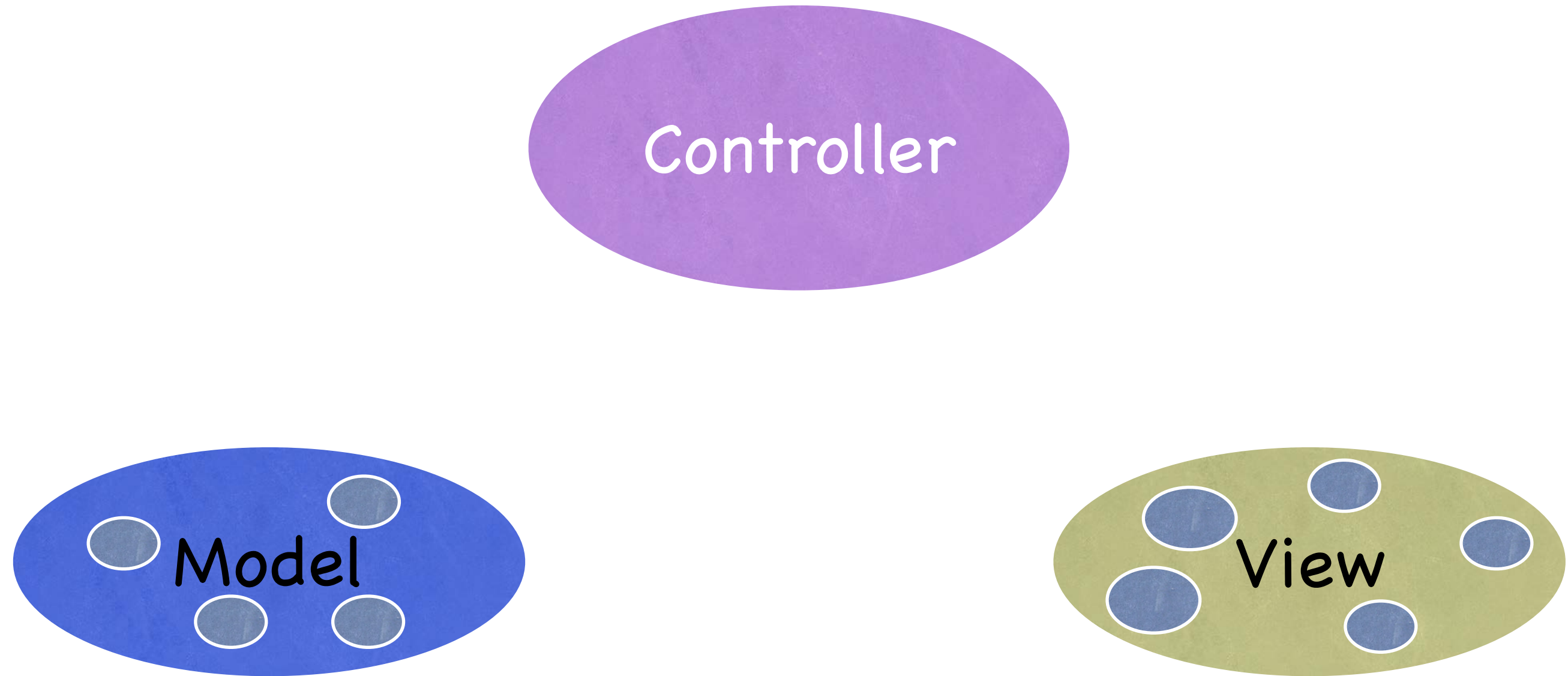


MVC



Divide objects in your program into 3 "camps."

MVC



Controller

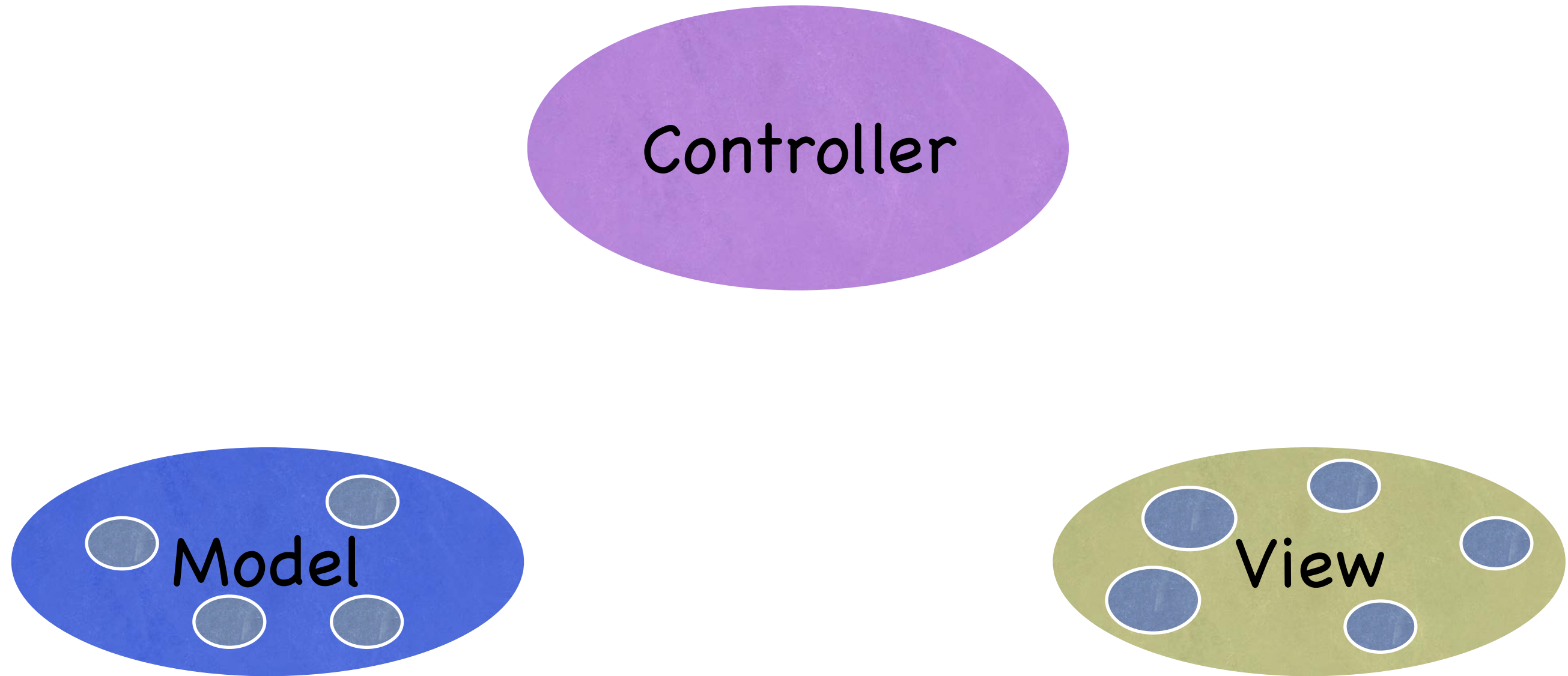
The diagram illustrates the MVC (Model-View-Controller) pattern. At the top center is a large purple oval labeled 'Controller'. Below it, to the left, is a blue oval labeled 'Model' containing four small white circles. To the right of the 'Model' oval is a greenish-yellow oval labeled 'View' containing five small blue circles. The 'Controller' oval is positioned centrally above the 'Model' and 'View' ovals.

Model

View

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

MVC



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

MVC



Controller

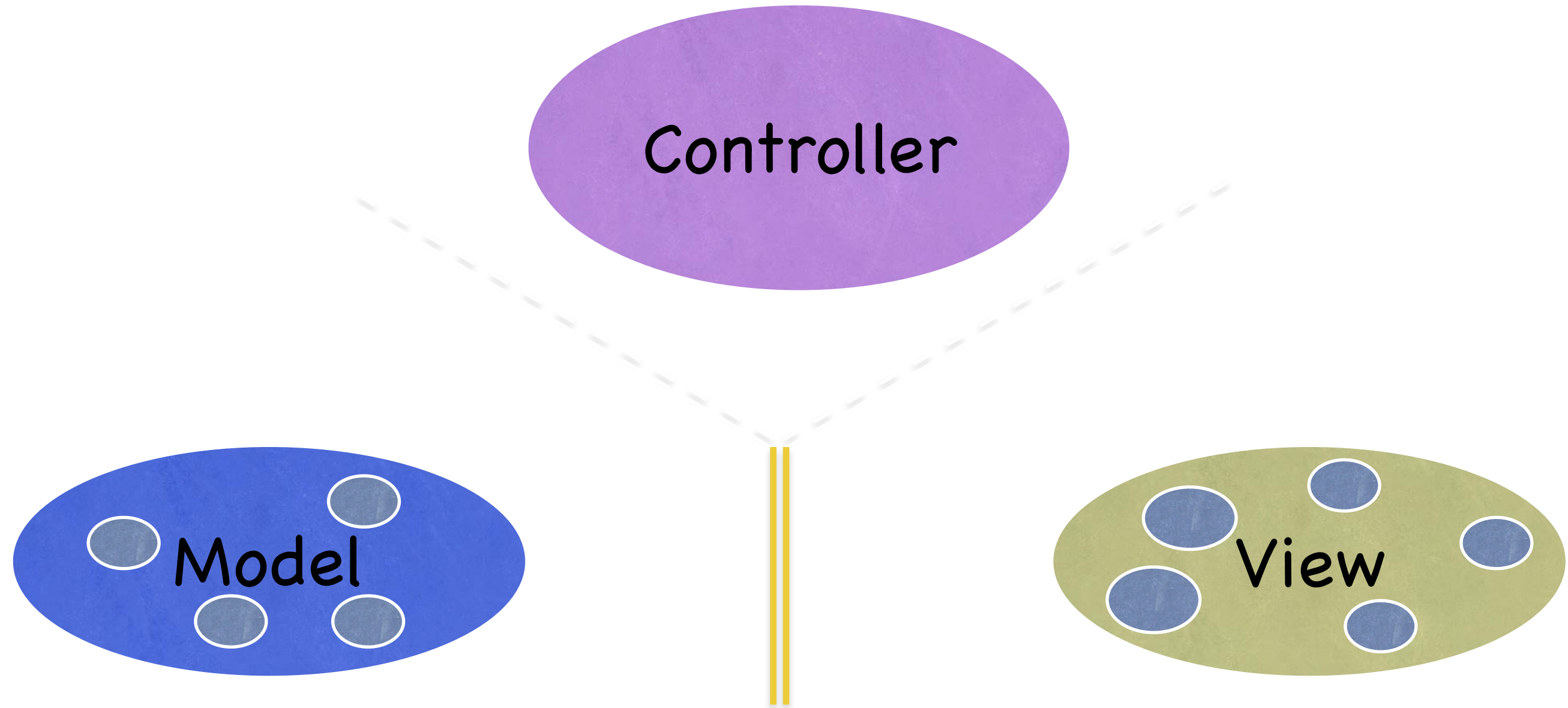
The diagram illustrates the MVC (Model-View-Controller) pattern. At the top is a large purple oval labeled 'Controller'. Below it are two smaller ovals: a blue one on the left labeled 'Model' and a green one on the right labeled 'View'. The 'Model' oval contains four small white circles, and the 'View' oval contains five small blue circles. The 'Controller' oval is positioned centrally above the other two, suggesting its role in managing both the data (Model) and the presentation (View).

