App-Entwicklung für iOS und OS X

Aktualisiert für iOS 10, Xcode 8 und Swift 3

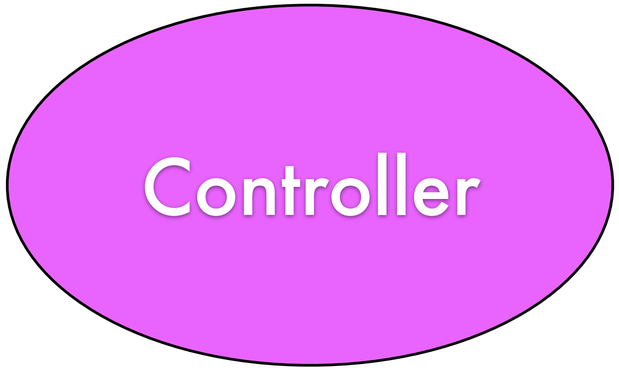
SS 2017

Stephan Gimbel

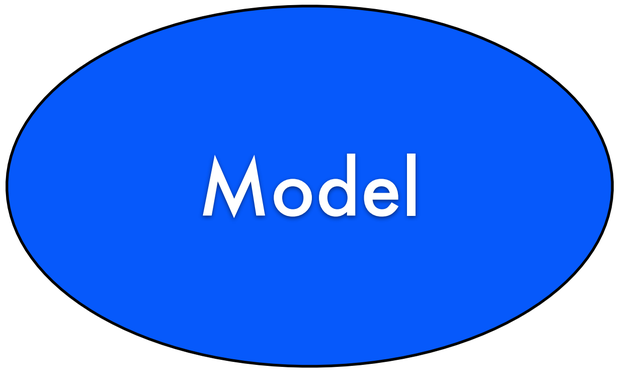
Heute

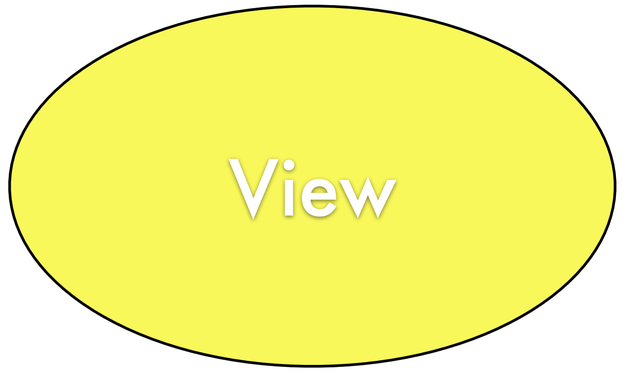
- MVC
  - Objekt-orientiertes Design Konzept
- Fortsetzung der Demo von letzter Woche
  - Computed Properties, MVC, UI für unterschiedliche Devices

Model View Controller



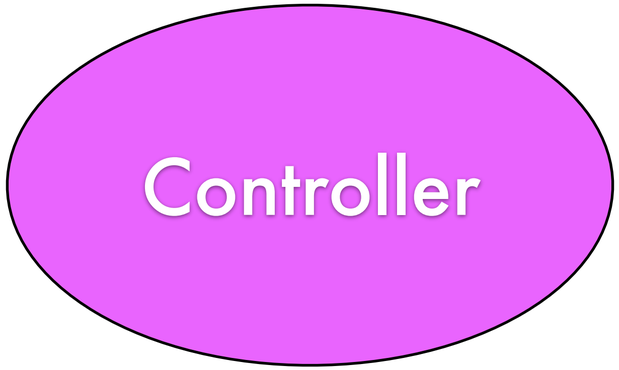
One Pattern to
Rule Them All



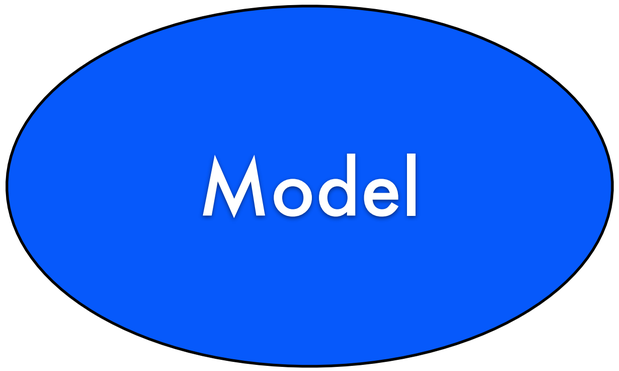


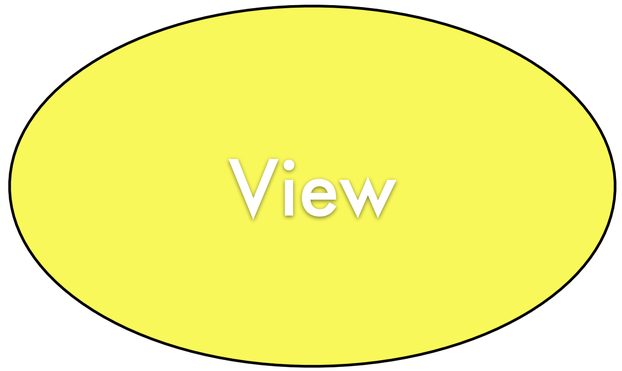
Definition von drei verschiedenen Rollen und wie diese miteinander kommunizieren

Model View Controller



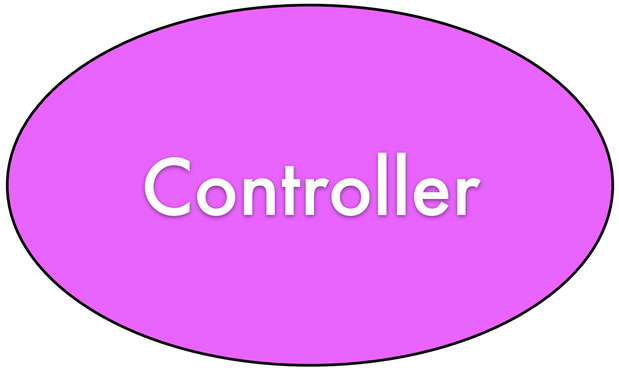
One Pattern to
Rule Them All



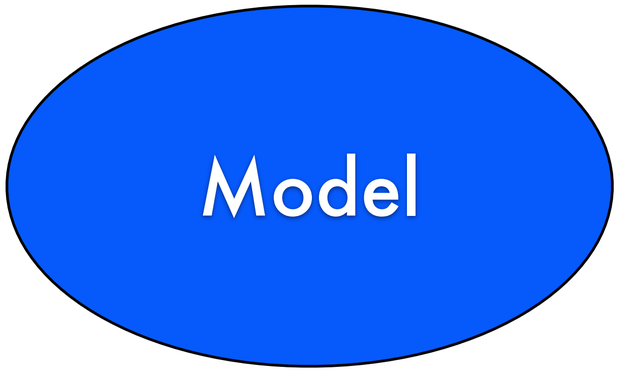


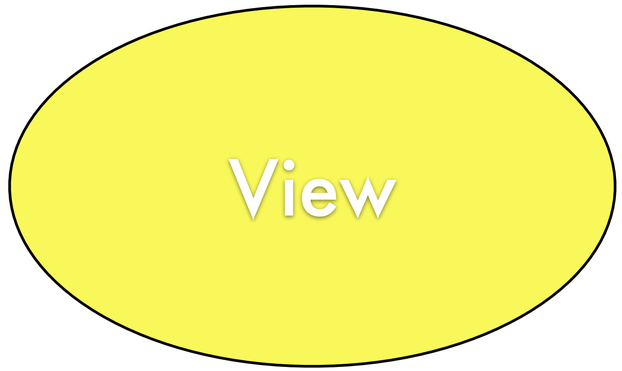
Model: Gibt an, was die App ist (aber nicht wie diese an gezeigt werden)

Model View Controller



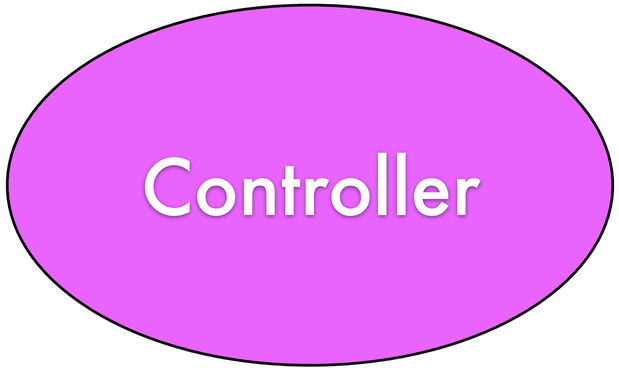
One Pattern to
Rule Them All



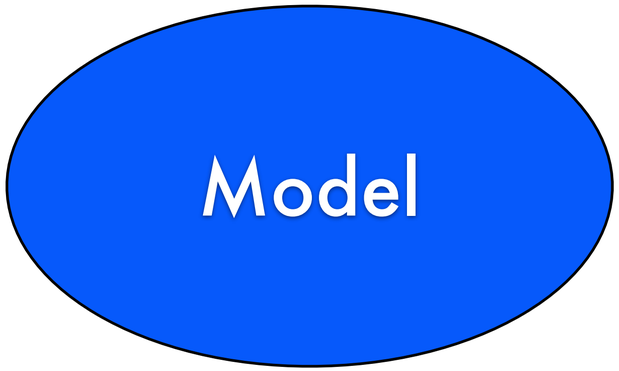


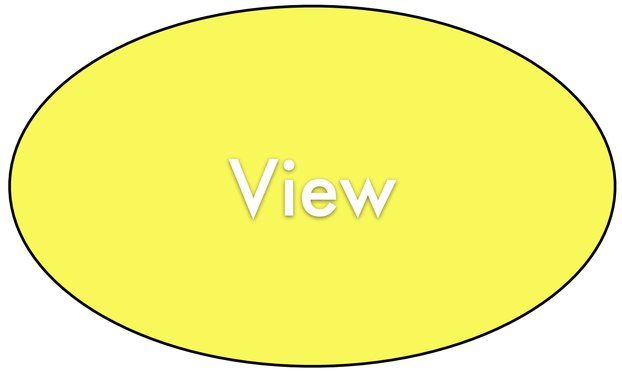
Controller: Wie das Model dem User präsentiert wird (UI Logik)

