Universitatea din Bucuresti
Facultatea de Matematica si Informatica

AppleLab @ UniBuc
Dezvoltare aplicatii iOS cu Swift 3

- CURS 5 -

Model-View-Controller (MVC), Error Handling, UIScrollView, Multithreading, Gestures

05.04.2017

iOS Course - 5

Today - Part 1

Model-View-Controller:

- Object-Oriented Pattern

Error Handling:

- try

UIScrollView:

- Scroll si zoom asupra "lucrurilor" mari pe un ecran mic

MVC

Imparte obiectele* (se refera la instantele claselor) din aplicatie in 3 categorii: Model, View si Controller

MVC

Imparte obiectele* (se refera la instantele claselor) din aplicatie in 3 categorii: Model, View si Controller

MVC

CONTROLLER

MODEL

VIEW

MODEL - Datele. Ce face aplicatia? (nu cum arata)
Modelul defineste logica datelor. *Ex: calculatorBrain (CS2)*

MVC

MODEL - Datele. Ce face aplicatia? (nu cum arata)
Modelul defineste logica datelor. *Ex: calculatorBrain (CS2)*

MVC

MODEL

VIEW - UI Logic. Cum arata si se simte aplicatia?
Se "deseneaza" pe ecran si raspunde la actiunile userului

MVC

VIEW - UI Logic. Cum arata si se simte aplicatia?
Se "deseneaza" pe ecran si raspunde la actiunile userului

MVC

CONTROLLER - Cum este prezentat modelul utilizatorului prin intermediul view-ului?

MVC

CONTROLLER - Cum este prezentat modelul utilizatorului prin intermediul view-ului?

MVC

CONTROLLER

MODEL

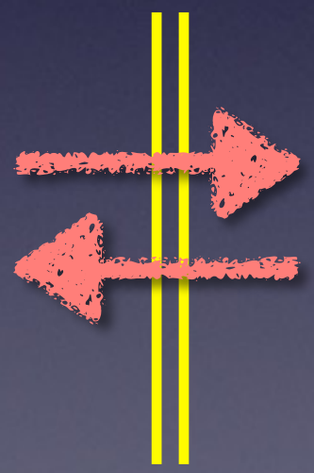
VIEW

Model si **View** cunosc logica datelor si UI-ului
Controller-ul vine si pune in aplicare cele doua

MVC

CONTROLLER

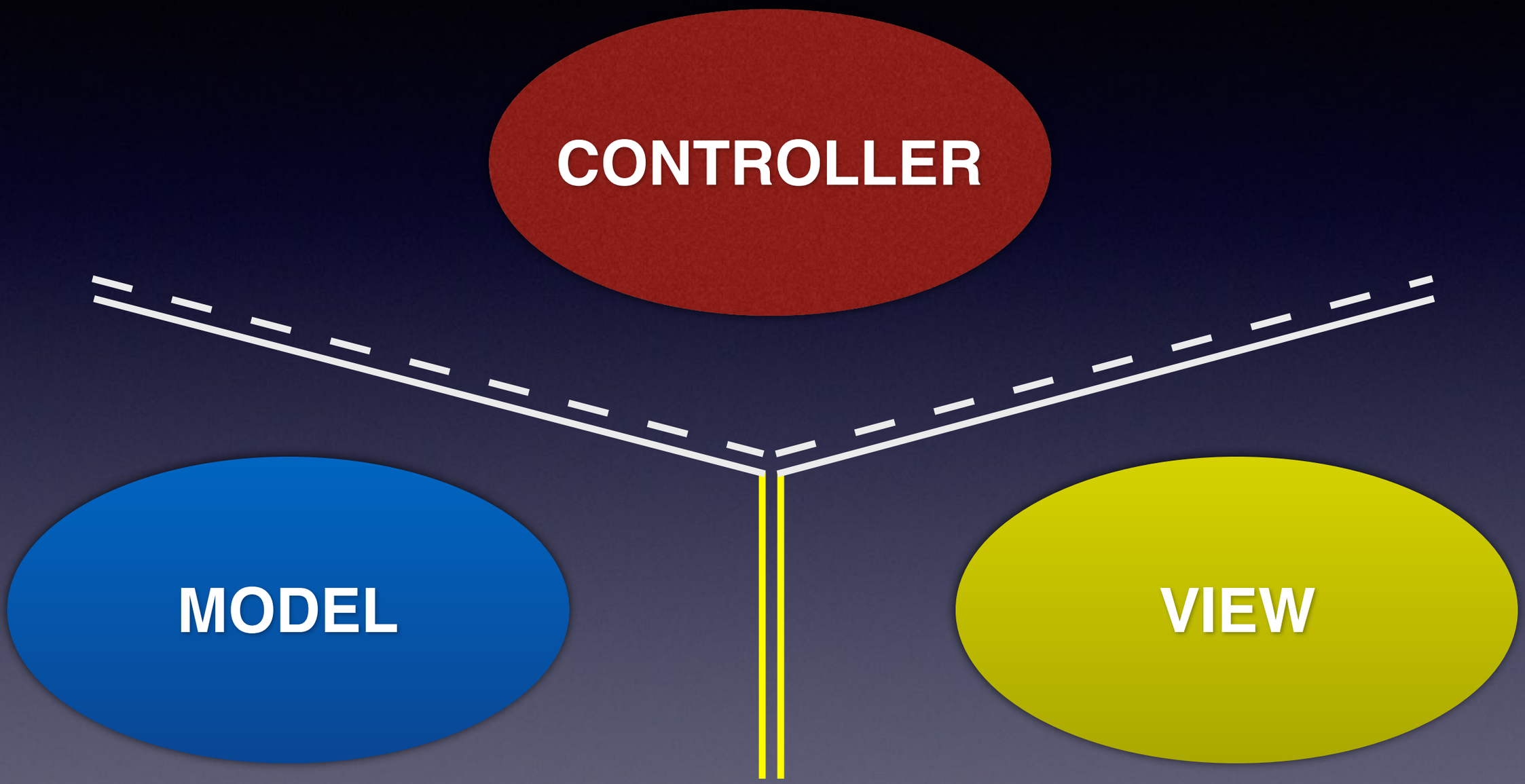
MODEL



VIEW

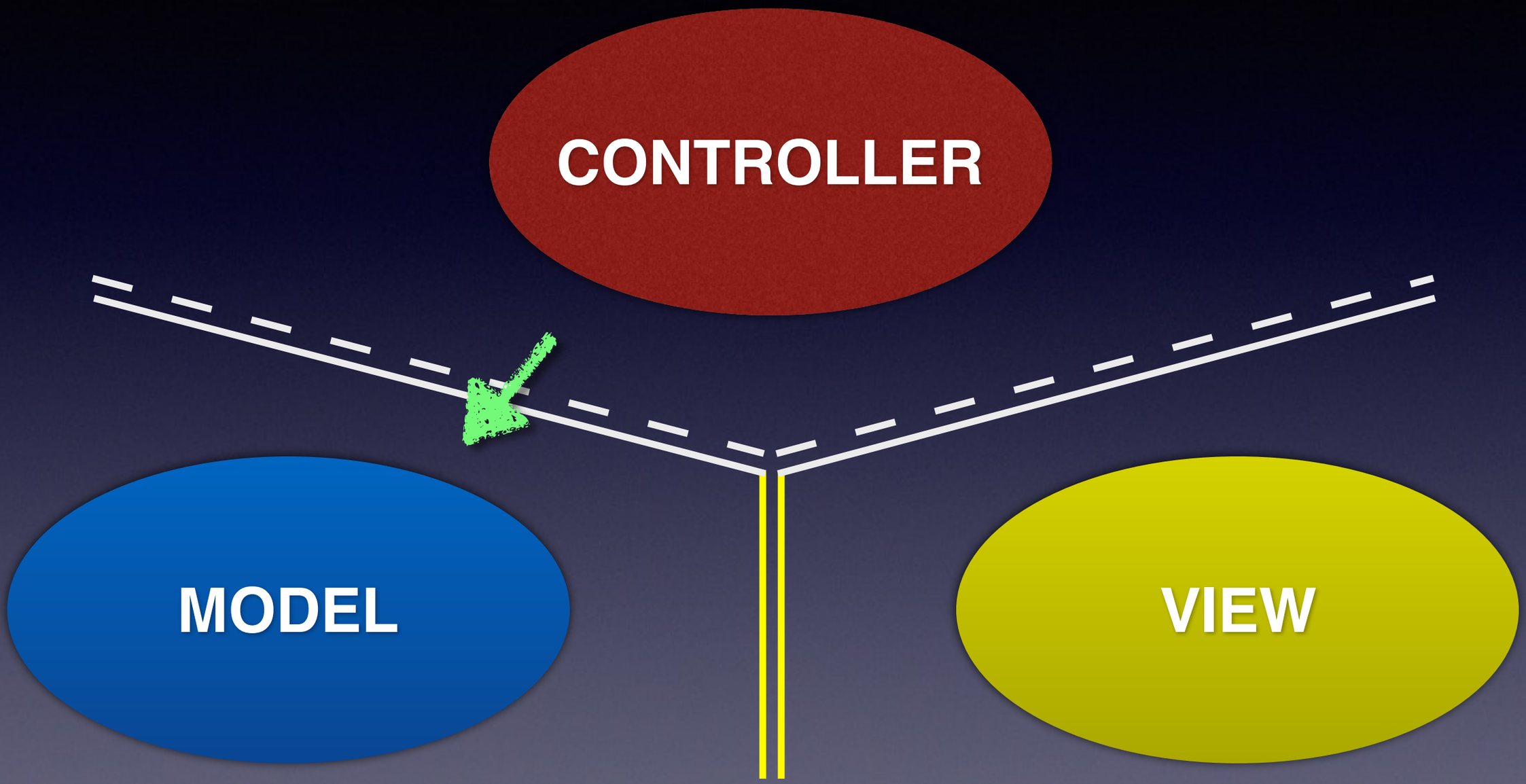
In MVC **Model** si **View** nu vor putea **NICIODATA** comunica intre ele.

