

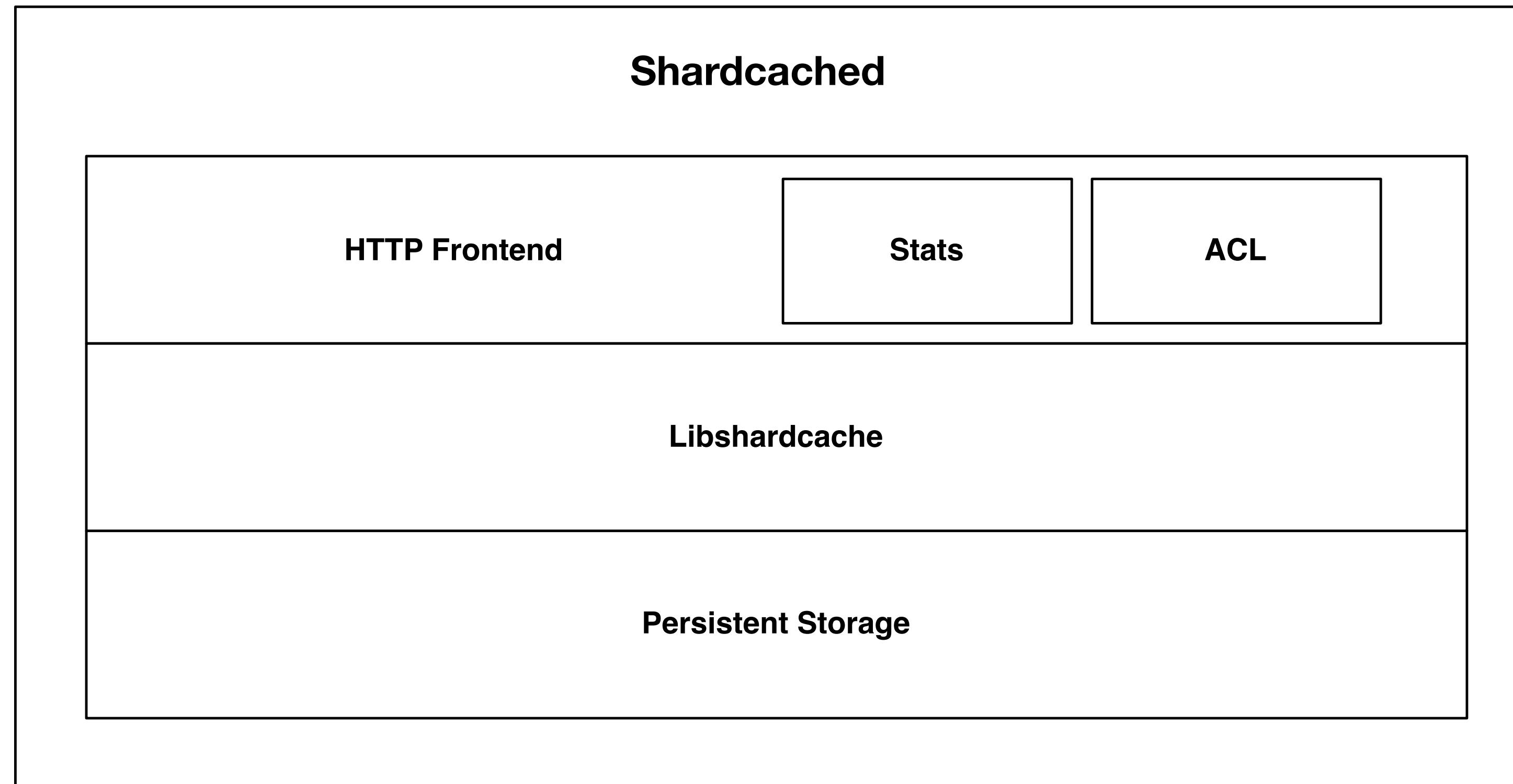
Booking.com

Shardcached

distributed caching @ booking.com

Shardcached

A distributed key/value store with a caching layer

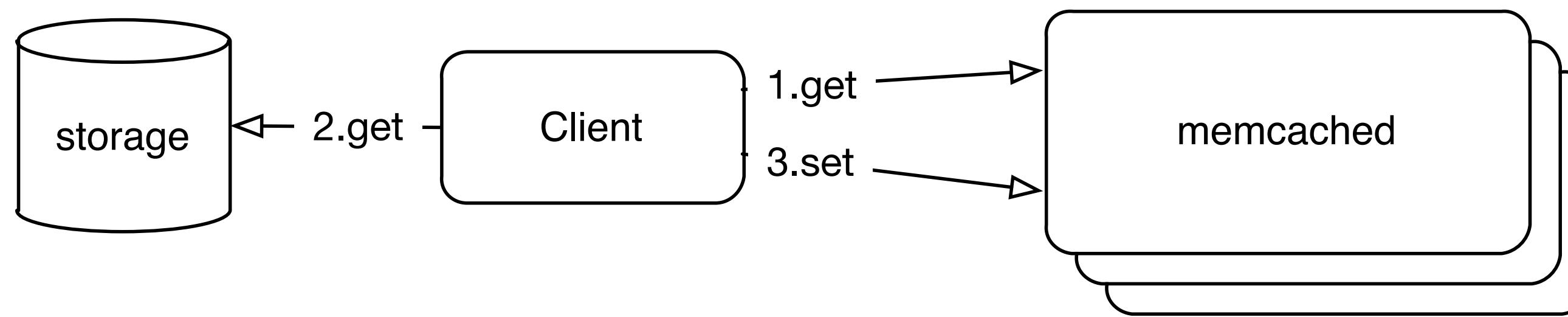


libshardcache:

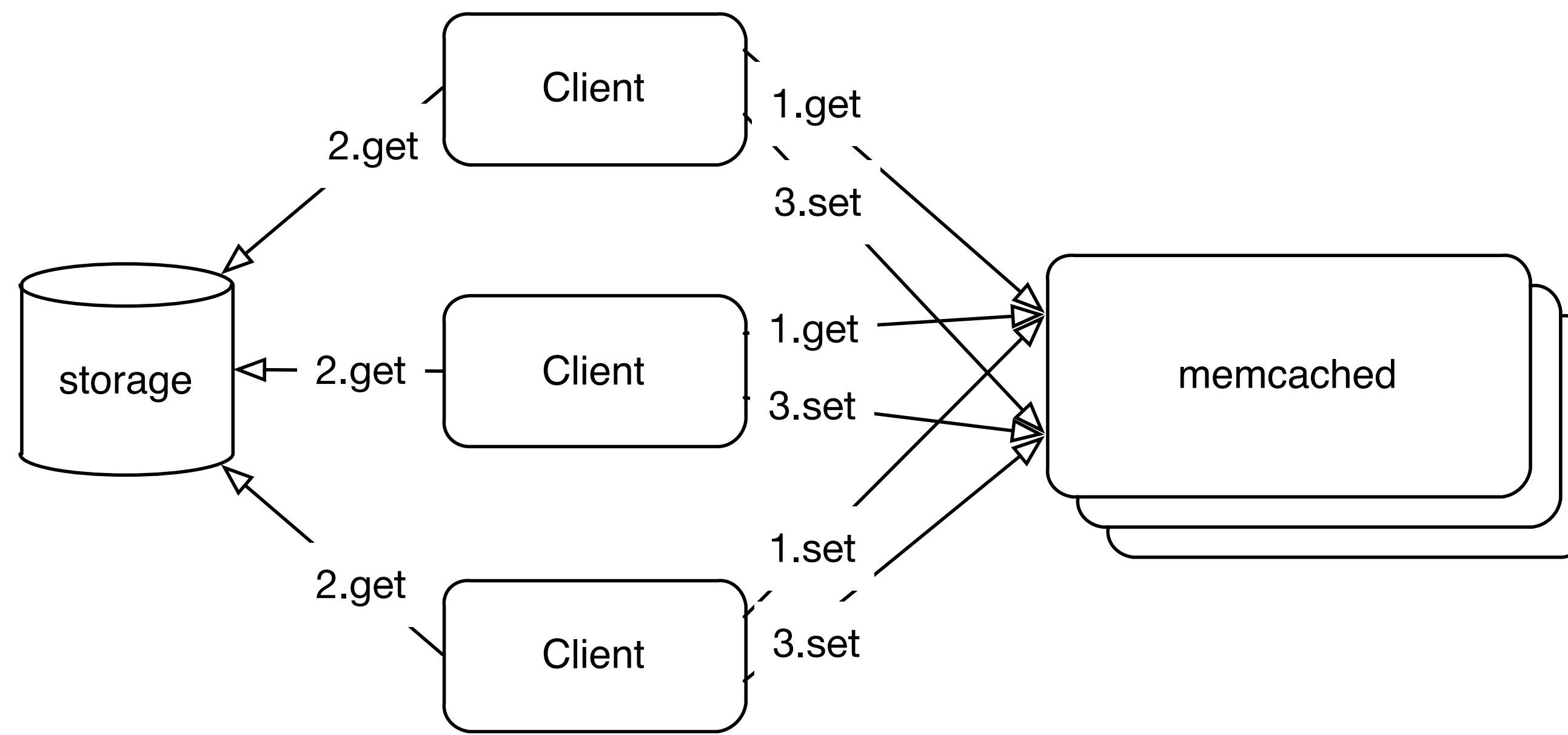
- In-memory caching using an adaptive algorithm (frequency vs recency)
- Loadable plugin interface to the persistent storage
- Minimize access to the storage and handle concurrency
- Nodes are aware of the sharding and communicate to each other accordingly (*a single node can be contacted for any key and not only the owned ones, commands for not-owned keys will be forwarded to the node responsible for them*)

shardcached:

