Contact

```
func collisionBehavior(_ behavior: UICollisionBehavior,
                       beganContactFor item: UIDynamicItem,
                       withBoundaryIdentifier identifier: NSCopying?,
                       at p: CGPoint) {
  print("Item: \n\t\(item) hit point: \n\t\(p)")

  // Cast the colliding item as a Box
  let collidingView = item as! Box

  // Highlight the collision
  collidingView.backgroundColor = UIColor.yellow
  UIView.animate(withDuration: 0.3, animations: {
    collidingView.backgroundColor = collidingView.color
  })




}
```

DYNAMIC BEHAVIORS

- Contact protocol methods

begin Contact

```
//| ------------------------------------------------------------------------
//  This method is called when square1 begins contacting a collision boundary.
//  In this demo, the only collision boundary is the bounds of the reference
//  view (self.view).
//
- (void)collisionBehavior:(UICollisionBehavior*)behavior beganContactForItem:
(id<UIDynamicItem>)item withBoundaryIdentifier:(id<NSCopying>)identifier atPoint:
(CGPoint)p
{
    // Lighten the tint color when the view is in contact with a boundary.
    [(UIView*)item setTintColor:[UIColor lightGrayColor]];
}
```

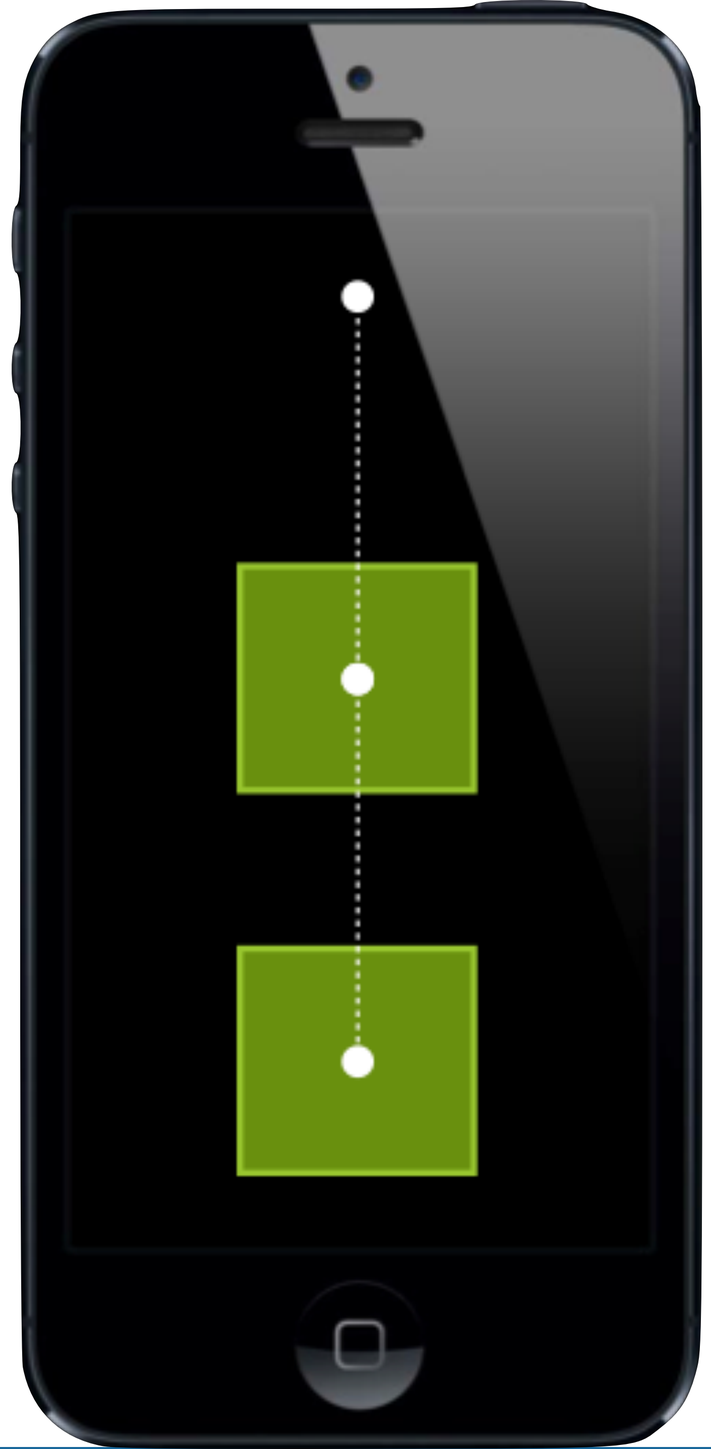
end Contact

```
//| ------------------------------------------------------------------------
//  This method is called when square1 stops contacting a collision boundary.
//  In this demo, the only collision boundary is the bounds of the reference
//  view (self.view).
//
- (void)collisionBehavior:(UICollisionBehavior*)behavior endedContactForItem:
(id<UIDynamicItem>)item withBoundaryIdentifier:(id<NSCopying>)identifier
{
    // Restore the default color when ending a contcact.
    [(UIView*)item setTintColor:[UIColor darkGrayColor]];
}
```

