



# PROFESSIONAL SOFTWARE ENGINEERING

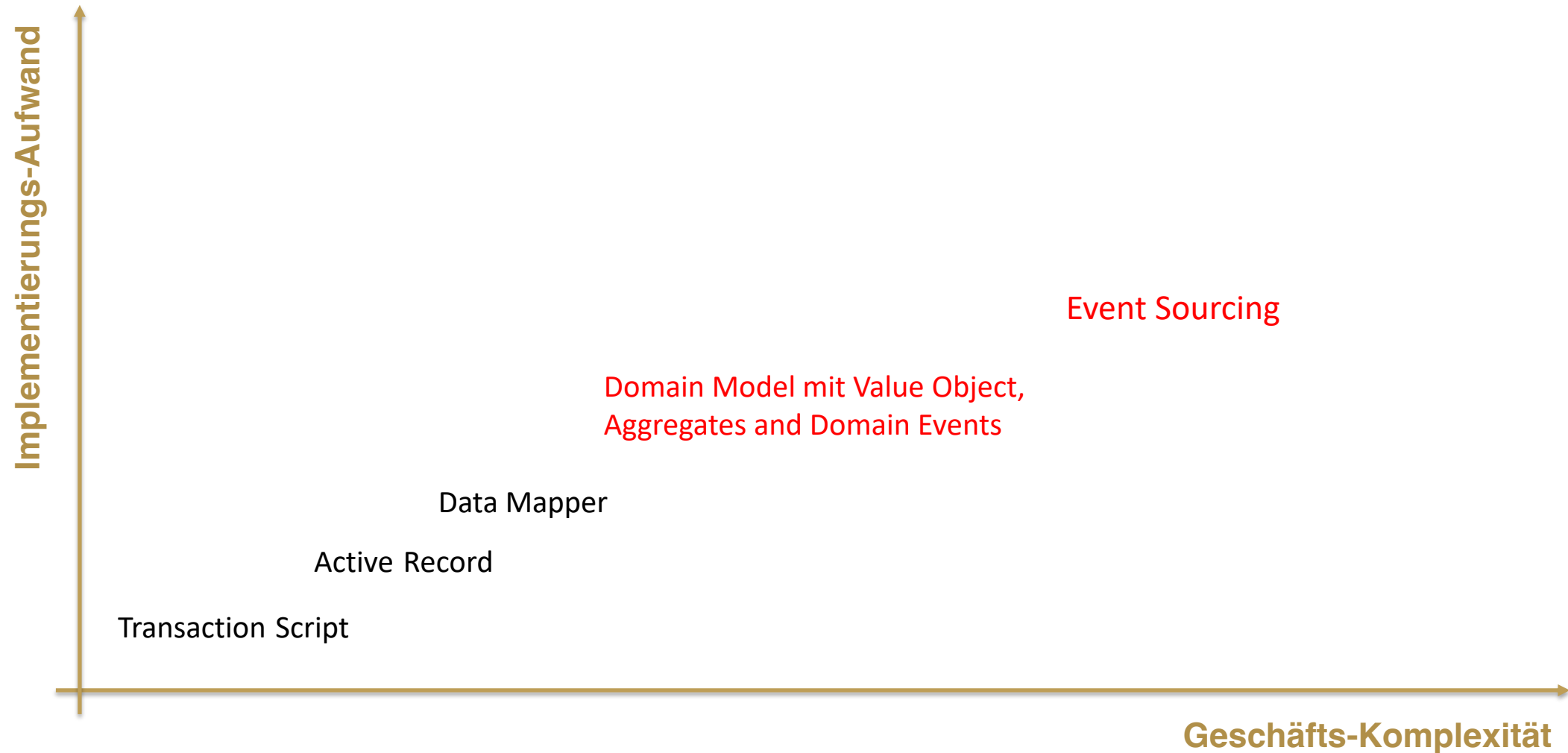
PSE SWE LE 4 und 5 - Domain Driven Design

Taktisches DDD

Dominik Neumann

# ENTWURFS-HEURISTIKEN FÜR GESCHÄFTSLOGIK MIT PERSISTENZ

# ENTWURKS-HEURISTIKEN FÜR PERSISTENZ



# RECAP: DATENBANK-TRANSAKTION

## Transaktions-Klammer

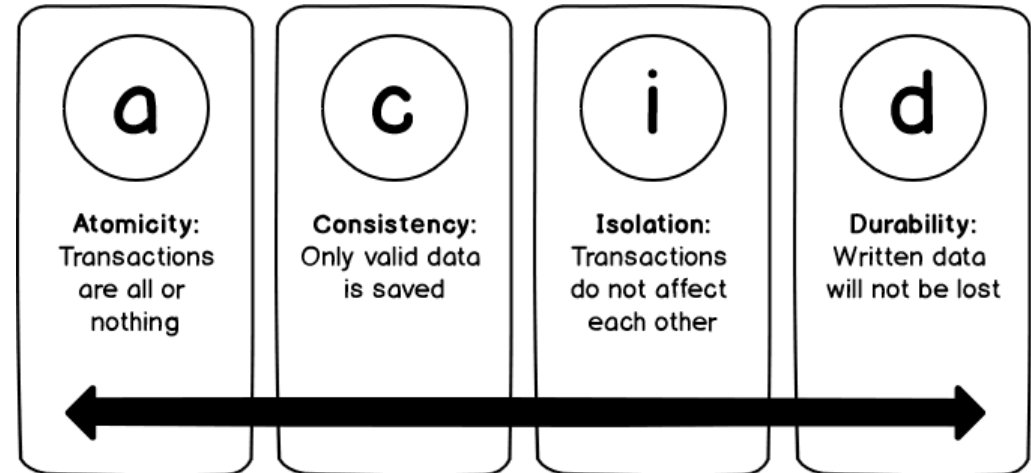
```
begin of transaction  
  write x  
  write y  
end of transaction
```

Transaktionen entweder durch einen **Commit** ausgelöst. Es gibt zwei denkbare Ausgänge:

Commit oder Abort (und damit rollback)

## ACID-Prinzip

Bei der Ausführung von Transaktionen muss das Transaktionssystem die ACID-Eigenschaften garantieren



# RECAP: DATENBANK-TRANSAKTION

- Atomarität (**A**tomicity): Eine Transaktion wird entweder ganz oder gar nicht ausgeführt. Transaktionen sind also „unteilbar“. Wenn eine atomare Transaktion abgebrochen wird, ist das System unverändert.
- Konsistenz (**C**onsistency): Nach Ausführung der Transaktion muss der Datenbestand in einer konsistenten Form sein, wenn er es bereits zu Beginn der Transaktion war.
- Isolation (**I**solation): Bei gleichzeitiger Ausführung mehrerer Transaktionen dürfen sich diese nicht gegenseitig beeinflussen.
- Dauerhaftigkeit (**D**urability): Die Auswirkungen einer Transaktion müssen im Datenbestand dauerhaft bestehen bleiben. Die Effekte von Transaktionen dürfen also nicht verloren gehen oder mit der Zeit verblassen.



# HEURISTIK: TRANSACTION-SCRIPT

CRUD Operationen ohne Objekt-relationales Mapping

„Transaction Script organizes all this logic primarily as a **single procedure**, making calls directly to the database or through a thin database wrapper.“



Quelle: <https://martinfowler.com/eaCatalog/transactionScript.html>

## CRUD-Operationen ohne Objekt-relationales Mapping

Wann sollte man Transaction Script verwenden?

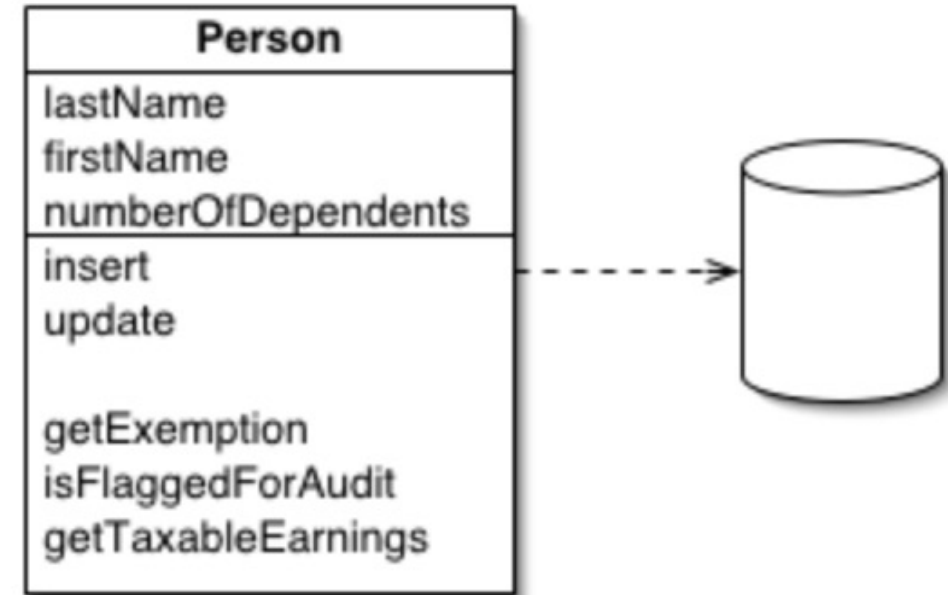
- Fowler: Transaction Script pattern is OK for the 90% of business database systems in which most transactions have simple logic.
- Evans: “The Transaction Script separates UI from application, but does not provide for an object model.”
- Khononov: Transaction script pattern is well adapted to the most straightforward problem domains in which the business logic resembles simple procedural operations. (e.g. Extract Transform Load - ETL)

# HEURISTIK: ACTIVE RECORD

als einfachste Form eines ORM

## Active Record

- is an object that wraps a row in a database table,
- encapsulates the database access,
- and adds domain logic on that data.
- – Fowler, 2002



What you see in the database or objects is likely what exists in the other.

<https://martinfowler.com/eaCatalog/activeRecord.html>



„The essential concept of the active record pattern is that your database records are **active** in your system. Practically what that means is that if you're touching five BlogPost objects, and you save them into your database, they'll end up as five rows in your blog\_posts database table.“

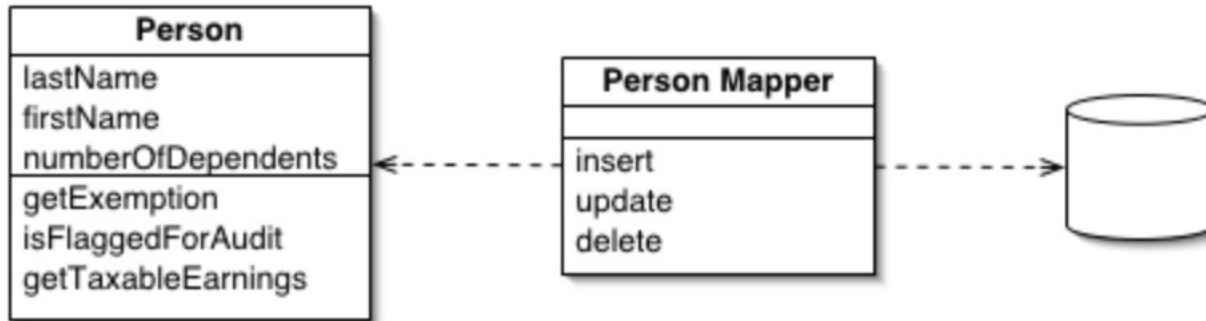
<https://martinfowler.com/eaCatalog/activeRecord.html>

## Wann sollte man Active Record verwenden?

- Khononov: Because an active record is essentially a transaction script that optimizes access to databases, this pattern can only support relatively simple business logic, such as CRUD operations, which, at most, validate the user's input.

# HEURISTIK: DATA MAPPER

- A layer of Mappers that moves data between objects and a database while keeping them independent of each other and the mapper itself.

