

iOS Development Workshop @ Cohab

Brent Nycum
brentnycum@gmail.com

Wifi Password

- cohab610

Slides Online

- <http://ios.itsbrent.net>

- Pointers
- Model-View-Controller
- Syntax
- Memory Management

Pointers

Pointers

- A programming language data type whose value refers directly to (or "points to") another value stored elsewhere in the computer memory using its address.

Wikipedia

Pointers

- A “pointer” is a variable that indirectly tracks the location of some piece of data.
- The value of a pointer is often the memory address of the data you are interested in.

Pointers

- Used to allocate and deallocate dynamic variables in memory
 - After variable has served its purpose, it is a waste of memory to keep it
 - Good practice to deallocate it when it is no longer needed
 - Failure to do so may result in a memory leak

Model-View-Controller

MVC

Controller

Model

View

MVC

Controller

Model

View

Model = Describe your data

MVC

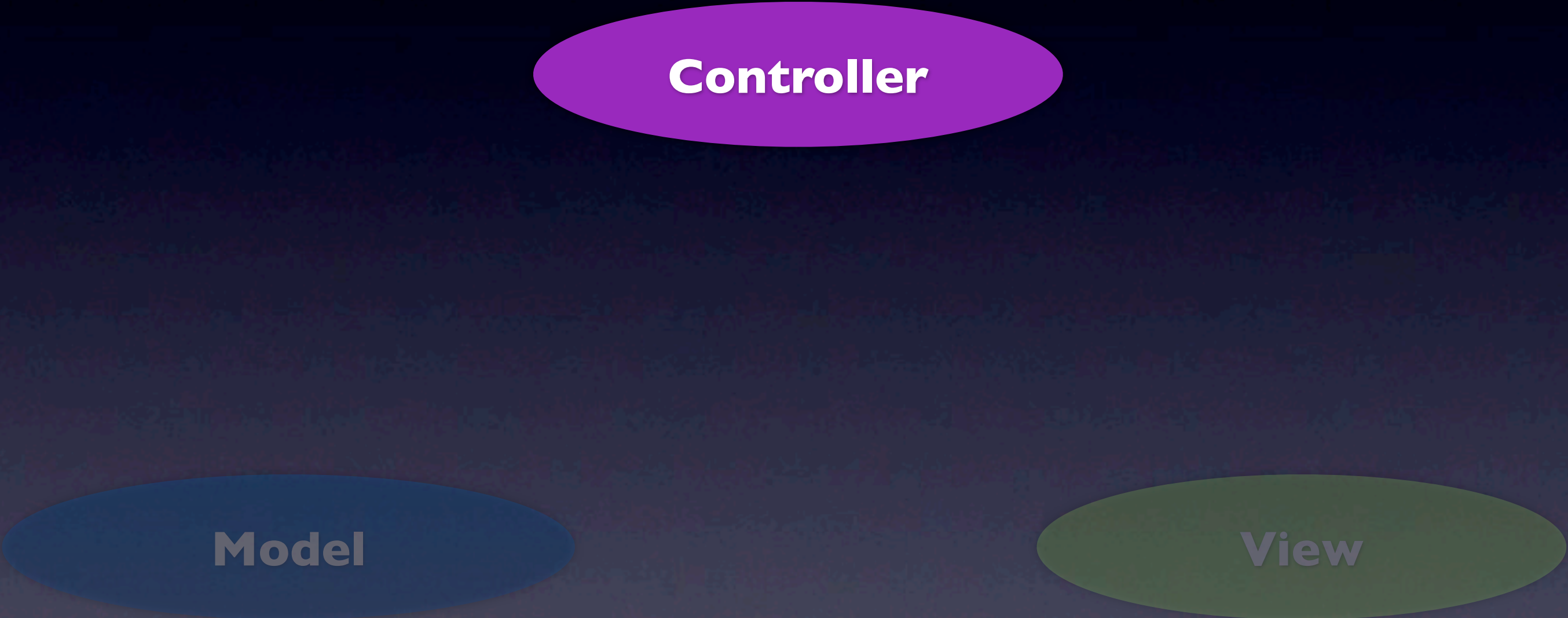
Controller

Model

View

View = GUI

MVC



The diagram illustrates the MVC (Model-View-Controller) pattern. At the top center is a purple oval labeled 'Controller'. Below it, to the left, is a blue oval labeled 'Model', and to the right is a green oval labeled 'View'. The Controller is positioned centrally above the other two components.

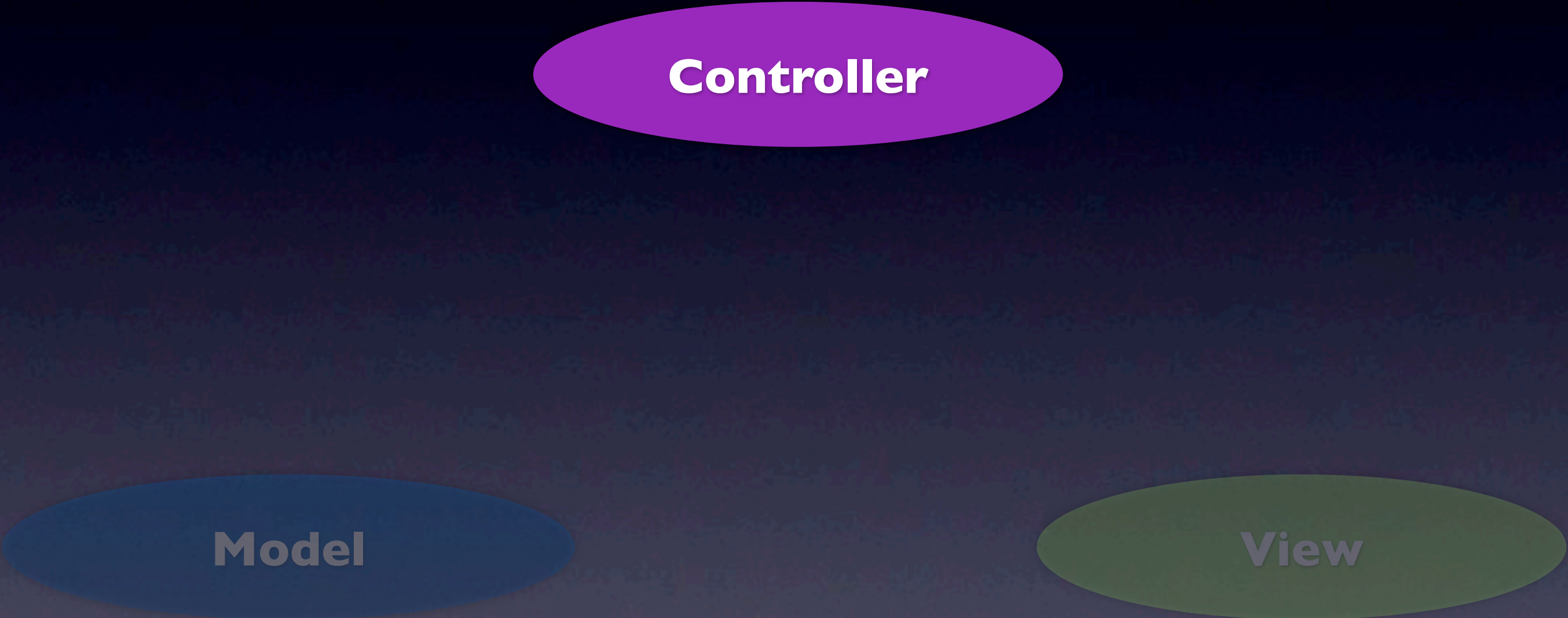
Controller

Model

View

Controller = Receives input and makes calls to model objects

MVC



The diagram illustrates the MVC (Model-View-Controller) pattern. At the top center is a purple oval labeled 'Controller'. Below it, to the left, is a blue oval labeled 'Model', and to the right is a green oval labeled 'View'. The Controller is positioned centrally above the other two components.

