2. Klassifikation von Mehrrechner-DBS

- **Ziel:**

  Einsatz mehrerer Prozessoren oder DBVS
  zur DB-Verarbeitung

- **Arten der Rechnerkopplung**

- **Räumliche Verteilung**
  - feste Kommunikationsinfrastruktur
  - dezentral - selbstorganisierend - mobil
  - Externspeicheranbindung

- **Klassifikationsschema für MRDBS**

- **Shared-Nothing vs. Shared-Disk**
  - Leistungsfähigkeit
  - Verfügbarkeit
  - Erweiterbarkeit
  - Technische Probleme

- **Integrierte vs. Föderierte Mehrrechner-DBS**
  - Anwendungs- vs. Datenintegration
  - Architekturvorschlag zur Daten- und Funktionsintegration
  - Allgegenwärtige Anfrageverarbeitung im Internet

- **Prozessorfunktionalität**

- **Workstation/Server-DBS**

- **Grobbewertung von MRDBS**

Klassifikationsmerkmale

- **Rechnerkopplung**

  enge, lose oder nahe Kopplung

- **Räumliche Verteilung**
  - lokal oder ortsverteilt
  - Kommunikation:
    infrastrukturbasiert oder infrastrukturlos

- **Externspeicheranbindung**

  gemeinsam ('shared') oder partitioniert

- **Art der Systemkooperation**

  Integrierte vs. föderierte Mehrrechner-DBS

- **Art der Systemkomponenten**

  Homogene vs. heterogene DBS

- **Funktionale Spezialisierung vs. Gleichstellung der Prozessoren**

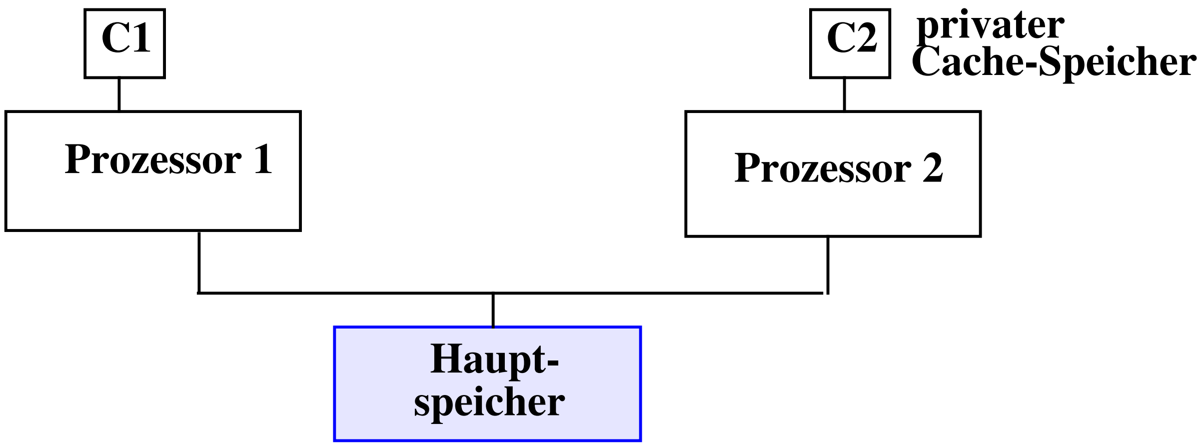
  vertikale vs. horizontale Verteilung

➥ Wie weit geht die Orthogonalität der Klassifikationsmerkmale?

Enge Rechnerkopplung (tightly coupled systems)

■ **Eigenschaften**

- Gemeinsamer Hauptspeicher für alle Prozessoren
- 1 Kopie von SW-Komponenten (BS, DBS, Anwendung, ...)
- HW-Cache pro Prozessor
- weit verbreitet (Symmetric Multiprocessing, SMP)



■ **Vorteile:**

- einfache Realisierung, wenig neue DB-Probleme
- effiziente Kommunikation über Hauptspeicher
- Lastbalancierung durch Betriebssystem
- Single System Image

■ **Nachteile:**

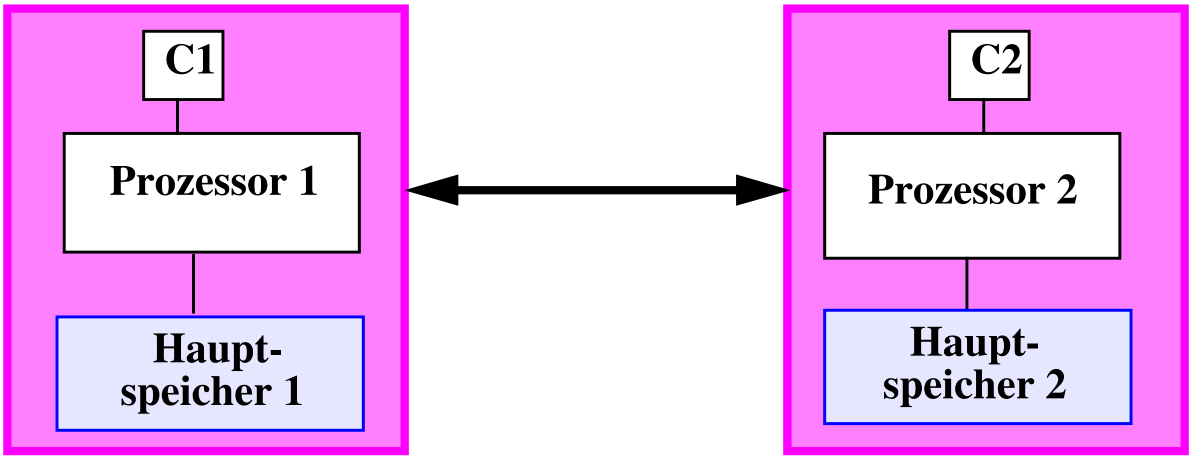
- Mangelnde Fehlerisolation
  (gemeinsame Speicher und SW-Komponenten)
- begrenzte Erweiterbarkeit (N < 30, meist N < 10)
- Cache-Kohärenz

■ **Beispiel:**

DBS auf eng gekoppelten Mainframes

Lose Rechnerkopplung
(loosely coupled systems)

- **Eigenschaften**
  - N autonome Rechner
    (separater Hauptspeicher pro Knoten,
    eigene Kopie von BS und DB/DC-System)
  - Kommunikation über Nachrichtenaustausch



- **Vorteile:**
  - höhere Fehlerisolation / Verfügbarkeit
  - bessere Erweiterbarkeit

- **Nachteile:**
  - Nachrichtenaustausch aufwendig (Kommunikations-Overhead)
  - kein Single System Image

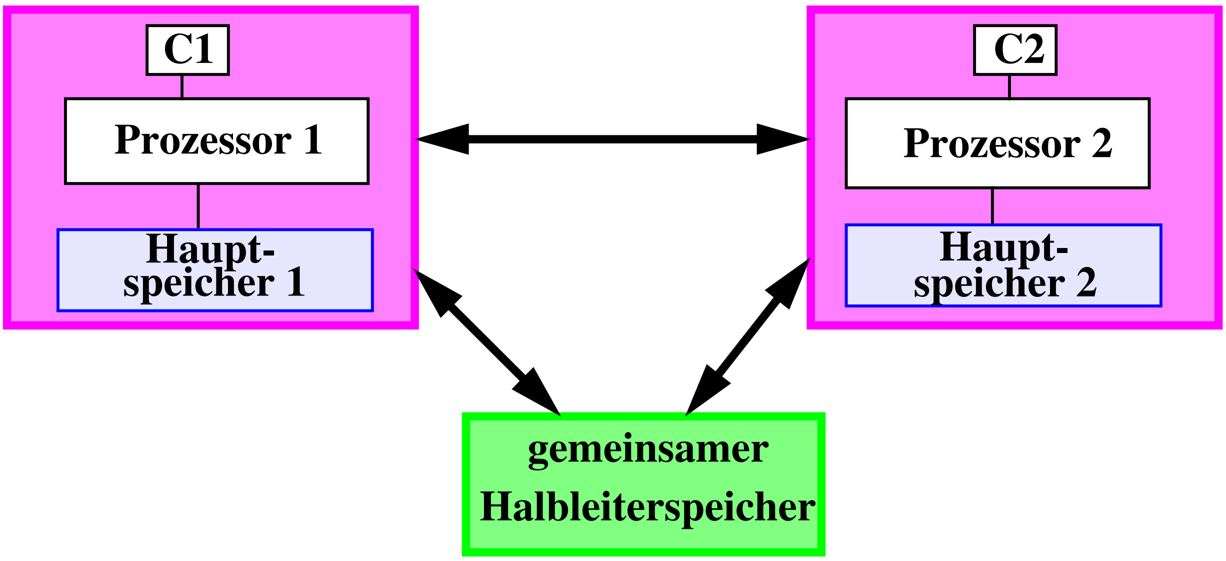
Nahe Rechnerkopplung
(closely coupled systems)

■ **Kompromiß zwischen enger und loser Kopplung**

Ziel: Effizientere Kommunikation als mit loser Kopplung unter Beibehaltung einer ausreichenden Fehlerisolation und Erweiterbarkeit

■ **Merkmale**

- N autonome Rechnerknoten
- Kommunikation (z. T.) über gemeinsame Halbleiter-Speicherbereiche
- Voraussetzung: lokale Rechneranordnung



■ **Speichereigenschaften**

- schneller Zugriff (Mikrosekunden und darunter) zur Umgehung von Prozeßwechseln (*synchroner Zugriff*)
- i. allg. keine Instruktionsadressierbarkeit
- ggf. nichtflüchtig
- ggf. doppeltes Führen von Speicherinhalten

■ **Weitere Einsatzform einer nahen Kopplung:**

Verwendung von Spezialprozessoren, z. B. einer 'Lock Engine' zur globalen Synchronisation

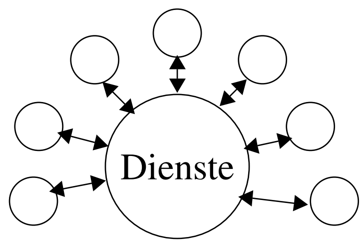
Räumliche Verteilung

■ **lokal:**

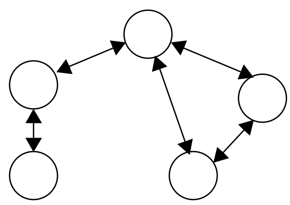
- schnelle Rechnerkopplung möglich
  (gemeinsame Speicher, Hochgeschwindigkeitsbus usw.)
- Nachrichtenaustausch effizienter und robuster als in Weitverkehrsnetzen (WAN: Wide Area Network)
  - einfachere Kommunikationsprotokolle
  - Broadcast-, Multicast-Verfahren
- effektive dynamische Lastverteilung möglich
- Unterstützung von Intra-Transaktionsparallelität
- einfachere Administration

■ **ortsverteilt:**

- Kommunikation
  - „klassisch": Festnetz (Internet)
  - mobil (z. B: MobileIP):
    Festnetzinfrastruktur erforderlich
  - drahtlos (IrDA, Bluetooth, WaveLAN):
    Festnetzinfrastruktur nicht zwingend erforderlich



- künftige Kommunikation
  - über selbstorganisierende Infrastrukturen
    ohne zwingendes Festnetz
  - mit mobilen Geräten, Diensten und Anwendungen

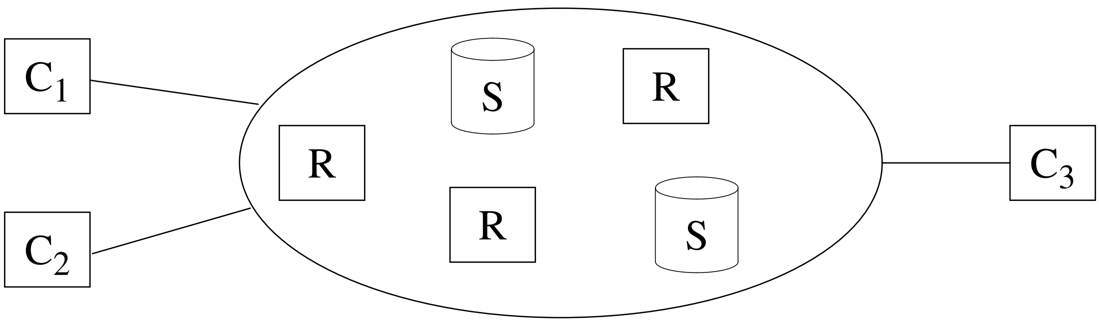


- Unterstützung dezentraler Organisationsstrukturen
- Voraussetzung für schnelle Katastrophen-Recovery
  (replizierte DB an entfernten Knoten)

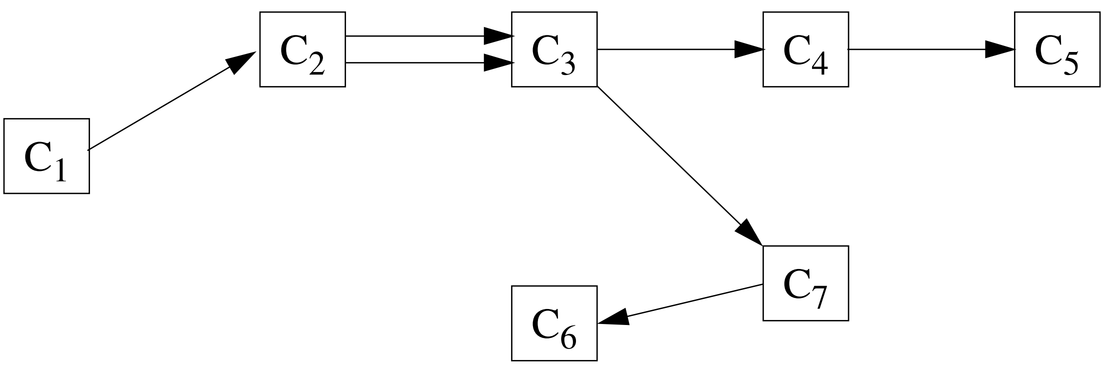
Dezentral - Selbstorganisierend - Mobil

Welche Kommunikationsinfrastruktur ist bei künftigen mobilen Systemen erforderlich?

- **Bisher: klare Trennung zwischen Endsystemen (Clients), Routern und Servern**
  - Anwendungen auf Endsystemen
  - Router sind dedizierte Systeme zum Weiterleiten (fest konfiguriert)
  - Client/Server-Paradigma mit Servern inner- und außerhalb der Netze



- **Bei dezentralen, selbstorganisierenden Infrastrukturen**
  - Peer-to-Peer-Paradigma
  - Mobile Geräte integrieren sowohl Funktionen der Endsysteme, der Router als auch der Server
  - Keine feste Infrastruktur erforderlich: **spontane Vernetzung**



Dezentral - Selbstorganisierend - Mobil (2)

Welche Charakteristika besitzen künftige Systeme?

- **Hohe Mobilität**
  - Geräte, Dienste und Anwendungen sind mobil
  - wesentlich höhere Dynamik als bei portablen (nomadischen) Systemen
  - Interaktionen mit Festnetz möglich, aber nicht vorausgesetzt
- **Selbstorganisation**
  - keine fest vorgegebene Infrastruktur (z.B. Server)
  - bedarfsorientierte Organisation
- **Verteilt, dezentral**
  keine zentralen Infrastrukturelemente
- **Heterogenität**
  ggf. sehr unterschiedliche Geräte und Kommunikationsmöglichkeiten

Welche (neuen) Dienste sind erforderlich?

- **Basisdienste**
  - Anwendungs-, Kontext-, dienstabhängiges Routing
  - Ressourcenverwaltung, Lastkontrolle, Lastverteilung
  - fortgeschrittene Kommunikationsformen (Anycast, Multicast)
- **Dienstplattformen und höherwertige Dienste**
  - Lokations- und kontextabhängige Dienste
  - Migration von Diensten
  - Verzeichnis- und Informationsdienste im Kontext hoher Dynamik - Welche Strukturen sind geeignet?

Dezentral - Selbstorganisierend - Mobil (3)

- **Semantische Interoperabilität**
  - für hochmobile Kommunikationspartner
  - unter Berücksichtigung existierender Infrastrukturen
  - Nutzung von Ontologien[1] wesentlich

Welche neuen Anwendungsgebiete?

