InnoSQL - NoSQL in MySQL

Starring as Kaspele: Ulf Wendel

Bed time story: Káschberl!

Foreword and disclaimer

This presentation was given as a „Night Session" at the PHP Summit 2013 in Munich, capital of the Bavarian Kingdom. After a day packed with trainings the audience deserves a rest. The bed time story is a Kasperltheater, a traditional puppet theater, which has its roots in the 17th century. According to Wikipedia, the puppet character actually named Kasper first appeared in Munich in 1858 in a marionette play...

Todays fairy tale is about NoSQL in MySQL. Remember, all databases are named after the children of their inventors. The children are: My, Max and Maria. There is no son called No. Thus, there cannot be NoSQL in MySQL - ever!

Foreword and disclaimer

Disclaimer: If in the portrayal of internal processes you think you identify similarities with persons still living and working or the procedures of the "PhoenixSQL", such similarities are neither intended nor coincidental but unavoidable....

Like so many fairy tales, there may be a true story behind.
Like so many bed time stories, this one could have a moral.
Like so many presentations, this works best live.

For example, Slideshare cannot play the puppet scenes in front of you... or, give you a vodka shot.

InnoSQL - NoSQL in MySQL

Starring as Kaspele: Ulf Wendel

The speaker says…

„Lady Sakila gets a key value store“

Starring:

Sakila, the desperate beauty

Phoenix, the creative cock

Innobear, the allmighty

A snoopy seagull

Kasperle

Free bonus:

„Beyond the fairy tale“ by Kasperle

Season style:
Hot NoSQL outfits

BREAKING NEWS:
NoSQL explained

We love NoSQL

Drizzle?
SQL free cloud solutions that don't leave you in the rain!

*** International jet set trend: polyglot persistence ***

First act: the wretch

*** Couchbase and CouchDB split – exclusive photo ***

*** Framed workers: Having been grouped by aggregates ***

The speaker says...

Sakila: (crying, silently)

Kasperl: Sakila, my dear, what happened?

Sakila: Oh, Kasperl! I am so desperate. I made superior database technology available and affordable to anybody. MySQL became the M in the industrial standard called LAMP.

Kasperl: (interrupts her) My one and only - you are the #1!

Sakila: I read the IT yellow press at the beauty parlour. NoSQL celebrities all over. Not a single mention of myself. Kasperl, my glorious times are over (crying, loud)....

Kasperl: Sweetheart, no! Together with our mighty friends, we will find a new outfit for you. The front page is yours.

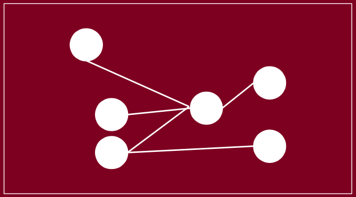
Swipe, tap, click, and zoom

1234 - I am super FAST
4567 - LIGTHNING fast
7890 - [1][2][3][4][5]
abcd - [1[a,b,c]],[2[d,e,f]]
_de_- 1,281,2828,2,173,8

Key Value Store

{"documents":"rock"}
{"mysql":"not"}
{"documents":"rock"}
{"mysql":"not"}
{"documents":"rock"}

Document Database



Graph Database

0101010010110110
1010101101001011
1010100101010110
1010101101010111
1010101101111101

Big Data/Column-oriented

The speaker says...

Kasperl leaves the stage to learn who those NoSQL celebrities are.

There are many, that's for sure. Celebrity central at nosql-database.org lists more than 150+ newcomers.

There are many fans, that' s for sure. Market researchers predict 25% less MySQL fans within five years*. Some 30% looked into or already use NoSQL**.

There are four major groups of celebrities, that's what the records say. Lines between them are not always clear.

* http://de.slideshare.net/mattaslett/mysql-vs-nosql-and-newsql-survey-results-13073043

** MySQL users of the 451 Research sample

Swipe, tap, click, and zoom

Memcache, Redis, Riak, LevelDB, BerkeleyDB, Dynamo, ...

Key Value Store

MongoDB, CouchDB, RethinkDB, ...

Document Database

Neo4j, FlockDB...

Graph Database

BigTable, Hadoop, Cassandra...

Big Data

The speaker says...

Fan count goes somewhat down from top left to bottom right.

For many years already, Sakila showed up happily with her friend Memcache. MySQL was the leading one-fits-all solution co-existing with Memcache as a cache front end in a traditional architecture.

Then came a change (sometimes referred to as): polyglot persistence. The market changed. After decades of RDBMS dominance new, specialized products appeared in public. Often driven by Web 2.0 challenges. Some new ideas, many old ideas are incorporated into them.

A new era of databases

Explore the benefits

- Scaleable
- Elastic
- Highly Available
- Easy To Use

The speaker says...

Not Only SQL databases aim to be scalable. From one node to one tousand nodes in a bit. And, back depending on both query load and data size! Sharding built-in! Chewing gums of the cloud area!

Master down? No problem. Some don't do lame primary copy. Paxos and others ensure the database cluster survives the failure of nodes – including primaries, if any.

Hot on conferences: JavaScript, HTTP, JSON you name it - ingredients of todays web applications. So easy with hierarchical structures, weak types and no schema changes!

Zoom! Key Value Store

1234 - I am super FAST
4567 - LIGTHNING fast
7890 - [1][2][3][4][5]
abcd - [1[a,b,c]],[2[d,e,f]]
_de_- 1,281,2828,2,173,8

Key Value Store

High Performance

Limited Search and Types

Scaleable

Limited Persistence

The speaker says…

A Key Value Store strikes for its simple data model which is that of an **associative array/hash**. The data model is poor at ad-hoc queries: loose the key and you lock your data in the treasure. But, it is fast. A **need for speed** has led to many in-memory solutions in this class. A perfect model for use as a cache. If used as a cache, persistence is often secondary. Generally speaking the data model is perfect for partitioning/sharding. There are no operations covering multiple values, thus values can be distributed on multiple nodes to scale the system.

Most operations are basic (CRUD). Redis stands out with complex data types and correspondig commands.

Using MySQL as a NoSQL - A story for exceeding 750,000 qps on a commodity server

[...] MySQL/InnoDB 5.1 [...] When you run simple multi-threaded "memcached get" benchmarks, you can probably execute 400,000+ get operations per second, even though memcached clients are located on remote servers. [...] network i/o bound [...] 260,000 qps on MySQL via HandlerSocket, 220,000 qps on memcached [...]

Second act: Phoenix, the super, super, super star

The speaker says...

Meanwhile in Farfaraway. A former colleguage of Sakila becomes a super, super, super star over night.

Voice from the off: (applaudes) We have a winner!

Phoenix: I am the first to squeeze out 750,000 queries per second from stock MySQL 5.1 using commodity hardware!

