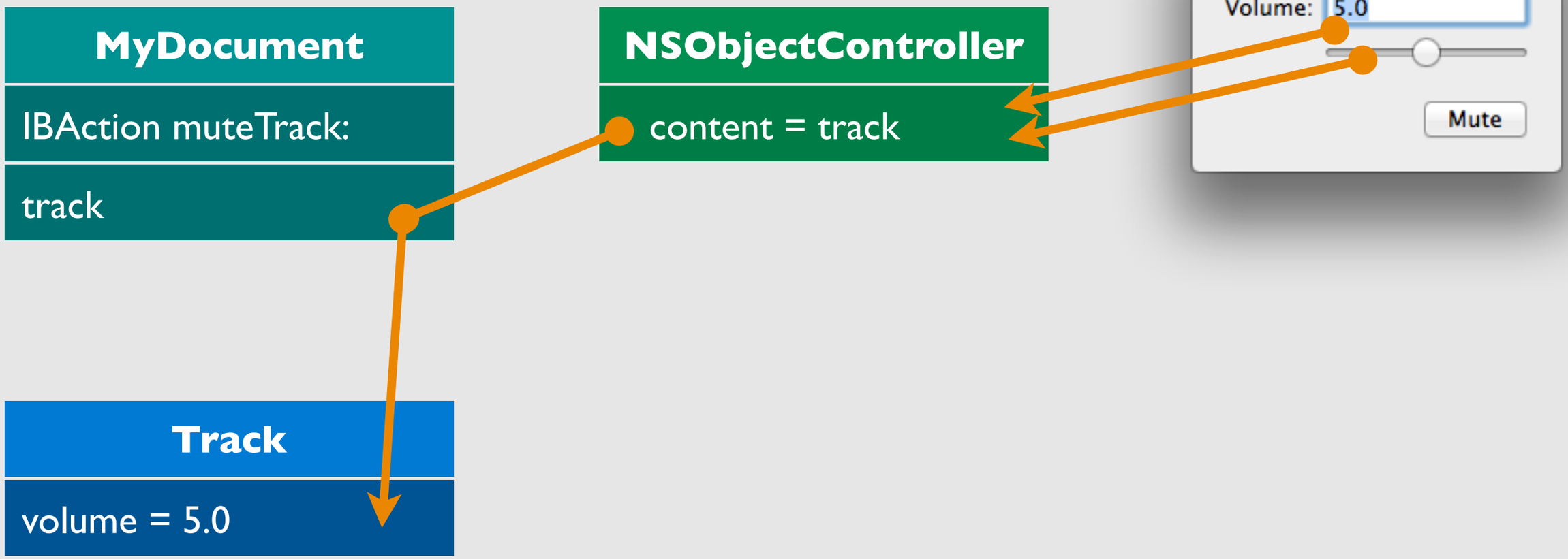


BINDINGS

+ KVC, KVO, NSObjectController and a bit of CoreData
Or: How to avoid writing code.

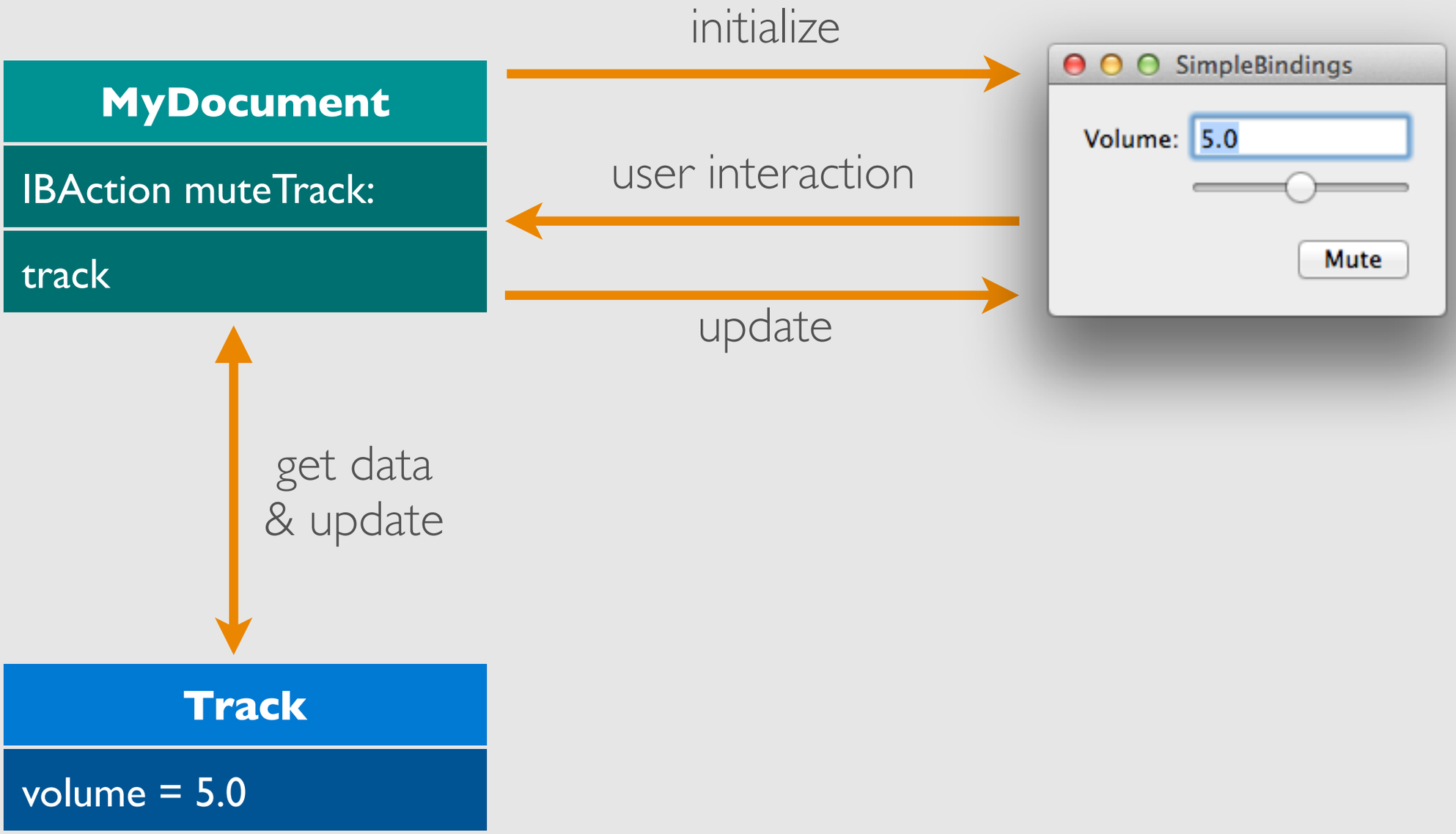
Joachim Kurz

INHALT

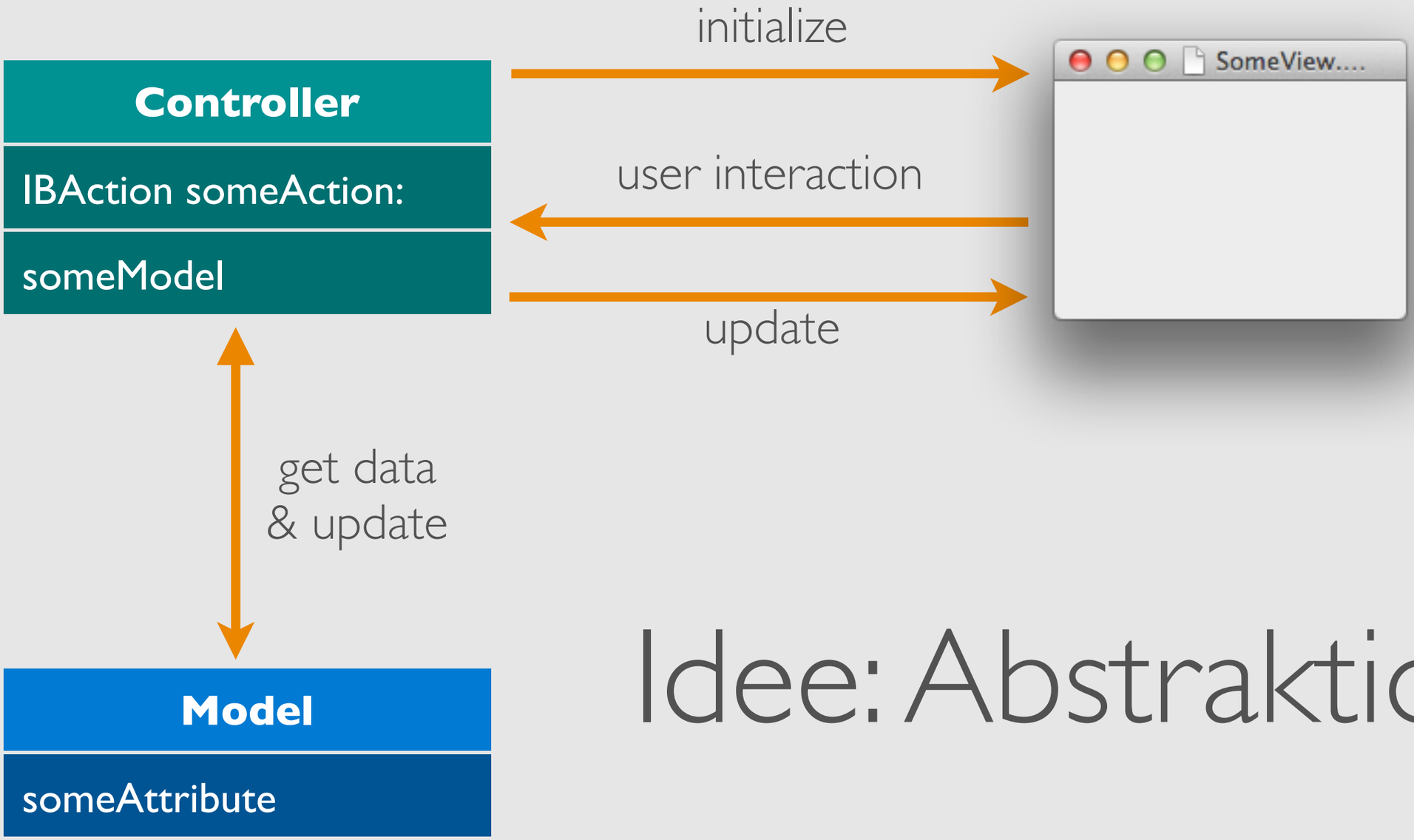
- What are Bindings? Why and what for?
- How do Bindings work?
  - KVC, KVO, KVB
- How to use Bindings + Examples
- Advanced Bindings
- (Binding Compatibale Views)

