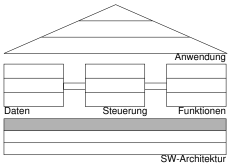
9. Architekturmodelle

- **GBIS-Rahmen: Einordnung**



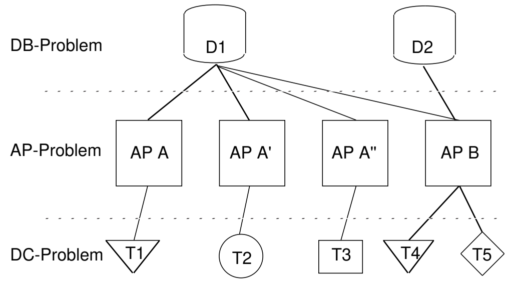
- **Anwendungsentwicklung – allgemeine Probleme**
- **Sicht des Endbenutzers**
  Dialogschritt, Transaktion, Vorgang
- **Entwurfsaufgaben**
  - Zerlegung/Kooperation der Komponenten
  - Komponentenzuordnung in verteilter Umgebung
- **Client/Server-Modell**
  Unterscheidende Eigenschaften
- **Zwei- und dreistufige C/S-Architekturen**
  - weites Spektrum von Anwendungen
  - Zusammenspiel der Komponenten
- **TP-Monitor**[1]
  - Eigenschaften und Funktionalität
  - Einsatz in C/S-Umgebungen
- **Aufbau eines DB-Servers**
  - Drei-Schichten-Architektur
  - Ablaufbeispiel

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

AW-Entwicklung – allgemeine Probleme

- **Problem**

  Komplexität der Kommunikations- und Verarbeitungsbeziehungen
  - K Terminals (Typen, Prozesse)
  - M Anwendungen
  - N Dateien
- **Herkömmliche (statische) Struktur**



- **Generelle Aufgaben**
  - Ablaufkontrolle
  - Betriebsmittelzuteilung
  - Synchronisation und Sicherung der Datei-Zugriffe
  - Verwaltung und Abbildung von Dateistrukturen
  - Steuerung der Kommunikationsbeziehungen

➥ Wer hält alles zusammen ?

AW-Entwicklung – allgemeine Probleme (2)

- **DB-Problem**

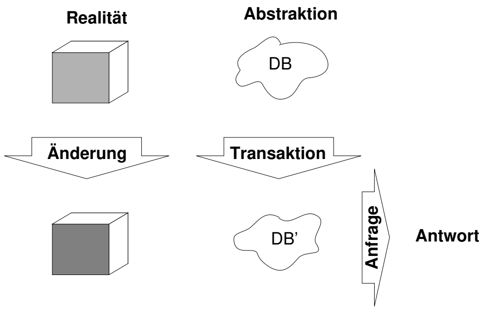
  individuelle Datenverwaltung führte zu
  - umständlichem Zugriff
  - geringer Flexibilität
  - Redundanz
  - Konsistenzverlust
- **AP-Problem**

  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
  - . . .

Grundlegende Abstraktionen von TA-Systemen

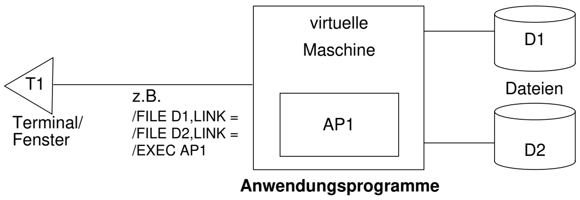
- **Abstraktion**
  - Der reale Zustand (einer Miniwelt) wird durch eine Abstraktion – DB genannt – dargestellt
  - Eine Veränderung des realen Zustands wird durch ein Programm – Transaktion genannt – in der DB nachvollzogen



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

Sicht des Endbenutzers

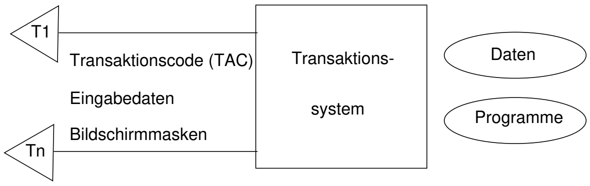
- **Benutzerschnittstelle für TA-Systeme?**



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

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

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



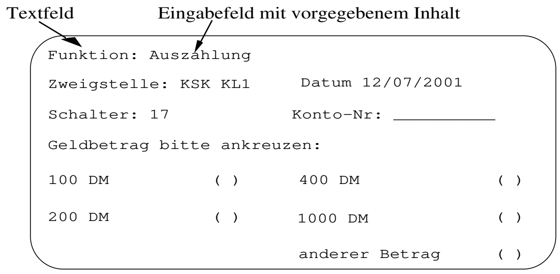
- problembezogene Funktionen
- einfache Bedienung
- Datenunabhängigkeit
- Programmunabhängigkeit

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

Sicht des Endbenutzers (2)

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

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

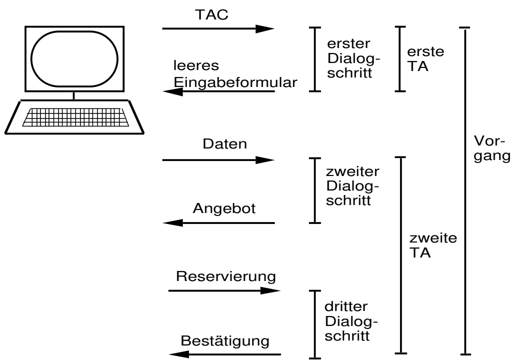


- **Transaktionsprogramm „Auszahlung“:**

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 (transparent) 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 dazugehörenden Ausgabenachricht; Abwicklung eines einzelnen Auftrags

