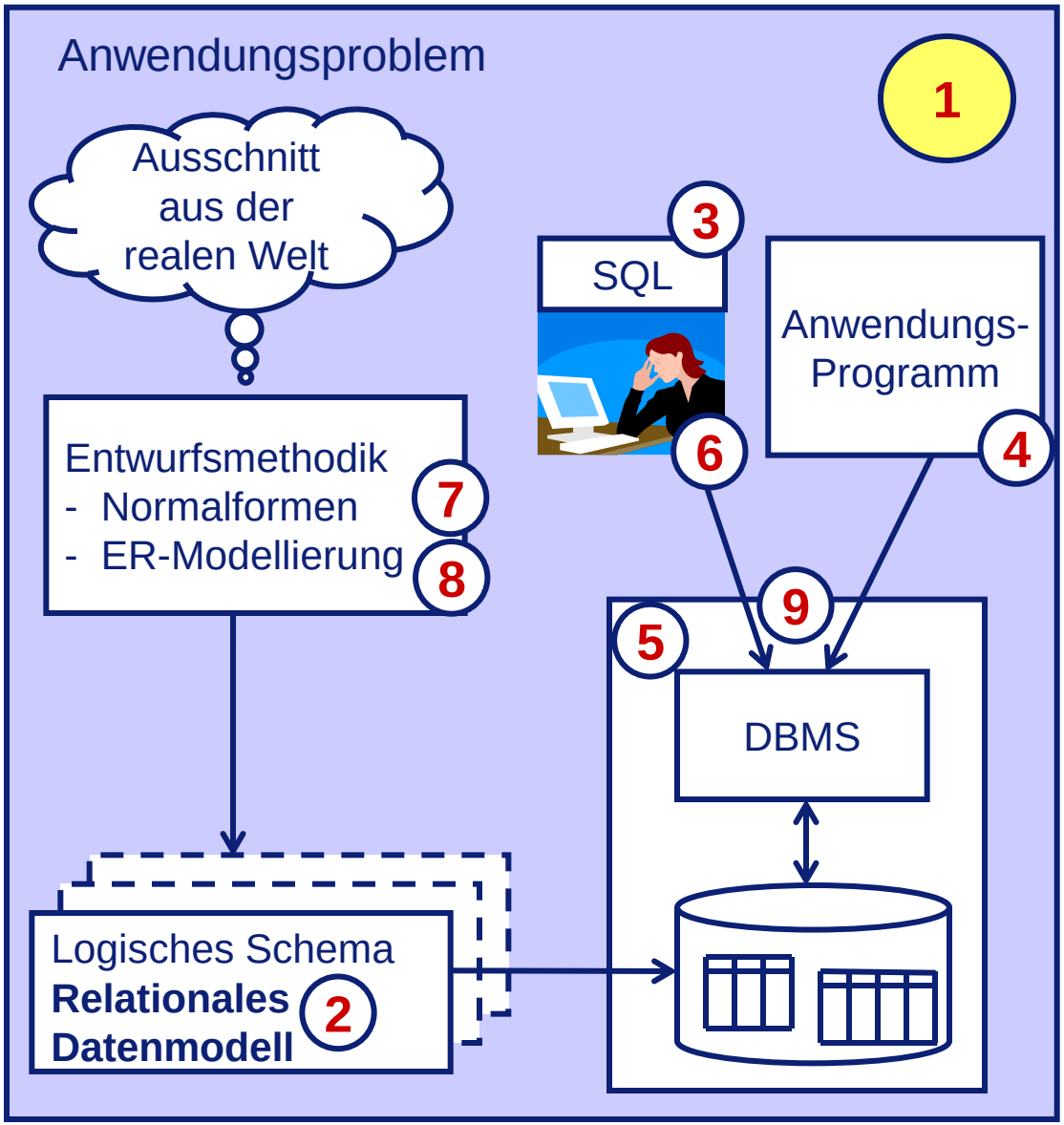
Datenbanksysteme

Kap 1: Einführung

Übersicht über die Inhalte

1. Einführung
2. Das relationale Datenmodell
3. SQL (Datendefinition und –manipulation im rel. Modell)
4. Clientseitige Datenbank-Programmierung (Anwendungsentwicklung)
5. Serverseitige Programmierung
   - Stored Procedures, Trigger
6. Fortgeschrittene DB-Objekte
7. Entwurfstheorie
   - Normalformen
8. Entwurfspraxis
   - ER-Modellierung
9. Transaktionen



Grundlagen

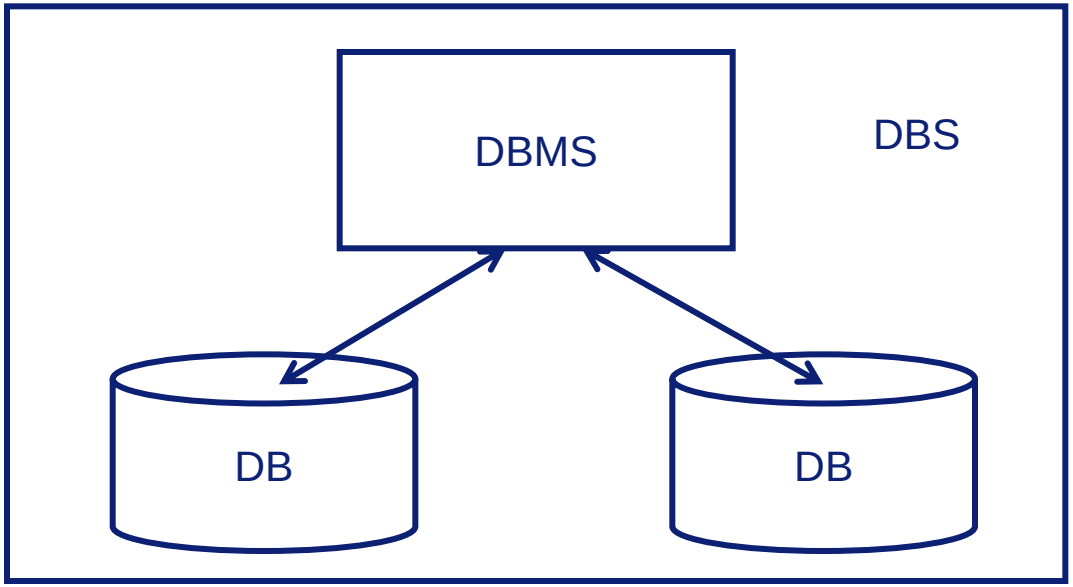
- Definition der Grundbegriffe
- Motivation für den Einsatz von Datenbank-Systemen
- Berufsbilder und Rollen
- Komponenten und Architektur eines DBMS

Was wissen Sie über Datenbanken?

- Wer hat schon mal mit einem relationalen Datenbank-Managementsystem gearbeitet?
- Wem sagen Begriffe wie Primärschlüssel, Fremdschlüssel und Transaktionen etwas?
- Wer spricht SQL?

Begriffe

DB	Datenbank	Strukturierter Datenbestand, der von DBMS verwaltet wird
DBMS	Datenbank-Managementsystem	Software zu Verwaltung von Datenbanken
DBS	Datenbanksystem	DBMS + Datenbank

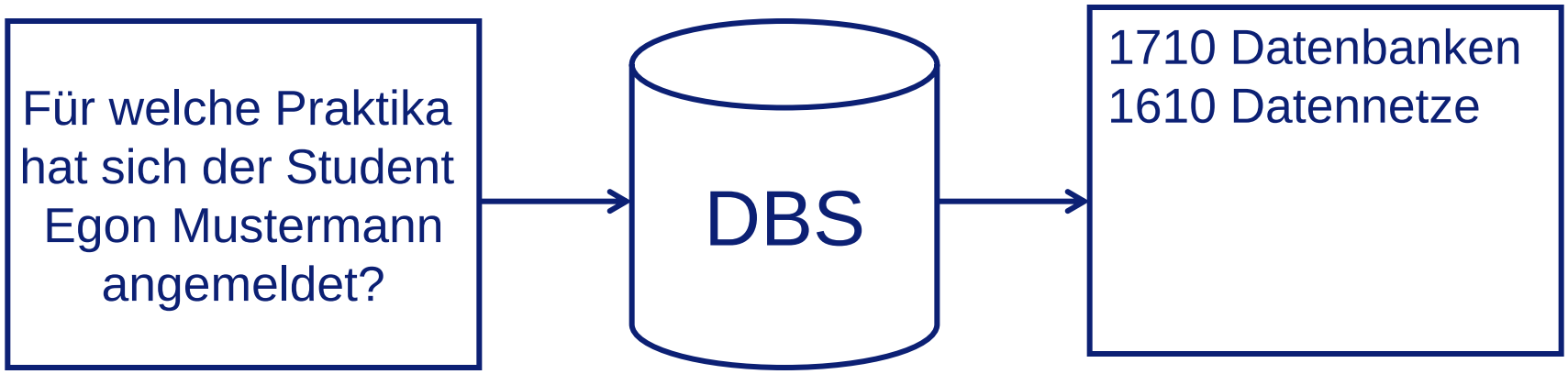


Einsatzgebiete von DBMS (kleine Auswahl)

