



libdrizzle

Eric Day

eday@oddmments.org

MySQL Conference & Expo 2009

<https://launchpad.net/libdrizzle>

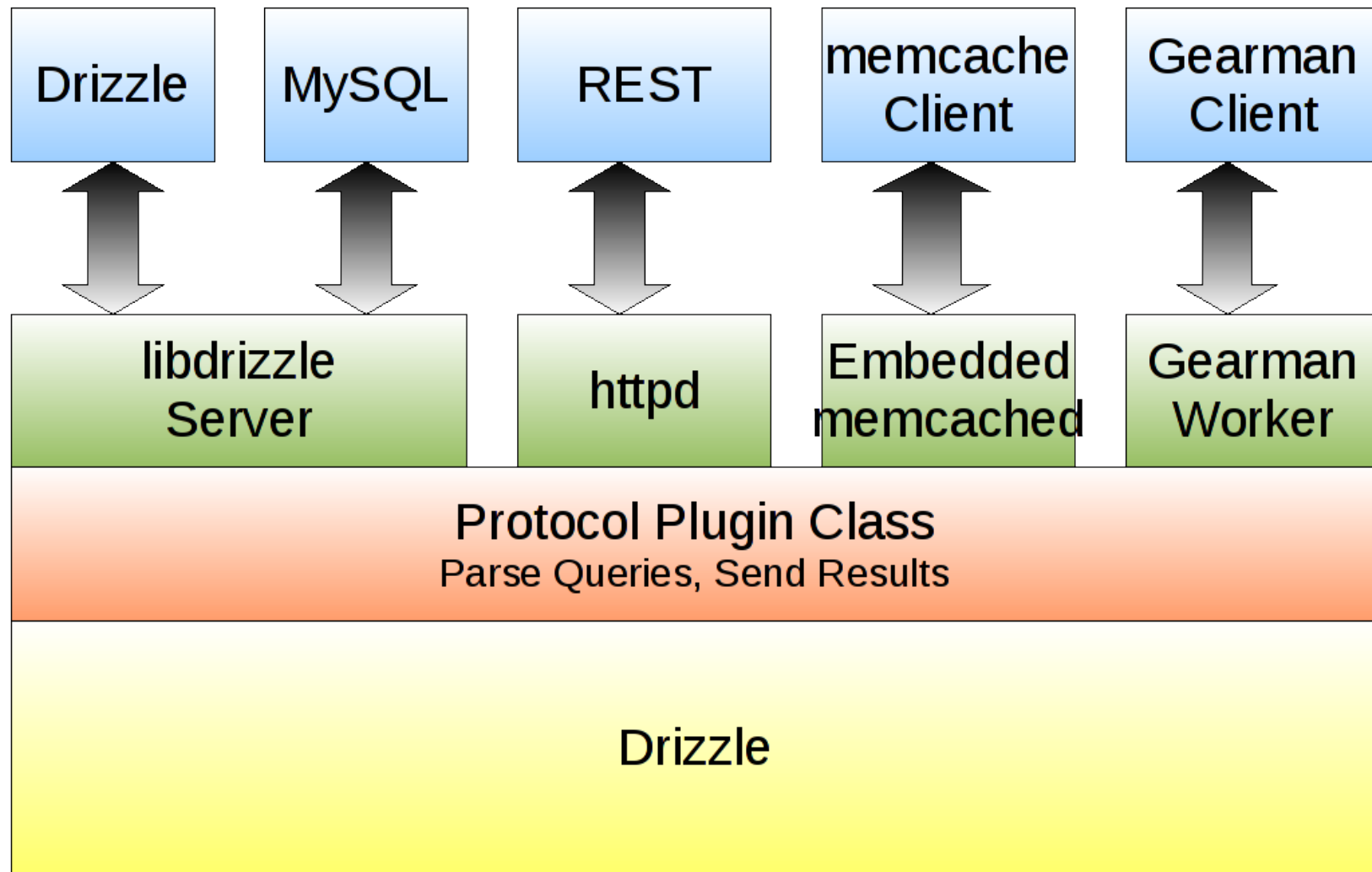
Overview

- What is libdrizzle?
- Client example
- Result buffering
- Non-blocking I/O
- Concurrent client example
- Server interface
- New protocol
- Future

Wait, what is Drizzle?

