

## 2. Client/Server-Systeme

- **Ziel: Darstellung von Systemaufbau und -dynamik**

- Modellhafte Darstellung der Systemarchitekturen
- Zusammenspiel der Komponenten

- **Â Transaktionssysteme sind meist Client/Server-Systeme!**

- **Sicht des Endbenutzers**

Dialogschritt, Transaktion, Vorgang

- **Entwurfsaufgaben**

- Zerlegung/Kooperation der Komponenten
- Komponentenzuordnung in verteilter Umgebung

- **Client/Server-Modell**

- Unterscheidende Eigenschaften
- Eigenschaften von zwei- und dreistufigen C/S-Systemen

- **Wie macht man Server effizient?**

- Server-Strukturen und -Techniken
- Prozess- und Programmverwaltung

- **TP-Monitor<sup>1</sup>**

- Eigenschaften und Funktionalität
- Einsatz in C/S-Umgebungen

- **Aufbau eines DB-Servers**

Drei-Schichten-Architektur

- **Ablaufbeispiel**

- Wie kommen die Antwortzeiten zustande?
- Optimierungsmöglichkeiten

- **Client/Server-Verarbeitung in offenen Systemen**

- **Zusammenfassung – Vielfalt an Bezeichnungen**

---

1. Gray, J., Reuter, A.: Transaction Processing: Concepts and Techniques, Morgan Kaufmann, 1993

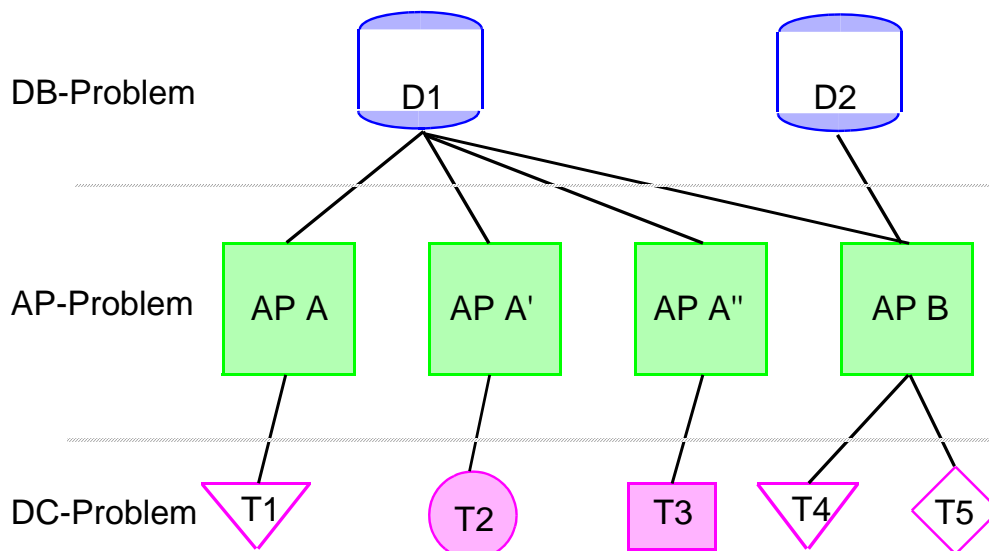
## AW-Entwicklung – allgemeine Probleme

- **Problem**

Komplexität der Kommunikations- und Verarbeitungsbeziehungen

- K E/A-Stationen (Terminals, PCs, PDAs, Mobile Telefone...)
- M Anwendungen
- N Dateien

- **Herkömmliche (statische) Struktur**



- **Generelle Aufgaben**

- Ablaufkontrolle
- Zuteilung der Betriebsmittel (BM) (Ressourcen-Allokation)
- Synchronisation und Sicherung der Datei-Zugriffe
- Verwaltung und Abbildung von Dateistrukturen
- Steuerung der Kommunikationsbeziehungen

Â **Wer hält das alles zusammen?**

## AW-Entwicklung – allgemeine Probleme (2)

- **DB-Problem**

Individuelle Datenverwaltung führte zu

- umständlichem Zugriff
- geringer Flexibilität
- Redundanz
- Konsistenzverlust

- **AP-Problem**

Individuelle Programmentwicklung führte zu

- „amorphen“ Strukturen
- komplexer Ablaufkontrolle und Betriebsmittel-Verwaltung
- Redundanz

- **DC-Problem**

Individuelle Terminalverwaltung führte zu

- Typ-Abhängigkeit
- geringer Flexibilität
- Redundanz

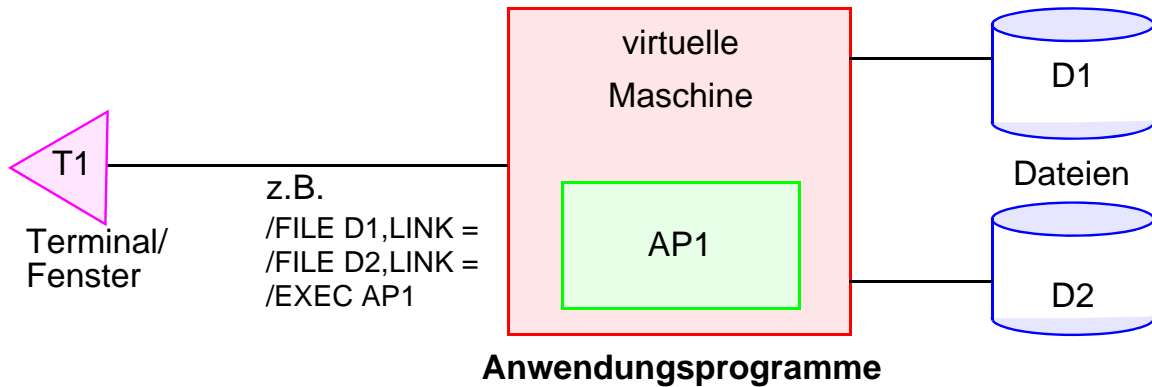
- **Zusätzliche Probleme**

- Verteilung
- Heterogenität
- offene Systeme / Interoperabilität
- ...

Â **Client/Server-basiertes TA-System soll alles lösen!**

## Sicht des Endbenutzers

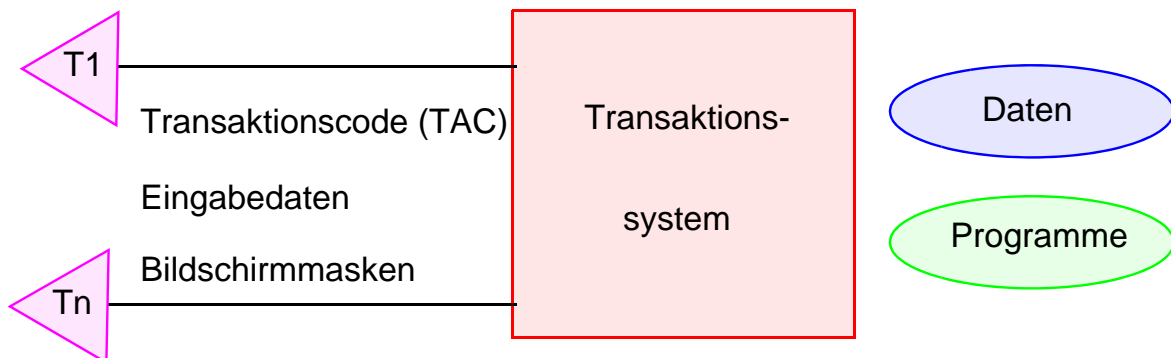
- **Benutzerschnittstelle für TA-Systeme?**



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

Â Wie kommt das System mit n Benutzern (mit separaten Prozessen) zurecht?

- **besser:** TA-System „führt“ den Benutzer



- problembezogene Funktionen
- einfache Bedienung
- Datenunabhängigkeit
- Programmunabhängigkeit

Â **System kann Ressourcen (Zeit, Speicher, Programme, Daten, ...)**  
besser zuteilen

## Sicht des Endbenutzers (2)

- **Einfachster Fall: Vorgang = Dialogschritt = Transaktion**
  - Eingabe: TAC + Daten („Parameter“; oft mehrere Formulare)
  - Ausgabe: Rückmeldung (Bestätigung oder Fehler)  
+ „Bitte neue Funktion eingeben“

z. B. Formular AUSZAHLUNG wird **am Bildschirm** ausgefüllt:

Textfeld

Eingabefeld mit vorgegebenem Inhalt

Funktion: Auszahlung  
Zweigstelle: KSK KL1      Datum 06/05/2009  
Schalter: 17      Konto-Nr: \_\_\_\_\_  
Geldbetrag bitte ankreuzen:  
100 €      ( )      400 €      ( )  
200 €      ( )      1000 €      ( )  
anderer Betrag      ( )

- **Transaktionsprogramm „Auszahlung“:**

**Read message (kontonr, schalternr, zweigstelle, betrag) from Terminal;**

**BEGIN TRANSACTION**

**UPDATE Konto**

SET kontostand = kontostand - *betrag*

WHERE konto\_nr = *kontonr* and kontostand >= *betrag*

...

**UPDATE Schalter**

SET kontostand = kontostand - *betrag*

WHERE schalter\_nr = *schalternr*

**UPDATE Zweigstelle**

SET kontostand = kontostand - *betrag*

WHERE zweig\_stelle = *zweigstelle*

**INSERT INTO Ablage** (zeitstempel, werte)

**COMMIT TRANSACTION;**

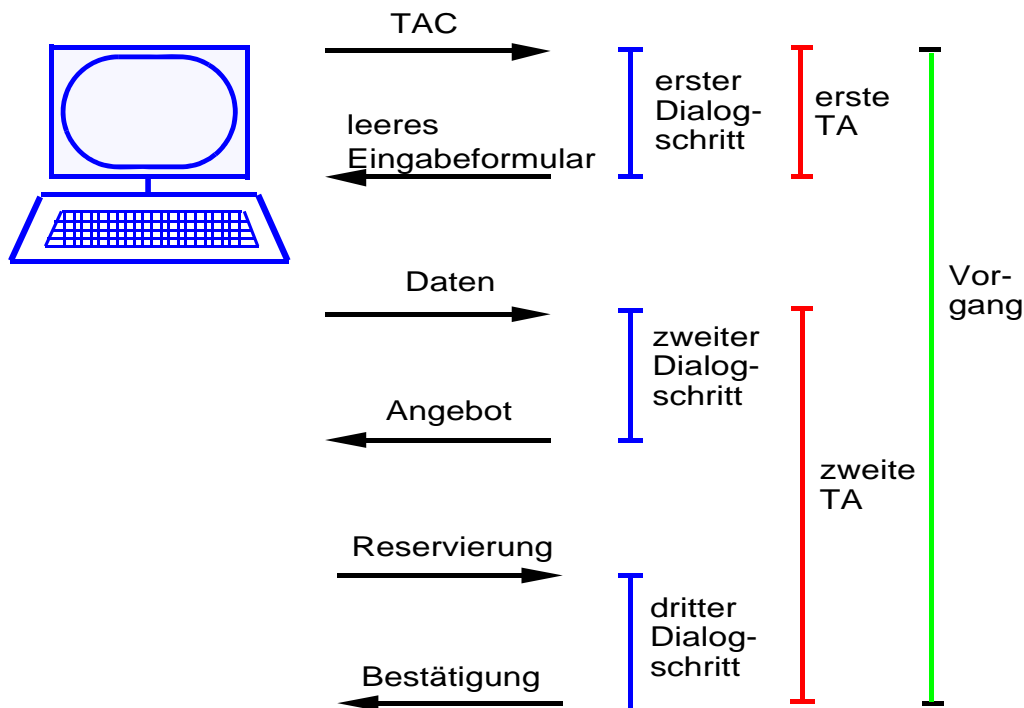
**Write message (kontonr, kontostand, . . .) to Terminal**

## Sicht des Endbenutzers (3)

- Standardfall: Mehr-Schritt-Vorgänge

- Beispiel: Reservierung

- erst zeigen, was frei ist, dann reservieren
- Abwicklung/Ausführung einer Funktion im Dialog



- Dialogschritt  $\leq$  Transaktion  $\leq$  Vorgang  
Sind n Dialogschritte in einer Transaktion akzeptabel?

- Typische Interaktionsmuster

- **zwei Dialogschritte:** Angebot – Auswahl
- **n Dialogschritte:** z. B. Buchung mit allen Einzelposten
- wenig Auswahlmöglichkeiten für den Benutzer

- Fehlerbehandlung

- Anwendungsfehler, Systemausfall, ... verborgen für den Client
- Benutzer erwartet von einem Dialogschritt: **Alles oder nichts!** (d. h. ACID-Transaktion)

## Begriffe

- **Eingabenachricht**

Übertragungseinheit vom Client / Terminal an das Transaktionssystem;  
ein einzelner Auftrag

- **Ausgabenachricht**

Übertragungseinheit vom Transaktionssystem an Client / Terminal;  
Ergebnis eines einzelnen Auftrags

- **Dialogschritt (Transaktionsschritt)**

Zusammenfassung aller Verarbeitungsschritte im Transaktionssystem  
vom Eintreffen einer Eingabenachricht bis zum Absenden der dazu-  
gehörenden Ausgabenachricht; Abwicklung eines einzelnen Auftrags

- **Transaktionscode (TAC)**

Aufrufname einer Funktion, meist vier oder acht Zeichen lang oder kurzer  
charakterisierender Begriff

- **Vorgang**

Zusammenfassung aller Dialogschritte, die zur Abwicklung einer Funktion  
notwendig sind

- **Transaktion**

- unteilbarer (atomarer) Übergang des Transaktionssystems von einem  
konsistenten Zustand in den nächsten
- besteht aus einem oder mehreren Dialogschritten
- **Achtung:** Viele Transaktionssysteme erlauben **keine Dialogschritt-  
übergreifenden Transaktionen!**

# Entwurfsaufgaben

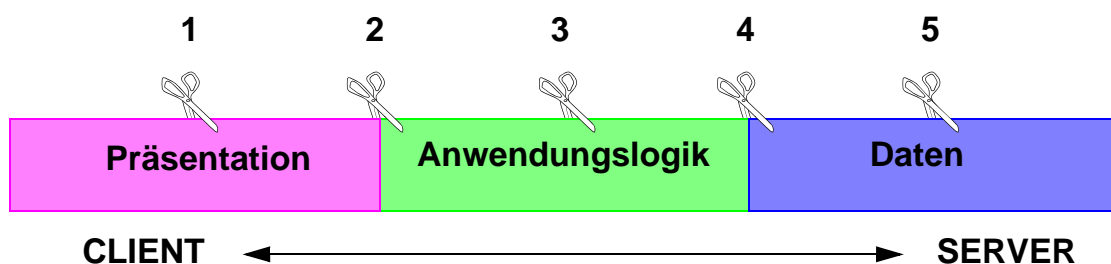
- **Allgemeine Fragen**

- Aus welchen Komponenten ist es aufgebaut?
- Wie und wo arbeiten diese Komponenten zusammen?

- **Applikationen in betrieblichen Informationssystemen**

- lassen sich häufig grob durch folgende Funktionalität charakterisieren:
  - Präsentation oder Benutzerschnittstelle:  
oft als GUI (*graphical user interface*) ausgeprägt
  - Applikationslogik (*business logic*) und
  - Datenhaltung
- werden typischerweise durch verteilte Client/Server-Systeme realisiert

- **Logische Strukturierung/Zerlegung** kann auf verschiedene Weise erfolgen



- **Zuordnung in C/S-Systemen**

- **2-stufige Architektur (2-tier)** ---> ein Schnitt
  - Spektrum vom „thin client“ zum „fat server“ und umgekehrt
  - Gibt es einen „goldenen Schnitt“?
- **3-stufige Architektur (3-tier)** ---> zwei Schnitte
  - bessere Gliederungs-/Skalierungsmöglichkeiten
  - Ist die Zerlegung offensichtlich?
- allgemein: **n-stufige Architektur (n-tier)**

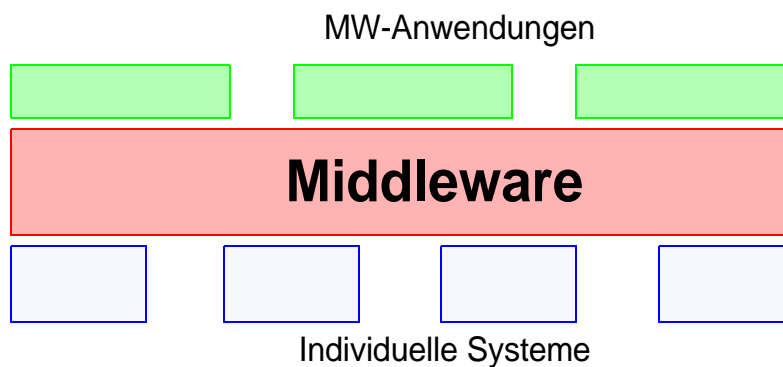


## Entwurfsaufgaben (2)

- **Kooperation der Komponenten**

- Ablauf in heterogenen Umgebungen (Plattformen, Kommunikationsprotokolle)
- Wie werden die Komponenten wieder „zusammengeklebt“?
- Einsatz von Verteilungsplattformen
  - Sie bieten eine integrierte Sicht auf die Dienste der individuellen Systeme
  - sog. Middleware implementiert die integrierte Sicht
  - „/“ von C/S

- **Verteilungsplattform** garantiert transparente Verteilung



Â Vielzahl an Protokollen zur Unterstützung der Interaktion zwischen verteilt ablaufenden Anwendungskomponenten

- **Verbergen von Entwurfs- und Ablaufentscheidungen**

- Unterschiedliche Transparenzanforderungen für
  - Sachbearbeiter
  - AW-Programmierer
  - Systemadministrator
- Einsatzspektrum: lokal, weltweit verteilt

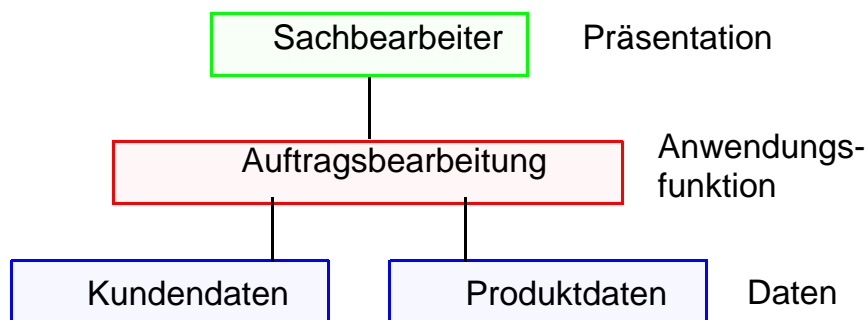
Â **Ortstransparenz** ist besonders wichtig

