1. Architekturen von TA-Systemen

- **Ziel: Ableitung von Schichtenmodellen für TA-Systeme**
  - Schrittweise Abstraktion von einem Bitstring auf der Magnetplatte zu der Abwicklung von Transaktionen, die durch TACs aufgerufen werden
  - Prinzipieller Aufbau durch Schichtenbildung:
    Konkrete Realisierungen werden als Client/Server-Systeme diskutiert
- **TA-Systeme**
  - Grundlegende Abstraktionen
  - Grobaufbau eines zentralisierten TA-Systems
  - Schichtenmodell
  - Anforderungen
- **Verteilte TA-Systeme**
  - Aspekte/Prinzipien der Verteilung
  - Verteilung unter Kontrolle des TP-Monitors
  - Verteilung unter Kontrolle des DBS (Mehrrechner-DBS)
- **Web als TA-System?**
  - DB-Zugriffe von Web-Anwendungen
  - Datenbank-Caching
  - Probleme und Lösungsvorschläge
- **Benchmarks — Nachbildung von TA-Lasten**
  - Eigenschaften von Benchmarks, Testumgebung
  - TPC-A, -B, -C, -D
  - TPC-H, -R (Verfeinerungen von -D)
  - TPC-W, TPC-App
  - TPC-Benchmarks: Kennzahlen

Bereiche typischer TA-Anwendungen

- **Kommunikationssysteme**
  - Telefonanrufe sind TA
    - Verbindungsaufbau und -freigabe (inkl. Betriebsmittel)
    - Anrufweiterleitung, Benachrichtigungen (Voice Mail), ...
    - komplexe Suche bei 800/900-Nummern
    - Bezahlung kann mehrere Telefon-Firmen betreffen
  - Web: e-Commerce, e-Business, ...
- **Finanzwelt / Banken**
  - Point-of-Service-Terminals
    - Kreditkartenvalidierung
    - Direktbuchung
  - Aktienhandel
- **Reisebüro**
  - Buchungen
  - Fahrkartenverkauf
- **Produktion**
  - Bestellverarbeitung, Einsatz- und Lagerplanung, ...
  - CIM integriert verschiedenartige TA-Anwendungen
    - Just-in-time-Lagerkontrolle
    - Werkzeug- und Werkstück-Transport
    - automatische Handhabungssysteme
    - Qualitätskontrolle, ...
- **Prozesskontrolle**
  - Chemische Abläufe / Reaktionen
  - Energieerzeugung und -verteilung
  - automatisierte Warenhäuser
  - Flugzeuge, ...

➥ Trend zum Einsatz von Standard-SW und TA-Systemen

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.

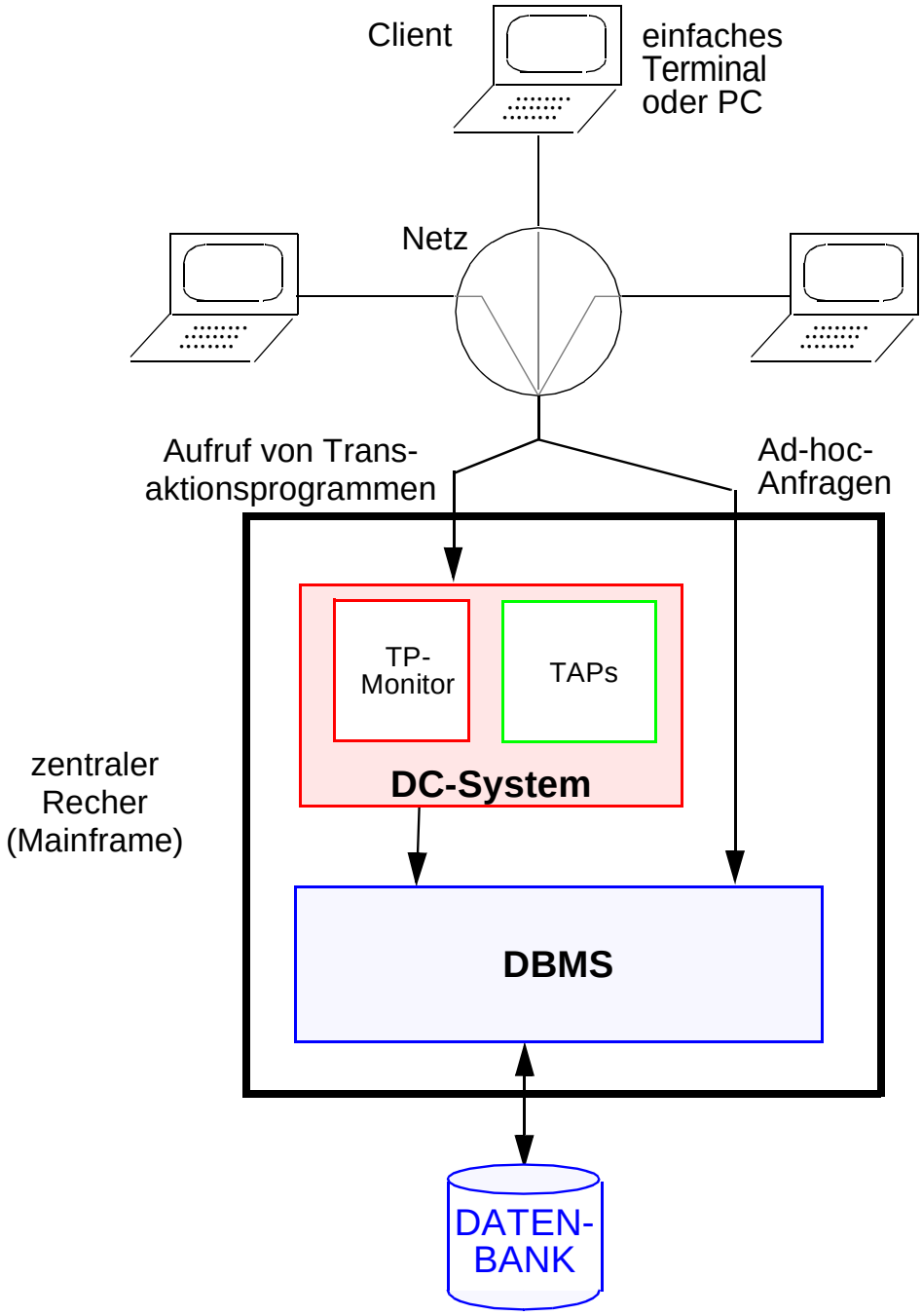
- ***Durability***

  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 TA-Systemes



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

TAP = TA-Programm/Anwendungsprogramm

TP-Monitor = *Transaction Processing Monitor*

DC-System = Datenkommunikationssystem

Entwurfsziele bei TA-Systemen

- **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

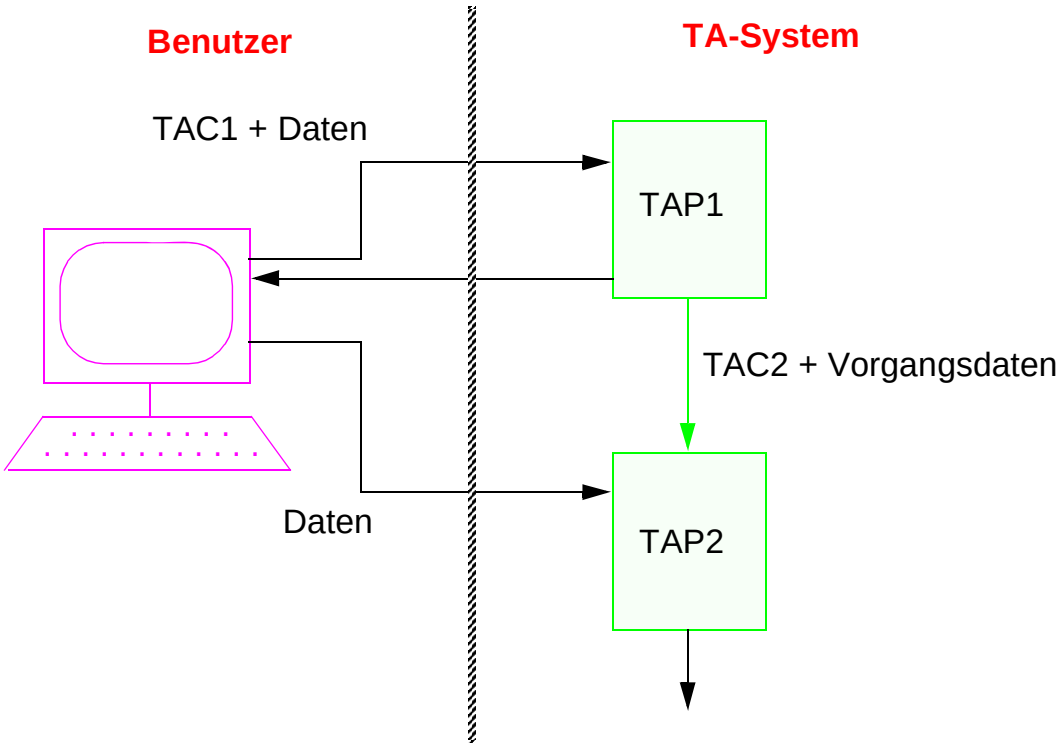
- logische Sicht auf die Daten
- Isolation der Programme
  - von den Zugriffspfaden und Speicherungsstrukturen ***(Datenunabhängigkeit)***
  - von den Maßnahmen zur Sicherung und zum Schutz vor Wechselwirkungen ***(Kontrollunabhängigkeit)***

DC-System

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

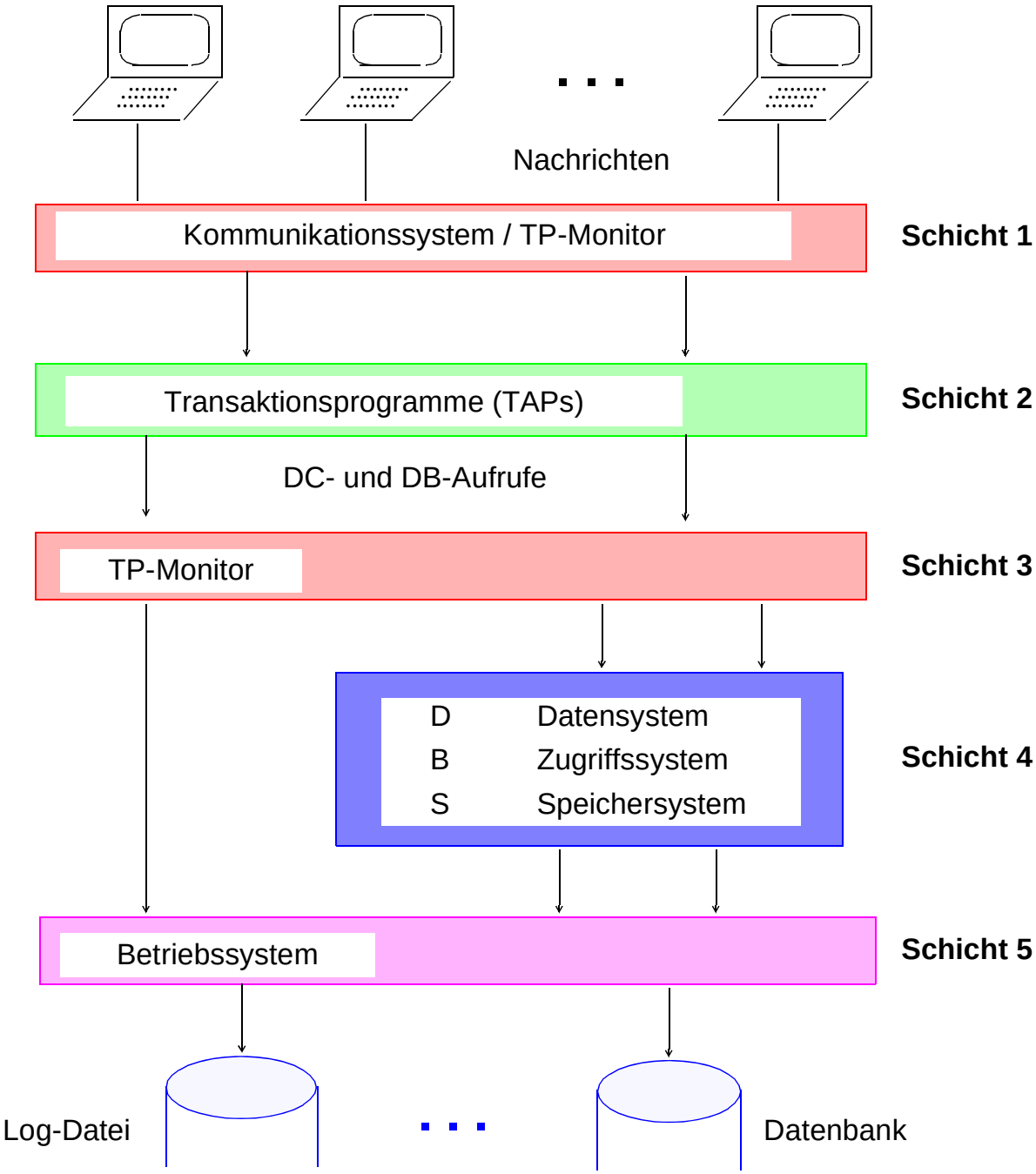
TA-Systeme –
operationale Eigenschaften

- **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 TA-Systems



Anforderungen an TA-Systeme

- **Abwicklung sehr hoher TA-Raten**[1]
  - TA-Typ „Kontenbuchung“: Maßeinheit bei den Benchmarks TPC-A, -B
    ➥ n Ktps vom Typ TPC-B?
  - Komplexere Transaktionen „Abwicklung von Bestellungen“ beim TPC-C
    ➥ $n*10^2$ 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!)
  - n-stufig: Einbezug von Web-Servern (und Präsentations-Servern)

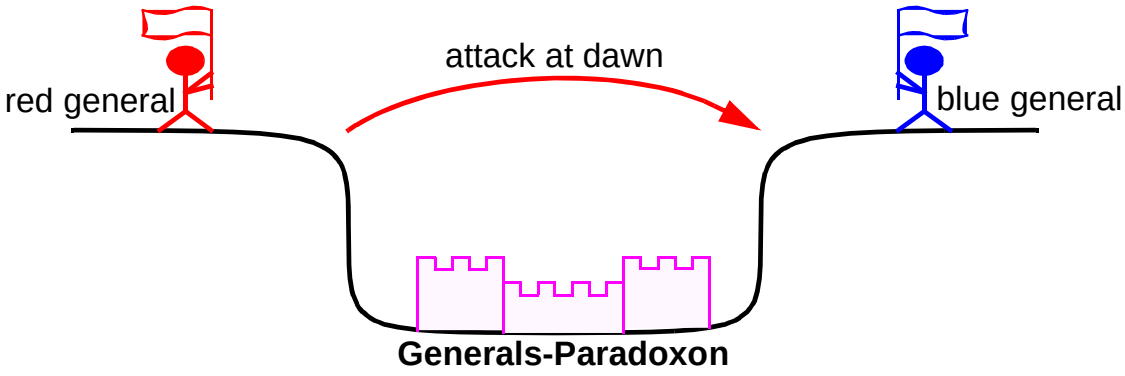
➥ **TA-Systeme sind** i. Allg. horizontal und/oder vertikal **verteilt**!

1. Sehr große TA-Systeme: 150.000 nebenläufige Benutzer/System auf CICS und 750.000 auf TPF

Verteilte TA-Systeme

- Ein ***verteiltes System*** besteht aus autonomen Subsystemen, die koordiniert zusammenarbeiten, um eine gemeinsame Aufgabe zu erfüllen
  - Client/Server-Systeme
  - Mehrrechner-DBS, . . .

