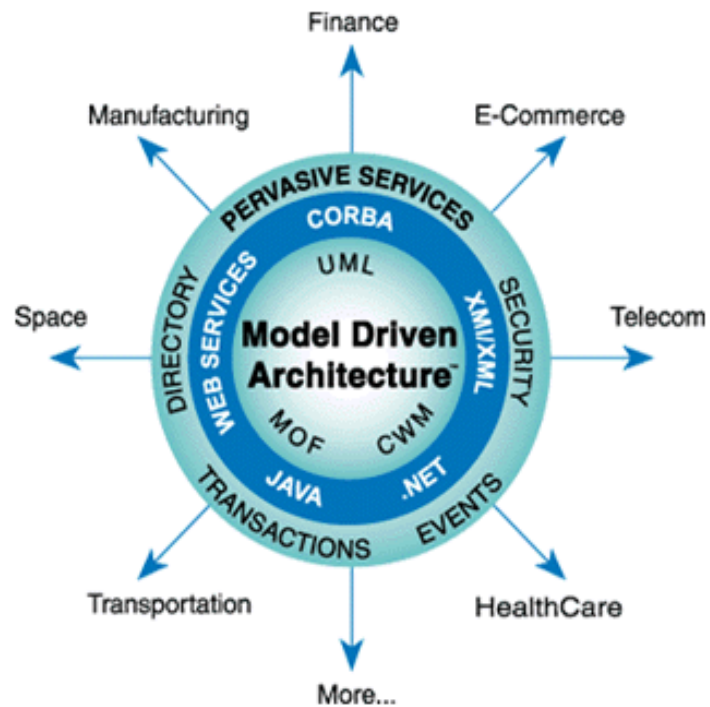


# Model Driven Architecture



“The Architecture of Choice for a Changing World“

# Inhalt

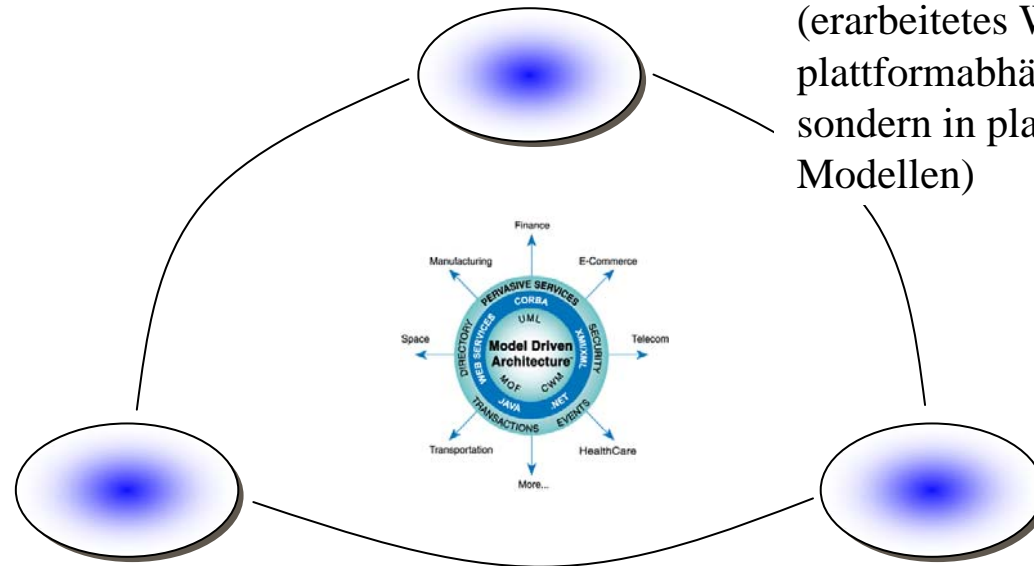


- Warum MDA ?
  - Über Produktivität, Qualität und Schutz von Investitionen
- Was bedeutet der Einsatz von MDA ?
  - Über Paradigmenwechsel und neue Denkweisen
- Was ist MDA ?
  - Entwicklung von Softwaredevelopmentansätzen in den letzten Jahren/Jahrzehnten
  - Abstraktionen als Schlüssel zur Bewältigung zunehmender Komplexität
  - Modelle ersetzen Code als zentrale Entwicklungsartefakte
- Grundbegriffe von MDA ?
  - Modelle, Transformationen und mehr ...
  
- F & A

# Model Driven Architecture Warum ?

## Investitionssicherung

(erarbeitetes Wissen steckt nicht in plattformabhängigem Code - sondern in plattformunabhängigen Modellen)



## Produktivitätserhöhung

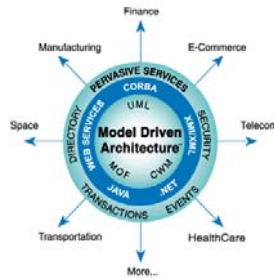
(durch Nutzung höherer Abstraktionsebenen – bereichsspezifische Modellierung, generativen Ansatz)

## Qualitätsverbesserung

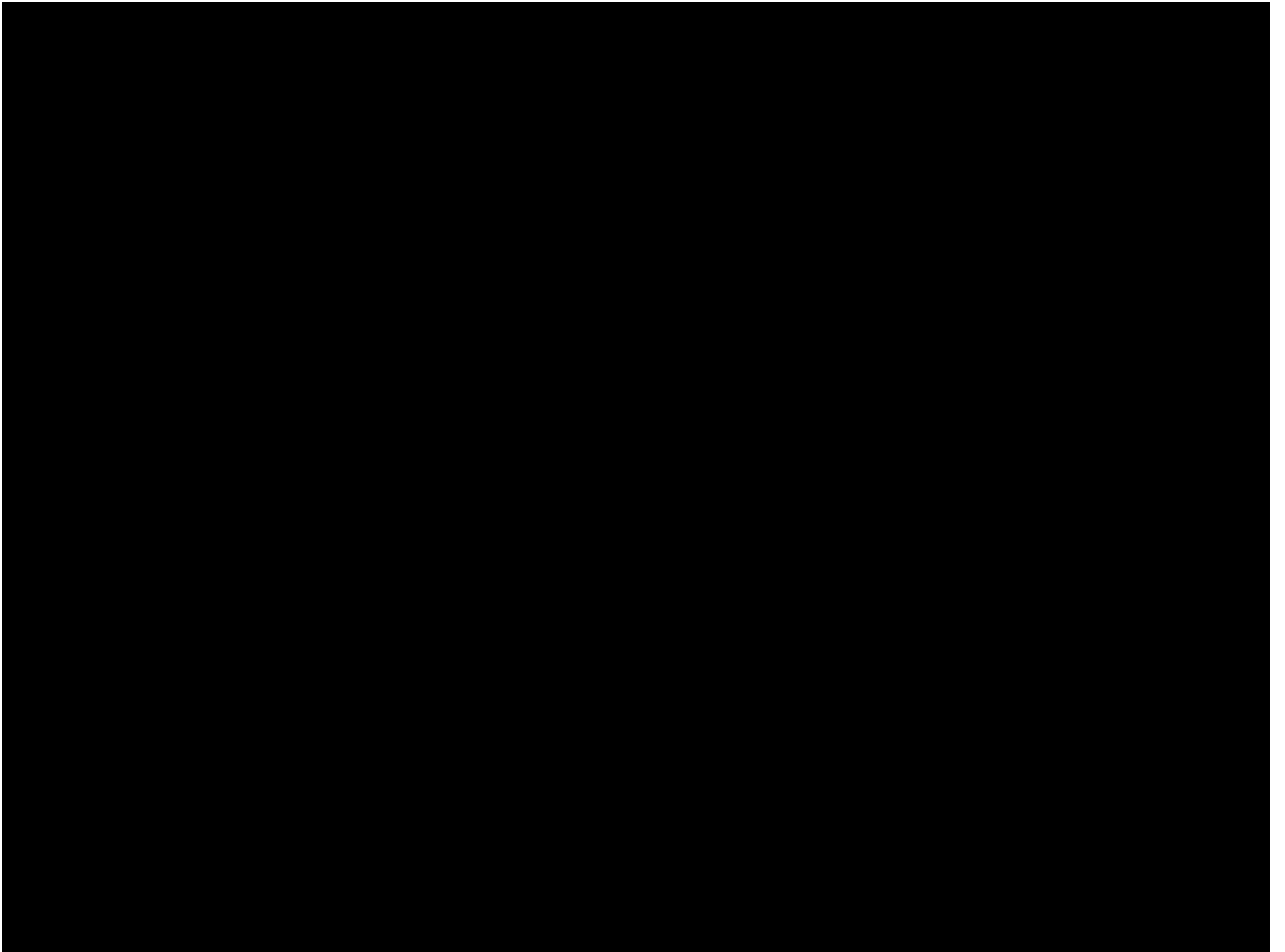
(durch generativen Ansatz – wiederkehrende Muster werden generiert und nicht wiederholt händisch implementiert)