MVC



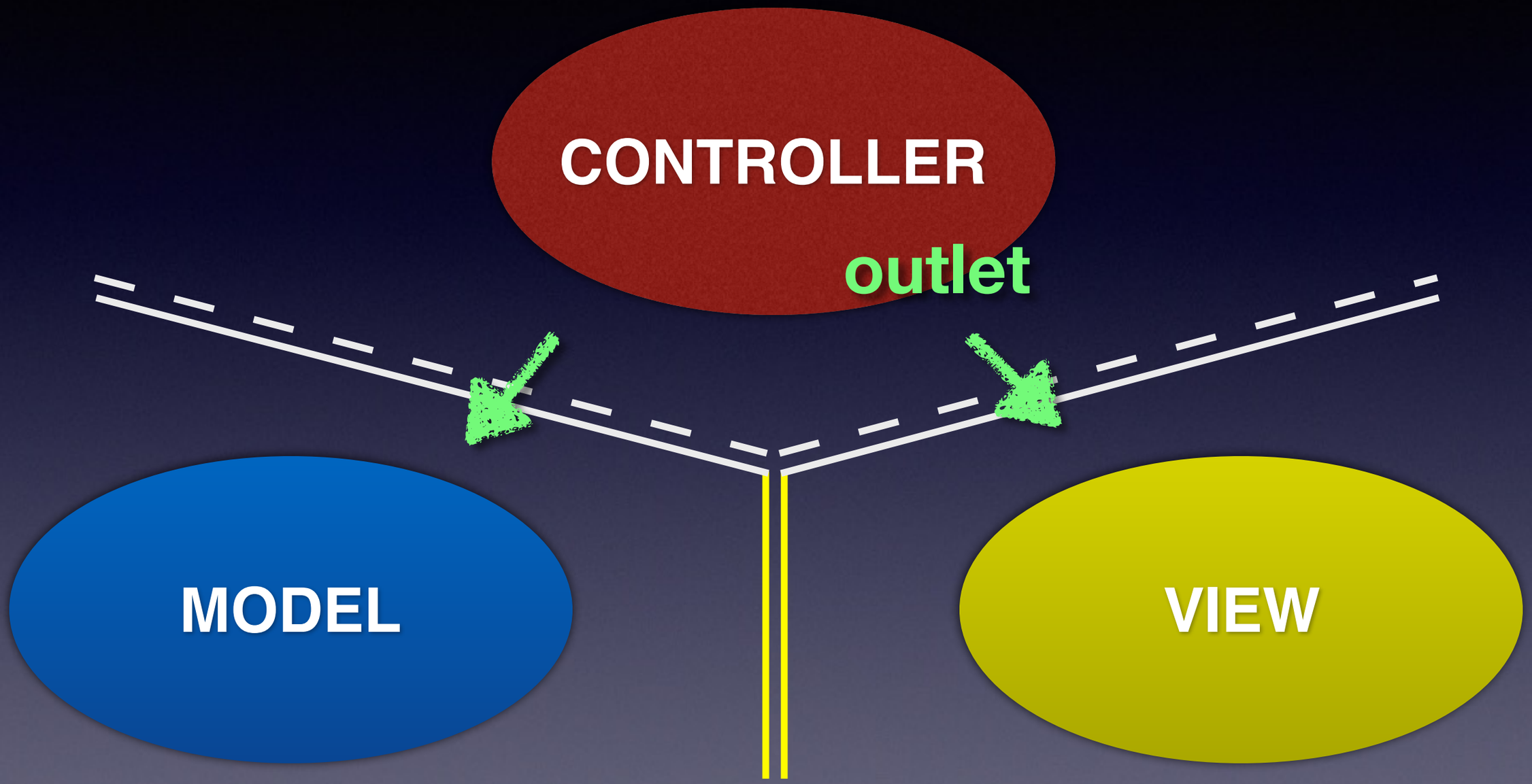
Comunicarea se va face mereu prin **Controller**
Controller-ul controleaza modul in care logica datelor si a elementelor vizuale se intersecteaza

MVC



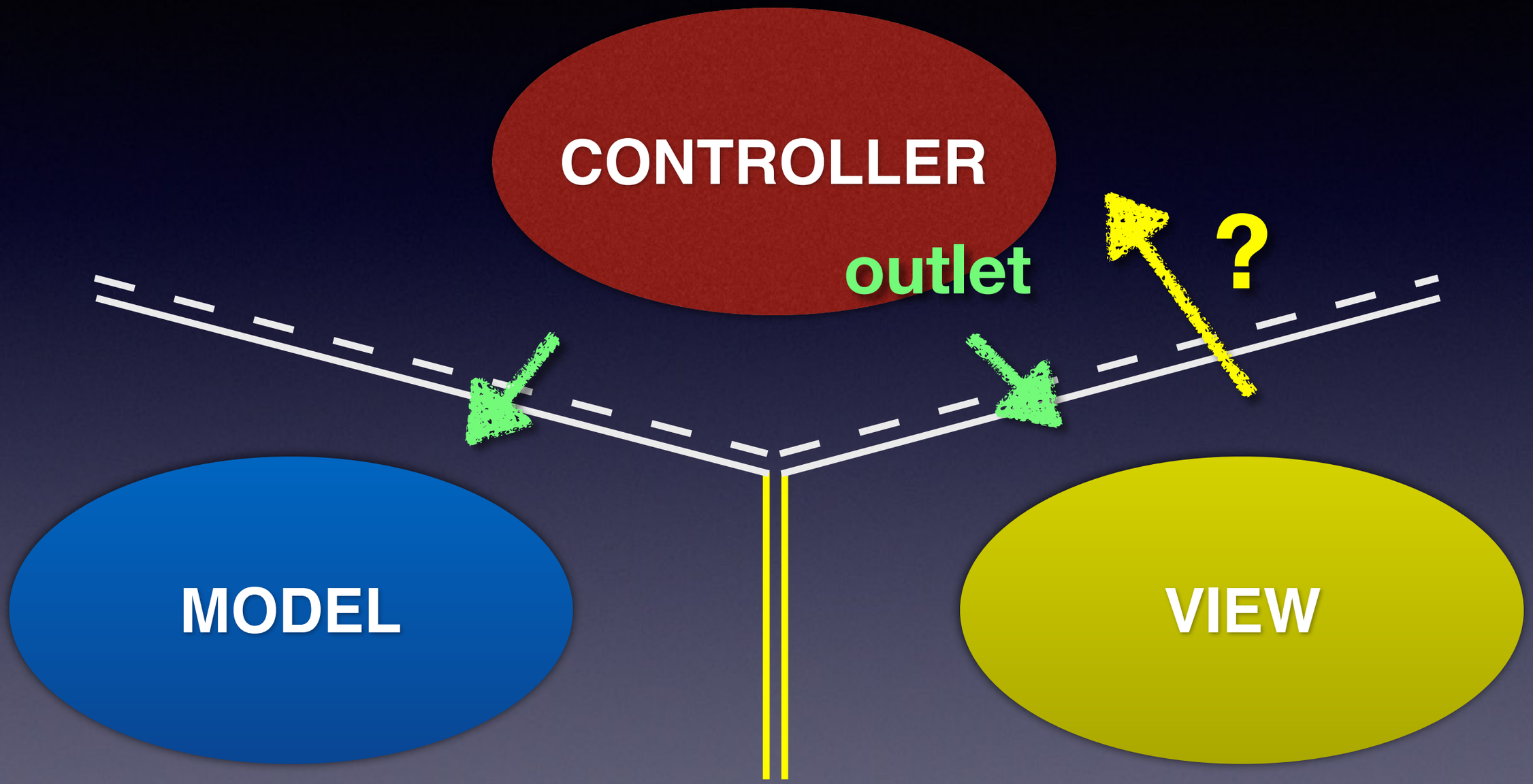
Controller-ul poate mereu "vorbi" cu **Modelul**

