Kapitel 2

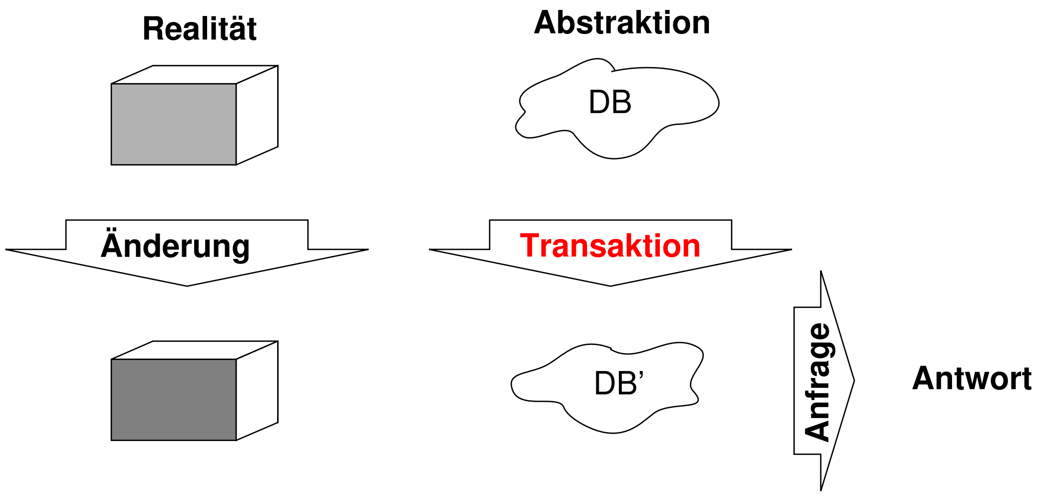
Verteilte Transaktionsverwaltung

Inhalt

- ❑ Transaktionsparadigma
- ❑ (Verteilte) Transaktionssysteme
- ❑ Zusammenspiel der Transaktionsdienste
- ❑ Aufgaben und Struktur eines TP-Monitors
- ❑ Zusammenfassung

Transaktionssysteme (1)

- ❑ 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



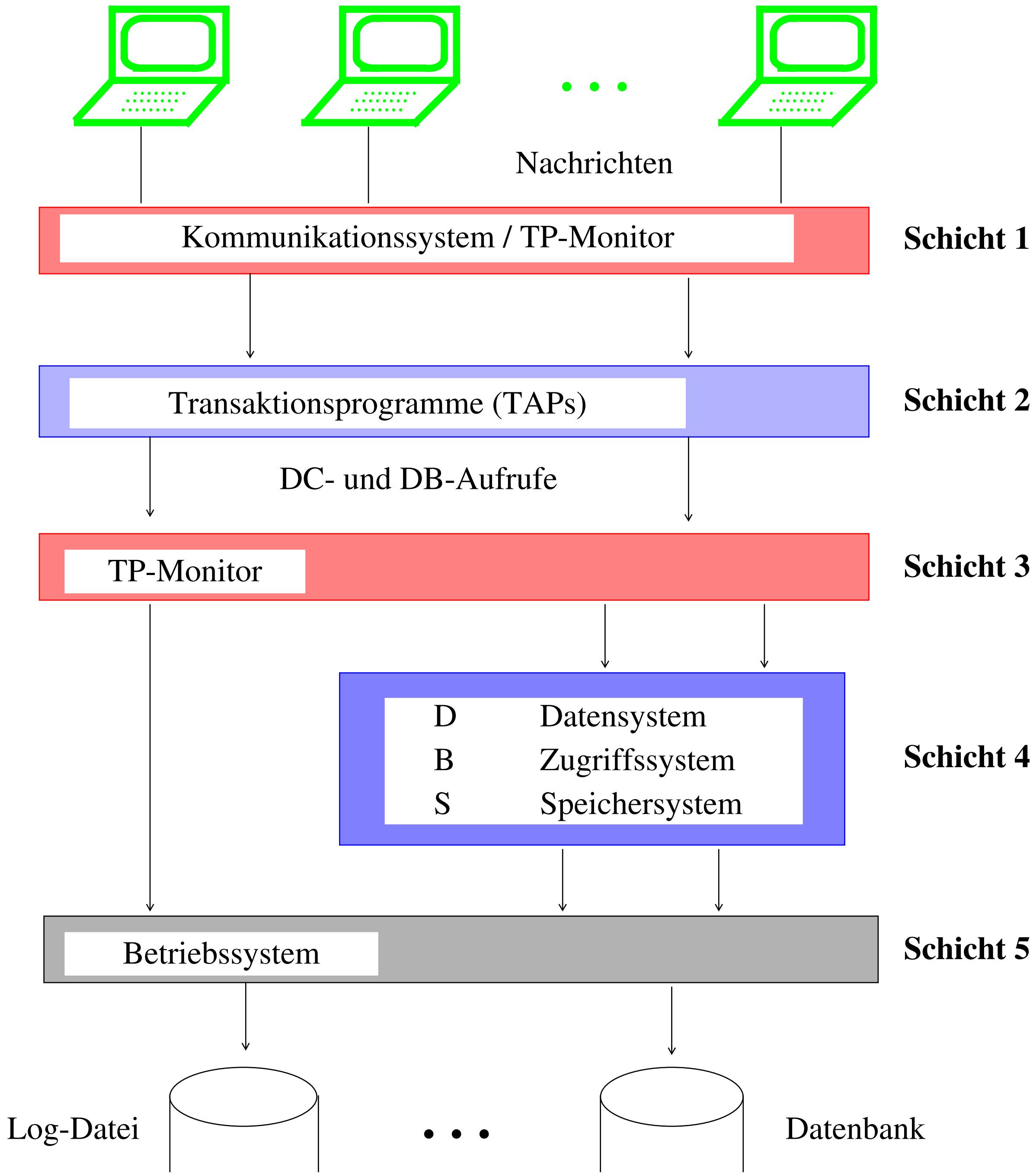
- ❑ **Definitionen einer TA aus verschiedenen Sichten**
  - ➪ Mit einer Transaktion (TA) wird ein Vorgang einer Anwendung in einem Rechensystem abgewickelt. Ein solcher Vorgang bildet typischerweise einen nicht-trivialen Arbeitsschritt (*unit of work*) in betriebl. Abläufen.
  - ➪ Eine (On-line-)Transaktion ist die Ausführung eines Programms, 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.

