Objektorientierte Datenbanken

Vorlesung 6
Sebastian Iwanowski
FH Wedel

JDO

- **Persistenzkonzept**

  Persistenzfähige Klassen, objektspezifische Persistenz, Persistenz durch Erreichbarkeit

- **Transaktionskonzept**

  mehrere Transaktionsmanagementstrategien

- **Anfragesprache (JDOQL)**

  mit objektorientiertem Fokus, Java-Syntax

- **Konzept zum Management von Datenveränderungen**

  über so genannte Lebenszykluszustände von Daten
  definiert Mechanismen für den Lebenszyklusübergang

- **Datenidentitätskonzepte**

  berücksichtigt unterschiedliche Anforderungen von Datenbank und Programm

JDO-Transaktionskonzept

Technische Realisierung:

- **in jedem PersistenceManager nur eine Transaktion gleichzeitig**

  erhältlich durch **`PersistenceManager.currentTransaction()`**

  Verschachtelte Transaktionen sind verboten (anderenfalls **`JDOUserException`**).

- **verschiedene PersistenceManager in verschiedenen Threads**

  Damit sind parallele Transaktionen möglich.

- **individuelle Einstellung des Transaktionstyps**

  durch **`Transaction.setOptimistic(bool)`**

- **Prüfmöglichkeit auf zwischenzeitliche Änderungen**

  durch **`PersistenceManager.refresh(obj)`**
  oder **`PersistenceManager.refreshAll()`**

FastObjects-Transaktionskonzept

Prinzip:

Der JDO-Standard wird um die ODMG-Implementierung erweitert

Zusätzliche Funktionalitäten:

- **verschachtelte Transaktionen sind erlaubt**
  durch **`PersistenceManagerFactories.setNestedTransactionEnabled()`**
- **es gibt die Möglichkeit, Zwischenstände abzuspeichern**
  durch **`Transactions.checkpoint(txn)`**
- **Transaktionen haben Zugriffsrechte und lösen Sperren aus**
  Schreibsperren abhängig vom Typ (pessimistisch oder optimistisch)
  Schreibzugriff kann abgeschaltet werden (durch Property-Key **`readOnly`**)
- **viele weitere Funktionalitäten in den Klassen**
  `PersistenceManagerFactories`, `PersistenceManagers`, `Transactions`

Datenbankanfragemöglichkeiten ohne Anfragesprache

a) **Extraktion eines einzelnen Objects (spezifiziert durch Name / ID)**

b) **Extraktion des Extents einer Klasse**

Ungeeignet für das Problem:

Finde Objekte mit bestimmten Eigenschaften !

a) Hoher Programmieraufwand

b) Hoher Hauptspeicherbedarf

Ziele der Objektanfragesprache OQL

1) **gezielte Extraktion von Objekten mit bestimmten Eigenschaften**

   mehr Komfort, mehr Effizienz

2) **Abfragemöglichkeiten ohne detaillierte Java-Kenntnisse**

   politischer und sozialer Grund

Für JDOQL gilt nur das 1. Ziel !

Wesentliche Merkmale von JDOQL-Queries

- **Jede Query bezieht sich auf genau eine Klasse**
- **Jede Query sucht aus einer Menge von Elementen dieser Klasse diejenigen Elemente, die eine bestimmte Eigenschaft erfüllen**
- **Die gesuchte Eigenschaft einer Query wird durch eine Booleschen Ausdruck bestimmt (*Filter* genannt)**

ODMG-OQL-Beispiel

```
Database db = new Database ();
db.open("fastObjects://LOCAL/MyBase", Database.OPEN_READ_WRITE );
Transaction txn = new Transaction( db );
txn.begin();
Extent AlleProfessoren = new Extent (db, Professoren.class);

// Frage formulieren:
String queryString = " select p.Name from p in AlleProfessoren " +
                              " where p.Rang = \"C4\" ";
// Frageobjekt erzeugen:
OQLQuery query = new OQLQuery( queryString );
// Frage stellen:
Object result = query.execute();
// Ergebnis auswerten (testen, welcher Typ zurückgegeben wurde):
if (SetOfObject.class.isInstance (result))
   SetOfObject names = (SetOfObject) result;
else if (ArrayOfObject.class.isInstance (result))
   ArrayOfObject names = (ArrayOfObject) result;

txn.commit();
db.close();
```

Dasselbe Beispiel in JDOQL

```
Properties pmfProps = new java.util.Properties();
pmfProps.put("javax.jdo.PersistenceManagerFactoryClass",
             "com.poet.jdo.PersistenceManagerFactories" );
pmfProps.put("javax.jdo.option.ConnectionURL", "fastobjects://LOCAL/MyBase" );
PersistenceManagerFactory pmf = JDOHelper.getPersistenceManagerFactory( pmfProps );
PersistenceManager pm = pmf.getPersistenceManager();
Transaction txn = pm.currentTransaction();
txn.begin();
Extent AlleProfessoren = pm.getExtent (Professoren.class, true);

// Frage formulieren:
String queryString = " Rang == \"C4\" ";
// Frageobjekt erzeugen:
Query query = pm.newQuery( AlleProfessoren, queryString );
// Frage stellen:
Collection result = (Collection) query.execute();
// Ergebnis auswerten (aus Professorenmenge muss Namenmenge gemacht werden):
Iteration iter = result.iterator();
HashSet names = new HashSet (); Professoren professor;
while (iter.hasNext()) {
   professor = (Professoren) iter.next();
   names.add (professor.name);}

txn.commit();
```