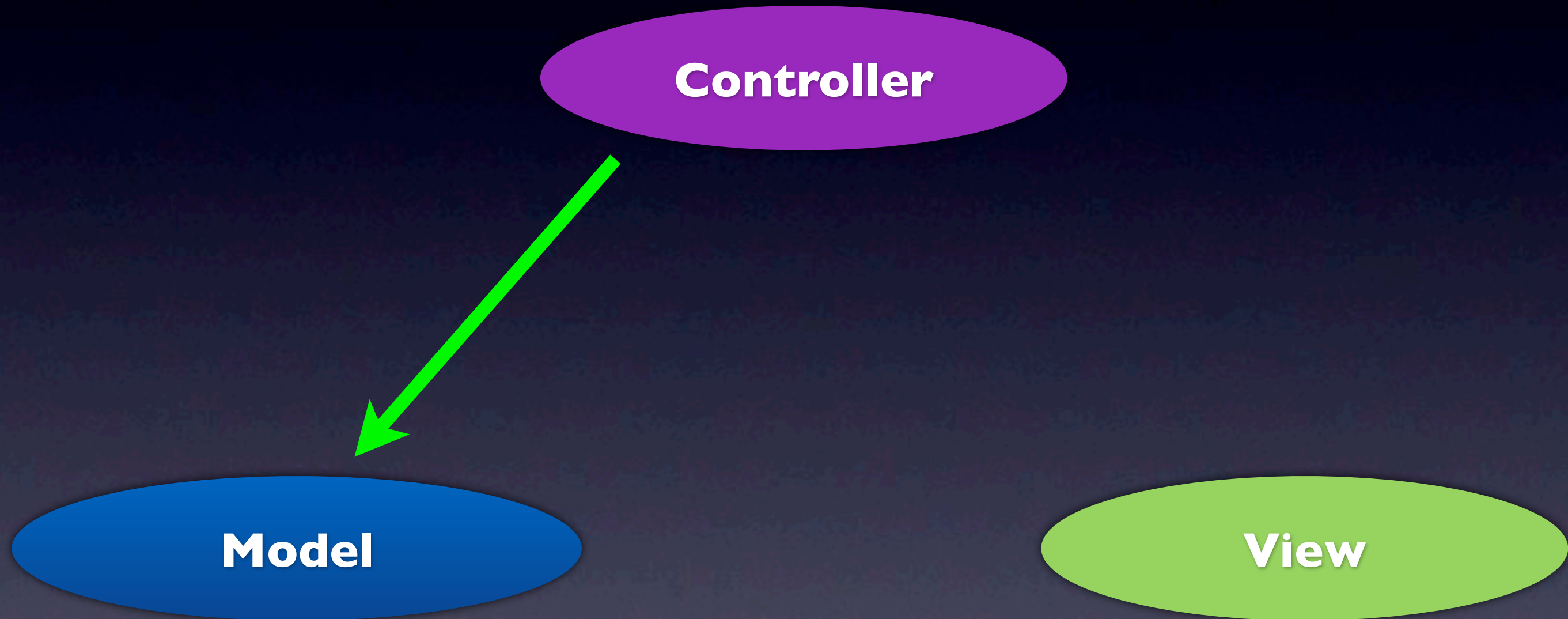
Controller

Model

View

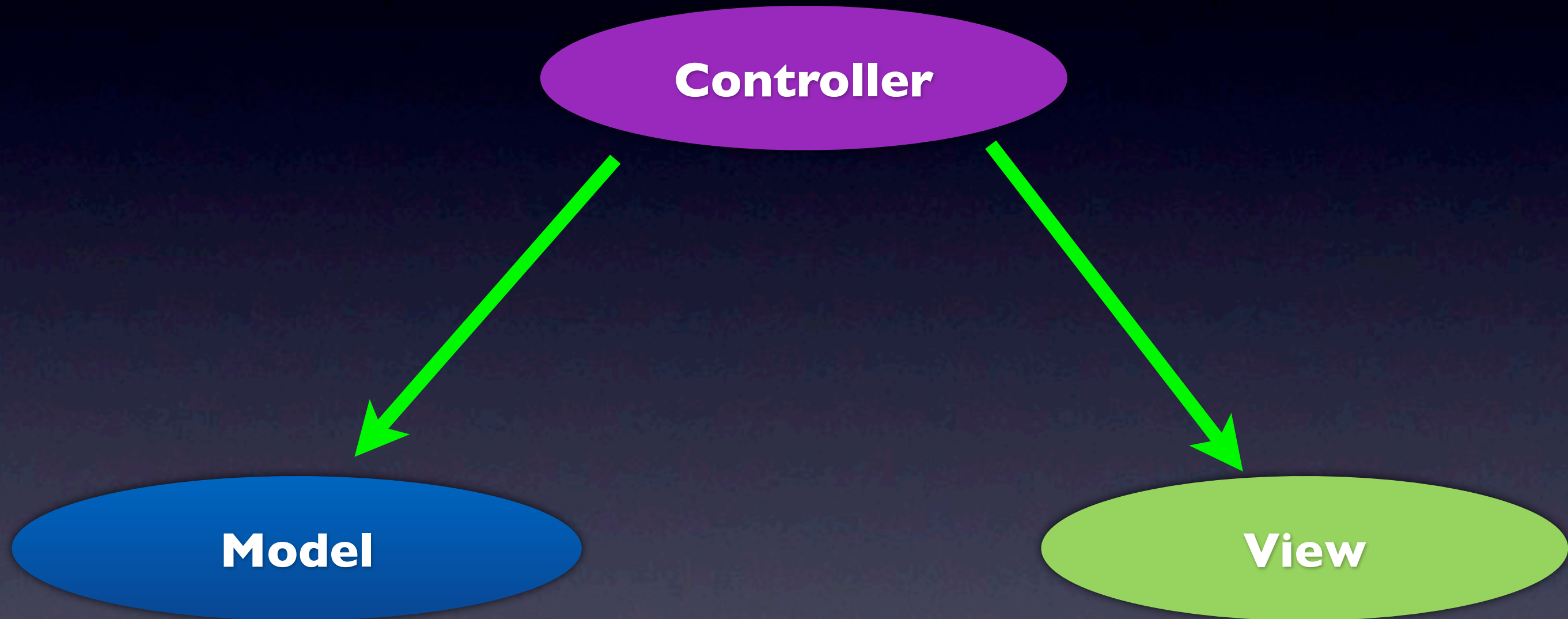
Controller = Interpret/format Model data for the View

MVC



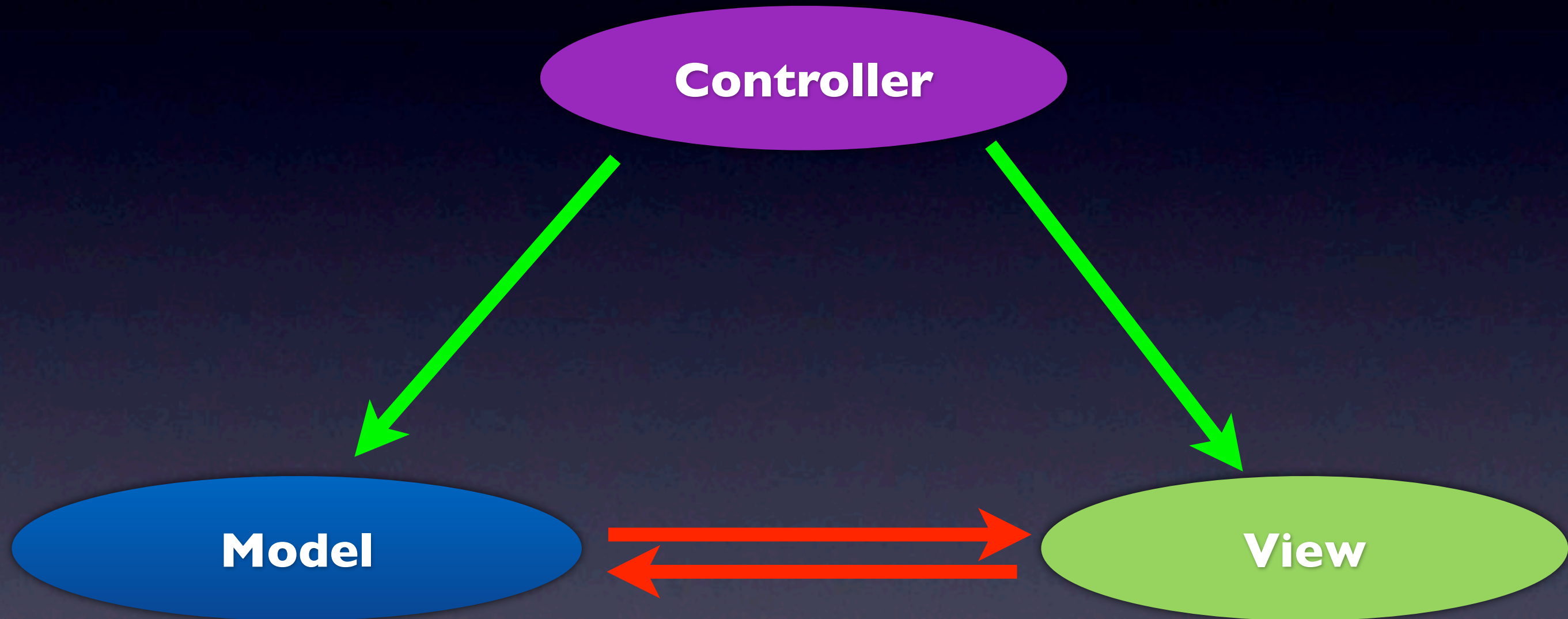
Controllers can always talk to their Models

