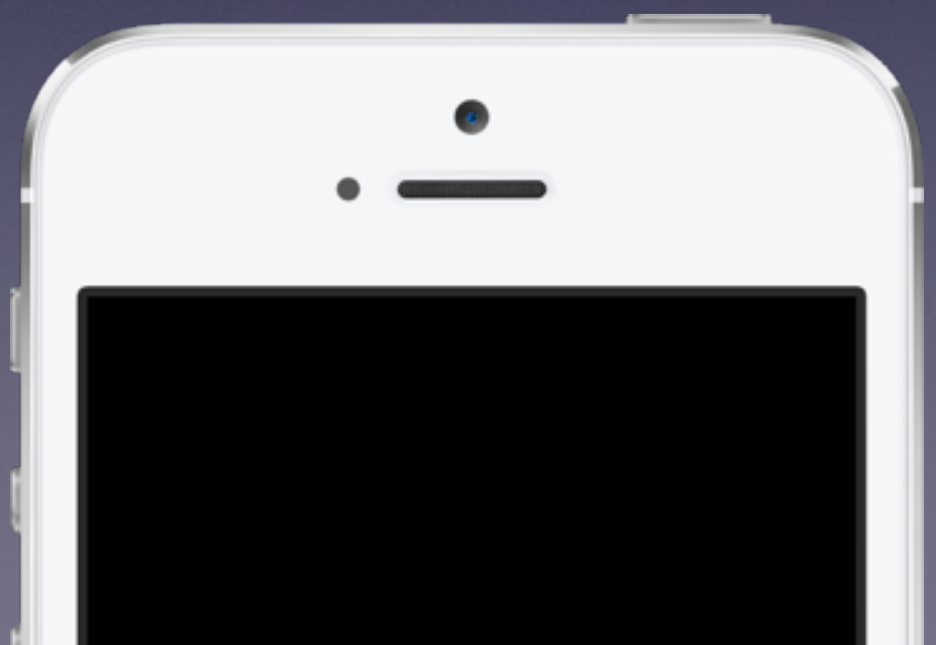
# COMS W3101: Programming for iOS

Michael Vitrano



# Course Goals

- Walk away with the toolset to build real world apps
- Focus on a foundation in Objective-C and core iOS concepts and frameworks
- Exposure to real-world problem domains and app categories
- A resource for transitioning from academic coding to professional coding

# Course Prerequisites

- Fluency in at least one programming language
- Strong understanding of Object-Oriented programming
- Familiarity with Model-View-Controller architecture

# Course Structure

- Each class is going to be a split between lecture and a live coding demo
- Both of these will be posted to the course Github page
  - http://columbia-w3101-ios.github.io/
- Each class will cover a specific building block for creating an iOS app

# Grading

- There will be one course-long project in the class
- We're going to build a note taking app
  - There will be a set of features that your app must have
  - Over the course of the semester you'll gain the skills you need to implement those features
- We'll talk more about the specifics next class

