Datenintegration & Verwaltung

Methoden zur Integration und Verwaltung von Portaldaten

Dr. Andreas Walter

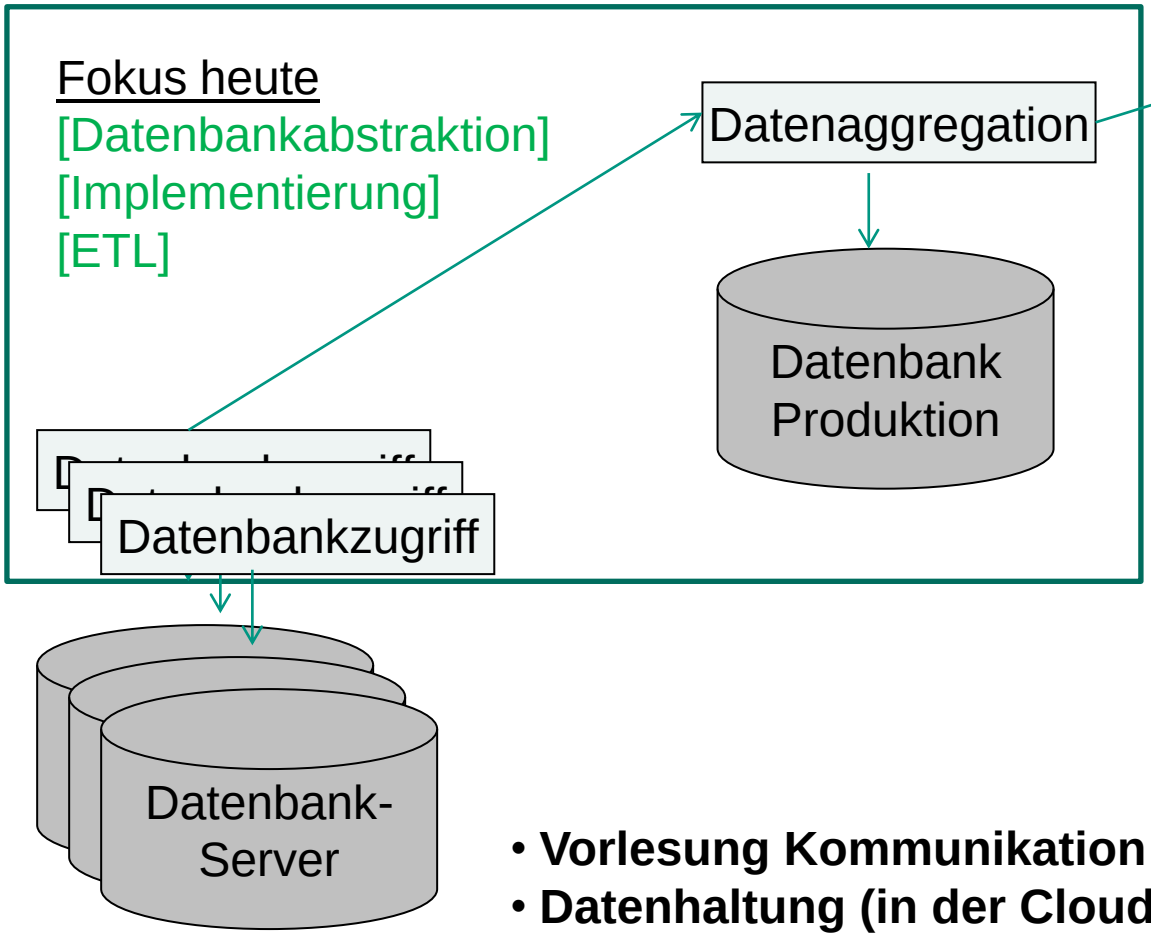
Klick-And-Bau

Informationsintegration und Webportale

Inhalt

- Motivation
- **Verwaltung von Daten**
  - Datenbankzugriff
  - Objektrelationales Mapping
    - Leichtgewichtiges Mapping: Beispiel mittels Spring
    - Schwergewichtiges Mapping: Beispiel mittels Hibernate
- **Integration von Daten**
  - Pull and Push Verfahren
  - ETL Verfahren : Extract – Transform - Load

Abgrenzung: Datenfluss / Fokus



- **Analyse grosser Datenbestände – Böhm**
[Datawarehouse] [Datamining] [Clusering]

- **Vorlesung Kommunikation / Datenhaltung** (Zitterbart)
- **Datenhaltung (in der Cloud)** – Böhm
[SQL] [Transaktionen] [Persistierung]