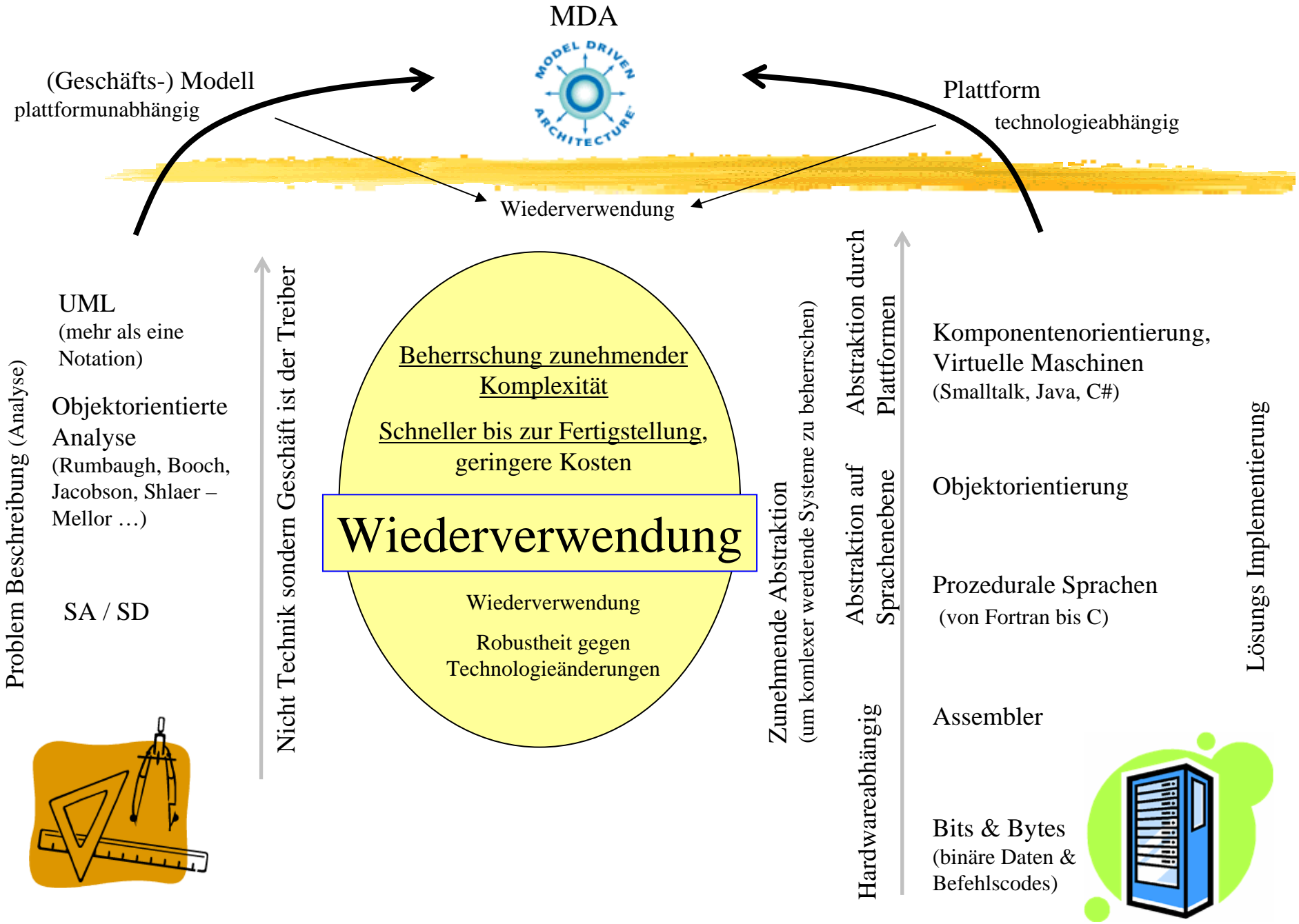


# Model Driven Architecture ist



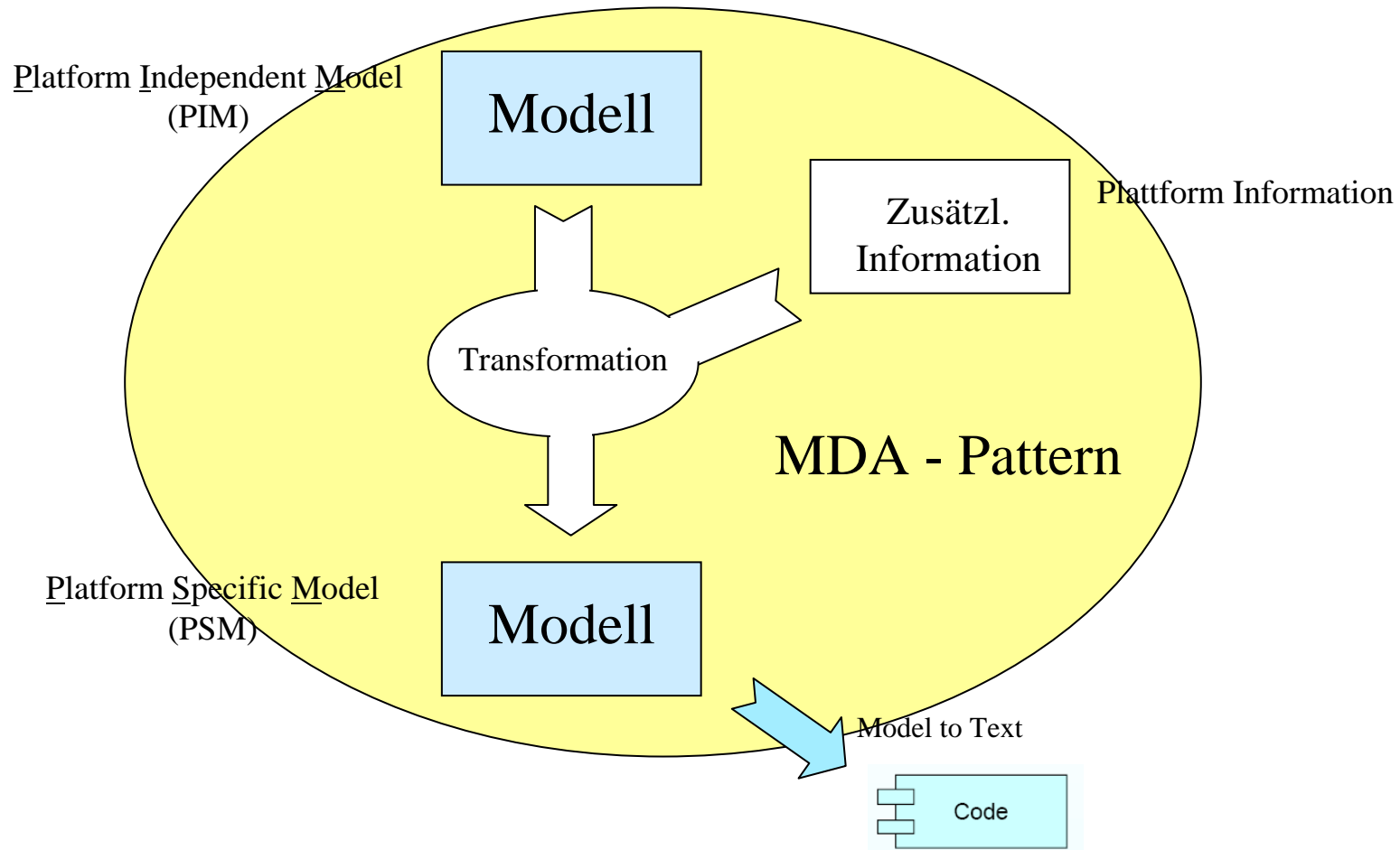
Der logische nächste Schritt, als  
konsequente Weiterentwicklung im  
Bereich des Software-Engineering






