Spring Data

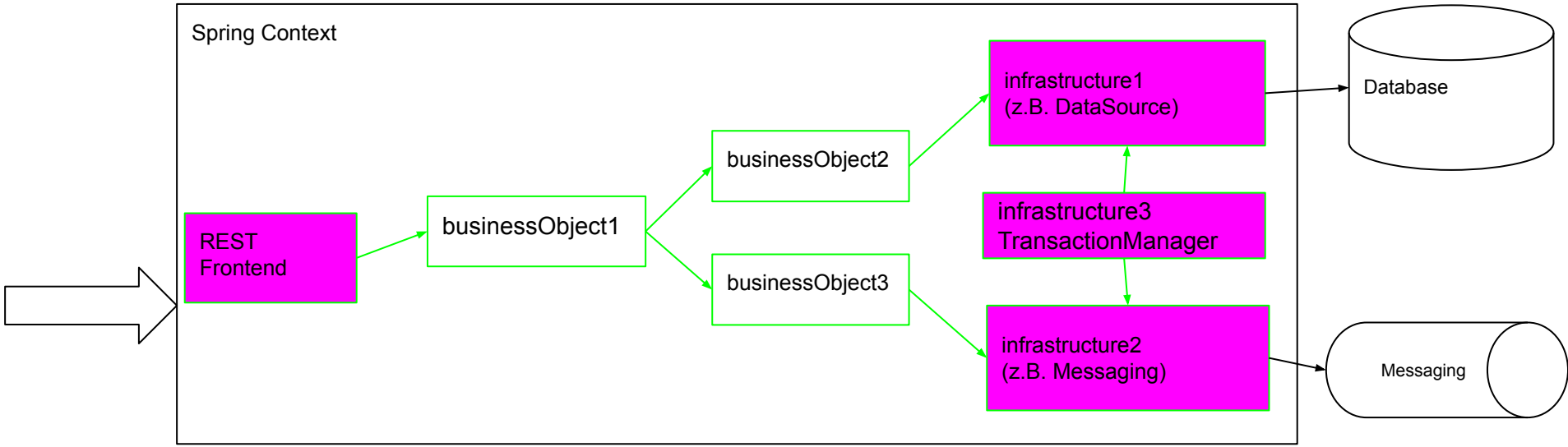
Vorstellungsrunde

- Name und die Rolle im Unternehmen
- Themenbezogene Vorkenntnisse
- Aktuelle Problemsituation
- Individuelle Zielsetzung

Ausgangssituation

Spring und Spring Boot

integrata cegos



Context Configuration
- XML
- Annotations
- JavaConfig

Auto Configuration

Initialer Projektstand

- https://github.com/Javacream/org.javacream.training.spring
  - Branch init_core_rest
- Übersicht Anwendung
  - StoreService
    - Bestandsabfragen
      - “Wie viele Items einer bestimmten Kategorie sind auf Lager”
  - BooksService
    - Verwaltet Bücher
      - Buch: ISBN (id), title, price, available
      - CRUD-Operationen
      - Eindeutige IDs werden vom IsbnGenerator erzeugt
- Übersicht Design
  - api mit Interfaces und Datentypen
  - impl mit Implementierungen
  - web mit Rest-Services

Spring Recap

integrata cegos

- In der Anwendung werden primär Spring-Context-Annotations benutzt
  - @Component, @Service, @Repository, @RestController
  - Jede Klasse, die so annotiert ist, ist für den Context relevant
    - Instanzierung + Lifecycle wird von Spring übernommen
- Jede Spring-Komponente bekommt eine eindeutige Identifizierung
  - Java-Typsystem
    - Klassenname, Schnittstelle sowie @Qualifier
  - Namen
    - Standard-Name ist der Klassenname ohne Paket, erster Buchstabe klein
- Dependencies: @Autowired
  - Gültig für Attribute, Konstruktor-Parameter, Bean-Methoden (später)
  - @Autowired funktioniert nur, wenn auf Grund des Typsystems (inklusive Qualifier) eine eindeutige Auflösung möglich ist

Spring Recap Teil 2

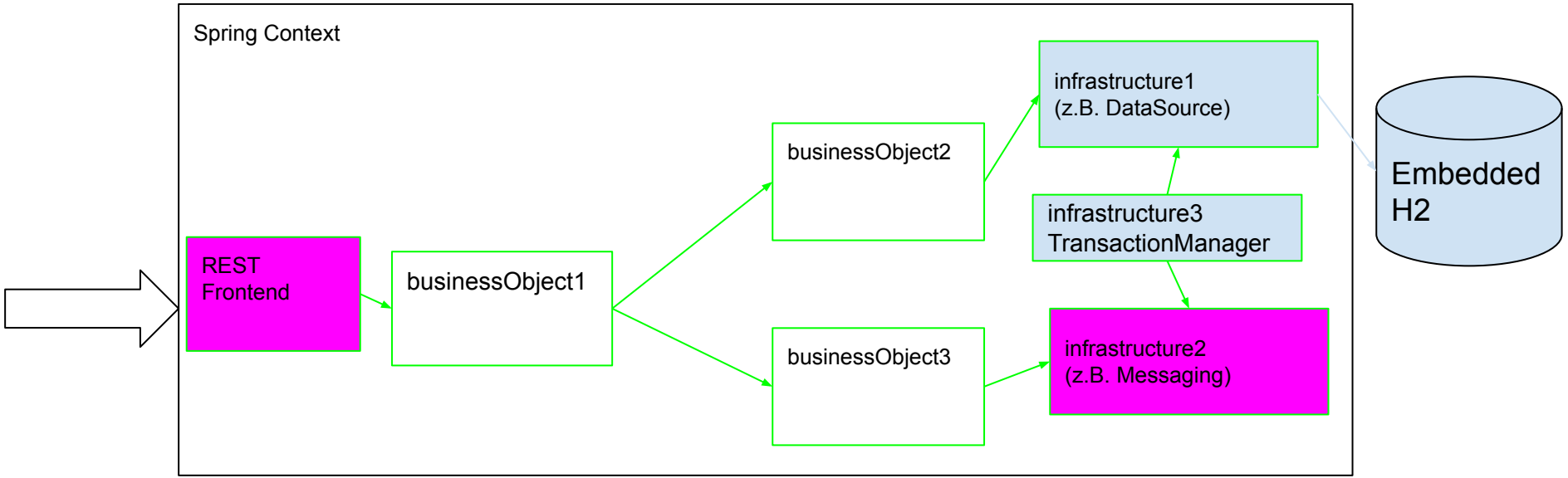
integrata cegos

- Konfiguration
  - application.properties
    - Standard-Name einer SpringBoot-Application “application”
  - application.yaml, application.yml
- Konfigurationseinstellungen von Spring Boot
- Eigene Konfigurationseinstellungen können jederzeit ergänzt werden
- Auslesen:
  - @Value(“${property}”)
  - @Value(“${isbngenerator.prefix}”)
- Profiles
- @Configuration-Klassen
  - @Bean-Methoden
  - Träger weiterer Spring-Annotationen, z.B. @PropertySource