Transaktionssysteme (2)

- ACID-Paradigma:
  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.
- Struktur eines TA-Programms
  - BEGIN_WORK(): alle nachfolgenden Operationen gehören zur TA
  - COMMIT_WORK(): neuer konsistenter Zustand wird persistent
  - ROLLBACK_WORK(): Fehler erzwingt rückwirkungsfreies Rücksetzen der TA

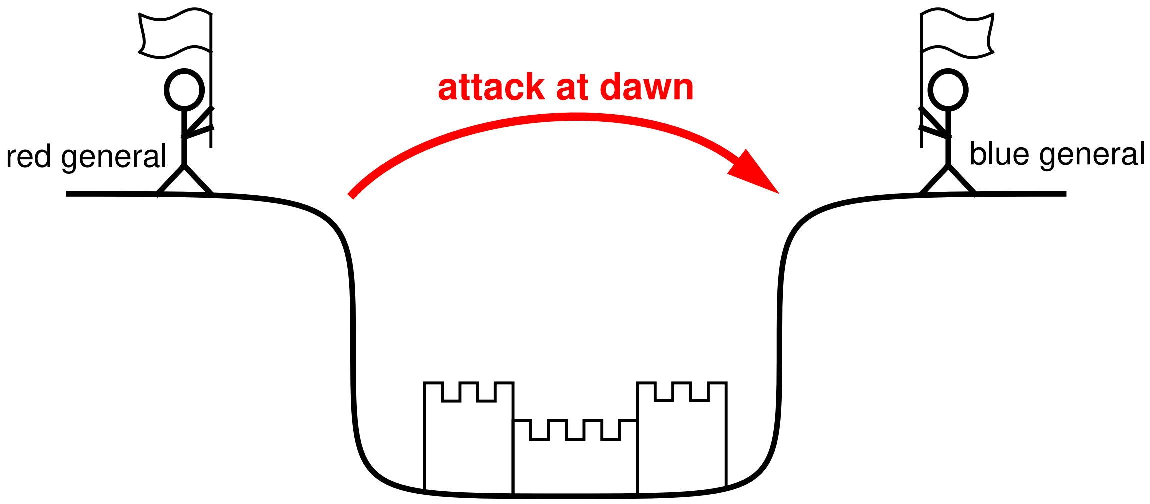
Transaktionssysteme (3)

- Schichtenmodell eines Transaktionssystems



Verteilte Transaktionssysteme (1)

- ❑ Ein *verteiltes System* besteht aus autonomen Subsystemen, die koordiniert zusammenarbeiten, um eine gemeinsame Aufgabe zu erfüllen.
  - ➪ Beispiel: The „Coordinated Attack“ Problem



Generals-Paradoxon

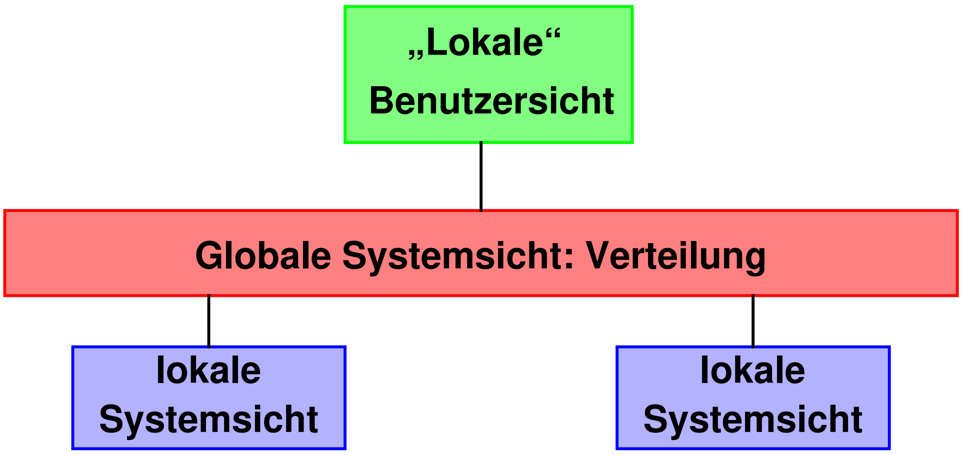
- ❑ Grundproblem verteilter Systeme: Mangel an globalem (zentralisiertem) Wissen!
  - ➪ symmetrische Kontrollalgorithmen sind oft zu teuer oder zu ineffektiv
  - ➪ fallweise Zuordnung der Kontrolle

Verteilte Transaktionssysteme (2)

- Wichtige Aspekte eines (verteilten) **TP-Systems**
  - TP-Systeme sind geographisch verteilt und heterogen
  - TP-System stellt Werkzeuge zur Vereinfachung oder Automatisierung der Anwendungsprogrammierung, -ausführung und -verwaltung zur Verfügung
  - Anfragen und Änderungsanforderungen werden über ein Netz von Geräten weitergeleitet und mit Hilfe von DBS verarbeitet
  - Ausgaben treiben Aktuatoren und Geräte, die den Zustand der Miniwelt verändern oder kontrollieren
  - Anwendungen, Datenbanken und Netzwerke wachsen oft über mehrere Jahrzehnte
  - Es wird ein unterbrechungsfreier Betrieb verlangt (continuous operation, keine geplanten Abschaltzeiten)
  - Es existieren strikte Antwortzeitanforderungen
  - Ein **TP-Monitor** verkörpert in einem TP-System eine Menge zentraler Dienste, die den Fluß der TA durchs System verwalten und koordinieren

