

# MVC

iOS Club

刘安博 QQ: 330332717



# MVC



Controller

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' containing four small white circles. To the right of the 'Model' oval is a greenish-yellow oval labeled 'View' containing five small blue circles. The entire diagram is set against a dark blue, chalkboard-like background.

Model

View

Divide objects in your program into 3 “camps.”



# MVC



Controller

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' containing four small white circles. To the right of the Model is a greenish-yellow oval labeled 'View' containing five small blue circles. The Controller is positioned centrally above the other two components.

Model

View

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



# MVC



Controller

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' containing four small white circles. To the right of the 'Model' oval is a greenish-yellow oval labeled 'View' containing five small blue circles. The background is a dark, textured grey.

Model

View

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



# MVC



Controller

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' containing four small white circles. To the right of the 'Model' oval is a greenish-yellow oval labeled 'View' containing five small blue circles. The entire diagram is set against a dark, textured background.

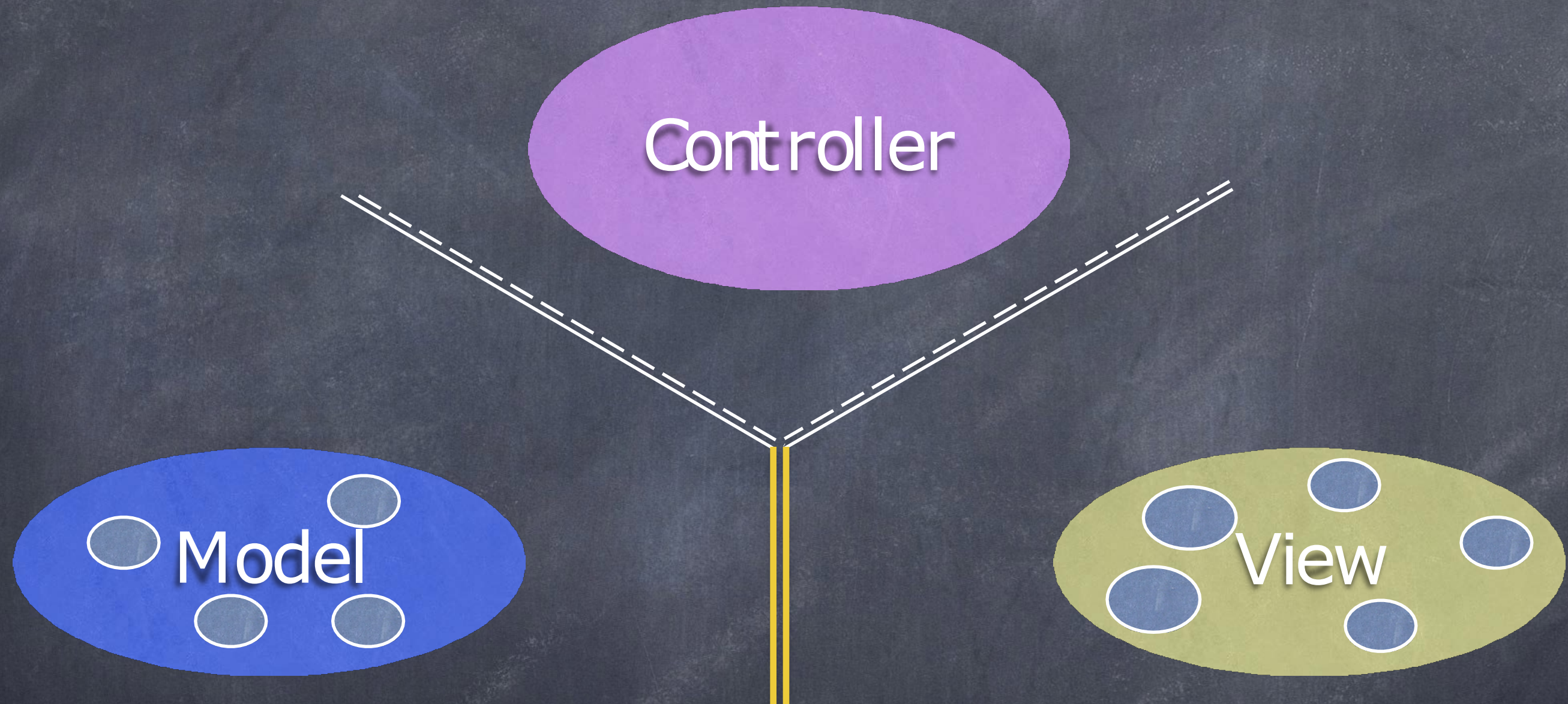
Model

View

View = Your Controller's minions



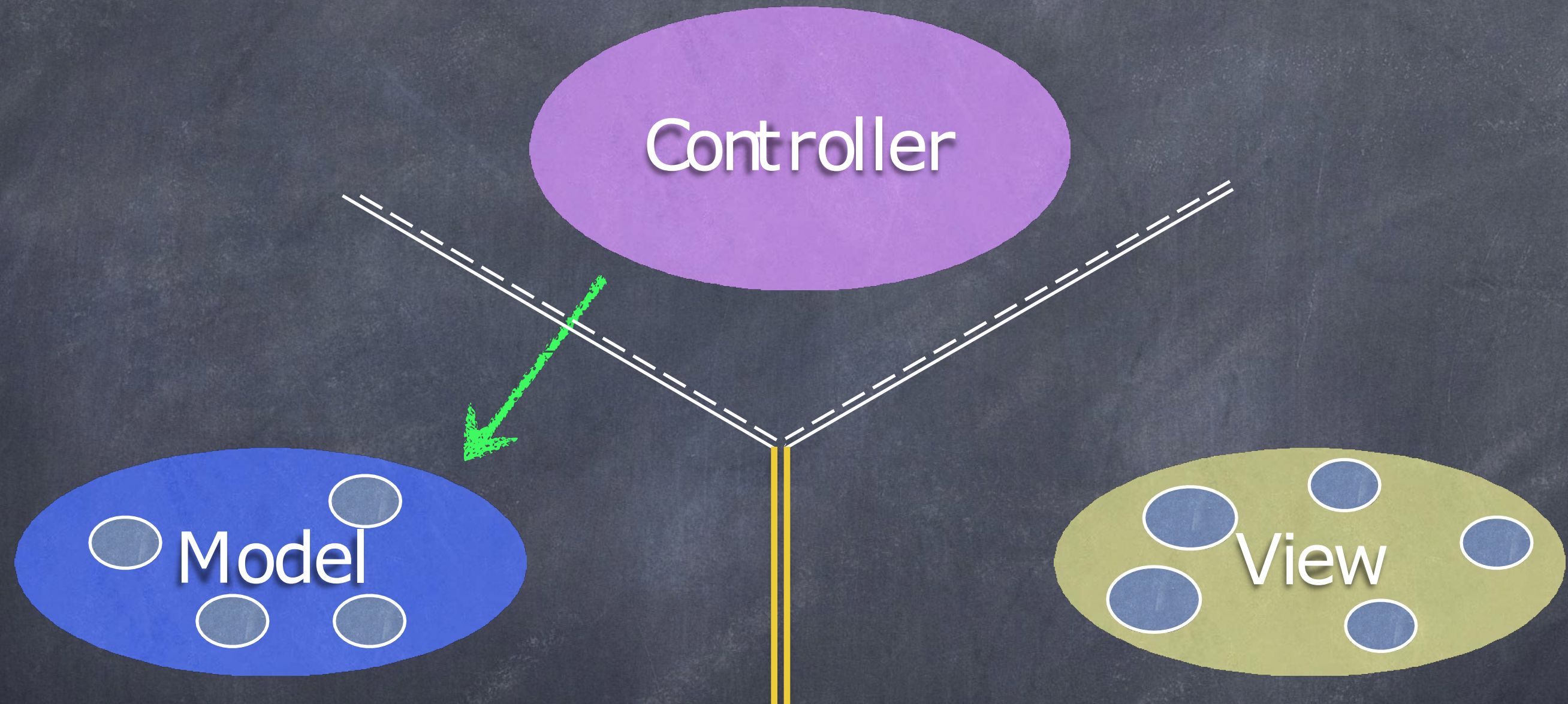
# MVC



It's all about managing communication between camps



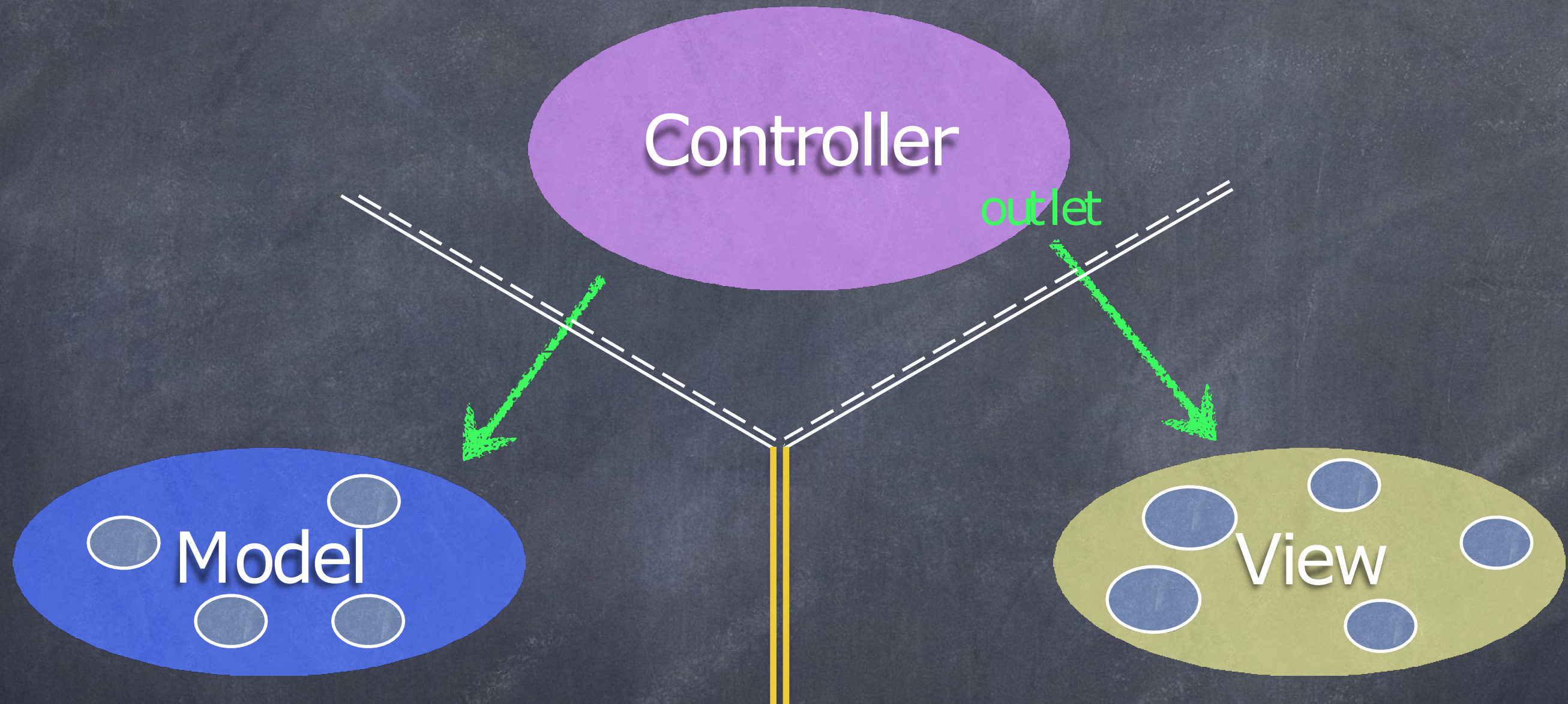
# MVC



Controllers can always talk directly to their Model.