Funktionalität von JDOQL-Queries

Grundfunktionalität:

Interface PersistenceManager

public Query newQuery ();

public Query newQuery (Class class);

public Query newQuery (Class class, Collection elements);

public Query newQuery (Class class, Collection elements, String filter);

public Query newQuery (Extent elements);

public Query newQuery (Extent elements, String filter);

Interface Query

public void setClass (Class class);

public void setCandidates (Collection elements);

public void setCandidates (Extent elements);

public void setFilter (String booleanExpression);

public Object execute (); gibt immer eine Collection zurück

Funktionalität von JDOQL-Queries

Filterdefinition:

- **Der Filter entspricht einer Java-Methode, die `boolean` zurückgibt.**
- **Die Syntax des Filters ist gleich normaler Java-Syntax.**
- **Es sind fast alle Java-Operationen zulässig (Ausnahmen später).**

Filterauswertung:

- **Der Filter wird auf jedes Element der Kandidatenmenge angewandt.**
- **Die Attribute der Kandidatenelemente können im Filtercode direkt angesprochen werden. Das angesprochene Attribut bezieht sich auf das Element, das gerade ausgewertet wird. Alle Elemente, für die der Filter `true` ergibt, werden im `execute` zurückgegeben.**
- **Wenn kein Filter explizit definiert wurde, wird die gesamte Kandidatenmenge im `execute` zurückgegeben.**

Beim nächsten Mal (25.05.):
JDOQL im Detail

<u>Jetzt</u> als Vorbereitung für den
Kolloquiumsvortrag
heute um 17 Uhr:

Datenbanken in betriebswirtschaftlichen Anwendungen

OLTP (SAP)

