

курс

development

12 недель

iOS: разработка приложений с 0

r_d

iOS

программа курса

20 занятий

1

Swift: начало

2

ООП: основы

3

Создание iOS-приложения в Xcode

4

Создание
интерфейса iOS-
приложения

5

Динамические
интерфейсы, часть 1

6

Динамические
интерфейсы, часть 2

iOS

программа курса

20 занятий

7

Динамические
интерфейсы, часть 3

8

Навигация в
приложении, часть 1

9

Навигация в
приложении, часть 2

10

Анимации в iOS

11

Работа с памятью
в iOS

12

Многозадачность
в iOS, часть 1

iOS

программа курса

20 занятий

13

Многозадачность в
iOS, часть 2

14

Дебаг
iOS-приложения

15

Тестирование

16

Хранение данных
в приложении

17

Работа с сетью
в приложении

18

Сборка приложения

iOS

программа курса

20 занятий

19

Современные
архитектуры для
iOS приложений

20

Защита курсовых
проектов

Создание интерфейса iOS приложения

- Создание интерфейса в iOS приложении (demo)
- Основные компоненты: UIView и UIControl
- Контейнеры компонентов: UIViewController
- Storyboard и основы Auto Layout
- [OPTIONAL] Ветвление в git

Создание первого экрана в iOS приложении

Demo Time 🍰 🍰 🍰

UIView. Базовый элемент интерфейса

“An object that manages the content for a rectangular area on the screen.” (c)

UIView

- Класс `UIView` — базовый класс для всех остальных компонентов пользовательского интерфейса в iOS приложении.
- Можно использовать как наследников класса `UIView`, так и непосредственно `UIView` при разработке интерфейсов. Например, `UIView` удобно использовать, если нам нужно закрасить прямоугольный фон цветом. Или нарисовать что-то более сложное.
- Класс `UIView` имеет набор важных функций для работы любого iOS приложения.

