# Stanford CS193p

## Developing Applications for iOS
## Spring 2012

# Today

- What is this class all about?
  - Description
  - Logistics
  - Requirements/Prerequisites
  - Homework/Final Project
- iOS Overview
  - What's in iOS?
- MVC
  - Object-Oriented Design Concept
- Objective C
  - New language!
  - Just going to get a "glimpse" of it today, nothing more.

# What will I learn in this course?

- How to build cool apps
  - Easy to build even very complex applications
  - Result lives in your pocket!
  - Very easy to distribute your application through the AppStore
  - Vibrant development community
- Real-life Object-Oriented Programming
  - The heart of Cocoa Touch is 100% object-oriented
  - Application of MVC design model
  - Many computer science concepts applied in a commercial development platform:
    - Databases, Graphics, Multimedia, Multithreading, Animation, Networking, and much, much more!
  - Numerous students have gone on to sell products on the AppStore

# Requirements

- Must have a Mac
  - Intel-based
  - Lion
- Must have an iOS 5 Device
  - Required for final project!
  - First 4 or 5 homeworks in a simulator.
  - Limited iPod Touch loaners available
- Textbook
  - Apple on-line documentation
  - http://developer.apple.com
- Apple Developer Program
  - Free, but no posting to AppStore.
  - Or $99 for full program if you want.

# Prerequisites

- **Most Important Prereq!**
  Object-Oriented Programming
  CS106A&B required
  CS107 & CS108 recommended
  Other CS will be helpful too
- **Object-Oriented Terms**
  Class (description/template for an object)
  Instance (manifestation of a class)
  Message (sent to object to make it act)
  Method (code invoked by a Message)
  Instance Variable (object-specific storage)
  Superclass/Subclass (Inheritance)
  Protocol (non-class-specific methods)
  (we might have to cover this last one)
- **You should know these terms!**
  If you are not very comfortable with all of these, this might not be the class for you
- **Programming Experience**
  This is an upper-level CS course.

  If you have never written a program where you had to design and implement more than a handful of classes, this will be a big step up in difficulty for you.

# Assignments

- Weekly Homework
  - 6 weekly (approximately) assignments
  - Assigned Thursdays after lecture
  - Due the following Wednesday at 11:59pm
  - Individual work only
  - Homework graded ✓, ✓+ and ✓- based on Required Tasks and Evaluation criteria
  - Lots of extra credit available, bank it
  - Only 3 "free" late days per quarter
  - #1 fail: falling behind on homework
- Final Project
  - 3 weeks to work on it
  - But weighted like 4 weeks of homework
  - Proposal requires instructor approval
  - Some teams of 2 might be allowed
  - Keynote presentation required (3 mins or so)


## Core OS

| | |
|---|---|
| OSX Kernel | Power Management |
| Mach 3.0 | Keychain Access |
| BSD | Certificates |
| Sockets | File System |
| Security | Bonjour |


## Core Services

| | |
|---|---|
| Collections | Core Location |
| Address Book | Net Services |
| Networking | Threading |
| File Access | Preferences |
| SQLite | URL Utilities |


## Media

| | |
|---|---|
| Core Audio | JPEG, PNG, TIFF |
| OpenAL | PDF |
| Audio Mixing | Quartz (2D) |
| Audio Recording | Core Animation |
| Video Playback | OpenGL ES |


## Cocoa Touch

- Multi-Touch
- Core Motion
- View Hierarchy
- Localization
- Controls
- Alerts
- Web View
- Map Kit
- Image Picker
- Camera

# Platform Components

- Tools

Xcode 4

Instruments

- Language

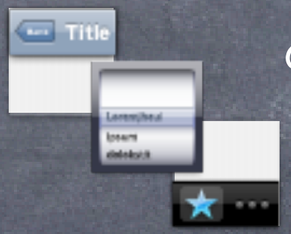
```
[display setTextColor:[UIColor blackColor]];
```

- Frameworks

Foundation

Core Data

Map Kit



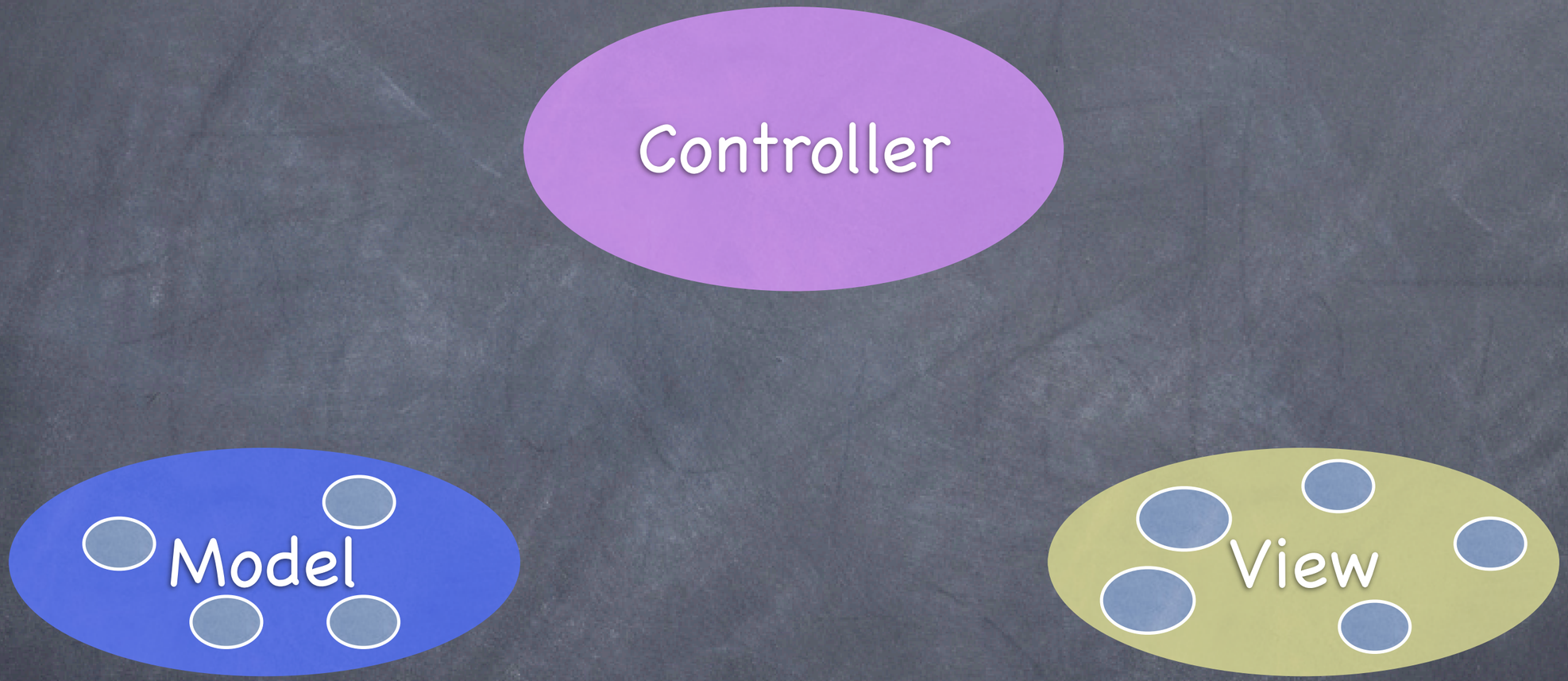

UIKit

Core Motion

- Design Strategies

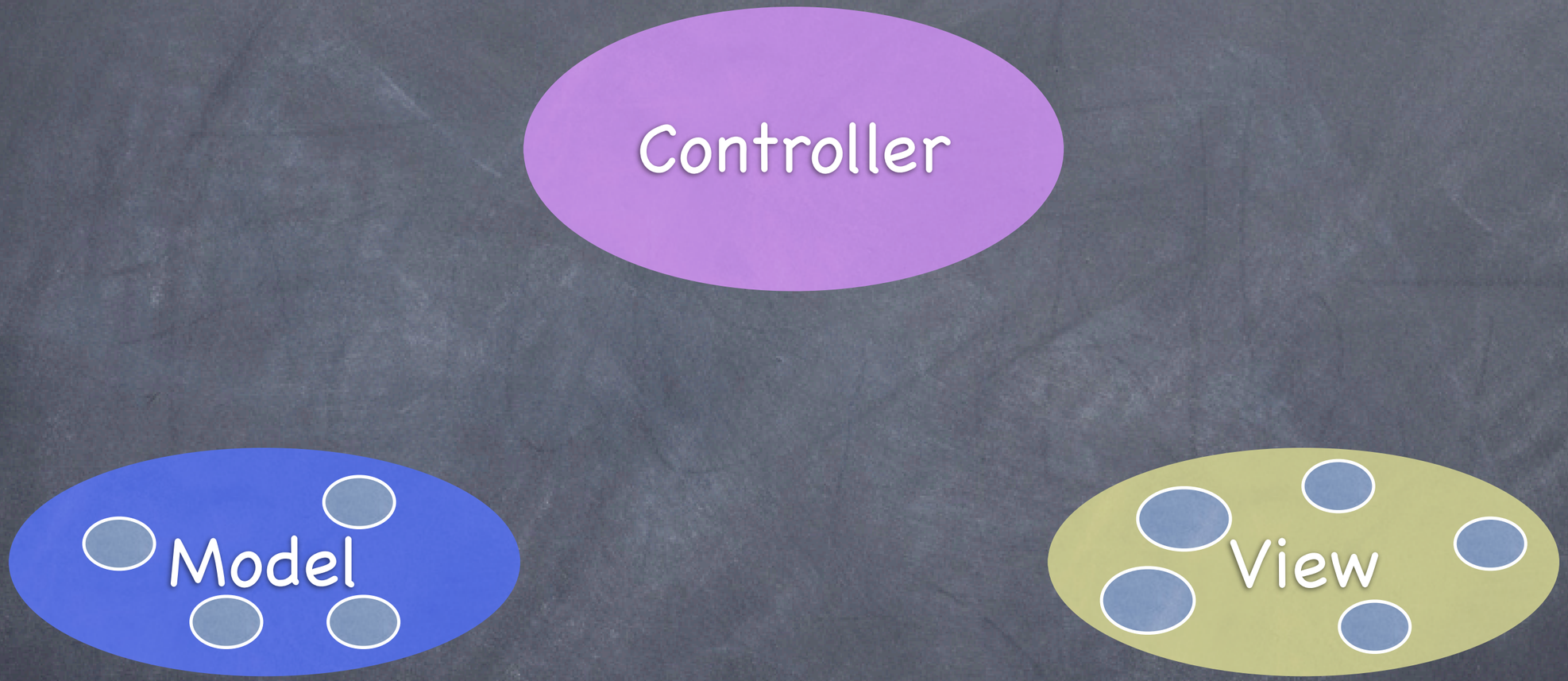
MVC

# MVC



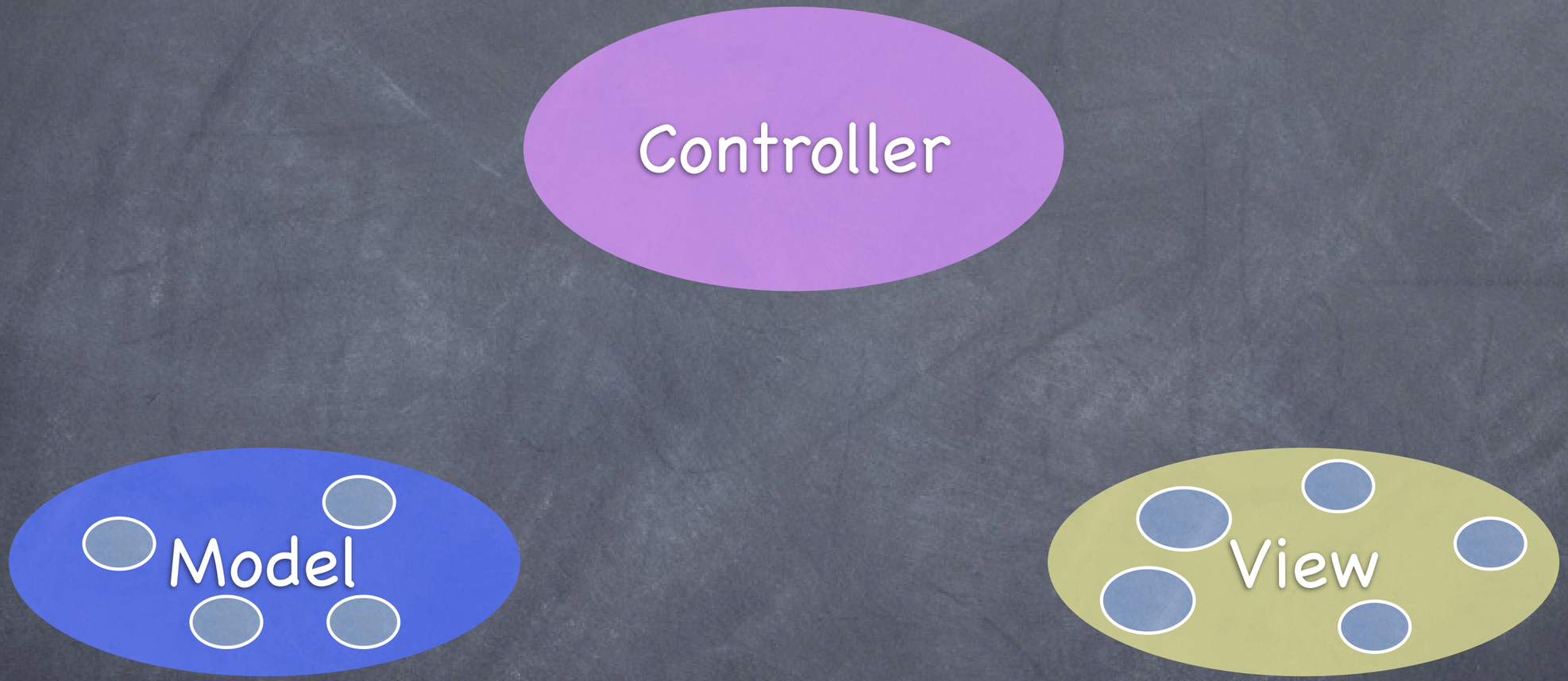

Divide objects in your program into 3 "camps."