Demo

PROBLEM

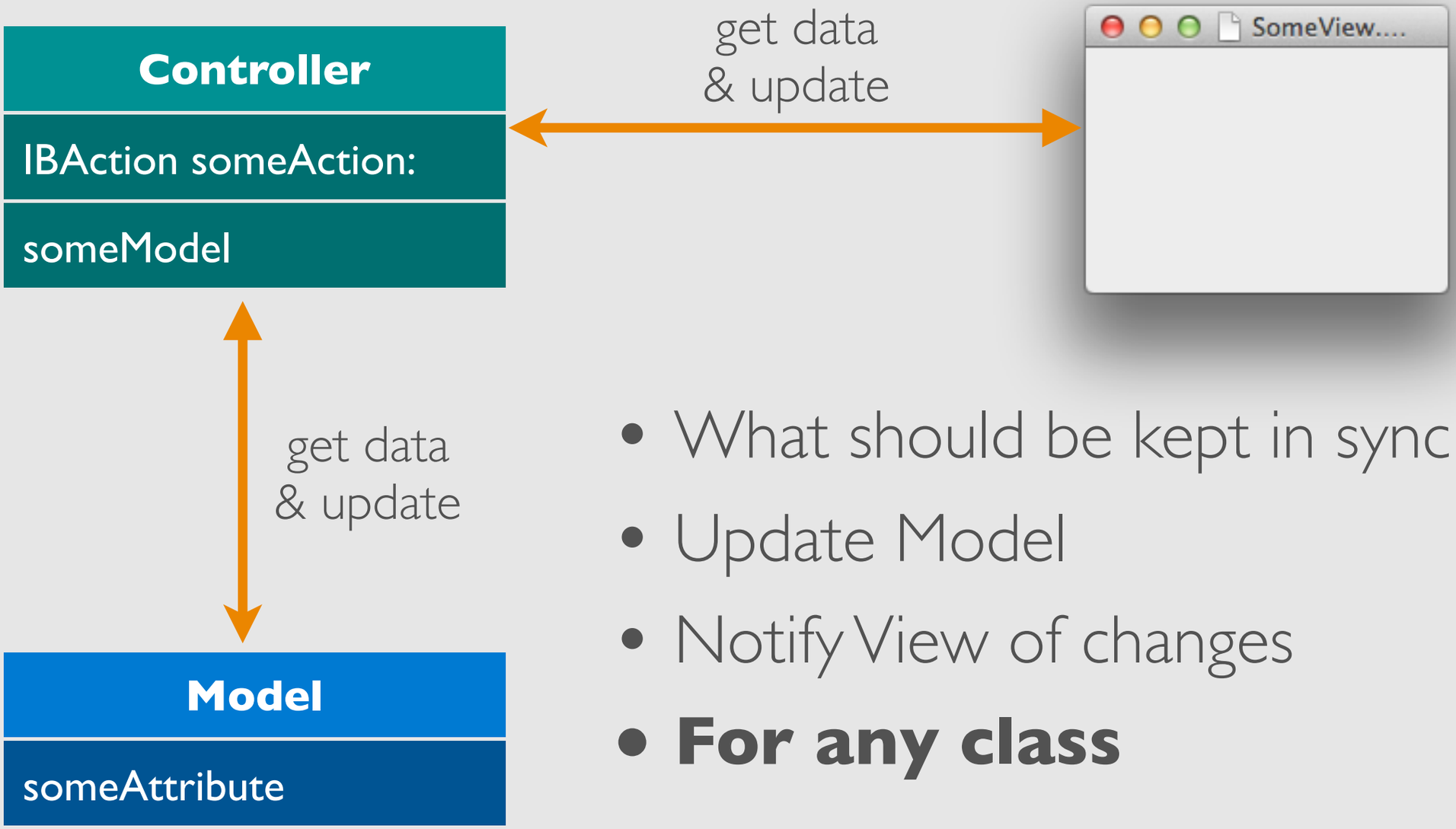

MyDocument
IBAction muteTrack:
track
initialize
user interaction
update
get data
& update
SimpleBindings
Volume:
5.0
Mute
Track
volume = 5.0

PROBLEM



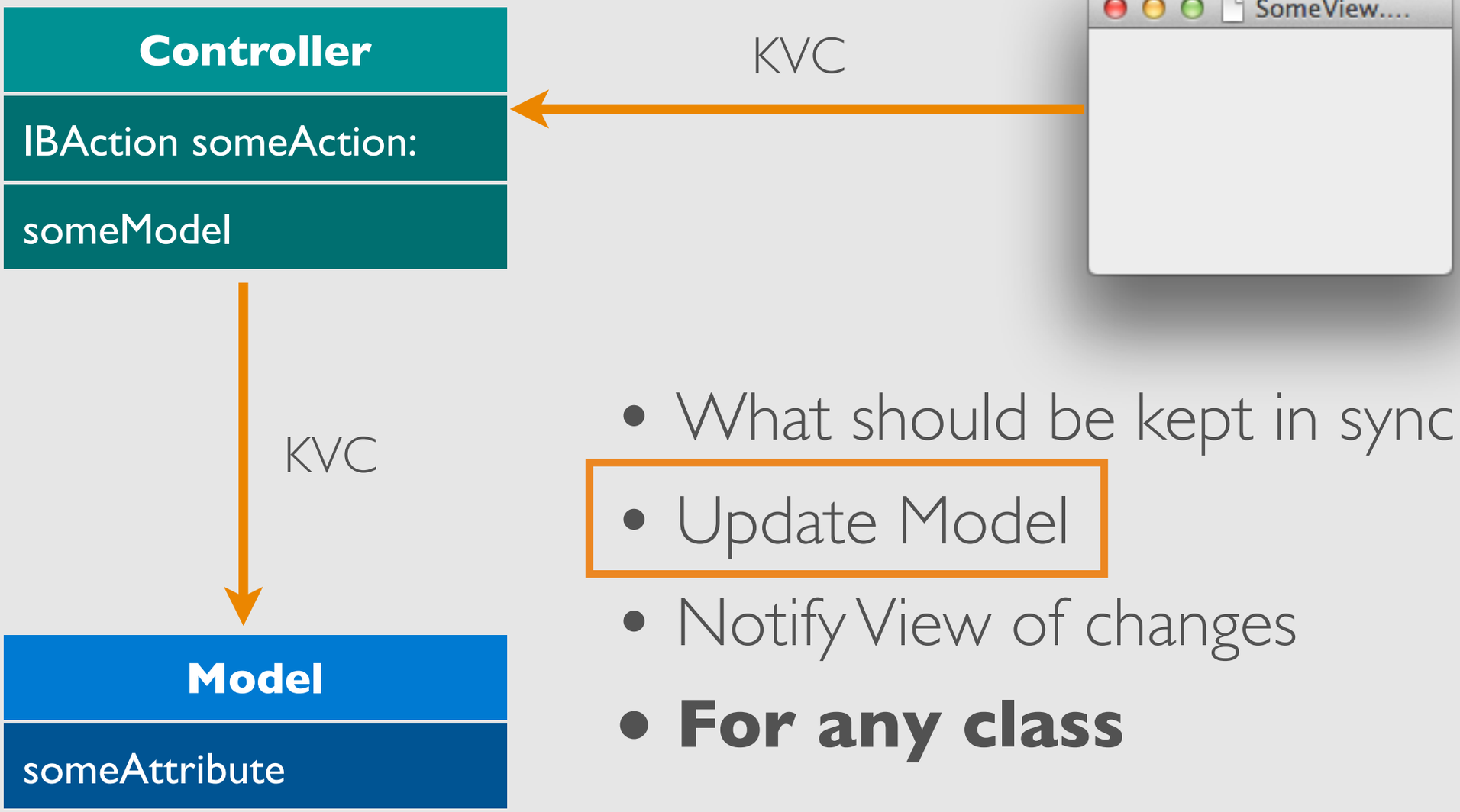
Idee: Abstraktion!

AUTOMATISIERE UPDATES



- What should be kept in sync
- Update Model
- Notify View of changes
- **For any class**

AUTOMATISIERE UPDATES



- What should be kept in sync
- Update Model
- Notify View of changes
- **For any class**

KEY-VALUE CODING

```
-(void)setValue:(id)value forKey:(NSString *)key
-(id)valueForKey:(NSString *)key
```

- If a custom accessor exists (naming convention: `-set<Key>` OR `-get<Key>` or `-<key>` or `-is<Key>`) it will be used
- If not, check for field with name `_<key>`, `_is<Key>`, `<key>` or `is<Key>.` If that exist, read/set it
- If primitive type (bool, int, double, etc.), wrap it in NSNumber
- If struct, wrap it in NSValue
- If not found: call `valueForUndefinedKey:`

KEY-VALUE CODING WITH KEY PATHS

```
-(void)setValue:(id)value forKeyPath:(NSString *)key
-(id)valueForKeyPath:(NSString *)key
```

- Dot-separated path of keys: `@“album.track.volume“`
- Calls `valueForKey:` for each key in front of the last one
- Calls `valueForKey:` / `setValue:forKey:` on the last key

