

Konstrukce aplikace v iOS

—

MVC

IZA, Martin Hrubý, FIT VUT, 2019

Literatura

- Zdroje Apple.
- Web: objc.io
- Ole Begeman: Understanding UIScrollView

Úvod

- Další koncepty Swiftu budeme probírat v kontextu aspektů programování aplikací.
- Cílem je poznat základy konstrukce aplikace pro iOS.
- Přehled View-Controllerů (VC).
- UIKit — knihovna pro psaní iOS / tvOS app.
 - Swift Standard Library. Foundation. (Cocoa Touch, Obj-C).

XCode

- IDE pro programování aplikací.
- Editor pro interaktivní návrh konceptu obrazovek aplikace (Storyboard).
 - Programování s / bez Storyboardu.
- Editace rozvržení Views (geometrie) v okně apod.
- Xcode generuje základní rámce zdrojů aplikace.

První kontakt s programem

- XCode nabídne vygenerovat zdrojáky pro základní koncepty UI:
 - Single View, Tab bar, Master-Detail.
- Především generuje třídu pro UIApplication delegate.
 - Případně infrastrukturu pro CoreData a unit-testy.

Architektura aplikace

- UIApplication+delegate — API na jádro iOS.
 - Základy Views a geometrie views.
 - ViewControllers — řízení "jedné obrazovky" a řízení programu (Storyboard, Segue).
- M-V-C. M-VC.
 - V-C jsou platformově závislé (iOS / tvOS, macOS, watchOS).
 - Model — DB, komunikace vně, komunikace uvnitř.
 - Aplikace = Zobecnitelná funkcionalita (lib) + platformová specifika. **Koncepce v programu!**

Start aplikace

- Instance UIApplication, UIApplicationDelegate.
 - didFinishLaunchingWithOptions
- Instanciace UIWindow.
 - Počáteční ViewController — rootViewController.
 - UIWindow začne být "key and visible".
- Startovní výpočty — vedlejší vlákna.
- Co nejrychlejší start aplikace.
 - Obsah aplikace "dodělavat za běhu".

UIApplication

- Knihovnní třída. Nepředpokládá se odvozování vlastní. Singleton (*UIApplication.shared*).
 - Dědí od UIResponder — přijímá události z vnějšku.
- Rozhraní mezi OS a aplikací (události).
- Ovládá základní podobu aplikace v systému (ikonka, horní lišta apod.).
- Referencuje: *app-delegáta*, windows a `keyWindow`.

UIApplication

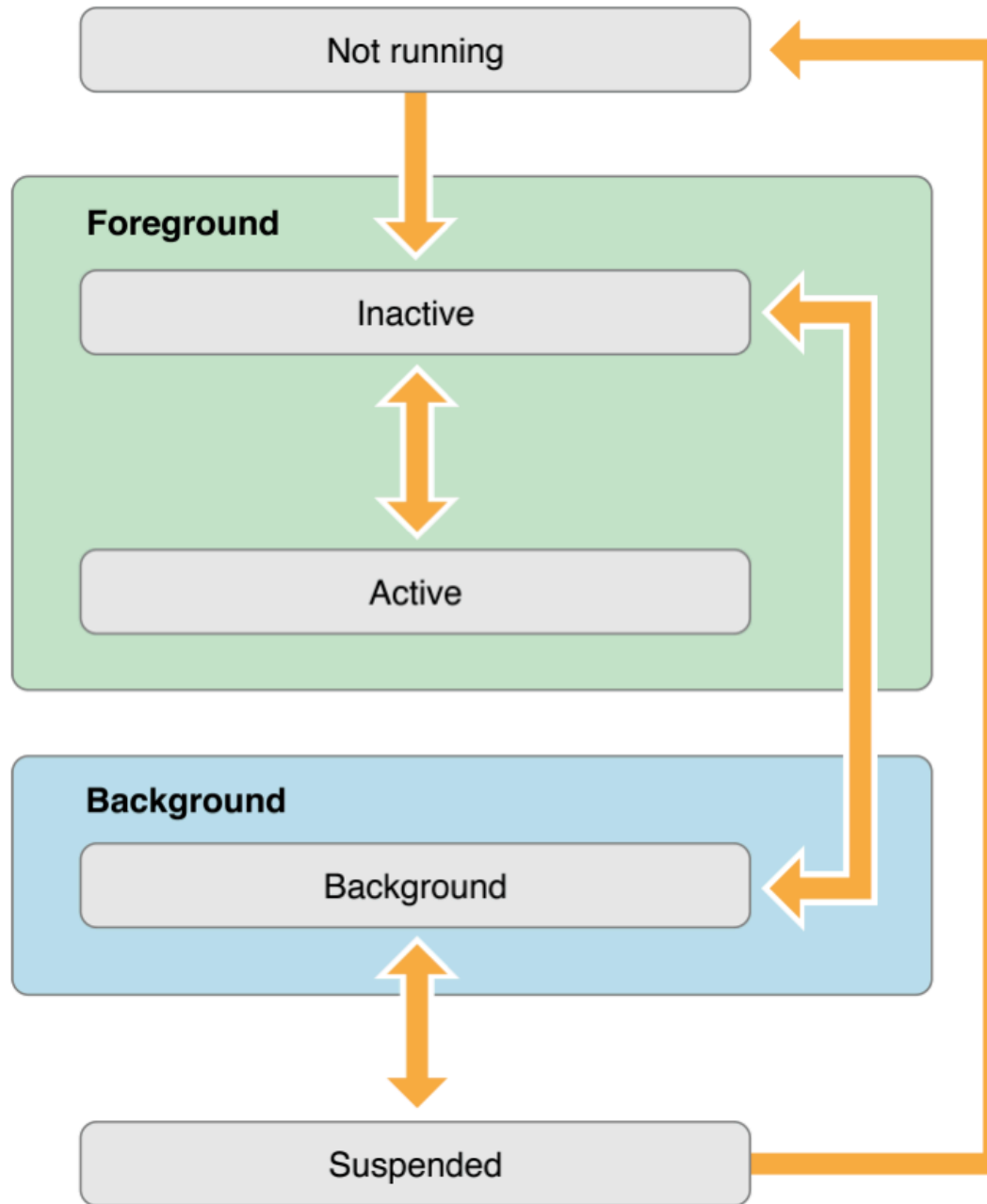
```
class UIApplication : UIResponder {  
    class var shared: UIApplication { get }  
    unowned(unsafe) var delegate: UIApplicationDelegate?
```

```
//  
func appMAIN() -> AppDelegate {  
    //  
    return UIApplication.shared.delegate as! AppDelegate  
}
```

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`,

Stavy aplikace



Stavy aplikace

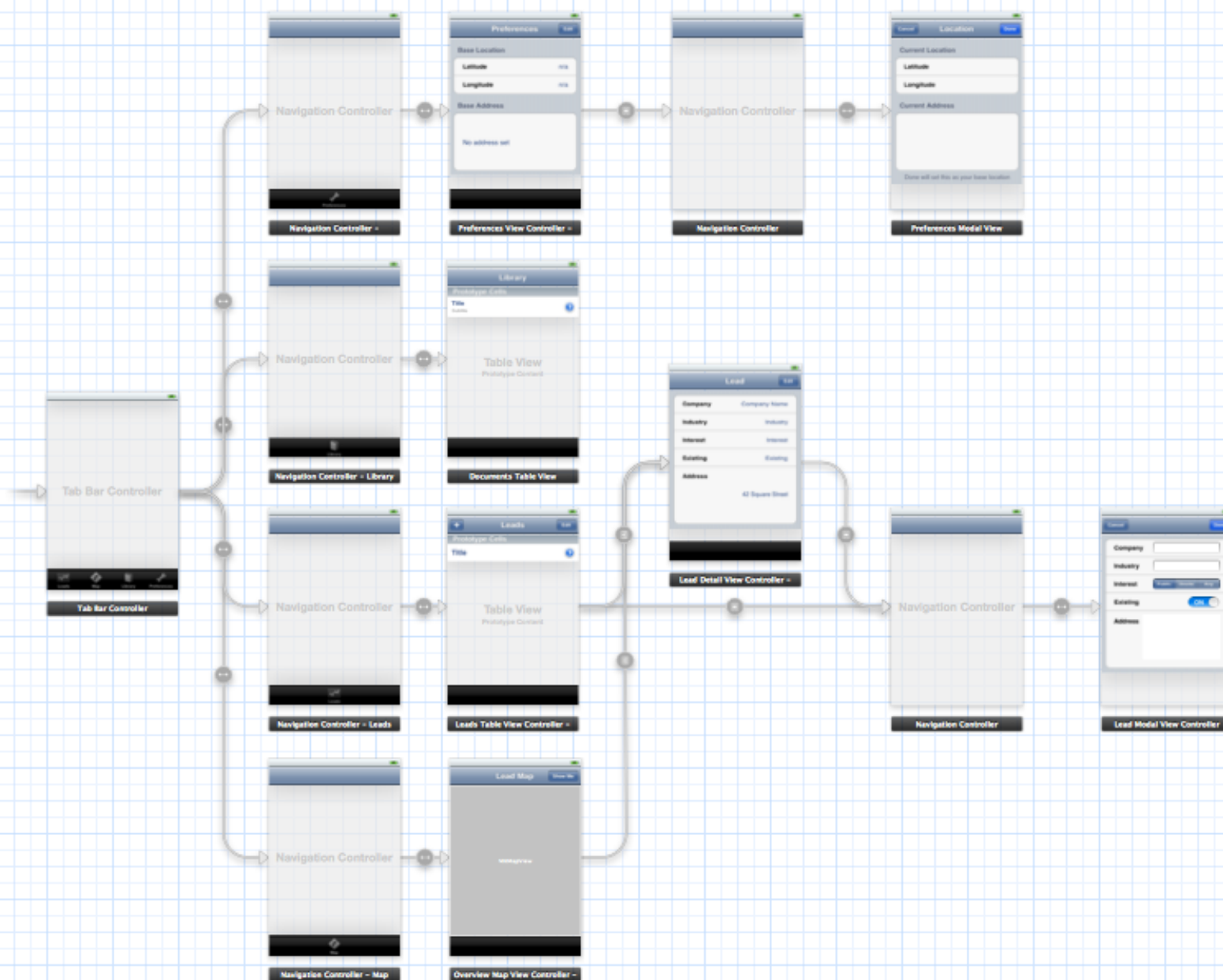
- *Not running* — nespouštěna nebo ukončena.
- Popředí (je viditelná):
 - *Active* — vykonává kód, přijímá události.
 - *Inactive* — nepřijímá události, běží. Typicky krátký moment.
- *Pozadí* (není viditelná): smí vykonávat kód.
- *Suspended* — je v paměti, neběží. Systém smí aplikaci kdykoliv ukončit.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

