Kapitel 1

Motivation

Inhalt

- Komponentenorientierte AWS-Architekturen
- *Universal Storage* vs. *Universal Access*
- *DB-Middleware*
- Zusammenfassung

Moderne Anwendungssysteme (1)

- *Datenbank- und Transaktionssysteme*
  - bisher waren sie das Kernstück von immer komplexer werdenden Anwendungssystemen (AWS) oder auch Informationssystemen (IS)
  - bisher waren AWS- oder IS-Architekturen überwiegend **DBS-zentriert**
  - ihr Transaktionskonzept bezog sich nur auf die DB (ACID wird nur für DB-Daten zugesichert)
  - der Regelfall bei der Datenverwaltung waren homogene DBS, und zwar zentralisiert oder verteilt (MRDBS)
- Probleme
  - Integration bestehender, heterogener Datenquellen innerhalb eines neu zu entwickelnden AWS
  - mangelnde Flexibilität bei Einbindung/Übernahme von neuen oder unverträglichen Daten(typen) in bestehendes komplexes AWS

Moderne Anwendungssysteme (2)

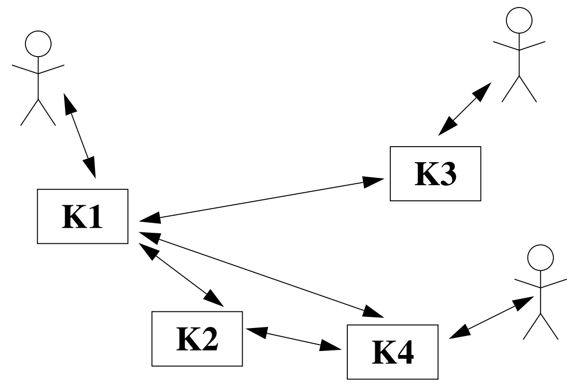
- Neue Anforderungen an *Anwendungs- und Informationssysteme*:
  - vereinheitlichte/homogenisierte Datenhaltung (gleichförmiges API)
  - Kooperation in heterogenen Systemumgebungen mit vielfältigen (und ggf. autonomen) Anwendungs- und Komponentensystemen
  - DBS-Zentrierung wird schrittweise aufgelöst (DBS sind nur (wichtige) Komponenten unter mehreren)
  - der Bereich der transaktionsorientierten Verarbeitung ist auf komplexer strukturierte SoCs (*Sphere of Control*) auszudehnen (z. B. unter Kontrolle von Transaktionssystemen)

Moderne Anwendungssysteme (3)

- Charakteristika
  - komponentenbasiert
  - Wiederverwendung bei der Entwicklung
  - Interoperabilität in heterogenen Umgebungen
  - Einbindung bestehender Anwendungssysteme und Datenquellen (Legacy) ohne deren Änderung
  - flexible Erweiterung um neue Komponenten
  - homogenisierte Sicht für den Benutzer und den Anwendungsprogrammierer
    - Funktionsintegration
    - Datenintegration
  - transaktionsgeschütze (verteilte) Abläufe
- Grobe Architekturmodelle
  - Interoperable Komponenten
  - Verteilte Transaktionsverwaltung
  - Funktionsintegration
  - Integration <u>in 1</u> ORDBS
  - Datenintegration
  - Kombinationen der vorgenannten

Interoperable Komponenten (1)