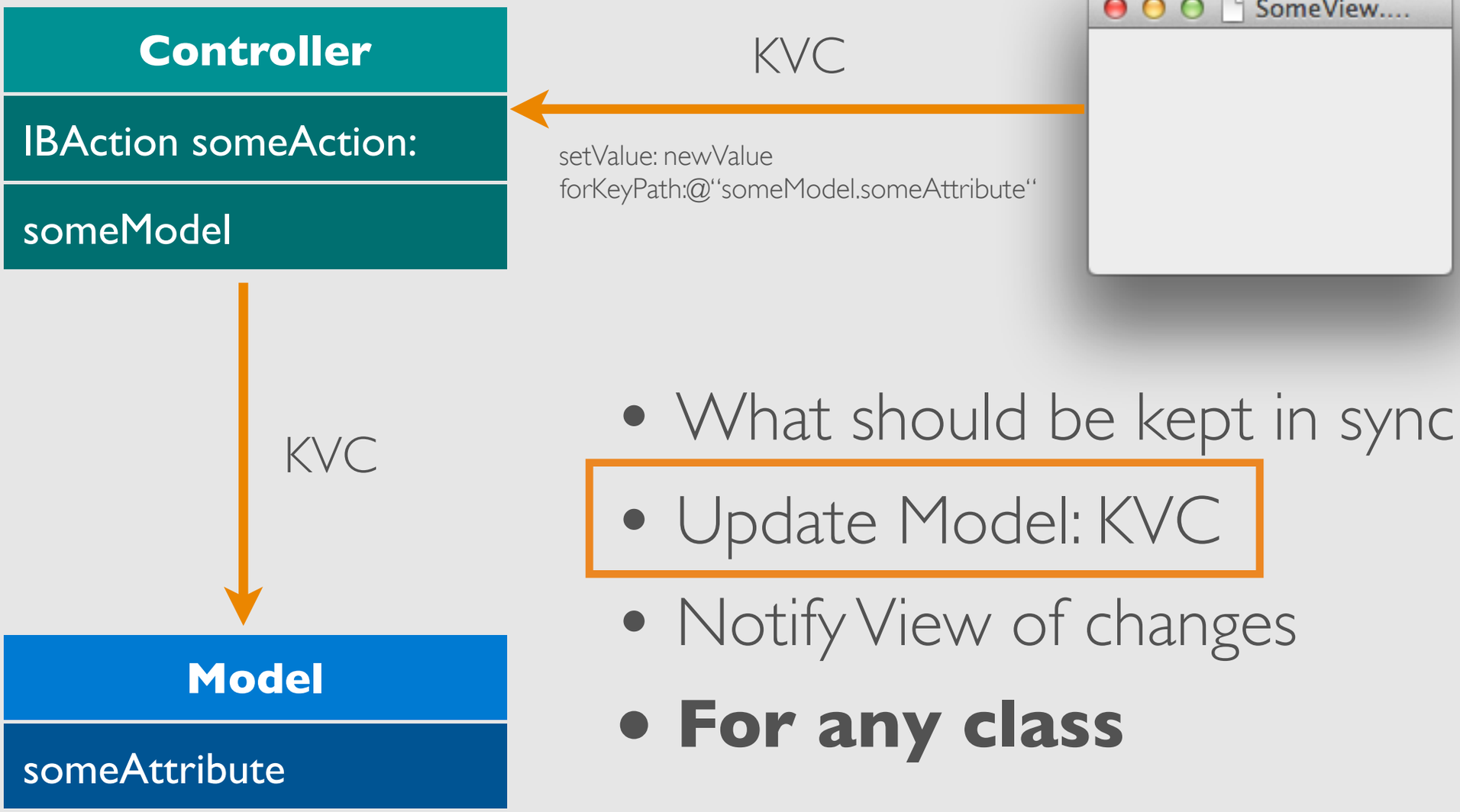
KEY-VALUE CODING COMPLIANCE

- If you follow the Objective-C naming convention your class is most likely KVC-compliant
- Using properties: Definitely

➡ Usually no additional work!

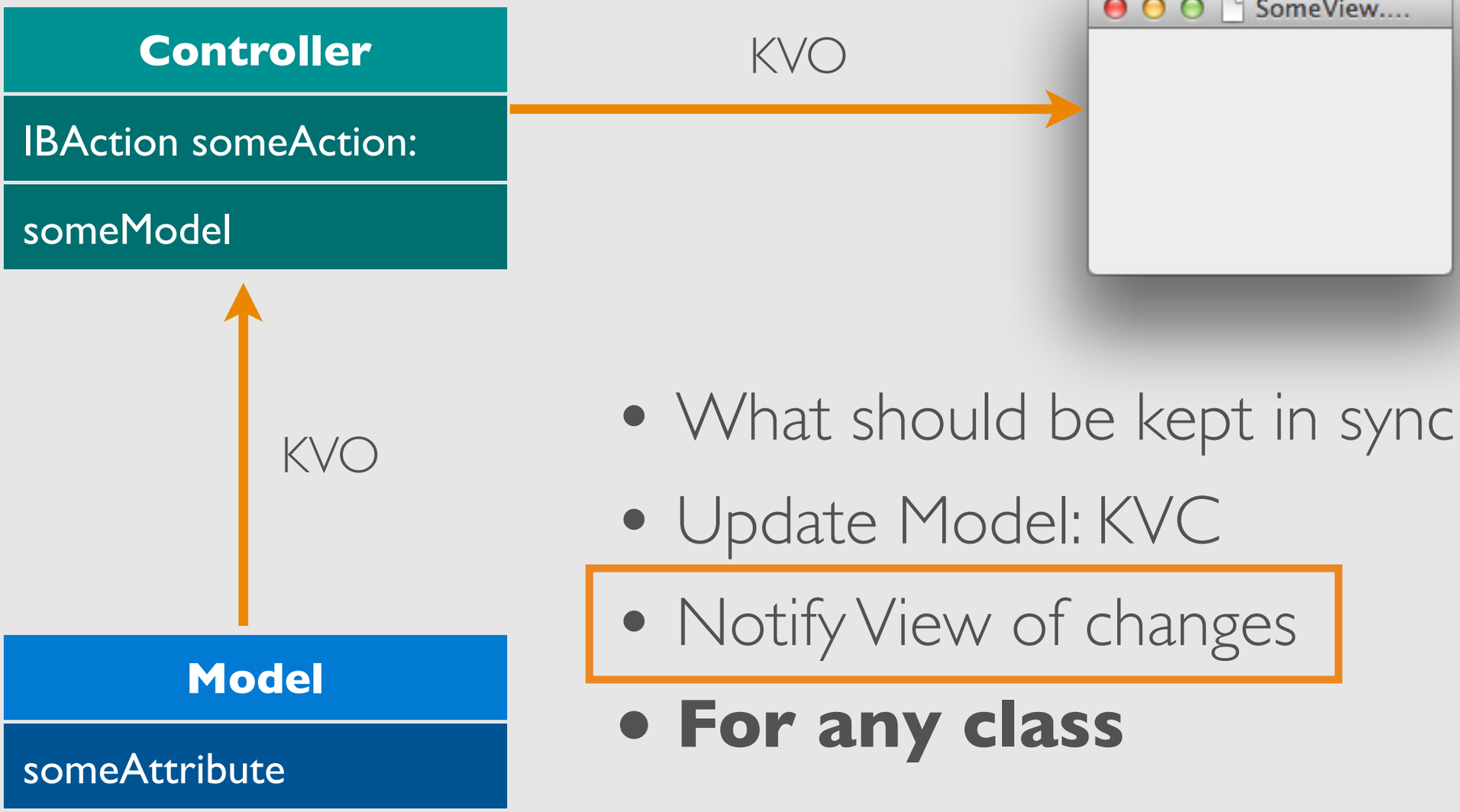
- slightly more difficult for mutable collections (coming up later)

AUTOMATISIERE UPDATES



- What should be kept in sync
- Update Model: KVC
- Notify View of changes
- **For any class**

AUTOMATISIERE UPDATES



- What should be kept in sync
- Update Model: KVC
- Notify View of changes
- **For any class**

KEY-VALUE OBSERVING

```
-(void)addObserver:(NSObject *)observer
        forKeyPath:(NSString *)keyPath
           options:(NSKeyValueObservingOptions)options
           context:(void *)context
```

- Adds an observer for the specified key path

- Whenever the value at the keypath changes the following is called:

```
-(void)observeValueForKeyPath:(NSString *)keyPath
                     ofObject:(id)object
                       change:(NSDictionary *)change
                      context:(void *)context
```

KEY-VALUE OBSERVING COMPLIANCE

- Whenever a KVC-compliant setter is called a notification is sent automatically

- When `setValue:forKey:` is called a notification is sent

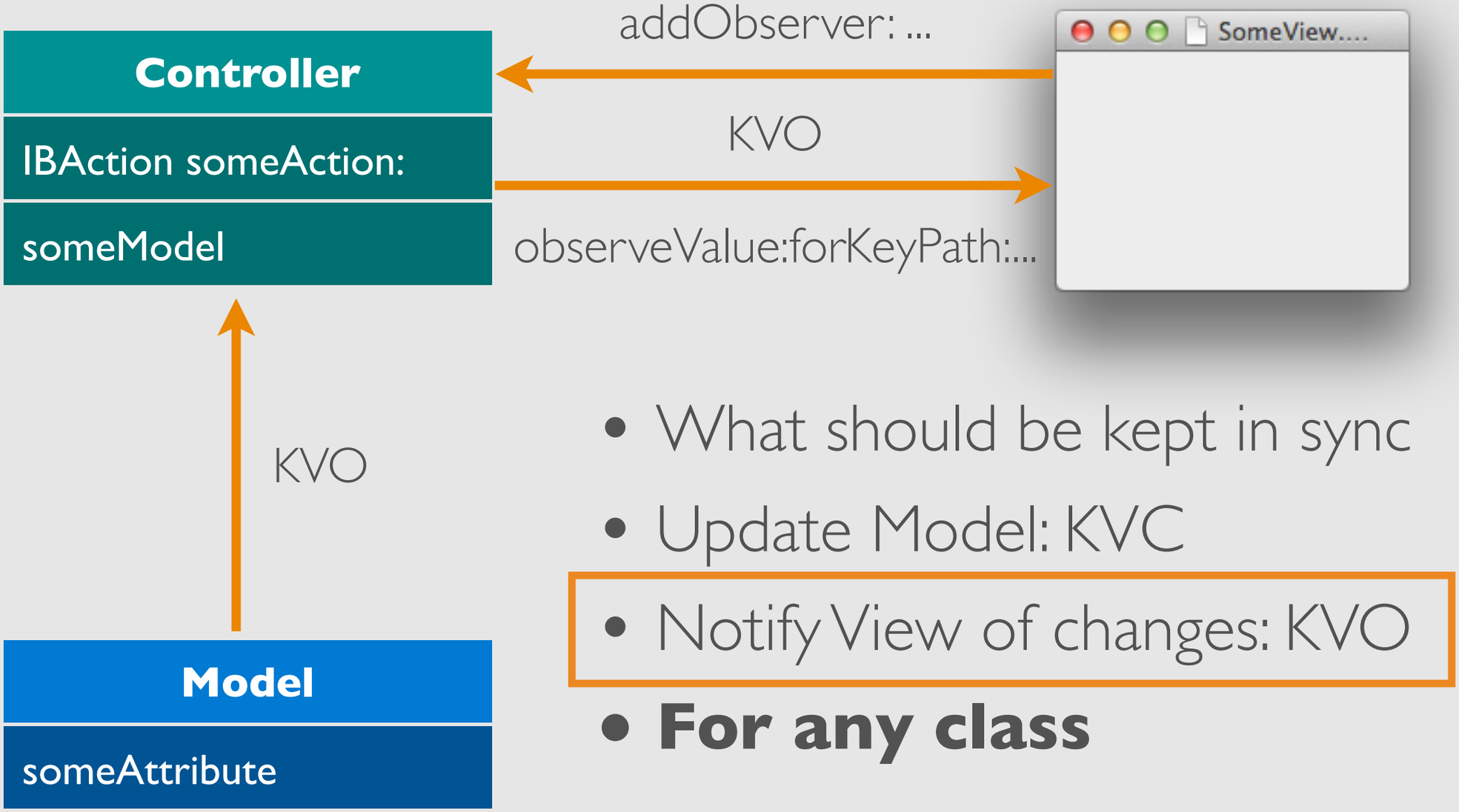
➡ Use naming conventions / @property ⇒ no additional work

