1. Architekturen von DB-Systemen

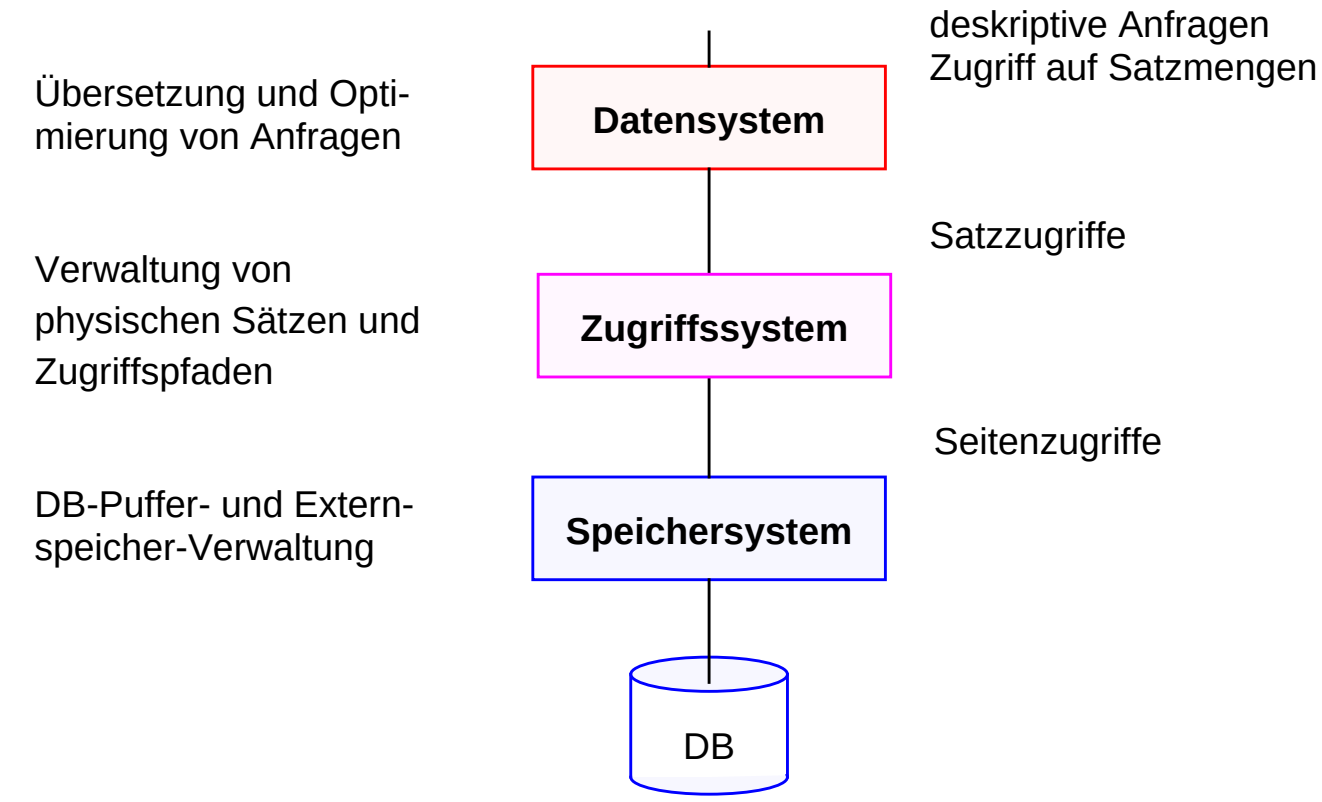
- **Schichtenmodell eines DBS**
  - Grundlegende Konzepte zentralisierter DBS (Wiederholung aus DBAW)
  - Schrittweise Abstraktion von einem Bitstring auf der Magnetplatte zu logischen Sichten und SQL-Operationen oder komplexen Objekten
  - Prinzipien der Schichtenbildung
  - Effiziente (dynamische) Abbildung über mehrere Schichten hinweg?
- **DBS-Caching für Web-Anwendungen**
  - Verbesserung von Skalierbarkeit und Leistungsverhalten von DB-basierten Web-Anwendungen
  - Einsatz von DBS in der “Nähe” von Applikations-Servern
- **Verteilte DBS**
  - Aspekte/Prinzipien der Verteilung
  - Verteilung unter Kontrolle des DBS (Mehrrechner-DBS)
- **Schichtenmodelle für Client/Server-DBS**
  - Anforderungen und Verarbeitungscharakteristika
  - „Verschicken von Anfragen” vs. „Verschicken von Daten”
  - Client/Server-Architekturen für objektorientierte DBS
- **DBS-Einsatz in Entwurfsumgebungen**
  - Verarbeitung in Entwurfssystemen - prinzipielle Schnittstellen
  - Arbeitsplatzorientierte DBS-Architekturen
  - Verarbeitungsprinzip: Laden, Verarbeiten, Integrieren
  - Nutzung von Entwurfstransaktionen mit modifizierten ACID-Eigenschaften