MVC



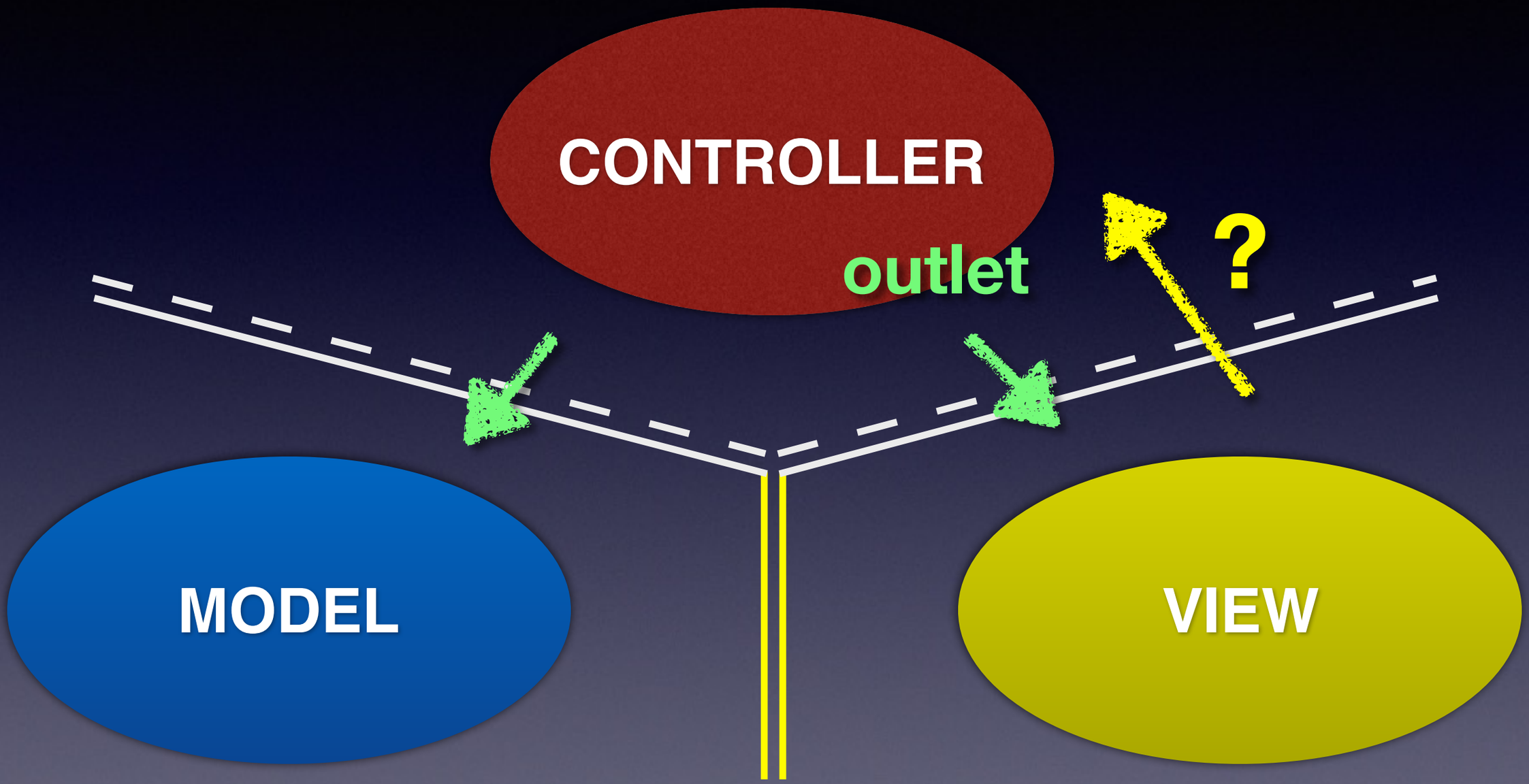
Controller-ul poate mereu "vorbi" cu **VIEW**-ul

MVC



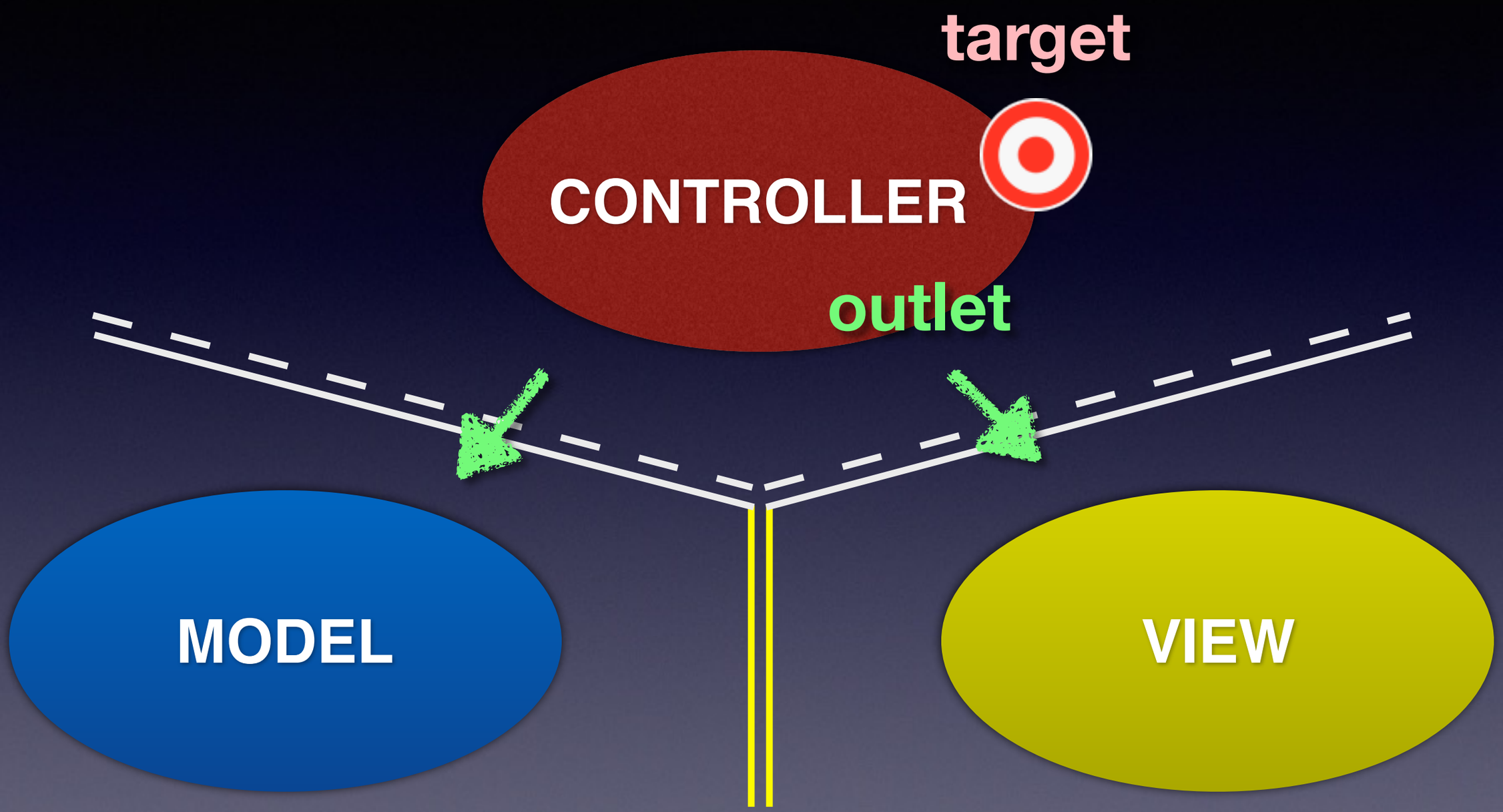
Dar oare **VIEW**-ul poate "vorbi" cu **Controller**-ul ?

MVC



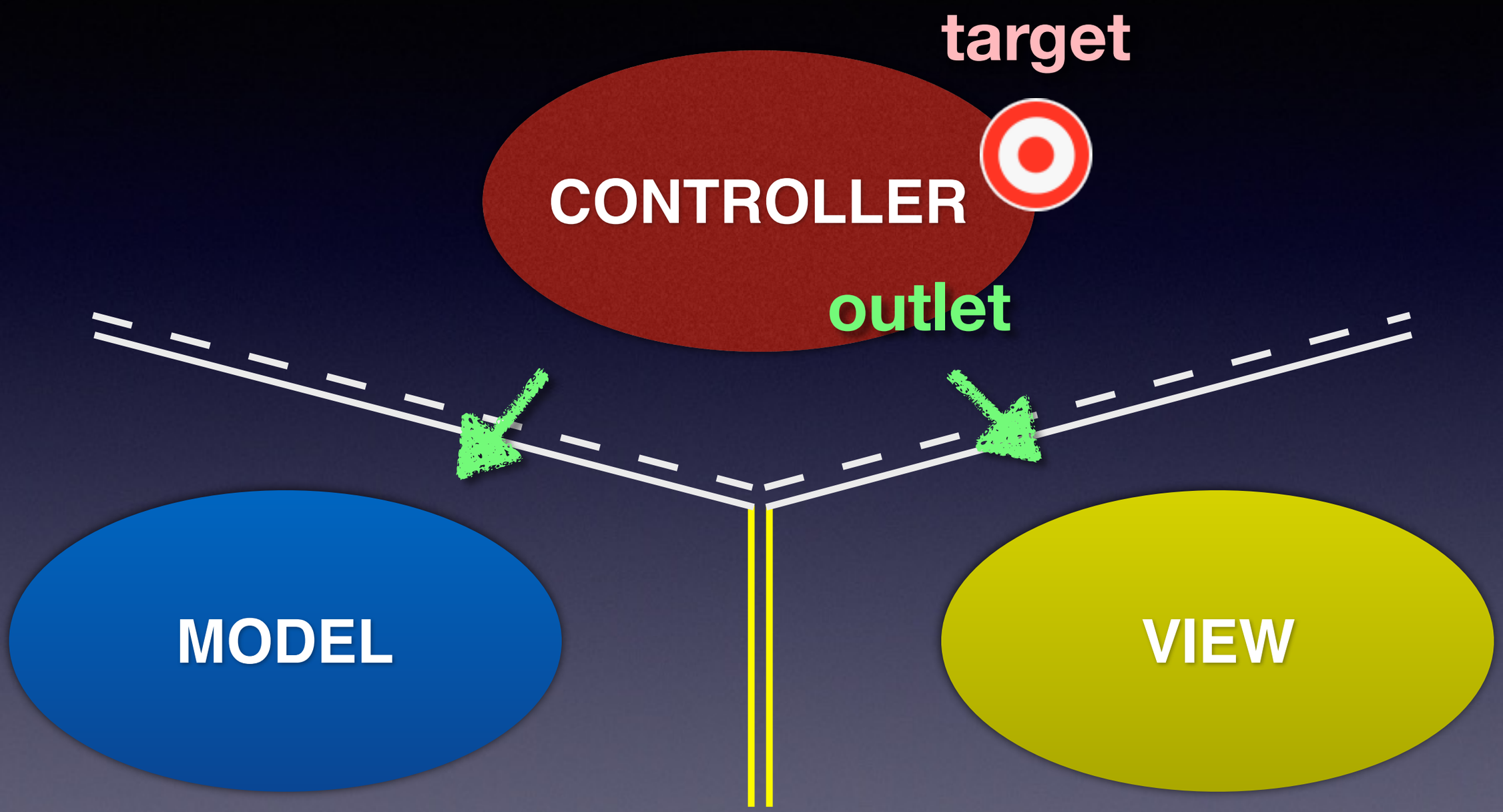
Cand userul apasa un buton (**View**), cum facem sa comunicam cu **controller**-ul pentru ca, mai apoi, el sa vina si se ocupe de "vorbitul" cu partea de **Model**?