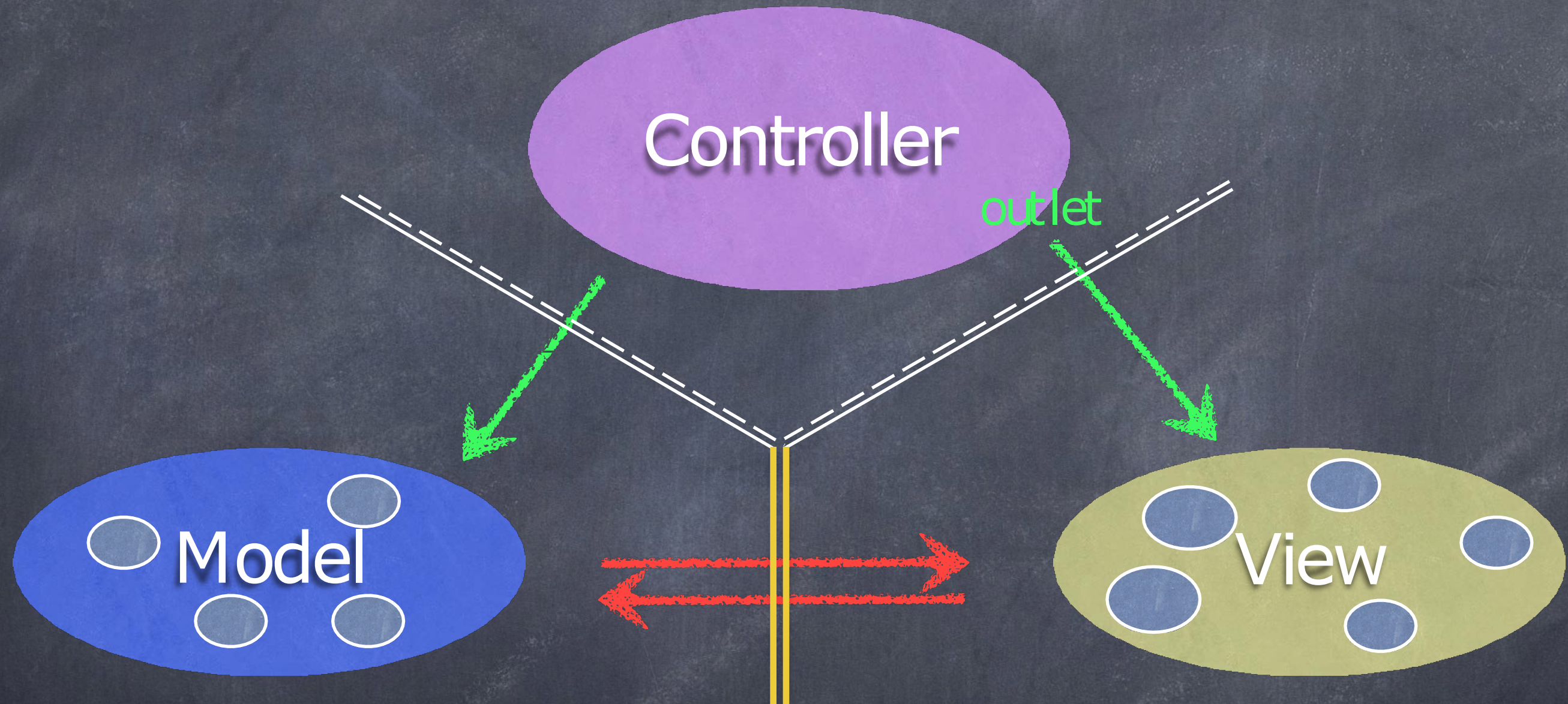
# MVC



Controllers can also talk directly to their View.



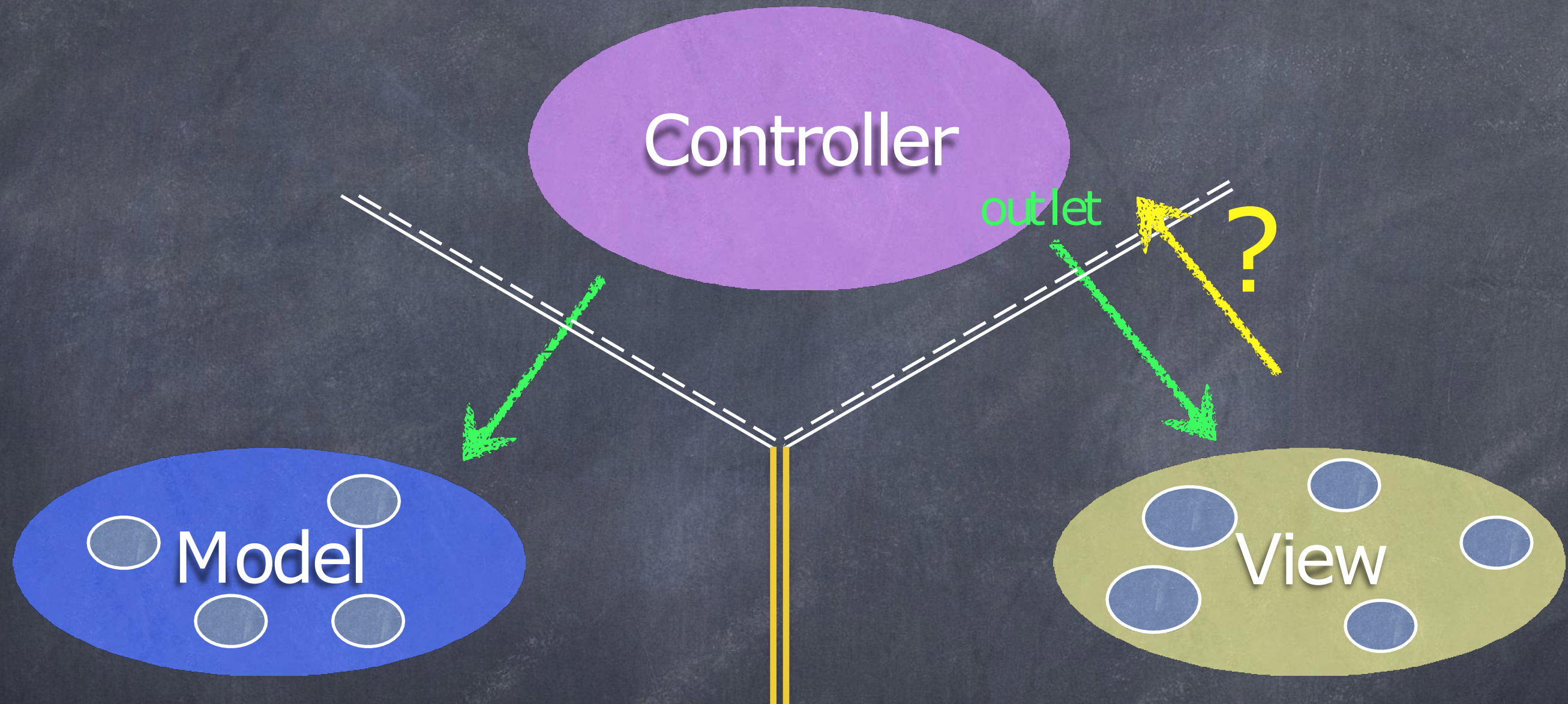
# MVC



The Model and View should never speak to each other.



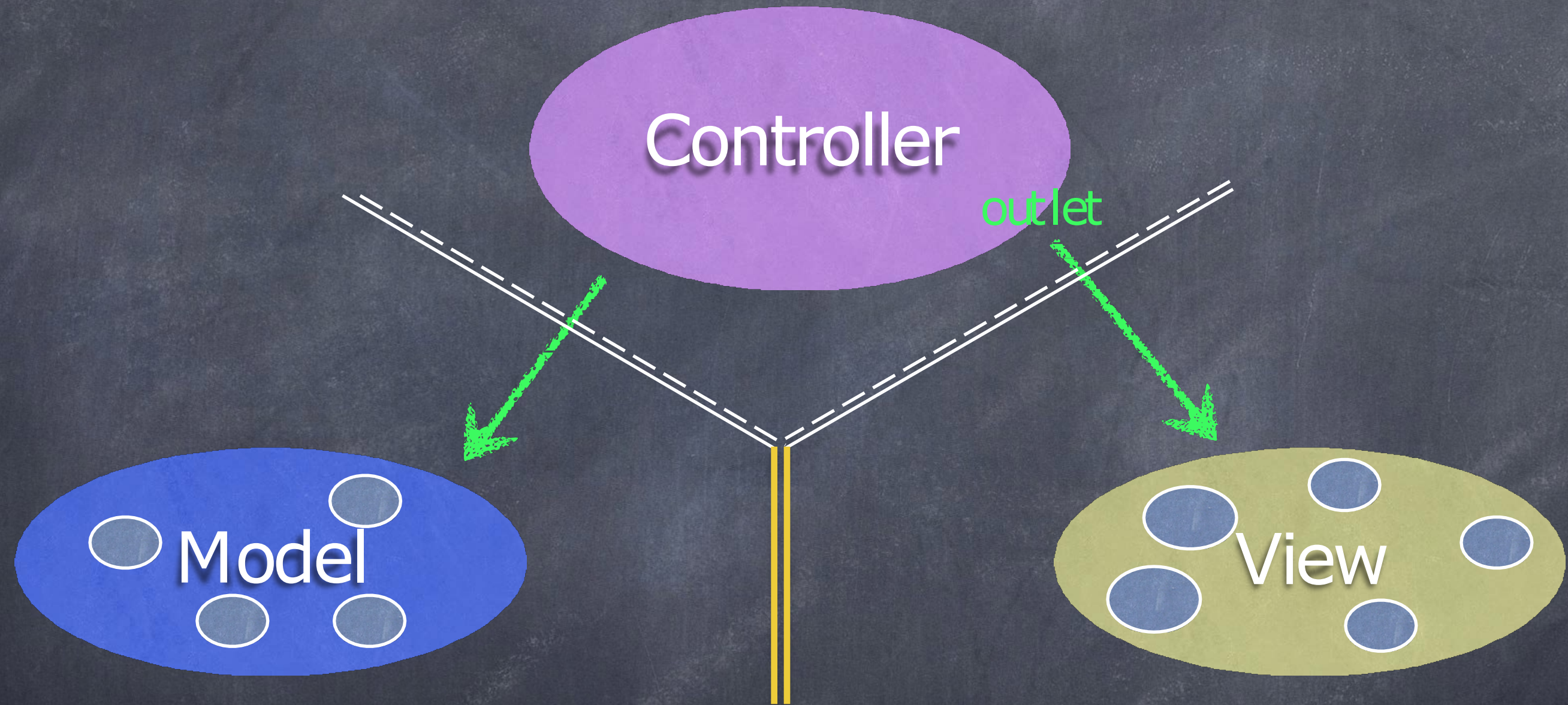
# MVC



Can the **View** speak to its **Controller**?



