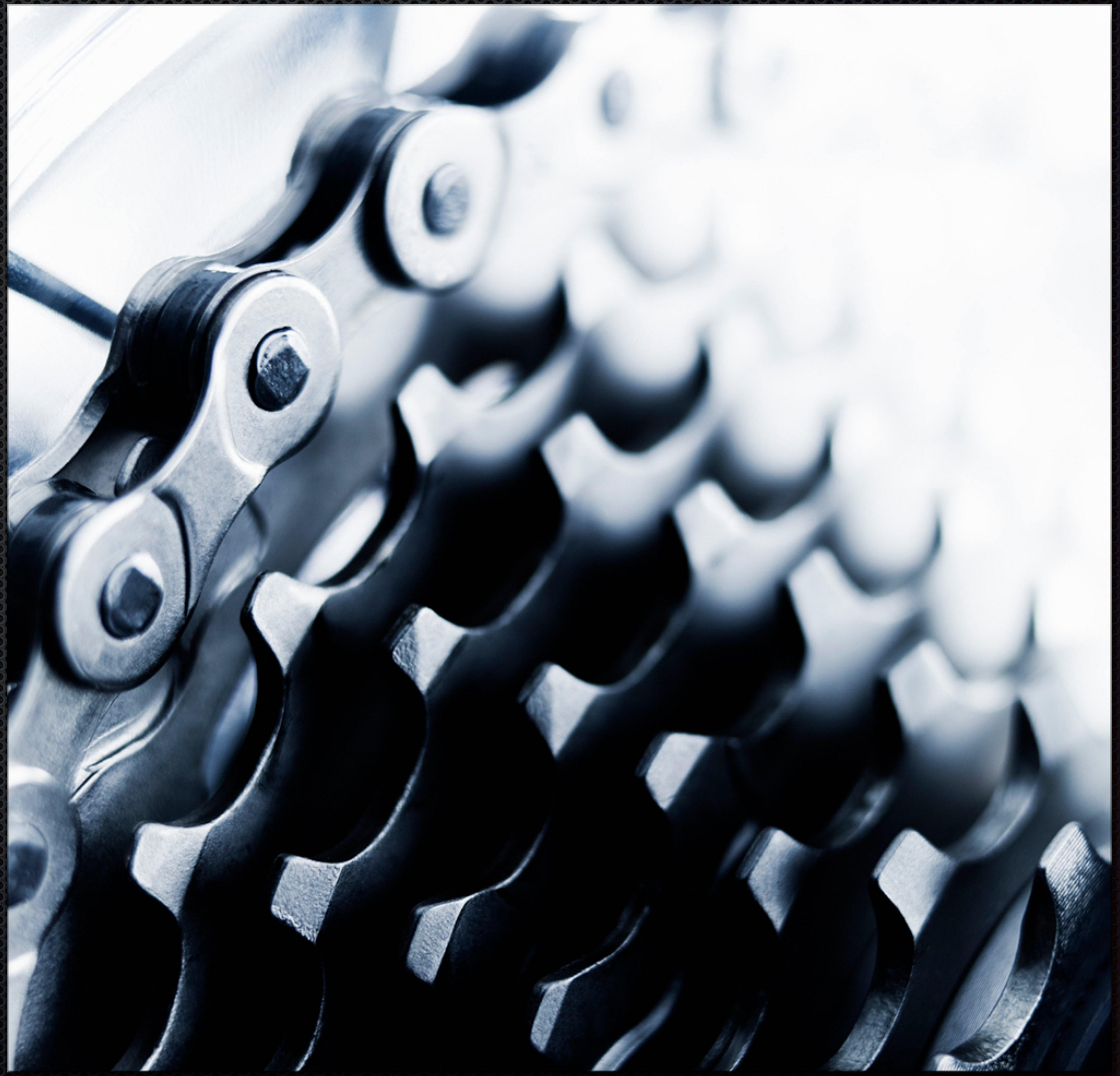
ProxySQL's Internals

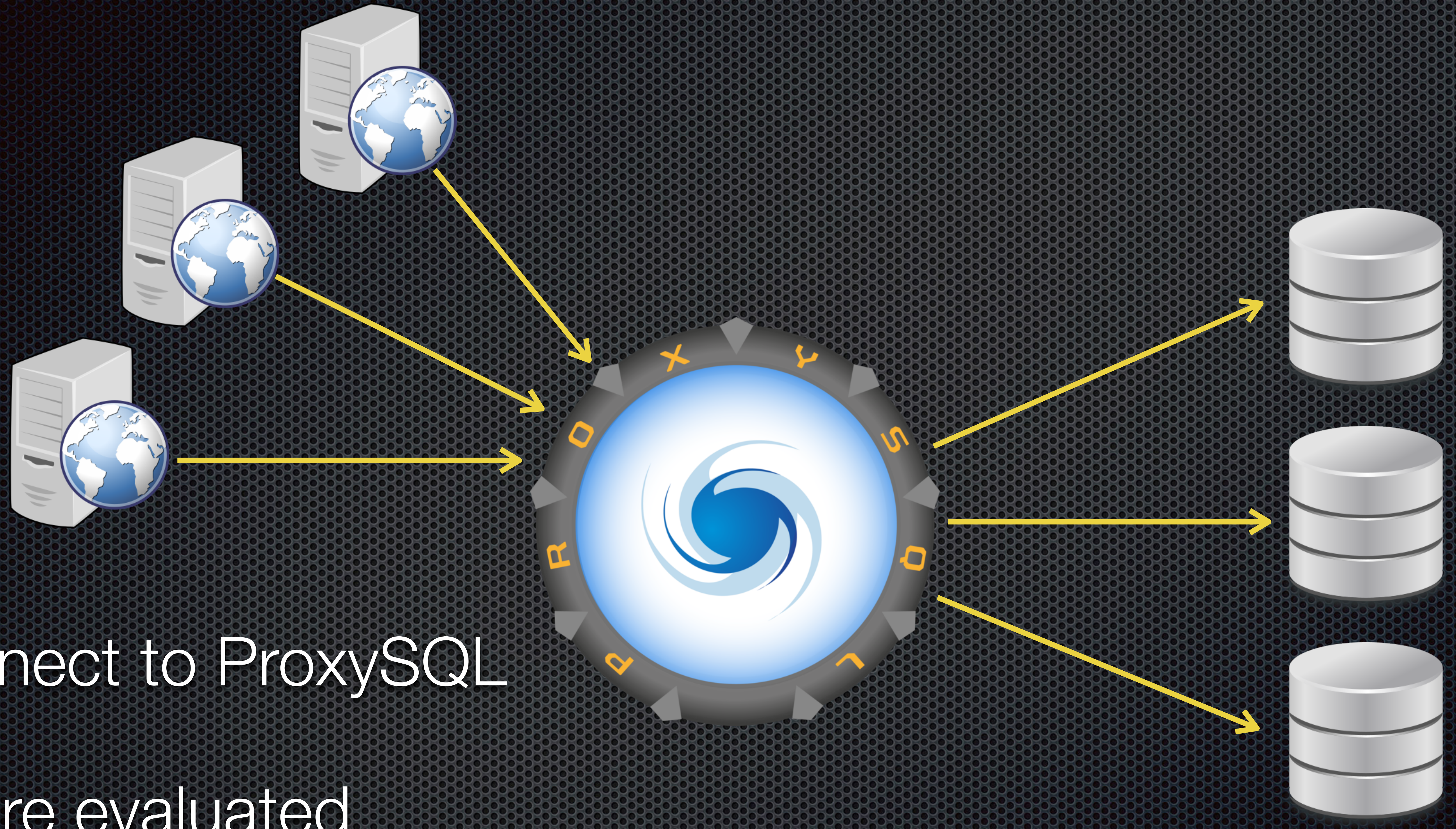
Implementation details on handling millions of connections and thousands of servers

What is ProxySQL?

- A "Layer 7" database proxy
- MySQL / ClickHouse protocol aware
- High Performance
- High Availability



Architecture Overview



- Clients connect to ProxySQL
- Requests are evaluated
- Actions are performed

High Performance

- Maximize throughput
- Reduce latency
- Scale

>> Built to handle hundreds of thousands of connections

>> Built to handle thousands of backend servers

Threading Models

- One thread per connection
  - Easier to develop
  - Blocking I/O
- Thread pooling
  - Non blocking I/O
  - Scalable