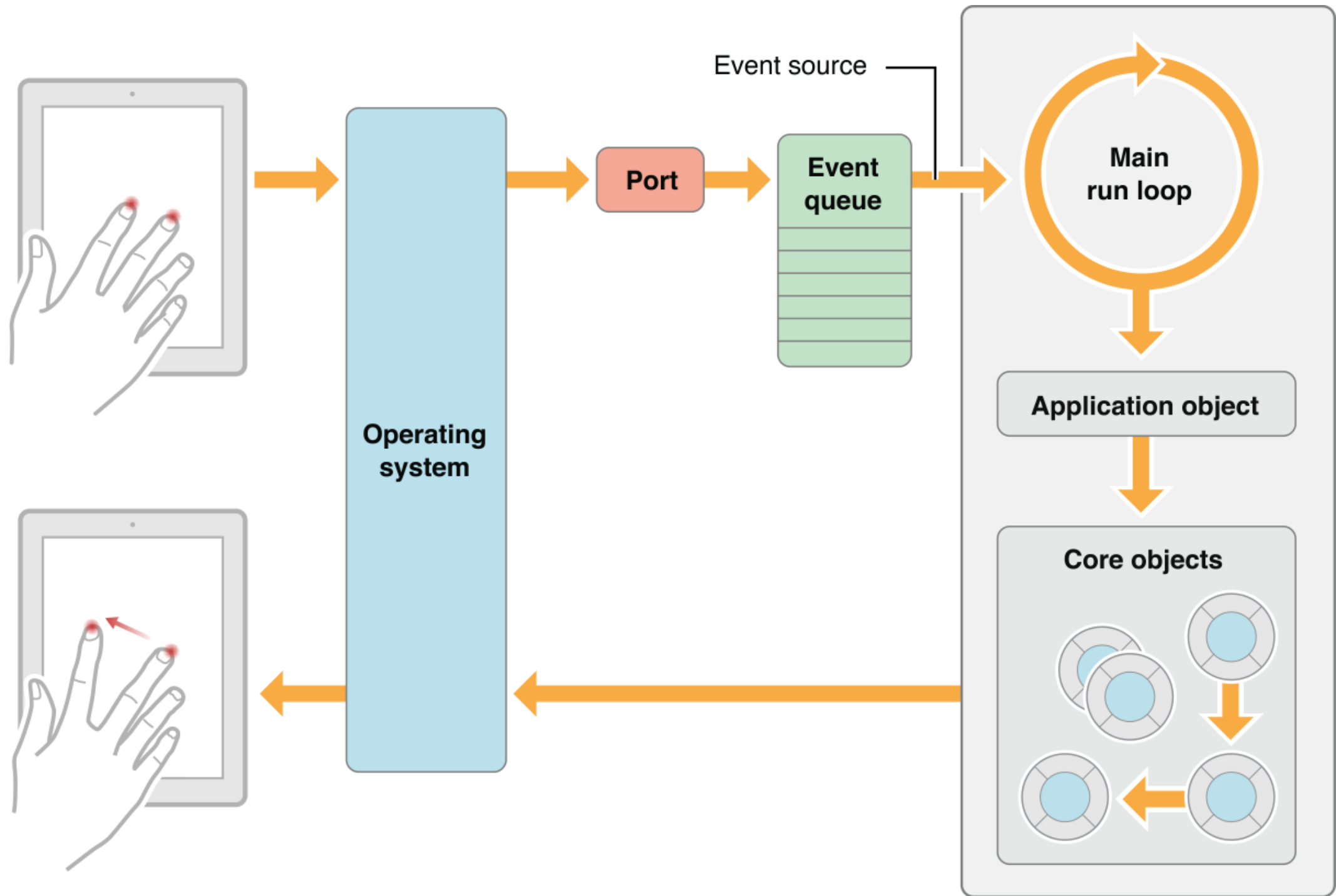
Systemové objekty aplikace

- *UIDevice* — metadata, orientace, baterie, typ zařízení (phone, pad, tv, carPlay).
 - multi-platformní aplikace (typicky iPad / iPhone).
- *UIScreen* — obrazovka zařízení.
 - Hlavní a sada vedlejších. Definuje "bounds" (orientace).
- *UIWindow* — kontejner pro *UIViews*.
 - *rootViewController*, přeposílá zprávy.
 - Další *UIWindow* — Alerts, (popovers).

Hierarchie vlastnictví views

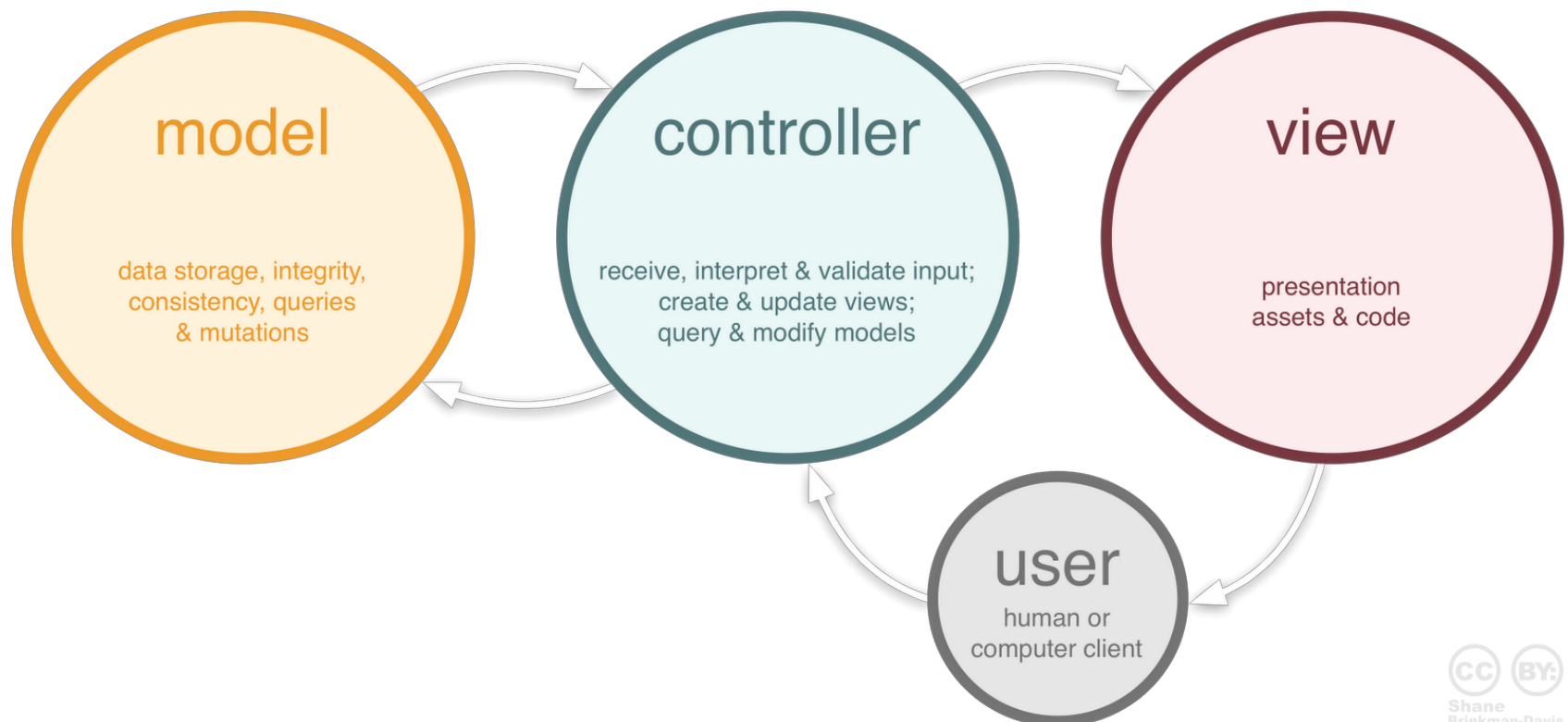
- *AppDelegate* vlastní *UIWindow*:
 - *UIWindow* vlastní *rootViewController*, referencuje *UIScreen*:
 - *rootVC* vlastní *UIView*
 - *UIView* má subviews, VC má "child" VC
 - ... hierarchická struktura aplikace a viewControllers
 - UINavigationController, TabBarVC, ...





Model-View-Controller (MVC)

- Model — datová část aplikace.
- View — zobrazovací prvky do View / Window.
- Controller — řídicí objekty.



MVC

- MVC koncept byl poprvé použit ve Smalltalku (Xerox) — dále okno, myš, ... návaznost na Mac.
- Jde o balík obecně více objektů (M, V, C).
- V systému MVC je reprezentantem tria M-V-C vždy Controller (jedna "obrazovka").
 - tj. ostatní objekty referencují Controller,
 - controller drží reference na view (vlastník) a model,
 - přecházíme od jednoho VC k druhému VC.

MVC v aplikaci

- Programujeme (typicky) Model a Controller.
- Je snaha automatizovat přesun dat z Modelu do Views (Bindings, macOS).
- Znovupoužitelnost ViewControllerů (VC).
 - dědičnost,
 - konektory na views (IBOutlets),
 - konektory na model.
- Typizované VC. Styly aplikace.

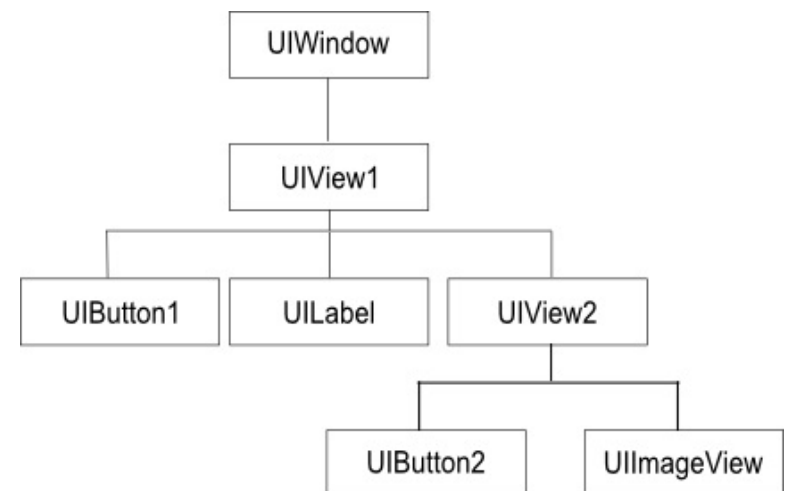
Views

- UIView, UILabel, UISwitch, UITableViewCell,
- Superview, subviews — hierarchie.
- Smyslem je skládat UI z knihovných komponent (versus drawRect:).
- 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()`
- 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.

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))`.
 - Alokují se prázdná bitmapa 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 rastrizace 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 $X.frame.origin - S.bounds.origin$.

