

# **Client/Server-Systeme**

**Prof. Dr.-Ing. Wilhelm G. Spruth**

**WS 2005/06**

**Teil 8**

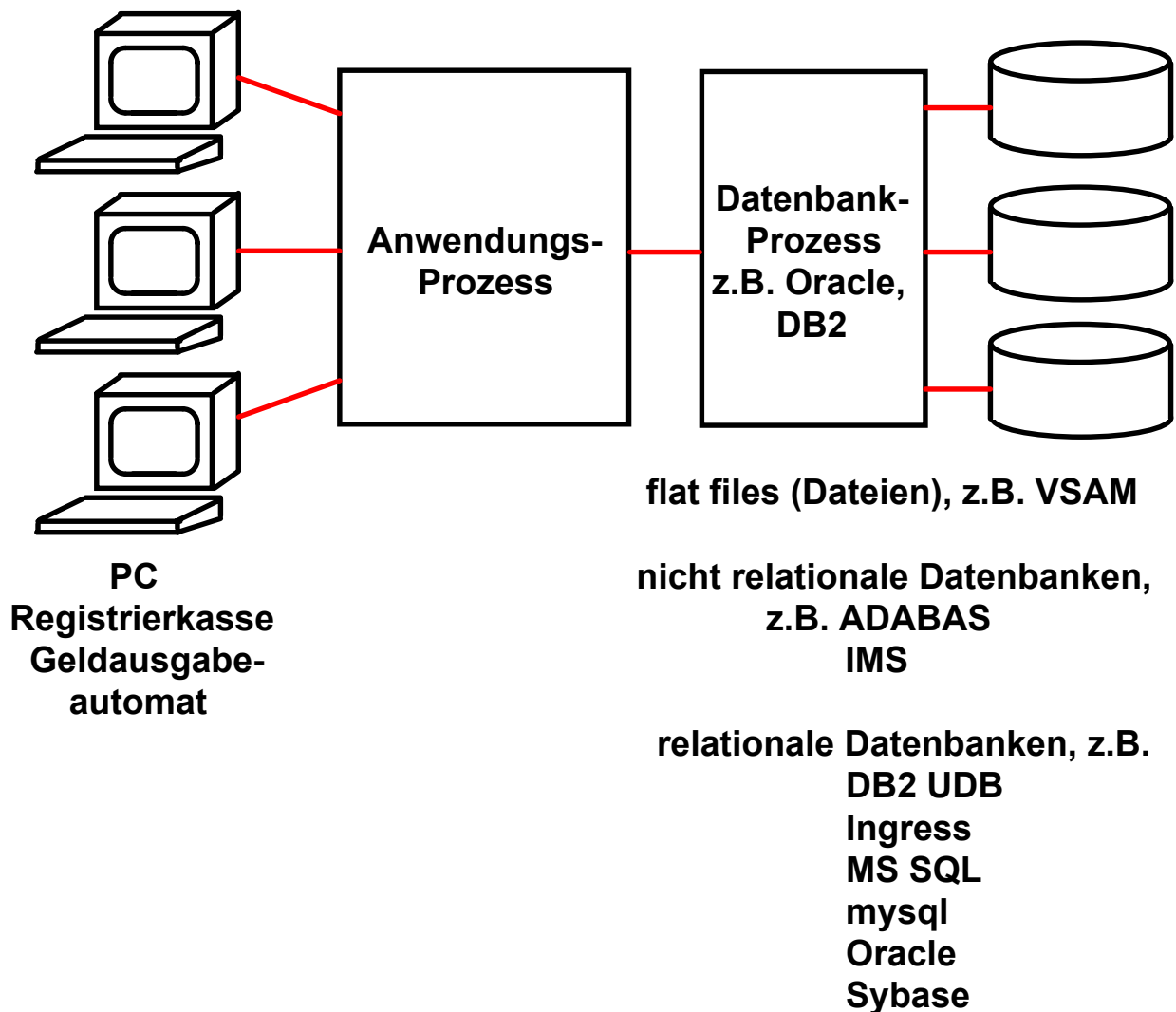
**Stored Procedures**

# Datenbanken

**Eine Datenbank besteht aus einer Datenbasis (normalerweise Plattenspeicher) und Verwaltungsprogrammen (Datenbank Software), welche die Daten entsprechend den vorgegebenen Beschreibungen abspeichert, auffindet, oder beliebige weitere Operationen mit den Daten durchführt.**

**Wenn wir im Zusammenhang mit Client/Server Systemen von einer Datenbank sprechen meinen wir in der Regel damit die Verwaltungsprogramme (Datenbank im engeren Sinne).**

**Ein Datenbankprozess läuft fast immer in einem eigenen virtuellen Adressenraum. Häufig sind es sogar mehrere virtuelle Adressenräume (z.B. drei im Fall von OS/390 DB2).**

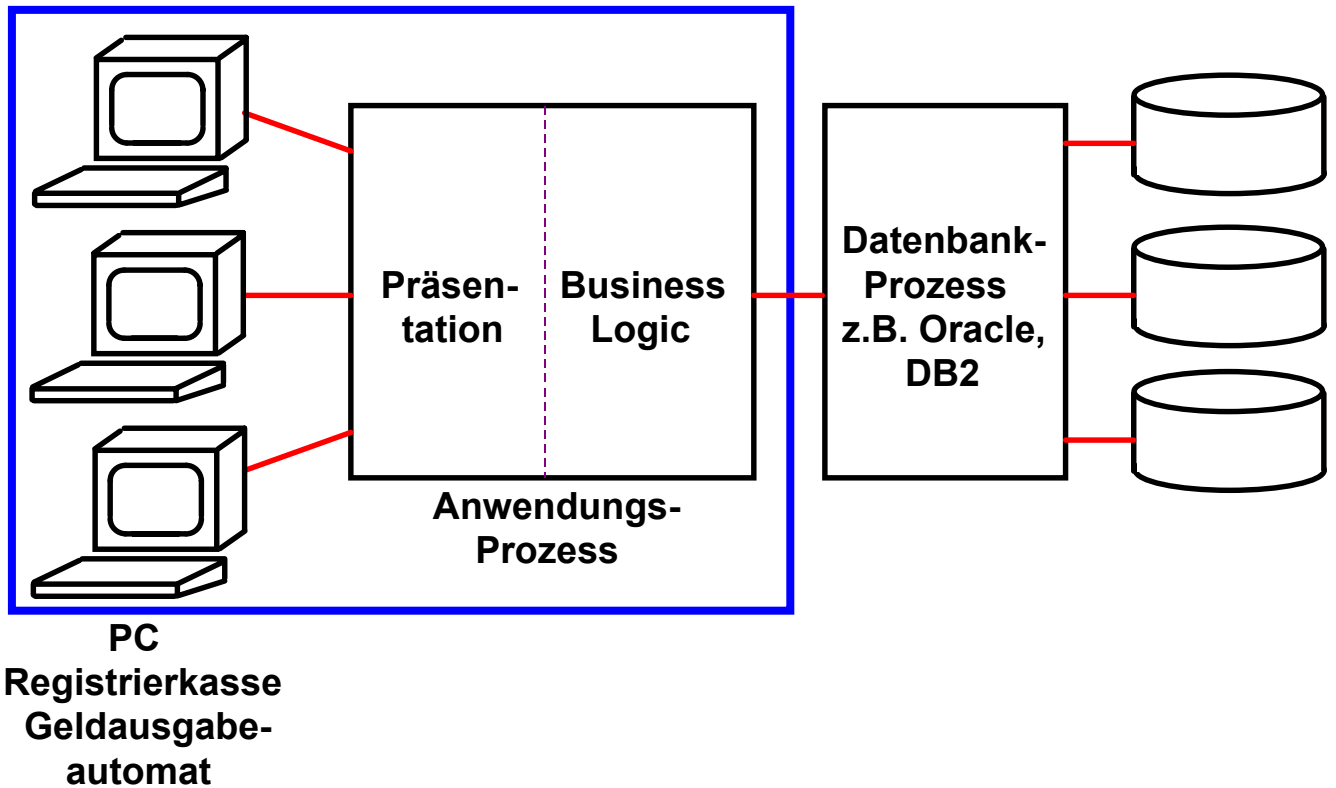


## Datenbank Server in einer typische Client/Server Anwendung

Dargestellt ist eine logische Struktur. 2-Tier, 3-Tier oder n-Tier Konfigurationen unterscheiden sich dadurch, wie diese Funktionen auf physikalische Server abgebildet werden.

Im einfachsten Fall (Beispiel Diplomarbeit) sind Anwendung und Datenbank getrennte Prozesse auf dem gleichen Rechner wie der Klient.

## Fat Client

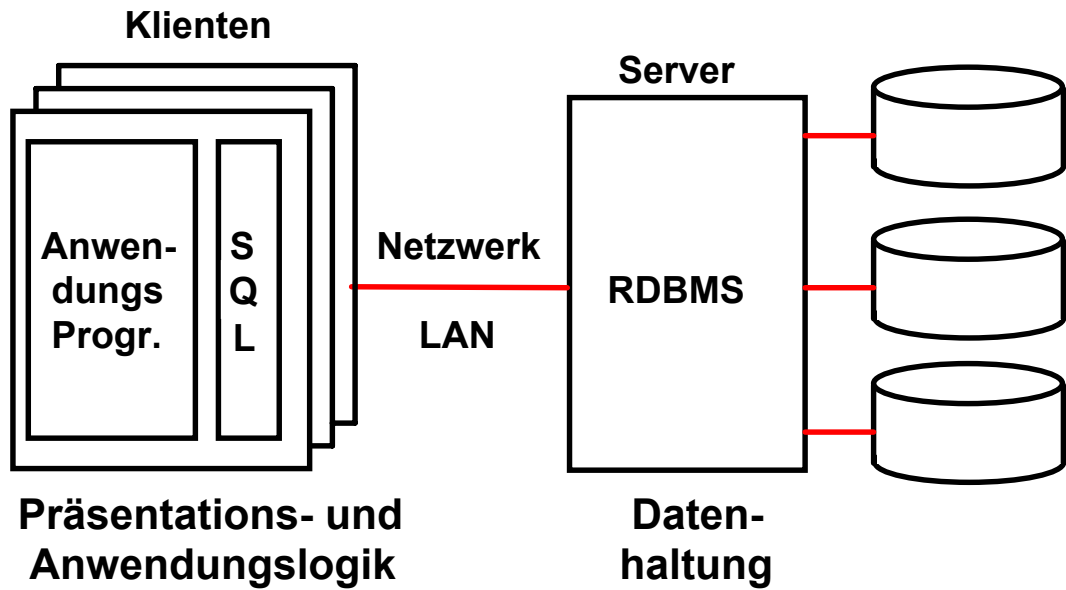


## Typische Client/Server Anwendung

**Business Logic (Anwendungslogik) verarbeitet die Eingabedaten des Endbenutzers und erzeugt Ausgabedaten für den Endbenutzer, z.B. in der Form einer wenig strukturierten Zeichenkette.**

**Präsentationslogik formt die rohen Ausgabedaten in eine für den Endbenutzer gefällige Form um.**

**Anwendungsprozess und Datenbankprozess laufen in getrennten virtuellen Adressenräumen, ggf. auch auf getrennten physikalischen Servern.**