➡ Use dot-syntax / setter when changing your object fields

- manual notifications possible

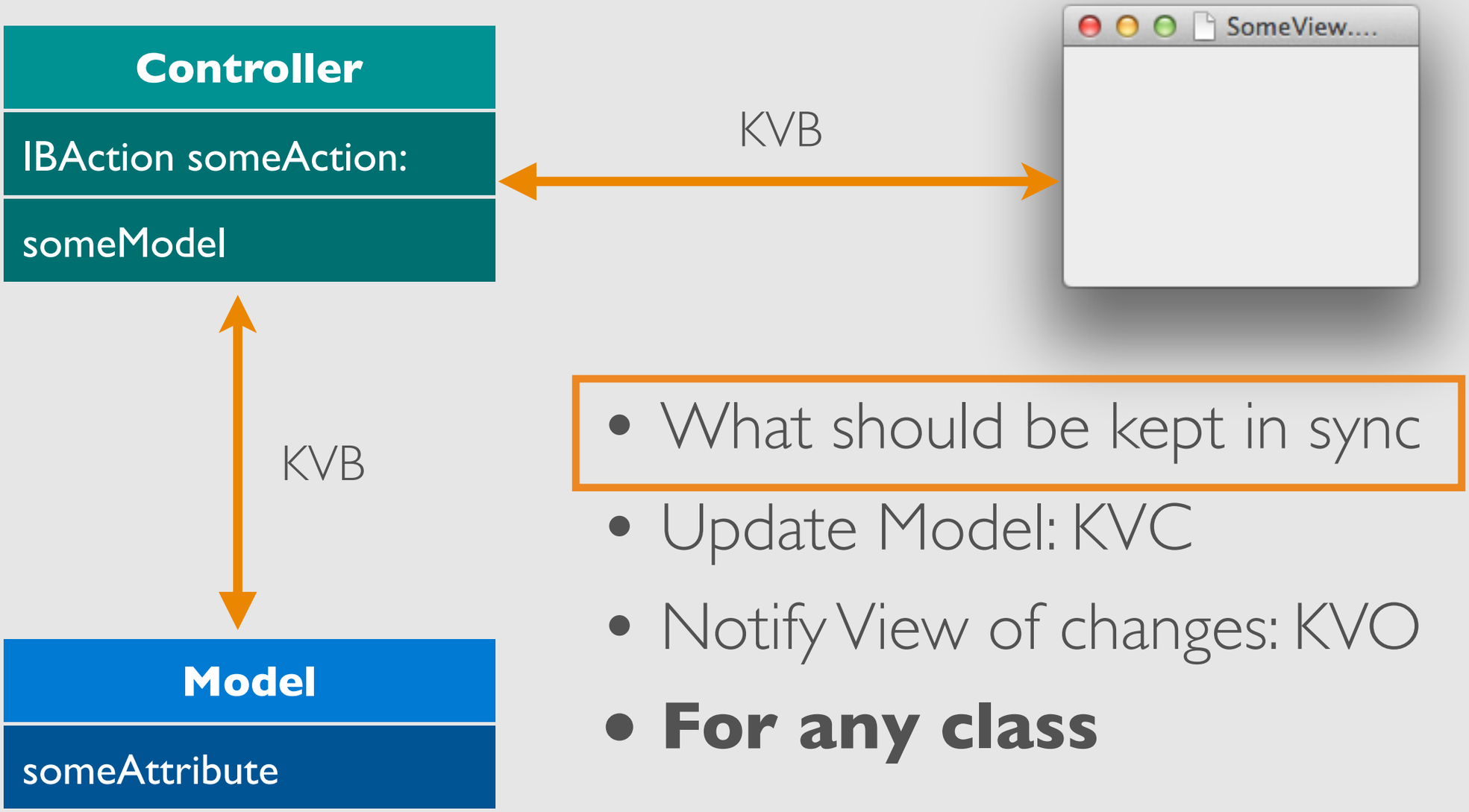
- slightly more difficult for mutable collections (coming up later)

AUTOMATISIERE UPDATES



- What should be kept in sync
- Update Model: KVC
- Notify View of changes: KVO
- **For any class**

AUTOMATISIERE UPDATES



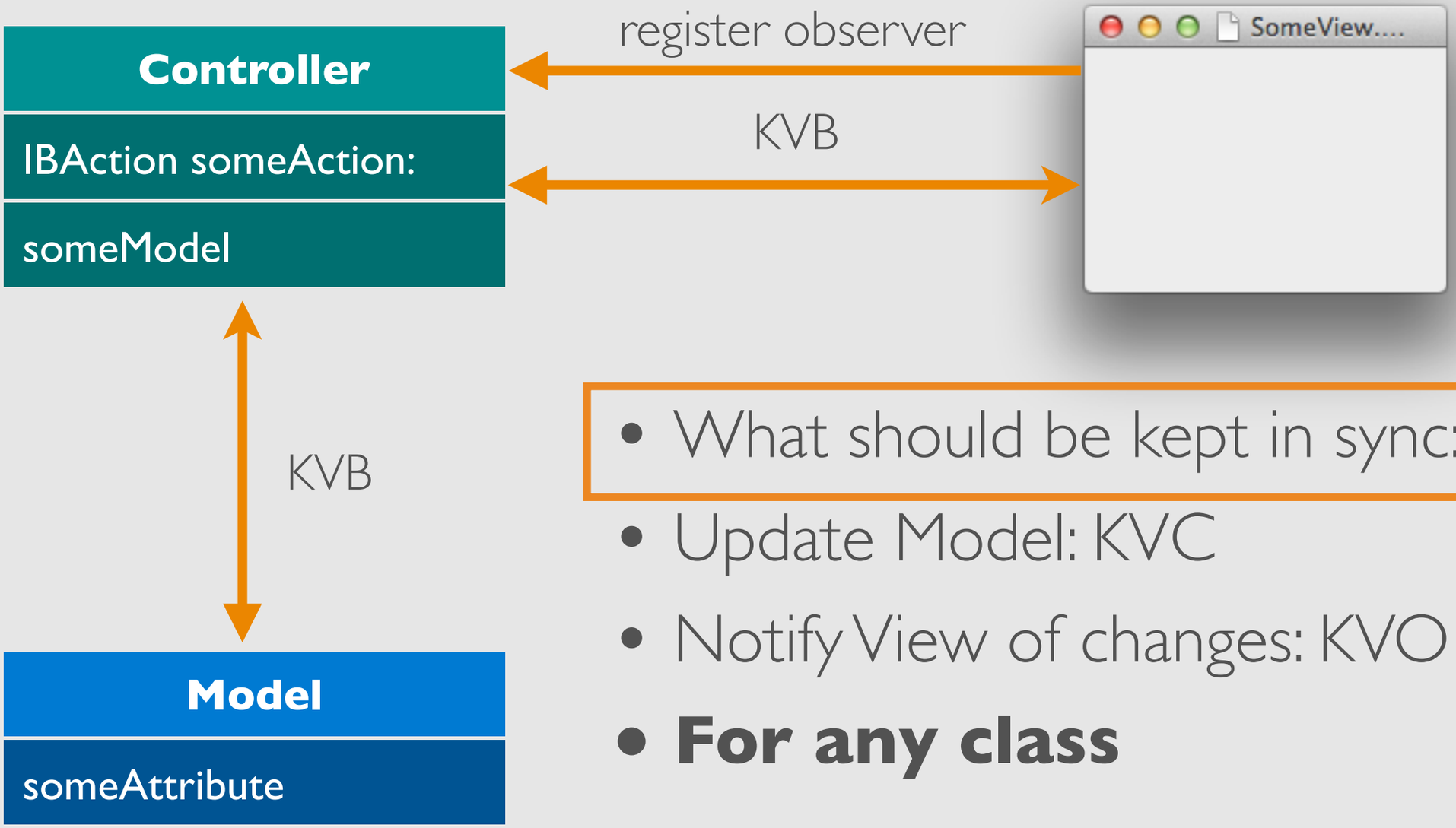
- What should be kept in sync
- Update Model: KVC
- Notify View of changes: KVO
- **For any class**

KEY-VALUE BINDING

```
-(void)bind:(NSString *)binding
   toObject:(id)observable
withKeyPath:(NSString *)keyPath
    options:(NSDictionary *)options
```

- Tells the receiver to observe the `keyPath` of `observable`
- Whenever the value at `binding` changes update the value at the `keyPath` of the `observable`
- Whenever the `keyPath` of `observable` changes, update the value at `binding`

AUTOMATISIERE UPDATES

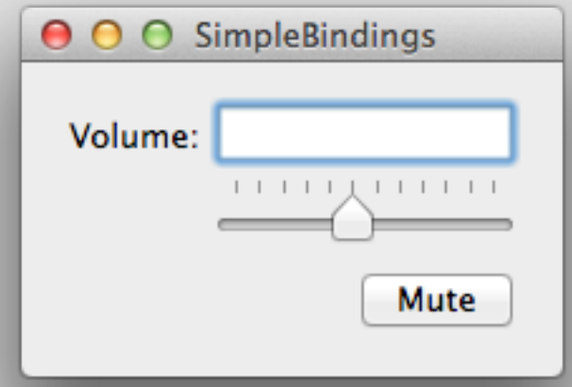


- What should be kept in sync: KVB
- Update Model: KVC
- Notify View of changes: KVO
- **For any class**