- Bankwesen
  - Kontoführung, Depots, Überweisungen
- Unternehmensinformationssysteme (ERP)
  - Buchhaltung, Personal, Logistik
- eCommerce
  - eShops, Produktkataloge, B2C/B2B
- Content Management Systeme für das Web
  - Community Sites, Foren
- Bibliotheken
  - Literatursuche, Volltext-Datenbanken, Ausleihverwaltung
- Medizin
  - Krankenhaus-Informationssysteme, elektr. Patientenakte, Gendaten
- CAx-Systeme
  - Konstruktionspläne, Konfigurations- und Versionsmanagement
- Technik & Naturwissenschaften
  - Messdaten, Geodaten, meteorologische Daten

Aufgaben eines Datenbanksystems (1)

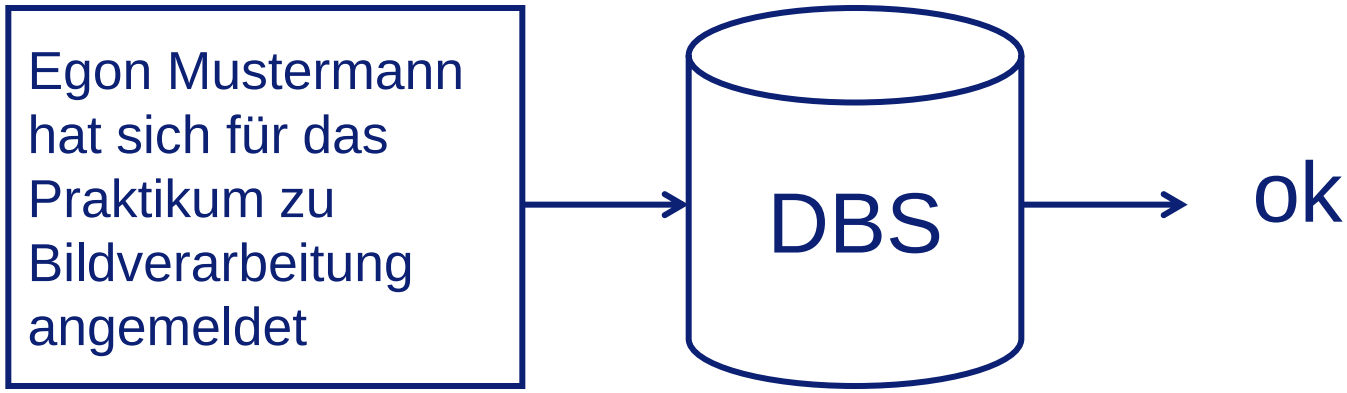
- Worin bestehen die Hauptaufgaben eines DBS?
- Ein DBS soll Antworten auf bestimmte Abfragen zu einem definierten Ausschnitt der realen Welt liefern
- Beispiel: Hochschul-Informationssystem



Aufgaben eines Datenbanksystems (2)

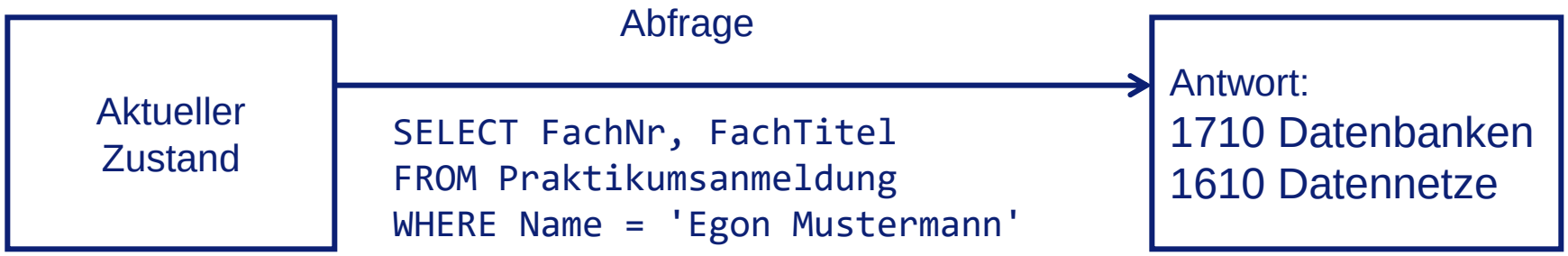
- Ein DBS dient der Speicherung von Daten
- Informationen müssen vom Benutzer in das DBS eingegeben und aktuell gehalten werden

- Beispiel: Hochschul-Informationssystem

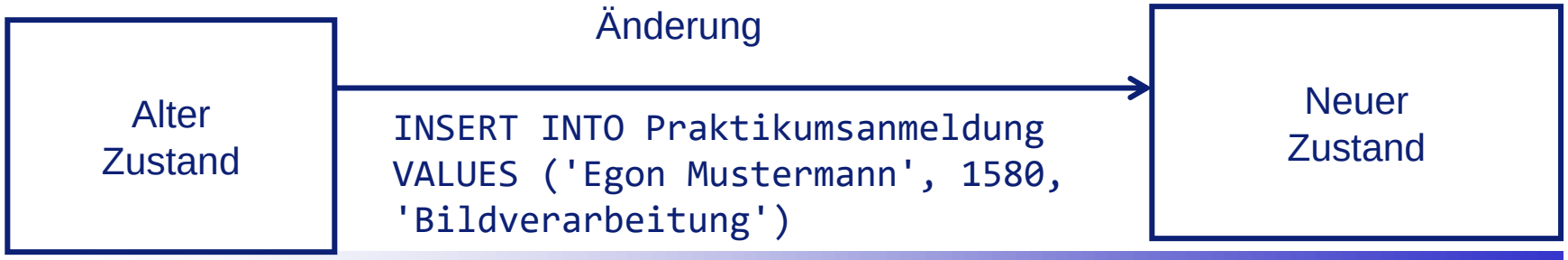


Datenbank-Zustand, Abfrage, Änderung

- Datenbank-Zustand (State)
  - Die zu einem bestimmten Zeitpunkt gespeicherte Information wird durch den **Datenbank-Zustand** (State) charakterisiert
- Anfragen (Query) extrahieren Teile des Zustands

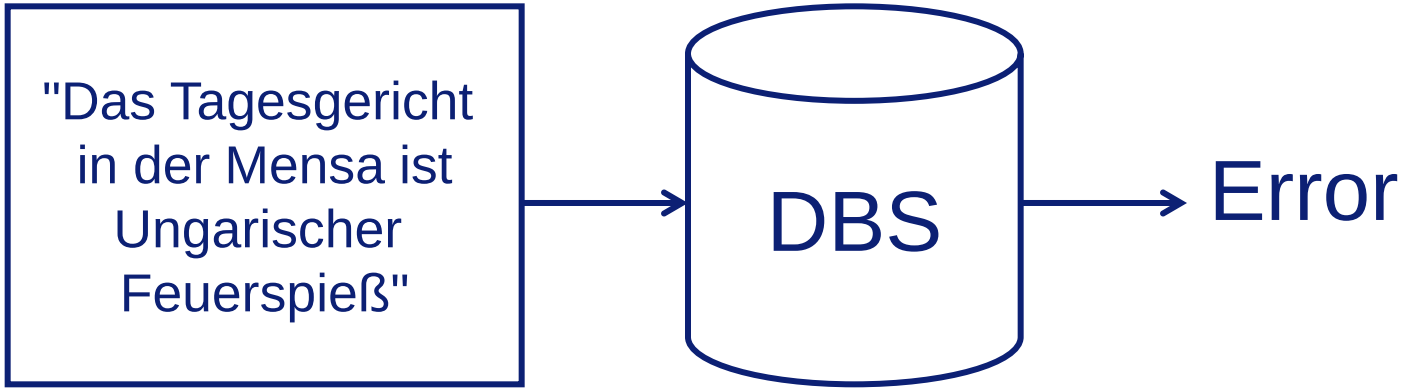


- Änderungen (Einfügen, Aktualisieren, Löschen) überführen den alten Zustand in einen neuen



Strukturierte Information

- Datenbanksysteme können Informationen nur gemäß einer vordefinierten Struktur speichern



- Da die Daten strukturiert vorliegen – und nicht nur als Text – können Datenbanksysteme komplexe Abfragen beantworten:
  - Wieviele Praktika hat jeder Studierende belegt?

Strukturierte Daten

- Ein DBS speichert im Grunde genommen nur Daten (Zeichenketten, Zahlen, ...) und keine Information
- Daten werden zu Information durch Interpretation
  - Es muss klar sein, wie die Zeichenketten 'Egon Mustermann' und 'Datenbanken' zu interpretieren sind
- Je mehr ein DBS über die Struktur der Daten "weiß", desto besser kann es den Nutzer unterstützen
  - Daher müssen Konzepte wie Studierender und Praktikum zunächst definiert/deklariert werden
  - Natürlich kann ein DBS beliebige Texte speichern. Dann kann es allerdings nur nach Zeichenketten suchen und keine komplexen Abfragen beantworten.