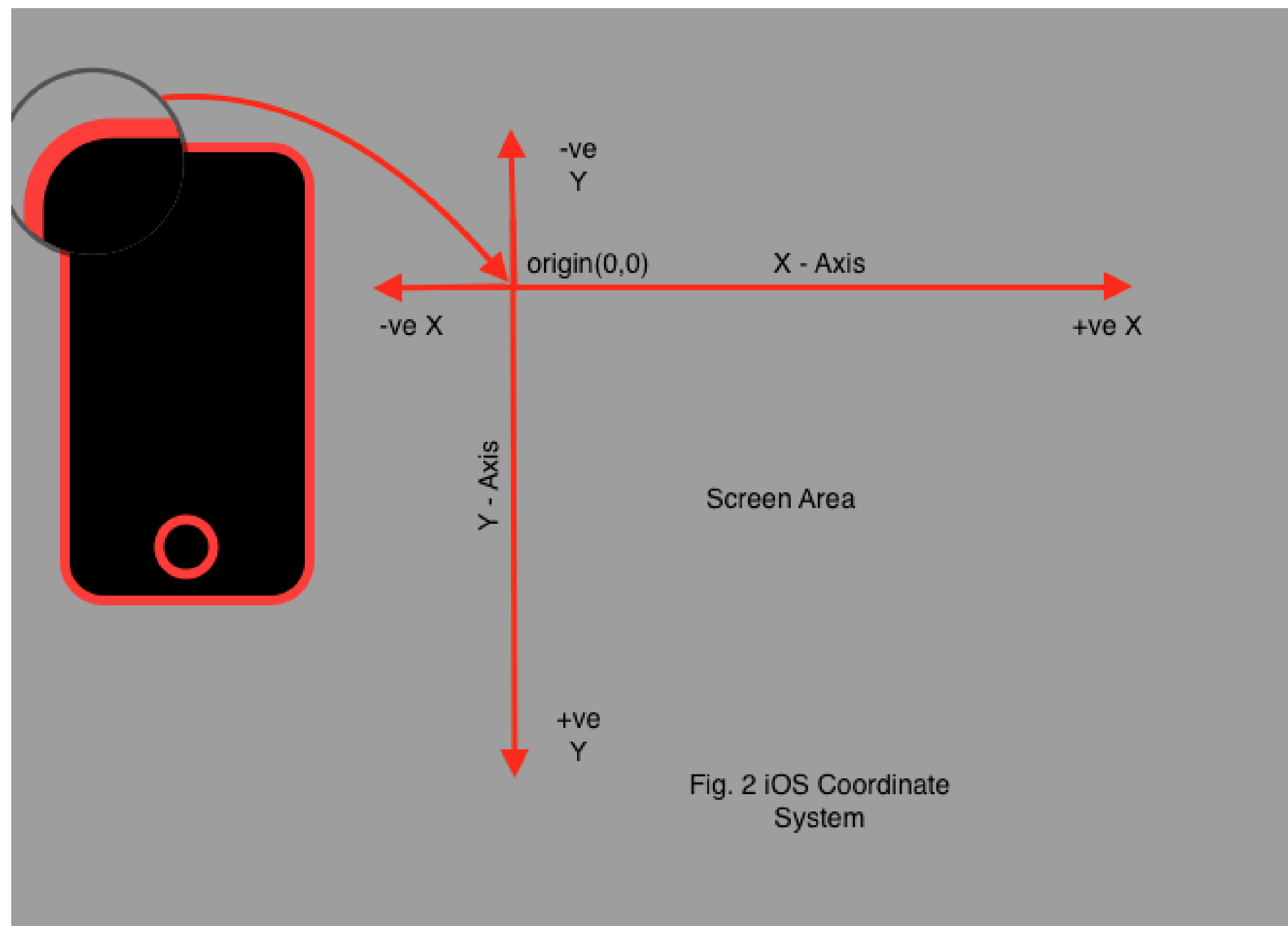
UIView

Рисование и анимация (drawing and animation)

- Каждый объект класса UIView отрисовывает какой-то графический контекст в прямоугольной области, заданный двумя координатами (origin и size).
- Графические средства для отрисовывания чего-либо в UIView находятся в 2-х фреймворках: UIKit и Core Graphics.
- Координаты описывают верхний левый угол и нижний правый соответственно.
- Такое специфическое расположение начальной и конечной точки обусловлено тем, что декартова система координат (ось X и Y) располагаются вправо и вниз от нулевой координаты.

UIView

Рисование и анимация (drawing and animation)



UIView

Рисование и анимация (drawing and animation)

- Элементы типа UIView могут быть анимированы на экране.
- Анимация элементов типа UIView достигается за счёт анимации некоторых свойств класса UIView. Пример: frame, bounds, center, backgroundColor, alpha.
- Любую работу с элементами типа UIView нужно выполнять на основной очереди (main queue). С очередями и многопоточностью мы будем разбираться в лекциях 12 и 13.

UIView

Расположение на экране. Управление subviews

- Любой объект типа UIView может иметь от нуля и больше вложенных объектов класса UIView. Это значит — мы можем строить иерархию объектов класса UIView (дерево объектов).
- Любая view может управлять размером и задавать позицию своим subviews. Мы можем создавать определенные ограничения для subview для правильной отрисовки и расположения при разных размерах экрана.

UIView

Расположение на экране. Управление subviews

- UIKit предоставляет нам специальный мат.механизм, который называется Auto Layout для удобного размещения наших views на экране относительно других таких views.
- Один из наследников класса UIView, класс UIWindow. Корневой объект UIWindow находится по умолчанию в классе SceneDelegate. Этот объект является первым (корневым) в иерархии всех графических объектов класса UIView. Размер нашего объекта UIWindow всегда равен размеру экрана конкретного iPhone, на котором запущено приложение.

UIView

Обработка входящих событий (event handling)

- Класс UIView является наследником класса `UIResponder`. Класс `UIResponder` отвечает за обработку таких событий в системе, как пользовательские нажатия (touches), свайпы (swipes) и некоторые другие.
- Мы можем создавать объекты для распознавания жестов (gesture recognizers), чтобы определять и обрабатывать нужные виды нажатий в нужных нам местах. Каждый объект класса UIView может с помощью таких распознавателей жестов обрабатывать жесты.
- Есть нюанс в иерархии объектов :) Но это вам уже на самостоятельный разбор.

UIViewController

“An object that manages a view hierarchy for your UIKit app.” (c)

UIViewController

- Класс `UIViewController` является базовым для всех возможных контейнеров наших объектов класса `UIView`.
- Мы редко в коде создаем объекты конкретно класса `UIViewController`. Как правило, мы создаём свой подкласс, в котором уже определяем логику по управлению каким-либо объектом(ами) класса `UIView`.
- Класс `UIViewController` определяет набор методов, которые управляют иерархией всех объектов `UIView`, начиная от корневого `UIView`, который принадлежит нашему подклассу `UIViewController` (а также всех его `subviews`).