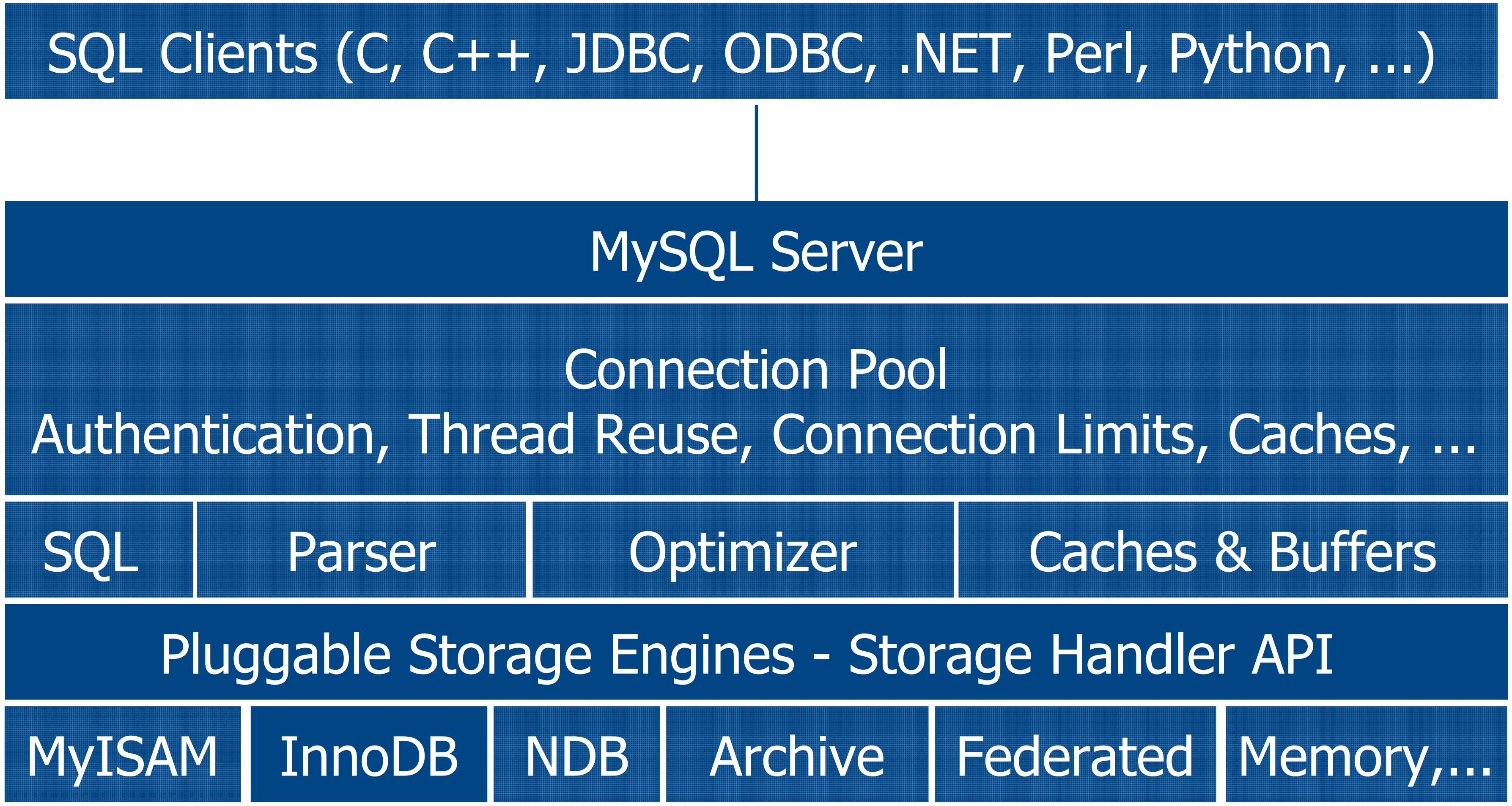
Audience 1: Finally! For simple queries SQL parsing took some 50% of the total query execution time.

Audience 2: Pfft, MySQL Cluster reached that in 2004...

Audience 3: Oh, those youngsters. We have HANDLER since ISAM times!

... what, why, how ?!

Flexible storage engine API


SQL Clients (C, C++, JDBC, ODBC, .NET, Perl, Python, ...)
MySQL Server
Connection Pool
Authentication, Thread Reuse, Connection Limits, Caches, ...
SQL
Parser
Optimizer
Caches & Buffers
Pluggable Storage Engines - Storage Handler API
MyISAM
InnoDB
NDB
Archive
Federated
Memory,...

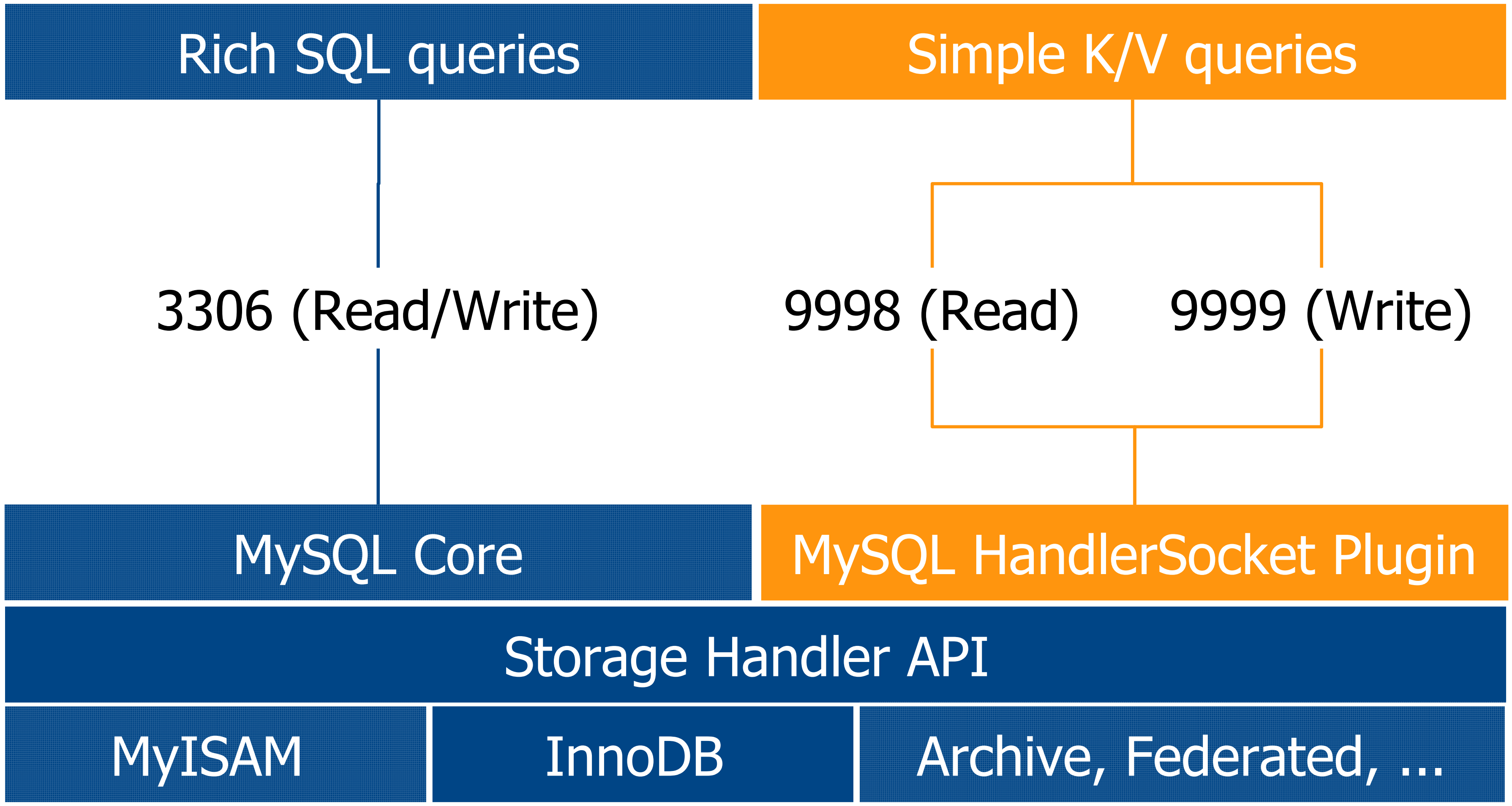
The speaker says...

The creative and innovative phoenix took a deep look inside MySQL. He was wondering where MySQL spends most time for the most common type of queries his company issued: simple primary key based look-ups and limited range scans.

SQL parsing and frequent thread locking for synchronization tasks turned out to be expensive whereas in-memory and CPU efficiency of the InnoDB storage engine proved to be competitive. Phoenix recalled previous work: „Mycached: memcached protocol support for MySQL […] The QPS (queries per second) of mycached is roughly 2x compared to using SQL [...]"