Datawarehouselösungen

OLTP: Online Transaction Processing

Beispiele

- Flugbuchungssystem
- Bestellungen in einem Handelsunternehmen

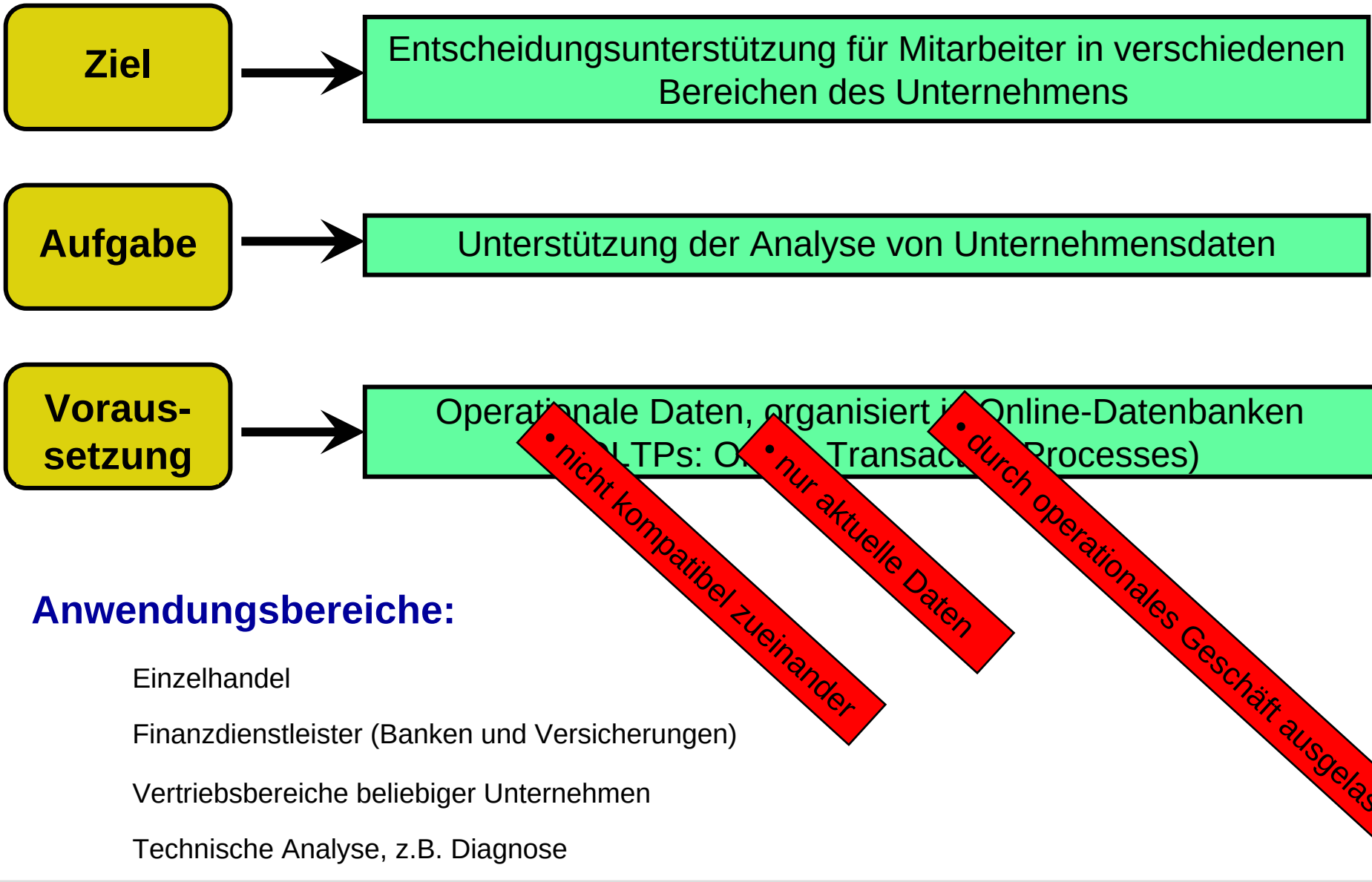
Eigenschaften

- Hoher Parallelitätsgrad
- Viele (Tausende pro Sekunde) kurze Transaktionen
- Jede einzelne Transaktion bearbeitet nur ein kleines Datenvolumen
- „mission-critical“ für das Unternehmen
- Hohe Verfügbarkeit muss gewährleistet sein

Was charakterisiert OLTPs ?

- Häufige Aktualisierungen
- Seltene Anfragen

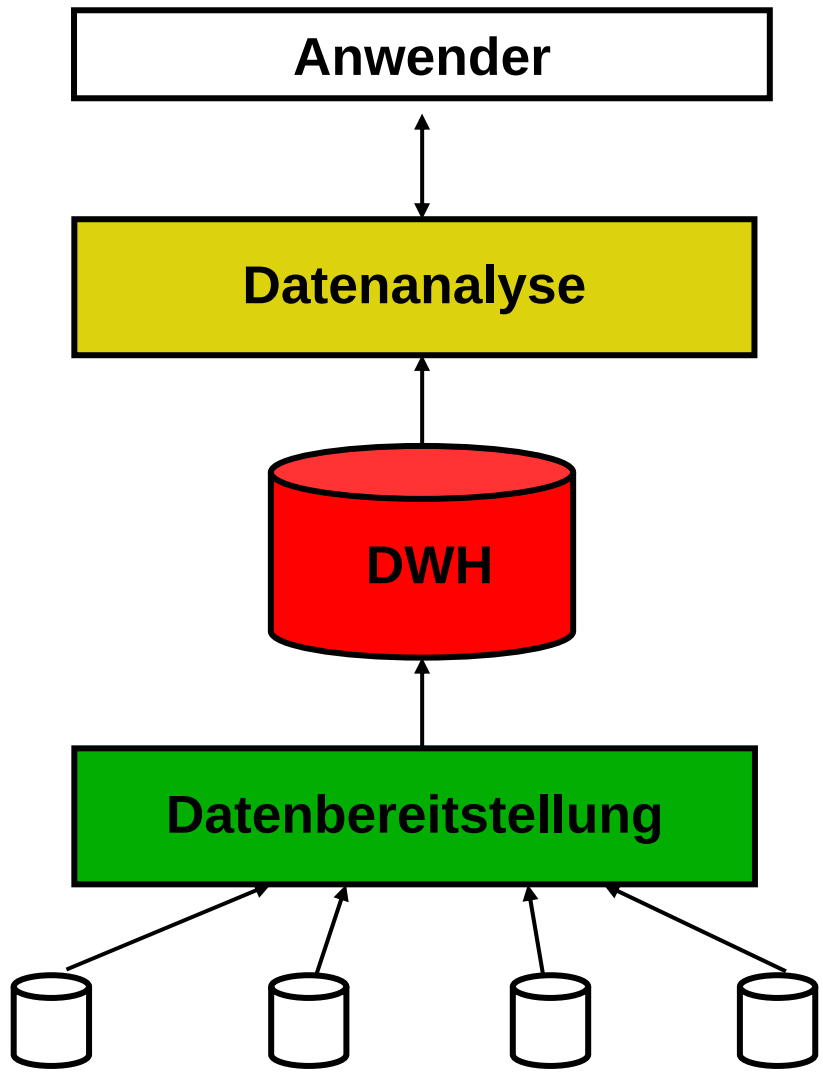
Datawarehouselösungen: Problemstellung



Anwendungsbereiche:

- Einzelhandel
- Finanzdienstleister (Banken und Versicherungen)
- Vertriebsbereiche beliebiger Unternehmen
- Technische Analyse, z.B. Diagnose

