PHP mysqlnd plugins as an alternative to MySQL Proxy

Get in, hurry up!
The mysqlnd plugin talk starts!

international
PHP 2010
conference
- spring edition -

Ulf Wendel, MySQL/Sun/Oracle/WhatIsNext

The mysqlnd green house

Gardening mysqlnd – Concepts and Internals

Use the URL not the slides!

The slides are a quick copy of the real stuff at:

http://blog.ulf-wendel.de/mysqlnd_plugin_ipc2010.html

Target audience

PHP developers with
C programming knowledge.

Familarity with PHP
extension writing is benefitial.

„Das MySQL-Treibhaus erweitern ...
Level: Experte“, Session description

The speaker says...

Relax, it won't get that complicated.

I am trying to sieve out potential mysqlnd plugin implementors from the audience.

Quick poll: are you a man?

Raise your hand, if you call yourself a man.
Not a lamer, not a sissy - a real man.

Cowboys, who of you calls himself an Open Source developer?

Please state your name!

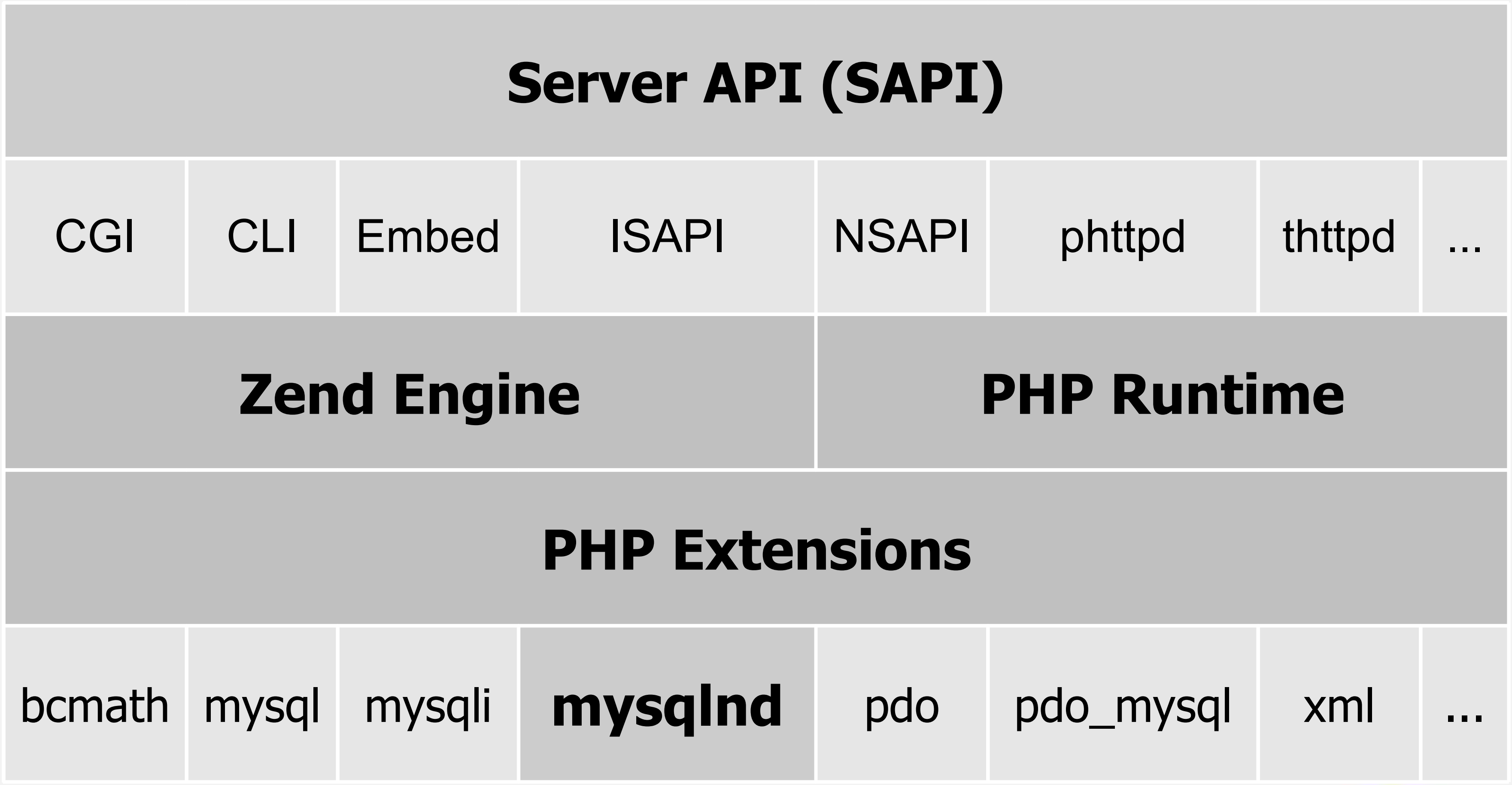
The pub, the idea, the demo

The speaker says…

No offline demo for Slideshare.

Maybe next time we meet in person! Cu!