Schichtenmodell eines DBS – Überblick

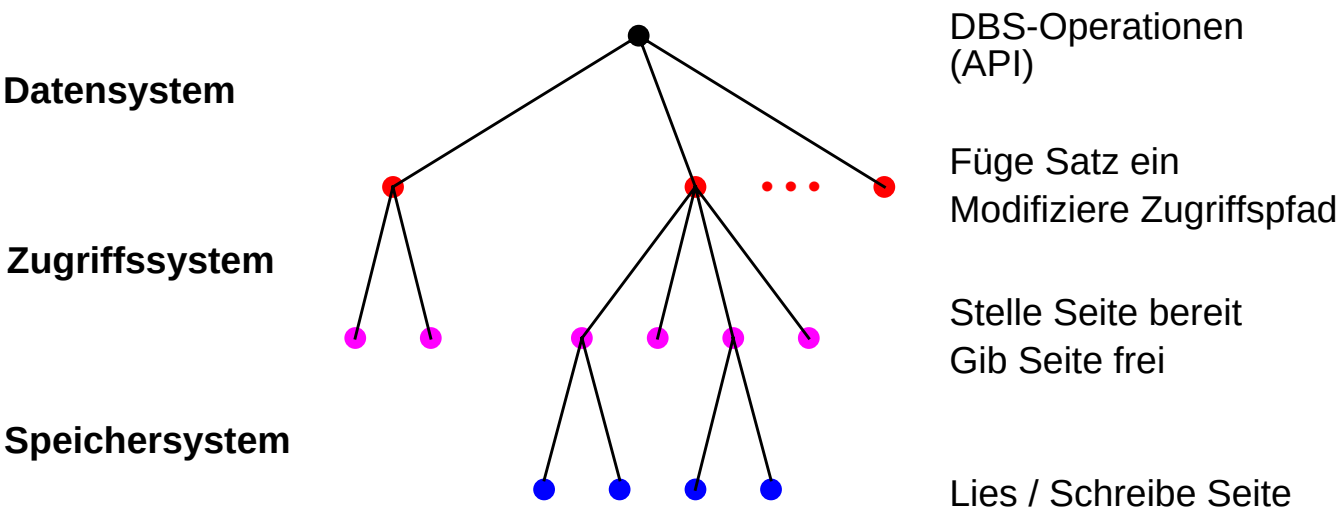
- **Vereinfachtes Schichtenmodell**

Aufgaben der Systemschicht

Art der Operationen an der Schnittstelle



- **Dynamischer Kontrollfluß einer Operation an das DBS**



Schichtenbildung im Schichtenmodell

- **Ziel: Architektur eines datenunabhängigen DBS**

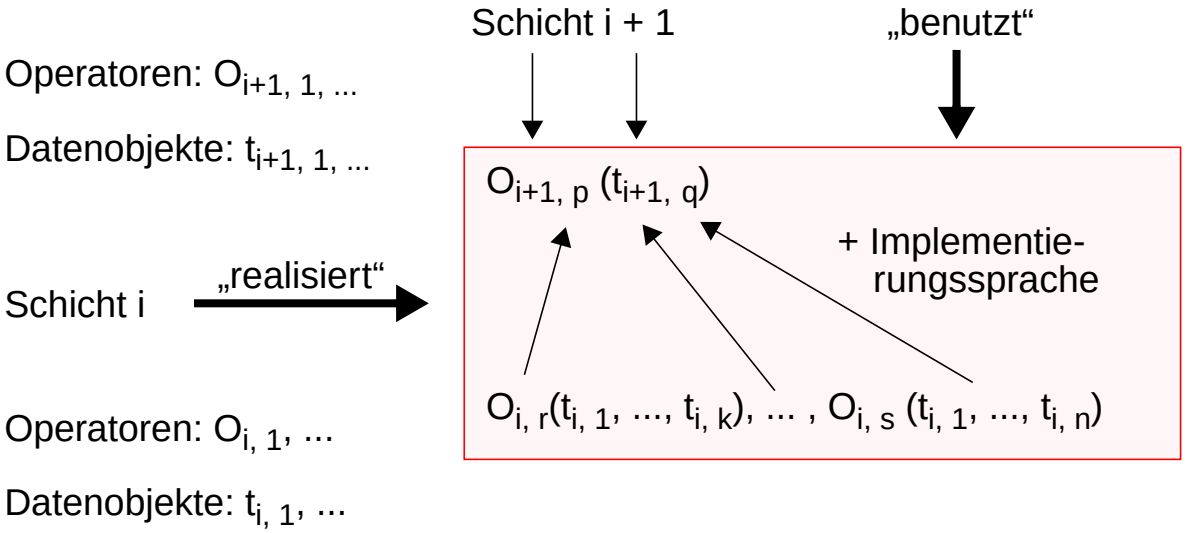
- **Wie viele Schichten braucht man?**

  ➥ Es gibt keine Architekturlehre für den Aufbau großer SW-Systeme

- **Empfohlene Konzepte:**

  - Geheimnisprinzip (*Information Hiding*)
  - Hierarchische Strukturierung

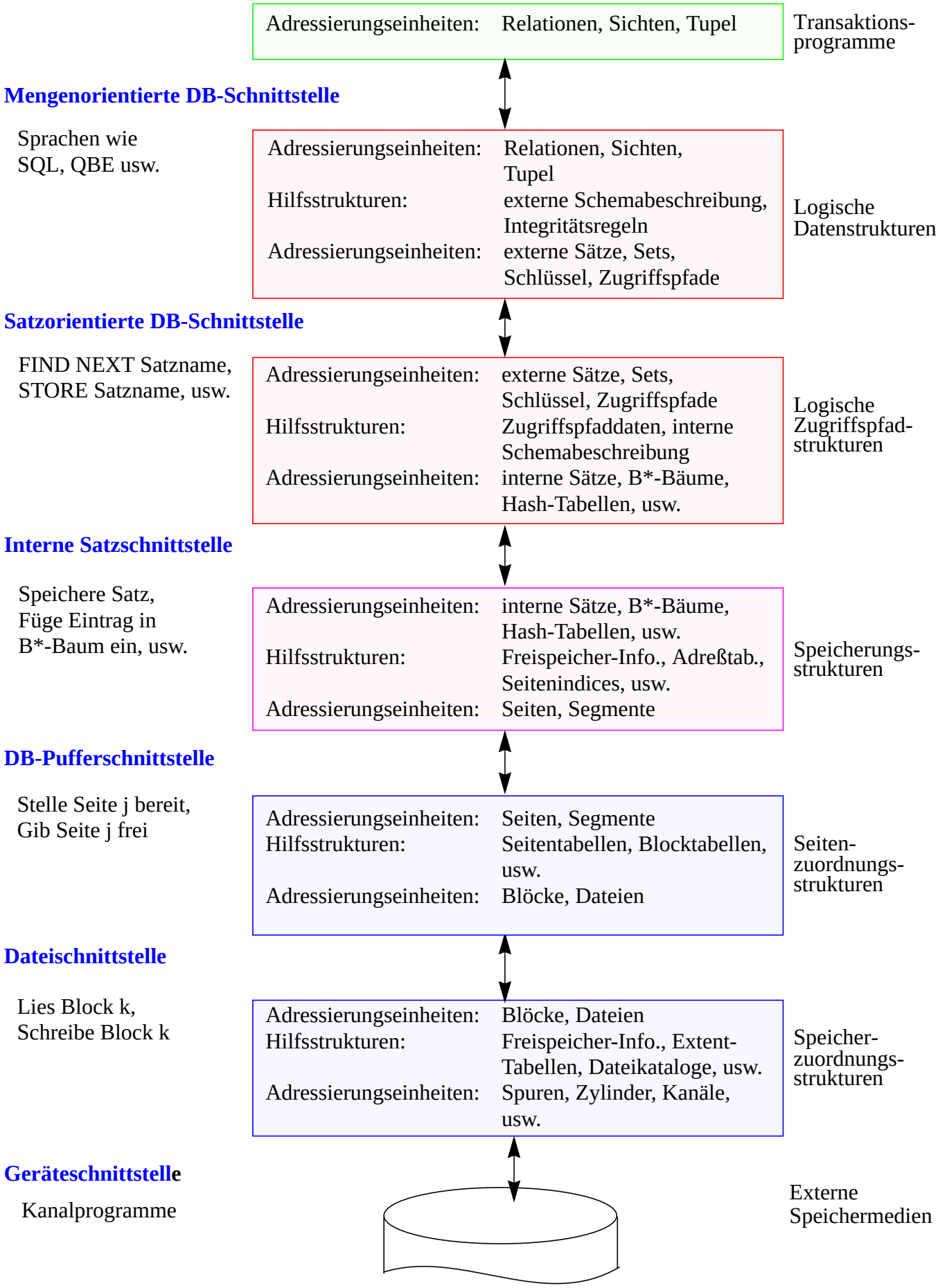
- **Aufbauprinzip:**



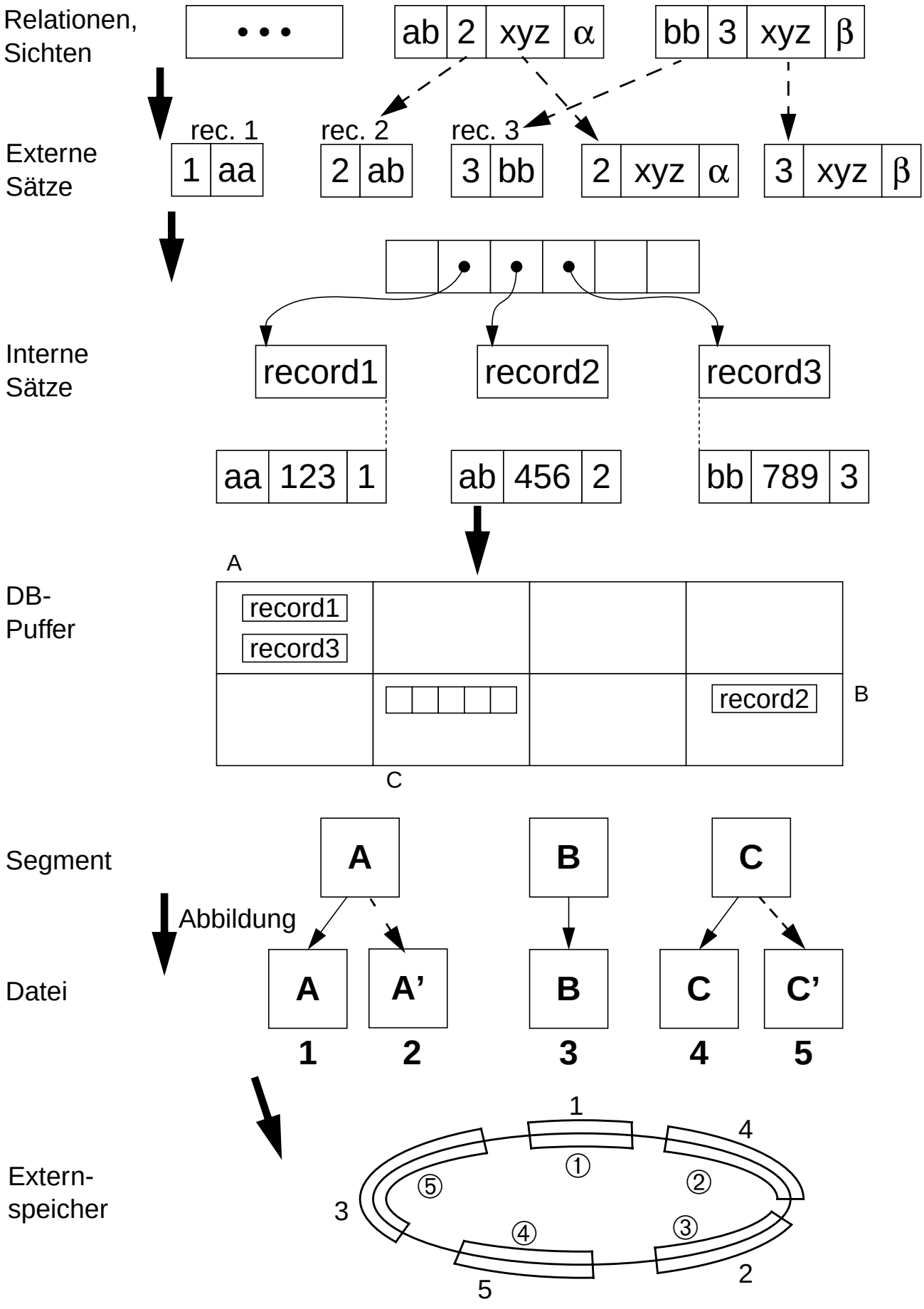
- **„benutzt“-Relation:**

  A **benutzt** B, wenn A B aufruft und die korrekte Ausführung von B für die vollständige Ausführung von A notwendig ist

