MySQL Proxy
Advanced Lua scripting

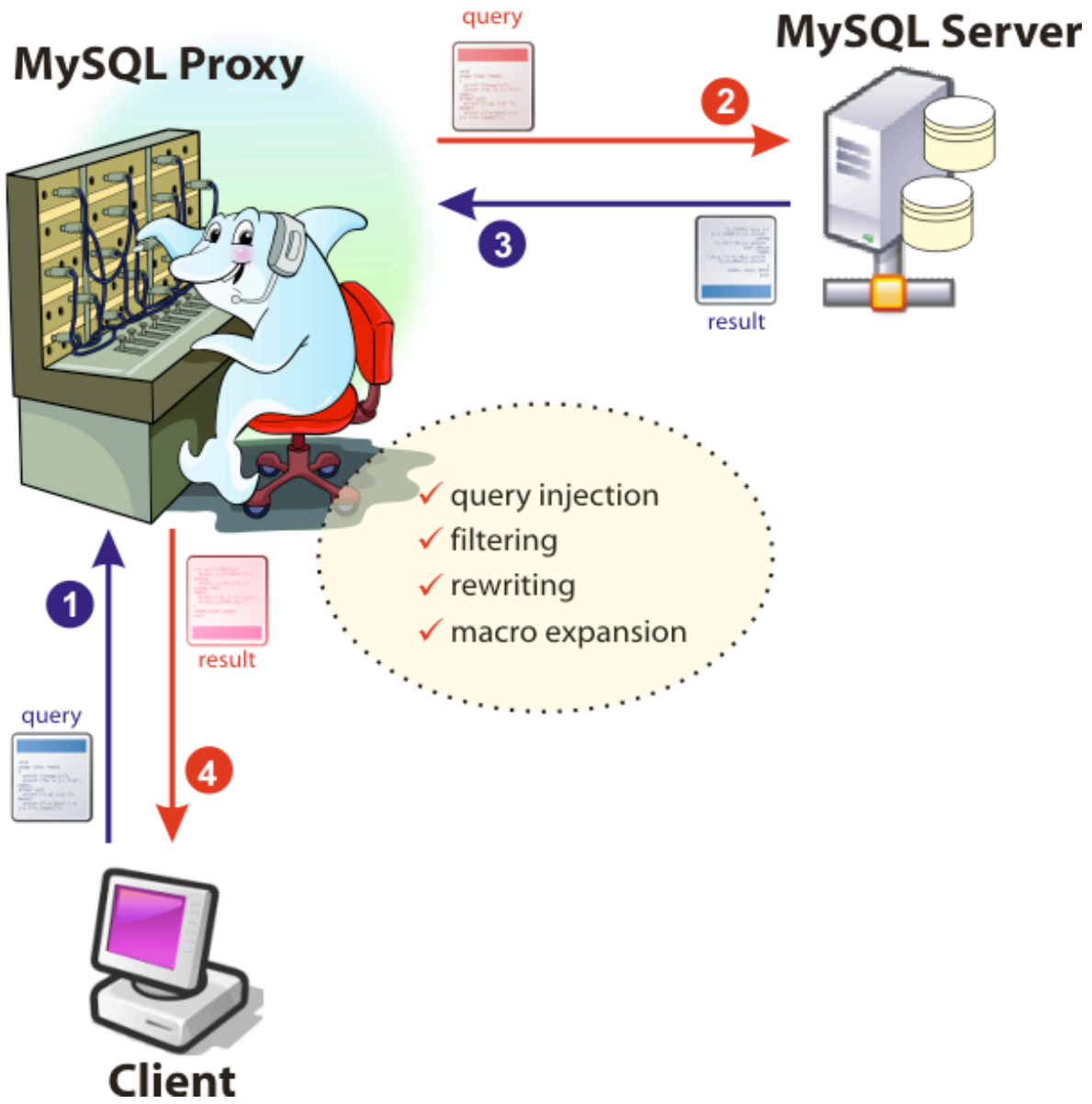
Giuseppe Maxia,

MySQL Community Team Lead,

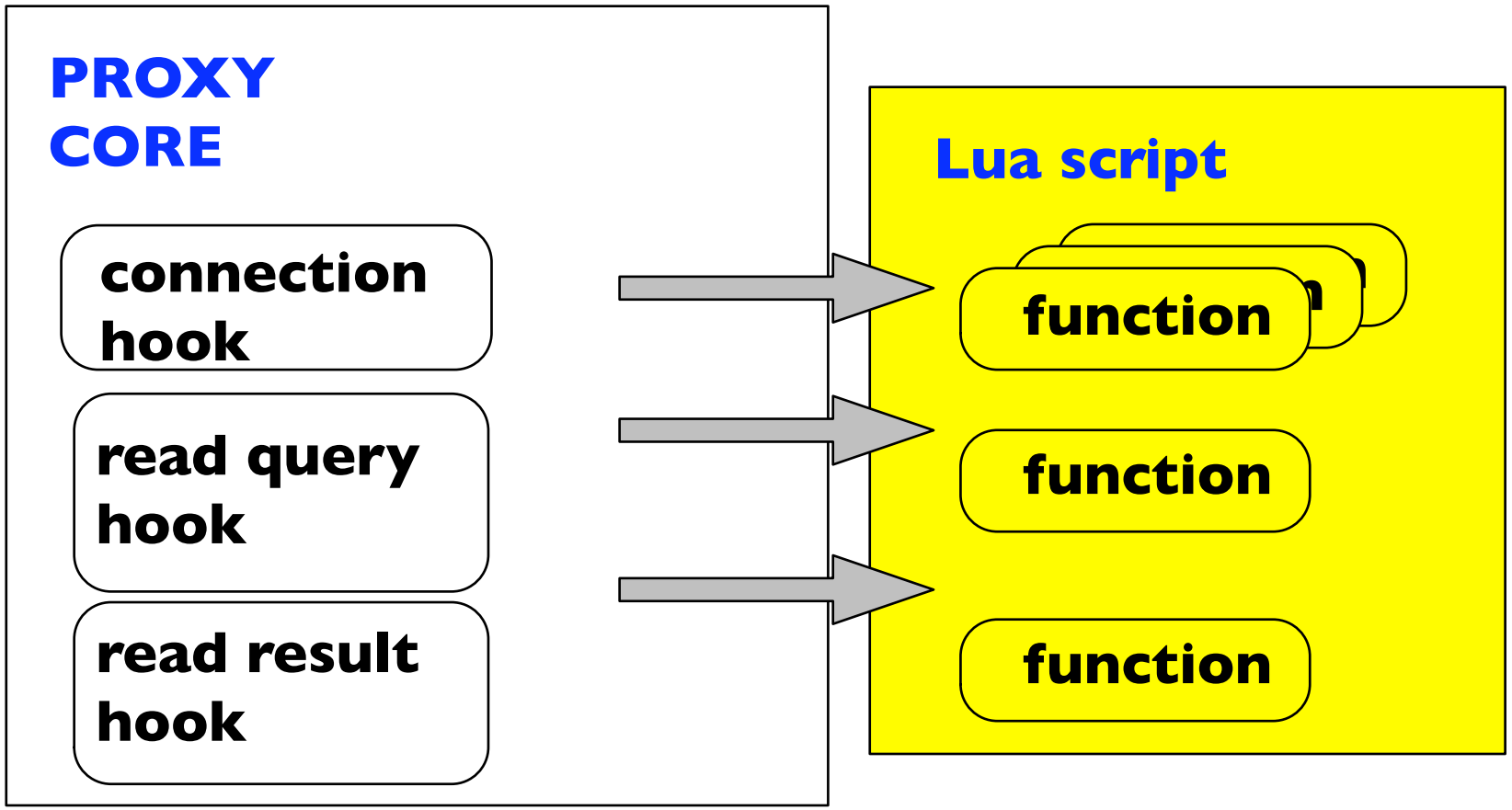
Sun Microsystems

Database Group

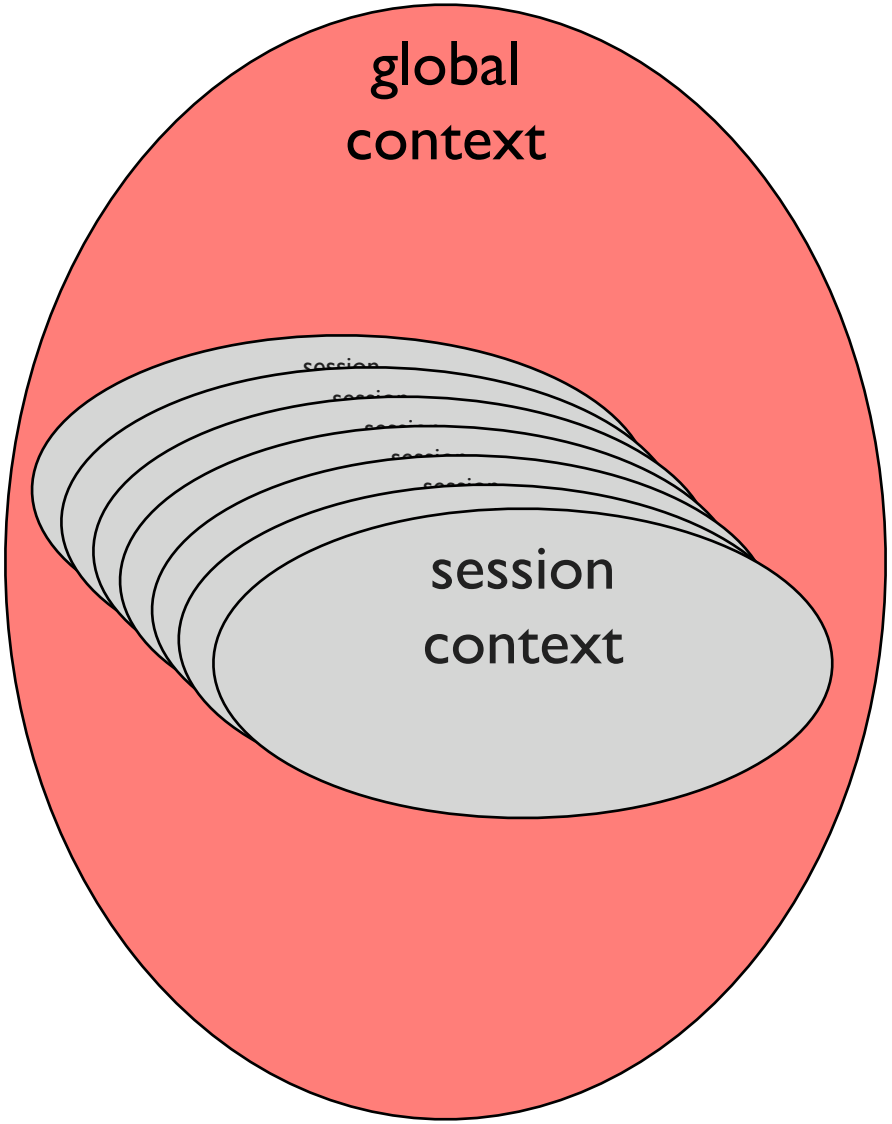
basic principles



basic principles



Proxy - Lua overview



Lua script

- connect_server
- read_handshake
- read_auth
- read_auth_result
- read_query
- read_query_result
- disconnect_client

Using Lua Files

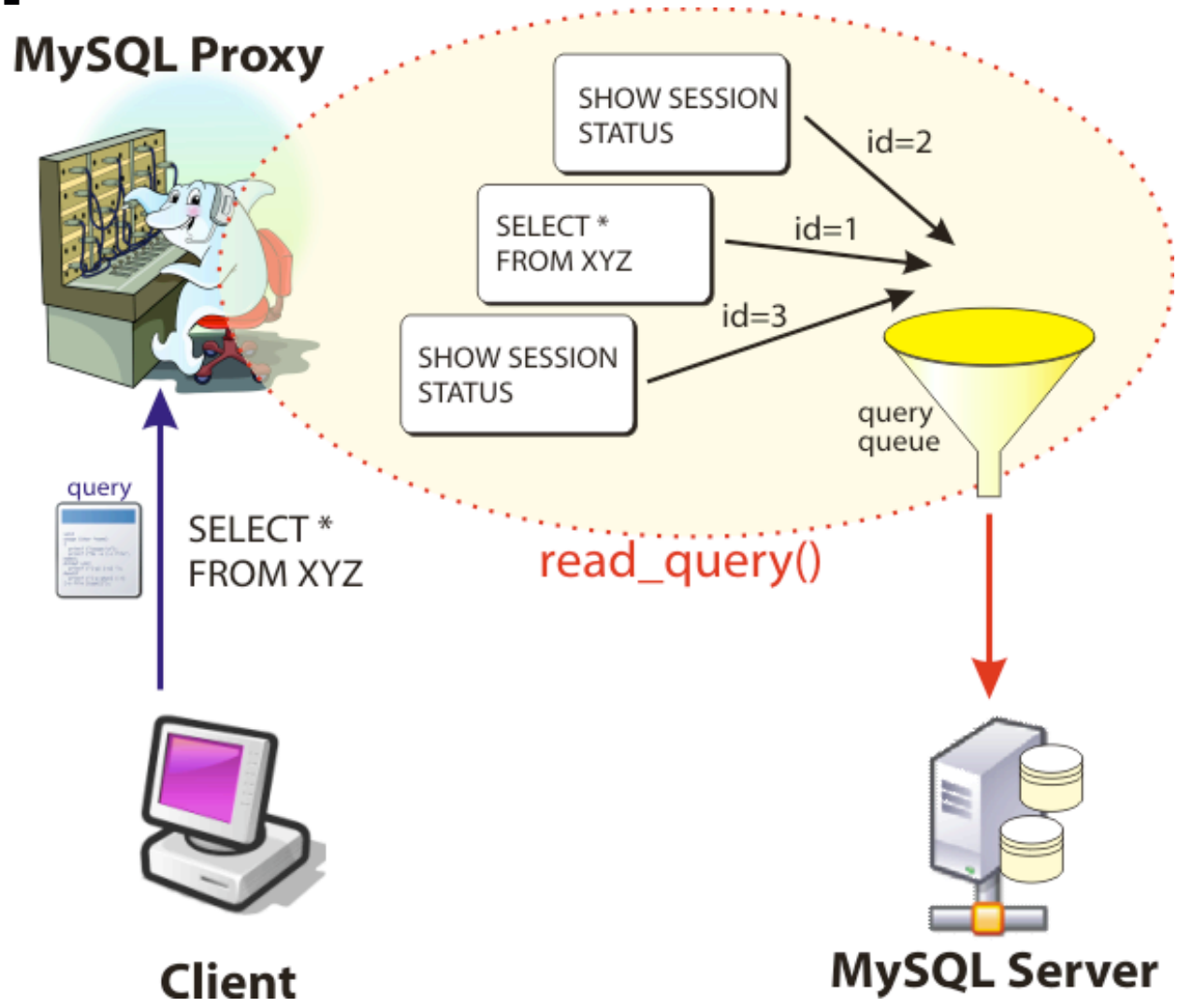
```
/usr/local/sbin/mysql-proxy \
  --proxy-lua-script=/path/name.lua
```