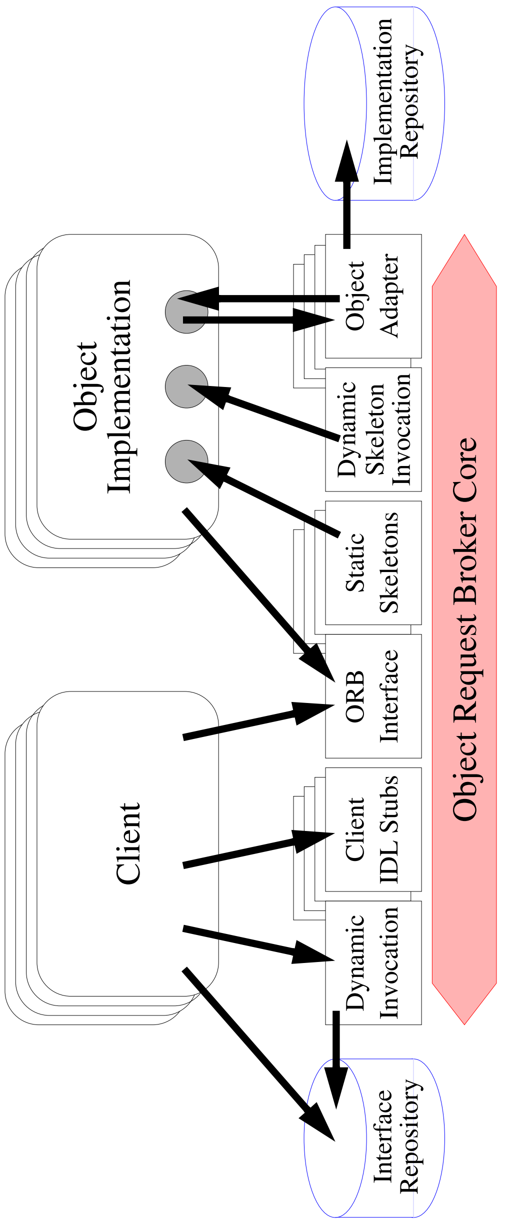
❑ Illustration



- Komponenten: Objekte, Anwendungssysteme, Datenquellen, Dateisysteme, ...
- erfordert:
  - Komponententechnolgie, z. B. CORBA, EJB, ...
  - Kommunikationstechnologie, z. B. Internet-Protokolle
  - Datenaustauschtechnologie, z. B. XML
- wesentlicher Vorteil:
  - Interoperabilität
- wesentlicher Nachteil:
  - keine homogenisierte Sicht

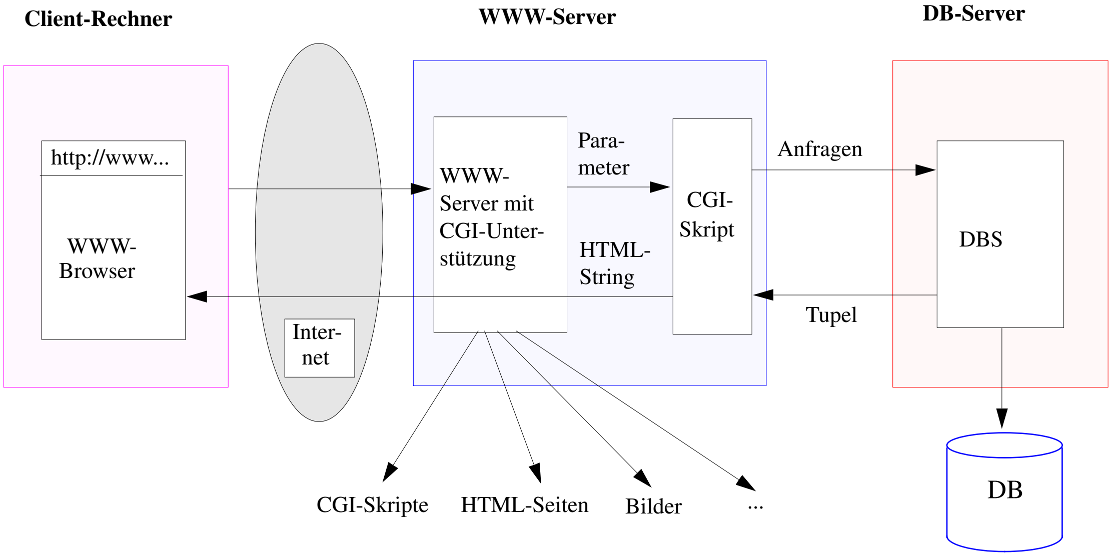
Interoperable Komponenten (2)

❑ Beispiel 1: CORBA



Interoperable Komponenten (3)

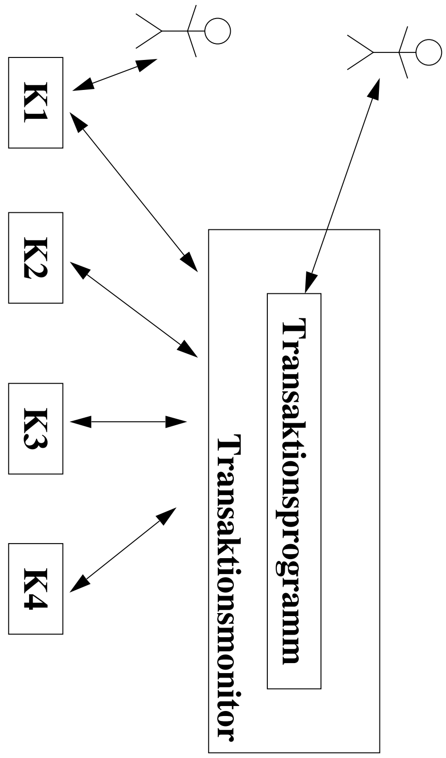
❑ Beispiel 2: DB-Zugriff übers Web (hier CGI-Lösung)