UIViewController

Основные функции класса UIViewController

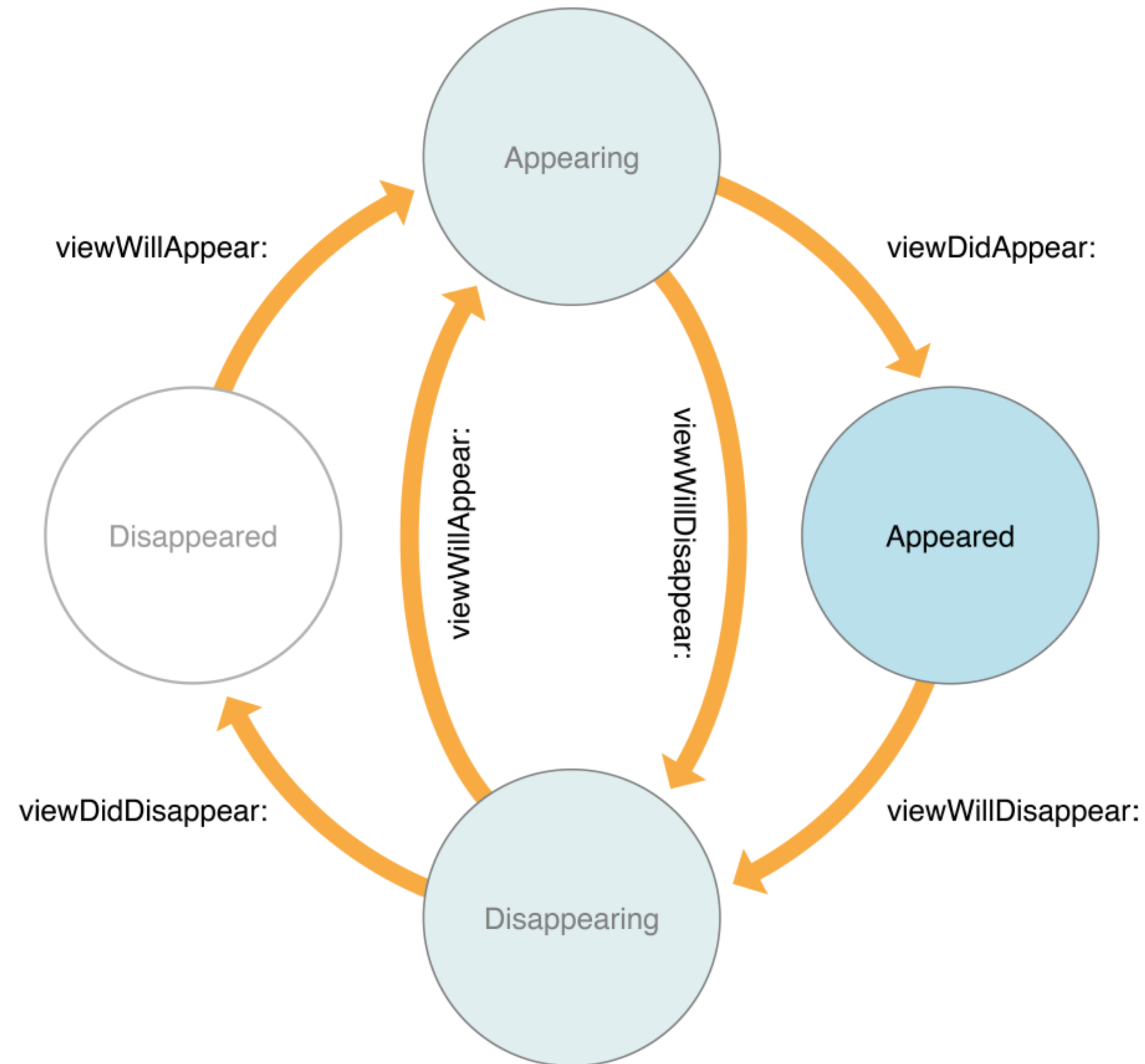
- Обработка входных данных от корневого объекта UIView и всех его subviews. Обновленные данные для view и его subviews устанавливает также наш подкласс UIViewController.
- Обработка событий от нажатий и жестов (UIGestureRecognizer, привязанные к объектам UIView).

UIViewController

Основные функции класса UIViewController

- Управление размером своей view и её subviews. Изменение их размеров, их отрисовка в разные моменты жизни программы и самого экрана (UIViewController lifecycle).
- Координация в приложении. Как правило, любое iOS приложение содержит от нескольких различных экранов с разным пользовательским интерфейсом, между которыми пользователь должен перемещаться.
- Передача данных в приложении. Кроме переходов между экранами, необходимо также как-то передавать данные между экранами (чтобы не дублировать логику работы программы).

UIViewController. Lifecycle



UIViewController

UIViewController как часть другого UIViewController

- Кроме всего прочего, UIViewController является наследником UIResponder (как и класс UIView). Так сделали намеренно.
- Двойное наследование и вложенность объекта UIView в UIViewController даёт последнему возможность очень тесно взаимодействовать со своей view. В таком дуэте очень легко управлять не только самой view, но и всеми её subviews, определяя иерархию компонентов почти любой сложности.