- **E-Learning**
  - Einbindung mobiler Lernender
  - Spontane Bildung von Lernteams
- **E-Medicine**
  - Höhere Mobilität für Kranke und damit eine höhere Lebensqualität
  - Sensoren am Körper (Body Area Networks)
    - Beobachtung des Gesundheitszustandes
    - Alarmmeldung in Notsituationen unter Benutzung drahtloser Infrastruktur
- **Katastrophensituationen**
  - spontane Vernetzung von Notfallhelfern (Koordination der Helfer)
  - Bereitstellung kontextbezogender Information
- **Verkehrstelematik**
- **Ubiquitäre Systeme, „Ambient Internet“ (4. Gen.)[2]**
- **....**

---

1 Ontologie ist ursprünglich eine philosophische Disziplin, neuerdings ein Modebegriff der Informatik. Nach Meyers Enzyklopädischem Lexikon ist Ontologie die „Lehre von dem Wesen und den Eigenschaften des Seienden”. In der Informatik ist sie die „formale Spezifikation eines bestimmten Gegenstandsbereichs in Form eines Begriffssystems”.

2 IPonAir: BMBF-Forschungsprogramm zur Konzipierung des „drahtlosen Internet der nächsten Generation“ (3. Gen.)

Kommunikationskosten

- **Kosten zum Senden/Empfangen** einer Nachricht enthalten drei wesentliche Komponenten
  - CPU-Kosten für Kommunikationsprotokoll
  - Signallaufzeiten für Übertragung des ersten Bits (Lichtgeschwindigkeit)
  - Übertragungsdauer für gesamte Nachricht aufgrund der vorhandenen Bandbreite des Kommunikationsmediums

	Shared Memory	LAN	WAN
typische Entfernung	< 10 m	1 km	10.000 km
CPU-Kosten pro SEND/RECEIVE	250 Instr.	2500 Instr.	25.000 Instr.
Signallaufzeit	0,1 µs	10 µs	100.000 µs
Bandbreite 1990 2002	1 Gbps 1 Gbps	10 Mbps 1 Gbps	50 Kbps 100 Mbps

- **Gesamtkosten für Nachricht von 1 KByte:**

CPU-Kapazität von 10 Mips (1990) bzw. 1000 Mips (2002)

	Shared Memory	LAN	WAN
CPU-Kosten 1990 2002			
Übertr.dauer 1990 2002			
Summe 1990			
Summe 2002			

Externspeicheranbindung

■ **gemeinsam:**

- lokale Rechneranordnung
- lose oder nahe Kopplung (***Shared-Disk; DB-Sharing***) bzw. enge Kopplung

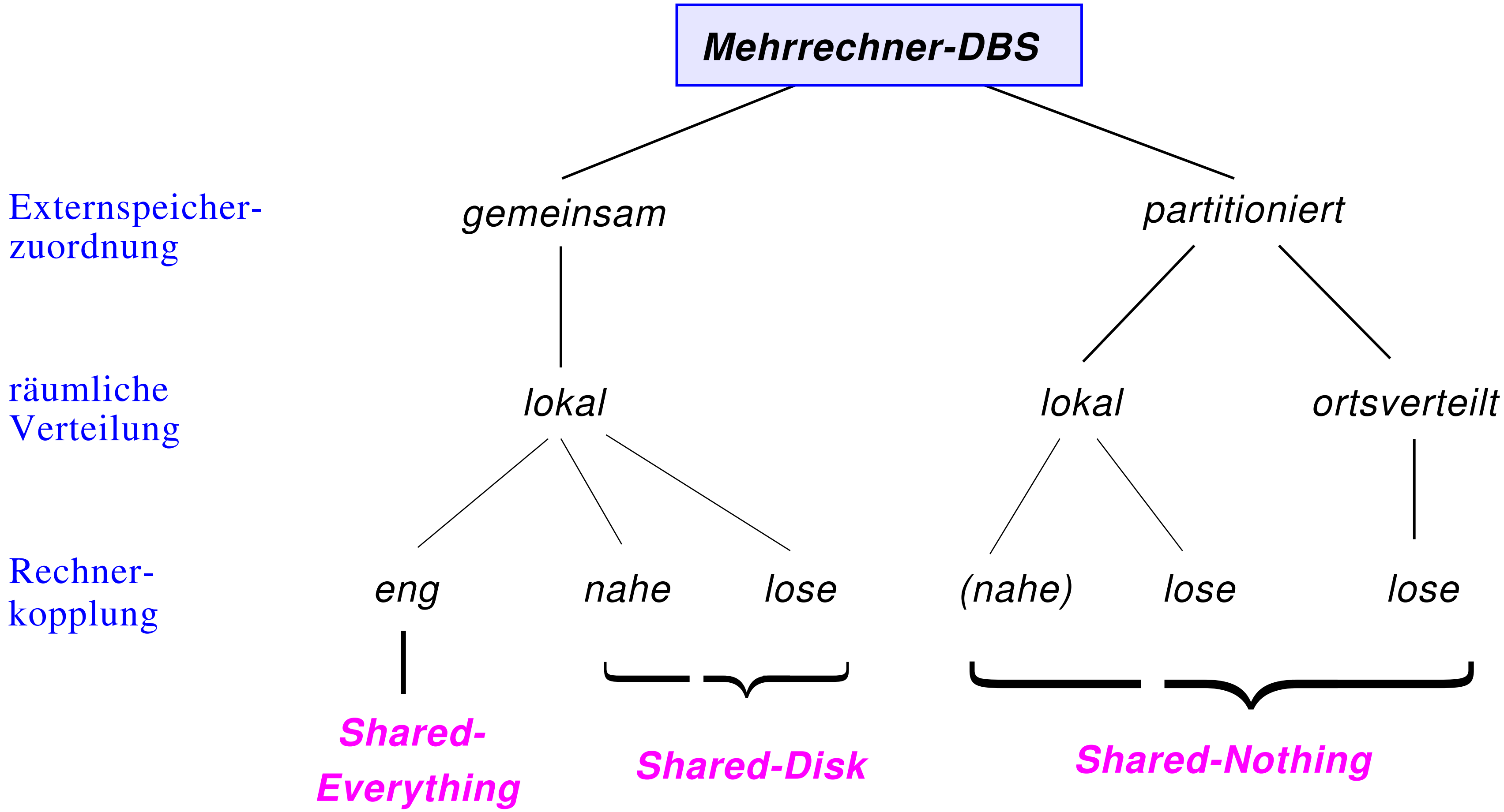
  \+ **jeder Prozessor kann alle Daten direkt erreichen** (hohes Potential zur Lastbalancierung)

  \+ **keine Partitionierung der Datenbank erforderlich**

- neue DB-Probleme bzgl. Synchronisation, Pufferverwaltung, Logging/Recovery, ...

■ **partitioniert:**
(Shared-Nothing; DB-Distribution)

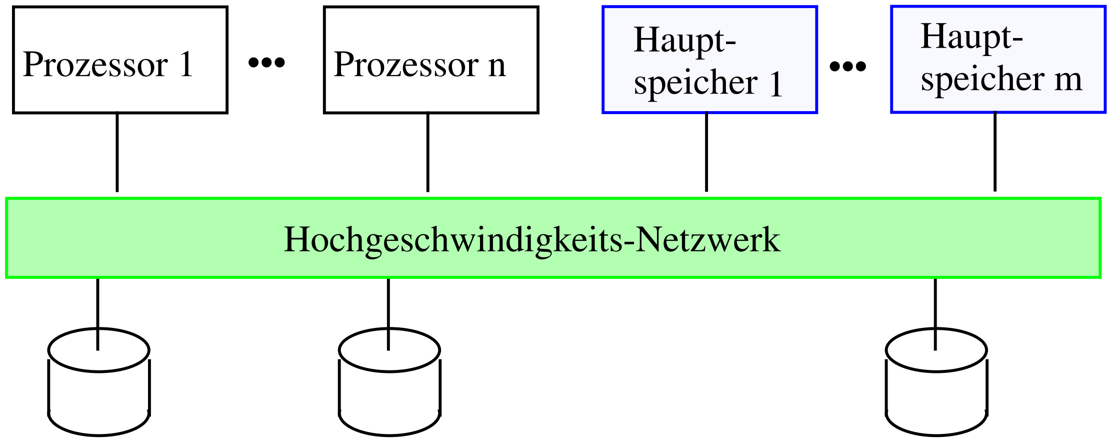
- lokale oder ortsverteilte Rechneranordnung
- i. allg. lose Rechnerkopplung
- (statische) Replikation der Daten möglich
- verteilte Transaktionsausführung, um auf entfernte Daten zuzugreifen



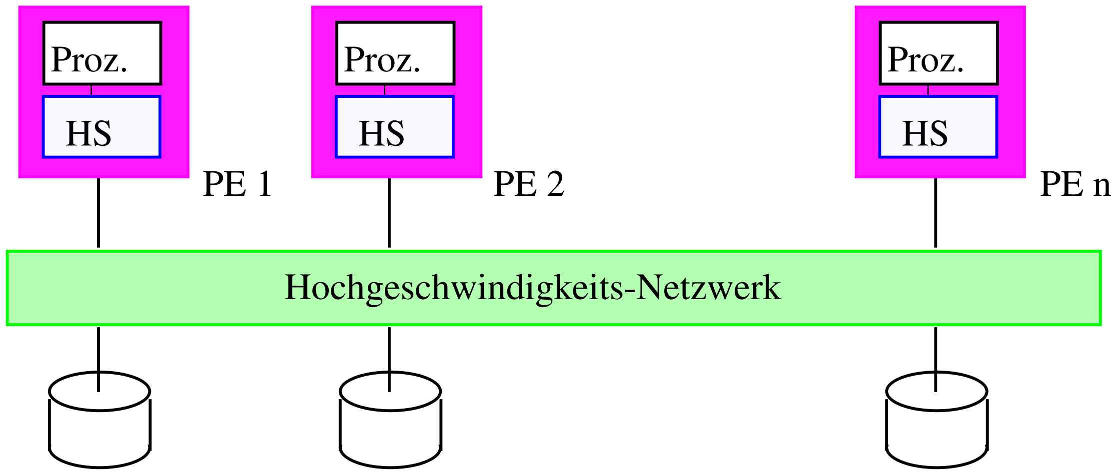
System-beispiele

Shared-Everything	Shared-Disk	Shared-Nothing
z.B. UDS/UTM, IMS/DC	IMS/VS Data Sharing	TANDEM NonStop SQL
auf eng gekoppelten Systemen	TPF	ORACLE/STAR
. . .	ORACLE	CICS-ISC
	Fujitsu AIM SRFC	UDS-D

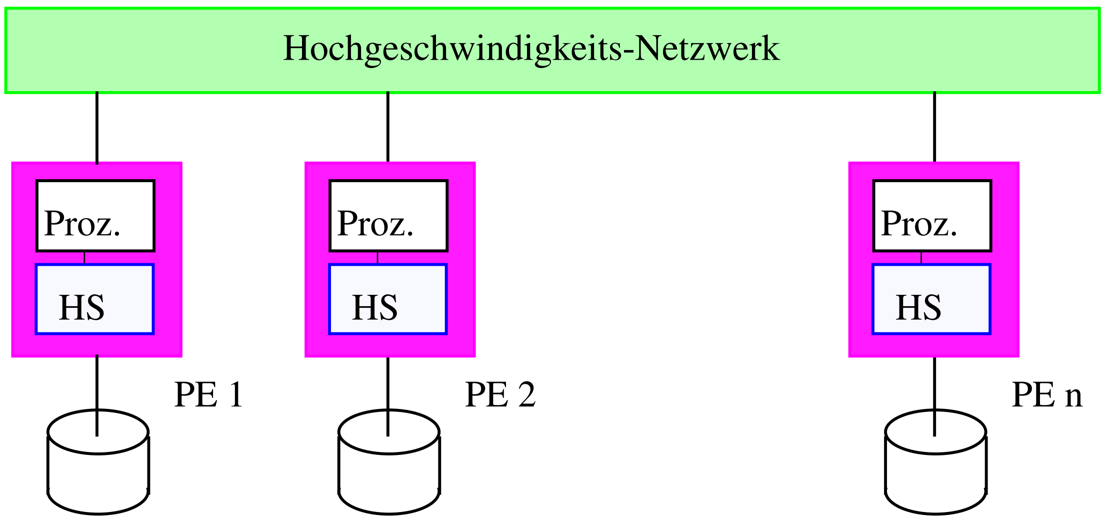
Parallele DBS



a) Shared-Everything (Symmetric Multiprocessing)

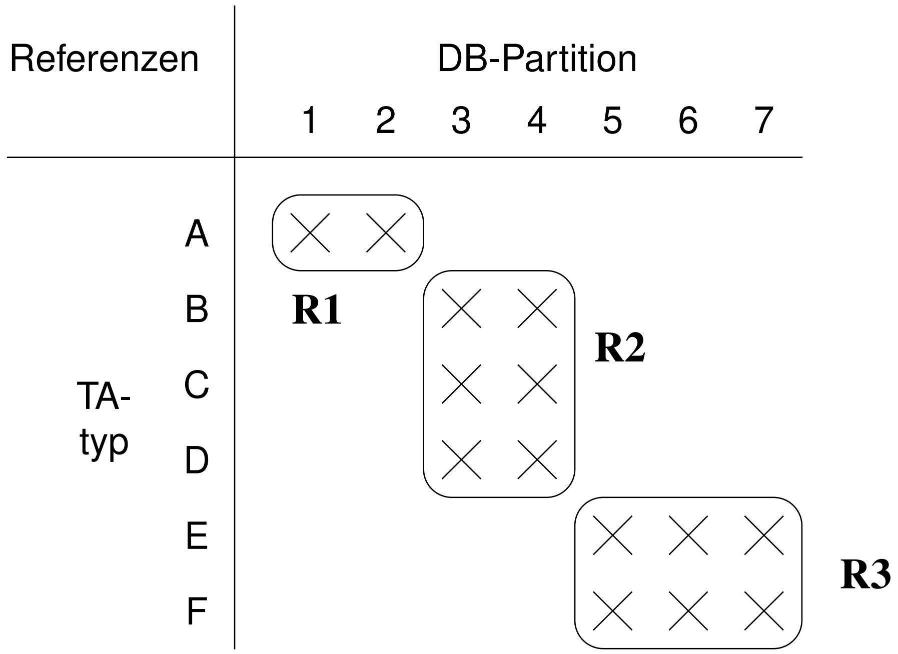


b) Shared-Disk (SD)



c) Shared-Nothing (SN)

