PHP 5 and Databases

Marcus Börger

Intro

☑ Review of PHP 4 Situation

☑ PHP 5 News

☑ PHP 5 Situation

PHP and Databases

- ☑ PHP can connect to all important RDBMs
  - ☑ Oracle
  - ☑ PostgreSQL
  - ☑ MySQL
  - ☑ MS-SQL
  - ☑ mSQL
  - ☑ Sybase
  - ☑ Interbase/Firebird
  - ☑ ODBC

  - ☑ DBM-style databases

PHP 4 Situation

☑ PHP can connect to all important RDBMS

☒ Each RDBMS needs a separate extension

☒ Each extension has a different interface

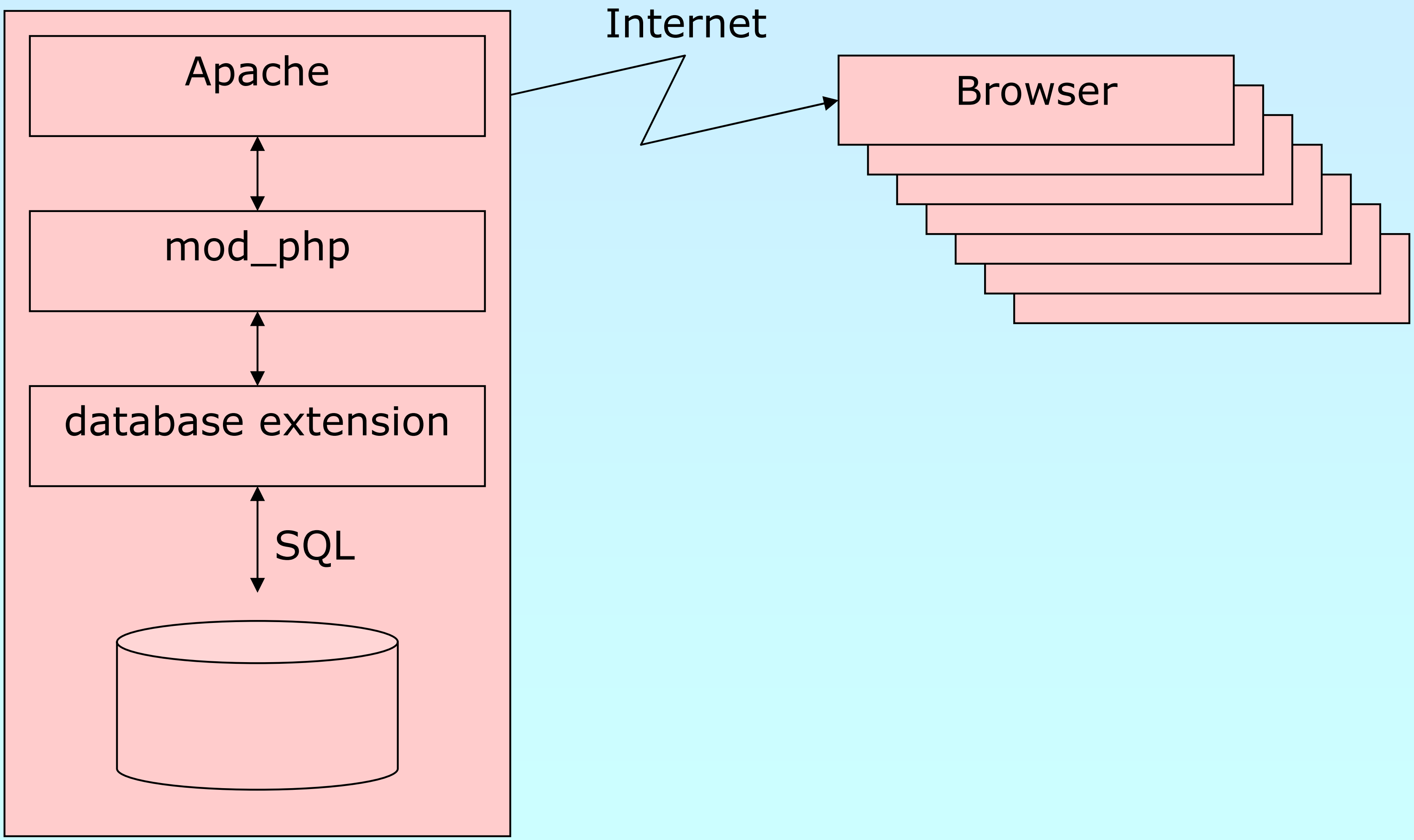
- ☒ ext/dbx is an inefficient abstraction