Datenzugriff mit Spring Boot und Spring Data

Spring und Spring Boot und Spring Data

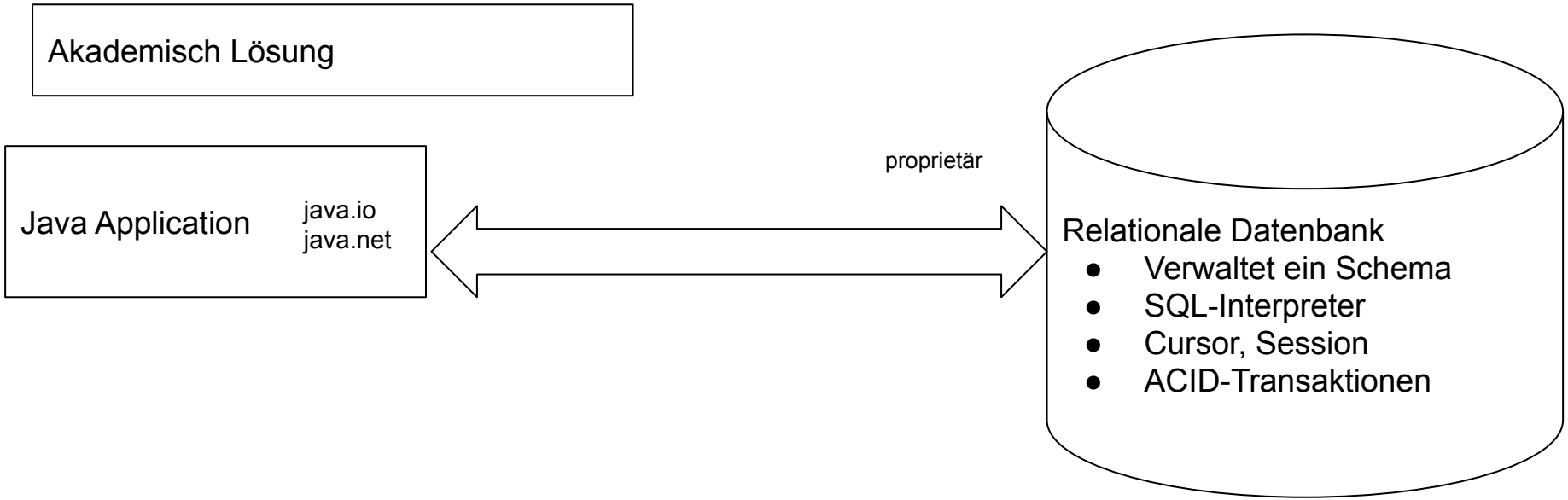
integrata cegos



Context Configuration
- XML
- Annotations
- JavaConfig

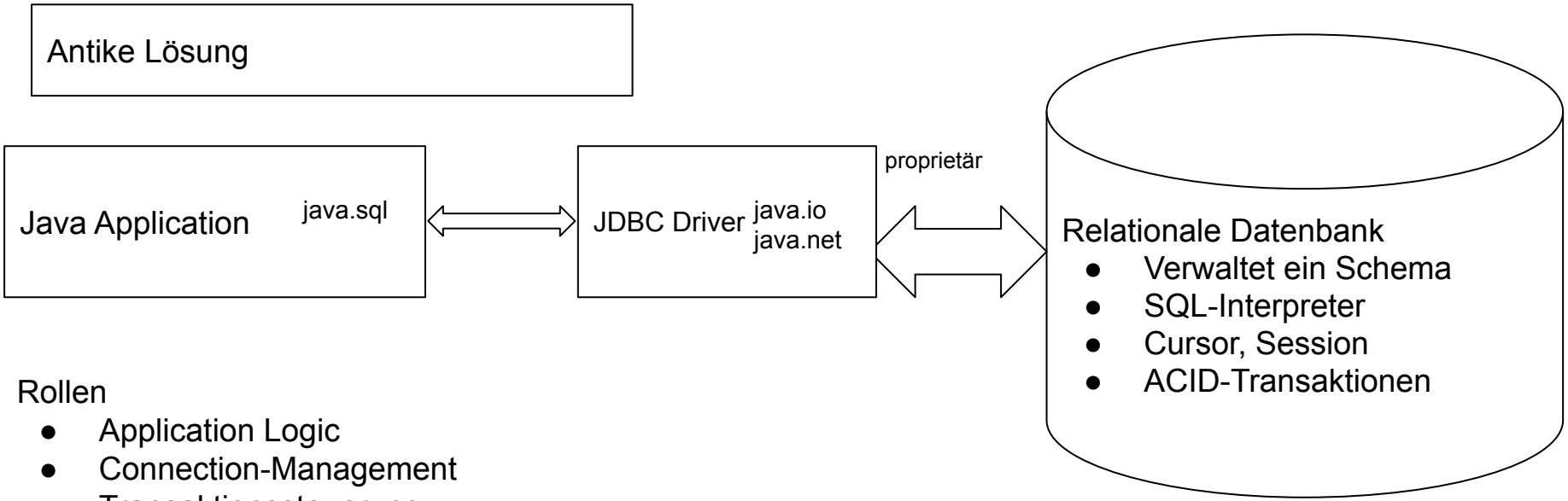
Auto Configuration

Datenbank-Zugriffe in Java



Atomar
Consistent
Isolated
Durable

Datenbank-Zugriffe in Java

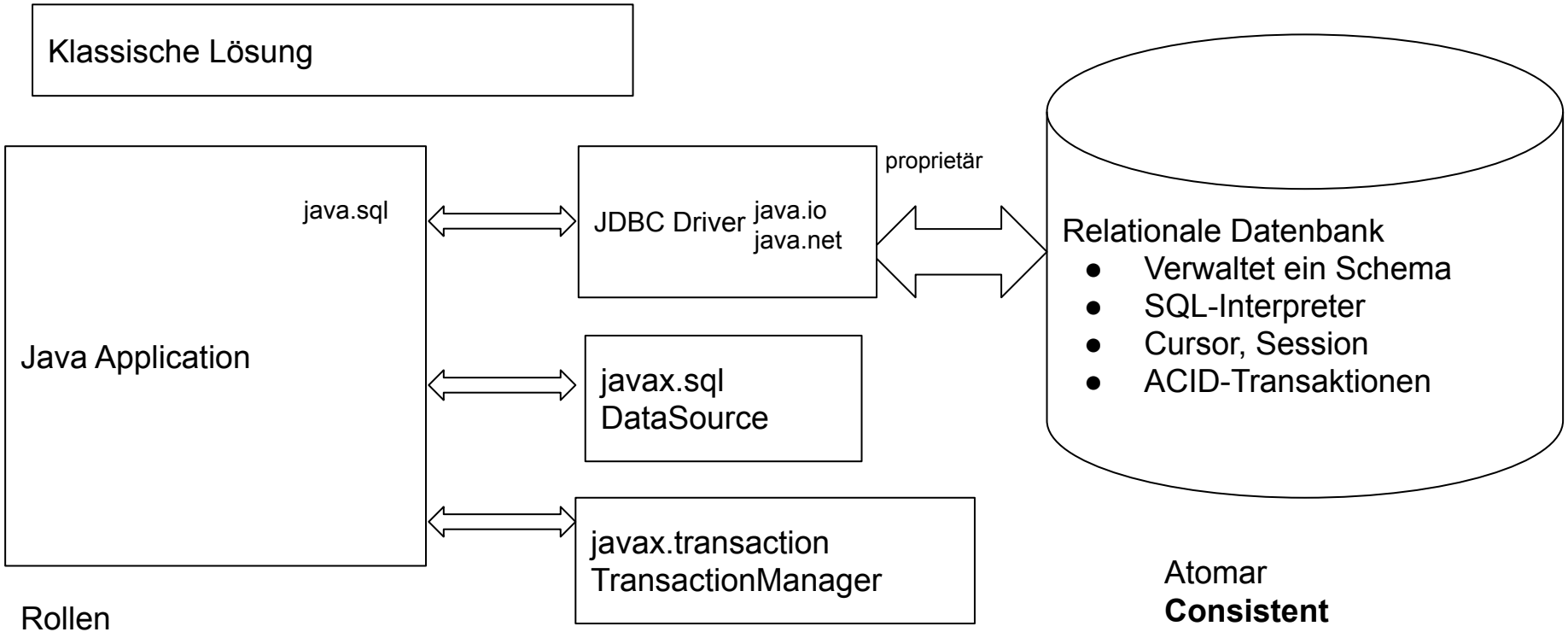


