3. Verteilte Datenbanksysteme - Schemaarchitektur und Katalogverwaltung

■ **Definition**

Ein verteiltes Datenbanksystem (VDBS) ist ein integriertes SW-System zum Aufbau, zur Verwaltung und Kontrolle von Datenbanken, die auf mehrere Rechner verteilt sein können. Es bietet allen Anwendungsprogrammen eine lokale und logische Sicht der Daten.

■ **Einführung**

■ **Schemaarchitekturen**

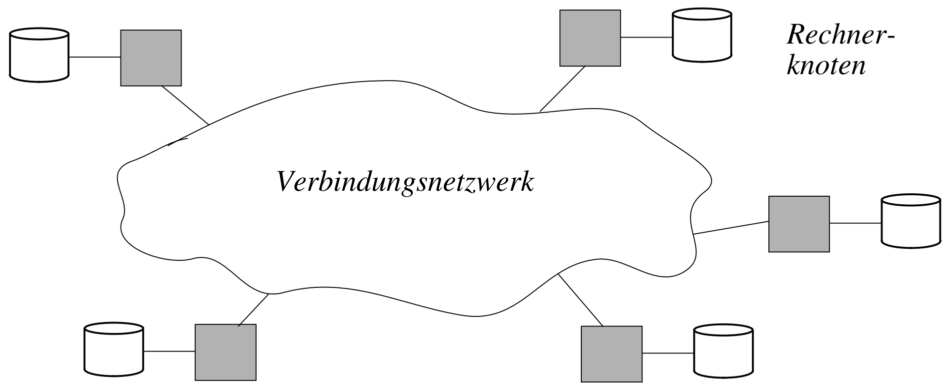
- Drei-Schema-Architektur
- Homogene verteilte DBS
- Homogene integrierte DBS
  (Phasen der Schema-Integration)
- Heterogene integrierte DBS
- Kopplung heterogener DBS

■ **Katalogverwaltung**

■ **Namensverwaltung**

■ **Namensauflösung**

Grobaufbau eines verteilten DBS



■ **Beispiele kommerzieller Systeme**

Viele kommerzielle DBS unterstützen nicht die volle Funktionalität von verteilten DBS

- Tandem NonStop SQL
- Oracle
- Sybase Replication Server
- IBM DRDA (DB2, SQL/DS, ...)
- UDS-D

■ **Frühe Prototypen**

- R* (IBM)
- SDD-1 (CCA)
- Distributed Ingres
- VDN, POREL, DDM, DDTS, Sirius-Delta, Polypheme, . . .

Wünschenswerte Eigenschaften von verteilten DBS

■ **Fundamentales Prinzip:**
Für den Benutzer sollen alle Aspekte der Verteilung verborgen bleiben (Verteilungstransparenz)

■ **12 Regeln für verteilte DBS** (nach Date)**:**

1. Größtmögliche lokale Autonomie:
   lokale Verwaltung von lokalen Daten
2. Keine Abhängigkeit zu zentralen Knoten
3. Permanenter Betrieb
4. Ortsunabhängigkeit (Ortstransparenz)
   - physische Lokation von Daten muß verborgen bleiben
   - Datenumverteilungen haben keine Auswirkungen auf Programme
5. Fragmentierungsunabhängigkeit
6. Replikationsunabhängigkeit
7. Verteilte Anfragebearbeitung
   - erforderlich für Zugriff auf nicht-lokale Daten
   - Optimierung verteilter Anfragen
8. Verteilte Transaktionsverwaltung:
   Synchronisation, Recovery (verteiltes Commit-Protokoll)
9. Hardware-Unabhängigkeit
10. Betriebssystemunabhängigkeit
11. Netzwerkunabhängigkeit
12. DBVS-Unabhängigkeit (➥ heterogene DBVS)

Schema-Architekturen

■ **Wichtigstes Entwurfsziel:**
möglichst hoher Grad an Datenunabhängigkeit

■ **Ausgangspunkt:**
Drei-Schema-Architektur nach ANSI/SPARC

- externes Schema
- konzeptuelles Schema
- internes Schema

➥ Kapselung und explizite Abbildung (Transformation)

■ **Anforderungen**

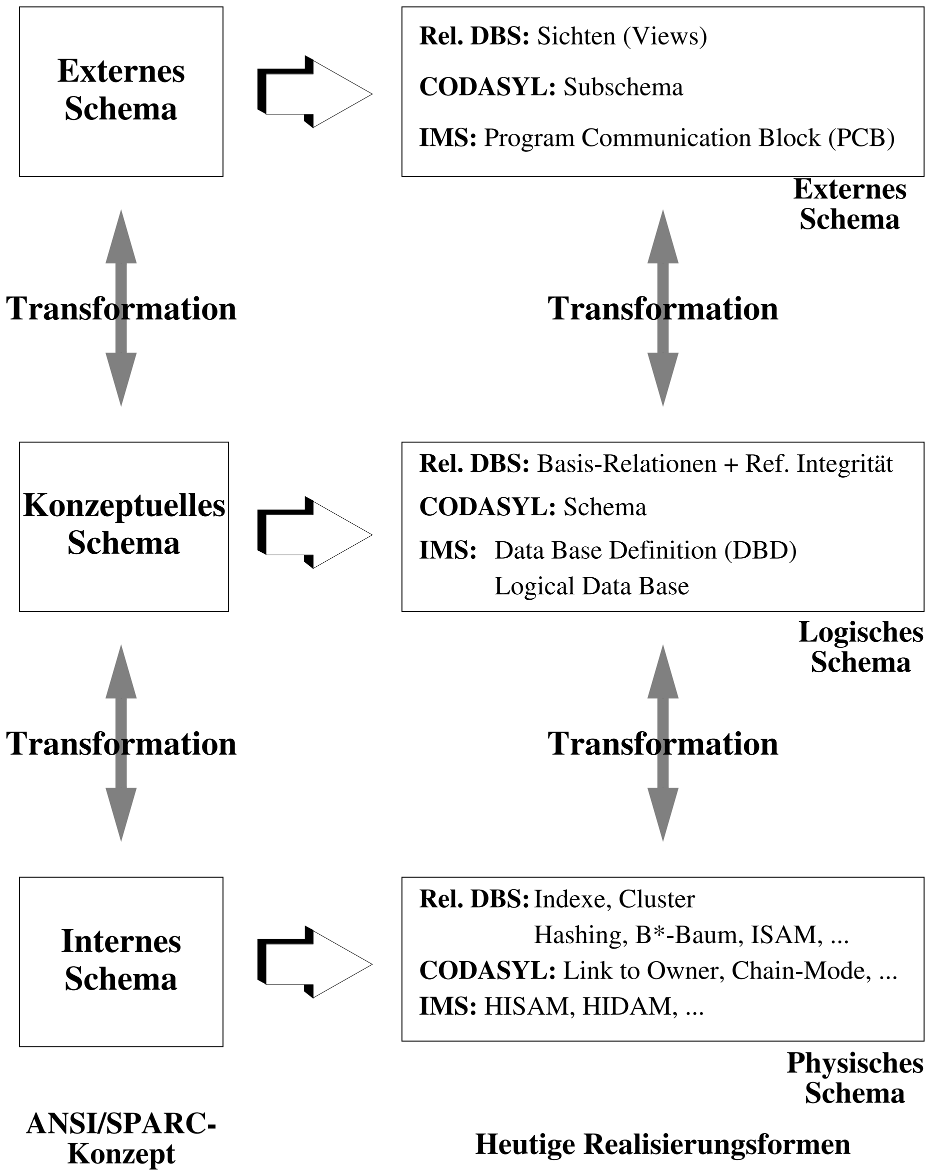
- Bereitstellung der Sicht <u>eines</u> globalen Schemas
  (➥ globale Relationen)
- Bereitstellung von externen Sichten auf das globale Schema
- **„Verstecken“** (vor dem Benutzer / AP)
  - von Heterogenitäten in den lokalen Schemata/Datenmodellen
  - der Fragmentierung der globalen Relationen
  - der Allokation der Fragmenten/Relationen
  - von redundanten Speicherungen von Fragmenten
- Bereitstellung von Informationen über
  (für Anfrage-Prozessor / DBA)
  - Fragmentierung der globalen Relationen
  - physische Speicherungsorte von Fragmenten
  - redundant gespeicherte Fragmente
  - anzuwendende Aktualisierungsstrategie für Kopien

