

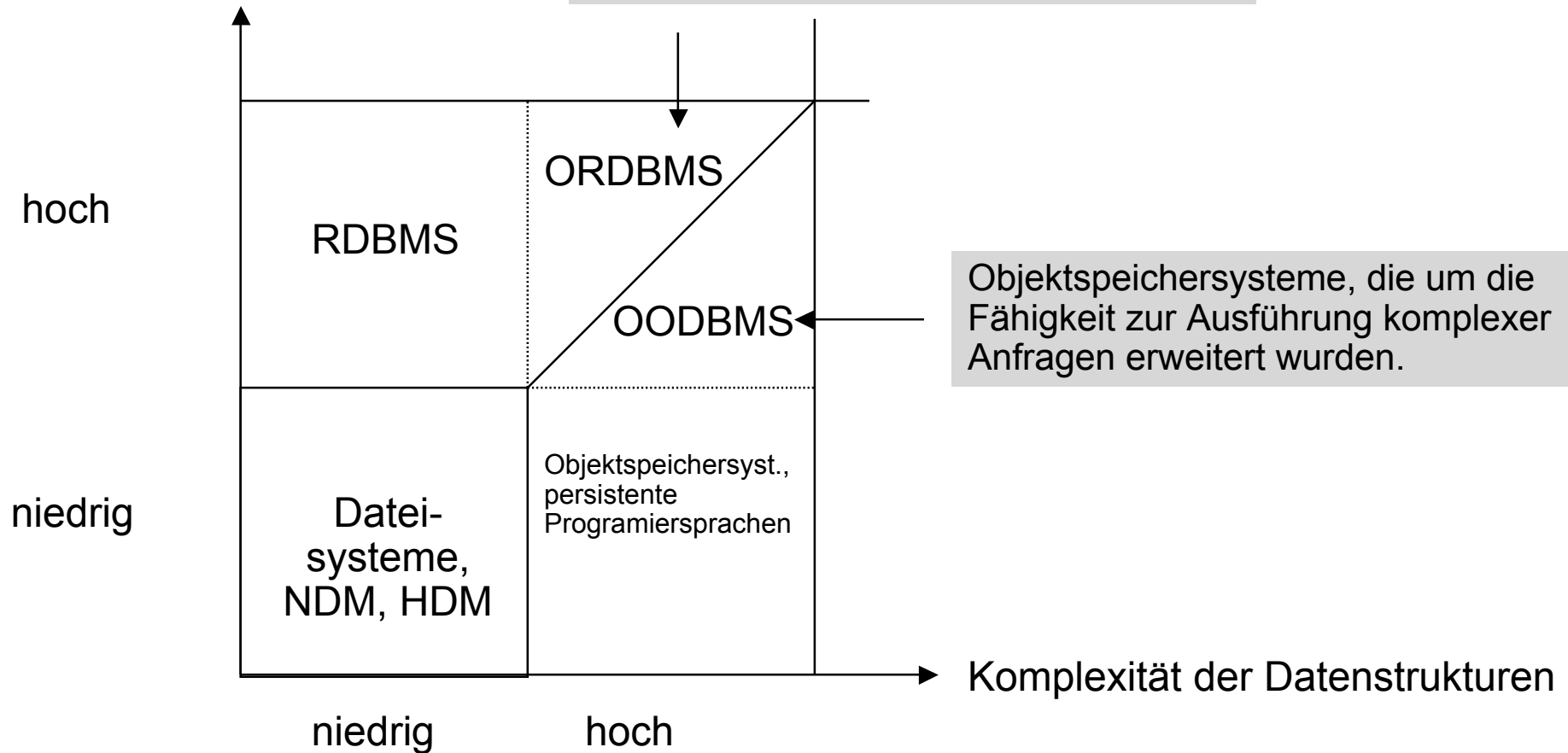
# Kapitel 6: Objektorientierte DB-Modelle & ODMG

	Relationales Datenmodell (RDM)	Objekt-orientierte Datenmodelle (OODM)	Objekt-relationale Datenmodelle	Semi-strukturierte Datenmodelle
Überblick über die Konzepte	<b>3.1</b>		<b>6.1</b>	<b>7.1</b> <b>7.2</b>
Darstellung von Assoziationen				
Datendefinition				
Anfragen				
Aktualisierungsoperationen				
Spezifika	<b>3.2 SQL</b>	<b>5.1</b>		

