

# Full Stack iOS Entwicklung mit Swift

WPF FSIOS  
Alexander Dobrynin, M.Sc.

# Heute

Layout und Koordinatensystem

CustomViews

Autolayout

StackView

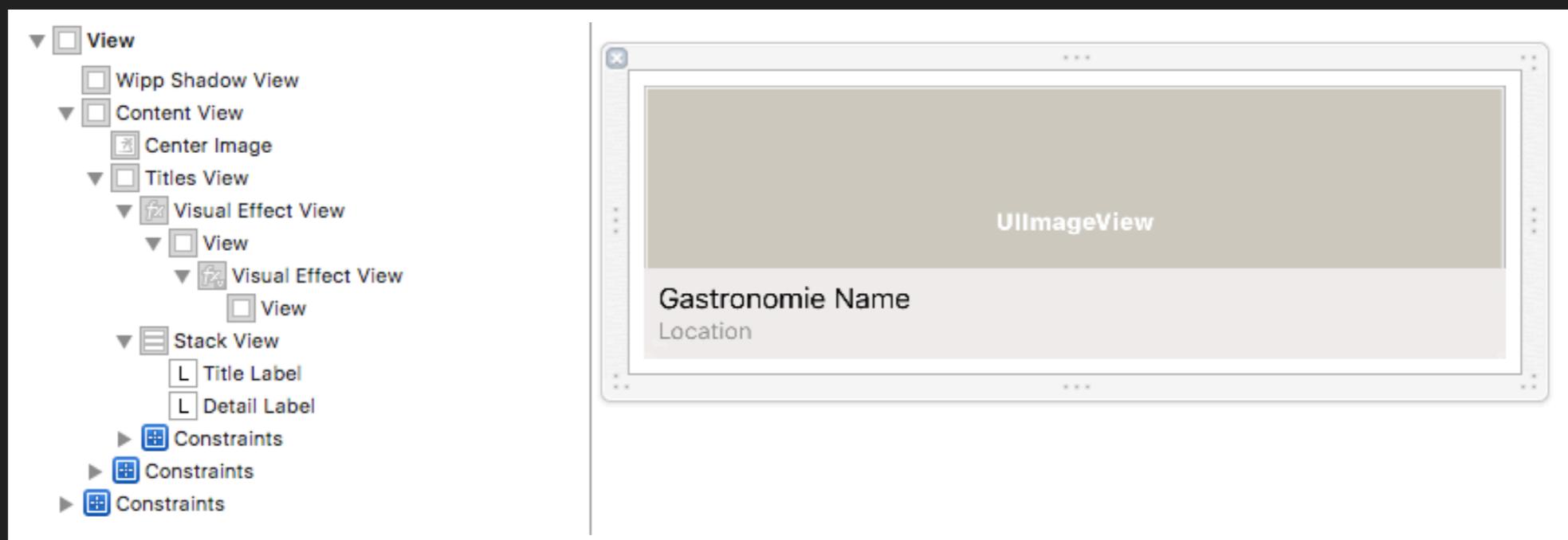
Animationen

Size Classes

Zusammenfassung

# Layout

- Jedes Layout besteht aus einer View Hierarchie
- Jede View hat eine SuperView und 0 bis n SubViews
- Aufgebaut wird eine View Hierarchie entweder im Storyboard per Drag & Drag von UI-Objekten aus der Palette in den ViewController (siehe Outline) oder im Code
  - Der Weg über das Storyboard wird empfohlen, weil die Struktur visuell unterstützt wird
  - Zudem lassen sich Autolayout Constraints im Storyboard veranschaulichen und testen
  - Das Hinzufügen von Views im Code erfordert das Angeben von Koordinaten, ermöglicht aber volle Kontrolle



# Koordinatensystem

- Ursprung (0,0) des Koordinatensystems ist oben Links
  - Einheit sind **Points**, eine Abstraktion über Pixel
    - 1 pt = 1 px bei @1x Devices
    - 1 pt = 2 px bei @2x Devices
    - 1 pt = 3 px bei @3x Devices
  - Jedes Device kennt seinen Scale (`UIScreen.main.scale`), wodurch iOS die UI-Elemente entsprechend an
- Wichtige Datentypen für den Umgang mit dem Koordinatensystem sind
  - `CGFloat(1.0)`
  - `CGPoint(x: 100, y: 100)`
  - `CGSize(width: 50, height: 50)`
  - `CGRect(origin: CGPoint(x: 100, y: 100), size: CGSize(width: 50, height: 50))`
- Die wichtigsten Properties einer View bezüglich des Koordinatensystems sind
  - `var bounds: CGRect // local coordinate system`
  - `var frame: CGRect // super-view's coordinate system`

# Layout und Koordinatensystem

```
class MyViewController : UIViewController {

    override func loadView() {
        let view = UIView()
        view.backgroundColor = .white
        self.view = view // superView
    }

    override func viewDidLoad() {
        super.viewDidLoad()

        let position = CGRect(x: 100, y: 100, width: 250, height: 50)
        let label = UILabel(frame: position) // position of 'label'
        label.numberOfLines = 2
        label.text = "frame:\(label.frame)\nbounds:\(label.bounds)"
        label.textColor = .white
        label.backgroundColor = .red

        view.addSubview(label) // add 'label' as subView
    }

    // remove given 'subView' from superView
    func removeFromSuperview(subView: UIView) {
        subView.removeFromSuperview()
    }
}
```

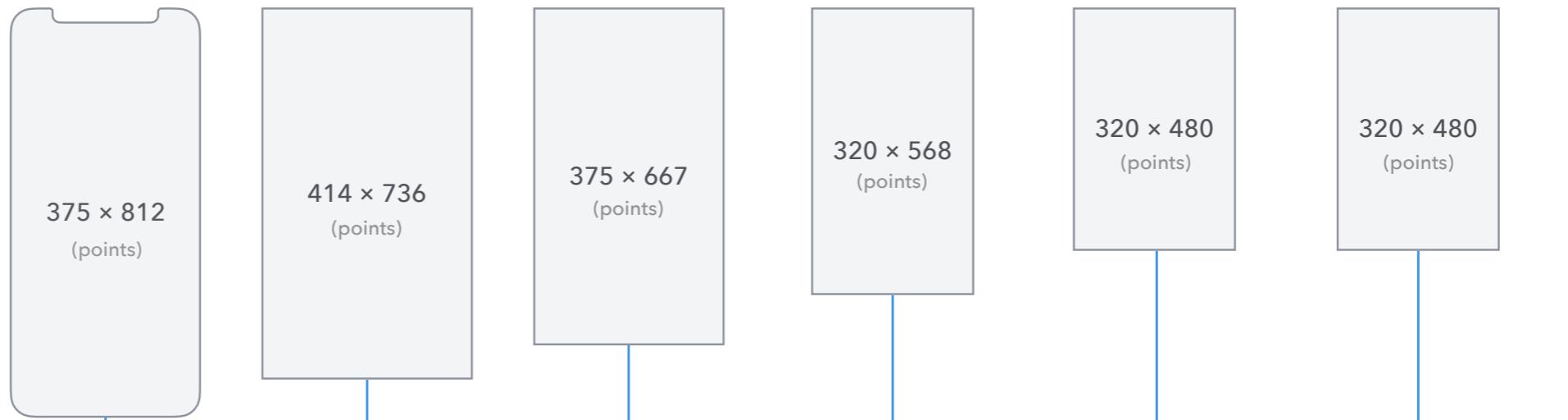
frame:(100.0, 100.0, 250.0, 50.0)  
bounds:(0.0, 0.0, 250.0, 50.0)



### Points

At the beginning, coordinates of all drawings are specified in *points*.

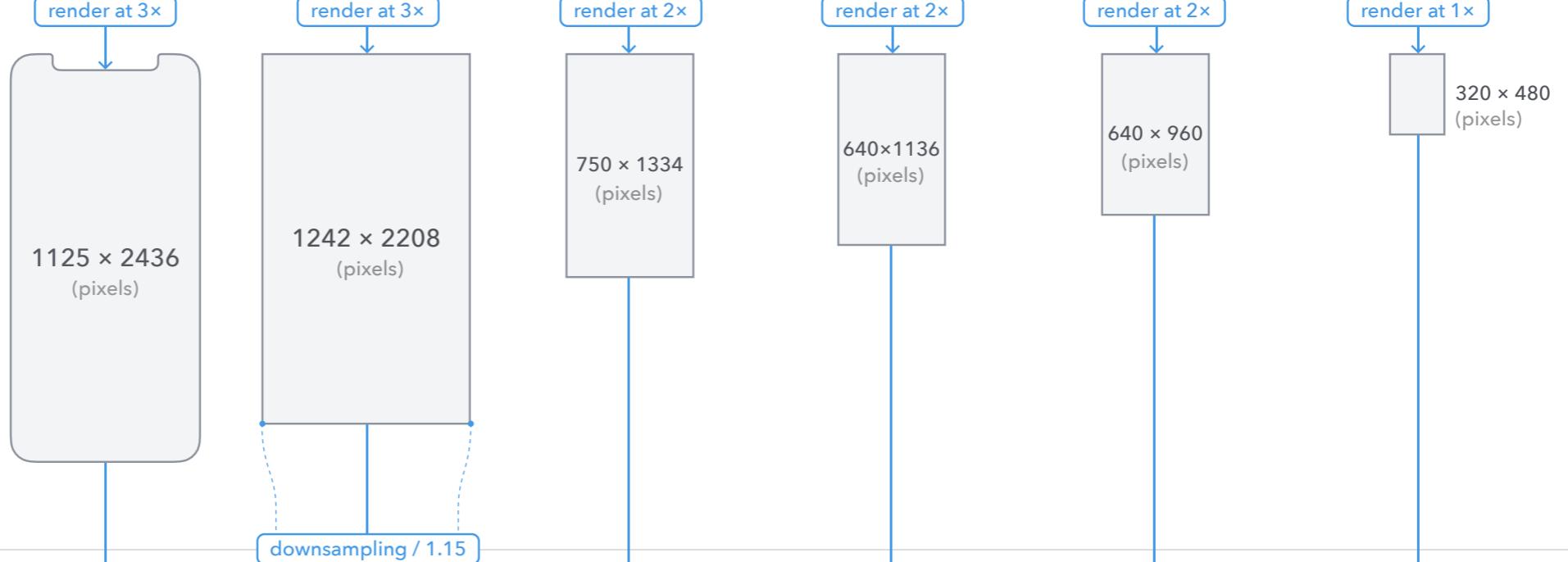
Points are abstract units, they only make sense in this mathematical coordinate space.



