Prof. Dr. Th. Härder
Universität Kaiserslautern
Fachbereich Informatik
E-mail: haerder@informatik.uni-kl.de
WWW: http://wwwdbis.informatik.uni-kl.de/

Vorlesung

Realisierung von Datenbanksystemen

SS01

Th. Härder

Vorlesung:

Mo., 10.00 - 11.30 Uhr, 46-280

Beginn: Mo., 23. 4. 2001

Ziele

- **Vermittlung von vertieften Kenntnissen**
  zu Entwurf, Aufbau, Realisierung und Programmierung von DBS, insbesondere
  - Architektur von zentralisierten und verteilten DBS (Schichtenmodelle),
  - Speichersystem, Zugriffssystem und Datensystem
  - Implementierungstechniken für Konzepte und Verfahren, mit denen die einzelnen Schichten eines DBS aufgebaut werden

- **Erarbeitung eines tiefergehendes Verständnisses**
  - für das Zusammenspiel der DBS-Schichten und -Komponenten
  - zur Erklärung des Leistungsverhaltens bei der Abwicklung von DB-Operationen
  - für das Entwerfen, Aufbauen und Warten von Datenbanken sowie für die Programmierung von DB-Anwendungen

- **Voraussetzungen für Übernahme von Tätigkeiten:**
  - DB-Systementwicklung und -implementierung
  - Entwicklung und Einsatz von DB-gestützten Anwendungen
  - Integration von kooperativen Informations- und Anwendungssystemen
  - Installation, Administration und Tuning von DB- und Transaktionssystemen
  - Systemverantwortlicher für Datenbanksysteme, insbesondere Unternehmens-, Datenbank-, Anwendungs- und Datensicherungsadministrator, Informationswirt, ...

ÜBERSICHT

1. Architekturen von DB- und TA-Systemen

- statische Abbildungshierarchie eines DBS – Schichtenmodell
- Aufbau von zentralisierten Transaktionssystemen
- Verteilte Transaktionssysteme: Einsatz von Mehrrechner-DBS
- Client/Server-DBS

2. Externspeicherverwaltung

- Datei- und Segmentkonzepte
- Maßnahmen zur Fehlertoleranz
- Einbringverfahren

3. DB-Pufferverwaltung

- Nutzung von Lokalität
- Verwaltung des DB-Puffers (Suche, Speicherzuordnung)
- Ersetzungsverfahren und Vorausplanung

4. Speicherungsstrukturen

- Freispeicherverwaltung
- Adressierung von Sätzen
- Satzabbildung, Cluster-Bildung

5. Eindimensionale Zugriffspfade

- Mehrwegbäume, statische und dynamische Hashverfahren
- Zugriffspfade für Sekundärschlüssel
- Zugriffspfade für Setstrukturen
- Verallgemeinerte Zugriffspfadstrukturen

ÜBERSICHT (2)

6. Mehrdimensionale Zugriffspfade

- Unterstützung für raumbezogene Zugriffe
- Organisation der Datensätze
- Organisation des umgebenden Datenraums
- Grid-File, R-Baum u. a.
- Vergleich verschiedener Zugriffsverfahren

7. Satzorientierte Schnittstelle

- Data-Dictionary-Funktionen
- Scan-Konzepte,
- Sortier-Operator

8. Implementierung relationaler Operatoren

- Operatoren auf einer und auf mehreren Relationen
- Implementierung der Verbundoperation

9. Mengenorientierte Schnittstelle

- Formen der Spracheinbettung
- Übersetzung von DB-Anweisungen
- Anfrageoptimierung (Standardisierung, Vereinfachung, Restrukturierung und Transformation, Kostenmodelle)
- Code-Erzeugung
- Ausführung von DB-Anweisungen

LITERATURLISTE

Härder, T., Rahm, E.: Datenbanksysteme — Konzepte und Techniken der Implementierung, Springer-Verlag, 1999 (Es sind Hörerscheine erhältlich).

Gray, J., Reuter, A.: Transaction Processing—Concepts and Techniques, Morgan Kaufmann Publishers, Inc., San Mateo, CA., 1998 (5th printing).

Ramakrishnan, R.: Database Management Systems, McGraw-Hill, Boston, 1998.

O'Neil, P.: Database—Principles, Programming, Performance, Morgan Kaufmann Publishers, Inc., San Mateo, CA., 1994.

Saake, G., Heuer, A.: Datenbanken: Implementierungstechniken, MITP-Verlag, 1999.

Mitschang, B.: Anfrageverarbeitung in Datenbanksystemen — Entwurfs- und Implementierungskonzepte, Reihe Datenbanksysteme, Vieweg, 1995.

ZEITSCHRIFTEN:

ACM TODS Transactions on Database Systems, ACM-Publikation (vierteljährl.)

THE VLDB Journal VLDB Foundation (vierteljährlich)

Information Systems Pergamon Press (8-mal jährlich)

ACM Computing Surveys ACM-Publikation (vierteljährlich)

TAGUNGSBÄNDE:

ICDE Proceedings, „International Conference on Data Engineering", jährliche Konferenz der IEEE

SIGMOD Proceedings, jährliche Konferenz der ACM Special Interest Group on Management of Data

VLDB Proceedings, jährliche Konferenz „Very Large Data Bases" (ACM, IEEE, ...)

BTW Tagungsbände der alle 2 Jahre stattfindenden Tagungen „Datenbanksysteme in Büro, Technik und Wissenschaft" der Gesellschaft für Informatik (GI),
sowie weitere Tagungen innerhalb des GI-FA 2.5 Informationssysteme

Datenbanktechnologie - Was ist das?

Konzepte, Methoden, Werkzeuge und Systeme für die

- dauerhafte — Lebensdauer Daten > Dauer Erzeugungsprozeß
- zuverlässige — Integrität, Konsistenz, Verlustsicherheit
- unabhängige — wechselseite Änderungsimmunität AWP-DB

Verwaltung und

- komfortable — „höhere“ abstrakte Schnittstelle
- flexible — Ad-hoc-Zugriffsmöglichkeit

Benutzung von

- großen — Größe Daten >> Größe Hauptspeicher
- integrierten — kontrollierte Redundanz von / für mehrere Anwendungen,
- mehrfachbenutzbaren — paralleler Zugriff

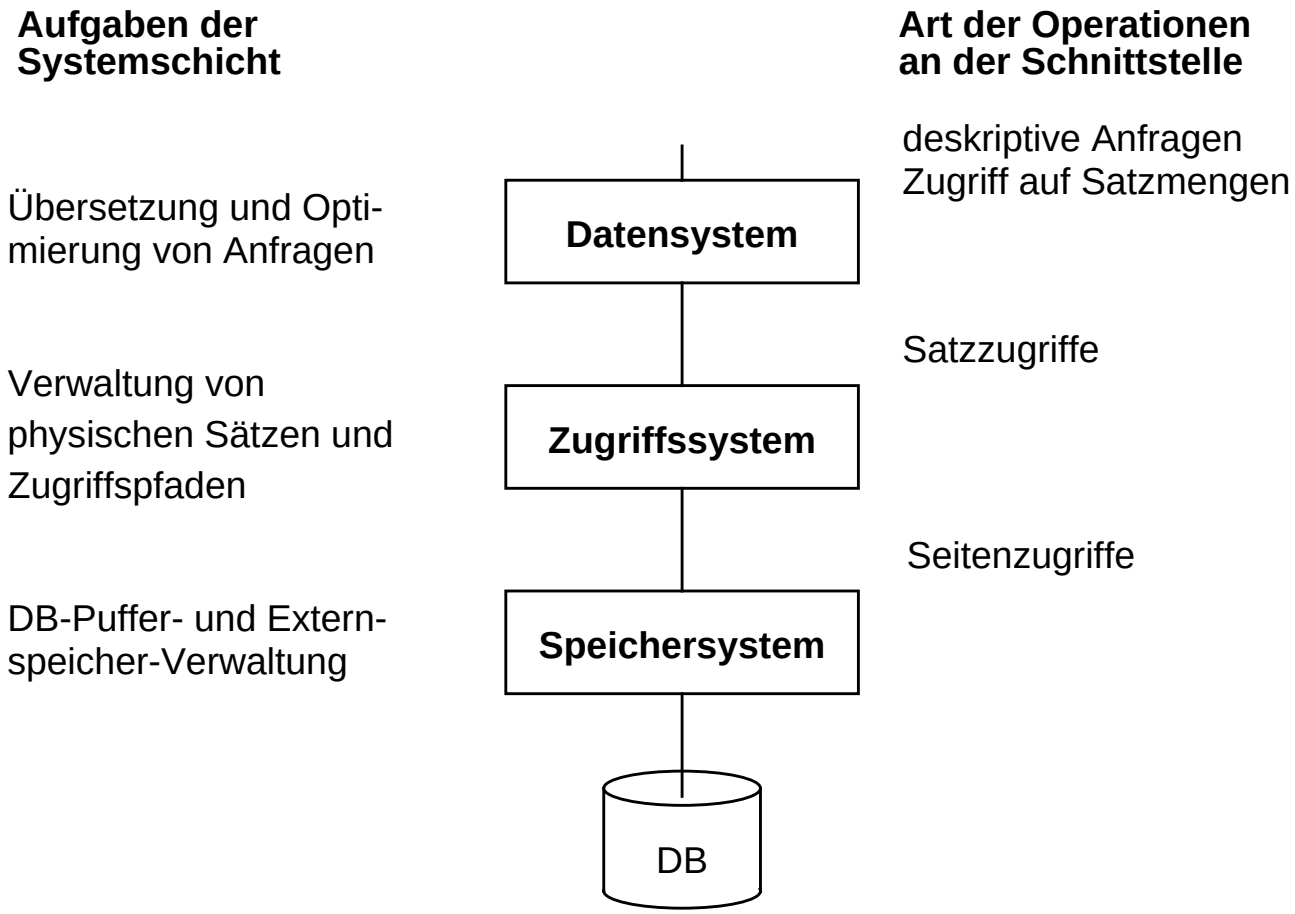
Datenbasen

1. Architekturen von DB- und TA-Systemen

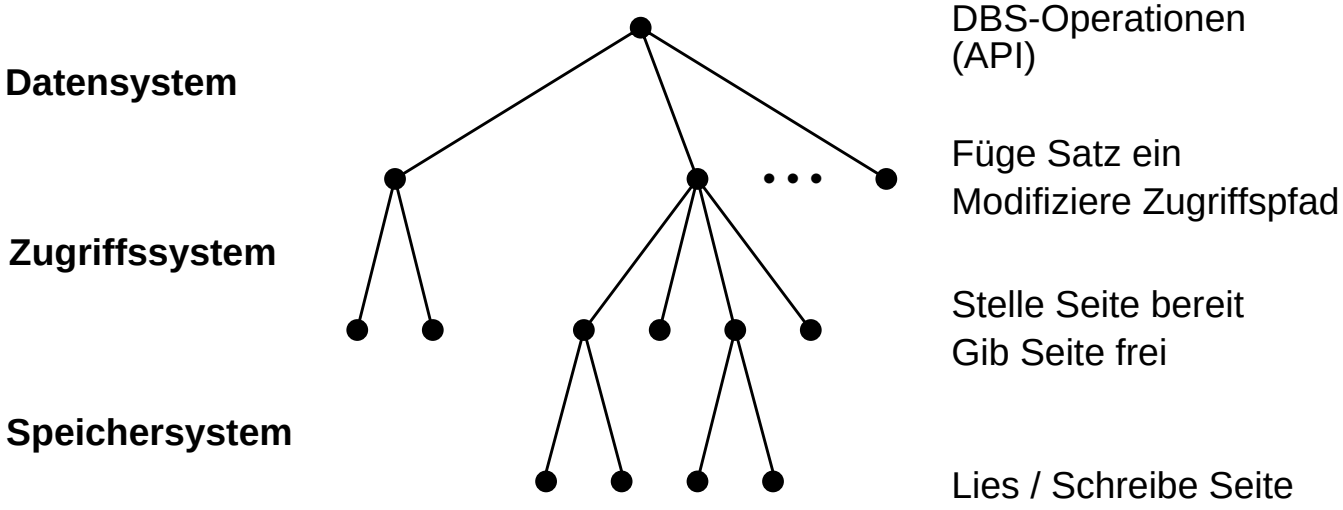
- **Schichtenmodell eines DBS**
  - schrittweise Abstraktion von einem Bitstring auf der Magnetplatte zu logischen Sichten und SQL-Operationen oder komplexen Objekten
  - Prinzipien der Schichtenbildung
  - effiziente (dynamische) Abbildung über mehrere Schichten hinweg?
- **Transaktionssysteme**
  - Grundlegende Abstraktionen und Grobaufbau
  - Schichtenmodell
  - Anforderungen
- **Verteilte Transaktionssysteme**
  - Aspekte/Prinzipien der Verteilung
  - Verteilung unter Kontrolle des TP-Monitors
  - Verteilung unter Kontrolle des DBS (Mehrrechner-DBS)
- **Client/Server-DBS**
  - „Verschicken von Anfragen” vs „Verschicken von Daten”
  - Client/Server-Architekturen für objektorientierte DBS

Schichtenmodell eines DBS – Überblick

- **Vereinfachtes Schichtenmodell**



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



Schichtenbildung im Schichtenmodell

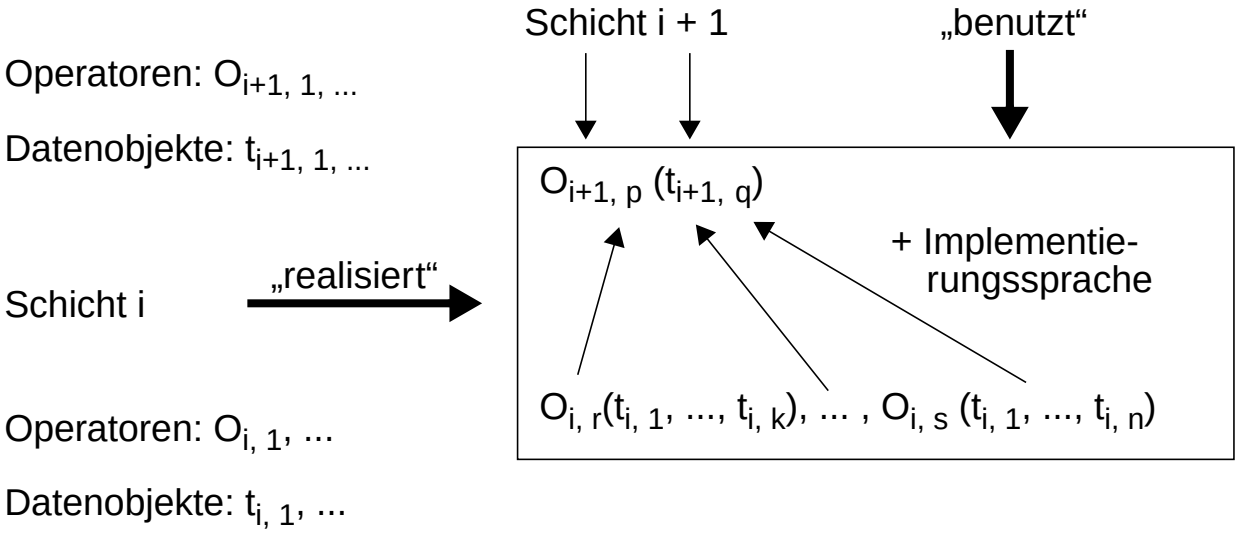
- **Ziel: Architektur eines datenunabhängigen DBS**

- **Wieviele Schichten braucht man?**

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

- **Empfohlene Konzepte:**
  - Geheimnisprinzip (*Information Hiding*)
  - Hierarchische Strukturierung

