Модели акторов в C++ миф или реальность

Боргардт Александр Александрович
https://github.com/smart-cloud

What is in the report:

1. The general concept of the actor model;
2. Consider the example of a solved problem with Productions;
3. The internal device of the actor;
4. A little about the life cycle of threadpool;
5. About Different strategies work threadpool.

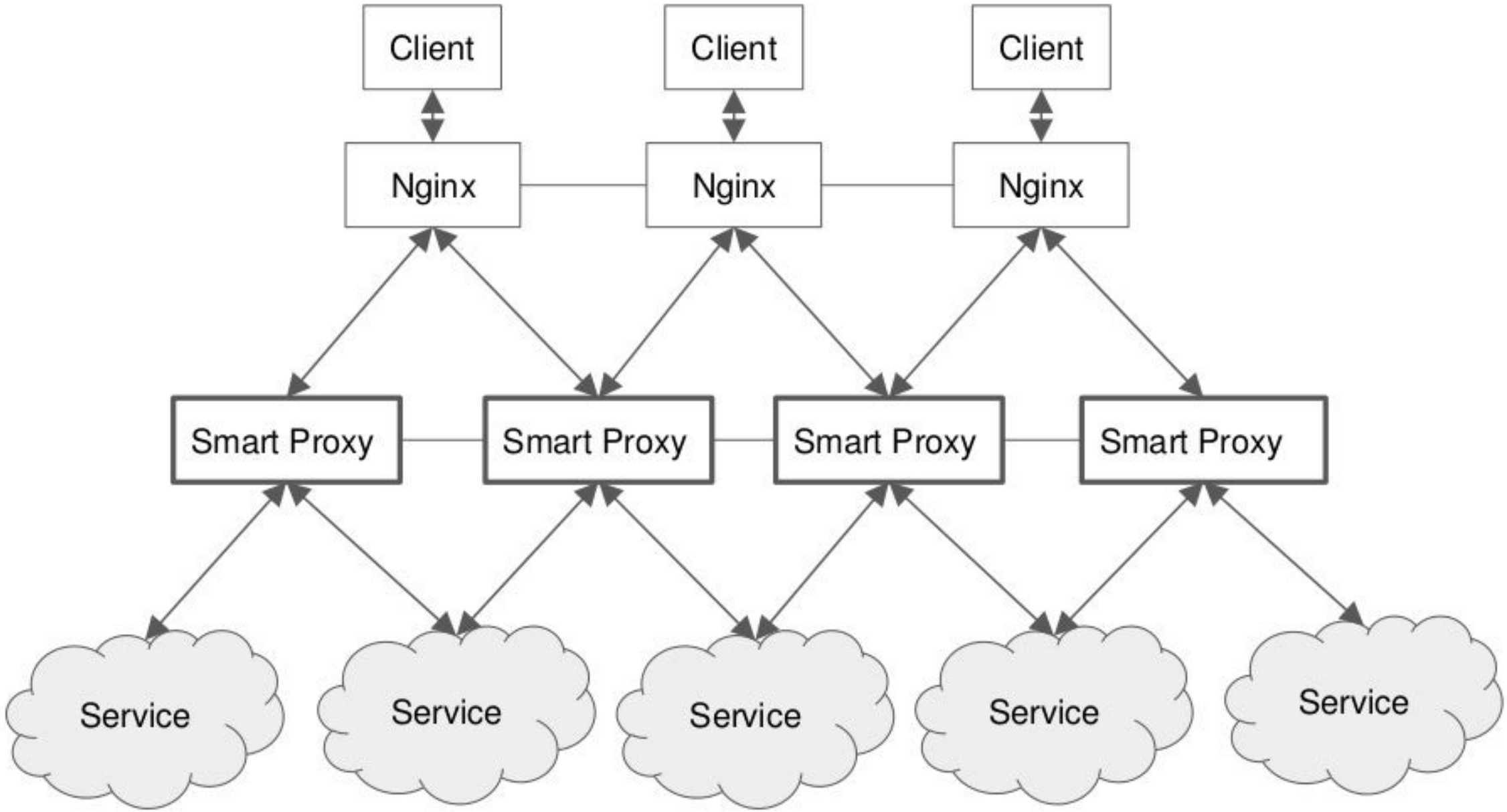
Mythology

Mythology

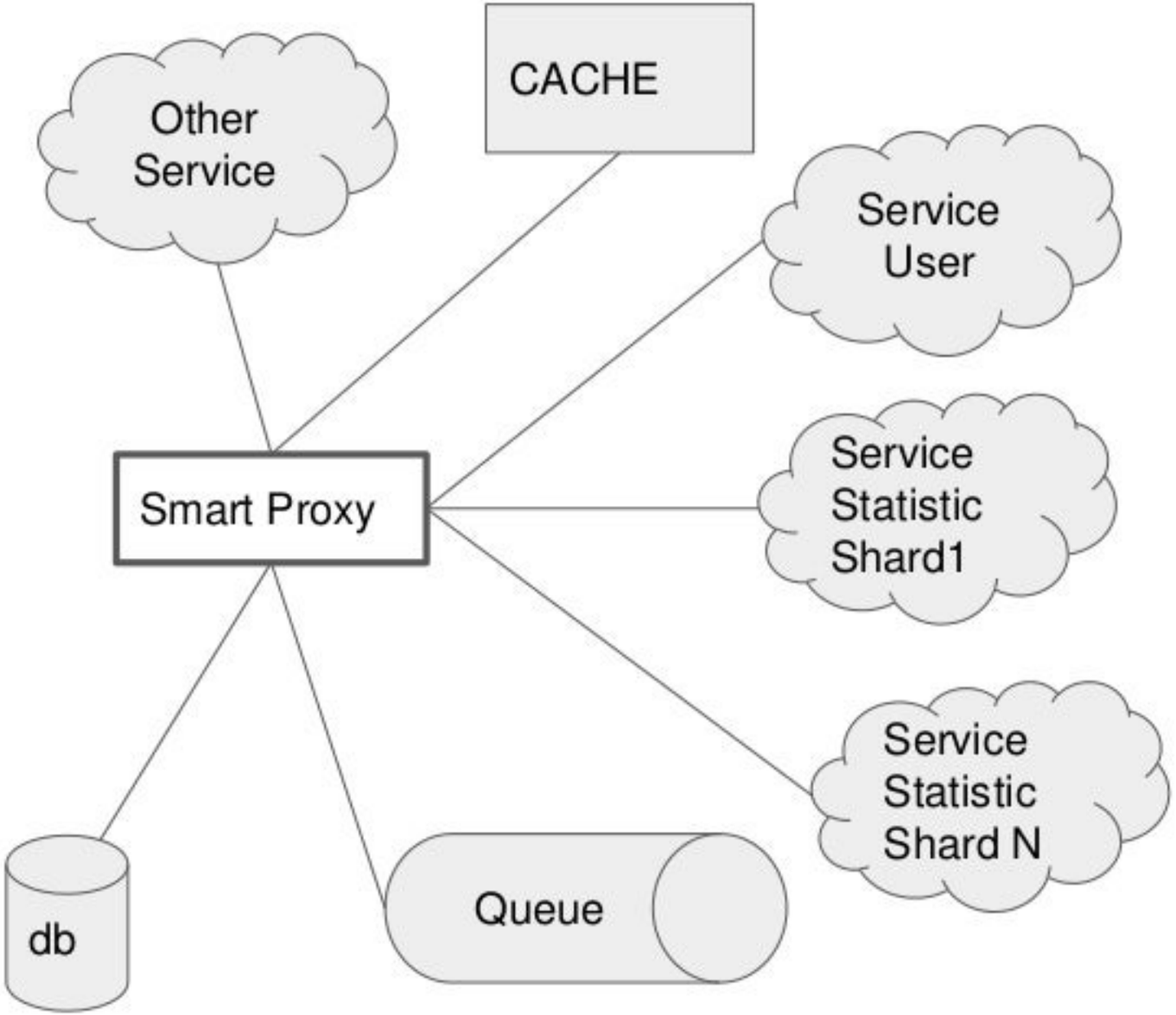

Если ты всё сделал правильно,
Это ещё не значит,
Что у тебя всё будет хорошо...

Intro

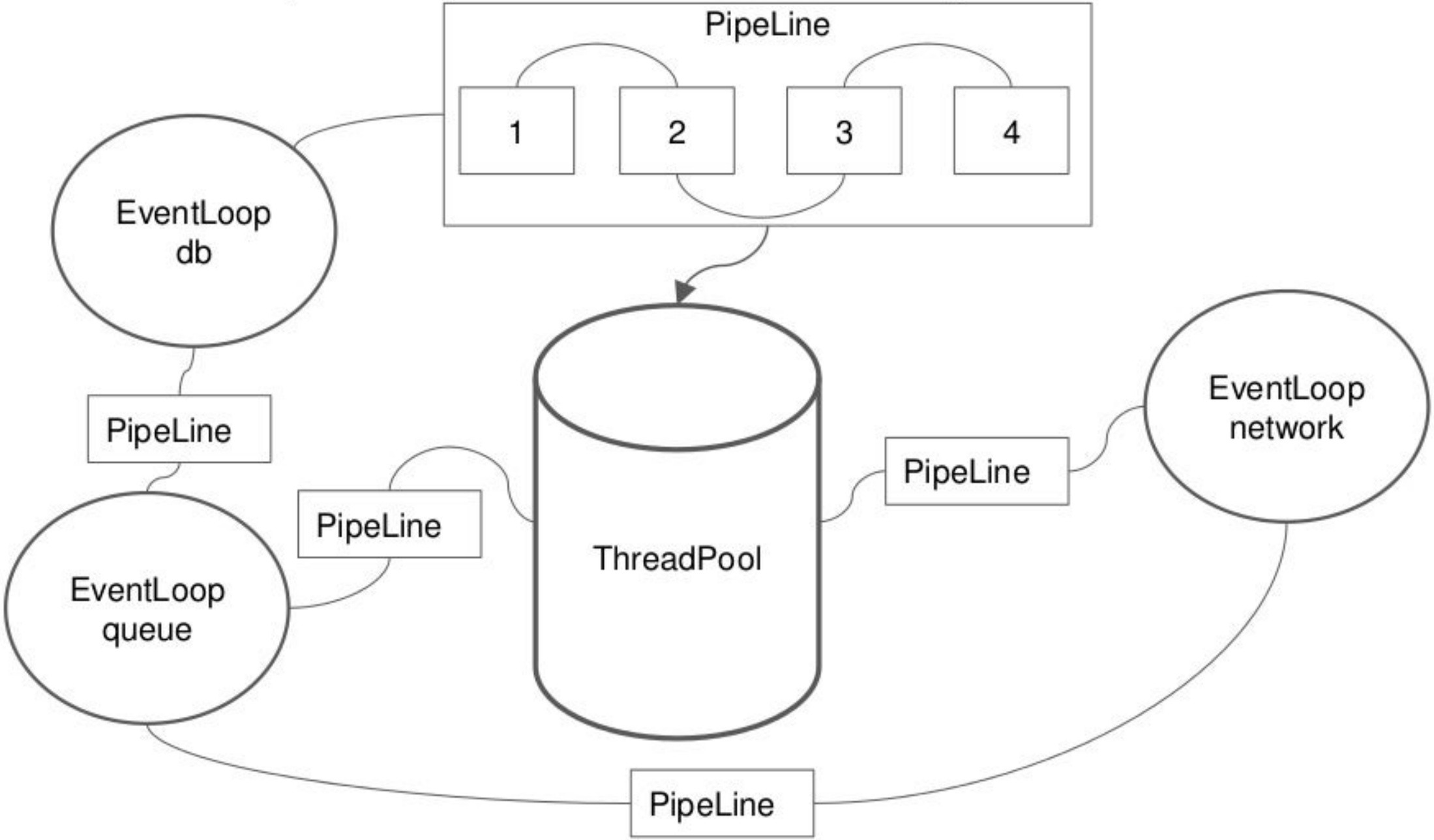
Intro Project Architecture



Intro Part of the project Architecture



Intro The components that hold the system



Intro

What models of concurrent programming are?

- Single Thread -> Nodejs, python3.4+
- Actor model -> Erlang, Elixir, jvm family, C#
- Communicating sequential processes -> go
- Software transactional memory -> clojure
- etc ...

Actor Model what kind of animal?

Formally, the description of the actor model

It has an ID on which the actor can be identified by other actors;

May interact with other actors only by sending messages with the identifiers of the actors addresses;

Implements its behavior depending on the incoming messages;

May create one or more new actors;

Can change its state;

Can terminate its execution.

https://en.wikipedia.org/wiki/Actor_model

What looks like an actor?