Let the hacking begin!

HandlerSocket daemon plugin



The speaker says...

A MySQL daemon plugin is a „do-whatever-you-want“ extension to MySQL. A plugin is a dynamic library running as part of the server process.

HandlerSocket starts a multi-threaded network server. The network server listens on ports 9998 and 9999 by default. One port handles read requests, the other port handles write requests. Data stored in MySQL is accessed through the internal low-level storage engine API of MySQL. A lightweight and simple one-line based request response style protocol is used by HandlerSocket. Attention: new protocol means new client libraries required!

Using HandlerAPI does not mean, it works with MyISAM. HS + MyISAM is buggy: ACK for failed insert, wrong defaults, ...

Setting things up: compiling...

```
> git clone https://github.com/DeNA/HandlerSocket-Plugin-for-MySQL.git
> cd HandlerSocket-Plugin-for-MySQL/
> ./autogen.sh
> ./configure --with-mysql-source=/data/nixnutz/ftp/mysql-5.5.30/ --with-mysql-
bindir=/data/nixnutz/ftp/mysql-5.5.30/install/bin --with-mysql-plugindir=/data/nixnutz/ftp/mysql-
5.5.30/install/lib/plugin/
> make clean && make -j3
> sudo make install
> nano /data/nixnutz/ftp/mysql-5.5.40/install/my.cnf
> mysql -uroot -S/tmp/mysql5530.sock test
```

```
mysql> INSTALL PLUGIN handlersocket SONAME 'handlersocket.so';
mysql> SHOW PLUGINS;
mysql> SHOW PROCESSLIST;
```

```
> wget http://php-handlersocket.googlecode.com/files/php-handlersocket-0.3.1.tar.gz
> tar xvzf php-handlersocket-0.3.1.tar.gz
> cd handlersocket
> phpize && ./configure && make clean && make -j3
> sudo make install
> nano /usr/local/lib64/php.ini
```

The speaker says...

HandlerSocket is distributed in source „as-is" by the original authors. It has originally been developed to work with MySQL 5.1. The latest version can be compiled against MySQL 5.5. MySQL 5.6 is currently not supported but may so in the future. MySQL forks may distribute patched versions, however, I went for the original repository.

PHP users can choose from two PHP based client libraries implementing the text-based protocol and one C/C++ PHP based extension. For all tests the PHP extension has been used. The handlersocket PHP extension is using the C++ client library shipped with HandlerSocket.

HandlerSocket: PK lookup

```
try {
  $hs_r = new HandlerSocket("127.0.0.1", 9998);

  if (!($hs_r->openIndex(
                1, "mysql", "plugin", 'PRIMARY', 'name,dl')))
    throw new Exception($hs_r->getError());

  if (false === ($v = $hs_r->executeSingle(
                               1, "=", array("handlersocket"))))
   throw new Exception($hs_r->getError());

  var_dump($v);
} catch (Exception $e) {
  var_dump($e);
}
```

The speaker says...

Generally speaking the HandlerSocket API mimics some principles of the MySQL internal storage handler API. The internal API was not designed to be exposed to a web user. Means, HandlerSocket API may not look very appealing to you.

To read or write data one has to open an index first. No index, no fun. Then, the index identifier is used to issue a query. The slide shows the counterpart of: `SELECT name, dl FROM mysql.plugin WHERE name = 'handlersocket'`. The query is run in „autocommit" mode. Transaction isolation level should be dirty read. Data is returned as string to PHP. Strings are returned „as-is" - no special charset handling.

HandlerSocket: INSERT

```
try {
  $hs_w = new HandlerSocket("127.0.0.1", 9999);

  if (!($hs_w->openIndex(
              1, "test", "ulf", 'PRIMARY', 'col_value')))
    throw new Exception($hs_w->getError());

  if (false === ($pk = $hs_w->executeInsert(
                             1, array("value"))))
   throw new Exception($hs_w->getError());

  var_dump($pk);
} catch (Exception $e) {
  var_dump($e);
}
```

The speaker says...

The PHP function calls to insert a record are very similar to those for fetching data: open index, run command. The executeInsert() function returns the primary key value upon successful insertion using a PRIMARY key index. The table used in this example is: `CREATE TABLE test.ulf(col_id INT AUTO_INCREMENT PRIMARY KEY, col_value VARCHAR(255)) ENGINE=InnoDB .` The INSERT statement run through the HandlerSocket API is: `INSERT INTO test.ulf(col_value) VALUES ('value') .`

The write request could also have been sent on the port dedicated for read requests. The split into read and write port exists only to allow distinct, dedicated thread pools.

Complex statements on indicies

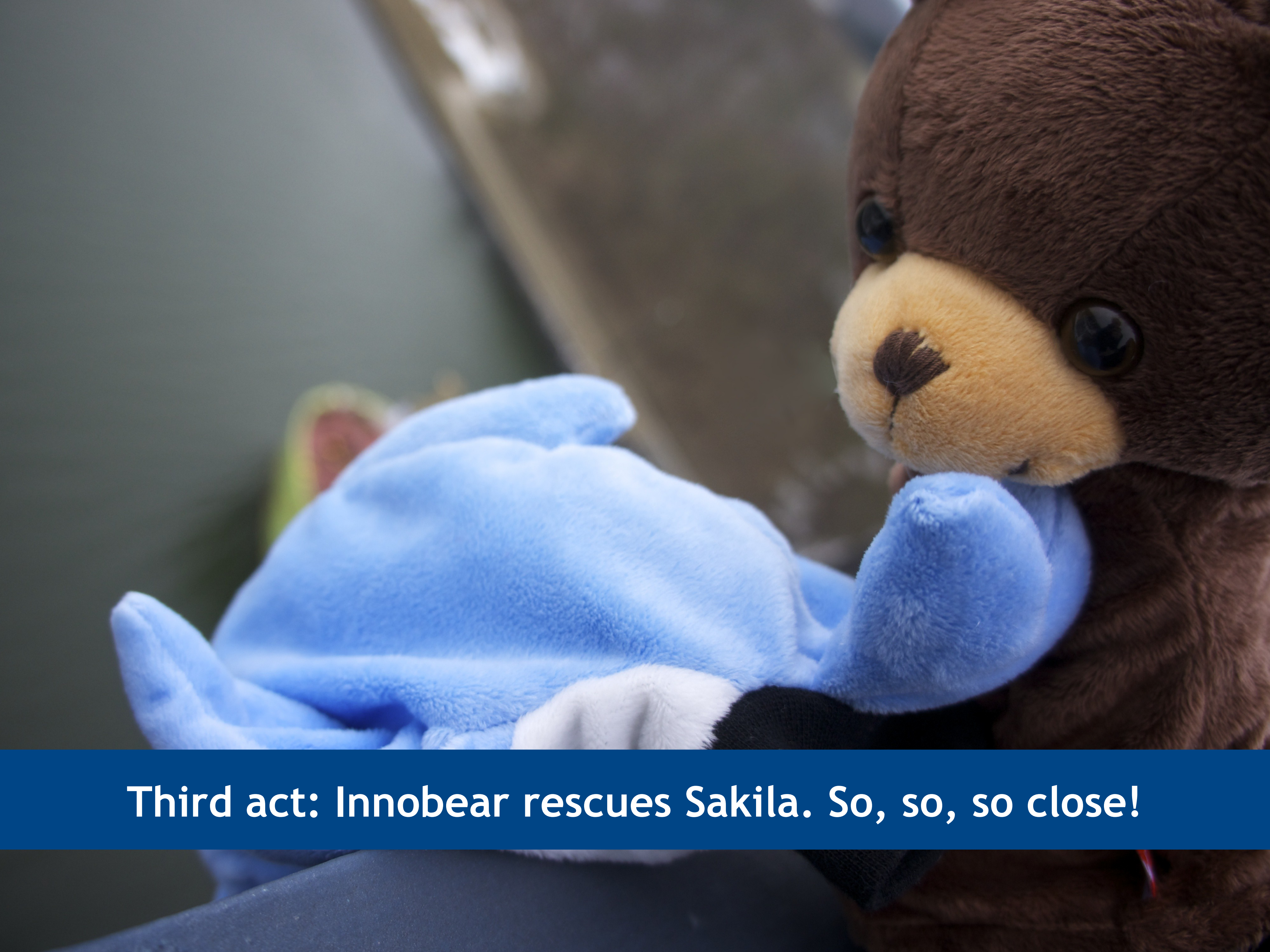
```
where_cond: <idx_col> [=,<,<=,>, >=] <value>
SELECT <idx_col>[,idx_col, ...] FROM table WHERE where_cond
SELECT ... FROM table WHERE ... LIMIT offset[,row_count]
SELECT ... FROM table WHERE idx_col IN(<value>[, value, ...]
SELECT ... FROM table WHERE ... [AND …]

$ret = $hs->executeSingle(1, '>=', array('K1'), 10, 0, null, null,
array(array('F', '>', 0, 'F1'), array('F', '<', 1, 'F10')));
/* SELECT k, v FROM table WHERE k >= 'K1' AND f1 > 'F1' AND f2 <=
'F20' LIMIT 10 */

UPDATE table SET ...[, ...]
UPDATE table WHERE ... SET ...
Options: return previous value, return affected rows, increment,
decrement
```

