iOS Dev Accelerator
Week2 Day4

- Collection View Layouts
- Gesture Recognizers
- AVFoundation
- Generics
- Technical Thursday - Linked Lists

CollectionView Layout

UICollectionViewLayout

- Computes layout attributes as needed for:
- CollectionView Cells
- CollectionView Supplementary Views
- Decoration Views

UICollectionViewLayout

- Used to by `UICollectionView` to calculate position of each cell
- Apple provides a subclass of `UICollectionViewLayout` called `UICollectionViewFlowLayout` for a grid or linear layout.
- `UICollectionViewFlowLayout` will fit most needs.
- A collection view's layout is highly customizable. When you want to create a custom layout, you first need to determine if it is suitable for you to subclass flow layout (less work), or create a brand new subclass of UICollectionViewLayout(more work).

UICollectionViewLayoutAttributes

- Manages the layout-related attributes for a given item in a collection view:
  - Position
  - Size
  - Opacity
  - zIndex (overlapping cells, above or below)
  - Transforms
- One attribute instance per view!

UICollectionViewFlowLayout

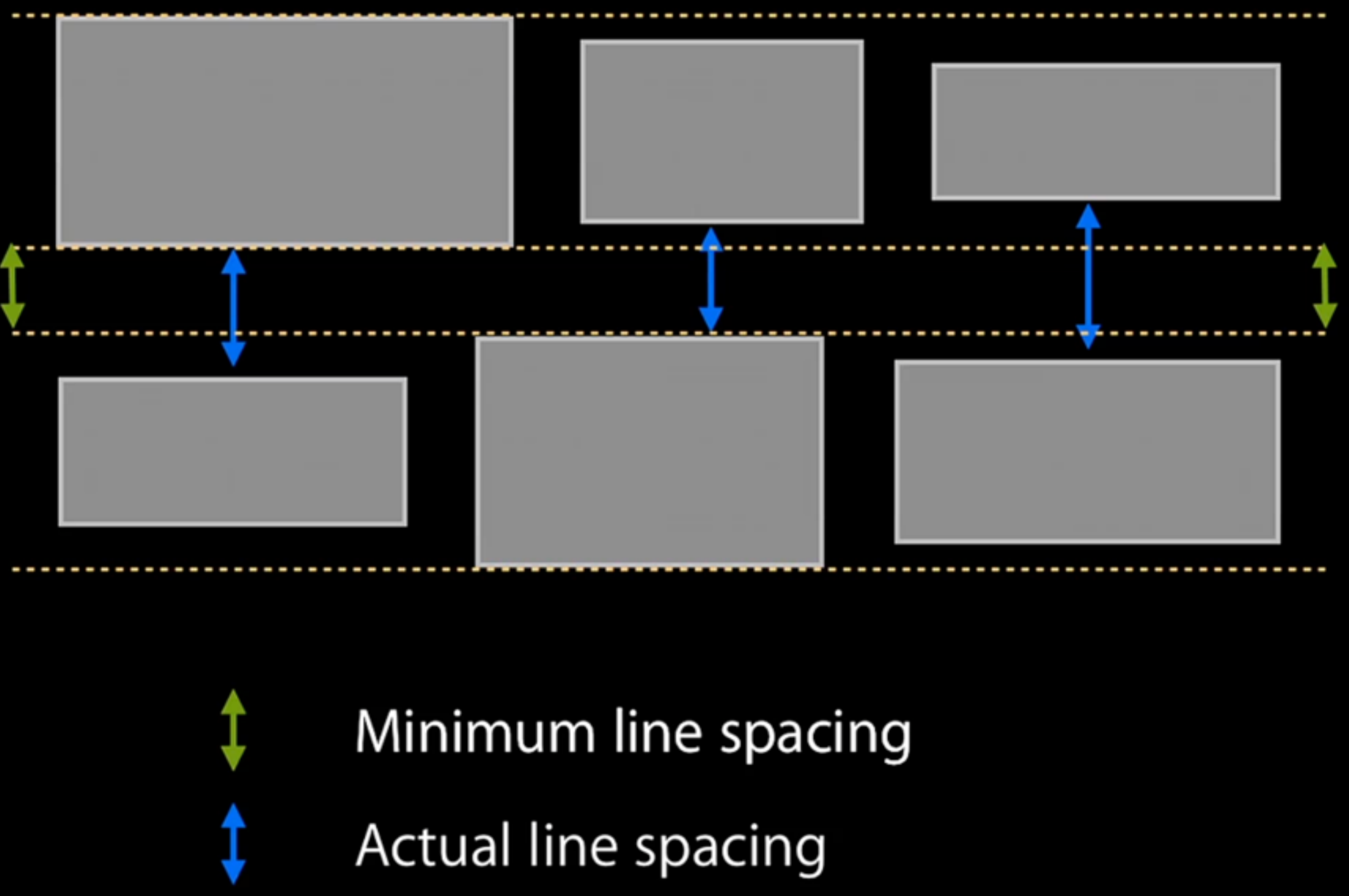
- Flow layout is a line-oriented layout. The layout object places cells on a linear path and fits as many cells as it can along the line.When the line runs out of room, it creates a new line and continues the process.
- Can be configured as a grid or as a group of lines.
- Out of the box, it has lots of things you can customize:
  - Item Size
  - Line Spacing and Inter Cell spacing
  - Scrolling direction
  - Header and footer size
  - Section Inset
- And you customize each of those things either globally with a single property, or through a delegate

Item Size

- The item size for each cell can be set globally by setting the itemSize property on your flow layout.
- Or if you want different size per item, you can do it through the delegate method collectionView:layout:sizeForItemAtIndexPath()