Model View Controller



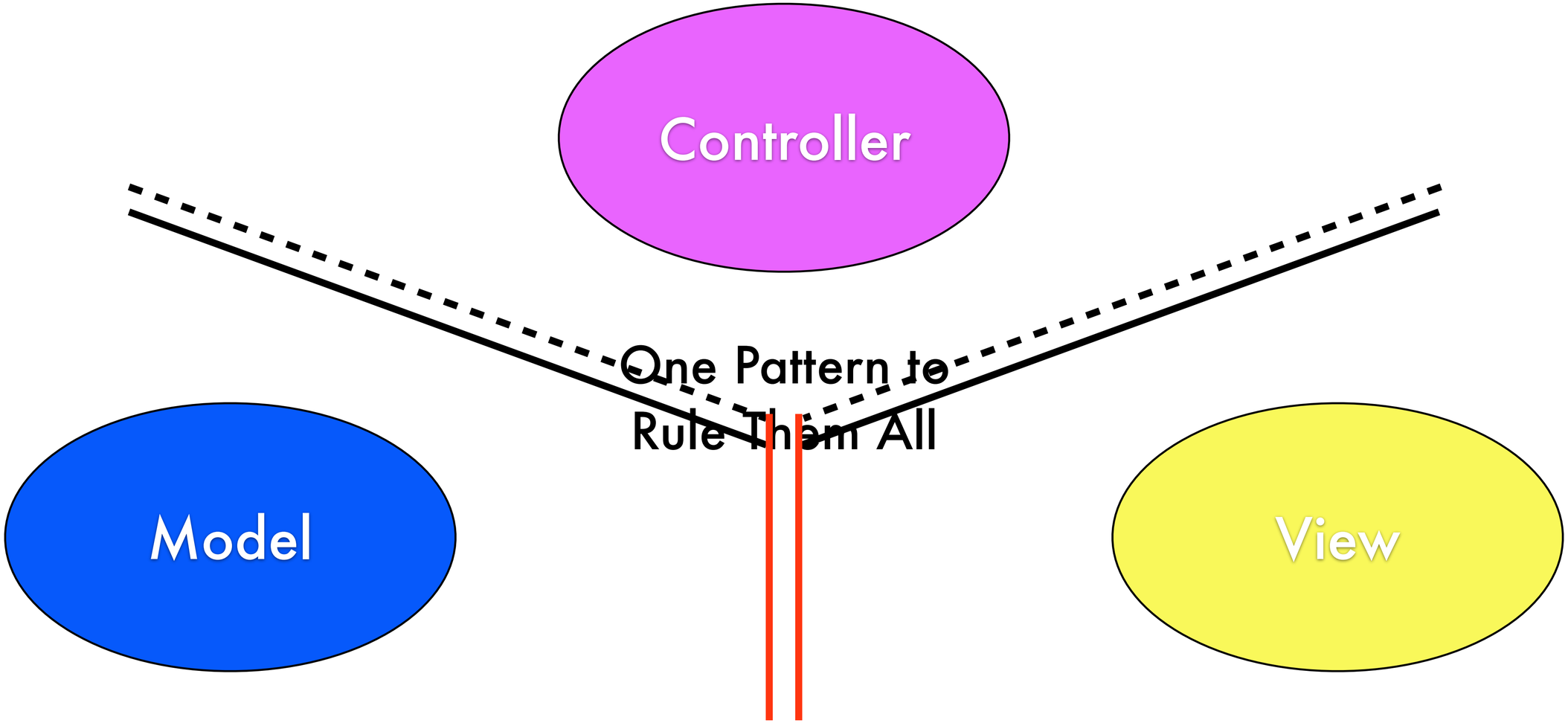
One Pattern to
Rule Them All





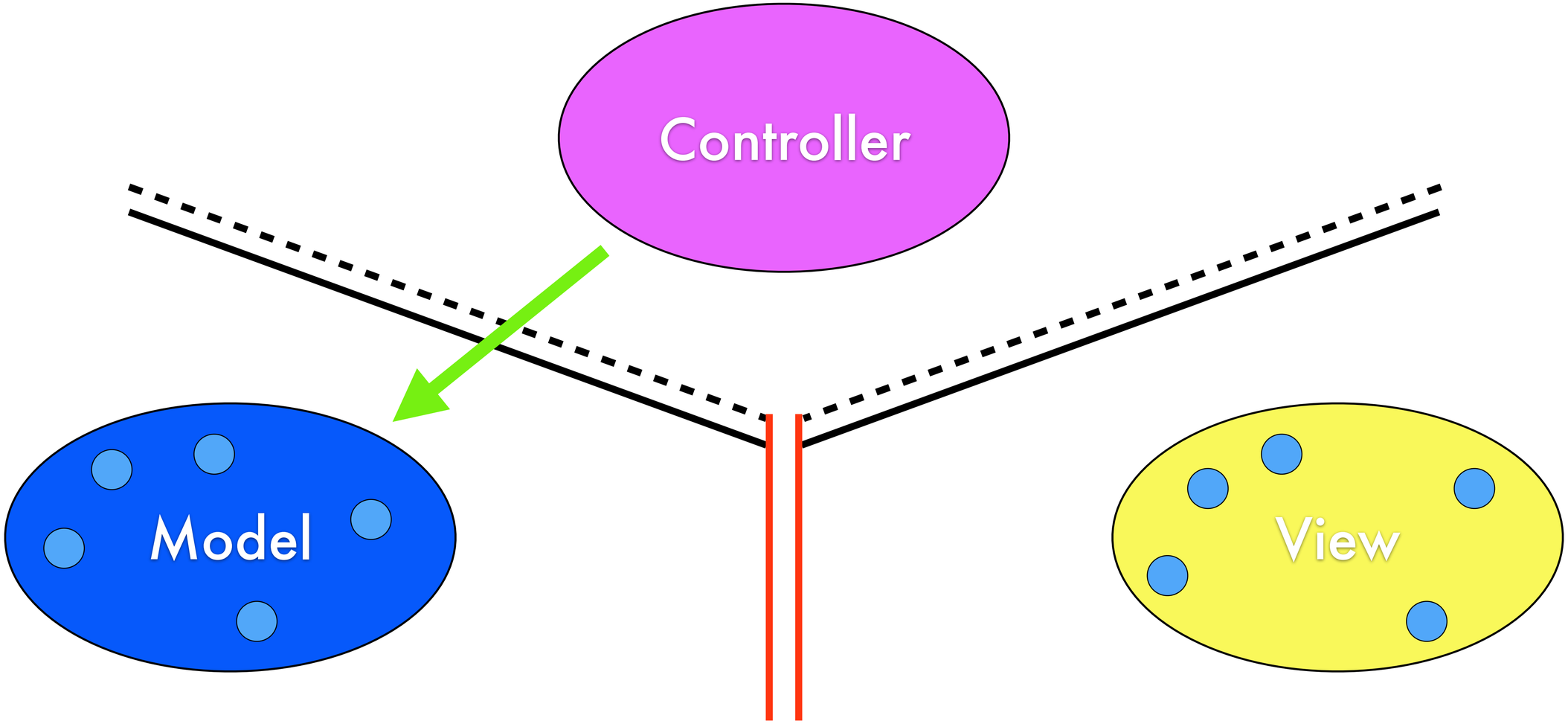
View: das, was der User sehen und mit dem er interagieren kann: Buttons, Text-Felder, Grafiken, usw.
(View weiß wie er gezeichnet wird und auf User-Action reagiert)

Model View Controller



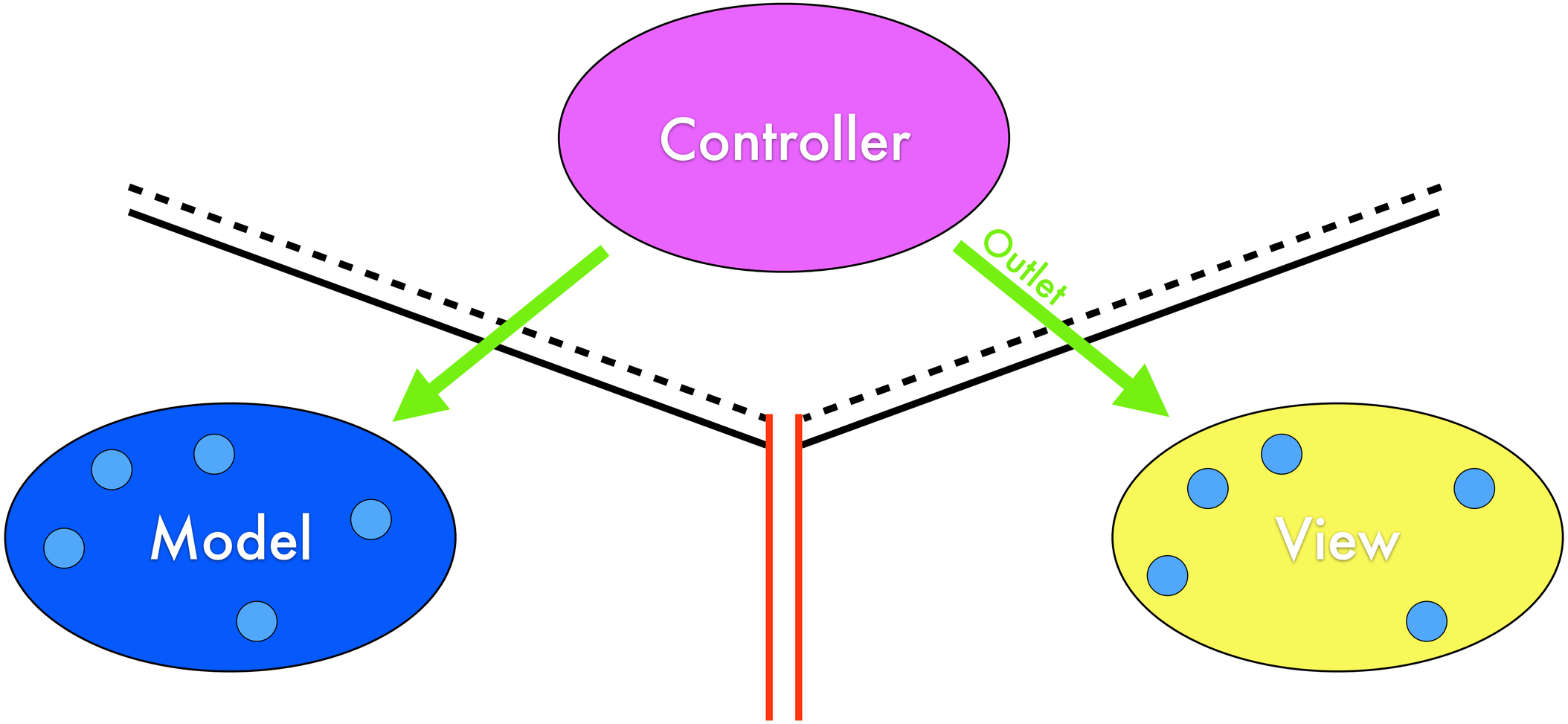
Das MVC Pattern managed die Kommunikation zwischen diesen Rollen

Model View Controller



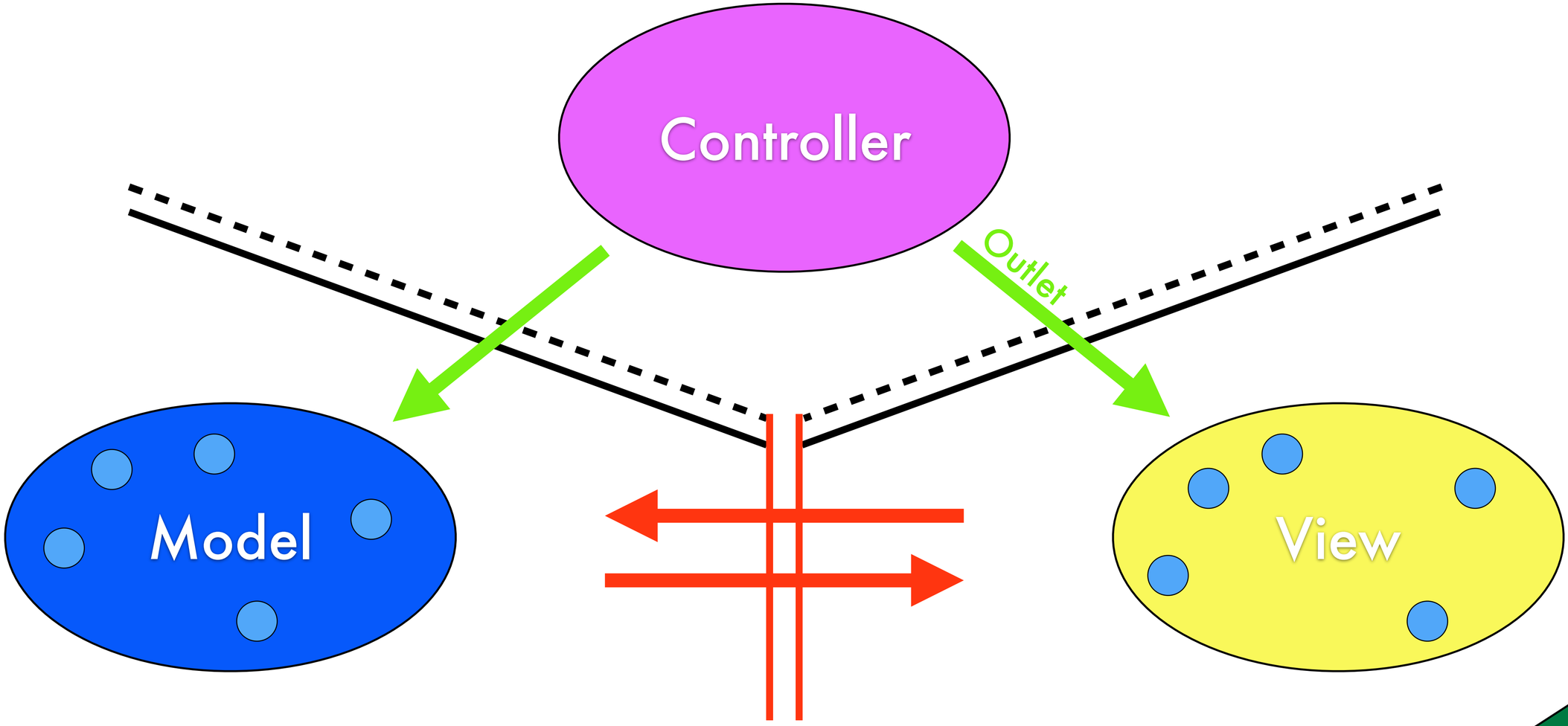
Controller können immer direkt mit Ihrem Model kommunizieren