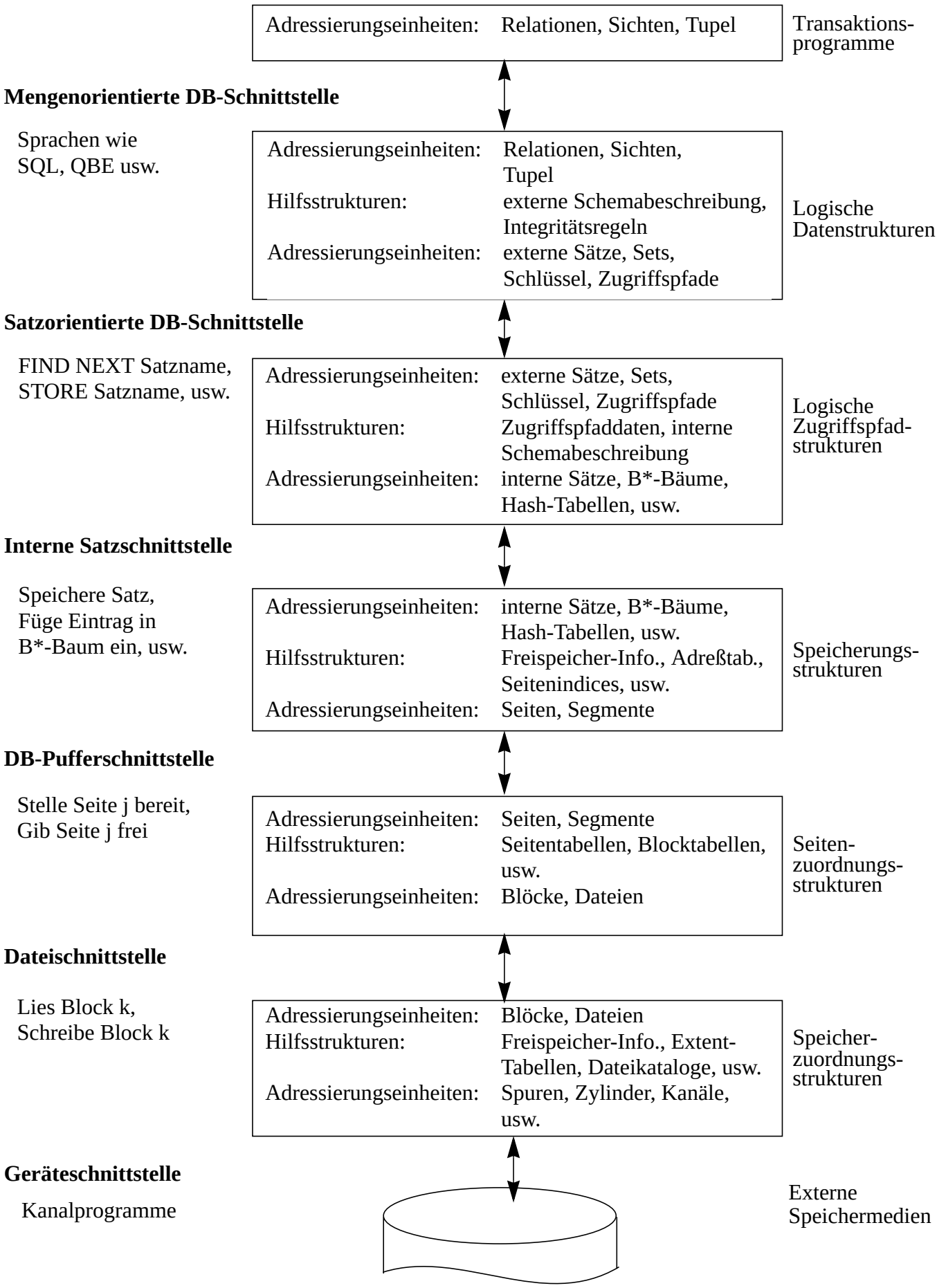
- **Aufbauprinzip:**



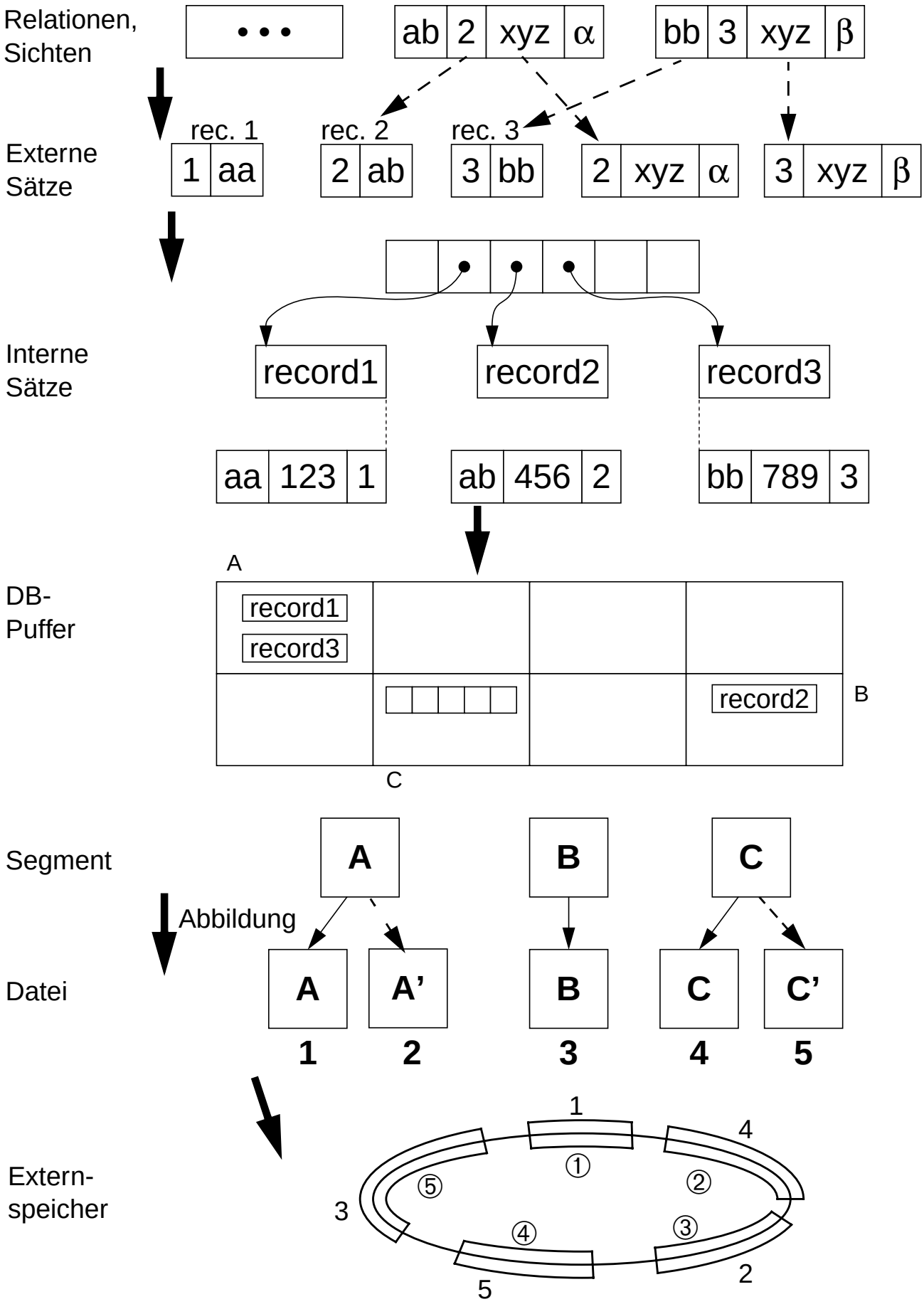
- **„benutzt“-Relation:**

  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

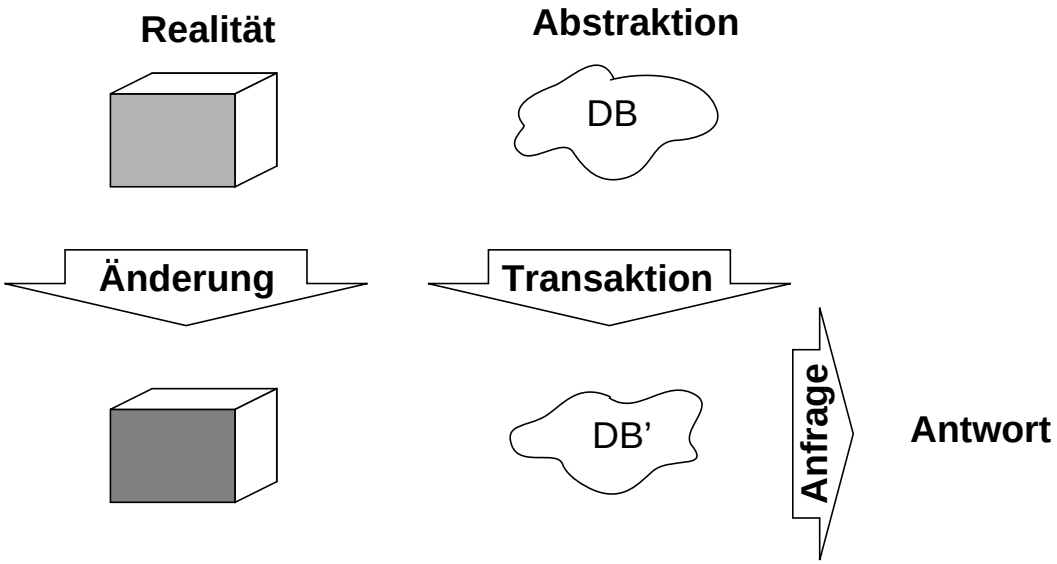


Schichtenweise Abbildungen in einem DBS



Transaktionssysteme – Grundlegende Abstraktionen

- **Abstraktion**
  - Der reale Zustand (einer Miniwelt) wird durch eine Abstraktion – DB genannt – dargestellt
  - Eine Veränderung des realen Zustands wird durch ein Programm – Transaktion genannt – in der DB nachvollzogen



- **Definitionen einer TA aus verschiedenen Sichten**
  - Mit einer Transaktion (TA) wird ein Vorgang einer Anwendung in einem Rechensystem abgewickelt.
    Ein solcher Vorgang bildet typischerweise einen **nicht-trivialen Arbeitsschritt (*unit of work*)** in betrieblichen Abläufen.
  - Eine (On-line) Transaktion ist die Ausführung eines Programmes, das mit Hilfe von Zugriffen **auf eine gemeinsam genutzte Datenbank** (DB) eine Anwendungsfunktion erfüllt.
  - Eine DB-Transaktion ist eine **ununterbrechbare Folge von DB-Operationen**, welche die Datenbank von einem logisch konsistenten in einen logisch konsistenten Zustand überführt.

ACID-Eigenschaften

Eine Transaktion ist eine **Sammlung von Aktionen** mit folgenden Eigenschaften:

- ***Atomicity***

  Die Änderungen einer TA, die den Zustand der Miniwelt betreffen, sind atomar; es gilt „Alles oder Nichts“. Zu diesen Änderungen gehören **DB-Aktualisierungen, Nachrichten und Operationen auf Steuerungsgeräten.**

- ***Consistency***

  Eine TA ist eine korrekte Transformation des Miniwelt-Zustandes. Die Operationsfolge verletzt keine der Integritätsbedingungen, die mit dem Zustand verknüpft sind. Deshalb muß eine TA ein korrektes Programm sein.

- ***Isolation***

  Trotz der konkurrenten Abwicklung von TA erscheint jeder TA T, daß andere TA entweder vor T oder nach T ablaufen, aber nicht beides zusammen.

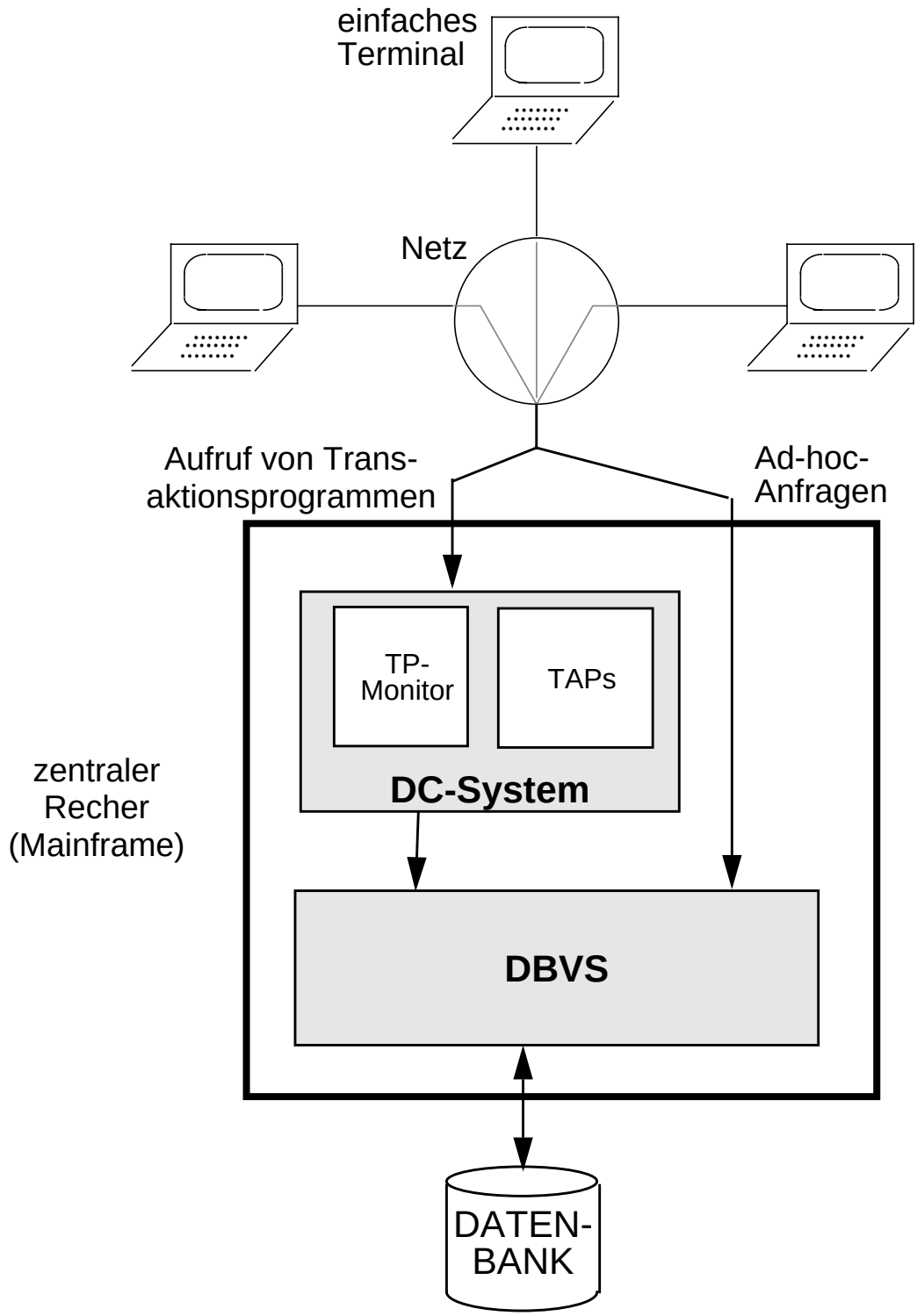
- ***Durability***

  Sobald eine TA erfolgreich beendet wird (*commit* ausführt), überleben ihre Zustandsänderungen alle erwarteten Fehler

➥ Was garantieren diese Eigenschaften beim Ablauf einer Kontenbuchungs-TA?

