

App-Entwicklung für iOS und OS X

SS 2016
Stephan Gimbel



Hello & Welcome

⚙️ Organisation

Ziele

Kurs

Voraussetzungen

Vorlesung & Praktika

Software

Prüfung & Note

⚙️ iOS Überblick

Komponenten

⚙️ MVC

Objektorientierte Design Konzepte

⚙️ Swift 2

Sprache

Konzepte (Grundlagen)

Organisation

Ziele

⚙ Apps entwickeln

Komplexe Ideen einfach umsetzen

Ergebnisse direkt ausprobieren

Veröffentlichen von Apps über den AppStore

Community

⚙ Objektorientierung

Cocoa (Touch) ist OO

Anwendung von Design Modellen (MVC, ...)

Viele andere Konzepte wie: Grafik, Datenbanken, Multimedia, Multi-Threading, Netzwerk-Kommunikation, Cloud, usw...

Kein (!!!) “zusammenklicken” von Apps via Interface Builder, sondern echtes Softwareengineering

⚙ Erfahrung

Wir entwickeln echte Apps und keine Beispiele zum testen im Praktikum

→ Verkauf im AppStore

Organisation

Kurs



Die hier gelernten Inhalte können zum Teil auch für C++, Java, Python, Linux, Windows, Android, etc. verwendet werden!

⚙ 2+2 Veranstaltung V+P

Seminaristische Veranstaltung

5 ECTS Punkte (Credit Points) → 150 Zeitstunden (schaffen wir nicht)

⚙ Vorlesung

Montags 2. Block (10:15 - 11:45 Uhr) in D14/103

⚙ Praktikum

14-tägig (Doppelblock), Do56x/y (16:00 - 19:15 Uhr) in D14/310

In 2-er Teams

Organisation

Kurs



**Änderung: Das Praktikum beginnt am 14.04.2016!!!
5 Termine pro Gruppe.**



Organisation

Kurs

⚙️ Voraussetzung

Programmieren / Algorithmen und Datenstrukturen I + 2 (1.+2. Sem.)

Objektorientierte Analyse und Design / Software Engineering (2.+3. Sem.)

Generell gute Kenntnisse in OO

⚙️ Programmiererfahrung

Sie sollten schon mal größere Programme (10+ Klassen) selbst entwickelt haben (ENA, PSE, Nebenjob,... ?)

Organisation

Kurs

⚙ Benotung

70% aus mündlicher Prüfung (Termine werden rechtzeitig bekannt gegeben)
Dauer, ca. 30-40 Minuten (Zweiergruppen)

30% aus dem letzten Praktikum (Bearbeitungszeit 4 Wochen)

Hinweis! Fangen Sie rechtzeitig mit der Bearbeitung des letzten Praktikums an! Ein Teil Ihrer Endnote hängt davon ab!

⚙ Unterlagen

Moodle (einloggen mit st-Account)
Einschreibeschlüssel: helloagain

⚙ Anregungen / Kommentare / Fragen

stephan.gimbel@h-da.de

Oder noch besser direkt in der Vorlesung / Praktikum

Organisation

Software

⚙ Xcode 7 & Developer Tools

Frei erhältlich im AppStore

Benötigt Rechner mit OS X (kein Windows, kein Linux - vielleicht aber, in Eigenregie in einer VM?)

Labor: bereits installiert in D14/310 (gerne auch eigenes MacBook mitbringen)

⚙ Developer Certificate & Provisioning Profile

Freischaltung des eigenen iOS Devices für das Semester

Benötigt (per Mail): Vor-/Nachname und email Adresse (wie im Dev-Center verwendet: developer.apple.com), sowie UUID des iOS Devices

Auch falls Sie ein iOS Device aus dem Labor nutzen möchten, brauchen Sie auf jeden Fall einen eigenen (kostenlosen) Developer Account

Organisation

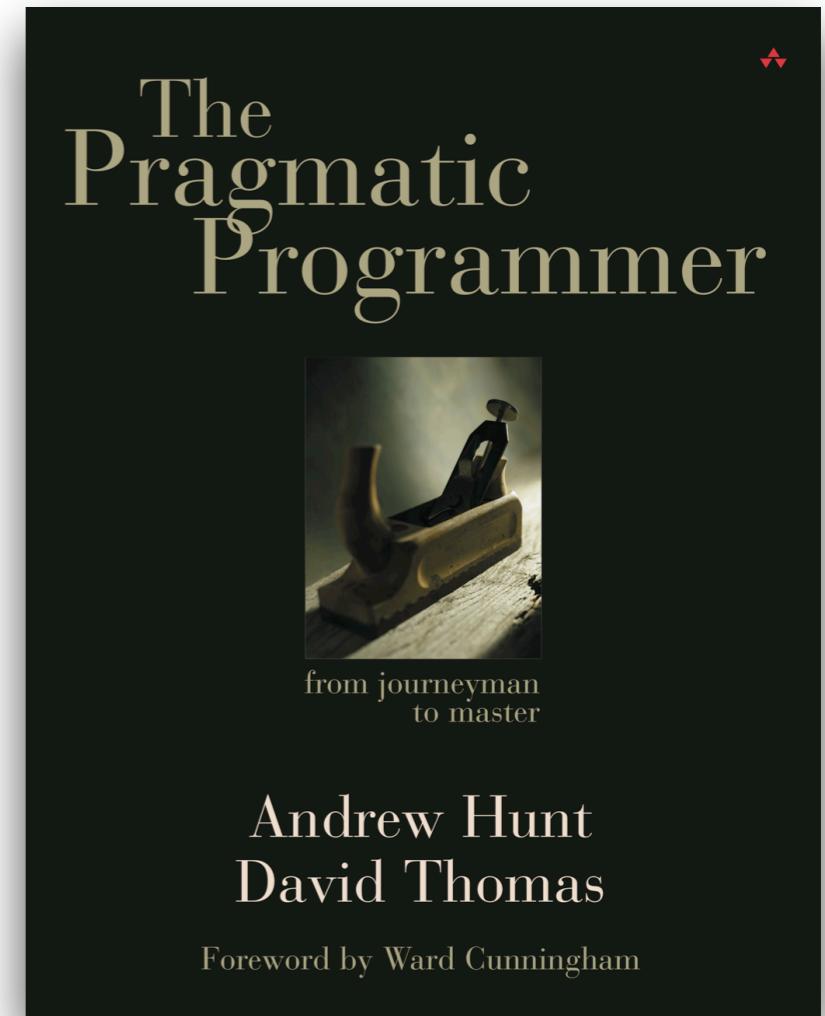
Literatur

⚙️ Apple

Es gibt viele Tutorials, Bücher, usw., wir beschränken uns auf die offizielle Dokumentation von Apple → developer.apple.com

⚙️ Generelle Empfehlung

Sollte jeder Informatiker gelesen haben...



Andrew Hunt
David Thomas

Foreword by Ward Cunningham

iOS Überblick



⚙️ Core OS

- OS X Kernel
- Mach 3.0
- BSD
- Security
- Power Management
- Keychain Access
- Certificates
- Bonjour

iOS Überblick



⚙️ Core Services

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

iOS Überblick



⚙️ Media

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

iOS Überblick



⚙️ Core Touch

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

Meet the (current) iOS Family

iPhone

Model	6S Plus	6S	6 Plus	6	5S
Chip	A9 + M9	A9 + M9	A8 + M8	A8 + M8	A7 + M7



Meet the (current) iOS Family

iPad

Model	Pro	Air 2	Air	mini 4	mini 2
Chip	A9X + M9	A8X + M8	A7 + M7	A8 + M8	A7 + M7



Meet the (current) iOS Family Chips



CPU	ARM Dual-Core 1.3/1.4 GHz (28nm)	ARM Dual-Core 1.4 GHz (20nm)	ARM Triple-Core 1.5 GHz (20nm)	ARM Dual-Core 1.85 GHz (14/16nm)	ARM Dual-Core 2.26 GHz (16nm)
Art	64 Bit system-on-a-chip	64 Bit system-on-a-chip	64 Bit system-on-a-chip	64 Bit system-on-a-chip	64 Bit system-on-a-chip
GPU	PowerVR Quad-Core (115.2 GFLOPS)	PowerVR Quad-Core (115.2 GFLOPS)	PowerVR Octa-Core (230.4 GFLOPS)	PowerVR 7XT GT7600 Hex-Core (384/768 GFLOPS)	Custom PowerVR 7XT 12-Core (zwischen 1024 und 2048 GFLOPS)
OpenGL ES	3.1	3.1	3.1	3.1	3.1

Komponenten

⚙ Tools



Xcode 7



Instruments

⚙ Sprache

`display.textColor = UIColor.blackColor()`

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

⚙ Frameworks



Foundation

Core Data



Map Kit

UIKit

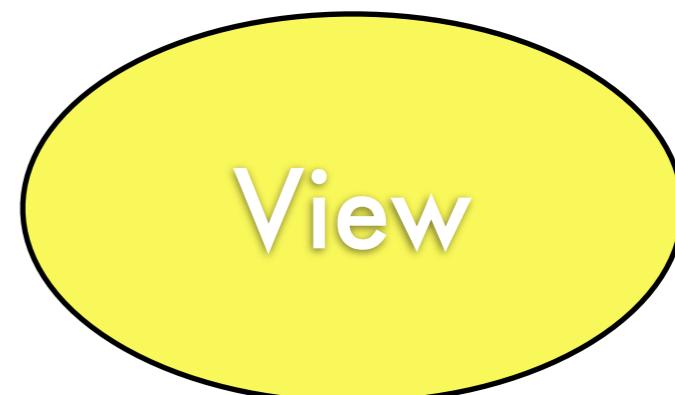
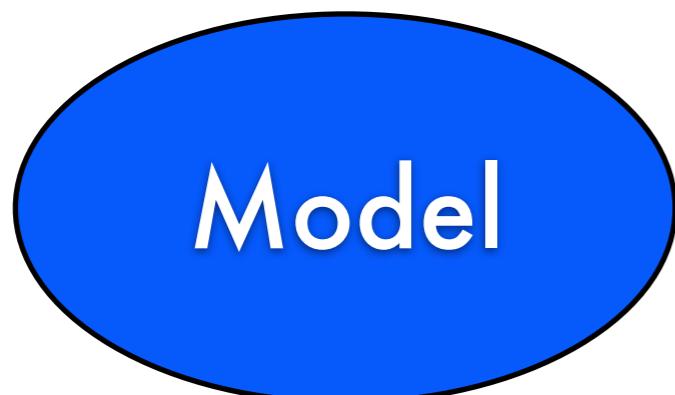
⚙ Design Strategien