UIViewController

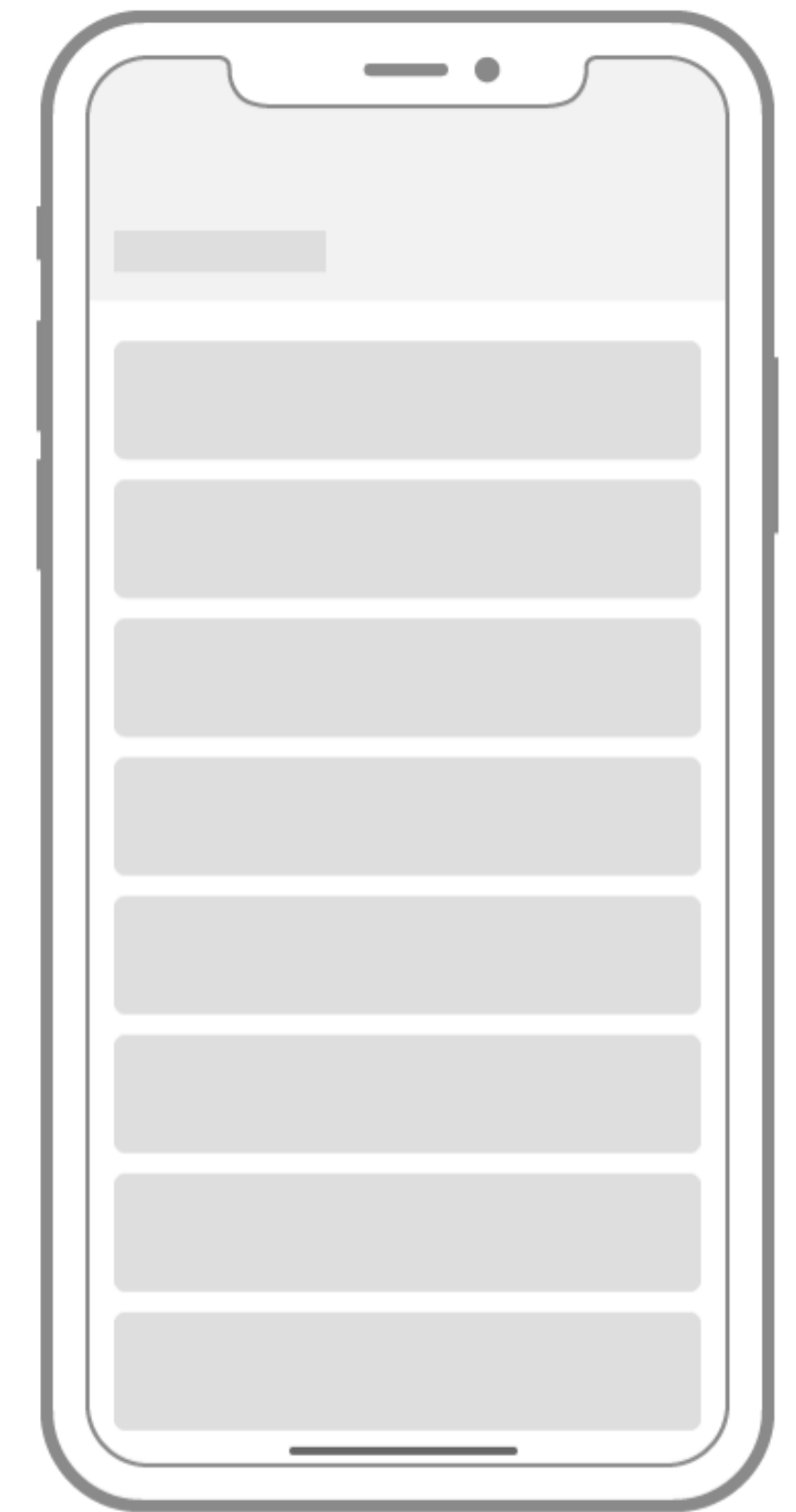
UIViewController как часть другого UIViewController

- Но управлять самим объектом UIView не очень удобно, в сравнении с UIViewController. У UIView довольно ограниченный набор API, с которыми мы можем работать. Поэтому строить интерфейсы на UIView довольно “больно” и трудно.
- Как же тогда нам быть?

UIViewController

UIViewController как часть другого UIViewController

- Как и с UIView, объект UIViewController может содержать в себе несколько вложенных объектов типа UIViewController.
- **Пример:** главный UIViewController содержит в себе объект класса UITableView (мы детально будем разбирать этот объект в след. лекции). У UITableView есть множество subviews — это его ячейки, объекты класса **UITableViewCell**. Каждая такая ячейка может содержать в себе отдельный объект класса UIViewController.
- Такая связь между главным и вложенным объектами класса UIViewController называется “parent-child relationship”.

