Datenbanksysteme

Kap 4: Client-seitige DB-Programmierung

Client-Programmierung

Interaktive Eingabe von SQL

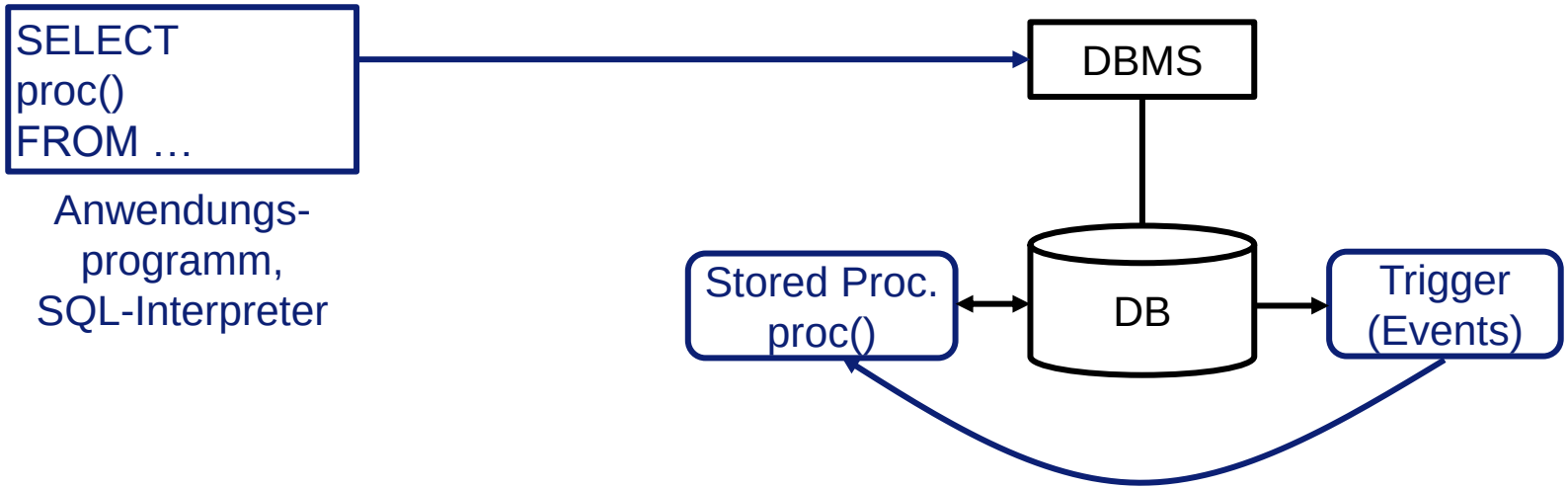
- DBS liefern SQL-Interpreter mit (Oracle: sqlplus, PostgreSQL: psql)
  - Geeignet für DB-Administration und Ad-hoc-Abfragen
  - Nicht praktikabel für Endanwender

Automatisierung der DB-Zugriffe

- Server-seitige Programmierung
  - Im DB-Server hinterlegt und von allen Anwendungen genutzt
- Client-seitige Programmierung
  - SQL-Kommandos werden aus Anwendungsprogramm aufgerufen

Server-seitige Programmierung

- Ablauflogik in Form von Stored Procedures auf dem DB-Server



- Hauptanwendungsgebiet:
  - Event-gesteuerte Programmierung mit Triggern
- Vorteil:
  - Unabhängig davon, wie auf die Daten zugegriffen wird (z.B. auch SQL-Interpreter)

Client-seitige Programmierung

- Vorherrschende Form der DB-Programmierung
  - Anwendungsprogramm stößt nur elementare SQL-Kommandos an
  - Ablauflogik wird in anderer Sprache (Host Language, z.B. C++) programmiert

- Frage
  - Wie können aus der Host Language heraus SQL-Kommandos ausgeführt werden?

SQL und Host Language

Methode	Beschreibung
SQL-Script	• Batch-Aufruf SQL-Interpreter. • Keine Einbindung in Host-Language.
Embedded SQL (ESQL)	• Mischen von SQL und Host Language. • Präprozessor übersetzt *exec sql* Statements
Call Level Interface (CLI)	• Routinen in Host Language, SQL-Kommandos ggf. als Parameter. Natives CLI: • auf konkretes DBMS zugeschnittene Bibliothek, z.B. OCI (Oracle), libpq (PostgreSQL) Abstraktes CLI: • DBS-unabhängige abstrakte Bibliothek. • Für konkretes DBS "Treiber" nötig. • Beipiele: odbc, jdbc, bde, perl-dbi

Typische Einsatzgebiete

Schnittstelle	Einsatzgebiet
SQL-Script	Einfache administrative Aufgaben, z.B. • User anlegen • DB-Schema einspielen • …
ODBC	DB-unabhängige Massensoftware, z.B. Office-Pakete
PERL-DBI	• cron-gesteuerte Serverprozesse, • als CGI-Script in Web-Programmierung
ESQL native CLIs	• Individualsoftware, • DBS-spezifische Tools, • Implementierung eigener abstrakter Interfaces, • Programmierung Treiber für abstrakte CLIs

SQL-Scripte

SQL-Interpreter wie `psql` können nicht nur interaktiv verwendet werden:

- Ausführen einer externen Datei (z.B. *script.sql*) mit SQL-Kommandos
  - innerhalb psql-Session mit Metakommando: *\i script.sql*
  - mit entsprechender Aufrufoption: *psql -f script.sql*
  - *script.sql* kann auch Metakommandos enthalten
- Übergabe eines SQL-Kommandos als Kommandozeilenparameter
  - Beispiel: *psql -c "truncate table produkt;"*

SQL-Scripte

Alternative: "Fernsteuerung" des SQL-Interpreters durch Umlenkung von *stdin*

- In der Shell:

```
#!/bin/sh
psql <<EOF
/* SQL-Kommandos */
EOF
```

- Im C-Programm:
  - unter Unix mit `popen()` (Einweg-Pipe) oder `pipe()` + `fork()` + `dup2()` + `exec()` (Zweiwege-Pipe)
  - unter Windows mit `CreatePipe()` + `DuplicateHandle()` + `CreateProcess()` + `CreateThread()`

SQL-Scripte

Nachteile

- keine Kontrollflusssteuerung (nur SQL)
- Fehlerbehandlung schwierig

Kann umgangen werden bei *psql -c*

- Kontrollfluss durch Shell-Befehle
- Abfragen des Exit-Codes möglich
- Aber: großer Overhead, da pro SQL-Befehl ein Aufruf von psql

→Einsatz begrenzt auf einfache Aufgaben

SQL-Scripte

Beispiel: *dropuser* von PostgreSQL

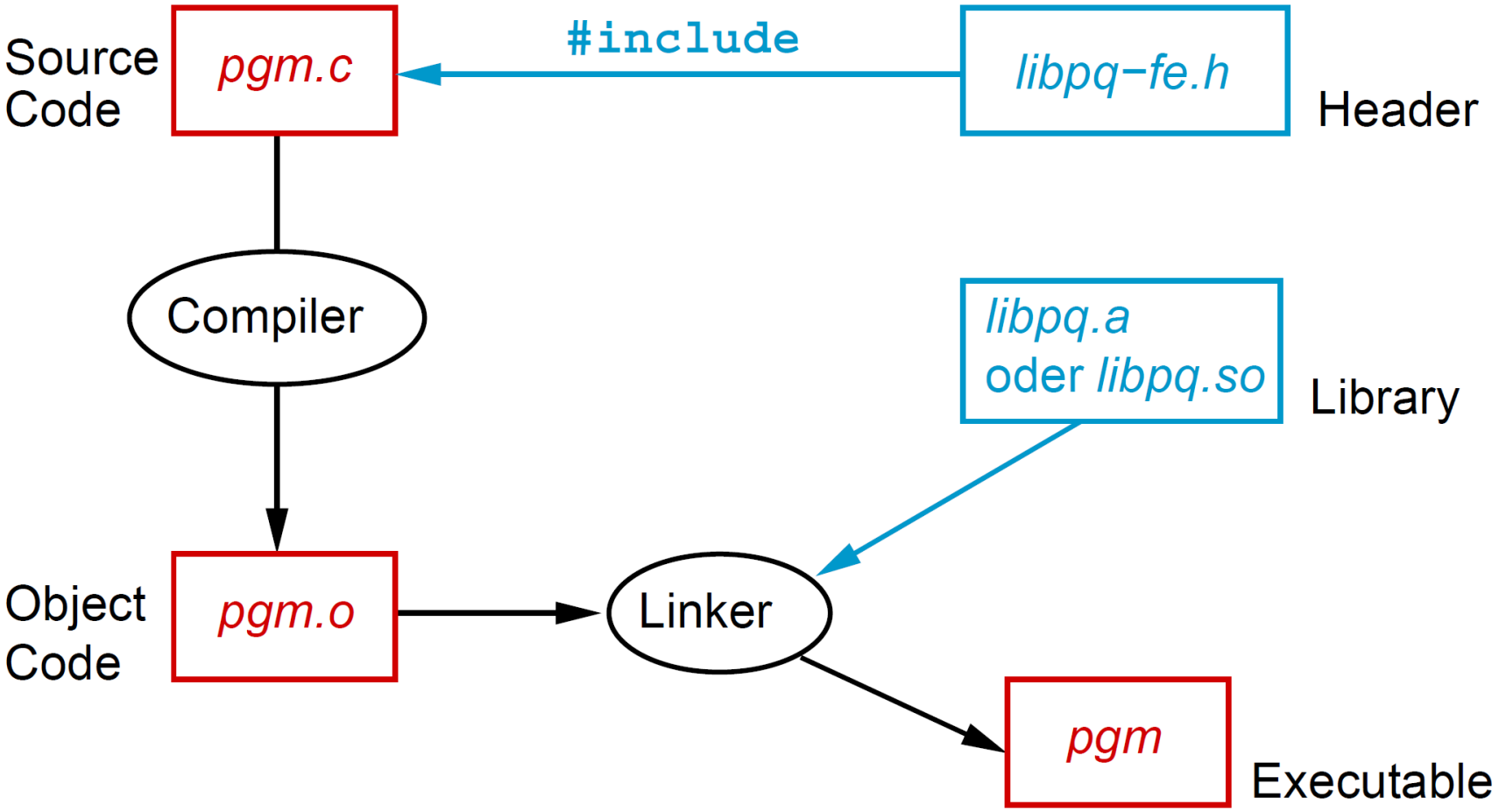
```
#!/bin/sh
# (...)
# Commandline Parsing schaufelt
# zu löschenden User in Variable $DelUser

psql $PSQLOPT -d template1 -c "DROP USER $DelUser"

# Abfragen Exitcode
if [ "$?" -ne 0 ]; then
   echo "deletion of user \\"$DelUser\\" failed" 1>&2
   exit 1
fi
exit 0
```