- Derived from MySQL 6 source code
- Rethink **everything**
- Community driven development
 - 100+ Contributors
 - 400+ Mailing list
- Focus on micro-kernel design, multi-core hardware (think 128+), “The Cloud”
- <http://drizzle.org/>
- <https://launchpad.net/drizzle>

Drizzle Protocol Plugin



Why libdrizzle?

- The derived MySQL client library needed an overhaul, easier to start from scratch
- Rethink I/O and result buffering
- Concurrency
- Improve the developer experience through an intuitive interface
- Written in straight C with other language interfaces built on it (except Java)

libdrizzle

- The new libdrizzle was born
- Complete client and low level protocol library
- Supports the MySQL protocol as well
- BSD license
- C - <https://launchpad.net/libdrizzle>
- PHP- <https://launchpad.net/drizzle-php-ext>
- Monty Taylor - SWIG (Perl, python, Ruby, ...)
<https://launchpad.net/drizzle-interface>
- Perl DBD coming soon

Code!

```
$drizzle= new drizzle();

$con= $drizzle->add_tcp("server", 4427,
                        "user", "pass",
                        "db", 0);
$result= $con->query("SELECT ...");

$result->buffer();

while (($column= $result->column_next()) != NULL)
    print $column->name() . ':';

while (($row= $result->row_next()) != NULL)
    print implode(':', $row) . "\n";
```

Query Result Buffering Options

- Don't buffer anything, use data directly from buffer used for read()
- Buffer at the field data level
- Buffer at the row level
- Buffer the entire result
- Or mix and match, can switch between modes by using another function (great for BLOBs)
- Last example was full result buffering

Row buffering

```
$result= $con->query("SELECT ...");  
  
while (($column= $result->column_read()) != NULL)  
    print $column->name() . ':';  
  
while (($row= $result->row_buffer()) != NULL)  
    print implode(':', $row) . "\n";
```

Field buffering

```
while (($column= $result->column_read()) != NULL)
    print $column->name() . ':';
```

```
while (($row= $result->row_read()) != 0)
{
    while (1)
    {
        list($ret, $field)= $result->field_buffer();
        if ($ret == DRIZZLE_RETURN_ROW_END)
            break;

        print $field . ':';
    }
}
```

No buffering!

```
while (($column= $result->column_read()) != NULL)
    print $column->name() . ':';
```

```
while (($row= $result->row_read()) != 0)
{
    while (1)
    {
        list($ret, $field, $offset, $total)=
            $result->field_read();
        if ($ret == DRIZZLE_RETURN_ROW_END)
            break;

        print $field;

        if ($offset + strlen($field) == $total)
            print ':';
    }
}
```

```
lap> valgrind ./client -d "sakila" "SELECT name FROM category" | wc -l
...
==9361== malloc/free: in use at exit: 0 bytes in 0 blocks.
==9361== malloc/free: 4 allocs, 4 frees, 37,340 bytes allocated.
...
64
```

```
lap> valgrind ./client -d "sakila" "SELECT * FROM film JOIN actor" | wc -l
...
==9367== malloc/free: in use at exit: 0 bytes in 0 blocks.
==9367== malloc/free: 4 allocs, 4 frees, 37,340 bytes allocated.
...
3800192
```

Non-blocking I/O

- Normally, `read()` and `write()` will block until done
- Enable non-blocking with `fcntl()` `O_NONBLOCK`
- Now `read()` and `write()` do what they can and return `EAGAIN` when it would block
- Block for I/O on multiple file descriptors using `poll()`, `select()`, ...
- Partial reads and writes possible
- You need to keep more state