Model

View

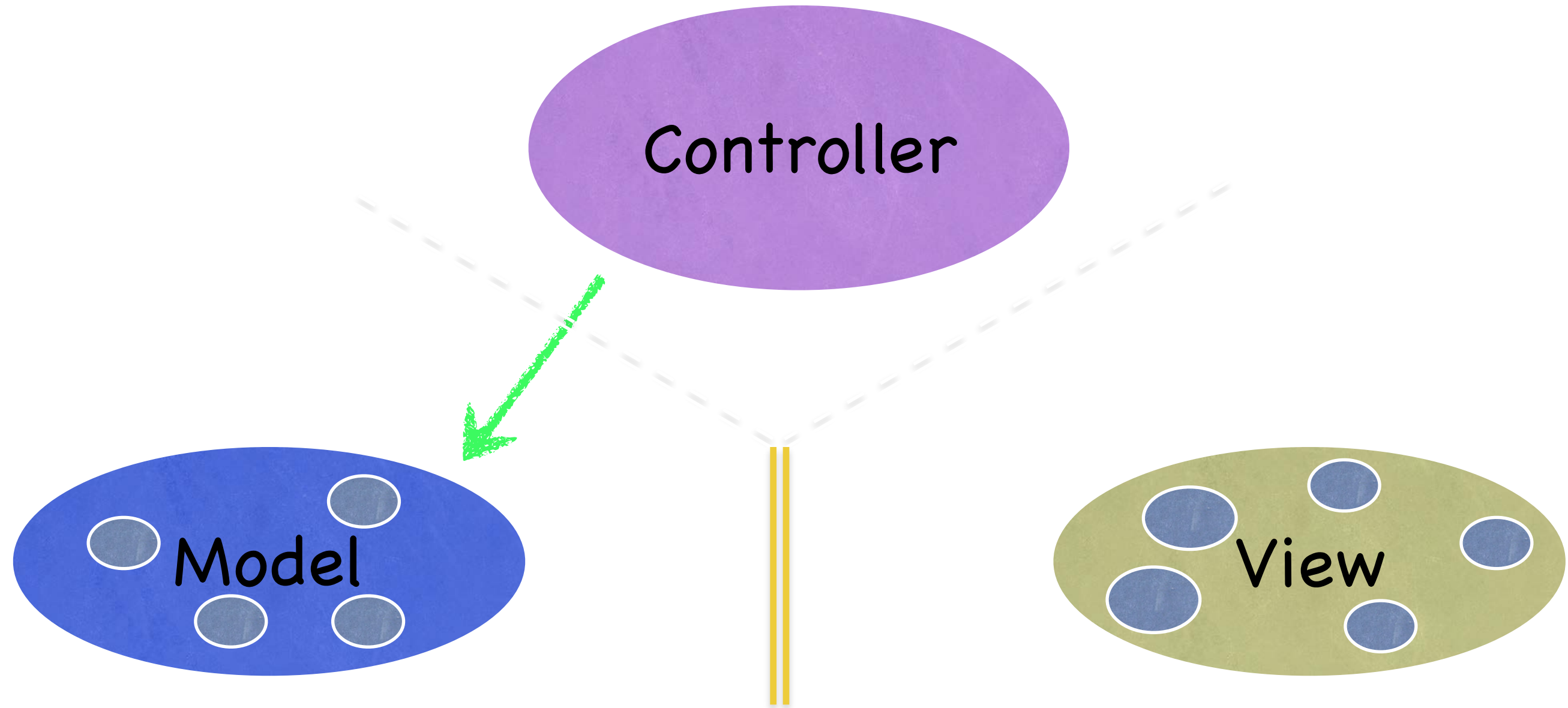
View = Your Controller's minions

MVC



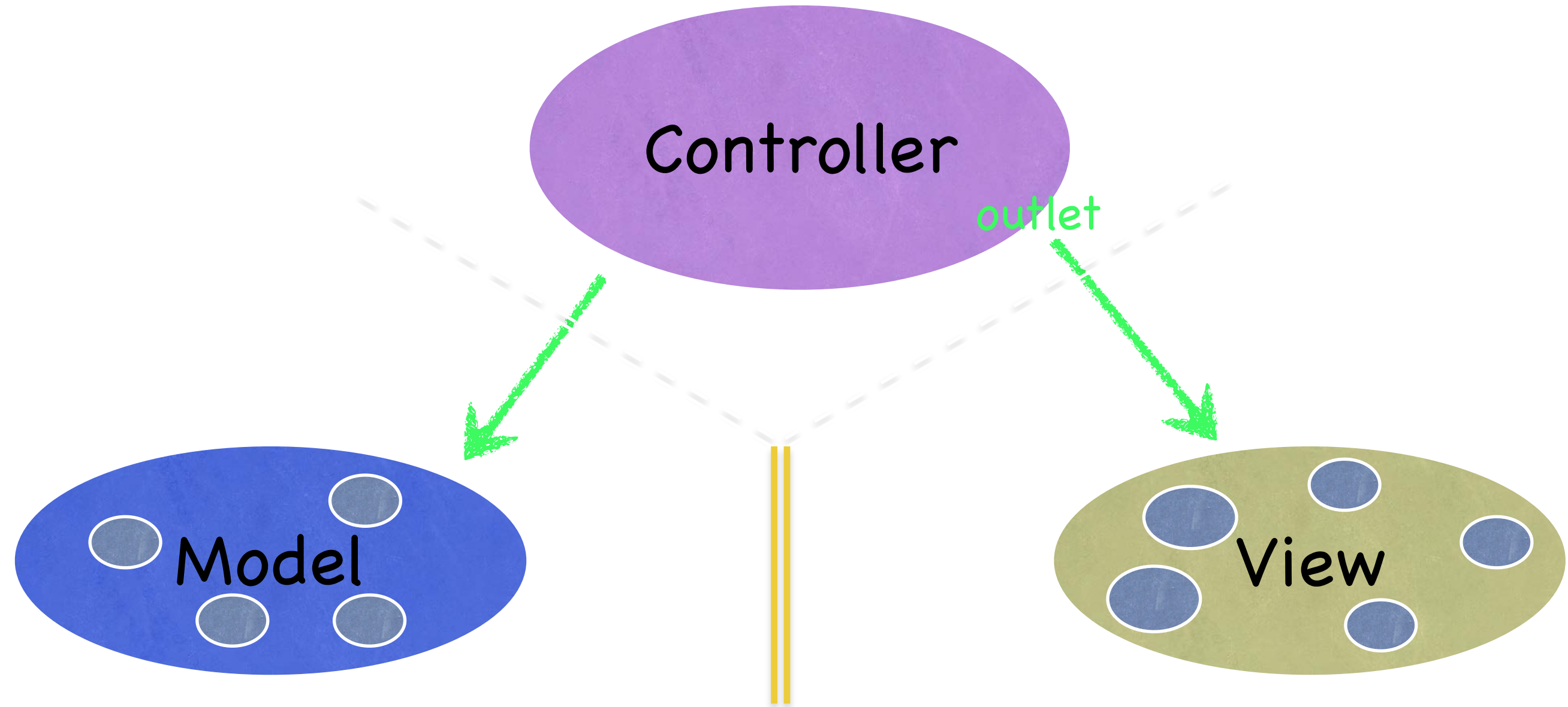
It's all about managing communication between camps

MVC



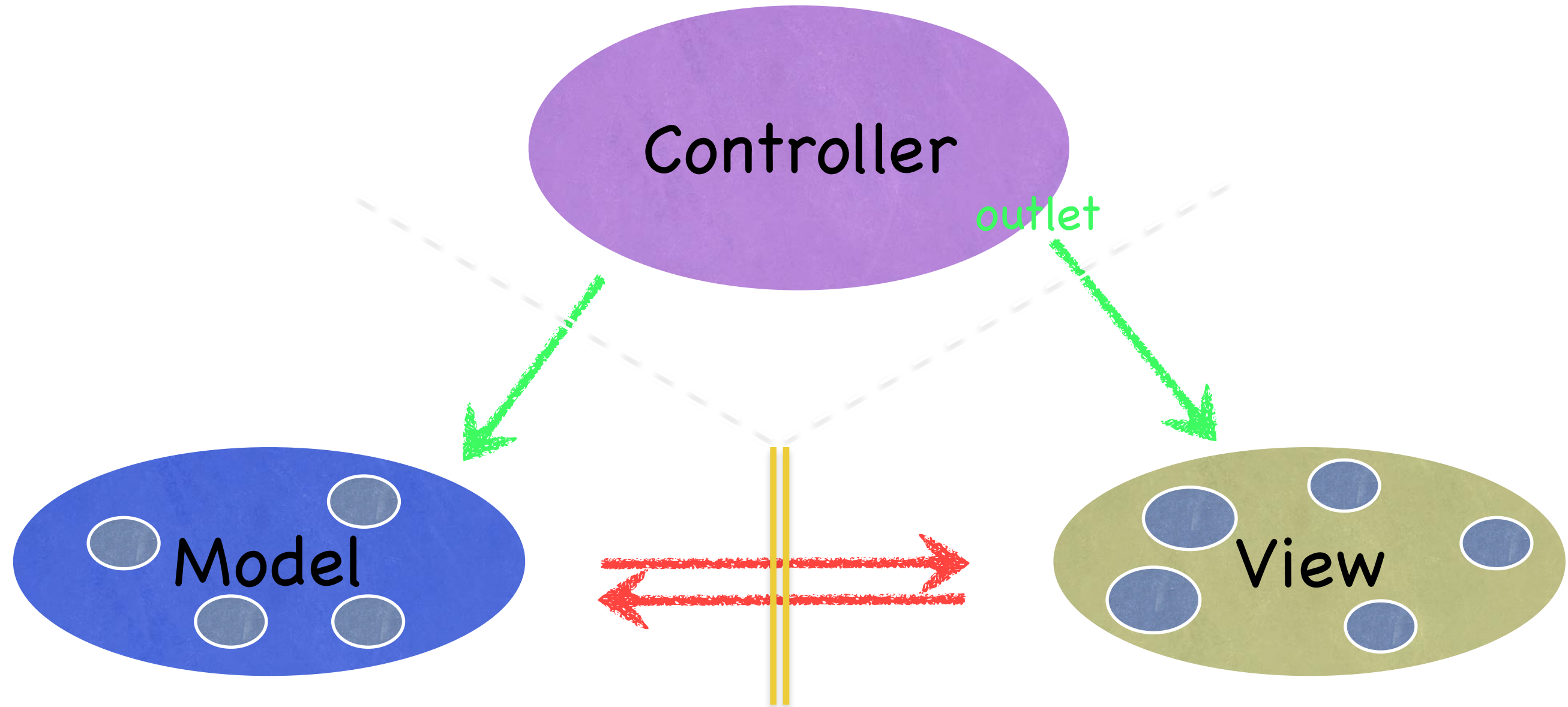
Controllers can always talk directly to their Model.

MVC



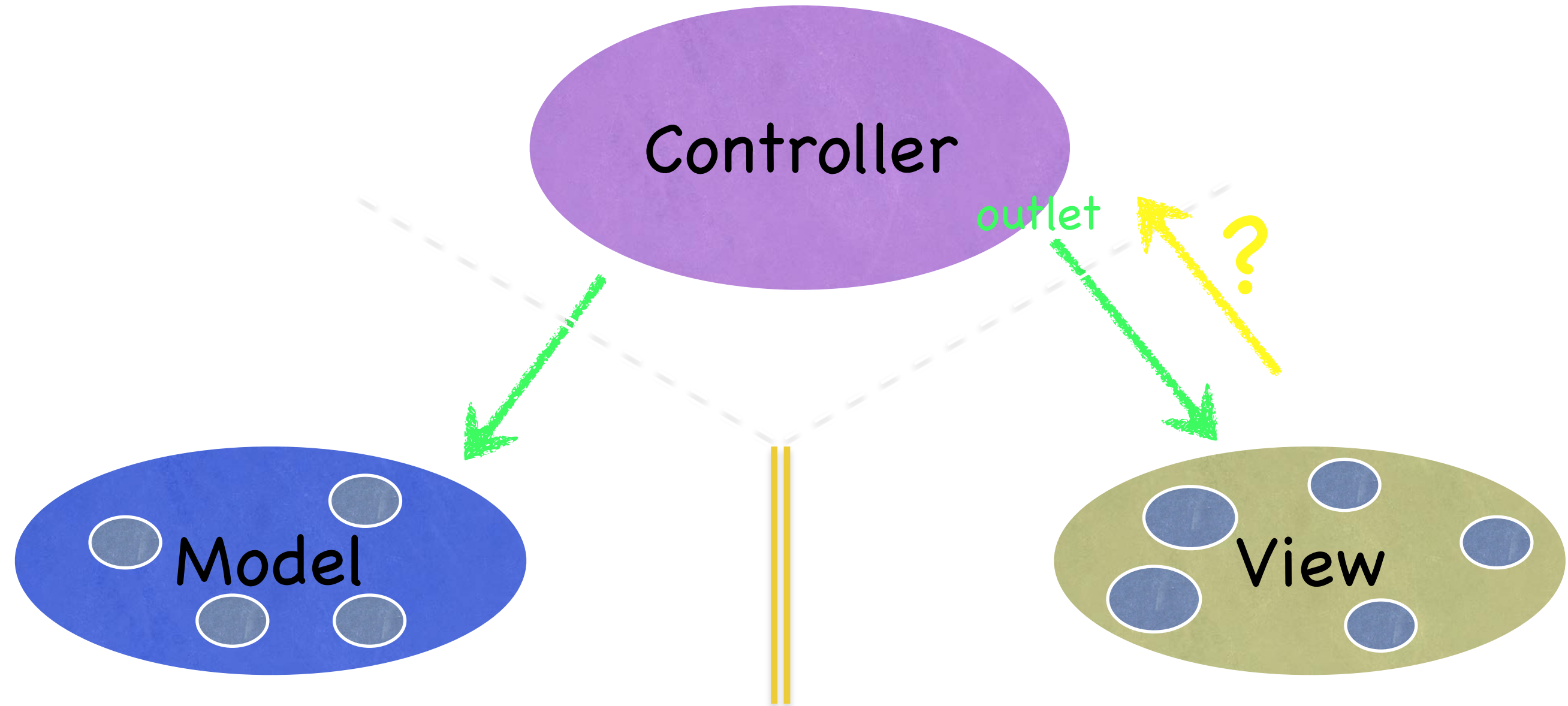
Controllers can also talk directly to their View.