■ **Komplexität der Schema-Architektur** bestimmt durch

- Homogenität/Inhomogenität der lokalen konzeptuellen Schemata
- Art der Datenverteilung (disjunkt/überlappend)

Schema-Architekturen (2)

■ **Aktuell realisierte Formen der Drei-Schema-Architektur**



Beschreibung der Datenstrukturen

■ **Problemstellung**
Es soll Datenunabhängigkeit wie im zentralen Fall erzielt werden

- zusätzlich: Verteilungsunabhängigkeit und
- in heterogenen verteilten DBS:
  ein globales Datenmodell (➥ eine Anfragesprache)

➥ Ausgangspunkt: Drei-Schema-Architektur

■ **Lokales Internes Schema (LIS):**
wie in zentralen DBS

■ **Lokales Konzeptuelles Schema (LKS):**
wie KS im zentalen DBS, Datenmodell, Datentypen, Maßeinheiten evtl. von Knoten zu Knoten verschieden

■ **Lokales Repräsentations-Schema (LRS):**
bildet LKS auf ein homogenes Datenmodell ab (evtl. mit Informationsverlust). Vereinheitlichung von Datentypen, Maßeinheiten, ... LRS repräsentiert die lokale DB nach "außen".

■ **Globales Verteilungsschema (GVS)**
setzt sich zusammen aus **globalem Fragmentierungsschema (GFS)** und **globalem Allokationsschema (GAS):**
Es beschreibt Zerlegung und Ortsverteilung der Daten (Fragmente, Replikate, ...)

■ **Globales Konzeptuelles Schema (GKS):**
vereinigt alle LRS_i (simuliert Sicht eines DBS)

■ **Globales Externes Schema (GES):**
erlaubt verschiedene Sichten auf die globale DB

Beschreibung der Datenstrukturen (2)

- **"Lokale" Benutzersicht**

  Welche Daten sieht der Benutzer?

➥ Globales Externes Schema

- **Globale Systemsicht**
  - Relationen und Sichten im Gesamtsystem

➥ Globales Konzeptuelles Schema

  - Aufteilung der Relationen in Fragmente
  - Zuordnung der Fragmente zu Rechnern (ggf. mit Replikation)

➥ Globales Verteilungs-Schema

- **Lokale Systemsicht**
  - Repräsentation der lokalen DB nach "außen"

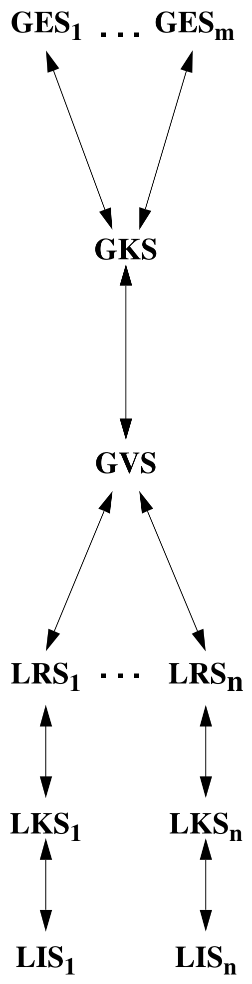
➥ Lokales Repräsentations-Schema

  - logischer Aufbau der lokalen DB

➥ Lokales konzeptuelles Schema

  - Speicherung der lokalen DB

➥ Lokales Internes Schema



- **Abbildungsprobleme (auch beim RM)**
  - Unverträgliche Datentypen, fehlende Funktionen . . .

Homogene verteilte DBS

- **Neuentwicklung eines Anwendungssystems**
  - Datenhaltung wird von vornherein als verteilte DBS ausgelegt
  - freie Wahl eines geeigneten Datenmodells
  - allgemeine Vorteile
    - angepaßtes Leistungsverhalten
    - Ausfallsicherheit
    - Inkrementelle Erweiterbarkeit
    - . . .

  ➥ **dadurch keine „Altlasten“ (keine existierenden AP)**

- **Ausgangssituation:**
  - keine Entwurfsbeschränkungen („grüne Wiese“)
  - nach logischem DB-Entwurf globale Relationen vorgegeben
  - Berücksichtigung von Leistungsaspekten führt zu abgeleiteten Fragmenten und ihre Allokation

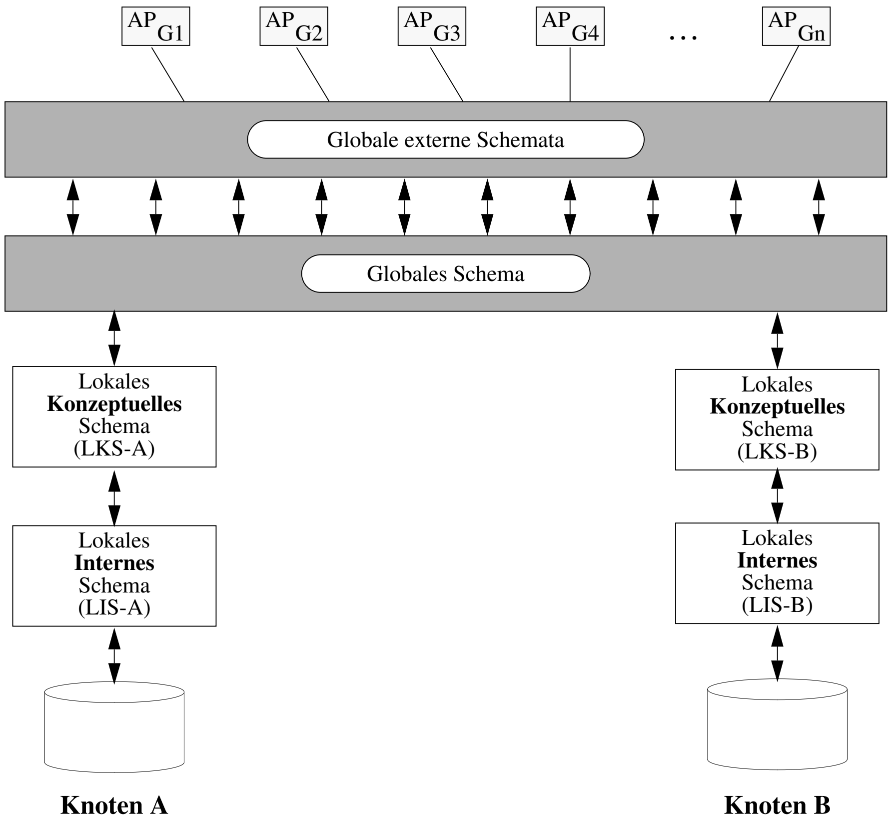
  ➥ **Schemaentwurf „aus einen Guß”**

- **Heterogenität** in den lokalen Schemata wird vermieden
  - gleiches Datenmodell
  - strukturell homogen
  - semantisch homogen
  - einfache Ableitung/Integration der lokalen/globalen Schemata

  ➥ **lokale Repräsentationsschemata sind nicht erforderlich**

Homogene verteilte DBS (2)