Datenbank-Schema vs. Zustand

- Schema
  - Formale Definition der Struktur der Datenbank-Inhalte
  - Bestimmt die möglichen Datenbank-Zustände
  - Wird (in der Regel) nur einmal definiert, wenn die Datenbank erzeugt wird
  - Analog zur Deklaration von Variablen (Typdefinition)
- Zustand
  - Ausprägung oder Instanz des Datenbank-Schemas
  - Enthält die eigentlichen Daten, strukturiert gemäß Schema
  - Ändert sich häufig:
    - immer dann, wenn Updates ausgeführt werden (Einfügen, Ändern, Löschen)
  - Analog zur aktuellen Wertebelegung einer Variablen

Schema und Zustand im relationalen Datenmodell

- Im relationalen Datenmodell werden Daten in Form von Tabellen (Relationen) strukturiert
- Jede Tabelle hat einen Namen, eine Sequenz von benannten, typisierten Spalten (Attribute) und eine Menge von Zeilen (Tupel)

Praktikumsanmeldung		
Name	FachNr	FachTitel
Egon Mustermann	1710	Datenbanken
Egon Mustermann	1610	Datennetze
...	...	...
Heidi Musterfrau	1580	Bildverarbeitung

Schema

Zustand/
Ausprägung

Transiente vs. Persistente Daten

- Transiente Daten
  - Befinden sich auf flüchtigem Medium, z.B. Arbeitsspeicher
  - Sind einem laufenden Prozess (Programm in Ausführung) exklusiv zugeordnet
  - Hören auf zu existieren, sobald der Prozess sich beendet ==> bei Neustart nicht mehr verfügbar
- Persistente Daten
  - Befinden sich auf nichtflüchtigem Speichermedium, z.B auf Festplatte, organisiert als Dateien
  - Können von verschiedenen Prozessen erzeugt, verändert und gelöscht werden (im Prinzip auch nebenläufig)
  - Lebensdauer der Daten nicht an Lebensdauer der Prozesse gekoppelt
  - Existieren nach Prozessende weiter und stehen auch beim Neustart zur Verfügung

Was ist ein Datenbank-Managementsystem?

Datenbank-Managementsystem (DBMS)

=

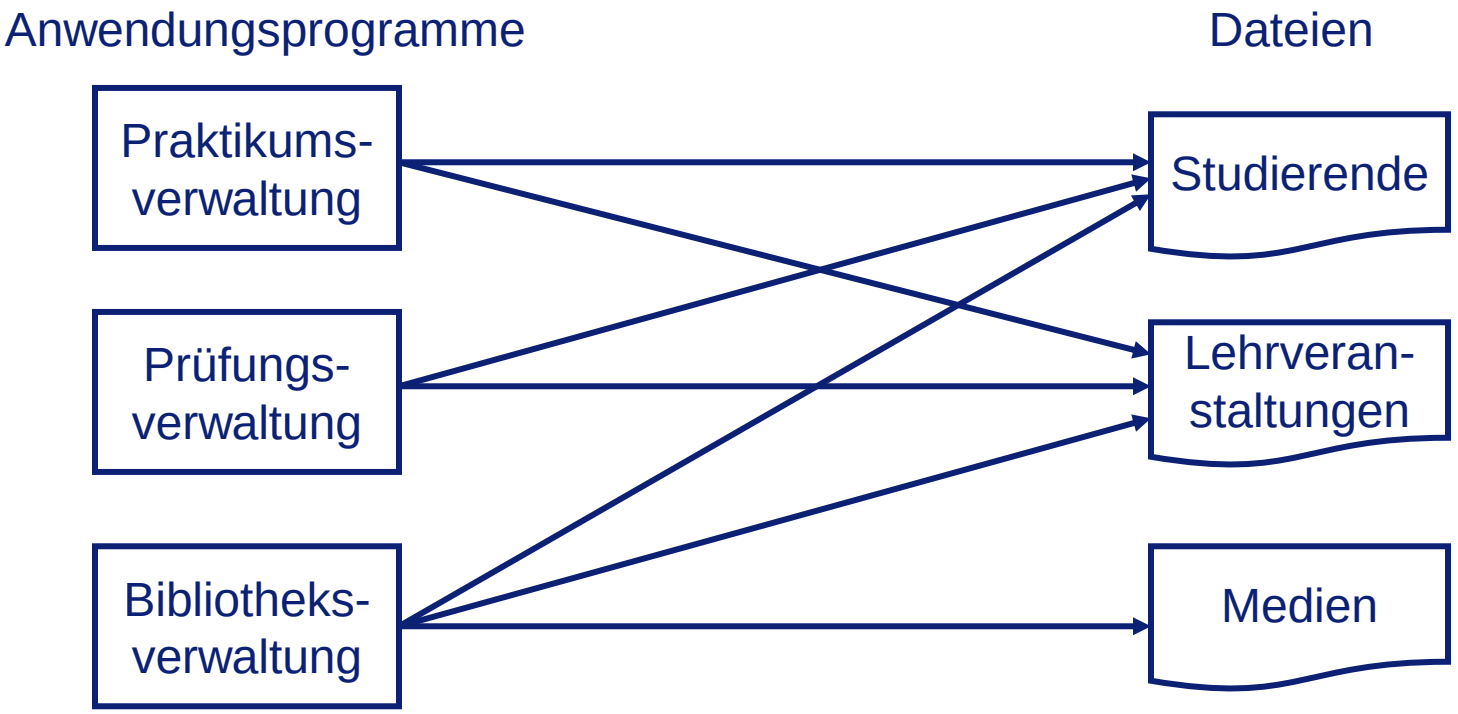
Verwaltungs-, Speicherungs- und Abfragekomponente
für persistente und strukturierte Daten

- Datenbank, Datenbasis (DB)
  - Persistent gespeicherte Daten
- Datenbanksystem (DBS) = DB + DBMS

- Frage:
  - Warum reicht dafür nicht das Dateisystem eines Computers?

Traditionelle verarbeitungsorientierte DV

- Anwendungen arbeiten direkt auf Dateien
- Datei-Zugriffsoperationen
  - z.B.: java.io: open(), seek(), read(), write(), close()
- Welche Probleme treten beim unkoordinierten Zugriff auf?



Probleme bei dateibasiertem Zugriff

- Inkonsistenz
  - Bei überlappenden Zugriffen ("Lost Update")
  - Klassisches Beispiel: Gleichzeitige Überweisung zwischen zwei Bankkonten (siehe Übung)
- Mangelhafter Datendurchsatz
  - Bei Sperren auf ganzen Dateien
  - Feinkörnigere Sperrung bei dateibasiertem Zugriff nur mit erheblichem Zusatzaufwand möglich
- Overhead
  - Zugriffslogik sowie Schutz- und Sicherungsmechanismen müssen in allen Anwendungsprogrammen implementiert werden

