9.1 DB-Systemarchitekturen und -schnittstellen

Lernziel:

- ❏ Entwurfsprinzipien für sehr große Softwaresysteme
- ❏ Architekturvorschläge für DBMS
  - "Strawman"-Architektur von CCA/NBS
    (funktional/modular/konzeptuell)
  - "Drei-Schichten"-Architektur von ANSI/SPARC
    (unternehmerisch/prinzipiell/konzeptuell)
  - "Fünf-Schichten"-Architektur von Senko (Überblick)
    (architekturell/abstrakt)
- ❏ Einbettung von DBMS in Betriebssysteme

DBMS Architekturen

Datenbankverwaltungssysteme (DBMSs) sind Softwaresysteme erheblicher Größenordnung. Deshalb spielen Probleme des *"Programmierens im Großen"* (SE-Probleme) bei ihrer Erstellung eine große Rolle:

- ❏ Modularisierung, Modulspezifikationen
- ❏ Gruppierung von Modulen, Schichtenbildung
- ❏ Beziehungen zwischen Modulen, Hierarchien

→ DBMS-Architektur

Anforderungen an Module:

- ❏ Arbeitsteiligkeit
- ❏ Prüfbarkeit
- ❏ Anpaßbarkeit
- ❏ Austauschbarkeit
- ❏ Kalkulierbarkeit

Weitere Anforderungen an “Middleware”:

Standardisierbarkeit (API, ABI, Network Protocols)

Kompatibilität (nach “oben” und “unten”)

Verteilbarkeit (auf Client / auf Server)

Marktstärke des Herstellers (Oracle, Microsoft, ...)

Entwurfsprinzipien für große Systeme (1)

- **Abstraktion:** Konzentration auf das Wesentliche, Verdrängen des Unwesentlichen.
- **Lokalität:** Physische Zusammenfassung von Zusammengehörigem (Daten und Algorithmen).
- **Verdecken:** Beschränkung der Sichtbarkeit von Details auf diejenigen Teile eines Systems, die sie auch benötigen.

Beispiel: Dateiverwaltung

- Durch *Abstraktion* wird der Satz von Datentypen und Operatoren bestimmt, der zur Dateiverwaltung benötigt wird.
- Nach dem Prinzip der *Lokalität* werden diese Operatoren in einem Modul zusammengefaßt.
- Nach dem Grundsatz des *Verdeckens* wird man den Dateiverwaltungsblock verbergen (von außen nicht benötigt).

(s. Vorlesung “Object-Oriented Analysis and Design”)

Entwurfsprinzipien für große Systeme (2)

- **Vollständigkeit:** Wie stellt man sicher, daß auf jeder Abstraktionsebene die gesamte dort angestrebte Funktionalität realisiert ist? Häufig wird in frühen Systemversionen nur der einfache Regelfall, nicht aber die Ausnahme berücksichtigt.
- **Verifizierbarkeit:** Wie kann man sich davon überzeugen, daß die einzelnen Module und damit auch das Gesamtsystem der Spezifikation genügt? Tests? Beweise? Sonstige Überzeugungsarbeit?

Bemerkung:

Praktikable Antworten auf diese Fragen setzen ein Zusammenspiel zwischen

- Programmiersprache,
- Entwurfs- und Programmiermethodik,
- Programmierumgebung

voraus.

Entwurfsprinzipien für große Systeme (3)

In der Regel läuft ein Entwurf *sehr großer* Systeme auf Schichten abstrakter Maschinen hinaus (s. nächste Folie).

