



# L'environnement de développement iOS

Brendan GUEGAN

Segment « Multi Devices »

Développeur iOS – Référent technique iOS.

[brendan.guegan@orange.com](mailto:brendan.guegan@orange.com)

+33 2 99 87 92 83

# sommaire

**partie 1 introduction**

**partie 2 Xcode**

**partie 3 UIKit**

**partie 4 Interface Builder**

**partie 5 Core Data**

**partie 6 autres frameworks**

**partie 7 ressources**

**partie 8 licences, certificats et profils d'approvisionnement**

# introduction

## historique

**2007** sortie de l'iPhone sous iOS 1 avec une mise à jour majeure tous les ans depuis

**2010** sortie de l'iPad sous iOS 3.2

**2013** version actuelle : iOS 7

# introduction

## les principaux outils

- Xcode : IDE pour développer sur iOS
- Interface Builder : l'outil pour définir les interfaces graphiques (intégré à Xcode depuis la version 4.0)
- iOS Simulator : pour simuler un iPhone ou un iPad sur sa machine (le simulateur n'a pas de ressources limitées, i.e. il utilise le processeur de la machine sans limitation)
- Instruments : pour détecter les fuites mémoires et observer les performances d'une application
- Application Loader : pour uploader des application vers l'AppStore
- Printer Simulateur : pour simuler une imprimante AirPrint

# sommaire

partie 1 introduction

partie 2 **Xcode**

partie 3 UIKit

partie 4 Interface Builder

partie 5 Core Data

partie 6 autres frameworks

partie 7 ressources

partie 8 licences, certificats et profils d'approvisionnement

# Xcode

## fichiers principaux - main.m

- point d'entrée de l'application
- contient le pool d'autorelease principal de l'application
- à ne modifier que très rarement

# Xcode

fichiers principaux - Prefix.pch

- header principal de l'application
- importé automatiquement dans tous les fichiers sources de l'application
- contient les imports à utiliser dans toute l'application

# Xcode

## fichiers principaux – Info.plist

- contient des réglages de l'application :
  - nom
  - version
  - identifiant (bundle identifier)
  - orientations supportées
  - ...

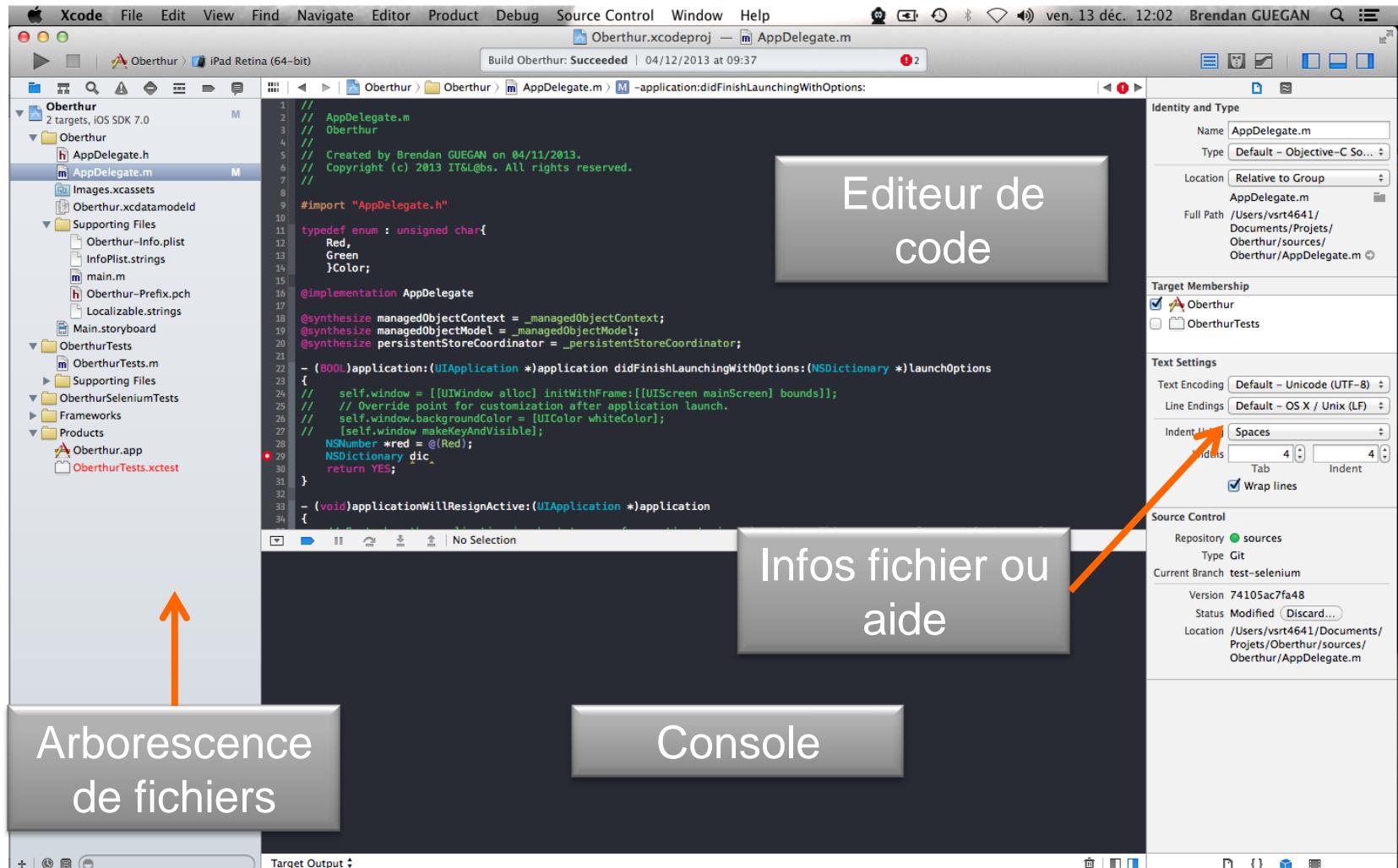
# Xcode

## fichiers principaux - AppDelegate

- classe qui reçoit les principaux évènements du cycle de vie de l'application et les notifications
- application:didFinishLaunchingWithOptions: : lancement de l'application
- applicationWillResignActive: : l'application devient inactive, par exemple lors de la réception d'un appel
- applicationDidBecomeActive: : l'application redevient active
- applicationDidEnterBackground: : l'application passe en arrière-plan
- applicationWillEnterForeground: : retour au premier plan
- applicationWillTerminate: : fin d'exécution (plus appelée par les applications supportant l'arrière plan)

# Xcode

## interface



# sommaire

**partie 1** introduction

**partie 2** Xcode

**partie 3** **UIKit**

**partie 4** Interface Builder

**partie 5** Core Data

**partie 6** autres frameworks

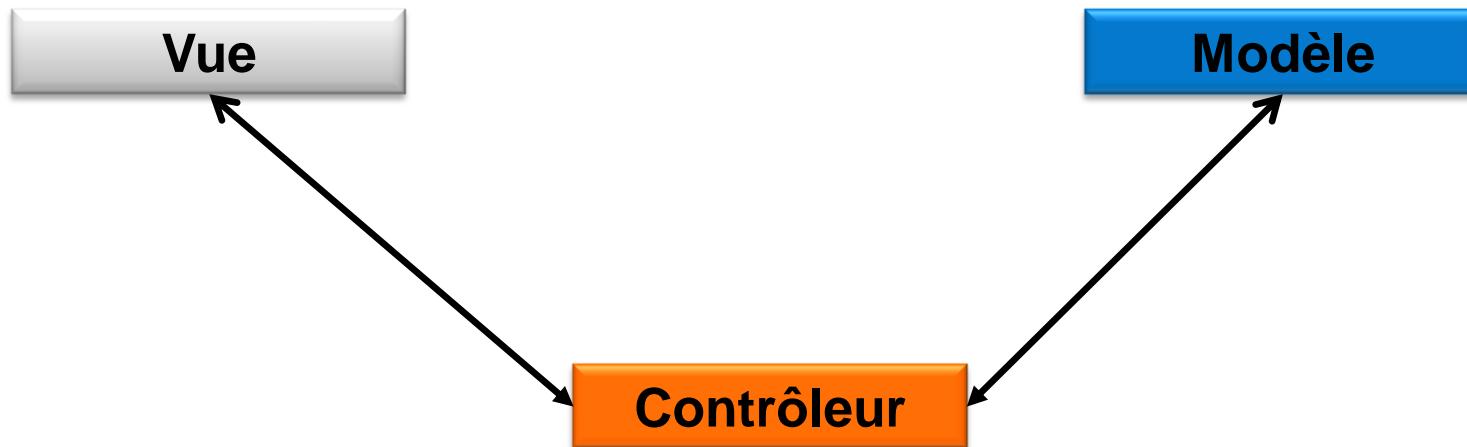
**partie 7** ressources

**partie 8** licences, certificats et profils d'approvisionnement

# UIKit

pattern MVC

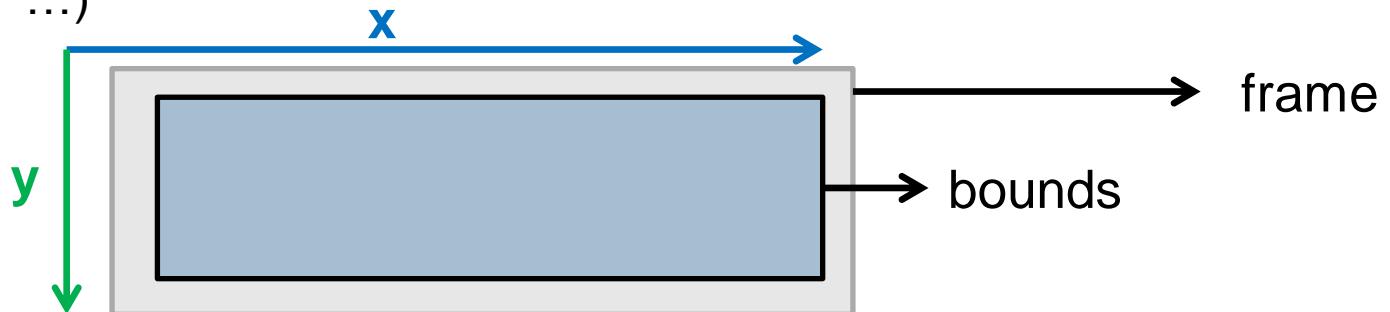
- séparation de la vue et du métier
- c'est un contrôleur qui gère l'interaction entre la vue et le modèle



# UIKit

## les vues - présentation

- classe de base d'une vue sur iOS : `UIView`
- hérite de `UIResponder` pour la gestion de l'interaction utilisateur (touches, secousses, ...)
- une vue :
  - est un rectangle, qui correspond à sa `frame`
  - reçoit des évènements
  - gère des sous-vues dans un sous-rectangle nommé `bounds`
  - peut être animée (position, taille, transparence, transformation 2D/3D, ...)



# UIKit

## les vues - principales classes

- **composants classiques** : UIButton, UILabel, UITextField, UITextView, UIImageView, UISlider, UIPickerView
- **conteneurs** : UICollectionView, UITableView, UIScrollView, UIWindow, UIWebView
- **autres** :
  - UIGestureRecognizer : reconnaissance de « gestes »
  - UIFont : police utilisée dans un label ou champ de texte
  - UIDevice, UIScreen : pour accéder à des données du périphérique

# UIKit

## les vues - collection view

- gère une liste de cellules regroupées en section qui défilent verticalement ou horizontalement avec pour principes :
  - on ne crée pas autant de cellules qu'il y a d'éléments à afficher, mais seulement les cellules qui sont visibles
  - les cellules qui deviennent non visibles sont réutilisées pour afficher d'autres éléments → diminution de la mémoire utilisée
  - la liste peut être de taille infinie et conserver un affichage fluide (les cellules ne doivent cependant pas être trop complexes)
- le contenu des cellules et l'interaction sont gérés via les mécanismes de délégation et data source

# UIKit

## les vues - collection view - délégation

- principe utilisé dans beaucoup de classes du framework iOS
- protocole décrivant des méthodes que la vue va utiliser pour son affichage ou pour indiquer un évènement, par exemple dans le cas de la collection view :
  - l'utilisateur a sélectionné/désélectionné une cellule
  - une cellule a été surlignée (highlight)
  - ...
- en général, le délégué d'une vue est déclaré via une propriété non retenue (assign ou weak) de la manière suivante :

```
@property (nonatomic, assign)  
id<UICollectionViewDelegate> delegate;
```

# UIKit

## les vues - collection view - data source

- utilisation similaire au mécanisme de délégation
- le data source a le contrôle sur les données affichées par la vue par exemple dans le cas de la collection view :
  - combien y'a-t-il de sections, de cellules dans telle section ?
  - quelle cellule afficher à tel endroit?
- en général, le dataSource d'une vue est déclaré via une propriété non retenue (assign ou weak) de la manière suivante :

```
@property (nonatomic, assign)  
id<UICollectionViewDataSource> dataSource;
```

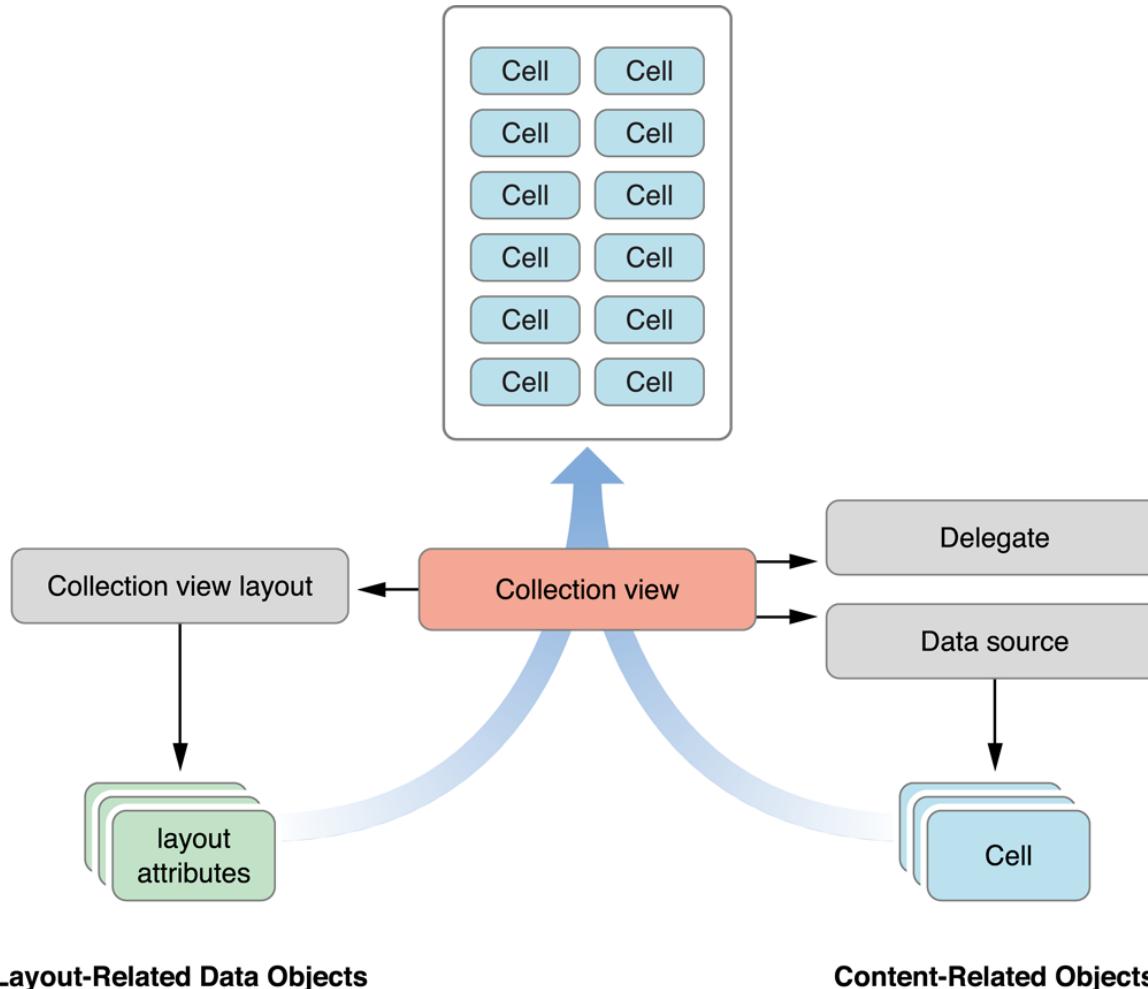
# UIKit

## les vues - collection view - layout

- responsable des positionnement, taille et attributs visuels des cellules
- classe de base : UICollectionViewLayout
- à ne sous-classer que si UICollectionViewFlowLayout n'est pas suffisant car la mécanique est complexe
- la classe UICollectionViewFlowLayout permet d'afficher sous forme d'une grille ou d'une liste avec un élément par ligne

# UIKit

## les vues - collection view - schéma récapitulatif



# UIKit

## les vues - collection view - utilisation du dataSource

- voici un exemple d'implémentation : la classe qui implémente UICollectionViewDataSource possède une variable membre nommée texts qui est un tableau de chaînes

### Exemple

```
- (NSInteger) collectionView: (UICollectionView*) collectionView  
    numberOfItemsInSection: (NSInteger) section{  
    return [texts count];  
}
```

# UIKit

## les vues - collection view - utilisation du dataSource

### Exemple

```
- (UICollectionViewCell*)  
    collectionView: (UICollectionView*)collectionView  
    cellForItemAtIndexPath: (NSIndexPath*)indexPath{  
// identifiant du type de cellule  
    NSString *identifier = @"item 1";  
// on demande une cellule réutilisable à la vue  
    UICollectionViewCell *cell = [collectionView  
        dequeueReusableCellWithReuseIdentifier:identifier  
        forIndexPath:indexPath];  
    cell.textLabel.text = [texts objectAtIndex:indexPath.row];  
    return cell;  
}
```

# UIKit

## les vues - collection view - utilisation du dataSource

### Exemple

```
- (UICollectionViewCell*)  
    collectionView: (UICollectionView*)collectionView  
    cellForItemAtIndexPath: (NSIndexPath*)indexPath{  
    // identifiant du type de cellule  
    NSString *identifier = @"item 1";  
    // on demande une cellule réutilisable à la vue  
    UICollectionViewCell *cell = [collectionView  
        dequeueReusableCellWithReuseIdentifier:identifier  
        forIndexPath:indexPath];  
    cell.textLabel.text = [texts objectAtIndex:indexPath.row];  
    return cell;  
}
```

# UIKit

## les vues - collection view - utilisation du dataSource

### Exemple

```
- (UICollectionViewCell*)  
    collectionView: (UICollectionView*)collectionView  
    cellForItemAtIndexPath: (NSIndexPath*)indexPath{  
// identifiant du type de cellule  
    NSString *identifier = @"item 1";  
// on demande une cellule réutilisable à la vue  
    UICollectionViewCell *cell = [collectionView  
        dequeueReusableCellWithReuseIdentifier:identifier  
        forIndexPath:indexPath];  
    cell.textLabel.text = [texts objectAtIndex:indexPath.row];  
    return cell;  
}
```

# UIKit

## les vues - collection view - utilisation du dataSource

- la méthode `dequeueReusableCellWithIdentifier: forIndexPath`: permet de récupérer un type de cellule enregistré auprès de la collection view
- 3 méthodes pour enregistrer un type de cellule :
  - `registerClass:forCellReuseIdentifier:`
  - `registerNib:forCellReuseIdentifier:`
  - par interface builder
- les 2 premières méthodes sont en général appelées dans le méthode `viewDidLoad` de la classe `UIViewController`
- l'identifiant de réutilisation identifie un **type de cellule**

# UIKit

## les vues - collection view - utilisation du dataSource

### Exemple

```
- (void) viewDidLoad {
    ...
    [myCollectionView registerClass: [MyCustomCell class]
        forCellReuseIdentifier:@"item 1"];
    ...
}
```

