

## **Web Engineering**

# **Persistenz (Teil 1)**

Adrian Herzog

*(basierend auf der Arbeit von Michael Faes, Michael Heinrichs & Prof. Dierk König)*

# Organisatorisches

## *Nächstes Mal: Kickoff für die bewertete Übung («Projekt»)*

Ihr erhaltet nächstes Mal genauere Infos zum Projekt und habt dann eine Woche Zeit, mir eure Projektidee zu senden. Das Projekt ist eine Einzelarbeit.

Du kannst jetzt bereits Ideen für das Projekt sammeln. Hier eine Liste als Inspiration: <https://flaviocopes.com/sample-app-ideas/>

# Persistenz

Wie werden Daten einer Web-App gespeichert?

Bisher:

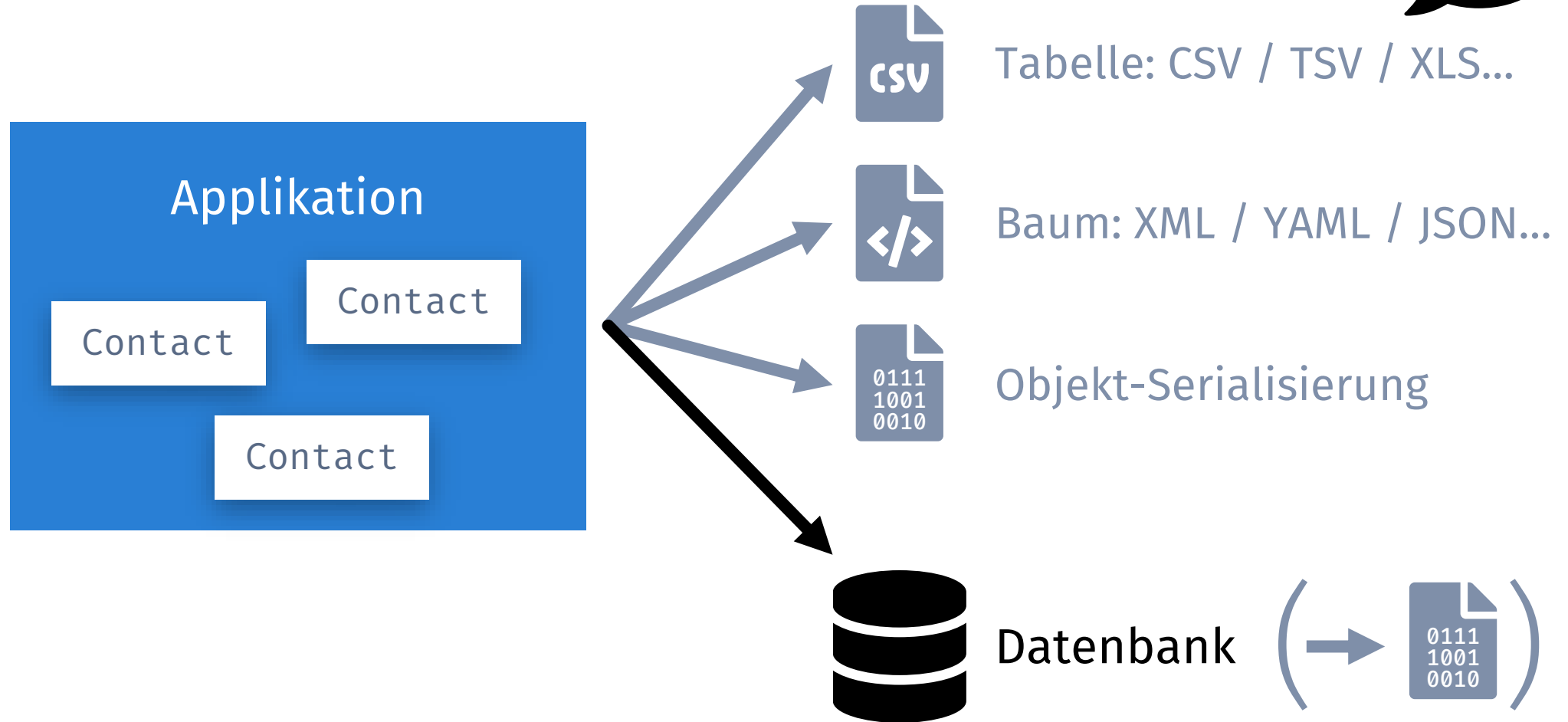
```
public class ContactService {  
    private List<Contact> contacts;  
  
    public void add(String first, String last, ...) {  
        var contact = new Contact();  
        ...  
        contacts.add(contact);  
    }  
}
```

