Foundation Framework

Jean-Baptiste.Yunes@univ-paris-diderot.fr

2014—2015

- Les frameworks OSX sont des bibliothèques (en plus complexe)
- Ils sont nécessaires pour programmer car ils fournissent des classes et objets prédéfinis relatifs à un contexte
- Le framework de base est `Foundation`
  - essentiel comme son nom l'indique

- On trouve dans `Foundation` :
  - des types et collections utiles
  - un mécanisme de notification
  - des mécanismes de sérialisation et d'archivage
  - des contrôles d'exécution
  - des mécanismes d'E/S
  - gestion des préférences
  - ...

NSObject

Jean-Baptiste.Yunes@univ-paris-diderot.fr / 2014–2015 Master II

- C’est normalement la classe racine de toutes les hiérarchies d’Objective-C
  - il est possible de ne pas l’utiliser mais les avantages à l’utiliser sont infiniment nombreux
    - **aucune** bonne raison de ne pas l’utiliser…
  - de nombreuses fonctionnalités reposent sur les services fournis dans `NSObject`

NSString

Jean-Baptiste.Yunes@univ-paris-diderot.fr / 2014–2015 Master II

- La classe la plus utilisée :
  - manipuler des chaînes de caractères
  - gère correctement l'UTF
  - elles sont immuables
    - il existe des `NSMutableString`
  - l'affichage passe par la directive `%@` pour les objets, qui déclenche un appel à la méthode `description` (de `NSObject`!)

```
NSArray *tab = [NSArray arrayWithObjects:
  @“un“, @“deux“, @“trois“, nil];


NSLog(@“il y a %d elements dans %@“,
  [tab count],tab);
```

- de nombreuses méthodes

```
NSArray *tab = [NSArray arrayWithObjects:
                  @"ici",@"là", nil];
 for (NSString *s in tab) {
   NSString *r =[@"Bonjour," stringByAppendingFormat:
                   @" %@ il fait beau non ?",s];
   NSLog(@"%@",r);
 }
```

- des conversions

```
NSString *a = @"1234";
 int v = [a intValue];
NSLog(@"%d",v);
```

NSNumber

Jean-Baptiste.Yunes@univ-paris-diderot.fr / 2014–2015 Master II

- La classe wrapper pour les types primitifs nombres du C

```
NSNumber *n = [NSNumber numberWithInt:10];
NSLog(@"%@ %d",n,[n intValue]);
```