- **Transaktionscode (TAC)**

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

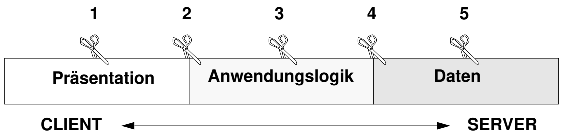
- **Vorgang**

  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

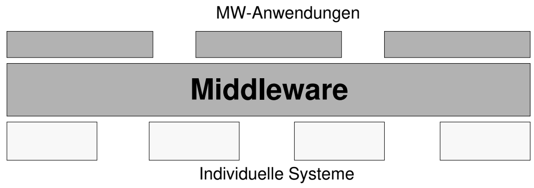
- **Architektur eines SW-Systems**
  - Aus welchen Komponenten ist es aufgebaut?
  - Wie und wo arbeiten diese Komponenten zusammen?
- **Applikationen in betrieblichen Informationssystemen**
  - lassen sich häufig grob durch folgende Funktionalität charakterisieren:
    - Präsentation oder Benutzerschnittstelle: 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 Gesamtmenge der Dienste, die durch die individuellen Systeme bereitgestellt werden
    - sog. Middleware implementiert die integrierte Sicht
    - „/“ von C/S
- **Verteilungsplattform** garantiert transparente Verteilung

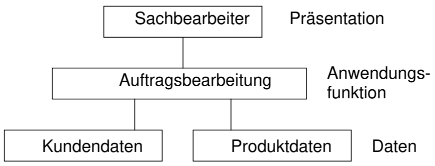


  ➥ Sie benötigt eine Vielzahl an Protokollen, um die Interaktion zwischen verteilt ablaufenden Anwendungskomponenten unterstützen zu können

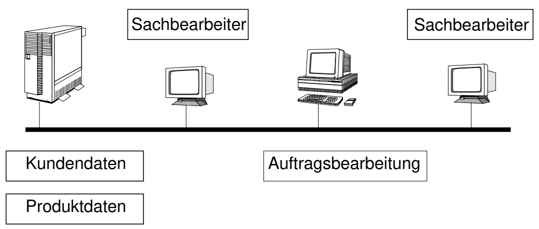
- **Verbergen von Entwurfs- und Ablaufentscheidungen**
  - Unterschiedliche Transparenzanforderungen für
    - Sachbearbeiter
    - AW-Programmierer
    - Systemadministrator
  - Einsatzspektrum
    - lokale Anwendungen: abteilungsbezogenes C/S-System
    - weltweit verteilte AW: „intergalaktisches“ C/S-System
    - ➥ Ortstransparenz bei verteilten und ggf. heterogenen Systemen besonders wichtig

Entwurfsaufgaben (3)

- **Zuordnung in einer verteilten Rechnerumgebung**
  - 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



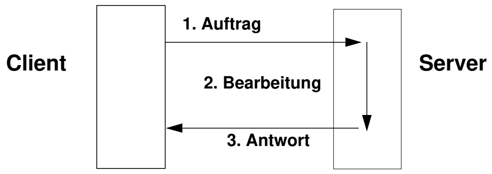
  ➥ **Mögliche Abbildung** auf ein konkretes Rechnersystem (Konfigurationsproblem)



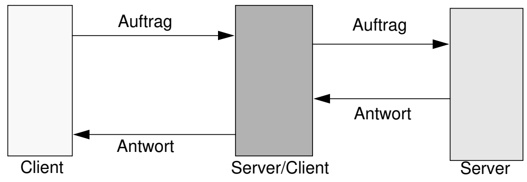
  ➥ Die Bewertung einer gegebenen Konfiguration und die Bestimmung der optimalen Lösung sind i. allg. sehr schwierige Probleme (wird nicht vertieft)

Client/Server-Modell

- **SW-Architekturmodell mit folgendem Grundschema der Kooperation:**



- **Eigenschaften**
  - Initiative zur Interaktion geht vom Client aus
  - Server hat seine Bereitschaft, bestimmte Aufträge entgegenzunehmen, veröffentlicht
  - 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
  - Die Kommunikationsinfrastruktur wird nicht explizit modelliert
  - Das Modell bezieht sich auf logische Verteilungen, d. h., es sind lokale und entfernte Aufrufe möglich

Client-/Server-Modell (2)

- **Grobe Definition**

  Client und Server sind separate logische Einheiten, die (über ein Netzwerk) zusammenarbeiten, um eine Aufgabe durchzuführen.[1]

- **Unterscheidende Eigenschaften**

  - **Dienste (*Services*)**
    C/S verkörpert primär eine Beziehung zwischen Prozessen (die auf verschiedenen Rechnern ablaufen). C/S bewirkt eine klare Funktionstrennung, die durch die Nutzung von Diensten erzielt wird.

  - **Gemeinsame Betriebsmittel (Ressourcen)**
    Ein Server kann gleichzeitig viele Clients „bedienen“ und deren Zugriffe auf gemeinsame Betriebsmittel steuern und überwachen.

  - **Asymmetrische Protokolle**
    Es besteht eine (1:n)-Beziehung in beiden Richtungen. Ein Client kann im Laufe der Verarbeitung auf mehrere verschiedene Server zugreifen, und ein Server kann viele verschiedene Clients bedienen.
    Server bieten (passiv) Dienste an, während Clients (aktiv) durch Anforderung von Diensten den Dialog beginnen.

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

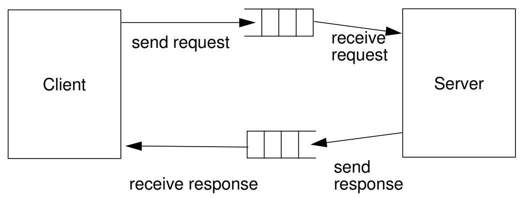
Client-/Server-Modell (3)

- **Unterscheidende Eigenschaften (Fortsetzung)**

  - **Ortstransparenz**
    Die C/S-Software „maskiert“ gewöhnlich den Ort des Servers vor den Clients und leitet Dienstaufrufe, falls erforderlich, um. Ein Programm kann die Rolle eines Clients, eines Servers oder von beiden einnehmen.