**Problem:** Java-Objekte sind nur im RAM abgelegt. Wenn App neugestartet wird, sind sie weg!

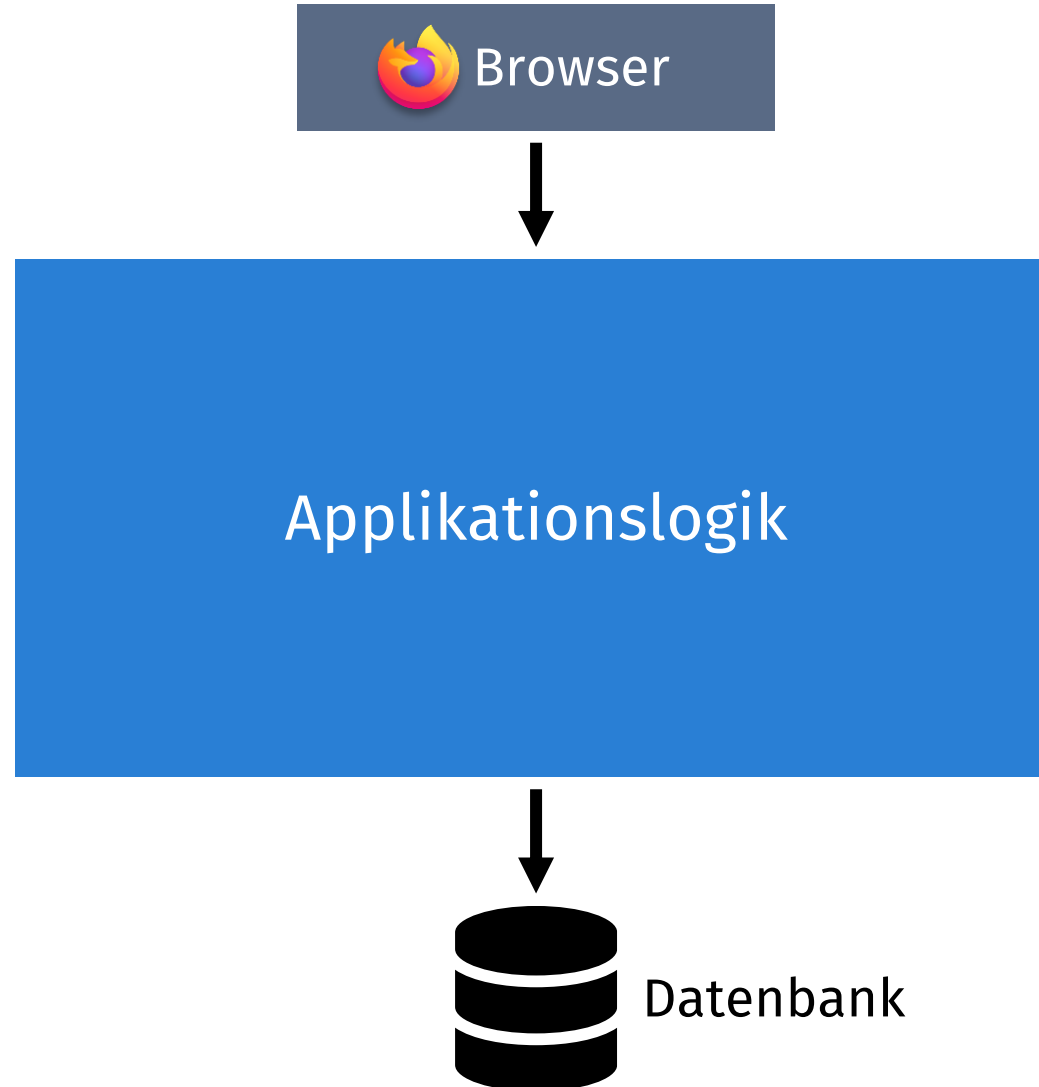
**Persistenz:** Speichern von Applikationsdaten auf *nichtflüchtigem* Speichermedium, z. B. Festplatte.