Model View Controller



Controller können auch direkt mit ihrem View kommunizieren

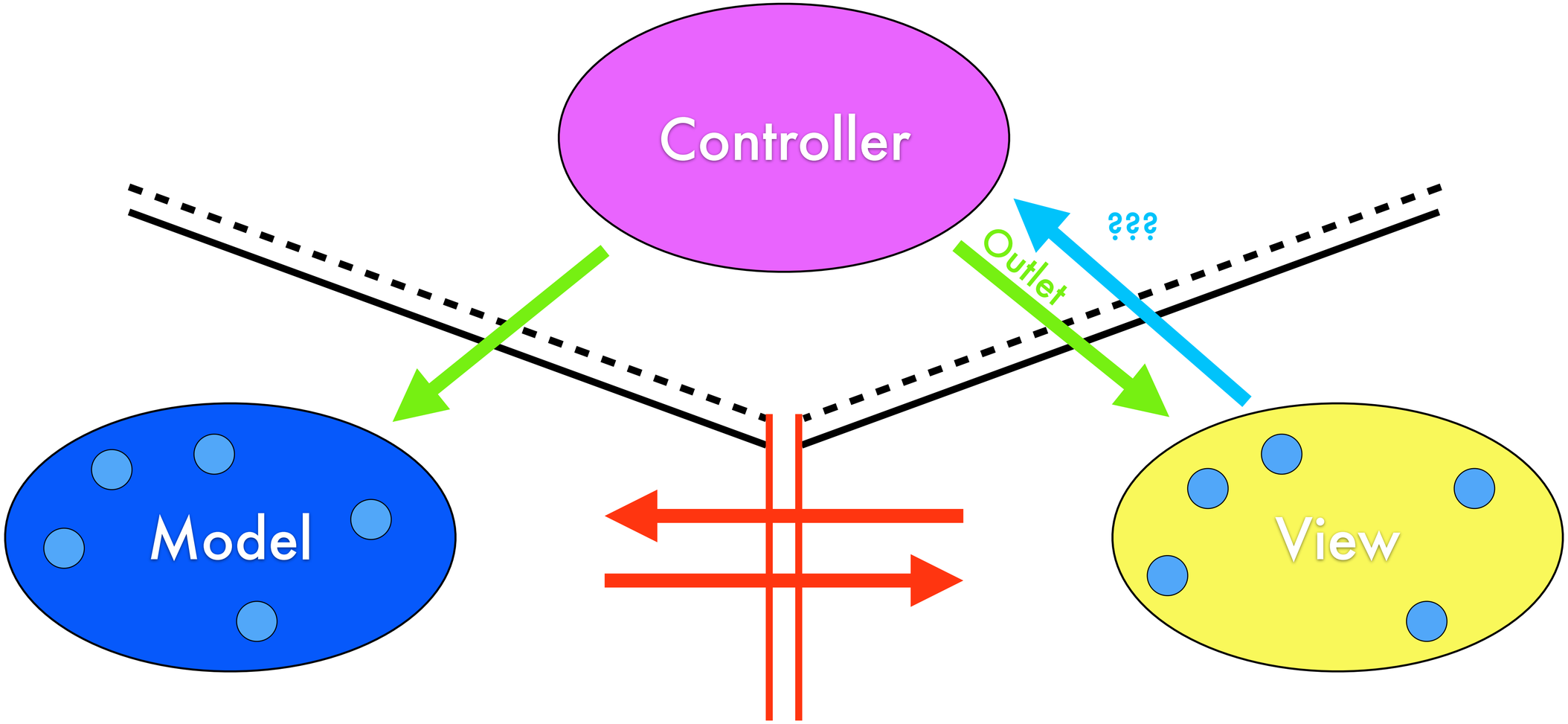
Model View Controller



Controller und View kommunizieren niemals miteinander!

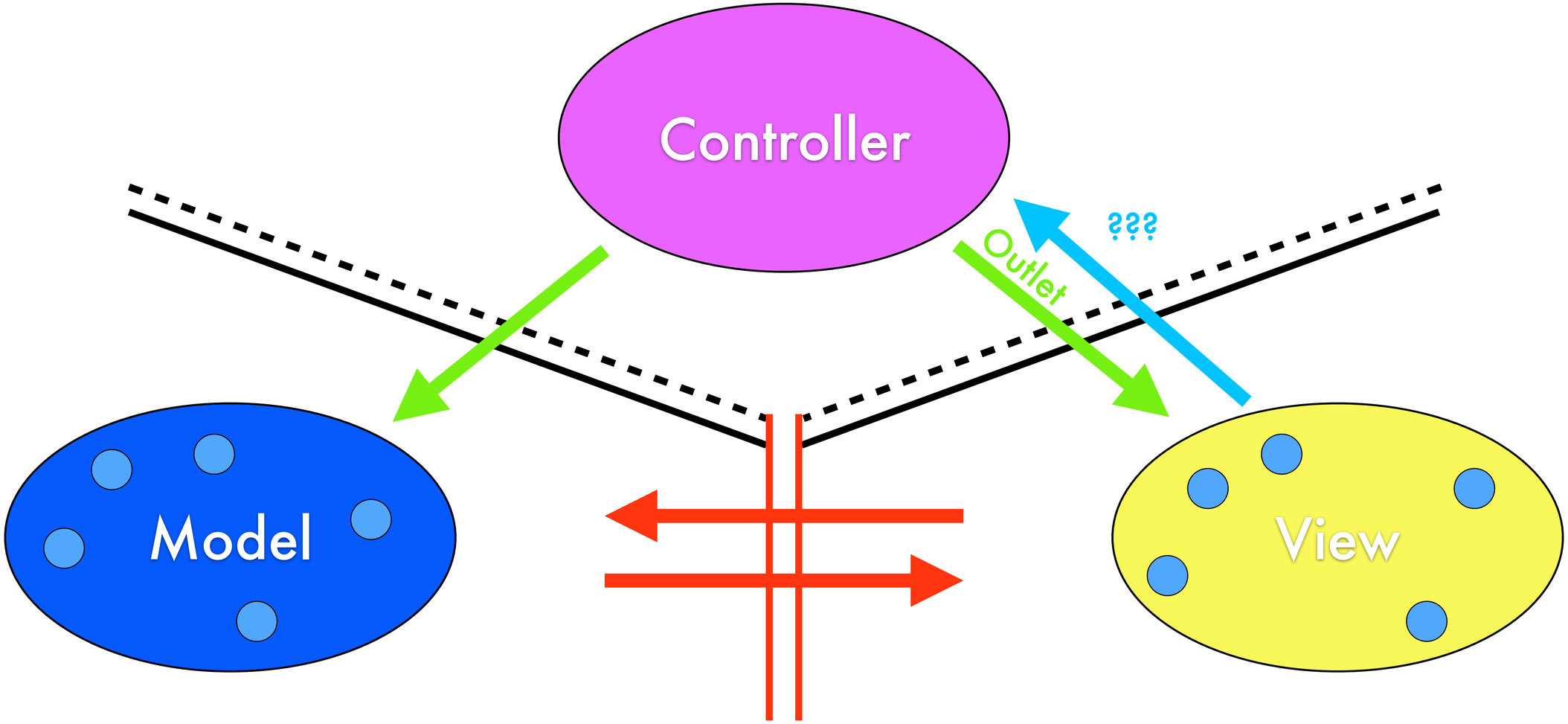
Weshalb?

Model View Controller



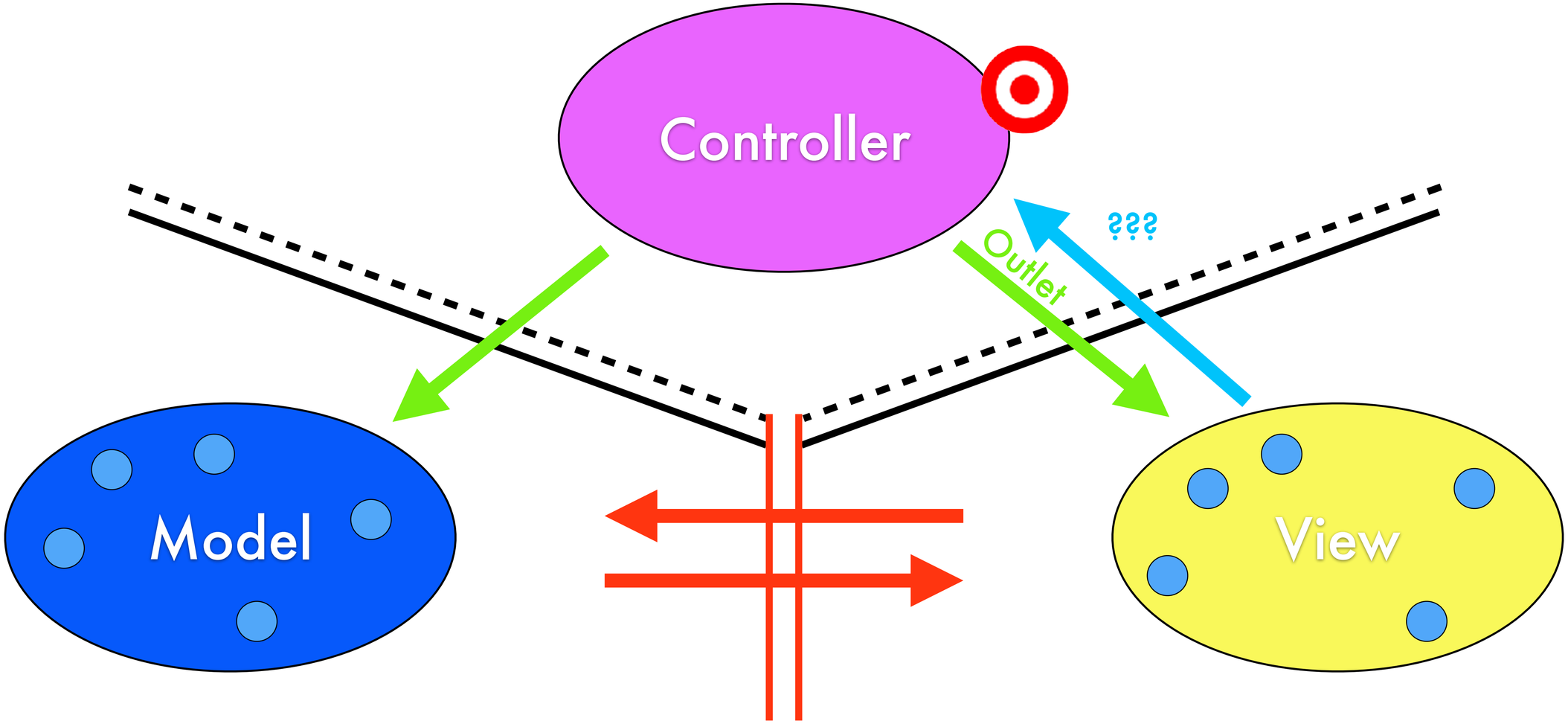
Kann der View mit dem Controller kommunizieren?

Model View Controller



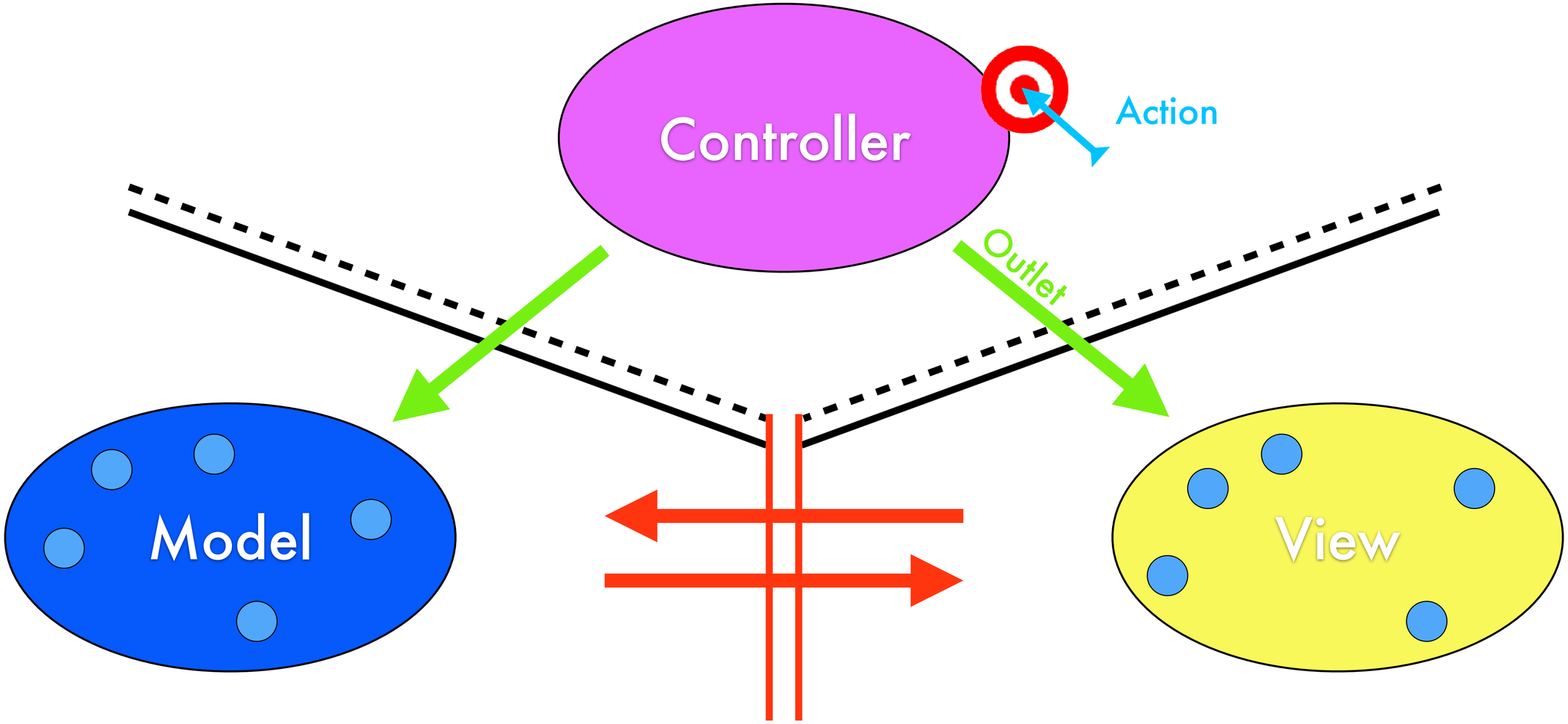
Ja, irgendwie - die Kommunikation ist aber “blind” und strukturiert…
(der View kennt die konkrete Implementierung des Controllers nicht)