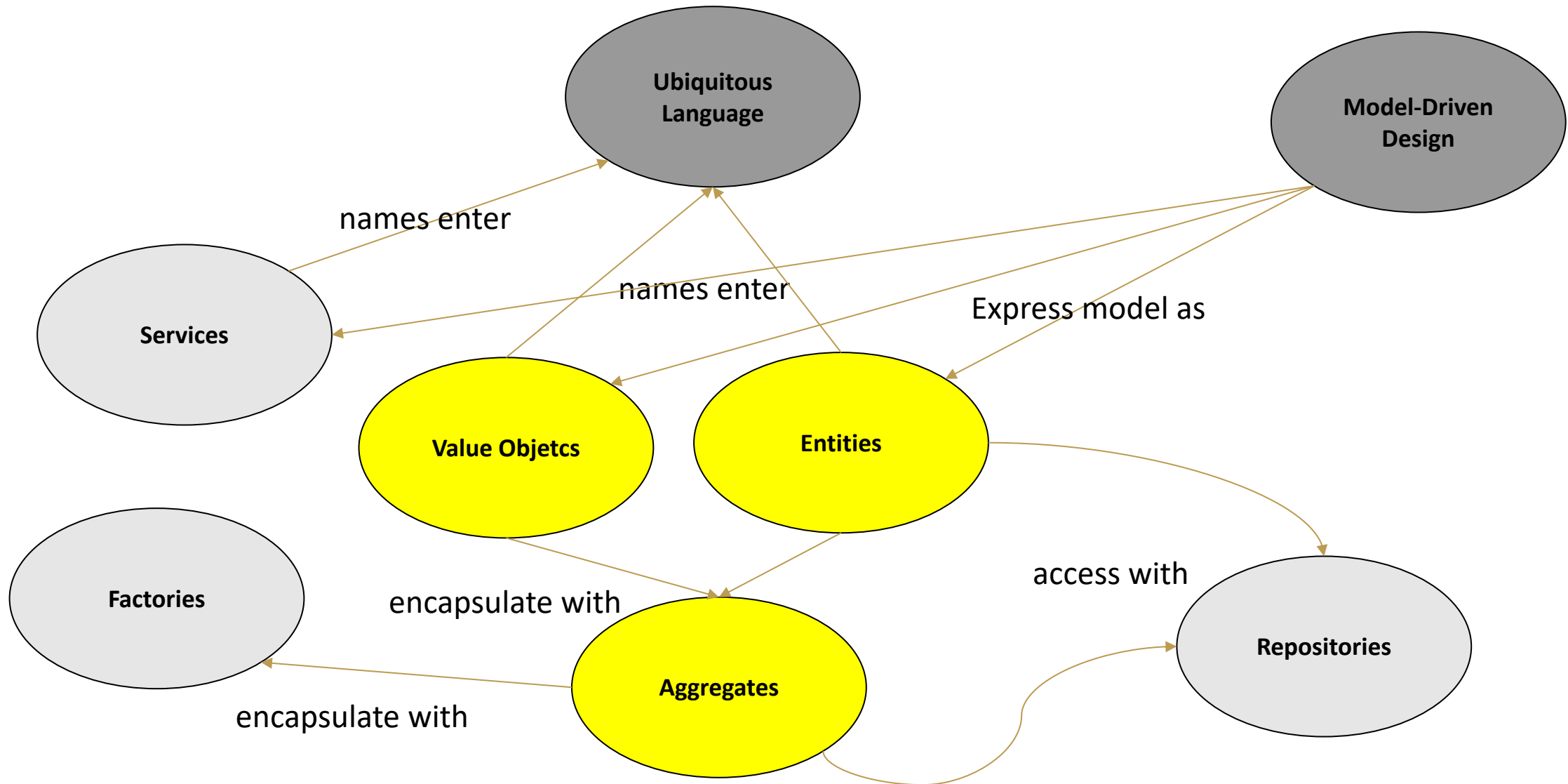


- „The biggest difference between the data mapper pattern and the active record pattern is that:
  - the data mapper is meant to be a layer between the actual business domain of your application and the database that persists its data.
  - where active record seeks to invisibly bridge the gaps between the two as seamlessly as possible, the role of the data mapper is to allow you to consider the two more independently.“

- Objekte und relationale Datenbanken haben unterschiedliche Mechanismen für das Strukturieren von Daten:
  - Fremdschlüssel versus Collections
  - Vererbung
  
- „When you build an object model with a lot of business logic it's valuable to use these mechanisms to better organize the data and the behavior that goes with it. Doing so leads to variant schemas; that is, the object schema and the relational schema don't match up.“

# TAKTISCHES DDD MIT AGGREGATES

# TAKTISCHE MUSTER IM ÜBERBLICK (AUSSCHNITT)



# VALUE OBJECT

## Definition

- „Many objects have no conceptual identity. These objects describe some characteristic of a thing.“ – Eric Evans 2003
- Ein Value Object ist etwas **unveränderliches** (immutable)
- Es hat **keine Identität**. Seine Eindeutigkeit wird durch die Summe seiner Attribute definiert.

- ```
public final class Color {
```
- ```
    public final int red;
```
- ```
    public final int green;
```
- ```
    public final int blue;
```
- ```
    public Color(int r, int g, int b) {
```

```
        this.red = r;
```

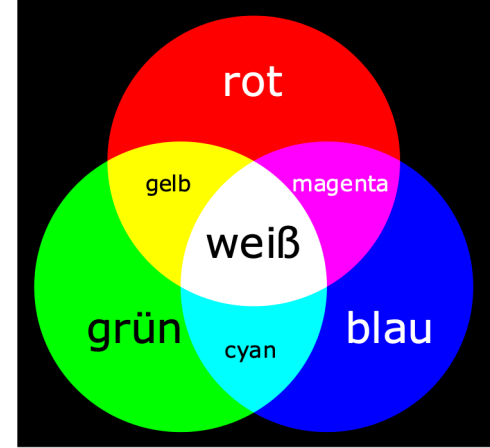
```
        this.green = g;
```

```
        this.blue = b;
```
- ```
    }
```
- ```
    public Color mix(int red, int green, int blue) {
```

```
        return new Color(red, green, blue)
```

```
    }
```

```
}
```





Welche der folgenden Objekte sind Value Objects? Und warum?

- Sozialversicherungsnummer
- Personalausweis
- Adresse
- Kunde
- Bild
- Euro
- Steueridentifikationsnummer
- Auftrag
- Auftragsposition
- Farbe
- Umrechnungsfaktor (zwischen Währungen)
- Person
- Produkt

# VALUE OBJECT

An **immutable** class is a class whose state cannot be changed after it is created.

- Wie definiere ich eine immutable class in Java? was muss man beachten?
- Wie definiert man eine immutable class in Python?

# VALUE OBJECT AS IMMUTABLE CLASS IN JAVA

An **immutable** class is a class whose state cannot be changed after it is created:

- ❑ Make the class **final**: (verbiestet subclassing)
- ❑ All fields should be **final**
- ❑ Provide a **constructor to set all fields**
- ❑ Do not provide **setter** methods
- ❑ Do **not modify the state in any methods**
- ❑ Ensure that **mutable objects are defensive copies**
- ❑ Override the **equals** and **hashCode** methods to ensure that objects with the same state are considered equal.

## Definition

*„Many objects are not fundamentally defined by their attributes, but rather by a thread of continuity and identity.“*

*- Eric Evans 2003*

- Eine Entity modelliert ein **individuelles** Ding. Sie ist einzigartig.
- Jede Entity hat eine eindeutige **Identität** und einen **Zustand**, der sich im Gegensatz zur Identität über den Zeitverlauf ändern kann.

```
class Person {  
  
    private int id;  
    private String firstName;  
    private String lastName;  
    private String eMail;  
    private String phoneNumber;  
    private Date birthDate;  
    .....  
  
}
```



# ENTITY

- Nutze Value Objects so oft Du kannst!

```
class Person {
```

```
    private int id;
```

```
    private String firstName;
```

```
    private String lastName;
```

```
    private String eMail;
```

```
    private String phoneNumber;
```

```
    private Date birthDate;
```

```
    .....
```

```
}
```

```
class Person {
```

```
    private PersonId id;
```

```
    private Name firstName;
```

```
    private Name lastName;
```

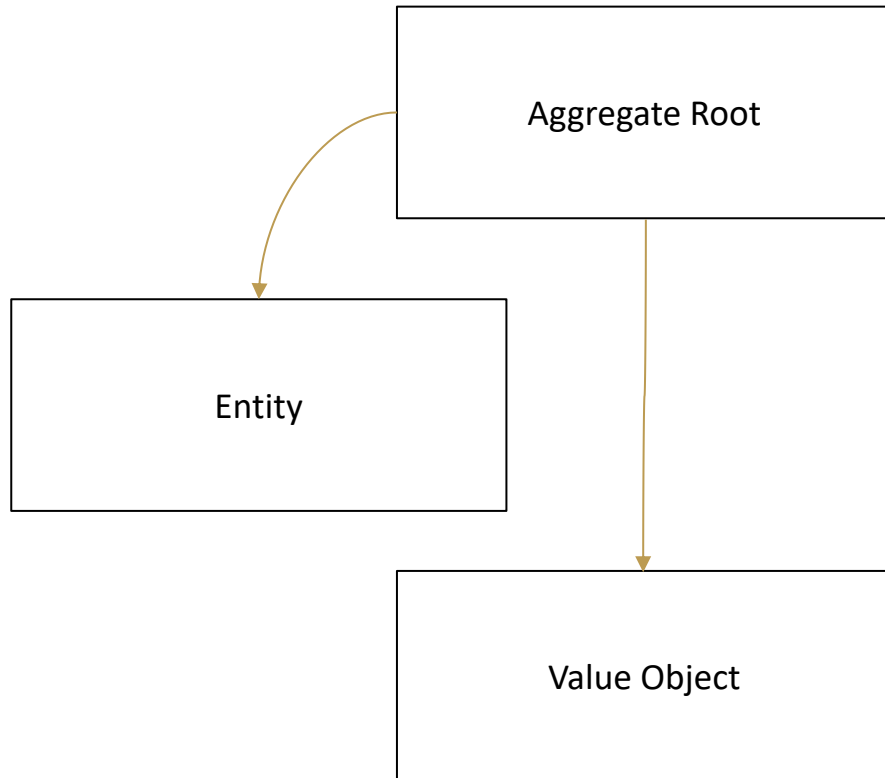
```
    private EmailAdress eMail;
```

```
    private PhoneNumber phoneNumber;
```

```
    private Date birthDate;
```

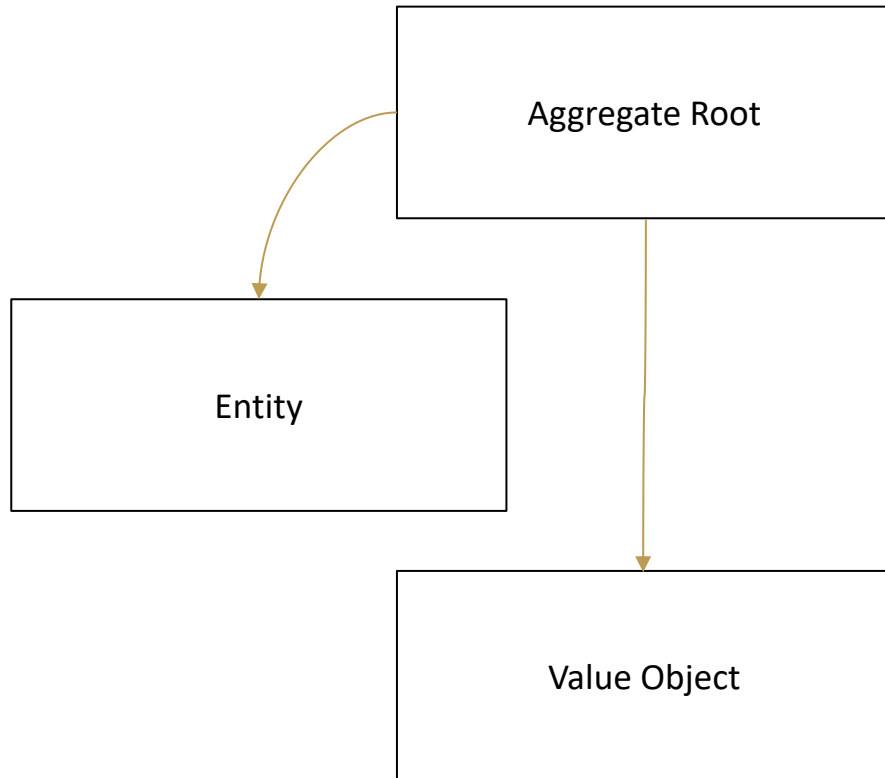
```
    .....
```

```
}
```



- *Aggregates* bestehen aus einem oder mehreren *Entities*.
- Genau eine Entity bildet die *Aggregate Root*.
- Aggregates können auch *Value Objects* enthalten.