- **Struktur eines TA-Programms**
  - BEGIN_WORK() → alle nachfolgenden Operationen gehören zur TA
  - COMMIT_WORK() → neuer konsistenter Zustand wird persistent (*Durability*)
  - ROLLBACK_WORK() → Fehler erzwingt rückwirkungsfreies Rücksetzen der TA (*Atomicity*)

Grobaufbau eines zentralisierten Transaktionssystems



TP-System = Transaktionssystem (*Transaction Processing System*)

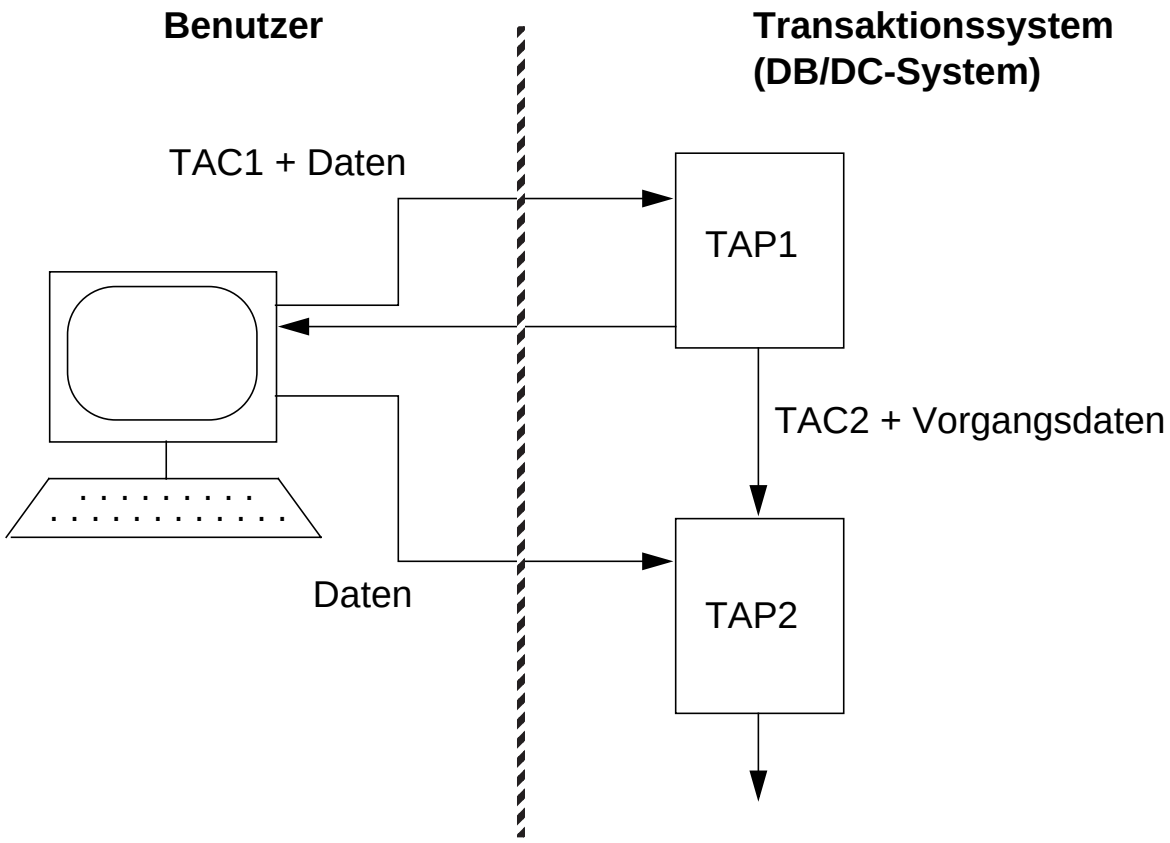
TAP = TA-Programm/Anwendungsprogramm

TP-Monitor = *Transaction Processing Monitor*

DC-System = Datenkommunikationssystem

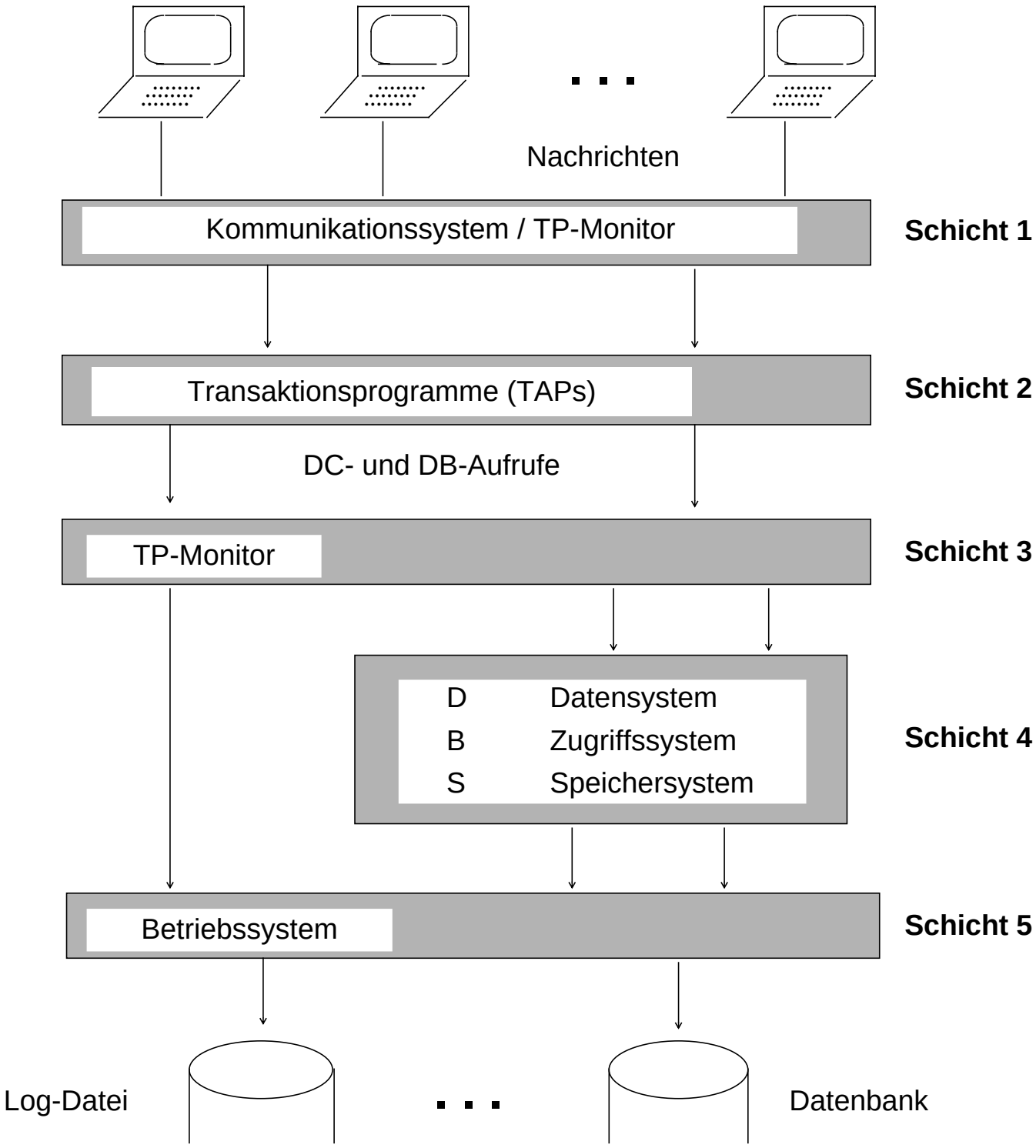
Transaktionssysteme – operationale Eigenschaften

- **Prinzipieller Ablauf**



- **System- und Betriebsmerkmale**
  - Benutzung im Dialog: „parametrischer“ Benutzer
  - wenige kurze Transaktionstypen: hohe Wiederholrate
  - sehr viele Benutzer gleichzeitig
  - gemeinsame Datenbestände mit größtmöglicher Aktualität
  - kurze Antwortzeiten als Optimierungsziel vor Auslastung
  - stochastisches Verkehrsaufkommen
  - hohe Verfügbarkeit des Systems

Schichtenmodell eines Transaktionssystems



Anforderungen an Transaktionssysteme

- **Abwicklung sehr hoher TA-Raten**
  - TA-Typ „Kontenbuchung“: Maßeinheit bei den Benchmarks TPC-A, TPC-B
    ➥ n Ktps vom Typ TPC-B?
  - komplexere Transaktionen „Abwicklung von Bestellungen“ beim TPC-C
    ➥ n*100 Ktpm vom Typ TPC-C
  - Es ist immer mehr Funktionalität gefragt!
    (benutzerdefinierte Datentypen, große Objekte, Multimedia, ...)

- **Gewährleistung hoher Verfügbarkeit**
  - Vermeidung eines Systemausfalls
  - Verfügbarkeit aller Daten
  - **Mehrrechner-Architekturen** erforderlich

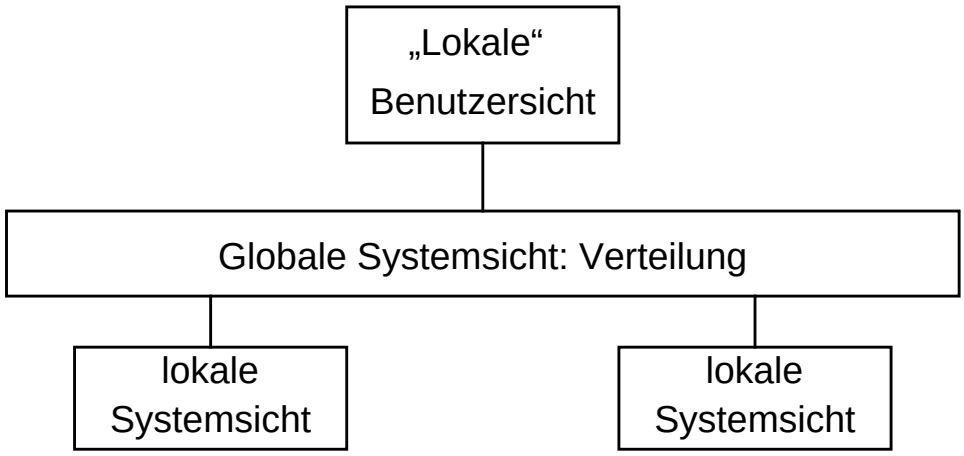
- **Modulares Systemwachstum**
  - Subsysteme als Einheiten des Entwurfs, der Kontrolle und des Ausfalls
  - annähernd lineare Durchsatzerhöhung
  - Zuordnung von Prozessoren und **Daten** (Platten)

- **Einbindung in Client/Server-Architekturen**
  - 2-stufig: Präsentation - Anwendung/Datenhaltung
  - 3-stufig: Präsentation - Anwendung - Datenhaltung (Skalierbarkeit!)
  - 4-stufig: Einbezug von Web-Servern

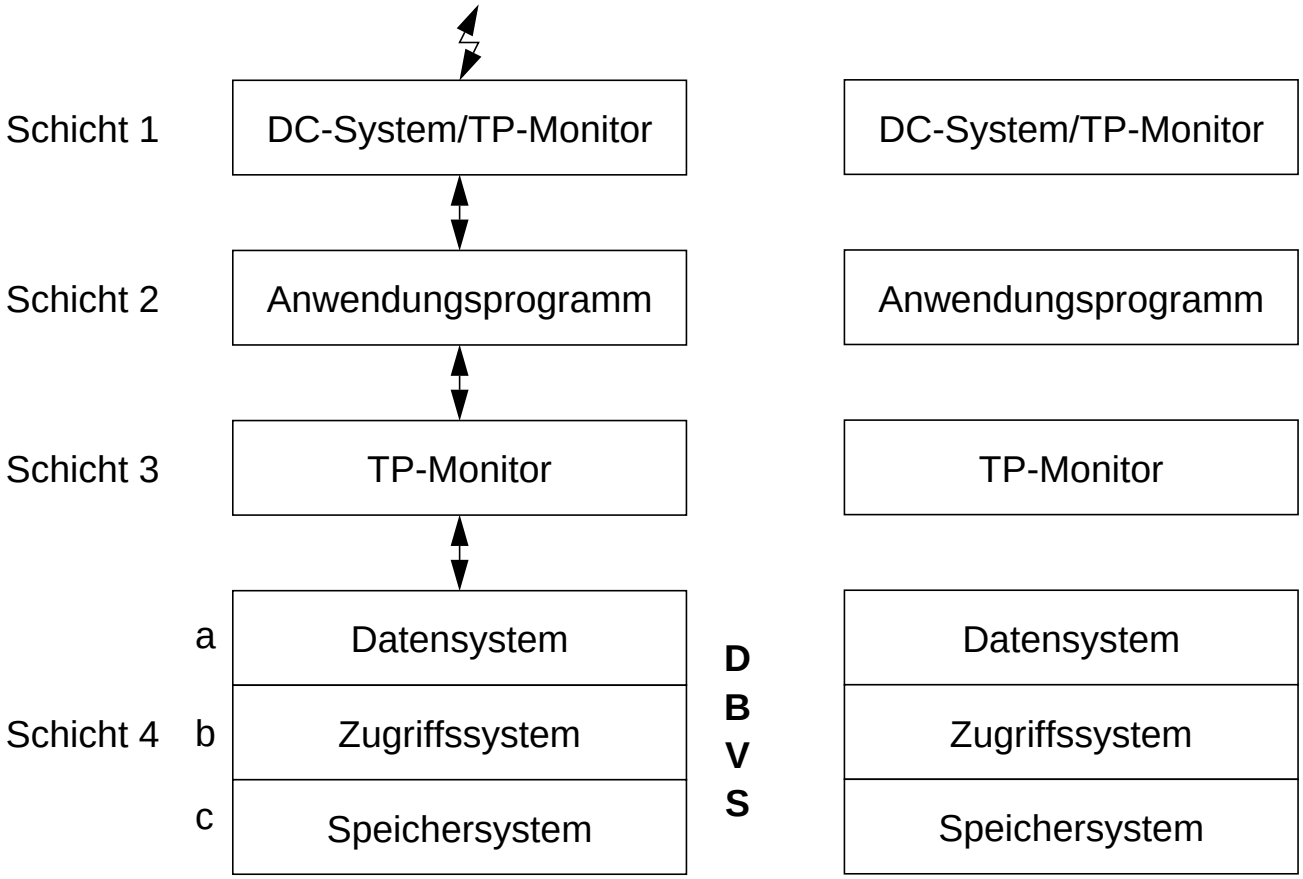
➥**Transaktionssysteme sind** i. allg. horizontal und/oder vertikal **verteilt**!

Klassifikation verteilter Transaktionssysteme

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



- **Homogenes Systemmodell**



Klassifikation verteilter Transaktionssysteme (2)

- **Realisierung der globalen Systemsicht: wo?**
  - Forderung: lokale Sicht des Benutzers (Ortstransparenz, „single system image“)
  - Abbildung auf lokale Sichten der Knoten

- **Verteilung unter Kontrolle des TP-Monitors *(Verteilte DC-Systeme)***
  - TP-Monitor verbirgt weitgehend Heterogenität bezüglich Kommunikations-protokollen, Netzwerken, BS und Hardware
  - DBS bleiben weitgehend unabhängig (keine Kooperation zwischen DBS)
  - heterogene DBS möglich

    ➥ Einsatz von DBS-Gateways
  - geringe Implementierungskomplexität

- Drei wesentliche Alternativen (werden hier nicht vertieft):

  1. ***Transaction Routing***
     - globale Sicht in Schicht 1 (Kommunikationssystem)
     - Einheit der Verteilung ist die Transaktion (TA-Typ)

  2. **Programmierte Verteilung**
     - globale Sicht in Schicht 2 (im AP)
     - Einheit der Verteilung ist eine Teil-Transaktion (Programmfragment)

  3. **Verteilung von DB-Operationen (*Function Request Shipping*)**
     - globale Sicht in Schicht 3 (TP-Monitor)
     - Einheit der Verteilung ist ein DML-Befehl

Klassifikation verteilter Transaktionssysteme (3)

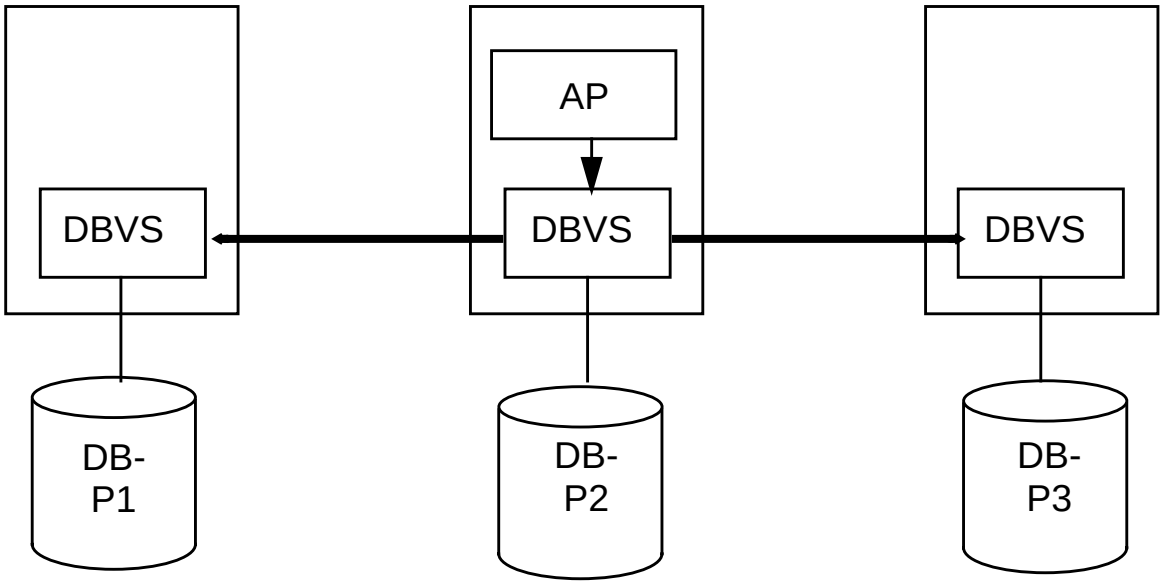
- **Verteilung unter Kontrolle des DBS ($\rightarrow$ Mehrrechner-DBS)**
  - vollständige Verteilungstransparenz für AP
  - setzt eine logische Datenbank voraus
  - geringe Eignung für Knotenautonomie und Heterogenität

- **mehrere (homogene) Realisierungsalternativen**
  - globale Sicht in Schicht 4
  - Verteilung ganzer DML-Befehle bzw. von Teiloperationen (Schicht 4a)
  - Verteilung von internen Satzoperationen (Schicht 4b)
  - Verteilung von Seitenzugriffen (Schicht 4c)

- **gleichgestellte Verarbeitungsrechner vs. Spezialisierung (Client/Server-Konfigurationen)**

- **Realisierung allgemeiner Resource-Manager-Architekturen**
  - Resource Manager (RMs) sind Systemkomponenten, die **Transaktionsschutz für ihre gemeinsam nutzbaren Betriebsmittel** (BM) bieten
  - Sie gestatten die externe Koordination von BM-Aktualisierungen durch ein 2PC-Protokoll
  - Beispiele: DBS, Dateisysteme, PS-Laufzeitsystem, Mail-Service, Window-Manager, ...

