Les bonnes pratiques

Let’s do things the right way...

Les bonnes pratiques

- Design pattern
- MVC
- Objets assistants
- Catégories

Design pattern

Design Pattern

[...]Un patron de conception est un arrangement caractéristique de modules, reconnu comme bonne pratique en réponse à un problème de conception d'un logiciel.

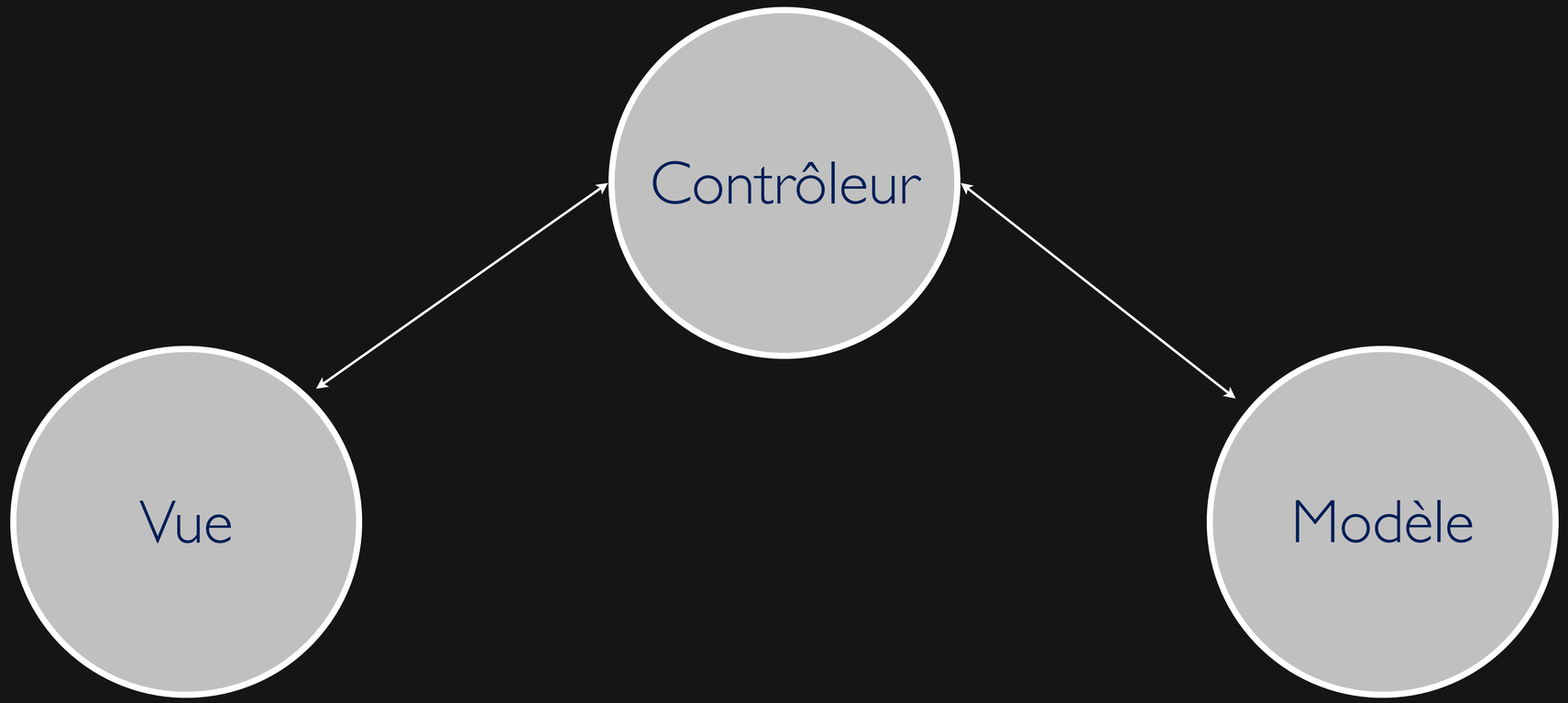
Design Pattern

Il décrit une solution standard, utilisable dans la conception de différents logiciels.