Natives CLI (Call Level Interface)

- Zugriff über vom DBS-Hersteller bereitgestellte Bibliotheksroutinen (Oracle: oci, Postgres: libpq)

Natives CLI

- Erzeugung Client-Programm:



Natives CLI

Konkrete Schritte der Programmierung:

- Source Code in Editor erstellen
  - Funktionsprototypen mit *#include <libpq-fe.h>* einbinden
- Zu Object Code compilieren
  - *gcc -c -I/usr/include/pgsql pgm.c*
  - */usr/inlude/pgsql* ist Verzeichnis mit Postgres-Headern
- Mit *libpq* linken
  - *gcc -o pgm pgm.o -L/usr/lib/pgsql -lpq*
  - */usr/lib/pgsql* ist Verzeichnis mit Postgres-Libraries
- Mit dem Tool `pg_config` lassen sich die Include- und Library-Pfade ermitteln
  - Verwendung: siehe Makefile bei Praktikum 2

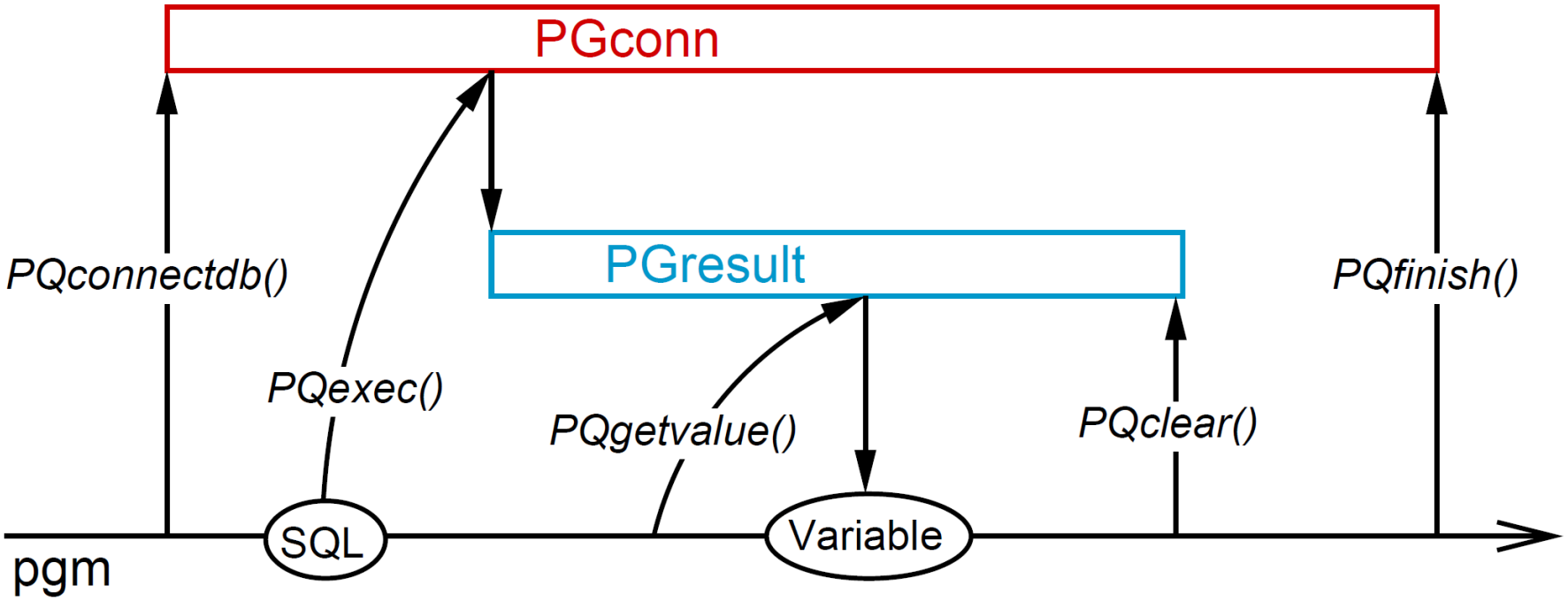
Natives CLI: Klassifikation der libpq-Routinen

- Verbindungsaufbau, -abbau:
  - *PQconnectdb(), PQfinish(), PQstatus()*
- Ausführen von SQL-Statements:
  - *PQexec(), PQresultStatus(), PQcmdTuples(), PQclear()*
- Verarbeiten von Abfrageergebnissen:
  - *PQntuples(), PQgetvalue(), PQgetlength()*

- Wie die ANSI C *stdio*-Bibliothek ist *libpq* eine mit C-Mitteln realisierte objektorientierte Bibliothek
  - Zuvor konstruierte Strukturen werden als Funktionsparameter übergeben
  - Destruktoren müssen selbst aufgerufen werden

Natives CLI – Hauptobjekte in libpq

Datentyp	Konstruktor	Destruktor
`PGconn`	`PQconnectdb()`	`PQfinish()`
`PGresult`	`PQexec()`	`PQclear()`