Datawarehouselösungen: Architektur



- **Seltene Aktualisierungen**
- **Häufige Anfragen**

Data Warehouse: Datenbank

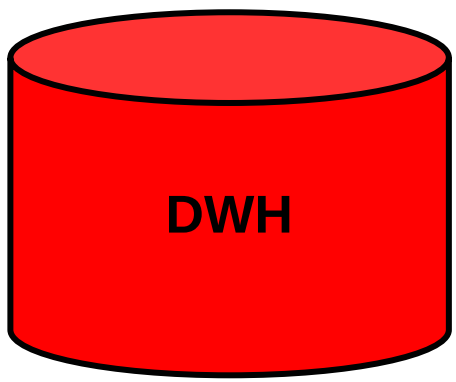
Operationale Daten aus OLTPs

Datawarehouselösungen:

Anforderungen an das DWH

Bill Inmon (1993)

Die Daten eines DWH sollen folgende Eigenschaften haben:

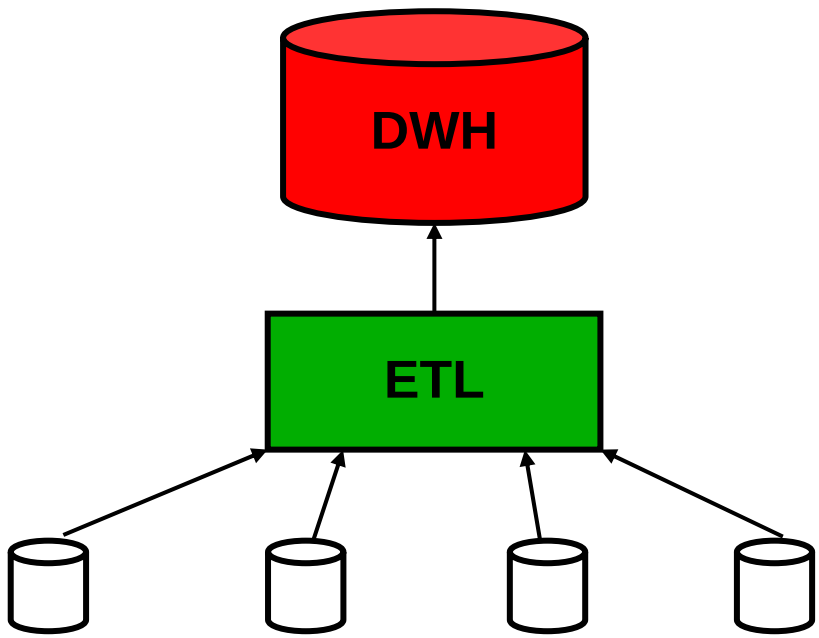


- **anfrageorientiert**
- **einheitliche Darstellung**
- **unveränderbar**
- **zeitraumbezogen**

Datawarehouselösungen:

Datenbereitstellung für das DWH

ETL: Extract Transform Load



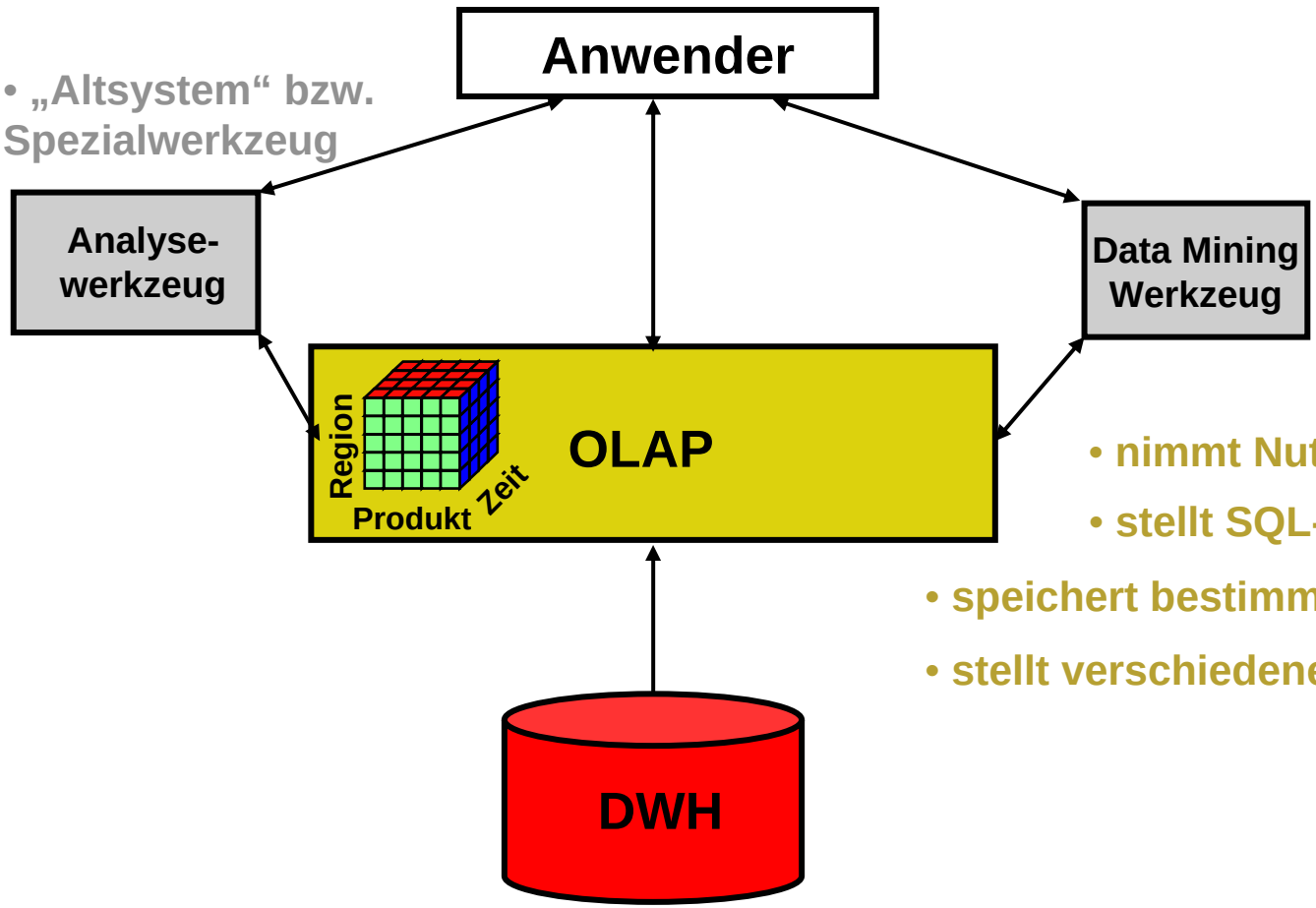
ETL enthält folgende Funktionen:

- **auswählen**
- **kopieren**
- **anreichern**
- **mit Zeitstempel versehen**
- **periodisch aktualisieren**

Datawarehouselösungen:

Datenanalyse

OLAP: Online Analytical Processing



- „Altsystem“ bzw. Spezialwerkzeug
- sucht automatisch neue Zusammenhänge
- nimmt Nutzeranfragen entgegen
- stellt SQL-Anfragen an DWH
- speichert bestimmte Fragen und Antworten
- stellt verschiedene Sichten mit Datenwürfel her

Datawarehouselösungen:

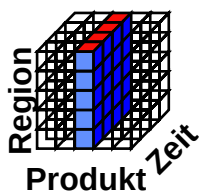
Datenanalyse

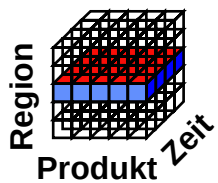
OLAP: Online Analytical Processing

Wie benutzt man den Würfel ?

- **slice**

  „herausschneiden“ der interessierenden Dimension





Datawarehouselösungen:

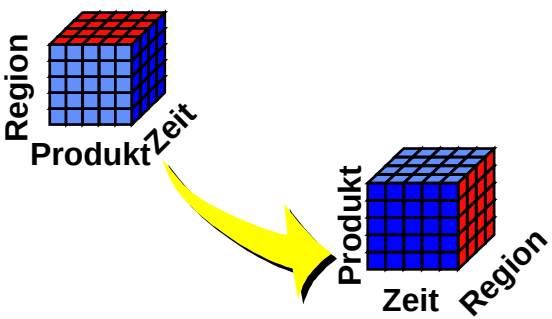
Datenanalyse

OLAP: Online Analytical Processing

Wie benutzt man den Würfel ?

- **dice**

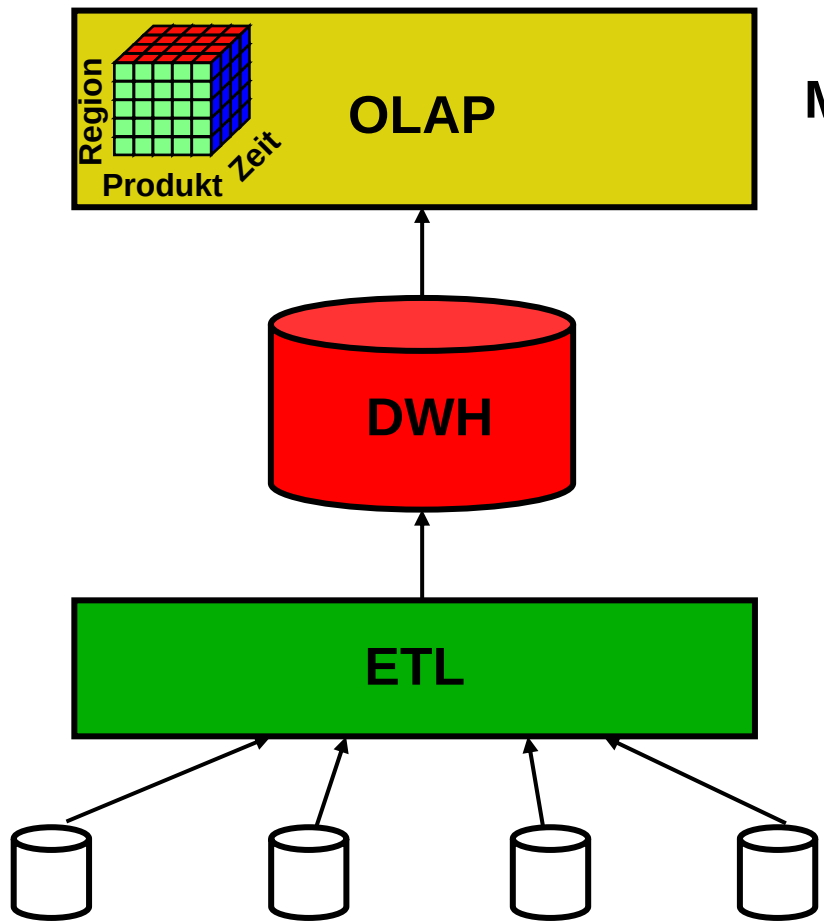
konfigurieren der interessierenden Datenzusammenhänge
(zerlegen des großen Würfels in kleine)