☒ Multiple PEAR solutions

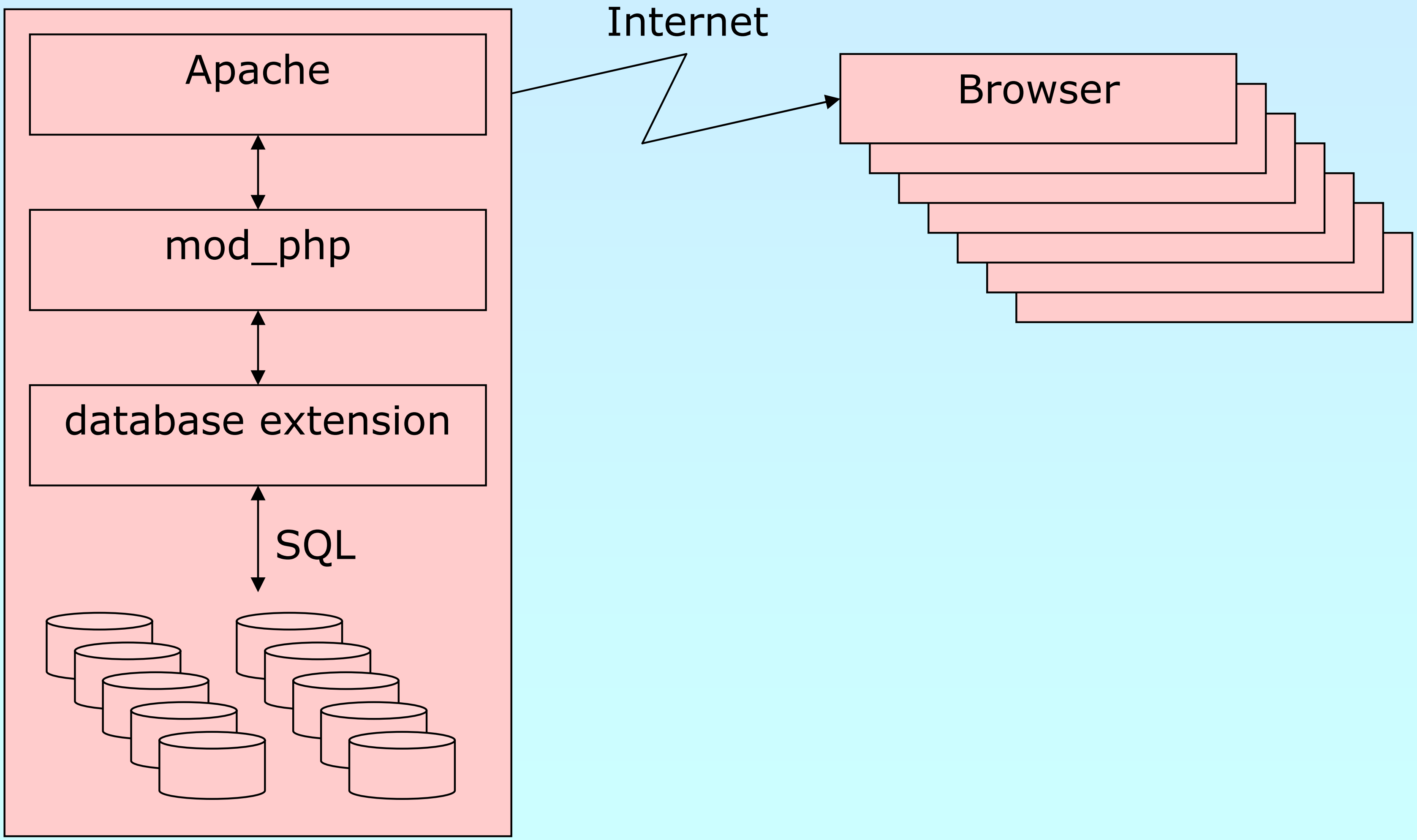
- ☑ Abstraction layers
- ☑ Query builders
- ☑ Data Access Objects . . . Nested Set support