- Das Aggregate fasst also Entitäten und Value Objects zusammen.
- Der Name der Root Entity ist der konzeptionelle Name des Aggregate.

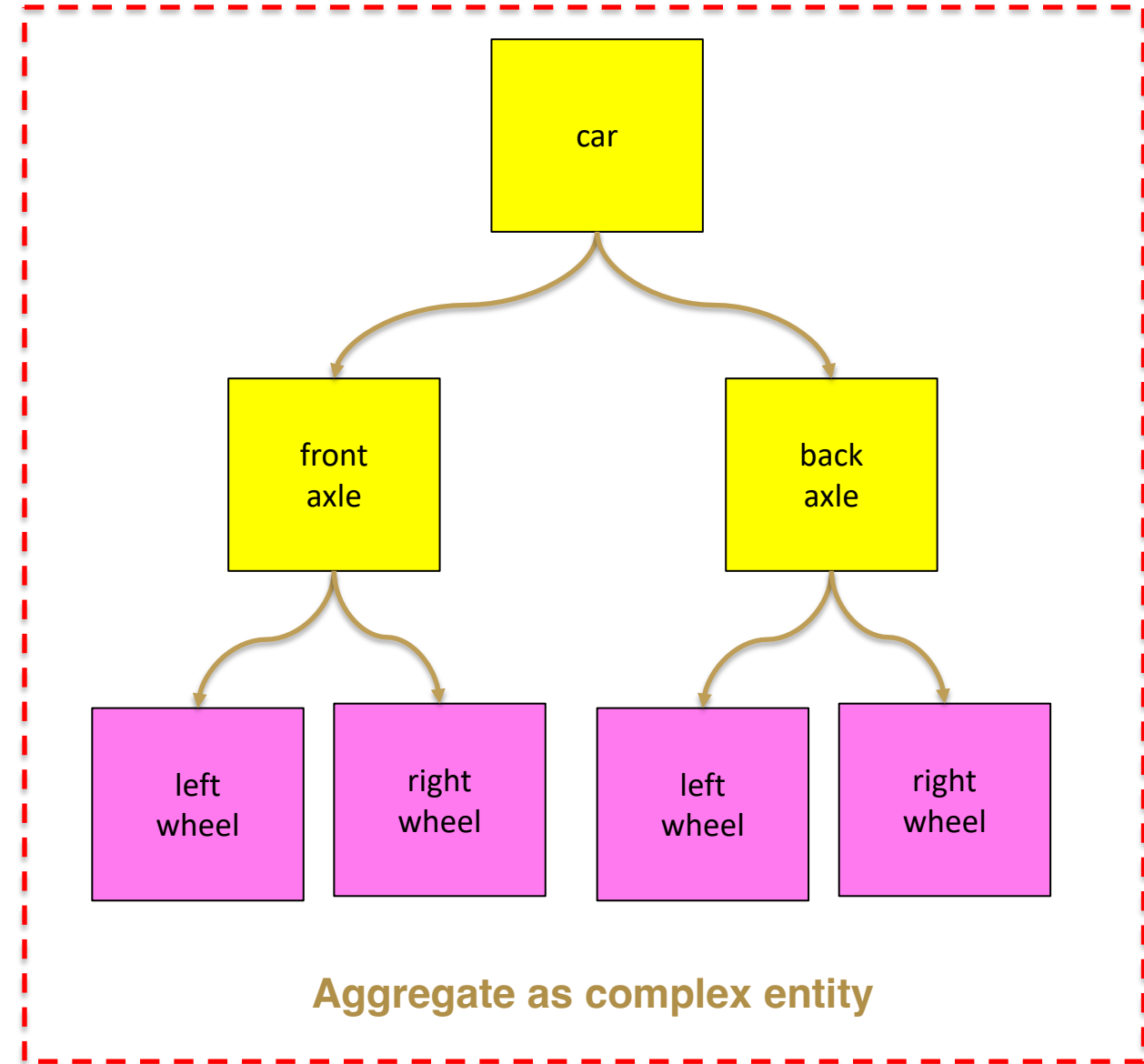
# AGGRGATES

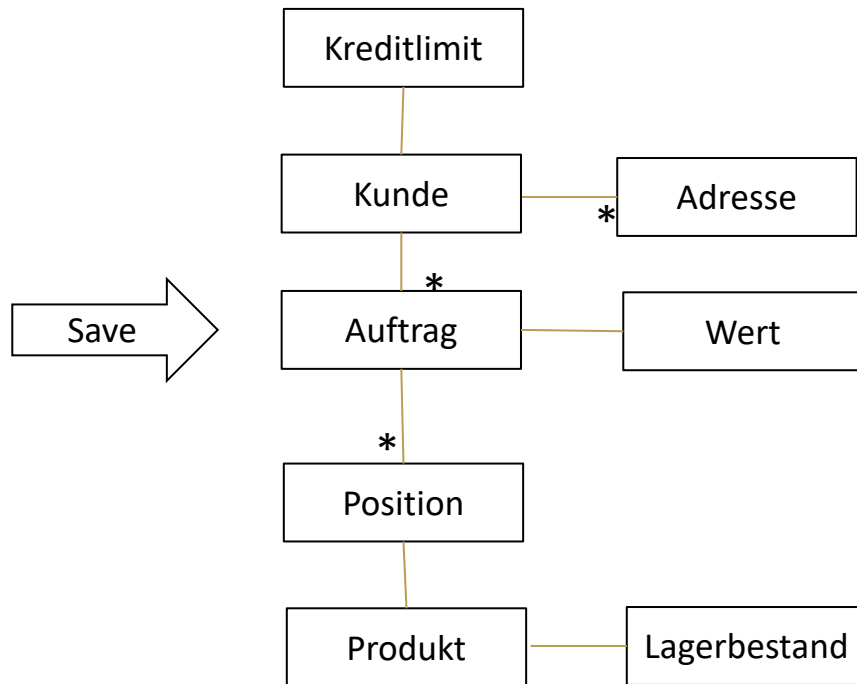
Aggregate is a unit of

- consistency

and

- concurrency





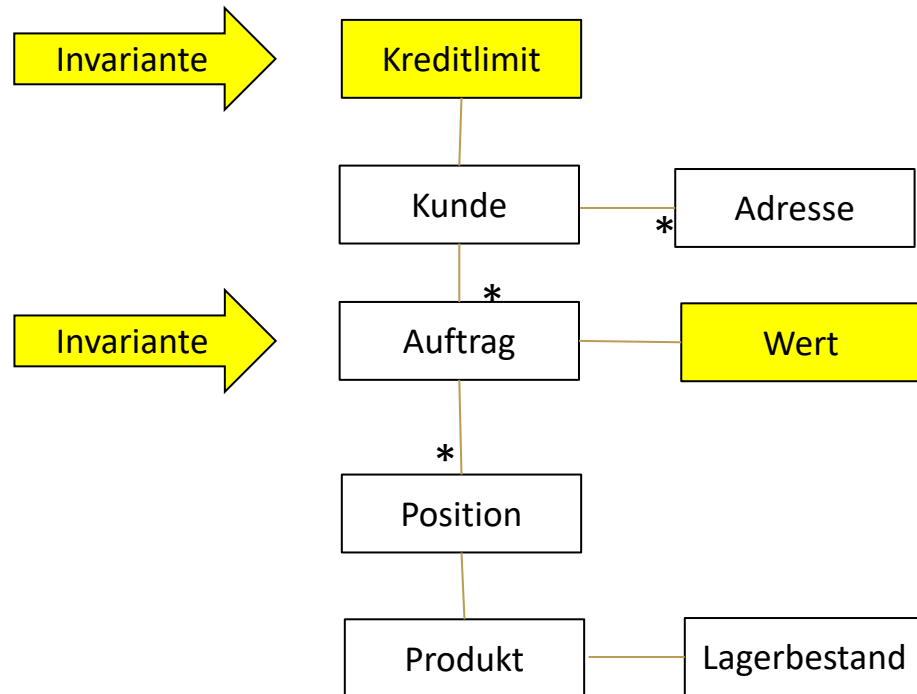
Wer kann schnell ein SQL-Statement schreiben, das

- eine Auftragsposition hinzufügt,
- den Lagerbestand reduziert,
- den Wert des Auftrags berechnet,
- prüft, ob der Wert des Auftrags das Kreditlimit des Kunden überschreitet,
- prüft ob der Mindest-Auftragswert erreicht ist
- und dann den Auftrag mit der neuen Position, sowie das neu ausgereizte Kreditlimit am Kunden speichert?

# INVARIANTEN UND AGGREGATES

**IF THERE ARE NO INVARIANTS –  
THERE IS NO NEED FOR AN  
AGGREGATE?**

# INVARIANTEN UND AGGREGATES



*Invariants need to be maintained that apply to closely related groups of objects, not just discrete objects.*

*- Eric Evans 2003*

**Invariant: observable property that is always true**

## Definition:

- Eine Invariante ist eine Bedingung oder Regel, die in einem bestimmten Kontext immer wahr sein muss, unabhängig von Änderungen oder Zustandsübergängen.
- Im Kontext von Aggregates bedeutet dies, dass Invarianten sicherstellen, dass der Zustand eines Aggregates stets valide bleibt, entsprechend den geschäftlichen Anforderungen.
- Die Invarianten definieren die Regeln, die zwischen den Objekten des Aggregate eingehalten werden müssen, um einen konsistenten Zustand zu gewährleisten.

Beispiel: In einem Bestell-Aggregate könnte die Invariante lauten: "Die Gesamtsumme einer Bestellung muss der Summe aller Positionen entsprechen."



## Definition:

- Eine Invariante ist eine Bedingung oder Regel, die in einem bestimmten Kontext immer wahr sein muss, unabhängig von Änderungen oder Zustandsübergängen.
- Im Kontext von Aggregates bedeutet dies, dass Invarianten sicherstellen, dass der Zustand eines Aggregates stets valide bleibt, entsprechend den geschäftlichen Anforderungen.
- Die Invarianten definieren die Regeln, die zwischen den Objekten des Aggregate eingehalten werden müssen, um einen konsistenten Zustand zu gewährleisten.

Beispiel: In einem Bestell-Aggregate könnte die Invariante lauten: "Die Gesamtsumme einer Bestellung muss der Summe aller Positionen entsprechen."

Tipp: Invarianten helfen dabei, die Grenzen eines Aggregate zu definieren.

- Ein Aggregate ist verantwortlich für die Einhaltung seiner eigenen Invarianten und kann nicht garantieren, dass externe Daten ebenfalls konsistent sind.
- Alle Änderungen an einem Aggregate müssen in einer einzigen Transaktion erfolgen, sodass die Invarianten stets gewahrt bleiben.
- Dadurch wird vermieden, dass das Aggregate in einem inkonsistenten Zustand bleibt, etwa durch parallele Änderungen oder unvollständige Updates.
- Ein System, das seine Invarianten effektiv durchsetzt, ist weniger anfällig für Fehler, da fehlerhafte Zustände frühzeitig erkannt und verhindert werden.