■ **Homogenität erlaubt einfachere Schema-Architektur**



■ **Globales Schema: Intern wiederum in Drei-Schema-Architektur**

- Globales konzeptuelles Schema (GKS)
  ➥ (globale) „Außensicht“
- Globales Fragmentierungs-Schema (GFS)
  ➥ Fragmentierung der Relationen des GKS (nur intern sichtbar)
- Globales Allokations-Schema (GAS)
  ➥ physische Plazierung der Fragmente (redundant/nicht-redundant)

Homogene verteilte DBS (3)

■ **Globales Schema (Gesamtsicht)**

GKS

Globaler Katalog

Abteilung (AbtNr, AbtName, Bereich, MgrPersNr, Budget)
Angest (PersNr, AngName, Gehalt, AbtNr, Anschrift)
Inventar (InvNr, Bezeichnung, AnschJahr, AktWert, AbtNr)
Lagerort (TeileNr, LagerNr)
Lieferant (LiefNr, LiefName, Stadt)
Teile (TeileNr, LiefNr, Preis)

GFS

Globale Relationen

Glob. Relation	Abbildung
Abteilung	Abteilung1 **UN** Abteilung2
Angest	Angest1 **JN** Angest2
Inventar	Inventar
Lagerort	Lagerort
Lieferant	Lieferant1 **UN** Lieferant2
Teile	Teile1 **UN** Teile2

Fragmente

Fragment	Attribute	Glob. Relation	SelektionsPräd
Abteilung1	AbtNr, AbtName, Bereich, MgrPersNr, Budget	Abteilung	AbtNr<=300
Abteilung2	AbtNr, AbtName, Bereich, MgrPersNr, Budget	Abteilung	AbtNr > 300
Angest1	PersNr, AngName, AbtNr, Anschrift	Angest	-
Angest2	PersNr, Gehalt	Angest	-
Inventar	InvNr, Bezeichnung, AnschJahr, AktWert, AbtNr	Inventar	-
Lagerort	TeileNr, LagerNr	Lagerort	-
Lieferant1	LiefNr, LiefName, Stadt	Lieferant	LiefNr <= 200
Lieferant2	LiefNr, LiefName, Stadt	Lieferant	LiefNr > 200
Teile1	TeileNr, LiefNr, Preis	Teile	TeileNr <= 500
Teile2	TeileNr, LiefNr, Preis	Teile	TeileNr > 500

GAS

Allokation

Fragment	Lokaler Name	Primärknoten	...
Abteilung1	Abteilung@KnotenA	KnotenA	
Abteilung2	Abteilung@KnotenC	KnotenC	
Angest1	Angest1@KnotenA	KnotenA	
Angest2	Angest2@KnotenB	KnotenB	
Inventar	Inventar@KnotenC	KnotenC	
Lagerort	Lagerort@KnotenA	KnotenA	
Lieferant1	Lieferant1@KnotenB, Lieferant1@KnotenC	KnotenB	
Lieferant2	Lieferant2@KnotenA, Lieferant2@KnotenB	KnotenA	
Teile1	Teile1@KnotenA	KnotenA	
Teile2	Teile2@KnotenB, Teile2@KnotenC	KnotenC	

LKS-A
Abteilung(...)
Angest1(...)
Lagerort(...)
Lieferant2(...)
Teile1(...)

LKS-B
Angest2(...)
Lieferant1(...)
Lieferant2(...)
Teile2(...)

LKS-C
Abteilung(...)
Inventar(...)
Lieferant1(...)
Teile2(...)

Globales konzeptuelles Schema und globales Fragmentierungs-Schema

GKS

Globaler Katalog

Abteilung (AbtNr, AbtName, Bereich, MgrPersNr, Budget)
Angest (PersNr, AngName, Gehalt, AbtNr, Anschrift)
Inventar (InvNr, Bezeichnung, AnschJahr, AktWert, AbtNr)
Lagerort (TeileNr, LagerNr)
Lieferant (LiefNr, LiefName, Stadt)
Teile (TeileNr, LiefNr, Preis)

GFS

Globale Relationen

Glob. Relation	Abbildung
Abteilung	Abteilung1 **UN** Abteilung2
Angest	Angest1 **JN** Angest2
Inventar	Inventar
Lagerort	Lagerort
Lieferant	Lieferant1 **UN** Lieferant2
Teile	Teile1 **UN** Teile2

Fragmente

Fragment	Attribute	GlobaleRelation	SelektionsPräd
Abteilung1	AbtNr, AbtName, Bereich, MgrPersNr, Budget	Abteilung	AbtNr<=300
Abteilung2	AbtNr, AbtName, Bereich, MgrPersNr, Budget	Abteilung	AbtNr > 300
Angest1	PersNr, AngName, AbtNr, Anschrift	Angest	-
Angest2	PersNr, Gehalt	Angest	-
Inventar	InvNr, Bezeichnung, AnschJahr, AktWert, AbtNr	Inventar	-
Lagerort	TeileNr, LagerNr	Lagerort	-
Lieferant1	LiefNr, LiefName, Stadt	Lieferant	LiefNr <= 200
Lieferant2	LiefNr, LiefName, Stadt	Lieferant	LiefNr > 200
Teile1	TeileNr, LiefNr, Preis	Teile	TeileNr <= 500
Teile2	TeileNr, LiefNr, Preis	Teile	TeileNr > 500

Globales Allokations-Schema und lokale konzeptuelle Schemata

GAS

Allokation

Fragment	Lokaler Name	Primärknoten	...
Abteilung1	Abteilung@KnotenA	KnotenA	
Abteilung2	Abteilung@KnotenC	KnotenC	
Angest1	Angest1@KnotenA	KnotenA	
Angest2	Angest2@KnotenB	KnotenB	
Inventar	Inventar@KnotenC	KnotenC	
Lagerort	Lagerort@KnotenA	KnotenA	
Lieferant1	Lieferant1@KnotenB, Lieferant1@KnotenC	KnotenB	
Lieferant2	Lieferant2@KnotenA, Lieferant2@KnotenB	KnotenA	
Teile1	Teile1@KnotenA	KnotenA	
Teile2	Teile2@KnotenB, Teile2@KnotenC	KnotenC	

Abteilung(...)
Angest1(...)
Lagerort(...)
Lieferant2(...)
Teile1(...)

LKS-A

Angest2(...)
Lieferant1(...)
Lieferant2(...)
Teile2(...)

LKS-B

Abteilung(...)
Inventar(...)
Lieferant1(...)
Teile2(...)

LKS-C

Homogene verteilte DBS - Beispiel

■ **Anfrage**

```
Select   LiefName
From     Lieferant
Where    Stadt = 'KL'
```

- Zugriff auf welche Fragmente?
- Welche Knoten?
- Welche Zugriffsunterstützung?

■ **Einfügen**

```
Insert Into  Teile   (TeileNr, LiefNr, Preis)
             Values  (1000, 500, ...)
```

- Zugriff auf welche Fragmente?
- Welche Knoten?
- Welche Aktionen?

Homogene verteilte DBS - Beispiel (2)

■ **Löschen**

```
Delete From Lieferant Where LiefNr = 100
```

- Zugriff auf welche Fragmente?
- Welche Knoten?
- Welche Aktionen?