# Klassifikation bekannter Datenbanksysteme

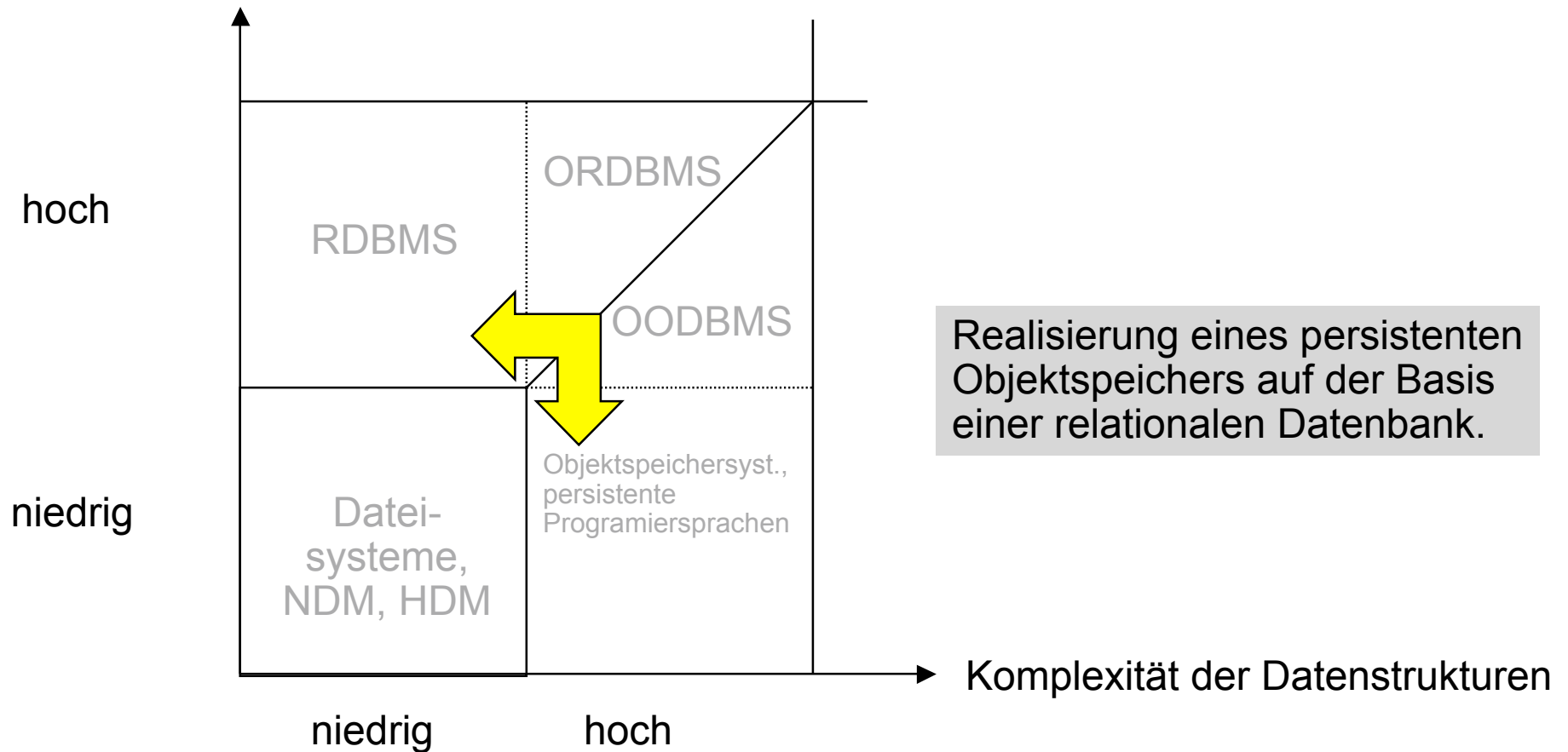
Auswertungskomplexität  
der Zugriffe/Anfragen

Ehemals relationale Datenbanken, die  
um Konzepte aus der Objektorientierung  
erweitert wurden.



# Rolle objektrelationaler Middleware

Auswertungskomplexität  
der Zugriffe/Anfragen



# Objektrelationale Middleware - Einordnung

---

## ❑ Quadrant 4-Systeme

- Ein ORDBMS ist ein um objektorientierte Funktionalität erweitertes RDBMS. Das RDM wird nicht durch ein anderes Modell ersetzt, sondern lediglich ergänzt, um die zusätzlichen Anforderungen von Anwendungen im vierten Quadranten zu erfüllen.
- Ein OODBMS implementiert mit dem OODM ein Datenmodell, das bereits vollständig die Anforderungen für Quadrant-4-Anwendungen erfüllt.

## ❑ Objektrelationale Middleware

- Führt kein neues Datenmodell ein und
- erweitert kein vorhandenes Datenmodell,
- sondern dient als Vermittler zwischen einer rein objektorientierten Anwendung und einem rein relationalen Datenbanksystem.

# Objektrelationale Middleware

---

Bibliotheken zur Nutzung von Diensten relationaler Datenbanken innerhalb von objektorientierten Sprachen (z.B. Java).

