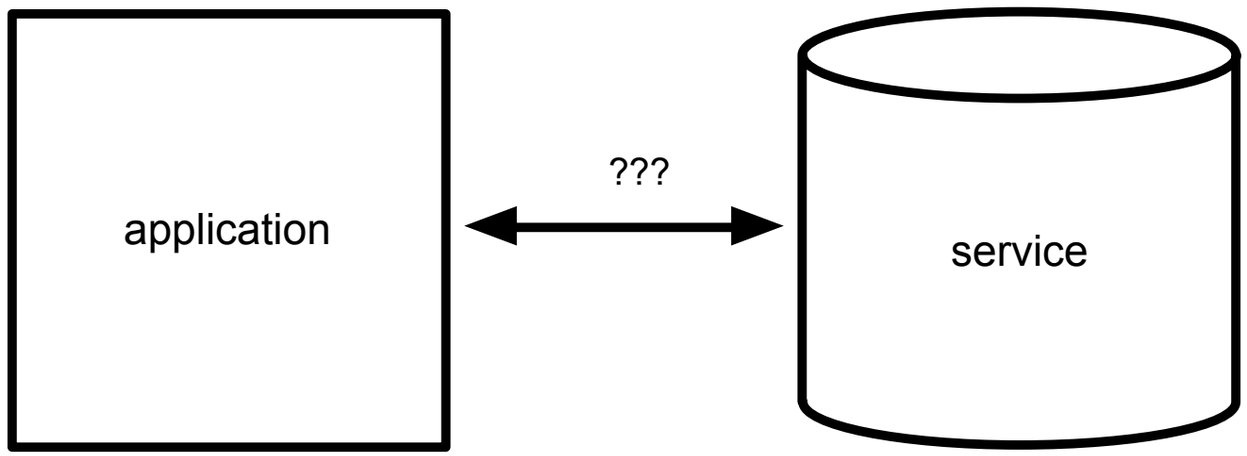
Building high-performance Erlang clients using Shackle

Louis-Philippe Gauthier
Director, Product Engineering

adgear

What…?

Problem



adgear

Service…?

- communicates over socket
- ascii or binary protocols
- synchronous or asynchronous protocols

e.g. cassandra, memcached, kafka, http2

adgear

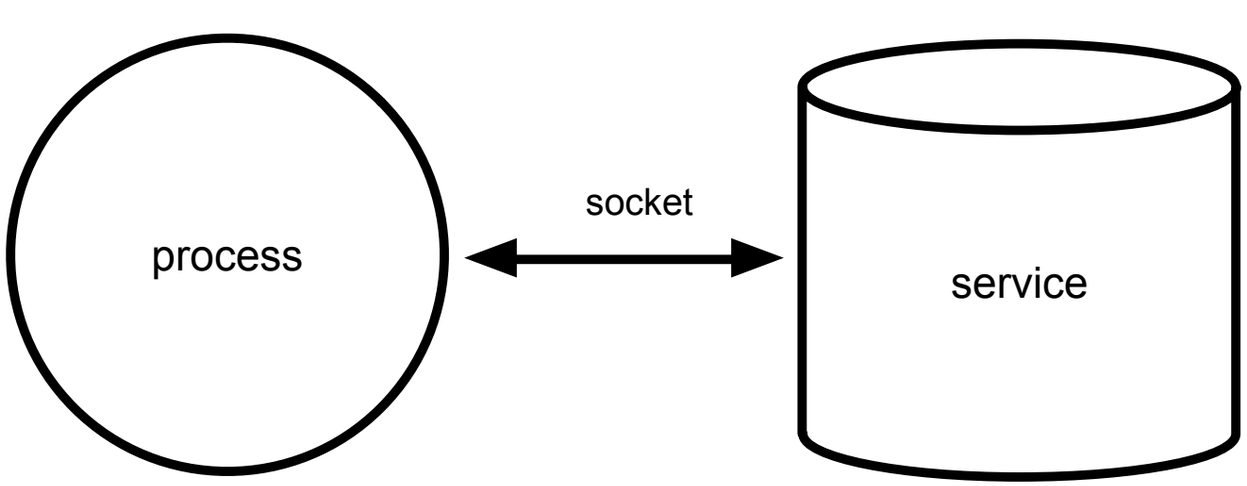
Goals?!

