REdis: Implementing Redis in Erlang

A step-by-step walkthrough

Edis: Implementing Redis in Erlang

A step-by-step walkthrough

My Background

- Microsoft Visual Studio
- Visto Corporation
- Founded Inaka
- Moved to Argentina 2008

@chaddepue / cdepue

Inaka Overview

- Started in 2009
- iPhone/Android Apps
- Erlang Systems
- Ruby/Rails

Our Vision

- Remote Startup Incubator
- Focus on New Media
- Social Media
- Big Data

Our Apps

- MTV WatchWith
- MovieNightOut
- Whisper
- Campus Sentinel

My Goals For This Talk

- See a server app you know in Erlang
- Walk through gen_tcp/gen_fsm
- See the Redis API implementation
- See how to extend Edis

What are Redis and Edis?

Redis - a C-based fast in-memory, disk-backed key/value database

Edis - an Erlang-based, leveldb-backed key/value store that speaks the Redis protocol

What specifically is Edis?

Edis – an Erlang-based server that...

- Uses gen_tcp
- Uses gen_fsm
- Uses LevelDB
- Implements full Redis command set
- Respects Redis algorithms

What makes Redis worth copying?

3 things...

- Speed
- Expressivity of the command set
- Ease of Deployment

Speed

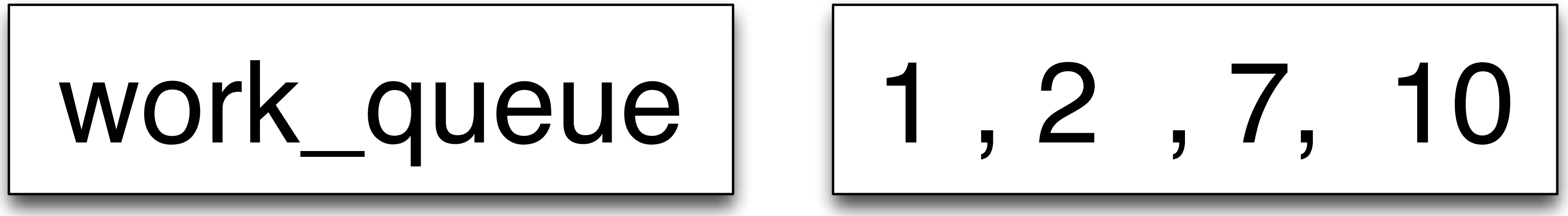
> *"Raw speed is bound to queries per watt. Energy is a serious problem, not just for the environment, but also cost-wise. More computers running to serve your 1000 requests per second, the bigger your monthly bill."*
>
> Salvatore Sanfilippo

Expressive Command Set

Command Group	*Selected Commands*
Key/Value	SET/GET
Hashes	HSETNX
Lists	RPOP/LPOP
Sets	SUNION/SPOP
Sorted Sets	ZADD/ZRANGE
Publish/Subscribe	SUBSCRIBE/PUBLISH
Transactions	MULTI/EXEC

Expressive Command Set

RPOPLPUSH



work_queue

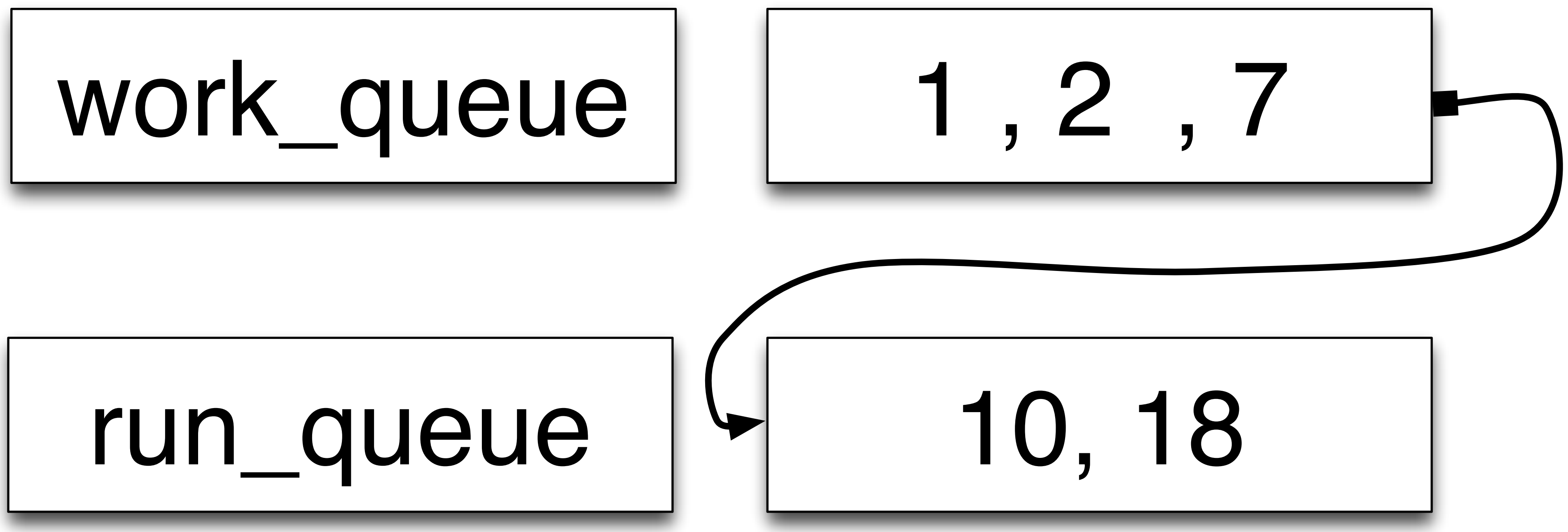
1 , 2 , 7, 10

Expressive Command Set

RPOPLPUSH

work_queue	1 , 2 , 7, 10
run_queue	18

Expressive Command Set



Easy Deployment

```
$ redis-server

$ redis-cli
```

Traded for ...

- "Weak" Persistence
  - not a great disk-backed DB
  - data must fit in-memory

* great article by Salvatore on Monday about AOF and RDB options

Traded for ...

- Lack of Expandability*

*lua scripting is here in 2.6...
but it's blocking the main thread!?

Comparisons

Redis	**Edis**
Incredibly Fast	Slow
Amazing Command Set	Almost The Same Set
Guarantees in ˜ 1s	*Guaranteed Persistance*
Limited Extensibility	*Extensible*
Data can't exceed RAM	*Data can exceed RAM*