  - **Plattformunabhängigkeit**
    Die C/S-Software ist unabhängig von HW- oder Betriebssystem-Plattformen. Client- und Server-Plattformen sollen kombiniert werden können.

  - **Nachrichtenbasierter Austausch**
    Clients und Server sind lose gekoppelte Systeme, die durch Nachrichten (*message-passing mechanisms*) kooperieren. Nachrichten verkörpern den Zustellmechanismus für Dienstanforderung und Ergebnisbereitstellung.



Anforderungs- und Ergebnis-WS sind oft persistente Objekte

Client-/Server-Modell (4)

- **Unterscheidende Eigenschaften (Fortsetzung)**

  - **Kapselung von Diensten**
    Ein Server erkennt aus der Nachricht, welcher Dienst angefordert wird. Wie der Dienst erbracht wird, bestimmt allein der Server. Solange die Nachrichten-Schnittstelle nicht verändert wird, können Server ohne Rückwirkung auf Clients „ausgebaut“ oder erweitert werden.

  - **Skalierbarkeit**
    C/S-System können horizontal oder vertikal erweitert werden.

    Horizontale Skalierbarkeit: Hinzufügen/Entfernen von Clients ohne (oder nur geringfügige) Leistungsbeeinflussung.

    Vertikale Skalierbarkeit: Einsatz von größeren/schnelleren Servern (Rechnern) oder Multi-Servern.

  - **Integrität**
    Programme und Daten eines Servers werden zentral gewartet, was kostengünstig ist und eine höhere Datenintegrität erwarten läßt. Clients bleiben unabhängig und personenbezogen; die Verantwortung für die Integrität von Daten und Programmen liegt bei den Clients.

Zweistufige C/S-Architekturen

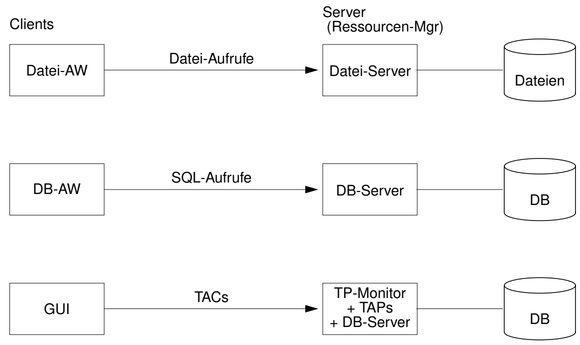
- **Vielfalt an C/S-Anwendungen**

  - C/S-Eigenschaften bieten einen geeigneten Entwurfsrahmen für lose gekoppelte, netzbasierte Awendungen.

  - Was sind alles C/S-Technologien?
    (Datei- und DB-Server, Verteilte Objekte, TP-Monitore, Groupware, Internet-Anwendungen, ...)

  - **Grundidee:** Zerlegung von Anwendungen mit Hilfe von C/S-Schnittstellen
    ➥ letzte Dekade: viele abteilungsbezogene C/S-Anwendungen in LANs; heute: „intergalaktische“ C/S-Anwendungen im Internet

- **Vereinfachte Beispiele**

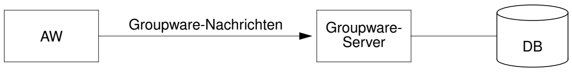


- Ein Ressourcen-Mgr (RM) ist ein Programm, das **gemeinsam benutzbare** Ressourcen (DBs, Dateien, Archive, Warteschlangen usw.) verwaltet.

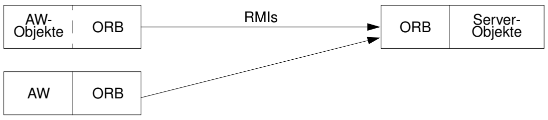
Zweistufige C/S-Architekturen (2)

- **Großes Spektrum weiterer Anwendungen**

- **Groupware-Server**
  - dienen der Verwaltung von semi-strukturierten Dokumenten wie Text, Bild, Mail, Bulletin-Boards sowie der Workflow-Kontrolle
  - Beispiel:
    Dokumentenverwaltungssystem zur Begutachtung von Publikationen
  - Einsatz von Skriptsprachen und formularbasierten Schnittstellen
  - künftig: Internet wird die **Middleware für Groupware**

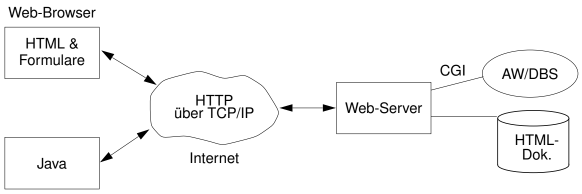


- **Objekt-Server**
  - C/S-Anwendung ist als Menge von kommunizierenden Objekten realisiert
  - Client-Objekte kommunizieren mit Server-Objekten unter Benutzung eines ORB (Object Request Broker)
  - Server-Objekte müssen konkurrierende Zugriffe und gemeinsame Benutzung unterstützen
  - Wichtige Infrastrukturen
    - **CORBA:** architektur- und sprachunabhängig
    - **DCOM:** Distributed Component Object Model



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
  - Verteilte Objekte (in Java) und das Web kommen sich „immer näher"



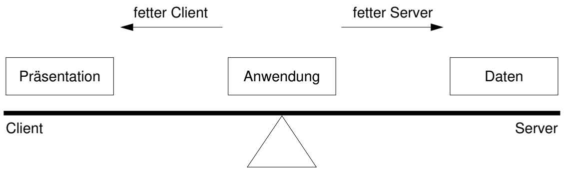
- **Merkmale eines TA-Systems**
  - Benutzung im Dialog: „parametrischer" Benutzer
  - wenige kurze Transaktionstypen: hohe Wiederholrate
  - sehr viele Benutzer gleichzeitig
  - gemeinsame Datenbestände mit größtmöglicher Aktualität
  - kurze Antwortzeiten als Optimierungsziel vor Auslastung
  - stochastisches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