- reusability
- speed
- concurrency
- safety

adgear

Design process

Design #1



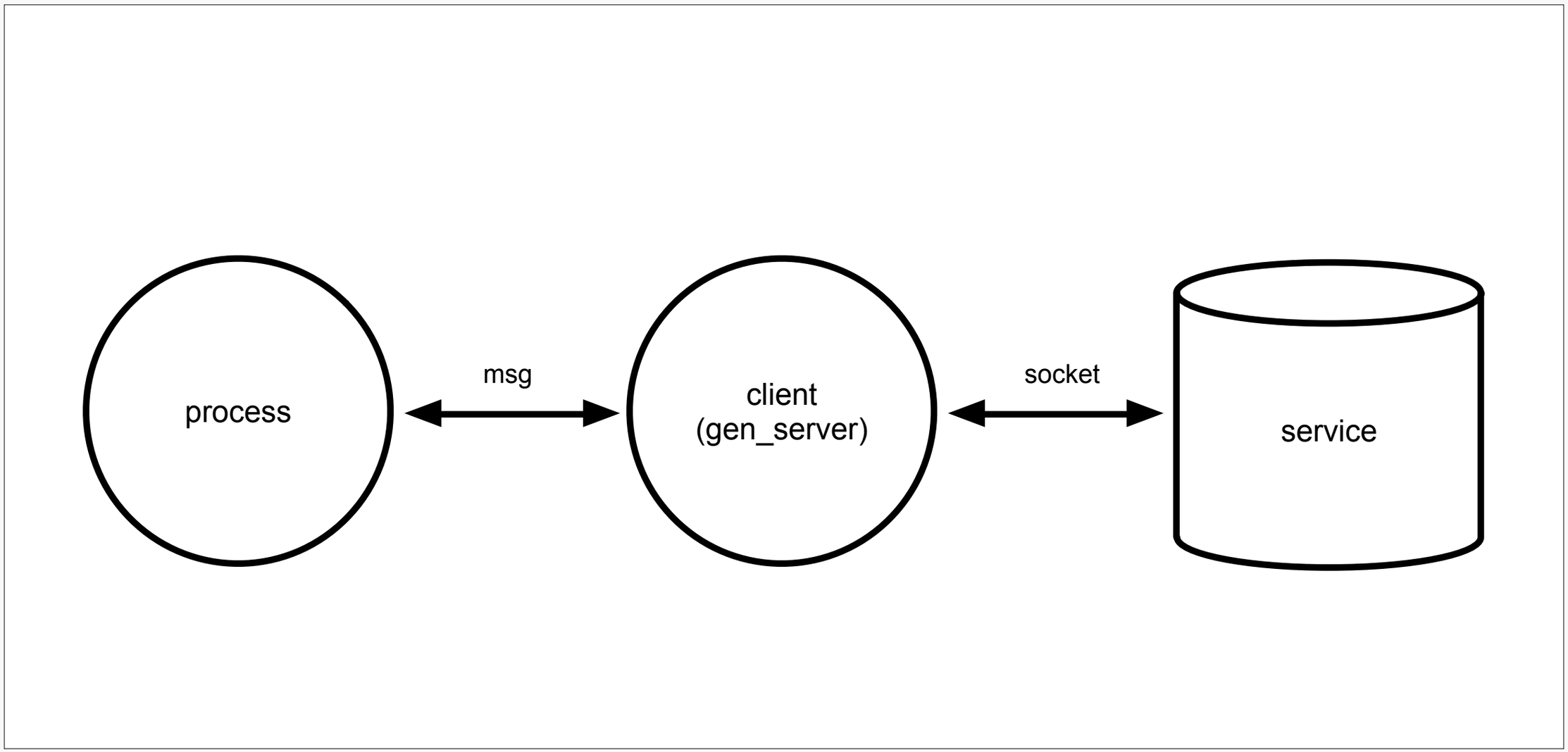
adgear

Design #1 problems

- connect / open overhead
- setup overhead
- no connection limit (one per request)

adgear

Design #2



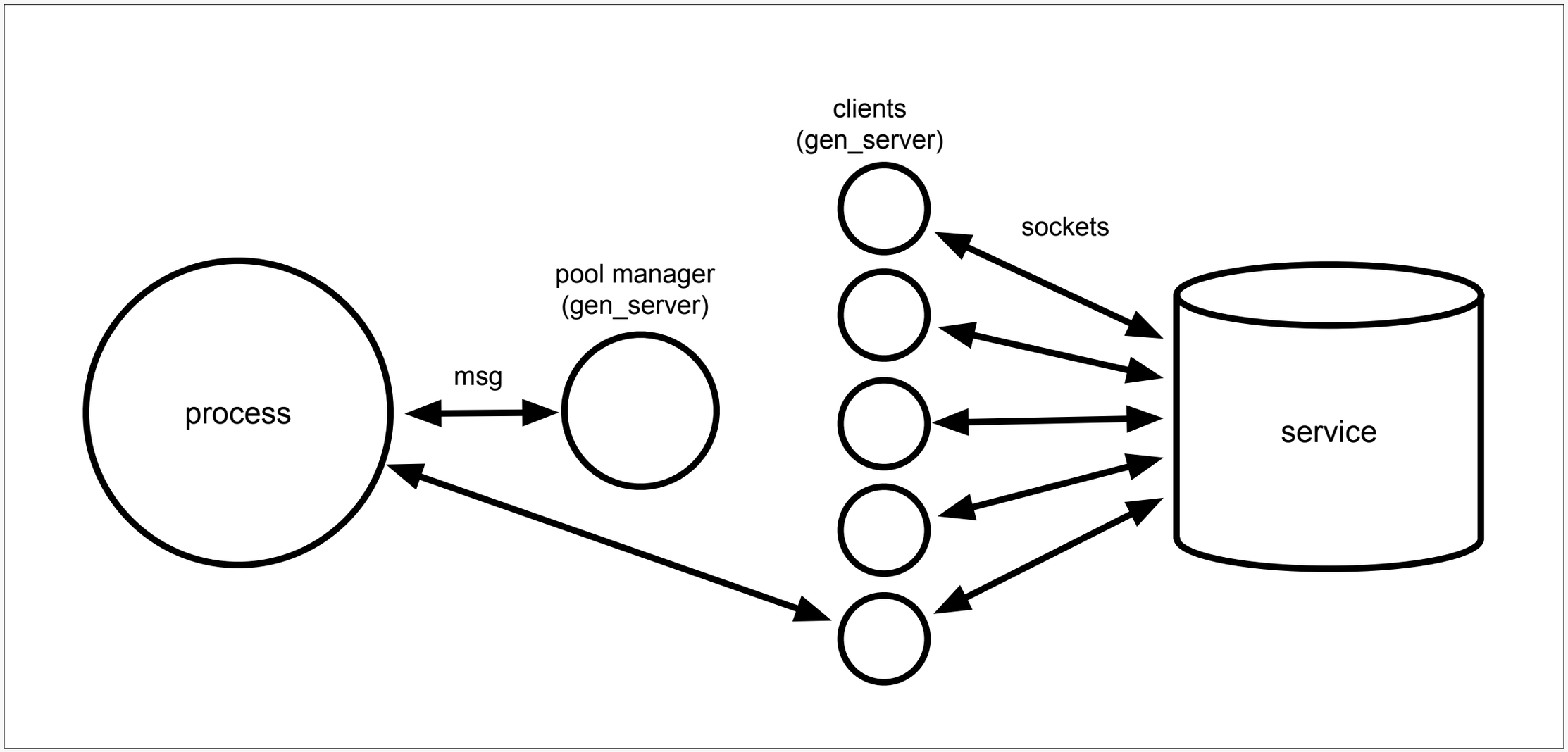
adgear

Design #2 problems

- how do I find my server? registered name?