Smalltalk
Class

$\approx$

Actor Model
Actor

$\neq$

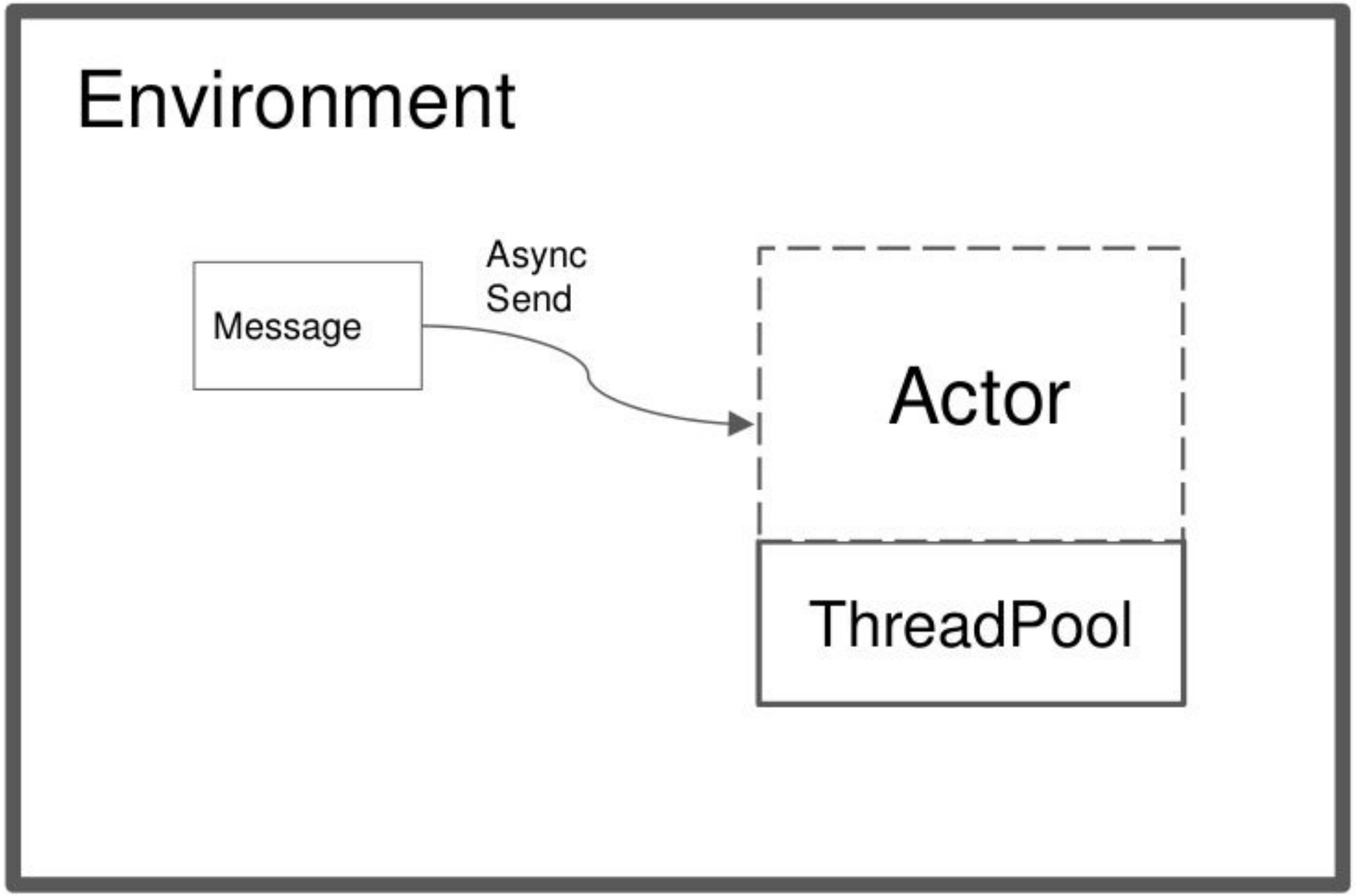
C++
OPP

What are the implementation?

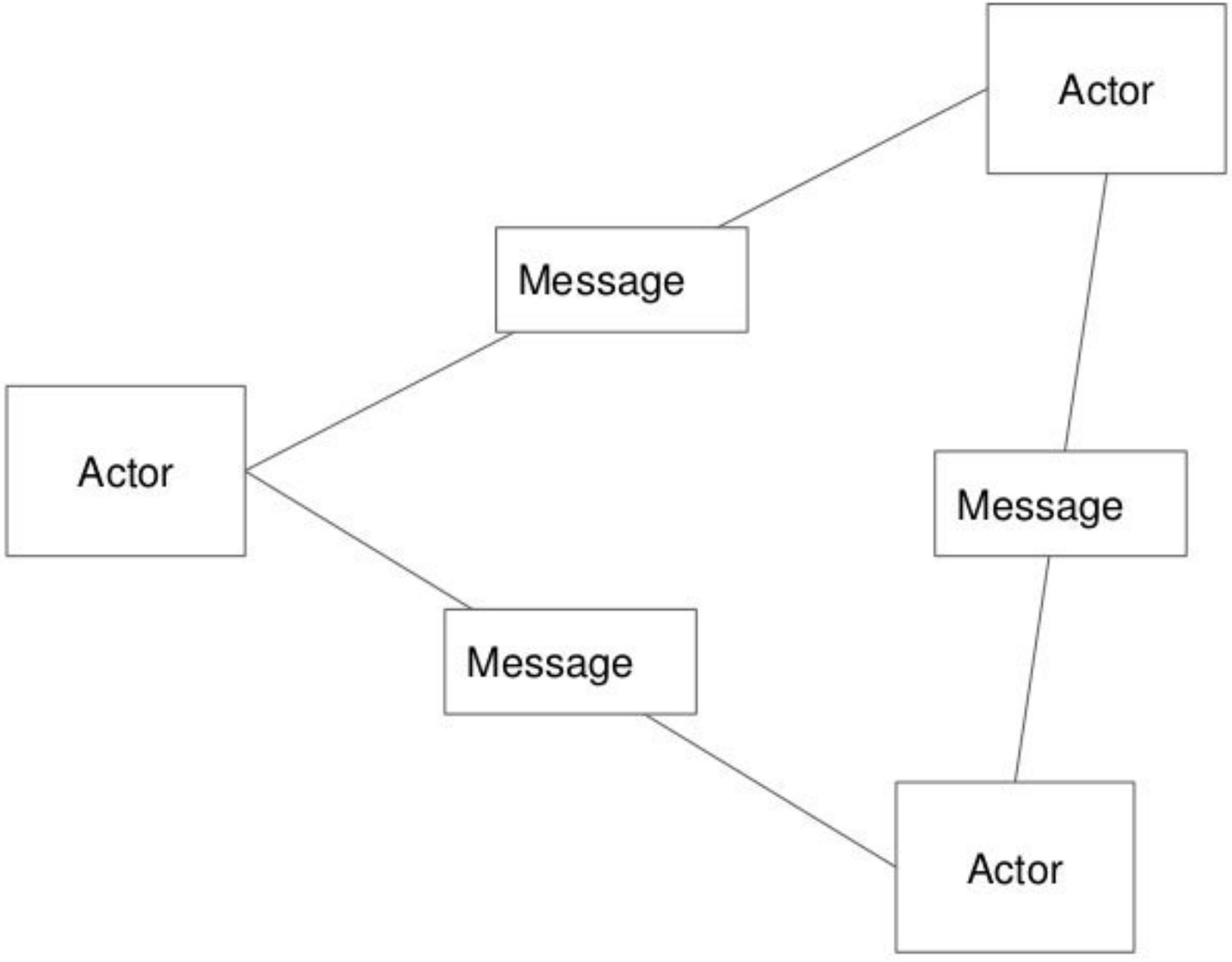
C++ Actor Framework: CAF
https://www.actor-framework.org/
CAF -> КАФ