# MVC



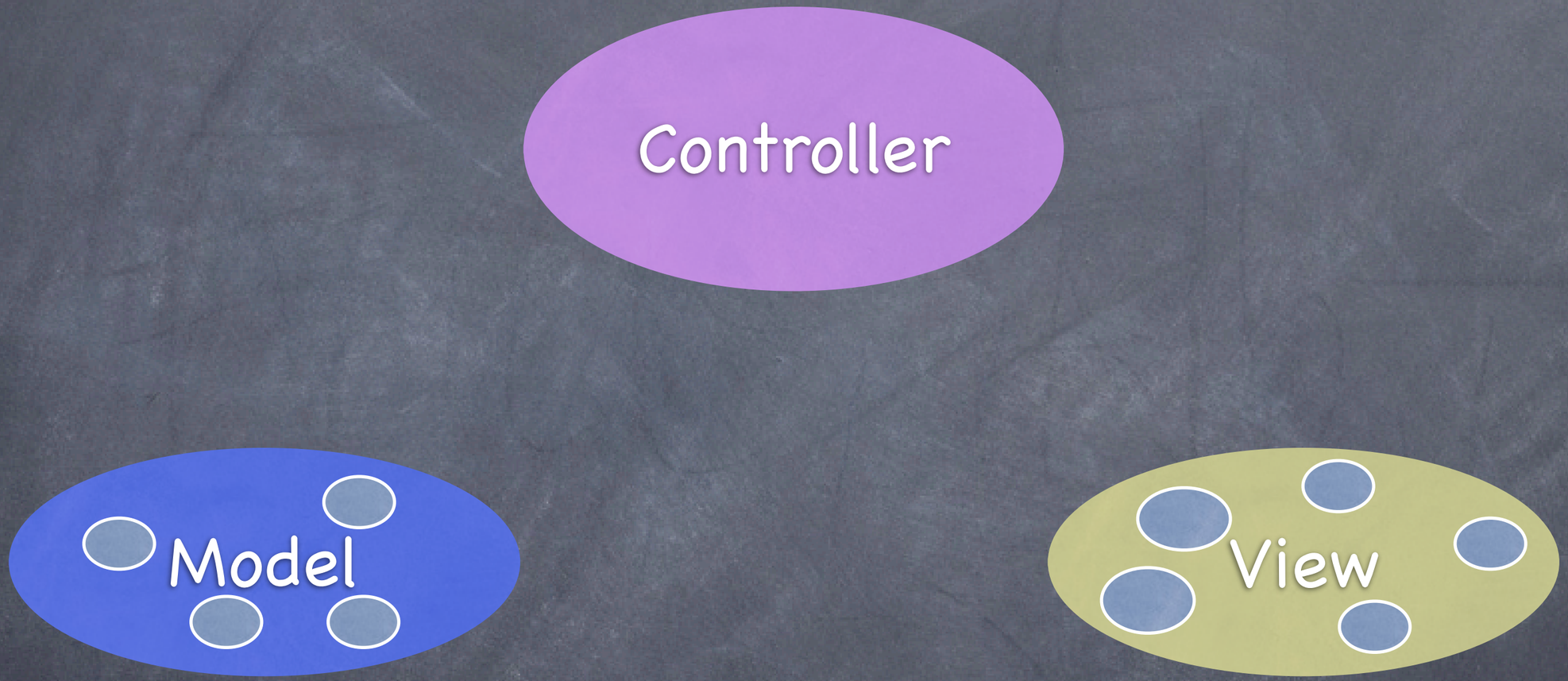

Model = What your application is (but not how it is displayed)

# MVC



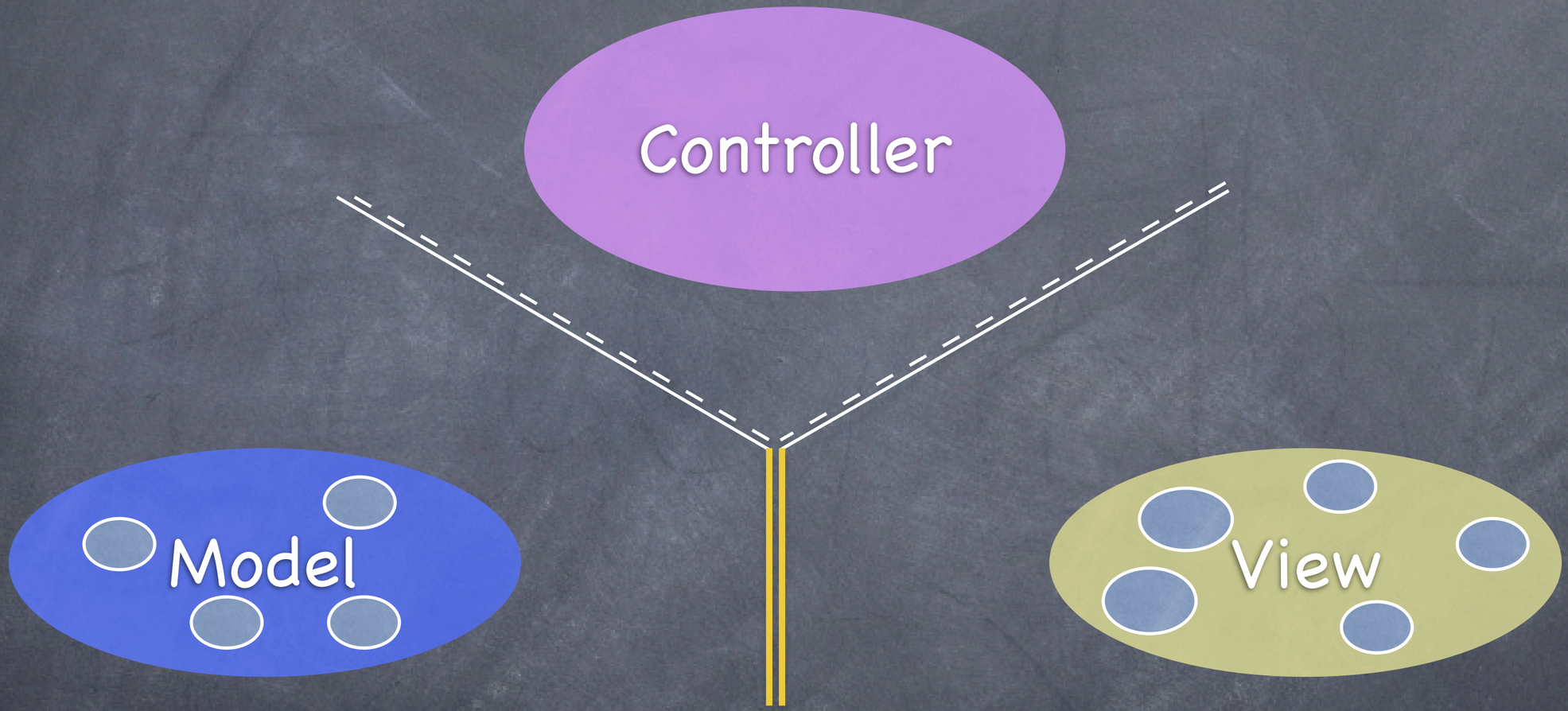

Controller = How your Model is presented to the user (UI logic)