MVC

Model View Controller

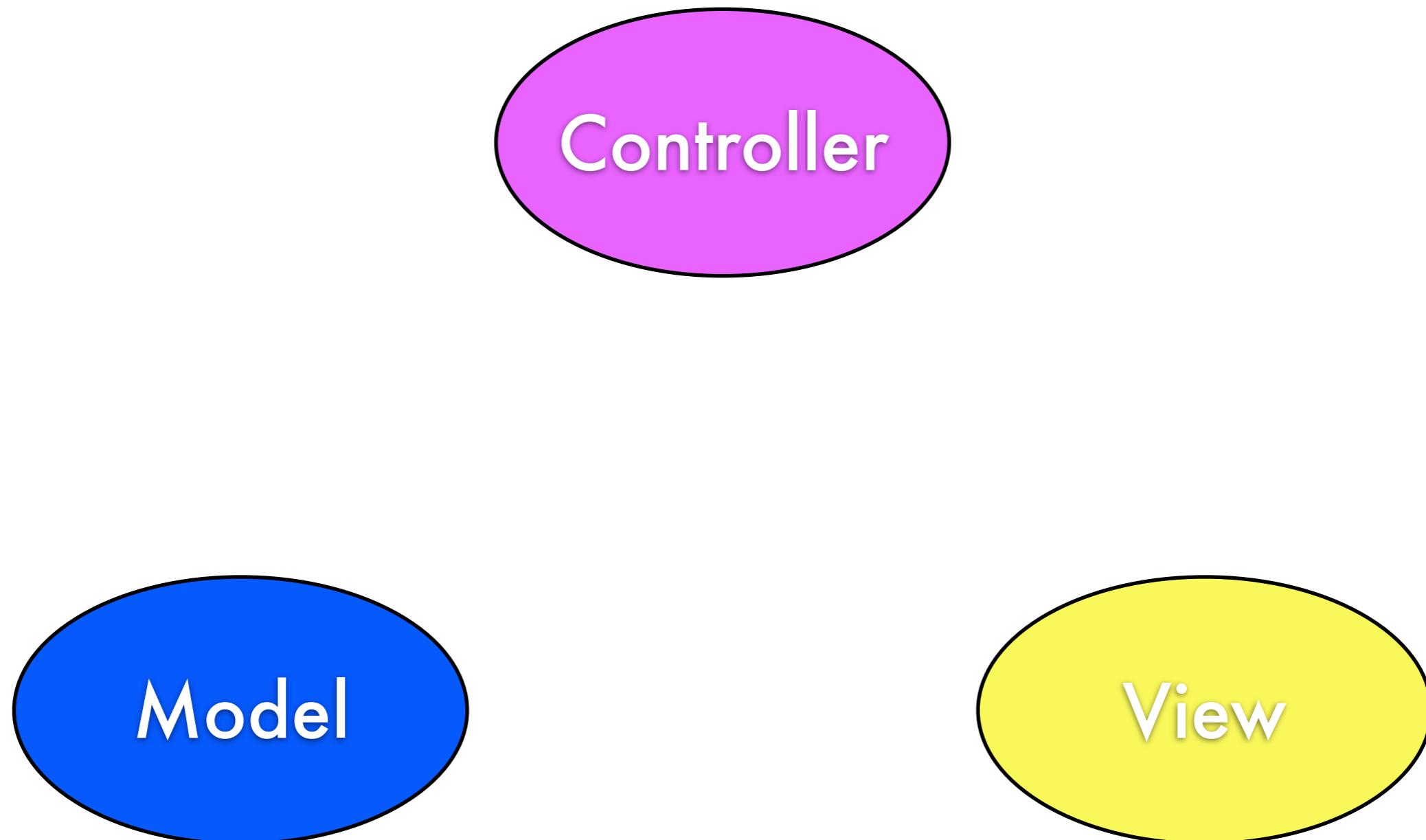


One Pattern to Rule
Them All



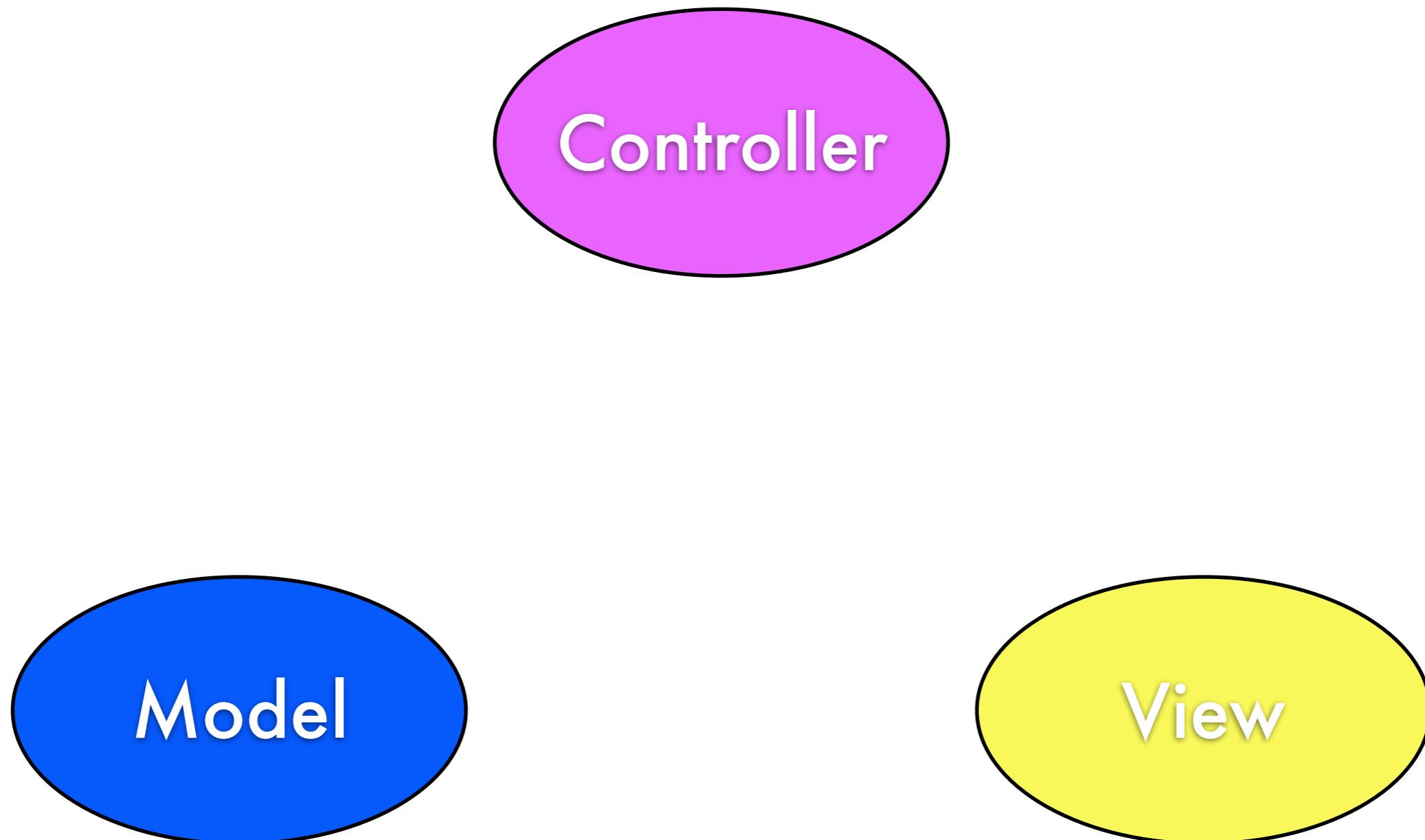
Definition von drei verschiedenen Rollen und wie diese miteinander
kommunizieren

Model View Controller



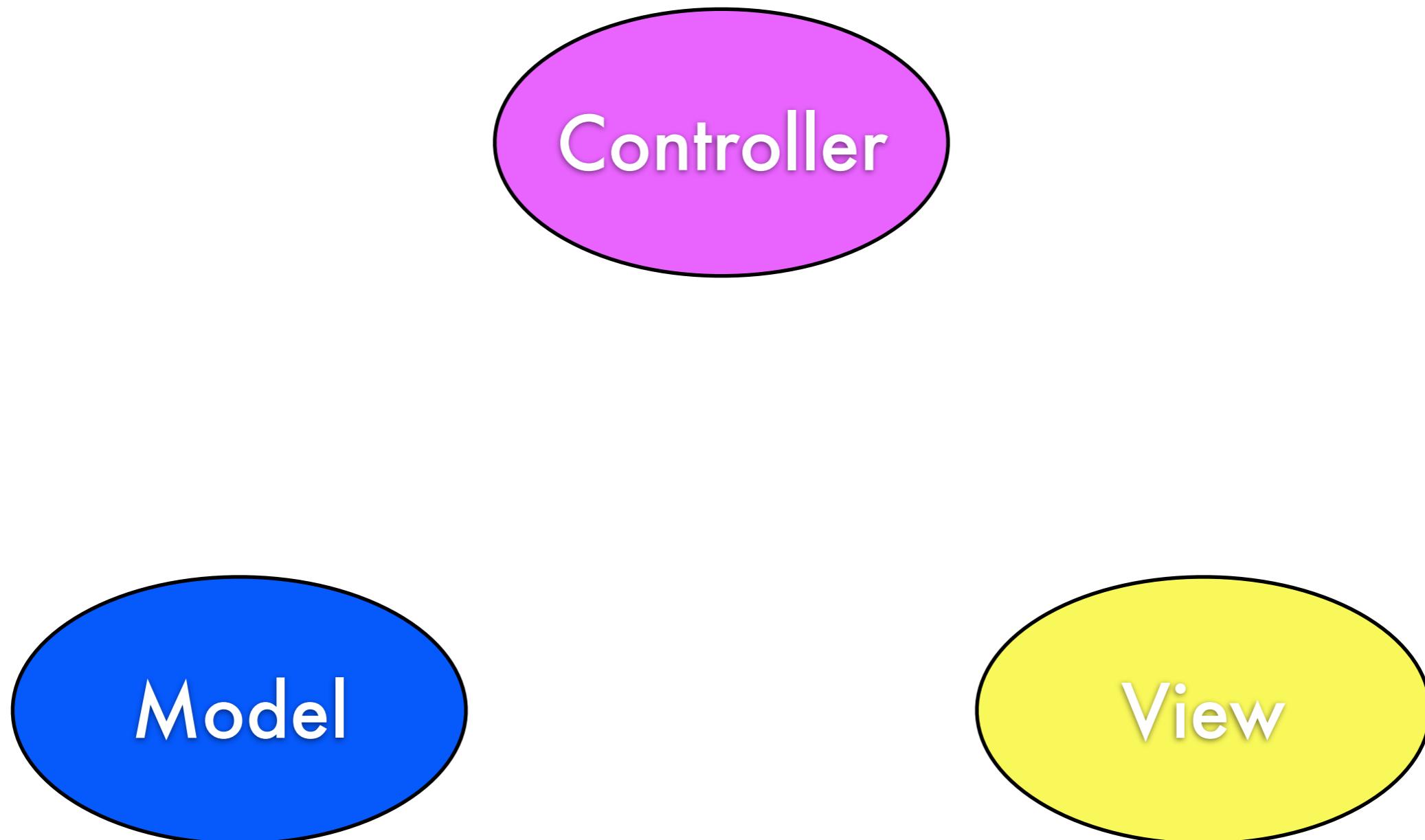
Model: Gibt an, was die App ist (Daten, inkl. Teile der Logik wie diese Daten verarbeitet, aber nicht wie diese an gezeigt werden

Model View Controller



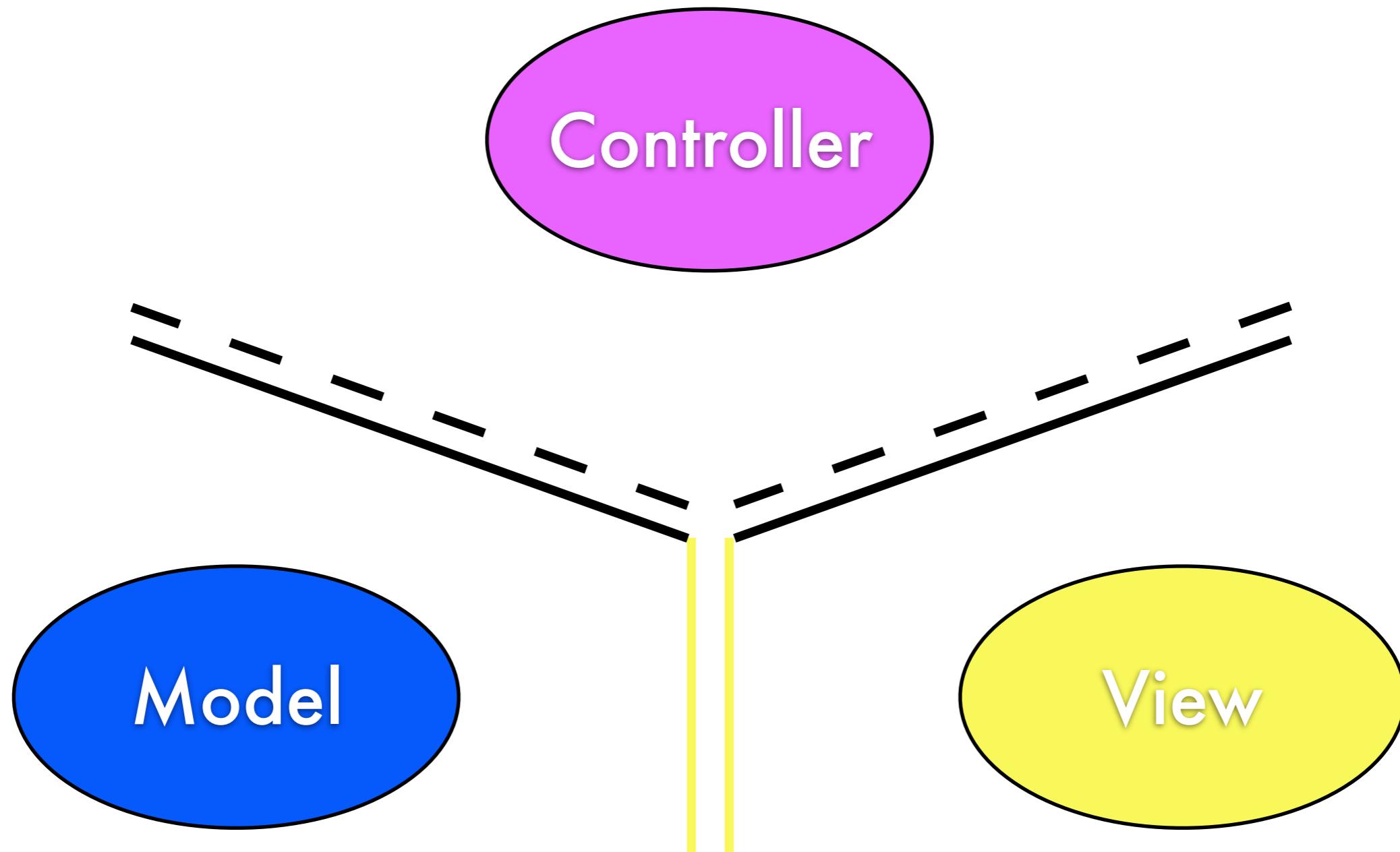
Controller: "vermittelt" zwischen Model und View und koordiniert Aufgaben innerhalb der App

Model View Controller



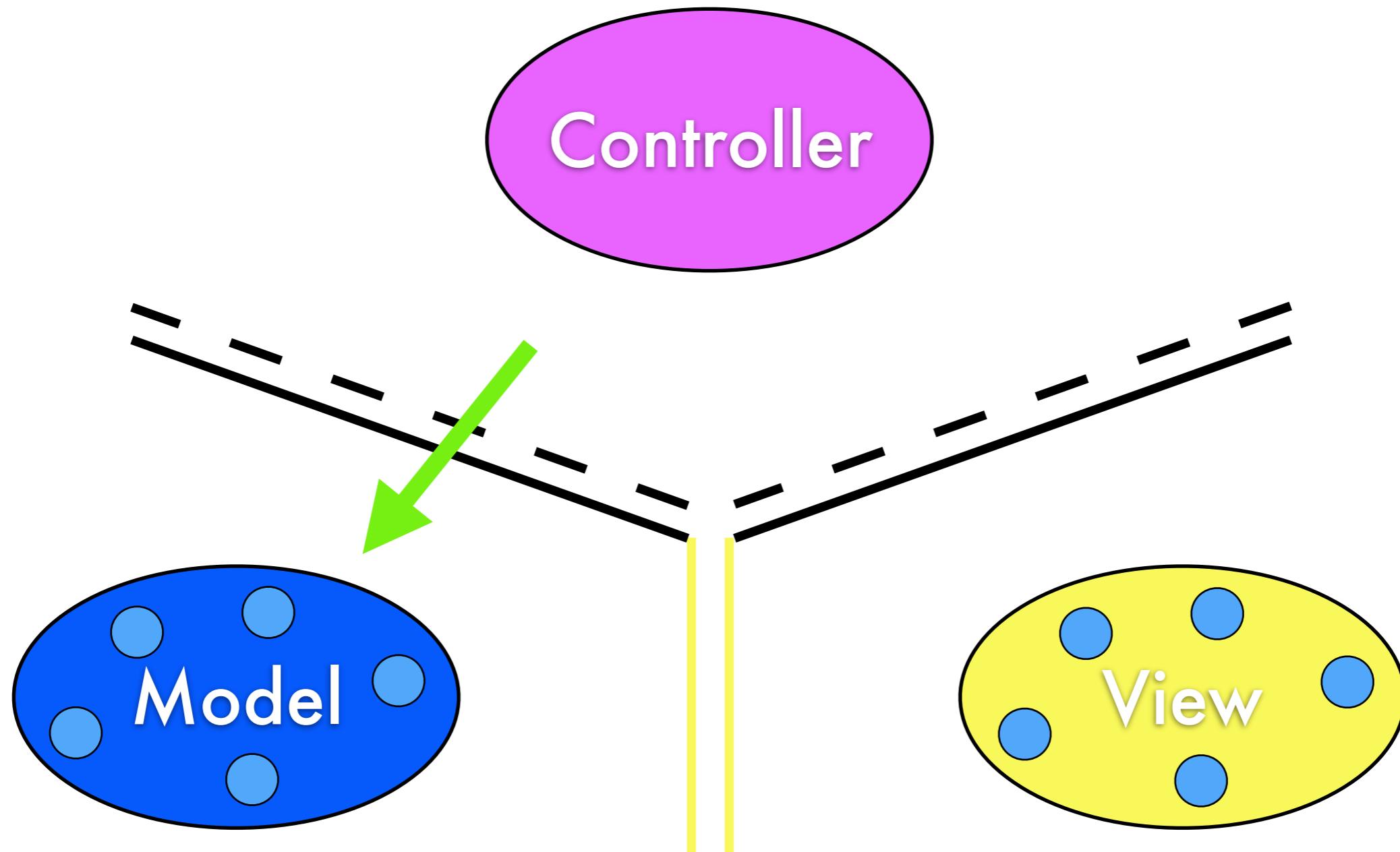
View: das was der User sehen und mit dem er interagieren kann: Buttons, Text-Felder, Grafiken, usw. (View weiß wie er gezeichnet wird und auf User-Action reagiert)