SObjectizer
https://github.com/eao197/so-5-5
SObjectizer -> собжектайзер

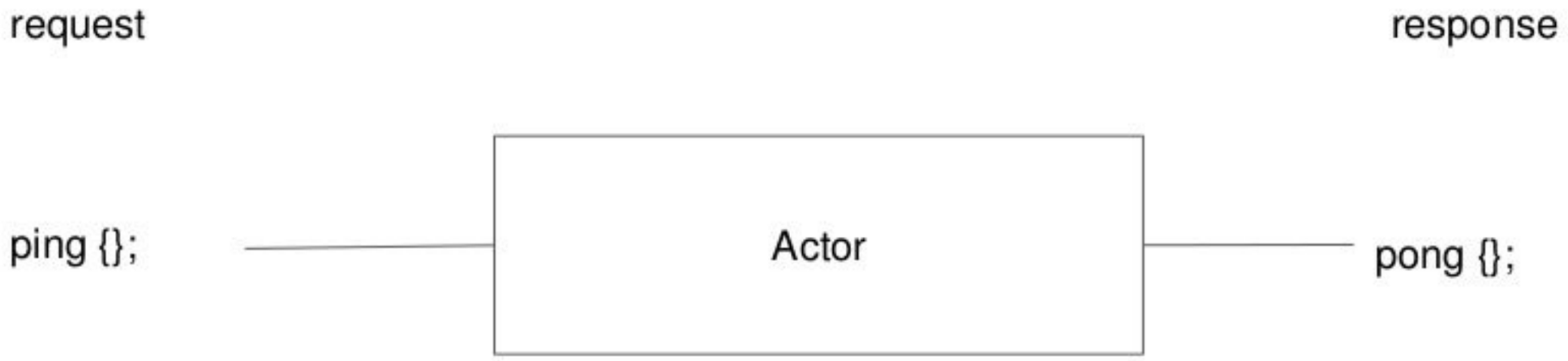
General Schematic

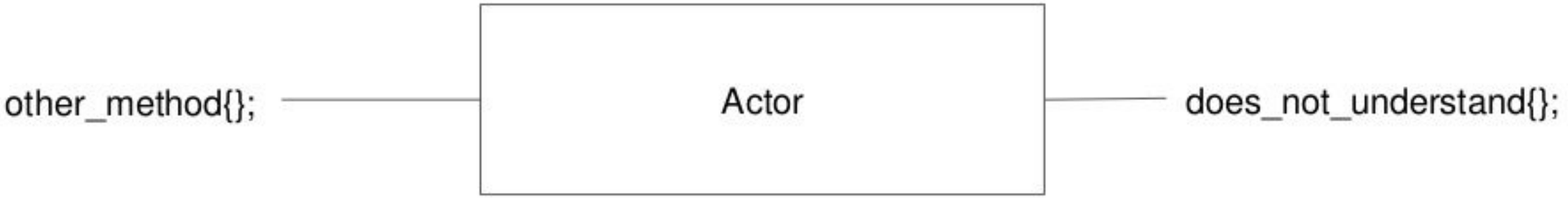


One of the main ideas of the actor model



One of the main ideas of the actor model

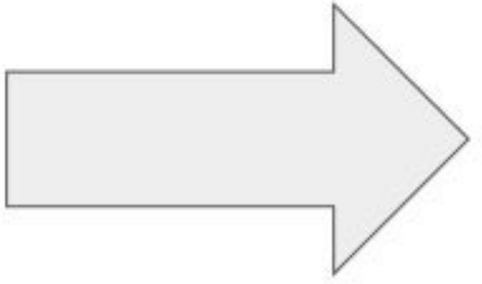




Analogy

http://mysuperservice.ru/update?d=1&ds=3

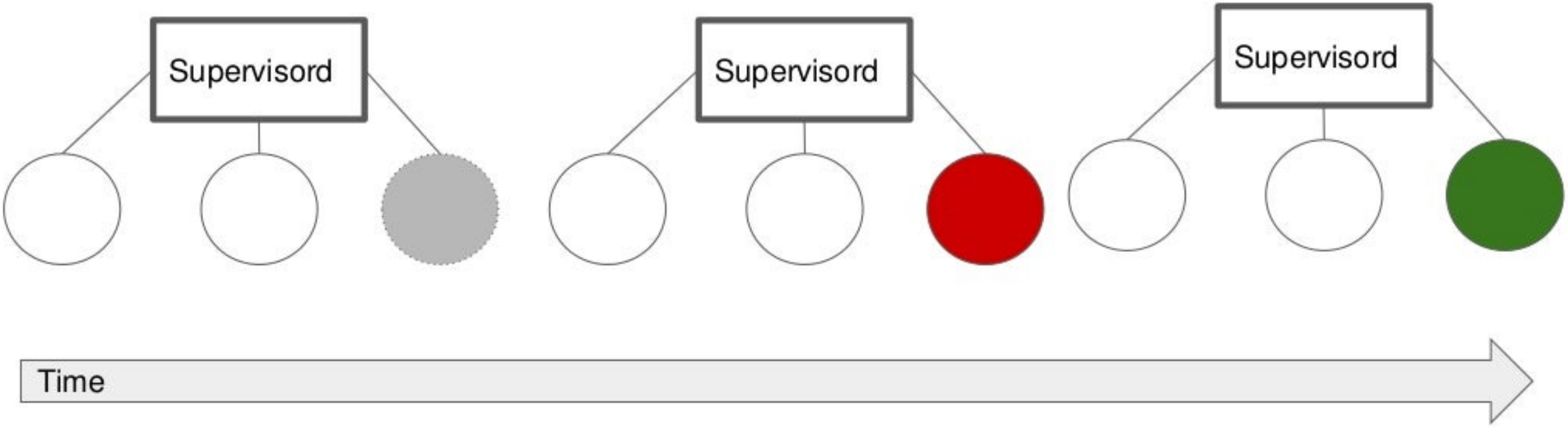
Browse



service

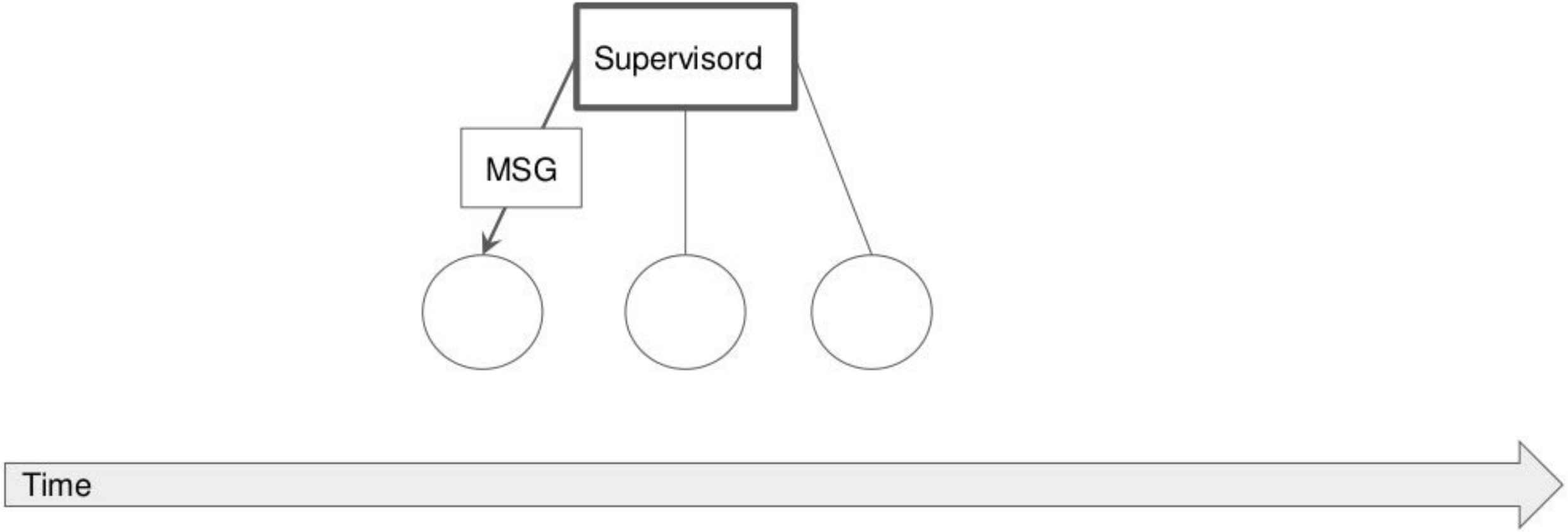
Supervisor

Case : restored after the fall of the actor