# MVC



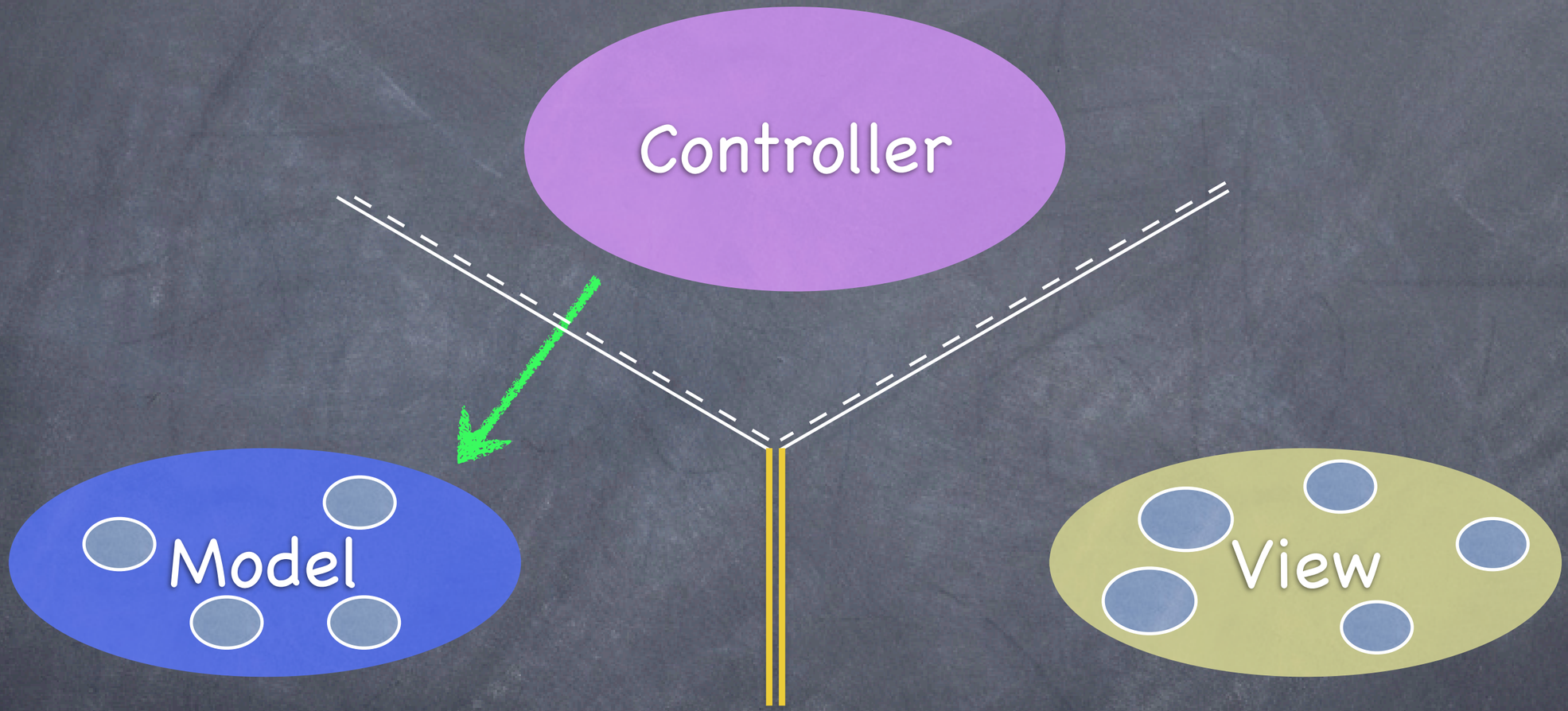

View = Your Controller's minions

# MVC



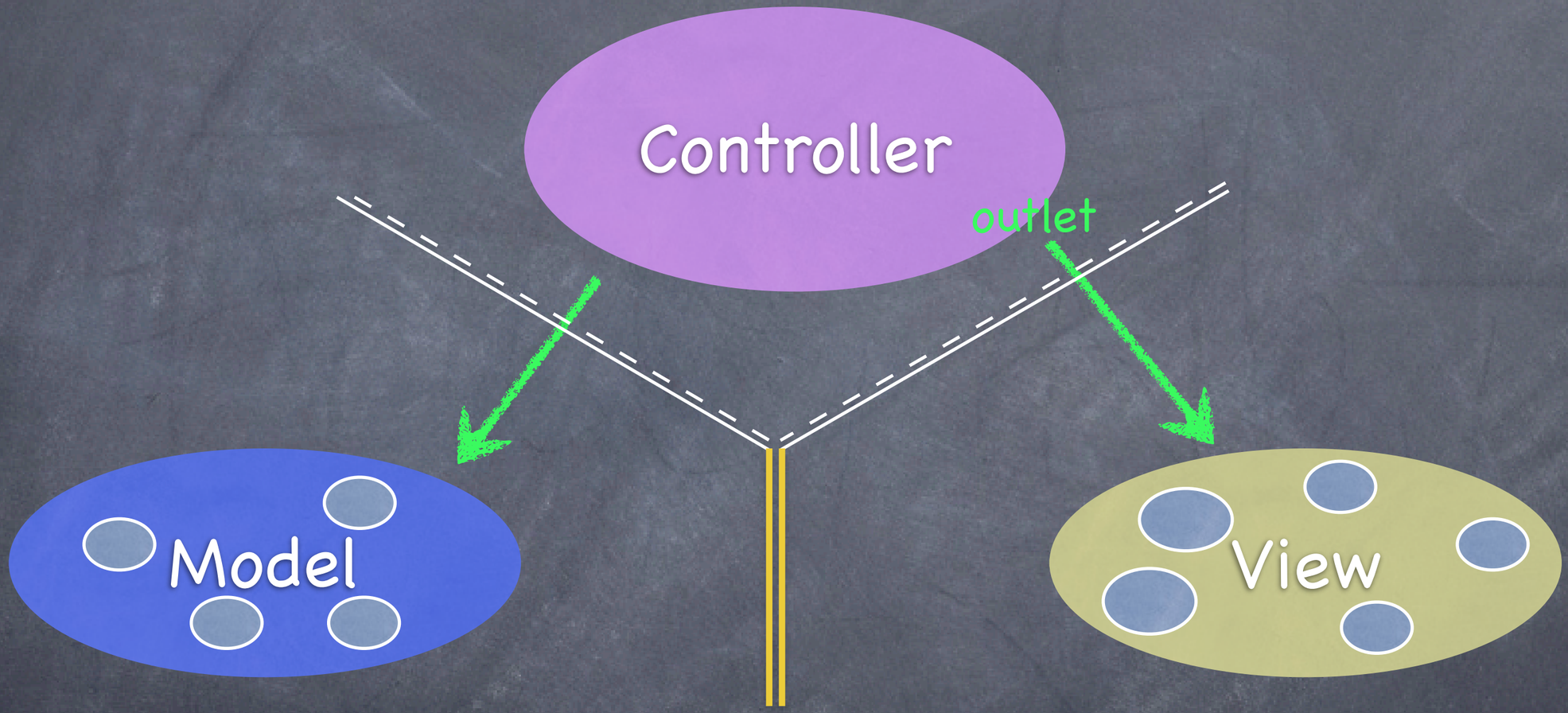

It's all about managing communication between camps

# MVC



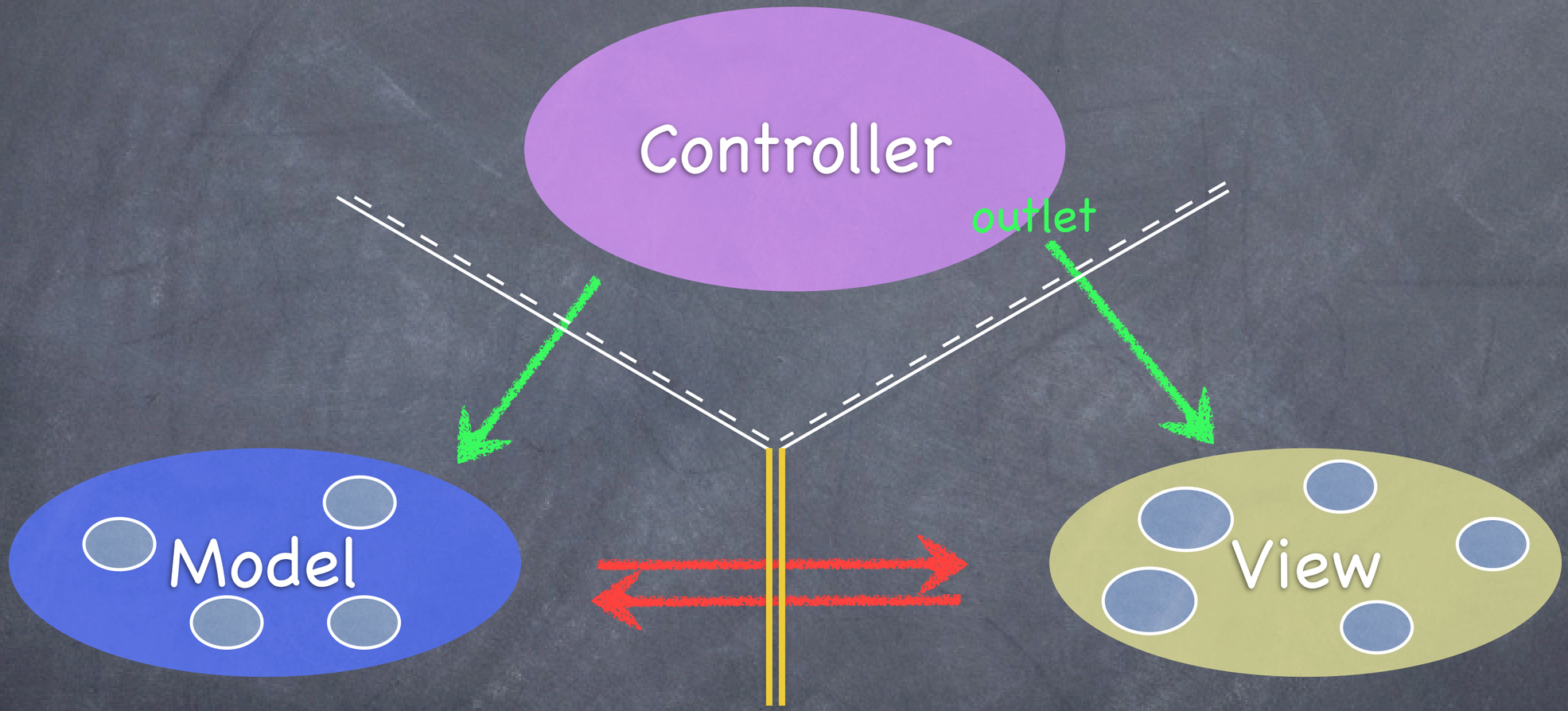

Controllers can always talk directly to their Model.

# MVC



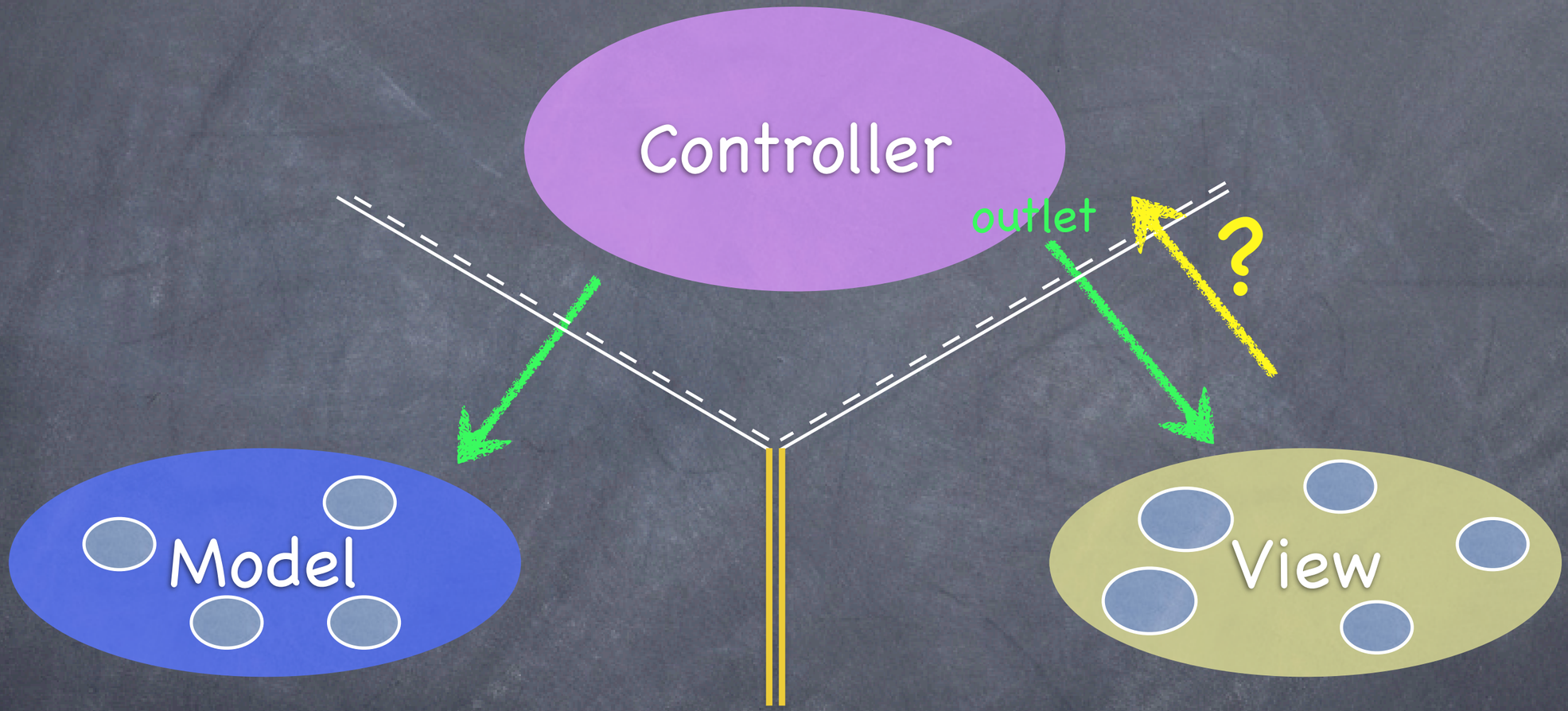

Controllers can also talk directly to their View.

# MVC



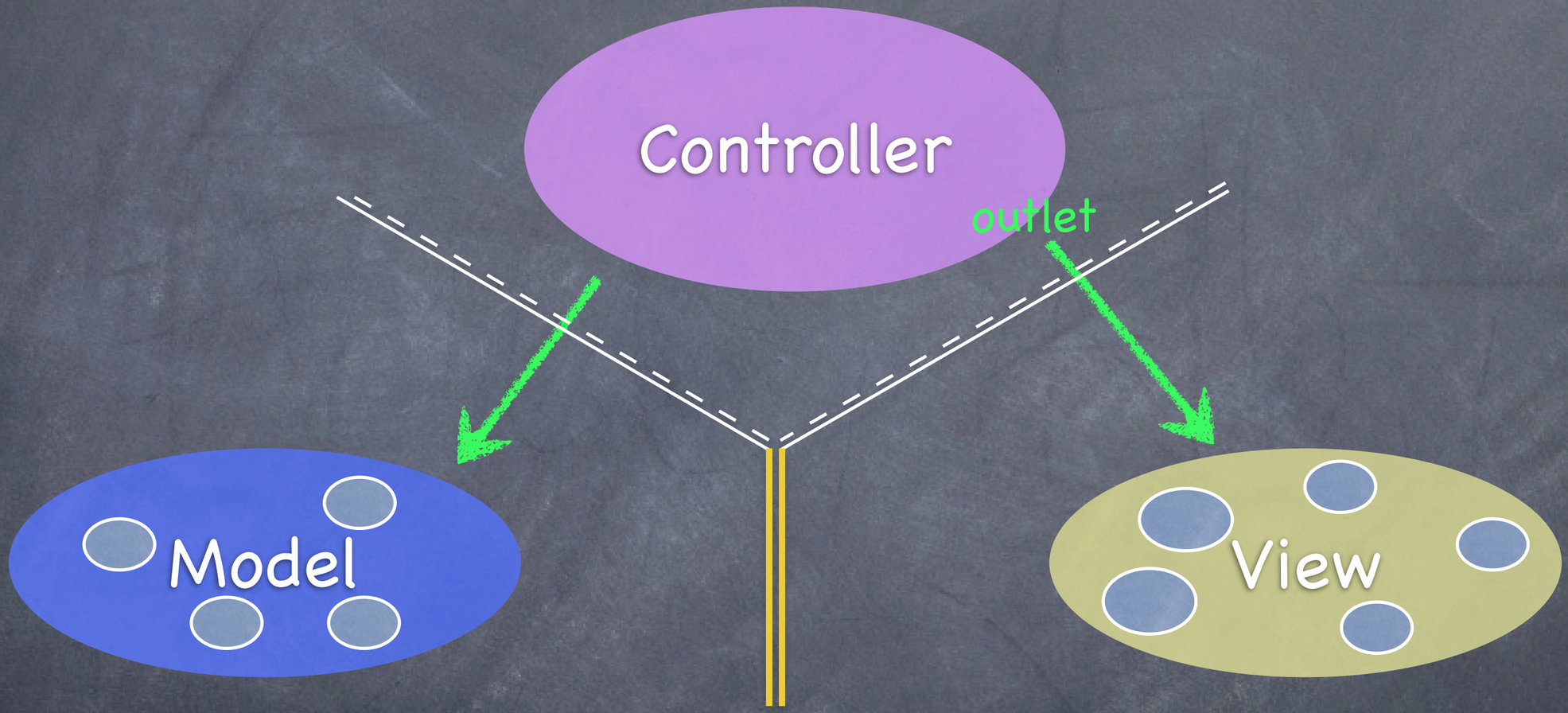

The Model and View should never speak to each other.

# MVC



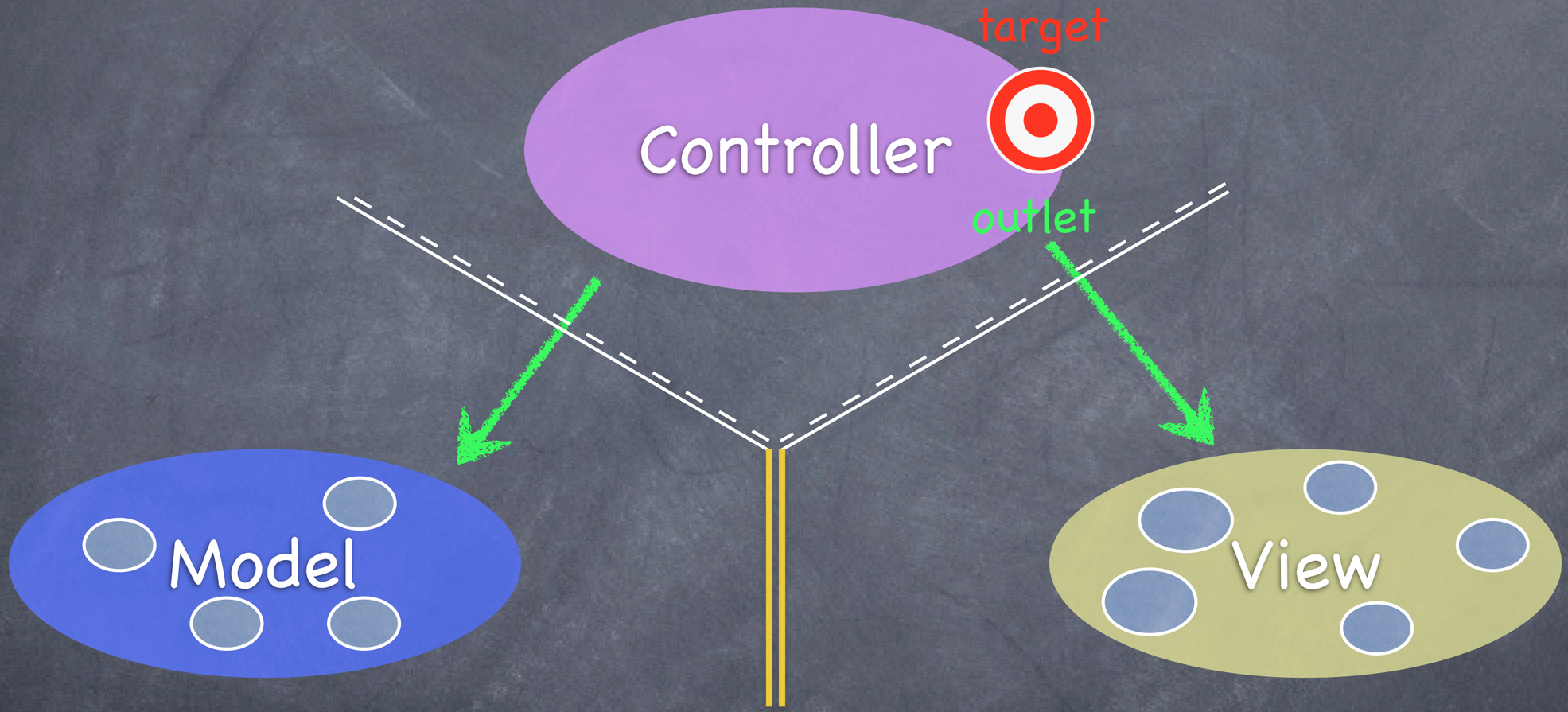

Can the View speak to its Controller?