The speaker says...

The HandlerSocket user API covers more than the most basic use case of a primary key lookup, it offers more than a `GET key` operation. For example, `WHERE` conditions can be composed of multiple `AND`ed condition. The `IN` operator is supported. `LIMIT` can be used.

It is, however, not possible to sort results, or to write a query that spawns multiple tables. For rich querying, SQL has to be used.

Some tweaks for `UPDATE` operations exist.


Third act: Innobear rescues Sakila. So, so, so close!

The speaker says...

The NoSQL news made it to Kasper.

Kasper: Innobear, news: MySQL is faster than Memcache! Let me show you what Phoenix developed.

Innobear: Hmm... Great idea, probably we can do better...

Kasper: We must gather the team. You go to find Sakila, I have a rough idea where to find Seagull.

Innobear: (loud) Sakila, where are you! We got news!

Sakila: (whining from a distant, ready to jump off a bridge)

Innobear: (comes close to her) Please, don't jump!

Sakila: Goodbye fast friend. (jumps)

Innobear: (grabs her tail fin the very last second)

Third act: Seagull is a valuable team member

The speaker says…

Kasper: Seagull!

Seagull: Oh, oh, oh, … I promise: I'll do everything to raise shareholder value. The company has my full attention. Also, remember what our all first work contracts said about drinking. We must not begin working when drunk. It didn't say anythink about drinking during...

Kasper: Stop it! Shareholder value is more important!

Kasper and Seagull return to meet Innobear and Sakila. Together they decide to develop a secret plan!

Sakila takes a rest at a beauty farm.

Third act: at labs.mysql.com

The speaker says...

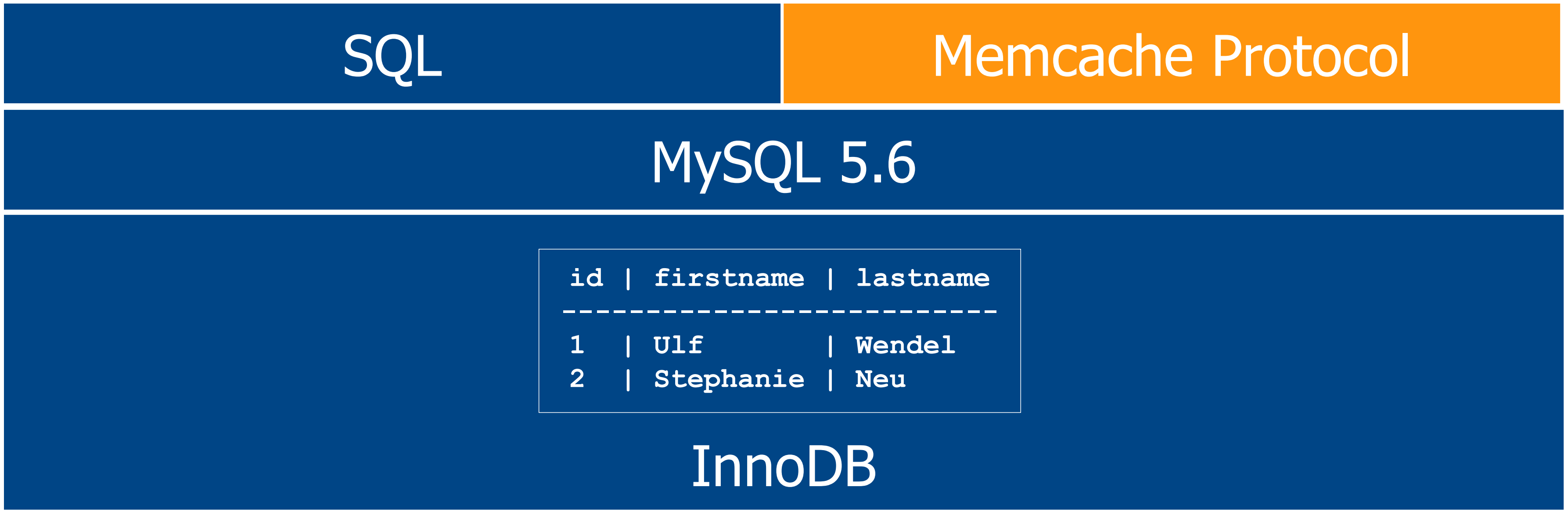
Innobear, Kasper and Seagull go down to the cellar to begin their work. They shut all doors and windows to focus on their secret task. For reasons of fire security and data protection, we cannot go into details.

Fairywikileaks reports that bookface.com technicians commented to Phoenixs' and other blogs that an embedded InnoDB with Memcached interface would be of great value. Isotipp from book-kings-hotels.com publishes a blog post in which his colleguage reports up to 10x higher query rates using HandlerSocket, confirming its faster than Memcached.

5.6: InnoDB Memcache Plugin

RDBMS and Key Value Store combined

- Benefits of a mature RDBMS
- High performance key lookup plus rich SQL ad-hoch querying
- No need to synchronize between cache and RDBMS