# UIKit

## les vues - collection view - utilisation du délégué

- le délégué est facultatif et doit être conforme au protocole `UICollectionViewDelegate`
- le délégué indique :
  - la sélection/désélection d'une cellule par l'utilisateur
  - gestion du menu (copier-coller)

# UIKit

## les contrôleurs - présentation

- tout contrôleur est un `UIViewController`
- gestion de l'interaction entre vues et données
- gère des sous-contrôleurs
- reçoit les évènements du cycle de vie de sa vue
  - `viewDidLoad` : la vue a été chargée
  - `viewDidAppear/Disappear` : la vue a été affichée/masquée
  - `viewWillAppear/Disappear` : la vue va être affichée/masquée
- 2 possibilités pour créer sa vue :
  - soit en code (via `viewDidLoad` ou `loadView`)
  - soit via Interface Builder

# UIKit

## les contrôleurs - présentation

- gestion des sauvegarde et restauration de l'état
- gestion des contraintes d'affichage
- gestion d'alertes mémoire
  - méthode `didReceiveMemoryWarning`
  - relâchement des objets graphiques
  - relâchement des objets non-critiques, i.e. qui peuvent être recréés par le contrôleur

# UIKit

## les contrôleurs - présentation - gestion mémoire

### Exemple

```
- (void) didReceiveMemoryWarning {
    [super didReceiveMemoryWarning];
    // Add code to clean up any of your own resources that are
    // no longer necessary.
    if ([self.view window] == nil)
    {
        // Add code to preserve data stored in the views that
        // might be needed later.
        ...
        // Add code to clean up other strong references to the
        // view in the view hierarchy.
        self.view = nil;
    }
}
```