MVC



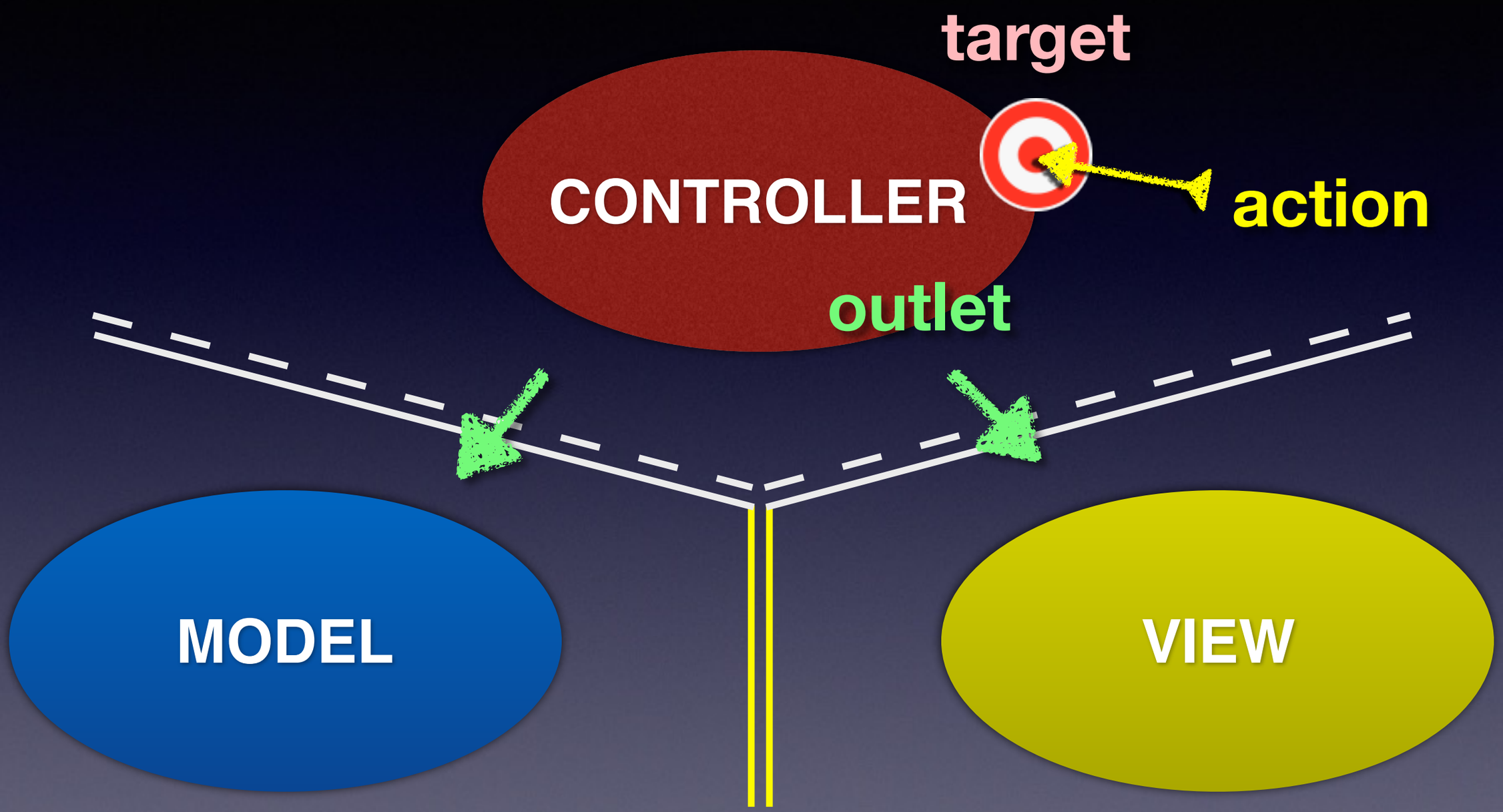
Raspuns: **Controller**-ul poate seta un **target** catre sine

MVC



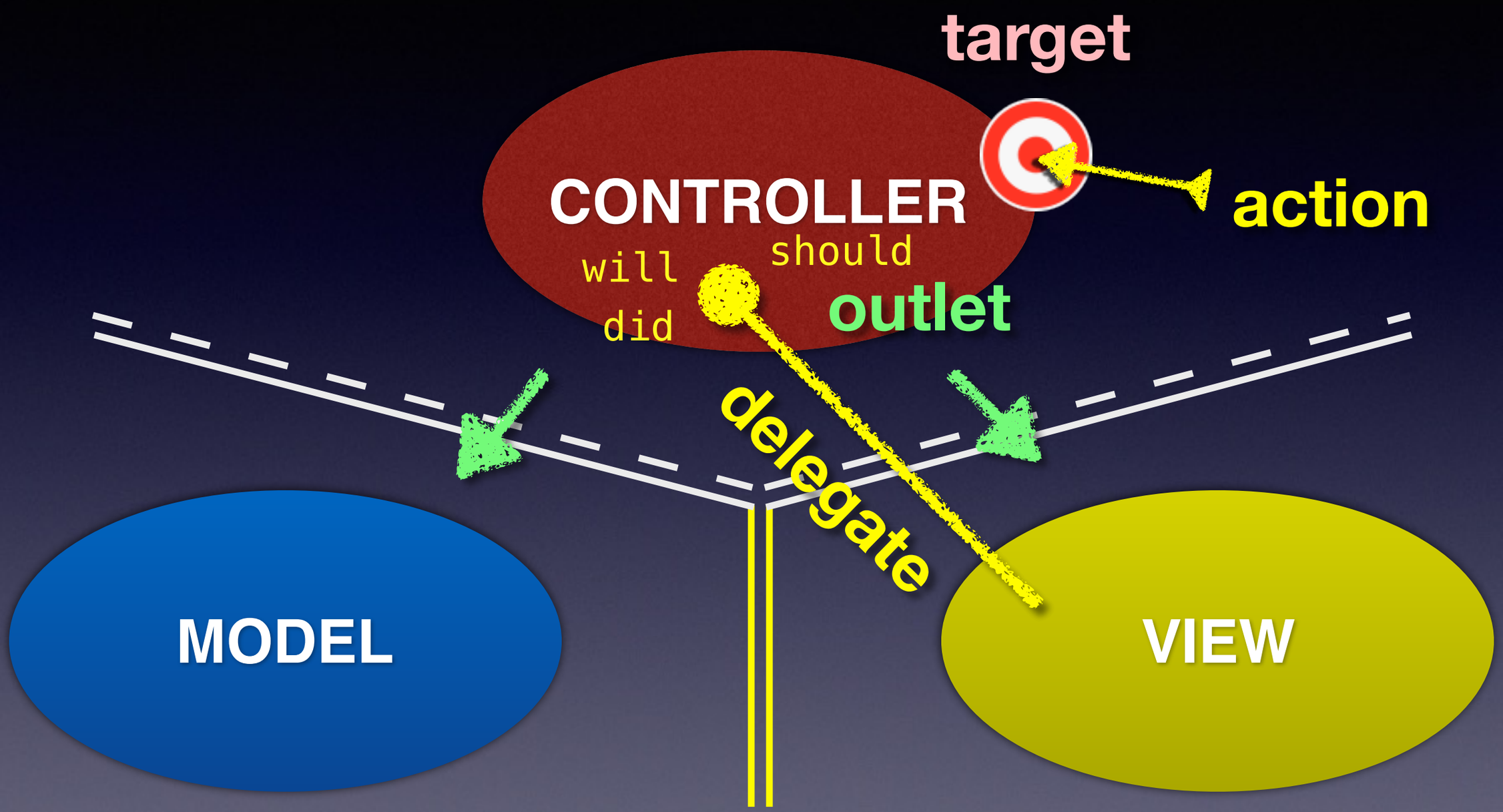
View va comunica mereu cu **Controller**-ul prin **actiuni**
View-ul trimite o **actiune** cand ceva se intampla in UI