adgear

Design #3



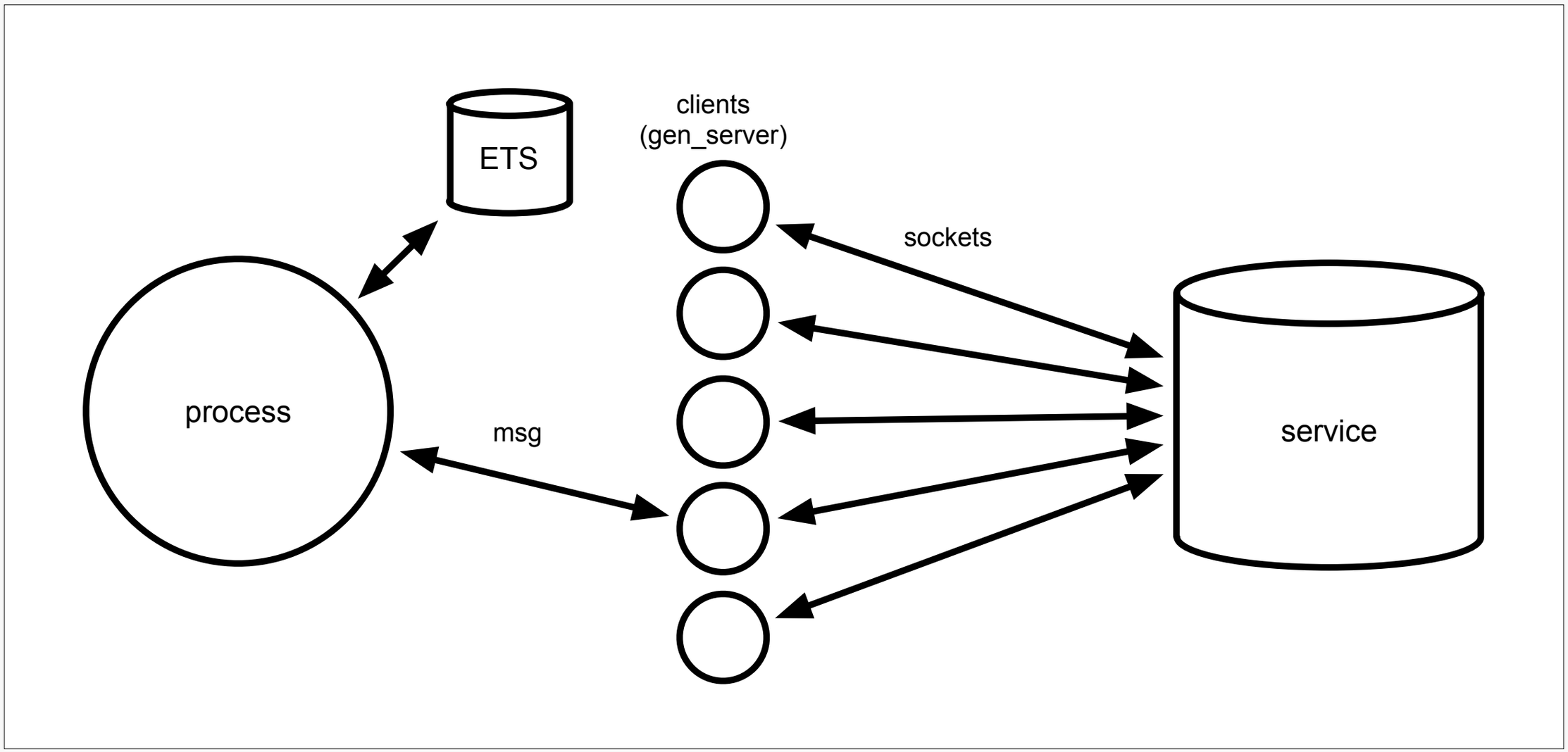
adgear

Design #3 problems

- single point of contention (pool manager)
- two messages per call (buffer bloat)

adgear

Design #4



adgear

Design #4 problems

- concurrency?
  - no pipelining

adgear

Design #4 problems

```
handle_call(Request, From, State) ->

    %% send request
    %% receive / decode response

    {reply, Response, State}.
```

