

# iOS App Development

2.1 Objective-C Part II: Property & nil

# Agenda

- Why Properties?
- Dot Notation
- **strong** vs. **weak**
- **nil**

# Properties

## • Why Properties?

### 1. Instance Variable의 보호

- Subclass에서 Instance Variable의 값을 임의로 변경할 경우 예상치 못한 상황이 될 위험

### 2. Valve (유일한 통로)

- Lazy Instantiation
- UI update
- 값의 범위 제한
- Getter/Setter가 호출되는 시점

# Properties

- Instance Variables
  - @property 에는 Instance Variable을 연결할 수도, 연결하지 않을 수도 있다
  - 예: 저장된 값이 아닌 계산된값 return할 때
  - 가급적 Instance Variable은 @property 를 통해서 생성할 것을 권장 (주: 코드 가독성의 혼란으로 인한 실수를 막기 위함인 것으로 추정)
  - 참고: @synthesize Directive

# Properties

- Why dot notation?
  - Method 호출보다 @property 접근이 가독성이 뛰어나고 코드가 간결 (Dot Notation: object.name = ...; )
  - C struct의 변수접근과 유사해서 혼합된 코드에서 일관성 유지
  - Objective-C로 된 코드에서는 C struct를 heap에 할당하는 일이 드물어서 ‘->’ 보다는 ‘.’으로 변수 접근
  - value = view.center.x;

# Dot Notation

- @property 접근이 C struct member 접근과 유사

```
typedef struct {  
    float x;  
    float y;  
} CGPoint;
```

```
@interface Bomb  
@property CGPoint position;  
@end
```

```
@interface Ship : Vehicle  
  
@property float width;  
@property float height;  
@property CGPoint center;  
  
- (BOOL) getsHitByBomb:(Bomb *) bomb;  
@end
```

● CGPoint

● 마치 @p

있음

● @implementation Ship

● @synthesize width, height, center;

● - (BOOL) getsHitByBomb:(Bomb \*) bomb

{

● float leftEdge = self.center.x - self.width/2;  
● float rightEdge = ...;

● return ((bomb.position.x >= leftEdge) &&  
● (bomb.position.x <= rightEdge) &&  
● (bomb.position.y >= topEdge) &&  
● (bomb.position.y <= bottomEdge));

}

● @end

Normal  
C

● @property

# strong vs. weak

- It's About Object Memory Management
- ARC: Automatic Reference Counting (iOS5)
- ARC 지원전에는 Object에 대한 Reference Counting 관련 method를 호출 해 줘야 했다
  - [object **retain**]; // +1
  - [object **release**]; // -1
  - Reference Count == 0: Deallocate