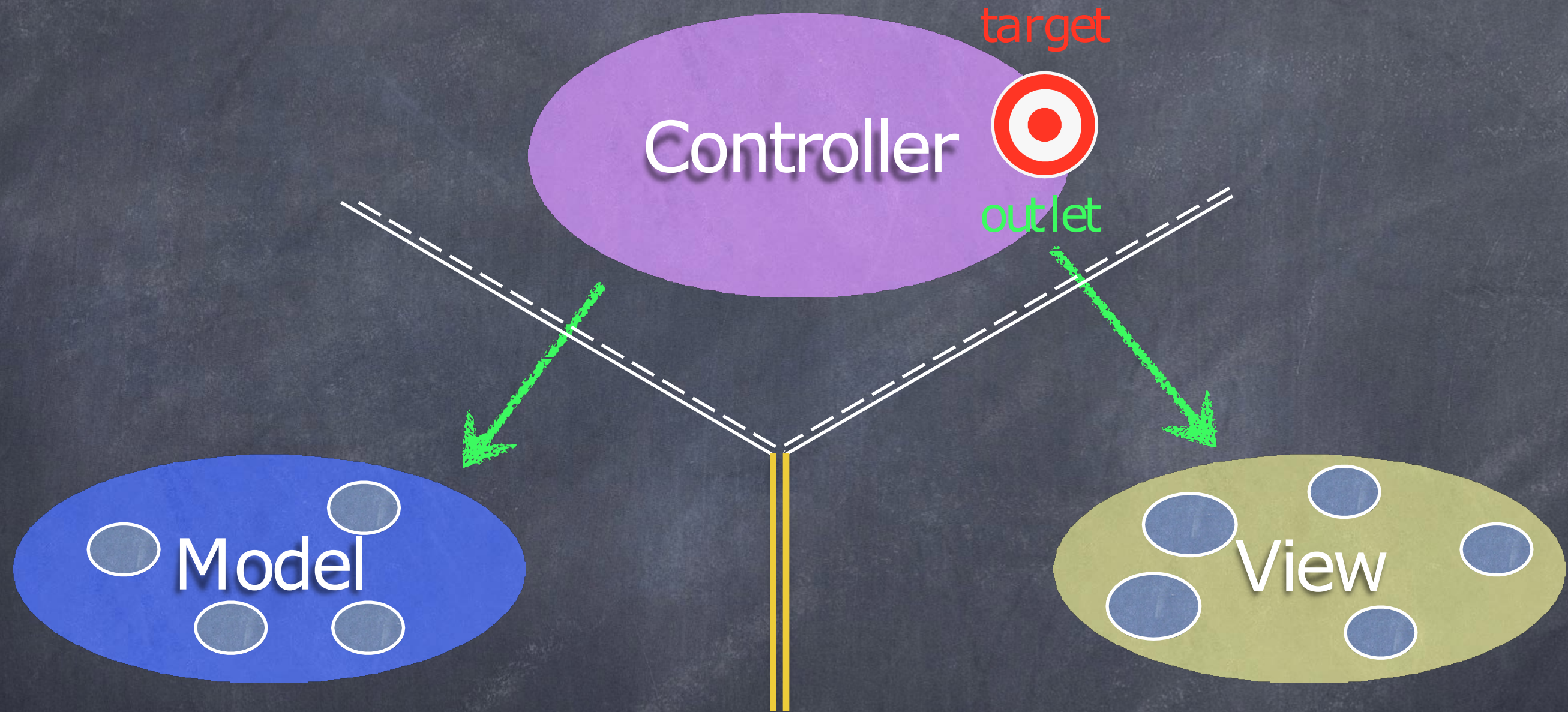
# MVC



Sort of. Communication is "blind" and structured.



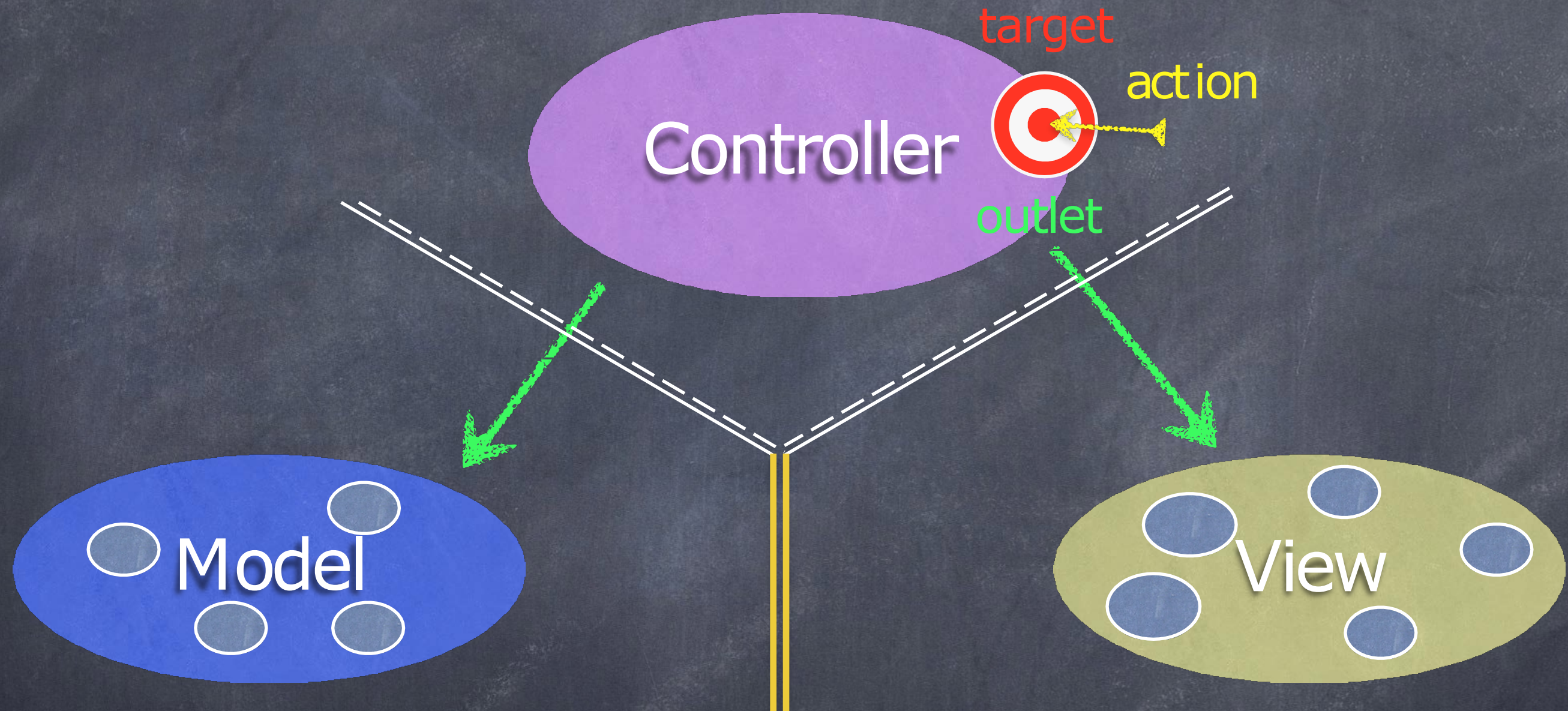
# MVC



The **Controller** can drop a **target** on itself.



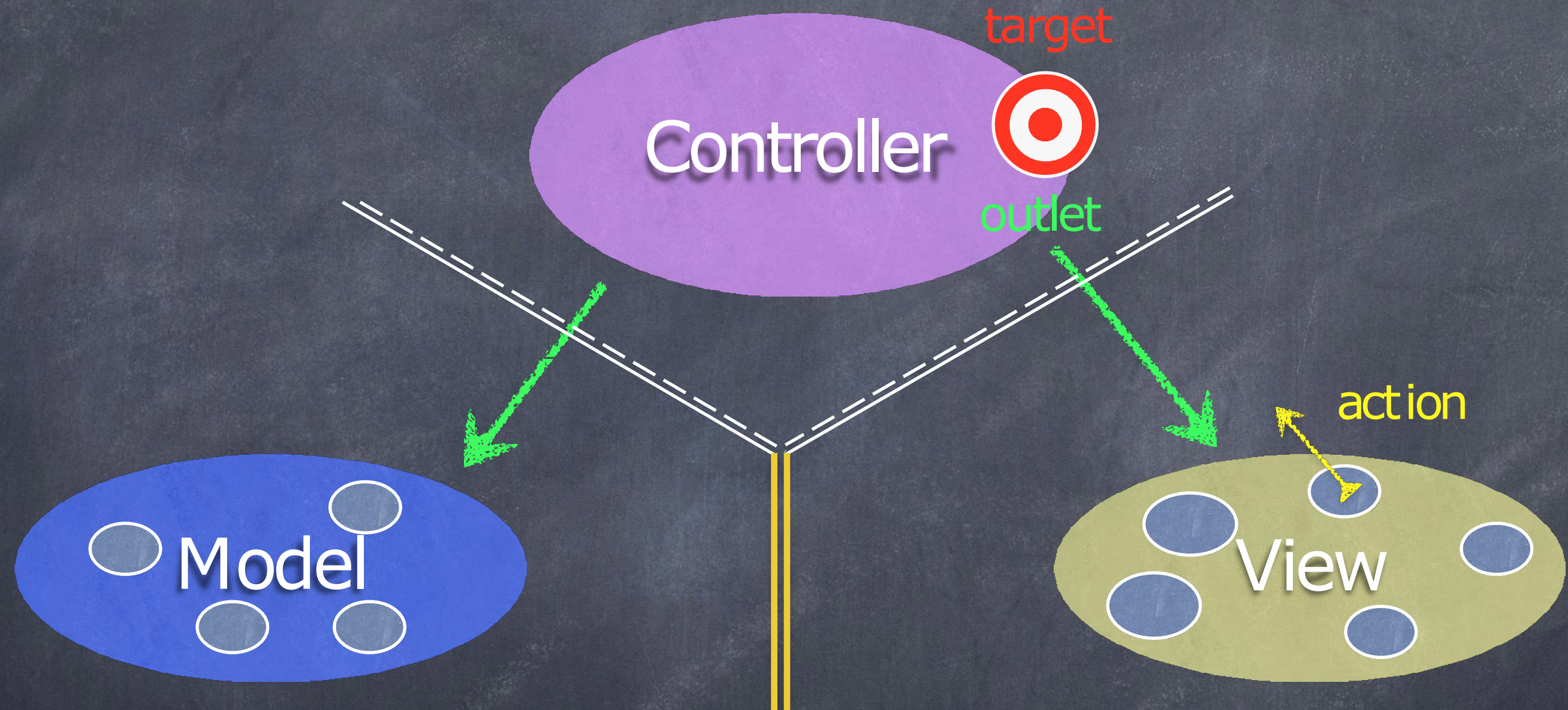
# MVC



Then hand out an **action** to the **View**.



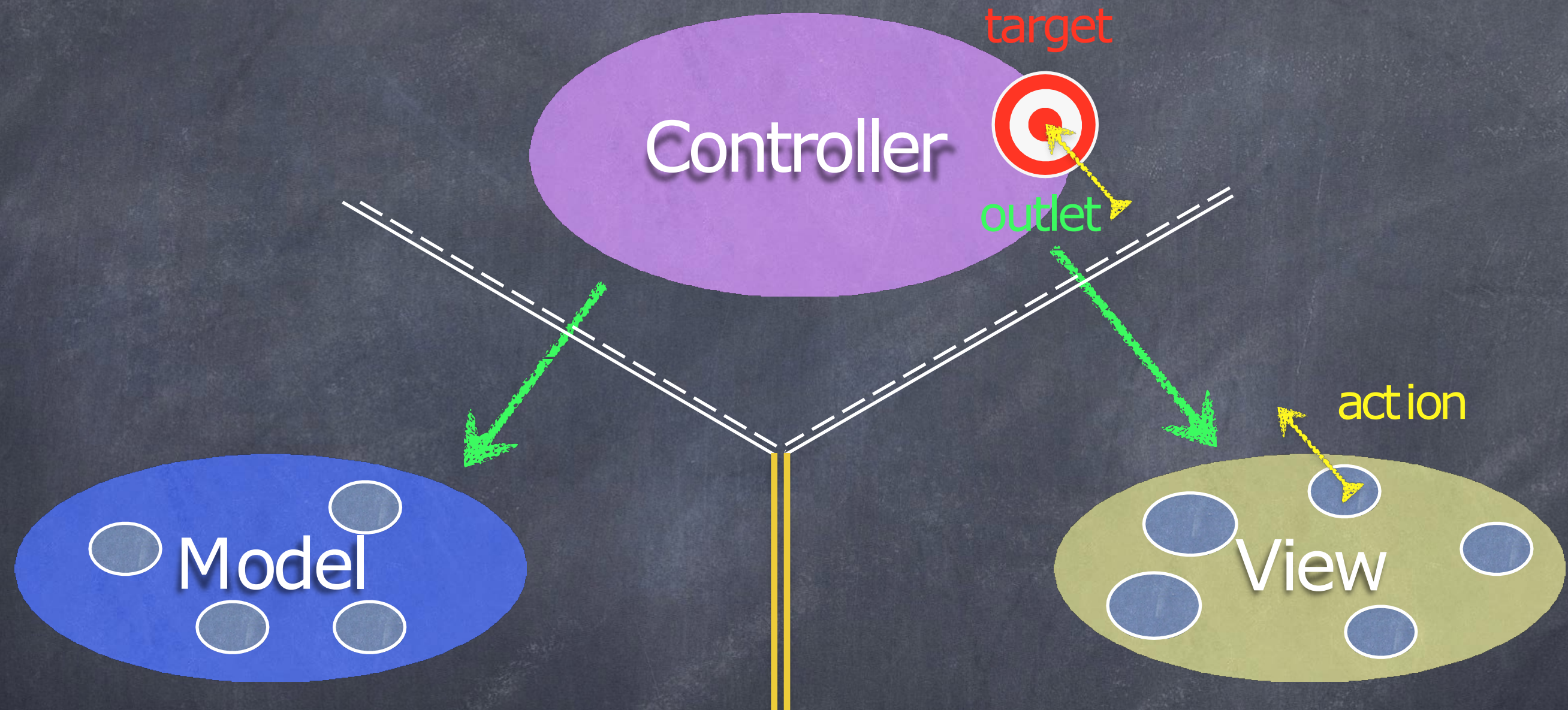
# MVC



Then hand out an **action** to the **View**.