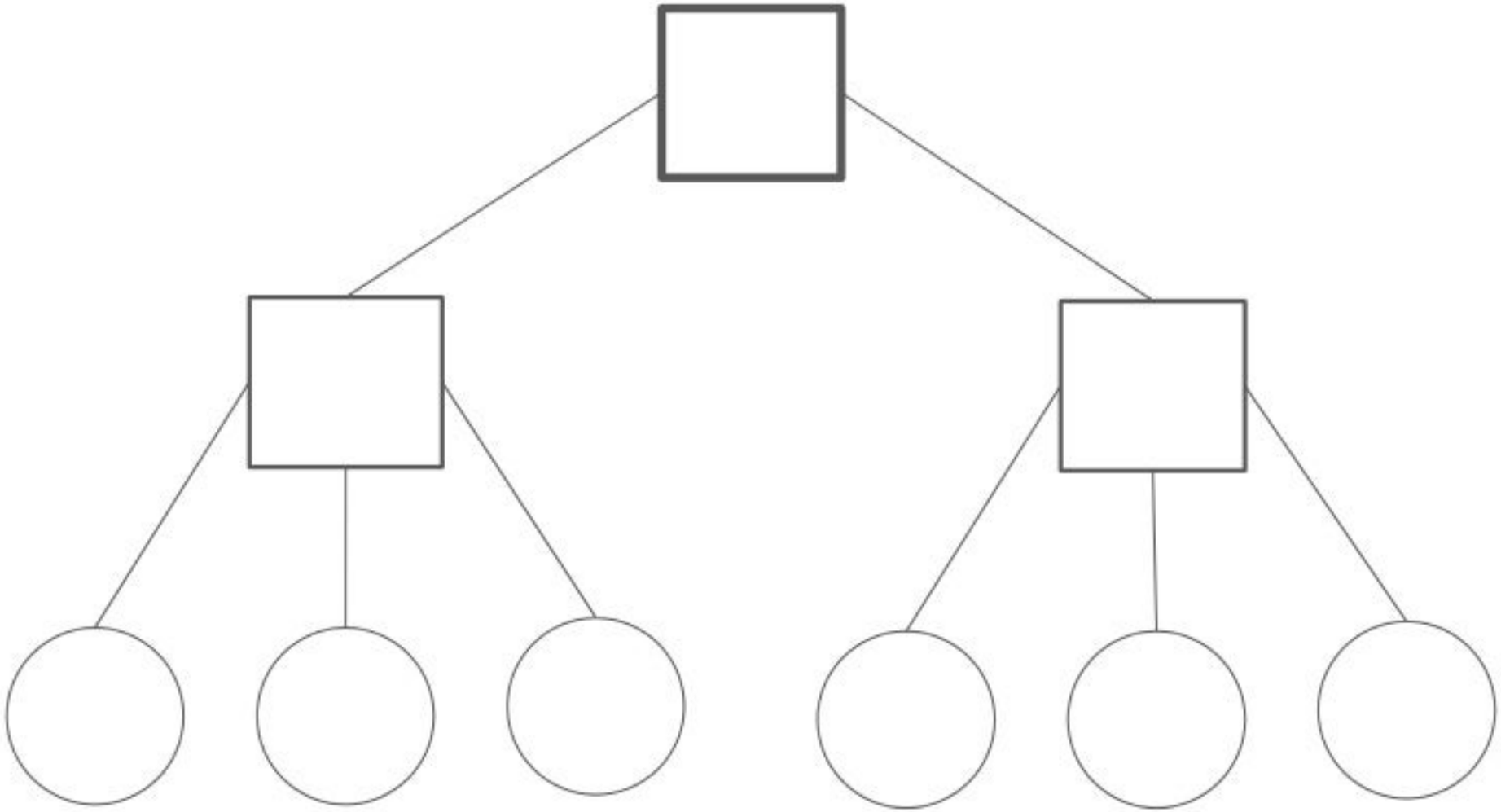


Supervisor

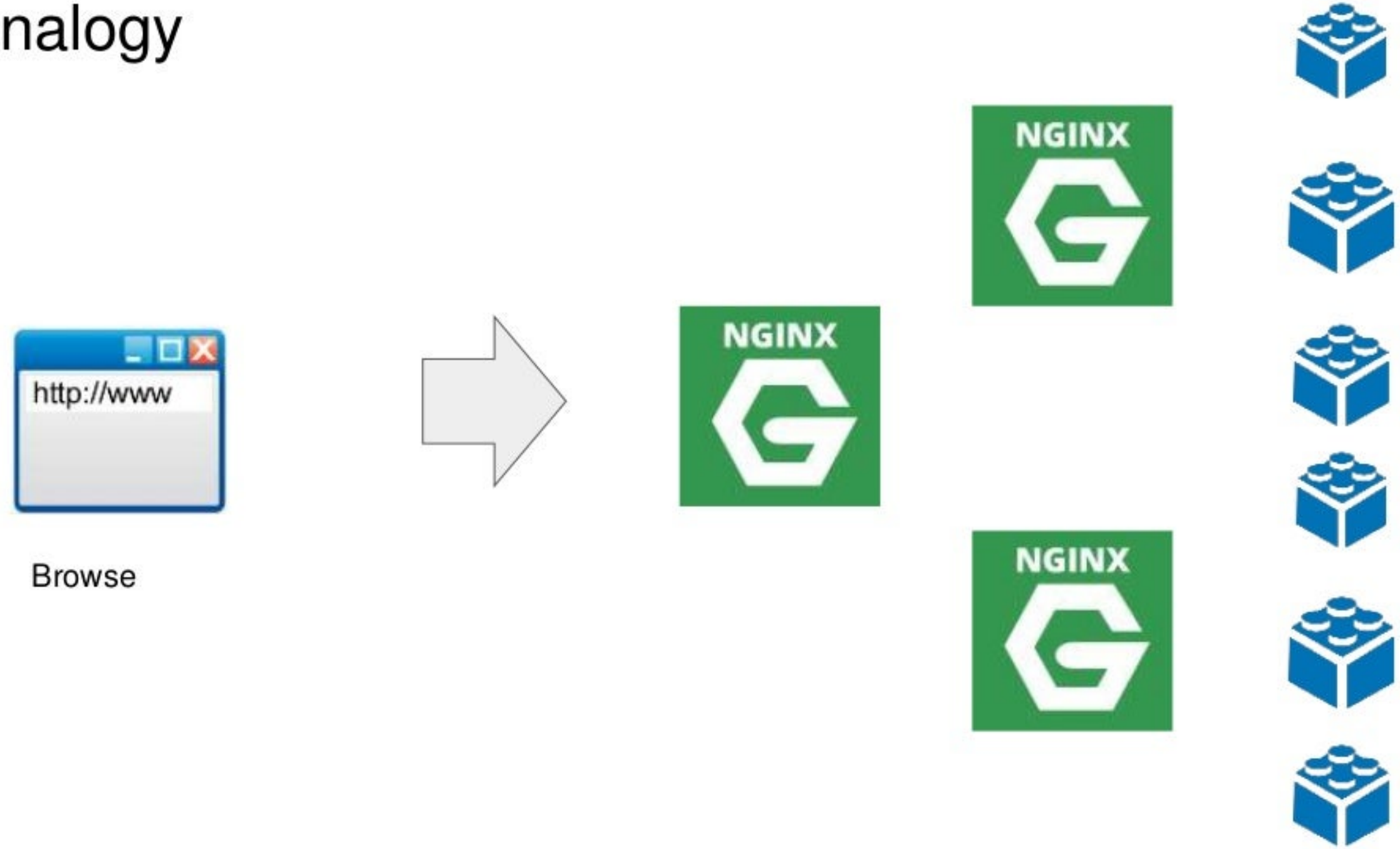
Case : balancing messaging



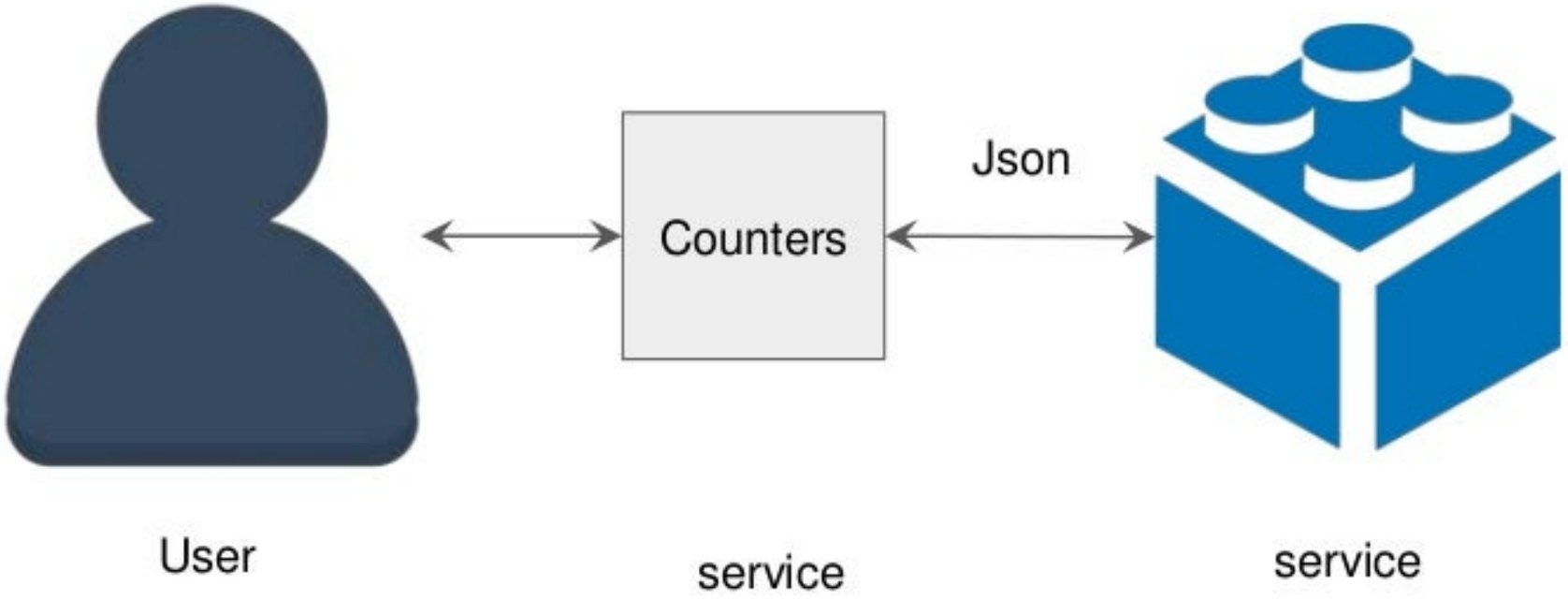
Supervisor



Analogy



Description of a problem Productions



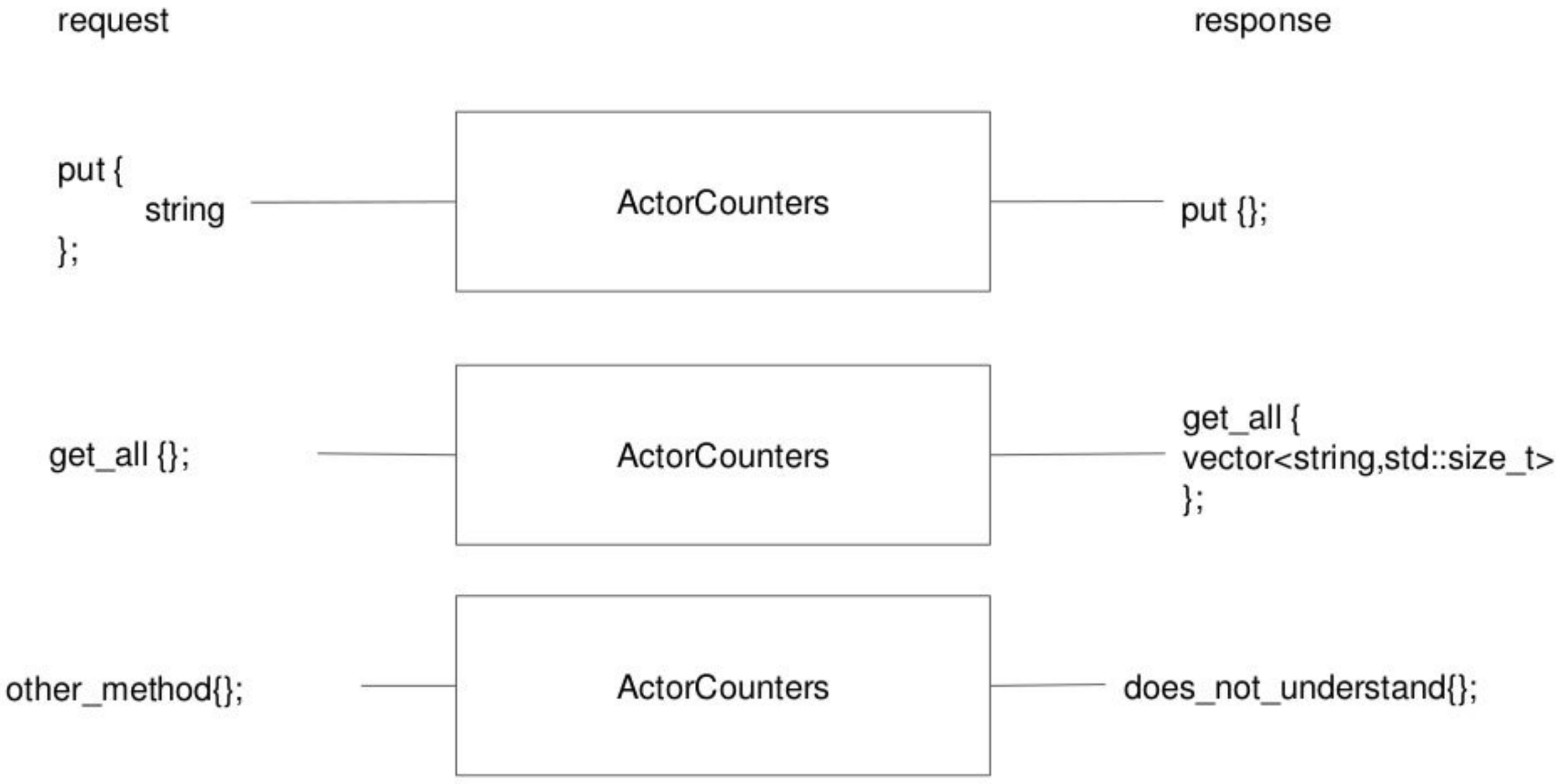
Case 1:

We obtain the unique identifier and increase the counter by one

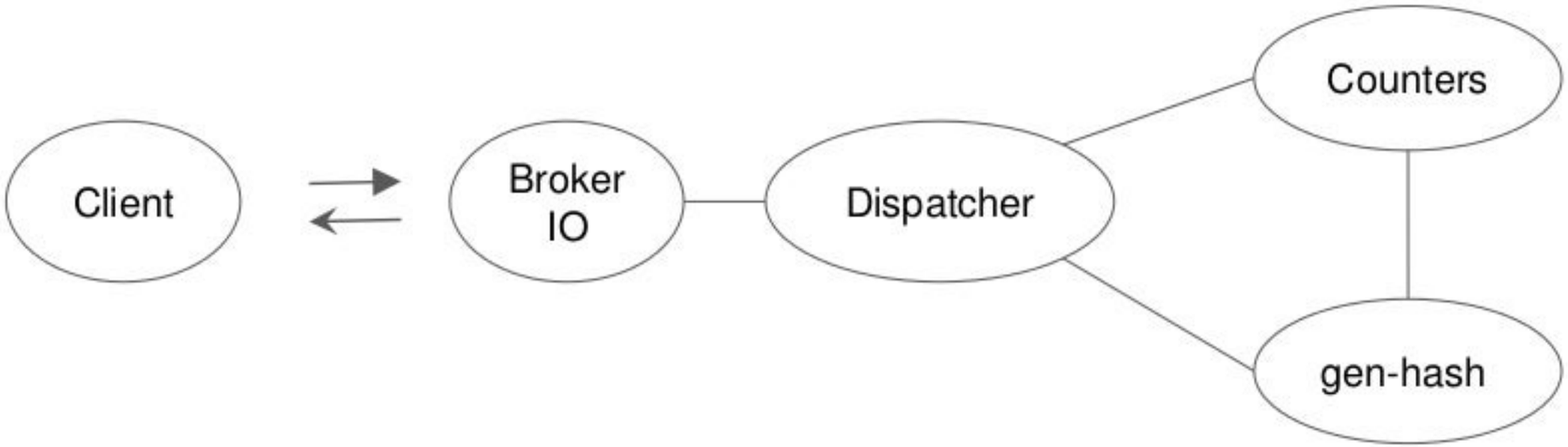
Case 2:

Return a list of all the stored hash from uid and the counter

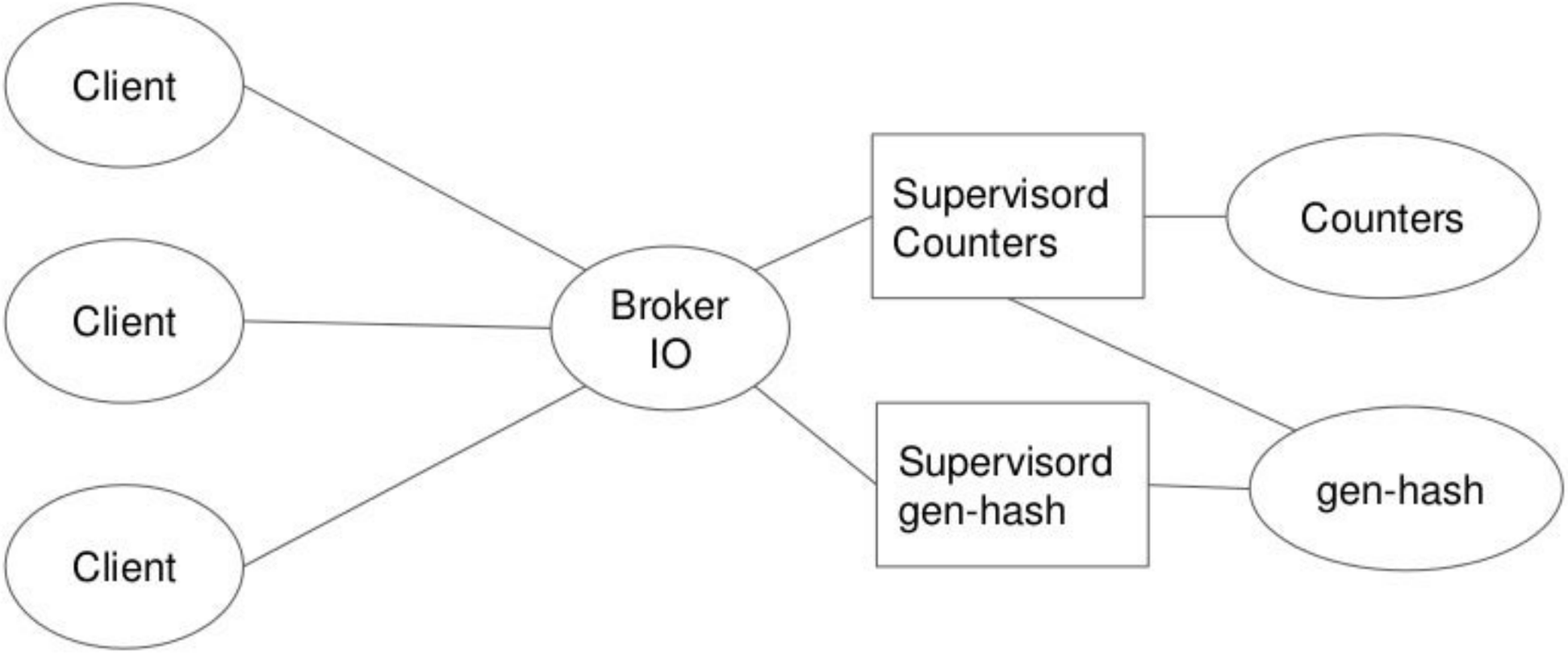
Description of possible types of messages.