### Rendered Pixels

Point-based drawings are rendered into pixels. This process is known as rasterization.

Point coordinates are multiplied by scale factor to get pixel coordinates. Higher scale factors result in higher level of detail.



### Physical Pixels

The device screen may have lower pixel resolution than the image rendered in previous step.

Before the image can be displayed, it must be downsampled (resized) to lower pixel resolution.

downsampling / 1.15

1080 × 1920  
(device pixels)

# Heute

Layout und Koordinatensystem

CustomViews

Autolayout

StackView

Animationen

Size Classes

Zusammenfassung

# CustomViews

- CustomViews erweitern die Objekt-Palette bestehender Views und lassen sich im Storyboard als solche anzeigen, wenn diese als `@IBDesignable` anmontiert sind
- Eine Möglichkeit eine CustomView zu designen ist es, selber auf dem Layer zu zeichnen
  - Zuerst eine Klasse erstellen, die von `UIView` erbt
  - `func draw(_ rect: CGRect)` wird überschrieben und enthält das Zeichnen auf dem Layer
  - `draw()` wird niemals direkt aufgerufen, sondern mit `customView.setNeedsDisplay()` angestoßen

# CustomViews

```
class DrawingCustomView: UIView {

    override init(frame: CGRect) { // init from code
        super.init(frame: frame)
        setup()
    }

    required init?(coder aDecoder: NSCoder) { // init from storyboard
        super.init(coder: aDecoder)
        setup()
    }

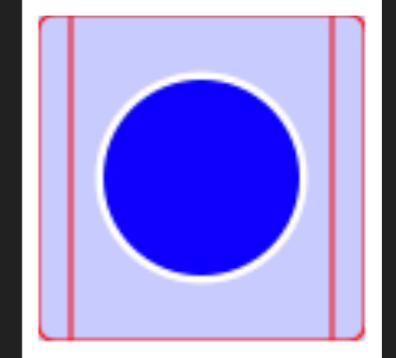
    private func setup() { backgroundColor = UIColor.blue.withAlphaComponent(0.2)

        override func draw(_ rect: CGRect) { // 'rect' is used for delta drawing (optimization)
            let roundedRect = UIBezierPath(roundedRect: bounds, cornerRadius: 5.0) // draw rounded rect
            roundedRect.move(to: CGPoint(x: 10, y: 0))
            roundedRect.addLine(to: CGPoint(x: bounds.minX + 10, y: bounds.maxY)) // draw line on the left
            roundedRect.move(to: CGPoint(x: bounds.maxX - 10, y: bounds.maxY))
            roundedRect.addLine(to: CGPoint(x: bounds.maxX - 10, y: 0)) // draw line on the right

            UIColor.red.setStroke(); roundedRect.stroke()

            let arc = UIBezierPath(arcCenter: CGPoint(x: bounds.midX, y: bounds.midY), radius: 30.0, startAngle: 0, endAngle: CGFloat.pi * 2,
clockwise: true) // draw arc
            arc.lineWidth = 5.0

            UIColor.white.setStroke(); arc.stroke()
            UIColor.blue.setFill(); arc.fill()
        }
    }
}
```



# CustomViews

```
class DrawingViewController: UIViewController {

    var customView: DrawingCustomView!

    override func viewDidLoad() {
        super.viewDidLoad()

        view.backgroundColor = .white

        customView = DrawingCustomView(
            frame: CGRect(x: 50, y: 50, width: 100, height: 100)
        )

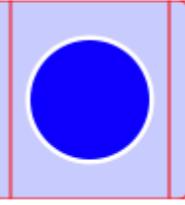
        customView.frame // x: 50, y: 50, w: 100, h: 100
        customView.bounds // x: 0, y: 0, w: 100, h: 100

        // customView.transform = CGAffineTransform(rotationAngle: CGFloat.pi / 4)

        view.addSubview(customView)
    }

    override func traitCollectionDidChange(_ previousTraitCollection:
    UITraitCollection?) {
        super.traitCollectionDidChange(previousTraitCollection)

        customView?.setNeedsDisplay() // redraw by forcing draw()
    }
}
```



# CustomViews

```
class DrawingViewController: UIViewController {

    var customView: DrawingCustomView!

    override func viewDidLoad() {
        super.viewDidLoad()

        view.backgroundColor = .white

        customView = DrawingCustomView(
            frame: CGRect(x: 50, y: 50, width: 100, height: 100)
        )

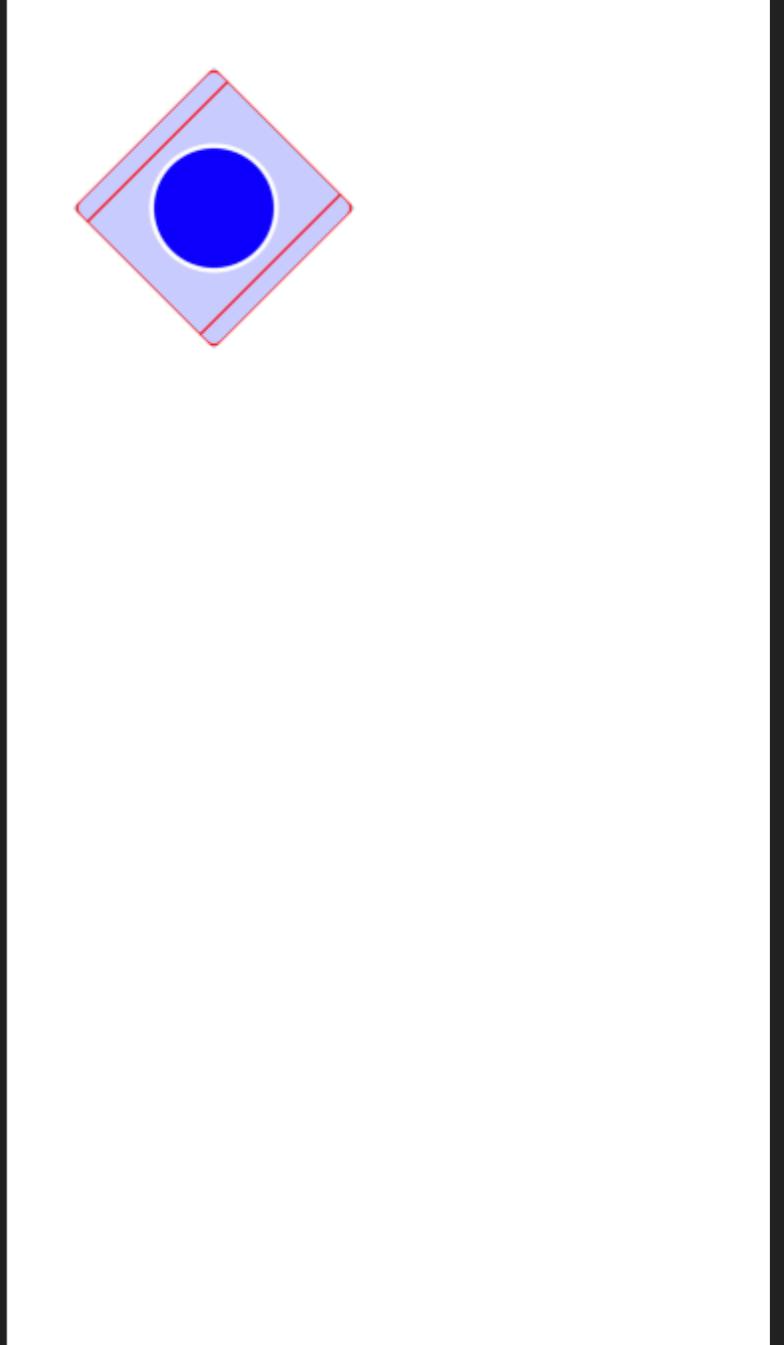
        customView.frame // x: 50, y: 50, w: 100, h: 100
        customView.bounds // x: 0, y: 0, w: 100, h: 100

        customView.transform = CGAffineTransform(rotationAngle: CGFloat.pi / 4)

        view.addSubview(customView)
    }

    override func traitCollectionDidChange(_ previousTraitCollection:
    UITraitCollection?) {
        super.traitCollectionDidChange(previousTraitCollection)

        customView?.setNeedsDisplay() // redraw by forcing draw()
    }
}
```



# CustomViews

- CustomViews erweitern die Objekt-Palette bestehender Views und lassen sich im Storyboard als solche anzeigen, wenn diese als `@IBDesignable` anmontiert sind
- Eine Möglichkeit eine CustomView zu designen ist es, selber auf dem Layer zu zeichnen
  - Zuerst eine Klasse erstellen, die von `UIView` erbt
  - `func draw(_ rect: CGRect)` wird überschrieben und enthält das Zeichnen auf dem Layer
  - `draw()` wird niemals direkt aufgerufen, sondern mit `customView.setNeedsDisplay()` angestoßen
- Eine weitere Möglichkeit ist es, die CustomView aus bestehenden UI-Elementen im Storyboard zu designen und unter einem eignen UI-Element anzubieten
  - Zuerst eine gleichnamige `.xib`-(Storyboard) und `.swift`-(assoziierte Klasse) Datei erstellen
  - `File's Owner` der `.xib` auf die gleichnamige `.swift` Klasse setzen
  - `.swift` Klasse von `CustomView*` erben lassen
  - CustomView designen, indem entweder selbst gezeichnet wird oder bestehende Views verwendet werden
  - Im Storyboard eine `UIView` aus der Palette herausziehen und die `Identität` auf die CustomView setzen