# Model Driven Architecture

## Es geht um Modelle und Transformationen



# Model Driven Architecture

## Entwicklungsschritte im SE sind geprägt durch

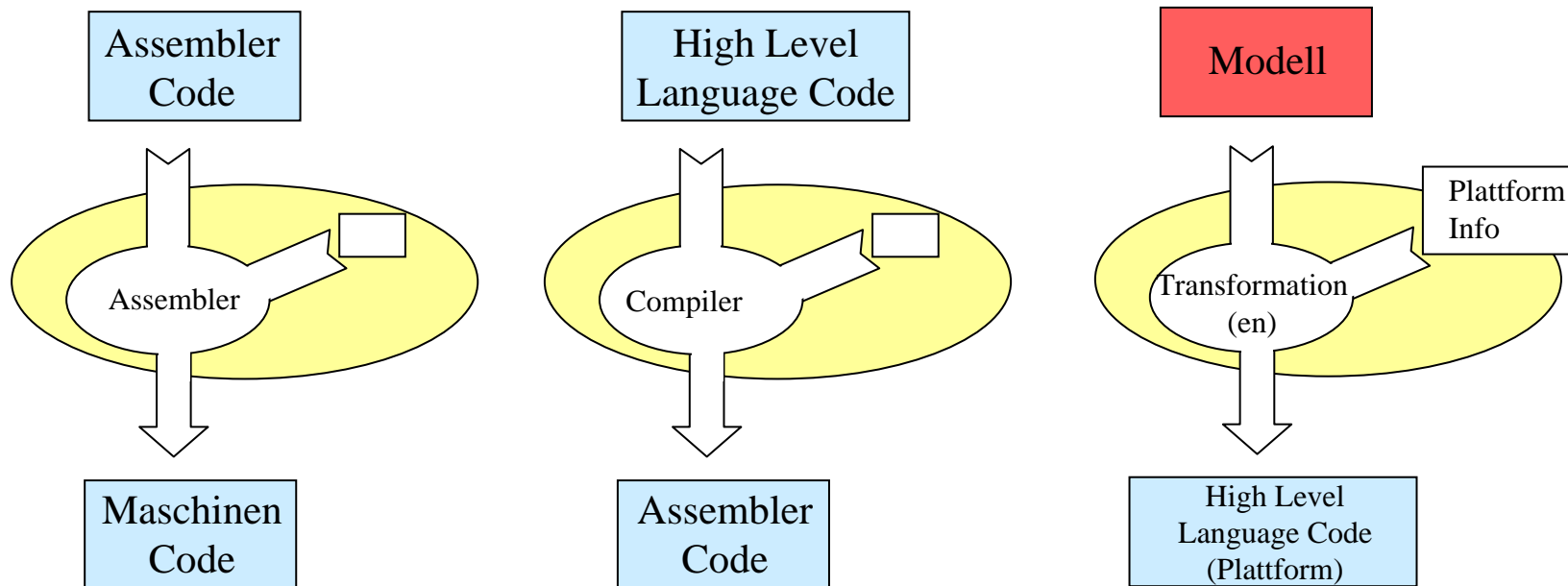


➤ Anhebung des Abstraktions-Niveaus

➤ Erhöhung der Granularität der Wiederverwendung

# Model Driven Architecture

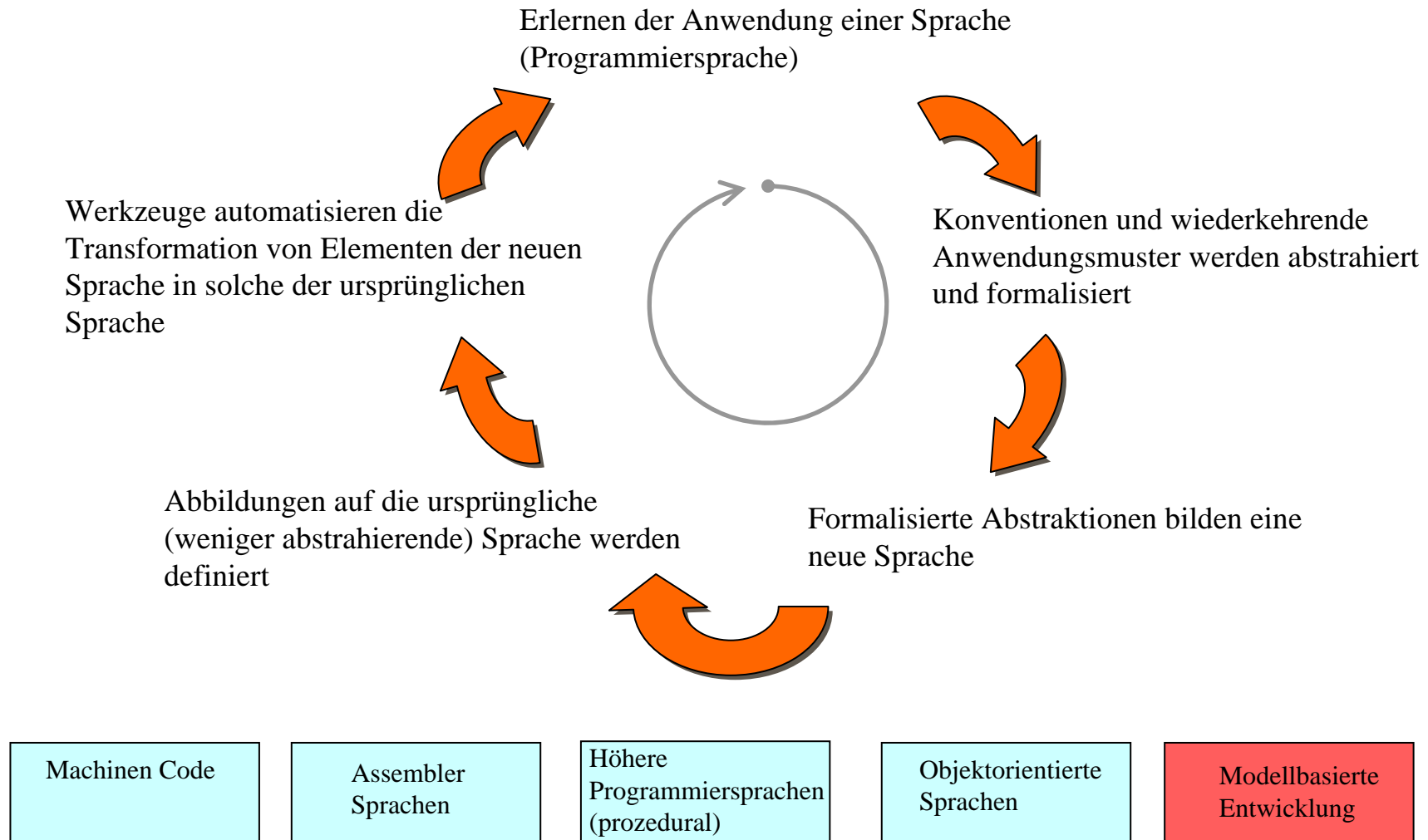
## Anhebung des Abstraktions-Niveaus



Die Geschichte der Softwareentwicklung ist geprägt durch die wiederholte Anhebung des Abstraktions-Niveaus

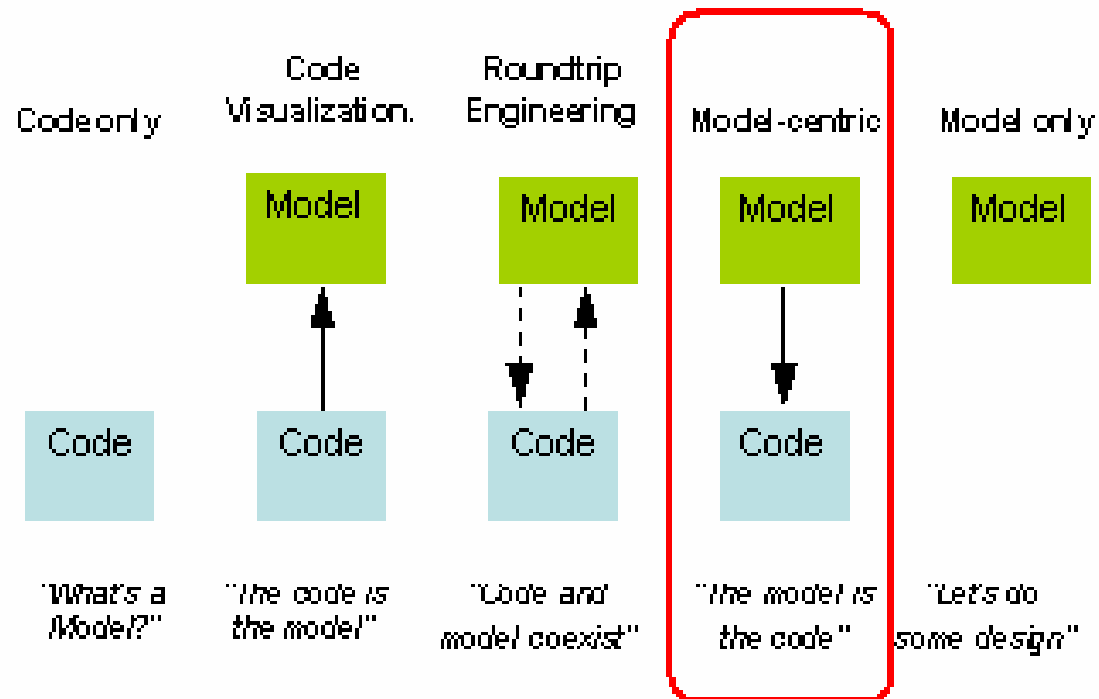
# Model Driven Architecture

## Anhebung des Abstraktions-Niveaus folgt wiederkehrendem Muster



# Model Driven Architecture

## Entwicklungsschritte im Software-Engineering



# Model Driven Architecture

## Entwicklungsschritte im SE sind geprägt durch

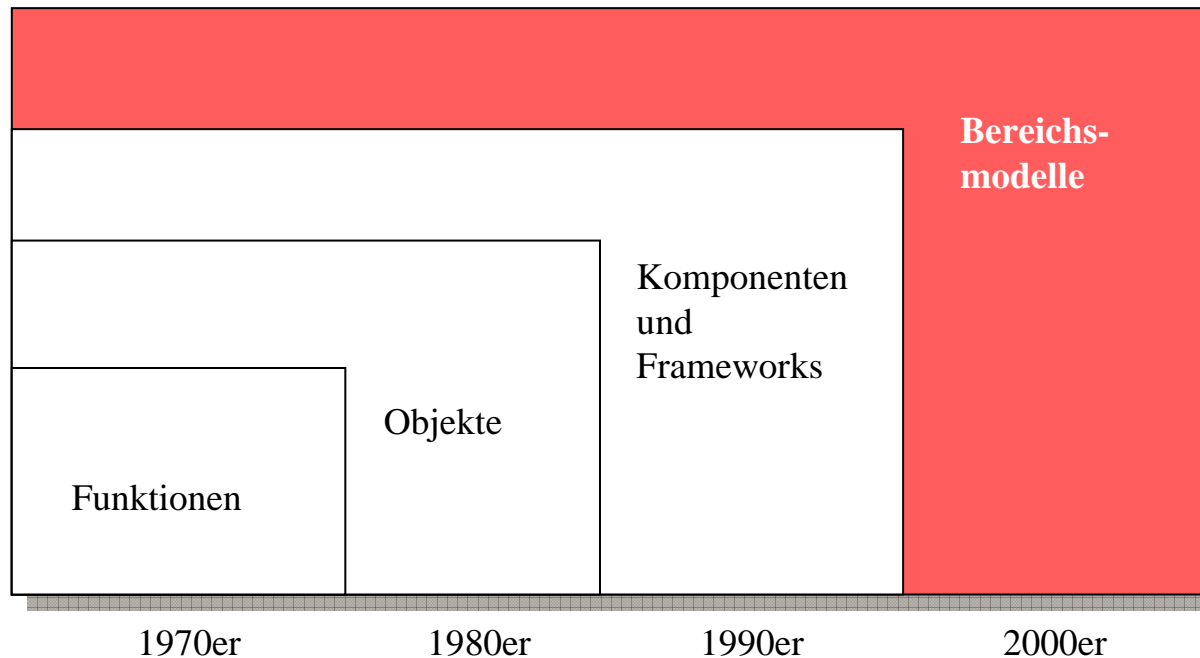


➤ Anhebung des Abstraktions-Niveaus

➤ Erhöhung der Granularität der Wiederverwendung

# Model Driven Architecture

## Granularität der Wiederverwendung



Objektbasierte Programmierung (Objekt-Orientierung) ist die dominante Paradigma der Wiederverwendung (Geschäftsmodell, Datenbank, Berechtigungen, etc.) (Regelbasierte Wiederverwendung zwischen den Wiederverwendungsgruppen ist immer noch ein Spielzeug).  
 Plug and Play von Komponenten funktioniert nicht wirklich, aufwendiger Verbindungscode Updates müssen berechtigt sein, jede Datenänderungsoperation muß bestätigt werden, (glue-code) muß geschrieben werden, starke Abhängigkeit von Laufzeitumgebung (Plattform) Verbindungscode Maß an Wiederverwendung werden, Plattformabhängigkeit gibt es auf dieser Ebene nicht.