Mehrrechner-DBS



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

- **Architekturklassen**

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

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

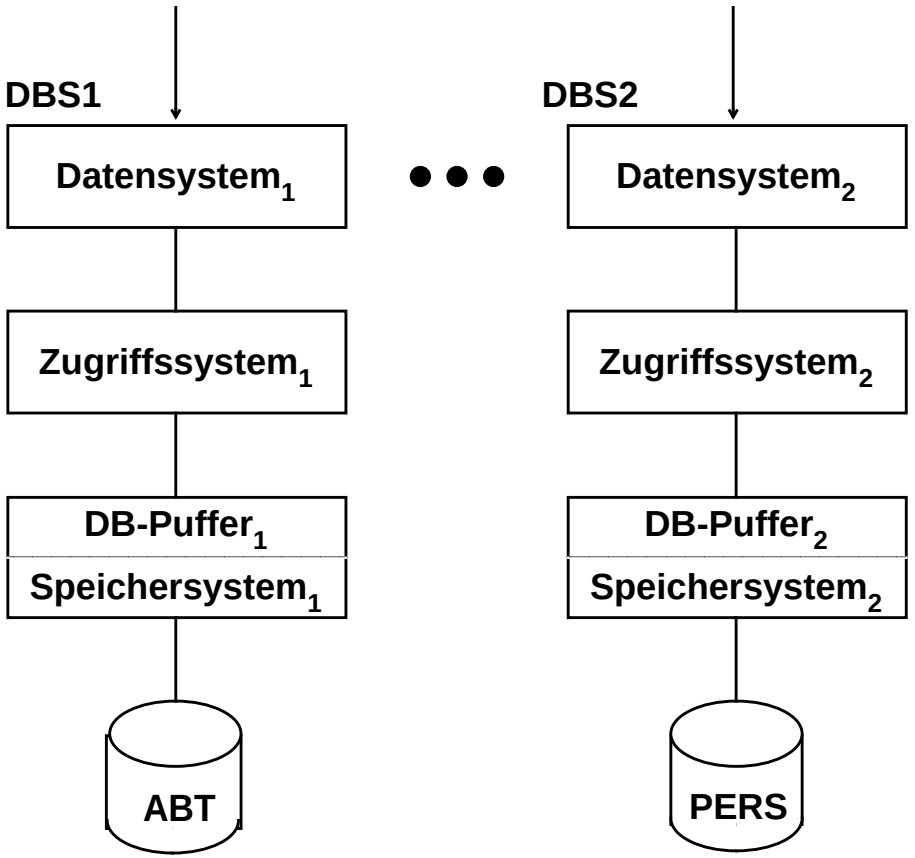
  2. DB-Sharing (*Shared Disk*)

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

Transaktionsverarbeitung [1] in 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 bisherigen 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.

  ➥ keine Nutzung vorhandener Referenzlokalität im Client

- **Weiteres Verarbeitungskonzept**

  Folgende Vorgehensweise wird von den meisten 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

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

---

1. Härder, T., Meyer-Wegener, K.: Transaktionssysteme in Workstation/Server-Umgebungen, Informatik - Forschung und Entwicklung (1990), 5: 127-143.

Transaktionsverarbeitung[1] in Client/Server-DBS (2)

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

- **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

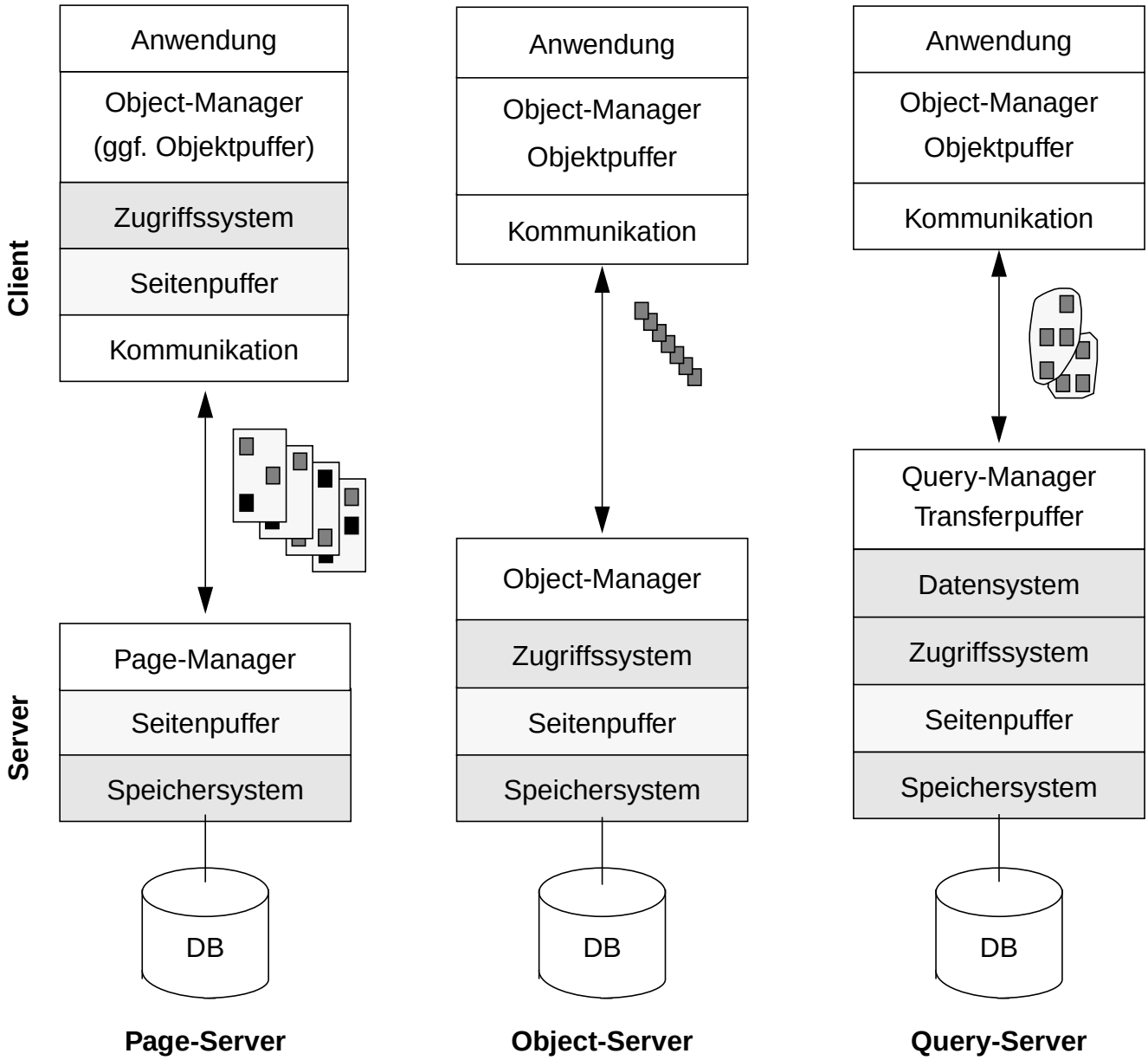
  ➥ DBS, die Entwurfsumgebungen unterstützen sollen, benötigen auf Client- und Server-Seite geeignete Funktionalität zur Datenverwaltung/Transaktionsunterstützung.

---

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.

Schichtenmodelle für Client/Server-DBS

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



➥ Im Objektpuffer erfolgt typischerweise navigierende Verarbeitung!

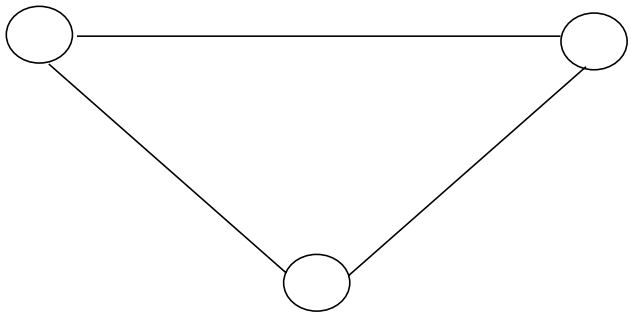
Zusammenfassung

- **Allgemeine Aspekte des Schichtenmodells**
  - Schichtenmodell ist allgemeines Erklärungsmodell für die DBS-Realisierung
  - Schichtenbildung läßt sich zweckorientiert verfeinern/vergröbern: Anwendbarkeit für TA-Systeme, 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
- **Klassifikation Verteilter Transaktionssysteme**
  - Transaction Routing, Programmierte Verteilung, Aufruf von DB-Operationen
  - Mehrrechner-DBS als Shared-Nothing oder Shared-Disk-Systeme
- **Client/Server-Systeme**
  - Data Shipping ist vor allem bei hoher Referenzlokalität vorteilhaft. Es wird durch das Checkout/Checkin-Konzept genutzt.
  - 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

Zusammenfassung (2)

Verteilte Transaktionssysteme

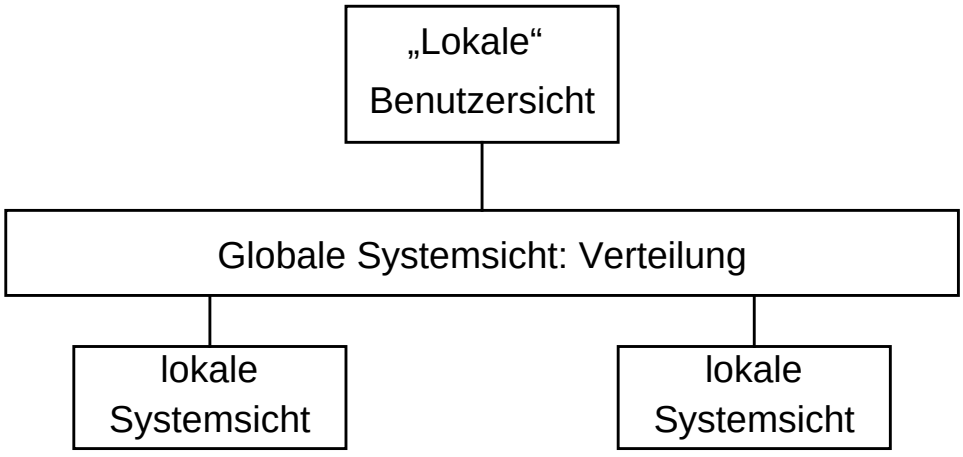
- **wirtschaftliche Gründe**
  - reduzierte HW-Kosten
  - verfügbare Kommunikationseinrichtungen
- **organisatorische Gründe**
  - lokale Unterstützung organisatorischer Strukturen
  - Integration existierender Datenbanken
  - lokale Autonomie
- **technische Gründe**
  - Erhöhung der Leistung: Lokalität der Verarbeitung, parallele Verarbeitung
  - größere Verfügbarkeit und Zuverlässigkeit: Minimierung der Auswirkung von „entfernten“ Fehlern, Fehlermaskierung durch Redundanz
  - modulare Wachstumsfähigkeit
- **Achtung!**
  Rechner können repliziert werden, Daten müssen **gemeinsam** benutzt werden



- **aber:** Verteilte Datenbanksysteme oder DB/DC-Systeme (selbst relationale Systeme) sind kein Allheilmittel!

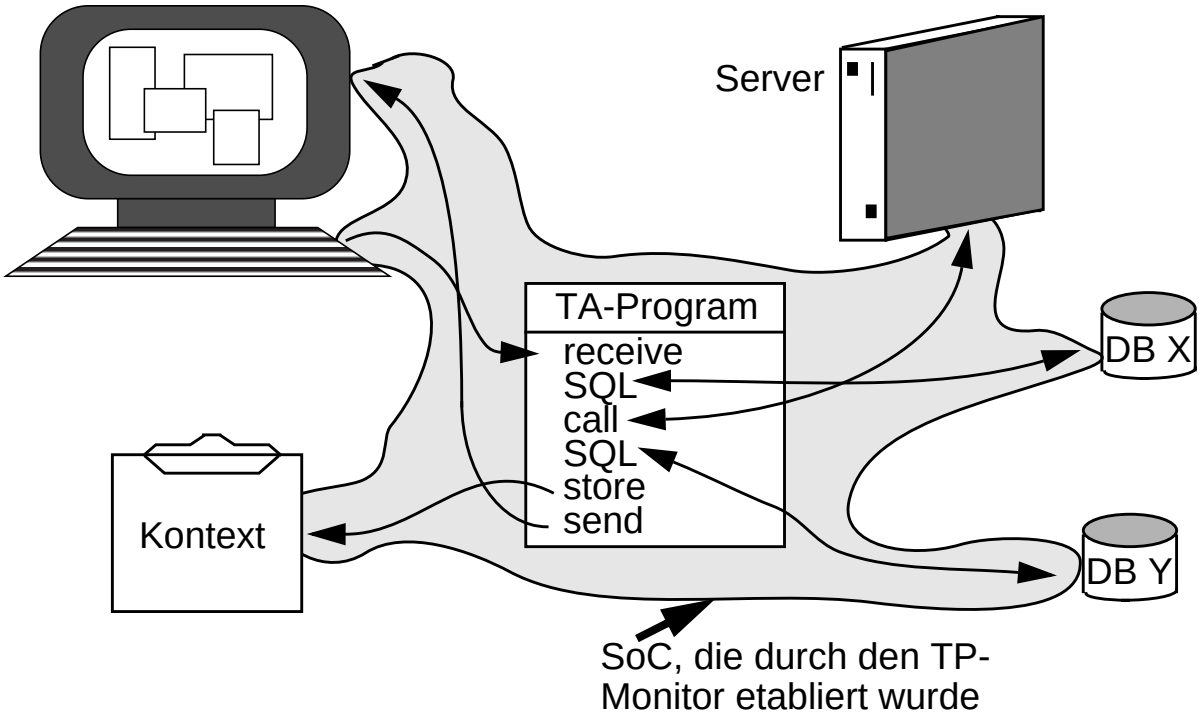
Verteilte Transaktionssysteme (2)

- **Prinzipien: Funktions-, Daten- und Lastverteilung**
- **Funktionszuordnung**
  - homogener Ansatz: Replikation der Funktionen
  - heterogener Ansatz: Partitionierung von Funktionen
    z. B. Client/Server-Architekturen
- **Datenzuordnung**
  - Partitionierung
  - Replikation
  - Sharing (erfordert lokale Rechneranordnung)
- **Lastzuordnung**
  - Verteileinheiten durch Funktions- und Datenallokation mitbestimmt:
    Transaktionsaufträge, Teilprogramme, DB-Operationen (DML-Befehle)
- **Vorgehensweise bei der Verteilung**
  - Partitionierung der Daten (ggf. mit Replikation)
  - Kommunikation zwischen Prozessen, welche die Zugriffe auf jeweils lokalen Daten ausführen
- **gewünschte Sichtbarkeit**



Verteilte Transaktionssysteme: heterogene Komponenten

- **Integrierte Kontrolle durch Transaktionsdienste** – dargestellt in einer typischen Umgebung zur Transaktionsverarbeitung



➥ Die gesamte verteilte Verarbeitung in einer SoC ist eine ACID-Transaktion

- alle Komponenten werden durch die TA-Dienste integriert
- für die Kooperation ist eine Grundmenge von Protokollen erforderlich

- **Transaktionsschutz durch kooperierende Resource Manager**

  ➥ Gewährleistung der ACID-Eigenschaften für DB-Daten und auch für andere Betriebsmittel (persistente Warteschlangen, Nachrichten, Objekte von persistenten Programmiersprachen)

- ***Resource Manager*** **(RMs)**
  sind Systemkomponenten, die **Transaktionsschutz für ihre gemeinsam nutzbaren Betriebsmittel** (BM) bieten, d. h., sie gestatten die externe Koordination von BM-Aktualisierungen durch ein 2PC-Protokoll. Beispiele: DBS, Dateisysteme, PS-Laufzeitsystem, Mail-Service, Window-Manager, ...