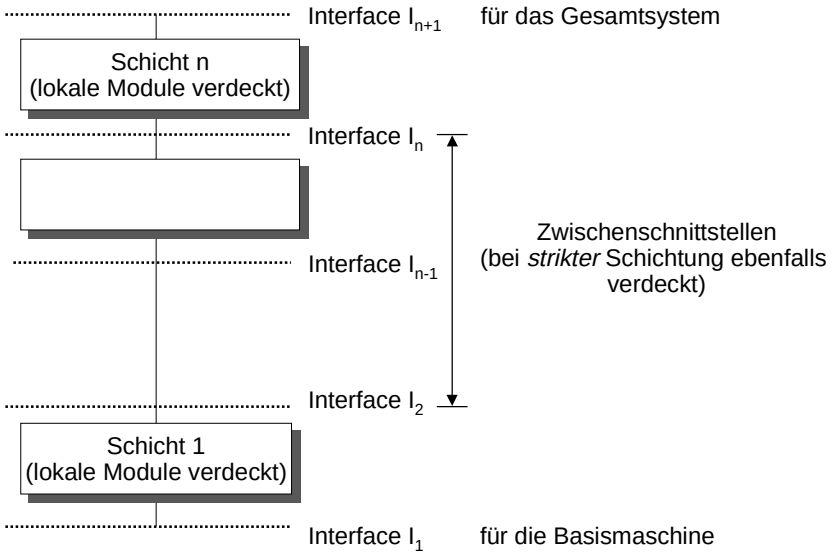
Schichtenbildung verwendet das Prinzip der Abstraktion zur Einhaltung des Lokalitäts- und Verdeckungsprinzip. Eine Schicht besteht dabei häufig aus ca. 10-100 Modulen, die ihrerseits mehrere Objekttypen kapseln können.

Modell der *Abstrakten Maschinen:*

- ❏ Sie bietet an ihrer Schnittstelle eine Menge von Daten und Operatoren (*Ressourcen*) an.
- ❏ Funktionen für das Arbeiten mit den Datenobjekten einer *Abstrakten Maschine.*
- ❏ *Zwei* solcher *Abstrakter Maschinen* sind bei jedem Entwurfsproblem vorgegeben:
  - die *Basismaschine*: *auf ihr* wird implementiert
  - die *Zielmaschine*: *sie* wird implementiert.

Weitere bekannte Beispiele: Betriebssysteme (BIOS, DOS, Windows), Netzwerkprotokolle (ISO / OSI), Compilerarchitekturen (Maschinencode, Assembler, Zwischensprache, AST, ...)

Entwurfsprinzipien für große Systeme (4)



Anforderungen an DB-Verwaltungssysteme (1)

Dominierende Entwurfsfaktoren:

- **Primär:** *PQRI-Anforderungen* (Persistenz, Quantität, Reaktivität, Integrität)
  - Verpflichtende Datenbankdienste, Datenbankkerndienste und erweiterte Datenbankdienste wurden bereits in Kapitel 1 unterschieden.
  - Unterscheide “DBMS” Mac Filemaker, MS Access 1.0 vs. Unix Oracle, IBM IMS
  - Wichtig: Skalierbarkeit durch Schichtenbildung!
  - In den letzten Jahren zunehmend *Verteilungsaspekte:* Anwendungs-Klienten zu DBMS-Server(n), DBMS zu DBMS *peer-to-peer*, Internet-Nutzer zu DBMS, ...

Anforderungen an DB-Verwaltungssysteme (2)

- **Sekundär:** *Sprachschnittstellen* (Cursor, Embedded SQL, Persistente Objekte, ...) und *semantische Objekte* (Baum, Relation, Persistenter Heap) der konkreten Datenmodelle (HDM, NDM, RDM, OODM).
- **Tertiär:** *Basismaschine*
  - Geräteschnittstelle: Zylinder & Blöcke *(raw devices* in Unix)
  - Einbenutzer-Dateischnittstelle (FAT, HPFS, NFS, ...)
  - Transaktionales Dateizugriff über Betriebssystem (Tandem, VMS)

Anforderungen an DB-Verwaltungssysteme (3)

