ViewControllers

IZA, Martin Hrubý, FIT VUT, 2018

Co je ViewController (VC)

- Je to prvek konceptu M-V-C.
- Není zobrazovaným prvkem, ale referencuje V.
- Programování aplikací = programování VC.
  - … a modelů.
  - Neuvažujeme tvorbu vlastních *elementárních* View.
  - Předpokládáme tvorbu vlastních *složených* View, typicky děděním.

Smysl VC

- Aktualizuje obsah Views podle změn v Modelu.
- Reaguje na události pocházející z Views.
- Aktualizuje geometrii Views.
  - Kdy se mění geometrie View?
- Spolupracuje s dalšími VC aplikace.
- Je UIResponder — je v řetězci zpracování události.

Instanciace VC

- Programem v kódu. Pro drsňáky :)
- Storyboard — XIB (NIB), Storyboard.
  - Automatizace správy viewControllerů.

```
//
let story = UIStoryboard(name: "Main", bundle: nil)
let controller = story.instantiateViewController(withIdentifier:
"mujModalek")
```

```
// instanciace MUJV : UIViewController
// XIB: mujv.xib
let mujv = MUJV(nibName: "mujv", bundle: nil)

//
self.navigationController?.pushViewController(mujv, animated: true)
```

Zahájení činnosti VC

- Někdo ho ”musí nějak spustit”.
- UIWindow — rootViewController.
- *Spustí ho jiný VC* (Navigation, TabBar).
  - show / present VC.
  - modální (účelový) — přebírá kontrolu.
  - navigation / tabbar — VC je zakomponován do nav / tab VC.

Modální prezentace

- Spuštení: programově nebo Storyboard.
  - Lze pomocí ”segue” ([segvej]), plynule přejít.
- Ukončení: programově.
  - self.dismiss(animated: completion:)
  - Předat zprávu původnímu VC, např. data.

Prezentace modálně

```
class MujModal: UIViewController {
    //
    @IBAction func butt() {
        ///
        self.dismiss(animated: true, completion: nil)
    }
}

class ViewController: UIViewController {

    @IBAction func buttRUN() {
        //
        let story = UIStoryboard(name: "Main", bundle: nil)
        let controller = story.instantiateViewController(withIdentifier:
"mujModalek")
        self.present(controller, animated: true, completion: nil)
    }
```

Vztah VC na V a další VC

- VC referencuje svůj View, dále hierarchie Views.
  - Typicky z nich chce některé prvky přímo adresovat (Outlets).
- VC je referencován nadřazeným VC.
  - parentViewController — referencování, předávání některých zpráv.

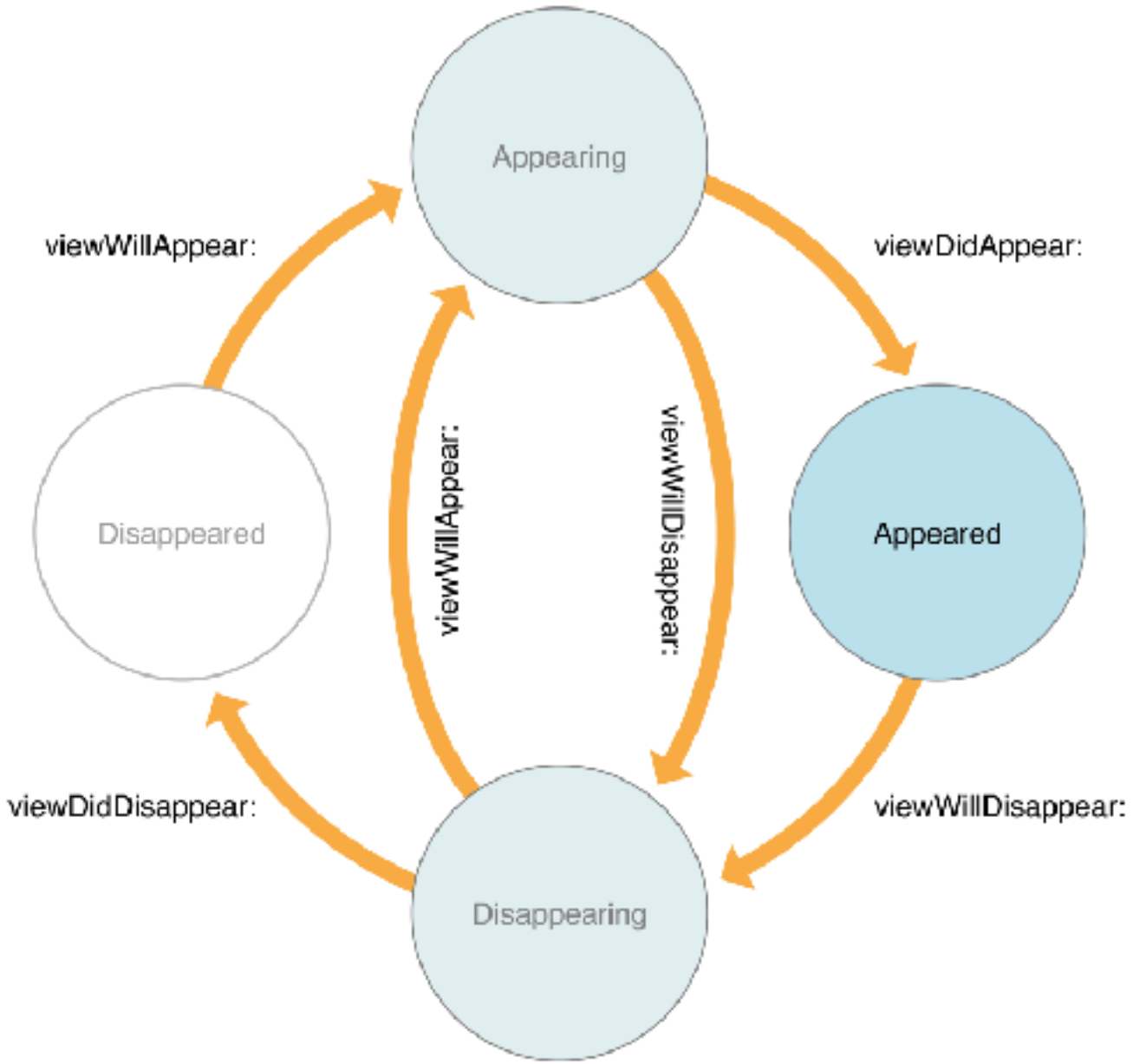
Prezentace VC, Navigation

- Deaktivace původního vc : UIViewController:
  - vc.willMove(toParentViewController: nil)
  - vc.view.removeFromSuperview()
  - vc.removeFromParentViewController()
- Pozn.: výměnu views (starý / nový) lze provést animovaně.