DYNAMIC BEHAVIORS

```
a1 = [[UIAttachmentBehavior alloc]
      initWithItem:v1 attachedToAnchor:ap];
a2 = [[UIAttachmentBehavior alloc]
                initWithItem:v1 attachedToItem:v2];
```

- UIAttachmentBehavior
  - Between a view and an anchor point
  - Optional inset

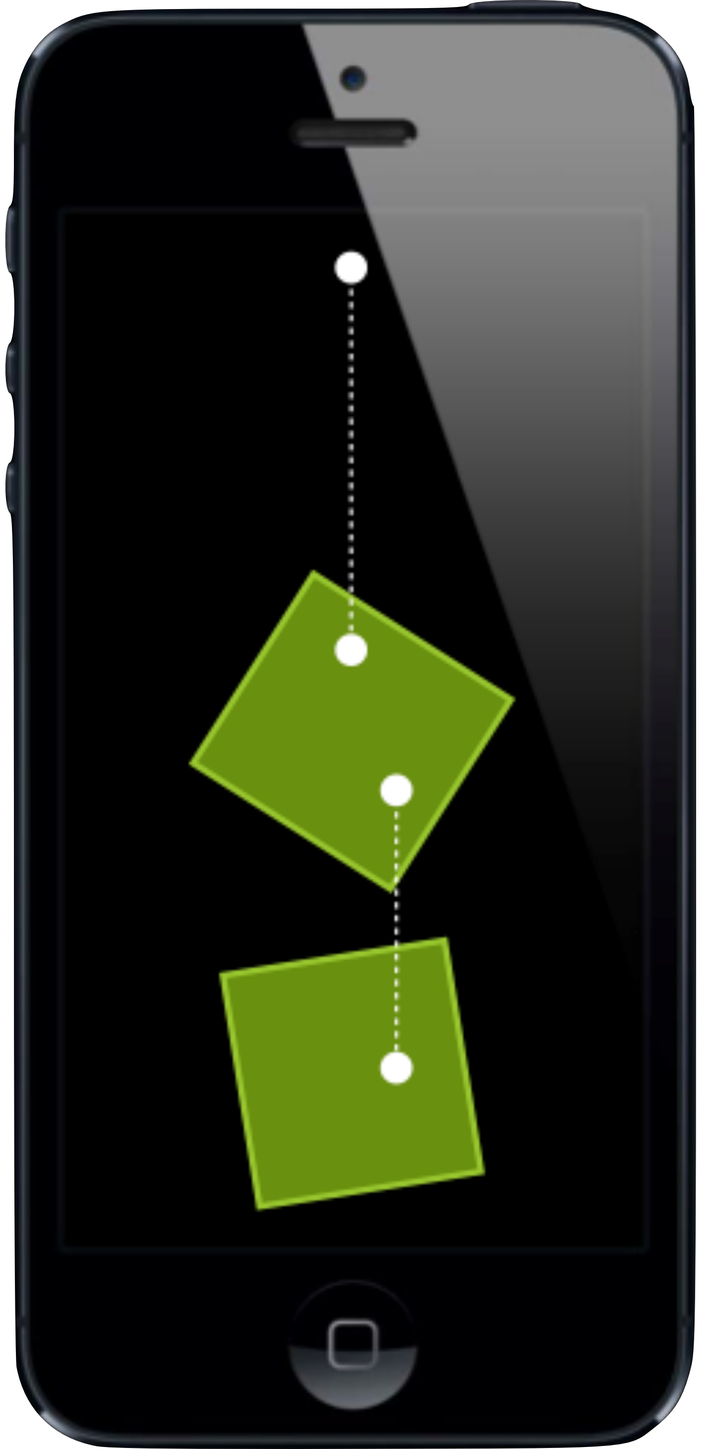


DYNAMIC BEHAVIORS