# CustomViews

```
@IBDesignable
class CustomView: UIView {

    var contentView: UIView!

    override init(frame: CGRect) {
        super.init(frame: frame)
        setUp()
    }

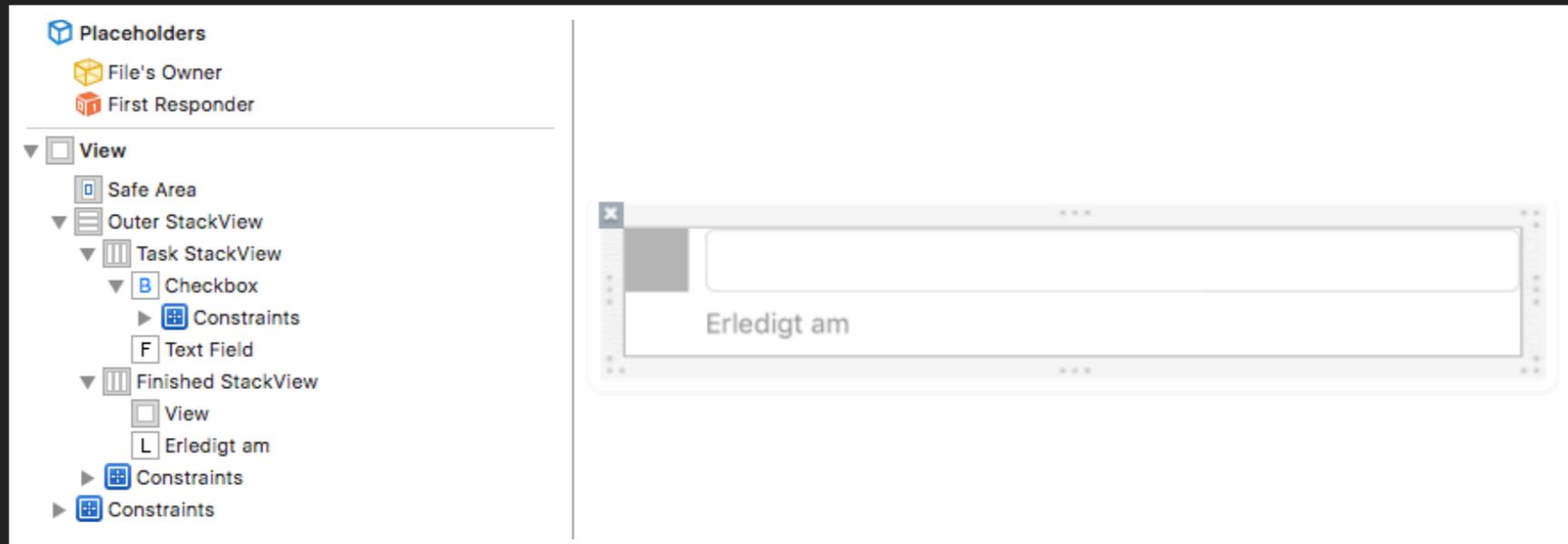
    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        setUp()
    }

    private func setUp() {
        let t = type(of: self)
        let b = Bundle(for: t)
        let nib = UINib(nibName: String(describing: t), bundle: b)

        if let view = nib.instantiate(withOwner: self, options: nil).first as? UIView {
            contentView = view
            contentView.frame = bounds
            addSubview(contentView)
        } else {
            fatalError("cant instantiate custom view form nib")
        }
    }
}
```

# CustomViews

```
class TodoView: CustomView {  
    @IBOutlet weak private var checkbox: TDCheckbox!  
    @IBOutlet weak private var textField: UITextField!  
  
    var todo: String? {  
        get { return textField.text }  
        set { textField.text = newValue }  
    }  
  
    @IBAction func checkboxTapped(_ sender: UIButton) { // do something }  
}
```



# Heute

Layout und Koordinatensystem

CustomViews

Autolayout

StackView

Animationen

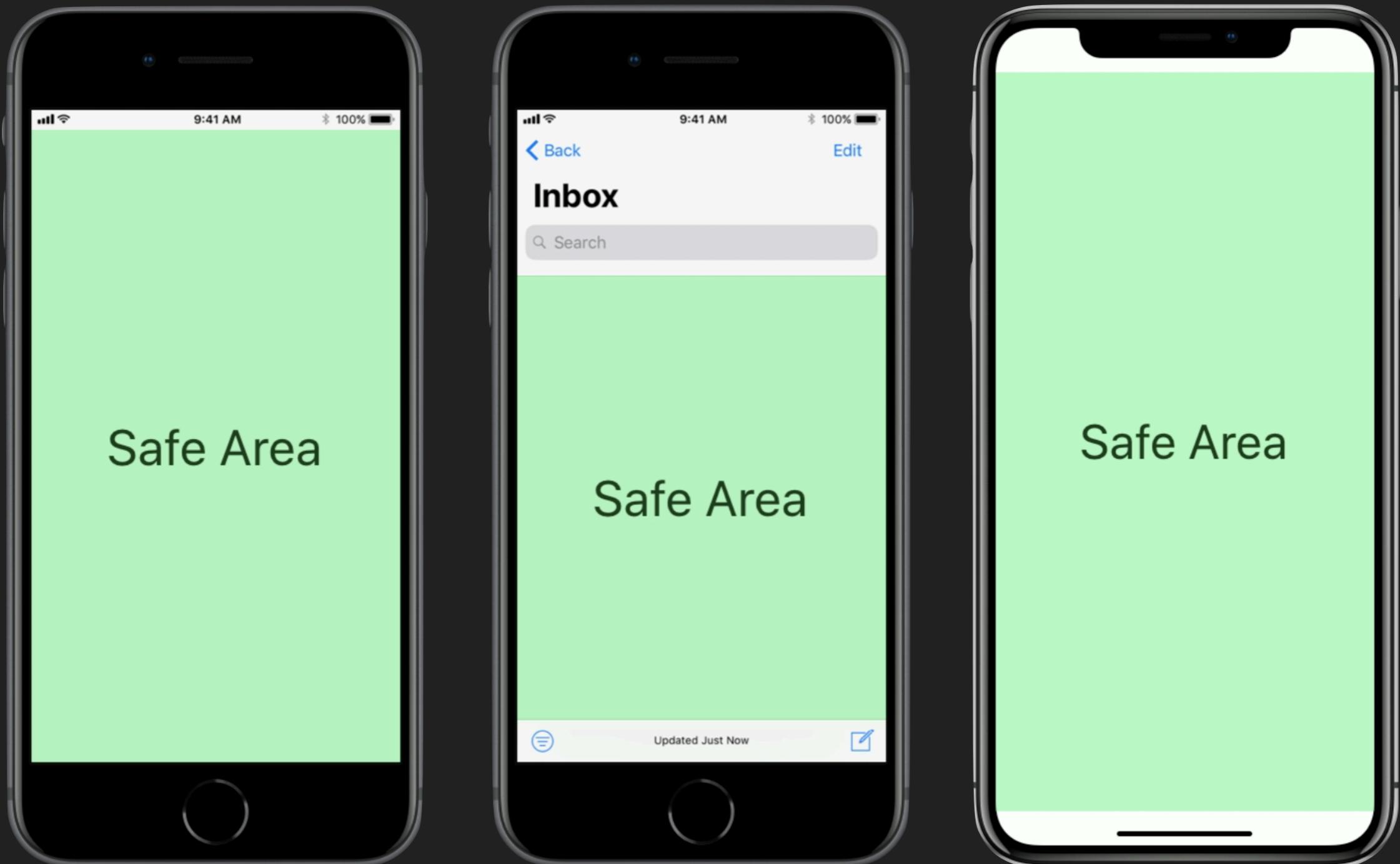
Size Classes

Zusammenfassung

# Autolayout

- “Universal” Apps mittels automatischer Adaption von Layout
  - Views für unterschiedliche Displaygrößen und Auflösungen
  - Rotationen (Landscape und Portrait)
  - Text-Styles (Fonts) für Barrierefreiheit
  - Designs für “Links zu Rechts” und “Rechts zu Links” Sprachen
- “Design (almost) once, ship anywhere”
- Autolayout findet innerhalb der Safe Area statt Neu in iOS 11
- Autolayout und Custom-Controls sind die Basis vom UI-Design
- Advanced: Size Classes (Compact und Regular), um displayspezifische Anpassungen vorzunehmen

