

# Konstrukce aplikace v iOS



## UIKit / Storyboard

IZA, Martin Hrubý, FIT VUT, 2022 / 23

# Úvod

- Několik střípků z předchozí technologie:
  - Pro lepší pochopení konceptů SwiftUI.
  - SwiftUI aplikace smí využívat prvky UIKit.
- Univerzální koncepty iOS aplikací.
- Budeme průběžně srovnávat se SwiftUI.
  - ... a hledat kořeny... :)

# Architektura aplikace

- UIApplication+delegate — API na jádro iOS.
  - ViewControllers — řízení "jedné obrazovky" a řízení programu (Storyboard, Segue).
  - Základy Views a geometrie Views.
- M-V-C. M-VC.
  - V-C jsou platformově závislé (iOS/tvOS, macOS, watchOS).
  - Cocoa (AppKit, NSView...). CocoaTouch (UIKit, UIView...).
  - Model — DB, komunikace vně, komunikace uvnitř.

# Start aplikace

- Instance *UIApplication*, *UIApplicationDelegate*.
  - `didFinishLaunchingWithOptions` — jako `main()` programu.
  - globální data aplikace — model. Je dobré je mít pohromadě.
- Instancie *UIWindow* / *UISceneWindow*.
  - Počáteční *ViewController* — `rootViewController`.
  - *UIWindow* začne být "*key and visible*".
- Startovní výpočty — vedlejší vlákna.
  - Co nejrychlejší start aplikace.
  - Obsah aplikace "dodělávat za běhu".

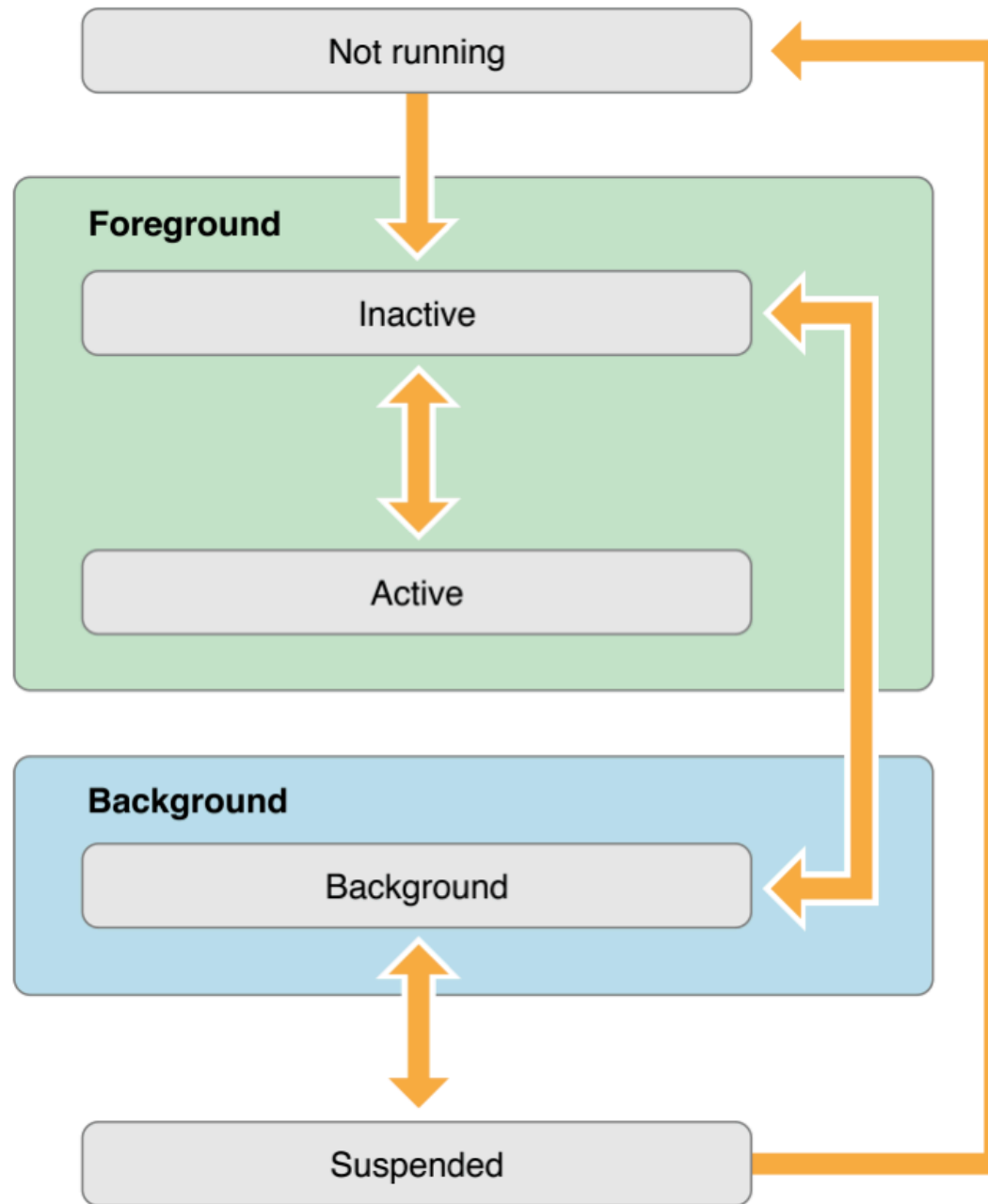
# UIApplication

- Knihovná třída. Nepředpokládá se odvozování vlastní.
- Singleton (*UIApplication.shared*).
  - Rozhraní mezi OS a aplikací (události).
  - Ovládá základní podobu aplikace v systému (ikonka, horní lišta apod.).
  - Referencuje: *app-delegáta*, *windows* a *keyWindow*.

# UIApplication delegate

- Singleton. Může referencovat objekty globálního charakteru (typicky data, CoreData, apod.)
  - didFinishLaunchingWithOptions. Neblokovat příliš start aplikace (globální fronta).
  - dřív konstrukce UIWindow, rootVC apod. Dnes StoryBoard.
- Události změny stavu aplikace — popředí/ pozadí, will / did,
  - ... se přesunuly do UIWindowSceneDelegate

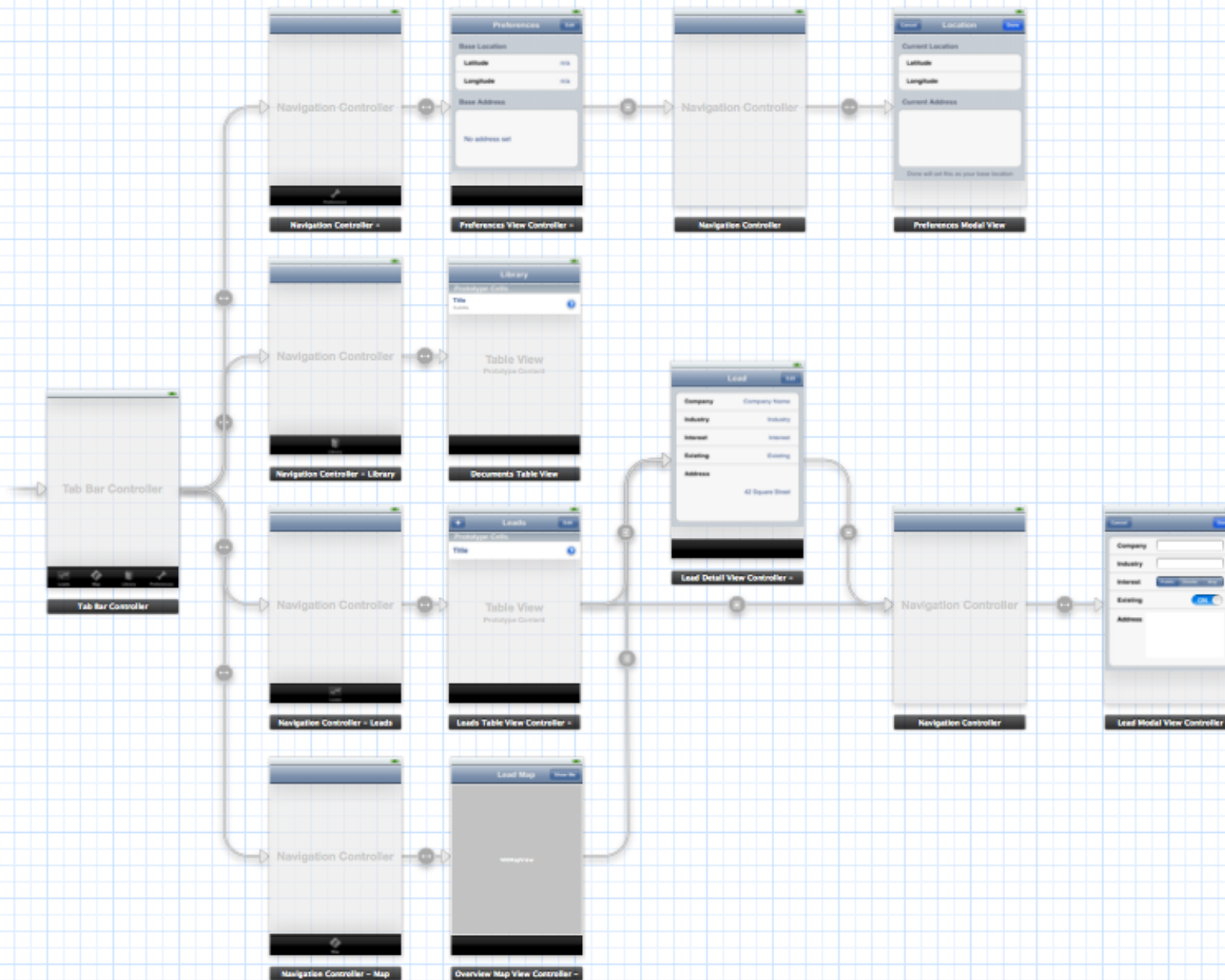
# Stavy aplikace / scény

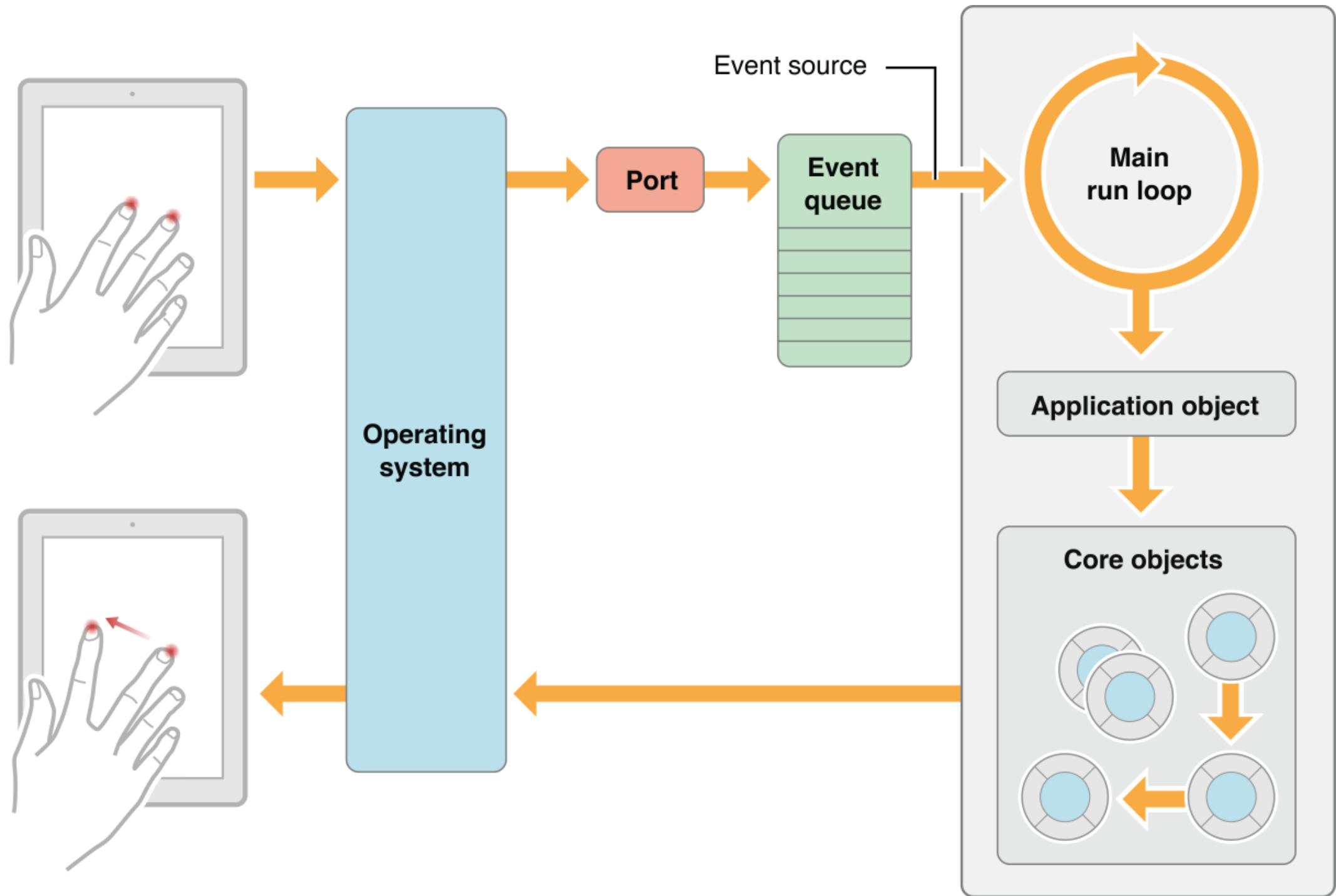


# Stavy aplikace / scény

- *Not running* — nespouštěna nebo ukončena.
- Popředí (je viditelná):
  - *Active* — vykonává kód, přijímá události.
  - *Inactive* — nepřijímá události, běží. Typicky krátký moment.
- *Pozadí* (není viditelná): smí vykonávat kód.
- *Suspended* — je v paměti, neběží. Systém smí aplikaci kdykoli ukončit.
  - Aplikace je zodpovědná za uložení dat při přechodu do BG.