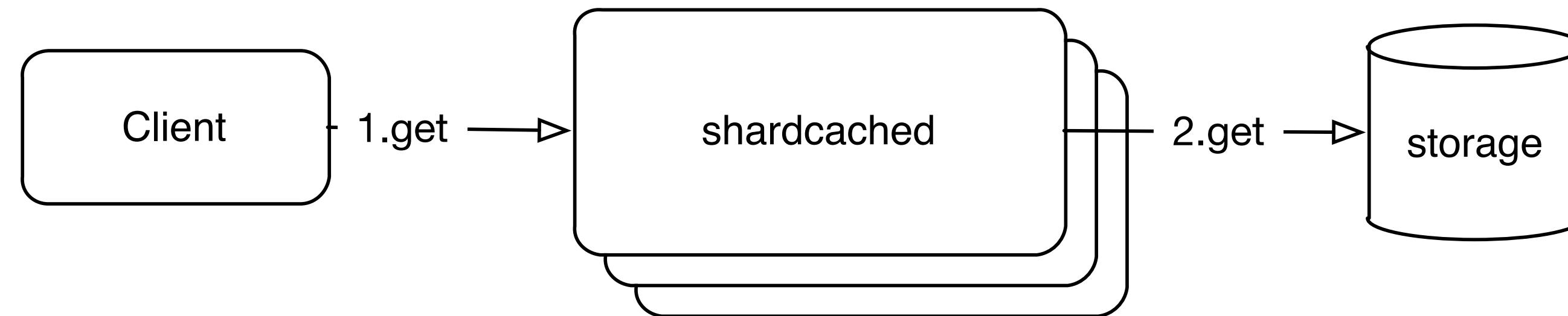
- Implements a daemon embedding libshardcache
- RESTful interface (HTTP)
- Implements various storage plugins
(+ filesystem and memory builtin)
- Access to stats and counters via the http interface
- ACLs
- MIME-Types



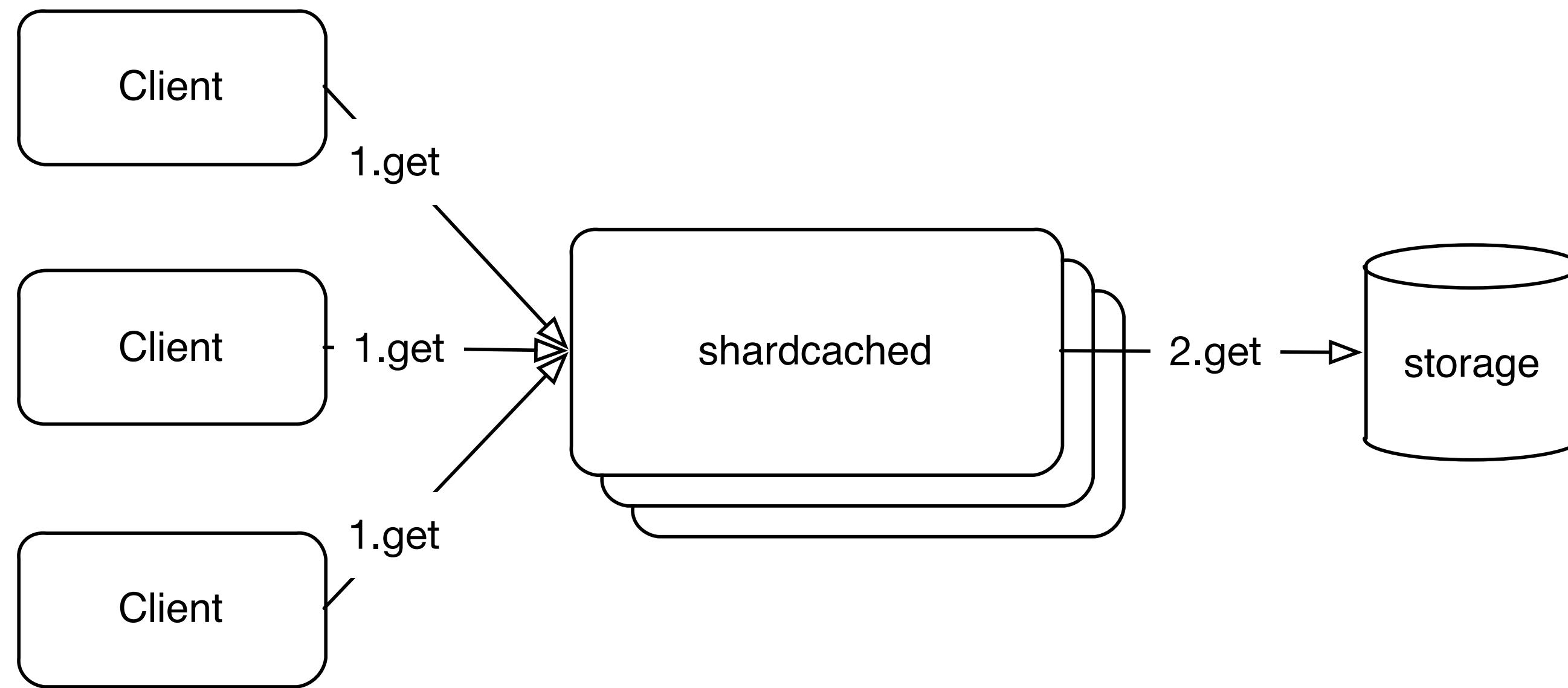
- 1.get the item from memcached**
-> **not found**
- 2. get the item from the storage**
-> **found**
- 3. store the item on memcached**



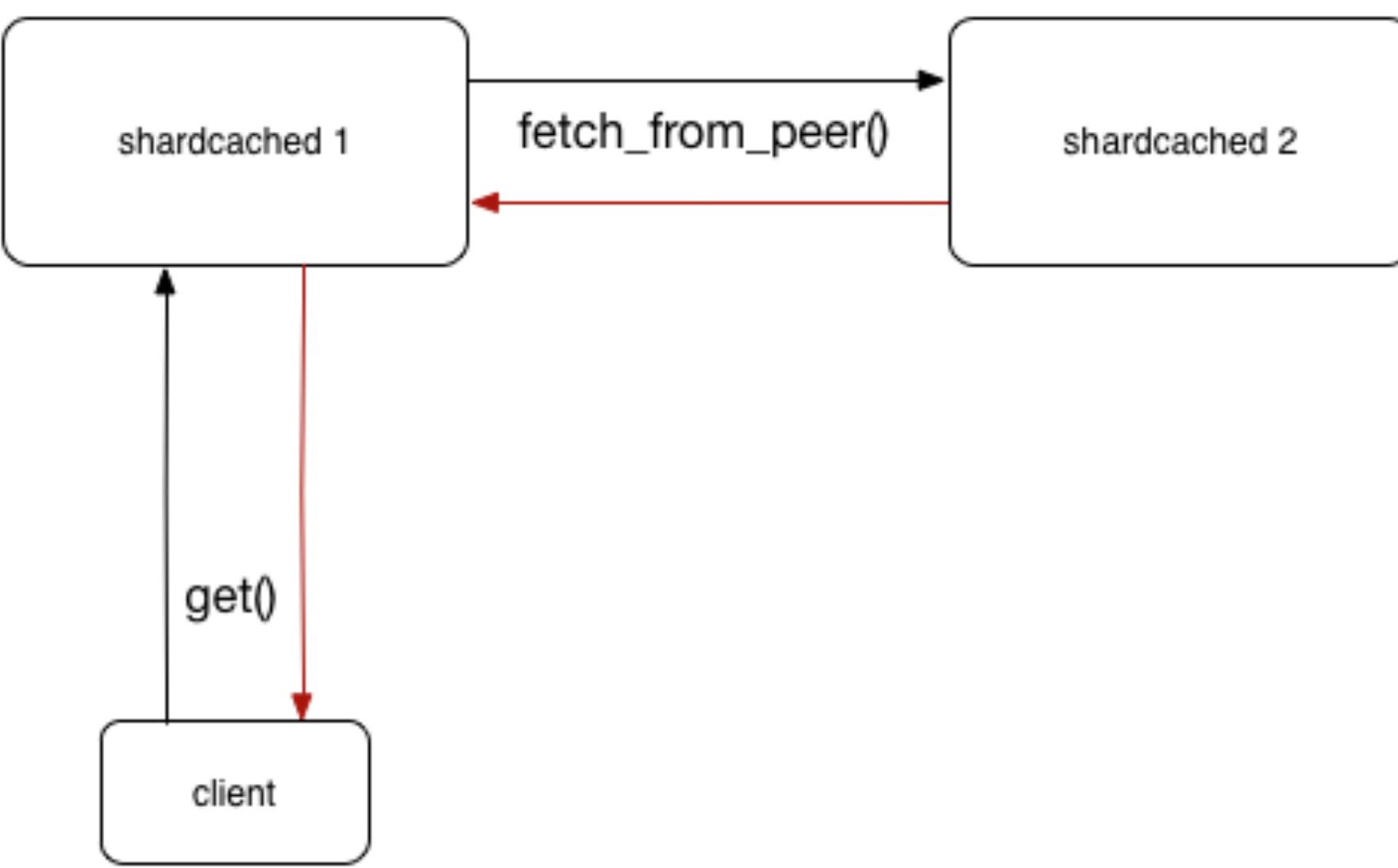
- 1.get the item from memcached**
-> not found
- 2. get the item from the storage**
-> found
- 3. store the item on memcached**