Three Topics

Implementing the Redis Protocol

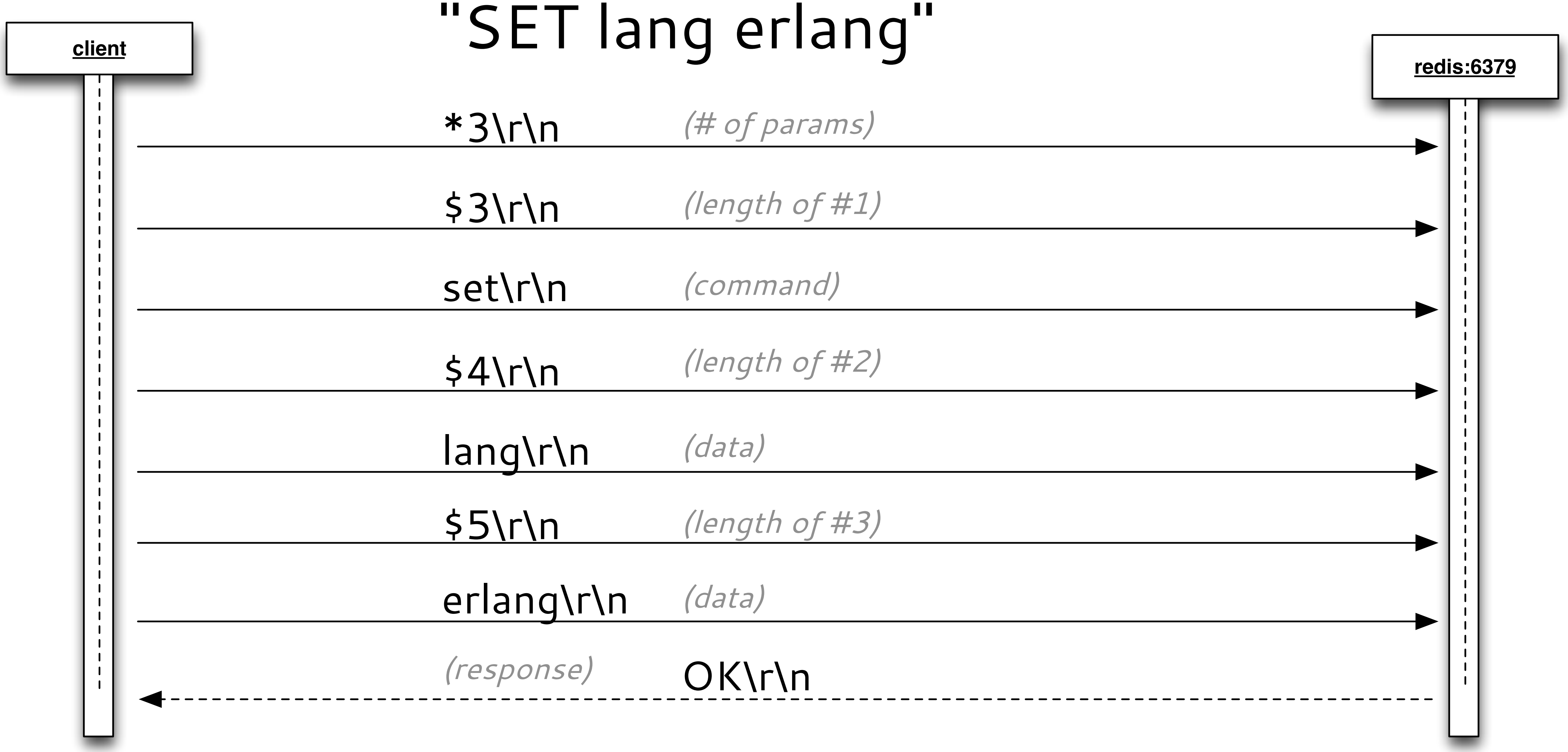
Measuring Edis Performance

Extending Edis

Protocol

Redis protocol

"SET lang erlang"



*3\r\n (# of params)

$3\r\n (length of #1)

set\r\n (command)

$4\r\n (length of #2)

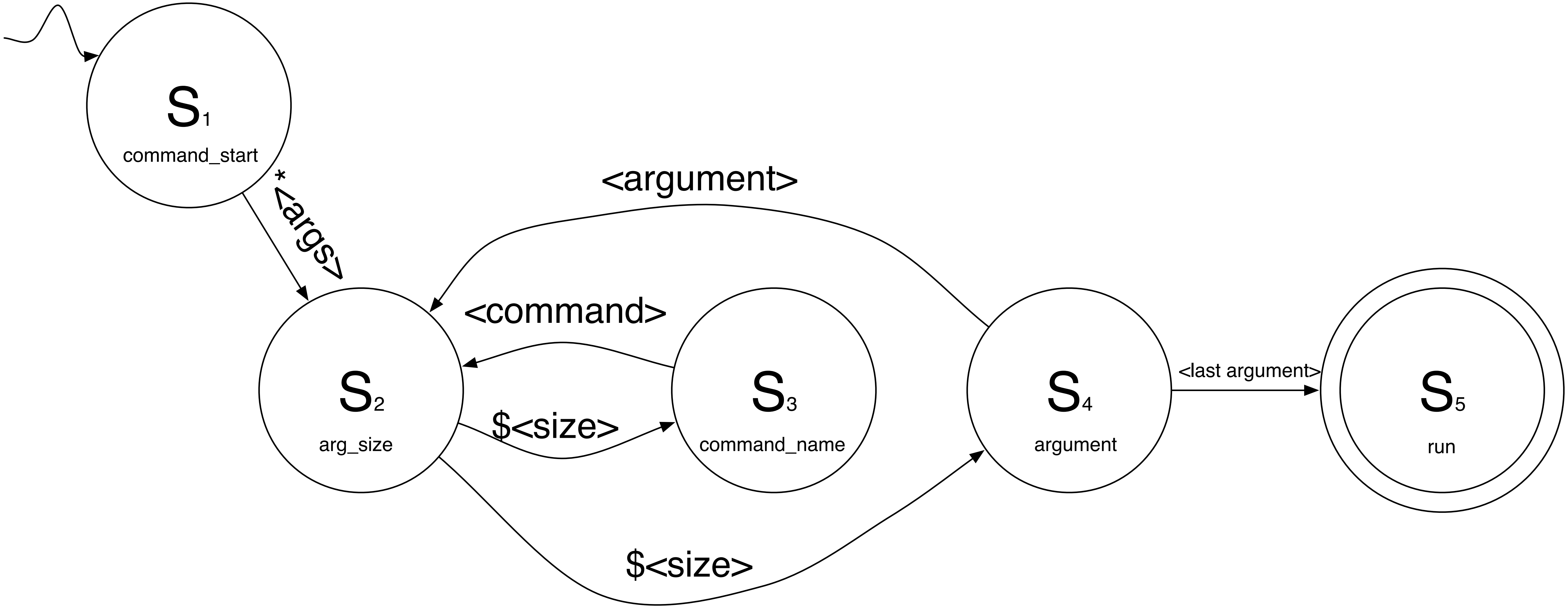
lang\r\n (data)

$5\r\n (length of #3)

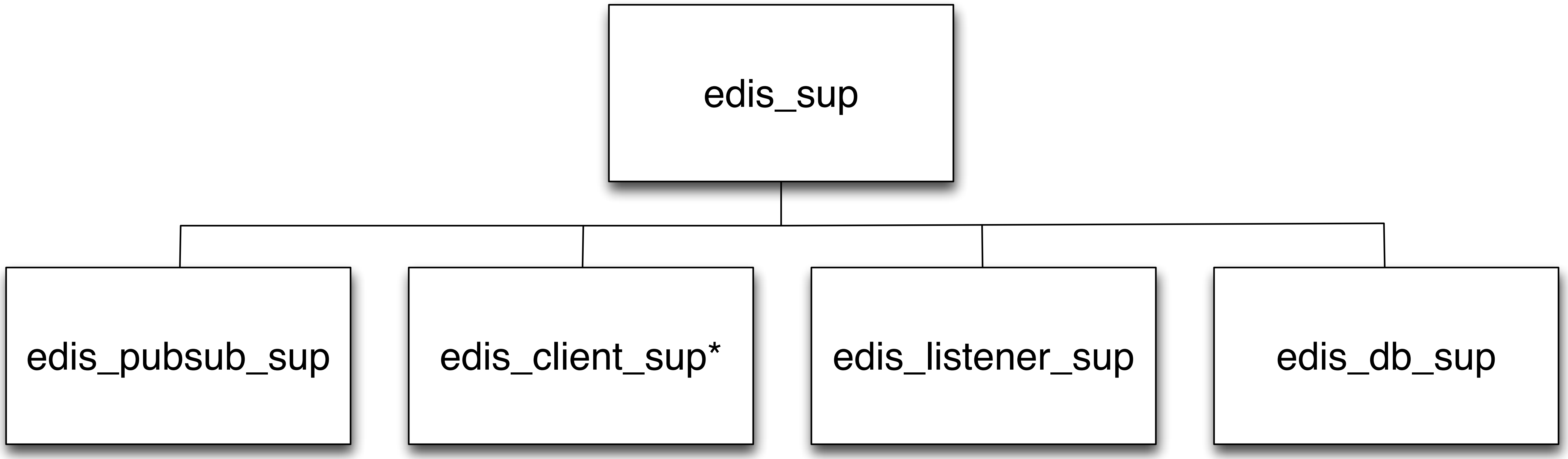
erlang\r\n (data)