# Model Driven Architecture

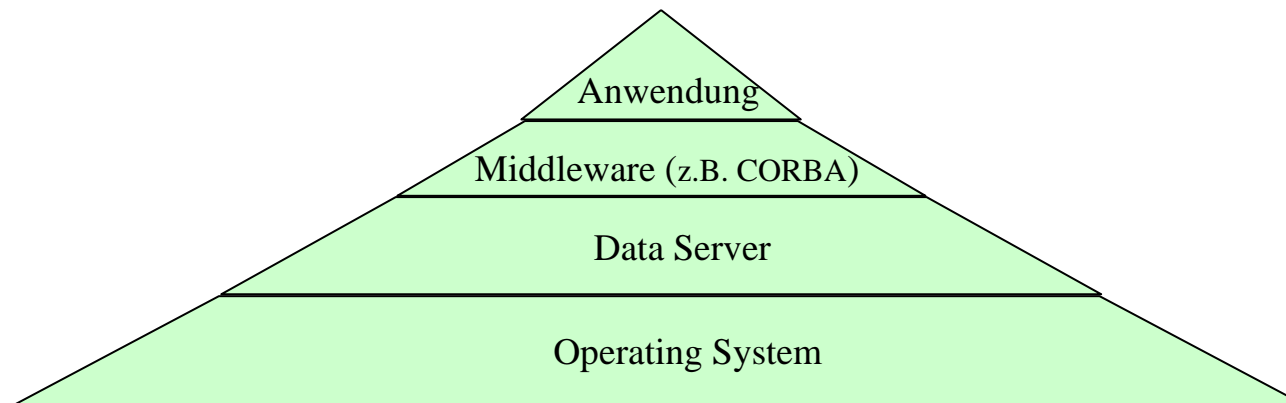
## Wiederverwendung auf der Anwendungsebene

Wiederverwendung auf der Anwendungsebene ?

Laufend wird existierende Funktionalität neu implementiert, um neue Technologien zu nützen.

Viele Systeme können Features neuer Plattformen nicht nützen, da sie zu eng an existierende Anwendungen gekoppelt sind.

Komponenten und Frameworks helfen zwar, aber nach wie vor gibt es deutlich mehr Wiederverwendung auf maschinennahen Ebenen (Datenbanken, Daten-Server), als auf anwendungsnahen Ebenen (dort aber wird viel von Unternehmen investiert).



# Model Driven Architecture

## Wiederverwendung auf der Anwendungsebene



### Woran scheitern wir ?

➤ Die Sprachen, in denen wir Anwendungen entwickeln (Java, C++, ...), sind auf einem zu niederen Abstraktionsniveau, um unterschiedliche Bereiche (Geschäftsmodelle, Prozesse, Berechtigung, Persistenz, Userinterface, ...) effizient zu beschreiben.

➤ Verschiedene Bereiche (Geschäftsmodelle, Prozesse, Berechtigung, Persistenz, Userinterface, ...) sind orthogonal zueinander, im Code, der das lauffähige System beschreibt, aber vielfach miteinander verwoben.

Eine Änderung in einem der Bereiche erfordert meist Änderungen an vielen Stellen in anderen Bereichen.

# Model Driven Architecture

## Wiederverwendung auf der Anwendungsebene



### Was brauchen wir ?

➤ Eine Basis um Sprachen (auch graphische Sprachen) für Bereichsmodelle definieren zu können.

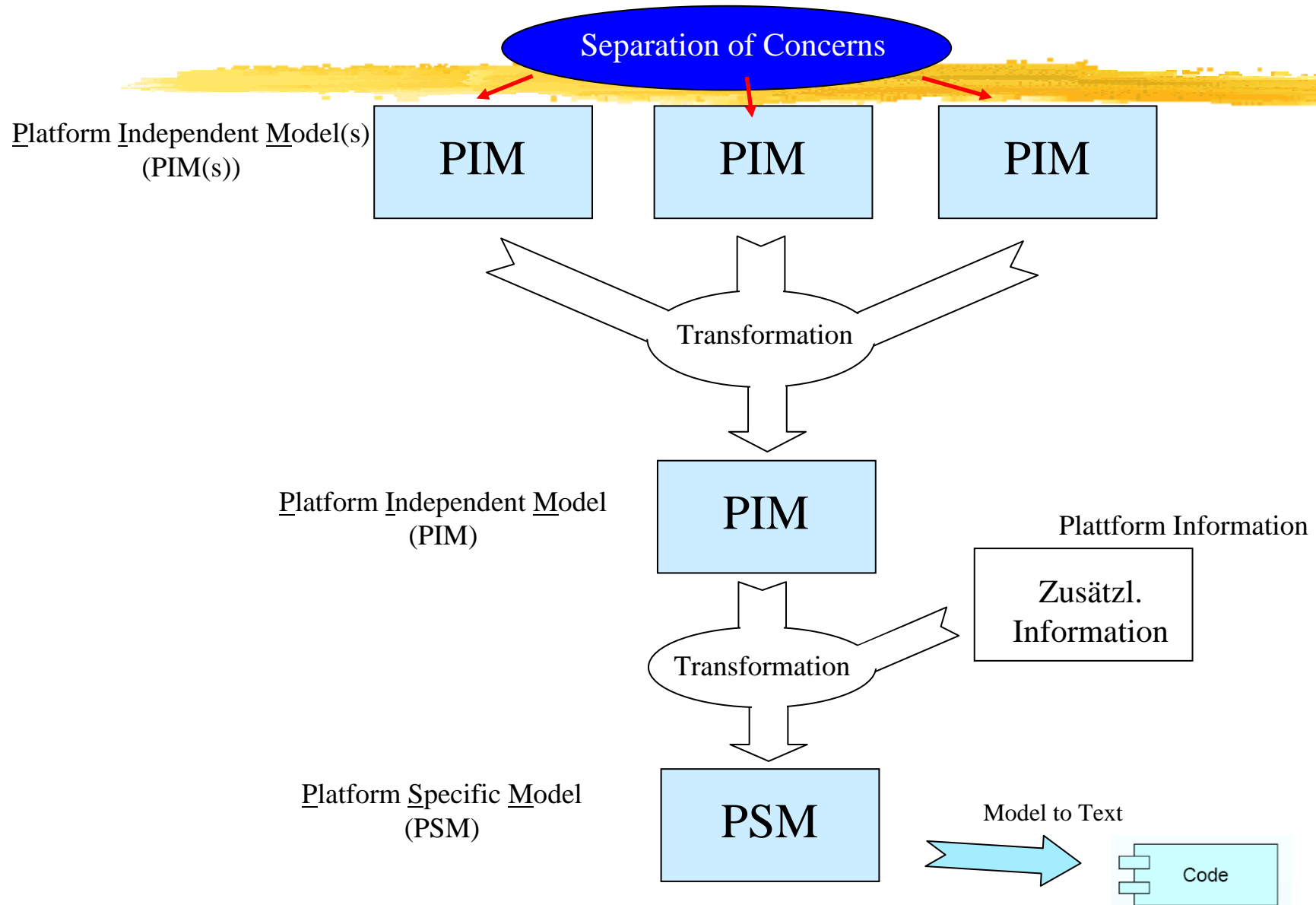
➤ Die Möglichkeit, verschiedene Bereichsmodelle unabhängig voneinander zu erstellen, zu bearbeiten, zu verwalten (**separation of concerns** ... AOP!!!).

Die Möglichkeit, aus verschiedenen Bereichsmodellen ein lauffähiges System zu erzeugen.

Informationen, die zum Zusammenfügen der Bereichsmodelle benötigt werden, müssen außerhalb der Bereichsmodelle verwaltet werden.