Rollen

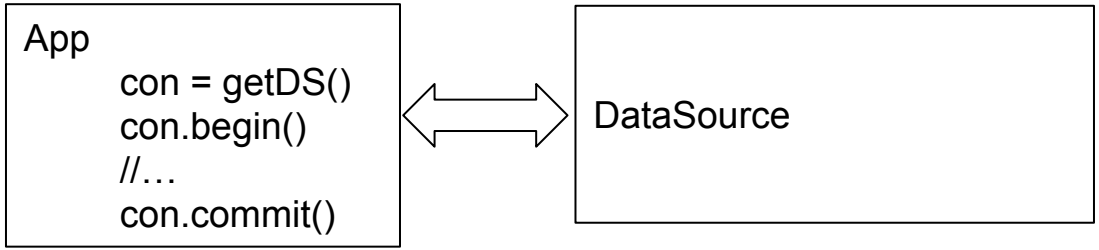
- Application Logic
- Connection-Management
- Transaktionssteuerung

Atomar
Consistent
Isolated
Durable

Datenbank-Zugriffe in Java

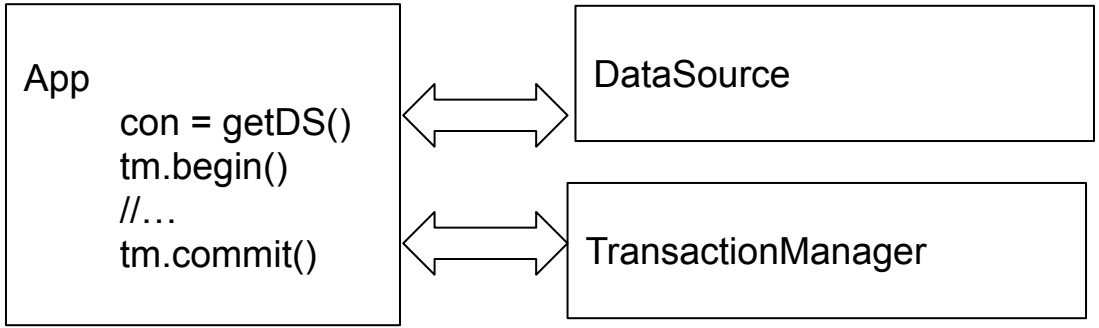


Exkurs: Warum der TransactionManager?



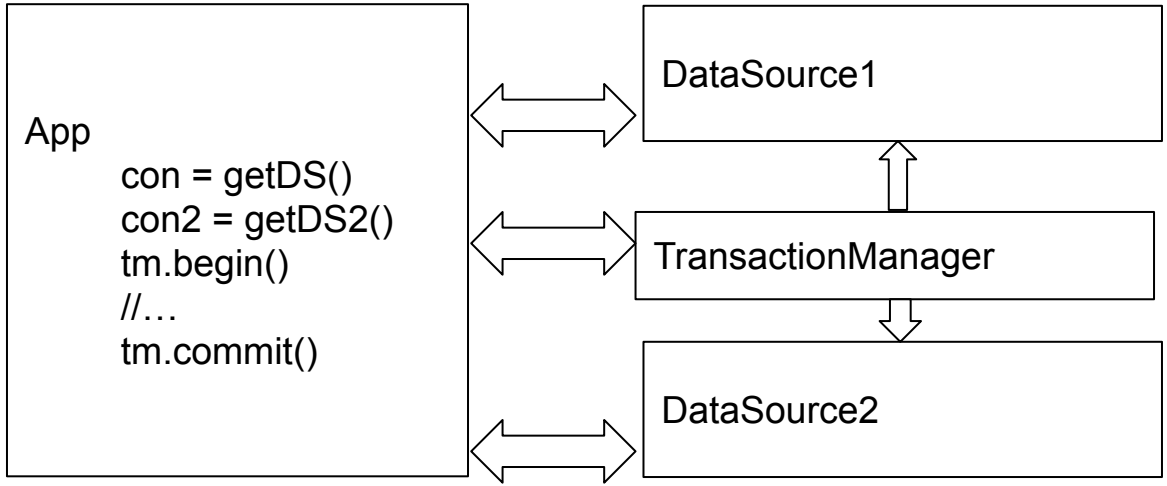
ohne

so finde ich (Sawitzki) keine gute Begründung…



mit

Exkurs: Warum der TransactionManager?