(response) OK\r\n

State Machine - Client

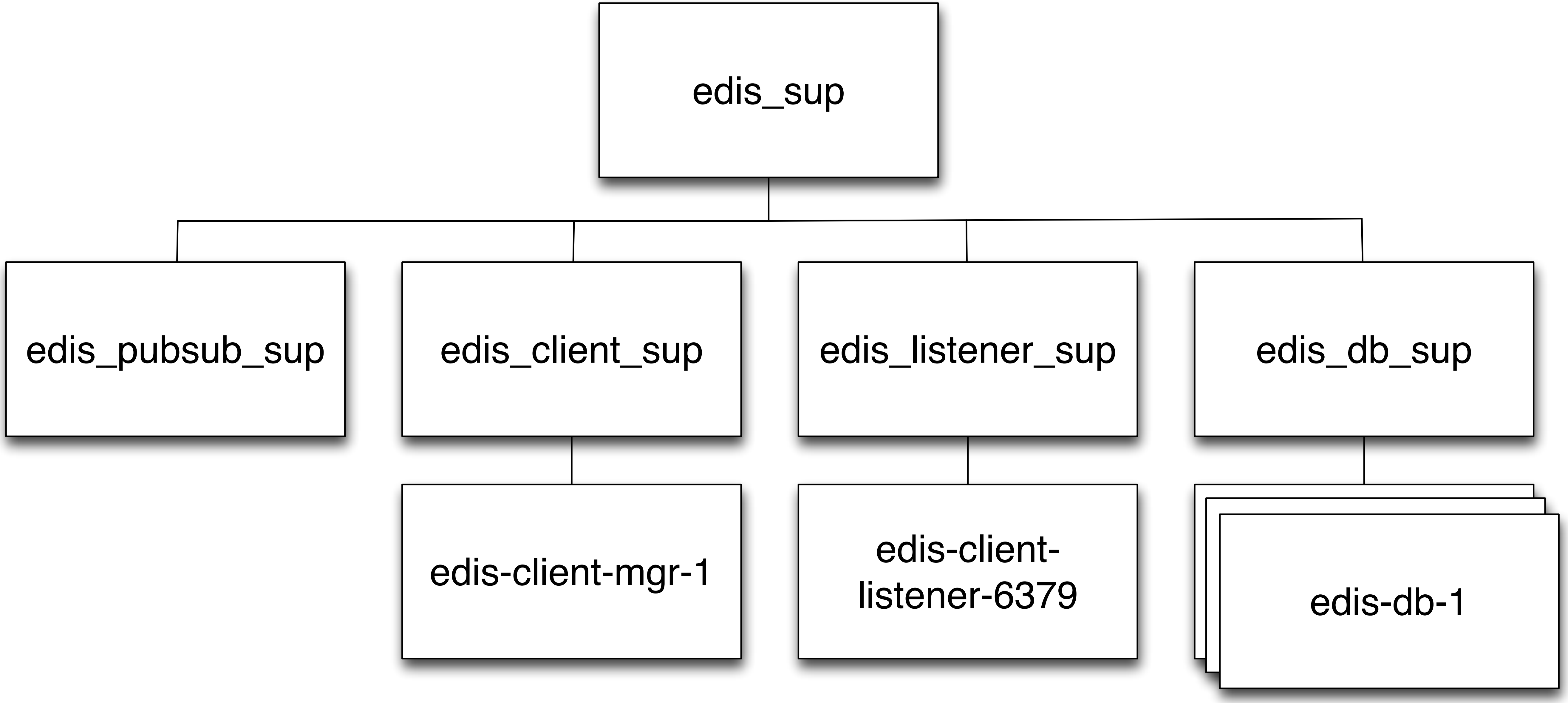


Application Structure

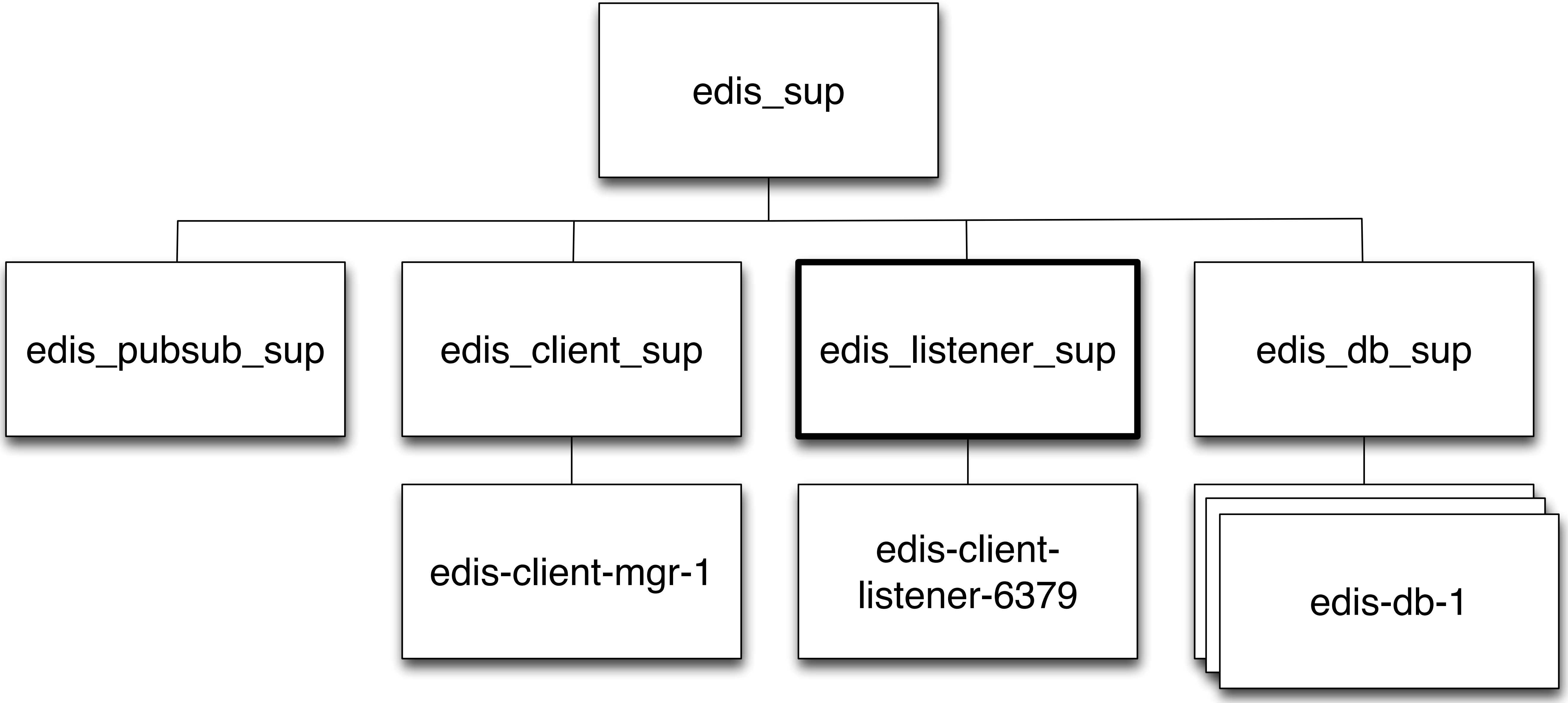


* simple_one_for_one

Ready for connections



Ready for connections



edis_listener_sup.erl

```
init([]) ->
  {MinPort, MaxPort} = edis_config:get(listener_port_range),
  Listeners =
    [{list_to_atom("edis-listener-" ++ integer_to_list(I)),
      {edis_listener, start_link, [I]}, permanent, brutal_kill,
       worker, [edis_listener]}
     || I <- lists:seq(MinPort, MaxPort)],
  {ok, {{one_for_one, 5, 10}, Listeners}}.
```

edis_listener_sup.erl

```
init([]) ->
  {MinPort, MaxPort} = edis_config:get(listener_port_range),
  Listeners =
    [{list_to_atom("edis-listener-" ++ integer_to_list(I)),
      {edis_listener, start_link, [I]}, permanent, brutal_kill,
       worker, [edis_listener]}
     || I <- lists:seq(MinPort, MaxPort)],
  {ok, {{one_for_one, 5, 10}, Listeners}}.
```

edis_listener.erl init(Port)

```erlang
init(Port) ->
  case gen_tcp:listen(Port, ?TCP_OPTIONS) of
    {ok, Socket} ->
      {ok, Ref} = prim_inet:async_accept(Socket, -1),
      {ok, #state{listener = Socket,
                  acceptor = Ref}};
    {error, Reason} ->
      {stop, Reason}
    end.