- **Beispiel: The „Coordinated Attack“ Problem**



Generals-Paradoxon

- **Grundproblem verteilter Systeme**

> *Das für verteilte Systeme charakteristische Kernproblem ist der Mangel an globalem (zentralisiertem) Wissen*

➥ **symmetrische Kontrollalgorithmen sind oft zu teuer oder zu ineffektiv**

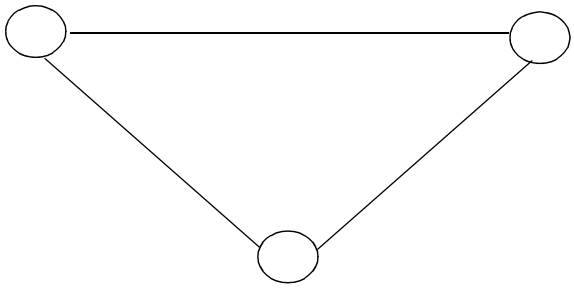
➥ **fallweise Zuordnung der Kontrolle**

- **Annahmen im allgemeinen Fall**
  - nicht nur Verlust (omission) von Nachrichten (Bote/Kurier wird gefangen)
  - sondern auch ihre bösartige Verfälschung (commission failures)

➥ **dann komplexere Protokolle erforderlich:**
Distributed consensus (bekannt als Byzantine agreement)

Verteilte TA-Systeme (2)

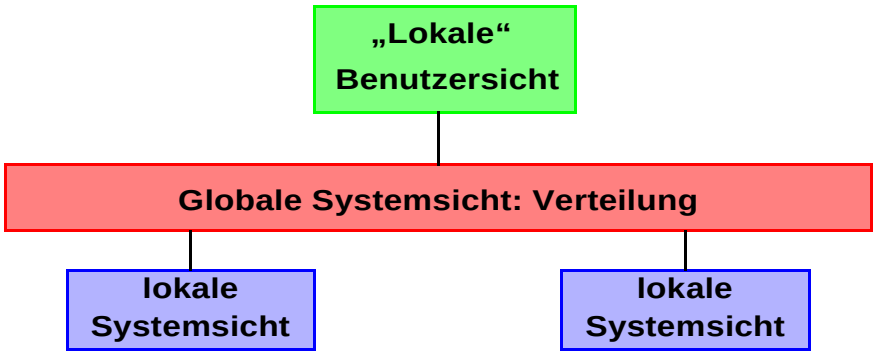
- **Wirtschaftliche Gründe**
  - reduzierte HW-Kosten
  - verfügbare Kommunikationseinrichtungen
- **Organisatorische Gründe**
  - lokale Unterstützung organisatorischer Strukturen
  - Integration existierender Datenbanken
  - lokale Autonomie
- **Technische Gründe**
  - Erhöhung der Leistung: Lokalität 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!

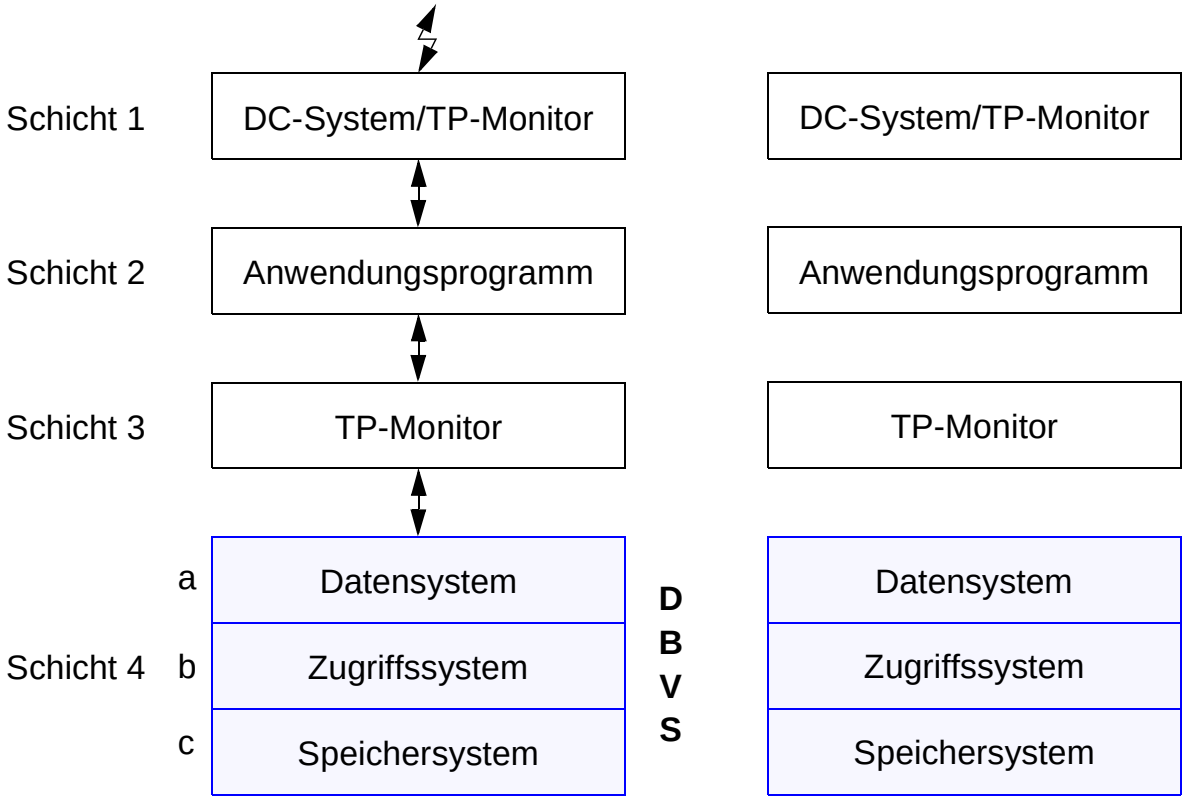
Verteilte TA-Systeme (3)

- **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 TA-Systeme (4)

- **Homogenes Systemmodell**



- Forderung: lokale Sicht des Benutzers
  (Ortstransparenz, „single system image“)

- **Realisierung der globalen Systemsicht: wo?**

- Abbildung auf lokale Sichten der Knoten

Klassifikation verteilter TA-Systeme

- **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:**

  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

Verteilte Transaktionsanwendungen

- **Vergleich der Verfahren zur TAP-Anbindung an DB-Server**
  - mehrere Datenquellen – homogen oder heterogen
  - mehrere eigenständige DBMS
  - knotenübergreifende Anwendungsfunktionen wünschenswert

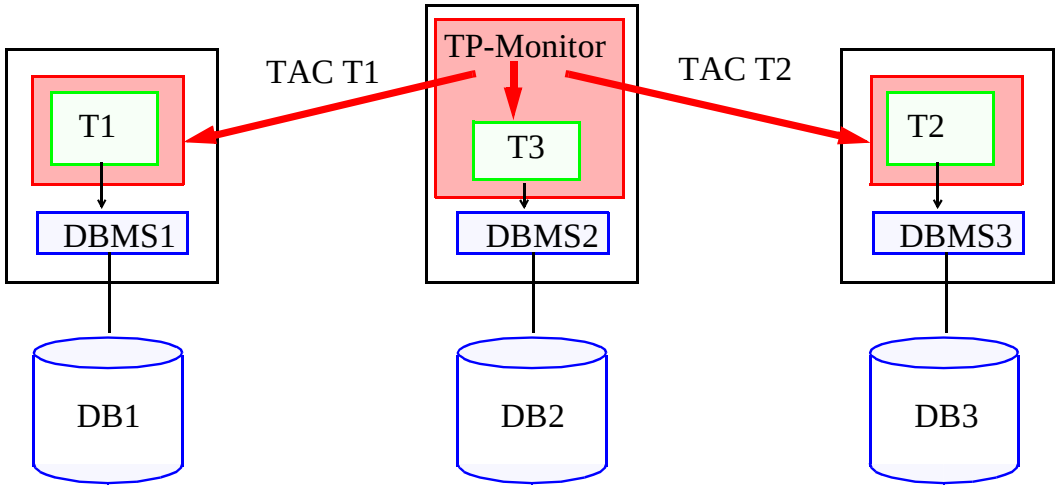
	Transaction Routing	Programmierte Verteilung	Aufruf von DB-Operationen
Granulat der Verteilung			
Heterogenität der Daten			
Ressourcen des TAP			
Übergreifende Funktionen möglich			
globales Schema erforderlich			
Abwicklung verteilter TAs			

- **Anwendungsanbindung bei VDBS**
  - eine logische, lokale DB-Sicht: ein globales DB-Schema
  - eine gleichförmige AP-DB-Schnittstelle (API), z. B. SQL

- **Gleichförmiges API bei heterogenen Datenquellen wünschenswert!**

Transaction Routing

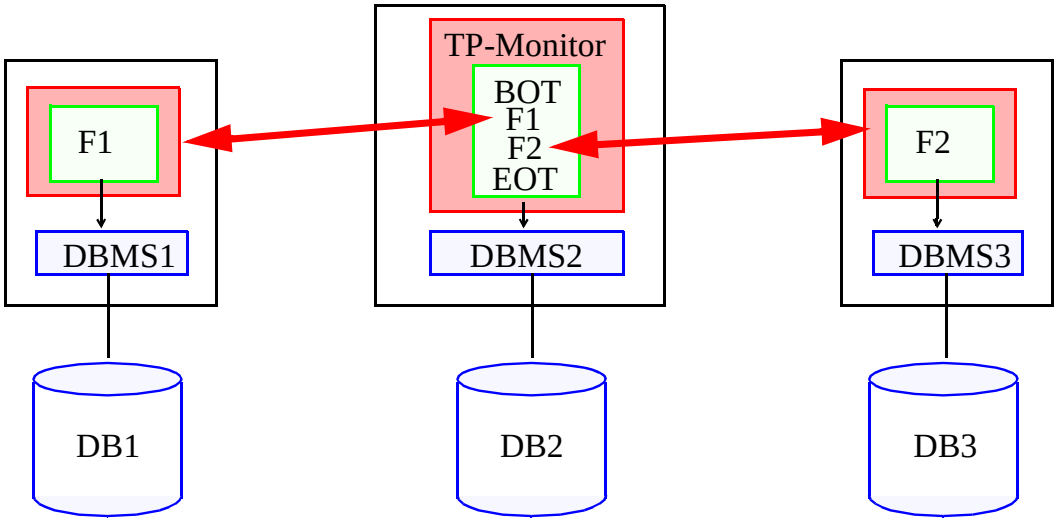
- **Prinzip**
  - Jeder TA-Typ ist einem Rechner fest zugeordnet
  - TP-Monitor kennt TA-Typ-Zuordnung: Erkennung und Weiterleitung des TAC (→ Ortstransparenz)
  - lokale Transaktionsausführung innerhalb eines Rechners
  - Ausgabenachricht muss ggf. über Zwischenrechner an Benutzer zurückgesendet werden



- **Keine Kooperation zwischen DBMS**
  - pro Rechner eigene DB / Schemata
  - heterogene DBS möglich
- TA als alleiniges Verteilgranulat zu inflexibel
  (keine echt verteilte Transaktionsausführung)
- Beispiel: IMS MSC (*multiple systems coupling*)

Programmierte Verteilung

- **Verteilung auf Ebene von Anwendungsprogrammen**
  - Zugriff auf externe DB durch Aufruf eines Teilprogrammes (RPC) an dem betreffenden Rechner
  - Jedes Teilprogramm greift nur auf lokale DB zu (mit jeweiliger Anfragesprache)



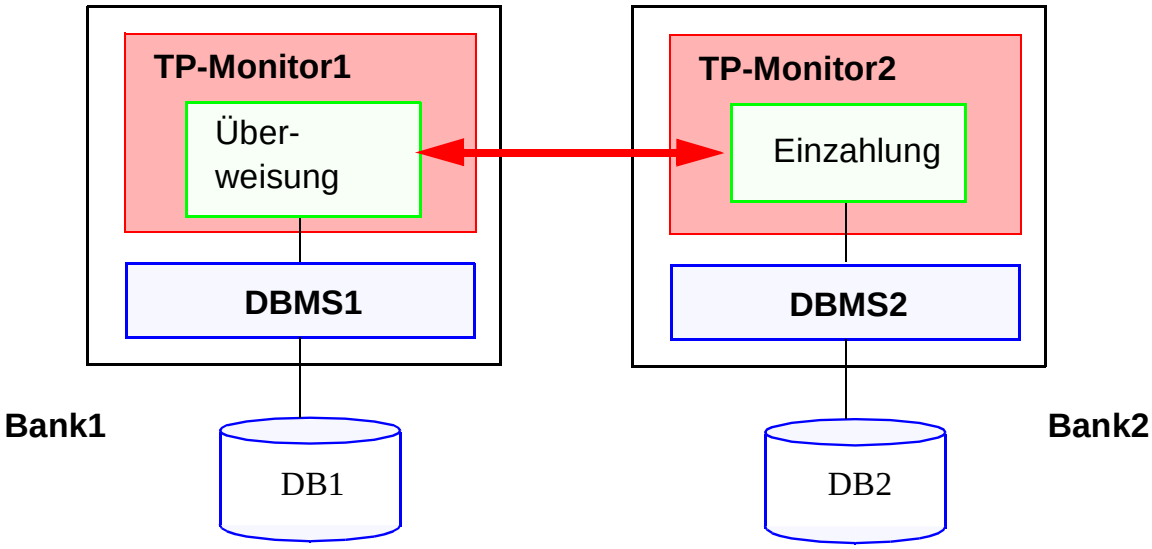
- **Transaktionsverwaltung**
  - TP-Monitor koordiniert verteiltes Commit-Protokoll
  - DBS müssen DB-Operationen von nicht-lokalen Transaktionen akzeptieren sowie am Commit-Protokoll teilnehmen
  - Auflösung globaler Deadlocks über Timeout

- **Ortstransparenz** kann prinzipiell durch TP-Monitor erreicht werden

  Beispiel: Tandem Pathway
  UTM-D
  CICS Distributed Transaction Processing

Programmierte Verteilung: Beispiel

Überweisung zwischen unabhängigen Banken



Bemerkung: Solche verteilten AW werden zunehmend asynchron mit Hilfe von persistenten Warteschlangen abgewickelt

Transaktionsprogramm in Bank1 für Überweisung von Summe S von Konto K1 auf Konto K2 von Bank2:

Eingabenachricht (Parameter) lesen
BEGIN WORK
{ Zugriff auf lokale DB zur Abhebung des Betrags }

```
UPDATE ACCOUNT
SET BALANCE = BALANCE - :S
WHERE ACCT_NO = :K1;
```

• • •

CALL EINZAHLUNG (Bank2, S, K2)

COMMIT WORK

Ausgabenachricht ausgeben

Transaktionsprogramm EINZAHLUNG in Bank2

Parameter übernehmen

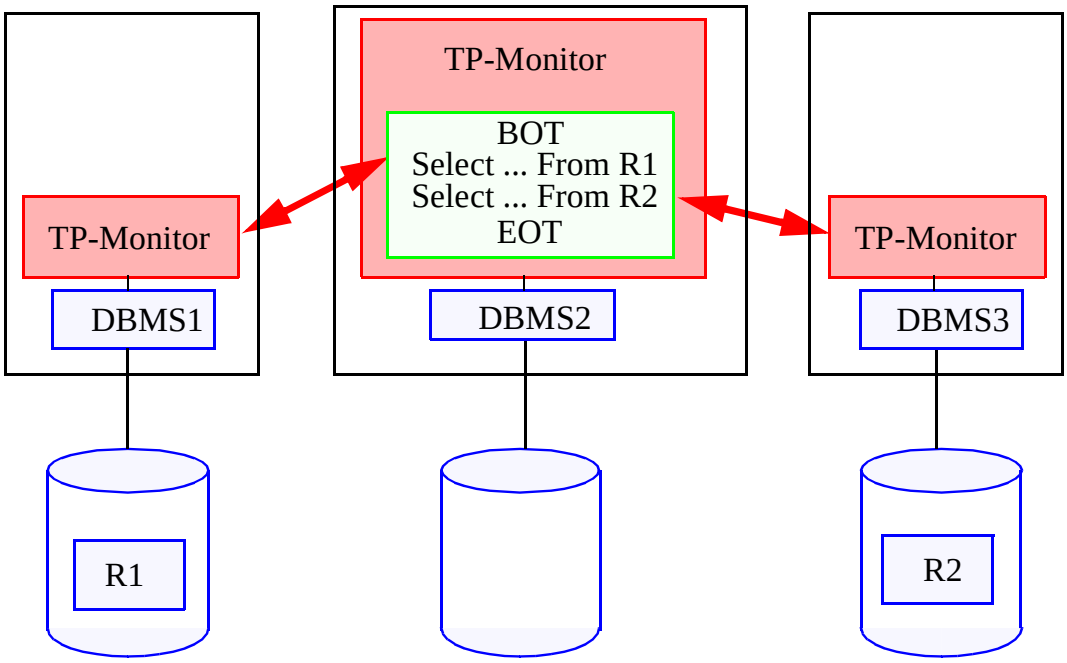
• • •

```
UPDATE GIROKTO
SET KSTAND := KSTAND+ :S
WHERE KNUMMER = :K2;
```