**Annahmen:**

**< 200 Klienten**

**< 100 000 Transaktionen / Tag**

**LAN Umgebung**

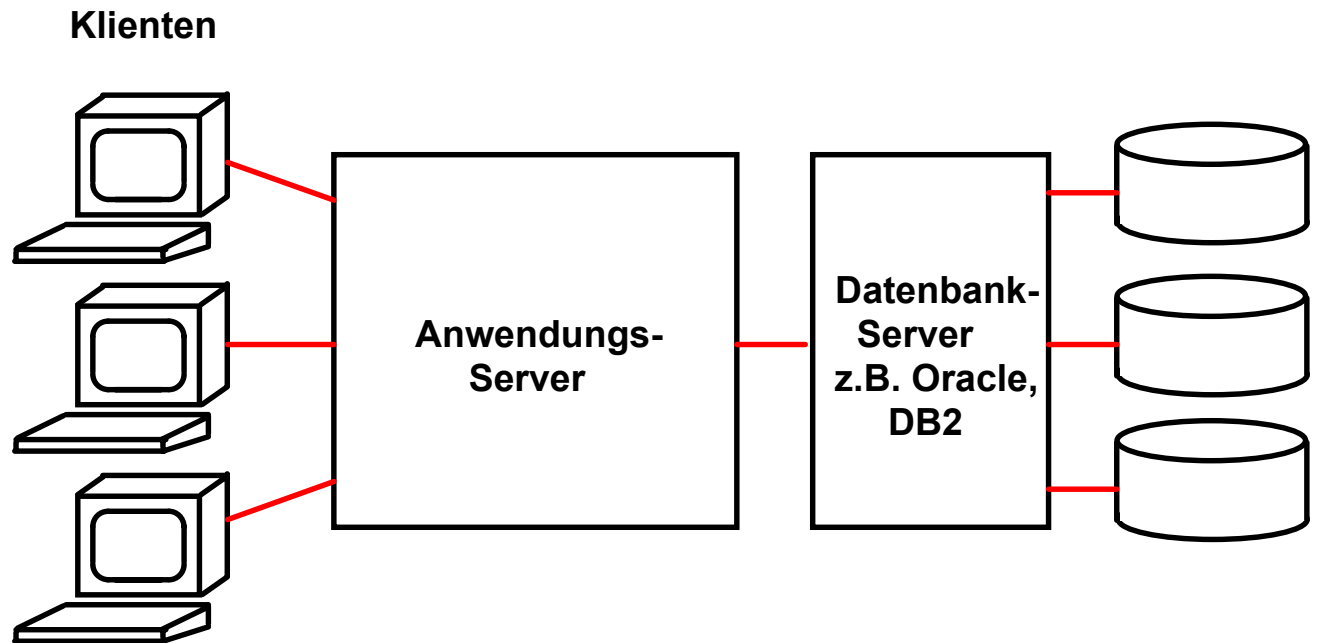
**1 oder wenige Server**

**Mäßige Sicherheitsanforderungen**

## **2-Tier Client/Server Architektur**

### **Zweistufige Client/Server Architektur**

**Anwendungsentwicklung z.B. mit Power Builder oder Visual Basic**



## **3-Tier Client/Server Architektur**

### **Dreistufige Client/Server Architektur**

#### **Zwei Arten**

**Präsentationslogik läuft auf dem Klienten**  
**Beispiel: SAPGUI des SAP R/3 Systems**

**Präsentationslogik läuft auf dem**  
**Anwendungsserver**  
**Beispiel: Servlets und Java Server Pages**

# Transaktionen

**Transaktionen sind Client-Server-Anwendungen, welche die auf einem Server gespeicherten Daten von einem definierten Zustand in einen anderen überführen.**

**Eine Transaktion ist eine atomare Operation. Die Transaktion wird entweder ganz oder gar nicht durchgeführt.**

**Eine Transaktion ist die Zusammenfassung von mehreren Datei- oder Datenbankoperationen, die entweder**

**erfolgreich abgeschlossen wird, oder die Datenbank unverändert läßt**

**Die Datei/Datenbank bleibt in einem konsistenten Zustand: Entweder vor Anfang oder nach Abschluß der Transaktion**

**Im Fehlerfall, oder bei einem Systemversagen werden alle in Arbeit befindlichen Transaktionen abgebrochen und alle evtl. bereits stattgefundenen Datenänderungen automatisch rückgängig gemacht.**

**Wird eine Transaktion abgebrochen, werden keine Daten abgeändert**

# **ACID Eigenschaften**

**Atomizität (Atomicity)**

**Konsistenzerhaltung (Consistency)**

**Isolation**

**Dauerhaftigkeit (Durability)**

# ACID Eigenschaften

## Atomizität (Atomicity)

**Eine Transaktion wird entweder vollständig ausgeführt oder überhaupt nicht**

**Der Übergang vom Ursprungszustand zum Ergebniszustand erfolgt ohne erkennbare Zwischenzustände, unabhängig von Fehlern oder Crashes. Änderungen betreffen Datenbanken, Messages, Transducer und andere.**

## Konsistenzerhaltung (Consistency)

**Eine Transaktion überführt die transaktionsgeschützten Daten des Systems von einem konsistenten Zustand in einen anderen konsistenten Zustand.**

**Diese Eigenschaft garantiert, dass die Daten der Datenbank nach Abschluss einer Transaktion schemakonsistent sind, d. h. alle im Datenbankschema spezifizierten Integritätsbedingungen erfüllen**

**Daten sind konsistent, wenn sie durch eine Transaktion erzeugt wurden.**

## Isolation

**Die Auswirkungen einer Transaktion werden erst nach ihrer erfolgreichen Beendigung für andere Transaktionen sichtbar**

**Single User Mode Modell. Selbst wenn 2 Transaktionen gleichzeitig ausgeführt werden, wird der Schein einer seriellen Verarbeitung gewahrt.**

## Dauerhaftigkeit (Durability)

**Die Auswirkungen einer erfolgreich beendeten Transaktion gehen nicht verloren**

**Das Ergebnis einer Transaktion ist real, mit allen Konsequenzen. Es kann nur mit einer neuen Transaktion rückgängig gemacht werden. Die Zustandsänderung überlebt nachfolgende Fehler oder Systemcrashes.**

# Transactionssysteme

## Interaktive Beispiele:

**Auskunftssysteme**

**Buchungssysteme (z.B. Flugplatzreservierung)**

**Geldausgabeautomaten**

**Auftragsbearbeitung, Buchbestellung bei Amazon**

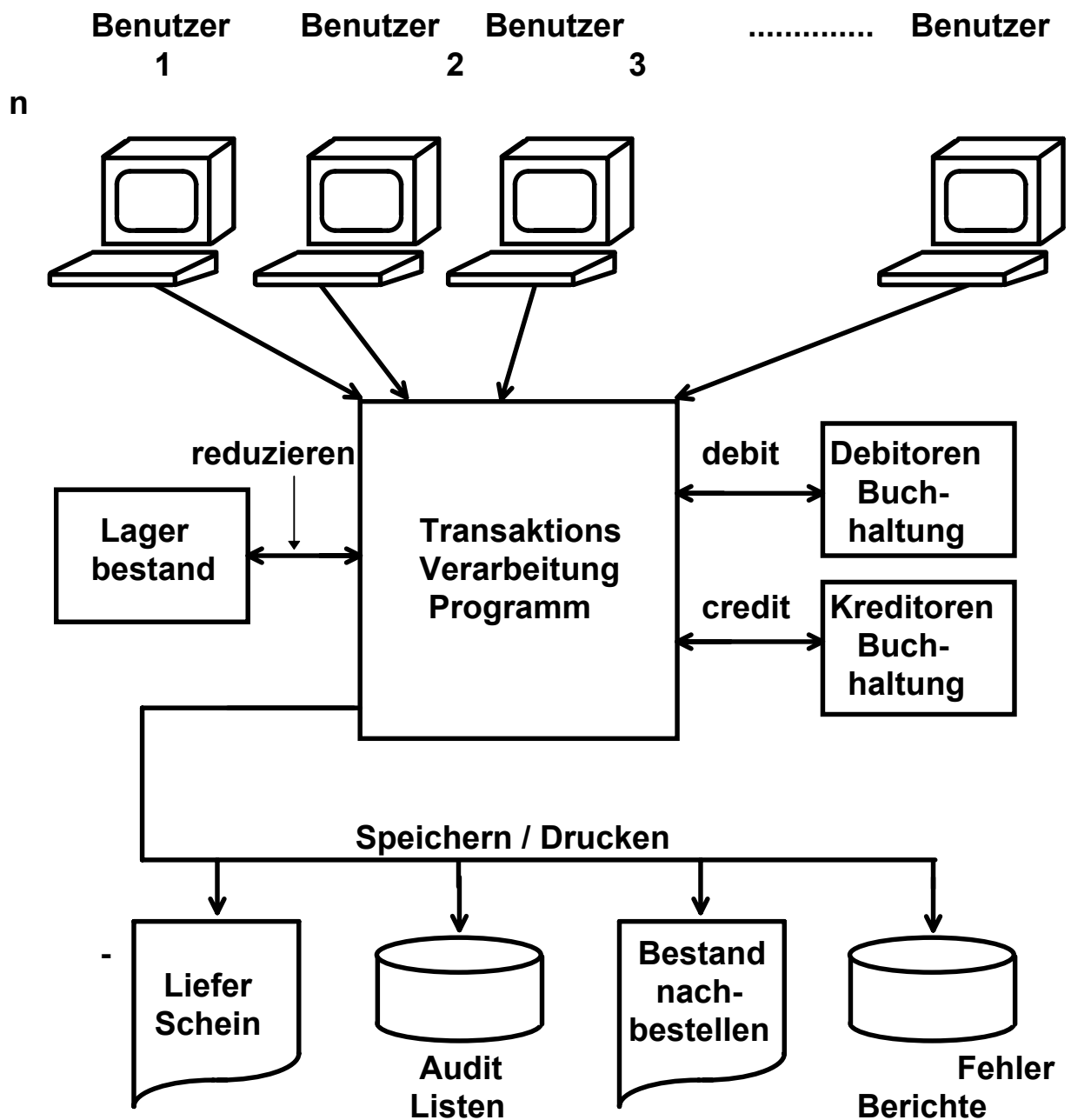
**Angebot bei eBay abgeben**

## Stapelverarbeitung Beispiele:

**Monatliche Lohn/Gehaltsabrechnung**

**Jährliche Bilanz erstellen**

**Rechnerbelastung interaktiv zu Stapel etwa 60 : 40**



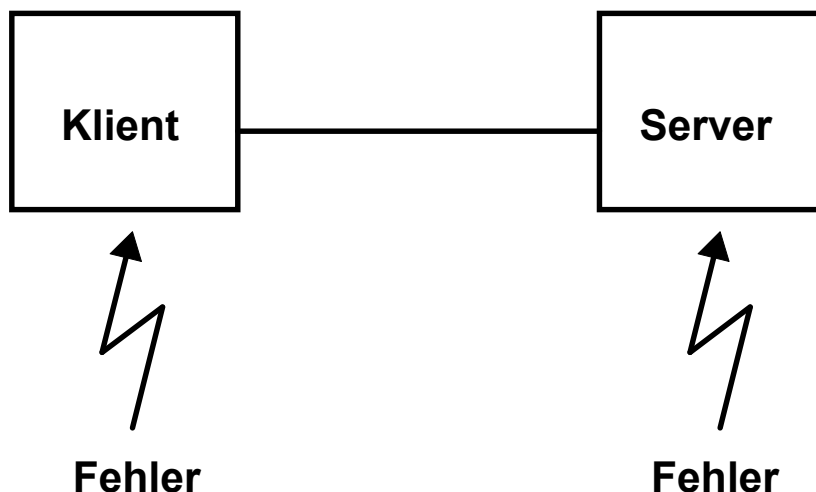
**Beispiel für eine  
Transaktionsverarbeitungsanwendung:  
Auftragseingang-Bearbeitung**

# Fehlerbehandlung

## Fehlermöglichkeiten

- **Server kann Prozedur nicht beenden**  
(z.B. Rechnerausfall, SW-Fehler)
- **Klient wird während des RPCs abgebrochen**  
(z.B. Rechnerausfall, SW-Fehler)

Im Gegensatz zum lokalen Prozeduraufruf sind Fehlerbehandlungsmaßnahmen erforderlich.