```

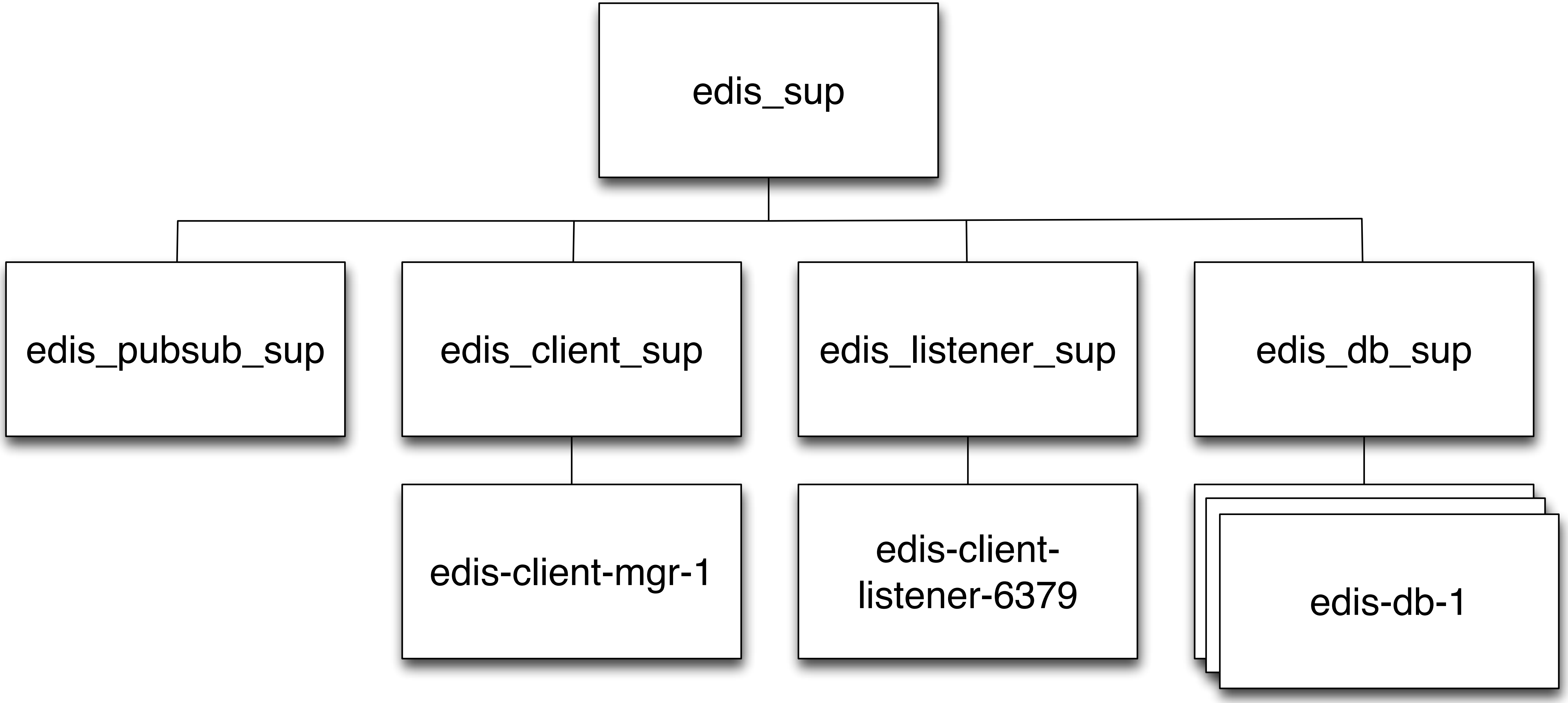
edis_listener.erl init(Port)

```erlang
init(Port) ->
  case gen_tcp:listen(Port, ?TCP_OPTIONS) of
    {ok, Socket} ->
      {ok, Ref} = prim_inet:async_accept(Socket, -1),
      {ok, #state{listener = Socket,
                  acceptor = Ref}};
    {error, Reason} ->
      {stop, Reason}
    end.

```

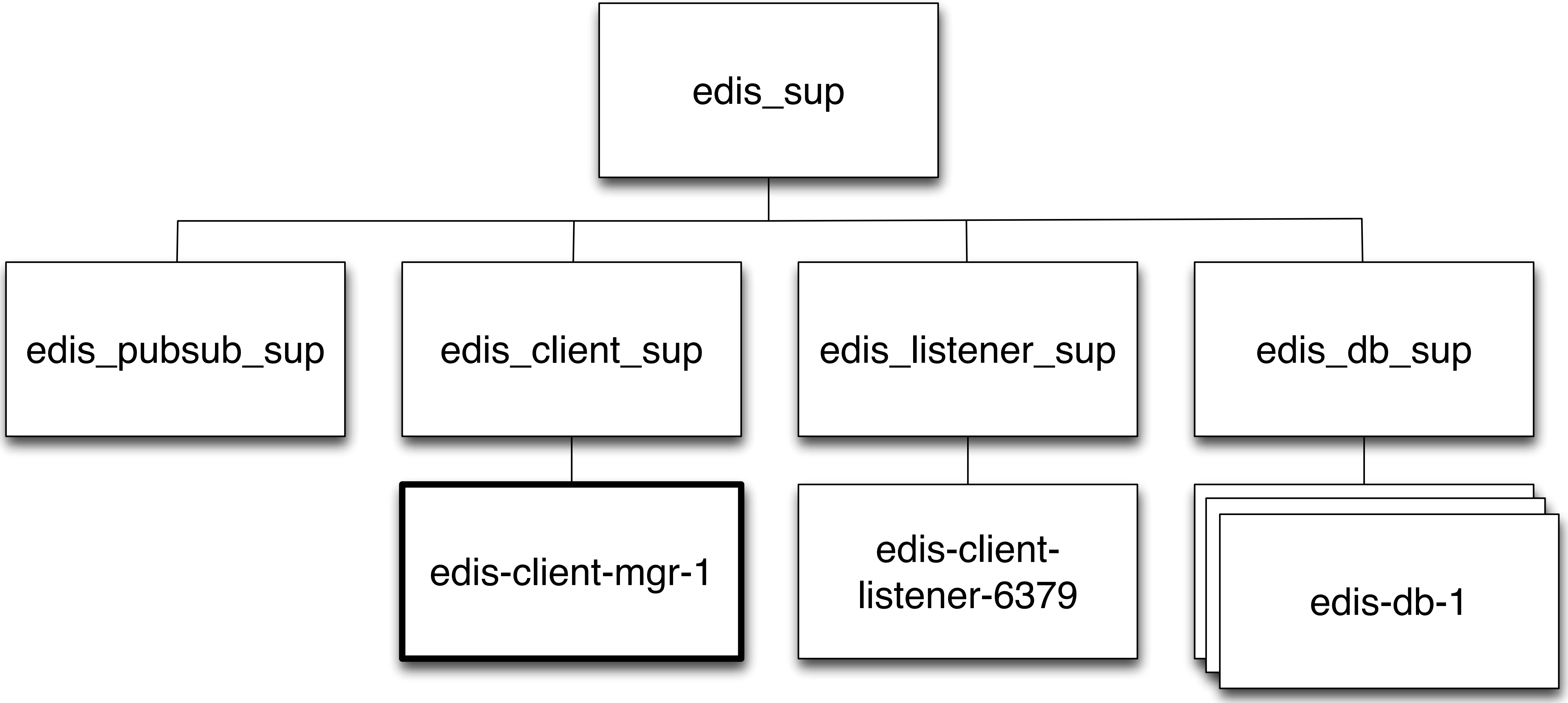
edis_listener.erl handle_info/2

```
handle_info({inet_async, ListSock, Ref, {ok, CliSocket}},
            #state{listener = ListSock, acceptor = Ref} = State) ->
..
    %% New client connected…
    ?DEBUG("Client ~p starting...~n", [PeerPort]),
    {ok, Pid} = edis_client_sup:start_client(),

    ok = gen_tcp:controlling_process(CliSocket, Pid),

    %% Instruct the new FSM that it owns the socket.
    ok = edis_client:set_socket(Pid, CliSocket),

    %% Tell the network driver we are ready for another connection
    NewRef = prim_inet:async_accept(ListSock, -1)
...
    {noreply, State#state{acceptor = NewRef}}
```