• • •

Ausführung zurückmelden

Verteilung von DB-Operationen (durch TP-Monitor)



- **DB-Operationen als Verteileinheiten**
  - AP können auf entfernte Datenbanken zugreifen; Aufrufweiterleitung durch *Function Request Shipping*
  - Programmierer sieht mehrere Schemata; jede Operation muss innerhalb eines Schemas (DB-Partition) ausführbar sein
  - Gemeinsame Anfragesprache wünschenswert
  - Ort der Verteilung kann für AP prinzipiell transparent gehalten werden

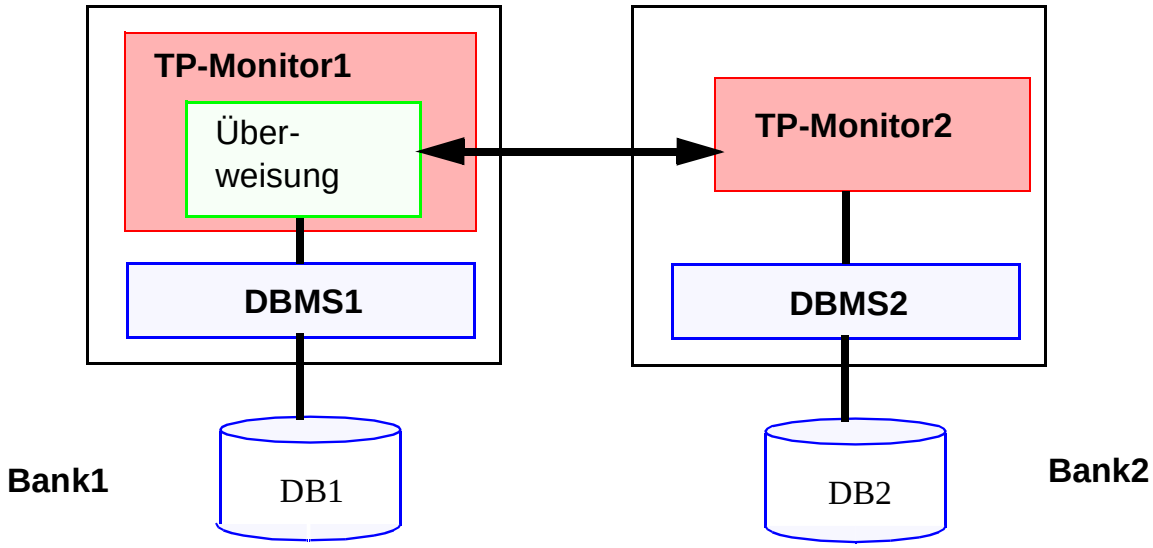
- **Transaktionsverwaltung**

  im Prinzip wie bei Programmierter Verteilung

- Beispiel:

  CICS *Function Request Shipping*

Verteilung von DB-Operationen: Beispiel



Bemerkung: Bei heterogenen DBMS spricht man auch von Gateway-Einsatz.
Die AW sieht hier die verschiedenen DBMS-Schnittstellen (heterogenes API).

Transaktionsprogramm „Überweisung“

Eingabenachricht (Parameter) lesen

BEGIN WORK

CONNECT TO R1.DB1 ...

UPDATE ACCOUNT
SET BALANCE = BALANCE - :S
WHERE ACCT_NO = :K1;

CONNECT TO R2.DB2 ...

UPDATE GIROKTO
SET KSTAND := KSTAND+ :S
WHERE KNUMMER = :K2;

• • •

COMMIT WORK

DISCONNECT ...

Ausgabenachricht ausgeben

Verteilte TA-Systeme: Bewertung

- **Transaction Routing**
  - \+ sehr einfach, ...
  - \+ in heterogenen Umgebungen mit HTTP/HTML
  - \- sehr starr, keine knotenübergreifenden Transaktionen
  - \- ...

- **Programmierte Verteilung**
  - \+ sehr hohe Unabhängigkeit gewährleistet
  - \+ Unterstützung von heterogenen DBS und Client-/Server-Ansatz
  - \+ geringe Kommunikationshäufigkeit
  - \+ Nutzung bestehender Anwendungsfunktionen möglich
  - \+ gute Administrierbarkeit
  - \+ von vielen Systemen unterstützt
  - \- sehr geringe Flexibilität des Datenzugriffs (nur Aufruf bestimmter Anwendungsfunktionen möglich, keine Ad-hoc-Anfragen)
  - \- keine rechnerübergreifenden DB-Operationen
  - \- geringes Maß an Verteilungstransparenz
  - \- schwierige Programmierung (Behandlung von Fehlerfällen)
  - \- weitere Vorteile verteilter DBS entfallen: hohe Verfügbarkeit, systemkontrollierte Datenreplikation, Fragmentierung

- **Verteilung von DB-Operationen**
  - \+ größere Freiheitsgrade für DB-Zugriff
  - \- Verteilungstransparenz, Administrierbarkeit und Kommunikationshäufigkeit schlechter als bei programmierter Verteilung
  - \- schwierige Programmierung (mehrere DB-Schemata)
  - \- Bereitstellung einer einheitlichen Programmierschnittstelle (API) für DB-Zugriff und Kommunikation erforderlich

Klassifikation verteilter TA-Systeme (2)

- **Verteilung unter Kontrolle des DBS**

  (➥ Mehrrechner-DBS)

  - **Achtung: Rechner können einfach repliziert werden, Daten nicht!**
  - Vollständige Verteilungstransparenz für TAP setzt eine logische Datenbank voraus
  - geringe Eignung für Knotenautonomie und Heterogenität

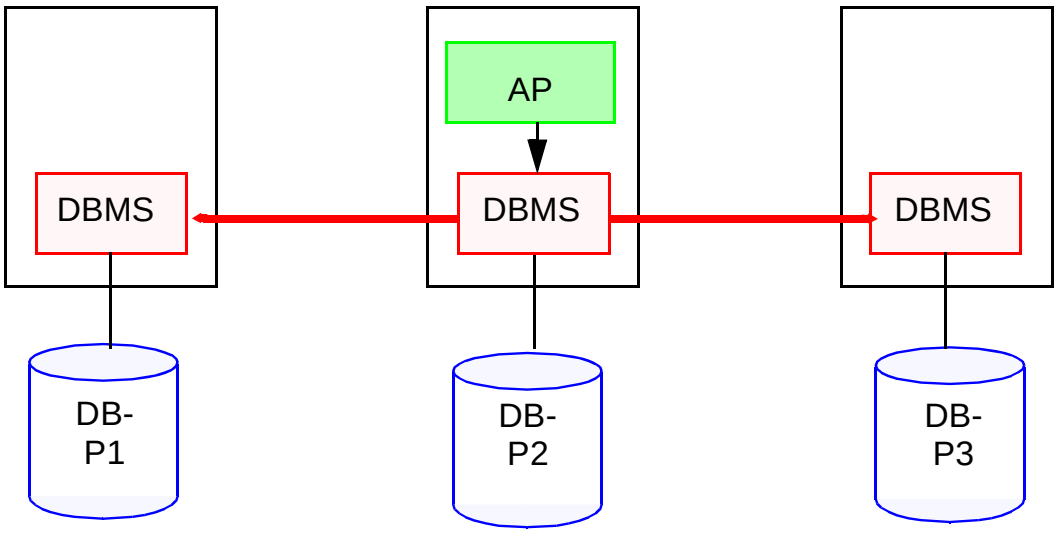
- **Mehrere (homogene) Realisierungsalternativen**

  - globale Sicht in Schicht 4
  - **Verteilung ganzer DML-Befehle bzw. von Teil-Operationen (Schicht 4a)**
  - Verteilung von internen Satzoperationen (Schicht 4b)
  - Verteilung von Seitenzugriffen (Schicht 4c)

- **Mischformen möglich**

  - Datenverteilung unter ausschließlicher Kontrolle eines verteilten DBS
  - partitionierte Datenverteilung unter Kontrolle mehrerer verteilter DBS
  - Bei Einsatz heterogener DBS oder DBS-Föderationen ist eine Homogenisierung durch Bereitstellung globaler Sichten erforderlich

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 (SN)*, 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**

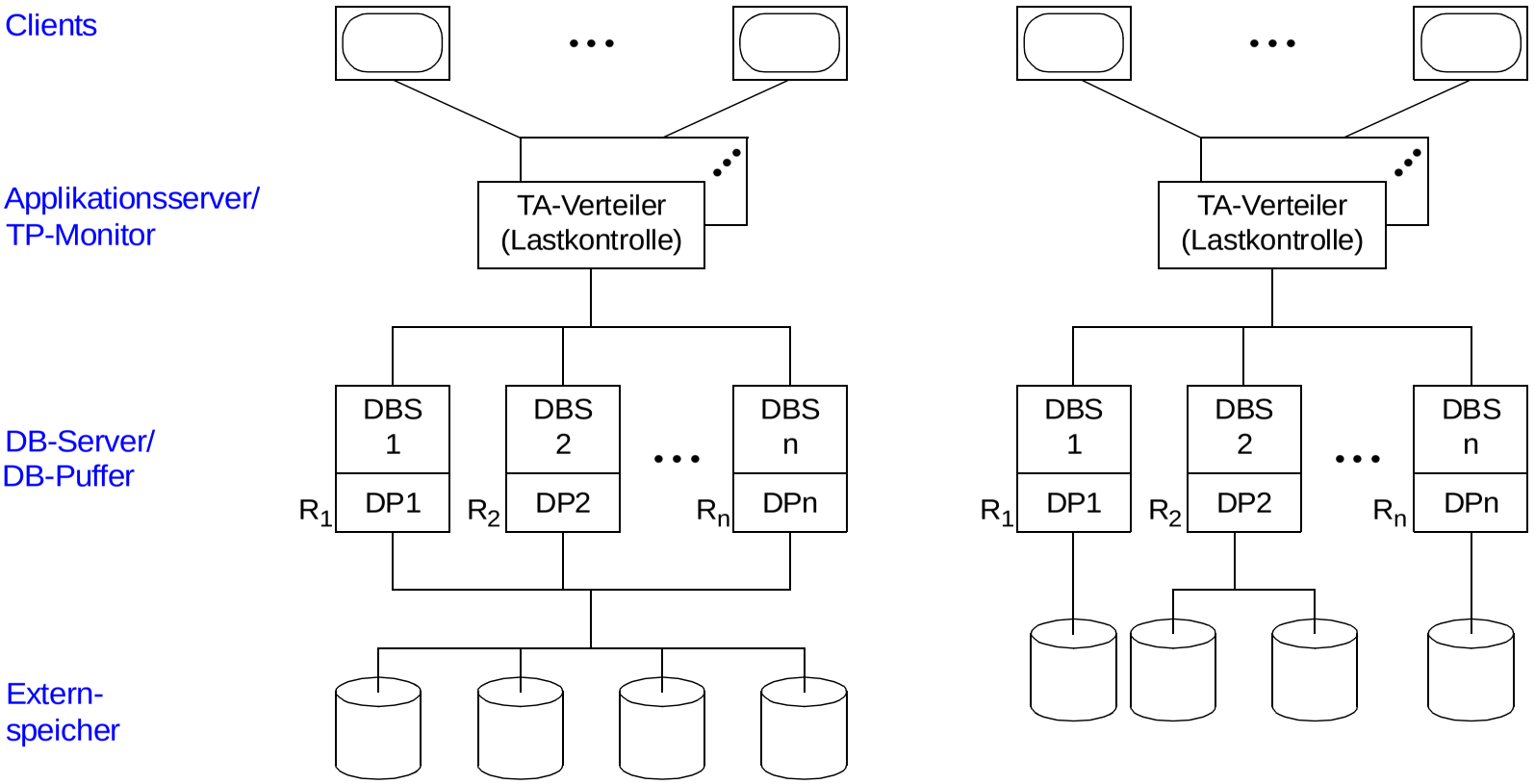
  2. DB-Sharing (*Shared Disk (SD)*)

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

Grobarchitektur

SD-System

SN-System



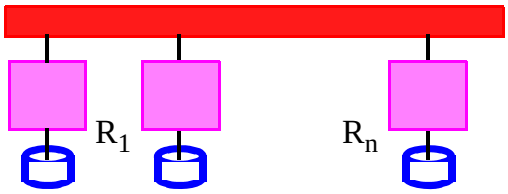
Konfiguration:

- **Rechneranzahl ?**
- **breitbandige Kommunikationsmöglichkeit/lokale Anordnung**
- **kein “single point of failure”**
- **DB-System auf jedem Rechner**

SN vs. SD: Leistungsfähigkeit

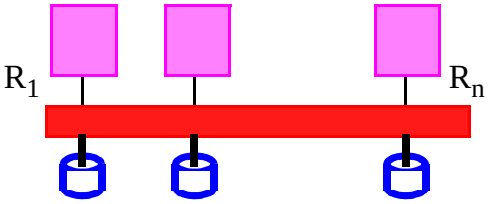
- **starke Abhängigkeiten zu**
  - Anwendungscharakteristika und
  - Realisierung einzelner Systemfunktionen

- **Shared-Nothing:**



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

- **Shared-Disk:**



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

Klassifikation verteilter TA-Systeme (3)

- **Bereitstellen globaler Sichten bei heterogenen DBS**

  DBS bietet für das Anwendungsprogramm eine vereinheitlichte Sicht auf Daten aus heterogenen Datenquellen (einheitliche Programmierschnittstelle (API))

  - großes Spektrum: DBS, Datei-, Multimedia-, Spreadsheet-, Mail-Systeme
  - Herausragende Rolle bei **Informationsintegration**
    - Integration einer großen Bandbreite an Datentypen für eine Anwendung möglich
    - Einsatz u. a. bei Legacy-Systemen
    - Wichtiges Forschungsthema: **Dynamische Informationsfusion**
  - Schwierigkeiten bei heterogenen Datenquellen (Web) nicht zu unterschätzen:
    - Homogenisierung auf Schema- und Instanzenebene
    - Einsatz von Ontologien usw.
  - **zwei wesentliche Alternativen:**
    1. Schemaabbildung und Anfragetransformation (*schema mapping*, *view translation*)
    2. Middleware-Übersetzer/Optimierer mit Wrapper-Einsatz

- **Realisierung allgemeiner Resource-Manager-Architekturen**

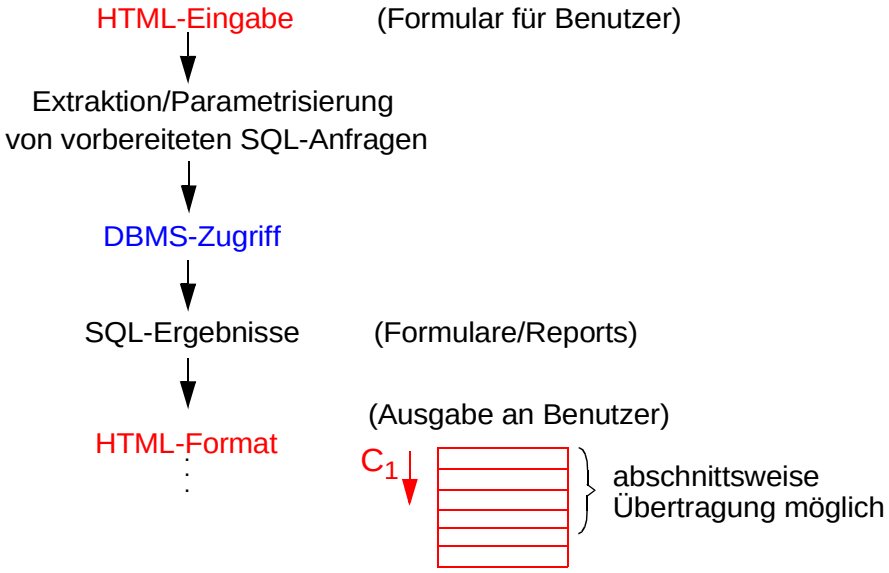
  - Resource Manager (RMs) sind Systemkomponenten, die **Transaktionsschutz für ihre gemeinsam nutzbaren Betriebsmittel** (BM) bieten
  - Sie gestatten die **externe Koordination** von BM-Aktualisierungen durch ein 2PC-Protokoll

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

➥ Wie ist das Web als „TA-System“ einzuordnen?