DEMO: SIMPLE BINDINGS

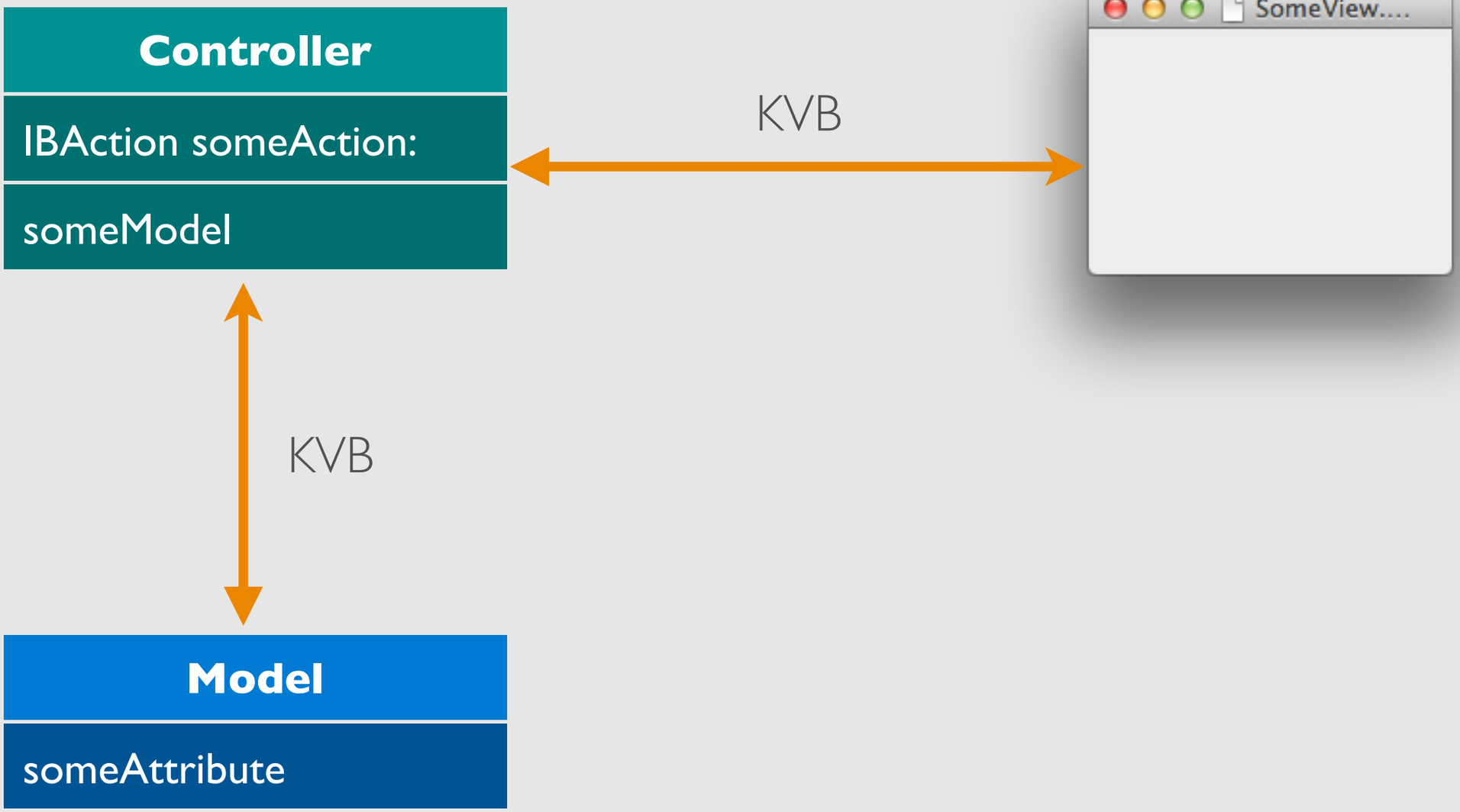
```objc
@interface SBAppDelegate : NSObject [...]

// [...]
@property (assign) IBOutlet NSSlider *slider;
@property (assign) IBOutlet NSTextField *textfield;

@property (retain) SBTrack *currentTrack;

@end
```

```objc
- (void)applicationDidFinishLaunching:[...]
{
    self.currentTrack = [[SBTrack alloc] init];
}
-(IBAction)muteTrack:(id)sender
{
    self.currentTrack.volume = 0;
}
```

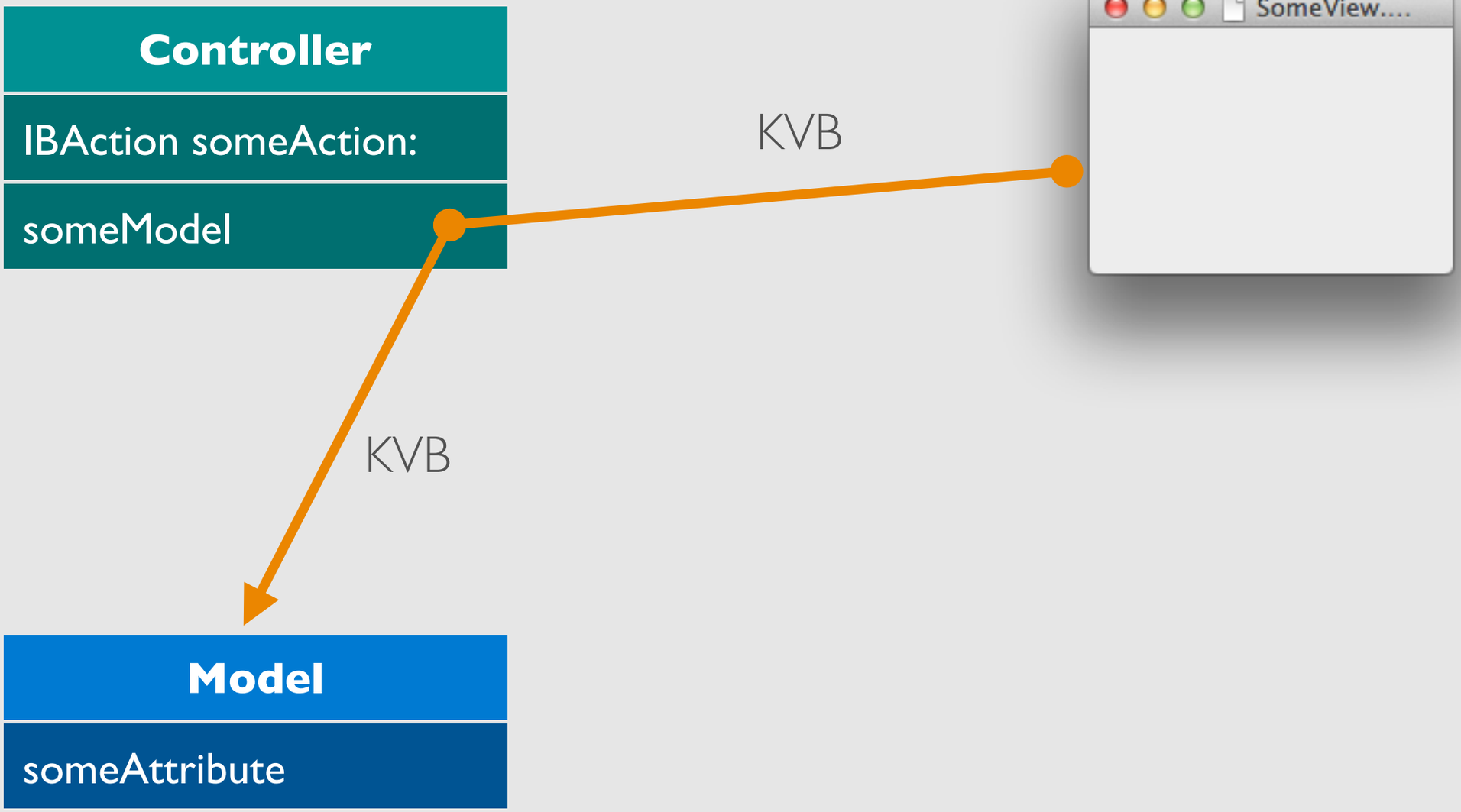



```objc
@interface SBTrack : NSObject

@property (assign) double volume;

@end
```

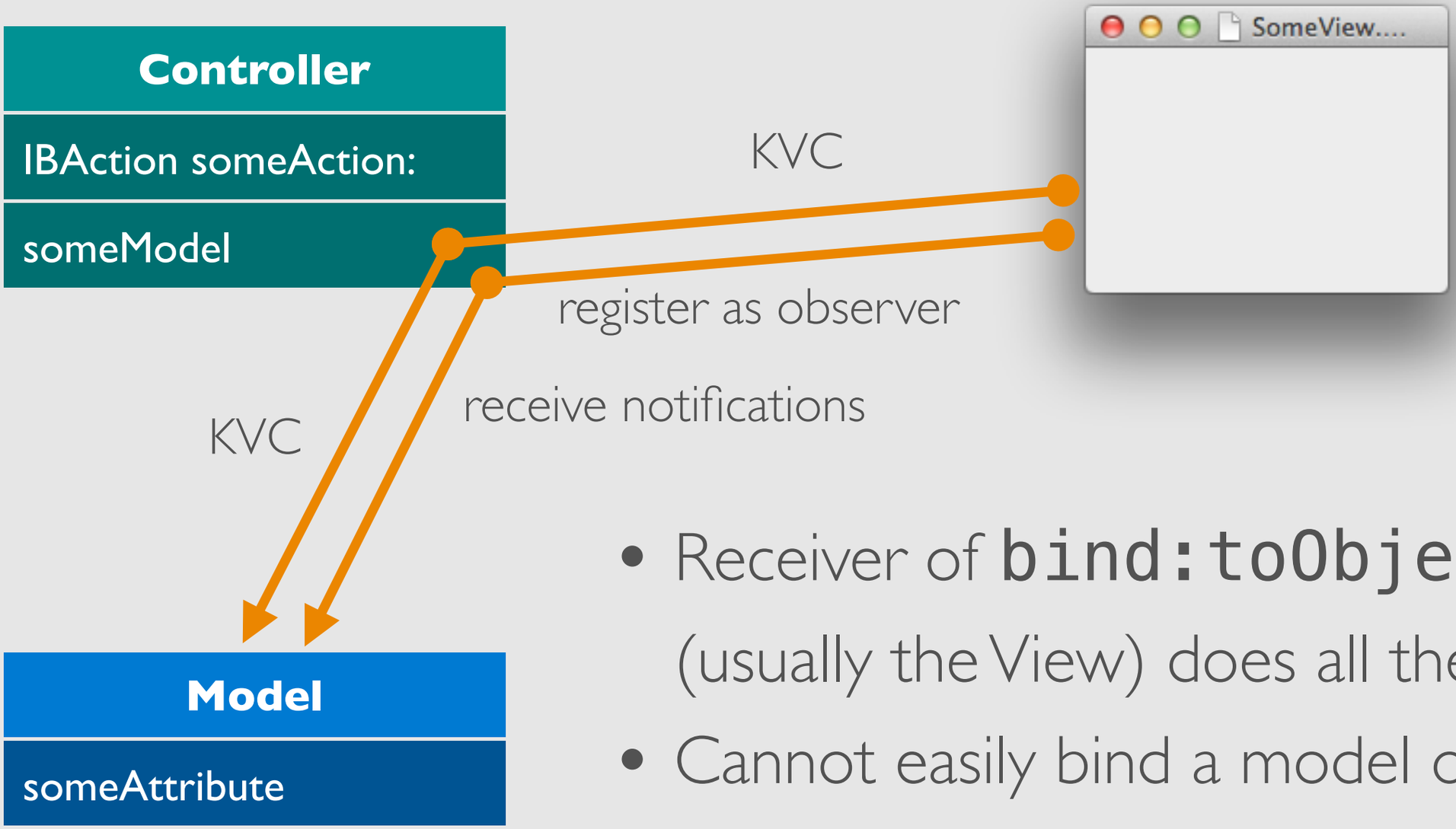
WARNING: BINDINGS ARE NOT SYMMETRIC!