Natives CLI – Verbindungsauf- und abbau

```
PGconn* conn;

/* Login */
conn = PQconnectdb("dbname=db user=usr ...");

/* Fehlerprüfung */
if (PQstatus(conn) == CONNECTION_BAD) /* ... */

/* Das so konstruierte PGconn-Objekt conn wird beim
   Absetzen von SQL-Kommandos verwendet (s. nächste Folien)
   (...)
*/

/* Logout */
PQfinish(conn);
```

Natives CLI – Beispiel Non-SELECT

```
PGconn *conn; /* Konstruktion auf vorheriger Folien */
PGresult *res;

/* Absetzen SQL-Statement */
res = PQexec(conn, "DELETE FROM produkt WHERE preis>'3.0'");

if (PQresultStatus(res) == PGRES_COMMAND_OK) {
   /* Rückmeldung Auswirkungen */
   printf("%s Sätze gelöscht\\n", PQcmdTuples(res));
}
else {
   /* Fehlerbehandlung */
}

/* Speicher freigeben nicht vergessen! */
PQclear(res);
```

Natives CLI – Beispiel SELECT

```
PGconn *conn;
PGresult *res;

/* Absetzen SQL-Statement */
res = PQexec(conn, "SELECT usename FROM pg_user");

if (PQresultStatus(res) == PGRES_TUPLES_OK) {
   /* Ausgabe Ergebnisse */
   for (i = 0; i < PQntuples(res); i++)
      printf("%2d. %s\\n", i+1, PQgetvalue(res,i,0));}
else {
   /* Fehlerbehandlung */
}

/* Speicher freigeben nicht vergessen! */
PQclear(res);
```

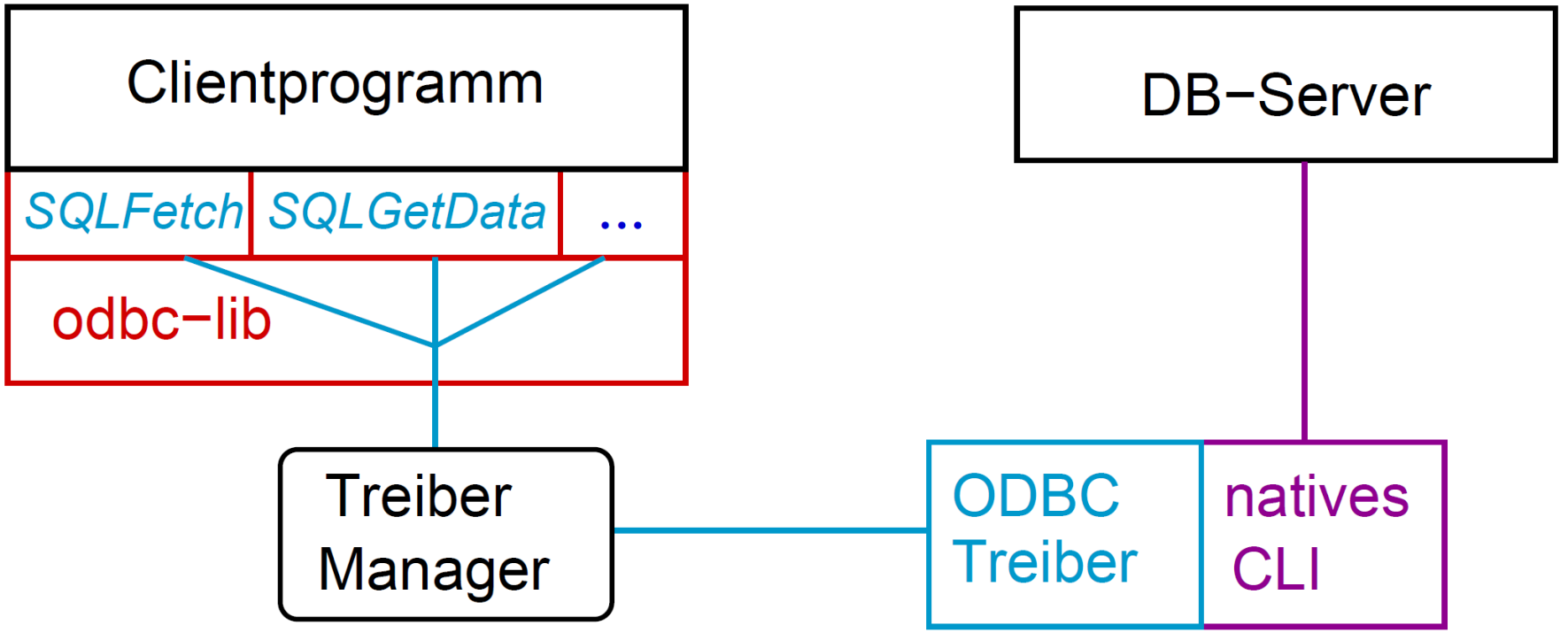
Natives CLI

Weiterführende Literatur

- Hartwig: PostgreSQL - Professionell und praxisnah. Kapitel 9.1 (Semesterapparat TWY Hart)
  - Hartwig beschreibt auch die C++ Bibliothek libpq++.
  - Achtung: diese Schnittstelle ist veraltet!
- PostgreSQL Programmer's Guide: Client Interfaces - libpq.
  - Im PG-Paket enthalten. Online verfügbar unter https://www.postgresql.org/docs/15/libpq.html

Abstraktes Call Level Interface

- Zugriff über DBS-unabhängige Bibliotheksroutinen
- Passender "Treiber" wird zur Laufzeit vom "Treiber-Manager" geladen