The MySQL native driver for PHP



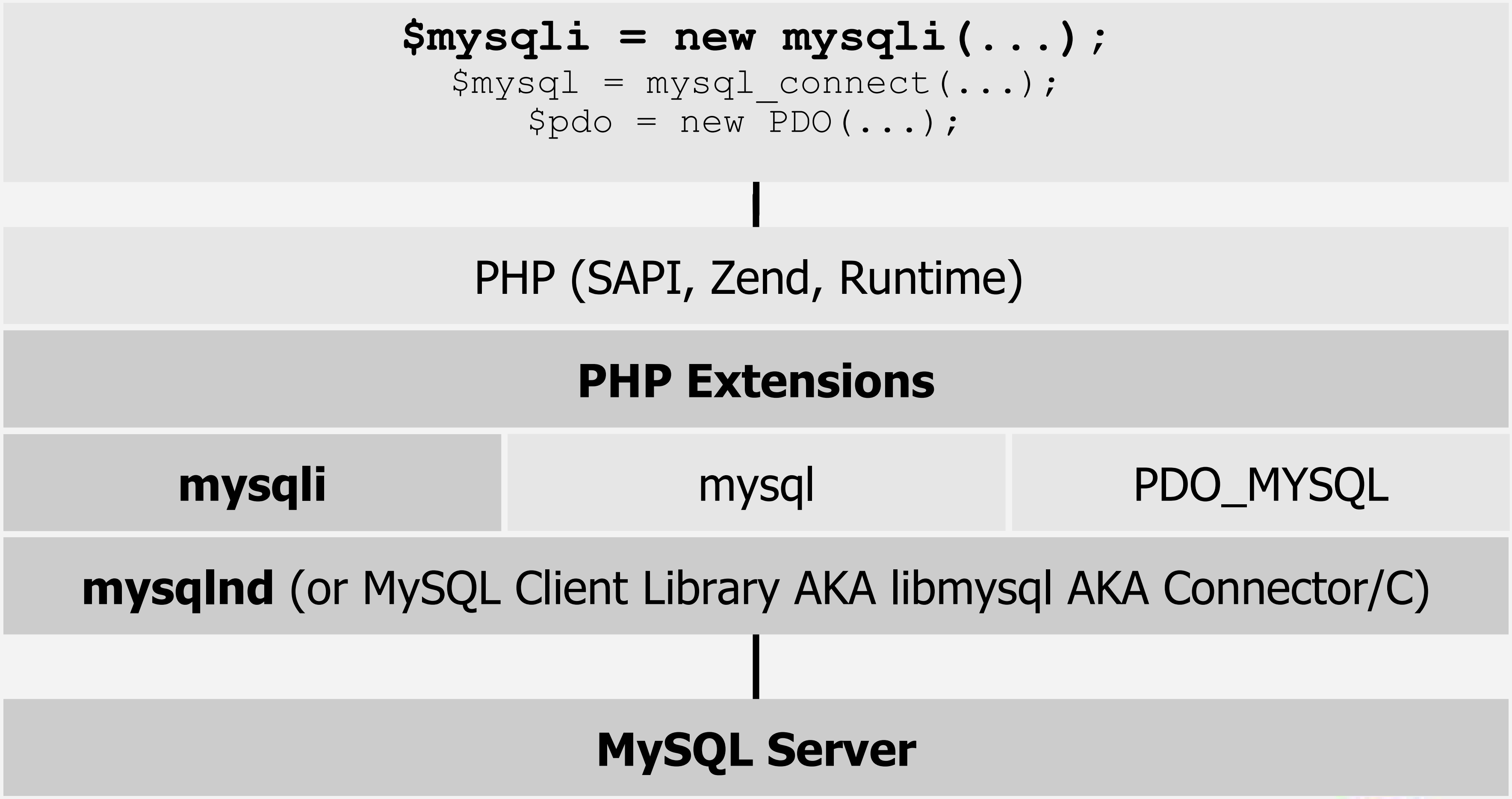
The speaker says...

The MySQL native driver for PHP (mysqlnd) is a C library which implements the MySQL Client Server Protocol. **It serves as a drop-in replacement for the MySQL Client Library (AKA libmysqlclient** AKA Connector/C).

mysqlnd is part of the PHP source code repository as of PHP 5.3. **Every PHP source code distribution contains it.**

mysqlnd is is a special kind of PHP extension. Like ext/PDO it does not export any userland functions. **It serves as a C library for other extensions**, like ext/PDO.

Replacement for libmysql



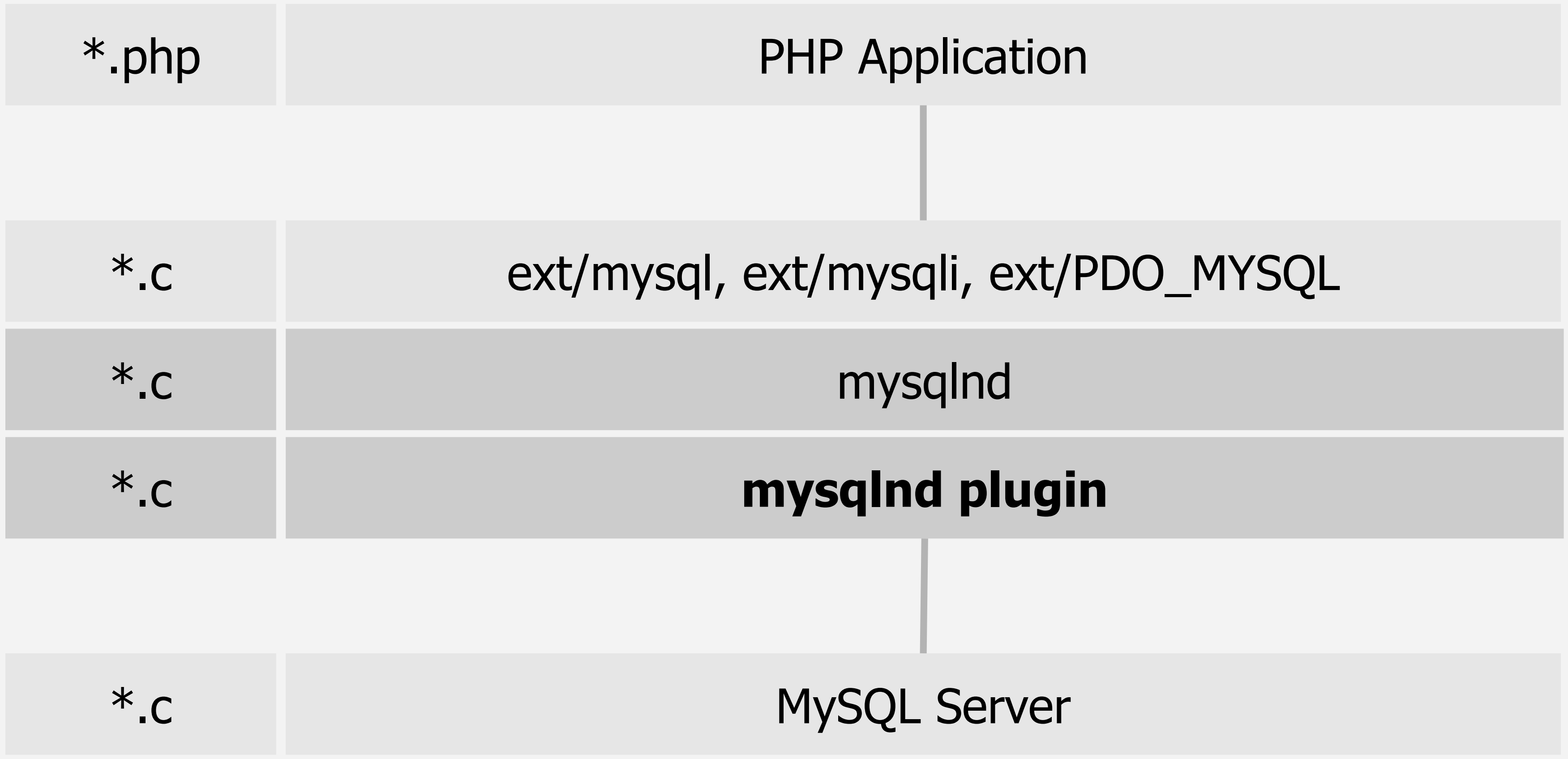
```
$mysqli = new mysqli(...);
$mysql = mysql_connect(...);
$pdo = new PDO(...);
```