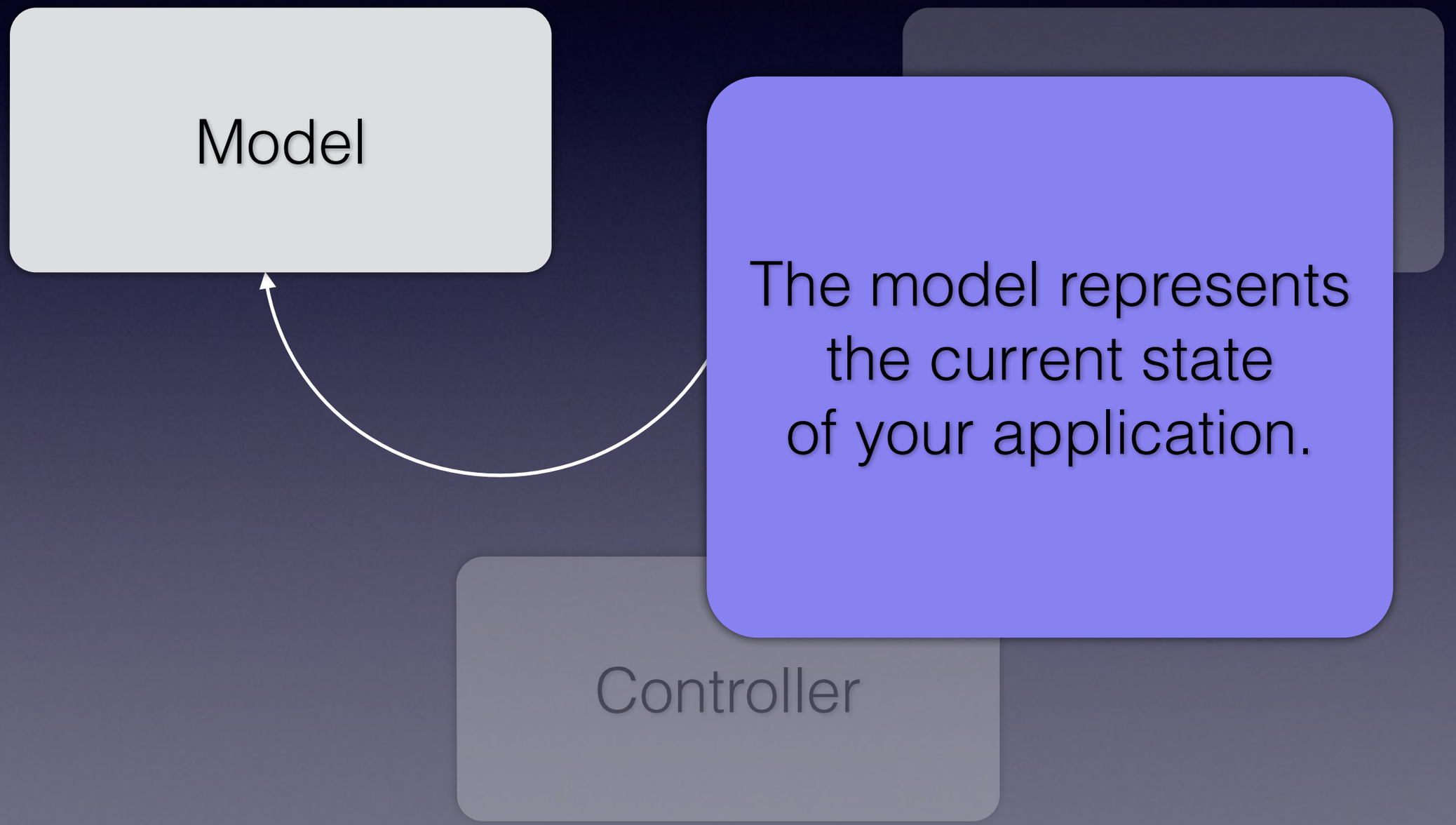
# Model-View-Controller

Model

View

Controller

# Model-View-Controller



# Model-View-Controller

The view is how your application displays its state to the user.