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

Zweistufige C/S-Architekturen (4)

- **Bisher**
  Klassifikation der C/S-Modelle erfolgte nach den bereitgestellten Diensten

- **Andere 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
  - Der größte Lastanteil fällt auf der 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
  - versuchen den Netzverkehr zu minimieren, indem sie mächtigere Dienste anbieten (höhere Abstraktionen bei den Diensten, Export von Prozeduren/Methoden)
  - Client bietet GUI und kooperiert mit Server über RPC oder RMIs

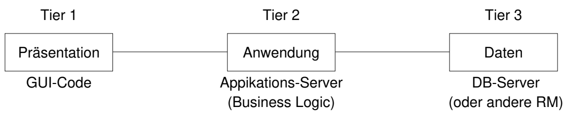
    ➥ Beide C/S-Modelle haben ihre Vorteile. Sie komplementieren sich und koexistieren oft in einer Anwendung

Zweistufige C/S-Architekturen (5)

- **Bisher: 2-stufige C/S-Architekturen**
  - Client: Benutzerschnittstelle (GUI) oder Präsentation
    Server: gemeinsam genutzte Daten
    Anwendungs-Code (weitgehend) auf einer der beiden Seiten
  - 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:** Einsatz als unternehmensweite C/S-Anwendung und beim Internet-basierten Electronic Commerce gefordert

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



  - Unterstützung von komplexen und **verteilten** Anwendungen
    - Web-basierte Clients, viele SW-Komponenten, heterogene Daten
    - Wachstum und Evolution sind immanent: heterogene Rechner/Plattformen, Versionen von Programmen und Daten, Altlasten-Probleme (legacy)
  - Anwendungslogik (*middle tier*) läuft in eigenen Prozessen ab
  - Leichte Konfigurierbarkeit der Prozesse („first-class citizens"). Sie lassen sich separat von GUI und DB-Server verwalten und einsetzen
  - Verbesserte Skalierbarkeit, Sicherheit, Zuverlässigkeit, . . .

C/S-Architekturen – Vergleich

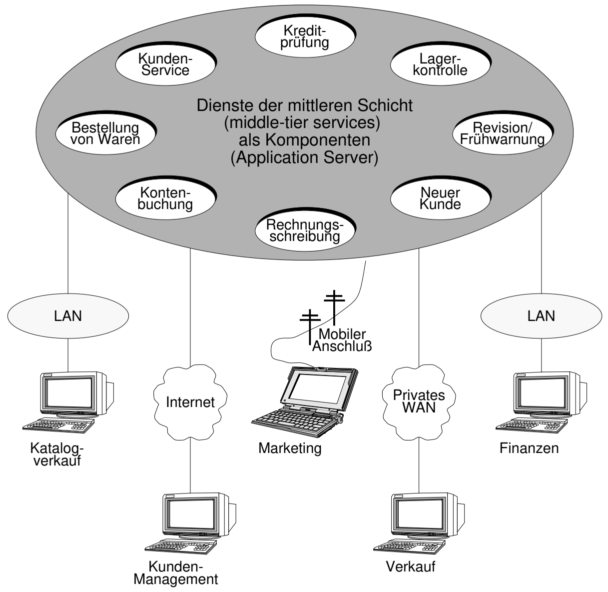
- **Vergleich der Anwendungscharakteristika**
  Intergalaktisches C/S vs. abteilungsbezogenes C/S

Anwendungscharakteristika	abteilungsbezogenes C/S	intergalaktisches C/S
Anzahl von Clients pro AW	< 100	> 10^6
Anzahl von Servern pro AW	1 oder 2 homogene Server	10^5 ++ (viele heterogene Server in verschiedenen Rollen)
Geographie	Campus	global
Interaktionen zwischen Servern	nein	ja
Middleware (Verteilungsplattform)	SQL und stored procedures	Komponenten im Internet und in Intranets
C/S-Architektur	2-stufig	3-stufig
transaktionsgeschützte Änderungen	selten	sehr häufig
Multimedia-Inhalt	gering	hoch
Mobile Agenten	keine	zunehmend mehr
Client-Front-Ends	fette Clients (GUIs)	On-demand-Clients, Web-Tops, komplexe Dokumente
Zeitrahmen	1985 - heute	1997 - 2002 und danach

Dreistufige C/S-Architekturen

- **Mittlere Schicht**
  - Sammlung von Komponenten (kein monolithisches Programm)
  - Jede Komponente realisiert eine relativ kleine Geschäftsfunktion als Transaktionsprogramm (TAP)
  - Clients kombinieren oft mehrere Komponenten zu einer Geschäftstransaktion

- **Großes Spektrum an Übertragungsprotokollen**
  ➥ Wichtigkeit der Programm- und Kommunikationsunabhängigkeit



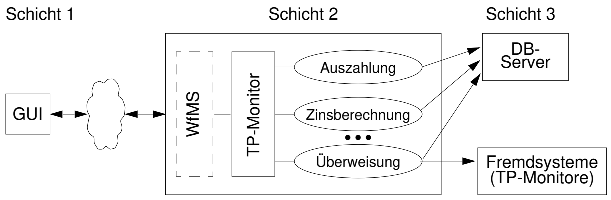
Dreistufige C/S-Architekturen (2)

- **Vorteile der Komponentenorientierung**
  - Entwicklung großer Anwendungen in kleinen Schritten
  - Wiederbenutzung (*Reuse*) von Komponenten (AW-Funktionen)
  - 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

    ➥ **Programm- und Datenunabhängigkeit**
  - Einsatz „vorgefertigter" Komponenten; Vorteile der Nutzung von Standard-SW
  - Komponentenumgebung (*Framework*) erlaubt durch API-Standardisierung schnelle Entwicklung neuer Anwendungen (zusätzliche Funktionen, neue Clients, Reuse, ...)