Homogene integrierte DBS

- **Ausgangssituation:**
  - Existierende DBS mit dem gleichen Datenmodell
  - **Nachträgliche** Integration in verteiltes DBVS
  - Existierende AP für lokale DBS müssen weiterhin ablauffähig bleiben
    - Investitionsschutz
    - „Über-Nacht-Portierung“ i.d.R. nicht möglich

- **Anforderungen**
  - Neuübersetzung (falls möglich) ist tolerierbar
  - (Größere) Eingriffe in Quelltexte sind nicht tolerierbar

➥ **Konsequenz:** Für existierene Anwendungen muß nach wie vor die „alte“ Sicht verfügbar sein

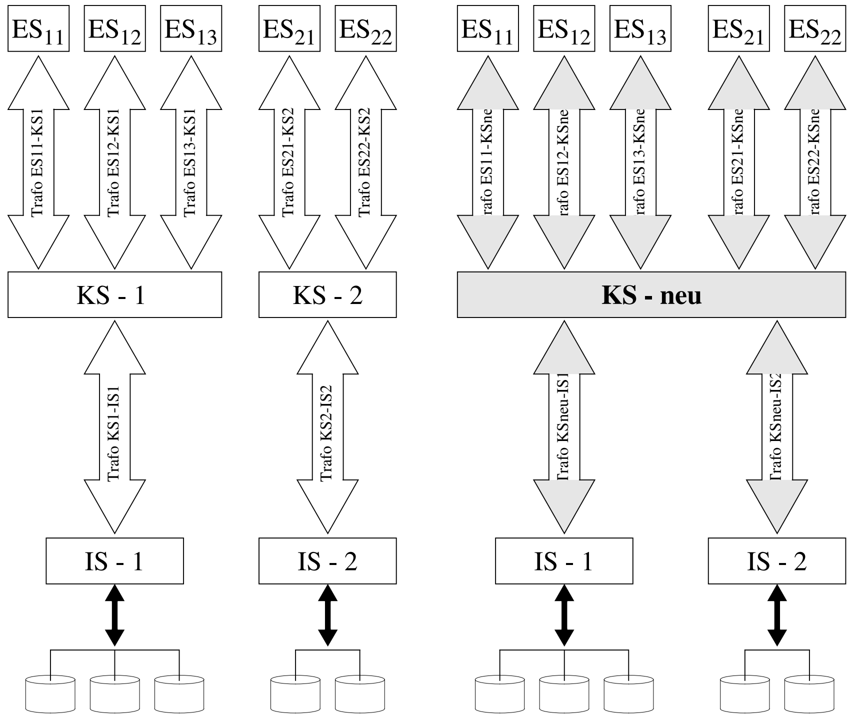
- **Vorgehensweisen**
  - „voll integrierte lokale Datenbanken” in einem Schritt: Ist das realistisch?
  - Beibehaltung der alten Sichten als lokale externe Schemata und separate Schema-Integration für globale Anwendungen
    - Bereitstellung eines lokalen Repräsentationsschemas (LRS) (Exportschemas) pro Knoten
    - LRS führen Homogenisierung der lokalen Schemata durch

➥ Schema-Integration läuft in mehreren Phasen ab

Homogene integrierte DBS (2)

- **Bei voller Integration der lokalen Datenbanken** ist im Prinzip folgende Vorgehensweise möglich:

1. Festlegung eines einheitlichen (globalen) konzeptuellen Schemas[1]
2. Anpassung der Abbildung konzeptuell ➥ intern
3. „Simulation“ der alten konzeptuellen Schemata durch Bereitstellung der „alten“ Sichten

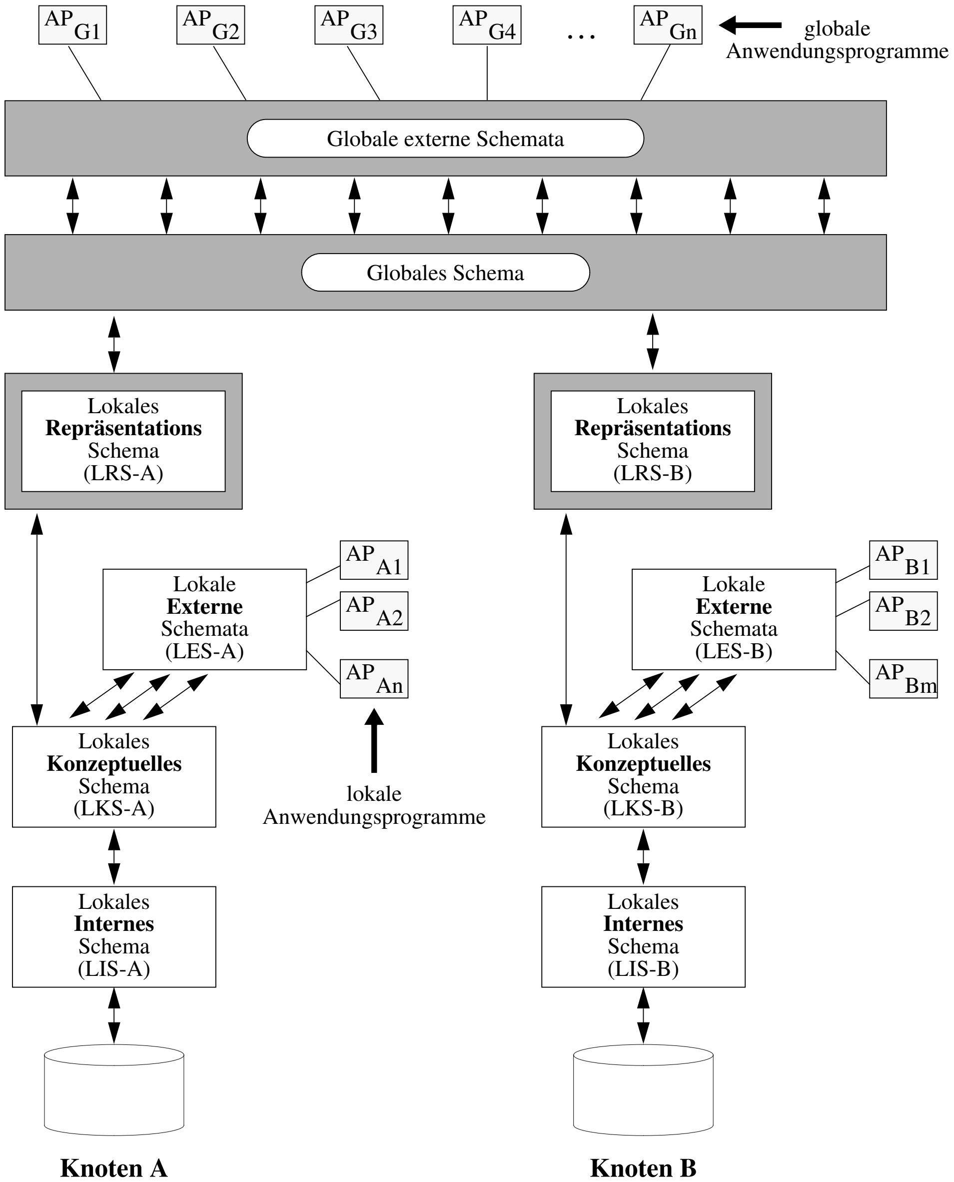


- **Problem:** Gleichzeitige Änderung aller vorhandenen konzeptuellen Schemata erforderlich (➥ Aufwand!)