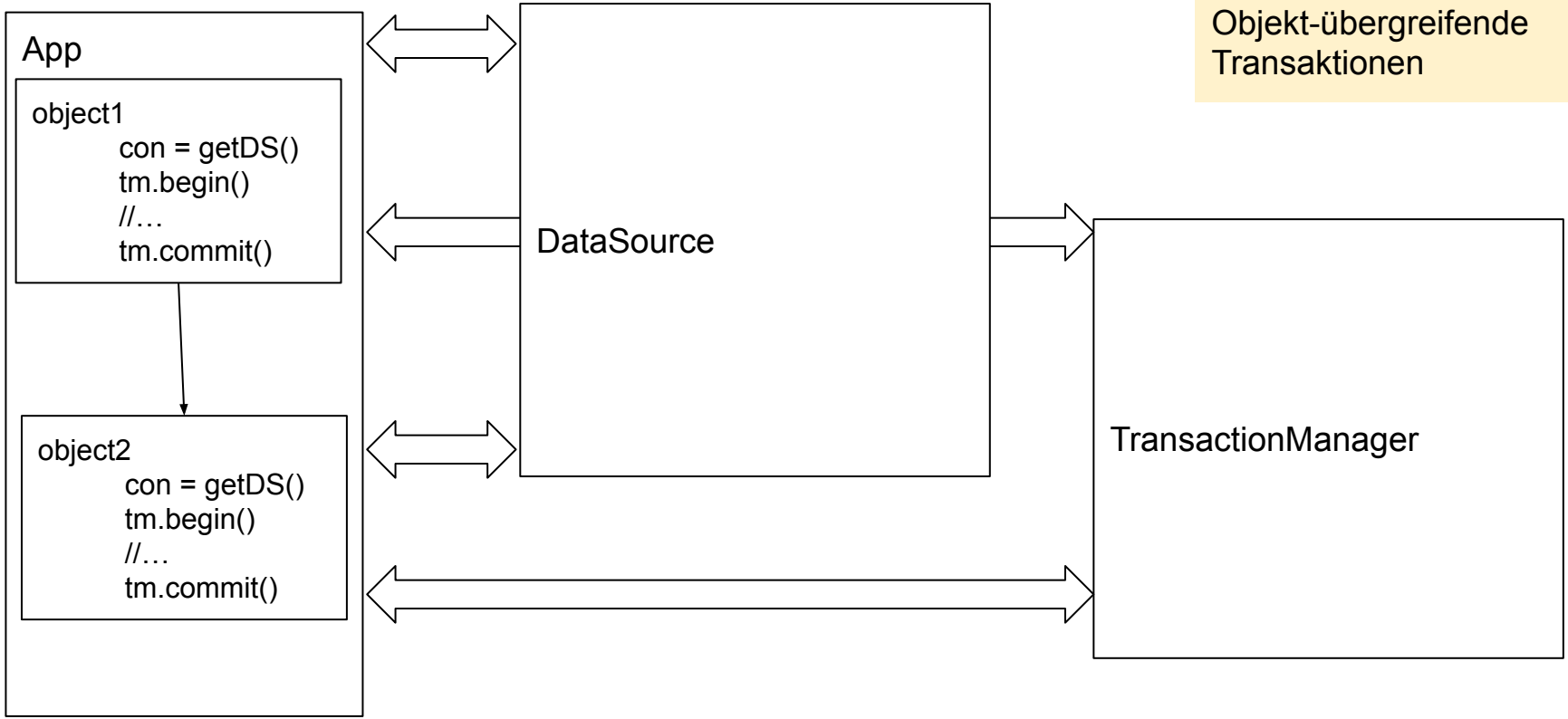


Orchestrierung der commits an Hand des Two-Phase-Commit-Protokolle

```
for (){
        prepareCommit
}
commit()
```

Exkurs: Warum der TransactionManager?