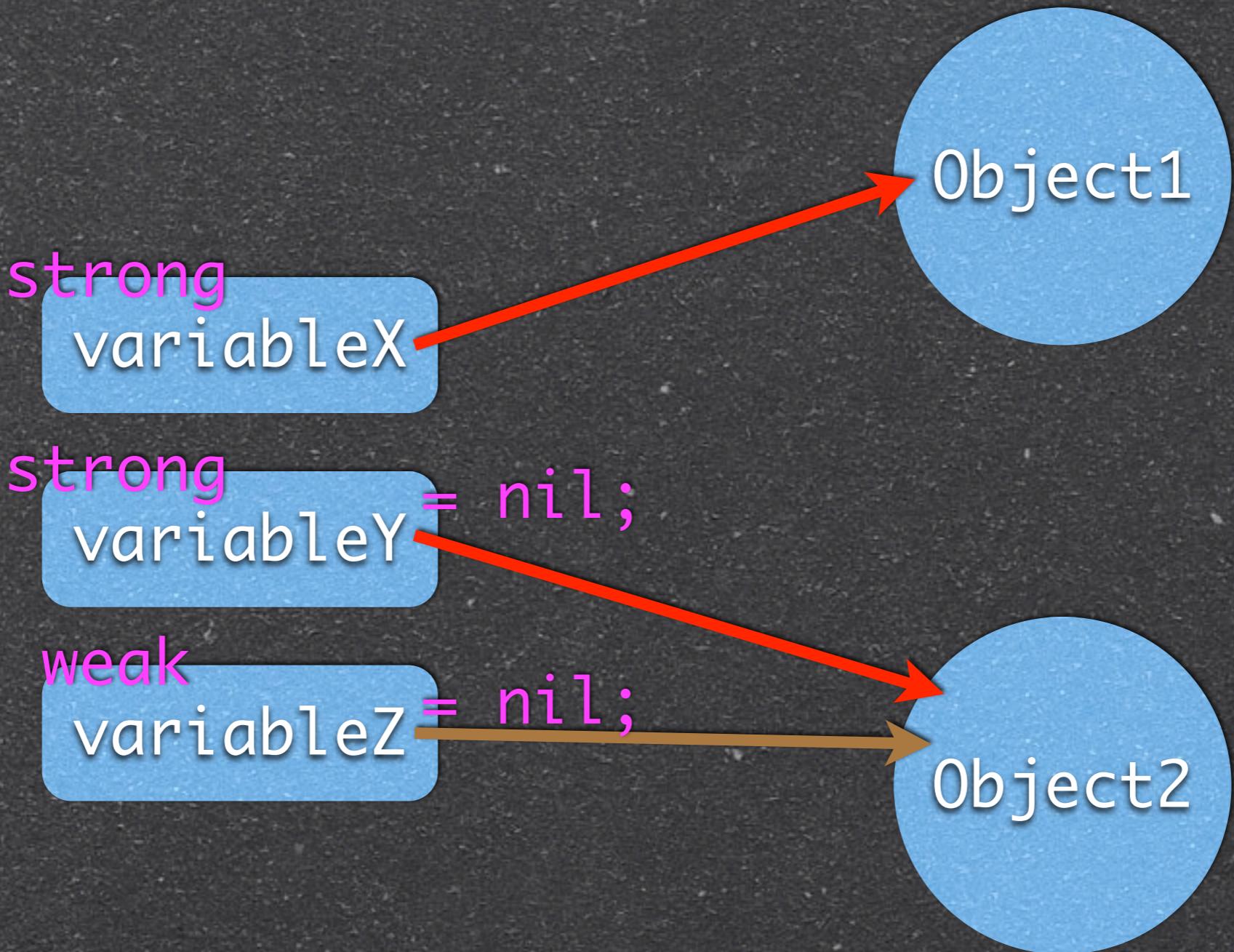
# strong vs. weak

- ARC Enabled (iOS 5, Compiler flag)
- 변수는 Object를 ‘strong’ 혹은 ‘weak’ reference 하도록 속성이 주어진다
  - NSObject \_\_strong object = ...;
  - 생략되면 \_\_strong이 default
- Property attribute
  - @property (nonatomic, strong) ...;
  - @property (nonatomic, weak) ...;

# strong vs. weak

- **strong** “내(object) 가 참조(point)하는한 heap에 머물것”
  - pointer에 **nil**을 대입하면 더이상 참조하지 않음
  - 혹은 내(object)가 heap에서 사라지면 참조 끝
- **weak** “누가 **strong** 참조하는동안만 heap에 머물것”
  - Heap에서 사라질때 pointer에 **nil**이 자동대입
  - Object가 heap에서 사라지는 순간
    - `(void) dealloc; // Instance method of NSObject`

# strong vs. weak



# nil

- 아무것도 참조(point)하지 않는 object pointer의 값

```
id obj = nil;  
NSString *hello = nil;
```

- Primitive Type (int, double같은)의 “zero”같은 존재
  - 실제 “zero”
- 모든 instance variable은 zero로 초기화 된다
  - Object가 생성 ([alloc])될 때
  - Object pointer instance variable은 별도의 초기화를 해 주지 않아도 nil이 대입된다