View

Controller

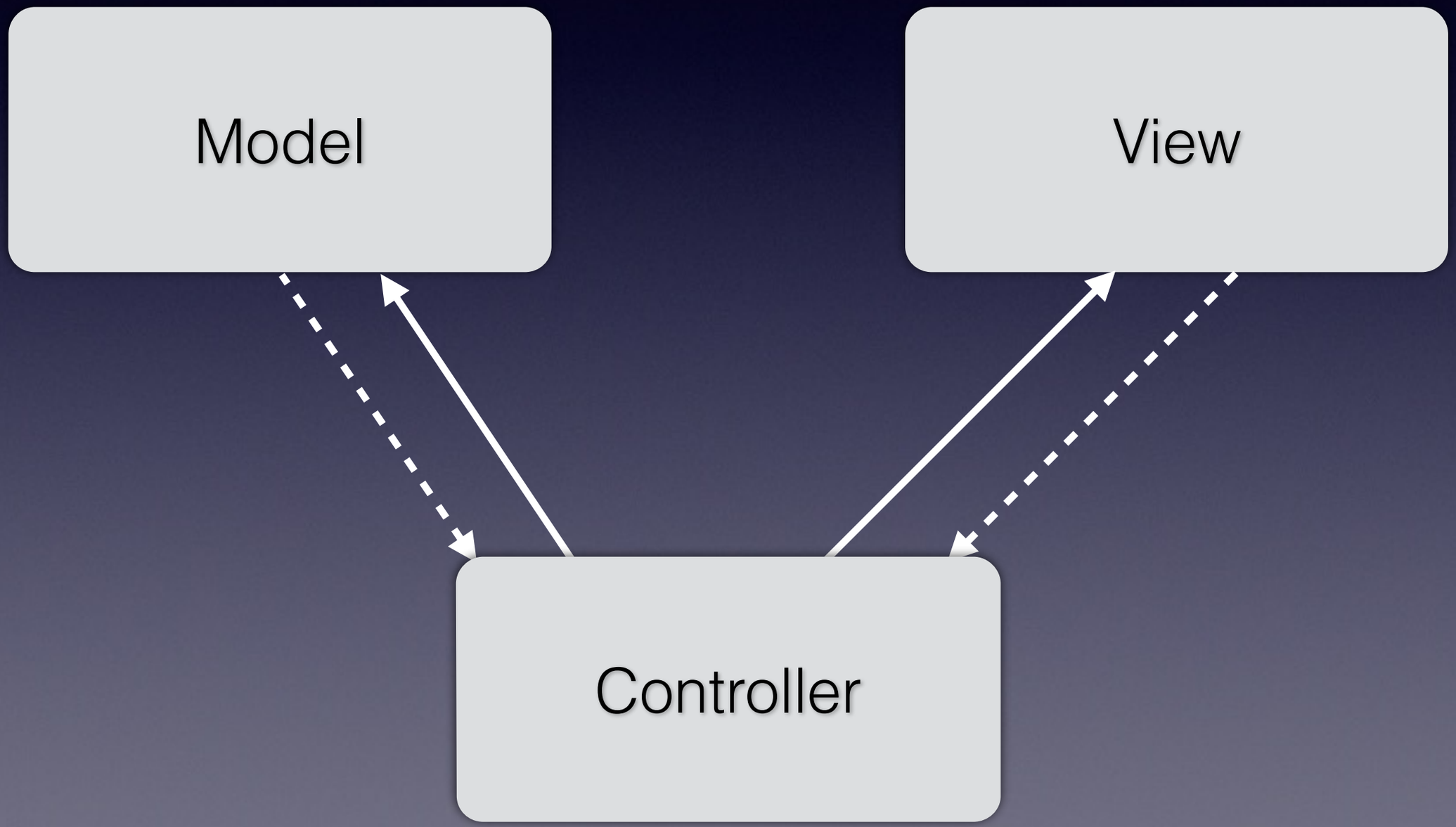
# Model-View-Controller

Model

The controller mediates the communication between the model and the view.