NSData

Jean-Baptiste.Yunes@univ-paris-diderot.fr / 2014–2015 Master II

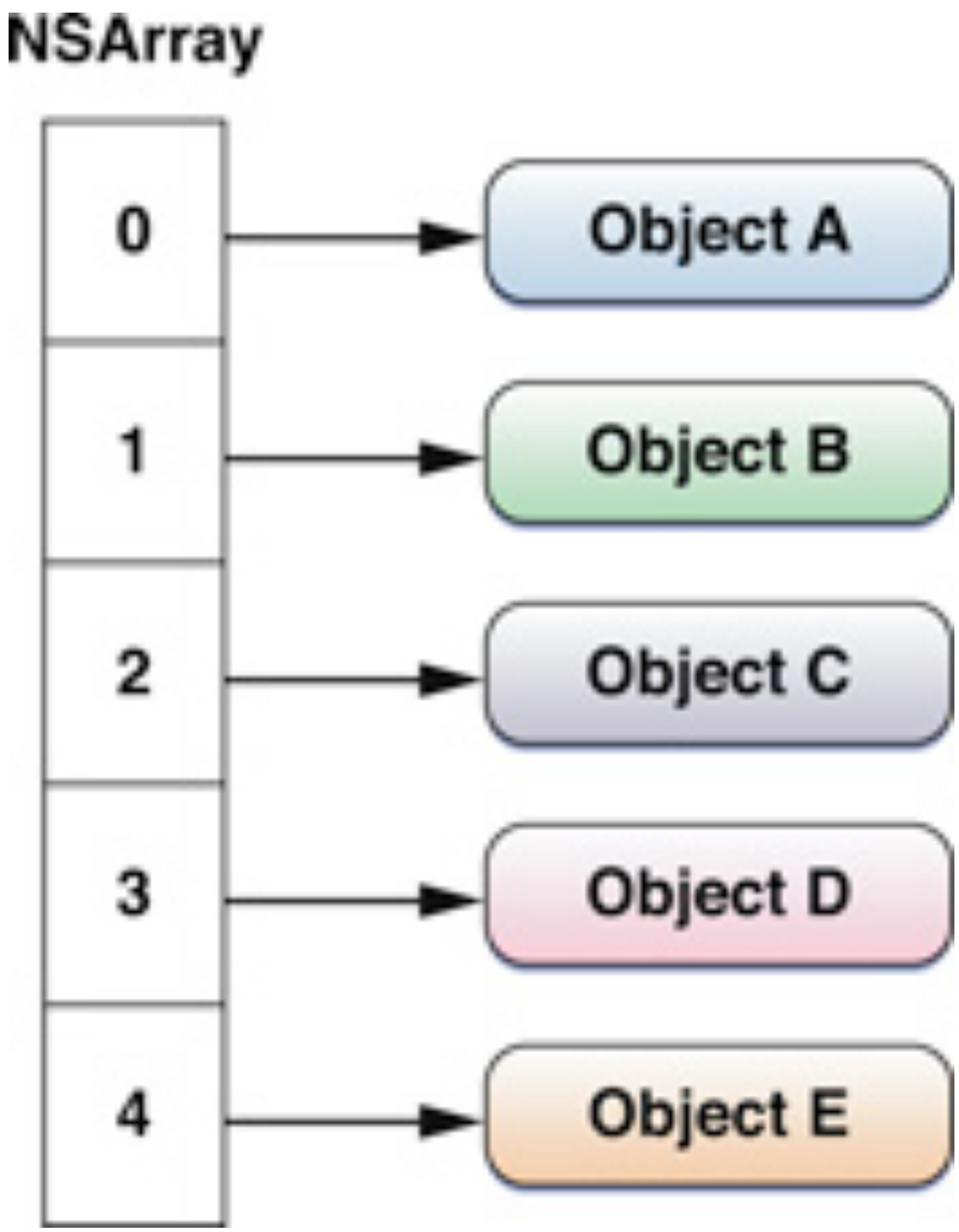
- La classe wrapper pour les zones mémoire (attention copie de la zone d'origine, ObjC a sa propre gestion de la mémoire)