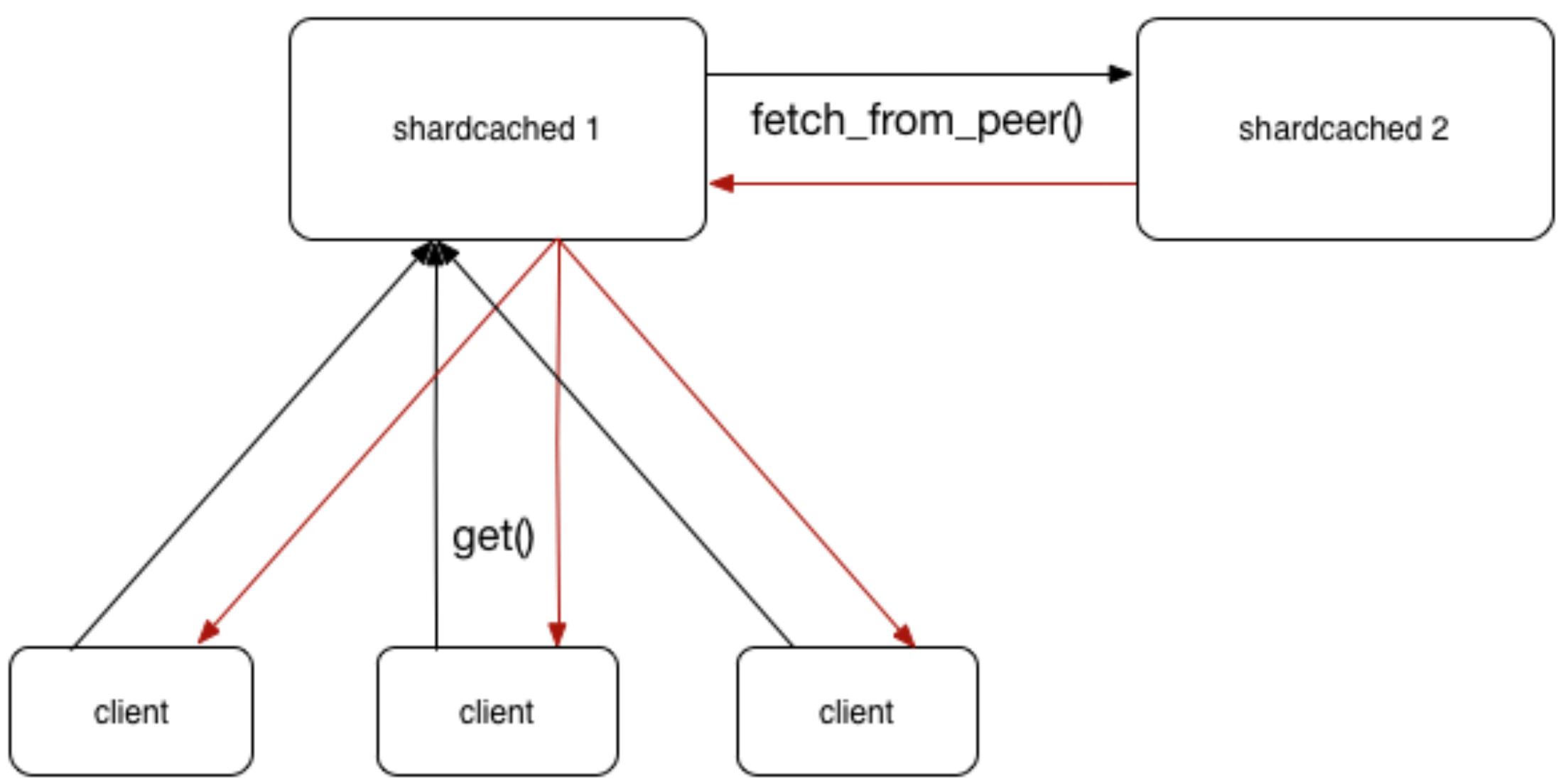


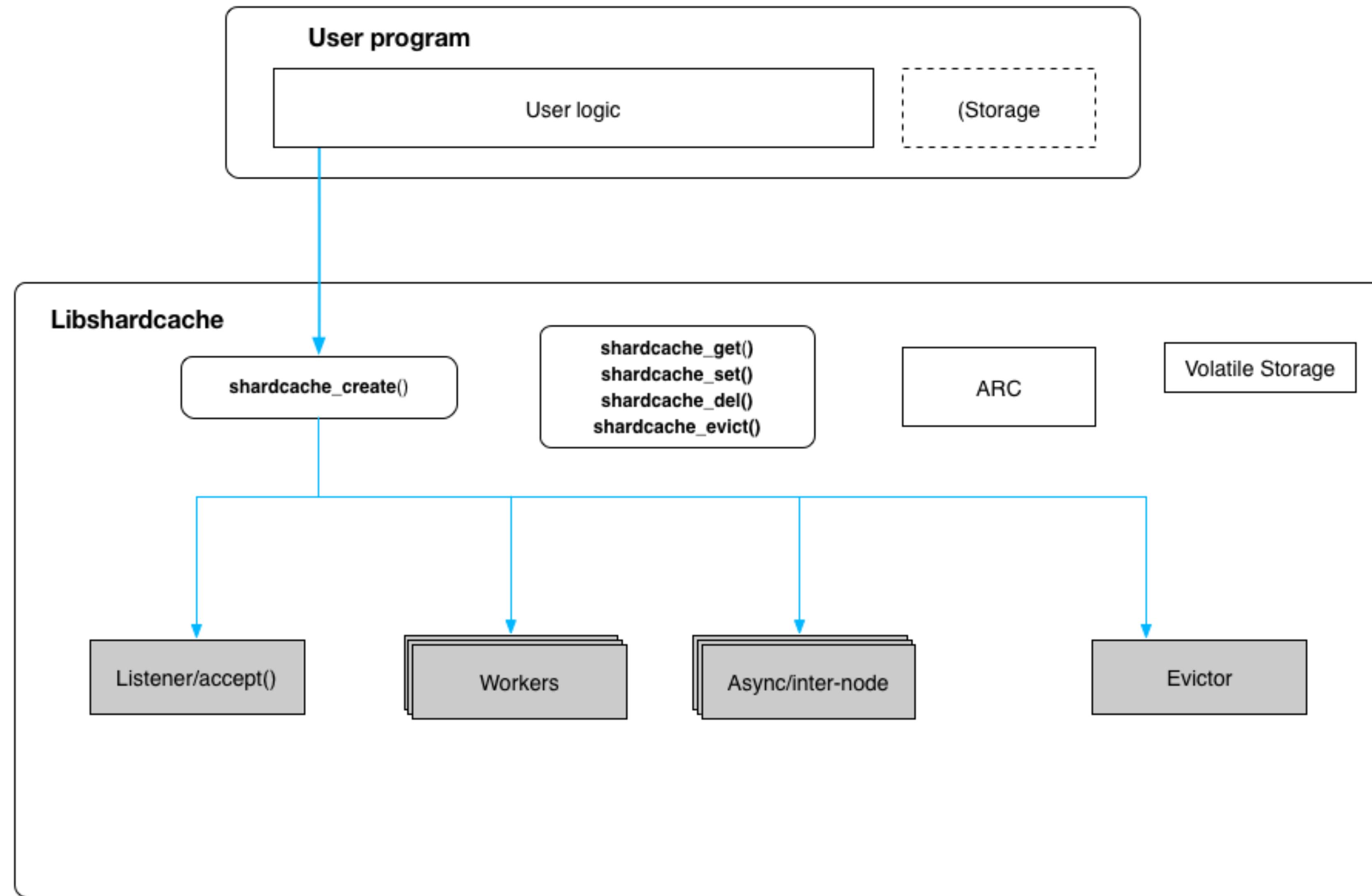
- 1.get the item from shardcache
-> not found**
- 2. get the item from the storage
and return it to the client**