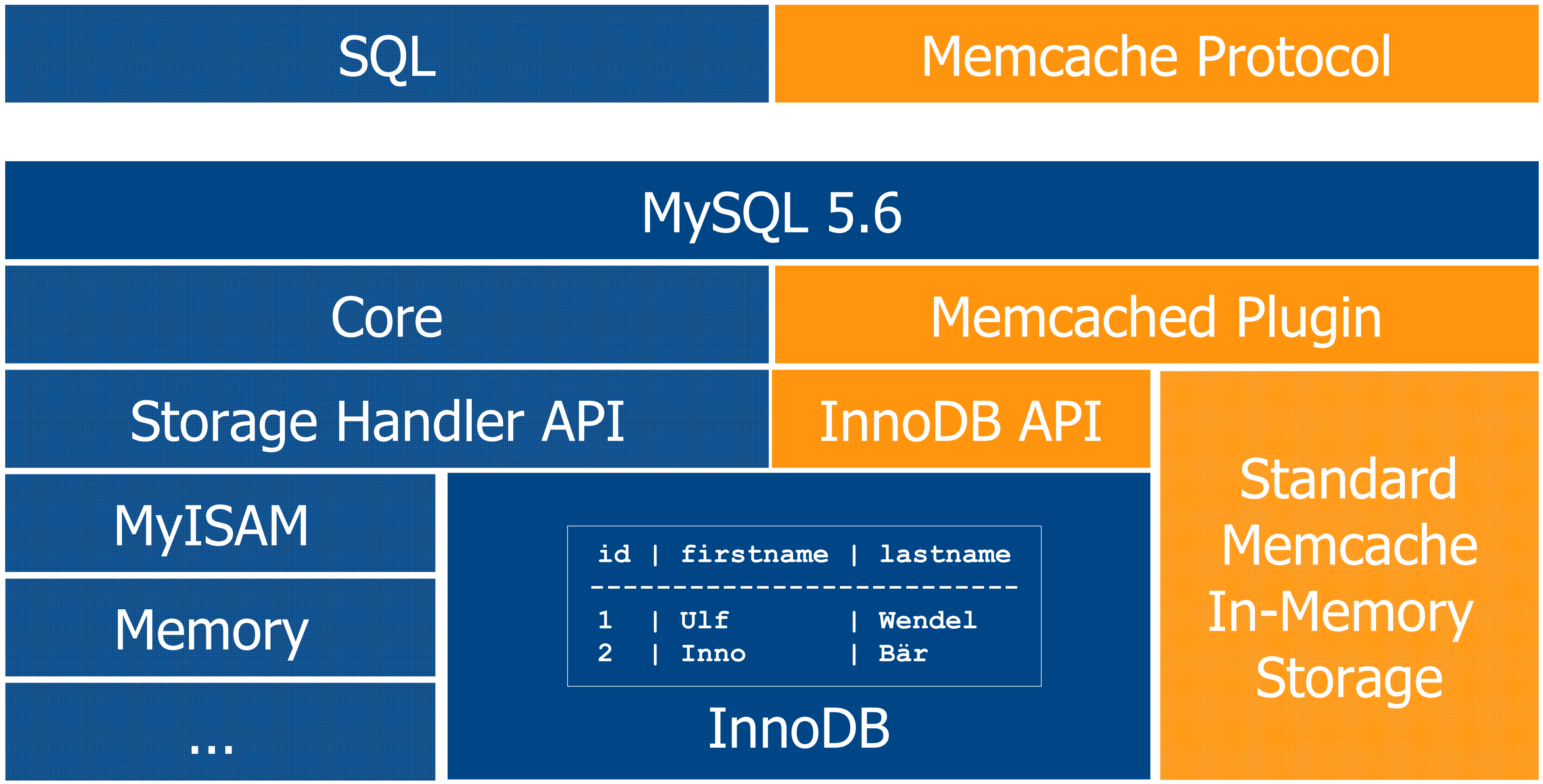
```
id | firstname | lastname
--------------------------
1  | Ulf       | Wendel
2  | Stephanie | Neu
```

The speaker says...

Eventually, MySQL 5.6 gets released. Innobear and Seagull start a presentation... MySQL 5.6 has both a SQL and a NoSQL interface. The proven, lightweight Memcache protocol gets used. Many MySQL users set on Memcached since years. Memcached language bindings are available for all major programming languages. Familarity and stability of the APIs are a given.

The full potential of the stable and CPU-efficient B-tree based storage engine InnoDB, which features automatic, adaptive hashing ever since, gets unveiled. In-memory performance is great because it has to be. Estimated 20% use 64 – 256 GB RAM with MySQL.

Zoom! InnoDB Memcache



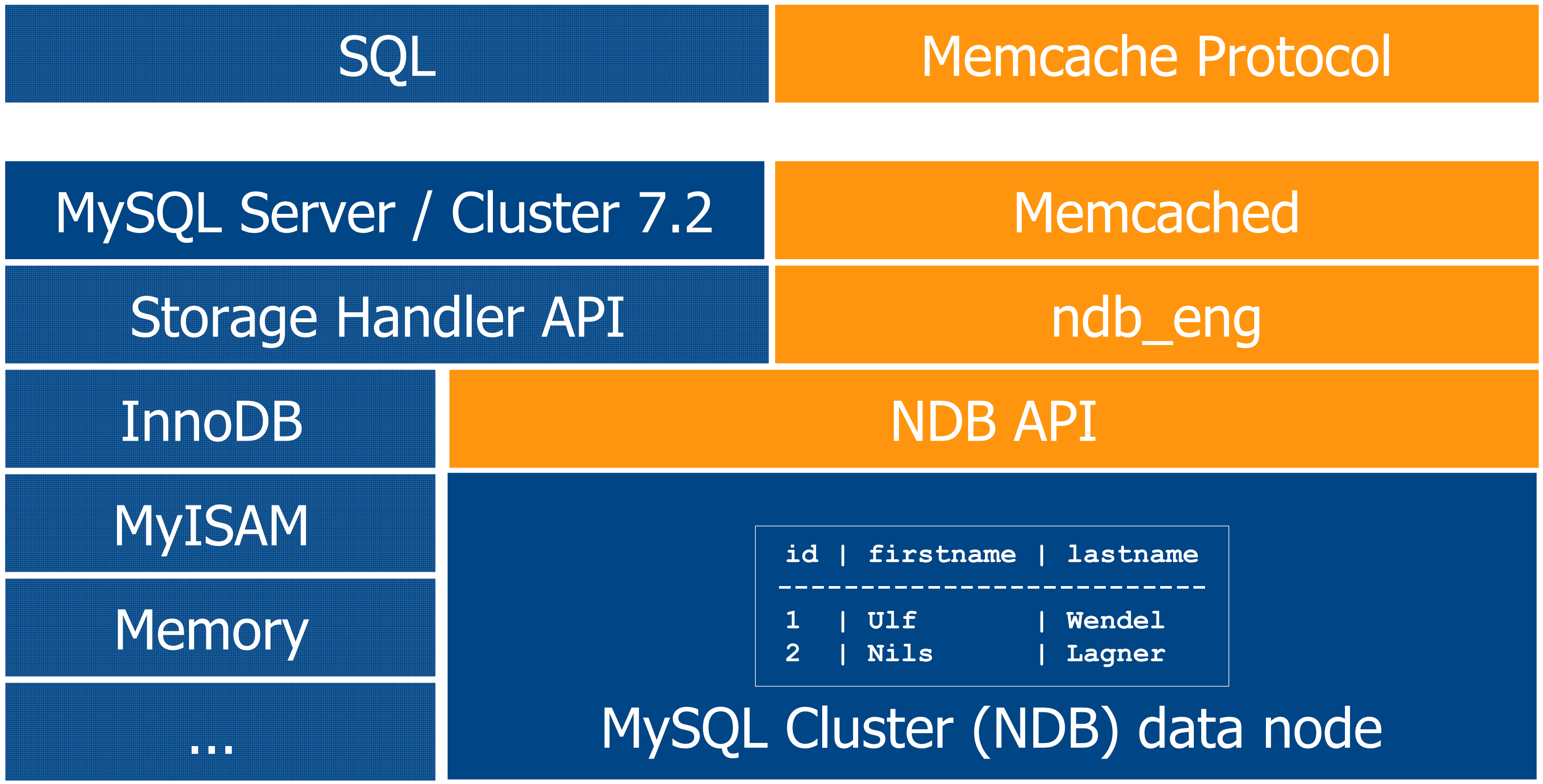
```
id | firstname | lastname
----------------------------
1  | Ulf       | Wendel
2  | Inno      | Bär
```

The speaker says...

The InnoDB Memcached Plugin integrates a Memcached network server into MySQL. The integrated Memcached server can either use main memory or InnoDB for storage. Similar to MySQL, Memcached allows multiple storage backends. The Memcached server integrated into the plugin can use InnoDB as a storage backend.

InnoDB is accessed through the InnoDB API (basically the former Embedded InnoDB API). The InnoDB API is lowest-level API to communicate with InnoDB, thus it is the fasted way. This promises even higher performance than the Storage Handler API. This, however, means no loss of features: Replication, transactions etc. are supported.