IMPORTANT!
THE SCRIPT DOES NOT START UNTIL THE FIRST CLIENT CONNECTION

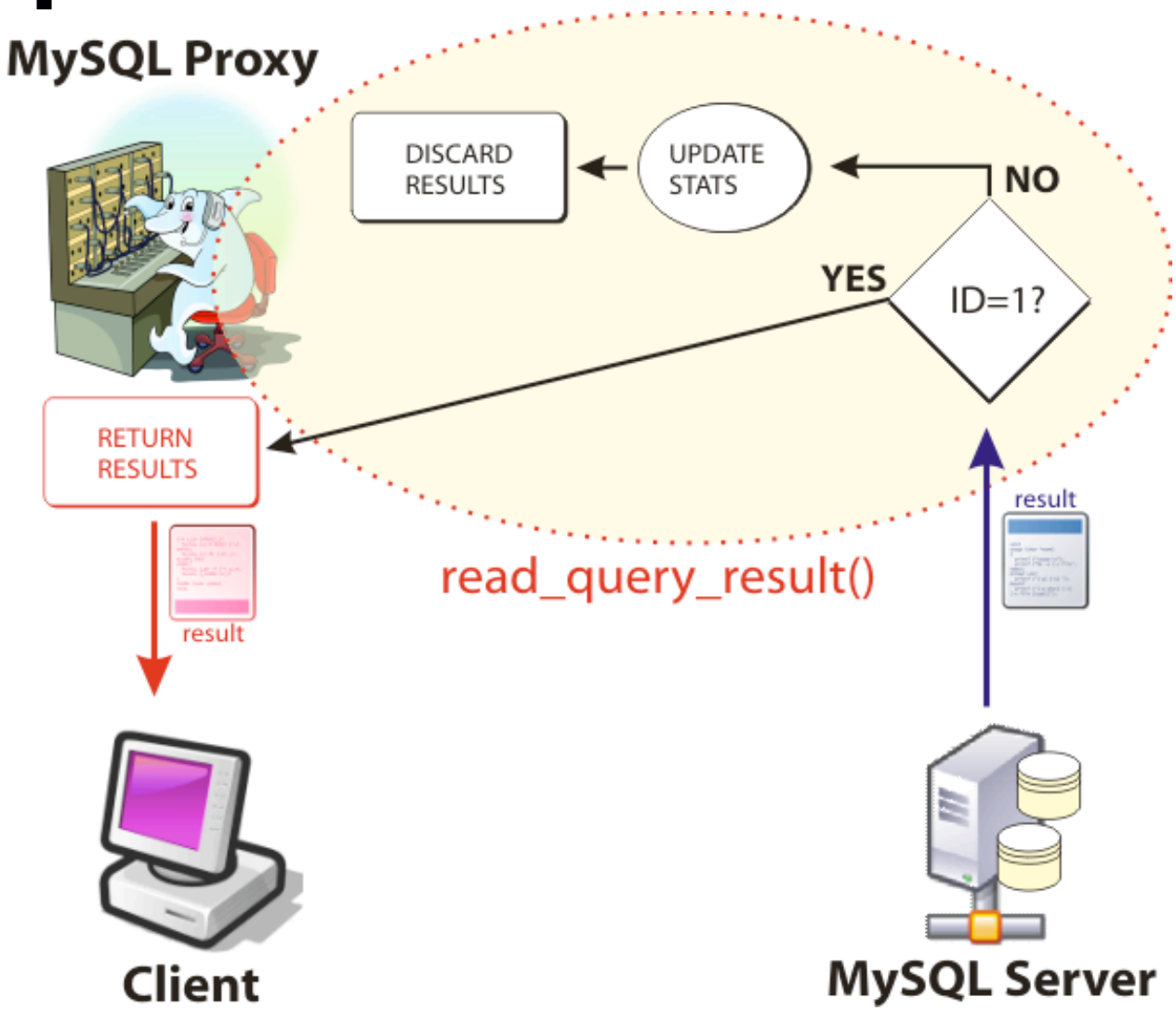
intercepting

```
function read_query(packet)
 if packet:byte() == proxy.COM_QUERY
  then
  local query = packet:sub(2)
  print("Hello world! Seen query: "
      .. query )
 end
end
```

injecting queries



injecting queries



injecting queries

```
function read_query(packet)
   -- ...
   proxy.queries:append(2, query1 )
   proxy.queries:append(1, packet )
   proxy.queries:append(3, query2 )

   return  proxy.PROXY_SEND_QUERY

end
```

injecting queries

```
function read_query_result(inj)

  if inj.id == 1 then
    return -- default result
  else
    -- do something
    return  proxy.PROXY_IGNORE_RESULT
end
```

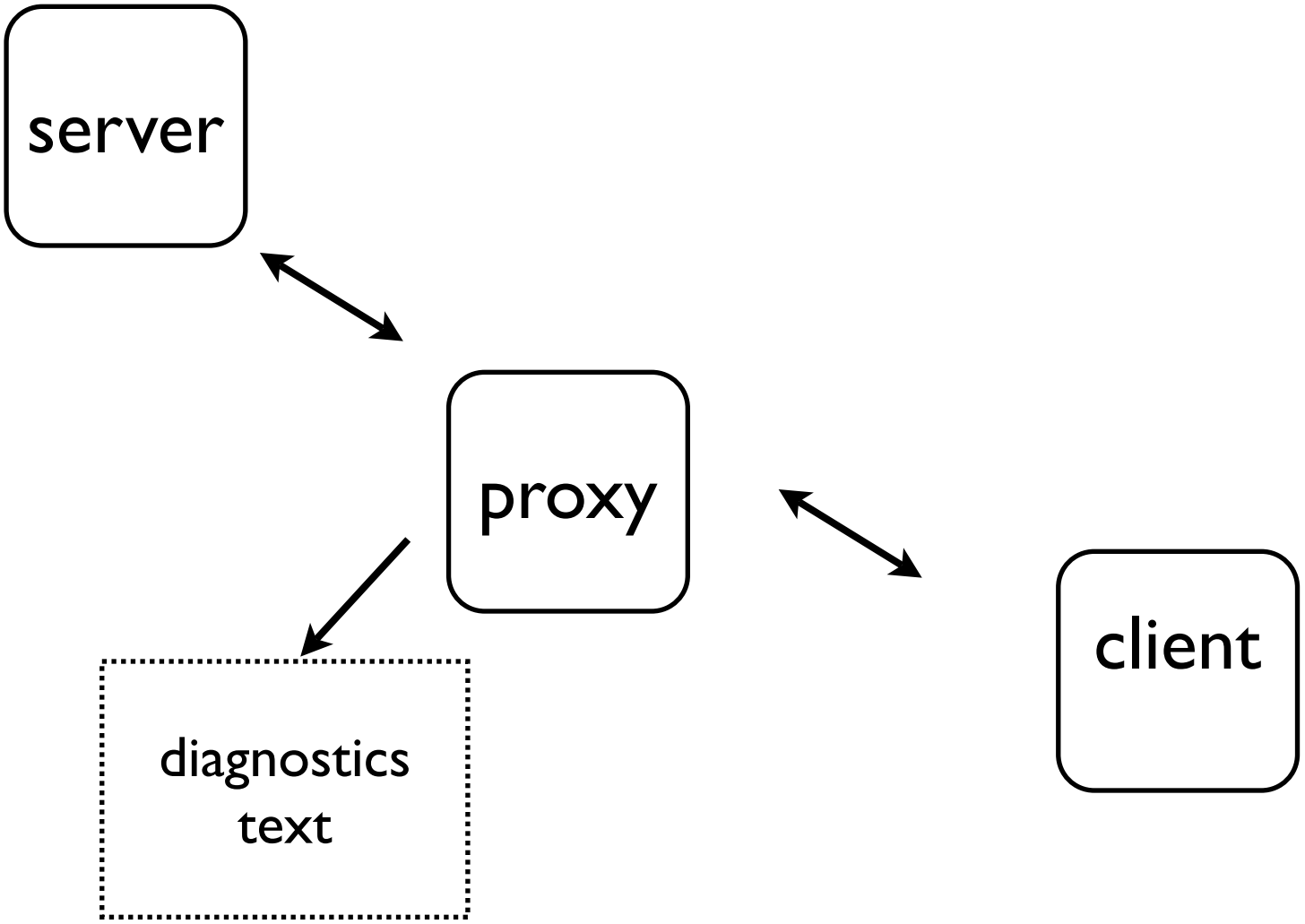
working with results

- return the original result
- return a fake result
- return an error
- alter the original result
- return something different (affected/retrieved)

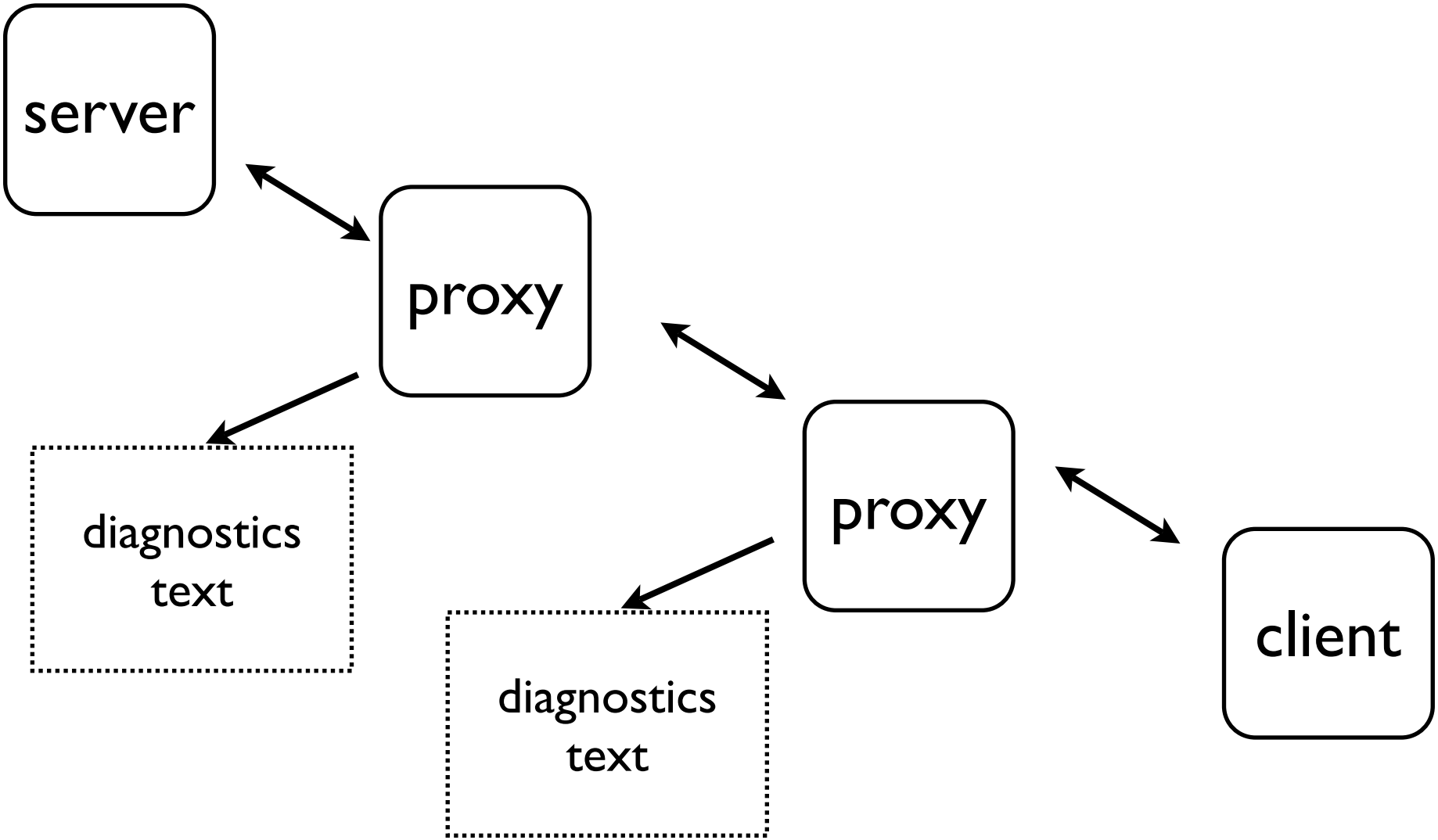
debugging

- Put a Proxy in between
- use a sensible script to see what's going on (e.g. tutorial-packets.lua or tutorial-states.lua)