Fokussieren auf andere Sichtweise
(drehen des Würfels)

Das DWH als Bindeglied zwischen OLAP und ETL

Wie sollten die Daten in einem DWH organisiert sein ?



Mehrdimensionales Datenmodell

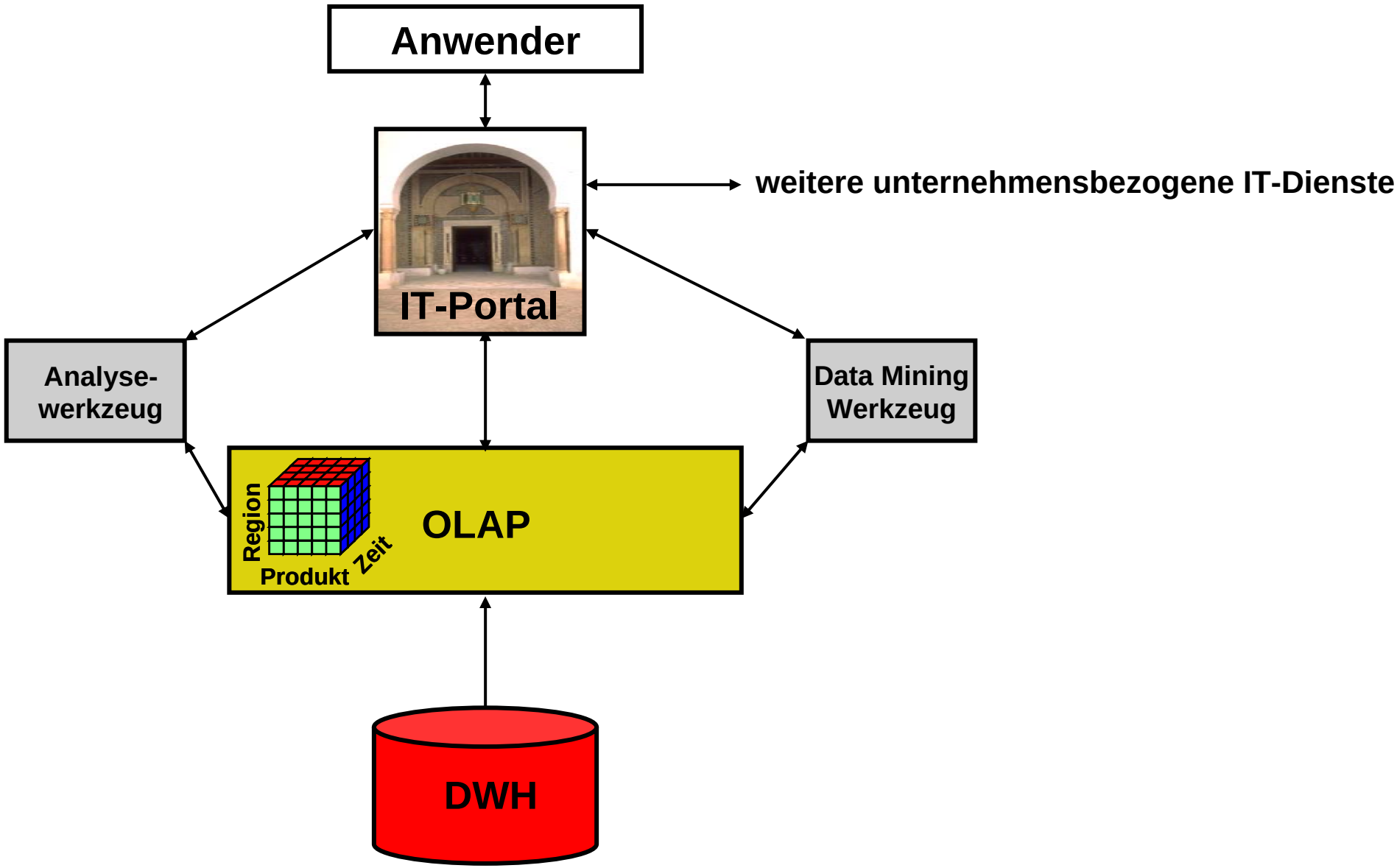
MOLAP

Data Warehouse: Datenbank

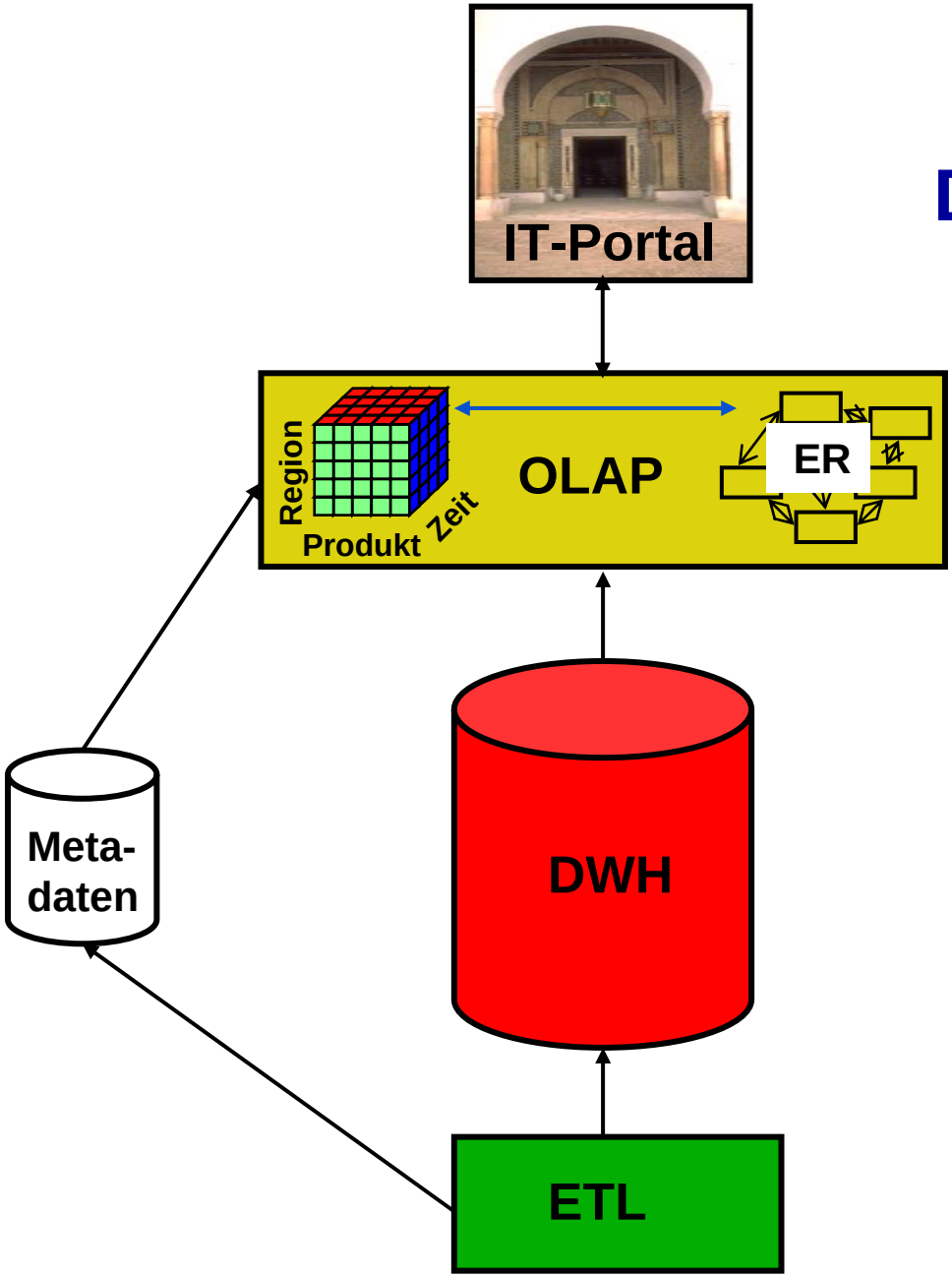
ROLAP

Relationale Datenmodelle

Anbindung an ein Portal

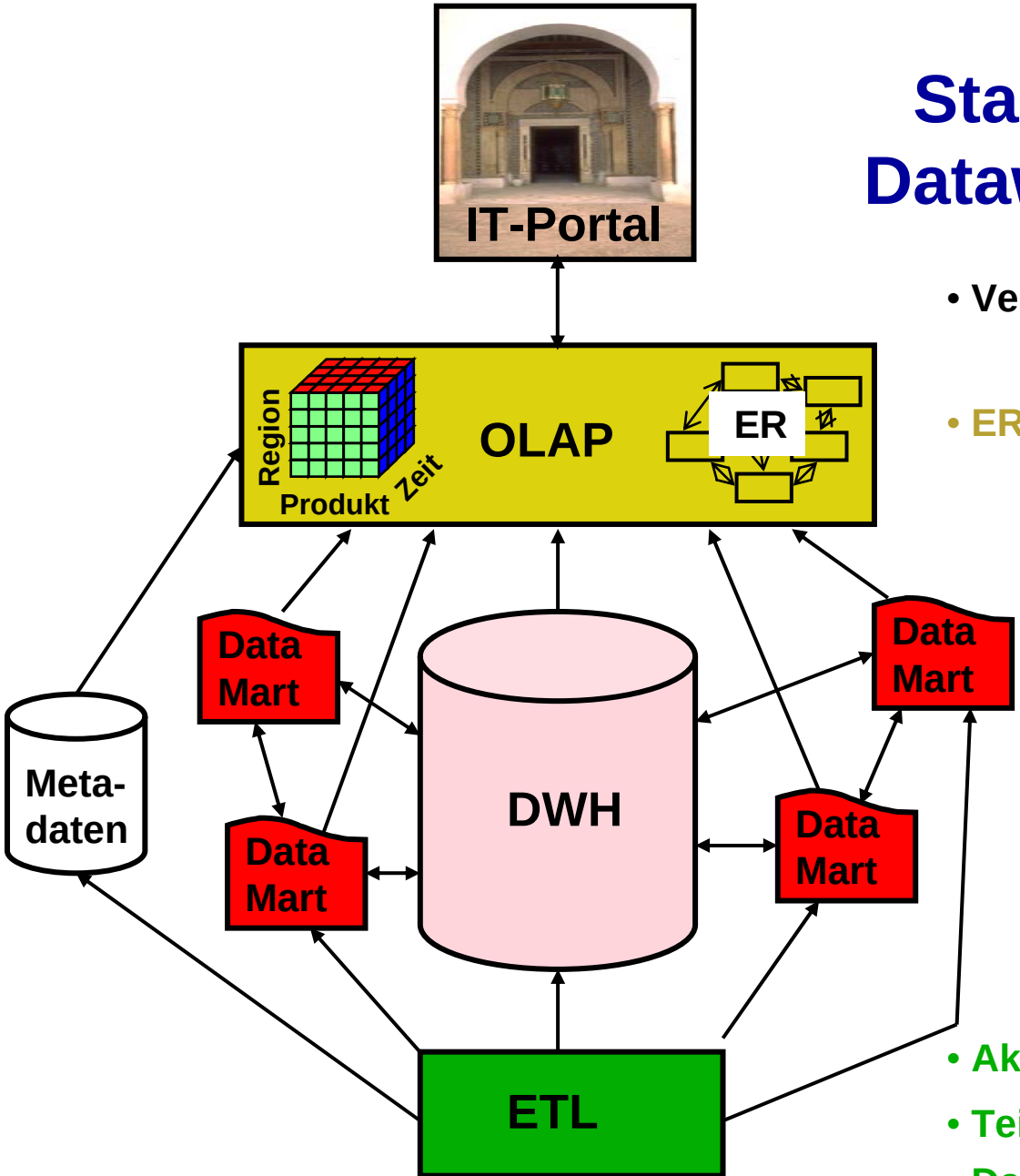


Stand der Technik bei Datawarehouselösungen



- Verwendung benutzerspezifischer Profile
- ER-Modellierung der Nutzerfragen
- ca. 50 Terabyte Speicher
- Aktualisierung max. 1 x pro Tag