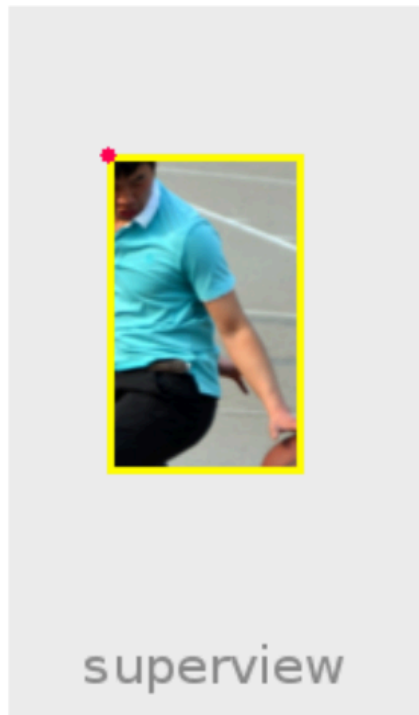
Frame

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

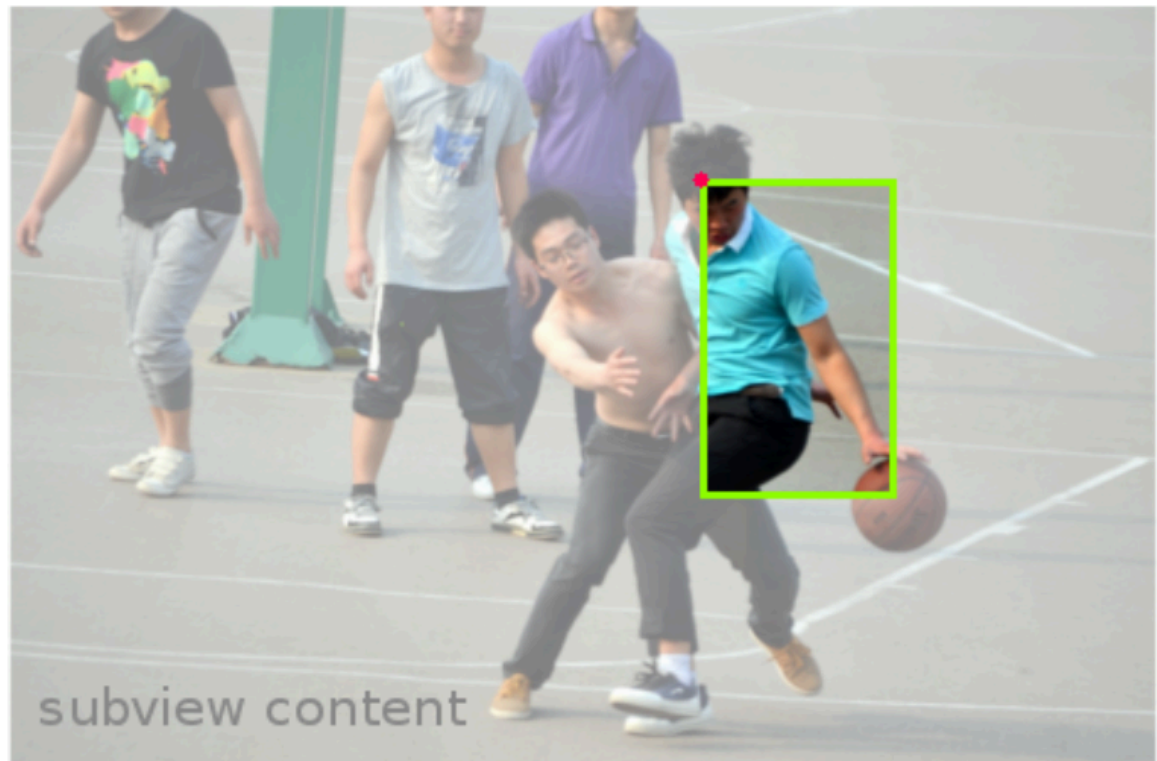
Bounds

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

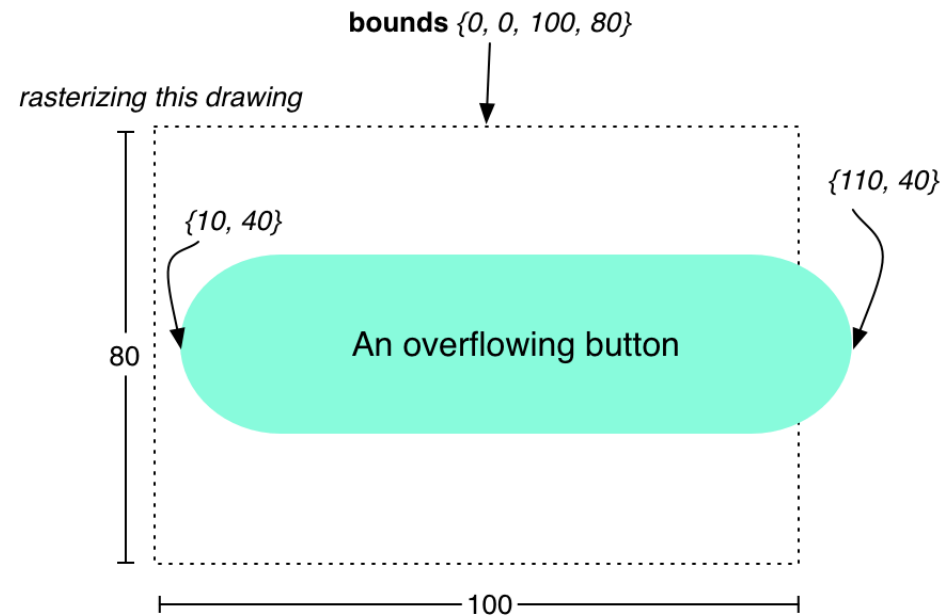
Frame



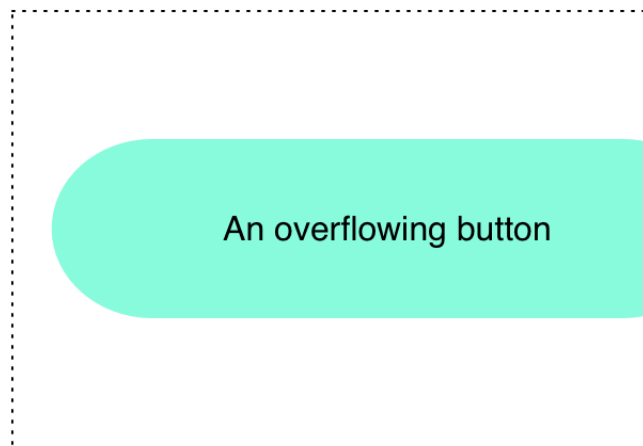
Bounds



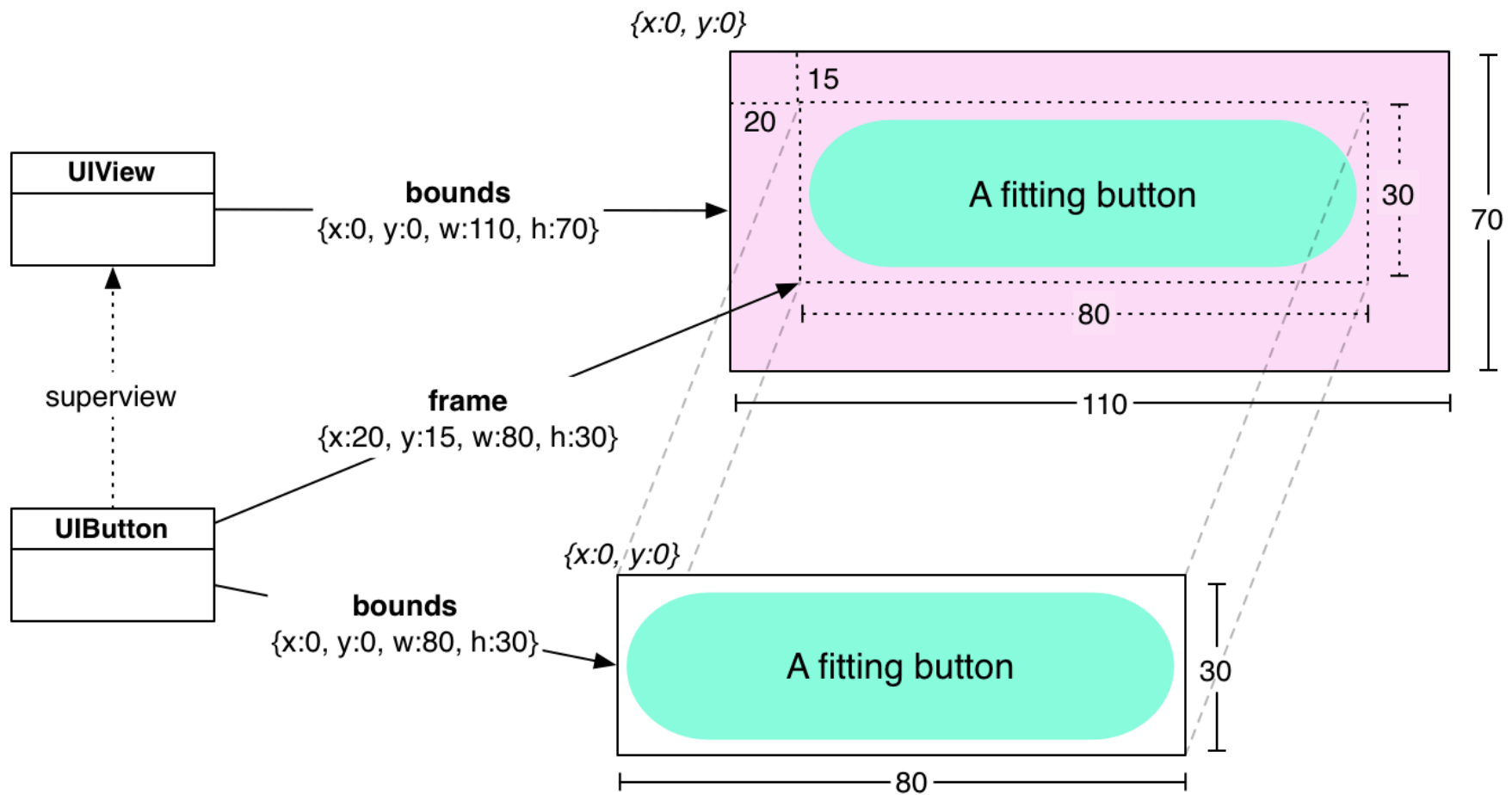
Rastrizace do bounds



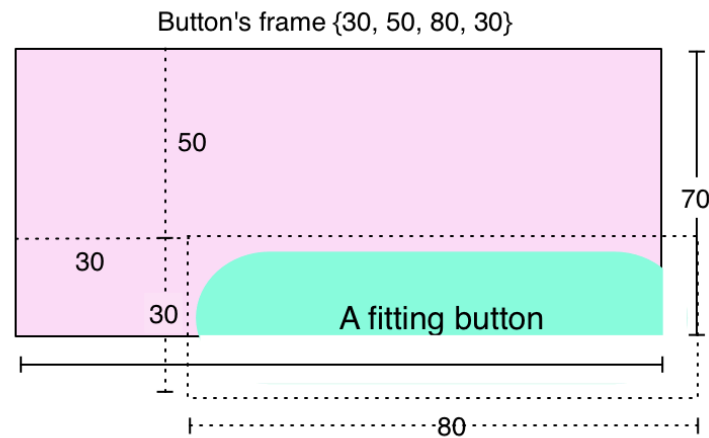
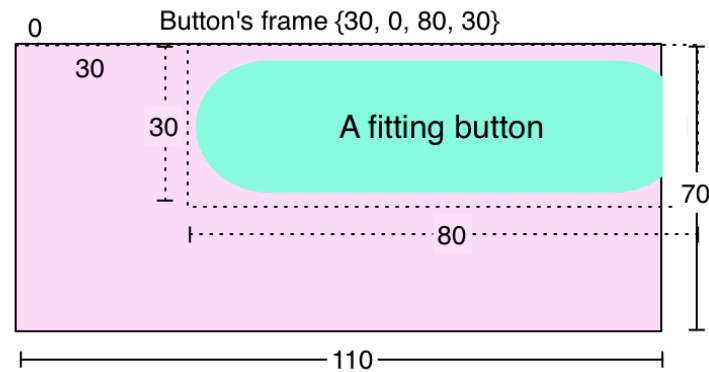
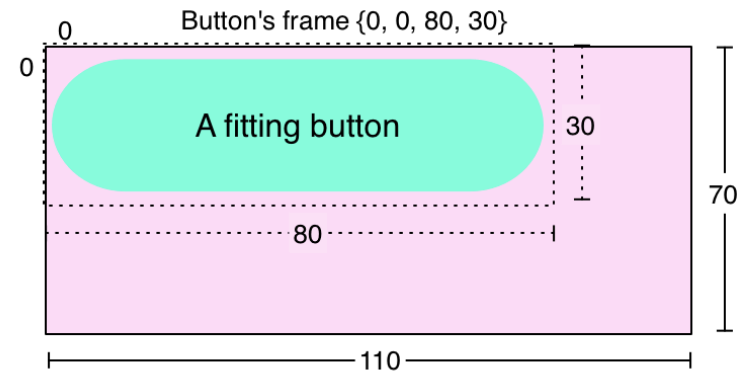
creates this image



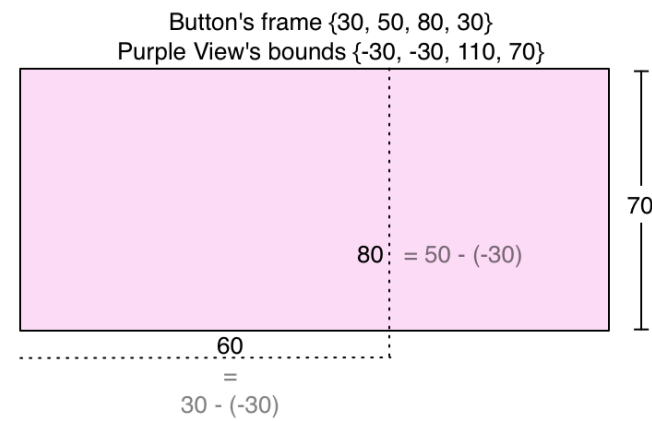
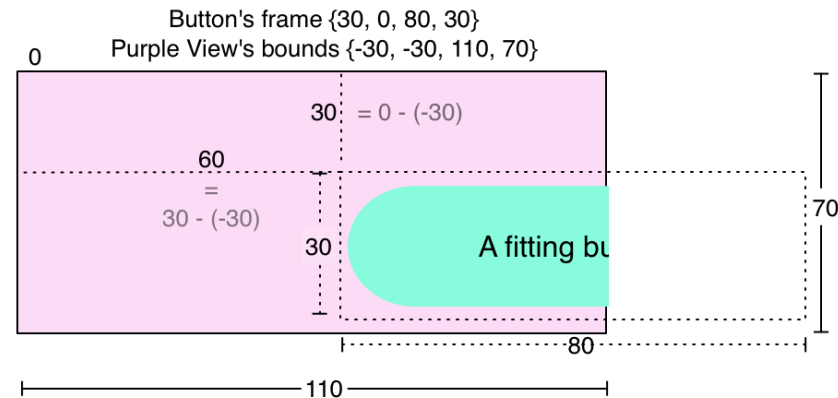
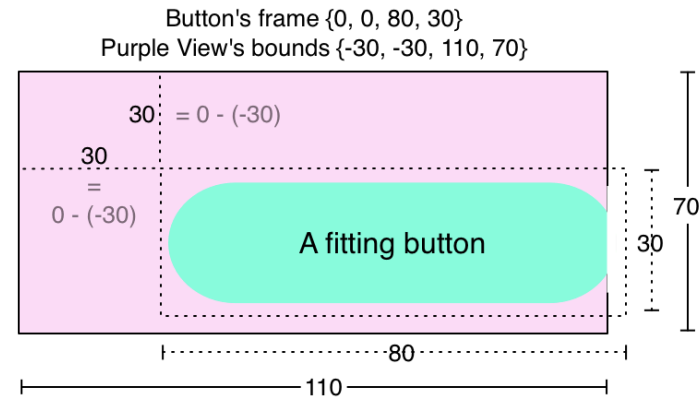
Kompozice view



Kompozice view

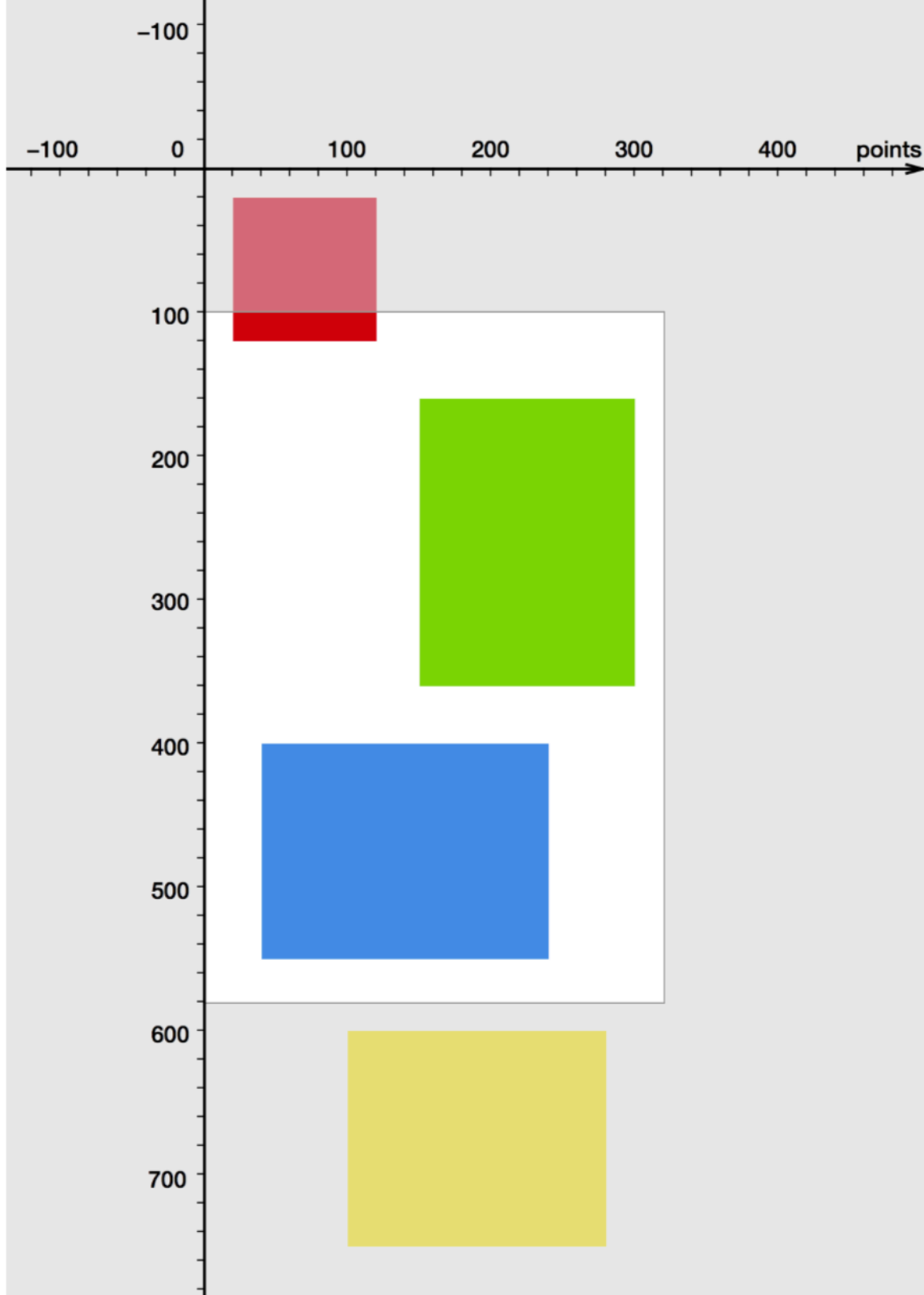


Kompozice, superview bounds



Jak pracuje *UIScrollView*

- Potřebujeme *View* s rozměrem přesahujícím rozměr obrazovky. Typicky *TableView*, *ImageView*, ...
- Takže na událost posuvu pohledu (výřez do *ScrollView*), kdo se hýbe?
 - 1) přesunují se subviews, 2) přesunuje se *ScrollView*.
- Nějaký nápad, jak efektivně změnit výsledek rastrizace *ScrollView* při scrollování?



ScrollView

- Je to opět *view*, tj. má *frame* a *bounds* (zobrazovanou velikost).
- Chceme však virtualizovat jeho velikost, tj. *contentView > bounds.size*;
- *contentOffset* — viditelná část *view* (*bounds*) se nastaví levým horním rohem na *contentOffset*.
 - tj. zprostředkovaně nastavují *bounds.origin*.

Komunikace Controller <-> View

- Controller chce změnit stav svého view.
 - Tok událostí od Model (Controller) do View.
- View detekuje uživatelskou událost (gesto), volá controller.
 - Target-action. Objektu X pošli zprávu Y.
 - Typicky směřováno do Controlleru.
 - View obdrží vstup (text) a chce jej propagovat do modelu.