## Eigenschaften

- ❑ Einfach strukturierte Daten können in relationale DB ausgelagert werden und von deren Fähigkeiten (Indizierung, Konsistenzsicherung, hohe Verfügbarkeit, Mehrbenutzerbetrieb) profitieren.
- ❑ Komplex strukturierte Daten werden (soweit möglich) transparent in einfacher strukturierte, dem RDBMS zugängliche Strukturen zerlegt.
- ❑ Der Benutzer „sieht“ ausschließlich Strukturen der objektorientierten Programmiersprache, die Umsetzung von OO-Operationen auf relationale Operationen erfolgt transparent.
- ❑ Historisch gewachsene und in relationalen Datenbanken gespeicherte Daten können mit den Mitteln des objektorientierten Systems weiterverwendet werden.
- ❑ Dienstleistungen, die bereits von der relationalen Datenbank erbracht werden, brauchen in der objektorientierten Anwendung nicht erneut realisiert zu werden.

# Forward Engineering - Objects first

---

Die objektorientierte Anwendung verwendet die relationale Datenbank als persistenten Objektspeicher entweder für all ihre Objekte oder eine Menge besonders gekennzeichneteter Objekte.

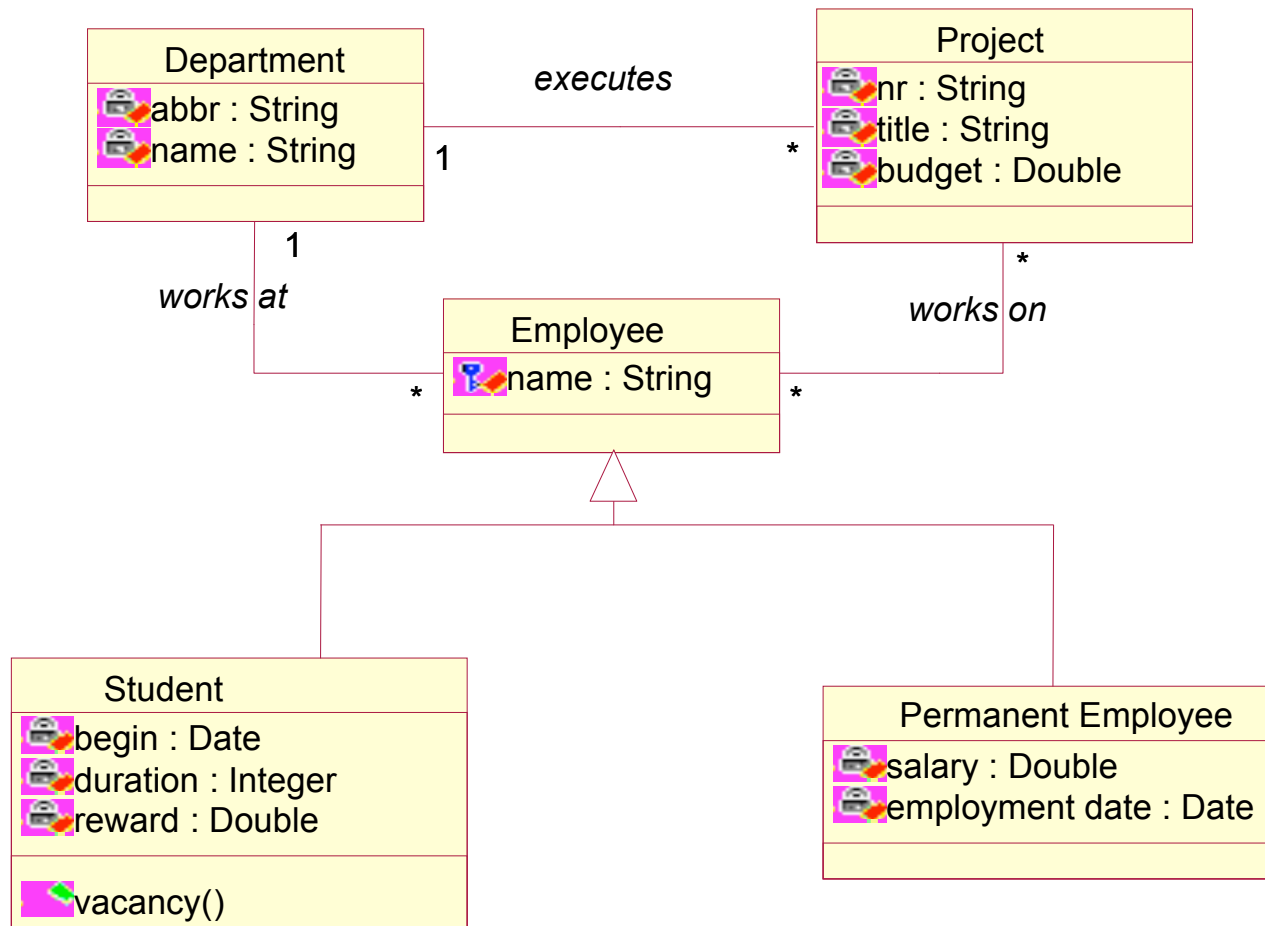
## **Vorteil**

Der Speicher für alle persistenten Daten eines Unternehmens liegt in Form einer einzigen, zentral administrierbaren, relationalen Datenbank vor. Für objektorientierte Daten fallen keine zusätzlichen Verwaltungsaufgaben (z.B. Datensicherung) an.