WARNING: BINDINGS ARE NOT SYMMETRIC!



WARNING: BINDINGS ARE NOT SYMMETRIC!

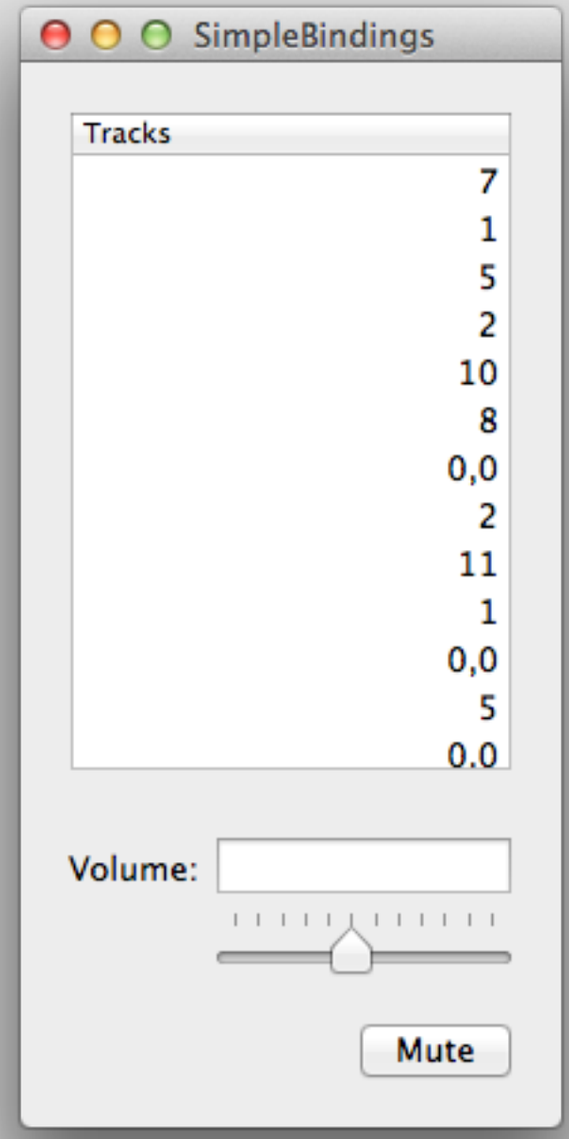


- Receiver of `bind:toObject:`… (usually the View) does all the work
- Cannot easily bind a model object to another model object

DEMO: ARRAY BINDINGS

```
@interface SBAppDelegate : NSObject [...]

// [...]
@property (retain) NSArray *tracks;

@end
```

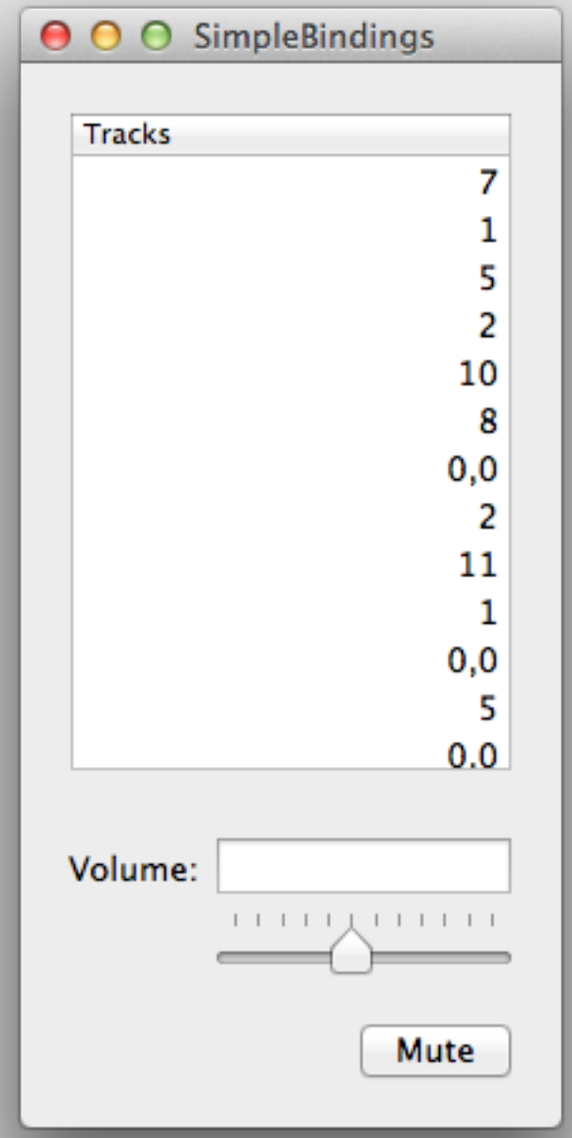
```
- (void)applicationDidFinishLaunching:[...]
{
    // initialize tracks with 100 random ones
}
```



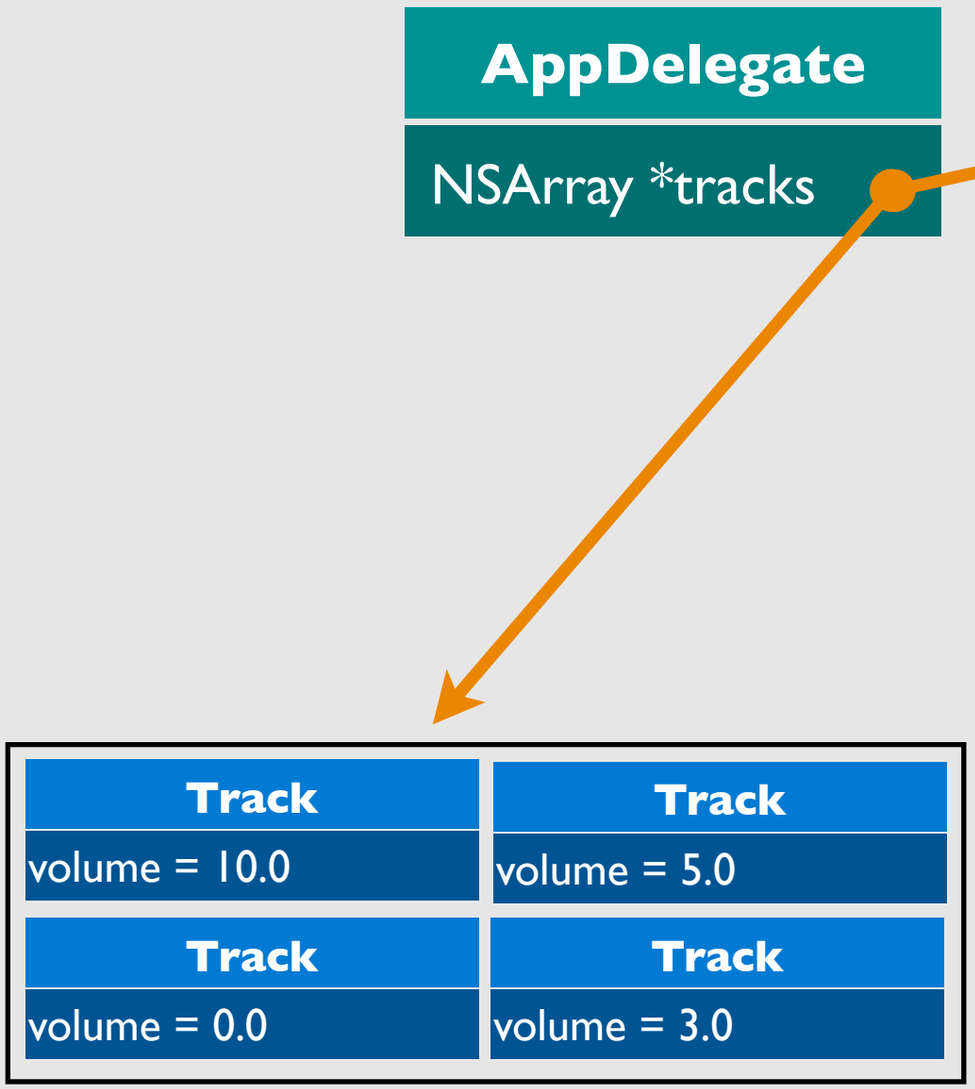

```
@interface SBTrack : NSObject

@property (assign) double volume;

@end
```

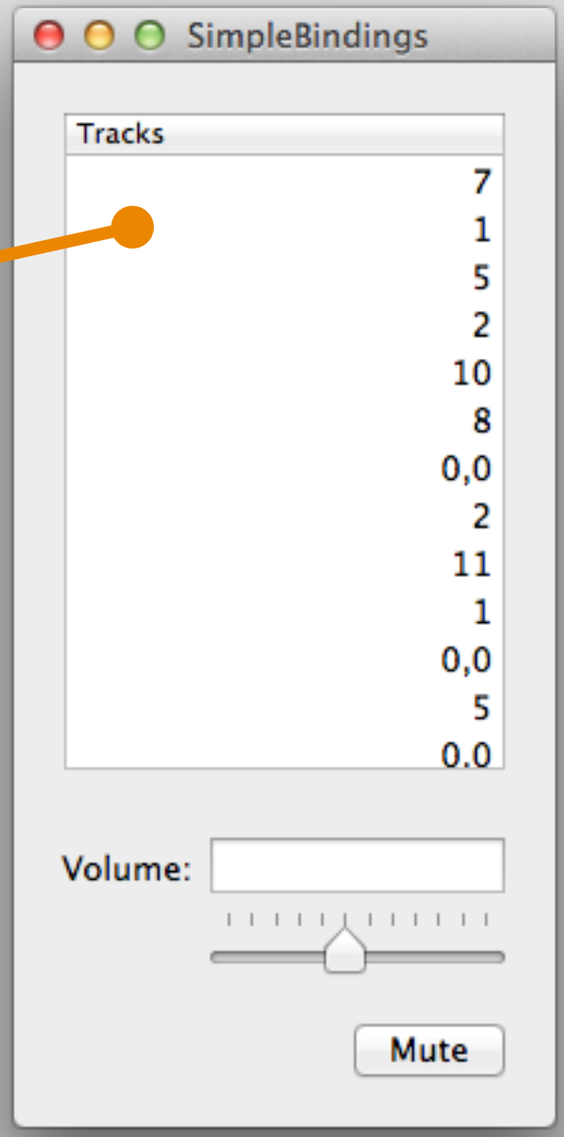
PROBLEM: HOW TO BIND TO CURRENT SELECTION?