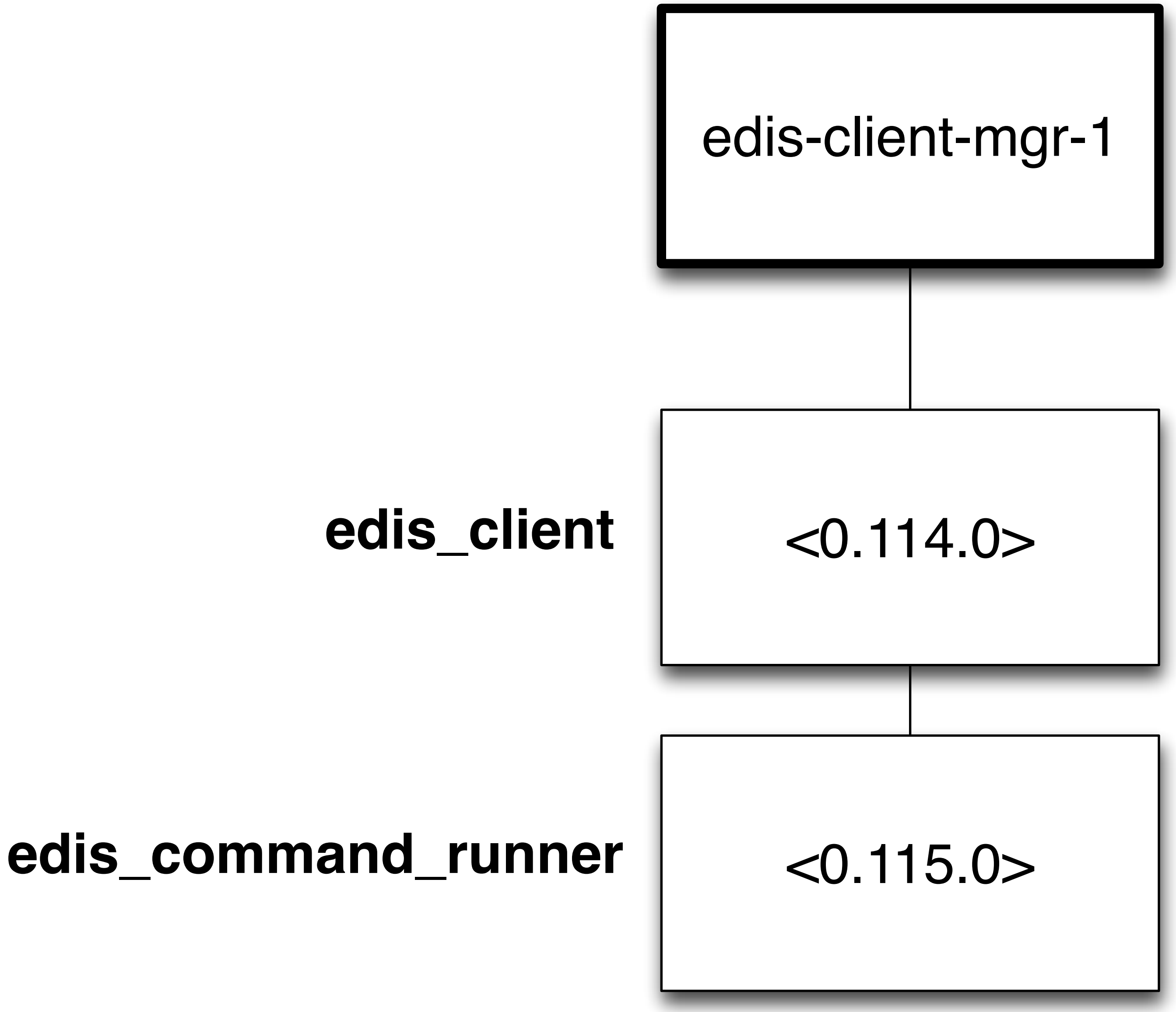
… Connection Established

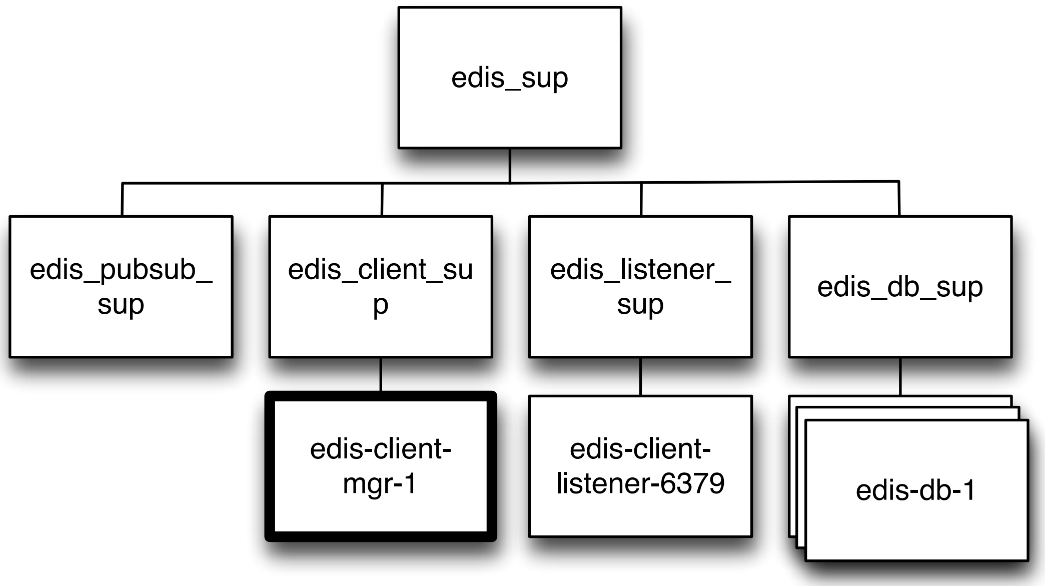


… Connection Established



… Connection Established





edis_client.erl

```
socket({socket_ready, Socket}, State) ->
  % Now we own the socket
  PeerPort = inet:peername(Socket),

  ok = inet:setopts(Socket, [{active, once},
                             {packet, line}, binary]),
  _ = erlang:process_flag(trap_exit, true),
  {ok, CmdRunner} = edis_command_runner:start_link(Socket),
  {next_state, command_start,
     State#state{socket         = Socket,
                 peerport       = PeerPort,
                 command_runner = CmdRunner}, hibernate};
```