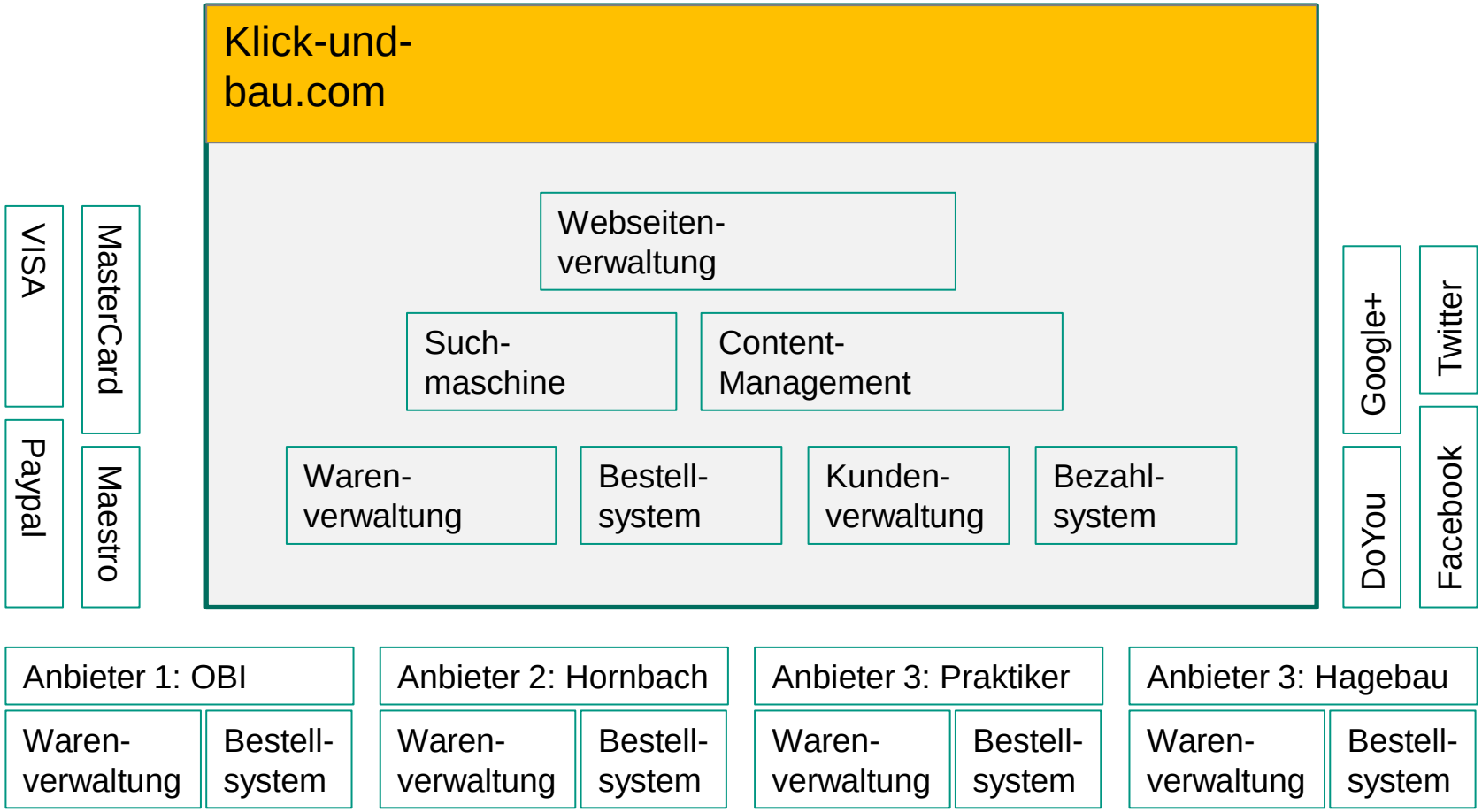
Portal: Erstellung von Produktkatalog

Klick-und-bau.com

Produkte mehrerer Anbieter

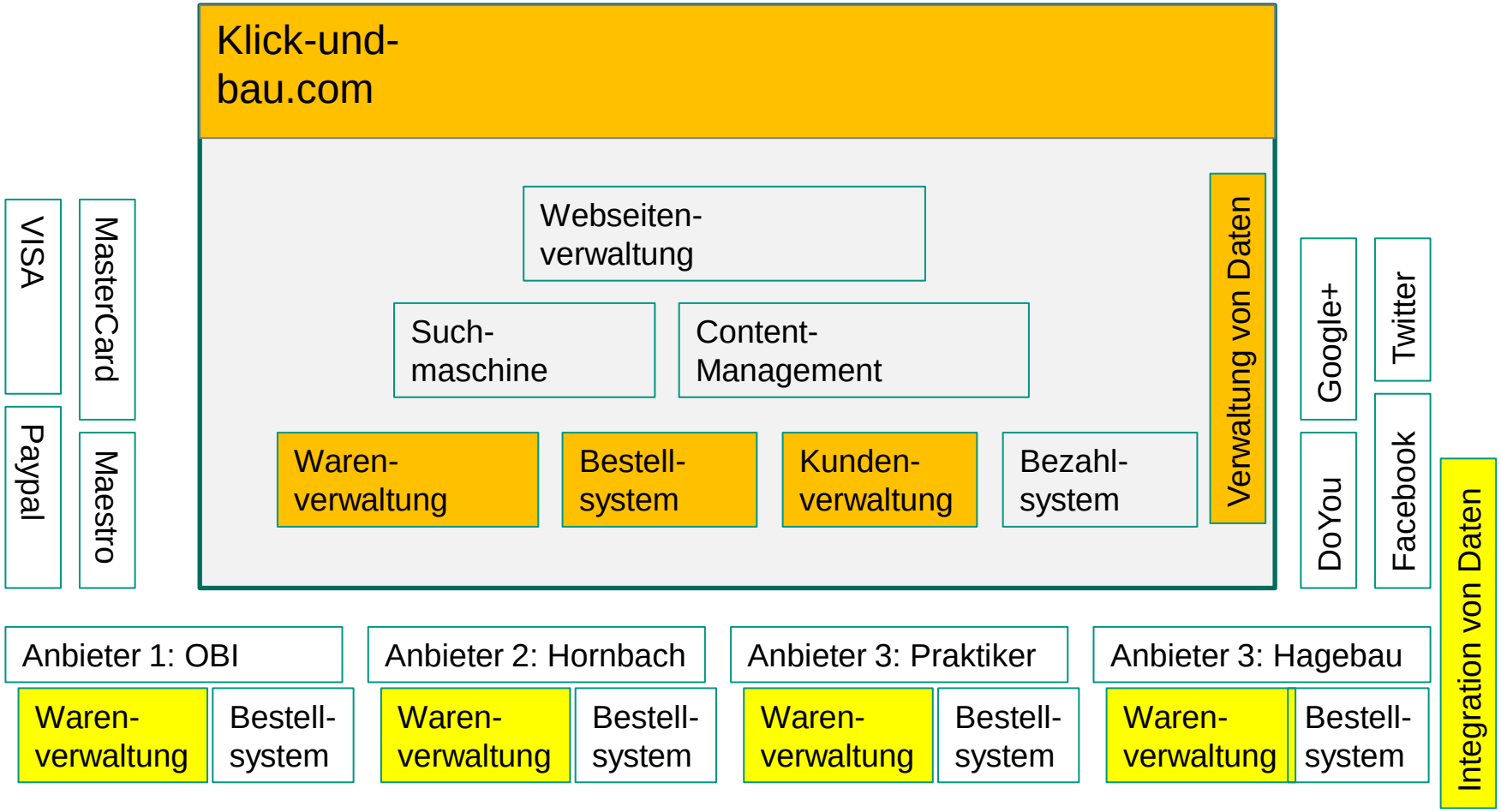
- Einheitliche Integration der Artikeldaten
- Geeignete Zusammenführung von Daten: Duplikate erkennnen, einordnen in Produktkatalog
- Einheitliche Such- und Filterfunktion

Motivation: Datenquellen im Webportal



Motivation: Datenquellen im Webportal

Es müssen eine Vielzahl von Daten integriert / verwaltet werden.



Resultierende Fragen und Probleme

- **Verwaltung von Daten**
  - Speicherung der Daten : Interaktion mit Datenbank(en)
  - Integration der Daten in die Anwendung : Daten -> JavaObjekte

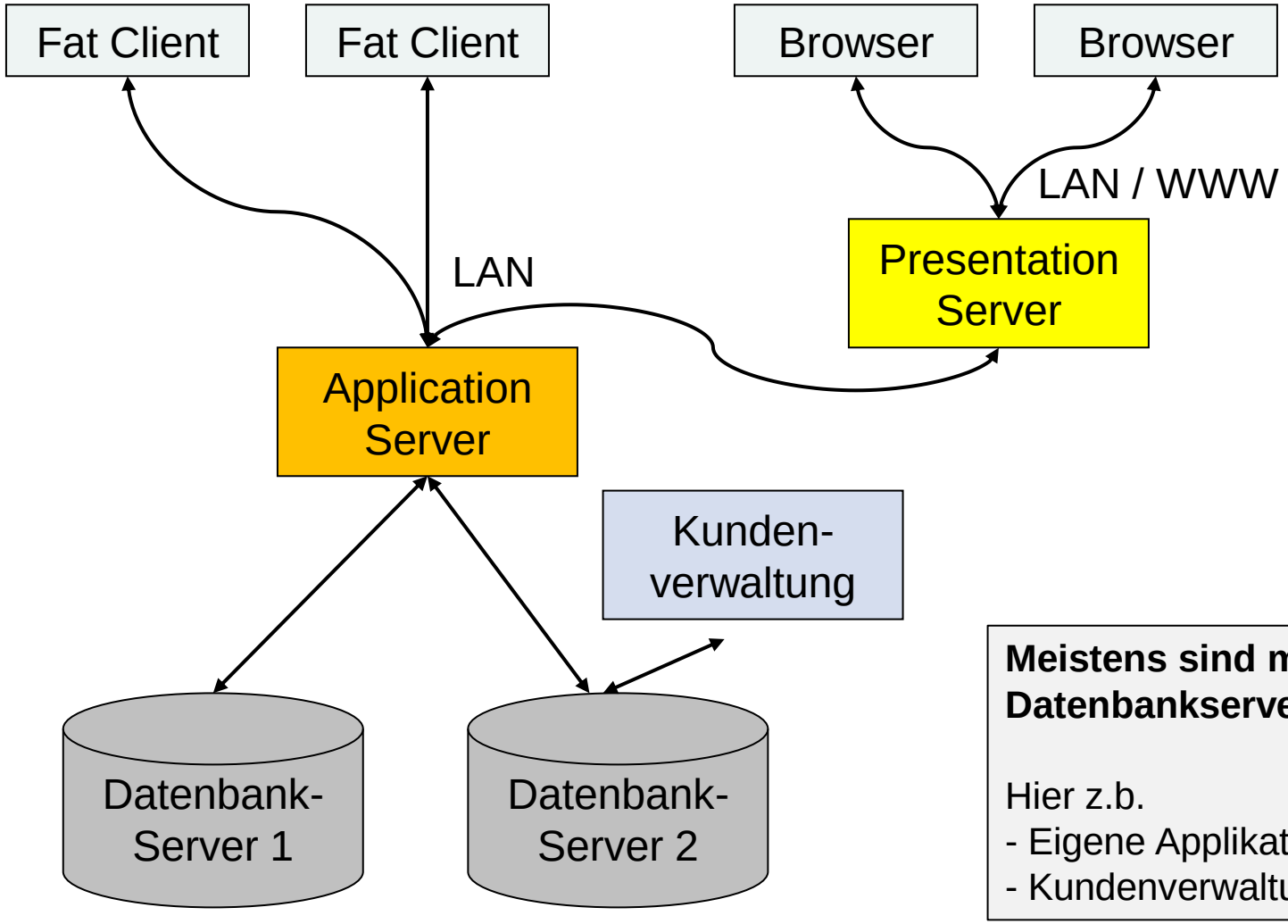
- **Integration von Daten**
  - Erhalt der Daten: Pull vs. Push
  - Formate der Daten
  - Erkennung von Duplikaten etc.

Gezeigt werden
-Verfahren und Vorgehen aus der Praxis.
- Umsetzungen mit generellen Beispielen / kein Programmier-kurs.

Erstellung von Datenbanken / SQL etc. siehe Vorlesungen Prof. Böhm

VERWALTUNG VON DATEN

Ausgangspunkt: Mehrschichtenarchitektur



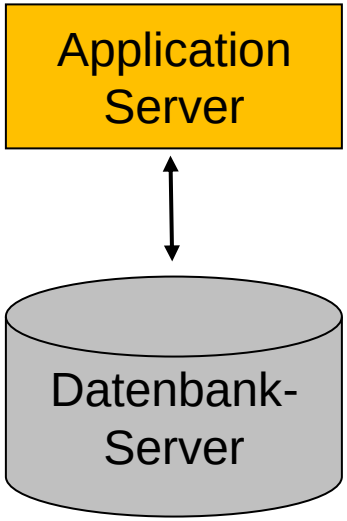
Meistens sind mehrere Datenbankserver im Einsatz.

Hier z.b.
- Eigene Applikation: mySQL
- Kundenverwaltung: Oracle

Datenbankzugriff

Herstellerspezifische Unterschiede

- SQL Standard und Erweiterungen
- Protokoll für Zugriff
- Authentifizierung
- Wartung



Generelle Unterschiede

- Zu verwendende Datenbank
- Benutzername / Passwort
- URL / Port

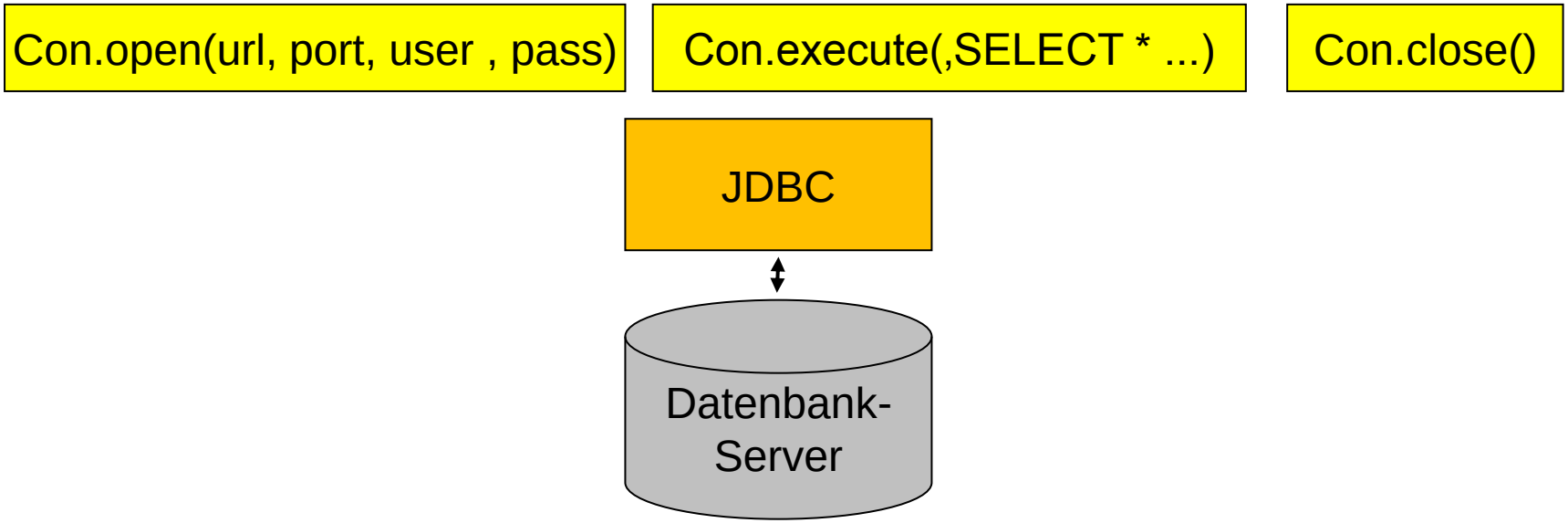
Lösung: Abstraktion von Zugriff auf Datenbank