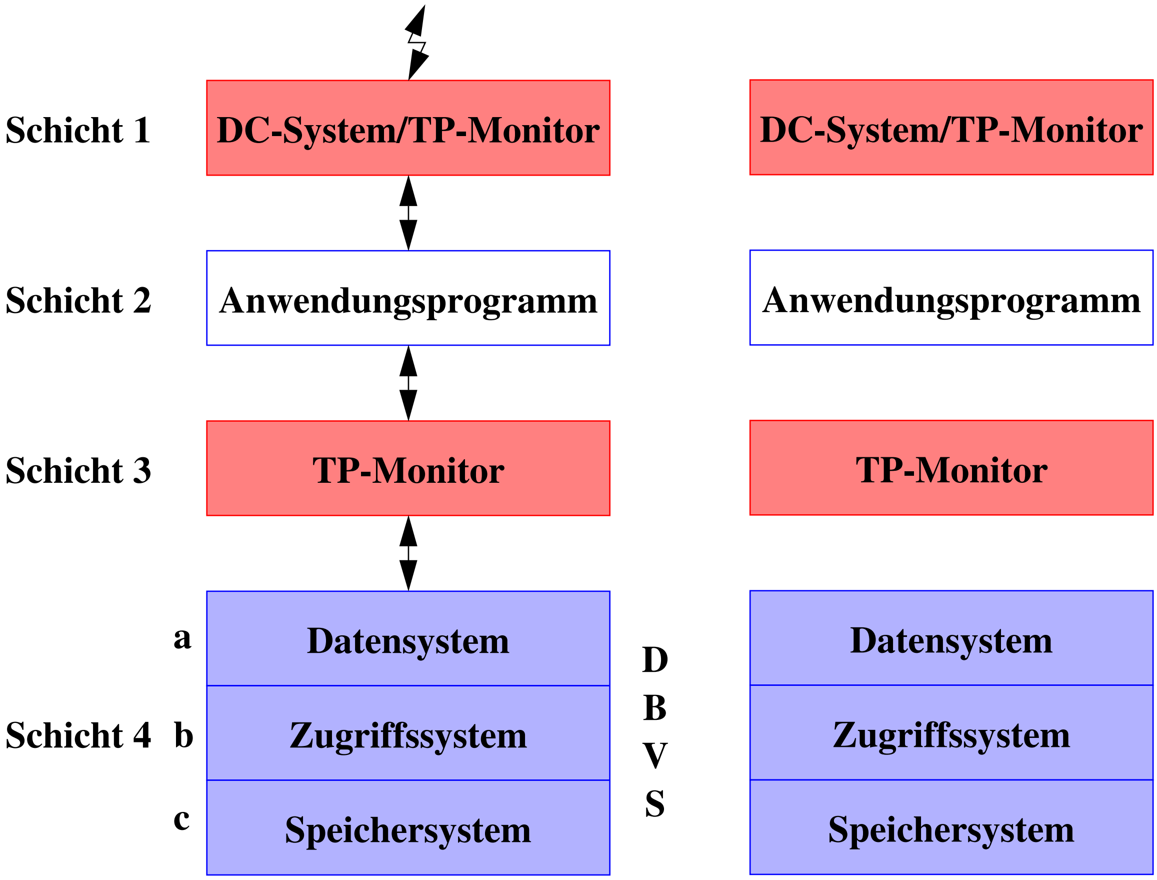
Verteilte Transaktionssysteme (3)

- ❑ Wichtige Aspekte eines (verteilten) **TP-Systems** (Forts.)
  - ➪ Prinzipien: Funktions-, Daten- und Lastverteilung
  - ➪ Vorgehensweise bei der Verteilung
    - Partitionierung der Daten (in der Regel auf ‘natürliche Weise gewachsen’, ggf. mit Replikation)
    - Kommunikation zwischen Prozessen, die die Zugriffe auf jeweils lokale Daten ausführen
- ❑ gewünschte Sichtbarkeit



Verteilte Transaktionssysteme (4)

❑ Homogenes Systemmodell



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

Verteilte Transaktionssysteme (5)

❑ Klassifikation

➯ Verteilung unter Kontrolle des TP-Monitors
(Verteilte DC-Systeme)

- TP-Monitor verbirgt weitgehend Heterogenität bezüglich Kommunikationsprotokollen, Netzwerken, BS und Hardware
- DBS bleiben weitgehend unabhängig (keine Kooperation zwischen DBS)
- heterogene DBS möglich: Einsatz von DBS-Gateways
- geringe Implementierungskomplexität

➯ Drei wesentliche Alternativen:

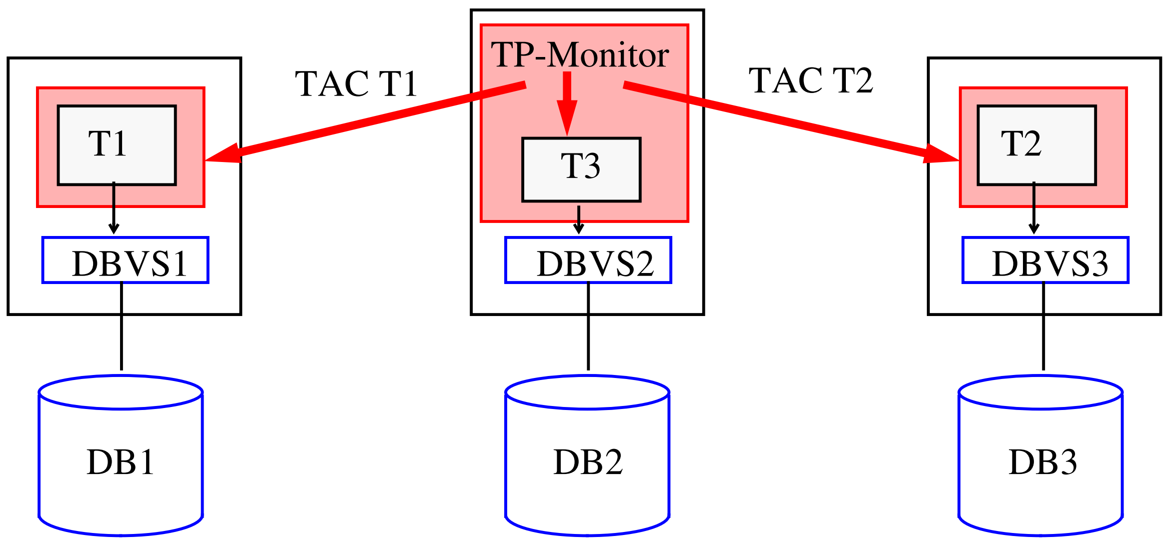
1. ***Transaction Routing***
   - globale Sicht in Schicht 1 (Kommunikationssystem)
   - Einheit der Verteilung ist die Transaktion (TA-Typ)

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

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

Verteilte Transaktionssysteme (6)

- Transaction Routing
  - Prinzip
    - jeder TA-Typ ist einem Rechner fest zugeordnet
    - TP-Monitor kennt TA-Typ-Zuordnung: Erkennung und Weiterleitung des TAC ($\rightarrow$ Ortstransparenz)
    - lokale Transaktionsausführung innerhalb eines Rechners
    - Ausgabenachricht muß ggf. über Zwischenrechner an Benutzer zurückgesendet werden