Model View Controller



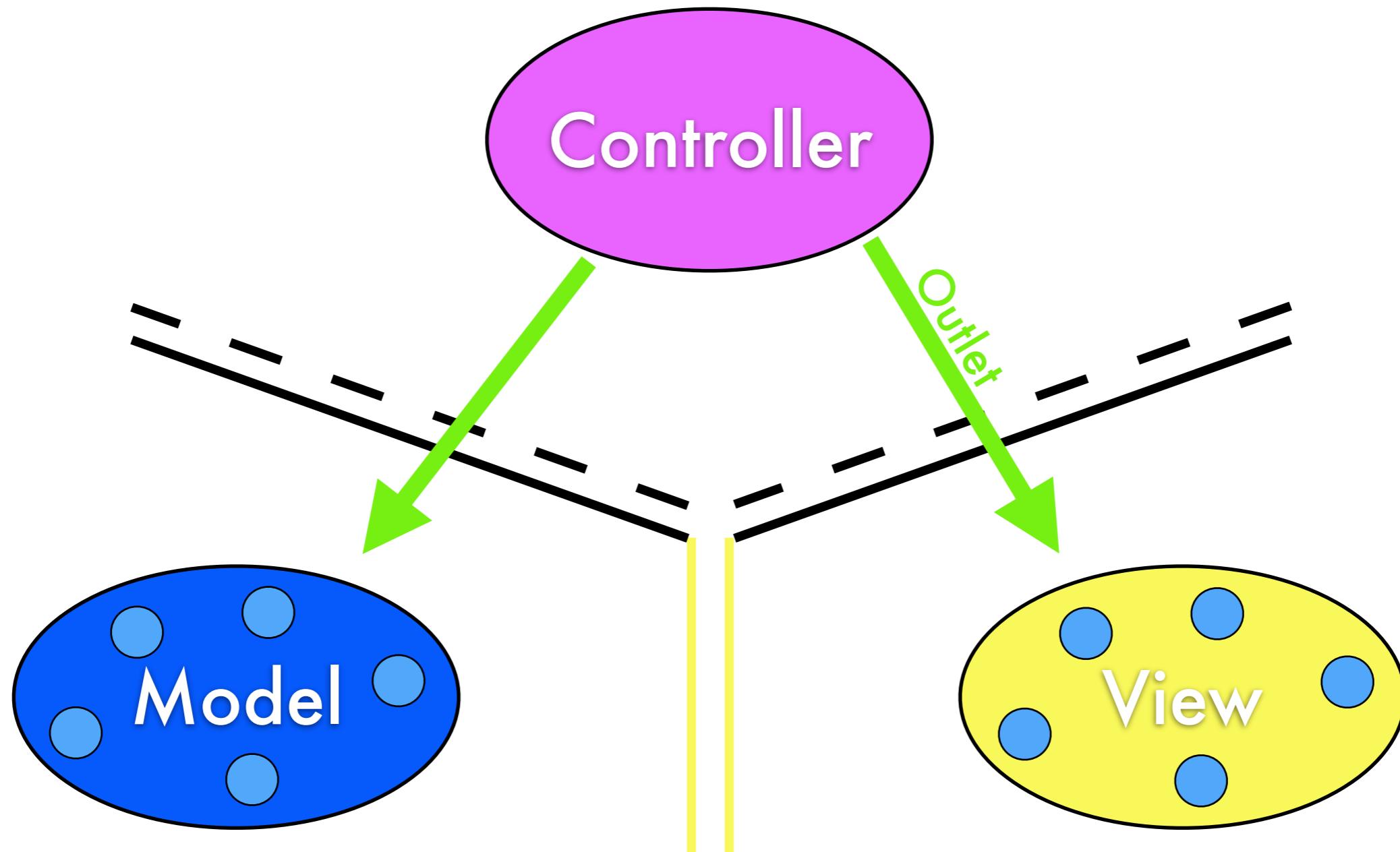
Das MVC Pattern manages die Kommunikation zwischen diesen Rollen

Model View Controller



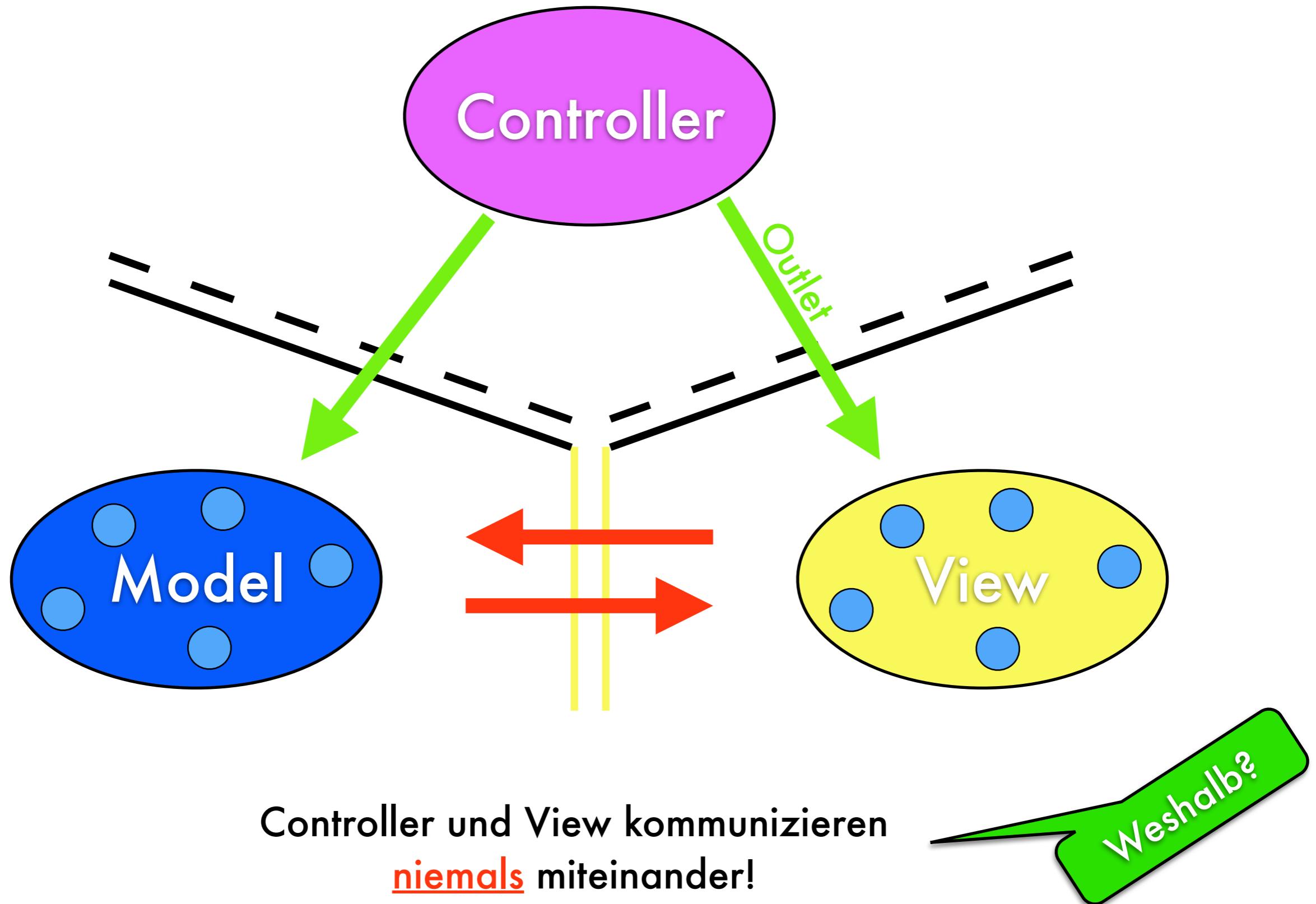
Controller können immer direkt mit Ihrem Model kommunizieren