# nil

- if문에서 편리하게 사용 (implicit test)

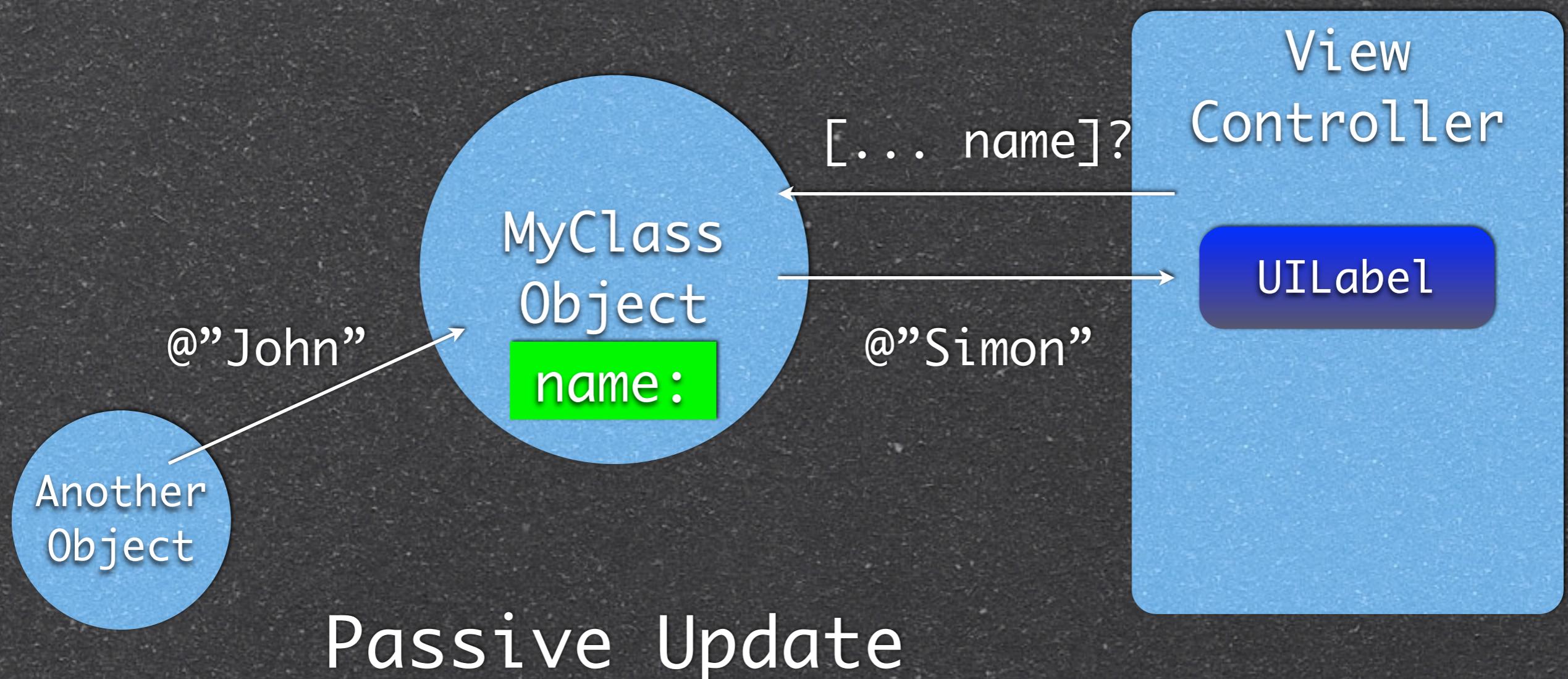
```
if (obj) { /* ... */ }
```

- nil에 message보내도 (어지간하면) 괜찮음. 아무코드도 실행안됨
- Method가 return 하기로 되어 있으면 zero가 return 된다 : `int i = [obj methodReturnsInt];`
- 하지만! return type이 C struct이면 뭐가 return 될지 알수없음: `CGPoint p = [obj getLocation];`