- vendor: mysql
- version: 5.5
- url: db2.somehost.com
- port: 3306
- database: kbau
- user: aw
- pass: xyz

Abstraktion von Zugriff auf eine Datenbank (1)

Datenbank-APIs

- Verdecken herstellerspezifische Protokolle und systembezogene Eigenschaften der Datenbanken
- Verfügbar in den gängigen Programmiersprachen, z.b. JDBC in JAVA
- Bieten Methoden zum Verbindungsaufbau, Abbau und Ausführen von SQL-Statements



Abstraktion von Zugriff auf eine Datenbank (2)

- **Vorteil Datenbank-API**: verdeckt systemspezifische Eigenschaften
- **Nachteil**: Verbindungsaufbau, Halten von Verbindungen, Abbau etc muss manuell verwaltet werden (Sehr fehleranfällig!)

Lösung: Verwendung von Datenbank-Tool Bibliotheken.

Beispiel in JAVA: Apache Commons DBCP
(http://commons.apache.org/dbcp/)

Ermöglicht unter anderem:

- Konfiguration von zu verwendender Datenbank-API
- Konfiguration von Anzahl zu verwendenden, parallelen Verbindungen, deren Zuweisung und Aufrechterhaltung

  => deutlich schnellerer Einzelzugriff, Parallele Verbindungen.

Beispiel für XML Konfiguration von DBCP

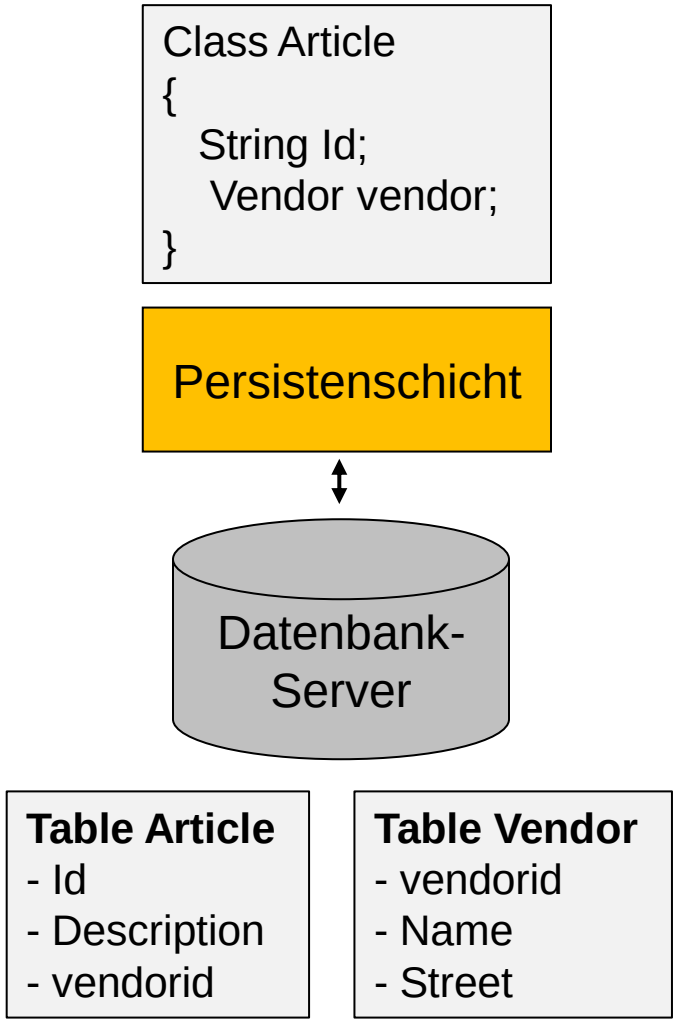
```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
        destroy-method="close">
    <property name="driverClassName"><value>org.postgresql.Driver</value>
    </property>
    <property name="url„><value> db2.somehost.com:3306</value>
    </property>
    <property name="username"><value>aw</value></property>
        <property name="initialSize">
            <value>10</value>
        </property>
        <property name="maxActive">
            <value>100</value>
        </property>
        <property name="maxIdle">
            <value>300</value>
        </property>
    </bean>
```

Abstraktion von Zugriff auf eine Datenbank (3)

- Überwiegende Anzahl von Datenbanken sind RDBS – relational
- Programmiersprachen wie JAVA sind objektrelational

Erfordert ORM (object relational mapping)

- Teil der Persistenzschicht der Anwendung
- Kennt die vorhandenen Klassen zur Datenspeicherung und
- Kennt die vorhandenen Tabellen in der relationalen Datenbank
- Verwaltet deren korrekte **Transformation** und **Synchronisation**



Arten des Objektrelationalen Mappings

- **Leichtgewichtige Rahmenwerke für ORM**
  - Unterstützen die Anfrageausführung, z.B. Die Verwaltung von Transaktionen
  - Bieten Methoden zum einfachen Iterieren über Tabellen an, um von Tabelleninhalt in Objektinhalte zu transformieren ; Bieten das vereinfachte Erstellen von SQL Anfragen an
  - Verdecken nicht von spezifischen Eigenschaften von SQL und deren Erweiterungen des Datenbanksystems.
- **Schwergewichtige Rahmenwerke für ORM**
  - Verwalten selbständig alle Anfragen an die Datenbank, inkl. Transaktionen
  - Verwalten selbständig alle Transformationen von den Tabellen der Datenbank zu den Objekten der Anwendung.
  - Verdecken vollständig von allen spezifischen Eigenschaften von SQL des Datenbanksystems
  - Erstellen und verwalten selbständig alle notwendigen Tabellen im Datenbanksystem.

Einsatz leichtgewichtiger Rahmenwerke für ORM

Bewertung leichtgewichtiger Rahmenwerke für ORM

Nachteil

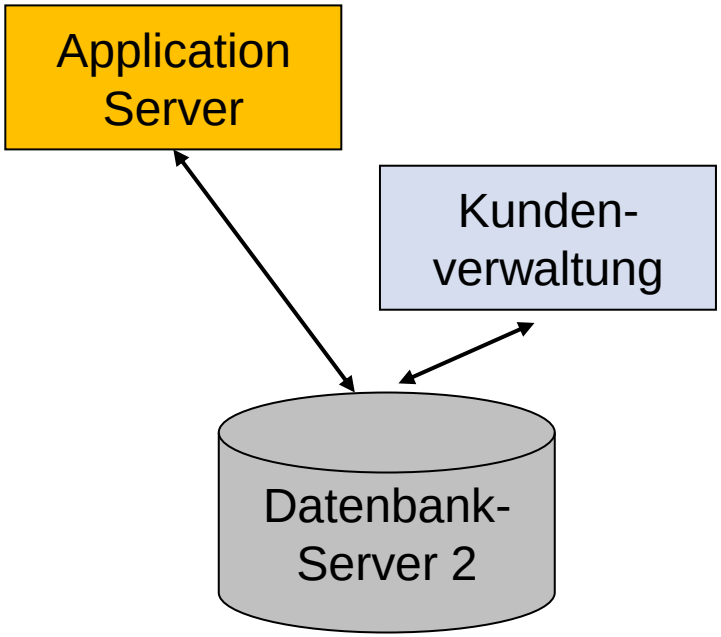
- Entwickler müssen SQL Syntax des Datenbanksystems beherrschen

Vorteil

- Einfluss auf Tabellenschema
- Umfangreiche Optimierung möglich

Übliches Einsatzgebiet

Direkter Datenbankzugriff auf bestehende Datenbanken



Beispiel: Spring für JAVA

- Repräsentiert ein leicht (und schwer) gewichtiges Rahmenwerk für ORM für die Programmiersprache JAVA

.. Hier nur ein fokussierter Einblick in Spring !

- Inversion of Control: XML basierte Konfiguration von zu verwendender Implementierung (Factory)
- Roo: Rapid application development
- Spring Data: Datenzugriff .

Beispiel hier: selbskonfiguriertes ORM.

(Anmerkung: Spring auch als schwergewichtiges Rahmenwerk in Verbindung mit Hibernate möglich)

Selbstkonfiguriertes ORM mit Spring

Anforderung: Zugriff auf Daten in Drittsystem (z.b. Artikelverwaltung von Anbieter A) soll ermöglicht werden.