MVC



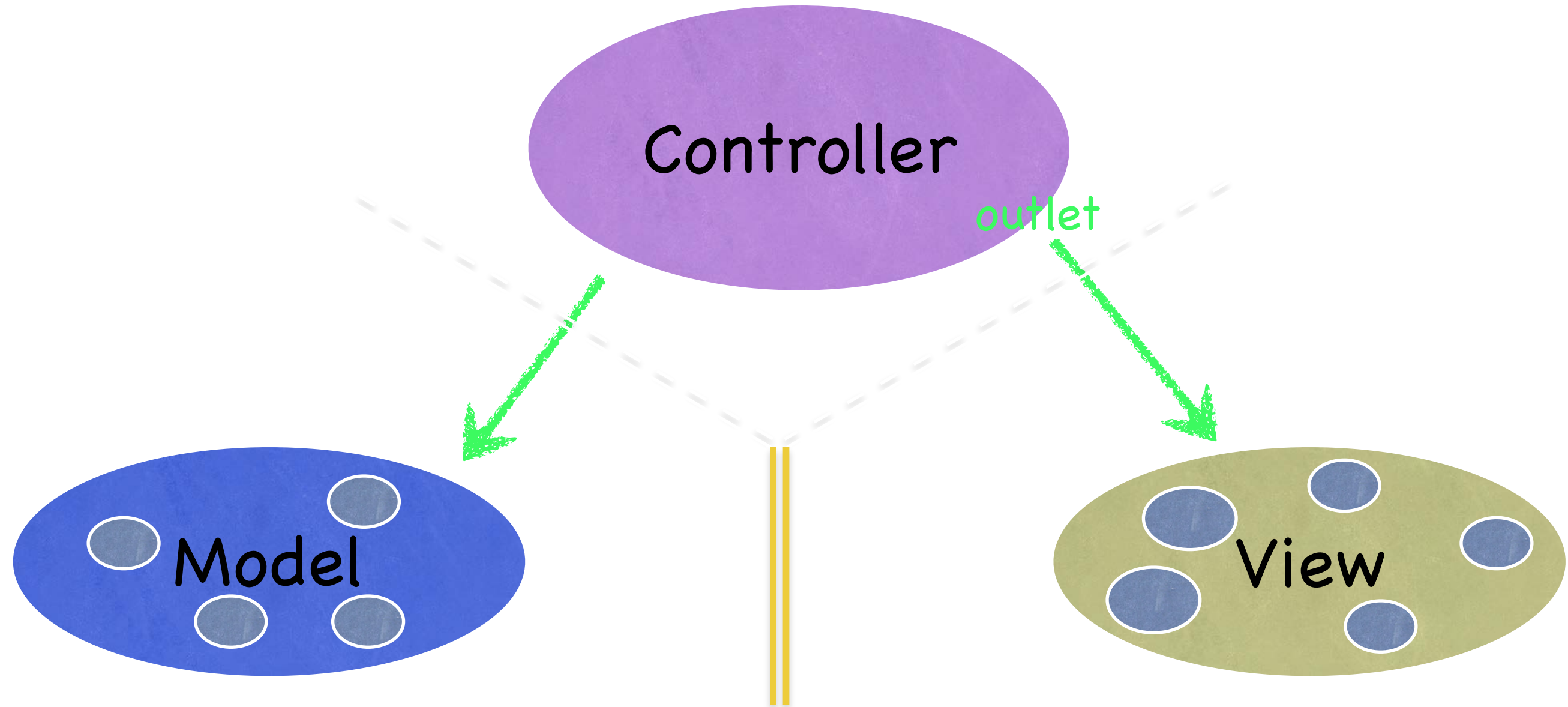
The Model and View should never speak to each other.

MVC

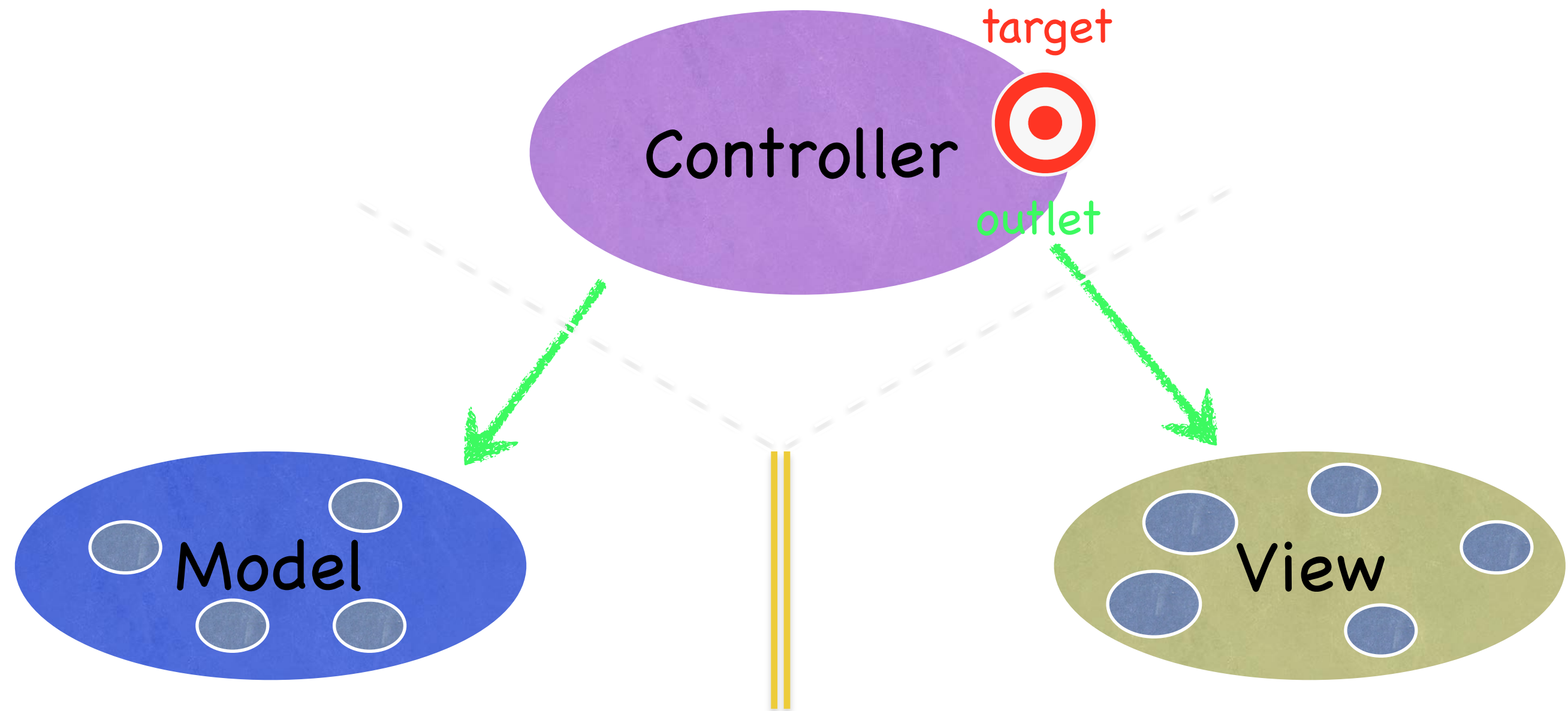


Can the View speak to its Controller?