integrata cegos

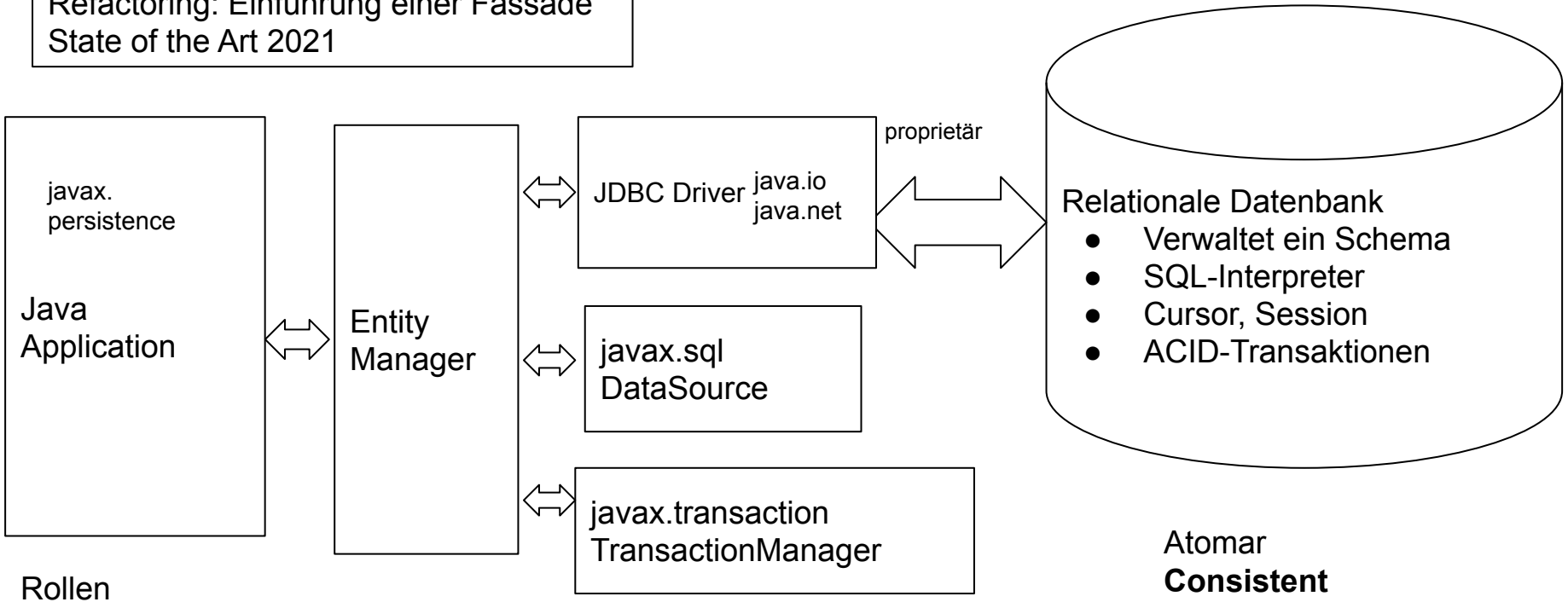


Objekt-übergreifende Transaktionen

Datenbank-Zugriffe in Java

integrata cegos

Refactoring: Einführung einer Fassade
State of the Art 2021



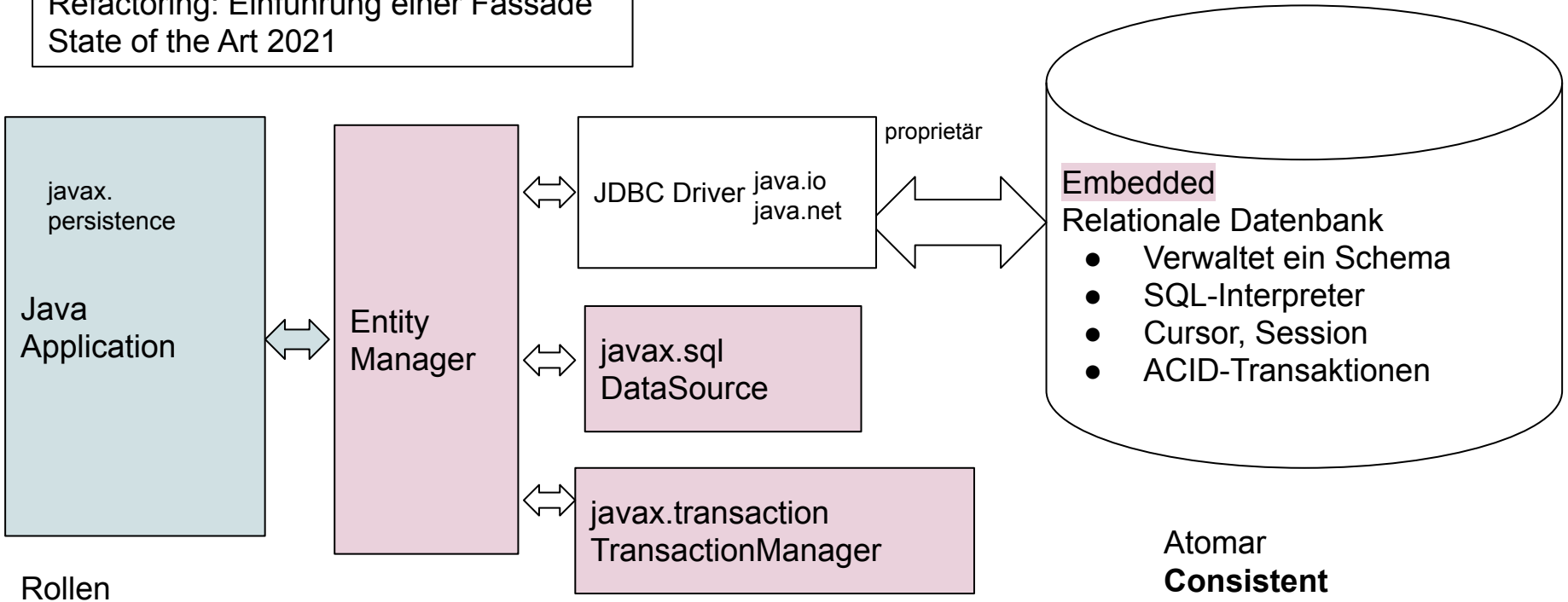
Rollen

- Application Logic

Atomar
Consistent
Isolated
Durable

integrata cegos

Refactoring: Einführung einer Fassade
State of the Art 2021



Rollen
- Application Logic

Atomar
Consistent
Isolated
Durable

Programmierung mit Spring Data JPA

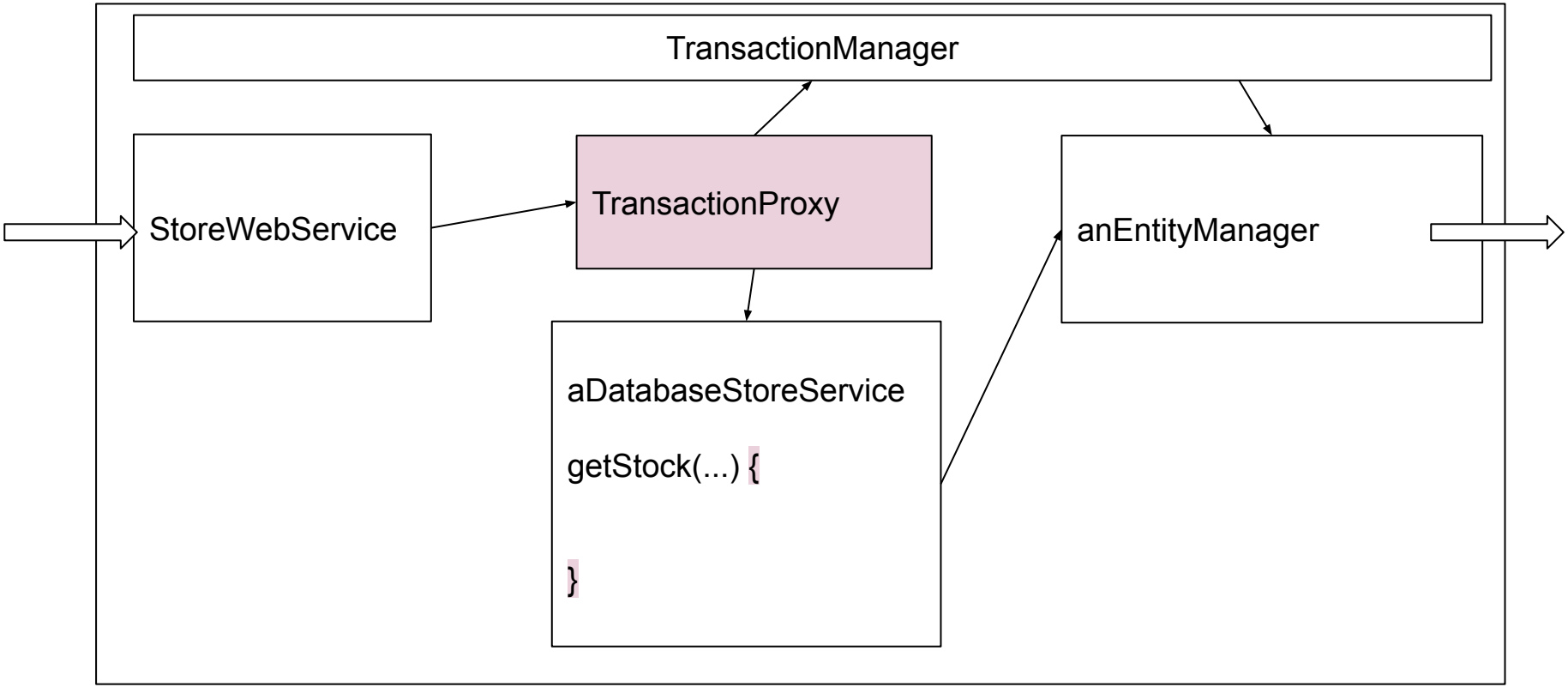
Vorbemerkung

- Im nächsten Beispiel benutzen wir den EntityManager direkt
  - eigentlich JPA, und damit eigentlich nicht Spring Data
- Trotzdem keine Thema-Verfehlung
  - Spring Data arbeitet immer mit einer Form des so genannten O/R-Mappings
  - Für die Store-Anwendung komplett oversized

Übersicht Transaktions-Management

Spring Context Revisited mit @Transactional

integrata cegos



Wie arbeitet der TransactionProxy?

integrata cegos

- Kommt drauf an…
  - und zwar auf die Transaction Propagation
- 6 Stück (+ 1 von Spring)
  - REQUIRES_NEW
  - REQUIRED
  - MANDATORY
  - NOT_SUPPORTED
  - NEVER
  - SUPPORTS
  - + Spring-Erweiterung
    - NESTED

Wie arbeitet der TransactionProxy?

integrata cegos

- Wichtigkeit
- 6 Stück (+ 1 von Spring)
  - **REQUIRES_NEW**
  - **REQUIRED**
  - MANDATORY
  - NOT_SUPPORTED
  - NEVER
  - SUPPORTS
  - + Spring-Erweiterung
    - NESTED

@Transactional(propagation REQUIRES_NEW)

integrata cegos

{

- Erzeuge eine neue Transaktion
  - Falls bereits eine Transaktion vorhanden ist wird diese “zur Seite gelegt”

}

```
catch{RuntimeException e){
    tm.rollback()
}
tm.commit()
```

@Transactional(propagation REQUIRED)

{

- Prüfe: Transaktion vorhanden
  - Falls Nein: Neue Transaktion anlegen und merken “ich habe eine neue Transaktion erstellt, created=true”