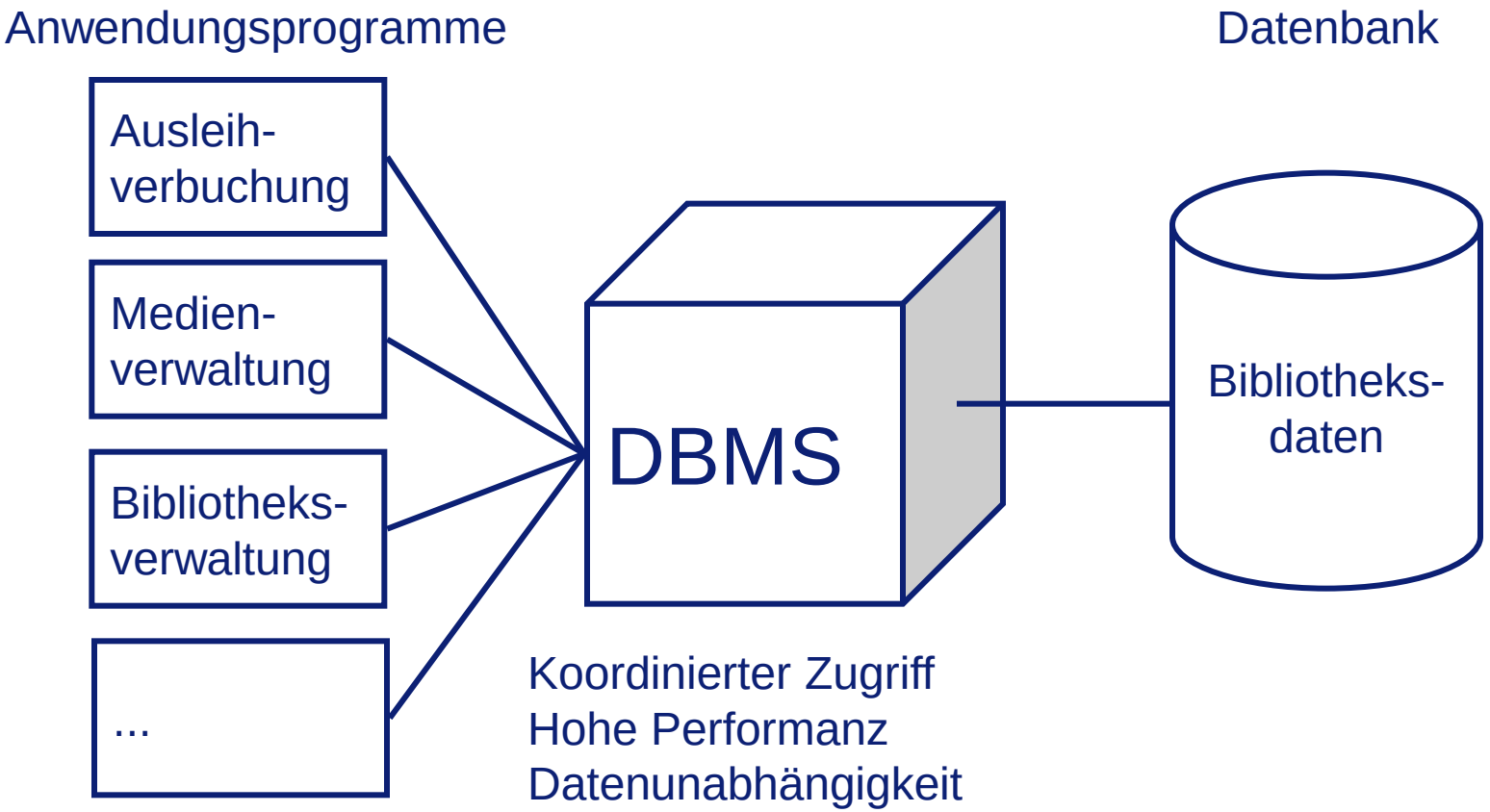
Rolle von Applikationen vs. Daten

- In der traditionellen SW-Entwicklung spielen Applikationen die Hauptrolle
  - Datenorganisation auf Erfordernisse der Applikation zugeschnitten
  - Keine offene, unabhängige Dokumentation der Daten
  - Daten lassen sich nicht für unvorhergesehene Zwecke nutzen
  - Frustrierend, wenn man weiß, dass die Daten zwar "da sind", aber eine neue Auswertung zu schwierig/aufwändig zu programmieren

- Aber: Daten oft langlebiger als Applikationen
  - Datenmigration bei neuen Programmversionen erforderlich

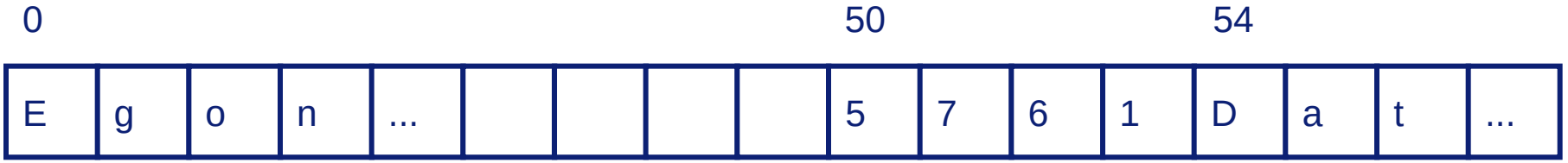
Lösung durch DBMS

- Entkopplung zwischen Applikationen und Daten
  - Daten unabhängig von spezifischen Applikationen
  - Applikationen unabhängig von der Art der Datenspeicherung



Datendefinition in DBMS vs. Dateisystem

- Implementierung mit Dateien
  - Dateien als Bytefolgen



- Struktur der Datensätze muss vom Programmierer festgelegt werden
- Information über Dateistruktur existiert im Code der Anwendungsprogramme (bzw. in den Köpfen der Programmierer)
- Betriebssystem kann nicht vor Fehlern schützen, da es die Struktur nicht kennt

Datendefinition in DBMS vs. Dateisystem

- Implementierung mit DBMS
  -
```
CREATE TABLE Praktikumsanmeldung (
    Name: VARCHAR(50),
    FachNr: INTEGER,
    FachTitel: VARCHAR(20)
);
```
- Die Struktur der Daten ist formal definiert und dem DBMS explizit bekannt
- Das DBMS kann Typfehler in den Applikationsprogrammen erkennen
- Höherer Abstraktionsgrad, vereinfachte Implementierung

DBMS als Middleware

- DBMS = Softwareschicht über dem Betriebssystem
  - Zugriff auf Daten nur über das DBMS
  - Baut (meistens) auf dem Dateisystem des Betriebssystems auf
- DBMS als Unterprogrammbibliothek für Datenzugriffe
  - Operationen auf höherer Ebene
    - z.B. Zugriff auf das Feld 'Name' statt auf Bytes ab Position 0
  - Enthält viele Algorithmen, die ein Anwendungsprogramm sonst selbst implementieren müsste
    - Sortieren (MergeSort), Suchen (B-Bäume), Aggregation, Pufferverwaltung, Freispeicherverwaltung, ...
    - Optimiert für große Datenmengen und viele Benutzer

DBMS als Sammlung abstrakter Datentypen

- Indirektion des Datenzugriffs über DBMS erlaubt es, interne Veränderungen zu verstecken
- Idee abstrakter Datentypen
  - Implementierung ändert sich
  - Schnittstelle bleibt stabil

→Datenunabhängigkeit

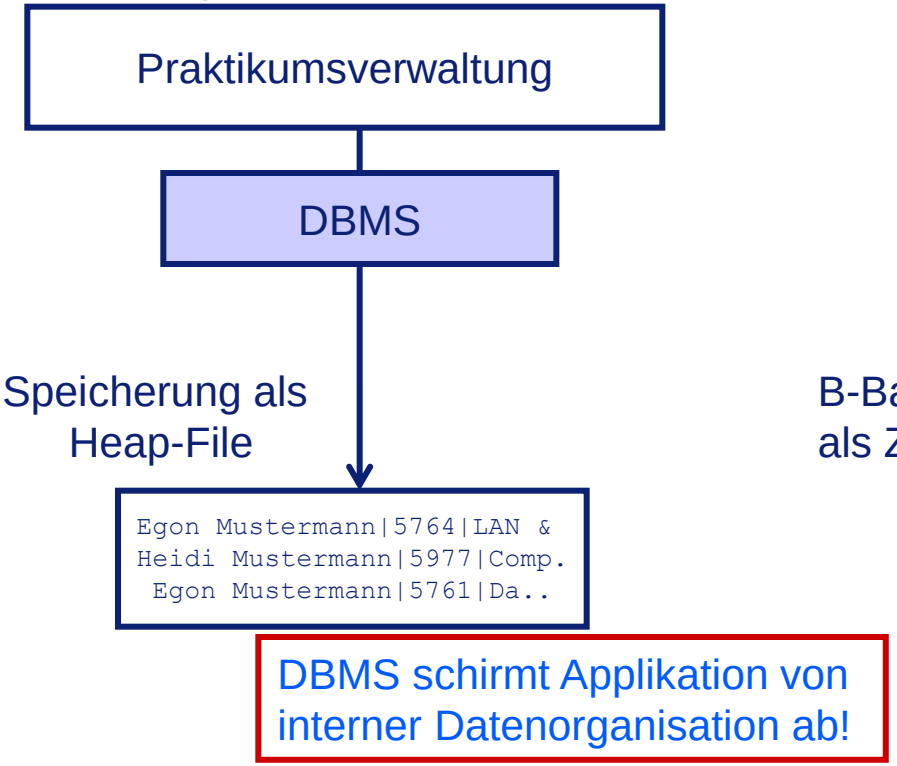
Physische Datenunabhängigkeit

- Änderungen an den physischen Speicherstrukturen und Zugriffspfaden sollen für Applikationen "transparent" (=unsichtbar) sein
- Nur das DBMS kennt die interne Datenorganisation, nicht die Applikationen
- Beispiele für nachträgliche Änderungen
  - Auslagern auf mehrere Platten
    - Die Last ist so groß geworden, dass eine einzelne Festplatte nicht die geforderte Anzahl von Datenzugriffen pro Sekunde erfüllt

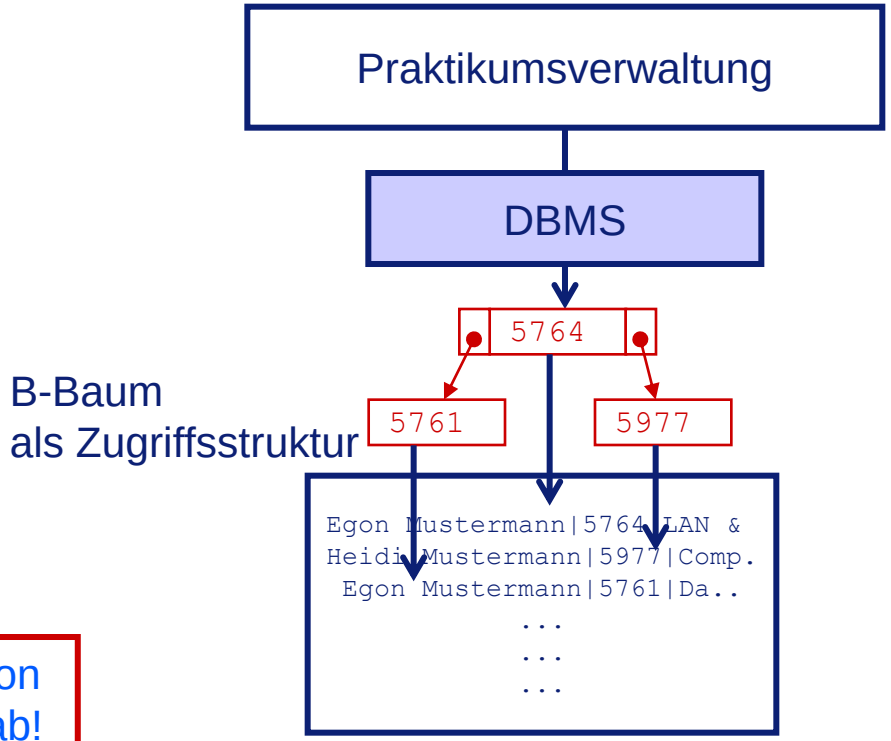
    → Aufteilen des Datenbestands auf mehrere Platten
  - Anlegen eines Index (siehe nächste Folie)