Stav objektů View

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

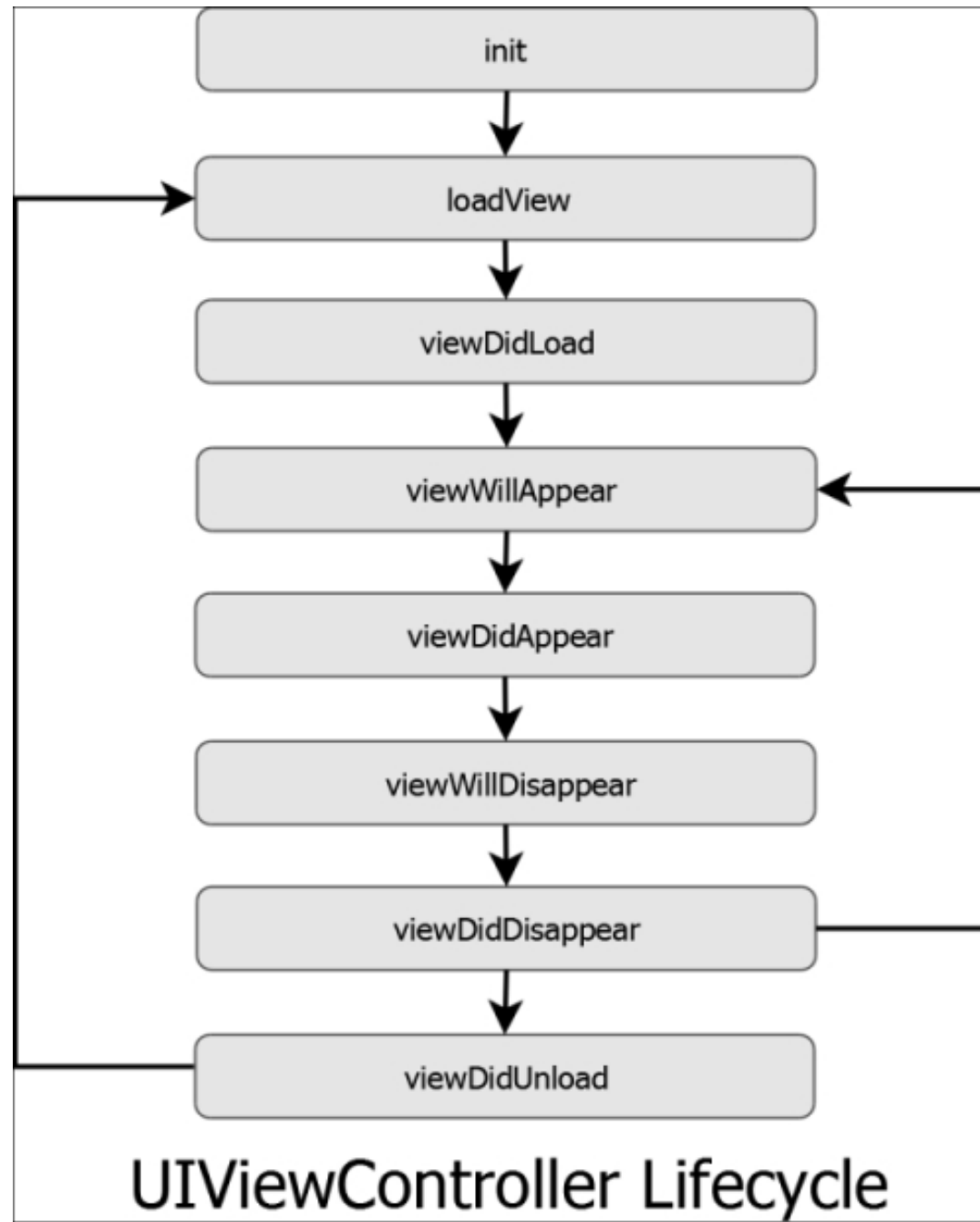
Stav objektů view

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

ViewController (VC)

- Je vlastníkem objektu "view". Ten dále tvoří hierarchii dalších subview.
- V aplikaci se manipuluje s VC.
- UIWindow referencuje svůj "root" VC.
 - Ten může organizovat další VC: navigation, tabbar, ...
- UINavigationController — základní prvek řízení.
- Životní cyklus ViewControlleru.

Životní cyklus ViewController

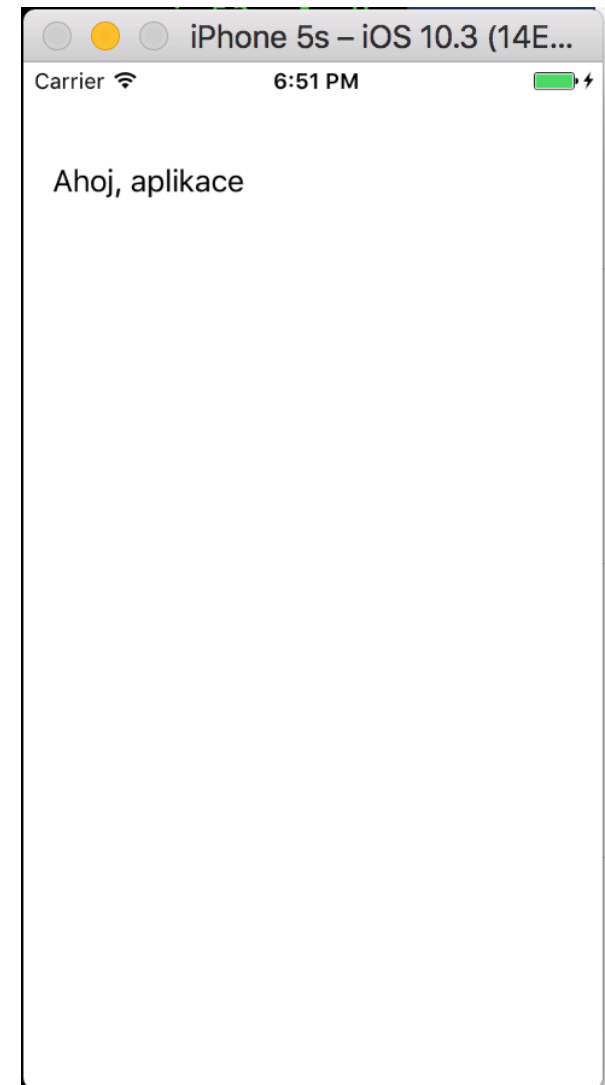


Single-VC aplikace

- *UIWindow.rootViewController* je nějaký jednoduchý VC, např. *UITableViewController*
- Jinak uvažujeme vždy přepínání VC prostřednictvím nějakého řídicího VC (container VC): *Navigation*, *TabBar*, ...

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  
    }  
}
```



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 "`needDisplay`".
 - 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?)
    {
        DispatchQueue.main.async {
            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  
    }  
}
```

M-V-C, didSet, korektně

```
protocol MujModelDelegate { func update(from: MujModel); }

//
class MujModel {
    //
    weak var delegate : MujModelDelegate

    // synchronni, zustavame ve stejnem vlakne
    func updateDelegate() {
        //
        delegate.update(from: self)
    }

    // asynchronni, prechazim do MainThread
    func updateDelegeteAsync() {
        //
        DispatchQueue.main.async {
            delegate.update(from: self)
        }
    }

    var obsah : String = "Ahoj, aplikace" {
        didSet {
            updateDelegate();
        }
    }
}
```

M-V-C, NotificationCenter

```
var obsah : String = "Ahoj, aplikace" {
    didSet {
        //
        NotificationCenter.default.post(name:
            NSNotification.Name(rawValue: "mojeZprava"), object:
nil)
    }
}

//
override func viewDidLoad() {
    super.viewDidLoad()

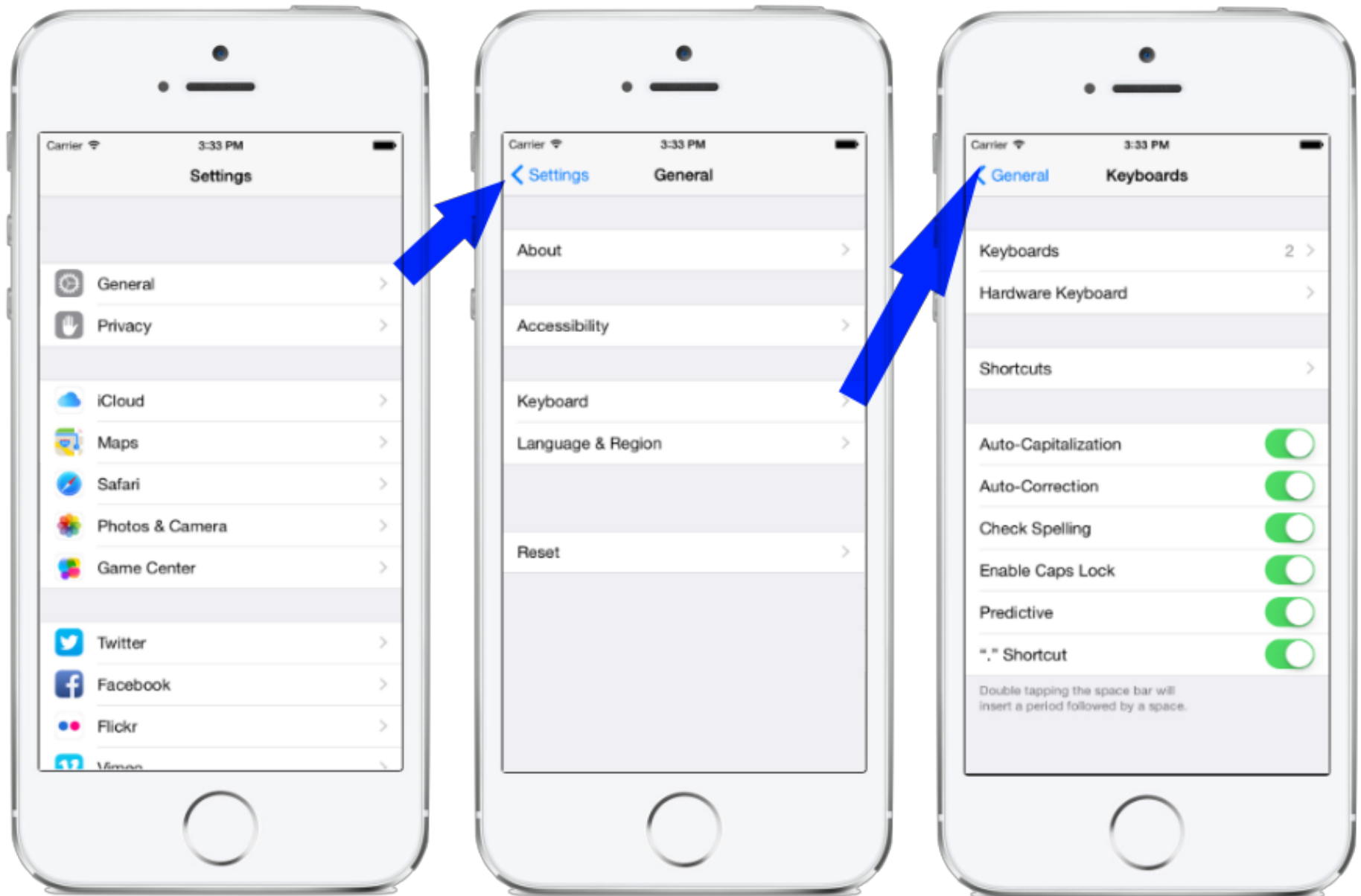
    //
    NotificationCenter.default.addObserver(self, selector:
#selector(update), name: NSNotification.Name(rawValue: "mojeZprava"),
object: nil)
}

func update() {
    //
    lei?.text = mujmodel.obsah
}
```