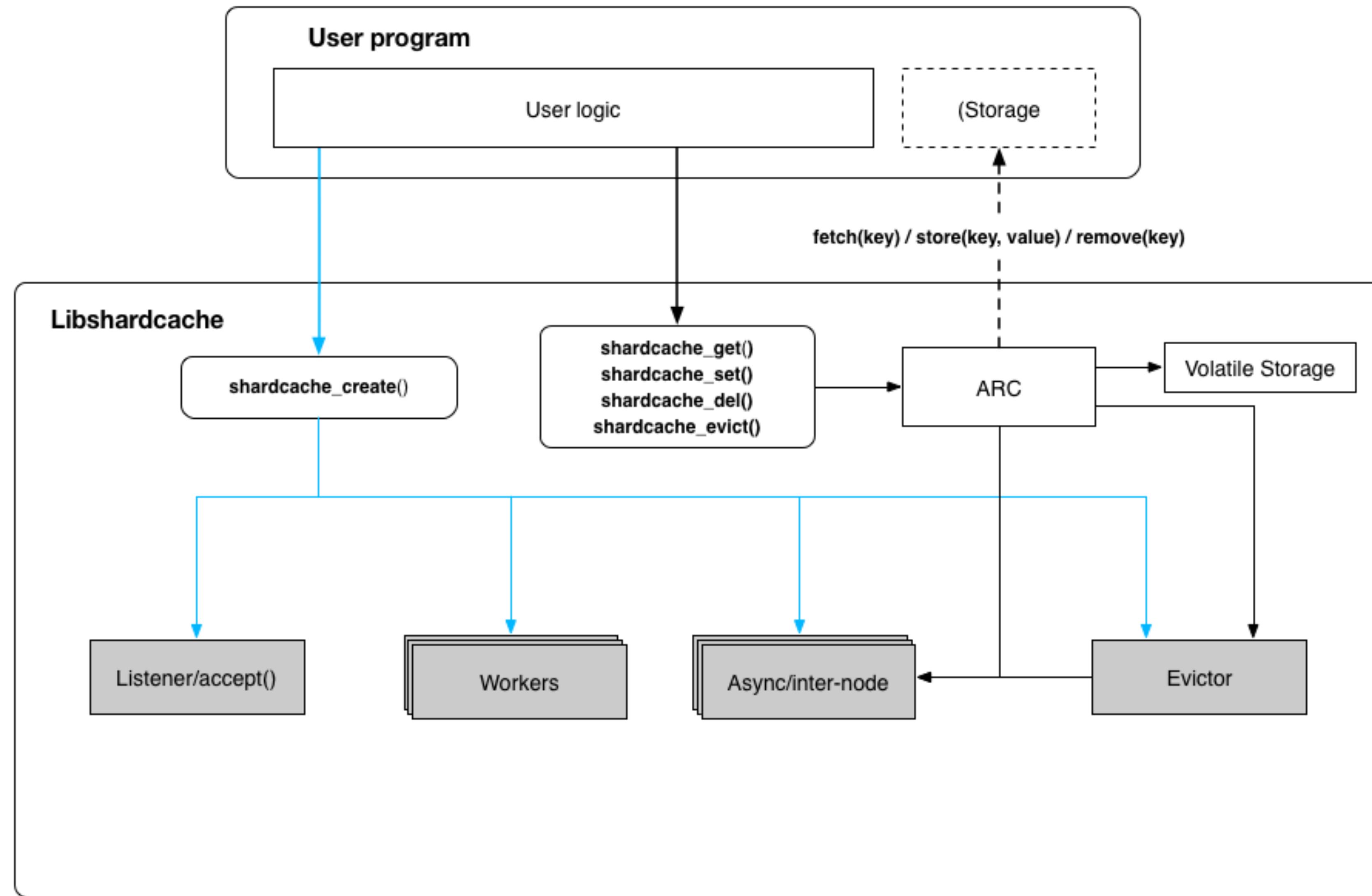


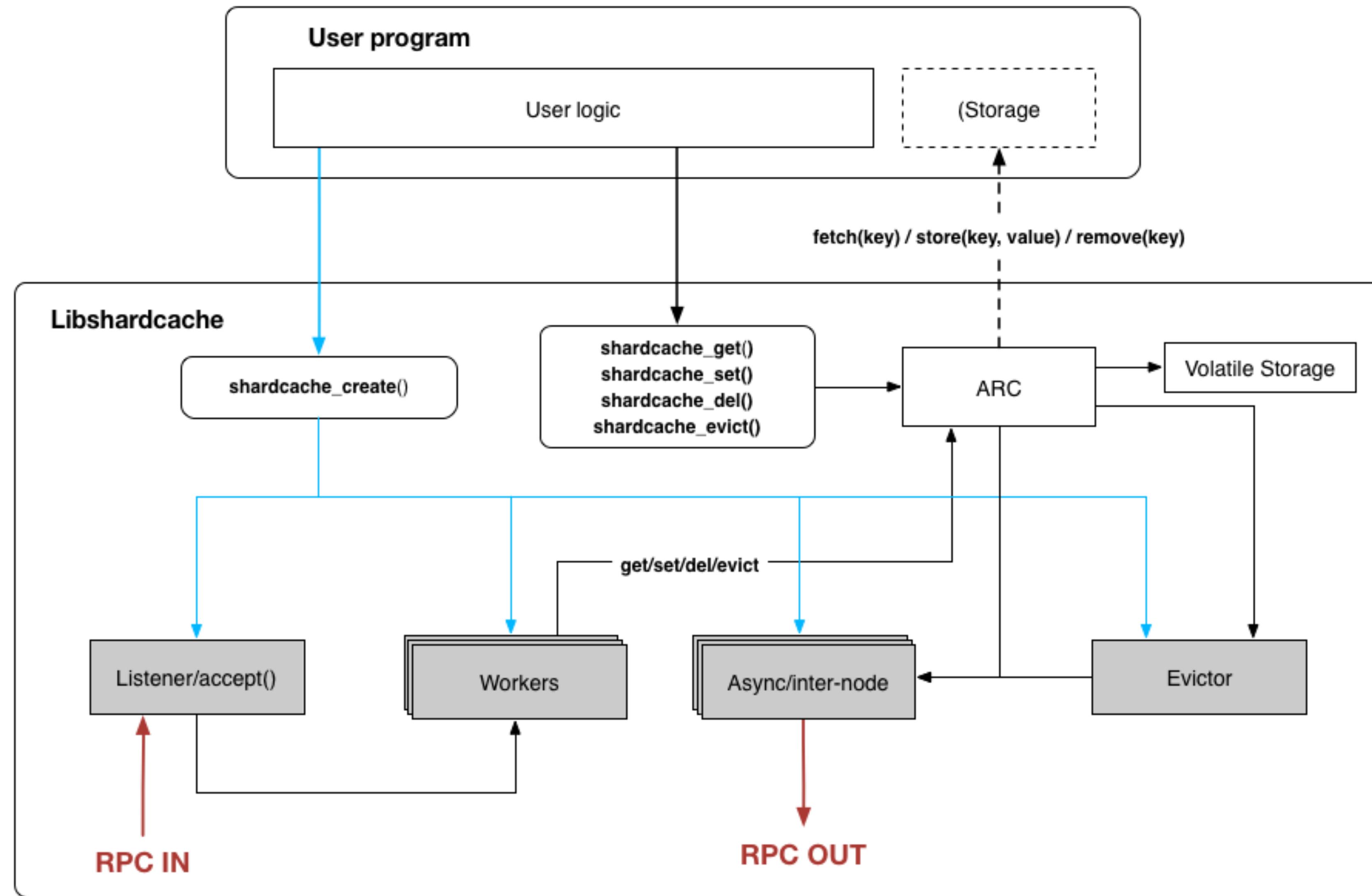
- 1.get the item from shardcache
-> not found**
- 2. get the item from the storage
and return it to the client**