❑ vgl. Kapitel 7, 8, 9 und (teilw.) 3

Verteilte Transaktionsverwaltung (1)

❑ Illustration



⇨ Komponenten: DBS

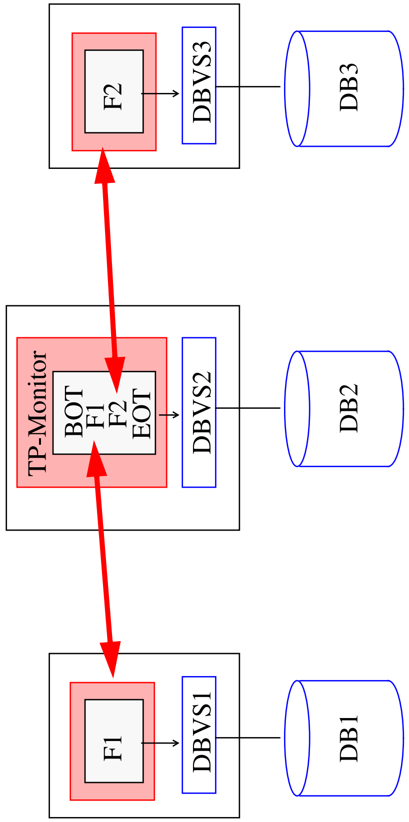
⇨ erfordert:
- TP-Monitor-Technologie

⇨ wesentlicher Vorteil:
- Transaktionsschutz für verteilte Abläufe

⇨ Nachteile:
- nur DBS
- niedriger Integrationsgrad

Verteilte Transaktionsverwaltung (2)

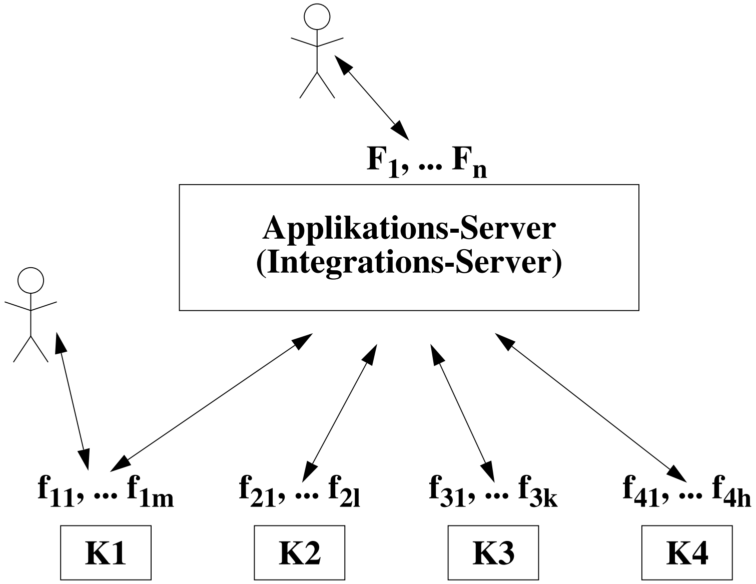
❑ Beispiel: Programmierte Verteilung



❑ vgl. Kapitel 2 und (teilw.) 3

Funktionsintegration (1)

❑ Illustration



➪ Komponenten: Objekte, Anwendungssysteme, Datenquellen (DBS), Dateisysteme, ...