MVC



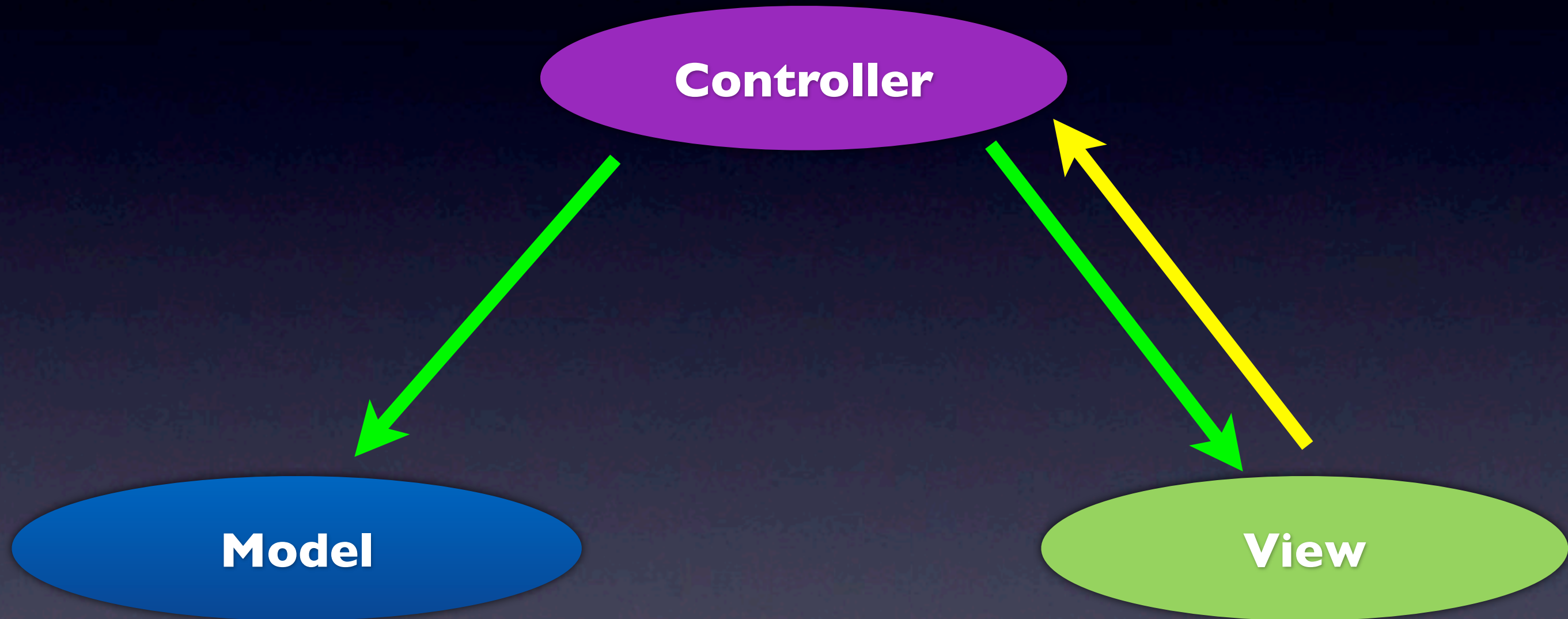
Controllers can also talk directly to their View

MVC



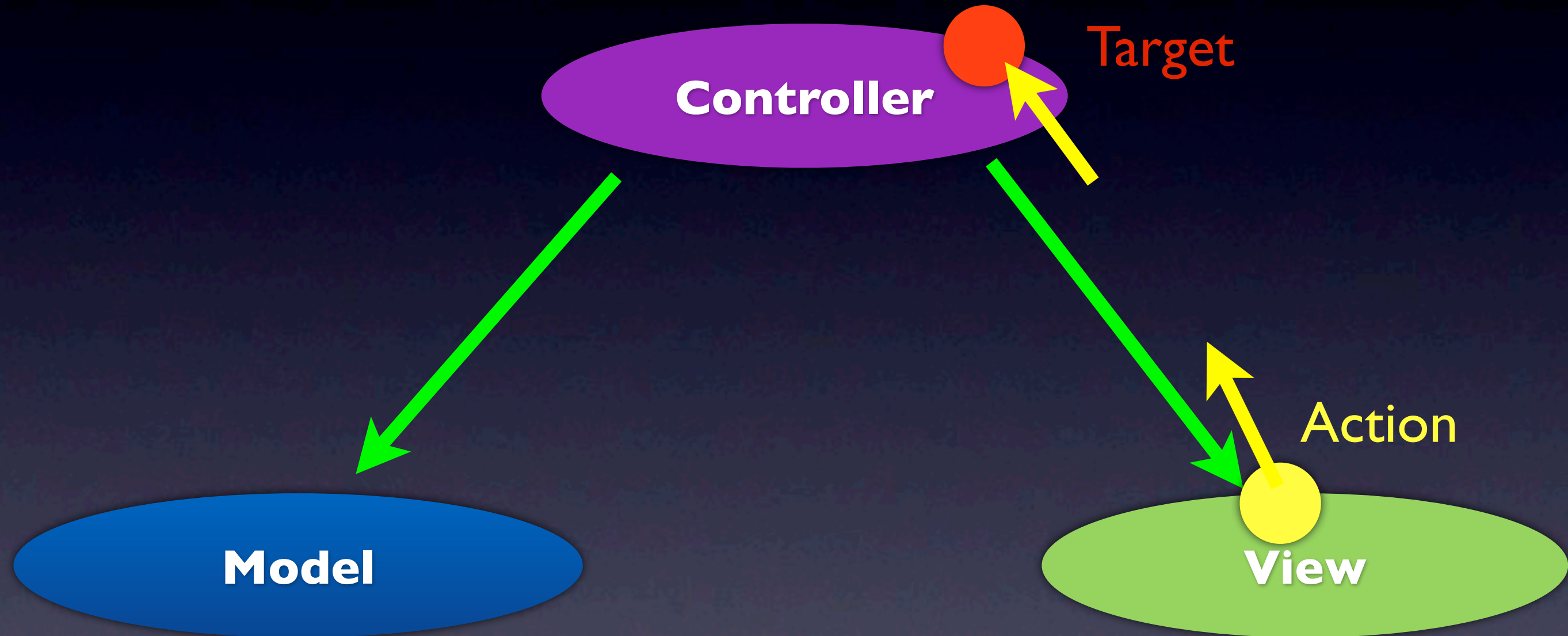
The Model and View shouldn't talk together