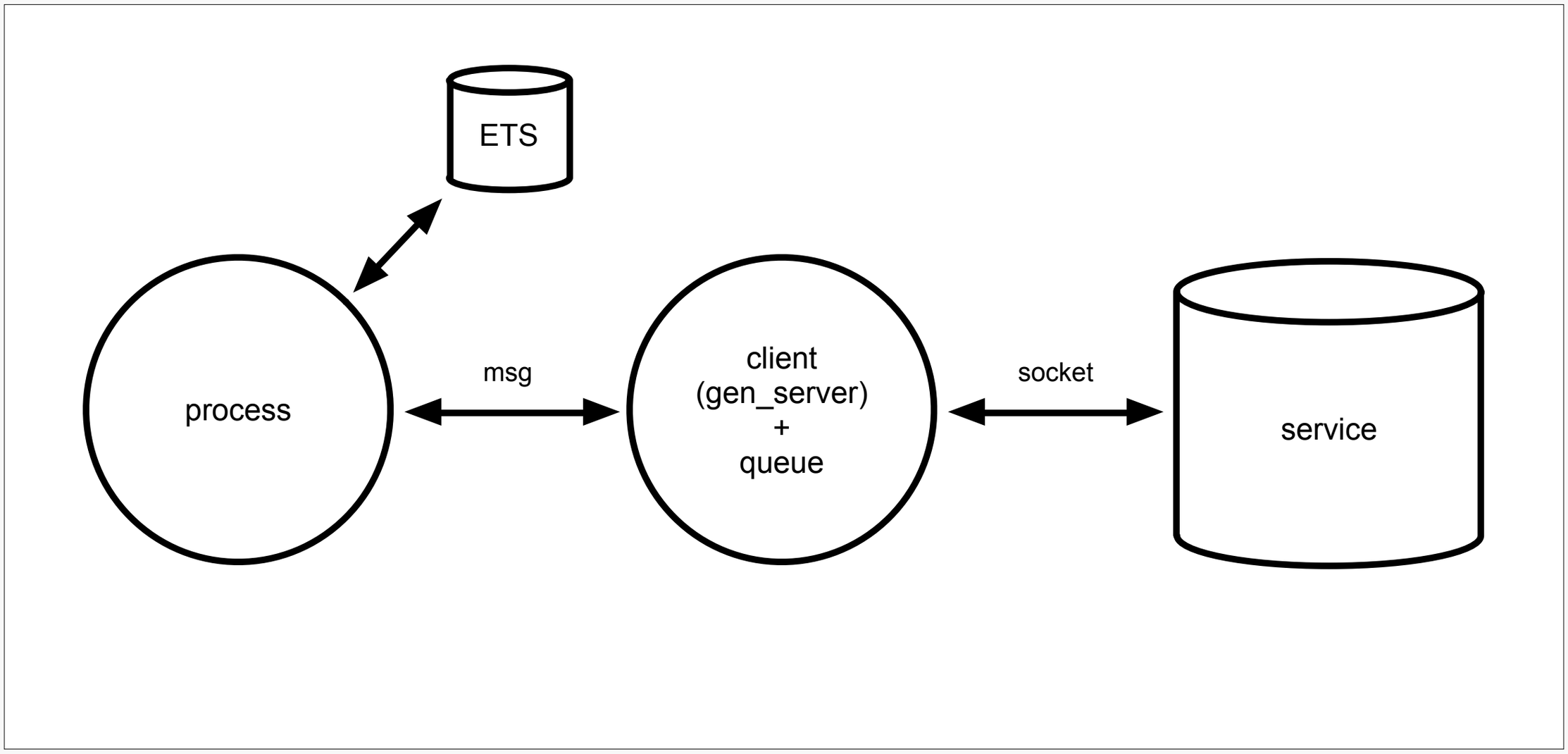

adgear

Design #5

```
handle_call(Request, From, #state {queue = Queue} = State) ->

    %% send request

    {noreply, State#state {queue = queue:in(From, Queue)}}.

handle_info({tcp, Socket, Data}, #state {queue = Queue} = State) ->

    %% decode response

    {{value, From}, Queue2} = queue:out(Queue),
    gen_server:reply(From, Response),
    {noreply, State#state {queue = Queue2}}.
```