Erfordert:

1. Konfiguration der Klassen und Interface für Daten in der eigenen Anwendung
2. Definition des zu verwendenden SQL Statements
3. Implementierung des ORM
4. Konfiguration der Spring Beans (XML)

1: Spring: Klasse und Interface

Ziel: Artikeldaten aus Datenbank von Anbieter A laden

```
Class Article
{
    String Id;
    Vendor vendor;
}
```

Interface articleDao

```
Public List<Article> loadNewArticlesFromAnbieterA();
```

2 Definition des zu verwendenden SQL Statements

Alle nötigen SQL-Anfragen
an Datenbank müssen definiert werden
(oder werden von Anbieter vorgegeben)

Zum Laden neuer Artikel soll
verwendet werden:

String sqlStatement=
‚*SELECT a.id, a.description, v.vendorid, v.name, v.street FROM article a, vendor v WHERE a.vendorid=v.vendorid order by v.name*‘

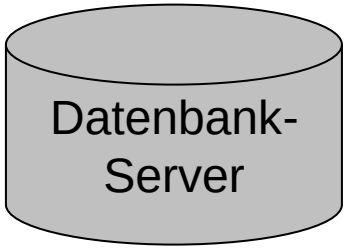


Table Article
- Id
- Description
- vendorid

Table Vendor
- vendorid
- Name
- Street

3. Implementierung des ORM

```
public class ArticleQuery extends TypedMappingSqlQuery<Article> {
    ArticleQuery(final DataSource ds) {
        super(ds,"<sql_statement>");
        // super.declareParameter(new SqlParameter("sid",
         Types.INTEGER));
        compile();
    }
    protected Article mapRow (final ResultSet rs, final int rowNum)  {
                Article article=new Article();
                article.set(rs.getInt(1));
                ..
                return article;
        }
}
```

4. Konfiguration der Spring Beans (XML)

```
<bean id=„articleDao"
        class=„de.fzi.aw.vorlesung.articleDaoImpl"
        scope="singleton">

        <property name="dataSource">
                <ref bean="dataSource" />
        </property>

    </bean>
```

(hier auch gezeigt: Inversion of Control für articleDao))

Einsatz schwergewichtiger Rahmenwerke für ORM

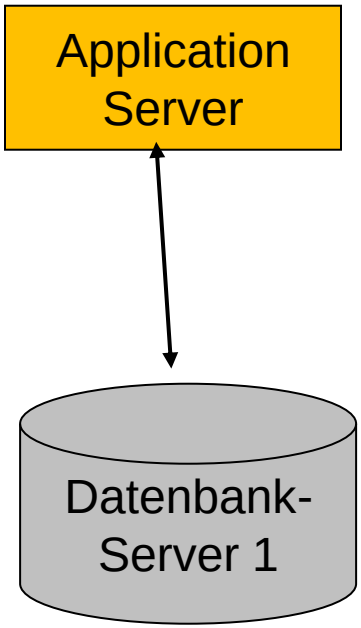
Bewertung schwergewichtiger Rahmenwerke für ORM

Nachteil

- Kein (bzw. geringer) Einfluss auf resultierendes Datenbankschema

Vorteil

- *„Programmierung“* von Datenbankanfragen
- Komplette Abstraktion von der Datenbank



Übliches Einsatzgebiet

Objekte der eigenen Applikation

Beispiel: Hibernate (JPA)

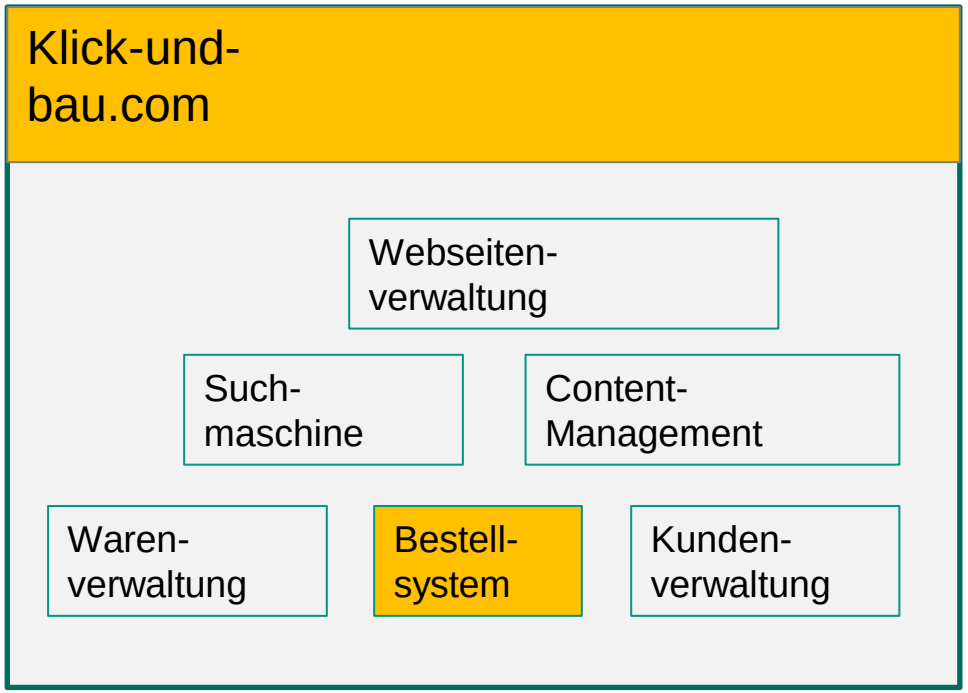
ORM mit Hibernate

Anforderung: Zugriff auf Datenobjekte aus der Anwendung heraus, ORM soll verwaltet sein. Beispiel: Bestellsystem.

Verwendung von Hibernate für das Bestellsystem erfordert:

1. Erstellung von Persistenz - Objekten
2. Annotation der Klasse und Attribute
3. Definition von Datenbankabfragen mit Hibernate Query Language (HQL)
4. Definition und Implementierung einer Abfrageschnittstelle.
5. Konfiguration des Entity-Managers

1) Erstellung von Persistenz - Objekten




```
public class Order
{
    private String orderId;
    private List<Article> articles;
    private Customer customer;
}
```

2) Annotation der Klasse und Attribute

```
@javax.persistence.Entity
@Table(name = „order")
public class Order
{
    @ID
    private String orderId;

    @ManyToMany
    @JoinTable(name = „articlesinorder")
    private List<Article> articles;

    @OneToOne(optional = true, cascade = CascadeType.ALL)
    @JoinColumn(name = „customeroforder")
    private Customer customer;
...
```

- Annotation als Persistenzklasse
- Festlegung der Tabelle: order

- Festlegung, welches Attribut als Primärschlüssel dient

Weitere Relationsarten:
- **OneToMany**: Elemente exklusiv der Klasse zugewiesen
- **ManyToOne**: geteiltes Element

3) Definition von Datenbankabfragen mit HQL

Hibernate Query Language (HQL) ist eine objketbezogene Abfragesprache innerhalb von Hibernate.

- Die Abstraktionsschicht in Hibernate übersetzt die Anfrage auf die unterstützten Datenbanken in spezifische SQL Anfrage (.. Abfolgen)
- Das komplette ORM wird hierbei verdeckt

Einige Beispiele für NamedQueries (vgl. Prepared Statements) SQL:

```
@NamedQueries({
    @NamedQuery(name = "getAllOrders",
    query = "FROM Order ") })
```

.. FROM **Order**
=> Abfragen direkt auf Objekten!

.. Fragt alle Order Objekte an, intern eine Abfolge von SQL Anfragen an Tabellen für Artikel und Kunden.

.

4) Definition und Implementierung einer Abfrageschnittstelle (1)

```
Interface CrudOrders
{
    Order createOrder(Order order);
    Order updateOrder(Order order);
    List<Order> getAllOrders();
    void deleteOrder(Order order);
}
```

- Exemplarisch eine Abfrageschnittstelle zum Erstellen (C) , Laden (R) , Aktualisieren (U) und Löschen (D) von Bestellungen [CRUD]

4) Definition und Implementierung einer Abfrageschnittstelle (2)

```
@TransactionAttribute(javax.ejb.TransactionAttributeType.REQUIRED)
public class CrudOrders Bean implements CrudOrders{

@PersistenceContext
private EntityManager entityManager;

public Order createOrder(Order order)
{
    entityManager.persist(order);
    return order;
}
```

4) Definition und Implementierung einer Abfrageschnittstelle (3)

```
public Order updateOrder(Order order)
{
    return entityManager.merge(order);
}

public List<Order> getAllOrders()
{
     return entityManager. createNamedQuery(„getAllOrders”).
    getResultList();
}

Public void deleteOrder(Order order)
{
    entityManager.remove(order);
}
```

5) Konfiguration des Entity-Managers (1)

```
<persistence-unit name="order-unit" transaction-type="JTA">
  <provider>org.hibernate.ejb.HibernatePersistence</provider>
  <jta-data-source>java:/orderDatabase</jta-data-source>
  <properties>
    <property name="hibernate.hbm2ddl.auto" value="update" />
    <property name="hibernate.default_batch_fetch_size" value="16" />
    <property name="hibernate.format_sql"       value="true" />
    ...
  </properties>
</persistence-unit>
```

Erstellungsstrategien für Datenbank
- update: Schemaupdate bei start
- create: Schema wird immer neu erstellt
- create-drop: Schema wird bei Stop der Anwendung gelöscht
- validate: Validierung, ob Tabellenschema valide für Objekte (für Produktionbetrieb)

5) Konfiguration des Entity-Managers (2)