Prezentace VC, Navigation

- Aktivace nového VC, nvc:
  - addChildViewController(nvc)
  - view.addSubView(nvc.view)
  - nvc.view.frame = view.bounds
  - nvc.view.autoresizingMask = [.flexibleWidth, .flexibleHeight]
  - nvc.didMove(toParentViewController: self)

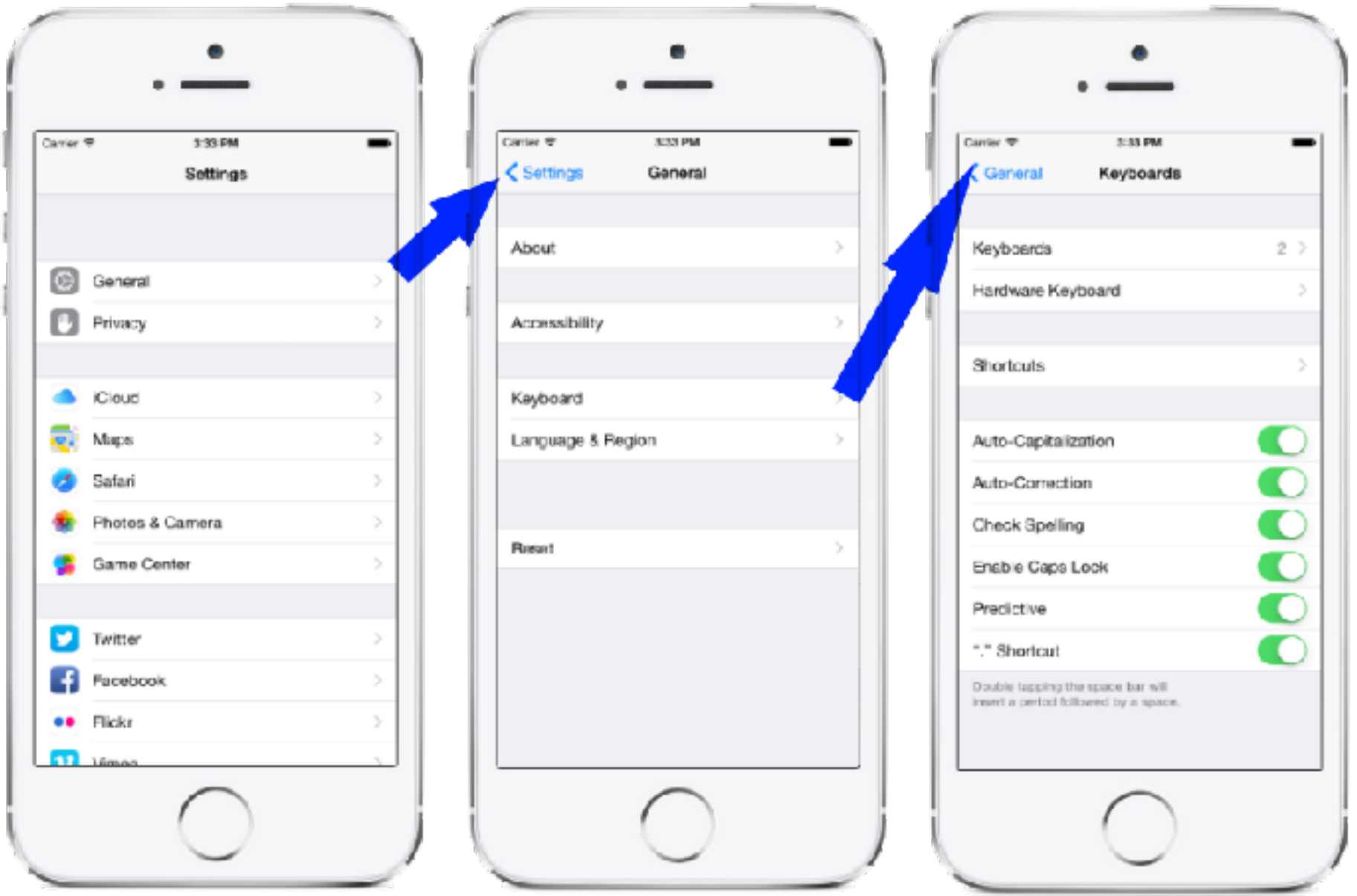
Životní cyklus zobrazování VC



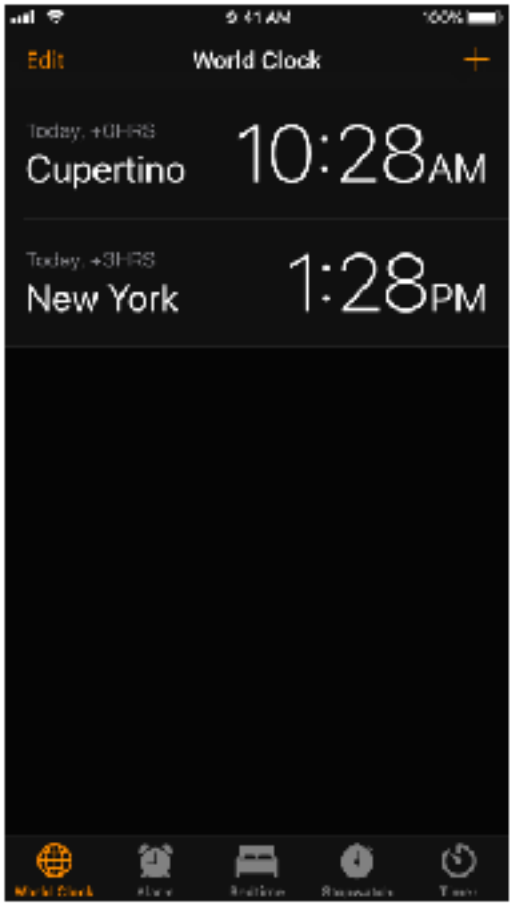
Přehled základních VC

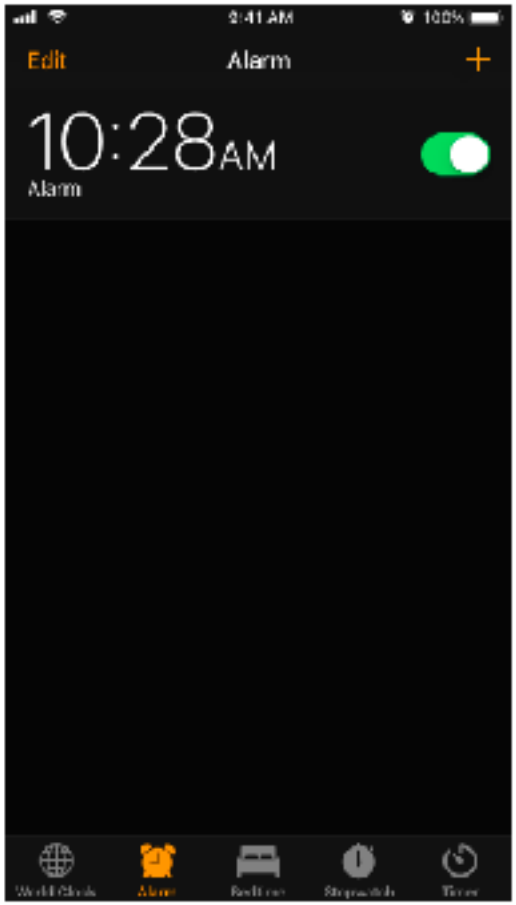
- *NavigationVC* — kontejner VC. Zavádí horní lištu (tlačítka). Obaluje jiný VC.
- *TabBarVC* — pevně stanovený rozsah VC. Spodní tlačítková lišta.
- *TableViewController* — tabulka/seznam.
- *SplitViewVC* — řídí gemetrii více VC.
  - Zejména v aplikacích pro macOS.
- VC má properties na ref příp. kontextu VC.