MVC



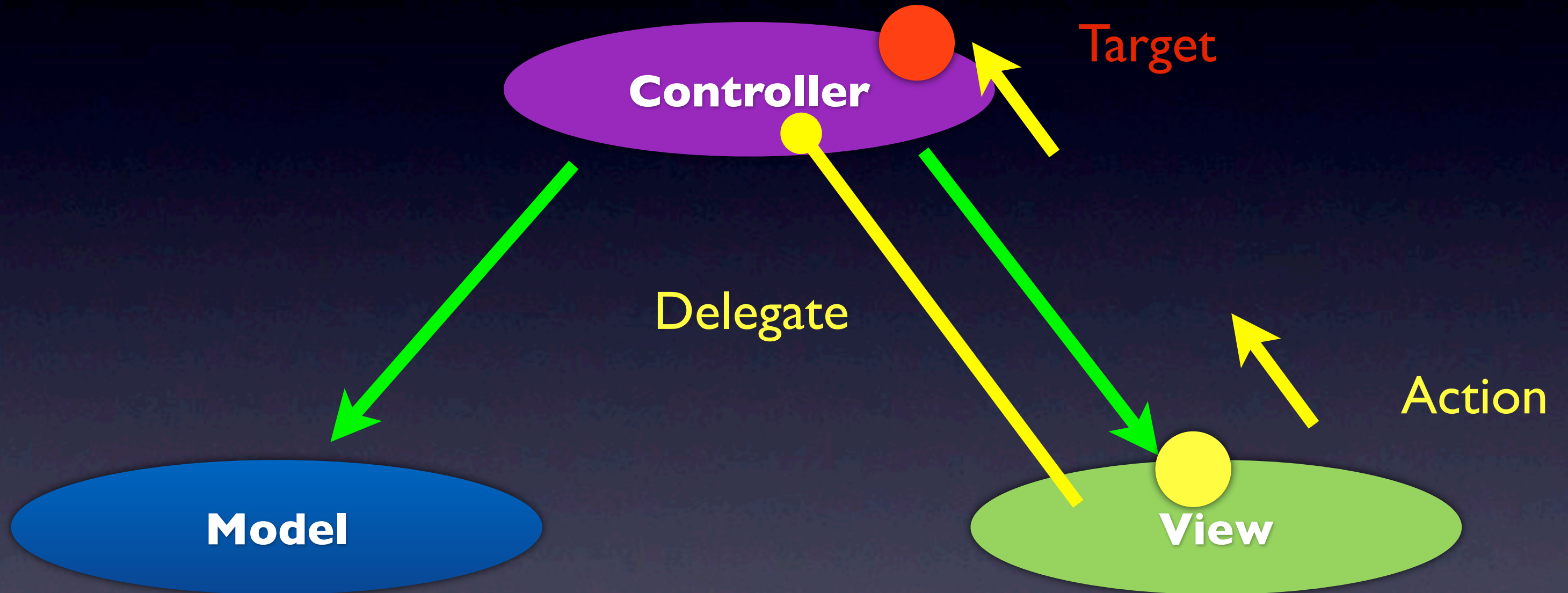
Can the View speak to the Controller?

MVC



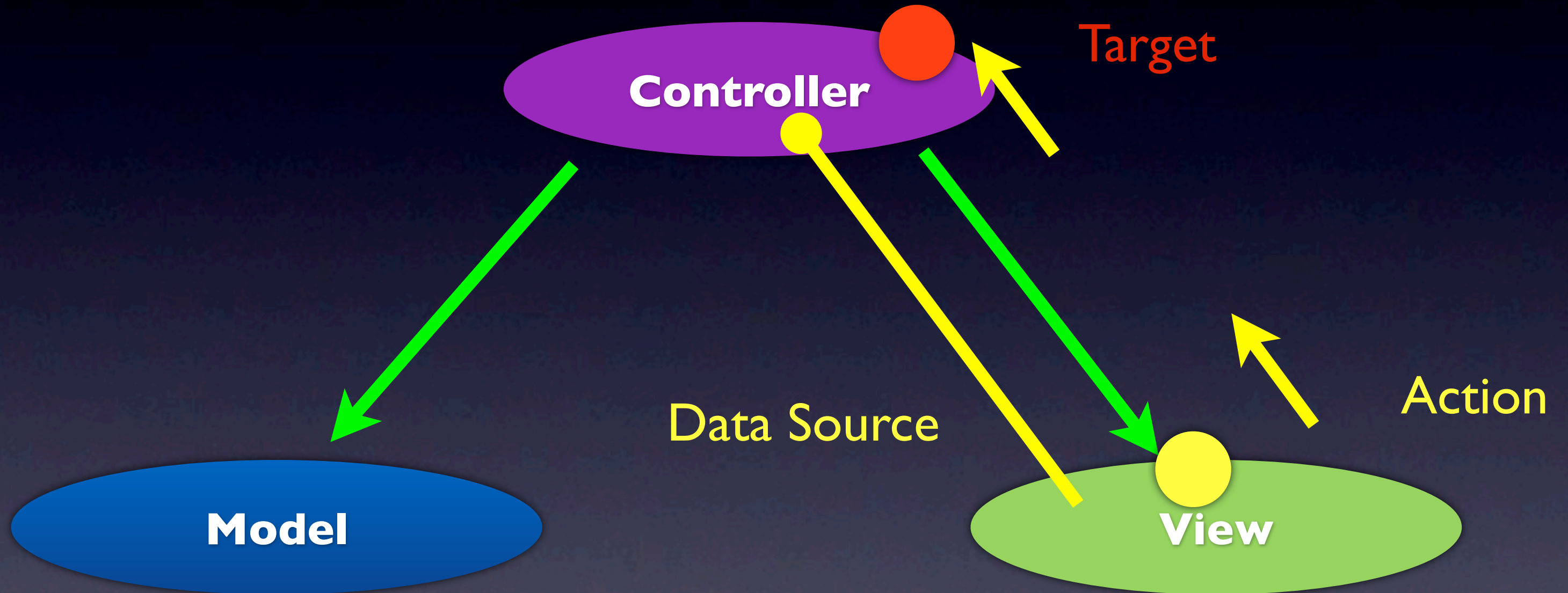
Sort of? Targets and actions

MVC



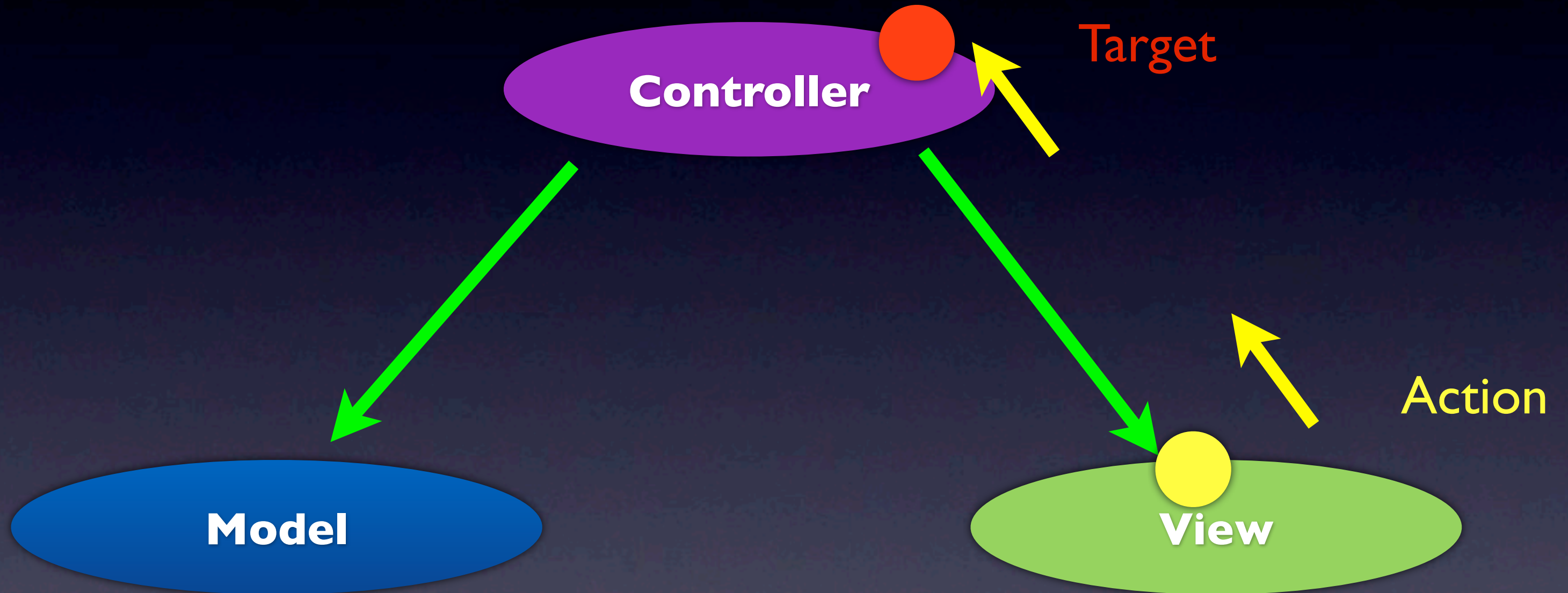
Delegates - Handle the (should, would, did)