Wikipedia

MVC

MVC

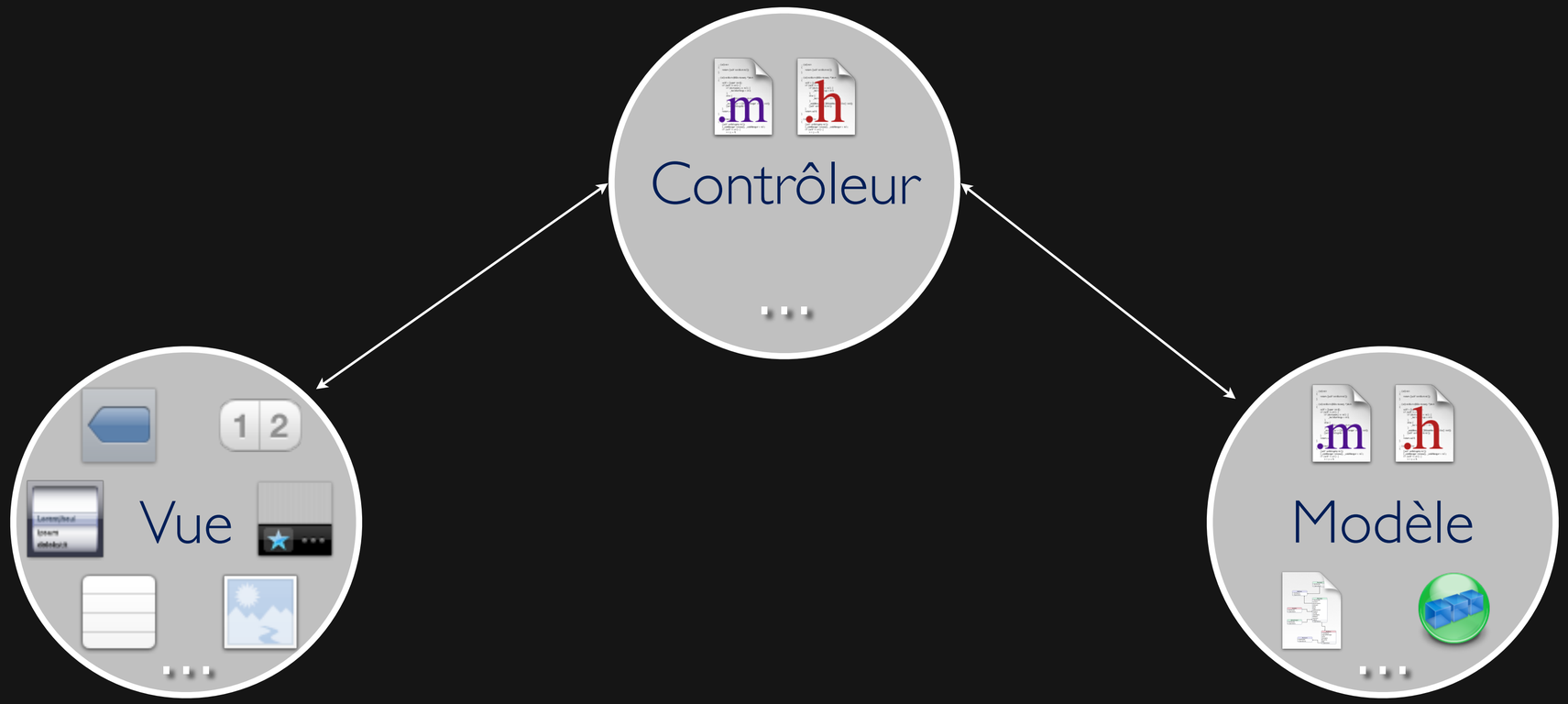


- Modèle
  - Gestion et manipulation des données
  - Contient la base de données éventuelle
  - Gère le cache
  - Définit la représentation des données
  - Réutilisable