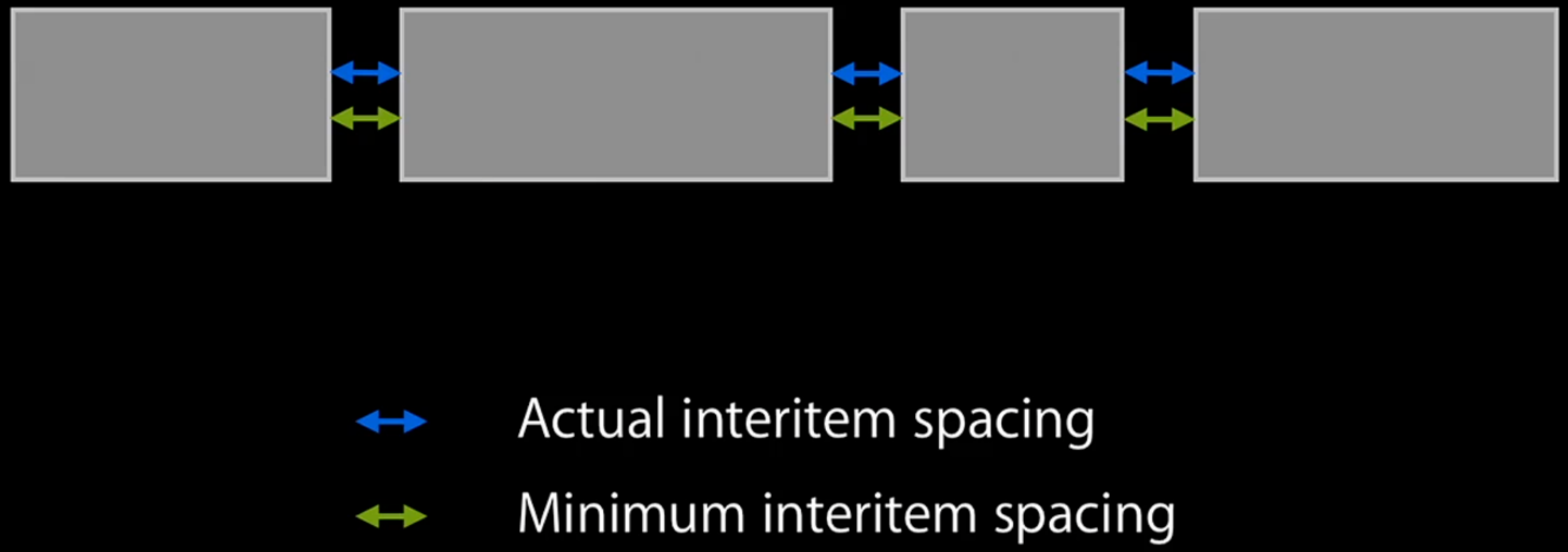
Line Spacing

- You can set a minimum line spacing, either globally or through the delegate:



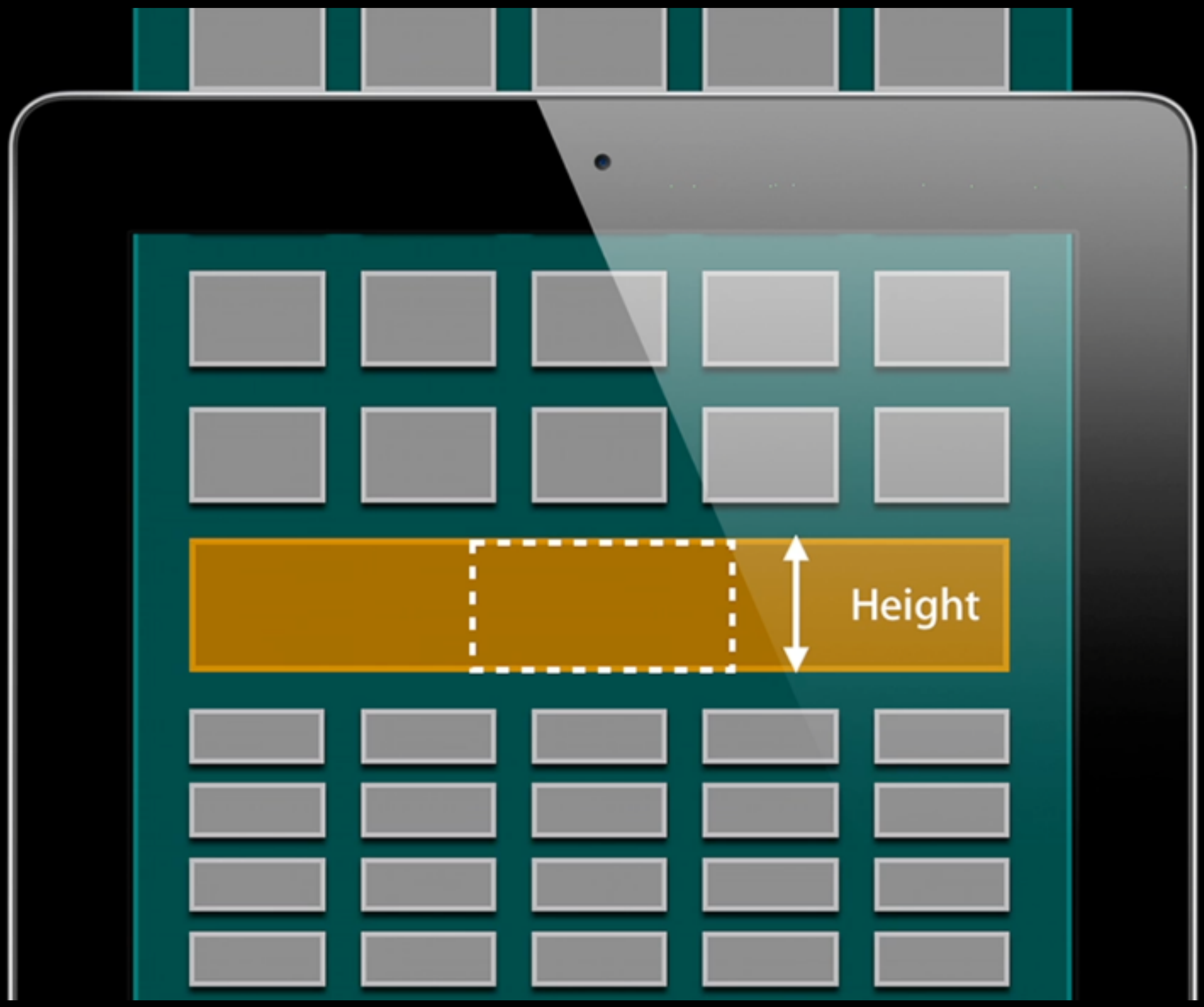
Inter-item Spacing

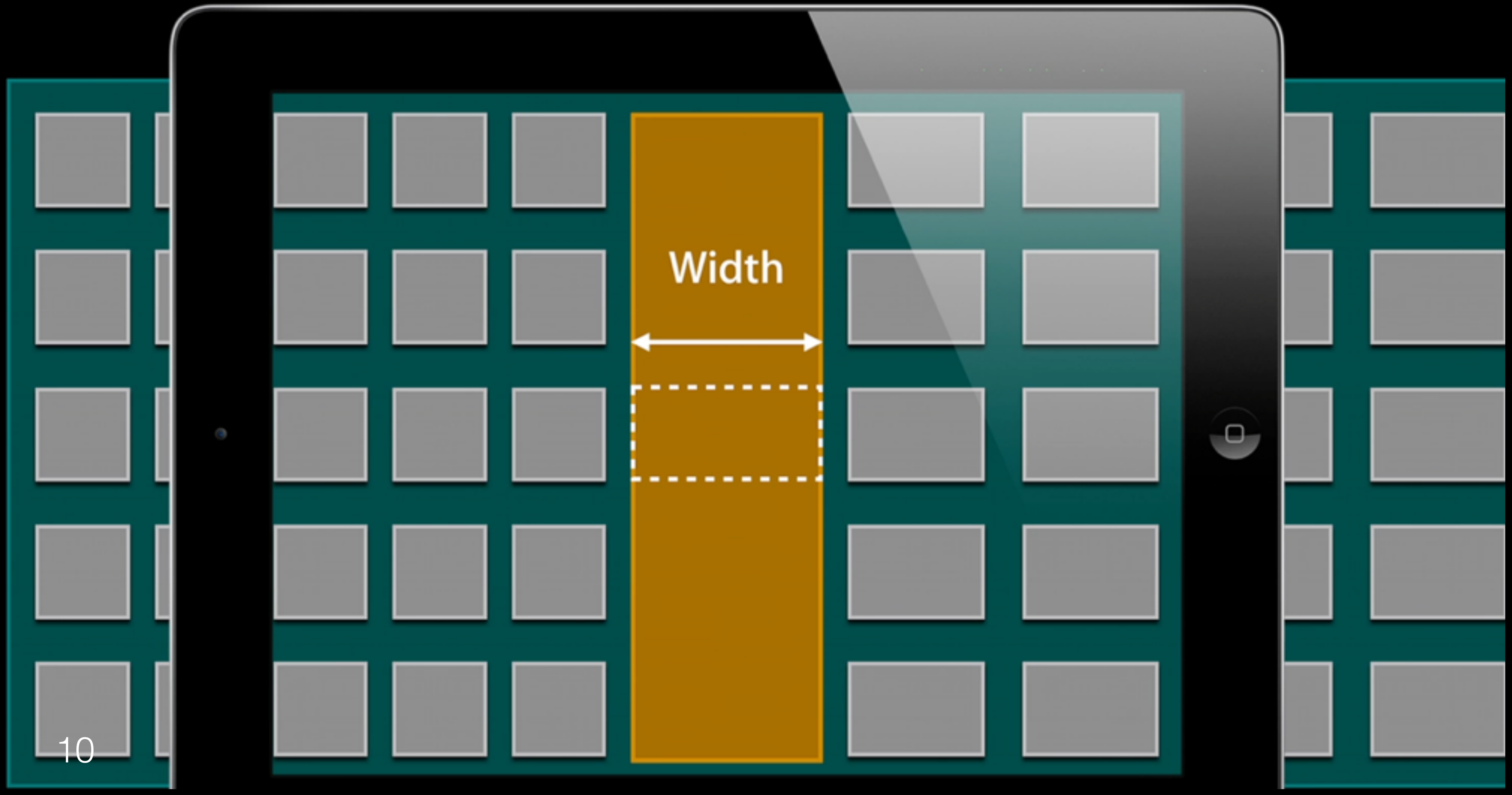
- Same with spacing between individual items:



Scrolling Direction

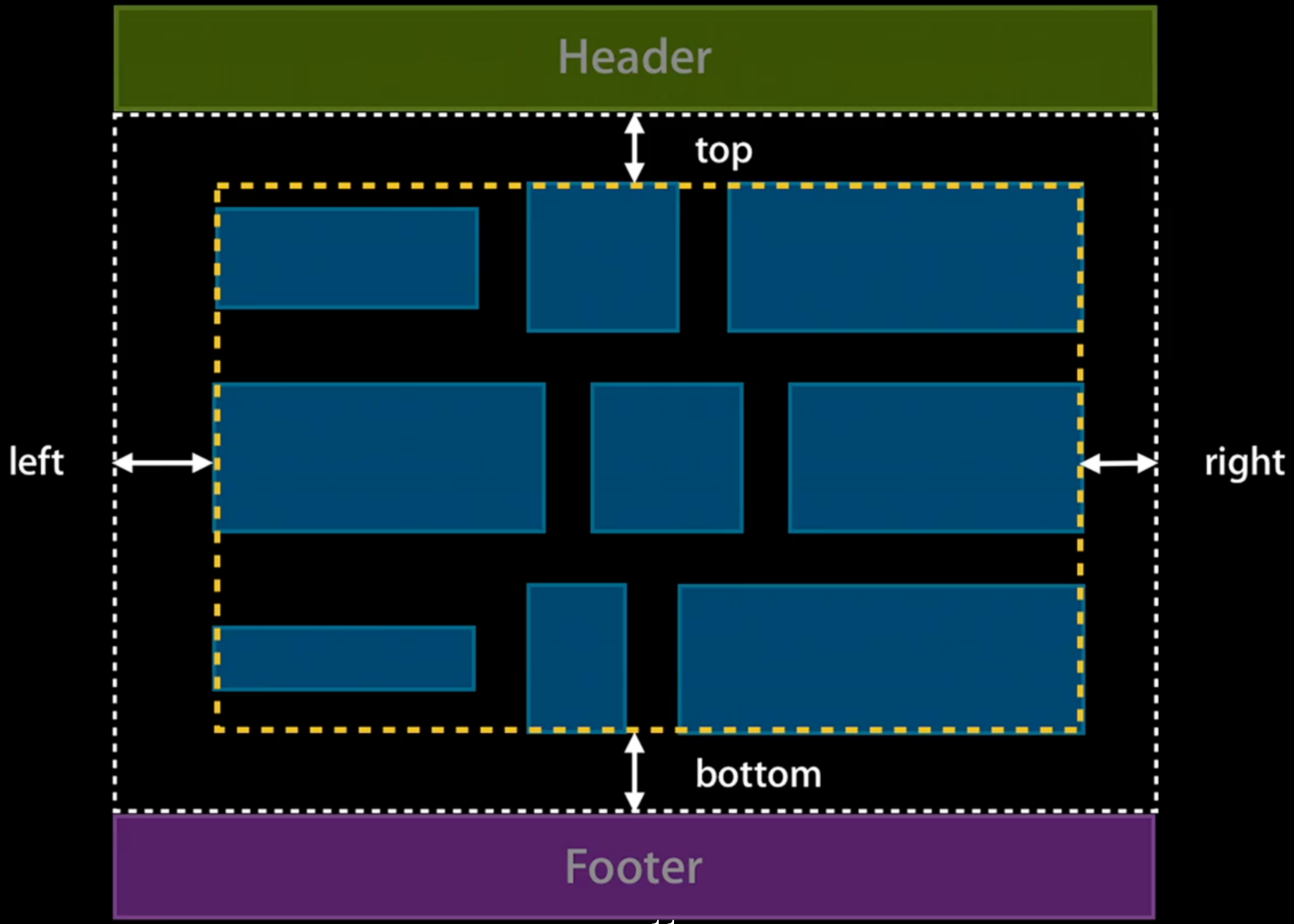
- The scroll direction of your flow layout can defines the base behavior of your entire flow layout
- Dictates the dimensions of the header and footer views:





Section Insets

```
inset = UIEdgeInsetsMake(top, left, bottom, right)
```



Changing the layout

- When you want your layout to change, you need to **invalidate** your layout.
- Call **`invalidateLayout`** to trigger a layout update.
- You can use **`performBatchUpdates:completion:`** and anything you change inside the update block will invalidate the layout AND cause awesome animations.
- Whenever the bounds of the collection view changes, the layout is invalidated (rotation, scrolling)

When to go custom?