# Autolayout



# Autolayout

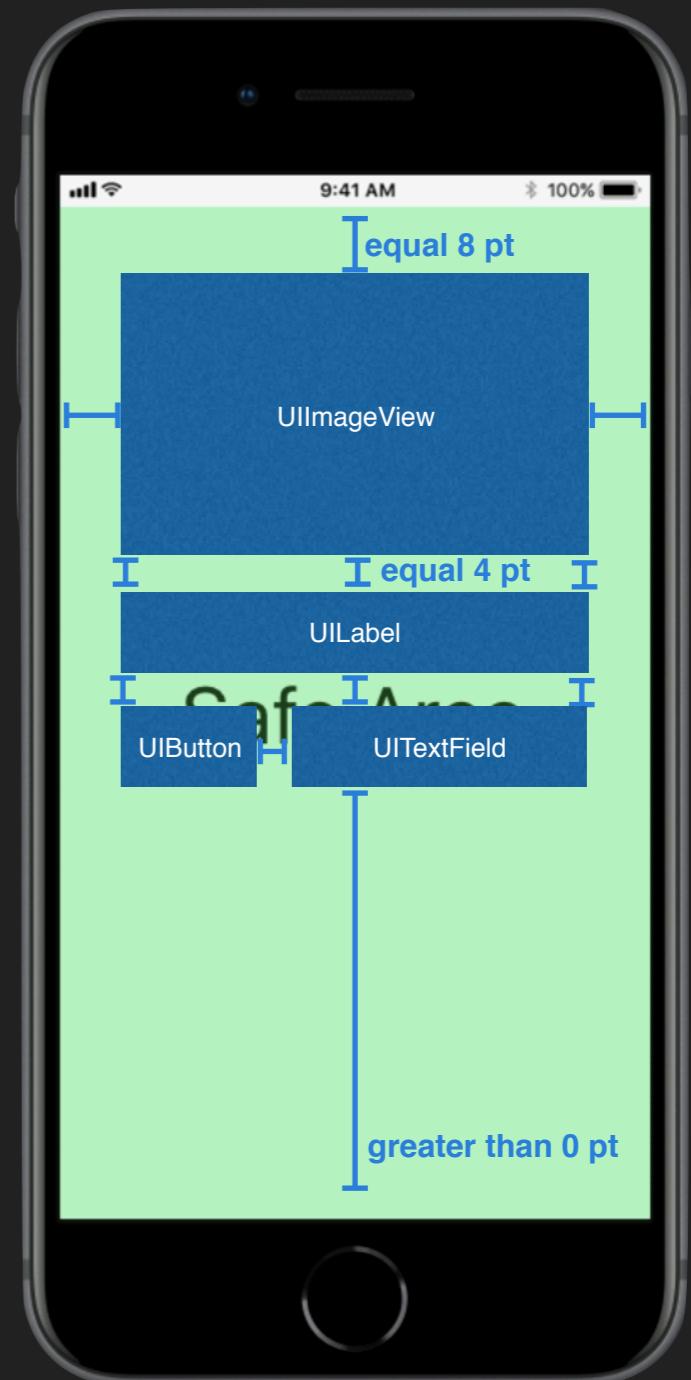
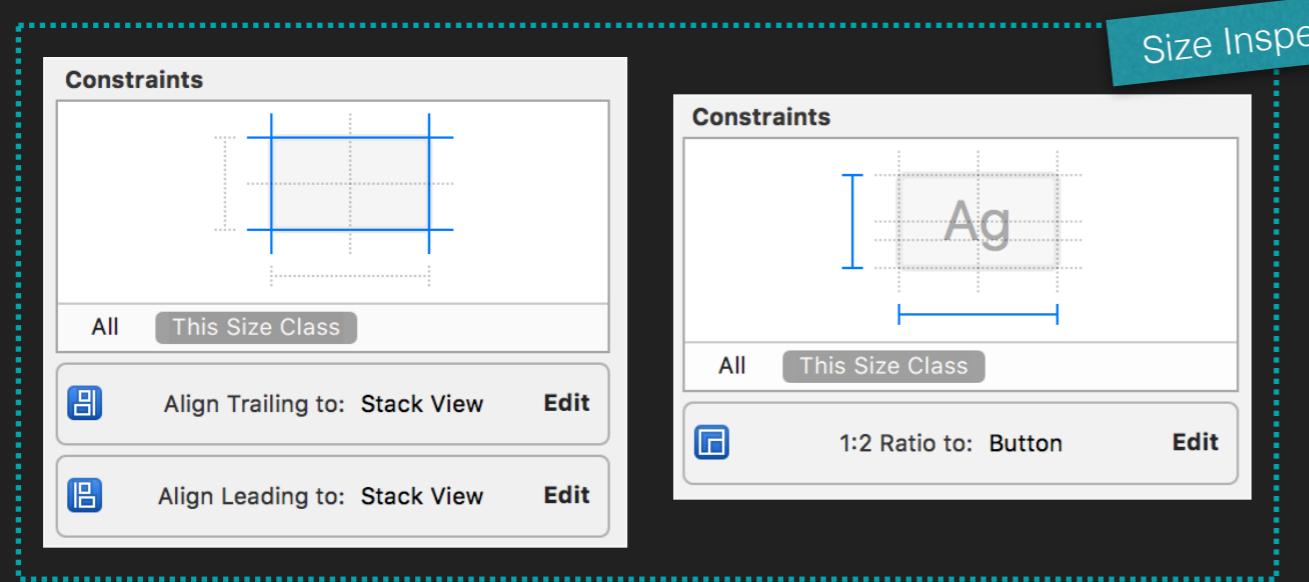


# Autolayout

- Setzen von **Constraints** für Views, welche Beziehungen zwischen jenen Views beschreibt
- Ausrichtung von Views an den **blauen Linien**
  - entweder **Reset to Suggested Constraints**
  - oder **Ctrl-Drag** von View zu View oder von View zu Kanten (Edges)
  - oder **Add Constraints (with Spacing) to Nearest Neighbour**
- Verifizieren im **Size Inspector**
- Eliminieren von **Magic Numbers**, indem die **Offsets** auf **Default** oder **0** gesetzt werden  
**Achtung:** Safe Area erlaubt aktuell keinen Default Offset (scheint ein Bug zu sein), weshalb Magic Numbers notwendig sind
- Beim Autolayout können Konflikte entstehen, wenn die Constraints nicht eindeutige (**ambiguous**) Informationen liefern
- Konflikte werden in der **Document Outline** oder im **Attribute Inspector** analysiert und behoben
- Constraints können sowohl mithilfe des Storyboards (primär) als auch im Code gesetzt werden
- Zudem lassen sich Constraints jederzeit aktivieren und deaktivieren
- Ziel: Autolayout so viele Constraints geben, das **genau eine** UI-Lösung produziert wird

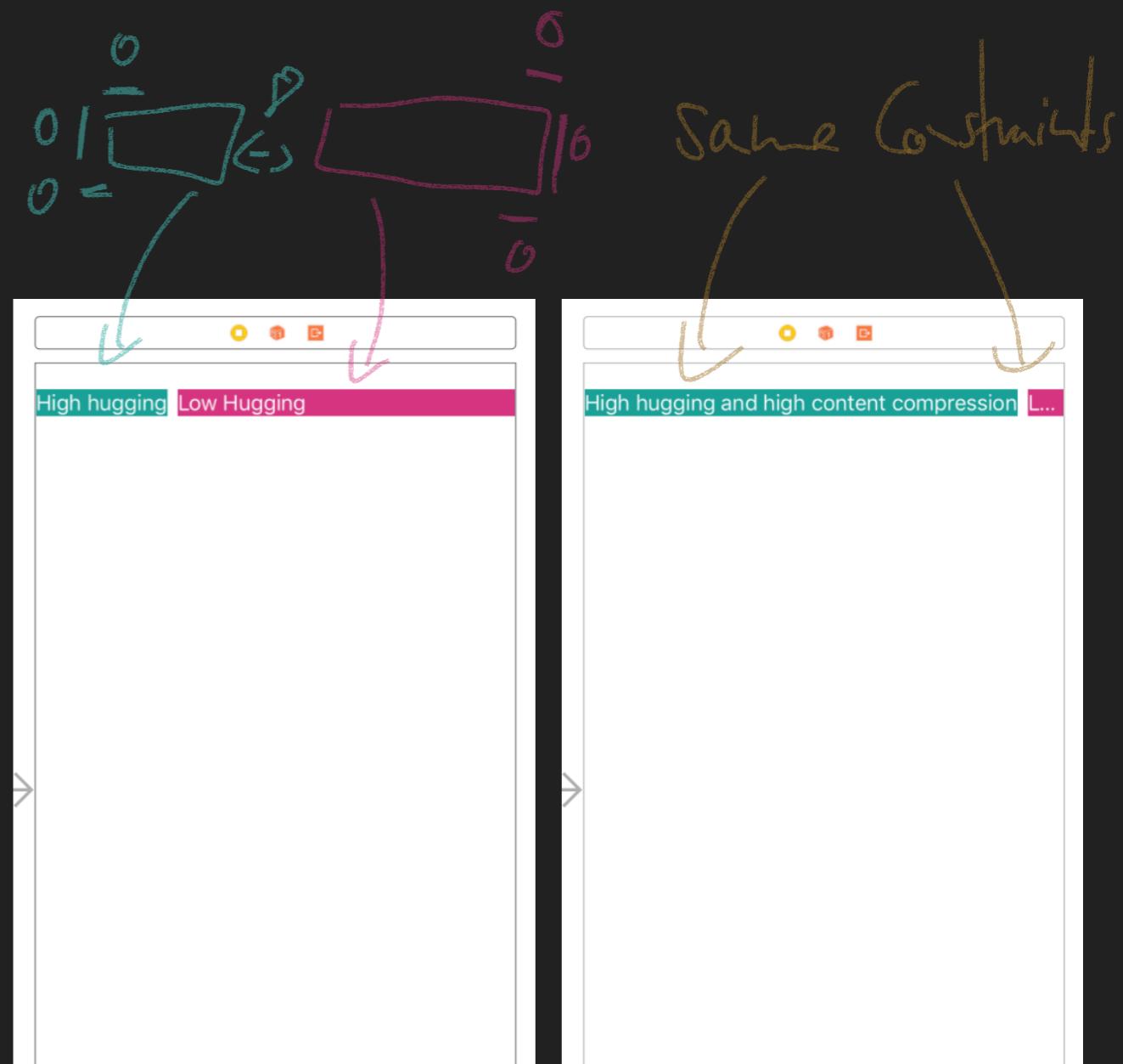
# Autolayout

- Constraints beschreiben eine Beziehung zwischen zwei Views
- Beispielhafte Beziehungen sind
  - Leading-, Trailing-, Top-, Bottom- Spaces oder Alignments
  - Center Vertically oder Horizontally
- Zudem gibt es auf sich bezogene Beziehungen wie Width, Height oder Aspect-Ratio
- Jede Beziehung hat zusätzlich einen Offset von n, 0 oder Default und einem Multiplier
- Zudem ist die Beziehung Equal, Greater Than or Equal oder Less Than or Equal



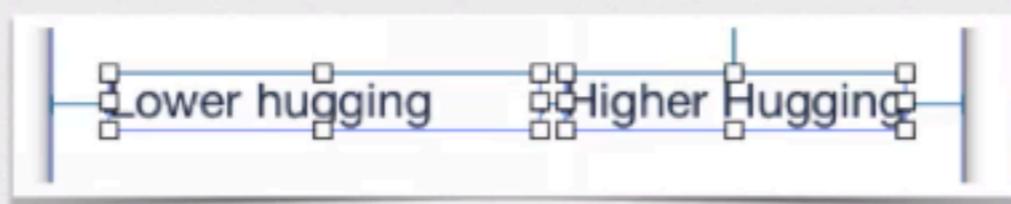
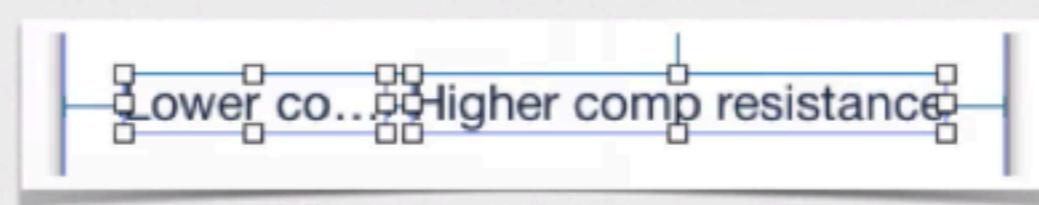
# Autolayout