1. Datenmanipulation (DML)

2. Datendefinition (DDL)

3. Datenintegrität und -sicherheit

- ❏ Die Datenbestände einer DB sind oft die wertvollsten Ressourcen eines Unternehmens.
  - Die Benutzer einer DB sind oft keine Informatiker.
  - Eine Datenbank hat meist viele Benutzer.
  - Datenbankbenutzer arbeiten oft gleichzeitig.
- ❏ Maßnahmen:
  - Datenbasissicherung und Fehlererholung (*Recovery*)
  - Datenschutz, Zugriffsberechtigung (Authentisierung, Autorisierung)
  - Transaktionen mit den Eigenschaften Atomarität, Konsistenz, Isolation, Dauerhaftigkeit

→ Kapitel 8

Anforderungen an DB-Verwaltungssysteme (4)

4. Parallelbetrieb

- ❏ Synchronisationsmaßnahmen (isolierte Transaktionen, kooperierende Workflows)
- ❏ Blockadebehandlung (*deadlock detection*)

5. Leistungssteuerung

Oft stehen die verschiedenen Anforderungen an ein DBMS im Widerspruch.

Beispiel: Datenumfang ↔ Puffergröße ↔ Antwortzeit

Wesentliche leistungsbestimmende Charakteristika eines DBMS müssen deshalb *dynamisch* geändert werden können: (Datenbankadministrator, DBA)

- ❏ Zugriffspfade, Puffergrößen
- ❏ Sicherungs- und Synchronisationsverfahren
- ❏ Daten- und Transaktionsverteilung in DBMSs

Anforderungen an DB-Verwaltungssysteme (5)

6. Schnittstellen für unterschiedliche Benutzer

Ein DBMS muß sehr unterschiedliche Benutzerklassen effizient unterstützen:

- ❏ Datenbankverwalter
- ❏ Anwendungsprogrammierer und deren Programmiersysteme
- ❏ Endbenutzer und ihre Werkzeuge

Bemerkung: Wandel der Anforderungen in den letzten Jahren:

- ❏ Dauerhaft hochqualifiziertes Personal wird immer teurer
- ❏ Hardware wird immer billiger
- ❏ “Cost of Ownership” wird wichtiger als “Systemeffizienz”
- ❏ Popularität von Endbenutzerwerkzeugen nimmt rapide zu (MS-Access, Filemaker, ...)

Architekturvorschläge für DBMS

- ❏ Die diskutierte Vorgehensweise beim Entwurf großer Systeme läßt dem Konstrukteur naturgemäß noch viel Freiheit.
- ❏ Haben sich bestimmte DBMS-Architekturen in der Praxis durchgesetzt? Gibt es eine Art Standard?
- ❏ Ist ein solcher Standard für "DBMS im allgemeinen" überhaupt zu erwarten? Wie weit ist eine DBMS-Architektur von den individuellen Anforderungen bestimmt?

Beispiele von DBMS-Architekturen:

- ❏ "Strawman"-Architektur von CCA/NBS (historisch relevant)
- ❏ "Drei-Ebenen"-Architektur von ANSI/SPARC (konzeptuell relevant)
- ❏ "Fünf-Schichtenmodell" von Senko (technisch noch relevant)

Gemeinsamkeiten:

- ❏ Allgemein einsetzbar
- ❏ Datenunabhängigkeit

"STRAWMAN" - Architektur (1)

Ziele: (Auftrag des *National Bureau of Standards* für Studie an die *Computer Corporation of America*, 1982 abgeliefert)