## Entwurfsaufgaben (3)

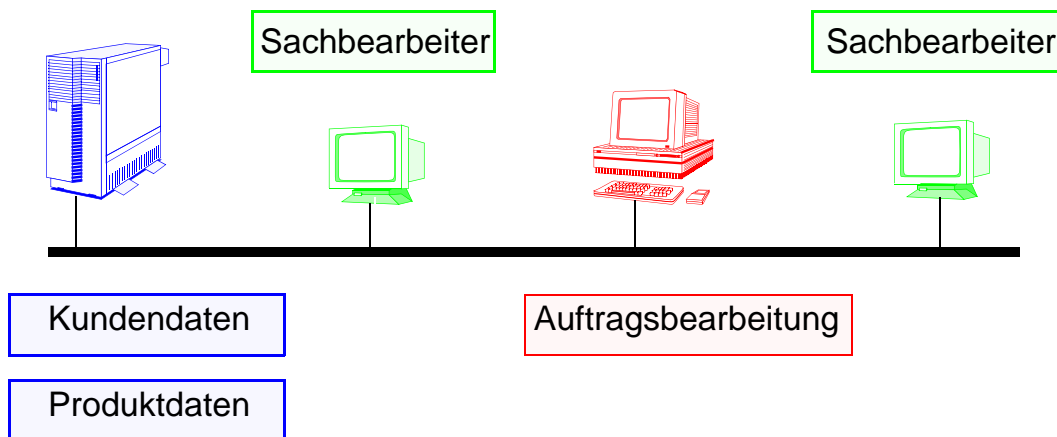
- **Zuordnung**

- 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

- **Beispiel:** SW-Module einer einfachen Client/Server-Anwendung



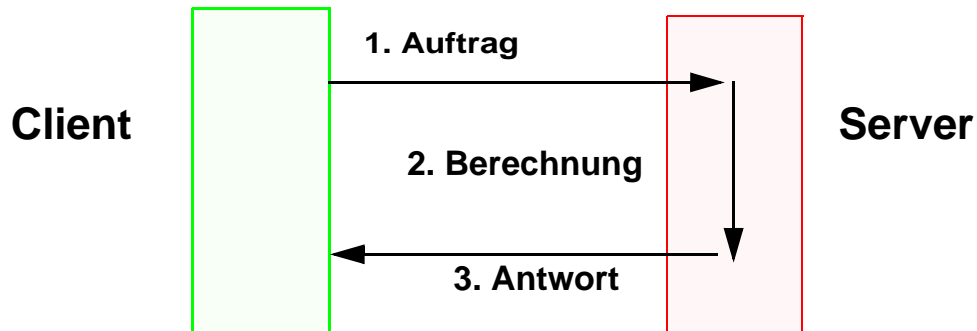
Â **Mögliche Abbildung** auf ein konkretes Rechnersystem  
(Konfigurationsproblem)



Â **Die Bewertung** einer gegebenen Konfiguration und die **Bestimmung der optimalen Lösung** sind i. Allg. sehr schwierige Probleme (wird nicht vertieft)

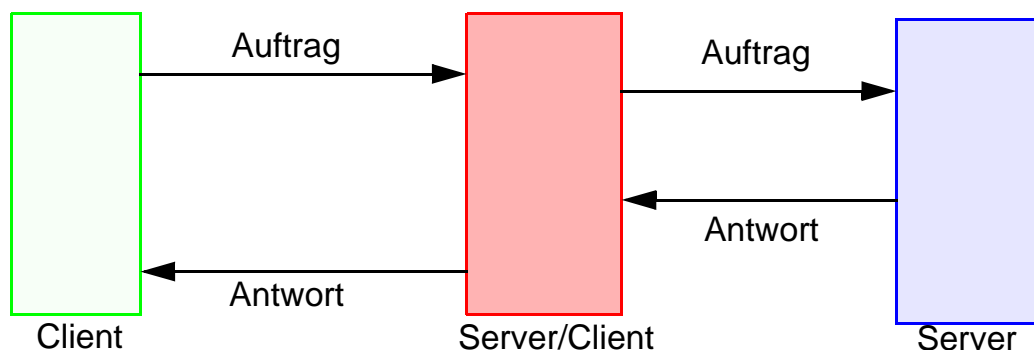
# Client/Server-Modell

- **Grundschemata der Kooperation:**



- **Eigenschaften**

- Initiative zur Interaktion geht vom Client aus
- Server veröffentlicht Bereitschaft, Aufträge entgegenzunehmen
- Ein Auftrag wird als ein Funktionsaufruf **kontextfrei** verarbeitet
- Client/Server-Modell legt Rollen der Beteiligten und die zeitliche Abfolge der Interaktionsschritte fest (inhärente Asymmetrie der Rollen)
- Es existiert i. Allg. eine **n:m-Beziehung zwischen Clients und Servern**
- Die Rollen von Client und Server können wechseln



- **Kommunikation**

- Typischerweise wird ein logischer Funktionsaufruf unterstellt
- Modell bezieht sich auf logische Verteilungen (lokal, entfernt)

## Client-/Server-Modell (2)

- **Grobe Definition**

Client und Server sind separate logische Einheiten, die (über ein Netzwerk) zusammenarbeiten, um eine Aufgabe durchzuführen.<sup>2</sup>

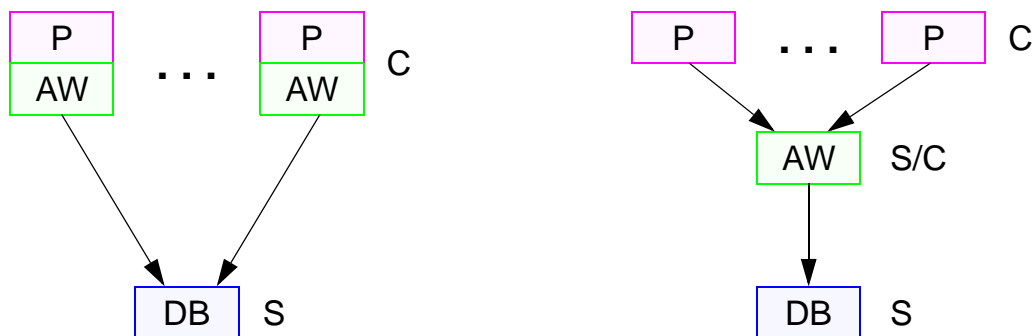
- **Verfeinerung der Eigenschaften**

- **Dienste (Services)**

C/S verkörpert primär Beziehung zwischen Prozessen. C/S bewirkt klare Funktionstrennung, die durch Nutzung von Diensten erzielt wird.

- **Gemeinsame Betriebsmittel (Ressourcen)**

Server kann gleichzeitig viele Clients „bedienen“ und deren Zugriffe auf gemeinsame Betriebsmittel steuern und überwachen.



AW-Code im Client oder im Server?

- **Asymmetrische Protokolle** ((1:n)-Beziehung in beiden Richtungen)

Server bieten (passiv) Dienste an, während Clients sie (aktiv) anfordern.

- **Ortstransparenz**

Die C/S-Software „maskiert“ gewöhnlich den Ort des Servers vor den Clients und leitet Dienstaufträge, falls erforderlich, um.

---

2. In no way does internet computing replace client/server computing. That's because it already is client/server computing (Herb Edelstein).

## Client-/Server-Modell (3)

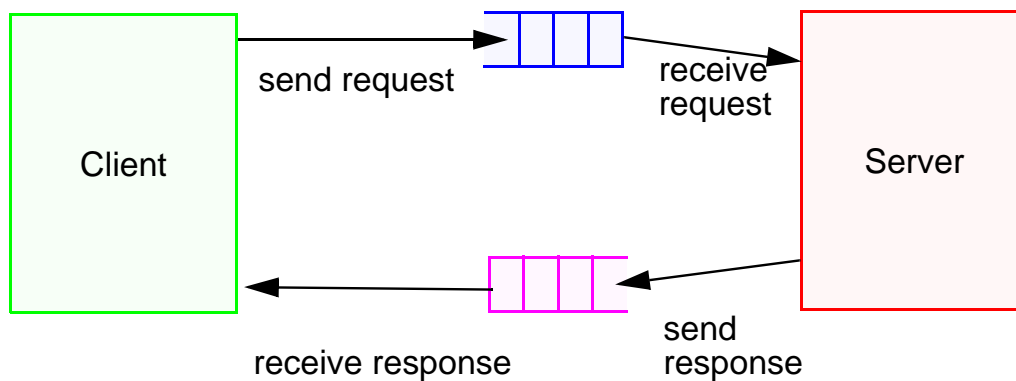
- Verfeinerung der Eigenschaften (Forts.)

- **Plattformunabhängigkeit**

C/S-Software ist unabhängig von HW- oder BS-Plattformen.

- **Nachrichtenbasierter Austausch** (*message-passing mechanisms*)

Clients und Server sind lose gekoppelte Systeme. Nachrichten für Dienst-anforderung und Ergebnisbereitstellung: synchron, asynchron.



Anforderungs- und Ergebnis-WS sind oft **persistente Objekte**

### Â Wichtige Infrastrukturen

- **MOM:** Message-Oriented Middleware (z. B. MQ-Series)
- **CORBA:** architektur- und sprachunabhängig
- **DCOM:** Distributed Component Object Model

- **Kapselung von Diensten**

Server erkennt aus Nachricht, welcher Dienst angefordert wird.

Wie der Dienst erbracht wird, **bestimmt allein der Server**.

- **Integrität**

Programme und Daten eines Servers sollten **zentral gewartet** werden, (kosteneffektiv, höhere Datenintegrität).

## Client-/Server-Modell (4)

- Verfeinerung der Eigenschaften (Forts.)

- Skalierbarkeit

- Horizontale Skalierbarkeit: Mehr Clients ohne Leistungsbeeinflussung.

- Vertikale Skalierbarkeit: Größere/schnellere Server (Server-Klasse)

- Server-Aspekte: verallgemeinert

- Auftrag — Berechnung — Ergebnis (Request — Response)
  - Verschiedenste Protokolle
    - Fernaufruf (Remote Procedure Call, RPC)
    - HTTP, RMI, ...
  - verbindungsorientiert oder verbindungslos
  - Ein wichtiges **zusätzliches Konzept: „Session“**
    - Zusammenhang zwischen aufeinanderfolgenden Aufträgen
    - Server muss **Kontext** verwalten und benutzen
    - Berechnung meldet mit Antwort: Session geht weiter oder nicht

- Server: Erscheinungsformen (**Namen ändern sich, Probleme nicht!**)

- Transaktionssystem („OLTP“)
  - Datei-Server (NFS, ...)
  - RPC-Server („Client-Server-Betrieb“)
  - Datenbank-Server (SQL-Server, Sybase, ...)
  - Objekt-Server (CORBA)
  - Web-Server
  - Applikations-Server,
  - EJB Container
  - Web Services<sup>3</sup>, ...

---

3. Web services are **application functionalities** supporting **direct interaction** by responding to **service requests** based on open Internet Standards (SAP-Def.).

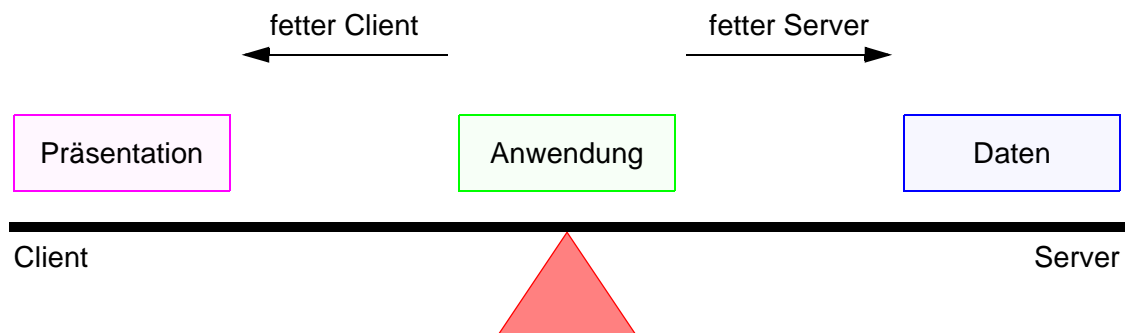
A Web service is a software application identified by a URL, whose interfaces and binding are capable of being **defined, described, and discovered by XML artifacts**, and supports direct interactions with other software applications using **XML-based messages via Internet-based protocols** (W3C-Definition).

Eingesetzte Technologien und Standards: XML, WSDL, UDDI, SOAP, HTTP

## Zweistufige C/S-Architekturen

- **Möglichkeit der Unterscheidung**

Wie ist die verteilte Anwendung dem Client und Server zugeordnet?  
(fat client vs. fat server)



- **Fette Clients**

- traditionelle Form von C/S, **Einfachheit**
- Größter Lastanteil fällt auf Client-Seite an
- Anwendungen: benutzerbezogene SW, Entscheidungsunterstützung
- Flexibilität bei der AW-Entwicklung und Nutzung von Front-End-Tools

- **Fette Server**

- sind leichter zu verwalten und zu installieren
- minimieren den Netzverkehr, indem sie mächtigere Dienste anbieten (**höhere Abstraktionen bei den Diensten**, Export von Prozeduren/Methoden)
- Client bietet GUI und kooperiert mit Server über RPC oder RMI

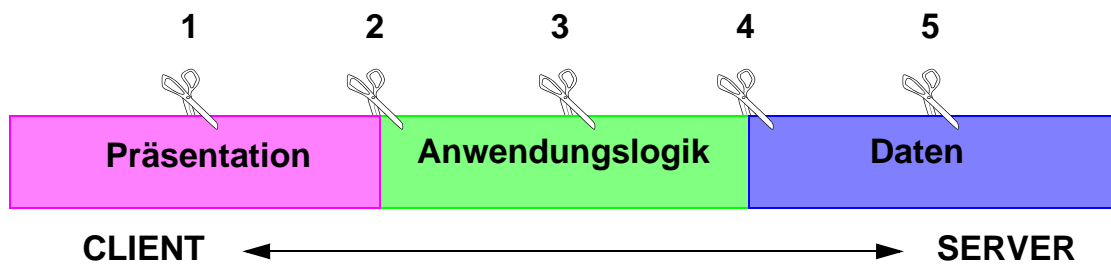
Â **Beide C/S-Modelle haben ihre Vorteile. Sie komplementieren sich und koexistieren oft in einer Anwendung**

## Zweistufige C/S-Architekturen (2)

- **Umsetzung von C/S-Architekturen**

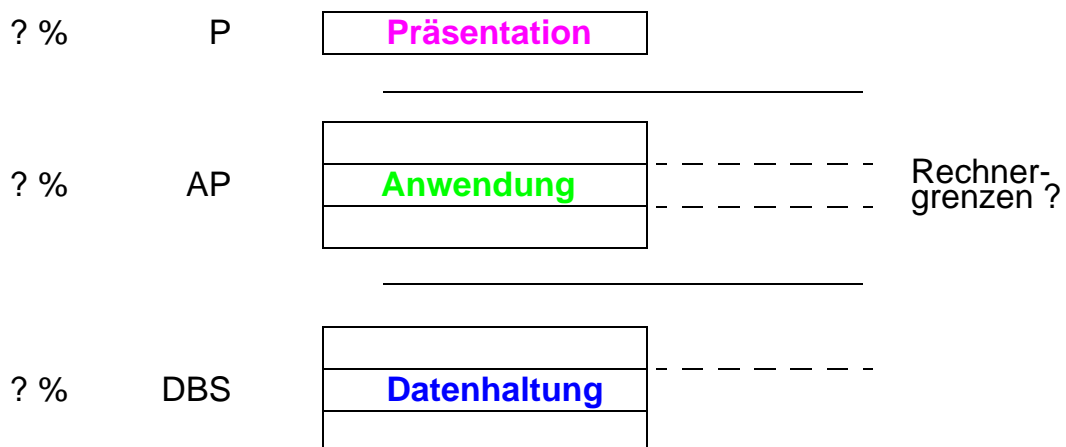
- C/S ist **nicht** an eine bestimmte HW- und SW-Konfiguration gebunden (z. B. Mainframe und PCs)
- Sie implizieren nicht bestimmte SW-Funktionen bei Client oder Server
- Welche Aspekte spielen bei der Umsetzung eine Rolle?

- **Fünf prinzipiell mögliche Trennlinien**



- **Mögliche C/S-Schnittstellen**

Aufwandsanteile  
zur Abwicklung  
einer kurzen TA

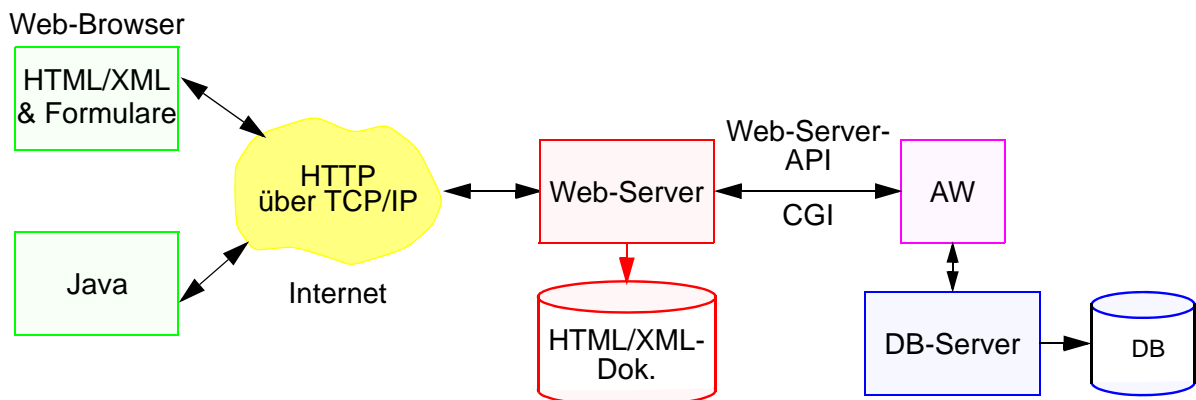




## Zweistufige C/S-Architekturen (3)

- **Web-Server**

- WWW ist die erste wirklich „intergalaktische“ C/S-Anwendung
- In der einfachsten Form kommunizieren Browser (*thin, portable, „universal“ clients*) mit Web-Servern (*superfat servers*)
- Kommunikation wird über ein **RPC-ähnliches Protokoll** durchgeführt: HTTP (*hypertext transfer protocol*)
- HTTP ist **zustandslos** und definiert eine einfache Menge von Anweisungen; Parameter werden als **Strings und ohne Typbindung** übertragen
- Es werden zunehmend mehr Formen der Interaktivität eingeführt
- DB-Anschluss erlaubt die Bereitstellung „**dynamischer**“ **Dokumente** (**4-stufiges C/S**)



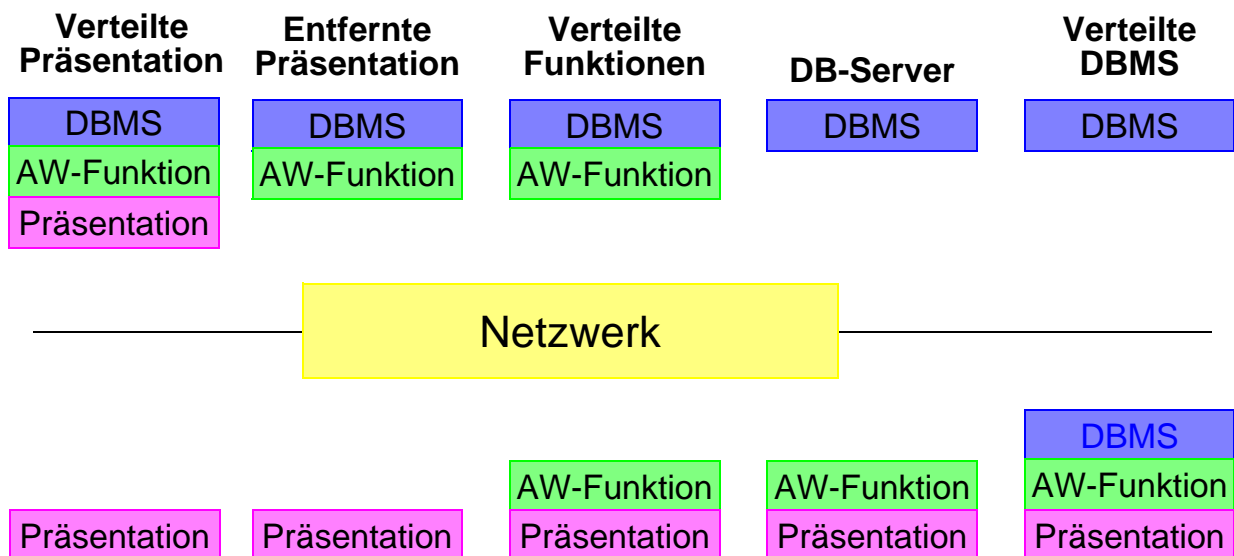
- Betrieb weist alle Merkmale eines TA-Systems auf (siehe Kap. 1)  
(Benutzungsart, TA-Typen, gemeinsame DBs, ...)