MVC



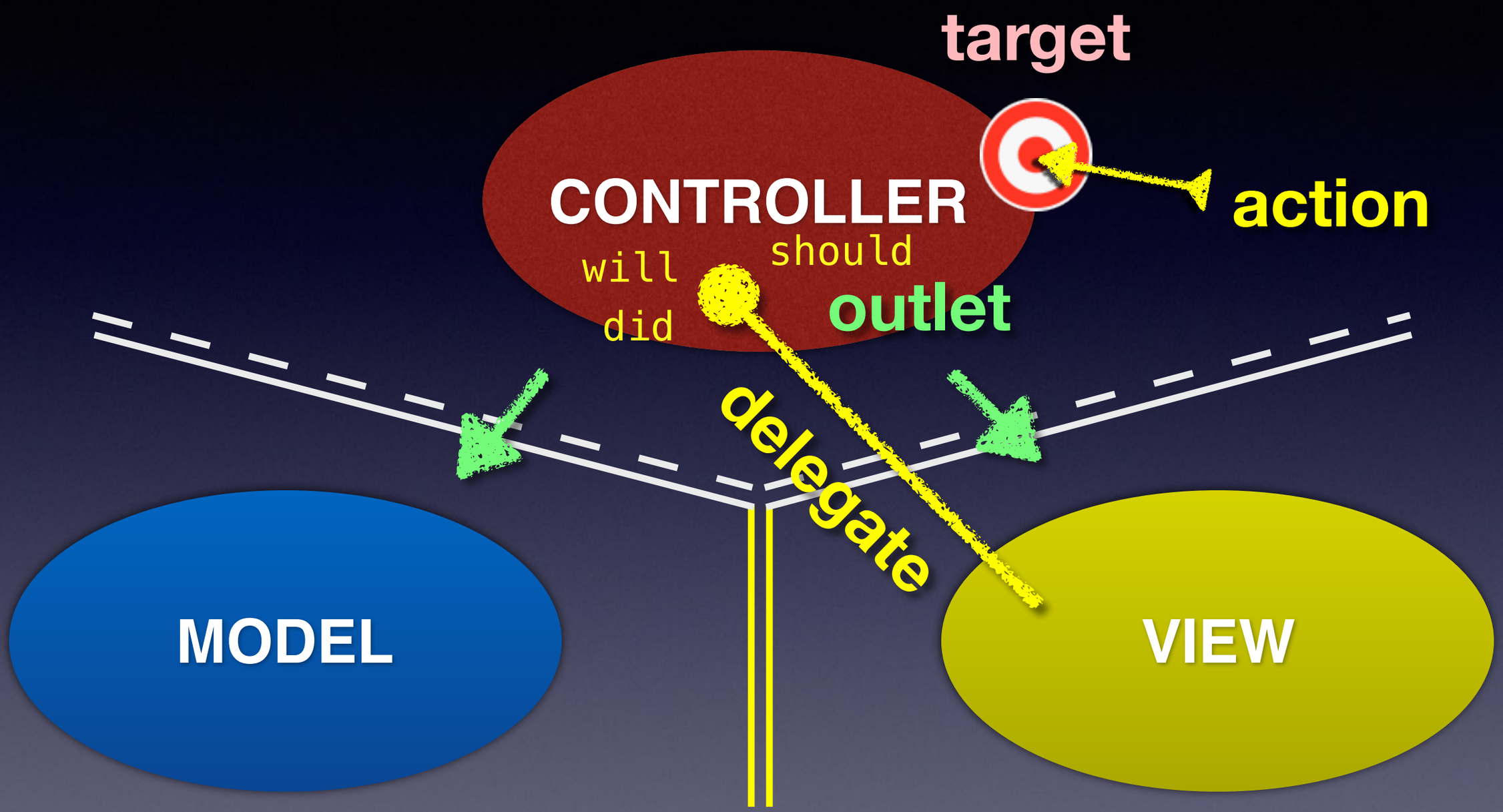
View va comunica mereu cu **Controller**-ul prin **actiuni**
View-ul trimite o **actiune** cand ceva se intampla in UI

MVC



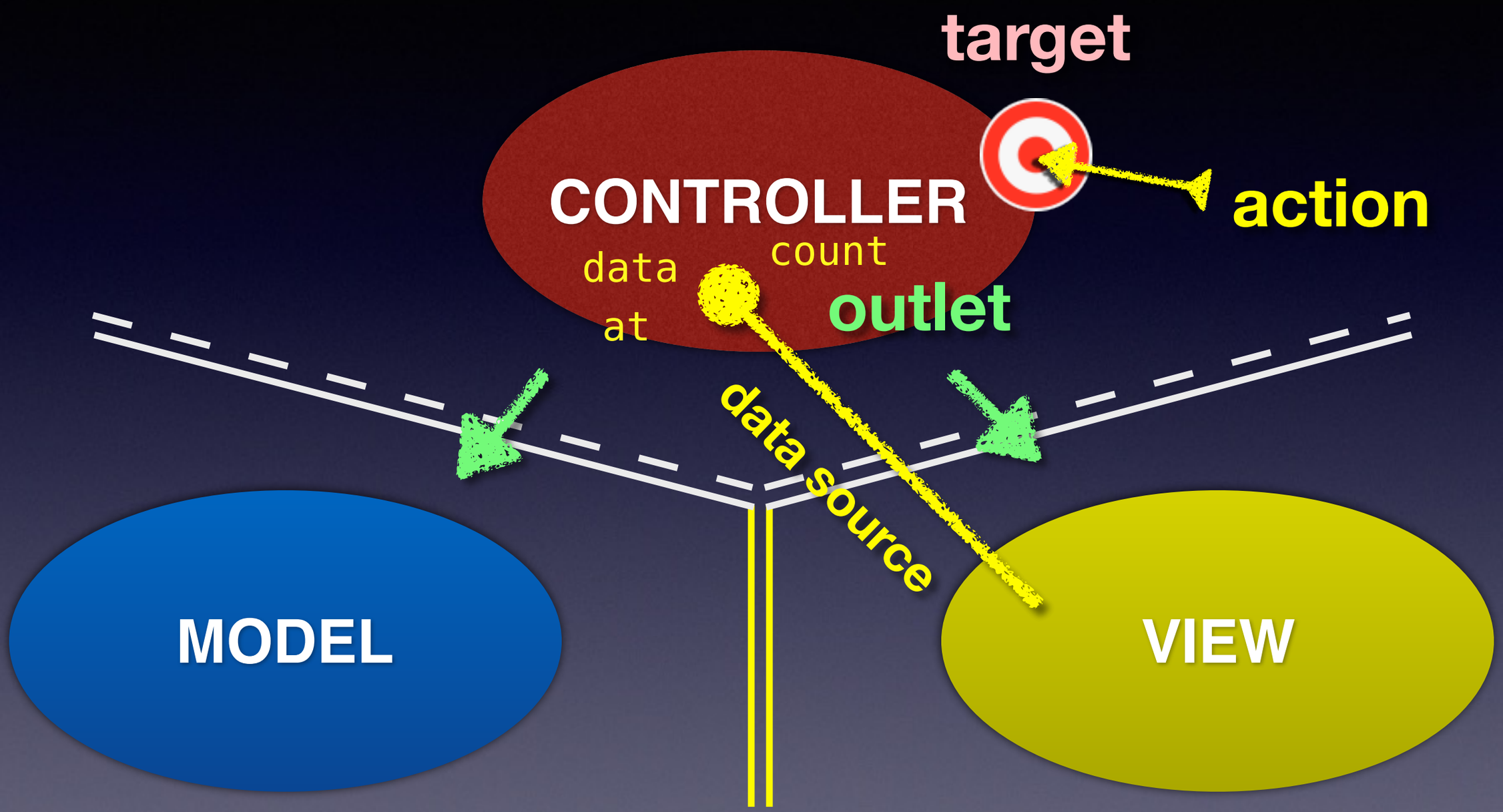
Controller-ul poate sa se seteze pe sine insu-si drept delegat al lui View

MVC



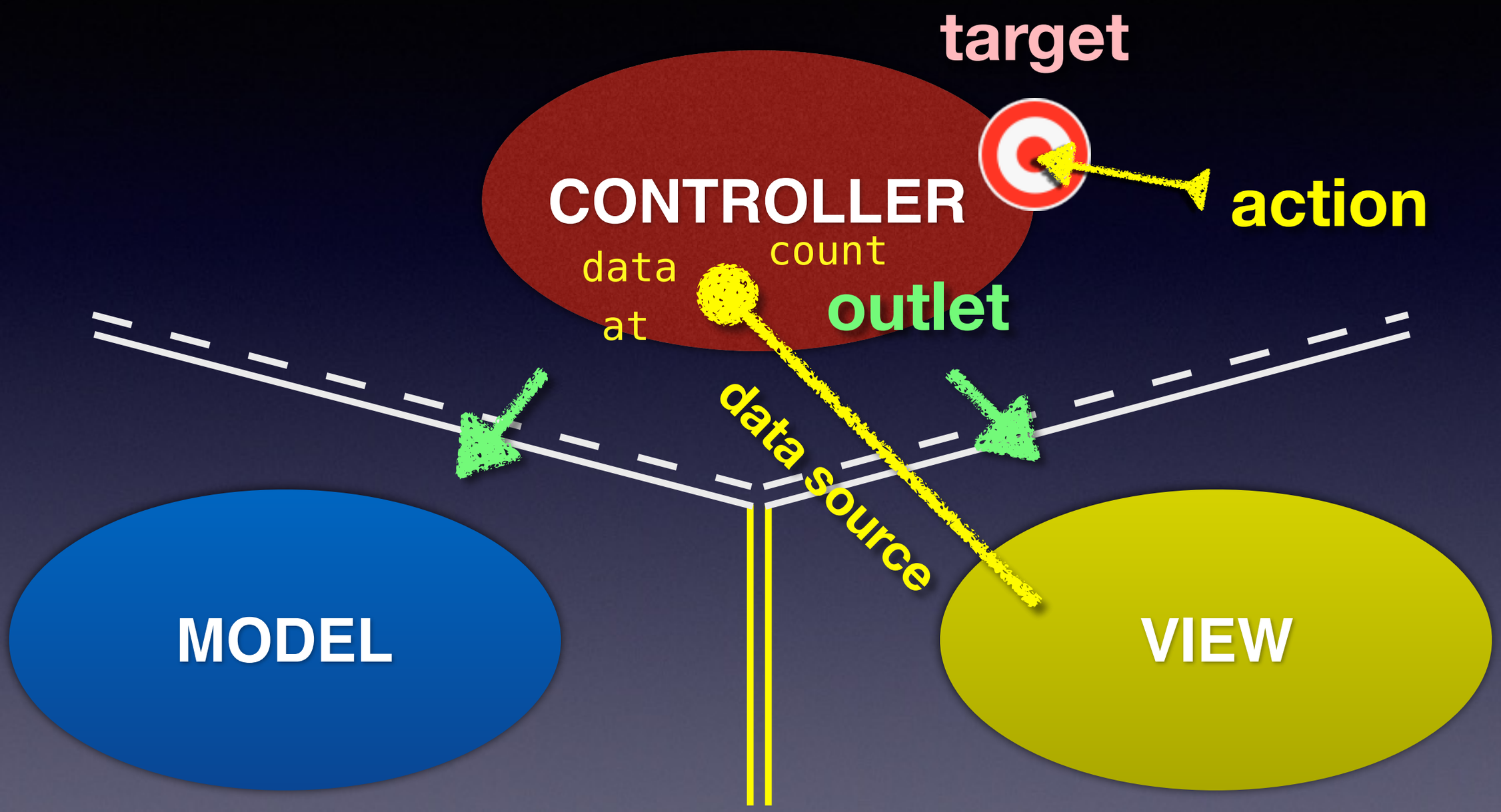
Si sa implementeze functionalitati printr-un protocol, ex:
UITableViewDelegate - tableViewDidEndDisplaying