- Bei einigen Views hängt die Größe vom Inhalt ab (**Intrinsic Content Size**)
  - Benutzergenerierter Inhalt oder willkürlicher Text aus einer Web-API, darstellt in einem UITextField
  - Localization von Strings als Text in UILabel
- Sind zwei solcher Views vertikal oder horizontal constrained, muss angegeben werden, welche View welche Priorität hat, um
  - bei zu **wenig Platz komprimiert** zu werden (**Compression-Resistance**)
  - bei zu **viel Platz gedehnt** zu werden (**Content-Hugging**)
- Keine Angabe führt zu einem Autolayout Fehler



# Autolayout

## Hugging vs. Resistance

- ✿ **Content Hugging:** Don't want to grow
- ✿ **Content Compression Resistance:** Don't want to shrink

raywenderlich.com

# Heute

Layout und Koordinatensystem

CustomViews

Autolayout

StackView

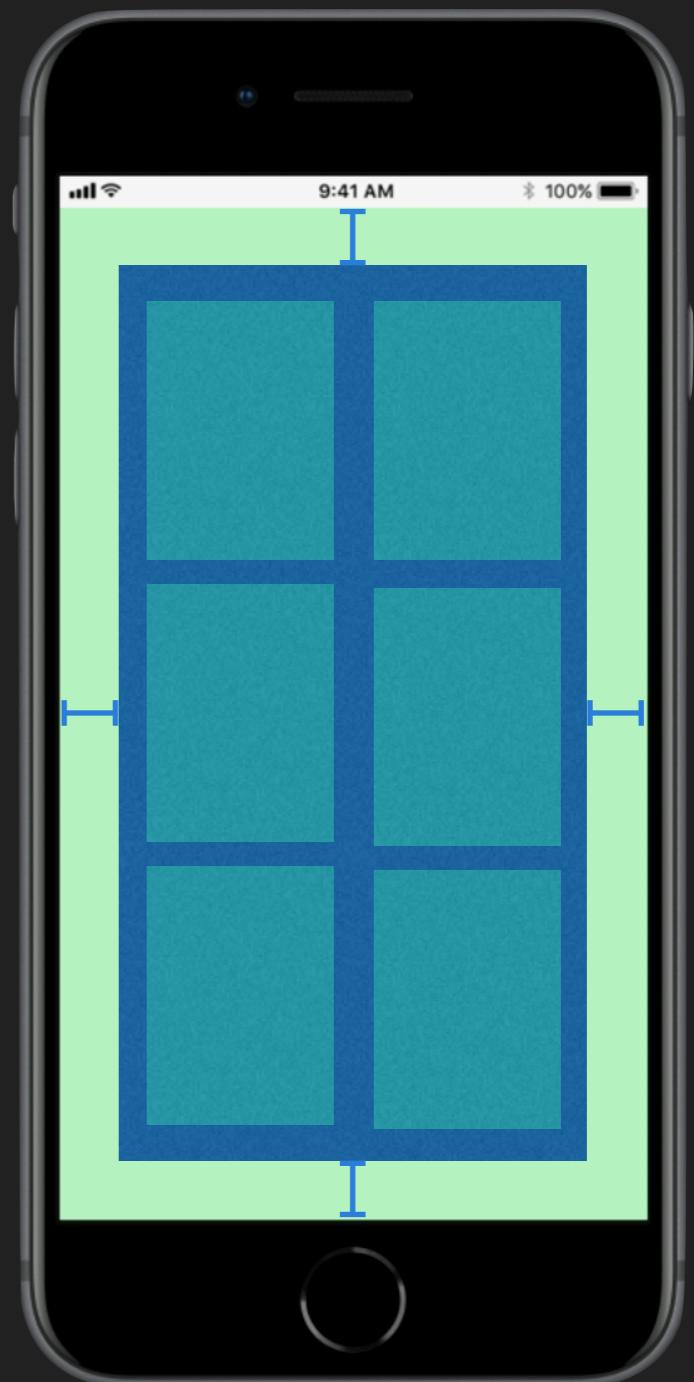
Animationen

Size Classes

Zusammenfassung

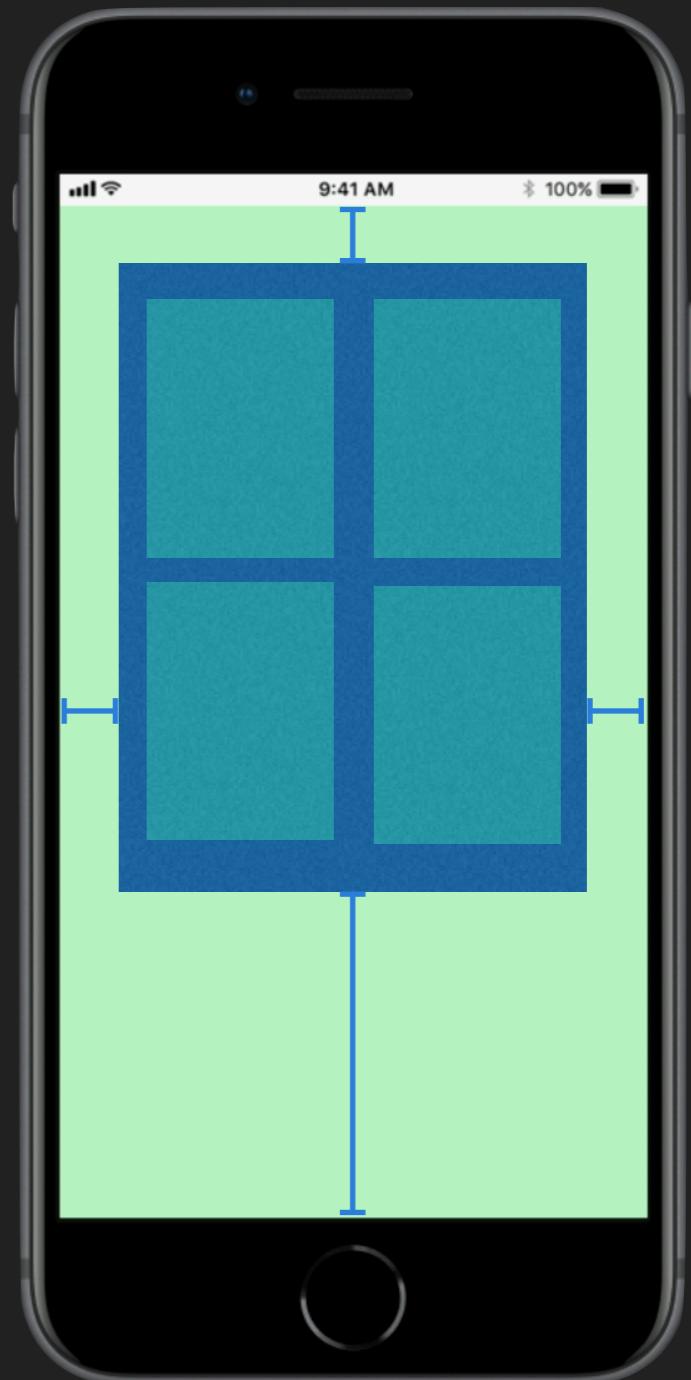
# StackView

- Container für mehrere Views
- Vorteil: StackView ausrichten und konfigurieren, SubViews passen sich an
- StackView wird mit Autolayout ausgerichtet
- Attribute der StackView steuern die interne Ausrichtung der Views auf Basis der äußeren Constraints
  - Horizontale oder Vertikale Orientierung der Views
  - Alignment, Distribution und Spacing
- Zur Laufzeit können Views
  - hinzugefügt, entfernt,
  - eingeblendet oder ausgeblendet werden.
- In jedem Falle greifen alle Constraints der StackView, sofern eine Änderung erfolgt ist
- Ohne StackView müsste man a) die View entfernen und b) die jeweiligen Autolayout Constraints finden und anpassen



# StackView

- Container für mehrere Views
- Vorteil: StackView ausrichten und konfigurieren, SubViews passen sich an
- StackView wird mit Autolayout ausgerichtet
- Attribute der StackView steuern die interne Ausrichtung der Views auf Basis der äußeren Constraints
  - Horizontale oder Vertikale Orientierung der Views
  - Alignment, Distribution und Spacing
- Zur Laufzeit können Views
  - hinzugefügt, entfernt,
  - eingeblendet oder ausgeblendet werden.
- In jedem Falle greifen alle Constraints der StackView, sofern eine Änderung erfolgt ist
- Ohne StackView müsste man a) die View entfernen und b) die jeweiligen Autolayout Constraints finden und anpassen



# Heute

Layout und Koordinatensystem

CustomViews

Autolayout

StackView

Animationen

Size Classes

Zusammenfassung

# Animationen

- Animation von **ausgewählten Properties**
  - Sichtbarkeit, Größe und Position, Transformationen, Alpha, Farbe, ...
  - Interpolation von Properties oder Transitionen zwischen Views
  - Closure beschreibt die Animation, welche über Zeit sichtbar wird
  - `UIView.animate` und `.transition` sind die alte APIs
  - `UIViewControllerAnimated` ist die neue API (seit iOS 10) und hat mehr Features wie interaktive Animationen mit Pause und Continue
- Animationen von **UIViewController** Transitionen, Rotationen etc.
- Direkter Zugriff auf **Core Animation** (fundamentales Framework)
- **Dynamic Animator** für physikalische Animationen
  - Collision, Gravity, Push, Snap, ...