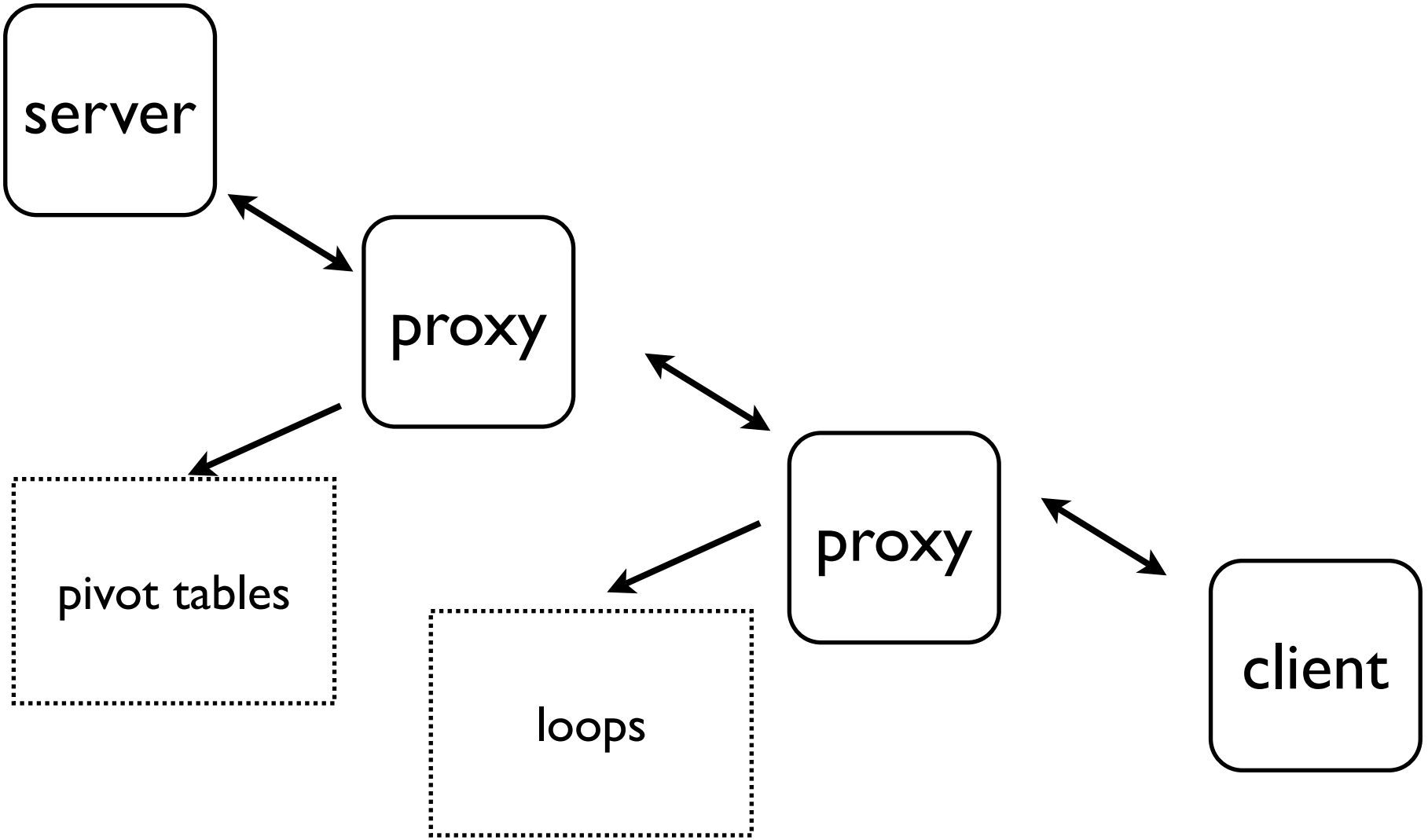
debugging



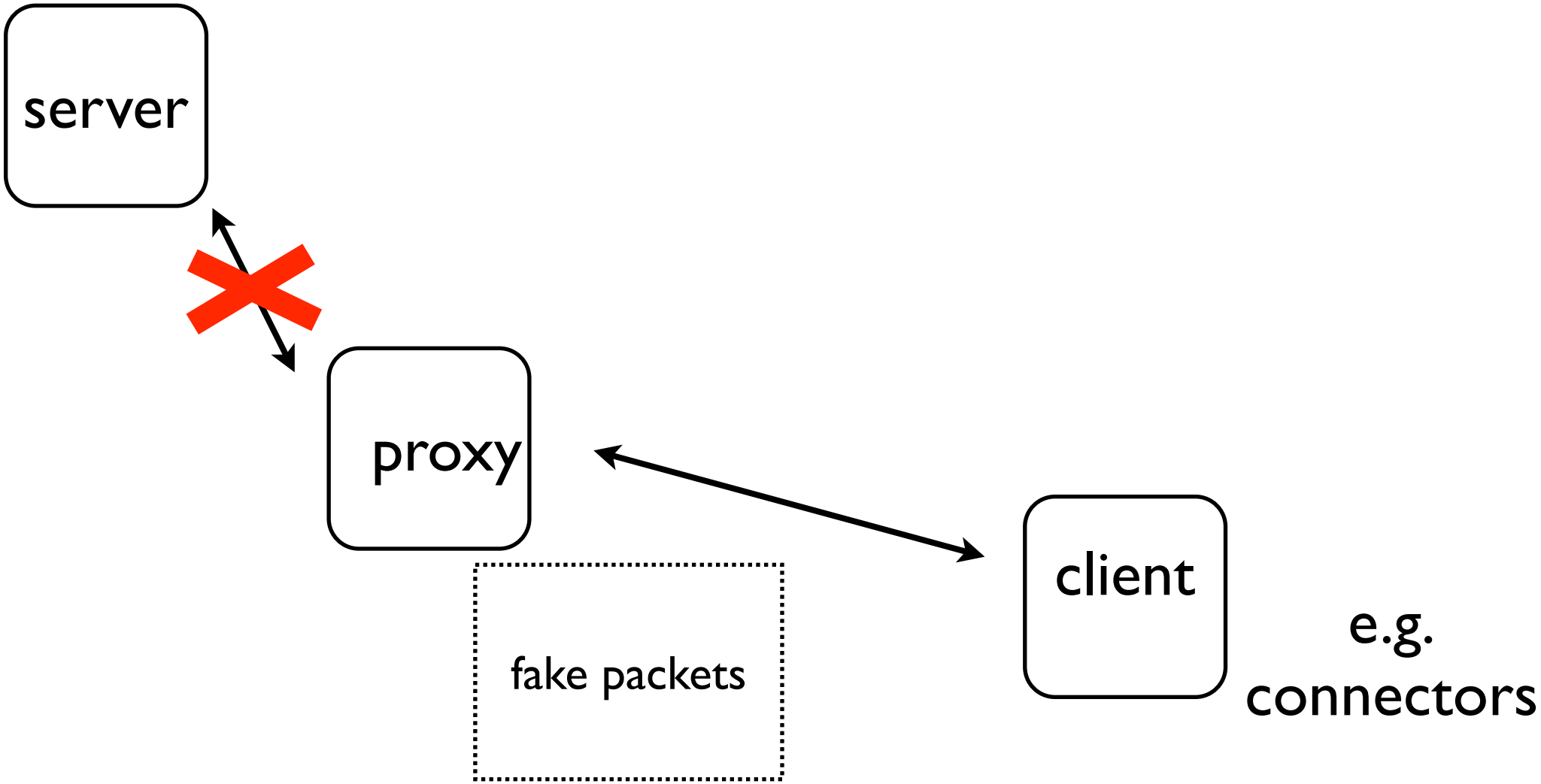
debugging scripts



chained proxy: double features

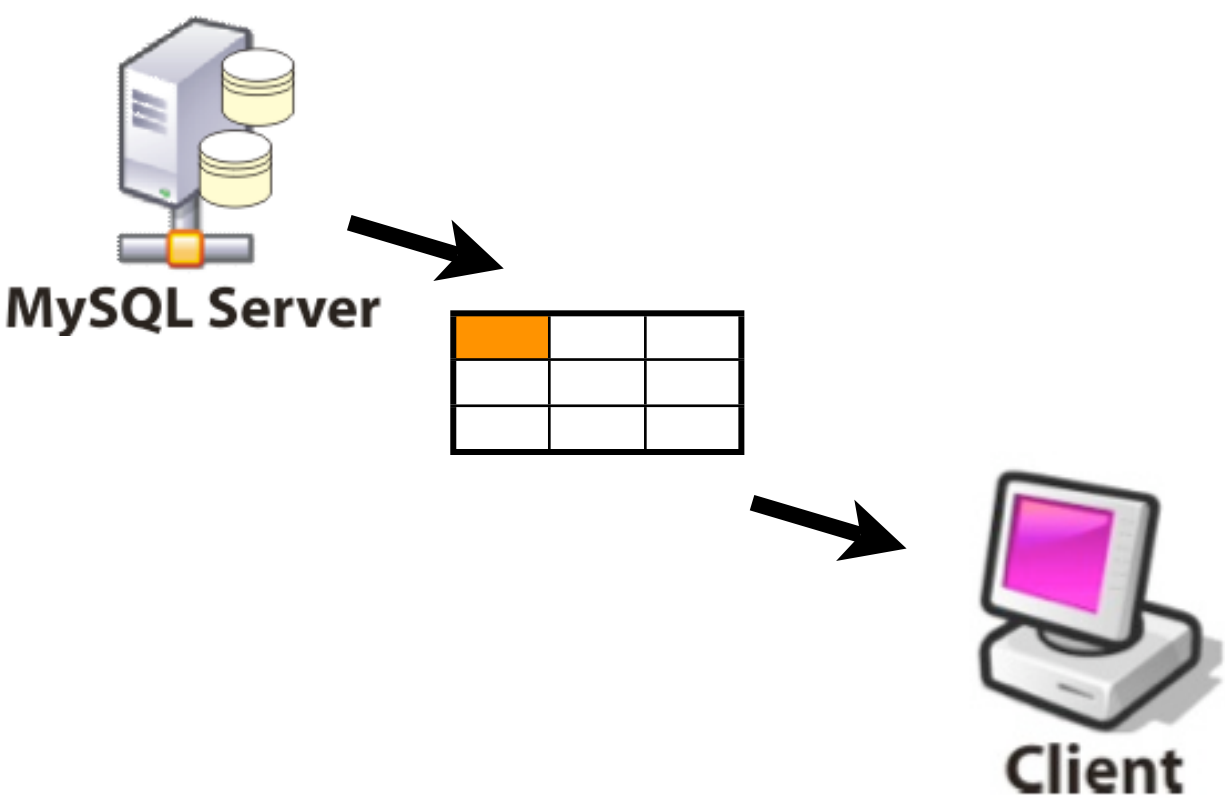


testing



MySQL Proxy recipes

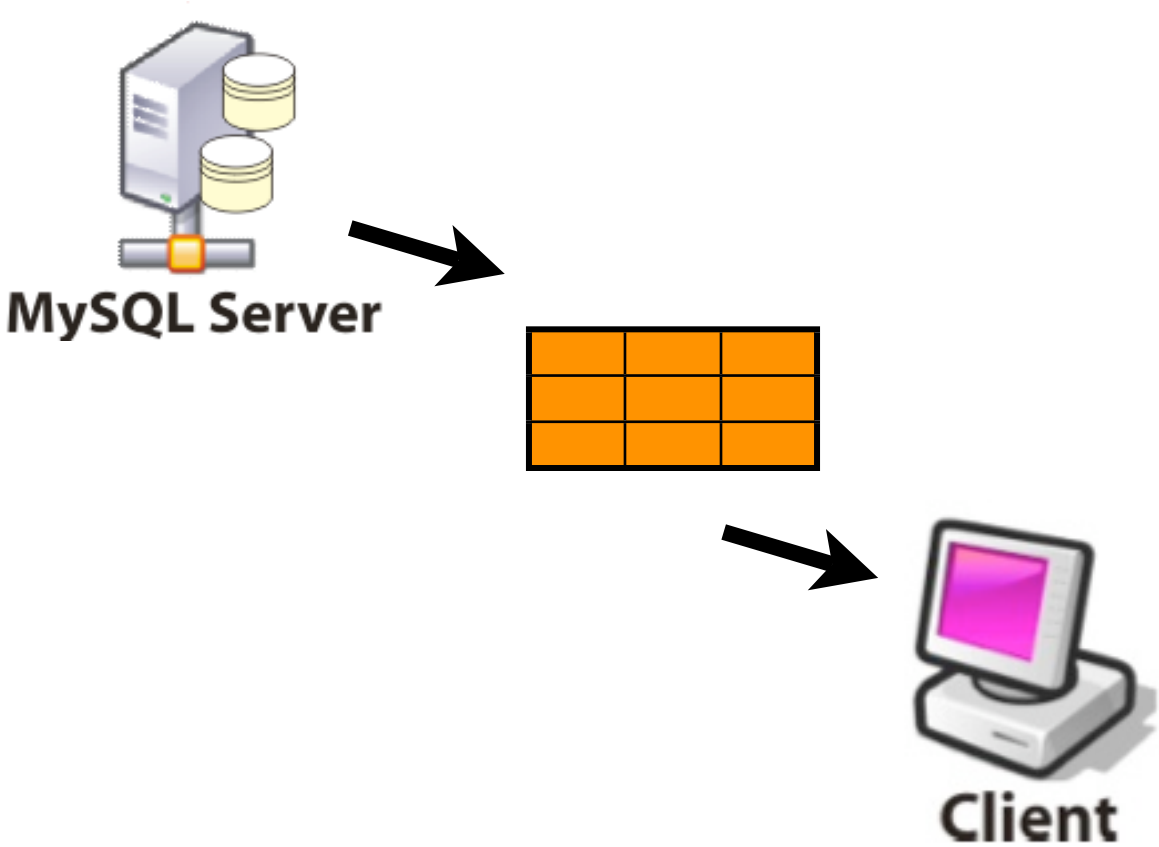
cookbook: returning a simple dataset



cookbook: returning a simple dataset

```
function simple_dataset (header, message)
proxy.response.type = proxy.MYSQLD_PACKET_OK
proxy.response.resultset = {
    fields = {
        {type = proxy.MYSQL_TYPE_STRING, name = header
    },
    rows = {
        { message}
    }
}
return proxy.PROXY_SEND_RESULT
end
```