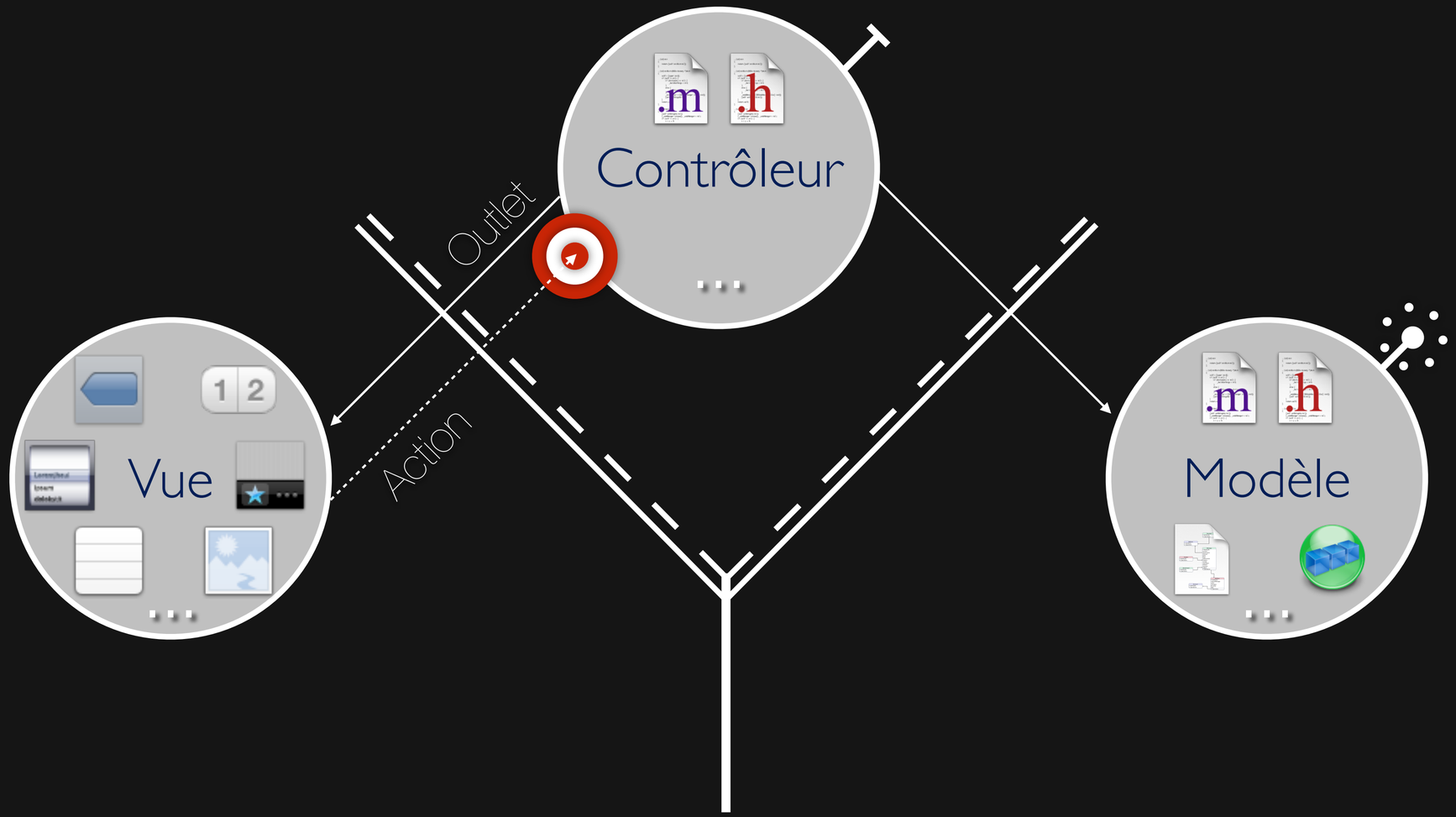
- Vue
  - Elément en interaction avec l’utilisateur
  - Affiche les résultats à l’utilisateur
  - Récupère les actions de l’utilisateur
  - Réutilisable

- Contrôleur
  - Fait le lien entre la vue et le modèle
  - Adapte les données à la vue
  - Interprète les actions sur la vue
  - «glue-code»
  - Rarement réutilisable

En pratique



En pratique



Sur iOS

- UIViewController
  - Une sous classe par écran ou fonctionnalité

Objets assistants

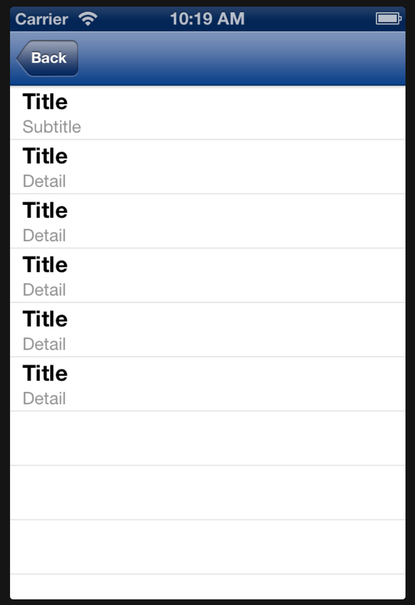
- Déléguer certains fonctionnements à un objet externe
- Prévoir une modification du comportement sans sous classer
- Nécessite l’utilisation de protocole