SUBTITLE

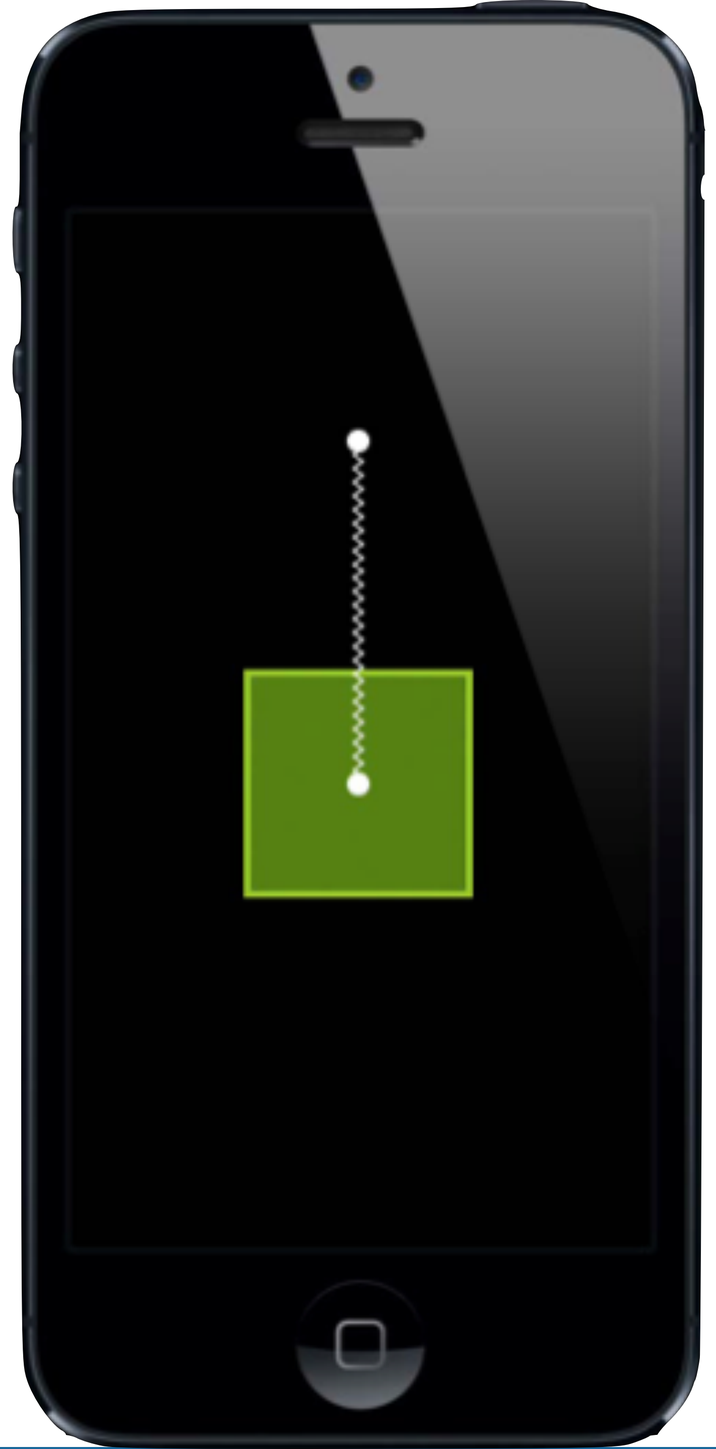
- UIAttachmentBehavior
  - Offset

```
a1 = [[UIAttachmentBehavior alloc]
                initWithItem:v1 point:p1
                attachedToAnchor:ap];
a2 = [[UIAttachmentBehavior alloc]
                initWithItem:v1 point:p2
                attachedToItem:v2];
```