PHP (SAPI, Zend, Runtime)
PHP Extensions
mysqli
mysql
PDO_MYSQL
mysqlnd (or MySQL Client Library AKA libmysql AKA Connector/C)
MySQL Server

The speaker says...

All PHP-MySQL APIs can either make use of the MySQL Client Library or mysqlnd to connect to MySQL. The decision to use one or the other library is made at compile time. Within a PHP binary you can mix mysqlnd and the MySQL Client Library as you like: one PHP MySQL API may use mysqlnd and another PHP MySQL extension may use the MySQl Client Library. To use the MySQL Client Library, you must have it installed on your system (at least when building PHP), whereas mysqlnd ships with PHP and thus has no pre-requisites.

How to grow/extend mysqlnd



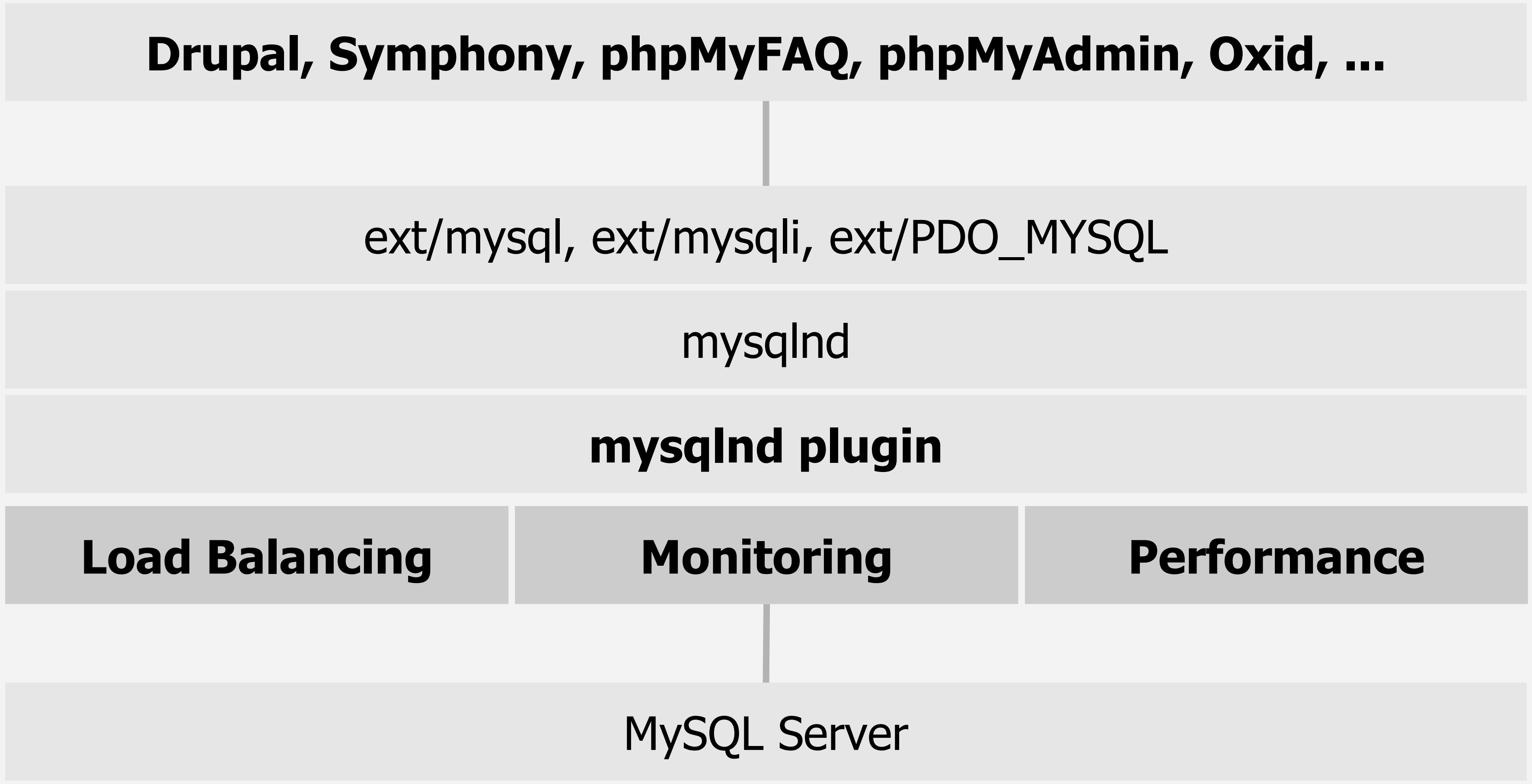
The speaker says...