# Möglichkeiten für Persistenz

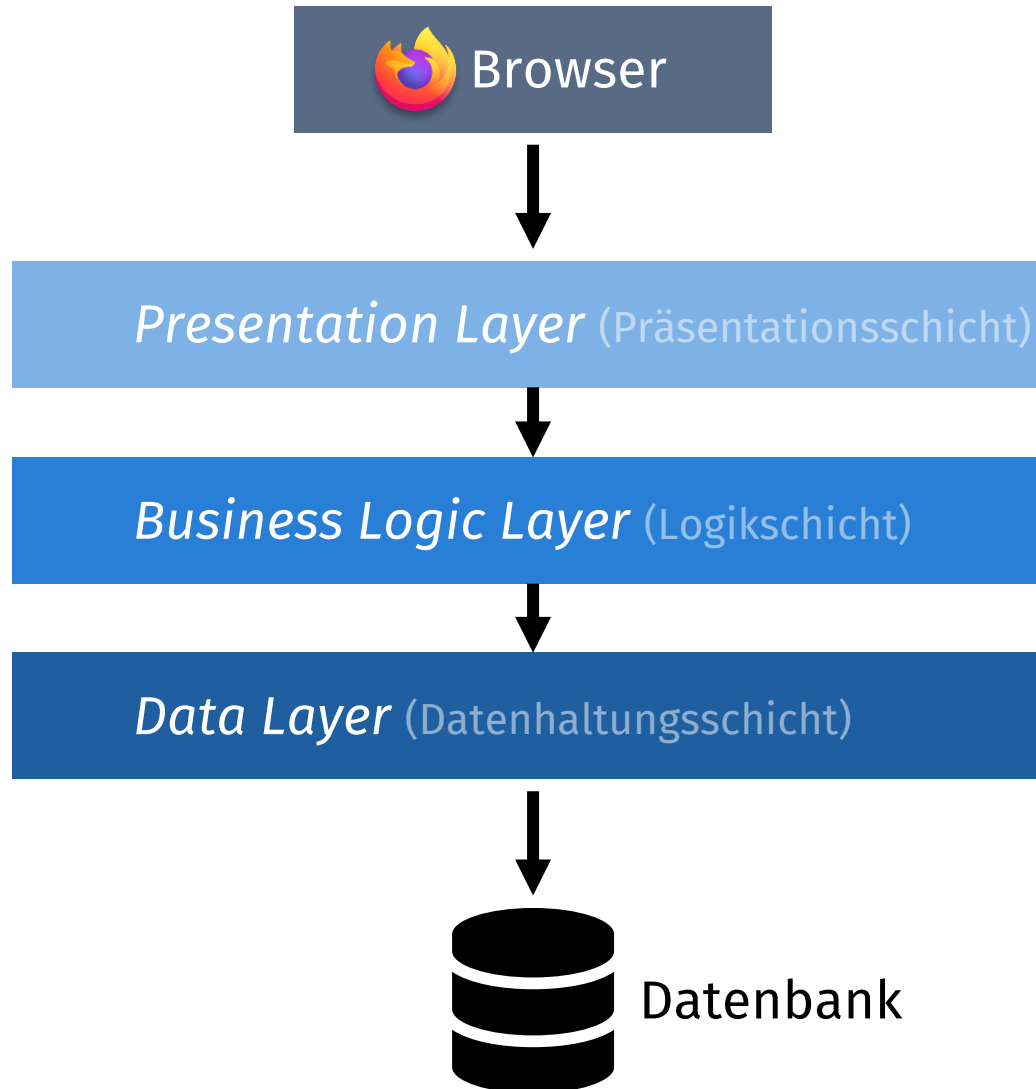
OOP2



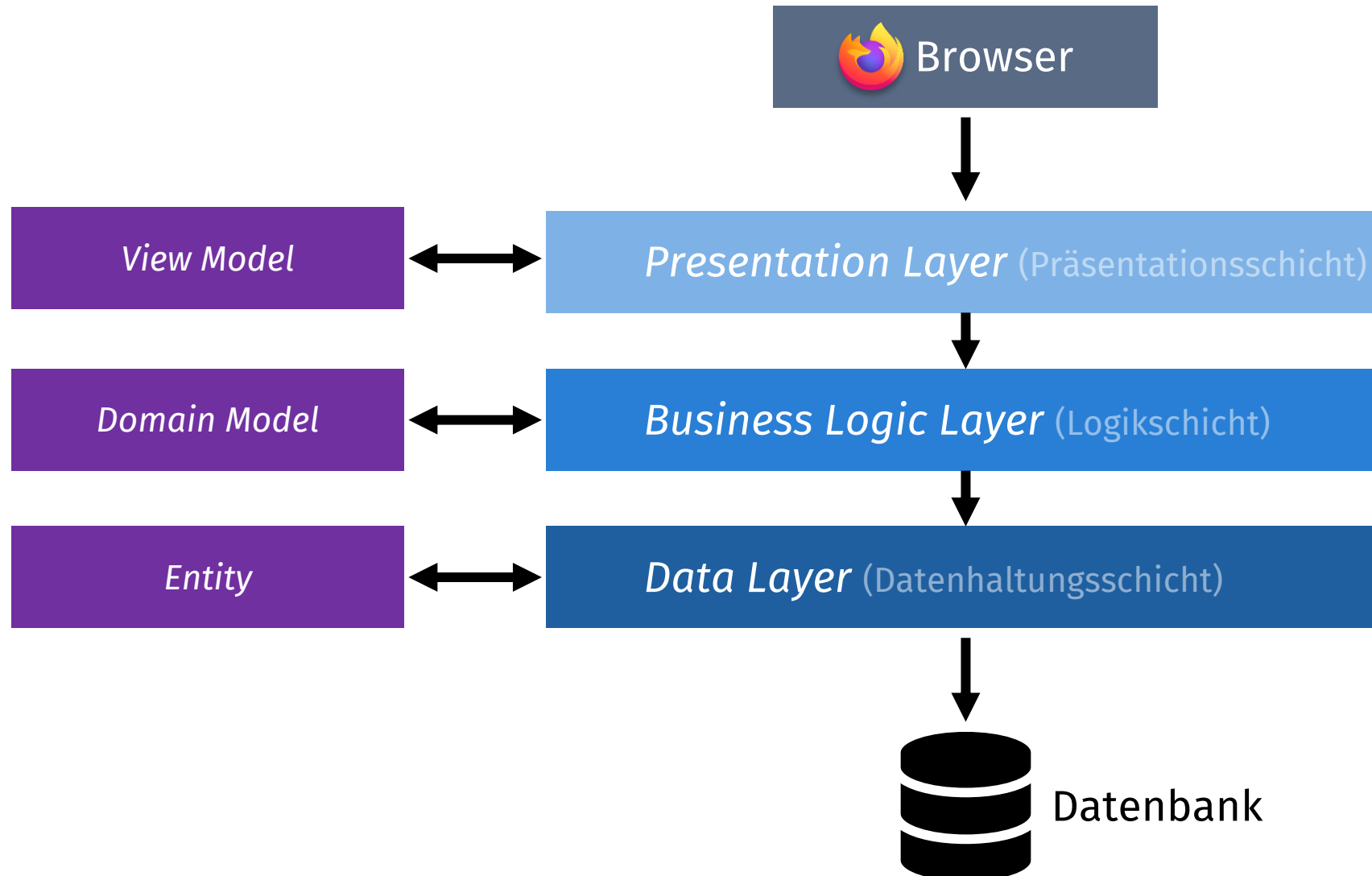
# Architektur einer Web-App: vereinfacht



