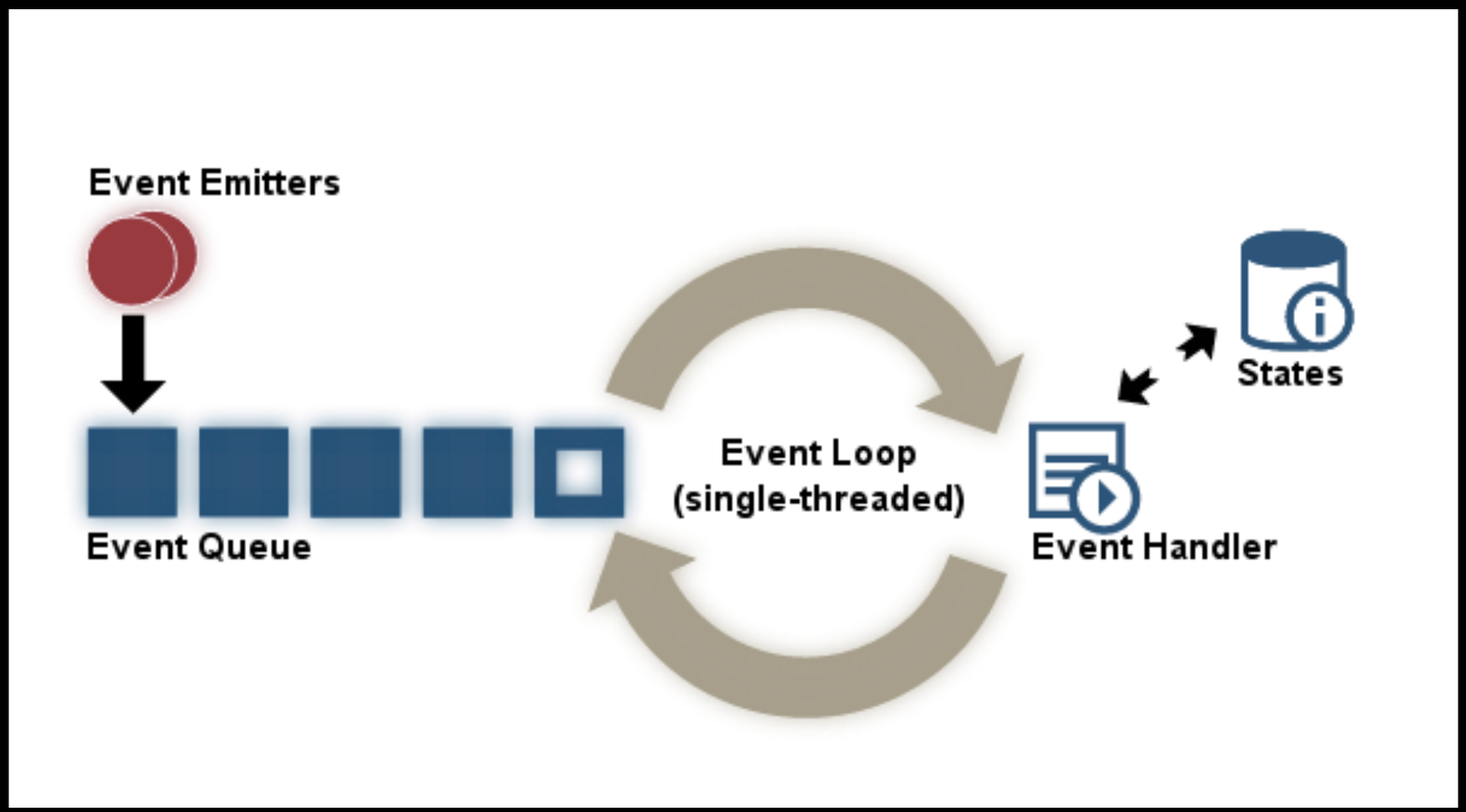
# Openresty

nginx on Steroids

# nginx

- HTTP Server with High Performance
- Event based architecture
- Small Footprint on memory and processing
- Declarative configuration