# UIKit

les contrôleurs - présentation - cycles de vie

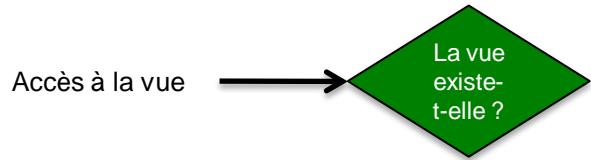
# UIKit

## les contrôleurs - présentation - cycles de vie

Accès à la vue

# UIKit

## les contrôleurs - présentation - cycles de vie



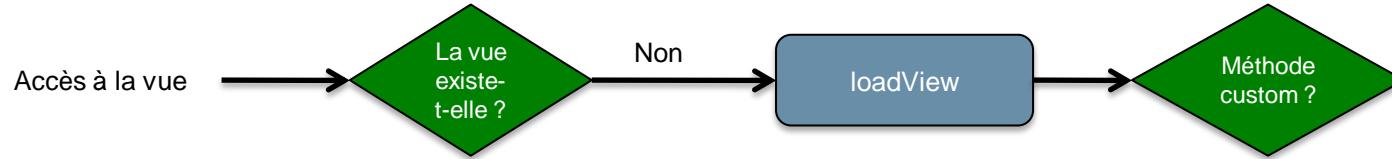
# UIKit

## les contrôleurs - présentation - cycles de vie



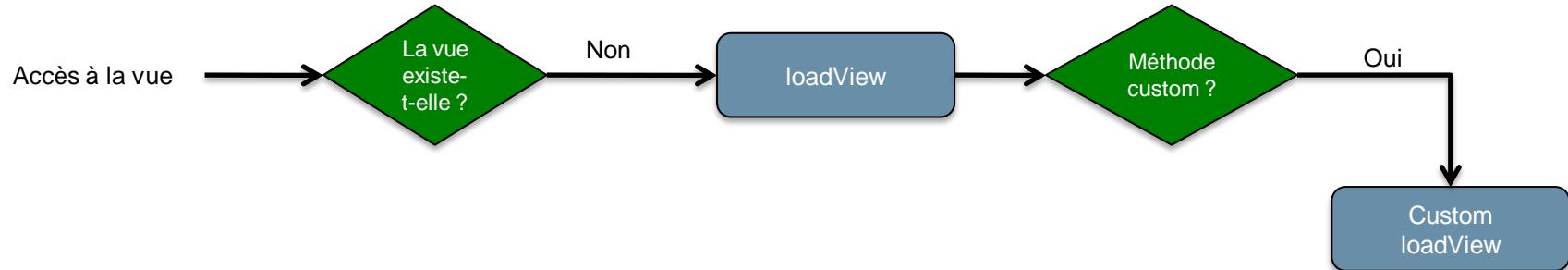
# UIKit

## les contrôleurs - présentation - cycles de vie



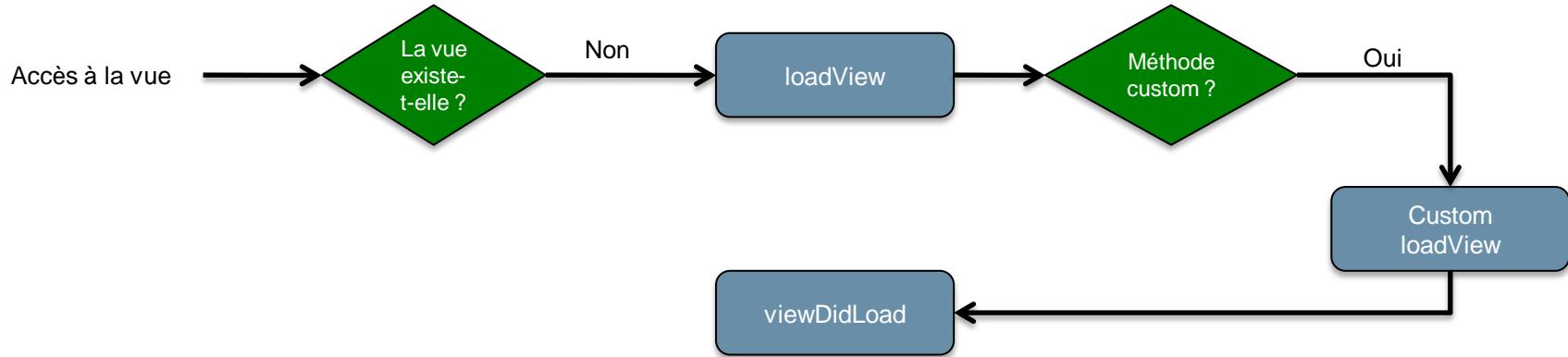
# UIKit

## les contrôleurs - présentation - cycles de vie



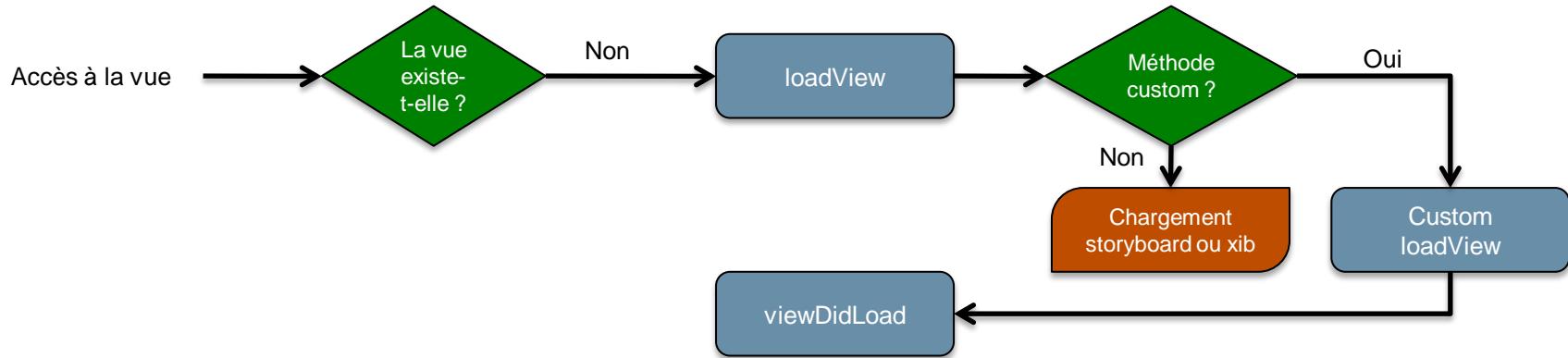
# UIKit

## les contrôleurs - présentation - cycles de vie



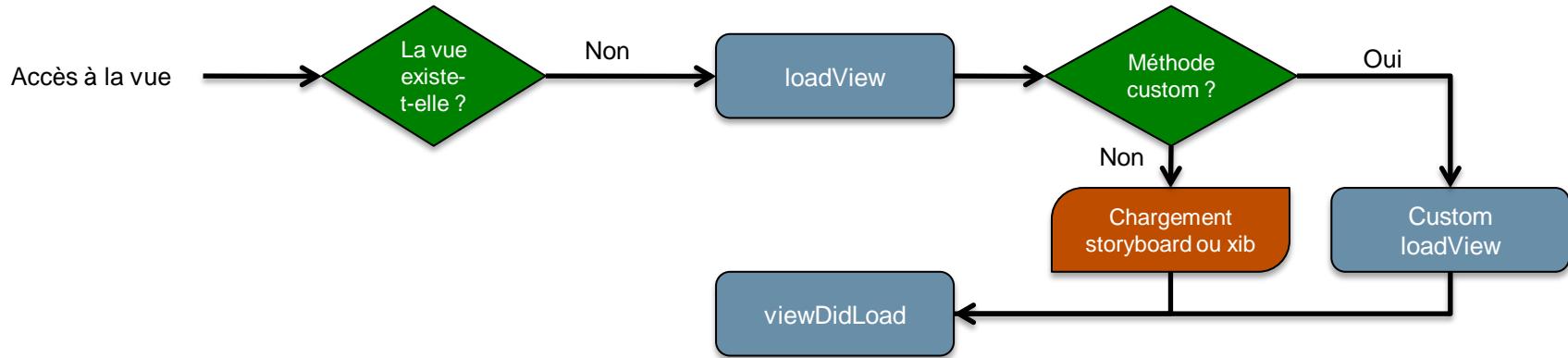
# UIKit

## les contrôleurs - présentation - cycles de vie



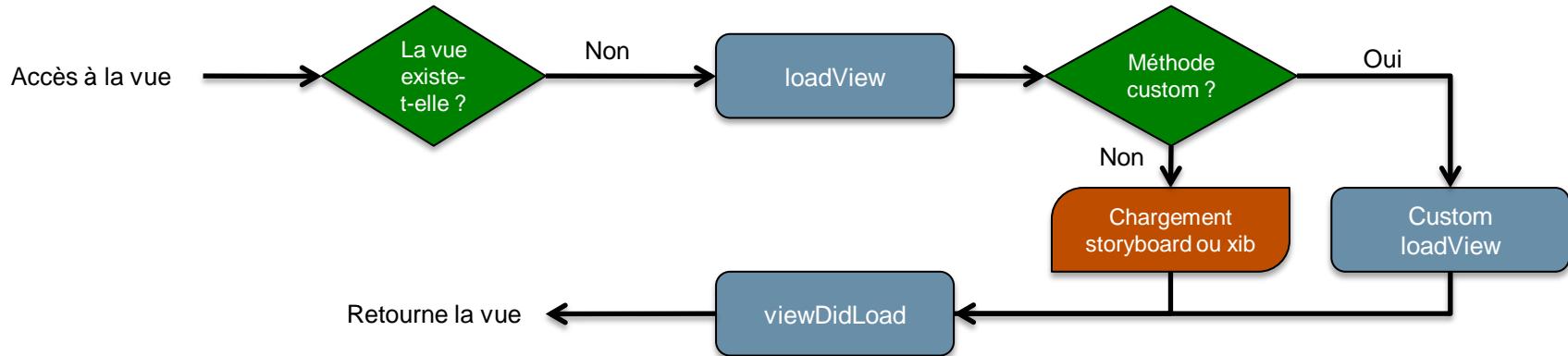
# UIKit

## les contrôleurs - présentation - cycles de vie



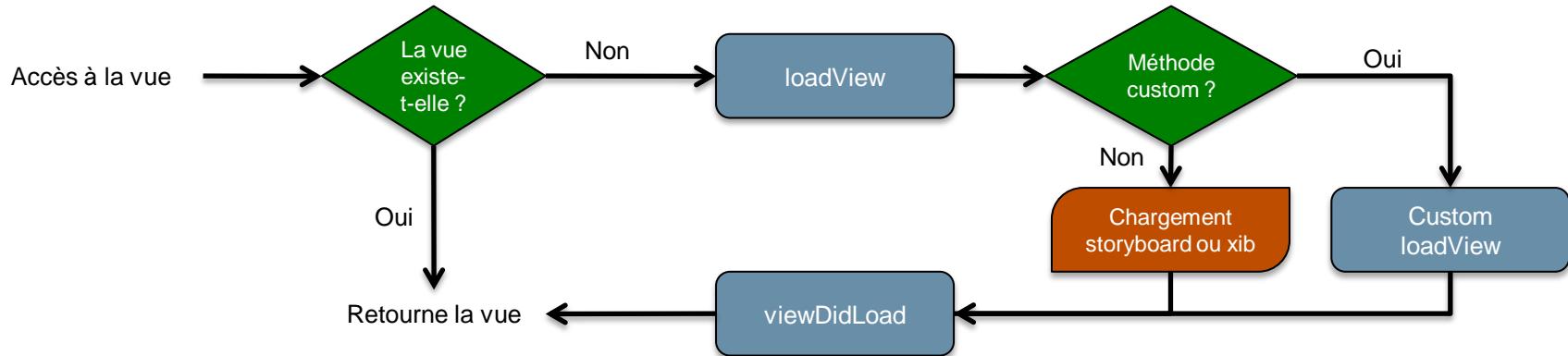