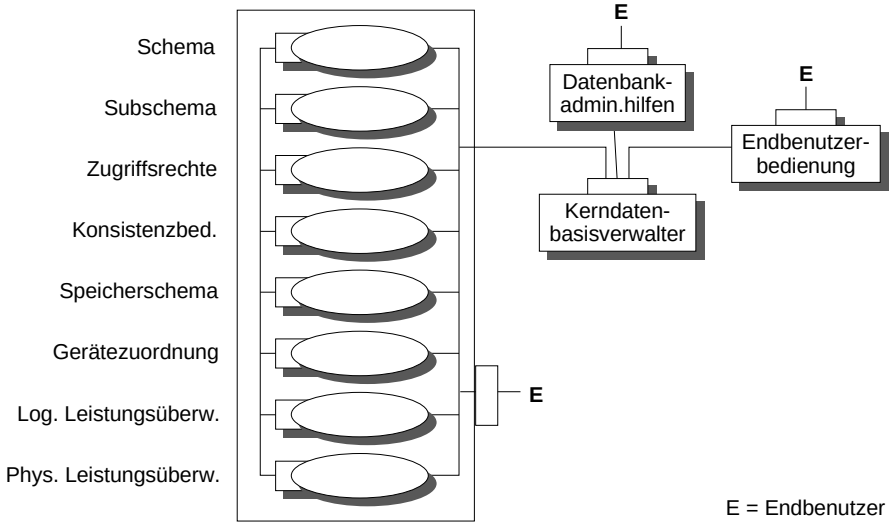
- Standardisierung der Schnittstellen für Endbenutzer und Anwendungsprogrammierer. Erhöhung der Portabilität von Anwendungen
- Standardisierung von *internen* Schnittstellen. Austauschbarkeit von und freier Wettbewerb um DBMS-Komponenten. Systemkonfigurierung durch "Zusammenstecken" von Komponenten
- "Lückenlose" Erfassung der DBMS-Funktionalität

Die Schnittstellen werden auf *drei* Abstraktionsebenen beschrieben:

- **Niedrig:** Einbeziehung der speziellen Wirtsmaschine und des speziellen Datenbankmodells
- **Mittel:** Abstraktion von der speziellen Wirtsmaschine, aber Einbeziehung des speziellen Datenbankmodells
- **Hoch:** Abstraktion von der speziellen Wirtsmaschine und von dem speziellen Datenbankmodell

"STRAWMAN" - Architektur (2)

Beispiel: “Steuerdatenbearbeitung” (Metadaten im Datenwörterbuch)



"STRAWMAN" - Architektur (3)

Gründe für das Scheitern: (?)

- ❏ Rein akademischer Zugang (79 Artikel und Lehrbücher gesichtet)
- ❏ “Über-Generalisierung” (Gültigkeit für beliebige Datenmodelle)
- ❏ Keine Fokussierung und Normierung durch eine Implementierung
- ❏ Kein Markt für “offene” Datenbanksysteme (1981, 1997?)
- ❏ Zu viele Quer-Schnittstellen erschweren Austauschbarkeit von Systemkomponenten

ANSI/SPARC-Architektur (1)

Ziel:

- ❏ Unabhängigkeit zwischen Daten und Anwendungen
- ❏ Unterstützung mehrerer Benutzersichten
- ❏ Literatur: Tsichritzis,Klug, 1978

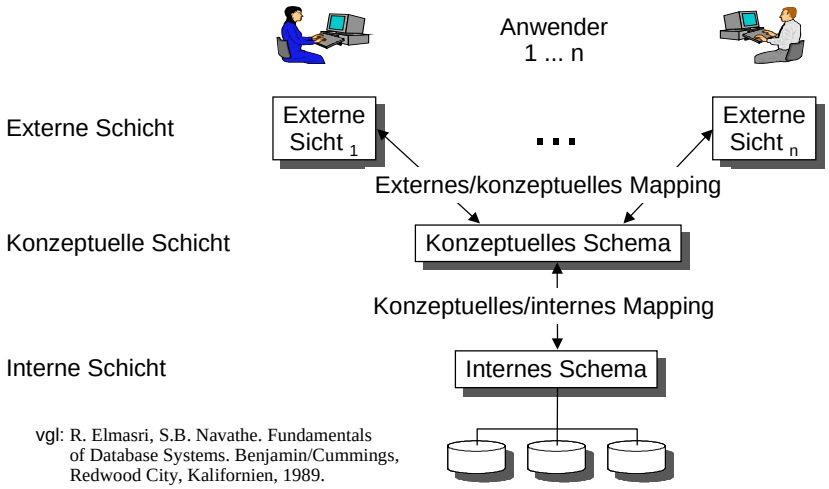
Drei Schichten:

- ❏ Interne Schicht
- ❏ Konzeptuelle Schicht
- ❏ Externe Schicht

- ❏ Definition verschiedener Schemata in den einzelnen Schichten
- ❏ Ablage der Schemata im Datenwörterbuch (data dictionary, repository, ...)