Statisches Modell eines Datenbanksystems



Schichtenweise Abbildungen in einem DBS

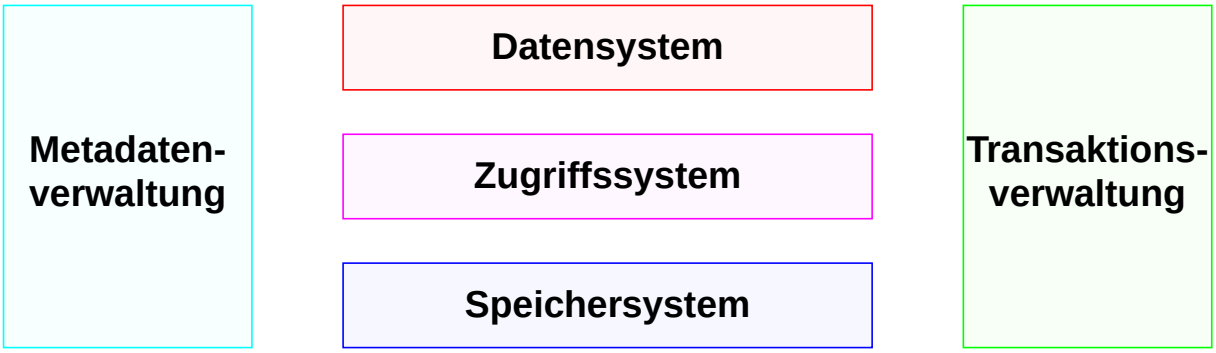


Datenunabhängigkeit im Überblick

Ebene	Was wird verborgen?
Logische Datenstrukturen	Positionsanzeiger und explizite Beziehungs-konstrukte im Schema
Logische Zugriffspfade	Zahl und Art der physischen Zugriffspfade; interne Satzdarstellung
Speicherungsstrukturen	Pufferverwaltung; Recovery-Vorkehrungen
Seitenzuordnungsstrukturen	Dateiabbildung, Recovery-Unterstützung durch das BS
Speicherzuordnungsstrukturen	Technische Eigenschaften und Betriebs-details der externen Speichermedien

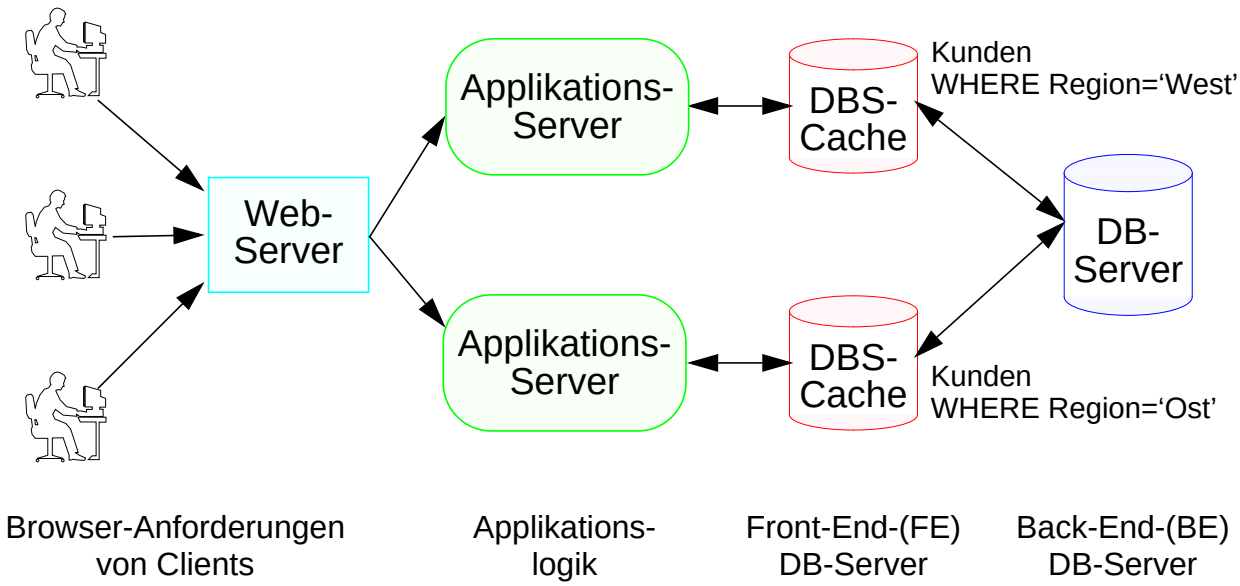
Grobarchitektur eines DBS - weitere Komponenten

- **Verwaltung der Daten, die Daten beschreiben**
  (DB-Katalog, Integritätsbedingungen, Zugriffskontrollinformationen, ...)
- **Realisierung der ACID-Eigenschaften**
  (Synchronisation, Logging/Recovery, Integritätssicherung)



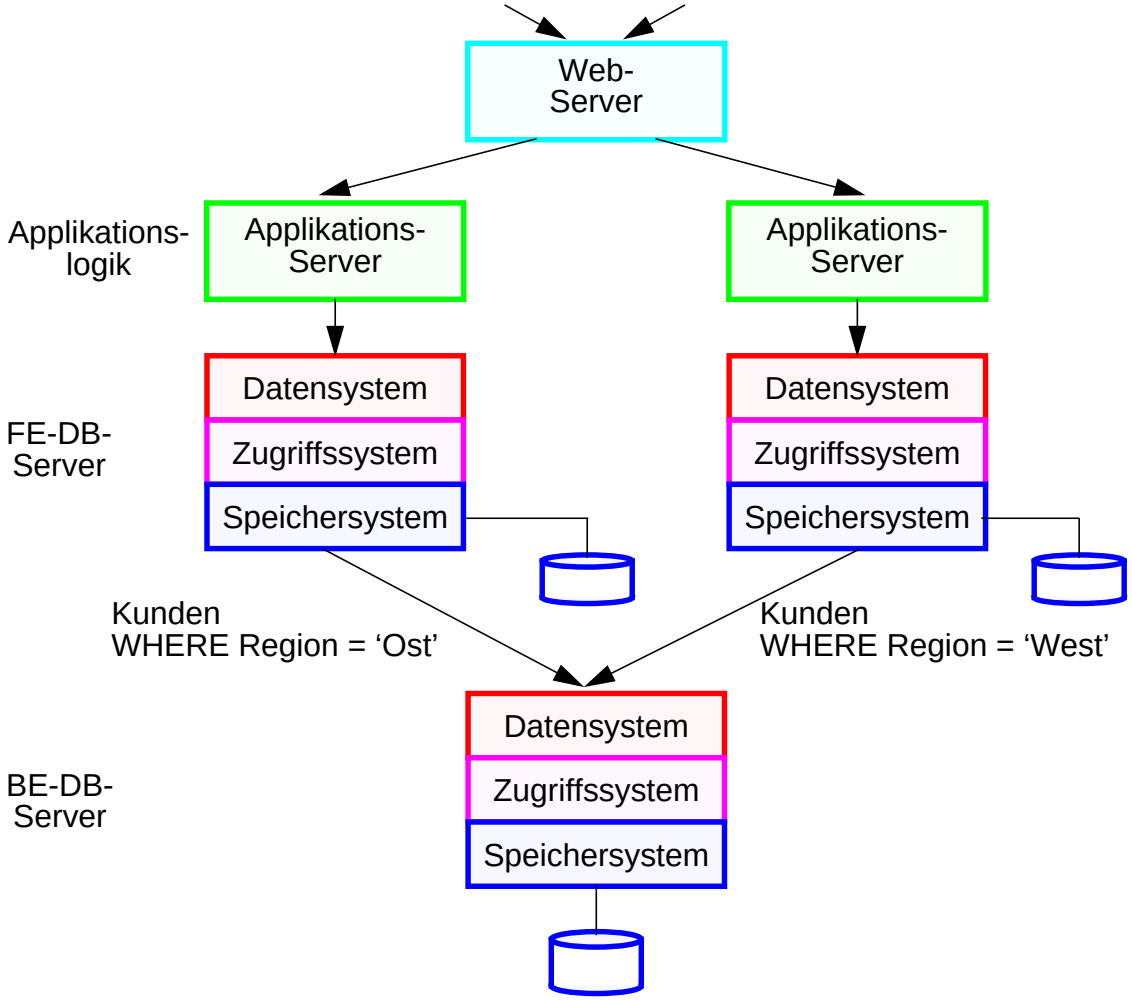
DBS-Caching

- **Caching von DB-Daten in Web-Anwendungen**
  - Caching ist die **bewährte Technik**, um Skalierbarkeit und Leistungsverhalten in großen, verteilten Systemen zu verbessern!
  - Vorgenerierung von Dokumenten mit dynamischen Inhalten
  - Inkaufnahme von veralterter Information (Inkonsistenzen)
  - Information des Benutzers durch Angabe der Generierungszeit
  - Caching-Verfahren
    - Passives oder Exact-Match Caching von Anfrageergebissen (query result (QR) caching) in Form von HTML- oder XML-Fragmenten
    - Statisches Caching (Replikation) von ganzen Tabellen, materialisierten Sichten oder Sub-Tabellen (spezifiziert über einfache Prädikate)
    - **Deklaratives Caching** erfordert BE-Metadaten, um den Cache dynamisch zu laden (Caching mehrerer QRs in gemeinsamen Tabellen)
    - **On-Demand Caching** verwendet BE-Metadaten und „Hinweise“ der TA, um Daten dynamisch in den Cache zu füllen oder zu ersetzen



  - **DBS-Caching** „in der Nähe“ des Applikations-Servers
    - Beschleunigung des lesenden DB-Zugriffs
    - (bislang noch) Weiterleitung von Änderungsanweisungen zum BE
    - Konsistenzprobleme