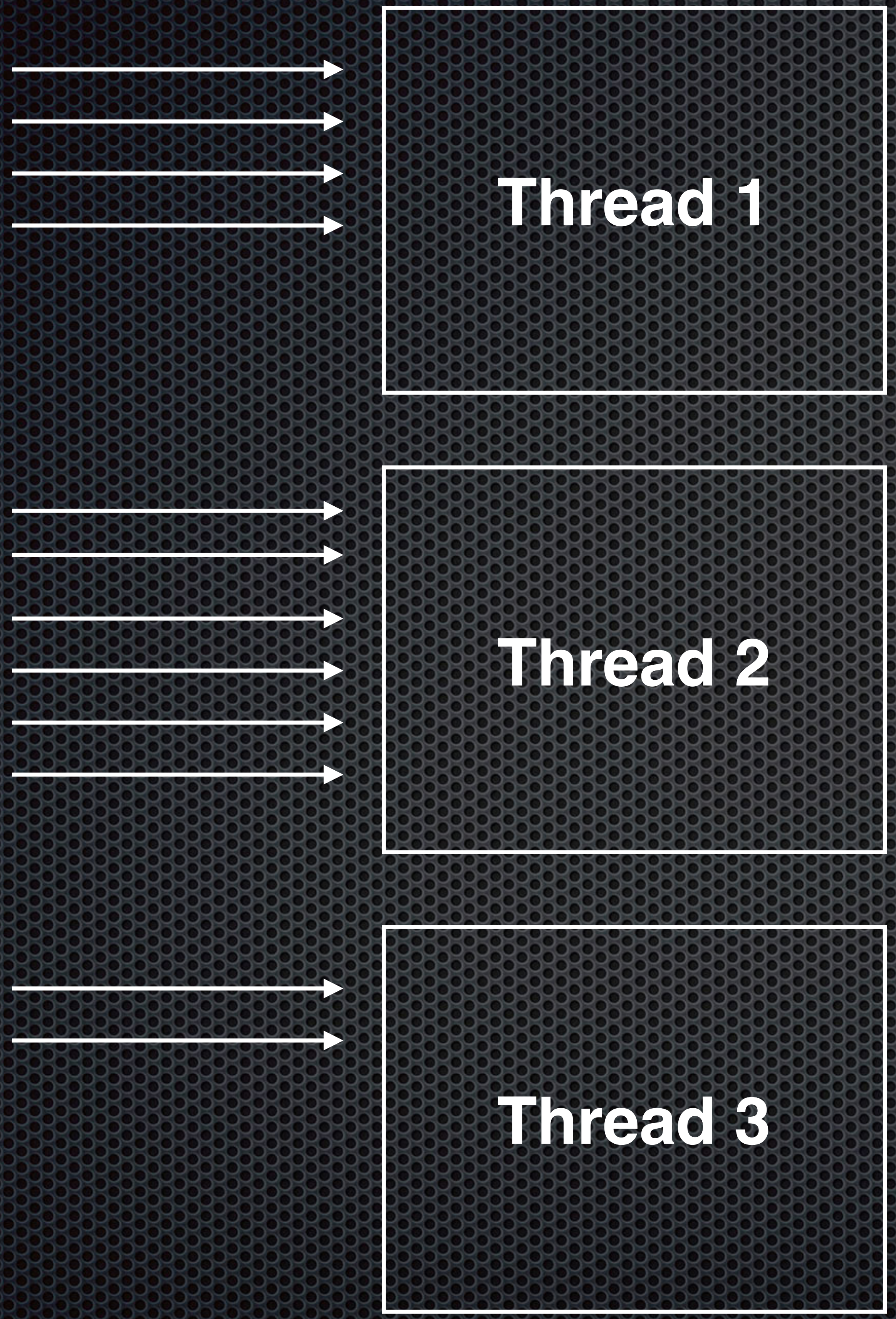
Common Thread Pool Implementations

- One thread accepts connections
- Connections are passed to worker threads
- One or more threads perform network I/O
- I/O queuing occurs
- Fixed or dynamic number of worker threads

ProxySQL's Thread Pool Implementation

- Threads in ProxySQL are known as "*MySQL Threads*"
- Fixed number of worker threads (configurable)
- All threads listen on the same port(s)
- Client connections are not sharded between threads
- All threads perform their own network I/O
- Uses "**poll()**"... (does it scale?)

Threads never share client connections



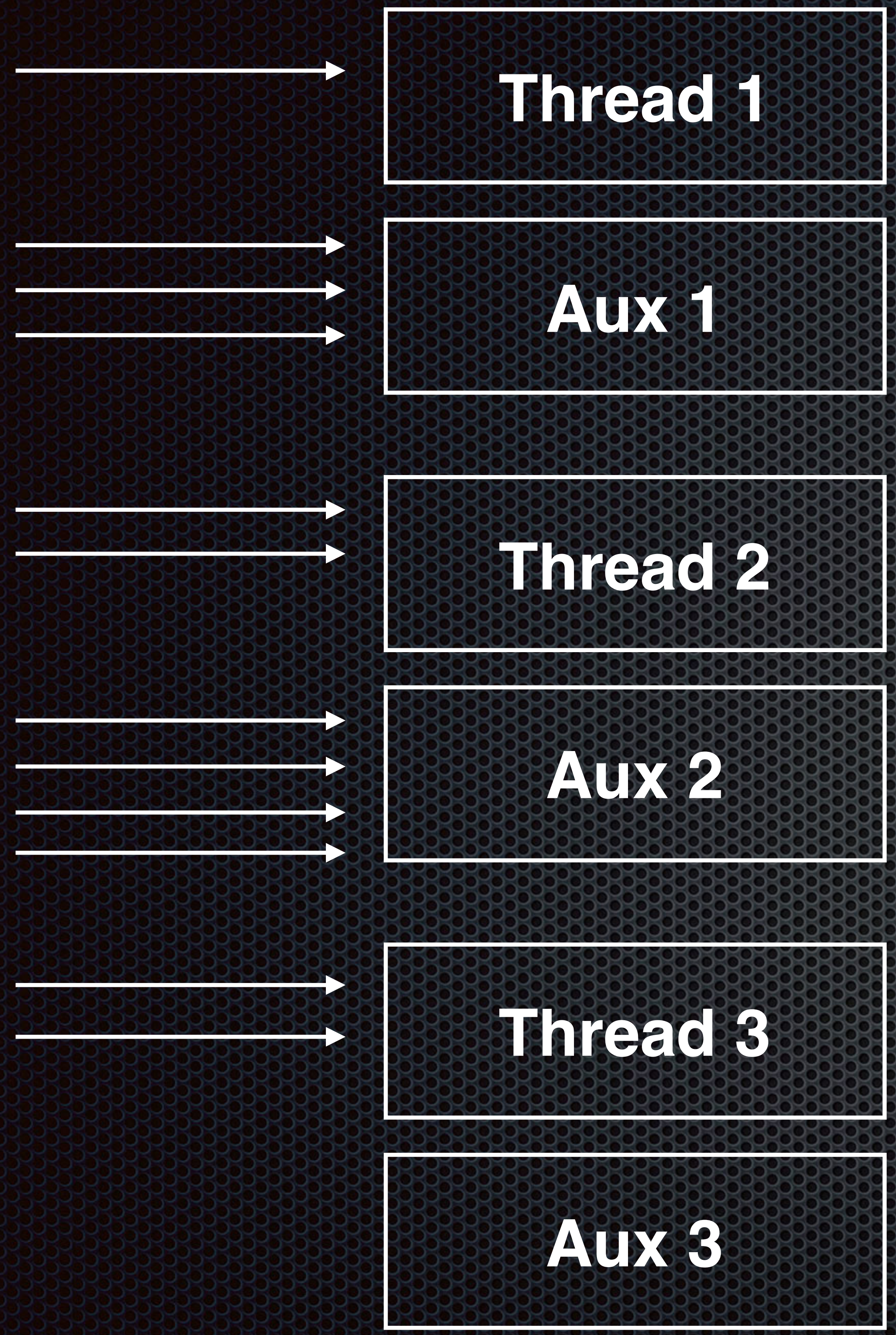
- Pros:
  - Thread contention is reduced
  - No need for synchronization
  - Each thread calls "poll()"
- Cons:
  - Possibly imbalanced load

poll() vs. epoll()