    ➥ **Wer liefert den „Klebstoff"?**

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

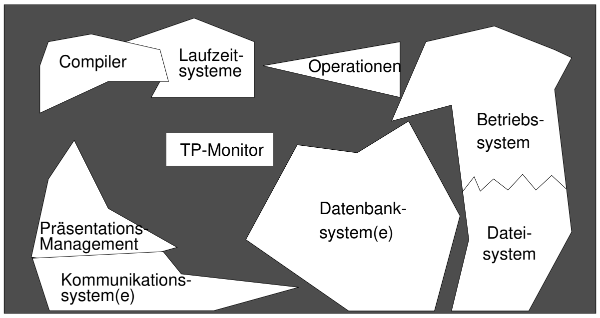


➥ TP-Monitor dirigiert und kontrolliert!

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 (BM), Protokolle usw. bei der Abwicklung von Transaktionen
  - realisieren und optimieren Funktionen, die von Betriebssystemen typischerweise nur sehr schlecht oder gar nicht unterstützt werden
  - verwalten Prozesse und starten/überwachen Programme (Dienste). Sie erlauben die Integration unabhängiger Dienste und ihre Abwicklung als Transaktionen

- **TP-Monitor[1] und zugeordnete Systemkomponenten**

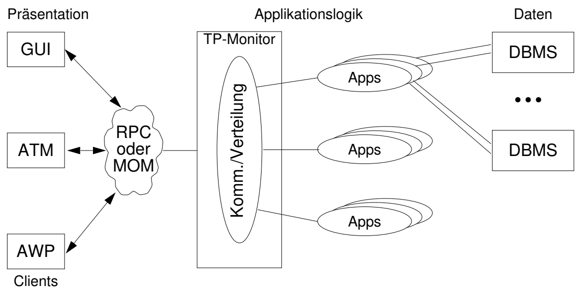


Der TP-Monitor integriert verschiedenartige Systemkomponenten, um eine gleichförmige Schnittstelle für Anwendungen und Operationen mit demselben Verhalten im Fehlerfall (*failure semantics*) zu bieten.

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

Was sind TP-Monitore? (2)

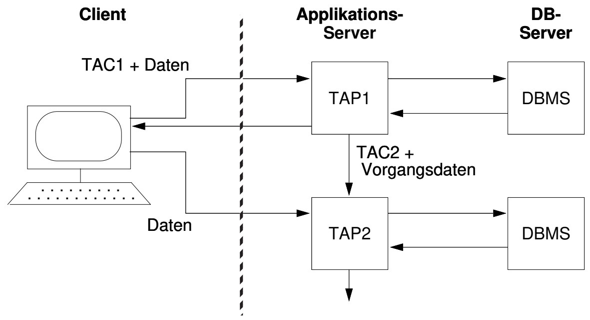
- **Neue TP-Monitor-Generation**[1]
  - ist „offen“ (standardisierte Schnittstellen zu Plattformen, Kommunikation, Benutzern)
  - läßt sich auf mehreren Betriebssystemen (Rechnerplattformen) einsetzen
  - ist Client-/Server-basiert
- **Was tun TP-Monitore im C/S-Umgebungen?**
  - Sie verwalten Transaktionen 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



1. 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 arbeiten TP-Monitor-basierte Anwendungen?**
  - 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, wiederbenutzbare Dienste entwikkeln, 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 abgewikkelt werden (ein oder mehrere Prozesse pro Server-Klasse)
  - Sie führen auf Server-Seite einen **ereignisgetriebenen Programmierstil** ein (TACs initiieren TAPs)
- **Prinzipieller Ablauf**



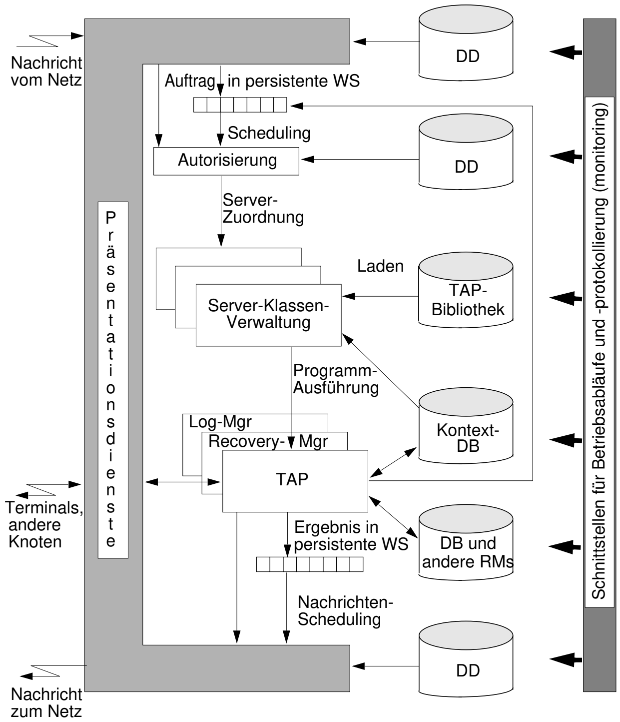
Was sind TP-Monitore? (4)

- **Was ist nun genau ein TP-Monitor?**
  - Er läßt sich als Betriebssystem 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 Initieren von TAPs,
      das Kontrollieren ihres Ablaufs und
      die Lastbalancierung ein
    - **Transaktionsverwaltung**[1]**:**
      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, ...