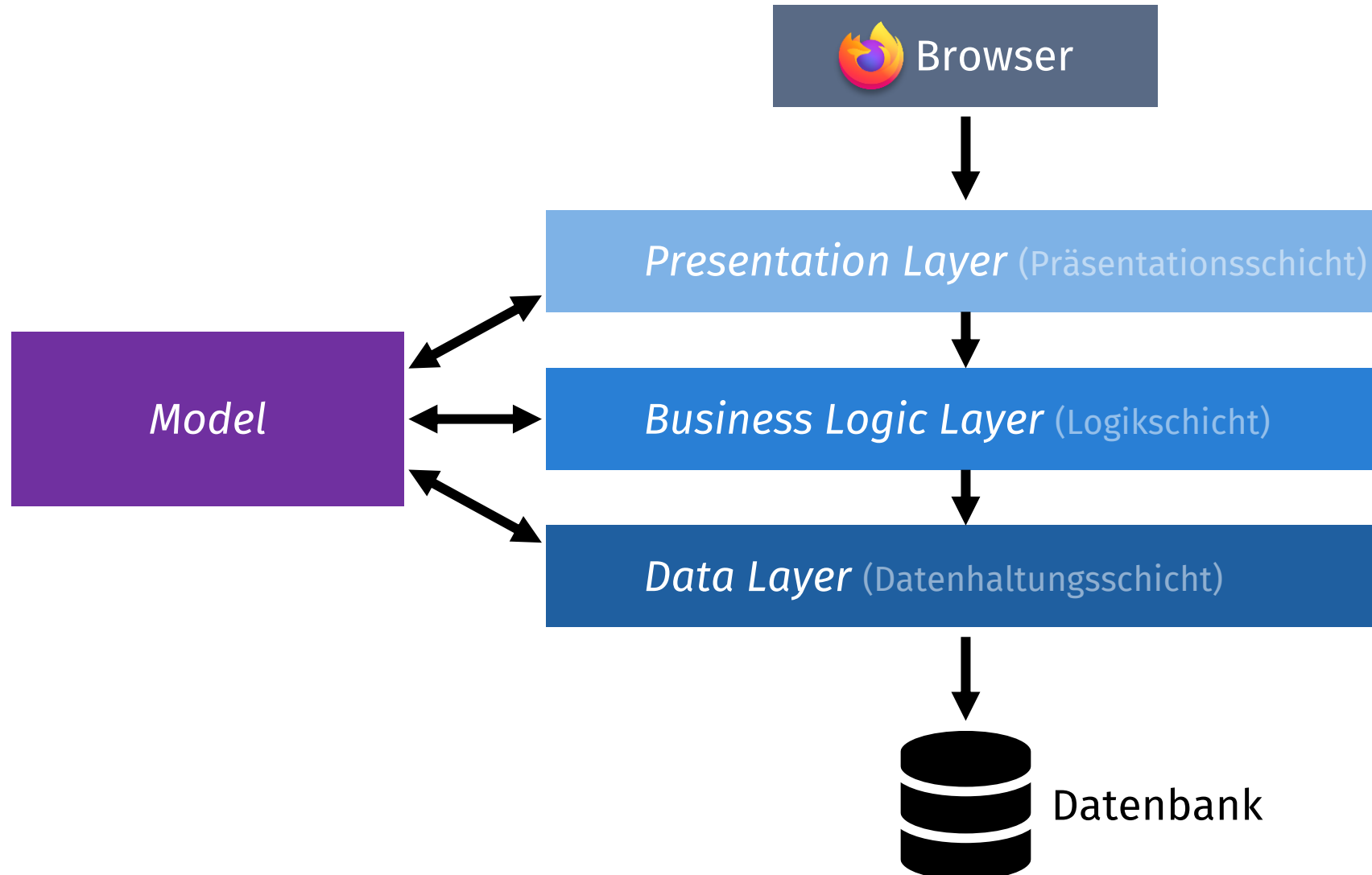
# Schichtenarchitektur (*layered*)



# Separate Models



# Ein einziges Model





# Einziges Model vs. separate Models

## Einziges Model

- Einfach, wenig Duplizierung
- Weniger flexibel
- Details von gewählten Libraries/Frameworks schleichen sich ein
- Kann problematisch werden wenn App wächst