- "poll()" is O(N)
- "epoll()" is O(1)
- "epoll()" scales better than "poll()"

- Why does ProxySQL use "poll()"?
  - It is faster than "epoll()" for fewer connections (~1000)
  - Performance degrades when there are a lot of connections

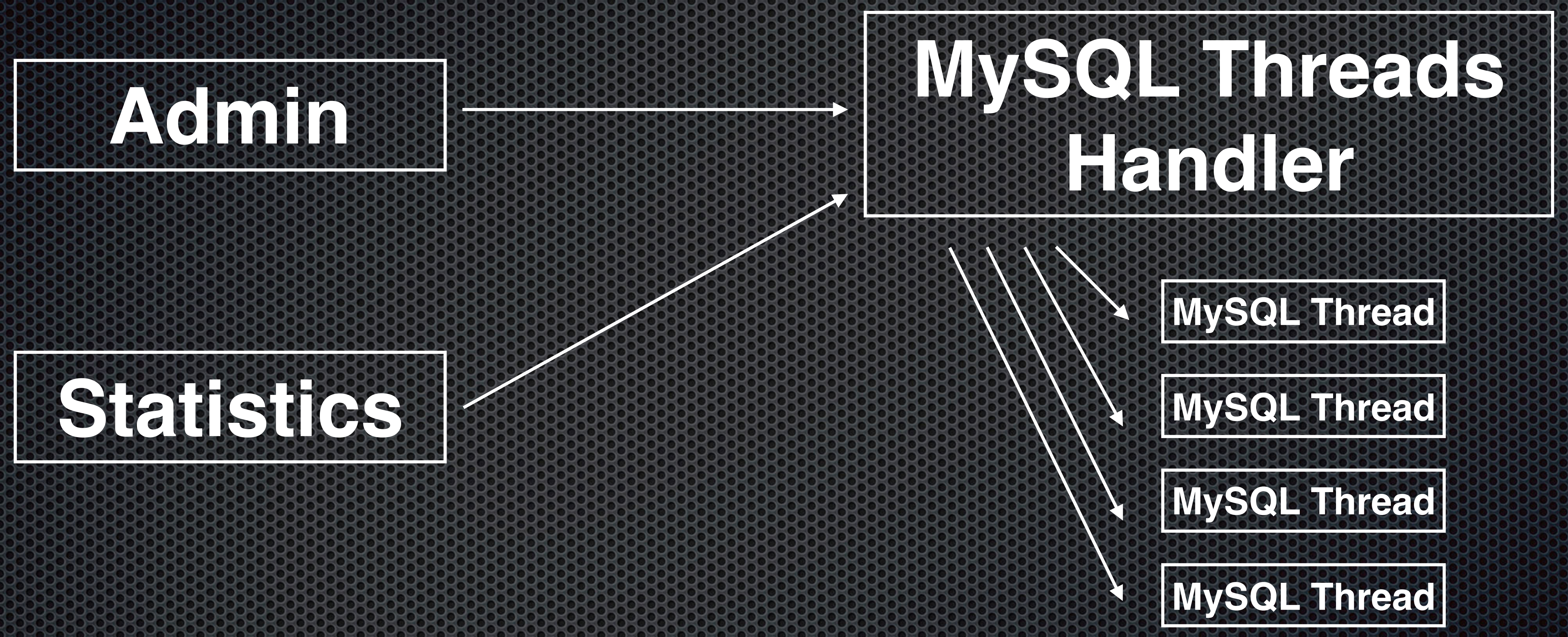
ProxySQL Auxiliary Threads



- Each worker thread has an auxiliary thread
- Worker thread uses "poll()"
- Auxiliary thread uses "epoll()"
- Worker thread passes idle connections to auxiliary thread
- When a connections becomes active auxiliary thread passes connection to the worker thread