# MVC



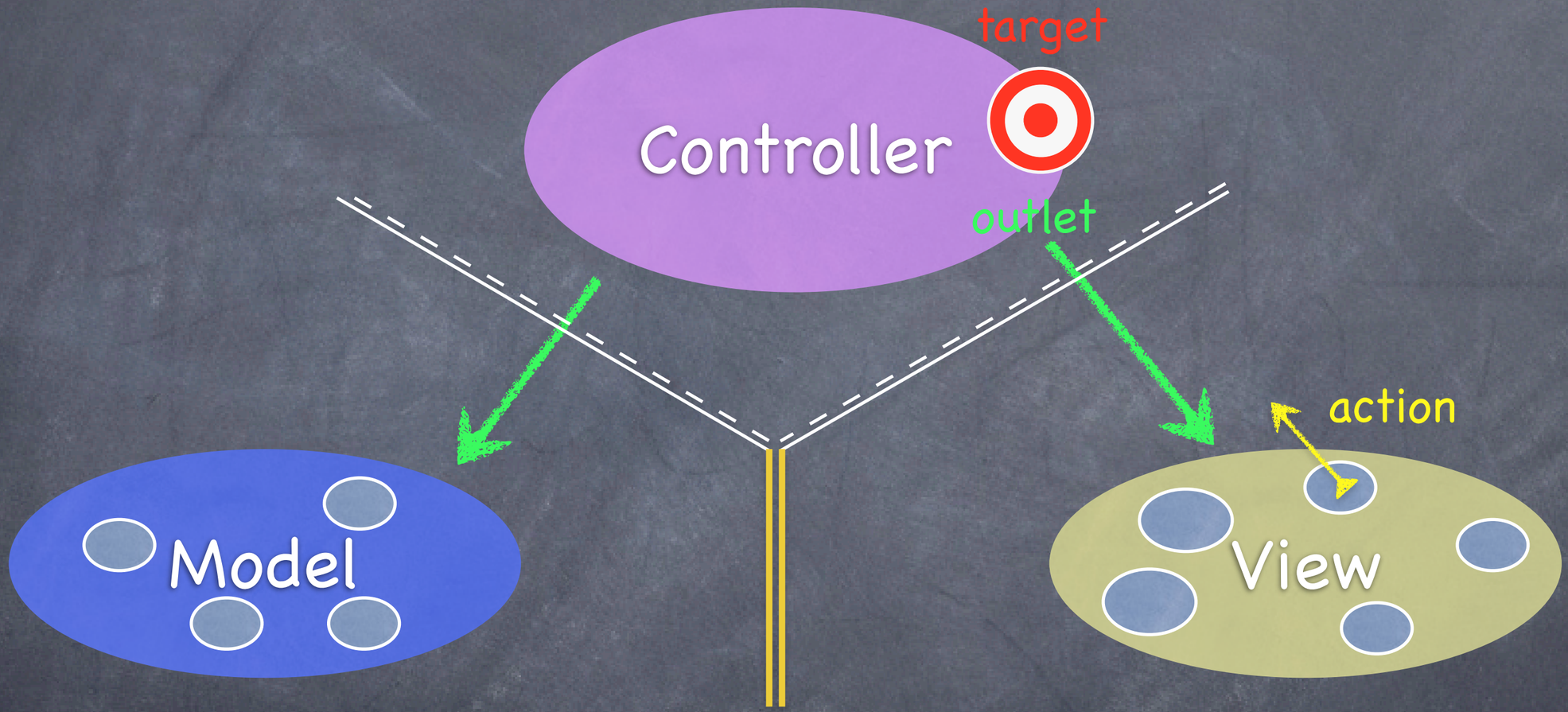

Sort of. Communication is "blind" and structured.

# MVC



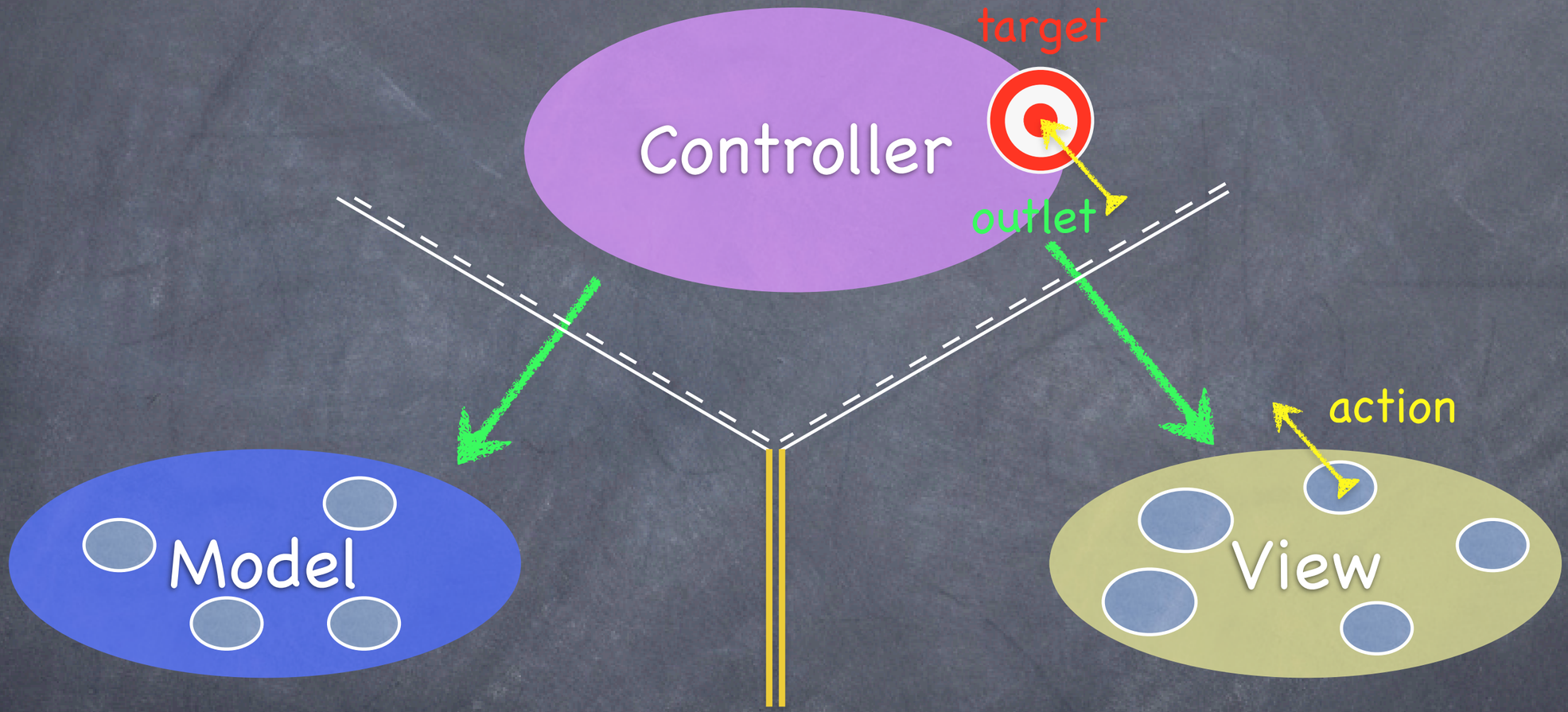

The Controller can drop a target on itself.

# MVC



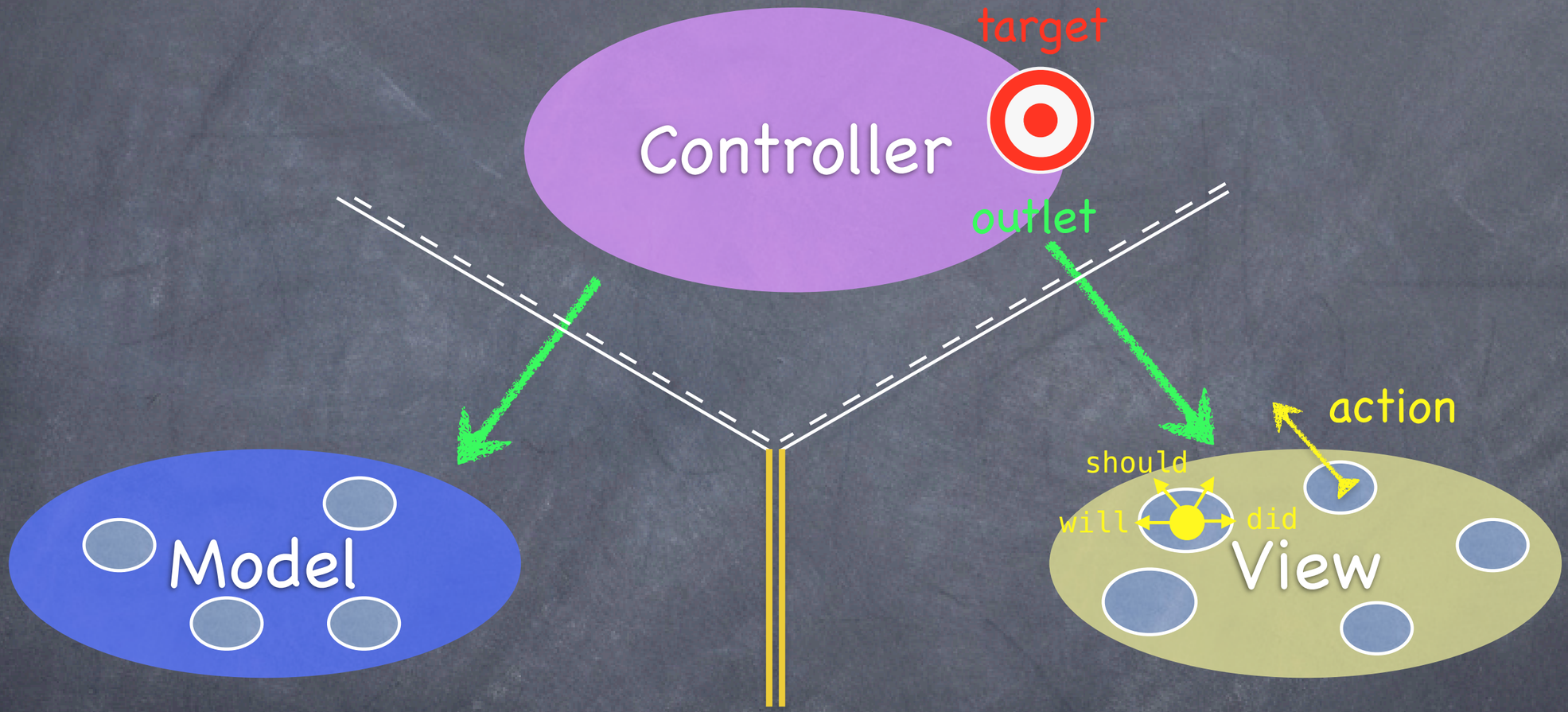

Then hand out an action to the View.

# MVC



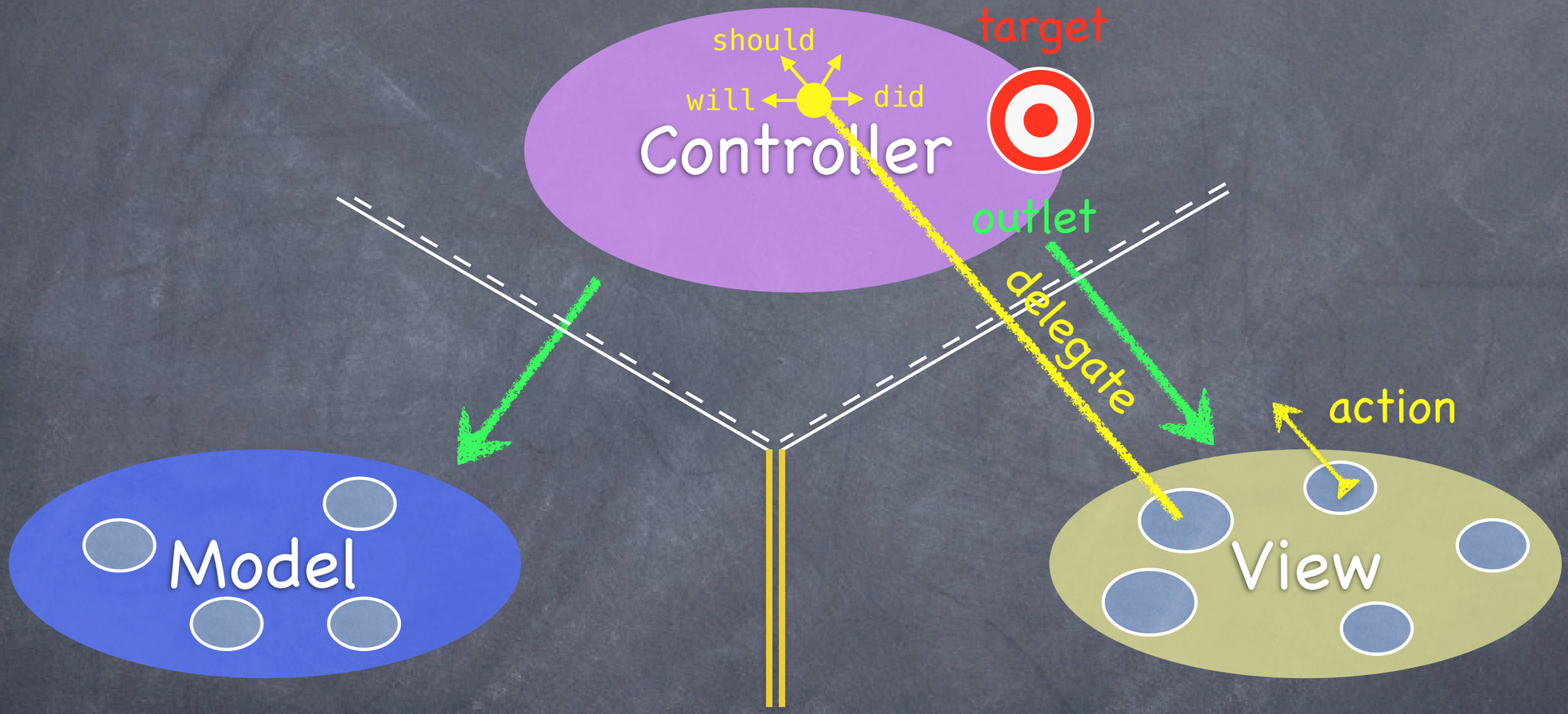

The View sends the action when things happen in the UI.