## Praktische Umsetzung von Invarianten

- Invarianten werden meist in der Root Entity (dem Aggregate Root) implementiert, da diese die Verantwortung für die Konsistenz des gesamten Aggregates trägt.

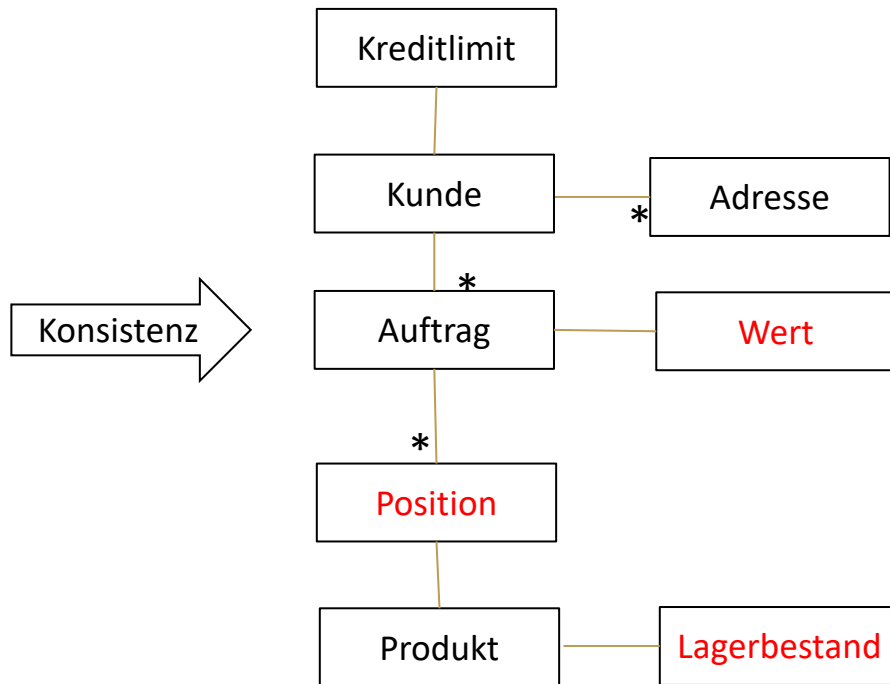
Beispiel: Methoden der Root Entity validieren Zustandsänderungen und werfen Fehler, falls eine Invariante verletzt wird.

- Änderungen an einem Aggregate müssen die Invarianten überprüfen und durchsetzen.

Beispiel: Beim Hinzufügen eines neuen Bestellpostens wird geprüft, ob die maximale Anzahl an erlaubten Posten nicht überschritten wird.

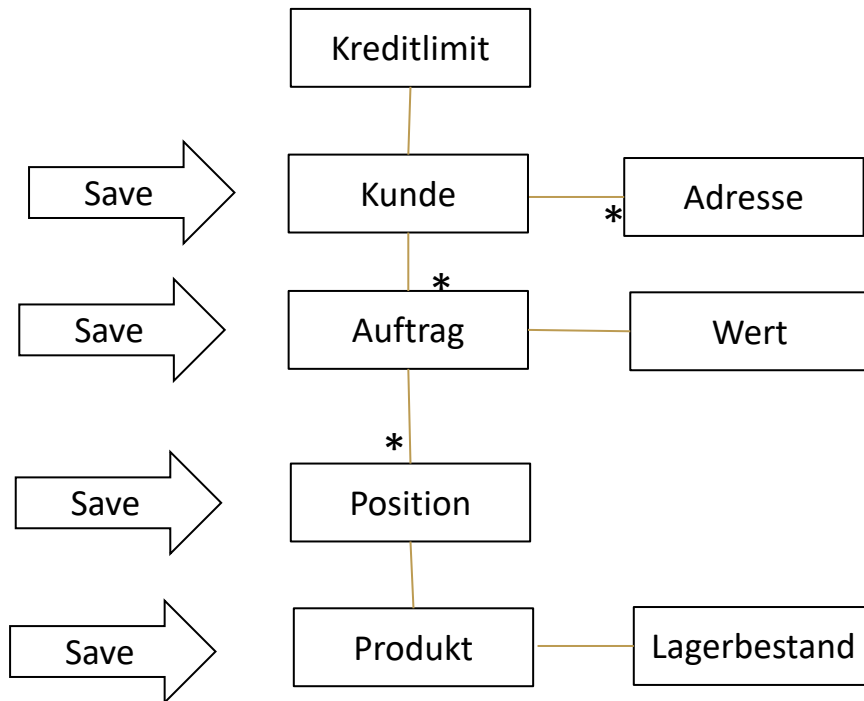
- Unit-Tests und Integrationstests werden genutzt, um sicherzustellen, dass die Invarianten auch bei komplexen Zustandsübergängen eingehalten werden.

# AGGREGATES UND TRANSAKTIONSGRENZEN



*It is difficult to guarantee the consistency of changes to objects in a model with complex associations.*

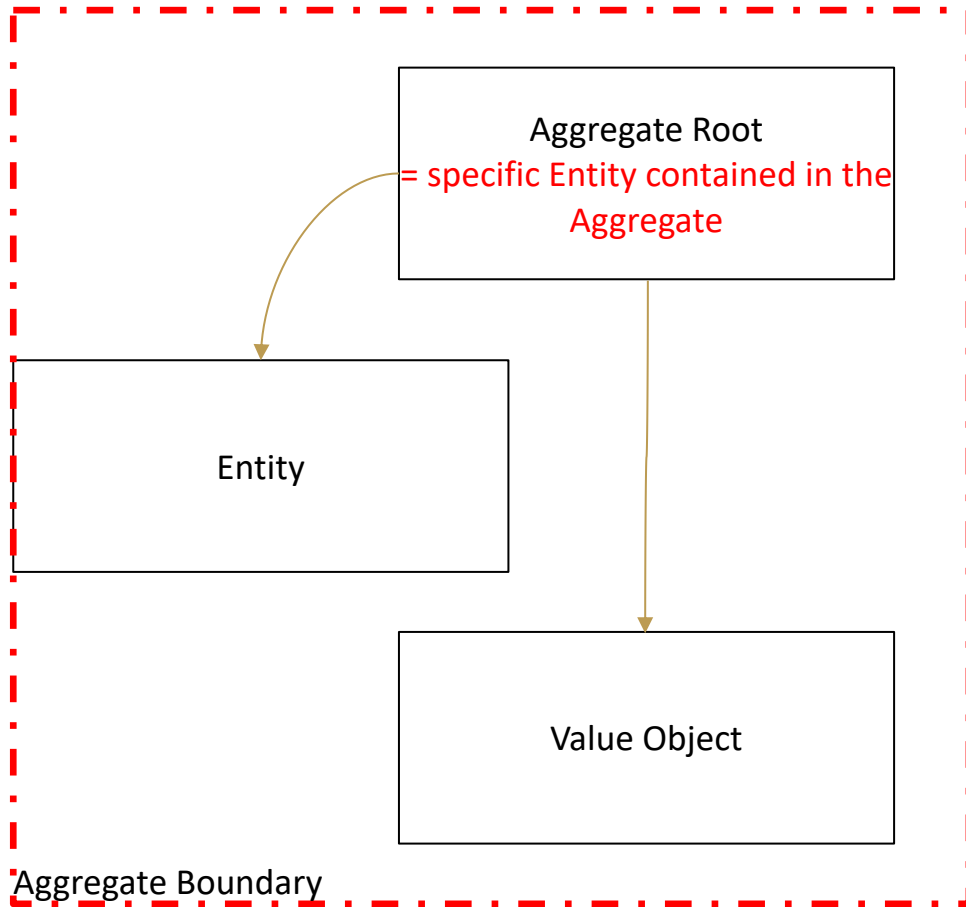
*- Eric Evans 2003*



*The web of relationships in a typical object model gives no clear limit to a potential change.*

*- Eric Evans 2003*





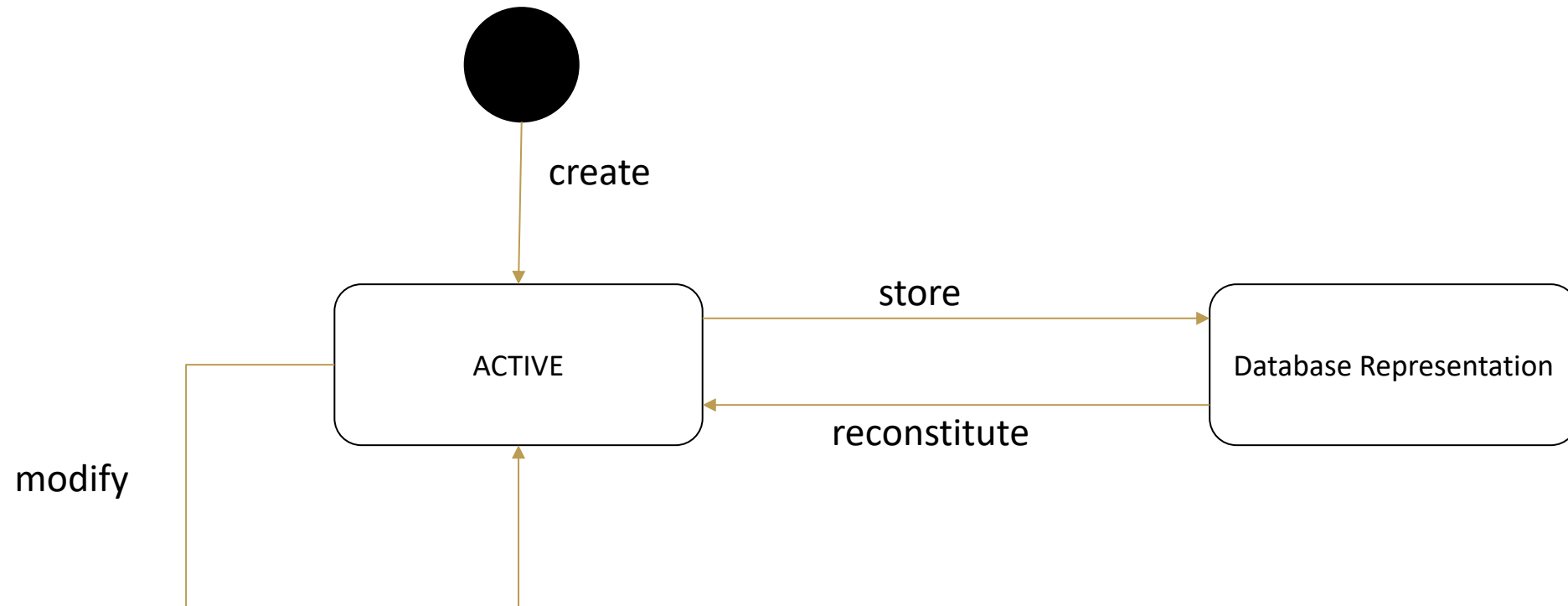
- Jedes Aggregate bildet eine **transaktionale Konsistenzgrenze**:
- „Was sich gleichzeitig gemeinsam ändert gehört in ein Aggregate.“
- Das bedeutet, dass innerhalb eines Aggregate gemäß Geschäftsregeln immer ein konsistenter Zustand sein muss.
- Ein Aggregate modelliert immer ein konzeptionelles Ganzes (hohe Kohäsion)
- Der Zugriff erfolgt immer über die Root.