Die “ideale” Last



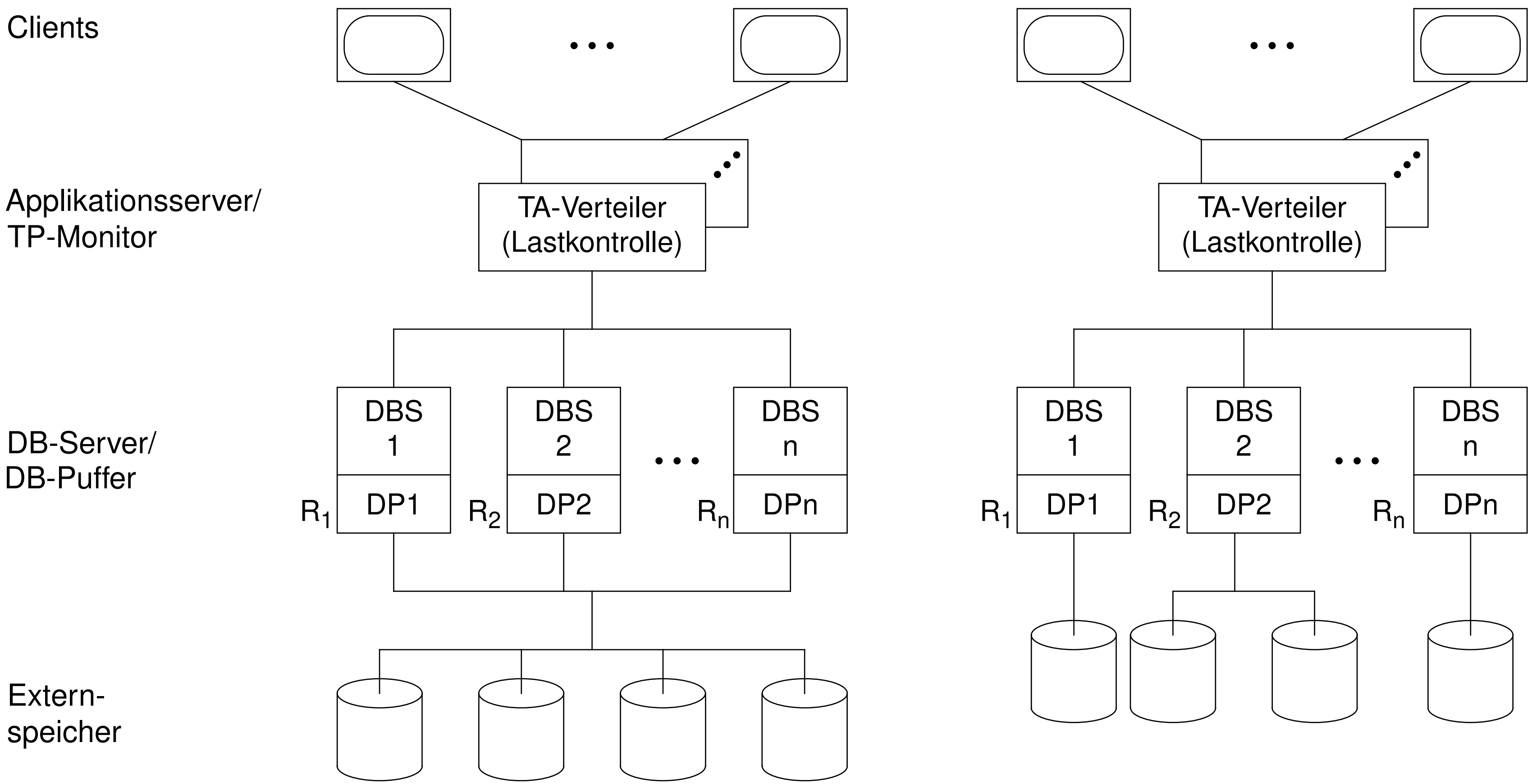
■ **Partitionierung von Last und Daten**

- Zuordnung von TA-Typen zu disjunkten Datenbereichen
- statisches Lastaufkommen
- gleichmäßige Verteilung der Last (Aufwand, Zeit)

■ **Delightful transactions**

- lokale Bearbeitung aller Transaktionen
  (z. B. Kontenbuchung)
- geringe Synchronisationsprobleme

Grobarchitektur Shared-Disk-System Shared-Nothing-System



Konfiguration:

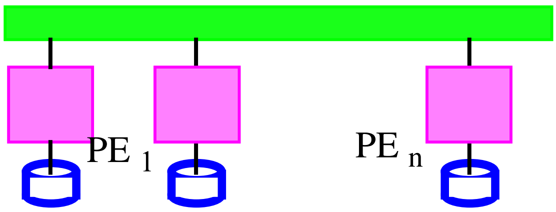
- **Rechneranzahl ?**
- **breitbandige Kommunikationsmöglichkeit/lokale Anordnung**
- **kein "single point of failure"**

SN vs. SD: Leistungsfähigkeit

■ **starke Abhängigkeiten zu**

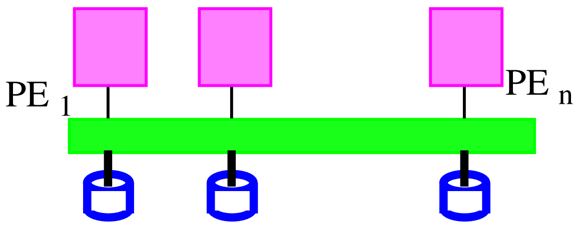
- Anwendungscharakteristika und
- Realisierung einzelner Systemfunktionen

■ **Shared-Nothing:**



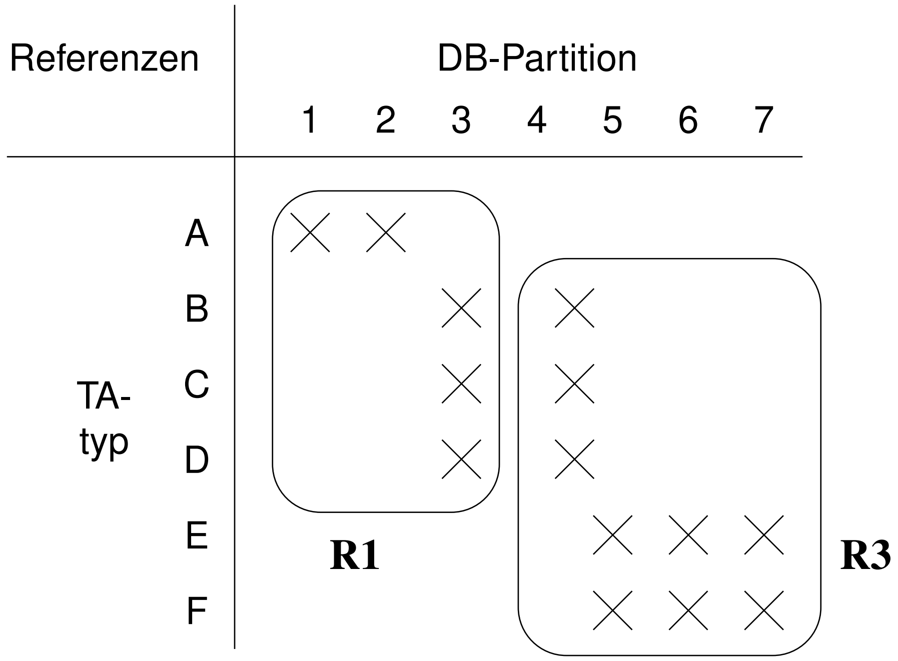
- statische Datenpartitionierung bestimmt Ausführungsort von DB-Operationen
- geringe Möglichkeiten zur Lastbalancierung oder Einsparung von Kommunikationsvorgängen
- **problematisch**: "dominierende" Transaktionstypen und DB-Bereiche

■ **Shared-Disk:**



- lokale Erreichbarkeit aller Daten:
  ➥ größere Möglichkeiten zur Lastbalancierung
- Kommunikation für Synchronisation und Kohärenzkontrolle (Behandlung von Pufferinvalidierungen)
- nahe Kopplung kann zur Leistungssteigerung eingesetzt werden
- höhere Flexibilität zur Parallelisierung

SN vs. SD: Verfügbarkeit



- **Shared-Nothing:**
  - Partition eines ausgefallenen Rechners nicht mehr erreichbar
  - Übernahme der betroffenen Partition durch anderen Rechner vorzusehen (Überlastungsgefahr)
  - Recovery durch übernehmenden Rechner
  - Replikation ermöglicht schnelle Katastrophen-Recovery

- **Shared-Disk:**
  - gesamte DB bleibt nach Rechnerausfall erreichbar;
    jeder einzelne Rechner oder alle zusammen können Last
    des ausgefallenen Rechners übernehmen
  - komplexe Crash-Recovery
  - Erstellung einer globalen Log-Datei

SN vs. SD: Erweiterbarkeit

■ **Modulares Wachstum** für Last (Transaktionen) und Daten

Referenzen	DB-Partition 1	2	3	4	5	6	7	8
TA-typ A	X	X						
B			X	X				
C			X	X				
D			X	X				
E					X	X	X	
F					X	X	X	
G	X					X		X
H				X				X

■ **Shared-Nothing:**

- neuer Rechner erfordert physische Neuaufteilung der DB (N -> N+1)
- einfache Plattenanbindung
- besonders problematisch für nicht-relationale DBS

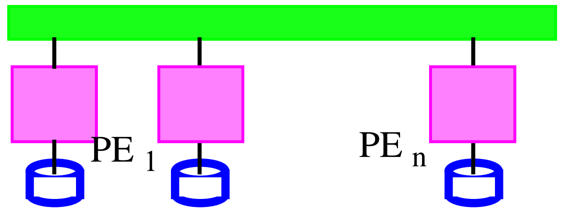
■ **Shared-Disk:**

- keine physische (Neu-) Aufteilung der DB
- Hinzunahme von Rechnern bleibt ohne Auswirkungen auf DB-Schema und Programme
- direkte Plattenanbindung kann Rechneranzahl begrenzen

➥ „nachrichtenbasierte" E/A-Schnittstelle

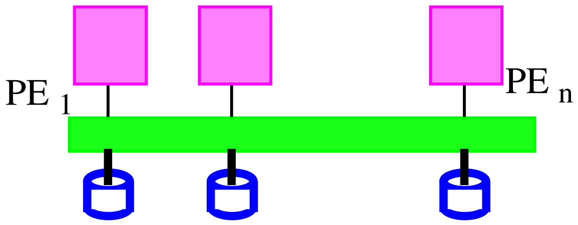
SN vs. SD: Technische Probleme

■ **Shared-Nothing:**



- Bestimmung der physischen DB-Partitionierung
  (Fragmentierung + Allokation)
- verteilte Anfrageverarbeitung (Optimierung)
- Behandlung replizierter Datenbanken
- verteiltes Commit-Protokoll
- globale Deadlock-Behandlung
- Lastverteilung, -balancierung
- Administration
- besondere Probleme in ortsverteilten Systemen
  (Netzwerkpartitionierungen, Knotenautonomie, ...)

■ **Shared-Disk:**

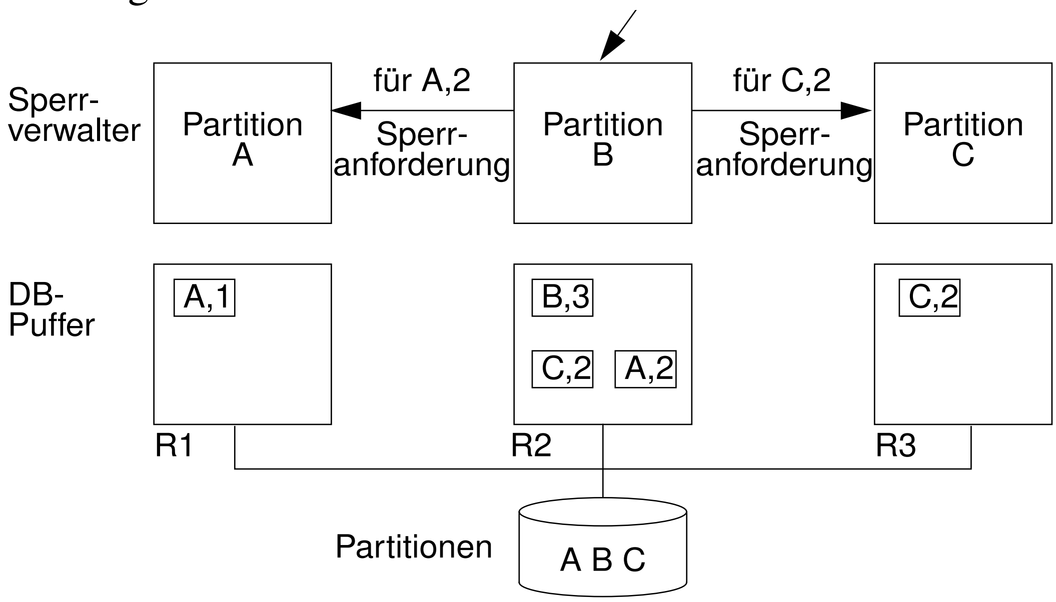


- Synchronisation
- globale Deadlock-Behandlung
- DB-Pufferverwaltung, Kohärenzkontrolle
- Logging, Recovery
- Lastverteilung, -balancierung
- parallele Anfrageverarbeitung
- Administration

Synchronisation

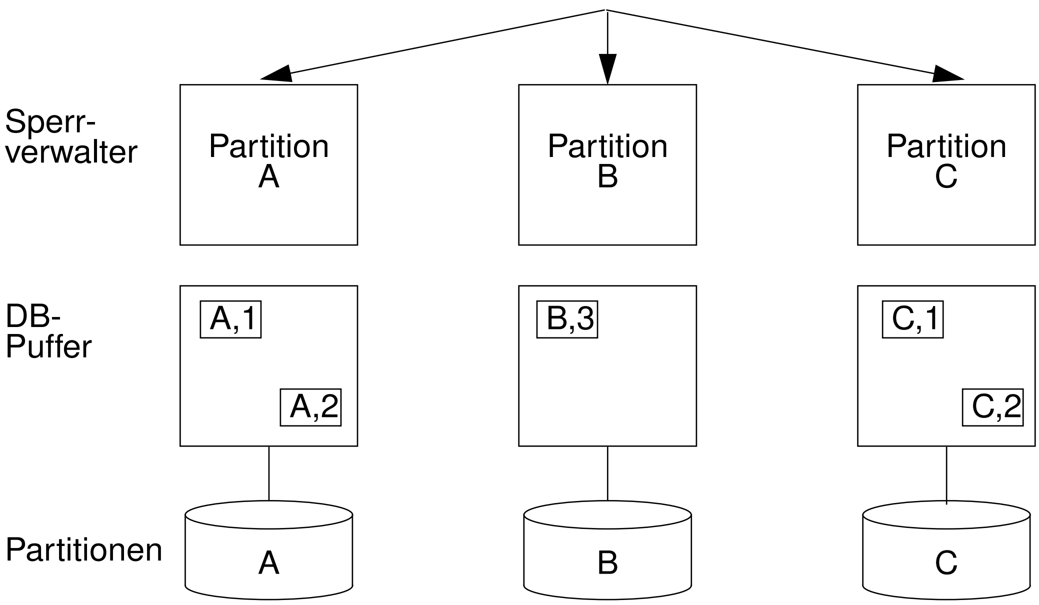
■ bei Shared-Disk

- logische und deshalb dynamische Zuordnung der DB-Partitionen und der Synchronisationsverantwortung
- Zuordnung der TA zu R2



■ bei Shared-Nothing

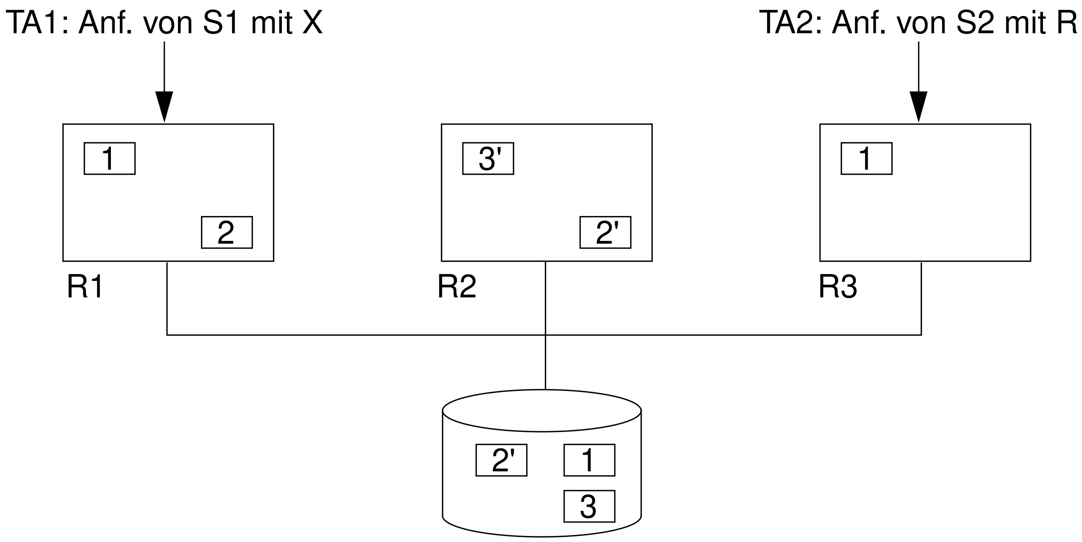
- statische Zuordnung der DB-Partitionen und der Synchronisationsverantwortung
- Verschicken von Operationen
- Sperren partitionsweise (wie im zentralen DBS)