# MVC



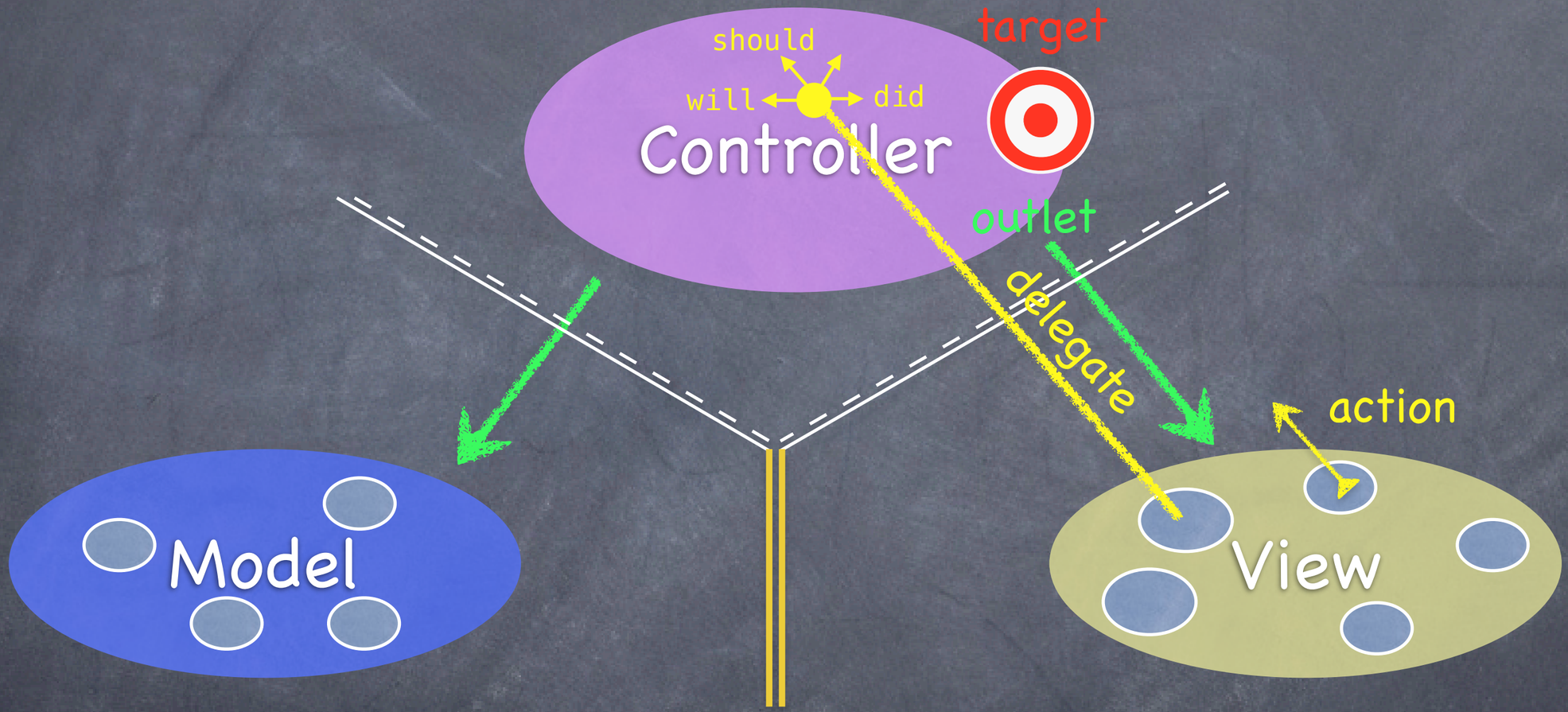

Sometimes the View needs to synchronize with the Controller.

# MVC



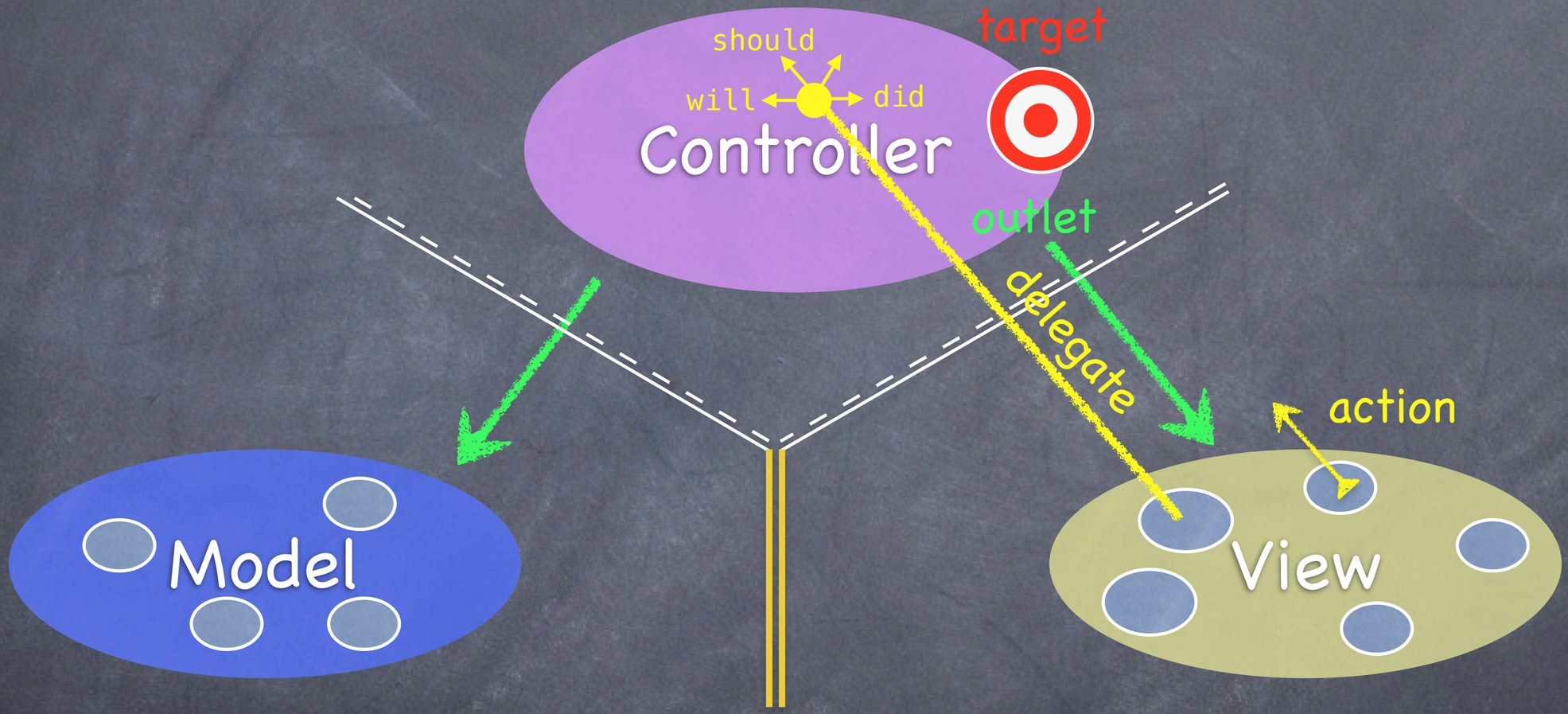

The Controller sets itself as the View's delegate.

# MVC



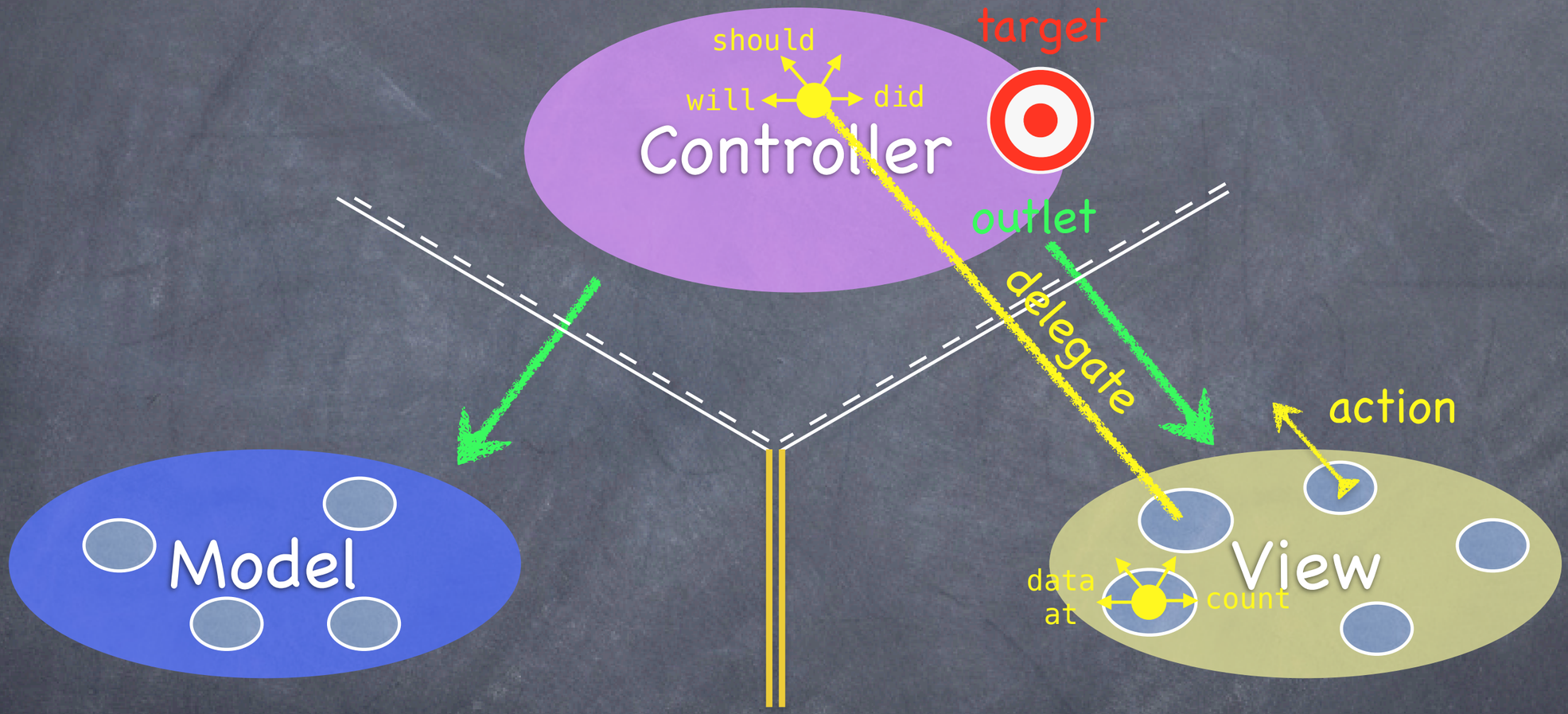

The delegate is set via a protocol (i.e. it's "blind" to class).

# MVC



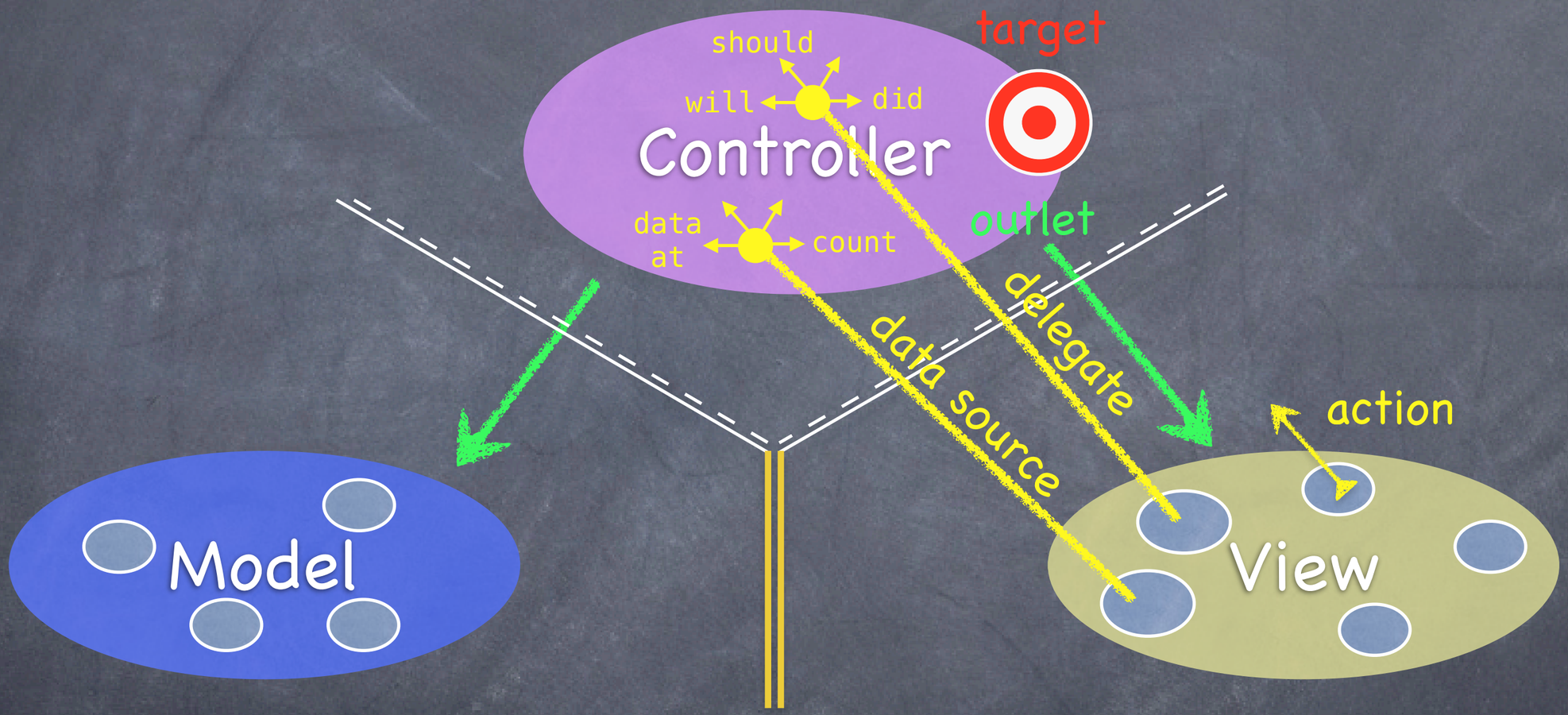

Views do not own the data they display.

# MVC



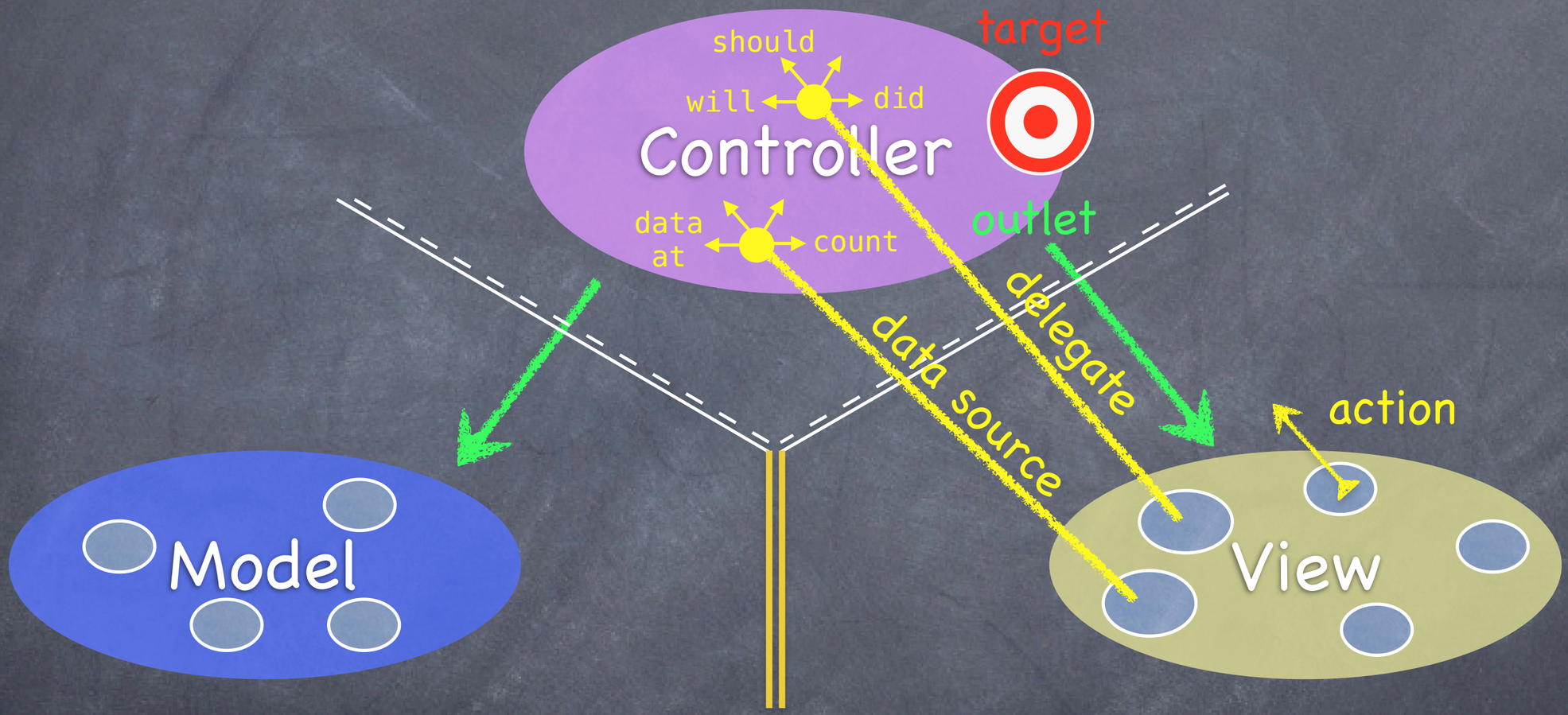

So, if needed, they have a protocol to acquire it.

