Objektorientierte Datenbanken

Vorlesung 10

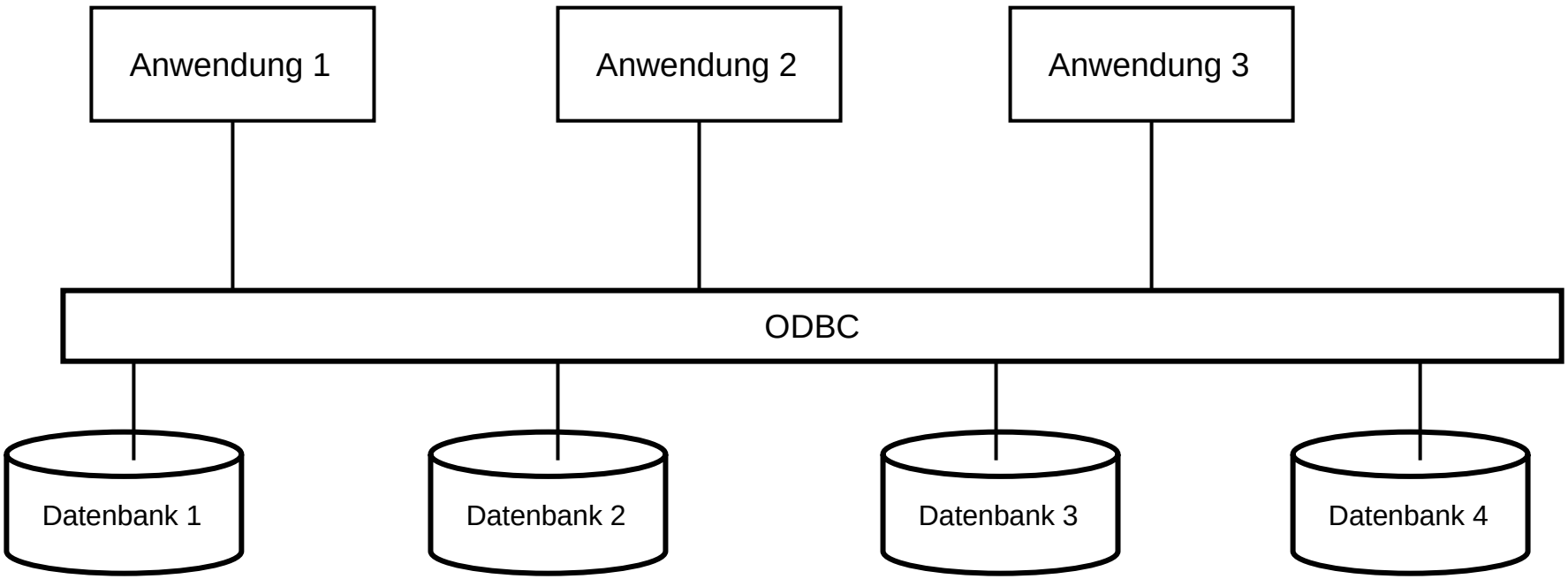
Sebastian Iwanowski
FH Wedel

JDBC: Java Database Connectivity

ODBC: Open Database Connectivity

Ziel

- **Standardisierter Zugriff auf beliebige Datenquellen mit SQL-Befehlen**

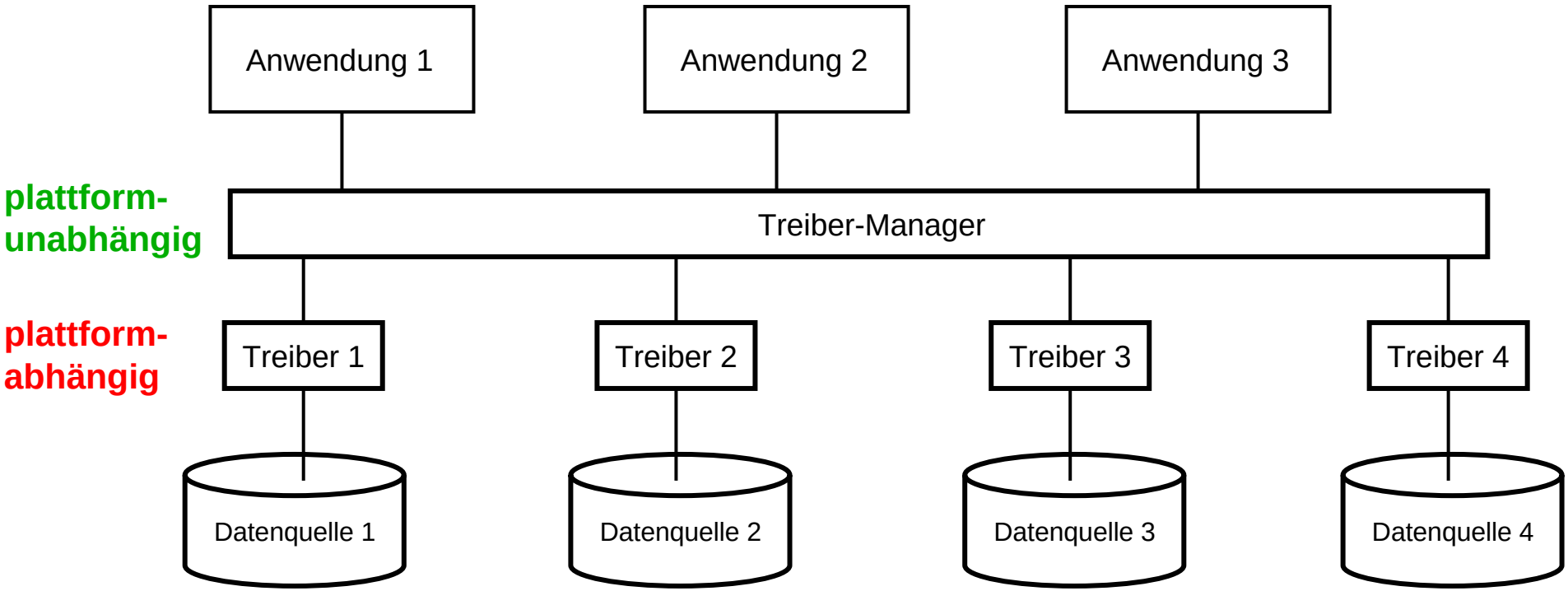


Geschichte

- **entwickelt Anfang der 90‘er Jahre mit Microsoft-Unterstützung**
- **Anwendungsfokus: C- / C++- Anwendungen**