*„An Aggregate is a cluster of associated objects that we treat as a unit for the purpose of data change.“ – Evans 2003*

# AGGREGATE - A LIFECYCLE PATTERN

# AGGREGATE - A LIFECYCLE PATTERN

Aggregate is a lifecycle pattern

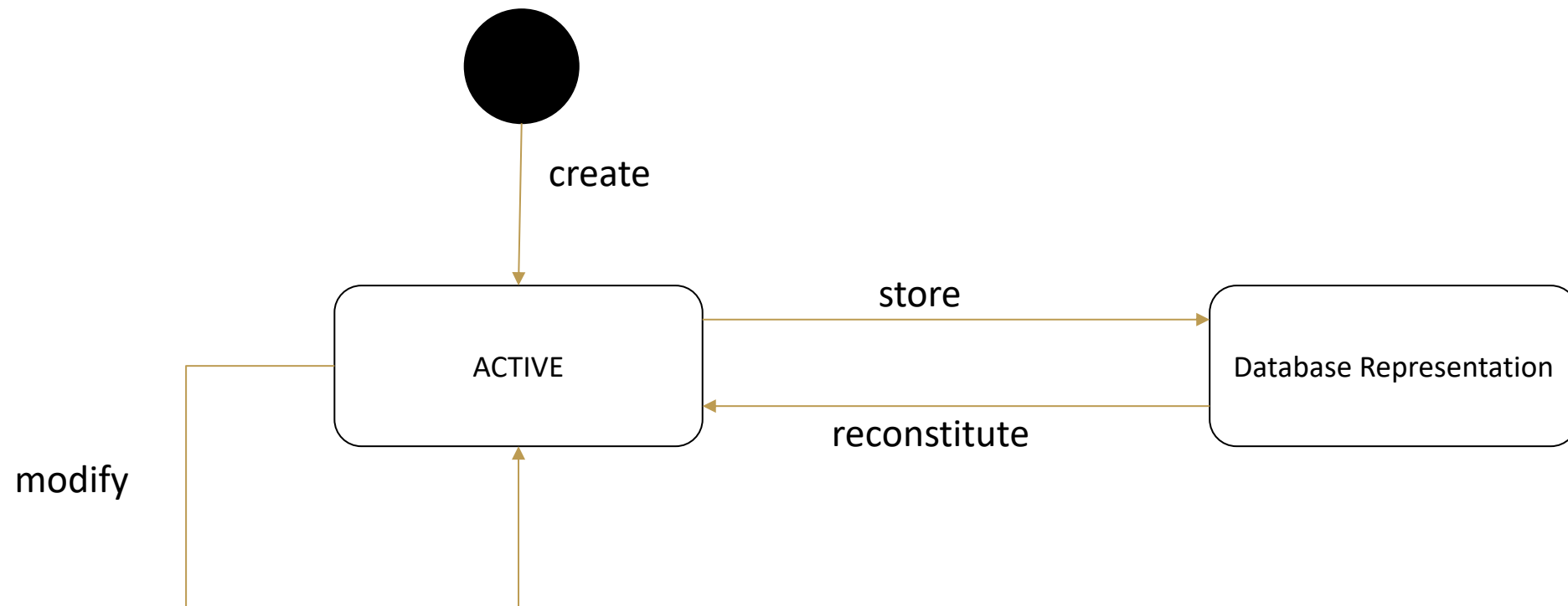


Quelle: Evans, Chapter 6 The Life Cycle of Domain Objects

# AGGREGATE - A LIFECYCLE PATTERN

Es gibt zwei Herausforderungen:

- Wie bewahren ich die **Integrität** des Modells während des **gesamten Lebenszyklus**.
- Wie schütze ich das Modell vor der Komplexität der **Verwaltung des Lebenszyklus** (→ ORM)

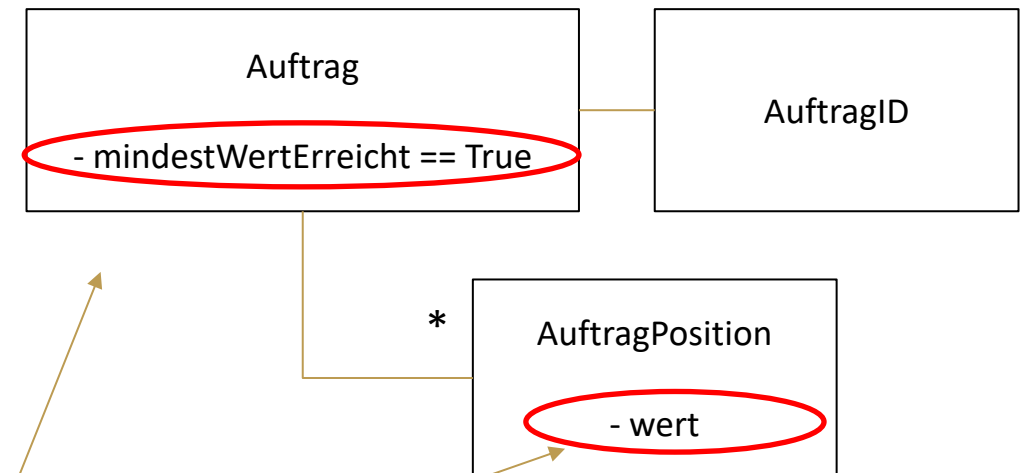


Quelle: Evans, Chapter 6 The Life Cycle of Domain Objects

# DAUMENREGELN FÜR DEN ENTWURF MIT AGGREGATES

Schütze fachliche Invarianten innerhalb von Aggregate-Grenzen

Der **Fachbereich** bestimmt die Zusammensetzung eines Aggregate auf Grundlage dessen, was konsistent sein muss!

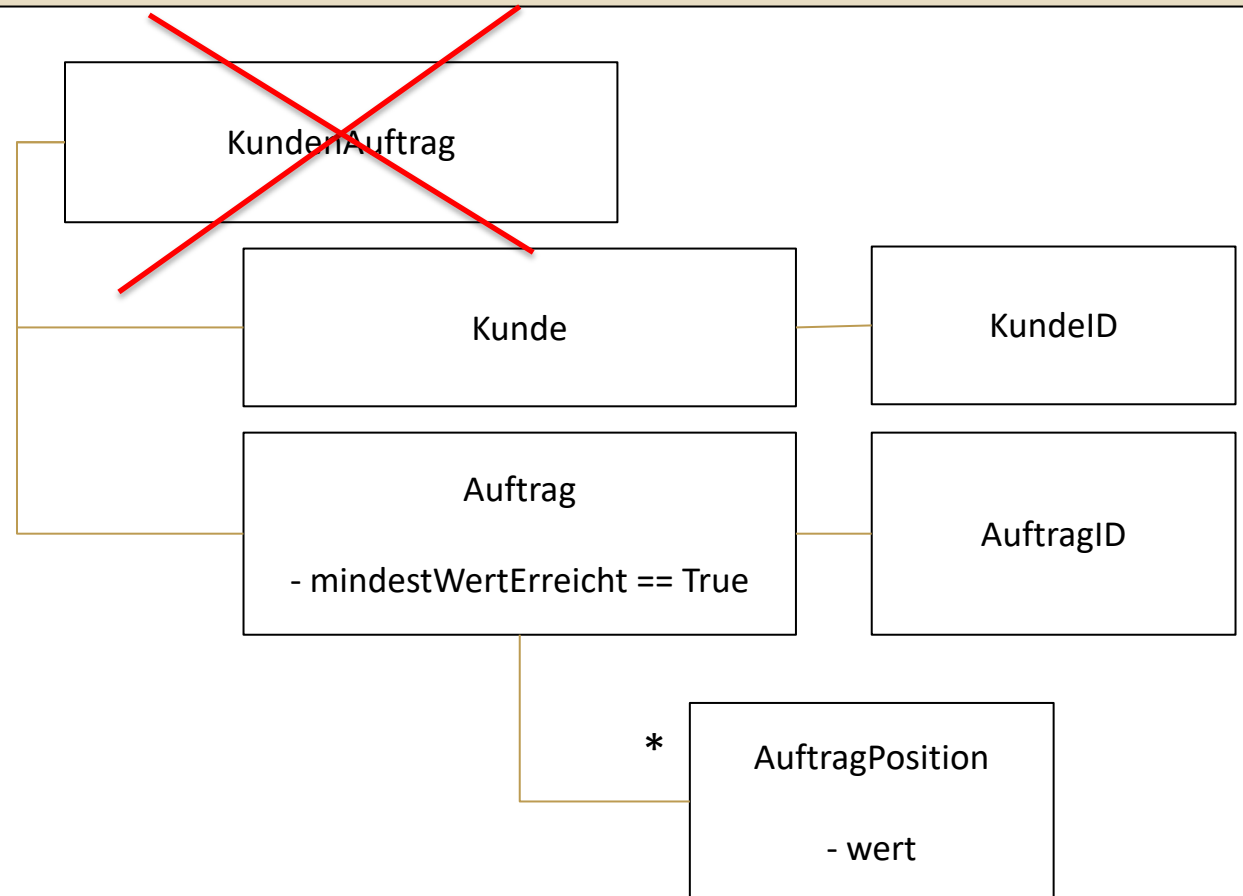


Fachliche Invariante basierend auf Geschäftsregel:  
Auftrag darf nur dann angenommen werden, wenn der Mindestwert erreicht ist.

Ein Aggregate genügt dem Single-Responsability-Principle (SRP)

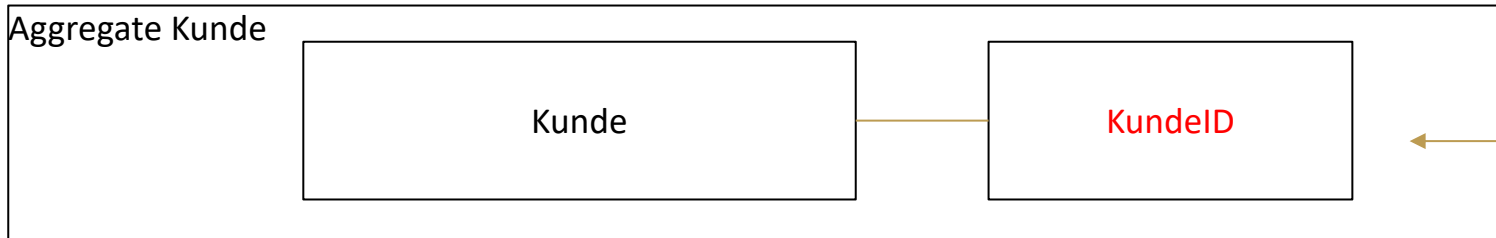
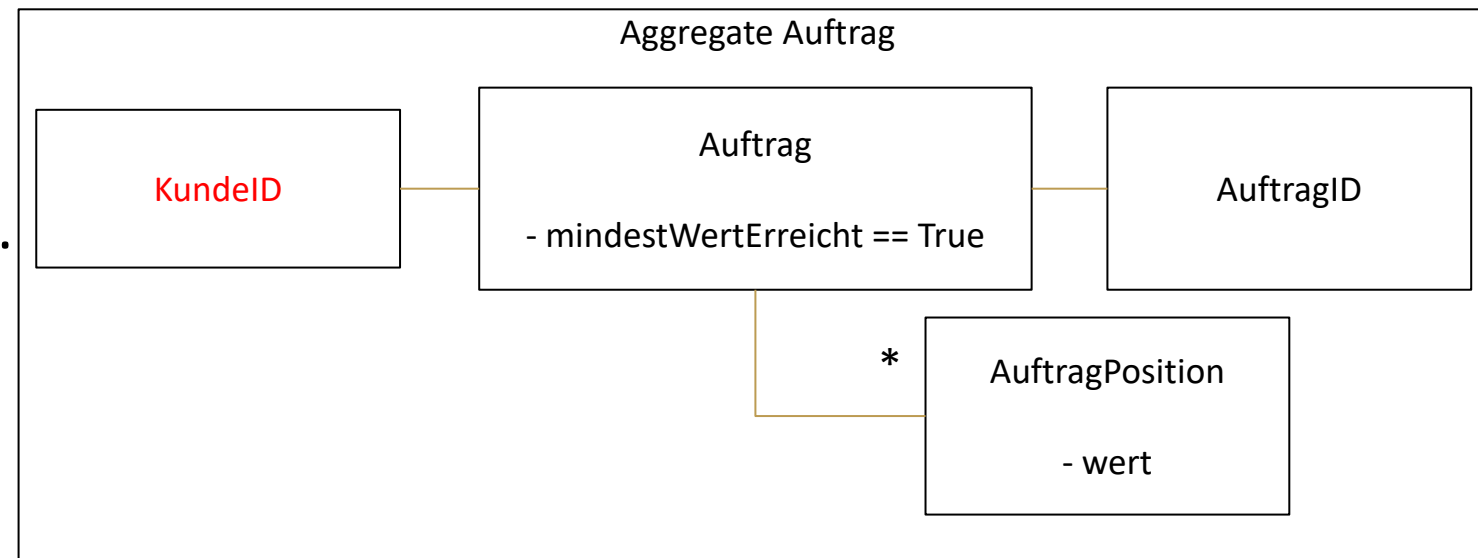
SRP: there should be only **one** reason for change

Entwerfe kleine Aggregates!