# UIKit

## les contrôleurs - présentation - cycles de vie



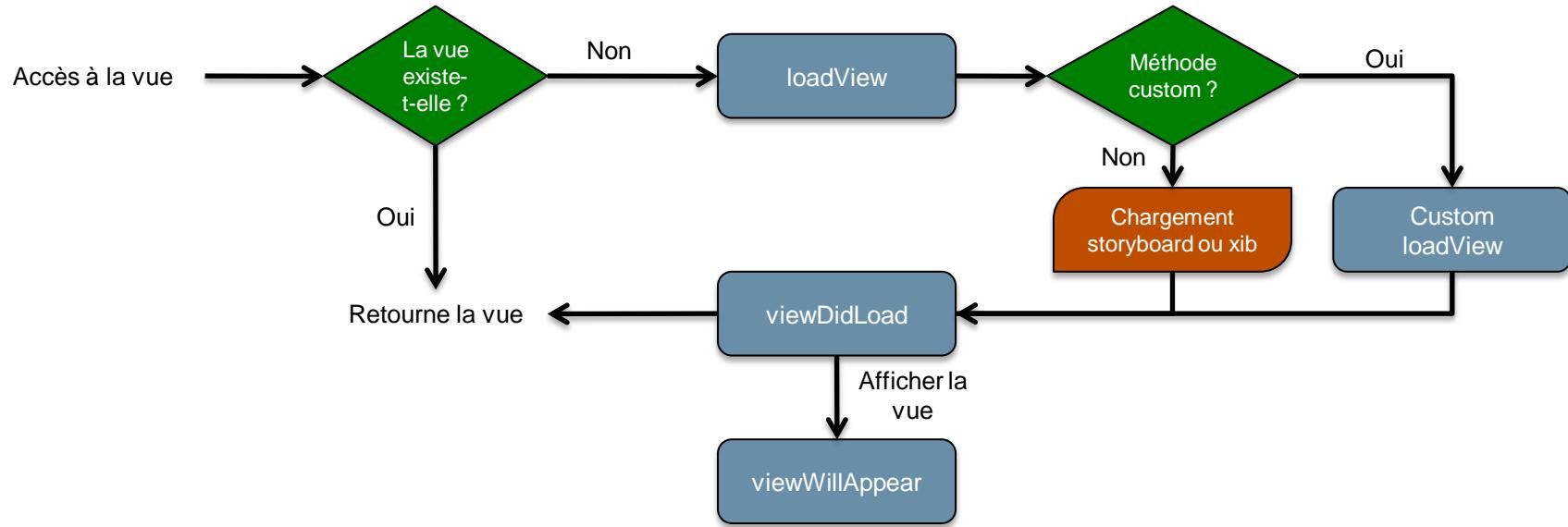
# UIKit

## les contrôleurs - présentation - cycles de vie



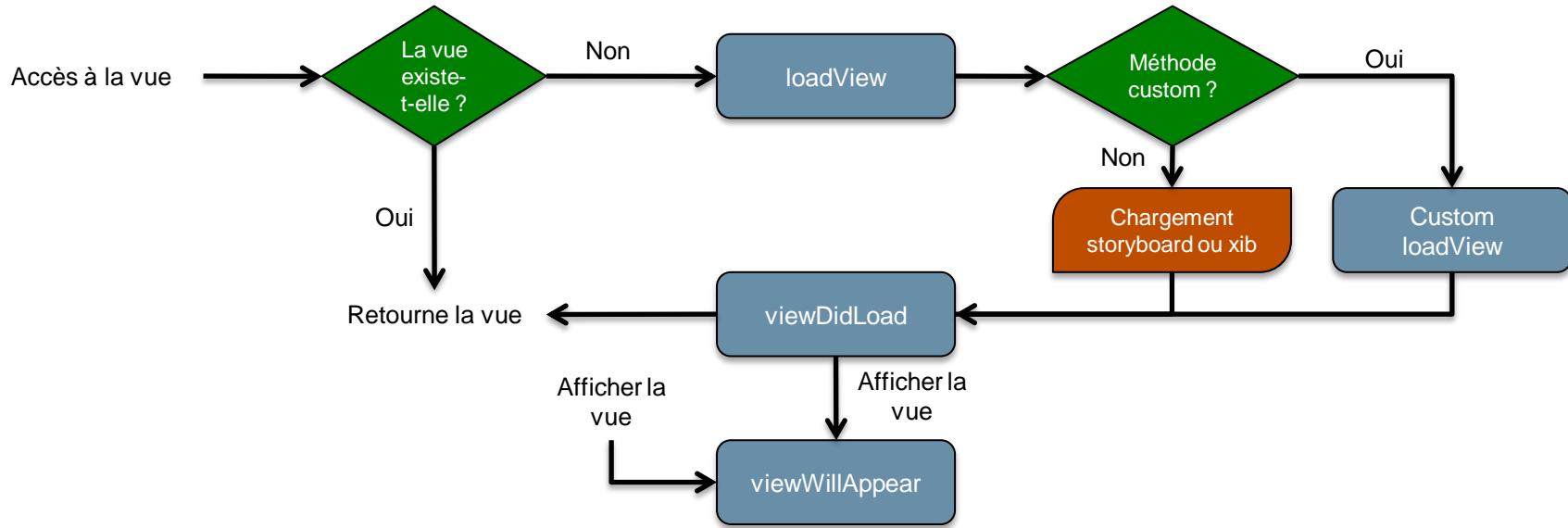
# UIKit

## les contrôleurs - présentation - cycles de vie



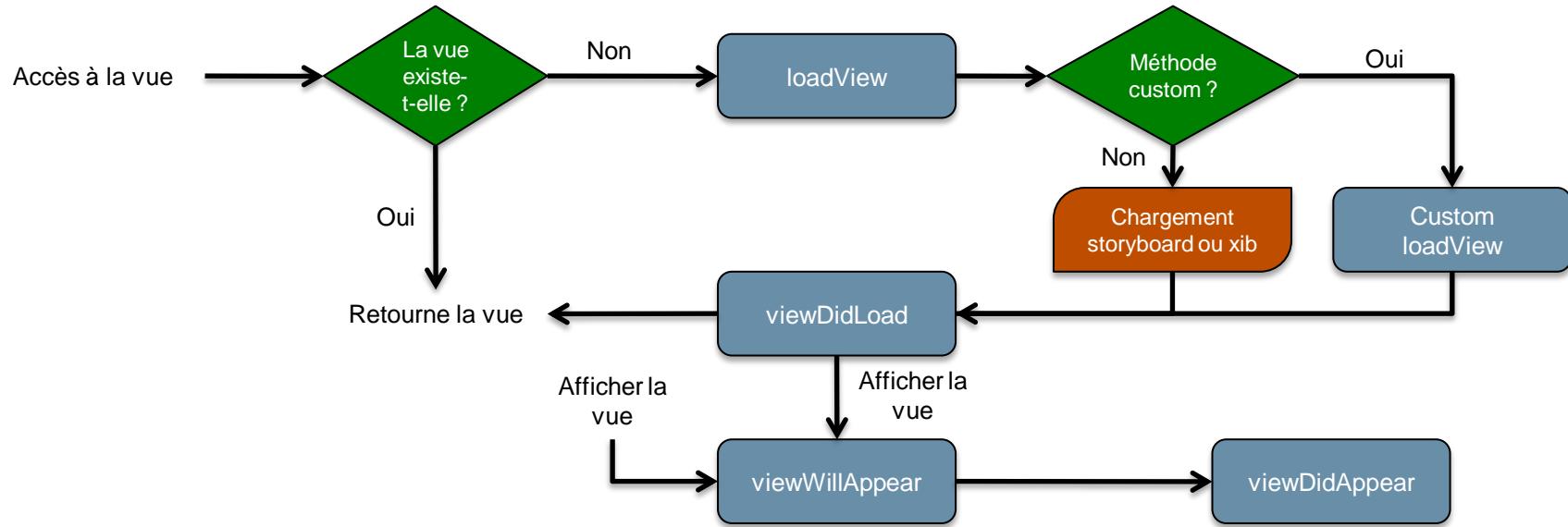
# UIKit

## les contrôleurs - présentation - cycles de vie



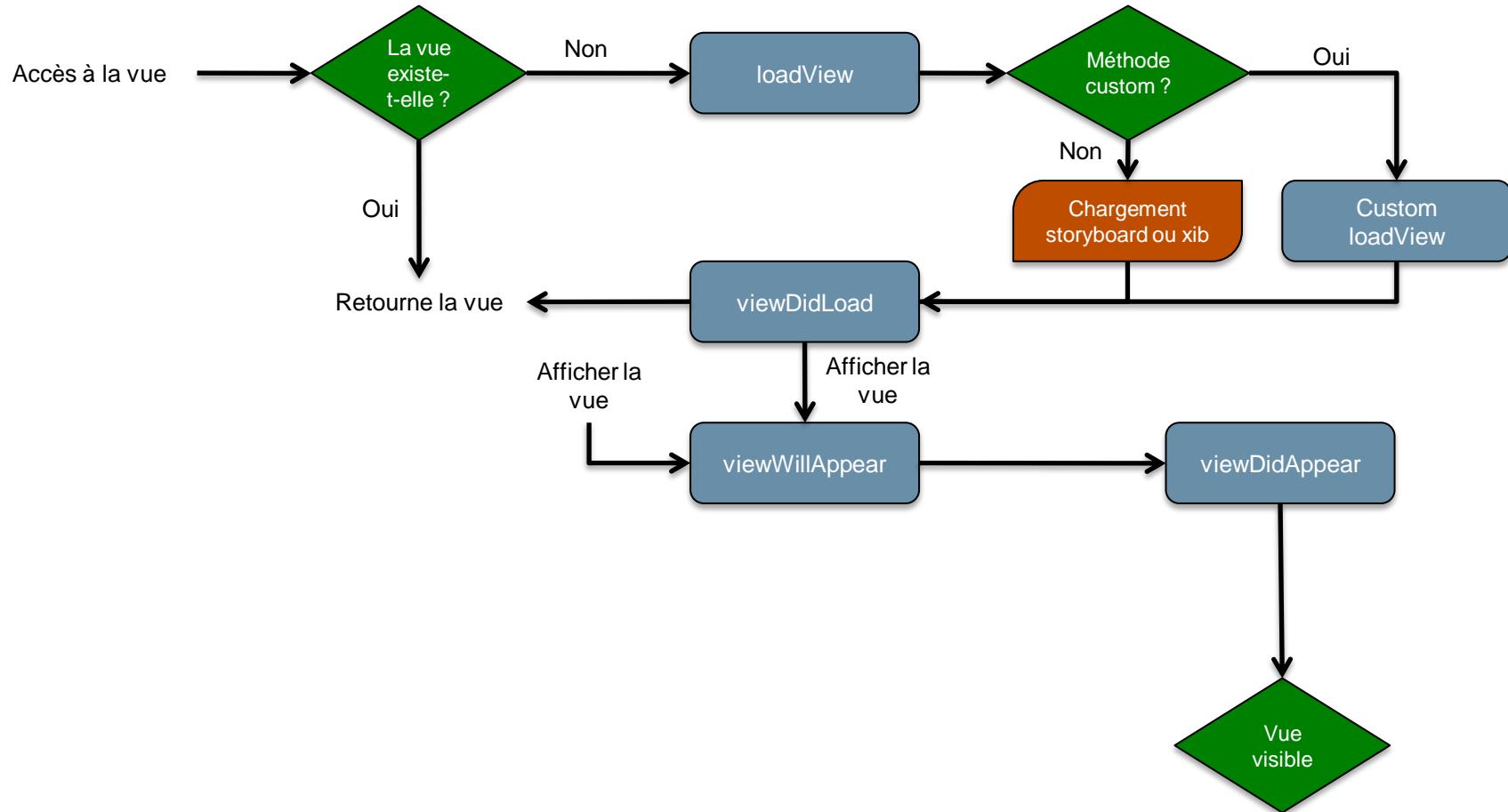
# UIKit

## les contrôleurs - présentation - cycles de vie



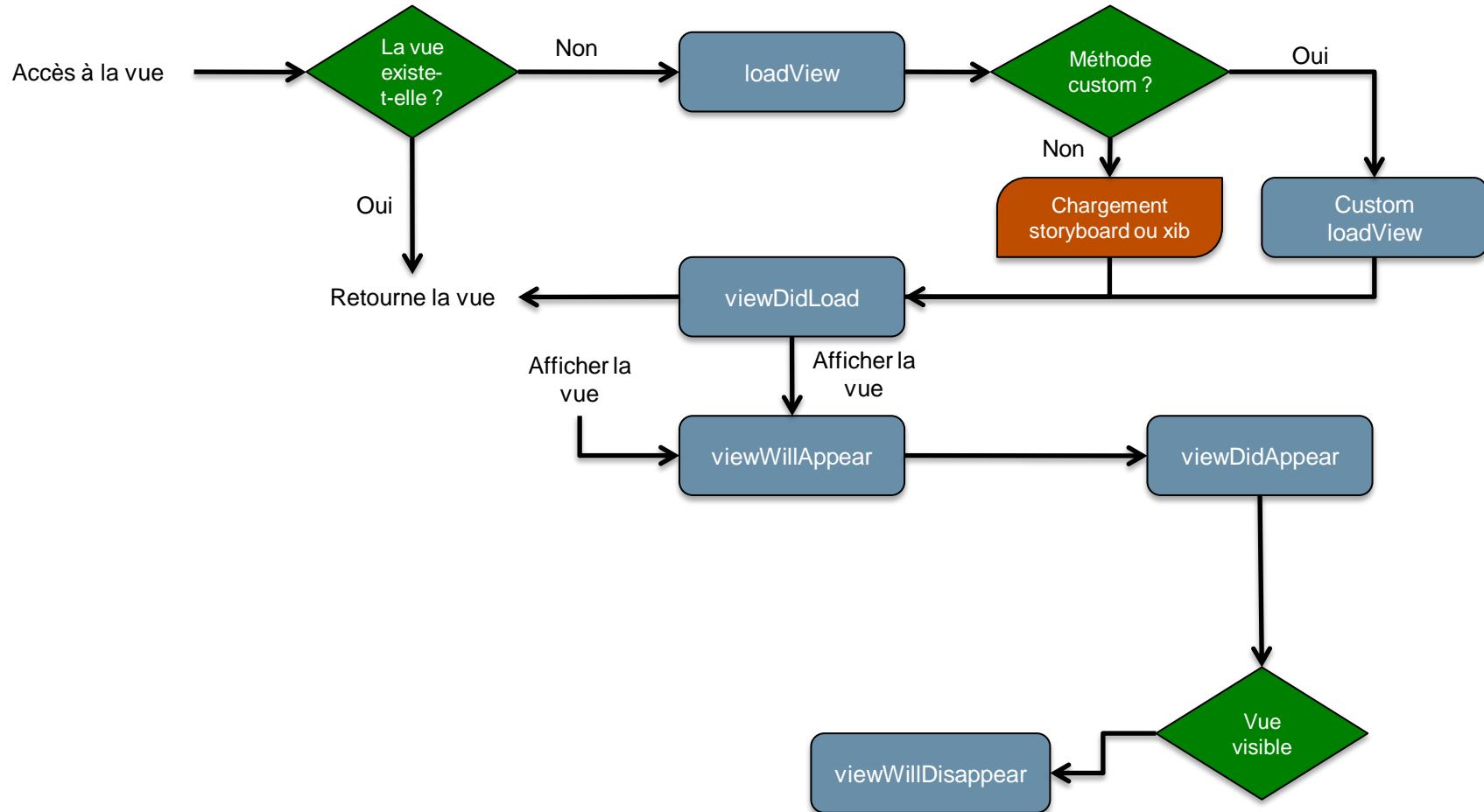
# UIKit

## les contrôleurs - présentation - cycles de vie



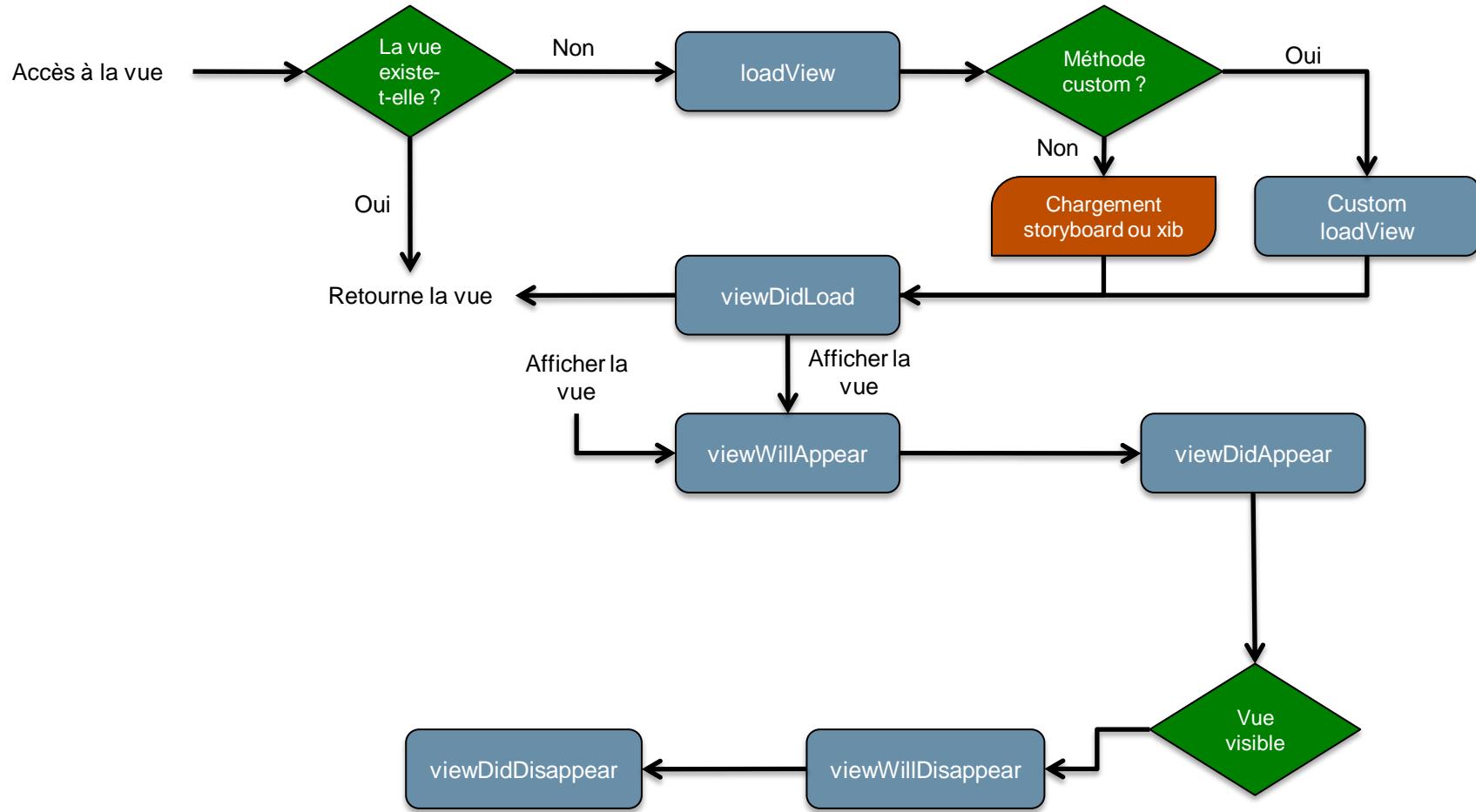
# UIKit

## les contrôleurs - présentation - cycles de vie



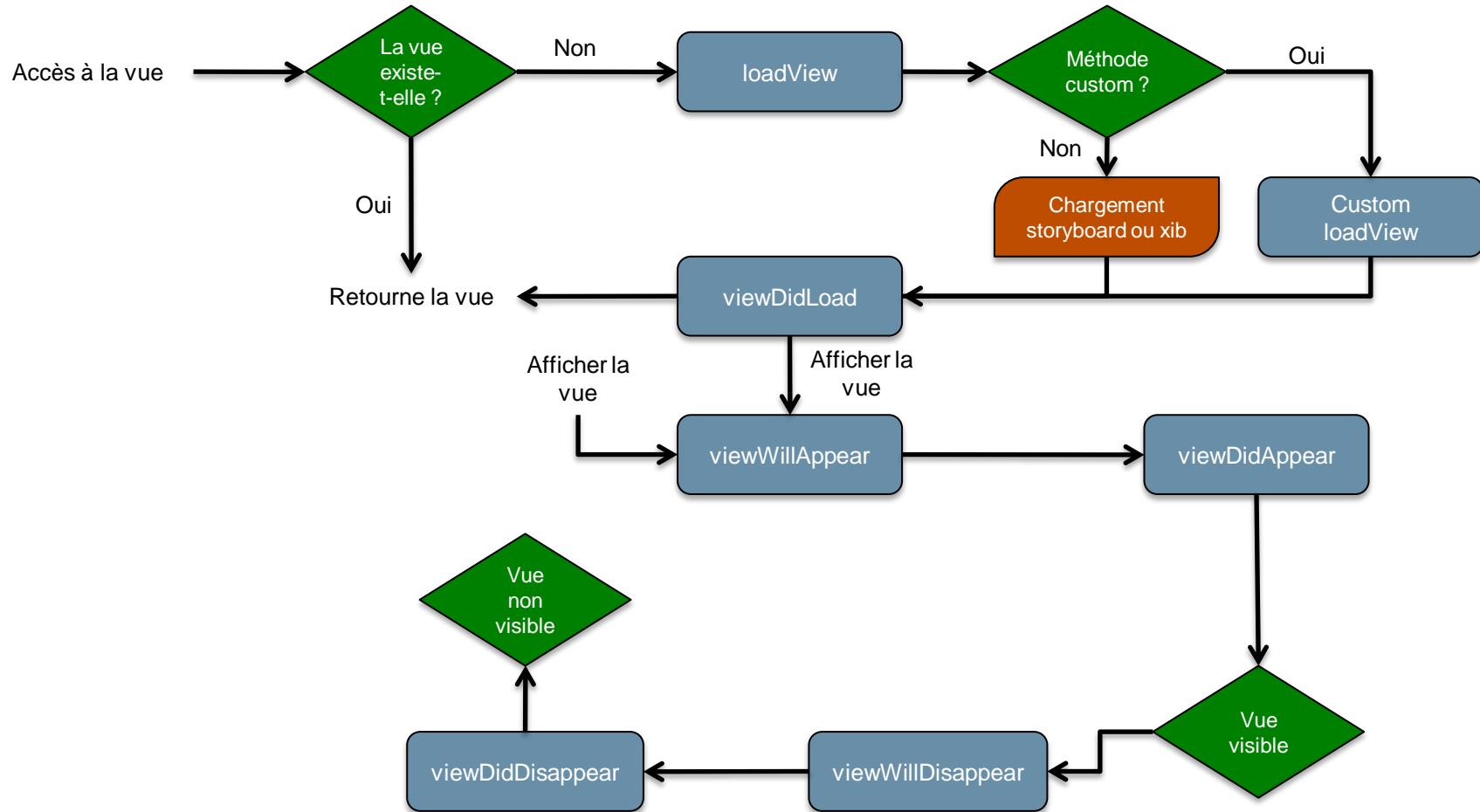
# UIKit

## les contrôleurs - présentation - cycles de vie



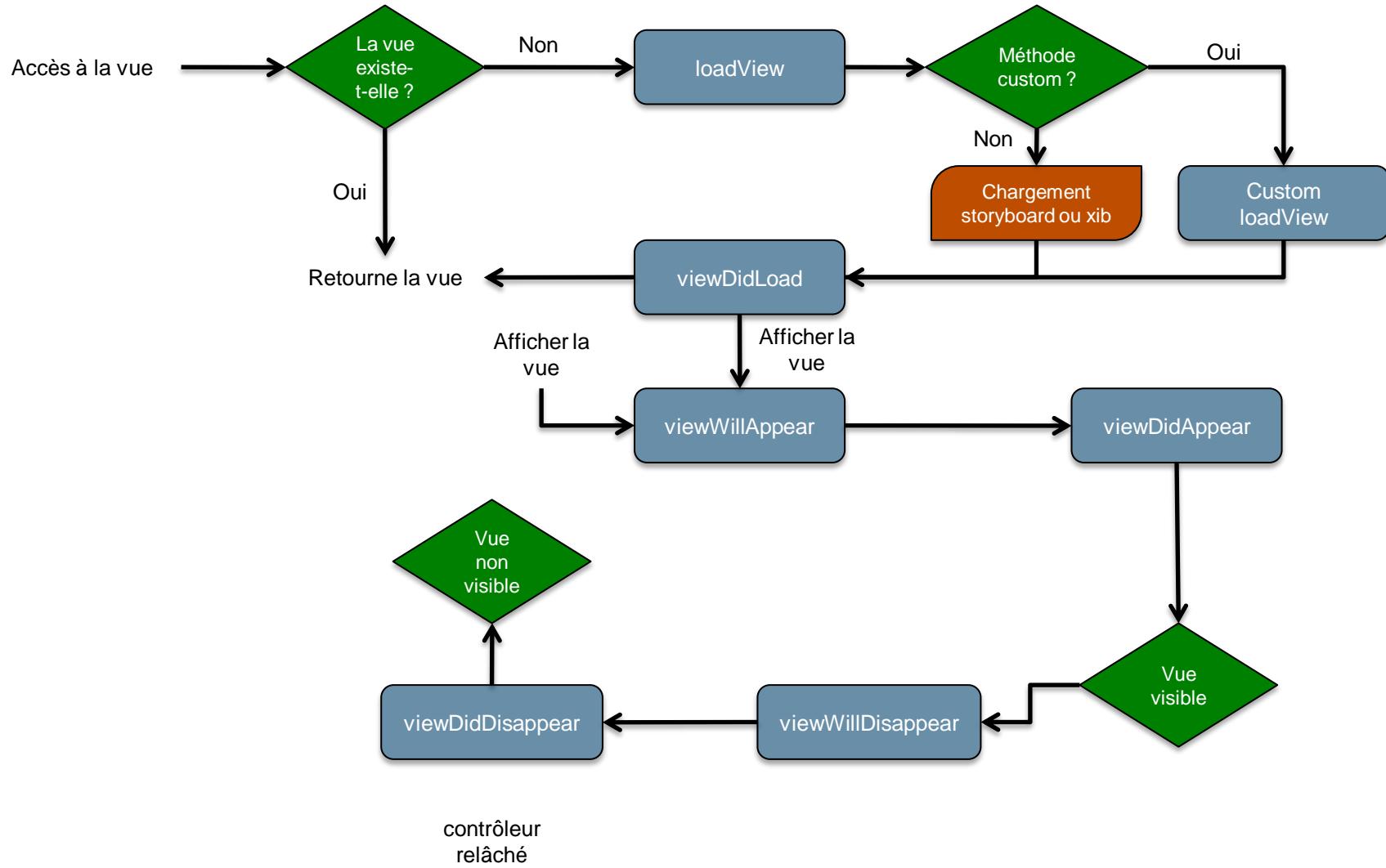
# UIKit

## les contrôleurs - présentation - cycles de vie



# UIKit

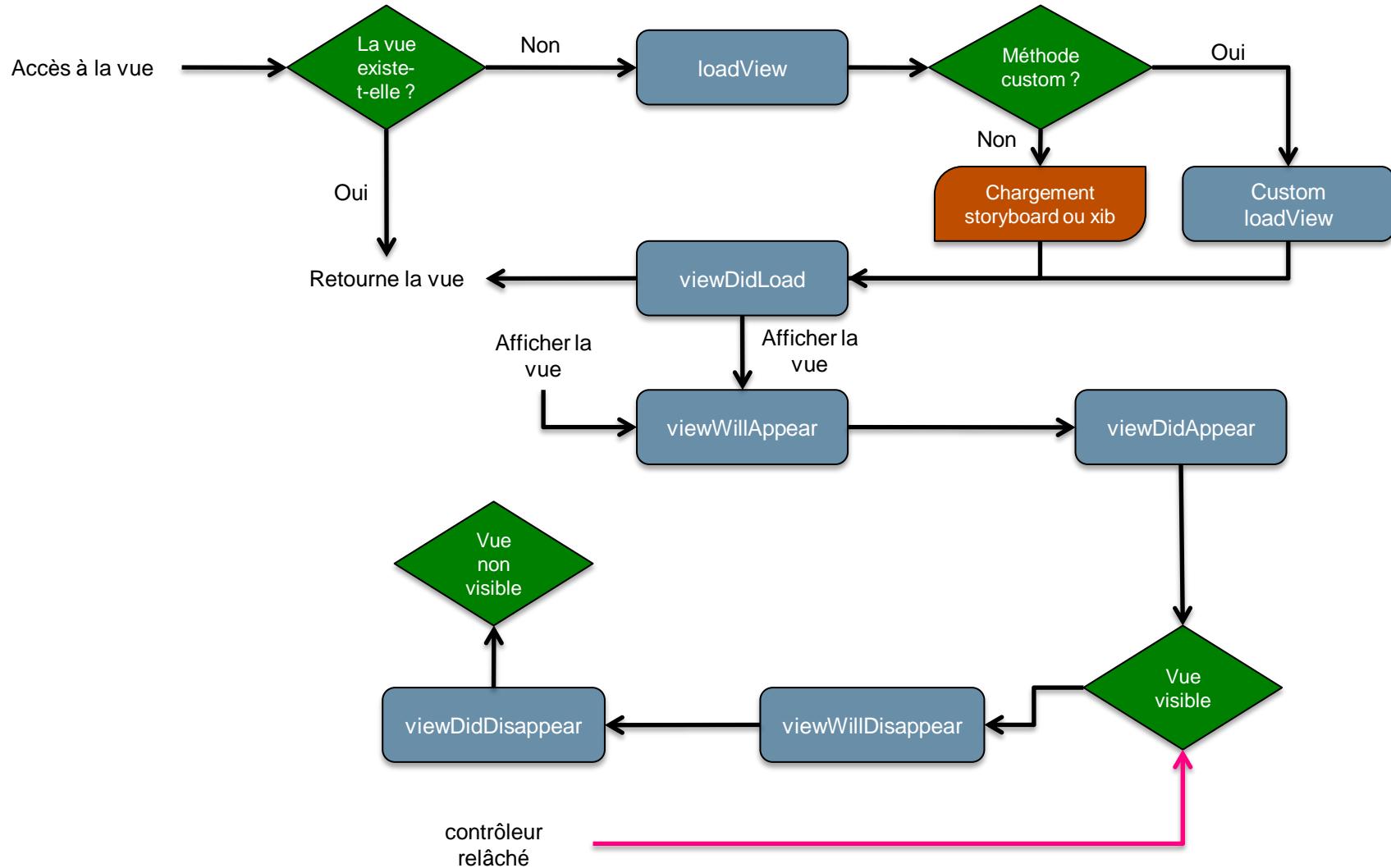
## les contrôleurs - présentation - cycles de vie



