Working with perl modules

Lesson 1

Ties

Databases and DBI

Ties

- The `tie` function associates a variable with an underlying data structure and a set of subroutines
- Manipulation of the variables perform the appropriate actions on the data structure
  - implemented by a well defined set of subroutines that perform basic actions (fetch, store, delete,…)
  - standard interface of the variables is changed
  - ties bind methods from packages to variables

Ties and Objects

- Tied variables act like objects
  - interface is defined
  - methods from a specific package get called
  - difference: predefined set of methods
- Ties have the same procedures for finding the proper method (polymorphism) as objects
  - ties tend therefore to be slow (as the OO methods)

Tie Implementation

- Select a package (class) for implementing a tie
- Code methods with predefined names
  - for a scalar: **`TIESCALAR, FETCH, STORE, DESTROY`**
  - for a hash: 9 methods, e.g. **`FIRSTKEY, NEXTKEY`**
- Tie the methods to the variable:

  `tie $var, $class, @args;` this gets translated to

  `$class->TIESCALAR @args; # constructor`
- Further methods get called when using $var
- removing the special interface: `untie $var;`

Tie Applications

- The following data types can be tied
  - Scalars, Arrays, Hashes, File Handles
- Many interesting applications using this mechanism
  - many modules on CPAN dealing with tie
  - using hashes most popular (like for object storage)
  - Tied file handles most recent
  - many tie modules provide interfaces to databases (LDAP, DB(UNIX) based, DBI(perl) based)

Core Perl Database Support

- known from UNIX: DBM and similar "simple" DB's
  - at least one of SDBM, NDBM, GDBM, MLDBM, Berkeley DB contained in core Perl
  - all these DB's implement a disk based hash
  - used when single table with (key, value) sufficient
- Berkeley DB is first choice (simultaneous updates, transactions)
  - installed at DESY (both on NT and UNIX)
- SDBM (simple Database) guaranteed to be implemented in perl
- MLDBM has support for storing data structures as values
- Perl provides an interface to tie these databases to hashes

Usage of Databases with tie

- Connecting to a database means calling **tie**

```
use Fcntl;      #Constants O_RDWR and O_CREAT
use DB_File;    #Berkeley DB
tie %h,'DB_File',$file,O_RDWR|O_CREAT,0666,$DB_BTREE;
```

- Read and write is done using hash assignments

```
$val=$h{key1}; $h{key2}='new text';
```

- Erasing rows means deleting a key/value pair

```
delete $h{key2};
```

- Saving changes back to disk by calling **untie**

```
untie %h;
```

A working example

```
use Fcntl;     # for the constants O_RDWR and O_CREAT
use SDBM_File; # Simple DB, in Perl always available
tie %hash,'SDBM_File','C:\Temp\mydb',O_RDWR|O_CREAT,0666;
$hash{key1} = 11.2;
$hash{key2} = 'text';
$date = localtime(time); #use the date stamp as key
$hash{$date} = '';
print "In memory: ", join ("\n\t", keys %hash), "\n";
untie %hash;
print "After untie:", keys %hash, "\n";
tie %hash,'SDBM_File','C:\Temp\mydb',O_RDWR|O_CREAT,0666;
print "Read from file: ", join ("\n\t", keys %hash), "\n";
untie %hash;
```

Other DBM Applications

- NIS (Yellow Pages) maps are DBM Files
  - simple access using the tie mechanism
- Converting between different DBM formats
  - by two different tie calls and copying **`%new=%old`**
- Text file manipulation using Berkeley DB(RECNO)
  - then simple line addressing possible
- See **`perldoc AnyDBM`** for a comparison of DB's
- Try to use **`tie`** instead of **`dbmopen`** (portability)

Editing text with Berkeley DB

```
use DB_File;
tie @lines, 'DB_File', 'tfile', O_RDWR|
  O_CREAT, 0666,$DB_RECNO;
$lines[0] = 'New first line';
push @lines, 'yet another new line';
$lines[5] = 'replacement for line 6';
$lines[8] = 'last line';
$lines[-1] = 'remove this line later';
$last = pop @lines; #last line gets removed
untie @lines;
```

Databases and DBI

- DBI (Data Base Interface) is the binding glue between perl and relational SQL databases
  - has the components DBI (database independent)
  - and the specific DBD's (Data Base Drivers)
- Homogeneous API to access different RDBMS
  - coding of applications independent from RDBMS
- Similar concept in the Windows world (ODBC)
- Access to databases with ODBC Interface with
  - DBD::ODBC or Win32::ODBC

Further information on DBI

- **`perldoc DBI; perldoc DBD::Oracle ...`**
- Programming the Perl DBI, Alligator Descartes & Tim Bunce, O'Reilly (2000)
- DBI Homepage: http://dbi.perl.org
- DBI talks by Tim Bunce (files DBI_Talk…tar.gz):
- ftp://ftp.uni-hamburg.de:
  /pub/soft/lang/perl/CPAN/authors/id/TIMB/

Structured Query Language

- Required to work with a RDBMS
- Perl passes SQL to RDBMS, no checks done
- Here only simple constructs mentioned

  INSERT INTO table (colx, coly, ...) VALUES (val1, val2, ...)