Wir machen es  
im Kurs so

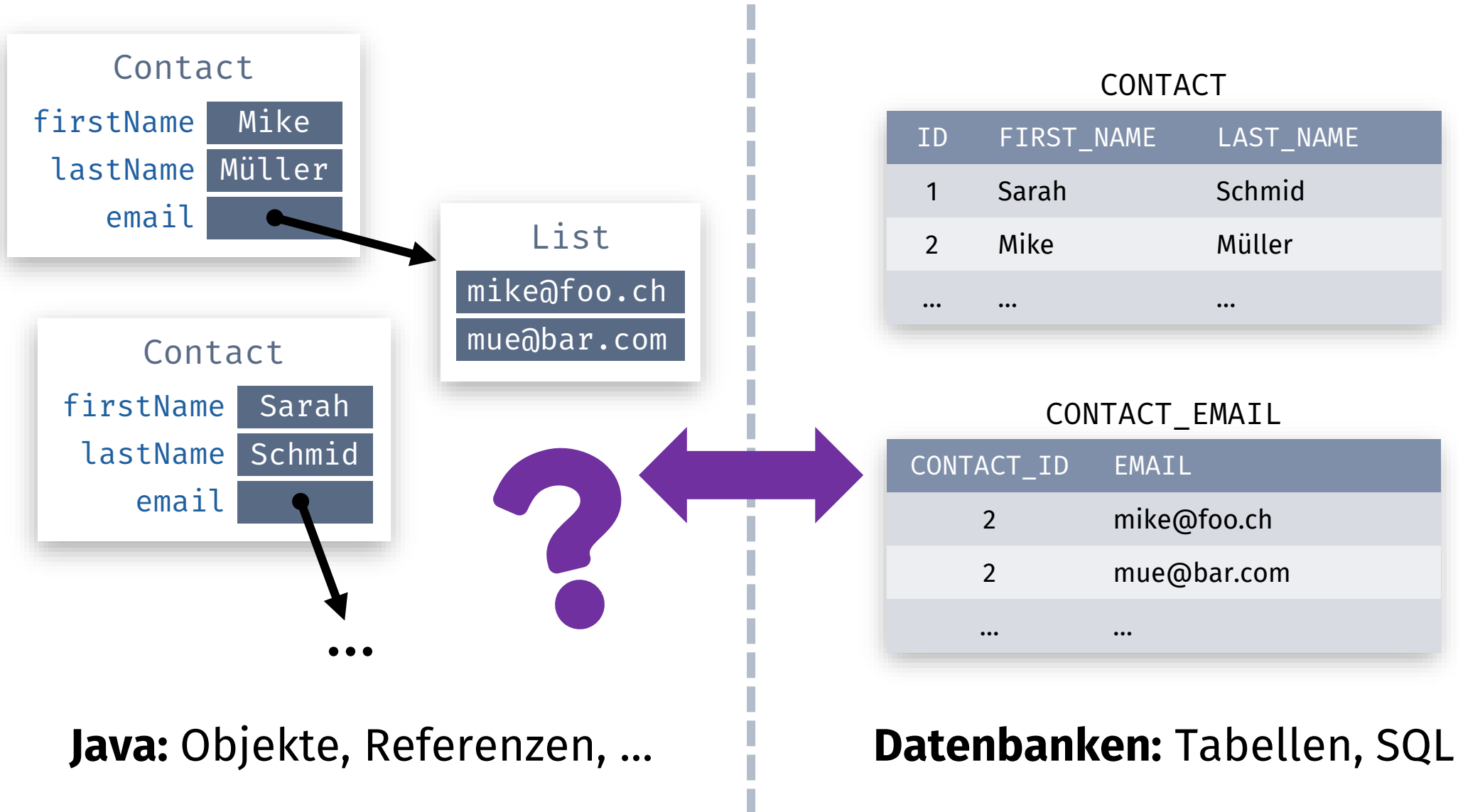
## Separate Models

- Viel Duplizierung und Boilerplate-Code zum Umwandeln zwischen Models
- Entsprechend aufwändig zu pflegen/ändern
- Klare Trennung von Zuständigkeiten
- Keine Abhängigkeit von Data- oder Presentation-Layer-Technologien

# ***Object–relational Mapping (ORM)***

«Objektrelationale Abbildung»