Â **Web-Server ist ein TA-System!** (eingeschränkte ACID-Eigenschaften!)

## Zweistufige C/S-Architekturen — Beispiele

- **Architekturvorschläge**  
(unterschiedlicher praktischer Relevanz)

### Server-Seite



### Client-Seite

- **Verteilte Präsentation** (Schnitt 1)  
(Bsp. X-Window-System)
  - X-Server läuft auf Rechner des Benutzers, verwaltet Fenster seiner Clients und ist für direkte Bildschirm-E/A zuständig
  - Er erhält Aufträge von einem oder mehreren X-Clients, die jeweils ihre eigenen Fenster steuern
  - Anwendungen fungieren als X-Clients

Bem.: Am Bildschirm kann mit mehreren AW gleichzeitig gearbeitet werden (Datenaustausch mit Cut-and-Paste)
- **Entfernte Präsentation** (Schnitt 2)
  - Aufgabenverteilung wie bei traditioneller Terminal-Host-Konfiguration
  - Beispiel: reine Terminalemulation auf einem PC

Â ebenso wie bei der verteilten Präsentation wird eine vergleichsweise geringe Entlastung der Server-Seite erzielt

## Zweistufige C/S-Architekturen — Beispiele (2)

- **Verteilte Funktionen** (Schnitt 3)

Kennzeichnend ist die Verteilung der Moduln der AW-Funktionen auf verschiedene Rechner (distributed function model).

Aufteilung der Moduln ist leistungskritisch; sie muss die Art der Datenzugriffe und ggf. Lokalität bei (wiederholten) Datenreferenzen berücksichtigen.

Bei kurzen Transaktionen empfiehlt sich folgende Aufteilung:

- **Server**

- gesamte Datenhaltung
- transaktionsbezogener Teil der Anwendung (gesamte Datenmanipulation, Commit)

- **Client**

- lokale Anwendungsverarbeitung
- Steuerung der graphischen Benutzeroberfläche

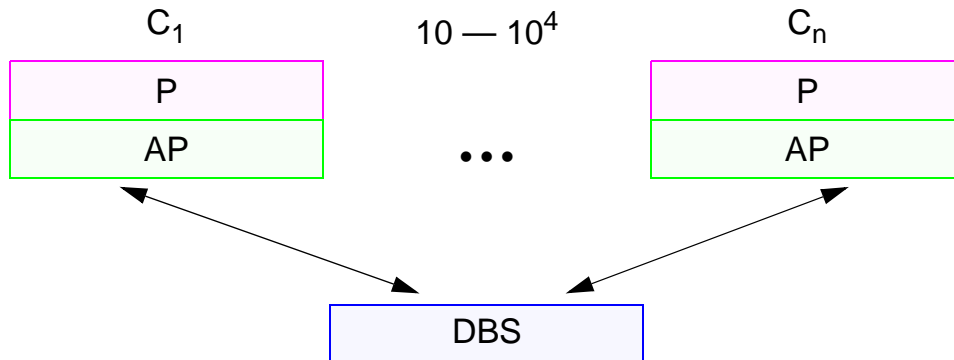
Â **Trend: neuentwickelte Anwendungssysteme basieren auf dieser kooperativen Verarbeitung“**

- **DB-Server und verteiltes DBMS** (Schnitt 4 und 5)

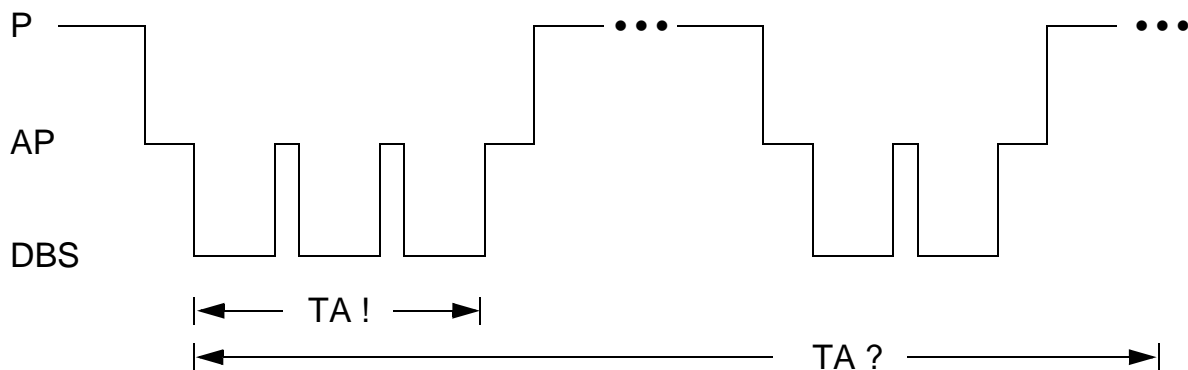
- Daten sind „entfernt“ und werden von einem dedizierten Datei- oder DB-Server verwaltet. Bei Verteilung der Daten ist ein verteiltes Dateisystem oder ein VDBS einzusetzen.
- Server-seitig wird keine AW-bezogene Verarbeitung durchgeführt
- Beispiel:  
Datei-Server speichert gemeinsame Daten für PC-Anwendungen

## Zweistufige C/S-Architekturen (4)

### 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

## Dreistufige C/S-Architekturen

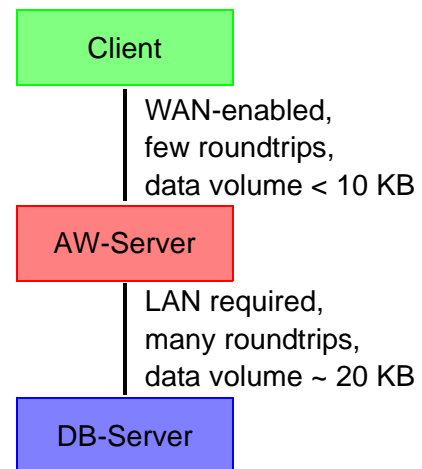
- **Bisher: 2-stufige C/S-Architekturen**

- Erfolgreicher Einsatz bei abteilungsbezogenen Anwendungen (begrenzte Reichweite, beschränkte Anzahl von Benutzern)

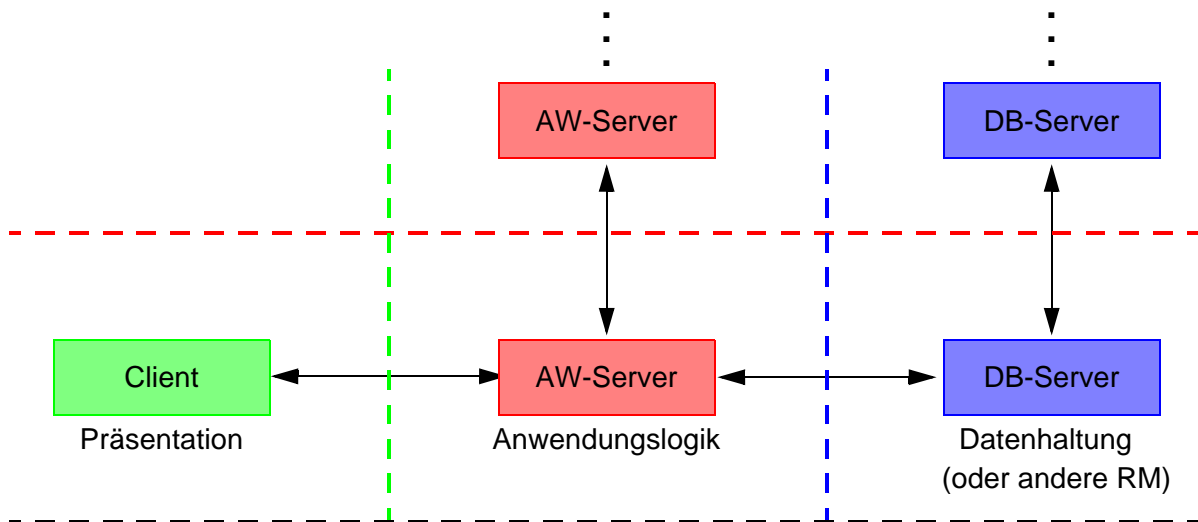
- **Praxiserfahrung**

- mangelnde Skalierbarkeit
- unzureichende Zuverlässigkeit bei komplexeren Anwendungen, insbesondere bei Anwendungskomponenten auf Client-Seite

⌘ **aber:** unternehmensweite C/S-Anwendungen und weltweite e-Commerce-Anwendungen gefordert



- **Abhilfe: 3-stufige C/S-Architekturen**



- Unterstützung von komplexen und **verteilten Anwendungen**
- Anwendungslogik (*middle tier*) läuft in eigenen Prozessen ab
- Leichte Konfigurierbarkeit der Prozesse („first-class citizens“)
- Verbesserte Skalierbarkeit, Sicherheit, Zuverlässigkeit, . . .

⌘ **heterogene Systeme: Standardisierung der Schnittstellen**

## Dreistufige C/S-Architekturen (2)

- **Komponentenumgebung** (*Framework*)

- Schnelle Entwicklung neuer Anwendungen durch API-Standardisierung (zusätzliche Funktionen, neue Clients, ...)
- Einsatz „vorgefertigter“ Komponenten; Vorteile der Nutzung von Standard-SW
- Einfacher und sicherer Daten- und Funktionszugriff
  - Komponenten kapseln die AW-Logik als Transaktionsprogramm (TAP)
  - Client sieht nur „abstrakte Dienste“, die er über ihren Namen (TAC) aufruft

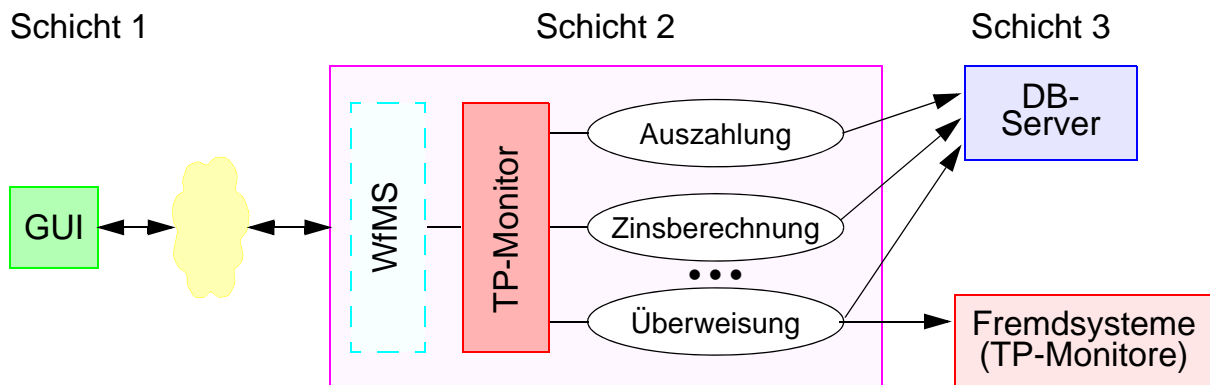
Â **Programm- und Datenunabhängigkeit**

- Vorkehrungen für Betrieb und Administration

Â **Wer liefert den „Klebstoff“?**

- **Dienste-basierte Applikation**

- Dienste sind Komponenten (TAPs), die Geschäftsfunktionen kapseln
- Applikation besteht aus einer (relativ kleinen) Menge solcher Dienste
- Einsatz (optional) von WfMS zur Steuerung/Verknüpfung von Diensten



- **zusätzliche Programme („Middleware“)**

- TP-Monitor
- WfMS
- Container, Dispatcher, ...

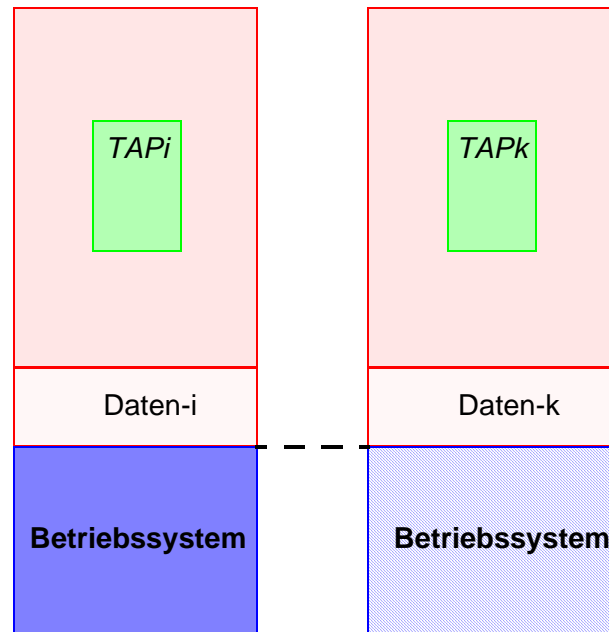
Â **TP-Monitor dirigiert und kontrolliert!**

## Dirigent und Kontrolleur - Wie kommt man zu einer effizienten Realisierung oder welche Aufgaben sind zu lösen?

- **Beobachtung**

- „Java (EJB, ...) ist ja so langsam!“
- „Aber CICS ist inzwischen ganz schön schnell!“

Â „Was ist CICS?“ ;)



- **Wie macht man Server effizient?**

- Tausende von Anfragen
- Tausende von Kontexten

- **Prozess-Strukturen im Betriebssystem (BS)**

- Verwaltungsaufwand?
- Prozess<sup>4</sup> wird (klassischerweise) durch zwei Merkmale charakterisiert:
  - **Ressourcenbesitz:**  
Virtueller Adressraum zur Aufnahme des Prozessabbilds, Zuordnung von Ressourcen, Ausübung einer Schutzfunktion
  - **Ablaufplanung/Ausführung:**  
Er hat Ausführungszustand (aktiv, bereit, usw.), eine Zuteilungspriorität und ist Einheit der Ablaufplanung

Â Wie wird **kurzfristige, verteilte, klein-granulare** BM-Zuteilung erreicht?

---

4. (Virtual memory) process is domain of **addressing, protection, scheduling!** Bei einigen BS hat die Unterscheidung der Konzepte Ressourcenbesitz und Ausführung zur Entwicklung des Thread-Konzeptes geführt. Für die Unterscheidung der beiden Merkmale wird die Einheit der Zuteilung oft als Thread oder Leichtgewichtsprozess und die Einheit des Ressourcenbesitzes als Prozess bezeichnet

# Server-Programmierung

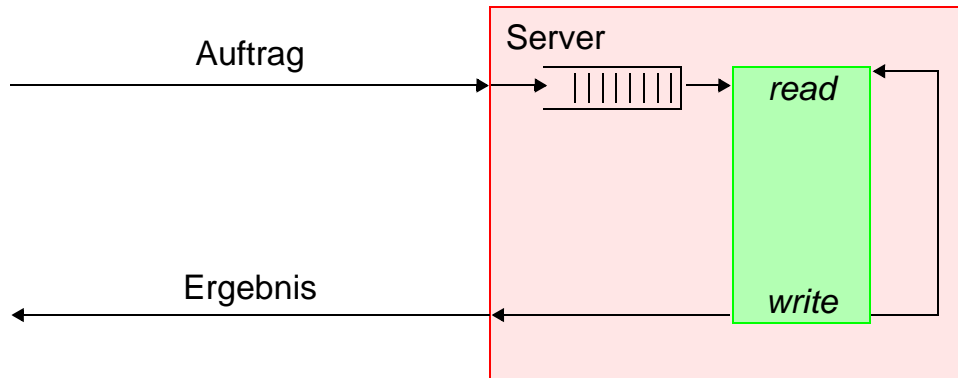
- **Programmierung der Anwendungen**
  - so einfach wie möglich
  - transaktionale Zusicherungen!
- **Wie schreibt man die Programme, die**
  - einen Auftrag entgegennehmen,
  - die Berechnung durchführen und
  - die Antwort mit dem Ergebnis zurücksenden?
- **Zugeständnisse an die Effizienz**
  - **Kontext anders behandeln** als lokale Variablen
  - **mehrere kooperierende Programme** schreiben für eine Aufgabe
  - **Mehrfachausführung** des Programms berücksichtigen („Threads“ erzeugen)
  - **Synchronisation von Zugriffen** auf gemeinsame Daten
- **Â Zerlegung, Vorplanung, Minimierung der BM-Zuteilung!**
- **oder nichts von alledem?**
  - wenn die Middleware das alles übernimmt ...