cookbook: returning a full dataset



cookbook: returning a full dataset

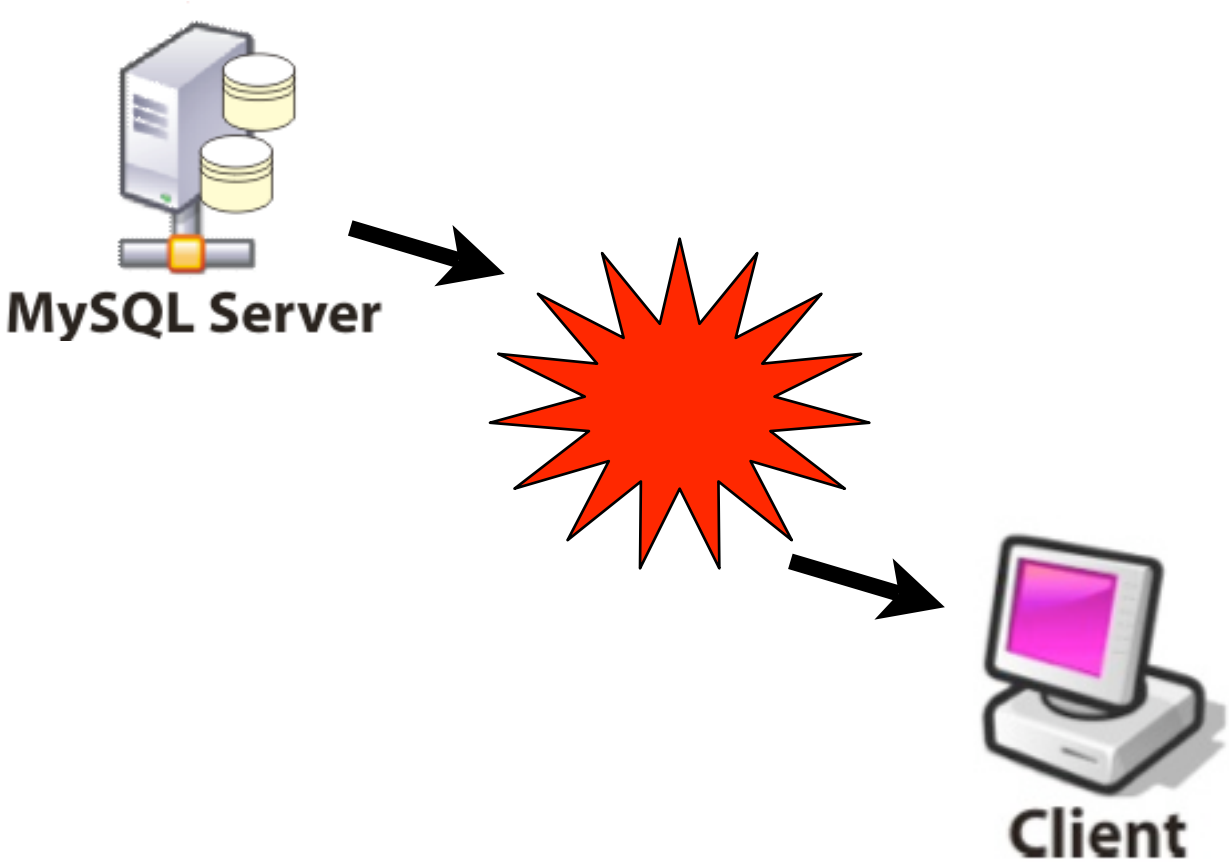
```
function make_dataset (header, dataset)
  proxy.response.type = proxy.MYSQLD_PACKET_OK

  proxy.response.resultset = {
      fields = {}, rows = {}}
  for i,v in pairs (header) do
      table.insert(
        proxy.response.resultset.fields,
            {type = proxy.MYSQL_TYPE_STRING, name = v
  end
  for i,v in pairs (dataset) do
      table.insert(proxy.response.resultset.rows, v )
  end
  return proxy.PROXY_SEND_RESULT
end
```

cookbook: returning a full dataset

```
return make_dataset(
  {'command', 'description' },        -- the header
  {                                   -- the rows
    {'FOO', 'removes the database'},
    {'BAR', 'drops all tables'},
    {'FOOBAR', 'makes the server explode'},
  }
)
```

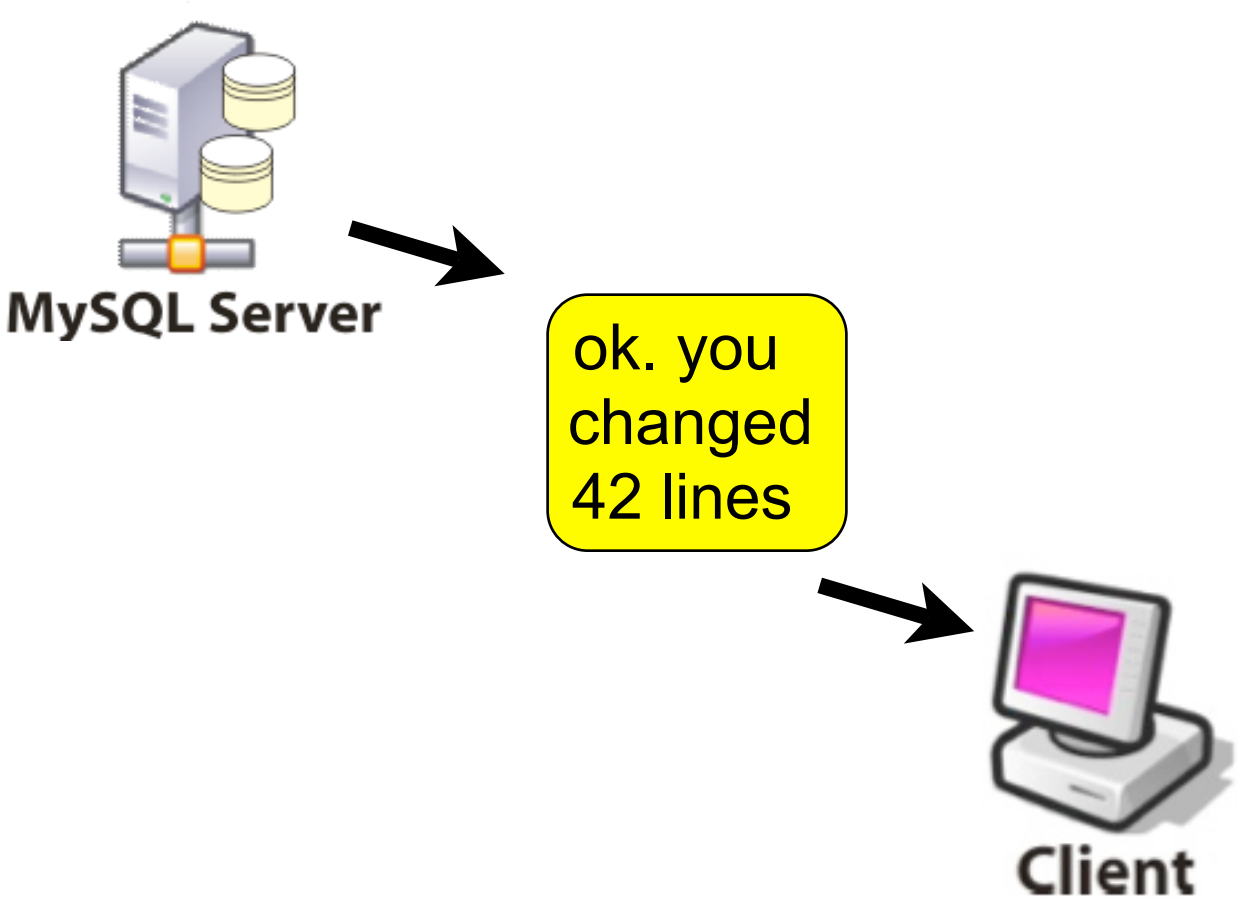
cookbook: returning an error



cookbook: returning an error

```
function error_result (msg, code,state)
  proxy.response = {
      type         = proxy.MYSQLD_PACKET_ERR,
      errmsg       = msg,
      errcode      = code,
      sqlstate     = state,
  }
  return proxy.PROXY_SEND_RESULT
end
```

cookbook: returning a non dataset result



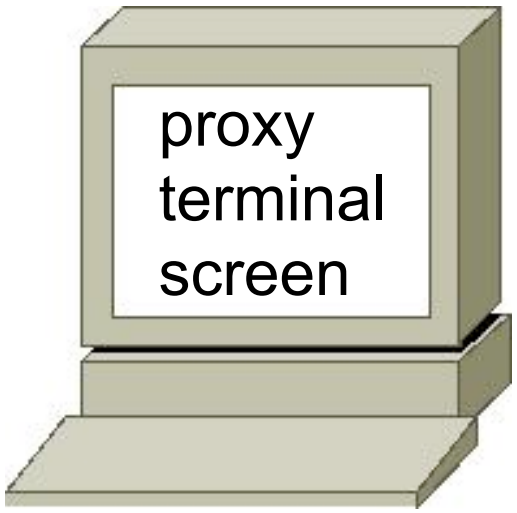
cookbook: returning a non dataset result

```
function affected_rows (rows,id)
  proxy.response = {
      type          = proxy.MYSQLD_PACKET_OK,
      affected_rows = rows,
      insert_id     = id,
  }
  return proxy.PROXY_SEND_RESULT
end
```

cookbook: debug messages

MySQL Server

got that query, blah, blah



Client

cookbook: debug messages

```
local DEBUG = os.getenv('DEBUG') or 0
DEBUG = DEBUG + 0

function read_query (packet )
 if packet:byte() ~= proxy.COM_QUERY then return end
 print_debug(packet:sub(2),1)
 print_debug('inside read_query', 2)
end

function print_debug(msg, level)
 level = level or 1
 if DEBUG >= level then
     print (msg)
 end
end
```

cookbook: verbose level at run time

```
local DEBUG = os.getenv('DEBUG') or 0
DEBUG = DEBUG + 0

function read_query (packet )
 if packet:byte() ~= proxy.COM_QUERY then return end
 local vlevel=query:match('^VERBOSE=(%d)$')
 if vlevel then
    DEBUG = vlevel+0
    return simple_dataset('verbose',vlevel)
 end
end
```

cookbook: keep info inside a session

proxy script

my variable value

my variable value

my variable value

cookbook: keep info inside a session

- nothing to do :)
- Proxy scripts have session scope by default

```
local tot_q = 0

function read_query (packet )
  if packet:byte() ~= proxy.COM_QUERY then return end
  tot_q = tot_q + 1
  print('queries ' .. tot_q)
end
```