➪ erfordert:

- Konnektoren
- Mechanismen zur Funktionsintegration

➪ wesentlicher Vorteil:

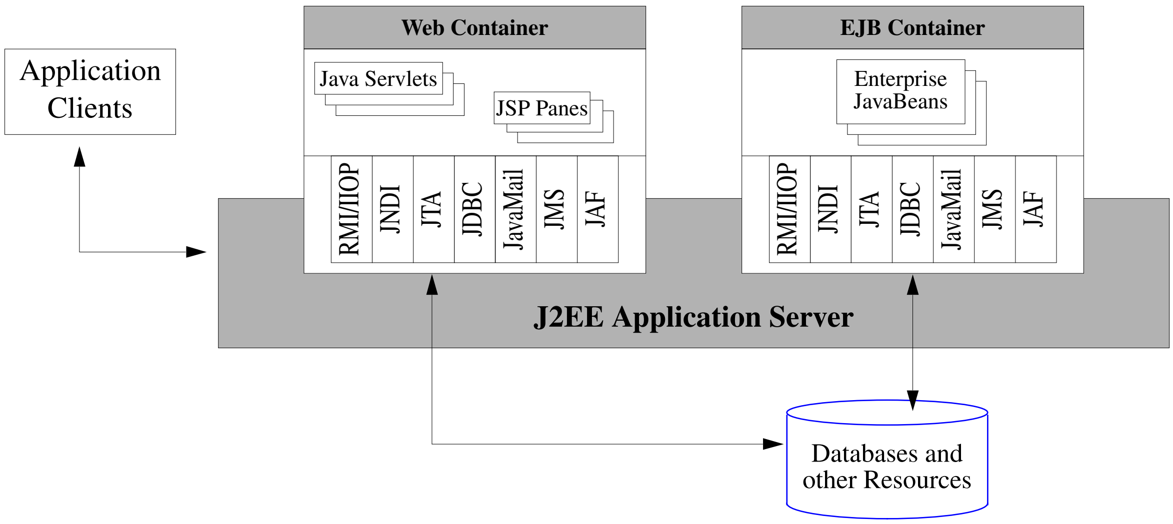
- homogenisierte Sicht bzgl. Funktionen

➪ wesentlicher Nachteil:

- Integration der Datenstrukturen schwierig

Funktionsintegration (2)

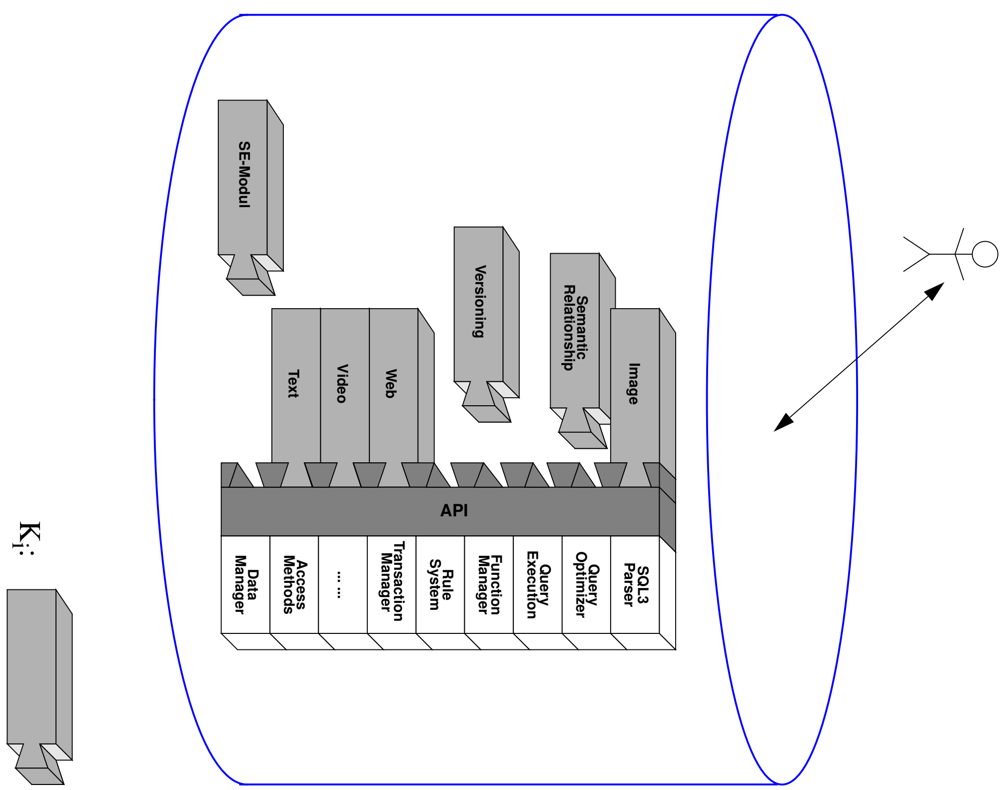
- ❑ Beispiel: J2EE



- ❑ vgl. Kapitel 10 und Teile von 4, 7, 9

Integration in 1 ORDBS (1)