edis_client.erl

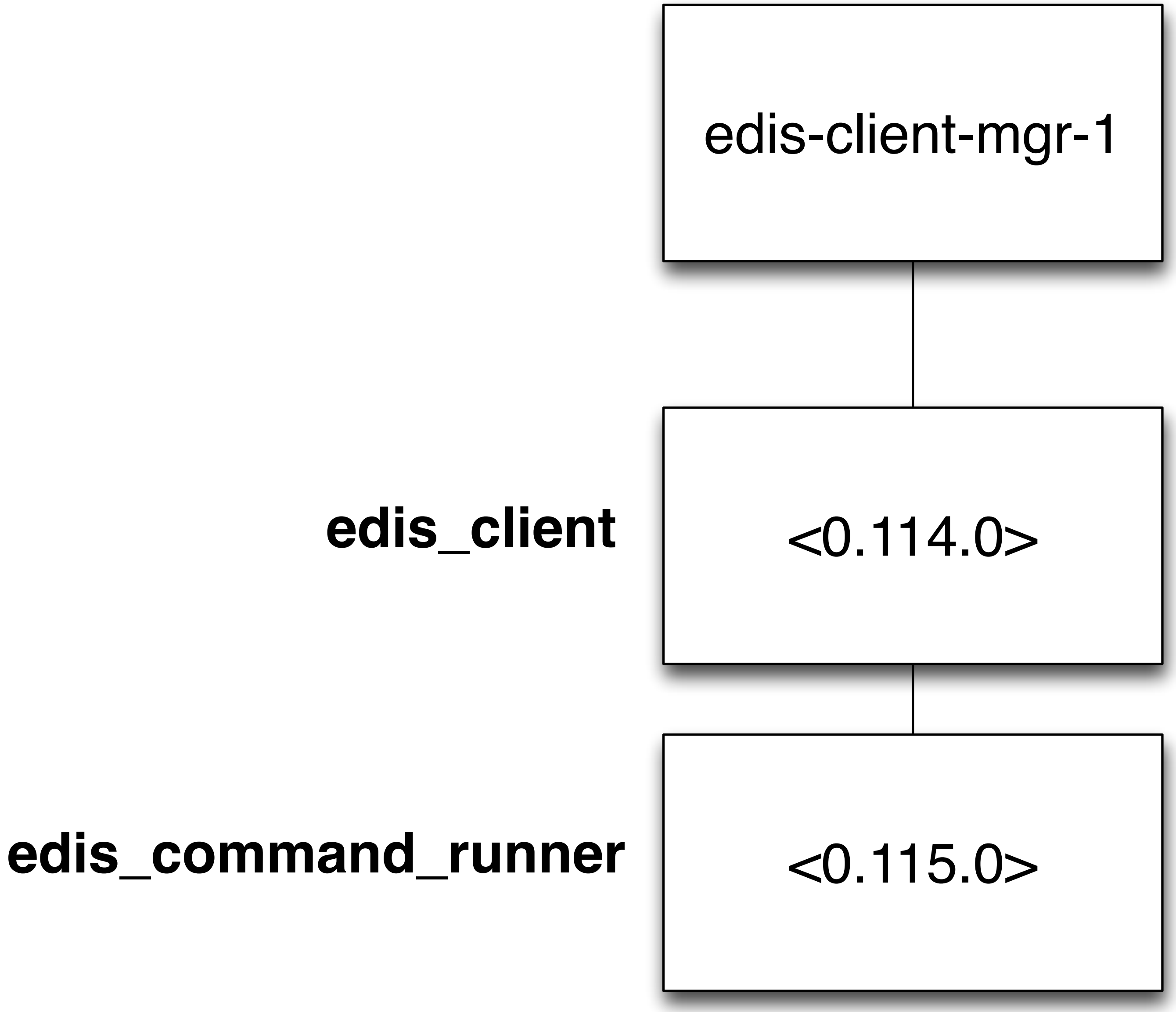
```erlang
socket({socket_ready, Socket}, State) ->
  % Now we own the socket
  PeerPort = inet:peername(Socket),

  ok = inet:setopts(Socket, [{active, once},
                             {packet, line}, binary]),
  _ = erlang:process_flag(trap_exit, true),
  {ok, CmdRunner} = edis_command_runner:start_link(Socket),
  {next_state, command_start,
     State#state{socket         = Socket,
                 peerport       = PeerPort,
                 command_runner = CmdRunner}, hibernate};
```