The core feature set of mysqlnd is defined by the MySQL Client Libary feature set. It has taken about 15k lines of C (without comments) to implement the core functionality. Further growth will complicate maintenance. Further growth will hinder an understanding of the code base.

mysqlnd growth must be stopped without preventing the addition of new features. Some new features may be far beyond mainstream user requirements.

Good reasons to introduce mysqlnd "plugins". Plugins can hook and replace all mysqlnd C API calls.

What mysqlnd plugins can do!


Drupal, Symphony, phpMyFAQ, phpMyAdmin, Oxid, ...
ext/mysql, ext/mysqli, ext/PDO_MYSQL
mysqlnd
mysqlnd plugin
Load Balancing
Monitoring
Performance
MySQL Server

The speaker says...

A different way to name the plugin concept is to call it a "mysqlnd client proxy". From a PHP application point of view plugins can do everything that can be done using the MySQL Proxy product. For example:

- Load Balancing
  - Read/Write Splitting
  - Failover
  - Round-Robin, least loaded
- Monitoring
  - Query Logging
  - Query Analysis
  - Query Auditing
- Performance
  - Caching
  - Throttling
  - Sharding

What are mysqlnd plugins?

- Extension
  - Adds new functionality
- Proxy
  - Surrogate
  - Intermediary

The speaker says...

A better understanding of the capabilities of the "mysqlnd plugin concept" can be gained by distinguishing between extensions and proxies.

Extensions add new functionality to a software. For example, it is possible to add a new, alternative wire protocol implementation to mysqlnd for supporting the native Drizzle client server protocol.

Another way of understanding is to look at **plugins as proxies. This is the dominating one-sided viewpoint used in the following.**

Plugin vs. MySQL Proxy (I)

Hardware	**Software**		
Application server	PHP application	C/Java/.NET/PHP... application	
	mysqlnd plugin	**MySQL Proxy**	
Dedicated server			**MySQL Proxy**
Database server	MySQL Server		

The speaker says...

Hardware topology: MySQL Proxy can either be installed on the PHP application server or be run on a dedicated machine. A mysqlnd plugin always runs on the application server.

Running **a proxy on the application machines has two advantages:**

- **no single point of failure**
- **easy to scale out (horizontal scale out, scale by client)**

The speaker says...

Hardware topology: MySQL Proxy can either be installed on the PHP application server or be run on a dedicated machine. A mysqlnd plugin always runs on the application server.

Running **a proxy on the application machines has two advantages:**

- **no single point of failure**
- **easy to scale out (horizontal scale out, scale by client)**

Choose: C API or wire protocol

Layer	**Software**	
	PHP application	C/Java/.NET/PHP... application
C API	**mysqlnd plugin**	
Wire protocol	**mysqlnd plugin**	**MySQL Proxy**

The speaker says...

MySQL Proxy works on top of the wire protocol. With MySQL Proxy you have to parse and reverse engineer the MySQL Client Server Protocol. Actions are limited to what can be done by manipulating the communication protocol. If the wire protocol changes, which happens very rarely, MySQL Proxy scripts need to be changed as well.

Mysqlnd plugins work on top of the C API (and thus also top of the wire protocol). You can hook all C API calls. PHP makes use of the C API. Therefore you can hook all PHP calls. There is **no need to go down to the level of the wire protocol. However, you can, if you want.**

Follow me or leave the room!

Start your GCC's, boys!

- You want mysqlnd plugins because
  - 100% transparent = 100% compatible
  - Cure applications without touching .php
  - No extra software, no MySQL Proxy!
  - Extend, add driver functionality
- All you need to is
  - … write comments into comment files (*.c)

The speaker says...

mysqlnd plugins can be written in C and PHP - as we will see.

We need to look at C first. C is the "natural" choice. However, we will use it to carry the mysqlnd plugin functionality into the userspace.

mysqlnd modules

PHP Extension Infrastructure			
php_mysqlnd.c			
Core			
mysqlnd.c			
Modules			**Statistics**
Connection	**Resultset**	**Resultset Meta**	*statistics.c
mysqlnd.c	*_result.c	*_result_meta.c	
Statement	**Network**	**Wire protocol**	
*_ps.c	*_net.c	*_wireprotocol.c	

The speaker says...

Andrey Hristov is the core developer of mysqlnd. He is probably the only person in the world to know each line of mysqlnd inside out.

Andrey tried to modularize mysqlnd from the very beginning. First he created modules. Later on the modules became objects. Object oriented concepts crept into the design. Without knowing he had layed the foundations of what is called the mysqlnd plugin API today.

The above listed modules can be understood as classes. The classes can be extended by plugins.

mysqlnd modules are objects

```
struct st_mysqlnd_conn_methods {
  void (*init)(MYSQLND * conn TSRMLS_DC);
  enum_func_status (*query)(
     MYSQLND *conn, const char *query,
     unsigned int query_len TSRMLS_DC);
  /* ... 50+ methods not shown */
};

struct st_mysqlnd_connection {
  /* properties */
  char       *host;
  unsigned int  host_len;
  /* ... ... */
  /* methods */
  struct st_mysqlnd_conn_methods *m;
};
```

The speaker says...

Mysqlnd uses a classical C pattern for implementing object orientation.

In C you use a struct to represent an object. Data members of the struct represent properties. Struct members pointing to functions represent methods.

This always reminds me of PHP 4 but any comparison would only distract you.

The classes

	# public	#private (not final!)	#total
Connection	48	5	53
Resultset	26	0	26
Resultset Meta	6	0	6
Statement	35	1	35
Network	11	0	11
Wire protocol	10	0	10
Total	**136**	**6**	**142**

Revision 299098 = PHP 5.3 on May, 7 2010 -
Andrey continued working since then...

The speaker says...

Some, few mysqlnd functions are marked as private. Private does not mean final. It is possible to overwrite them but it is discouraged. Those private functions usually take care of internal reference counting.

Extending Connection: methods

```
/* a place to store orginal function table */
struct st_mysqlnd_conn_methods org_methods;

void minit_register_hooks(TSRMLS_D) {
  /* active function table */
  struct st_mysqlnd_conn_methods * current_methods
   = mysqlnd_conn_get_methods();

  /* backup original function table */
  memcpy(&org_methods, current_methods,
   sizeof(struct st_mysqlnd_conn_methods);

  /* install new methods */
  current_methods->query =
      MYSQLND_METHOD(my_conn_class, query);
}
```

The speaker says...