MVC



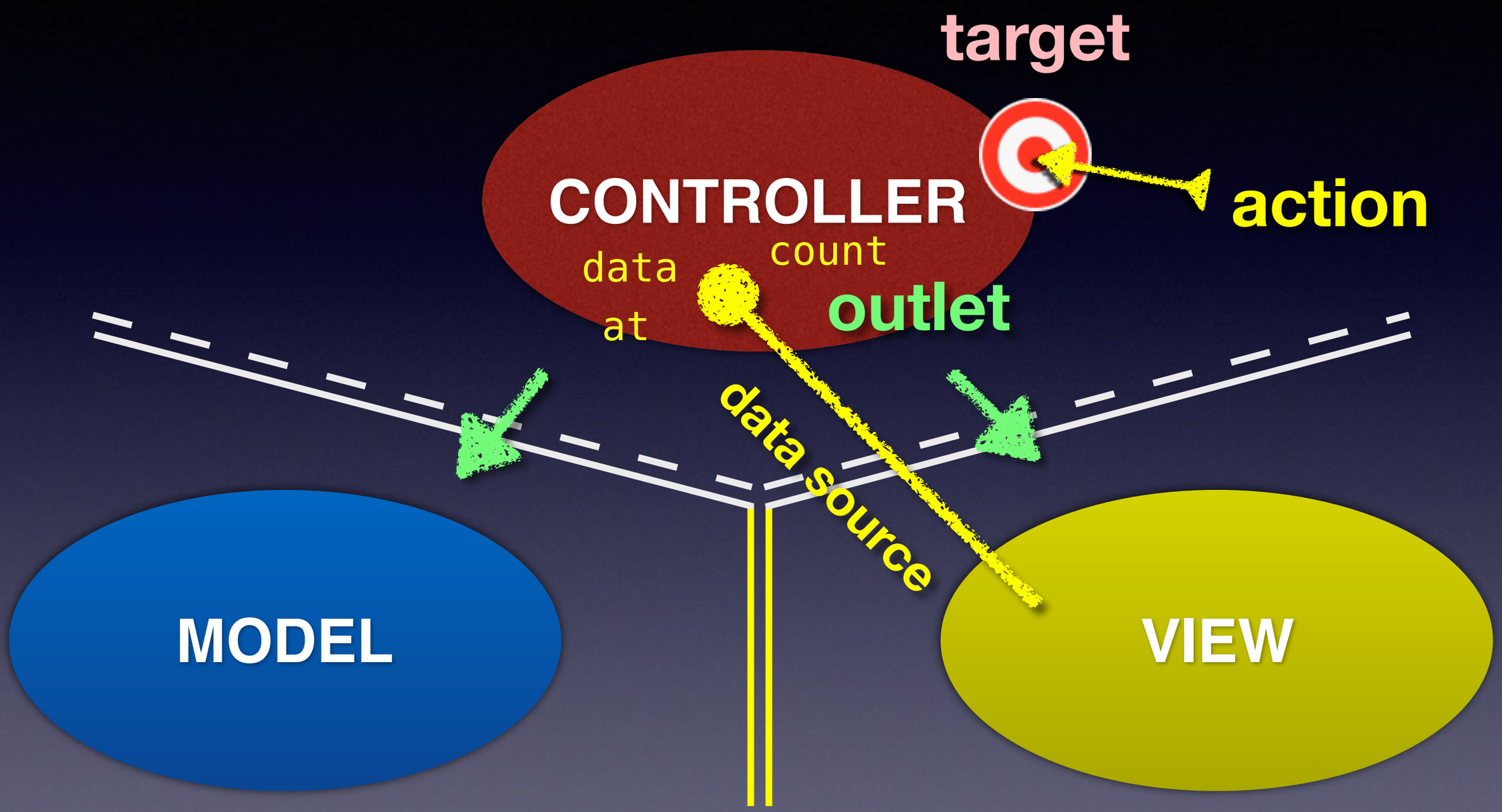
View-ul **NU** "detine" si **NU** are acces direct la datele pe care le afiseaza

MVC



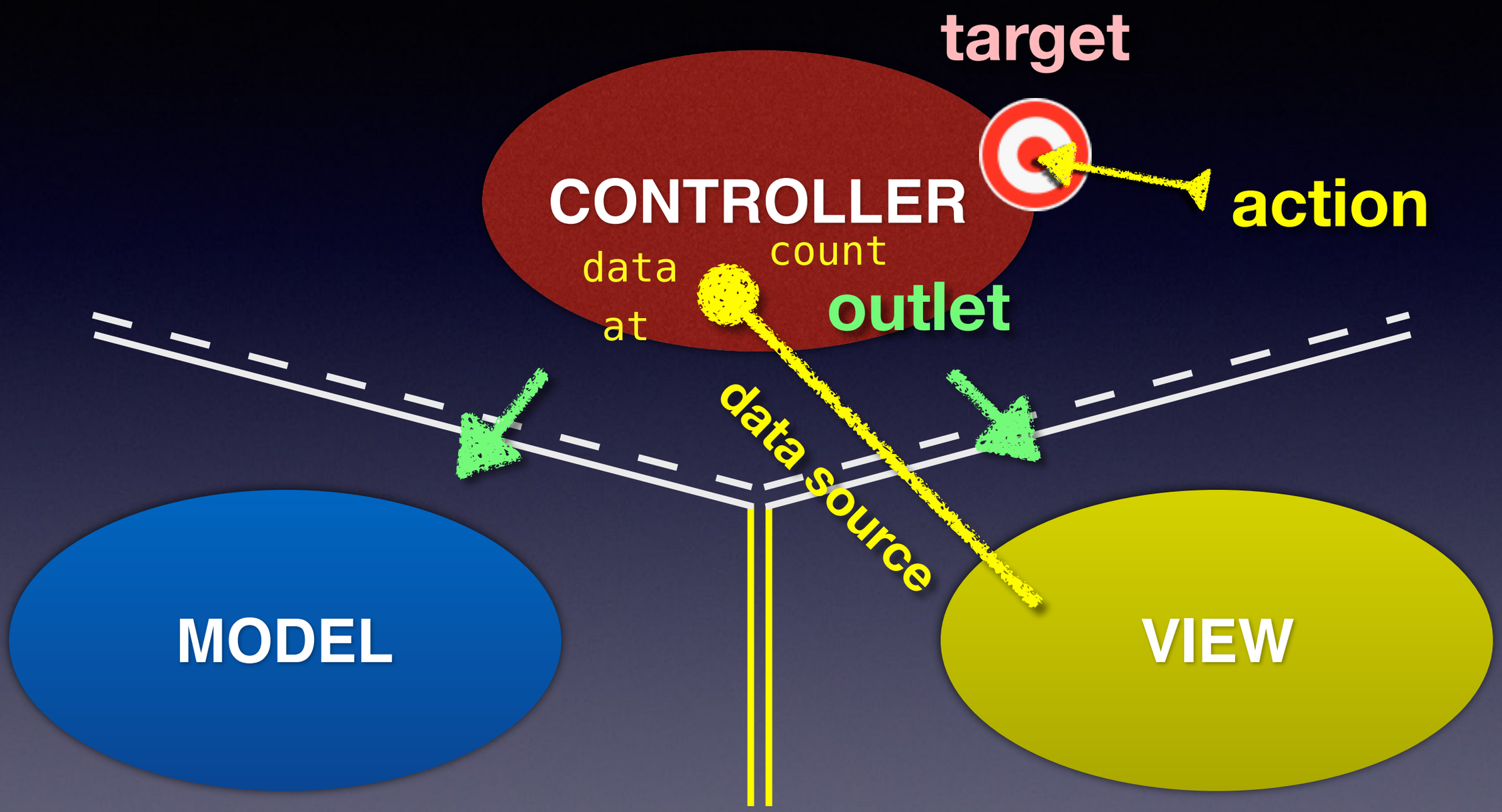
Daca View-ul are nevoie de date atunci vom folosi un protocol. Ex: UITableViewDataSource