ODBC: Open Database Connectivity

Architektur

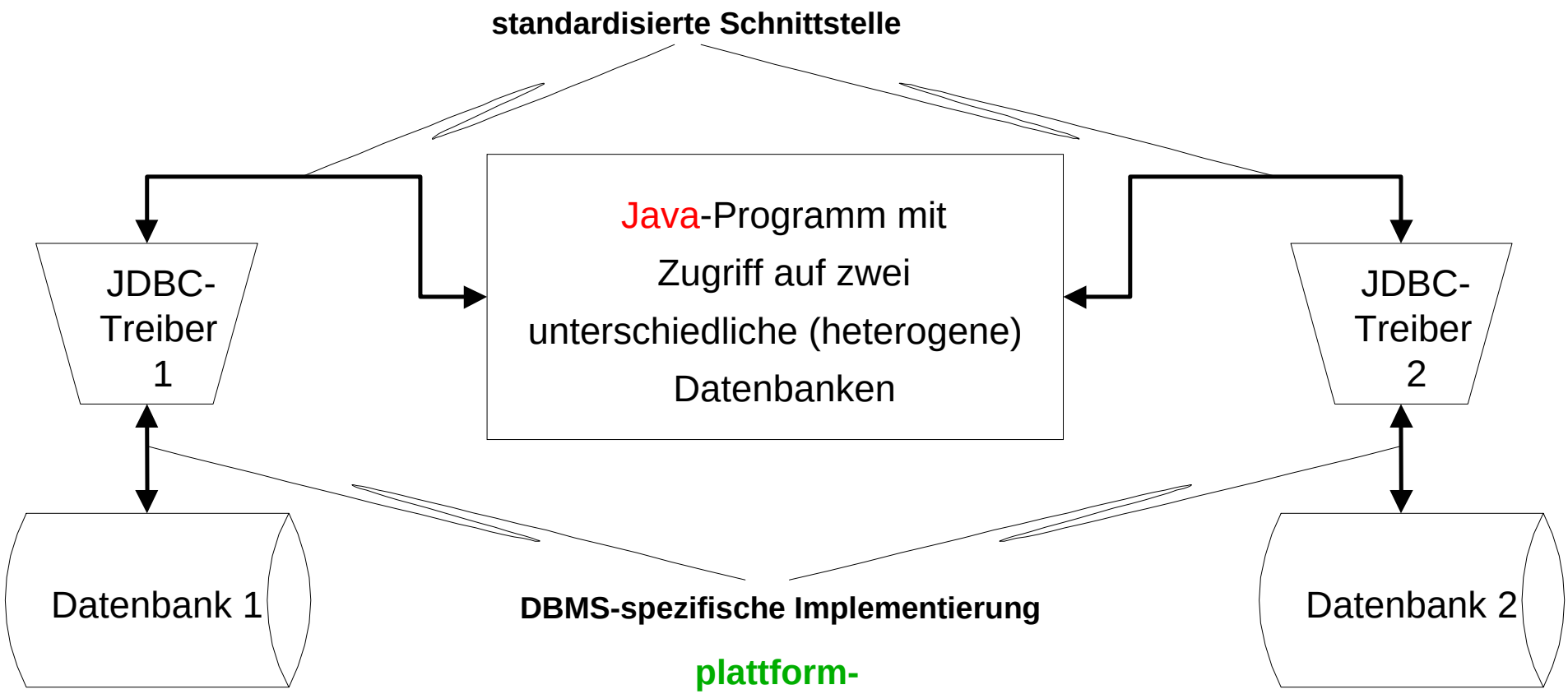


Mögliche Datenquellen:

- Desktop-Datenbanken (für Einzelbenutzer)
- Server-Datenbanken (für parallelen Mehrbenutzerbetrieb)
- Textdateien in beliebigem Format
- Tabellenkalkulationen