Beispiel: Physische Datenunabhängigkeit

- Zunächst:
  - Professor benutzt Praktikumverwaltung nur für seine Vorlesung im aktuellen Semester
  - Wenige Datensätze, wenige Zugriffe



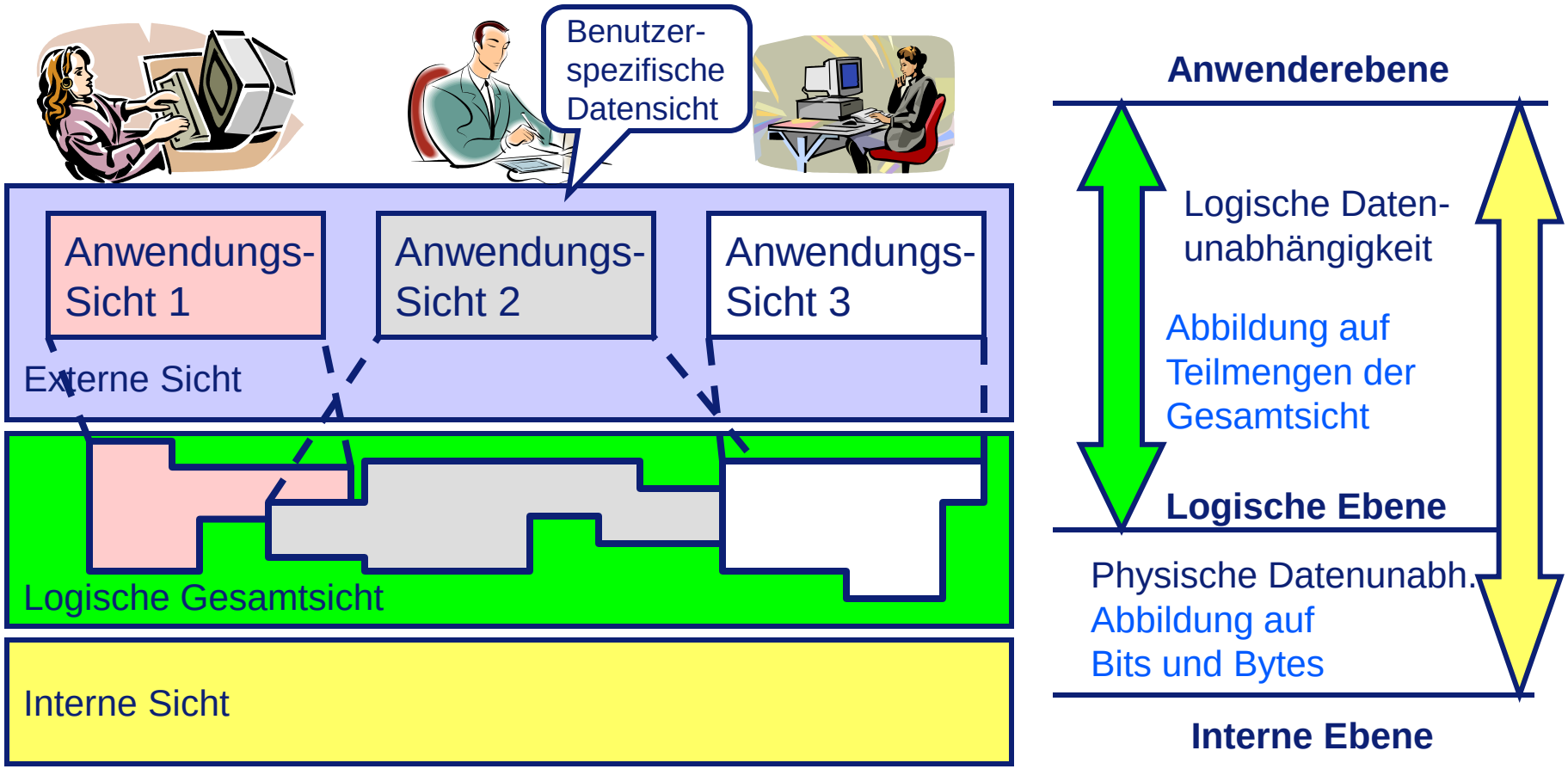
- Später:
  - System wird hochschulweit eingesetzt
  - Verändertes Lastprofil: viele Datensätze, viele Zugriffe



Logische Datenunabhängigkeit

- Applikationen werden häufig für bereits existierende Datenbanken entwickelt
- Dabei ergeben sich oft neue Anforderungen an das existierende Datenbank-Schema
  - Beispiele
    - Hinzufügen von Tabellen/Spalten
    - Umbenennungen
    - Aufteilen eines Attributs auf mehrere (Name → Vorname, Nachname)
- Logische Datenunabhängigkeit heißt:
  - (Kleinere) Änderungen am Datenbank-Schema sollen sich nicht auf existierende Applikationen auswirken
- Sichten-Mechanismus
  - Applikation operieren nicht direkt auf dem Datenbank-Schema, sondern auf externen Sichten
  - Externe Sichten definieren eine applikationsspezifische Teilmenge/Abstraktion des Datenbank-Schemas

ANSI/SPARC* Architektur



*1975 vom Standards Planning and Requirements Commitee (SPARC) des American National Standards Institute (ANSI) entwickelt

Ziele: Datenunabhängigkeit

- Physische Datenunabhängigkeit
  - Änderungen an den physischen Speicherstrukturen und Zugriffspfaden sollen für Applikationen "transparent" (=unsichtbar) sein
- Logische Datenunabhängigkeit:
  - Änderungen am logischen Gesamt-Schema sollen sich nicht auf existierende Applikationen auswirken
  - Gelingt nur für Erweiterungen und kleinere Änderungen
- Wird als ein Kernkonzept von DBMS angesehen:
  - Christopher J. Date: „Data independence is the immunity of applications to change in storage and access strategy“

Ziele (Fortsetzung)

- Einfache, aber mächtige Operationen für Datenmanipulation
  - Einfügen (insert)
  - Ändern (update)
  - Löschen (delete)
  - Auffinden (retrieve)
  - Verknüpfen
- Performanz und Skalierbarkeit
  - Massendaten
  - Hoher Durchsatz
  - Viele, nebenläufige Anwendungen

Ziele (Fortsetzung)

- Datensicherheit
  - Garantie konsistenter Daten, selbst bei
    - Abgebrochenen oder fehlerhaften Anwendungen
    - Software/Hardware-Abstürzen
  - Automatisierte Backup und Recovery-Mechanismen
- Concurrency Control und Synchronisation
  - Koordination des Zugriffs bei Mehr-Benutzerbetrieb
  - Verschränkter Zugriff durch mehrere Applikationen
    - Schutz gegen unerwünschte Interferenzen
  - Isolation: DBMS garantiert, dass sich Anwendungen nicht gegenseitig beeinflussen
- Integritätskontrolle
  - DBMS erlaubt die Definition von Konsistenzregeln (Integritätsbedingungen)
  - DBMS garantiert, dass alle ihm bekannten Konsistenzregeln eingehalten werden
  - Keine Anwendung kann Schaden anrichten (inkonsistenten Datenbank-Zustand herstellen)