  - Falls ja: Nichts passiert

}

```
catch{RuntimeException e){
    if (created){tm.rollback()}
    else{tm.setRollbackOnly()}
} if (created){tm.commit()}else (nichts zu tun)
```

Spring Data

Übersicht

- “Umbrella-Projekt”
  - Es ist nicht Ziel von Spring Data, alle Datenzugriffs-Technologien zu vereinheitlichen
    - “Java Connector Architecture” hier als schlechtes Beispiel
  - Ein Anwendung mit z.B. Spring Data JPA umzustellen auf eine z.B. Spring Data Couchbase Variante erfordert immer massive Umprogrammierung
  - Projekt-Übersicht
    - Supported Sub-Projekte als Bestandteil der Spring Data Distribution
    - Spring-Community-Subprojekte
    - Community- bzw. Hersteller-Subprojekte

Übersicht: Spring Data Commons

integrata cegos

- Alle gehaltenen Daten benötigen eine eindeutige Id
  - Zugriffe z.B. auf eine Tabelle ohne Primary Key geht nicht
- Operationen auf Daten sind CRUD-Operationen
- O/X-Mapping wird eingeführt
  - bestes Beispiel: O/R-Mapping
  - Datenmodell wird mit einem Objekt-Modell synchronisiert
- Programmiermodell ist ausgerichtet auf Schnittstellen (“Repositories”), die im Endeffekt von Spring Data zur Laufzeit implementiert werden
  - Das funktioniert nur, wenn Meta-Informationen, Java-Annotations beigesteuert werden

Einführung des JPA-Repositories

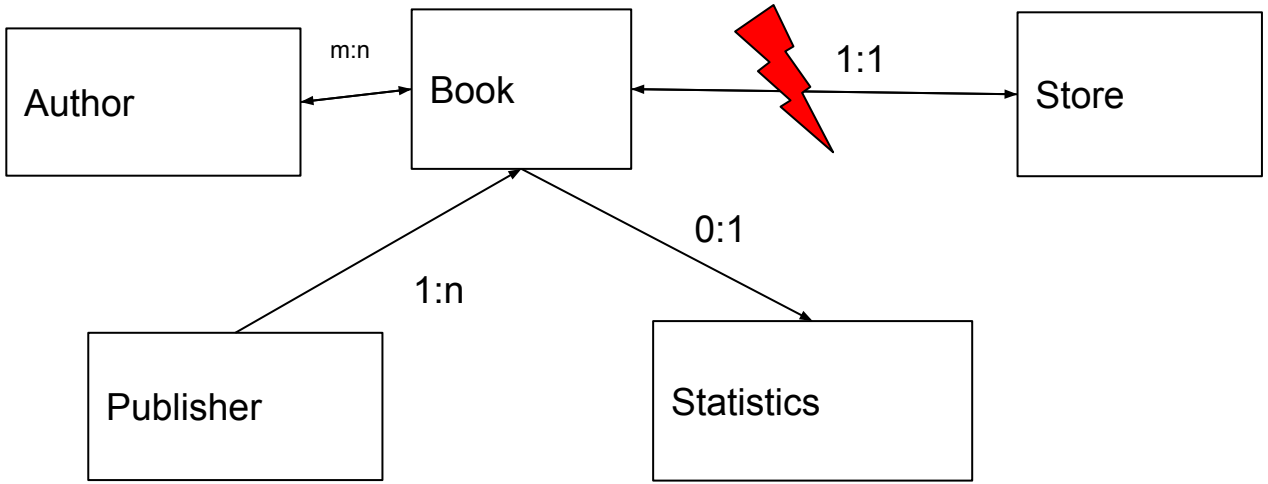
- Definition einer eigenen Schnittstelle, BooksRepository
  - extends CrudRepository genügt, aber nachdem wir wissen, dass wir es mit JPA machen werden nutzen wir JpaRepository
- Hinzufügen der JPA-Annotations in der Book-Klasse
- Umstellen des MapBooksServices auf den JpaBooksService
  - Im Wesentlichen ersetzen der Map-Aufrufe auf das BooksRepository
- Lösung:
  - https://github.com/Javacream/org.javacream.training.spring/commit/85007b93c34c8a173788875a7f66df7cc56c4f18