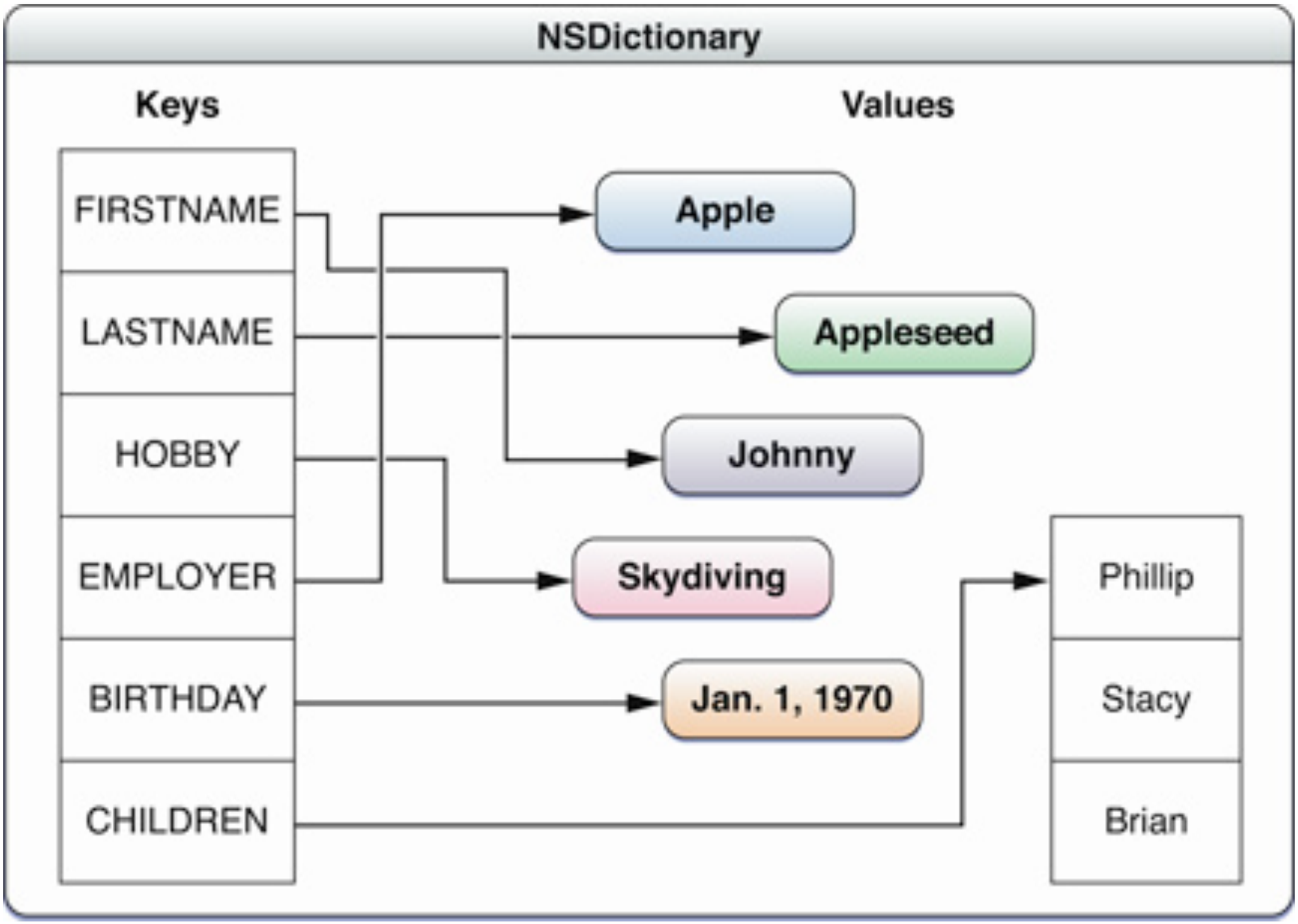
```
void *mem = malloc(123);

NSData *buffer = [NSData
            dataWithBytes:mem length:123];