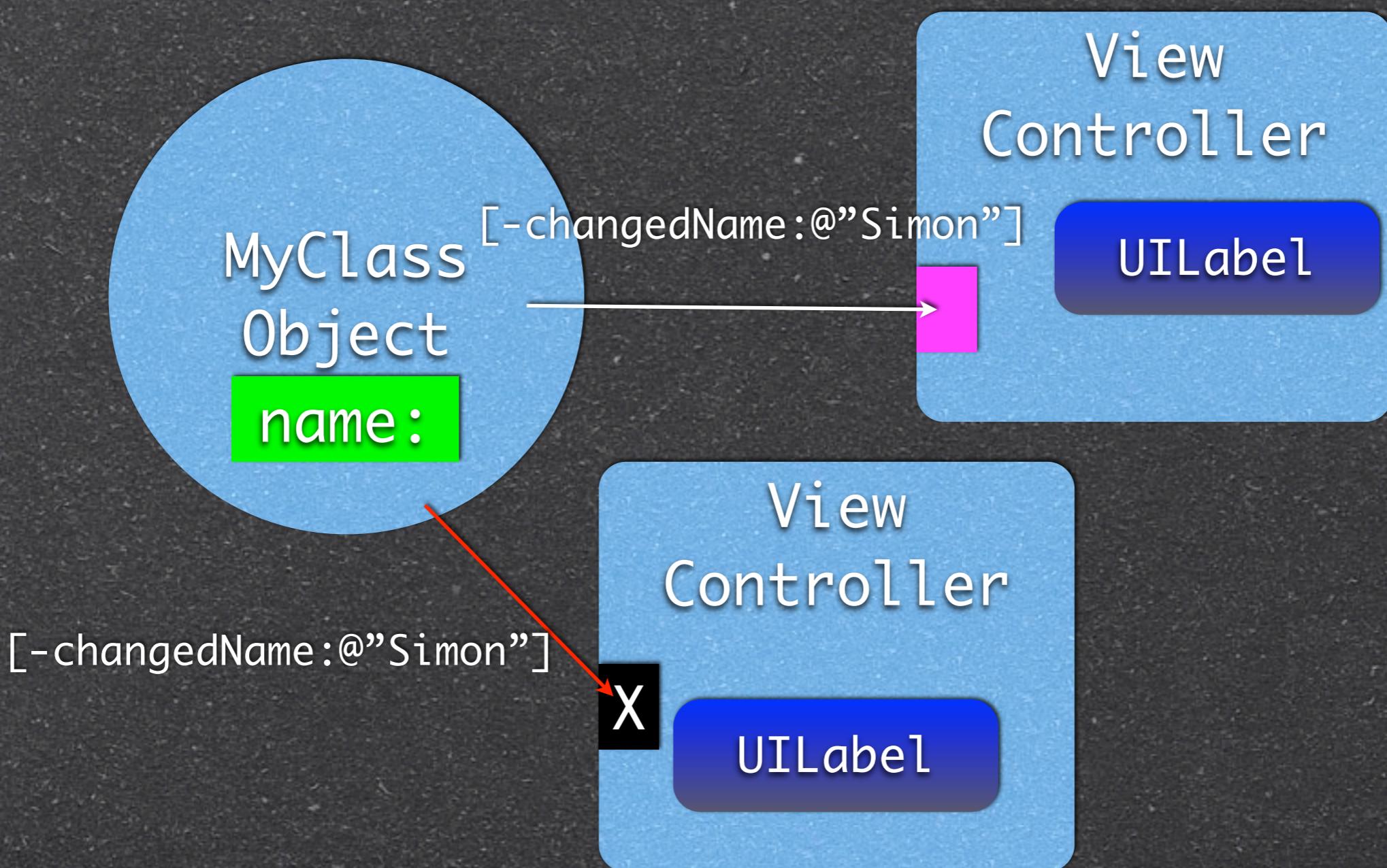
# Protocol

- @protocol Directive
  - Defines what messages can be handled
  - methods
- No @implementation