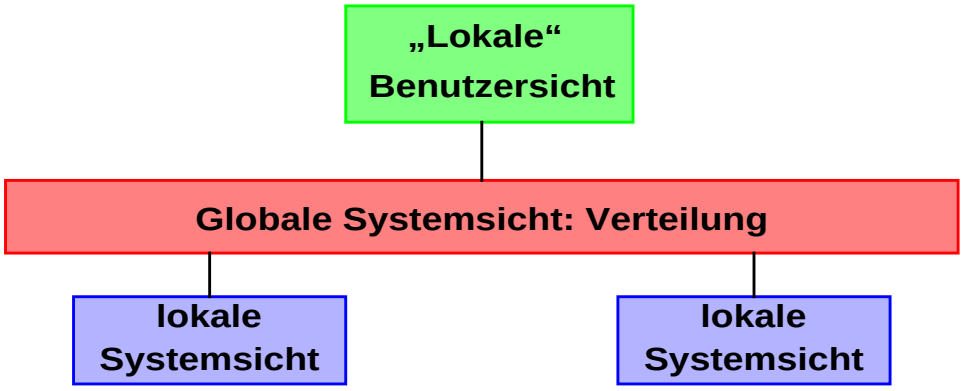
DBS-Caching (2)



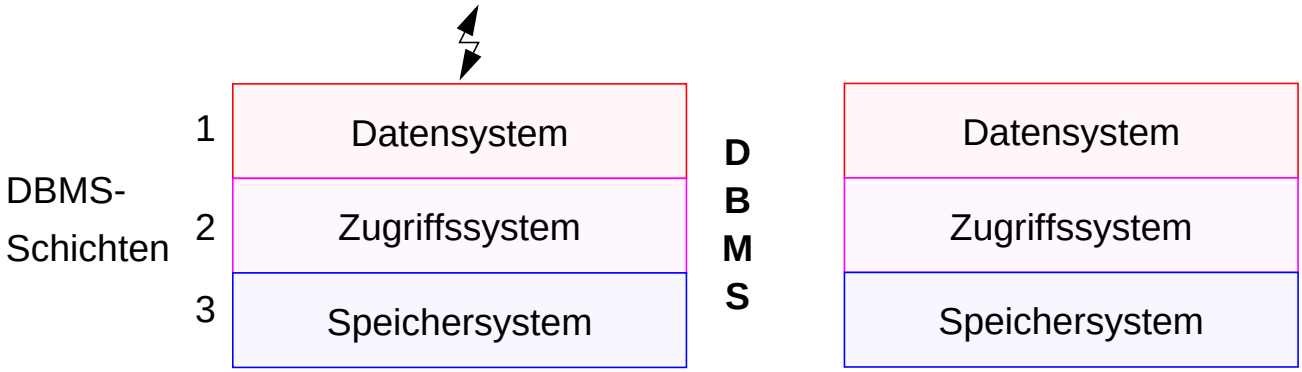
- **Caching von DB-Daten (Forts.)**
  - Wünschenswert: DBS-Caching mit
    Abwicklung von Anfragen mit einfachen Prädikaten und n Verbunden
  - Spalten von Tabellen können im DBS-Cache mit der Eigenschaft „bereichs-vollständig“ (domain complete) versehen werden,
    d. h., wenn ein Spaltenwert im Cache gefunden wird, garantiert der Cache-Mgr, daß alle Sätze (Zeilen) mit diesem Wert sich im Cache befinden.
    UNIQUE-Spalten sind somit immer bereichsvollständig
  - „Verknüpfung“ von Tabellen im Cache durch Referential Cache Constraints
    - Wichtigster Fall: Primär-/Fremdschlüsselbeziehungen (Owner-Member)
    - Wenn ein PS-Wert im Cache gefunden wird, garantiert der Cache-Mgr, daß **alle Sätze** mit dem gleichen FS-Wert im Cache sind
    - So lassen sich sog. **Cache Groups** bilden, die Verbundoperationen im Cache unterstützen

Klassifikation verteilter Datenbanksysteme

- **Vorgehensweise bei der Verteilung**
  - Partitionierung der Daten (ggf. mit Replikation)
  - Kommunikation zwischen Prozessen zum Zugriff auf jeweils lokalen Daten
- **Gewünschte Sichtbarkeit**

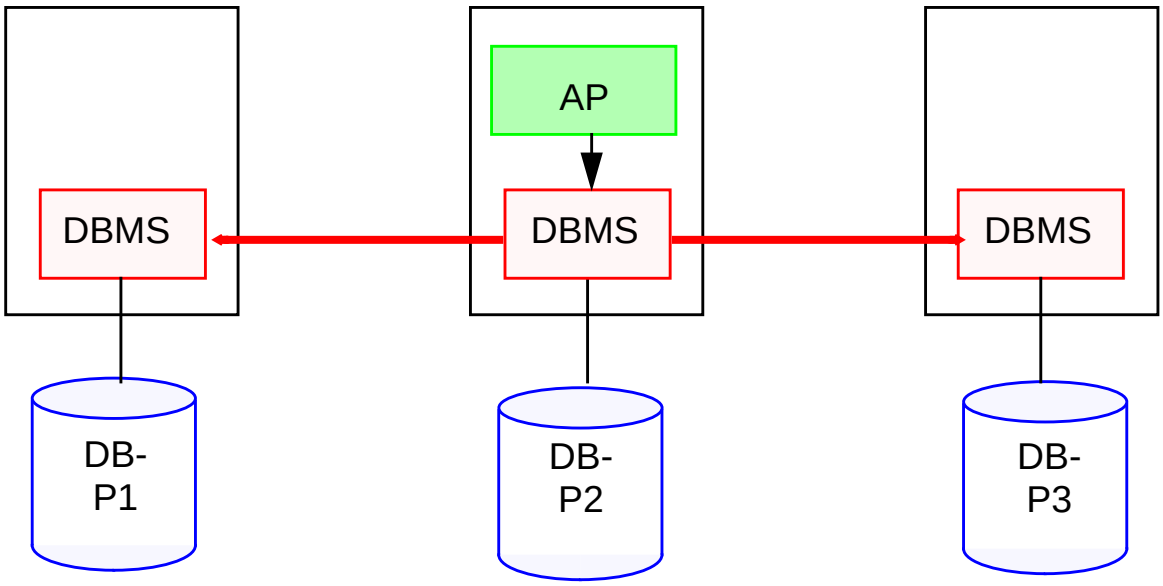


- **Homogenes Systemmodell**



- **Mehrere (homogene) Realisierungsalternativen**
  - Globale Systemsicht wird in einer (zusätzlichen) DBMS-Schicht hergestellt
  - Verteilung ganzer DML-Befehle bzw. von Teiloperationen (Schicht 1)
  - Verteilung von internen Satzoperationen (Schicht 2)
  - Verteilung von Seitenzugriffen (Schicht 3)

Mehrrechner-DBS



- **Verteilung unter Kontrolle der DBMS**
  - Kooperation zwischen (homogenen) DBMS
  - Ortstransparenz für AP (ein DB-Schema)*: single system image*
  - Flexibilität für Daten- und Lastverteilung

- **Architekturklassen**

  1. DB-Partitionierung (*Shared Nothing*, VDBS)

  - Jeder Knoten besitzt volle Funktionalität und verwaltet eine DB-Partition
  - Datenreplikation erhöht Verfügbarkeit und Zuverlässigkeit
    - Minimierung der Auswirkung von „entfernten“ Fehlern,
    - Fehlermaskierung durch Redundanz
  - Verarbeitungsprinzip: **Die Last folgt den Daten**

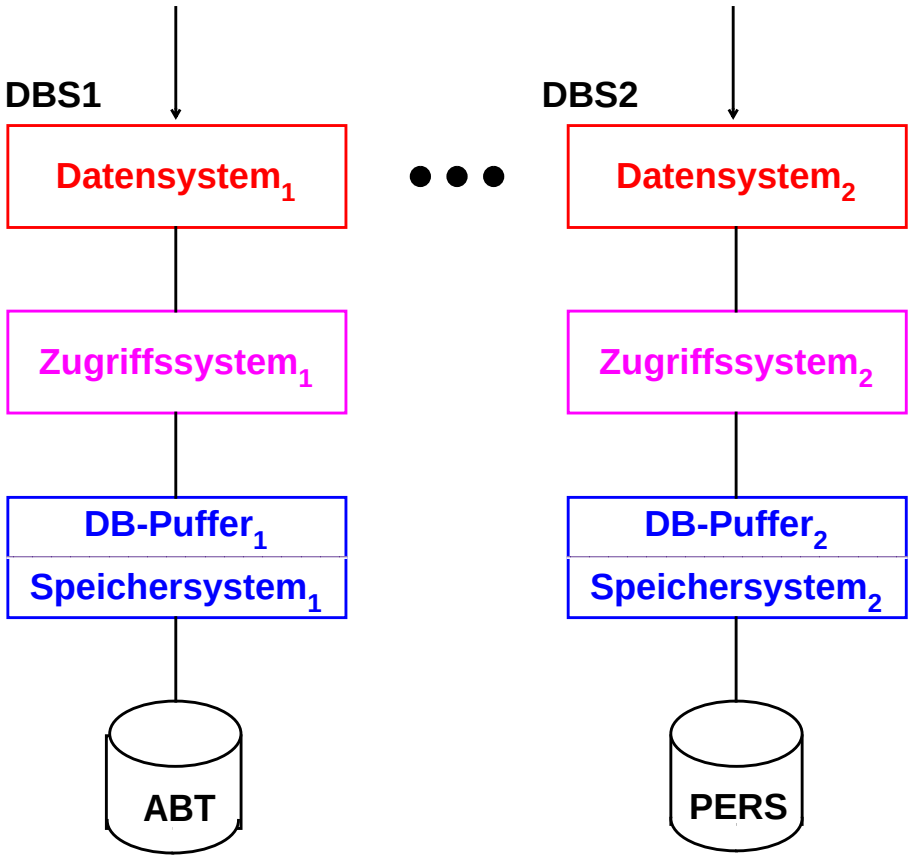
  2. DB-Sharing (*Shared Disk*)