1. 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)



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

Struktur eines TP-Monitors (2)

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

    ➥ wegen der vielen verschiedenen Präsentationsdienste (Window-Protokolle, Graphikstandards) implementieren oft eigene RM diese Dienste

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

    ➥ Der WS-Mgr ist i. allg. als RM implementiert, der zum TP-Monitor gehört

- **Server-Klassen-Verwaltung**
  - Zuständig dafür, daß für jedes TAP eine Server-Klasse definiert und aktiv ist
  - Server werden entweder ‘by default’ (Systemstart) oder ‘on demand’ aktiviert
  - Aufgaben: Erzeugen von Prozessen und WS, Laden der TAP-Codes in die entsprechenden Adreßräume, Besorgen der Zugriffsrechte für die Zugriffe der Prozesse auf die WS, Festlegen der Server-Prioritäten u.a.

    ➥ Diese Dienste fallen auch in die Verantwortlichkeit der Lastbalancierung. Sie sind deshalb in enger Kooperation zu lösen. 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

    ➥ wegen der spezifischen Betriebscharakteristika von TA-Systemen, ist die dynamische Autorisierung für jeden individuellen Auftrag sehr wichtig

- **Kontextverwaltung**
  - 1. Speicherung von Verarbeitungskontexten, die **über TA-Grenzen** gehalten werden sollen (Mehr-TA-Vorgänge)

    ➥ 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 die Nachricht beim Server-Aufruf zu aufwendig; Kontextverwaltung speichert die Daten zwischen und erlaubt nachfolgende Server den Zugriff auf die Daten

    ➥ Diese Zugriffe erfolgen innerhalb einer TA; deshalb sind hier keine persistenten Kontexte erforderlich

Struktur eines TP-Monitors (4)

- **Metadatenverwaltung**

  Hier wird ein globales Repository (DD, Data Dictionary) angenommen. Ein TP-Monitor braucht folgende Metadaten:
  - Die zum verteilten TA-System gehörigen Knoten: Namen, Adressen, . . .
  - Lokale 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
  - Autorisierungs-Information: z. B. Zugriffskontrollisten für TAPs und RMs
  - Konfigurationsdaten für Server-Klassen über Prozesse, Tasks, Prioritäten
  - Daten über die dem System bekannten Benutzer; Autorisierungs-Codes (Paßwörter), Sicherheitsprofile, . . .
  - Konfigurationsdaten über Betriebs- und Administrations-Schnittstellen
  - Wiederanlauf-Konfigurationen und -Prozeduren (Restart-Reihenfolge der RMs)

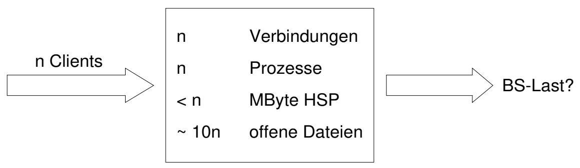
➥ Der Inhalt diese Kataloge wird gewartet und aktualisiert über System-TAs. Beim Restart wird versucht, die hier beschriebene Konfiguration “hochzufahren”

Ablauf in dreistufigen C/S-Architekturen

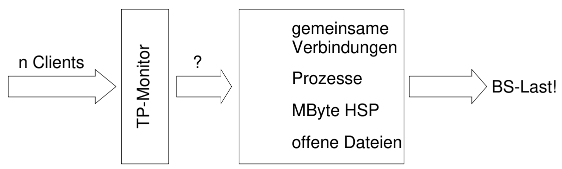
- **Typischer BM-Bedarf pro Client auf einem Server**
  - eine Kommunikationsverbindung
  - 0.5 - 1 MByte Hauptspeicher (RAM)
  - 1 oder 2 Prozesse
  - ~ 10 offene Dateikontrollblöcke (file handle)

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

- **Herkömmliche BS-Abbildung**



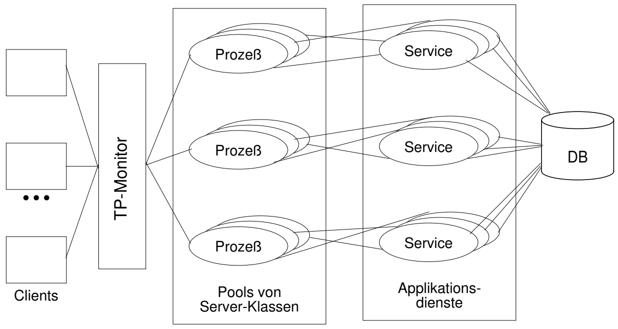
- **Einsatz eines TP-Monitors[1]**



1. “TP-Monitor: The 3-Tier Workhorse” (Jeri Edwards)

Ablauf in dreistufigen C/S-Architekturen (2)

- **BM-Zuteilung erfolgt auftragsbezogen**
  - OLTP-Applikation besteht typischerweise aus einer Menge von Diensten (TAPs)
  - Eine Server-Klasse besteht aus einem Pool von (statisch erzeugten) Prozessen oder Threads, welche die vorab geladenen TAPs abwickeln können. Der TP-Monitor kann dynamische neue Prozesse starten (Lastbalancierung)
  - Eine Applikation kann eine oder mehrere Server-Klassen besitzen
  - Der TP-Monitor weist einer Server-Klasse einen ankommenden Auftrag (TAC) zu. Nach Abwicklung werden die auftragsbezogenen BM wieder freigegeben. Der 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
  - . . .

Aufbau des DB-Severs

- **Allgemeine Aufgaben/Eigenschaften von DBS**
  - Verwaltung von persistenten Daten (lange Lebensdauer)
  - effizienter Zugriff (Suche und Aktualisierung) auf große Mengen von Daten (GBytes – PBytes)
  - flexibler Mehrbenutzerbetrieb
  - Verknüpfung / Verwaltung von Objekten verschiedenen Typs (➥ typübergreifende Operationen)
  - Kapselung der Daten und ihre Isolation von den Anwendungen (➥ möglichst hoher Grad an Datenunabhängigkeit)
  - . . .