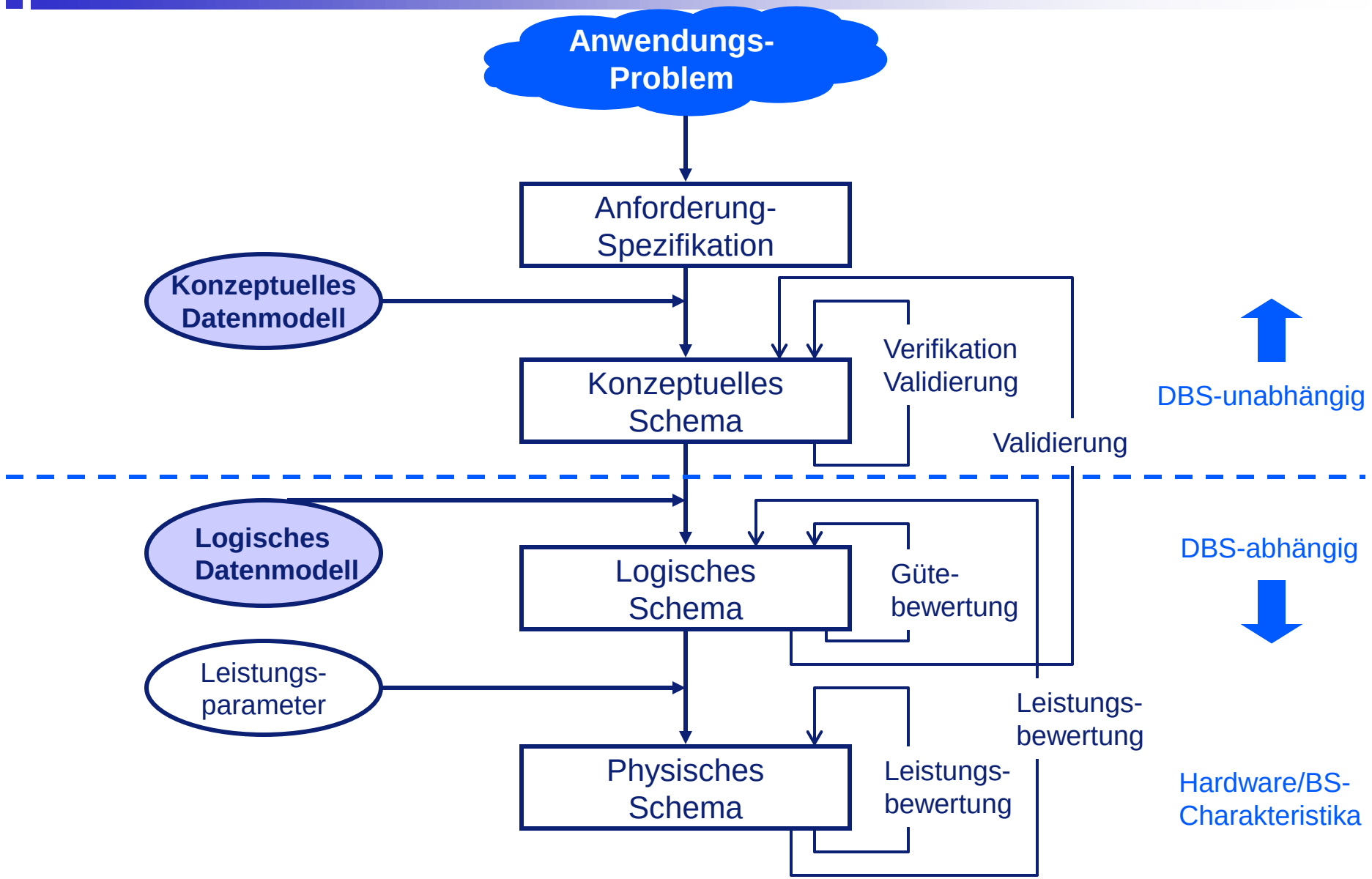
Ziele (Fortsetzung)

- Redundanzvermeidung
  - Daten über ein Objekt sollte nur einmal in der Datenbank gespeichert sein
  - Kontrollierte Redundanz nur für schnellen Zugriff oder Datensicherheit
- Datenschutz
  - Betriebliche oder gesetzliche Regelungen für den Umgang mit Daten
    - Speicherung
    - Nutzung
    - Veröffentlichung
    - Löschung
  - DBMS bietet Schutz gegen unberechtigte Nutzung durch Kontrolle von
    - Zugriff
    - Sichtbarkeit

Ziele von DBMS: Zusammenfassung

- Verwaltung persistenter Daten
- Datenunabhängigkeit
- Operationen für Datenmanipulation
- Performanz und Skalierbarkeit
- Datensicherheit
- Concurrency Control
- Integritätskontrolle
- Datenschutz

Phasen des Datenbank-Entwurfs

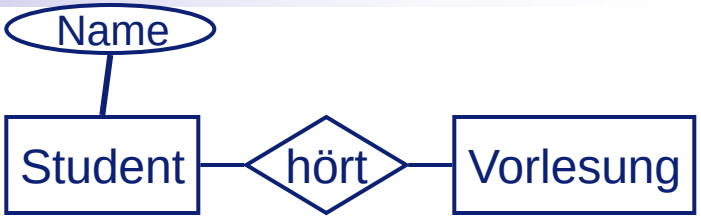


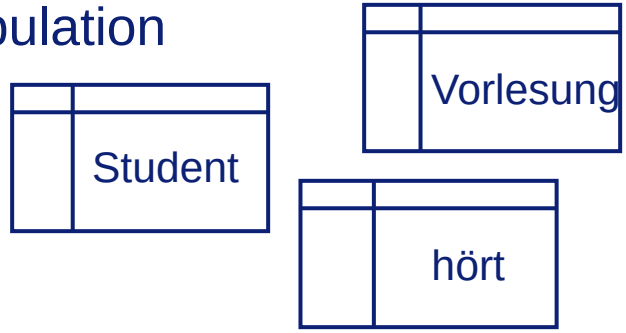
Datenmodelle

- System von Konzepten zur Darstellung eines Ausschnitts der realen Welt mittels Daten
- Bestehen aus
  - Strukturen (statische Eigenschaften)
  - Operationen (dynamische Eigenschaften)
  - Integrity Constraints (Korrektheitsbedingungen)

Konzeptuelle vs logische Datenmodelle

- Konzeptuelle Datenmodelle
  - Entity-Relationship-Modell
    - Graphisches, semiformales Modell zur Darstellung eines Ausschnitts der realen Welt
    - Hohe Abstraktionsebene, gut geeignet zur Kommunikation mit “naiven” Anwendern
    - Wenige strukturelle Constraints, keine Operationen: es gibt keine ER-DBMS !
  - UML (Unified Modeling Language)
- Logische (Implementierungs-)Datenmodelle
  - Relationales Modell
    - Extrem einfache Struktur: alles ist eine Tabelle
    - Generische Operationen zur Datenmanipulation
    - Abfragen filtern/verknüpfen Tabellen und liefern als Ergebnis wieder Tabellen





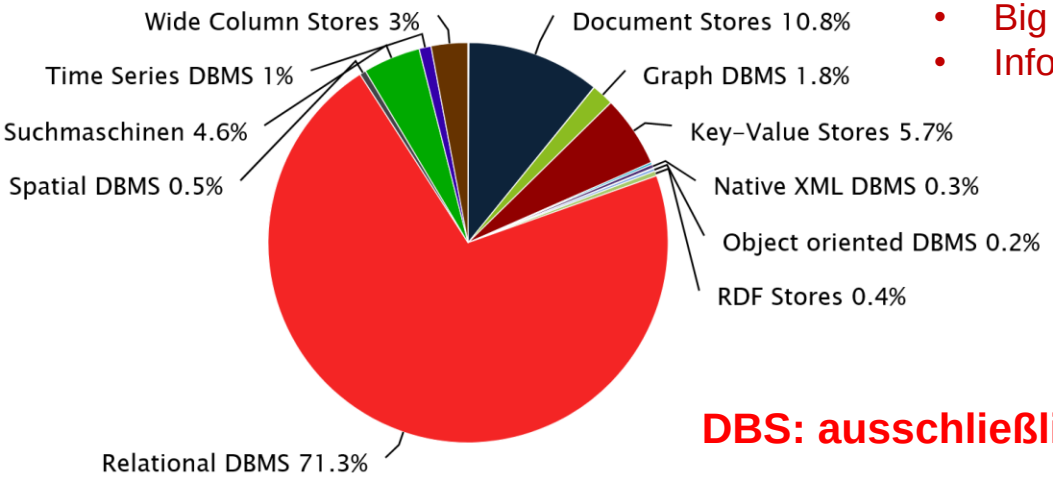
Weitere Logische Datenmodelle

- Hierarchisches Modell
  - Historisch ältestes Modell
  - Alles ist ein Baum
    - Zugriffpfade nur über Parent-Child-Beziehungen (1:N)
  - Revival in Dokumentorientierten Datenbanken mit XML/JSON-Format zur Datenrepräsentation (z.B. mongoDB)
- Netzwerkmodell
  - Vorherrschendes Datenmodell in den frühen 70er Jahren
  - Alles ist ein Graph
    - Flexiblere Zugriffspfade
    - Bessere Abbildung von N:M-Beziehungen
  - Revival in Form von Graphdatenbanken (z.B. neo4J)
- Objektorientiertes Datenmodell
  - Konzepte aus der Objektorientierten Programmierung
  - Struktur und Verhalten in Klassen gekapselt
  - Komplexe Objektgeflechte

Relationale vs. NoSQL-Datenbanken

- NoSQL (Not only SQL)-Datenbanken basieren auf alternativen Datenmodellen und DBMS-Paradigmen
  - Schwerpunkt auf Performanz, Skalierbarkeit und Verteilung (BigData-Anwendungen)
  - Abstriche bei Konsistenz, Datenintegrität und Ausdrucksmächtigkeit der Anfragesprache