void *memBack = [buffer bytes];
```

Les collections

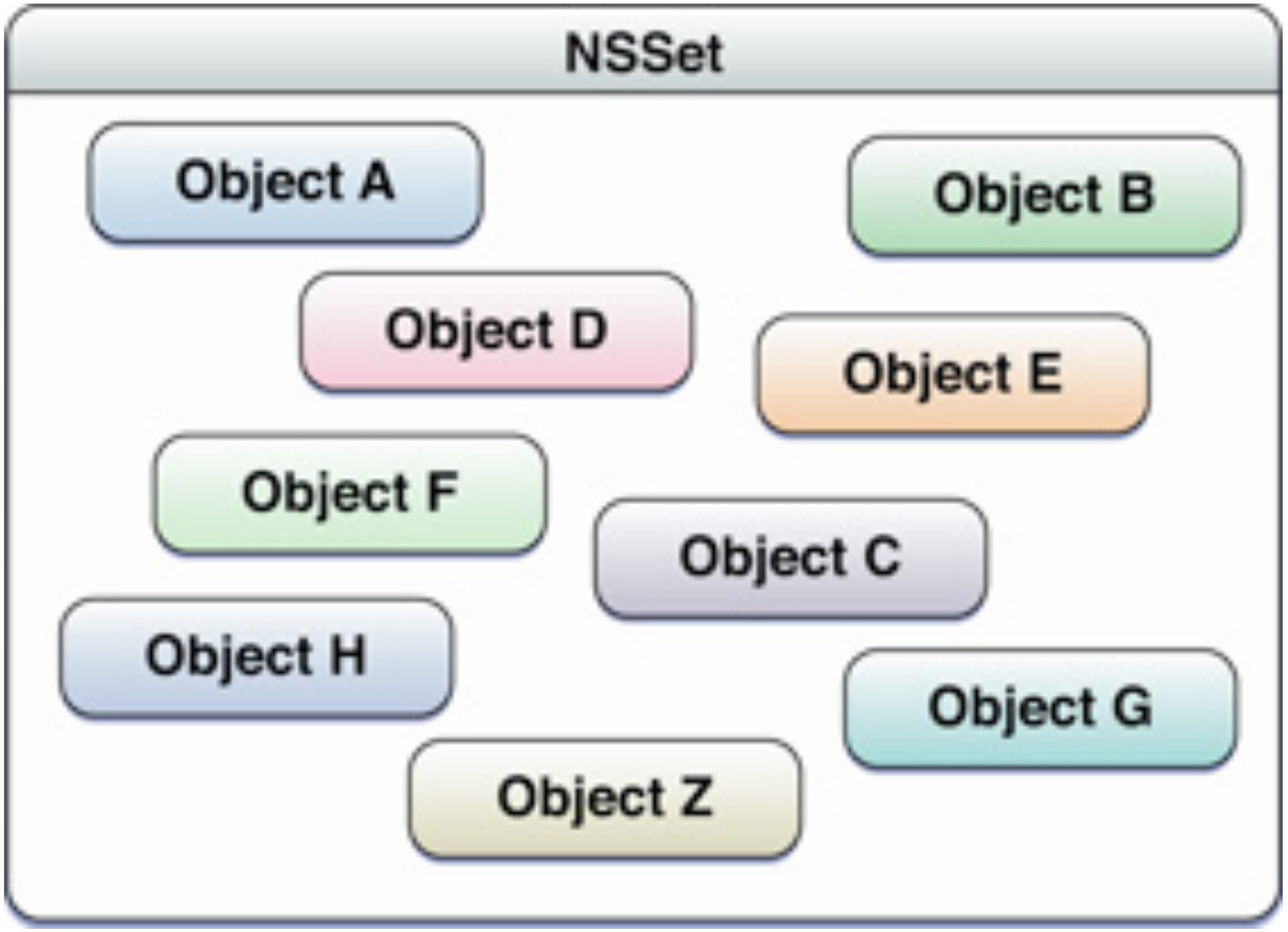
Jean-Baptiste.Yunes@univ-paris-diderot.fr / 2014–2015 Master II