Web als TA-System

- **Ist das Programmiermodell von SQL für Web-Anschluss geeignet?**
  - deklarative Anfragesprache (keine Navigation)
  - Mengenorientierung
  - leichte und effiziente Übertragung von Ergebnismengen (Cursor)
- **Prinzipielle Vorgehensweise beim DB-Zugriff übers Web**

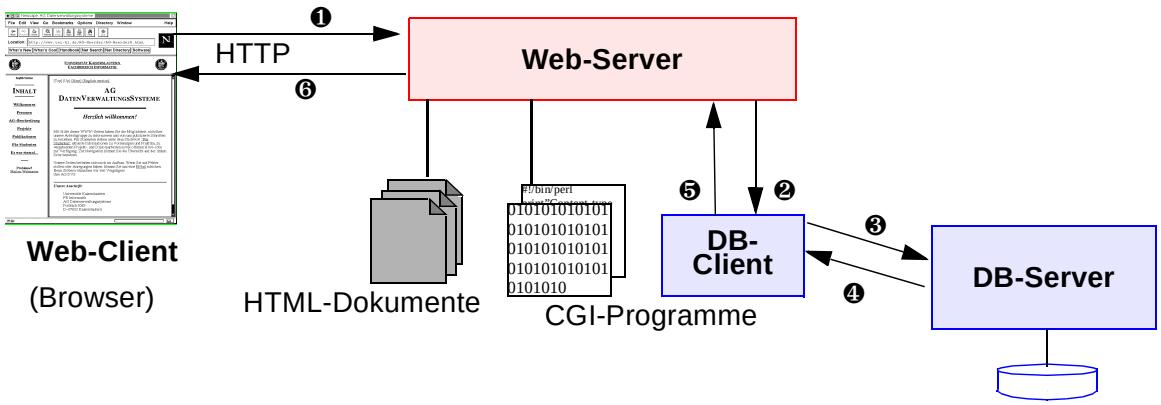


Zusätzliche Zugriffe auf DBMS und HTML-Seiten, beispielsweise von eingebetteten HyperLinks in den ausgegebenen Ergebniszeilen (Reports)

- **DB-Zugriff über CGI / Web-Server-API / JSP / ASP**
  - Zustandslose Verbindung zwischen Browser und Web-Server über HTTP (*HyperText Transfer Protocol*)
  - DB-Zugriff durch externes CGI-Programm (*Common Gateway Interface*) bzw. Server-Erweiterungsmodul, Active Server Pages oder Java Server Pages als DB-Client
  - HTML (*HyperText Markup Language*) dient zur Ergebnispräsentation
- **DB-Zugriff über Java oder ActiveX**
  - Java-Applet als direkter DB-Client (Ausnahmen möglich) mit SQLJ oder JDBC (*Java DataBase Connectivity*) als Schnittstelle zu RDBMS
  - ActiveX Controls mit ADO.NET als Alternative zu Java/Applets

Web als TA-System (2)

- **Anbindung über CGI**



- **Vorgehensweise:**
  - ❶ Abschicken des Formulars mit aktuellen Parameterwerten
  - ❷ Starten des entsprechenden CGI-Programms (DB-Client)
  - ❸ Anfragen des CGI-Programms an den DB-Server
  - ❹ Übertragung der Ergebnismenge zum DB-Client
  - ❺ Rückgabe der erstellten HTML-Seite an den Web-Server
  - ❻ Übertragung der HTML-Seite zum Browser

- **Vor-/Nachteile von CGI:**

  \+ wenig Ressourcen bzw. Voraussetzungen beim Benutzer erforderlich

  \+ (einfache) zentrale und damit kostengünstige Administration

  \+ Verschlüsselung im Rahmen von Verfahren für HTTP möglich

  \- **Zustandslosigkeit von HTTP beschränkt Transaktionslänge** (ein Request)

  \- mehrstufige Aufrufhierarchie => Web-Server mit DB-Client pro Request wird Engpass – CGI-Prog. wird in eigenem Prozess gestartet (Optimierung durch fastCGI mit Prozess/Pool)

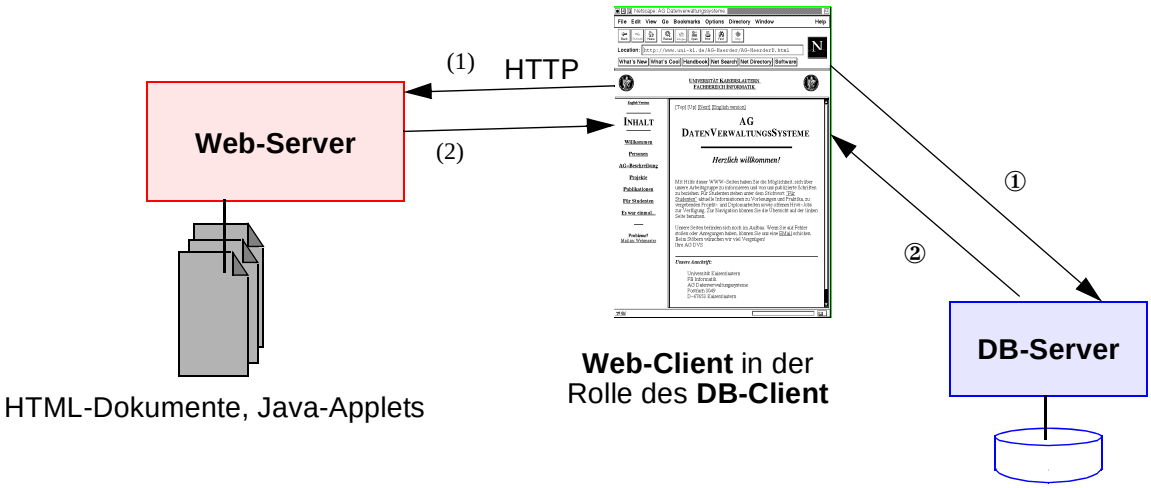
  \- Unsicherheit bzgl. der Ausführung (Fehler an jeder Stelle möglich)

  \- **Zugangskontrolle schwierig zu realisieren**, da Benutzerzuordnung problematisch; erfordert entweder Mapping von User-IDs (von Web-Client zu DB-Client) oder zusätzliche DB-Client- oder DB-Server-seitige Logik (z. B. SQL-Anfragen zur Prüfung von Rechten)

  ➥ **für kleine Datenmengen und seltene Nutzung geeignet (AW-Sicht)**

Web als TA-System (3)

- **Anbindung über Java auf Web-Client-Seite**



- **Vorgehensweise (zweistufig):**

  (1) Anfrage an Web-Server zum Laden des Applets

  (2) Laden des Applets

  ① DB-Anfrage des Applets (JDBC, eigene Protokolle)

  ② Ergebnisübertragung der DB-Anfrage an das Applet

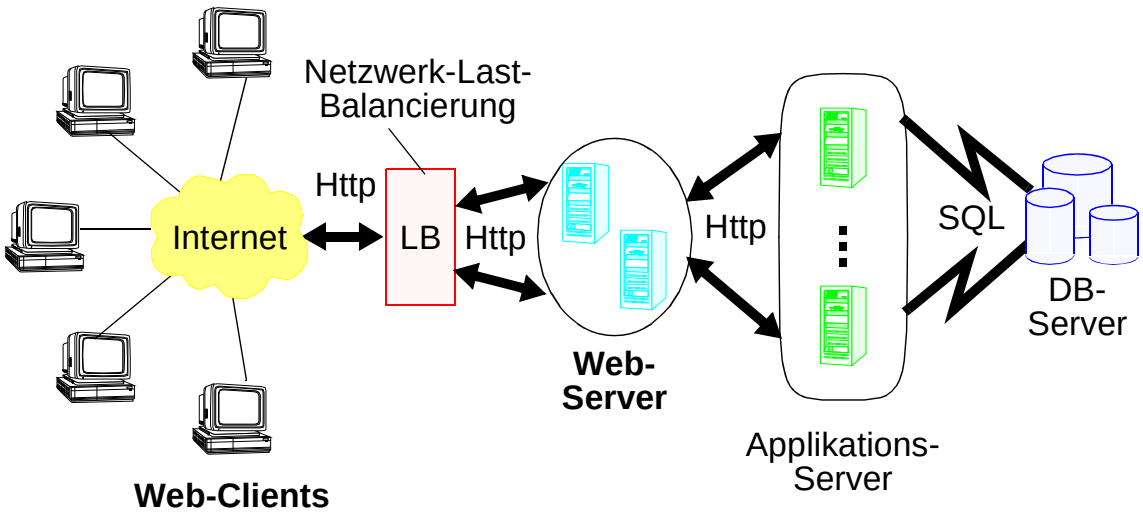
  ① & ② beliebig oft wiederholen

- **Vor-/Nachteile Java:**

  \+ Web-Server nur für die Bereitstellung des Applets nötig
  \+ DB-Client (Applet) kann eigene Anwendungslogik besitzen
  \+ Datenverschlüsselung (eigene Protokolle realisierbar)
  \+ **lange Transaktionen möglich** (allerdings: Unsicherheitsfaktor WAN)
  \+ direkter Zugriff auf (mehrere) DBS
  \- lange Ladezeiten des Applets (Abhilfe durch JAR: *Java ARchive*)
  \- GUI des DB-Client muss selbst in Java erstellt werden
  \- eingeschränkte Verbindungsmöglichkeiten durch **Java-Sicherheitskonzept** (-> *signed applets* heben teilweise Beschränkungen auf)

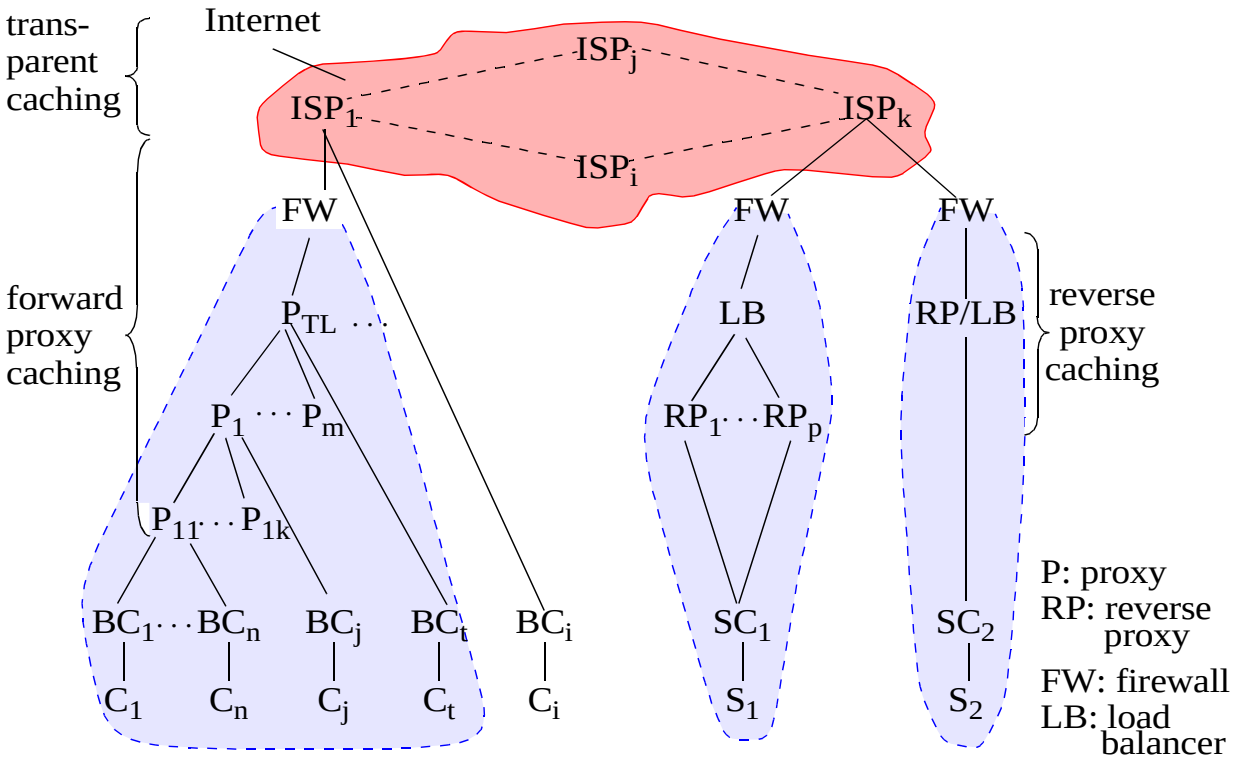
  ➥ **direkte DB-Verbindung ermöglicht datenintensive Anwendungen**

Web als TA-System - Probleme und Lösungsvorschläge



- **Übernahme/Optimierung der Web-Server-Funktionalität**
  - durch erweiterten Applikations-Server
  - mit J2EE- oder .NET-Technologie
- **Antwortzeiten bei Web-Seiten mit dynamischen Inhalten**
  - **Web-Caching** von statischen Dokumenten mit **ID-basiertem Zugriff** in **Proxies im Client-to-Web-Server-Pfad**
  - **Unterstützung des Caching im Web-Server-to-DB-Server-Pfad, da**:
    - immer mehr dynamisch generierte Dokumente
    - personalisierte Web-Seiten, zielgerichtete Werbung, 1-to-1-Marketing, interaktives E-Commerce
  - **Techniken zur Generierung**
    - Scripting-Sprachen (JSP) zur dynamischen Assemblierung von Seiten aus Fragmenten
    - Statische Fragmente können irgendwo gelagert werden. Wo findet man die aktuellen Daten zur Generierung dynamischer Inhalte?
    - Zugriff zur Backend-DB (BE-DB), um aktuelle Daten für eine formularbasierte Anfrage zu generieren
    - Seiten-Zusammenbau in einem Cache möglichst nahe am Web-Client

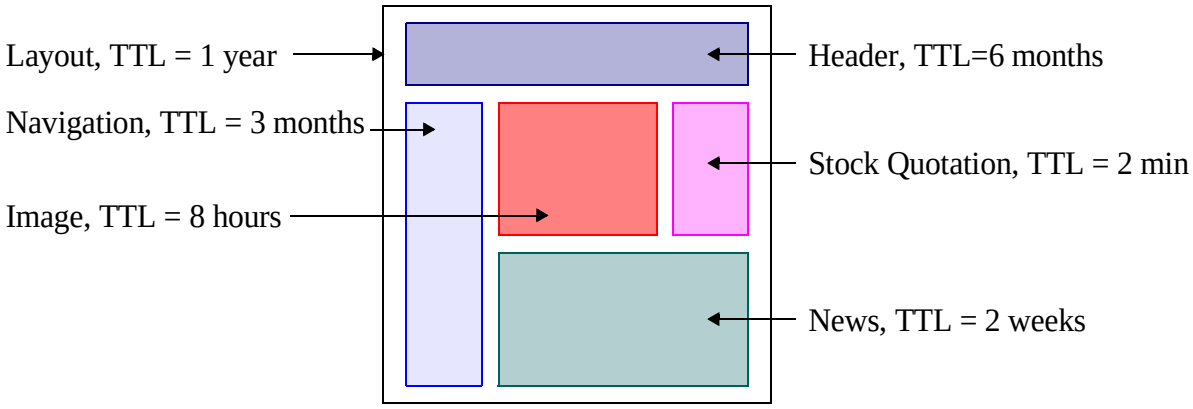
Client-to-Server-Pfad durchs Internet



- ***Browser cache (BC):***
  For all user requests, this cache dedicated to the browser is first searched. If the specific content is located, it is checked to make sure that it is "fresh". Such a private cache is particularly useful if a user scrolls back in his request history or clicks a link to a page previously looked at.
- ***Proxy cache:***
  While working on the same principle, but at a much larger scale, such a cache is shared, performs demand-driven *pull caching*, and serves hundreds or thousands of users in the same way. It can be set up on the firewall or as a stand-alone device. Unless other search paths are specified, a cache miss sends the request to the next proxy cache in the client-to-server path.
- ***Reverse proxy cache:***
  This kind of cache is an intermediary also known as "edge cache", "surrogate cache", or "gateway cache". While not demand-driven, such caches reverse their role as compared to proxy caches, because they are supplied by origin servers with their most recent offerings—a kind of *push caching*. Furthermore, they are not deployed by network administrators to save bandwidth and to reduce user-perceived delays which are characteristic for proxy caches, but they are typically deployed by Web masters themselves, to unburden the origin servers and to make their Web sites more scalable, reliable, and better performing.
- ***Server cache:***
  It keeps generated content and enables reuse without interaction of the origin server. Intermediate results and deliverable Web documents reduce the server load and improve server scalability.