# Animationen

```
extension UIView {

    @available(iOS 4.0, *)
    open class func animate(withDuration duration: TimeInterval, delay: TimeInterval, options: UIViewAnimationOptions = [], animations: @escaping () -> Swift.Void, completion: ((Bool) -> Swift.Void)? = nil)

    @available(iOS 4.0, *)
    open class func animate(withDuration duration: TimeInterval, animations: @escaping () -> Swift.Void, completion: ((Bool) -> Swift.Void)? = nil) // delay = 0.0, options = 0

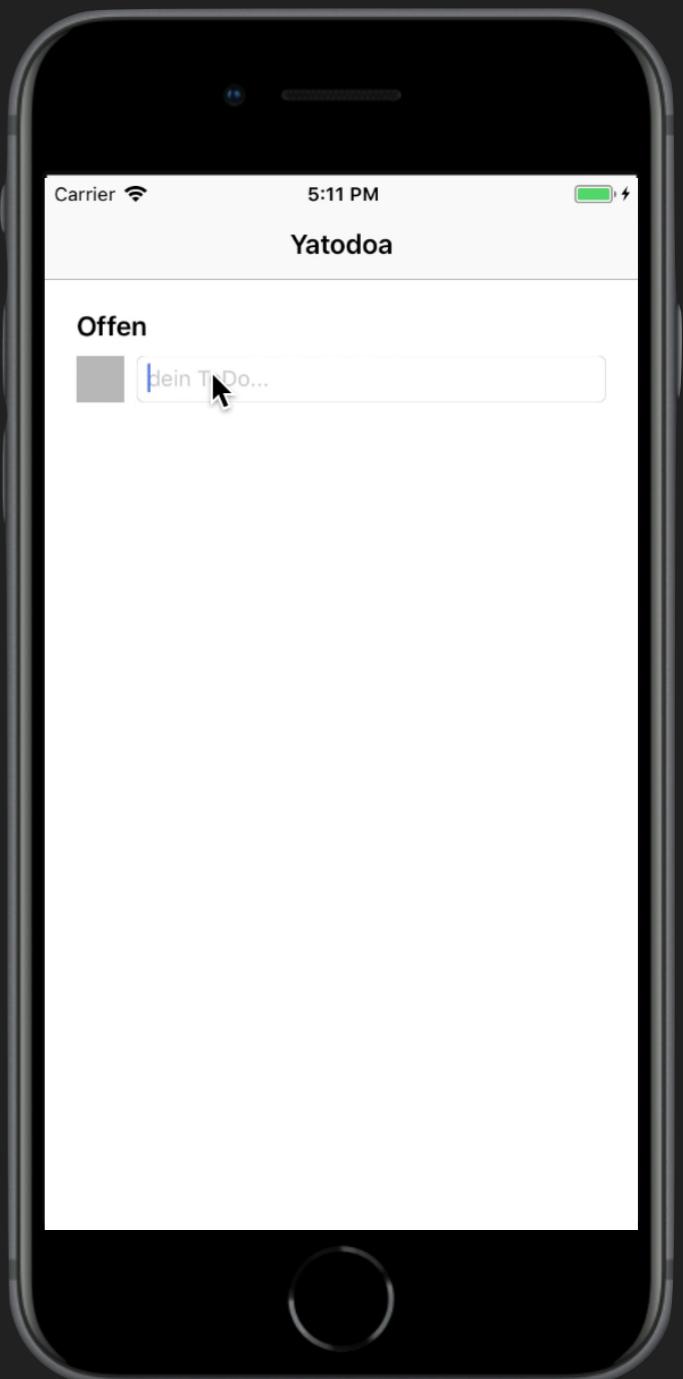
    @available(iOS 4.0, *)
    open class func animate(withDuration duration: TimeInterval, animations: @escaping () -> Swift.Void) // delay = 0.0, options = 0, completion = NULL

    @available(iOS 4.0, *)
    open class func transition(with view: UIView, duration: TimeInterval, options: UIViewAnimationOptions = [], animations: (() -> Swift.Void)?, completion: ((Bool) -> Swift.Void)? = nil)

    @available(iOS 4.0, *)
    open class func transition(from fromView: UIView, to toView: UIView, duration: TimeInterval, options: UIViewAnimationOptions = [], completion: ((Bool) -> Swift.Void)? = nil) // toView added to fromView.superview, fromView removed from its superview
}
```

# Animationen

```
class TodoViewController: UIViewController {  
  
    let todoStackView = UIStackView()  
  
    func addTodoButtonPressed() {  
        let todoView = TodoView()  
        todoView.isHidden = true // 'todoView' is hidden  
  
        todoStackView.addArrangedSubview(todoView) // still hidden  
  
        UIView.animate(withDuration: 0.3, animations: {  
            todoView.isHidden = false // interpolation of 'isHidden'  
            property within 0.3s from 'isHidden = true' to 'isHidden = false'  
        })  
    }  
}
```