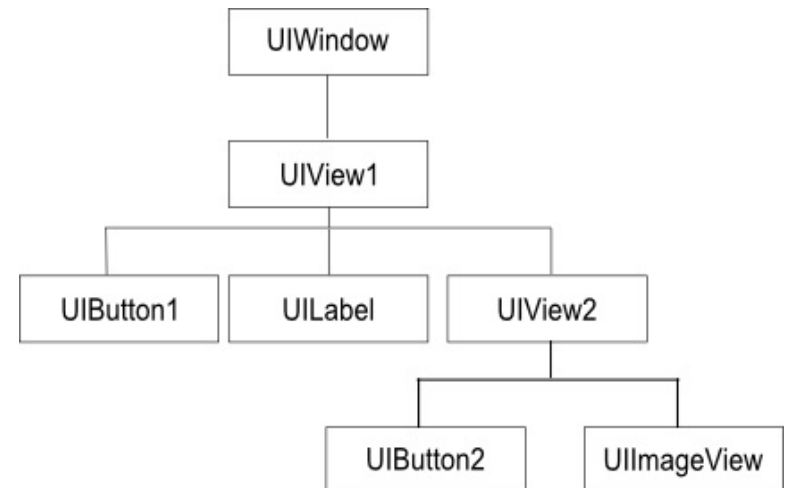
# Koncepce Views

- UIView, UILabel, UISwitch, UITableViewCell,
- Superview, subviews — hierarchie.
- Smyslem je skládat UI z knihovných komponent (versus drawRect:). Zobrazovací postscript.
- Provádět změny do UI smí pouze *hlavní vlákno*.

```
class ViewController: UIViewController {  
    @IBOutlet var textik : UILabel?  
  
    func inicializuj() {  
        textik?.text = "Napis hello";  
    }  
}
```

# Topologie views

- View má: superview a subviews:
  - `superview.addSubview(myView)`
  - `myView.removeFromSuperview()`
- Viditelnost View na obrazovce. Dynamika.
- Předpokládáme stromovou topologii.



# Geometrie View

- *Frame* — definuje umístění *view* v jeho *superview*.
- *Bounds* — definuje velikost *view* a jeho lokální souřadný systém.
- *Frame* a *bounds* nemusí mít stejné velikosti.
  - `CGRect(x: y: width: height:)`. Origin. Size.
- Souřadnice (0, 0) — levý horní roh, (macOS).
- Pozor: jednotkou souřadnice NENÍ 1 pixel.
  - souřadný systém je veden v číslech typu `CGFloat` (float).

# Vytvoření objektu view

- *var label = UILabel(frame: CGRect(x: 0, y: 0, width: 200, height: 21))*
  - Definuje frame,
  - nastaví *bounds* na *origin=(0,0)*, *size=(200,21)*.
- `view.addSubview(label)`

```
public struct CGRect {  
    public var origin: CGPoint  
    public var size: CGSize  
    public init()  
    public init(origin: CGPoint, size: CGSize)
```

# Proces rasterizace a kompozice

- Rasterizace (`drawRect:`), *bounds*(( $x,y$ ), ( $w,h$ )).
  - Alokujeme prázdnou bitmapu velikosti ( $w,h$ ).
- Představa projekce rasterizace:
  - V abstraktním souřadném systému grafického objektu se objekt (myšleně) vykreslí s počátkem ( $0, 0$ ).
  - Tento (myšlený) obraz se obtiskne do bitmapy ( $w, h$ ), kterou přiložíme do bodu ( $x, y$ ) obrazu.
  - Obecně *pouze část obrazu* se obtiskne do bitmapy ( $w, h$ ).

# Kompozice

- Mějme superview  $S$ ,  $S.frame$ ,  $S.bounds$ ,
- Představa: kompozice bitmapy  $X$  subview je pokračování rasterizace do myšleného obrazu.
- Tj. do myšleného souř. sys. umístím na  $X.frame.origin$  bitmapu  $X$  (oříznutou  $X.frame.size$ ).
- Pro  $S$  mám bitmapu  $S.bounds.size$ , kterou obtisknu umístěnou do  $S.bounds.origin$ .
- Efektivně  $X$  jde na  $X.frame.origin - S.bounds.origin$ .