ANSI/SPARC-Architektur (2)

Die Drei-Schichten-Architektur



vgl: R. Elmasri, S.B. Navathe. Fundamentals of Database Systems. Benjamin/Cummings, Redwood City, Kalifornien, 1989.

ANSI/SPARC-Architektur (3)

Externe Schicht:

- ❏ Jedes externe Schema (Benutzersicht: Benutzer = Anwendungsprogramm) beschreibt die Sicht eines oder mehrerer Benutzer auf die Daten.
- ❏ Für den Benutzer nicht relevante Daten werden vor ihm verborgen.
- ❏ Beispiel: Das Schema eines Projektinformationssystems verbirgt das Gehalt der Mitarbeiter.

Konzeptuelle Schicht:

- ❏ Das konzeptuelle Schema legt die Strukturen der konzeptuellen Sicht der gesamten Datenbank für die gesamte Benutzergemeinde fest (Vereinigung aller Anwendersichten zu einer gemeinschaftlichen Sicht).
- ❏ Berücksichtigt werden Entitäten, Datentypen, Beziehungen und Integritätsbedingungen.
- ❏ Die physikalischen Speicherstrukturen werden verborgen.
- ❏ Beispiel: Das Schema eines Firmeninformationssystems sammelt *alle* Informationen über die Mitarbeiter.

ANSI/SPARC-Architektur (4)

Interne Schicht:

- ❏ Internes Schema beschreibt die physikalischen Speicherstrukturen der Datenbank.
- ❏ Unter Benutzung eines physikalischen Datenmodells werden Details der Datenspeicherung und Zugriffspfade beschrieben.
- ❏ Beispiele:
  - Separate Speicherbereiche für Festangestellte und Werkstudenten
  - B-Tree: Zugriff auf Projekte über die Projektnummer

Bemerkungen:

- ❏ Internes Schema wird zunehmend verborgen (vgl. OODBMS)
- ❏ Unterstützung mehrerer externer Schemata in OODBMS noch ungenügend
- ❏ Möglichkeiten zur Realisierung eingeschränkter relationaler konzeptueller Sichten auf NDM, HDM oder auch Dateisystem

ANSI/SPARC-Architektur (5)

Datenunabhängigkeit:

- ❏ Schutz des Benutzers eines DBMS vor nachteiligen Auswirkungen im Zuge von Änderungen in der Systemumgebung (1996 ... 2040).
- ❏ Arten der Datenunabhängigkeit:
  - **Logische Datenunabhängigkeit:**
    - Das konzeptuelle Schema kann ohne Konsequenzen für das externe Schema geändert werden.
    - Beispiel: Erweiterung der Datenbank um eine neue Klasse oder Zusammenfassung mehrerer Klassen durch Generalisierung im konzeptuellen Schema.
  - **Physische Datenunabhängigkeit:**
    - Das interne Schema kann unabhängig vom konzeptuellen Schema geändert werden, ohne daß Funktionsänderungen in den Anwendungen auftreten.
    - Beispiel: Reorganisation der Daten oder Einrichtung neuer Zugriffspfade.

Fünf-Schichten-Architektur von Senko (1)

- ❏ Starke Betonung der *Datenunabhängigkeit* (*DIAM*, *Data Independent Access Model*) (vgl. M. Senko, 1973)
- ❏ Gesteigerte Anpassungsfähigkeit (neue SQL-Standards, neue Wirtssprachen, ...)
- ❏ Bessere Übertragbarkeit zwischen Betriebssystemen (Unix, VMS, MVS, OS/2, MSDOS, ...)
- ❏ Verwendung und Weiterentwicklung in System/R (ca. 1980) sowie von Härder und Reuter (s. Datenbankhandbuch)

Fünf Schichten = sechs Schnittstellen!