1 d.h. ein gemeinsames Schema für alle lokalen DBVS

Homogene integrierte DBS (3)

■ **Gängige Lösung: Lokale Repräsentationsschemata**



Schema-Integration

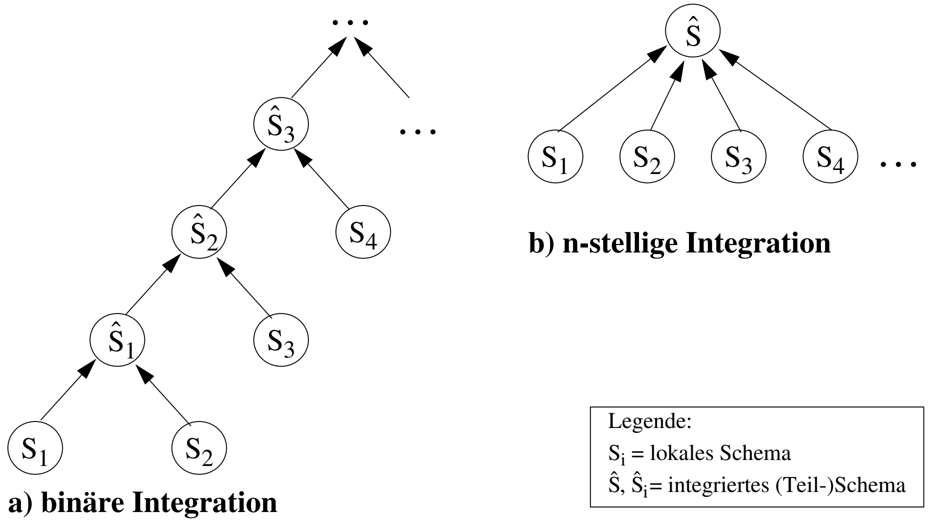
■ **Ziel:**
Abbildung der lokalen Schemata in ein einheitliches („integrierendes“) globales Schema

■ **Typischer Ablauf in vier Phasen:**

1. **Prä-Integrationsphase:**
   Festlegung der Vorgehensweise,
   Ermittlung der Objekte und Beziehungen
2. **Vergleichsphase:**
   Ermittlung von Namens- und Strukturkonfliken
3. **Vereinheitlichungsphase:**
   Festlegung des Zielschemas
4. **Restrukturierungs-** und Zusammenfassungsphase:
   Festlegung der lokalen Repräsentationsschemata und erforderlichen Abbildungen

■ **Prä-Integrationsphase**

- Binäre Integration vs. n-stellige Integration

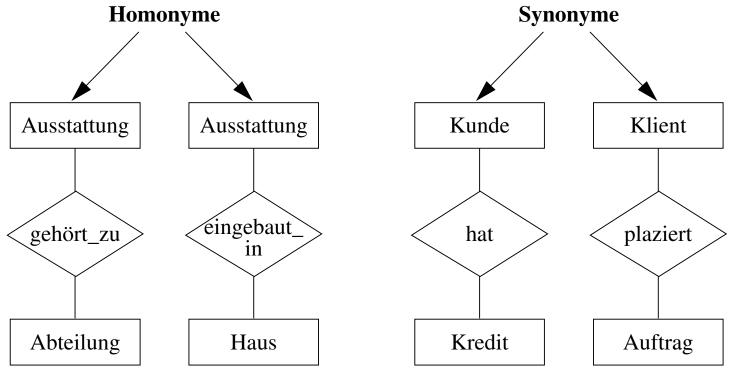


a) binäre Integration

b) n-stellige Integration

Schema-Integration (2)

- **Vergleichsphase**
  - Ermittlung von Namens- und Strukturkonfliken
  - ungeschickte und ungenaue Begriffe aufdecken und beseitigen



1. Synonyme kontrollieren!

   Wörter, die dieselbe Bedeutung haben und gegeneinander ausgetauscht werden können.

2. Homonyme beseitigen!

   Wörter, die gleich geschrieben und gesprochen werden, aber eine deutlich andere Bedeutung haben.

3. Äquipollenzen aufdecken!

   Dieselben Objekte werden unter verschiedenen Blickwinkeln betrachtet.

   z. B.: LAGERBESTAND als mengenmäßige und WARENKONTO als wertmäßige Rechnung über den Artikelbestand einer Firma.

4. Vagheiten klären!

   Da inhaltlich keine klaren Abgrenzung (Definition) der Begriffe erfolgt, treten hinsichtlich der Objekte, die unter diesen Begriff fallen, Unklarheiten und Unsicherheiten auf.

   z. B.: Gehört WOHNSITZ als Ort der Berufsausübung eines Beraters noch zum Begriff UNTERNEHMEN.

Schema-Integration (3)

- **Konfliktarten auf**
  - **Strukturebene** (Typebene):
    - Typkonflikte (Modellierung als Attribut vs. als Entity)
    - Beziehungskonflikte (1:1 vs. n:m)
    - Schlüsselkonflikte (unterschiedliche Primärschlüssel)
    - Verhaltenskonflikte (kaskadierendes vs. manuelles Löschen)
  - **funktionaler Ebene:**
    unterschiedliche DB-Operationen
  - **Instanzebene:** Ambiguitätsproblem
    - Mehrfachspeicherung mit unterschiedlichen Primärschlüsseln in den lokalen DBS
    - Verwendung desselben Primärschlüsselwertes (jeweils lokal) für global unterschiedliche Objekte

Knoten A

LiefRel	LiefNr	Name	. . .
	1826	ABC-Firma	
	2157	GHI-Firma	
	3984	JKL-Firma	
	. . .	. . .	

Knoten B

LiefRel	LiefNr	Name	. . .
	1977	ABC-Firma	
	2157	DEF-Firma	
	3572	MNO-Firma	
	. . .	. . .	

Schema-Integration (4)

■ **Vereinheitlichungsphase**
Entscheidung darüber, wie mit den zuvor erkannten Namens- und Strukturkonflikten verfahren werden soll

- **Homonyme:** Evtl. Namenserweiterung
(als Präfix Entitytyp und/oder Schemaname)

- **Strukturkonflikte:** Umformungen erforderlich:[1]
  - Entities in Attribute oder Beziehungen
  - Beziehungen in Entities
  - Attribute in Entities oder Beziehungen

➥ Dadurch oftmals nur noch lesender Zugriff möglich

- **Ambiguitätsprobleme** (auf Instanzebene)
Im Prinzip zwei Vorgehensweisen möglich:
  1. Einführung eines neuen, global eindeutigen Schlüssels
  2. Globale Verwendung von lokal erweiterten Schlüsseln

1 Ausführliche Diskussion in Batini, C., Lenzerini, M., Navathe, S.B.: A Comparative Analysis of Methodologies for Database Schema Integration, ACM Computing Surveys 18:4, 1986, pp. 323-364

Schema-Integration (5)

■ **Globale Schlüssel-Umsetzungstabelle (1)**[1]

GlobLiefTab	GlobLiefNr	Name	...	KnotenA_ID	KnotenB_ID
	1001	ABC-Firma	...	1826	1977
	1002	DEF-Firma	...	0	2157
	1003	GHI-Firma	...	2157	0
	1004	JKL-Firma	...	3984	0
	...	...	...	...	...

■ **Globale Schlüssel-Umsetzungstabelle (2)**

GlobLiefTab	LiefNr	KnotenID	Name	...
	1826	A	ABC-Firma	...
	1977	B	ABC-Firma	...
	2157	A	GHI-Firma	...
	2157	B	DEF-Firma	...
	3572	B	MNO-Firma	...
	...	...	...	...

globaler Schlüssel

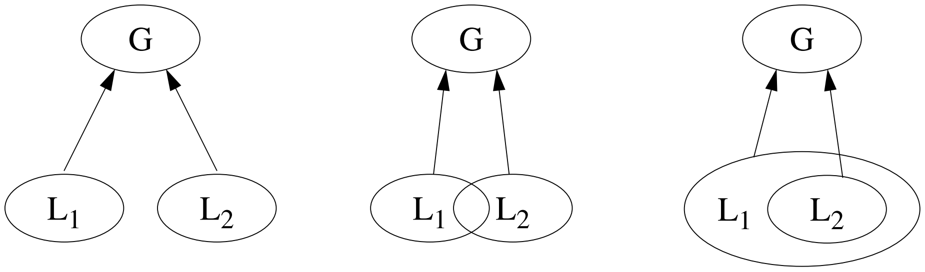
■ **Restrukturierungs- und Zusammenfassungsphase**

- Endgültige Entscheidung über das globale Schema
- Entwurfsziele: Globales Schema soll sein
  - vollständig
  - minimal (➥ Redundanzfreiheit auf Schemaebene)
  - verständlich

1 Bei großer Knotenanzahl oder häufigen Änderungen hinsichtlich Knotenzu- und -abgängen wäre sicherlich eine Modellierung der 1:n-Beziehung zwischen globalem Schlüssel und lokalen Schlüsseln mittels einer eigenen („Abbildungs“-)Tabelle vorteilhafter.