- ☒ But there is ‘no’ OO in PHP 4

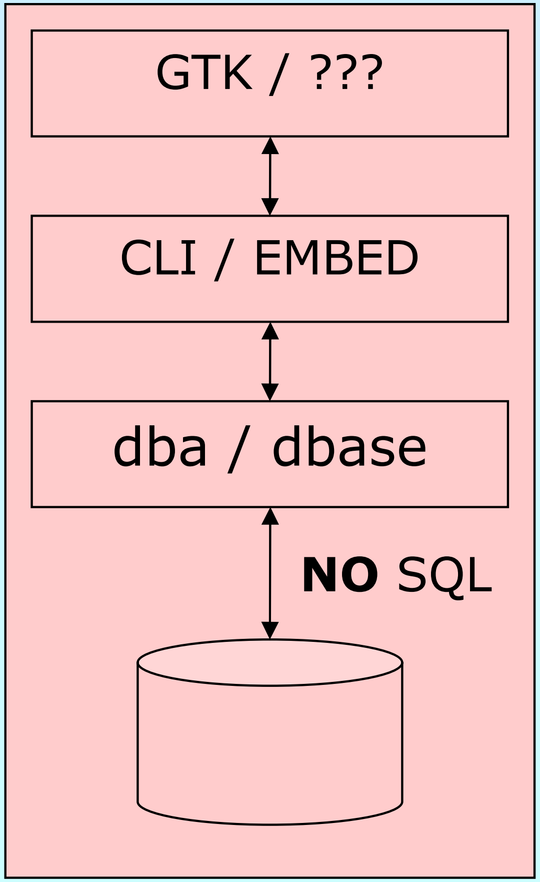
Dedicated Host



ISP/Shared Host



Embedded



PHP5 is the future

☑ New SAPIs

☑ New internal features

☑ New extensions

☑ ZendEngine 2 and its revamped object model

ZendEngine 2 and its revamped object model

☑ Objects are referenced by identifiers
☑ Constructors and Destructors
☑ Static members
☑ Default property values
☑ Constants
☑ Visibility
☑ Interfaces
☑ Final and abstract members
☑ Interceptors
☑ Exceptions
☑ Reflection API

New extensions

☑ New extensions

- ☑ DOM
- ☑ MySQLi
- ☑ PDO
- ☑ PHILI
- ☑ SimpleXML
- ☑ SPL
- ☑ SQLite
- ☑ XML + XSL

New extensions: MySQLi

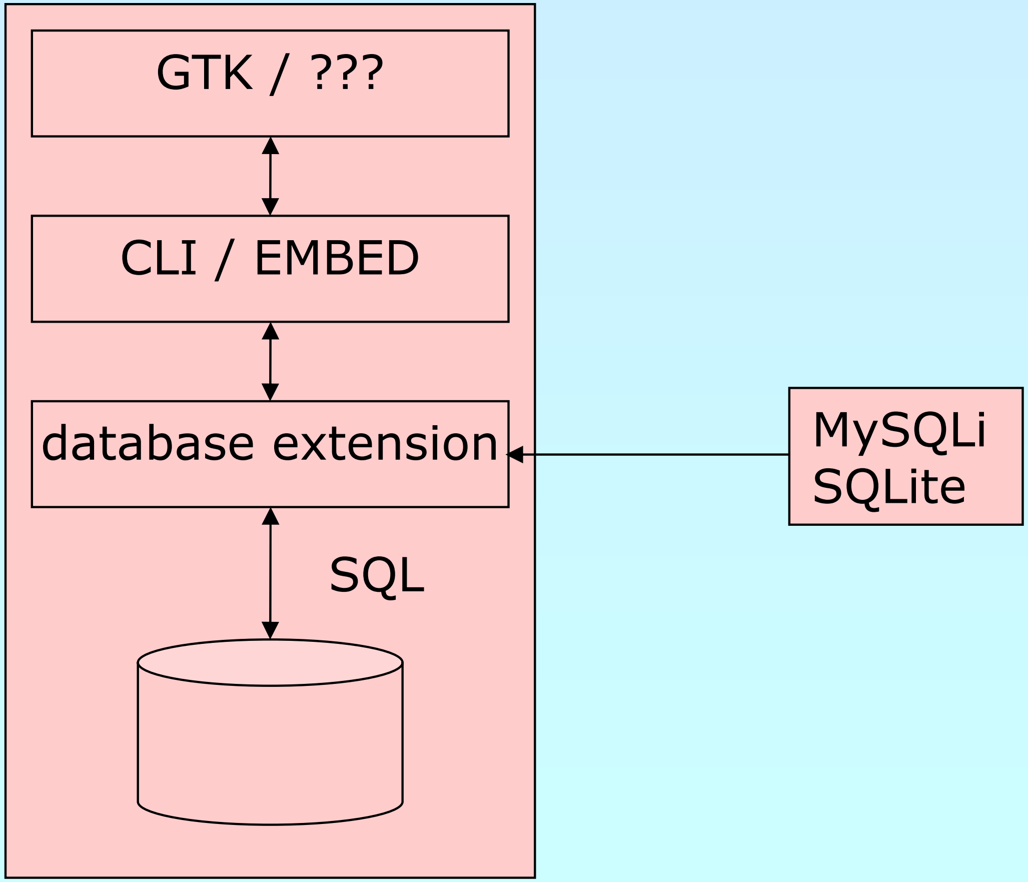
☑ Mysql grows to become more and more an enterprise ready RDBMS but sticks to its origin fastness, easiness