MVC

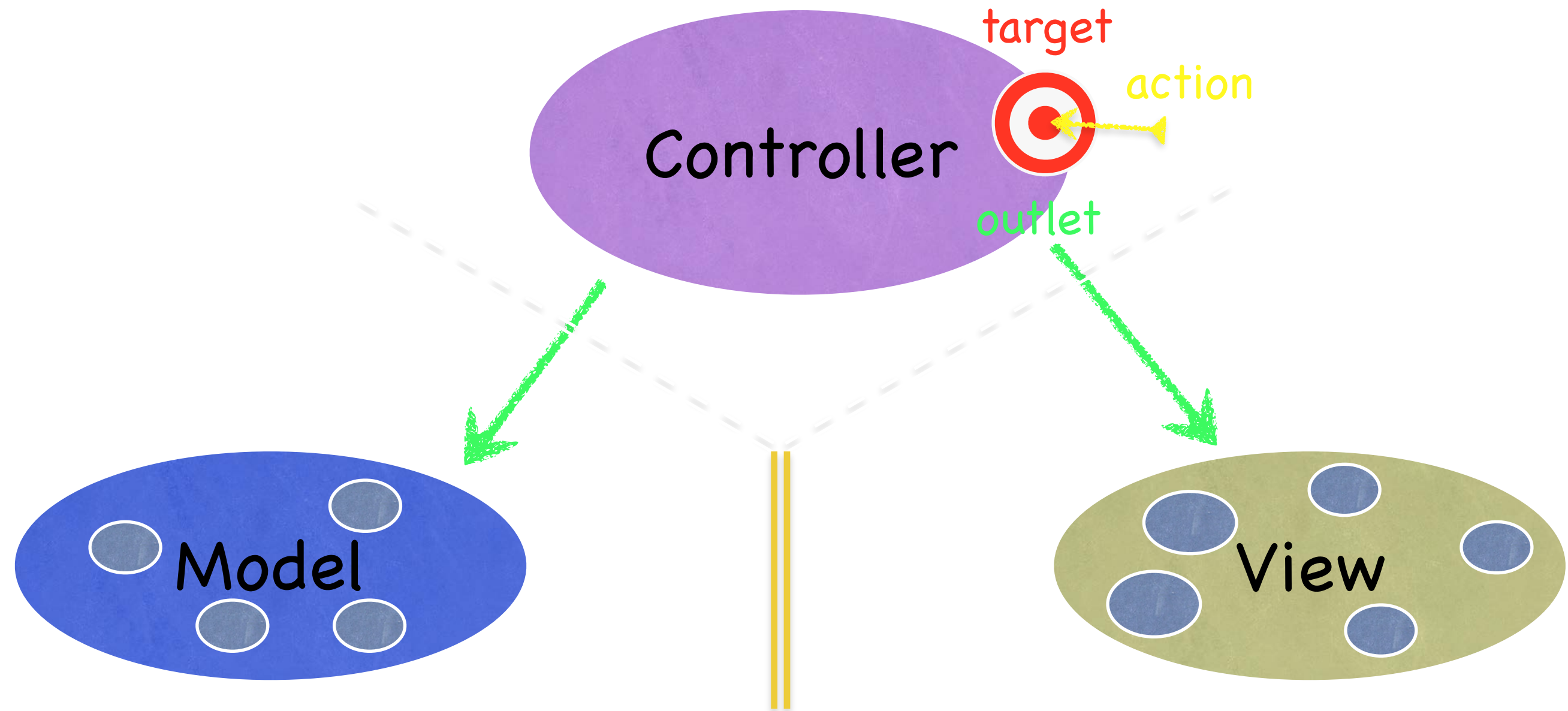


MVC



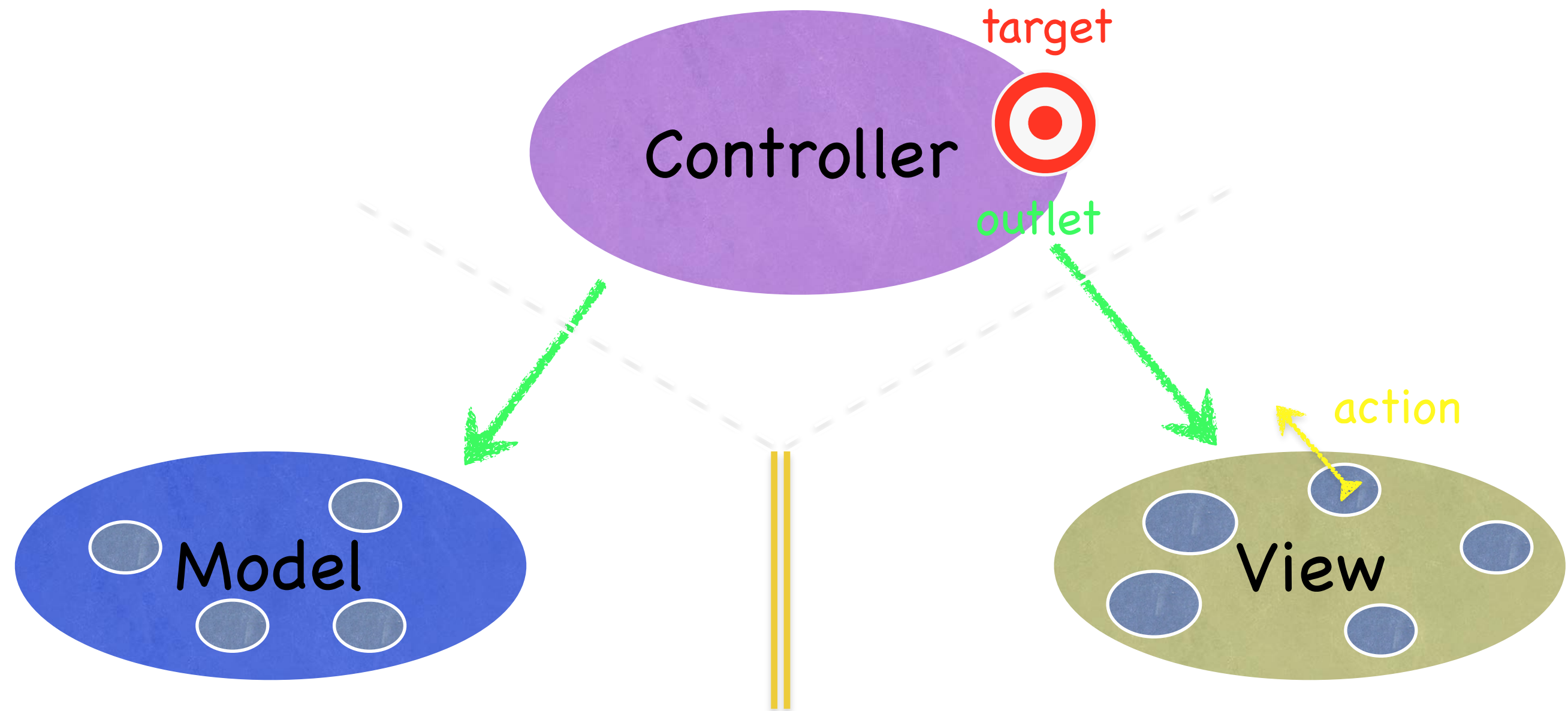
The Controller can drop a target on itself.

MVC



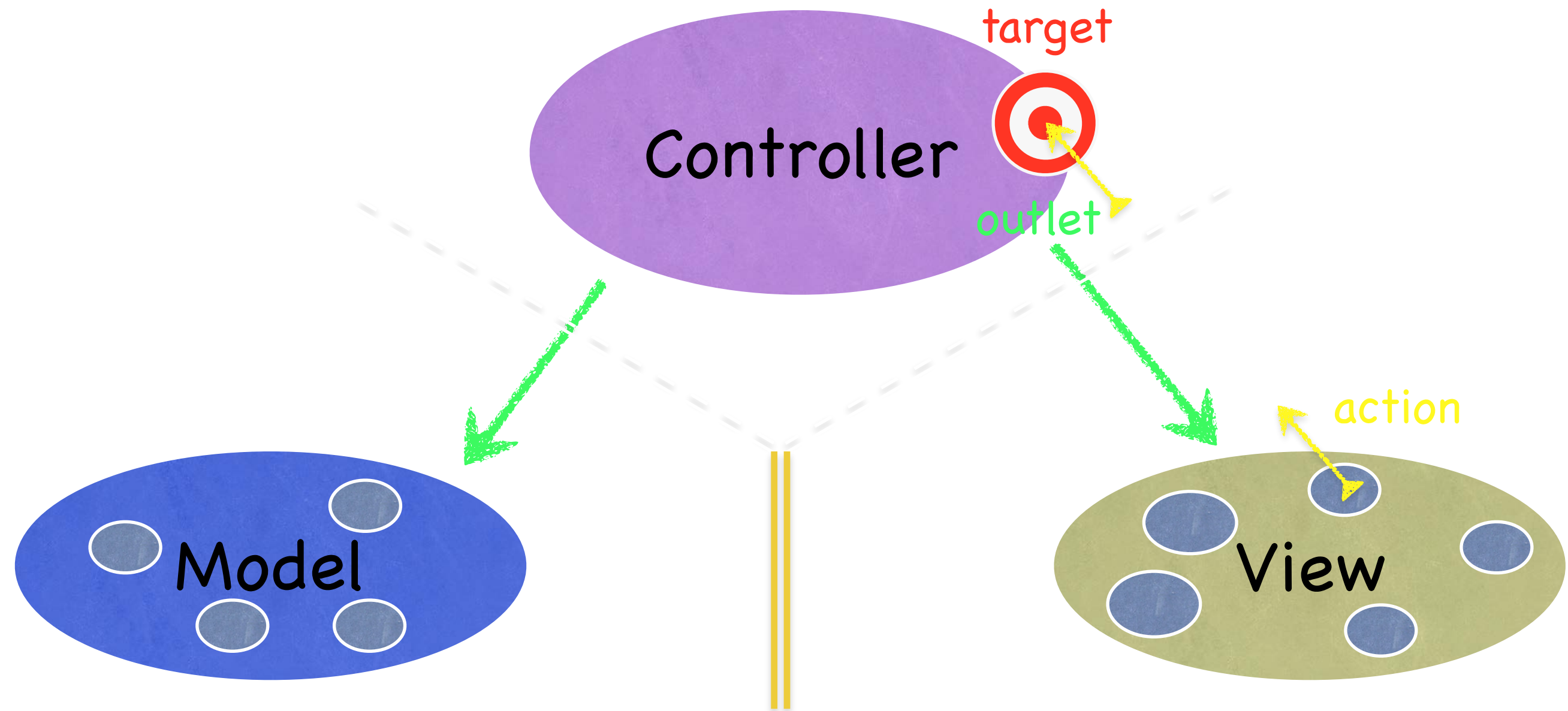
Then hand out an action to the View.

MVC



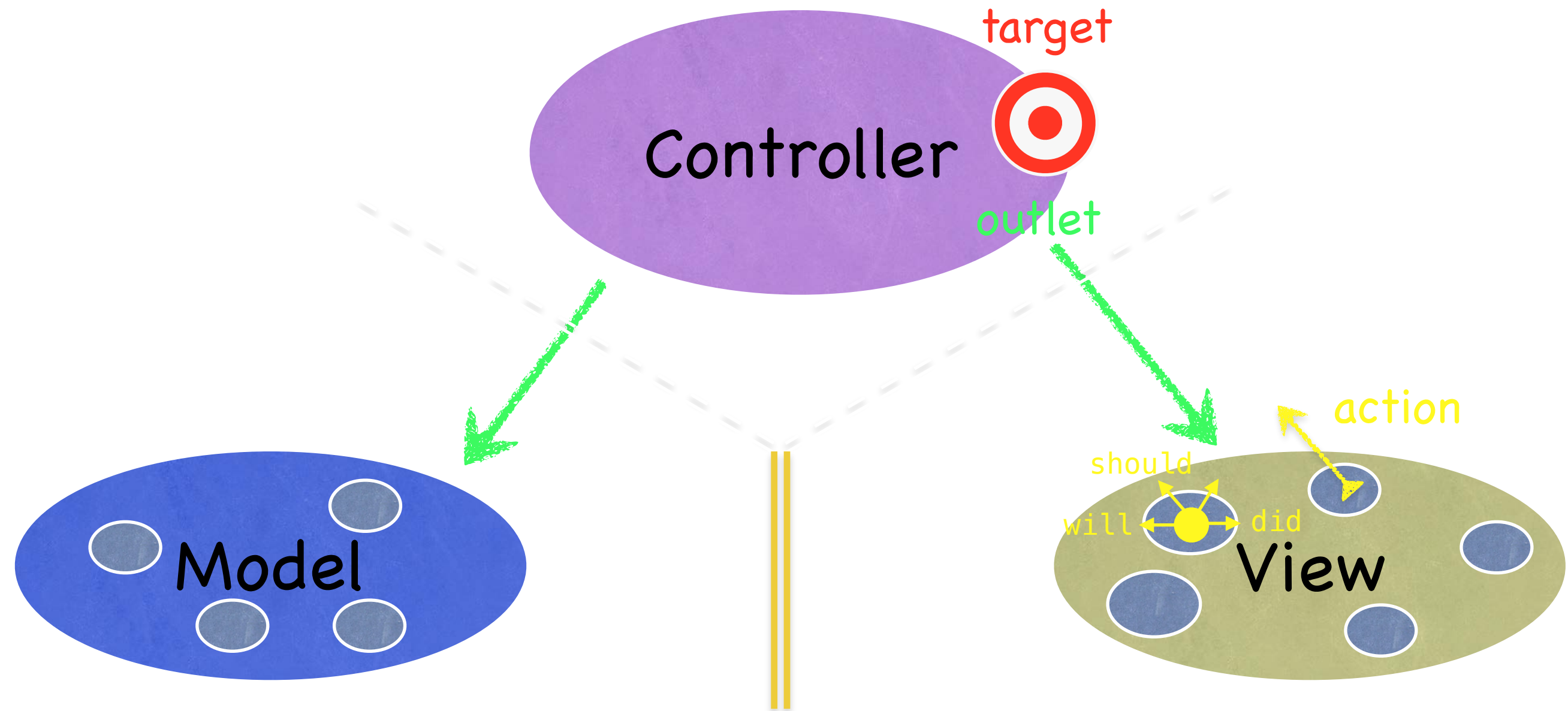
Then hand out an action to the View.

MVC



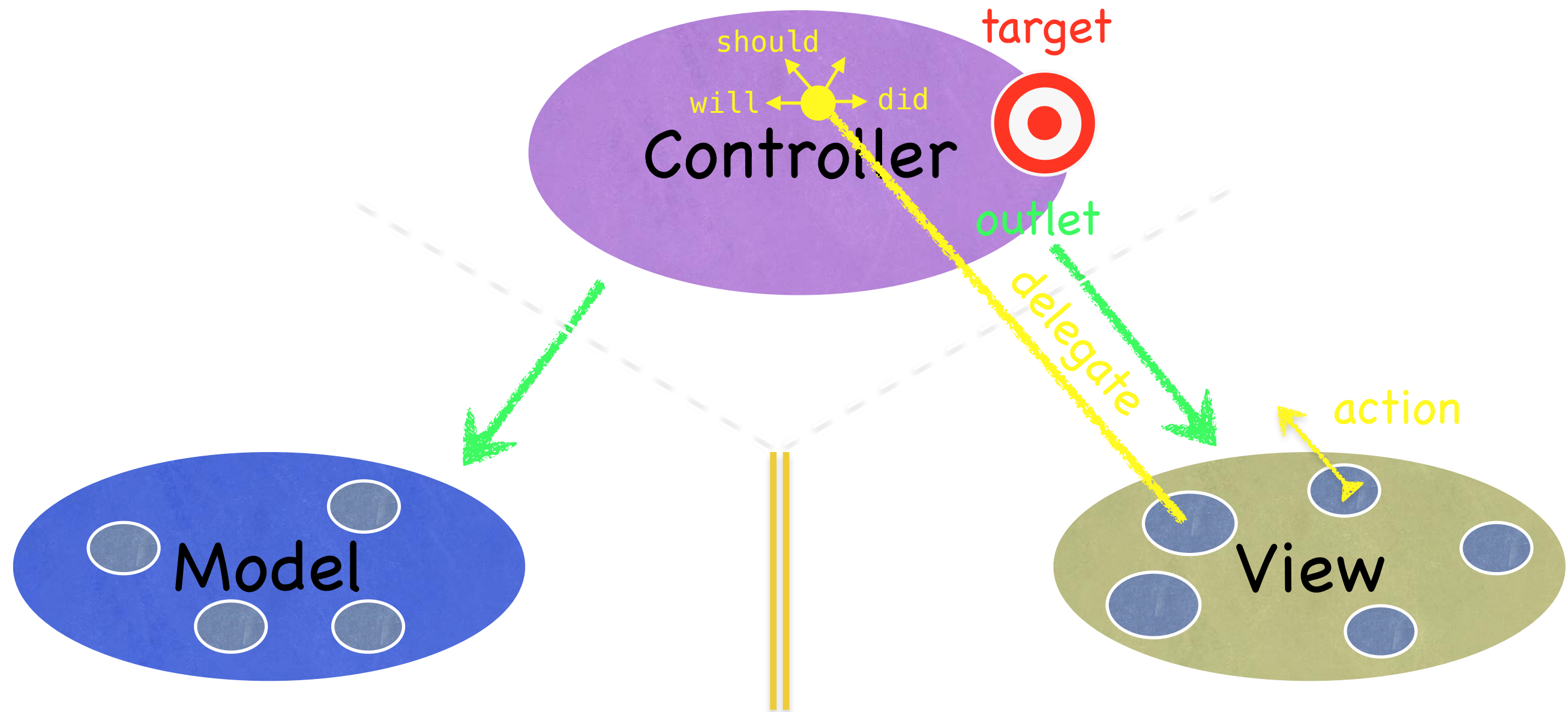
The View sends the action when things happen in the UI.