Solution scales to 1 million connections

MySQL Threads Handler



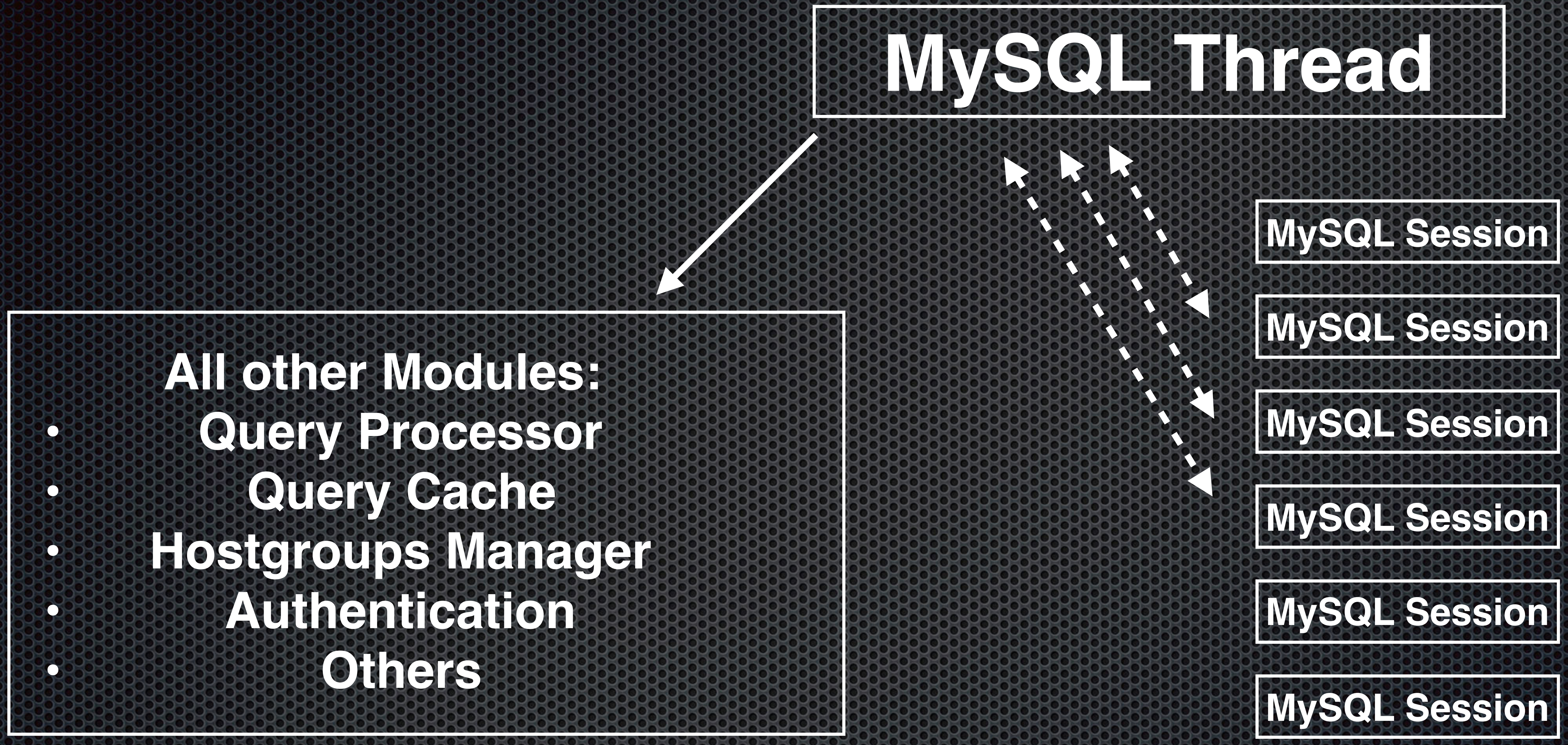
MySQL_Threads_Handler()

A set of functions to simultaneously control the MySQL Threads, for example:

- Starting threads
- Stopping threads
- Getting metrics by atomic operations
- Getting metrics by locking

Used mostly by ProxySQL Admin and ProxySQL Statistics modules

MySQL Thread Overview



All other Modules:

- Query Processor
- Query Cache
- Hostgroups Manager
- Authentication
- Others

* Every object has a pointer to its parent

MySQL_Thread()

- Represent a **worker thread**
- Accepts new connections and creates MySQL Sessions
- Processes MySQL Sessions
  - Performs network I/O
  - Interacts with other modules: Admin, Authentication, Query Cache, Query Processor, Connection Pool, Hostgroups Manager, Prepared Stmt. Manager, etc.

MySQL_Thread()

For low contention, threads independently:

- Track internal metrics
- Store values for mysql-XXX variables
- Store a **copy** of the defined query rules

MySQL_Session()

- Represents a client connection / session
- Created when a client connects to ProxySQL
- Implemented as a state machine
- Stores metadata associated with the client session:
  - Running timers
  - Transaction persistence
  - Mirroring
  - Default Hostgroup, etc.
- A **"virtual / internal"** session can also be created for pinging backends and mirroring traffic

MySQL_Data_Stream()

Abstraction on top of the network socket

- Reads data from network and generate packets
- Converts packets into data to be written into sockets
- Transparently handles compression, encryption and decryption

Mostly useful for frontend connection

- Used for backends in versions prior to the introduction of the MariaDB Client Library
- Also used for backend connection in fast forward mode