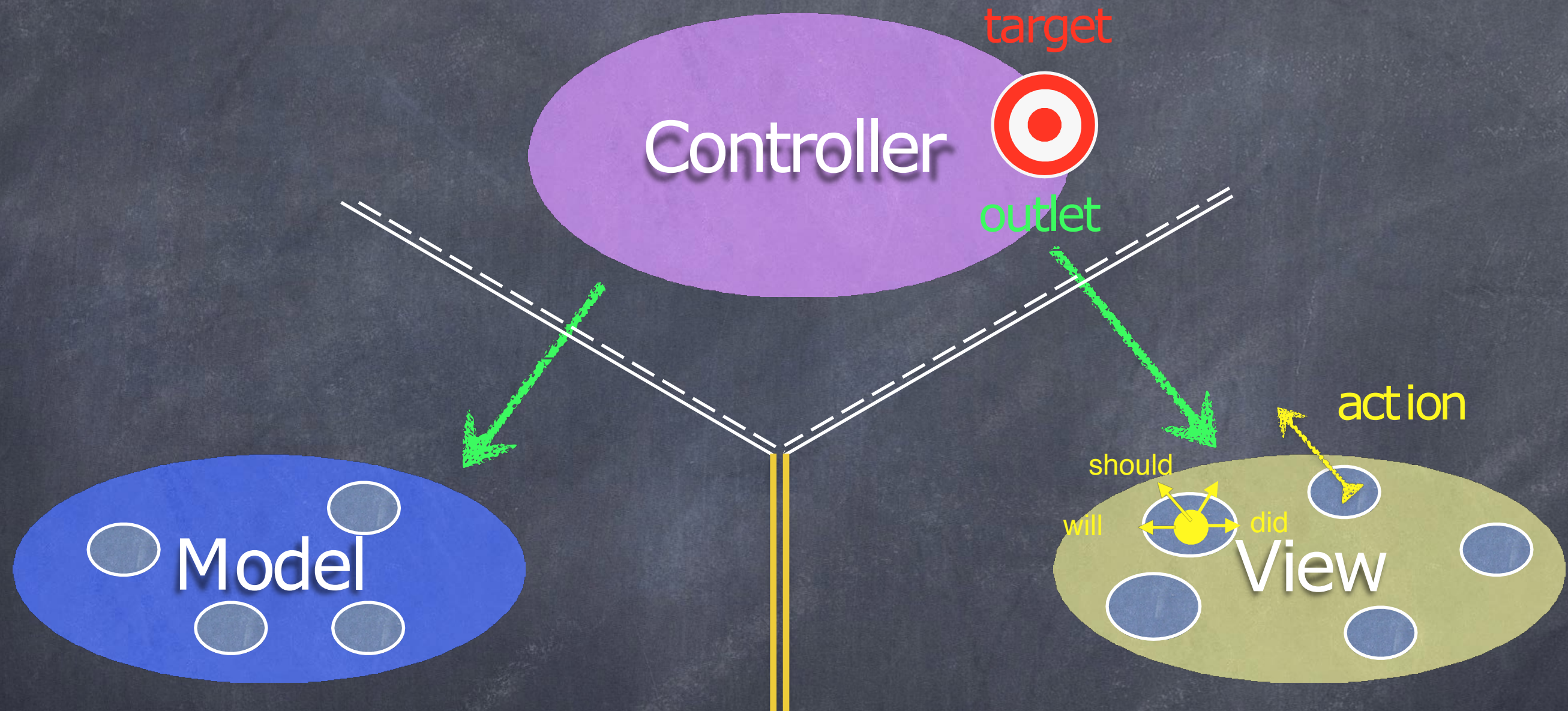
# MVC



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



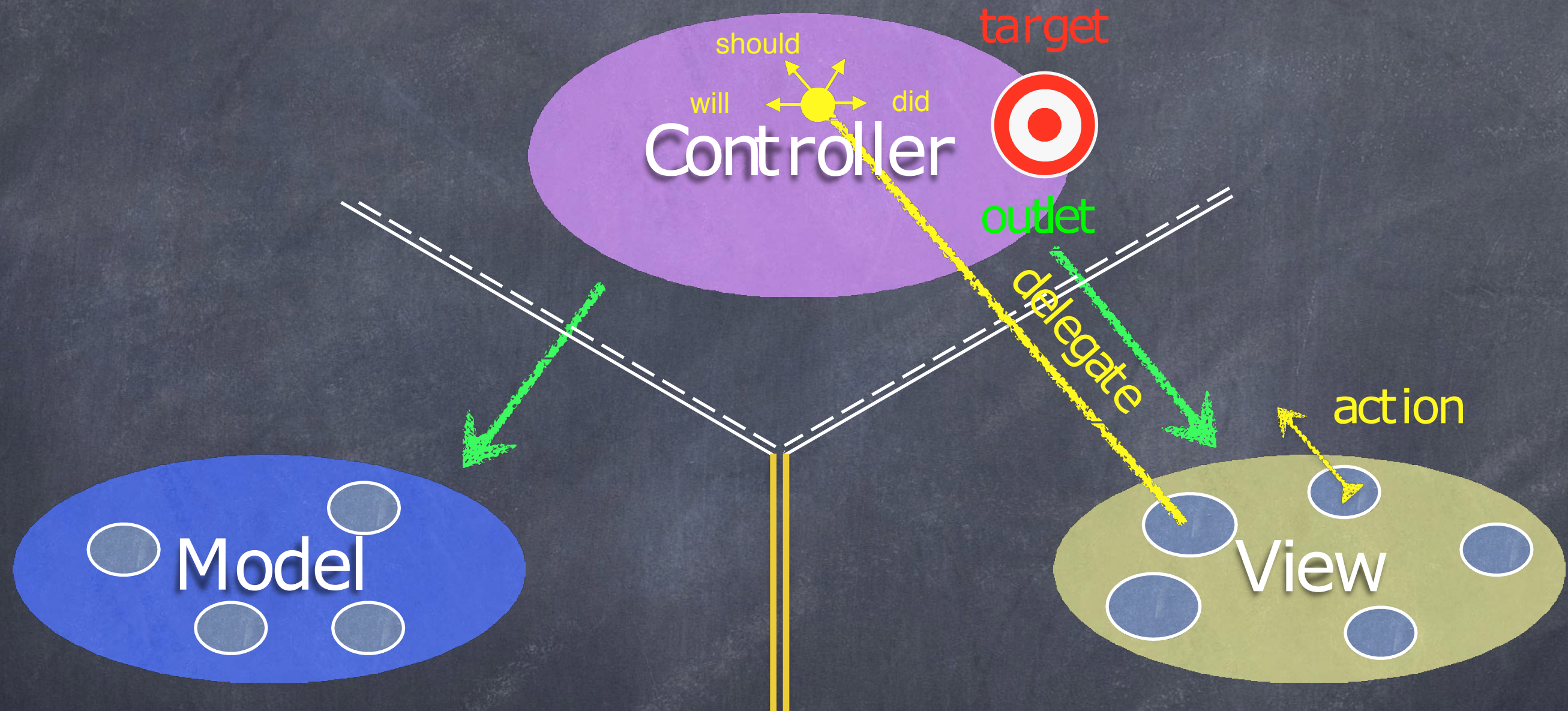
# MVC



Sometimes the **View** needs to synchronize with the **Controller**.



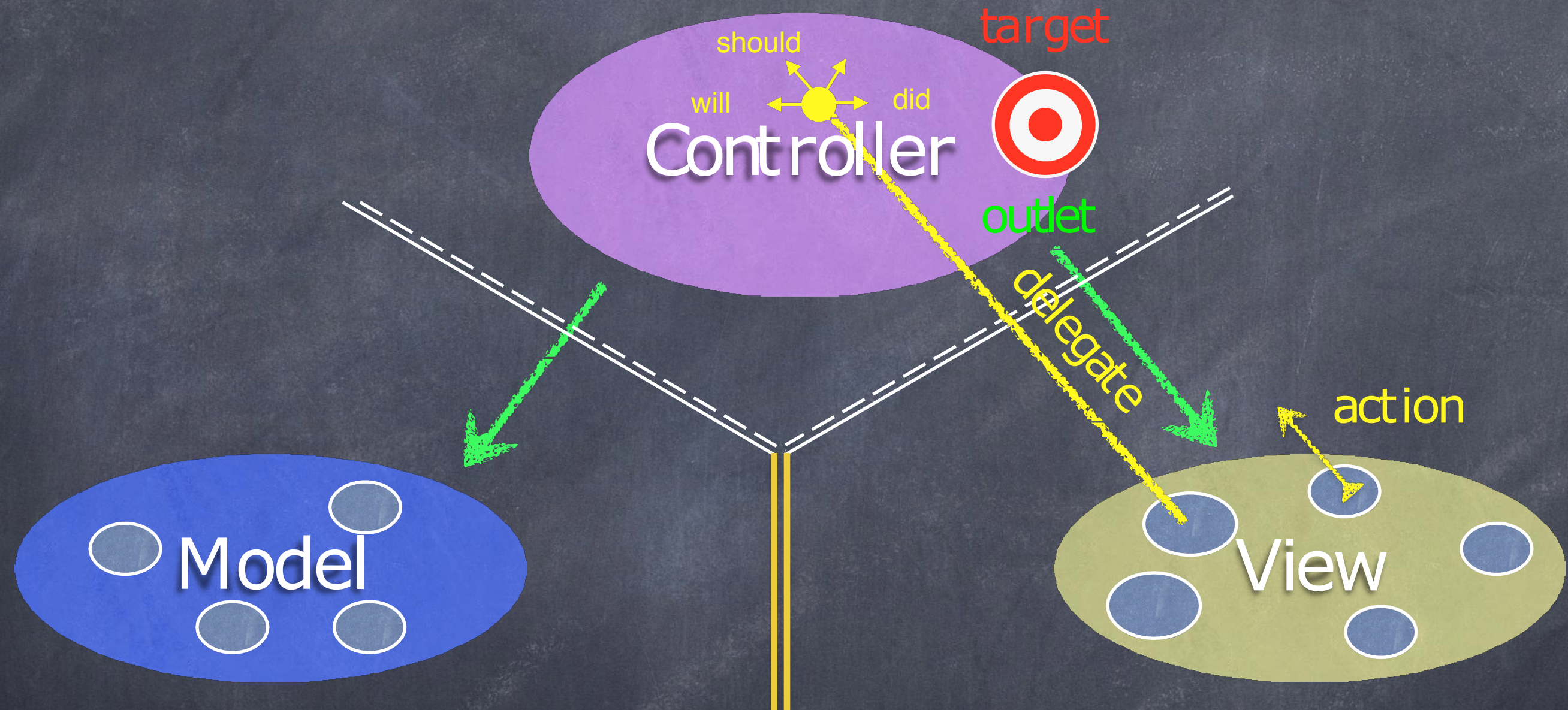
# MVC



The **Controller** sets itself as the **View's** delegate.