## Referenziere Aggregates NUR über ihre Identität

- Die Identität ist ein Value Object.
- Im Modell und im Code ist **keine** direkte Objekt Referenz vorhanden.
- Vorteil: Persistenz kann auf verschiedene Weisen implimentiert werden:  
SQL, JSON, Key-Value, ...

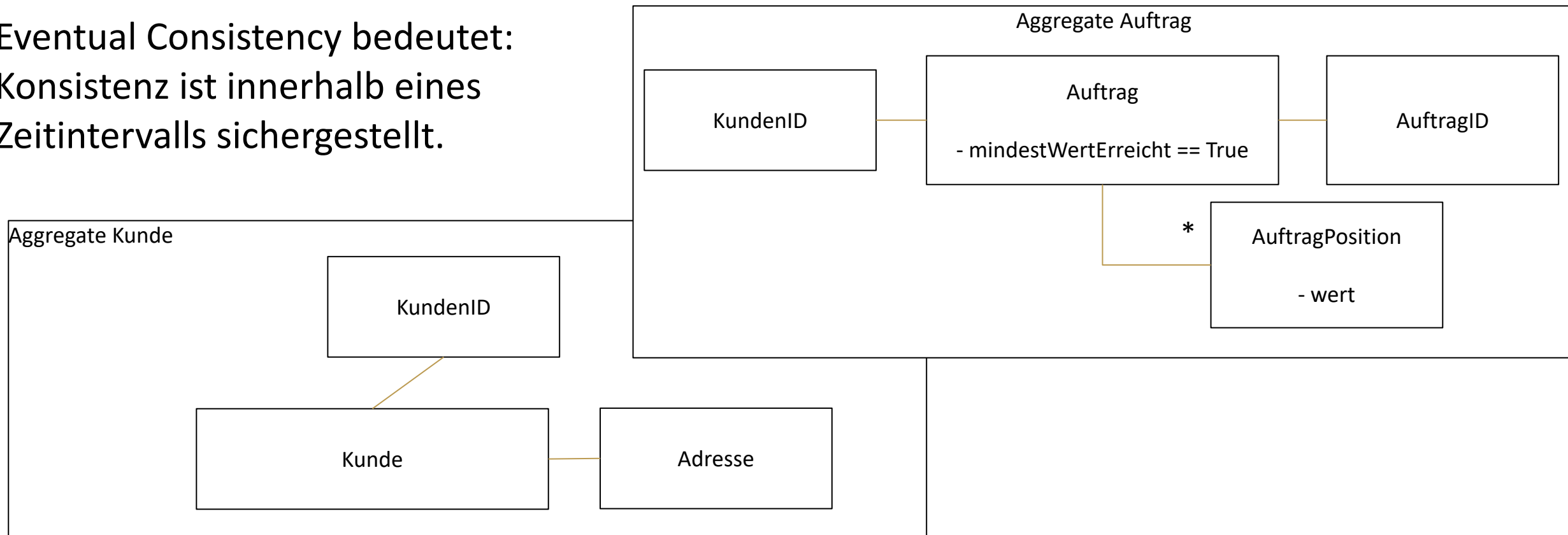


Zugriff erfolgt immer über  
die Identität der Root-Entity!



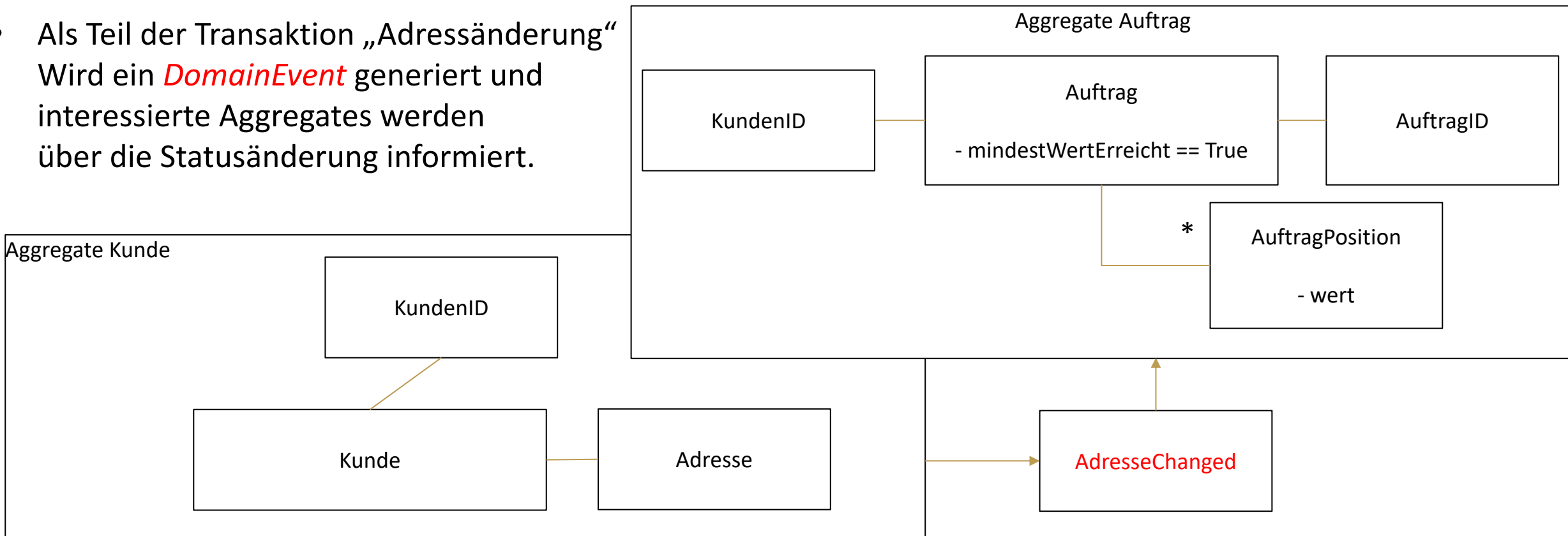
## Aktualisiere andere Aggregates unter Verwendung von Eventual Consistency

Eventual Consistency bedeutet:  
Konsistenz ist innerhalb eines  
Zeitintervalls sichergestellt.



## Aktualisiere andere Aggregates unter Verwendung von Eventual Consistency

- Als Teil der Transaktion „Adressänderung“  
Wird ein *DomainEvent* generiert und interessierte Aggregates werden über die Statusänderung informiert.



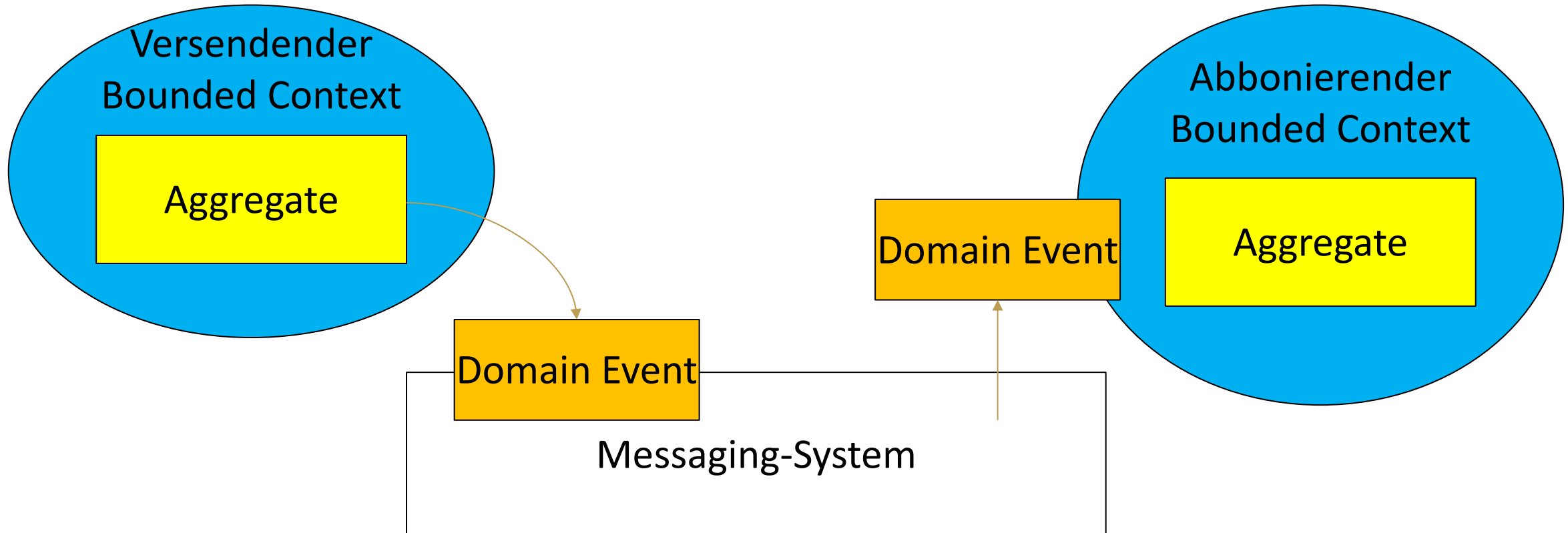
- Was könnte zusammen in einem Aggregate modelliert werden und was besser nicht?
  - Was für DomainEvents fallen uns dazu ein?
- 
- Kunde
  - Auftrag
  - Auftragsposition
  - Lieferadresse
  - Rechnung
  - Produkt
  - Menge und Wert
- Zahlungskondition
  - Rechnungsanschrift
  - Lagerbestand
  - Auftragsposition
  - Versandkosten
  - Mehrwertsteuer
  - ....

# TAKTISCHES DESIGN MIT DOMAIN EVENTS

# DOMAIN EVENTS

Ein Domain Event ist die Verkörperung eines **geschäftsrelevanten Ereignisses** in einem Bounded Context

Ein Domain Event bildet die Schnittstelle zur Außenwelt.