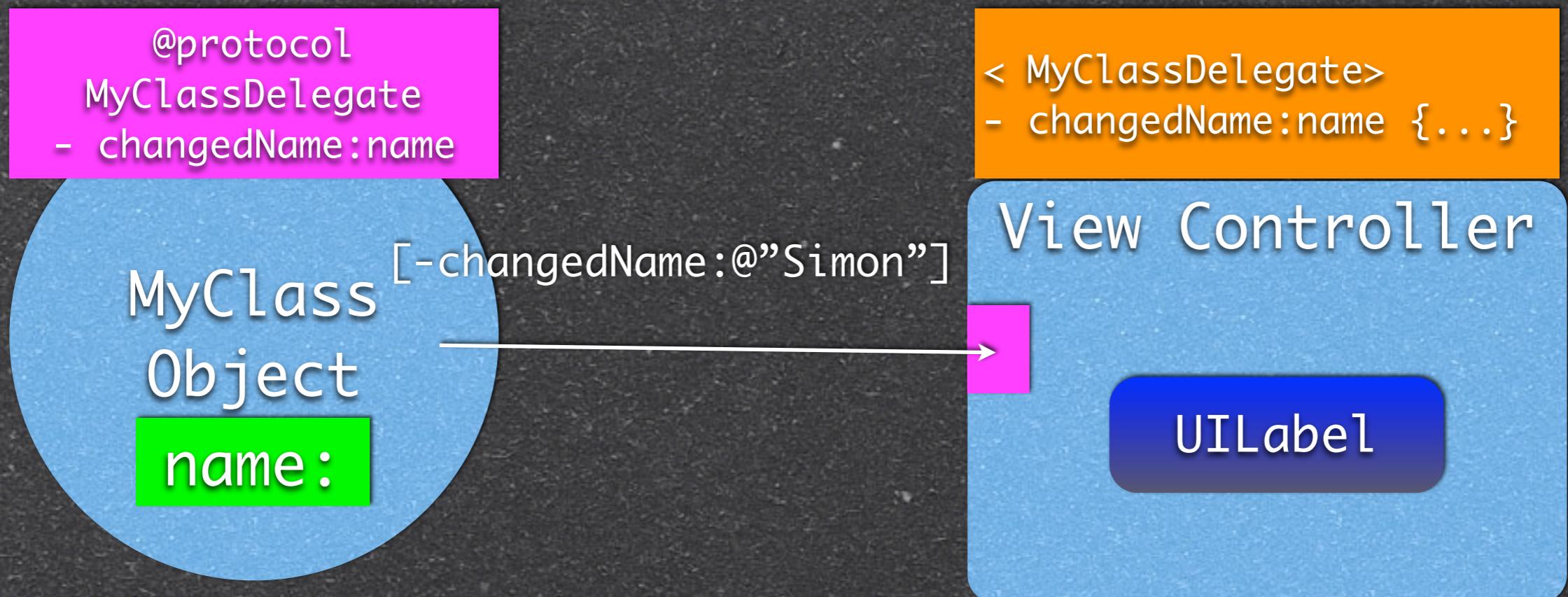
# Object Communication



# Object Communication



# Object Communication



Active Update

# Protocol

- @protocol Definition

```
@protocol MyClassDelegate <NSObject>
- (BOOL) myClass:(MyClass *) object shouldChangeName:(NSString *) name;
- (void) myClass:(MyClass *) object didChangeName:(NSString *) name;
@end
```

# Conforming Protocol

```
@interface MyViewController : UIViewController <MyClassDelegate>
@end

@implementation MyViewController

- (BOOL) myClass:(MyClass *) object shouldChangeName:(NSString *) name
{
    return YES;
}
- (void) myClass:(MyClass *) object didChangeName:(NSString *) name
{
    // ...
}
@end
```