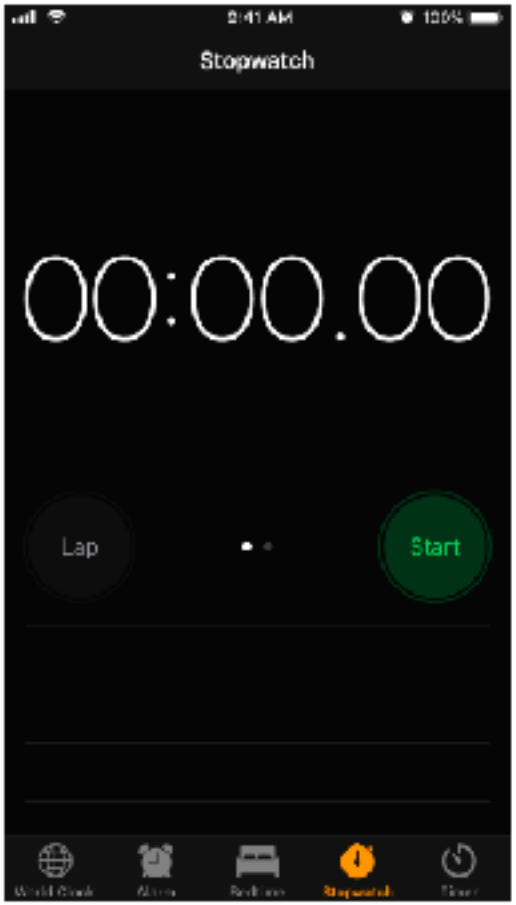
Navigation VC

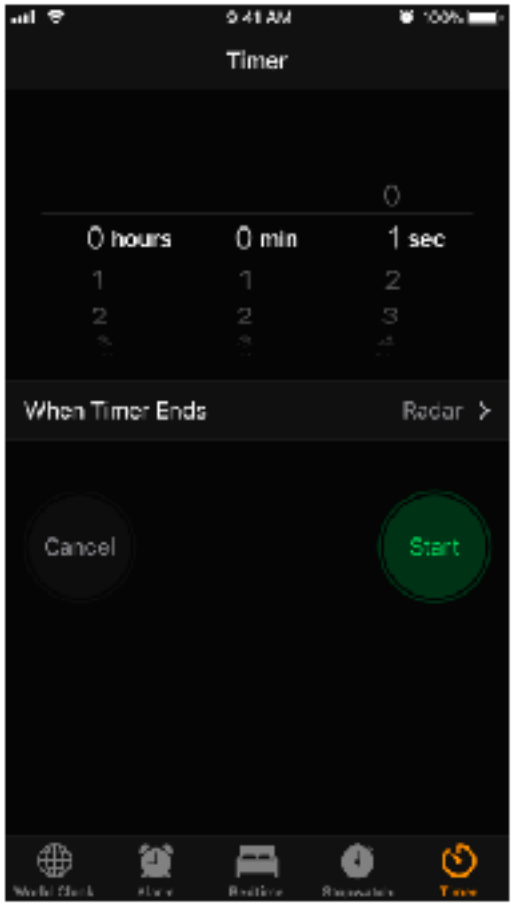


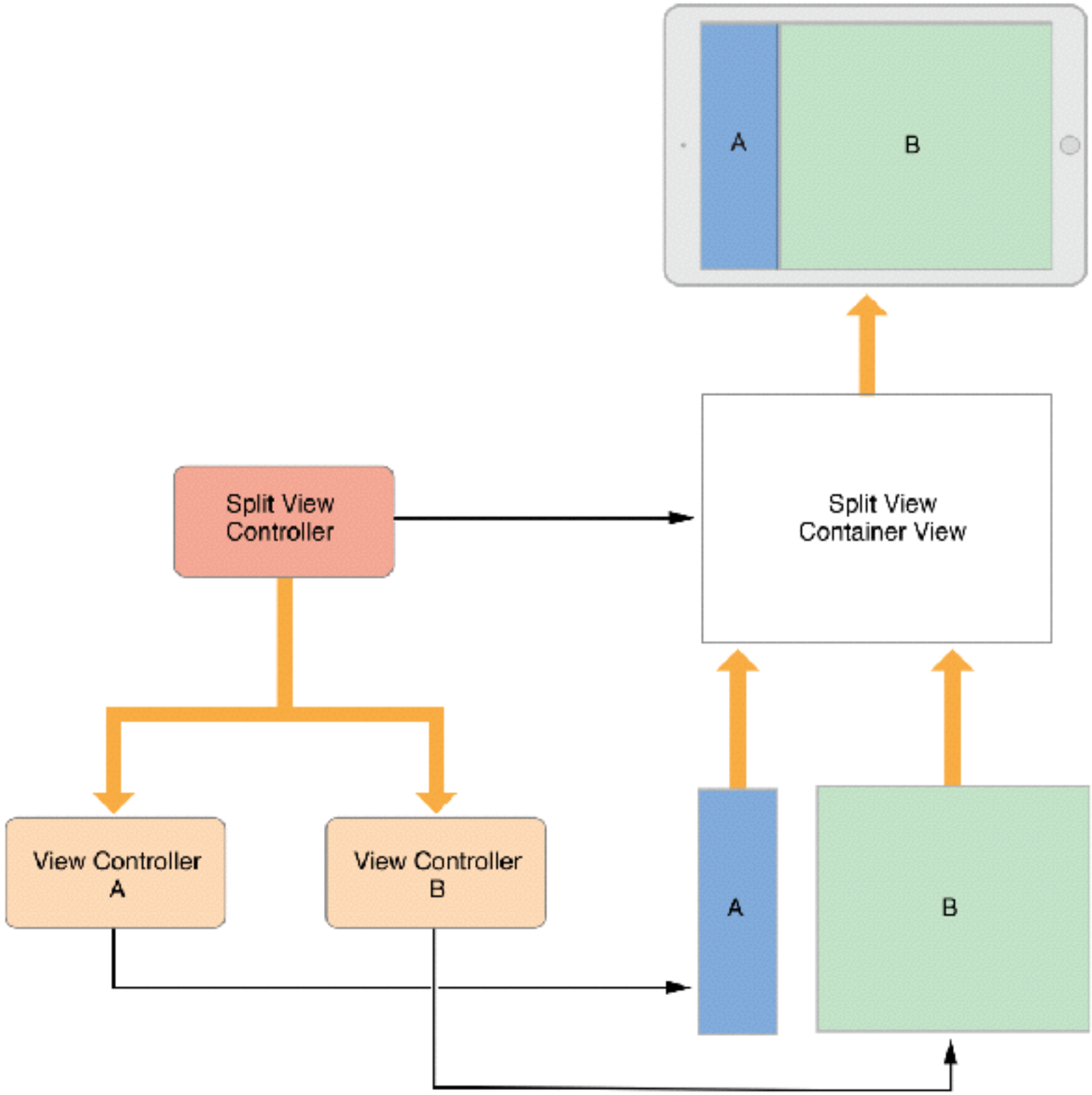
TabBar VC


Edit
World Clock
Today, +0HRS
Cupertino
10:28AM
Today, +3HRS
New York
1:28PM


Edit
Alarm
10:28AM
Alarm


Stopwatch
00:00.00
Lap
Start


Timer
0 hours
0 min
1 sec
When Timer Ends
Radar
Cancel
Start


A
B
Split View
Controller
Split View
Container View
View Controller
A
View Controller
B
A
B

Spuštění VC v Navigation

- Segue nebo programově.
- self.navigationController,

```
@IBAction func buttRUN() {
        //
        let story = UIStoryboard(name: "Main", bundle: nil)
        let controller = story.instantiateViewController(withIdentifier:
"mujModalek")

        //
        self.navigationController?.pushViewController(controller,
animated: true)
    }
```

Řízení: vztah View-Controller

- View je pasivní prvek (UILabel). Je to @IBOutlet.
- Přes referenci lze přistupovat na prvek.

```
class ViewController: UIViewController {
    // reference na prvek navržený ve StoryBoard
    @IBOutlet var textik : UILabel!

    // akce na stisk klavesy
    @IBAction func buttRUN() {
        //
        self.textik.text = "Stiskl jsi button"