MVC



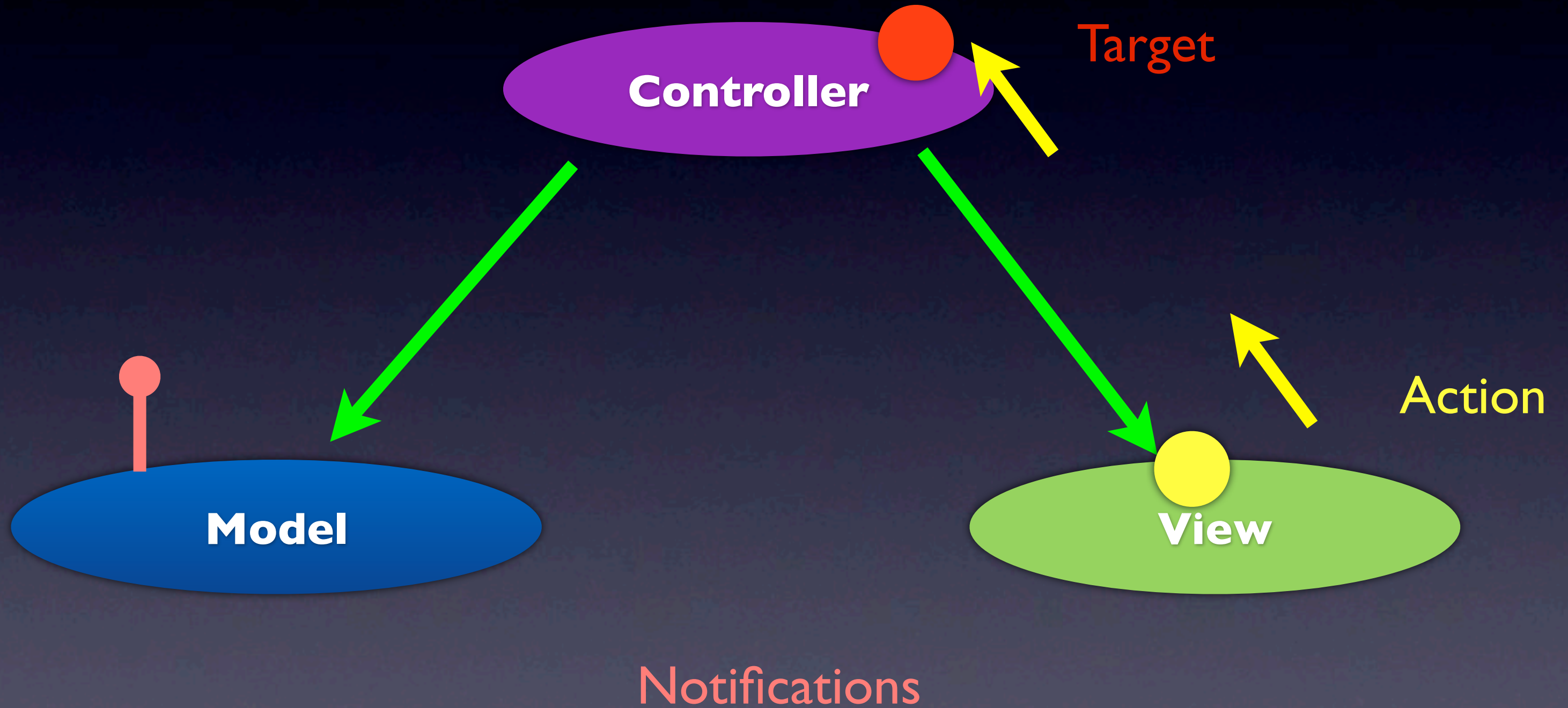
Data Source - Handle the (data at position, count of data)

MVC

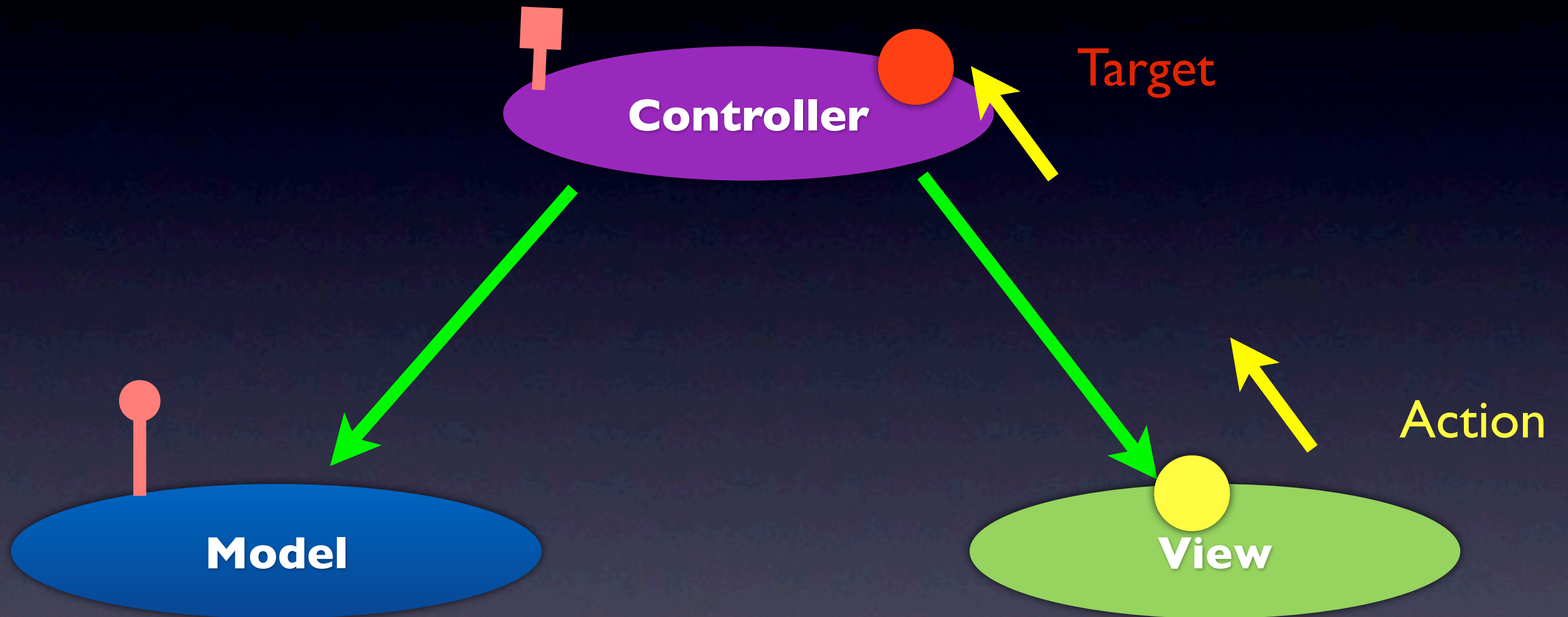


What if the Model (our data) changes?