## **Nachteile**

- ❑ Speicherung der Objekte erfolgt oft in Form von BLOBs, wodurch Stärken des RDBMS, wie schneller Schlüsselzugriff oder komplexe Anfragen nur unzureichend genutzt werden.
- ❑ Strukturiert gespeicherte Objekte werden oft nur von der Anwendung genutzt, für die sie entwickelt wurden. Überlegungen zur durch das RDBMS ermöglichten gemeinsamen Nutzung von Daten durch andere Anwendungen, werden oft vernachlässigt.

# OO-DB Mapping: The Issue (1)

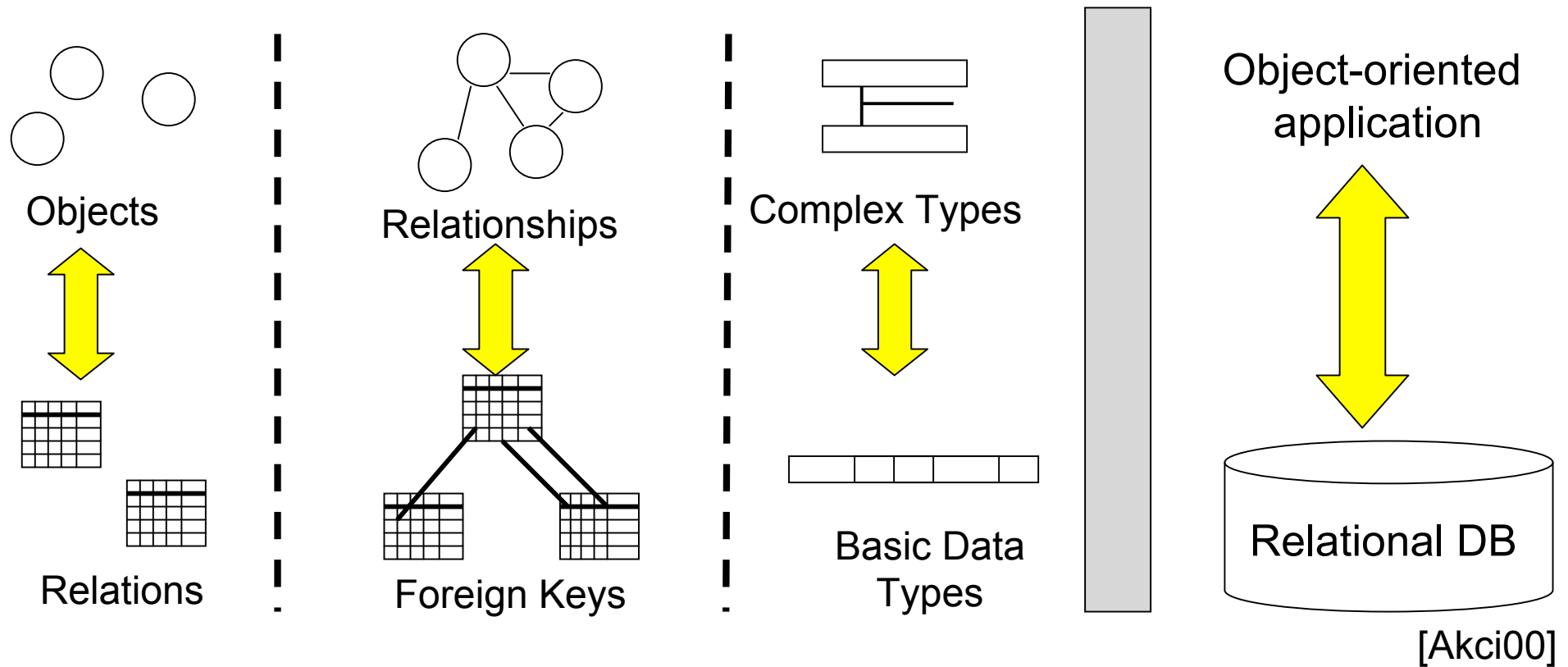


[Akci00]