# MVC



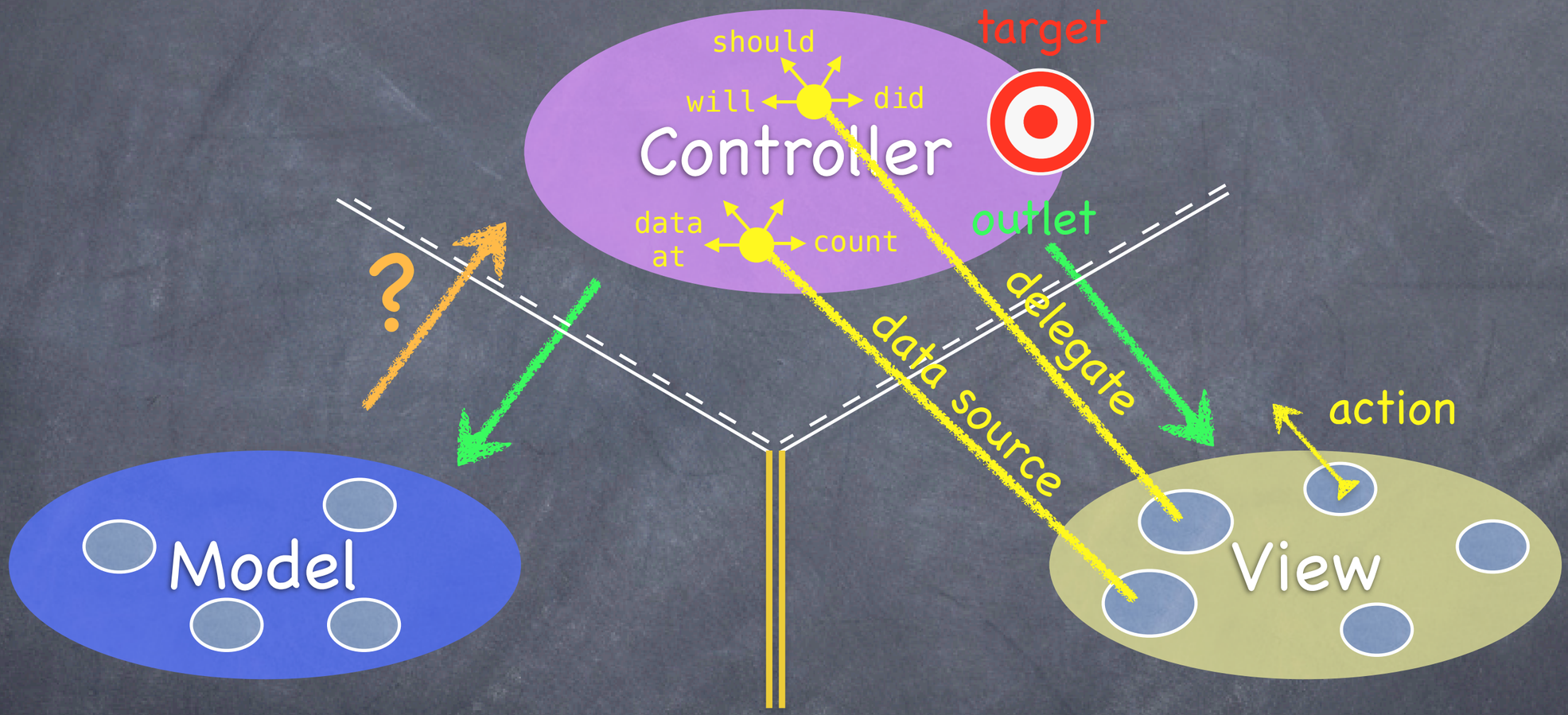

Controllers are almost always that data source (not Model!).

# MVC



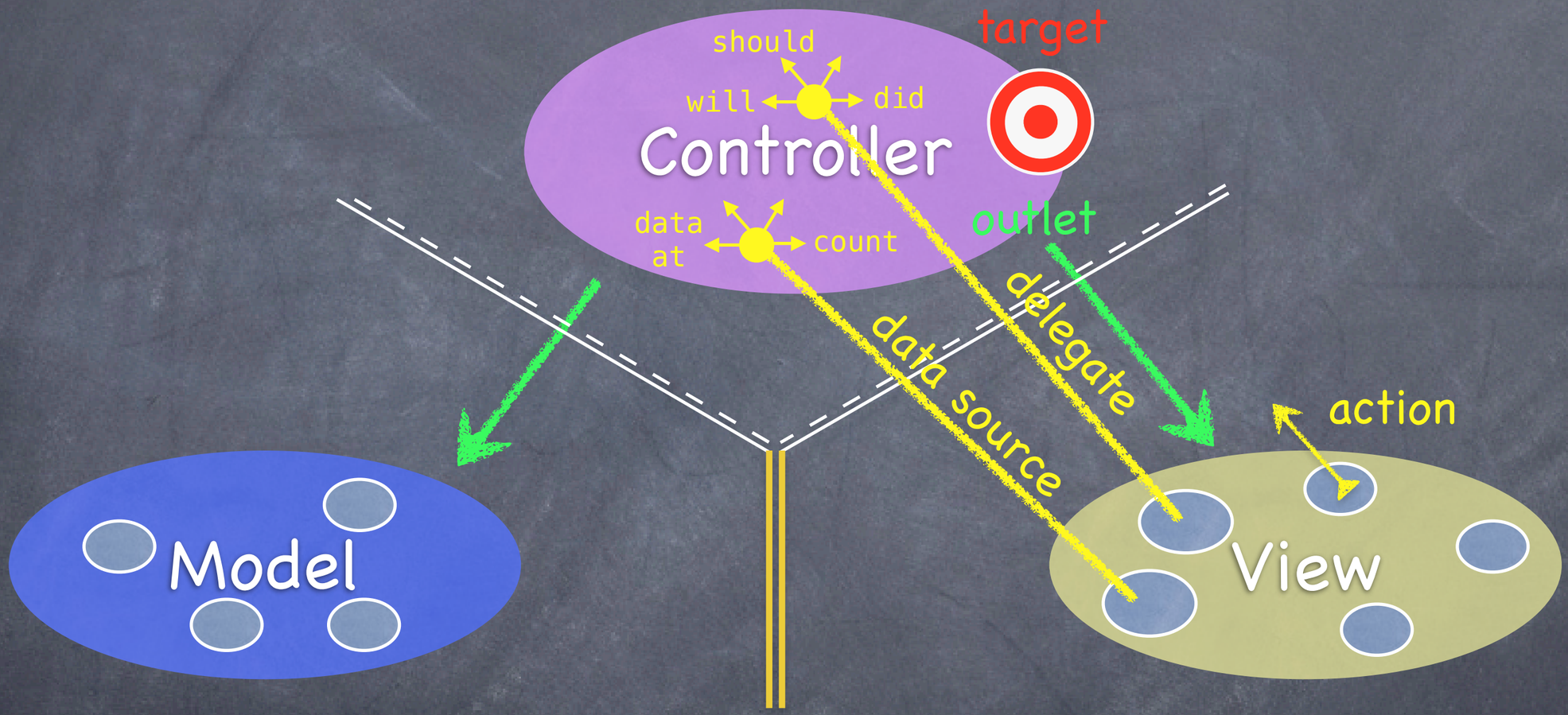

Controllers interpret/format Model information for the View.

# MVC



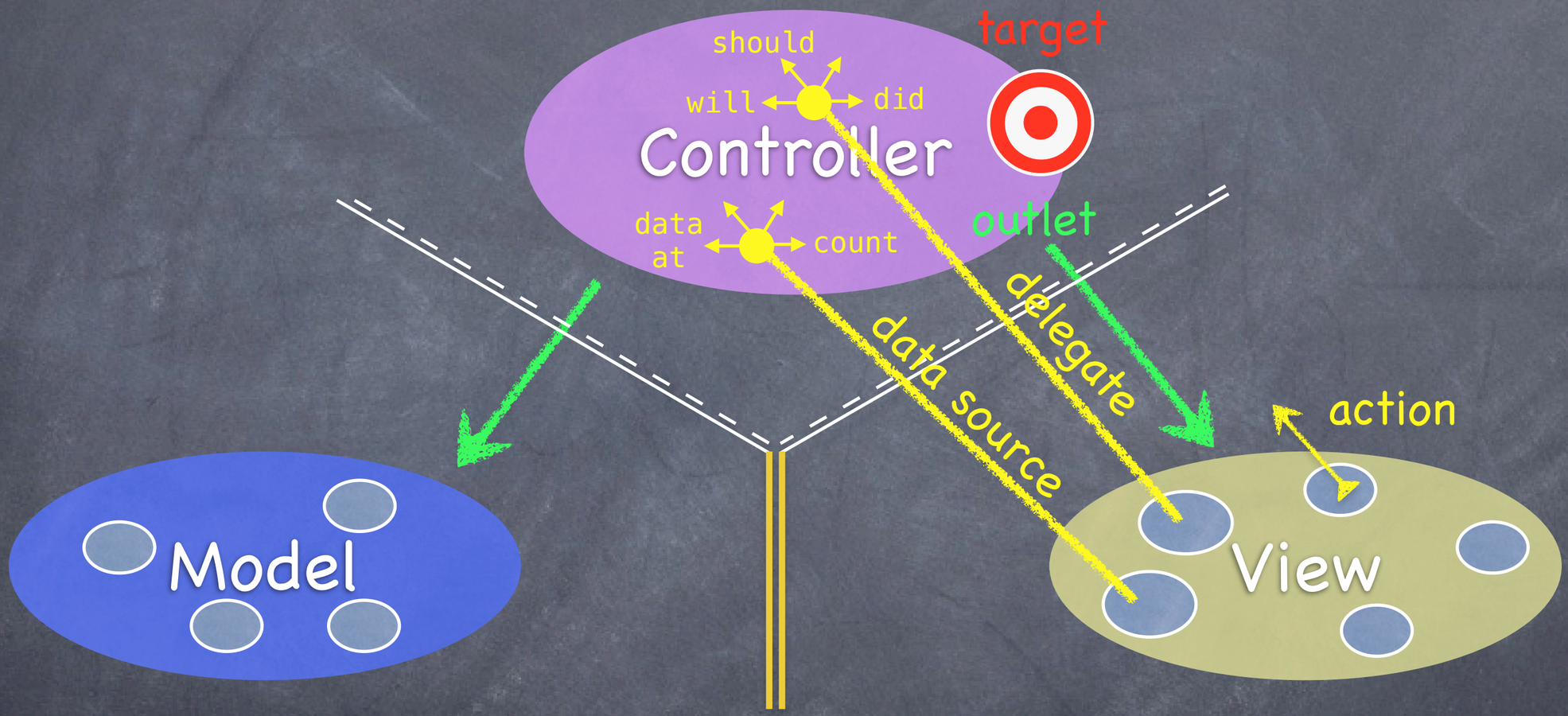

Can the Model talk directly to the Controller?

# MVC



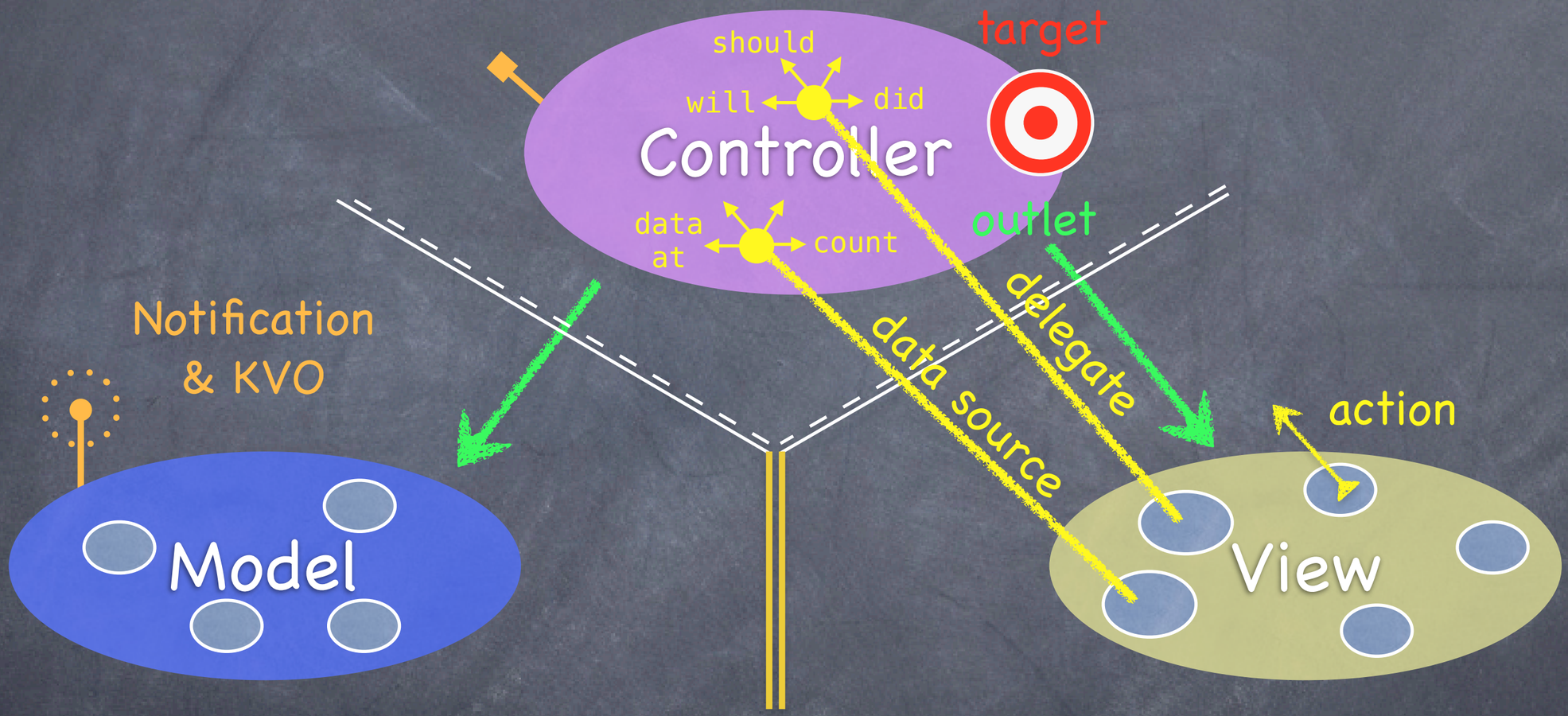

No. The Model is (should be) UI independent.