  ➥ globaler Transaktionsschutz durch **Resource-Mgr-Architektur** unter Kontrolle eines TP-Monitors

Transaktionsverarbeitung [1] in Entwurfsumgebungen

- **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 zum Client/Server-Ansatz 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.

  ➥ keine Nutzung vorhandener Referenzlokalität im Client

- **Weiteres Verarbeitungskonzept**

  Folgende Vorgehensweise wird von den meisten 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

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

---

1. Härder, T., Meyer-Wegener, K.: Transaktionssysteme in Workstation/Server-Umgebungen, Informatik - Forschung und Entwicklung (1990), 5: 127-143.

Transaktionsverarbeitung in Entwurfsumgebungen (2)

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

- **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

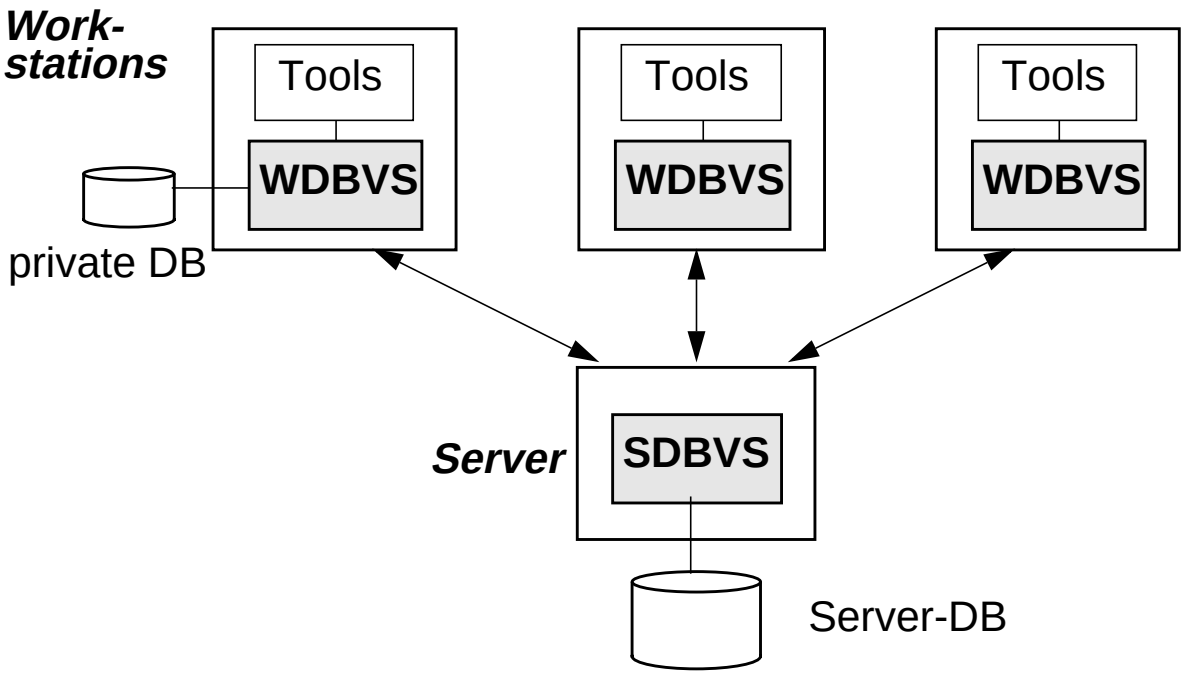
  ➥ DBS, die Entwurfsumgebungen untersützen sollen, benötigen auf Client- und Server-Seite geeignete Funktionalität zur Datenverwaltung/Transaktionsunterstützung. Neben dem Begriff Client/Server-Architektur wird auch der Begriff „**Workstation/Server-Architektur**“ benutzt; er soll explizit machen, daß auf Client-Seite Workstation-Eigenschaften benötigt werden.

TA-Verarbeitung in Entwurfsumgebungen (3)

- **Neue Anforderungen**
  - Entwurfsaufgaben, 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

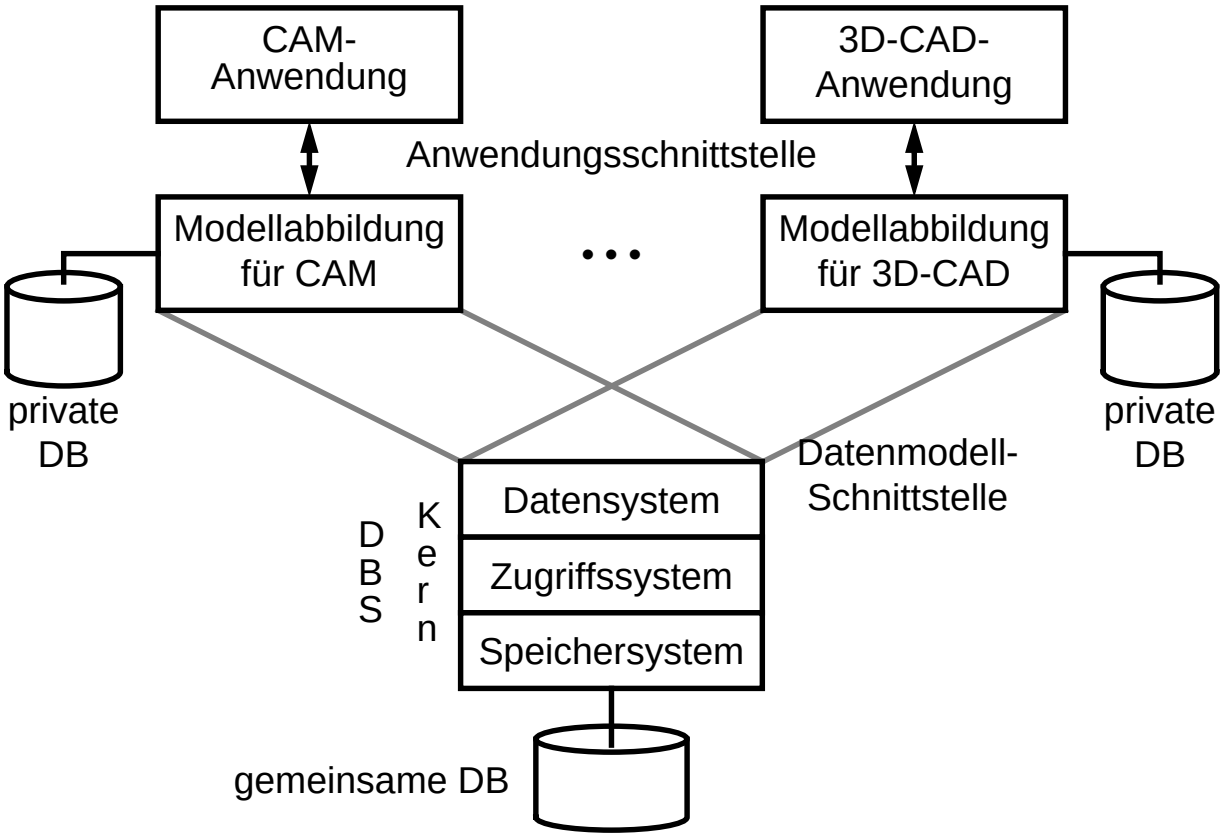
- **ClientServer-Architektur: Prinzip**



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

TA-Verarbeitung in Entwurfsumgebungen (4)

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



- **Trennung**
  - anwendungsunabhängige Funktionen im Server
  - anwendungsbezogene Verarbeitung 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

TA-Verarbeitung in Entwurfsumgebungen (5)

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

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

- **Leistungsanforderungen**
  - navigierende Tool-Operationen: 10^5 Ref./sec bei CAD
  - Lokalität in der Nähe der Anwendung
    (alle Objekte mit Rereferenz im Objektpuffer)
  - direkte Repräsentation von Beziehungen
    (bei 3 Referenzen: Pointer Swizzling)

- **Weitere Merkmale**
  - Lange Dauer von Entwurfsvorgängen (Wochen/Monate)

➥ in der Regel keine konkurrierenden Zugriffe auf Entwurfsdaten

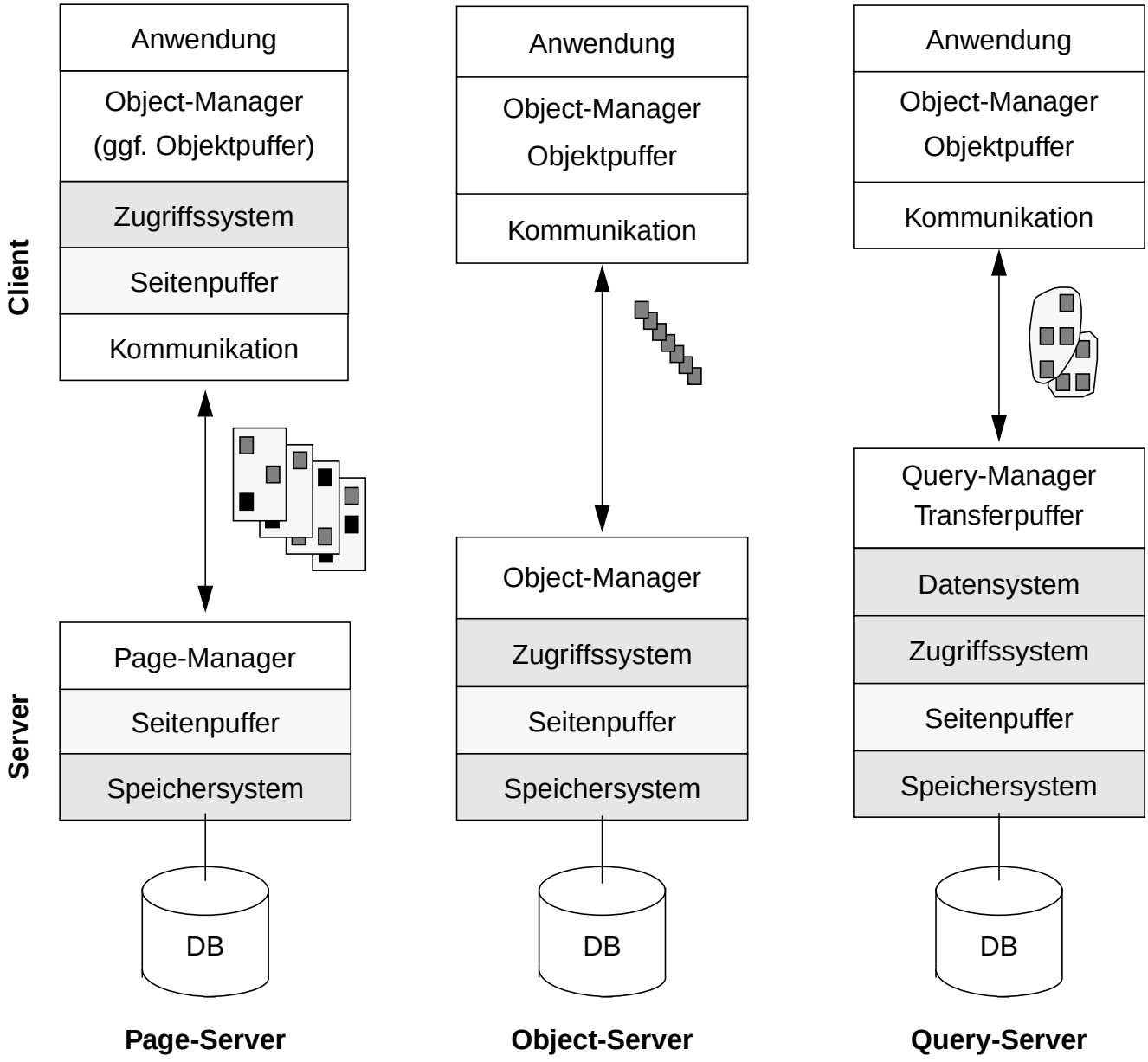
  - Unterstützung von Versionen
  - Kontrollierte Kooperation zwischen mehreren Entwerfern

TA-Verarbeitung in Entwurfsumgebungen (6)

- **Weitere Ansätze:**

  Client/Server-Architekturen für objektorientierte DBS

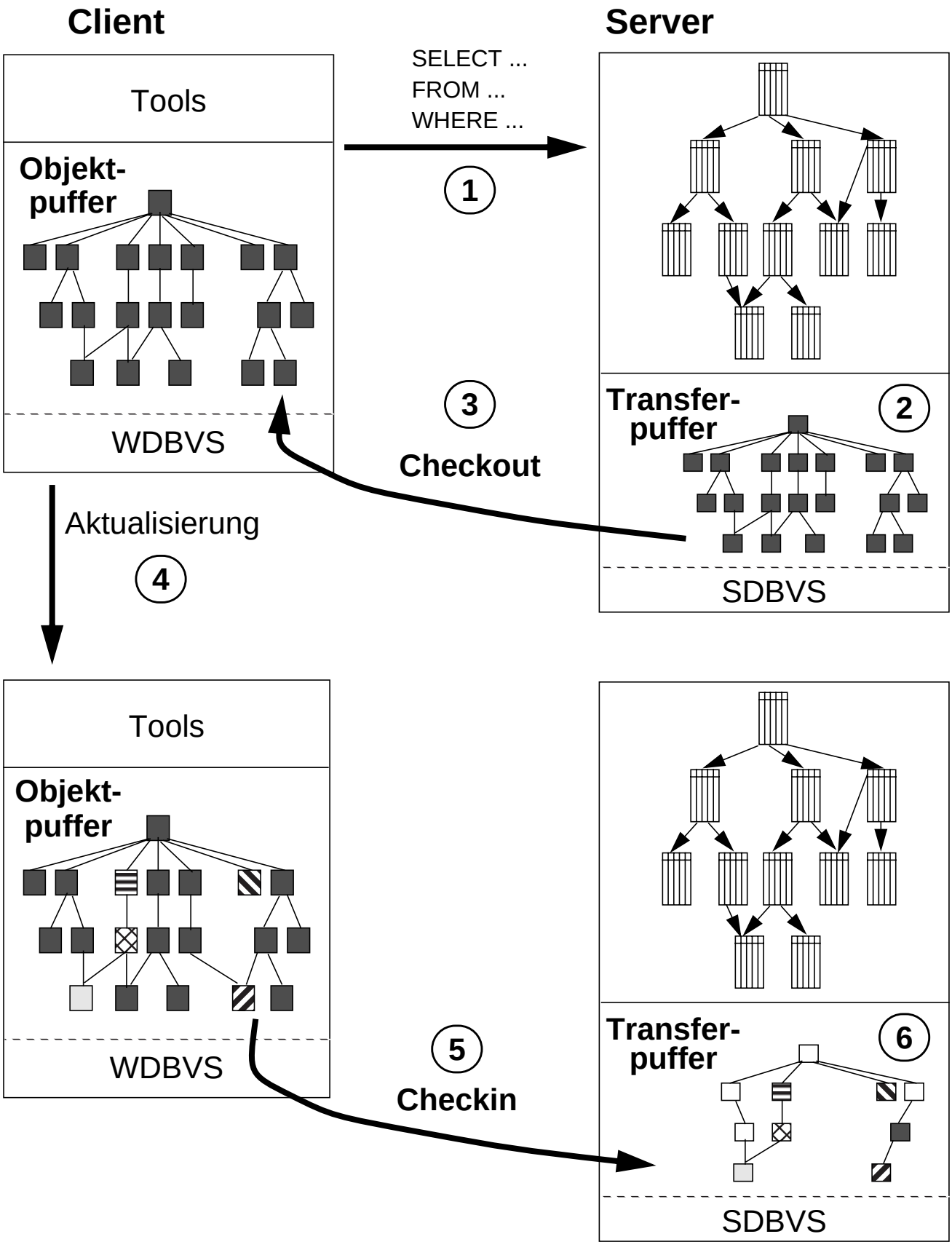
  - File-Server
  - Page-Server
  - Object-Server
  - Query-Server