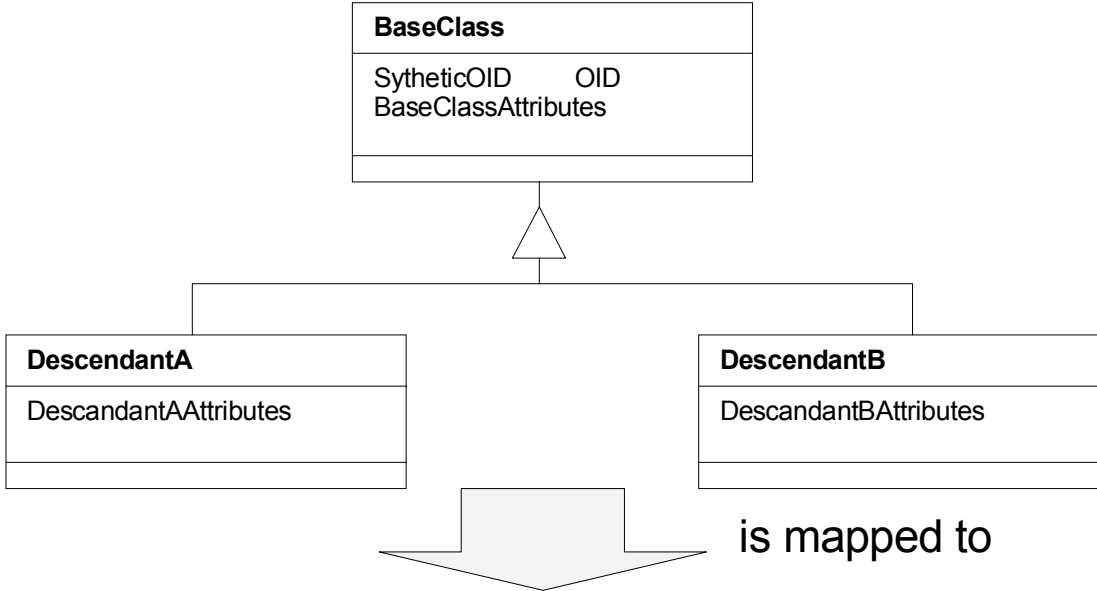
# OO-DB Mapping: The Issue (2)

Problem: Mapping of Objects to Relational Databases

- ❑ Use of *Mapping-Tools* (Middleware)
- ❑ Diverse Mappings



# Mapping Subclass Hierarchies: Option A



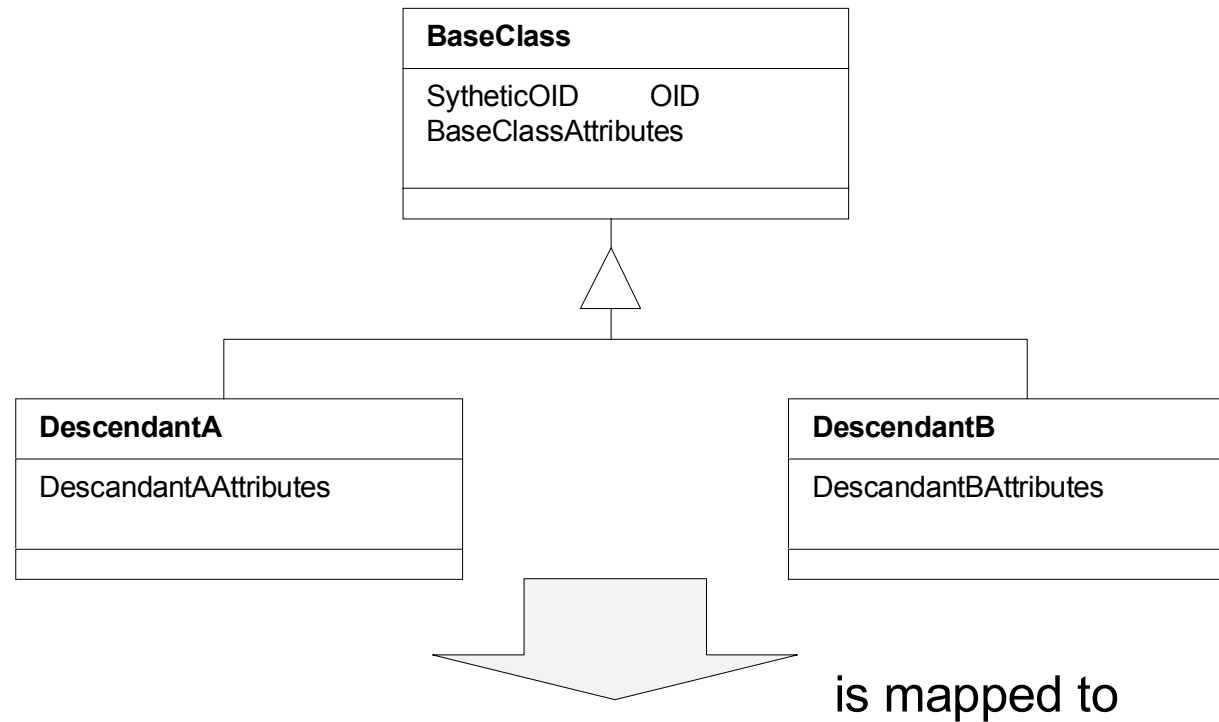
is mapped to

Table for BaseClass, DescendantA, DescendantB			
	SytheticOID, BaseClassAttributes	DescendantAAttributes	DescendantBAttributes
<b>BaseClassInstance</b>	Attribute Values	Null Values	Null Values
<b>DescendantA Instance</b>	Attribute Values	Attribute Values	Null Values
<b>DescendantB Instance</b>	Attribute Values	Null Values	Attribute Values

[Akci00]

# Mapping Subclass Hierarchies: Option B

---



DescendantATable	
SytheticOID	OID
DescendantAAttributes	
.....	