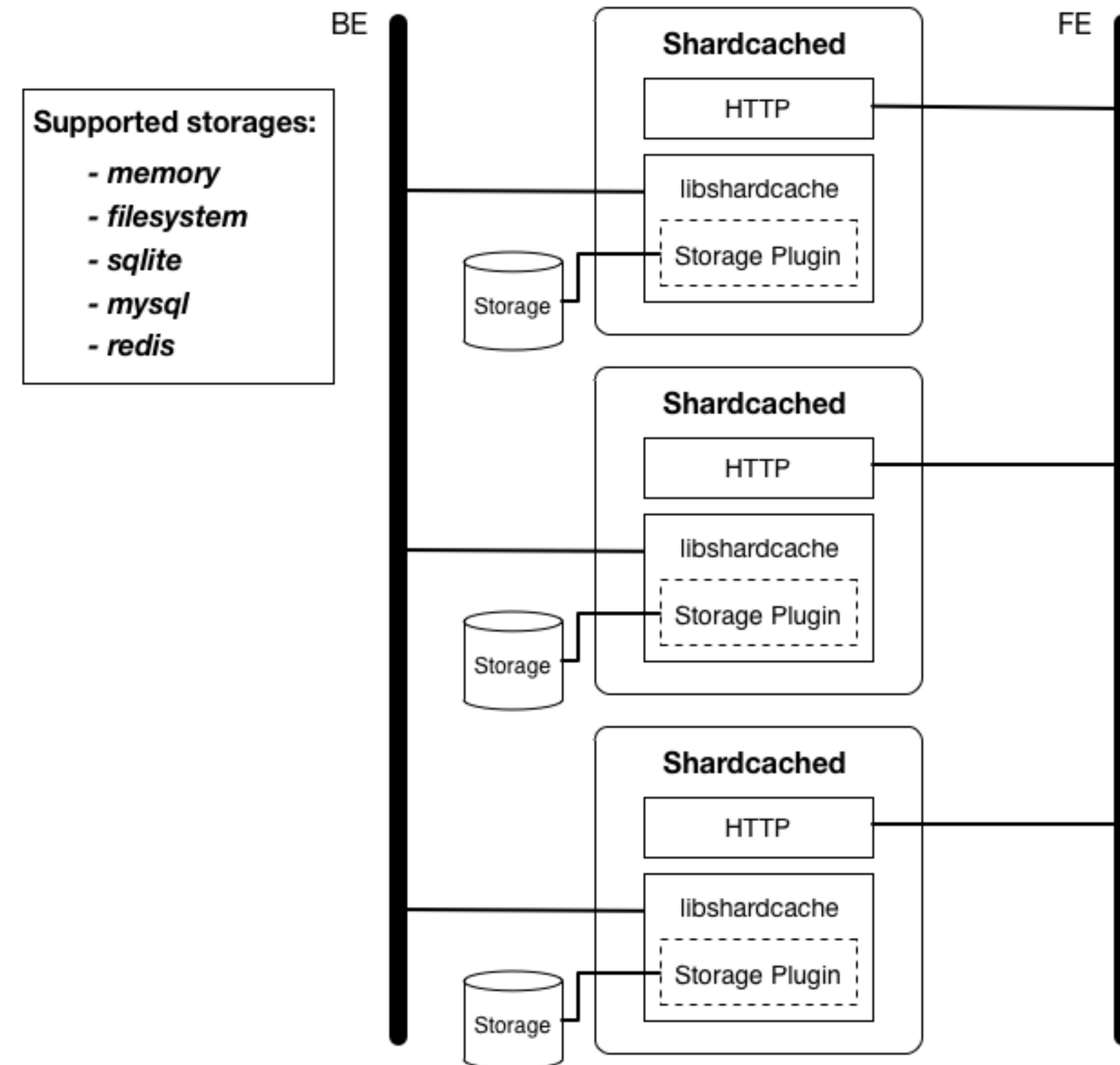


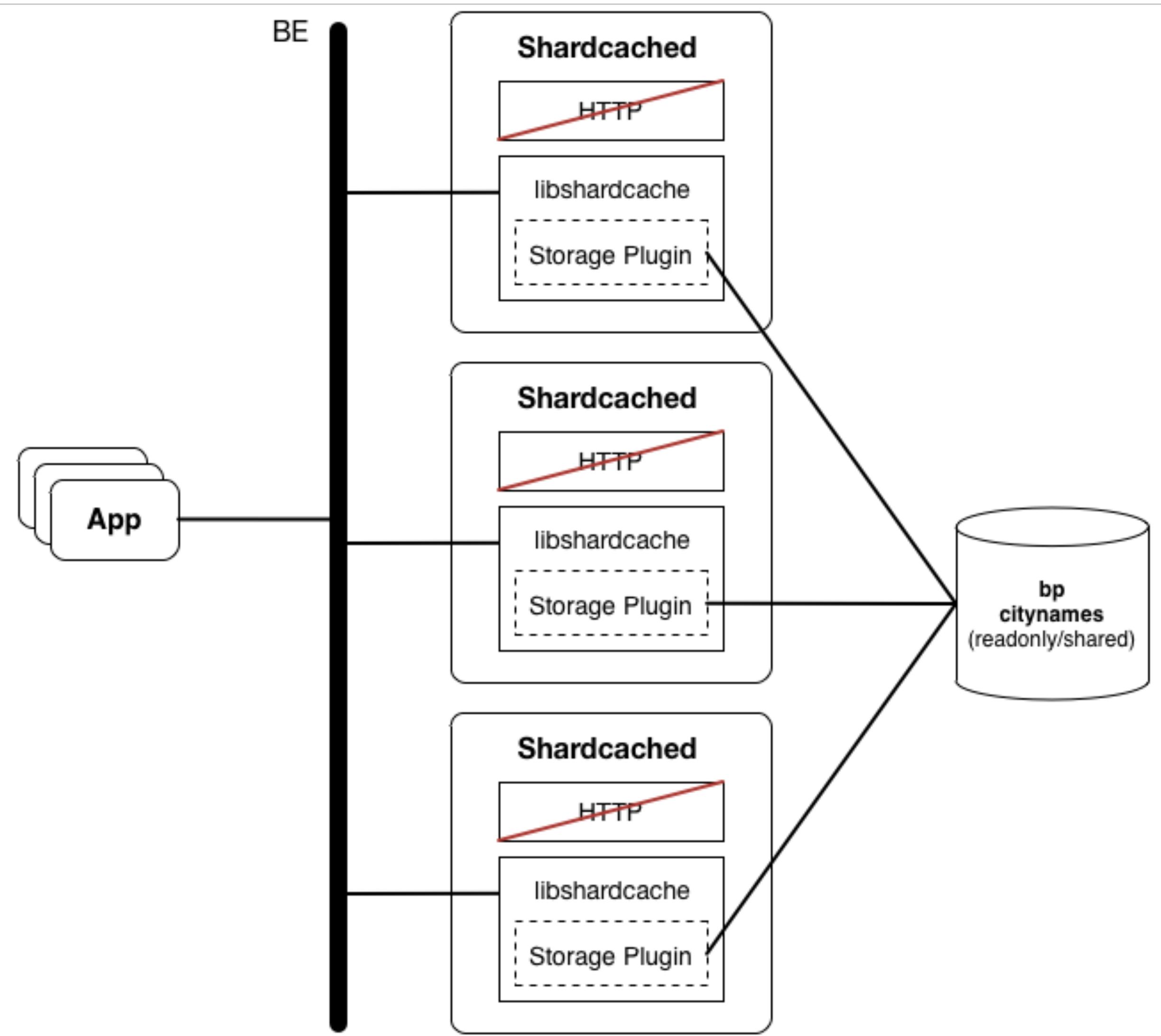






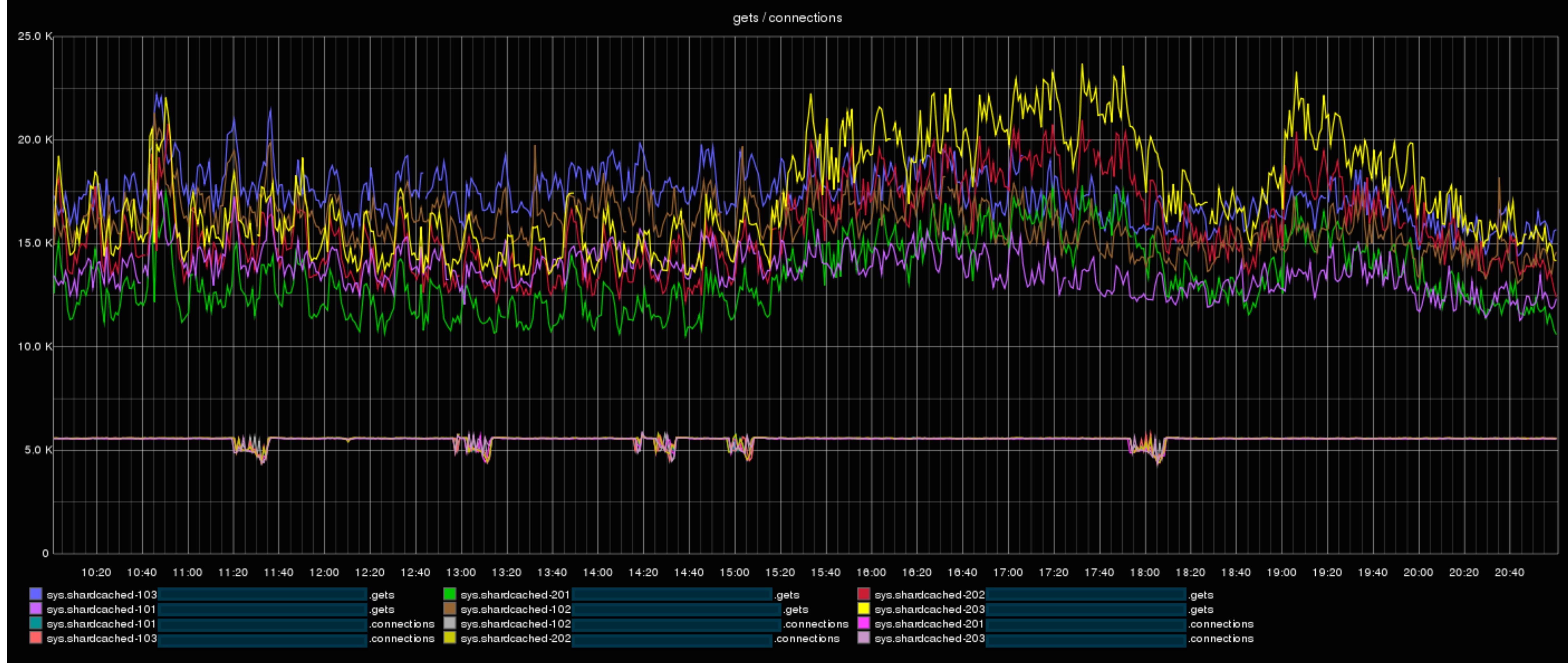