Plugins can overwrite methods by replacing function pointer.

Connection function table manipulations must be done at Module Initialization (MINIT). The function table is a global shared resource. In an threaded environment, with a TSRM build, the manipulation of a global shared resource during the request processing is doomed to cause trouble.

Do not use any fixed-size logic: new methods may be added at the end of the function table. Follow the examples to avoid trouble!

Extending: parent methods

```
MYSQLND_METHOD(my_conn_class, query)(MYSQLND *conn,
  const char *query, unsigned int query_len TSRMLS_DC) {

    php_printf("my_conn_class::query(query = %s)\n",
       query);

    query = "SELECT 'query rewritten' FROM DUAL";
    query_len = strlen(query);

    return org_methods.query(conn, query, query_len);
  }
}
```

international PHP2010 conference - spring edition -

The speaker says...

If the original function table entries are backed up, it is still possible to call the original function table entries - the parent methods.

However, there are no fixed rules on inheritance - it is all based on conventions. We will ignore this problem for now because we want to show how to use the plugin API. Once you have an understanding of the basics, we can talk about edge-cases.

In some cases, for example in case of Conn::stmt_init(), it is vital to call the parent method prior to any other activity in the derived method. Details will be given below.

Extending: properties (concept)

OO concept	mysqlnd C struct member	comment
Methods	struct object_methods * methods	Function table
Properties	c_type member	Parent properties
Extended properties	void ** plugin_data	List of void*. One void* per registered plugin

The speaker says...

Basic idea: allow plugins to associate arbitrary data pointer with objects.

See below for technical details.

Extending: properties (API)

```
void minit_register_hooks(TSRMLS_D) {
  /* obtain unique plugin ID */
  my_plugin_id = mysqlnd_plugin_register();
  /* snip - see Extending Connection: methods */
}

static PROPS** get_conn_properties(const MYSQLND *conn TSRMLS_DC) {

  PROPS** props;
  props = (PROPS**)mysqlnd_plugin_get_plugin_connection_data
                   (conn, my_plugin_id);

  if (!props || !(*props)) {
    *props = mnd_pecalloc
             (1, sizeof(MY_CONN_PROPERTIES), conn->persistent);
    (*props)->query_counter = 0;
  }
  return props;
}
```

The speaker says...

Arbitrary data (properties) can be added to a mysqlnd objects using an appropriate function of the mysqlnd_plugin_get_plugin_<object>_data() family. When allocating an object mysqlnd reserves space at the end of the object to hold void* to arbitrary data. mysqlnd reserves space for one void* per plugin.

The management of plugin data memory is your task (allocation, resizing, freeing)! See the below notes on constructors and destructors for hints.

Andrey recommends to use the mysqlnd allocator for plugin data (mnd_*loc()). This is not a must.

Daddy, it is a plugin API !

- mysqlnd_plugin_register()
- mysqlnd_plugin_count()

- mysqlnd_plugin_get_plugin_connection_data()
- mysqlnd_plugin_get_plugin_[result|stmt]_data()
- mysqlnd_plugin_get_plugin_[net|protocol]_data()

- mysqlnd_conn_get_methods()
- mysqlnd_[result|result_meta]_get_methods()
- mysqlnd_[stmt|net|protocol]_get_methods()

The speaker says…

It just happened, …

What you see is the first version. It is far from perfect. No surprise.

ABI breaks should become very rare, However, there may be API additions.

Risks: a (silly) man's world

- Security: sissy
- Limitations: use your leadfoot
- Chaining: alphabetical = random order

- Recommended to call parent methods
- Recommended to be cooperative

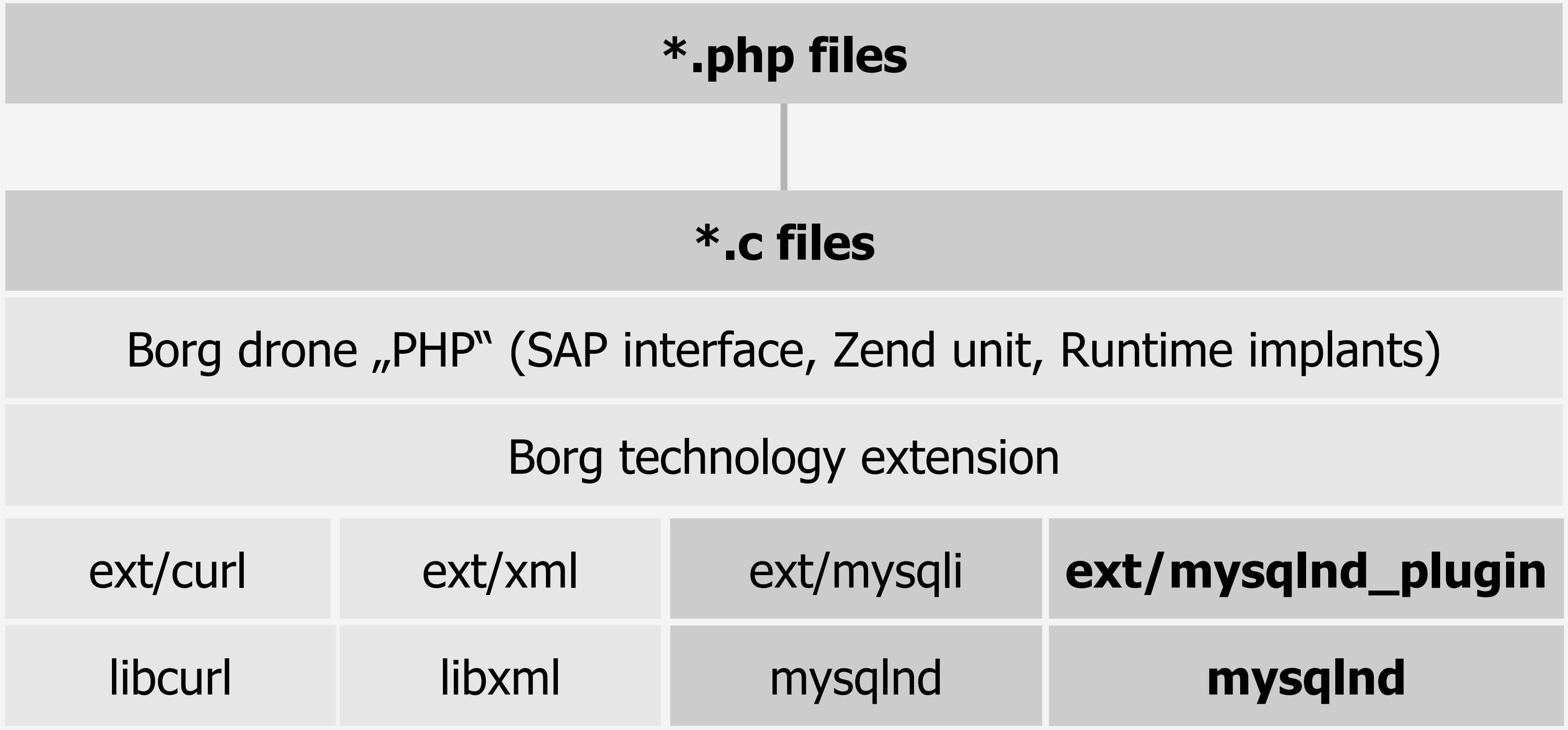
The speaker says...