  - Lokale Ausführung von DML-Befehlen
  - Verarbeitungsprinzip: **Datentransport zum ausführenden Rechner**
  - Lokale Erreichbarkeit der Externspeicher

Verarbeitungsaufwand bei VDBS

Q_1: SELECT ...
FROM PERS P
WHERE P.ANR = 'K55'
AND P.GEH < P.PROV

Q_2: SELECT ...
FROM PERS P, ABT A
WHERE P.ANR = A.ANR
AND P.ANR = 'K55'



Function Request Shipping

	nach DBS2	nach DBS1
→	gesamtes Q1	A.ANR = 'K55' nach DBS1
←	Q1-Resultat	Resultat: 1 'K55'-Satz
	Scan für alle Sätze mit P.ANR = 'K55'	**Scan für alle Sätze mit A.ANR = 'K55'**
→	nächsten Satz anfordern	nächsten Satz anfordern
←	Satz übertragen	Satz übertragen
	für alle Seiten von PERS-Datei:	**für alle Seiten von ABT-Datei:**
→	nächste Seite anfordern	nächste Seite anfordern
←	Seite übertragen	Seite übertragen

Annahmen:

- PERS-Datei habe 10^5 Seiten
- P.ANR = 'K55': 100 Sätze in PERS
- ABT-Datei habe 10^3 Seiten
- P.ANR = 'K55' AND P.GEH < P.PROV: 5 Sätze in PERS

Schichtenmodelle für Client/Server-DBS

- **Entscheidend für die DB-bezogene Leistungsfähigkeit**
  - Art und Komplexität der DB-Operationen (mengenorientiert, navigierend)
  - Nutzung der Referenzlokalität bei den Datenzugriffen

- **Bisheriges Verarbeitungskonzept**
  - Bei allen Architekturvorschlägen wurde ein „Verschicken der Anfragen“ zum (Server)-DBMS unterstellt (query shipping approach):
  - Die DML-Anweisung wird zum Server geschickt
  - Nach ihrer Verarbeitung wird das Ergebnis der AW (Client) zur Verfügung gestellt
  - **Wichtige Eigenschaft:** Es gibt keine Replikation von DB-Daten

  ➥ keine Nutzung vorhandener Referenzlokalität im Client

- **Weiteres Verarbeitungskonzept**

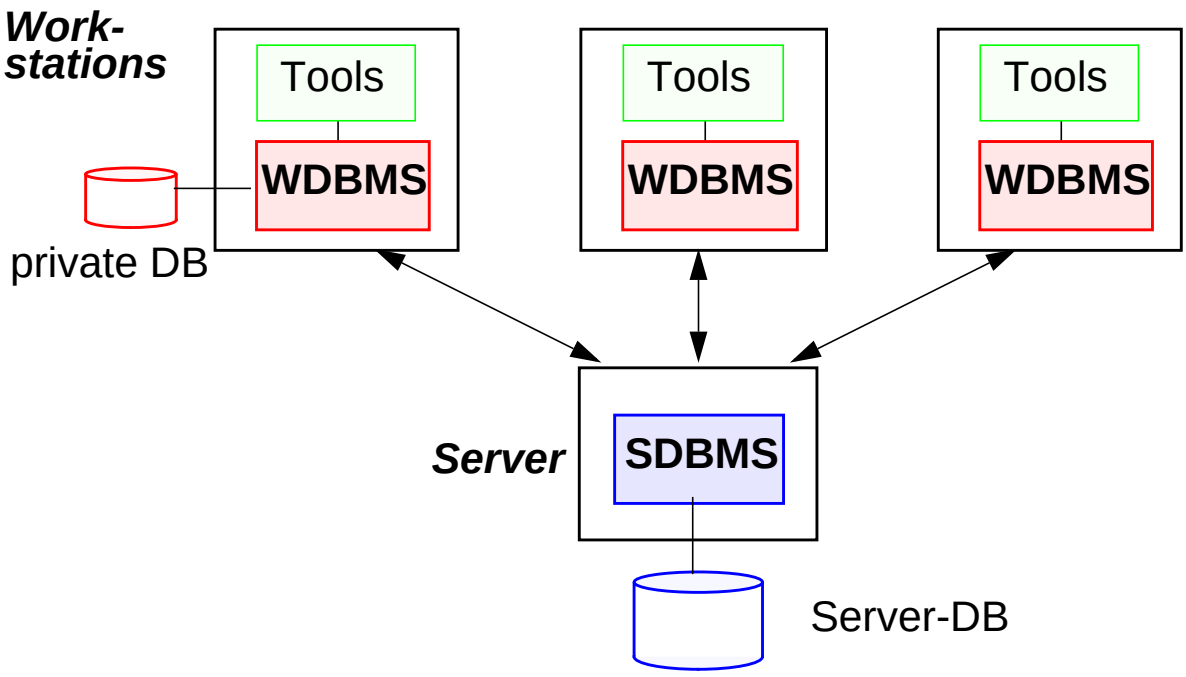
  Folgende Vorgehensweise wird von den meisten Objektorientierten DBS (OODBS) verfolgt (data shipping approach):
  - Alle Daten, die zur Ausführung einer Anfrage benötigt werden, werden zum Client geholt und dort gepuffert
  - Danach wird die Anfrage (oder mehrere Anfragen hintereinander) im Client-Puffer ausgewertet
  - **Achtung:** Die Komplexität der Anfragen ist stark eingeschränkt
    - Typisch sind Navigationsoperationen
    - Anfragen mit einfachen Suchargumenten (SSA: simple search arguments) werden manchmal unterstützt, jedoch keine Verbund-operationen, Aggregationen, ...

  ➥ Dieses Konzept ist vor allem bei hoher Referenzlokalität vorteilhaft. Es wird durch das Checkout/Checkin-Konzept genutzt.

Schichtenmodelle für Client/Server-DBS (2)

- **Vorteile des „Verschickens von Daten“**
  - Lokale Datenpufferung reduziert die Interaktionen zwischen Client und Server (Verminderung der Netzbelastung; Vermeidung von wiederholten Server-Anfragen)
  - Aggregierte Rechnerleistung und die insgesamt verfügbare Speicherkapazität des Systems erhöht sich
  - Server-Last wird reduziert
  - Skalierbarkeit des Systems wird verbessert

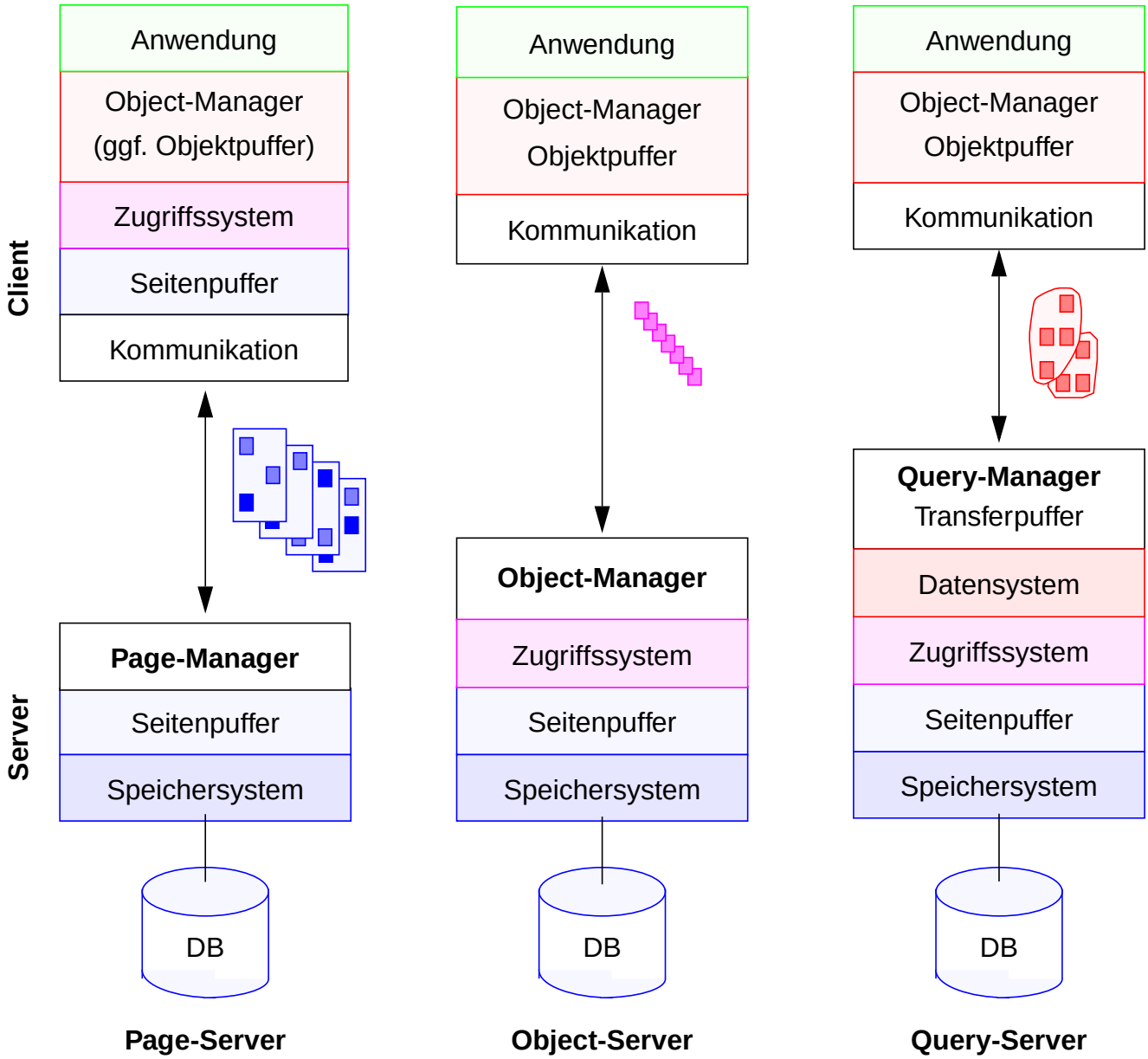
- **Client/Server-Architektur: Prinzip**