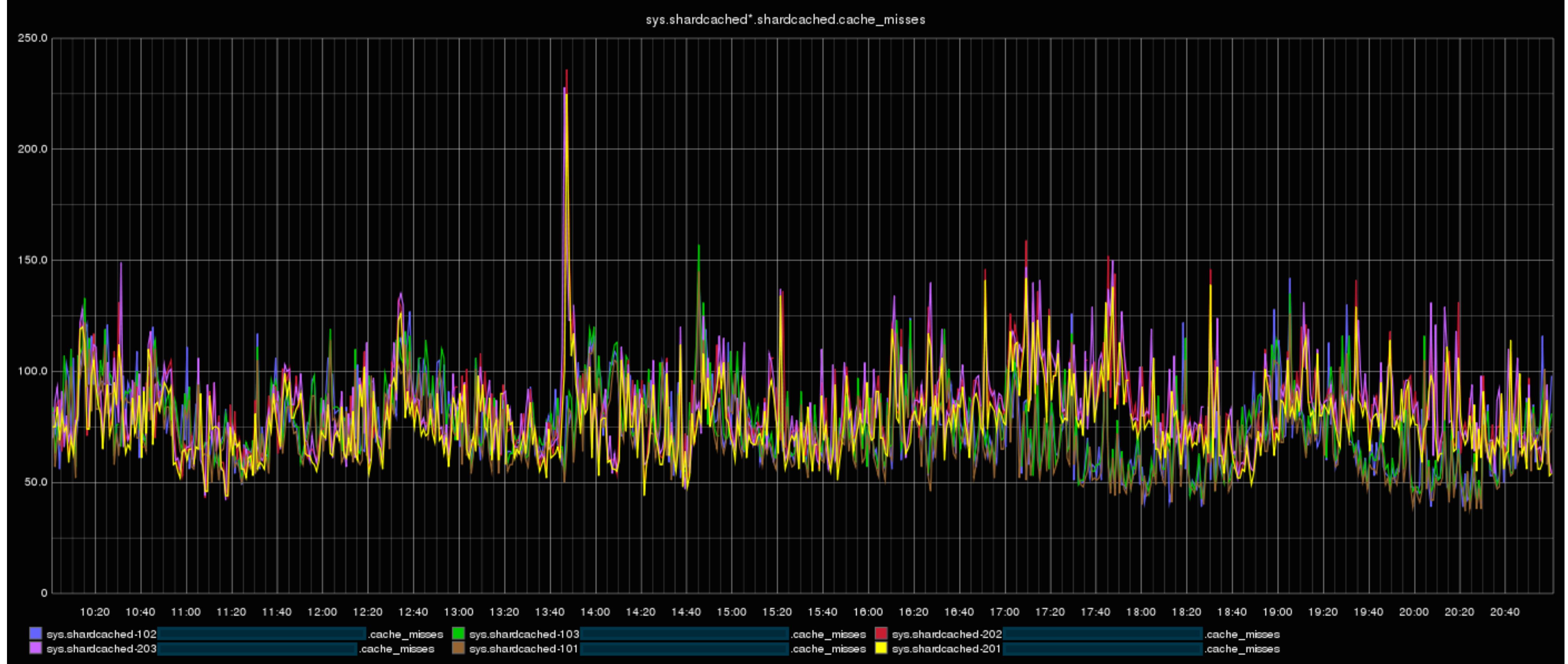


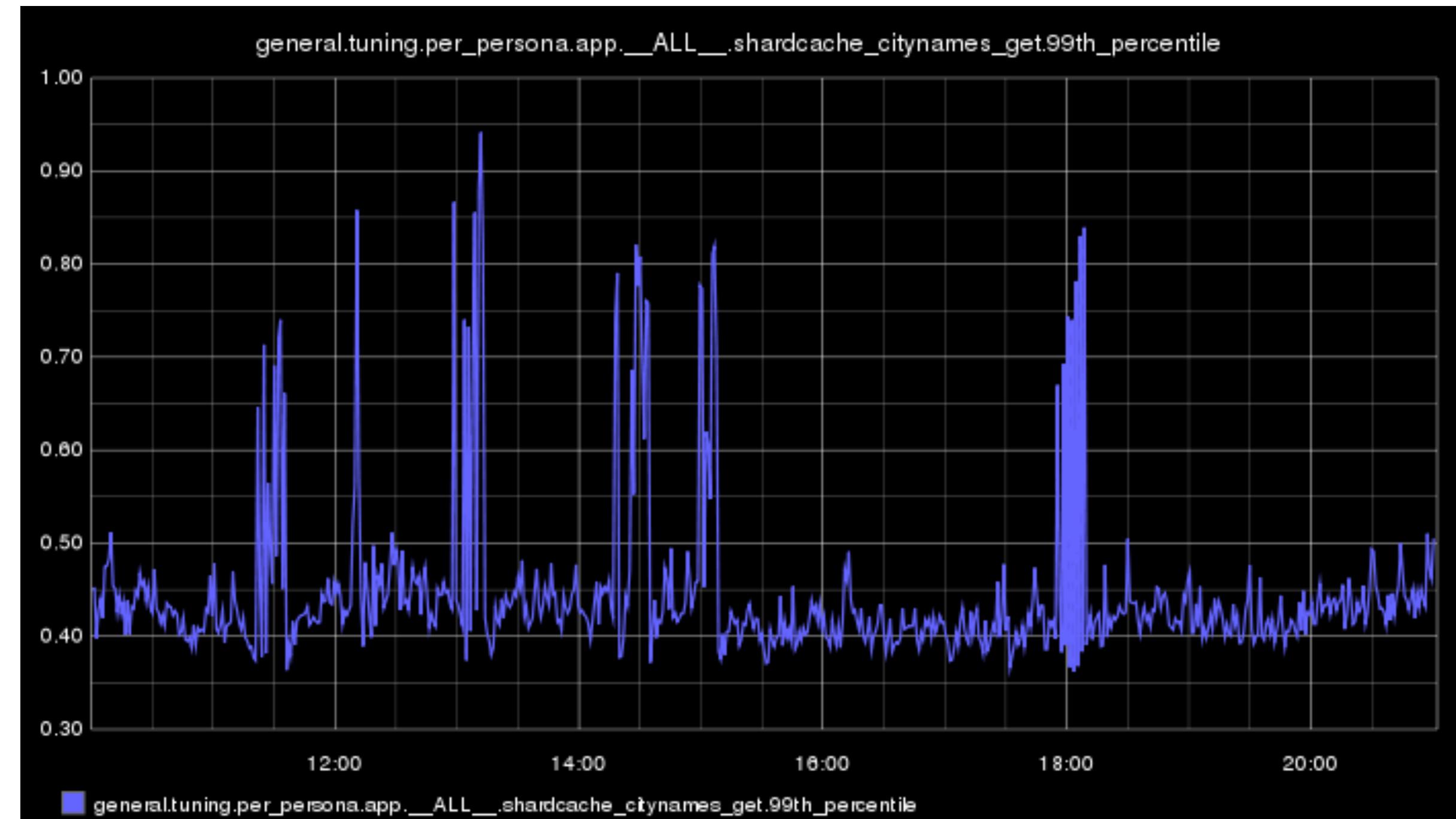