MVC



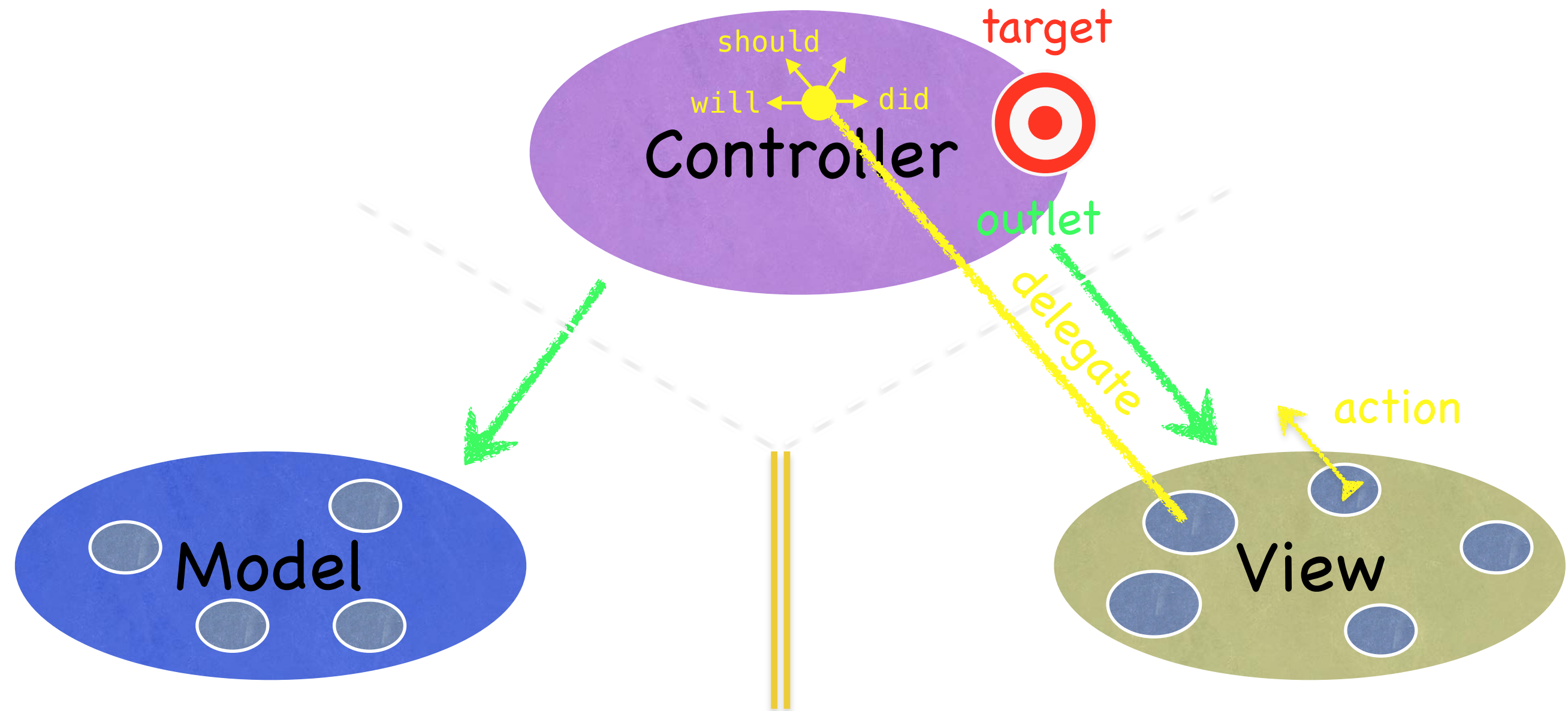
Sometimes the View needs to synchronize with the Controller.

MVC



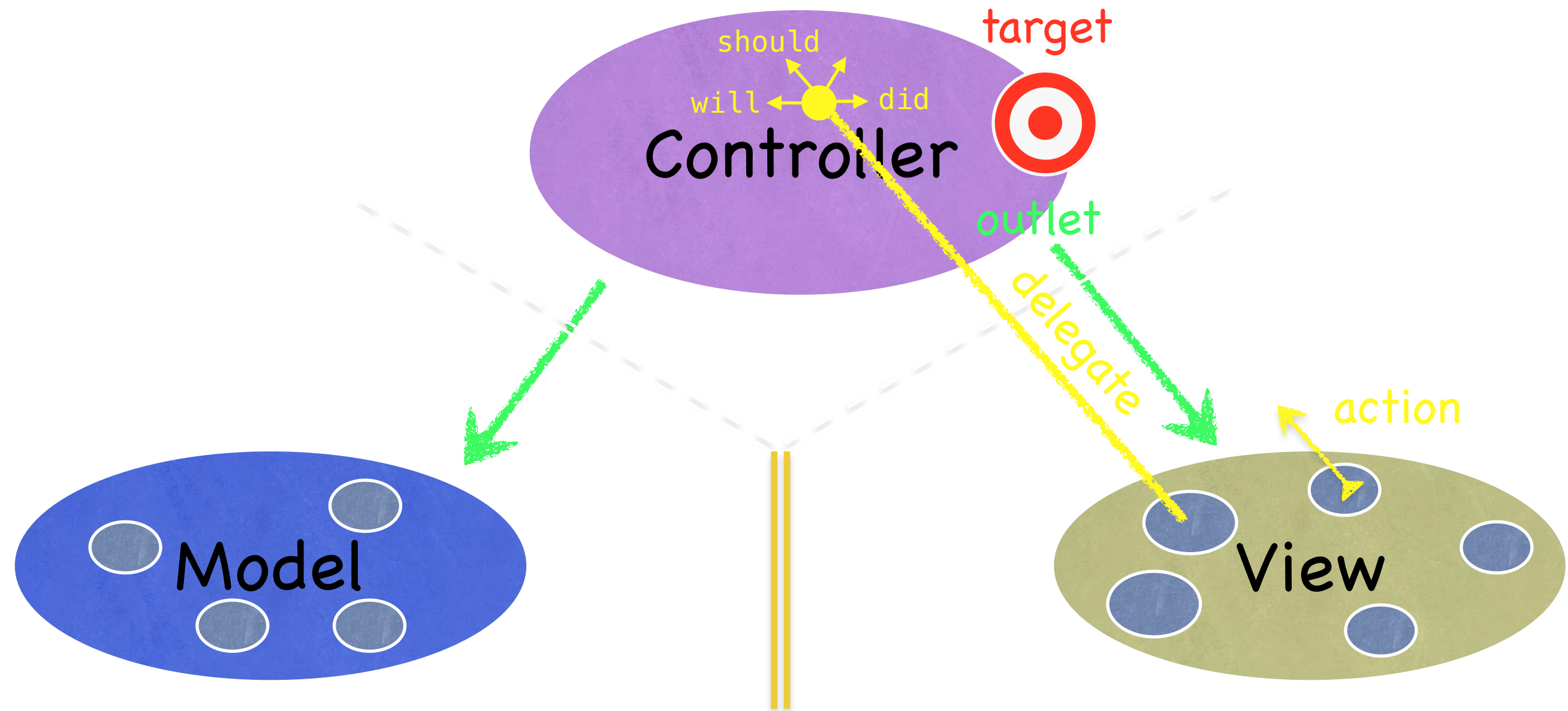
The Controller sets itself as the View's delegate.

MVC



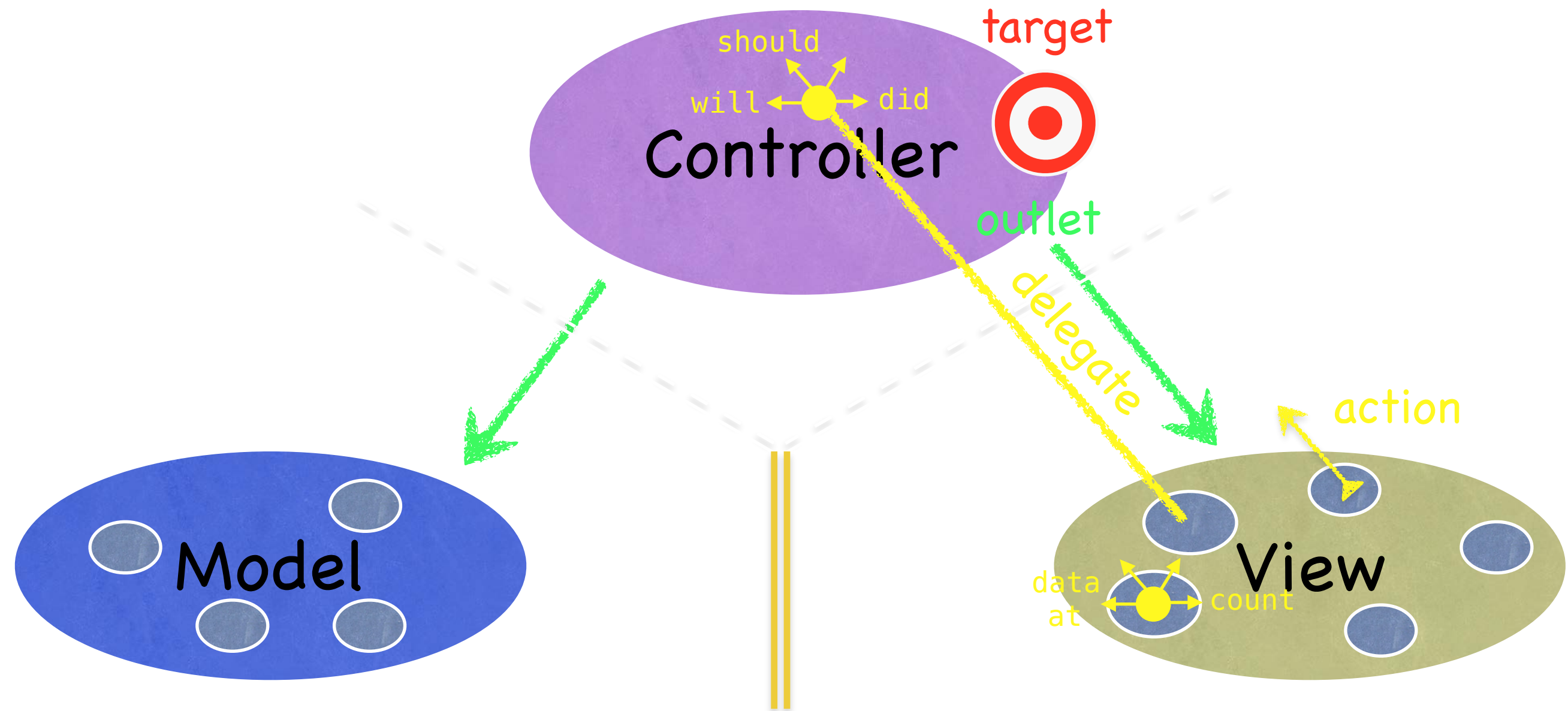
The delegate is set via a protocol (i.e. it's "blind" to class).