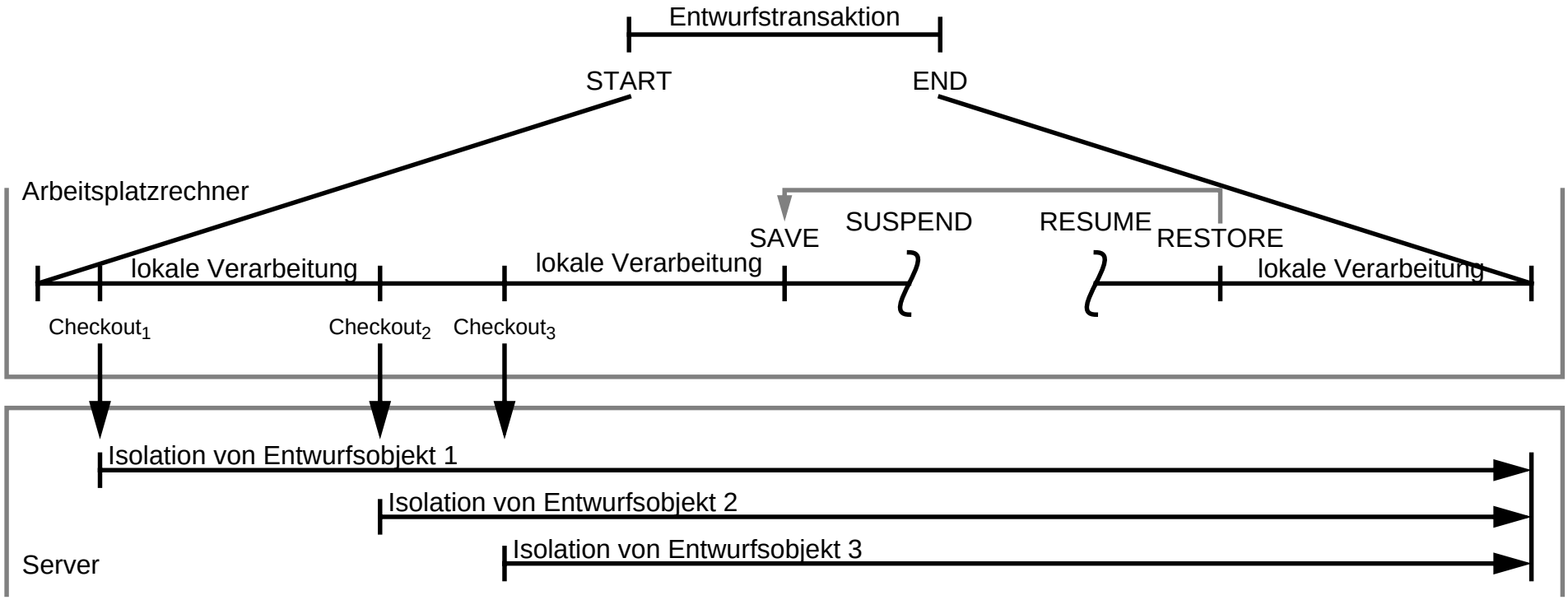
Verarbeitungsprinzip:
Laden, Verarbeiten, Integrieren

- **Datenanforderung für eine Entwurfstransaktion**
  - eine oder mehrere Checkout-Anforderungen
  - Entwurfsobjekte werden in den Objektpuffer eingelagert
- **Datenmanipulation**
  - komplex-strukturierte Objekte: Einsatz hierarchischer Cursor
  - gesamte Objektverarbeitung ist lokal: Akkumulation von Änderungen
  - Anwendungsmodell: Menge von ADT’s
- **Verarbeitungskontrolle und Recoverymaßnahmen**
  - Client-DBS schreibt Recovery Points
  - lokale Rücksetzungsmaßnahmen durch Savepoints (SAVE, RESTORE)
  - Unterbrechung der Verarbeitung (SUSPEND, RESUME)
- **Datenpropagierung**
  - ein Checkin am Ende der Entwurfstransaktion
  - Überprüfung von Integritätsbedingungen in der Server-DB

Verarbeitungsprinzip: Laden, Verarbeiten, Integrieren (2)



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)

TA-Verarbeitung in offenen Systemen

- **Die Idee der Client/Server-Verarbeitung korrespondiert mit dem Konzept von Offenen Systemen.**

  Definition von IEEE im Rahmen der POSIX-Aktivität:

  Ein offenes System ist

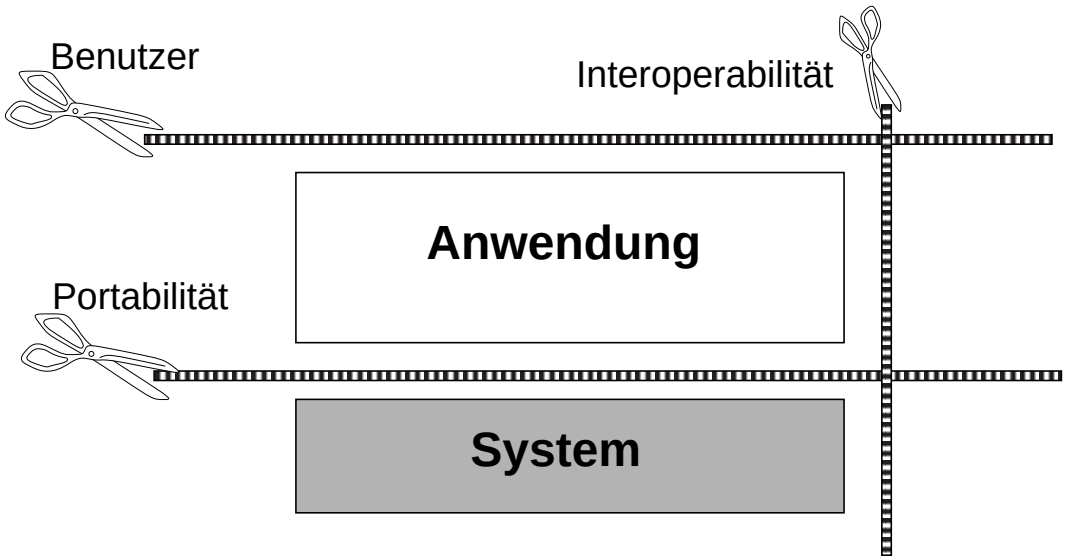
  „ein System, das in ausreichendem Maße offengelegte Spezifikationen für Schnittstellen und dazugehörige Formate implementiert, damit entsprechend gestaltete Anwendungssoftware

  - auf eine Vielzahl verschiedener Systeme portiert werden kann (mit Anpassungen),
  - mit anderen Anwendungen lokal und entfernt interoperabel ist,
  - mit Benutzern in einer Art interagiert, die das Wechseln der Benutzer zwischen Systemen erleichtert“.

  ➥ Diese Anforderungen spiegeln sich z. T. auch in den Zielvorstellungen von Client/Server-Systemen wider.

- **Offenheit impliziert Standardisierung**

  - Definition und Veröffentlichung von Schnittstellen zwischen Komponenten



  - **Systemschnittstellen** gewährleisten die Portabilität der AW-Software
  - **Aufrufschnittstellen** erlauben die Interoperabilität zwischen Anwendungen und zwischen Systemen
  - **Benutzerschnittstellen** regeln einen einheitlichen Benutzerzugang

TA-Verarbeitung in offenen Systemen (2)

- **Standards zur TA-Verwaltung**

  TP-Monitore benötigen Standards, weil sie letztendlich den „Klebstoff“ verkörpern, der die unterschiedlichen und heterogenen SW-Komponenten zusammenfügt.

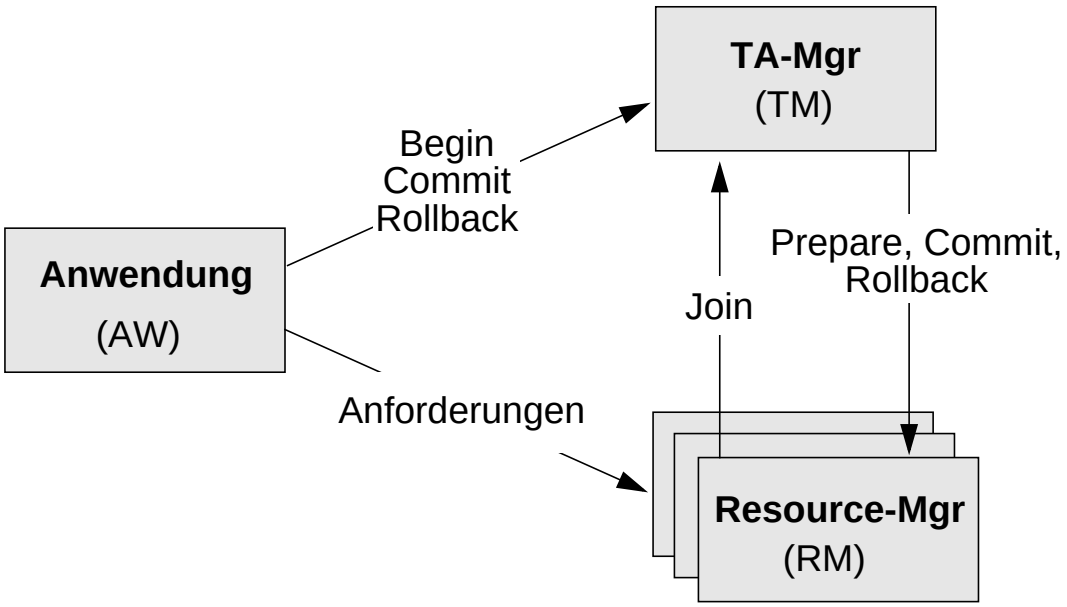
  - Ihre AW laufen auf verschiedenartigen Plattformen und
  - haben Zugriff zu verschiedenen DBs und RMs
  - Die AW haben absolut kein Wissen voneinander

  ➥ Einziger Weg zur Verknüpfung: „**Offene Standards**“
  Bereitstellung von Schnittstellen zwischen TP-Monitor und RMs, TP-Monitor untereinander und TP-Monitor und AW

- **Standards kommen aus zwei Quellen**

  - **ISO:**
    OSI-TP spezifiziert die Nachrichtenprotokolle zur Kooperation von TP-Monitoren
  - **X/OPEN** definiert den allgemeinen Rahmen für TA-Verarbeitung
    X/Open DTP (distributed transaction processing) stellt eine SW-Architektur dar, die mehreren AWP erlaubt, gemeinsam Betriebsmittel mehrerer RMs zu nutzen und die Arbeit der beteiligten RMs durch globale Transaktionen zusammenzufassen.

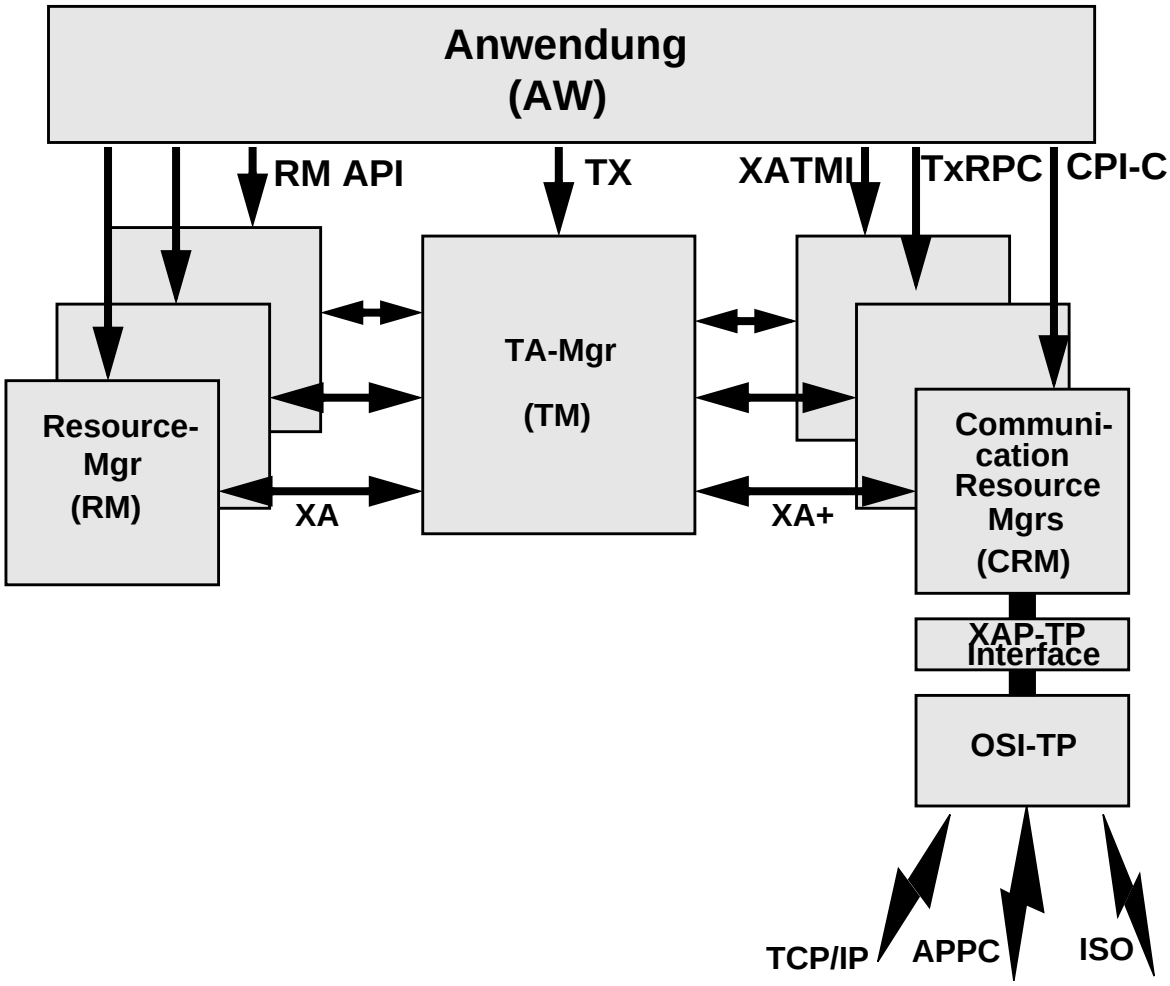
TA-Verarbeitung in offenen Systemen (3)



- **X/Open DTP (1991)** für die lokale Zusammenarbeit von Anwendung (AW), TA-Mgr (TM) und Resource-Mgrs (RMs)
- **Standardisierung**
  - Schnittstelle zur Transaktionsverwaltung (TX: AW - TM)
  - Schnittstelle zwischen TM und RMs
    (XA: zur TA-Verwaltung und ACID-Kontrolle)
  - zusätzlich: Anforderungsschnittstellen (API, z. B. SQL)
- **TA-Ablauf**
  - Die AW startet eine TA, die vom lokalen TA-Mgr verwaltet wird
  - RMs melden sich bei der ersten Dienst-Anforderung beim lokalen TA-Mgr an (Join)
  - Wenn die AW Commit oder Rollback ausführt (oder zwangsweise zurück-gesetzt wird), schickt der TA-Mgr das Ergebnis den RMs (über Broadcast)