Datenquellen müssen keine richtigen Datenbanken sein

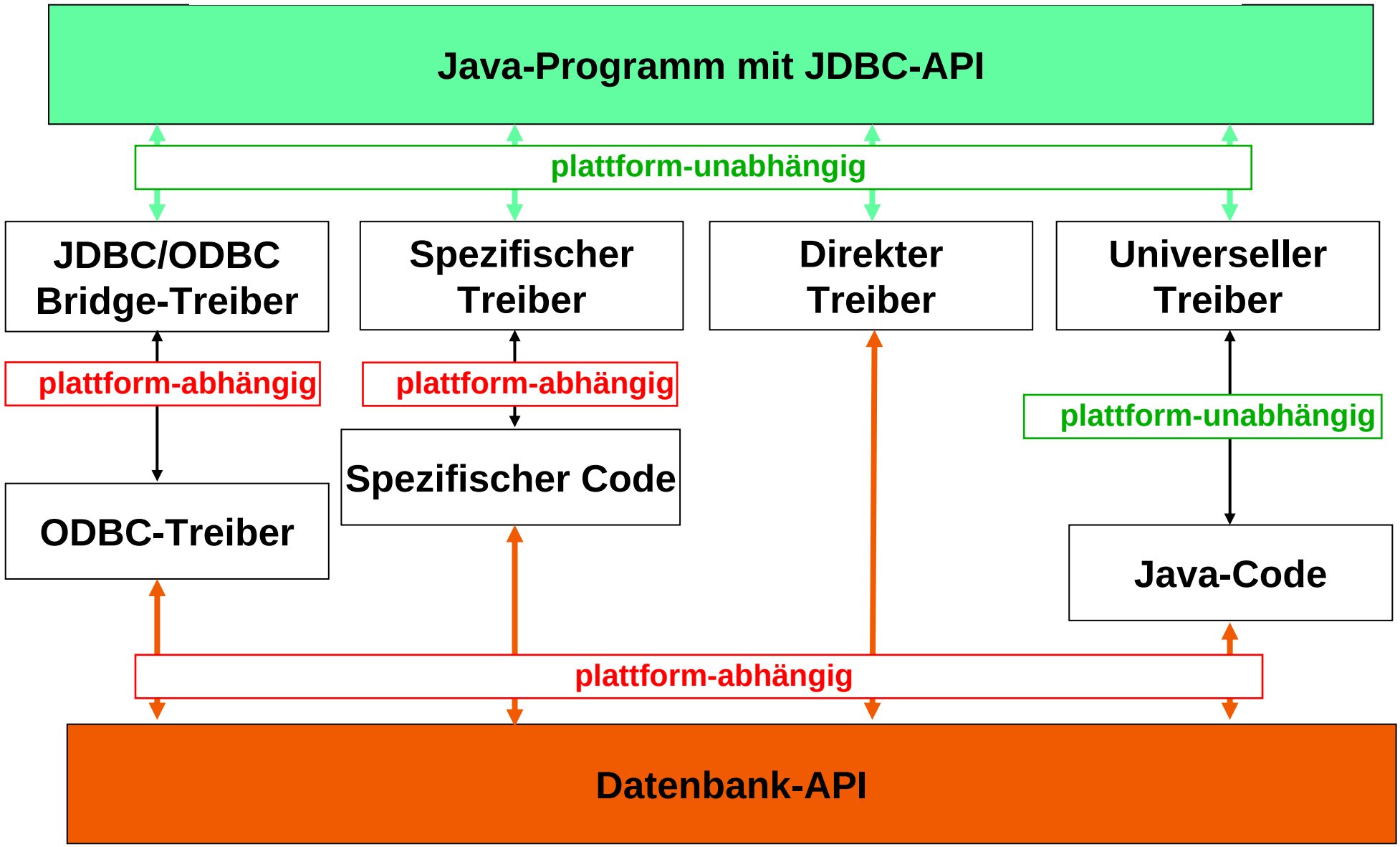