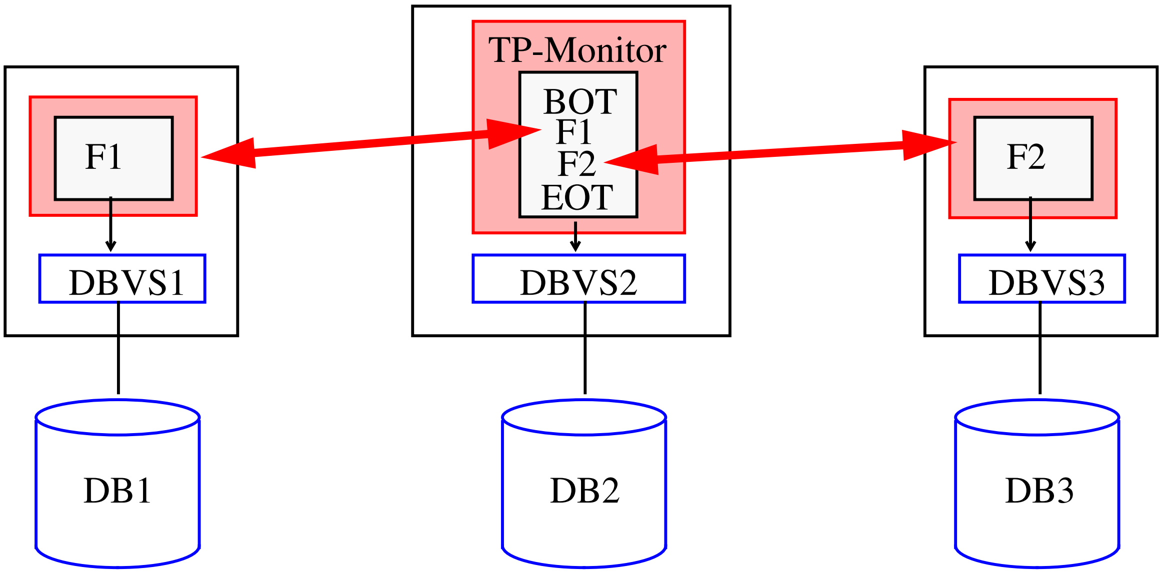
  - Keine Kooperation zwischen DBVS
    - pro Rechner eigene DB/Schemata
    - heterogene DBS möglich
  - Als alleiniges Verteilgranulat zu inflexibel (keine echt verteilte Transaktionsausführung)
  - Beispiel: IMS MSC (*multiple systems coupling*)

Verteilte Transaktionssysteme (7)

❑ Programmierte Verteilung

➪ Verteilung auf Ebene von Anwendungsprogrammen

- Zugriff auf externe DB durch Aufruf eines Teilprogramms (RPC) auf dem betreffenden Rechner
- jedes Teilprogramm greift nur auf lokale DB zu (mit jeweiliger Anfragesprache)



➪ Transaktionsverwaltung

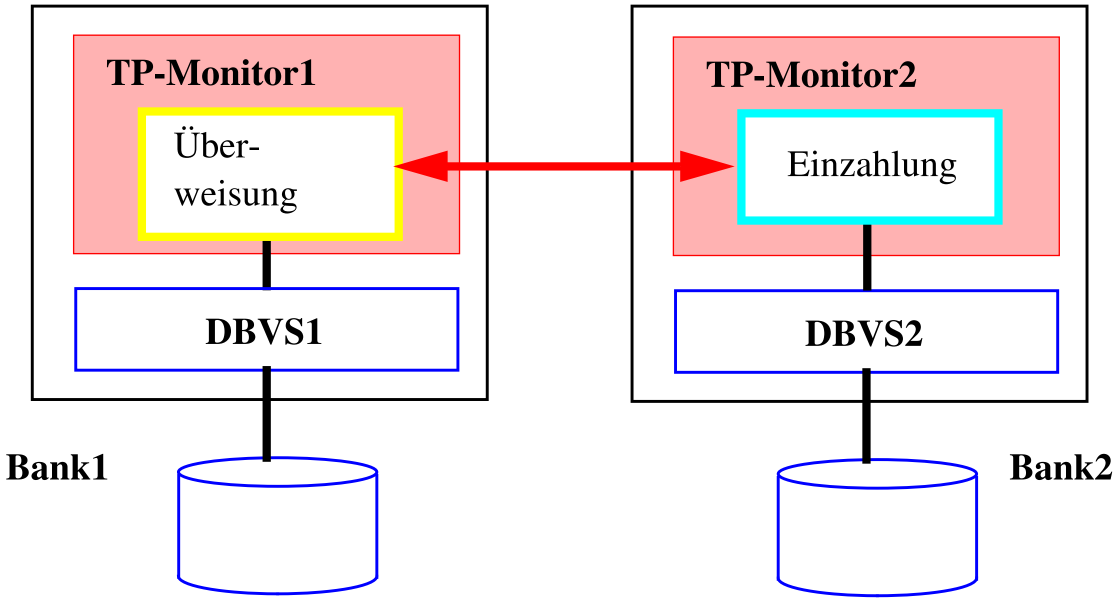
- TP-Monitor koordiniert verteiltes Commit-Protokoll
- DBS müssen DB-Operationen von nicht-lokalen Transaktionen akzeptieren sowie am Commit-Protokoll teilnehmen
- Auflösung globaler Deadlocks über Timeout

➪ Ortstransparenz kann prinzipiell durch TP-Monitor erreicht werden

➪ *Beispiele:* Tandem Pathway, UTM-D,
CICS Distributed Transaction Processing

Verteilte Transaktionssysteme (8)

- Programmierte Verteilung (Forts.)
  - Beispiel: Überweisung zwischen unabhängigen Banken

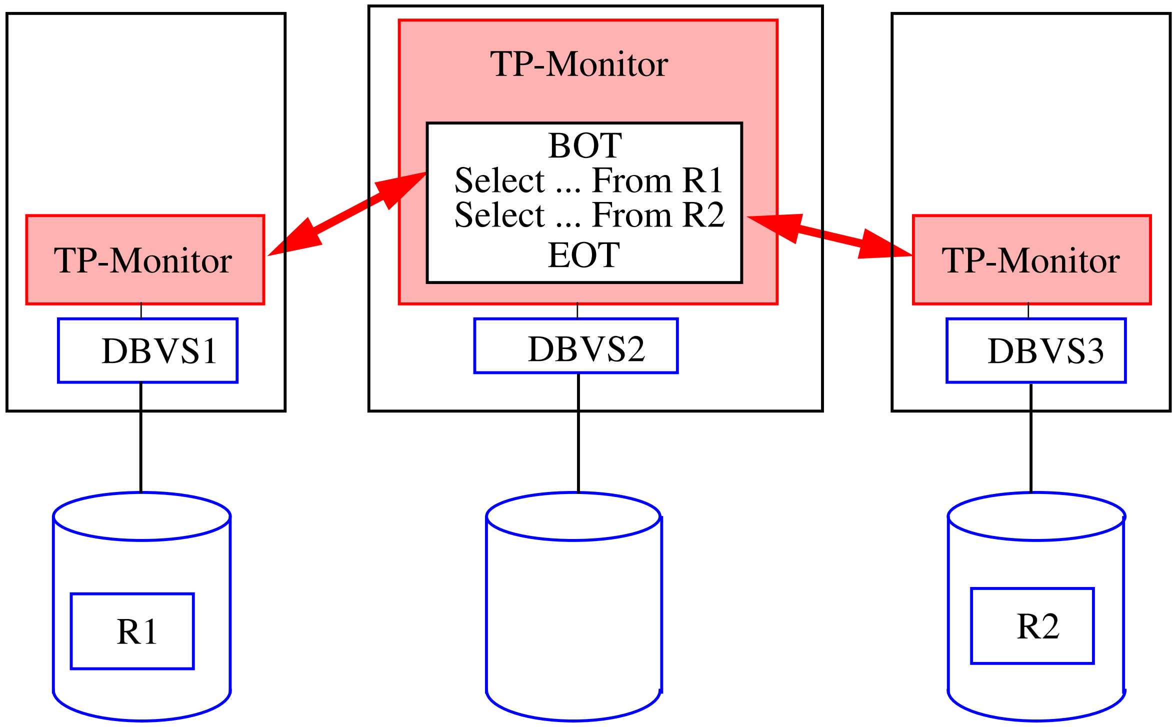



```
Eingabenachricht (Parameter) lesen
BEGIN WORK
{ Zugriff auf lokale DB zur Abhebung
      des Betrags }
          UPDATE ACCOUNT
          SET BALANCE = BALANCE - :S
          WHERE ACCT_NO = :K1;
          • • •
CALL EINZAHLUNG (Bank2, S, K2)
COMMIT WORK
Ausgabenachricht ausgeben
```

```
Parameter übernehmen
        • • •
UPDATE GIROKTO
SET KSTAND := KSTAND+ :S
WHERE KNUMMER = :K2;
        • • •
Ausführung zurückmelden
```