Klarstellung

- “Wir benutzen Spring Data”
  - Unvollständig: “Wir benutzen Spring Data JPA”
- Zum Arbeiten mit einem Spring Data Subproject sind Kenntnisse der zugrund liegenden Repository-Technologie verpflichtend
- Das eigentliche Spring Data stellt eine Reihe von Programmiermodellen zur Verfügung
  - Direkt Verwendung der zugrunde liegenden Technologie
    - also z.B. den EntityManager
  - Annotations-basierte Queries
  - Methoden-Namen werden als Selektionsausdrücke interpretiert
- Transaction Management bleibt exakt so wie ursprünglich

Spring Repositories und Aggregate

- Umsetzung eines EntityModells



Prinzipiell genügt hier ein PublisherRespository -> Aggregat-Entity

in der Praxis werden hierfür sicherlich mehrere Repositories sinnvoll sein

Technische Umsetzung mehrerer Repositories

integrata cegos

- Beispiel: JPA-Repository
  - PublishersRepository
  - BooksRepository
  - Frage: Wie viele EntityManager-Instanzen werden pro Aufruf benutzt?
    - Abhängig von der Transaktionssteuerung
      - Pro Transaktions-Kontext wird ein einziger EntityManager benutzt
- Unterschiedliche Repository-Technologien benutzen natürlich komplett unterschiedliche “EntityManager”
  - Book-Objekt geladen über ein JpaRespository ist natürlich nie identisch (via Referenz) mit einem Book-Objekt aus einer z.B. Couchbase
  - Transaktionalität ist abhängig von den kéingesetzten Datenbanken

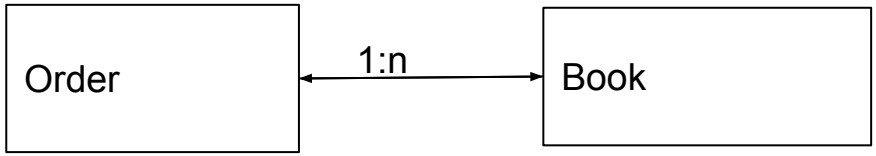
Erweiterung unserer Anwendung

integrata cegos

OrderService
order (String isbn, int number): Order

Order
orderId
isbn
number
totalPrice
status (OK, PENDING, UNAVAILABLE)

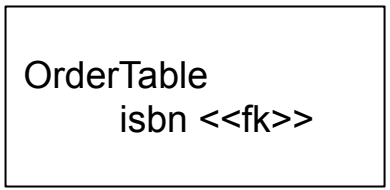
Technisches Modell

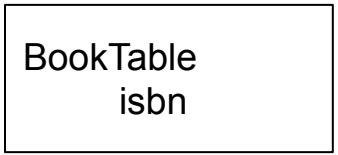


VORSICHT: Weder notwendig noch in der Praxis immer sinnvoll

Die Tabellen müssen zwangsläufig in einer gemeinsamen Datenbank liegen

Widerspruch zu einer Microservice-basierten Architektur





ToDo

integrata cegos

- Machen Sie sich klar, dass eine Relation zwischen OrderTable und BookTable nicht unbedingt notwendig ist
- In unserem Beispiel werden wir diese Relation einführen
  - OrderRepository mit OrderEntity in Relation zur BookEntity
  - Diskussion: Wie wäre die Alternative
- Nicht vergessen: Zum Zugriff ist ein kleiner WebService nötig
  - Neue Klasse “Order” (Vorsicht: Order ist ein SQL-Schlüsselwort, @Entity(“OrderEntity”)
  - OrderRepository-Interface
  - Anpassung der Klasse Book: private Set<Order> orders
  - OrderWebService
    - order Anlegen
    - alle Orders anzeigen

Custom Queries

Namenskonventionen (Erfindung von Ruby on Rails)

integrata cegos

- findById als Bestandteil des JpaRepositories
  - select * from BOOK where id = :id
- findByTitle
  - select * from BOOK where title = :title
- findByTitleAndPrice
  - …
- findByTitleAndPriceOrderByIsbnAscending
- findPriceByTitle()
  - select price from …
- OrderRepository
  - findBookTitleByOrderId

@Query

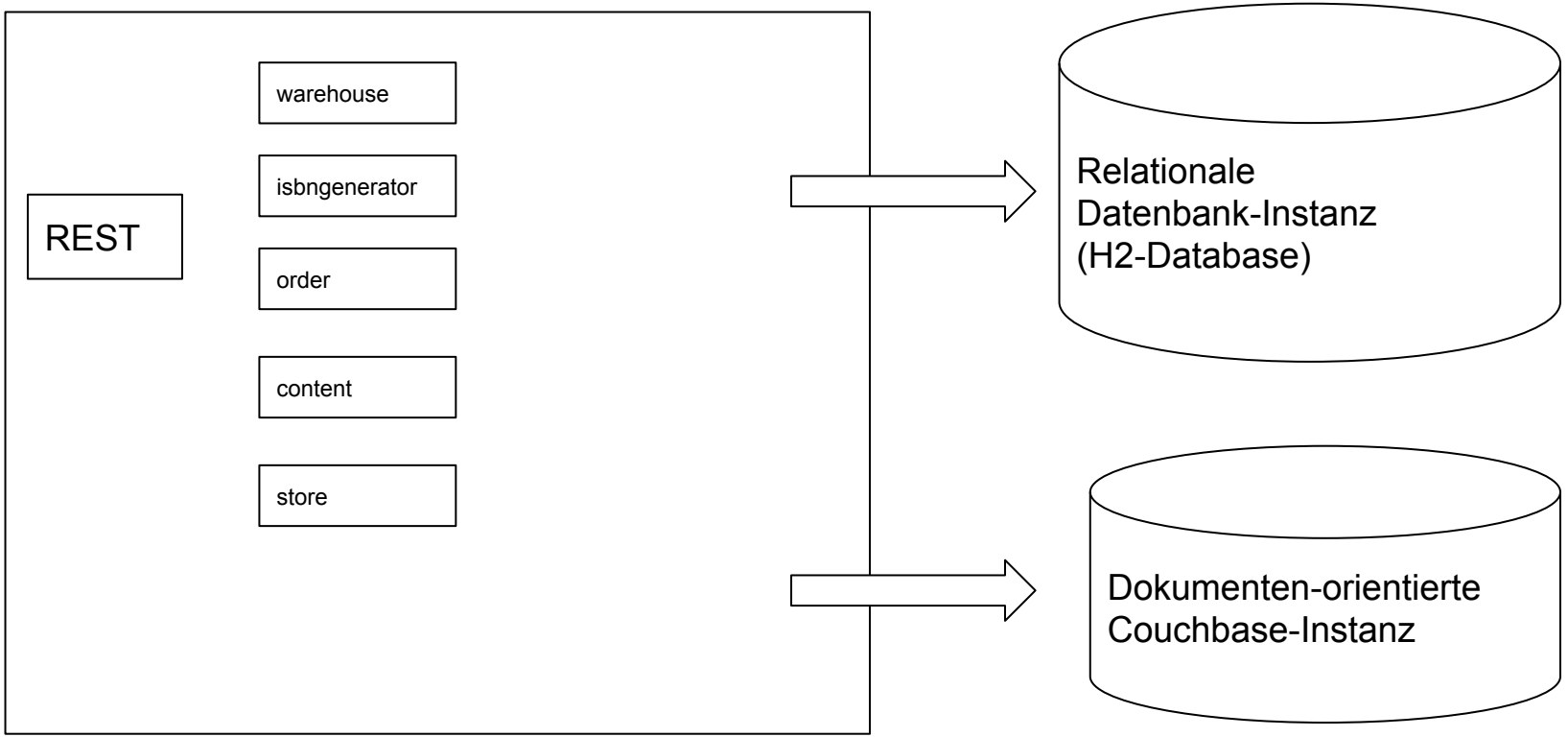
- Damit werden custom Repository-Methoden annotiert
- Mit @Query können auch Daten-Ändernde Abfrage formuliert werden
  - Vorsicht: Die Query-Methoden sind auf Abfragen optimiert
    - z.B. @Transactional ist auf “read-only” gesetzt
  - @Modifying