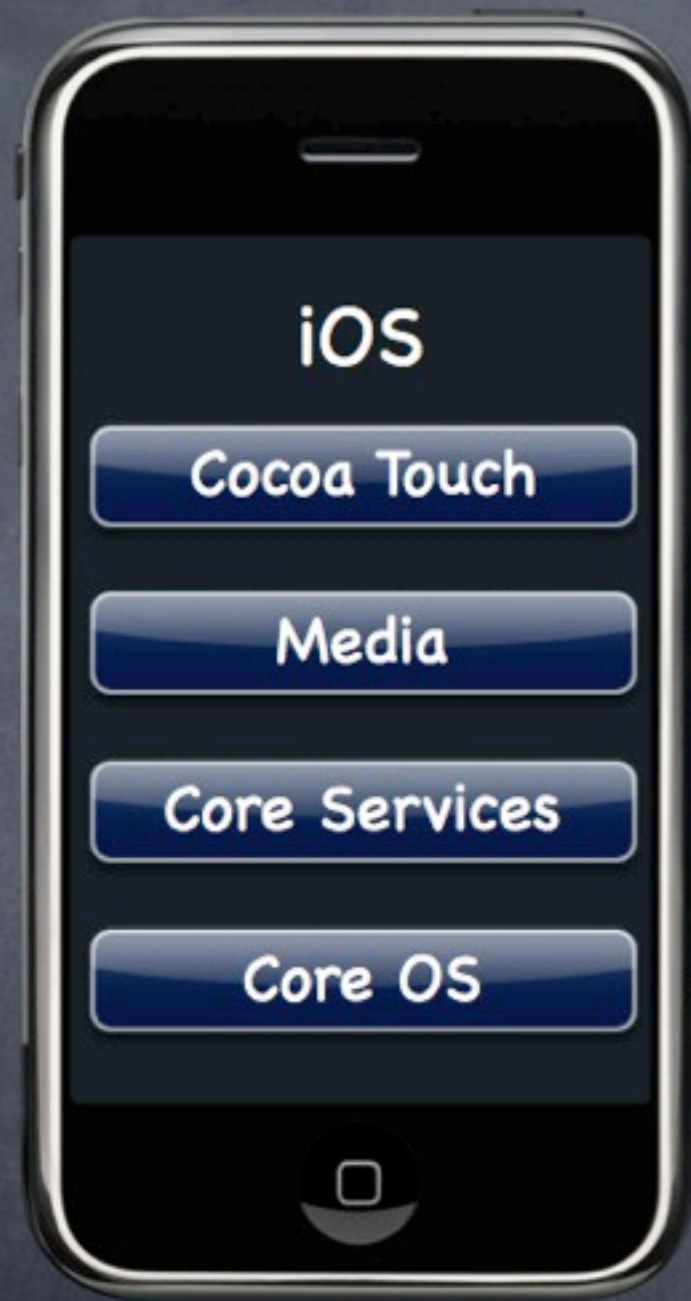
MVC



MVC



Objective-C



Core OS

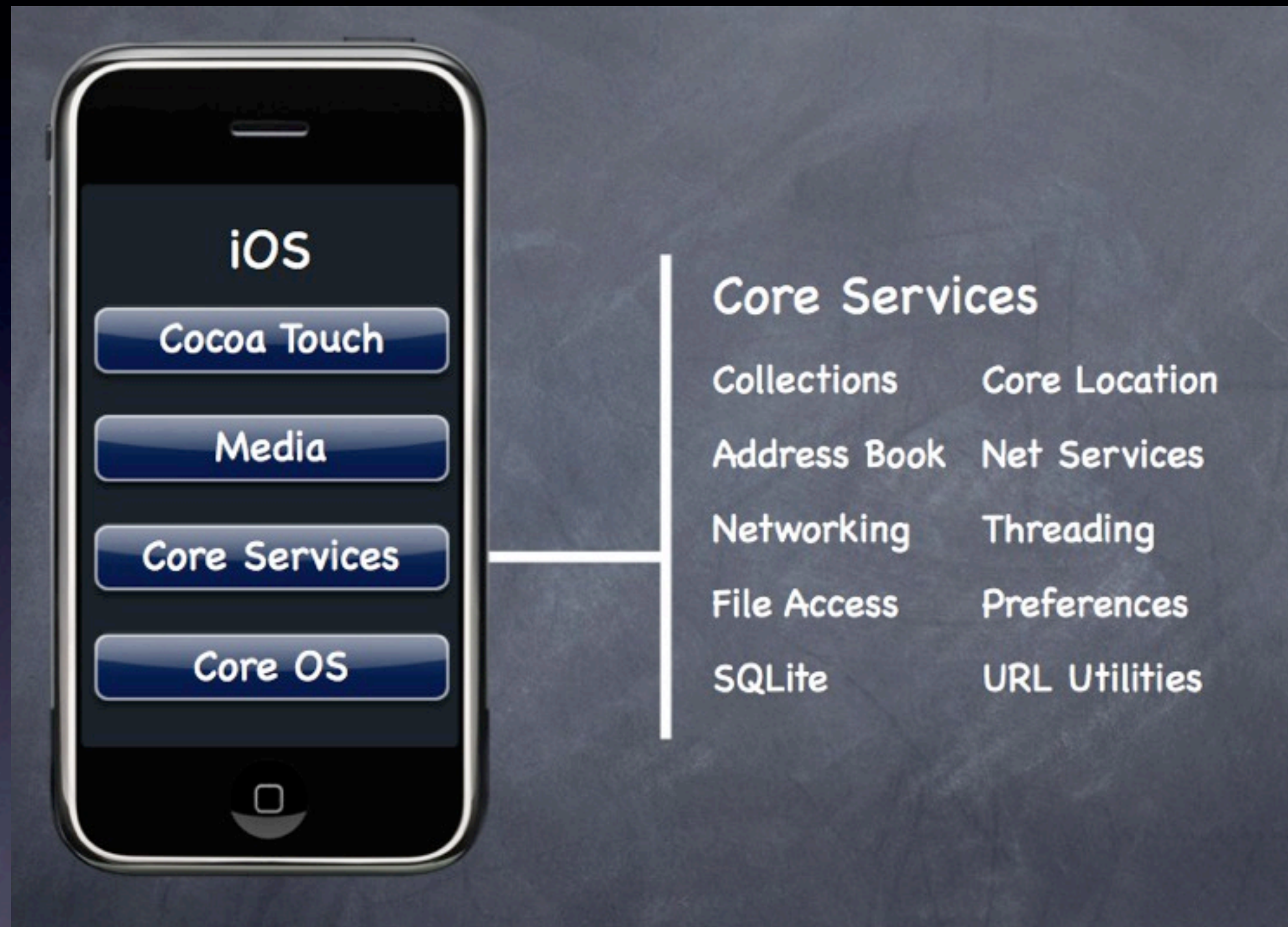
OSX Kernel Power Management

Mach 3.0 Keychain Access

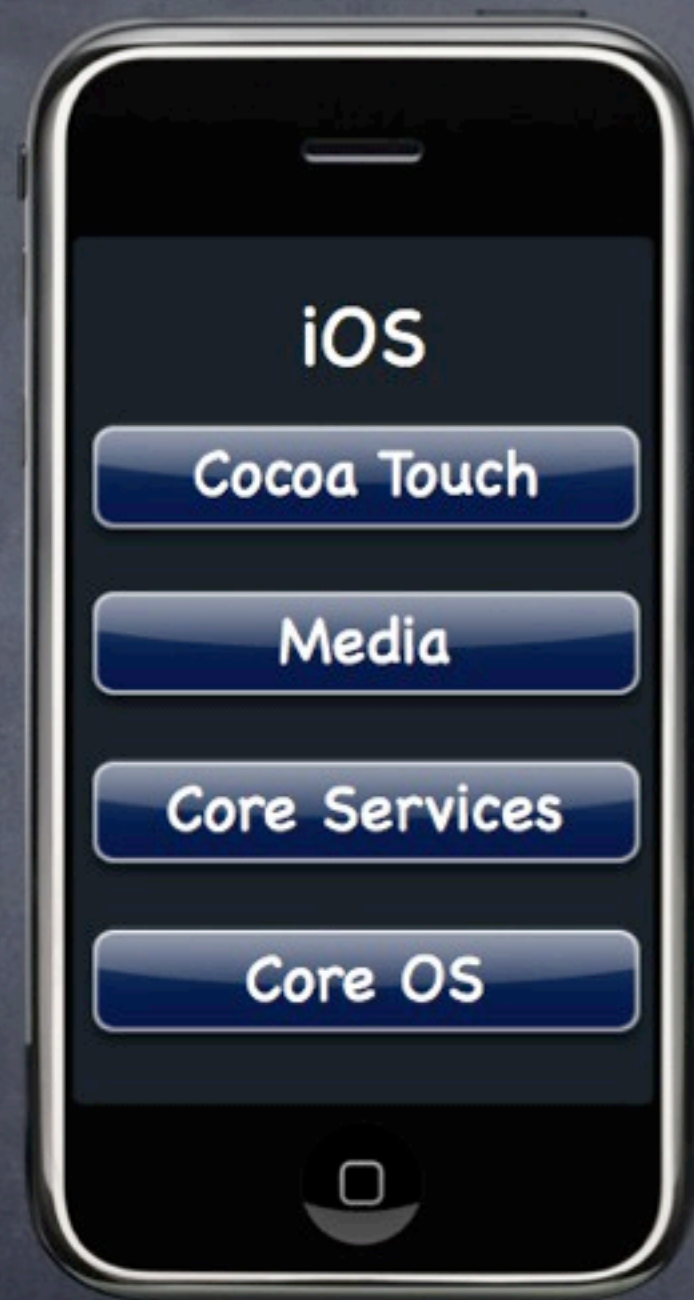
BSD Certificates

Sockets File System

Security Bonjour







Cocoa Touch

Multi-Touch

Alerts

Core Motion

Web View

View Hierarchy

Map Kit

Localization

Image Picker

Controls

Camera

Objective-C

- Scary
- Confusing
- Weird Syntax
- What are those brackets? []

What is Objective-C?