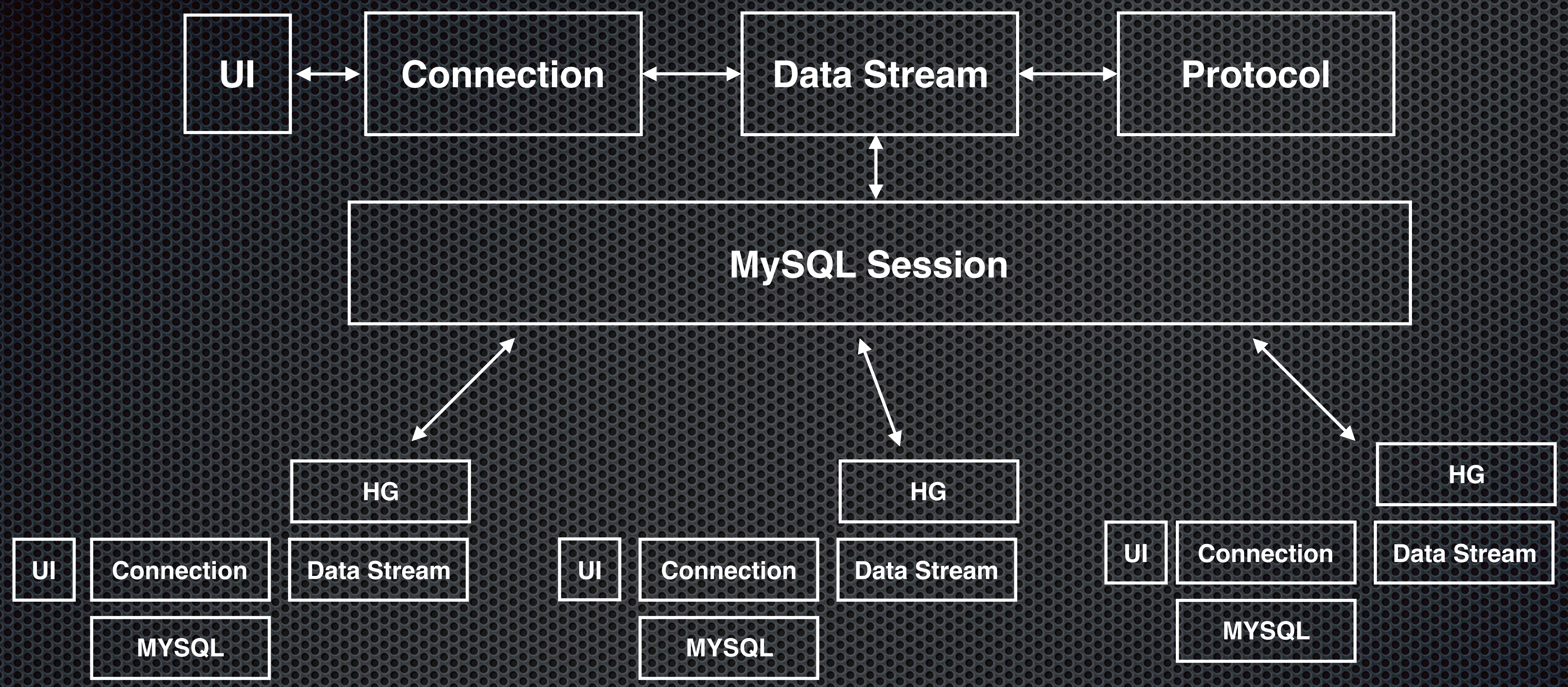
MySQL_Protocol()

- Associated with a MySQL_Data_Stream
- Generates packets to be sent to the client:
  - Handshake packets
  - OK, ERR, EOF packets
  - Resultset (rows, fields, etc)
  - PREPARE_RESPONSE
- Also performs input validation

MySQL Connection

- Stores metadata related to a MySQL connection - MySQL_Connection_userinfo():
    - username, schema name, current schema, time_zone, sql_mode, autocommit, statuses, etc.
- For backend connections it is also a wrapper to all the functions of the MariaDB Client Library