  - Lokale Platten für private DB und Log-Daten
  - Steigerung der Leistungsfähigkeit:
    verteiltes Server-System, parallele DB-Verarbeitung

Schichtenmodelle für Client/Server-DBS (3)

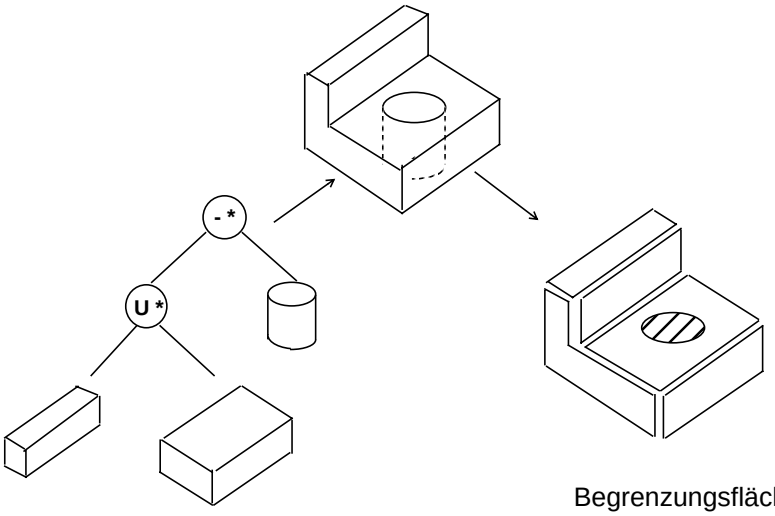
- **Client/Server-Architekturen für objektorientierte DBS**
  - File-Server, Page-Server
  - Object-Server
  - Query-Server



➥ Im Objektpuffer erfolgt typischerweise navigierende Verarbeitung!

Verarbeitung in Entwurfssystemen - Beispiel

- **Konstruktion von zusammengesetzten Körpern**

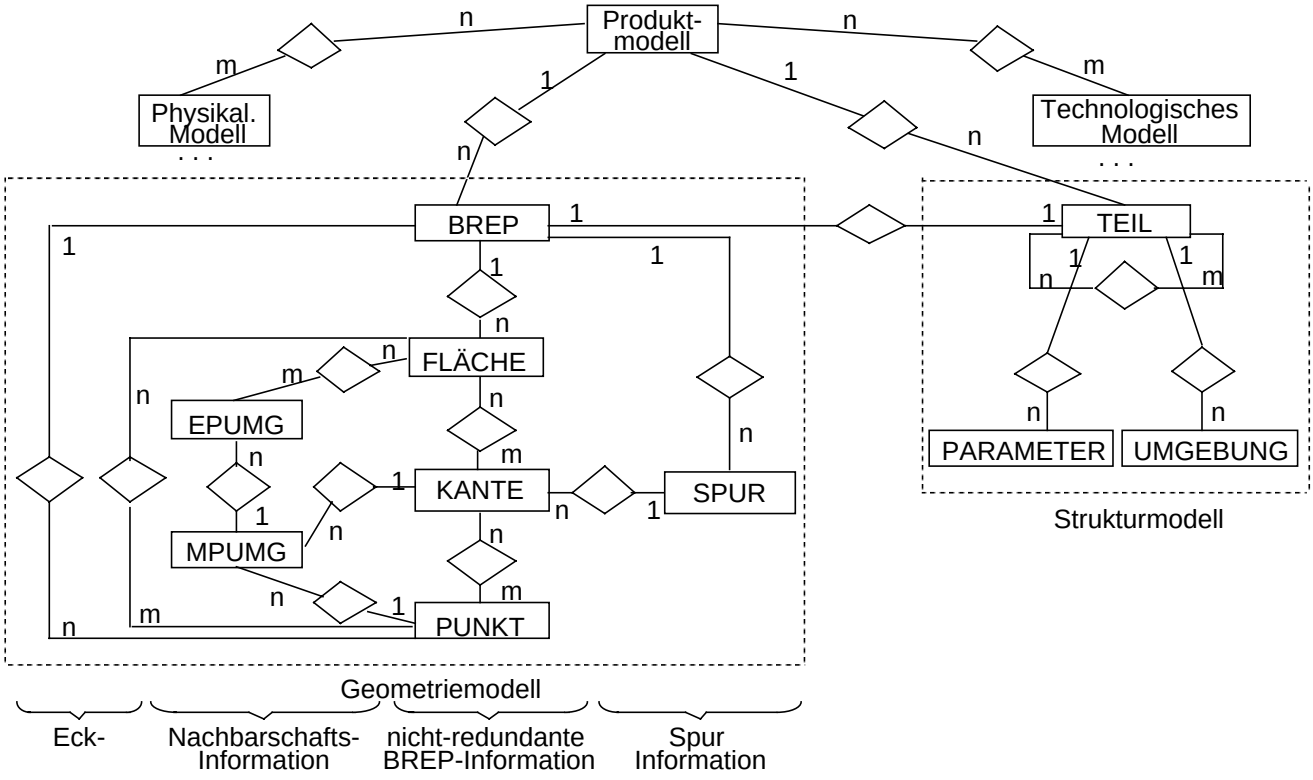


Konstruktionsbaum

Begrenzungsflächen-Darstellung
(vereinfacht)

➥ Sicht des Konstrukteurs an der Dialog-Schnittstelle
(mit Anwendungsmodell-Operationen)

- **ER-Darstellung des Produktmodells für Werkstücke**



➥ Abstrakte Sicht der Datenmodell-Schnittstelle

Verarbeitung in Entwurfssystemen - Beispiel (2)

- **Abbildung der BREP-Information ins Relationenmodell (Ausschnitt)**

BREP (<u>BID</u>, . . .)

FLÄCHE (<u>FID</u>, . . . , BID)

FL-KA (<u>FID, KID</u>, . . .)

KANTE (<u>KID</u>, KLÄNGE, . . .)

KA-PKT (<u>KID, PID</u>, . . .)

PUNKT (<u>PID</u>, . . .)

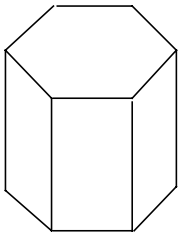
a) DB-Schema

```
SELECT PID, X-KOORD, Y-KOORD, Z-KOORD
FROM PUNKT
WHERE PID IS IN
    SELECT PID
    FROM KA-PKT
    WHERE KID IS IN
        SELECT KID
        FROM KANTE
        WHERE KLÄNGE > ’10’
        AND KID IS IN
            SELECT KID
            FROM FL-KA
            WHERE FID IS IN
                SELECT FID
                FROM FLÄCHE
                WHERE BID = ’4711’
```

b) Anfragebeispiel

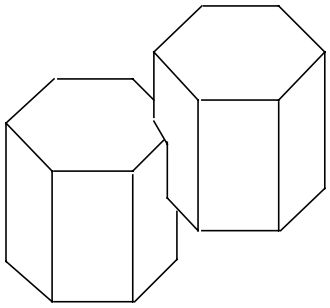
➥ Relationenmodell ist schlecht geeignet! (Objektorientierte oder objekt-relationale Modelle erlauben “natürlichere” Modellierung und Verarbeitung

- **Auswertung der Objekterzeugung und -manipulation**



POLYCYL(n)	5	25	50
Dialog-Schnittstelle	1	1	1
Anwendungsmodell-S.	64	304	604
Datenmodell-S.	1958	15078	41828

a) Generierung eines durch einen Polyeder angenäherten Zylinders



Operation	#DMLs
UNION(CYL(3), CYL(3))	16051
UNION(CYL(4), CYL(4))	20658
UNION(CYL(6), CYL(6))	31432

b) UNION von zwei parametrisierten Polyedern

➥ Nutzung vorhandener Referenzlokalität im Client erforderlich

DBS-Einsatz in Entwurfsumgebungen[1]

- **Ingenieur-AW brauchen eine lokale Datenhaltung**
  - Ausnutzung der Referenzlokalität bei der Verarbeitung
  - Lange Speicherung (Objektpuffer) großer Datenmengen im HSP
  - Lokale Darstellung der Objekte am Bildschirm
  - Schnelle Transformation und Verarbeitung komplexer Objekte
  - Erforderliche Eigenschaften auf Client-Seite, insbesondere
    - große Hauptspeicherkapazität
    - hohe Verarbeitungsleistung
    - breitbandige Datenübertragung
    - hochauflösende Darstellungsqualität
    - ggf. lokale Externspeicheranbindung

  ➥ DBS, die Entwurfsanwendungen unterstützen sollen, benötigen auf Client- und Server-Seite geeignete Funktionalität zur Datenverwaltung/Transaktionsunterstützung.