# Fehlerbehandlung - Ausfall des Servers

## 1. Idempotente Arbeitsvorgänge

"Idempotent" sind Arbeitsvorgänge, die beliebig oft ausgeführt werden können; Beispiel: Lese Block 4 der Datei xyz. Nicht idempotent ist die Überweisung eines Geldbetrages von einem Konto auf ein anderes.

## 2. Nicht-idempotente Arbeitsvorgänge

Problem: Wie wird festgestellt, ob Arbeitsgang durchgeführt wurde oder nicht.

"Genau einmal" (exactly once).

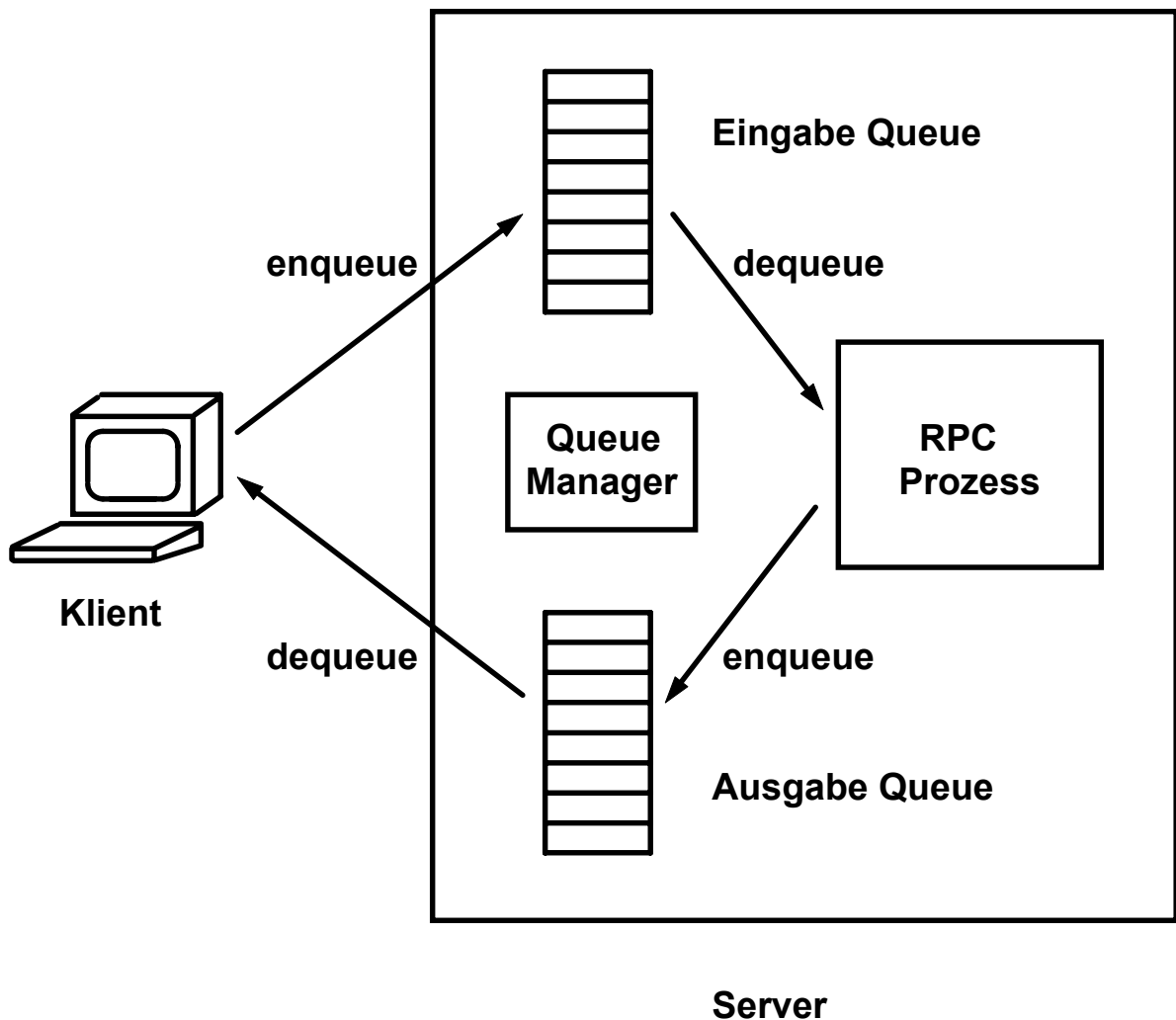
Überweisung Konto x y

"Höchstens einmal" (at most once).

Stimmabgabe Bundestagswahl

"Wenigstens einmal" (at least once). Ideal für idempotente Vorgänge.

Update Virenschanner



## Fehlerbehandlung - Ausfall des Klienten

Getrennte Warteschlangen für eingehende und ausgehende Nachrichten lösen viele Probleme. Wenn der Klient abstürzt (Waise), schließt der Server seine Arbeit ab. Das Ergebnis der RPC Verarbeitung steht in der Ausgabe Queue, und kann beim Wiederanlauf des Klienten dort abgeholt werden.

Andere Aufgaben der Queues:

- Prioritätensteuerung
- Lastverteilung auf mehrere Server

# Arten von Datenbanken

## Relationale Datenbanken

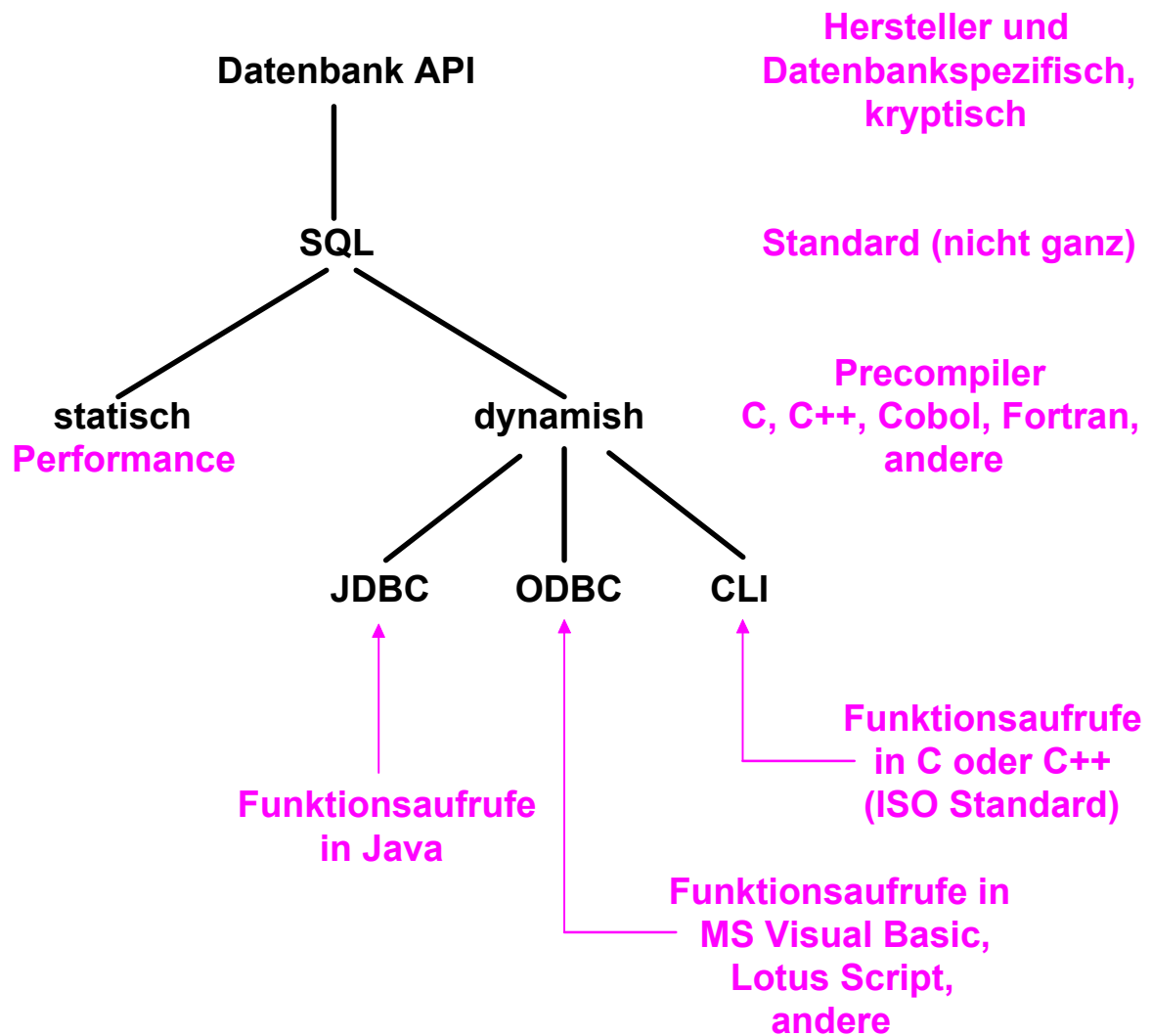
z.B. Oracle  
DB2  
MS SQL  
mySQL

## Nicht-relationale Datenbanken

z.B. IMS  
ADABAS

## Objekt orientierte Datenbanken (sinkende Bedeutung)

z.B. Poet



## Zugriff auf SQL Datenbank

Dynamisches SQL ermöglicht Angabe von Parametern erst zur Laufzeit. Z.B. Benutzer gibt gewünschten Datenbanknamen und Tabellennamen in Datenfeld einer HTML Seite ein.