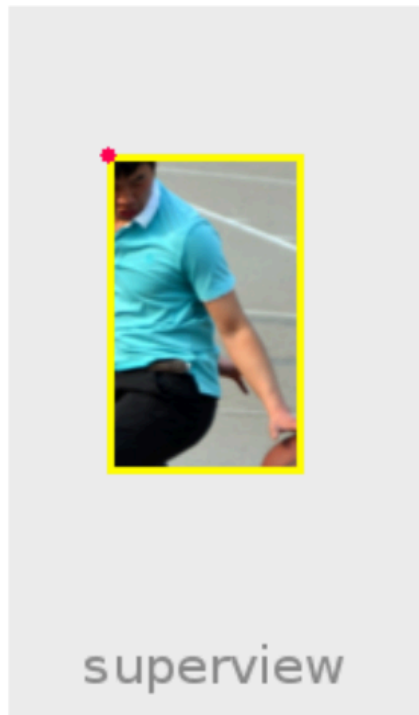
## Frame

origin = (40, 60)  
width = 80  
height = 130

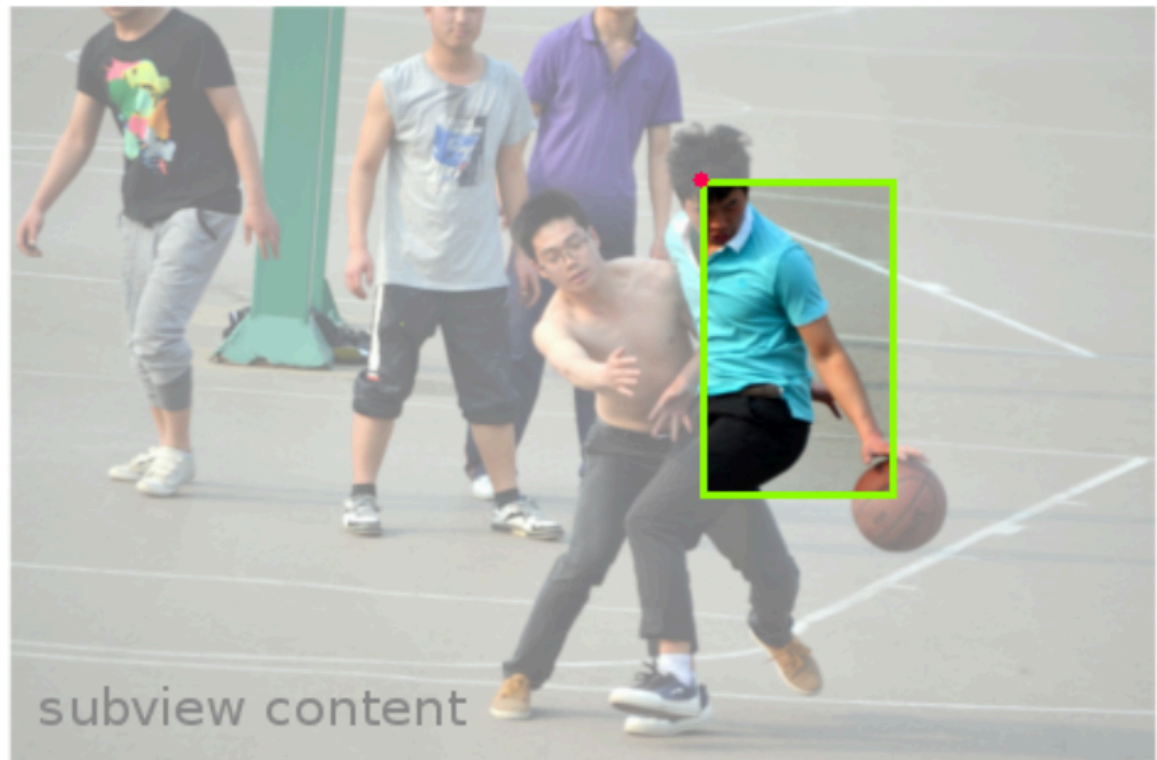
## Bounds

origin = (280, 70)  
width = 80  
height = 130

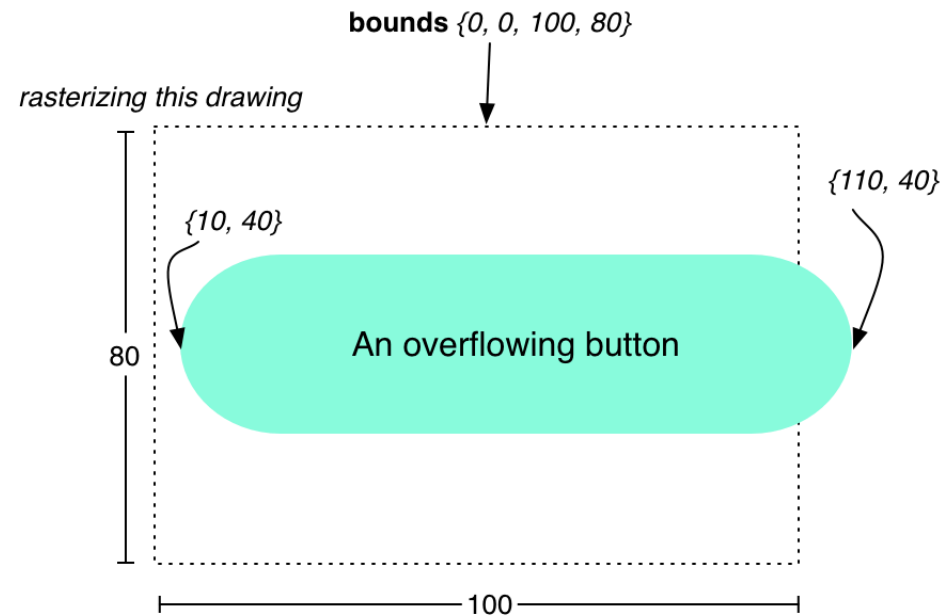
## Frame



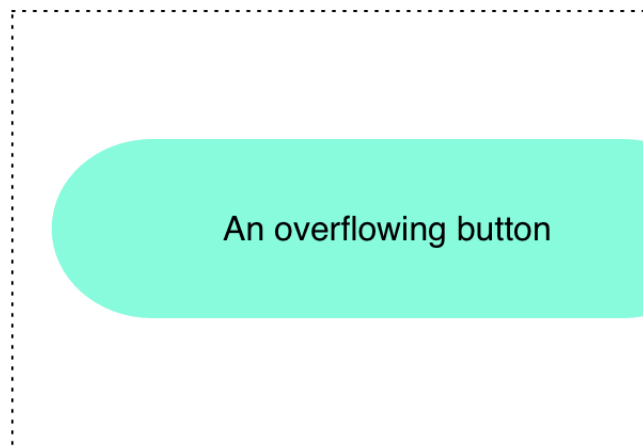
## Bounds



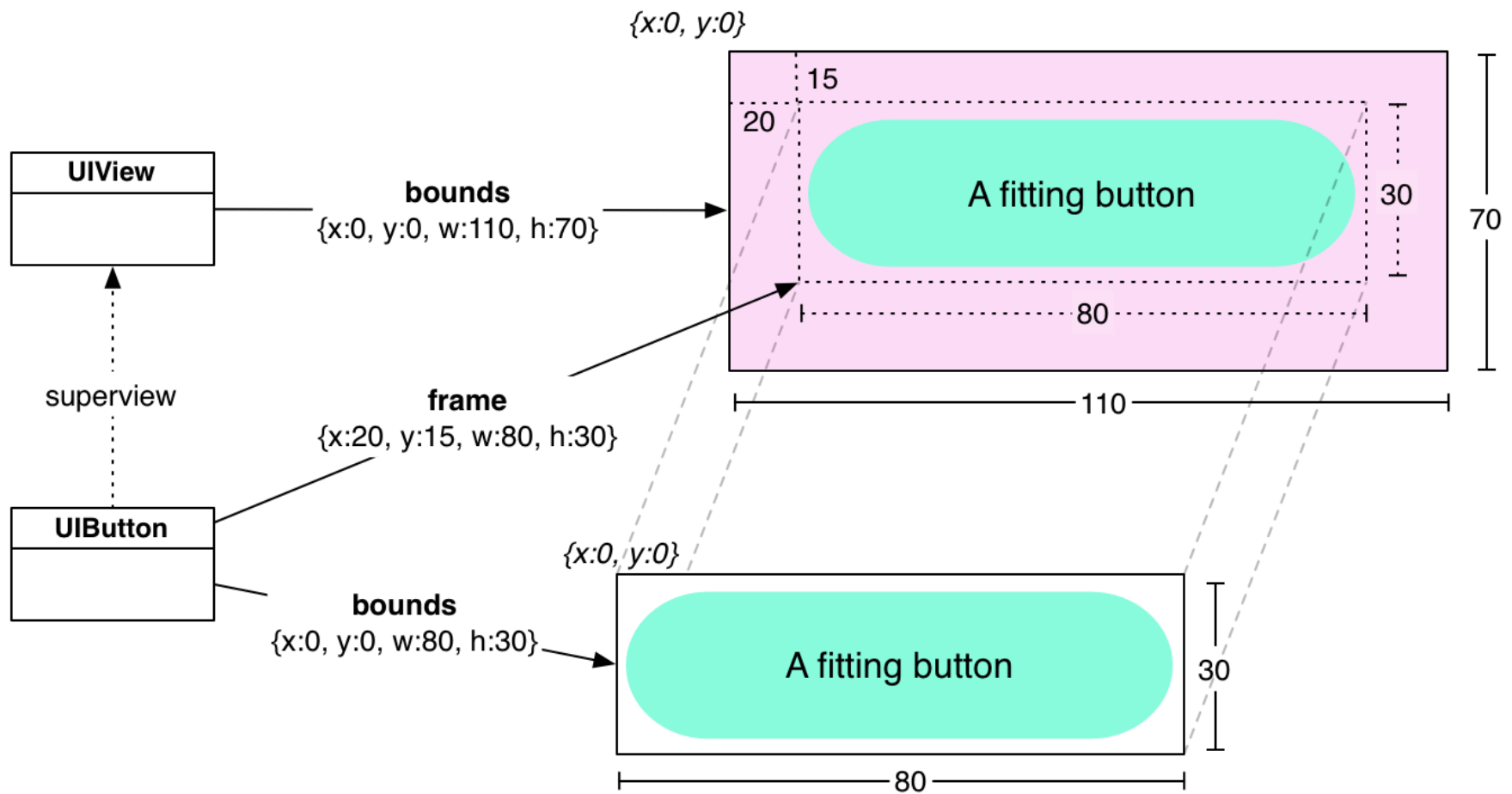
# Rasterize do bounds



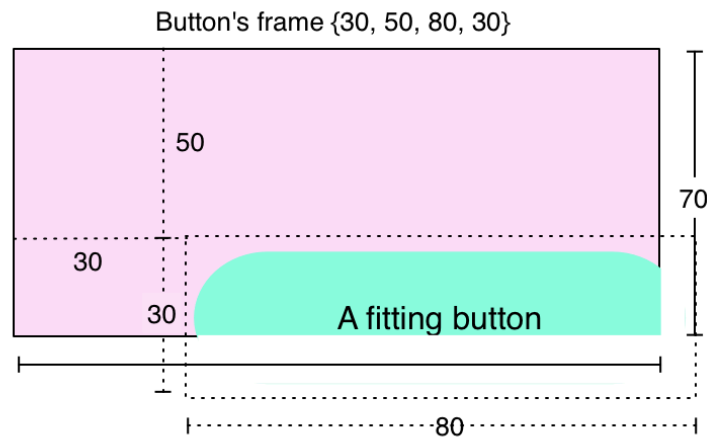
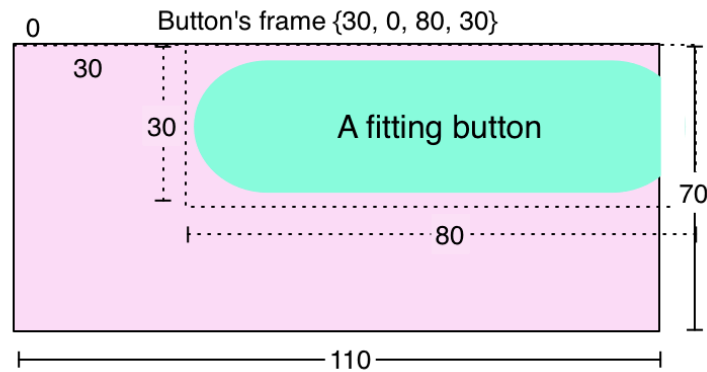
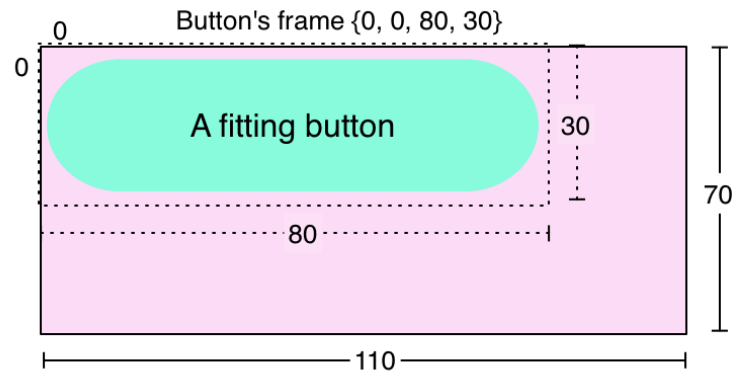
*creates this image*



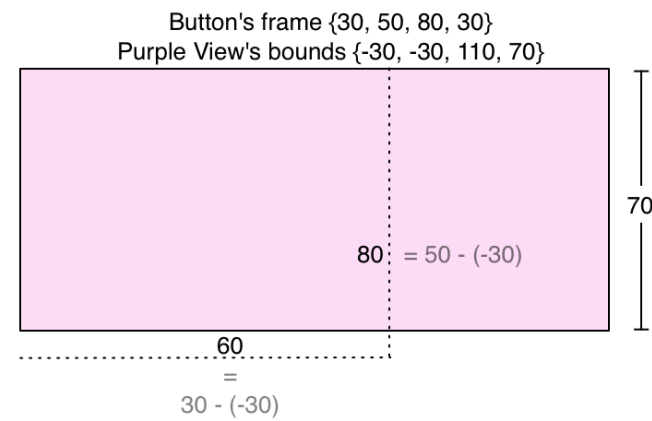
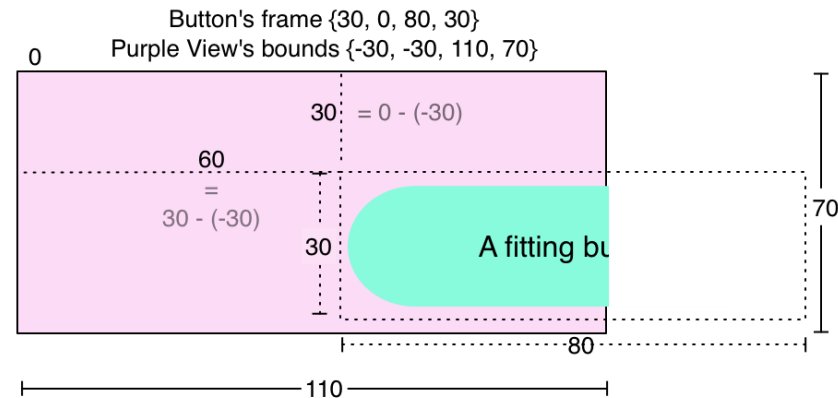
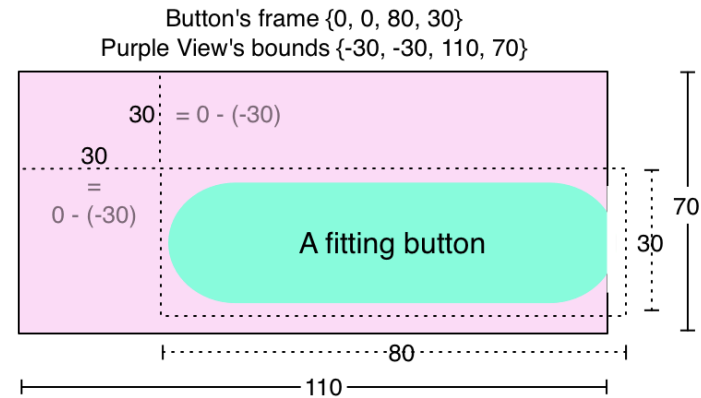
# Kompozice view



# Kompozice view

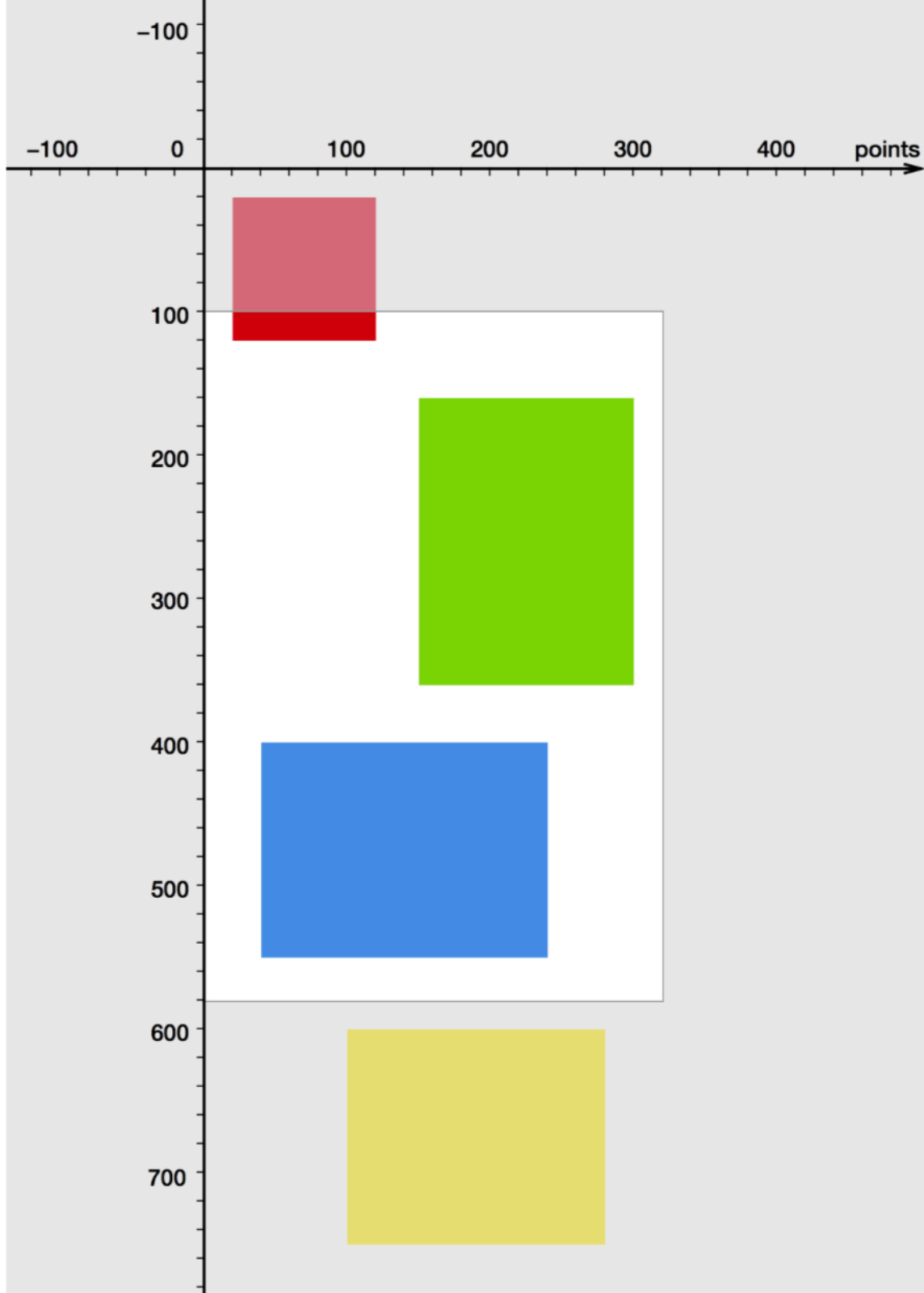


# Kompozice, superview bounds



# Jak pracuje *UIScrollView*

- Potřebujeme *View* s rozměrem přesahujícím rozměr obrazovky. Typicky *TableView*, *ImageView*, *CollectionView*, ...
- *ScrollView.ContentView* a subviews:
  - různě umístěné
  - scrolling
  - dynamika



# Komunikace Controller <-> View

- Controller (+View) vyjadřuje stav UI. (jako VM)
- Controller chce *změnit stav* svého view.
  - Tok událostí od Model (Controller) do View.
- View detekuje uživatelskou událost (gesto, klávesnice, rotace), volá controller.
  - Target-action. Objektu X pošli zprávu Y.
  - Typicky směřováno do Controlleru.