- If you are constantly changing the location of all the cells.
- If you want finer grain control over layout attributes of each cells
- If your layout isn’t some sort of grid, implement a custom layout

Required overrides on UICollectionViewLayout

- collectionViewContentSize: Returns the width and height of the collection view's contents. **This is the entire size of the collection view's content, not just what is visible**.
- layoutAttributesForElementsInRect: Returns the layout information for the cells and views that intersect the specified rectangle. **In order for the collection view to know which attribute goes to cells or views, you must specify the elementCategory on the attribute (cell, supplementary view, decoration view)**
- layoutAttributeForItemAtIndexPath: Use this method to provide layout information for your collection view's cells. Do not use this method for supplementary or decoration views.
- layoutAttributesForSupplementaryViewOfKind:atIndexPath: & layoutAttributesForDecorationViewWithReuseIdentifier:atIndexPath:

UICollectionViewLayout Order of Operations

1. prepareLayout
2. collectionViewContentSize
3. layoutAttributeForElementsInRect (which will probably call layoutAttributesForIndexPath)

If the layout is invalidated, prepareLayout is called and this cycle is repeated.

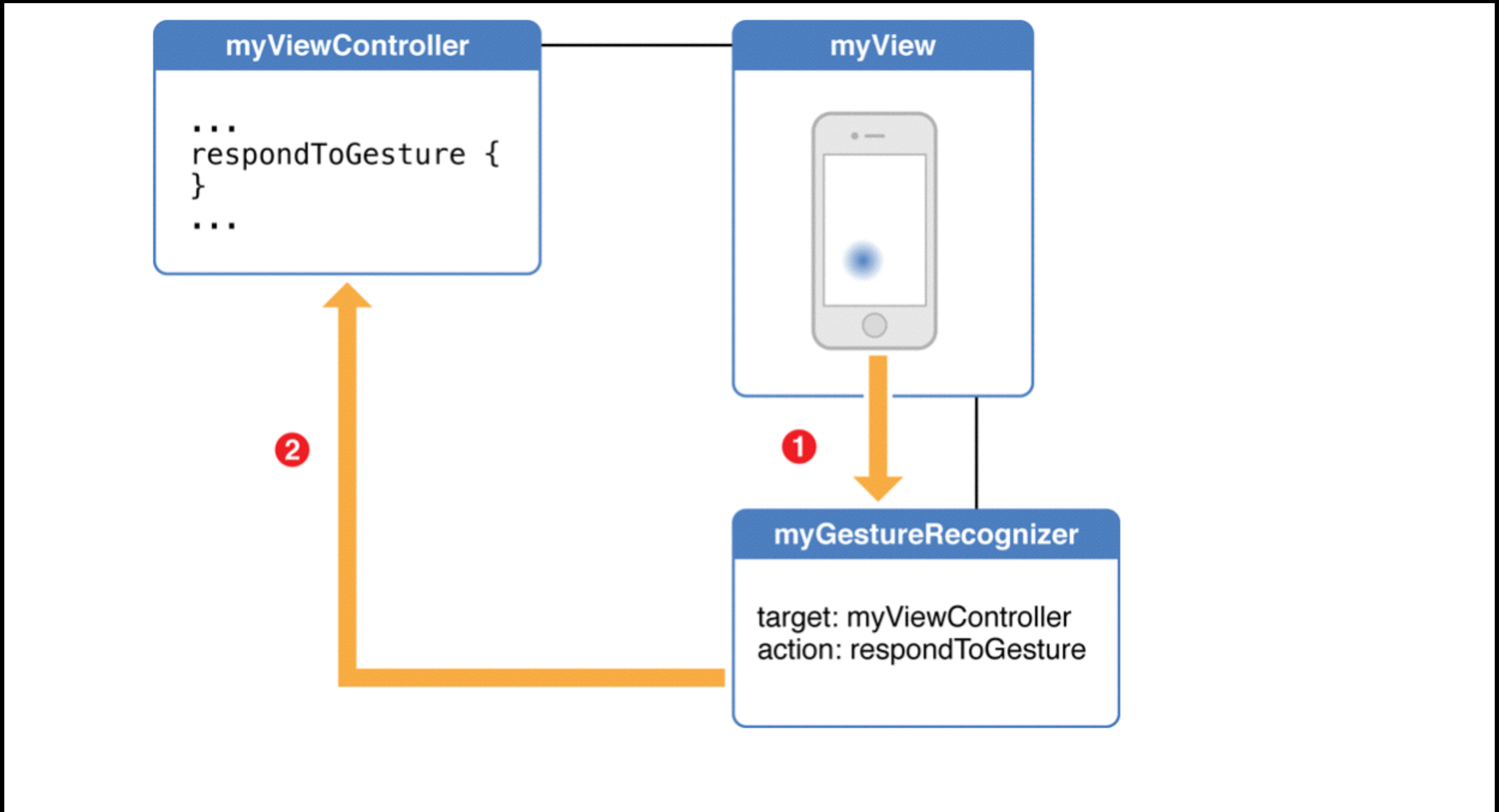
Demo

Gesture Recognizers

Gesture Recognizers

- Convert low level event handling code into higher level actions
- Are attached to views, using `addGestureRecognizer`
- Have an **action** function that is triggered when gesture occurs.
- The target is usually the view’s view controller.
- Follow the target/action patterns as buttons

Gesture Recognizers



Predefined vs Custom Gesture Recognizers

- UIKit has predefined gesture recognizers to fit most needs
- Using a built-in recognizers is preferred to writing your own
- If your app needs to recognize a custom gesture, like a figure 8 or checkmark, you will need to implement your own custom gesture recognizer.

Built-in Gesture Recognizers

- **UITapGestureRecognizer** - any number of taps
- **UIPinchGestureRecognizer** - pinch in and out for zooming
- **UIPanGestureRecognizer** - panning or dragging
- **UISwipeGestureRecognizer** - swiping in any direction
- **UIRotationGestureRecognizer** - finger moving in opposite direction
- **UILongPressGestureRecognizer** - touch and hold for a certain amount of time
- Refer to the HIG for recommended usage for each type of gesture