DB2connect ist ein Klient für statische SQL Zugriffe Kann als Remote Procedure Call für SQL betrachtet werden. SQLJ ist die Java - spezifische Version von DB2connect.

```

#sql iterator SeatCursor(Integer row, Integer col,
    String type, int status);
Integer status = ?;
SeatCursor sc;
#sql sc = {
select rownum, colnum from seats where status <=
    :status
};
while(sc.next())
{
#sql { insert into categ values(: (sc.row()),
    : (sc.col())) };
}
sc.close();

```

**SQLJ**

```

Integer status = ?;
PreparedStatement stmt = conn.prepareStatement("select
    row, col from seats where status <= ?");
if (status == null) stmt.setNull(1,Types.INTEGER);
else stmt.setInt(1,status.intValue());
ResultSet sc = stmt.executeQuery();
while(sc.next())
{
int row = sc.getInt(1);
boolean rowNull = sc.wasNull();
int col = sc.getInt(2);
boolean colNull = sc.wasNull();
PreparedStatement stmt2 =
    conn.prepareStatement("insert into categ
        values(?, ?)");
if (rowNull) stmt2.setNull(3,Types.INTEGER);
else stmt2.setInt(3,rownum);
if (colNull) stmt2.setNull(4,Types.INTEGER);
else stmt2.setInt(4,colnum);
stmt2.executeUpdate();
stmt2.close();
}
sc.close();
stmt.close();

```

**JDBC**

**Für statisches SQL ist SQLJ eine höhere Programmiersprachen-Schnittstelle als JDBC. Benötigt weniger Zeilen Code.**

# Embedded SQL, Beispiel für C (1)

Ein Anwendungsprogramm implementiert typischerweise statische Datenbankzugriffe über eingebettete SQL-Anweisungen. Diese werden durch "exec sql" eingeleitet und durch ein spezielles Symbol (hier ";") beendet. Dies erlaubt einem Precompiler die Unterscheidung der exec sql Anweisungen von anderen Anweisungen.

```
main( )
{
    .....
    .....
    exec sql insert into PERS (PNR, PNAME) values (4711, 'Ernie');
    .....
    .....
}
```

## Advantages of static SQL

When a static SQL statement is prepared, an executable form of the statement is created and stored in the package in the database. The executable form can be constructed either at precompile time, or at a later bind time. In either case, preparation occurs *before* run time.

The authorization of the person binding the application is used, and optimization is based upon database statistics and configuration parameters that may not be current when the application runs.

Programming using static SQL requires less effort than using embedded dynamic SQL. Static SQL statements are simply embedded into the host language source file, and the precompiler handles the necessary conversion to database manager run time services API calls that the host language compiler can process.

Because the authorization of the person binding the application is used, the end user does not require direct privileges to execute the statements in the package. For example, an application could allow a user to update parts of a table without granting an update privilege on the entire table. This can be achieved by restricting the static SQL statements to allow updates only to certain columns or a range of values.

Static SQL statements are *persistent*, meaning that the statements last for as long as the package exists. Dynamic SQL statements are cached until they are invalidated, or freed for space management reasons, or the database is shut down. If required, the dynamic SQL statements are recompiled implicitly by the DB2 SQL compiler whenever a cached statement becomes invalid.

The key advantage of static SQL, with respect to persistence, is that the static statements exist after a particular database is shut down, whereas dynamic SQL statements cease to exist when this occurs. In addition, static SQL does not have to be compiled by the DB2 SQL compiler at run time, while dynamic SQL must be explicitly compiled at run time (for example, by using the PREPARE statement). Because DB2 caches dynamic SQL statements, the statements do not need to be compiled often by DB2, but they must be compiled at least once when you execute the application.

There can be performance advantages to static SQL. For simple, short-running SQL programs, a static SQL statement executes faster than the same statement processed dynamically since the overhead of preparing an executable form of the statement is done at precompile time instead of at run time.

```

exec sql include sqlca; /* SQL Communication Area*/
main ()
{
exec sql begin declare section;
    char X[8];
    int   GSum;
exec sql end declare section;
exec sql connect to dbname;
exec sql insert into PERS (PNR, PNAME) values (4711, 'Ernie');
exec sql insert into PERS (PNR, PNAME) values (4712, 'Bert');
printf ( "ANR ? " ) ; scanf(" %s", X);
exec sql select sum (GEHALT) into :GSum from PERS
    where ANR = :X;
printf ("Gehaltssumme %d\n", GSum)
exec sql commit work;
exec sql disconnect;
}

```

## Embedded SQL, Beispiel für C (2)

### Anmerkungen

Eingebettete SQL-Anweisungen werden durch "EXEC SQL" eingeleitet und durch spezielles Symbol (hier ";") beendet, um einem Precompiler die Unterscheidung von anderen Anweisungen zu ermöglichen

Die Oracle und DB/2 Precompiler greifen sich die exec sql Befehle heraus und übersetzen sie in Anweisungen, die der C-Compiler versteht.

Die connect Anweisung baut die Verbindung zwischen Klienten und Server auf.

Es wird eine Kopie von einem Teil der DB Tabelle erstellt, gegen die alle SQL Befehle Änderungen vornehmen. Die commit Anweisung macht die vorhergehenden SQL Befehle entgültig.

Kommunikationsbereich SQLCA (Rückgabe von Statusanzeigern u.ä.)

```

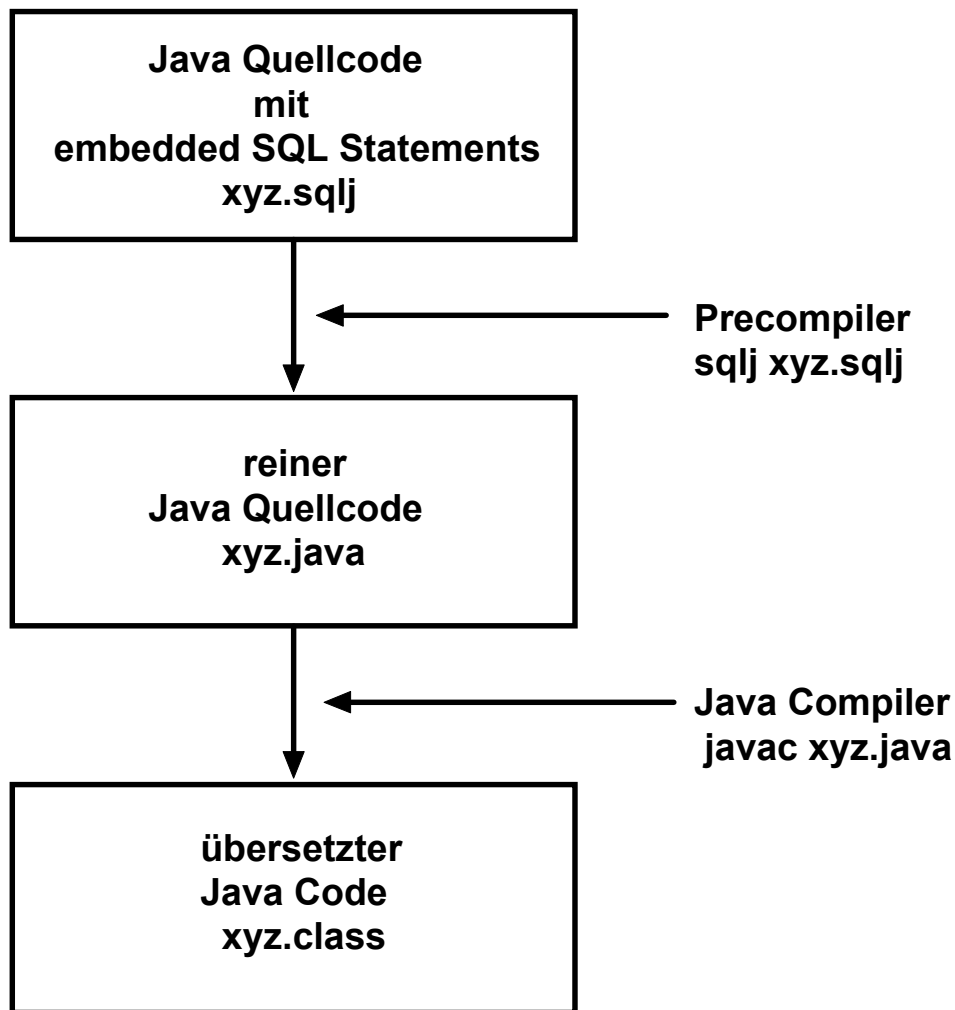
main()
{
    EXEC SQL DECLARE C1 CURSOR FOR
        SELECT VNAME,NNAME FROM ZEYPRAK.TABPRAK;
    EXEC SQL OPEN C1;
    EXEC SQL FETCH C1 INTO :vname, :nname;
    memcpy(liste.listei.vnam1i,vname,20);
    memcpy(liste.listei.nnam1i,nname,20);
    EXEC SQL FETCH C1 INTO :vname, :nname;
    memcpy(liste.listei.vnam2i,vname,20);
    memcpy(liste.listei.nnam2i,nname,20);
    EXEC SQL FETCH C1 INTO :vname, :nname;
    memcpy(liste.listei.vnam3i,vname,20);
    memcpy(liste.listei.nnam3i,nname,20);
    EXEC SQL FETCH C1 INTO :vname, :nname;
    memcpy(liste.listei.vnam4i,vname,20);
    memcpy(liste.listei.nnam4i,nname,20);
    EXEC SQL CLOSE C1;

    EXEC CICS SEND MAP("liste") MAPSET("prakset") ERASE;

}

```

## Embedded SQL, Beispiel für C (3)



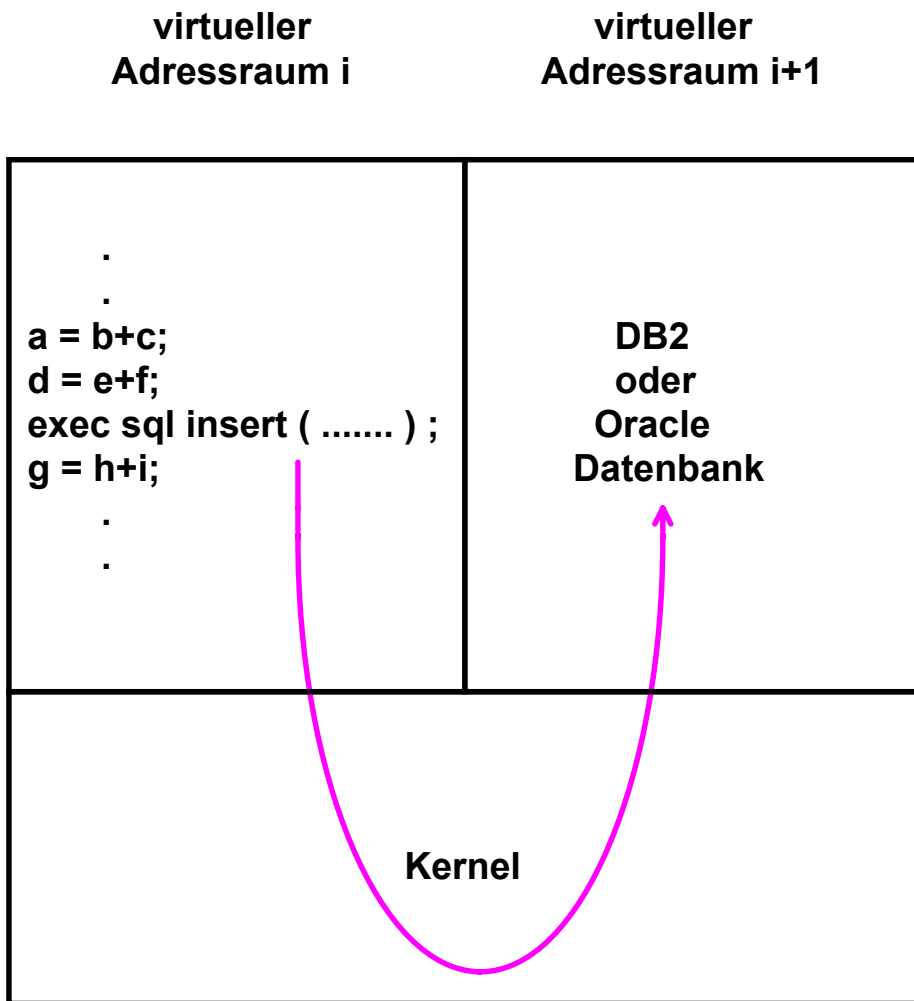
## Beispiel: SQLJ Precompiler

Embedded SQL bedeutet, wir fügen SQL Kommandos in normale Programmiersprachen wie Cobol, C/C++ oder Java ein.