Schema-Integration - Aktualisierung

■ **Beziehung zwischen Entity-Mengen**



a) Disjunktheit **b) Überlappung** **c) Enthaltensein**

■ **Konflikte auf Ausprägungsebene möglich**

- Seien L1 und L2 Entity-Mengen und $dom(PK_{L1})$ und $dom(PK_{L2})$ die Wertebereiche ihrer Primärschlüssel.
- Für die Entity-Mengen L1 und L2 kann dann gelten:
  - **Disjunktheit:** $dom(PK_{L1}) \cap dom(PK_{L2}) = \emptyset$
    - kein Informationsverlust bei Integration
    - jedem globalen Entity eindeutig ein lokales Entity zuordenbar
    - damit Aktualisierung über das globale Schema prinzipiell in vollem Umfang möglich
  - **Überlappung:** $dom(PK_{L1}) \cap dom(PK_{L2}) \neq \emptyset$
    - kritisch bzgl. Einfügeoperationen
    - „Abhilfe“:
      - Zuordnungstabelle für Speicherungsort (aber hoher Aufwand!)
      - Einfügeregeln
    - Relativ unkritisch: Reine Wertänderungen und Löschungen
  - **Enthaltensein:** $dom(PK_{L1}) \subseteq dom(PK_{L2})$ (oder umgekehrt)
    - Problemstellung im wesentlichen wie bei Überlappung

Schema-Integration - Strukturelle Heterogenität

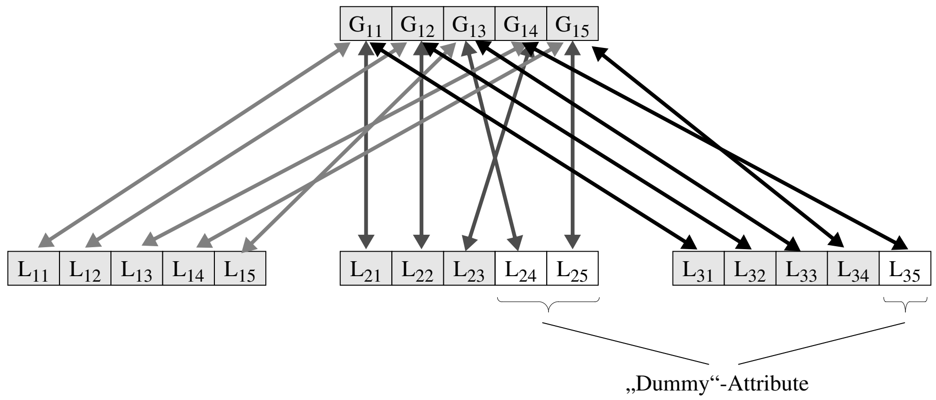
■ **Betrachtung von struktureller Heterogenität**

- In einfachen Fällen Mächtigkeit der Relationenalgebra bzw. SQL ausreichend
- In komplizierteren Fällen jedoch mächtigere Abbildungssprache erforderlich

■ **Integration von Relationen unterschiedlicher Stelligkeit**

- falls nur lesender Zugriff zu unterstützen:
  entweder „kleinster gemeinsamer Nenner“ oder
  „Vereinigungsmenge“ (➥ Nullwerte, „Dummy-Attribute“)
- Problem: Änderungen / Einfügungen
  Bei Aufrechterhaltung von Verteilungstransparenz grundsätzliche Versorgung der „Dummy-Attribute“ notwendig

■ **Verschärfung des Problems** bei unterschiedlichen Wertebereichen (z.B. Knoten A numerisch, Knoten B Text)



Homogenisierung struktureller Heterogenität

Schema-Integration - Strukturelle Heterogenität (2)

■ **Behandlung von erheblichen Strukturabweichungen**

- Beispiel: Verwaltung von Aktienbeständen an zwei Knoten

Knoten A habe die Aktien in der folgenden Form gespeichert:

Aktien	Firma	Menge	Kaufdatum	Kurs
	HP	1.000	01.05.10	...
	IBM	2.000	01.08.13	...
	Sun	1.500	02.02.18	...
	Unilever	500	01.08.09	...
	Henkel	800	02.03.18	...
	...	...	...	...

Knoten B hingegen habe folgende Form gewählt:

HP-Aktien	Menge	Kaufdatum	Kurs
	500	01.01.10	...
	...	...	...

IBM-Aktien	Menge	Kaufdatum	Kurs
	1.500	01.12.15	...
	...	...	...

SUN-Aktien	Menge	Kaufdatum	Kurs
	1.000	02.03.10	...
	...	...	...

SonstigeAktien	Firma	Menge	Kaufdatum	Kurs
	Unilever	1.000	02.01.20	...
	Ciba	1.500	01.11.02	...
	...	...	...	...

➥ **Einsatz** von (Prädikaten-)Logik-basierten **Transformationssprachen**

Heterogene integrierte DBS

■ **Ausgangs-Situation**

- ähnlich wie bei homogenen integrierten DBS
- Zusätzlich jedoch: unterschiedliche Datenmodelle

■ **Ziel (weiterhin): Globale Sicht als <u>ein</u> DBS**

- beteiligte Datenmodelle sind auf globales „Datenmodell" (relationales Datenmodell) abzubilden
- Problem: Bereitstellung einer voll-funktionalen relationalen LRS-Schicht (Schema + Operationen)

■ **Einfacher Fall:**
Nicht-relationales DBVS stellt selbst SQL-Schnittstelle bereit[1]

➥ Problem reduziert sich auf Beschreibung des verfügbaren SQL-Subsets, wie z.B.

- kein Update, keine DDL-Operationen (wie z.B. CREATE TABLE)
- Beschränkung auf „einfache“ SELECT-Ausdrücke (z.B. keine Exists-Klausel, kein Join, kein Group By usw.)
- ...

■ **Im allgemeinen Fall**

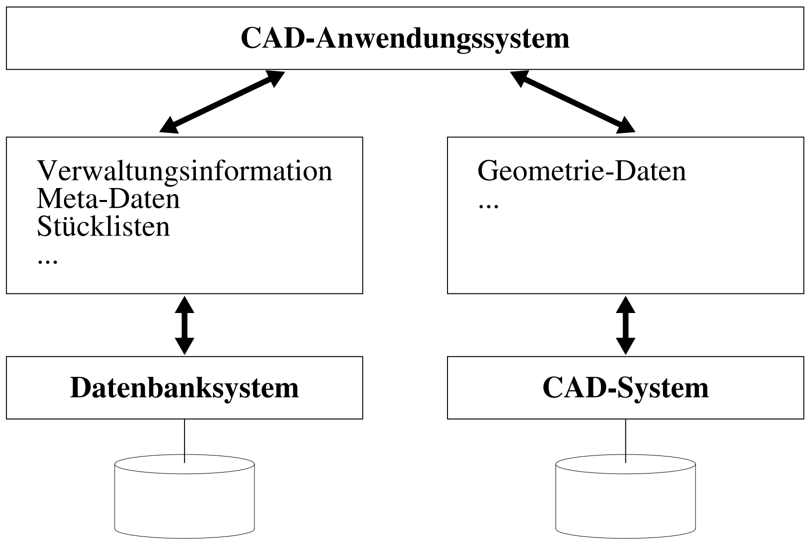
- Bereitstellung einer „homogenisierenden“ LRS-Schicht
- Bereitstellung von relationalen LRS-Operationen
- Bereitstellung des verfügbaren SQL-Subsets
- ...