- ❑ Illustration



Integration in 1 ORDBS (2)

❑ Eigenschaften

➪ Komponenten: DBS-Erweiterungsmodule

➪ erfordert:

- DBS-Erweiterungstechnologie (OR)

➪ wesentlicher Vorteil:

- homogenisierte Sicht bzgl. Daten und Funktionen

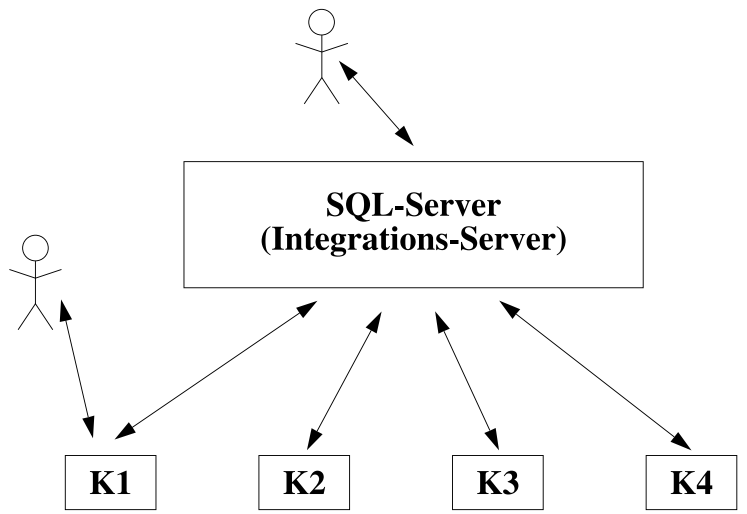
➪ Nachteile:

- Nutzung der Erweiterungstechnologie bzgl. funktionaler Aspekte noch nicht ausgereift
- Komponenten müssen (bzgl. Erweiterungsinfrastruktur) neu entwickelt werden

❑ vgl. Teile von Kapitel 9

Datenintegration (1)

❑ Illustration



➪ Komponenten: Objekte, Anwendungssysteme, Datenquellen (DBS), Dateisysteme, ...

➪ erfordert:

- Wrapper
- Mechanismen zur Datenintegration

➪ wesentlicher Vorteil:

- homogenisierte Sicht bzgl. Daten

➪ wesentlicher Nachteil:

- funktionale Integration schwierig

Datenintegration (2)

❑ Beispiel 1: DataJoiner

DB2/MVS

CUST

ACCT NO	NAME	. . .
101	Ajax	. . .
303	Super	. . .
099	Acme	. . .
.	.	.

Sybase

SJBR

ACCT NO	Balance	. . .
303	250,000	. . .
099	100,000	. . .
.	.	.
.	.	.

Oracle

SFBR

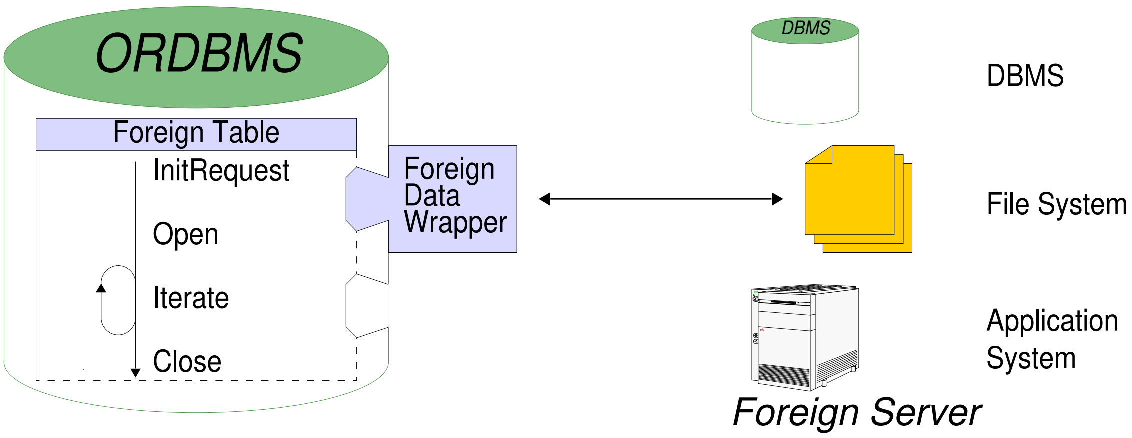
ACCT NO	Crllmlt	. . .
101	319,000	. . .
303	100,000	.
.	.	.