Rankingpunkte pro Kategorie in Prozent, September 2022



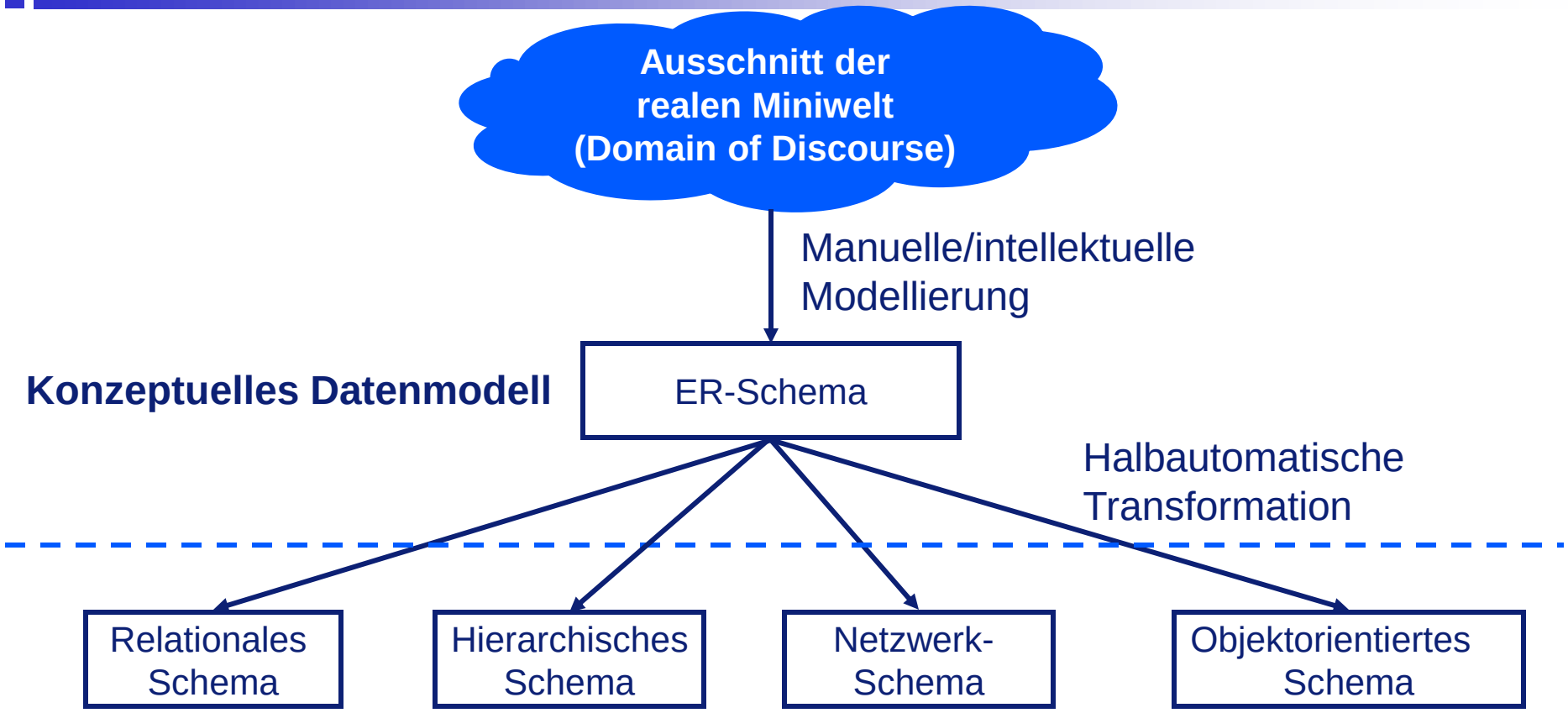
NoSQL:
- WPM Data Science (Bachelor, Quix)
- Big Data Technologien (Master, Quix)
- Information Retrieval (Master, Weidenhaupt)

Dieses Diagramm zeigt die Popularität der Kategorien, basierend auf der Popularität der einzelnen Systeme. Dabei entspricht die Summe der Ranking-Punkte aller Systeme 100%.

DBS: ausschließlich relationale DBMS

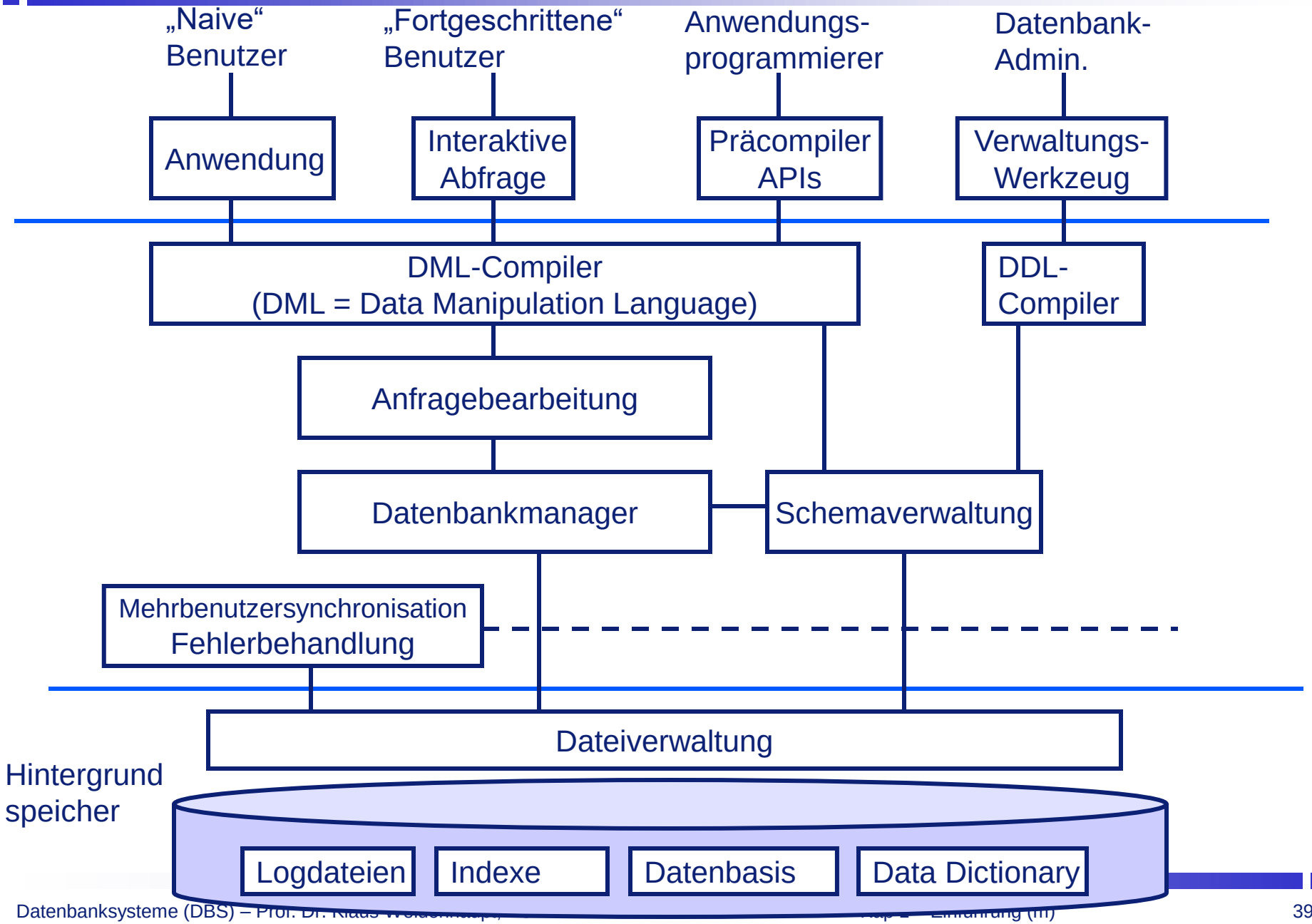
Quelle: https://db-engines.com/de/ranking_categories

Datenmodellierung



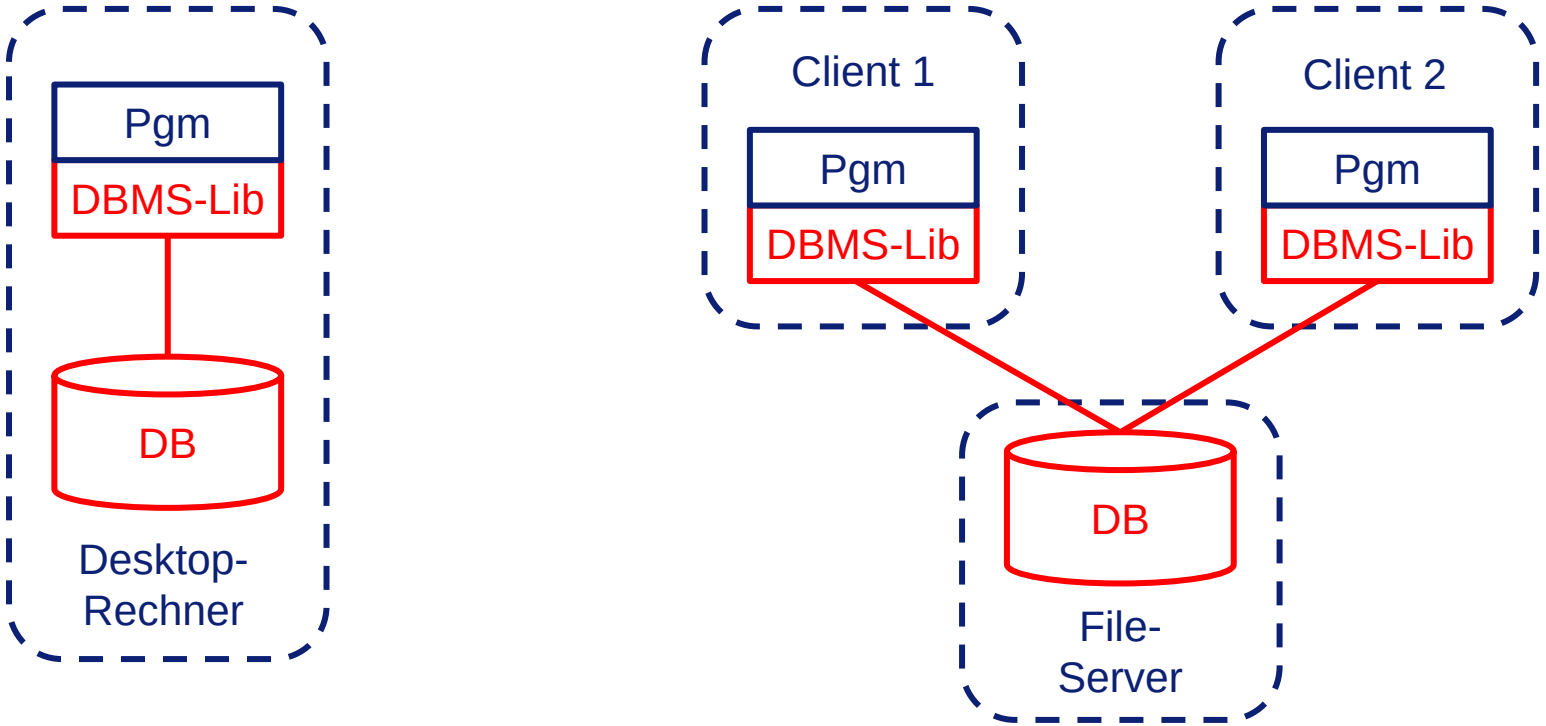
- Konzeptuelle Modellierung ist der schwierige, kreative Teil der Datenmodellierung!
- Erfordert Verständnis der Anwendungsdomäne und intensive Zusammenarbeit mit Fachabteilungen