# Stav objektů View

- Uvažujme *var o: UILabel* (property: text).
- V proceduře kontroleru chceme změnit obsah *UILabelu* (prováděno nějakým vláknem).
  - *o.text = newValue;*
  - *o.setNeedsDisplay()*
- Změny obsahu UI se provádí pouze vláknem zpracovávajícím *MainQueue*!

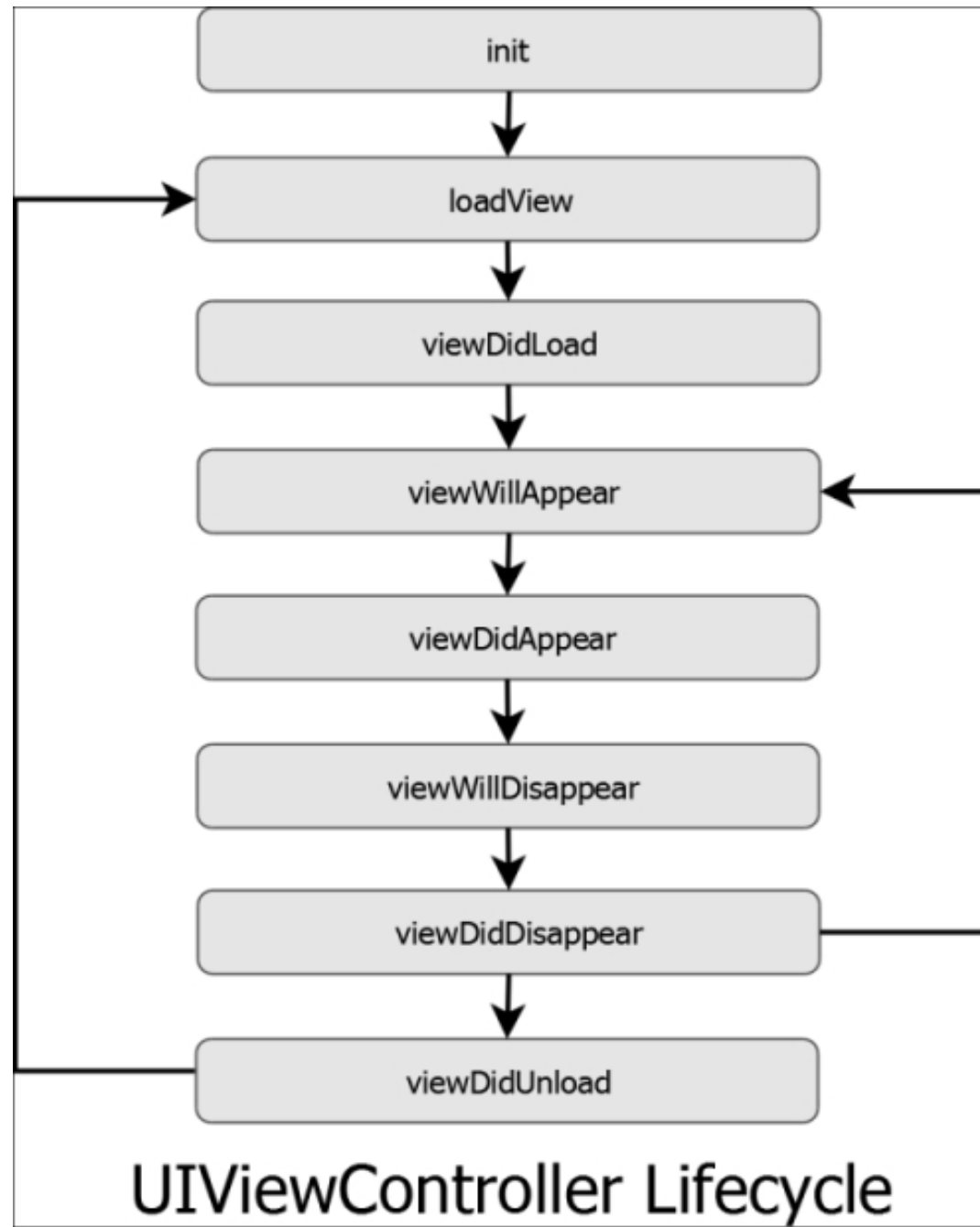
# Stav objektů view

- Když uživatel změní stav objektu *view*, *view* posílá zprávu svému "target" (target-action mechanism). Cílem je typicky controller.
- *UISwitch*, *UITextField*, *UIButton*, ...
- Typicky:
  - `func nejakaZmenaTlacitka(sender: UISwitch);`
  - `@IBAction`
  - přechod na closures.

# MVC Stav UI

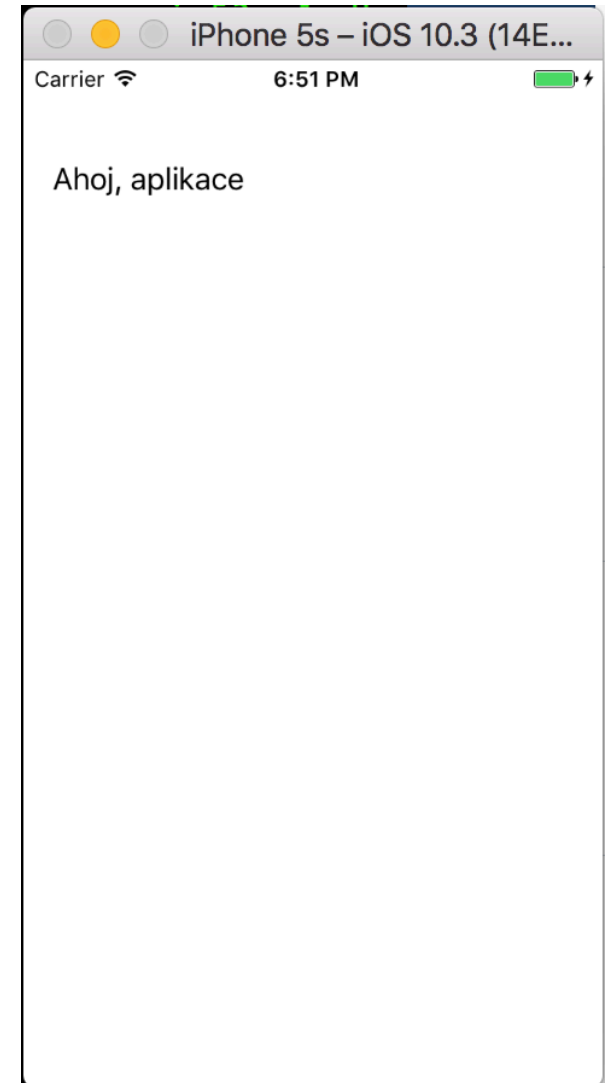
- VC je vlastníkem View. (následně subviews).
- UIWindow — key and visible. (UIResponder)
- ViewController — předávání řízení (toku událostí z iOS).
- Přechod od jednoho VC k druhému.
  - Předání toku událostí.
  - Změny v topologii Views.
- Životní cyklus ViewControllerů.

# Životní cyklus ViewController



# M-V-C, Základní demo

```
class MujModel {  
    var obsah : String = "Ahoj, aplikace"  
}  
  
class ViewController: UIViewController {  
    @IBOutlet var lei : UILabel?  
  
    let mujmodel = MujModel()  
  
    override func viewDidLoad() {  
        super.viewDidLoad()  
  
        //  
        lei?.text = mujmodel.obsah  
    }  
}
```



# Forma referencování Outlets

- Vlastníkem Outletu je View, ne VC.
  - (Co se stane, když outlet odstráním ze superView?) setHidden.
- @IBOutlet var textik : UILabel!
  - strong reference s garantovanou / předpokládanou hodnotou
- @IBOutlet var textik : UILabel?
  - je v XCode výchozí. Zřejmě reaguje na lazy loadView.
  - @IBOutlet weak var textik : UILabel?
- Totálně v rozporu s myšlenkou Swiftu.

# Co se stane po `lei?.text = "..."`

- Předpokládejme *mainThread*.
  - Jinak kritická chyba. Stále: jaké vlákno vykonává tento kód?
- Nastaví se datový obsah `UILabel.text`, setter.
- Nastaví se příznak "`needsDisplay`".
  - Propaguje se směrem k super-views až po `UIWindow`.
- V dalším cyklu `RunLoop` vyvolá překreslení `UIWindow`.
  - tj. `UILabel.setText` neprovádí okamžitou změnu obrazovky!

# M-V-C, KVO verze

```
//
class MujModel : NSObject {
    dynamic var obsah : String = "Ahoj, aplikace"
}
//
class ViewController: UIViewController {
    @IBOutlet var lei : UILabel?
    let mujmodel = MujModel()
    //
    override func viewDidLoad() {
        super.viewDidLoad()

        lei?.text = mujmodel.obsah

        mujmodel.addObserver(self, forKeyPath: #keyPath(MujModel.obsah), options:
[.new, .old], context: nil);
    }
    //
    override func observeValue(forKeyPath keyPath: String?,
                                of object: Any?,
                                change: [NSKeyValueChangeKey : Any]?,
                                context: UnsafeMutableRawPointer?)
    {
        self.lei?.text = self.mujmodel.obsah
    }
}
```