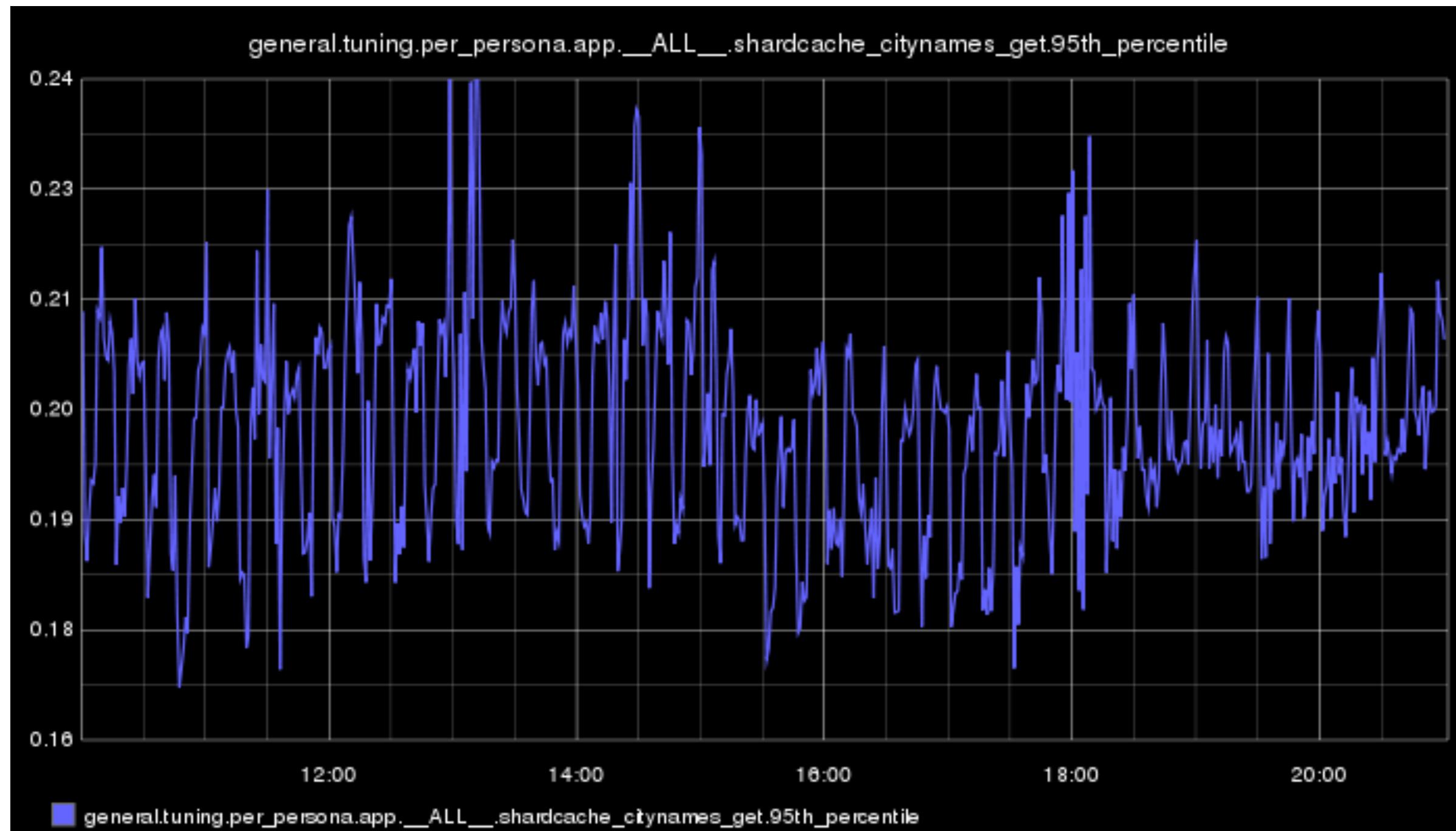
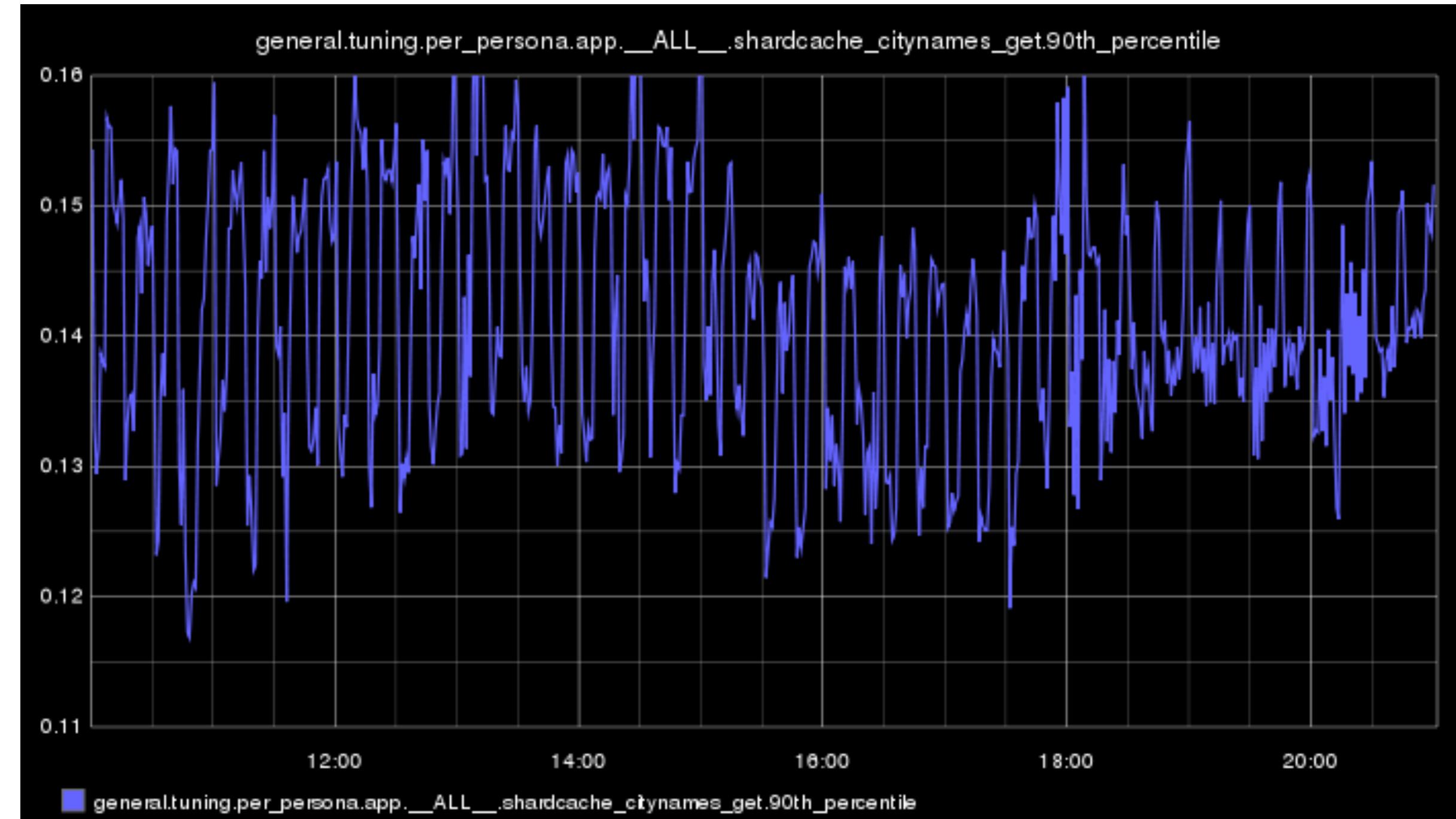
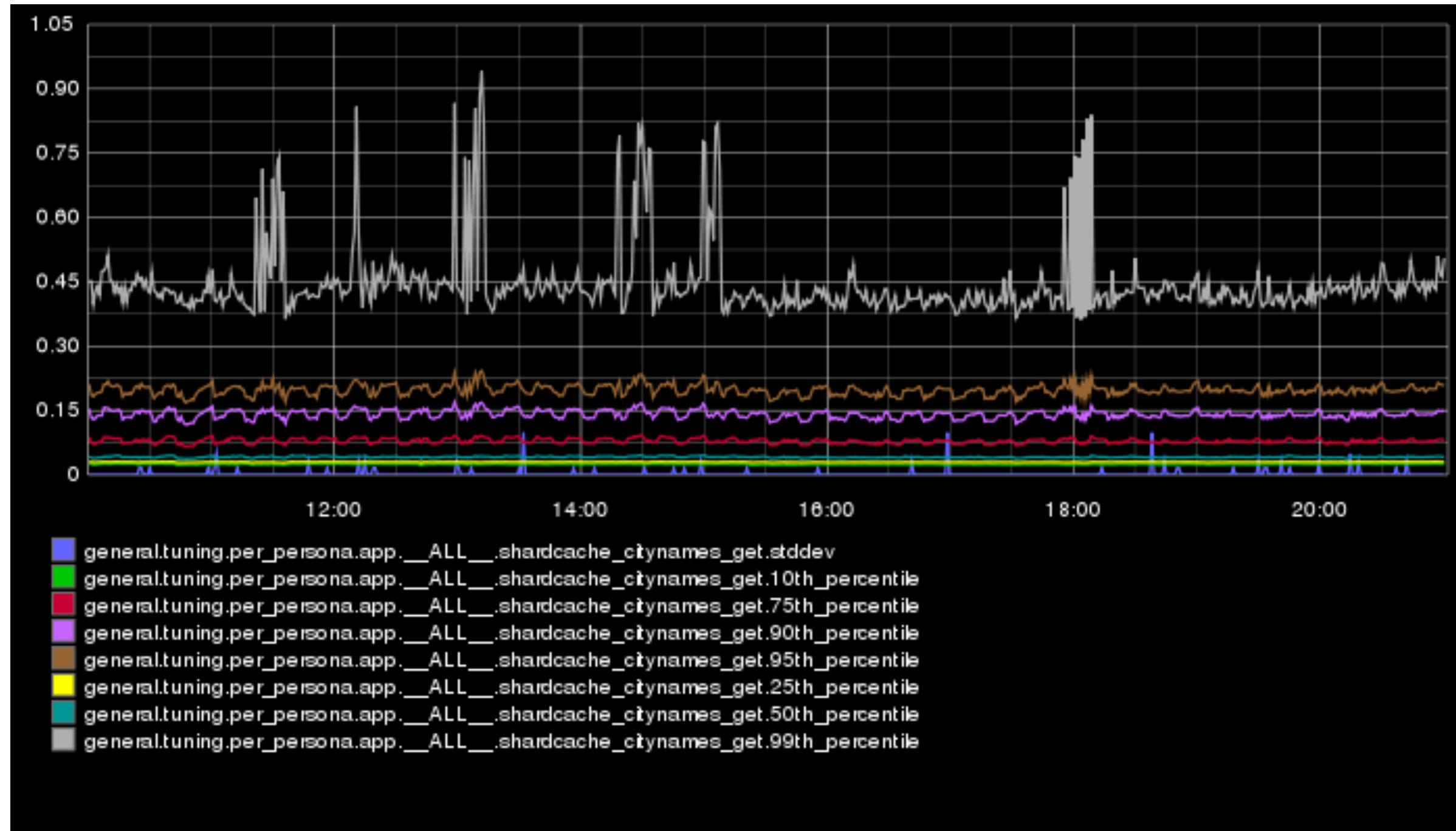