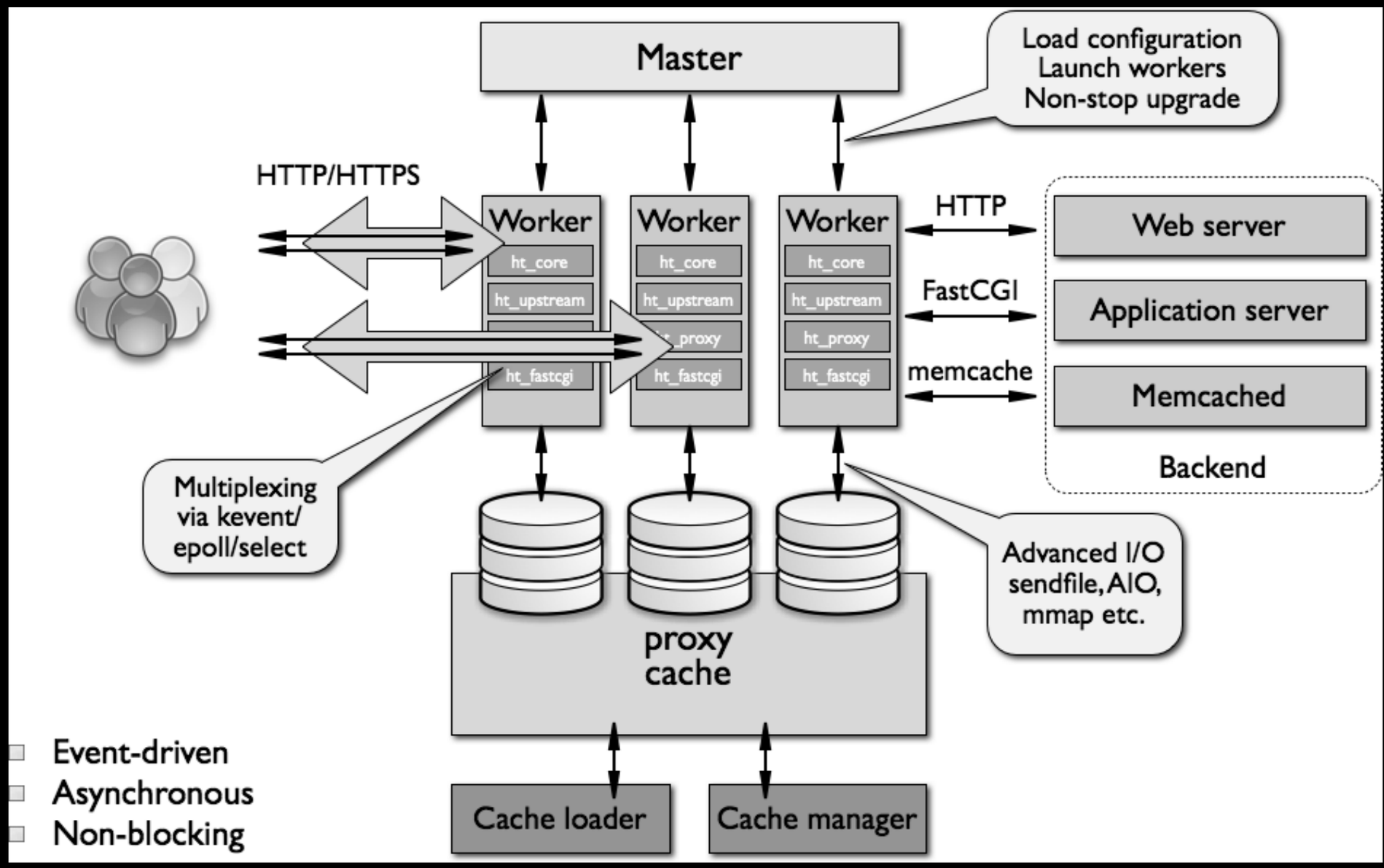
# Event Loop


Event Emitters
Event Queue
Event Loop
(single-threaded)
Event Handler
States

# Event Loop

- Reactor Pattern
- Concurrent event demultiplexing
- Event handler dispatching

# Nginx and Event Loop



nginx is a true grandmaster

# nginx - Component

- Master
  - Handles signals and notify workers
- Workers
  - Process Client requests and handles connections
  - Handles a complicated run-loop
- Key Principle: Be as non-blocking as possible