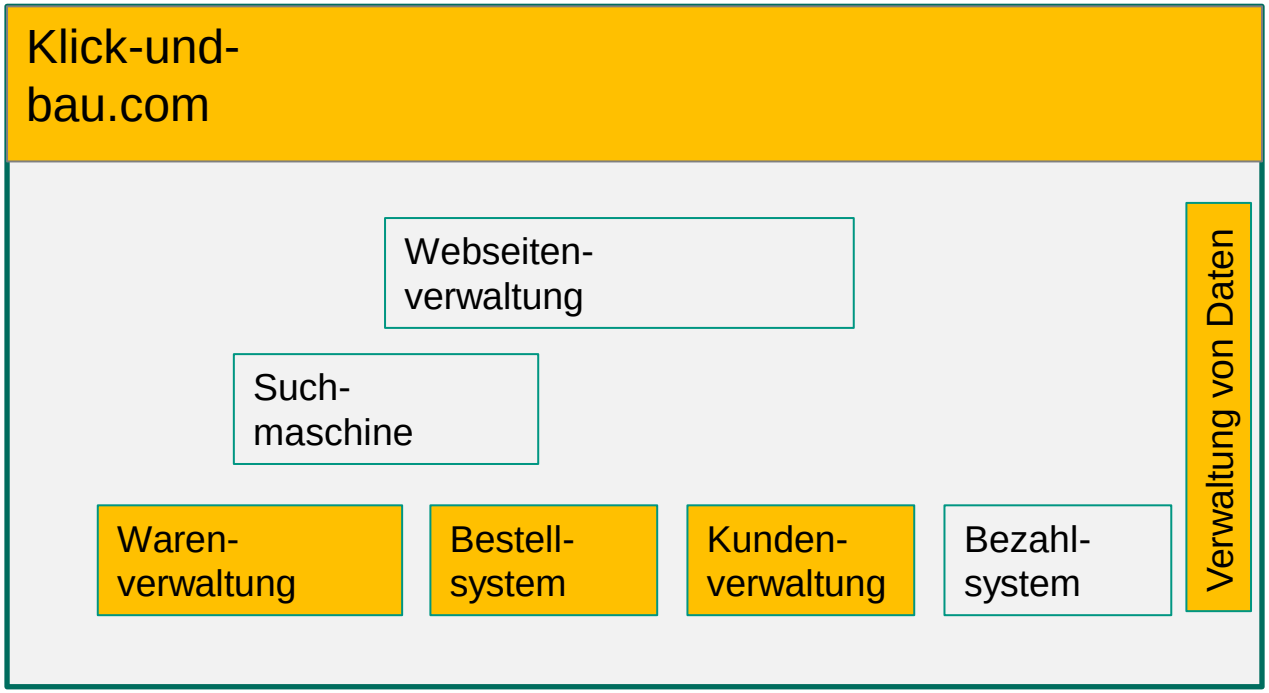
Konfiguration der Datenbank orderDatabase

z.b. In Jboss Database, standalone.xml

```
<datasources>
    <datasource jndi-name="java:/orderDatabase" pool-
    name=„orderDatabase" enabled="true" use-java-context="true">
    <connection-url>#db.connectionurl#</connection-url>
    <driver-class>#db.driver#</driver-class>
    <driver>#db.vendor#</driver>
    <security>
            <user-name>#db.user#</user-name>
            <password>#db.pass#</password>
    </security>
```

Zusammenfassung: Verwaltung von Daten

Ziel war: Verwaltung von Daten im Portal selbst

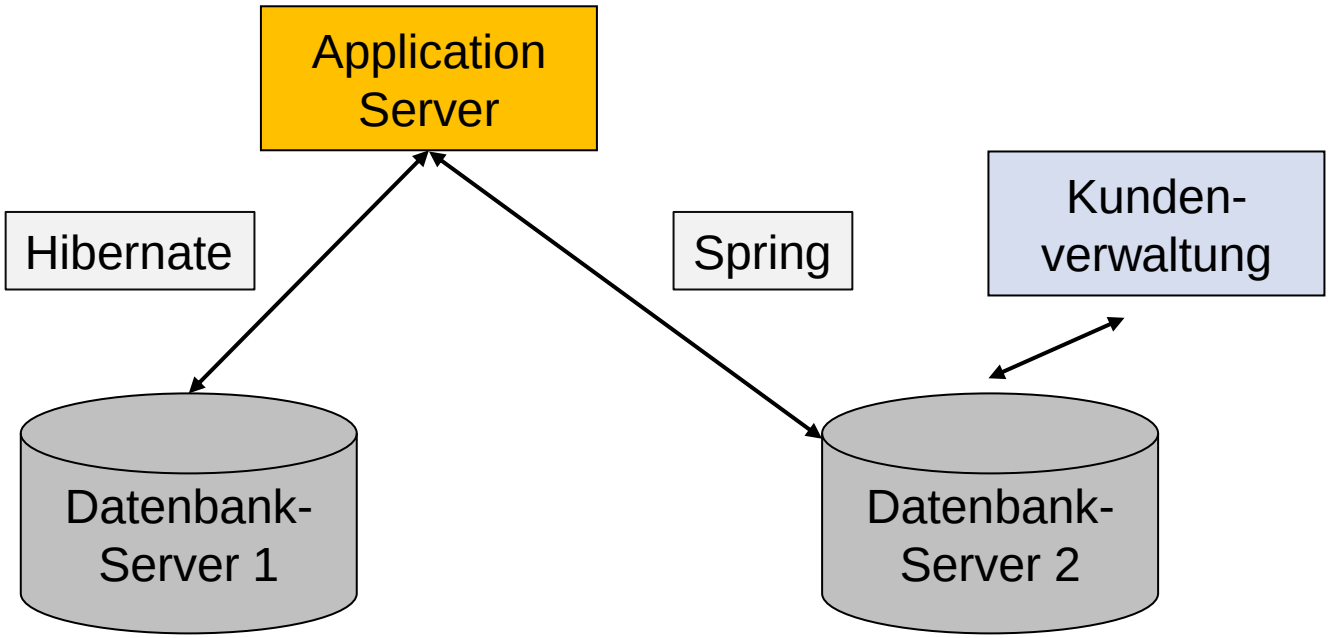


Umsetzung: Einsatz von ORM Verfahren

- Daten in externen Datenbanksystemen: leichtgewichtiges ORM
- Daten in der eigenen Anwendung: schwergewichtiges ORM

=> Weitestgehende Abstraktion von Datenbanksystem und SQL

classOrder() class Customer() class Article() ...

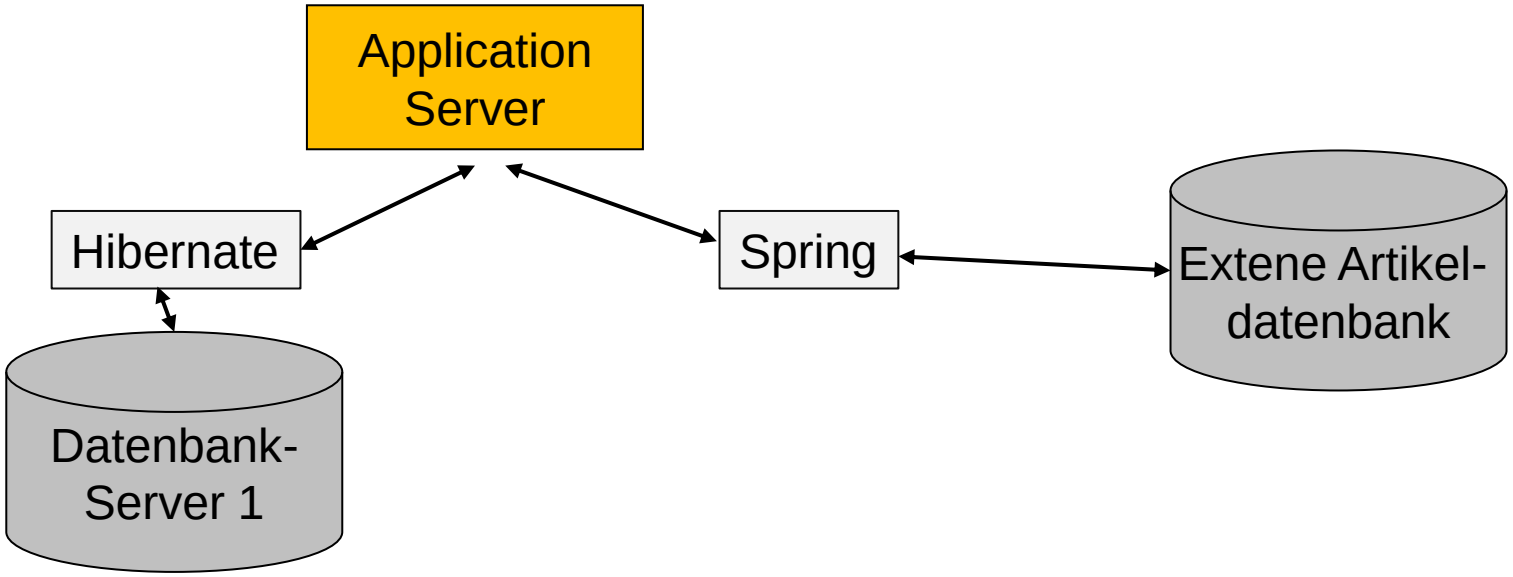


ORM Verfahren für Portal kombinierbar

Beispiel: Integration von Artikeldaten von Anbieter

- Spring: lädt Daten von externer Datenbank, verwendet Article Klasse
- Ruft in Hibernate addArticle auf => Daten geladen in eigener DB