# Model Driven Architecture

## Die Lösung - Modelle und Transformationen



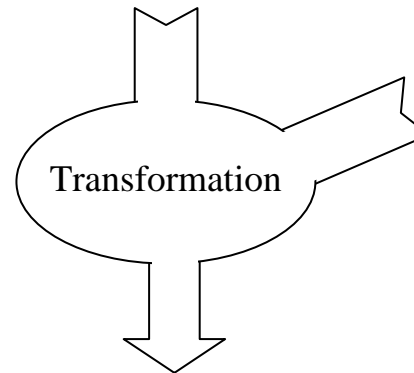
# Model Driven Architecture

## Grundbegriffe

➤ Modelle

Modell

➤ Transformationen



# Model Driven Architecture Modelle

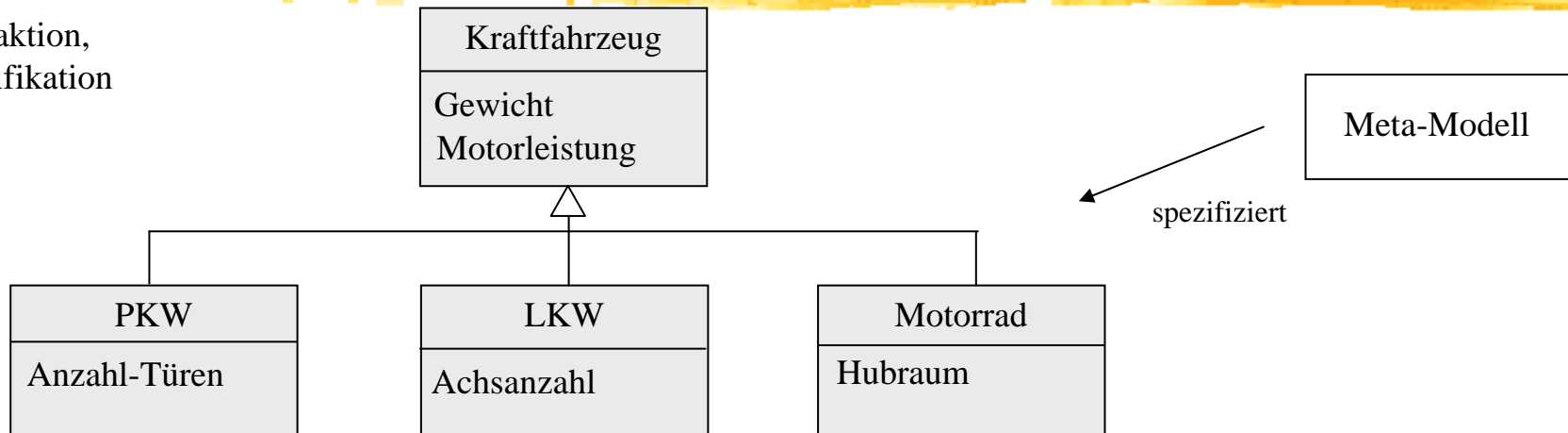


- Ein Modell ist die vereinfachte und idealisierte Beschreibung einer physikalischen, abstrakten oder auch hypothetischen Wirklichkeit.
  
- Es gibt zwei wesentliche Konzepte, die bei der Erstellung eines Modelles zum Einsatz kommen:
  - Abstraktion:
    - Das Weglassen von Details, welche für die betrachteten Zusammenhänge unwesentlich sind
  
  - Klassifikation:
    - Die Gruppierung wichtiger Informationen, basierend auf gemeinsamen Eigenschaften.

# Model Driven Architecture

## Modelle, Modellierungssprachen

Abstraktion,  
Klassifikation



graphische Sprache

textuelle Sprache

```
public class Kraftfahrzeug {
    public double Gewicht;
    public double Motorleistung;
}
public class PKW extends Kraftfahrzeug {
    public int Anzahl-Türen;
}
public class LKW extends Kraftfahrzeug {
    public int Achsanzahl;
}
public class Motorrad extends Kraftfahrzeug {
    public int Hubraum;
}
```

spezifiziert

Meta-Modell

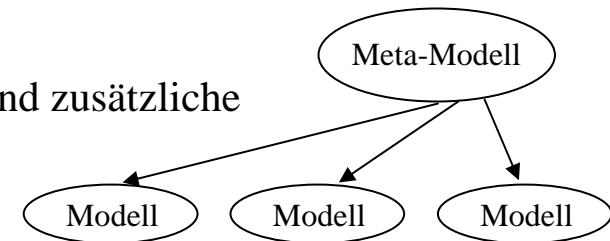
# Model Driven Architecture

## Meta-Modelle

### ➤ Was sind Meta-Modelle ?

➤ Meta-Modelle definieren Elemente, Struktur, Semantik und zusätzliche Einschränkungen (Constraints) von Modellen.

➤ Meta-Modelle spezifizieren die Konzepte (das Vokabular) und die Grammatikregeln für Modellierungssprachen.



### ➤ Wozu braucht man Meta-Modelle ?

➤ Meta-Modelle vereinfachen die Kommunikation über Modelle und ermöglichen es, das gemeinsame Verständnis darüber, was Modelle beschreiben, zu formalisieren.

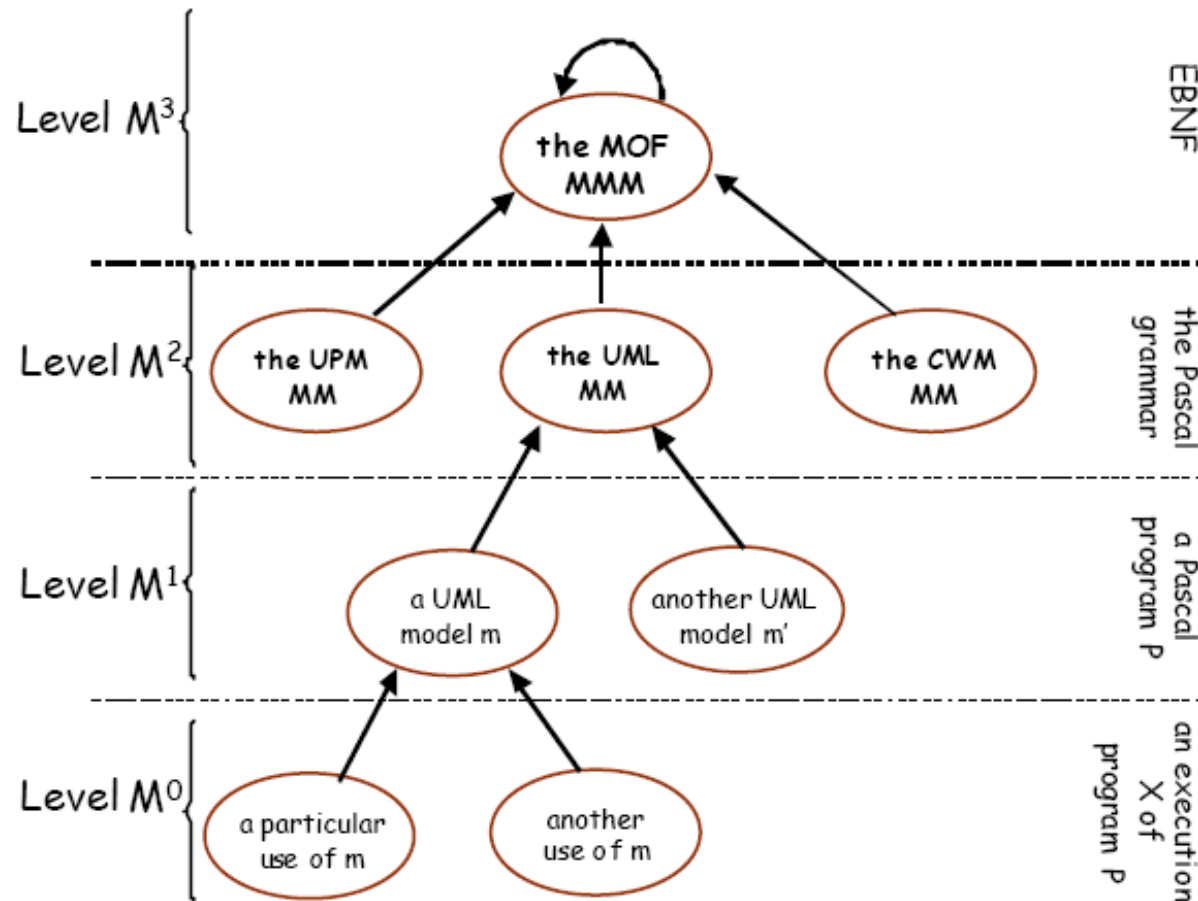
➤ Meta-Modelle werden benötigt, um die Richtigkeit und Korrektheit von Modellen zu überprüfen, um Modelle automatisiert verarbeiten zu können.

➤ Modell-Transformationen werden in Form von Abbildungsfunktionen und Abbildungsregeln zwischen Meta-Modellen definiert.

# Model Driven Architecture

## MOF (Meta Object Facility), der Meta-Modellierungs Standard

➤ Die OMG (Object Management Group) definiert Meta-Modelle, Modelle und Instanzen in vier Ebenen.



**XMI** – definiert die XML-Repräsentation von MOF-basierten Meta-Modellen (Level M<sup>2</sup>), sowie von konkreten Modellen (Level M<sup>1</sup>)