➥ **Komplexität kann „beliebig” anwachsen!**
Deshalb oft nur Kopplung der beteiligten DBS

1 wie z.B. UDS/SQL

Kopplung heterogener DBS

- **Existierende heterogene DBS**
  - sollen im Rahmen einer **Anwendungsintegration** zusammenarbeiten
  - oft Spezialsysteme mit eigener DBVS-Komponente
- **Beispiel: CAD-Systeme mit DB-Komponente**
  - Geometrische Objekte, interne Objekt-Darstellung
    - ➥ Datenhaltung in CAD-Dateisystem
  - Verwaltungsdaten
    (Bemaßungen, Werkstoffdaten, Stückliste, ...)
    - ➥ relationale Datenbank



- **Eher Kopplung statt Integration**
  - Kein gemeinsames „Datenmodell“
  - Auf Implementierungsebene Heterogenität voll sichbar
  - „Verstecken“ der Heterogenität nur durch Anwendungssystem

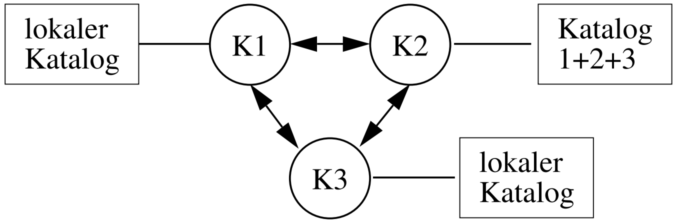
Kopplung heterogener DBS (2)

- **Mögliche zukünftige Entwicklungen**
  - Standard-DBVS, Geo-DBVS
  - Image-DBVS, Video-DBVS
  - Wissensbank-Verwaltungssystem
  - . . .
  - ➥ **Spezielle Realisierungen für Speicherungsstrukturen, Indexstrukturen, DB-Pufferverwaltung und zugehörige Operationen durch jeweiliges DBVS**
- **Idee der Komponentenorientierung** ist an sich wünschenswert:
  Verhalten nach außen hin wie ein monolithisches DBVS
- **Mögliche Probleme**
  - Unterschiede in den ACID-Eigenschaften
    (z. B. unterschiedliches Verhalten im Fehlerfall)
  - Aufeinander abgestimmte, äquivalente Basis-Funktionalitäten müssen in allen Komponenten verfügbar sein
    - Integritätssicherung
    - Synchronisation
    - Recovery
    - Transaktionsverwaltung
    - . . .
- **Gemeinsames Datenmodell, gleiche Basis-Operationen**
  - überhaupt realisierbar?
  - überhaupt erstrebenswert?
  - ➥ **Sind ORDBS ein Integrationsvehikel?**

Katalogverwaltung

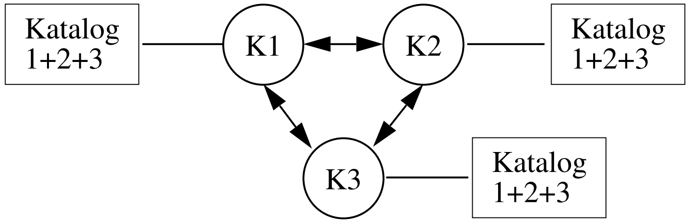
- **Katalog ist ein überall bekanntes Objekt**
- **Er führt Metadaten für DB-Verarbeitung**
  - Namen u. Adressen externer Knoten (= DBVS-Instanzen)
  - Angaben zur Datenverteilung
  - Angaben zu Relationen, Sichten, Attribute, Integritätsbedingungen, Benutzern, Zugriffsrechten, Indexstrukturen, Statistiken, ...
- Jeder Knoten sollte **für lokale Objekte Katalogdaten lokal** halten. Warum ist dieses Caching wichtig ?
- **Alternativen für globalen Katalog** (Verteilungsinformationen, Angaben zu nicht-lokalen Objekten und Benutzern):

1. Zentralisierter Katalog



➥ **Vermeide zentralisierte Kataloge**

2. Vollständig replizierter Katalog



➥ **Wartung sehr teuer, volle Redundanz unnötig**

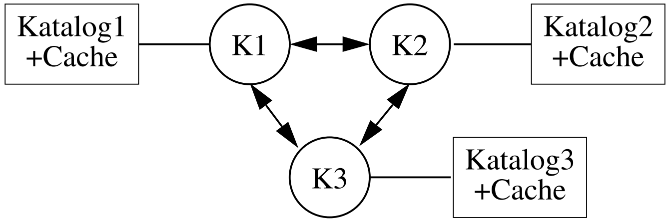
Katalogverwaltung (2)

- **Weitere Alternativen**

3. Mehrfachkataloge: Kombination aus den beiden ersten Ansätzen
   Pro Cluster enthält ein Knoten jeweils den gesamten Katalog

4. Partitionierter Katalog

   hier mit Caching



- **Variante:** Partitionierte Kataloge + Pufferung (Caching) von entfernten Katalogdaten
- **2 wesentliche Alternativen** zur Behandlung veralteter Katalogangaben:
  - Besitzerknoten vermerkt sich, wo Katalogdaten gepuffert sind und invalidiert diese bei einer Änderung (SDD-1)
  - Verwendung von Zeitstempeln: bei Ausführung einer Operation wird festgestellt, ob veraltete Katalogdaten verwendet wurden; ggf. Neuübersetzung und -ausführung mit aktualisierten Daten (R*)

Namensvergabe

■ **Anforderungen**

- Eindeutige Bezeichner globaler Objekte: Relationen, Sichten, Indexe usw.
- Stabilität gegenüber Datenumverteilungen (Migration)
- Unterstützung von Verteilungstransparenz
- Lokale Namensvergabe wünschenswert

■ **Struktur des Namensraums**

- global
  - Einsatz von Namens-Servern oder Namenskonventionen
  - Zuverlässigkeitsproblem
- hierarchisch
  - gewährleistet Knotenautonomie
  - toleriert Netzwerk-Partitionierung
  - paßt sich dem Wachstum an

■ **Denkbarer Ansatz: dreiteilige Objektbezeichnungen**

[[<node-id>.]<user-id>.]<object-id>

- lokale Namenswahl durch Benutzer wie in zentralisierten Systemen
- verschiedene Benutzer können die gleichen Objektnamen verwenden
- Referenzierung lokaler Objekte wie im zentralen Fall
- jedoch Verwendung von <node-id> für externe Objekte erforderlich
  ➥ Verletzung der Ortstransparenz
  ➥ Änderung der Datenallokation erfordert Programmänderungen !