- **Neue Anforderungen**
  - Entwurfsaufgaben (CA*), wissensbasierte Anwendungen, ...
  - DB-gestützte Verarbeitung großer Datenmengen im Client
  - Lange Transaktionen
  - Große Datenmengen und Referenzlokalität

  Konsequenzen:

  - Datenverwaltung in Client und Server
  - Replikation von Daten in DB-Puffer und Objektpuffer
  - Erhöhter Aufwand für die Integritätskontrolle

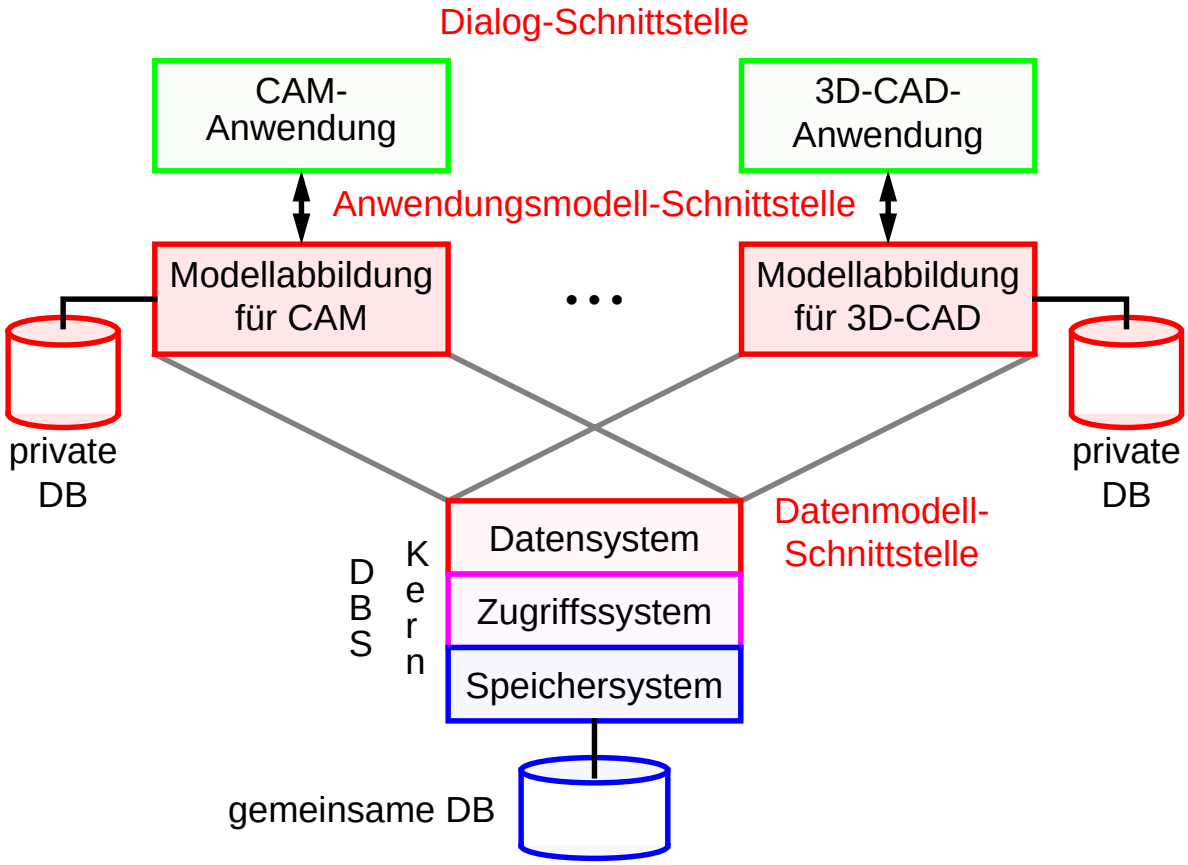
---

1. Neben dem Begriff Client/Server-Architektur wird auch der Begriff „**Workstation/Server-Architektur**“ benutzt; er soll explizit machen, daß auf Client-Seite Workstation-Eigenschaften benötigt werden.

DBS-Einsatz in Entwurfsumgebungen (2)

- **Arbeitsplatzorientierte DBS-Architekturen**
  - Sie sollen sog. Non-Standard-Anwendungen unterstützen
  - Abbildungen der DBS-Verarbeitung auf Client/Server-Architekturen

- **DBS-Kernarchitektur**



- **Trennung von**
  - **anwendungsunabhängigen** Funktionen im Server und
  - **anwendungsbezogenen** Funktionen im Client

- **Schlüsselkonzepte**
  - Datenmodell für komplexe Objekte an der Server-Schnittstelle
  - Zugeschnittenes Datenmodell an der AW-Schnittstelle
  - Objektpuffer zur Nutzung der Lokalität und zur Einsparung von Kommunikationsvorgängen

DBS-Einsatz in Entwurfsumgebungen (2)

- **Verarbeitungsunterstützung durch**
  - mengenorientierte Datenversorgung (generische Anfragesprache)
  - anwendungsbezogene Repräsentation der Objekte (Sichten)
  - flexible Integritätskontrolle (immediate, deferred)
    (Zusicherungen, Referentielle Integrität, ECA-Regeln, ...)
  - mengenorientiertes Einbringen

➥ Minimierung der Abhängigkeiten zwischen Client und Server
(Kommunikation, Fehlerisolation)

- **Hohe Leistungsanforderungen**
  - für navigierende Tool-Operationen: 10^5 Ref./sec bei CAD
  - implizieren Nutzung von Lokalität in der Nähe der Anwendung
    (alle Objekte mit Rereferenz im Objektpuffer)
  - erfordern hauptspeicherbasierte Adressierung der DB-Objekte
    (bei > 3 Referenzen: Pointer Swizzling)

- **Weitere Merkmale**
  - Lange Dauer von Entwurfsvorgängen (Wochen/Monate)
  - Wie verhält es sich mit der Synchronisation?

➥ In der Regel sind konkurrierenden Zugriffe auf Entwurfsdaten selten

  - Unterstützung von Versionen (spezielle Sperrmodi)
  - Kontrollierte Kooperation zwischen mehreren Entwerfern
    (spezielle Synchronisationsmaßnahmen oder Austauschoperationen)

Verarbeitungsprinzip:
Laden, Verarbeiten, Integrieren

- **Datenanforderung für eine Entwurfstransaktion**
  - Bereitstellung durch eine oder mehrere Checkout-Anforderungen
  - Entwurfsobjekte werden in den Objektpuffer eingelagert
  - Einsatz spezieller Speicherungsstrukturen und Zugriffspfade

- **Datenmanipulation**
  - bei komplex-strukturierten Objekten: Einsatz hierarchischer Cursor
  - Gesamte Objektverarbeitung ist lokal: Akkumulation von Änderungen
  - Anwendungsmodell: Menge von ADTs