## Server-Programmierung (2)

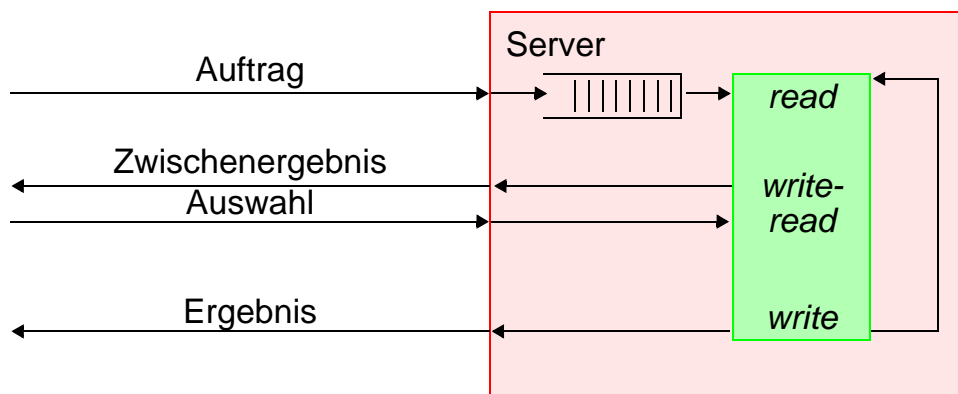
### • Ausführungsmodell 1

- Wie führt man die Programme aus (einfachste Lösung)?:



Â Achtung: Prozess-Struktur wird hier nicht berücksichtigt (Server ~ Prozess)!

- **read und write** charakterisieren Nachrichten vom/zum Client
- Berücksichtigung des Kontextes:



### • Einfachste Art der Programmierung

- immer nur ein Auftrag zur Zeit
- Kontext direkt im Programm

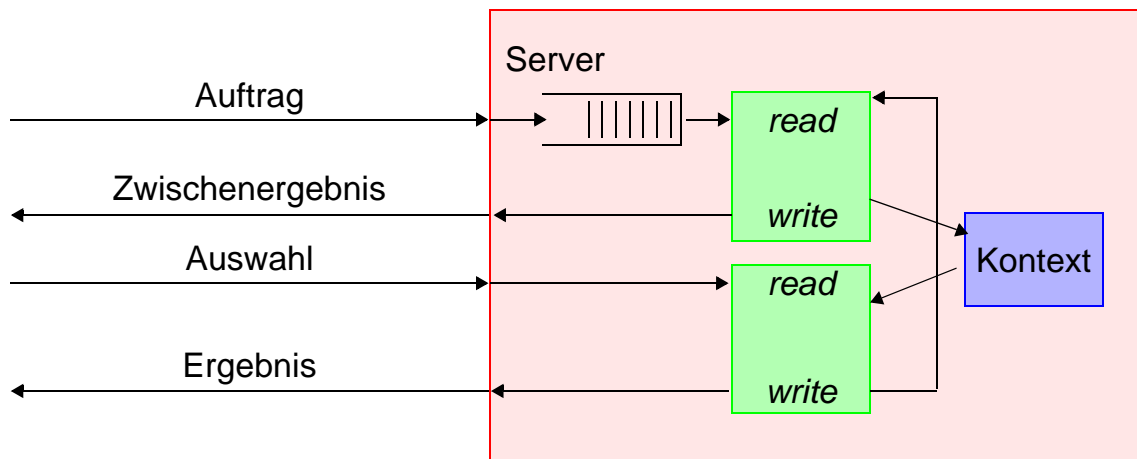
### • Ineffizient!

- Wartezeit auf nächste Eingabe („Auswahl“) ungenutzt
- Wartezeiten während der Bearbeitung
  - Ein-/Ausgabe
  - Aufträge an andere Server (!) ebenfalls ungenutzt

## Server-Programmierung (3)

### • Ausführungsmodell 2

- Kontext separieren und Programme verketteten
- explizite Verwaltung der Kontexte:

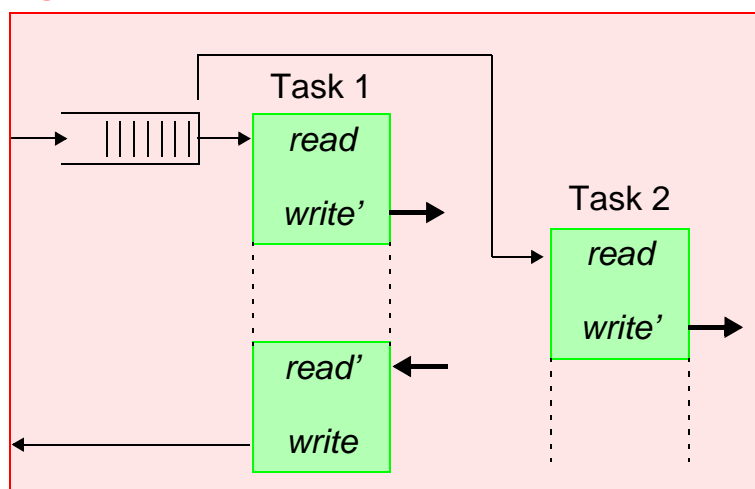


Â Programmierung wird aufwendiger!

Welche Anforderungen ergeben sich für die Kontextverwaltung?

### • Ausführungsmodell 3

- zusätzlich: Wartezeiten in der Berechnung nutzen
- **read' und write'** charakterisieren Nachrichten von/zum weiteren Servern
- **Multi-Tasking:**

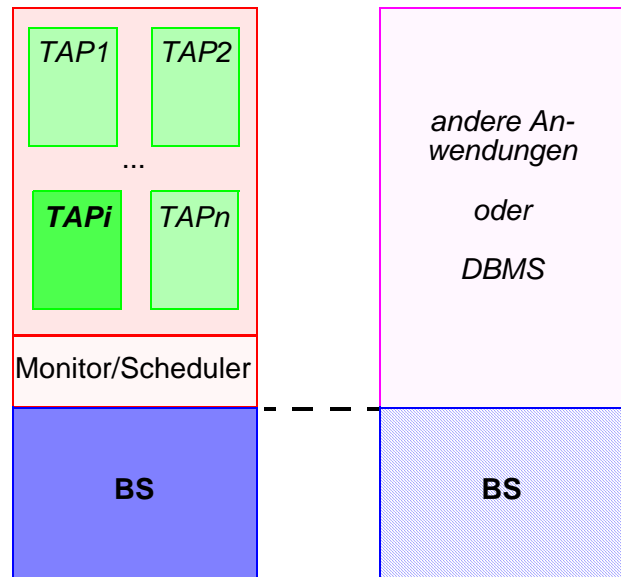


Â Wer programmiert das?

## Prozesse

- **Ausführungsmodell**

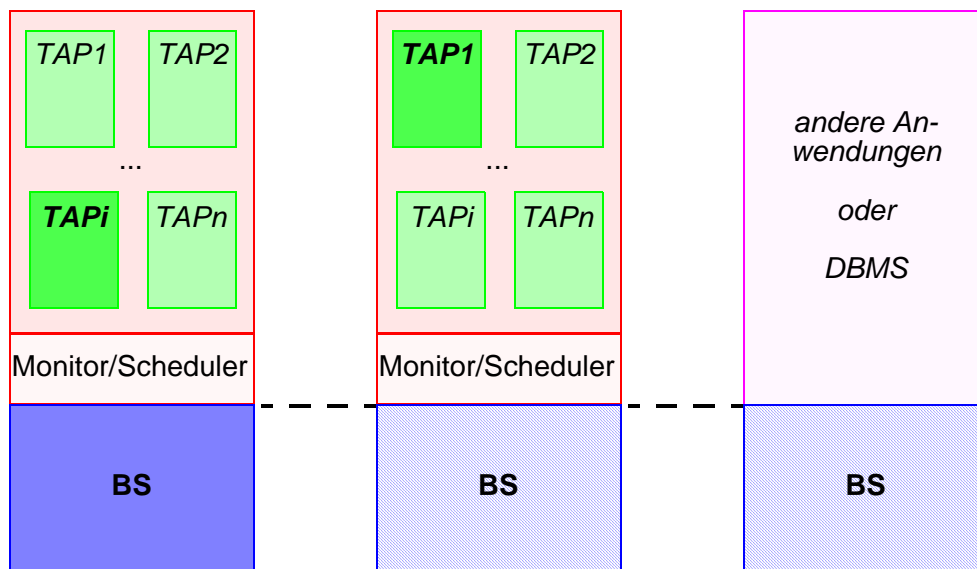
- soweit **ein Prozess pro Server**
  - auf Prozessebene überschaubar
  - einfache interne Programmstrukturen (bei Single-Tasking)
  - braucht hohe Priorität
  - bei Seitenfehler oder Zeitscheibenende wird der gesamte Server inaktiv
- ein Prozess pro Auftrag?



- Verwaltungsaufwand zu hoch,  
**deshalb Multi-Process und/oder Multi-Tasking (Multi-Threading)!**

- **Mehrere Prozesse pro Server (10 — 100)**

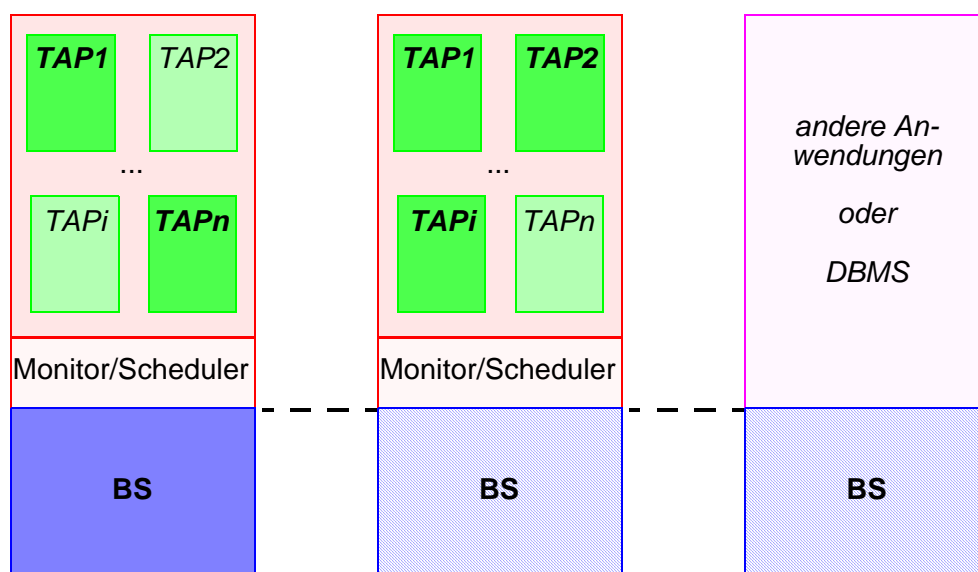
- skalierbar
- Prozesskommunikation: Kontexte müssen „wandern“
- interne Programmstrukturen einfach
- Verwaltungsaufwand mittelmäßig



## Prozesse (2)

- **Ausführungsmodell von Threads**

- Ablaufeinheit innerhalb eines Prozesses (leichtgewichtiger Prozess)
  - Trennung von **Ressourcenbesitz** und **Ablaufplanung/Ausführung**
  - Umschalten
    - bei Ein-/Ausgabe und Aufruf anderer Server,
    - evtl. auch bei Seitenfehler, aber
    - nicht bei Zeitscheibenablauf (Merkmal Ausführung)
  - gleicher Adressraum (Merkmal Ressourcenbesitz)
  - effizienter Datenaustausch (Kontexte), TA-Fortsetzung kann durch Nutzung des gleichen Prozesses unterstützt werden
  - Synchronisation in vielen Fällen durch Scheduling
  - Verwalten deutlich einfacher als beim Prozess, jedoch komplexe interne Programmstrukturen (bei Multi-Tasking)
  - Erzeugen und Vernichten oder Thread-Pool bereithalten
- **Single-Process oder Multi-Process**
- ein Thread oder mehrere Threads pro Prozess, hier Verwaltung und Scheduling im Benutzeradressraum (**user-space threads**):



## Prozesse (3)

- **Ausführungsmodell von Threads**

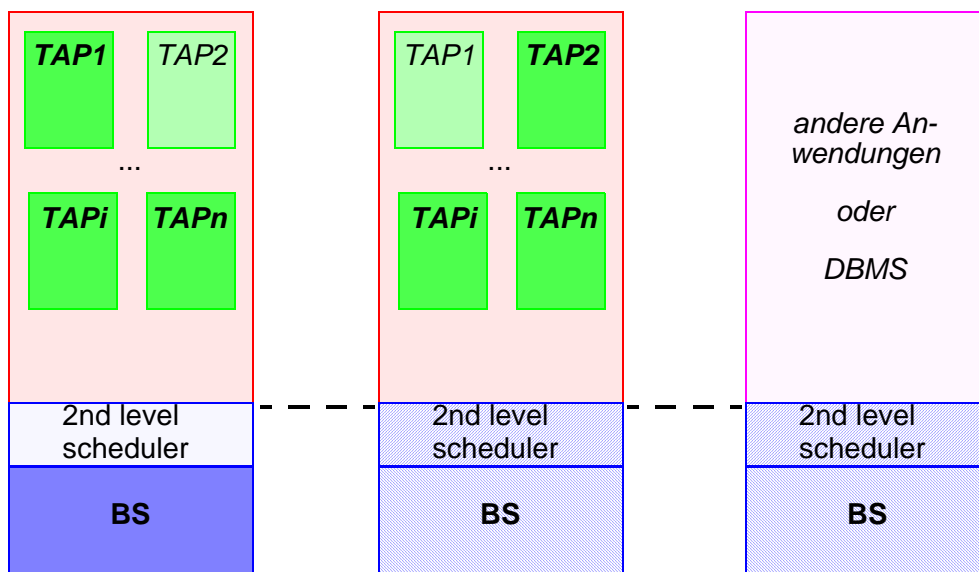
- **Scheduling zweiter Stufe**

- BS oder Subsystem
- Welche Unterschiede ergeben sich zum Modell der **user-space threads**?
- Wann ist Umschalten möglich?

- **Single-Process oder Multi-Process**

- Ablauf von einem Thread oder mehreren Threads pro Prozess, hier Verwaltung und Scheduling im Betriebssystem (**kernel-supported threads**):

- bessere Bezeichnung wäre: **Task<sup>5</sup>** (wird im folgenden benutzt)

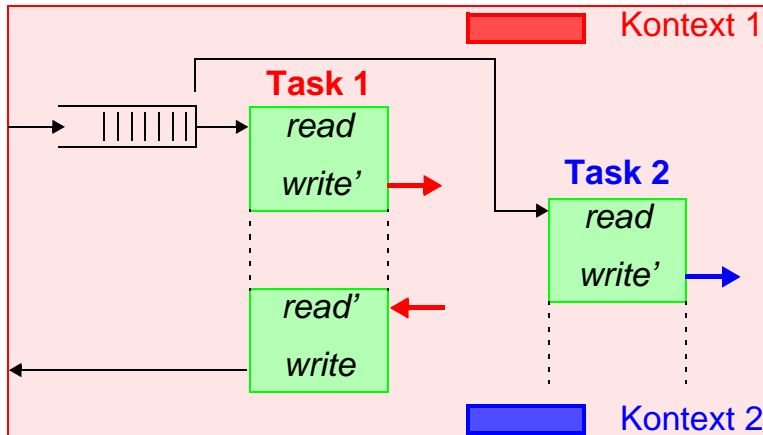


5. Bei Threads erfolgt ihre Freigabe typischerweise explizit durch spezielle Anweisungen im abzuwickelnden Programm. Da TAPs unabhängig von ihrer Ablaufumgebung geschrieben werden, muss die Thread-Freigabe und damit das Scheduling auf zweiter Stufe auf eine „semantische“ Art erfolgen (z. B. write-Anweisung). Deshalb bezeichnen wir die Einheiten der Ausführung innerhalb eines Prozesses als Tasks.

# Programmverwaltung

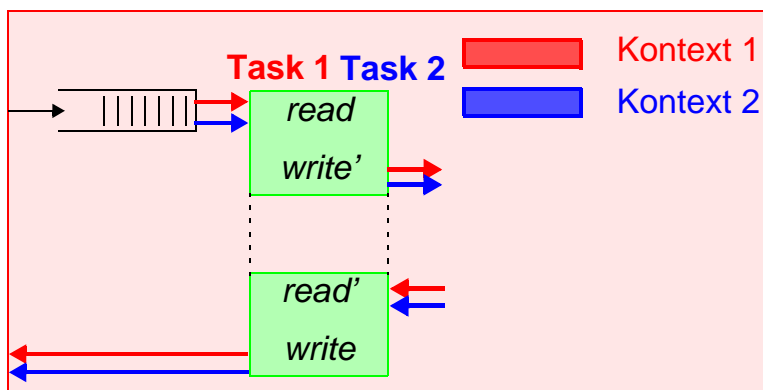
- **Ausführungsmodelle für Programme**

- Kombination mit Ausführungsmodell 3



Annahme:  
Task1 und Task2  
führen TAP  
Auszahlung aus

- Mehrere Tasks führen das gleiche Programm aus
  - zur gleichen Zeit (z. B. Kontenbuchung)
  - Jeder erzeugt einen „**Thread**“ durch ein Programm
- einfachste Variante
  - Jeder Task erhält **eigene Kopie** („**Single-Threading**“)
  - hoher Speicherplatzbedarf (Seitenwechselrate steigt)
- **besser:**
  - **Mehrere Tasks** benutzen **gemeinsame Kopie** („**Multi-Threading**“)
  - geht nur unter bestimmten Voraussetzungen:
    - **alle Variablen** im Kontext, Trennung von Daten- und Code-Segment
    - keine Code-Modifikation (Code ist „**reentrant**“ oder „**ablaufinvariant**“)
  - Aufgabe des Compilers

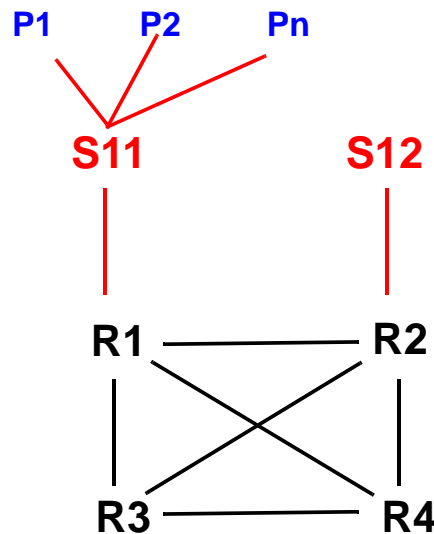


Annahme:  
Task1 und Task2  
führen TAP  
Auszahlung aus

## Zuordnungen — Namen

- **Server-Klasse**

- n:m-Beziehung mit Netzknoten
- 1:n-Beziehung mit Servern
- Eine Server-Klasse auf mehreren Netzknoten bedeutet: Server sind austauschbar  
     $\hat{=}$  **Lastbalancierung**
- kann durch URL adressiert werden

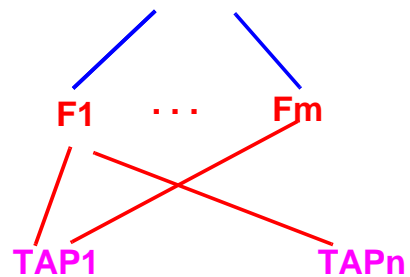


- **Funktion (Dienst, Service)**

- n:1-Beziehung mit Server-Klasse

- **Server**

- 1:n-Beziehung mit Prozessen
- Prozess bietet Single-Tasking oder Multi-Tasking (wird manchmal auch als Threading bezeichnet)
- Task führt TAP aus und erzeugt dabei einen Thread durch das Programm



- **Programm**

- n:m-Beziehung mit Funktionen
- extern nicht sichtbar
- interne Zuordnung von Programmen zu Funktionen oder auch nur Teilen (Abschnitten) davon
- Abwicklung durch Tasks unter **Single-Threading oder Multi-Threading**

- **Zahlreiche externe Dienste verfügbar**

- X.500, LDAP, Jini, UDDI, ...