Discrete vs Continuous Gestures

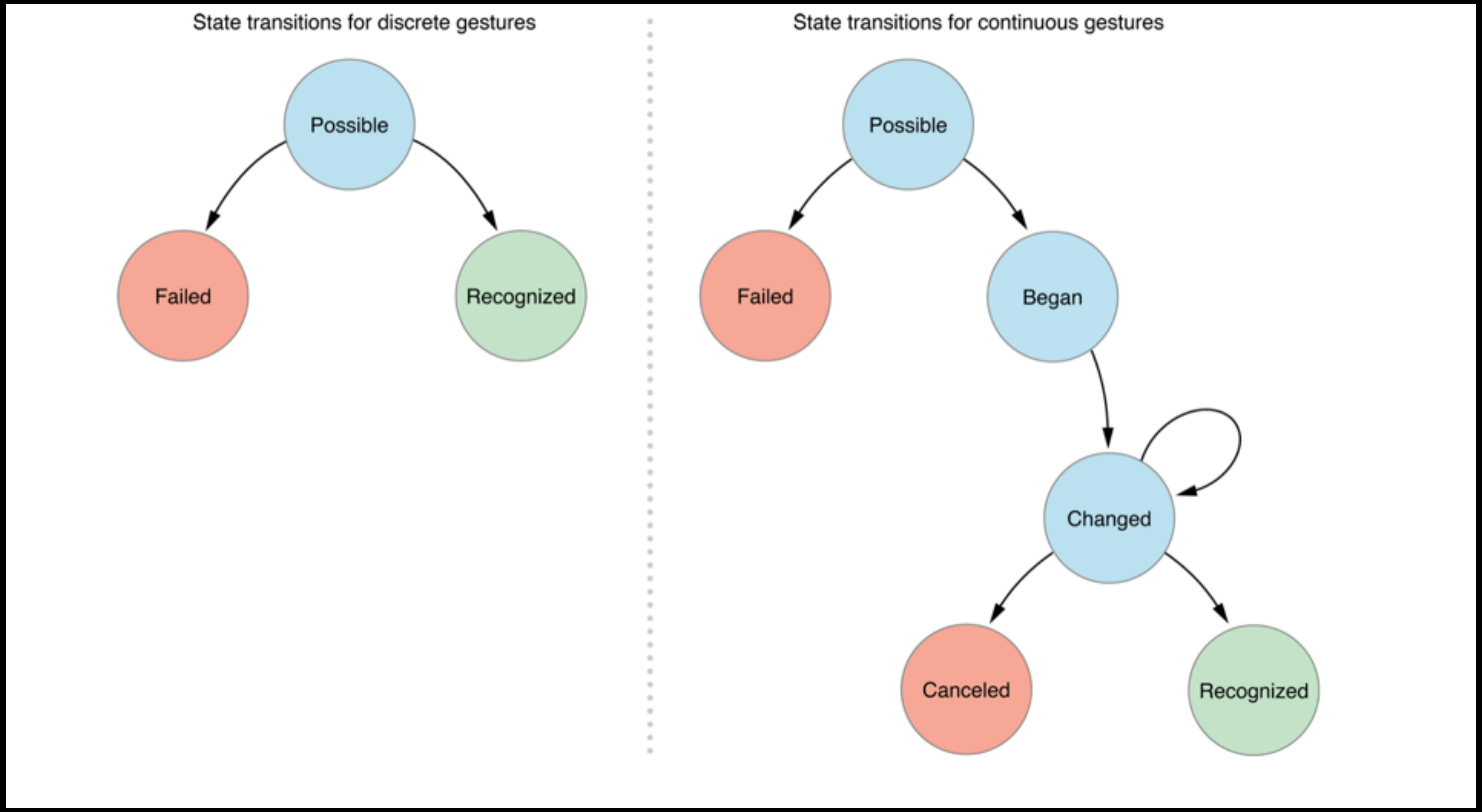
- Gestures are either discrete or continuous.
- A discrete gesture only happens once. Like a tap.
- A continuous gesture takes place over time, like a pan.
- If it is discrete, only one action message is sent. If it continuous, many action messages are sent until the gesture is over.

Gesture Recognizer Setup

1. Create and configure a gesture recognizer instance. Either in code or in storyboard. If its storyboard, this includes step 2.
2. Attach the gesture recognizer to a view.
3. Implement the action method that handles the gesture.

Gesture Recognizer State

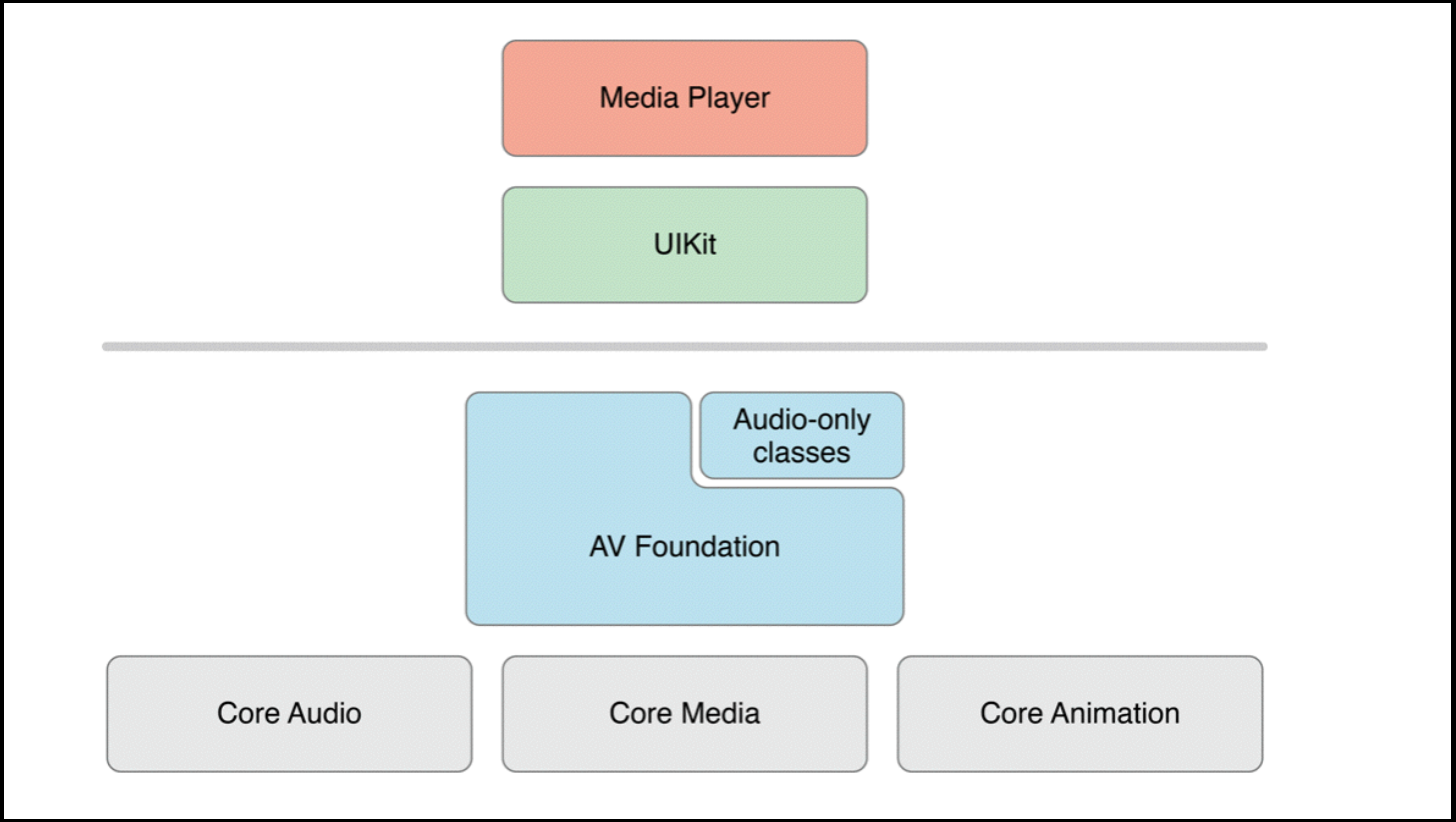
- Gesture Recognizers transition from one state to another in a predefined way.
- From each state, they can move to one of several possible next states based on whether they meet certain conditions:


State transitions for discrete gestures
Possible
Failed
Recognized
State transitions for continuous gestures
Possible
Failed
Began
Changed
Canceled
Recognized

Demo

AVFoundation

AVFoundation

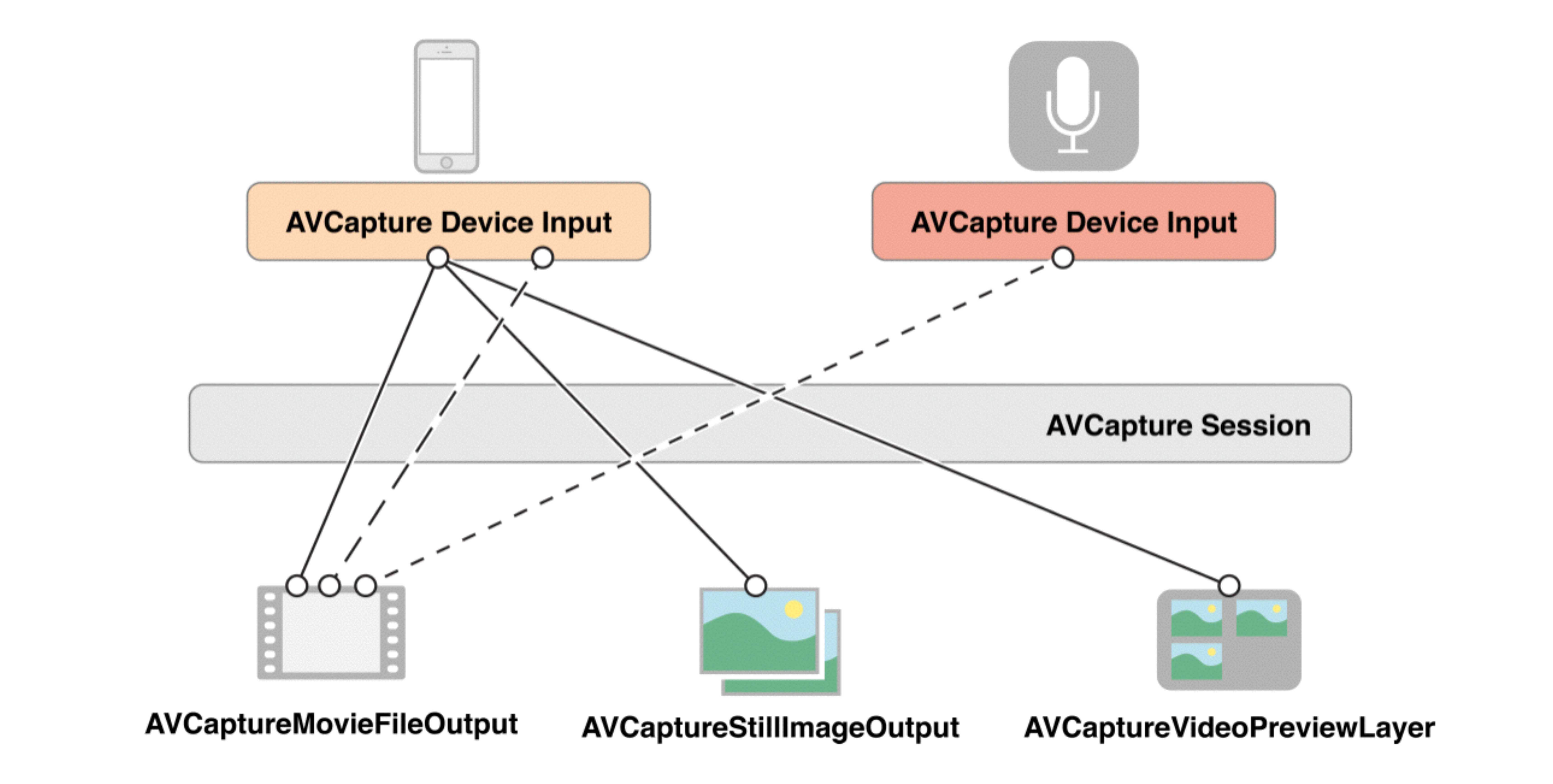


- A framework used to play and create time-based audiovisual media.

AVFoundation Assets

- The primary model class that AVFoundation uses to represent media is AVAsset.
- AVAsset is an aggregated representation of one or more pieces of media data.
- Provides info like the title,duration,natural size,etc.
- Each piece of media data inside the asset is considered a track.
- An asset or track that has been initialized may not ready to be used right away, so the API is highly asynchronous using callbacks.

AVFoundation Capturing



- To manage the capture of media, you create objects to represent inputs and outputs.
- You then use an instance of AVCaptureSession to coordinate the flow of data between them.

AVFoundation Capturing

- You will need the following objects setup for capture:
  - An an instance of AVCaptureDevice to represent the input device, like the phones camera or mic.
  - An instance of AVCaptureInput to configure the ports from the input device.
  - An instance of AVCaptureOutput to manage the output to a movie file or still image.
  - An instance of AVCaptureSession to coordinate the flow of data from input to output.
  - An instance of AVCaptureVideoPreviewLayer to show the user a preview of what the camera is recording.

AVCaptureDevice

- Represents a physical capture device and the properties associated with the device.
- An instance of AVCapture devices allows you to configure the underlying device.
- Provides input data to an AVCaptureSession

AVDeviceInput

- A concrete sub-class of AVCaptureInput.
- Used to capture data from an AVCapture Device.
- initWithDevice:Error:

AVCaptureSession

- You use this class to coordinate the flow of data from input devices to outputs.
- use addInput: and addOutput: methods to add those streams.
- tell a session to startRunning() when everything is configured.
- Run all session setup and startRunning on a background queue because it is potentially blocking and we want to keep the interface responsive to the user.

AVCaptureConnection

- Represents the connection between an input device and capture output running on a capture session.
- Has an inputPorts property that returns an array of all the ports data is streaming through.
- Find the port with mediaType AVMediaTypeVideo, this is your camera.
- Use that connection as the first parameter on the method captureStillImageAsynchronouslyFromConnection() on your AVCaptureStillImageOutput instance.