JDBC: Java Database Connectivity



plattform-unabhängig

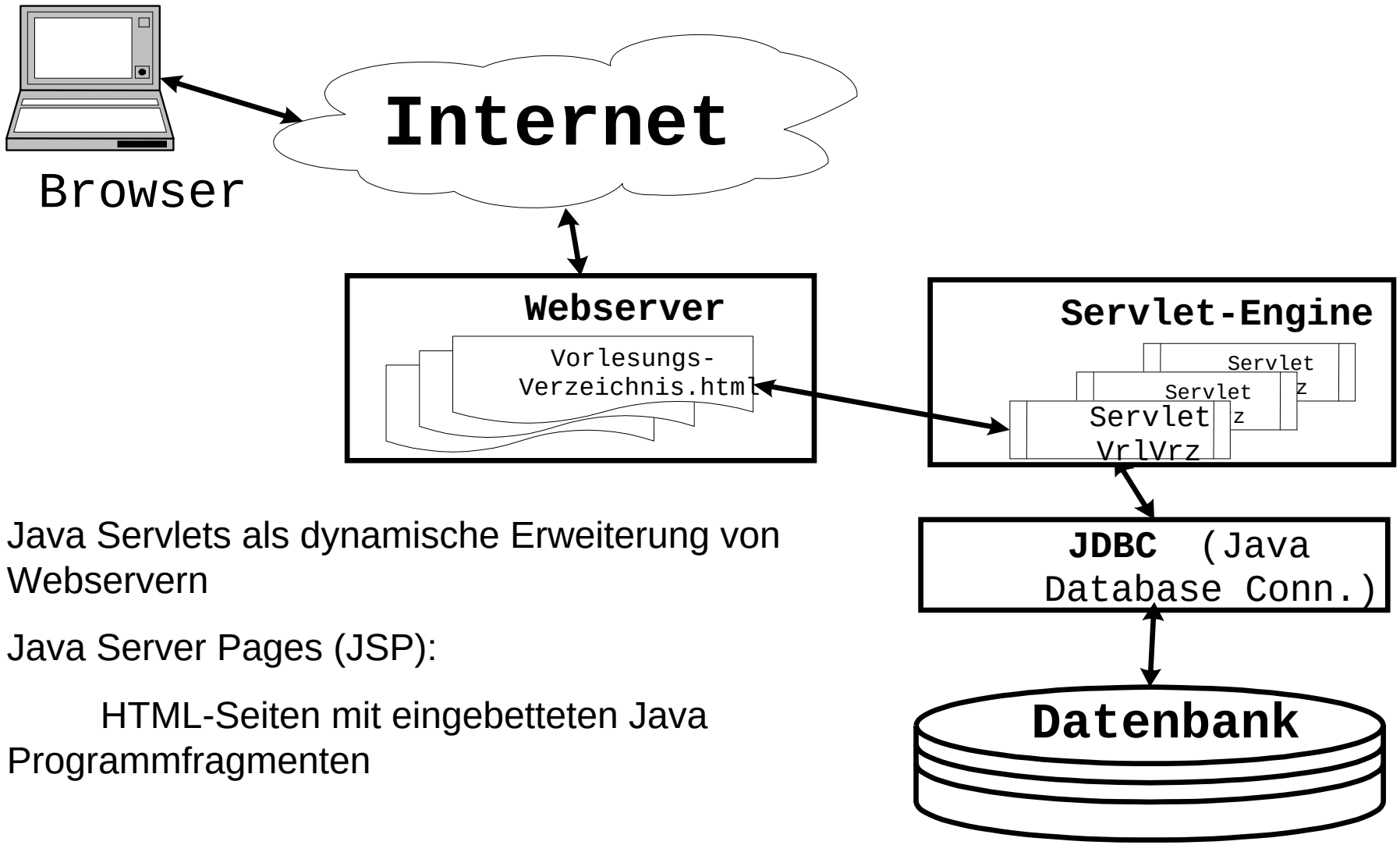
(im Prinzip …)