edis_client.erl

```
socket({socket_ready, Socket}, State) ->
```

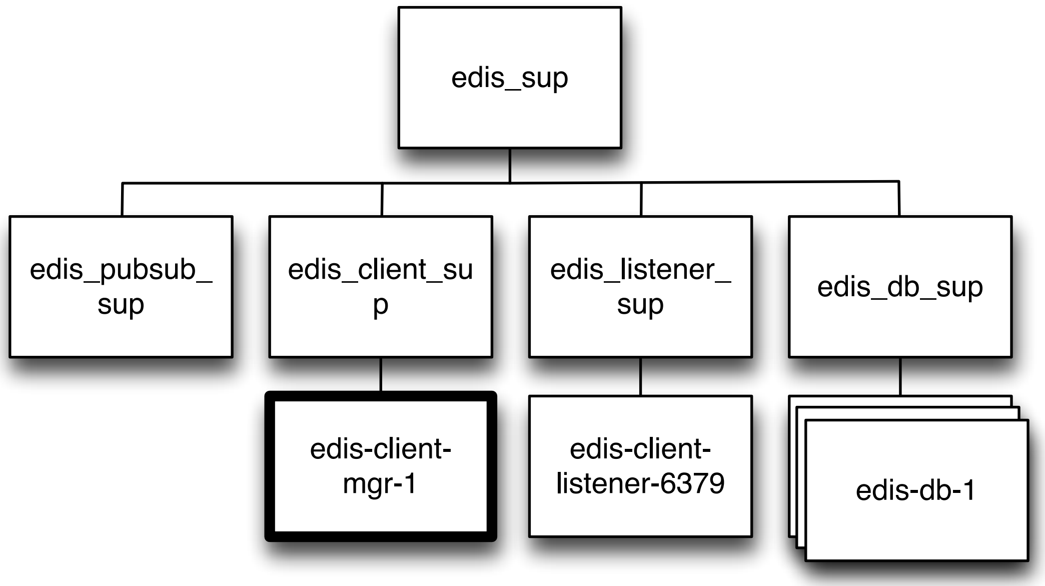
```
{active, once},
{packet, line}
```

… Connection Established

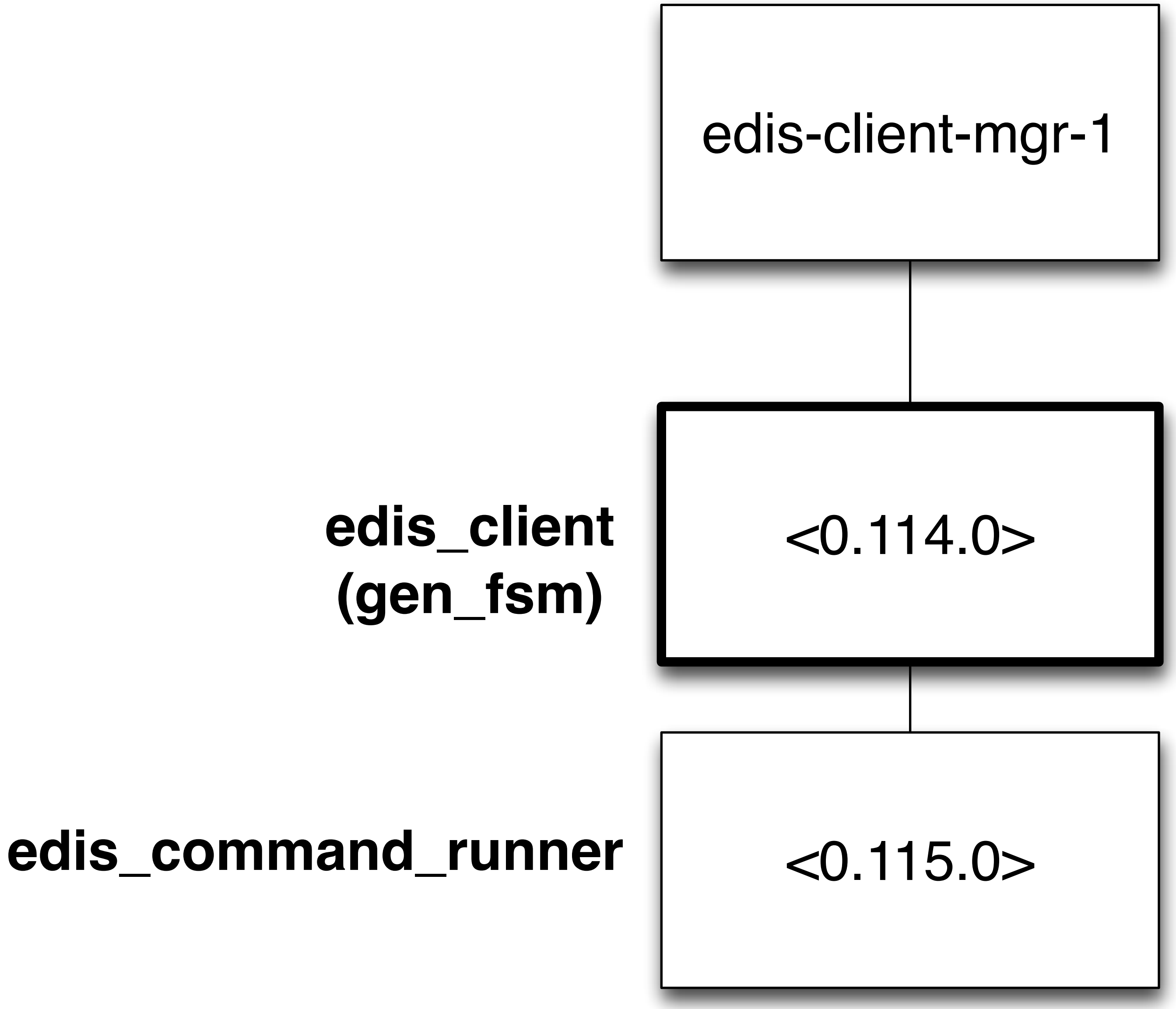

edis-client-mgr-1

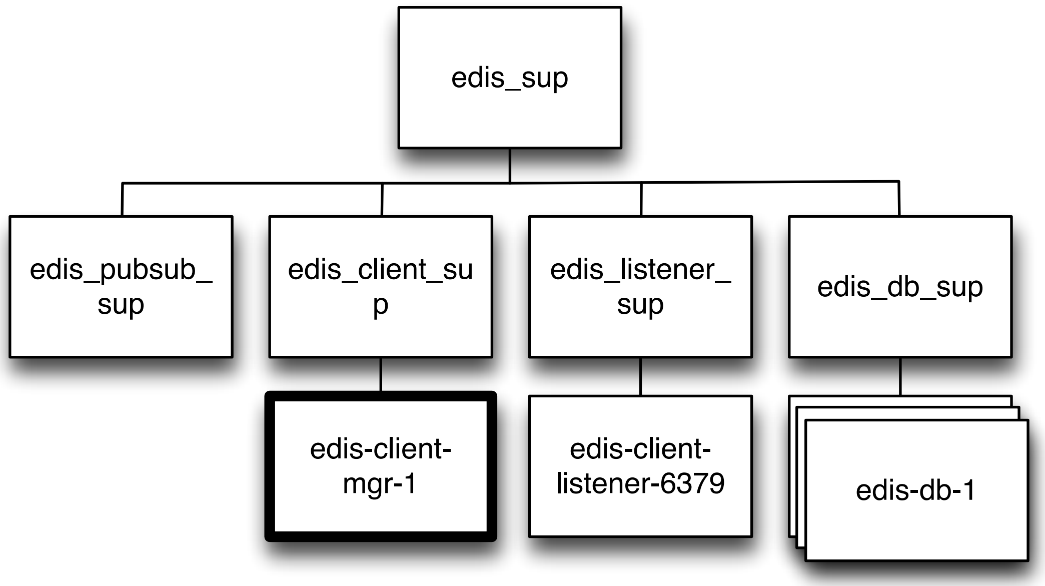
edis_client <0.114.0>

edis_command_runner <0.115.0>



… Connection Established





edis_client.erl

```erlang
socket({socket_ready, Socket}, State) ->
  % Now we own the socket
  PeerPort = inet:peername(Socket),

  ok = inet:setopts(Socket, [{active, once},
                             {packet, line}, binary]),
  _ = erlang:process_flag(trap_exit, true),
  {ok, CmdRunner} = edis_command_runner:start_link(Socket),
  {next_state, command_start,
     State#state{socket         = Socket,
                 peerport       = PeerPort,
                 command_runner = CmdRunner}, hibernate};
```

edis_client.erl

```
socket({socket_ready, Socket}, State) ->
```

```
{next_state, StateName, State};
```

```
  {next_state, command_start,
    State#state{socket         = Socket,
                peerport       = PeerPort,
                command_runner = CmdRunner}, hibernate};
```

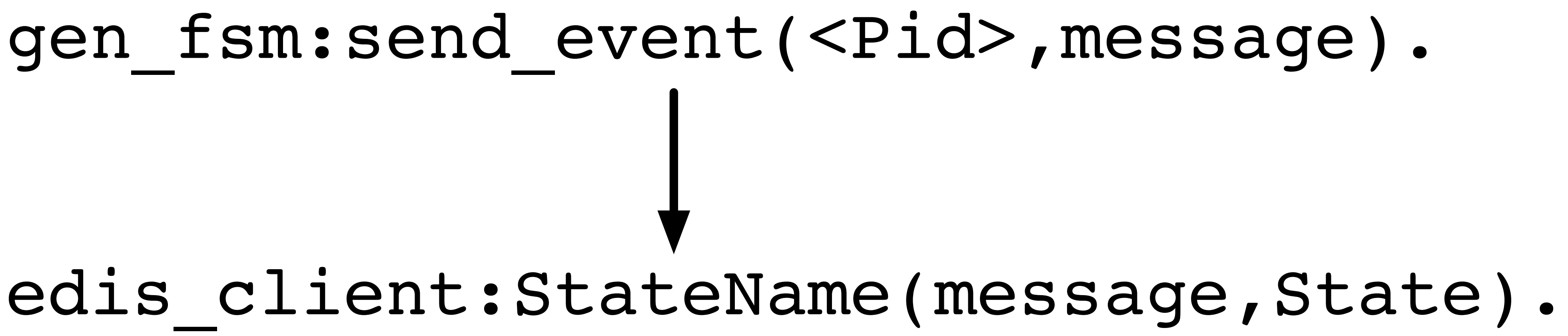
edis_client.erl

```
socket({socket_ready, Socket}, State) ->
```

```
    {next_state, StateName, State};
```

```
  {next_state, command_start,
     State#state{socket         = Socket,
                 peerport       = PeerPort,
                 command_runner = CmdRunner}, hibernate};
```