Verteilte Transaktionssysteme (9)

❑ Verteilung von DB-Operationen (durch TP-Monitor)



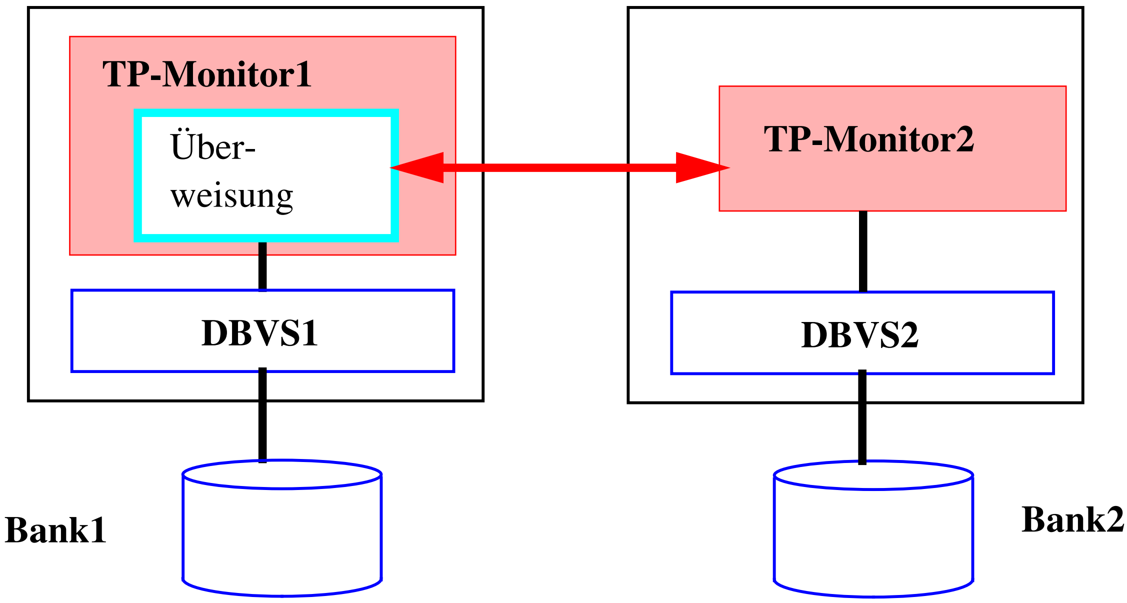
➪ DB-Operationen als Verteileinheiten

- 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*

Verteilte Transaktionssysteme (10)

- Verteilung von DB-Operationen (Forts.)
  - Beispiel



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 Transaktionssysteme (11)

- Bewertung

  - Transaction Routing

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

  - Programmierte Verteilung

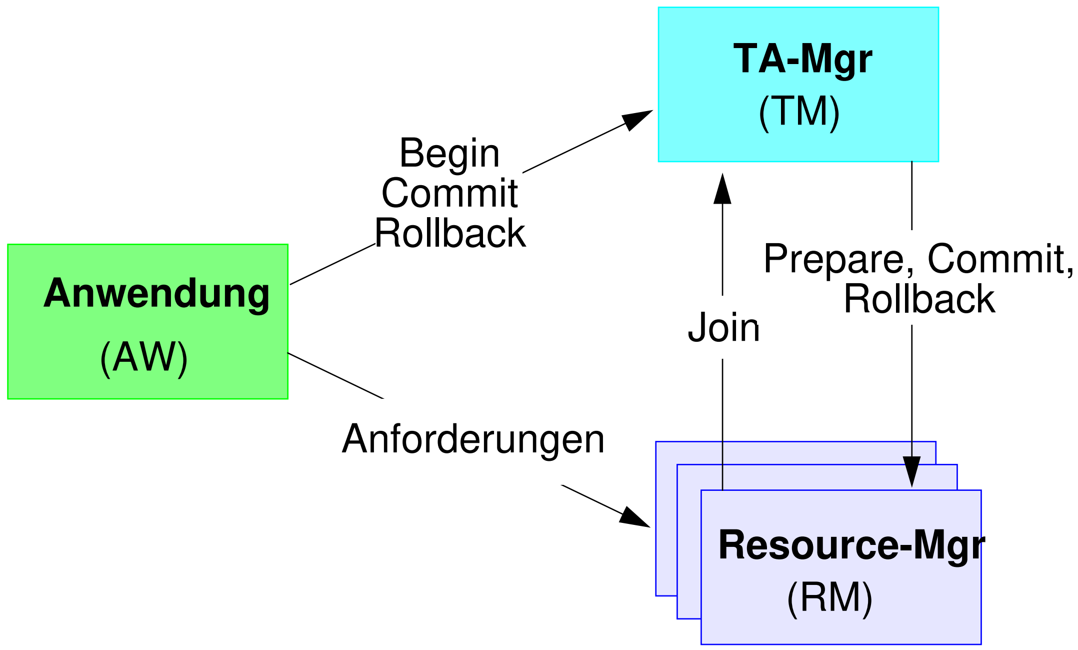
    + sehr hohe Unabhängigkeit gewährleistet
    + Unterstützung von heterogenen DBS und Client-/Server-Ansatz
    + geringe Kommunikationshäufigkeit
    + Nutzung bestehender Anwendungsfunktionen möglich
    + gute Administrierbarkeit
    + von vielen Systemen unterstützt
    - sehr geringe Flexibilität 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