    }
    // zprava, ze alokace/inicializace VC je dokoncena
    override func viewDidLoad() {
        //
        super.viewDidLoad()

        // zde smim ocekavat hodnotu v self.textik
        self.textik.text = "Ahoj"
    }
```

Forma referencování Outlets

- Vlastníkem Outletu je View, ne VC.
- @IBOutlet var textik : UILabel!
  - strong reference s garantovanou/předpokládanou hodnotou
- @IBOutlet weak var textik : UILabel!
  - weak — připouští NULL, specifikace "!" je pochybná
- @IBOutlet weak var textik : UILabel?
  - dává smysl

Řízení: vztah View-Controller

- View je (aktivní) prvek s vnitřním stavem (UISwitch).
- Reaguje na události od uživatele — dotyk.
- Posílá zprávy o průběhu:
  - předpokládejme typicky svému VC.

```
// akce na stisk klavesy
    @IBAction func buttRUN() {
        //
        self.textik.text = "Stiskl jsi button"

        //
        if self.swi.isOn == true {
            self.swi.setOn(false, animated: true)
        }
    }

    //
    @IBOutlet weak var swi : UISwitch!
    var pocetZmen = 0

    @IBAction func  swiChanged() {
        //
        pocetZmen = pocetZmen + 1

        //
        self.textik.text = "Zmen: \(pocetZmen)"
    }
```