Der Cobol, C/C++ oder Java Compiler versteht die SQL Kommandos nicht. Sie müssen zunächst mit einem *Pre-Compiler* in Funktionsaufrufe der entsprechenden Programmiersprache übersetzt werden.

Der Pre-Compiler übersetzt embedded SQL Kommandos in entsprechende Datenbank API Calls.

### Unterschiedliche Precompiler für DB2, Oracle, Sybase



## SQL Datenbank Zugriff auf dem gleichen Rechner

In einem Datenbanksystem wie Oracle oder DB2 hat der sql insert Aufruf ACID Eigenschaften

**Rechner 1**

**Rechner 2**

```
a = b+c; .  
d = e+f;  
exec sql insert ( ..... ) ;  
g = h+i;  
. .
```

**DB2  
oder  
Oracle  
Datenbank**



---

**Kommunikationsmechanismus TCP/IP, NetBios, IPX oder SNA,  
mit Socket, RPC, SMB, APPC oder CPI-C Protokoll**

**Der Precompiler erzeugt für den SQL Aufruf  
Client-Code für den Datenbankserver**

# Embedded SQL Transaktion

```
DebitCreditApplication( )
{
    receive input message;
    exec sql BEGIN WORK;                               /* start transaction */
    Abalance = DoDebitCredit (.....);
        .
        .
        .
    if (Abalance < 0 && delta < 0)
        { exec sql ROLLBACK WORK; }
    else {
        send output message;
        exec sql COMMIT WORK;                          /* end transaction */
    }
}
```

## Anmerkung

Der Rechner führt die Anweisungen zwischen

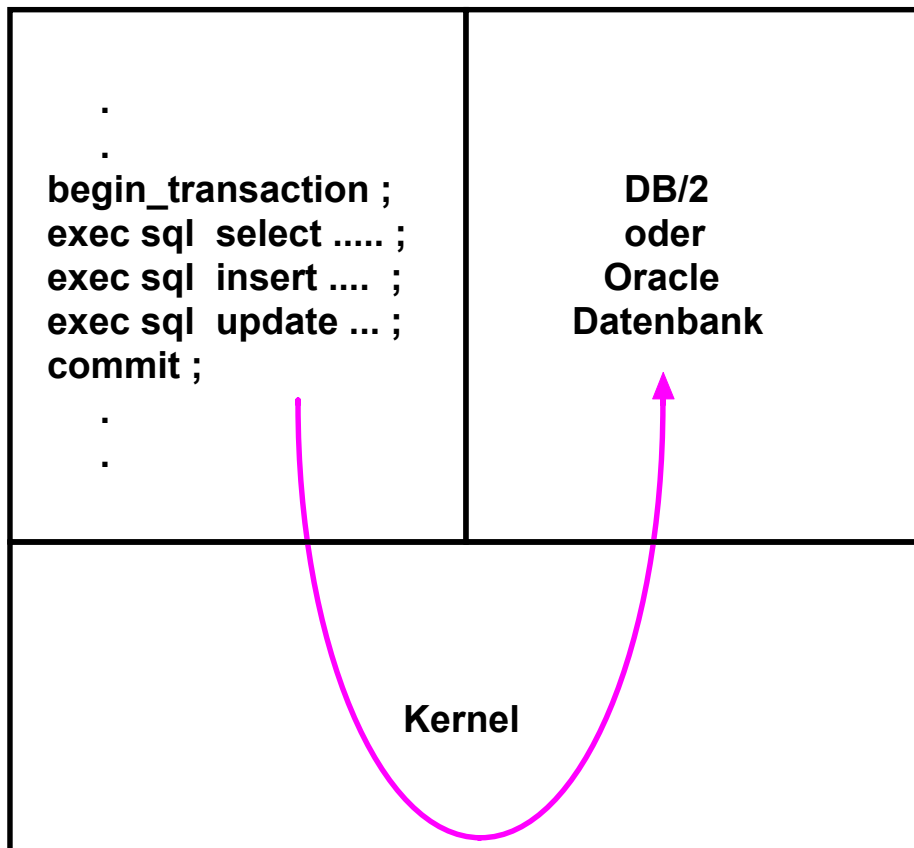
**exec sql BEGIN WORK**

und

**exec sql COMMIT WORK**

als Arbeitseinheit (Work Unit) entsprechend den ACID Anforderungen aus.

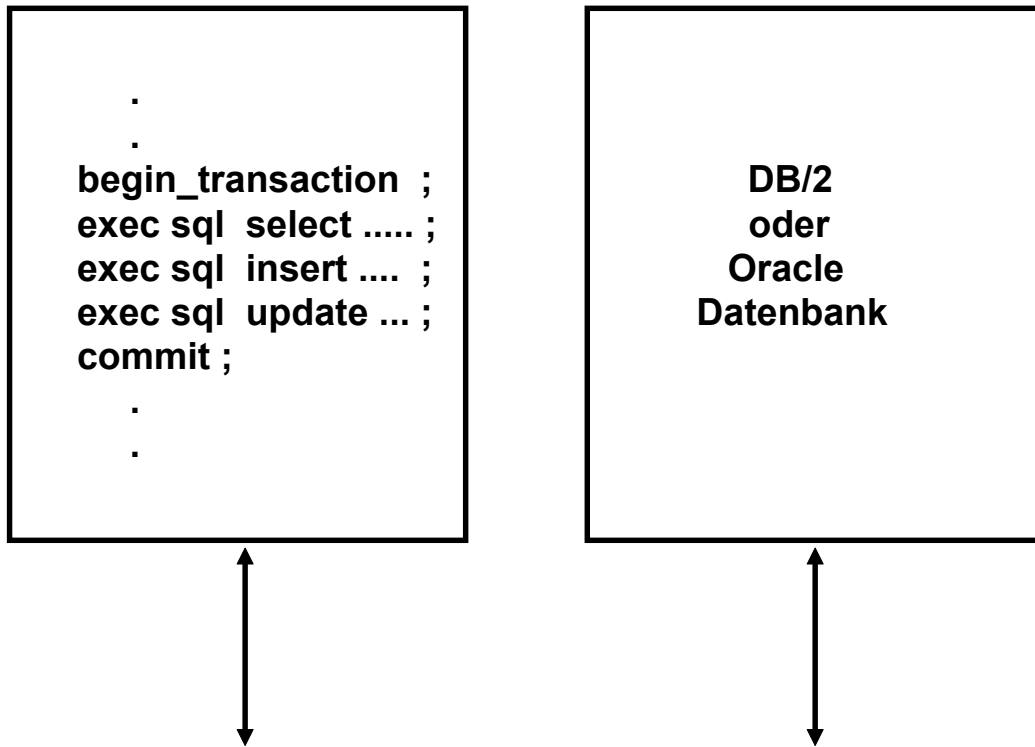
virtueller Adressraum i    virtueller adressraum i+1



## ACID Eigenschaften für eine Gruppe von SQL Aufrufen

Rechner 1

Rechner 2



Kommunikationsmechanismus TCP/IP oder SNA,  
mit Socket, RPC, APPC oder CPI-C Protokoll

## ACID Eigenschaften für eine Gruppe von SQL Aufrufen

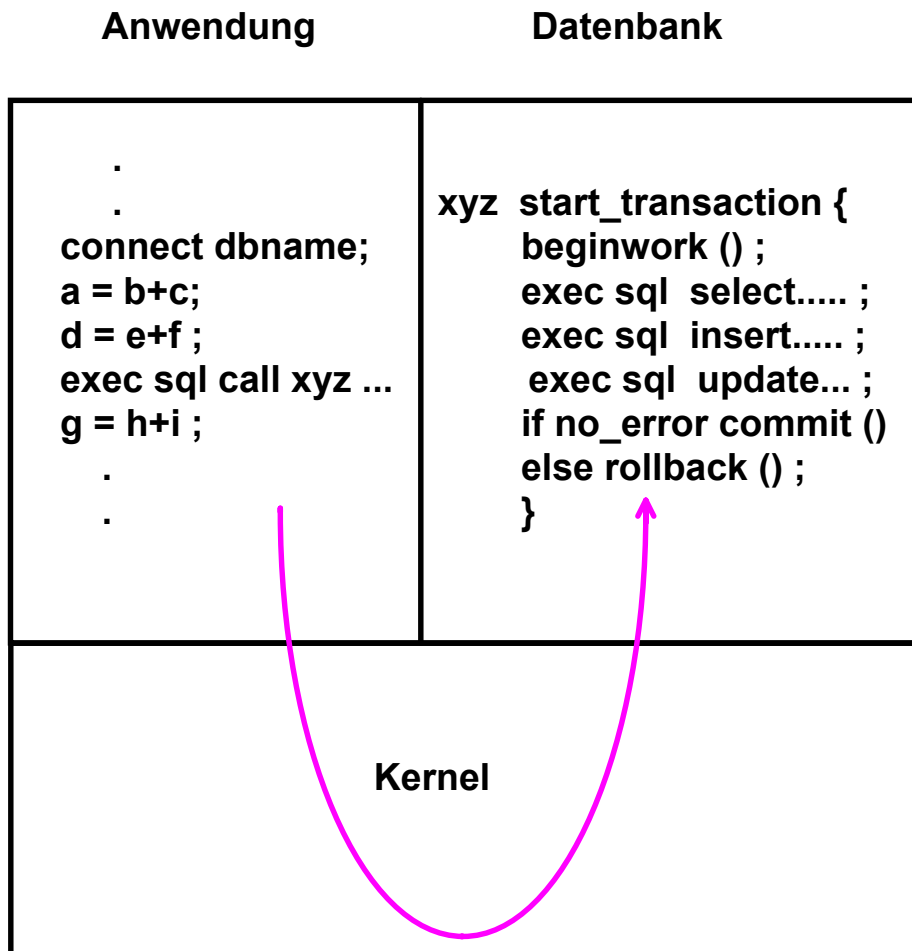
# **Multiple SQL Statements**

**This is a relatively simple model, which makes the application program easy to design and implement.**

**Because all application code resides at the client, a single application programmer can take responsibility for the entire application. However, there are some disadvantages to using this approach.**

**Because the application logic runs only on the client workstations, additional network input/output (I/O) operations are required for most SQL requests. These additional operations can result in poor performance. This approach also requires the client program to have detailed knowledge of the server's database design. Thus, every change in the database design at the server requires a corresponding change in all client programs accessing the database. Also, because the programs run at the client workstations, it is often complicated to manage and maintain the copies there.**

**Stored procedures enable you to encapsulate many of your application's SQL statements into a program that is stored at the DB2 server. The client can invoke the stored procedure by using only one SQL statement, thus reducing the network traffic to a single send and receive operation for a series of SQL statements. It is also easier to manage and maintain programs that run at the server than it is to manage and maintain many copies at the client machines.**



## Stored Procedure

Die Stored Procedure führt eine Gruppe von zusammenhängenden SQL Statements aus. Die Gruppe hat ACID Eigenschaften.