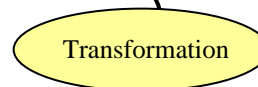
# MDA Models

Computation independent model

Platform independent model

Platform specific model

Platform model



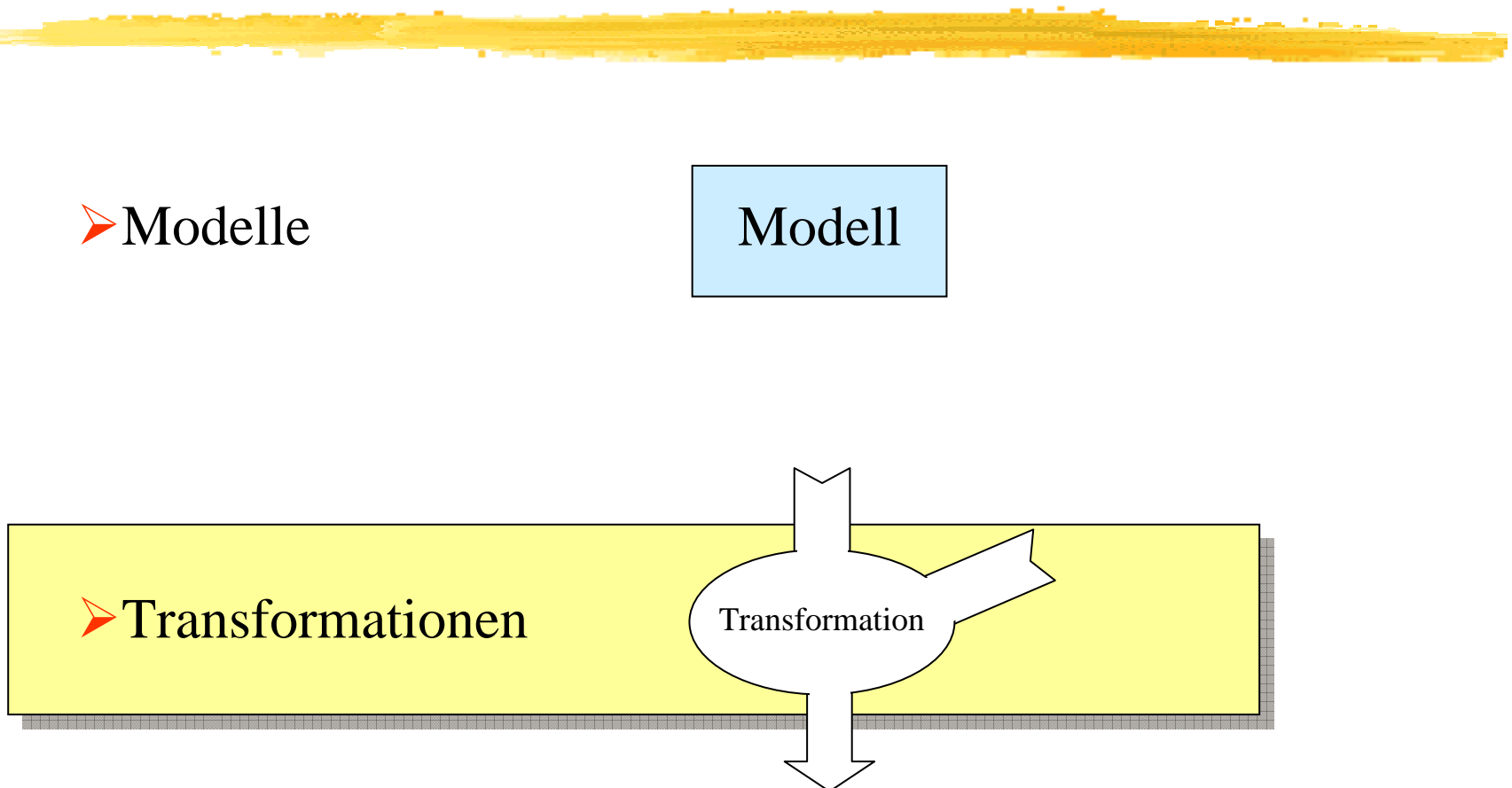
# Model Driven Architecture Grundbegriffe

➤ Modelle

Modell

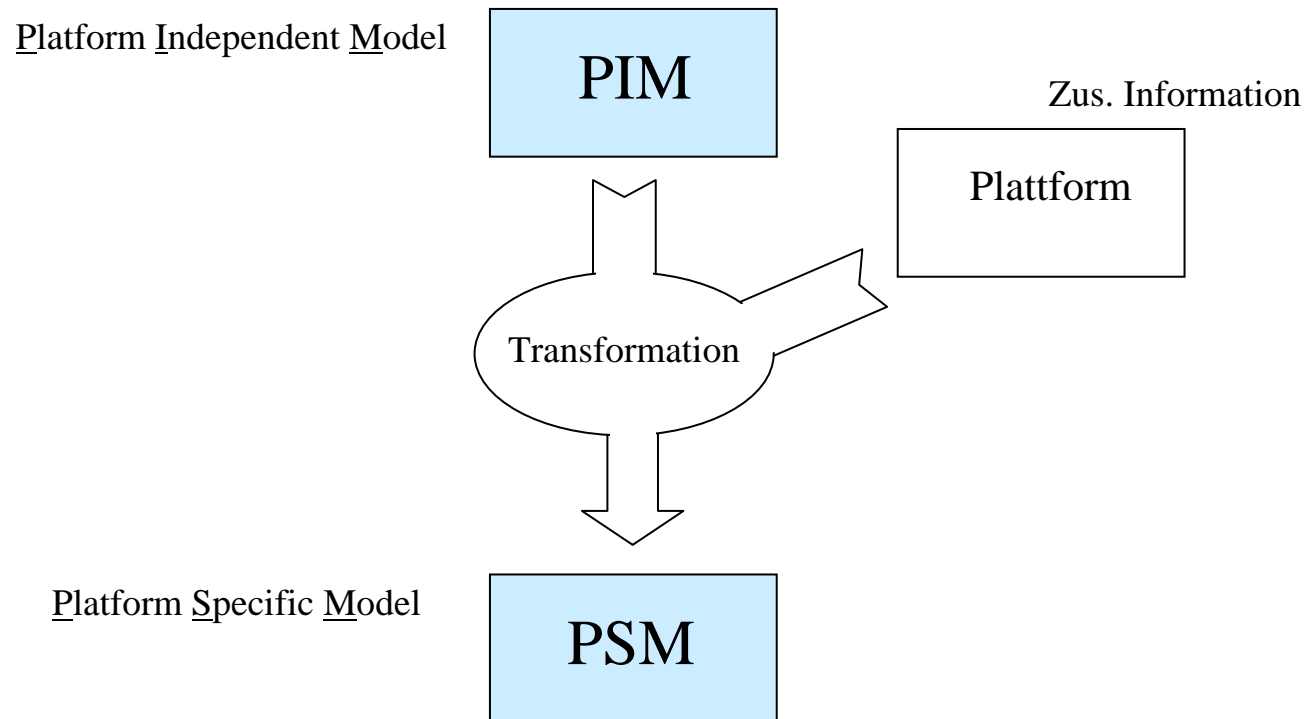
➤ Transformationen

Transformation



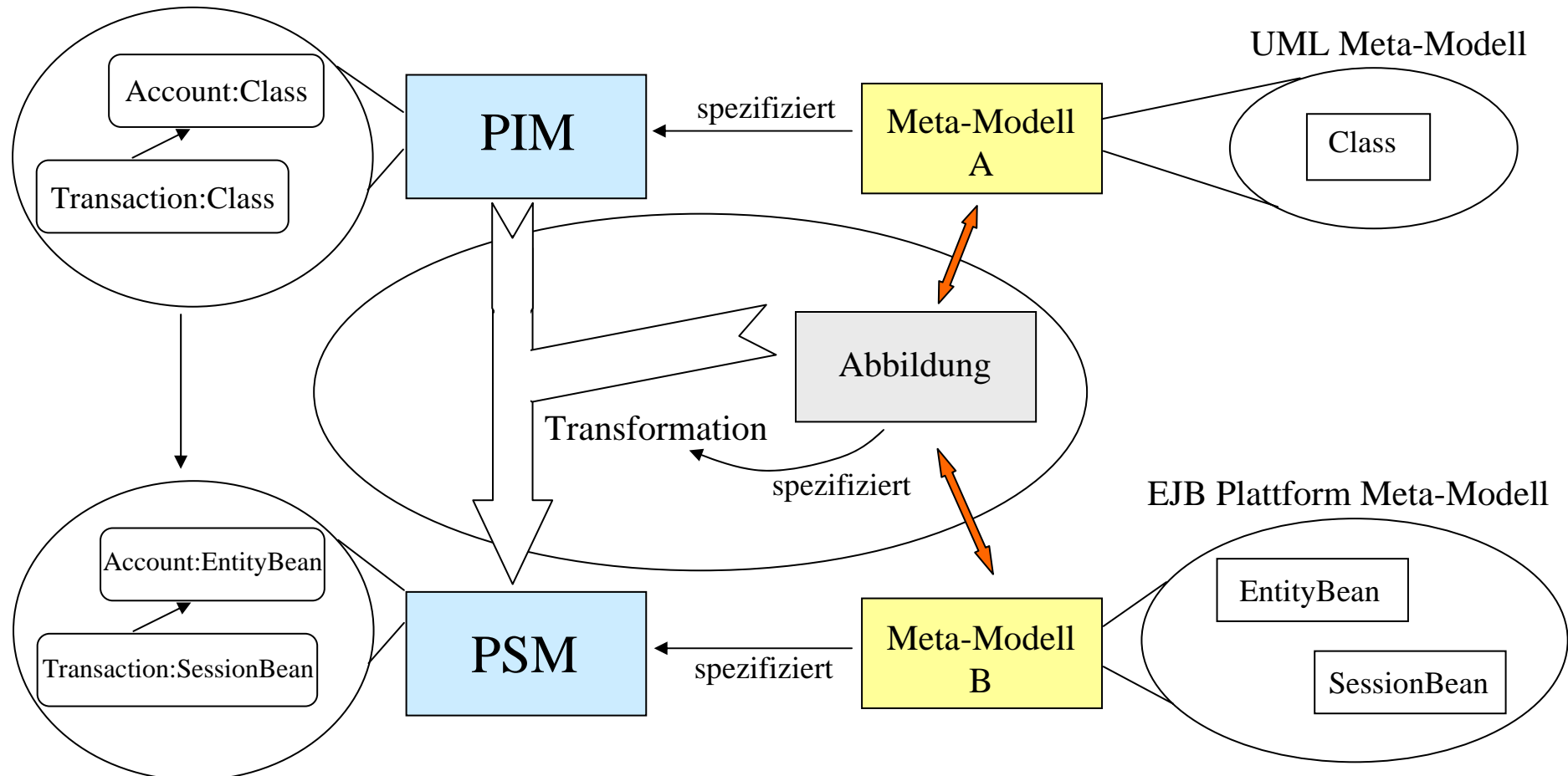
# Model Driven Architecture Transformationen

➤ Warum sind zusätzliche Informationen (Plattform Infos) nötig ?