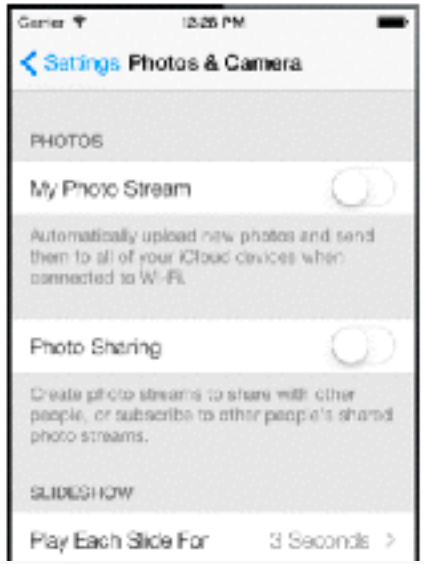
Views s komplexním chováním

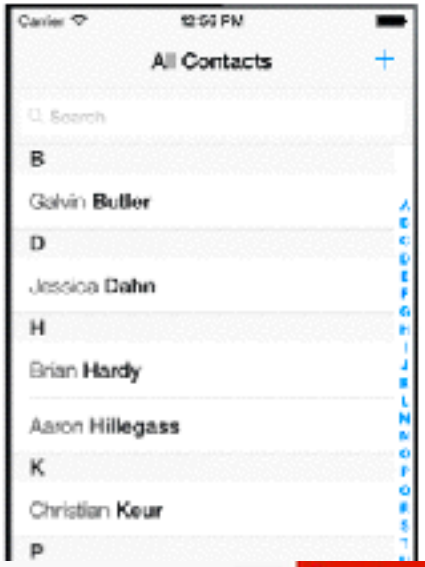
- UILabel, UISwitch — triviální chování.
- Views prezentující rozsáhlý, dynamický, strukturovaný obsah.
  - UITableView.
  - UICollectionView.
- DataSource, Delegate — vede na specializované VC.

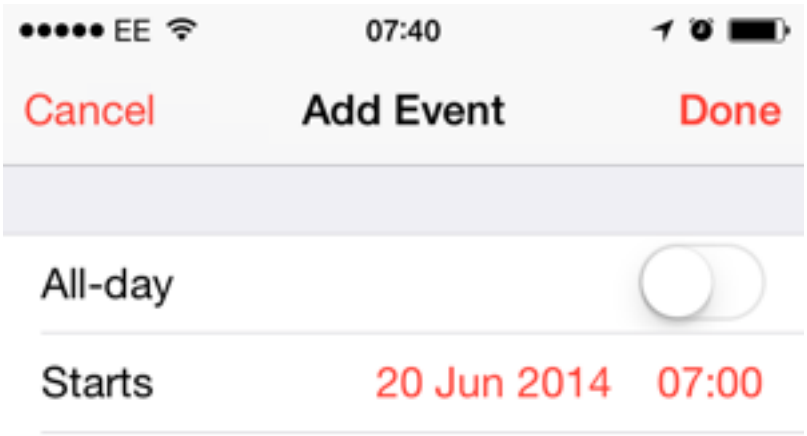
Table View Controller

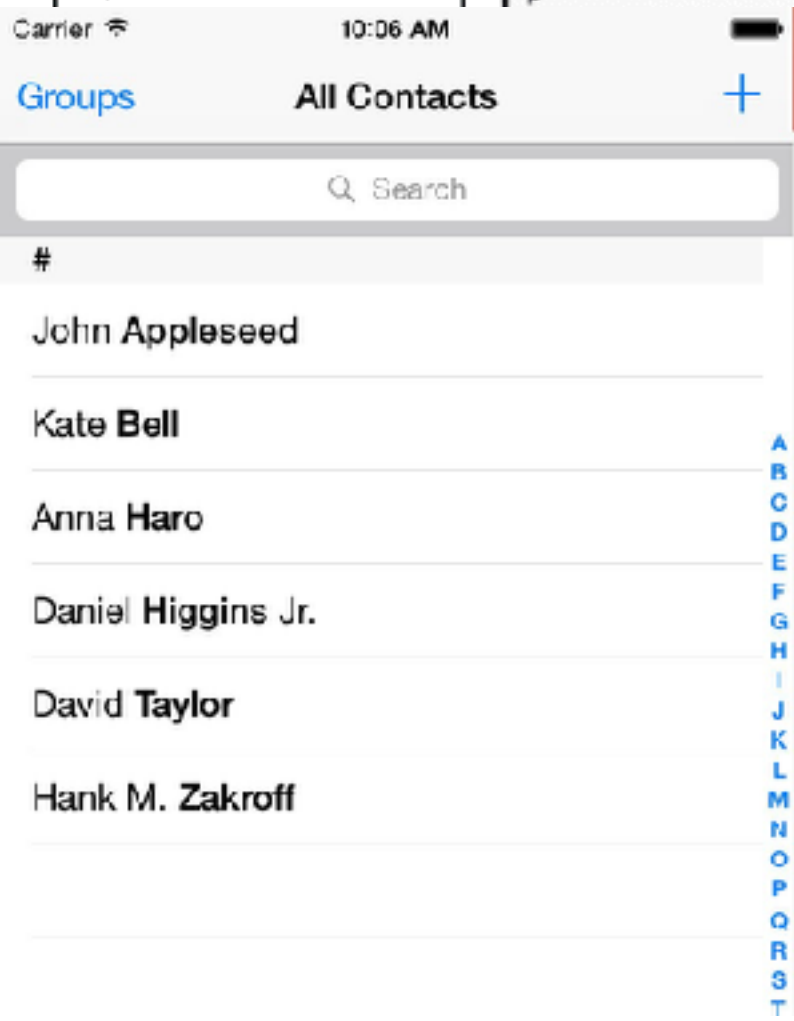
- Nejdůležitejší prvek UI iOS.
- Seznam buněk (TableViewCell).