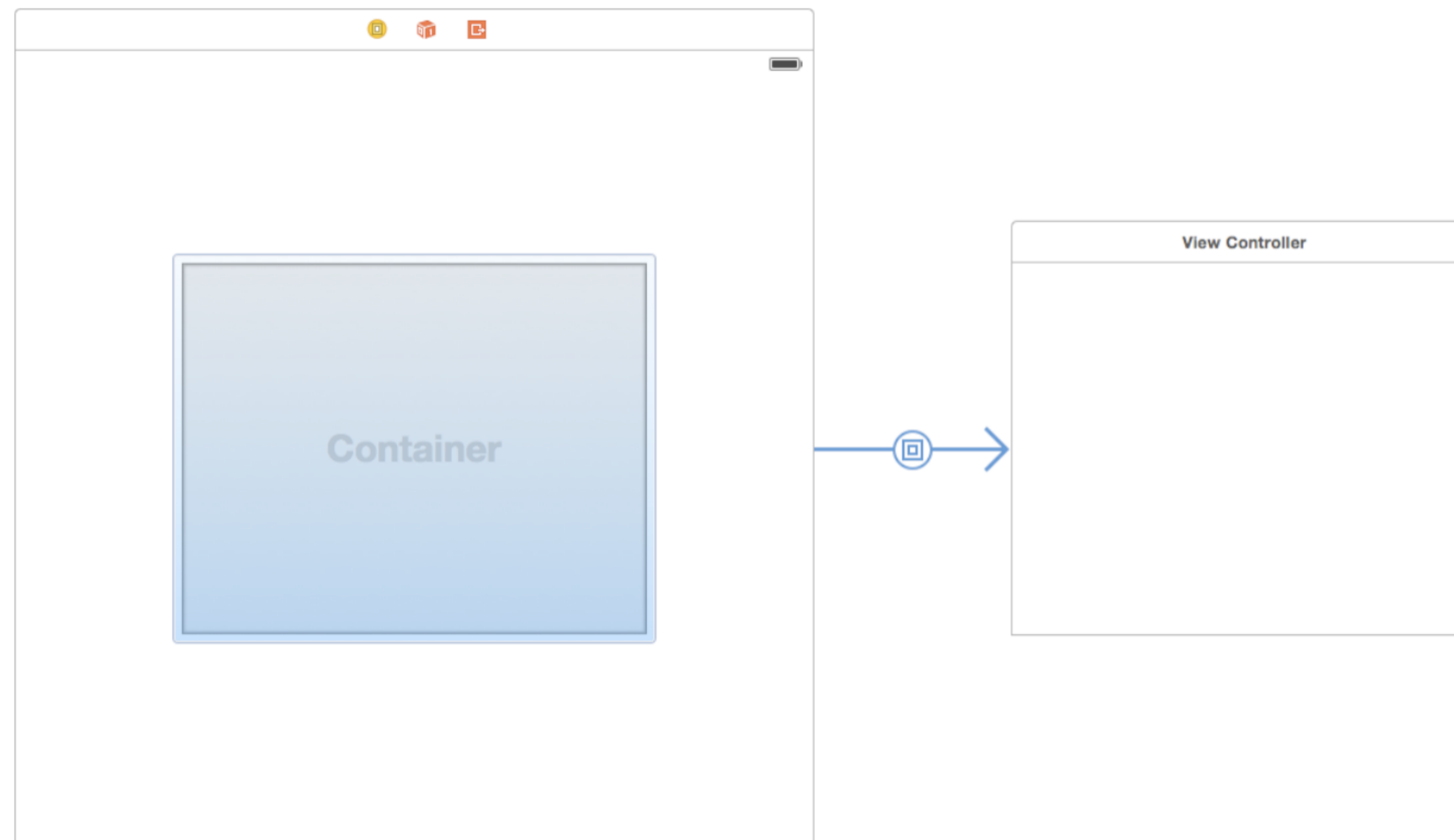


Tabular interface

UIViewController

UIViewController как часть другого UIViewController

Figure 5-3 Adding a container view in Interface Builder



Основы механизма Auto Layout

- Auto Layout — механизм UIKit, который занимается динамическим вычислением позиции и размера (точки origin и size) всех view в иерархии, основываясь на наборе правил или ограничений (constraints).
- При использовании Auto Layout у нас нет нужды считать размеры и позиции всех views на экране.
- Также не нужно создавать одинаковый интерфейс каждого экрана приложения для всех форматов и размеров экрана iPhone / iPad.