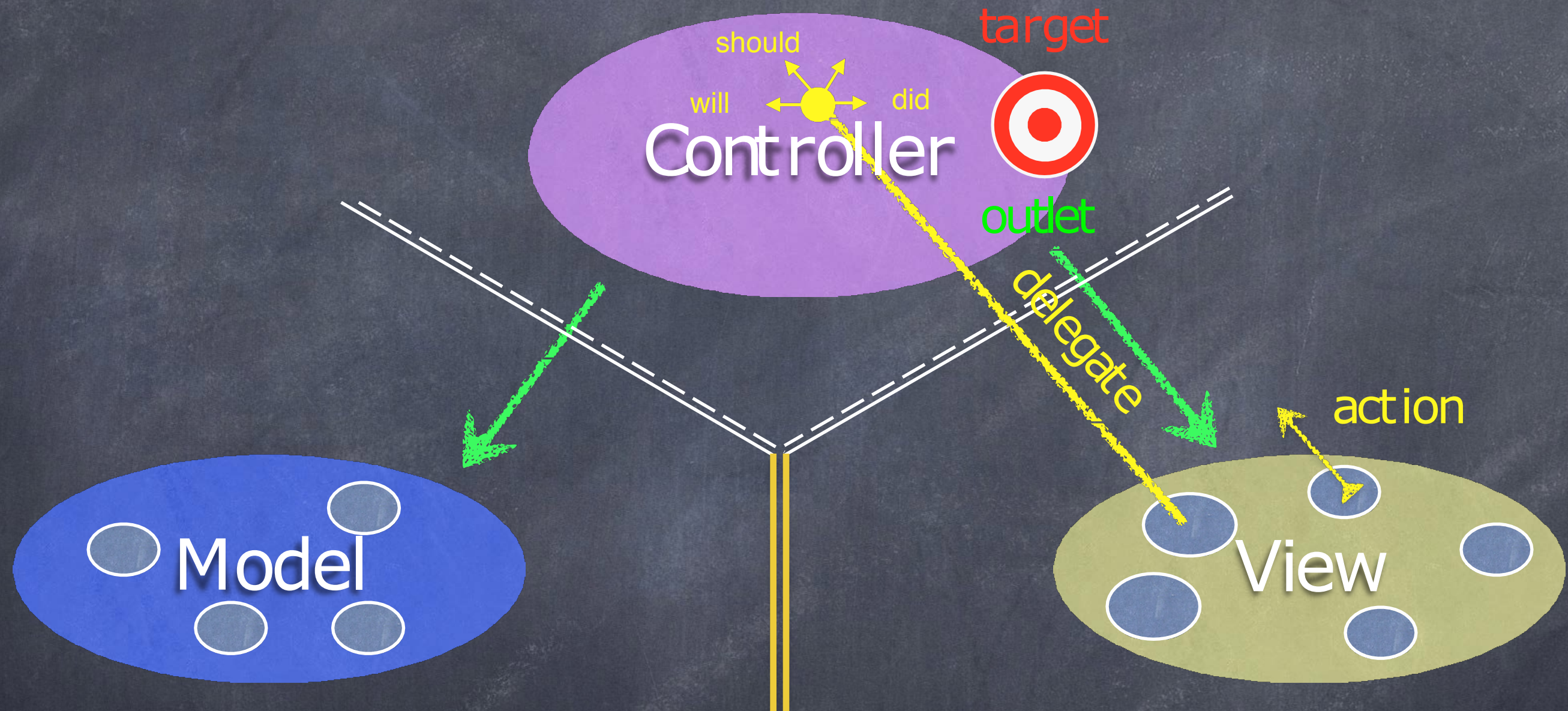
# MVC



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



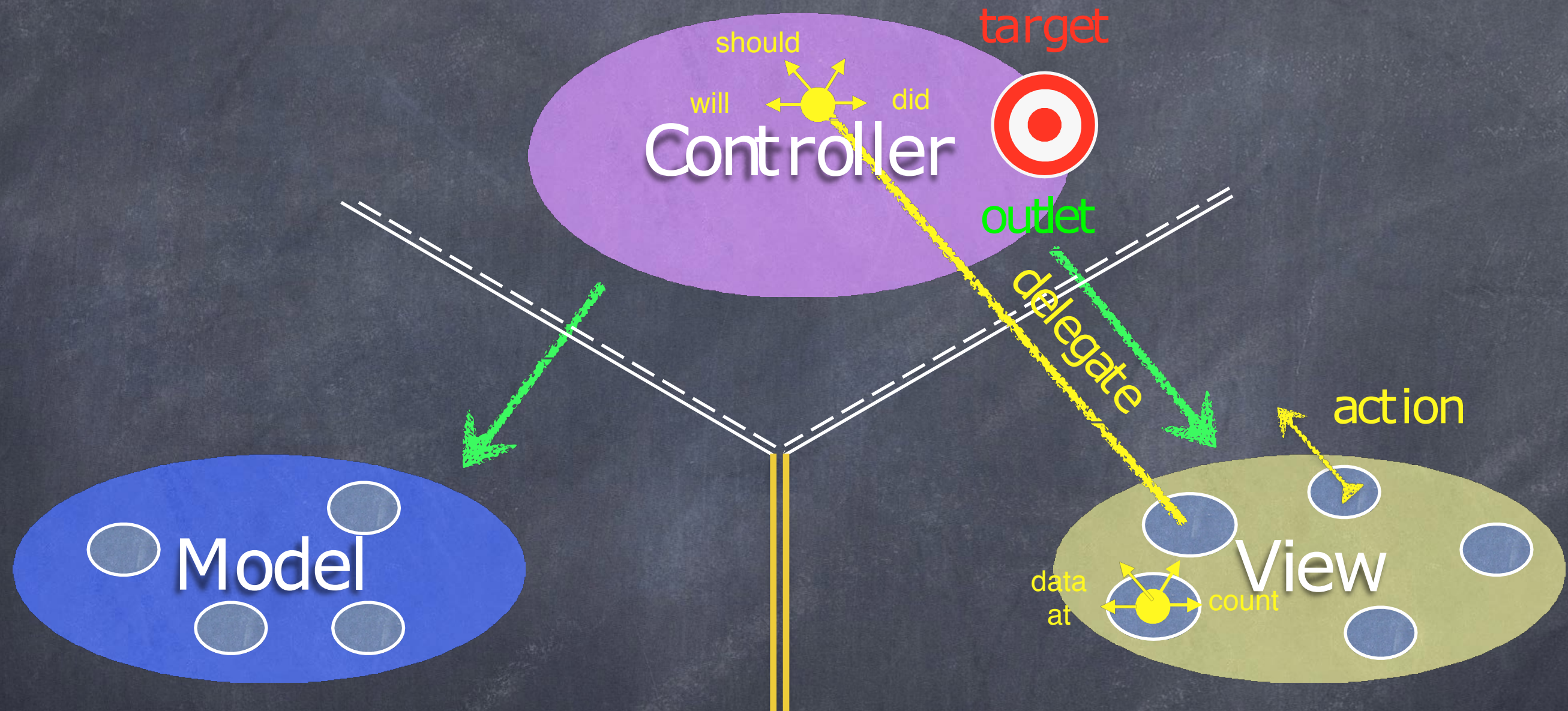
# MVC



Views do not own the data they display.