- Protocole
  - Déclaration de méthode abstraite
  - Définit l’interface à implémenter par d’autres développeurs
  - Peut hériter d’un autre protocole
  - Correspond aux «interfaces» en Java
  - On déclare la conformité à un protocole en le rajoutant entre <> après le nom de la classe

```
@interface MaClasse : NSObject <MonProtocole>
```

- 2 types principaux d’objets assistants
  - DataSource
    - Sert de source de données à afficher
  - Delegate
    - Effectue certaines actions pour le compte d’un autre objet

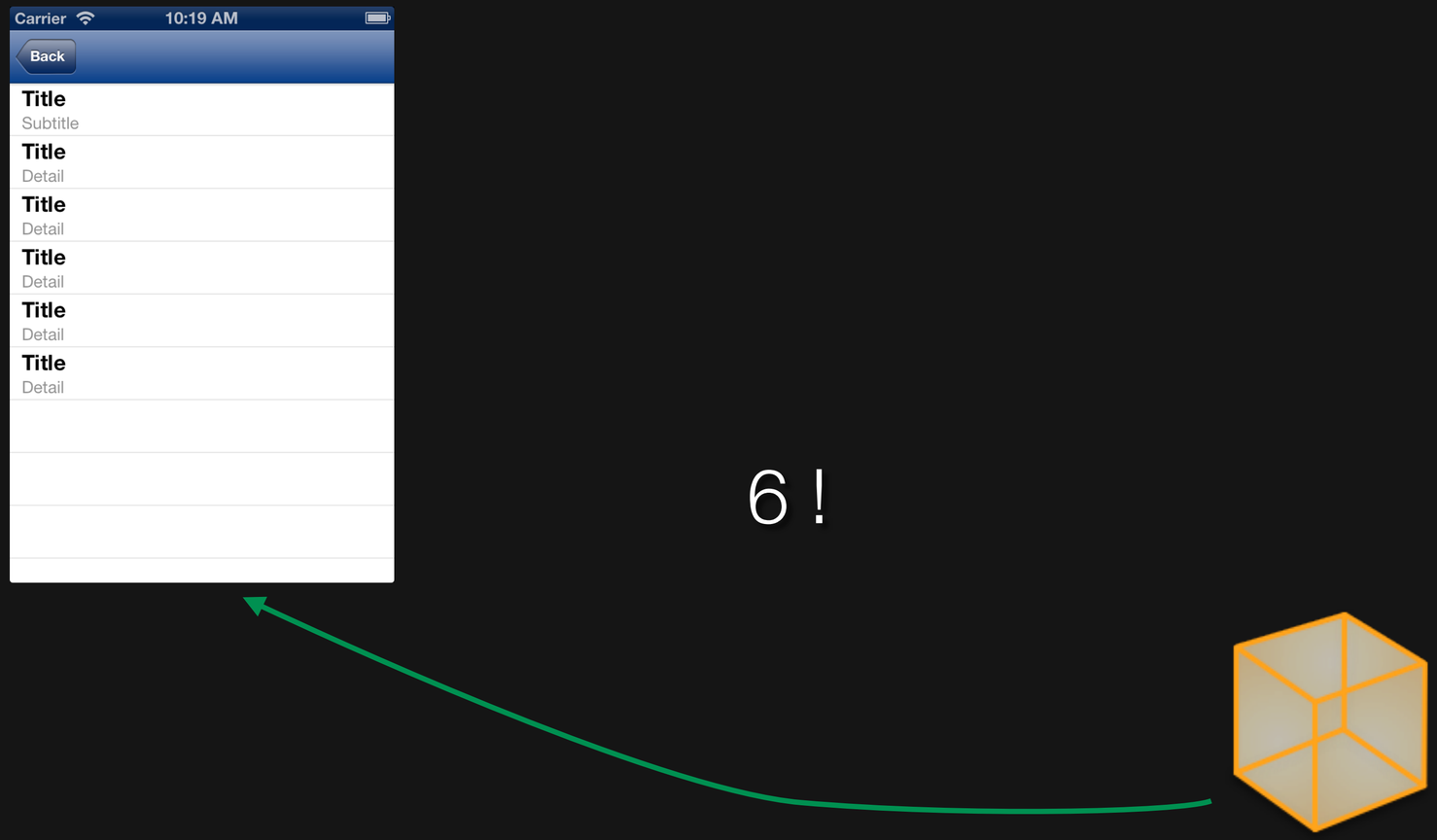
Objets assistants



Combien de lignes ?