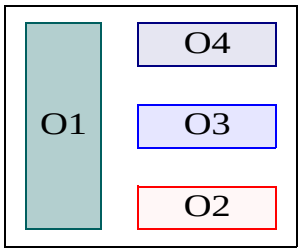
Client-to-Server-Pfad durchs Internet (2)

- **Fragmentierung mit ESI (Edge-Side Includes) mit dem Ziel:**
  häufigeres Refresh ausschließlich der dynamischen Teile von Web-Dokumenten



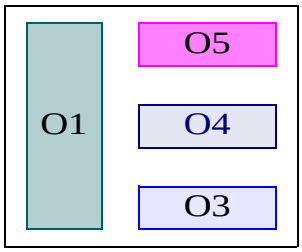
- **Personalisierung und räumliche Verschiebung von Fragmenten**

a) Template1 (t=t1, p=p1)



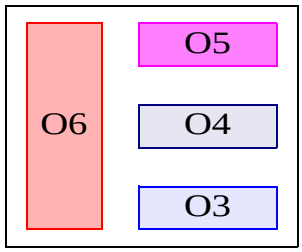

```
template {
<esi:include src=o1>
<esi:include src=o4/>
<esi:include src=o3/>
<esi:include src=o2/>
}
```

b) Template1 (t=t2, p=p1)




```
template {
<esi:include src=o1>
<esi:include src=o5/>
<esi:include src=o4/>
<esi:include src=o3/>
}
```

c) Template1 (t=t2, p=p2)



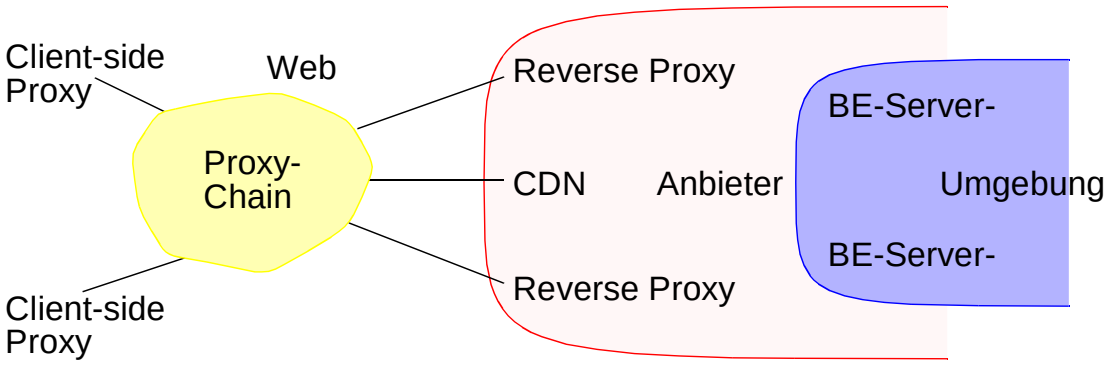

```
template {
<esi:include src=o6>
<esi:include src=o5/>
<esi:include src=o4/>
<esi:include src=o3/>
}
```

- **Vergleich von der Größe von Templates und Objects**

	NY Times	India Times	Slashdot
Template Size	17 KB	15 KB	1.7 KB
Avg. Object Size	3.6 KB	4.8 KB	0.6 KB
Mapping Table Size	1.0 KB	0.8 KB	2.2 KB

Web als TA-System - Probleme und Lösungsvorschläge (2)

- **Verbesserung der Skalierbarkeit**
  - Einsatz von Edge-Servern
    - Dazu gehören Client-side Proxies, Server-side Reverse Proxies oder CDN-Knoten (content distribution network)

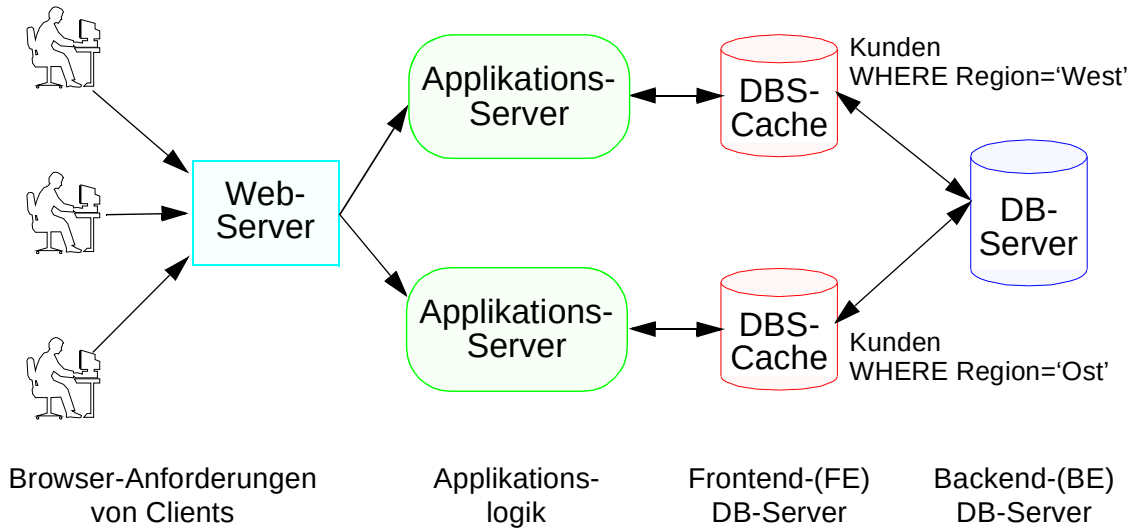


    - Edge-Server zählen zur erweiterten BE-Server-Umgebung; sie sind vertrauenswürdige Server und gehören zur selben Administrationsdomäne wie die BE-Server
    - Verlagerung der Ausführung von Web-Anfragen hin zu Edge-Servern erhöht die Skalierbarkeit der BE-Server, reduziert die Antwortzeit der Clients und vermeidet Überlastung der BE
    - Applikations-Server und CDN-Lösungen wie EdgeSuite (Akamai) und WebSphere Edge Server (IBM) erlauben das Verlagern von Anwendungskomponenten (Servlets, JSPs, Enterprise Beans, Page Assembly), die gewöhnlich im BE ablaufen
    - Bereitstellung von dynamischen Inhalten erfolgt, wenn möglich, in Edge-Servern (Caching!). Sonst, bei fehlenden Daten oder bei veralteten Daten, die den Konsistenzansprüchen der Clients nicht genügen, werden die Anforderungen zum BE weitergeleitet

➥ **Neue Herausforderung: Caching von strukturierten DB-Daten und Auswertung deklarativer Anfragen in Edge-Servern**

Web als TA-System - Probleme und Lösungsvorschläge (3)

- **Caching von DB-Daten**[2]
  - Caching ist die **bewährte Technik,** um Skalierbarkeit und Leistungsverhalten in großen, verteilten Systemen zu verbessern!
  - Vorgenerierung von Dokumenten mit dynamischen Inhalten (Angabe der Generierungszeit – Inkaufnahme von leicht veralterter Information)
  - **DB-Caching-Verfahren** mit entfernter deklarativer Anfrageauswertung (im Gegensatz zu Web-Caching mit ID-basiertem Zugriff)
    - Statisches Caching (Replikation) von ganzen Tabellen oder materialisierten Sichten (spezifiziert über einfache Prädikate für Project/Select-Anfragen (PS), auch query result caching genannt)
    - **On-Demand Caching** (Constraint-basiertes parametrisiertes Caching) verwendet BE-Metadaten und „Hinweise“ der TA, um Daten dynamisch in den Cache zu füllen oder zu ersetzen; hoher Grad an Adaptivität erwünscht!



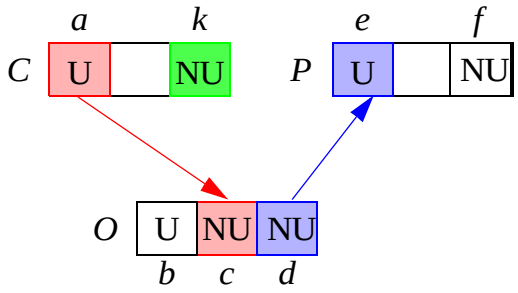
  - **DB-Caching** „in der Nähe“ des Applikations-Servers
    - Beschleunigung des lesenden DB-Zugriffs
    - (bislang noch) Weiterleitung von Änderungsanweisungen zum BE
    - Konsistenzprobleme

2. “The three most important parts of any Internet application are caching, caching, and, of course, caching …”—Larry Ellison, Oracle Chairman & CEO.

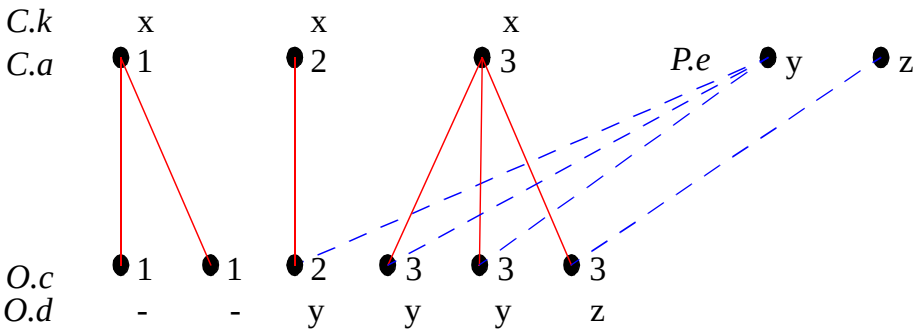
Web als TA-System - Probleme und Lösungsvorschläge (4)

- **Caching von DB-Daten (Forts.)**
  - Wünschenswert: DB-Caching mit Abwicklung von **PSJ-Anfragen (π, σ, ⋈)**
  - Listen von Spaltenwerten definieren, welche Gleichheitsprädikate im Cache ausgewertet werden sollen. Diese Werte besitzen die Eigenschaft „wertvollständig“ (value complete), Spezialfall ist „spaltenvollständig“ (column complete), d. h., wenn ein solcher Wert im Cache gefunden wird, garantiert der Cache-Mgr, dass alle Sätze (Zeilen) mit diesem Wert im Cache sind. Im Gegensatz zu (NU) sind UNIQUE-Spalten (U) somit immer spaltenvollständig
  - „Verknüpfung“ von Tabellen durch Referential Cache Constraints (RCCs), um **PSJ-Anfragen** auswerten zu können. Es muss **Prädikatenvollständigkeit** garantiert werden!
    - Wichtigster Fall: Primär-/Fremdschlüsselbeziehungen (Owner-Member)
    - Wenn ein Wert von C.a mit RCC C.a→O.c im Cache ist, garantiert der Cache-Mgr, dass **alle Sätze** mit dem gleichen Wert in O.c im Cache sind
    - Sog. **Cache Groups** unterstützen Verbundoperationen im Cache
  - Beispiel

Cache Group COP



C.k = x sei als wertvollständig definiert; Ablauf des Ladevorgangs C.a → O.c und O.d → P.e



Auswertbare Prädikate: C.k=x and C.a=O.c and O.d=P.e and ...
C.a=3 and C.a=O.c and O.d=P.e and ...
...

Web als TA-System - Probleme und Lösungsvorschläge (5)[3]

- **DB-Anfragen über mehrere HTML-Seiten, Folgeanfragen?**
  - Cookies zur Identifikation des Browsers
  - In HTML-Seiten einkodierte Session-IDs
  - Server-seitiger Anwendungsprozess mit langer Transaktion (erfordert Wissen über Anwendungsabfolge)
- **Wie lange bleibt eine Transaktion offen?**
  - Timeout-Verfahren (was ist eine geeignete Wartezeit?)
  - Server-seitiger Agent (Proxy-Funktion) als DB-Client nimmt Anfragen aus dem Web entgegen und verwaltet offene Transaktionen. Nach Timeout Zurücksetzen der entsprechenden TAs
- **Geschäftstransaktionen (business transactions, BTs)**
  - ACID-TAs als Bausteine
  - „hierarchisch zusammengebaute“ Aggregationen von ACID-TAs in heterogenen und autonomen Umgebungen
  - Agreement-Protokolle statt globales ACID
  - „Bemühenszusage“ bzgl. Rollback/Recovery

  ➥ Atomare Schritte für Kernaufgaben (z. B. Bestellung), nicht-atomare Schritte in Hilfsprozessen (Führung von Reisekosten-Konten)
- **TAs in einer Web-Umgebung (z.B. e-Commerce)**
  - sind komplex
  - schließen mehrere (nicht-vertrauenswürdige) Teilnehmer ein
  - überspannen Organisationsgrenzen
  - sind langlebig

---

3. Loeser, H.: Datenbankanbindung an das WWW – Techniken, Tools und Trends, in: Tagungsband der GI-Fachtagung ’Datenbanksysteme in Büro, Technik und Wissenschaft’ (BTW’97), K. R. Dittrich, A. Geppert (Hrsg.), Informatik aktuell, Ulm, März 1997, Springer-Verlag, S. 83 - 99.

Benchmarks[4] – Nachbildung von TA-Lasten

- **Bei interaktiven TA ist die Antwortzeit eine kritische Größe. Stapel-TA können dagegen Stunden oder Tage laufen und enorme Ressourcen verbrauchen (CPU, Speicher)**
- **Leistungsbestimmung eines TA-, DW- oder DSS-Systems ist sehr schwierig**
  - Komplexität des Systems
  - Leistungsverhalten verschiedenartiger Lasten oft **sehr verschieden**!

➥ Wie kann ein System bewertet und in seiner Leistung mit anderen verglichen werden?

- **Gesucht:** Test zur Evaluation der wesentlichen Leistungsmerkmale
  - Anforderungen spezifiziert als anwendungsbezogene Funktionalität (TA-Typen)
  - Aussagen über gesamte Systemkosten
  - Leistungsverhalten bei Wachstum der TA-Last und/oder der Datenvolumina (Skalierbarkeit)
  - objektive Leistungsmaße wie Durchsatz und Antwortzeit zur einfachen Vergleichbarkeit (bei Systemwachstum oder konkurrierenden Systemen)
- **Typische Leistungsmaße/-werte interaktiver TA**

Leistung/TA	einfach	mittel	komplex
Instr./TA	100 K	1 M	100 M
Platten-E/A/TA	1	10	1000
Lokale Nachr. (B)	10 (5KB)	100 (50KB)	1000 (1MB)
Entfernte Nachr. (B)	2 (300B)	2 (4KB)	100 (1MB)
Kosten/TA/sec.	< 10$/tps	< 10$/tpm	<100$/QphH
Spitzen-TPS/Knoten	> 1000	> 100	> 1

4. Benchmark: Vergleichspunkt, Bezugswert, Maßstab

Benchmarks für TA-Systeme[5]

- **Im Laufe der Zeit wurden unterstützende Verfahren entwickelt**
  - Es wurden verschiedene **Nutzungsmuster** von Anwendungen erkannt, die als Benchmark nachgebildet wurden
  - Probleme früher Benchmarks
    - SUT (system under test) lieferte nichtssagende Leistungszahlen, da Netz- und Benutzerinteraktion vernachlässigt wurden
    - Sie waren unvollständig spezifiziert und wurden nicht überwacht (durch “Notar”)

    ➥ Marketing-Aussagen waren oft unglaubwürdig!