MVC



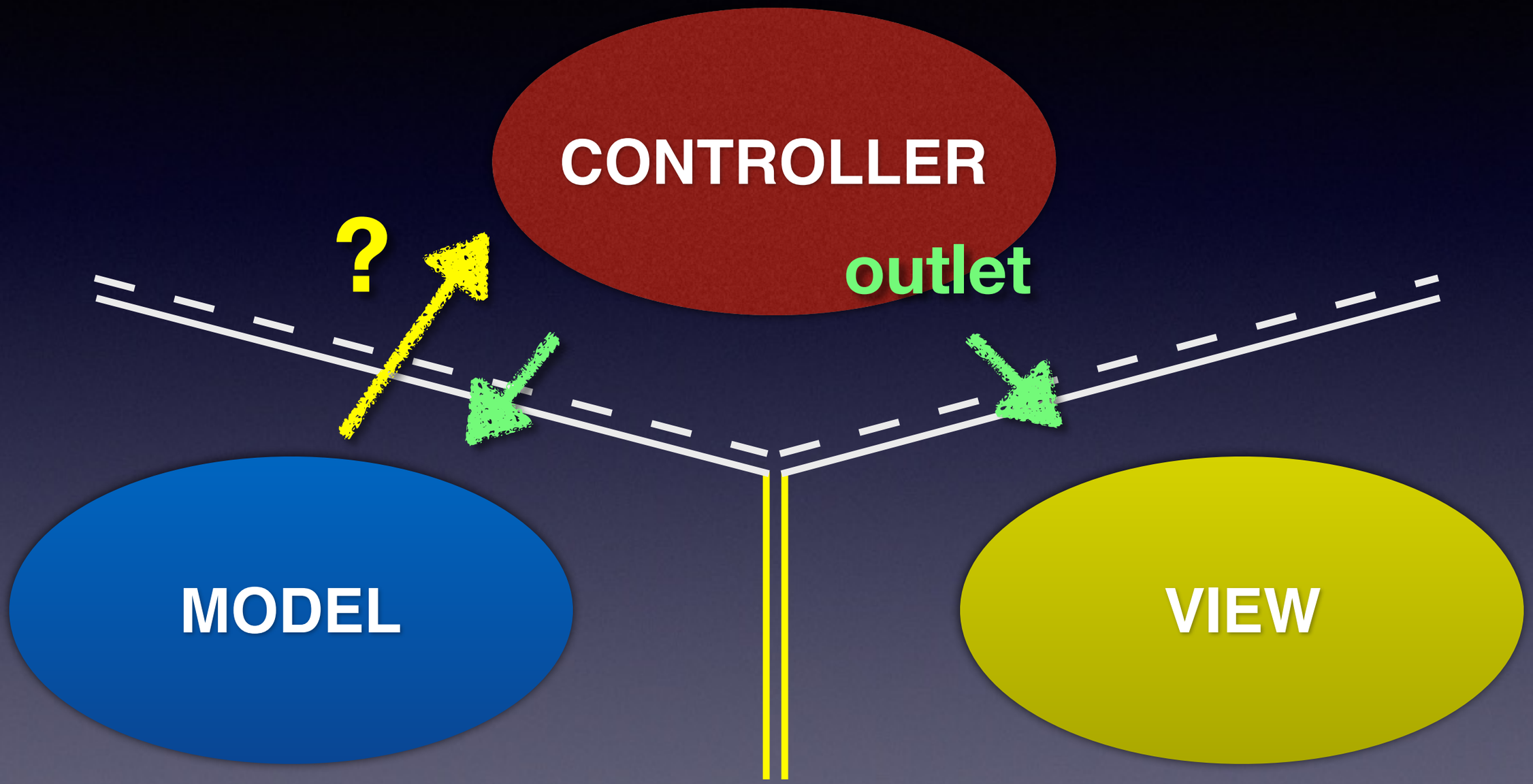
Controller-ul este mereu dataSource pentru **View** (nu **Modelul**)

MVC



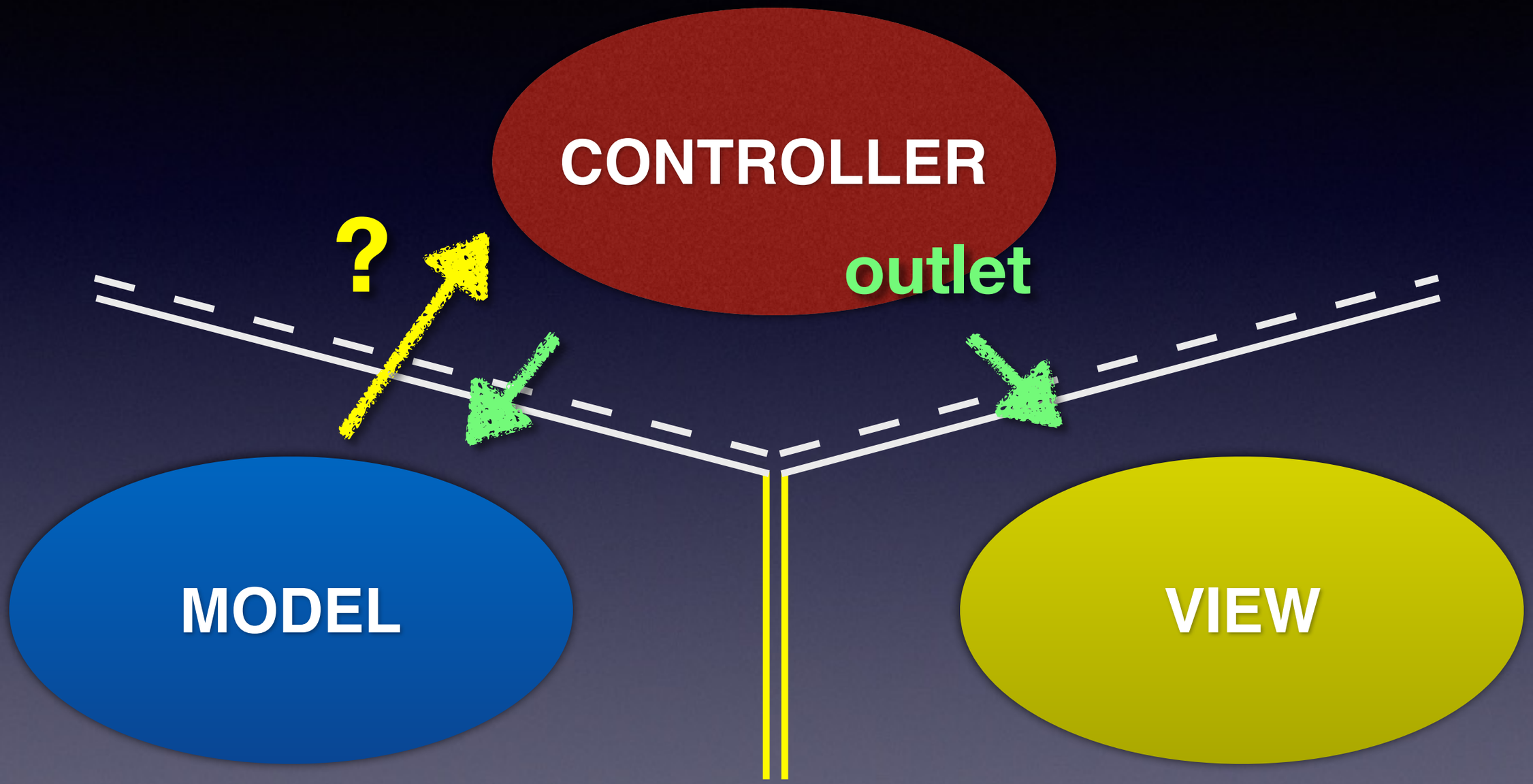
Controller-ul interpreteaza datele de la **Model** si le trimite catre **View**

MVC



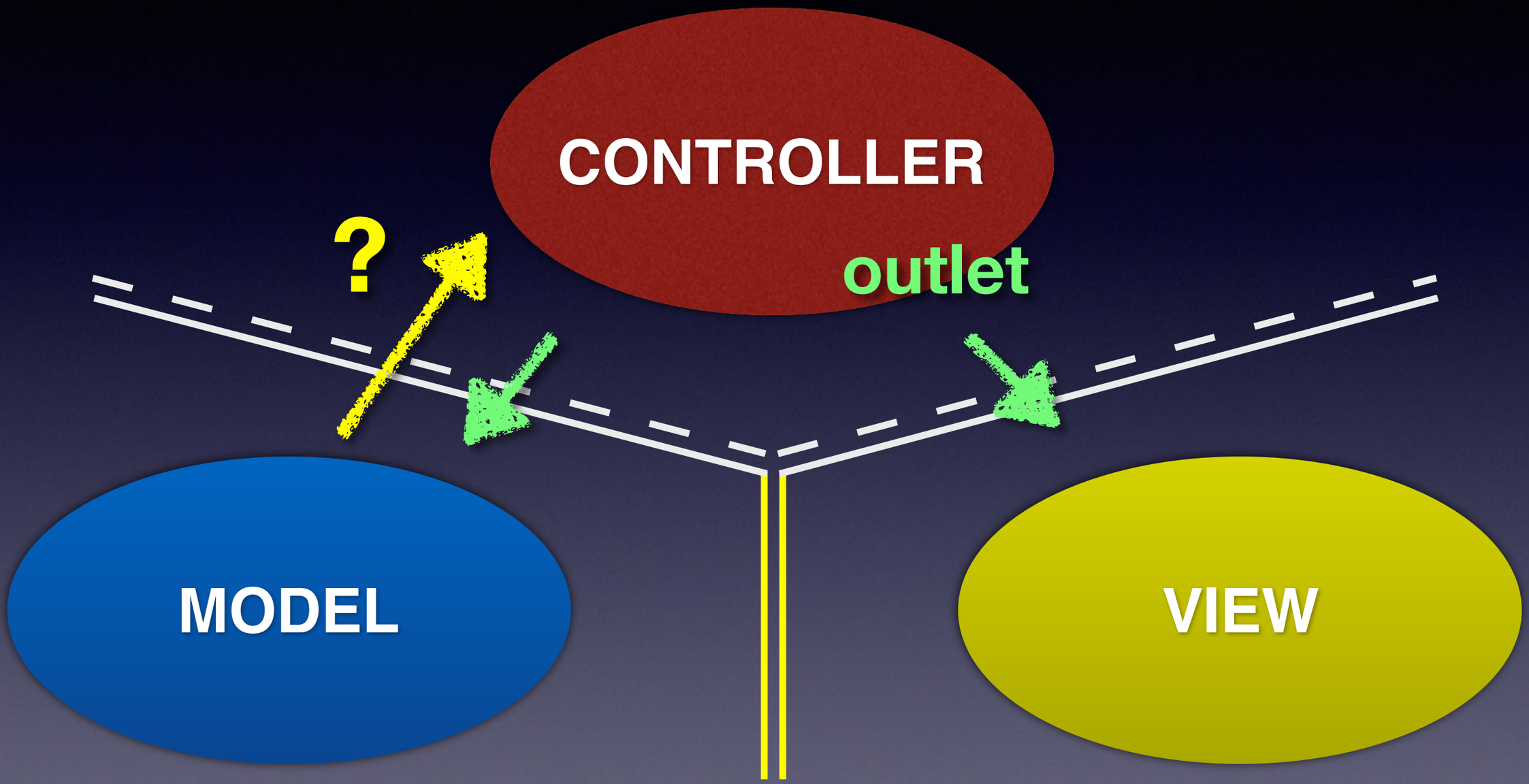
Dar oare **Modelul** poate "vorbi" cu **Controller**-ul ?