# Transaktionen

- **Transaktionale Zusicherungen**

- bisher: nur ACID-Garantie für DB-Daten
- Fehler und Ausfälle berücksichtigen (nicht nur im DB-Server!)
  - Berechnung im Server begonnen, aber noch nicht vollständig
  - Client kann diesen Zwischenzustand weder verstehen noch verlassen
  - manueller Eingriff durch Server-Administration?
- **„Transaktion“** als Dienstleistung des Servers (Ressourcen-Mgr) (und der Middleware)
  - Alles oder nichts — vom System gewährleistet:
    - unvollständige Berechnungen rückgängig machen, so dass der Client den Auftrag wiederholen kann
    - vollständige Berechnungen wiederherstellen
  - nicht der Programmierung überlassen, sondern **in der Middleware anbieten**
- Beispiel:
  - EJB als Server-seitige Komponententechnik erlaubt die Erstellung von (J2EE)-Anwendungen **unabhängig von Systemdiensten** wie Transaktionsverwaltung und persistente Datenhaltung.
  - Dienste werden **deklarativ spezifiziert** und auf Wunsch von der **Laufzeitumgebung** (sog. EJB Container) **integriert** (Deployment)

Â Middleware erlaubt Trennung von Anwendungslogik und Infrastruktur



## Resümee zu Server-Strukturen und -Techniken

- **Ziel**

- Einsatz aller genannten Techniken **zur Steigerung der Effizienz**
  - Programmverkettung
  - Multi-Tasking
  - Multi-Threading
  - ablaufinvariante Programme
- **Programmierung dennoch so einfach wie möglich**
  - einen Auftrag bearbeiten
  - Kontext = lokale Variablen
  - Isolierung: gleichzeitig laufende Aktivitäten nicht sichtbar
  - Ablaufinvarianz des Codes ist Sache des Compilers
- Aufgabe der Middleware
  - insbesondere: **Strategien änderbar** (optimierbar)

- **Allgemeine Nutzung**

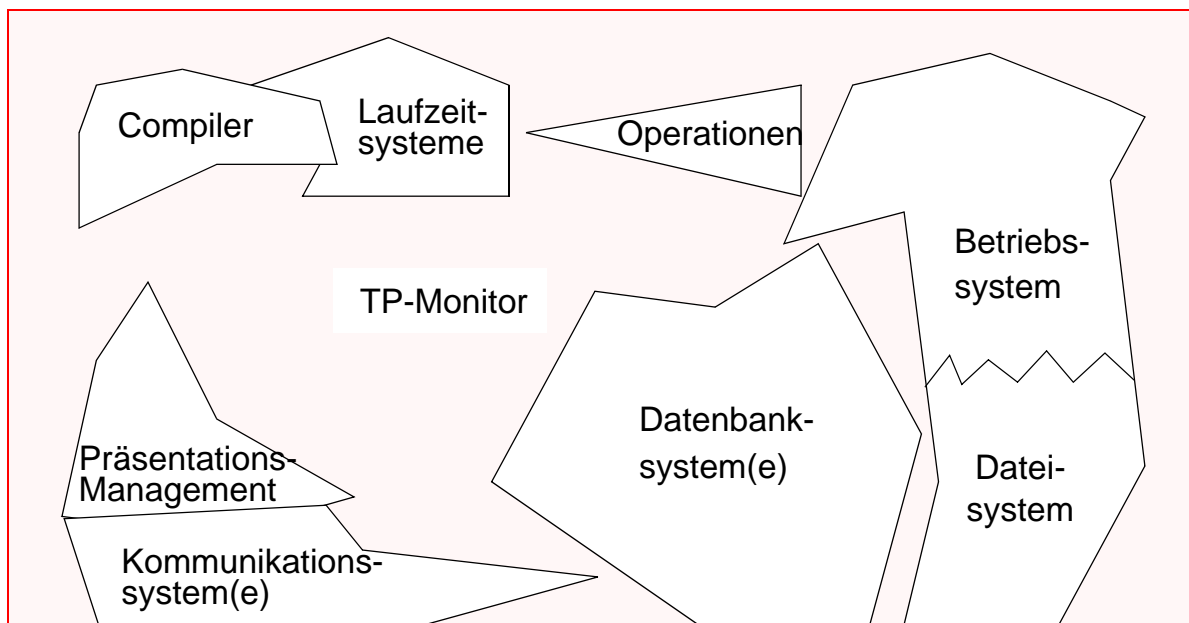
- Sie sind einsetzbar für:
  - **Transaktionssysteme** (Flug-, Konto-Buchung, SAP/R3-AW)
  - Datei-Server
  - RPC-Server („Client-Server-Betrieb“)
  - **Datenbank-Server**
  - Web-Server
  - **Applikations-Server**
  - EJB Container
  - **Web Services**
  - ...
- und werden dort (teilweise) eingesetzt
- **Bedarf an Leistungssteigerung ist ungebrochen**
- Trotzdem: **Trennung von Anwendungslogik und Infrastruktur!**

## Was sind TP-Monitore?

- **TP-Monitore (Transaction Processing Monitor)**

- bieten schon seit ~ 1970 auf Mainframes robuste Laufzeitumgebungen für große OLTP-Anwendungen (*on-line transaction processing*)
- liefern den „**Klebstoff**“ für das Zusammenwirken vieler Komponenten, Betriebsmittel, Protokolle usw. bei der TA-Abwicklung
- realisieren und optimieren Funktionen, die **von BS typischerweise nur sehr schlecht** oder gar nicht unterstützt werden
- verwalten Prozesse und starten/überwachen TAPs. Sie erlauben die **Integration unabhängiger Dienste** und ihre Abwicklung als TAs

- **TP-Monitor<sup>6</sup> und zugeordnete Systemkomponenten**



Der TP-Monitor integriert verschiedenartige Systemkomponenten, um gleichförmige Schnittstelle für Anwendungen und Operationen **mit demselben Verhalten im Fehlerfall** (*failure semantics*) zu bieten.

---

6. "In a contest for the least well-defined software term, TP-Monitor would be a tough contender" (J. Gray)

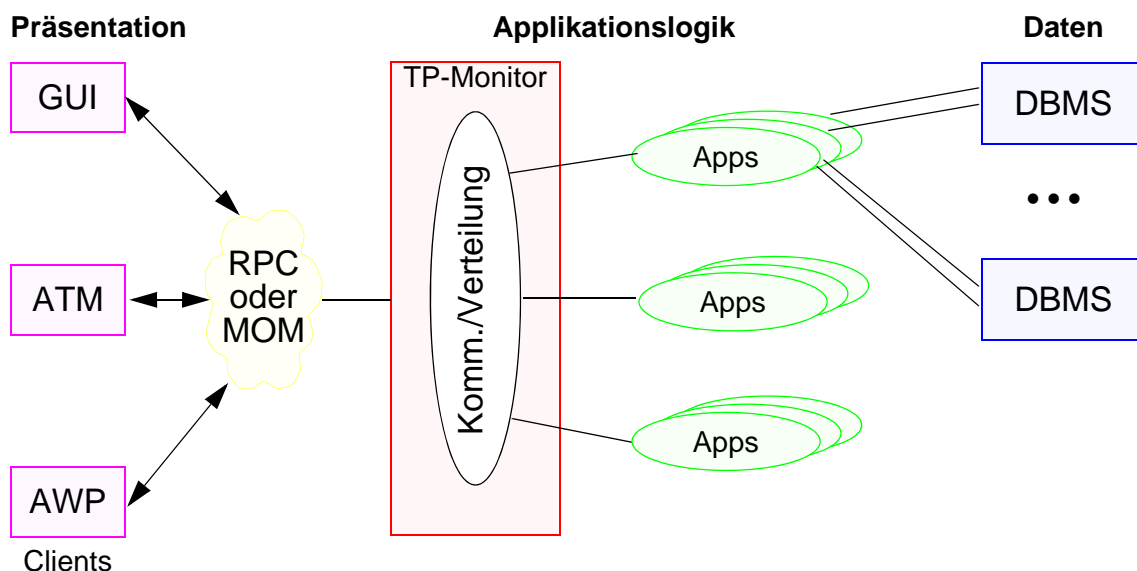
## Was sind TP-Monitore? (2)

- **Neue TP-Monitor-Generation<sup>7</sup>**

- ist „offen“ (standardisierte Schnittstellen zu Plattformen, Kommunikation, Benutzern)
- lässt sich auf mehreren BS (Rechnerplattformen) einsetzen
- ist Client-/Server-basiert

- **Was tun TP-Monitore in C/S-Umgebungen?**

- Sie verwalten **Transaktionen** und koordinieren ihren Ablauf im System
- Sie kontrollieren die **Kommunikationsflüsse** zwischen Tausenden von Clients und Hunderten von Servern
- Sie ergreifen **Maßnahmen zur Lastbalancierung** und verbessern das Leistungsverhalten
- Sie sind für den **Wiederanlauf** nach Fehlern (*Restart*) verantwortlich
- Sie stellen sicher, dass Transaktionen die **ACID-Eigenschaften** einhalten



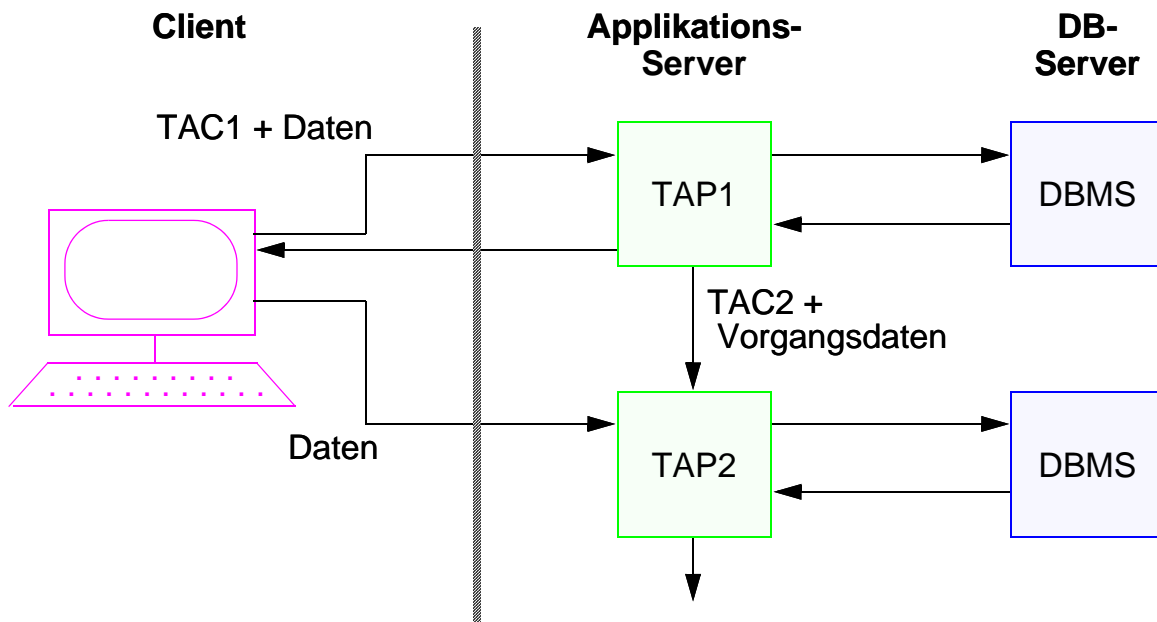
7. TP-Monitors are like the Rolling Stones – been around for a long time, but still drawing large crowds (David Linthicum)

## Was sind TP-Monitore? (3)

### • Wie sind TP-Monitor-basierte Anwendungen aufgebaut?

- TP-Monitor bietet ein **vordefiniertes Framework** für Entwicklung, Betrieb und Administration von C/S-Applikationen
- Auf der **Client-Seite** sind Tools zur Entwicklung/Definition von GUIs (Fenster, Formulare, Masken usw.) verfügbar
- Auf der **Server-Seite** lassen sich modulare, wiederverwendbare Dienste entwickeln, die von Ressourcen-Mgr (RM) gekapselt werden.
- TP-Monitore stellen **allgemeine Server-Klassen** zur Verfügung, für die Prozesse erzeugt werden können, in denen die Dienste der Applikation abgewickelt werden (ein oder mehrere Prozesse pro Server-Klasse)
- Sie führen auf Server-Seite einen **ereignisgetriebenen Programmierstil** ein (TACs initiieren TAPs)

### • Prinzipieller Ablauf eines Vorgangs



## Was sind TP-Monitore? (4)

- Was ist nun genau ein TP-Monitor?

- Er lässt sich als BS für transaktionsgeschützte Applikationen auffassen
- Er ist ein Framework für Applikations-Server

- **Er erledigt drei Aufgaben extrem gut:**

- **Prozessverwaltung:**

Sie schließt das Starten von Server-Prozessen, das Initiieren von TAPs, das Kontrollieren ihres Ablaufs und die Lastbalancierung ein

- **Transaktionsverwaltung<sup>8</sup>:**

Sie garantiert die ACID-Eigenschaften von allen Programmen, die unter ihrem „Schutz“ ablaufen.

Dazu muss sie im Normalbetrieb Logging (z. B. Protokollieren von Nachrichten) durchführen, um im Fehlerfall Recovery-Maßnahmen ergreifen zu können

- **C/S-Kommunikationsverwaltung:**

Es ist Client/Server- und Server/Server-Kommunikation zu unterstützen. Sie erlaubt den Client- und Server-Programmen, die an einer Anwendung beteiligten Dienste und Komponenten auf verschiedene Weise aufzurufen: RPCs, Konversationen, asynchrone Nachrichten über persistente Warteschlangen (MOM: *message-oriented middleware*), Broadcasts, ...

- **Er kann als sehr komplexer RPC-Mechanismus aufgefasst werden:**

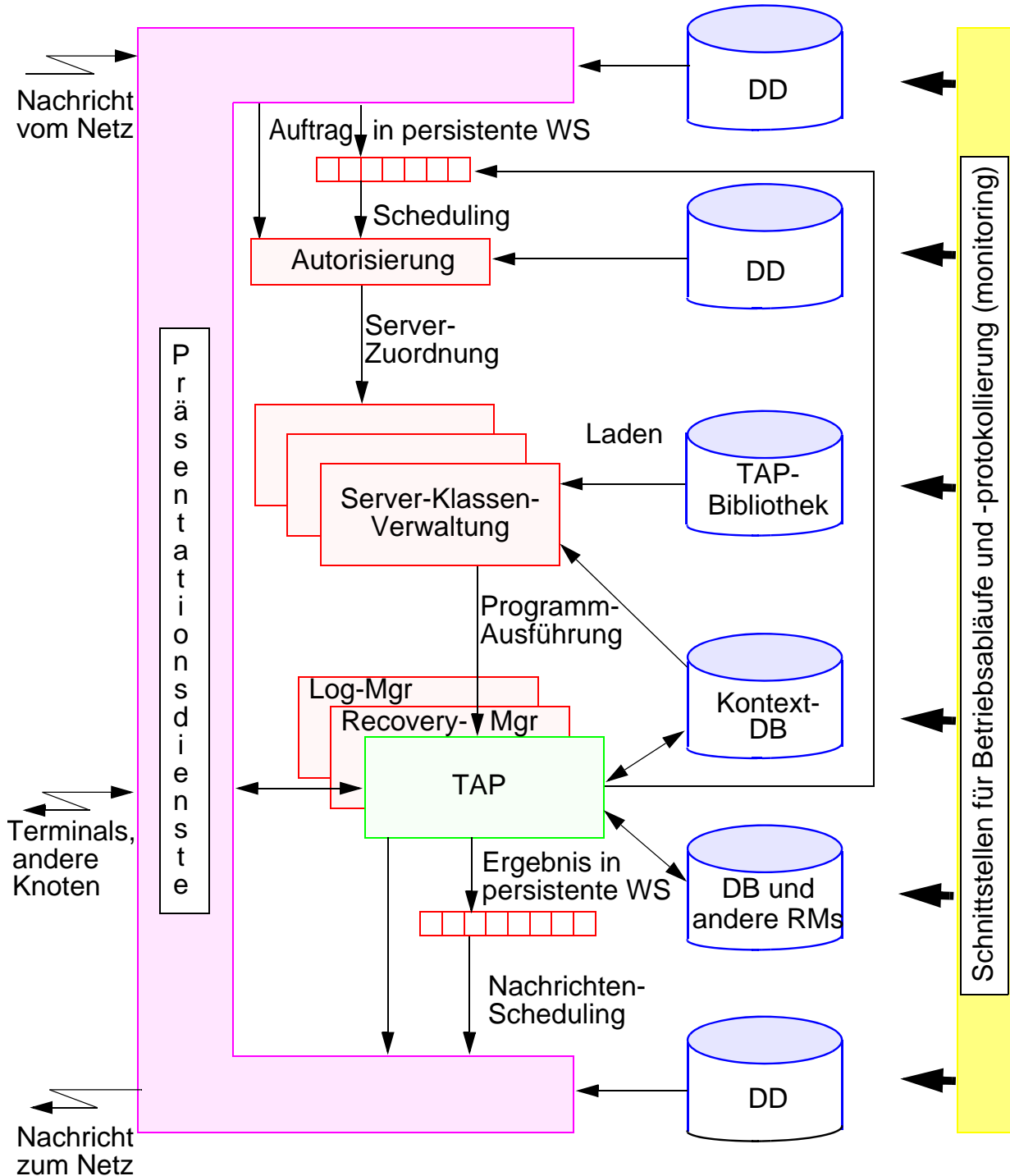
Transaktionaler RPC (TRPC) an Serverklasse, zusätzlich Auswahl, Ressourcenallokation, Start, Scheduling, Kontrolle (Sicherheit, Ablauf) der Funktion

---

8. The Standish Group estimates that the world electronically processes 68 million transactions every second. 53 million of the 68 million use a TP Monitor (Jim Johnson, Oct. 1998)

## Struktur eines TP-Monitors

- **Kontrollfluss durch einen TP-Monitor (vereinfacht)**



Â Diagramm repräsentiert die wichtigsten Grundfunktionen für TAP-Verwaltung und -Ausführung

## Struktur eines TP-Monitors (2)

- **Präsentationsdienste**

- bilden die Schnittstellen zwischen dem TAP und den E/A-Geräten
- bieten Geräteunabhängigkeit (Terminaltyp, Formatkontrolle, *Scrolling*) und Kommunikationsprotokollunabhängigkeit

Â wegen vieler verschiedener Präsentationsdienste (Window-Protokolle, Graphikstandards) **implementieren oft eigene RM diese Dienste**

- **Warteschlangenverwaltung**

- WS-Dienste müssen transaktionsorientiert sein: Auftrag in WS wird genau einmal ausgeführt ('exactly once')
- Nur bei Commit der Server-TA kommt Ergebnis in Ausgabe-WS
- Abhängig von Anwendung und Inhalt der Nachricht können für Auslieferung verschiedene Zusicherungen vereinbart werden: 'at least once', 'at most once' oder 'exactly once'

Â **WS-Mgr ist oft als RM implementiert, der zum TP-Monitor gehört**

- **Server-Klassen-Verwaltung**

- Zuständig dafür, dass für jedes TAP eine Server-Klasse definiert und aktiv ist
- Aktivierung der Server entweder 'by default' oder 'on demand'
- Aufgaben: Erzeugen von Prozessen und WS, Laden der TAP-Codes, Besorgen der Zugriffsrechte für die WS-Zugriffe, Festlegen der Server-Prioritäten u.a.