UIViewController

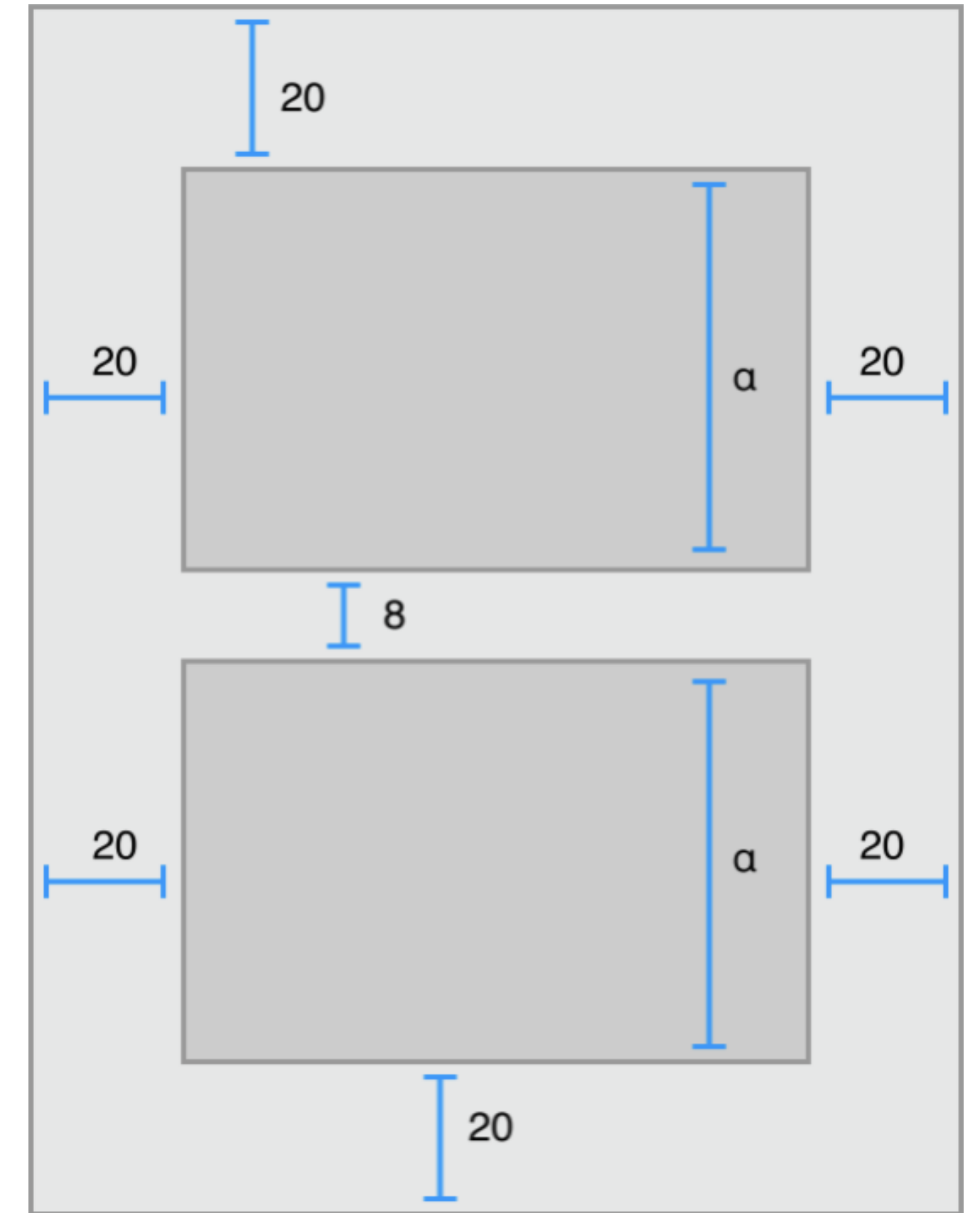
Когда срабатывает механизм Auto Layout?

- Разные размеры устройств по умолчанию
- Изменение ориентации экрана
- Новый контент на экране
- Изменение локализации приложения (на лету или после перезагрузки приложения)
- Изменение размера шрифтов для отдельного приложения или в рамках всей операционной системы

UIViewController

Анатомия ограничений (constraints)

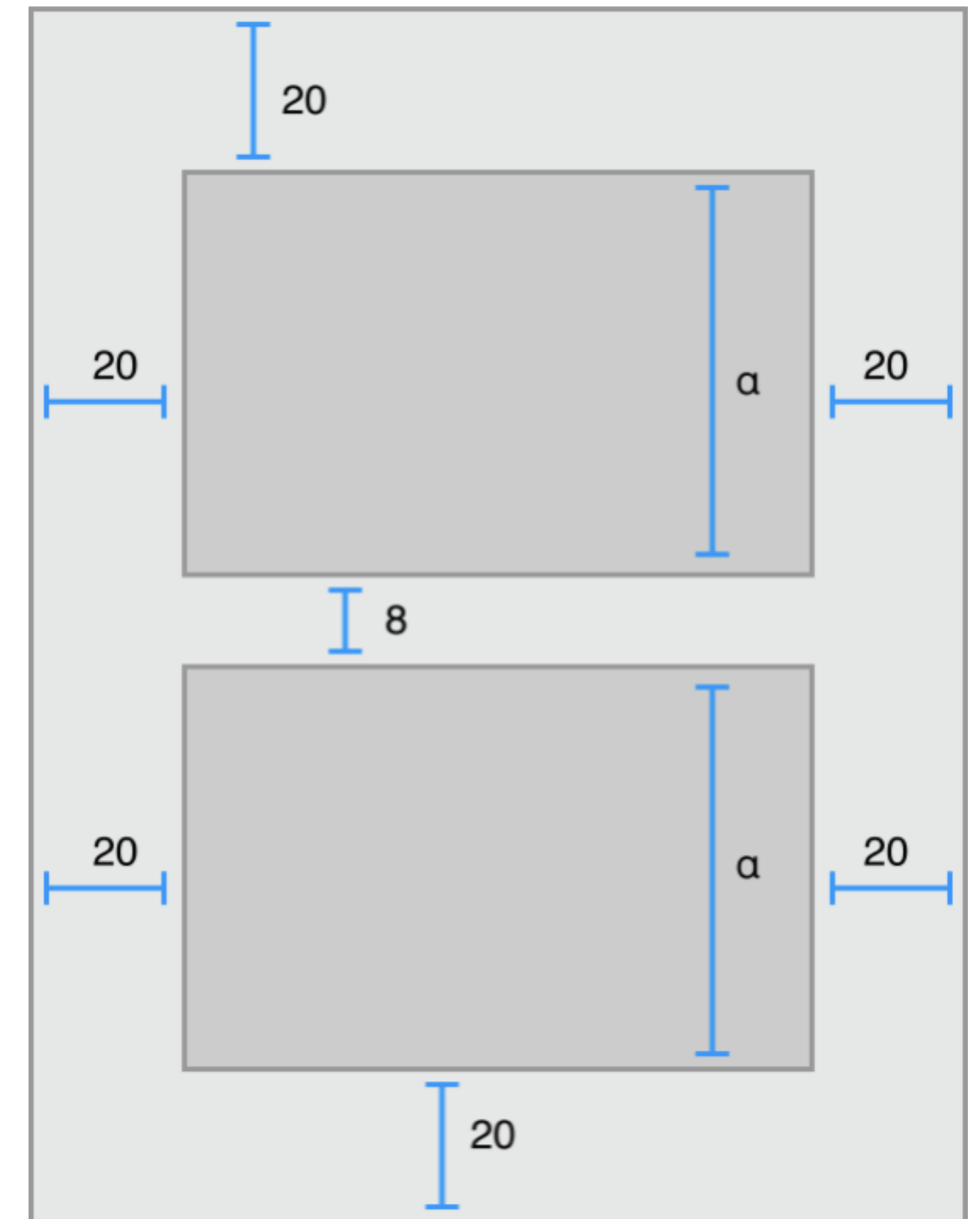
- Идея любого **constraint** — разместить выбранный `view` по одной из координатных осей (X или Y) относительно другого элемента `view`.
- При установке любого `constraint` для вашей `view`, вы думаете не про расположение, а про отношение между вашей `view`, другими `view` в той же иерархии и вашей `Superview` (начиная от `UIWindow`).