contrôleur  
relâché

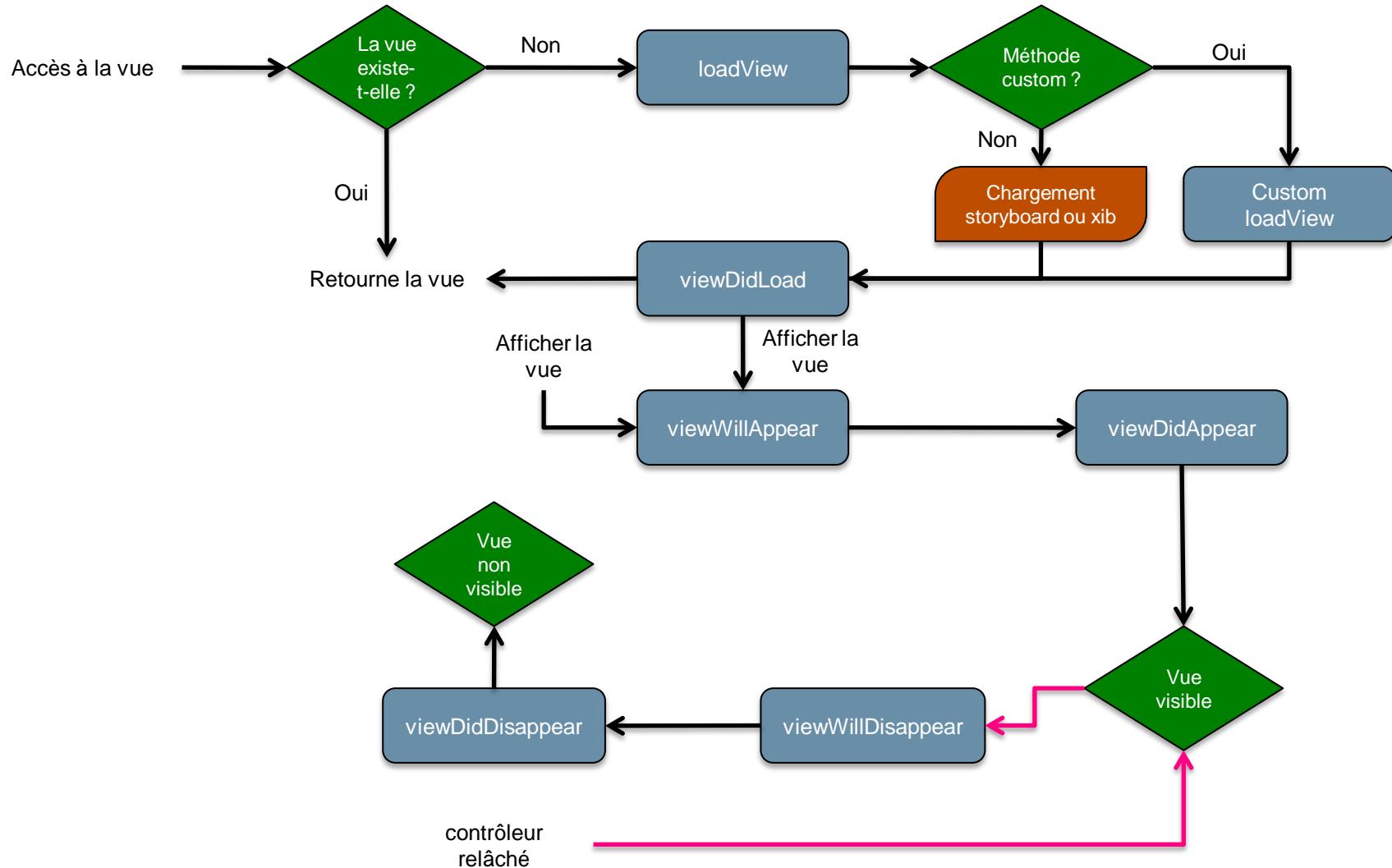
# UIKit

## les contrôleurs - présentation - cycles de vie



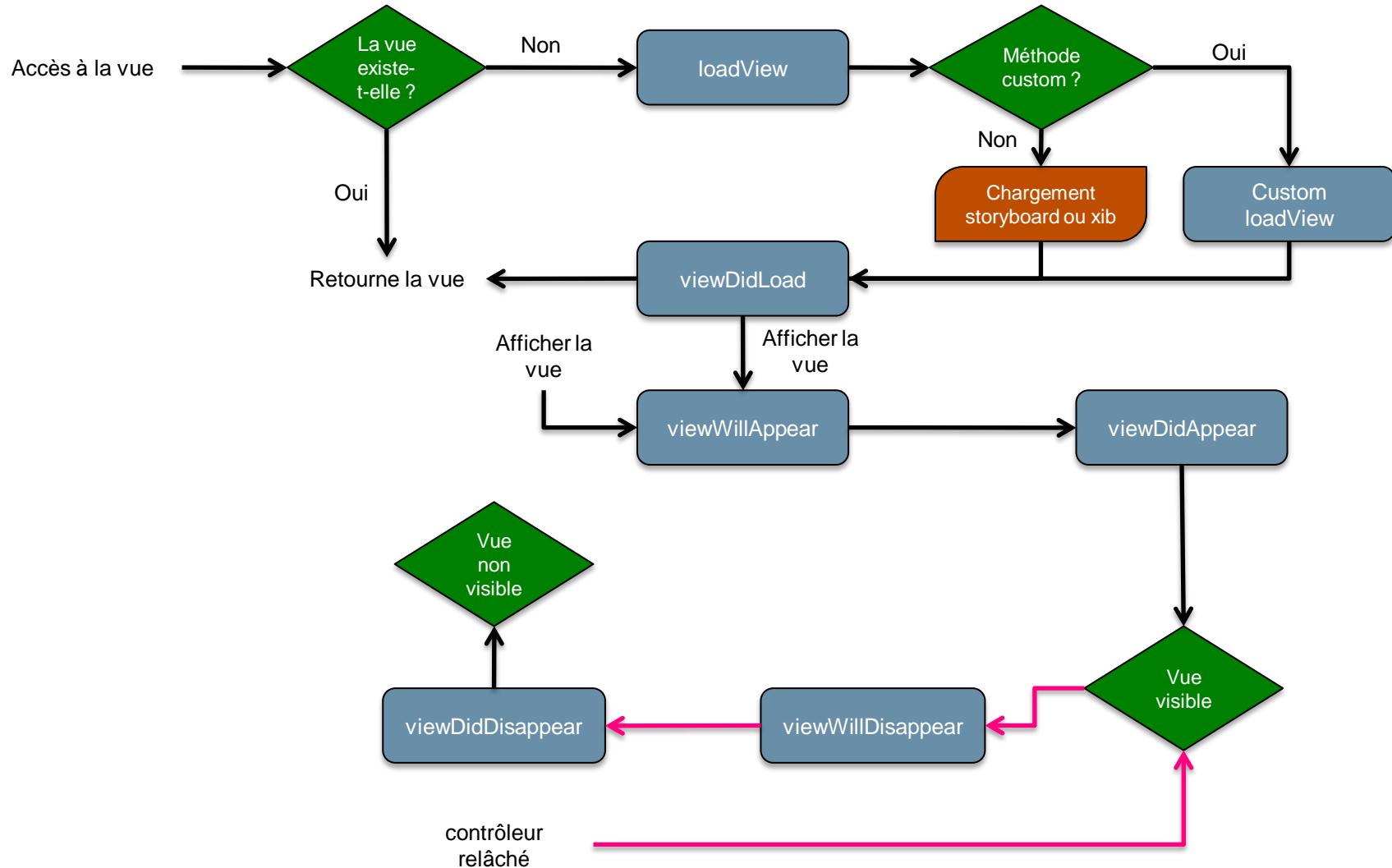
# UIKit

## les contrôleurs - présentation - cycles de vie



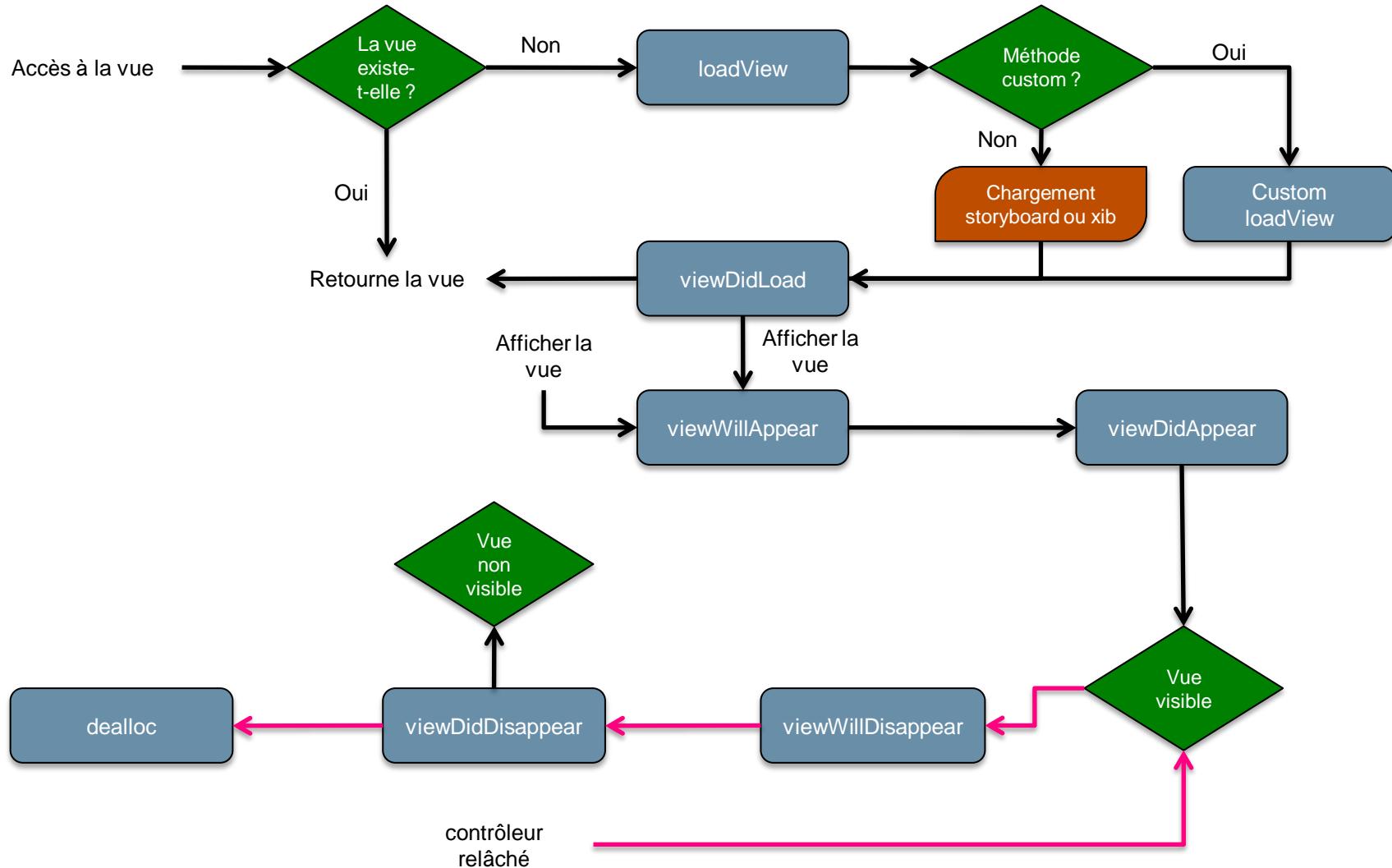
# UIKit

## les contrôleurs - présentation - cycles de vie



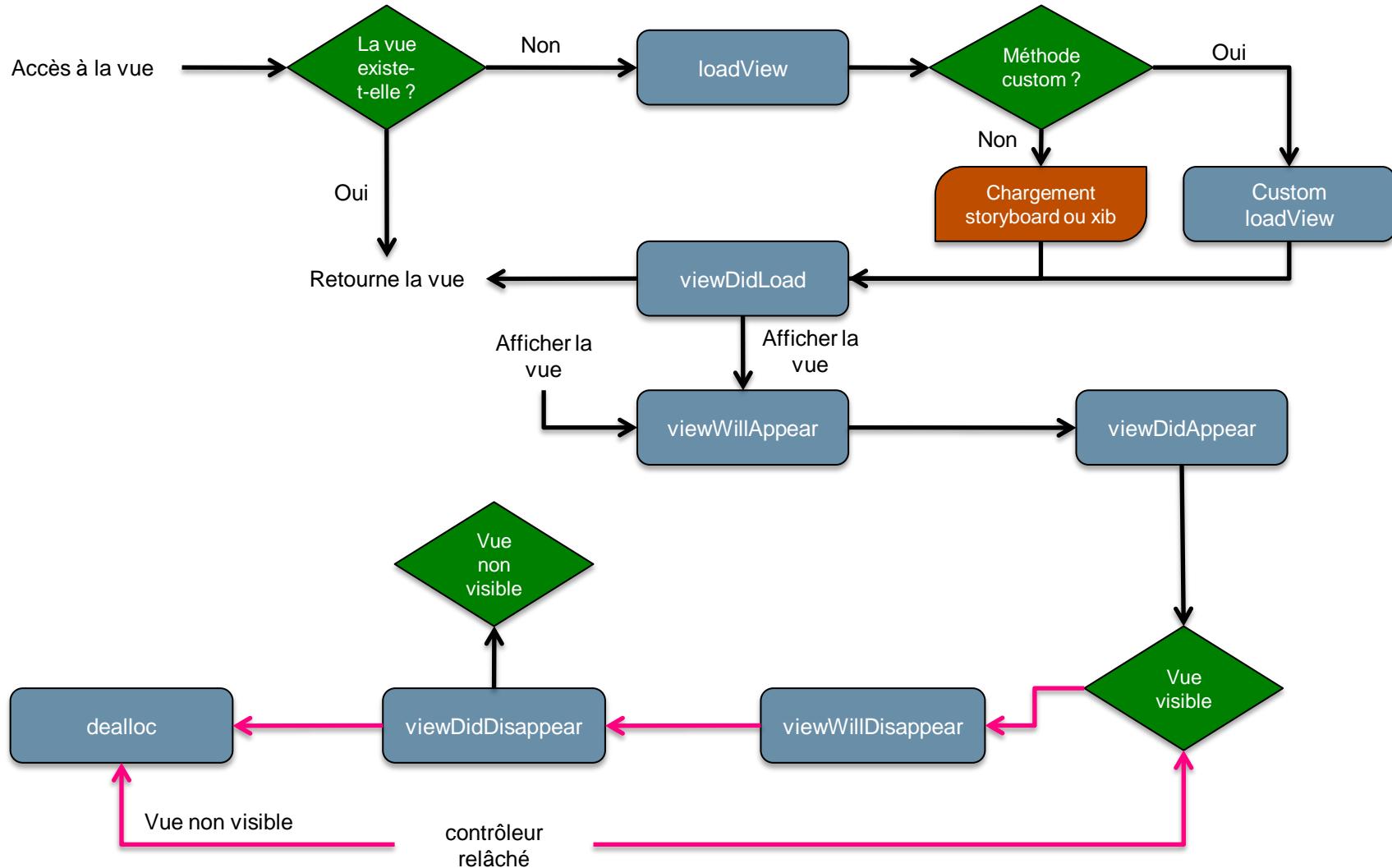
# UIKit

## les contrôleurs - présentation - cycles de vie



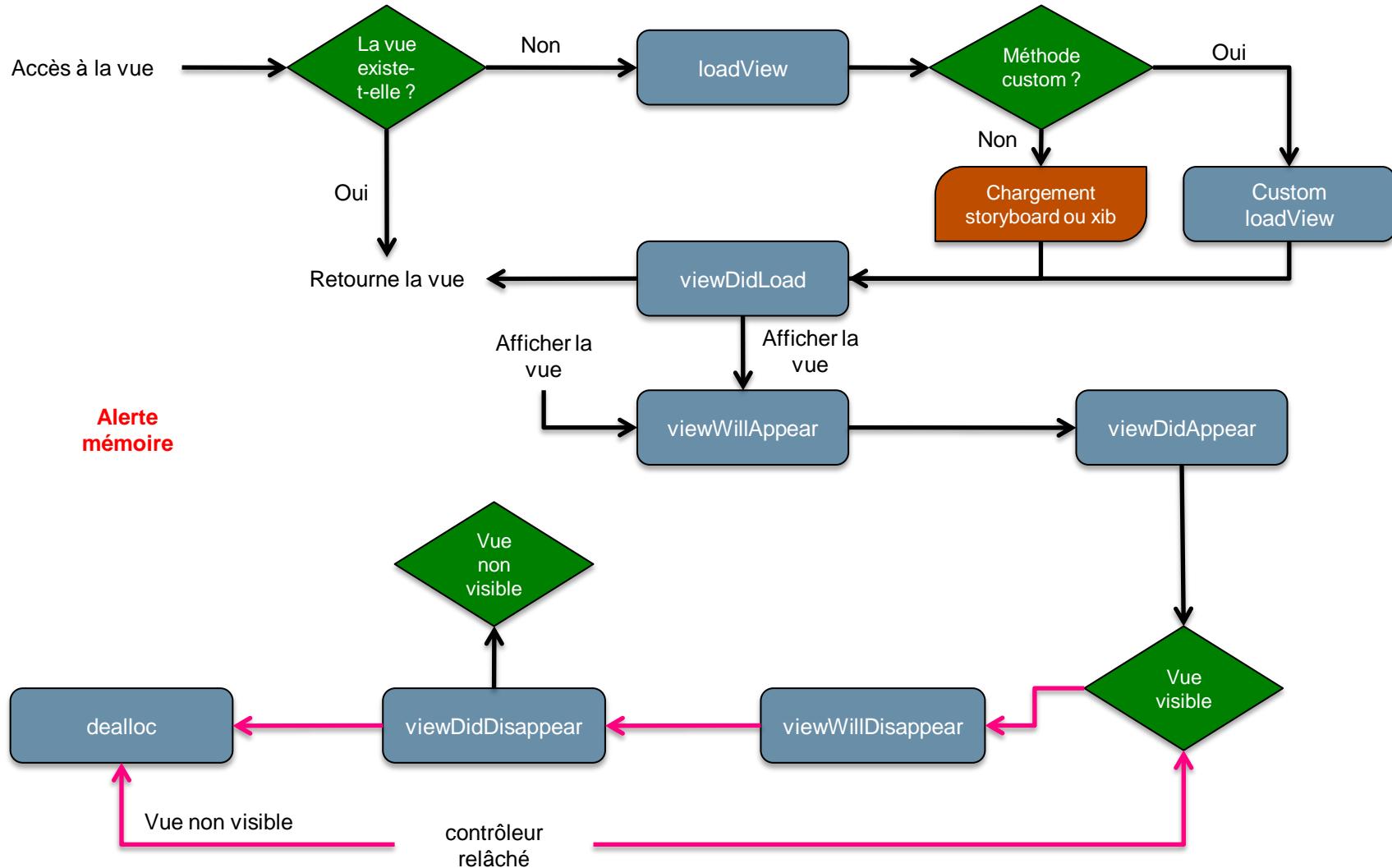
# UIKit

## les contrôleurs - présentation - cycles de vie



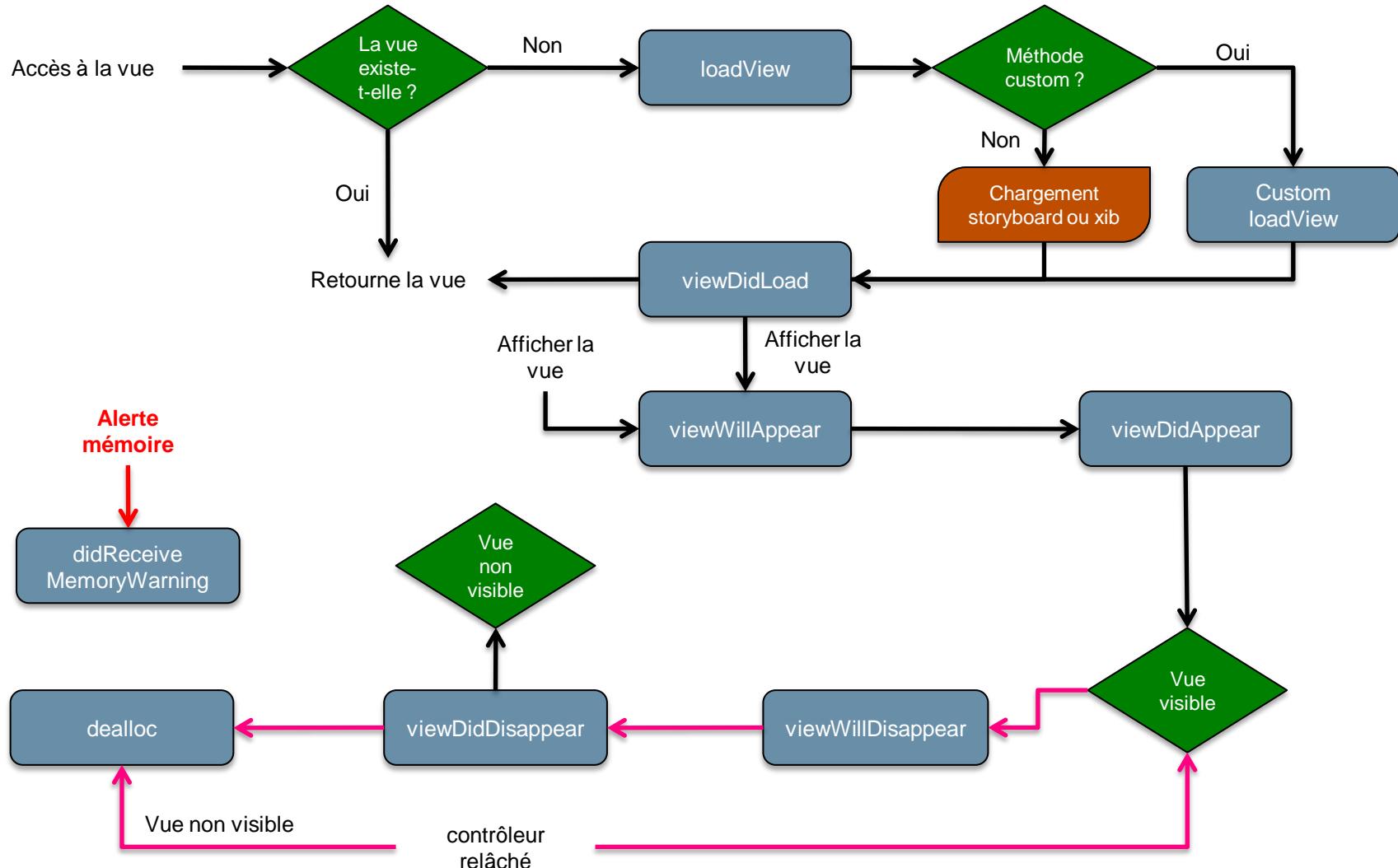
# UIKit

## les contrôleurs - présentation - cycles de vie



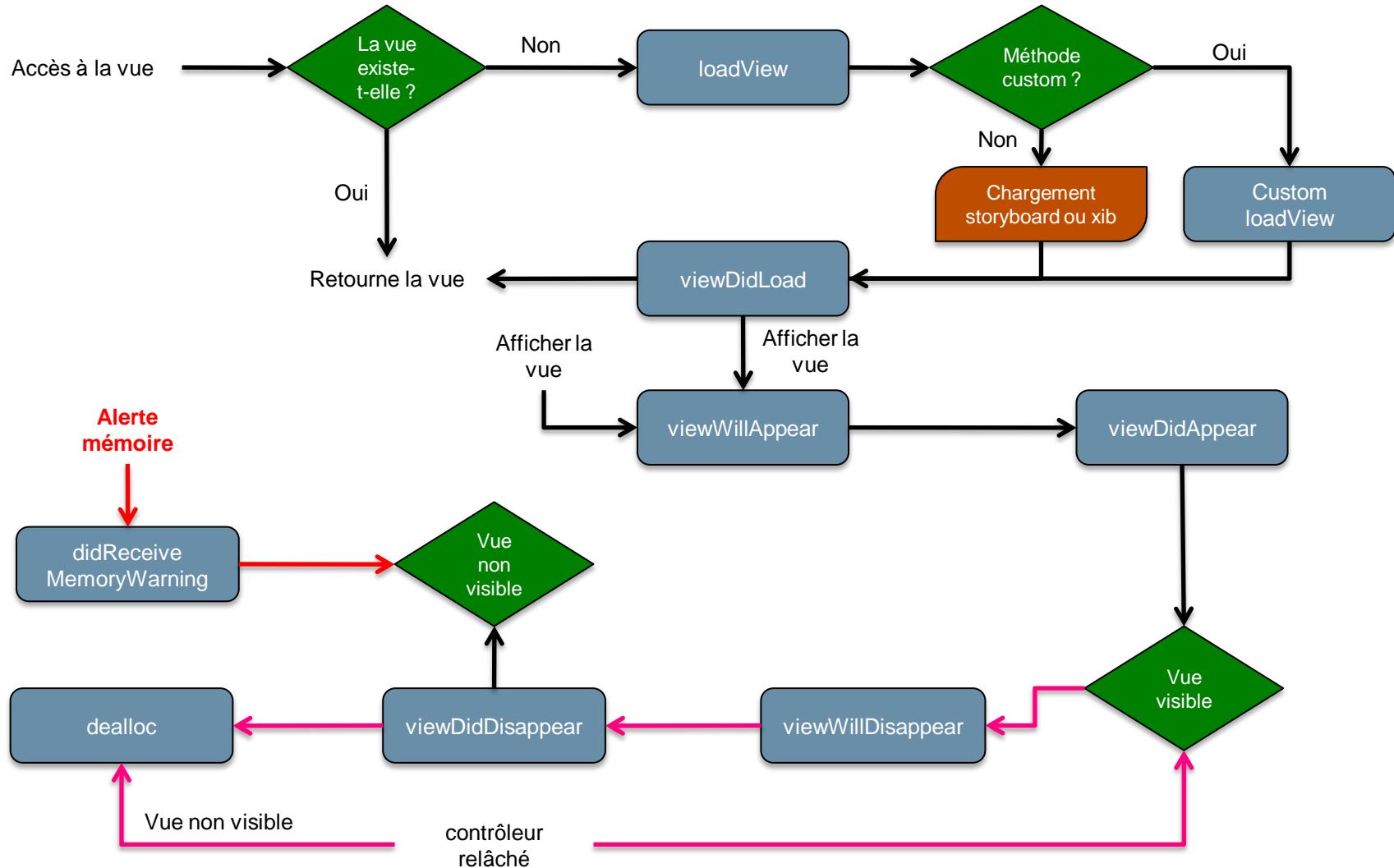
# UIKit

## les contrôleurs - présentation - cycles de vie



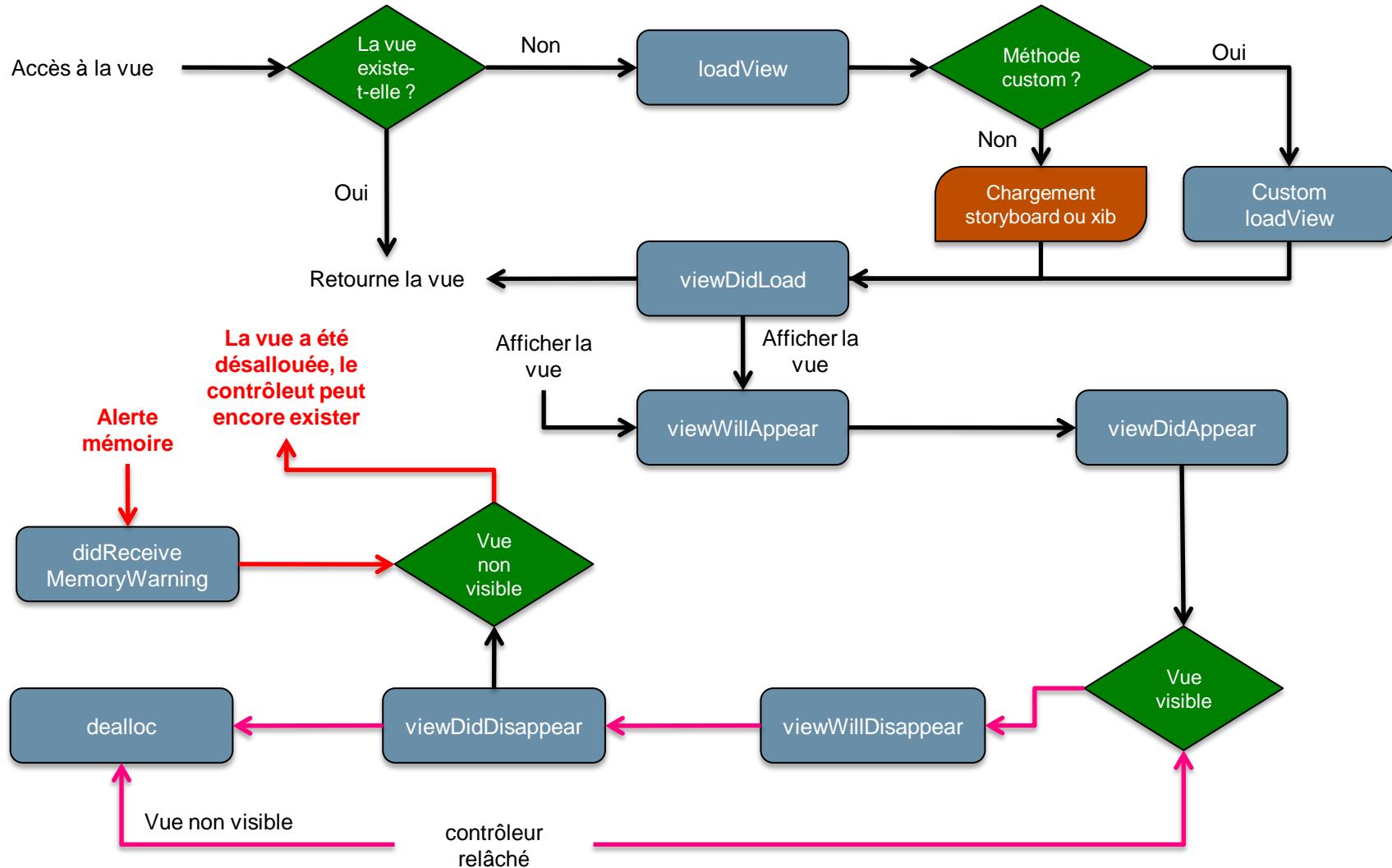
# UIKit

## les contrôleurs - présentation - cycles de vie



# UIKit

## les contrôleurs - présentation - cycles de vie



# UIKit

## les contrôleurs - principales classes

- `UINavigationController` : pour gérer une pile de contrôleurs
- `UITabBarController` : gère une collection de contrôleurs sous forme d'onglets
- `UISplitViewController` : gère 2 contrôleurs et divise l'écran en 2 (sur iPad seulement)
- `UIPopoverController` : affiche un contrôleur temporaire par-dessus le contrôleur actuel (iPad seulement)
- `UICollectionViewController` : contrôleur qui gère une `UICollectionView`
- `UISearchDisplayController` : contrôleur qui gère un champ de recherche

# UIKit

## les contrôleurs - navigation

- gère une **pile de contrôleurs**
- pour ajouter un contrôleur, on « push », celui-ci est alors retenu
- pour revenir, on « pop », le contrôleur est alors relâché

### Exemple

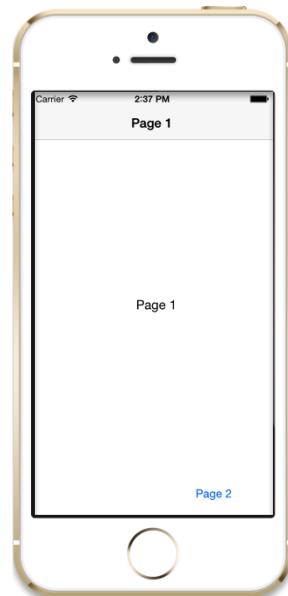
```
UIViewController *ctrl1 = [UIViewController new];
UINavigationController *navCtrl = [[UINavigationController
    alloc] initWithRootViewController:ctrl1];
```

# UIKit

## les contrôleurs - navigation - push

### Exemple

```
UIViewController *ctrl2 = [UIViewController new];
// ajout d'un nouveau contrôleur à la navigation
[navCtrl pushViewController:ctrl2 animated:YES];
```

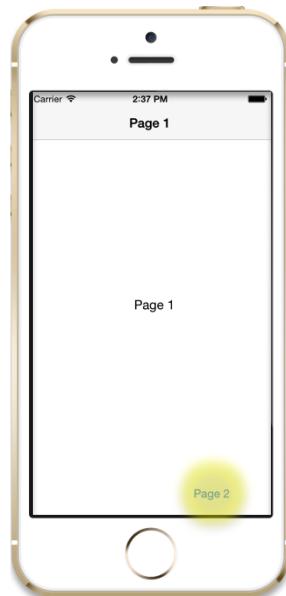


# UIKit

## les contrôleurs - navigation - push

### Exemple

```
UIViewController *ctrl2 = [UIViewController new];
// ajout d'un nouveau contrôleur à la navigation
[navCtrl pushViewController:ctrl2 animated:YES];
```

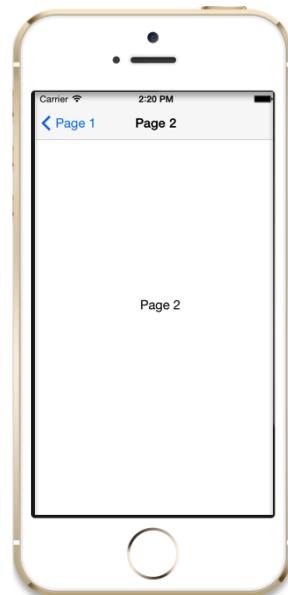


# UIKit

## les contrôleurs - navigation - push

### Exemple

```
UIViewController *ctrl2 = [UIViewController new];
// ajout d'un nouveau contrôleur à la navigation
[navCtrl pushViewController:ctrl2 animated:YES];
```

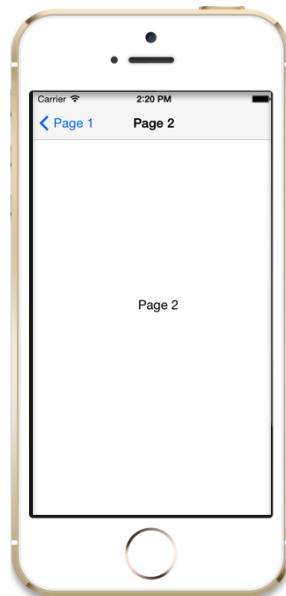


# UIKit

## les contrôleurs - navigation - pop

- par le bouton « back »
- ou par la commande :

```
[navCtrl popViewControllerAnimated:YES];
```



# UIKit

## les contrôleurs - navigation - pop

- par le bouton « back »
- ou par la commande :

```
[navCtrl popViewControllerAnimated:YES];
```