DB-Pufferverwaltung

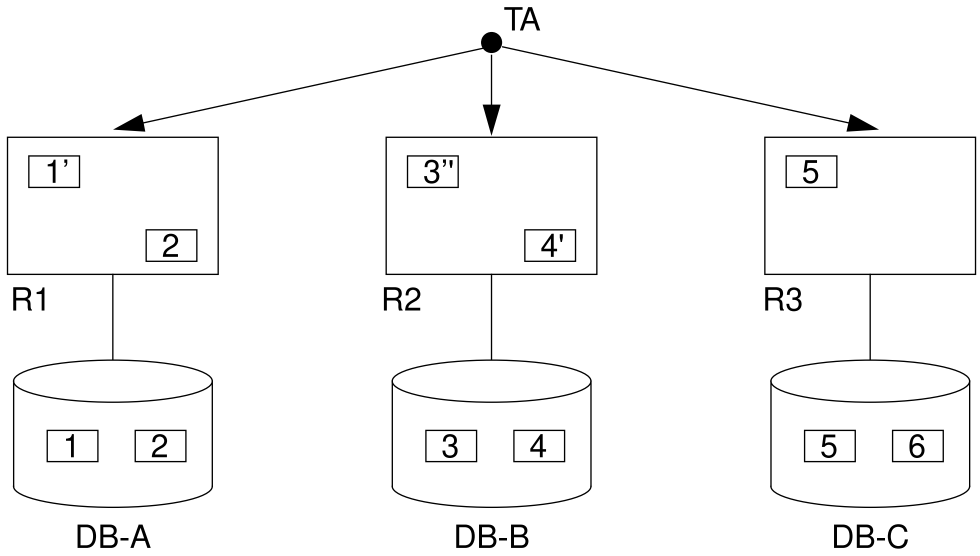
■ **bei Shared-Disk**

- Zuordnung neuer Transaktionen: Affinitätsproblem
- lokale Ausführung von DML-Befehlen
- Verarbeitungsprinzip: **Datentransport zum ausführenden Rechner**
- Kohärenzkontrolle erforderlich



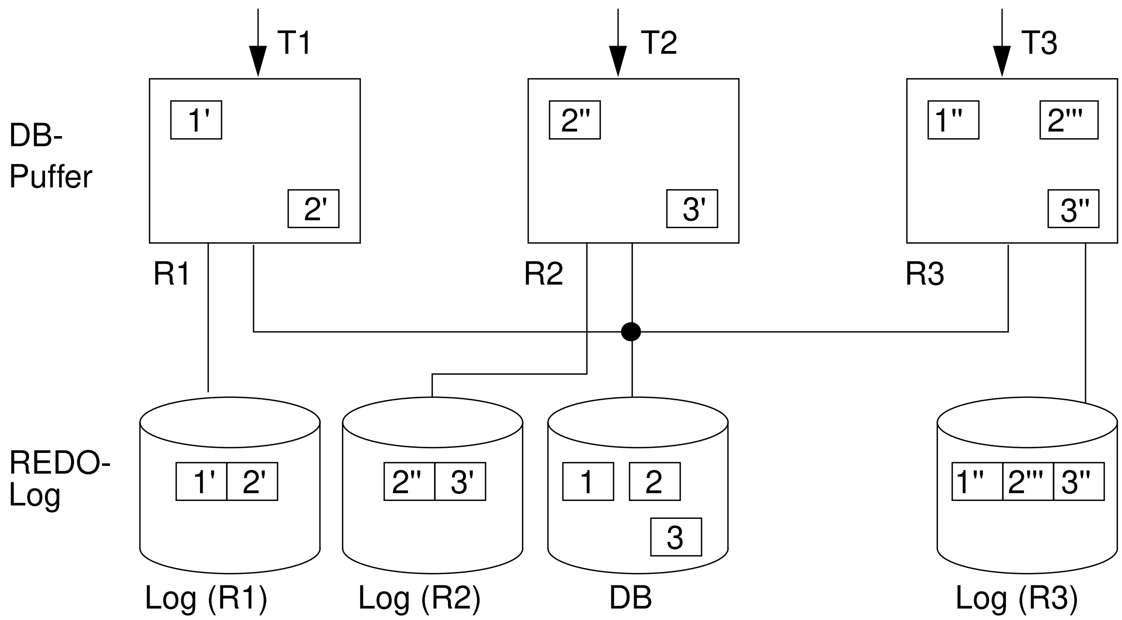
■ **bei Shared-Nothing**

- wie im zentralen DBS → keine Kopien
- Verarbeitungsprinzip: **die Last folgt den Daten**
- partitionsweise Verwaltung



Logging

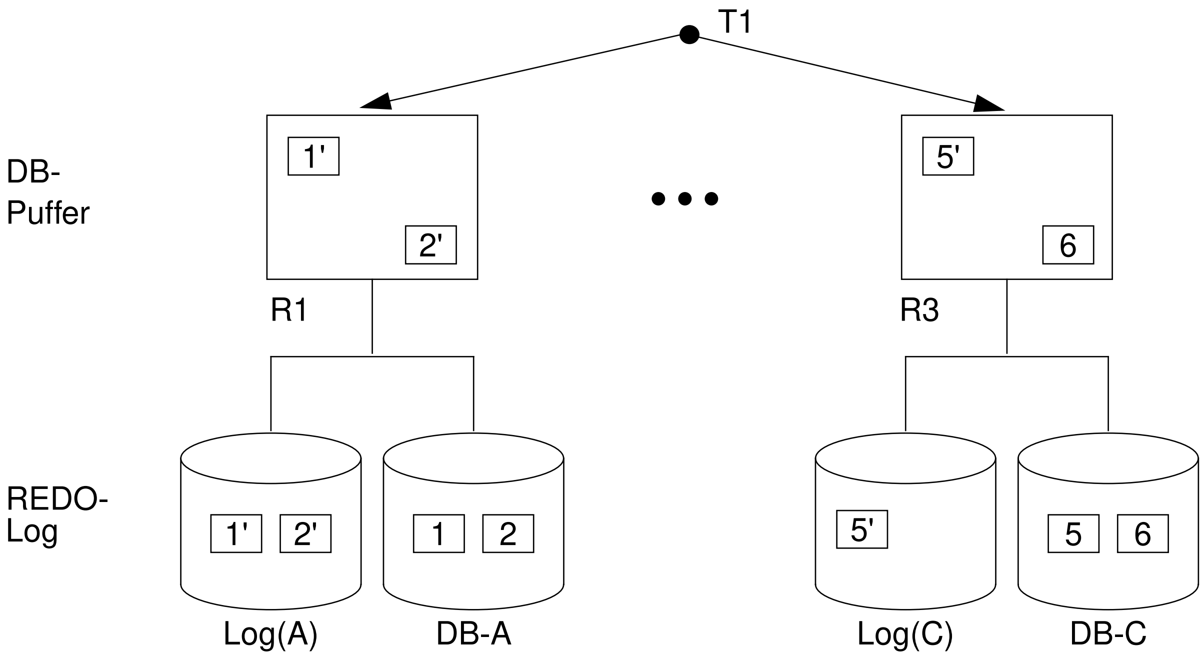
■ bei Shared-Disk



- T1 < T2 < T3
- globaler Log (Mischen) wegen Gerätefehler

■ bei Shared-Nothing

- partitionsweise (wie im zentralen Fall)
- Log-Dateien einer Transaktion können verteilt sein



Shared-Nothing vs. Shared-Disk

Kriterium	Shared-Nothing	Shared-Disk
Leistung		
Verfügbarkeit		
Erweiterbarkeit		
Ortstransparenz		
Heterogene Datenbanken		
Knotenautonomie		
geographische Verteilung		
Kosteneffektivität		
Administration		

PDBS-Architektur

■ **Hauptalternativen: SE, SD, SN sowie hybride Formen**

■ **Unterschiedliche Hardware-Plattformen möglich**

- **SMP**: Symmetric Multiprocessing[1]
  (alle Speicherzugriffe erfolgen über selben gemeinsamen Speicherbus)
- **NUMA**: Non-Uniform Memory Access: Sequent, Data General, ...
  (Speicherregionen sind über verschiedenartige physische Busse erreichbar: „lokaler und entfernter” Speicher))
- **Cluster**: VMS-Cluster, NT-Cluster, Sysplex
  (kurze Verbindungen, effiziente Inter-Prozeß-Kommunikation, geringe Fehlerraten, homogene Teilnehmer)
- **MPP** - Massively Parallel Processing: SP2, NCR, NCube, ...

■ **Wesentlich ist Sicht der DBVS-Instanzen**

- **SN:** Partitionierung der Datenbank unter den DBVS-Instanzen mit daraus abgeleiteter Anfrage- und Transaktionsverarbeitung
- **SE:** direkter Zugriff auf gesamte Datenbank für alle DBVS-Instanzen; zentrale Datenstrukturen (Sperrtabelle, DB-Puffer usw.), Datenaustausch über Haupt- oder Externspeicher
- **SD:** direkter Zugriff auf gesamte Datenbank für alle DBVS-Instanzen; lokale Puffer und Sperrtabellen, Datenaustausch über Externspeicher bzw. Nachrichten

■ **Hybride Lösungen mit erhöhter Komplexität**

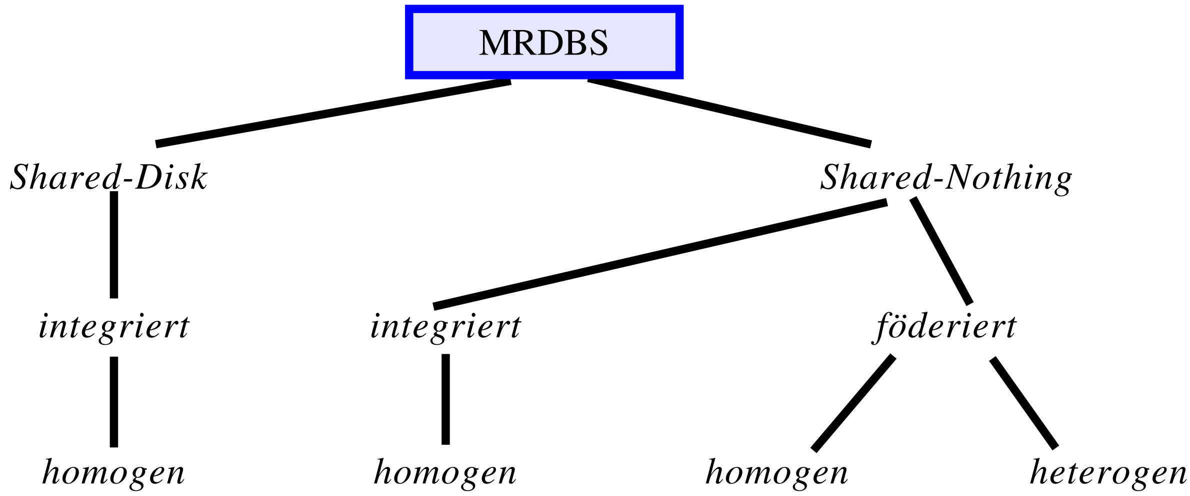
- SD- bzw- SN-artige Kopplung von SE-Systemen
- SN-artige Kopplung von SD-Systemen ...

---

1 auch SMMP: Shared Memory Multi-Processors

Integrierte vs. Föderierte Mehrrechner-DBS

■ Integrierte Mehrrechner-DBS



- **ein konzeptionelles DB-Schema**
- DB-Zugriff wie im zentralen Fall (Verteiltransparenz für AP)
- eingeschränkte Autonomie für beteiligte DBVS
- i. allg. identische DBVS-Instanzen (homogenes MRDBS)
- Bsp.: Verteilte DBS, Shared-Disk-DBS

■ Föderierte Mehrrechner-DBS

- weitgehend unabhängige DBVS mit privaten konzeptionellen DB-Schemata
- partielle Exportierung von Schemainformation für externe Zugriffe
- Heterogenität bei Datenmodellen und Transaktionsverwaltung möglich
- Probleme mit semantischer Heterogenität
- Verteilungstransparenz i. allg. nur bedingt erreichbar

Nutzung heterogener autonomer Datenbestände

■ **Anwendungsintegration**

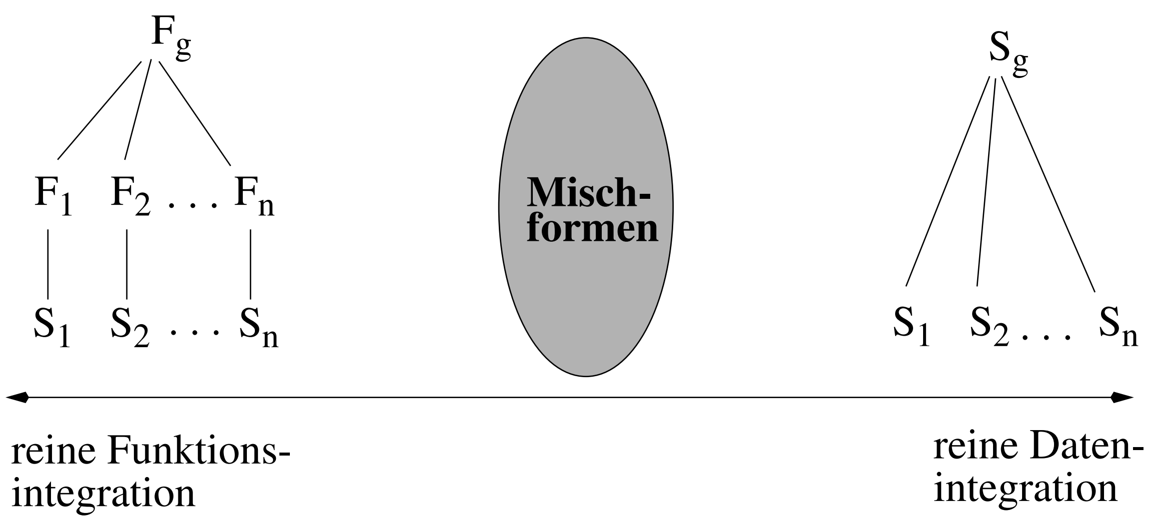
- DB-Zugriffe innerhalb von **Anwendungsfunktionen gekapselt**
- Zugriff auf entfernte Datenbanken durch Aufruf von **Anwendungsfunktionen** / Web Services
- Nutzung standardisierter Kommunikationsprotokolle und Nachrichtenformate (z.B. SOAP /XML)
- Unterstützung verteilter Transaktionen durch **Applikations-Server / TP-Monitor**
- gegenüber Datenintegration einfachere Realisierung, jedoch geringere Flexibilität und Funktionalität (keine übergreifenden DB-Operationen wie Joins, keine Ad-hoc-Anfragen usw.)