No limits, take care!

A plugin has full access to the inner workings of mysqlnd. There are no security limits. Everything can be overwritten to implement friendly or hostile algorithms. Do not trust unknown plugins blindly . **Do not use unknown plugins before checking their source!**

As we saw, plugins can associate data pointer with objects ("properties"). The pointer is not protected from other plugins in any meaningful way. The storage place is not secure. Simple offset arithmetic can be used to read other plugins data.

On PHP, the borg and ladies



The speaker says...

Few PHP users can write C code. PHP users love the convenience of a script language. Therefore it is **desired to expose C APIs to the userland.**

PHP is like the borg: it assimilates all technology it finds useful. PHP has been designed to assimilate C libraries. Assimilated C libraries are called extensions. Most PHP extensions expose a PHP API for use in *.php files. Mysqlnd is a C library. A mysqlnd plugin is yet another C library implemented as a PHP extension. **Nothing stops you from writing a mysqlnd plugin which exposes the mysqlnd plugin API to PHP users - for use in *.php files!**

What borg technology can do!

```
class proxy extends mysqlnd_plugin_connection {
  public function connect($host, ...) {
    /* security */
    $host = '127.0.0.1';
    return parent::connect($host);
  }
  public function query($query, ...) {
    error_log($query);
    return parent::query($query);
  }
}
mysqlnd_plugin_set_conn_proxy(new proxy());
(auto_prepend.inc.php)

any_php_mysql_app_main();
(index.php)
```

The speaker says...

Guess what will happen when running the fictive code!

The application will be restricted to connect to '127.0.0.1'. Assuming the proposed ext/mysqlnd_plugin is the only active myqlnd plugin, **there is no way for the application to work around this restriction.** Same about the query logging.

It works with all PHP applications. And, it does not require any application changes.

Userspace plugin motivation

- Availability: maximum
- Creative potential: endless
- Ease of use: absolutely
- Fun factor: tremendous

- US citizen: you must read and comply to the following security rules
  - Security, Limitations, Chaining: consult homeland security
  - The C API has not been designed to be exposed to the userspace
  - Continued on page PHP_INT_MAX.

The speaker says...

It is about rapid protoyping, it is about simplified technology access.

If you ever plan to work with userspace mysqlnd plugins ask yourself twice if it may be better to contract a C developer. The internal mysqlnd API has not been designed as a plugin API for C, and **mysqlnd methods have certainly never been designed to be exposed to the userspace! If you give users access to C stuff, as proposed, they can easily crash PHP.**

Starting w. ext/mysqlnd_plugin

- How to create an extension? Choose!
  - Ueber-lady book (ULB) templates
  - Extension generators
  - Striping an existing extension
  - Extension skeleton

The speaker says…

BUY BUY – The Über-Lady book (ULB) – BUY BUY

Sara Golemon,

Extending and Embedding PHP

It will teach you all you need to develop PHP extensions. Without that wonderful book mysqlnd development would have taken twice as long as it took!

Repetition: MINIT of a plugin

```
static PHP_MINIT_FUNCTION(mysqlnd_plugin) {
  /* globals, ini entries, resources, classes */

  /* register mysqlnd plugin */
  mysqlnd_plugin_id = mysqlnd_plugin_register();

  conn_m = mysqlnd_get_conn_methods();
  memcpy(org_conn_m, conn_m,
    sizeof(struct st_mysqlnd_conn_methods));

  conn_m->query = MYSQLND_METHOD(mysqlnd_plugin_conn, query);
  conn_m->connect = MYSQLND_METHOD(mysqlnd_plugin_conn, connect);
}
(my_php_mysqlnd_plugin.c)

enum_func_status MYSQLND_METHOD(mysqlnd_plugin_conn, query)(/* ...
*/) {
  /* ... */
}
(my_mysqlnd_plugin.c)
```

The speaker says...

Jippie - ext/mysqlnd_plugin is half way done!

There is absolutely nothing new here. The purpose of the slide is to recall basics on mysqlnd plugins. We are putting pieces together to outline an extension as a whole.

Task analysis: C to userspace

```
class proxy extends mysqlnd_plugin_connection {
  public function connect($host, ...) { .. }
}
mysqlnd_plugin_set_conn_proxy(new proxy());
```

- write a class "mysqlnd_plugin_connection" in C (->ULB)
- accept and register proxy object through "mysqlnd_plugin_set_conn_proxy()" (->ULB)
- call userspace proxy methods from C (-> ULB or - optimization - zend_interfaces.h)

The speaker says...