Anbindungsmöglichkeiten für JDBC



Aktuelle Verwendung von JDBC:

Anbindung von Datenbanken an das Internet



- Java Servlets als dynamische Erweiterung von Webservern
- Java Server Pages (JSP):

  HTML-Seiten mit eingebetteten Java Programmfragmenten

Java-API für JDBC

- **Laden des JDBC-Treibers**

  dynamisch:

  ```
  static Class Class.forName (String drivername)
  ```

  statisch: durch Spezifikation im properties-File

- **Verbindungsaufbau zur Datenbank**

  ```
  static Connection DriverManager.getConnection (String url)
  ```

- **Generierung eines Anfrageobjekts**

  ```
  Statement Connection.createStatement ()
  ```

  Hierdurch werden Eigenschaften der Antwort festgelegt.

- **Formulierung und Stellen der Frage**

  ```
  ResultSet Statement.executeQuery (String sqlQuery)
  ```

  generiert Datenbanktabelle (mit Zeilen und Spalten)

Java-API für JDBC

- **Navigieren in der Antwort:**

```
boolean ResultSet.next ()
boolean ResultSet.previous ()
```

navigiert zur nächsten bzw. vorigen Zeile der Antwort

```
String ResultSet.getString (int index)
int ResultSet.getInt (int index)
```

liest das Element in Spaltenposition index in der gegenwärtigen Zeile

```
String ResultSet.getString (String name)
int ResultSet.getInt (String name)
```

liest das Element in Spalte name in der gegenwärtigen Zeile

`ResultSet` bietet viele weitere nützliche Methoden.

Wichtig: Alle Zugriffe auf Datenbank mit `try ... catch` !

Die Interfaces `Connection`, `Statement`, `ResultSet` und die Klasse `DriverManager` befinden sich im package `java.sql`

Java-API für JDBC: Einfaches Beispiel

```
try {
     Class.forName("oracle.jdbc.driver.OracleDriver");
     conn = DriverManager.getConnection
               ("jdbc:oracle:oci8:@lsintern-db", "nobody", "Passwort");
     sql_stmt = conn.createStatement();
     }
catch (Exception e) {
     System.err.println("Folgender Fehler ist aufgetreten: " + e);
     System.exit(-1);  }
try {
     ResultSet rset = sql_stmt.executeQuery(
       "select Name, Raum from Professoren where Rang = 'C4'");
     System.out.println("C4-Professoren:");
     while(rset.next())  {
          System.out.println
               (rset.getString("Name") + " " +  rset.getInt("Raum"));
                              }
     rset.close();
     }
catch(SQLException e) {System.out.println ("Error: " + e); }
 try {
     sql_stmt.close(); conn.close();
     }
catch (SQLException e) {
     System.out.println("Fehler beim Schliessen der DB: " + e);
                              }
```

Professoren			
PersNr	Name	Rang	Raum
2125	Sokrates	C4	226
2126	Russel	C4	232
2127	Kopernikus	C3	310
2133	Popper	C3	52
2134	Augustinus	C3	309
2136	Curie	C4	36
2137	Kant	C4	7