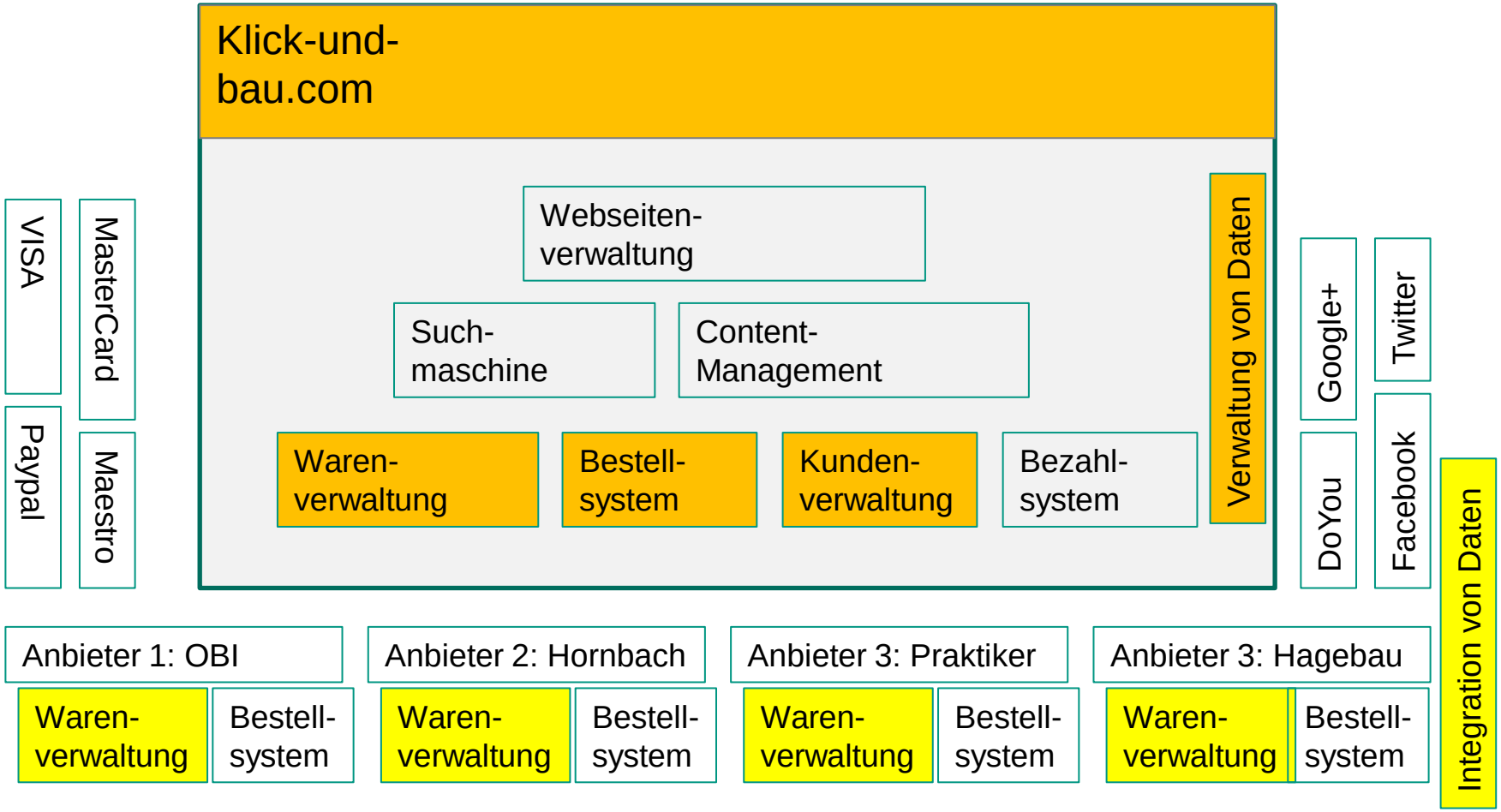
classOrder() class Customer() **class Article()** ...



INTEGRATION VON DATEN AUS EXTERNEN SYSTEM

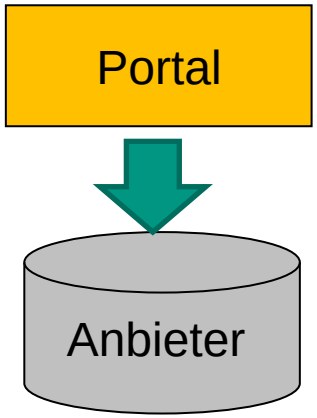
Integration von Daten aus externen Systemen

Wie werden die Daten aus externen Systemen ins Portal transferiert?

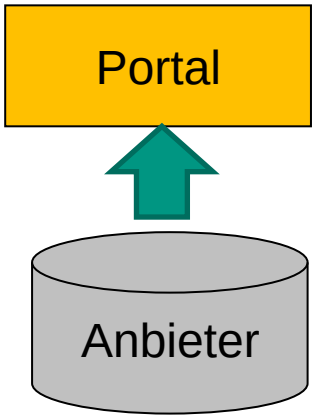


Verfahren der Datenintegration

- **Pull**
  - Zugriff auf die Daten beim Anbieter
  - Daten werden regelmässig abgefragt, zum Beispiel „neueste Artikeldaten“, „aktualisierte Artikeldaten“



- **Push**
  - Kein Zugriff auf die Daten beim Anbieter
  - Anbieter sendet Datenaktualisierung regelmässig an Portalbetreiber



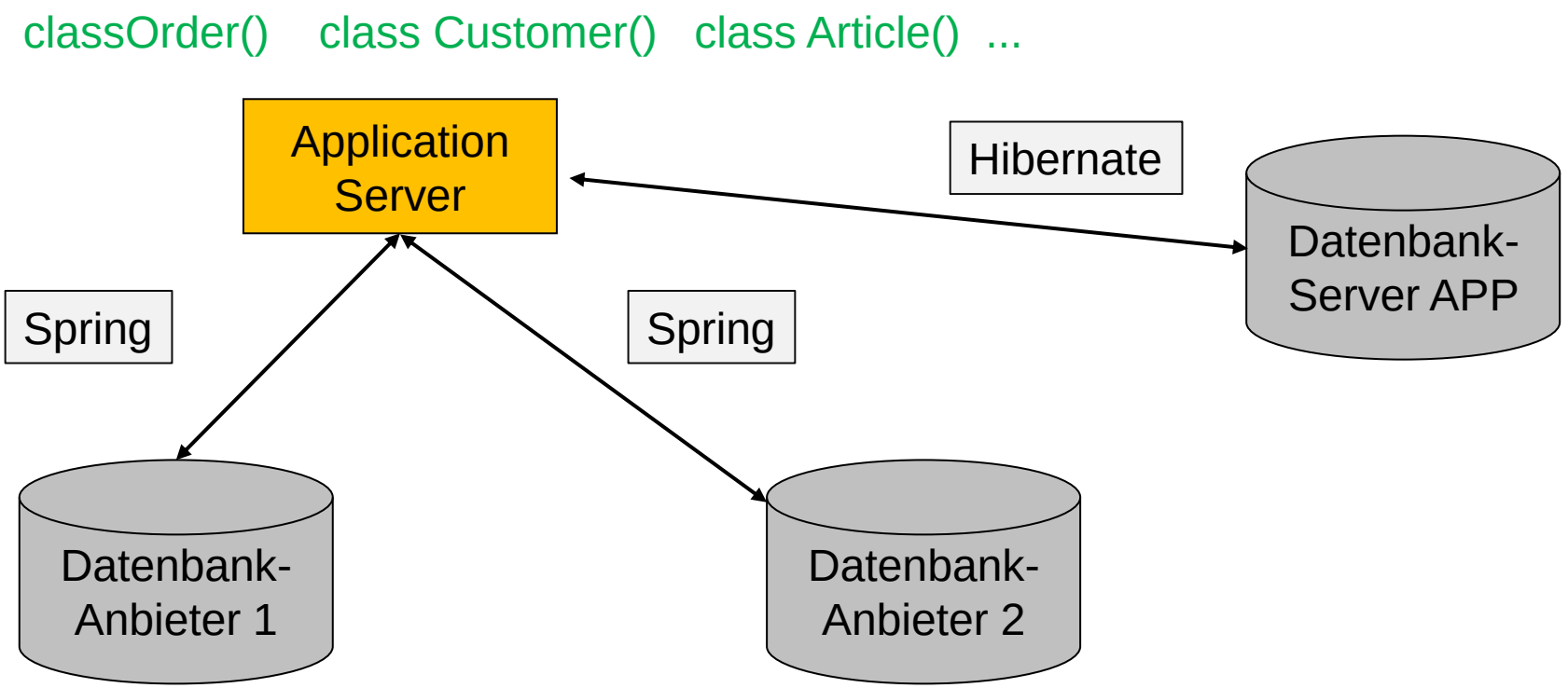
Unterscheidung

- Vollständige Aktualisierung: Alle Daten werden regelmässig gesendet
- Übermittlung der Änderungen: Nur Änderungen werden übermittelt

Pull – naheliegender Fall: Datenbankzugriff

Alle Anbieter ermöglichen Datenbankzugriff

=> Verwendung von leichtgewichtigen ORM Verfahren



Datenbankzugriff in Praxis oft unrealistisch!

Gründe

- Datenbank des Anbieters ist nicht an das Internet angebunden.
- Systemadministrator verweigert den Zugriff, häufig wegen
  - Sicherheitsbedenken
  - Performanz : Angst, dass man mit eigenen Anfragen das Produktivsystem verlangsamt.
- Einschränkung durch Systemhersteller:
  Daten sind z.B. verschlüsselt abgelegt, nur sehr kostenpflichtige Anpassungen würden Abfrage erlauben.

Angebotene Lösung häufig:

Daten werden in Reports abgefragt und gesendet.

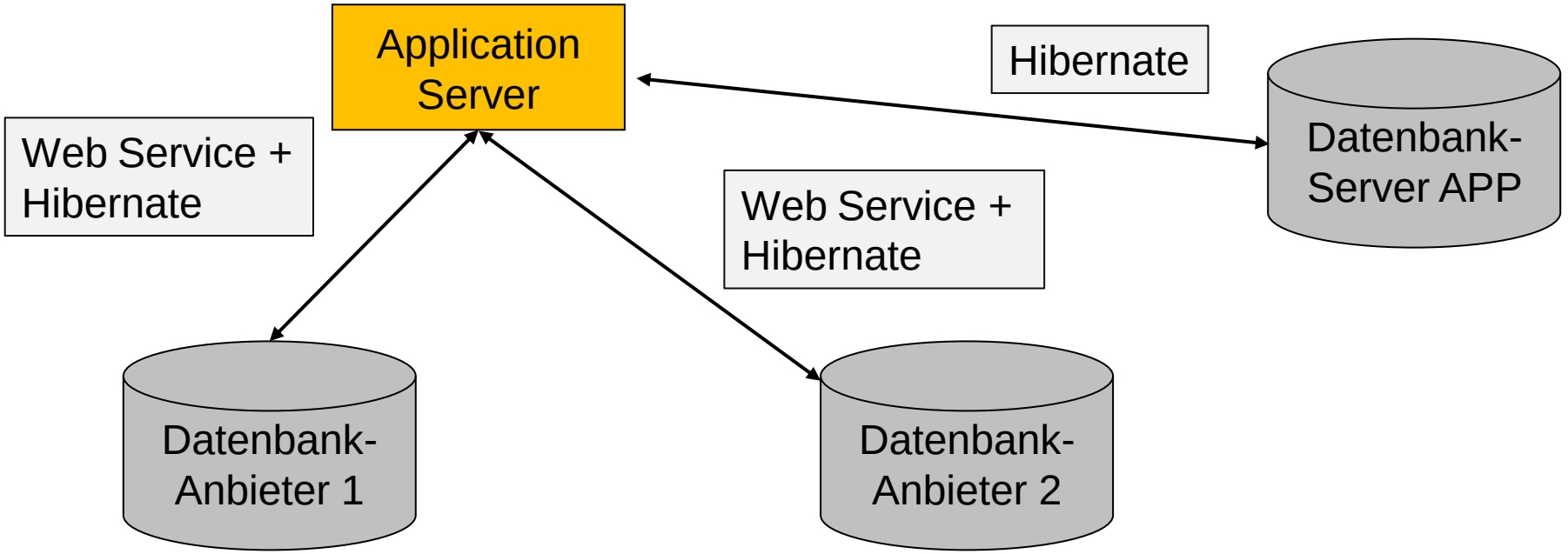
Pull – idealisierter Fall: Abfrageschnittstelle

Anbieter ermöglicht Abfrageschnittstelle (z.b Web Service)

⇒ Daten können strukturiert abgefragt werden

⇒ Noch idealer: Alle Anbieter implementieren eigene Spezifikation

classOrder() class Customer() class Article() ...