Model View Controller



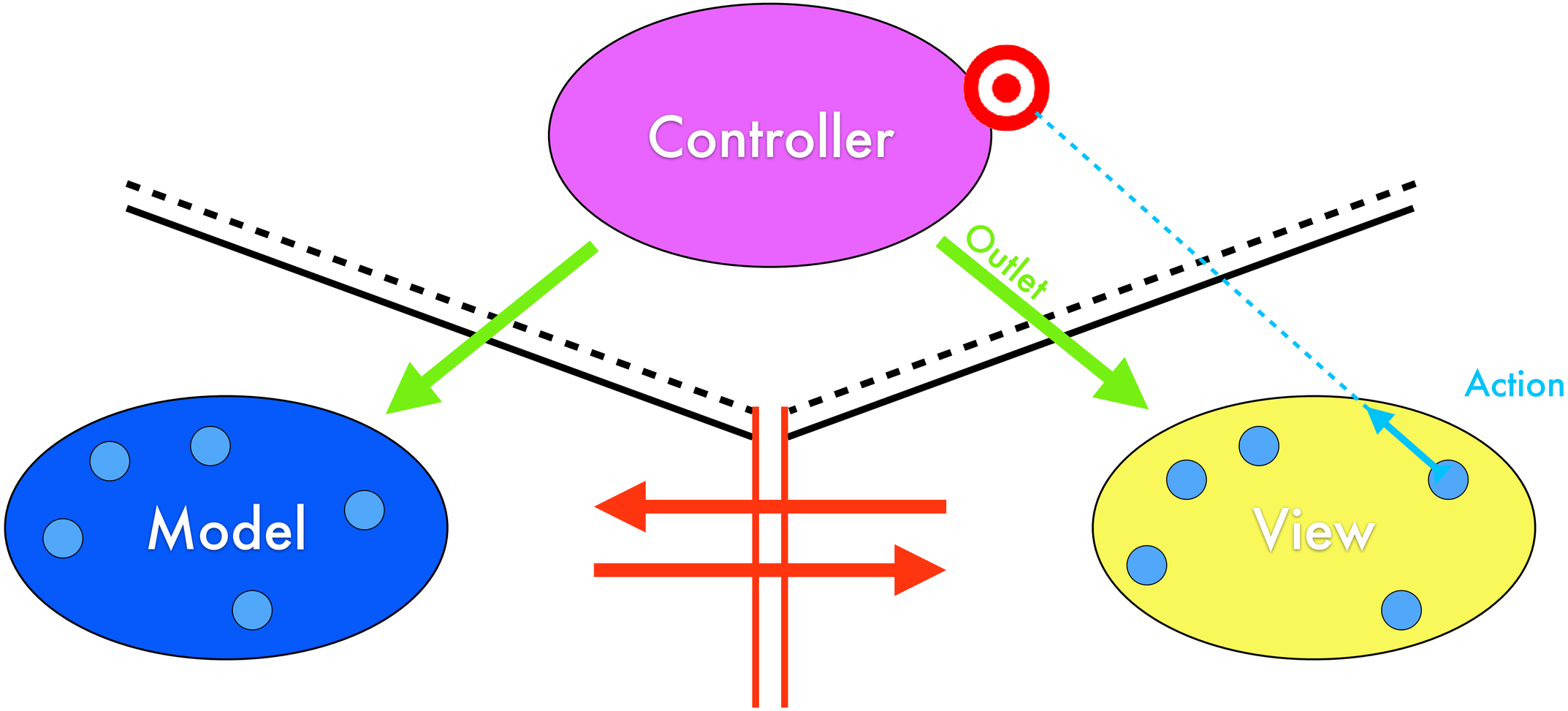
Der Controller stellt ein Target zur Verfügung…

Model View Controller



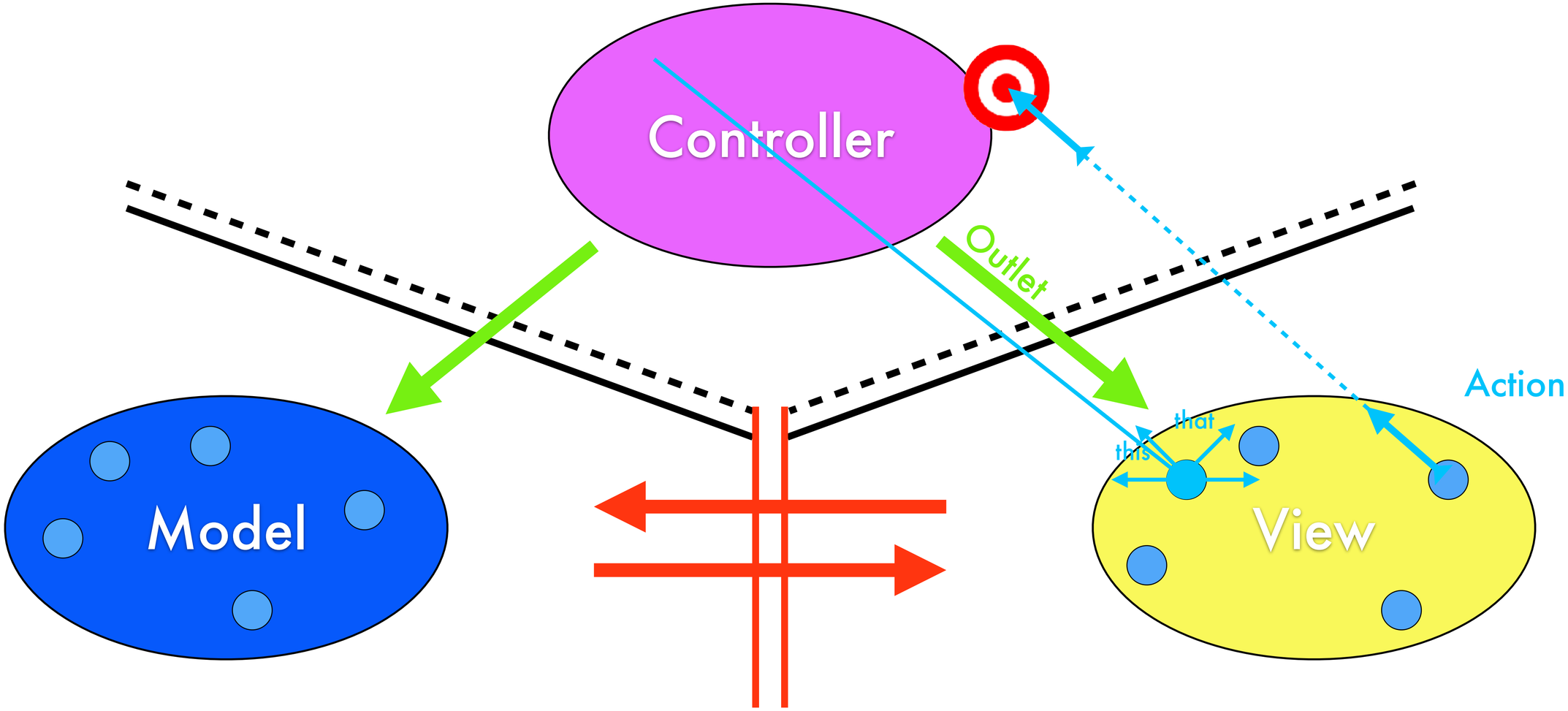
… und dem View eine Action bereit

Model View Controller



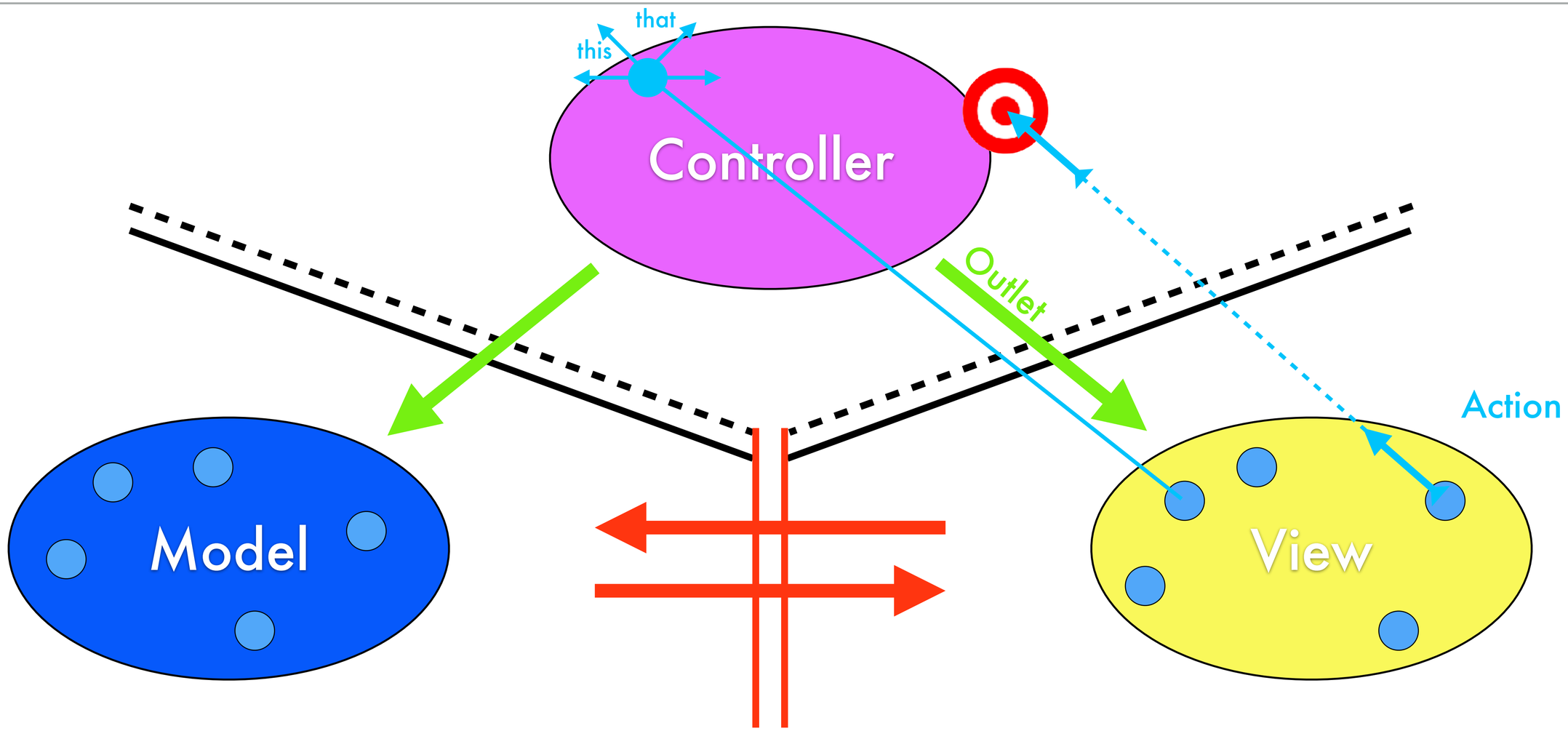
Wenn im UI ein Ereignis auftritt, schickt der View die Action an den Controller