Abstraktes CLI

- Vorteile
  - Programm läuft (im Prinzip) mit beliebigem DBS
    - Aber: ggf. abhängig vom SQL-Dialekt
  - Keine Bindung an konkretes DBS zur Compilezeit
    - Geeignet für Massensoftware (z.B. Office-Pakete)
  - Kann auch *ohne* DBS verwendet werden:
    - Z.B. gibt es Perl-DBI Treiber für Text Files
- Nachteile
  - Kleinster gemeinsamer Nenner
    - Fortgeschrittene DBS-Features nicht nutzbar
  - Im Einzelfall doch Fallunterscheidung DBS nötig
    - Beispiel: implizite Transaktionen in Oracle
  - Langsamer als direkt natives CLI
  - Erfordert Infrastruktur und Konfiguration

Abstraktes CLI

- Überblick

Abstraktion	Host Language	Hersteller
Open Database Connectivity (ODBC)	C Visual Basic	Microsoft Offener Standard
Java Database Connectivity (JDBC)	Java	Sun Offener Standard
Borland Database Engine (BDE)	Object Pascal C++	Borland
Perl Database Interface (DBI)	Perl	Tim Bunce Offener Standard

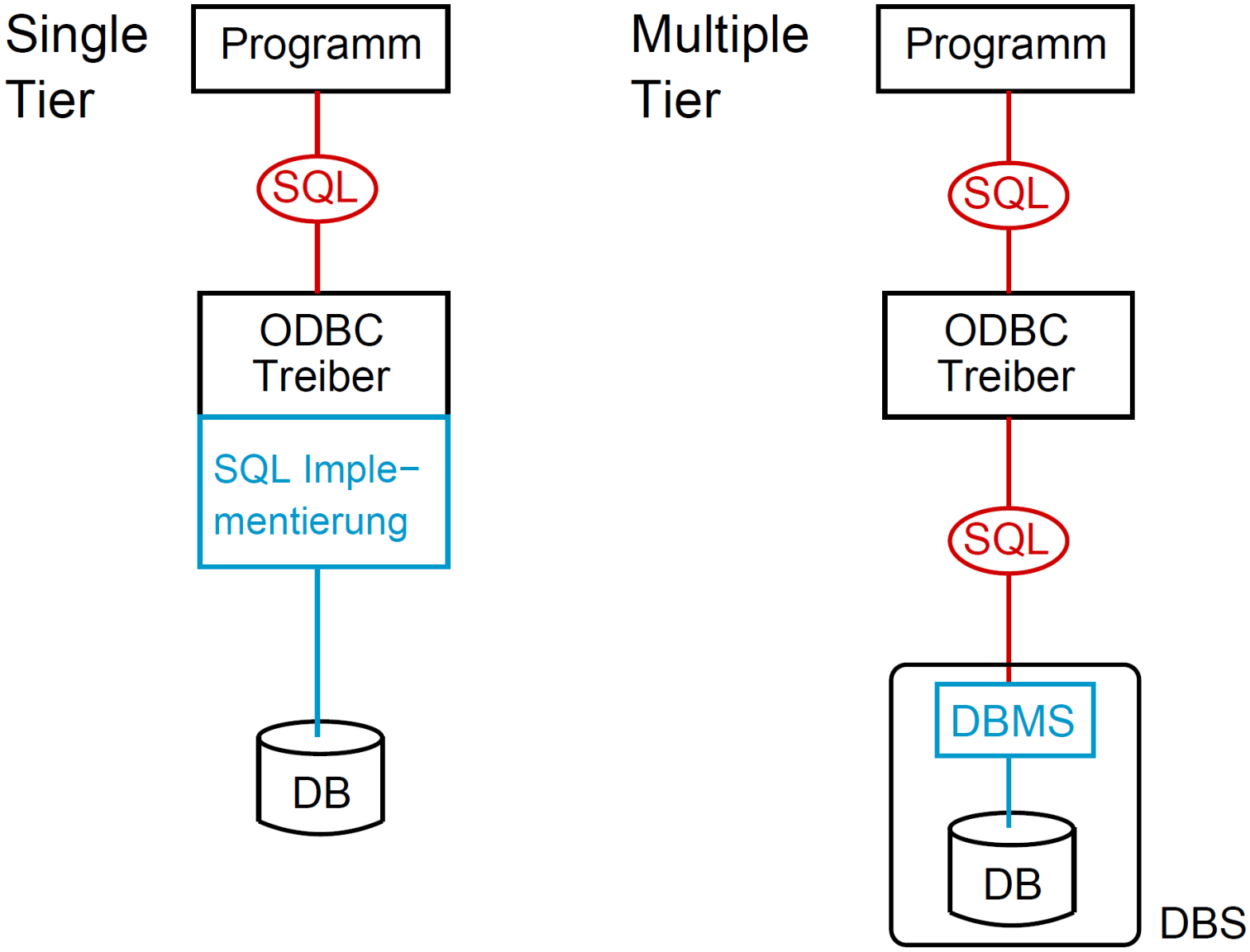
- Wir schauen uns konkret an:
  - ODBC (prinzipieller Aufbau)