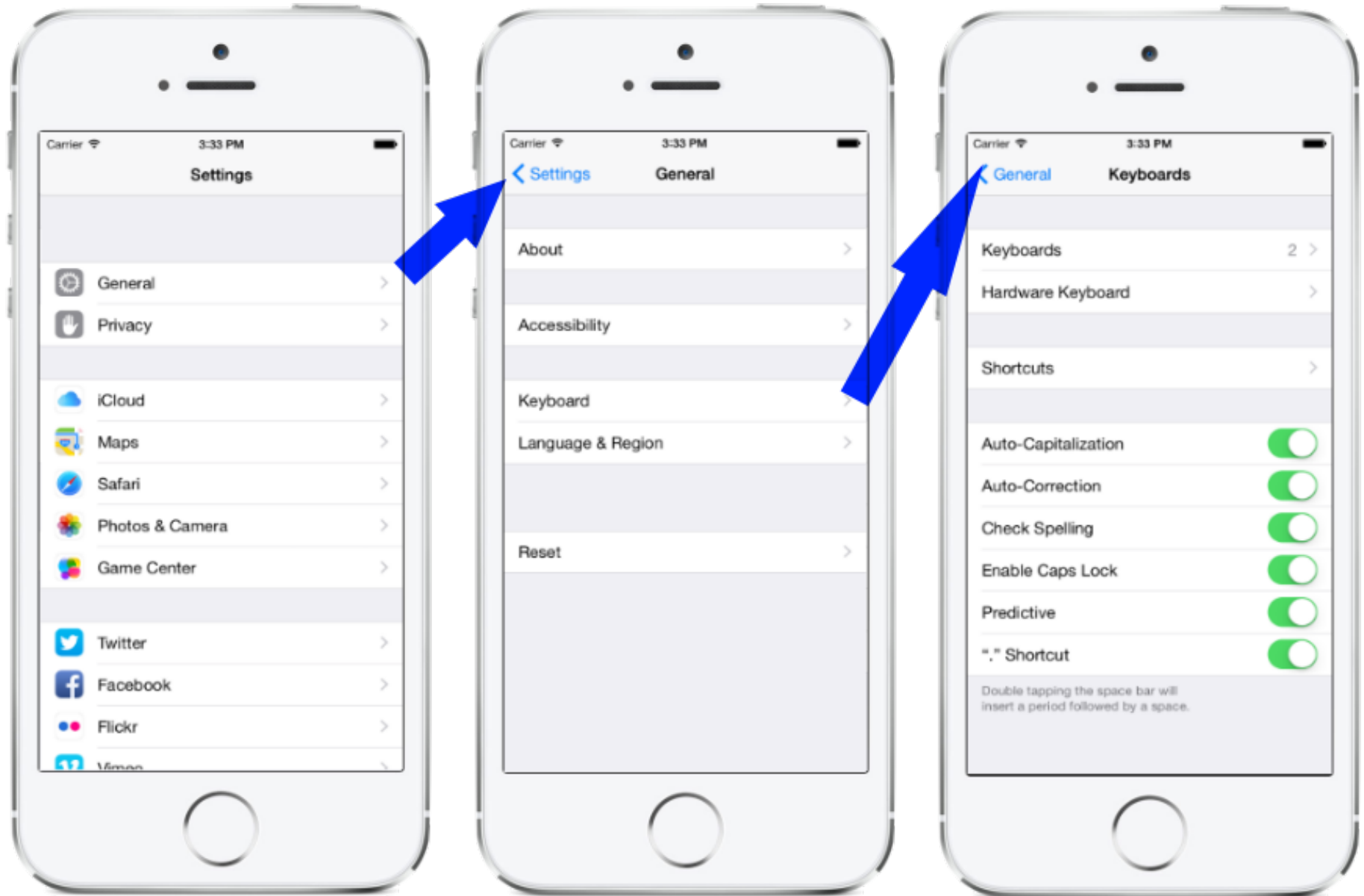
# M-V-C, didSet verze, obezřetně!

```
class MujModel {  
    //  
    weak var vc : ViewController? = nil  
  
    var obsah : String = "Ahoj, aplikace" {  
        didSet {  
            // bacha na vlakno  
            vc?.lei?.text = obsah;  
        }  
    }  
}  
  
class ViewController: UIViewController {  
    //  
    @IBOutlet var lei : UILabel?  
    //  
    let mujmodel = MujModel() ;  
    //  
    override func viewDidLoad() {  
        super.viewDidLoad()  
        //  
        mujmodel.vc = self;  
        //  
        lei?.text = mujmodel.obsah  
    }  
}
```

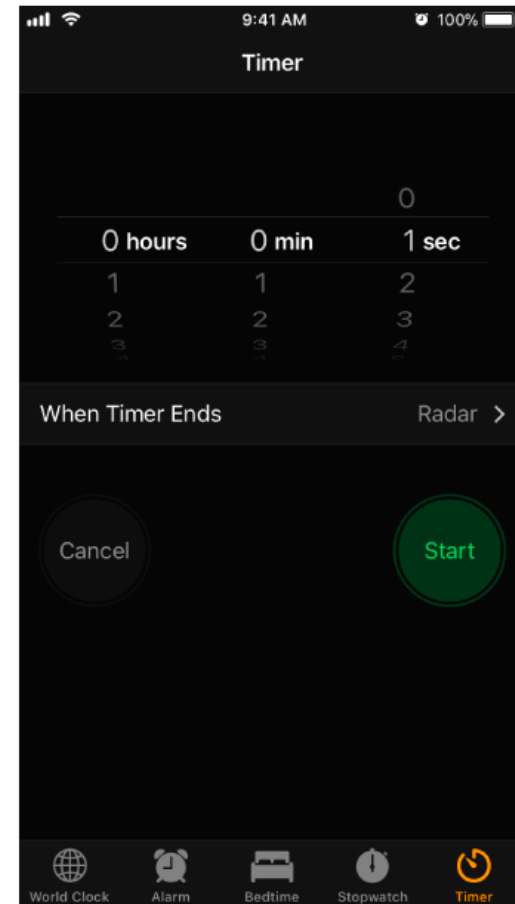
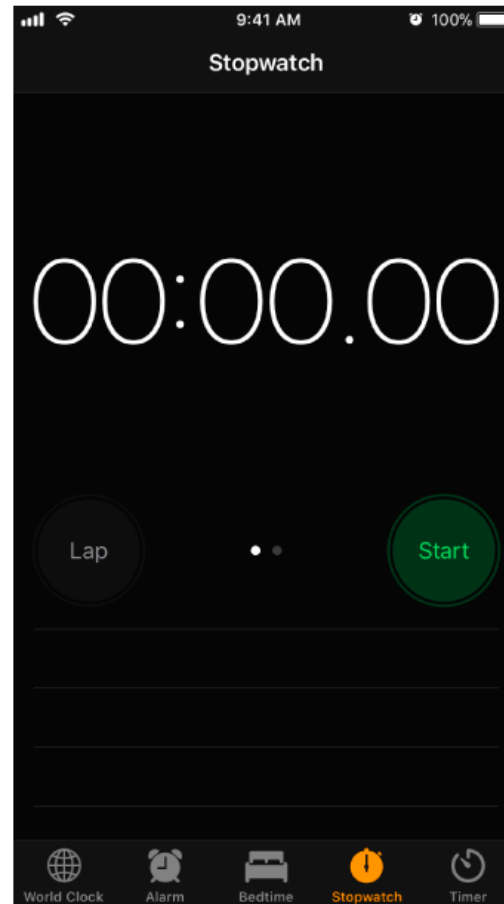
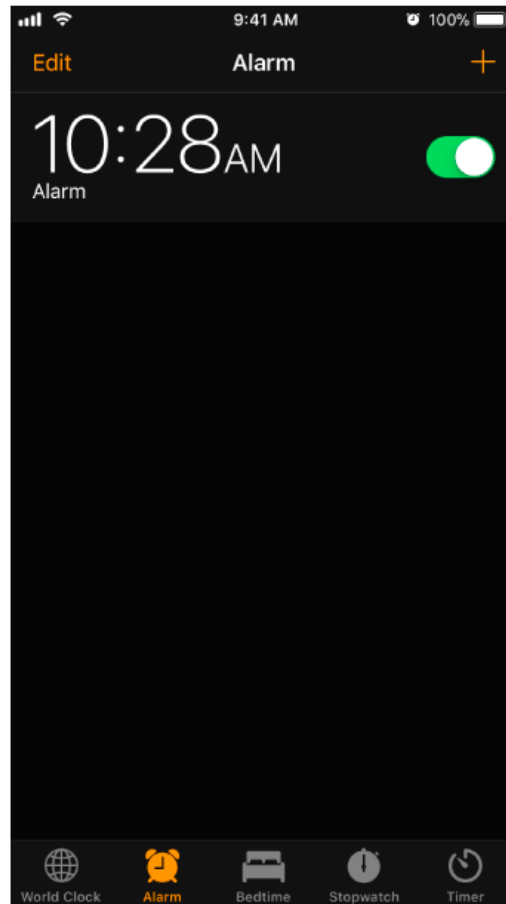
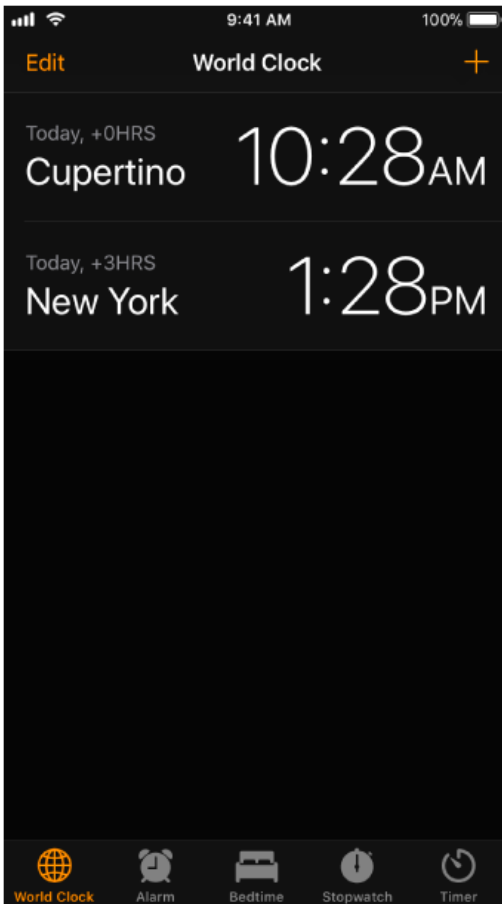
# Multi-VC aplikace

- Potřebujeme “přepínat obrazovky”, tj. potlačit jeden VC a aktivovat druhý VC.
  - Změna v hierarchii views, přesměrování VC.
  - Abstraktní VC — pouze řídí další (vnořené) VC.
- *UINavigationController*,
- *UITabBarController*,
- Styl *Master-Detail* — 2 VC spojené přes *UISplitViewController*.

# Navigation VC

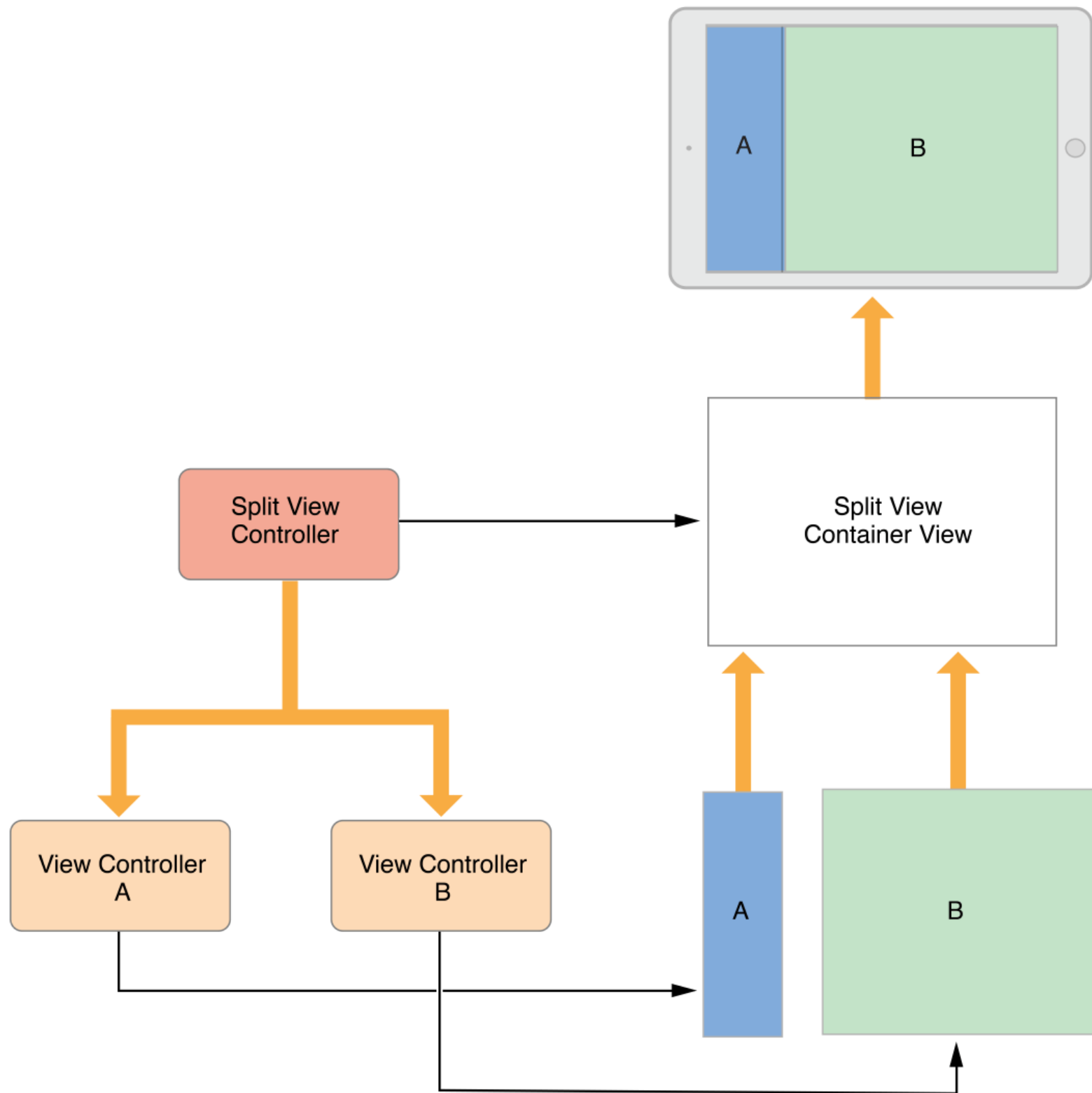


# TabBar VC



# Funkce Navigation-VC

- Tzv. "container VC". Má:
  - `rootViewController` — základní/počáteční VC.
  - `viewControllers : [UIViewController]`
  - `contentView` — View pro vnitřní obsah.
- *`pushViewController(o:VC, animated:Bool)`*
- *`popViewController(animated:Bool)`*
- Obdobně s `TabBarVC`.