MVC



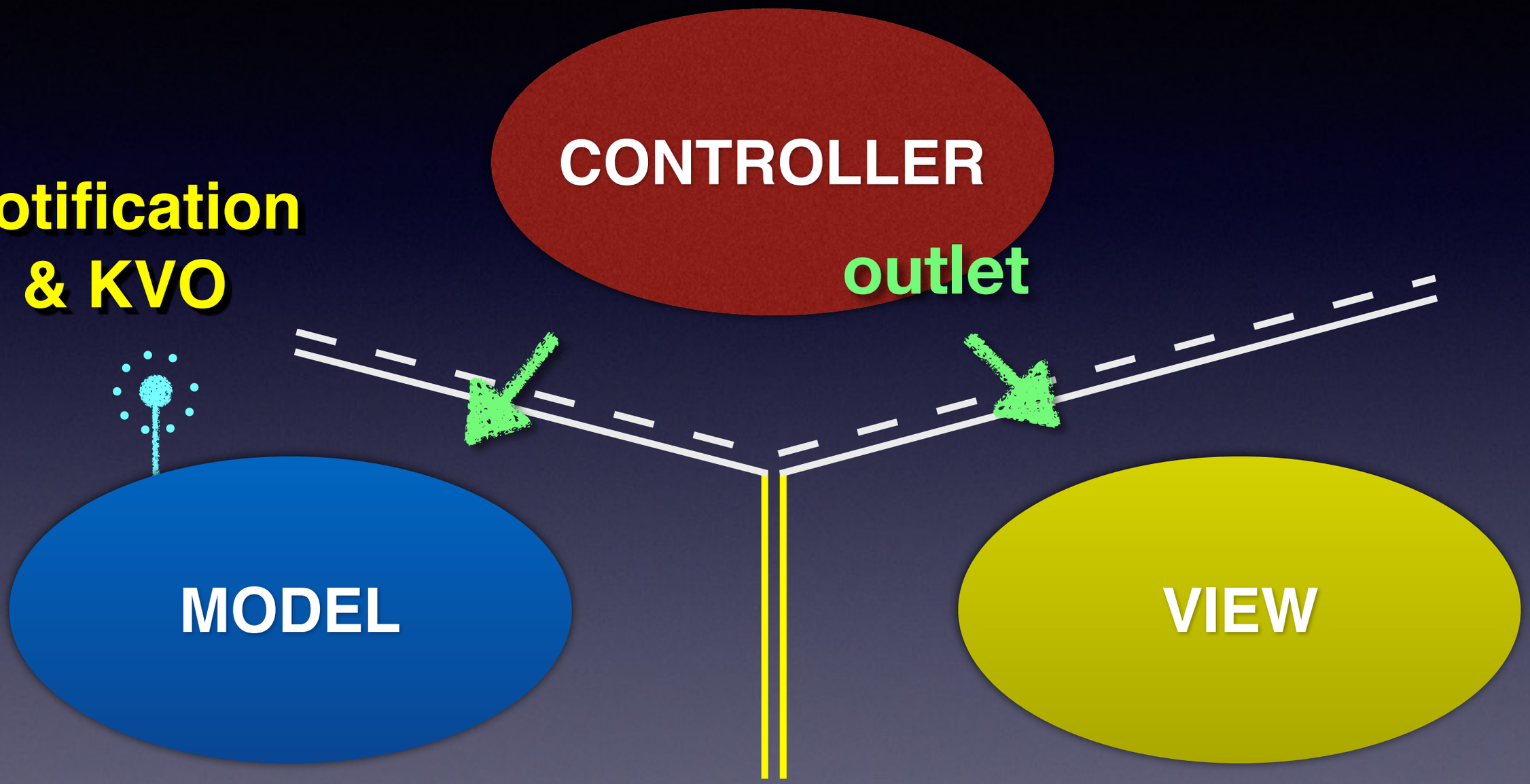
NU! Modelul este independent

MVC



Daca este nevoie ca modificarile datelor din **Model** sa fie "auzite" de **Controller** vom folosi un mecanism asemanator cu o statie radio.

MVC



Controller-ul va "sta" si va "asculta" daca se intampla modificari in **Model**

iOS Course - 5

Error Handling

In Swift putem preveni situatia in care aplicatia noastra da erori folosind do**-catch.** In cazul unei erori aplicatia nu va da crash ci va continua pe rapura catch

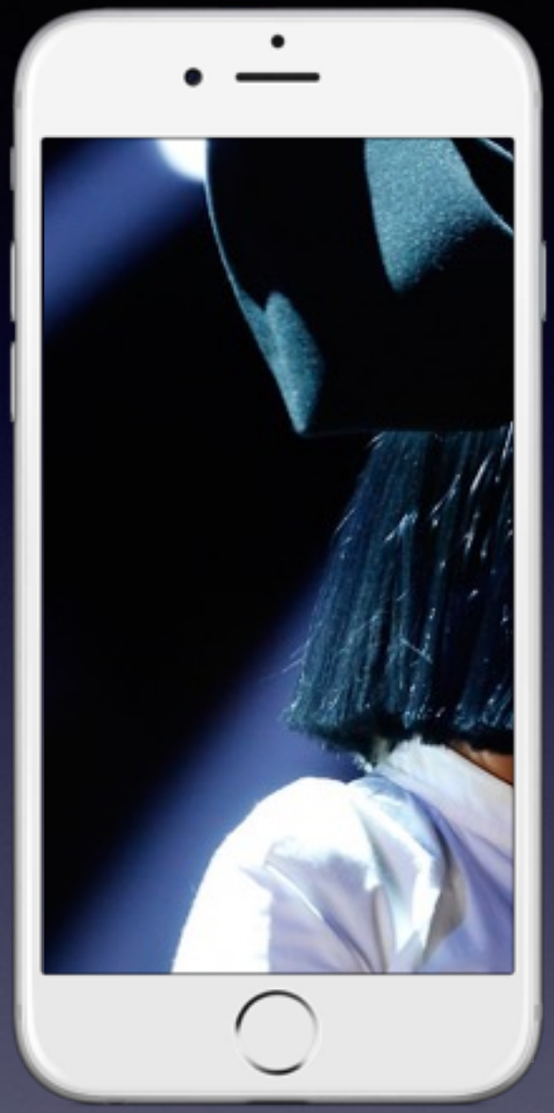
```
do {
      try context.save()
} catch let error {
      print(error)
}
```

Daca nu este nevoie de ramura catch si nu ai nevoie sa folosesti eroarea, atunci poti folosi doar try in fata instructiunilor, astfel:

```
try! context.save() //crash in caz ca da eroare
try? context.save() //ignora erorile si treci mai departe
```

Sia
WWDC 2016


Sia
WWDC 2016




Sia
WWDC 2016

Sia

WWDC 2016


Sia
WDC 2016

UIScrollView

Metode si Atribute:

- **contentSize**: CGFloat **//manipulat prin metoda *setContentSize***

Poate fi mai mare ca ecranul. Trebuie setat pentru a se putea face scroll

- **minimumZoomScale**: CGFloat, >= 0
- **maximumZoomScale**: CGFloat

Ambele trebuie setate pentru a se putea face zoom in / out

- **zoomScale**: CGFloat, contine zoomScale-ul curent
- **setZoomScale(scale: CGFloat, animated: true)**

UIScrollViewDelegate, va fi nevoie de un delegat care implementeaza acest protocol si care pune in actiune Zoom-ul prin metoda:

- **viewForZooming (in scrollView: UIScrollView) -> UIView?**

Demo

Aplicatie noua: ***Cassini***

Multithreading

Multithreading-ul se bazeaza in mare parte pe Cozi. O functie, in genere un closure, este plasat intr-o coada si executat si are un thread asociat lui.

Main Queue vs Global Queue:

- Main se ocupa de lucruri precum UI
- Global se ocupa de lucrul cu datele, nu UI
- Incarcarea sau modificarea datelor mari sau cantitatilor mari de date in Main Queue poate duce la blocarea aplicatiei, deoarece aici se executa si actiunile din UI.
- In Global Queue nu se executa nimic din zona de UI, astfel incarcarea datelor nu va bloca aplicatia. Daca este nevoie de modificarea UI-ului la finalizarea unei operatii in global queue vom folosi un closure separat, adaugat in Main Queue

Multithreading

```
DispatchQueue.main.{async/sync} {
    //do something with the UI
}
```

Global Queue:

```
DispatchQueue.global(qos: DispatchQoS).{async/sync} {
    //do something with your data
}

DispatchQoS.userInteractive // prioritate mare - task-uri scurte
DispatchQoS.userInitiated // prioritate mare - task-uri ceva mai
lungi
DispatchQoS.background // poate rula cat de incet este nevoie
DispatchQoS.utility // prioritate mica - task-uri foarte lungi
```

UIGestureRecognizer

Prin intermediul lui UIGestureRecognizer putem seta target-ul si actiunea unui View in momentul in care utilizatorul efectueaza un gest anume, spre exemplu long press sau tap.

Generic: **UIGestureRecognizer(target: Any?, action: Selector?)**
LongPress:

- **let longPress = UILongPressGestureRecognizer(target: self, action: #selector(myMethod))**

Tap:

- **let doubleTap = UITapGestureRecognizer(target: self, action: #selector(myMethod))**
- **doubleTap.numberOfTapsRequired = 2 // double tap**

```
myView.addGestureRecognizer(doubleTap)
myView.addGestureRecognizer(longPress)
```

Demo

Cassini

Proiect Xcode:

https://github.com/mariusjcb/iOS10_CS5