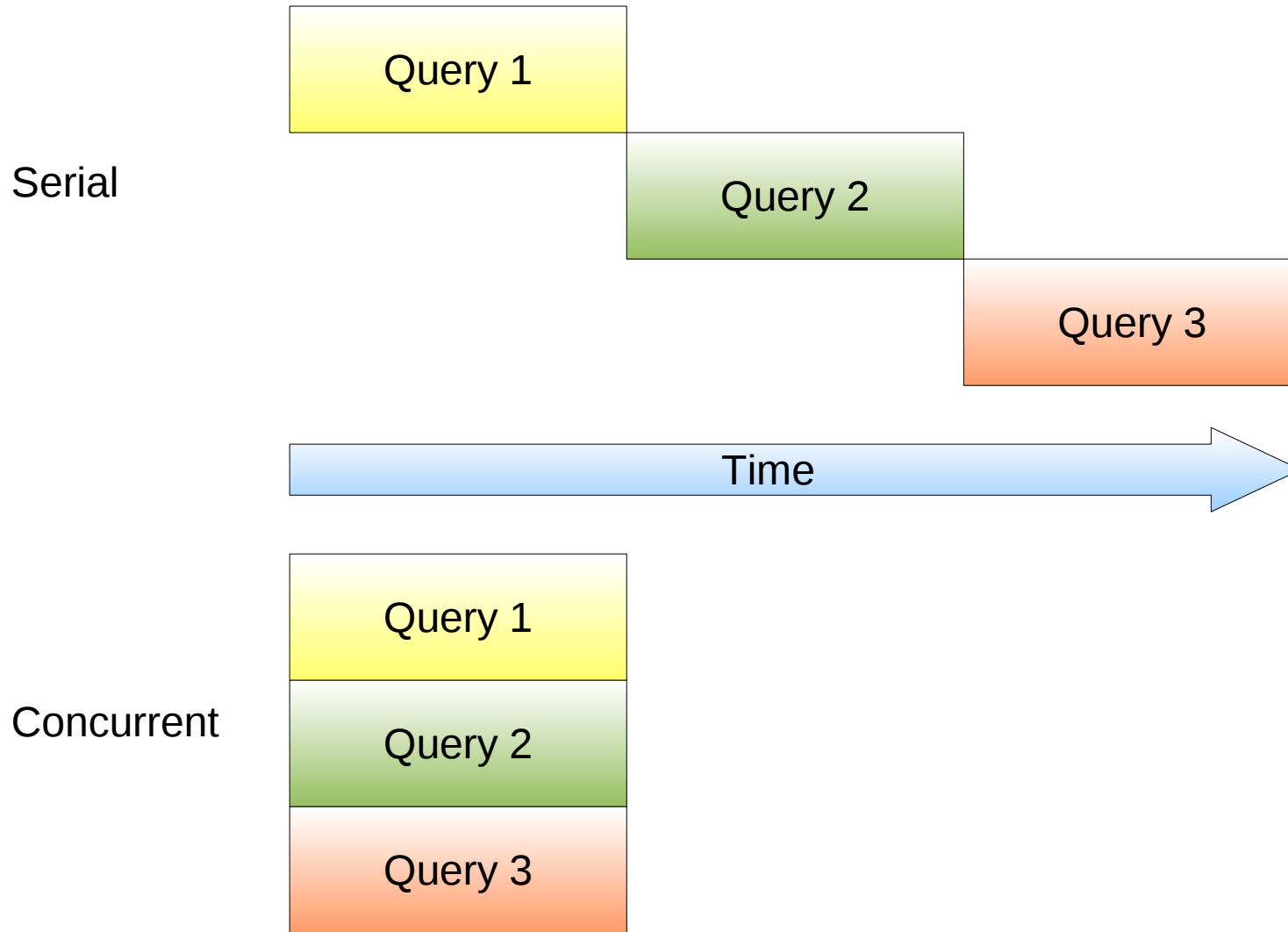
Non-blocking I/O in libdrizzle

- All I/O is actually non-blocking
- Flags expose non-blocking interface to user
- Uses poll(), will optionally use libevent soon
- Allows for concurrent connections
- ... and queries!