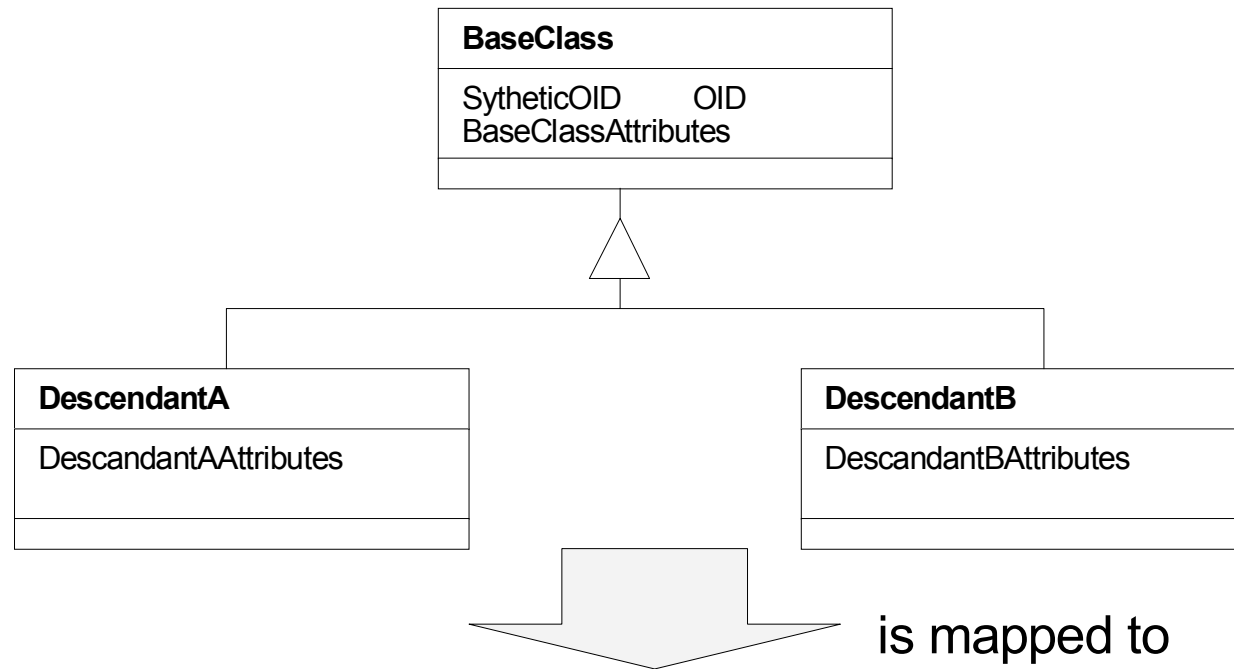
BaseClassTable	
SytheticOID	OID
BaseClassAttributes	
.....	

DescendantBTable	
SytheticOID	OID
DescendantBAttributes	
.....	

[Akci00]

# Mapping Subclass Hierarchies: Option C

---



DescandantATable
SytheticOID    OID
BaseClassAttributes
DescandantAAttributes
.....

BaseClassTable
SytheticOID    OID
BaseClassAttributes
.....

DescandantBTable
SytheticOID    OID
BaseClassAttributes
DescandantBAttributes
.....

[Akci00]

# Forward Engineering Prozeß

---

- ❑ Objektorientierter Entwurf der Anwendungsklassen mit Hilfe eines Modellierungswerkzeugs.
- ❑ Eingabe (des relevanten Teils) der Entwurfsdaten in einen Klassengenerator, der dann
  - zu den gegebenen Klassen entsprechende Code-Fragmente (Klassenrümpfe) in der Zielprogrammiersprache (z.B. C++, Smalltalk, Java) erzeugt und
  - SQL-Befehle zur Erzeugung von Datenstrukturen zur persistenten Speicherung der Anwendungsdaten generiert.
- ❑ Programmieren der Anwendungslogik innerhalb der erzeugten Klassenstrukturen.
- ❑ Verbindung des Programms mit einem Laufzeitsystem, das Zugriffe auf persistente Objekte transparent in Zugriffe auf die relationale Datenbank umsetzt.

# Reverse Engineering - Relations first

---

Das objektorientierte System kann vorhandene relationale Daten und Hilfsinformationen nutzen und in objektorientierte Strukturen übersetzen.

## Vorteile

- ❑ Vorhandene relationale Daten können weiterbenutzt werden.
- ❑ Daten können über das zentrale relationale Datenbanksystem mit anderen Anwendungsprogrammen geteilt werden.
- ❑ (Teil-)Anfragen und zugehörige Optimierungen können an das relationale Datenbanksystem delegiert werden
- ❑ Beziehungen zwischen Relationen über Fremdschlüssel können in Objektreferenzen übersetzt werden.