■ **Datenintegration**

- **globale Sicht** auf verteilten Datenbestand
- **unterschiedliche Stufen** der Integration mit partieller bis vollständiger Transparenz
- föderierte DBS (FDBS) / Mediator-Systeme, VDBS, Data Warehouses

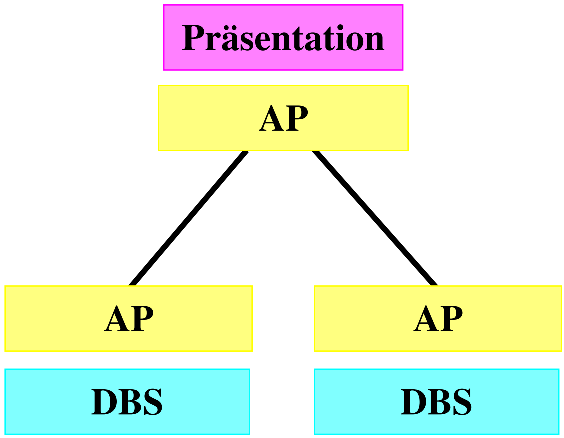
■ **Integrationsspektrum**

Zugriff über globale Funktion F_g (vordefiniert, zusammengesetzt aus F_i)
vs. Zugriff über globales Schema S_g (mit deklarativen Anfragen, generisch)



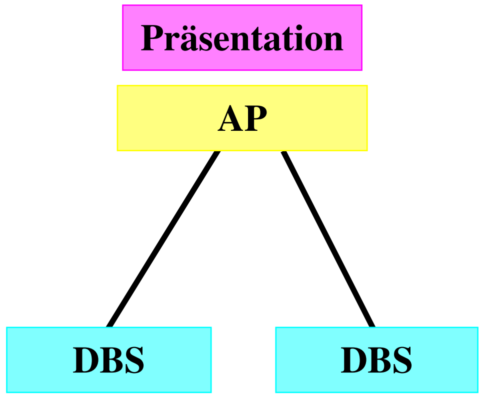
Integrationsmöglichkeiten bei autonomen und heterogenen DBS

Programmierte Verteilung



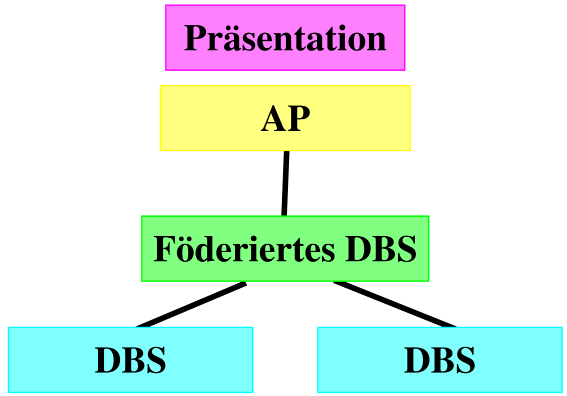
- **DB-Zugriff über Aufruf vordefinierter Programmfragmente bzw. von Prozeduren/Methoden**
- **Realisierungsformen:**
  - verteilte Transaktionssysteme (TP-Monitore)
  - verteilte OO-Systeme

Verteilung von DB-Operationen



- **Aufrufgranulat: DB-Operation (SQL-Anweisung)**
- **DB-Schemata müssen den AP bekannt sein**
- **Client-DBS kann AP-Rolle übernehmen**

Föderierte DBS



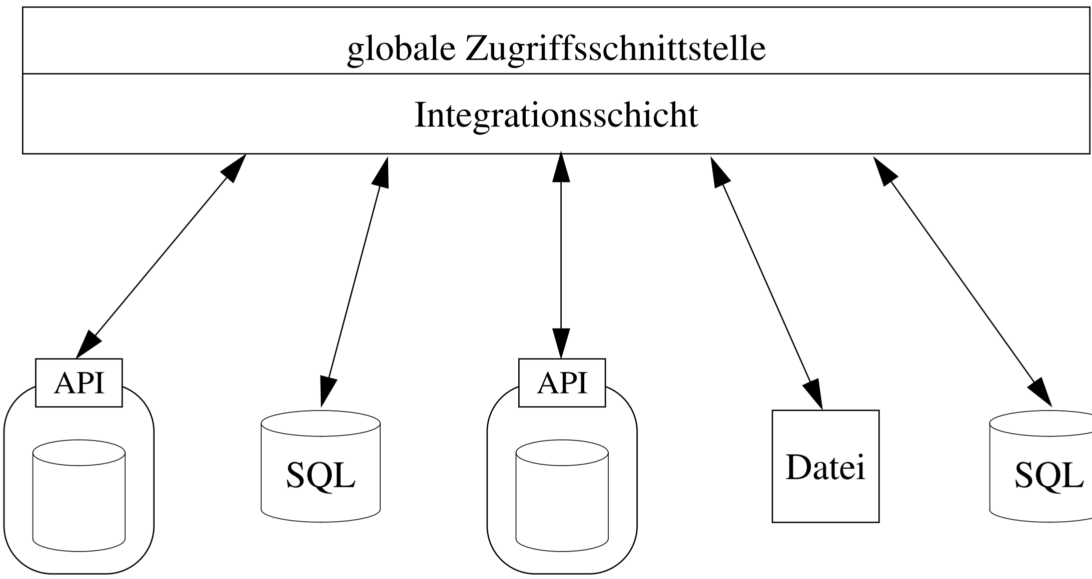
- **FDBS kann vereinheitlichte DB-Sicht unterstützen**
- **verteilte Ausführung einer DB-Operation**

Architekturvorschlag zur Funktions- und Datenintegration[1]

■ **Daten aus verschiedenartigen Quellen müssen gemeinsam verarbeitet werden**

- Datenbanksysteme mit generischen Zugriffsmöglichkeiten (SQL)
- Dateisysteme mit einfachen Zugriffsoperationen
- Anwendungssysteme (AWS) kapseln DB und Anwendung
  - Zugriff nur über API mit vordefinierten Funktionen möglich
  - keine DB-Schnittstelle verfügbar
- Standard-SW (R3, Produktdaten-Managementsystem (PDMS) wie Metaphase) sind solche AWS
  - Logik größtenteils in der Anwendung realisiert
  - Funktionalität des DBVS wird nicht ausgeschöpft

■ **Abstraktes Integrationszenario**

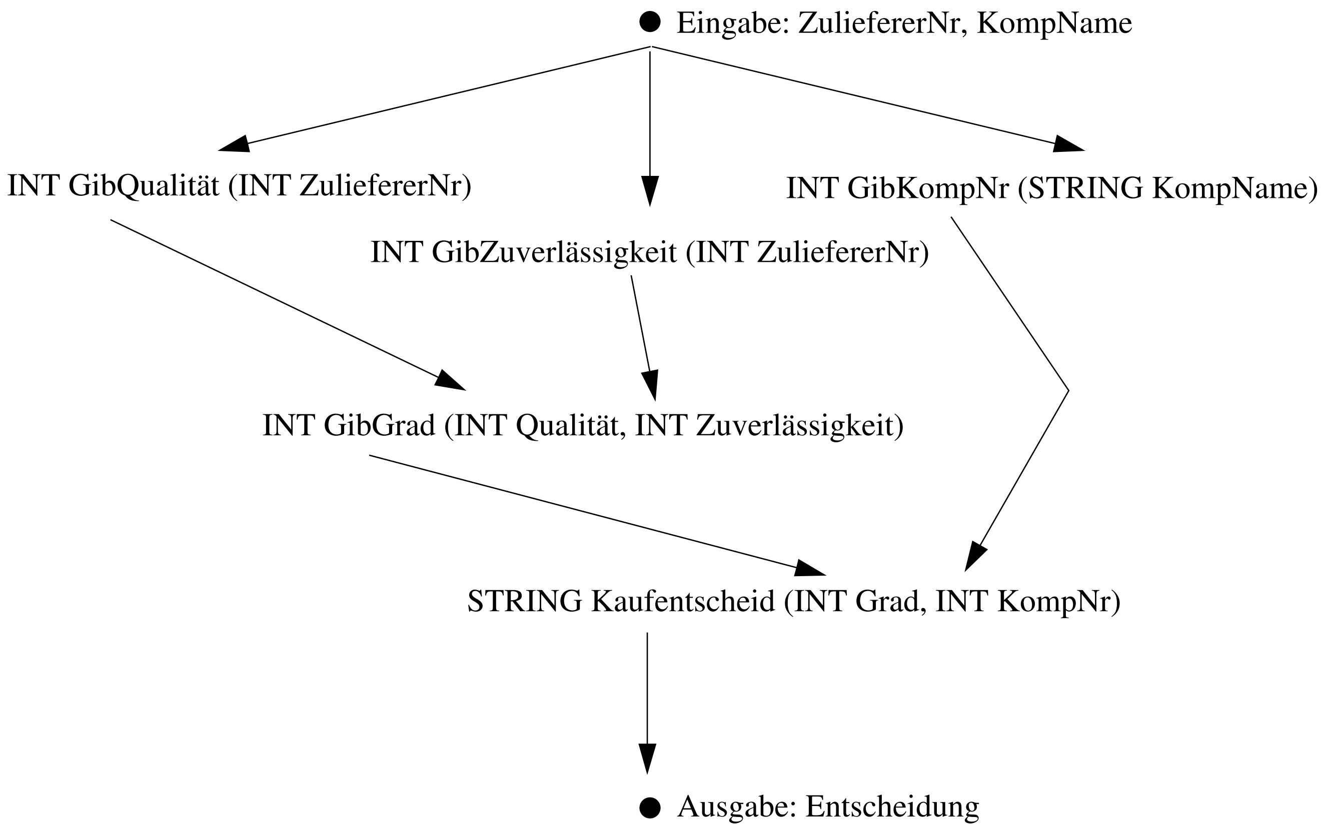


- reine Datenintegration nicht mehr ausreichend
- Integration des Zugriffs auf vordefinierte Funktionen erforderlich

1 Hergula, K., Härder, T.: Coupling of FDBS and WfMS for Integrating Database and Application Systems: Architecture, Complexity, Performance, in: Proc. 8th Int. Conf. on Extending Database Technology (EDBT'2002), Prague, March 2002, pp. 372-389.

Architekturvorschlag (2)

- **Oft „manuelle“ Integration mehrerer AWS erforderlich**
  - Scenario: Einkaufsabteilung eines Unternehmens
    - Mitarbeiter wird durch Funktion Kaufentscheid unterstützt
    - Parameter hängen vom Qualitäts- und Zuverlässigkeitsgrad und der Nummer der Komponente ab
    - Einsatz von Funktionen verschiedener AWS erforderlich (GibQualität von Lagerhaltungssystem, GibZuverlässigkeit von Einkaufssystem, GibKompNr von PDMS)

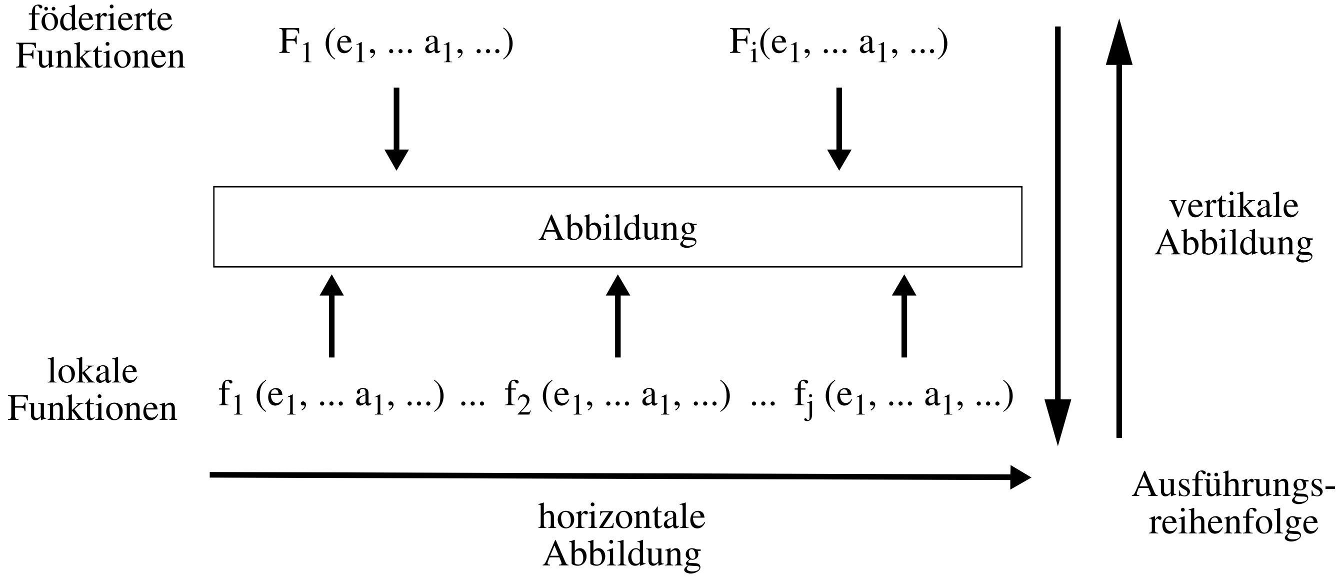


➥ **Ablauf der einzelnen Schritte des Benutzers bei „manueller“ Integration**

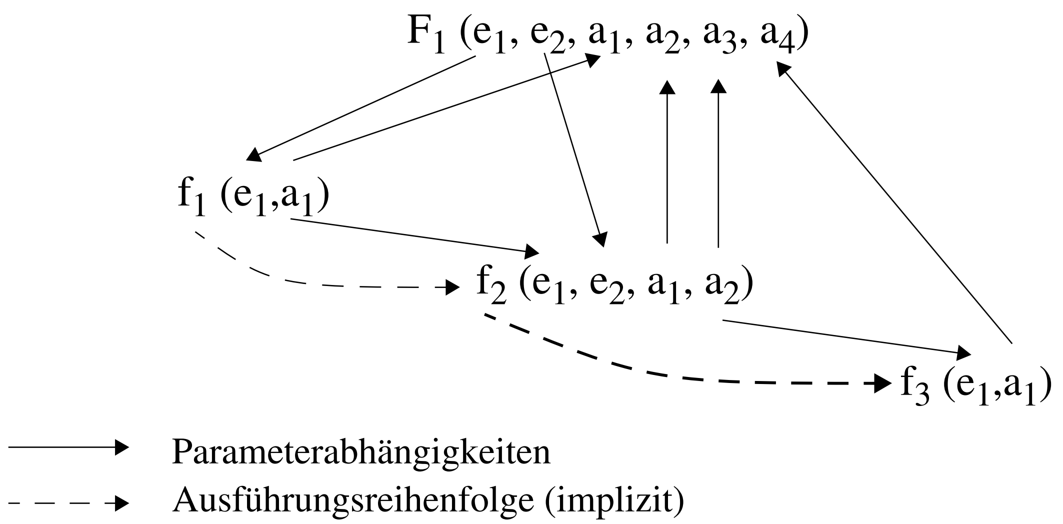
Architekturvorschlag (3)

■ Automatisierung der Funktionsintegration

- Abbildungsmodell: Immer wiederkehrende Sequenzen von (lokalen) Funktionsaufrufen sollen hinter einer föderierten Funktion verborgen werden
- Ausgangslage



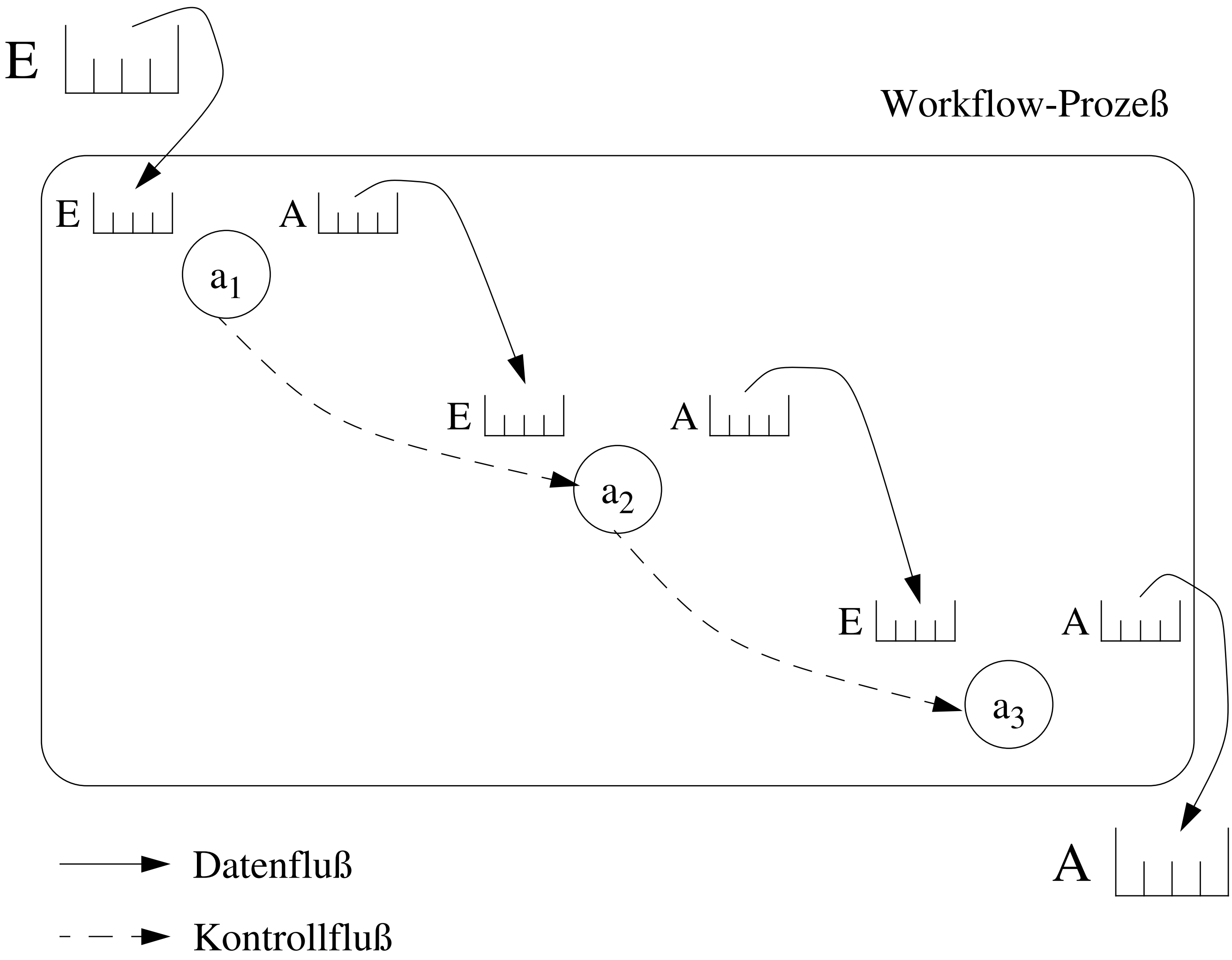
- Beschreibung durch Abhängigkeiten



➥ **gerichteter, azyklischer Graph**

Architekturvorschlag (4)