MySQL Session Overview

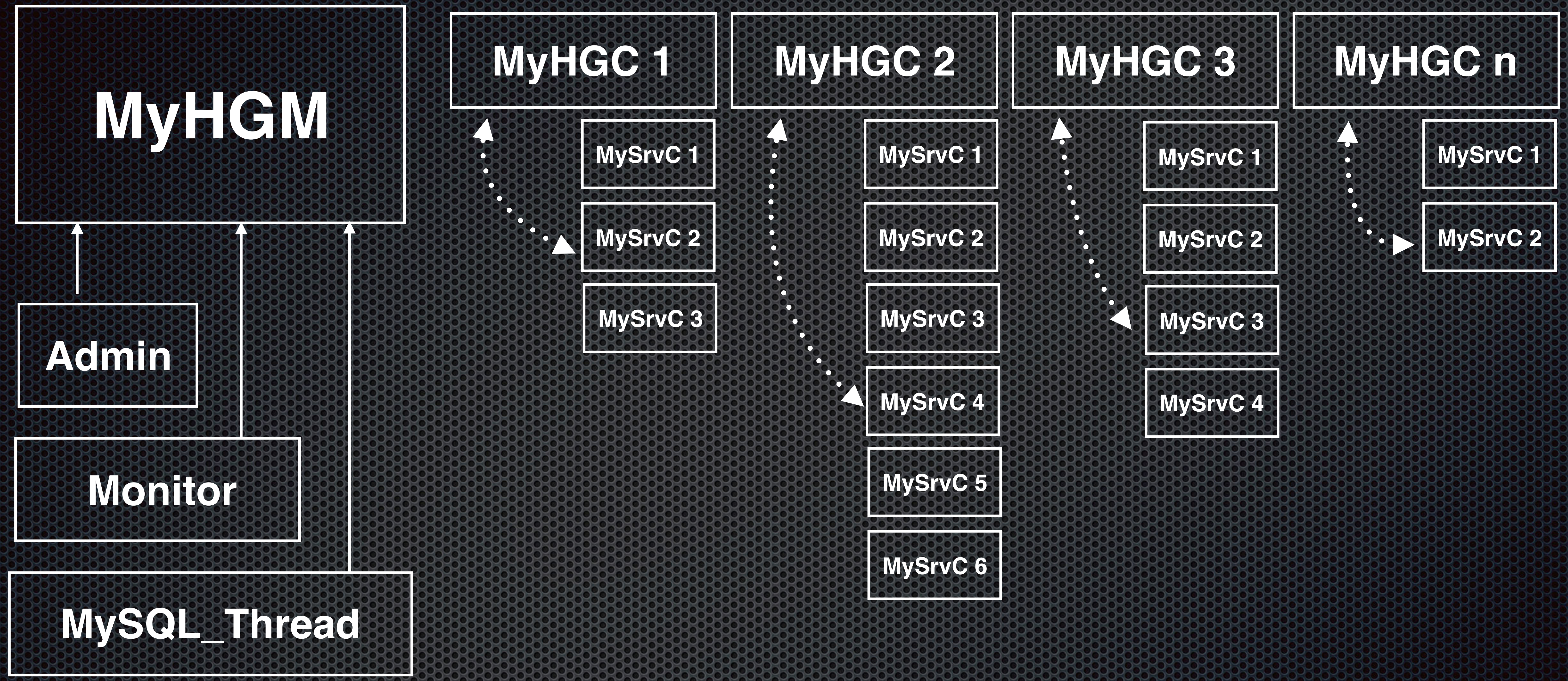


- Every object has a pointer to its parent

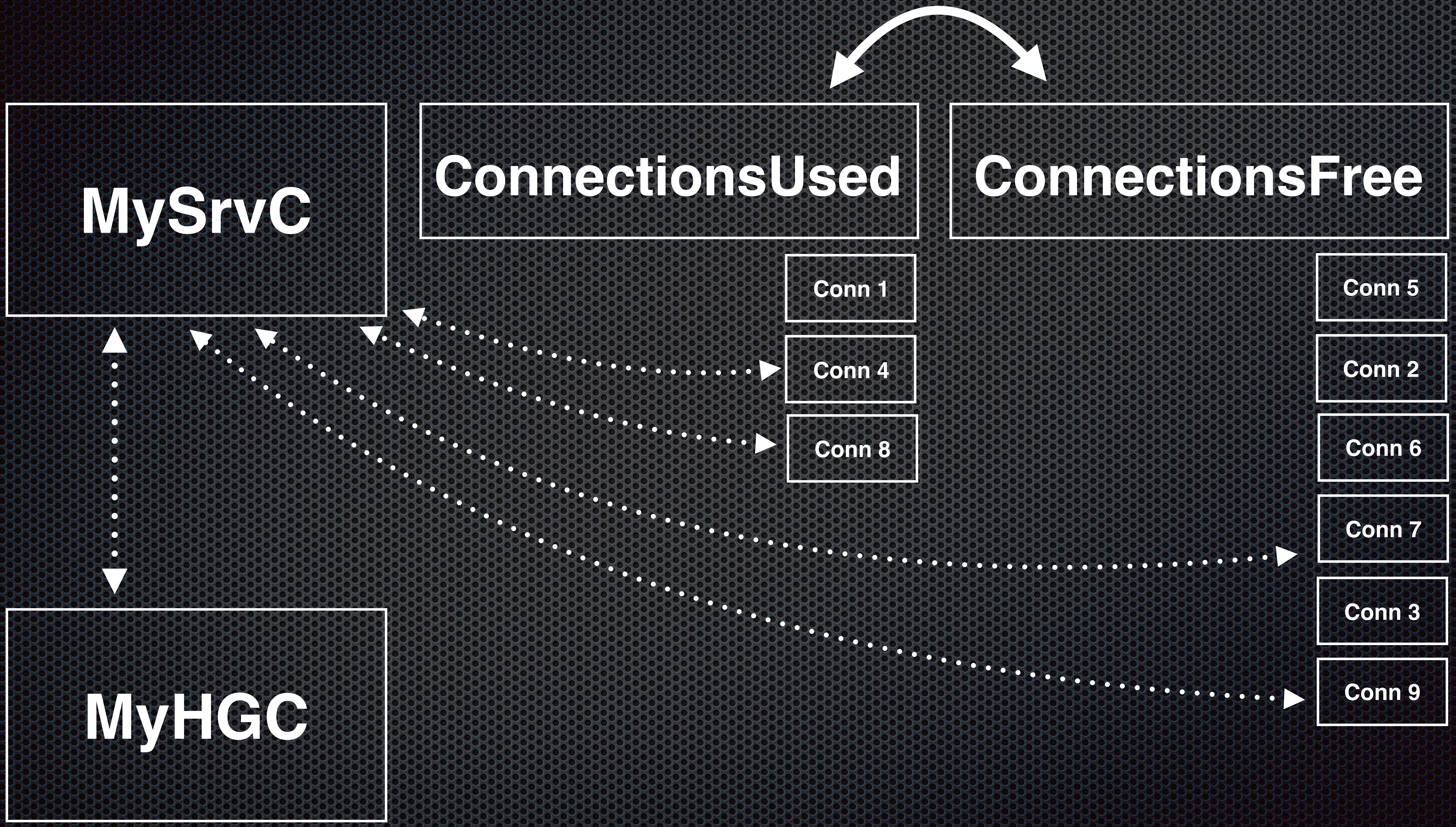
MySQL_Hostgroups_Manager()

- Manages **hostgroups, servers** and **connections**
- Used by **MySQL_Threads, MySQL_Connection, Admin, MySQL_Monitor** and **Statistics** to:
  - Get or return connections
  - Get the status of servers
  - Reconfigure hostgroups and servers
  - Get or set metrics

MySQL_Hostgroups_Manager()



MySrvC() - MySQL Server



Get Connection

- Identify hostgroup
- Get a random server based on **weight**
- Get a random connection from **ConnectionsFree**
- Move the connection to **ConnectionsUsed**
- Attach the connection to **MySQL_Data_Stream**