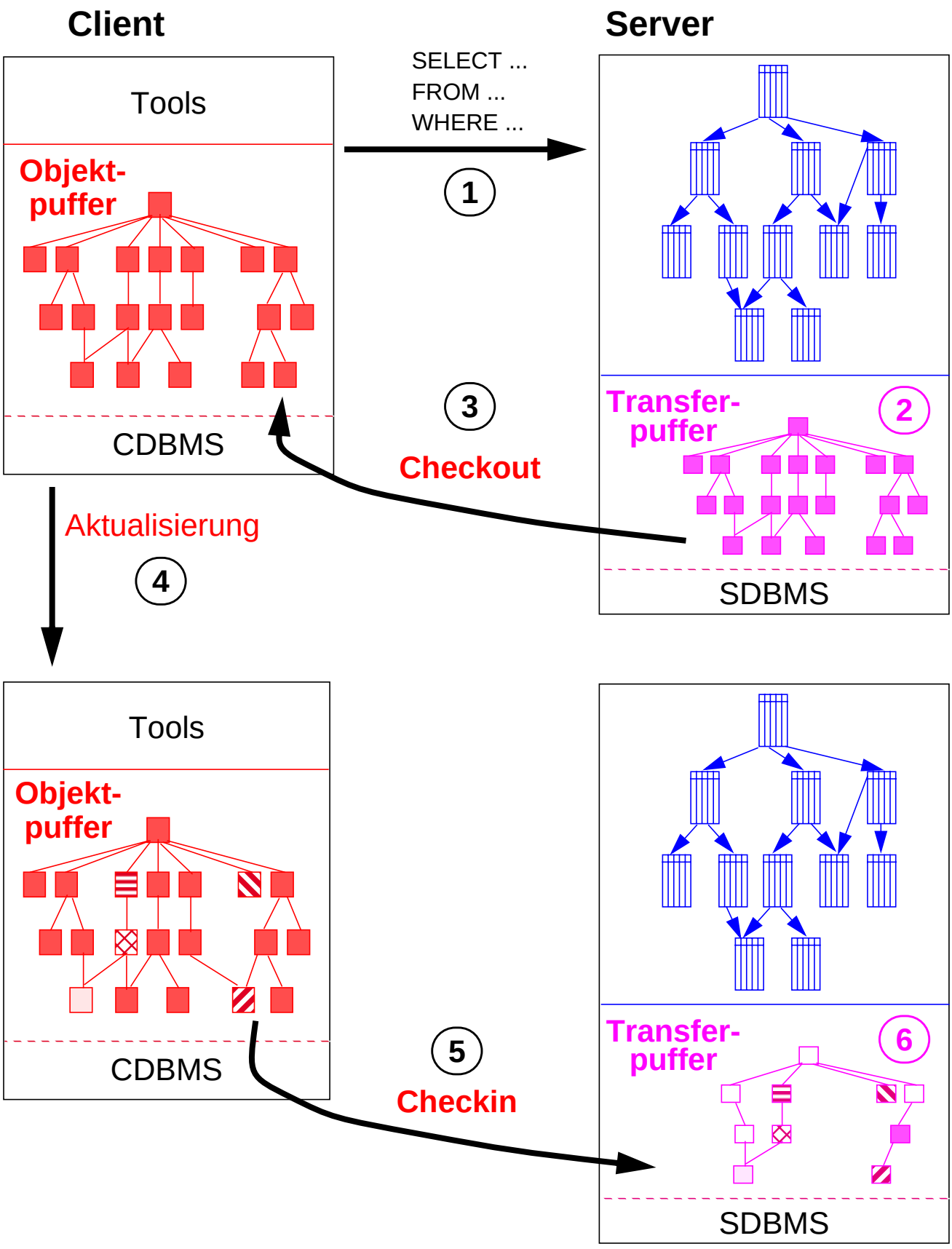
- **Verarbeitungskontrolle und Recovery-Maßnahmen**
  - Client-DBS schreibt persistente Zwischenzustände in private DB
  - Lokale Rücksetzmaßnahmen durch Savepoints (SAVE, RESTORE)
  - Unterbrechung der Verarbeitung (SUSPEND, RESUME)

➥ Wie verhält es sich mit den ACID-Eigenschaften bei Entwurfstransaktionen?

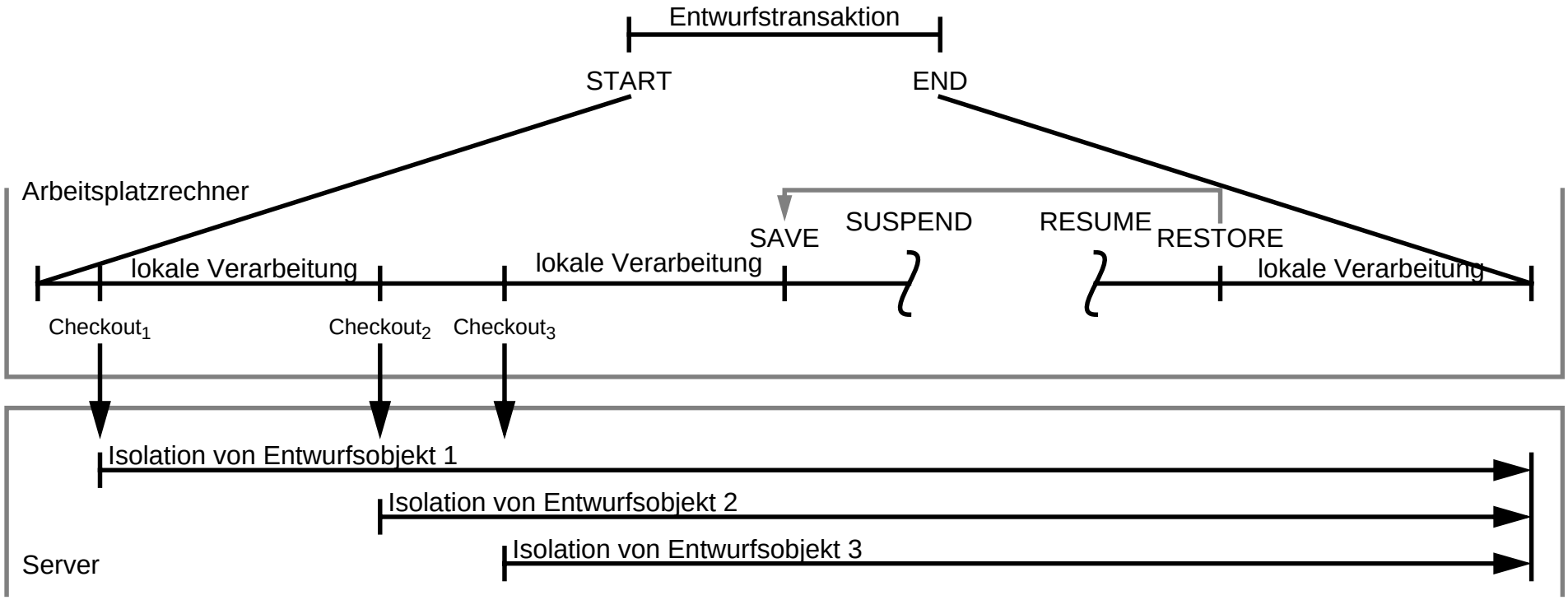
- **Datenpropagierung**
  - durch ein Checkin am Ende der Entwurfstransaktion,
  - dabei erfolgt (verzögerte) Überprüfung von Integritätsbedingungen in der Server-DB

➥ Was passiert beim Entdecken von Integritätsverletzungen?

Laden, Verarbeiten, Integrieren



Modell einer Entwurfstransaktion



Charakteristika: 0 .. n Checkout
0 .. 1 Checkin
Lange Dauer

Speicherung von Zwischenzuständen einer Entwurfstransaktion zum:

- Unterbrechen der Verarbeitung (SUSPEND, RESUME)
- Rücksetzen auf frühere Verarbeitungszustände (SAVE, RESTORE)

Zusammenfassung

- **Allgemeine Aspekte des Schichtenmodells**
  - Schichtenmodell ist allgemeines Erklärungsmodell für die DBS-Realisierung
  - Schichtenbildung läßt sich zweckorientiert verfeinern/vergröbern: Anwendbarkeit für Verteilte DBS, Client/Server-DBS, ...
  - Entwurf geeigneter Schnittstellen erfordert große Implementierungserfahrung
  - Konkrete Implementierungen verletzen manchmal die Isolation der Schichtenbildung aus Leistungsgründen ($\rightarrow$ kritische Funktionen haben „Durchgriff“)
- **Implementierungskonzepte**
  - Die grundlegenden Implementierungskonzepte zentralisierter DBS finden sich auch in jedem Knoten eines verteilten DBS
  - Die Kenntnis der Implementierungskonzepte ermöglicht es, existierende DBS objektiv zu beurteilen und zu vergleichen
  - Ihre genaue Kenntnis ist Voraussetzung für Leistungsanalysen bzw. Abschätzungen des Systemverhaltens
  - Durch Identifizieren weniger, grundlegender Konzepte gelangt man zu einem tieferen Verständnis dessen, was mit **„Datenunabhängigkeit“** gemeint ist
- **DBS-Caching**
  - Caching vs. Replikation zur Unterstützung Web-basierter DB-AW
  - Caching ist die **bewährte Technik,** um Skalierbarkeit und Leistungsverhalten in großen, verteilten Systemen zu verbessern!
  - Spalten von Tabellen können im DBS-Cache mit der Eigenschaft „bereichsvollständig“ (domain complete) versehen werden
  - **Cache Groups** unterstützen Verbundoperationen beim DBS-Caching

Zusammenfassung (2)

- **Klassifikation Verteilter DBS**
  - Single System Image
  - Flexibilität für Daten- und Lastverteilung
  - Mehrrechner-DBS als **Shared-Nothing oder Shared-Disk-Systeme**
- **Client/Server-Systeme**
  - Data Shipping vs. Query Shipping
  - Data Shipping ist vor allem bei hoher Referenzlokalität vorteilhaft. Es wird durch das Checkout/Checkin-Konzept genutzt.
  - Objektorientierte DBS als **Page-, Object- oder Query-Server**
  - Typische Verarbeitungsform im Client ist navigierend (für eingebettete Systeme und Entwurfssysteme)
  - Änderung der DB-Daten im Client und Server erforderlich
- **DB-Einsatz in Entwurfsumgebungen**
  - Prinzipielle Schnittstellen: Dialog-, Anwendungsmodell-, Datenmodell-Schnittstellen
  - Ingenieur-AW brauchen große Datenmengen (lokale Pufferung der von der AW benötigten DB-Objekte, Verarbeitung dieser Objekte in HSP-Strukturen, Objektdarstellung am Bildschirm)
  - **Nutzung von Lokalität** in einem Objektpuffer im Client
  - Verarbeitungsprinzip: Laden, Verarbeiten, Integrieren
  - Effiziente Anbindung der Werkzeuge
  - Neue Transaktionsmodelle für Entwurfsanwendungen