adgear

Design #5



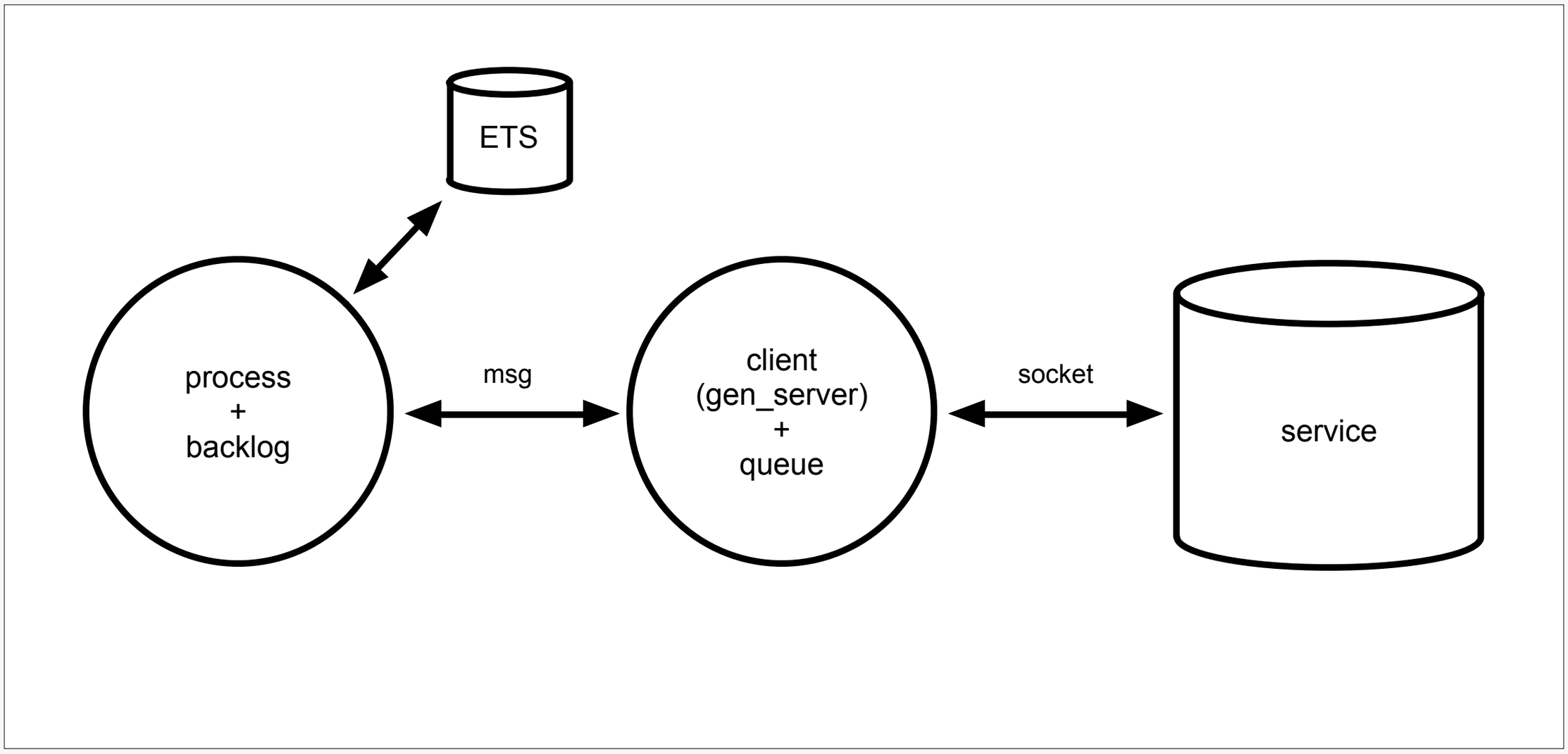
adgear

Design #5 problems

- safety?
  - no back pressure

adgear

Design #6



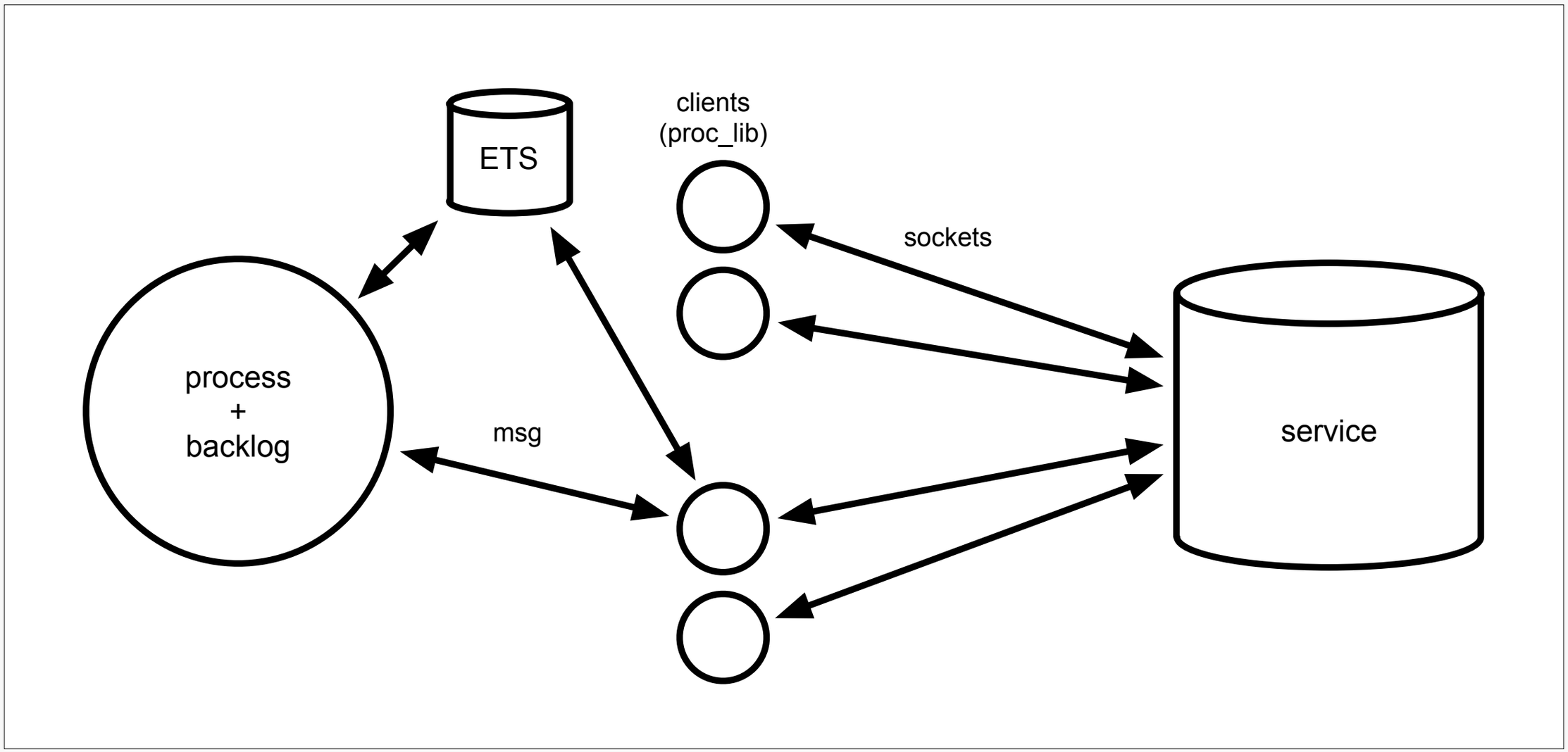
adgear

Design #6 problems

- speed?
  - gen_server is not optimal
  - queue implementation is not optimal

adgear

Design #7



adgear

Shackle’s architecture

adgear

Core

- shackle_pool
- shackle_backlog
- shackle_server
- shackle_queue

adgear

shackle_pool

- supports random or round robin strategy
- leverages ETS (no manager)
- no ETS contention since it only does reads
- very low overhead (almost constant time)

adgear

shackle_backlog

- protects from out of memory (OOM)
- leverages ETS (no manager)
- one backlog per server
- ETS update counter (fast and atomic)
- write_concurrency true

adgear

shackle_backlog

```
-spec check(server_name(), backlog_size()) -> boolean().

check(ServerName, BacklogSize) ->
    case increment(ServerName, BacklogSize) of
        [BacklogSize, BacklogSize] -> false;
        [_, Value] when Value =< BacklogSize -> true
    end.

increment(ServerName, BacklogSize) ->
    UpdateOps = [{2, 0}, {2, 1, BacklogSize, BacklogSize}],
    ets:update_counter(?ETS_TABLE_BACKLOG, ServerName, UpdateOps).
```


adgear

shackle_server

- uses proc_lib
- only asynchronous calls
- binary matching
- iolists
- skinny

adgear

shackle_queue

- leverages ETS
- faster than stdlib queue module
- items can be out of order
- no contention since items are keyed by server name (one producer / consumer)