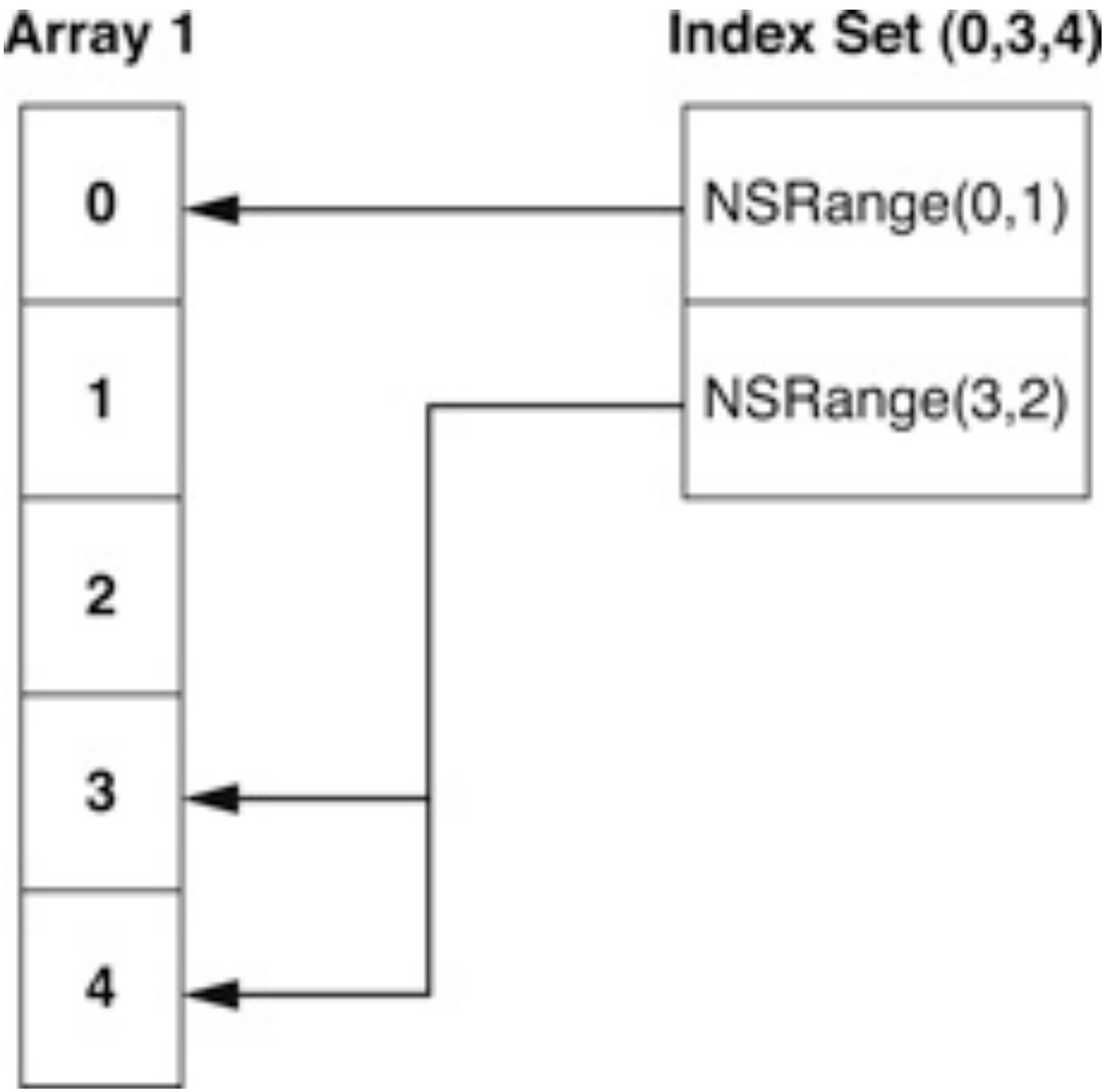
- Tableaux
  - `NSArray` (`NSMutableArrays`)
- Clé-Valeur
  - `NSDictionnary` (`NSMutableDictionnary`)
- Ensembles non ordonnés
  - `NSSet` (`NSMutableSet`), `NSCountedSet`
- Classes utilitaires
- `NSIndexSet`, `NSIndexPath`