Pull: Datenabfrage per Schnittstelle (1)

Erfordert

- Anbieter implementiert entweder eigene Spezifikation einer Schnittstelle.
- Es gibt eine existierende Schnittstelle die mitverwendet werden kann.

Einbindung

- Implementierung des Clients der Schnittstelle, z.B. mit JAX-WS in JAVA
  - Anbieter stellt Webservice und Beschreibung des Dienstes zur Verfügung (WSDL)
  - Generierung von Client-Klassen => regelmässige Datenabfrage

Vorteile

- Totale Abstraktion von Datenbanksystem

Pull: Datenabfrage per Schnittstelle (2)

Beispiel einer Schnittstelle zum Abfragen von Artikeldaten

```
/* lädt die neueste Artikel seit dem eingegebenen Datum */
public List<Article> getNewestArticle(Date lastUpdated);

/* lädt die aktualisierten Artikel seit dem eingegebenen Datum */
public List<Article> getUpdatedArticle(Date lastUpdated);
```

Der Anbieter selbst implementiert intern die Datenanfragen an die Artikeldatenbank.

Push: Daten werden regelmässig übertragen

Ablauf

1. Systemadministrator bei Anbieter erstellt einen Report. Ausgabeformat ist zum Beispiel Microsoft Excel oder eine XML Datei
2. Die erstellten Dateien werden übertragen, z.B. Per Email oder mittels einenem File Transfer (sftp, scp).
3. Eigene Anwendung nimmt Datei entgegen und soll diese laden.

Probleme beim Laden der erhaltenen Dateien

- Unterschiedliche Eingabeformate (xls, xlsx, rtf, txt, xml, sql, ...)
- Unterschiedliche Schemen. Anbieter verwendet eigenes Format -> Anwendung erfordert eigenes Format
- Unterschiedliche Primärschlüssel in den Schemen

=> Lösung: Extract – Transform – Load Verfahren

Extract – Transform – Load Verfahren

Ziel

- Laden von externen Daten soll vereinheitlicht werden
- Erkennung von Duplikaten soll vereinfacht werden

Schritte

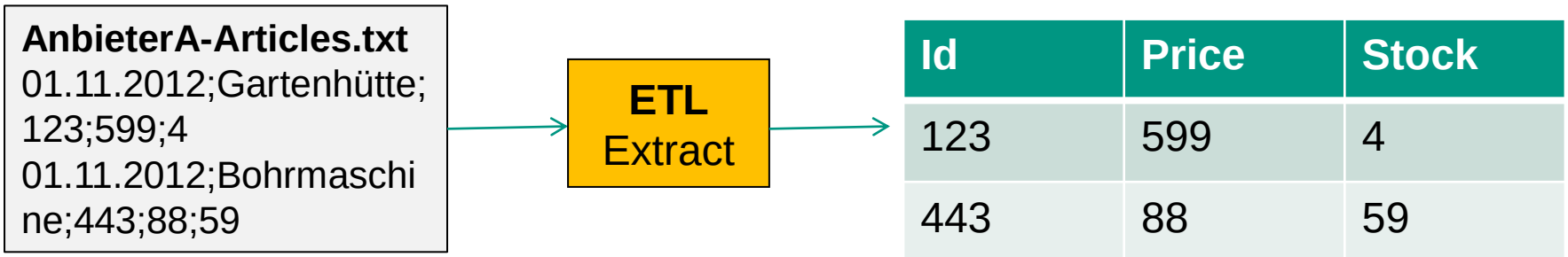
1. **Extract:** Daten werden aus einer Vielzahl von Formaten extrahiert.
2. **Transform:** Daten werden in Zielformat transformiert und aufbereitet.
3. **Load:** Daten werden in die Zieldatenbank geladen.

1) Extract

Mögliche Auslöser

- zeitgesteuert: Anbieter schickt immer zu festen Zeiten eine Aktualisierung, z.B. u Beginn einer Woche.
  Auslöser: Task Scheduler, z.b. Cron Job unter Linux
- eventgesteuert: Anbieter schickt zu beliebigen Zeiten eine Aktualisierung. Auslöser: z.B. Eingehende Email, Neue Datei in SCP

Ergebnis des Extract – Vorgangs sind Daten in strukturierter Form.

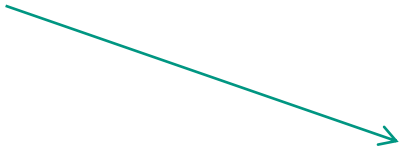


2) Transform (1)

Die Hauptaspekte der Transformation sind

- **Schematransformation**
  - Die aus Extract Schritt geladenen strukturierten Daten werden in das Zielschema transformiert

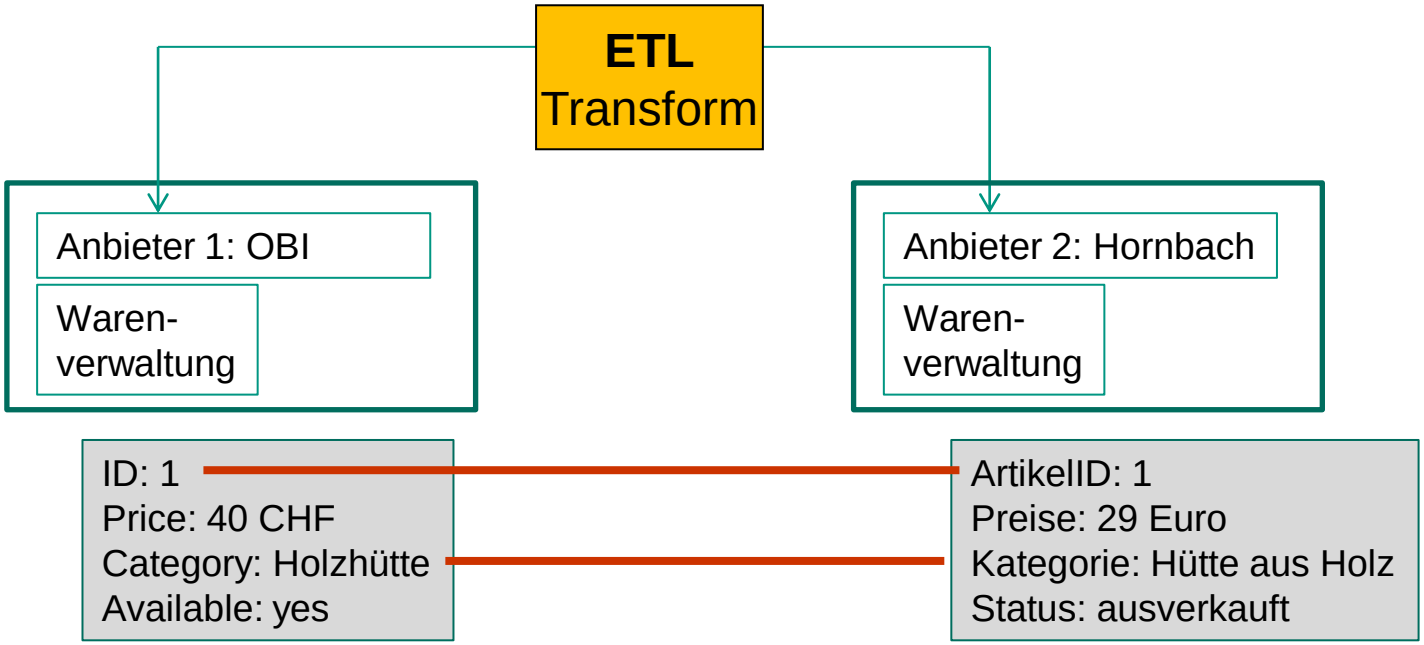
Id	Price	Stock
123	599	4
443	88	59



ArtikelId	Preis	Bestand
123	599	4
443	88	59

2) Transform (2)

- **Syntaktische Transformation**: z.B. Datumsformate
- **Semantische Transformation**:
  - z.B. Preise, hier: CHF -> Euro -> Durchführung von Berechnungen
  - z.B. Verfügbarkeit: „Available: yes“ => Verfübar: ja, „Status: ausverkauft“ => Verfügbar: no“ -> Answendung von Transformationsregeln



2) Transform (3)

- **Duplikateleminierung**: Zieldatenbank in Portal soll einen Artikel nur einmal aufführen. Falls mehrere Anbieter gleichen Artikel anbieten, soll eine Liste von Anbietern und deren Preise gezeigt werden.

  - **Duplikate bei gleichem Anbieter**: falls Artikel bereits früher gesendet, muss dieser überschrieben werden.
  - **Duplikate bei mehreren Anbietern**:
    - **Regeln**: Falls anbieterA.artikelID == anbieterB.artikelID => zusammenführen
    - **Machine Learning:**
      z.B. Natural Language Processing.
      „Hütte aus Holz“ == „Holzhütte“ => zusammenführen.
    - **Manuell unterstützt:** Portalbetreiber erstellt eigenen Artikelbaum.
      Artikel-Ids aus Anbietersysteme werden auf eigenen Artikelbaum abgebildet.