MVC



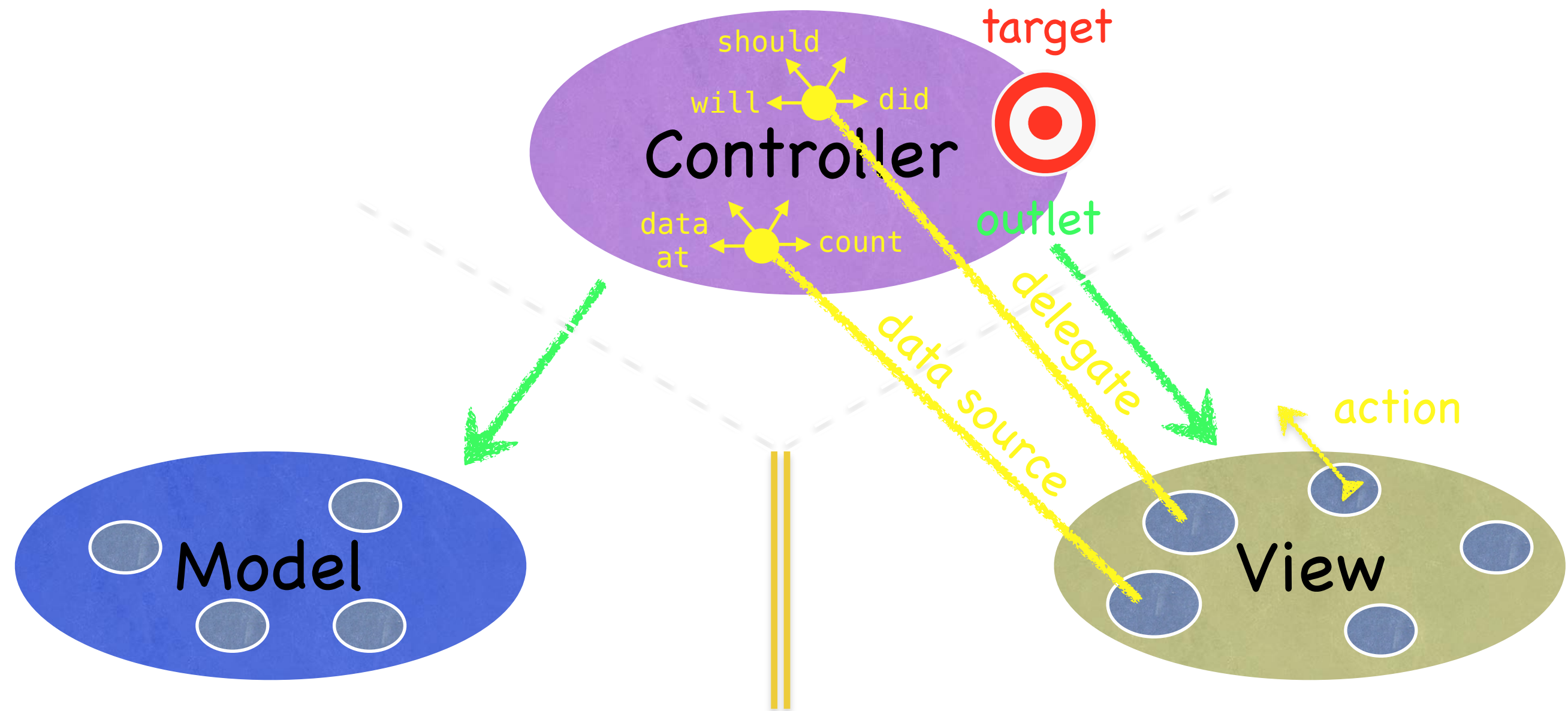
Views do not own the data they display.

MVC



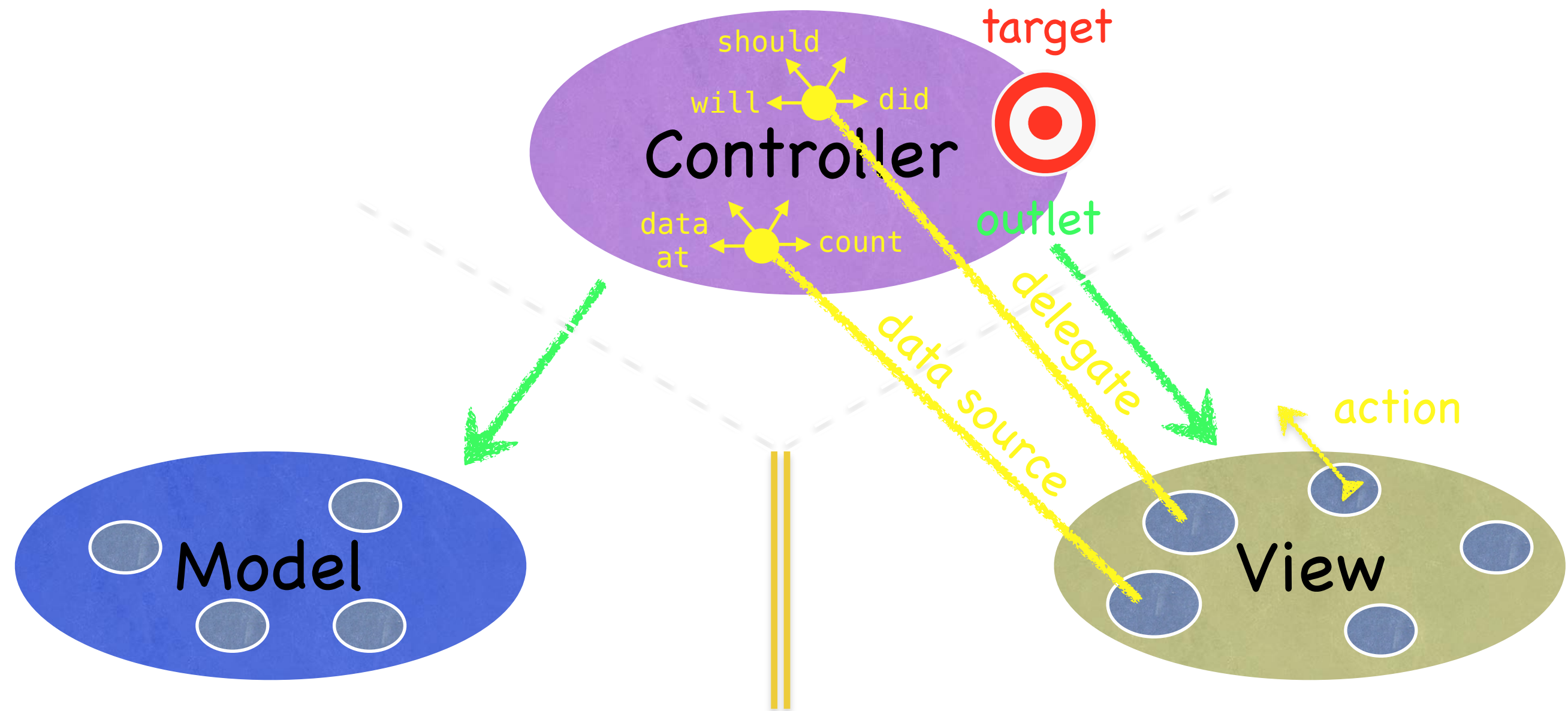
So, if needed, they have a protocol to acquire it.

MVC



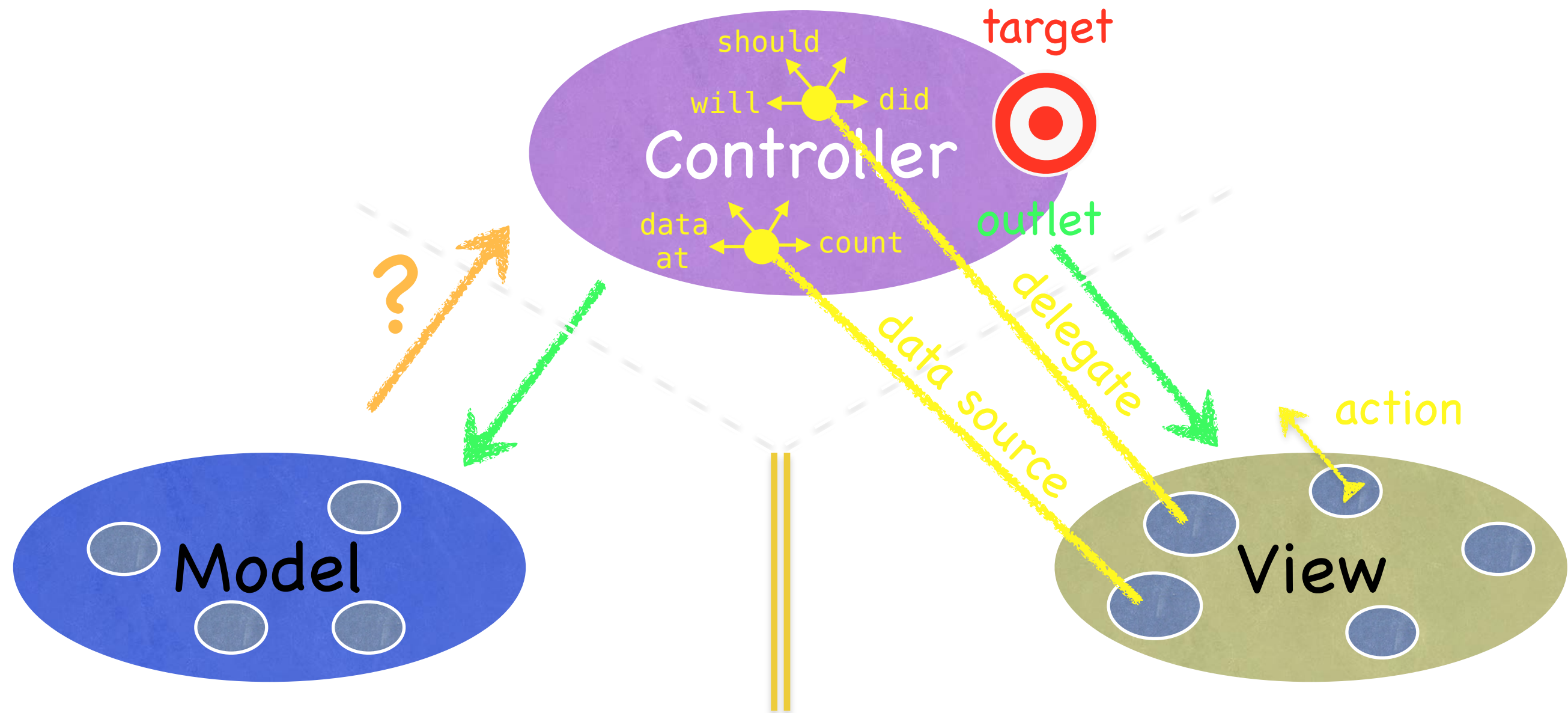
Controllers are almost always that data source (not Model!).