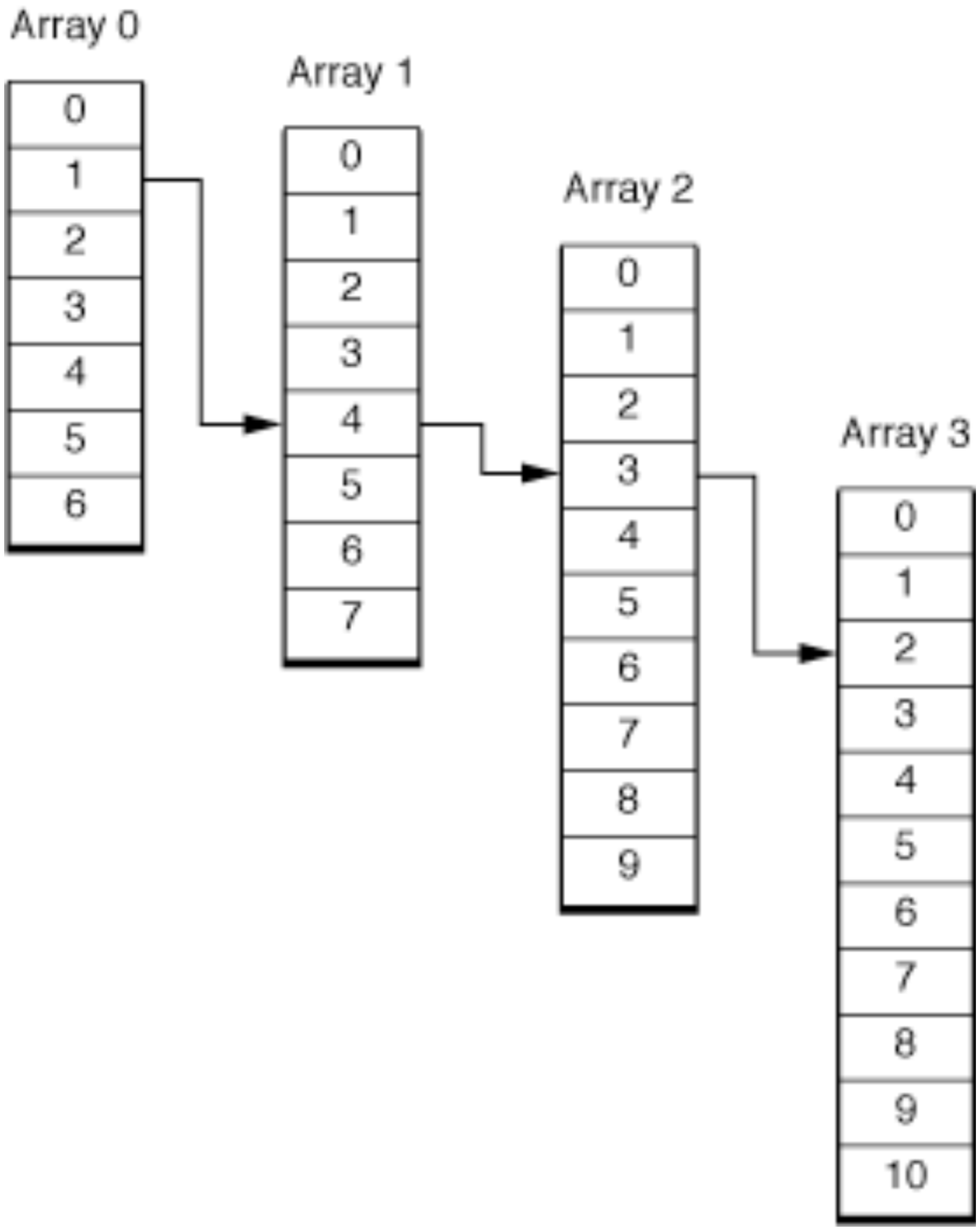

NSArray
0
1
2
3
4
Object A
Object B
Object C
Object D
Object E


NSDictionary
Keys
Values
FIRSTNAME
LASTNAME
HOBBY
EMPLOYER
BIRTHDAY
CHILDREN
Apple
Appleseed
Johnny
Skydiving
Jan. 1, 1970
Phillip
Stacy
Brian

```
NSString *last   = @"lastName";
NSString *first  = @"firstName";
NSString *suffix = @"suffix";
NSString *title  = @"title";

NSMutableDictionary *dict = [NSMutableDictionary
dictionaryWithObjectsAndKeys:
    @"Jo",first,@"Smith",last,nil];

NSDictionary *newDict = [NSDictionary
dictionaryWithObjectsAndKeys:
    @"Jones",last,@"Hon.",title,@"J.D.",suffix,
nil];

[dict addEntriesFromDictionary: newDict];
```


NSSet
Object A
Object B
Object D
Object E
Object F
Object C
Object H
Object G
Object Z