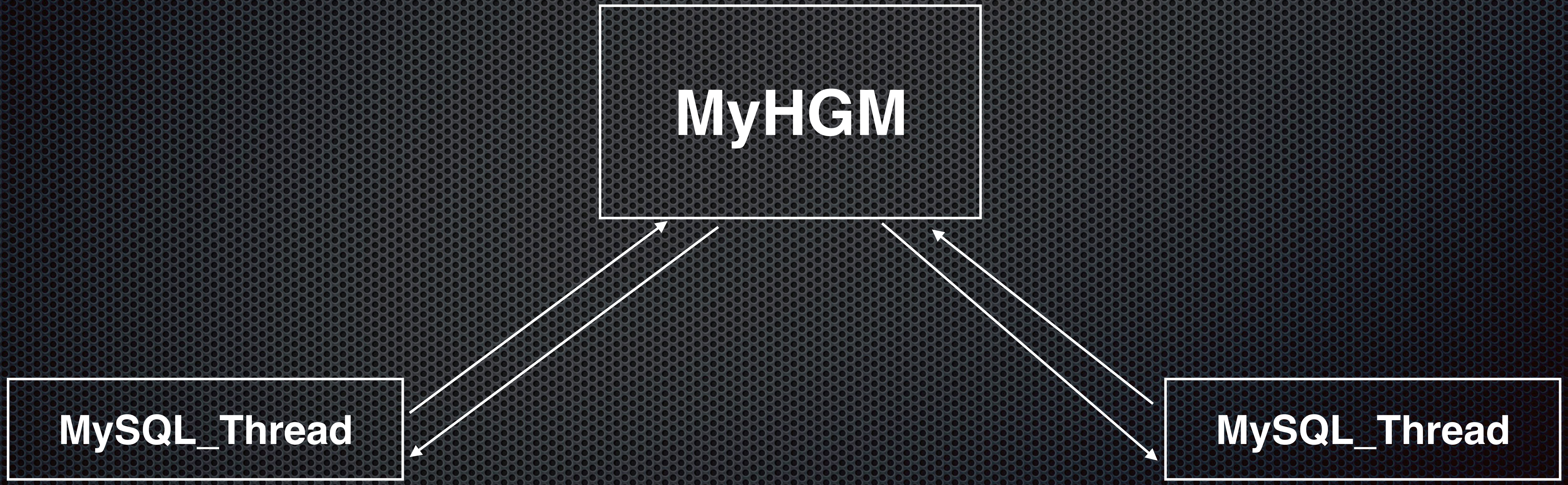
If a no connections exist yet then a new MySQL Connection object is created without a socket connection. MySQL Thread will then establish a new socket connection

Return Connection

- Detach the connection from MySQL Data Stream
- The pointer to MySrvC allows to immediately return the connection to the right server
- Find the connection in ConnectionsUsed and move it to ConnectionsFree

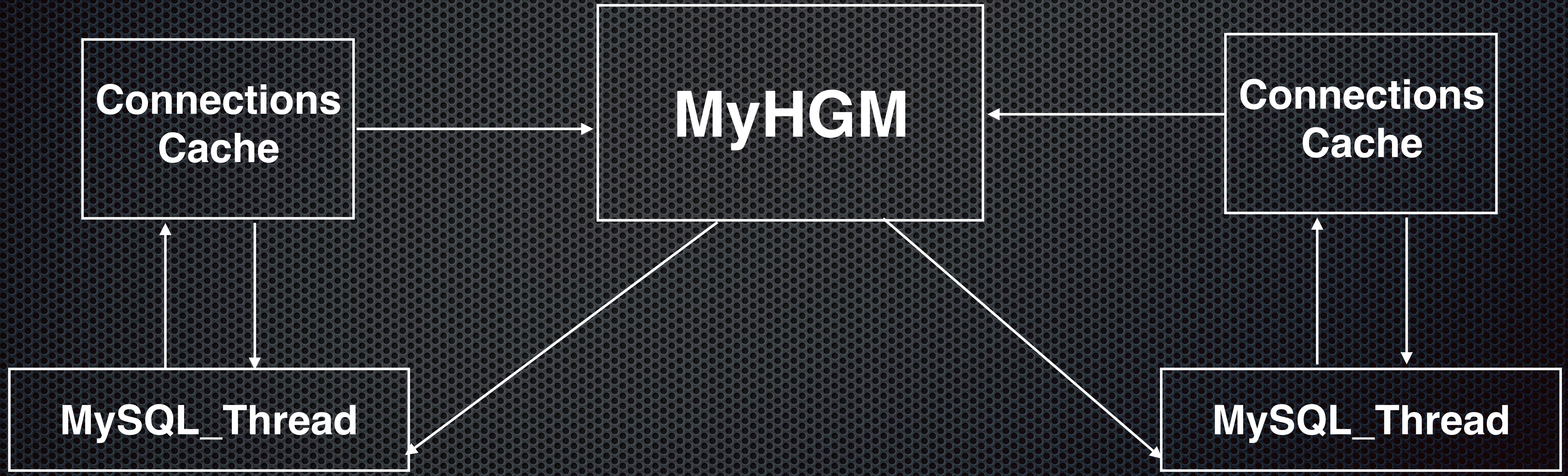
Contention on MyHGM

- MyHGM is a shared resource so it can cause **contention** when accessed by **MySQL Threads**



Thread Connection Cache

- Each MySQL Thread has a connection cache that is reset before calling poll()



Thank you!

- Please remember to report feature requests and bug reports: https://github.com/sysown/proxysql/
- Community support can be found on our forum: https://groups.google.com/forum/#!forum/proxysql
- Useful blog articles are available at our site: http://proxysql.com/blog
- Visit us at http://proxysql.com/support for subscription and support options