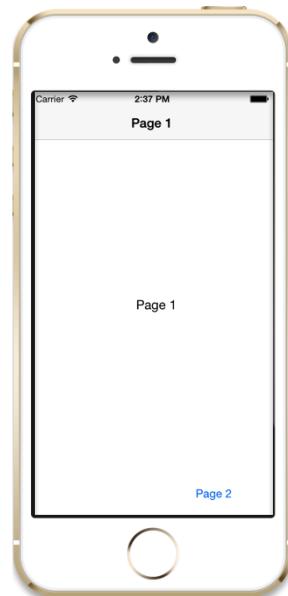


# UIKit

## les contrôleurs - navigation - pop

- par le bouton « back »
- ou par la commande :

```
[navCtrl popViewControllerAnimated:YES];
```



Le contrôleur  
dépilé  
est détruit

# UIKit

## les contrôleurs - tabbar - présentation

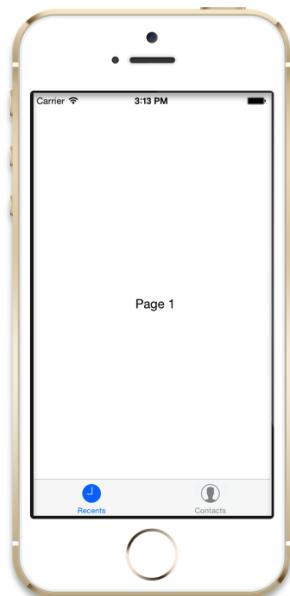
- gère une **collection de contrôleurs**
- équivalent à une liste d'onglets, chaque contrôleur étant accessible via un bouton dans la tab bar
- chaque contrôleur dans la collection possède un item avec une image et/ou un texte
- contrôleur principal d'une application, n'est pas censé être placé en sous-contrôleur (sauf dans un `UISplitViewController`)

# UIKit

## les contrôleurs - tabbar - création

### Exemple

```
UIViewController *ctrl1 = [UIViewController new];
ctrl1.tabBarItem = [[UITabBarItem alloc] initWithTitle:@"item1"
image:[UIImage imageNamed:@"image"] tag:1];
/* ... création d'autres contrôleurs ... */
UITabBarController *tabBarCtrl= [UITabBarController new];
tabBarCtrl.viewControllers = @[ctrl1, ctrl2, ..., ctrlN];
```

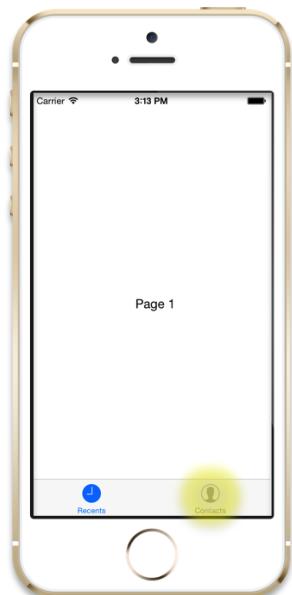


# UIKit

## les contrôleurs - tabbar - création

### Exemple

```
UIViewController *ctrl1 = [UIViewController new];
ctrl1.tabBarItem = [[UITabBarItem alloc] initWithTitle:@"item1"
image:[UIImage imageNamed:@"image"] tag:1];
/* ... création d'autres contrôleurs ... */
UITabBarController *tabBarCtrl= [UITabBarController new];
tabBarCtrl.viewControllers = @[ctrl1, ctrl2, ..., ctrlN];
```

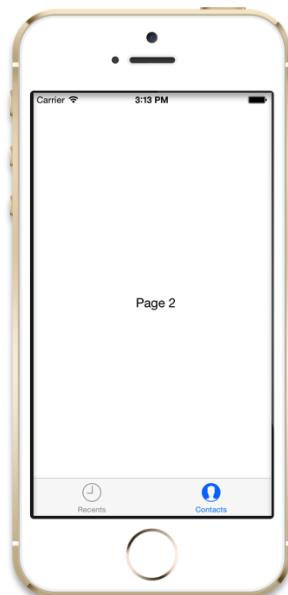


# UIKit

## les contrôleurs - tabbar - création

### Exemple

```
UIViewController *ctrl1 = [UIViewController new];
ctrl1.tabBarItem = [[UITabBarItem alloc] initWithTitle:@"item1"
image:[UIImage imageNamed:@"image"] tag:1];
/* ... création d'autres contrôleurs ... */
UITabBarController *tabBarCtrl= [UITabBarController new];
tabBarCtrl.viewControllers = @[ctrl1, ctrl2, ..., ctrlN];
```



# sommaire

partie 1 introduction

partie 2 Xcode

partie 3 UIKit

partie 4 **Interface Builder**

partie 5 Core Data

partie 6 autres frameworks

partie 7 ressources

partie 8 licences, certificats et profils d'approvisionnement

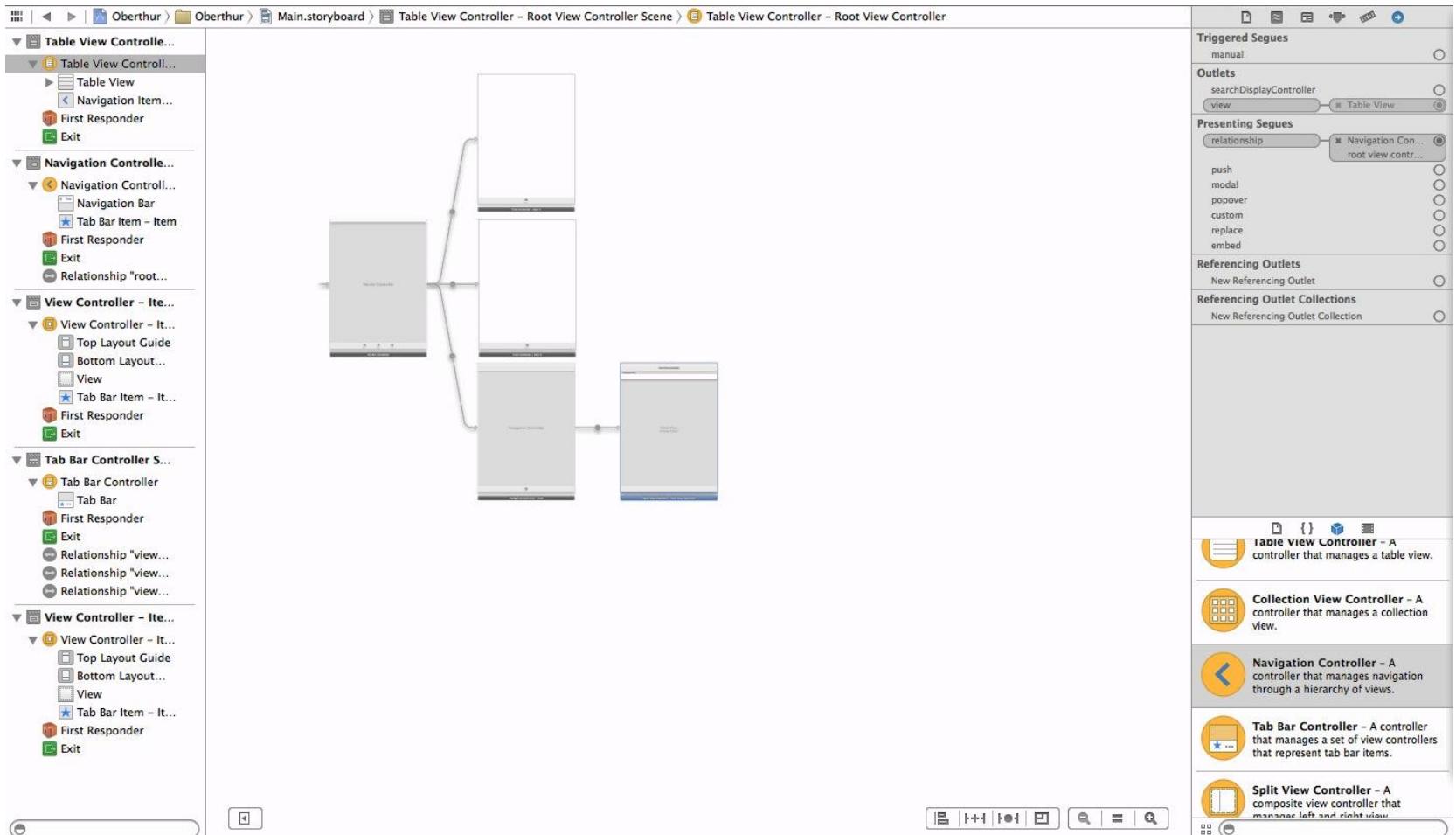
# Interface Builder

## présentation

- permet l'édition de vues via WYSIWYG :
  - taille
  - position
  - ancrage
  - ...

# Interface Builder

## présentation



# Interface Builder

## présentation

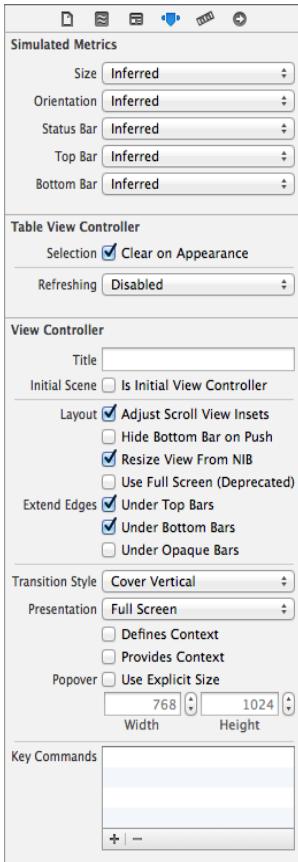


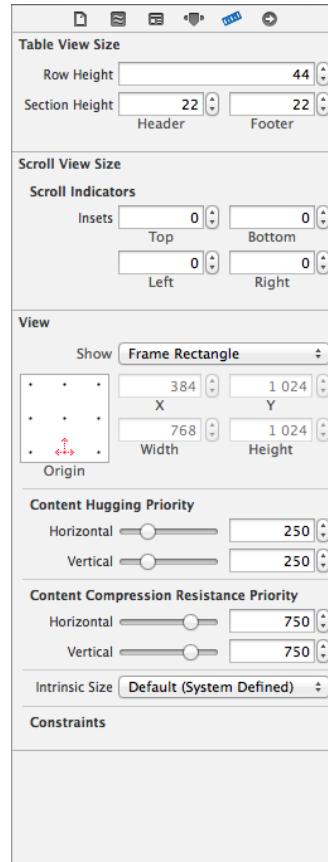
Table View Controller  
Selection  Clear on Appearance  
Refreshing

View Controller  
Title   
Initial Scene  Is Initial View Controller  
Layout  Adjust Scroll View Insets  
 Hide Bottom Bar on Push  
 Resize View From NIB  
 Use Full Screen (Deprecated)  
Extend Edges  Under Top Bars  
 Under Bottom Bars  
 Under Opaque Bars

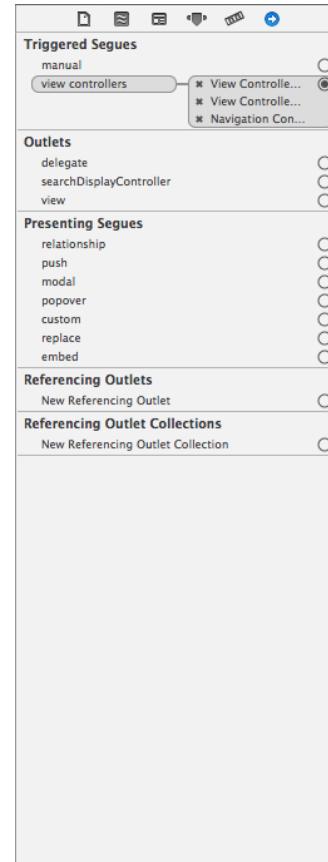
Transition Style   
Presentation   
 Defines Context  
 Provides Context  
Popover  Use Explicit Size  
   
Width Height

Key Commands  
  
+ | -

régLAGES de  
l'élément



position, taille et  
contraintes



branchements  
(outlets)

# Interface Builder

## Storyboard

- permet d'afficher tous les écrans de l'application ainsi que les transitions entre eux
- une transition entre deux écrans est un **segue** qui peut être de type :
  - push : à utiliser dans un contrôleur de navigation pour pousser l'écran suivant
  - modal : à utiliser pour afficher un écran par-dessus l'écran courant
  - custom : pour implémenter soi-même la transition
  - popover : pour afficher un écran dans un popover
  - replace : utilisé en général dans un split pour remplacer la partie droite après sélection d'un élément dans la partie gauche
  - embed : pour intégrer un sous-contrôleur (utilisation de Container View)

# Interface Builder

## storyboard - custom segue

- un segue doit avoir un **identifiant unique** pour permettre sa reconnaissance
- le déclenchement d'un segue se fait de la manière suivante :
  1. création du contrôleur final
  2. création du segue
  3. appel de la méthode `prepareForSegue:sender:` dans le contrôleur source
  4. appel de la méthode `perform` du segue

# Interface Builder

## storyboard - custom segue

- l'appel de la méthode `prepareForSegue:sender:` permet de configurer le contrôleur final, par exemple :
  - fournir l'objet lié à la cellule sélectionnée
  - indiquer au contrôleur final que le contrôleur est son délégué
- la méthode `perform` permet de customiser l'apparition du contrôleur final en ajoutant des animations

# Interface Builder

## storyboard - custom segue

### Exemple

```
@implementation MyCustomSegue

- (void) perform
{
    // fait apparaître le contrôleur final par fade-in
    [self.sourceViewController
addChildViewController:self.destinationViewController];
    [[self.sourceViewController view]
addSubview:[self.destinationViewController view]];
    [self.destinationViewController view].alpha = 0;
    [UIView animateWithDuration:1 animations:^{
        [self.destinationViewController view].alpha = 1;
    }];
}

@end
```

# Interface Builder

## Storyboard - limitations

- 1 seul fichier pour tous les écrans → risque de conflit si on travaille à plusieurs
  - Apple préconise de faire plusieurs storyboards dans ce cas
  - Il est également possible d'utiliser des fichiers xib
- un storyboard complexe peut ralentir Xcode (cela dépend de la machine)

# Interface Builder

## utilisation des branchements

- les outlets permettent de relier une propriété d'une classe à un élément
- les actions permettent de relier une action sur un élément à une méthode d'une classe (la méthode doit retourner le type `IBAction`)

### Exemple

```
@property (nonatomic, retain) IBOutlet UILabel *textLabel;  
-(IBAction) onTapButton:(UIButton*)button;
```

# sommaire

**partie 1** introduction

**partie 2** Xcode

**partie 3** UIKit

**partie 4** Interface Builder

**partie 5** **Core Data**

**partie 6** autres frameworks

**partie 7** ressources

**partie 8** licences, certificats et profils d'approvisionnement

# Core Data

c'est quoi ?

- persistance de données et mapping relationnel objet
- enregistre dans un fichier SQLite (XML et binaire disponibles sur OS X seulement)
- simple à mettre en place
- conçu pour les applications desktop (et mobile pour iOS), mais pas sur les serveurs

# Core Data

## éléments principaux

- persistent store coordinator : abstraction entre l'API et le format de fichier dans lequel sont stockées les données (SQLite, XML, binaire ou mémoire)
- managed object model : représente le modèle de données
- managed object context : représente le contenu du modèle de données (les objets qui le peuplent)
- un modèle est composé d'entités qui possèdent :
  - des attributs
  - des relations (pour faire un lien avec d'autres entités)
  - des propriétés récupérées (fetched properties), elles sont peu utilisées

# Core Data

## éléments principaux

- une relation est **bidirectionnelle** et possède une **multiplicité**
  - 1 : un objet d'une entité est lié à un seul objet maximum d'une autre
  - n : un objet d'une entité est lié à plusieurs objets d'une autre
- relations et attributs peuvent être définis obligatoires ou non
- à l'exécution :
  - une entité devient une classe (qui hérite de `NSManagedObject`)
  - un attribut devient une propriété
  - une relation devient :
    - une propriété si elle est de multiplicité 1
    - des méthodes si elle est de multiplicité n

# Core Data

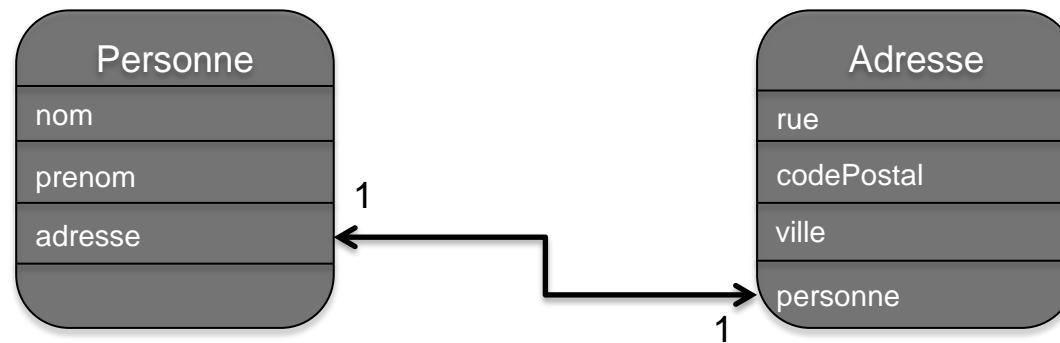
## éléments principaux

- une relation possède une **règle de suppression** avec les valeurs possibles :
  - Nullify : la suppression d'un objet entraîne la nullité de la relation inverse
  - Cascade : la suppression de l'objet entraîne également la suppression de ou des objets liés
  - Deny : la suppression de l'objet est interdite si la relation inverse n'est pas nulle
  - No Action : la suppression de l'objet n'entraîne aucune action

# Core Data

éléments principaux - nullify

- la règle de suppression de la relation **personne** de Adresse vers Personne est de type **nullify**

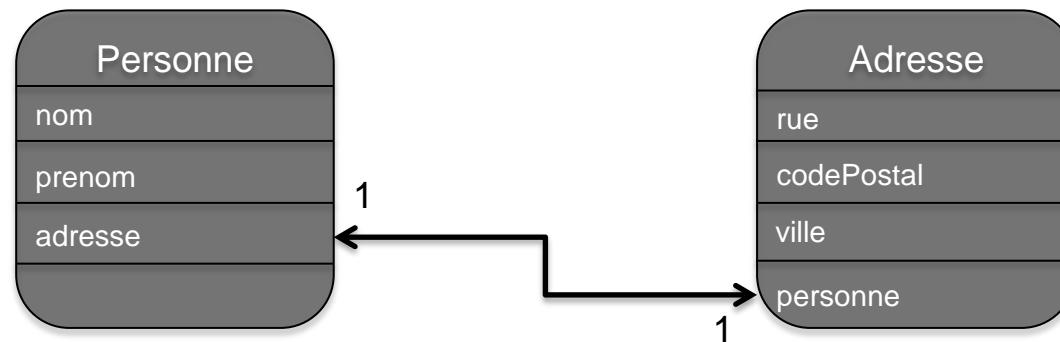


# Core Data

éléments principaux - nullify

- la règle de suppression de la relation **personne** de Adresse vers Personne est de type **nullify**

## Suppression de l'adresse

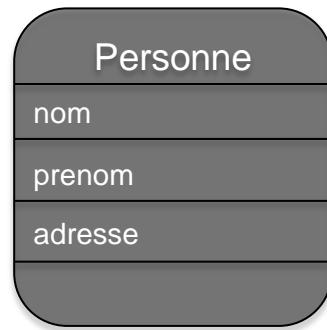


# Core Data

éléments principaux - nullify

- la règle de suppression de la relation **personne** de Adresse vers Personne est de type **nullify**