Â Aufgaben fallen auch in die Verantwortlichkeit der Lastbalancierung. Sie sind deshalb in enger Kooperation zu lösen.

Â Funktionen wie Prozesserzeugung und TAP-Ausführung werden vom BS zur Verfügung gestellt.

## Struktur eines TP-Monitors (3)

- **Scheduling von Aufträgen**

- die am **häufigsten angeforderte Funktion** des TP-Monitors
- Bestimmung des angeforderten Dienstes:  
lokal oder entfernter Knoten.
- Weiterleitung des Auftrags an den Server.

- **Autorisierung der Aufträge**

- Teil der systemweiten Sicherheitsvorkehrungen
- Spektrum der Lösungen: von einfacher, statischer Autorisierung bis zu wertabhängiger, dynamischer Autorisierung
  - Â Wegen spezifischer Betriebscharakteristika von TA-Systemen ist **dynamische Autorisierung für individuelle Aufträge sehr wichtig**

- **Kontextverwaltung**

- 1. Speicherung von **Verarbeitungskontexten, die über TA-Grenzen** gehalten werden sollen (Mehr-TA-Vorgänge)
  - Â **Kontext-DB hat alle ACID-Eigenschaften** und könnte durch ein SQL-DBS implementiert werden
- 2. Unterstützung der **Kontextnutzung innerhalb einer TA:**  
Weitergabe von Zwischenergebnissen einer TA über Nachricht beim Server-Aufruf zu aufwendig; Kontextverwaltung speichert die Daten zwischen und erlaubt nachfolgende Server Zugriff auf die Daten
  - Â **Diese Zugriffe erfolgen innerhalb einer TA;** deshalb sind keine persistenten Kontexte erforderlich



## Struktur eines TP-Monitors (4)

- **Metadatenverwaltung**

Annahme: globales Repository (DD, Data Dictionary)

Â Je mehr der TP-Monitor tun soll, desto genauer/umfangreicher müssen die Metadaten sein

- **Metadaten-Übersicht:** Beschreibung der

- zum verteilten TA-System gehörigen Knoten: Namen, Adressen, ...
- lokalen Komponenten der TA-Dienste wie Log-Mgr, Recovery-Mgr, Kommunikations-Mgr, ...
- Geräte und HW-Komponenten, die der TP-Monitor kennen muss, wie Terminals, Controller, physische Verbindungen, ...
- TAPs und RMs, die am betreffenden Knoten installiert sind
- Autorisierungsinformation
  - Zugriffskontrolllisten für TAPs und RMs
  - über die dem System bekannten Benutzer
  - Autorisierungs-Codes (Passwörter), Sicherheitsprofile, ...
- Konfigurationsdaten
  - für Server-Klassen über Prozesse, Tasks, Prioritäten
  - über Betriebs- und Administrations-Schnittstellen
  - für Wiederanlauf und spezielle Abläufe (Restart-Reihenfolge der RMs)

Â Inhalt diese Kataloge wird gewartet und aktualisiert über System-TAs.  
Beim Restart wird die hier hinterlegte Konfiguration „hochgefahren“

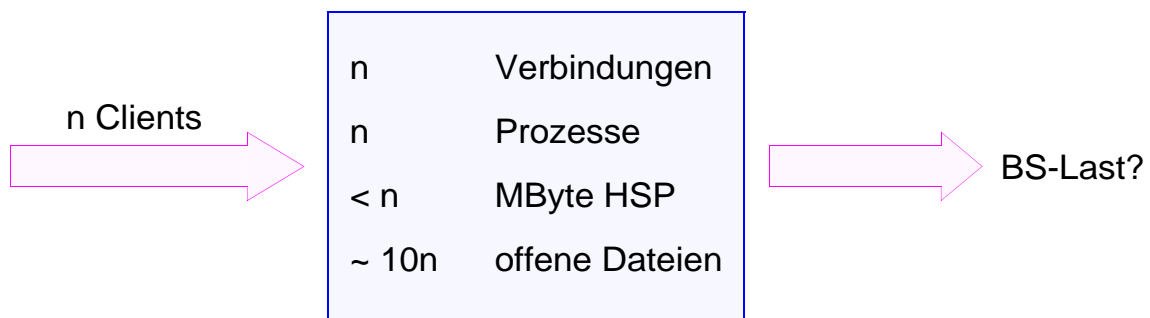
## Ablauf in dreistufigen C/S-Architekturen

- **Typischer BM-Bedarf pro Client auf einem Server**

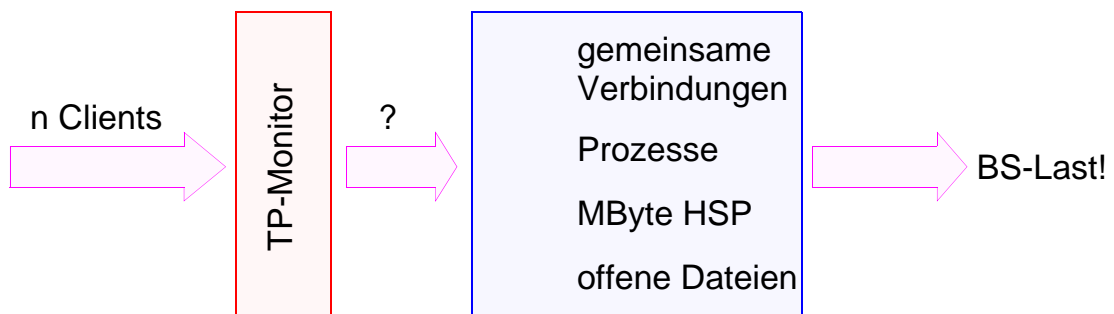
- eine Kommunikationsverbindung
- 0.5 — 1 MByte Hauptspeicher (RAM)
- 1 oder 2 Prozesse
- ~ 10 offene Dateikontrollblöcke (file handle)

Â Soll jeder Client seine eigenen BM statisch zugeordnet bekommen (virtueller Prozessor)?

- **Herkömmliche BS-Abbildung**



- **Einsatz eines TP-Monitors<sup>9</sup>**



9. "TP-Monitor: The 3-Tier Workhorse" (Jeri Edwards)

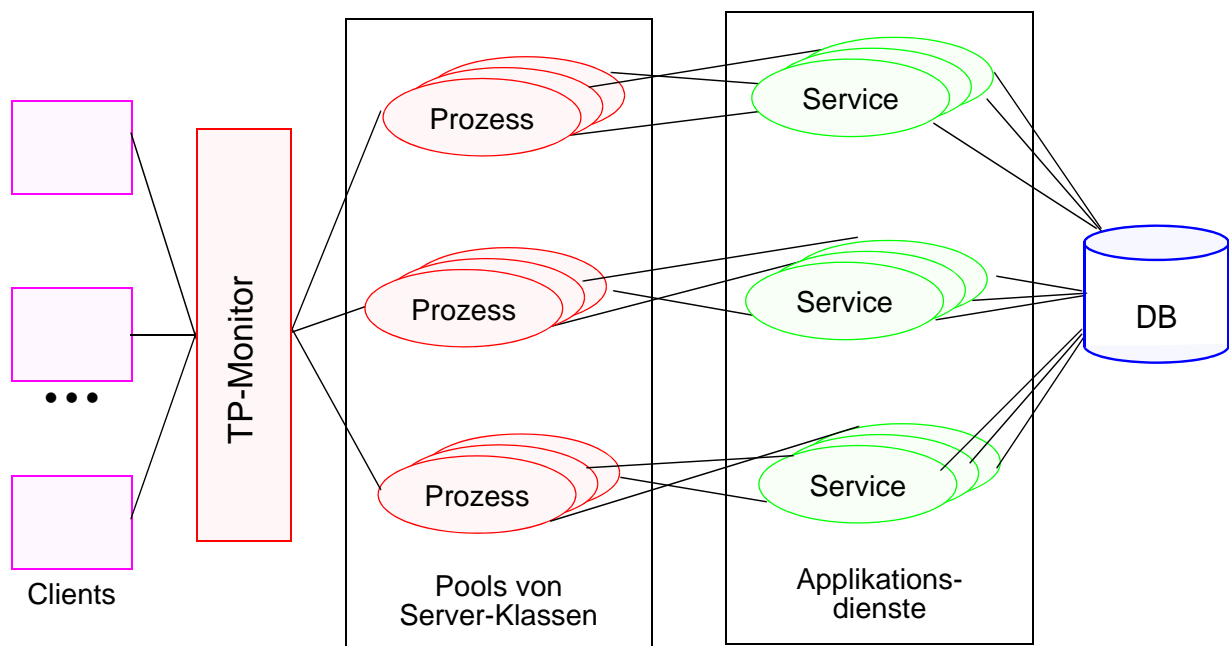
## Ablauf in dreistufigen C/S-Architekturen (2)

- **BM-Zuteilung erfolgt auftragsbezogen**

- OLTP-Applikation besteht aus einer Menge von Diensten (TAPs)
- Server-Klasse besteht aus Pool von (statisch erzeugten) Prozessen (mit Tasks/Threads), welche vorab geladene TAPs abwickeln können. TP-Monitor kann dynamisch neue Prozesse starten

### → Lastbalancierung

- Applikation kann eine oder mehrere Server-Klassen besitzen
- TP-Monitor weist Server-Klasse ankommenden Auftrag (TAC) zu; nach Abwicklung Freigabe der auftragsbezogenen BM. TP-Monitor leitet die Antwort an den Client weiter



- **Bildung von Server-Klassen**

- nach Prioritätsaspekten für die TAP-Ausführung
- nach Applikationstypen
- nach Antwortzeitvorgaben
- nach Fehlertoleranzanforderungen, ...

## Ablaufbeispiel

- Transaktionsprogramm „Auszahlung“:

Read message (kontonr, schalternr, zweigstelle, betrag) from Terminal;

**BEGIN TRANSACTION**

**UPDATE Konto**

```
SET kontostand = kontostand - betrag
WHERE konto_nr = kontonr and kontostand >= betrag
```

...

**UPDATE Schalter**

```
SET kontostand = kontostand - betrag
WHERE schalter_nr = schalternr
```

**UPDATE Zweigstelle**

```
SET kontostand = kontostand - betrag
WHERE zweig_stelle = zweigstelle
```

**INSERT INTO Ablage** (zeitstempel, werte)

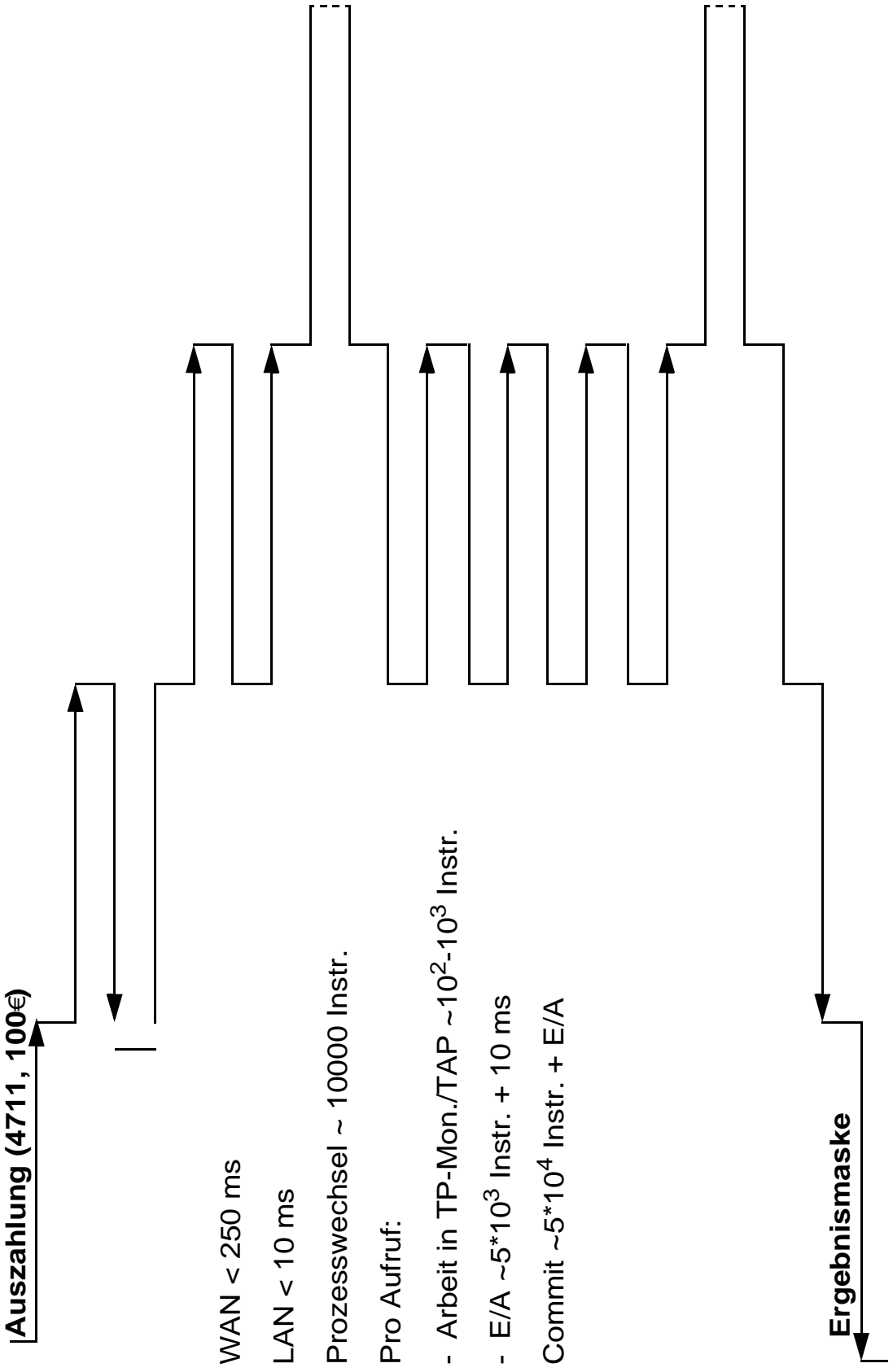
**COMMIT TRANSACTION ;**

Write message (kontonr, kontostand, . . .) to Terminal

- Annahmen

- Alle Indexstrukturen (B\*-Bäume) können im DB-Puffer gehalten werden
- Es gibt sehr viele Objekte vom Typ **Konto**. Sie müssen immer vom **Externspeicher** geholt werden
- Alle Objekte der Typen **Schalter** und **Zweigstelle** (kleine Mengen) sind bereits **im DB-Puffer**
- Alle Änderungen (UPDATE) werden „später“ verdrängt
- Alle Log-Daten werden **bei Commit auf einmal** geschrieben

Client TP-Monitor Applikations-server DB-Server DB



WAN < 250 ms

LAN < 10 ms

Prozesswechsel ~ 10000 Instr.

Pro Aufruf:

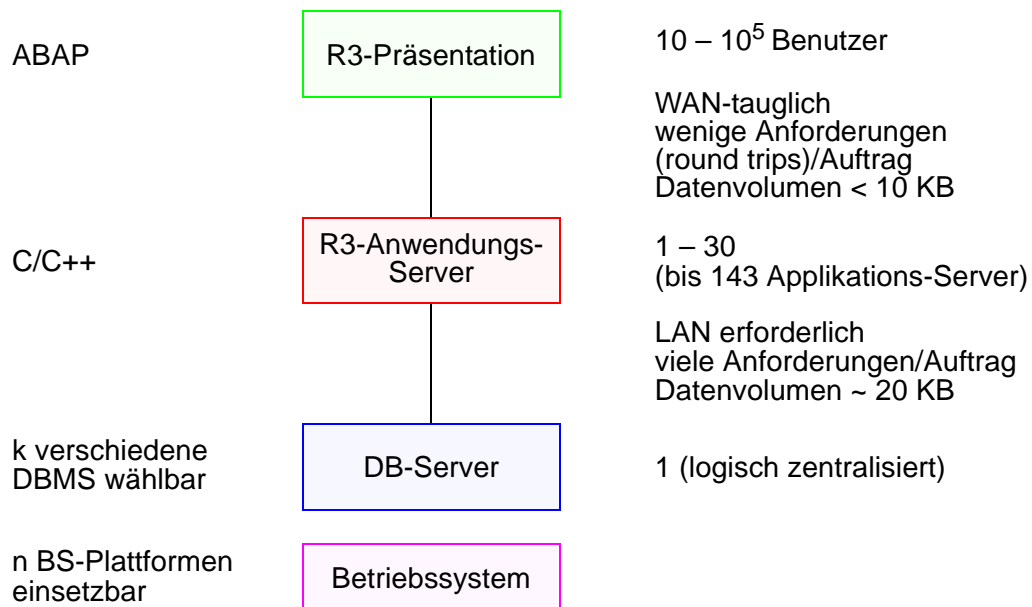
- Arbeit in TP-Mon./TAP ~10<sup>2</sup>-10<sup>3</sup> Instr.