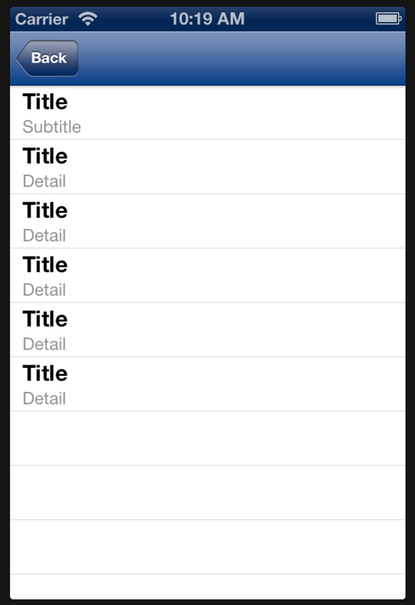
Exemple avec `UITableView` et son protocole `UITableViewDataSource`

Objets assistants



Exemple avec `UITableView` et son protocole `UITableViewDataSource`

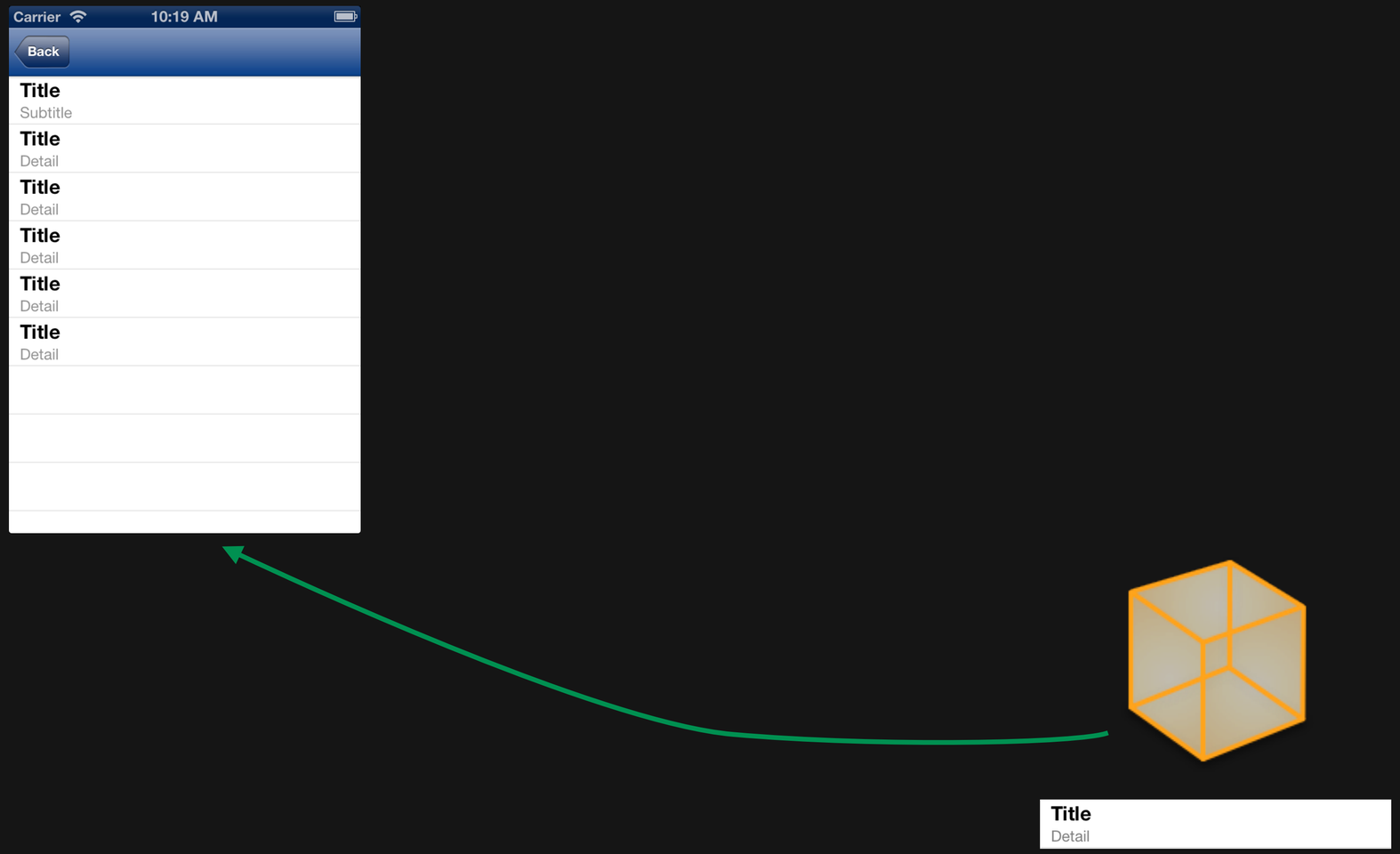
Objets assistants



Quel contenu pour
la ligne X ?

Exemple avec `UITableView` et son protocole
`UITableViewDataSource`

Objets assistants