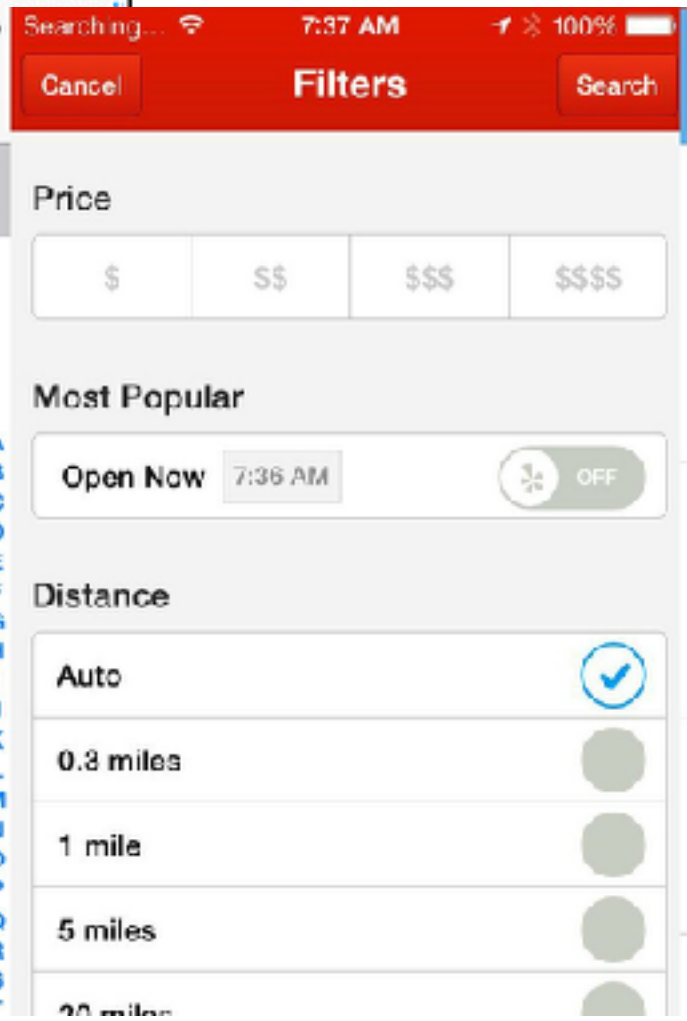
UITableViewCellStyleDefault UITableViewCellStyleSubtitle UITableViewCellStyleValue1 UITableViewCellStyleValue2


Settings Photos & Camera
PHOTOS
My Photo Stream
Photo Sharing
SLIDESHOW
Play Each Slide For
3 Seconds


All Contacts
B
Galvin Butler
D
Jessica Dahn
H
Brian Hardy
Aaron Hillegass
K
Christian Keur
P


EE
07:40
Cancel
Add Event
Done
All-day
Starts
20 Jun 2014
07:00


Carrier
10:06 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


Searching...
7:37 AM
100%
Cancel
Filters
Search
Price
$
$$
$$$
$$$$
Most Popular
Open Now
7:36 AM
OFF
Distance
Auto
0.3 miles
1 mile
5 miles


Searching...
7:36 AM
100%
Home
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/
191
485
KATY PERRY @katyperry 17m
Hey @JJWatt what do you think of our @ESPNMag cover? TOUCHDOWN?!
Timelines
Notifications
Messages
Me

UITableView

- Odvozen od UIScrollView (přesahuje rámec obrazovky).
- Provádí rozmístění (layout) buněk (Cell) ve skupinách.
- Obecný heterogenní dynamický soupis buněk vertikálně řazených.
- Je to kontejner geometricky uspořádaných Views (UITableViewCell).

Co musí TableView zjistit?

- Obsah pro zobrazení:
  - skupiny buněk a buňky (počet, obsah).
  - geometrii — buněk, rozložení do skupin (nadpisy apod), výšku buněk.
- Sestavení modelu obsahu:
  - počet skupin,
  - počet buněk v zadané skupině,
  - konfigurace buňky — TV alokuje buňku a chce ji pouze konfigurovat. Alokace UITableViewCell, pool.

Protokol UITableViewDataSource

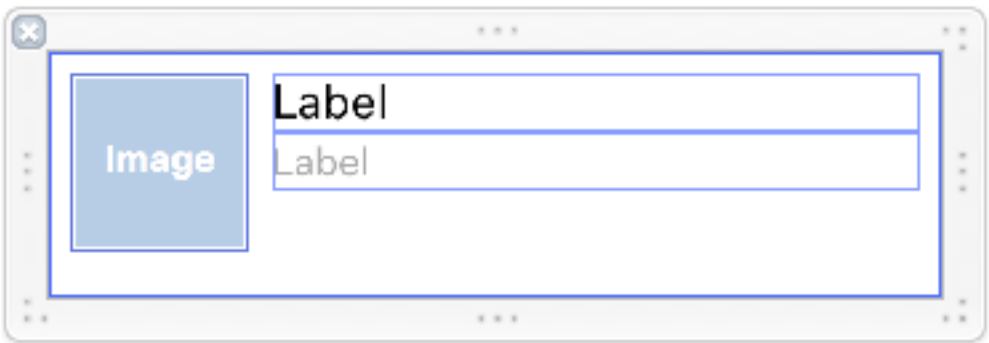
- 3 otázky o obsahu: skupiny, počet, buňka.
- func numberOfSections(in tableView: UITableView) -> Int
- func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int
- func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell

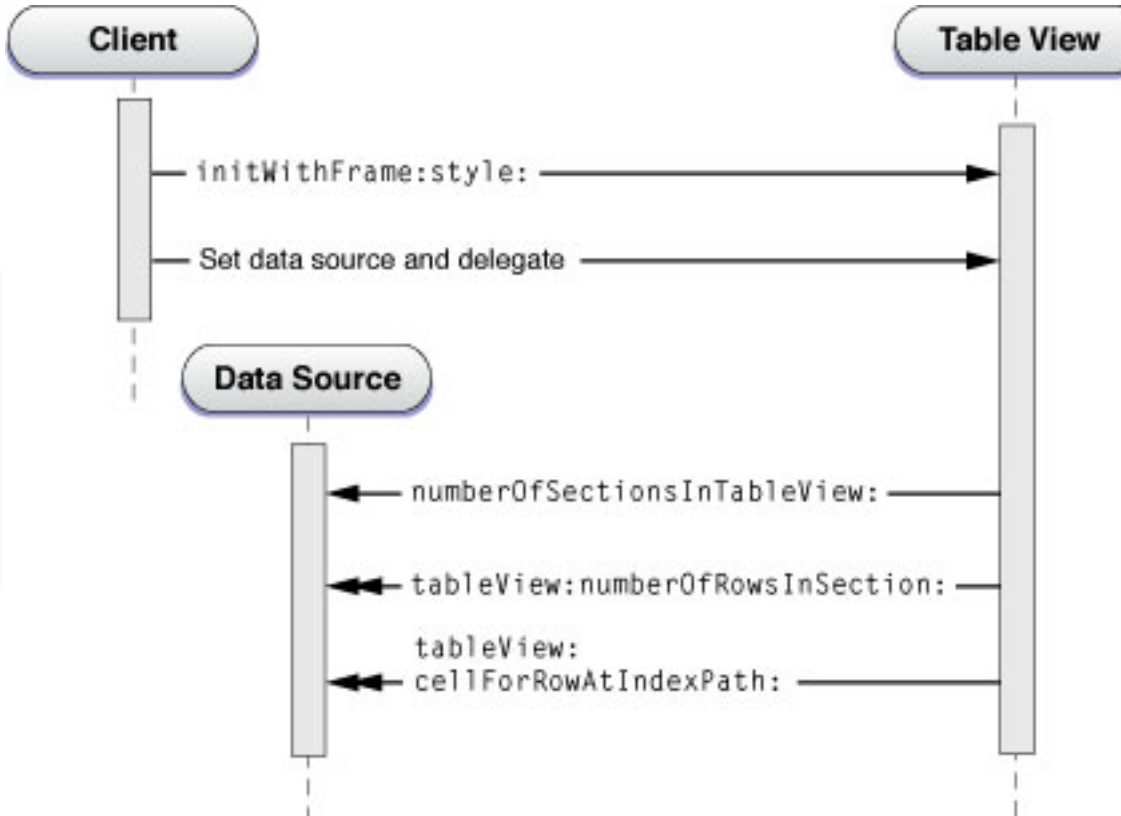
Řízení TV, dynamika

- TV referencuje dva objekty přes protokol:
  - dataSource — dodavatel obsahu,
  - delegate — reakce na události, konfigurace geometrie.
- UITableViewCell — odvozování vlastních.
- Oba mohou být stejný VC, ale lze uvažovat i jinak (parametrický dataSource).

Řízení — konfigurace buňky

- Výstupem je ref na zkonfigurovanou buňku.
- Alokace buňky: pool podle ID-jména buňky.