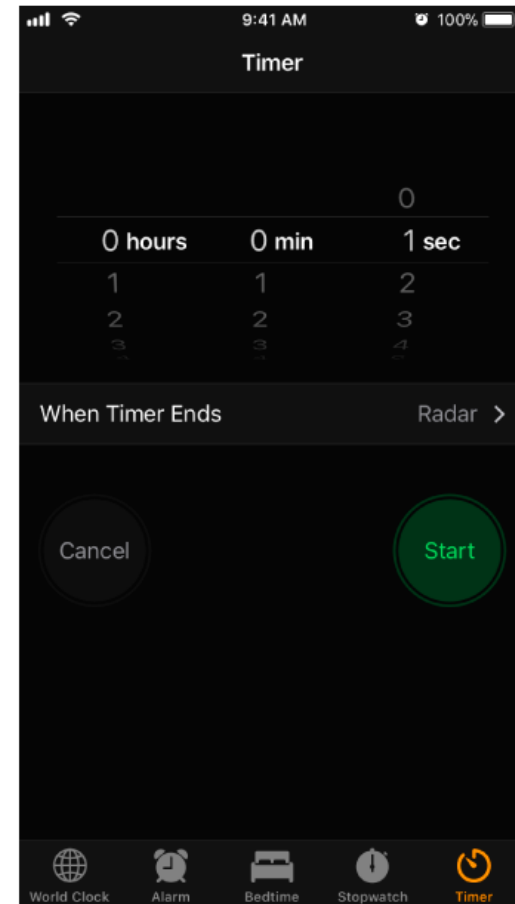
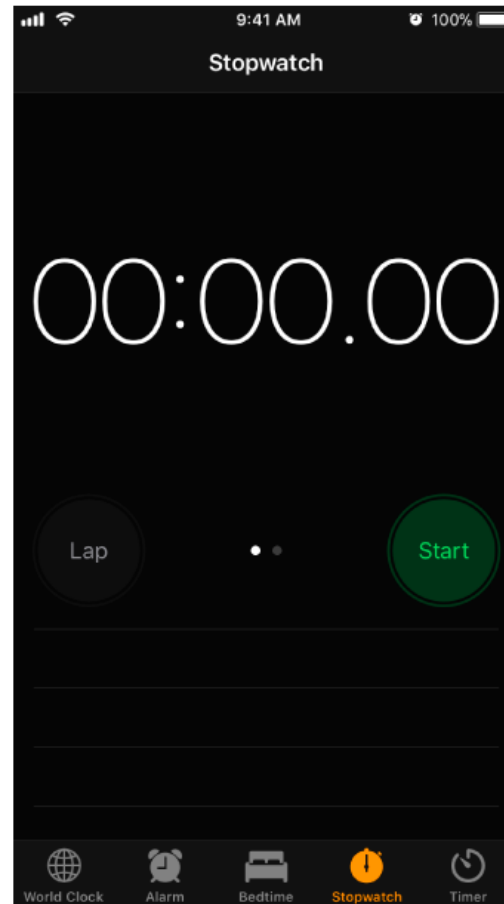
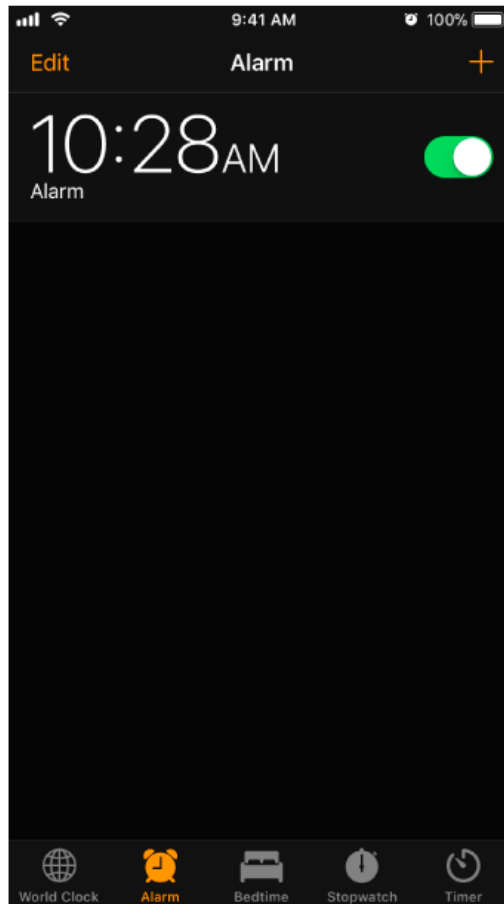
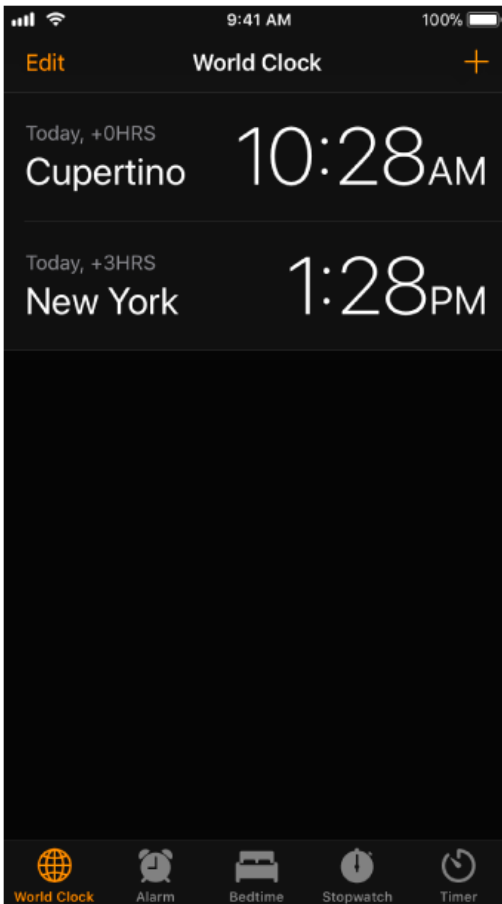
Multi-VC aplikace

- 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

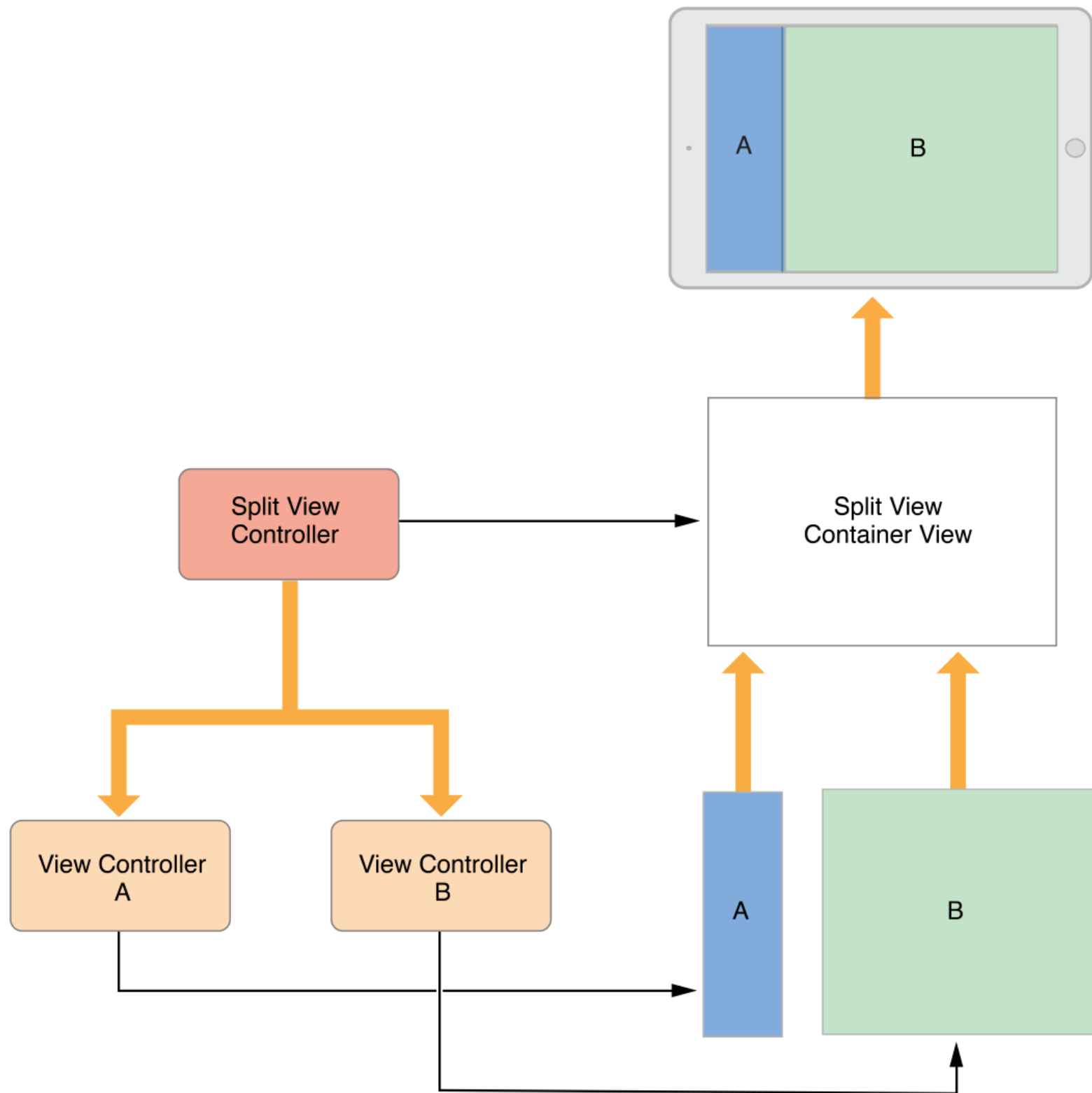


TabBar VC



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`.



Master-Detail styl

- Je složením dvou VC:
 - Master — typicky přehledový VC, tabulka se záznamy.
 - Detail — typicky záběr na detail záznamu z Master.
- Je směřován na tablety, ovšem lze jej konfigurovat i na iPhone.
 - Není SplitView, ale N-VC s obsahem Master, pak push na Detail.

M-D v režimu Landscape



M-D v režimu Portrait





VC referencuje container VCs

- Vazba na UINavigationController, TabBarVC, SplitVC je v aplikáciach tak typická, že UINavigationController automaticky v hierarchii VC najde/referencuje:
 - UINavigationController,
 - Tab Bar VC,
 - Split VC.

Přechod na druhý VC

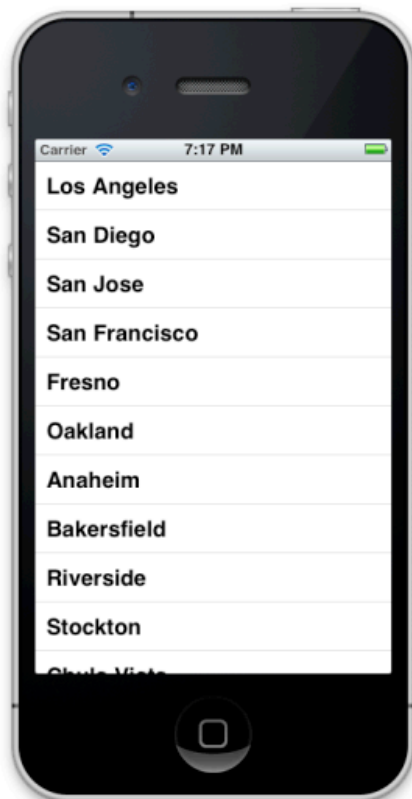
- Deaktivace původního vc : UIViewController:
 - `vc.willMove(toParentViewController: nil)`
 - `vc.view.removeFromSuperview()`
 - `vc.removeFromParentViewController()` — preposílání zpráv.
 - `(vc.viewWillAppear, vc.viewDidAppear)`.