## Nachteile

- ❑ Die Komplexität der in der objektorientierten Sprache nutzbaren Datenstrukturen wird durch das relationale Datenmodell begrenzt.
- ❑ Effizienz der Nutzung vorhandener relationaler Daten hängt von der Qualität der verfügbaren Metainformationen über ein Datenbankschema ab.

# Reverse Engineering Prozeß

---

## Ausgangssituation

- Die Anwendungsprogrammiersprache ist objektorientiert.
- Es existieren bereits Anwendungsdaten, die u.U. auch von anderen, nicht-objektorientierten Anwendungsprogrammen genutzt werden.
- Die Daten sind in einer relationalen Datenbank gespeichert.

## Prozeß

- Automatische oder halbautomatische Analyse des vorhandenen Datenbankschemas.
- Bereitstellung von Standardschnittstellen für den Datenbankzugriff oder Erzeugung von an das Datenbankschema angepaßten Klassen.
- Entwurf und Implementierung der Anwendungslogik unter Berücksichtigung der verfügbaren Programmierschnittstellen bzw. der in den erzeugten Klassen verfügbaren Methoden.
- Verbindung des Programms mit einem Laufzeitsystem, das Zugriffe auf persistente Objekte transparent in Zugriffe auf die relationale Datenbank umsetzt.

# Abbildung des Datenbankschemas

---

Soll ein Anwendungsprogramm ein existierendes Datenbankschema weiterbenutzen, sollte der Anwendungsentwickler es ohne tiefgreifende Kenntnisse des relationalen Modells benutzen können.

Übersetzung der Konzepte des relationalen Modells in analoge objektorientierte, z.B.

- ❑ Abbildung der Attribute einer Tabelle auf Objektattribute.
- ❑ Abbildung von Datenbankoperationen, die die Änderung einzelner Datensätze betreffen (Einfügen, Ändern, Löschen) auf Methoden der zu den Datensätzen korrespondierenden Objekte (*new*, Zuweisung, *delete*)
- ❑ Abbildung von Fremdschlüsselbeziehungen auf Objektreferenzen

relationale Datenbanken	Objektorientierte Programmierumgebung
Datensatz	Objekt
Tabelle	Klasse
Attribut	Objektattribut
Primärschlüssel	Objektidentität
Fremdschlüssel	Objektreferenz

# Unterstützung des Abbildungsprozesses

---

- ❑ Die meisten Datenbanken speichern Informationen über Tabellenstrukturen, verwendete Typen, Primär- und Fremdschlüssel in ihrem *Data Dictionary*.
- ❑ Spezielle Entwicklungswerkzeuge können diese Metadaten lesen und daraus automatisch korrespondierende objektorientierte Strukturen generieren.
- ❑ Häufig muß der Anwendungsentwickler dem Entwicklungswerkzeug fehlende Metainformation nachträglich mitteilen (inverse Beziehungen, Kardinalitätsbeschränkungen, Zugriffsmuster, ...).
- ❑ Aus den gegebenen Metadaten erzeugen diese Entwicklungswerkzeuge dann Klassen, mit deren Hilfe die relationalen Daten manipuliert werden können.

# Standardschnittstellen zum DBMS

---

- ❑ Anwendungsprogramme können mit jeder relationalen Datenbank zumindest über die Sprache SQL kommunizieren („*intergalactic dataspeak*“).
- ❑ Unterschiedliche Softwarehersteller bieten Bibliotheken an, mit deren Hilfe SQL-Befehle an die Datenbank gesendet und die Ergebnisse übertragen und untersucht werden können:
  - Z.B.: JDBC (Java-SQL-Schnittstelle)
- ❑ Klassengeneratoren erzeugen Code, der auf solchen standardisierten Schnittstellen aufsetzt.
- ❑ Damit können Programme, die auf einer Standardschnittstelle aufsetzen, mit jedem DBMS, für das entsprechende Treiberprogramme existieren, ausgeführt werden.

# Leistungen der Middleware (1)

---

## ❑ Umsetzung von Objektmanipulationen in Datenbankmanipulationen

- Änderung von Attributwerten - SQL-update-Befehl
- Erzeugung neuer Objekte - SQL-insert-Befehl
- Lesen vorhandener Daten - SQL-select
- Navigation mit Hilfe von Objektreferenzen
  - direkt im (flüchtigen) Objektspeicher der Programmiersprache
  - ggf. durch Formulierung einer geeigneten SQL-Anfrage (Join), wenn die Dereferenzierung zum ersten Mal erfolgt

## ❑ Konvertierung von atomaren Werten in eine korrespondierende objektorientierte Repräsentation

## ❑ Sicherung der Objektidentität

- Sicherung der Primärschlüsselintegrität auf Objektebene
- jedes Objekt korrespondiert mit genau einem Datensatz in der Datenbank

# Leistungen der Middleware (2)

---

- ❑ Beschleunigung der Zugriffe auf eine relationale Datenbank durch einen objektorientierten Cache:

Daten werden einmal aus der Datenbank gelesen und erst dann durch eine (vergleichsweise teure) SQL-Operation in der Datenbank geändert, wenn

- ein Anwendungsprogramm bzw. seine Änderungstransaktion beendet wird,
- kein Anwendungsprogramm die Daten mehr benutzt und der Datensatz aus dem objektorientierten Cache entfernt wird (vgl. *Garbage Collection*) oder
- das Anwendungsprogramm dies ausdrücklich verlangt.