☑ PHP5 reflects this development by providing a new extension named MySQLi

☑ Support for MySQL embedded into PHP

☑ Implements new MySQL features
  - ☑ Profiling queries
  - ☑ Analyzing queries: bad index or no index used

Embedded



New extensions: SQLite

☑ Started in 2000 by D. Richard Hipp

☑ Single file database

☑ Subselects, Triggers, Transactions, Views

☑ Very fast, 2-3 times faster than MySQL, PostgreSQL for many common operations

☑ 2TB data storage limit

☒ Views are read-only

☒ No foreign keys

☒ Locks whole file for writing

New extensions: SQLite

☑ PHP extension bundled with PHP 5

☑ Available via PECL since PHP4.3

☑ Used on php.net

☑ SQLite library integrated with PHP extension

☑ API designed to be logical, easy to use

☑ High performance

☑ Convenient migration from other PHP database extensions

☑ Call PHP code from within SQL

SQLite: Calling PHP from SQL

```
bool sqlite_create_function (resource db,
          string funcname, mixed callback [,
          long num_args ])
```

- Registers a "regular" function

```
bool sqlite_create_aggregate (resource db,
          string funcname, mixed step,
          mixed finalize [, long num_args ])
```

- Registers an aggregate function

SQLite: Calling PHP from SQL

```
<?php
  function md5_and_reverse($string) {
      return strrev(md5($string));
  }

  sqlite_create_function($db,
      'md5rev', 'md5_and_reverse');

  $rows = sqlite_array_query($db,

      'SELECT md5rev(filename) FROM files');

?>
```

SQLite: Calling PHP from SQL

```
<?php
    function max_len_step(&$context, $string) {
        if (strlen($string) > $context) {
            $context = strlen($string);
        }
    }

    function max_len_finalize(&$context) {
        return $context;
    }

    sqlite_create_aggregate($db,
        'max_len', 'max_len_step', 'max_len_finalize');

    $rows = sqlite_array_query($db,
        'SELECT max_len(a) FROM strings');
?>
```

New extensions: SPL

☑ SPL aka Standard PHP Library

☑ Filter iterators

```
<?php
interface Iterator {
  function rewind();
  function hasMore();
  function current();
  function key();
  function next();
}
?>
```

```
<?php
class Filter implements Iterator {
  function __construct(Iterator $input)...
  function rewind()...
  function accept($value)...
  function hasMore()...
  function current()...
  function key()...
  function next()...
}
?>
```

```
<?php
  $it = get_resource();
  foreach($it as $key=>$val) {
  // access data
}
?>
```

```
<?php
$it = get_resource();
for ($it->rewind(); $it->hasMore(); $it->next()) {
  $value = $it->current(); $key = $it->key();
}
?>
```

```
<?php
$it = get_resource();
foreach(new Filter($it, $filter_param) as $key=>$val) {
  // access filtered data only
}
?>
```

New extensions: PDO

- ☑ PDO aka PHP Data Objects
- ☑ Object oriented RDBMS abstraction
  - ☑ Sqlite
  - ☑ Mysql
  - ☑ PostgreSQL
  - ☑ ...
- ☑ Provides efficient data access strategies
- ☑ Hybrid function/method approach

```
<?php
$db = pdo_connect(…);
$res= pdo_query($db, $sql);
?>
```

```
<?php
$db = pdo_db::connect(…);
$res=$db->queryArray($sql);
?>
```

PDO: Query Functions

pdo_result pdo_db::queryBuffered(string sql **[**,
int result_mode**]**)

- ☑ Buffered query = Flexible
- ☒ More memory usage
- ☑ Also have a fast unbuffered variant:
  pdo_unbuffered pdo_db::queryUnbuffered

array **pdo_db::queryArray**(string sql **[**,
int result_mode**]**)

- ☑ Flexible, Convenient
- ☒ Slow with long result sets

mixed **pdo_db::querySingle**(string sql **[**,
bool first_row_only**]**)