Komponenten eines DBMS



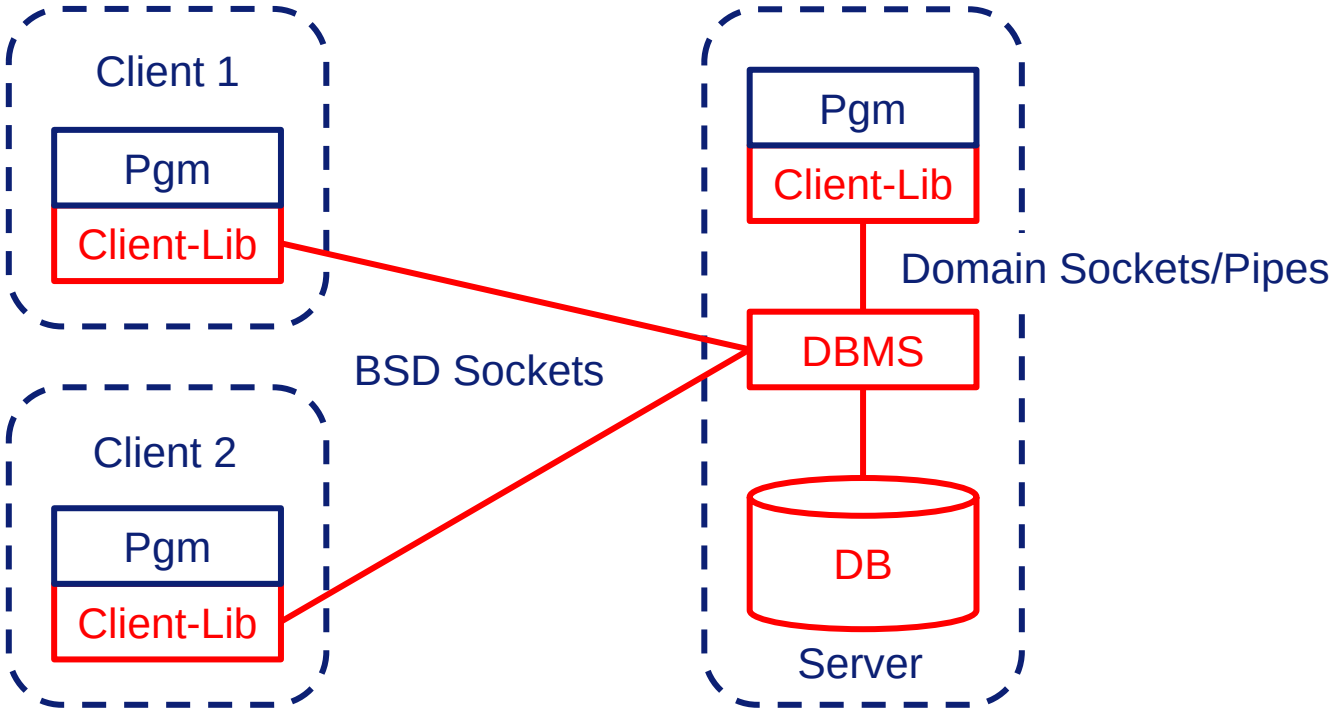
Architekturalternativen

- Desktopsysteme
  - Direktzugriff über dazugelinkte DBMS-Library
  - Mehrbenutzersynchronisation nur über primitives File-Locking
  - Single-User Systeme (z.B. MS Access, SQLite)



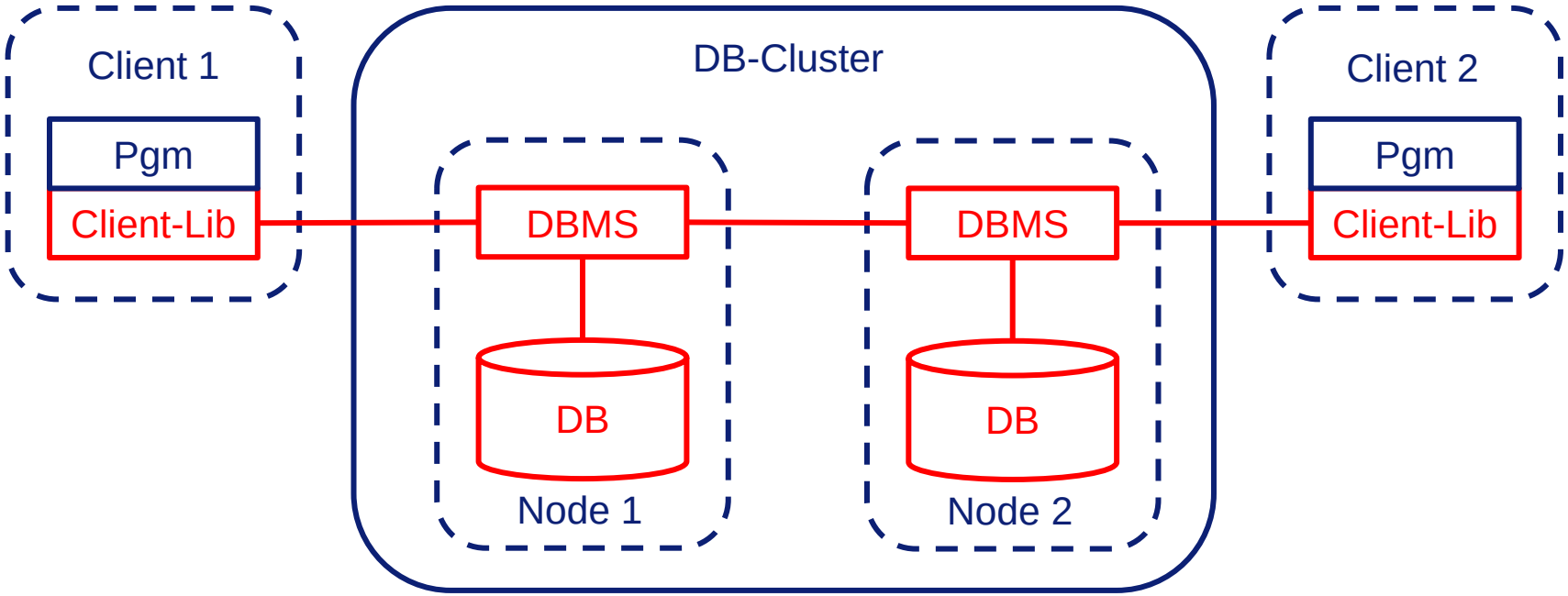
Architekturalternativen

- Client-Server-DBS
  - DBMS ist eigenständiger Prozess
  - Applikationszugriff über Client-Bibliotheken und IPC-Mechanismen
  - Multi-User-Systeme



Architekturalternativen

- Verteiltes Datenbanksystem
  - Datenbasis und DBMS-Funktionalität auf mehrere Rechner (Knoten) verteilt
  - Knoten bilden einen DB-Cluster
  - Partitionierung/Sharding und Replikation
  - "Big Data"



Relationale DBMS-Produkte

Name	Hersteller	Bemerkung
Oracle	Oracle	Marktführer, sehr teuer, viele Highend-Features, alle Plattformen (AS400, Unix, Windows)
Informix	Informix	von IBM gekauft
DB2	IBM	
Interbase	Borland	jetzt als "Firebird" Open Source
Sybase	Sybase	von SAP übernommen
Access	Microsoft	Desktop-Datenbank mit umfangreicher Benutzeroberfläche
MS SQL-Server	Microsoft	nur Windows
MySQL	freie Software	nach Übernahme durch Oracle neuer Fork MariaDB
PostgreSQL	freie Software	
SQLite	freie Software	sehr verbreitet als embedded SQL Datenbank