■ **Prozeßmodell eines Workflows**

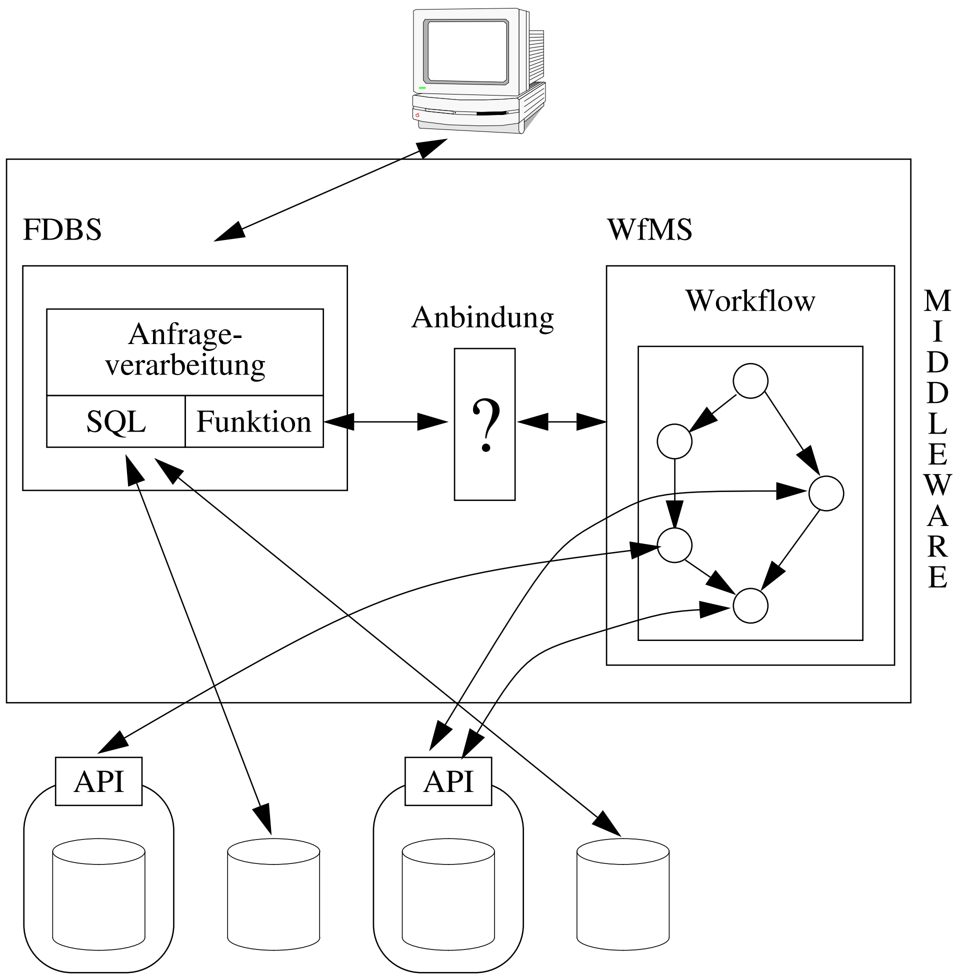


- Übereinstimmungen
  - gerichteter, azyklischer Graph
  - Ausführungsreihenfolge ≡ Kontrollfluß
  - Abhängigkeiten ≡ Datenfluß
  - Funktionen ≡ Aktivitäten

Architekturvorschlag (5)

■ **Vision: Kombination von Daten- und Funktionsintegration**

- FDBS integriert bereits verschiedene Datenquellen (SQL)



- Anbindung eines WfMS erforderlich
- Vorteile beim Einsatz eines Workflows zur Realisierung einer föderierten Funktion
  - Ausführendes System (WfMS) ist bereits vorhanden
  - Mit WfMS können ohne Modifikation des FDBS sehr komplexe AW-Szenarien umgesetzt werden
  - WfMS ermöglicht eine **generische** Realisierung **föderierter Funktionen**
  - Es bietet dem FDBS **transparenten Zugriff** zu heterogenen Systemen
  - Es verfügt über Schnittstellen zu vielen der zu integrierenden Quellsysteme und realisiert somit eine verteilte Programmierung über heterogene AW hinweg
  - Unterstützung der Wf-Erstellung und -Wartung durch Build-Time-Komponente des WfMS (Visualisierung des Wf-Prozesses als Graph)

Architekturvorschlag (6)

■ **Anbindungsmodell**

- FDBS muß erkennen, welche Teile einer Anfrage vom FDBS und welche vom WfMS auszuführen sind
- Übergabe von Funktionsaufrufen und Übernahme von (Zwischen-) Ergebnissen als **abstrakte Tabelle**

■ **Verfügbare Anbindungsmechanismen**

- **Wrapper** unterstützen die Anbindung von SQL- sowie Nicht-SQL-Quellen
  - **SQL/MED** definiert Wrapper als einen Mechanismus, mit dem ein SQL-Server auf externe Daten zugreifen kann, die von fremden Servern verwaltet werden
  - Pro Server-Typ gibt es einen Wrapper
  - Externe Daten werden dem FDBS in Form von fremden Tabellen bekannt gemacht
  - Wrapper stellt nicht nur eine Schnittstelle dar, sondern kann auch Funktionalität des fremden Servers erweitern

➥ **Nutzung zur Anfrageoptimierung**

  - Zugriff auf föderierte Funktion

```
Select   Entscheidung
From     KaufeKomponente
Where    KompName = 'xyz' And ZuliefererNr = 1234
```

Architekturvorschlag (7)

- **Verfügbare Anbindungsmechanismen** (Forts.)
  - Benutzerdefinierte Tabellenfunktionen (**UDTF**) erlauben die Einbindung von Nicht-SQL-Quellen
    - UDTF sind externe Funktionen, die eine Tabelle statt einem skalaren Wert zurückliefern
    - Sie unterstützen nur lesenden Zugriff
    - Sie liefern keinen Beitrag zur Anfrageoptimierung
    - Beispiel (nach DB2)

```
Select   Entscheidung
From     Table KaufeKomponente ('xyz', 1234) AS KK
```

  - **Gespeicherte Prozeduren**
    - erlauben die Umsetzung von lesendem und schreibenden Zugriff
    - müssen jedoch mit einer expliziten Call-Anweisung gestartet werden

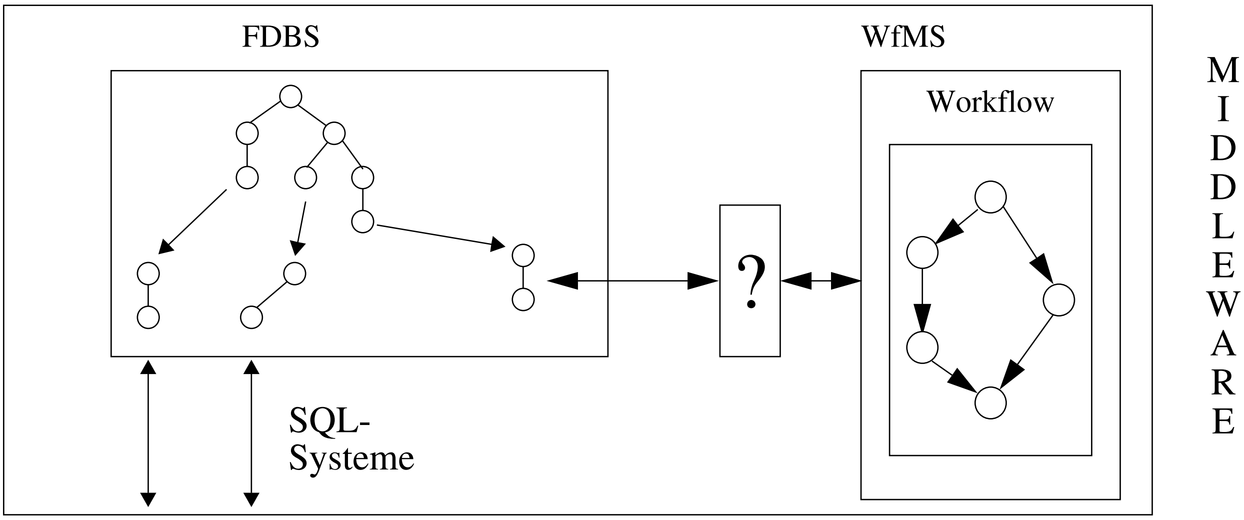
```
Call KaufeKomponente ('xyz', 1234)
```

➥ **kein Zugriff auf Daten und Funktionen in einer Select-Anweisung möglich!**

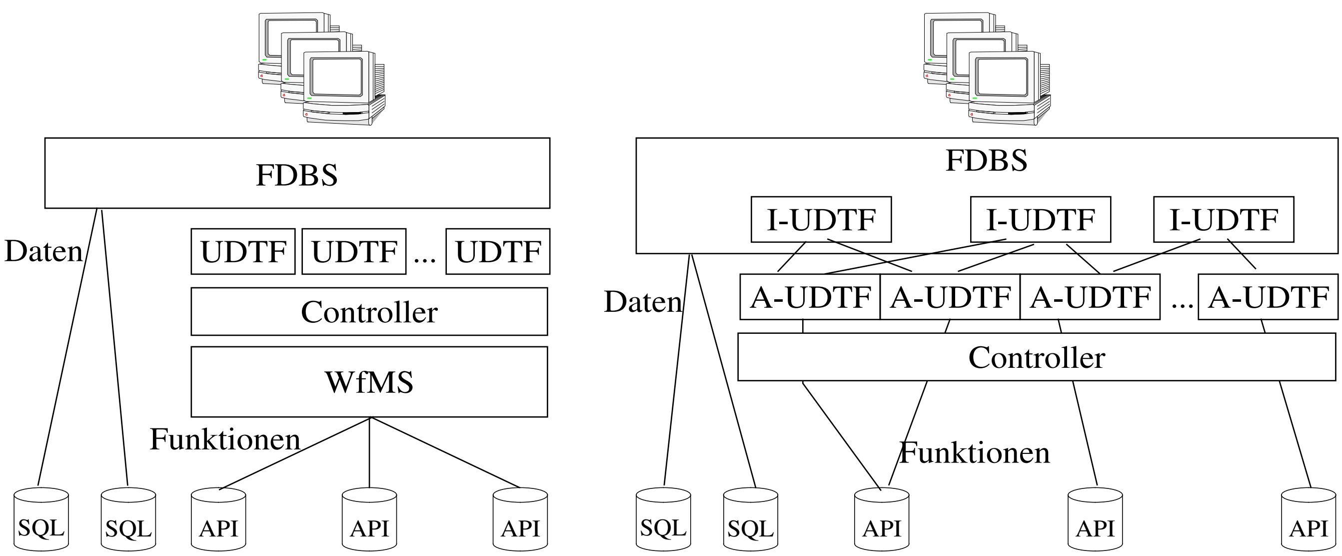
Architekturvorschlag (8)

■ Ausführungsmodell

- FDBS: Anfrageoptimierung und -auswertung mit Hilfe von Operatorgraphen, dynamisch
- WfMS: Ausführung definierter Workflow-Prozesse, statisch



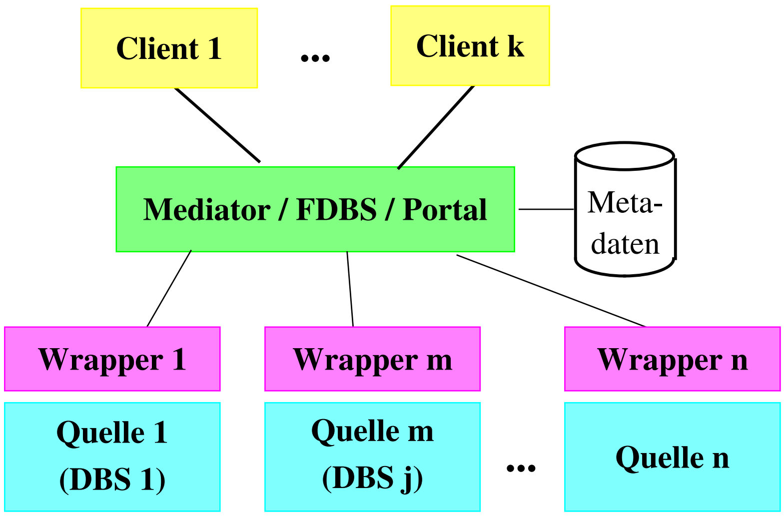
- Verbindung FDBS - WfMS
  - Wrapper: größte Flexibilität und bestes Optimierungspotential
  - UDTF (im Prototyp benutzt, da Wrapper noch nicht verfügbar)
- ohne WfMS-Einsatz
  - A-UDTF (Access-UDTF) zum Zugriff auf **eine** Datenquelle
  - I-UDTF (Integration UDTF) zur Realisierung der Integrationslogik im FDBS
- Prototyp-Architekturen mit und ohne WfMS



Weitere Alternativen bei der Datenintegration

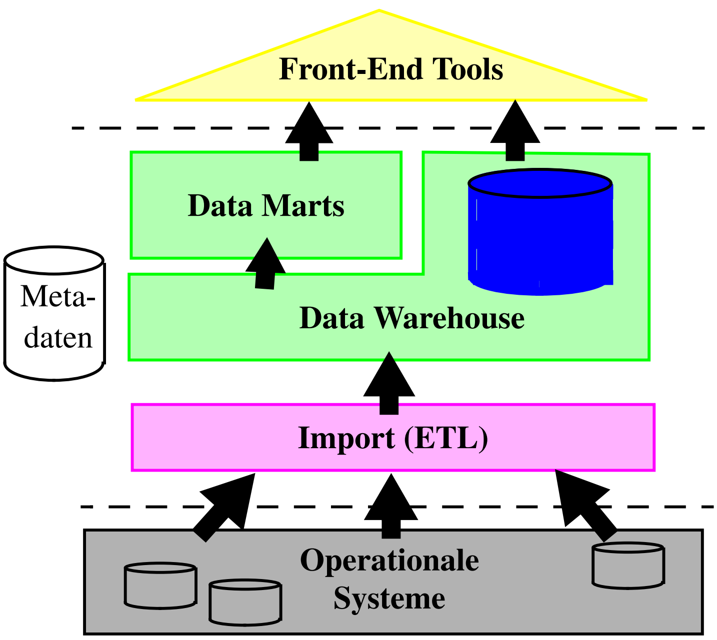
Virtuelle Integration

**(Mediator/Wrapper-Architekturen,
Föderierte DBS)**



Physische (Vor-) Integration

(Data Warehousing)