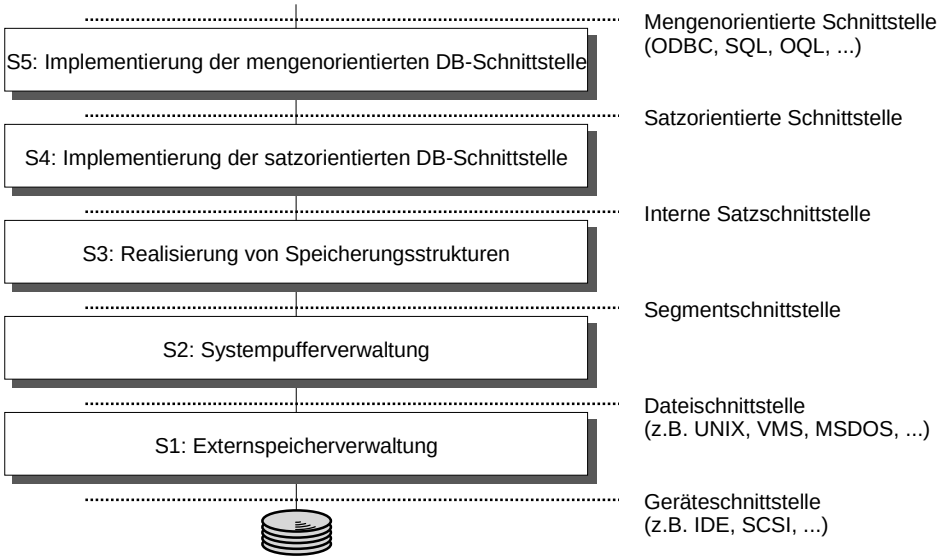
Beachte: Die Benennung der Schichten und der Objekte an den Schnittstellen ist in der Literatur sehr uneinheitlich. Jedoch hohe Übereinstimmung der Konzepte!

Fünf-Schichten-Architektur von Senko (2)