Model View Controller

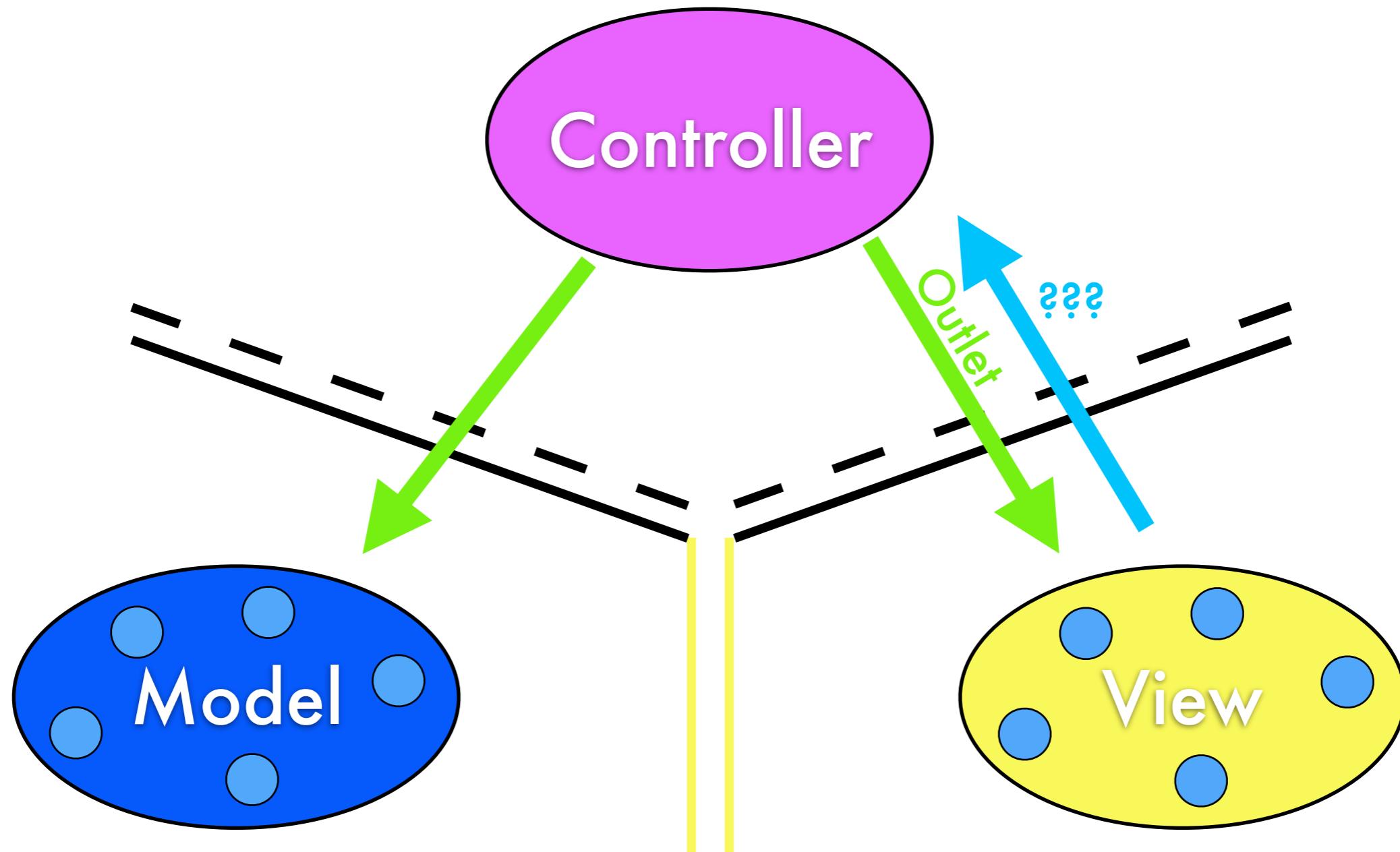


Controller können auch direkt mit ihrem View kommunizieren

Model View Controller

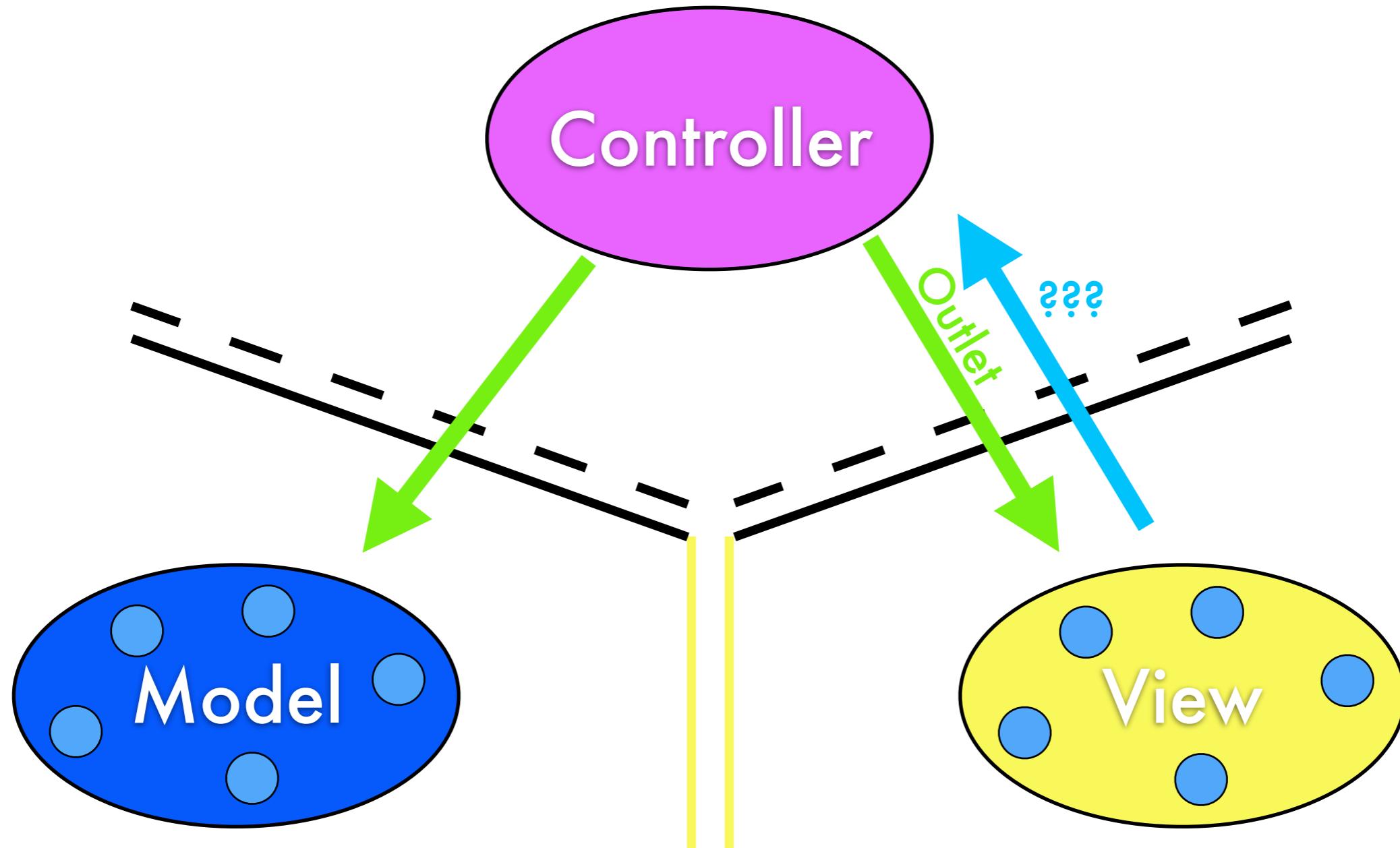


Model View Controller



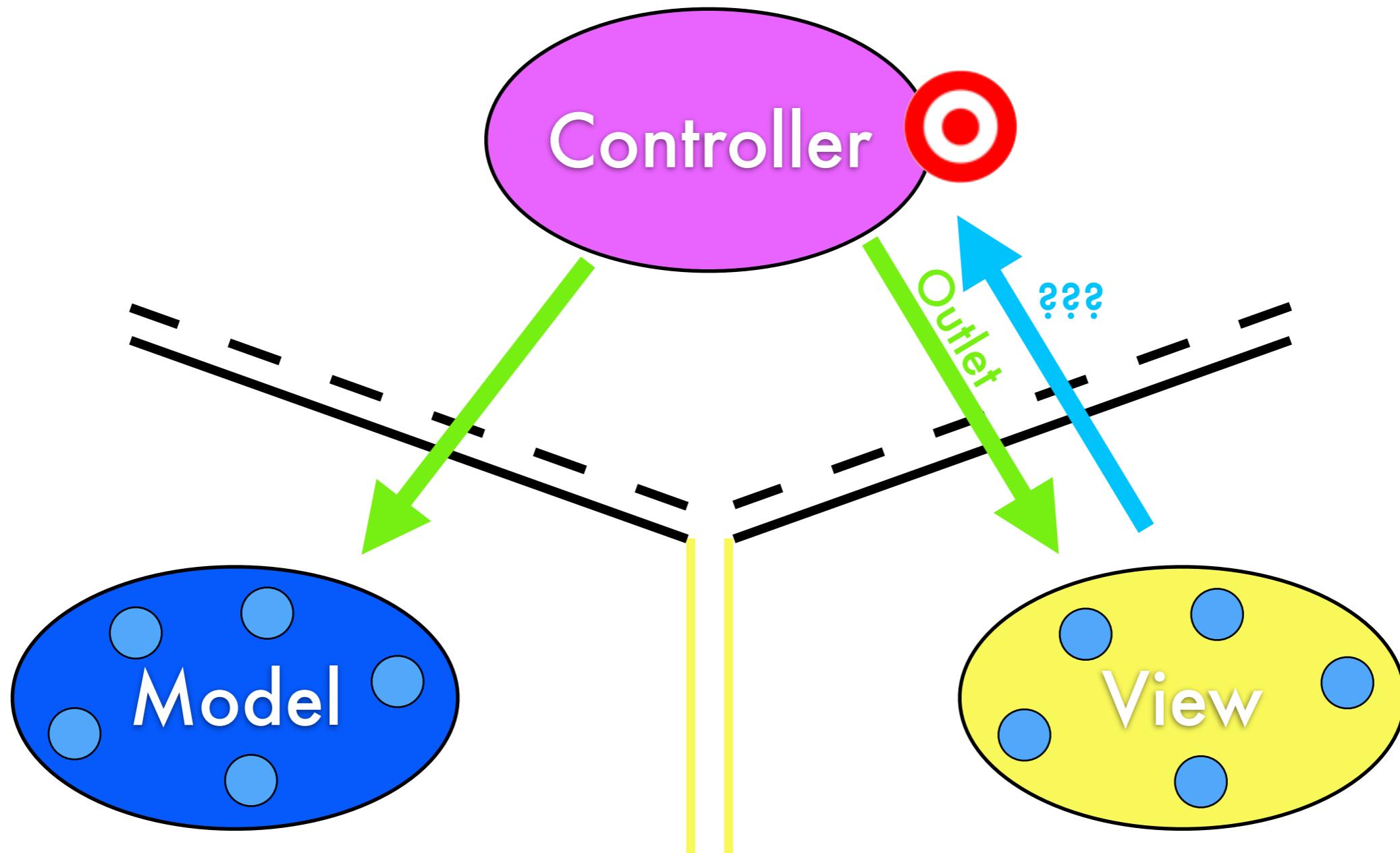
Kann der View mit dem Controller kommunizieren?

Model View Controller



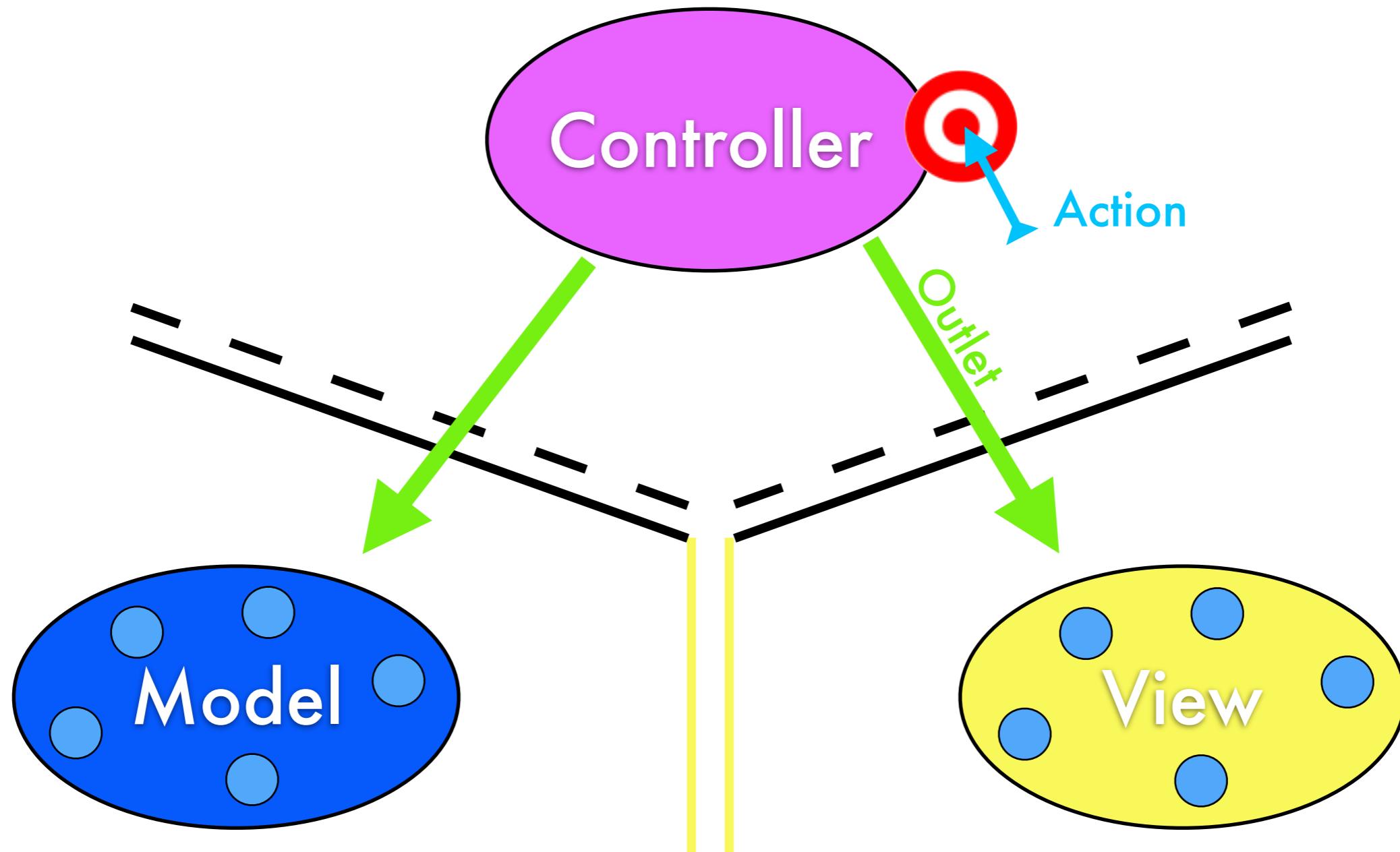
Ja, irgendwie - die Kommunikation ist aber "blind" und strukturiert...
(der View kennt die konkrete Implementierung des Controllers nicht)

Model View Controller



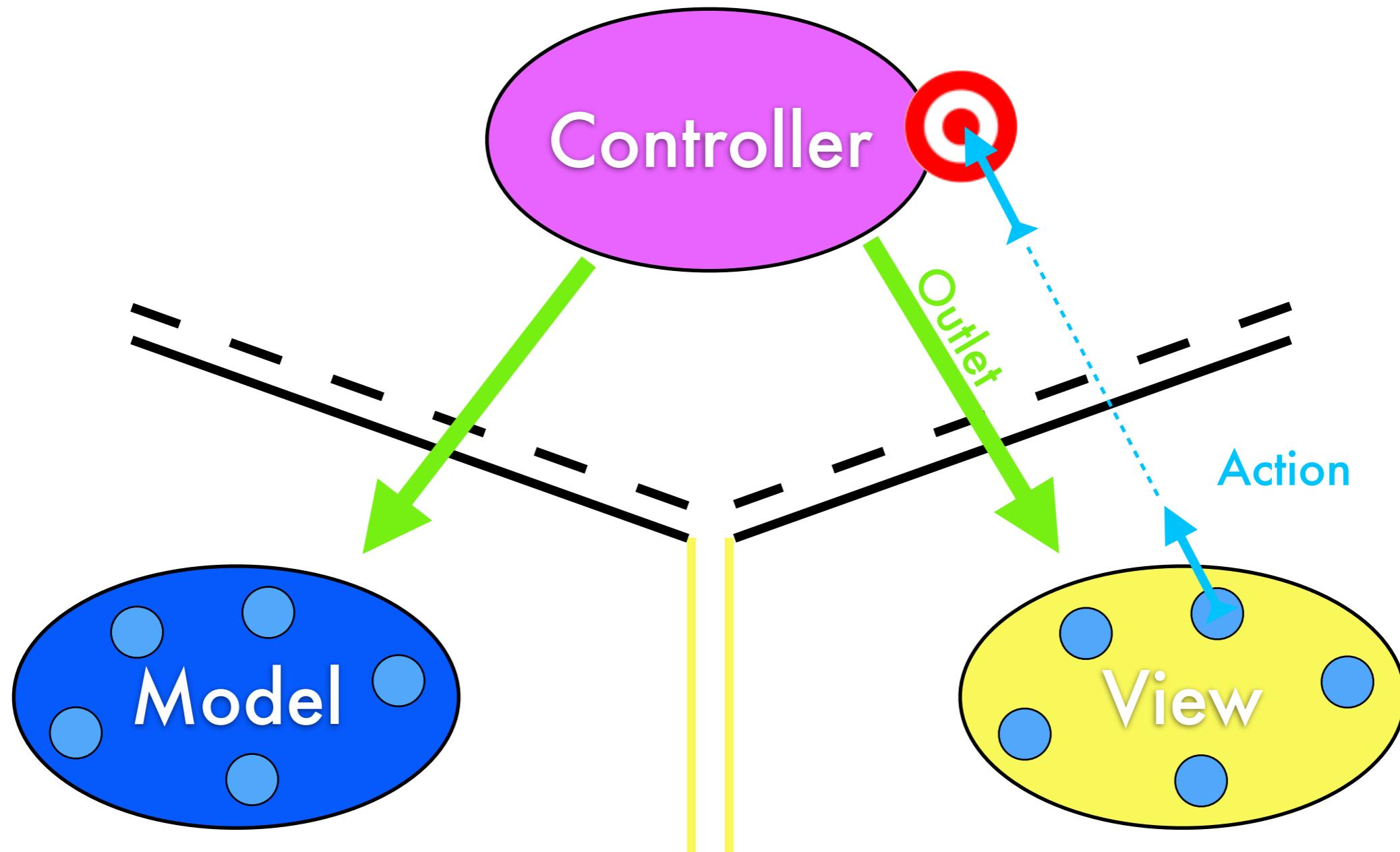
Der Controller stellt ein **Target** zur Verfügung...