Verteilte Transaktionssysteme (12)

- ❑ Bewertung (Forts.)
  - ➩ 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

Zusammenspiel der Transaktionsdienste (1)

❑ Verteilte Transaktionsverwaltung nach X/OPEN DTP (1991)



❑ TA-orientierte Verarbeitung basiert auf der Kooperation von RM

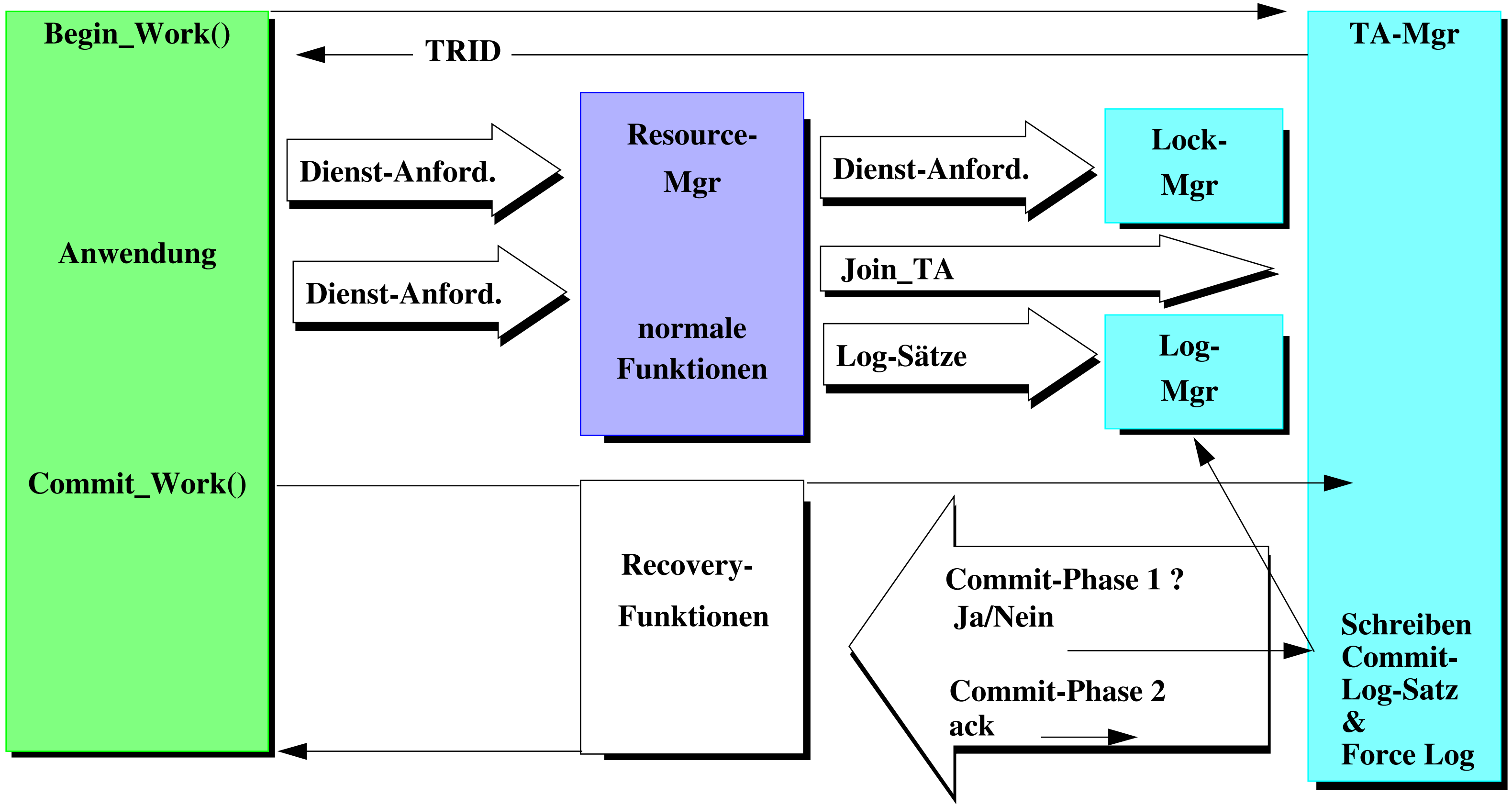
- ➪ RM ist ein Subsystem, das in den TP-Monitor eingebunden ist und 'geschützte Aktionen' auf seinem Zustand bereitstellt
- ➪ Jede SW-Komponente kann RM sein, wenn sie eine Bedingung erfüllt: sie muß an einer transaktionsorientierten Recovery teilnehmen können

Zusammenspiel der Transaktionsdienste (2)

- Drei Typen von RMs
  - Komponenten zur *Systemadministration*
    - bspw. der Katalog und die Präsentationsdienste
  - Basiskomponenten zur *Implementierung von TAs* (zentrale Transaktionsdienste)
    - bspw. TA-Mgr, Log-Mgr und Kommunikations-Mgr
  - alle Komponenten, die *Objekte implementieren,*
    - die von der Anwendung benutzt werden können und die das TA-Paradigma unterstützen
    - z. B. SQL-DBS, Dateisystem, WS-System, X-System, Mail-System, Mgr. für komplexe TA-Abläufe u.a.