# Přechod na druhý VC

- Deaktivace původního vc : UIViewController:
  - `vc.willMove(toParentViewController: nil)`
  - `vc.view.removeFromSuperview()`
  - `vc.removeFromParentViewController()` — preposílání zpráv.
  - `(vc.viewWillAppear, vc.viewDidAppear)`.

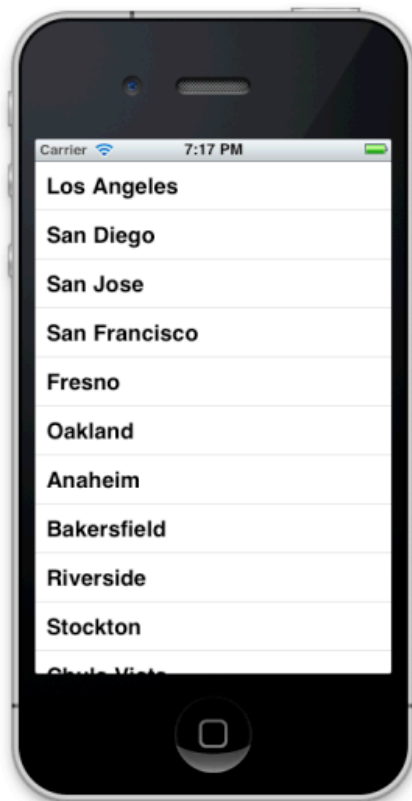
# Přechod na druhý VC

- Aktivace nového VC, nvc:
  - addChildViewController(nvc) — preposílání zpráv.
  - view.addSubview(nvc.view)
  - nvc.view.frame = view.bounds
  - nvc.view.autoresizingMask = [.flexibleWidth, .flexibleHeight]
  - nvc.didMove(toParentViewController: self)
- Nav-VC a TabBarVC takto přepínají VC.

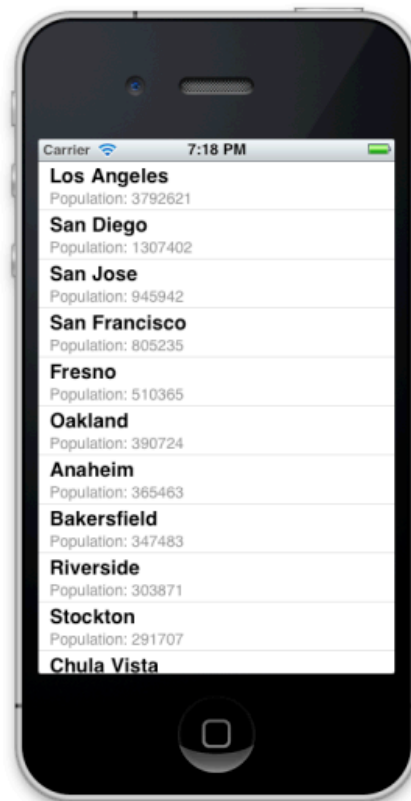


# Table View Controller

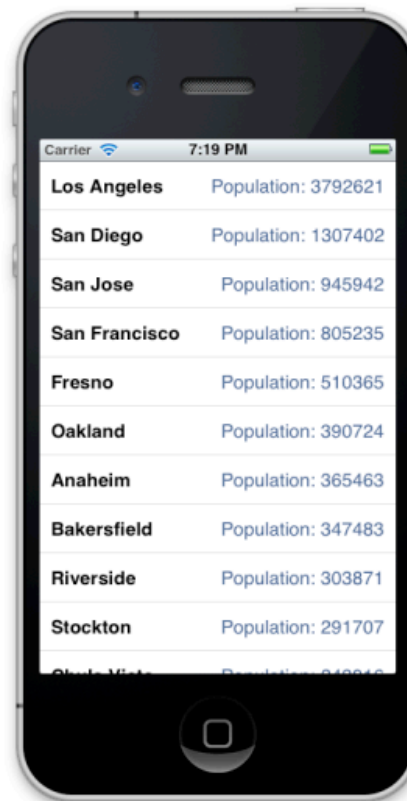
- Nejdůležitější prvek UI iOS.
- Seznam buněk (TableViewCell).



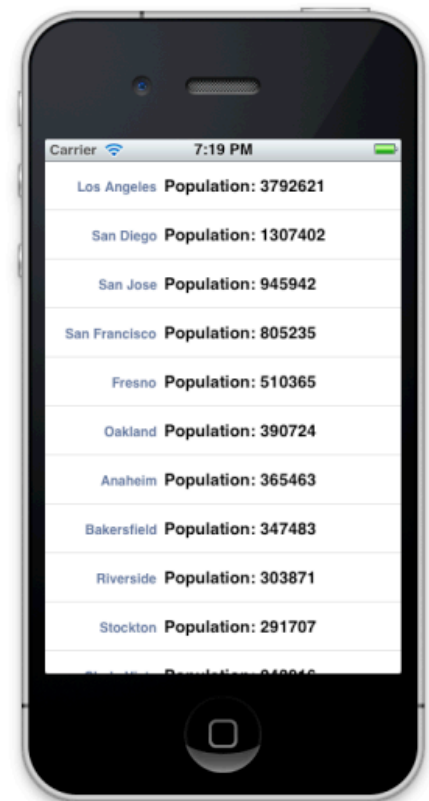
UITableViewCellStyleDefault



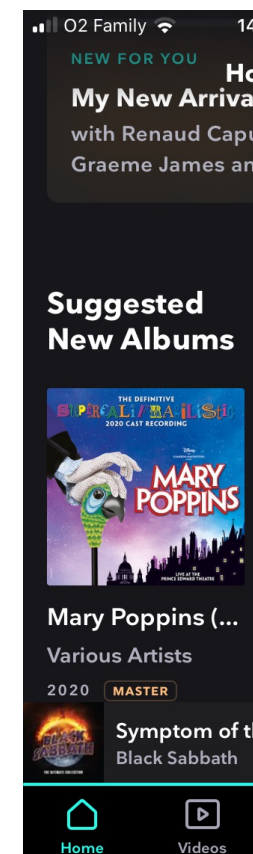
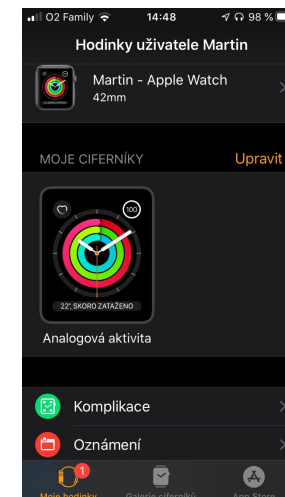
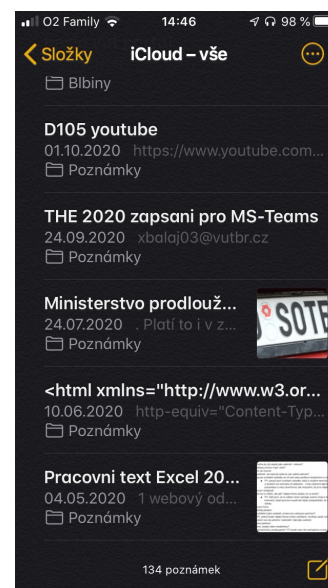
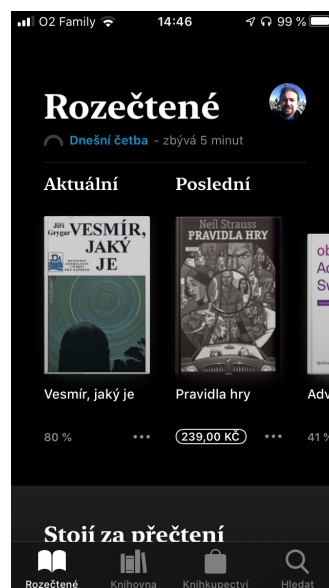
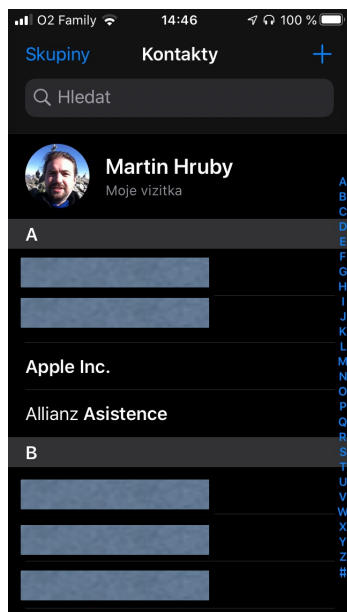
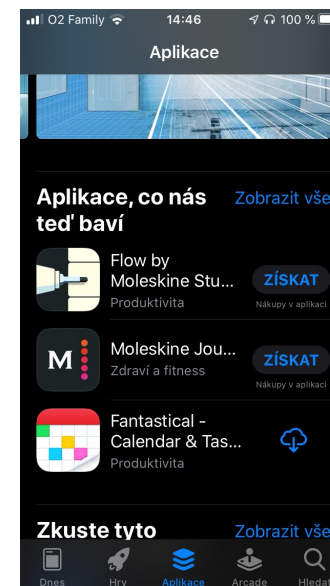
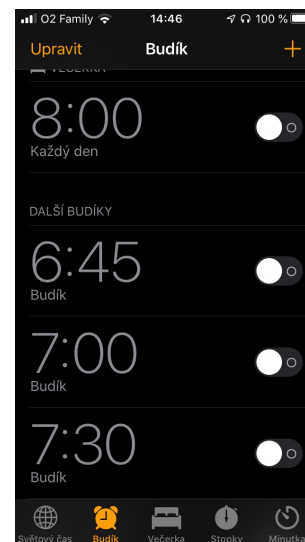
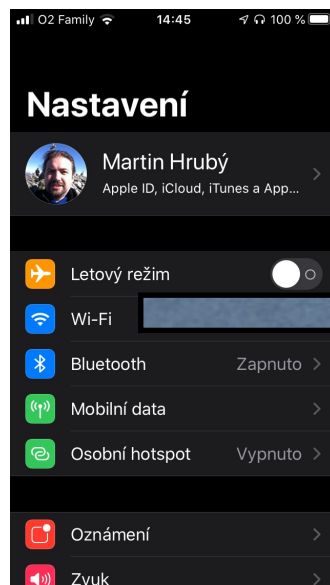
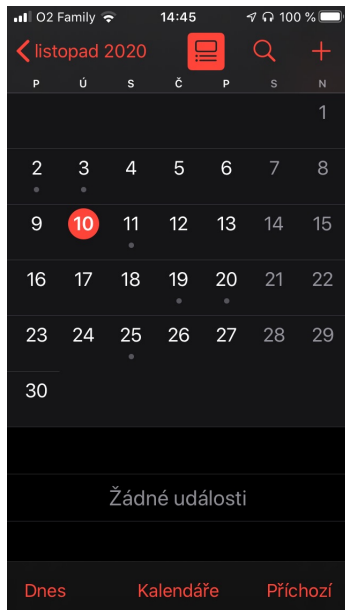
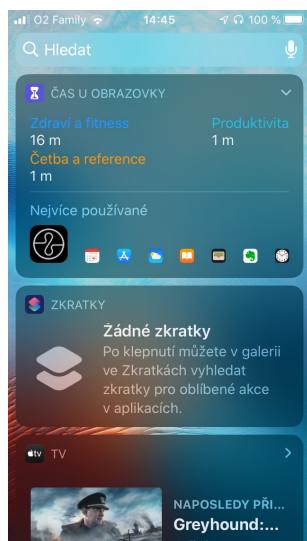
UITableViewCellStyleSubtitle