cookbook: share info among sessions

MySQL Server

proxy
script

my variable value

Client

Client

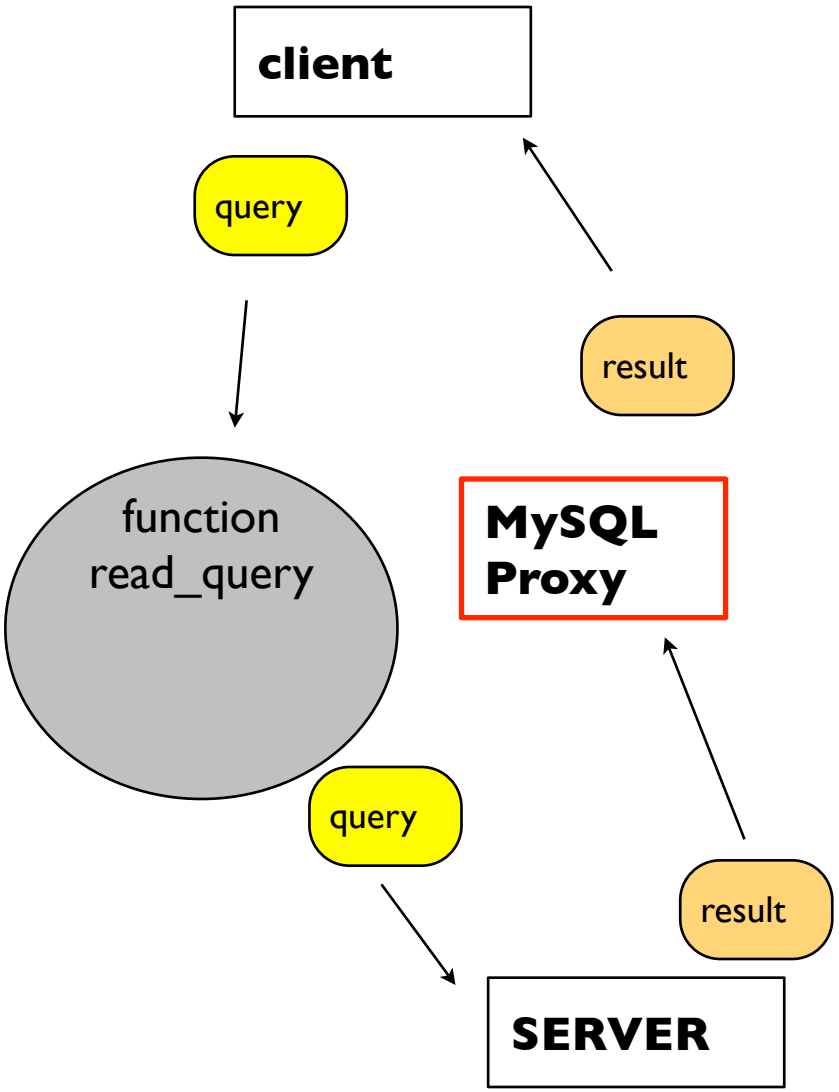
Client

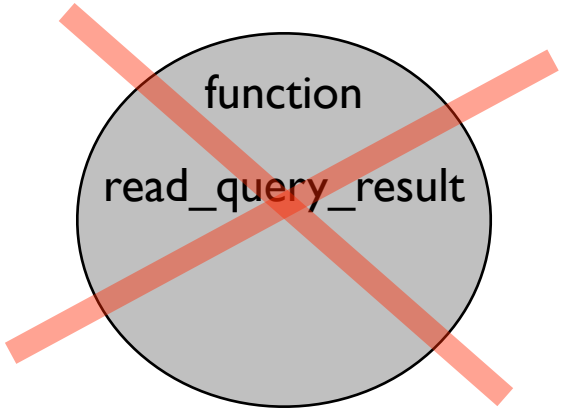
cookbook: share info among sessions

```
proxy.global.tot_q = proxy.global.tot_q or 0

function read_query (packet )
  if packet:byte() ~= proxy.COM_QUERY then return end
  proxy.global.tot_q = proxy.global.tot_q + 1
  print('queries ' .. proxy.global.tot_q)
end
```

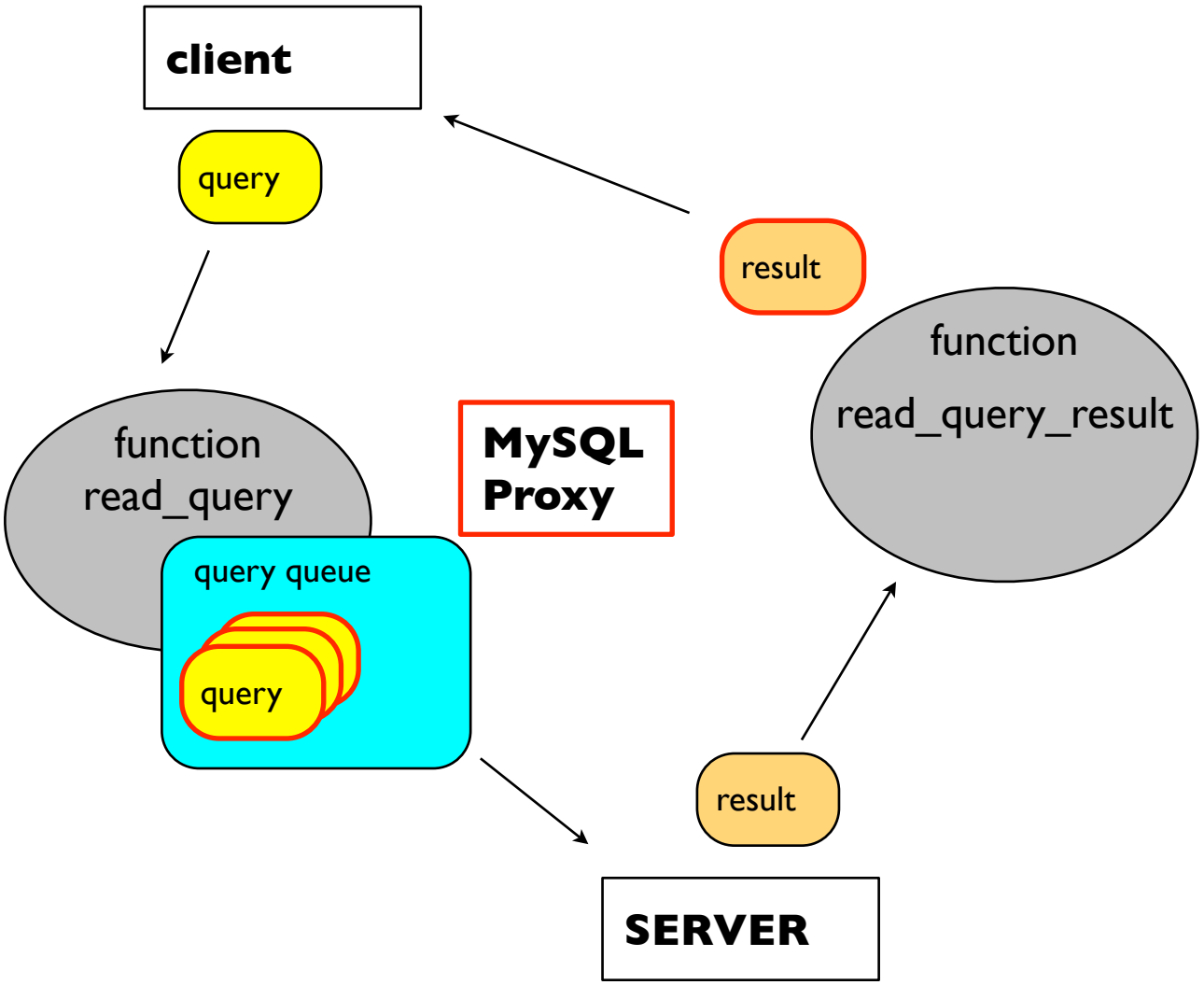
read_query and read_query_result





if a query is passed directly to the server, its result is **NOT** evaluated by read_query_result

read_query and read_query_result



only if a query is added to the **query queue**, its result is evaluated by read_query_result

multiple query execution - basic rule

Even if the proxy sends multiple queries to the server, it can return ONLY ONE RESULT to the client

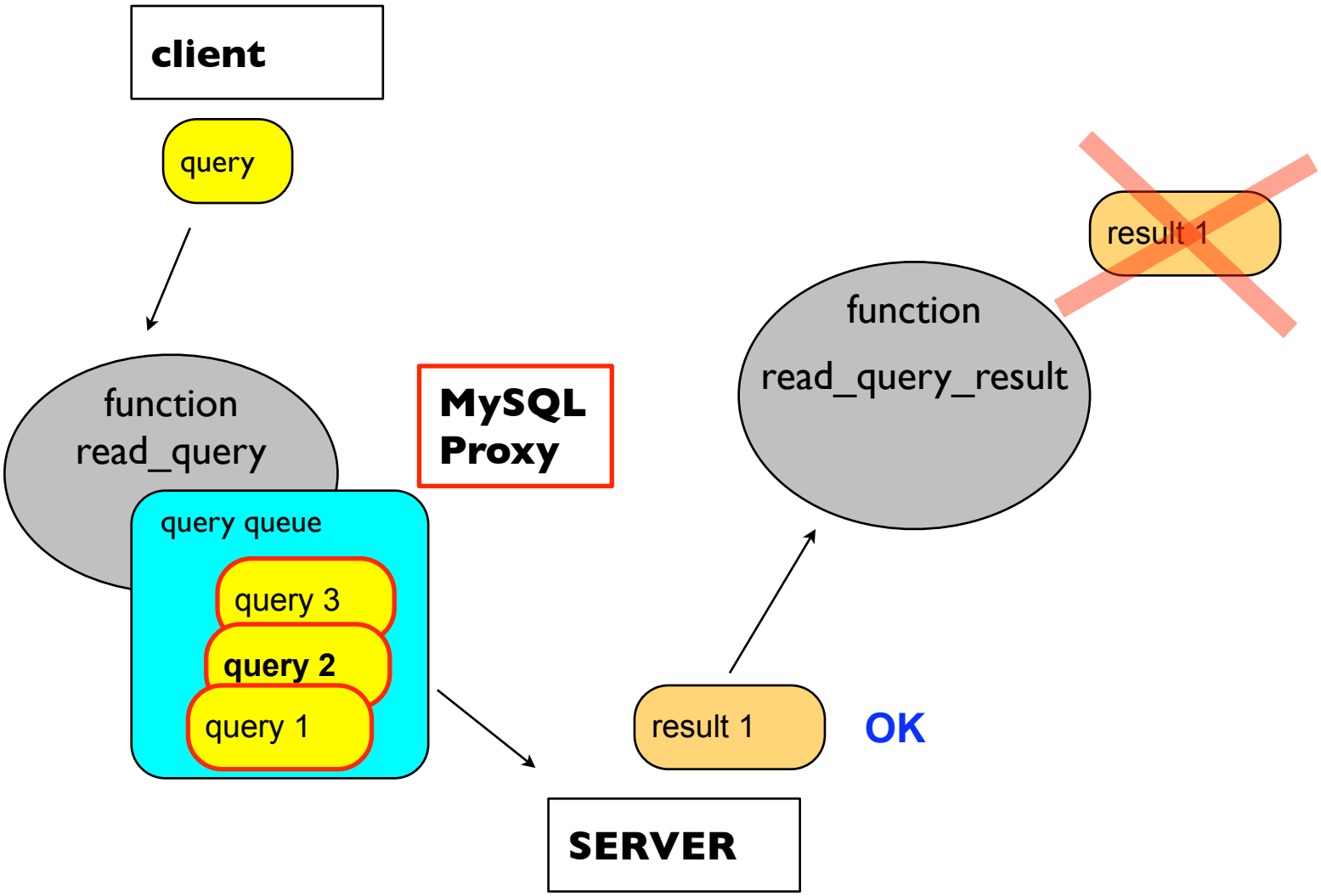
Multiple query execution

- simple queue
- queue management
- interactive play
- cumulative results

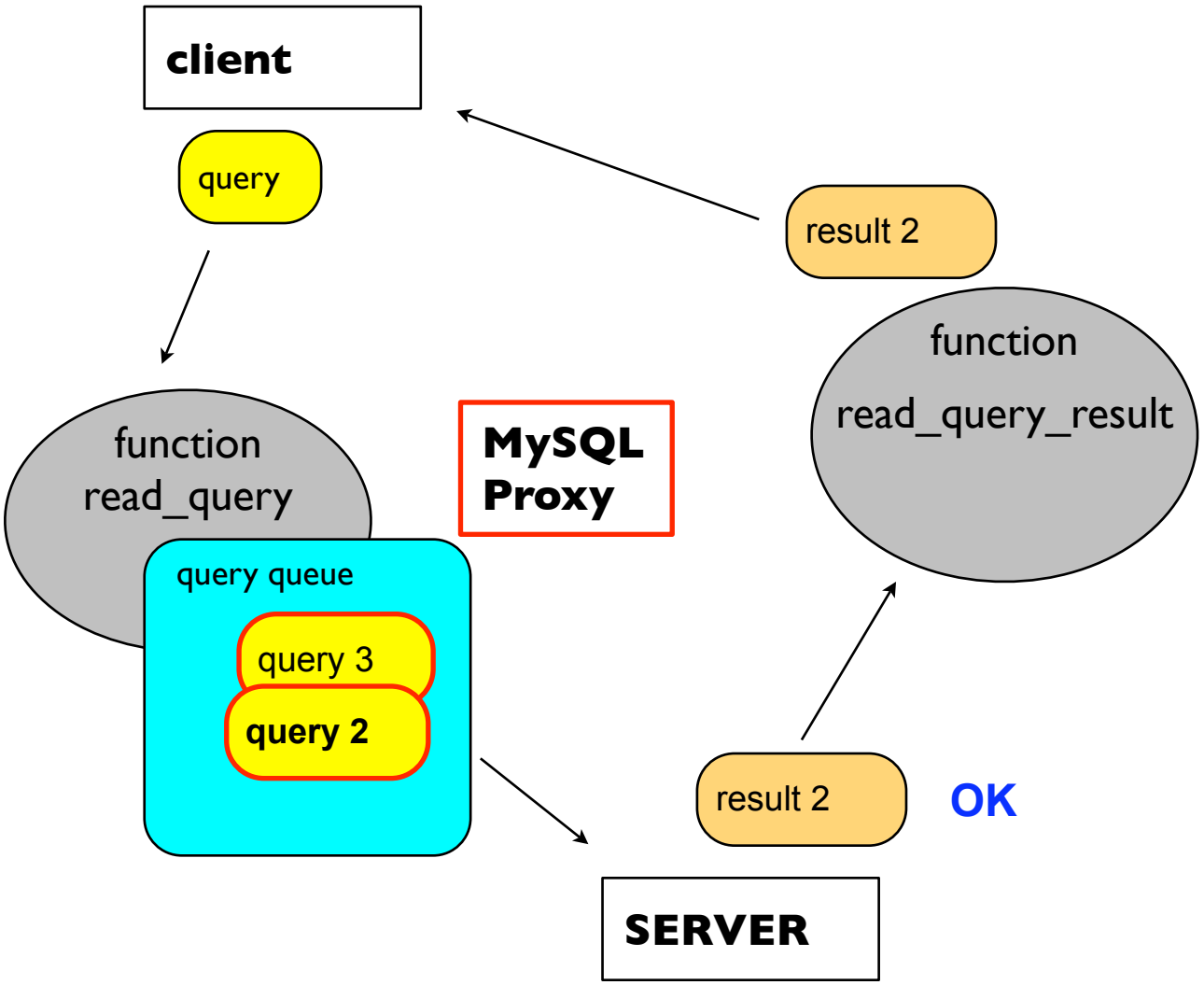
Multiple query execution - simple queue

- client sends one query
- Proxy adds 1 or more queries to the queue
- The Proxy processes each result
  - if the result is for the client, it is passed along
  - if the result is for internal calculation, it is discarded
- The client receives one result only