Zoom! MySQL Cluster Memcache



```
id | firstname | lastname
---------------------------
1  | Ulf       | Wendel
2  | Nils      | Lagner
```

The speaker says...

BTW, MySQL Cluster 7.2 now supports Memcached as well. MySQL Cluster/NDB nodes can serve as a storage backend for Memcached. Memcached is using NDB API, the native C++ API of MySQL Cluster, to fetch and store data.

The process model, however, is different from the MySQL InnoDB Memcached Plugin. Memcached is not integrated into the MySQL Server. Memcached is run „standalone" as a seperate network server.

You can choose whether to run the Memcached, the MySQL Cluster data nodes and the application on one machine (low latency) or on different ones (fail safety).

… back to InnoDB.

InnoDB Memcache Setup

```
mysql> source MYSQL_HOME/share/innodb_memcached_config.sql

mysql> INSTALL PLUGIN daemon_memcached SONAME 'libmemcached.so';
mysql> SHOW PLUGINS;
mysql> SHOW DATABASES;
mysql> USE innodb_memcache;
mysql> SHOW TABLES;
+---------------------------+
| Tables_in_innodb_memcache |
+---------------------------+
| cache_policies            |
| config_options            |
| containers                |
+---------------------------+
3 rows in set (0,00 sec)
```

The speaker says...

Before the InnoDB Memcached plugin can be installed using `INSTALL PLUGIN` statement, one has to setup the plugins configuration database `innodb_memcached` and its tables by executing a SQL script.

HandlerSocket puts no restrictions on which databases and tables can be manipulated through the network ports it openes whereas the Memcached plugin allows finer access control. Speaking of access control: HandlerSockets only way to secure access is requiring a clear-text password to be sent upon connect. Memcached supports Simple Authentication and Security Layer (SASL) based password protection, means: various password methods supported.

Container basics

```
mysql> CREATE TABLE test.memc_kv (
  col_key VARCHAR(32) NOT NULL DEFAULT '',
  col_value VARCHAR(250) DEFAULT NULL,
  memc_internal_flags INT(11) DEFAULT NULL,
  memc_internal_cas BIGINT(20) UNSIGNED DEFAULT NULL,
  memc_internal_expire_time INT(11) DEFAULT NULL,
  PRIMARY KEY (col_key)
) ENGINE=InnoDB;
mysql> INSERT INTO innodb_memcache.containers(name, db_schema,
db_table, key_columns, value_columns, flags, cas_column,
expire_time_column, unique_idx_name_on_key) VALUES ('container_a',
'test', 'memc_kv', 'col_key', 'col_value', 'memc_internal_flags',
'memc_internal_cas', 'memc_internal_expire_time', 'PRIMARY');
mysql> UNINSTALL PLUGIN daemon_memcached;
mysql> INSTALL PLUGIN daemon_memcached SONAME 'libmemcached.so';
```

The speaker says...

To make a table accessible through the Memcached interface, it has to be listed in the `innodb_memcached.containers` configuration table.

A most basic table must have a unique index to be used as key and a column to hold a value. Both key and value columns must be strings (CHAR, VARCHAR, TEXT).

The table should have three columns used by Memcached. A column to store flags, one to store the cas (compare-and-swap) and one for the expire value of Memcached. If those columns are omitted, Memcached protocol features and/or the plugin may not work properly.

InnoDB Memcache: key lookup

```
try {
  $memc = new Memcached();

  if (!$memc->addServer("127.0.0.1", 11211))
    throw new Exception($memc->getResultCode());

  if (false == $memc->set("A", "Value"))
    throw new Exception($memc->getResultCode());

  if (false === ($val = $memc->get("A")))
    throw new Exception($memc->getResultCode());
  var_dump($val);
} catch (Exception $e) {
  var_dump($e);
}
```

The speaker says...

No surprises for the most basic application: SQL table mapped to Memcache, standard PHP Memcache API used to access the SQL table.

Please note, whenever you change the container configuration you have reload the plugin to make it aware of configuration changes. Also, for your initial tests you may want to have no more than one container configured. More on this later.

Container: column mapping

```
mysql> CREATE TABLE test.memc_test2 (
  col_key varchar(32) NOT NULL DEFAULT '',
  col_val_a varchar(250) DEFAULT NULL,
  col_val_b varchar(250) DEFAULT NULL,
  memc_internal_flags int(11) DEFAULT NULL,
  memc_internal_cas bigint(20) unsigned DEFAULT NULL,
  memc_internal_expire_time int(11) DEFAULT NULL,
  PRIMARY KEY (col_key)
) ENGINE=InnoDB;
mysql> INSERT INTO innodb_memcache.containers(name, db_schema,
db_table, key_columns, value_columns, flags, cas_column,
expire_time_column, unique_idx_name_on_key) VALUES ('container_b',
'test', 'memc_test2', 'col_key', 'col_val_a,col_val_b',
'memc_internal_flags', 'memc_internal_cas',
'memc_internal_expire_time', 'PRIMARY');
```

The speaker says...

A Memcached value stored in an InnoDB table through the Memcached interface can spawn multiple columns. To map multiple SQL columns to a value, list the columns of the SQL table to be mapped in the container configuration tables `value_columns` column. Create a comma seperated list of the columns to be mapped and store it in `innodb_memcached.containers.value_columns`.

Remember, that there are constraints. Only string columns are supported. There are size limitations: key and value together are limited to 1MB. Please, see the manual for further limits primarily related to InnoDB indexes.

Multiple columns value

```
[...]
if (false == $memc->set("A", "ValueA|ValueB"))
  throw new Exception($memc->getResultCode());

if ($res = $mysqli->query(
            "SELECT * FROM memc_test WHERE col_key = 'A'")){
  $row = $res->fetch_assoc();
  printf("Key '%s': '%s' - '%s'",
    $row['col_key'], $row['col_val_a'], $row['col_val_b']);
} else {
  throw new Exception($mysqli->error);
}
[...]
```

```
Key 'A': 'ValueA' - 'ValueB'
```

The speaker says...

To store a value into multiple SQL columns, it is first split by a seperator. Then, the parts are mapped to the columns configured and stored. The reverse logic is applied when fetching a value through the Memcache interface. The mapped columns values are concatenated by the seperator.

The default seperator is `|`. The seperator is configurable:

```
REPLACE INTO
innodb_memcache.config_options(name, value)
VALUES ('separator', '@');
```

Restart the plugin to make the change take effect.

CAUTION, sad but true: there is no escape sign!

Containers (plur.): @@name.key

```
$memc->set("A", "Default, first or only container");
var_dump($memc->get("A"));
var_dump($mysqli->query("SELECT * FROM memc_default WHERE col_key =
'A'")->fetch_assoc()['col_val']);

$memc->set("@@container_b.A", "Container|named 'container_b'");
var_dump($memc->get("@@container_b.A"));

var_dump($mysqli->query("SELECT * FROM memc_test2 WHERE col_key =
'A'")->fetch_assoc()['col_val_a']);
```

```
string(32) "Default, first or only container"
string(32) "Default, first or only container"
string(29) "Container|named 'container_b'"
string(9) "Container"
```

The speaker says...

You can configure as many containers as you want. To access individual containers (SQL tables) through the Memcache interface, prefix the Memcache key with the containers name: name.key. The dot in name.key is configurable, the @@ part is not. Update the table `innodb_memcache.config_options` to change the seperator.

If multiple containers are configured and no prefix is used with the key, the value is stored in the container with the special name `default` . Caution: if you have multiple containers, none of which is named `default` and your key does not look like name.key, the value is not rejected but it goes into the first (alphabetic order) container set.

… natural and fresh from the country!
BTW, if you can read this, you don't need glasses!

Fourth act: subversive activities, or valuable team member?