- ❑ Verbergen der Komplexität, die durch die Mehrbenutzerfähigkeit der Datenbank und/oder des Cache entsteht:

- Isolation gegen parallele Änderungsoperationen anderer Datenbanktransaktionen.
- Behandlung von Konflikten, falls mehrere Anwendungsprogramme dieselben Objekte aus einem gemeinsamen Objektcache benutzen wollen.

# Isolation durch Sperren

---

## Vorteile

- ❑ Einmal aus der Datenbank gelesene Objekte können nur durch den Prozeß verändert werden, der sie auch gelesen hat.
- ❑ Änderungen an Objekten brauchen nur einmal am Ende einer Transaktion gesammelt in die Datenbank zurückgeschrieben werden.

## Nachteile

- ❑ Mit jedem neuen Datenbankzugriff (Anfragen) werden neue Datensätze gesperrt, zusätzlich u.U. alle Datensätze, die von ihnen abhängen.
- ❑ Folge: Andere Datenbankbenutzer werden mit der Zeit buchstäblich „ausgesperrt“.
- ❑ Am Ende einer Datenbanktransaktion werden grundsätzlich alle Sperren freigegeben. Findet danach noch ein Zugriff auf Objekte statt, müssen die Sperren neu angefordert werden (zusätzliche DB-Kommunikation).

"Pessimistische Sperren"

# Isolation durch Zeitstempel oder Versionen

---

Jedes Datum ist mit einer Versionsnummer oder einem Zeitstempel versehen. Die Middleware liest Daten ohne Sperren anzufordern. Änderungen erhöhen die Versionsnummer oder ändern den Zeitstempel. Beim Zurückschreiben der Daten wird die Versionsnummer im Cache mit der Versionsnummer in der DB verglichen. Bei Konflikten wird die Transaktion zurückgesetzt.

## Vorteile

- ❑ Daten können von vielen Datenbankbenutzern gleichzeitig verwendet werden.
- ❑ Wartezeiten auf die Freigabe gesperrter Datensätze entfallen.

## Nachteile

- ❑ Änderungen durch andere Datenbankbenutzer werden u.U. nicht oder zu spät bemerkt. Beispiel: Cache liefert noch Objekte, die in der Datenbank bereits gelöscht wurden.
- ❑ Transaktionen müssen evtl. zurückgesetzt werden, weil eine parallel laufende Transaktion dieselben Daten verändert hat und erfolgreich beendet wurde.

"Optimistische Sperren"

# Kommerzielle Lösungen zur Isolation

---

- ❑ Das Sperren von Daten liegt in der Verantwortung des Anwendungsprogrammierers - typisch für SQL-Programmierschnittstellen, wie JDBC oder DBTools.h++
- ❑ Ein in die Anwendung integrierter Objektmanager
  - sperrt nur pessimistisch, sperrt nur optimistisch oder kann vom Anwendungsprogrammierer seinen Anforderungen entsprechend konfiguriert werden (z.B. ONTOS);
  - ist Teil der Anwendung, d.h. jede Anwendung benutzt einen eigenen Objektmanager und erscheint für die Datenbank als eigener Client.
- ❑ Ein zentraler Objektmanager für  $n$  verschiedene Anwendungen
  - tritt gegenüber der relationalen Datenbank als einzige Client-Anwendung auf und implementiert u.U. ein von der Datenbank völlig unabhängiges Sperrprotokoll (Beispiel: Persistence);
  - beliefert als sogenannter „Application Server“ („Middle Tier“)  $n$  Anwendungsprogramme mit Daten.

# Bewertung: Zentrale Objektmanager

---

## Vorteile

- ❑ Eine Menge von Objekten, die mit Inhalten einer relationalen Datenbank korrespondieren, können vielen Anwendungen gleichzeitig zur Verfügung gestellt werden (auch sprachunabhängig als CORBA-Dienst).
- ❑ Die Einzelanwendungen brauchen kein eigenes Cache- oder Sperrprotokoll zu implementieren.
- ❑ Objektnavigation kann direkt im Objektmanager erfolgen.
- ❑ Objektmanager kann alternative Sperrprotokolle implementieren, z.B. durch Vergabe von Versionsnummern an Objekte.

## Nachteile

- ❑ Alternative/leistungsfähigere Zugriffsmethoden können nur sinnvoll genutzt werden, wenn *alle* Anwendungen über *denselben* Objektmanager mit der Datenbank kommunizieren.
- ❑ Wie reagiert der Objektmanager auf Änderungen, die durch *direkte* Datenbankzugriffe erfolgen (z.B. durch ältere Datenbankanwendungen)?