- **Datenmodell / DBS-Schnittstelle (SQL-API)**

Schema: **ANGESTELLTER**

Satztyp (Relation)

Ausprägungen

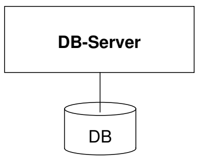
PNR	NAME	TAETIGKEIT	GEHALT	ALTER
496	PEINL	PFOERTNER	2100	63
497	KINZINGER	KOPIST	2800	25
498	MEYWEG	KALLIGRAPH	4500	56

  - Operationen zur Definition von Objekttypen (Beschreibung der Objekte)
    ➥ DB-Schema: Welche Objekte sollen in der DB gespeichert werden?
  - Operationen zum Aufsuchen und Verändern von Daten
    ➥ AW-Schnittstelle: Wie erzeugt, aktualisiert und findet man DB-Objekte?
  - Definition von Integritätsbedingungen (*Constraints*)
    ➥ Sicherung der Qualität: Was ist ein akzeptabler DB-Zustand?
  - Definition von Zugriffskontrollbedingungen
    ➥ Maßnahmen zum Datenschutz: Wer darf was?

Aufbau des DB-Severs (2)

- **Wie werden diese System- und Schnittstelleneigenschaften technisch umgesetzt?**
  - Komplexität des DBS
  - Modularität, Portabilität, Erweiterbarkeit, Zuverlässigkeit, . . .
  - Laufzeiteigenschaften (Leistung[1], Betriebsverhalten, Fehlertoleranz, . . .)
- **Ist ein monolithischer Ansatz sinnvoll?**
  - Beliebige mengenorientierte Operationen auf abstrakten Objekten (Tabellen) müssen „auf einmal" umgesetzt und abgewickelt werden!
  - DBS-Schnittstelle: Beispiel

```
Select *
From    ANGESTELLTER P, ABTEILUNG A, . . .
Where   P.GEHALT > 8000 AND ALTER < 25 AND P.ANR = A.ANR . . .
```



  - Schnittstelle zum Externspeicher: Lesen und Schreiben von Seiten (DB ist ein sehr langen Bitstring!)
- **Außerdem: Alles ändert sich ständig!**
  - DBS-Software hat eine Lebenszeit von > 20 Jahren
  - permanente Evolution des Systems
    - wachsender Informationsbedarf: Objekttypen, Integritätsbedingungen, ...
    - neue Speicherungsstrukturen und Zugriffsverfahren, ...
    - schnelle Änderungen der eingesetzten Technologien: Speicher, ...

1. Denn für DBS und ihre Anwendungen gilt folgender populäre Spruch in besonderer Weise: „Leistung ist nicht alles, aber ohne Leistung ist alles nichts".

Schichtenmodelle für DBS

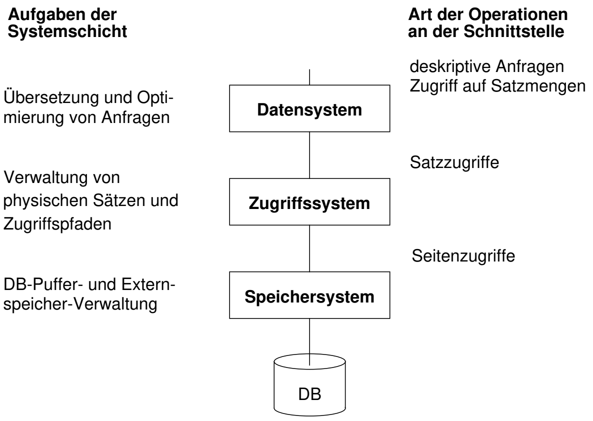
- **Deshalb als wichtigstes Entwurfsziel: Architektur eines datenunabhängigen DBS**
- **Systementwurf**
  - Was sind die geeigneten Beschreibungs- und Kommunikationstechniken? Sie sind notwendigerweise informal.
  - Was ist auf welcher Beschreibungsebene sichtbar? Es ist angemessene Abstraktion erforderlich![1]
  - Wie kann eine Evolution des Systems erfolgen? Es muß eine Kontrolle der Abhängigkeiten erfolgen!
- **Aufbau in Schichten:**
  - „günstige Zerlegung" des DBS in „nicht beliebig viele" Schichten
  - optimale Bedienung der Aufgaben der darüberliegenden Schicht
  - implementierungsunabhängige Beschreibung der Schnittstellen

    ➥ Es gibt keine Architekturlehre für den Aufbau großer SW-Systeme
- **Empfohlene Konzepte:**
  - Geheimnisprinzip (*Information Hiding*)
  - hierarchische Strukturierung
  - generische Auslegung der Schnittstellen:
    Nur bestimmte Objekttypen mit charakteristischen Operationen sind vorgegeben, jedoch nicht ihre anwendungsbezogene Spezifikation und Semantik

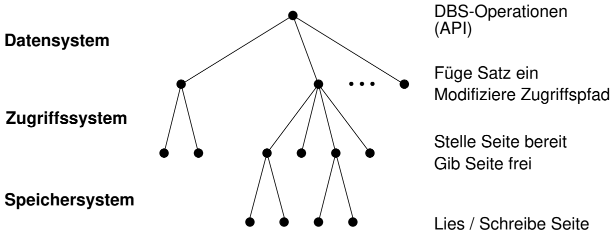
1. „Die durch Abstraktion entstandenen Konstrukte der Informatik als Bedingungen möglicher Information sind zugleich die Bedingungen der möglichen Gegenstände der Information in den Anwendungen" (H. Wedekind in Anlehnung an eine Aussage Kants aus der „Kritik der reinen Vernunft")
Vereinfacht ausgedrückt: Informatiker erfinden (konstruieren) abstrakte Konzepte; diese ermöglichen (oder begrenzen) wiederum die spezifischen Anwendungen.