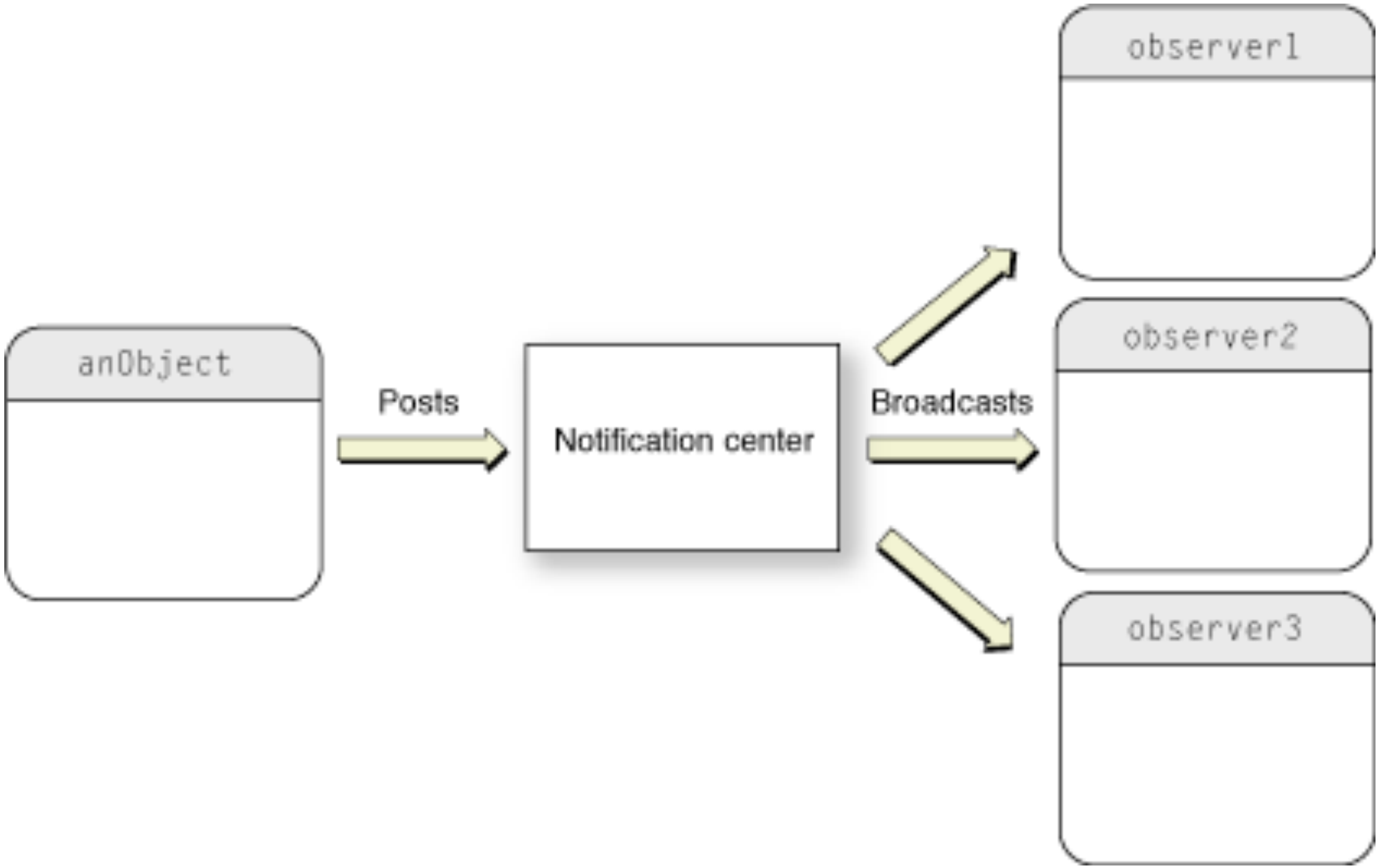

Array 1
Index Set (0,3,4)
0
1
2
3
4
NSRange(0,1)
NSRange(3,2)


Array 0
0
1
2
3
4
5
6
Array 1
0
1
2
3
4
5
6
7
Array 2
0
1
2
3
4
5
6
7
8
9
Array 3
0
1
2
3
4
5
6
7
8
9
10

Les notifications

Jean-Baptiste.Yunes@univ-paris-diderot.fr / 2014–2015 Master II

- Le mécanisme permet de communiquer entre objets qui ne se connaissent pas
  - par exemple un fichier souhaite être prévenu lorsque qu'un fenêtre l'éditant se ferme (le fichier ne connaît pas directement la fenêtre!)
- `NSNotificationCenter`, (intraprocess) `NSDistributedNotificationCenter` (interprocess)
- `NSNotification`


observer1
anObject
Posts
Notification center
Broadcasts
observer2
observer3

```objc
// UneClasse.h
#import <Foundation/Foundation.h>

@interface UneClasse : NSObject
- (id)somethingHappened:(NSNotification *)notification;
@end

// UneClasse.m
#import "UneClasse.h"
@implementation UneClasse
- (id)somethingHappened:(NSNotification *)notification {
    NSLog(@"ok, received!");
}
@end

// Un objet receveur
UneClasse *obj = [[UneClasse alloc] init];

// on l'enregistre auprès du centre de notification choisi
[[NSNotificationCenter defaultCenter] addObserver:obj
  selector:@selector(somethingHappened:) name:@"bip" object:nil];

// On poste l'évènement (il sera reçu à un moment adéquat)
[[NSNotificationCenter defaultCenter] postNotificationName:@"bip" object:nil];

// On enlève le receveur du système de notification
[[NSNotificationCenter defaultCenter] removeObserver:obj];
```

La sérialisation

Jean-Baptiste.Yunes@univ-paris-diderot.fr / 2014–2015 Master II

- l'insertion d'un objet dans une archive nécessite son codage (l'inverse est le décodage)
  - un objet devant être archivé doit supporter le protocole NSCoding