  UPDATE table SET colx = val1 WHERE coly LIKE val2

  DELETE FROM table WHERE colz=num1

  SELECT colx, coly, ... FROM table WHERE ... ORDER BY colz, ...

- Different syntax already for WHERE test operators
  - string comparison: UPPER(val) LIKE .. or val CLIKE ..

List of supported RDBMS

- Many RDBMS supported (CPAN), at DESY:
  - Oracle, mysql
  - access to ODBC and ADO databases (Windows)
  - CSV text files (comma separated values)
  - **all** above mentioned databases on **all platforms** using the intermediate Proxy driver
- Usage with

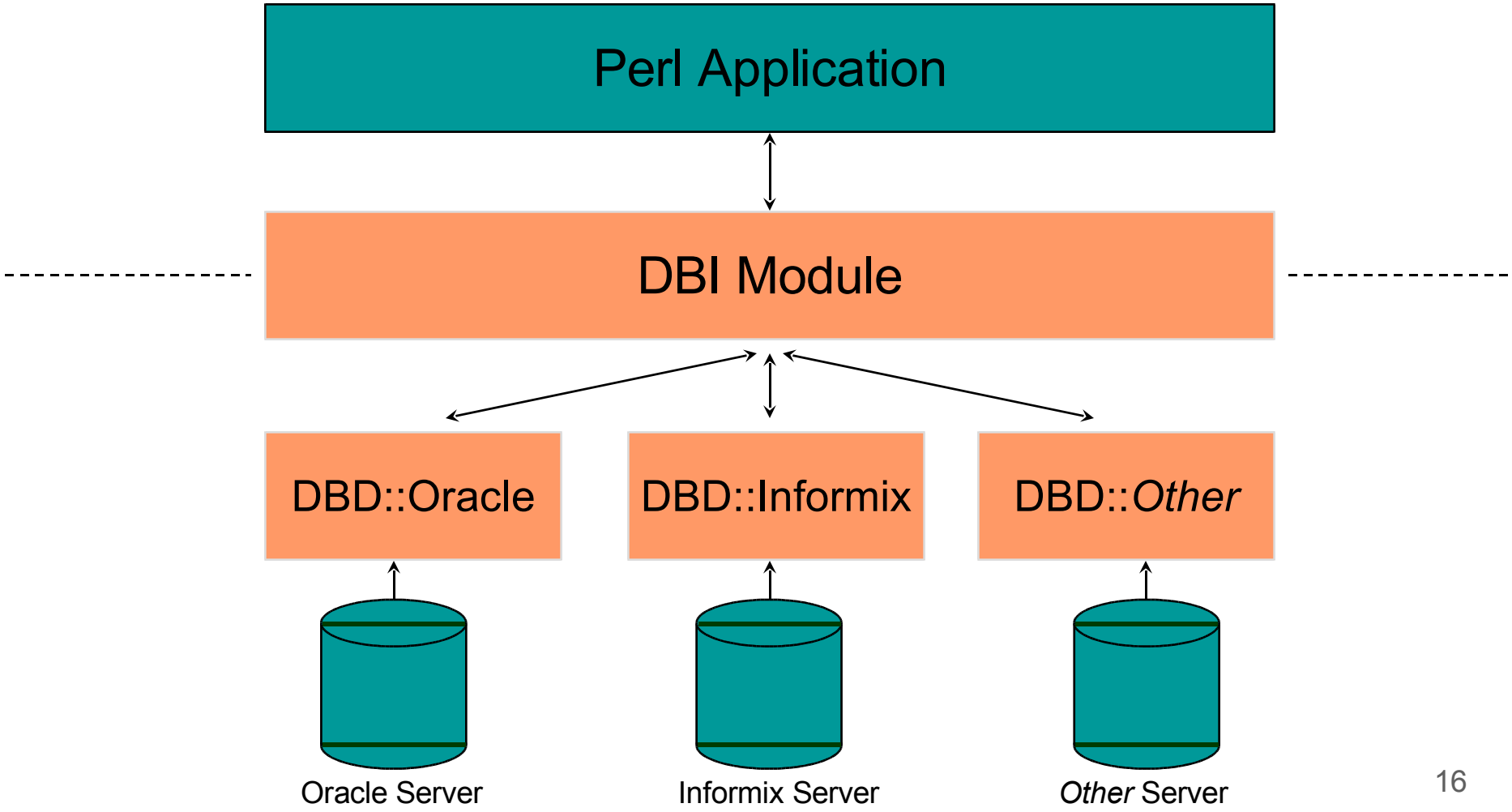
```
use DBI; #use DBD::xxx usually not required
```