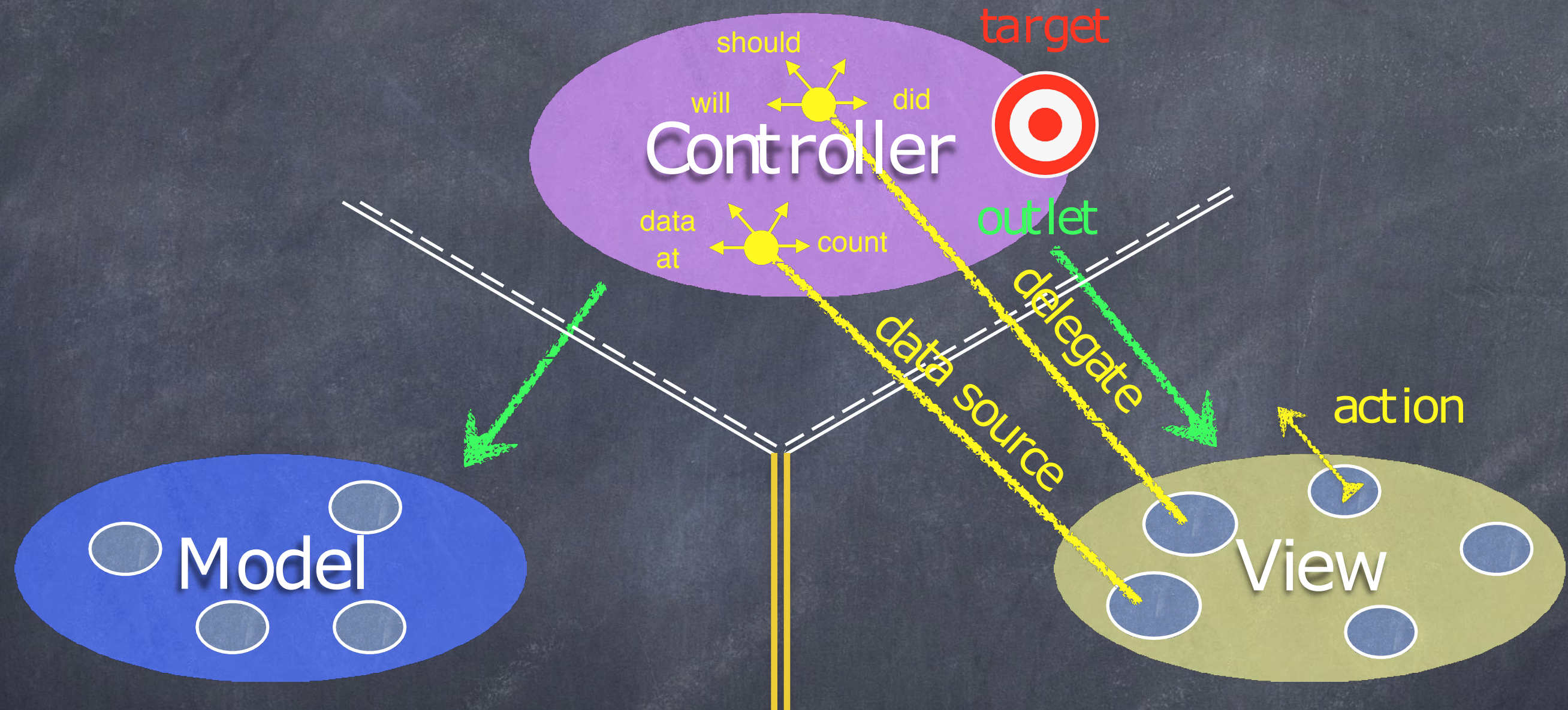
# MVC



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



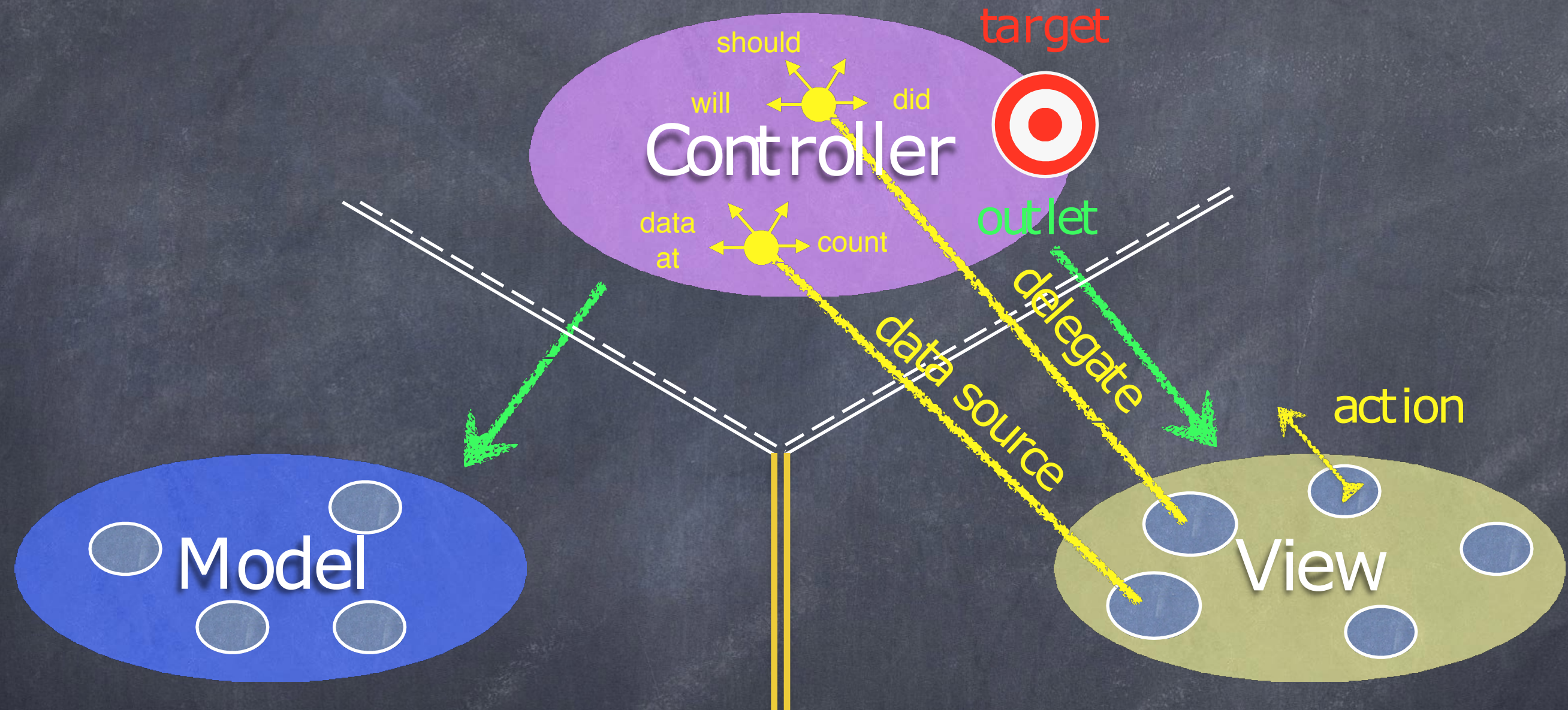
# MVC



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



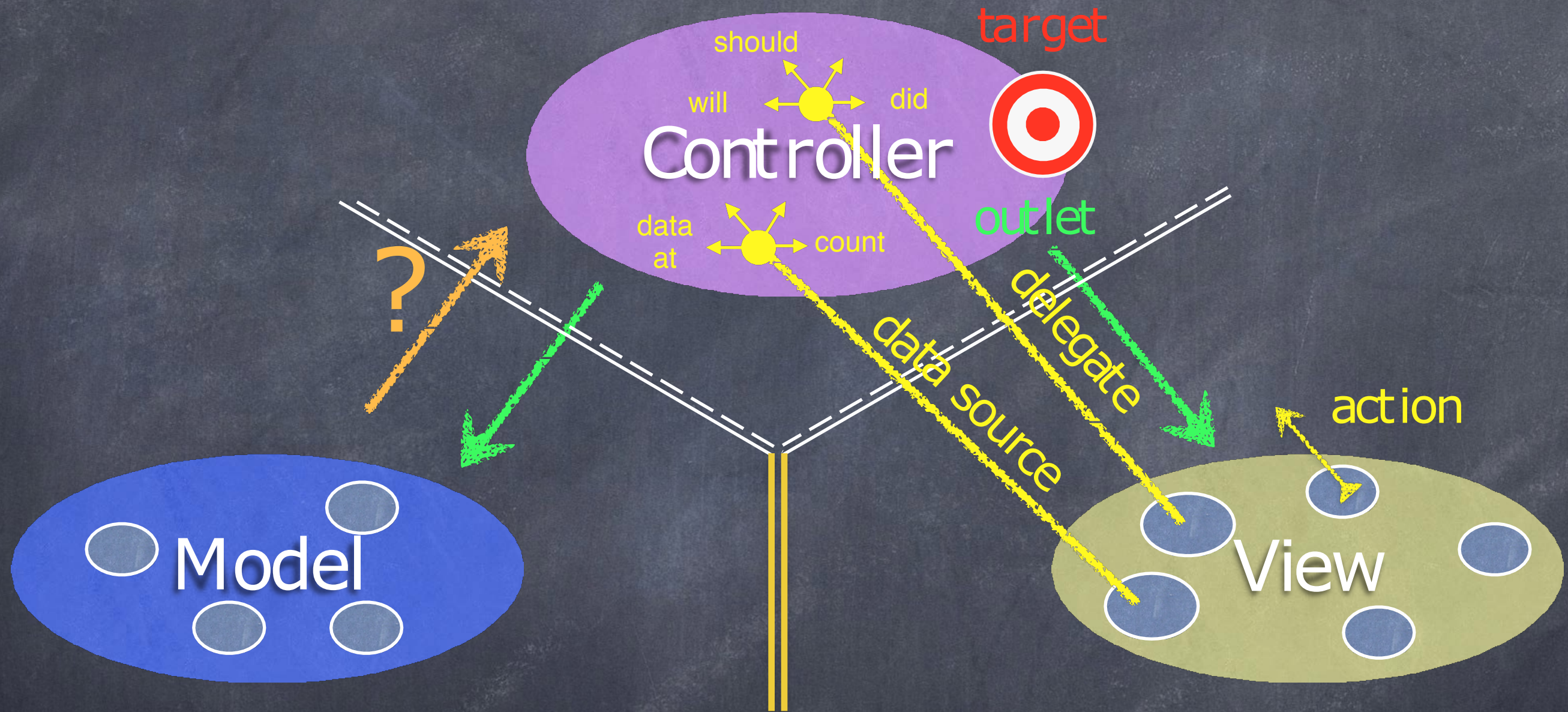
# MVC



Controllers interpret/format Model information for the View.