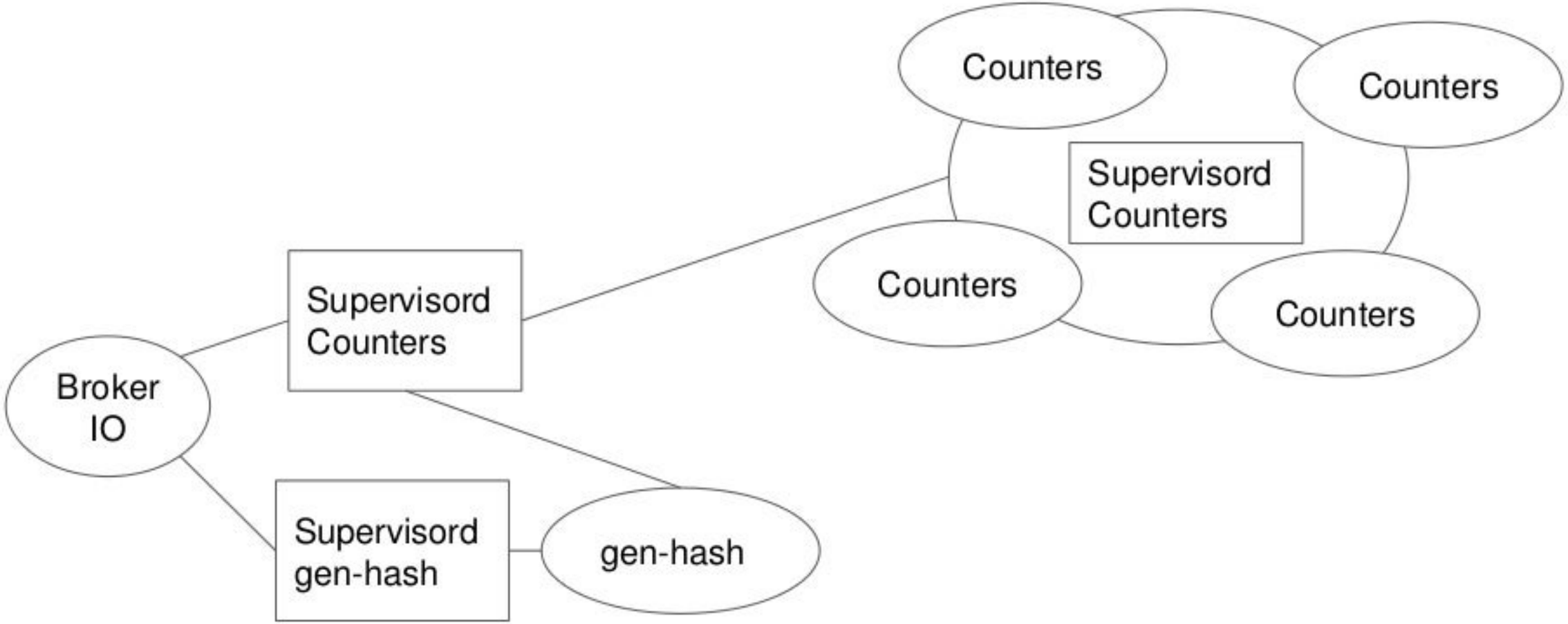
Scheme of the initial solution



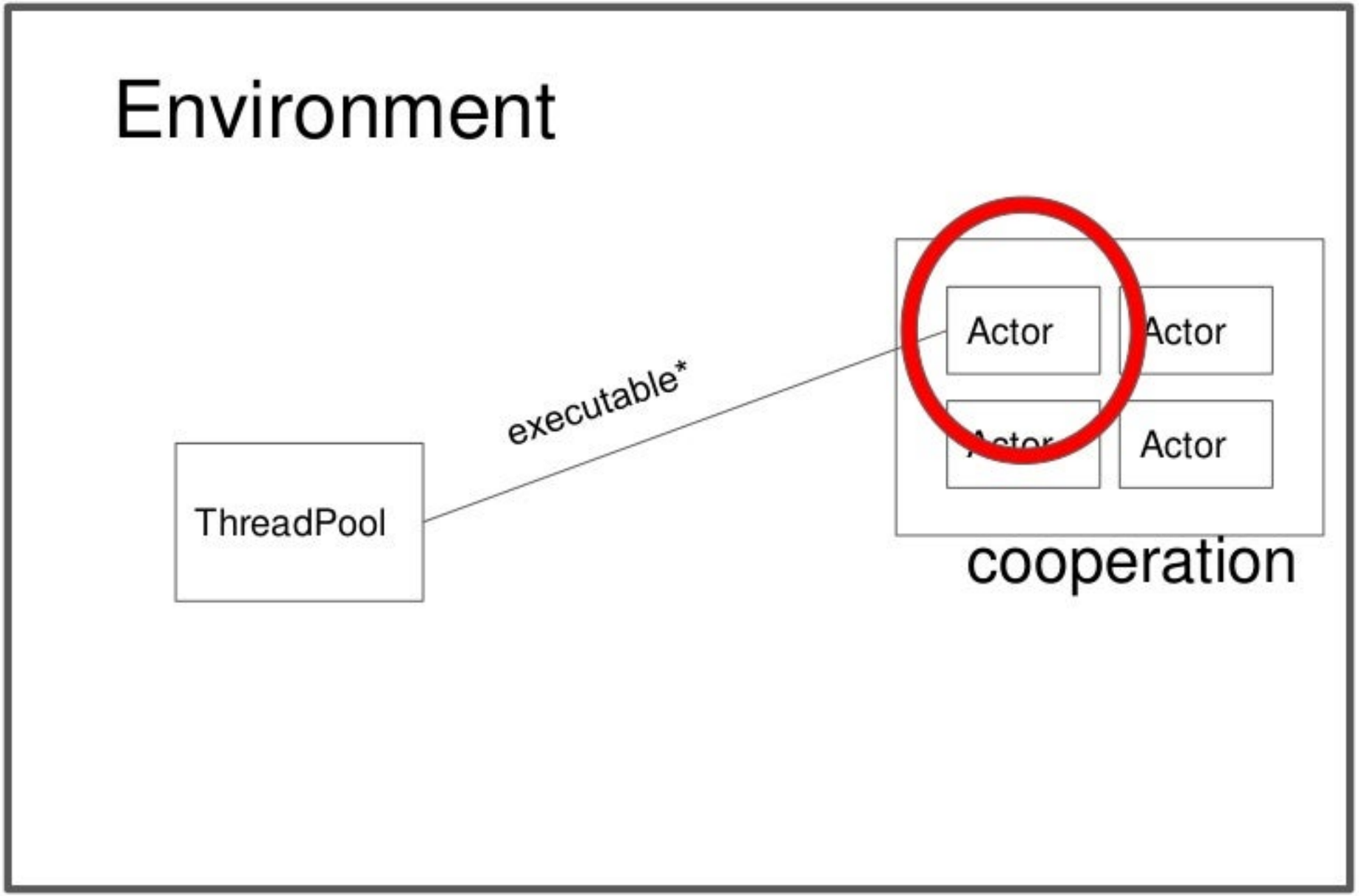
Driving the initial decision to supervisord



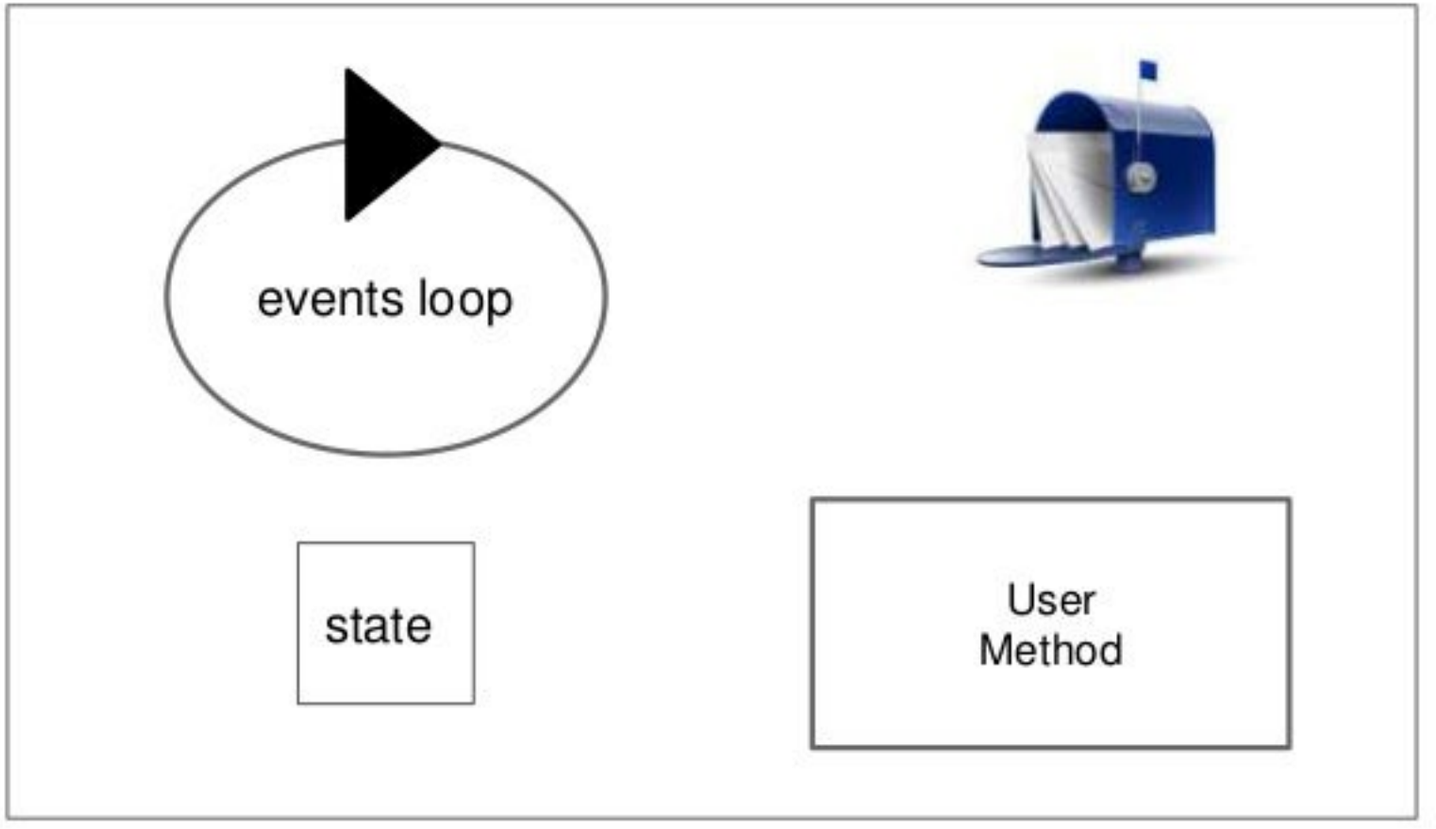
Driving the initial decision to supervisord