- Teilaktualisierungen
- Datenaufbereitung

Stand der Technik bei Datawarehouselösungen



- Verwendung benutzerspezifischer Profile
- ER-Modellierung der Nutzerfragen

Verteiltes Datawarehouse

- Aktualisierung max. 1 x pro Tag
- Teilaktualisierungen
- Datenaufbereitung

Vorteile einer Datawarehouselösung

- **Analysen sind nachvollziehbar und wiederholbar**

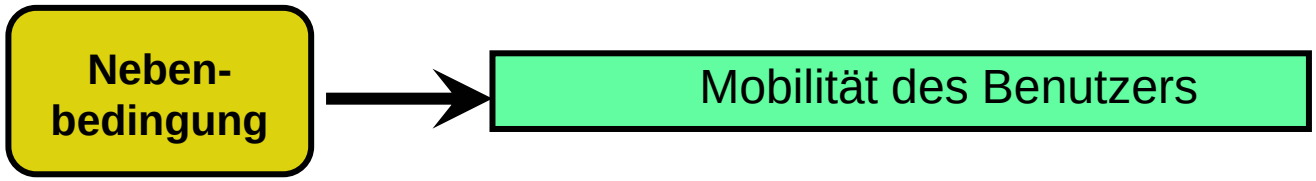
- **Analysen belasten nicht das operationale Geschäft**

Aufbau eines Datawarehouses ➔ **Offenlegung und Strukturierung der Prozesse im Unternehmen**

Grenzen einer Datawarehouselösung

- nicht geeignet für dynamische Daten
- nicht geeignet für dynamische Produkte
- schwierig für dynamische Unternehmen
- Offenlegung der operationalen Daten erforderlich

Verallgemeinerung der Problemstellung



Nutzerkreis:

Mobile Vertriebsmitarbeiter

Mobile Controller

Aktienmakler

Wartungstechniker im Feldeinsatz

? ? ?

Ein Einblick in die Praxis
<u>heute</u> um 17 Uhr, HS 5:

Internationales DataWarehouse
auf Basis eines Java Frameworks:
Ein Projektüberblick

Stefan Weyrauch (iXCASE)
Roy Rademacher (iXCASE)
Axel Mannhardt (Diplomand FH Wedel)