adgear

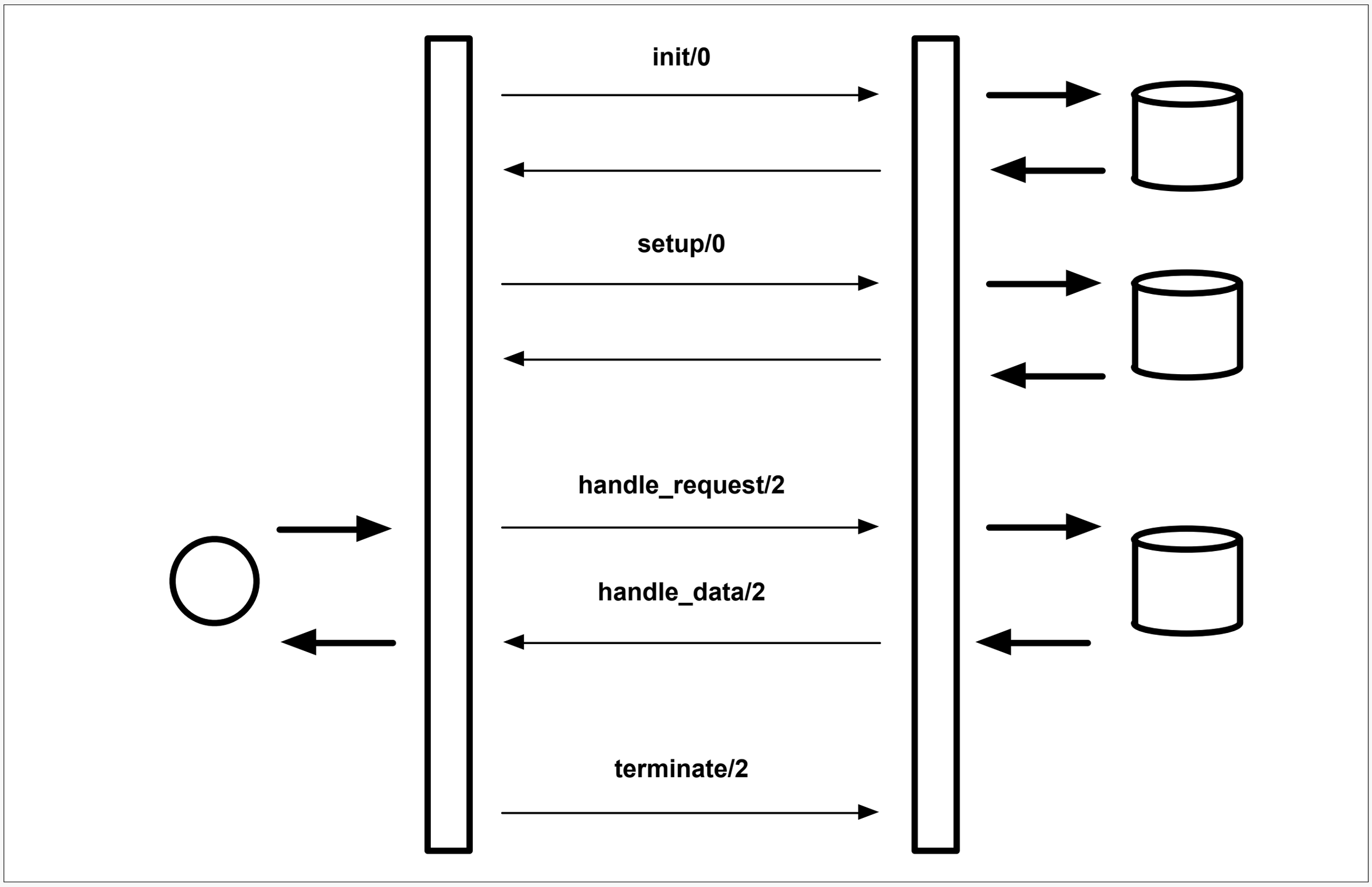
Building a client with Shackle

adgear

shackle_client behaviour

- init/0
- setup/2
- handle_request/2
- handle_data/2
- terminate/1

adgear



adgear

Arithmetic service

- TCP over port 8080
- session setup
- supports two operations:
  - addition
  - multiplication

Setup

request:

“INIT”

response:

“OK”

adgear

Operations

request:

request id (tinyint), operator (tinyint), operand (tinyint), operand (tinyint), operand (tinyint)

response:

request id (tinyint), result (smallint)

adgear

init/0

```
-record(state, {
    buffer = <<>>,
    request_counter = 0
}).


-spec init() ->
    {ok, State :: term()}.

init() ->
    {ok, #state {}}.
```


adgear

setup/2

```
-spec setup(Socket :: inet:socket(), State :: term()) ->
    {ok, State :: term()} |
    {error, Reason :: term(), State :: term()}.

setup(Socket, State) ->
    case gen_tcp:send(Socket, <<"INIT">>) of
        ok -> case gen_tcp:recv(Socket, 0) of
                {ok, <<"OK">>} -> {ok, State};
                {error, Reason} -> {error, Reason, State}
              end;
        {error, Reason} -> {error, Reason, State}
    end.
```


adgear

handle_request/2

```
-spec handle_request(Request :: term(), State :: term()) ->
    {ok, RequestId :: external_request_id(), Data :: iodata(),
        State :: term()}.

handle_request({Operation, A, B}, #state {
        request_counter = RequestCounter
    } = State) ->

    RequestId = request_id(RequestCounter),
    Data = request(RequestId, Operation, A, B),

    {ok, RequestId, Data, State#state {
        request_counter = RequestCounter + 1
    }}.
```


adgear

Protocol encoding

```
-define(MAX_REQUEST_ID, 256).

-spec request_id(non_neg_integer()) -> tiny_int().

request_id(RequestCounter) ->
    RequestCounter rem ?MAX_REQUEST_ID.
```

Protocol encoding

```
-spec opcode(operation()) -> 1..2.

opcode(add) -> 1;
opcode(multiply) -> 2.

-spec request(tiny_int(), operation(), tiny_int(), tiny_int()) ->
    binary().

request(RequestId, Operation, A, B) ->
    <<RequestId, (opcode(Operation)), A, B>>.
```


adgear

handle_data/2

```
-spec handle_data(Data :: binary(), State :: term()) ->
    {ok, [Response :: response()], State :: term()}.

handle_data(Data, #state {
        buffer = Buffer
    } = State) ->

Data2 = <<Buffer/binary, Data/binary>>,
{Replies, Buffer2} = parse_replies(Data2, []),

{ok, Replies, State#state {
    buffer = Buffer2
}}.
```


adgear

Protocol decoding

```
-spec parse_replies(binary()) ->
    {[response()], binary()}.

parse_replies(<<ReqId:8/integer, A:16/integer, R/binary>>, Acc) ->
    parse_replies(R, [{ReqId, A} | Acc]);
parse_replies(Buffer, Acc) ->
    {Acc, Buffer}.
```


adgear

terminate/1

```
-spec terminate(State :: term()) -> ok.

terminate(_State) ->
    shackle_utils:info_msg(?MODULE, "terminating~n", []),
    ok.
```


adgear

Starting client pool

```
-spec start() ->
    ok | {error, shackle_not_started | pool_already_started}.

start() ->
    shackle_pool:start(?POOL_NAME, ?CLIENT_TCP, [
        {port, ?PORT},
        {reconnect, true},
        {socket_options, [
            binary,
            {packet, raw}
        ]}
    ], [
        {pool_strategy, random}
    ]).
```


adgear

Starting client pool

```
-type client_option() ::
    {ip, inet:ip_address() | inet:hostname()} |
    {port, inet:port_number()} |
    {protocol, protocol()} |
    {reconnect, boolean()} |
    {reconnect_time_max, time()} |
    {reconnect_time_min, time()} |
    {socket_options, [gen_tcp:connect_option() | gen_udp:option()]}.

-type protocol() :: shackle_tcp | shackle_udp.
-type time() :: pos_integer().
```

Starting client pool

```
-type pool_option() ::
    {backlog_size, backlog_size()} |
    {pool_size, pool_size()} |
    {pool_strategy, pool_strategy()}.

-type backlog_size() :: pos_integer() | infinity.
-type pool_size() :: pos_integer().
-type pool_strategy() :: random | round_robin.
```

Starting client pool

```
1> arithmetic_client:start().
ok

T 127.0.0.1:54269 -> 127.0.0.1:8080 [AP]
  49 4e 49 54                                          INIT

T 127.0.0.1:8080 -> 127.0.0.1:54269 [AP]
  4f 4b                                                OK
```


adgear

Calling clients

```erlang
-spec add(tiny_int(), tiny_int()) -> pos_integer().

add(A, B) ->
    shackle:call(?POOL_NAME, {add, A, B}).

-spec multiply(tiny_int(), tiny_int()) -> pos_integer().

multiply(A, B) ->
    shackle:call(?POOL_NAME, {multiply, A, B}).
```


adgear

Calling clients

```
2> arithmetic_client:add(5,10).
15

T 127.0.0.1:54269 -> 127.0.0.1:8080 [AP]
  00 01 05 0a                                                ....

T 127.0.0.1:8080 -> 127.0.0.1:54269 [AP]
  00 00 0f                                                   ...
```