# Openresty

- Embeds Lua in Nginx (HTTP Server)
- Leverage nginx subrequests
- Integrates Lua coroutines into nginx event model
- Expose Nginx environment to Lua via API
- 100% non-blocking
- Blazingly fast due to LuaJIT

# Openresty Directives

- init_by_lua and init_worker_by_lua
  - Perform tasks during nginx startup
  - Example: Load data from Redis to Shared Dict

# Openresty Directives

- content_by_lua
- Content handler phase
- Non blocking
- Generates HTTP response

# Openresty Directives

- rewrite_by_lua
  - Lua-fy the nginx rewrite phase
  - Make api calls
  - Dynamic rewrites

# Shared Data

- shared memory zone
  - serve as storage for the shm based Lua dictionary
  - shared by all the nginx worker processes in the current nginx server instance

```
http {
    lua_shared_dict dogs 10m;
    ...
}
```

- ngx.shared.DICT
  - Fetching the shm-based Lua dictionary object for the shared memory zone

# Shared Data

```
http {
    lua_shared_dict dogs 10m;
    server {
        location /set {
            content_by_lua_block {
                local dogs = ngx.shared.dogs
                dogs:set("Jim", 8)
                ngx.say("STORED")
            }
        }
        location /get {
            content_by_lua_block {
                local dogs = ngx.shared.dogs
                ngx.say(dogs:get("Jim"))
            }
        }
    }
}
```

# Use cases

- Process output from Redis/Memcache/MySQL
    - URL Redirection Tool
- Access control and traffic control
    - Rate limiting
    - Manipulate response header
- Web applications
    - URL shortener
- Complex URL dispatch
    - URL redirection Tool
    - Gateway authentication

# Openresty - Subrequests

- ngx.location.capture
  - Synchronous but still non-blocking Nginx Subrequest
    - Mimic the HTTP interface but there is no extra HTTP/TCP traffic
    - Example: Authenticaton Layer

**local res = ngx.location.capture("/v1/api/user/checksession", {method = ngx.req.get_method() == 'POS**
**and ngx.HTTP_POST or ngx.HTTP_GET, body = '', args = ngx.req.get_uri_args()})**

# Openresty Cosockets

- Send and receive on TCP or Unix domain sockets
- API compatible with LuaSocket, yet non-blocking to Nginx
- Has a keepalive mechanism to avoid connect/close for each request

# Cosockets based Libraries

- lua-resty-redis
- lua-resty-memcache
- lua-resty-mysql
- Example

```
local ok, err = red:connect("127.0.0.1", 6379)
if not ok then
     ngx.log(ngx.ERR, "Failed to connect to redis: ", err)
    ngx.exit(ngx.OK)
end
redirect_uri, err = red:hget("ir", uri)
```

# Openresty - Rate Limiting

- Library - openresty/lua-resty-limit-traffic

```
local limit_req = require "resty.limit.req"
local lim, err = limit_req.new("my_limit_req_store", 2, 0)

if not lim then
    ngx.log(ngx.ERR, "failed to instantiate a resty.limit.req object: ", err)
    return ngx.exit(500)
end
local key = ngx.req.get_headers()["X-USER-ID"]
if key ~= '' then
        local delay, err = lim:incoming(key, true)

        if not delay then
            if err == "rejected" then
                ngx.log(ngx.ERR, ' -- in rejected -- ')
                return ngx.exit(429)
            end

        end

        if delay >= 0.001 then
            local excess = err
            ngx.sleep(delay)
        end
end
```

# Openresty - URL Redirection

- ngx.req.set_uri
- Rewrite the current request's (parsed) URI by the uri argument

```
location /test {
    rewrite_by_lua_block {
        local uri = ngx.re.sub(ngx.var.uri, "^/test/(.*)", "/$1", "o")
        ngx.req.set_uri(uri)
    }
    proxy_pass http://my_backend;
 }
which is functionally equivalent to
 location /test {
    rewrite ^/test/(.*) /$1 break;
    proxy_pass http://my_backend;
 }
```

# Openresty - Other Use cases

- Ngx Lua Datadog
  - Leverage Log phase using log_by_lua
    - Push custom HTTP metrics from Nginx
- Dynamic Upstreams
  - Datadog(statsd) collection from inside lua scripts

Thank you

*Questions?*