■ **Abhilfemöglichkeiten**

- Verwaltung von Synonymen (Aliases) für jeden Benutzer (automatische Abbildung auf vollen Objektnamen durch DBS)
- bei Datenmigration nur Anpassung der Synonymtabellen bzw. in ursprünglichem Knoten wird Vorwärtsverweis auf neue Datenlokation gespeichert

Namensvergabe in R*

■ **Syntax:**

<NAME>:: = [<user> [@<user_ node>].] <object_name>
[@<birth_node>] *Ort der Objekterzeugung*

■ **Expansionsregeln für systemweite Namen**

- fehlender <user> wird ersetzt durch aktuelle USERID
- fehlender <user_node> wird ersetzt durch aktuelle KNOTENID
- fehlender <birth_node> wird ersetzt durch aktuelle KNOTENID

■ **Beispiel**

Erzeugung eines Objektes My-Rel in F durch Benutzer Ernie in KL:

Ernie@KL.My-Rel@F

	Aufruf durch	
	Ernie	Sonstige Benutzer
in KL		
in M		
in F		

■ **Vorteile:**

- Knotenautonomie
- Migration eines Objektes hat keine Auswirkungen auf Namen und damit bestehende Programme

■ **Umständliche Adressierung,** falls Objekt nicht an Benutzerknoten gespeichert wird

- mindestens 1 Knotenname muß angegeben werden
- Erleichterung durch Synonyme:

  DEFINE SYNONYM Test AS Ernie@KL.My-Rel@F

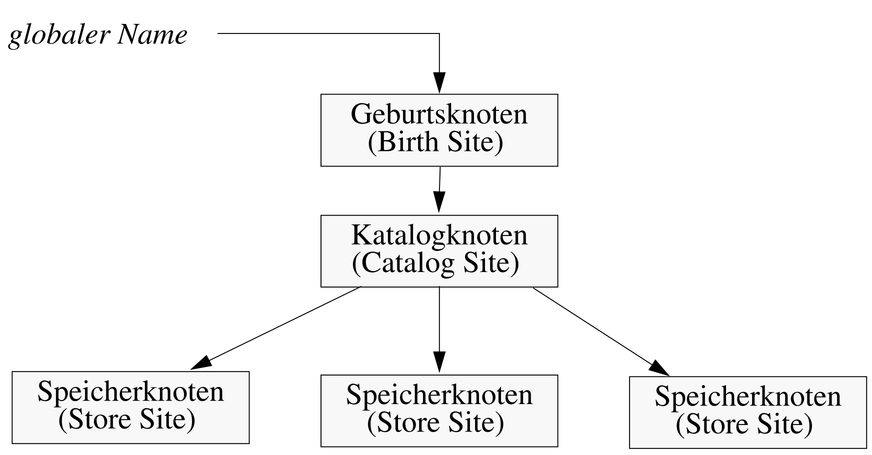
Namensauflösung

■ **Bestimmung der physischen Adresse** zu einem logischen Objekt

Beteiligte Rechner:

- Geburtsknoten ("birth site"); sei im globalen Namen enthalten
- Katalogknoten ("catalog site")
- Speicherknoten ("store site")

■ **Schrittweise Namensauflösung**



- Unterstützung von Replikation (mehrere Speicherknoten)
- **Trennung von Geburts- und Speicherknoten** erlaubt Stabilität gegenüber Datenumverteilungen
- Katalogknoten kann mit Geburts- oder Speicherknoten übereinstimmen (➥ Kommunikationseinsparungen)

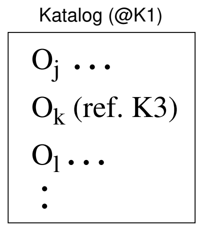
Namensauflösung (2)

■ **Ansatz: Partitionierte Kataloge + Pufferung**

- Der Katalog bei <birth_node> von O_i enthält einen Eintrag für O_i
- Jeder lokale Katalog enthält einen Eintrag für jedes lokal gespeicherte Objekt
- Die Suche nach O_i (lokal oder entfernt) benutzt diese Objekt-beschreibungen

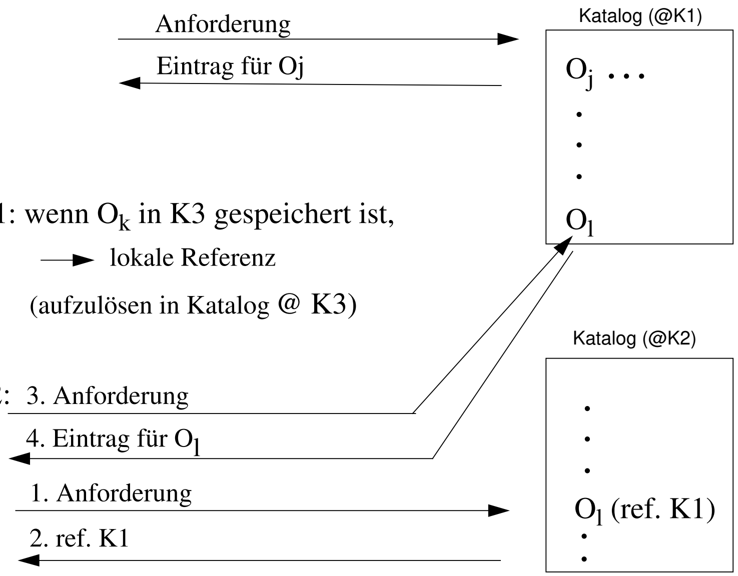
■ **Lokale Referenz**

- O_j @ K1 : (Speicherungsknoten)
- O_k @ K1 : (migriert nach K3)
- O_l @ K2 : (migriert nach K1)



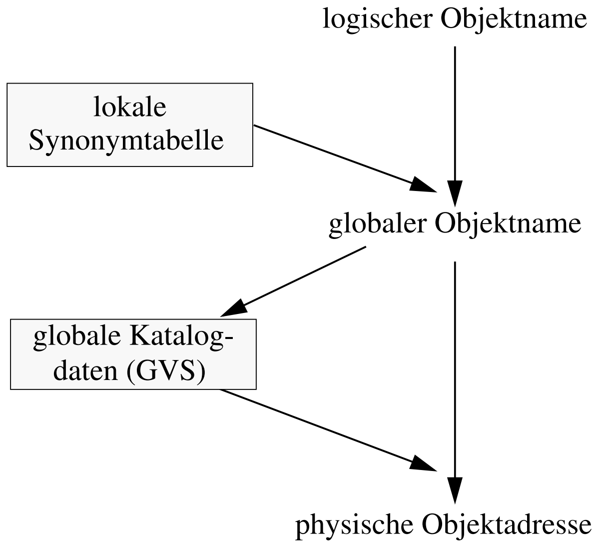
■ **Entfernte Referenz (von K3 aus)**

- O_j @ K1
- O_k @ K1: wenn O_k in K3 gespeichert ist,
  → lokale Referenz
  (aufzulösen in Katalog @ K3)
- O_l @ K2:



Namensauflösung über Synonyme

- **Synonyme (Alias-Namen):**
  Abbildung benutzerspezifischer logischer Namen in vollqualifizierte globale Namen
  - Verwaltung von Synonymtabellen durch DBS im lokalen Katalog
  - Verwendung in vielen kommerziellen Systemen:
    Tandem NonStop SQL, DB2, Oracle etc.



Zusammenfassung

- **Teilweise gegensätzliche Regeln / Ziele für verteilte DBS**
  (vollständige Transparenz vs. Knotenautonomie / Heterogenität)
- **Verteilungstransparenz**
  - Ortstransparenz
  - Fragmentierungstransparenz
  - Replikationstransparenz
- **Flexible Schemaarchitektur**
  - Lokale Benutzersicht: Globales Externes Schema
  - Globale Systemsicht: Globales Konzeptionelles Schema, Globales Verteilungs-Schema
  - Lokale Systemsicht: Lokales Repräsentations-Schema, Lokales Konzeptionelles Schema, Lokales Internes Schema
- **Schemaarchitekturen für**
  - Homogene verteilte DBS
  - Homogene integrierte DBS
  - Heterogene integrierte DBS
  - Kopplung heterogener DBS
- **Namensvergabe und Katalogarchitektur**
  - hierarchische Namensstruktur
  - Unterscheidung von Geburts-, Katalog- und Speicherknoten
  - Verteilte DBS: partitionierte Kataloge + Pufferung
  - PDBS: replizierte Kataloge
- **Globale Objektnamen**
  - lokale Vergabemöglichkeit über hierarchische Namen
  - Ortstransparenz über Synonyme