Přechod na druhý VC

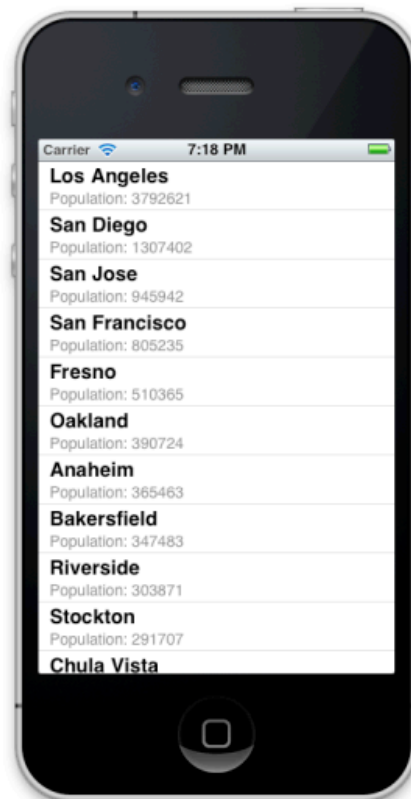
- Aktivace nového VC, `nvc`:
 - `addChildViewController(nvc)` — preposílání zpráv.
 - `view.addSubview(nvc.view)`
 - `nvc.view.frame = view.bounds`
 - `nvc.view.autoresizingMask = [.flexibleWidth, .flexibleHeight]`
 - `nvc.didMove(toParentViewController: self)`
- Nav-VC a TabBarVC takto přepínají VC.

Table View Controller

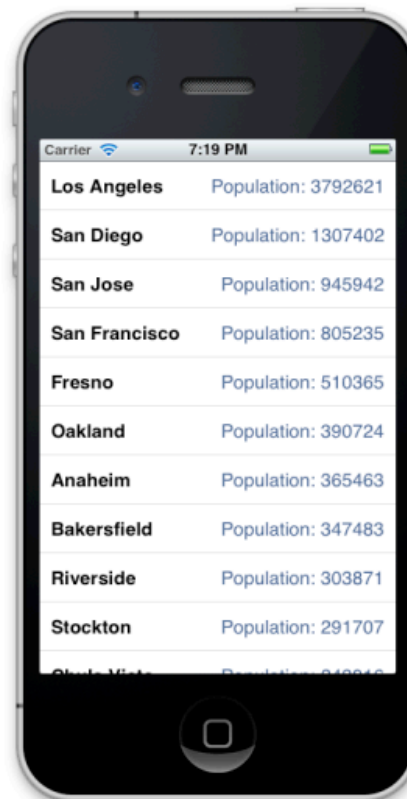
- Nejdůležitější prvek UI iOS.
- Seznam buněk (TableViewCell).



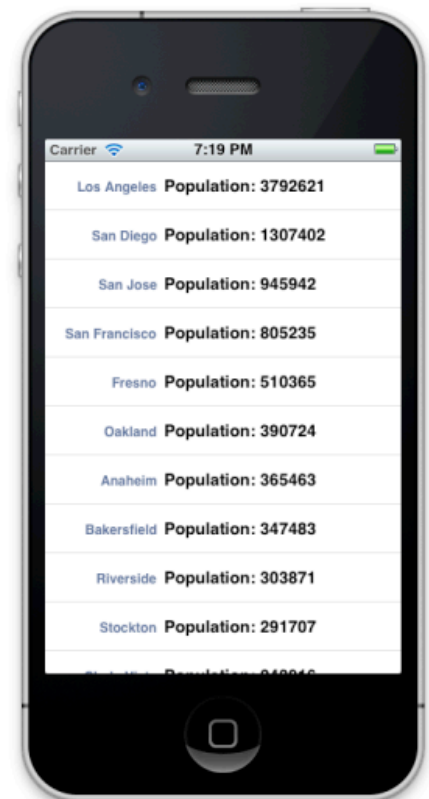
UITableViewCellStyleDefault



UITableViewCellStyleSubtitle

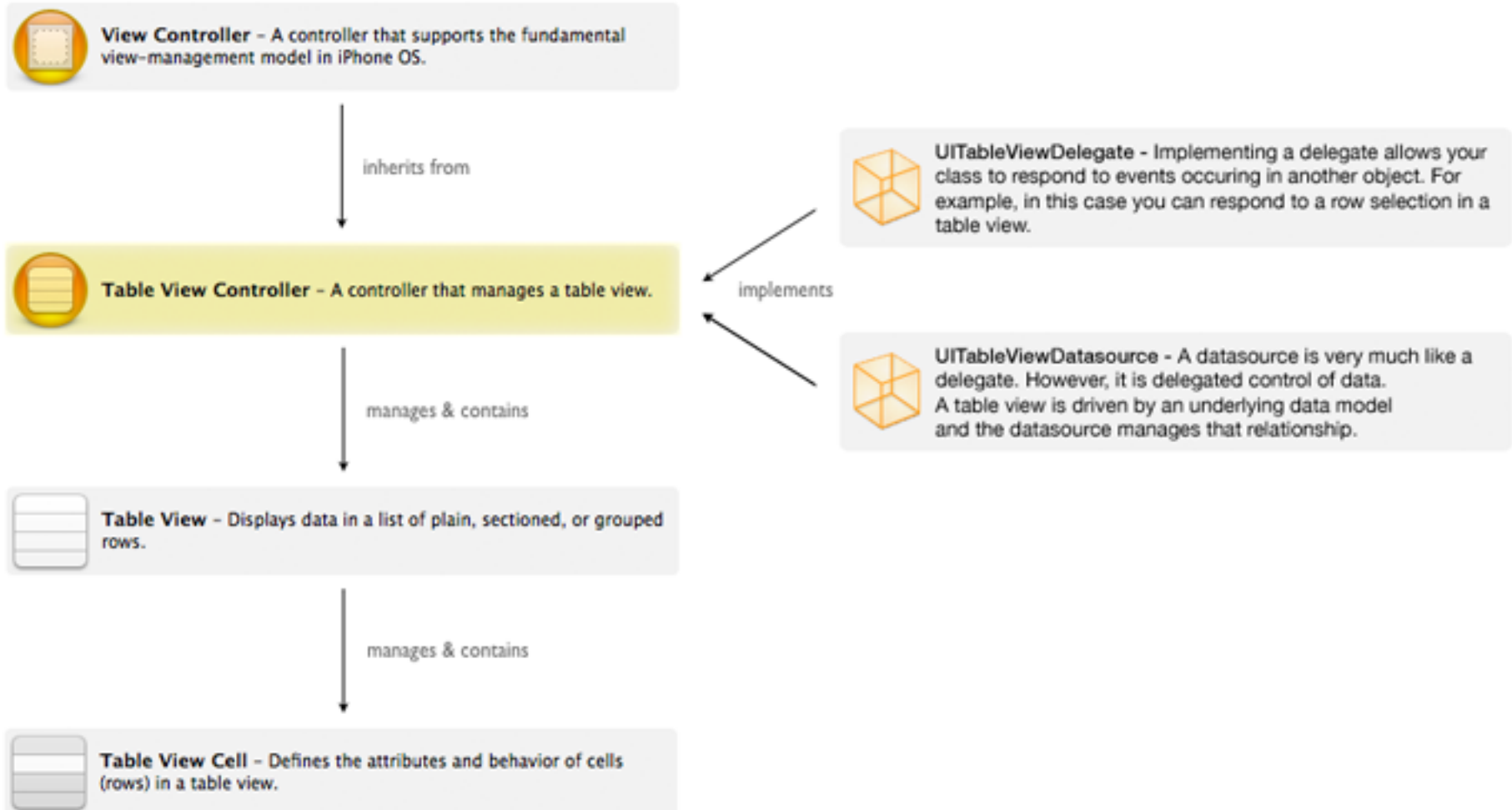


UITableViewCellStyleValue1



UITableViewCellStyleValue2

UITableViewController

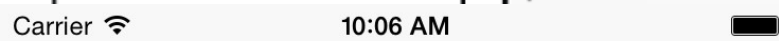
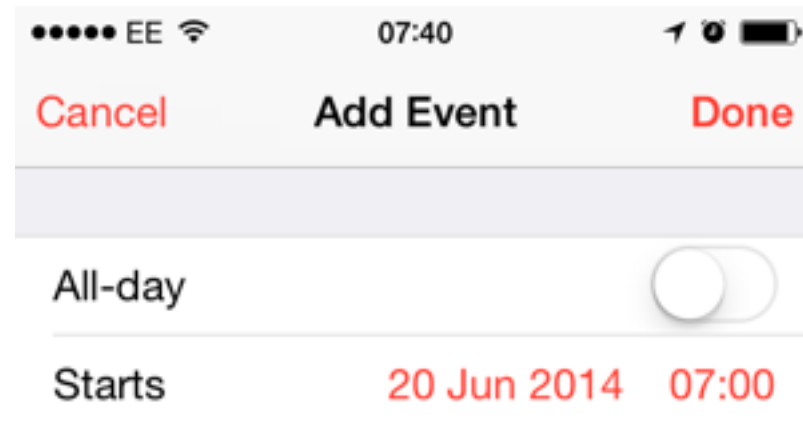
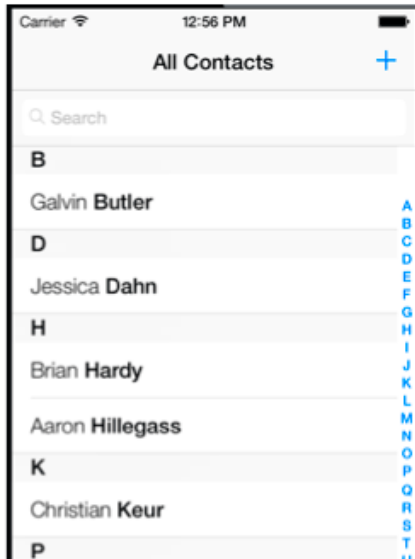
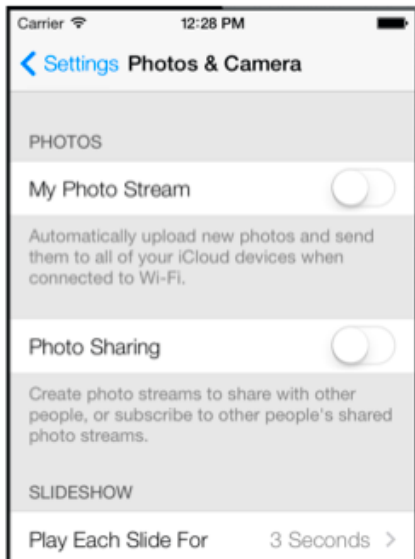


UITableViewController

- 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.