- Some RDBMS have additional requirements

The DBI/DBD Interface

- DBI defines a set of methods, which are implemented in the drivers for the various RDBMS
- DBD can define additional methods that are database specific
- Methods at the database or table level require a **database handle**, is obtained using `connect`
- Operations within a table require a **statement handle**, is obtained e.g. with `prepare`

DBI diagram (from Talk T. Bunce)



Access to Oracle from DBI

- Windows Users have to install SQLPlus (Netinstall)
- UNIX Users **have to** set ORACLE_HOME:

```
$ENV{ORACLE_HOME} =
'/afs/desy.de/products/oracle/product/rdbms'
```

- Windows users **must not** set this variable

Querying a Database (Oracle)

replace for other RDBMS, DB, table

```
use DBI;
$dbname = 'dbi:Oracle:desy';
$ENV{ORACLE_HOME}='/afs/desy.de/products/oracle/product/rdbms'
   if $dbname =~ /dbi:Oracle:/i and $^O ne 'MSWin32';
$dbuser = $ENV{ORACLE_USERID} || 'read/read';
$dbh = DBI->connect($dbname, $dbuser, '');
$sth = $dbh->prepare(qq{select * from
   bolewski.teilnehmer where NAME like ?});
$sth->execute('Fri%'); # insert parameters for ?
$fieldnames = $sth->{'NAME'}; # field names
while ($row = $sth->fetchrow_arrayref)  {
  print "$row->[1]: $fieldnames->[2]=$row->[2]\n"; }
$sth->finish;      $dbh->disconnect;
```

Optimization for Speed

- Use `connect_cached` instead of **`connect`**
  - new database connections are expensive
- Use `prepare` and `execute` instead of **`do`**
  - one **`prepare`** call can be reused for many **`execute`**s
  - use placeholders ? in **`prepare`** and substitute them by using **`execute`** arguments
- Use **`fetchrow_arrayref`** instead of **`fetchrow_array`**
  - transferring pointers causes less data moves

Error handling

- Most DBI methods return undef on error
- Then **`$DBI::errstr`** contains the error message
- Automatic error checking can be switched on
  - **`$handle->{RaiseError}=1; #die on error`**
  - **`$handle->{PrintError}=1; #warn on error`**
  - **`$DBI::errstr`** gets printed in both cases
- Error handling can be done using **`eval`**

```
$handle->{RaiseError} = 1;
  eval{ ...; $handle->method; ...};
  if ($@) { ... better error handling ... }
```

Debugging

- DBI has built in tracing (globally or at handle level)
- Tracing can be controlled using
  - `DBI->trace($level);  # global tracing`
  - `$handle->trace($level); # at handle level`
- Redirecting trace output into a file
  - `$handle->trace($level, $file);`
- Can be controlled with ENV variable `DBI_TRACE`