Verbreitete Irrtümer über ODBC

- ODBC ist nur für Windows-Programme
  - Infrastruktur gibt es für Windows, Unix, MacOS, OS/2
  - Aber oft vom DBS-Hersteller nur für Windows mitgeliefert →Treiber von Drittanbieter beziehen
- ODBC ist langsam
  - Unzulässige Verallgemeinerung der Erfahrungen mit Access + VisualBasic (historisch erste ODBC-Umgebung)
  - Nicht gültig für "Multiple-Tier"-Treiber, da dabei die ODBC-Abstraktionsschicht nur geringer Overhead ist

- Single Tier: Treiber implementiert SQL-Abfragen
- Multiple Tier: Treiber reicht SQL an DBS weiter

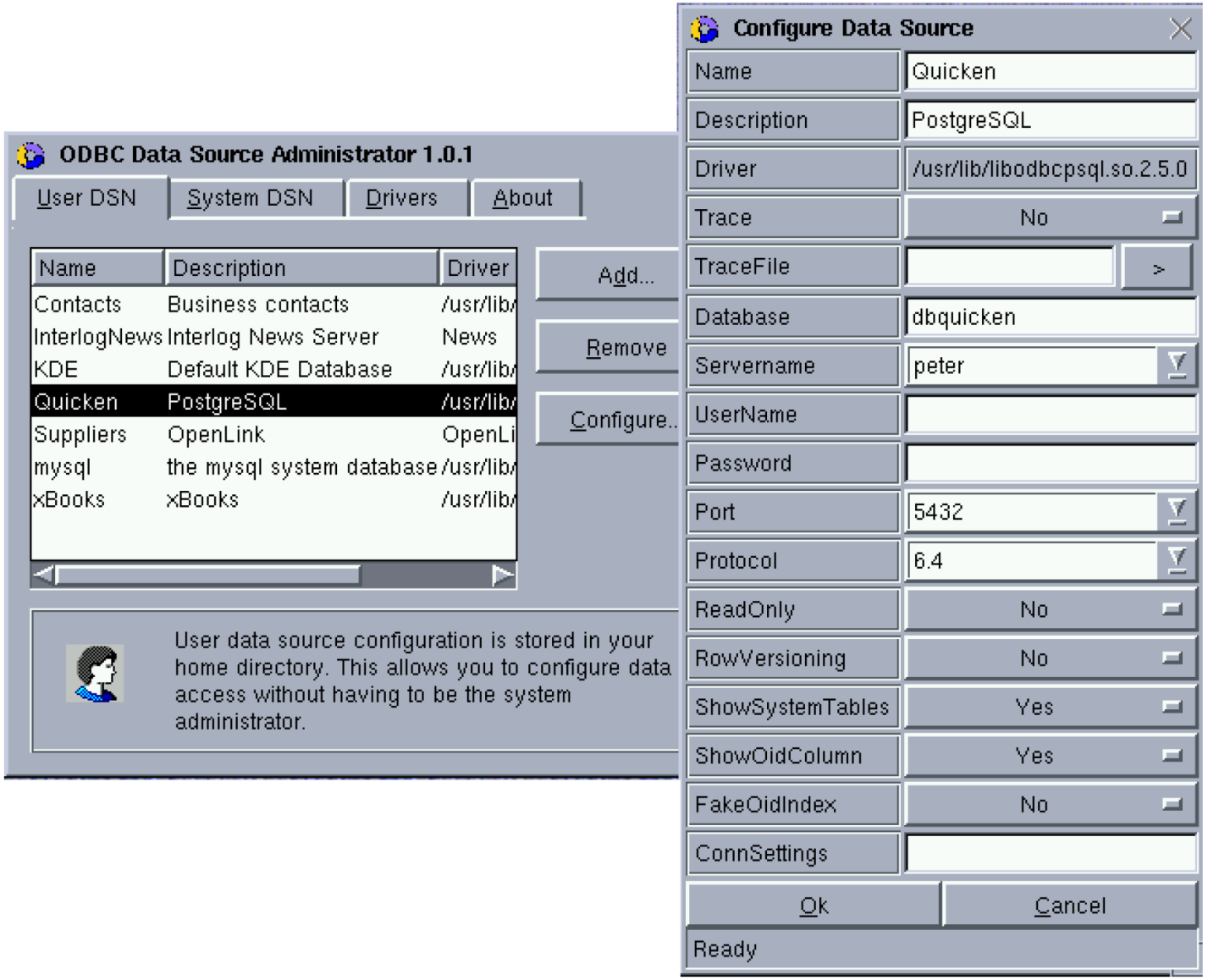
Single Tier vs. Multiple Tier



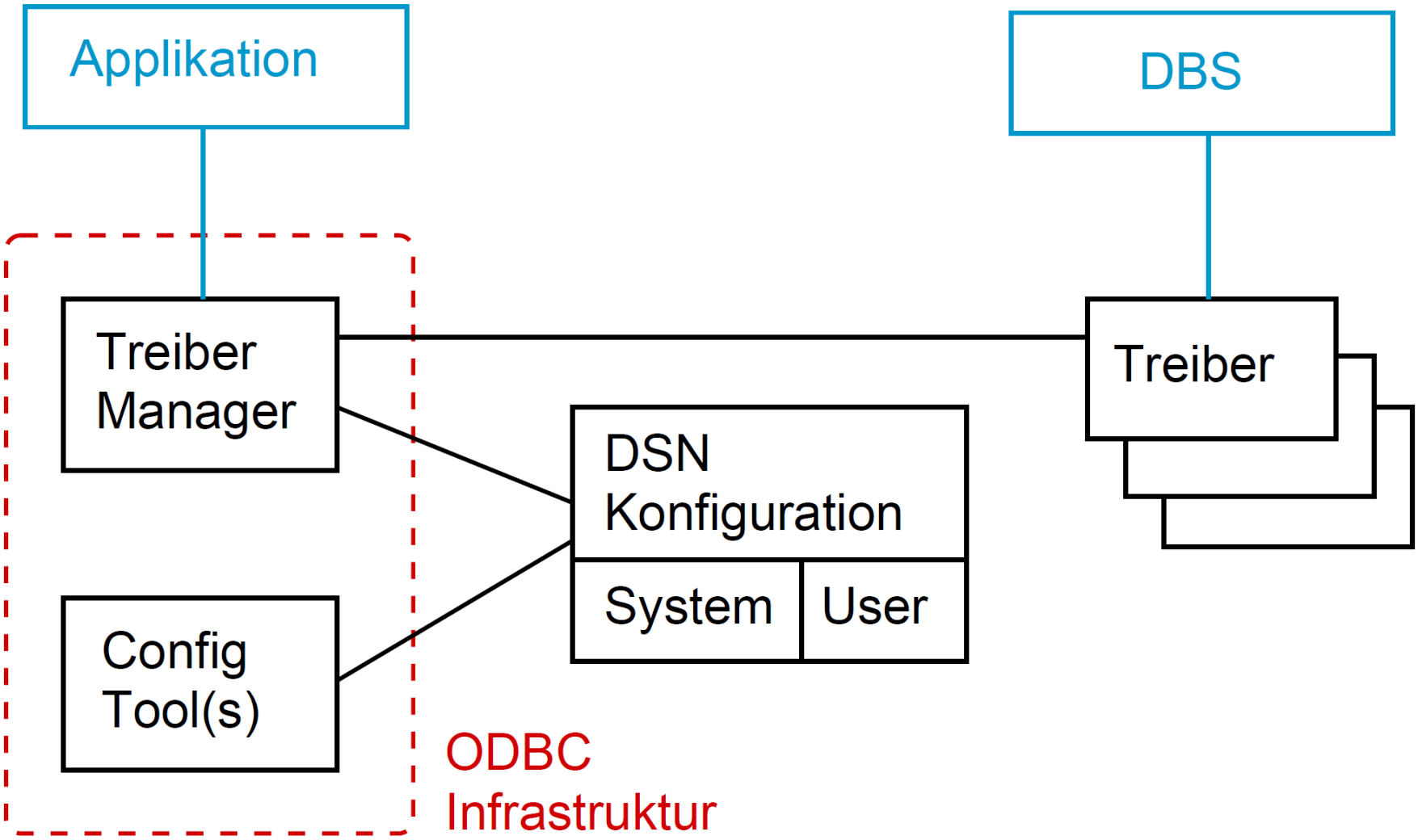
ODBC Datasource

- Abstraktion Verbindungsparameter
  - Zu verwendender ODBC-Treiber
  - Treiber-spezifische Parameter (z.B. pghost, pgdatabase, ...)
- Einem Satz Verbindungsparameter wird ein Data Source Name (DSN) zugewiesen
- Statt Parameter beim Login zu übergeben, gibt das Anwendungsprogramm den DSN an
- Zuordnung DSN zu Parametern:
  - Hinterlegt in Datei (Unix) oder Registry (Windows)
  - ODBC-Infrastruktur stellt Config-Tool(s) bereit

DNS-Konfiguration



ODBC Architektur



Vereinfachtes Beispiel ODBC Connection

```
SQLHENV    sqlenv;/* Handle ODBC environment */