- **Ziele**
  - Definition von aussagefähigen Benchmarks, um typische TA-Lasten bestimmter Anwendungsbereiche zu repräsentieren
  - Spezifikation eines durch gängigen Prozesses zur
    - Anpassung, Ergänzung und Überwachung von Benchmarks
    - Gewährleistung der Vergleichbarkeit der Messungen (und damit der Systeme)

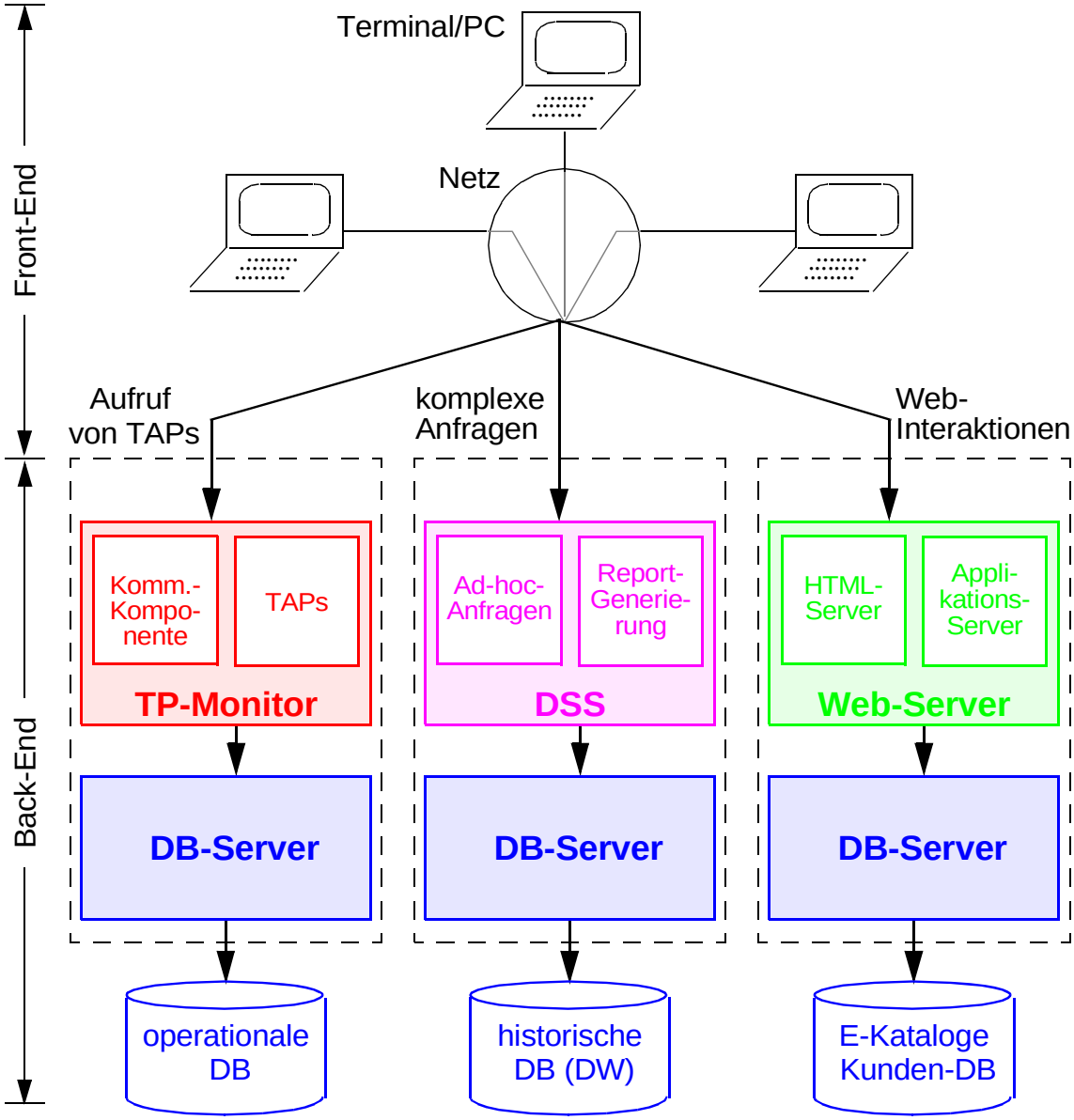
- **Standardisierung durch TPC (Transaction Processing Performance Council)**
  - Konsortium aus z. Zt. 41 Organisationen (HP, IBM, Microsoft, Sun, ...)
  - Formelles Verfahren zur Standardisierung von Benchmarks: TPC-A (1989), TPC-B (1990), TPC-C (1992), TPC-D (1994), ...
  - Unabhängige Instanzen zur Überwachung

➥ **“Good benchmarks are like good laws”**

---

5. Gray (ed.): The Benchmark Handbook for Database and Transaction Processing Systems. Morgan Kaufmann 1991

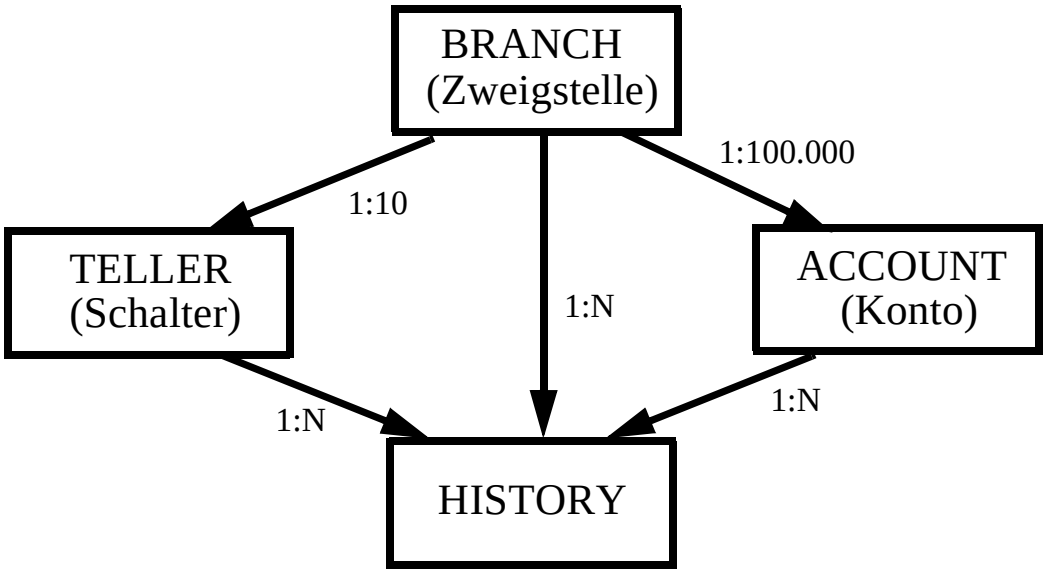
Benchmarks für TA-Systeme – Grobaufbau der Testumgebungen



- **Optionen für SUT (system under test):** Messung der Aktionen
  - im Gesamtsystem (End-to-End)
  - nur im Server (Back-End)
- **Struktur des Back-End**
  - zentralisierte Anordnung oder
  - verteilte Komponenten (Client/Server-Architektur)
  - m Anwendungs-, n DB-Server-Instanzen

Die Standard-Transaktion: Kontenbuchung („DebitCredit“)[6]

- **TPC-A: erste Standardisierung durch TPC**
  - basierend auf dem IBM-Benchmark TP1 (~1985), der die Systemleistung bei Abwicklung von ATM-Transaktionen (Automated Teller Machine) ohne Netz- und Interaktionskomponente (Denkzeit) erfasste
  - Vervollständigung des Definition (Datenstrukturen, Skalierung, ACID)
- **Vier Satztypen:**



- **Mengengerüst abhängig vom Durchsatzziel (in TA pro Sekunde (tps))**
  - pro tps:

    1 BRANCH-Satz
    10 TELLER-Sätze
    100.000 ACCOUNT-Sätze
    HISTORY für 90 Tage (90*8*60*60 = 2.592.000 Sätze)
  - daneben 10 Terminals pro tps

---

6. Anon et al.: *A Measure of Transaction Processing Power*, Datamation, April 1985; J. Gray und 24 Autoren gaben mit ihrem Artikel zu DebitCredit-Transaktionen den Anstoss zur Gründung des TPC

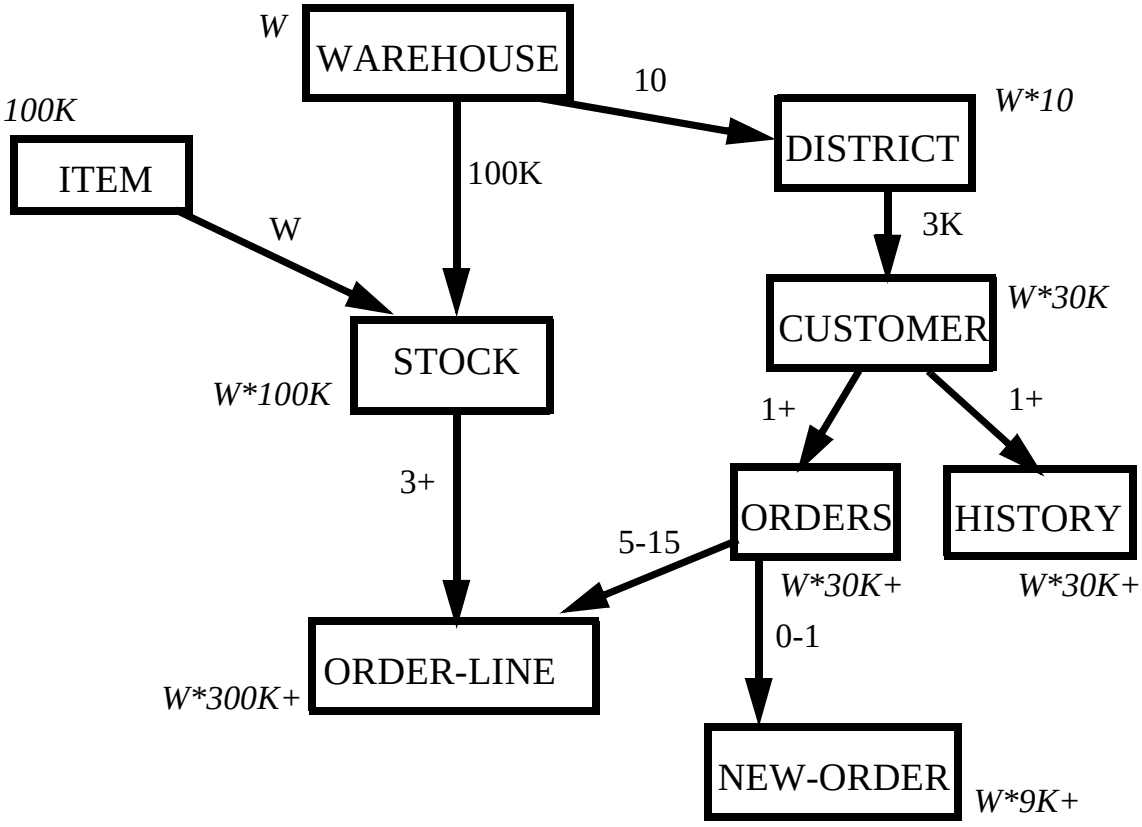
DebitCredit: Transaktionsprogramm

```
Read message from Terminal (acctno, branchno, tellerno, delta);
 BEGIN_WORK    { Beginn der Transaktion }

        UPDATE  ACCOUNT
            SET  balance  =  balance  +  :delta
            WHERE  acct_no = :acctno

        SELECT  balance  INTO :abalance
            FROM  ACCOUNT
            WHERE  acct_no = :acctno;
        UPDATE  TELLER
            SET  balance  =  balance  +  :delta
            WHERE  teller_no = :tellerno

        UPDATE  BRANCH
            SET  balance  =  balance  +  :delta
            WHERE  branch_no = :branchno

        INSERT  INTO  HISTORY  (Tid, Bid, Aid, delta, time)
            VALUES (:tellerno, :branchno, :acctno, :delta, CURRENT);

COMMIT_WORK  ;  { Ende der Transaktion }
 Write message to Terminal  (abalance, ...);
```

- **Benchmark-Forderungen:**
  - 15% der Transaktionen betreffen Konto einer anderen Zweigstelle
  - 90% der TA sollen Antwortzeit von höchstens 2 Sekunden haben
- **Bestimmung der Kosten $/tps:**
  - **Systemkosten** umfassen HW-, SW- und Kommunikations-Komponenten, Terminals, Backup-Speicher sowie Wartung für 5 Jahre
  - Systemkosten ($) / Durchsatz pro Sekunde (tps)
- **TPC-A:** Unterscheidung von geographisch und lokal verteilten Systemen
- **TPC-B:**
  - keine Berücksichtigung des Netzes und der Terminals (Back-End)
  - Kritik: tps künstlich hoch, $/tps künstlich niedrig

Benchmark TPC-C

- **Modellierung/Abwicklung von Aktivitäten im Großhandel**
  - repräsentativ für komplexe OLTP-Anwendungsumgebungen
  - Verwaltung, Verkauf, Verteilung von Produkten oder Dienstleistungen
  - Unternehmen besitzt geographisch verteilte Geschäftsdistrikte und zugeordnete Filialen
  - realistischere Transaktionslast aus mehreren Transaktionstypen unterschiedlicher Komplexität und Änderungshäufigkeit
- **Anwendung: Bestellverwaltung im Großhandel**
  - Betrieb umfasst W Warenhäuser, pro Warenhaus 10 Distrikte, pro Distrikt 3000 Kunden
  - 100.000 Artikel; pro Warenhaus wird Anzahl vorhandener Artikel geführt
  - 1% aller Bestellungen werden von nicht-lokalem Warenhaus angefordert
- **9 Satztypen**



TPC-C (2)

- **Haupttransaktionstyp: NEW-ORDER**

BEGIN_WORK { Beginn der Transaktion }

SELECT ... FROM **CUSTOMER**
WHERE c_w_id = *:w_no* AND c_d_id = *:d_no* AND c_id = *:cust_no*

SELECT ... FROM **WAREHOUSE**
WHERE w_id = *:w_no*

SELECT ... FROM **DISTRICT** (* -> *next_o_id* *)
WHERE d_w_id = *:w_no* AND d_id = *:d_no*

UPDATE DISTRICT
SET d_next_o_id := *:next_o_id* + 1
WHERE d_w_id = *:w_no* AND d_id = *:d_no*

INSERT INTO NEW_ORDER ...

INSERT INTO ORDERS ...

pro Artikel (im Mittel 10) werden folgende Anweisungen ausgeführt:

SELECT ... FROM **ITEM** WHERE ...

SELECT ... FROM **STOCK** WHERE ...

UPDATE STOCK ...

INSERT INTO ORDER-LINE ...

COMMIT_WORK { Ende der Transaktion }

- **im Mittel 48 SQL-Anweisungen**
  (BOT, 23 SELECT, 11 UPDATE, 12 INSERT, EOT)
- **Durchsatzangabe** für New-Order-Transaktionen in tpmC
  (Transaktionen pro Minute)

TPC-C (3)

- **Transaktionstypen:**
  - *New-Order:* Artikelbestellung (Read-Write)
  - *Payment:* Bezahlung einer Bestellung (Read-Write)
  - *Order-Status*: Status der letzten Bestellung eines Kunden ausgeben (Read-Only)
  - *Delivery:* Verarbeitung von 10 Bestellungen (Read-Write)
  - *Stock-Level:* Anzahl von verkauften Artikeln bestimmen, deren Bestand unter bestimmtem Grenzwert liegt (Read-Only)

- **Festlegung des Transaktionsmixes**
  - Payment-TA müssen mindestens 43% der Last ausmachen
  - Order-Status, Delivery und Stock-Level je mindestens 4%
  - New-Order-Anteil ist variabel; solange es die Antwortzeitrestriktionen zulassen, werden bei einer Messung diese Transaktionen generiert

- **Antwortzeitrestriktionen**
  - 90% unter 5 Sek. (New-Order, Payment, Order-Status, Delivery)
  - Stock-Level: 90% unter 20 Sek.