- Could implement TableView's delegate methods and set a currentTrack property
  - would work, but boring code: Already solved
- Solution: `NSObjectController`

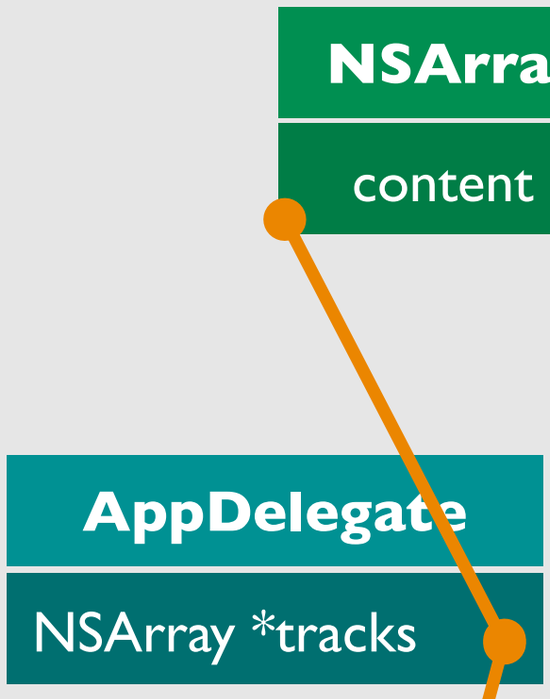


NSObjectController

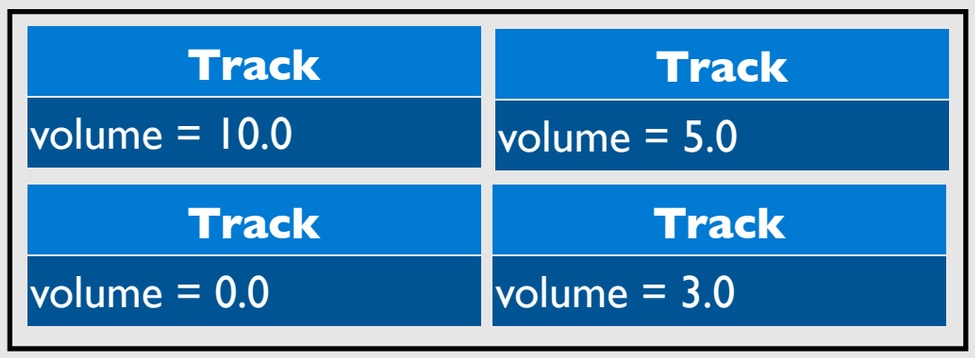


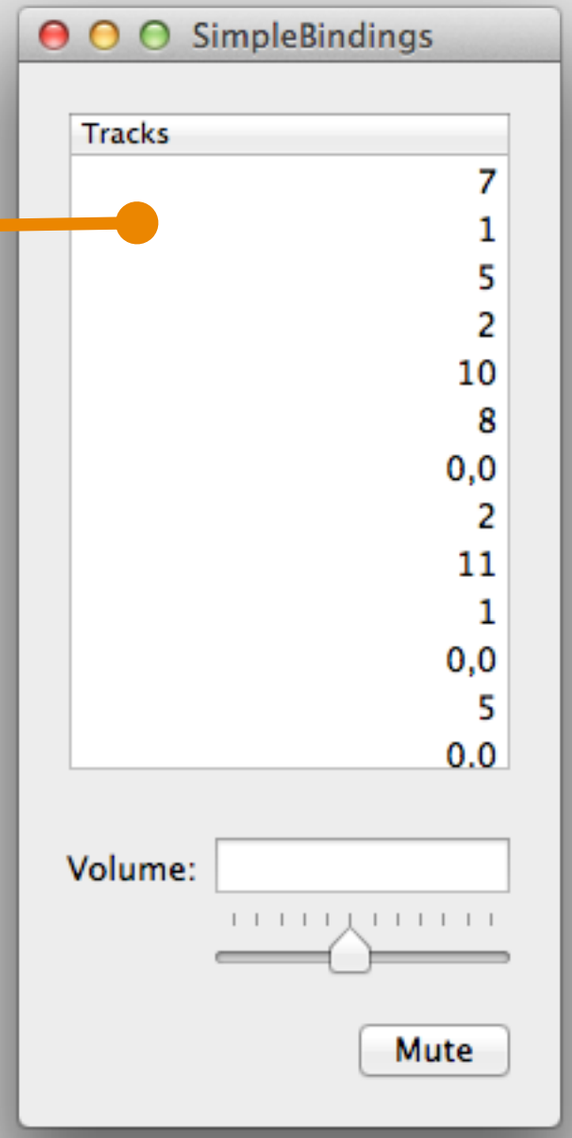


NSObjectController

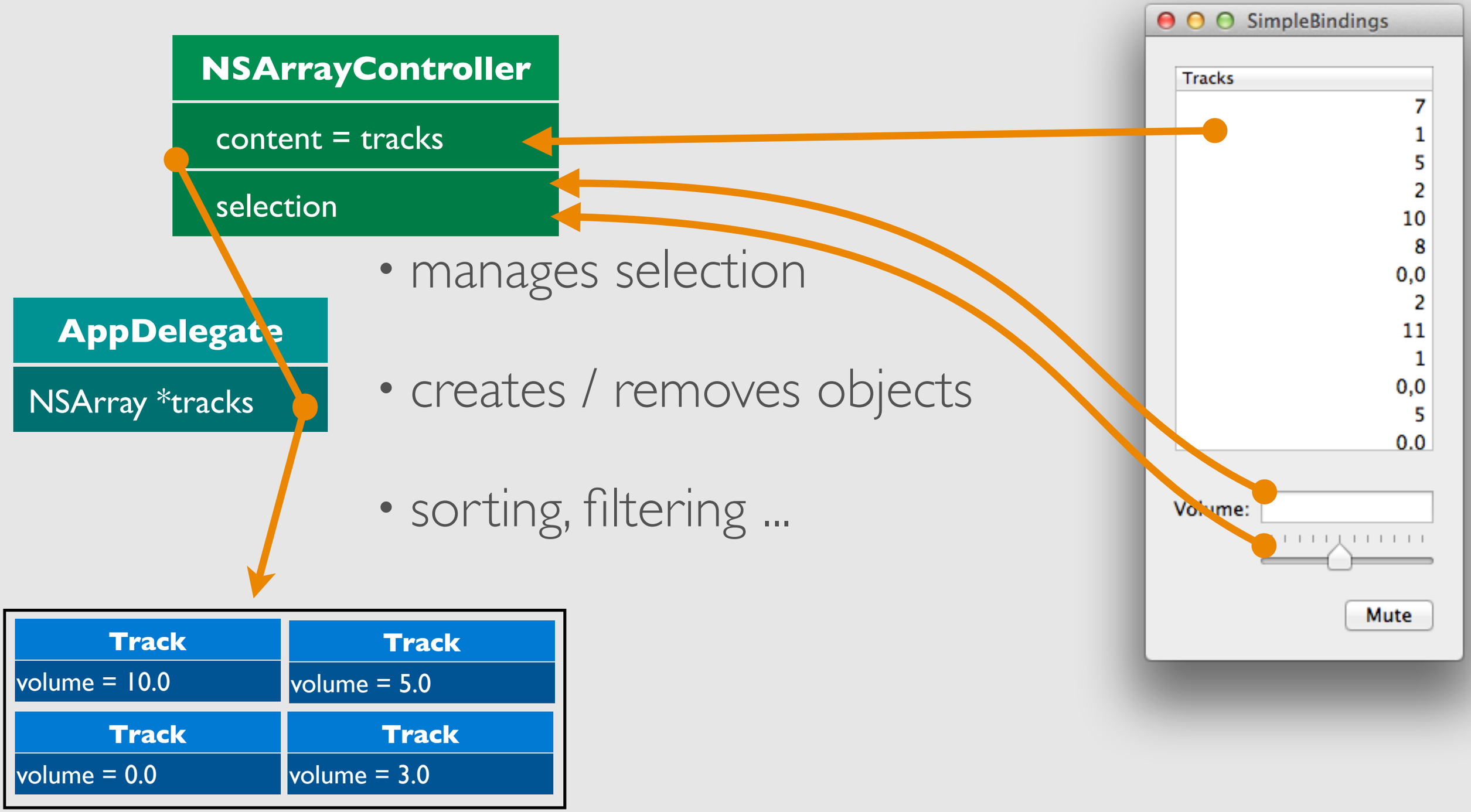


- manages selection
- creates / removes objects
- sorting, filtering ...