Suppression de l'adresse

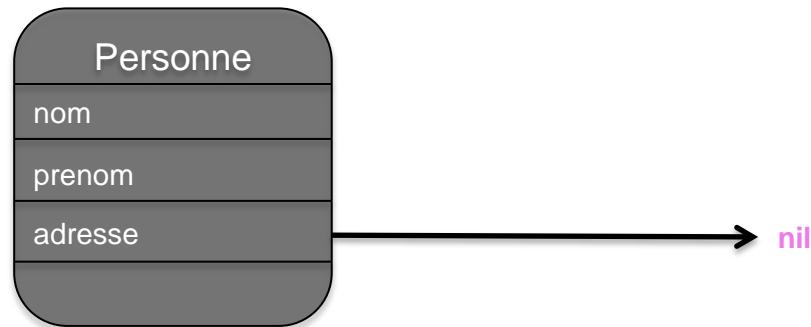


# Core Data

éléments principaux - nullify

- la règle de suppression de la relation **personne** de Adresse vers Personne est de type **nullify**

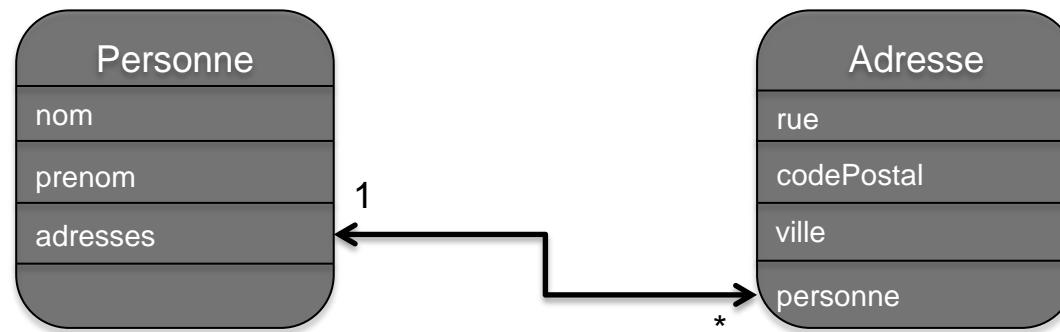
## Suppression de l'adresse



# Core Data

éléments principaux - cascade

- la règle de suppression de la relation **adresse** de Personne vers Adresse est de type **cascade**

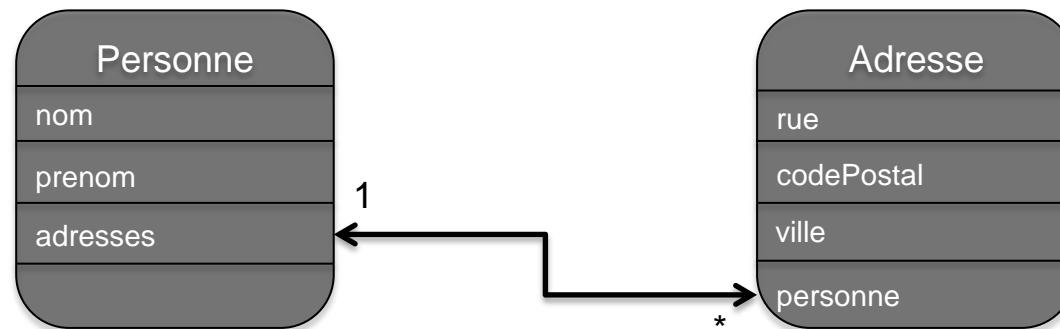


# Core Data

éléments principaux - cascade

- la règle de suppression de la relation **adresse** de Personne vers Adresse est de type **cascade**

Suppression de la personne



# Core Data

éléments principaux - cascade

- la règle de suppression de la relation **adresse** de Personne vers Adresse est de type **cascade**

Suppression de la personne



# Core Data

éléments principaux - cascade

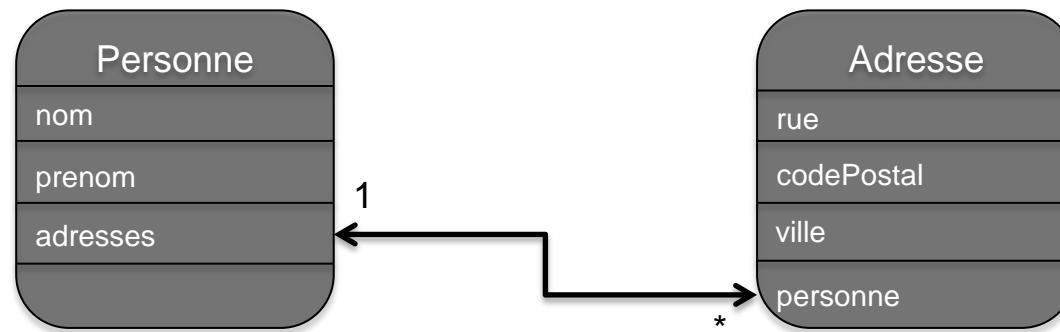
- la règle de suppression de la relation **adresse** de Personne vers Adresse est de type **cascade**

Suppression de la personne

# Core Data

éléments principaux - deny

- la règle de suppression de la relation **adresses** de Personne vers Adresse est de type **deny**

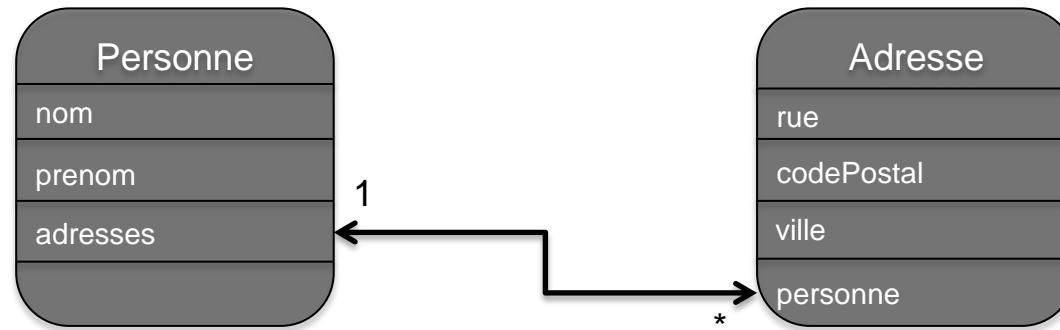


# Core Data

éléments principaux - deny

- la règle de suppression de la relation **adresses** de Personne vers Adresse est de type **deny**

Suppression d'une personne qui possèdent 2 adresses



# Core Data

éléments principaux - deny

- la règle de suppression de la relation **adresses** de Personne vers Adresse est de type **deny**

Suppression d'une personne qui possèdent 2 adresses

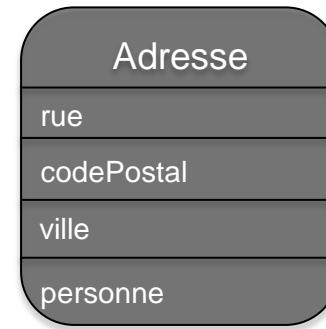


# Core Data

éléments principaux - deny

- la règle de suppression de la relation **adresses** de Personne vers Adresse est de type **deny**

Suppression d'une personne qui possèdent 2 adresses

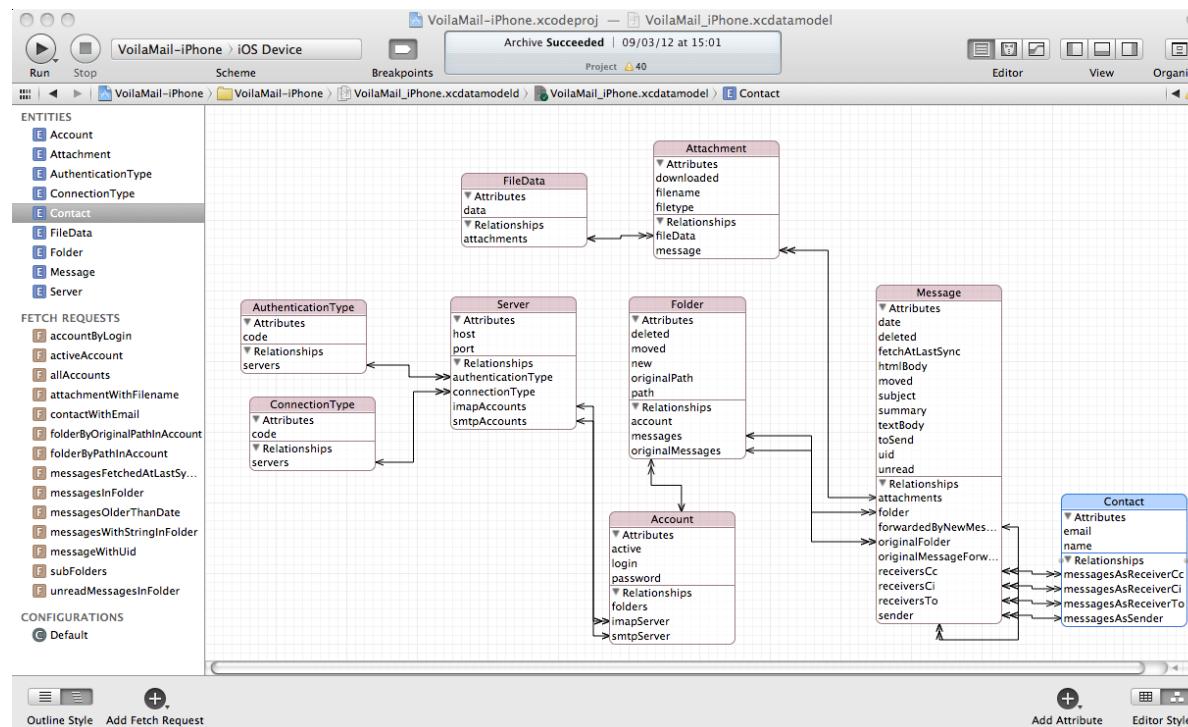


Erreur lors de la sauvegarde car les adresses n'ont pas été retirées de la relation au préalable

# Core Data

## modélisation

- via l'outil disponible dans Xcode qui permet :
  - l'édition du modèle
  - la génération des classes issues des entités du modèle



# Core Data

## modélisation

The screenshot shows the Xcode Core Data Entity Inspector for the 'Account' entity. The left sidebar lists entities and fetch requests. The main area is divided into three sections: Attributes, Relationships, and Fetched Properties.

**Attributes:**

Attribute	Type
B active	Boolean
S login	String
S password	String

**Relationships:**

Relationship	Destination	Inverse
M folders	Folder	account
O imapServer	Server	imapAccounts
O smtpServer	Server	smtpAccounts

**Fetched Properties:**

Fetched Property	Predicate

# Core Data

## requêtage

- pour récupérer des objets dans le modèle, il faut utiliser une requête (`NSFetchRequest`). Pour cela, deux possibilités :
  - en code
  - via l'éditeur de Xcode on crée la requête, et on l'appelle en code

# Core Data

## requêtage - en code

- une requête :
  - doit être faite sur une entité
  - peut filtrer des objets de cette entité (à l'aide d'un prédictat)
  - peut trier le résultat
- la classe permettant de faire une requête est `NSFetchRequest`
- une requête s'effectue obligatoirement dans un **managed object context**
- on prend pour exemple, une liste de messages. Un message a un titre, un contenu et une date. On va faire une requête pour récupérer les titres dont le contenu commence par « Le » et les trier par date du plus récent au plus ancien

# Core Data

## requêtage - en code

### Exemple

```
NSManagedObjectContext *context = ...;
NSError *error = nil;
NSFetchRequest *request = [[NSFetchRequest alloc]
    initWithEntity:@"Message"];
request.predicate = [NSPredicate predicateWithFormat:@"titre
    BEGINSWITH[cd] 'Le' "];
request.sortDescriptors = @[[NSSortDescriptor
    sortDescriptorWithKey:@"date" ascending:NO]];
// exécution de la requête
NSArray *result = [context executeFetchRequest:request
    error:&error];
...
```

# Core Data

## requêtage - en code

### Exemple

```
NSManagedObjectContext *context = ...;
NSError *error = nil;
NSFetchRequest *request = [[NSFetchRequest alloc]
    initWithEntity:@"Message"];
request.predicate = [NSPredicate predicateWithFormat:@"titre
    BEGINSWITH[cd] 'Le' "];
request.sortDescriptors = @[[NSSortDescriptor
    sortDescriptorWithKey:@"date" ascending:NO]];
// exécution de la requête
NSArray *result = [context executeFetchRequest:request
    error:&error];
...
```

# Core Data

## requêtage - en code

### Exemple

```
NSManagedObjectContext *context = ...;
NSError *error = nil;
NSFetchRequest *request = [[NSFetchRequest alloc]
    initWithEntity:@"Message"];
request.predicate = [NSPredicate predicateWithFormat:@"titre
    BEGINSWITH[cd] 'Le' "];
request.sortDescriptors = @[[NSSortDescriptor
    sortDescriptorWithKey:@"date" ascending:NO]]];
// exécution de la requête
NSArray *result = [context executeFetchRequest:request
    error:&error];
...
```

# Core Data

## requêtage - en code

### Exemple

```
NSManagedObjectContext *context = ...;
NSError *error = nil;
NSFetchRequest *request = [[NSFetchRequest alloc]
    initWithEntity:@"Message"];
request.predicate = [NSPredicate predicateWithFormat:@"titre
    BEGINSWITH[cd] 'Le' "];
request.sortDescriptors = @[[NSSortDescriptor
    sortDescriptorWithKey:@"date" ascending:NO]];
// exécution de la requête
NSArray *result = [context executeFetchRequest:request
    error:&error];
...
```

# Core Data

## requêtage - en code

### Exemple

```
NSManagedObjectContext *context = ...;
NSError *error = nil;
NSFetchRequest *request = [[NSFetchRequest alloc]
    initWithEntity:@"Message"];
request.predicate = [NSPredicate predicateWithFormat:@"titre
    BEGINSWITH[cd] 'Le' "];
request.sortDescriptors = @[[NSSortDescriptor
    sortDescriptorWithKey:@"date" ascending:NO]];
// exécution de la requête
NSArray *result = [context executeFetchRequest:request
    error:&error];
...
```

# Core Data

## requêtage - en code

### Exemple

```
NSManagedObjectContext *context = ...;
NSError *error = nil;
NSFetchRequest *request = [[NSFetchRequest alloc]
    initWithEntity:@"Message"];
request.predicate = [NSPredicate predicateWithFormat:@"titre
    BEGINSWITH[cd] 'Le' "];
request.sortDescriptors = @[[NSSortDescriptor
    sortDescriptorWithKey:@"date" ascending:NO]];
// exécution de la requête
NSArray *result = [context executeFetchRequest:request
    error:&error];
...

```



# Core Data

## requêtage - en code

### Exemple

```
NSManagedObjectContext *context = ...;
NSError *error = nil;
NSFetchRequest *request = [[NSFetchRequest alloc]
    initWithEntity:@"Message"];
request.predicate = [NSPredicate predicateWithFormat:@"titre
    BEGINSWITH[cd] 'Le' "];
request.sortDescriptors = @[[NSSortDescriptor
    sortDescriptorWithKey:@"date" ascending:NO]];
// exécution de la requête
NSArray *result = [context executeFetchRequest:request
    error:&error];
...
```

# Core Data

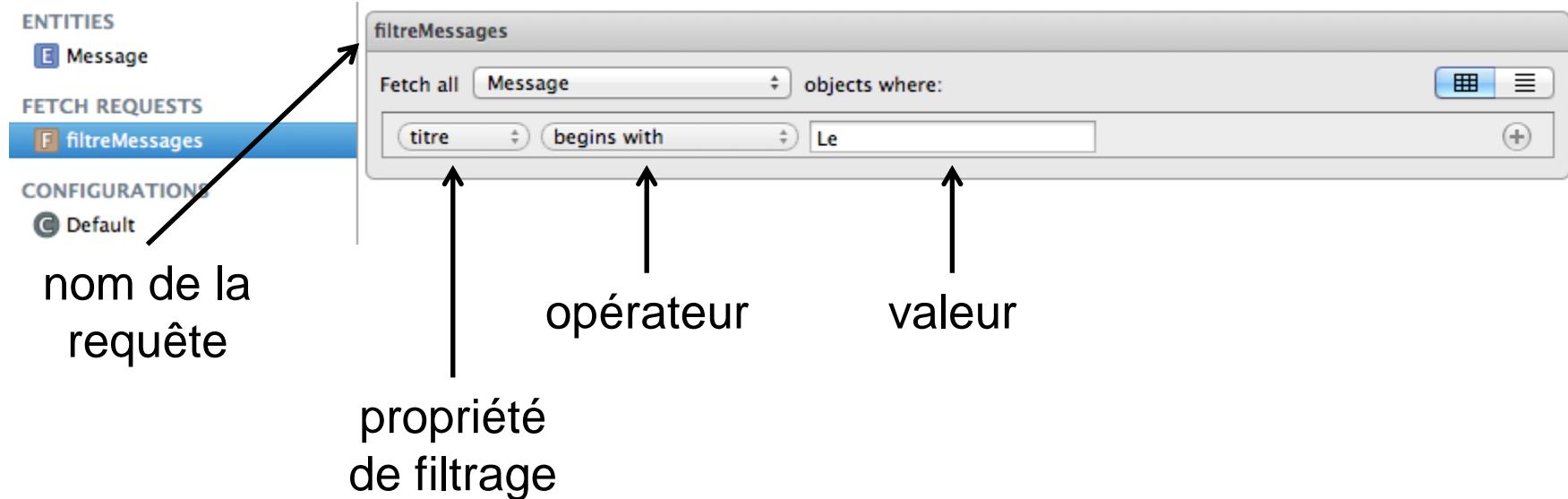
## requêtage - en code

### Exemple

```
NSManagedObjectContext *context = ...;
NSError *error = nil;
NSFetchRequest *request = [[NSFetchRequest alloc]
    initWithEntity:@"Message"];
request.predicate = [NSPredicate predicateWithFormat:@"titre
    BEGINSWITH[cd] 'Le' "];
request.sortDescriptors = @[[NSSortDescriptor
    sortDescriptorWithKey:@"date" ascending:NO]];
// exécution de la requête
NSArray *result = [context executeFetchRequest:request
    error:&error];
...
...
```

# Core Data

requêtage - via l'éditeur



- le tri n'est pas possible directement , il faut le faire en code

# Core Data

requêtage - via l'éditeur

## Exemple

```
NSManagedObjectContext *context = ...;
NSManagedObjectModel *model = ...;
NSError *error = nil;
NSFetchRequest *request = [model
    fetchRequestTemplateForName:@"filtreMessages"];
request.sortDescriptors = @[[NSSortDescriptor
    sortDescriptorWithKey:@"date" ascending:NO]];
// exécution de la requête
NSArray *result = [context executeFetchRequest:request
    error:&error];
...
```

# Core Data

## sauvegarde

### Exemple

```
NSManagedObjectContext *context = ...;
NSError *error = nil;
if (![context save:&error])
{
    NSLog(@"erreur lors de la suppression", [error
        localizedDescription]);
}
```

# Core Data

## insertion

- l'insertion d'un objet se fait en utilisant la classe `NSEntityDescription`
- Il faut penser à sauvegarder l'objet ajouté

### Exemple

```
NSManagedObjectContext *context = ...;
NSManagedObjectModel *model = ...;
NSError *error = nil;
Message *message = [NSEntityDescription
    insertNewObjectForEntityForName:@"Message"
    inManagedObjectContext:context]];
// exécution de la requête
message.titre = ...;
...
```

# Core Data

## suppression

- la suppression d'un objet se fait au niveau du contexte
-  on ne peut pas supprimer un objet qui ne fait pas partie du contexte  
(par exemple en environnement multithreadé)

### Exemple

```
NSManagedObjectContext *context = ...;
Message *message = ....;
// suppression
[context deleteObject:message];
```

# Core Data

points importants

- un contexte ne doit être **utilisé que dans un seul thread**, chaque thread devant donc gérer son propre contexte
- un `NSManagedObject` ne doit **pas être transmis d'un thread à un autre**, ni manipulé par un autre contexte que celui qui l'a créé ou récupéré via une requête
- pour transmettre un `NSManagedObject` dans un autre thread ou contexte, on utilise son `objectID` qui est un **identifiant unique** géré par Core Data
- il est possible de faire du « merge » entre deux contextes afin d'éviter les conflits