adgear

Calling clients

```
3> arithmetic_client:multiply(2,6).
12

T 127.0.0.1:54269 -> 127.0.0.1:8080 [AP]
  01 02 02 06                                          ....

T 127.0.0.1:8080 -> 127.0.0.1:54269 [AP]
  01 00 0c                                             ...
```


adgear

Casting clients

```
-spec add_async(tiny_int(), tiny_int()) -> {ok, request_id()}.

add_async(A, B) ->
    shackle:cast(?POOL_NAME, {add, A, B}).

-spec add_receive(request_id()) -> pos_integer().

add_receive(RequestId) ->
    shackle:receive_response(RequestId).
```


adgear

Tips & tricks

Performance tips

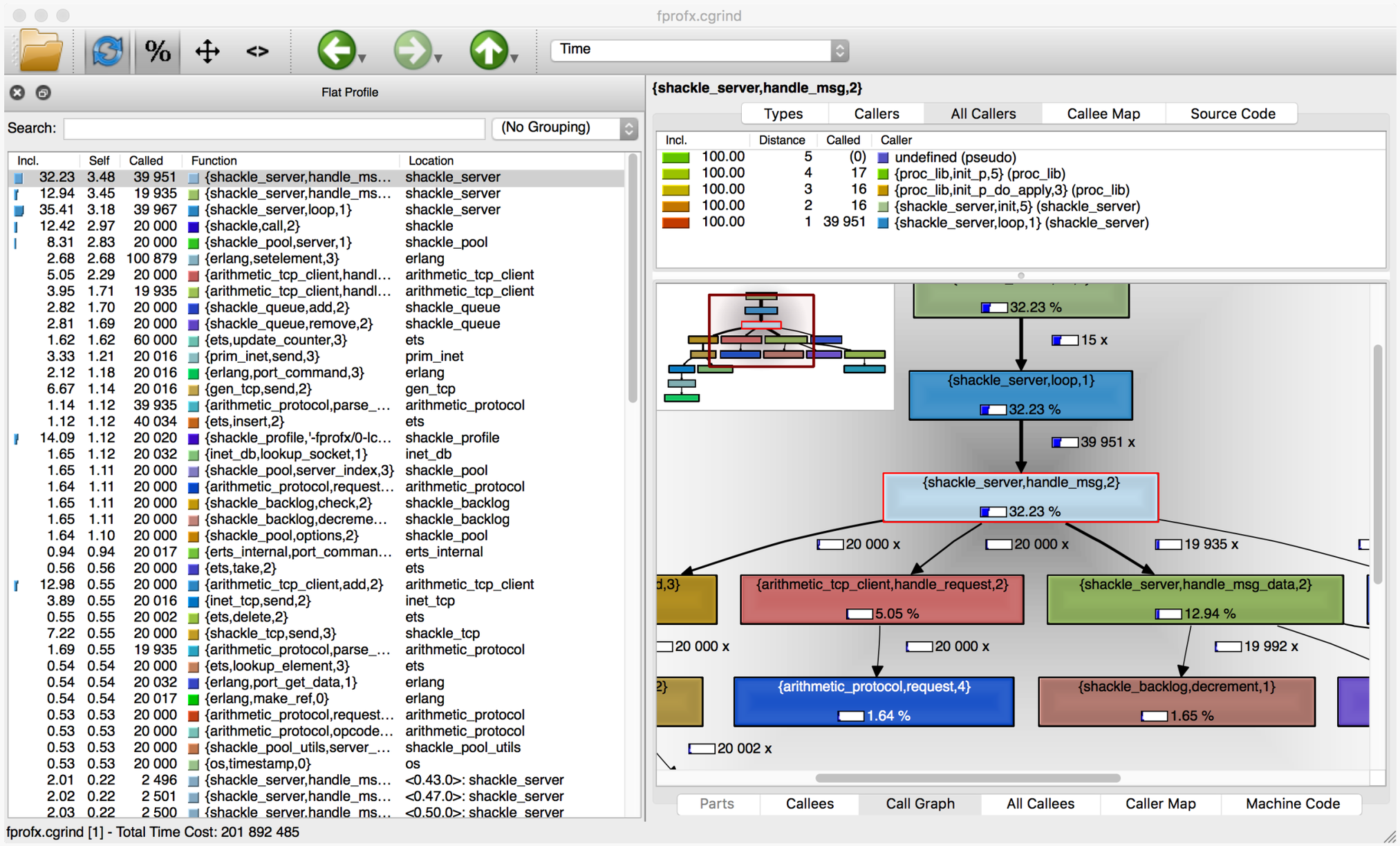
- pattern match everything
- use iolists when you can
- keep your client lean
- use ETS for shared state
- measure, measure, measure