ToDo

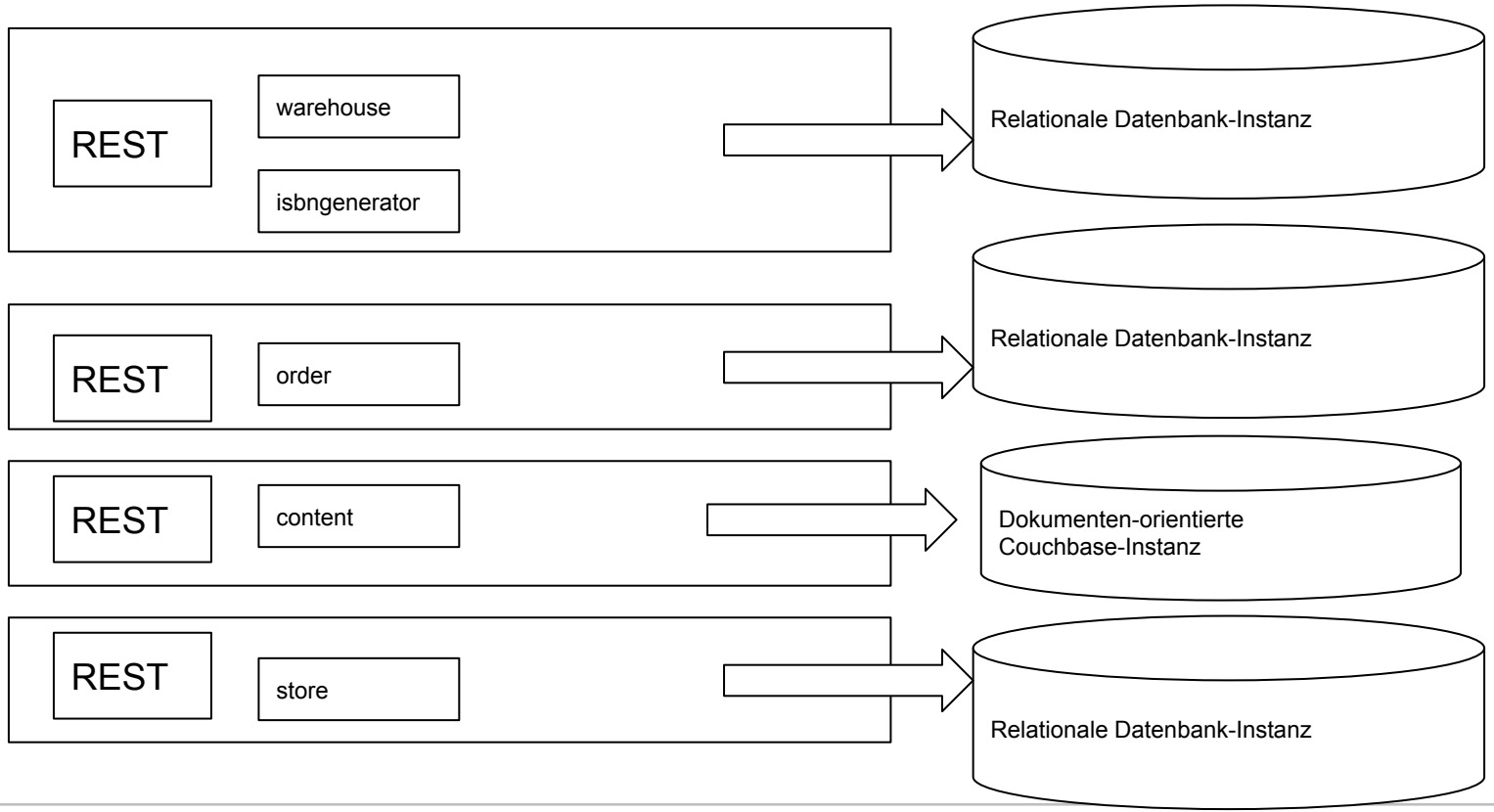
integrata cegos

- Erweitern Sie die Repository-Schnittstellen um weitere Operationen
  - Book: findByTitle
  - https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods
- @Query-Annotation
  - Eine simple Abfrage
  - @Modifying, z.B. ein deleteByTitle
- Schreiben Sie ein Custom Repository
  - https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.custom-implementations
- Example-API
  - https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#query-by-example

Stand unserer Applikation



Eine refactored Microservices-Applikation



Mehrere relationale Datenbanken in einer App

- Das kann die Spring Boot Autoconfiguration nicht
- Eigene Konfiguration pro Datenbank notwendig
  - Zuordnung der Entity-Klassen sinnvollerweise über Paketstruktur
  - https://www.baeldung.com/spring-data-jpa-multiple-databases