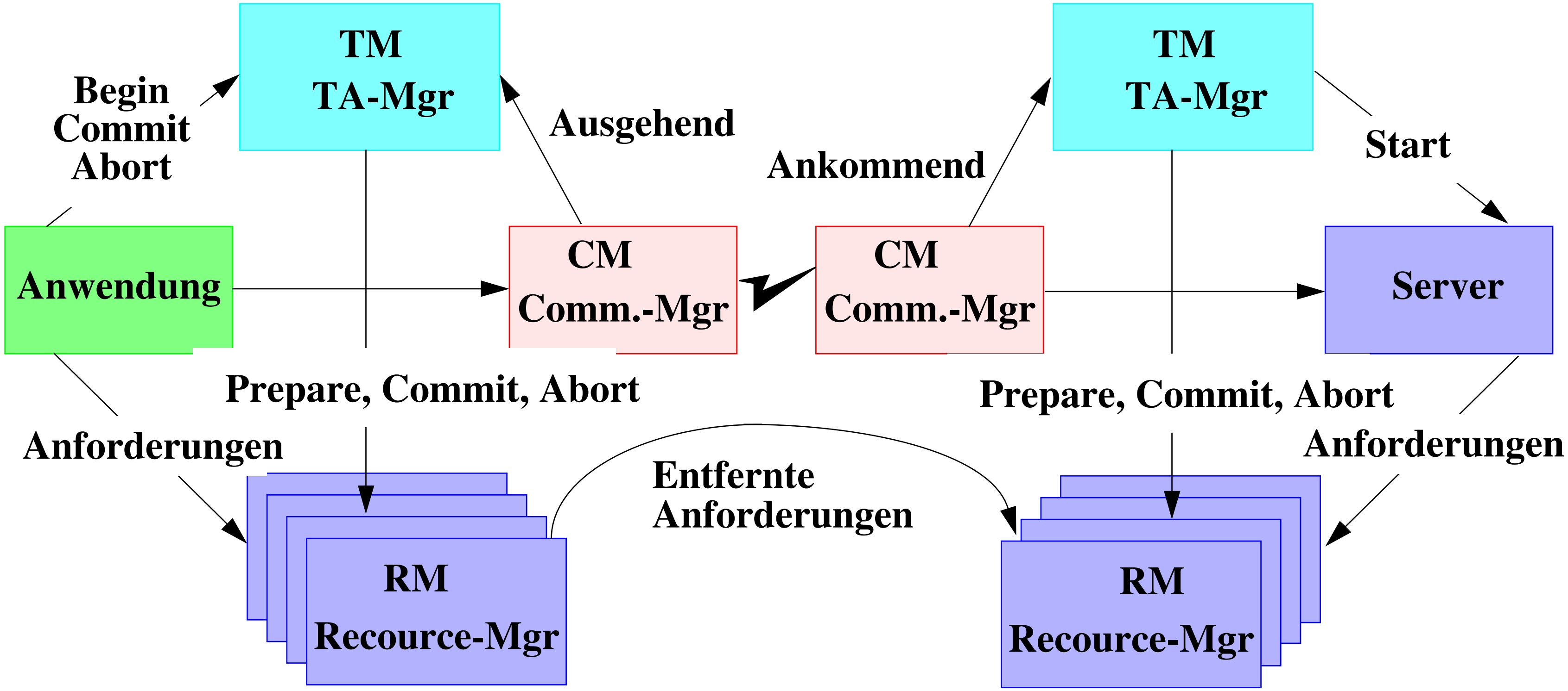
Zusammenspiel der Transaktionsdienste (3)

- Rolle der zentralen Transaktionsdienste (TA-Mgr, Log-Mgr, Lock-Mgr)



Zusammenspiel der Transaktionsdienste (4)

- Verteiltes TP-Modell nach X/Open



Zusammenspiel der Transaktionsdienste (5)

- Strikte Unterscheidung zwischen TP-Monitor und TA-Mgr
  - viele Systembenutzer benötigen den TP-Monitor nicht: Stapel-AW, Ad-hoc-Anfragen über eine direkte SQL-Schnittstelle
  - spezielle AW-Systeme (z. B. CAD) haben ihre eigene Terminalumgebung
  - alle Aktivitäten brauchen jedoch Transaktionsunterstützung
- Eine Konsequenz ist die Trennung der Komponenten zur Durchführung
  - der **Transaktionskontrolle** (TA-Mgr) und
  - des **transaktionsorientierten Betriebsmittel-Scheduling** (TP-Monitor)
- Schnittstellen
  - Aufrufe zwischen RM sind TRPCs; der entsprechende Mechanismus wird vom TP-Monitor bereitgestellt
  - Aufrufe von einem AP zu den verschiedenen RM (SQL, X-Windows u. a.) sind anwendungsspezifisch (sind mit einem TRID gekennzeichnet)
  - Log-Schnittstelle für die RM zum direkten Schreiben von Log-Sätzen

Zusammenspiel der Transaktionsdienste (6)

❑ TP-Monitor-Unterstützung für RM

➩ *Start und Restart des Systems*

- alle RM der spezifizierten Konfiguration müssen zunächst gestartet werden
- das Recovery-Protokoll bei Crash wird zwischen RM und TA-Mgr direkt abgewickelt
- der TP-Monitor hat hierbei nur Kontrollfunktion

➩ *Definition eines neuen RM*

- Übernahme der Beschreibungsdaten in den Katalog und Aktualisierung der Konfigurationsdaten

➩ *Änderung der Prozeßkonfiguration*

- z. B. Erweiterung einer Server-Klasse

➩ *Handhabung von TRPCs*

- als grundlegender Mechanismus für die Zusammenarbeit aller Komponenten

Aufgaben eines TP-Monitors (1)

- Grundlage: Resource Manager (RM)

  - Systemkomponenten, die **Transaktionsschutz für ihre Betriebsmittel** (BM) in der Art bieten, daß diese BM in globale Transaktionsdienste eines TP-Monitors integriert werden können, heißen **Resource Manager**.

- Aufgaben

  - **Bewältigung der Heterogenität**:

    - Zugriff auf heterogene DB oder verschiedenartige RM bei gleichzeitiger Gewährleistung von ACID für die gesamte Funktion

  - **Kontrolle der Kommunikation**:

    - Bei verteilter Verarbeitung unterliegt die Kommunikation auch der TA-Kontrolle. Das kann erreicht werden durch einen Mechanismus wie ‘transactional remote procedure call’ (TRPC).

    - Der Kommunikations-Manager handhabt und kontrolliert Nachrichten und Sitzungen; er ist deshalb ein RM.

  - **Verwaltung der Terminals**:

    - Nachrichtenkommunikation mit unterschiedlichsten Terminals (Workstations, . . ., Zapfsäulen)

    - Nachrichtenkommunikation muß Teil der TA sein, damit für den Endbenutzer die ACID-Eigenschaften gewährleistet werden können.