- **Was bedeutet tpmC?**
  - Es wird **nur der Durchsatz** von New-Order-Transaktionen pro Minute gemessen, wobei das System parallel die restlichen 4 Transaktionstypen unter Beachtung der Restriktionen des Transaktionsmixes und der Antwortzeit ausführt
  - SUT-Messung: End-to-End

- **Was bedeutet $/tpmC?**

  Systemkosten ($) / Durchsatz pro Minute (tpmC)

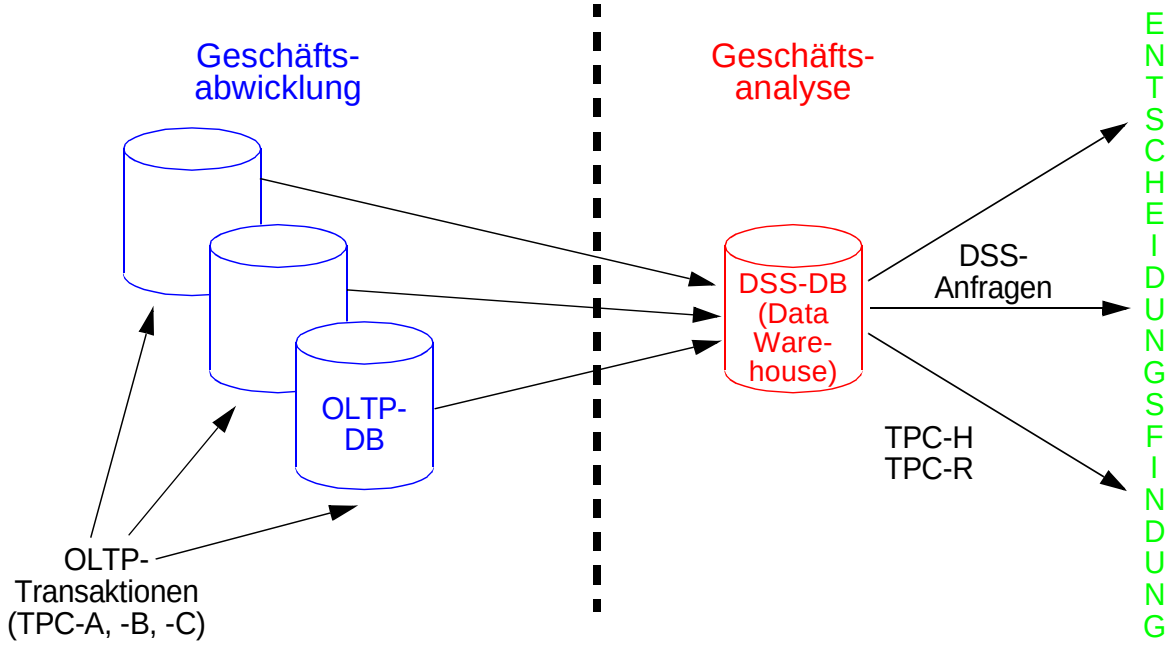
TPC-H

- **Benchmarks zur Auswertung komplexer Anfragen in großen DBs**
  - ursprünglich als TPC-D eingeführt (gültig bis 4/99)
  - bilden durch Anfragen typische Aktivitäten im Großhandel nach
  - zielen auf die Datenanalyse ab
    - Berechnung von Trends
    - Unterstützung der Entscheidungsfindung
  - übernehmen Schema, Skalierungsfaktoren und Anfragen vom TPC-D
  - erweitern sie auf 22 Anfragetypen und 2 Aktualisierungsfunktionen (DB refresh)

- **TPC-H (ad-Hoc, decision support)**
  nutzt kein Vorwissen aus (lange Ausführungszeiten für Anfragen)

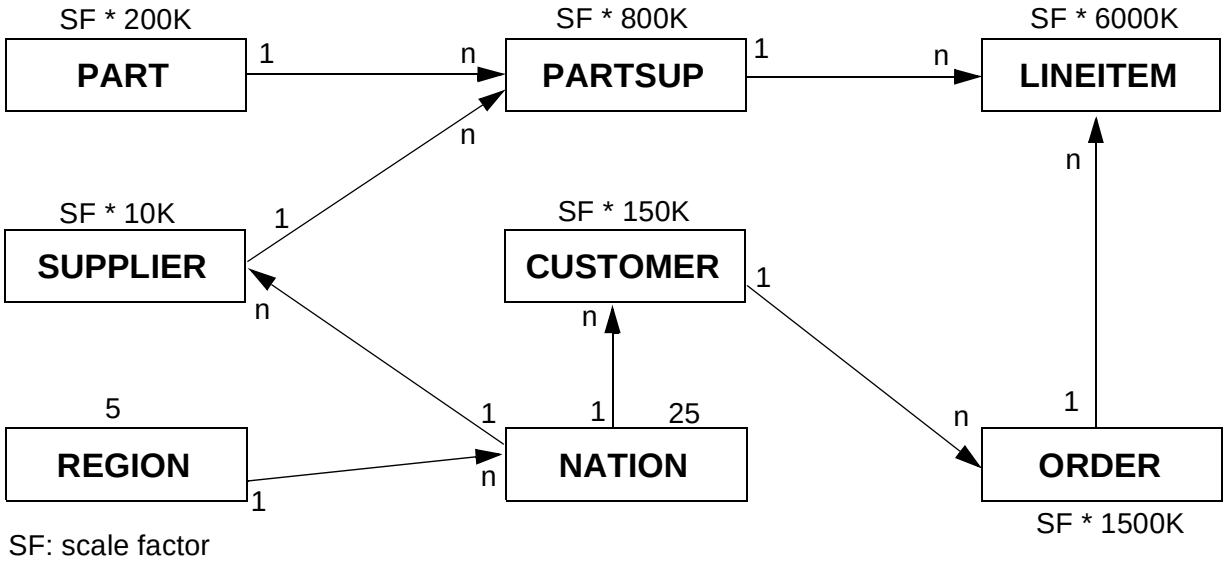
- **TPC-R (business Reporting, decision support)** (wird nicht mehr eingesetzt)
  unterscheidet sich vom TPC-H durch Ausnutzung von Vorwissen:
  DBS kann speziell auf Standard-Anfragen hin optimiert werden!

- **Einsatzumgebung**



TPC-H (2)

- **Schema:**



- Anfragebeispiel Q9**: Product Type Profit Measure Query**

  The query finds, for each nation and each year, the profit for all parts ordered in that year which contain a specified substring in their names and which were filled by a supplier in that nation. The profit is defined as the sum of (L_EXTENDEDPRICE*(1-L_DISCOUNT)) - (PS_SUPPLYCOST * L_QUANTITY)] for all lineitems describing parts in the specified line. The query lists the nations in ascending alphabetical order and, for each nation, the year and profit in descending order by year (most recent first).

- **Verfügbarkeit der Datenbank:** 24*7*52 h (mit Wartungspausen)
- **Maßeinheiten:**

  - Composite Query-per-Hour Metric:
    QphH@Size = sqrt(Power @ Size * Throughput @ Size)
    - query processing power at the chosen DB size (Power @ Size) within a single query stream
    - throughput at the chosen DB size (Throughput @ Size) in multi-user environment
  - Price/Performance Metric:
    Price-per-QphH@Size = $/QphH @ Size

TPC-H (3)

- Anfrage Q2: **Minimum Cost Supplier Query**

  The query finds, in a given region, for each part of a certain type and size, the supplier who can supply it at minimum cost. If several suppliers in that region offer the desired part type and size at the same (minimum) cost, the query lists the parts from suppliers with the 100 highest account balances. For each supplier, the query lists the supplier's account balance, name and nation, the part's number and manufacturer, the supplier's address, phone number and comment information.

- **SQL-Umsetzung von Q2:**

Note: return only the first 100 selected rows

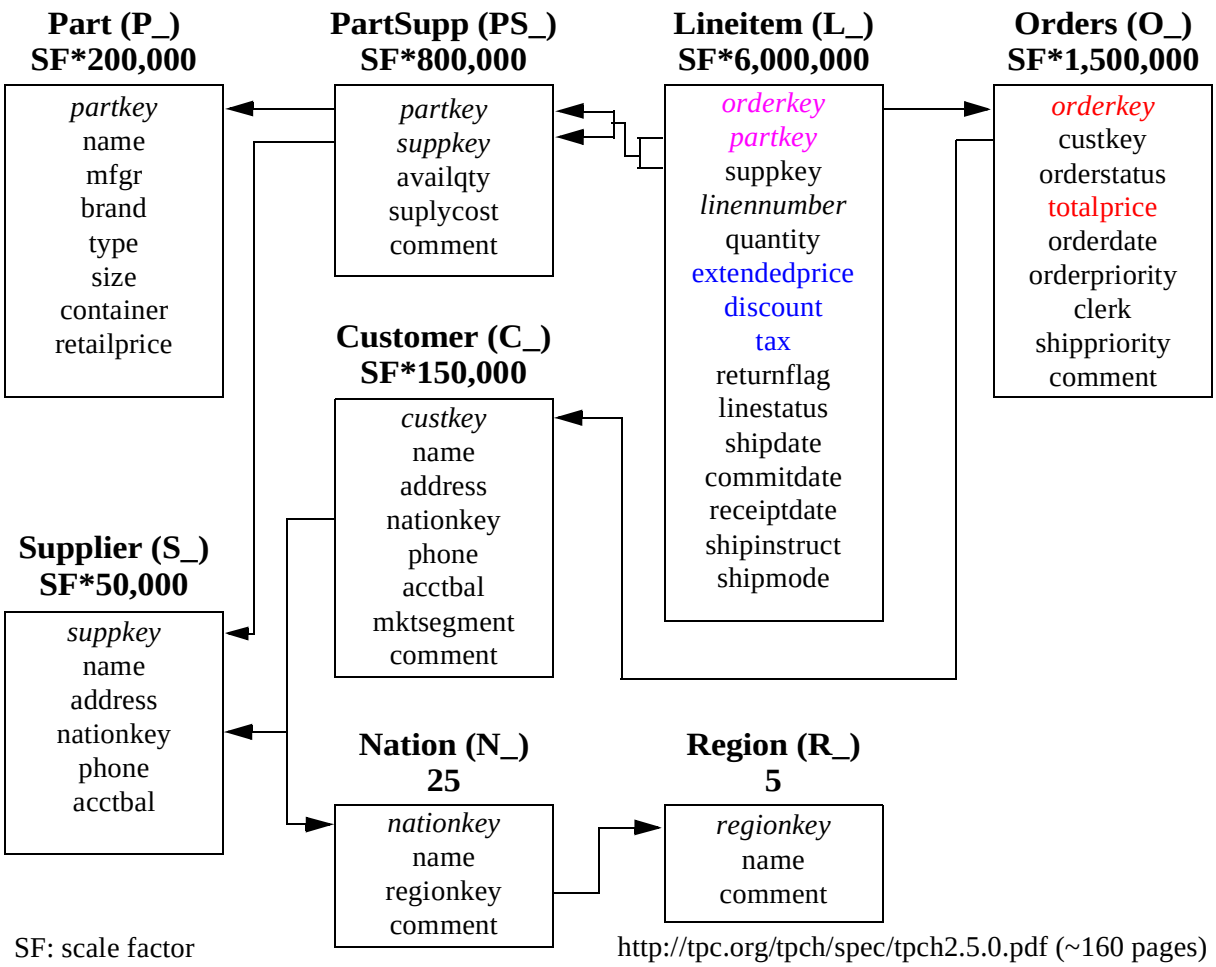
```
SELECT  S_ACCTBAL, S_NAME, N_NAME, P_PARTKEY, P_MFGR, S_ADDRESS,
        S_PHONE, S_COMMENT
        FROM    PART, SUPPLIER, PARTSUPP, NATION, REGION
        WHERE   P_PARTKEY = PS_PARTKEY
                AND S_SUPPKEY = PS_SUPPKEY
                AND P_SIZE = [size]
                AND P_TYPE LIKE '%[type]'
                AND S_NATIONKEY = N_NATIONKEY
                AND N_REGIONKEY = R_REGIONKEY
                AND R_NAME = '[region]'
                AND PS_SUPPLYCOST =
                (SELECT MIN(PS_SUPPLYCOST)
                        FROM    PARTSUPP, SUPPLIER, NATION, REGION
                        WHERE   P_PARTKEY = PS_PARTKEY
                           AND S_SUPPKEY = PS_SUPPKEY
                           AND S_NATIONKEY = N_NATIONKEY
                           AND N_REGIONKEY = R_REGIONKEY
                           AND R_NAME = '[region]')
        ORDER BY S_ACCTBAL DESC, N_NAME, S_NAME, P_PARTKEY;
```

TPC-H Benchmark

- **Summary**

The TPC Benchmark™H (**TPC-H**) is a *decision support benchmark* and consists of >20 business-oriented queries and concurrent data modifications. It represents decision support environments where users *run ad-hoc queries* against a database system. **TPC-R** (Decison Support, Business Reporting) is similar to TPC-H, but it allows additional optimizations based on advance knowledge of the queries. In this environment, *pre-knowledge of the queries is assumed and may be used for optimization* to run these standard queries very rapidly.

The *performance metric reported by TPC-R* is called the TPC-R Composite Query-per-Hour Performance Metric (*QphR@Size*), and reflects multiple aspects of the capability of the system to process queries. These aspects include the selected database size against which the queries are executed, the query processing power when queries are submitted by a single stream, and the query throughput when queries are submitted by multiple concurrent users. The *TPC-R Price/Performance metric* is expressed as *$/QphR@Size*.



SF: scale factor

http://tpc.org/tpch/spec/tpch2.5.0.pdf (~160 pages)

Materialized Views

- **Use of materialized views**
  - they are a powerful tool for **improving the performance** of complex queries
  - **efficient online maintenance** is needed
    - but only known for views defined by SQL queries composed of only Select, Project, Join, Group-by operators (SPJG views)
    - and limited to aggregation functions such as `sum` or `count` that can be computed incrementally
  - example uses a 10GB TPC-R database

Example Query Q against the TPC-R database contains two levels of aggregation and attempts to **find important parts**; a part is important if it contributed more than 90 % to the value of an order. To save space, `np` (net price) is used as a short-hand for (`l_extendedprice * (1 + l_tax) * (1 - l_discount)`.) The query cannot be well supported using only SPJG because of the two-level aggregation but allowing views to be stacked (views on views) changes the situation.

Q:
```
select l_partkey, count(*) ocnt, sum(sp) oval
from orders as o,
  (select sum(np) sp, l_orderkey, l_partkey
  from lineitem
  group by l_orderkey, l_partkey) as l
where o_orderkey = l_orderkey
  and o_totalprice * 0.9 < sp
group by l_partkey
```

Without any views: **9877 secs**

V_1:
```
select sum(np) sp, l_orderkey, l_partkey
from lineitem
group by l_orderkey, l_partkey
```

Using materialized view V_1: **349 secs**

V_2:
```
select l_partkey, count(*) ocnt, sum(sp) oval
from orders as o, V1 as l
where o_orderkey = l_orderkey
  and o_totalprice * 0.9 < sp
group by l_partkey
```

Using the concept of *stacked views (S-SPJG views)*, we can create a materialized view V_2 that combines orders and V_1.
The query reduces to a simple scan of V_2: **4.5 secs**

TPC-E

- **TPC Benchmark™ E is a new OLTP workload developed by the TPC**

The TPC-E benchmark simulates the OLTP workload of a **brokerage firm**. The focus of the benchmark is the central database that executes transactions related to the firm's customer accounts. Although the underlying business model of TPC-E is a brokerage firm, the database schema, data population, transactions, and implementation rules have been designed to be broadly **representative of modern OLTP systems**.