Model View Controller



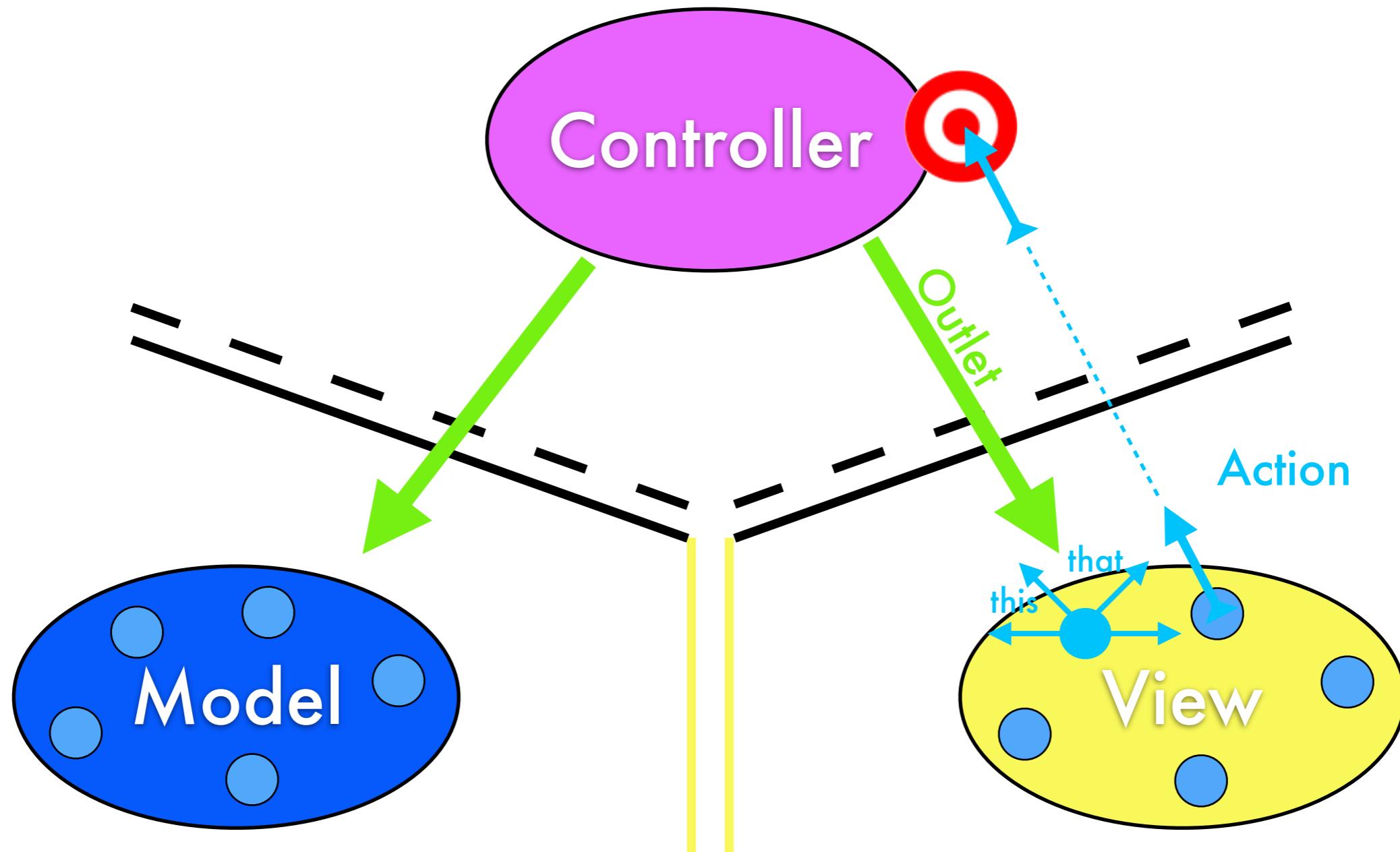
... und dem View eine **Action** bereit

Model View Controller



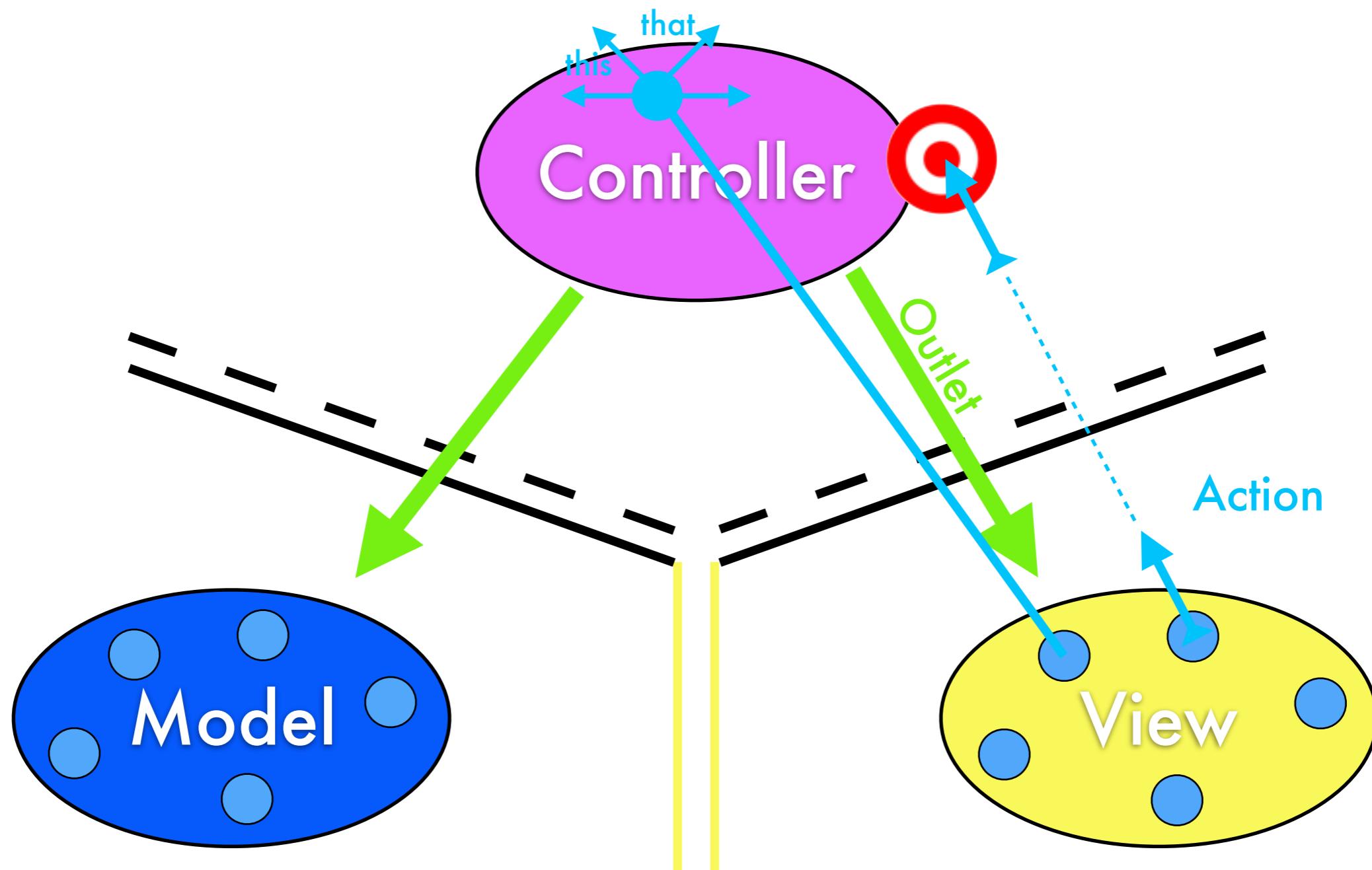
Wenn im UI ein Ereignis auftritt, schickt der View die **Action** an den Controller

Model View Controller



Manchmal muss der View sich mit dem Controller synchronisieren

Model View Controller

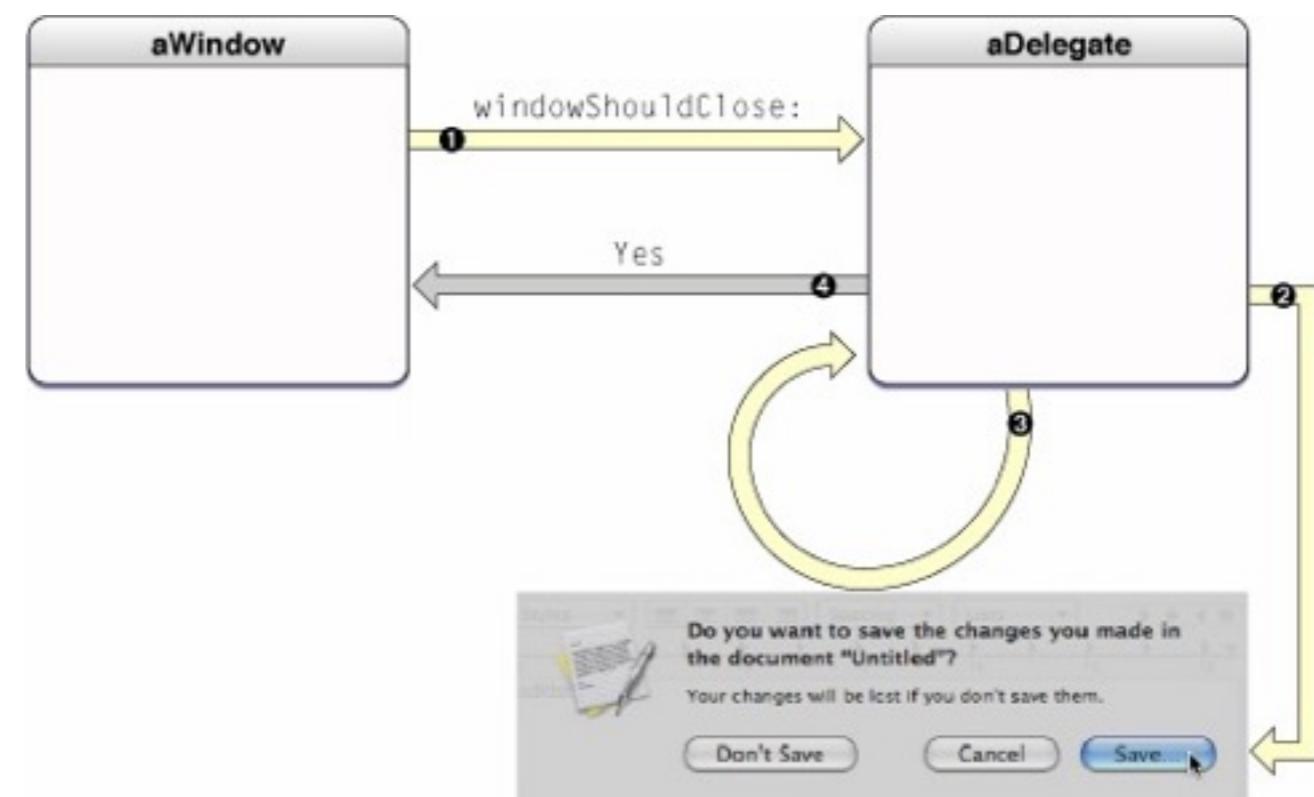
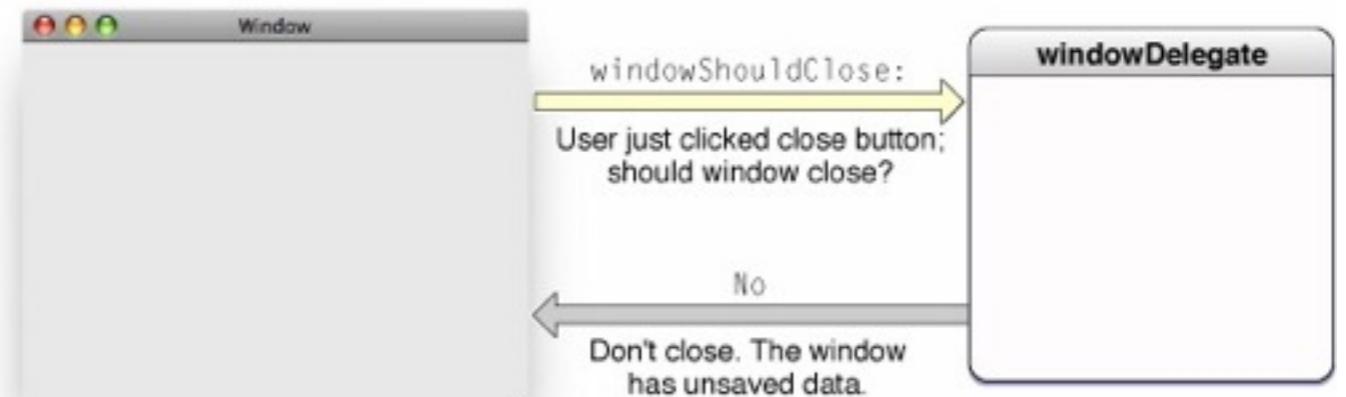


Der Controller setzt sich selbst als **Delegate** für den View

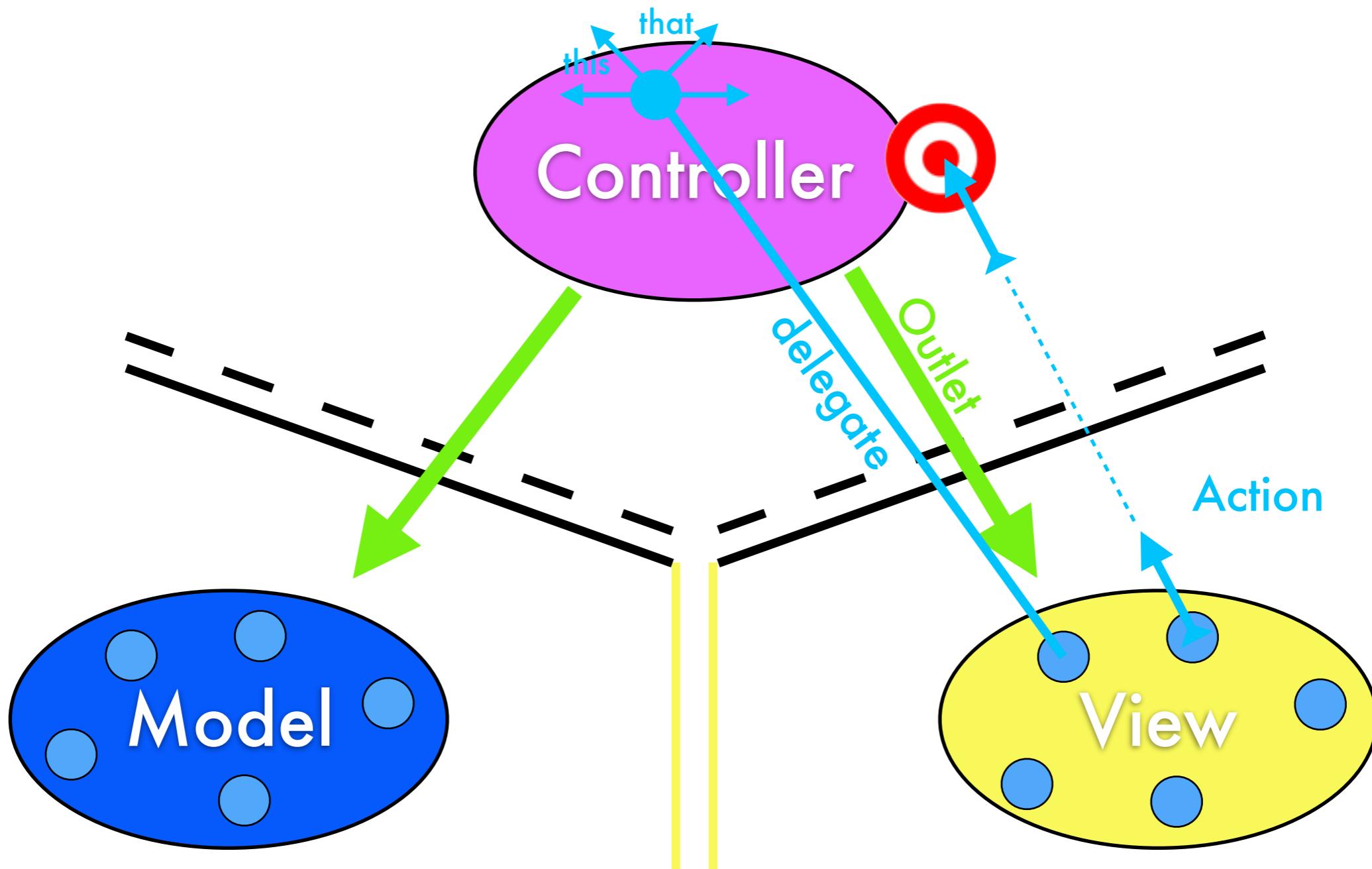
Was ist ein Delegate?

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.