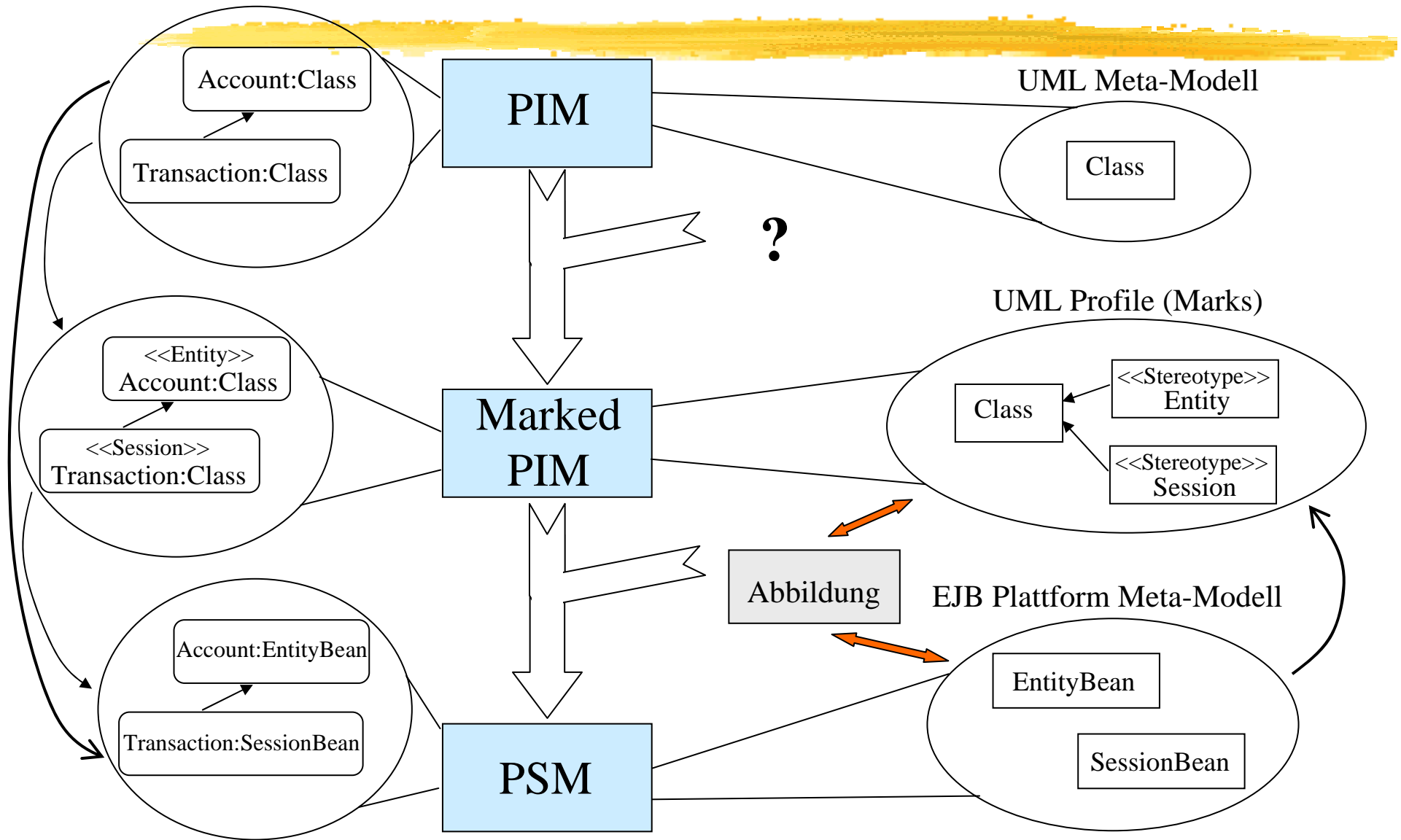
# Model Driven Architecture Transformationen

- Transformationen werden durch Abbildungen spezifiziert.
- Und zwar durch Abbildungen zwischen Elementen der Meta-Modelle.