Demo

Swift Generics

Swift Generics

- “Generic Code enables you to write flexible, reusable functions and types that can work with any type, subject to requirements that you define”
- Generics help us avoid code duplication
- Much of Swift’s standard library is built with generics.
- Swift Arrays and Dictionaries are built with generics

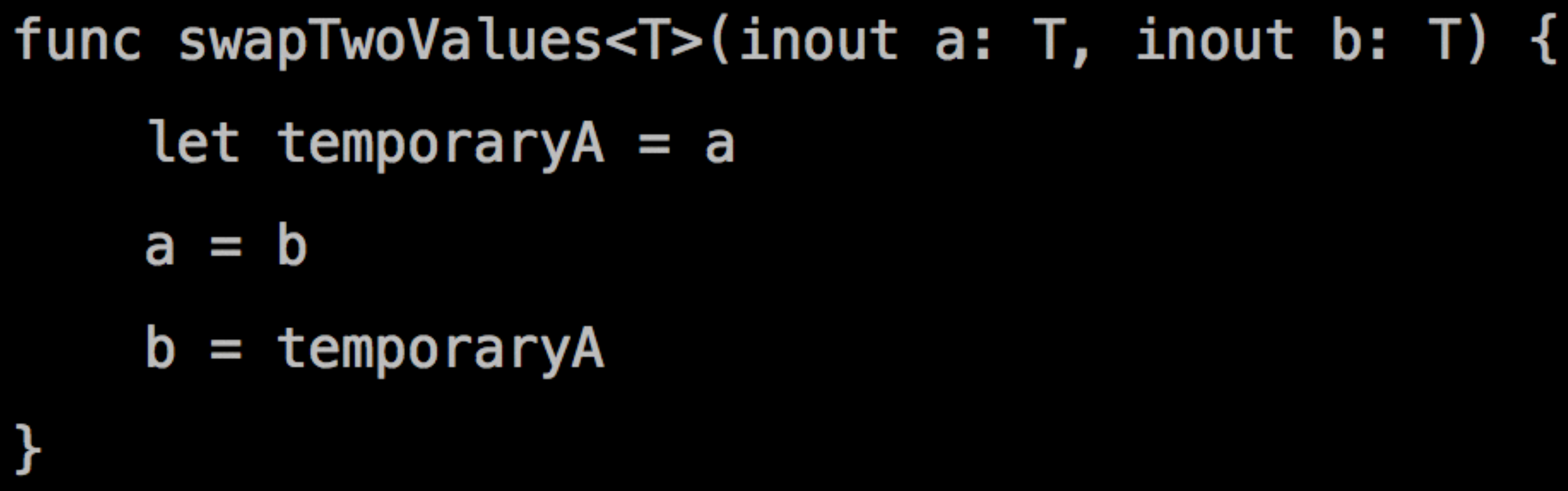
Apple Examples of duplicate code

```
func swapTwoInts(inout a: Int, inout b: Int) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

```
func swapTwoDoubles(inout a: Double, inout b:
        Double) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

```
func swapTwoStrings(inout a: String, inout b:
        String) {
    let temporaryA = a
    a = b
    b = temporaryA
```

And the generic function

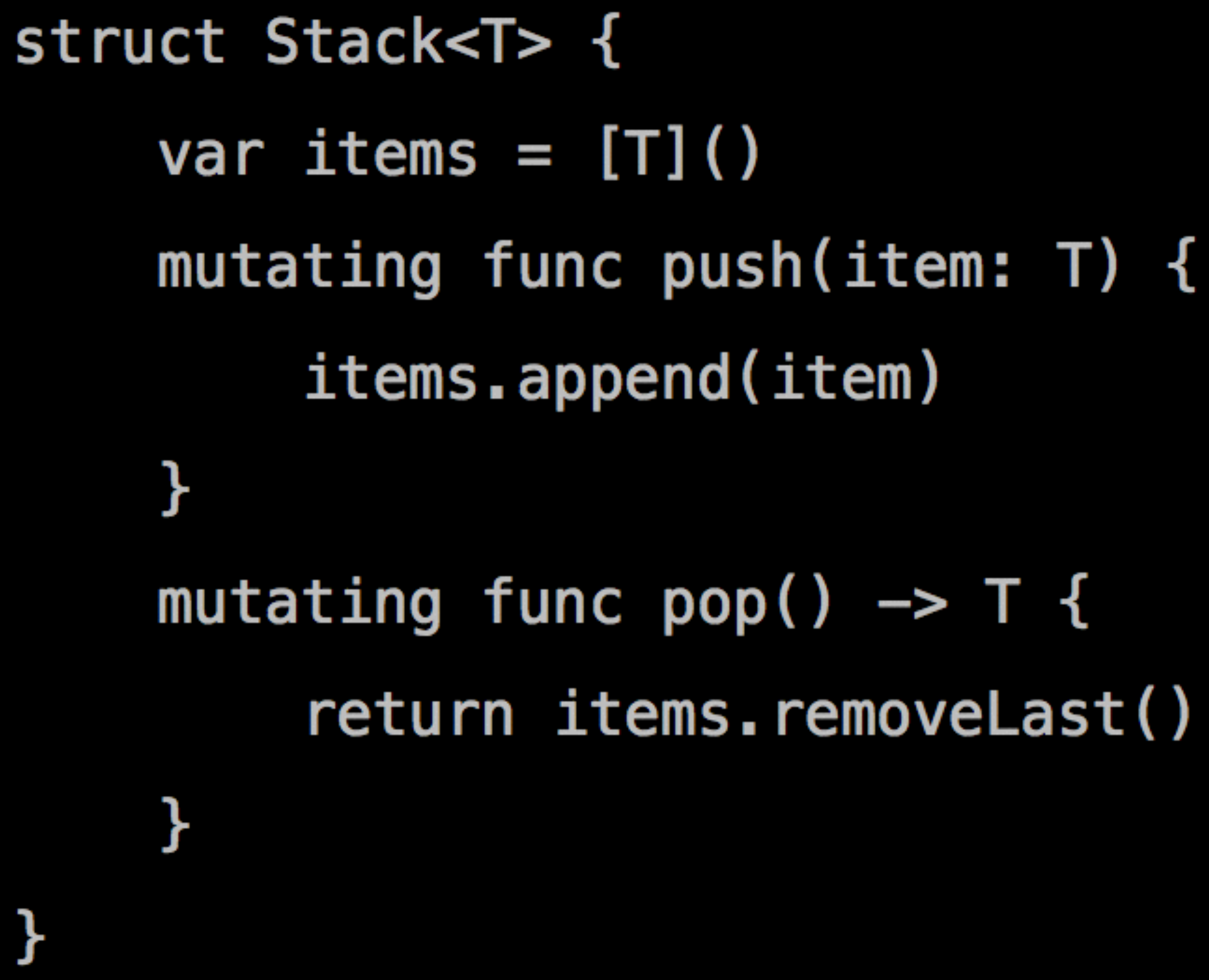


```
func swapTwoValues<T>(inout a: T, inout b: T) {
    let temporaryA = a
    a = b
    b = temporaryA
}
```

- The big difference here is in the functions signature.
- <T> denotes a placeholder type name.
- The placeholder name doesn't say anything about what type T is, just that both parameters are of type T.
- The actual type to use for T wont be determined until the function is actually called.