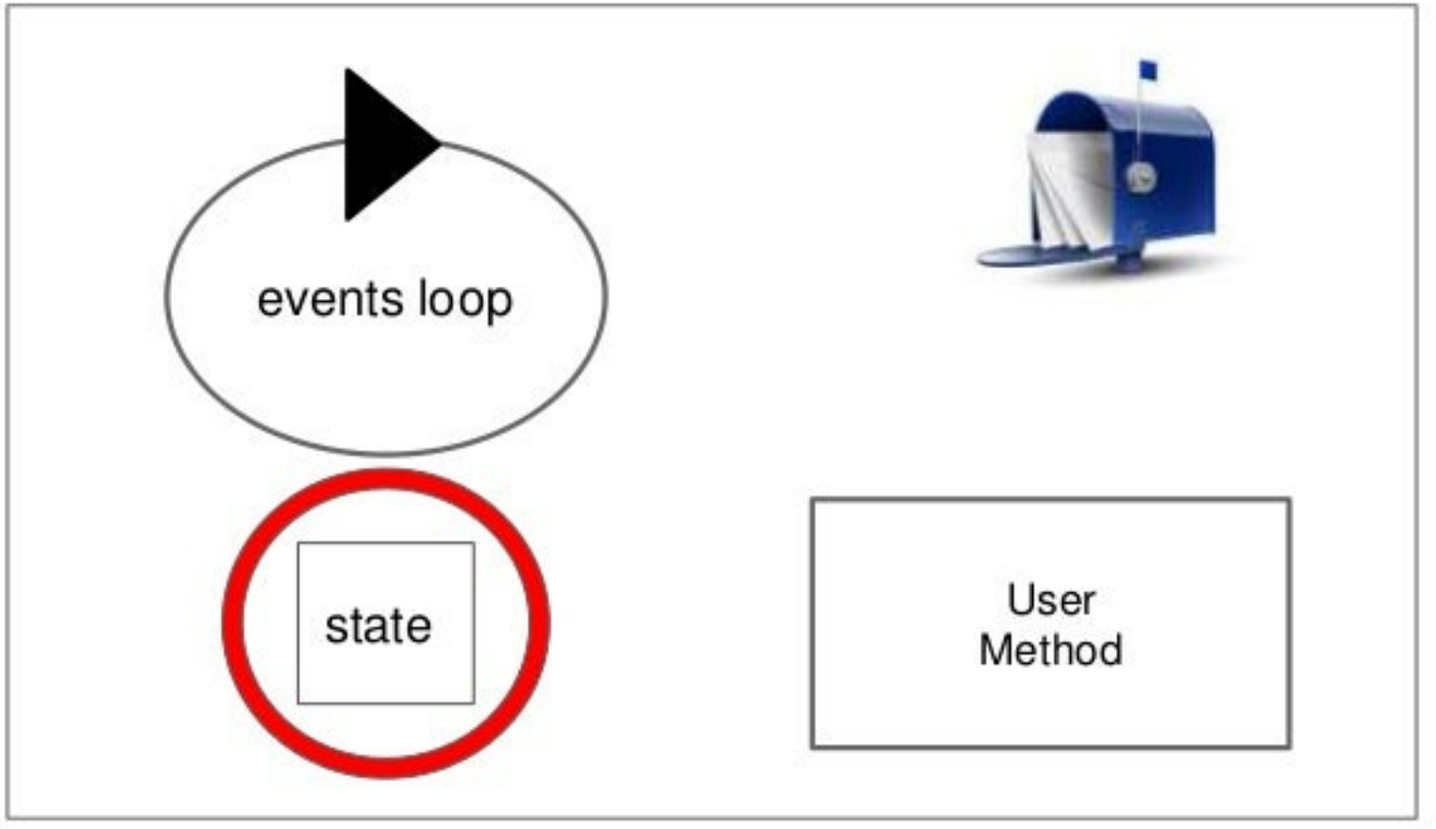
General Schematic



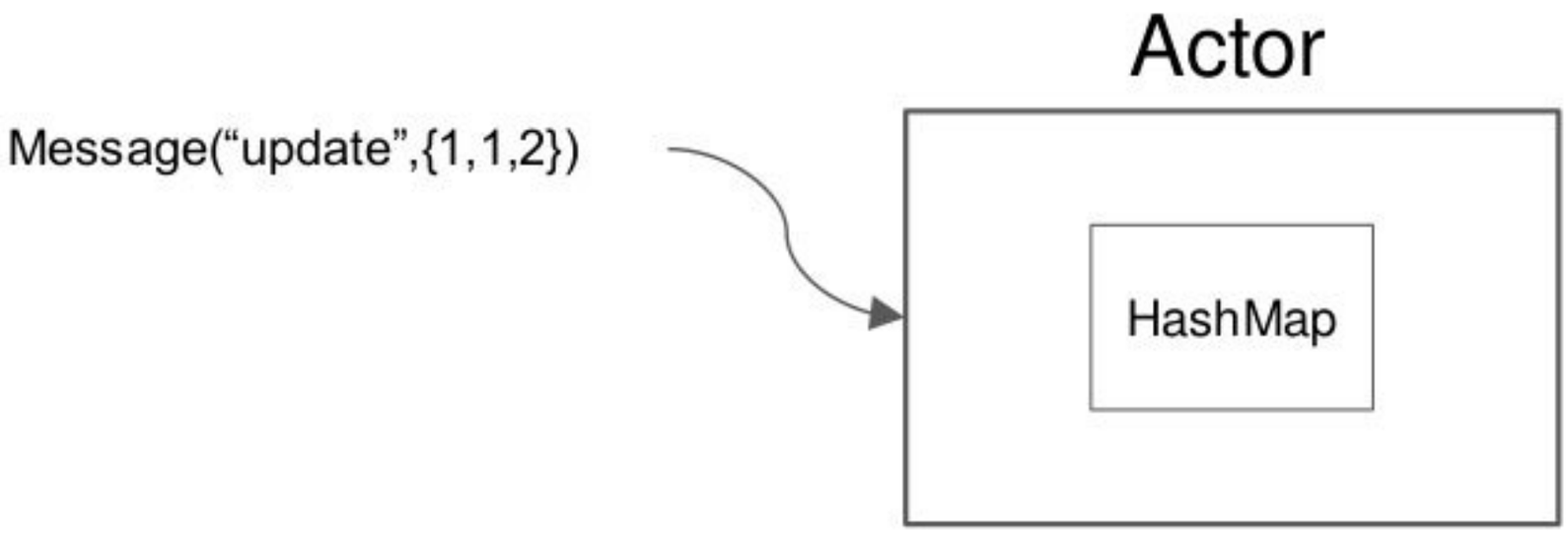
Actor



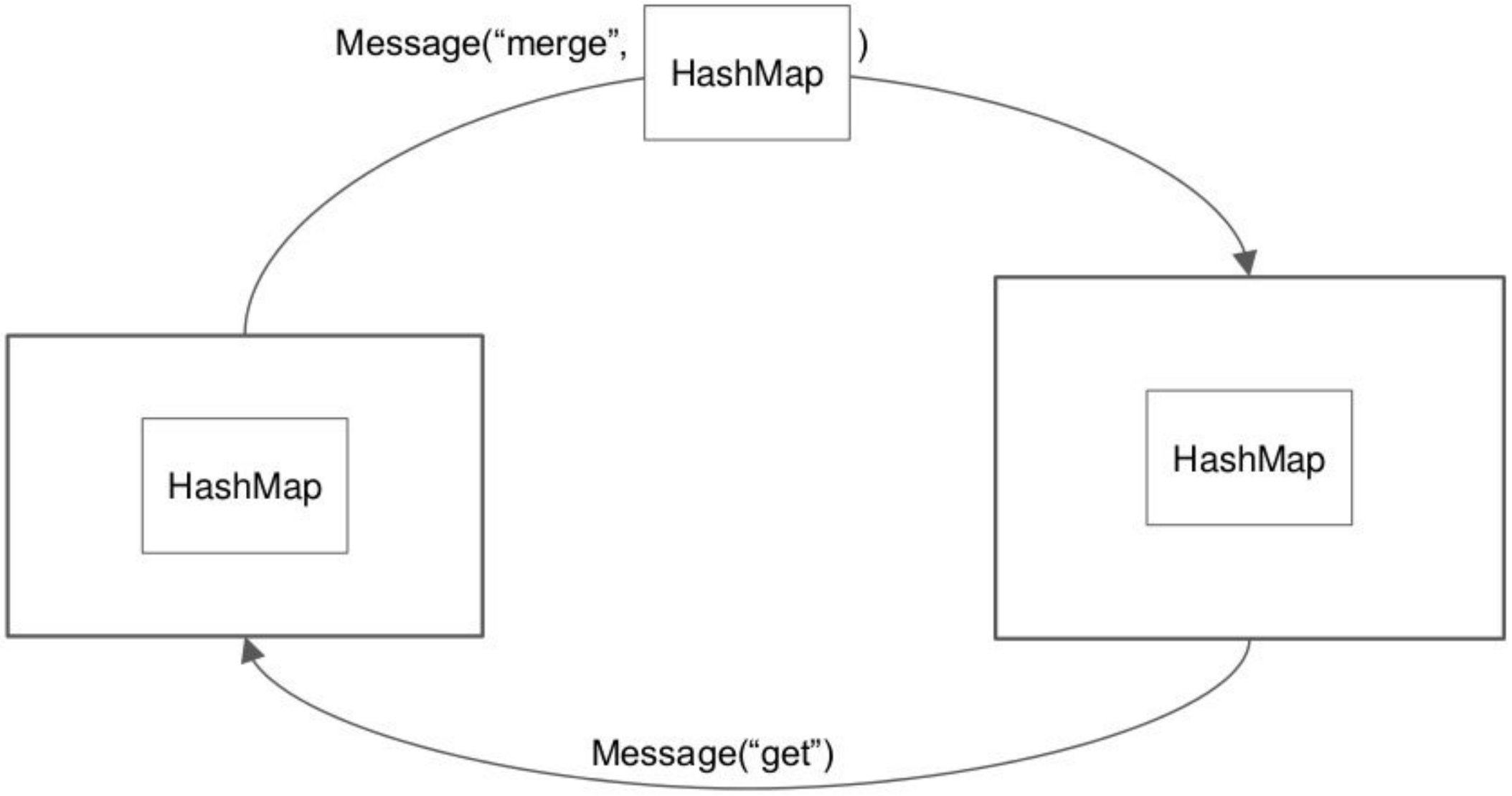
Actor



Data locality and Isolated

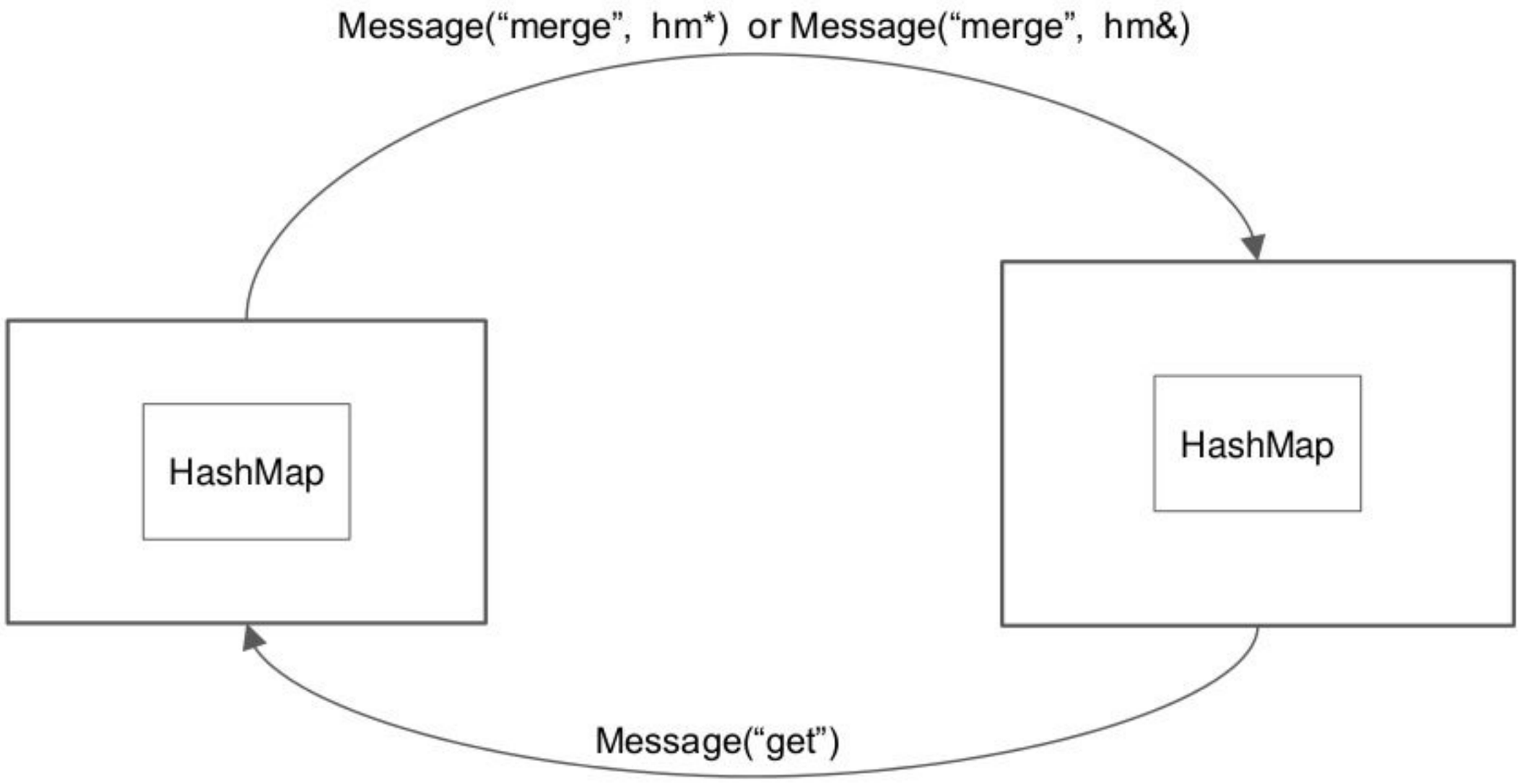


Data locality and Isolated

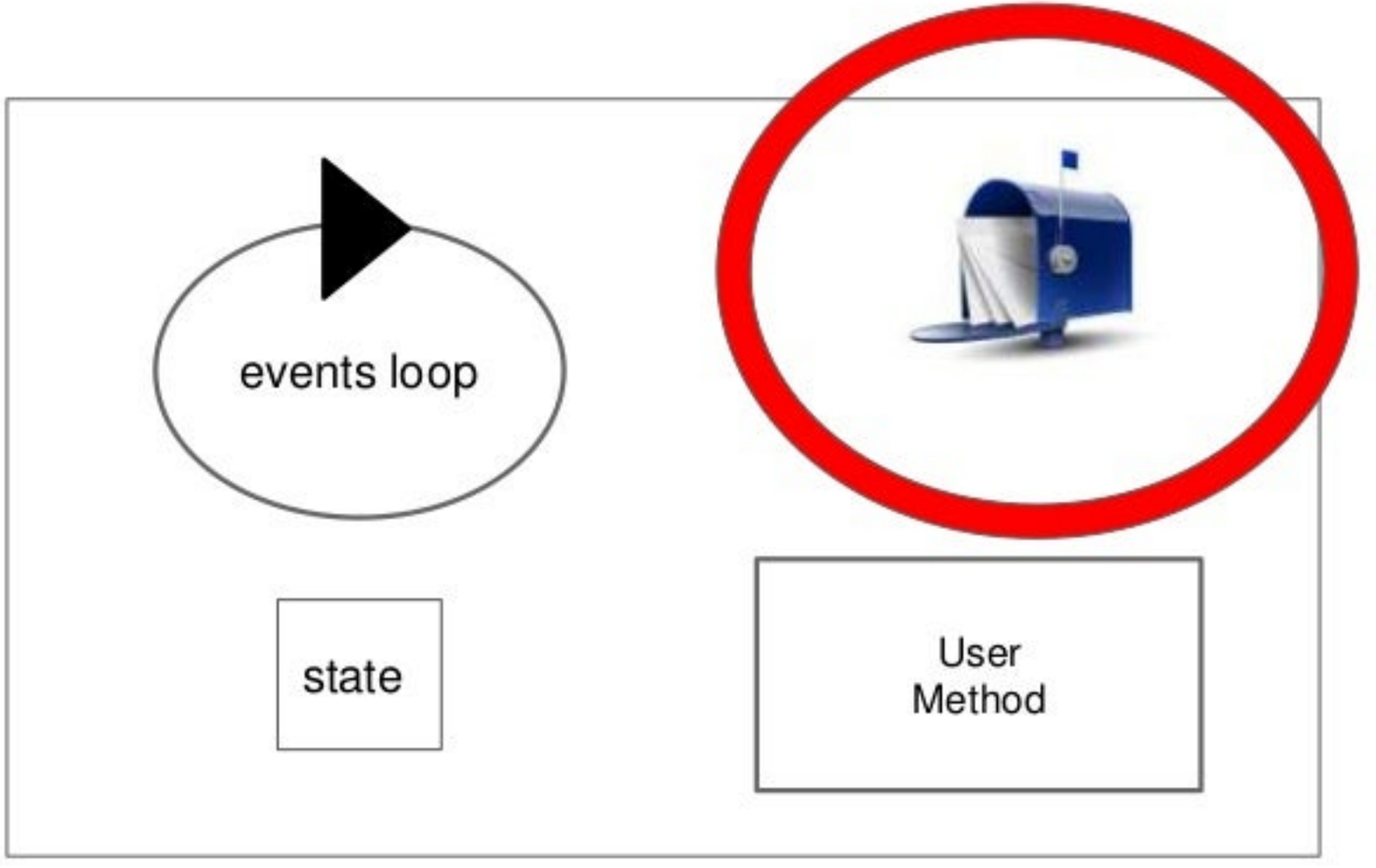


Shared State

~~Data locality and Isolated~~ Shared State



Actor



Queue …

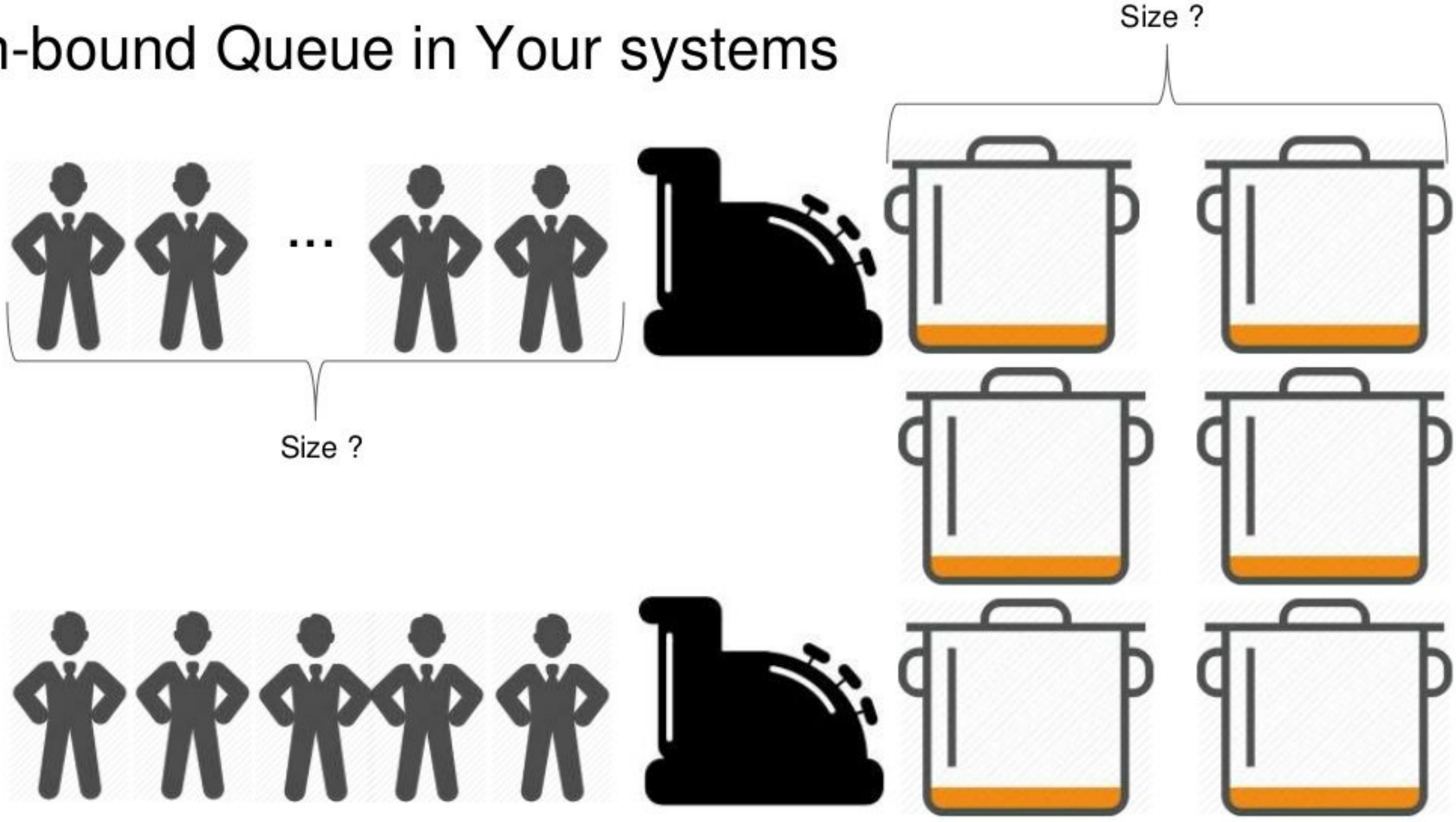
Un-bound Queue ????

Un-bound Queue in Your systems

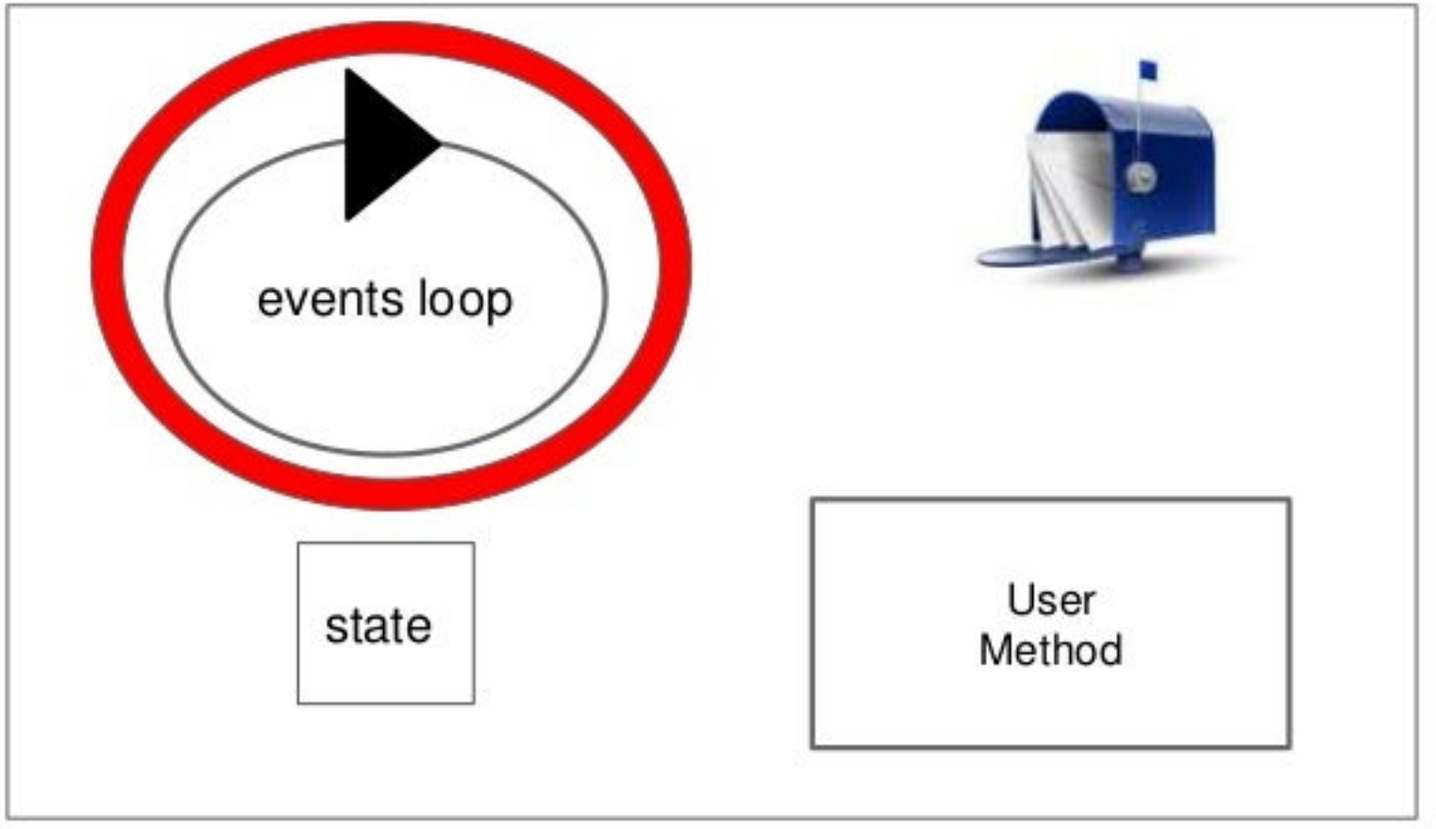
≈ System + Un-bound Queue

Key word:**Queueing theory**

Un-bound Queue in Your systems



Actor



EventLoop in Actor blocking approach

```
void run(args ..., args ...,)  {

        message *msg_ptr = nullptr;
        while (has_next_message()) {
            msg_ptr = next_message();
            if (msg_ptr != nullptr) {
        ****************************************
                auto response = life.invoke(msg_ptr);
        ****************************************
                        } else {
        ****************************************
        }
    }
```

EventLoop in Actor Pseudo non-blocking

```
auto run(args ..., size_t max_throughput, args ...,)  -> event_type {
        message *msg_ptr = nullptr;
        for (size_t handled_msgs = 0; handled_msgs < max_throughput;) {
                msg_ptr = next_message();
                if (msg_ptr != nullptr) {
                ************************************
                auto response = life.invoke(request);
                ************************************
                ++handled_msgs;
                ************************************
                } else {
                        return executable_result::awaiting;
                }
        }
        return executable_result::resume;
}
```

EventLoop in Actor Pseudo non-blocking

```
auto run(args ..., size_tmax_throughput, args ...,)  -> event_type {
        message *msg_ptr = nullptr;
        for (size_t handled_msgs = 0; handled_msgs < max_throughput;) {
                msg_ptr = next_message();
                if (msg_ptr != nullptr) {
                ************************************
                auto response = life.invoke(request);
                ************************************
                ++handled_msgs;
                ************************************
                } else {
                        return executable_result::awaiting;
                }
        }
        return executable_result::resume;
}
```