The TPC-E benchmark uses a database to model a brokerage firm with customers who generate transactions related to **trades, account inquiries, and market research**. The brokerage firm in turn interacts with financial markets to execute orders on behalf of the customers and updates relevant account information.

The benchmark is "scalable," meaning that the number of customers defined for the brokerage firm can be varied to represent the workloads of different-size businesses. The benchmark defines the required mix of transactions the benchmark must maintain. The TPC-E metric is given in transactions per second (tps). It specifically refers to the **number of Trade-Result transactions** the server can sustain over a period of time.

TPC-App

- **TPC-App is an Application Server and web services benchmark.**

The workload is performed in a managed environment that simulates the activities of a **business-to-business transactional application server** operating in a 24x7 environment. TPC-App showcases the performance capabilities of application server systems. The workload exercises commercially available **application server products, messaging products, and databases** associated with such environments, which are characterized by:

- Multiple on-line business sessions
- Commercially available application environment
- Use of XML documents and SOAP for data exchange
- Business to business application logic
- Distributed transaction management
- Reliable and durable messaging
- Dynamic web service response generation with database access and update
- Simultaneous execution of multiple transaction types that span a breadth of business functions.
- Databases consisting of many tables with a wide variety of sizes, attributes, and relationships
- Transaction integrity (ACID properties)

TPC-Benchmarks: Kennzahlen[7]

- **Viele Benchmark-Messungen durch 33 Hersteller**
  - TPC-A (TPC-B): >300 (>130) publizierte Ergebnisse von 115 (73) verschiedenen Systemen
  - zwei “goldene Zahlen”: tps und $/tps
  - TPC-A und TPC-B wurden ab 6/1995 nicht mehr verwendet

- **Leistungswerte für TPC-A**
  - ca. 100 - 200 KInstr. / TA (anfangs bis zu 1 Mill. Instr. pro TA)
  - 2 E/A-Vorgänge pro TA (anfangs bis zu 20)
  - 1990: 33 tpsA zu 25,500 $/tpsA
  - 1995: 3692 tpsA zu 4,873 $/tpsA

    ➥ 111 und 5 als Verbesserungsfaktoren

- **Leistungswerte für TPC-B**
  - ca. 75 KInstr. / TA
  - 1991: 103 tpsB zu 4,167 $/tpsB
  - 1994: 2,025 tpsB zu 254 $/tpsB

    ➥ 19 und 16 als Verbesserungsfaktoren

- **Warum hatten diese TPC-Benchmarks solche Erfolge?**
  - erste Benchmark-Messungen ohne spezielle Optimierung
  - reale Performance-Steigerungen durch HW- und SW-Produkte
  - Systemverbesserungen, um durch Benchmark aufgedeckte Performance-Schwächen zu eliminieren
  - **effektive Nutzung des “Benchmark-Game”:** Hersteller lernten voneinander, wie der Benchmark am besten abzuwickeln ist

7. http://www.tpc.org/

TPC-Benchmarks: Kennzahlen (2)

- **Leistungswerte für TPC-C bei Performance**
  - 1992: 54 tpmC zu 188,562 $/tpmC
  - 1998: 52,871 tpmC zu 135 $/tpmC
  - 2000: 505,302 tpmC zu 21 $/tpmC
  - 2001: 709,220 tpmC zu 14.96 $/tpmC (TPC-C V5)
  - 2007: 4,092,799 tpmC zu 2.93 $/tpmC (auf Oracle 10g R2)
  - 2009: 6,085,166 tpmC zu 2.81 $/tpmC (auf IBM DB2 9.5)

  ➥ **112,688** und **67,104** als Verbesserungsfaktoren

- **Leistungswerte für TPC-C (V5) bei Price/Performance**
  - 2002: 16,756 tpmC zu 2.78 $/tpmC
  - 2003: 82,226 tpmC zu 2.76 $/tpmC
  - 2004: 26,410 tpmC zu 1.53 $/tpmC
  - 2007: 82,774 tpmC zu 0.94 $/tpmC (MS SQL Server 2005)
  - 2009: 104,492 tpmC zu 0.60 $/tpmC
    (Oracle Database 11g Standard Ed. x 64)

- **Leistungswerte für TPC-D (bei 100 GB)** (wird nicht mehr eingesetzt)
  - 1995: 84 QphD und 52,170 $/QphD
  - 1998: 1,205 QphD und 1,877 $/QphD

  ➥ **14** und **28** als Verbesserungsfaktoren (bis 1998)

  - ab 1999: TPC-H und TPC-R;
    sie sind gegenüber den TPC-D von 17 auf 22 Anfragen erweitert

TPC-Benchmarks: Kennzahlen (3)

- **Leistungswerte für TPC-H[8] bei Performance**
  - 2002: 5,578 QphH und 358 $/QphH bei 100 GB
  - 2003: 12,216 QphH und 71 $QphH bei 100 GB (DB2 UDB 8.1)
  - 2007: 19,323 QphH und 10.67 $QphH bei 100 GB (MS SQL Server 2005)
  - 2009: 209,298 QphH und 1.25 $QphH bei 100 GB (EXASQL EXASolution 2.0)

  - 2002: 25,805 QphH und 203 $/QphH bei 1,000 GB
  - 2005: 53,451 QphH und 33 $/QphH bei 1,000 GB (DB2 UDB 8.2)
  - 2007: 40,411 QphH und 18.67 $/QphH bei 1,000 GB (Oracle 10g)
  - 2009: 1,018,321 QphH und 1.18 $QphH bei 1,000 GB (EXASQL EXASolution 2.1)

  - 2002: 27,094 QphH und 240 $/QphH bei 3,000 GB (V1)
  - 2005: 59,435 QphH und 114 $/QphH bei 3,000 GB (V1) (Oracle 10g)
  - 2007: 114,713 QphH und 36.68 $/QphH bei 3,000 GB (V1) (Oracle 10g)
  - 2009: 1,608,920 QphH und 1.36 $QphH bei 3,000 GB (EXASQL EXASolution 2.1)

  - 2002: 81,501 QphH und 243 $/QphH bei 10,000 GB (V2)
  - 2005: 86,282 QphH und 161 $/QphH bei 10,000 GB (V2) (Oracle 10g)
  - 2007: 180,108 QphH und 47 $/QphH bei 10,000 GB (V2) (DB2 UDB 8.2)
  - 2009: 343,551 QphH und 32.89 $QphH bei 10,000 GB (IBM DB2 Warehouse 9.5)

- **Leistungswerte für TPC-W bei Price/Performance**
  - 2001: 6,622 WIPS und 25.70 $/WIPS
  - 2002: 7,783 WIPS und 24.50 $/WIPS (SQL Server 2000)
    TPC-W wird seit 4/28/2005 nicht mehr eingesetzt
- **Bemerkung:**
  Bei „Transaktionen“ (besser mouse clicks) übers Internet wurde neuerdings scherzhafterweise das Kostenmaß „µ$/tps“ geprägt.

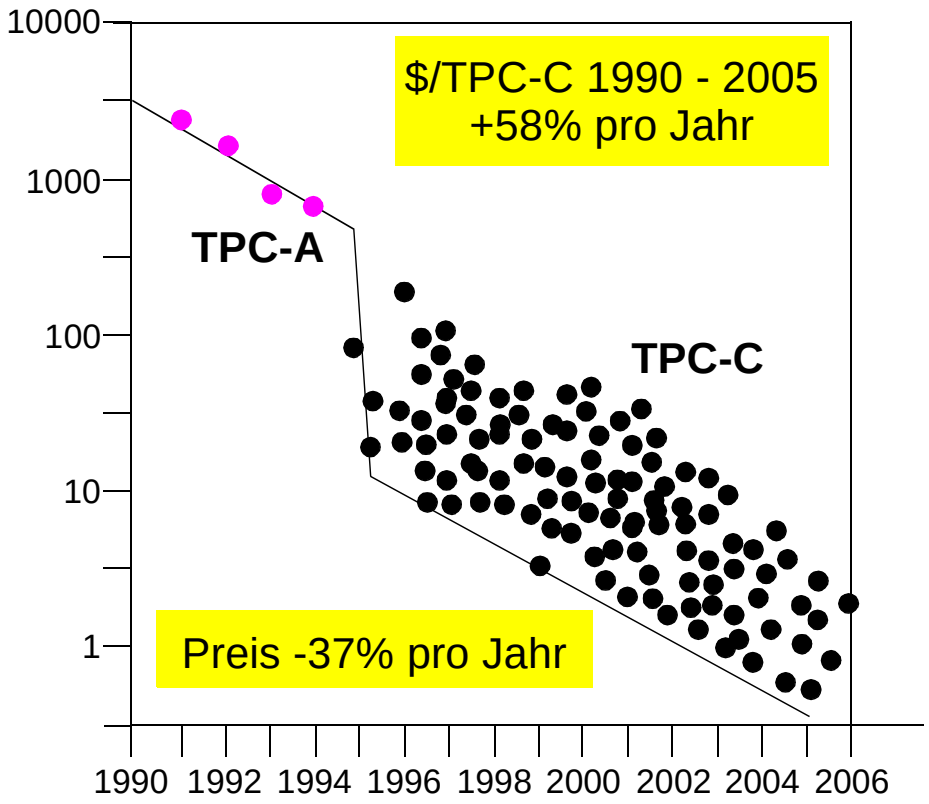
---

8. Note: The TPC believes that comparisons of TPC-H results measured against different database sizes are misleading and discourages such comparisons. The TPC-H results shown below are grouped by database size to emphasize that only results within each group are comparable.

Transaktionsverarbeitung 20 Jahre später

- **Jim Gray et al. definierten 1985 drei Benchmarks**
  - **DebitCredit**, ein Test für DBS und Transaktionssystem
  - **Sort**, ein Test für Betriebs- und E/A-System
  - **Copy**, ein Test für das Dateisystem

- **DebitCredit**[9]
  - ging über in TPC-A und dann in TPC-C
  - 100 TPS --> 100,000 TPS
  - bei TPC-C: Preis/Leistungstrend 1990 – 2005
    - Verbesserung 58%/Jahr
    - Preisverfall 37%/Jahr



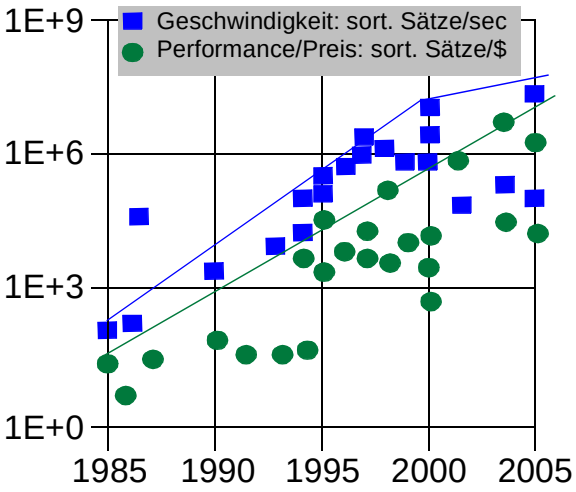
9. http://www.tpc.org

Transaktionsverarbeitung 20 Jahre später

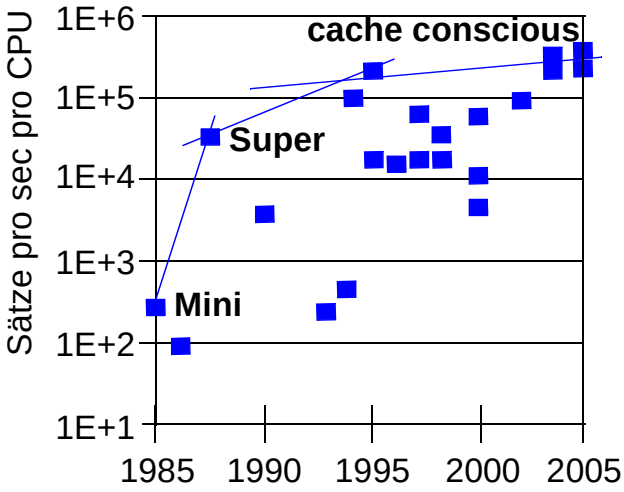
- **Benchmarks für Sortieren**
  - **Sortiere 1M Sätze** (nun im Bruchteil einer Sekunde)
  - **PennySort** (sortiere soviel du kannst für einen Penny)
  - **MinuteSort** (sortiere soviel du kannst in einer Minute)
  - **TerabyteSort** (sortiere 10^{12} Sätze)
- **Ergebnisse**
  - Sortiergeschwindigkeit verdoppelte sich jedes Jahr (1985 – 2000)
  - aber seither nur 20%

    ständige Verbesserung von Preis/Leistung ~68%/Jahr

Sortieren: +68%/Jahr bei Preis/Leistung



nur geringe Verbesserung nach 1995



- **Neuer Benchmark: TPC-App**[10]
  - TPC-App ist ein Applikations-Server- und Web-Services-Benchmark

10. The workload is performed in a managed environment that simulates the activities of a business-to-business transactional application server operating in a 24x7 environment. TPC-App showcases the performance capabilities of application server systems. The workload exercises commercially available application server products, messaging products, and databases associated with such environments.

Zusammenfassung

- **Allgemeine Aspekte des Schichtenmodells**
  - Schichtenmodell ist allgemeines Erklärungsmodell für die DBS-Realisierung
  - Schichtenbildung lässt sich zweckorientiert verfeinern/vergröbern: Anwendbarkeit für TA-Systeme, Verteilte TA-Systeme, DBS, ...
  - Entwurf geeigneter Schnittstellen erfordert große Implementierungserfahrung
  - Konkrete Implementierungen verletzen manchmal die Isolation der Schichtenbildung aus Leistungsgründen (➥ kritische Funktionen haben „Durchgriff“)

- **Klassifikation Verteilter TA-Systeme**
  - Transaction Routing, Programmierte Verteilung, Aufruf von DB-Operationen
  - Mehrrechner-DBS als Shared-Nothing oder Shared-Disk-Systeme
  - Aber: Mehrrechner-DBS verlangen heute in der Regel **ein** (geeignetes) **Datenmodell** und sind nicht überall verfügbar
  - Wie geht man vor, um heterogene Datenbanksysteme in die Anwendungssysteme zu integrieren?

- **Entwicklung verteilter Anwendungen: vor allem Programmierte Verteilung**
  - TP-Monitor übernimmt Kommunikation und Transaktionsverwaltung
  - Unterstützung von Knotenautonomie und Heterogenität
  - keine Verteilungstransparenz
  - eingeschränkte Flexibilität des Datenzugriffs

Zusammenfassung (2)

- **DB-Zugriff übers Web**
  - Transaktionale Eigenschaften?
  - Um hohe Lasten in erträglicher Zeit zu bewältigen, werden Daten-inkonsistenzen (veraltete Zustände), wo möglich, in Kauf genommen: z. B. Vorgenerierung von Angebotsseiten bei eBay
  - Neue Ansätze zur Skalierbarkeit und Beschleunigung
    - Einsatz von Edge-Servern
    - Nutzung von Replikation oder statischem / dynamischem Caching in vielen Variationen
    - DB-Caching mit strukturierten Daten (Cache Groups)
  - Unterstützung durch Middleware: Schnittstellen und Standards verfügbar
  - Probleme: allgemeines Transaktionskonzept und Sicherheit

- **TA-Systeme werden „überall“ eingesetzt**
  - Sie stellen eine ausgereifte Technologie dar
  - OLTP-Hochleistungssysteme sind in vielen Varianten (Plattform, TP-Monitor, Datenbanksystem) auf dem Markt verfügbar
  - Es ist ein Markt von **> 10^{11} $/Jahr**

- **Es gibt eine zunehmende Vielfalt an TPC-Benchmarks**
  - Standardisierung von Arbeitslasten für TA-Systeme
  - einfach strukturierte Lasten: TPC-A, TPC-B
  - komplex strukturierte Lasten: TPC-C, TPC-D, TPC-E, TPC-H, ... (D = *Decision Support*, E = *brokerage firm*, H = *ad hoc*)
  - TPC-App als neuer Benchmark für Applikationsserver und Web-Services
  - „dramatische“ Verminderung der $/tps-Kosten

    ➥ **Der Wettbewerb hat funktioniert!**