- E/A ~5\*10<sup>3</sup> Instr. + 10 ms

Commit ~5\*10<sup>4</sup> Instr. + E/A

## Dreistufige C/S-Architektur

- **SAP/R3-Realisierung in**



- **Funktionen in der mittleren Schicht (M)**

- ABAP-Interpreter
- TP-Monitor
- AW-Systeme
- Caching von DB-Daten
- DBMS-Schnittstelle

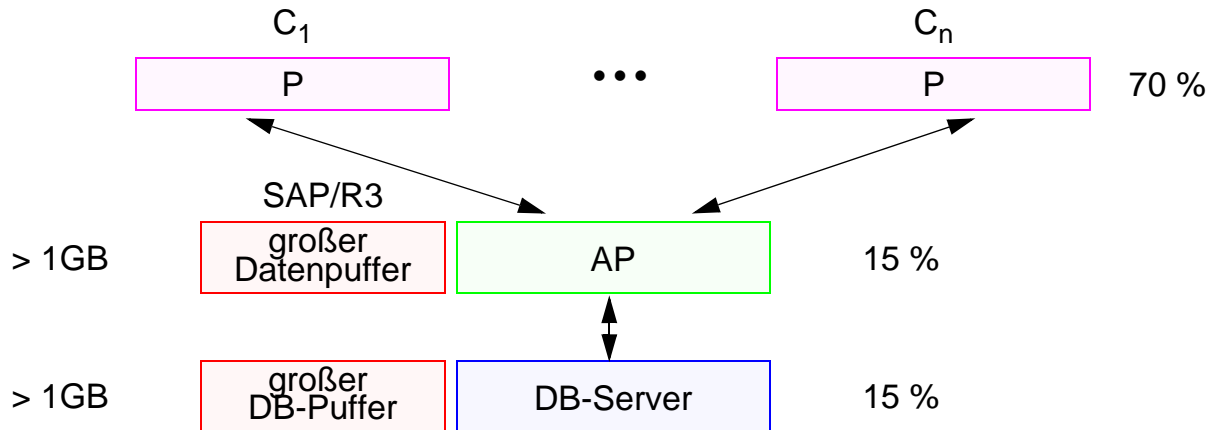
Â **Unternehmensintegration über eine einzige DB, Skalierbarkeit sehr wichtig**

- **Zahlen (SAP ERP Data Model (Part of Business Suite '05))**

- DB-Schema: > 67 000 Tabellen (Relationen), 10 000 Sichten  
700 000 Spalten, 13 000 Indexe, 10<sup>8</sup> Sätze (Anfangsgröße)
- > 270 M LOCs Anwendungsprogramme
- Wachstumsrate pro größerer Version: + 30%
- Systemgröße (leer) auf Platte: 57 GB (*disk foot print, initial size*)
- Unterstützung von mehr als 20 Sprachen

## Dreistufige C/S-Architektur (2)

- Einsatz von großen Puffern

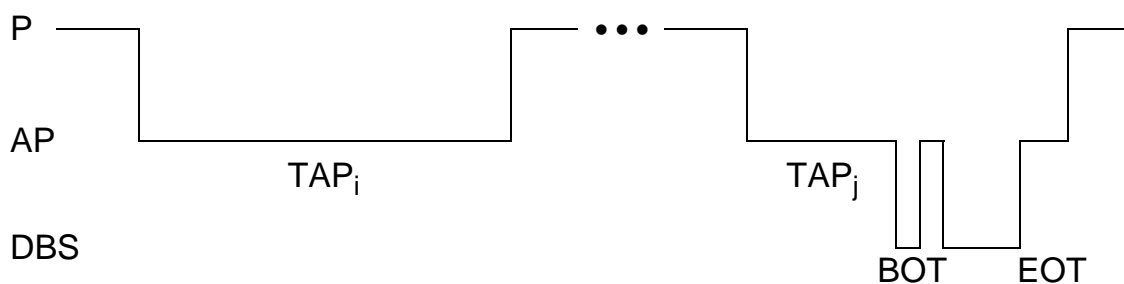


- Ziele

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

Â AP-Server enthält TP-Monitor-Funktionalität und muss viel DBS-Funktionalität reimplementieren

- Zeitlicher Ablauf einer Transaktion

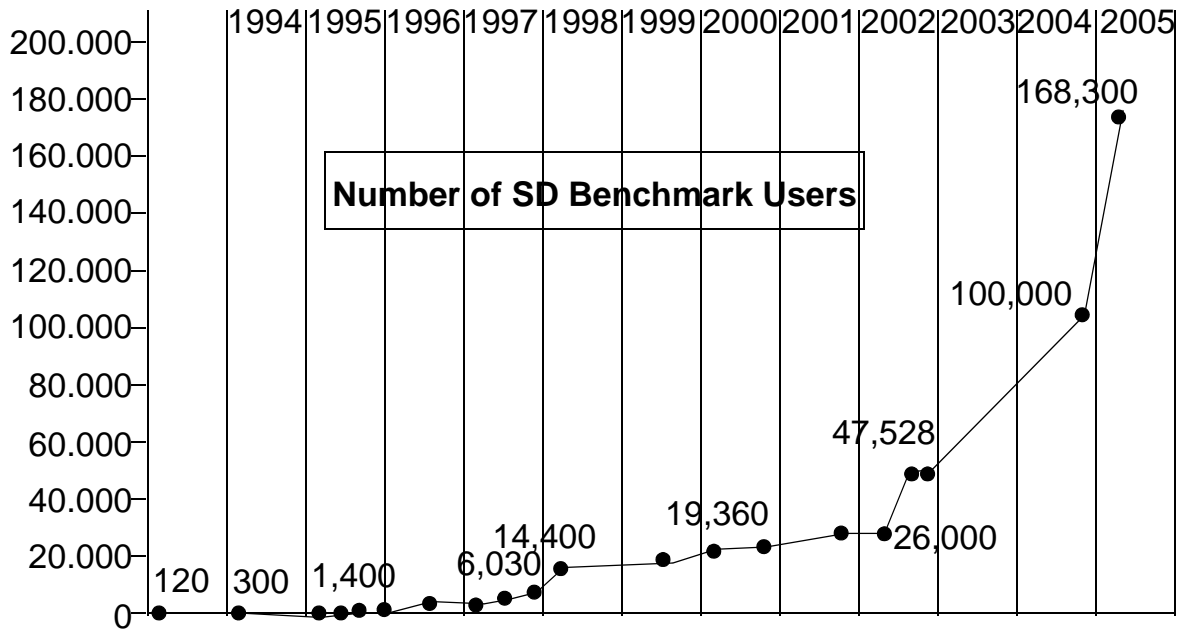


- Eigenschaften

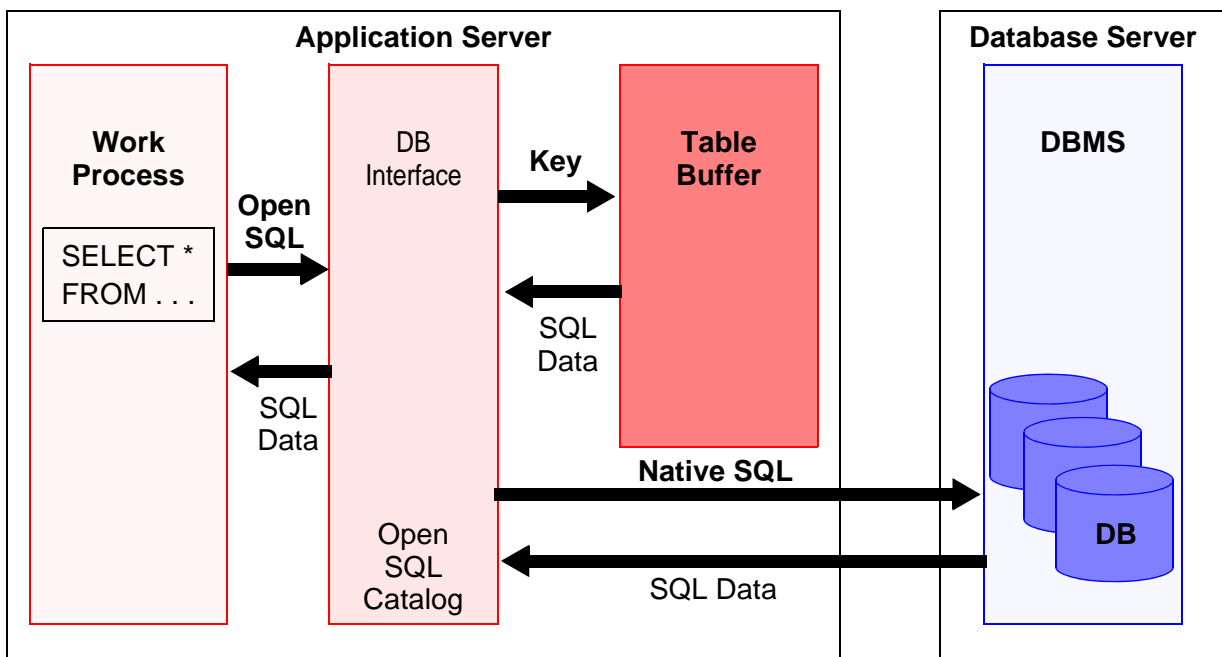
- Präsentation mit GUI ist sehr aufwendig; Präsentationslogik (Benutzerinteraktion) sollte nicht im TA-Pfad liegen!
- Mengenorientierter Zugriff (SQL) auf AP-Puffer ist sehr restriktiv
- Hauptsächlich genutzte DBS-Funktionalität: **Datenspeicher und Logging/Recovery**

## SAP/R3 – Results and Observations

- Published Results for SD Benchmark (three-tier)



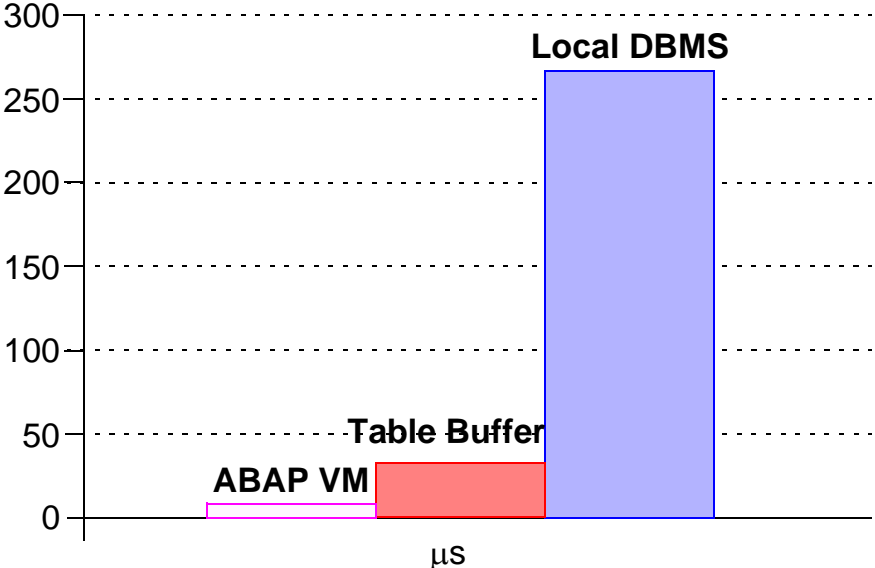
- Table Buffer in Application Server (caching close to the application)



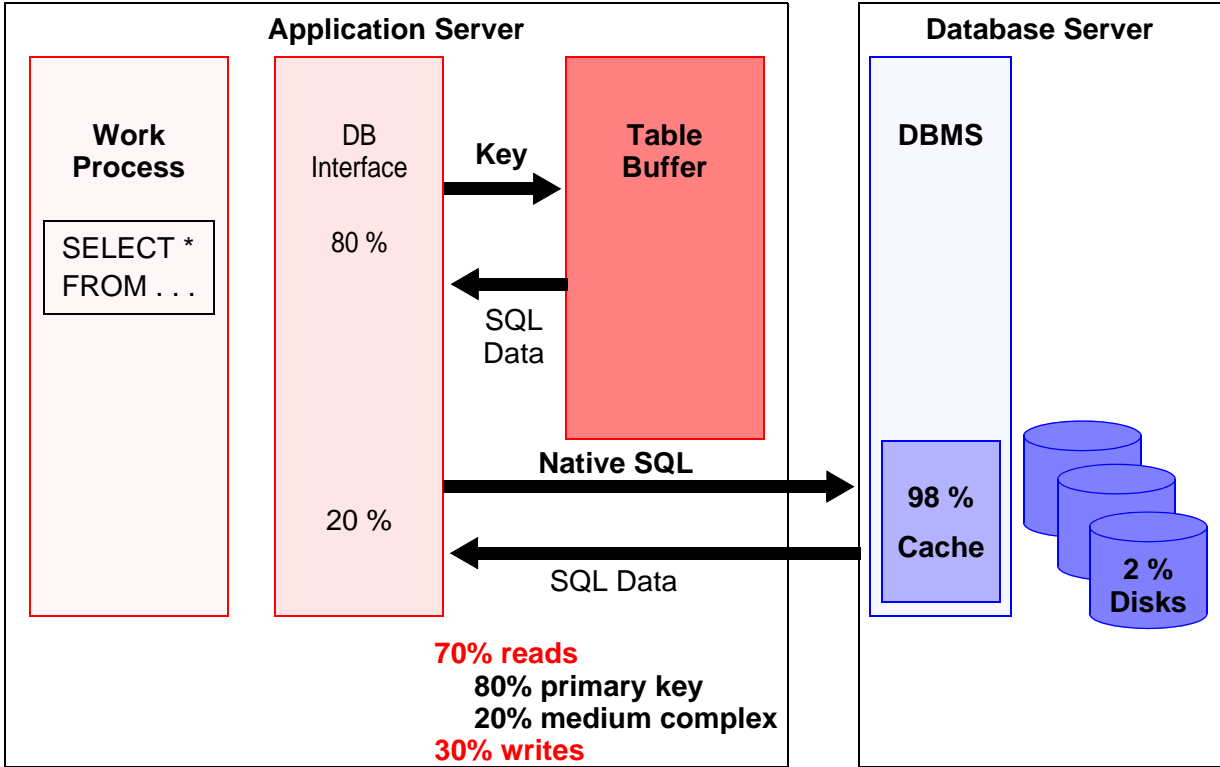


# SAP/R3 – Results and Observations (2)

- Performance of Table Buffer vs. DBMS (Primary Key Access)



- Typical OLTP Traffic Distribution



## TA-Verarbeitung in offenen Systemen

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

Definition von IEEE im Rahmen der POSIX-Aktivität:

Ein **offenes System** ist

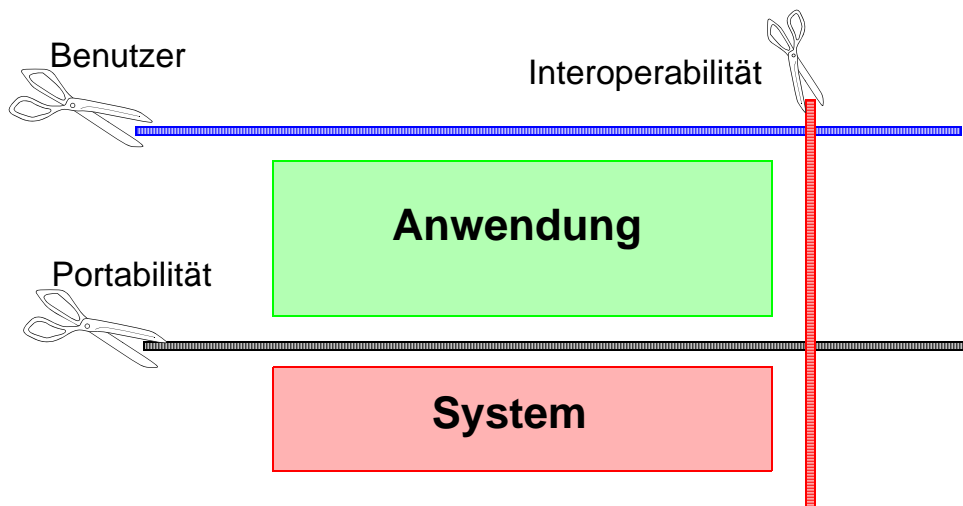
„ein System, das in ausreichendem Maße offengelegte Spezifikationen für Schnittstellen und dazugehörige Formate implementiert, damit entsprechend gestaltete Anwendungssoftware

- 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**“.

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

- **Offenheit impliziert Standardisierung**

- Definition und Veröffentlichung von Schnittstellen



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

## TA-Verarbeitung in offenen Systemen (2)

- **Standards zur TA-Verwaltung**<sup>10</sup>

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

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

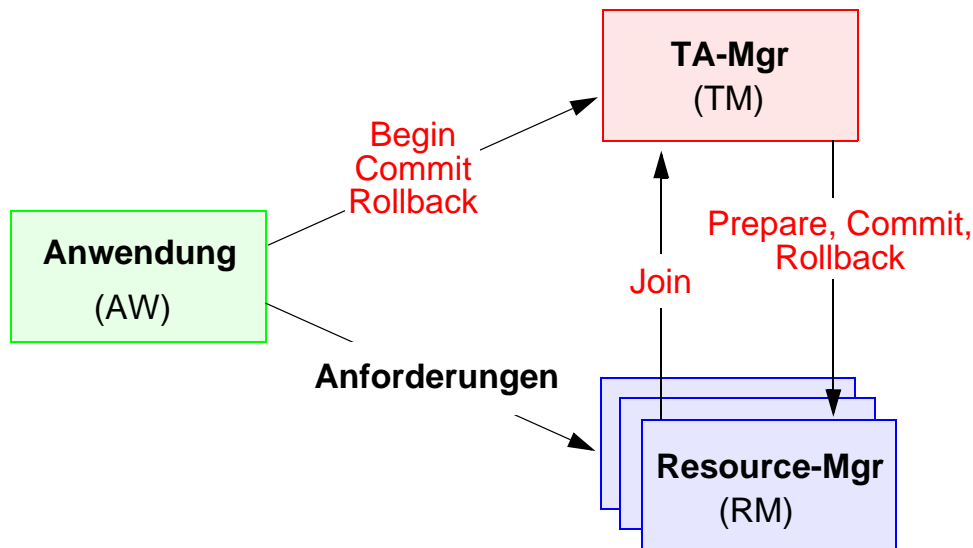
- **Standards kommen aus zwei Quellen**

- **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.