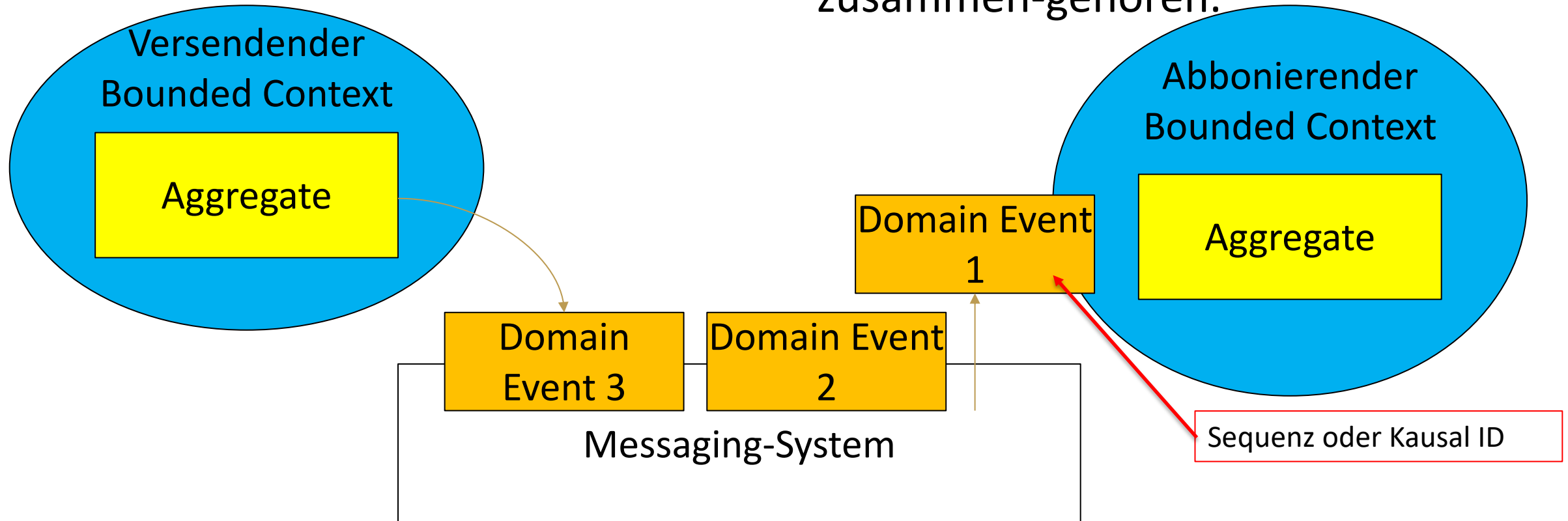


# DOMAIN EVENTS

**Causal Consistency:** Kausal zusammenhängende Operationen müssen in einer bestimmten Reihenfolge auftreten.

Domain Events haben eine Reihenfolge!

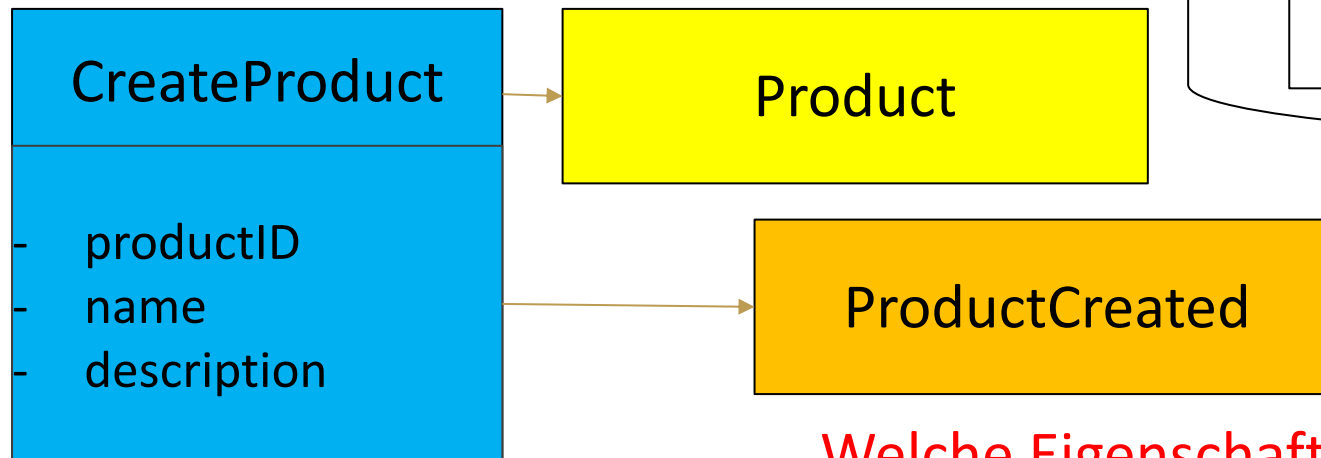
Event 2 darf nicht vor Event 1 abgearbeitet werden, wenn sie kausal zusammengehören.



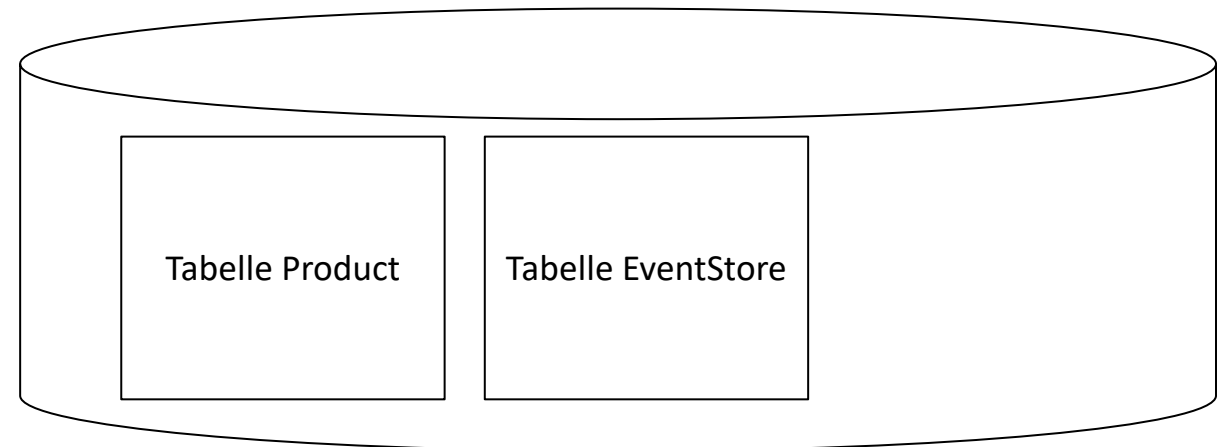
# DOMAIN EVENTS

Domain Events haben *Trigger (Ursachen)*:

- Zeitlicher Trigger
- *Command* (Objektform einer Methode): Commands werden auf Aggregates ausgeführt.



! Modifiziertes Aggregate und Domain Event bilden eine Transaktionsklammer.



Beispiel Persistenz in SQL Datenbank

Welche Eigenschaften des Produktes sollen in das Domain Event übernommen werden?

# ZEIT ALS WEITERE DIMENSION



# EVENT SOURCING

- **Event Sourcing** ist das Speichern aller Domain Events, die für eine Aggregate-Instanz aufgetreten sind.
- Der Zustand der Aggregate-Instanz zu einem bestimmten Zeitpunkt wird berechnet durch das „Abspulen aller Events“, die bis zum Zeitpunkt ereignet haben.
- Alle Events zu einer Aggregate-Instanz bilden in ihrer zeitlichen Reihenfolge einen *Event Stream*.

Auftrag

AuftragPositionAdded

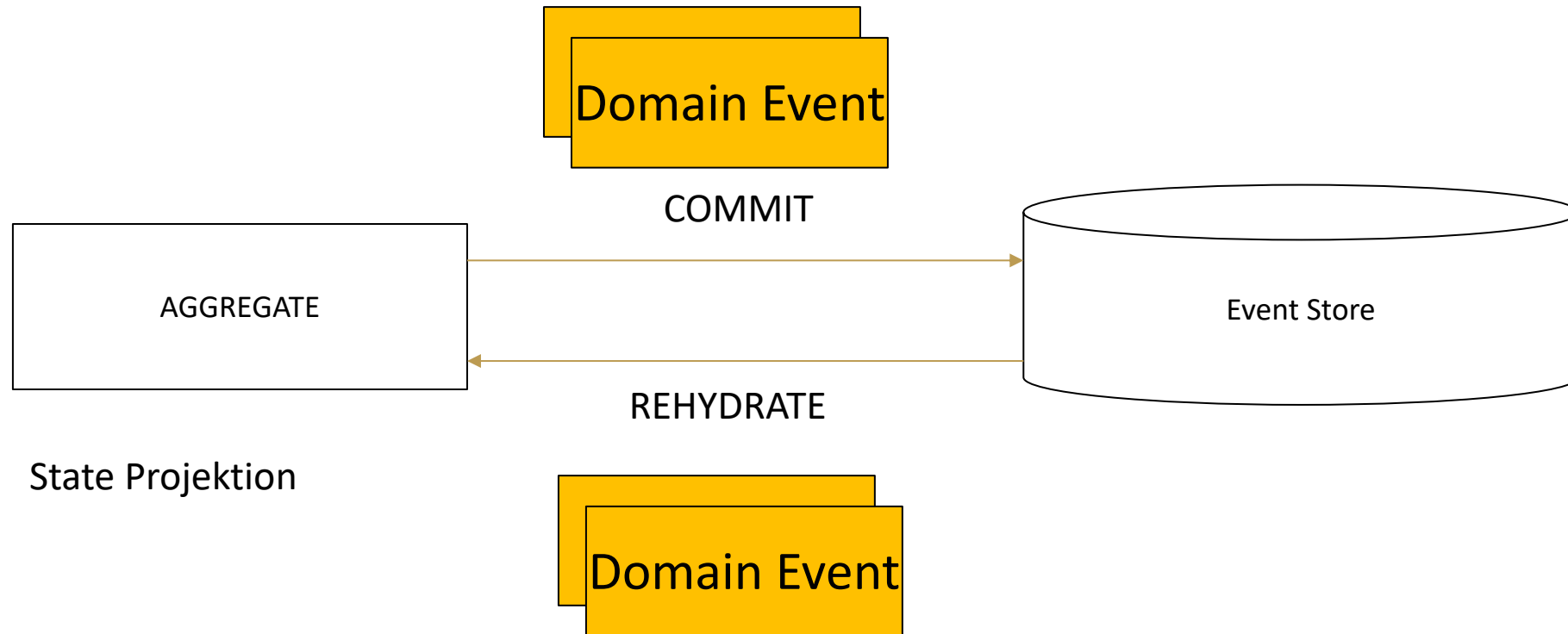
- nr
- quantity, value
- productID

.....

AuftragPositionAdded

- nr
- quantity, value
- productID

# EVENT STORE



**DOMAIN SERVICE**

**NOT EVERYTHING IS AN OBJECT**

Evans discusses the idea of (Domain) SERVICE operations that don't have a natural home in an entity or value object.

A good SERVICE has three characteristics:

- ❑ The operation relates to a domain concept that is not natural part of an ENTITY or VALUE OBJECT
- ❑ The interface is defined in terms of other elements of the domain model
- ❑ The operation is stateless

*Sometimes,  
it just isn't a thing.*

*- Eric Evans 2003*

Domain SERVICES are not the same thing as the services from the service layer (in a layered architecture), but they are often closely related.

A domain SERVICE represents a business concept or process, whereas a service-layer service represents a use case for your application. Often the service layer will call a domain service.

# REPOSITORY

**SEPARATE THE BUSINESS  
LOGIC FROM TECHNICAL  
CONCERNS**

## Definition:

- Das Repository abstrahiert den Zugriff auf persistente Daten. Ein Repository dient als eine Art Vermittler zwischen der Domain-Schicht (die Geschäftslogik enthält) und der Infrastrukturschicht (z. B. Datenbanken oder externe Systeme).
- Repository und Aggregate haben in der Regel eine 1:1 Beziehung. Es ermöglicht die Speicherung, Abruf und Verwaltung eines Aggregate.
- Abstraktion der Datenzugriffsschicht: Damit trennt man die Geschäftslogik von den technischen Details der Datenpersistenz. Das Repository versteckt die Details der Datenbankabfragen oder des Dateimanagements.

Typische Methoden sind:

- `save(AggregateRoot)`, `findById(id)`, `findAll()`, `deleteById(id)` ...