UITableViewCellStyleValue1



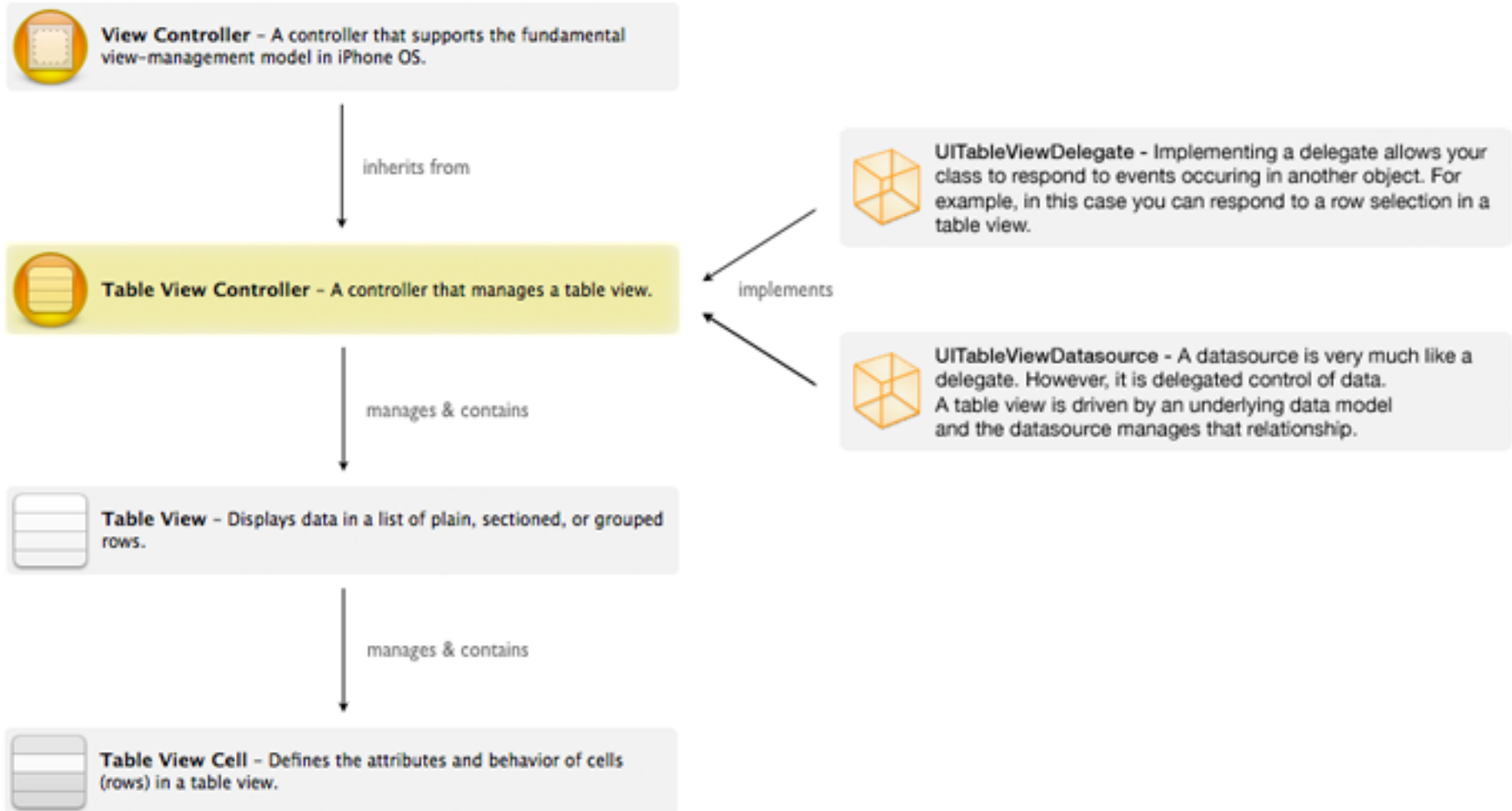
UITableViewCellStyleValue2



# Tabulka jako koncept

- *TableView* — *View*, dotazuje se přes *dataSource* / *delegate*.
- *TableViewDataSource* — struktura tabulky, obsah.
- *TableViewDelegate* — geometrie, události, povolení k ...
- *TableViewController* — VC implementující *dataSource* a *delegate*.
- Zapojení *TableView* v aplikačním VC:
  - Aplikační VC odvozuje od TVC, nahrazuje vybrané metody *dataSource* / *delegate*.
  - Aplikační VC má TV jako *subView* a tvoří *dataSource*. Co *delegate*?

# UITableViewController

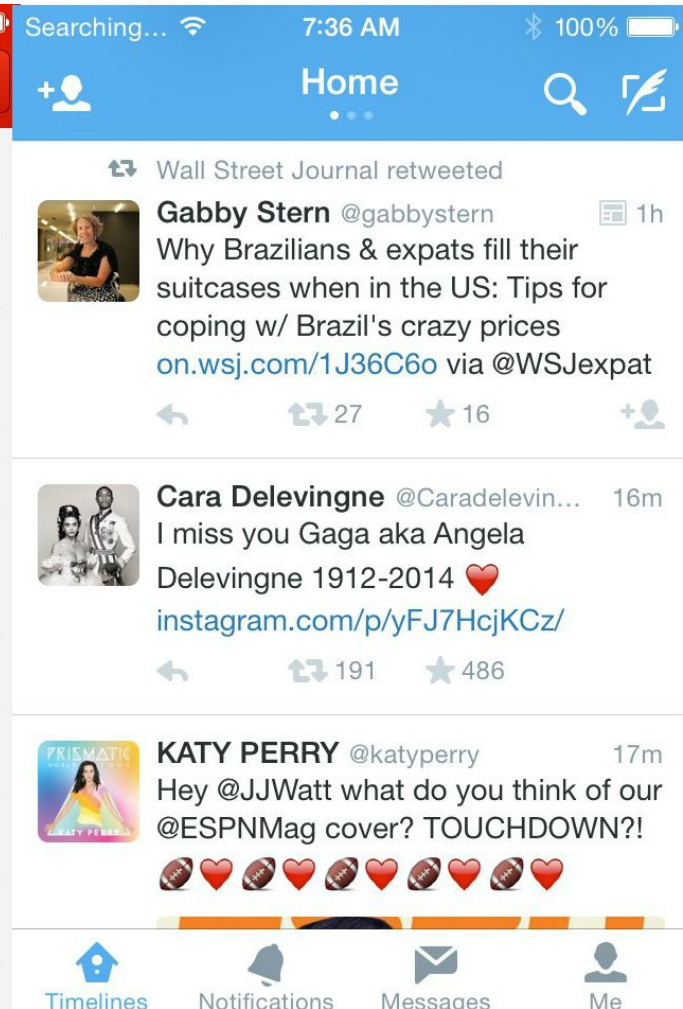
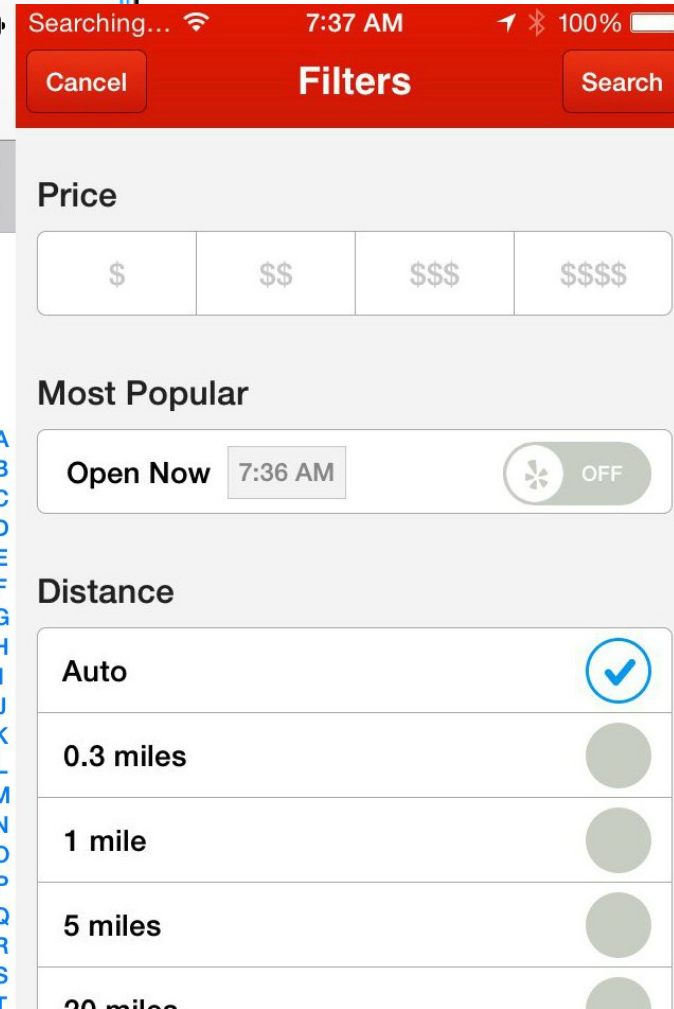
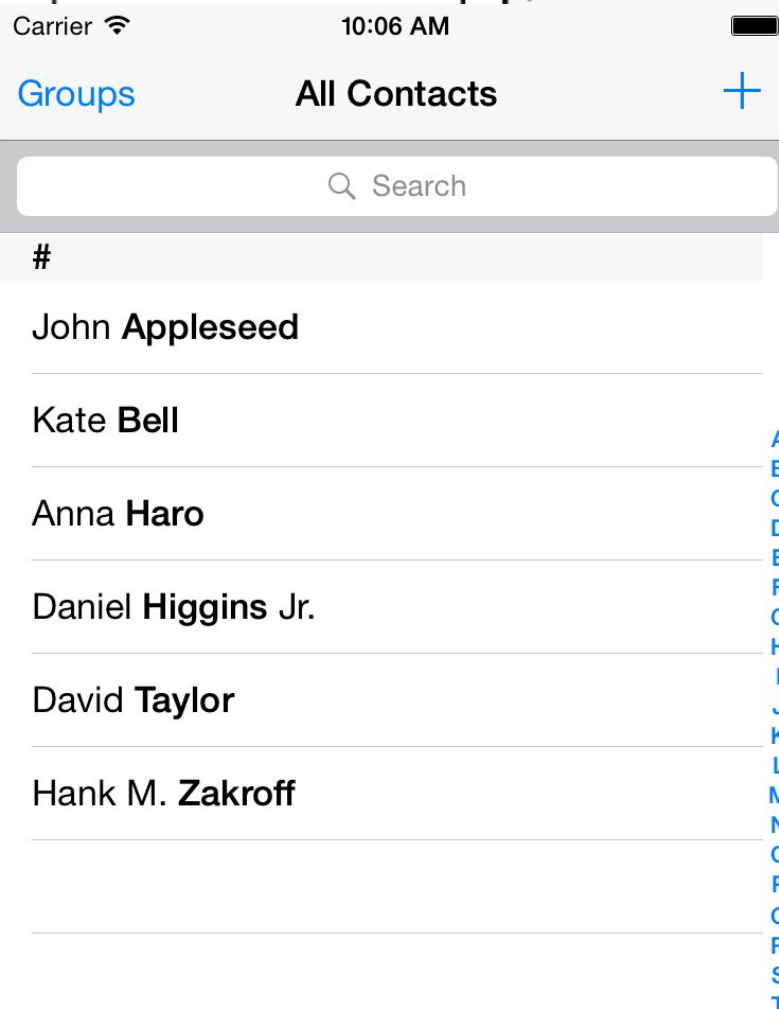
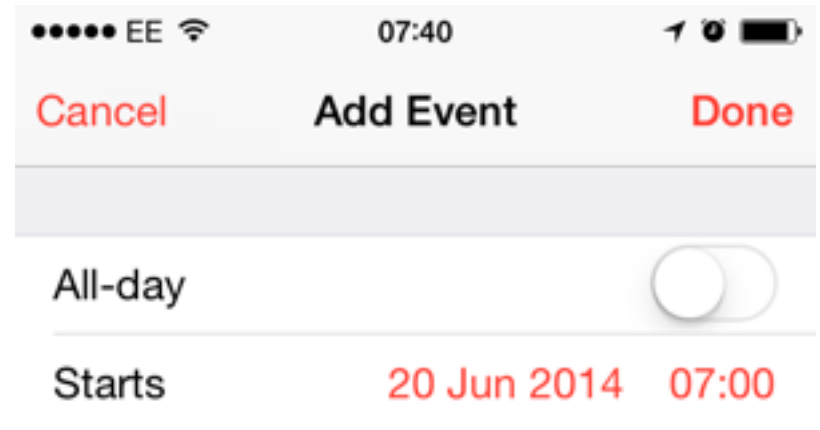
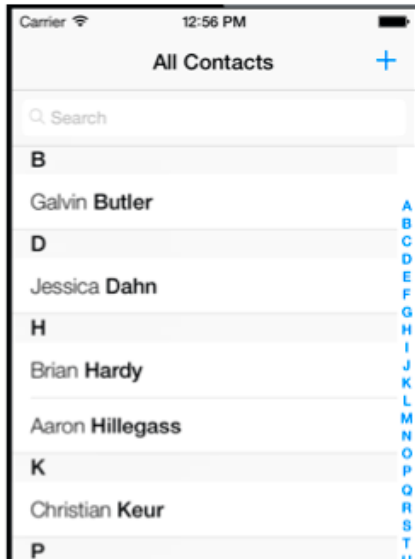
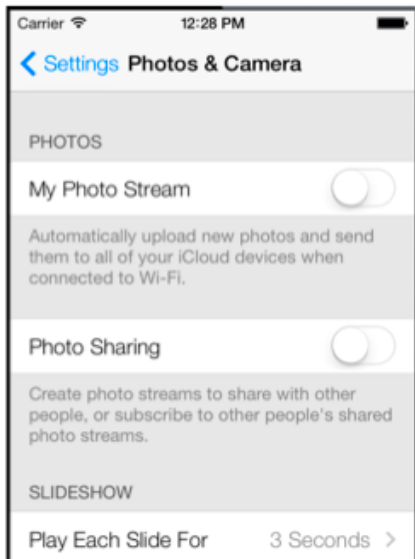