# Heute

Layout und Koordinatensystem

CustomViews

Autolayout

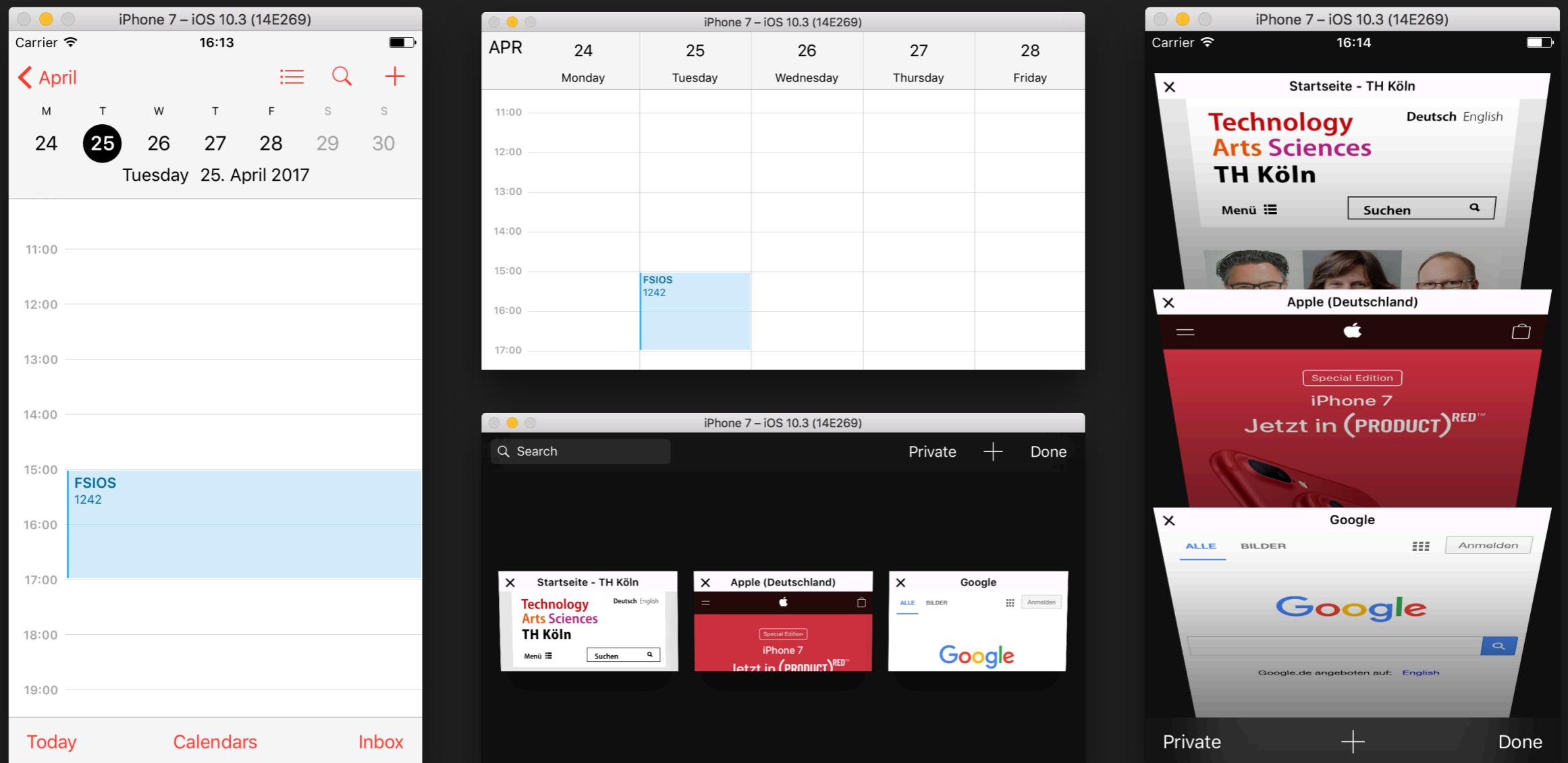
StackView

Animationen

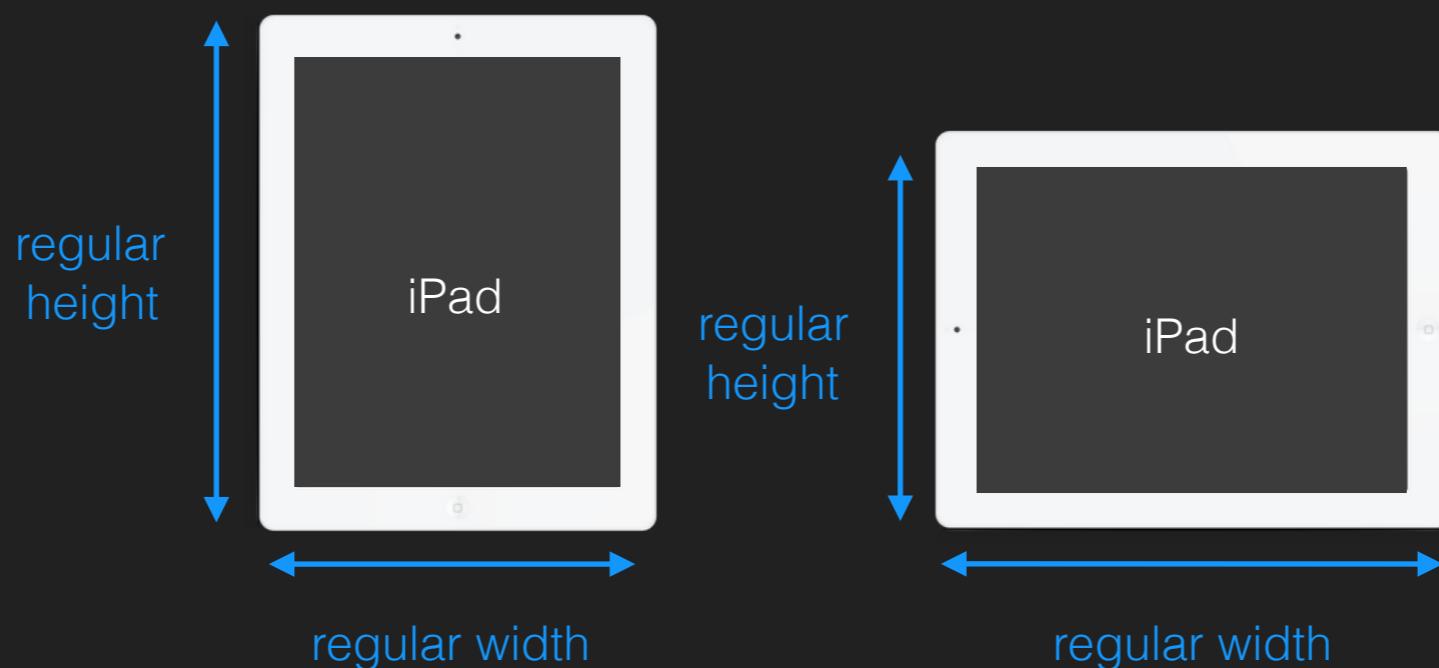
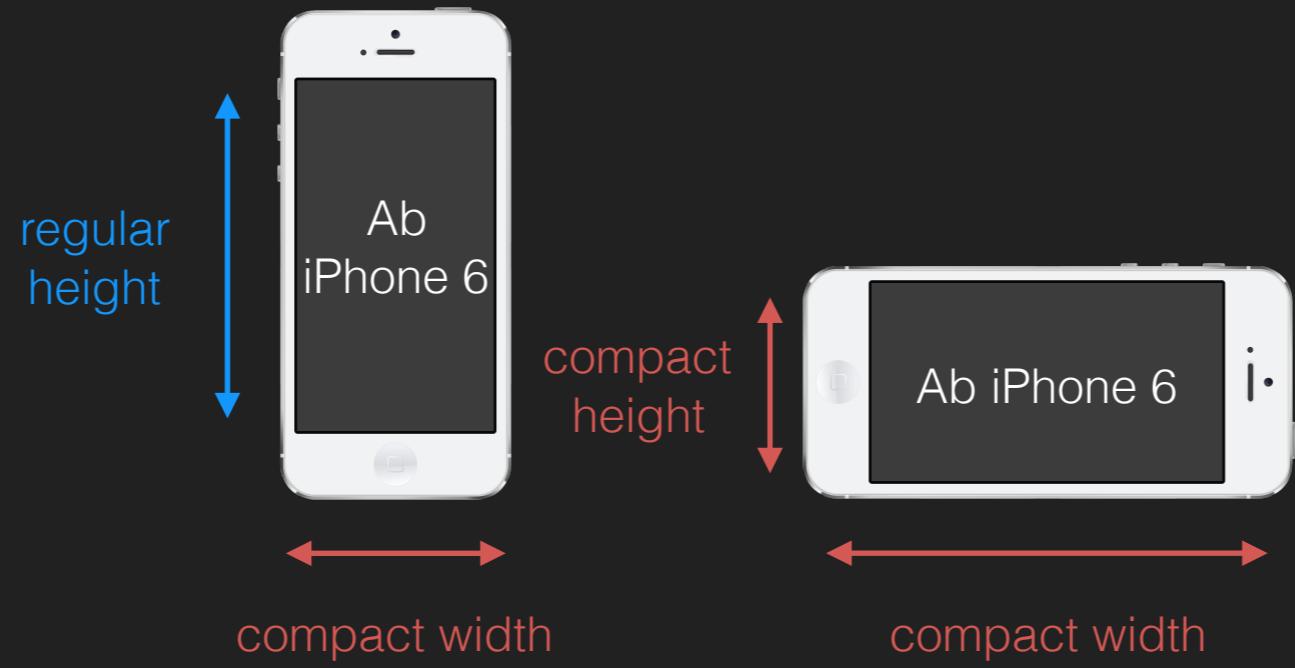
Size Classes

Zusammenfassung

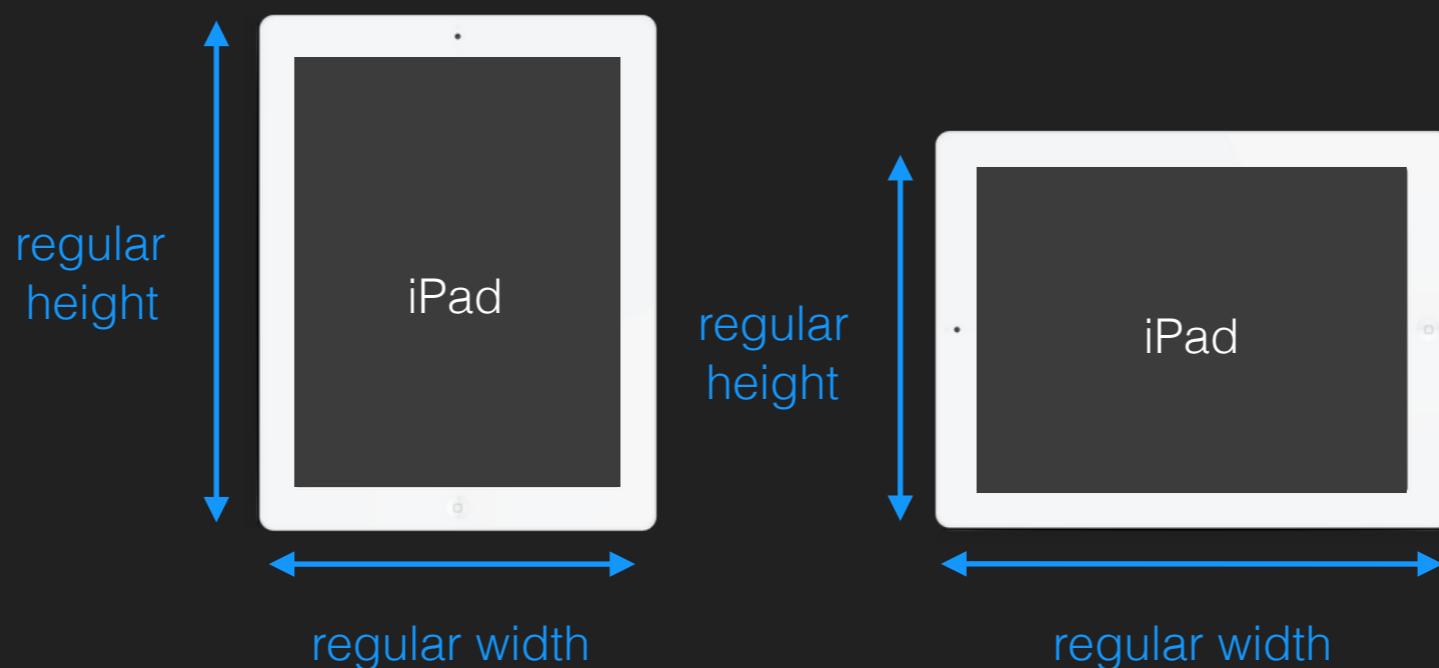
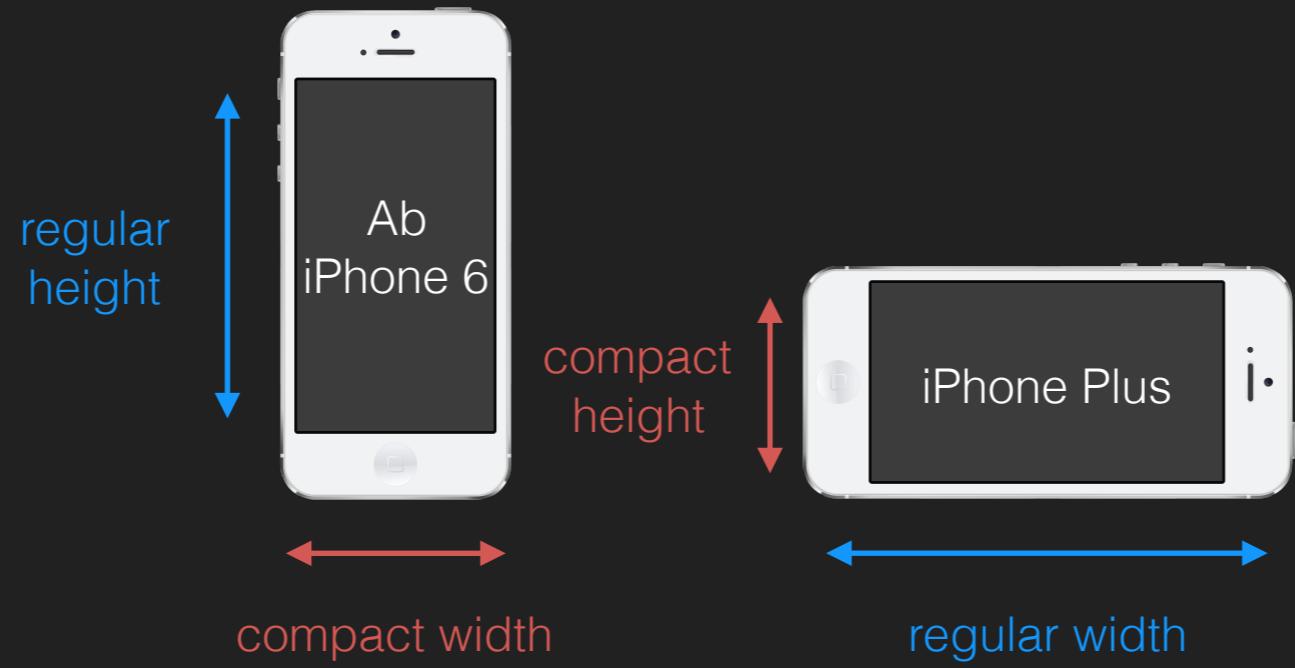
# Size Classes