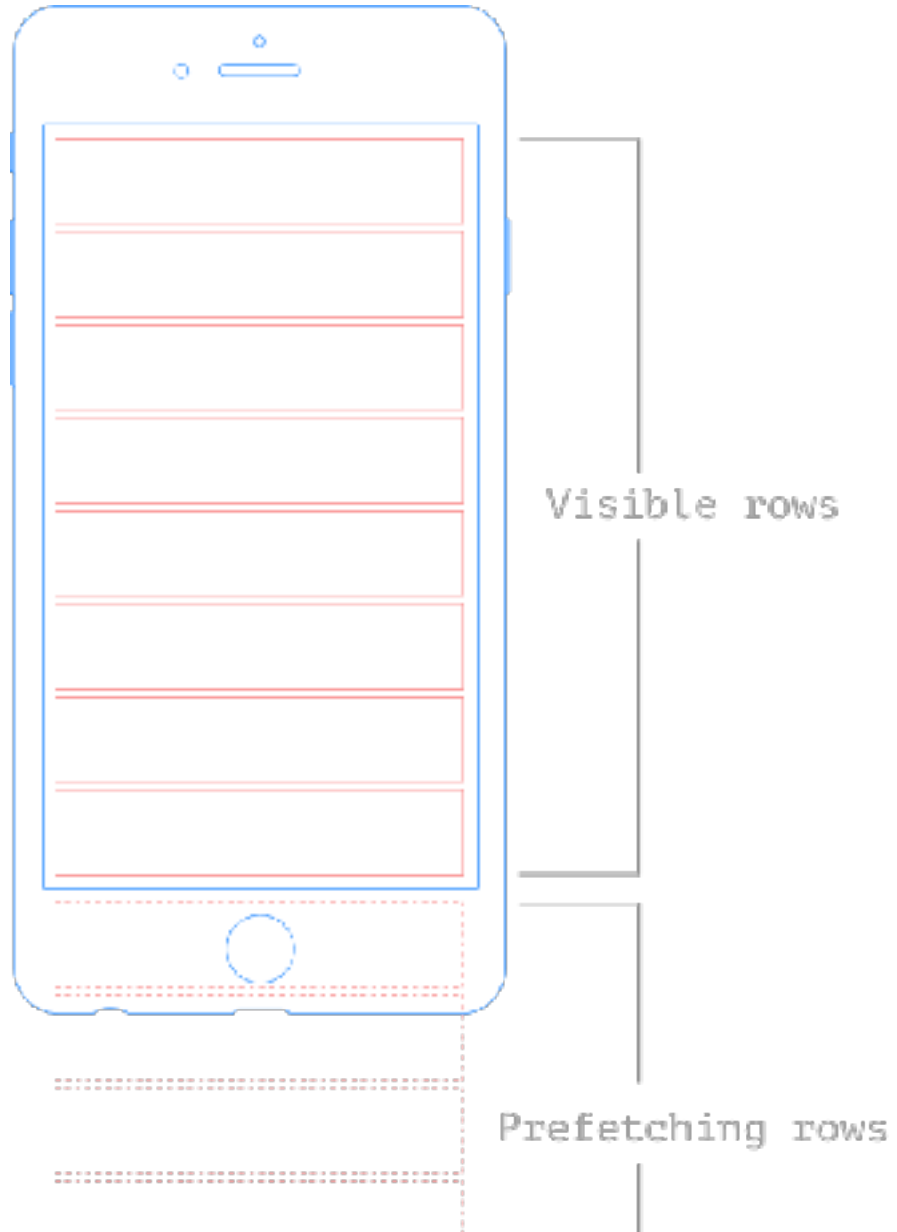




Pool objektů

- Kontejner pro odložené objekty. Alokátor objektů. Znovupoužitelnost objektů.
  - Očekáváme značnou dynamiku v alloc/free.
  - Organizace alokace — alokace se provádí z ”jednoho místa”.
  - reusableCode — ID buňky. Prototyp buňky.
- Časová úspora:
  - v malloc() pochybná,
  - v procesorových cache už může být značná!

Dynamika přístupu na dataSource



UITableViewController

- Jeho view je instance TableView.
- Implementuje dataSource a delegate protokoly tabulky.
- Uživatelský VC dědí z TVC, přepisuje potřebné metody dataSource a delegate protokolů.
- Tzn. obsah tabulky netvoří VC ale TableView.
- tableView.reloadData()

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.

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
    }
}
```

Heterogenní tabulky

- Každá buňka (UITableViewCell) tabulky může být individium.
- VC (popř. Model) musí správně reagovat na NSIndexPath (.row, .section).
  - tableView.dequeueReusableCell(*withIdentifier*: "cell", for: indexPath)
  - tzn. musí zde rozhodnout prototyp buňky pro danou ”section” a ”row”.
  - u heterogenních seznamů může být značně pracné.

Vlastní buňky

- UITableViewCell je View tvořící superview dalším prvkům (UILabel, …).
  - Jsou to jeho @IBOutlet.
- Odvodit vlastní buňky.
- Konfigurace buňky datovým objektem.

Demo, oddělený dataSource

```
class MyArrayModel: NSObject, UITableViewDataSource {
    var 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"]);
    }
}
```

Vztah buňka a model

- Zřejmě vytvářím uživatelskou UITableViewCell pro zobrazení obsahu konkrétní datové struktury.
- Sebe-konfigurace jako instanční metoda buňky.

```
struct DataItem {
    let jmeno : String
}

class BunkaJedna: UITableViewCell {
    //
    @IBOutlet var tlustyText : UILabel!

    //
    func selfConfig(withDataItem item: DataItem) {
        //
        self.tlustyText?.text = item.jmeno;
    }
}
```

Oddělení dataSource z VC

- Znovupoužitelnost kódu:
  - Prototyp buňky a sebe-konfigurace.
- Generické TableViewControllery.
  - Pořád programujeme to samé chování, pracujme genericky.

Generický DataSource

```
struct DataItem {
    let jmeno : String
}

protocol SelfConfigurable {
    //
    associatedtype T;

    //
    func selfConfig(withDataItem item: T);
}

class BunkaJedna: UITableViewCell, SelfConfigurable {
    //
    @IBOutlet var tlustyText : UILabel!

    typealias T = DataItem;

    //
    func selfConfig(withDataItem item: T) {
        //
        self.tlustyText?.text = item.jmeno;
    }
}
```

DS

```
class TableModel<CELL:SelfConfigurable> : NSObject,
UITableViewDataSource {
    typealias T = CELL.T;

    //
    var data: [T] = []

    let cellID = "bunka1"

    // sections, rows: data
    func tableView(_ tableView: UITableView,
                   cellForRowAt indexPath: IndexPath) -> UITableViewCell
    {
        let cella = tableView.dequeueReusableCell(withIdentifier:
cellID) as! CELL

        //
        cella.selfConfig(withDataItem: data[indexPath.row])

        return cella as! UITableViewCell
    }
```

TVC + DS

```
class ShowTableVC: UITableViewController {
    //
    let _data = [DataItem(jmeno: "prvni"), DataItem(jmeno: "druhy")]
    var _ds : TableModel<BunkaJedna>!
    //
    override func viewDidLoad() {
        //
        super.viewDidLoad();

        _ds = TableModel<BunkaJedna>(withData:_data)
        self.tableView.dataSource = _ds

        //
        self.tableView.reloadData()
    }

}
```

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;

        DispatchQueue.global().async {
            //
            if let loaded = myModel.load(cosi) {
                // TADY by se hodilo atomicky
                if let ncell = tableView.cellForRow(at: indexPath) {
                    //
                    DispatchQueue.main.async {
                        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.

Charakter obsahu TV

- Formulář — pevná struktura, uživatelský obsah načíst/uložit.
- Homogenní seznam generovaných buněk, pevná délka.
- Seznam s přidáváním na konec.
- Seznam s plnou editací (vkládání, rušení).

Dynamika obsahu TableView

- Triviální — změní se model, tv.realoadData().
  - Pozor na selekci buňky — reload zruší selekci.
- Synchronizujeme pořadí záznamů model - tabulka.
  - Model by měl sdělit podstatu aktualizace tabulky: vložit na pozici, smazat na pozici.
  - Drží stav selekce.
  - NSFetchedResultsController.

Navigation VC, push VC

```
override func tableView(_ tableView: UITableView,
                didSelectRowAt indexPath: IndexPath)
    {
        //
        let story = UIStoryboard(name: "Main",
                                 bundle: Bundle.main);

        let det = story.instantiateViewController(withIdentifier:
"jeden")

        self.navigationController?.pushViewController(det, animated:
true);
    }
```

Příště

- Closures.
- Vlákna, operace, fronty.
- Grand Central Dispatch.