Model View Controller



Manchmal muss der View sich mit dem Controller synchronisieren

Model View Controller

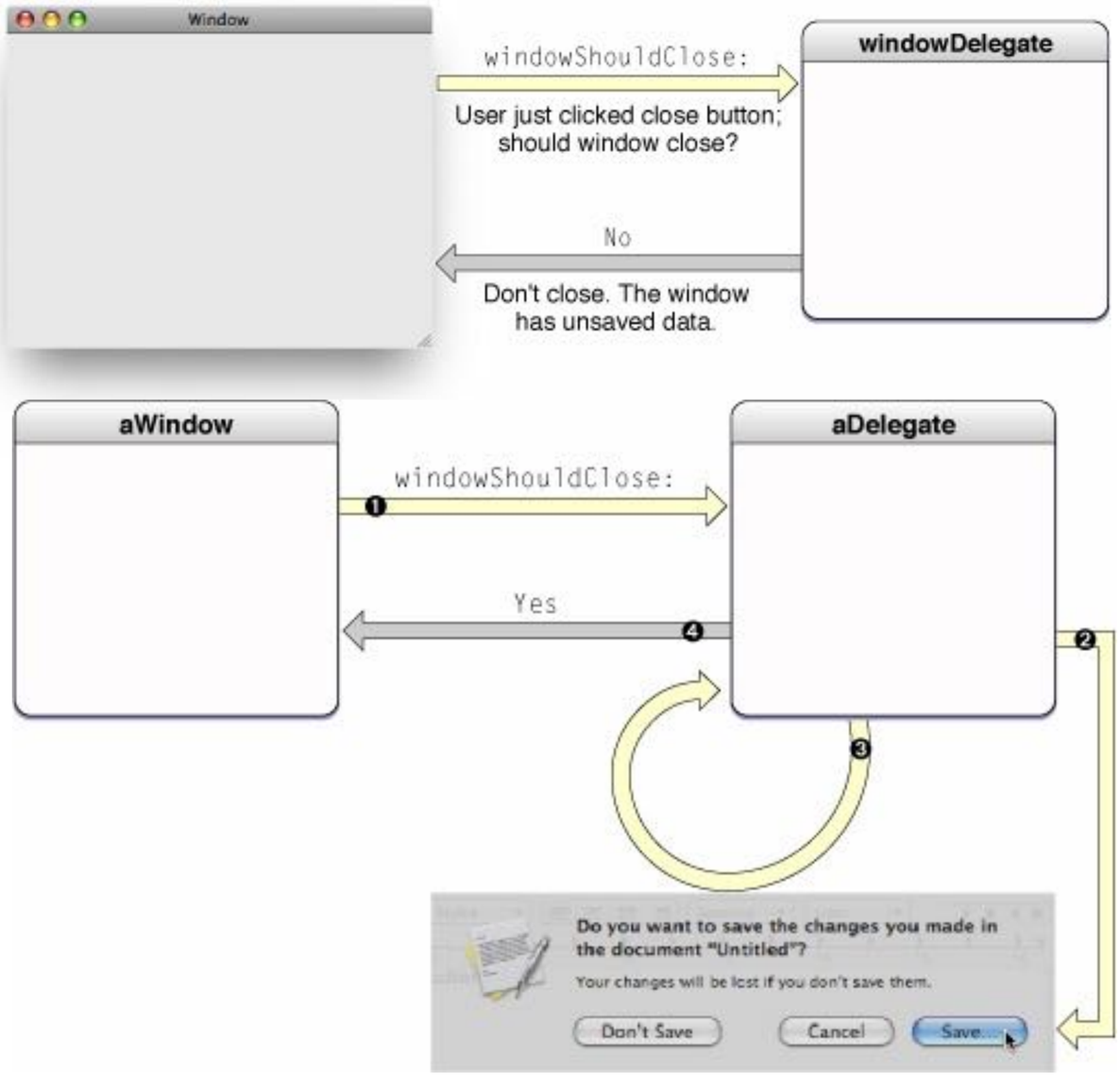


Der Controller setzt sich selbst als Delegate für den View

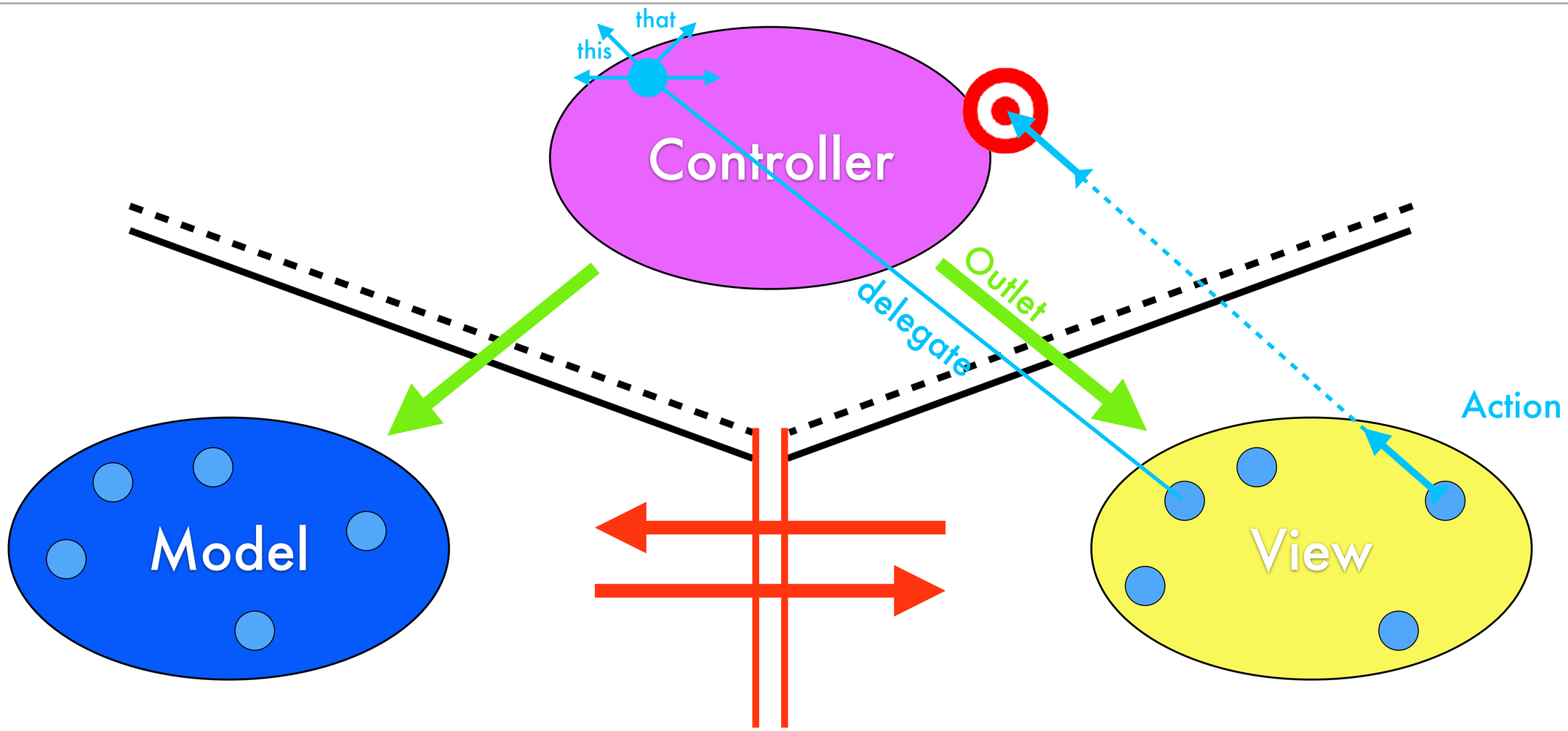
Was ist ein Delegate?

Ein delegate ist ein Objekt welches für ein anderes (oder in Koordination mit einem anderen) Objekt eine Aktion durchführt, sobald dieses Objekt auf ein Event trifft.

Das delegating Objekt ist meist ein Responder Objekt (NSResponder oder UIResponder) welches auf einen User-Event reagiert. Das Delegate ist das Objekt, an welches dann (zur Bearbeitung des Events) die Kontrolle übergeben wird.



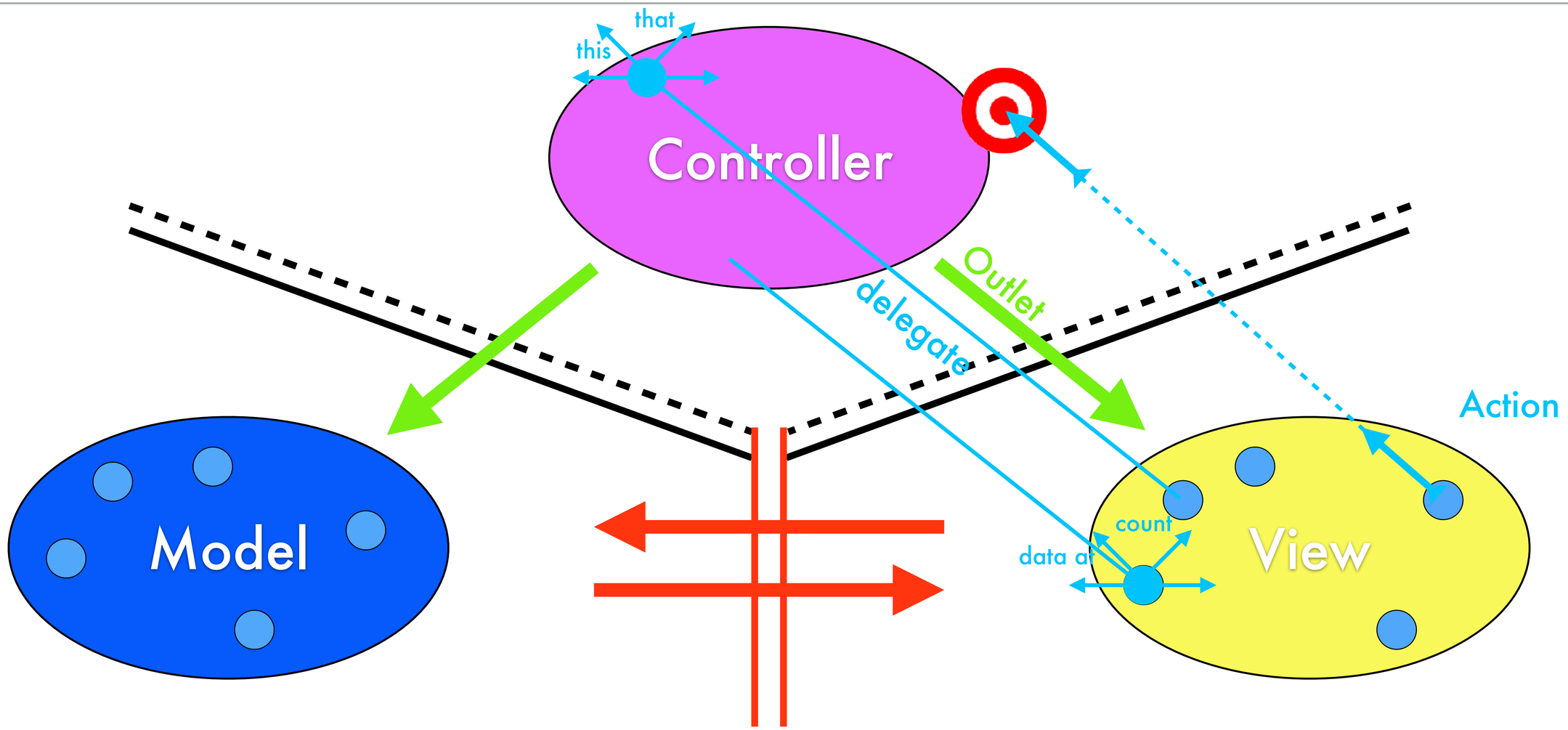
Model View Controller



Das Delegate wird per Protocol gesetzt (“blind” gegenüber der Klasse)

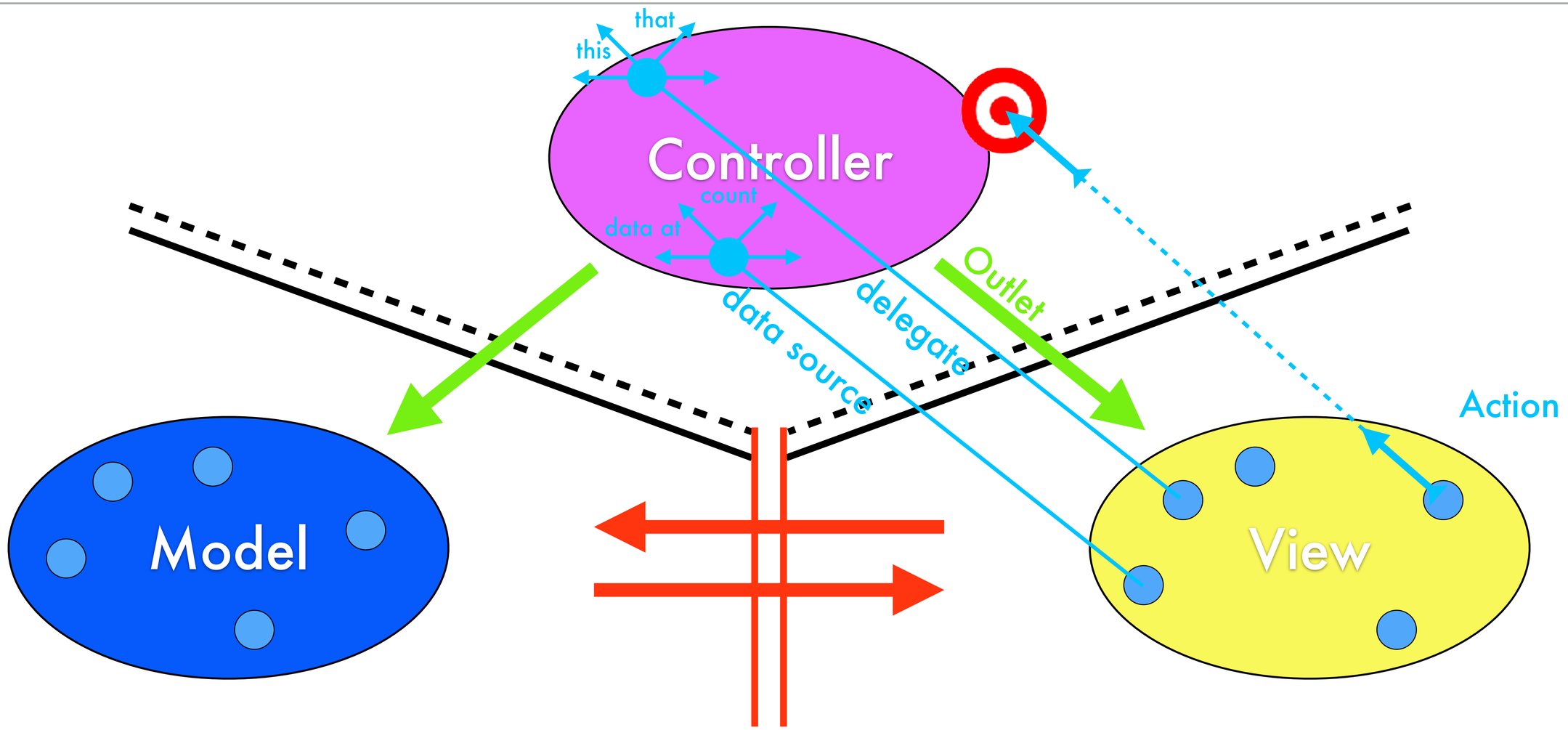
Wiederholung: Views besitzen die Daten nicht, die sie anzeigen