Controller

# Model-View-Controller



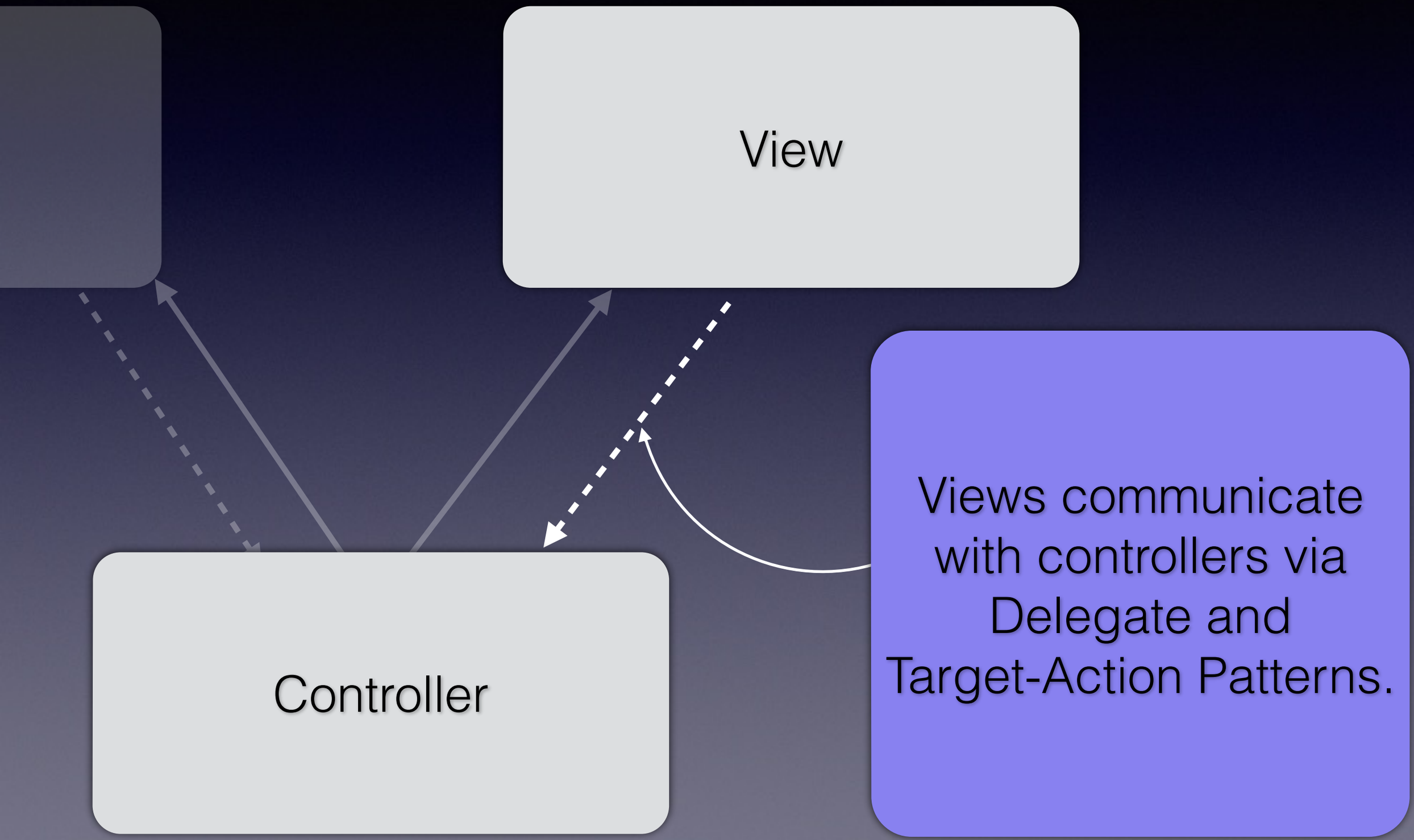
# Model-View-Controller

Controllers need to have direct knowledge about the view.

View

Controller

# Model-View-Controller



# Model-View-Controller

Model

Controllers must have direct knowledge about the model.

Vie

Controller

# Model-View-Controller

Model

Vie

Models broadcast information about changes with NSNotificationCenter.

Controller

MSVPowerPlant.h

MSVPowerPlant.h › No Selection

```
//
//  MSVPowerPlant.h
//
//

#import <Foundation/Foundation.h>

@interface MSVPowerPlant : NSObject

/*
 This is the basic declaration of a
 class called MSVPowerPlant. It
 inherits from NSObject. Objects are
 defined in header files (.h)
*/

@end
```

Counterparts › MSVPowerPlant.m › @implementation MSVPowerPlant

```
//
//  MSVPowerPlant.m
//
//

#import "MSVPowerPlant.h"

@implementation MSVPowerPlant

/*
 For every class defined in a header
 there is also its implementation
 in an implementation file (.m)
*/

@end
```

MSVPowerPlant.h

MSVPowerPlant.h › No Selection

```
//
//  MSVPower
//
//

#import <Foundation/Foundation.h>

@interface MSVPowerPlant : NSObject



@end
```