The speaker says…

Kasper is very proud of Innobear! But, he seems to be the only one on stage. Seagull was supposed to help Innobear!

Kasper: (from the off) Seagull!

Seagull: … *hit, yeah....

Kasper: (from the off) Seagull, your behaviour cannot be tolerated. I expect you to behave. Go back on stage to help Innobear. But, I warn you, take the presentation serious. This is a serious conference...

Seagull: (jumps on stage) Yes, Sir! I'll take over...

Innobear: (grumpy, whispering) Finally, I need to go to the restrooms.

GET @@name = USE name

```
$memc->set("@@container_b.A", "Container|named 'container_b'");
var_dump($memc->get("@@container_b"));
var_dump($memc->get("A"));
var_dump($memc->delete("A"));
var_dump($memc->get("A"));
var_dump($memc->get("@@default"));
$memc->set("A", "Will it go into default?");
var_dump($memc->get("A"));
```

```
string(17) "test/memc_default"
string(29) "Container|named 'container_b'"
bool(true)
bool(false)
string(17) "test/memc_default"
string(24) "Will it go into default?"
```

The speaker says...

Seagull seems to take his boss Kasper serious and gives his best...

Issuing a `GET` request for `@@name` switches a sessions default container. This „USE“ statement changes the default container used for all subsequent `GET`, `SET` and `ADD` commands but for no other command. For example, `INCR` and `DELETE`, should not support the syntax... says the manual. Hmm, however, example shows the opposide... whatever...

Kasper: (from the off) Seagull! We want positive news!

Transaction and lock control

- When to commit
  - `daemon_memcached_r_batch_size` (default: 1)
  - `daemon_memcached_w_batch_size` (default: 1)
  - `innodb_api_bk_commit_internal` (default: 5)
- Transaction Isolation level
  - `innodb_api_trx_level` (READ UNCOMMITTED)
- Assorted lock related
  - `innodb_api_enable_mdl` (OFF)
  - `innodb_api_disable_row_lock` (no docs)

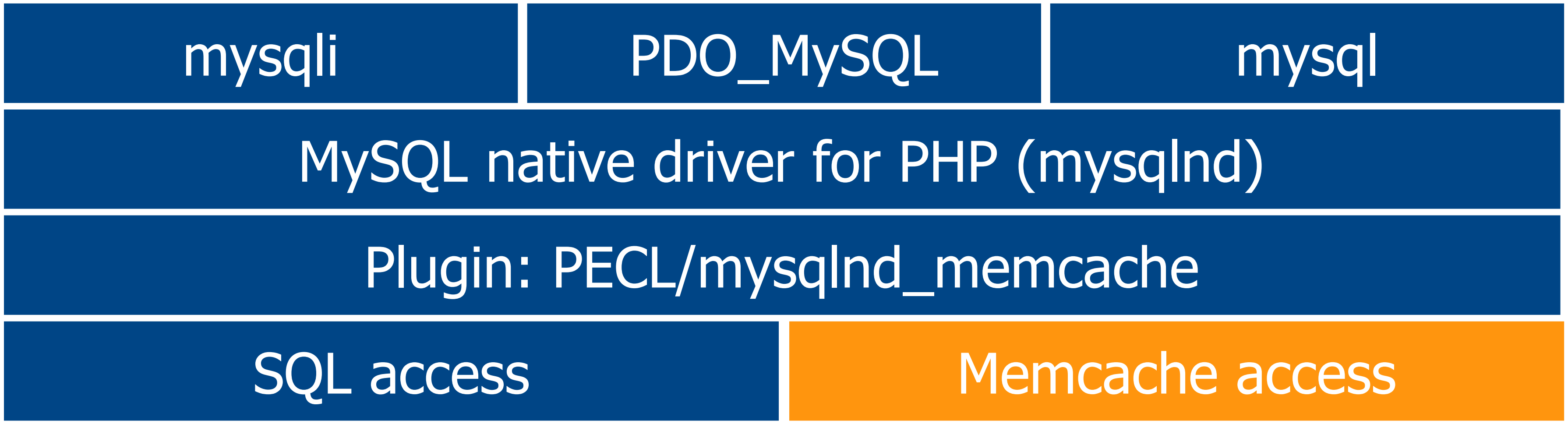
The speaker says...

Transaction control with the InnoDB Memcache Plugin is finer than with HandlerSocket. You can set how often read and write operations will commit and what transaction isolation level is used. Default is autocommit-style with READ UNCOMMITED. If data safety is not an issue, you may lower the commit rates to improve performance. `innodb_api_bk_commit_internal` is for idle client connections, not for active ones.

Enabling `innodb_api_enable_mdl` locks the table used by the InnoDB memcached plugin, so that it cannot be dropped or altered by DDL through the SQL interface.

Transparent fast key access

```
$mysqli = new mysqli("localhost", "usr", "pass", "test");
$memcache = new memcached();
$memcache->addServer("localhost", 11211);
mysqlnd_memcache_set($mysqli, $memcache);
$res1 = $mysqli->query("SELECT firstname FROM test WHERE id = 1");
$res2 = $mysqli->query("SELECT * FROM test);
```


mysqli
PDO_MySQL
mysql
MySQL native driver for PHP (mysqlnd)
Plugin: PECL/mysqlnd_memcache
SQL access
Memcache access

The speaker says...

PECL/mysqlnd_memcache is another free and open source plugin for the PHP mysqlnd library. Mysqlnd is the compile time default C library used for all PHP MySQL APIs (mysqli, PDO_MySQL and mysql).

Like other plugins it adds new features to all the APIs. Based on a configurable regular expression the plugin turns a SQL access into a Memcache access. Due to the lightweight protocol and direct access the Memcache access to MySQL is faster. No matter what protocol used by the library, the user gets a standard result set in return. Simple to use. However, note that no meta data is available if a key access has been performed.

Well... in theory... no escape character for multi-columns...

Benchmarking

- In general...
  - Big variation in methods, setup (cores!), and results
  - Risk of comparing apples and oranges
  - No „standard“ YCSB results published by anybody

- Roughly...
  - PK SELECT – 1.5x … 4x faster than SQL
  - PK INSERT – upto 9x faster than SQL
  - Connect is way faster than SQL
  - HandlerSocket: try yourself, script for download

The speaker says...

Kasper: (from the off, whispering) You better have some strong slides coming Seagull – stay on the positive side...

Benchmarking is hard to do. MySQL is optimized for multi-core machines, whereas Redis, for example, can hardly use more than one core with its single-threaded architecture! Thus, quick-and-dirty benchmarks from a developers notebook may point the wrong direction. An option for a serious benchmark would be the Yahoo! Cloud Serving Benchmark – no results known. For those of you that want to play single-core, grab this script. On my notebook it showed HandlerSocket (MySQL 5.5) behind InnoDB Memcached (MySQL 5.6) by 20-30% for connect + fetch.

Gotcha HandlerSocket : 30s

```
if (!($hs_r->openIndex(PHP_INT_MAX, …))
```

```
terminate called after throwing an instance of 'std::bad_alloc'
  what():  std::bad_alloc
21:14:12 UTC - mysqld got signal 6 ;
[...]
Attempting backtrace. You can use the following information to find
out where mysqld died. If you see no messages after this, something
went terribly wrong...
[...]
usr/lib64/libstdc++.so.6(_Znwm+0x7d)[0x7f1e08088ecd]
/data/nixnutz/ftp/mysql-
5.5.30/install/lib/plugin/handlersocket.so(_ZNSt6vectorIN4dena9prep
_stmtESaIS1_EE14_M_fill_insertEN9__gnu_cxx17__normal_iteratorIPS1_S
3_EEmRKS1_+0xb6)[0x7f1e064e2c46]
```

The speaker says…

Kasper: (from the off, loud) Seagull, you are running out of time. And, this is bad style. It would be one line to fix this. Plus, HandlerSocket manual explicitly asks to use small numbers.