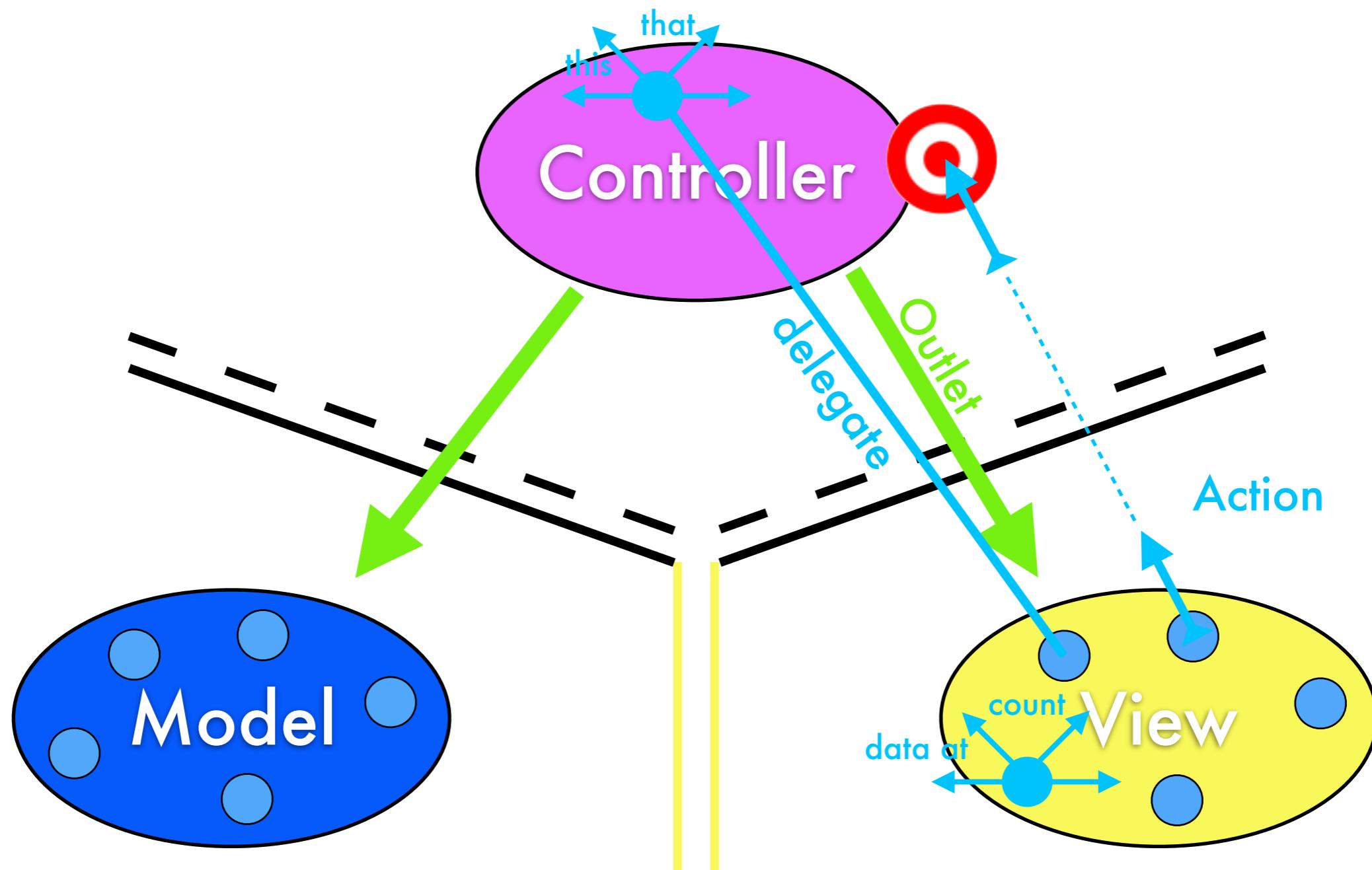


Model View Controller



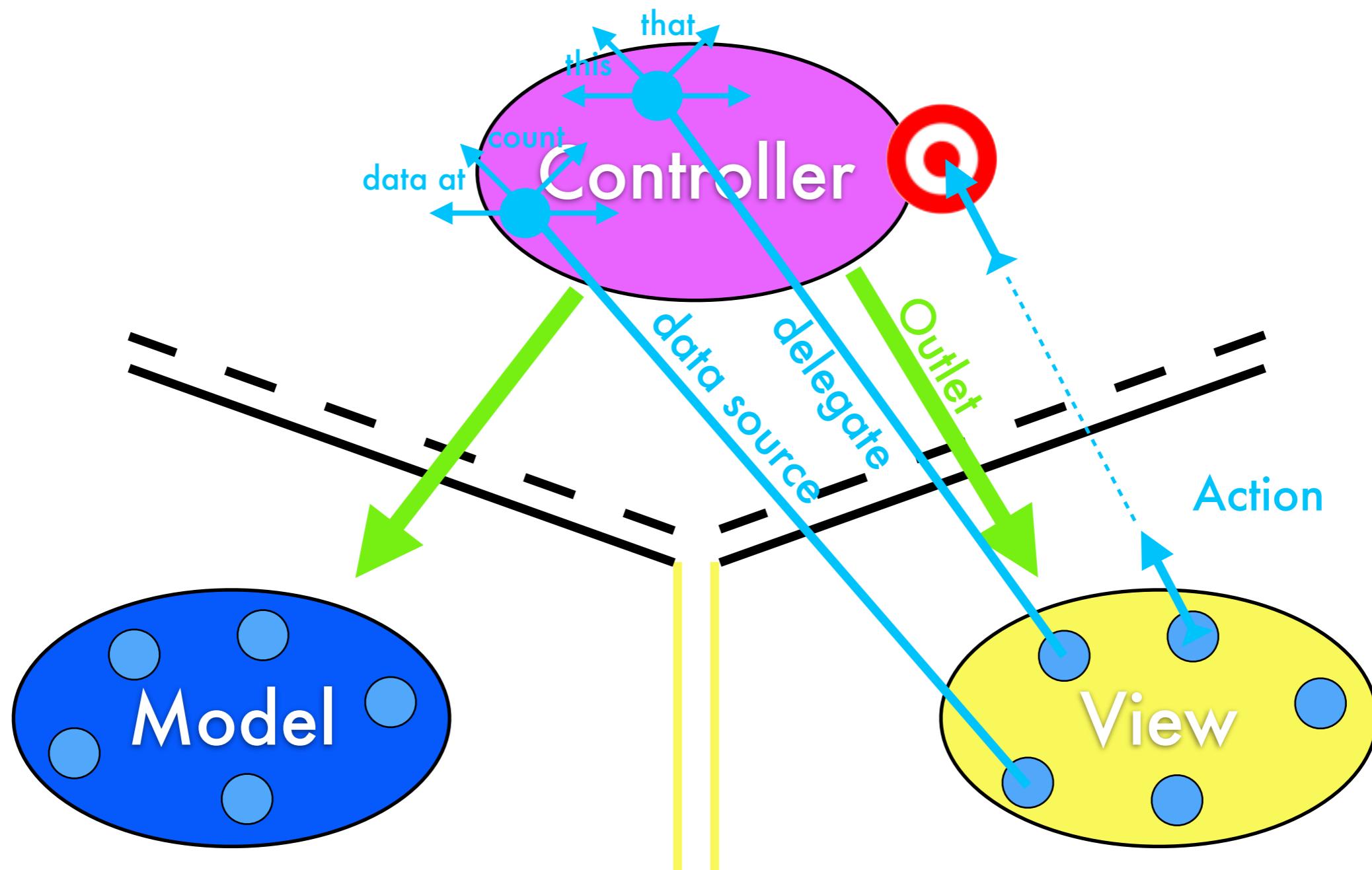
Das Delegate wird per Protocol gesetzt ("blind" gegenüber der Klasse)
Wiederholung: Views besitzen die Daten nicht, die sie anzeigen

Model View Controller



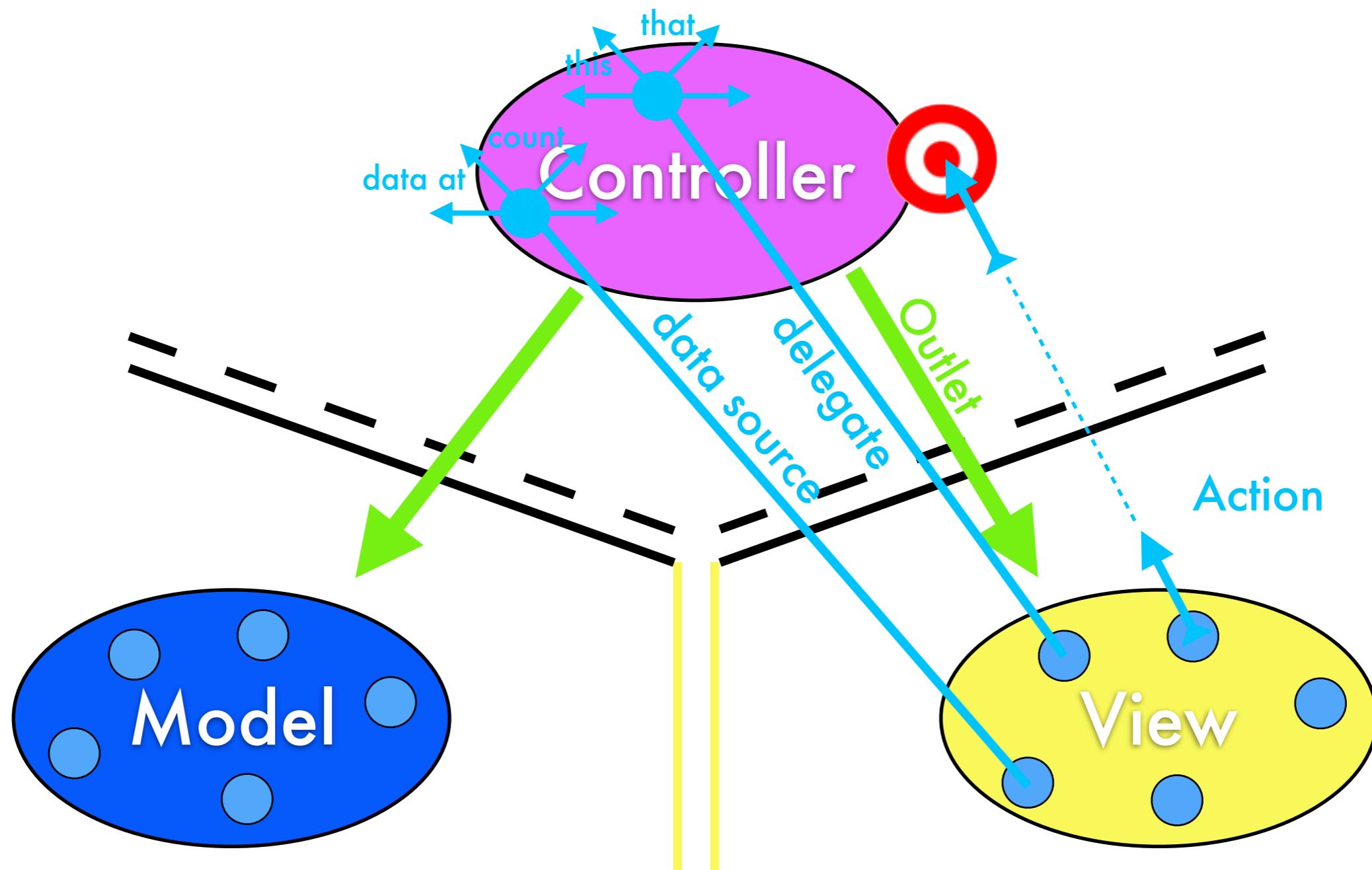
Sofern benötigt, haben sie ein Protocol um Daten zu erhalten

Model View Controller



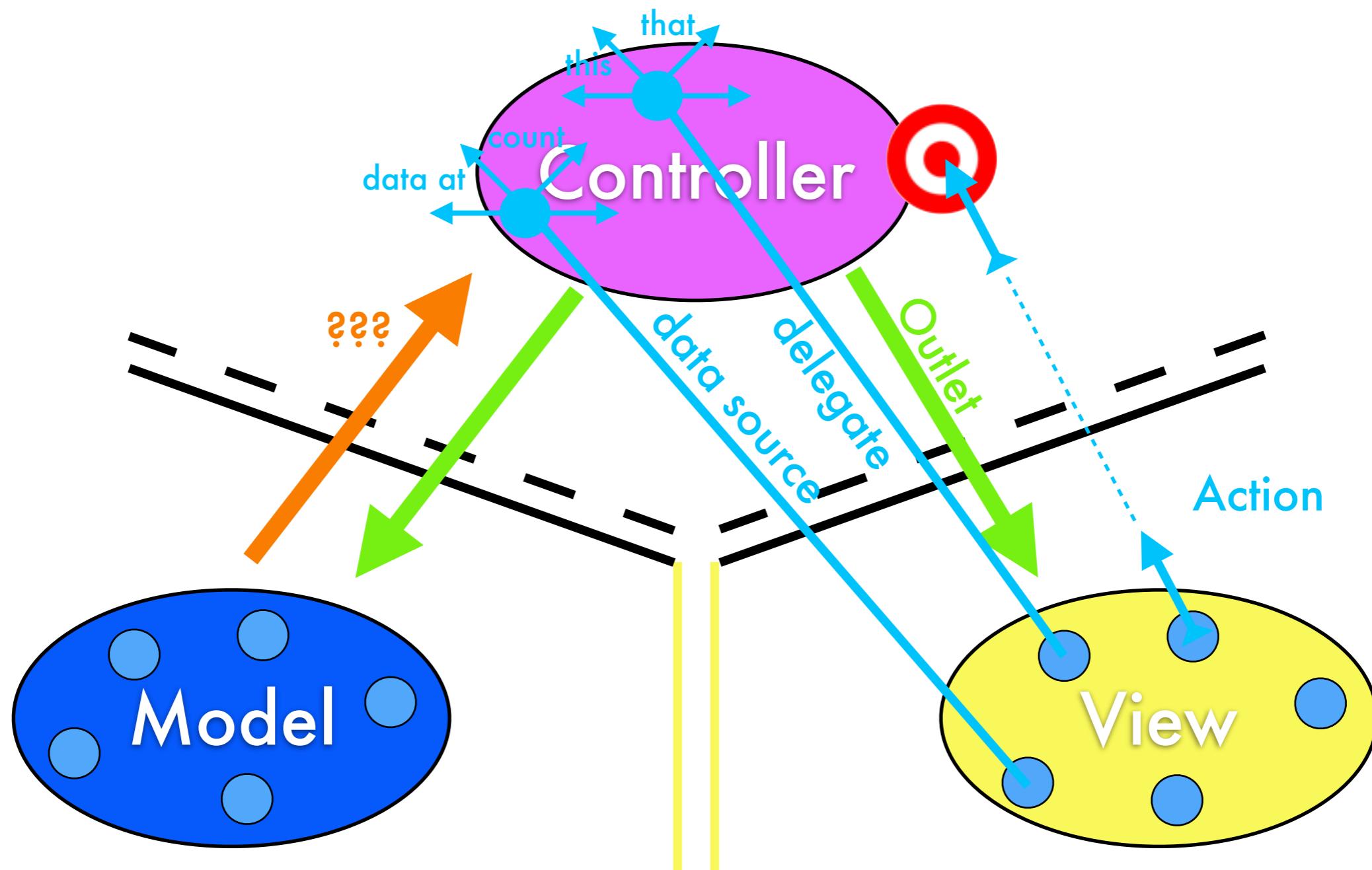
Normalerweise sind die Controller die Datenquelle (nicht das Model)

Model View Controller



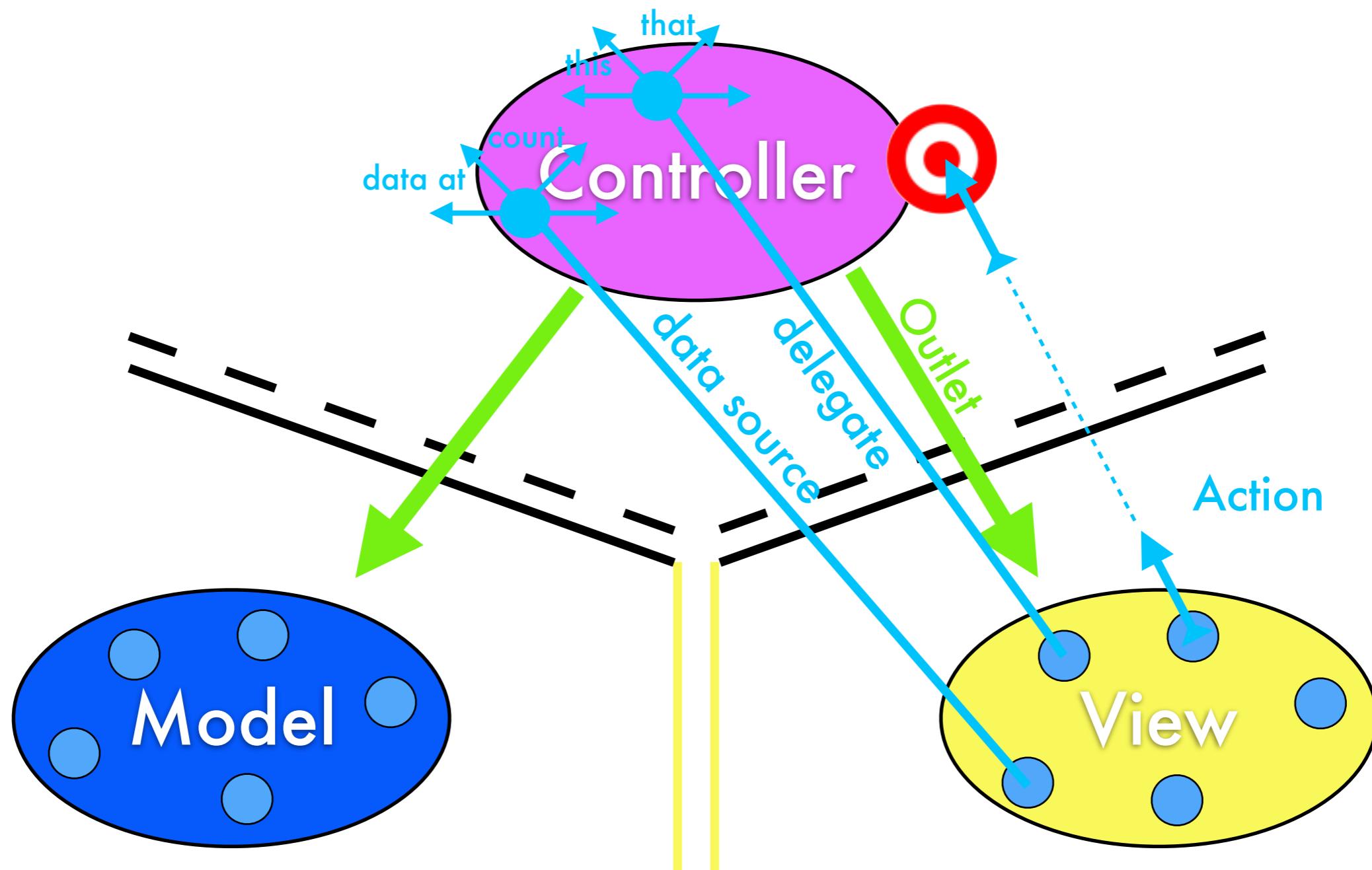
Controller formatieren/interpretieren die Model Information für den View