Stored Procedures werden manchmal als „TP light“ bezeichnet, im Gegensatz zu einem „TP heavy“ Transaktionsmonitor. Letzterer startet eigene Prozesse für mehrfache Aufrufe; innerhalb der Prozesse können nochmals Threads eingesetzt werden.

# Stored Procedures

**Stored Procedures bündeln SQL Statements bei Zugriffen auf Relationale Datenbanken. Sie ersetzen viele, vom Klienten an den Server übergebene, SQL Statements durch eine einzige Stored Procedure Nachricht**

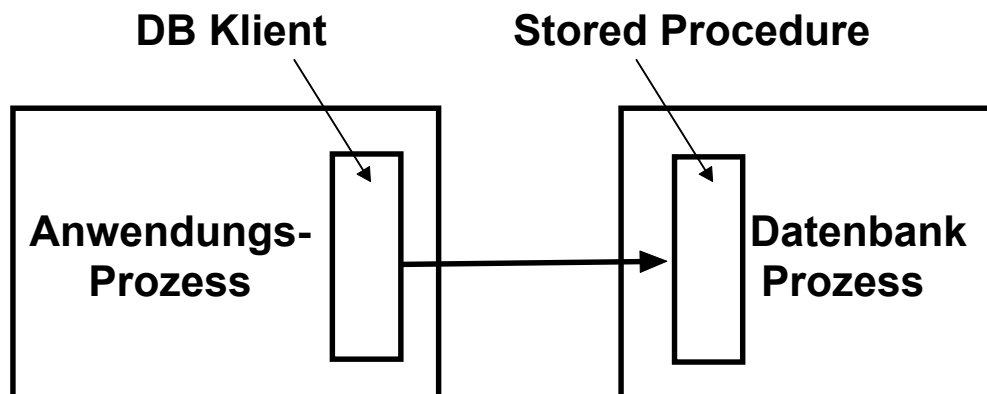
**Beispiel:**

**Bei einem Flugplatzreservierungssystem bewirkt eine Transaktion die Erstellung oder Abänderung mehrerer Datensätze:**

- **Passenger Name Record (neu)**
- **Flugzeugauslastung (ändern)**
- **Platzbelegung (ändern)**
- **Sonderbedingungen (z.B. vegetarische Verpflegung) (ändern)**

**Der Einsatz von Stored Procedures kann das Leistungsverhalten wesentlich verbessern**

# Stored Procedure Datenbank Klient



Bei einer Stored Procedure führt nicht der Anwendungsprozess, sondern der Datenbankprozess, die Gruppe von SQL Statements aus.

Datenbank Hersteller, z.B. IBM, Oracle, Sybase etc. stellen (proprietäre) Datenbank-Klienten zur Verfügung, die mit der Datenbank kommunizieren. Datenbank Klienten sind nicht kompatibel miteinander. Der Klient enthält einen Treiber, der ein „Format and Protocol“ (FAP) definiert.

FAP unterstützt mehrere Schicht 4 Stacks (TCP/IP, SNA, NetBios, ...)

# Transactional RPC

## Datenbank RPC

Stored Procedures werden bei manchen Herstellern in Proprietären Script Sprachen geschrieben, z.B. Sybase oder Oracle PL/SQL. Andere Hersteller, z.B. IBM verwenden reguläre Programmiersprachen wie Cobol, C++ oder Java.

Stored Procedures werden vom Datenbank Server in einer Library abgespeichert und mit einem Namen aufgerufen.

Das Klienten Anwendungsprogramm stellt Verbindung zur Datenbank her mit

**EXEC SQL CONNECT TO dbname**

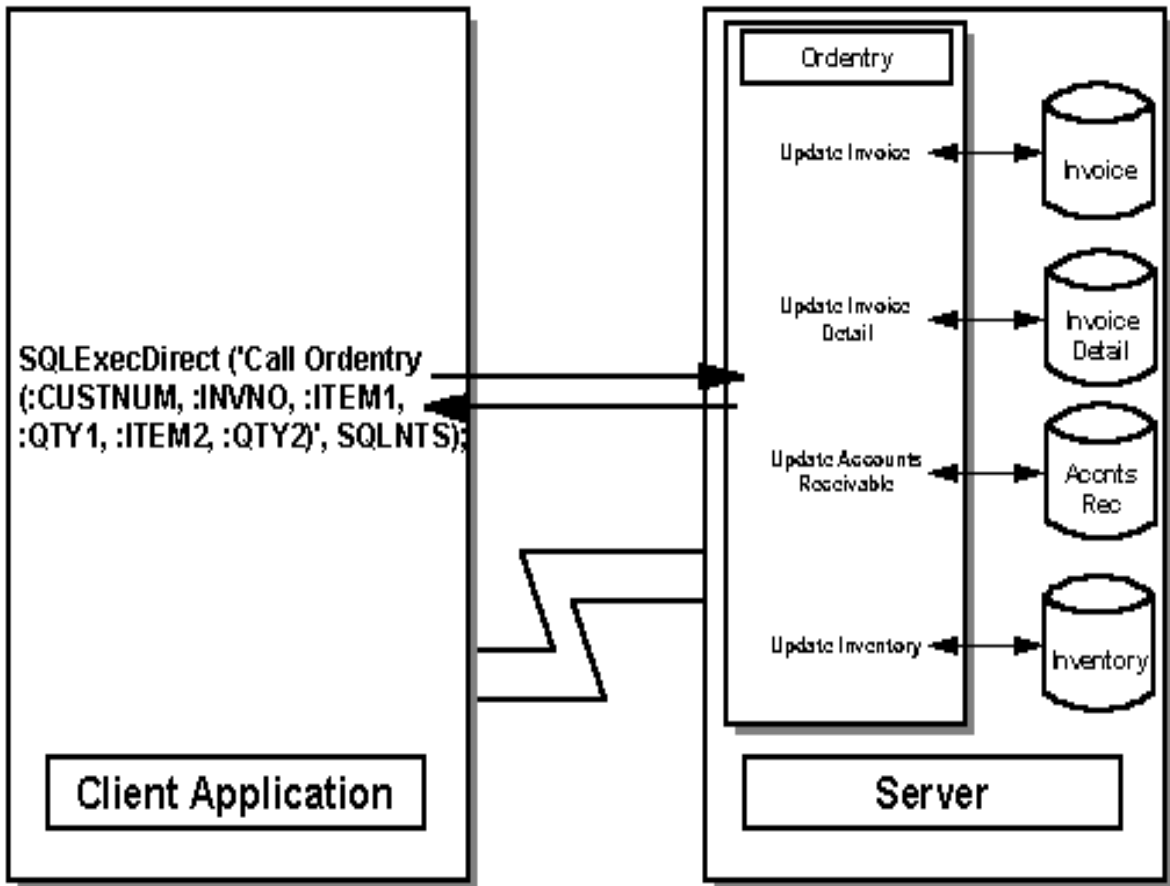
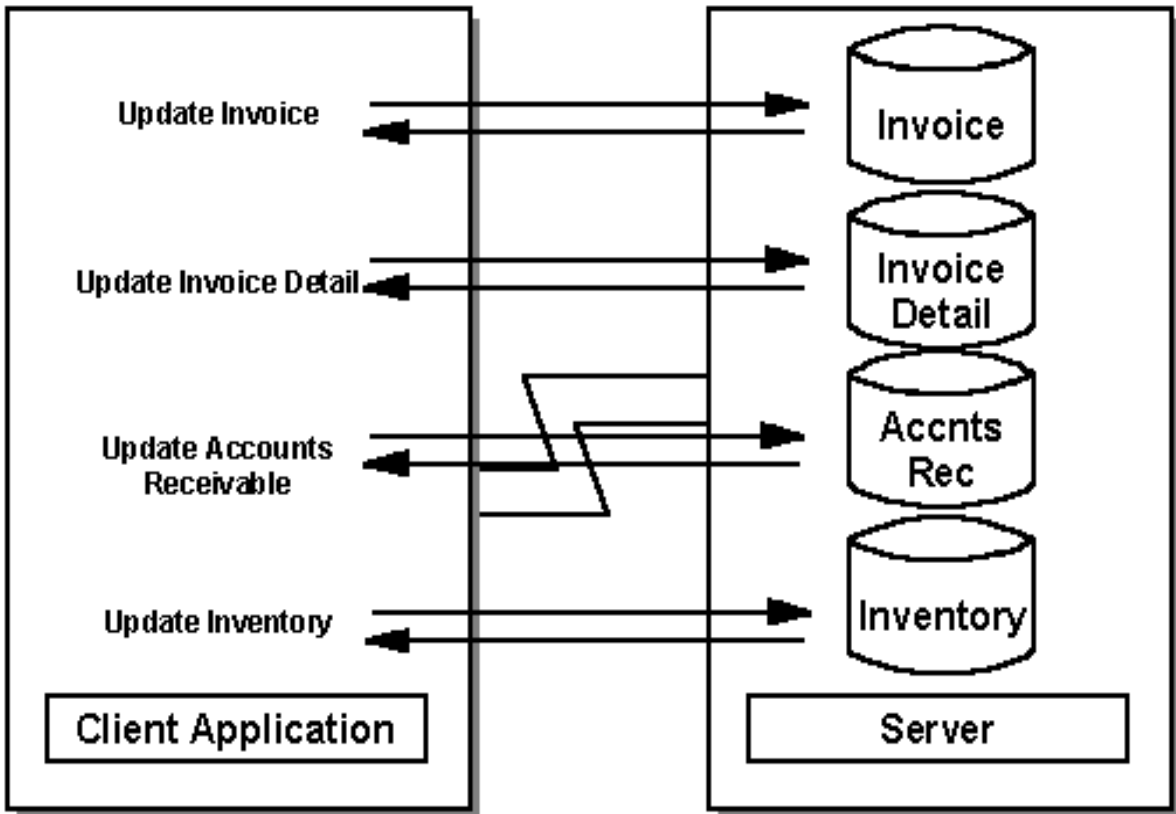
und ruft Stored Procedure auf mit

**EXEC SQL CALL ServProgName (parm1, parm2)**

Hierbei ist:

**parm1** Variablen Name (Puffer) der eine Struktur definiert (als SQLDA bei DB2 UDB bezeichnet), die zum Datenaustausch in beiden Richtungen benutzt wird.

**parm2** Variablen Name (Puffer) der eine Struktur definiert die für Return Codes und Nachrichten an den Klienten benutzt wird.