# Core Data

## merge - exemple

- une application possède un contexte principal que l'on nomme **mainCtx**
- l'application effectue une requête en asynchrone qui insère des objets dans un thread secondaire, donc avec un autre contexte nommé **webCtx**
- à la fin de la requête, il faut merger **webCtx** avec **mainCtx**, et donc insérer les nouveaux objets dans **mainCtx**
- pour cela, on utilise la notification `NSManagedObjectContextDidSaveNotification` envoyée automatiquement par **webCtx** lors de la sauvegarde

# Core Data

## merge - exemple

### Exemple

```
// à faire avant le lancement de la requête et après
// la création de webCtx
[[NSNotificationCenter defaultCenter] addObserver:self
selector:@selector(webCtxDidSave:)
name: NSManagedObjectContextDidSaveNotification object:webCtx];
```

# Core Data

## merge - exemple

### Exemple

```
// méthode recevant la notification
-(void) webCtxDidSave: (NSNotification*) notification
{
    // appel de la méthode
    mergeChangesFromContextDidSaveNotification: dans le thread
    principal
    [mainCtx
    performSelectorOnMainThread:@selector(mergeChangesFromContextD
    idSaveNotification:) withObject:notification
    waitUntilDone:YES];
}
```

# sommaire

**partie 1** introduction

**partie 2** Xcode

**partie 3** UIKit

**partie 4** Interface Builder

**partie 5** Core Data

**partie 6 autres frameworks**

**partie 7** ressources

**partie 8** licences, certificats et profils d'approvisionnement

# autres frameworks

## Foundation - contenu

- objets et structures de base :
  - NSArray, NSDictionary, NSSet
  - NSNumber, NSString, NSData, NSDate, NSTimer
- les structures et objets de base ne sont pas forcément modifiables par défaut, il faut utiliser des versions « mutables » :  
NSMutableArray, NSMutableDictionary, NSMutableString,  
NSMutableData, NSMutableSet
- le préfixe « NS » provient de NeXTStep

# autres frameworks

## Foundation - contenu

- `NSFileManager` : gestionnaire de fichiers et dossiers (création, suppression déplacement, copie)
- `NSBundle` : permet d'accéder aux ressources de l'application
- `NSURL*` : classes permettant d'effectuer des requêtes
- `NSXMLParser` : pour parser du XML
- `NSThread`, `NSOperation*` : pour du multi-threading

# autres frameworks

## Foundation - contenu

- presque tous les objets de Foundation ont une correspondance avec le framework CoreFoundation qui définit des structures équivalentes en C

### Exemple

```
NSString *str1 = @"blabla";
CFStringRef str2 = (CFStringRef)str1;
// str2 contient @"blabla" peut être manipulée par les fonctions
de CFString
CFStringRef str3 = CFStringCreateWithCString(NULL, "ma chaine",
    kCFStringEncodingUTF8);
NSString *str4 = (NSString*)str3;
// str4 contient @"ma chaine" peut être manipulée par les méthodes
de NSString
```

## autres frameworks

### CoreGraphics

- permet le dessin de formes (rectangles, ellipses, arcs, lignes) dans des contextes graphiques
- permet de dessiner les pages d'un document pdf
- permet d'examiner le contenu des méta-données d'un document pdf (sommaire, orientation, ...)



l'axe Y dans CoreGraphics est inversé par rapport aux vues



## autres frameworks

autres

- Map Kit : géolocalisation et résolution d'adresse à partir de coordonnées et inversement
- Media Player : lecture audio et vidéo
- Quick Look : prévisualisation de documents (office, pdf, image, rtf, ...)
- OpenGL ES : pour faire de la 3D
- Quartz Core : pour gérer des animations et des filtres
- Core Data : gestion de données
- PassKit : gestion de passes (cartes de fidélité, billets d'avion, ...)
- ...

# sommaire

**partie 1** introduction

**partie 2** Xcode

**partie 3** UIKit

**partie 4** Interface Builder

**partie 5** Core Data

**partie 6** autres frameworks

**partie 7 ressources**

**partie 8** licences, certificats et profils d'approvisionnement

# ressources

## fichiers de chaînes

- fichiers dont l'extension est .strings
- fichiers localisables
- association d'une clé avec une valeur
- format textuel ou "json"

### Exemple format textuel

```
"OneKey" = "Un exemple de label";  
"MainController.Button.Title" = "Titre du bouton";
```

### Exemple format "json"

```
{  
    OneKey = "Un exemple de label";  
    "MainController.Button.Title" = "Titre du bouton";  
}
```

# ressources

## fichiers de chaînes - utilisation

- utilisation de la fonction `NSLocalizedStringFromTable`
- paramètres :
  - clé dans le fichier `.strings`
  - nom du fichier `.strings` dans extension
  - un commentaire qui sert uniquement pour la compréhension du traducteur
- autre fonction `NSLocalizedString` qui va directement rechercher dans un fichier nommé `Localizable.strings`

### Exemple

```
//utilisation d'un fichier nommé Localizable.strings
NSLocalizedString (@"OneKey", @"Commentaire pour le traducteur");
//utilisation d'un fichier nommé MainController.strings
NSLocalizedStringFromTable (@"Button.Title", @"MainController",
 @"Le premier bouton");
```

# ressources

## plist

- fichiers xml permettant de déclarer des objets basiques Cocoa :
  - dictionnaires
  - Tableaux
  - chaînes
  - nombres
  - booléens
  - dates
  - données brutes
- parsing natif par les classes `NSArray` et `NSDictionary`

# ressources

## plist

### Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Test</key>
    <string>Test</string>
    <key>Une Date</key>
    <date>2013-12-19T16:52:01Z</date>
    <key>Un nombre</key>
    <real>23.56</real>
    <key>Tableau</key>
    <array>
        <data></data>
        <true/>
    </array>
</dict>
</plist>
```

# ressources

## réglages

- dossier (bundle) contenant un fichier plist et un fichier de chaînes
- le fichier plist contient une liste de réglages disponibles pour l'application
- types de réglage :
  - groupe
  - libellé
  - champ de texte
  - switch (booléen)
  - régllette (slider)
  - sélection parmi une liste de valeurs
- modifiables via l'application **Réglages** de l'appareil

# ressources

## réglages - exemple de fichier plist

### Exemple

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>PreferenceSpecifiers</key>
    <array>
        <dict>
            <key>Title</key>
            <string>Group</string>
            <key>Type</key>
            <string>PSGroupSpecifier</string>
        </dict>
        <dict>
            <key>DefaultValue</key>
            <true/>
            <key>Key</key>
            <string>enabled_preference</string>
            <key>Title</key>
            <string>Enabled</string>
            <key>Type</key>
            <string>PSToggleSwitchSpecifier</string>
        </dict>
    </array>
    <key>StringsTable</key>
    <string>Root</string>
</dict>
</plist>
```

## ressources

### réglages - fichier de chaînes

- permet d'associer des titres localisés aux différents réglages déclarés
- il faut donc utiliser des identifiants de ces chaînes dans le fichier plist des réglages

#### Exemple format textuel

```
"Group" = "Groupe" ;  
"Enabled" = "Activé" ;
```

# ressources

réglages - accès depuis le code

- l'accès et la modification sont possibles depuis l'application
- utilisation de la classe `NSUserDefaults`
- possibilité pour l'application d'être informée si l'utilisateur modifie les réglages

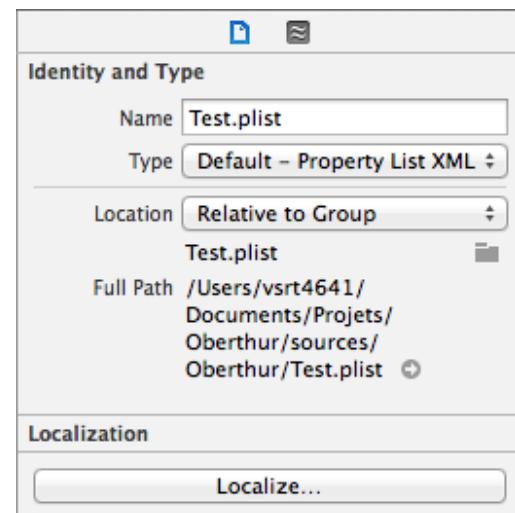


la valeur par défaut n'est pas récupérée automatiquement à la première lecture

# ressources

## localisation

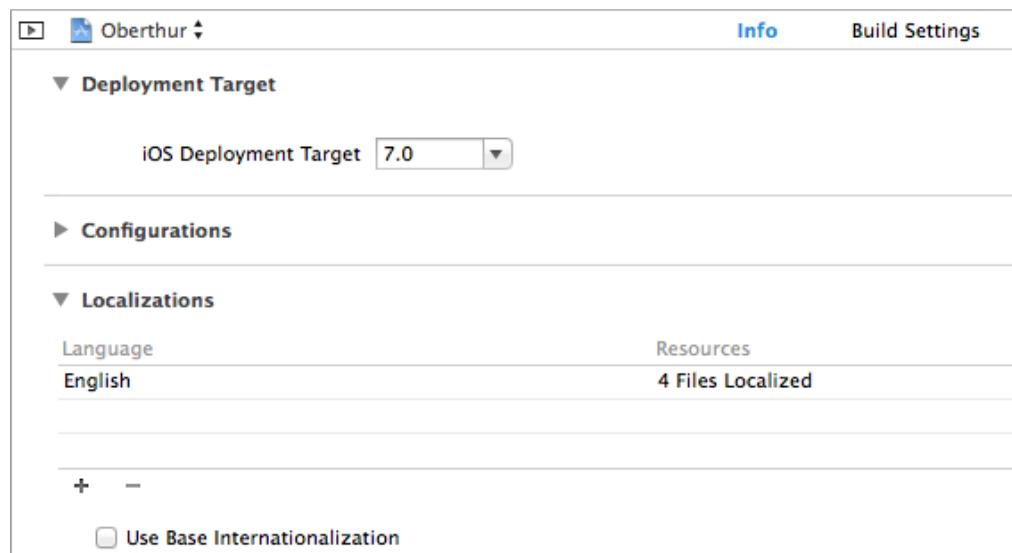
- chaque ressource peut être localisée
- les réglages sont localisés **automatiquement**
- les ressources localisées sont placées dans des dossiers dont l'extension **.lproj** et dont le nom est le code de la langue, par exemple :
  - en.lproj
  - fr.lproj
- pour localiser une ressource, il suffit d'utiliser le bouton



# ressources

## localisation

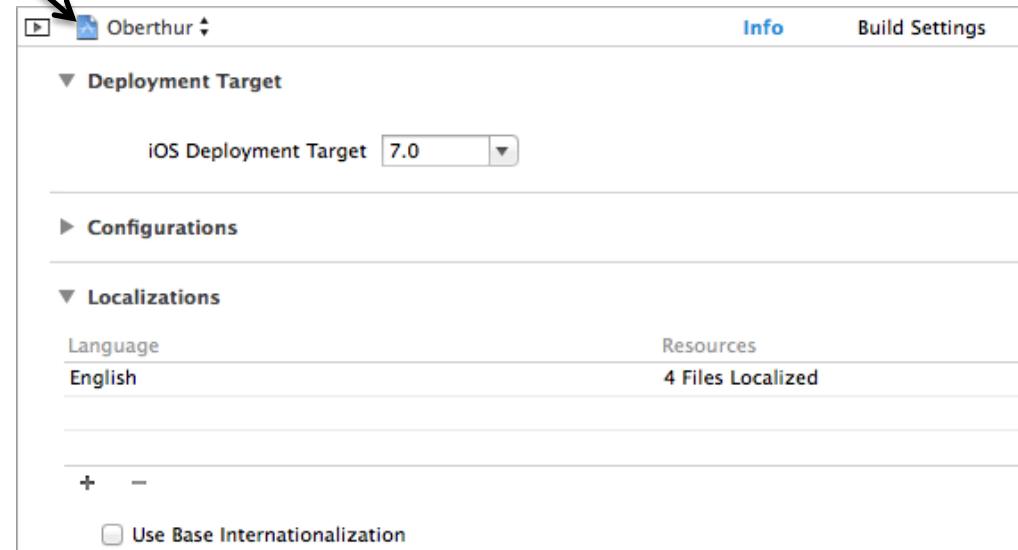
- pour ajouter une nouvelle langue, il faut aller dans les **réglements du projet** et ajouter une **localisation**



# ressources

## localisation

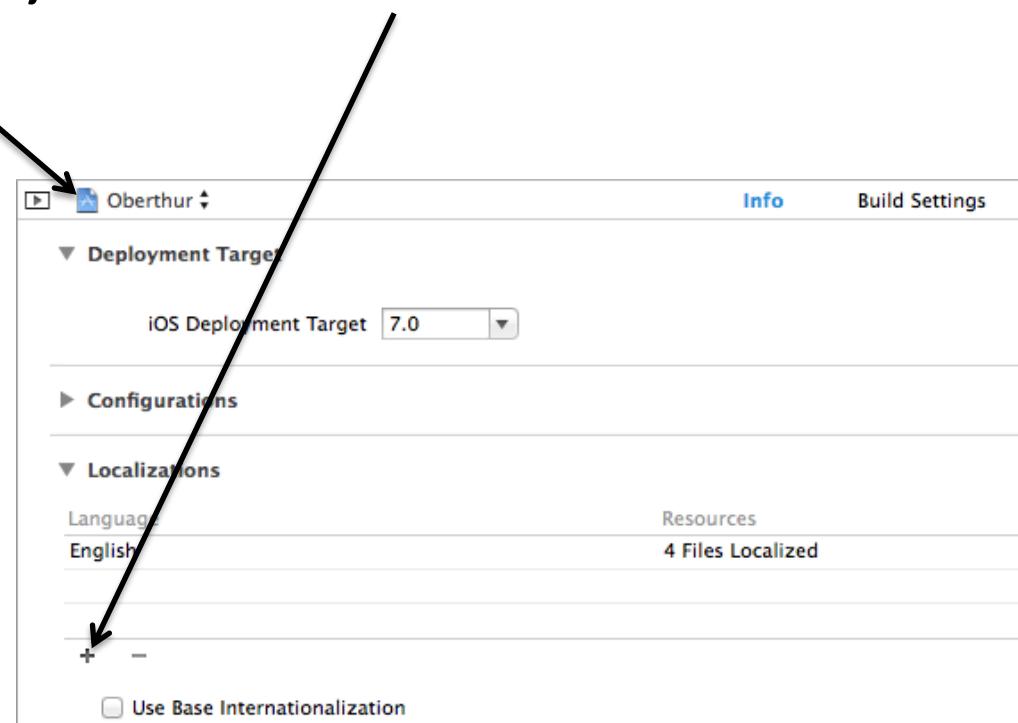
- pour ajouter une nouvelle langue, il faut aller dans les **régles du projet** et ajouter une **localisation**



# ressources

## localisation

- pour ajouter une nouvelle langue, il faut aller dans les **régles du projet** et ajouter une **localisation**



# sommaire

**partie 1** introduction

**partie 2** Xcode

**partie 3** UIKit

**partie 4** Interface Builder

**partie 5** Core Data

**partie 6** autres frameworks

**partie 7** ressources

**partie 8 licences, certificats et profils d'approvisionnement**

# licences, certificats et profils d'approvisionnement

## licences

- trois types de licence :
  - developer program (99\$/an)
  - developer enterprise program (299\$/an)
  - developer university program (gratuit)

# licences, certificats et profils d'approvisionnement

## licences - developer program

- développement
- test sur mobile avec déploiement :
  - développeur
  - ad hoc
- enregistrement de 100 appareils pour déploiement
- distribution sur App Store

# licences, certificats et profils d'approvisionnement

licences - developer enterprise program

- développement
- test sur mobile avec déploiement :
  - développeur
  - ad hoc
- enregistrement de 100 appareils pour déploiement
- distribution interne entreprise avec déploiement In House
- accès au support technique
- accès au forum de développeurs

# licences, certificats et profils d'approvisionnement

licences - developer university program

- création d'une équipe de maximum 200 personnes
- développement
- test sur mobile avec déploiement :
  - développeur
- partage d'applications au sein de la même équipe
  - par mail
  - par site privé

# licences, certificats et profils d'approvisionnement

## licences - rôles

- trois rôles existent :

- agent
- admin
- membre

# licences, certificats et profils d'approvisionnement

## licences - rôles

privilège	agent	admin	membre
Have legal responsibility for the team	✓	✗	✗
Be the primary contact with Apple	✓	✗	✗
View prerelease Apple content	✓	✓	✓
Enroll in additional developer programs and renew them	✓	✗	✗
Invite team admins and team members	✓	✓	✗
Request development certificates	✓	✓	✓
Approve team member requests for development certificates	✓	✓	✗
Request distribution certificates	✓	✓	✗
For Mac apps, request Developer ID certificates	✓	✗	✗
Add devices for development and testing	✓	✓	✗
Create App IDs and enable certain technologies and services	✓	✓	✗
Create development and distribution provisioning profiles	✓	✓	✗
Create SSL certificates for Apple Push Notification service	✓	✓	✗
Download development provisioning profiles	✓	✓	✓
Submit apps to the App Store or Mac App Store	✓	✗	✗
Sign app for In House Distribution	✓	✗	✗

# licences, certificats et profils d'approvisionnement

## certificats

- deux types :
  - développement
  - distribution
- est associé à une personne
- nécessaire pour signer une application

# licences, certificats et profils d'approvisionnement

## certificats - création

- création d'une demande de certificat avec l'application **Trousseau d'accès**
  - la demande est nominative
  - envoi de la demande pour création du certificat
  - téléchargement du certificat créé et installation sur la machine
- !** le certificat lie l'utilisateur **et** la machine, il peut donc être utile d'en garder une version exportée pour l'utiliser sur une autre machine, pour cela, il faut l'exporter au format **p12** à partir du **Trousseau d'accès**



# licences, certificats et profils d'approvisionnement

## App ID

- permet d'identifier les applications et les fonctionnalités disponibles
- possède un **Bundle ID** qui identifie la ou les applications
- deux types :
  - explicite : pour identifier une application unique, le Bundle ID est en général de la forme com.company.appName
  - wildcard : pour identifier un groupe d'applications, le Bundle ID se termine par le caractère \*, par exemple, com.company.\*

# licences, certificats et profils d'approvisionnement

## App ID

- certaines fonctionnalités nécessitent obligatoirement un App ID explicite :
  - Game Center
  - In-App Purchase
  - Push Notifications

# licences, certificats et profils d'approvisionnement

## profils d'approvisionnement

- associe un certificat, un App ID et éventuellement une liste d'appareils autorisés
- quatre types :
  - développement
  - distribution ad hoc
  - distribution App Store
  - distribution In House

# licences, certificats et profils d'approvisionnement

## profils d'approvisionnement - développement

- permet à un développeur d'installer sur un appareil iPad, iPhone ou iPod
- l'application est utilisable 3 mois
- l'application n'est installable que sur une liste définie d'appareils
- disponibles avec tous les types de licence

# licences, certificats et profils d'approvisionnement

## profils d'approvisionnement - ad hoc

- permet à d'installer sur une liste d'appareils iPad, iPhone ou iPod
- souvent utilisé pour des livraisons de tests aux clients
- installation possible en OTA
- uniquement avec une licence **Developer Program** et **Developer Enterprise Program**

# licences, certificats et profils d'approvisionnement

## profils d'approvisionnement - App Store

- permet de compiler l'application pour soumission à l'App Store
- installation impossible sur appareil
- uniquement avec une licence **Developer Program**

# licences, certificats et profils d'approvisionnement

## profils d'approvisionnement - In House

- permet à d'installer sur n'importe quel appareil
- installation possible en OTA
- uniquement avec une licence **Developer Enterprise Program**
- la diffusion des applications doit rester interne à l'entreprise, sinon cela peut être un motif de révocation de la licence

# questions ?

Business  
Services