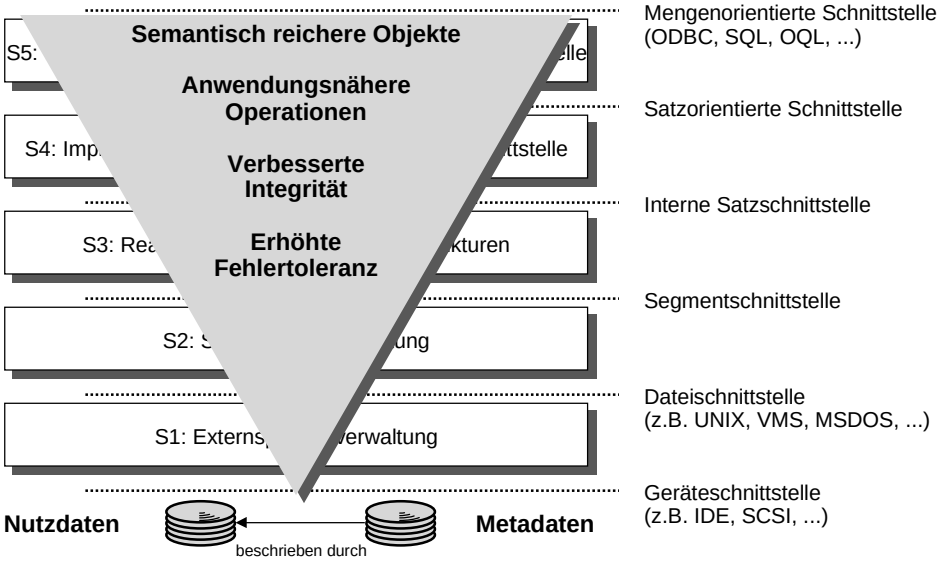
Sechs Schnittstellen

- ❏ **Mengenorientierte Schnittstelle:** Sie unterstützt die Mengen und Prädikate höherer Datenbanksprachen (SQL, OQL, DBPL) und entfällt für das Netzwerk- und das Hierarchiemodell.
- ❏ **Satzorientierte Schnittstelle:** Sie bietet logische Zugriffspfade auf einzelne Sätze, etwa den navigierenden Zugriff im NDM und kann Transaktionen verwalten und Integritätsanforderungen kontrollieren.
- ❏ **Interne Satzschnittstelle:** Sie hat keinen Bezug mehr zum konkreten Datenbankmodell und entscheidet über die Allokation von Sätzen und Satzfeldern hauptsächlich unter Effizienzgesichtspunkten. Auf ihr können Zugriffsmethoden (vom DBA) spezifiziert werden.
- ❏ **Segmentschnittstelle:** Sie bietet einen homogenen, linearen, "unendlichen", virtuellen Adreßraum an (in Seiten bzw. Segmenten). Der Zugriff auf diese Speichereinheiten wird synchronisiert und kann als robust vorausgesetzt werden (*Fehlererholung*).
- ❏ **Dateischnittstelle:** Sie stellt effizienten blockorientierten Dateizugriff zur Verfügung und wird in der Regel vom Betriebssystem des Wirtsrechners realisiert.
- ❏ **Geräteschnittstelle:** Sie wird direkt durch Hardware mit geeigneten Charakteristika (Direktzugriff auf persistente Daten) bereitgestellt.

Fünf-Schichten-Architektur von Senko (3)



Fünf-Schichten-Architektur von Senko (4)



Einbettung eines DBMS in das Betriebssystem (1)

Datenbankklienten:

- ❏ Im Regelfall (Ausnahme: "alte Terminalanwendungen") existiert ein Betriebssystemprozeß pro aktiven Anwendungsprogramm ("Klienten-Prozeß").
- ❏ Der Datenbankklient besitzt lokale Anwendungs- und DB-Sitzungsdaten (Cursor, privater Puffer, ...) und kommuniziert mit einem separaten DBMS-Prozeß ("Server-Prozeß").
- ❏ Durch die Trennung der Adreßräume wird die Integrität der Datenbank gesichert, eine einfache Möglichkeit zum entfernten Datenbankzugriff geliefert, aber auch ein Kopieren von Daten zwischen Prozessen erforderlich.

Ein Klient kann mit mehreren DB-Servern (evtl. von verschiedenen Herstellern) gleichzeitig interagieren. Problematisch:

- ❏ Elementweiser Datenaustausch über den Adreßraum des Klienten
- ❏ Transaktionale Bearbeitung über Datenbankgrenzen hinweg

Viele Datenbankhersteller bieten daher Gateways zu anderen Datenbanksystemen an.

Einbettung eines DBMS in das Betriebssystem (2)

Datenbankserver: Single-Server-Ansatz

- ❏ Ein einzelner zentraler DBMS-Betriebssystemprozeß verwaltet zahlreiche logische Arbeitskontexte, je einen pro aktive Transaktion.
- ❏ In einer Client-Server Umgebung kommunizieren viele Klienten mit diesem Server, der implizit für die Synchronisation zwischen den lokalen Daten dieser Klienten sorgt.
- ❏ In der Realität besteht der zentrale DBMS-Prozeß eher aus einer festen Anzahl von funktional spezialisierten, eng gekoppelten Prozessen (Lock-Manager, Archiv-Manager, Application-Manager, ...)

Datenbankserver: Multi-Server-Ansatz

- ❏ Auf Multiprozessor-Maschinen oder in eng gekoppelten Clustern kann es sinnvoll sein, gleichzeitig mehrere funktional äquivalente DBMS-Betriebssystemprozesse zu haben, die Zugriff auf die gleiche Datenbank bieten.
- ❏ Die Realisierung von Multi-Servern ist erheblich aufwendiger, da komplexe Synchronisations- und Fehlererholungaufgaben zu lösen sind, um die Illusion eines zentralen, konsistenten DBMS aufrechtzuerhalten.