References to "ULB" aim to point you to the book of the ueber-lady. Sarah Golemons' excellent book "Extending and Embedding PHP" (ULB) will teach you in depth how to write the code for those tasks. Because the book is truly outstanding - although a bit dated - **we will not discuss simple tasks marked with "-> ULB"**

Userspace object methods can either be called using call_user_function() as desribed in the ULB or you go one little step further down the ladder, closer to the Zend Engine and you hack zend_call_method().

Calling userspace

```
MYSQLND_METHOD(my_conn_class,connect)(
  MYSQLND *conn, const char *host /* ... */ TSRMLS_DC) {

  enum_func_status ret = FAIL;
  zval * global_user_conn_proxy = fetch_userspace_proxy();

  if (global_user_conn_proxy) {
    /* call userspace proxy */
    ret = MY_ZEND_CALL_METHOD_WRAPPER
           (global_user_conn_proxy, host, /*...*/);
  } else {
    /* or original mysqlnd method = do nothing, be transparent */
    ret = org_methods.connect(conn, host, user, passwd,
                    passwd_len, db, db_len, port,
                    socket, mysql_flags TSRMLS_CC);
  }
  return ret;
}
(my_mysqlnd_plugin.c)
```

The speaker says...

This is the hearth of linking userspace and C plugin world. Make sure you understand it. No further comments, you need to eat and learn it.

Repetition - how we get here:

- user runs MySQL query through any PHP MySQL API
- mysqlnd calls query method

Simple arguments

```
MYSQLND_METHOD(my_conn_class,connect)(/* ... */, const char *host,
/* ...*/) {
  /* ... */
  if (global_user_conn_proxy) {
    /* ... */
    zval* zv_host;
    MAKE_STD_ZVAL(zv_host);
    ZVAL_STRING(zv_host, host, 1);
    MY_ZEND_CALL_METHOD_WRAPPER
      (global_user_conn_proxy, zv_retval, zv_host /*, ...*/);
    zval_ptr_dtor(&zv_host);
    /* ... */
  }
  /* ... */
}
(my_mysqlnd_plugin.c)
```

The speaker says...

A standard task when calling when linking C and userspace are data type convertions. C variables need to be wrapped into PHP variables. **PHP variables are represented through zval structs on the C level.**

Passing all kinds of numeric and string C values to the userspace is quite easy. The ULB has all details.

Structs as arguments

```
MYSQLND_METHOD(my_conn_class, connect)(
  MYSQLND *conn, /* ...*/) {
  /* ... */
  if (global_user_conn_proxy) {
    /* ... */
    zval* zv_conn;
    ZEND_REGISTER_RESOURCE
      (zv_conn, (void *)conn, le_mysqlnd_plugin_conn);
    MY_ZEND_CALL_METHOD_WRAPPER
      (global_user_conn_proxy, zv_retval, zv_conn,/*, ...*/);
    zval_ptr_dtor(&zv_conn);
    /* ... */
  }
  /* ... */
}
```

The speaker says...

The first argument of many mysqlnd methods is a C "object". For example, the first argument of the connect() method is a pointer to MYSQLND. The struct MYSQLND represents a mysqlnd connection object.

The mysqlnd connection object pointer can be compared to a standard I/O file handle. **Like a standard I/O file handle a mysqlnd connection object shall be linked to the userspace using the PHP resource variable type.**

Are we done ?!

```
class proxy extends mysqlnd_plugin_connection {
  public function connect($conn, $host, ...) {
    /* "pre" hook */
    printf("Connecting to host = '%s'\n", $host);
    return parent::connect($conn);
  }

  public function query($conn, $query) {
    /* "post" hook */
    $ret = parent::query($conn, $query);
    printf("Query = '%s'\n", $query);
    return $ret;
  }
}
mysqlnd_plugin_set_conn_proxy(new proxy());
```

The speaker says...

We are almost done. One piece is missing, though. PHP users must be able to call the parent implementation of an overwritten method. To be able to call a parent implementation you need one, right? Let's hack it!

BTW, thanks to subclassing you may choose to refine only selected methods and you can choose to have "pre" or "post" hooks. It is all up to you.

Buildin class

```
PHP_METHOD("mysqlnd_plugin_connection", connect) {
  /* ... simplified! ... */
  zval* mysqlnd_rsrc;
  MYSQLND* conn;
  char* host; int host_len;
  if (zend_parse_parameters(ZEND_NUM_ARGS() TSRMLS_CC, "rs",
       &mysqlnd_rsrc, &host, &host_len) == FAILURE) {
    RETURN_NULL();
  }
  ZEND_FETCH_RESOURCE(conn, MYSQLND* conn, &mysqlnd_rsrc, -1,
    "Mysqlnd Connection", le_mysqlnd_plugin_conn);
  if (PASS == org_methods.connect(conn, host, /* simplified! */
TSRMLS_CC))
    RETVAL_TRUE;
  else
    RETVAL_FALSE;
}
(my_mysqlnd_plugin_classes.c)
```

The speaker says...

The code should bare no surprise. What you see is standard PHP C infrastructure code to call the original mysqlnd method. As usual, the ULB has all the details for you.

The End: implementors wanted!

Those who have stated their name at the beginning:

Would you mind hacking PECL/mysqlnd_plugin for me?

See you at the milk bar!

The speaker says...

From the IPC Spring 2010 (a PHP conference in Berlin): **to my own surprise a couple of companies showed interest in hacking mysqlnd plugins. Most of them want to open-source the development.**

If you have any question or would like to participate in any potential development, **feel free to contact me (ulf.wendel@sun.com)**. I'll try to get people in touch.

And, of course, we'll try to answer all technical questions.

Sugar!

```
class global_proxy extends mysqlnd_plugin_connection {
  public function query($conn, $query) {
    printf("%s(%s)\n", __METHOD__, $query);
    return parent::query($conn, $query);
  }
}
class specific_proxy extends mysqlnd_plugin_connection {
  public function query($conn, $query) {
    printf("%s(%s)\n", __METHOD__, $query);
    $query = "SELECT 'mysqlnd is cool'";
    return parent::query($conn, $query);
  }
}

mysqlnd_plugin_set_conn_proxy(new global_proxy());
$conn1 = new mysqli(...);
$conn2 = new PDO(...);

mysqlnd_plugin_set_conn_proxy(new specific_proxy(), $conn2);
(i_love_mysqlnd.php)
```

The speaker says...