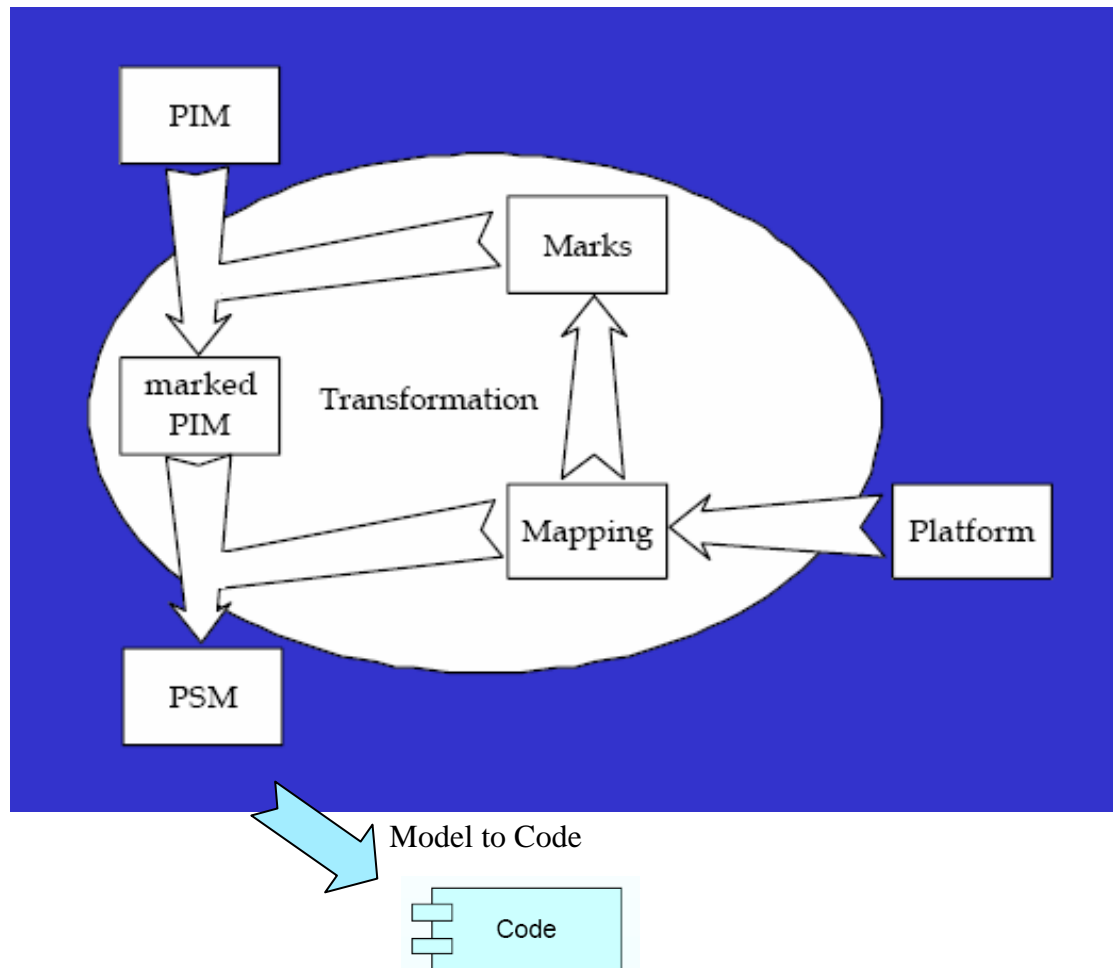


# Model Driven Architecture

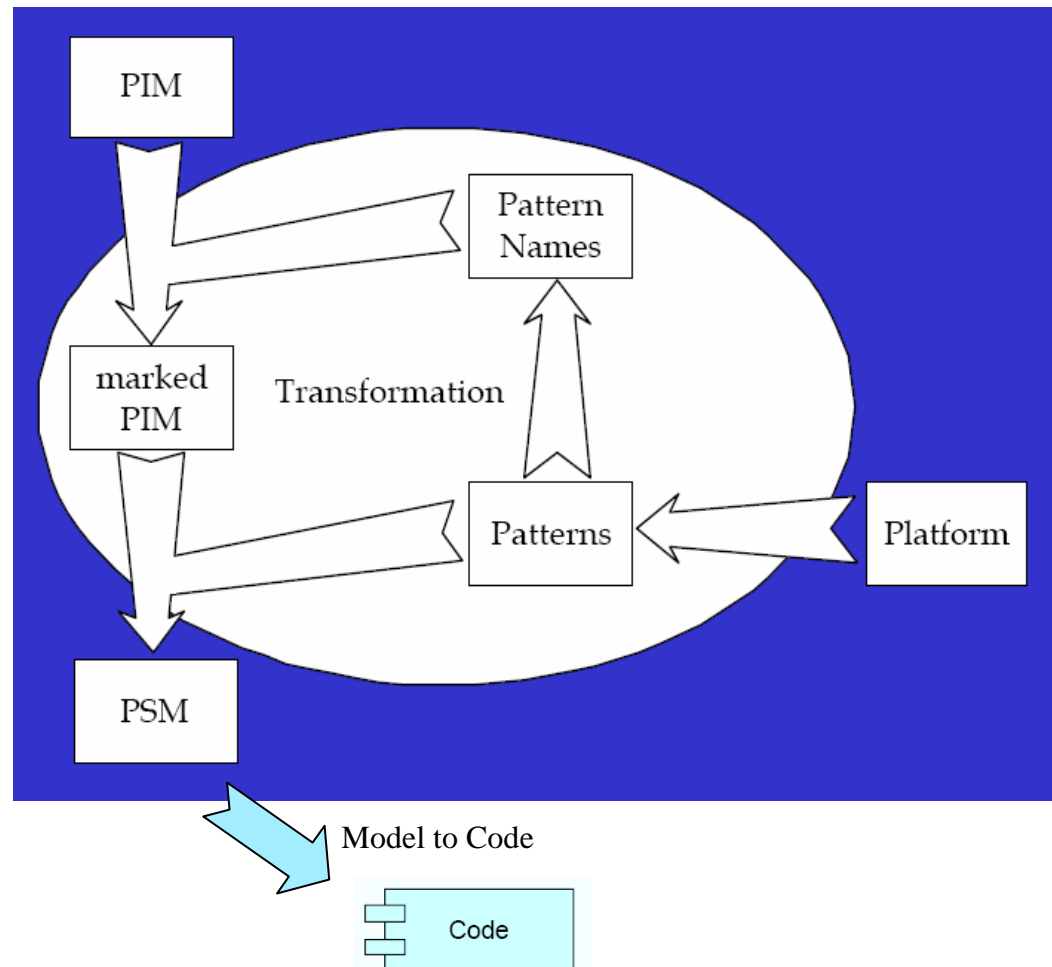
## Transformationen Model-Marking



# Model Driven Architecture Transformationen Model-Marking

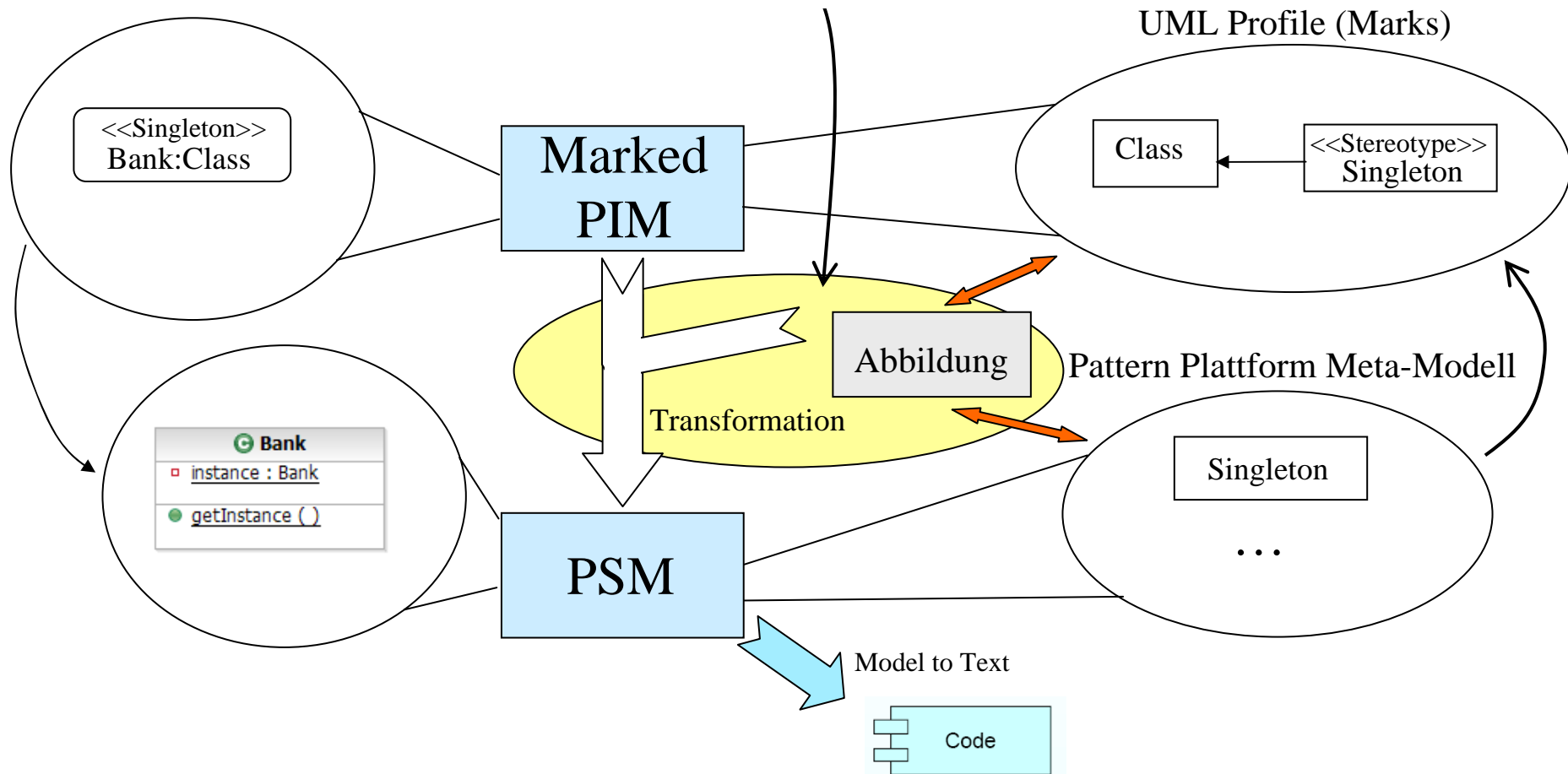


# Model Driven Architecture Transformationen - Patterns



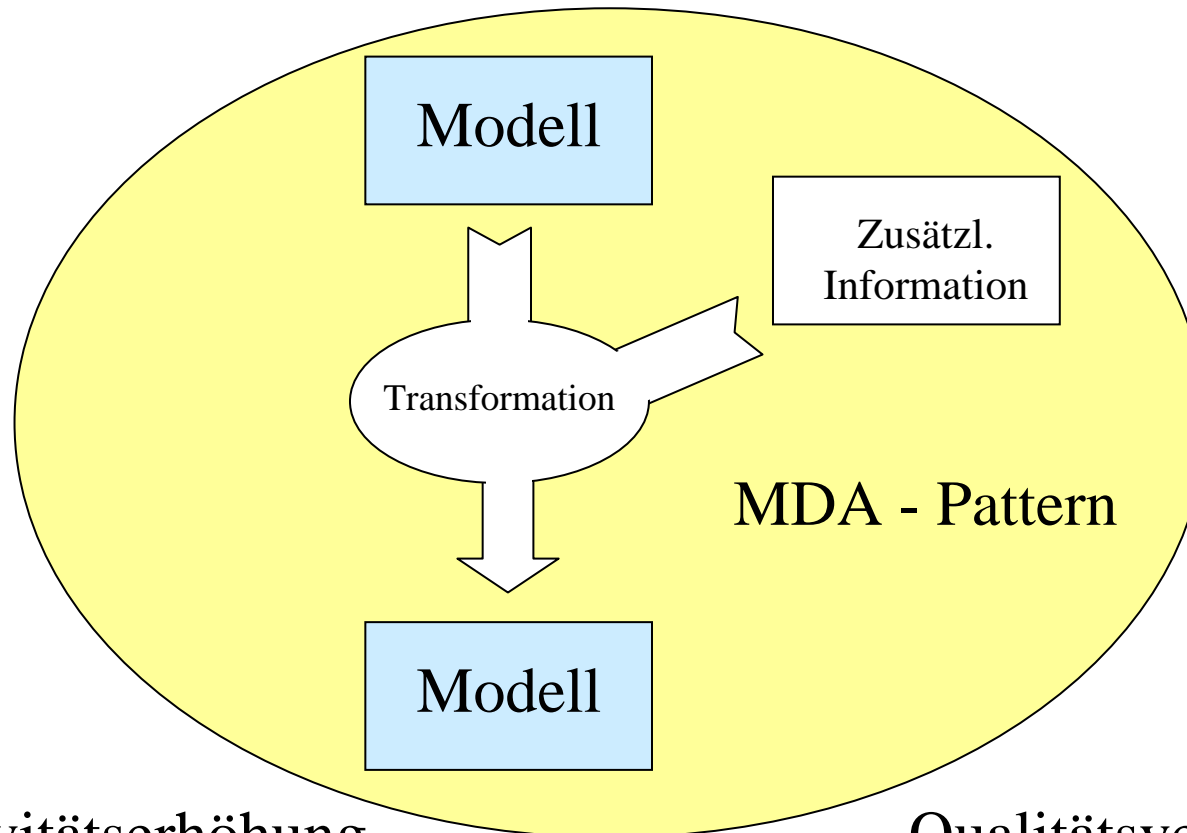
# Model Driven Architecture **Transformationen - Patterns**

- Transformationen bzw. deren Spezifikation machen Designentscheidungen explizit.
- Transformationsspezifikationen sind selbst Modelle



# Model Driven Architecture Zusammenfassung

Investitionssicherung

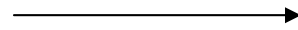


Produktivitätserhöhung

Qualitätsverbesserung

# Model Driven Architecture Zusammenfassung

MDA Konzept



verbessert

➤ Plattformunabhängige (Bereichs-) Modelle sind die zentralen Entwicklungsartefakte.

PIM

➤ Produktivität:

- Durch Konzentration auf Beschreibung der wesentlichen Aspekte eines Bereiches.
- Durch Modellierungssprache die für die Beschreibung des betrachteten Bereiches optimiert ist – leichtere Kommunikation mit dem Fachbereich.

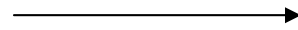
➤ Investitionsschutz:

- Wissen steckt in Plattformunabhängigen Modellen.
- Bereichsmodelle sind unabhängig voneinander wiederverwendbar.

# Model Driven Architecture

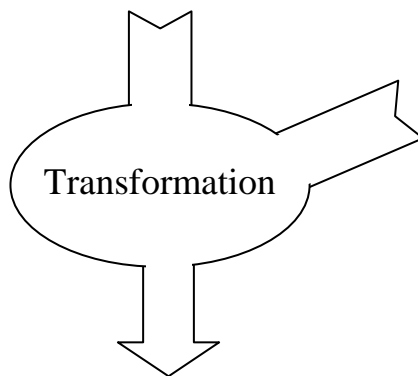
## Zusammenfassung

MDA Konzept



verbessert

➤ Transformationen werden explizit gemacht und sind selbst Modelle, die als “First Class“ Artefakte verwaltet und wiederverwendet werden.



➤ Produktivität:

➤ Wiederholte Tätigkeiten, die einem gleich bleibenden Muster folgen, werden automatisiert.

➤ Qualität:

➤ Verringerung der Fehleranzahl (ist eine Transformation fehlerfrei, dann ist es auch jede ihrer Anwendungen).

➤ Designmuster werden durchgängig und korrekt eingesetzt.

➤ Designentscheidungen sind explizit gemacht.

➤ Investitionsschutz:

➤ Umstieg auf neue Plattformen und Technologien wird stark erleichtert.





Danke !