    - Der TP-Monitor muß diese Aufgabe übernehmen (schwierige Probleme im Fehlerfall)

Aufgaben eines TP-Monitors (2)

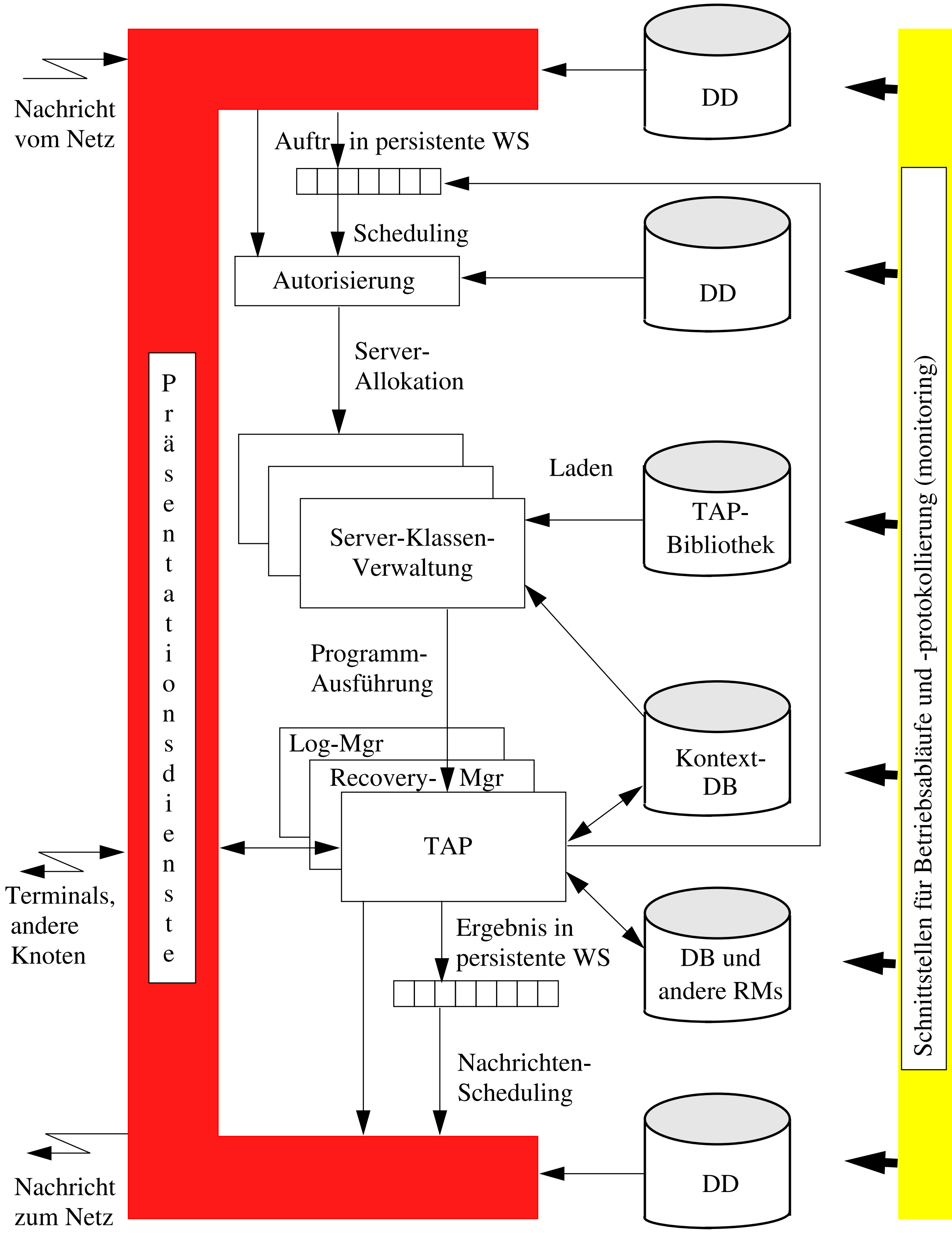
- Aufgaben (Forts.)
  - **Präsentationsdienste**:
    - Um ACID für den Endbenutzer zu realisieren, ist im Fehlerfall seine “Sicht” wiederherzustellen.
    - Wenn er bspw. ein X-Window-System verwendet, sind Fenster, Cursorpositionen u. a. wieder aufzubauen, d. h., ein X-Client muß in einer TA-orientierten Umgebung als RM agieren.
  - **Kontextverwaltung**:
    - Kontexte sind bspw. bei Mehrschritt-TA oder bei Abwicklung einer langen TA durch “Mini-Batches” erforderlich.
    - Die Speicherung und Wiederherstellung von Kontexten ist gebunden an die SoC der TA, die einen Kontext erzeugte oder zuletzt modifizierte.
    - Wartung der Kontexte ist Aufgabe des TP-Monitors.
  - **Start/Restart**:
    - Praktisch alle Komponenten einer TA-orientierten Ablaufumgebung benötigen Transaktionsdienste.
    - Deshalb muß der TP-Monitor auch den Restart nach einem Fehler übernehmen, um einen konsistenten Zustand nach den ACID-Regeln aufzubauen.

Aufgaben eines TP-Monitors (3)

- Aufgaben (Forts.)
  - **Programmverwaltung**
  - **Konfigurationsverwaltung**
  - **Lastbalancierung**
  - **Autorisierung**
    - Unterstützung des Basis-BS bei
      - Benutzer-Authentifikation
      - Session-Authentifikation
      - Zugriffskontrolle bei Dateien
      - Speicherkontrolle (Schutzringe)
    - darüberhinaus hat der TP-Monitor zu prüfen, ob
      - <u>dieser</u> Benutzer
      - <u>diesen</u> Service (Anwendung) von
      - <u>diesem</u> Terminal (dieser Anwendung) an
      - <u>diesem</u> Tag in der Woche (Tageszeit) auf
      - <u>diesem</u> Objekt mit
      - <u>diesen</u> Parametern aufrufen darf.
  - **Bereitstellung von Administrationsschnittstellen**

Struktur eines TP-Monitors

❑ Kontrollfluß durch einen TP-Monitor (vereinfacht)



Zusammenfassung (1)

❑ Middleware-Aspekte

- ➪ TA-Konzept als Basis für (DB-)Middleware
  - TA-Schutz für verteilte Abläufe notwendig
  - damit sind die Prinzipien der Verteilten Transaktionsverwaltung grundlegend für jede Art der Middleware-Integration
  - TP-Monitor/TA-Manager bzw. proprietäre Komponente, die Transaktionskontrolle durchführt, als notwendige Komponente jeder DB-Middleware
- ➪ TP-Monitor selbst gehört zur DB-Middleware
  - Integration von verteilten, heterogenen RM
  - Herstellung von globalen Sichten
    (TA-Routing, Vert. Progr., Vert. v. DB-Ops.)
  - jedoch kein integriertes Schema und keine verteilte Bearbeitung einzelner DB-Operationen

❑ Der **TP-Monitor ist ein RM,**

- ➪ der andere RM und Ressourcen wie Prozesse, Tasks, Zugriffsrechte, Programme und Kontexte verwaltet;
- ➪ von den Funktionen her ähnlich wie ein BS
  - aber: Allokation von Ressourcen auf Basis von Anforderungen
  - alle TP-Monitor-Aufgaben fallen bei jeder Anforderung an: **Authentifikation, Autorisierung, Lastbalancierung, Prozeß-Allokation, Speicherzuordnung**
  - Unterstützung der RM beim Restart