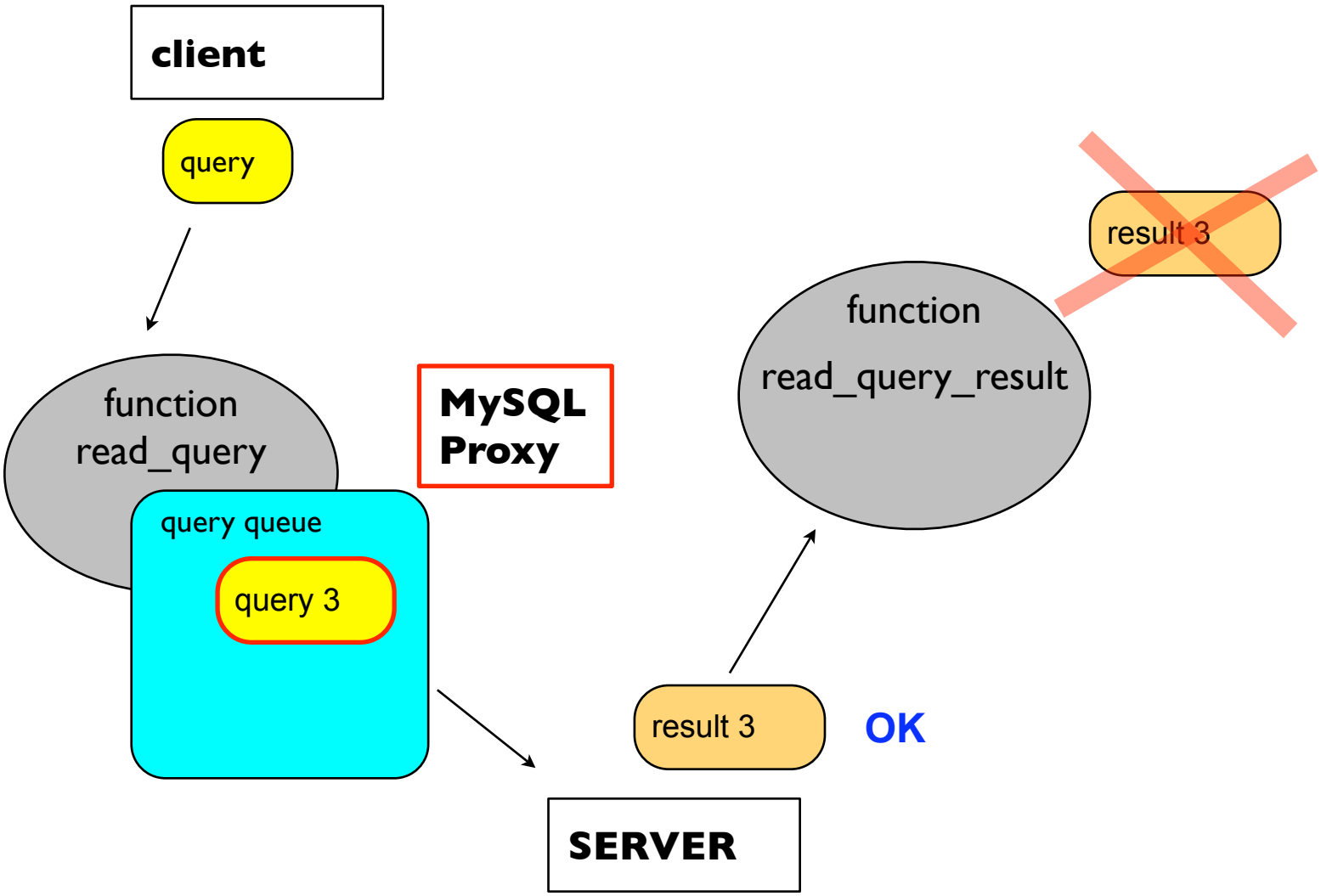
multiple query execution - 1



multiple query execution - 2



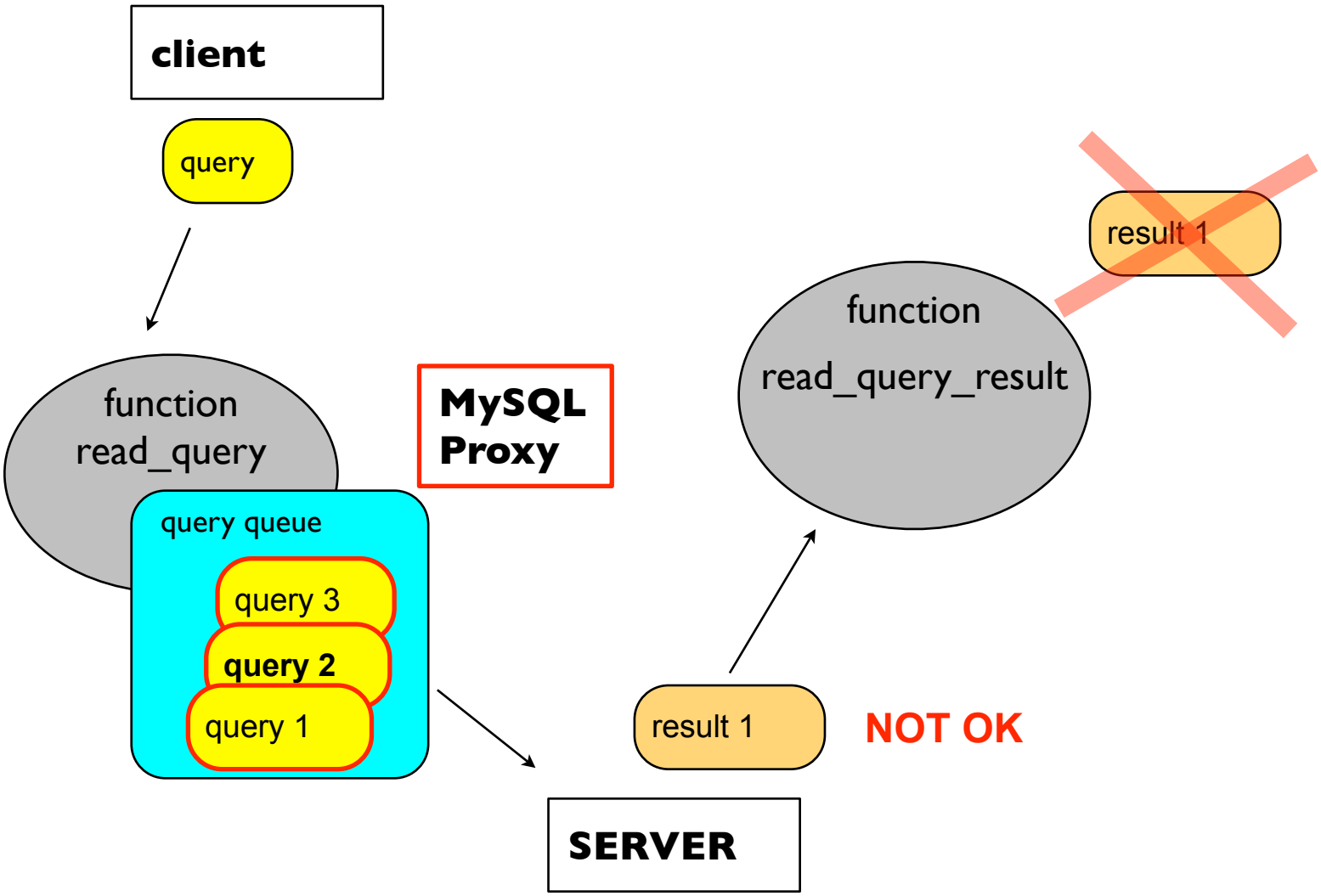
multiple query execution - 3



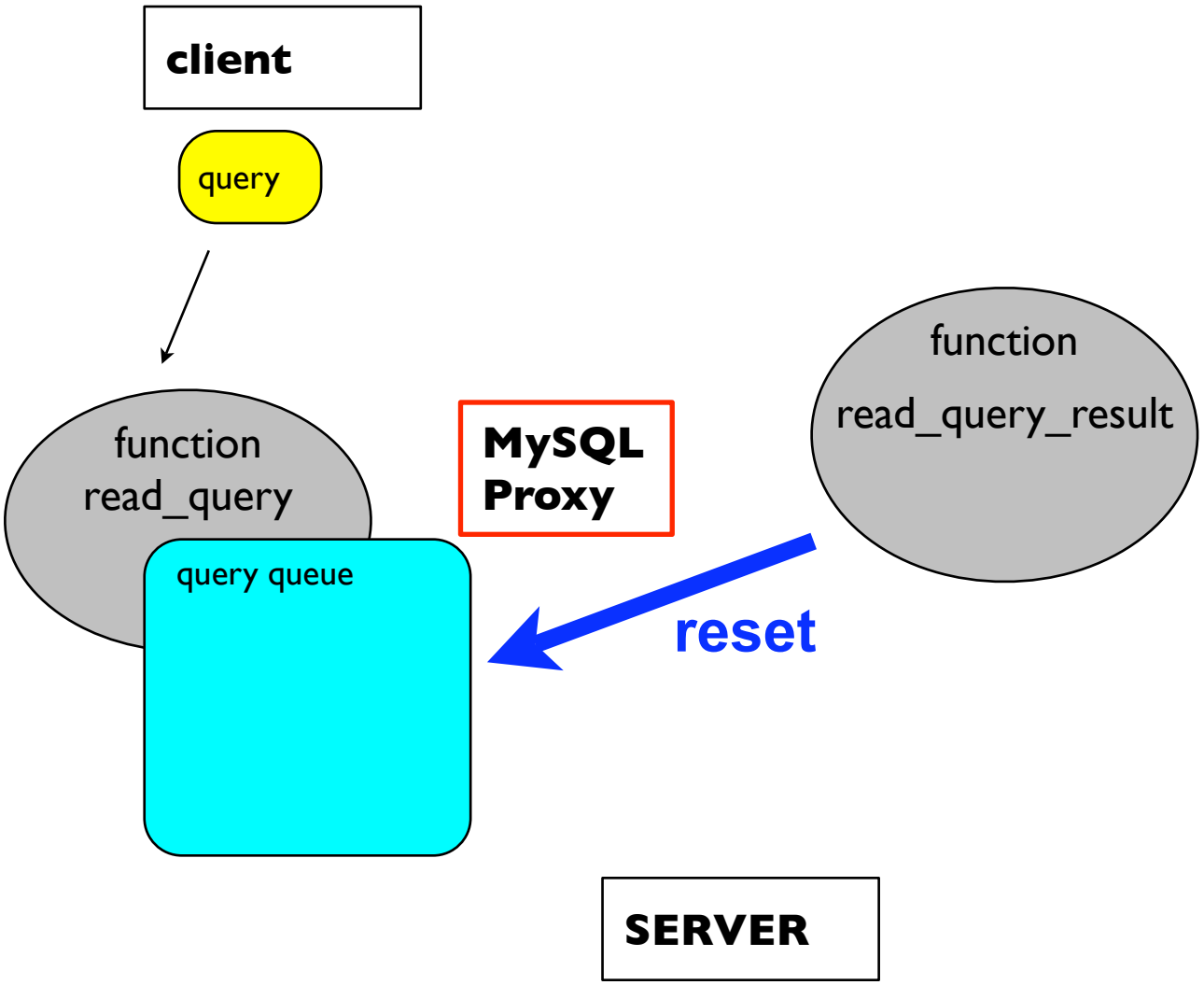
Multiple query execution - queue management

- the client sends one query
- the proxy adds N queries
- the proxy processes each result
  - if no error, the result is passed or discarded as needed
  - if error:
    - **clears the query queue**
    - an appropriate result is passed to the client
- The client receives one result