# Leistungsverhalten

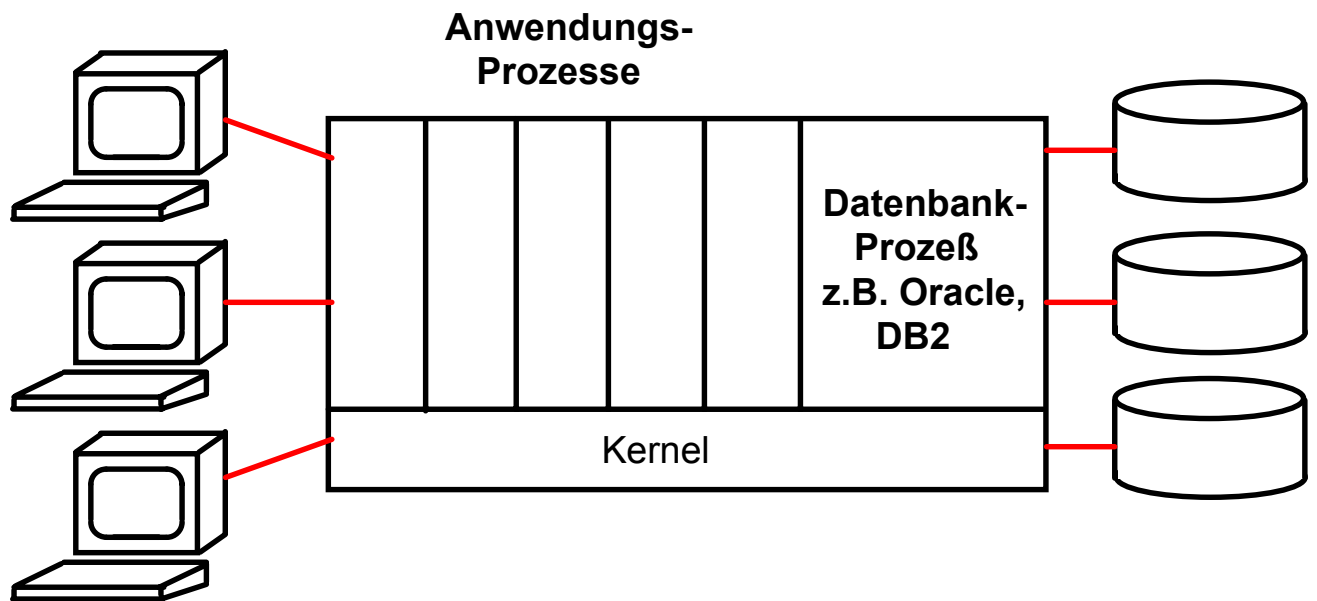
Meßergebnisse, einfache TP1 Transaktion, OS2, Intel 80486

**Dynamic SQL**                      **2,2 Transaktionen/s**

**Static SQL**                         **3,9 Transaktionen/s**

**Stored Procedure**                **10,9 Transaktionen/s**

**R. Orfali, D. Harkey: „Essential Client/Server Survival Guide“. John Wiley, 1994, S. 178**

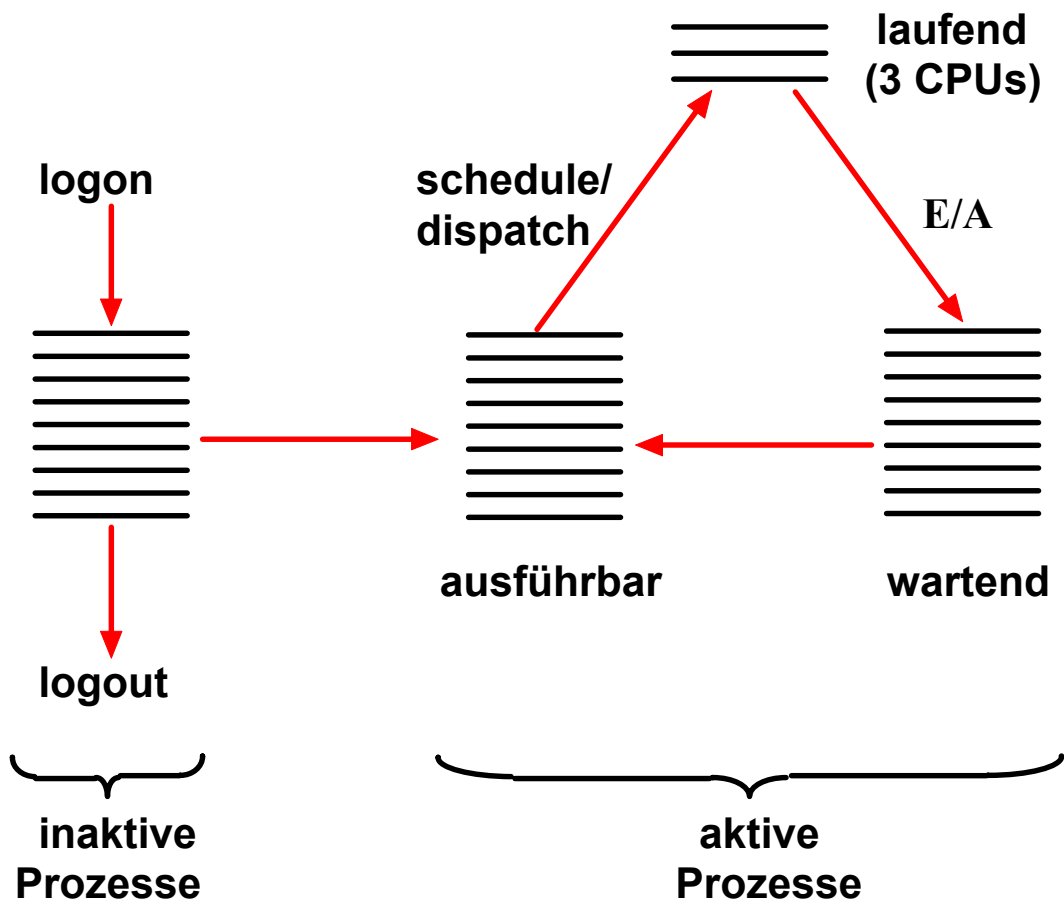


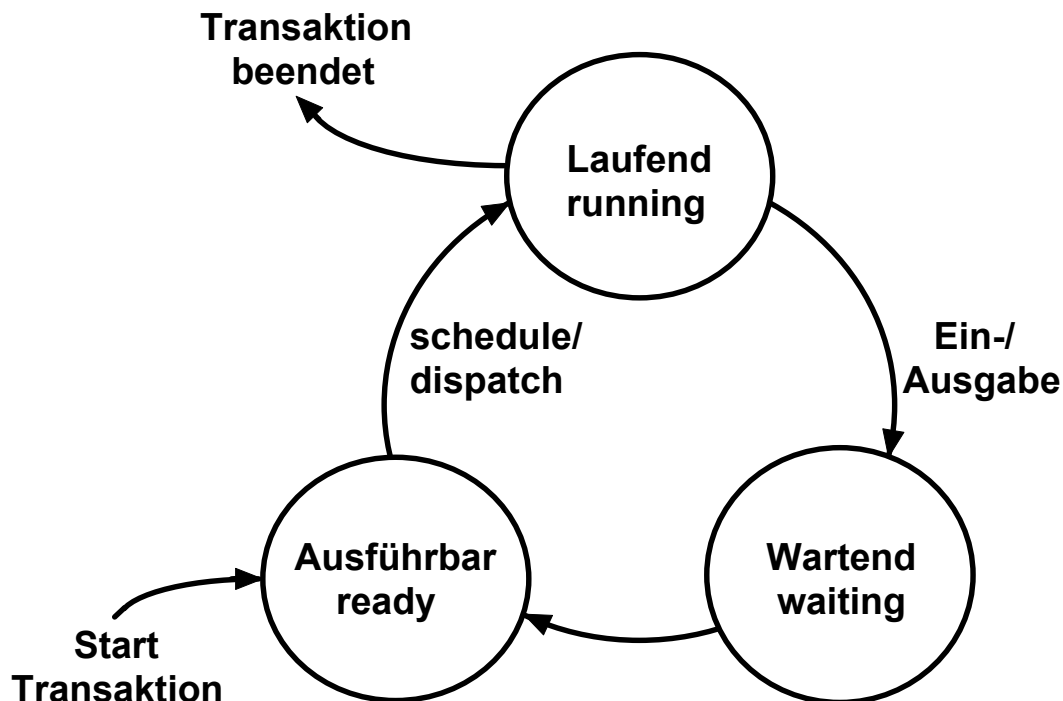
## Multiprogrammierte Verarbeitung von Transaktionen

In jedem Augenblick verarbeitet der Rechner gleichzeitig viele Transaktionen. Pro CPU ist nur eine Transaktion aktiv, die anderen sind ausführungsbereit oder warten auf den Abschluss einer Ein-/Ausgabeoperation. Hunderte oder Tausende paralleler Transaktionen sind denkbar.

Es muss verhindert werden, dass mehrere Transaktionen gleichzeitig auf die gleichen Daten zugreifen.

Leseberechtigungen und Schreibberechtigungen werden über Sperren (Locks) implementiert.





## Prozess Modell

Prozesse und Transaktionen verbringen den größten Teil der Verarbeitungsdauer mit dem Abschluss von E/A Operationen (Plattenspeicher).

Ohne Multiprogrammierung ist kein sinnvoller C/S Betrieb möglich.

2 Alternativen:

- Mehrfache Prozesse in eigenen virtuellen Adressenräumen
- Threads

# ACID Implementierung

## optimistischer Ansatz:

Daten mit Zeitstempel (oder Versionsnr.) versehen  
z.B. zusätzliches Feld in SQL Tabelle