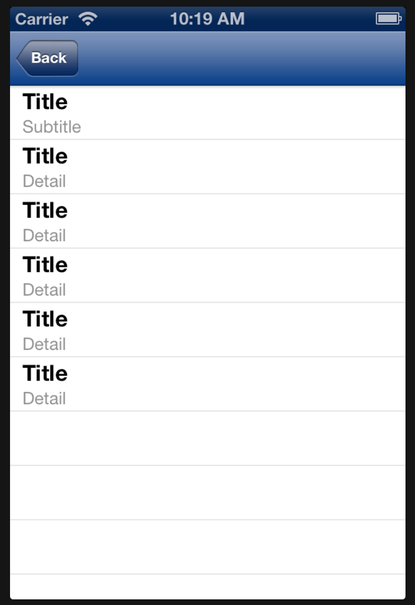


Exemple avec `UITableView` et son protocole `UITableViewDataSource`

UITableViewDataSource

- 2 méthodes obligatoires à implémenter

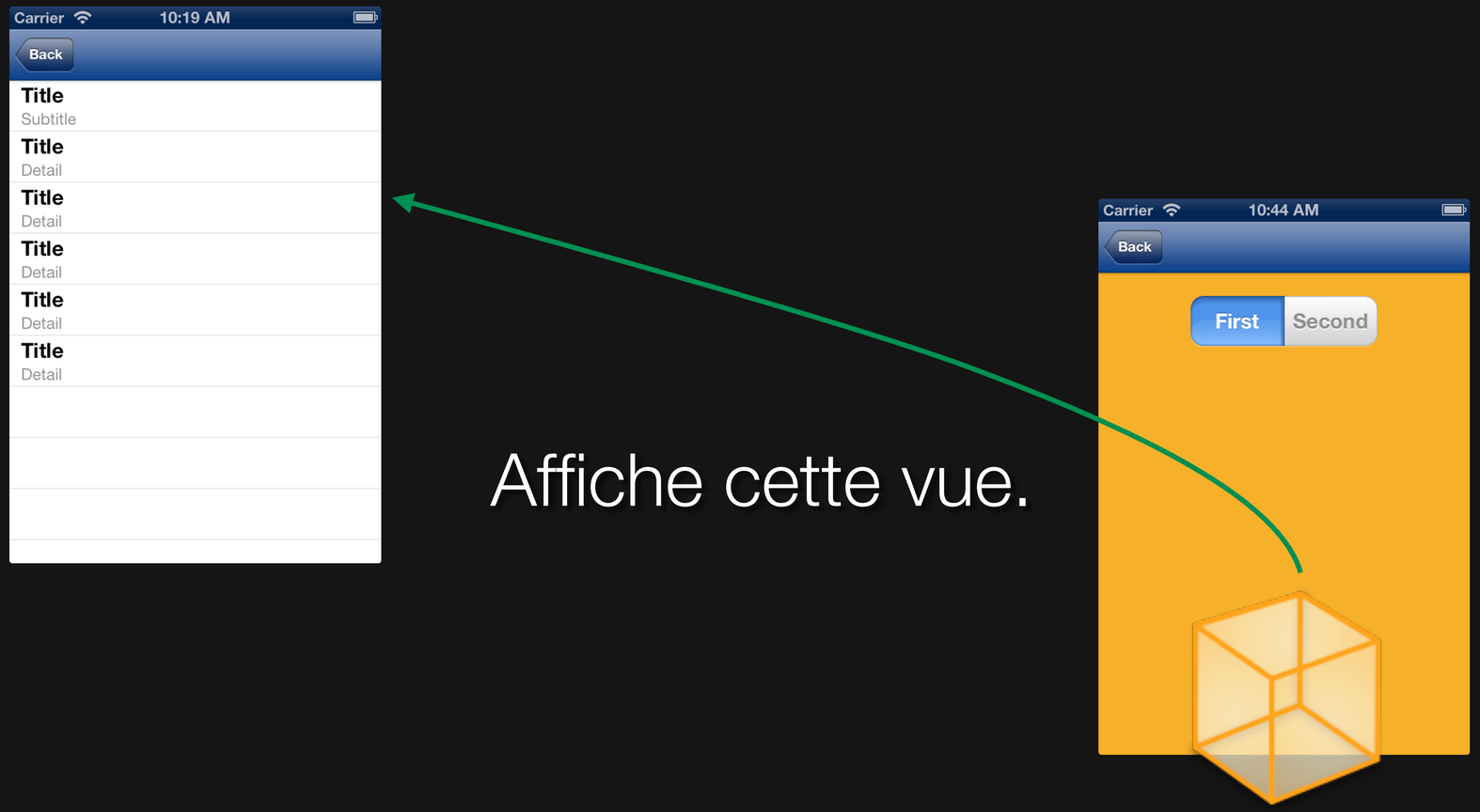
```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:
(NSInteger)section

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:
(NSIndexPath *)indexPath
```