DataJoiner

```
Select  *
From    Cust, SJBR, SFBR
Where   Cust.Acct No = SJBR.Acct No
And     SJBR.Acct No = SFBR.Acct No
```


Datenintegration (3)

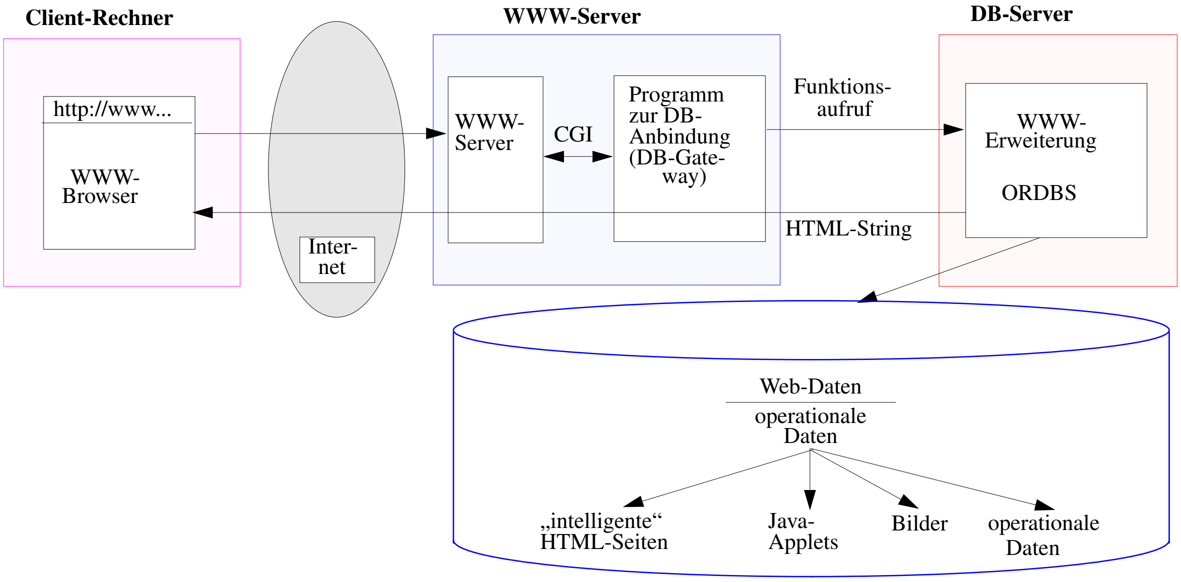
❑ Beispiel 2: ‘Foreign Data Wrapper’ in ‘SQL/MED’



❑ vgl. Kapitel 5, 6

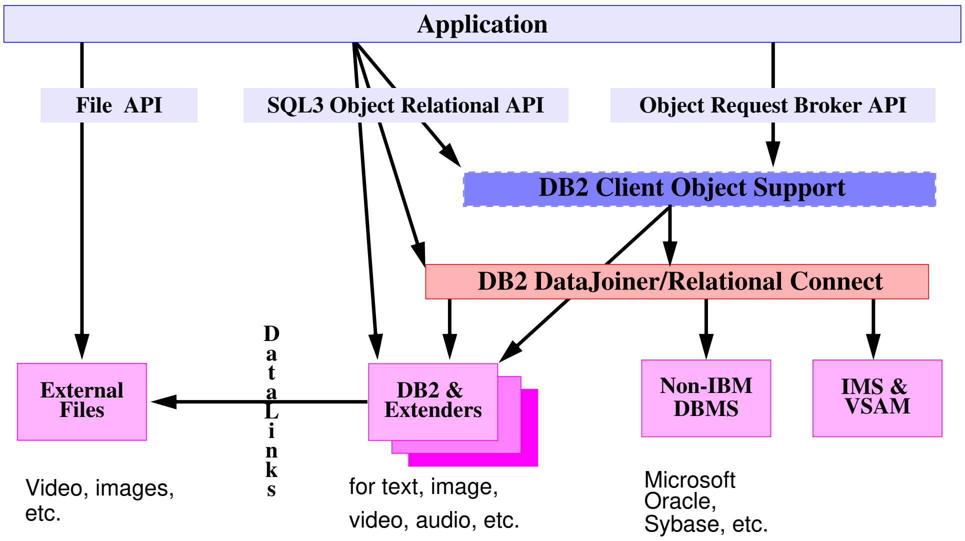
Kombinationen (1)