long       rc;/* result of functions */
SQLHDBC    sqlconn;/* Handle connection */

/* 1. Allocate Environment Handle and register Version */
rc = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&sqlenv);
rc = SQLSetEnvAttr(sqlenv, SQL_ATTR_ODBC_VERSION, (void*)SQL_OV_ODBC3, 0);

/* 2. Allocate Connection Handle, set Timeout */
rc = SQLAllocHandle(SQL_HANDLE_DBC, sqlenv, &sqlconn);
SQLSetConnectAttr(sqlconn, SQL_LOGIN_TIMEOUT, (SQLPOINTER*)5, 0);

/* 3. Connect to the Datasource "web" */
rc = SQLConnect(sqlconn, (SQLCHAR*) "web", SQL_NTS,
           (SQLCHAR*) "christa", SQL_NTS, (SQLCHAR*) "", SQL_NTS);

/* Typical Errorcheck */
if ((rc != SQL_SUCCESS) && (rc != SQL_SUCCESS_WITH_INFO)) {
            /* Error Handling */ }

/* Free Resources */
SQLFreeHandle(SQL_HANDLE_DBC, sqlconn);
SQLFreeHandle(SQL_HANDLE_ENV, sqlenv);
```

Weitergehende ODBC-Features

- Anfragen an die Fähigkeiten des Treibers
- Anfragen an den System Catalog
- Anfragen über verfügbare Datenquellen und gesetzte Optionen
- ODBC verwendet eigenen SQL-Dialekt, der in SQL-Dialekt des DBS übersetzt wird
  - kompliziertere SQL-Statements können an DBS "durchgereicht" werden → Abhängigkeit von SQL-Dialekt)

ODBC Referenzen:

- Kyle Geiger: Inside ODBC. Microsoft Press 1995
- ODBC Infrastruktur für Unix: http://www.unixodbc.org/
- Microsoft ODBC Seite: http://www.microsoft.com/data/odbc/

Weitere Ansätze

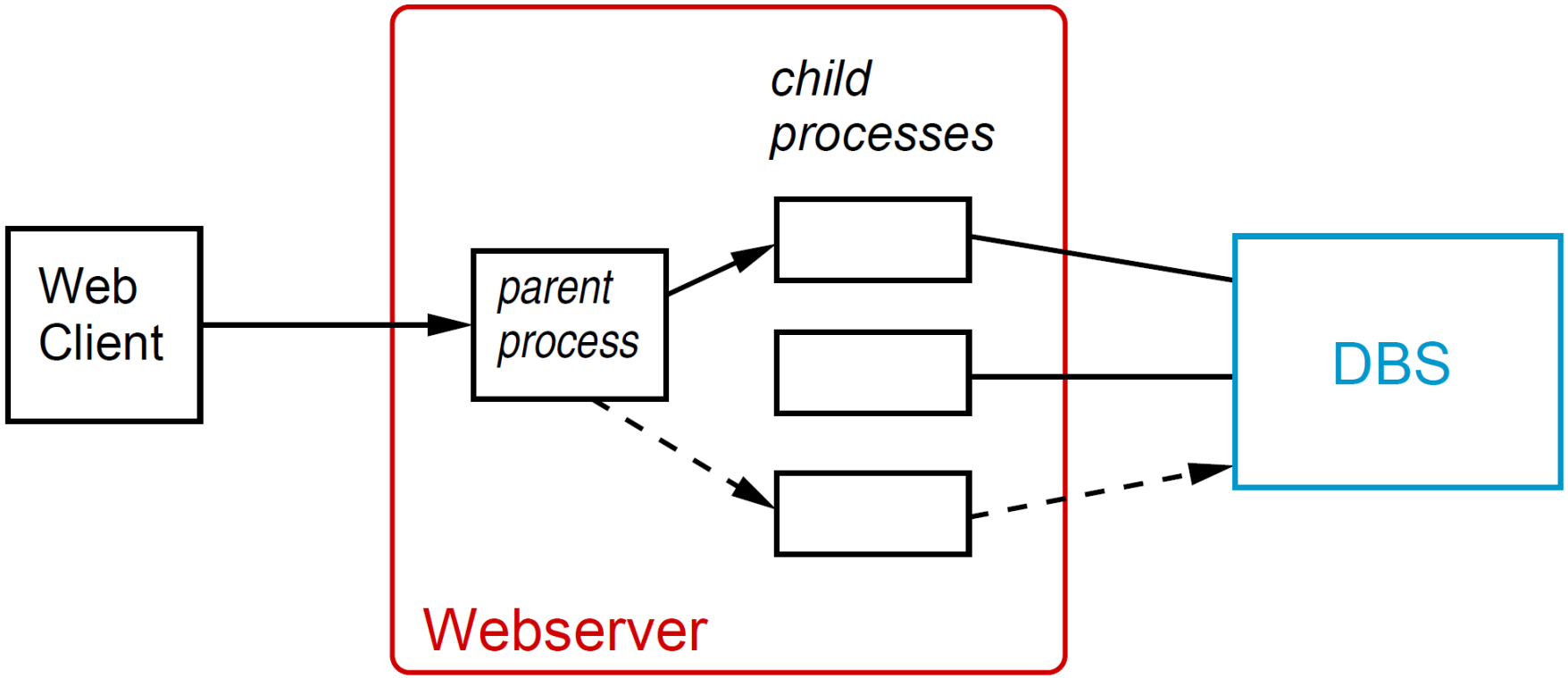
Webanwendungen

- Normale Client-Server-Anwendungen sind Session-orientiert:
  - Login → umfangreiche Verarbeitung → Logout
- Webanwendungen sind sessionlos:
  - Einzelne Seiten werden ohne weitere Verpflichtung angefordert

  →Login für jeden Seitenabruf erzeugt großen Overhead

- Lösung: *Persistent Database Connections*

Persistent Database Connections



- Voraussetzung: Verarbeitung nicht über CGI, sondern Script-Modul
- Allgemeine Lösung (unter GNU GPL): http://sqlrelay.sf.net/

Weitere Ansätze – Application Server

- Application Server
  - Anwendungen wollen eigentlich gar keinen Datenbank-Zugriff, sondern spezielle Funktionen
- Idee:
  - Entwerfe Protokoll für diese Funktionen
  - Implementiere dieses Protokoll in eigenem Client-Server Modell
  - Clients greifen über dieses Protokoll auf Application Server zu
  - Application Server implementiert DB-Zugriffe
- Vorteile:
  - Datenbank-Logik zentral in Application Server
  - Einfachere Client-Programmierung

Weitere Ansätze – OR-Mapper

- Objekt-Relationale Mapper
  - Stellen Daten dem Programmierer als Objekte zur Verfügung
  - Datenbankstruktur als "shadow information" verborgen
  - Umwandlung in relationales Modell erfolgt im Hintergrund
- Vorteile:
  - Keine Datenbankkenntnisse (auch kein SQL) erforderlich (solange alles glatt läuft)
  - Objektorientierter Zugriff entspricht Programmiersprache
- Nachteile:
  - Geringe Kontrolle über Qualität des Datenmodells und Transaktionen
  - Reduzierte Performance und Flexibilität