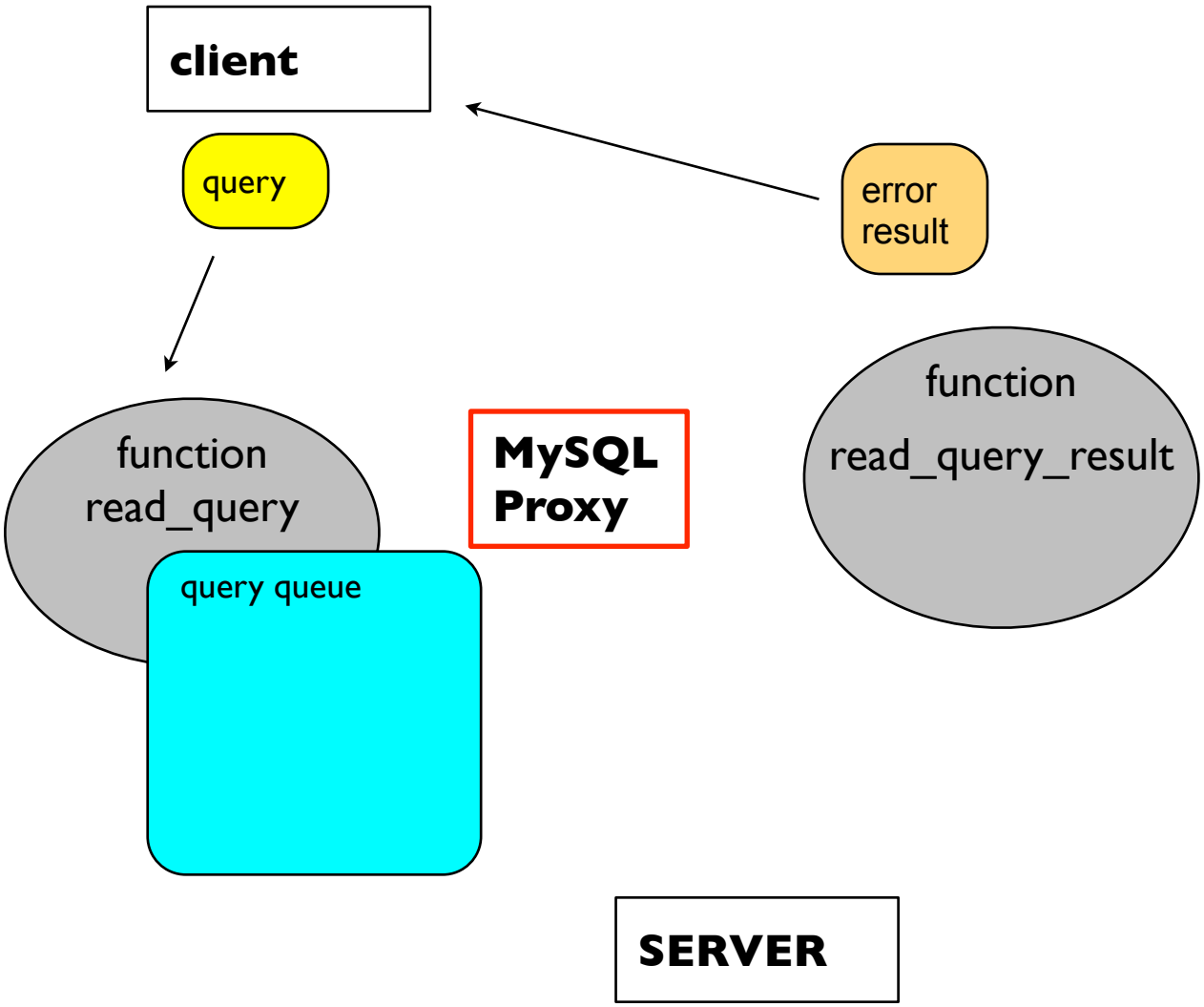
queue management - 1



queue management - 2



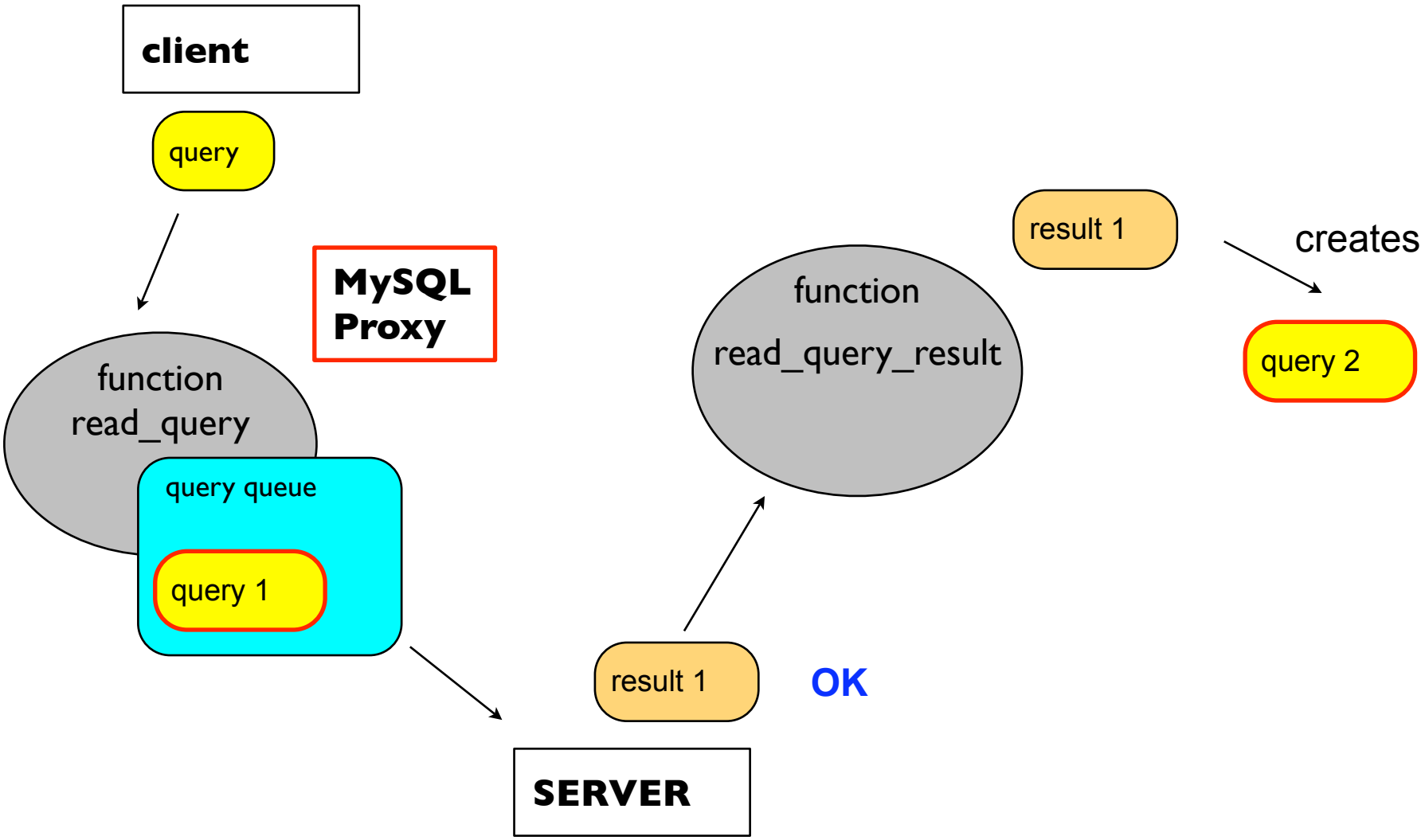
queue management - 3



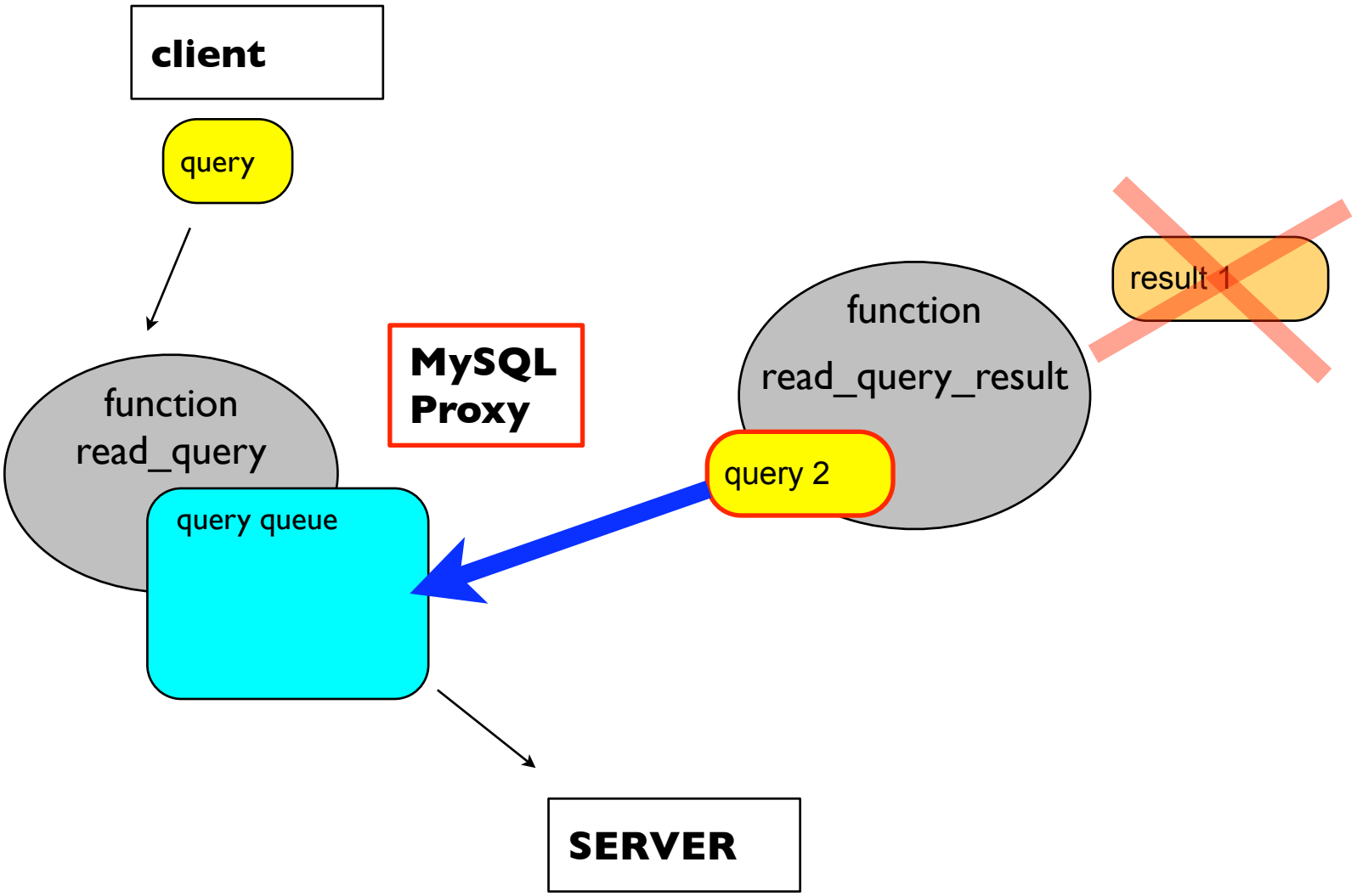
Multiple query execution - interactive play

- The client sends a query
- the query is sent through the query queue
- the proxy **uses the result** to **create** a new query
- the proxy **inserts the new query** to the queue
- the first result is discarded
- the next result is passed to the client
- the client receives only one result