UIViewController


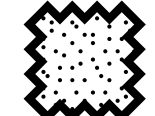
Анатомия ограничений (constraints)

- Любой constraint математически описывается как уравнение
- $Y1 = kY0 + b$
- где $Y0$ — значение координаты одной из осей той view, которую мех-м Auto Layout уже расположил
- k — коэффициент, множитель (multiplier), который позволяет нам пропорционально увеличивать / уменьшать пространство между нашей view и той, относительно которой мы располагаем
- b — константа (constant), также позволяет увеличивать / уменьшать пространство между нашей view и той, относительно которой мы располагаем, но не через операции “+” или “-“, что отличается от умножения :)



UIViewController

Пример

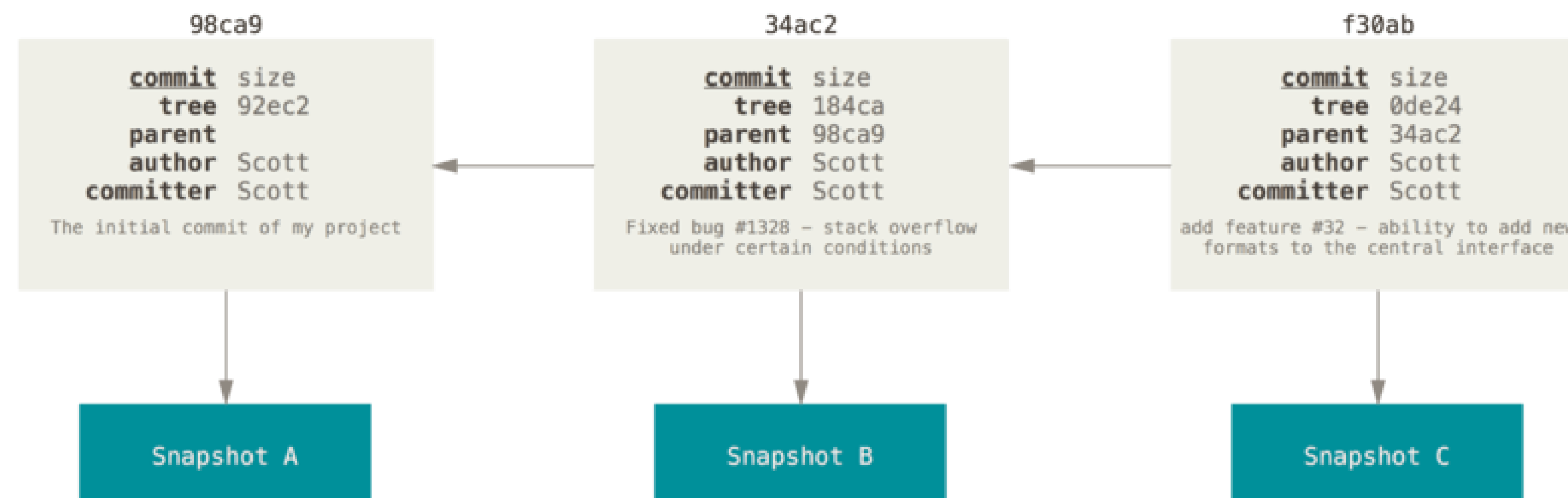
- Расположить элемент UILabel посередине экрана. Для этого нам нужно задать 2 правила: (1) центрировать наш объект по оси X и (2) центрировать по оси Y.
- Давайте посмотрим, как это выглядит в коде  