➥ in diesem Modell sorgen die RMs für Synchronisation und Logging in ihrem Bereich (private Lock-Mgr und Log-Mgr)

TA-Verarbeitung in offenen Systemen (4)

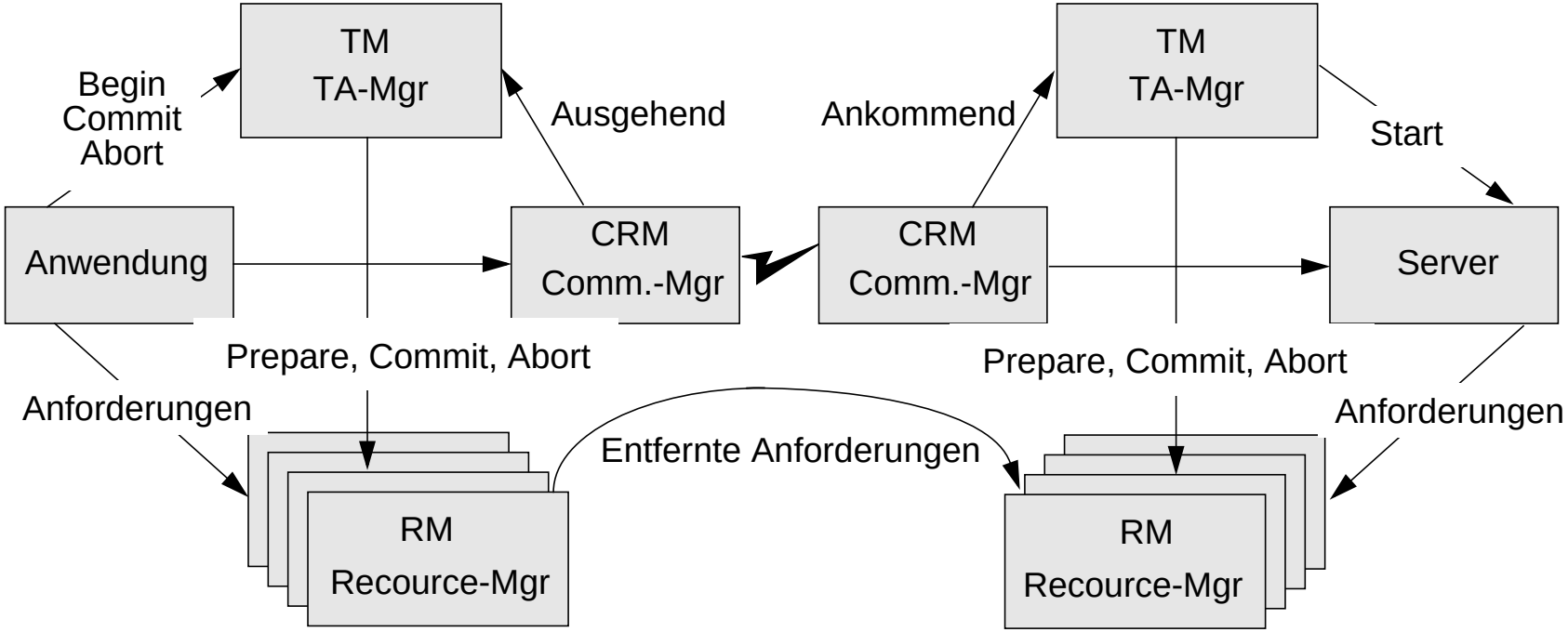


- **X/Open DTP (1993)** standardisiert die verteilte TA-Verarbeitung.
  - Es kommt der „Communication Resource Manager“ (CRM) hinzu.
  - XA+ erweitert die Schnittstelle XA für den verteilten Fall.

- **Definition von Schnittstellen auf der AW-Ebene**
  - TxRPC definiert einen transaktionalen RPC. Die TA-Eigenschaften können ausgeschaltet werden (optional)
  - CPI-C V2 ist eine Konversationsschnittstelle (peer-to-peer), die die OSI-TP-Semantik unterstützen soll
  - XATMI ist eine Konversationsschnittstelle für Client/Server-Beziehungen

  ➥ Sind drei Schnittstellen-Standards erforderlich? Kompromißlösung für drei existierende Protokolle (DCE, CiCS, Tuxedo)

Transaktionsverarbeitung in offenen Systemen (5)



- **TA-Ablauf**
  - Die AW startet eine TA, die vom lokalen TA-Mgr verwaltet wird
  - Wenn die AW oder der RM, der für die AW eine Anforderung bearbeitet, eine entfernte Anforderung durchführen, informieren die CRMs an jedem Knoten ihre lokalen TA-Mgr über die ankommende oder ausgehende TA
  - TA-Mgr verwalten an jedem Knoten jeweils die TA-Arbeit am betreffenden Knoten
  - Wenn die AW COMMIT oder ROLLBACK durchführt oder scheitert, kooperieren alle beteiligten TA-Mgr, um ein atomares und dauerhaftes Commit zu erzielen.

Zusammenfassung (2)

- **Entwicklung verteilter Anwendungen: Programmierte Verteilung**
  - TP-Monitor übernimmt Kommunikation und Transaktionsverwaltung
  - Unterstützung von Knotenautonomie und Heterogenität
  - keine Verteilungstransparenz
  - eingeschränkte Flexibilität des Datenzugriffs
- **X/OPEN DTP:** Interoperabilität über standardisierte Schnittstellen und Protokolle
- **Client/Server-Systeme**
  - Data Shipping ist vor allem bei hoher Referenzlokalität vorteilhaft. Es wird durch das Checkout/Checkin-Konzept genutzt.
  - 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
  - effiziente Anbindung der Werkzeuge
  - neue Transaktionsmodelle für Entwurfs-AW
- **Es gibt eine zunehmende Vielfalt an TPC-Benchmarks**
  - Standardisierung von Arbeitslasten für Transaktionssysteme
  - einfach strukturierte Lasten: TPC-A, TPC-B
  - komplex strukturierte Lasten: TPC-C, TPC-D, TPC-E, . . . (D = *Decision Support*, E = *Enterprise*)
  - TPC-W als kommender Benchmark für Web-DB-Anwendungen

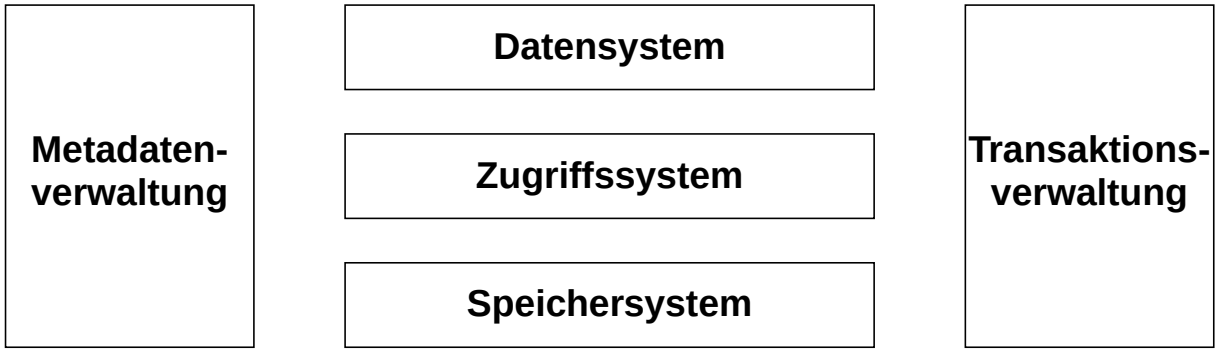
Datenunabhängigkeit im Überblick

Ebene	Was wird verborgen?
Logische Datenstrukturen	Positionsanzeiger und explizite Beziehungskonstrukte 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 Betriebsdetails der externen Speichermedien

Grobarchitektur eines DBS - weitere Komponenten

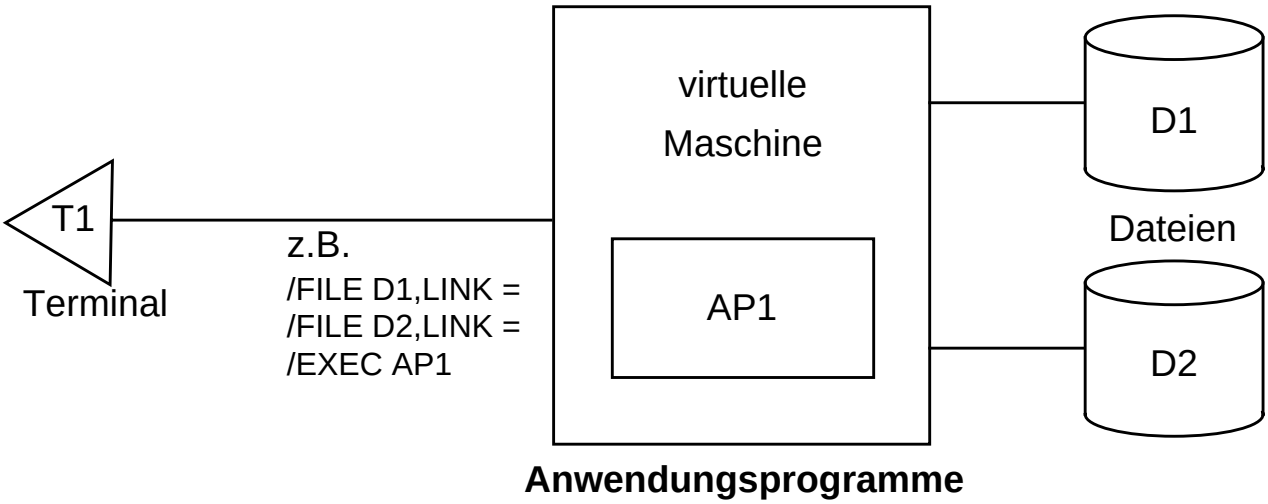
- **Verwaltung der Daten, die Daten beschreiben**
  (DB-Katalog, Integritätsbedingungen, Zugriffskontrollinformationen, ...)

- **Realisierung der ACID-Eigenschaften**
  (Synchronisation, Logging/Recovery, Integritätssicherung)



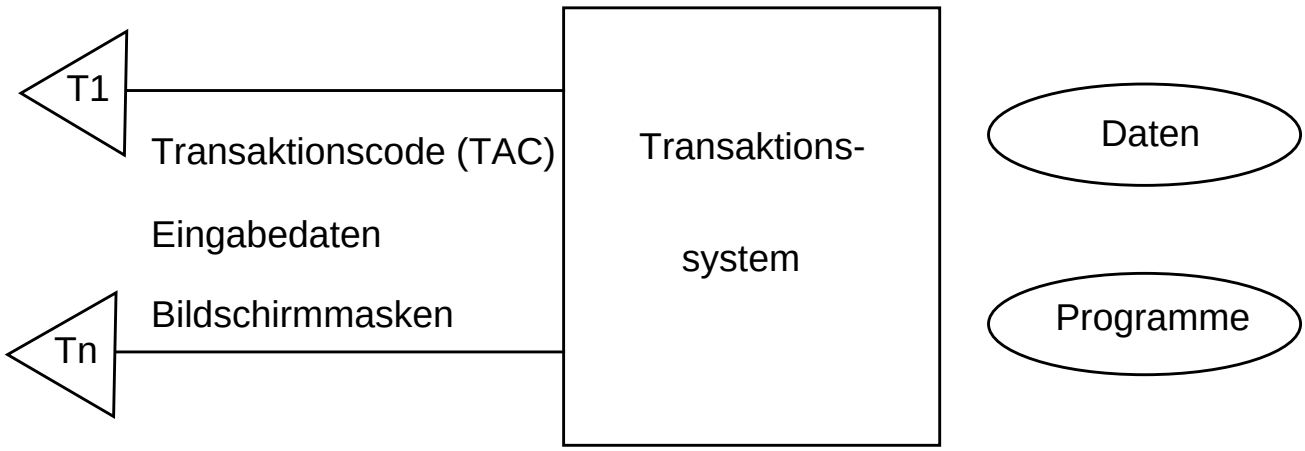
Sicht des Endbenutzers

- **Teilnehmersystem**



- komplexe BS-Schnittstelle
- Kenntnis aller Dateien
- Kenntnisse aller Programme

- **Teilhabersystem**



- problembezogene Funktionen
- einfache Bedienung
- Datenunabhängigkeit
- Programmunabhängigkeit

Entwurfsziele bei Transaktionssystemen

- **Sicht des Endbenutzers**
  - Aufruf von Funktionen (TACs)
  - Dateneingabe über vordefinierte Masken

  ➥ Konsequenz: Programm- und Datenunabhängigkeit

- **Sicht des Anwendungsprogrammierers**
  - Transaktionsprogramme (TAPs): Standardlösungen für immer wiederkehrende Aufgaben
  - Abstraktion für die TAPs: Datenzugriff und Kommunikation

DB-System

- logische Sicht auf die Daten
- Isolation der Programme
  - von den Zugriffspfaden und Speicherungsstrukturen ***(Datenunabhängigkeit)***
  - von den Maßnahmen zur Sicherung und zum Schutz vor Wechselwirkungen ***(Kontrollunabhängigkeit)***

DC-System

- logische Sicht auf die Terminals
- Isolation der Programme
  - von den Kommunikationspfaden und Terminaleigenschaften ***(Kommunikationsunabhängigkeit)***
  - von den Techniken des Multi-Tasking innerhalb eines Prozesses ***(Kontrollunabhängigkeit)***

Anforderungen an das DBS (2)

- **Beispiel: mehr Funktionalität**

(Quelle: Starburst)

- **Neue Datentypen**
  - Vektorgraphik: Grundriß
  - Rasterbilder: Photo des Hauses
  - Video: Tour durchs Haus

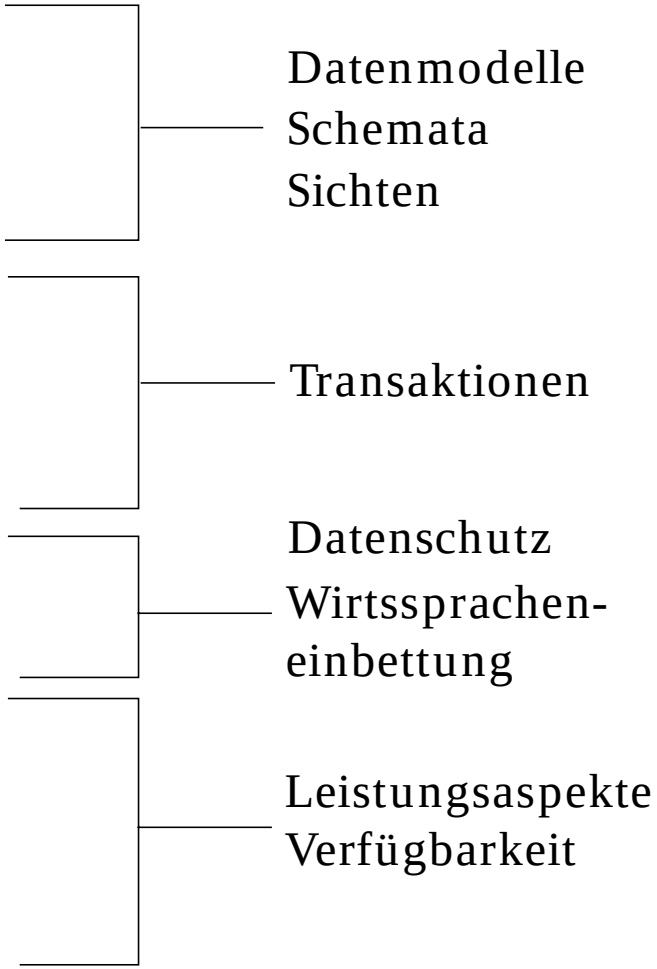
- **Neue Funktionen**

(z. B. DISTANCE_TO , ADJACENT_TO)

➥ Anforderungen werden vor allem durch Objekt-Relationale DBS (SQL3) und ihre „eingebauten“ Erweiterungsmöglichkeiten erfüllt