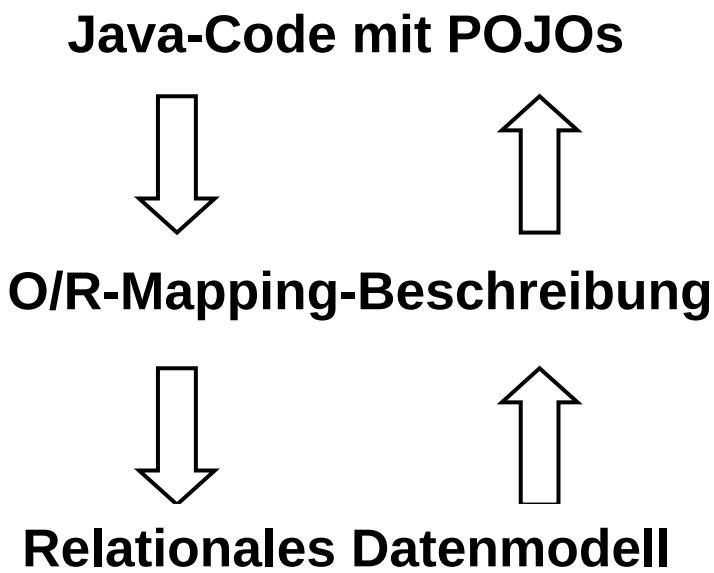
Hibernate: Automatisches O/R-Mapping für Java

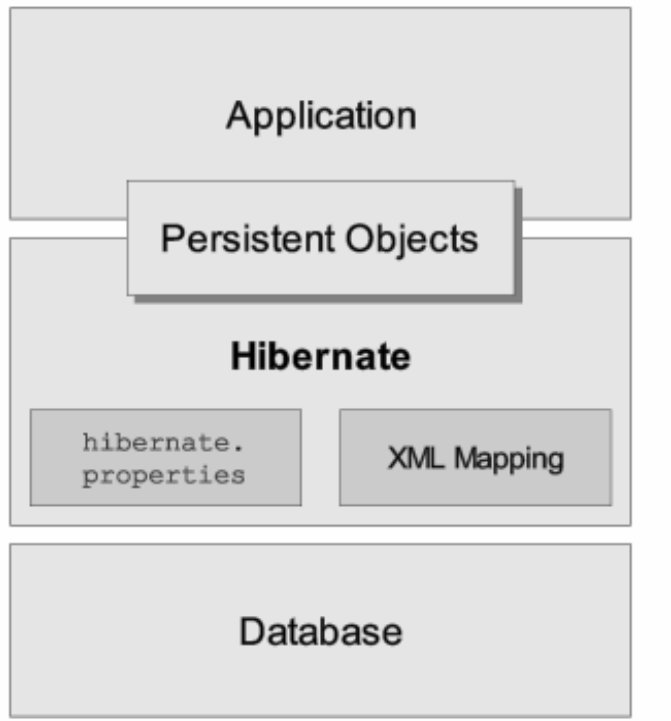
Historie von Hibernate

- **Ende 2001: initiiert von Gavin King**
- **weitergeführt als Open Source Projekt** www.hibernate.org
- **Ende 2003: aufgenommen von JBoss in J2EE-Entwicklung, vor allem als Alternative zum EJB2-Standard**
- **2004: 1. Auflage des Buchs „Hibernate in Action“**
- **2005: Aufnahme vieler Hibernate-Konzepte in EJB3-Standard angekündigt**

Prinzip von Hibernate

Ausgangssituation:





Gewünschte Funktionalität:

Automatische Überführung von einer Beschreibungsebene in die nächste

Prinzip von Hibernate

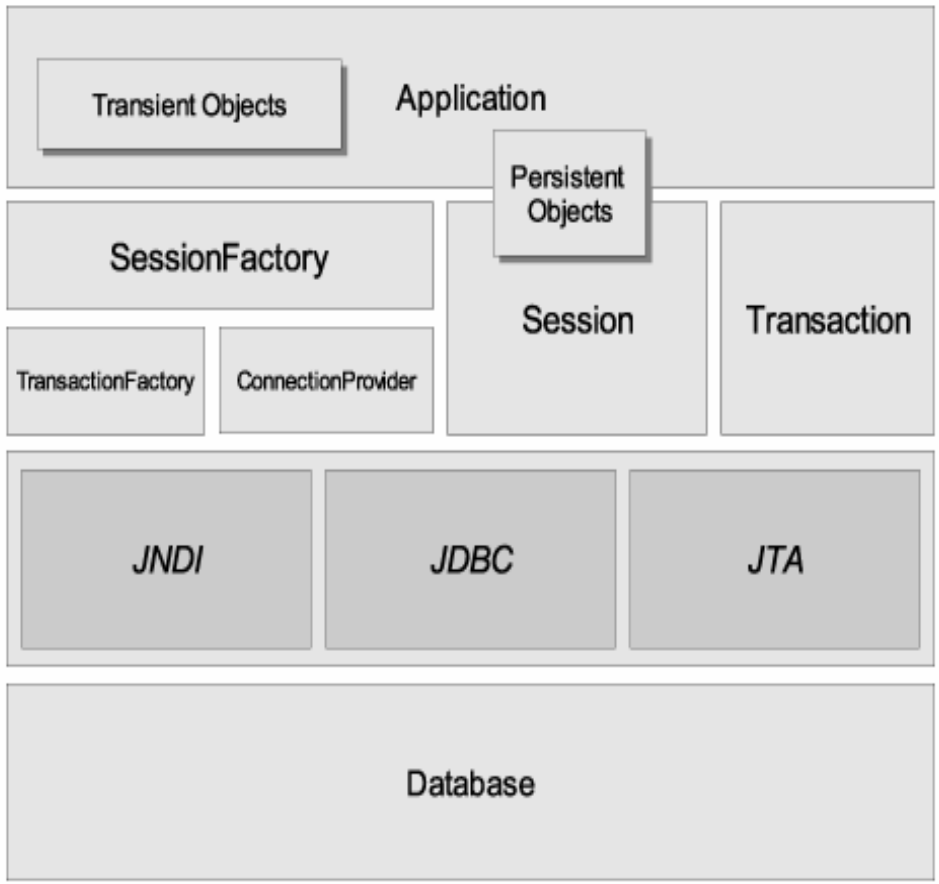
Ausgangssituation:

Java-Code mit POJOs

Hibernate-Schicht

Java-APIs

Relationales Datenmodell



Gewünschte Technologie:

Alle vorhandenen Java-APIs sollen genutzt werden

Wesentliche Eigenschaften von Hibernate

- **Transparente Persistenz**
- **Transitive Persistenz (Persistenz per Erreichbarkeit)**
- **Inheritence mapping strategies**
- **Automatic dirty object checking**
- **Intelligent fetching and caching**
- **Unterschiedliche Anfragekonzepte (Queries und Criteria)**

Transparente Persistenz

- **Jede Klasse darf persistent sein.**
  Es gibt keine Anforderungen an Superklassen, von denen die persistente Klasse erben muss oder an Interfaces, welche sie implementieren muss.

- **Persistente Klassen dürfen auch in anderen Umgebungen benutzt werden, z.B. in Testumgebungen.**

- **Persistente Objekte haben keine für den Anwender sichtbare Zustände.**
  Sie benötigen in der Anwendungsschicht keine besonderen Behandlungen.
  Alle persistenzspezifischen Aspekte werden in den Interfaces **`Session`** oder **`Query`** verwaltet.

Fazit:

Der Anwendungsprogrammierer kann Hibernate als Black Box betrachten:
Der konkrete Persistenzmechanismus wird verborgen.

Transitive Persistenz

Verallgemeinerung der Persistenz durch Erreichbarkeit:

Für alle Felder können individuell folgende Optionen eingestellt werden (in den XML-Metadaten):

- **Feld soll überhaupt nicht in der Datenbank verändert werden, wenn eine Datenbankänderung im referenzierenden Objekt vorgenommen wird.**
- **Feld soll genau dann in der Datenbank aktualisiert werden, wenn eine Aktualisierung im referenzierenden Objekt vorgenommen wird.**
- **Feld soll genau dann aus der Datenbank gelöscht werden, wenn eine Löschung des referenzierenden Objekts vorgenommen wird.**
- **Feld soll genau dann in der Datenbank verändert werden, wenn eine Datenbankänderung im referenzierenden Objekt vorgenommen wird.**

In JDO entspricht das der Unterteilung in First-Class und Second-Class-Objects, dort allerdings wesentlich grober.

Inheritance mapping strategies

Hibernate ermöglicht es, zwischen folgenden Abbildungsmechanismen für die Vererbung zu wählen (einstellbar für jede Klasse in den Metadaten):

- **Table per concrete class**
  Die Klasse und all ihre Unterklassen bekommen jeweils eine eigene Tabelle.

- **Table per class hierarchy**
  Es gibt eine einzige Tabelle für die gesamte Klassenhierarchie.
  Die Zugehörigkeit zu einer bestimmten Klasse wird in einer separaten Spalte abgespeichert.

- **Table per subclass**
  Es gibt gesonderte Tabellen für die Unterklassen.
  Dort werden aber nur die Felder abgespeichert, die nicht geerbt werden.

Beim nächsten Mal:

Hibernate, Teil 2