---

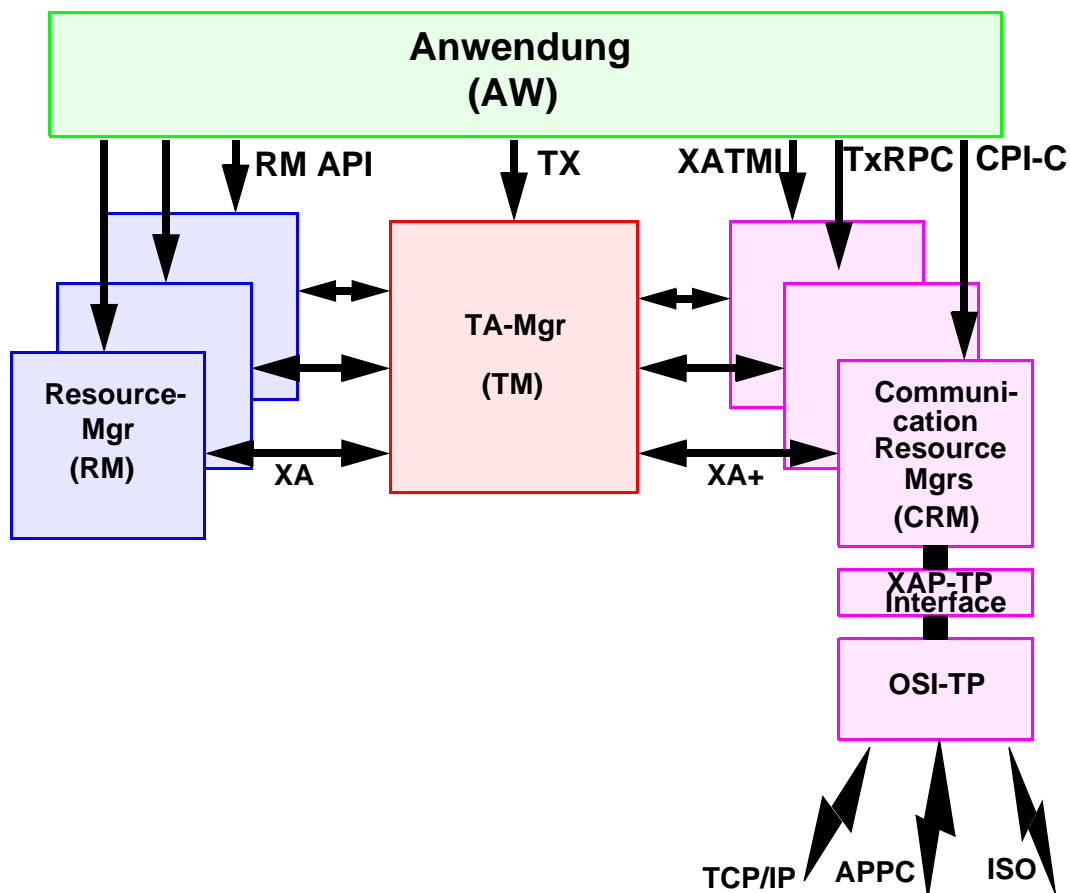
10. „May all your transactions commit and never leave you in doubt!“ (Jim Gray)

## 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 erster Dienst-Anforderung beim lokalen TA-Mgr an (Join)
    - Wenn AW Commit oder Rollback ausführt (oder zwangsweise zurückgesetzt wird), schickt TA-Mgr Ergebnis zu den RMs (über Broadcast)
- Â **hier sorgen die RMs für Synchronisation und Logging** in ihrem Bereich (private Lock-Mgr und Log-Mgr) — Warum?

## TA-Verarbeitung in offenen Systemen (4)



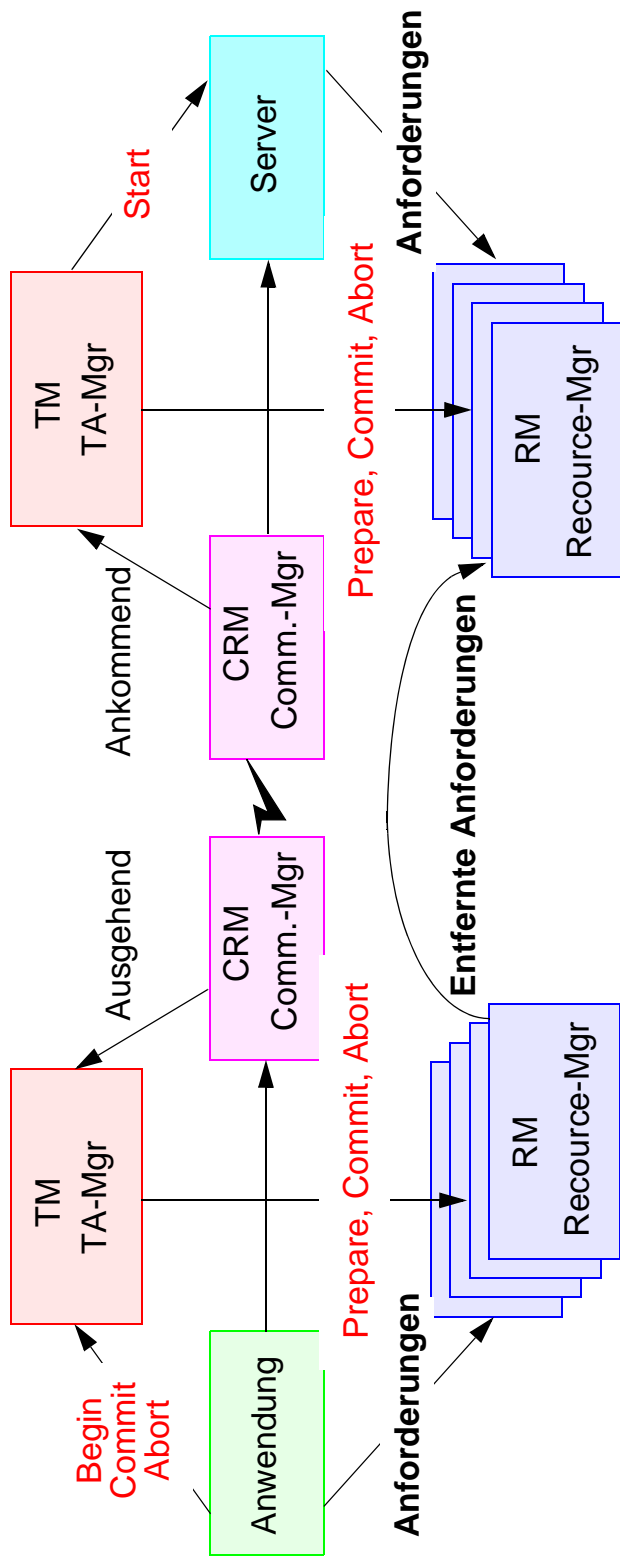
- **X/Open DTP (1993)** standardisiert die verteilte TA-Verarbeitung.
  - Es kommt der „Communication Resource Manager“ (CRM) hinzu.
  - XA+ erweitert Schnittstelle XA für den verteilten Fall.

- **Definition von Schnittstellen auf der AW-Ebene**

- TxRPC definiert transaktionalen RPC. Seine TA-Eigenschaften können ausgeschaltet werden (optional)
- CPI-C V2 ist Konversationsschnittstelle (peer-to-peer), welche die OSI-TP-Semantik unterstützen soll
- XATMI ist Konversationsschnittstelle für Client/Server-Beziehungen

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

## Transaktionsverarbeitung in offenen Systemen (5)



- **TA-Ablauf**

- AW startet 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

- **Wichtige Server-Eigenschaften**

- Auftrag — Berechnung — Antwort (Ergebnis)
- Verschiedenste Kommunikationsprotokolle
- verbindungsorientiert oder verbindungslos
- Ein wichtiges **zusätzliches Konzept: „Session“**

- **Bewertung des Client/Server-Ansatzes**

im Vergleich zu zentralen Systemstrukturen

- Er bietet eine Reihe von Vorteilen insbesondere bei Kosten, Wachstum, Flexibilität, Offenheit, Dezentralisierungsmöglichkeit usw.
- Prinzipielle Nachteile liegen vor allem in der **höheren Komplexität** bei **der Wartung und Verwaltung** großer, heterogener Client/Server-Systeme sowie im **Fehlen eines „single system image“**

- **Zusammenfassung der Vor- und Nachteile**

<b>Pro</b>	<b>Contra</b>
Kosten von Hard- und Software	höhere Komplexität durch Verteilung und Heterogenität
Herstellerunabhängigkeit	Gesamtkosten schwer zu überschauen
attraktive Funktionalität	komplexe Netzinfrastrukturen
Ergonomie der Benutzeroberfläche von PCs und Workstations	unausgereifte Managementlösungen
Flexibilität und Skalierbarkeit	unausgereifte Sicherheitsmechanismen
Entkopplung durch Dezentralisierung	Trainingsaufwand
Entsprechung organisatorischer Strukturen	wichtige Anwendungen nicht verfügbar

## Zusammenfassung (2)

- **C/S-Architekturen**

- 2-stufig: fette Clients vs. fette Server, keine Skalierung
- 3-stufig: gute Skalierbarkeit, usw.
- n-stufig: meist Einbezug von Web-Komponenten

- **Server-Strukturen und -Techniken**

- Zur Steigerung der Effizienz: Einsatz von Programmverkettung, Multi-Tasking, Multi-Threading, ablaufinvarianten Programmen
- Programmierung dennoch so einfach wie möglich: Einen Auftrag bearbeiten, Kontext = lokale Variablen, Isolierung, Code-Ablaufinvarianz durch Compiler
- Aufgabe der Middleware insbesondere: Strategien änderbar (optimierbar)

- **TP-Monitore**

- liefern den „Klebstoff“ in komplexen C/S-Umgebungen
- erlauben die Umsetzung des Transaktionskonzeptes als Grundlage zur Realisierung zuverlässiger verteilter Systeme
- sind Spezialisten für **Prozess-, Transaktions- und C/S-Kommunikations-Verwaltung**

- **X/OPEN DTP**

- Interoperabilität über standardisierte Schnittstellen und Protokolle
- für die lokale Zusammenarbeit von Anwendung, TA-Mgr und RMs
- und für die verteilte TA-Verarbeitung mit hierarchischen 2PC



## Zusammenfassung - Vielfalt an Bezeichnungen

- **RPC-basierte Systeme:**

Ein RPC ermöglicht die synchrone Kommunikation zwischen einem Client und einem Server, der sich auf einem anderen Rechner befindet. Aus der Sicht eines aufrufenden Programms (Client) sieht der RPC wie ein gewöhnlicher Prozeduraufruf aus. **Synchrone Kommunikation** bedeutet, dass der Client beim Aufruf des RPC den Programmfluss unterbricht und auf die Ergebnisse der Prozedur wartet. Der RPC verbirgt also die Etablierung eines Kommunikationskanals durch die Bereitstellung einer **standardisierten Programmierschnittstelle**. RPC-basierte Systeme bilden heute die Grundlage für fast jede Form von Middleware. So stellt beispielsweise auch das SOAP-Protokoll (Simple Object Access Protocol), das eine Interaktion zwischen Web Services unterstützt, im Wesentlichen eine Möglichkeit dar, RPC-Aufrufe auf XML-Nachrichten abzubilden.

- **TP-Monitore:**

TP-Monitore stellen die älteste Form von Middleware dar und gleichzeitig die stabilste und zuverlässigste. Ein TP-Monitor ist ein Steuerungsprogramm zur transaktionsorientierten Verwaltung von Anwendungsprogrammen, insbesondere OLTP-Anwendungen (Online Transaction Processing). Letztlich basiert auch ein TP-Monitor auf dem RPC, der hier jedoch **transaktionsgeschützt** abläuft. Ein so genannter **Transaktionaler RPC (TRPC)** gewährleistet die Einhaltung der ACID-Eigenschaften von Transaktionen, die entfernte Prozeduraufrufe tätigen.<sup>11</sup>

- **Objekt-Broker:**

Rein implementierungstechnisch unterscheiden sich Objekt-Broker kaum von anderen RPC-basierten Systemen. Der wesentliche Fortschritt gegenüber dem traditionellen RPC, der auf den Programmierparadigmen imperativer Programmiersprachen beruht, ist der **Bezug zur objektorientierten Programmierung**. Objekt-Broker unterstützen den entfernten Zugriff auf Objekte. Als wichtigste Vertreter **dieser Form der Middleware sind die auf COBRA (Common Object Request Broker Architecture) basierenden Objekt-Broker** zu nennen.

---

11. Die hier beschriebene Middleware des TP-Monitors stellt eine 3-tier-Architektur dar, die auch als „TP-heavy“ bezeichnet wird. Unter „TP-lite“ versteht man eine 2-tier-Architektur, bei der die Anwendungslogik (zumindest zum Teil) in Form so genannter „stored procedures“ auf der Seite der Ressourcen-Mgr angesiedelt wird.

## Zusammenfassung - Vielfalt an Bezeichnungen (2)

- **Objekt-Monitore:**

Der Trend hin zur objektorientierten Programmierung hat dazu geführt, dass die gut erprobte Funktionalität von TP-Monitoren, die zunächst für imperative Programmierumgebungen ausgelegt war, auch im Kontext verteilter objektorientierter Systeme gewünscht wurde. Objekt-Monitore sind somit das Resultat der **Konvergenz zwischen TP-Monitoren und Objekt-Brokern**. Im Wesentlichen handelt es sich dabei um **TP-Monitore mit objektorientierten Schnittstellen**.

- **Nachrichtenorientierte Middleware – Message-Oriented Middleware (MOM):**

Eine nachrichtenorientierte Middleware ist speziell für die Unterstützung einer **asynchronen Kommunikation auf der Basis persistenter Warteschlangen** ausgerichtet. Viele Anwendungen erfordern keine synchrone Kommunikation, für diese wäre ein RPC also ein unnötiger Aufwand. Im Rahmen der umfassenden Funktionalität von TP-Monitoren wurden daher bereits asynchrone Kommunikationsmechanismen, wie der asynchrone RPC und persistente Warteschlangen, zur Verfügung gestellt. Durch die persistente Zwischenspeicherung der Nachrichten wird eine bessere Entkopplung von Sender und Empfänger erreicht, da die Nachricht auch dann noch sicher zugestellt werden kann, wenn der Empfänger zwischenzeitlich nicht erreichbar war. Eine nachrichtenorientierte Middleware stellt typischerweise verschiedene Operationen zum Umgang mit persistenten Warteschlangen zur Verfügung. Dazu gehört **insbesondere ein transaktionsgeschützter Zugriff auf Warteschlangen**.

Eine **besondere Form der asynchronen Kommunikation**, die mit Hilfe persistenter Warteschlangen und MOM umgesetzt werden kann, stellen so genannte **Publish/Subscribe-Systeme** dar. Auch hierdurch wird eine Entkopplung von Sender und Empfänger erreicht: Ein Sender stellt Informationen ohne Angabe des Empfängers zur Verfügung (Publish), die dann von interessierten Abonnenten bezogen werden können (Subscribe).

## Zusammenfassung - Vielfalt an Bezeichnungen (3)

- **Message Broker**

stellen eine spezielle Form der MOM dar. Die Besonderheit hierbei ist, dass **Nachrichten nicht nur einfach vermittelt** werden, sondern dass der **Message Broker zusätzlich die Möglichkeit bietet, Nachrichten zu filtern und gemäß vorgegebener Regeln zu transformieren**. Auf diese Weise können beispielsweise die Empfänger einer Nachricht dynamisch aufgrund des Inhalts der Nachricht bestimmt werden. Die Möglichkeit zur Transformation ist hilfreich zur Abbildung verschiedener Datenformate auf Sender und Empfängerseite. Diese Form der Middleware lässt sich im Zusammenhang mit der Integration heterogener IT-Systeme hilfreich einsetzen.

- **Application Server**

unterscheiden sich in ihrer Kernfunktionalität nicht wesentlich von den bereits vorgestellten Middleware-Ansätzen. Der **Hauptunterschied besteht darin, dass Application Server darauf ausgerichtet sind, das Web als Zugriffskanal für die implementierten Middleware-Dienste nutzbar** zu machen. Damit verbunden ist vor allem eine Ausrichtung auf den dokumentenorientierten Informationsaustausch mittels HTTP (HyperText Transfer Protocol). Zur Ergänzung herkömmlicher Middleware-Plattformen benötigen Application Server daher vor allem Mechanismen zur **dynamischen Erzeugung und Verwaltung von Hypertext-Dokumenten**. Damit wird die nunmehr Web-orientierte Präsentationsebene wieder mit der Anwendungsebene verknüpft. Als Programmierschnittstellen (API) für Application Server haben sich im Wesentlichen zwei Ansätze herausgebildet: der von SUN propagierte **Java-basierte Ansatz J2EE** und der von Microsoft propagierte Ansatz **.Net**. Über diese APIs werden vielfältige Middleware-Dienste, die von TP-Monitoren, Objekt-Brokern und nachrichtenbasierter Middleware bereits bekannt sind, im Rahmen einer einheitlichen Schnittstelle für Anwendungsprogrammierer verfügbar gemacht.

## Zusammenfassung - Vielfalt an Bezeichnungen (4)

- **Workflow-Management-Systeme (WfMS)**

haben ihren Ursprung in der Büroautomation, wo es darum ging, gut strukturierbare Abläufe, die bisher auf Papierbasis stattfanden, durch den Einsatz der IT zu automatisieren. Viele der administrativen Workflows, die zunächst im Fokus waren, werden heute im Rahmen von Dokumenten-Management-Systemen (DMS) umgesetzt. DMS bilden damit letztlich eine spezielle Form von WfMS. Ein wesentliches Ziel der WfMS ist die Durchgängigkeit von der Geschäftsprozessmodellierung bis hin zur Ausführung von Workflows. Verschiedene Methoden zur Spezifikation von Workflows lassen sich unterscheiden: **Kommunikationsbasierte Methoden** sind in erster Linie für die Spezifikation von Dialogen mit menschlichen Akteuren geeignet. **Aktivitätsorientierte Methoden** betrachten die im Rahmen eines Geschäftsprozesses zu leistende Arbeit, indem sie sich auf die Aktivitäten und die Abhängigkeiten zwischen ihnen konzentrieren (z. B. Petri-Netze, Ereignis-Prozessketten (EPK)). **Entitätenbasierte Methoden** fokussieren auf die Modellierung von Objekten, die während des Anwendungsprozesses erzeugt, bearbeitet und verbraucht werden (z. B. UML). Die meisten WfMS basieren allerdings – zumindest hinsichtlich ihrer Benutzerschnittstelle – auf gerichteten Graphen (aktivitätsorientierten Modellierungsmethoden). Der Unterschied zu herkömmlichen Programmiersprachen besteht vor allem darin, dass WfMS langlaufende Abläufe mit vielen verschiedenen Beteiligten in einer verteilten Ausführungsumgebung unterstützen müssen. Das hat vor allem zur Folge, dass WfMS komplexe Techniken zur Fehlerbehandlung vorzusehen haben, die für die Abwicklung kurzer Transaktionen nicht nötig wären. Für die Ausführung von Workflows ist eine „Workflow Engine“ zuständig. Diese verwaltet Instanzen von Workflows, die im Sinne eines Schedulers schrittweise abgearbeitet werden: Die Workflow Engine bestimmt, welche ausführenden Komponenten im Rahmen des Workflows wann aufgerufen werden. Mit zunehmender Reife der WfMS und mit dem wachsenden Bedarf nach durchgängig IT-unterstützten Geschäftsprozessen wurden WfMS immer mehr auch als Werkzeuge zur Integration heterogener Anwendungssysteme angesehen. WfMS bilden damit Werkzeuge zur Unterstützung der „Programmierung im Großen“. Die Idee besteht darin, große Software-Module als Komponenten in die Spezifikation einer übergreifenden Geschäftsprozesslogik einzubinden. Dies erfordert allerdings eine Kombination mit herkömmlichen Middleware-Plattformen, die notwendig sind, um einen transparenten Zugriff auf heterogene Komponenten zu ermöglichen.