- ☑ Fast
- ☒ Only returns the first column

PDO: Array Interface

array **pdo_unbuffered::fetchArray** (**[**int result_mode**]**)

☑ Flexible

☒ Slow for large result sets

array **pdo_unbuffered::fetchAll** (**[**int result_mode**]**)

☑ Flexible

☒ Slow for large result sets; better use **pdo_db::queryArray** ()

PDO: Default result mode PDO_NUMERIC

```php
<?php
  $res = $db->queryBuffered(
             'SELECT first, last FROM names');
  $row = $res->fetchArray();
  print_r($row);
?>
```

```
Array
(
    [0] => Joe
    [1] => Internet
)
```

PDO: Column names only PDO_ASSOC

```
<?php
  $res = $db->queryUnbuffered(
                 'SELECT first, last FROM names',
                 PDO_ASSOC);
  $row = $res->fetchArray();
  print_r($row);
?>
```

```
Array
(
    [first] => Joe
    [last] => Internet
)
```

PDO: Column name and index: PDO_BOTH

```
<?php
  $res = $db->queryUnbuffered(
           'SELECT first, last FROM names');
  $row = $res->fetchArray(PDO_BOTH);
  print_r($row);
?>
```

```
Array
(
    [0] => Joe
    [1] => Internet
    [first] => Joe
    [last] => Internet
)
```

PDO: Collecting all rows

```php
<?php
  // Get the rows as an array of arrays of data
  $rows = array();

  $res = $db->queryUnbuffered(
      'SELECT first, last FROM names');

  // grab each row
  while ($row = $res->fetchArray()) {
      $rows[] = $row;
  }

  // Now use the array; maybe you want to
  // pass it to a Smarty template
  $template->assign('names', $rows);

?>
```

PDO: Querying all rows

```
<?php
   // The same but with less typing and
   // more speed

   // Get the rows as an array of arrays of data
   $rows = $db->queryArray(
         'SELECT first, last FROM names');

   // give it to Smarty
   $template->assign('names', $rows);
?>
```

PDO: Querying objects

```
<?php
  class Person {
     protected $first = '';
     protected $last = '';
     protected $db;
     function getFirst() { return $this->first; }
     function getLast() { return $this->last; }
     function __construct($db) { $this->db = $db; }
  }

  // Get all data
  $rows = $db->queryUnbuffered (
      'SELECT first, last FROM names');

  // Fetch data into an Instance of class Person
  $person = $rows->fetchObject('Person', array($db));
?>
```

PDO: Single Column Interface

mixed **pdo_db::singleQuery** (string sql **[**,
bool first_row_only**]**)

☑ Fast

☒ Only returns the first column

string **pdo_unbuffered::fetchSingle** (
[mixed which_column])

☑ Fast

☑ Flexible, Faster than array functions

☒ Slower than **pdo_db::singleQuery()**

PDO: Query a single value

```php
<?php

  $count = $db->singleQuery($db,
       'SELECT count(first) FROM names', 1);

  echo "There are $count names";
?>
```

```
There are 3 names
```

PDO: Query single columns

```
<?php

  $first_names = $db->singleQuery(
      'SELECT first FROM names');

  print_r($first_names);
?>
```

```
Array
(
    [0] => Joe
    [1] => Peter
    [2] => Fred
)
```

PDO: Iterator Interface

array **pdo_unbuffered::current** (**[** int result_mode**]**)

- ▣ Returns the current selected row

bool **pdo_unbuffered::next** / **pdo_result::prev** ()

- ▣ Moves to next / previous row

bool **pdo_unbuffered::hasMore**/ **pdo_result::hasPrev**()

- ▣ Returns true if there are more / previous rows

bool **pdo_result::rewind** ()

- ▣ Rewind to the first row of a buffered query

bool **pdo_result::seek** (int row)

- ▣ Seeks to a specific row of a buffered query

PDO: Using Iterators

```
<?php
    $db = pdo_db::connect('...');
    for ($res = $db->queryUnbuffered('SELECT...');
           $res->hasMore();
           $res->next())
    {
           print_r ($res->current());
    }
?>
```

```
<?php
    $db = pdo_mysql::connect('...');
    foreach ($db->queryUnbuffered('SELECT...') as $row)
    {
           print_r ($row);
    }
?>
```