ETL: Extract, Transform, Load

ObjectGlobe - Allgegenwärtige Anfrageverarbeitung im Internet[1]

■ **Ziele**

- Bereitstellung einer **Infrastruktur**, welche die Verteilung von Mechanismen zur Anfrageverarbeitung (Anfrageoperatoren) so einfach gestaltet wie das Publizieren von Daten und Dokumenten im Web
- Clients sollen in die Lage versetzt werden, komplexe Anfragen zu verarbeiten, die eine Ausführung von Operatoren **unterschiedlicher** Provider von **verschiedenen** Web-Sites und einen Zugriff auf Daten und Dokumente von **verschiedenartigen** Datenquellen einschließen

➥ Anfrageverarbeitung unter Einsatz von Anfrageoperatoren (von **Funktions-Providern**), von **Zyklen-Providern** (Rechner) und von Datenquellen (bereitgestellt von **Daten-Providern**), die alle unabhängig voneinander sind

■ **Voraussetzugen**

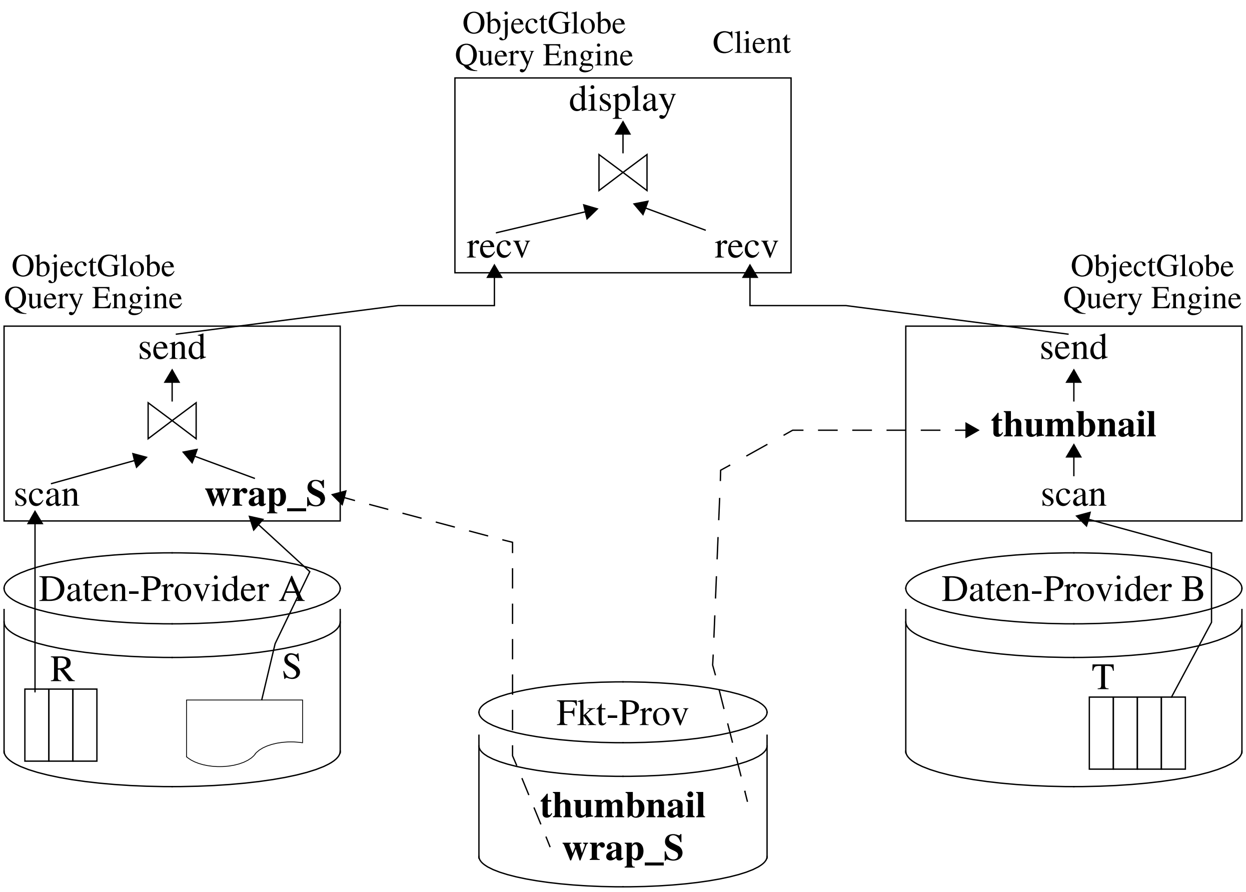
- Anfrageoperatoren können in verteilten Anfrageplänen ablaufen
- Sie lassen sich zu beliebigen Web-Sites verschieben, beispielsweise zu solchen, die **„in der Nähe“** der Datenquellen sind
- Verteilte Anfragepläne können aus **beliebigen Anfrageoperatoren** zusammengesetzt werden
- Anfrageoperatoren (in Java) entsprechen den sicheren Schnittstellen von ObjektGlobe

---

1 Braumandl, R. et al.: ObjectGlobe: ubiquitous Query Processing on the Internet, VLDB Journal 10:1, Aug. 2001, pp. 48-71.

ObjectGlobe (2)

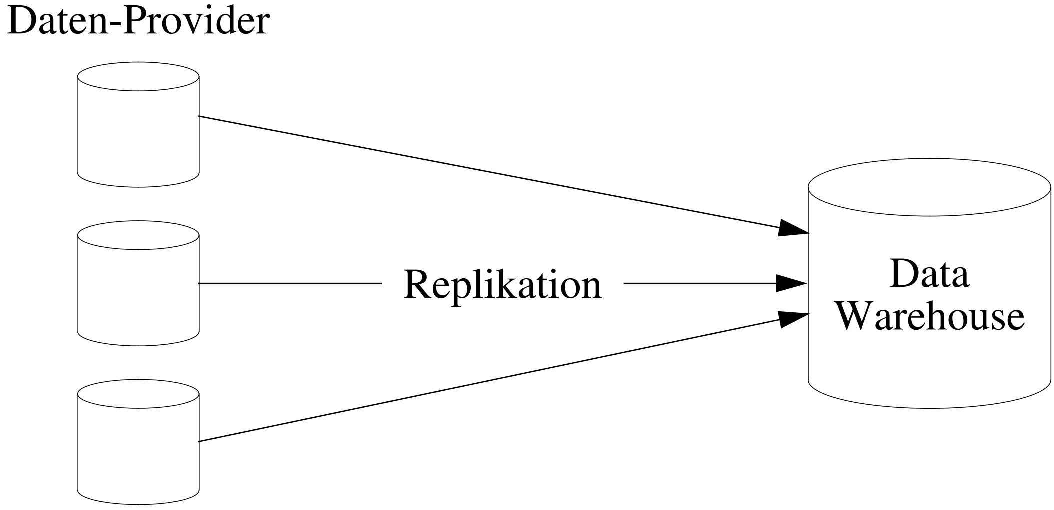
- **Eigenschaften**
  - ObjectGlobe ist ein **föderiertes System**
  - Es verkörpert einen **verteilten und offenen** Anfrage-Prozessor (Query Engine) für Internet-Datenquellen
  - **Metadaten-Repository** (Lookup-Service) registriert alle benutzbaren Datenquellen, Operatoren und Rechner
  - Optimierer generiert einen Plan zur Anfrageausführung (QEP: query evaluation plan), der nach Möglichkeit den Qualitätsanforderungen des Benutzers genügt
  - **Transparente Ad-hoc-Integration** von Operatoren und Funktionen stellt Herausforderung für Anfrageoptimierung dar
  - Integration heterogener Daten ist kein zentrales Thema. Daten liegen in einem **Standardformat** vor (relational, XML) oder sind durch Wrapper gekapselt
  - **Meta-Schema** beschreibt alle relevanten Eigenschaften aller Dienste (Struktur der Daten, Zugriffsmethoden, Statistiken, Kosten)



E-Marktplatz - Anwendung von ObjectGlobe

■ Herkömmliche Technik

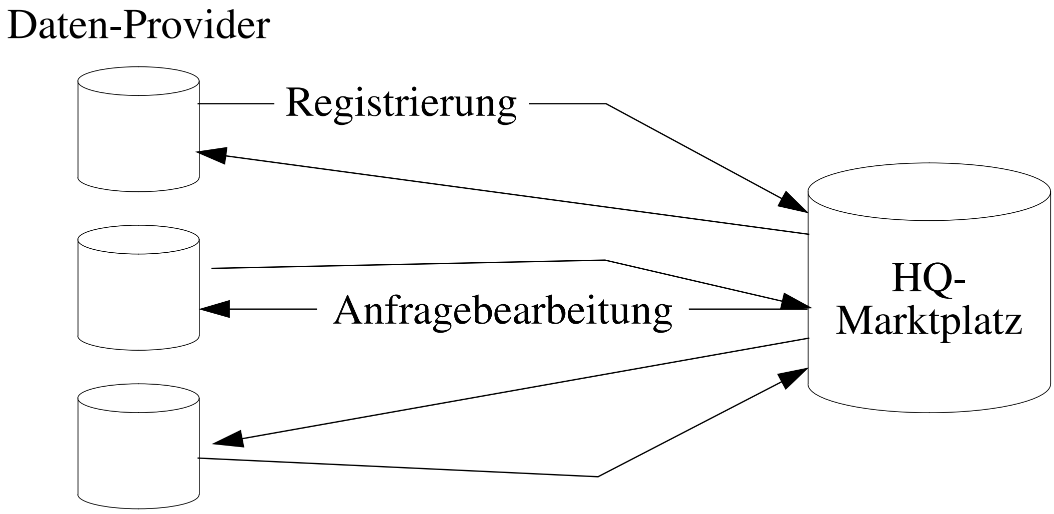
- Replikation von Daten in einem zentralen Data Warehouse



- Probleme bei der Schemaintegration
- eingeschränkte Operatoren, ...

■ HyperQuery-Ansatz

- Marktplatz wird „Vermittler“

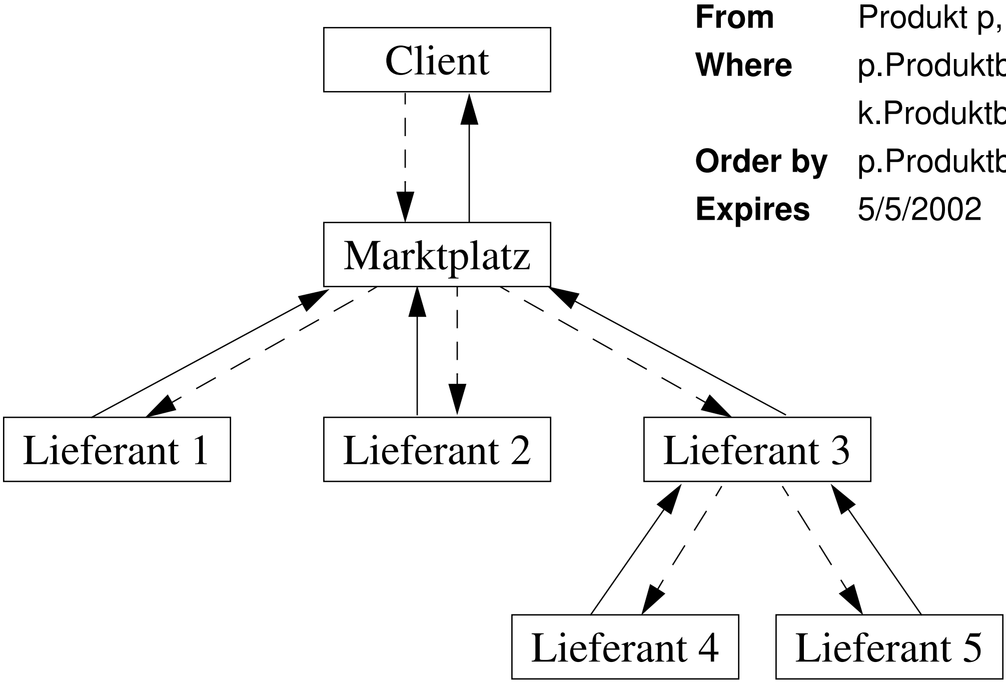


- Verteilung der Anfragebearbeitung entlang des impliziten Allokationsschemas der Daten
- Objekte und Anfragen „fließen“ durch das WWW
- Skalierbarkeit

E-Marktplatz (2)

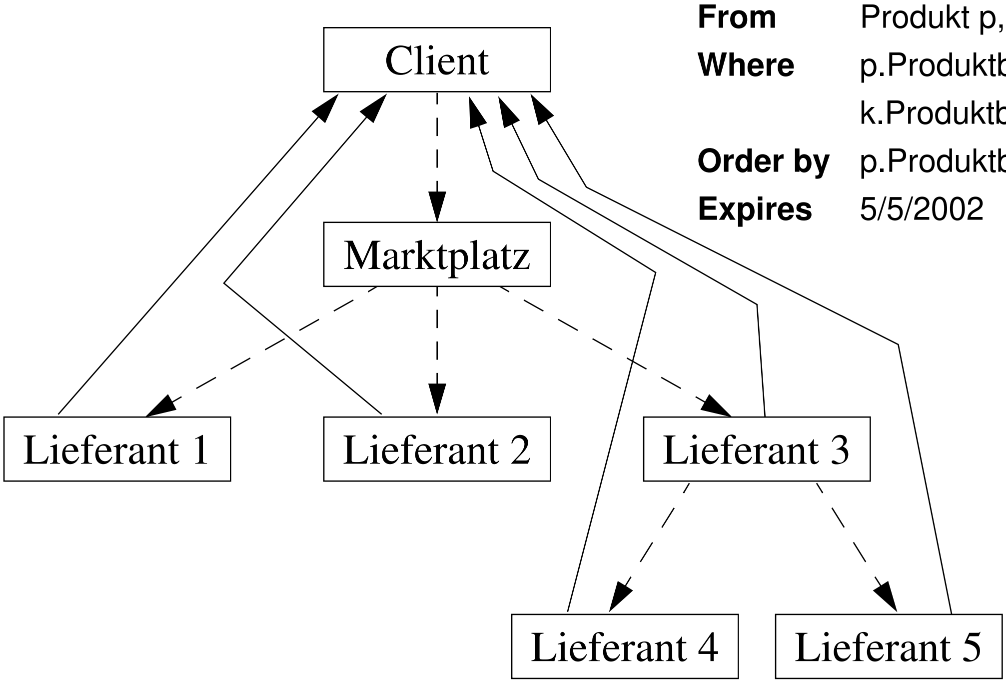
- **Ausführungsmodelle**
  - **Hierarchische Ausführung**

Select	p.Produktbeschreibung, k.Lieferant,k.Preis
From	Produkt p, Katalog@Marktplatz k
Where	p.Produktbeschreibung= k.Produktbeschreibung
Order by	p.Produktbeschreibung
Expires	5/5/2002



  - **Broadcast-Ausführung**

Select	p.Produktbeschreibung, k.Lieferant,k.Preis
From	Produkt p, Katalog@Marktplatz k
Where	p.Produktbeschreibung= k.Produktbeschreibung
Order by	p.Produktbeschreibung
Expires	5/5/2002



E-Marktplatz (3)

■ **Hyperlinks**

- Einbetten von Hyperlinks als virtuelle Attribute in die DB
- Hyperlinks verweisen auf HyperQueries

■ **Beispiel**

Produktbeschreibung	**Lieferant**	**Preis**
Batterie, 12 V 32 A	Lief 1	hq://Lieferant1.com/Elektro/Preis ?ProdId=CB1232
Reifen, 175/65 TR 14	Lief 2	hq://Lieferant2.com/Reifen/Preis ?ProdId=175_65TR14
. . .	...	...