- In der Regel Kombinationen der vorgenannten Ansätze
- Beispiel 1: ‘Interoperable Komponenten’ und ‘1 ORDBS’



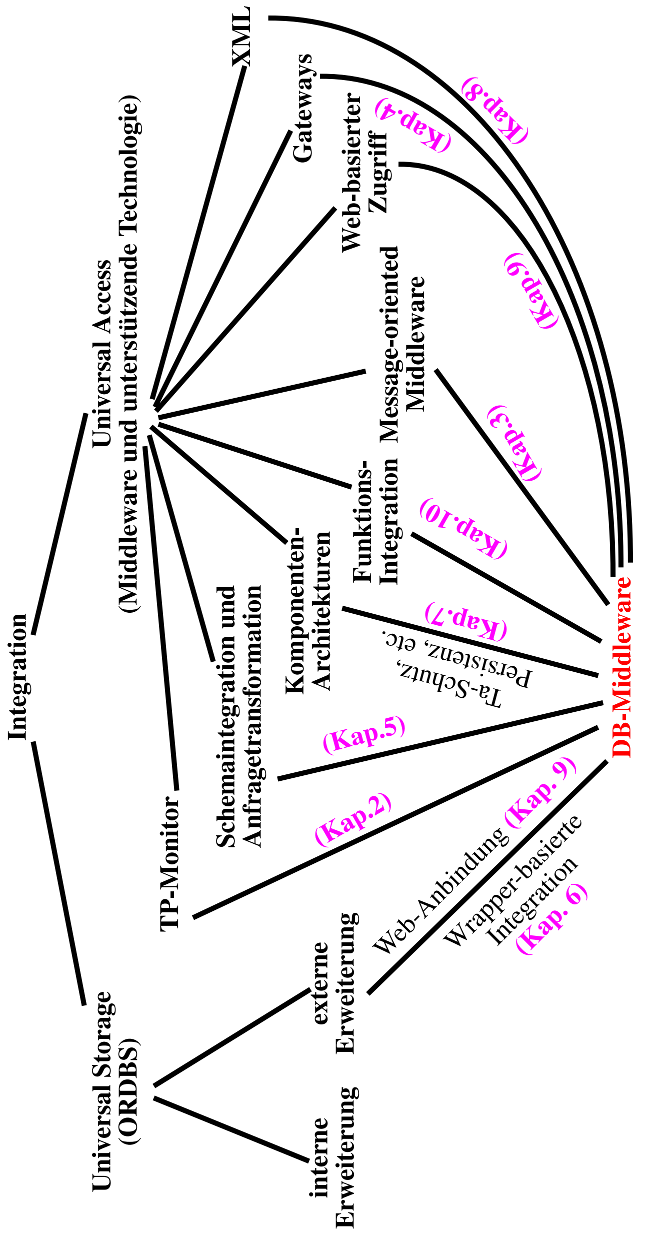
Kombinationen (2)

- Beispiel 2: ‘Interoperable Komponenten’, ‘Datenintegration’ und ‘1 ORDBS’ (‘Big Picture’ der IBM)



DB-Middleware

❑ Übersicht



Zusammenfassung

- ❑ Komponentenorientierung in der Entwicklung neuer (großer) Softwaresysteme
- ❑ DB-Sicht:
  - DBS-Zentrierung der Anwendungssysteme wird immer stärker ersetzt durch Komponentenorientierung
  - von zentraler Bedeutung: ***homogenisierte Sichten und sichere Abläufe***
  - **Universal Storage** mittels Erweiterbarkeit objekt-relationaler DBMS
  - **Universal Access** mittels *DB-Middleware*
  - in großen Informationssystemen sind kombinierte Ansätze von *Universal Storage* und *Universal Access* erforderlich!
- ❑ **DB-Middleware**
  - Verteilte Transaktionsverwaltung (TP-Monitor)
  - Gateways
  - MOM
  - Schemaintegration und Anfragetransformation
  - Wrapper-basierte (Daten-)Integration
  - DB-Aspekte von Komponentenarchitekturen
  - Datenaustausch (und ggf. Speicherung) mit XML
  - Web-basierter DB-Zugriff
  - Konnektor-basierte (Funktions-)Integration