Daten verarbeiten

if Zeitstempel unchanged then commit, else rollback

## pessimistischer Ansatz:

Daten mit Lock versehen

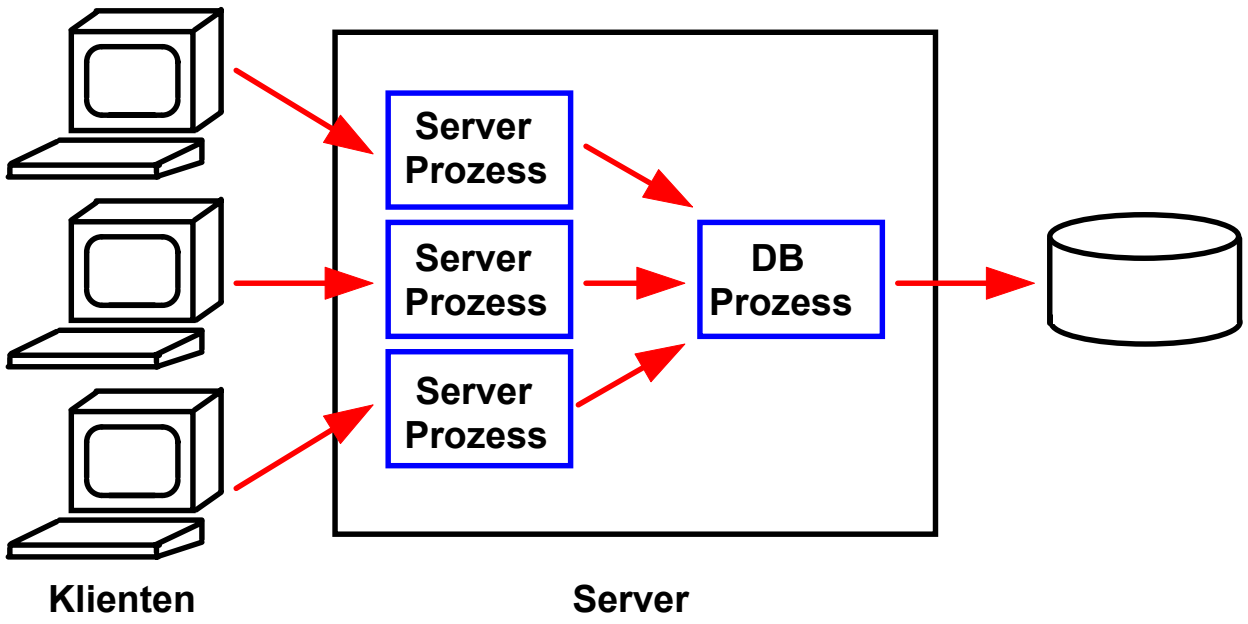
Daten verarbeiten

Ergebnis speichern

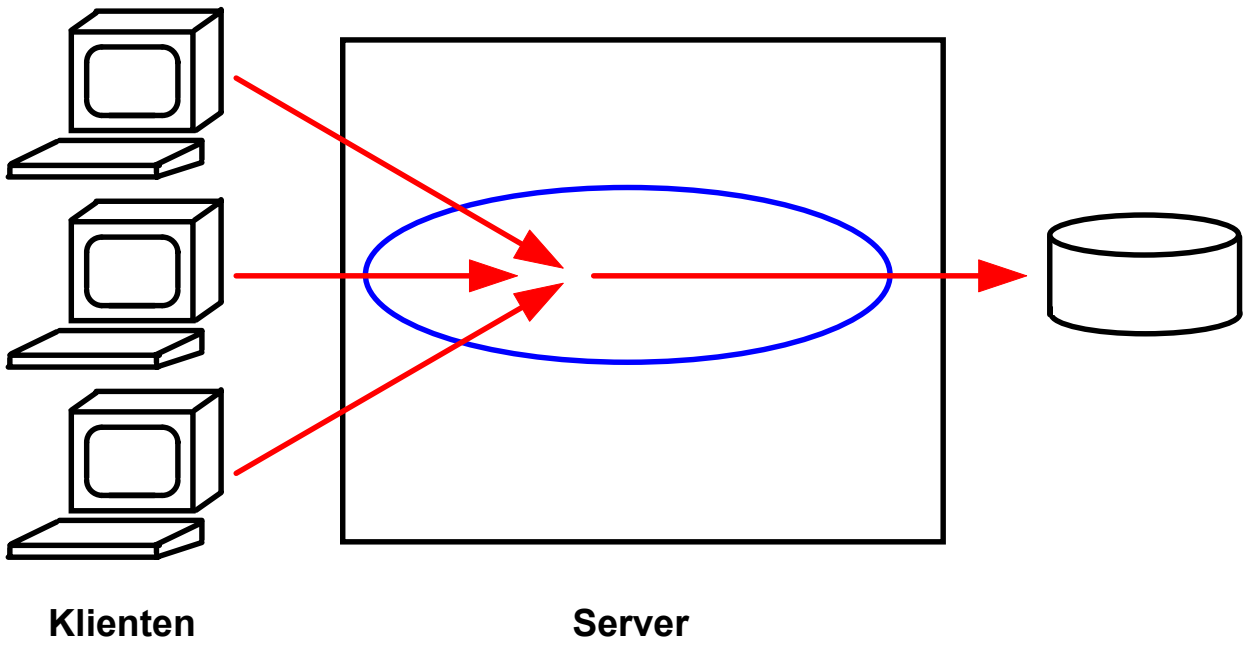
reset lock

**Der optimistische Ansatz geht von der Annahme aus, daß während der Verarbeitungszeit kein anderer Prozeß auf die gleichen Daten zugreift. Falls doch, dann rollback.**

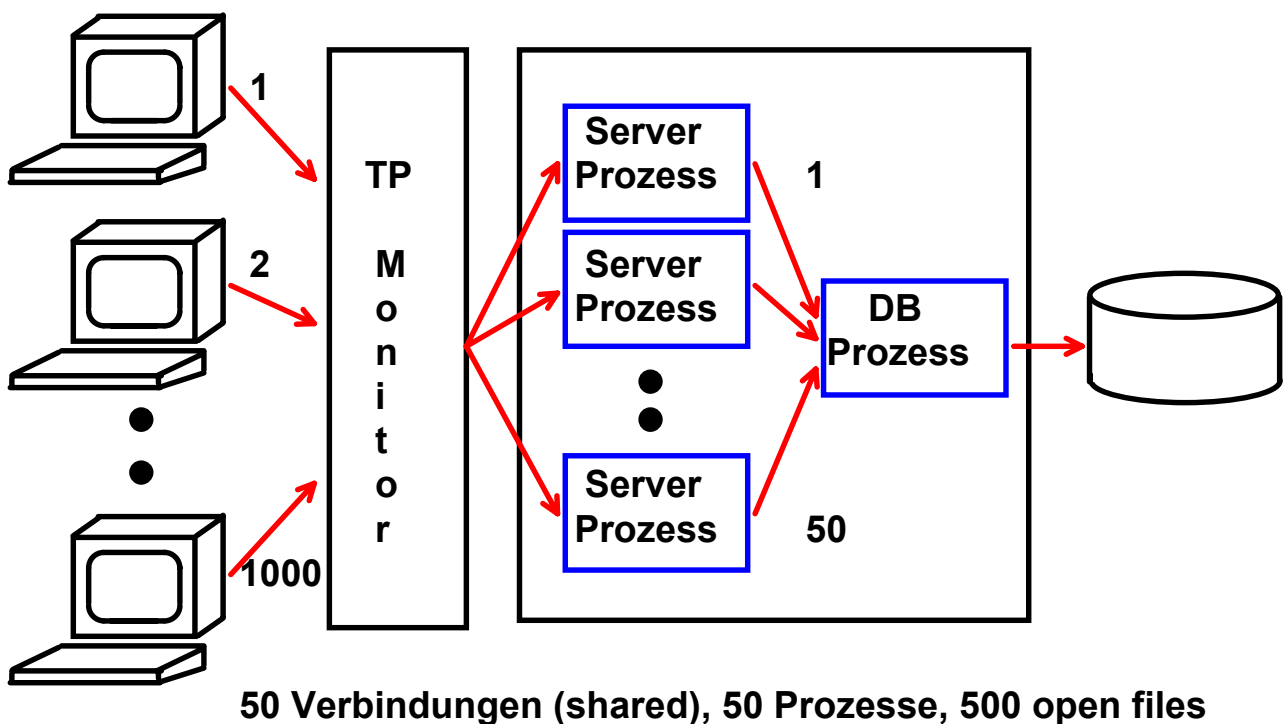
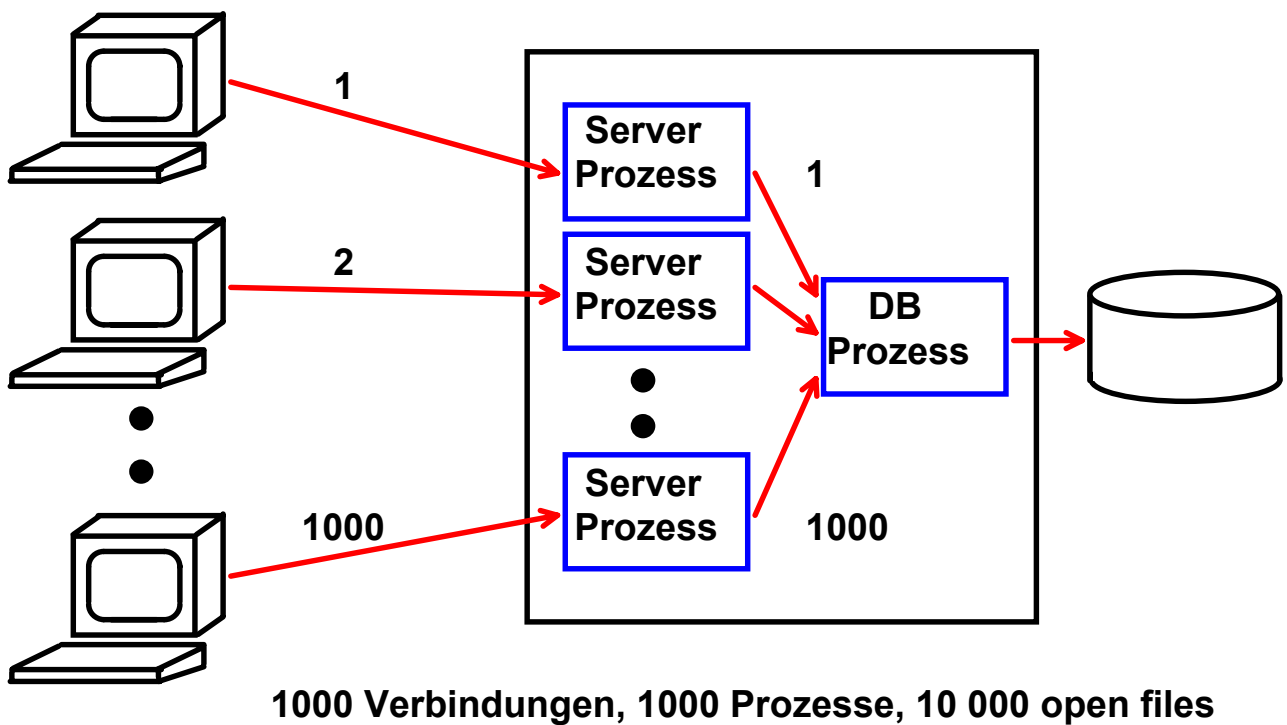
**Bei starker Belastung steigt die Anzahl der rollbacks exponentiell an. Deshalb hier Locks einsetzen und Prozesse auf explizite Datenfreigabe warten lassen.**



**1 Prozess pro Klient**  
 Beispiele: DB2, Informix, Oracle V6  
 Vorteil: robust



**Multithreaded Verarbeitung**  
 alle Server Arbeiten, einschl. DB, als ein einziger multithreaded  
 Prozess. Beispiele: Sybase, SQL Server  
 Vorteil: Leistungsverhalten



## Transaktionsverarbeitung mit und ohne TP Monitor

# **Client/Server-Systeme**

**Die Vorlesung findet wöchentlich statt, 2-stündig, jeweils Montag von 11:15 - 12:45. Es sind 7 Termine vorgesehen, erstmalig am Montag, den 8. Mai 2006. Weitere Termine am: 15.5, 22.5., 29.5., 12.6., 19.6. und 26.6.**

**Die Vorlesung ist als Wahlfach innerhalb der Technischen Informatik zugelassen, und kann mit 2 Stunden in den Prüfungsplan Technische Informatik aufgenommen werden.**

**Für das Wintersemester 2006/07 ist ein Praktikum Client/Server Systeme vorgesehen. Das Praktikum kann ebenfalls im Rahmen der Fachprüfung Technische Informatik in den Prüfungsplan mit 4 SWS aufgenommen werden.**

**Rückfragen bei Frau Reimold, Lehrstuhl Prof. Rosenstiel, oder:**

**spruth@informatik.uni-tuebingen.de  
spruth@informatik.uni-leipzig.de  
Tel.: 0172-8051-485**