- Object-Oriented Programming Language
- Smalltalk like messaging
- Thin Layer on top of C
- Obj-C

Superset of C

- Can compile C programs with an Objective-C compiler
- Include C code inside Objective-C classes

Objective-C++

- Combo of C++ and Objective-C
- Not discussing in this class, just an FYI

hello.m

```
#import <stdio.h>
```

```
int main( int argc, const char *argv[] ) {  
    printf( "hello world\n" );  
    return 0;  
}
```

Output

- hello world

#import

- Use #import instead of #include
- #include always causes a file's contents to be inserted
- #import each file is included only once

No Need for Inclusion Guards

Data Types

C Data Types

- int
- float
- double
- char
- enum
- No explicit boolean type

Objective-C

- NSObject
- NSString
- NSNumber
- BOOL

BOOL

- BOOL flag;

```
if (flag) { ... }  
if (!flag) { ... }
```

YES means “true”
NO means “false”

- BOOL flag = YES;

nil

- You might be used to null
- Non-valid object pointer
- Treated as boolean value false
- Can still call methods on a nil object

What's this NS Prefix?

- NS prefix stands for NeXTStep
- NSObject = NeXTStep Object

Other Prefixes

- UI = UIKit / Cocoa Touch
- CA = Core Animation
- CF = Core Foundation
- CG = Core Graphics
- CI = Core Image
-

Class Prefixes

- Popular in Objective-C culture
- Most Open-Source projects adopt their own prefix
- BNR - Big Nerd Ranch

NSObject

- Root class of most Obj-C Classes
- Implements memory management primitives

Code Style

Method Syntax

- `-(void)method;`
- `[myObject method];`
- `-(void)methodWithInput: (type)input;`
- `[myObject methodWithInput: input];`

Method Syntax

- Might be familiar with:
- `public void method()`
- `myObject.method();`
- `myObject->method();`

Method with Parameter

```
-(NSArray *)componentsSeparatedByString:(NSString *)separator
```

Java / C++ like Syntax

```
public NSArray* componentsSeparatedByString(NSString* separator)
```

Call this Method

```
NSArray *myArray = [myStr componentsSeparatedByString:myToken];
```

Multiple Parameters

- Used to
 - `public void setSize(int width, int height);`
 - `rectangle.setSize(20, 20);`
- `-(void)setWidth:(int)width height:(int)h;`
- `[rectangle setWidth:20 height:20];`

Interface Files

- “Header file”
- Declares the class interface
- Actual code is in the implementation file
- .h extension

Implementation File

- Where our code lives
- .m file extension, originally meant “messages”

Interface File

```
@interface Rectangle : NSObject {  
    int width;  
    int height;  
}
```

```
-(void)setHeight:(int)height;  
-(int)height;
```

```
-(void)setWidth:(int)width;  
-(int)width;
```

```
@end
```

Implementation File

```
@implementation Rectangle
```

```
-(void)setHeight:(int) h {  
    height = h;  
}
```

```
-(int)height {  
    return height;  
}
```

```
...
```

```
@end
```

Instance Variables

- Defined between {}'s of our interface
- Default variables are protected, but can be set to public or private

Properties

- Part of Objective-C 2.0
- Instance variables
 - `int width;`
 - `int height;`
- Methods:
 - `-(void)setWidth:(int)width;`
 - `-(int)width;`

Properties

- Interface file
 - `@property int width;`
- Implementation file
 - `@synthesize width;`
- `rectangle.width = 20;`

Interface File

```
@interface Rectangle : NSObject {  
    int width;  
    int height;  
}
```

```
@property int width;  
@property int height;
```

```
@end
```

Implementation File

```
@implementation Rectangle
```

```
@synthesize width;
```

```
@synthesize height;
```

```
@end
```

Piecing it Together

- `Rectangle *myRect;`
 - `myRect = [Rectangle alloc];`
 - `[myRect init];`
-
- `alloc` - Allocate memory for object
 - `init` - Initialize the object

Piecing it Together

- Java Syntax
 - `Rectangle myRectangle = new Rectangle();`

Piecing it Together

- Usually done in one line
 - `Rectangle *myRect = [[Rectangle alloc] init];`
- Will discuss freeing memory later

Piecing it Together

- `[myRect setWidth: 20];`
- `myRect.width = 20;`

- `int w = [myRect width];`
- `int w = myRect.width;`

The id Type

- Strictly for objects
- In Obj-C when you call a method on an object, it doesn't need to know the type
- Method just has to exist
- Referred to as message passing

NSString

- Used instead of C's `char*` type
- `@"foo"` notation
 - `NSString *myString = @"foo";`
 - `NSString *myStr = [[NSString alloc] initWithString:@"foo"];`
- Immutable - can't be modified

NSMutableString

- Mutable Version of NSString
- In-place changes (appending, removing, replacing)

NSNumber

- Object wrapper around: int, float, double, BOOL
- `NSNumber *num = [NSNumber numberWithFloat:72.5];`
- `float f = [num floatValue];`

NSArray

- Collection of Objects
- Immutable

NSArray

- In other languages you are used to set type of an array
 - `int[]`
- NSArray can hold any type of object
 - First object might be a string
 - Second object might be an image

NSMutableArray

- Mutable NSArray
- Can add objects, insert at a certain position, remove objects

NSDictionary

- Hash Table
- Keys and Values
- Keys are normally strings
- Values are any type of object
- { @"key1": value1, @"key2": value2 }
- NSMutableDictionary

Memory Management

Garbage Collection

- No Garbage Collection
- Must allocate and free memory manually

ARC

- Automatic Reference Counting
- Coming in iOS 5
- Til then, we must manually manage our memory

Memory Management

- Retain / Release
- To hold onto an object we retain it
 - [object retain];
- To let go of an object and free its memory
 - [object release];

Memory Management

- Every time you retain an object you bump it's retain count +1
- Every time you release an object you decrement it's retain count -1

Retain / Release

- Once an object has a retain count of 0 it is deallocated



Morkus

(my dog)



Roseanne

(just to be fair)

Morkus

- Born
- Bought from Breeder
- Original owner passed away from cancer
- We took in Morkus

Morkus

- Assume each person is a different Object
 - Breeder
 - Original Owner (Ralph)
 - Us

Morkus

Breeder (technically mother)

```
Corgi *morkus = [[Corgi alloc] init];
```

If you allocate an object you must release it

Retain Count: 1

Morkus

Original Owner (Ralph)

```
Corgi *morkus = [[breeder morkus] retain];
```

Now Ralph owns a copy of Morkus

Retain Count: 2

Morkus

Breeder releases Morkus

```
[morkus release];
```

Retain Count: 1

Morkus

Ralph gets cancer; we take in Morkus

```
Corgi *morkus = [ralph morkus] retain];
```