Performance

10 times Querying 10 rows using SQLite

☑ Iterators vs. query and fetch Array

- ☑ As engine hooks: 90% (scaling linear)
- ☑ Implemented as engine feature: 56%

⌧ Building an Array is expensive

☑ queryArray vs. query and fetchArray: 89%

⌧ Function calls are expensive

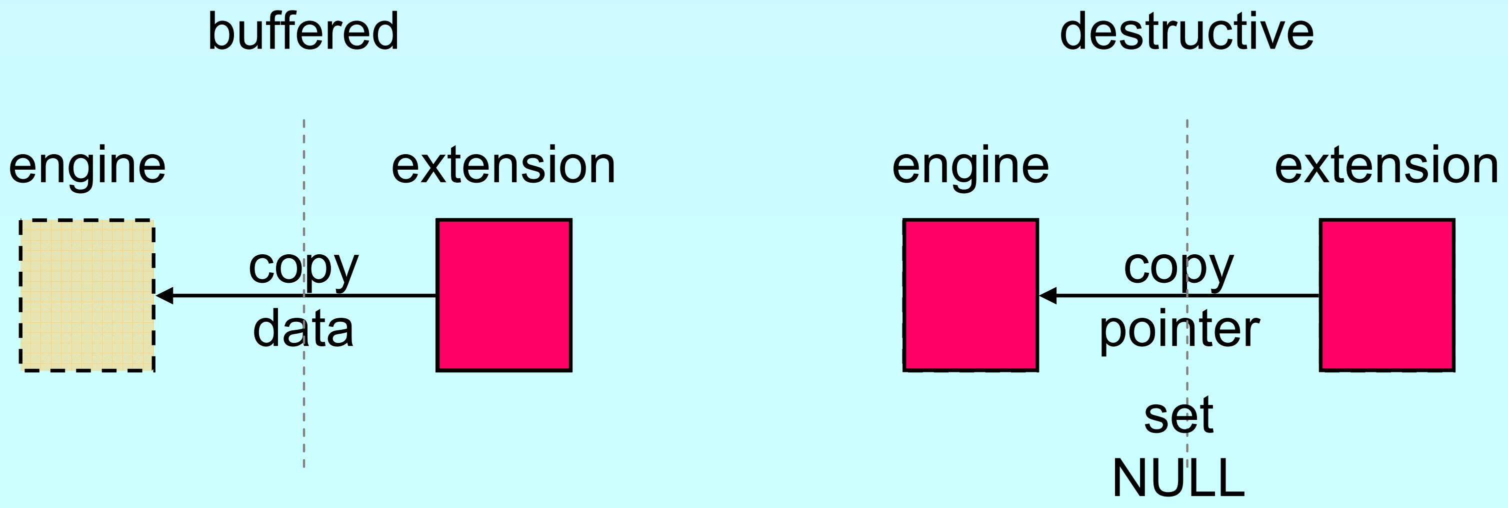
Performance

10 times Querying 10 rows using SQLite

☑ Buffered vs. Unbuffered: up to 60%

- ☑ Buffered queries need to build a hash table
- ☑ Buffered queries must copy data
- ☑ Unbuffered queries can use **destructive reads**

⌦ Copying data is expensive



Performance

☑ Comparing OO vs. Procedural code

- ☑ PC is easy to program?
- ☑ PC uses resources: O(n*log(n))
- ☑ PC uses a single function table: 2000 ... 4000

- ☑ OO code is little bit more to learn
- ☑ OO code is easy to maintain
- ☑ OO code uses object storage: O(n+c)
- ☑ OO uses small method tables: 10 ... 100

PHP 4 Situation

- ☑ PHP can connect to all important RDBMS
- ☒ Each RDBMS needs a separate extension
- ☒ Each extension has a different interface
  - ☒ ext/dbx is an inefficient abstraction
- ☒ Multiple PEAR solutions
  - ☑ Abstraction layers
  - ☑ Query builders
  - ☑ Data Access Objects . . . Nested Set support
  - ☒ But there is 'no' OO in PHP 4

PHP 5 Situation

- ☑ PHP can connect to all important RDBMS
- ☑ PDO provides a unified efficient abstraction
- ☑ PHP is ready for UML
- ☑ Specialized extensions allow detailed control
- ☑ Multiple PEAR solutions
  - ☑ More sophisticated abstraction layers
  - ☑ Query builders
  - ☑ Data Access Objects . . . Nested Set support
- ☑ Multiple ways of using databases with PHP
  - ☑ File based as ext/dba or ext/sqlite or embedded MySQL
  - ☑ Talking SQL with embedded RDBMS
  - ☑ Talking SQL with external RDBMS
  - ☑ Using ODBC