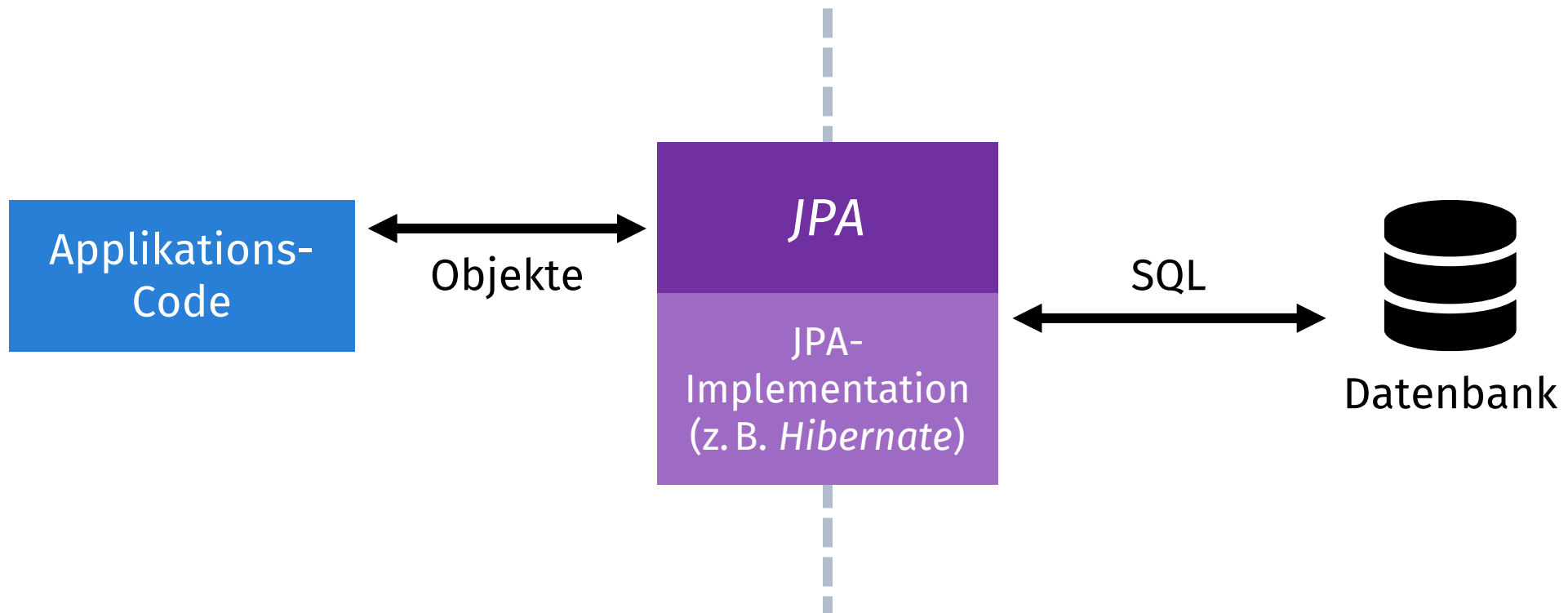
# OOP und Datenbanken



# *Object-relational Mapping*

Für viele Sprachen gibt es «ORM»-Tools, um Umwandlung zwischen Objekten und Datenbanken (SQL) zu automatisieren.

**In Java:** durch *Jakarta Persistence API* (JPA) standardisiert.



# JPA-Beispiel: Einfache Entity

generiertes  
SQL abhängig  
von DB

```
import javax.persistence.*;

@Entity
public class Contact {

    @Id
    @GeneratedValue
    private int id;

    private String firstName;
    private String lastName;

    public int getId() {
        return id;
    }

    ...
}
```

ansonsten normale  
Java-Klasse

```
create sequence HIBERNATE_SEQUENCE
start with 1
increment by 1;

create table CONTACT (
    ID integer not null,
    FIRST_NAME varchar(255),
    LAST_NAME varchar(255),
    primary key (ID)
);
```

!

CONTACT

ID	FIRST_NAME	LAST_NAME
1	Sarah	Schmid
2	Mike	Müller
...	...	...

# Element Collection

```
@Entity
public class Contact {

    @ElementCollection
    private List<String> emails;

    ...
}
```

keine Entity

CONTACT

ID	NAME
1	Sarah
2	Mike

CONTACT\_EMAILS

CONTACT_ID	EMAIL
1	sarah.schmid@fhnw.ch
2	mike@foo.ch
2	mue@bar.com

# Ausblick: Assoziationen

```
@Entity
public class Contact {

    @OneToMany
    pr... List<Address> addresses;

    ...
}
```

!

```
@Entity
public class Address {

    ...
}
```

CONTACT

ID	NAME
1	Sarah
2	Mike

«Join Table»

CONTACT\_ADDRESSES

CONTACT_ID	ADDRESS_ID
1	13
2	17
2	18

ADDRESS

ID	STREET_NO	CITY
13	Hauptstrasse 0	Gümlingen
17	Elchweg 33	Oberdorf
18	Mattenweg 4	Unterdorf

# Spring Boot & JPA

Spring Boot konfiguriert JPA-Implementation automatisch durch folgende Dependency:

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```



Zusätzlich brauchen wir Datenbank-*Driver*. Zum Start: In-Memory-DB.

```
<dependency>
  <groupId>com.h2database</groupId>
  <artifactId>h2</artifactId>
  <scope>runtime</scope>
</dependency>
```



# Übung 1: Contact als Entity-Klasse

- a) Füge dem Projekt «contactlist-persistence» die Dependencies für Spring Data JPA und für die H2-Datenbank hinzu.
- b) Mache aus der `Contact`-Klasse eine JPA-Entity, indem du die benötigten Annotationen hinzufügst. Entferne `setId()`; die IDs sollen automatisch von der Datenbank generiert werden.
- c) Starte die Web-App und beobachte folgende Ausgabe auf der Konsole:

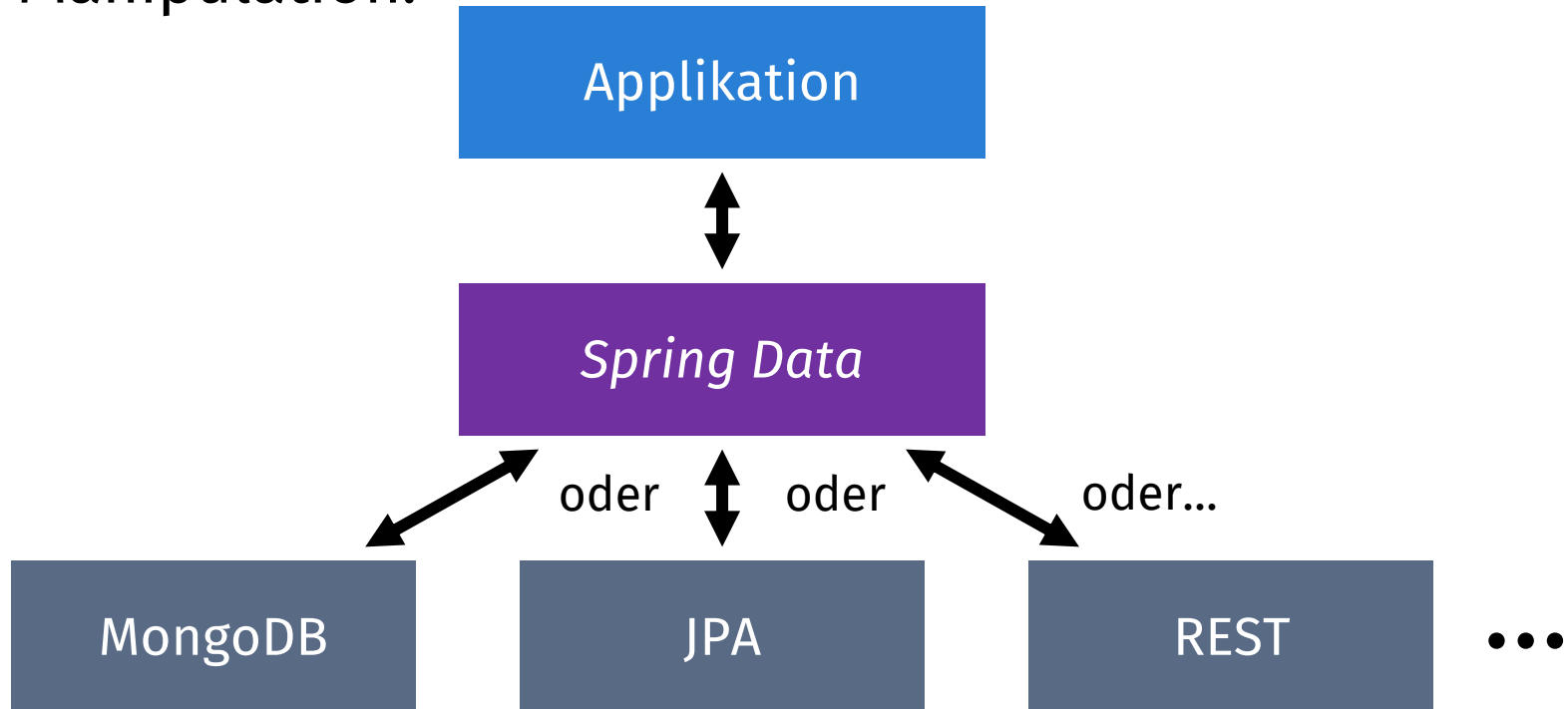
```
... H2 console available at '/h2-console'. DB available at 'jdbc:h2:mem:3297da63-9949-4c70-9c4c-1c80c52da336'
```

Rufe die URL [localhost:8080/h2-console](http://localhost:8080/h2-console) auf, logge dich bei der in der Konsole angezeigten Datenbank ein (Passwort: leer) und inspiziere die generierten Tabellen und Spalten.