Retain count: 2

```
[morkus release];
```

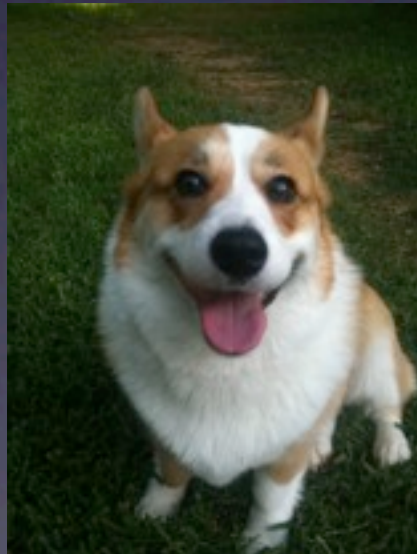
Retain count: 1

Second Memory Management Example

Owner

Trainer

Groomer

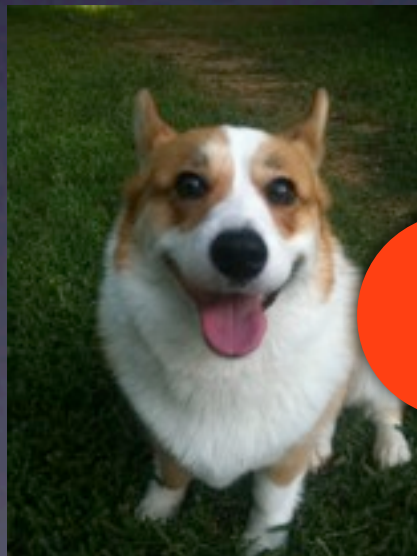


Owner

Trainer

Groomer

Retain



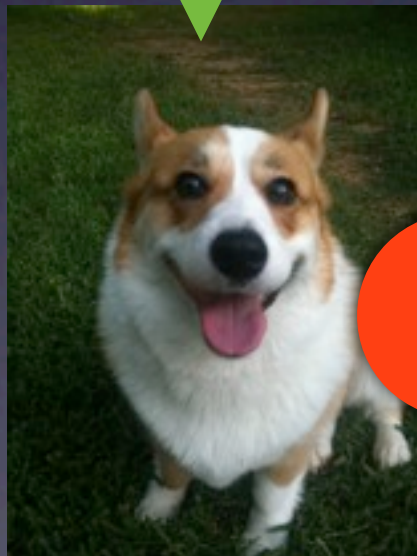
I

Owner

Trainer

Groomer

Retain



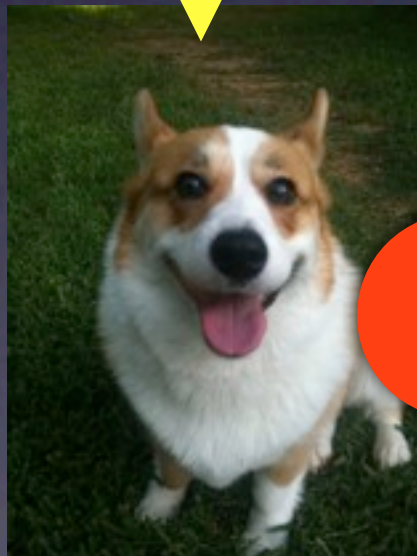
2

Owner

Trainer

Groomer

Release

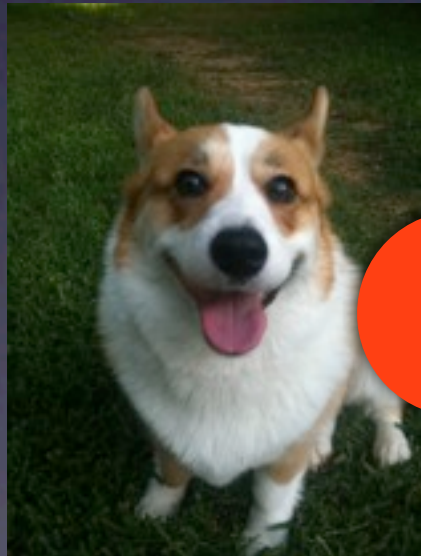


I

Owner

Trainer

Groomer

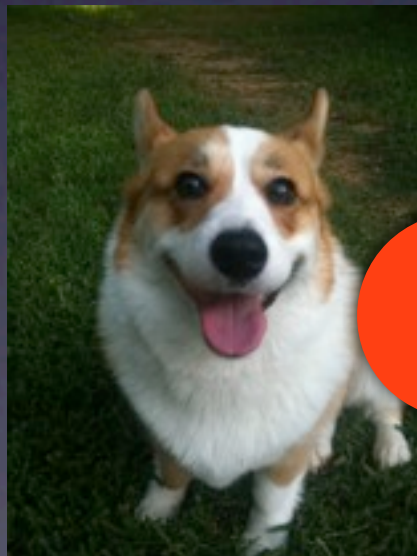


Owner

Trainer

Groomer

Retain



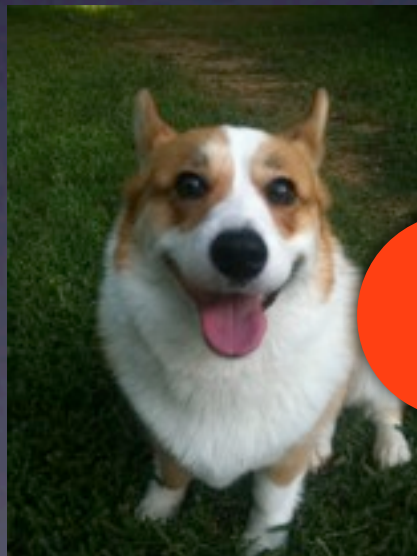
2

Owner

Trainer

Groomer

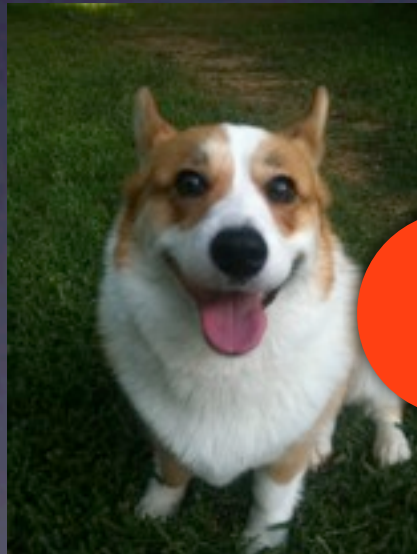
Release



Owner

Trainer

Groomer



Gotcha

- After releasing an object, passing a message to that object will crash
- Set object to nil after releasing it
 - `[myObject release];`
 - `myObject = nil;`

Dealloc

- `-(void)dealloc;`
- Called when Object being removed from memory
- Best time to release references

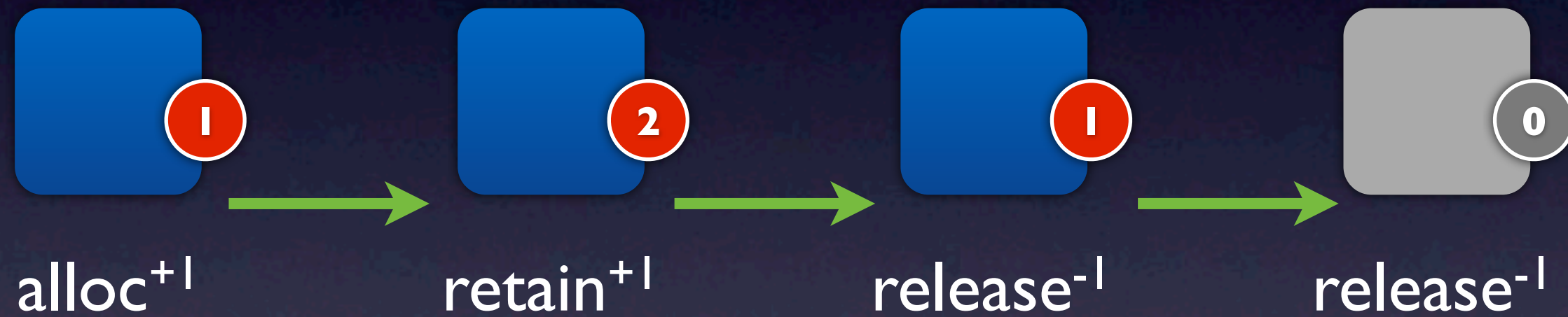
Dealloc

```
- (void)dealloc {  
    [morkus release];  
    [rosie release];  
    [super dealloc];  
}
```

Dealloc

- Last line is very important
- [super dealloc];
- If you don't do this the object won't be removed, which is a memory leak

Memory Management



Memory Management

- `NSString *string = [[NSString alloc] init];`
- `[string release];`

Avoiding Leaks

- `-(id)init {`
 `...`
 `_foo = [[NSString alloc] init];`
 `...`
 `}`

 `-(void)dealloc {`
 `[super dealloc];`
 `}`

Use Autorelease

- For temporary variables
- `NSString *s = [NSString string];`
- or
- `NSString *s = [[[NSString alloc] init] autorelease];`

Setter Methods

- `-(void)setFoo:(Bar *)foo {
 [_foo release];
 _foo = [foo retain];
}`

Setter Methods

- ```
-(void)setFoo:(Bar *)foo {
 if (_foo != foo) {
 [_foo release];
 _foo = [foo retain];
 }
}
```

# Setter Methods

- `-(void)setFoo:(Bar *)foo {  
 [_foo autorelease];  
 _foo = [foo retain];  
}`

Questions?

ios.itsbrent.net