- le protocole `NSCoding` impose deux méthodes :
  - `-(id)initWithCoder:(NSCoder *)decoder` pour créer un objet depuis un `NSCoder`
  - `-(void)encodeWithCoder:(NSCoder *)coder` pour encoder un objet vers un `NSCoder`
- `NSCoder` est la super-classe des archiveurs
- le corps des deux méthodes doit contenir de quoi archiver les attributs (donc des appels aux méthodes d'en/dé-codage des codeurs associés)

- NSCoder possède des méthodes pour encoder (NSDecoder pour décoder :-)
  - `-(void)encodeBool:(BOOL) b forKey:(String) k`
  - `-(void)encodeInt:(int) i forKey:(String) k`
  - …

- il y a deux types d’archives
  - les archives séquentielles (pour archiver des graphes d’objets)
  - les archives indexées (pour archiver des objets à l’aide de clés)

- `NSKeyedArchiver`
  - `+(BOOL)archiveRootObject:(id)root toFile:(NSString *)path` permet d'enregistrer un graphe d'objets dans un fichier
- `NSKeyedUnarchiver`
  - `+(id)unarchiveObjectWithFile:(NSString *)path` pour relire le graphe d'objets contenu dans un fichier

- pour archiver individuellement des objets il faut passer par un `NSMutableData` associé à un `NSKeyedArchiver`
  - la sérialisation s'effectuera dans le `NSMutableData` lequel sera ensuite écrit dans un fichier ou transmis par le réseau, etc
  - `-(id)initForWritingWithMutableData:(NSMutableData *)data`

Les préférences

- Les préférences sont un ensemble de valeurs stockées de sorte que l'application puisse s'en servir afin d'obtenir certains comportements :
  - police par défaut pour un éditeur
  - adresse mail
  - couleur de fond
  - …

- Les préférences sont construites à partir de 5 domaines de stockage
  - `NSArgumentDomain`
  - le domaine de l'application
  - `NSGlobalDomain`
  - le domaine de l'internationalisation
  - `NSRegistrationDomain`
- Sous OSX la commande `defaults` permet de manipuler les préférences en ligne de commande

- la classe `NSUserDefaults`
  - permet de manipuler les préférences relatives à une application donnée
  - on en obtient une instance par appel à :

```
+(NSUserDefaults)standardUserDefaults
```

KVO

Jean-Baptiste.Yunes@univ-paris-diderot.fr / 2014–2015 Master II

- Le Key Value Observing est un pattern permettant à un objet d'observer les propriétés d'autres objets
  - très utile pour obtenir un MVC!
    - les vues peuvent être notifiées de la modification d'une propriété du modèle
  - la classe à observer doit être KVC-compliant!

```
//  UneClasse.h
#import <Foundation/Foundation.h>
@interface UneClasse : NSObject
@property () int value;
@end

//  UneClasse.m
#import "UneClasse.h"
@implementation UneClasse
- (NSString *)description {
    return @"that's me!";
}
@end
```

```
//  UnObserver.h
#import <Foundation/Foundation.h>
@interface UnObserver : NSObject
- (void)observeValueForKeyPath:(NSString *)keyPath
          ofObject:(id)object change:(NSDictionary *)change
          context:(void *)context;
@end

//  UnObserver.m
#import "UnObserver.h"
@implementation UnObserver
- (void)observeValueForKeyPath:(NSString *)keyPath
          ofObject:(id)object change:(NSDictionary *)change
          context:(void *)context {
    NSLog(@"%@ for %@ changed",keyPath,object);
    NSLog(@"old value was %@",[change valueForKey:NSKeyValueChangeOldKey]);
    NSLog(@"new value is %@",[change valueForKey:NSKeyValueChangeNewKey]);
}
@end
```

```
UneClasse *obj = [[UneClasse alloc] init];
UnObserver *obs = [[UnObserver alloc] init];

[obj addObserver:obs forKeyPath:@"value"
     options: NSKeyValueObservingOptionNew+
              NSKeyValueObservingOptionOld
     context:nil];

obj.value = 3;
obj.value = 5;

[obj removeObserver:obs forKeyPath:@"value"];
```