Seagull: (hicks) Yes, Sir! However, his is what happened to me after 30 seconds. Only one more for the fair play...

Gotcha Memcached: 3000s?

```
uninstall plugin daemon_memcached; install plugin daemon_memcached
soname 'libmemcached.so';
Query OK, 0 rows affected (50,04 sec)

ERROR 2013 (HY000): Lost connection to MySQL server during query
```

```
[...]
2013-03-17 19:29:28 38575 [Note] Shutting down plugin
'daemon_memcached'
 InnoDB_Memcached: column 6 in the entry for config table
'containers' in database 'innodb_memcache' has an invalid NULL
value
Failed to initialize instance. Error code: 13
2013-03-17 19:29:37 38575 [Note] Shutting down plugin
'daemon_memcached'
```

The speaker says…

Kasper: (coming to the stage, loud) Seagull!

Seagull: (hicks) But, Sir! This is even documented. If you misconfigured the plugin accidently, the server may run into troubles. If so, one shall set daemon_memcached=OFF in the server configuration and restart the server. Then, fix the issue. However, do not issue UNINSTALL PLUGIN if SHOW PLUGIN shows an disabled InnoDB Memcached plugin. If you do, your server crashes...

Fifth act: Grand finale with Innobear

The speaker says…

Kasper: Ladies and gentlemen, Seagull needs a rest. Innobear has come back for the grand finale.

Zoom! MySQL as a KVS

Try the NoSQL APIs!	High Performance
SQL for ad-hoc querying	Limited Search and Types
Threaded/Multi-Core, Replication	Scaleable
In-memory, on-disk with fast recovery	Limited Persistence

The speaker says...

The InnoDB Memcache Plugin is certainly a step forward. MySQL is putting pressure on itself to modularize the server allowing users to slim MySQL, to strip off features not needed to get a certain job done.

Users get more choices. If you want to combine a fast and lean client protocol with simple and fast access operations but cannot accept compromises on persistence or scalability, here you go.

BTW: Cluster has been a speed monster ever since. In late 2012 we published benchmarks with 4.3B ops.

Happy End: Sakila, Sakila and Sakila in ITs' yellow press!

Supercharge your web apps with MySQL
Really rich queries!
Free Sakila collector card
*** Exclusive tipps: natural beauty at any age ***
*** True: NoSQL weekly recommends MySQL whitepaper ***

THE END

Contact: ulf.wendel@oracle.com

The speaker says…

Speaker grabs a bottle of beer.

His phone rings. He picks up.

Speaker: No, I did not forget anything. No, darling, I will not do that... (listens) … Ok, ok! (hangs up)

Speaker: Turns to the audience. Ok, you get to hear a nightcap. But, I will not sing for you.

Helan går
Sjung hopp faderallan lallan lej
Helan går
Sjung hopp faderallan lej
Och den som inte helan tar
Han heller inte halvan får
Helan går
(Drink)
Sjung hopp faderallan lej

(Music plays, singer: Kaj Arnö)

The whole goes
Sing "hup fol-de-rol la la la la"
The whole goes
Sing "hup fol-de-rol la la"
And the one who doesn't take the whole
Doesn't get the half either
The whole goes - (Drink)
Sing "hup fol-de-rol la la"

Nightcap: Helan Går

The speaker says…

Music plays:

http://www.youtube.com/watch?v=HyZpCs3i71s

Speaker: You have still not fallen asleep? Ah, you are looking at my beer bottle. Hmm, well, don't tell your mom... (hands over some bottles and shot glasses to the audience)

Please, serve yourself. By the end of free bonus coming now, I expect you all to snooze peacefully.

Free bonus: beyond the fairy tale

The speaker says…

As you are having your shots, allow me to summarize the talk.

Ulf's take... - Awareness

Not a bad attempt at all... Go try! Go ask for more!

A significant number of MySQL users is using Memcached

- Deploy only one data store instead of two
- Dual interface: can we skip a caching layer in our apps?

A good first step, but looking for more

- Persistence for Mem*cache* - more of a topic for Redis?
- No issues with warm-up or stampeding/slamming
- KVS is about performance, where is the proof @ 5.6...?
- Data model is about distribution/sharding, MySQL Cluster only?

The speaker says…

The InnoDB Memcached Plugin is a valuable addition to MySQL. It can be interpreted as both a reaction to NoSQL ideas but also (and foremost) as a reaction to customer demand. Using MySQL with Memcached is extremly popular – ever since. Sakila and Memcache are best friends. Check for how long the MySQL manual includes exhaustive documentation for using the two data stores together. Adding two persistent storage backends (MySQL Cluster, InnoDB) to Memcache makes sense: their performance is competitive.

Integrating the Memcached into MySQL means: simplified application architecture (no cache layer), thus lower costs. And, as a benefit on top, a dual SQL/KVS interface to the data.

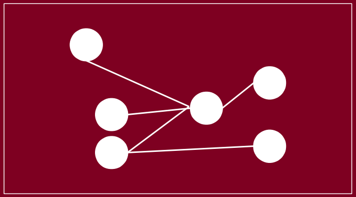
Swipe, tap, click, and zoom

```
1234 - I am super FAST
4567 - LIGTHNING fast
7890 - [1][2][3][4][5]
abcd - [1[a,b,c]],[2[d,e,f]]
_de_- 1,281,2828,2,173,8
```

Key Value Store

```
{"documents":"rock"}
{"mysql":"not"}
{"documents":"rock"}
{"mysql":"not"}
{"documents":"rock"}
```

Document Database



Graph Database

```
0101010010110 10
1010101101001 01
1010100101010 10
1010101101010 11
1010101101111 01
```

Big Data/Column-oriented

The speaker says...

As a reaction to NoSQL, however, it is a small step only. NoSQL is more than Key-Value-Stores. Though, Key-Value Stores may be the most popular kind of NoSQL deployed today. Read: the biggest customer demand.

InnoDB Memcached mimics one of the oldest Key-Value Stores. In a way, Memcached is a first generation KVS. Type system and protocol/query capability are puristic unlike a recent remote data infrastructure server (redis). Given that InnoDB can serve as a backend for highly complex SQL and very simple KVS, can it be bend to serve other purposes as well?

Zoom! MySQL vs Documents

Hmm...	Hierarchical/nested data, JavaScript/[J|B]SON
Replication: Ok, add 3rd party Cluster: beat it!	Highly Available
Replication: Write limit Cluster: beat it!	Scaleable, Map&Reduce
Database and tooling: good Need for ORM: hmm...	Easy To Use

The speaker says...

NoSQL is not only about interfaces. It is also a wonderful potpurri of new and old data models. For examples, the hierarchical document data model goes back to the 60th – think IMS. In the 70th and 80th Codds relational model and later SQL has seen criticized for allowing atomic data types only. SQL:2003 has 90% of what is needed to store and query arbitrarily structured JSON documents in a relational database! Read about it at: blog.ulf-wendel.de/ .

However, NoSQL is really pushing RDBMS on clustering. We are not talking 4, 40 or 100 nodes here. We are thousands of nodes, possibly used with elastic sharding. Kind of doable with MySQL but not out of the box!

Zoom! JavaScript/[J|B]SON

Proof of Concept

MySQL speaks HTTP and replies JSON.

JavaScript (v8) runs inside MySQL.

Map&Reduce jobs use the internal
low-level high performance interfaces.

… it could be done

http://de.slideshare.net/nixnutz/http-json-javascript-mapreduce-builtin-to-mysql

The speaker says...

Speaking of interfaces and key value stores that hold documents... it could be done. Thank you for your attendance!

Upcoming shows:

International PHP Unconference

Berlin, May 2013 (sold out)

International PHP Conference Spring Edition

Berlin, June 2013