Model View Controller



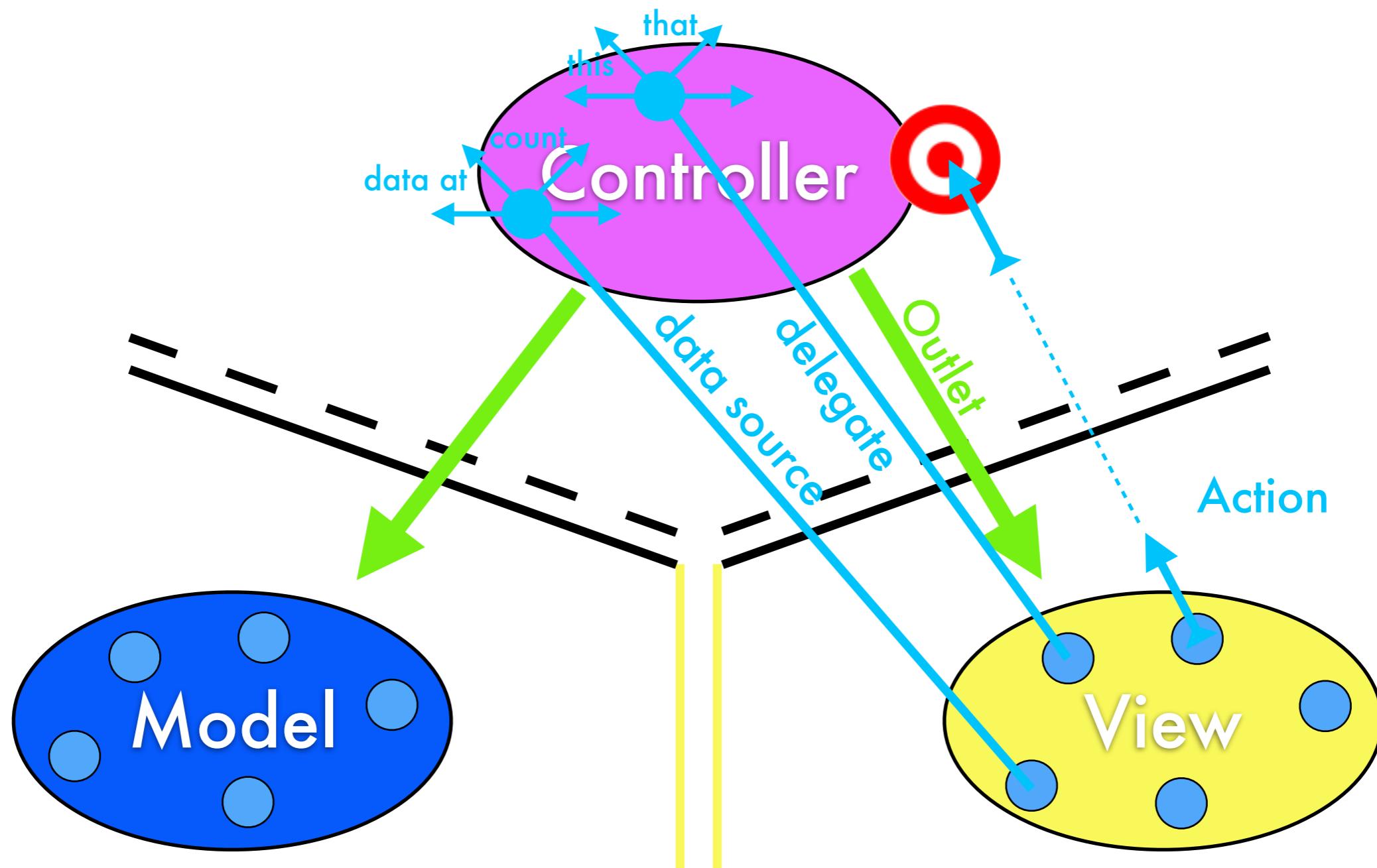
Kann das Model direkt mit dem Controller kommunizieren?

Model View Controller



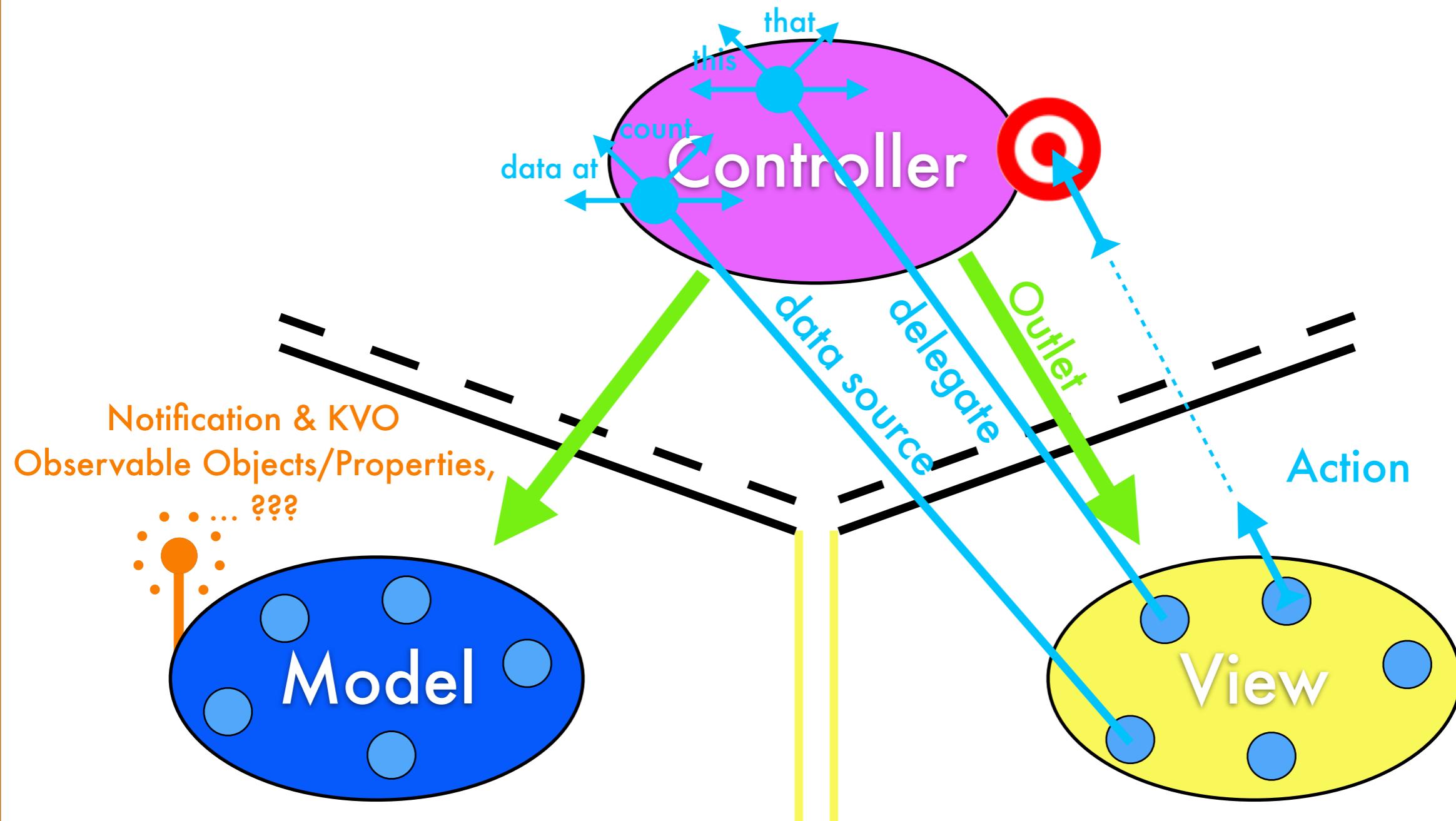
Nein! Das Model sollte unabhängig vom UI sein

Model View Controller



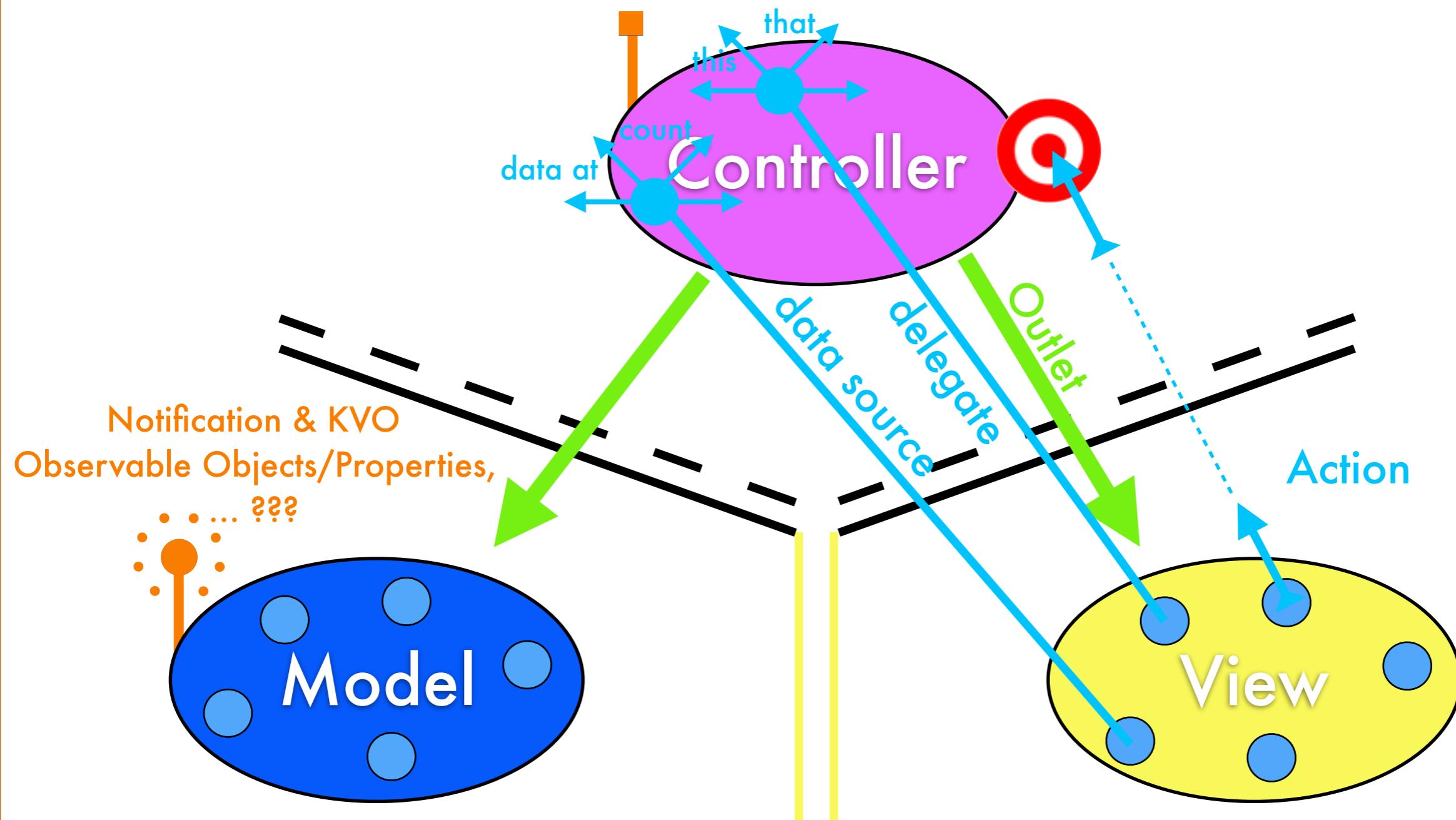
Was passiert also, wenn eine Information geupdated werden soll?

Model View Controller



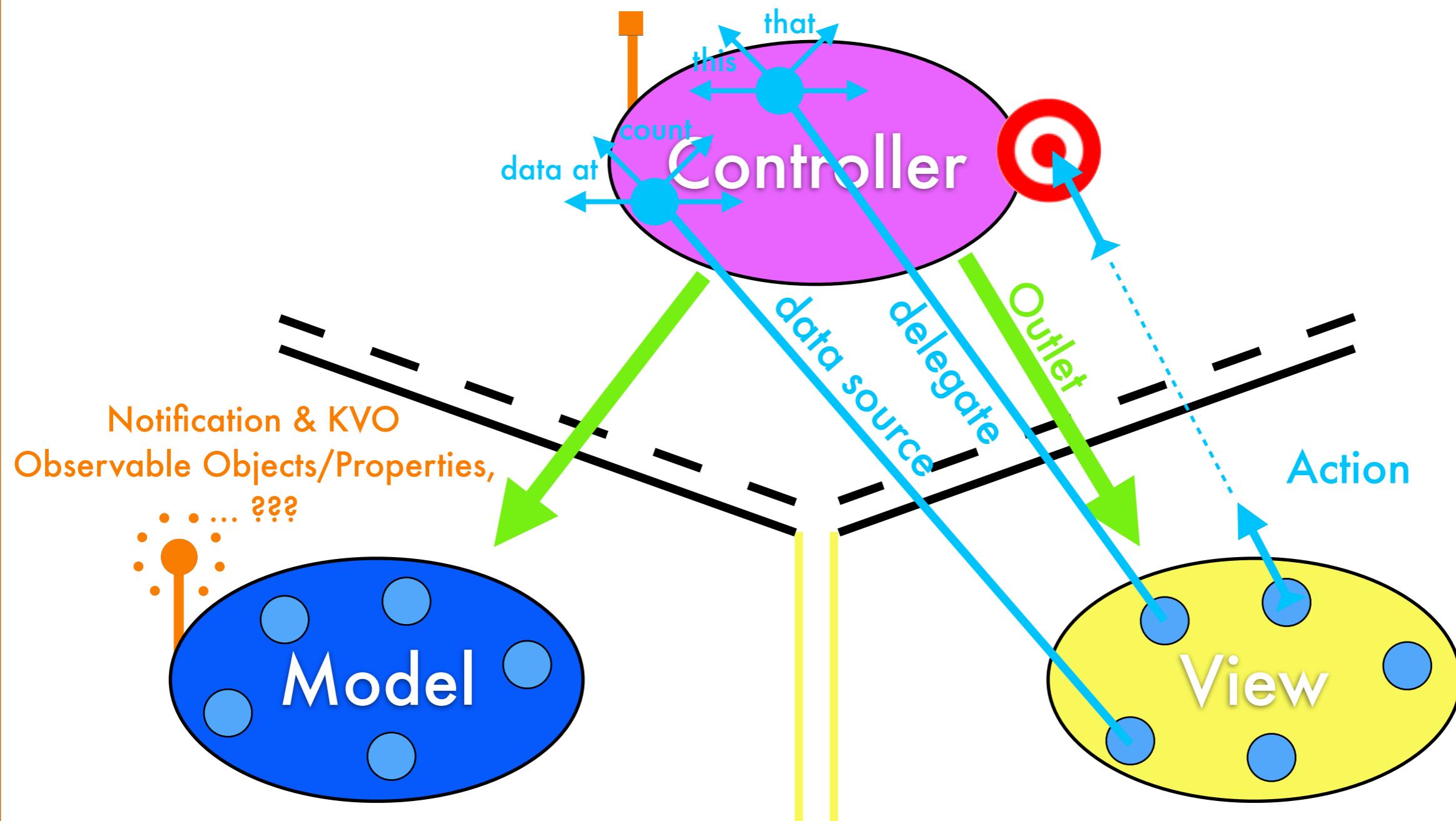
Ein Broadcast Mechanismus wird verwendet (wie ein Radio-Sender)