Superclass Header

Class Name

Superclass

Counterparts › MSVPowerPlant.m › @implementation MSVPowerPlant

```
//
//  MSVPo
//
//

#import "MSVPowerPlant.h"

@implementation MSVPowerPlant



@end
```

Defining Header

MSVPowerPlant.h

MSVPowerPlant.h › No Selection

```
//
//  MSVPowerPlant.h
//

#import <Foundation/Foundation.h>

@interface MSVPowerPlant : NSObject

/*
 Your class's public interface is
 defined in the header file. This file
 is included in class files that
 use this class.
*/

@end
```

Counterparts › MSVPowerPlant.m › No Selection

```
//
//  MSVPowerPlant.m
//

#import "MSVPowerPlant.h"

@interface MSVPowerPlant ()

/*
 Your class's internal interface is
 declared here inside the
 implementation file in a class
 extension.
*/

@end

@implementation MSVPowerPlant

/*
 The implementation of your public and
 private methods go here in the
 implementation file.
*/

@end
```

MSVPowerPlant.h

MSVPowerPlant.h › No Selection

```
//
//  MSVPowerPlant.h
//

#import <Foundation/Foundation.h>

/*
 We need to let the compiler know that
 a class named MSVCity exists so that
 we can reference it in our method.
*/
@class MSVCity;

@interface MSVPowerPlant : NSObject

// Define a method like so:
- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts;

/*
 This method takes a pointer to acity object
 and a double representing an amount of power
 in watts. It returns void.
*/

@end
```

Counterparts › MSVPowerPlant.m › No Selection

```
//
//  MSVPowerPlant.m
//

#import "MSVPowerPlant.h"
/*
 We import MSVCity's header to use the
 methods defined in its interface in this
 file.
*/
#import "MSVCity.h"

@interface MSVPowerPlant ()
@end

@implementation MSVPowerPlant

// Implement sendPowerToCity:amount:
- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts
{
// Call the recievePower: method on city
    [city recievePower:powerInWatts]
}

@end
```

MSVPowerPlant.h

MSVPowerPlant.h › @interface MSVPowerPlant

```objc
//
//  MSVPowerPlant.h

#import <Foundation/Foundation.h>
@class MSVCity;

@interface MSVPowerPlant : NSObject

/*
 We define storage for an object using
 properties. By declaring a propery, the
 compiler automatically generates accessor
 and setter methods, as well as a backing
 instance variable for currentPower.
*/

@property (nonatomic) double currentPower;

- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts;

@end
```

Counterparts › MSVPowerPlant.m › -sendPowerToCity:amount:

```objc
//
//  MSVPowerPlant.m

#import "MSVPowerPlant.h"
#import "MSVCity.h"

@interface MSVPowerPlant ()
@end

@implementation MSVPowerPlant

- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts
{
// The generated accessor method has the
// same name as the declared property.
    double currPower = [self currentPower];

    if (currPower >= powerInWatts) {
        double newPower =
            currPower - powerInWatts;

// The generated setter method is the
// capitalized name of the property prefixed
// by "set"
        [self setCurrentPower:newPower];

        [city recievePower:powerInWatts]
    }
}
@end
```

MSVPowerPlant.h — Edited

MSVPowerPlant.h › currentPower

```objc
//
//  MSVPowerPlant.h

#import <Foundation/Foundation.h>
@class MSVCity;

@interface MSVPowerPlant : NSObject

/*
 Some properties, such as currentPower,
 should not be writable by outsiders. We
 restrict the writability of a property using
 the "readonly" attribute. Now, only the
 accessor method is defined in the header.
*/

@property (nonatomic, readonly) double currentPower;

- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts;

@end
```

Counterparts › MSVPowerPlant.m › @interface MSVPowerPlant()

```objc
//
//  MSVPowerPlant.m

#import "MSVPowerPlant.h"
#import "MSVCity.h"

@interface MSVPowerPlant ()

// currentPower still needs to be writable for
// the internal bookeeping of the object. We
// accomplish this by redefining the property as
// writable inside the class extension

@property (nonatomic) double currentPower;

@end

@implementation MSVPowerPlant

- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts
{
    double currPower = [self currentPower];

    if (currPower >= powerInWatts) {
        double newPower =
            currPower - powerInWatts;

// The setter method is still available internally
// to the class's implementation
        [self setCurrentPower:newPower];

        [city recievePower:powerInWatts]
    }
}

@end
```