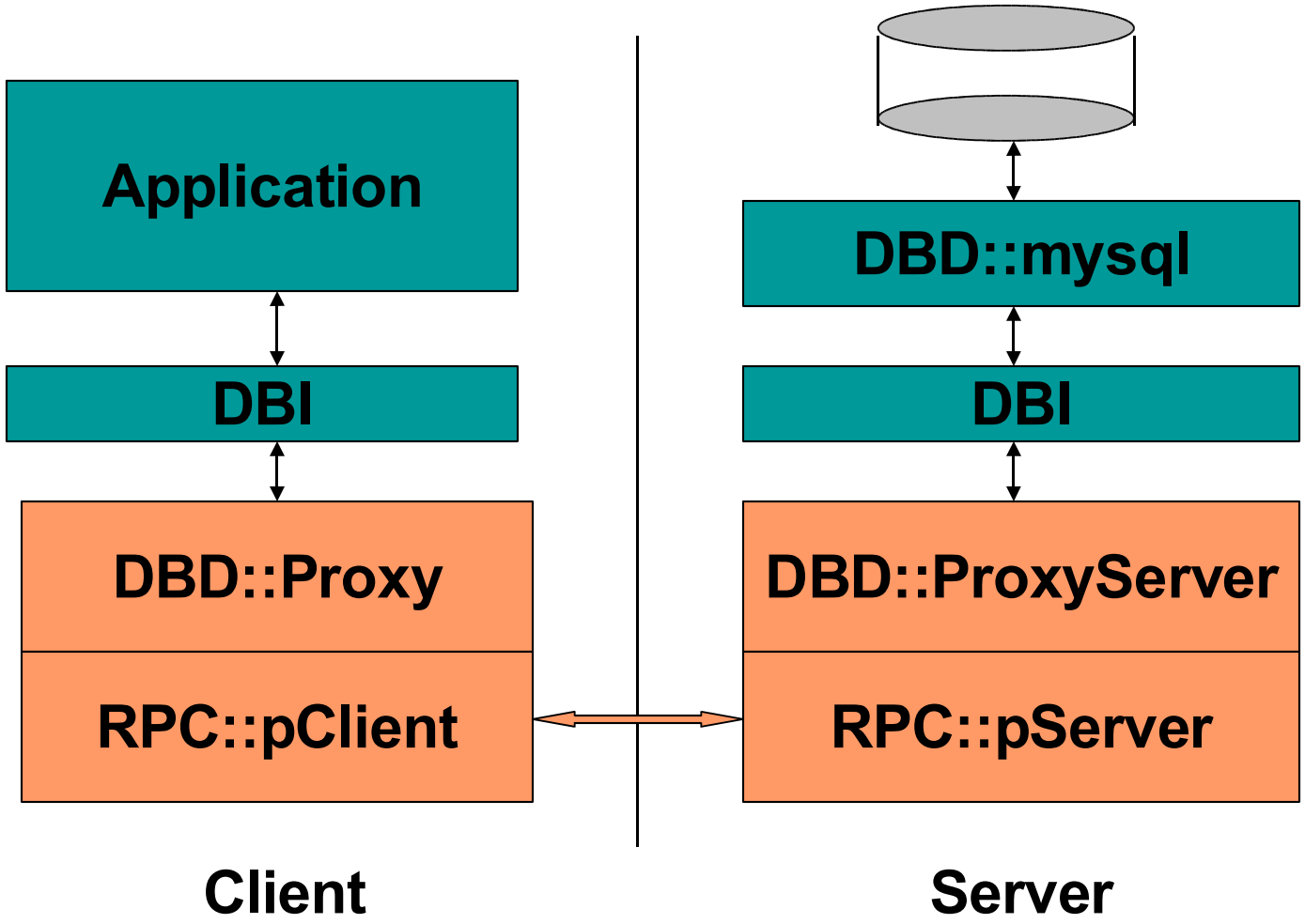
  `DBI_TRACE=`*`level`* `DBI_TRACE=`*`file`* `DBI_TRACE=`*`level=file`*

  usually levels 1 and 2 are more useful

Database Proxies

- Used to connect to databases where
  - the database itself is not remotely accessible
  - there is no driver for the current platform
  - only a limited number of clients should have direct access
- DBI::ProxyServer needs to get access to the DB
- DBI::Proxy connect to the server using TCP
- Proxy package has built in configurable data compression, encryption and access control

The Proxy Architecture



Using a Proxy Server

- Proxy Server gets started from command line
  - **`dbiproxy --localport`** ***`portnumber`***

  (UNIX: in /products/perl/bin, for NT in PATH)
- Standalone Application becomes client by
  - changing the DB name in the connect call or
  - setting DB name in ENV variable **`DBI_AUTOPROXY`**, then **no change** of (standalone) application!

mysql access using a proxy

- mysql driver only on NT, Linux and Solaris installed
- Start a proxy server on one of the above platforms

```
use DBI;
$dsn = 'DBI:mysql:test_zeulist;host=mysqlsrv.ifh.de';
# dsn has to begin with dbi:xxx: and be last part of $dbname !!!
#$dbname = "DBI:Proxy:hostname=ilos.ifh.de;port=1206;dsn=$dsn";
$ENV{DBI_AUTOPROXY} = 'hostname=ilos.ifh.de;port=1206';
$dbh=DBI->connect($dsn, 'readuser'); #$dbh=DBI->connect($dbname,..);
$sth = $dbh->prepare("SELECT * FROM phone WHERE lastname like ?");
$sth->execute('Vogt');
while (@row = $sth->fetchrow_array)  {
  print join(", ", @row[1,2]), "\n"; }
$sth->finish;
$dbh->disconnect;
```

Portability of DBI Code

- DBI based applications platform independent as perl and DBI working under UNIX and Windows
- ProxyServer increases flexibility (acces of Windows only databases like Access from UNIX!)
- DBI code not completely independent of RDBMS
  - SQL dialects and SQL extensions vary with RDBMS
  - DBD driver limitations
- For RDBMS independence restrict SQL usage to simple constructs and code remaining functionality in perl (may even be useful for time critical programs)

Advanced DBI functionality

- Multithreading for DBI
- Forking DBI server (since perl 5.6 also for NT)
- DBD::Multiplex driver
  - to keep several DB's in sync
  - load balancing for queries
  - fail safe operation
  - consistence checking of several DB's
- Tie::DBI to completely hide the SQL syntax
  - e.g. `$hash{table}->{field} = 42;#SQL UPDATE`