# UITableViewController

- Implementuje dataSource a delegate protokoly tabulky.
- Uživatelský VC dědí z TVC, přepisuje potřebné metody dataSource a delegate protokolů.

# UITableView

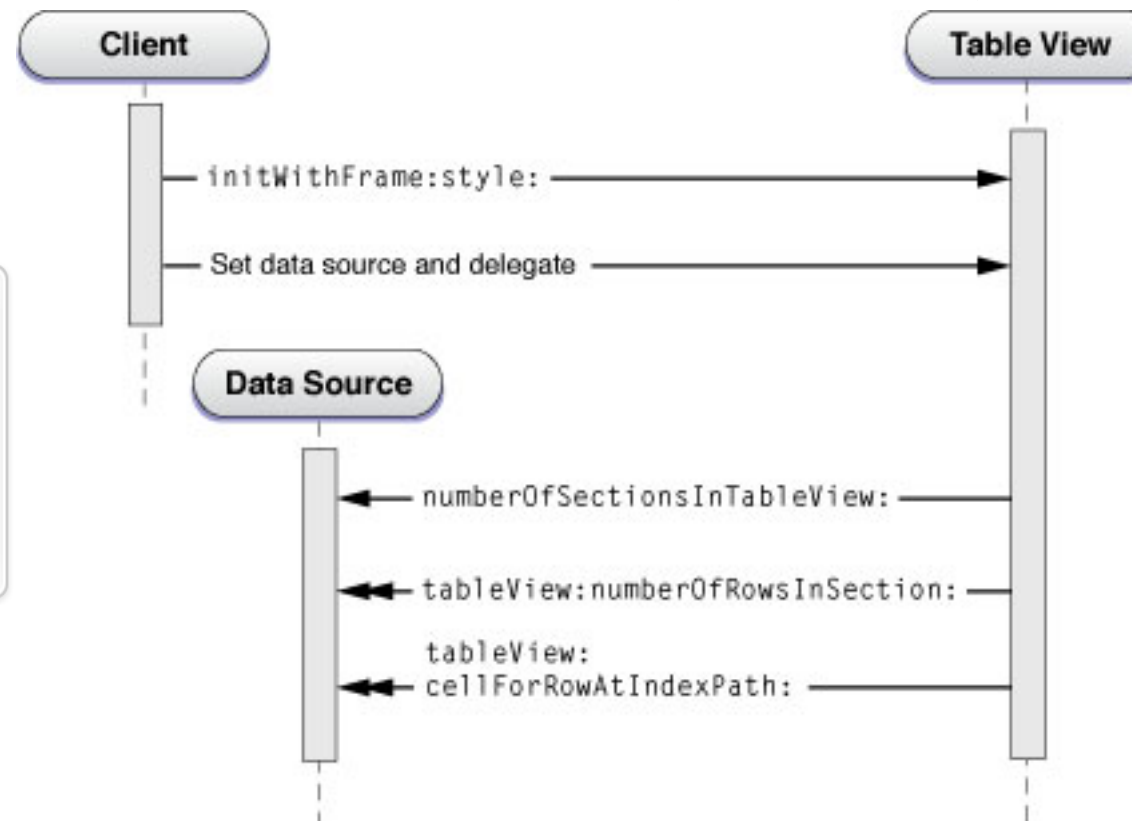
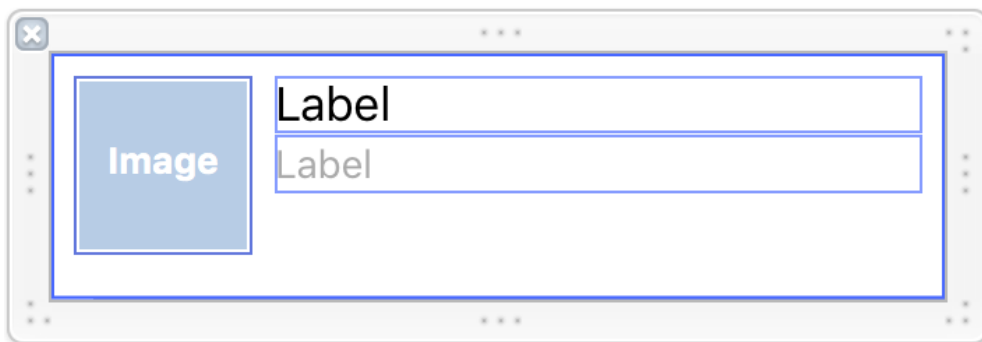
- Odvozen od UIScrollView (přesahuje rámec obrazovky).
- Provádí rozmístění (layout) buněk (Cell) ve skupinách.
- Obecný heterogenní dynamický soupis buněk vertikálně řazených.
- UITableViewCell — prototyp buňky tabulky





# Řízení — dataSource

- TVC zprostředkovává obsah (Model) do View.
- Model zde vystupuje jako "dataSource" API:
  - počet sekcí, počet řádků v sekci.
  - sestavení TableViewCell pro zadanou souřadnici (IndexPath).

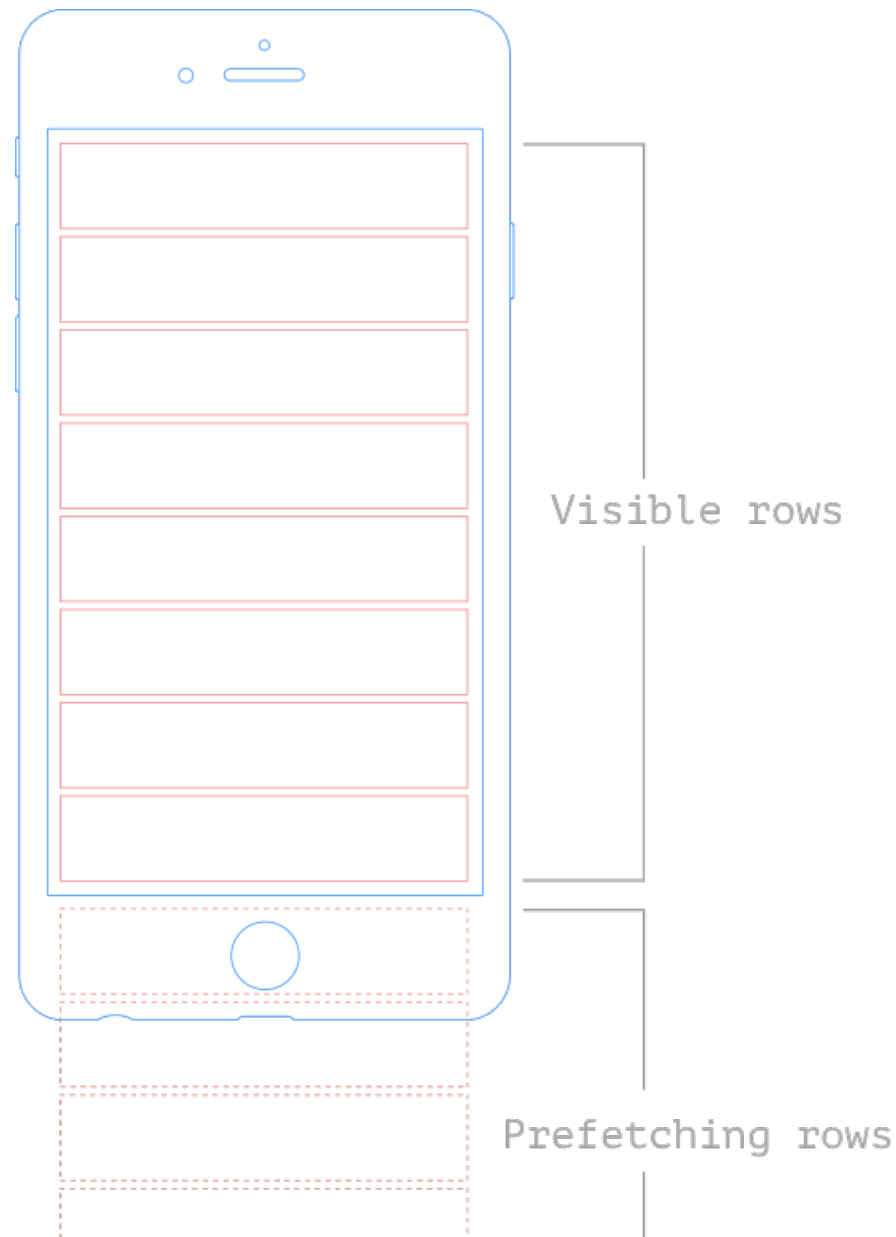




# Řízení TV, dynamika

- TV zjistí datový rozsah (sekce, řádky).
- Předpokládá se, že viditelná je pouze část buněk.
- Ty jsou alokovány a inicializovány.
- Při posuvu tabulkou se dynamicky volá dataSource na doplňování obsahu buněk.
- Znovupoužití objektů buněk (pool). Identifikátory buněk.

# Dynamika přístupu na dataSource



# Demo

```
class SimpleTab: UITableViewController {  
    var seznam = ["Jeden", "Druhy", "Treti"];  
  
    override func tableView(_ tableView: UITableView,  
        numberOfRowsInSection section: Int) -> Int  
    {  
        return seznam.count  
    }  
  
    override func tableView(_ tableView: UITableView,  
        cellForRowAt indexPath: IndexPath) -> UITableViewCell  
    {  
        let cell = tableView.dequeueReusableCell(withIdentifier: "cell",  
for: indexPath)  
  
        cell.textLabel?.text = seznam[indexPath.row]  
  
        return cell  
    }  
}
```

# Dynamika, posuv v Table

- Chod aplikace musí být hladký.
- Inicializace buněk může brzdit chod tabulky (rychlý posuv).
- Při náročnější inicializaci se přechází do bočních vláken (GCD).

```

override func tableView(_ tableView: UITableView,
                        cellForRowAt indexPath: IndexPath) ->
UITableViewCell
{
    let cell = tableView.dequeueReusableCell(withIdentifier: "tabik", for:
indexPath)

    cell?.imageView?.image = placeholder;
    // do vedlejsiho/globalniho vlakna naplanuju...
    DispatchQueue.global().async {
        // sestaveni obsahu pro bunku tabulky
        if let loaded = myModel.load(cosi) {
            // pokud mame obsah, jdeme zpatky do mainThread
            DispatchQueue.main.async {
                // pokud na indexPath stale existuje bunka
                if let ncell = tableView.cellForRow(at: indexPath) {
                    // aktualizuji
                    ncell.imageView?.image = loaded;
                }
            }
        }
    }
    //
    return cell;
}

```

# Závěr

- Exkurze do světa UIKit / StoryBoard.
- Některé koncepty ve SwiftUI přetrvávají.
- Příště:
  - Konstrukce typických částí v aplikaci SwiftUI.
  - ObservableObjects.
  - Úvod do Combine.