# DESIGN WITH CANVAS




# AGGREGATES

## □ Aggregate Design Canvas

| Aggregate Design Canvas       |                      |                                                                                                                                                                                                                                                                                                                                                                                                                          |  |         |         |                       |                      |                      |                               |                      |                      |                             |                      |                      |
|-------------------------------|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|---------|---------|-----------------------|----------------------|----------------------|-------------------------------|----------------------|----------------------|-----------------------------|----------------------|----------------------|
| 1. Name                       | 3. State Transitions |                                                                                                                                                                                                                                                                                                                                                                                                                          |  |         |         |                       |                      |                      |                               |                      |                      |                             |                      |                      |
| 2. Description                |                      |                                                                                                                                                                                                                                                                                                                                                                                                                          |  |         |         |                       |                      |                      |                               |                      |                      |                             |                      |                      |
| 4. Enforced Invariants        | 6. Handled Commands  | 8. Throughput <table><thead><tr><th></th><th>Average</th><th>Maximum</th></tr></thead><tbody><tr><td>Command handling rate</td><td><input type="text"/></td><td><input type="text"/></td></tr><tr><td>Total number of clients</td><td><input type="text"/></td><td><input type="text"/></td></tr><tr><td>Concurrency conflict chance</td><td><input type="text"/></td><td><input type="text"/></td></tr></tbody></table> |  | Average | Maximum | Command handling rate | <input type="text"/> | <input type="text"/> | Total number of clients       | <input type="text"/> | <input type="text"/> | Concurrency conflict chance | <input type="text"/> | <input type="text"/> |
|                               | Average              | Maximum                                                                                                                                                                                                                                                                                                                                                                                                                  |  |         |         |                       |                      |                      |                               |                      |                      |                             |                      |                      |
| Command handling rate         | <input type="text"/> | <input type="text"/>                                                                                                                                                                                                                                                                                                                                                                                                     |  |         |         |                       |                      |                      |                               |                      |                      |                             |                      |                      |
| Total number of clients       | <input type="text"/> | <input type="text"/>                                                                                                                                                                                                                                                                                                                                                                                                     |  |         |         |                       |                      |                      |                               |                      |                      |                             |                      |                      |
| Concurrency conflict chance   | <input type="text"/> | <input type="text"/>                                                                                                                                                                                                                                                                                                                                                                                                     |  |         |         |                       |                      |                      |                               |                      |                      |                             |                      |                      |
| 5. Corrective Policies        | 7. Created Events    | 9. Size <table><thead><tr><th></th><th>Average</th><th>Maximum</th></tr></thead><tbody><tr><td>Event growth rate</td><td><input type="text"/></td><td><input type="text"/></td></tr><tr><td>Lifetime of a single instance</td><td><input type="text"/></td><td><input type="text"/></td></tr><tr><td>Number of events persisted</td><td><input type="text"/></td><td><input type="text"/></td></tr></tbody></table>      |  | Average | Maximum | Event growth rate     | <input type="text"/> | <input type="text"/> | Lifetime of a single instance | <input type="text"/> | <input type="text"/> | Number of events persisted  | <input type="text"/> | <input type="text"/> |
|                               | Average              | Maximum                                                                                                                                                                                                                                                                                                                                                                                                                  |  |         |         |                       |                      |                      |                               |                      |                      |                             |                      |                      |
| Event growth rate             | <input type="text"/> | <input type="text"/>                                                                                                                                                                                                                                                                                                                                                                                                     |  |         |         |                       |                      |                      |                               |                      |                      |                             |                      |                      |
| Lifetime of a single instance | <input type="text"/> | <input type="text"/>                                                                                                                                                                                                                                                                                                                                                                                                     |  |         |         |                       |                      |                      |                               |                      |                      |                             |                      |                      |
| Number of events persisted    | <input type="text"/> | <input type="text"/>                                                                                                                                                                                                                                                                                                                                                                                                     |  |         |         |                       |                      |                      |                               |                      |                      |                             |                      |                      |

Aggregate Design Canvas v1.1 designed by Kacper Gunia & Domain-Driven Design Crew.  
For more information visit <https://github.com/ddc-crew/aggregate-design-canvas>.

This work is licensed under a Creative Commons Attribution 4.0 International License, CC BY 4.0.  
To view a copy of the license, visit <https://creativecommons.org/licenses/by/4.0/>.



<https://github.com/ddc-crew/aggregate-design-canvas>



# BOUNDED CONTEX

## □ Bounded Context CANVAS

|                                                                                                                                      |                                                                                                                                                             |                                                                                                                                                  |
|--------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Name</b>                                                                                                                          | <b>Model Traits</b><br>draft, execute, audit, enforcer, interchange, gateway, etc.                                                                          |                                                                                                                                                  |
| <b>Description</b><br>Summary of purpose and responsibilities                                                                        | <b>Messages Consumed and Produced</b>                                                                                                                       |                                                                                                                                                  |
|                                                                                                                                      | <b>Messages Consumed</b>                                                                                                                                    | <b>Messages Produced</b>                                                                                                                         |
|                                                                                                                                      | Commands Handled                                                                                                                                            | Commands Sent                                                                                                                                    |
|                                                                                                                                      | Events Handled                                                                                                                                              | Events Published                                                                                                                                 |
|                                                                                                                                      | Queries Handled                                                                                                                                             | Queries Invoked                                                                                                                                  |
| <b>Strategic Classification</b>                                                                                                      |                                                                                                                                                             |                                                                                                                                                  |
| <div>Domain</div> <ul style="list-style-type: none"> <li>- Core</li> <li>- Supporting</li> <li>- Generic</li> <li>- Other</li> </ul> | <div>Business Model</div> <ul style="list-style-type: none"> <li>- Revenue</li> <li>- Engagement</li> <li>- Compliance</li> <li>- Cost reduction</li> </ul> | <div>Evolution</div> <ul style="list-style-type: none"> <li>- Genesis</li> <li>- Custom built</li> <li>- Product</li> <li>- Commodity</li> </ul> |
| <b>Business Decisions</b><br>Key business rules, policies, and decisions                                                             | <b>Dependencies and Relationships</b>                                                                                                                       |                                                                                                                                                  |
|                                                                                                                                      | <b>Message Suppliers</b><br>Name                      Relationship                                                                                          | <b>Message Consumers</b><br>Name                      Relationship                                                                               |
| <b>Ubiquitous Language</b><br>Key domain terminology                                                                                 |                                                                                                                                                             |                                                                                                                                                  |

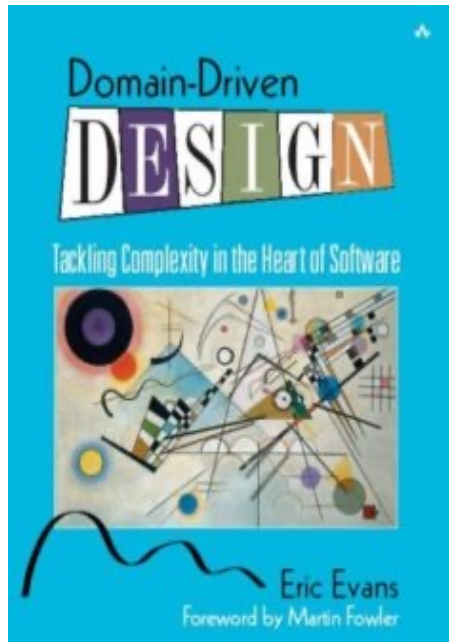
Bounded Context Canvas V3 Nick Tune | ntcoding.co.uk

<https://github.com/ddc-crew/bounded-context-canvas>

# FURTHER READING

# DDD LITERATURE

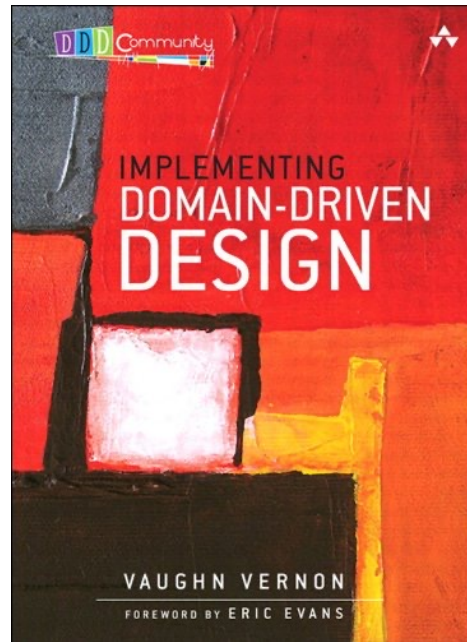
## The Big Blue Book



2003 - Eric Evans  
“Domain Driven Design”

*“not easy to read”*

## The Big Red Book



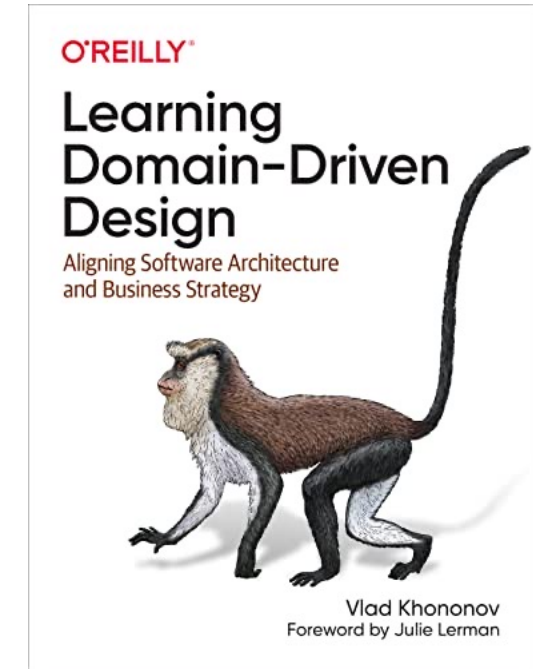
2013 - Vernon Vaughn  
“Implementing Domain  
Driven Design”

“Fokus auf State of the  
Art (2013) architecture  
styles”



2017 - Vernon Vaughn  
“Domain Driven Design -  
kompakt”

“deutsche Übersetzung  
von C. Lilienthal –  
kompakte Einführung”

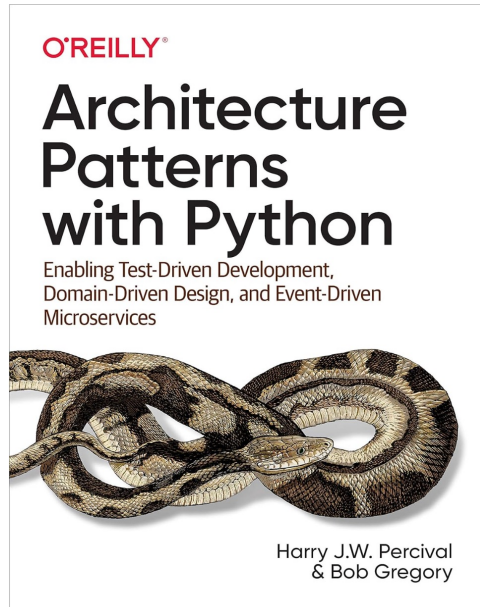


2022 – Vlad Khononov  
“Learning Domain Driven  
Design”

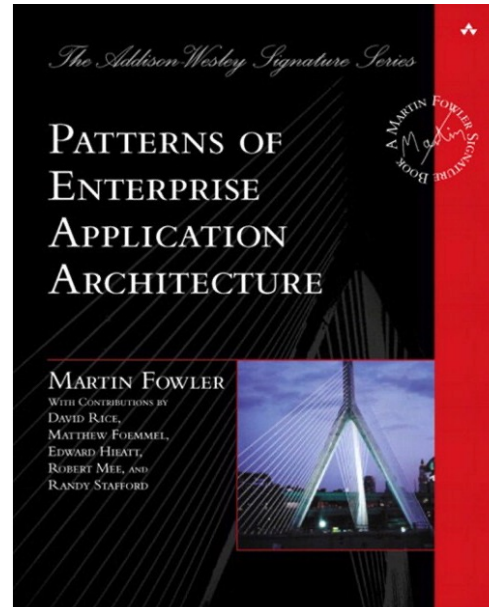
“adopted DDD to modern  
design principles”

# DDD LITERATURE

## Focus on Python



2020 – Percival & Gregory  
“Architecture Patterns  
with Python”



2002 – Martin Fowler  
“Patterns of Enterprise  
Application Architecture”

*“covers a lot of modern  
architecture pattern on  
top of DDD”*