Wichtige DBS-Konzepte und ihre Einordnung

- Datenintegration
- anwendungsorientierte Datenbeschreibung
- Datenunabhängigkeit
- Konsistenzkontrolle
- Mehrbenutzerbetrieb
- Datenintegrität
- Wiederanlauf
- Zugriffskontrolle
- Programmkopplung
- Ad-hoc-Anfragen
- Zugriffspfade
- Speicherungsstrukturen
- Verteilung



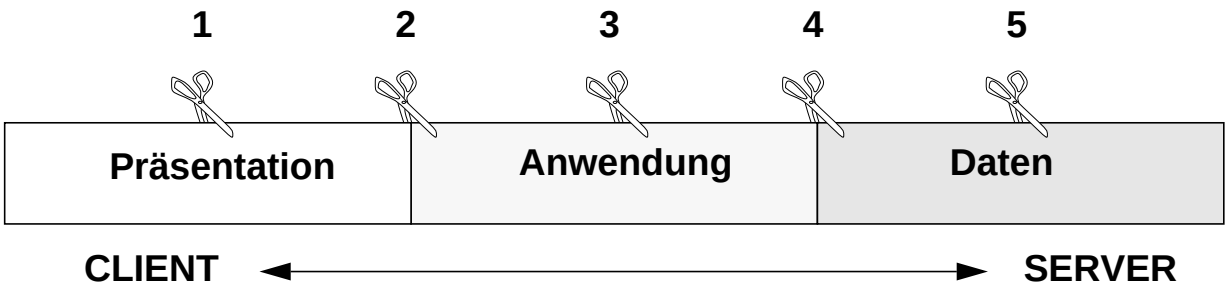
Client/Server-Architekturen

- **Umsetzung von C/S-Architekturen**
  - Für den praktischen Einsatz ist das Client/Server-Modell in ein typischerweise verteiltes Client/Server-System umzusetzen.
  - Das resultierende System heißt auch Client/Server-Architektur

  ➥ Client/Server-Systeme sind nicht an eine bestimmte HW- und SW-Konfiguration gebunden (z. B. Mainframe und PCs)

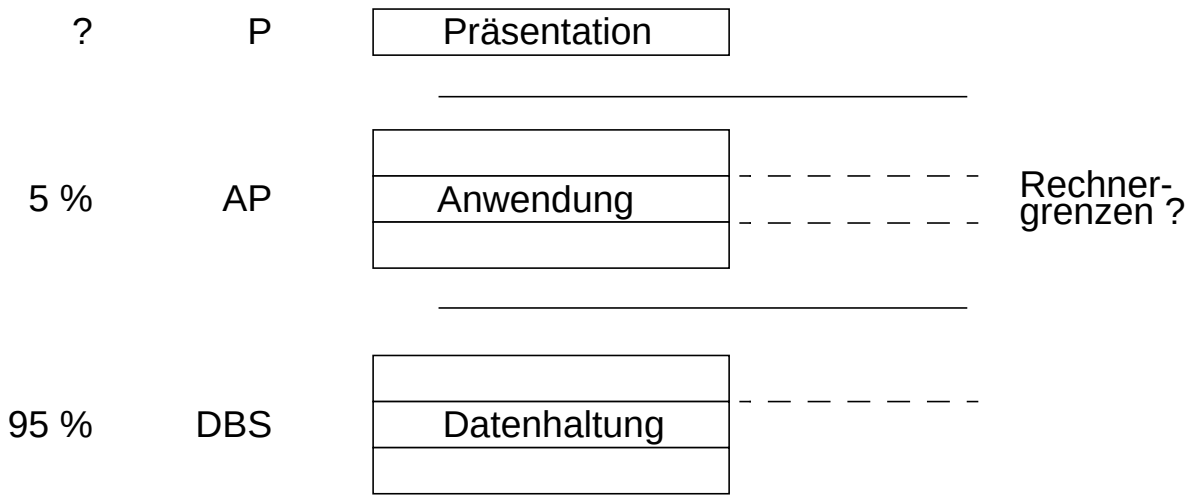
  ➥ Sie implizieren nicht bestimmte SW-Funktionen bei Client oder Server

- **Fünf prinzipiell mögliche Trennlinien**



- **Mögliche C/S-Schnittstellen**

Aufwandsanteile zur Abwicklung einer kurzen TA



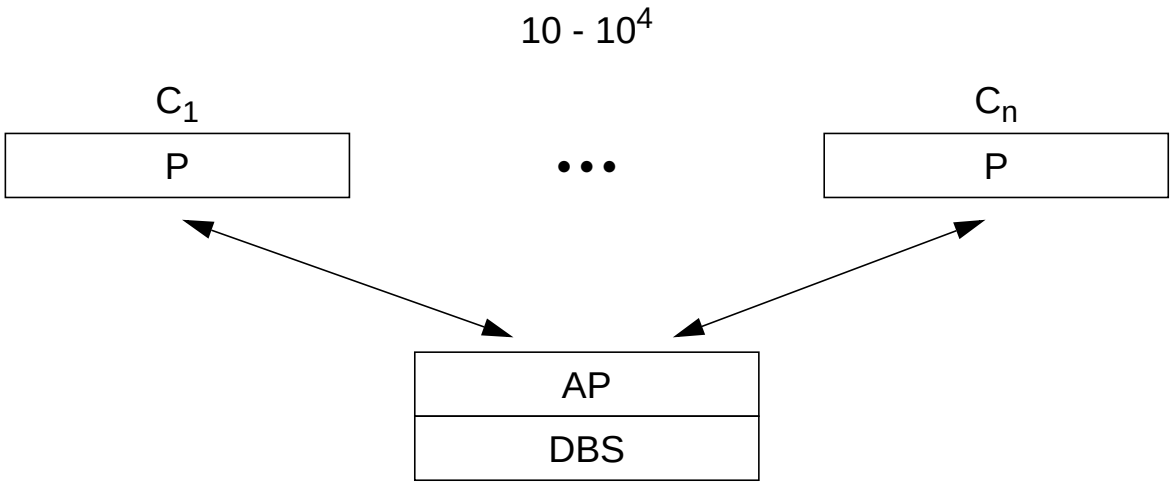
Client/Server-Architekturen (2)

- **Anwendungen**

  Geschäftstransaktionen, jedoch keine CAD-TAs

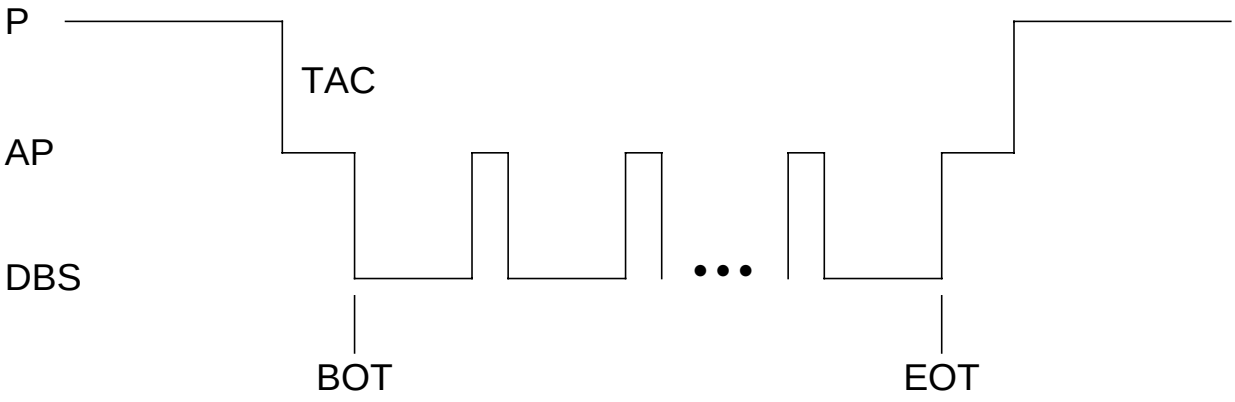
- **3-tier-Architekturen mit einer Rechnergrenze**

  Thin Clients



  - Echte Mehrschritt-Transaktionen bleiben problematisch
  - Nutzung der Client-Ressourcen ?
  - Skalierbarkeit ?

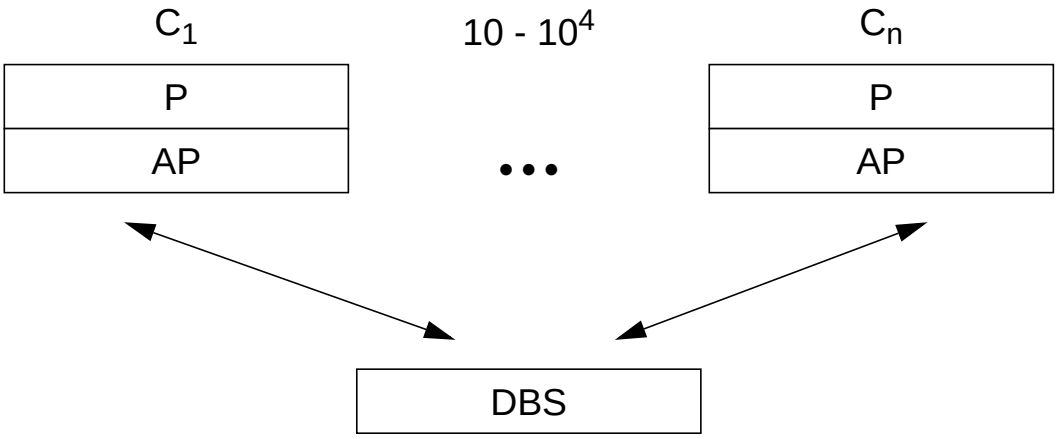
- **Zeitlicher Ablauf einer Transaktion**



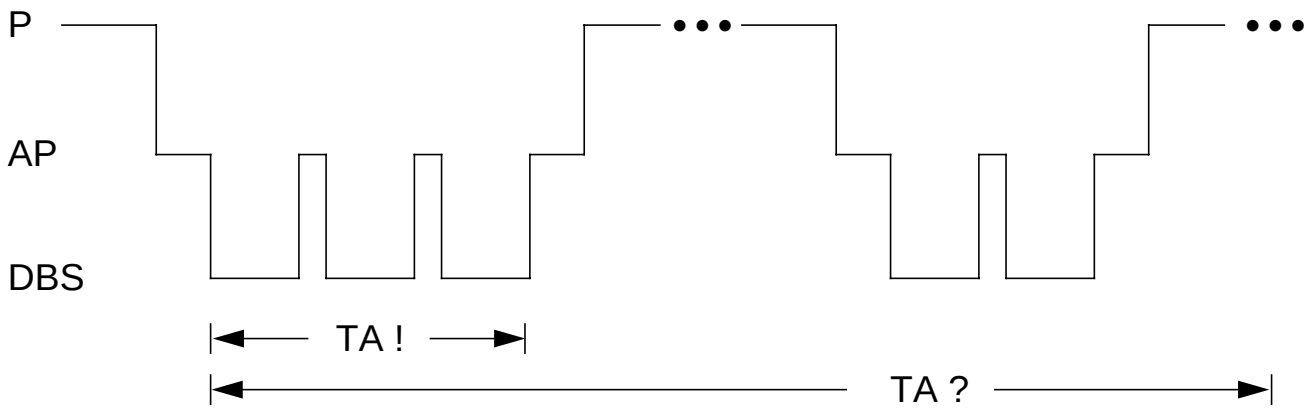
➥ Zeitlicher Ablauf einer TA ist analog zu dem in einem zentralisierten Transaktionssystem (TAP ↔ DBS)

Client/Server-Architekturen (3)

Fat Clients



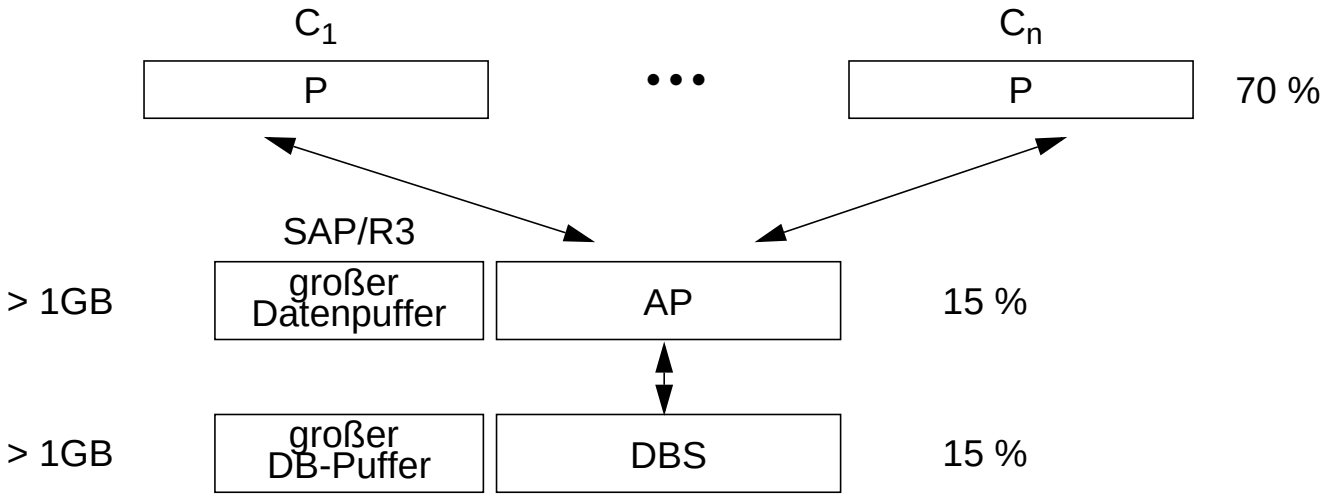
- **Zeitlicher Ablauf einer Transaktion**



- **Eigenschaften**
  - Datentransport zum C_i
  - Zurückschreiben von Änderungen
  - lange Sperrzeiten !
  - für kurze TA (Kontenbuchung) sehr schlecht !
  - sinnvoll bei TAs mit häufiger Rereferenz der Daten ! (Planung, CAx, ...)
  - Achtung: Präsentationslogik sollte nicht im TA-Pfad durchlaufen werden

Client/Server-Architekturen (4)

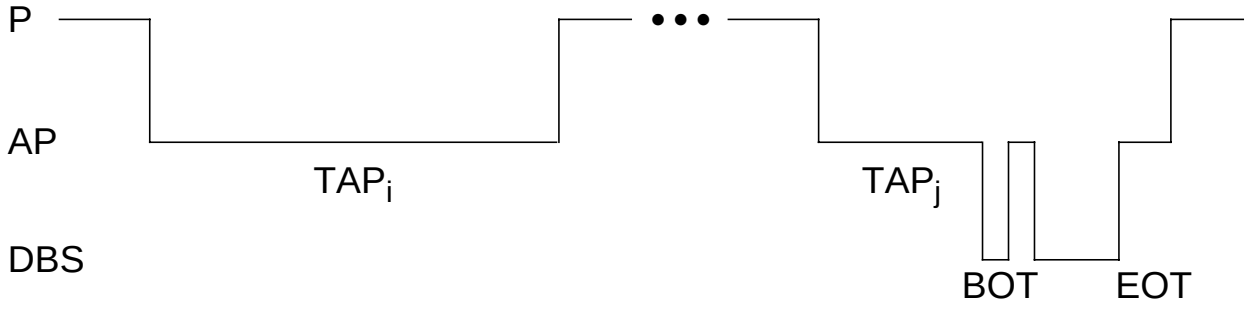
- **3-tier-Architektur mit zwei Rechnergrenzen**



- **Ziele**
  - Anwendungsdaten sollen in der Nähe der Anwendung gehalten werden (lokal im AP-Puffer)
  - Ausnutzung von Anwendungswissen bei der Zugriffssynchronisation auf Ebene des AP-Servers

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

- **Zeitlicher Ablauf einer Transaktion**



- **Eigenschaften**
  - Präsentation mit GUI ist sehr aufwendig; Präsentationslogik (Benutzerinteraktion) sollte nicht im TA-Pfad liegen!
  - Deklarativer, mengenorientierter Zugriff (SQL) auf AP-Puffer ist sehr problematisch
  - Hauptsächlich genutzte DBS-Funktionalität: Datenspeicher und Logging/Recovery