Schichtenmodelle für DBS (2)

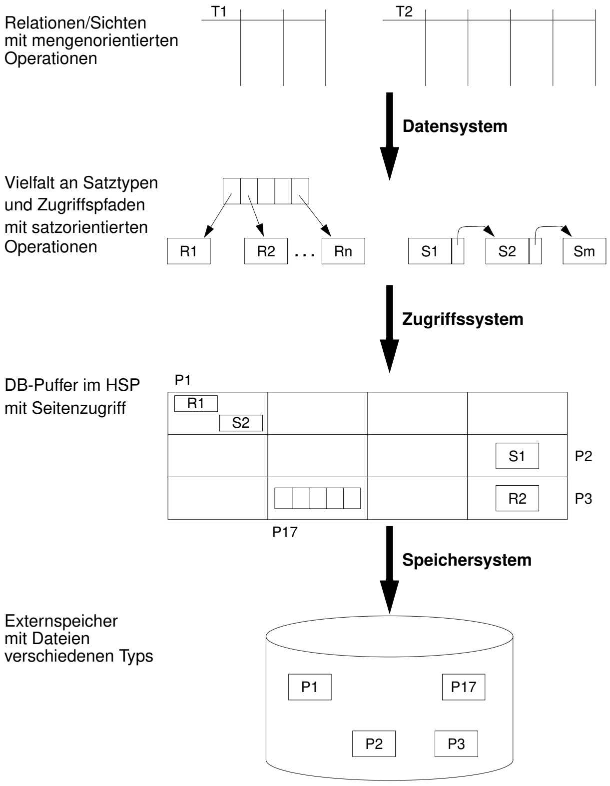
- **Vereinfachtes Schichtenmodell**



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



Drei-Schichten-Modell – Abbildungen



Architektur eines DBS – weitere Komponenten

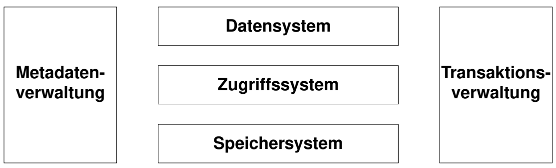
- **Entwurfsziel:**
  DBS müssen von ihrem Aufbau und ihrer Einsatzorientierung her in hohem Maße generische Systeme sein. Sie sind so zu entwerfen, daß sie flexibel durch Parameterwahl und ggf. durch Einbindung spezieller Komponenten für eine vorgegebene Anwendungsumgebung zu konfigurieren sind

- **Rolle der Metadaten**
  - Metadaten enthalten Informationen über die zu verwaltenden Daten
  - Sie beschreiben also diese Daten (Benutzerdaten) näher hinsichtlich Inhalt, Bedeutung, Nutzung, Integritätsbedingungen, Zugriffskontrolle usw.
  - Die Metadaten lassen sich unabhängig vom DBS beschreiben (für alle Schichten: Daten-, Zugriffs- und Speichersystem)

  ➥ Dadurch erfolgt das „Zuschneidern eines DBS“ auf eine konkrete Einsatzumgebung. Die separate Spezifikation, Verwaltung und Nutzung von Metadaten bildet die Grundlage dafür, daß DBS hochgradig „generische“ Systeme sind

- **Verwaltung der Daten, die Daten beschreiben:**
  - Metadaten fallen in allen DBS-Schichten an
  - Metadatenverwaltung, DB-Katalog, Data-Dictionary-System, DD-System, ...

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

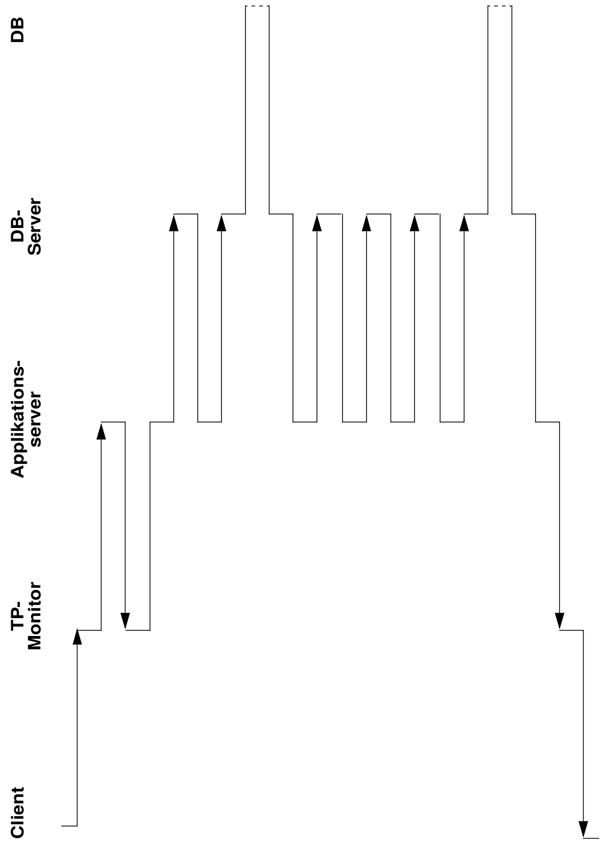


Ablaufbeispiel

- **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 (write back)
  - Alle Log-Daten werden bei Commit auf einmal geschrieben



Zusammenfassung

- **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**

Pro	Contra
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 Skalierbarbeit	unausgereifte Sicherheitsmechanismen
Entkopplung durch Dezentralisierung	Trainingsaufwand
Entsprechung organisatorischer Strukturen	wichtige Anwendungen nicht verfügbar

- **TP-Monitore**
  - liefern den „Klebstoff“ in komplexen C/S-Umgebungen
  - erlauben die Umsetzung des Transaktionskonzeptes als Grundlage zur Realisierung zuverlässiger verteilter Systeme

- **Schichtenmodell für den DB-Server**
  - Es ist allgemeines Erklärungsmodell für die DBS-Realisierung
  - Schichtenbildung läßt sich zweckorientiert verfeinern