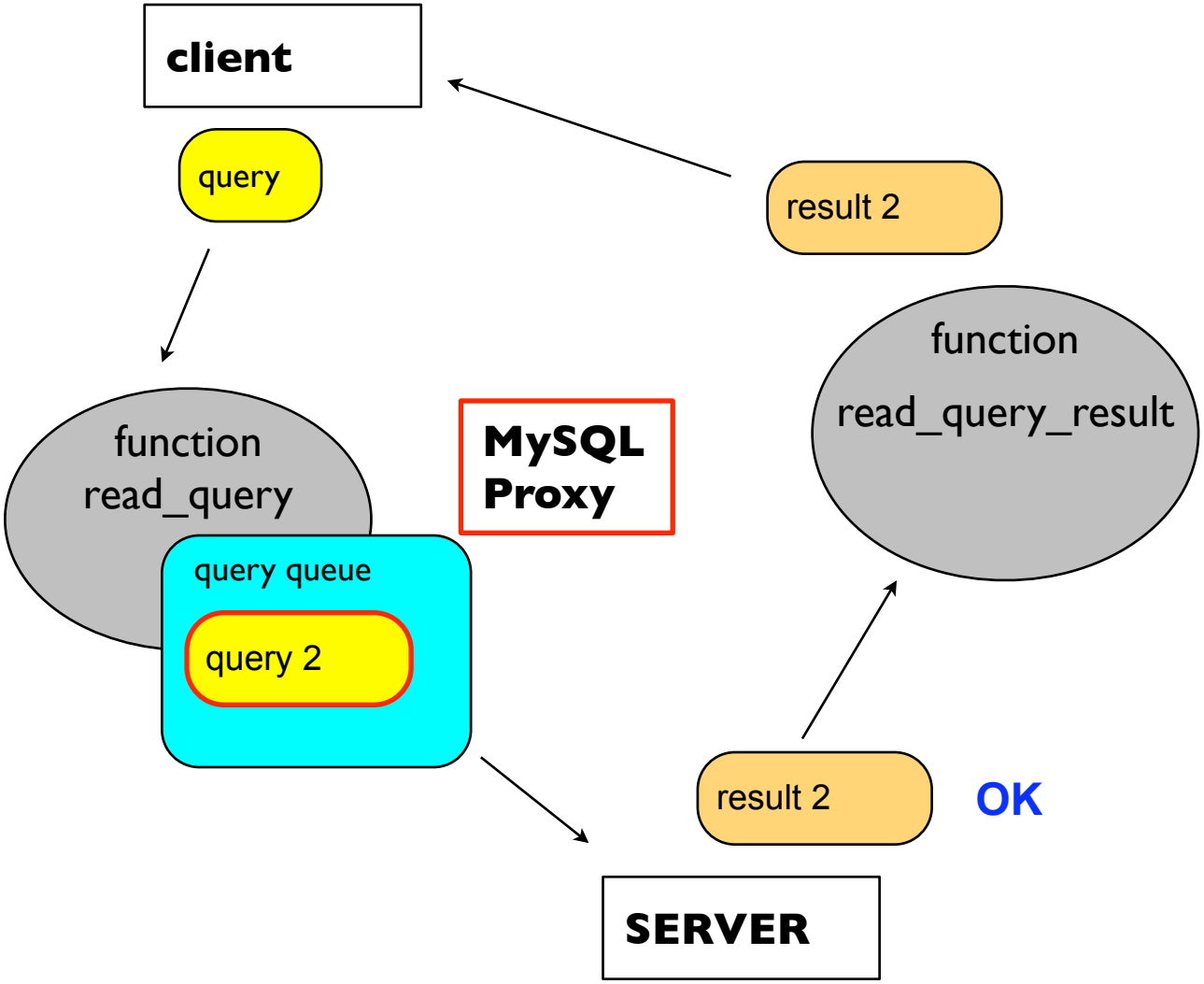
interactive execution - step 1



interactive execution - step 2



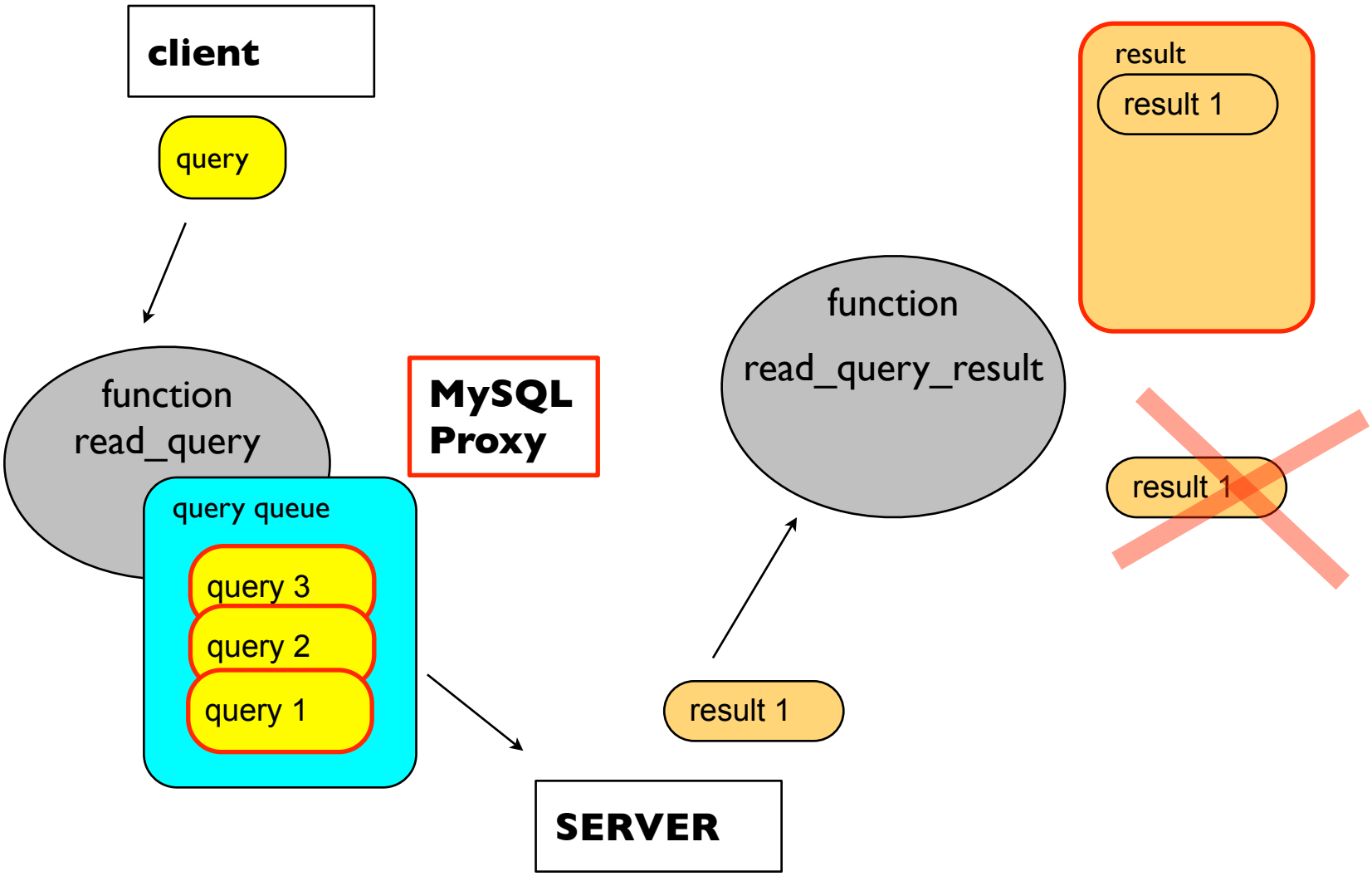
interactive execution - step 3



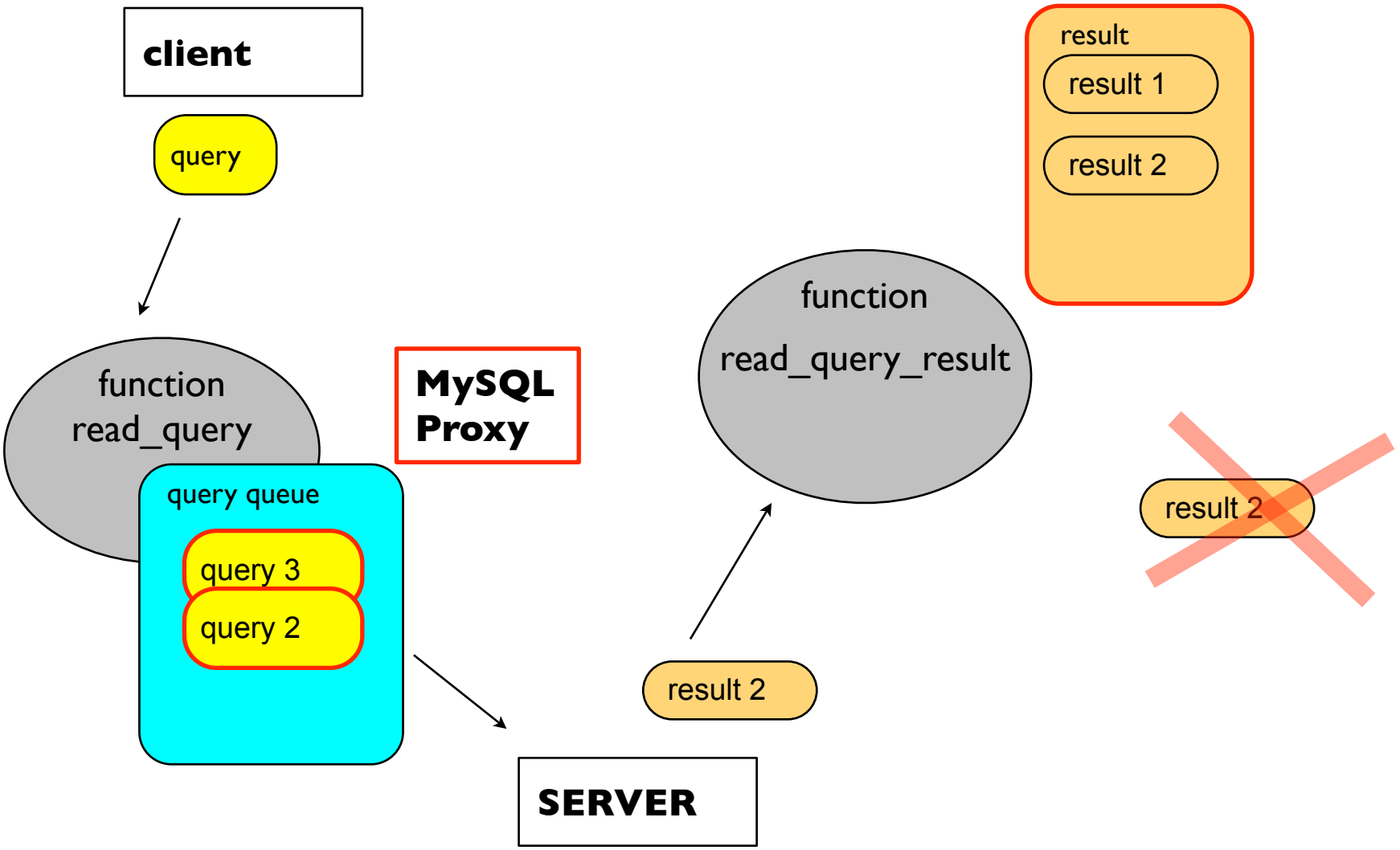
Multiple query execution cumulative result

- The client sends a query
- the proxy adds N queries to the queue (e.g. loop)
- each result is added to a cumulative
- the client receives only one (big) result

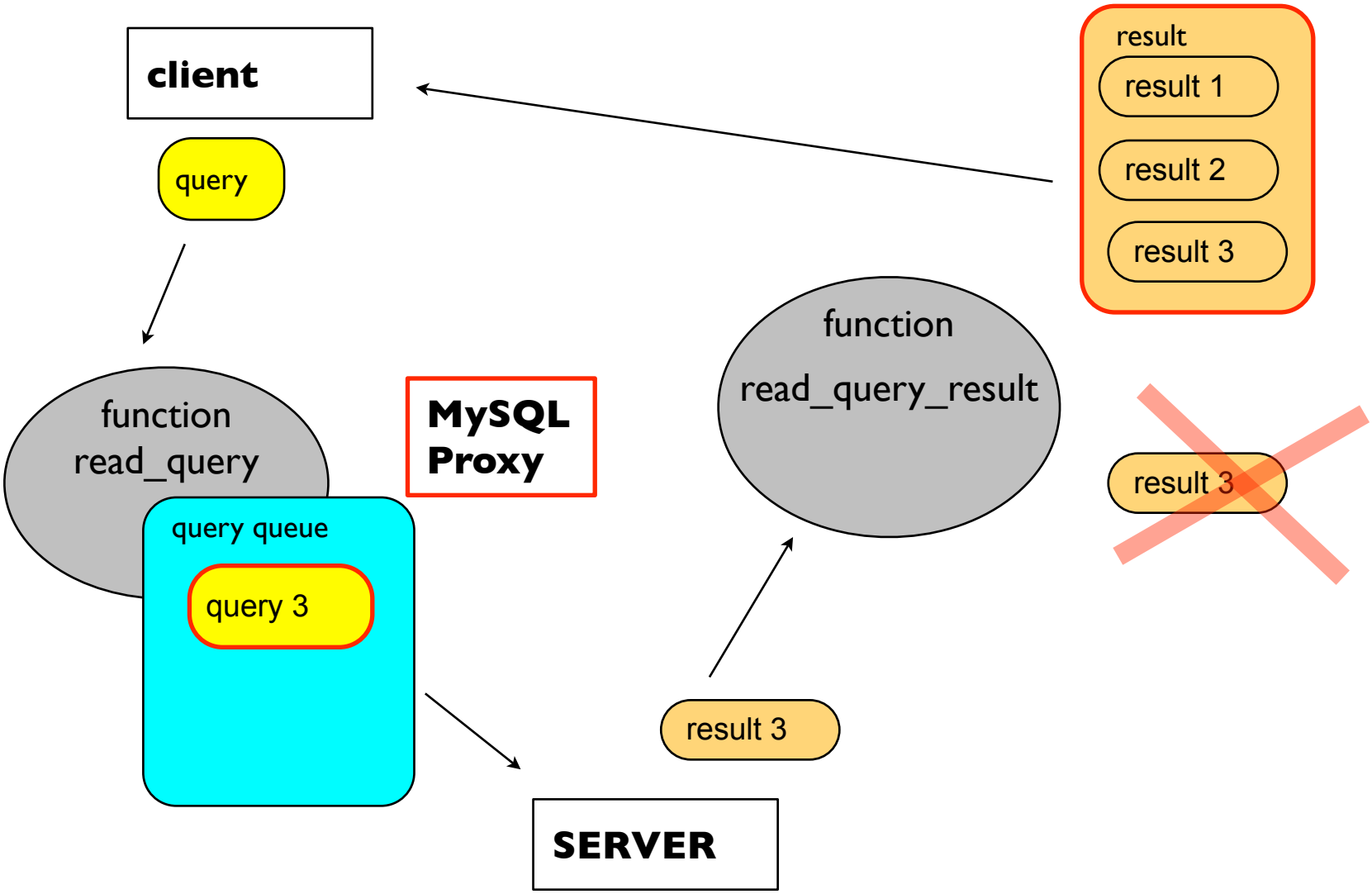
cumulative result - 1



cumulative result - 2



cumulative result - 3



Q&A

Let's talk!