Wait - every good movie has a trailer!

Mysqlnd allows a plugin, such as the fictive ext/mysqlnd_plugin, to associate arbitrary data with a connection. The data storage can be used to hold a pointer to a connection specific userspace proxy object.

Sugar, sugar!

```
class proxy extends mysqlnd_plugin_connection {
  public function connect($host /*... */) {
    $conn = @parent::connect($host /*... */);
    if (!$conn) {
        my_memcache_proxy_set("failed_host", $host);
        $failover_hosts = my_memcache_proxyget("failover_hosts");
        foreach ($failover_hosts as $host) {
            $conn = @parent::connect($host /* ... */);
            if ($conn) {
                my_memcache_proxy_set("working_host", $host);
                break;
            } else {
                my_memcache_proxy_set("failed_host", $host);
            }
        }
    }
    return $conn;
  }
}
(client_failover_with_config.php)
```

The speaker says...

One of the disadvantages of a mysqlnd plugin based client proxy approach is the non-persisent memory of the mysqlnd plugin. The mysqlnd plugin cannot recall decisions made earlier. One plugin cannot share information with another one.

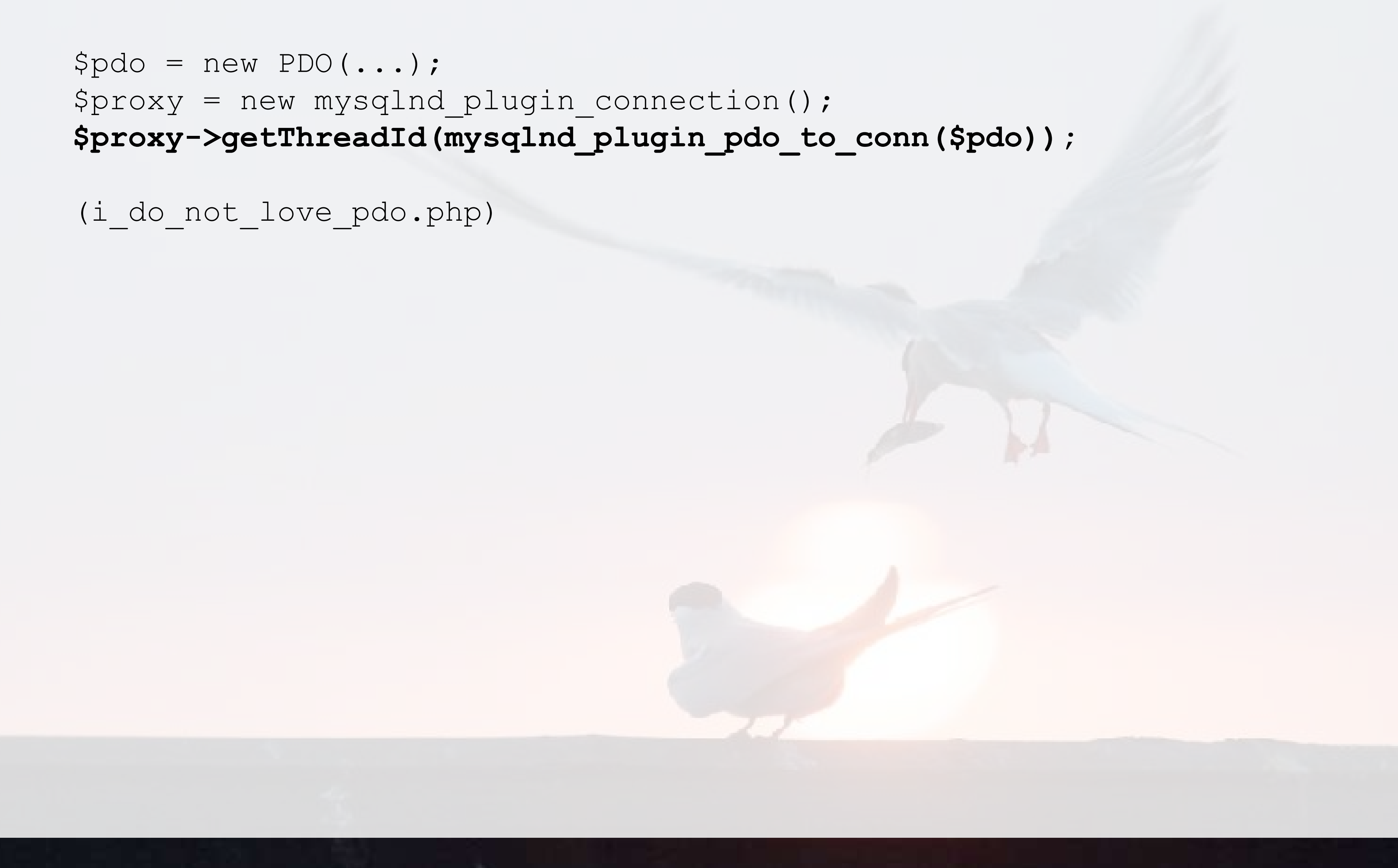
But you may ask your Memcache deamon to help out!

Yeah, a classical hack to maintain a state...

Sugar, sugar Baby!

```
$pdo = new PDO(...);
$proxy = new mysqlnd_plugin_connection();
$proxy->getThreadId(mysqlnd_plugin_pdo_to_conn($pdo));

(i_do_not_love_pdo.php)
```



The speaker says...

In our discussion we have looked at the userspace proxy and the default proxy class from of our fictive ext/mysqlnd_plugin as a passive component which gets called through mysqlnd.

Though, there is no reason why we would not be allowed to call proxy methods directly as long as we can provide the necessary arguments. For example, we can use the proxy as shown above to obtain the thread id of a PDO MySQL connection. This is something that is not possible through the standard PDO API.

THE END

Credits: Andrey Hristov, Contact: ulf.wendel@sun.com