Profiling

- make profile
- fprofx
- cachegrind

adgear

Profiling



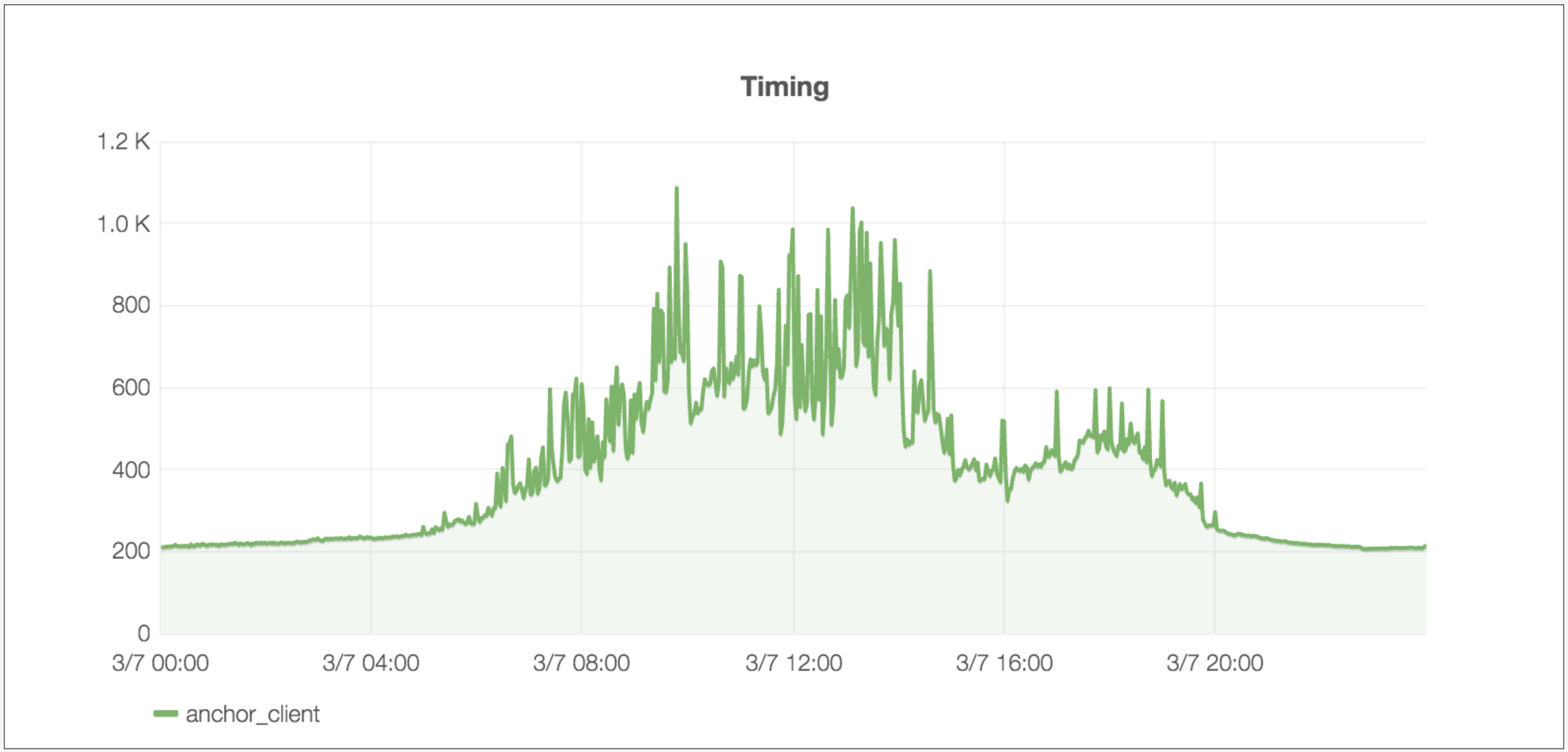
adgear

Performance

- removed handle_timing/2
- made round_robin default pool_strategy
- shackle_compiler

adgear

Performance



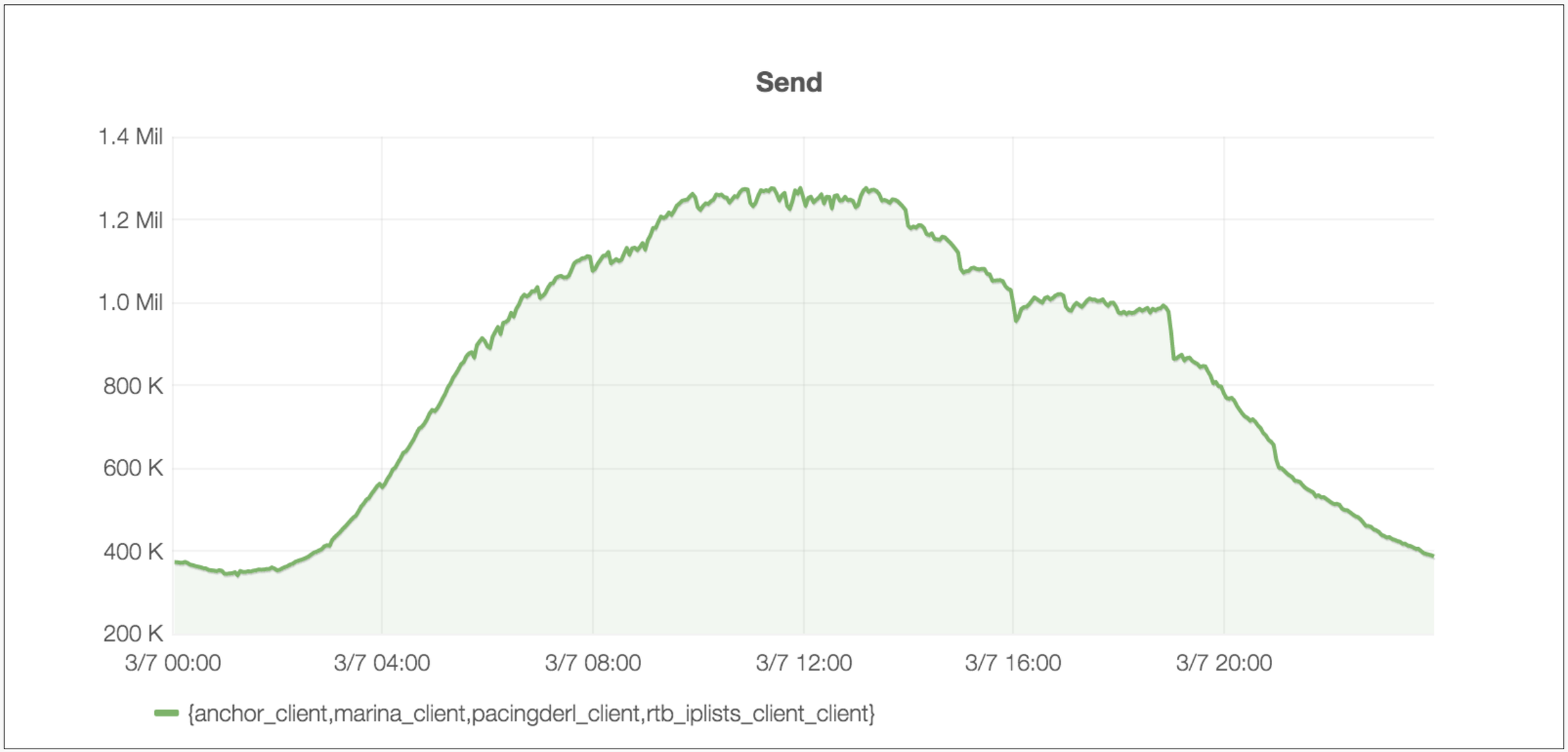
adgear

Usage

- anchor (memached binary client)
- buoy* (http 1.1 client)
- flare* (kafka producer client)
- marina (cassandra 2.1+ client)

adgear

Usage



adgear

Future

- cluster support (multi host)
- improve test coverage
- property-based tests

adgear

Links

- http://github.com/lpgauth/anchor
- http://github.com/lpgauth/buoy
- http://github.com/lpgauth/flare
- http://github.com/lpgauth/marina
- http://github.com/lpgauth/shackle

adgear

Thank you!

@lpgauth

adgear