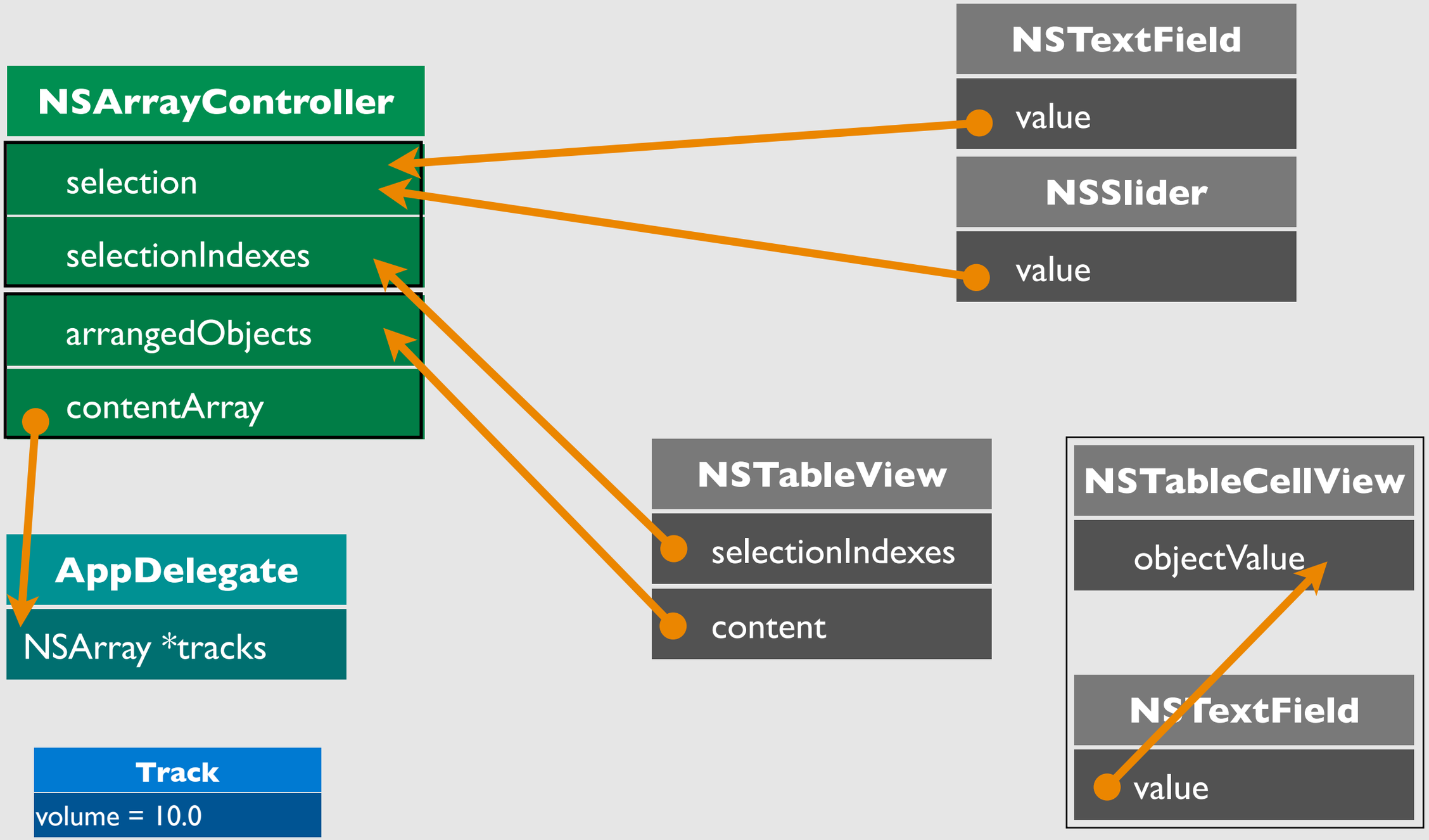


NSObjectController



- manages selection
- creates / removes objects
- sorting, filtering ...

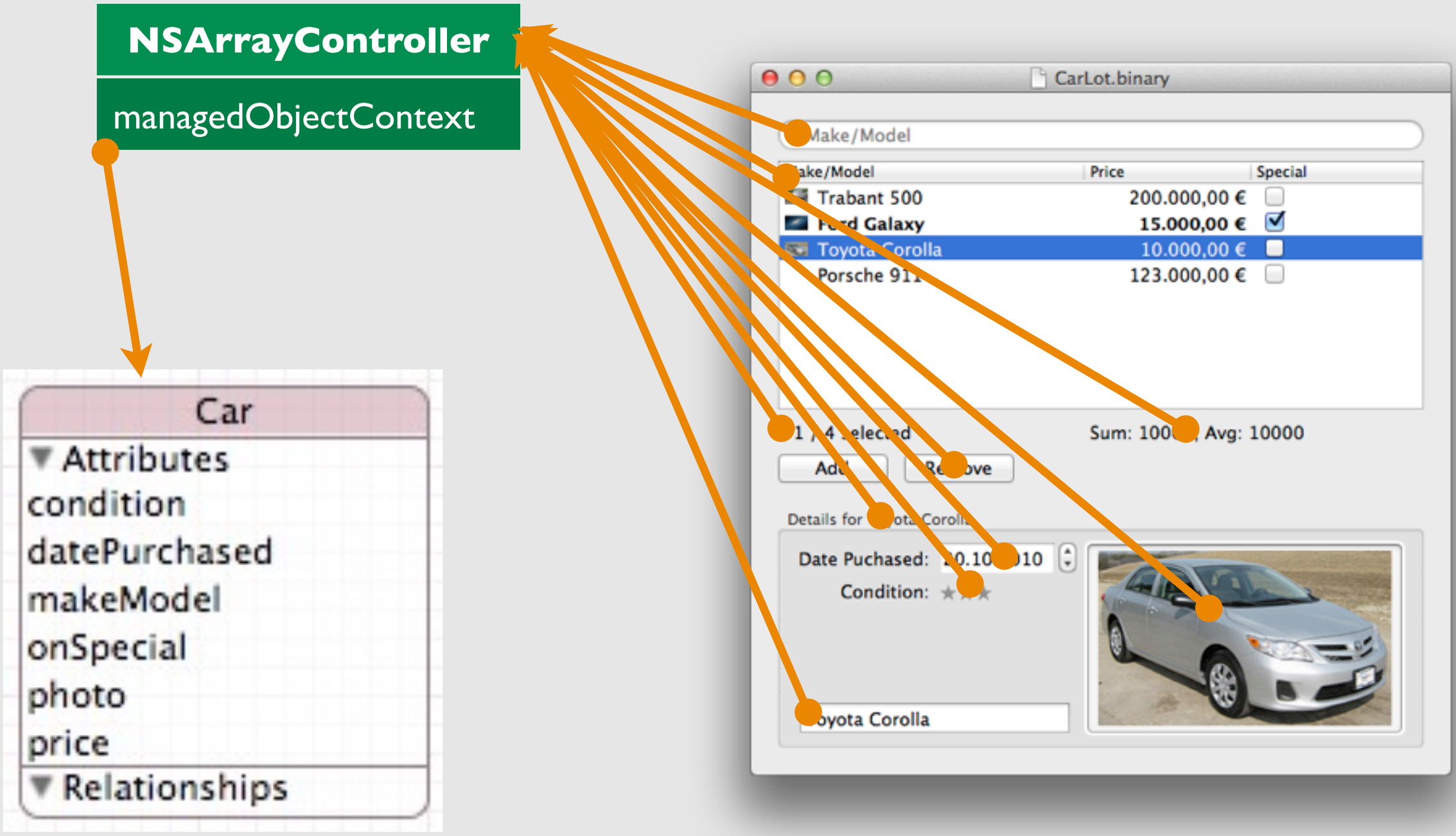
DEMO: SELECTION BINDING



CoreData And Bindings

- Use CoreData to create Model classes and IB for View
  - ➡ Avoid writing code altogether!
- CoreData fully KVC and KVO compliant
- NSObjectController (NSArrayController, NSTreeController) can just use `managedObjectContext` instead of `content` binding

Demo: Bindings And CoreData



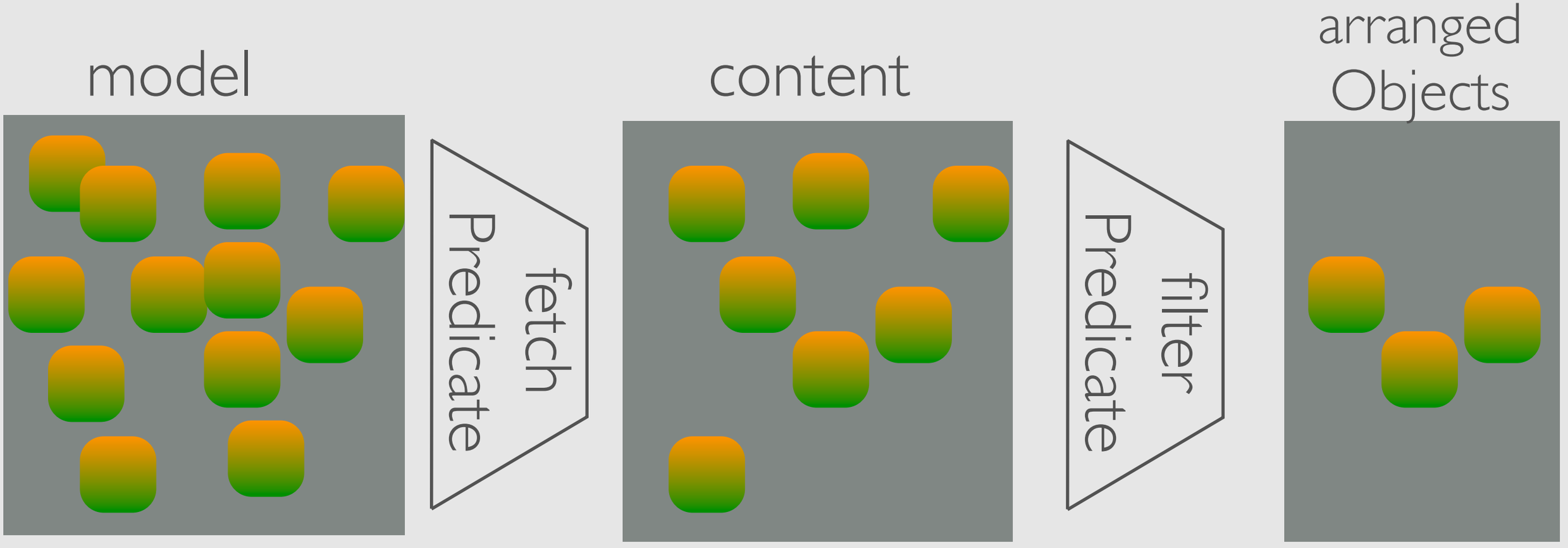
NSArrayController Is Great!

- -(IBAction)add:(id)sender
- -(IBAction)remove:(id)sender
- bindings for
  - canRemove
  - selection
  - sorting
- There is also NSTreeController, NSUserDefaultsController, NSDictionaryController
  - All subclasses of NSObjectController

More NSArrayController Fun

- fetchPredicate: to limit the content property of the controller (fetches less)

- filterPredicate: to limit the arrangedObjects (filter content)



NSValueTransformer And NSFormatter

- NSNumberFormatter, NSDateFormatter to format numbers and dates nicely
- NSValueTransformer if values of bindings need some massaging
  - implement
    `-(id)transformedValue:(id)value` and
    `-(id)reverseTransformedValue:(id)value`

COLLECTION OPERATORS

`keypathToCollection.@collectionOperator.keyPathToProperty`

- @count: Count of collection. e.g. `@“selection.count“`
- @avg: Average of properties in collection. e.g.
  `@“arrangedObjects.@avg.price“`
- @sum, @min, @max
- @distinctUnionOfObjects, @unionOfObjects
- @distinctUnionOfArrays,
  @unionOfArrays,
  @distinctUnionOfSets

DEMO:
COLLECTION OPERATORS

Bindings

Joachim Kurz

ENUMERATED BINDINGS

- Some binding properties can be bound to more than one value
  - enabled
  - hidden
  - display pattern
- Semantics depend on view
  - enabled: AND
  - hidden: OR

KVC, KVO & TO-MANY RELATIONSHIPS

- reading: implement one of the following
  - `-<key>`
  - `-countOf<Key>` and
    - `-objectIn<Key>AtIndex` or `-<key>AtIndexes:`
- modifying: implement
  - `-insertObject:in<Key>AtIndex:` or `-insert<Key>:atIndexes:`
  - `-removeObjectFrom<Key>AtIndex:` or `-remove<Key>AtIndexes:`
- similar for unordered to-many-relationships (see Documentation)

SUMMARY

- Bindings: Great!
- CoreData: Great!
- AutoLayout: Great!
- Combination: Awesome!

Twitter: @cocoafrog