Model View Controller



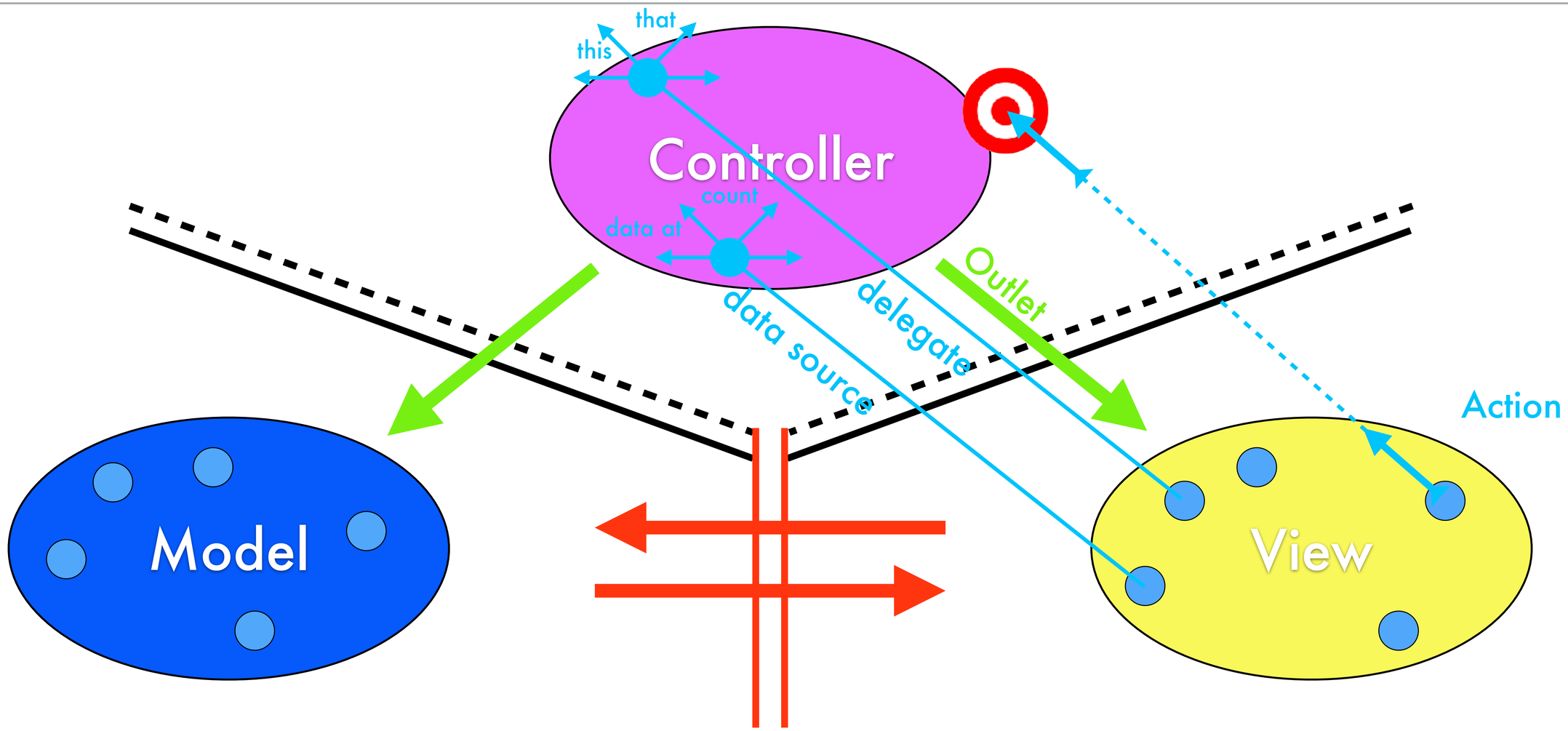
Sofern benötigt, haben sie ein Protocol um Daten zu erhalten

Model View Controller



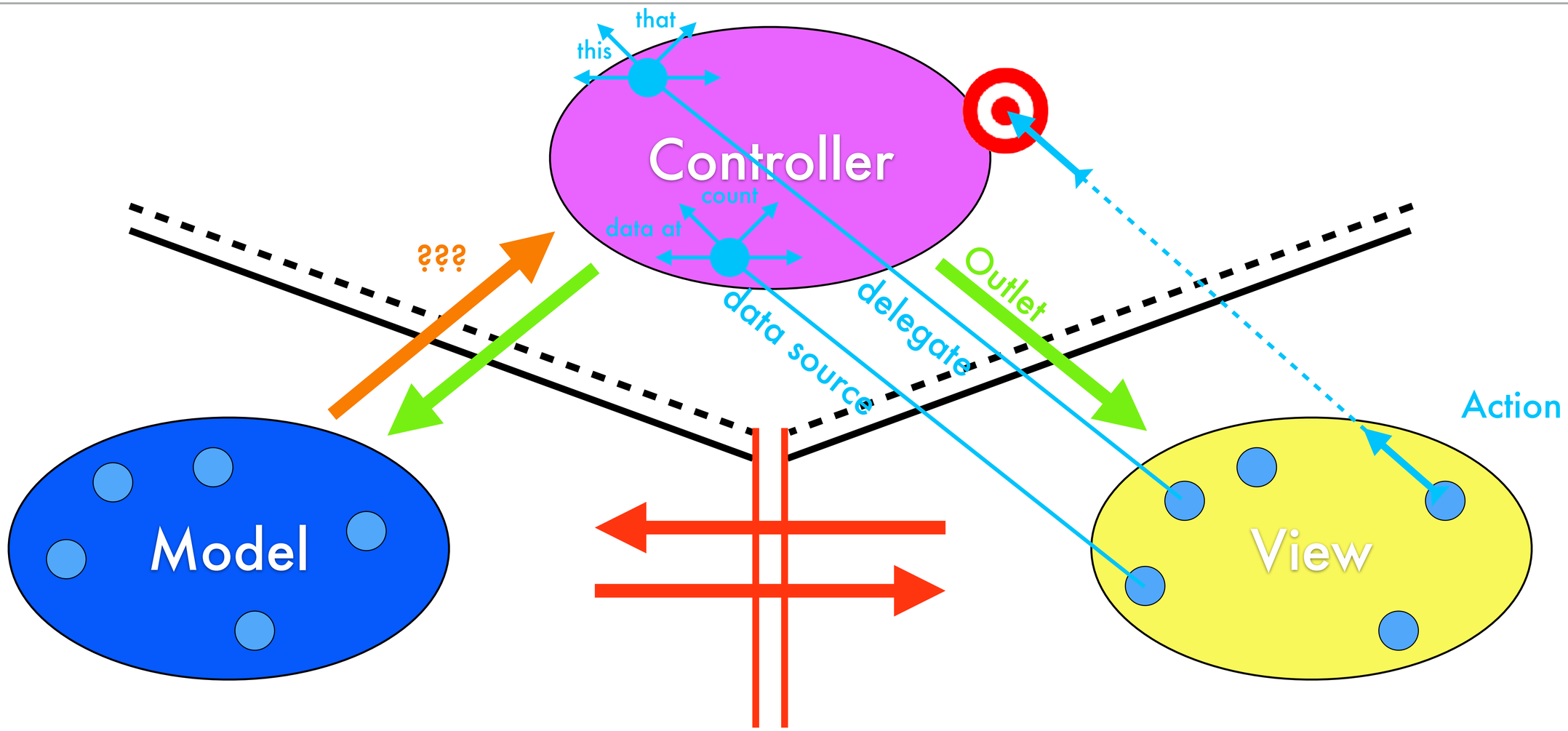
Normalerweise sind die Controller die Datenquelle (nicht das Model)

Model View Controller



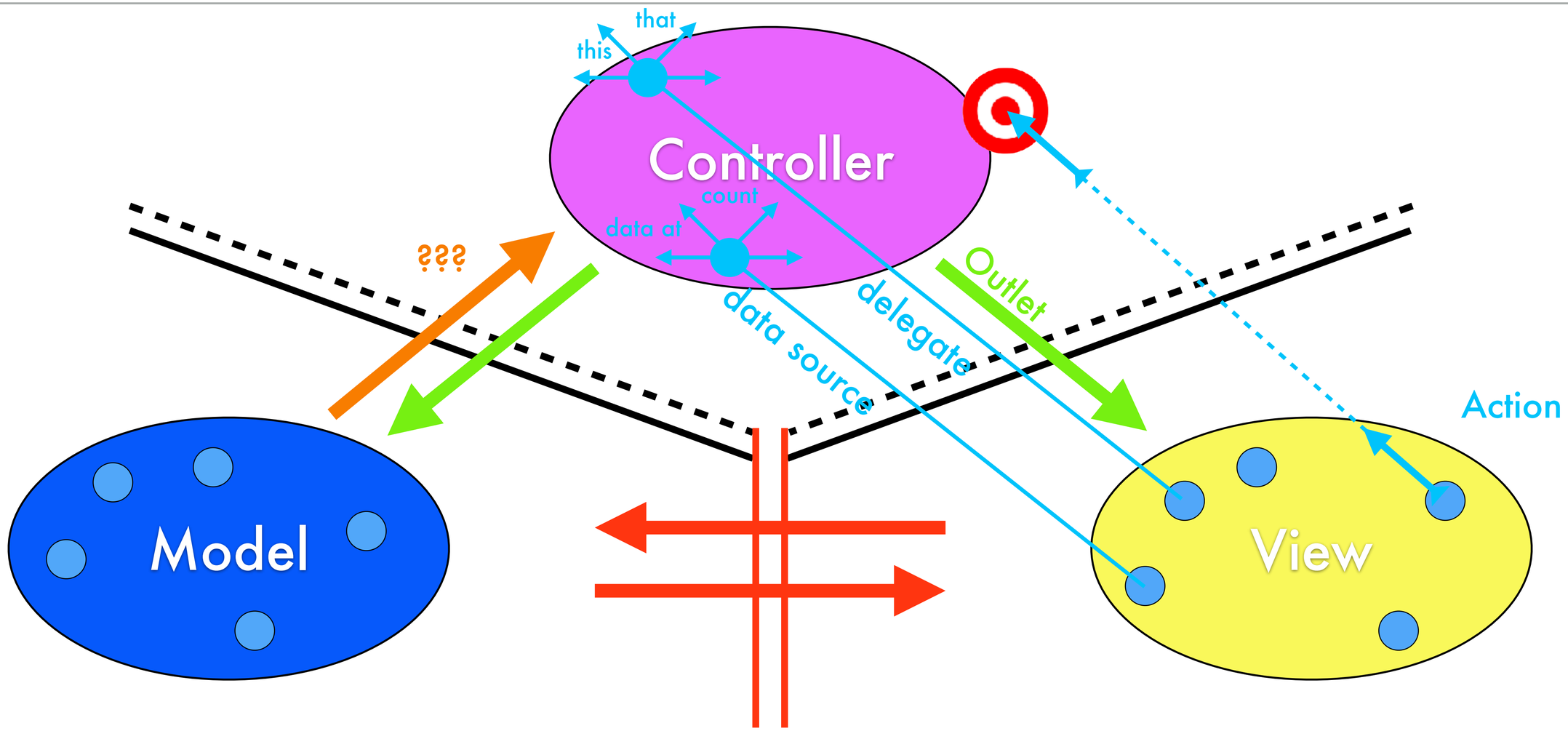
Controller formatieren/interpretieren die Model Information für den View