On a appuyé sur la cellule X, je fais quoi ?

Exemple avec `UITableView` et son protocole `UITableViewDelegate`

Objets assistants



Exemple avec `UITableView` et son protocole `UITableViewDelegate`

Catégories

- Sous classer n’est pas toujours nécessaire
- Possibilité de modifier une classe existante directement
- Nécessite uniquement le .h de la classe d’origine
- Porte sur tous les objets de ce type de notre application

Catégories

- Ajout de possibilité sur des objets récupérés
- Sert également à organiser son code

Exemple

```objc
#import "UIColor+MoreColors.h"

@implementation UIColor (MoreColors)

+ (UIColor*)pinkColor {

    double red = (double)255/255;
    double green = (double)45/255;
    double blue = (double)90/255;

    return [UIColor colorWithRed:red green:green blue:blue alpha:1];
}

@end
```

```objc
#import <UIKit/UIKit.h>

@interface UIColor (MoreColors)

+ (UIColor*)pinkColor;

@end
```

- Fonctionne pour les méthodes de classes et d’instances
- Ne peut pas redéfinir une méthode implémentée au même niveau
- Ne peut pas rajouter de variables d’instances

Protocoles

Généralités

- Un protocole défini le cadre à *respecter* en terme de méthodes pour réaliser une tâche spécifique.
- Le protocole peut ensuite être *adopté* par des classes pour fournir la fonctionnalité.
- On dit d'une classe qui satisfait aux prérequis d'un protocole, qu'elle s'y *conforme.*
- Le protocole ne fait que poser des déclaration. C'est lors de l'adoption que l'on se charge d'implémenter les méthodes.

Généralités

```
@protocol ProtocolName

- (void)anInstanceMethodThatShouldBeImplemented;
+ (void)aClassMethodThatShouldBeImplemented;

@end
```

Généralités

```
@protocol ProtocolName

- (void)anInstanceMethodThatShouldBeImplemented;
+ (void)aClassMethodThatShouldBeImplemented;

@end


@interface MyClass: NSObject <ProtocolName>

@end
```

- On indique la conformité de nos classes envers le protocole avec le nom du, ou des protocoles entre <> au niveau de l’interface

```objc
@protocol ProtocolName

- (void)anInstanceMethodThatShouldBeImplemented;
+ (void)aClassMethodThatShouldBeImplemented;

@end

@interface MyClass: NSObject <ProtocolName>

@end

@implementation MyClass

+ (void)aClassMethodThatShouldBeImplemented {
    <#code#>
}

- (void)anInstanceMethodThatShouldBeImplemented {
    <#code#>
}

@end
```

- On indique la conformité de nos classes envers le protocole avec le nom du, ou des protocoles entre <> au niveau de l'interface
- On implémente les méthodes dans le .m

Pour aller plus loin…

- http://developer.apple.com
- The Objective-C Programming Language