10:06 AM



7:37 AM



7:36 AM



Groups

All Contacts



Search

#

John Appleseed

Kate Bell

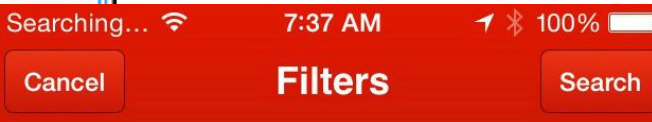
Anna Haro

Daniel Higgins Jr.

David Taylor

Hank M. Zakroff

A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
T



Price

\$

\$\$

\$\$\$

\$\$\$\$

Most Popular

Open Now

7:36 AM



Distance

Auto



0.3 miles



1 mile



5 miles



20 miles



Wall Street Journal retweeted



Gabby Stern @gabbystern

1h

Why Brazilians & expats fill their suitcases when in the US: Tips for coping w/ Brazil's crazy prices on.wsj.com/1J36C6o via @WSJexpat



27

16



Cara Delevingne @Caradelevin... 16m

I miss you Gaga aka Angela Delevingne 1912-2014 ❤️ [instagram.com/p/yFJ7HcjKCz/](https://www.instagram.com/p/yFJ7HcjKCz/)



191

486



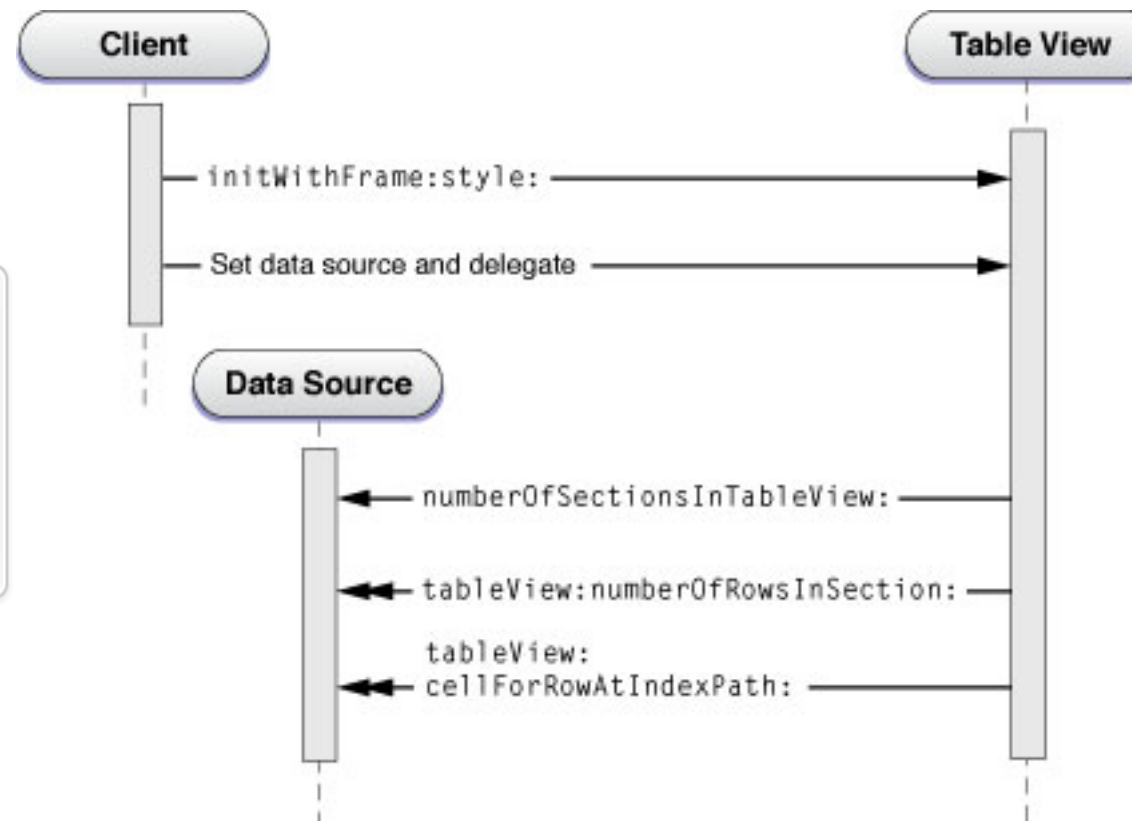
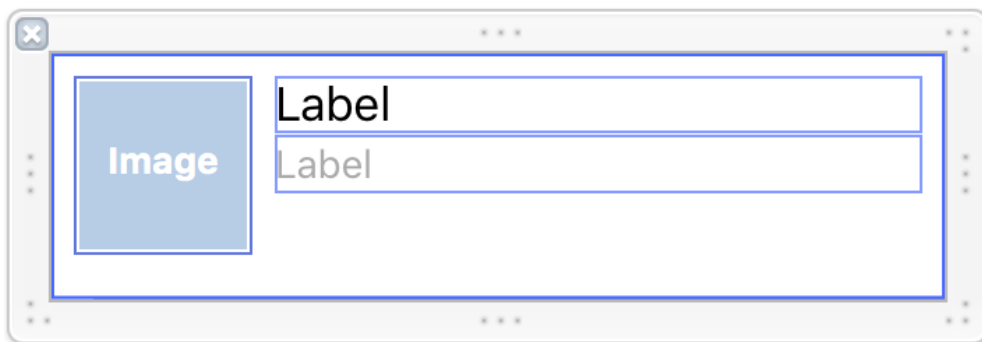
KATY PERRY @katyperry 17m

Hey @JJWatt what do you think of our @ESPMag cover? TOUCHDOWN?!



Řízení — dataSource

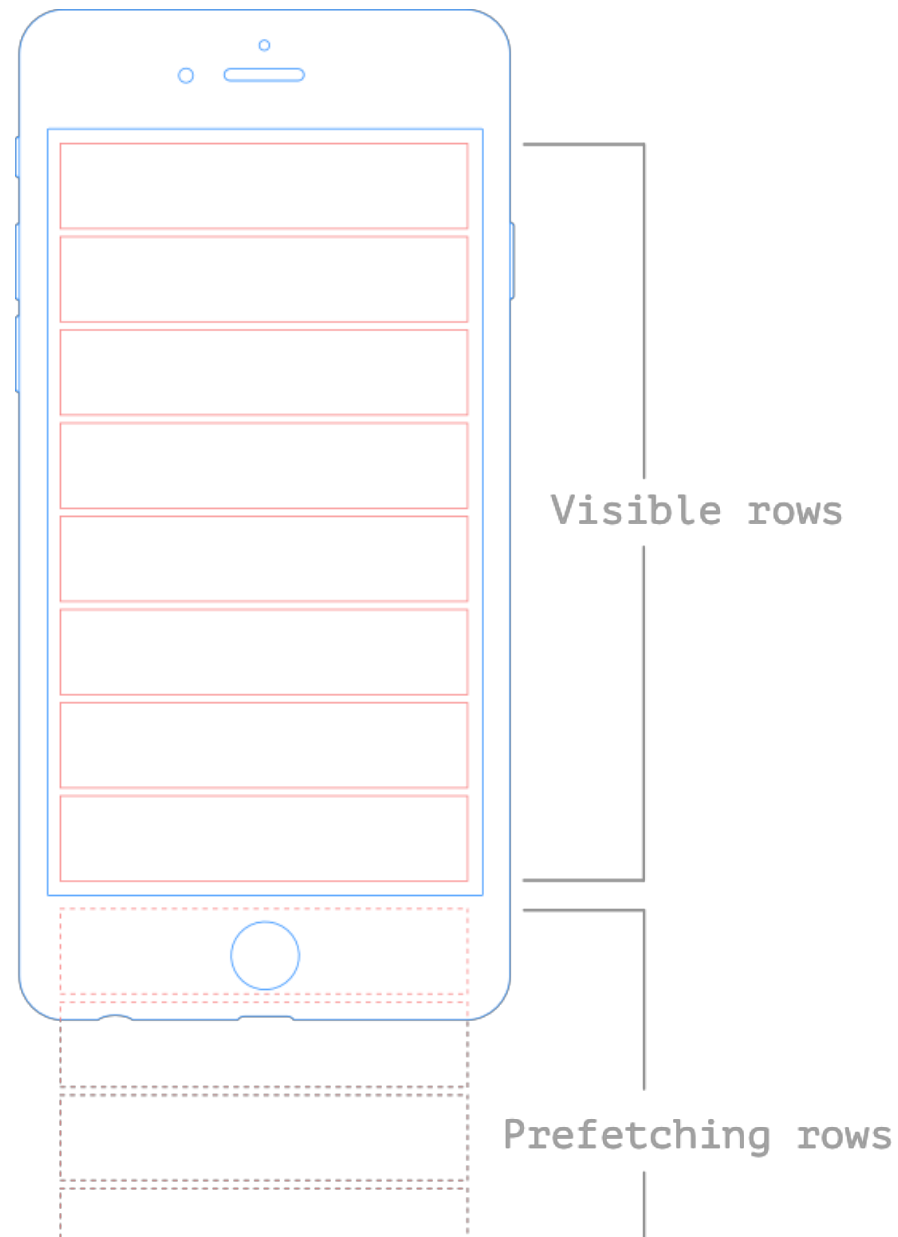
- TVC zprostředkovává obsah (Model) do View.
- Model zde vystupuje jako "dataSource" API:
 - počet sekcí, počet řádků v sekci.
 - sestavení TableViewCell pro zadanou souřadnici (IndexPath).



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  
    }  
}
```

Demo, oddělený dataSource

```
class MyArrayModel: NSObject, UITableViewDataSource {
    let seznam : [String]
    init(withStrings : [String]) {
        self.seznam = withStrings;
        super.init();
    }
    func tableView(_ tableView: UITableView,
                   numberOfRowsInSection section: Int) -> Int
    {
        return seznam.count
    }
    func tableView(_ tableView: UITableView,
                   cellForRowAt indexPath: IndexPath) -> UITableViewCell
    {
        let cell = tableView.dequeueReusableCell(withIdentifier: "cell",
for: indexPath)

        cell.textLabel?.text = seznam[indexPath.row]

        return cell
    }
}
```

Demo, oddělený dataSource

```
class ModelTab: UITableViewController {  
    override func viewDidLoad() {  
        super.viewDidLoad();  
        //  
        self.tableView.dataSource = MyArrayModel(withStrings:  
["1", "2", "3"]);  
    }  
}
```

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;
}

```

TableView, Delegate

- Dostává zprávy o událostech nad tabulkou.
 - Označení buňky — will / did. Zrušení značení — will / did.
 - Typicky se provede přesun do dalšího VC (detail obsahu buňky).
- Geometrie buněk, dodatečné texty, ...
- Rozsáhlý protokol — UITableViewController ho implementuje.

Závěr

- Příště podrobněji o Views a ViewControllers.
- Seminář (Filip Klembara):
 - Swift na Linuxu.
 - macOS ve virtualizaci.
 - Úvod do XCode.
 - Malé demo aplikace.