- Virtuelles Attribut : Preis
- HyperQuery-Protokoll : hq
- DNS : Lieferant1.com
- HyperQuery-ID : Elektro/Preis
- Objektspezifischer Parameter : ?ProdId = CB1232

■ **HyperQueries**

- Teilpläne an entfernten Rechnern
- Virtuelle Tabelle HyperQueryInputStream
- Anfragebeispiel: Elektro/Preis@Lieferant1.com als SQL-Dialekt

```
Select   h.*, p.Preis AS Preis
From     HyperQueryInputStream h, Produkte p
Where    h.ProdId = p.ProdId
```

Prozessorfunktionalität

■ **Funktionale Gleichstellung**
(„horizontale Verteilung“)

- jeder Knoten besitzt gleiche Funktionalität bzgl. DB-Verarbeitung
- i.allg. vollständige DBVS pro Knoten
- Replikation der Funktionen

■ **Funktionale Spezialisierung**
(„vertikale Verteilung“)

- Partitionierung von Funktionen
- Bsp. 1: DB-Maschinen mit Spezialprozessoren für bestimmte DB-Funktionen (Join-Prozessor, Sort-Prozessor usw.)
- Bsp. 2: Föderierte DBS
- Bsp. 3: Workstation/Server-DBS
- Bsp. 4: Web-Informationssysteme (Mehrschichtige Architekturen) mit DB-Server und DB-Verarbeitung auf Applikations-Server

■ **Spezialisierung erschwert Lastbalancierung, Erweiterbarkeit und Fehlertoleranz**

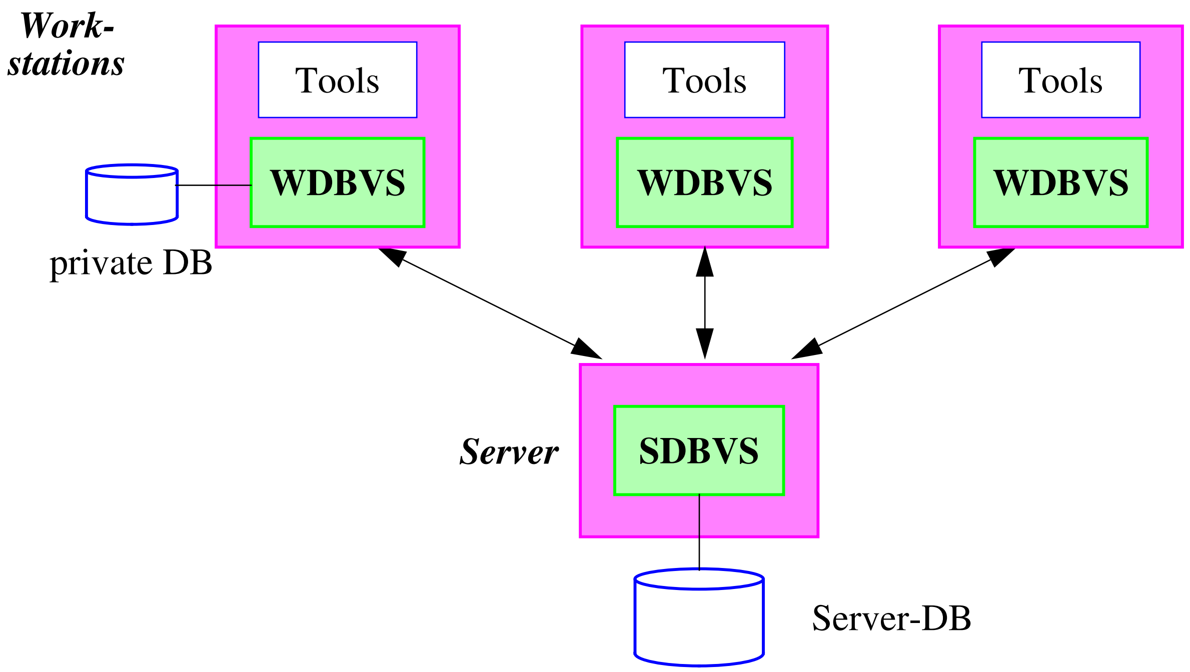
■ **Mischformen aus horizontaler/vertikaler Verteilung**
(Partitionierung und Replikation von DBS-Funktionen)

- n-Server-Systeme: homogen oder heterogen
- mobile Objekte sind dort gespeichert, wo sie gerade häufig referenziert werden

Workstation/Server-DBS

■ **OODBS, Ingenieur-Anwendungen**

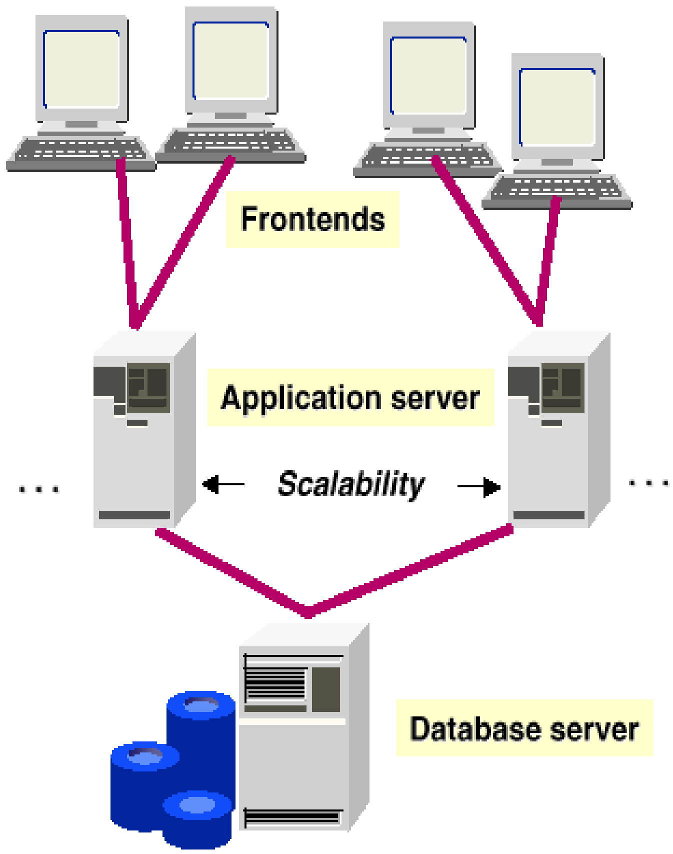
- DB-gestützte Verarbeitung großer, komplex-strukturierter Datenmengen in der Workstation
- hohe Rereferenz-Wahrscheinlichkeit bei den Daten
- lange Transaktionen



■ **Funktionale Spezialisierung**

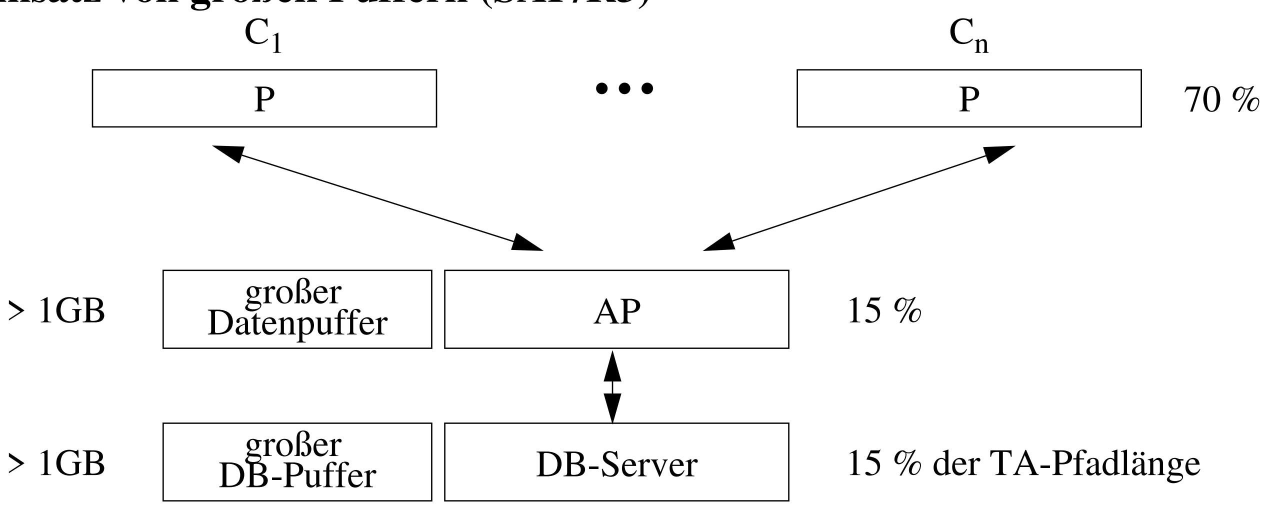
- Datenverwaltung in WS **und** Server
- WS-Objektpuffer: Einsparung von Kommunikationsvorgängen
- lokale Ausführung von Anfragen und Methoden
- globale Aufgaben auf dem Server: Logging, Synchronisation, Externspeicherverwaltung

Mehrschichtige Client/Server-Architekturen



Web-Server
+
Anwendungen +
TP-Monitor
bzw. Applikations-Server

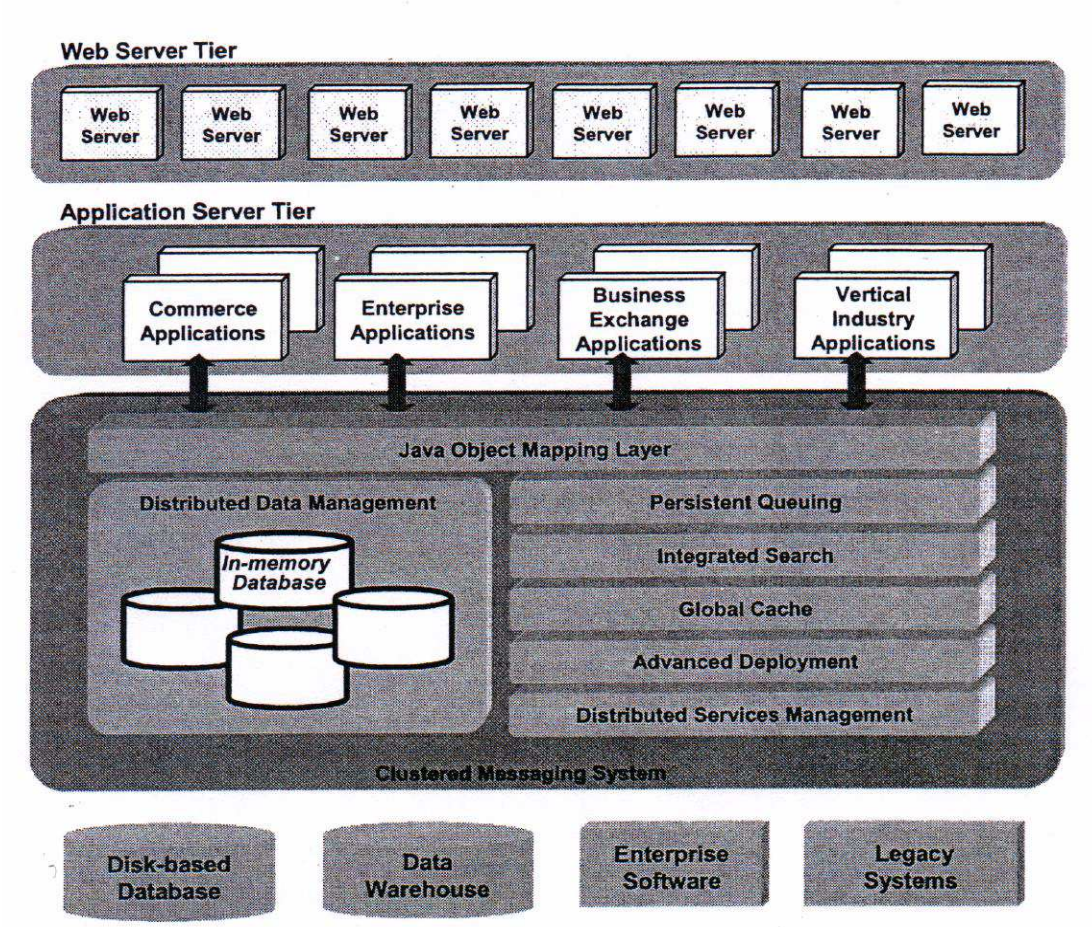
- **Einsatz von großen Puffern (SAP/R3)**



- **Ziele**
  - AW-Daten sollen in AW-Nähe gehalten werden (lokal im AP-Puffer)
  - Zugriffssynchronisation durch AW-Wissen auf Ebene des AP-Servers

  ➥ AP-Server enthält TP-Monitor-Funktionalität und muß viel DBS-Funktionalität reimplementieren

Mehrschichtige Client/Server-Architekturen (2)



Quelle: M. Carey et al.: The Propel Distributed Services Platform, Proc. VLDB 2001

■ **Ziele**

- Entwicklung von datenzentrierten Internet-Applikationen
- Zuverlässigkeit, Verfügbarkeit, Skalierbarkeit
- Entlastung des Flaschenhalses der DBVS-Verarbeitung durch hauptspeicherresidenten Datenverwaltungsdienst

■ **PDSP basiert auf Shared-Nothing-Architektur**

- pro Knoten wird intern TimesTen als hauptspeicherresidentes DBVS zur Speicherung und Anfrageverarbeitung eingesetzt
- persistente Speicherung der Daten durch plattenbasiertes DBVS (wie Oracle oder DB2)
- PDSP bietet den Anwendungen eine ortstransparente Sicht (single image) auf alle verteilten DB-Daten

Grobbewertung von Mehrrechner-DBS

	Parallele DBS (SD, SN)	Verteilte DBS	Föderierte DBS	Workst./ Server-DBS
Hohe Transaktionsraten	++	o/+	o	o
Intra-TA-Parallelität	++	o/+	-/o	o/+
Erweiterbarkeit	+	o/+	o	o
Verfügbarkeit	+/o	+	-	o
Verteilungstransparenz	++	+	o	++
geographische Verteilung	-	+	+	o
Knotenautonomie	-	o	+	-
DBS-Heterogenität	-	-	+	-/o
Administration	o	-	-/--	o

Zusammenfassung

■ **Klassifikationsmerkmale:**

- Rechnerkopplung
- räumliche Verteilung
- Externspeicheranbindung
- Integrierte vs. föderierte und homogene vs. heterogene DBS
- funktionale Spezialisierung vs. Gleichstellung

■ **Vielfältige Anforderungen** an Mehrrechner-DBS
führen zu **verschiedenen Architekturtypen:**

- Parallele DBS
- Verteilte DBS
- Föderierte DBS
- Workstation/Server-DBS

■ **Parallele DBS**

- Ziele: hohe Transaktionsraten, kurze Antwortzeiten,
  ggf. hohe Verfügbarkeit, Skalierbarkeit, Kosteneffektivität
- Nutzung von E/A- und Verarbeitungsparallelität
  (erfordert Datenverteilung über viele Platten)
- Lokale Rechneranordnung ermöglicht:
  effiziente Kommunikation,
  dynamische Lastbalancierung,
  effiziente Parallelisierung komplexer Anfragen
- Hauptansätze: Shared-Everything, Shared-Disk,
  Shared-Nothing und hybride Architekturen
- Parallelisierung in allgemeinem Mehrrechnersystem
  oder dediziertem Back-End-System (DB-Server)

Zusammenfassung (2)

- **Verteilte DBS:**
  - **Ziele:**
    Unterstützung dezentraler Organisationsformen,
    hohe Verfügbarkeit,
    Erweiterbarkeit (neue Knoten),
    Kosteneffektivität
  - **Hauptansatz:**
    räumlich verteilte, integrierte SN-Systeme (globales Schema)

- **Föderierte Mehrrechner-DBS**
  - lose (virtuelle) Integration unabhängiger, ggf. heterogener Datenbanken
  - Bewahrung einer relativ hohen Knotenautonomie
  - Integration von Daten und Funktionen, auch über das Web möglich (Web Services)

- **Alternativen zur Unterstützung heterogener Datenbanken:**
  - Data Warehouses (physische Datenintegration)
  - Anwendungsintegration

- **Mehrrechner-DBS mit funktionaler Spezialisierung**
  - z.B. Workstation/Server-DBS für objekt-orientierte DBS
  - Mehrschichtige Client/Server-Architekturen / Web-Informationssysteme mit DB-Server und DB-Verarbeitung auf Applikations-Servern
  - Nutzung von Spezial-Hardware („Datenbank-Maschinen“) weitgehend gescheitert:
    geringe Kosteneffektivität, Funktionalität und Flexibiltät