DYNAMIC BEHAVIORS

- UIAttachmentBehavior
  - An attachment can act as a spring
    - [a setFrequency:4.0];
    - [a setDamping:0.5];
  - An anchor point can be modified later
  - Attachments are invisible!



DYNAMIC BEHAVIORS

- UISnapBehavior
  - Snap a view to place
  - Ensure position and angle

```
s = [[UISnapBehavior alloc] initWithItem:v
                            snapToPoint:p];
[animator addBehavior:s];
```

DYNAMIC BEHAVIORS

```
override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
  if (snap != nil) {
    animator.removeBehavior(snap)
  }

  let touch = touches.first! as UITouch
  snap = UISnapBehavior(item: square, snapTo: touch.location(in: view))
  animator.addBehavior(snap)
}
```

- UISnapBehavior
  - Snap move view to a touch

DYNAMIC BEHAVIORS

- UIPushBehavior
  - Snap a view to place
  - Ensure position and angle
  - Damping is customizable
- A simple force vector
  - @property (readwrite,nonatomic) CGFloat xComponent;
  - @property (readwrite,nonatomic) CGFloat yComponent;
  - @property (readwrite,nonatomic) CGFloat angle;
  - @property (readwrite,nonatomic) CGFloat magnitude;

DYNAMIC BEHAVIORS

UIPushModeContinuous

- UIPushBehavior
  - Target point can be customizable

```
p = [[UIPushBehavior alloc]
                 initWithItems:@[view]
                 mode:UIPushBehaviorModeContinuous];
 [p setTargetPoint:x forItem:view];
```

DYNAMIC BEHAVIORS

- UIPushBehavior
  - UIKit newton 100 p/s2
  - Instantaneous mode
  - Velocity change is instantaneous

```
p = [[UIPushBehavior alloc]
                 initWithItems:@[view]
                 mode:UIPushBehaviorModeInstantaneous];
[p setActive:TRUE]
```

DYNAMIC ITEM BEHAVIOR

DYNAMIC ITEM BEHAVIOR

- UIDynamicItemProtocol
  - Applied to one or many items
  - Change item-level properties
    - friction
    - resistance
    - angularResistance
    - elasticity
    - density
    - allowsRotation
  - Directly add angular or linear velocities
    - i.e. map with a previous gesture

DYNAMIC ITEM BEHAVIOR

- A protocol for items associated to predefined behaviors
- Describe what UIKit needs to animate an item
- UIView implements it
- You can implement it

DYNAMIC BEHAVIOR

- UIDynamicItem
  - A protocol for items associated to predefined behaviors
  - Describe what UIKit needs to animate an item
  - UIView implements it
  - You can implement it

```
@protocol UIDynamicItem <NSObject>
@property (nonatomic, readwrite) CGPoint center;
@property (nonatomic, readonly)  CGRect bounds;
@property (nonatomic, readwrite) CGAffineTransform transform;
@end
```

ONLY 2D

DYNAMIC ITEM BEHAVIOR

```
behavior.action =  ^{
    NSLog(@"%@, %@",
          NSStringFromCGAffineTransform(square.transform),
          NSStringFromCGPoint(square.center));
};
```

- What else
  - Each dynamic behavior has an action property where you supply a block to be executed with every step of the animation

DYNAMIC ITEM BEHAVIOR

- Put them all together to have custom behavior and effects for your views
- Don't forget about SpriteKit

DEMO PROJECT
UIKITDYNAMICS

THE UNIVERSITY OF CHICAGO

ADVANCED iOS APPLICATION DEVELOPMENT

MPCS 51032 • SPRING 2020 • SESSION 2C