EventLoop in Actor Pseudo non-blocking

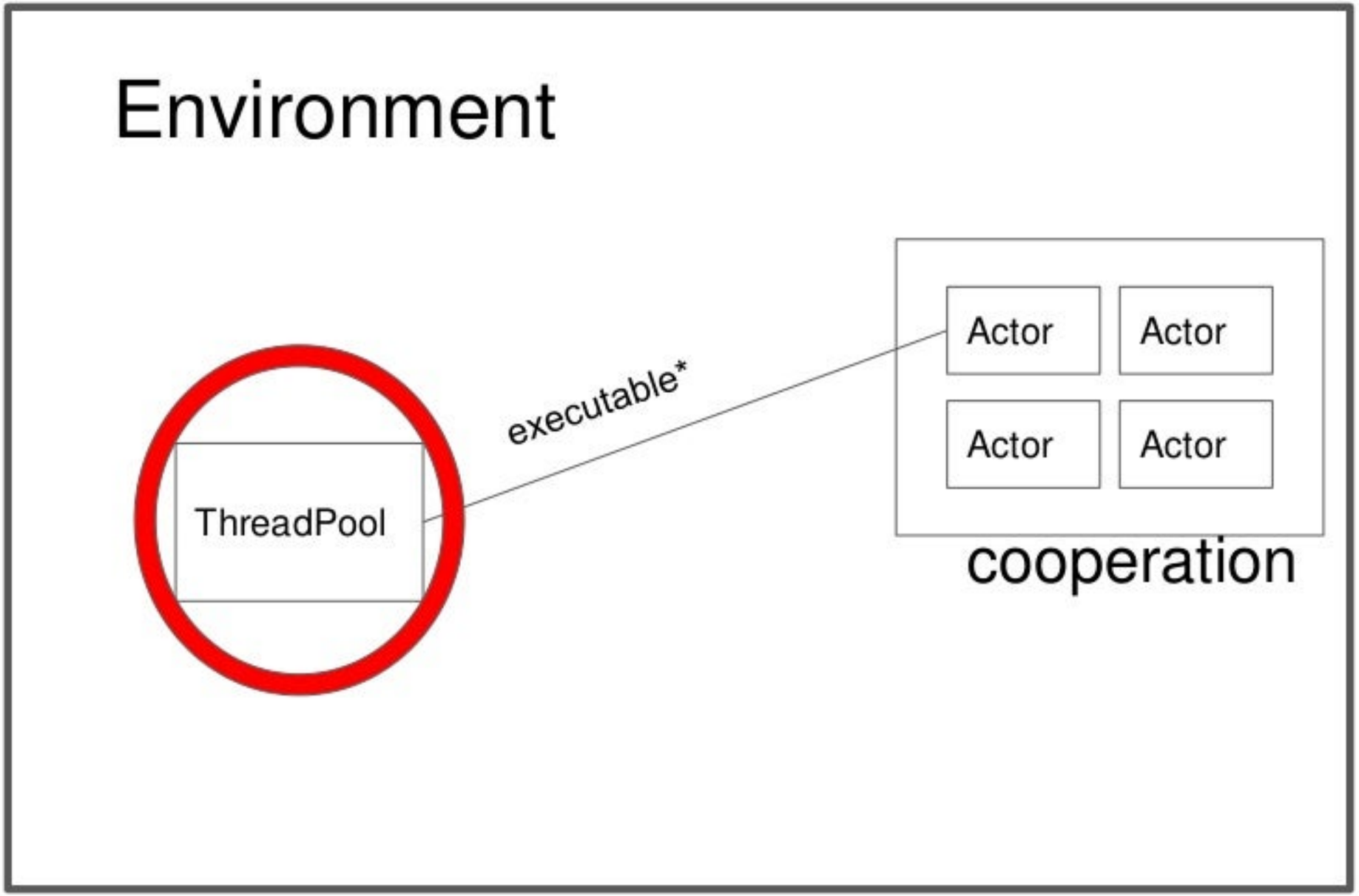
```
struct magick_method final : public abstract_action {
        ************************************
        response *operator()(request *msg) override final {
        ************************************
        a lot of data processing
        ************************************
        return nullptr;
        }
};
```

EventLoop in RunOnce

How to run once?

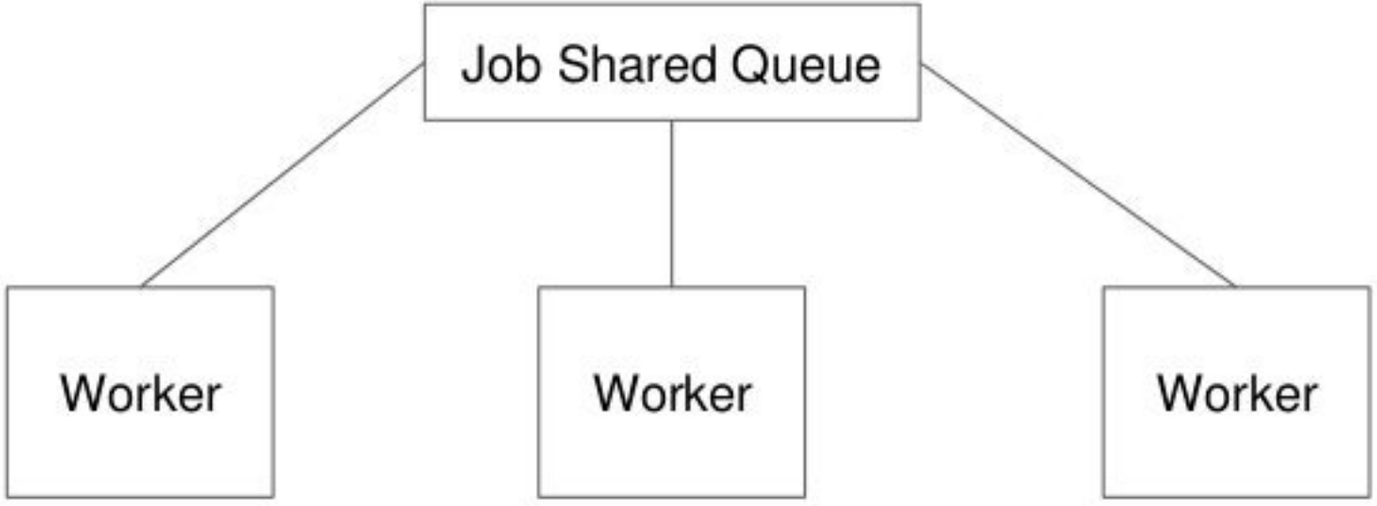
```
auto run_once(args ..., args ...,) {
        return run(args ..., 1, args ...,)
}
```

General Schematic



EventLoop in Actor System

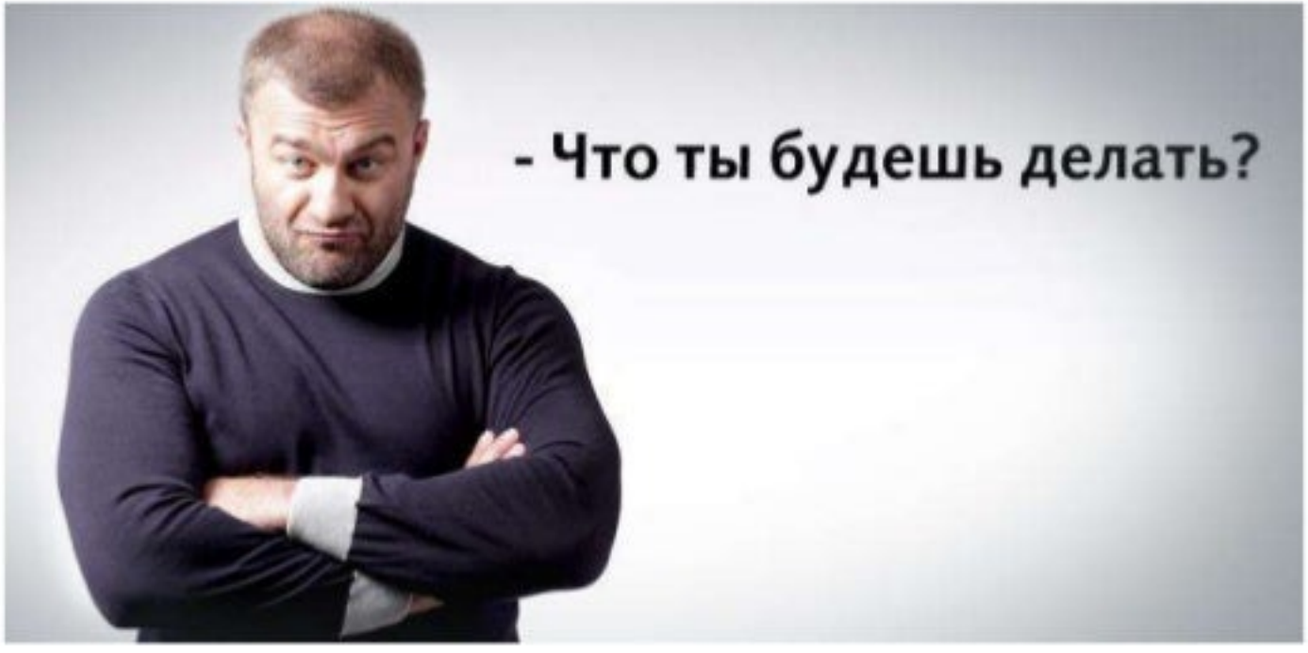
Strategy: shared-work



EventLoop in Actor System

Strategy: shared-work

EventLoop in Actor System



EventLoop in Actor System

And what are the tasks scheduling strategy?

	asynchronous	synchronous
proactive	**Work-balancing**	**Work-distribution**
reactive	**Work-stealing**	**Work-requesting**

EventLoop in Actor System

And what are the tasks scheduling strategy?

	asynchronous	synchronous
proactive	**Work-balancing**	**Work-distribution**
reactive	**Work-stealing**	**Work-requesting**

EventLoop in Actor System

SCHEDULER = (STEALING ^ REQUESTING) [+DISTRIBUTION] [+BALANCING]

	asynchronous	synchronous
proactive	**Work-balancing**	**Work-distribution**
reactive	**Work-stealing**	**Work-requesting**

Reality

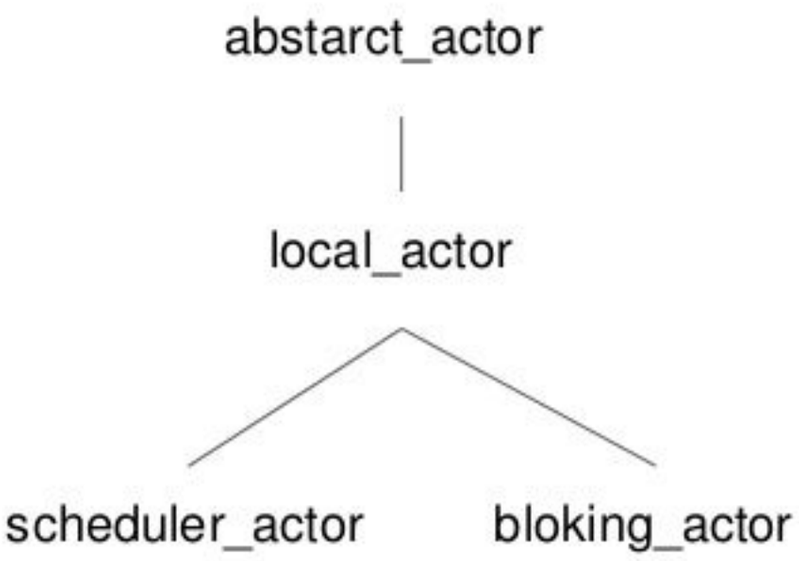
Possible applications:

- back-end game dev;
- micro-service;
- service;
- macro-service;
- distribution system;
- etc … .

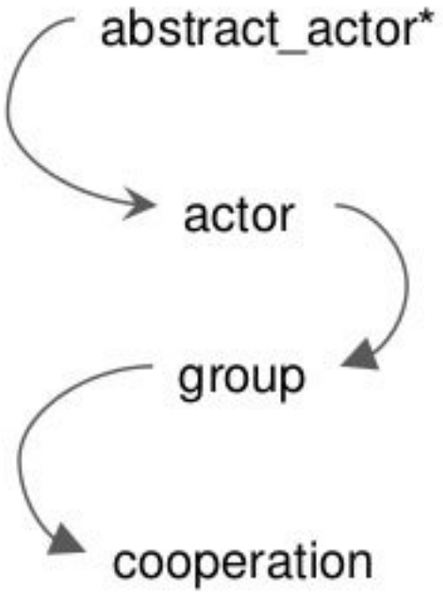
? Questions ?

Bonus slide

Inheritance Hierarchy



The hierarchy of nesting



Bonus slide

What do you want to change?

abstarct_actor

local_actor

base_supervisor vs fake_supervisor

Pseudo non-blocking aka async vs blocking

```
template<ParentActor=local_actor, Supervisor=base_supervisor, async=true  >
scheduler_actor:  public ParentActor,
                                    public Supervisor,
                  public executable{};
```

Urls:

http://www.ponylang.org/

http://junior.highload.ru/2015/#301215

http://www.highload.ru/2015/abstracts/1964.html

https://en.wikipedia.org/wiki/Actor_model

https://en.wikipedia.org/wiki/Work_stealing

http://www.1024cores.net

http://www.slideshare.net/gpadovani/c-actor-model

https://habrahabr.ru/post/216049/