generic types

- In addition to generic functions, Swift lets you create your own generic types.
- Here is a generic stack:

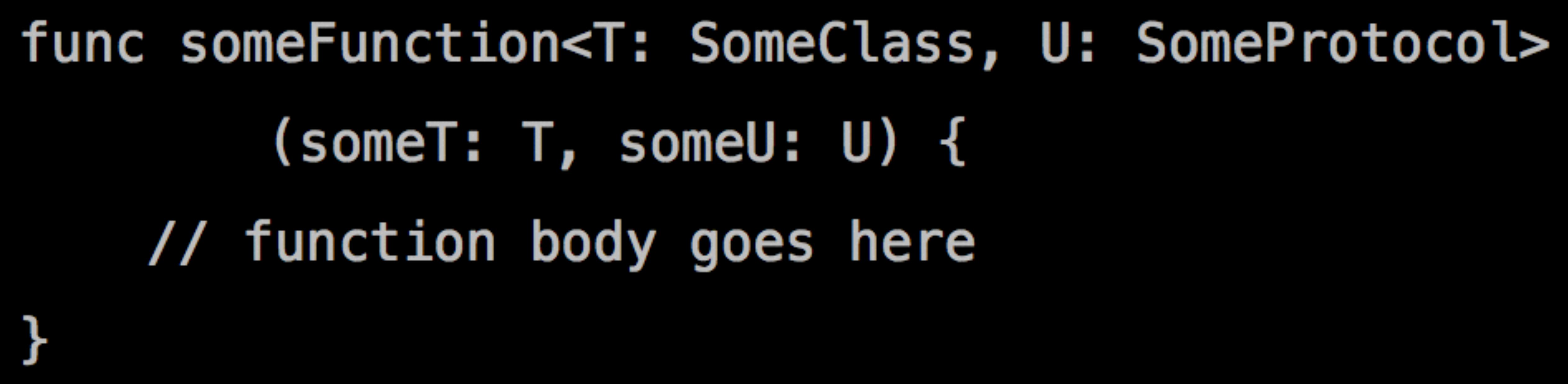


```
struct Stack<T> {
    var items = [T]()
    mutating func push(item: T) {
        items.append(item)
    }
    mutating func pop() -> T {
        return items.removeLast()
    }
}
```

type constraints

- It is sometimes useful to enforce certain type constraints on the types that can be used with generic functions and generic types.
- With type constraints you can specific that a type parameter must inherit from a certain class or conform to a protocol.
- Swift Dictionaries puts a type constraint on the types that can be used for keys. The keys must be hashable, so they must conform to the Hashable protocol.

type constraints syntax



```
func someFunction<T: SomeClass, U: SomeProtocol>
        (someT: T, someU: U) {
    // function body goes here
}
```

- In the function above, the first generic parameter T must inherit from the class SomeClass.
- The 2nd generic parameter U must conform to protocol Some Protocol

Demo

Linked Lists

A Linked List is…

A group of nodes linked together in a sequence.

Can think of it as a line of railroads cars

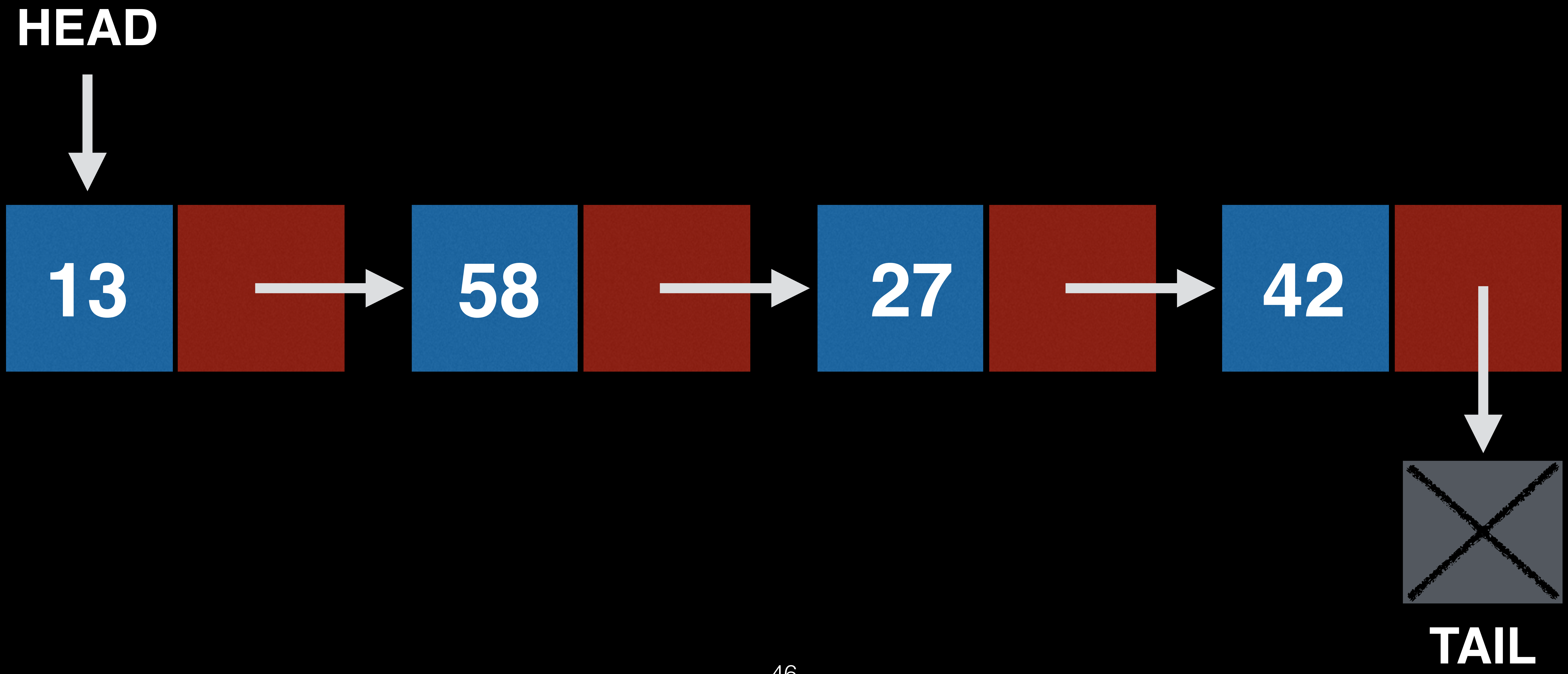
Each node stores some piece of **data**

Each node has a pointer to the **next** node

A favorite topic in Interviews… (and Queues!)

To Find a Node…

Use the next pointer to walk the List



Data Access

- No direct access to data, must look through nodes
- Operations at the front are quick, at the back are slow
- Accessing data further down the list requires a longer walk
- This “linear” time cost is more expensive than “constant” time of an Array
- Basically, Linked Lists slower than Arrays, $O(n) > O(1)$