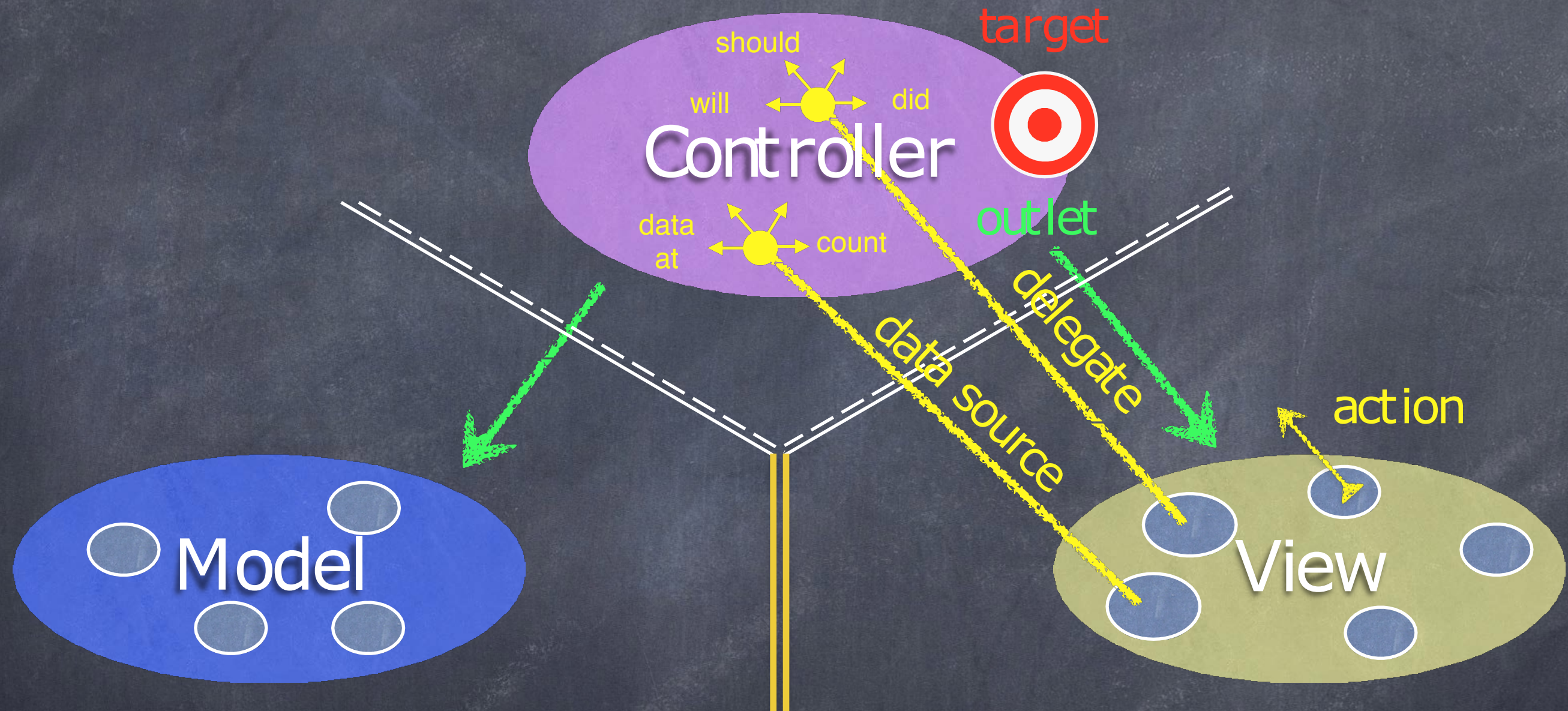
# MVC



# Can the Model talk directly to the Controller?



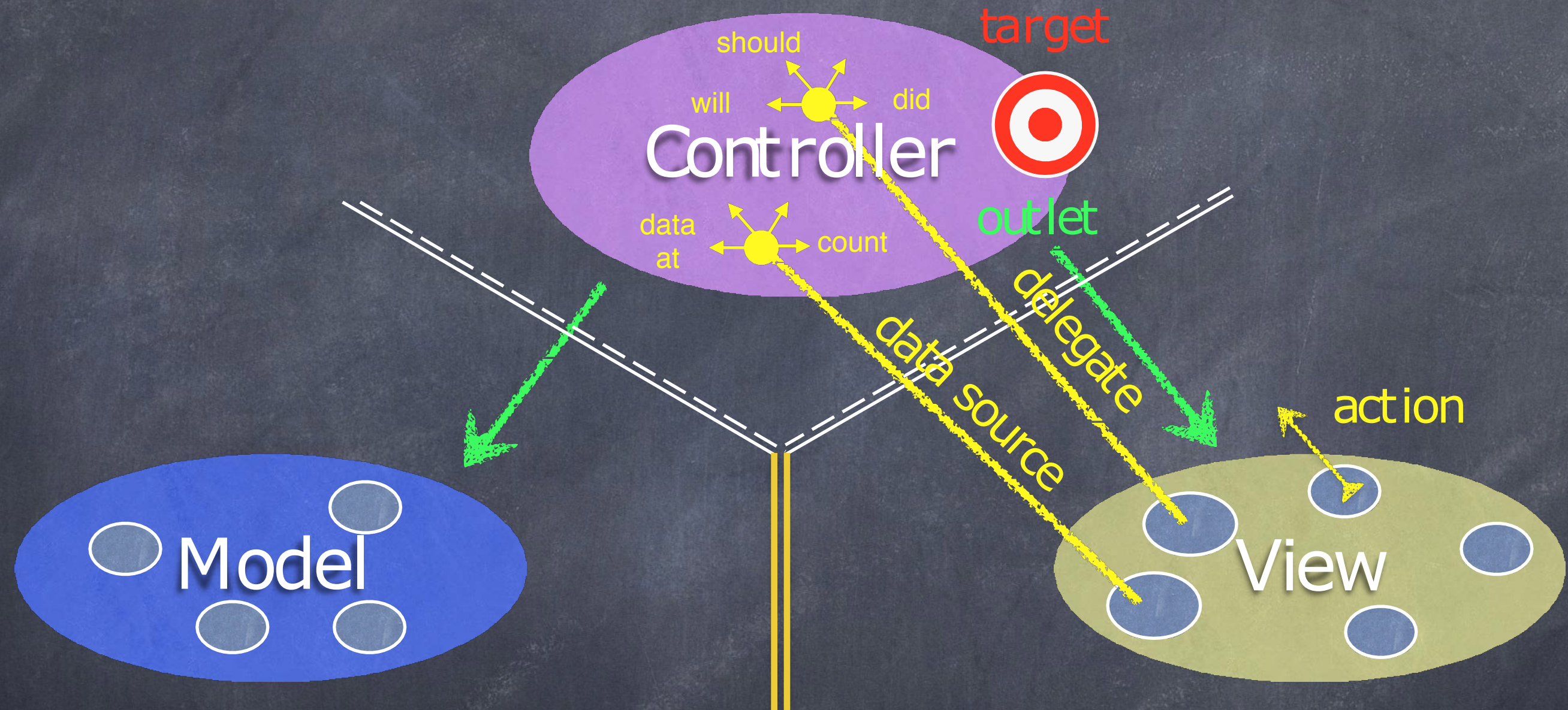
# MVC



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



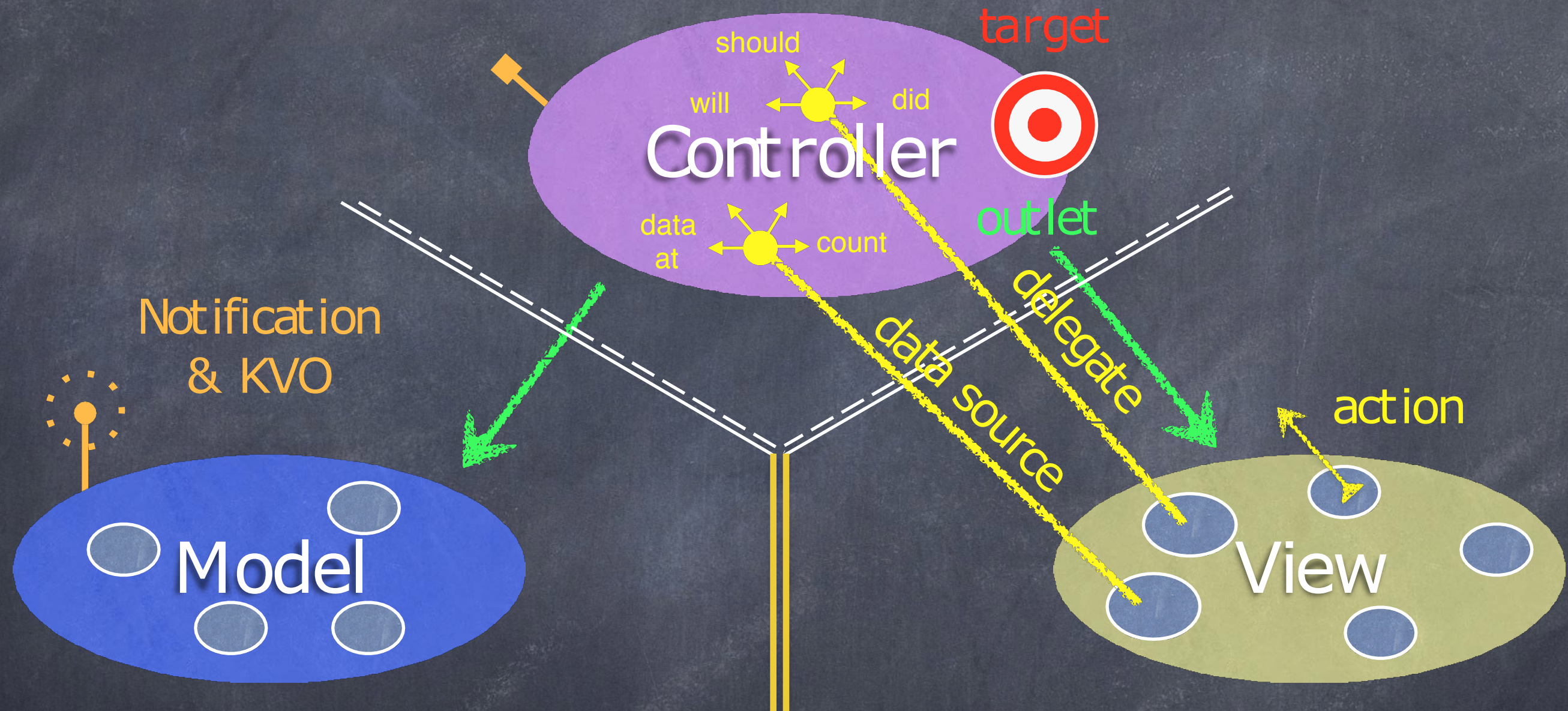
# MVC



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



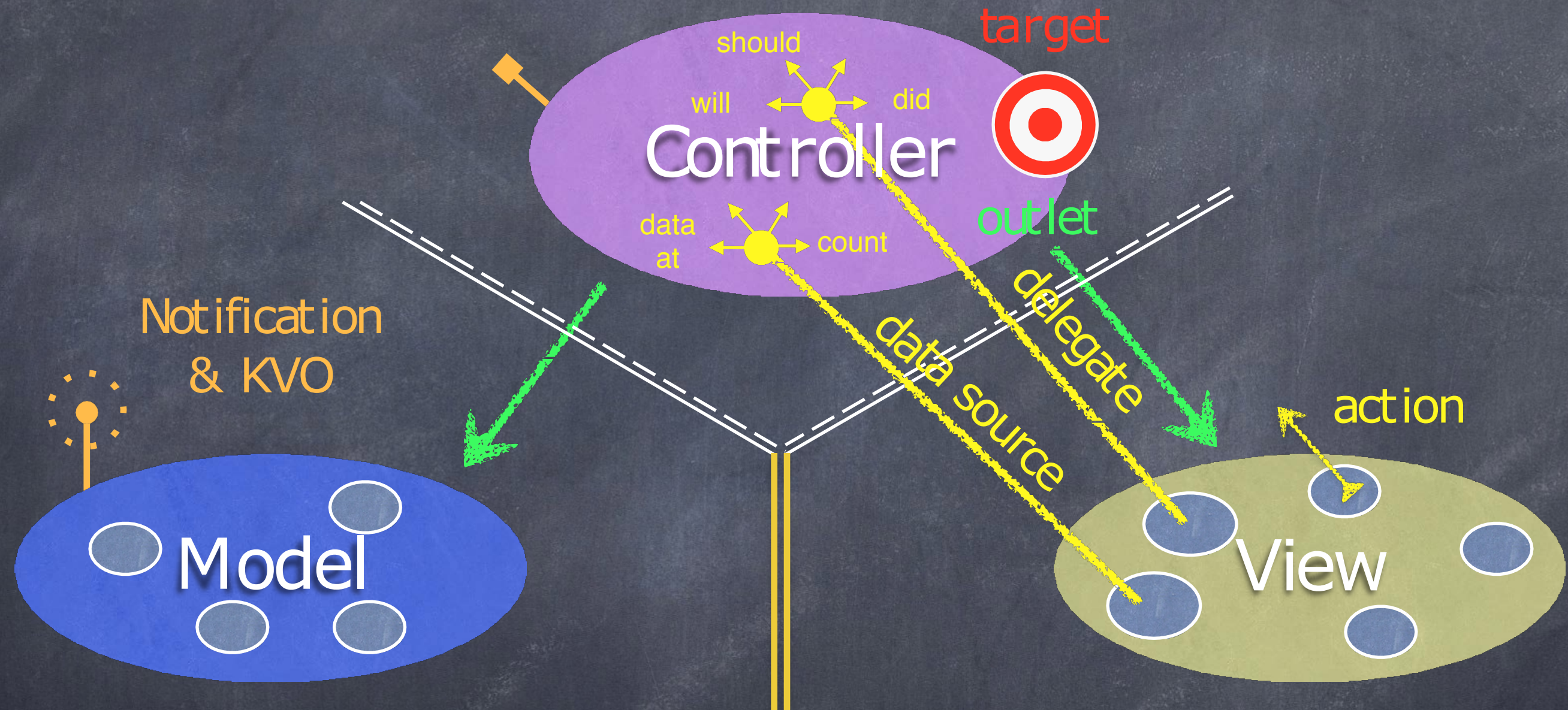
# MVC



It uses a “radio station”-like broadcast mechanism.