# MVC



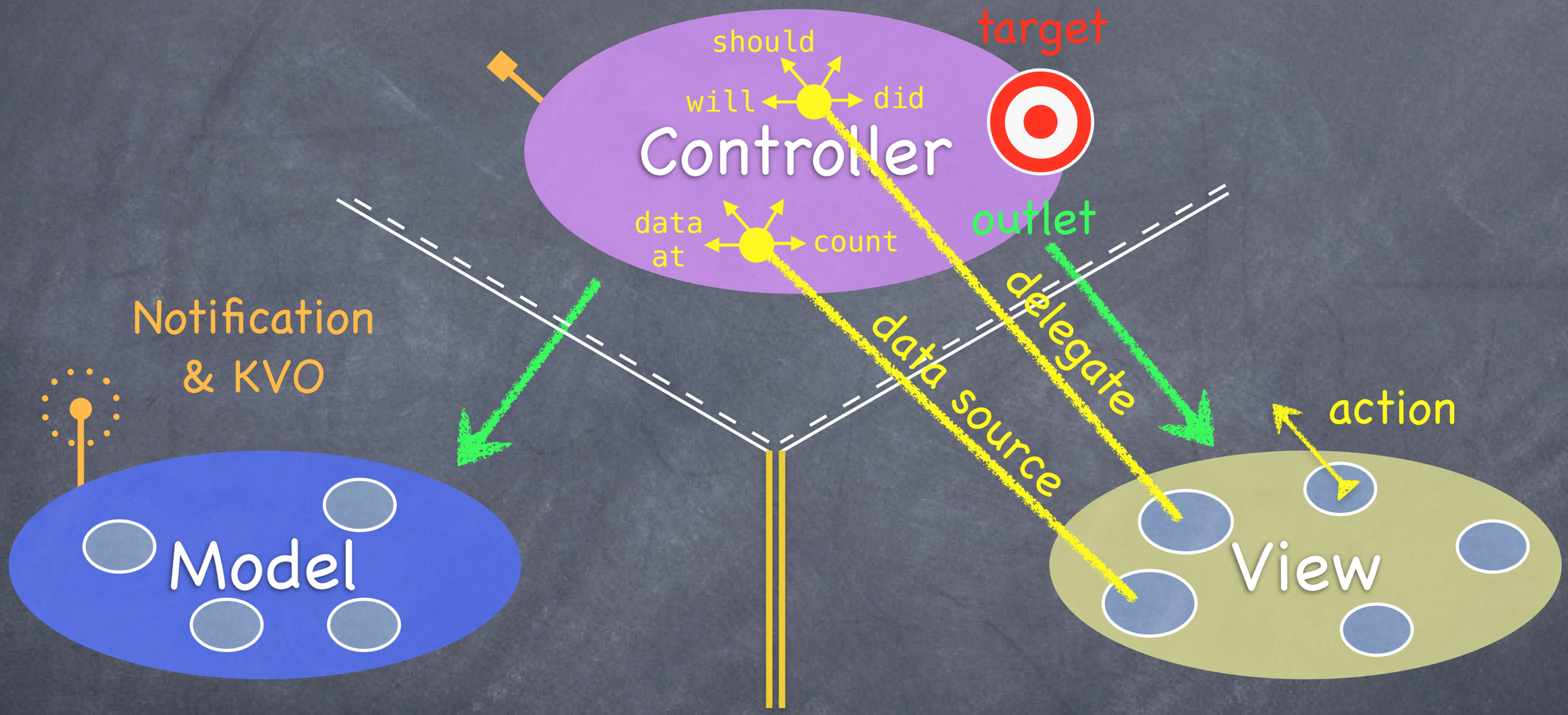

So what if the Model has information to update or something?

# MVC



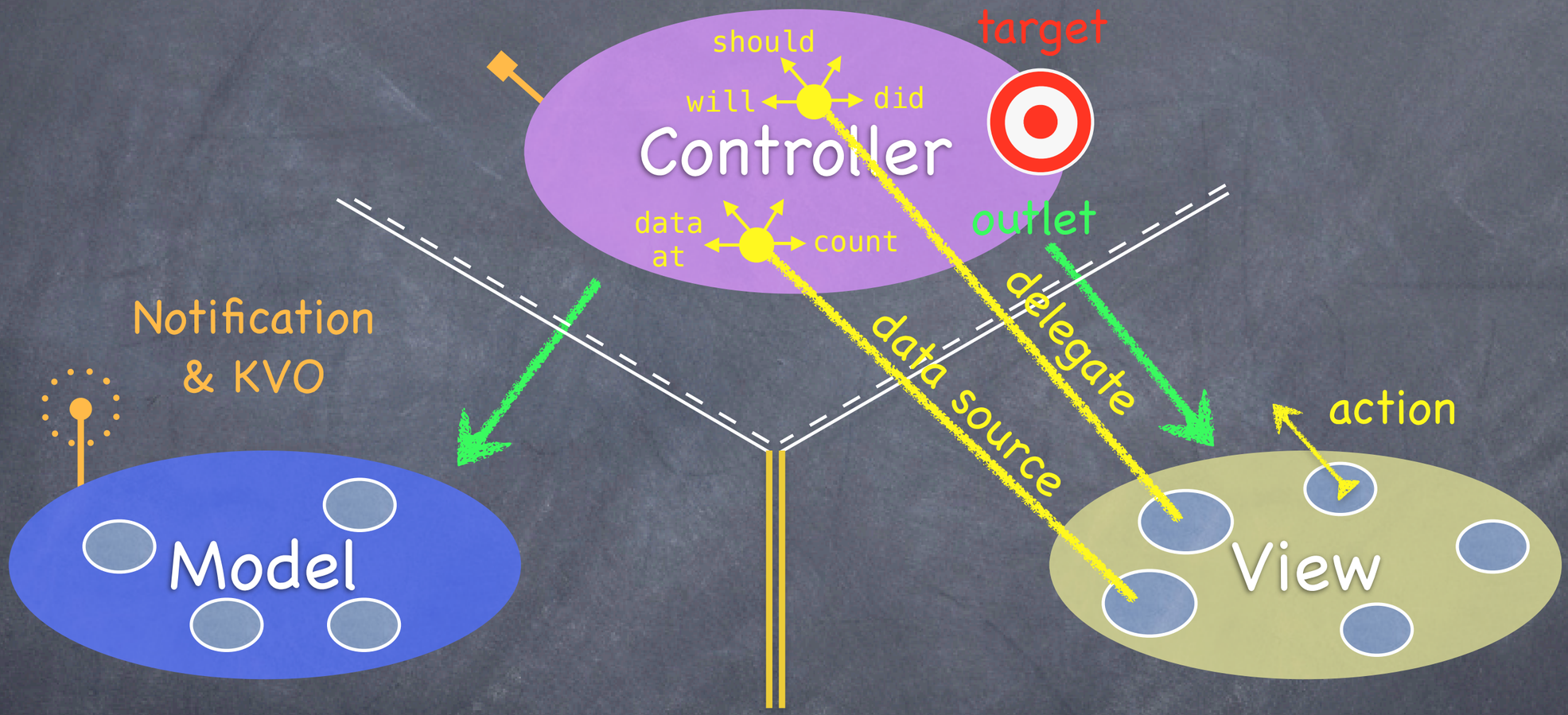

It uses a "radio station"-like broadcast mechanism.

# MVC



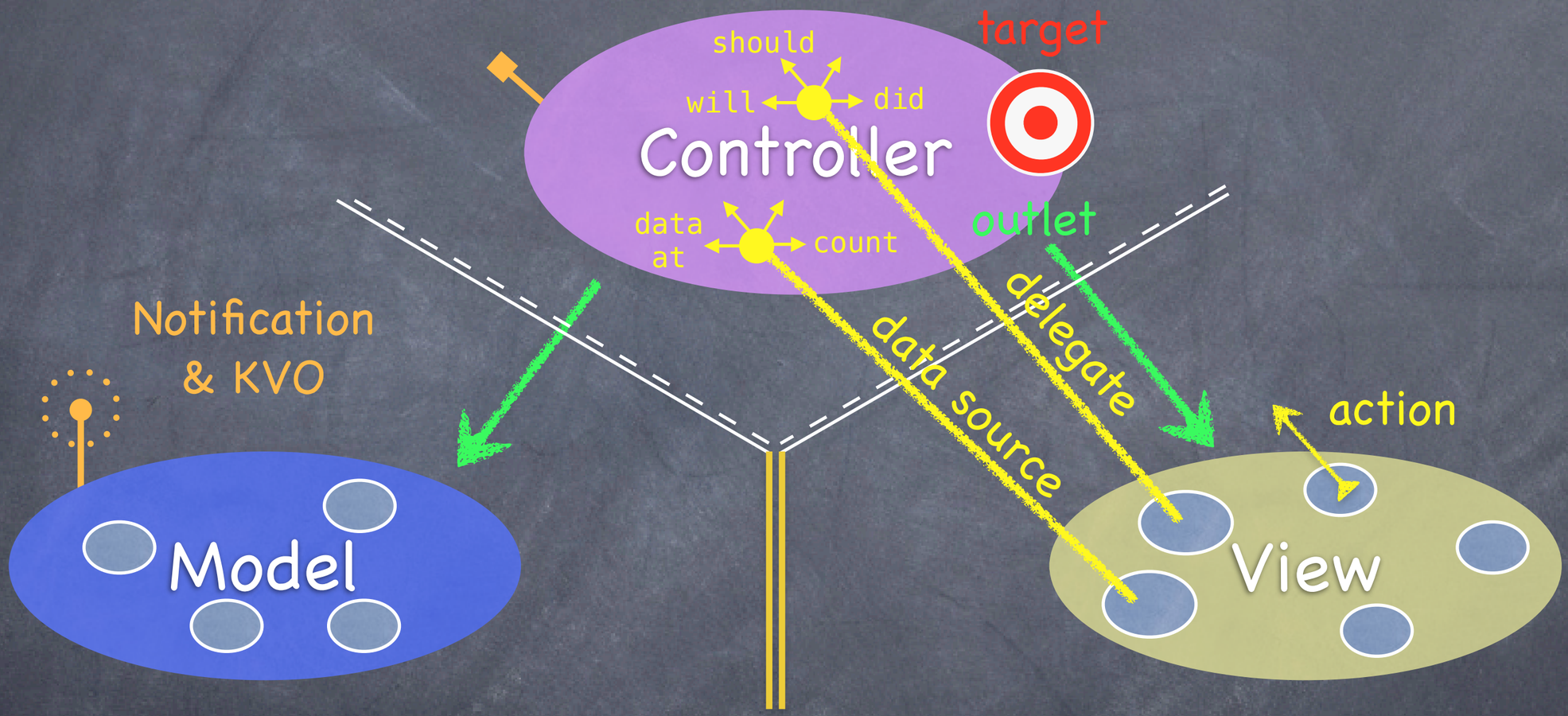

Controllers (or other Model) "tune in" to interesting stuff.

# MVC



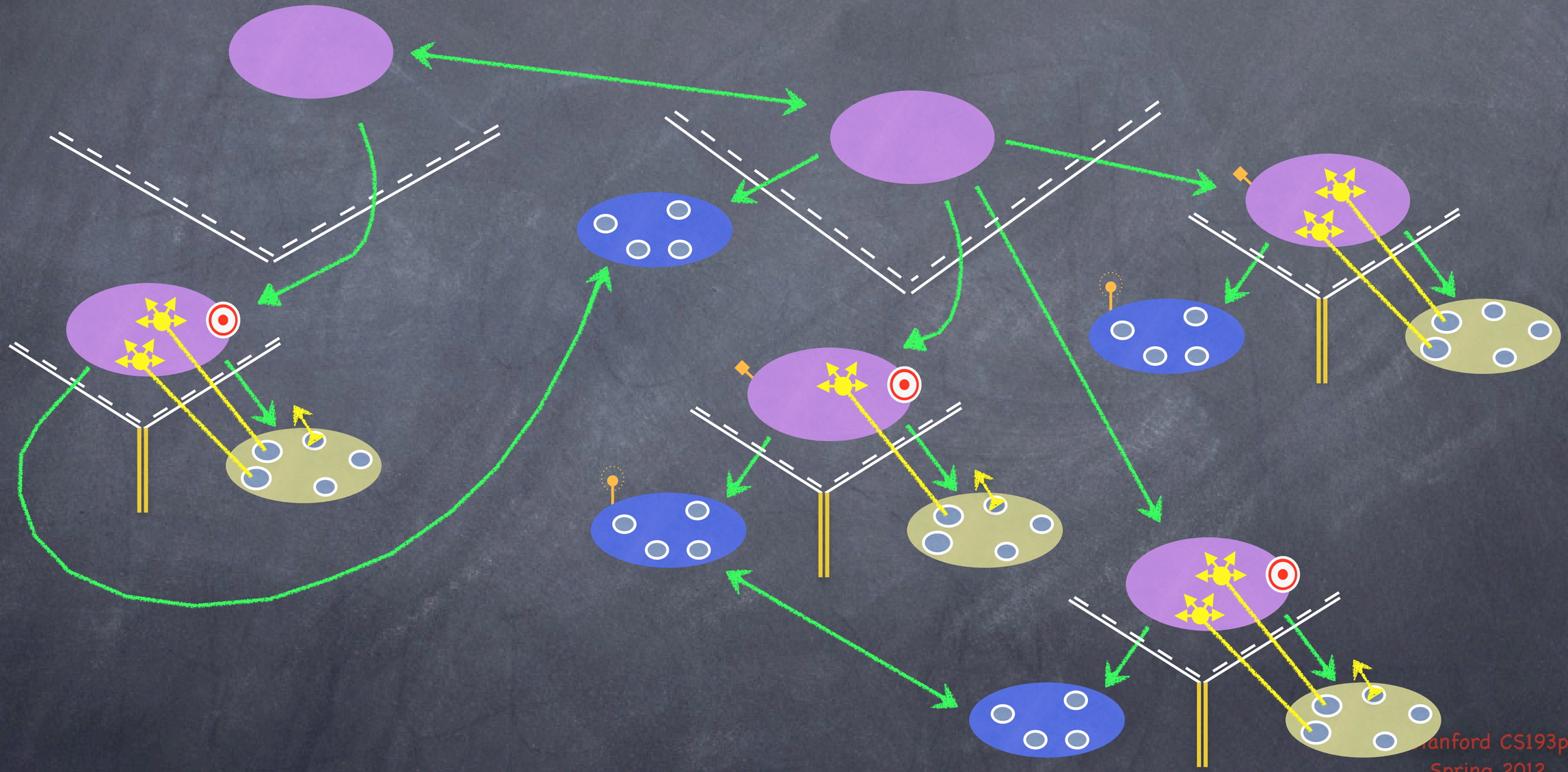

A View might "tune in," but probably not to a Model's "station."