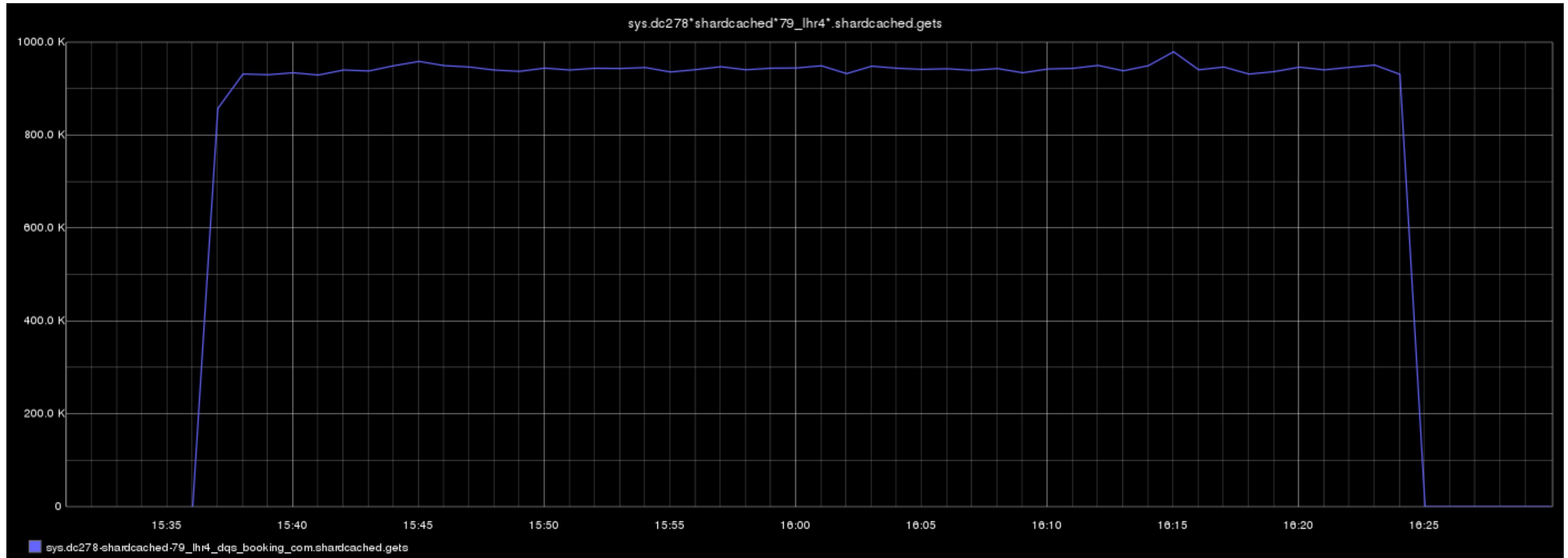
Operational options

- Selective-cache / Always-cache
- Lazy expiration / Active expiration
 - dataset size
 - available memory
 - performance requirements
- Use of persistent connections
- Number of requests to handle in parallel when pipelining

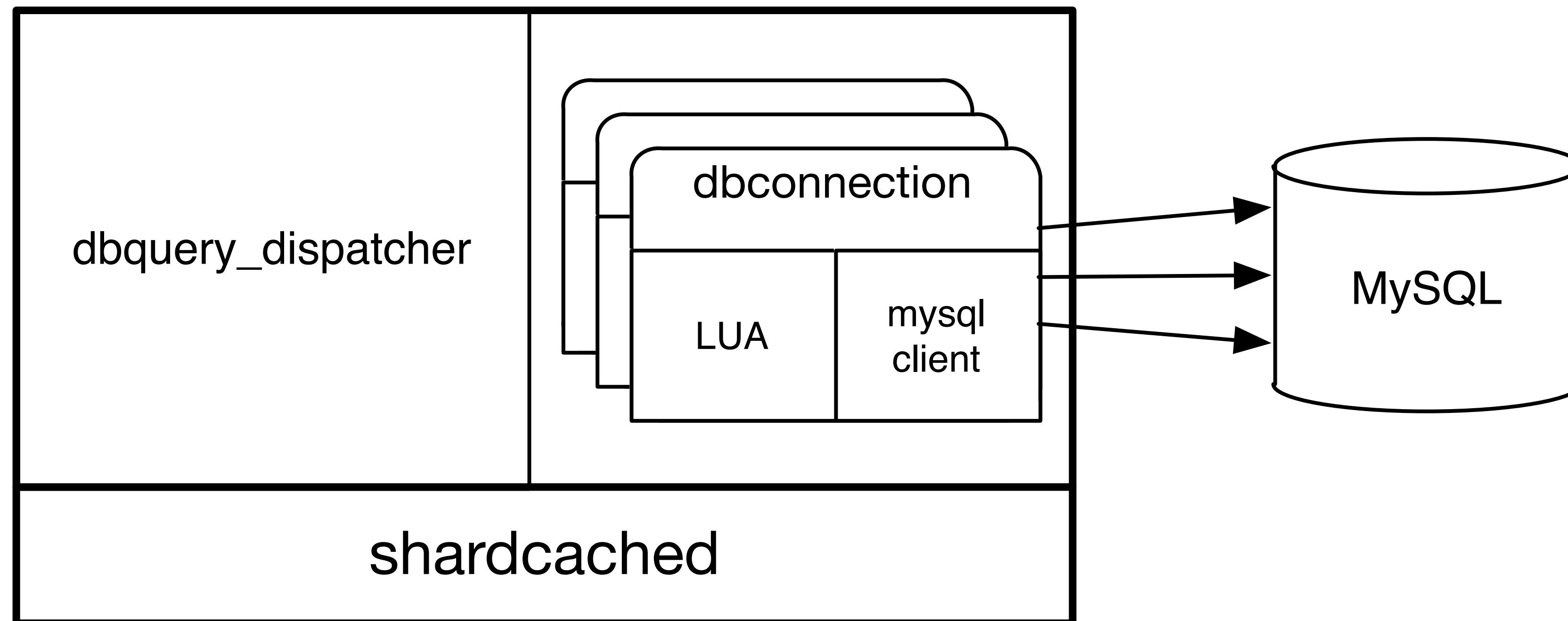








shardcached LUA/MySQL storage plugin



Why

- Many opportunities to cache multiple db queries
(e.g: citynames, roomnames, geoip-information, ...)
- They share the same structure
- No need to rebuild if the db schema changes
- Quick turnaround for deploying new queries

Why LUA

- Just one step ahead of a configuration file
- Small, easy to integrate C runtime API (even in a multithreaded environment)
- Small memory footprint
- Discourage abuse - in case of need, use a libshardcache storage module

- Set up MySQL connections
- Set up one LUA execution context per connection
- Initialise the prepared statements
- Select LUA module and execute queries on DB

C side

LUA side

- Define queries
- Parse the key and extract the values used in the where clause
- Combine the returned columns into a single result buffer

Example

```
{  
  'databases': { 'bp' },  
  'fetch': {  
    {  
      'database': 'bp',  
      'bind_in': 'SS',  
      'sql': 'SELECT col_a, col_b FROM table WHERE col_c = ? OR col_d = ?',  
      'bind_out': 'SS',  
      'pre': function (key) { ... },  
      'post': function (col_a, col_b) { ... },  
    }  
  },  
  ...  
}
```

Call Flow

Shardcache	get(key)	get('citynames:-3875374,de')
LUA	key_a, key_b = pre(key)	a = -3875374, b = de
MySQL	SELECT value_a, value_b WHERE column_a = key_a AND column_b = key_b	NULL, "Dublino"
LUA	value = post(value_a, value_b)	"Dublino" = post(NULL, "Dublino")
Shardcache	return value	return "Dublino"

Todo

- shardcached:
 - v2 protocol: 'get/set multi' command
 - inter-node pipelining
 - automate deployment
- dbquery_dispatcher:
 - Expose the 'store' command
 - Integrate Pure-Lua sereal
 - automate test and deployment of the lua modules

- **github.com/xant/shardcached**
- **github.com/xant/libshardcache**
- **git.booking.com/git/?p=extension.git**