Concurrent Queries

- Most web applications make multiple requests
- Some of those requests are non-transactional read-only requests
- Candidates to be batched concurrently
- Decreases overall load time

Concurrent Queries



Concurrent Queries

```
$drizzle= new drizzle();

for ($x= 0; $x < 3; $x++)
{
    $con= $drizzle->add_tcp(NULL, 0, "root", NULL, $db, 0);
    $drizzle->query_add($con, "SELECT ...");
}

while (1)
{
    list($ret, $query)= $drizzle->run();
    if (!$query)
        break;

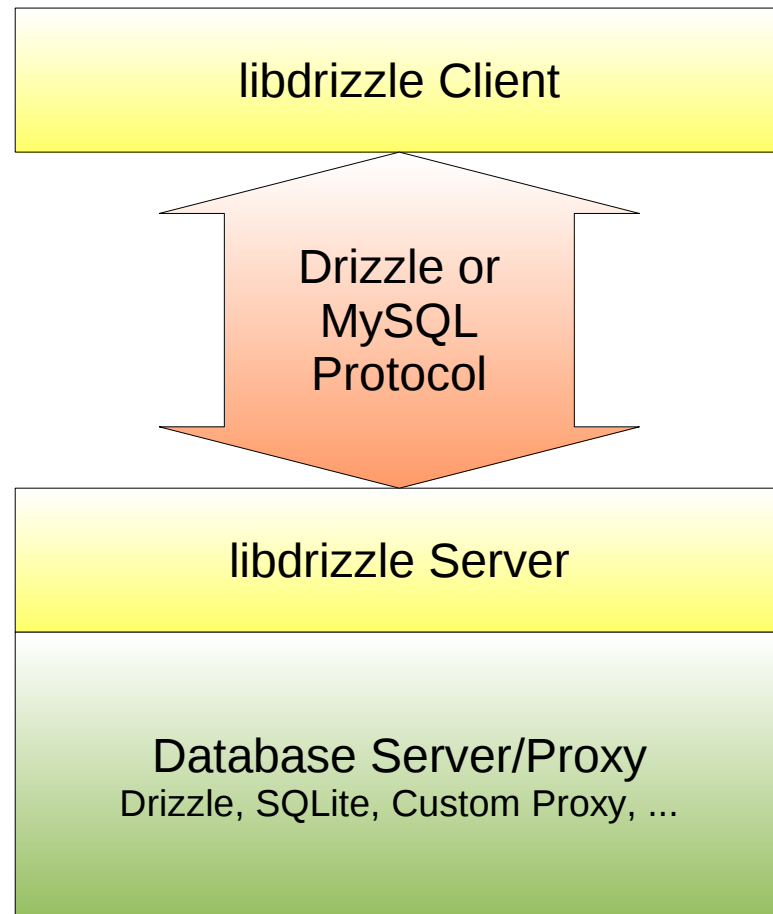
    $result= $query->result();

    # Print out result like before
}
```

Server Interface

- libdrizzle is not just a client interface
- Full protocol implementation library
- Provides server protocol interface
- Use it to write proxies or “fake” Drizzle and MySQL servers
- SQLite hack
 - `examples/sqlite_server.c` in libdrizzle

Server Interface



New Drizzle Protocol

- Asynchronous
- Two layers of packets
- Sharding key in lower layer packet
- Read-only hints for sharding
- UDP support for some queries
- Packets broken down into chunks
- Authentication plugins (optional)
- Checksums
- Concurrent queries on a single connection

Future

- Still under active development
- New protocol still being implemented
- Higher level interfaces still being developed
- PHP extension ready
- Python, Perl, Ruby almost ready
- Proxies will soon be developed
 - Will be able to use both MySQL and Drizzle

Questions?

- Drizzle developer day Friday!
- http://drizzle.org/wiki/Drizzle_Developer_Day_2009
- eday@oddmments.org
- <https://launchpad.net/libdrizzle>
- <https://launchpad.net/drizzle-php-ext>