# MVC

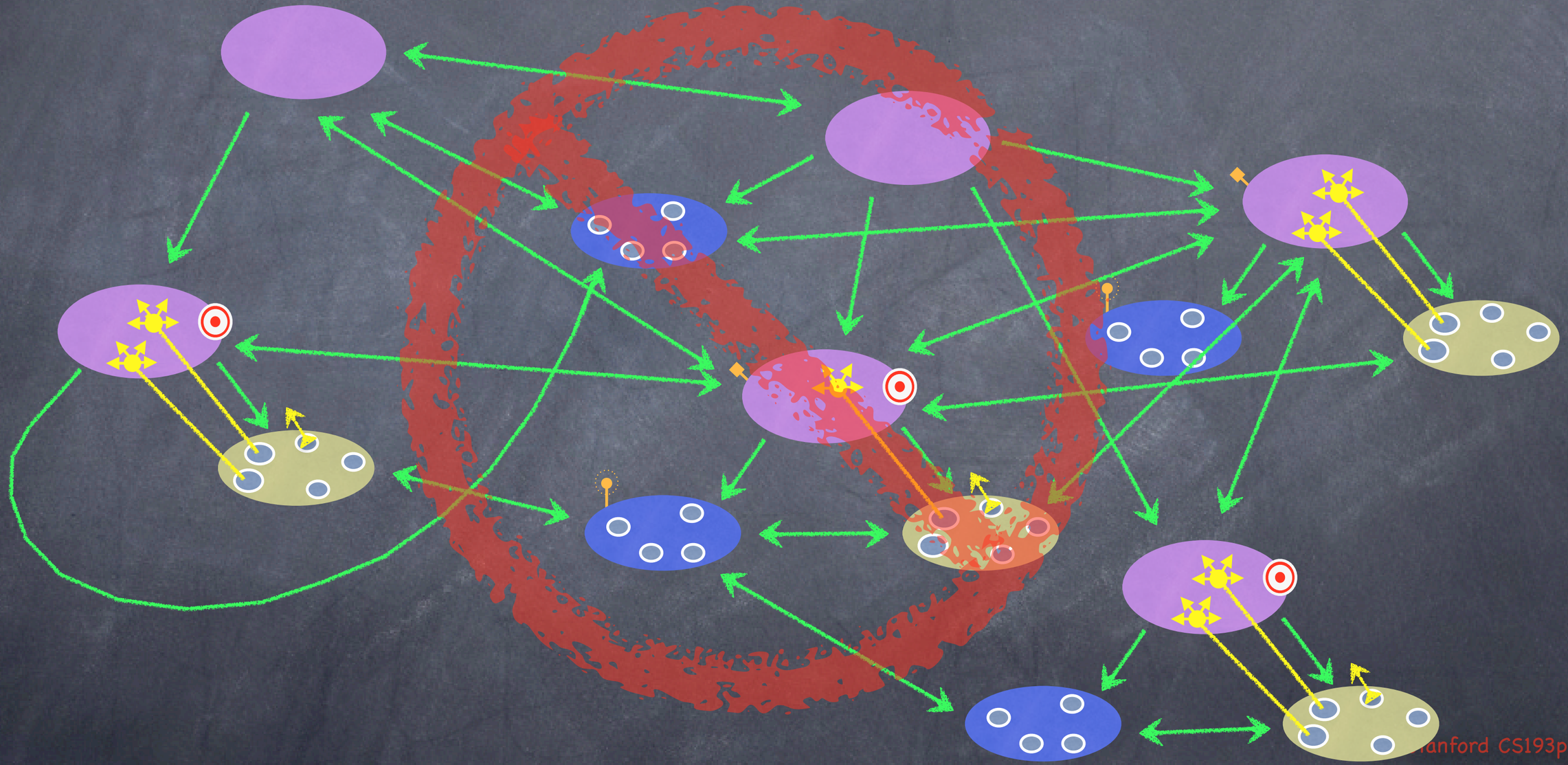



Now combine MVC groups to make complicated programs ...

# MVCs working together



# MVCs not working together



# Objective-C

- New language to learn!
  - Strict superset of C
  - Adds syntax for classes, methods, etc.
  - A few things to "think differently" about (e.g. properties, dynamic binding)
- Most important concept to understand today: Properties
  - Usually we do not access instance variables directly in Objective-C.
  - Instead, we use "properties."
  - A "property" is just the combination of a getter method and a setter method in a class.
  - The getter has the name of the property (e.g. "myValue")
  - The setter's name is "set" plus capitalized property name (e.g. "setMyValue:")
  - (To make this look nice, we always use a lowercase letter as the first letter of a property name.)
  - We just call the setter to store the value we want and the getter to get it. Simple.
- This is just your first glimpse of this language!
  - We'll go much more into the details next week.
  - Don't get too freaked out by the syntax at this point.

# Objective-C

Spaceship.h

```
#import "Vehicle.h"

@interface Spaceship : Vehicle

// declaration of public methods

@end
```

Superclass's header file. This is often <UIKit/UIKit.h> .

Class name

Superclass

Spaceship.m

```
#import "Spaceship.h"

@implementation Spaceship

// implementation of public and private methods

@end
```

Importing our own header file.

Note, superclass not specified here.

# Objective-C

Spaceship.h

```
#import "Vehicle.h"



@interface Spaceship : Vehicle

// declaration of public methods



@end
```

Spaceship.m

```
#import "Spaceship.h"
```

How do you declare a private method?

```
@implementation Spaceship

// implementation of public and private methods



@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"




@interface Spaceship : Vehicle

// declaration of public methods









@end
```

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods









@end
```

Don't forget the ().
(No superclass here either.)

# Objective-C

Spaceship.h

```
#import "Vehicle.h"



@interface Spaceship : Vehicle

// declaration of public methods


@end
```

So what does the declaration of a method look like?

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods


@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"



@interface Spaceship : Vehicle

// declaration of public methods



- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;

@end
```

Lining up the colons makes things look nice.

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@end
```

# Objective-C

## Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods



- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;



@end
```

We need to import `Planet.h` for the method declaration below to work.

The full name of this method is `orbitPlanet:atAltitude:`

It takes two arguments:
a pointer to a `Planet` object (`aPlanet`) & a `double` (`km`).
Note how each is preceded by its own part of the method name
(`orbitPlanet:` precedes the `aPlanet` argument and
`atAltitude:` precedes the `km` argument).

It does not return any value.

## Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods



@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods



- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;









@end
```

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods









@end
```

All objects are always allocated on the heap.
So we always access them through a pointer. Always.

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods



- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;









@end
```

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods
```

What does the implementation of this method look like?

```
@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods



- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;









@end
```

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods









- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)altitude
{
}


@end
```

No semicolon here.

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods



- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;









@end
```

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods









- (void)orbitPlanet:(Planet *)aPlanet
          atAltitude:(double)altitude
{
}


@end
```

Note that the names of the argument variables can be different than in header file.

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods



- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;









@end
```

We'll look at some code here a little later.

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods









- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)altitude
{
}


@end
```

# Objective-C

## Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods



- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;
```

Let's declare a couple of more public methods.

```
@end
```

## Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods









- (void)orbitPlanet:(Planet *)aPlanet
          atAltitude:(double)altitude
{
}


@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods



- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;

- (void)setTopSpeed:(double)percentSpeedOfLight;
- (double)topSpeed;



@end
```

These just set and get the top speed that this Spaceship is allowed to go.

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods









- (void)orbitPlanet:(Planet *)aPlanet
          atAltitude:(double)altitude
{
}


@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods



- (void)orbitPlanet:(Planet *)aPlanet
        atAltitude:(double)km;

- (void)setTopSpeed:(double)percentSpeedOfLight;
- (double)topSpeed;
```

These just set and get the top speed that this Spaceship is allowed to go.

```
@end
```

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods
```

We're only going to stub out the implementation for now because we're going to get the compiler to write them for us in a moment.

```
- (void)orbitPlanet:(Planet *)aPlanet
          atAltitude:(double)altitude
{
}


@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods



- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;

- (void)setTopSpeed:(double)percentSpeedOfLight;
- (double)topSpeed;









@end
```

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods



- (void)setTopSpeed:(double)speed
{
}

- (double)topSpeed
{
}

- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)altitude
{
}


@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
        atAltitude:(double)km;

- (void)setTopSpeed:(double)percentSpeedOfLight;
- (double)topSpeed;



@end
```

Both for declaration.

It is so common to have "setter and getter" combos like this that Objective-C has special syntax to support it.

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;

- (void)setTopSpeed:(double)speed
{
}

- (double)topSpeed
{
}

- (void)orbitPlanet:(Planet *)aPlanet
          atAltitude:(double)altitude
{
}



@end
```

And for implementation.

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
        atAltitude:(double)km;

- (void)setTopSpeed:(double)percentSpeedOfLight;
- (double)topSpeed;



@end
```

Both for declaration.

When you use @property, you do not need to declare the setter or getter explicitly.

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;

- (void)setTopSpeed:(double)speed
{
}

- (double)topSpeed
{
}

- (void)orbitPlanet:(Planet *)aPlanet
          atAltitude:(double)altitude
{
}



@end
```

And for implementation.

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;









@end
```

Both for declaration.

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;

- (void)setTopSpeed:(double)speed
{
}

- (double)topSpeed
{
}

- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)altitude
{
}



@end
```

And for implementation.

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;




@end
```

Both for declaration.

And when you use @synthesize, you do not need to implement the setter or getter.

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;

- (void)setTopSpeed:(double)speed
{
}

- (double)topSpeed
{
}

- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)altitude
{
}


@end
```

And for implementation.

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;









@end
```

Both for declaration.

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;









- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)altitude
{
}


@end
```

And for implementation.

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;









@end
```

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;
```

This (_topSpeed) is the name of the storage location (instance variable) that @synthesize will use to store the top speed.

```
- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)altitude
{
}


@end
```

# Objective-C

## Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;









@end
```

## Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;
```

_ (underbar) then the name of the property is a common naming convention.

If we don't use = here, @synthesize uses the name of the property (which is bad so always use =).

```
- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)altitude
{
}


@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;









@end
```

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;
```

Even if you use `@synthesize`, you are allowed to implement the setter and/or getter yourself. Your implementation will "win".

```
- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)altitude
{
}


@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;









@end
```

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;

- (void)setTopSpeed:(double)speed
{
    if ((speed < 1) && (speed > 0)) _topSpeed = speed;
}
```

You should only reference _topSpeed in your implementation of its setter and getter!
To access the top speed elsewhere in your code, call the setter or getter method.

```
- (void)orbitPlanet:(Planet *)aPlanet
          atAltitude:(double)altitude
{
}


@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;



@end
```

Let's take a quick look at a sample implementation of orbitPlanet:atAltitude:.

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;

- (void)setTopSpeed:(double)speed
{
    if ((speed < 1) && (speed > 0)) _topSpeed = speed;
}

- (void)orbitPlanet:(Planet *)aPlanet
          atAltitude:(double)altitude
{




}

@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;









@end
```

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;

- (void)setTopSpeed:(double)speed
{
    if ((speed < 1) && (speed > 0)) _topSpeed = speed;
}

- (void)orbitPlanet:(Planet *)aPlanet
          atAltitude:(double)altitude
{
    double speed = [self topSpeed];
    [self travelToPlanet:aPlanet atSpeed:speed];


}

@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;









@end
```

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;

- (void)setTopSpeed:(double)speed
{
    if ((speed < 1) && (speed > 0)) _topSpeed = speed;
}

- (void)orbitPlanet:(Planet *)aPlanet
          atAltitude:(double)altitude
{
    double speed = self.topSpeed; // [self topSpeed]
    [self travelToPlanet:aPlanet atSpeed:speed];


}

@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;









@end
```

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;

- (void)setTopSpeed:(double)speed
{
    if ((speed < 1) && (speed > 0)) _topSpeed = speed;
}

- (void)orbitPlanet:(Planet *)aPlanet
          atAltitude:(double)altitude
{
    double speed = self.topSpeed; // [self topSpeed]
    [self travelToPlanet:aPlanet atSpeed:speed];
    self.topSpeed = [aPlanet speedLimit];

}

@end
```

# Objective-C

Spaceship.h

```
#import "Vehicle.h"
#import "Planet.h"



@interface Spaceship : Vehicle

// declaration of public methods

@property (nonatomic) double topSpeed;

- (void)orbitPlanet:(Planet *)aPlanet
         atAltitude:(double)km;









@end
```

Spaceship.m

```
#import "Spaceship.h"

@interface Spaceship()
// declaration of private methods (as needed)
@end

@implementation Spaceship

// implementation of public and private methods

@synthesize topSpeed = _topSpeed;

- (void)setTopSpeed:(double)speed
{
    if ((speed < 1) && (speed > 0)) _topSpeed = speed;
}

- (void)orbitPlanet:(Planet *)aPlanet
          atAltitude:(double)altitude
{
    double speed = self.topSpeed; // [self topSpeed]
    [self travelToPlanet:aPlanet atSpeed:speed];
    self.topSpeed = aPlanet.speedLimit;

}

@end
```

# Coming Up

- Next Lecture
  - Overview of the Integrated Development Environment (IDE, i.e. Xcode 4)
  - Objective-C in action
  - Concrete example of MVC
  - Major demonstration of all of the above: RPN Calculator
    (HOMEWORK: if you do not know what an RPN Calculator is, look it up on the internet.)
- Next Week
  - Objective-C language in depth
  - Foundation classes: arrays, dictionaries, strings, etc.
  - Dynamic vs. static typing
  - Protocols, categories and much, much more!