Model View Controller



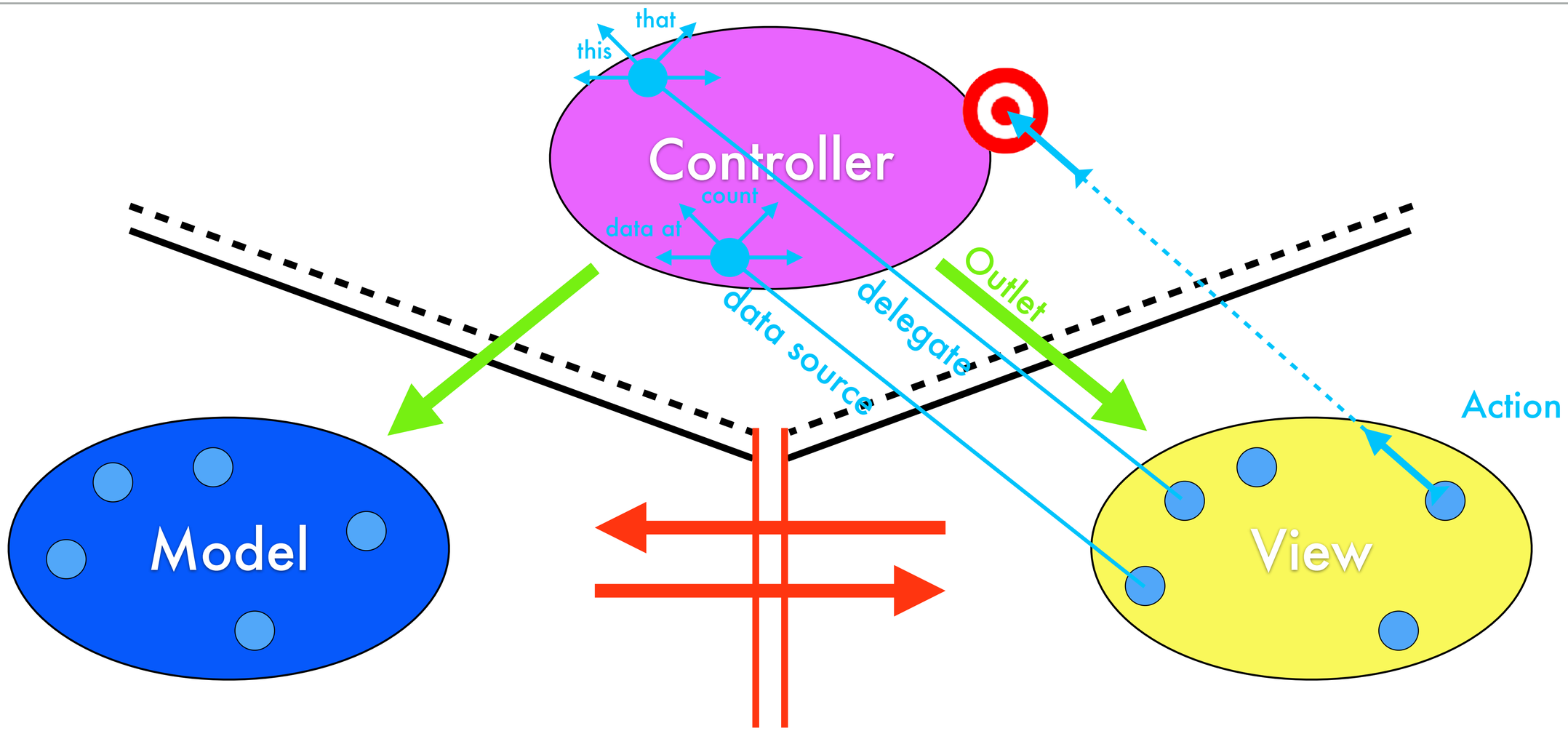
Kann das Model direkt mit dem Controller kommunizieren?

Model View Controller



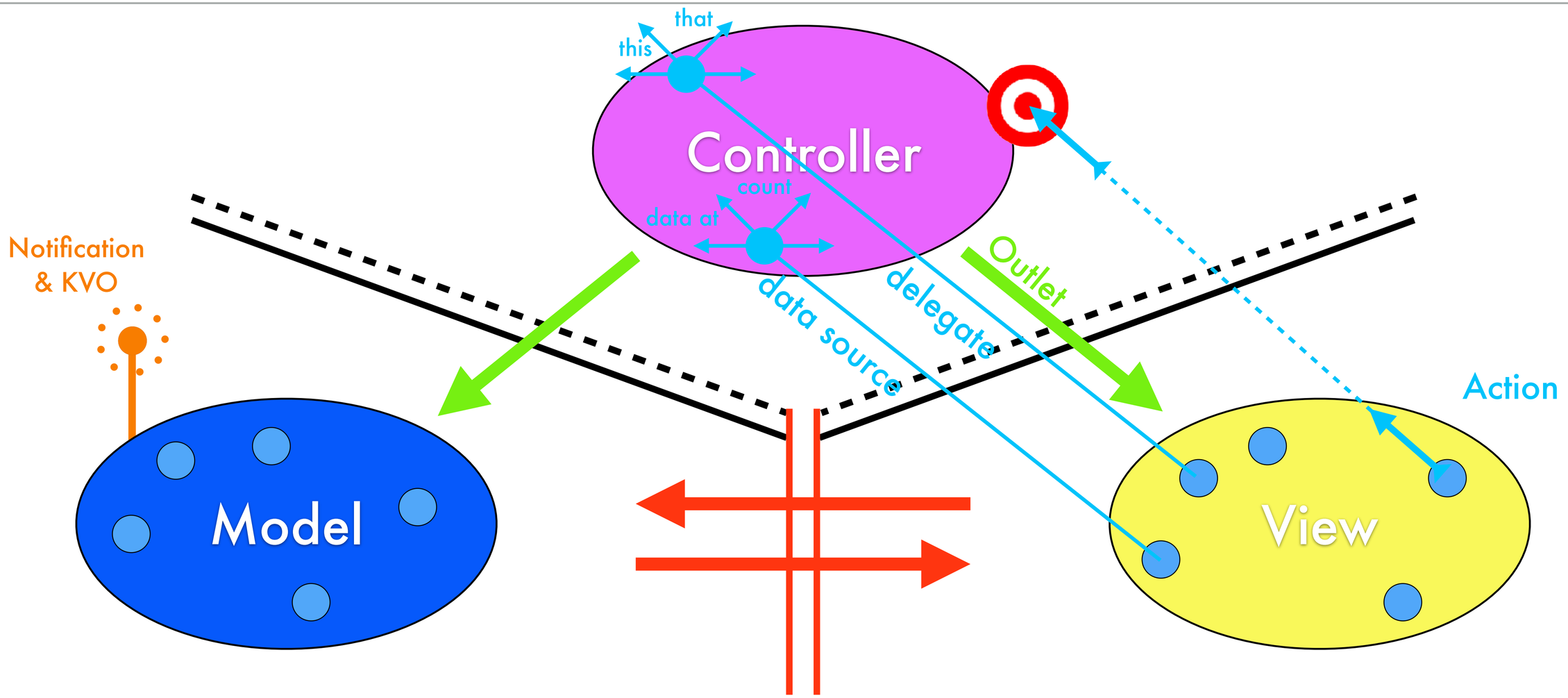
Nein! Das Model sollte unabhängig vom UI sein

Model View Controller



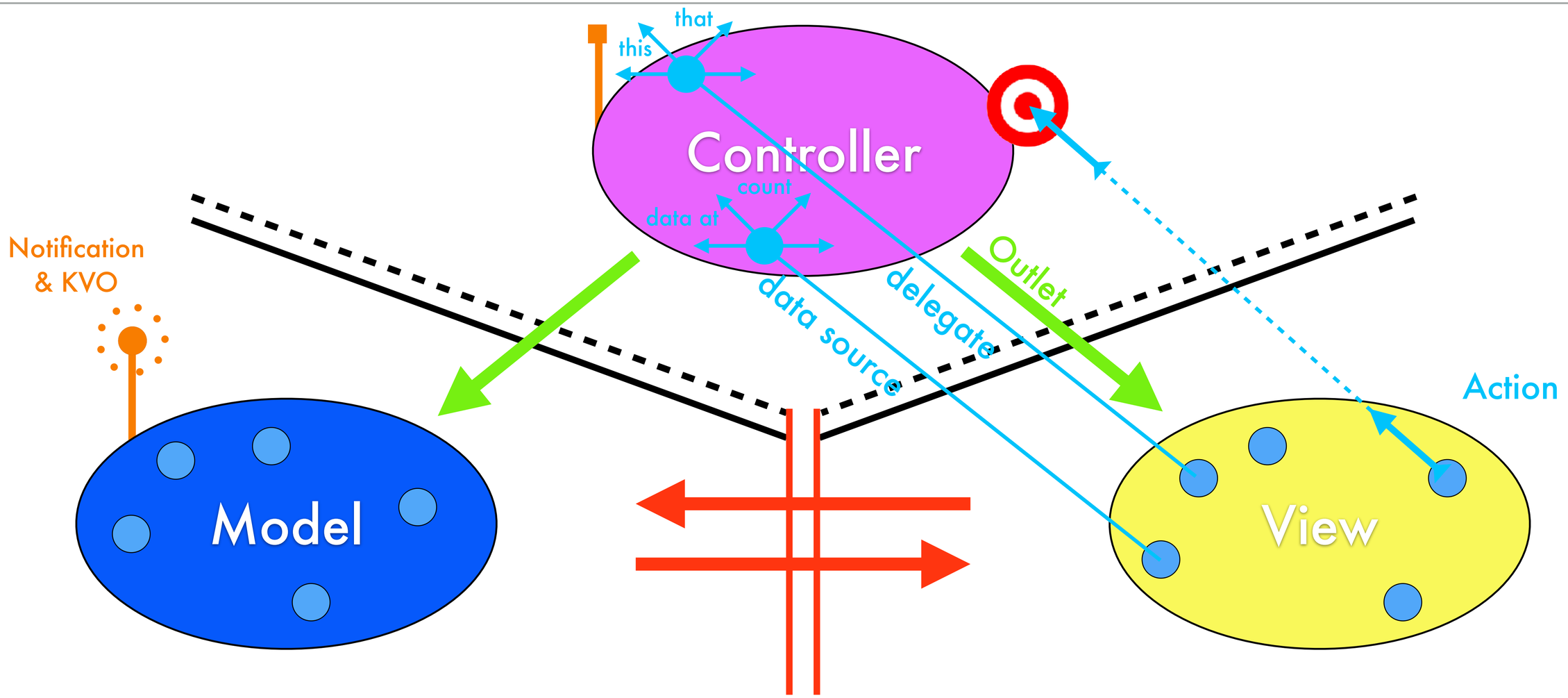
Was passiert also, wenn eine Information geupdated werden soll?

Model View Controller



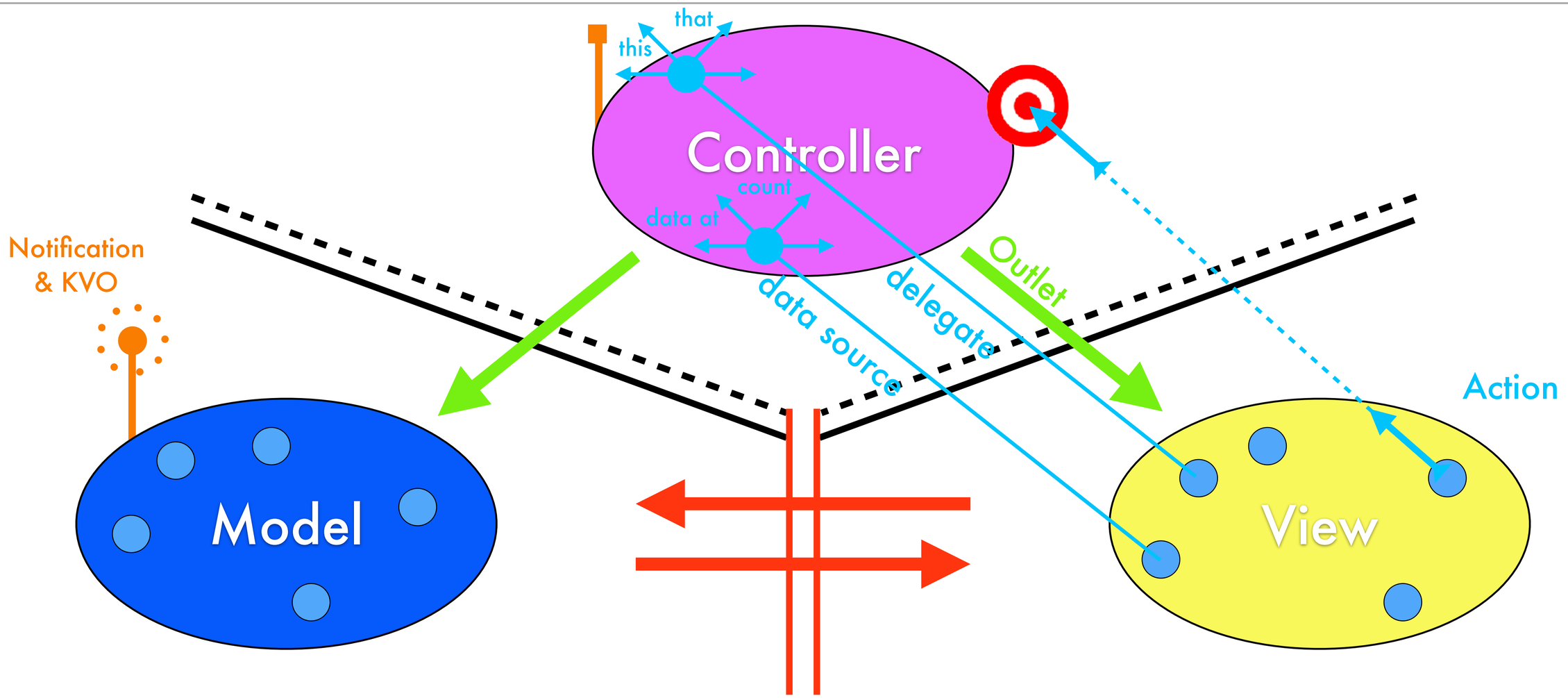
Ein Broadcast Mechanismus wird verwendet (wie ein Radio-Sender)