MVC



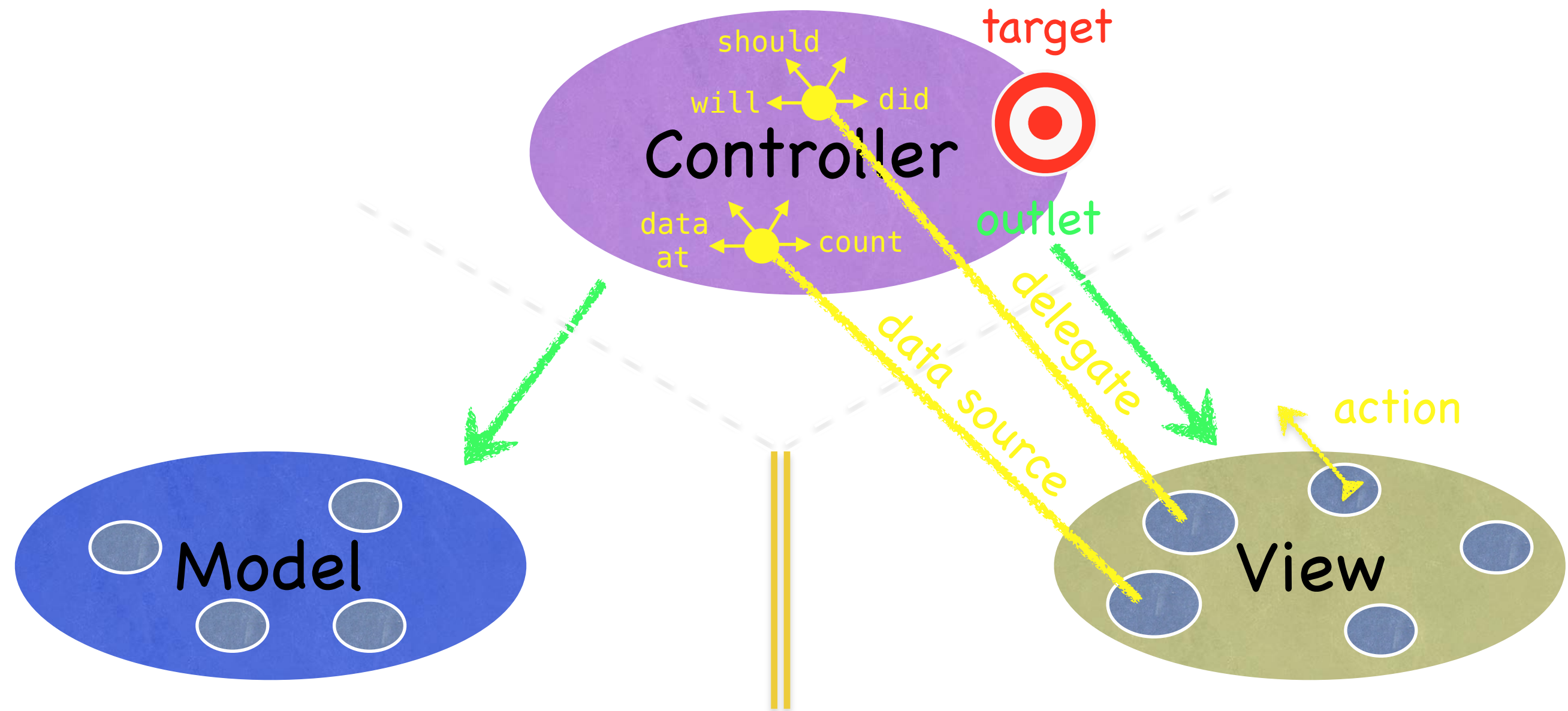
Controllers interpret/format Model information for the View.

MVC



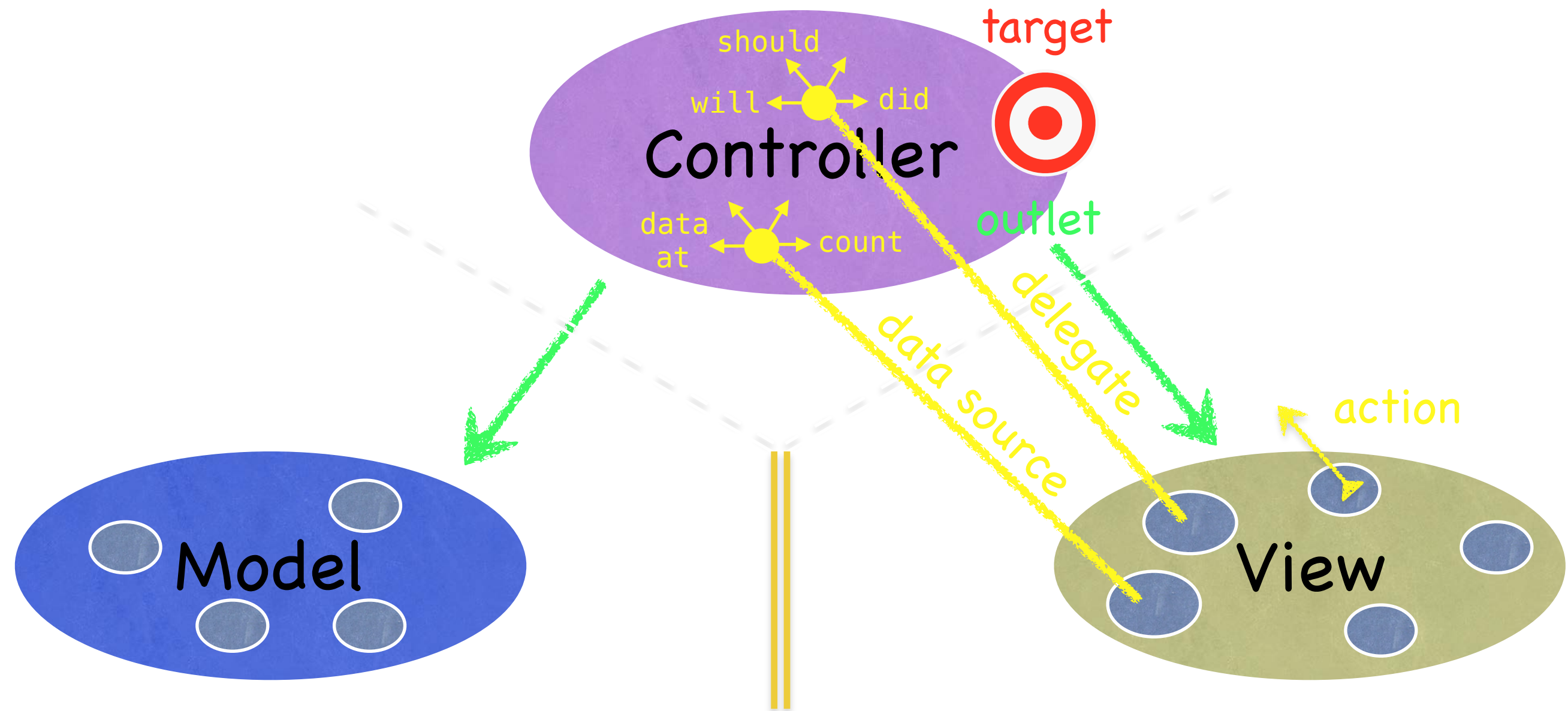
Can the Model talk directly to the Controller?

MVC



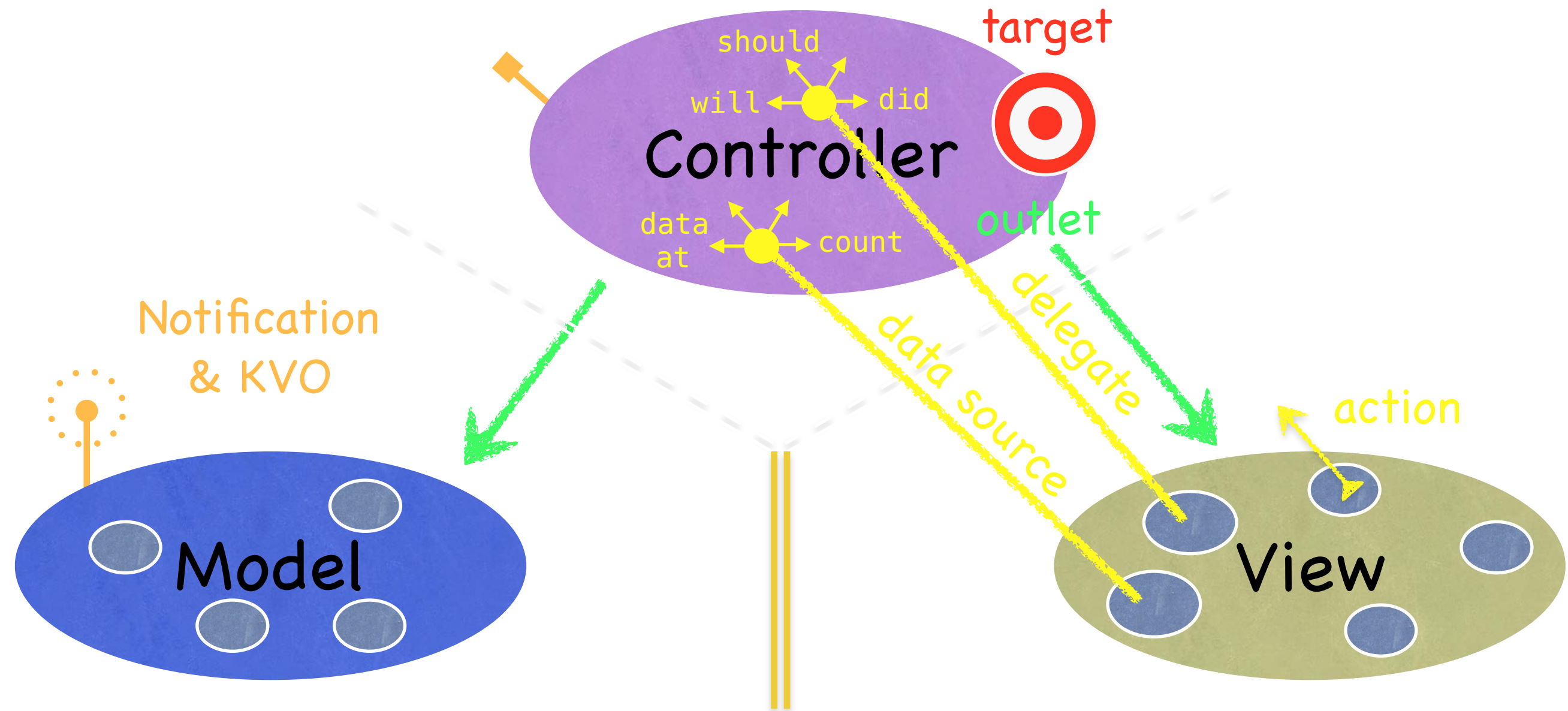
No. The Model is (should be) UI independent.

MVC



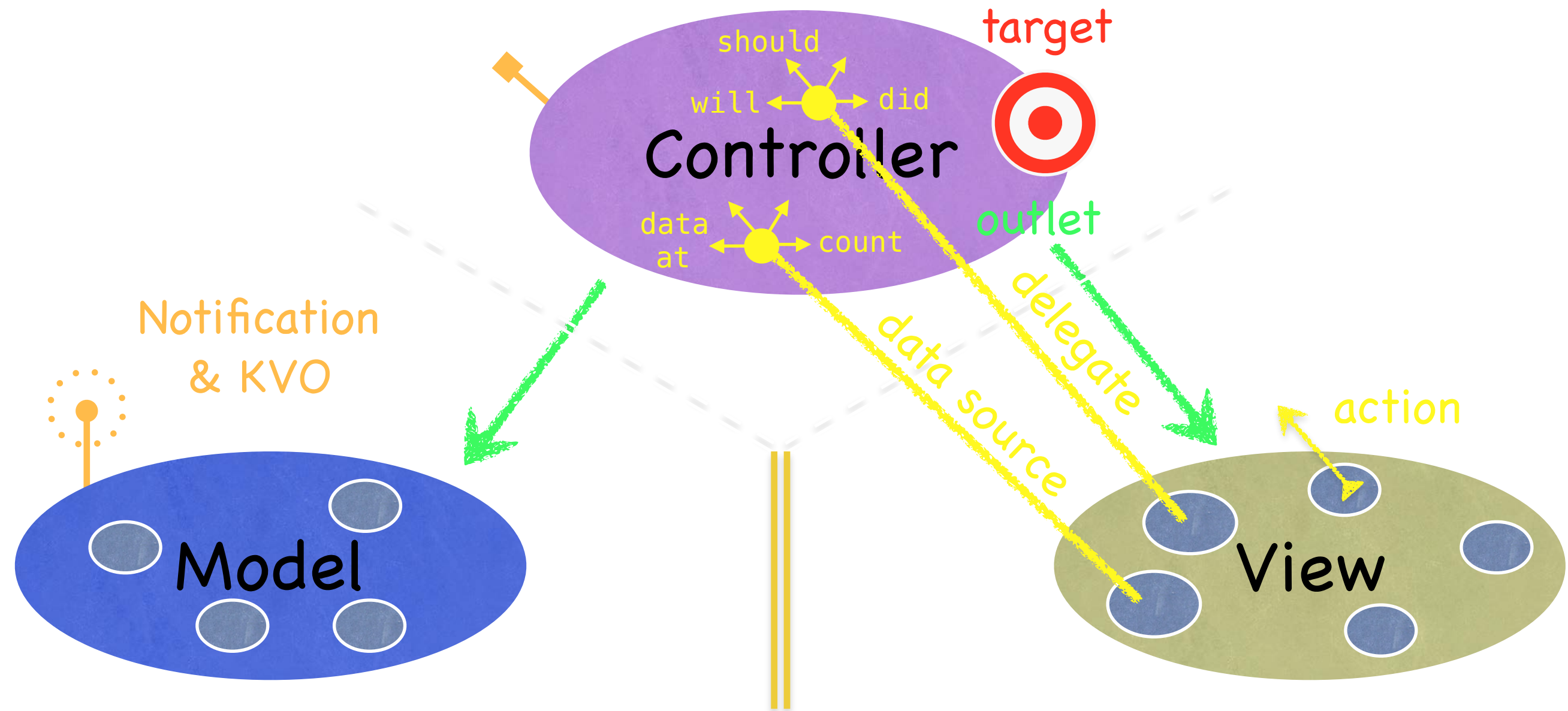
So what if the Model has information to update or something?