Model View Controller



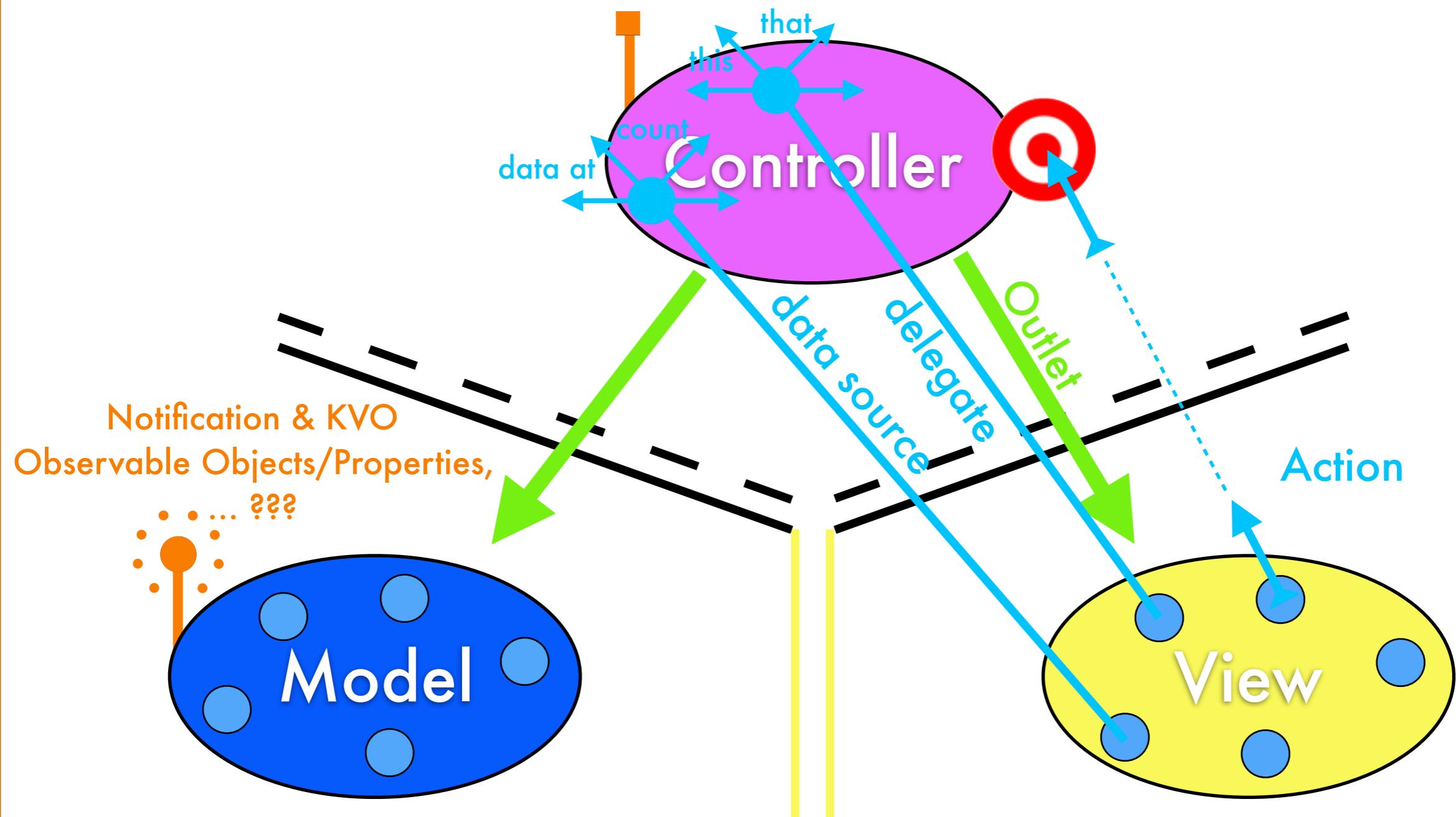
Controller (oder andere Models) stellen den Empfänger für interessante Dinge ein

Model View Controller



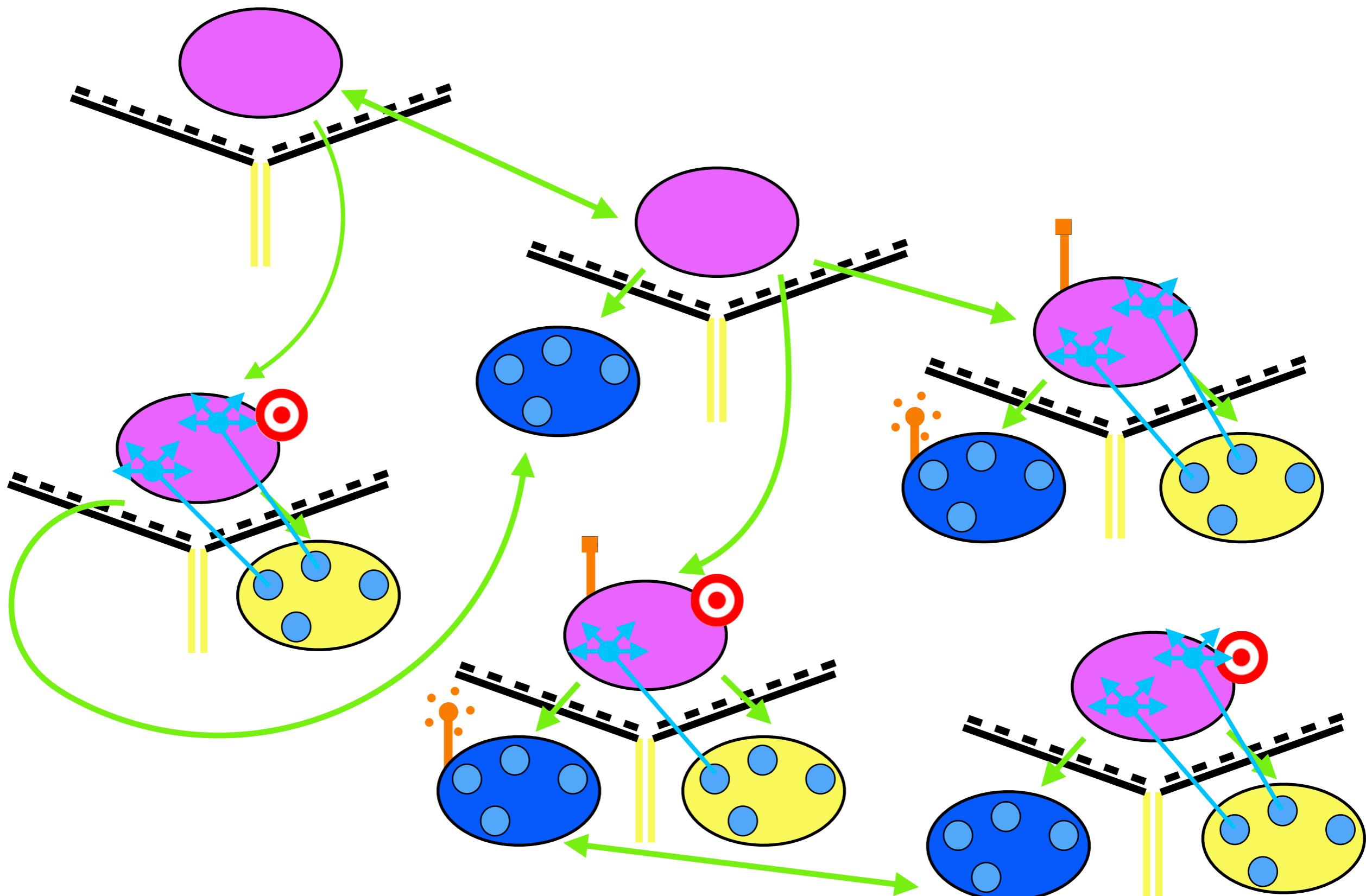
Ein View könnte “zuhören”, aber wahrscheinlich nicht einem Model-Sender

Model View Controller

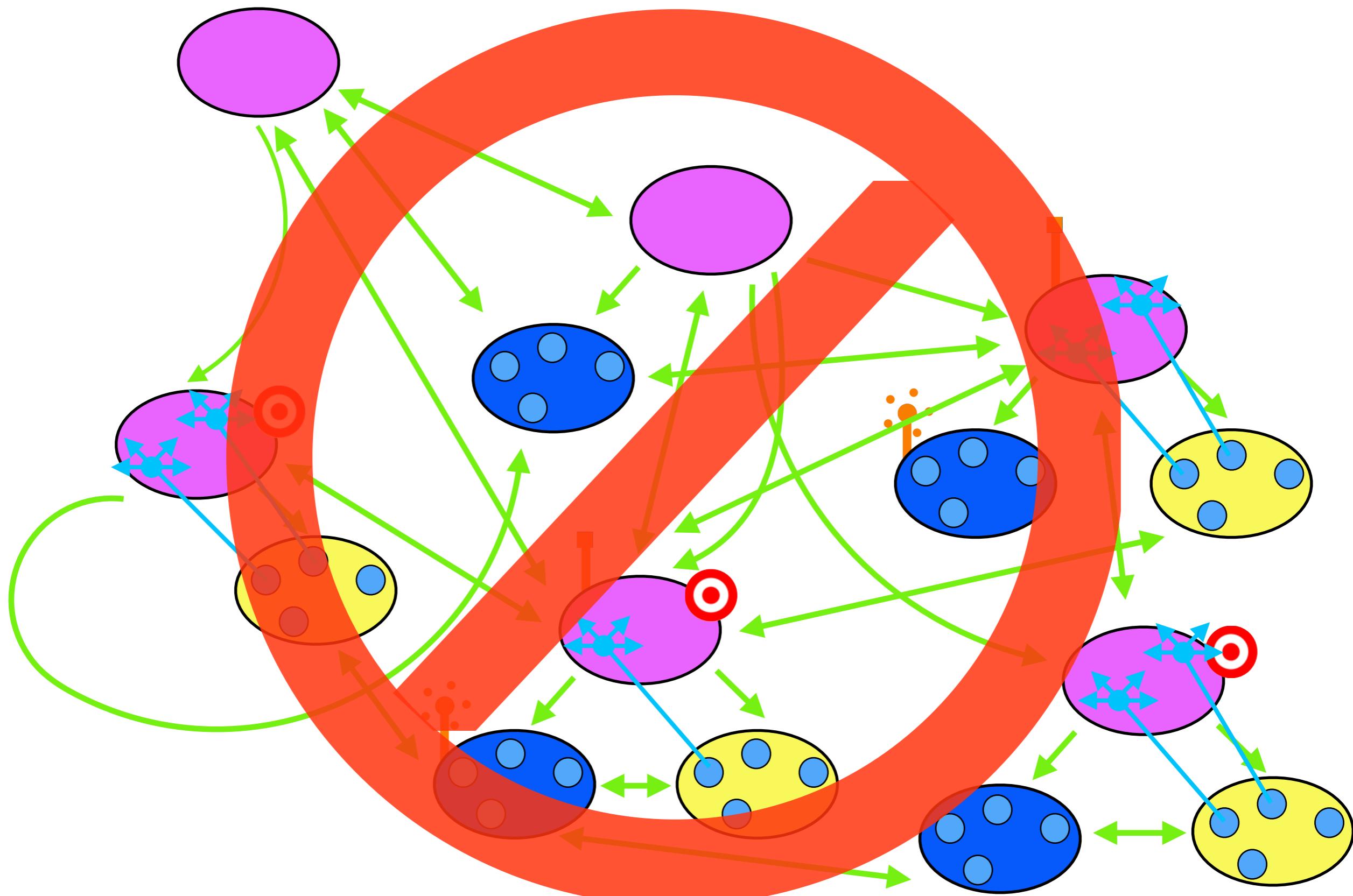


MVC Gruppen können kombiniert werden, um komplizierte Apps zu erstellen ...

MVCs arbeiten zusammen



MVCs die nicht zusammen arbeiten



Let's get started



Let's get started

Classes:



.xcodeproj

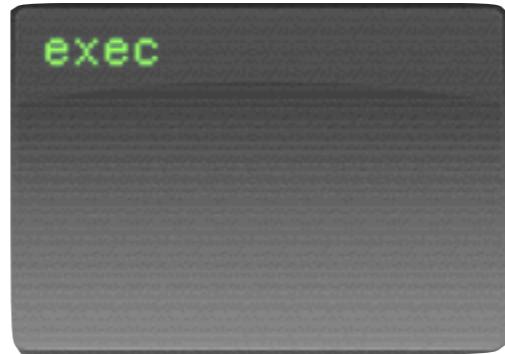
Resources:



Let's get started



.xcodeproj



Resources:



Let's get started



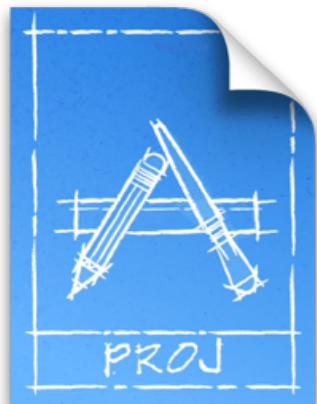
.xcodeproj



Resources:



Let's get started



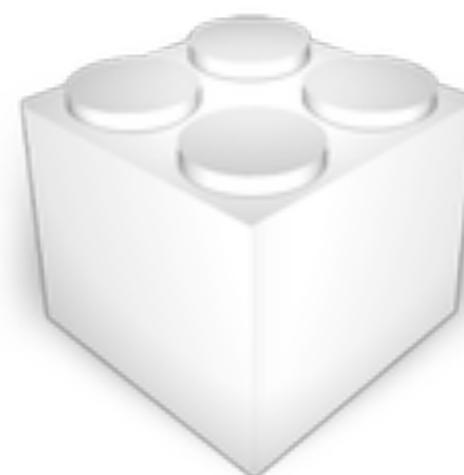
.xcodeproj



Application



Library



Bundle

Let's get started

⚙ Projects

Compile Source Code

Build and Code Sign

Vordefinierte Products (Debug, Release, Test, Profile, Analyze)

Output ist Application, Library oder Bundle

⚙ Weitere Möglichkeiten

Verwendung und Management von Workspaces, Configuration Files, usw.

Behandeln wir aber nicht

Let's get started

DEMO TIME