      Z.B. Artikelbaum: Gartenhütte. ID: 5
      - anbieterA.artikelID.123 == artikelbaum.id.5
      - anbieterB.artikelID.89 == artikelbaum.id.5

3) Load

- Daten werden in das Zielsystem nach erfolgter Transformation geladen.

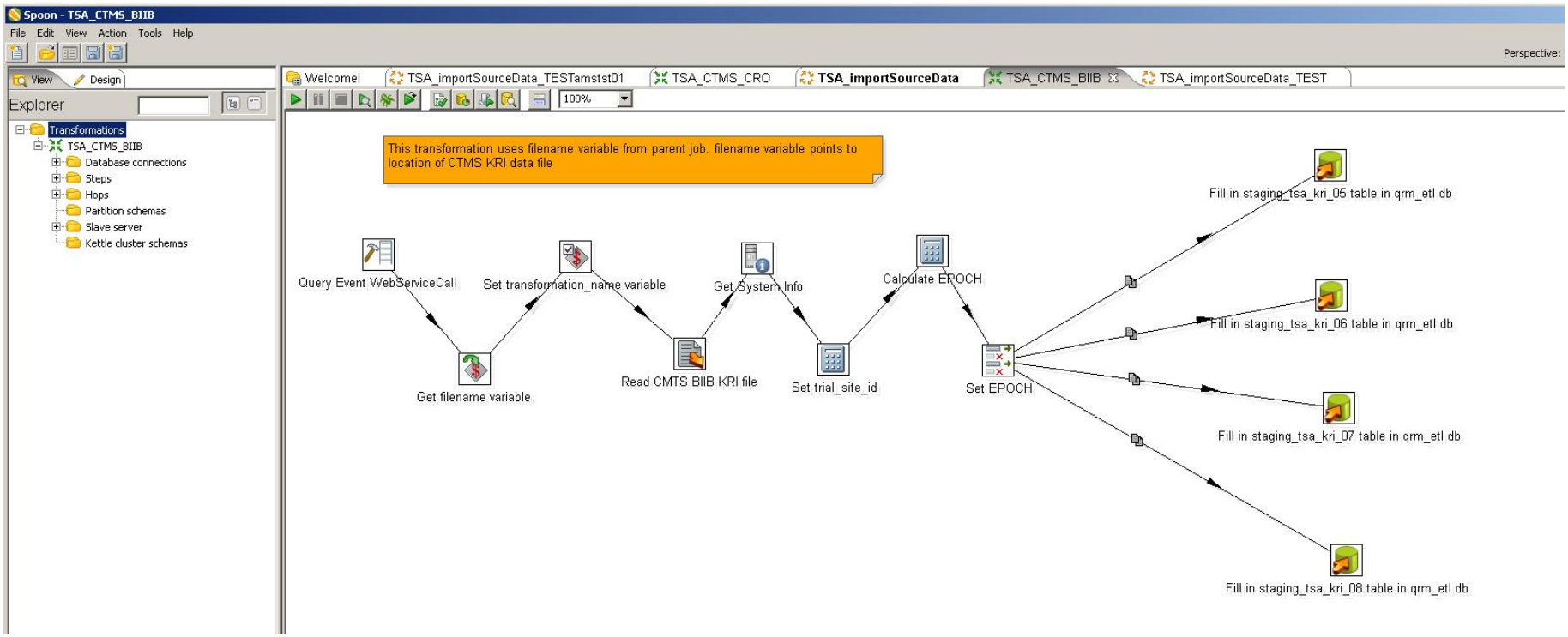
Ladestrategie ist abhängig von der Strategie der gesendeten Daten

- **Vollständige Aktualisierung**: Anbieter übersendet immer den kompletten Datenbestand.
  Load Strategie: z.B.
  bestehende Daten von Anbieter löschen -> neue Daten einspielen
- **Inkrementelle Aktualisierung**: Anbieter übersendet nur Änderungen innerhalb eines bestimmten Zeitraums, z.b. der letzten Woche,
  Load Strategie: z.B.
  - ermittle geänderte Daten, aktualisiere diese
  - Ermittle neue Daten, füge diese hinzu (lösche Daten entsprechend).

  Gefahr: ein Update wird verpasst => Daten nicht mehr synchron.

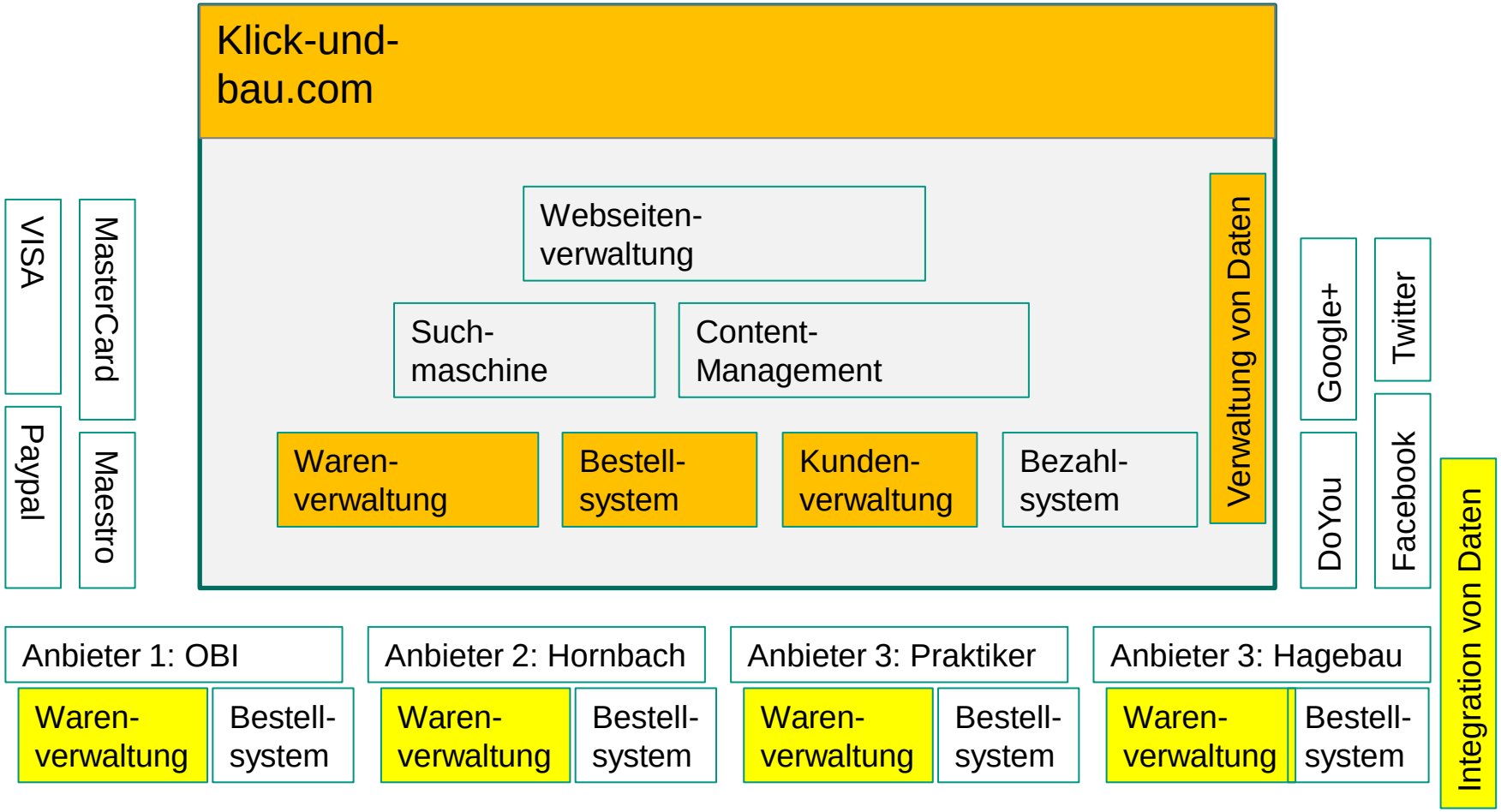
ETL Lösungen

- Vielzahl von kommerziellen Lösungen verfügbar, u.a. Microsoft, SAP, Oracle, IBM, Interessante open source Lösung: Pentaho Kettle



Zusammenfassung (1)

Fragestellung war: wie werden Daten im Portal verwaltet / integriert



Zusammenfassung (2)

- **Verwaltung von Daten**
  - Speicherung der Daten : Interaktion mit Datenbank(en)
    - Datenbankzugriff: Abstraktion von Datenbanksystemen durch Apis, Tools
    - Leichtgewichtige ORM Verfahren: Laden von Daten direkt aus Datenbank
  - Integration der Daten in die Anwendung : Daten -> JavaObjekte
    - Schwergewichtige ORM Verfahren: Verwaltung der Persiszenzobjekte, z.b. Mittels Hibernate

- **Integration von Daten**
  - Erhalt der Daten: Pull vs. Push
    - Pull: ebenfalls mittels ORM, ideal: Verfügbarkeit von Web Services
    - Push: Daten werden in verschiedensten Formaten übermittelt.
  - Formate der Daten
  - Erkennung von Duplikaten etc.
    - Einsatz von ETL Verfahren

Literatur

Datenverwaltung

- **Apache Commons DBCP**: Datenbankabstraktion
  http://commons.apache.org/dbcp/
- **Spring 3:** u.a. Leichtgewichtiges ORM.
  Get started with Spring: http://www.springsource.org/get-started
- **Hibernate:** ORM basierend auf JPA Spezifikation
  http://www.hibernate.org/

Integration von Daten

Pentaho Kettle Solutions: Building Open Source ETL Solutions with Pentaho Data Integration, Matt Casters, Wiley Verlag, 2011