MVC



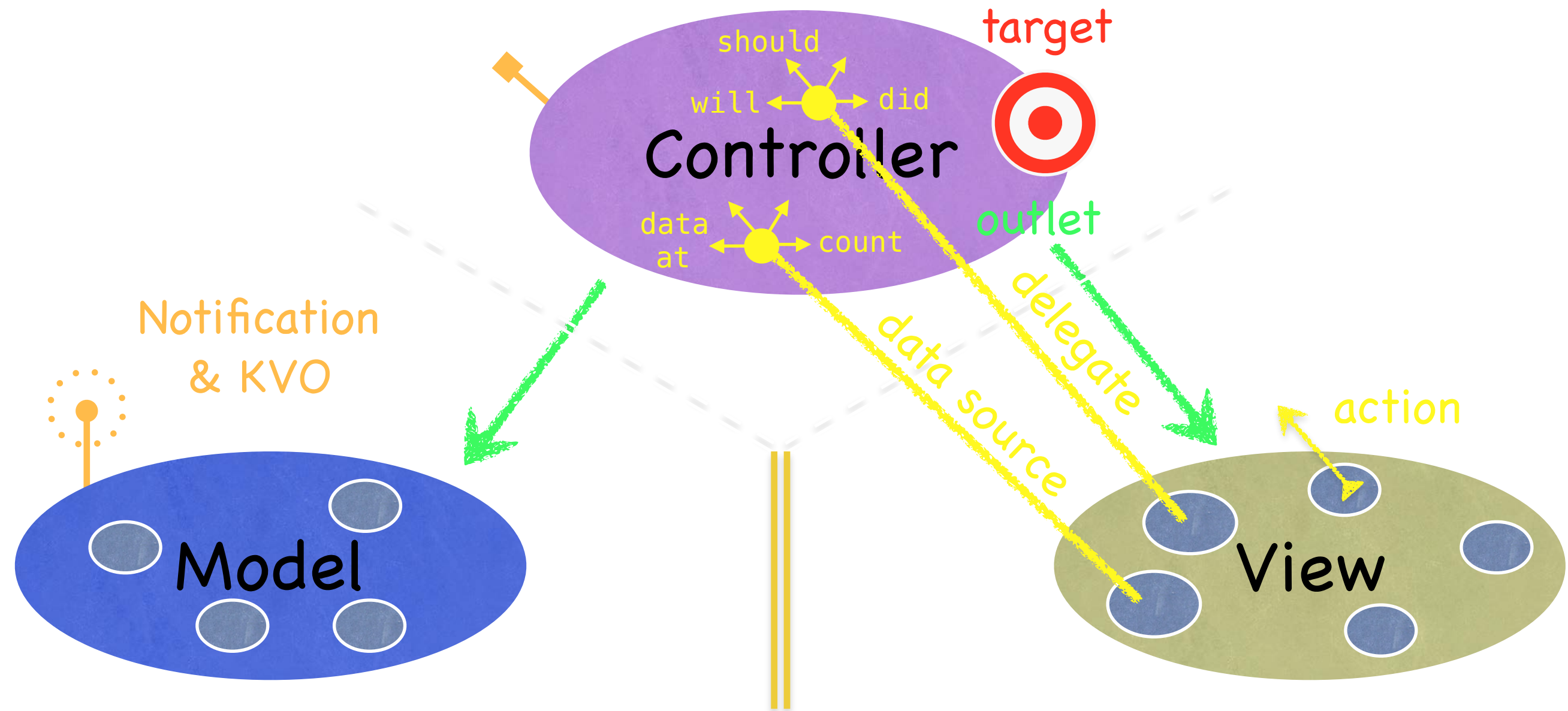
It uses a “radio station”-like broadcast mechanism.

MVC



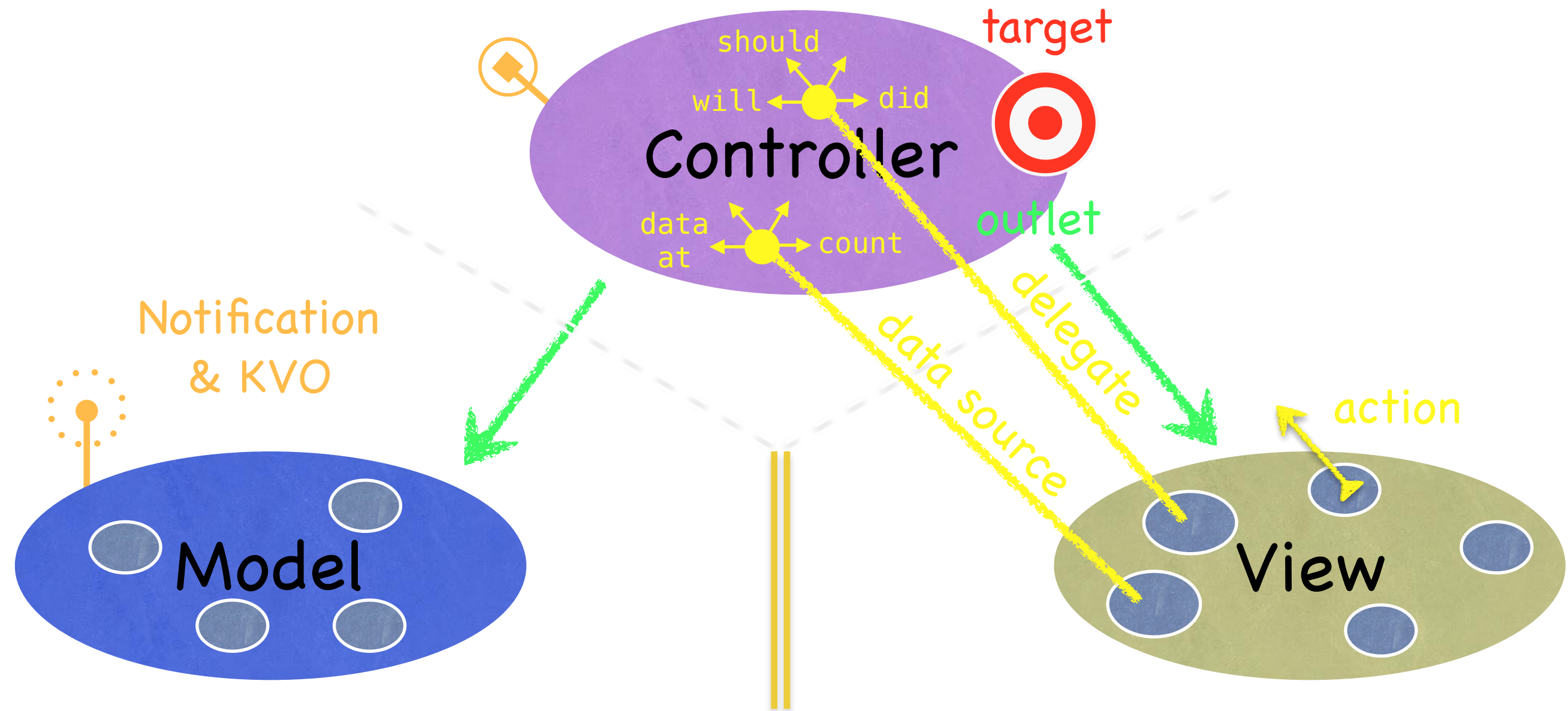
Controllers (or other Model) "tune in" to interesting stuff.

MVC



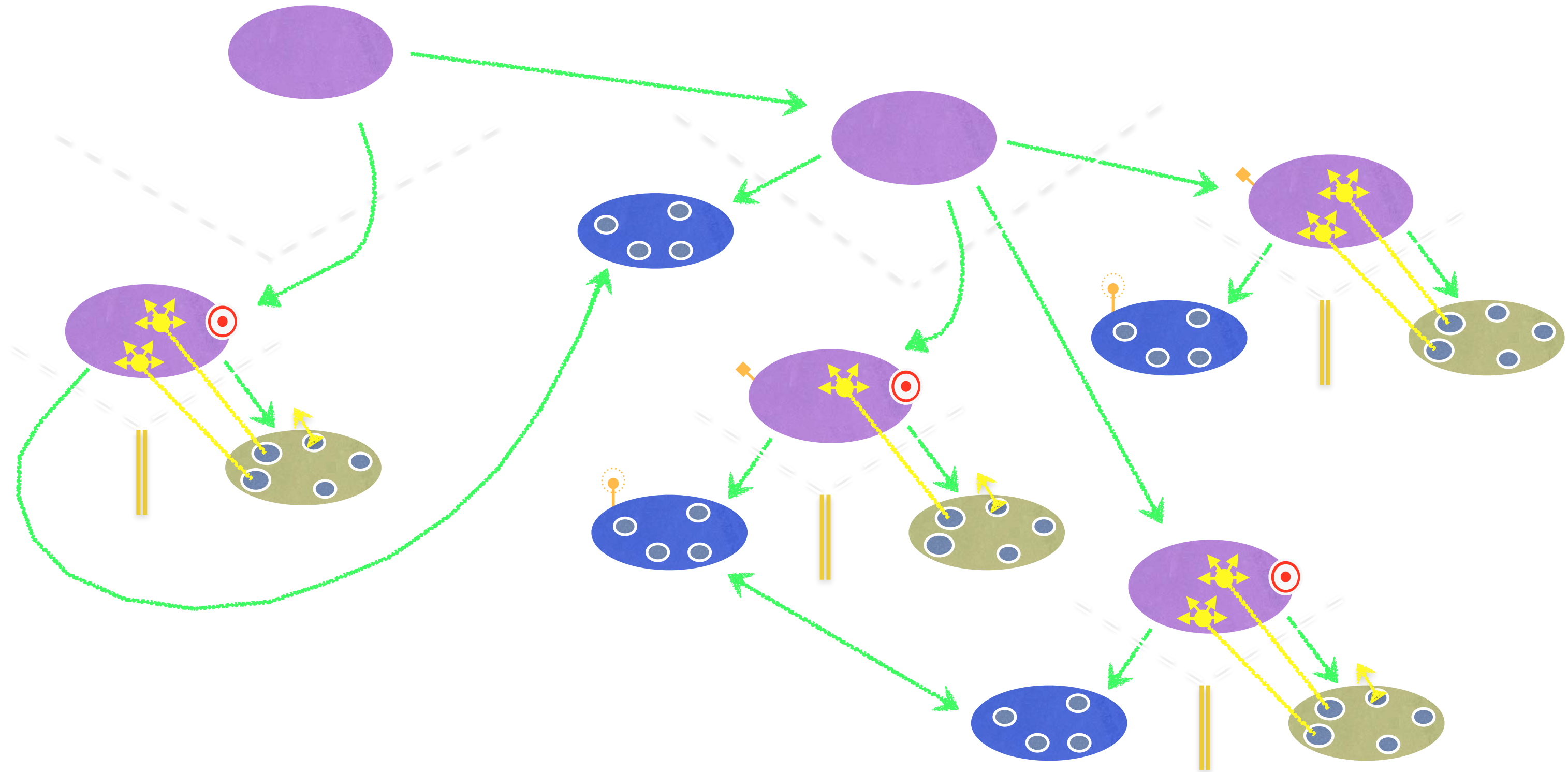
A View might "tune in," but probably not to a Model's "station."

MVC



Now combine MVC groups to make complicated programs ...

MVCs working together



MVCs not working together