Model View Controller



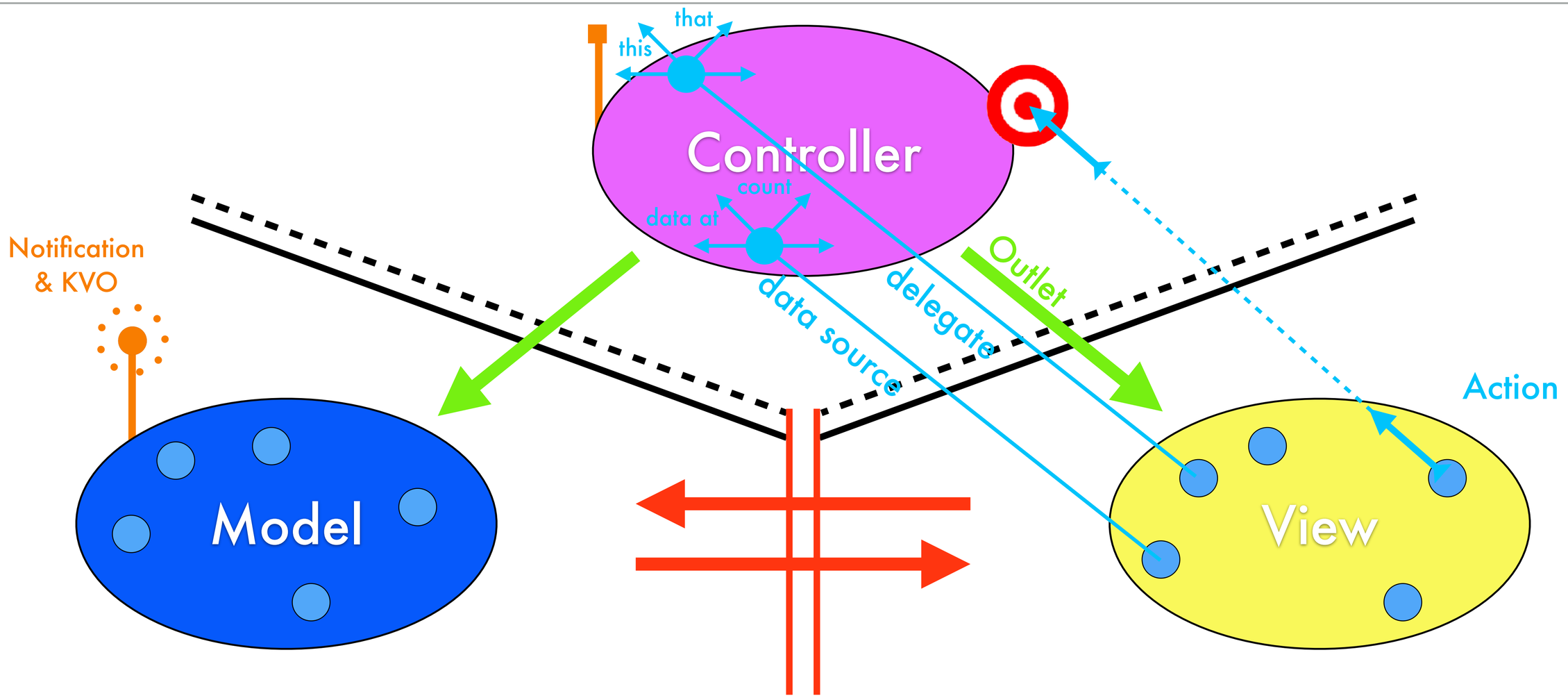
Controller (oder andere Models) stellen den Empfänger für interessante Dinge ein

Model View Controller



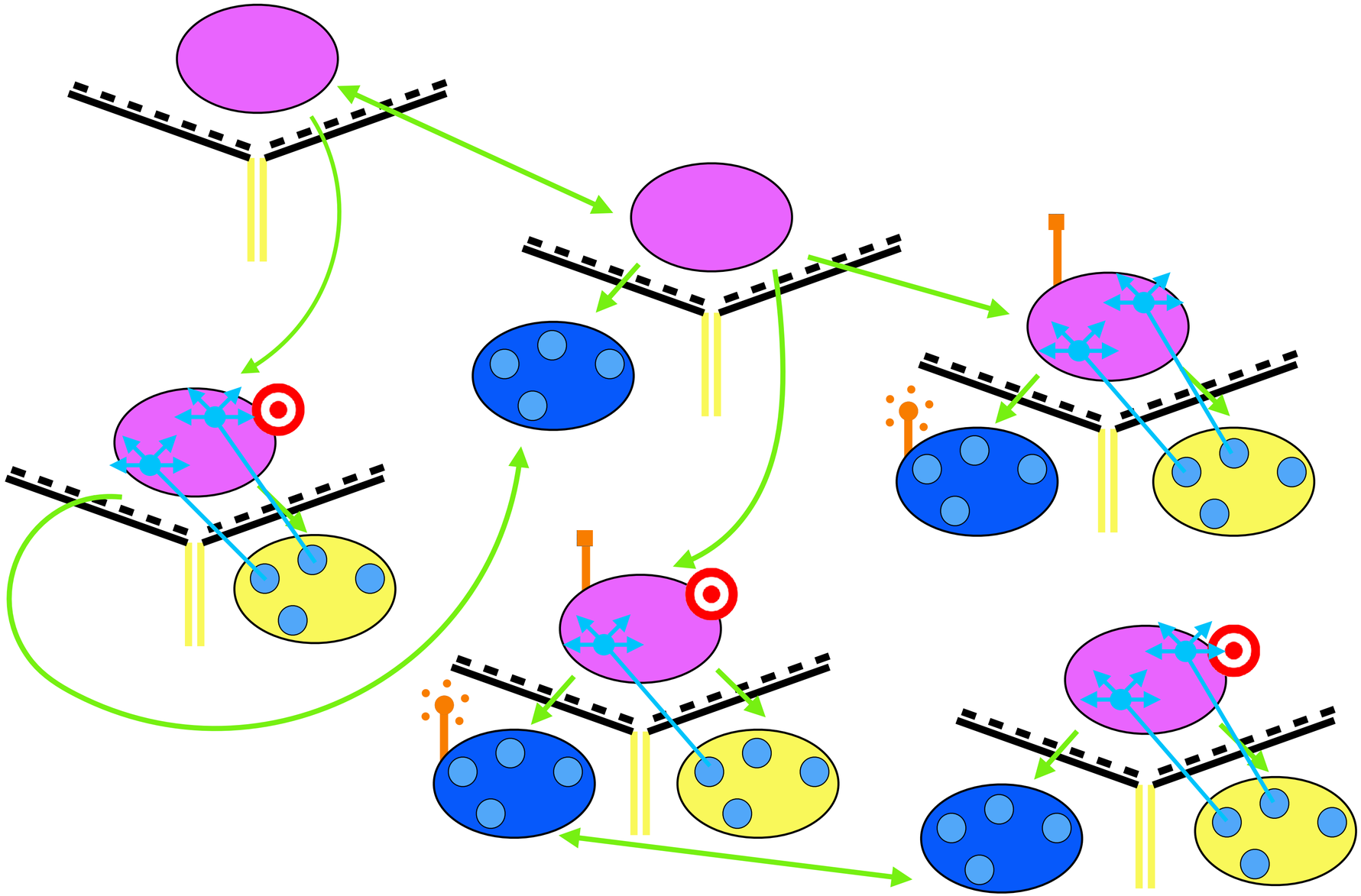
Ein View könnte "zuhören", aber wahrscheinlich nicht einem Model-Sender

Model View Controller

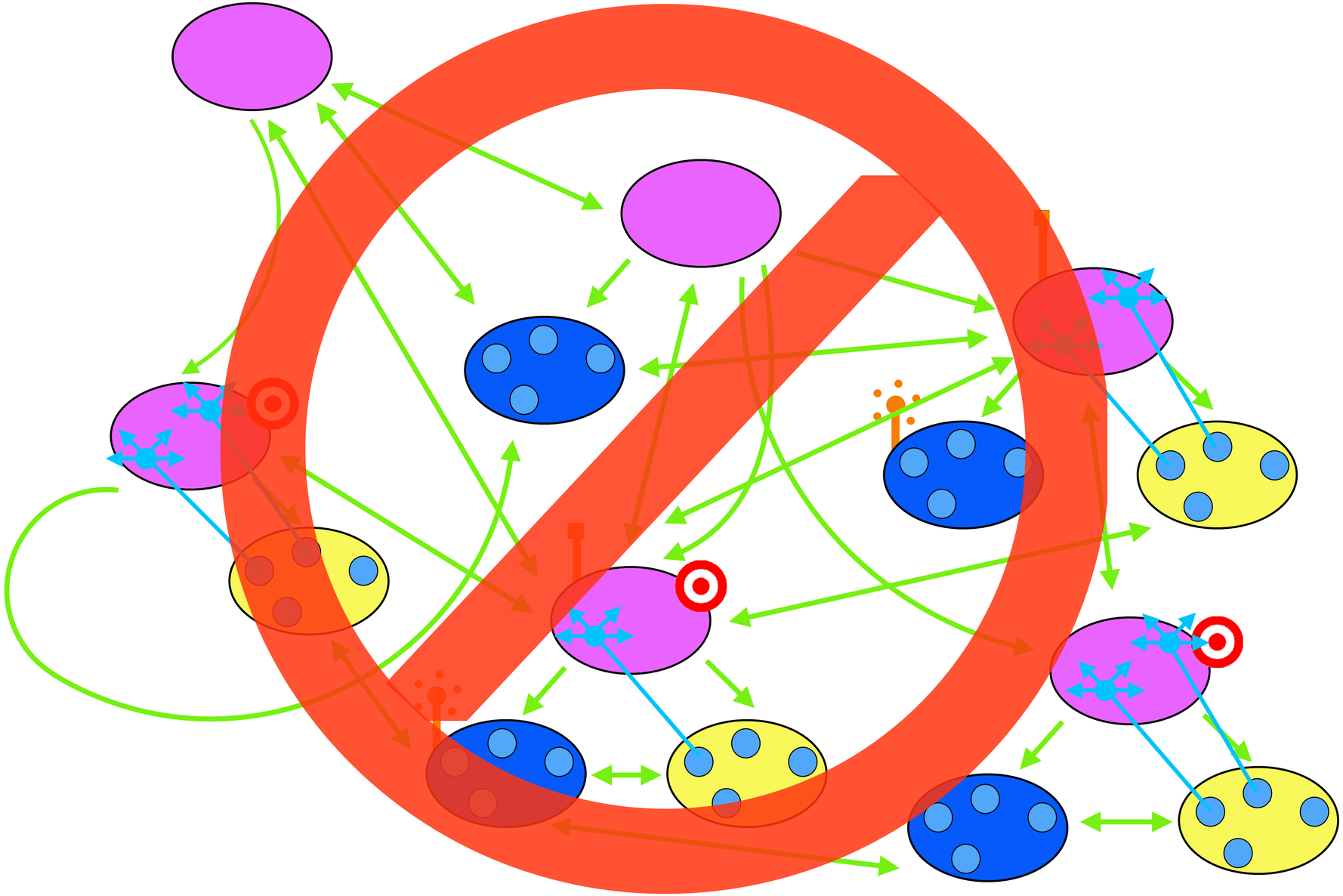


MVC Gruppen können kombiniert werden, um komplizierte Apps zu erstellen …

MVCs arbeiten zusammen…



MVCs die nicht zusammen arbeiten…



Let's get started...

- Fortsetzung der Demo...
  - MVC
  - `struct` vs. `class` (`mutating`, etc.)
  - public vs. `private` API
  - Mehr zu Optionals
  - `Dictionary<KeyType, ValueType>`
  - `enum`
  - Associated Values
  - `switch`
  - Funktionen als Typen
  - Closure Syntax zur Definition von Funktionen "on the fly"
  - `UIStackView`
  - Erster Blick auf Autolayout (pinnen von Dingen zu Rändern)