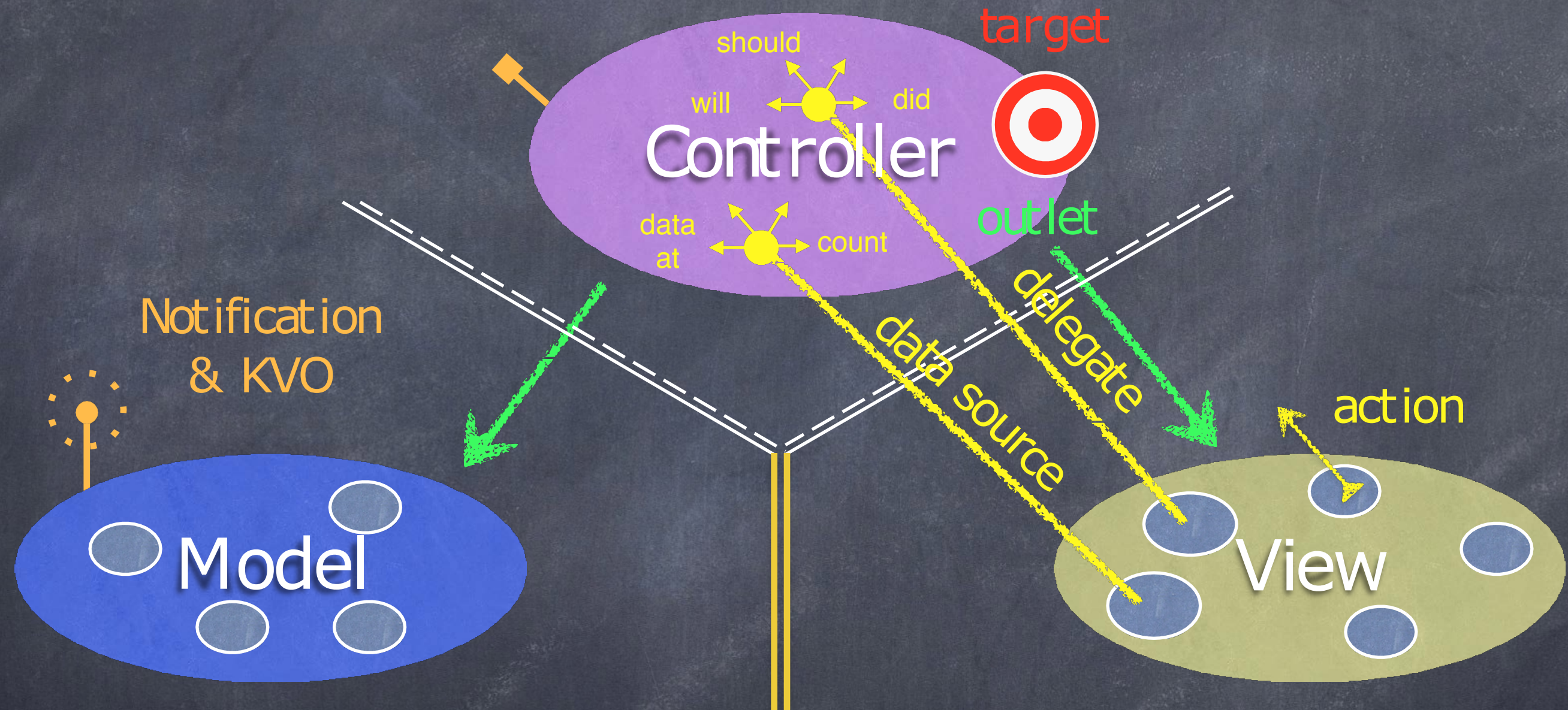
# MVC



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



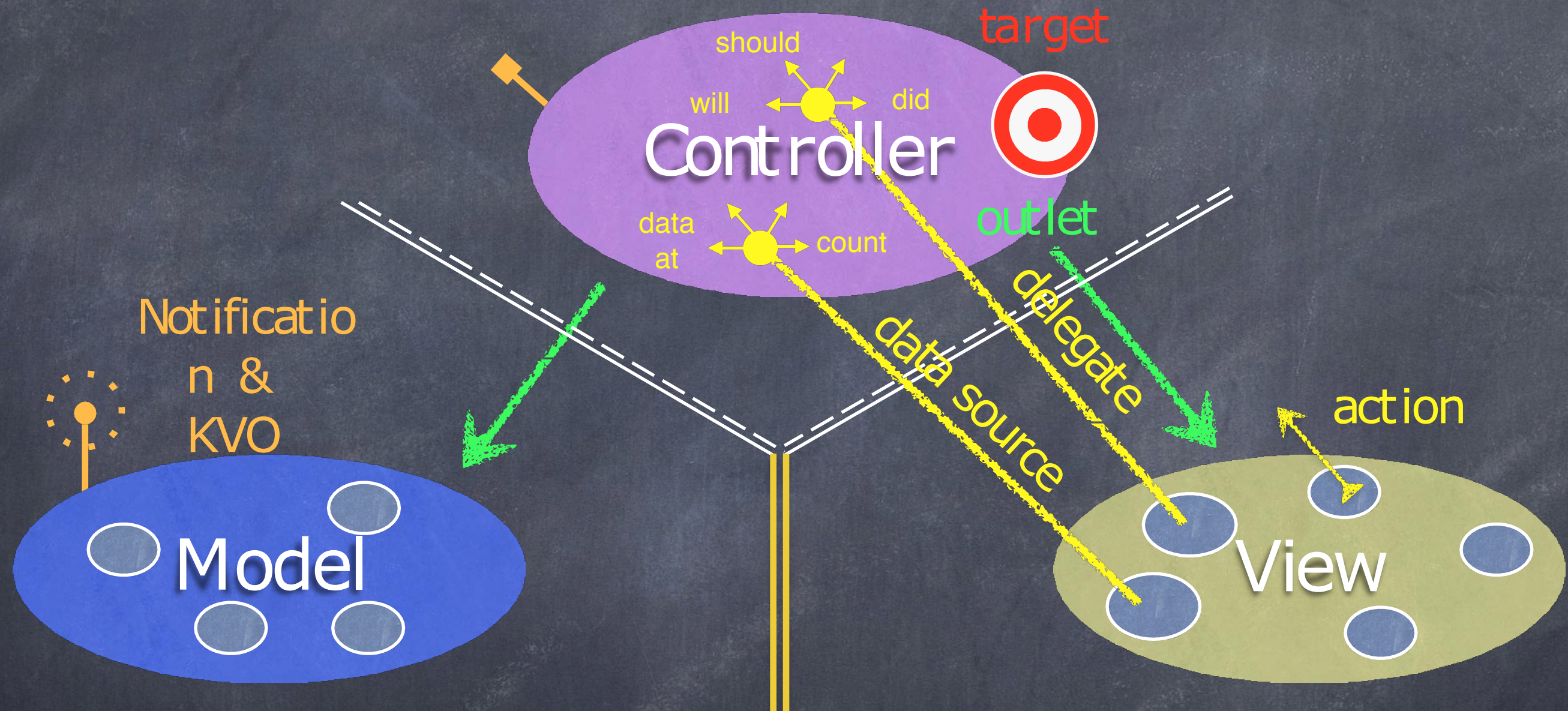
# MVC



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



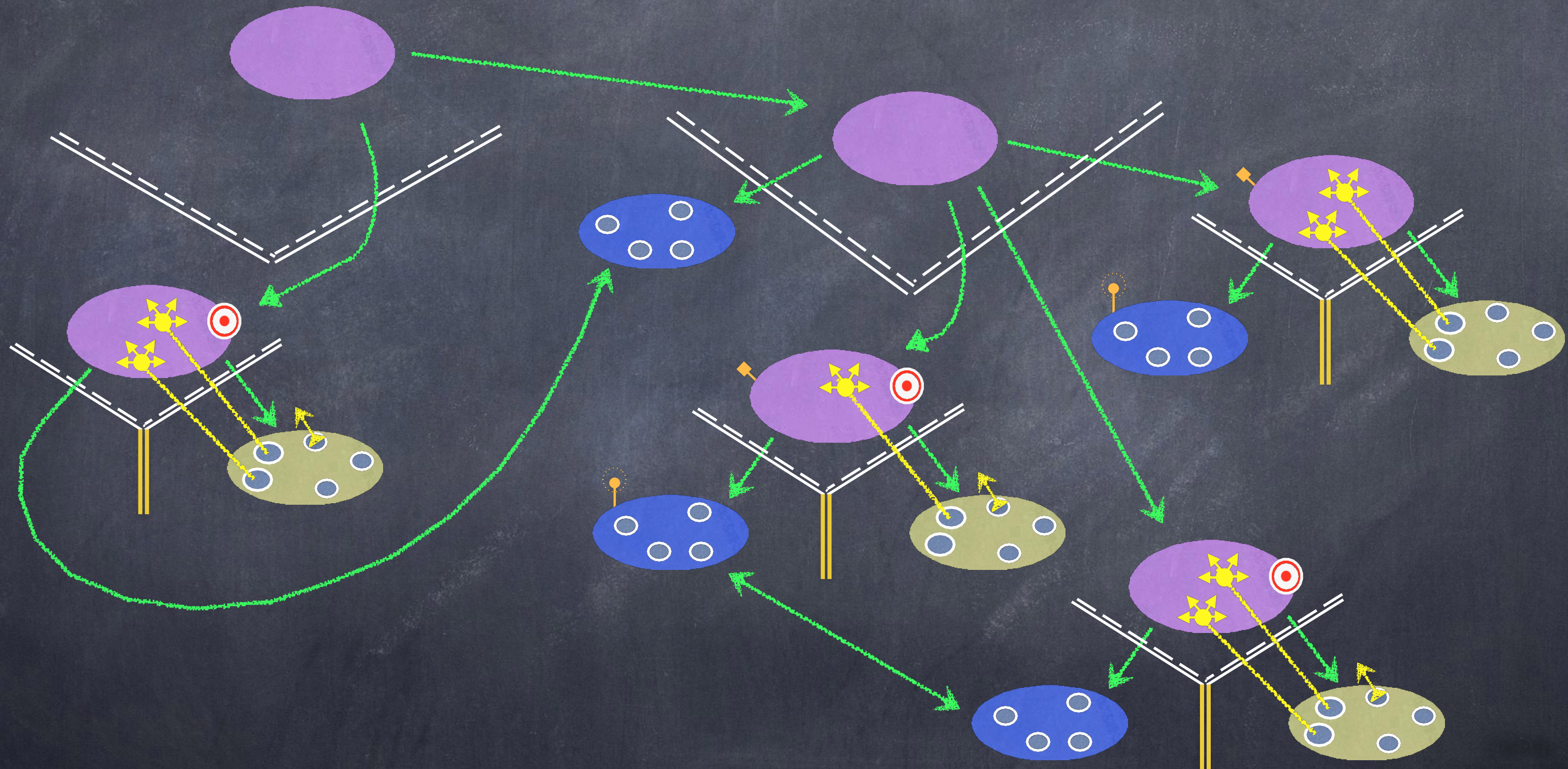
# MVC



Now combine MVC groups to make complicated programs ..

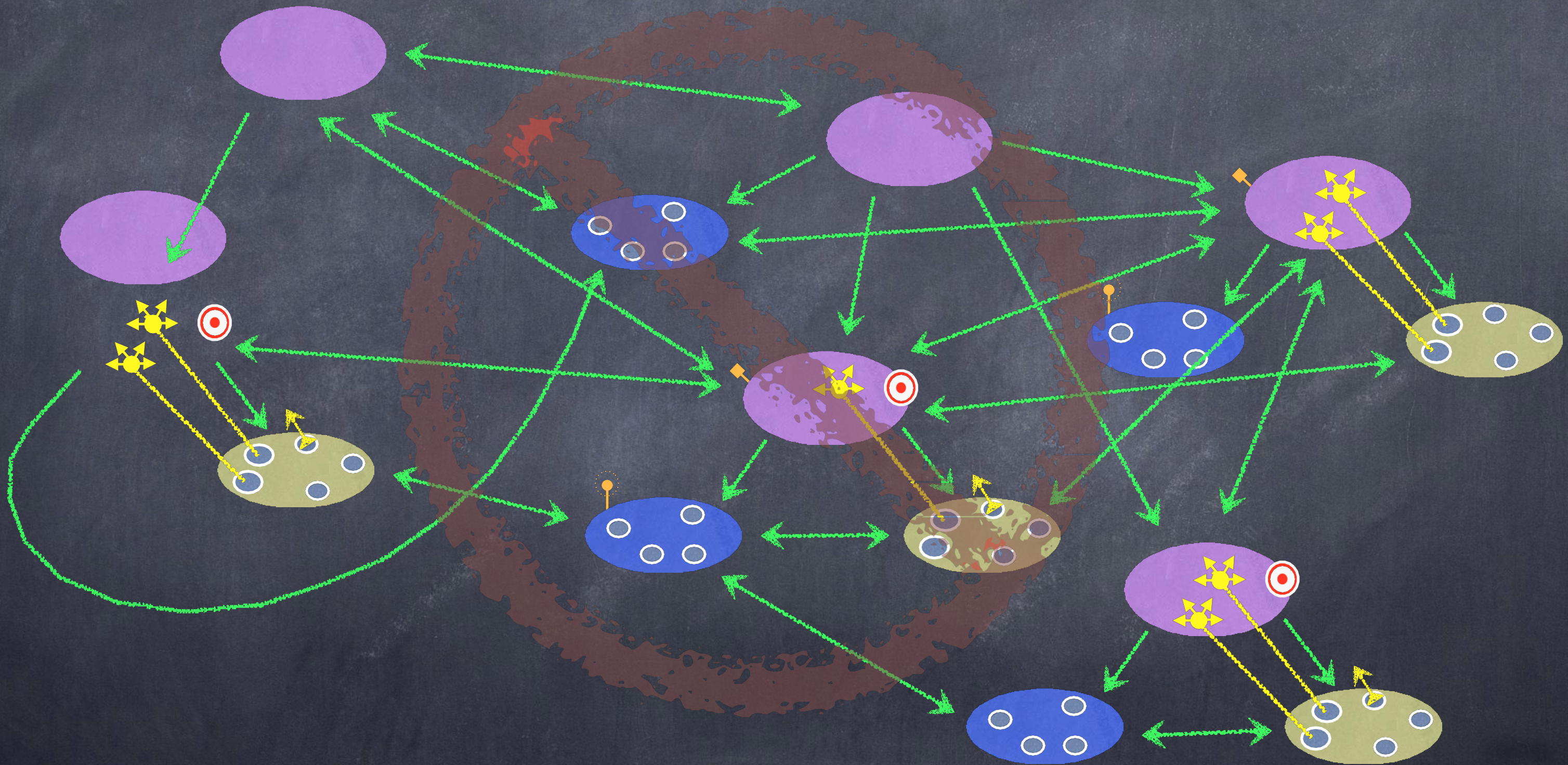


# MVCs working together





# MVCs not working together





# Demo: Calculator

- **MVC**

- struct vs. class (mutating, etc.)
- public versus private API
- more examples of Optional

- **Dictionary<KeyType, ValueType>**

- **enum**

- associated values
- switch
- Functions as types
- Closure syntax for defining functions “on the fly”

- **UIStackView**

- First peek at Autolayout (stick things to the edges)



# 真机测试

- 免开发者账号 (XCode 7以后)
- Apple ID
  - 邮箱
- iOS SDK 支持
  - `/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneOS.platform/DeviceSupport`



# 步骤1 添加Apple ID

1. “Xcode” >> “Preferences” >> “Accounts” >> “+”
2. “Apple IDs” >> “View Details”
3. iOS development 和 Mac development, create
4. Done

Provisioning Profiles 可能为空，因为未接真机



# 步骤2 连接真机

- 用数据线连接手机
- Navigator >> 项目 >> general >> Team
- “Xcode” >> “Product”>>”Destination” >> iPhone
- ToolBar >> Build and Run



# 步骤3 手机权限

- 手机添加信任  
**设置→通用→（访问限制→）设备管理，**  
选择信任之前添加的Apple ID
- 点击桌面，打开APP