# Size Classes



# Size Classes



# Size Classes

```
class TraitCollectionDependentViewController: UIViewController {

    override func viewDidLoad() {
        super.viewDidLoad()

        switch (traitCollection.verticalSizeClass, traitCollection.horizontalSizeClass) {
        case (.compact, .compact): print("landscape iphone 6 and above")
        case (.regular, .compact): print("portrait iphone 6 and above")
        case (.compact, .regular): print("portrait iphone plus")
        case (.regular, .regular): print("iPads")
        case _: print("something else ...")
        }
    }

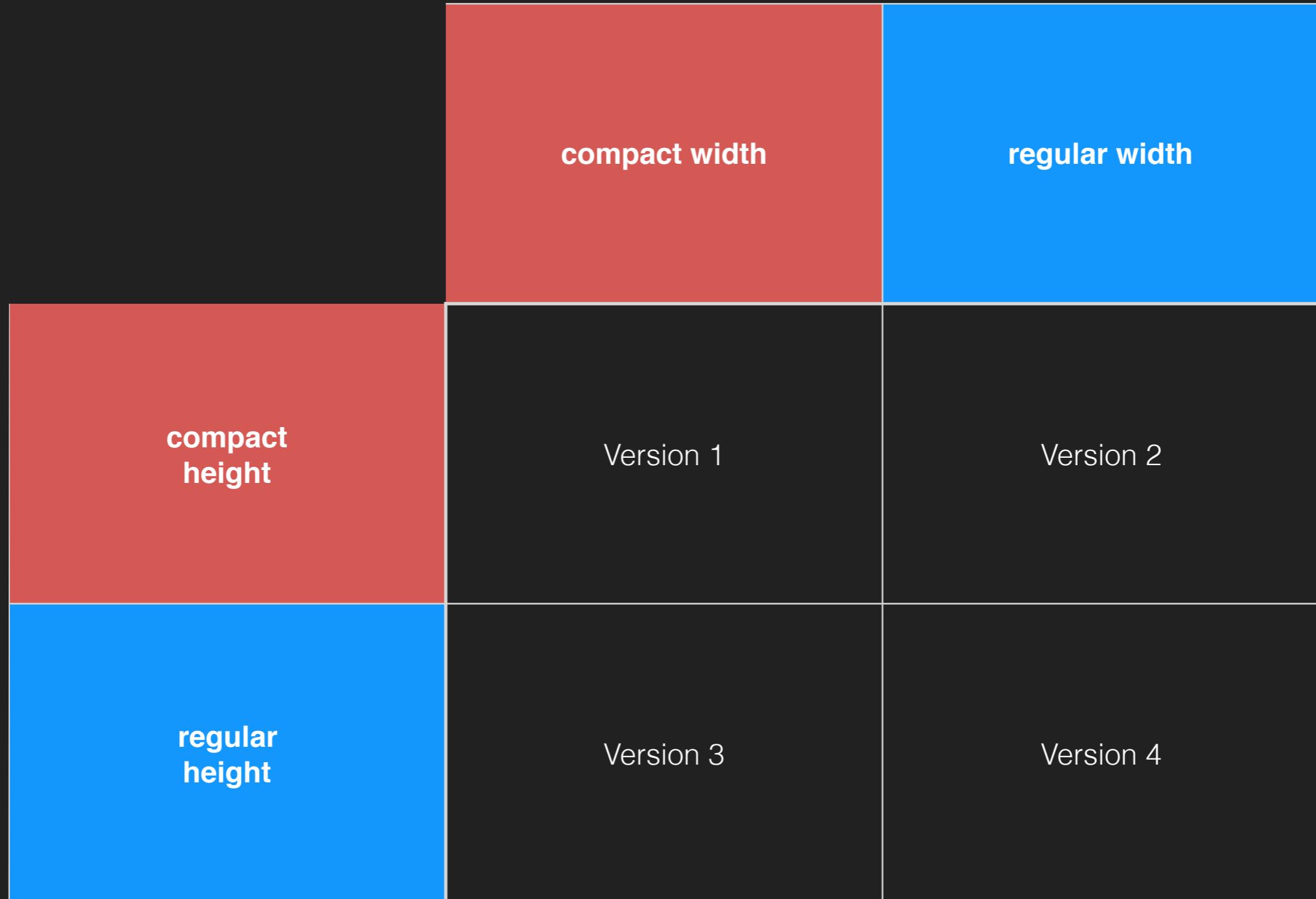
    override func willTransition(to newCollection: UITraitCollection, with coordinator: UIViewControllerTransitionCoordinator) {
        super.willTransition(to: newCollection, with: coordinator)

        print("called before traitCollectionDidChange(:) is called")
    }

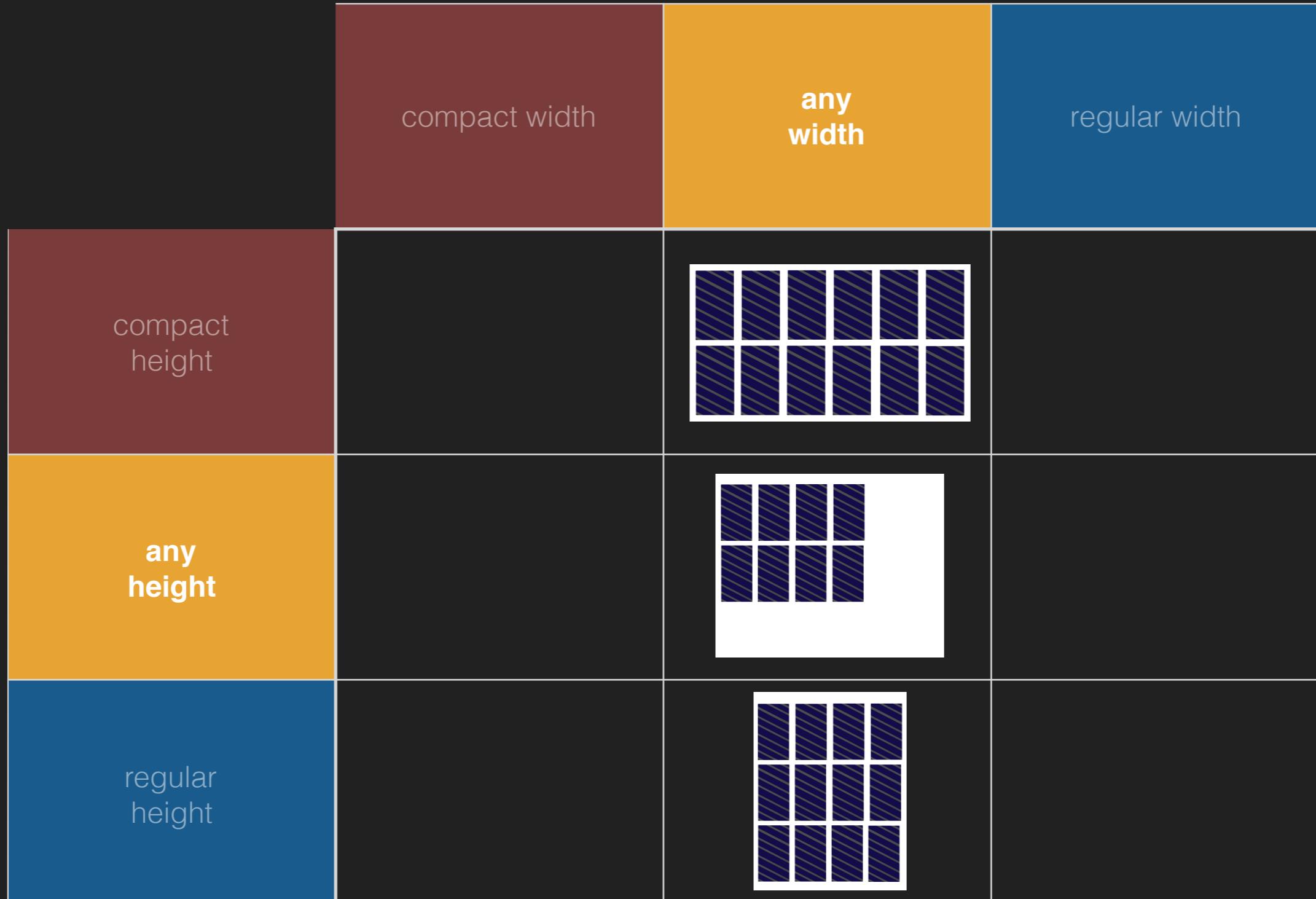
    override func traitCollectionDidChange(_ previousTraitCollection: UITraitCollection?) {
        super.traitCollectionDidChange(previousTraitCollection)

        if let previousTraitCollection = previousTraitCollection {
            print("adjust views when device is rotated")
        }
    }
}
```

# Size Classes

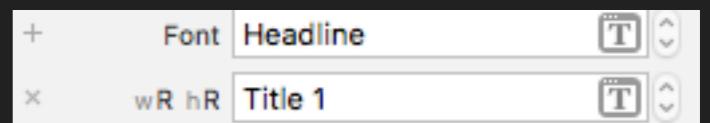


# Size Classes



# Size Classes

- Initiales Design für wA (Any Width) und hA (Any Height)
- Angepasstes Design für wC/R und hC/R
  - Constraints hinzufügen oder entfernen
  - Views einblenden oder ausblenden Neu mit Xcode 9
- Angepasste Attribute für eine bestimmte Size Class
  - Gegeben ist wC hR mit *Headline* als Font
  - Eine Variation der Font für wR hR setzt die Font auf *Title 1*



# Heute

Layout und Koordinatensystem

CustomViews

Autolayout

StackView

Animationen

Size Classes

Zusammenfassung

# Zusammenfassung

- Hierarchisches Layout mit Super- und SubViews, die jeweils ein lokales und globales Koordinatensystem haben
- CustomViews erweitern die Objekt-Palette und lassen sich wie System-Views im Storyboard anzeigen
- Autolayout adaptiert ein Design für mehrere Bildschirmgrößen, indem Beziehungen (Constraints) zwischen Views definiert werden, welche Position und Größe reglementieren
- StackViews gruppieren mehrere Views und propagieren Constraints zu den darin befindlichen SubViews
  - Zudem lässt sich das Hinzufügen und Entfernen von Views in StackViews animieren, da die Constraints bei jeder Aktion evaluiert werden
  - StackViews erleichtern das Layouten und sind, anders als Autolayout, keine Pflicht
- iOS bietet mehrere Möglichkeiten für Animationen, wobei die einfachste Art auf Interpolation von UI-Properties basiert
- Size-Classes passen Constraints und ausgewählte Attribute für bestimmte Gerätegrößen gesondert an