Ветки в git

- Когда вы делаете коммит, Git сохраняет его в виде объекта, который содержит указатель на снимок (snapshot) подготовленных данных.
- Этот объект также содержит имя автора и email, сообщение и указатель на коммит или коммиты непосредственно предшествующие данному (его родителей): отсутствие родителя для первоначального коммита, один родитель для обычного коммита, и несколько родителей для результатов слияния двух и более веток.
- У каждого коммита есть специальное hash значение, уникальный идентификатор в системе git, по которому любое изменение, сохраненное вами как commit можно будет всегда найти в истории.

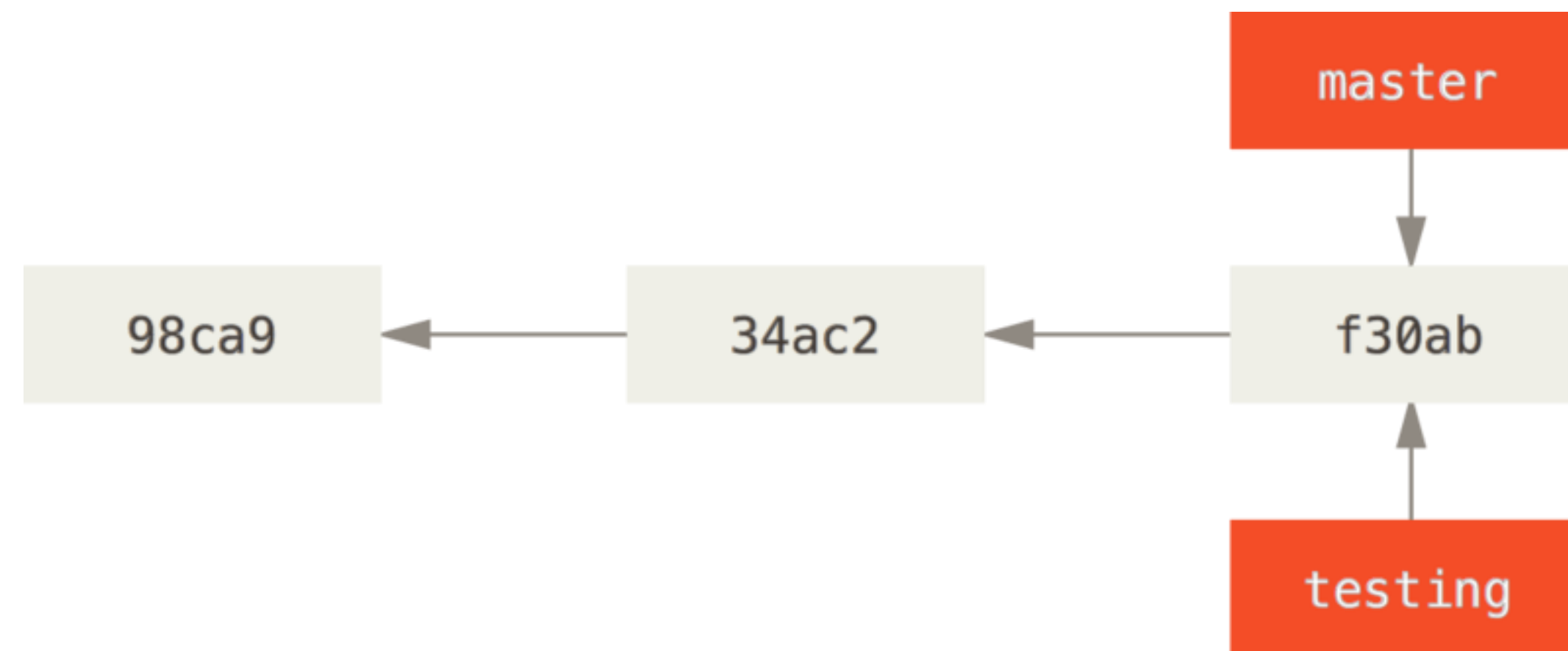
Ветки в git

- Ветка в Git — это простой перемещаемый указатель на один из таких КОММИТОВ.
- По умолчанию, имя основной ветки в Git — master. Как только вы начнёте создавать коммиты, ветка master будет всегда указывать на последний коммит. Каждый раз при создании коммита указатель ветки master будет передвигаться на следующий коммит автоматически.



Ветки в git

- `git branch <название_новой_ветки>` — создание “ветки” (новый указатель для хранения как текущих так и будущих коммитов).
- В результате создаётся новый указатель на текущий коммит.



спасибо

задавайте вопросы