MSVPowerPlant.h — Edited

MSVPowerPlant.h › @interface MSVPowerPlant

```objc
//
//  MSVPowerPlant.h

#import <Foundation/Foundation.h>
@class MSVCity;

@interface MSVPowerPlant : NSObject

@property (nonatomic, readonly) double currentPower;

- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts;

@end
```

Counterparts › MSVPowerPlant.m › No Selection

```objc
//
//  MSVPowerPlant.m

#import "MSVPowerPlant.h"
#import "MSVCity.h"

@interface MSVPowerPlant ()
@property (nonatomic) double currentPower;
@end

@implementation MSVPowerPlant

- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts
{
// Properties are so important to the language that
// there is special syntax for calling the accessor
// and setter methods of an object.

// Using dot-syntax, we can call the currentPower
// accessor method. This is equivalent to calling
// currentPower using bracket syntax.
    double currPower = self.currentPower;

    if (currPower >= powerInWatts) {
        double newPower =
            currPower - powerInWatts;

// Dot-syntax can also be used for the setter.
// This is the same as calling setCurrentPower:
// with bracket syntax.
        self.currentPower = newPower;

        [city recievePower:powerInWatts]
    }
}
@end
```

MSVPowerPlant.h

MSVPowerPlant.h › @interface MSVPowerPlant

```objc
//
//  MSVPowerPlant.h

#import <Foundation/Foundation.h>
@class MSVCity;

@interface MSVPowerPlant : NSObject

@property (nonatomic, readonly) double currentPower;

- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts;

@end
```

Counterparts › MSVPowerPlant.m › No Selection

```objc
//
//  MSVPowerPlant.m

#import "MSVPowerPlant.h"
#import "MSVCity.h"

@interface MSVPowerPlant ()
@property (nonatomic) double currentPower;
@end

@implementation MSVPowerPlant

// You can override the generated accessors and
// setters that the compiler generates
- (void)setCurrentPower:(double)currentPower
{
// _currentPower is an automatically generated
// variable of type "double" associated with this
// instance of MSVPowerPlant that is storage for
// the currentPower property

    _currentPower = MAX(currentPower, 0);
}

- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts
{
    double currPower = self.currentPower;

    if (self.currentPower >= powerInWatts) {
        self.currentPower =
            self.currentPower - powerInWatts;

        [city recievePower:powerInWatts]
    }
}
@end
```

MSVPowerPlant.h

MSVPowerPlant.h › @interface MSVPowerPlant

```objc
//
//  MSVPowerPlant.h

#import <Foundation/Foundation.h>
@class MSVCity;

@interface MSVPowerPlant : NSObject

@property (nonatomic, readonly) double currentPower;

- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts;

@end
```

Counterparts › MSVPowerPlant.m › @implementation MSVPowerPlant

```objc
//
//  MSVPowerPlant.m

#import "MSVPowerPlant.h"
#import "MSVCity.h"

@interface MSVPowerPlant ()
@property (nonatomic) double currentPower;
@end

@implementation MSVPowerPlant
/*
 When both the accessor and setter for a property
 are overridden, we need to tell the compiler to
 still generate the backing instance variable. We
 do this with the @synthesize directive, which
 tells the compiler "Please create an instance
 variable named _currentPower to back the property
 currentPower"
*/
@synthesize currentPower = _currentPower;

- (void)setCurrentPower:(double)currentPower
{
    _currentPower = MAX(currentPower, 0);
}

- (double)currentPower
{
    return _currentPower;
}

- (void)sendPowerToCity:(MSVCity *)city
                 amount:(double)powerInWatts
{
    double currPower = self.currentPower;

    if (self.currentPower >= powerInWatts) {
        self.currentPower =
```