# Spring Data

# Spring Data

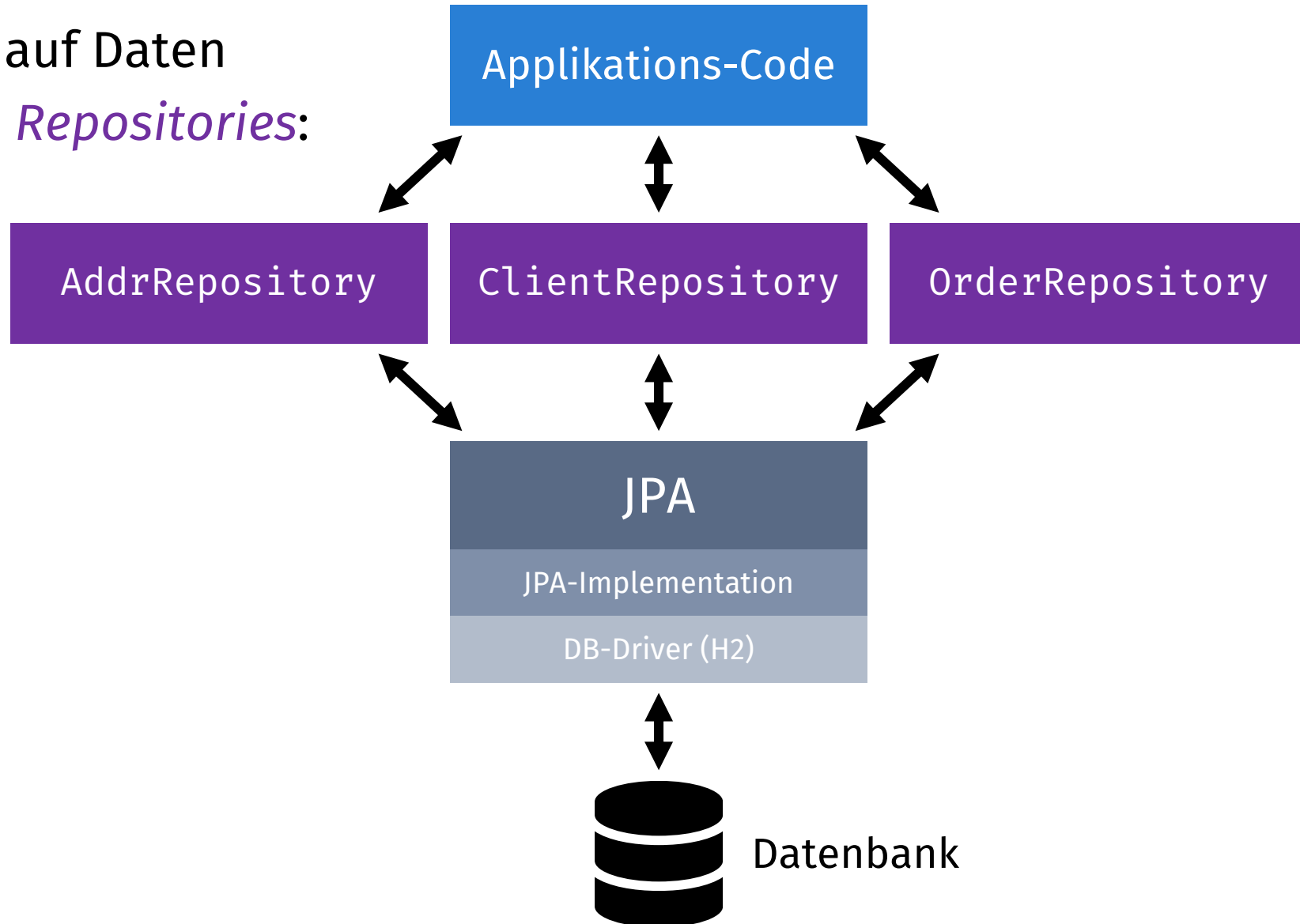
*Spring Data* bietet konsistentes Programmiermodell für Datenzugriff und Daten-Manipulation:



Eigenheiten der verschiedenen Technologien werden so gut wie möglich weg-abstrahiert. Verwenden aber trotzdem JPA-Annotationen.

# Spring Data Repositories

Zugriff auf Daten  
mittels *Repositories*:



# Ein Repository

Ein Repository ist ein Interface, das `JpaRepository` erweitert.  
Spring Data implementiert dieses automatisch:

```
public interface ContactRepository  
    extends JpaRepository<Contact, Integer> {  
}
```

Entity-Typ

@Id-Typ




Methoden	Beschreibung
<code>findById(id), findAll()</code>	Lädt benötigte Daten aus der DB und erstellt Entity-Objekte daraus
<code>delete(entity), deleteById(id), deleteAll()</code>	Löscht die Einträge, die den gegebenen Entities entsprechen, aus der DB
<code>save(entity), saveAll(entities)</code>	Speichert neue Einträge oder modifiziert die entsprechenden Einträge in der DB

# Repositories erweitern

Und damit noch nicht genug. Können weitere Methoden deklarieren, welche ebenfalls automagisch implementiert werden:

```
public interface ContactRepository
    extends JpaRepository<Contact, Integer> {
    List<Contact> findByLastName(String lastName);
    List<Contact> findByFirstNameAndAgeLessThan(String firstName, int age);
}
```



Spring Data erstellt SQL-Queries basierend auf Methodennamen.

Details in der Dokumentation:

<https://docs.spring.io/spring-data/jpa/reference/jpa/query-methods.html>

# Übung 2: Ein Repository für Contact

- a) Erstelle ein Spring Data Repository für `Contact`. Du brauchst keine Query-Methoden hinzuzufügen.
- b) Ändere den `ContactService` so ab, dass statt der `ArrayList` das neue Repository zum Speichern der Kontakte verwendet wird. Du bekommst eine Instanz des Repositories durch Dependency-Injection. Verwende vorerst folgende Repository-Methoden: `findAll()`, `findById(id)`, `save(entity)`.
- c) Starte die App und überprüfe, dass die Kontakte weiterhin alle angezeigt werden. Öffne nochmals die H2-Console und führe die Abfrage `SELECT * FROM CONTACT` aus. Die Tabellen sollten jetzt gefüllt worden sein.

# Fragen?