edis_client.erl

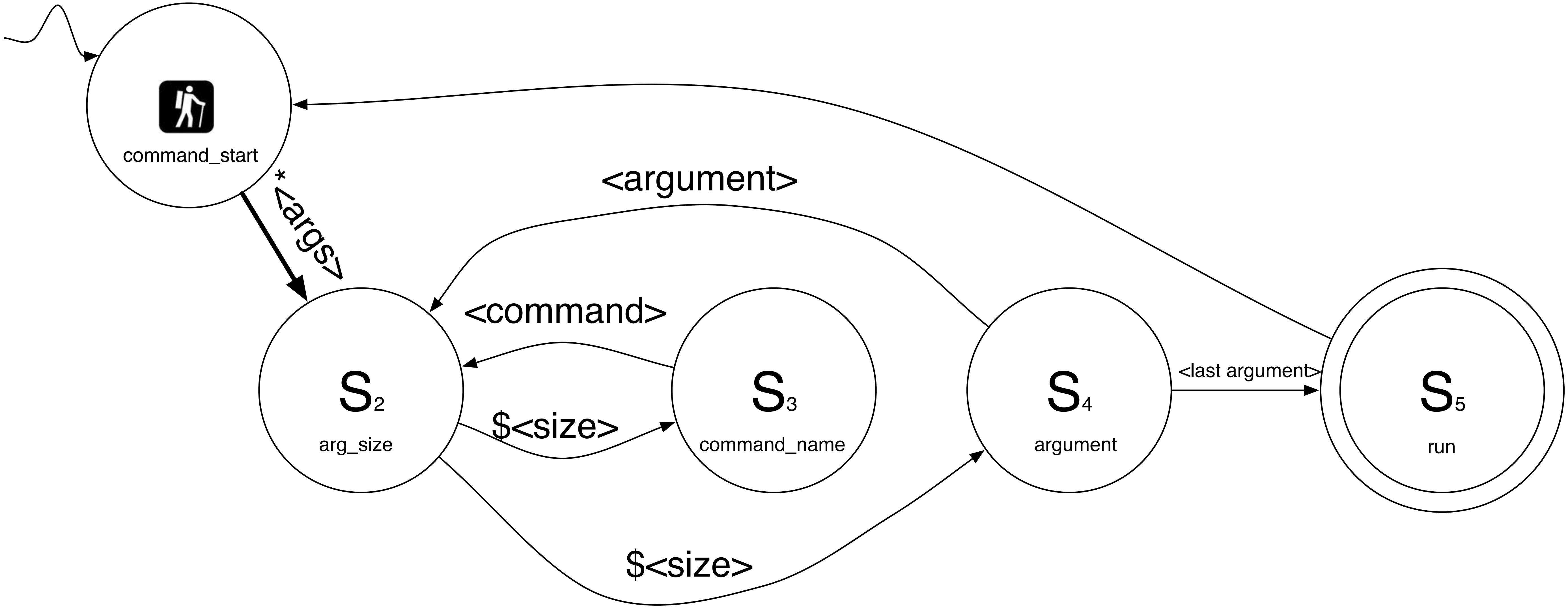


```
gen_fsm:send_event(<Pid>,message).

edis_client:StateName(message,State).
```

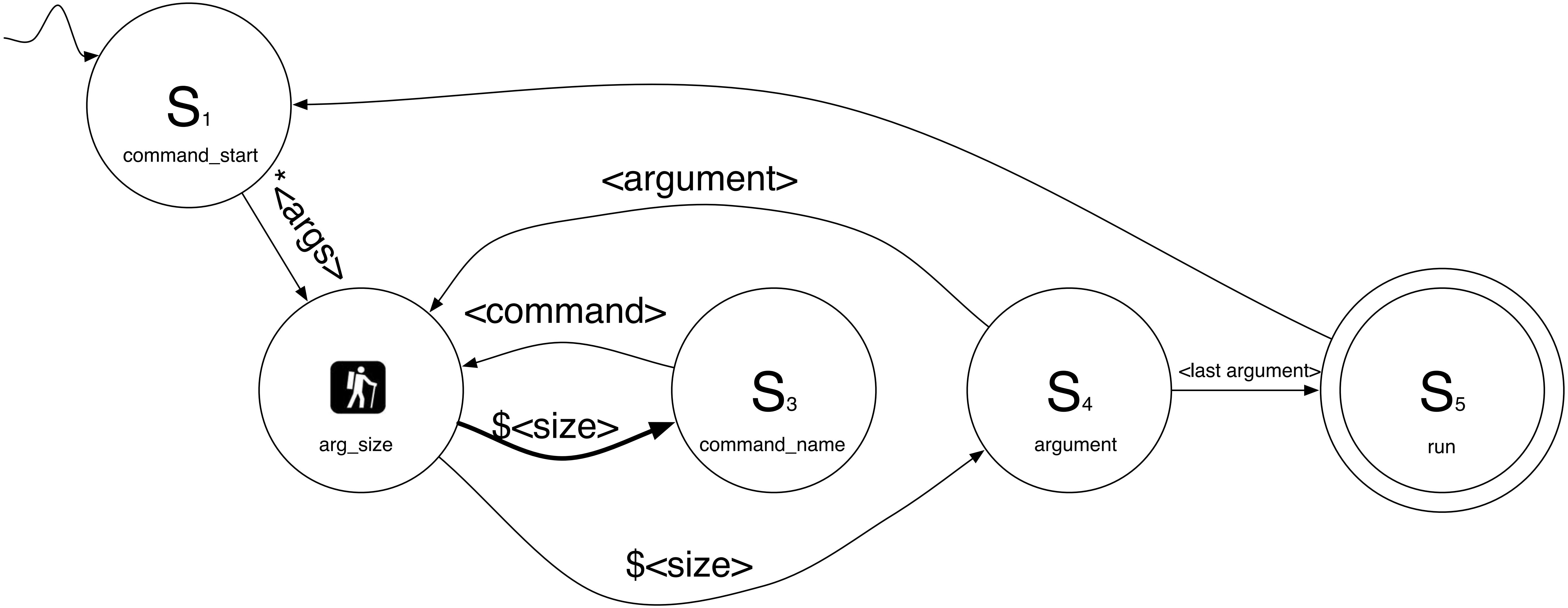
edis_client.erl

```
edis_client:command_start({data,Data},State).
```

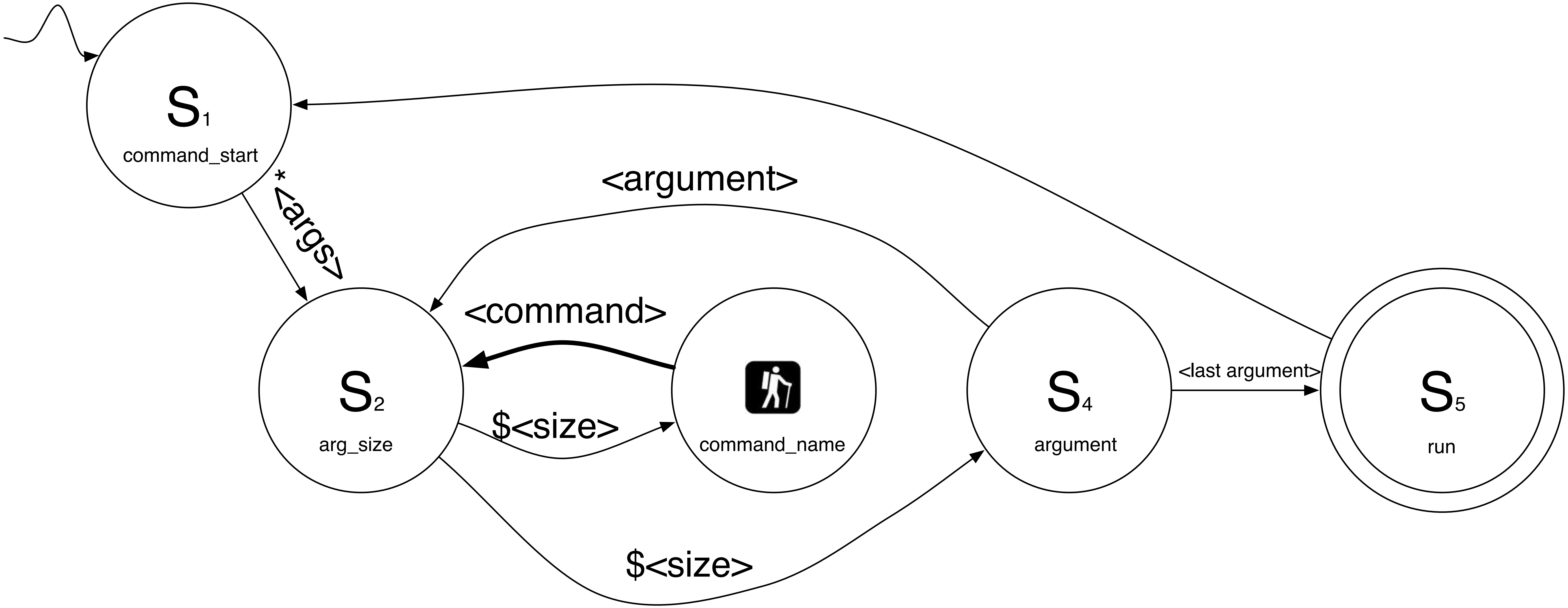
```
*DBG* <0.104.0> got {tcp,#Port<0.3714>,<<"*3\r\n">>} in state command_start
*DBG* <0.104.0> switched to state arg_size
```



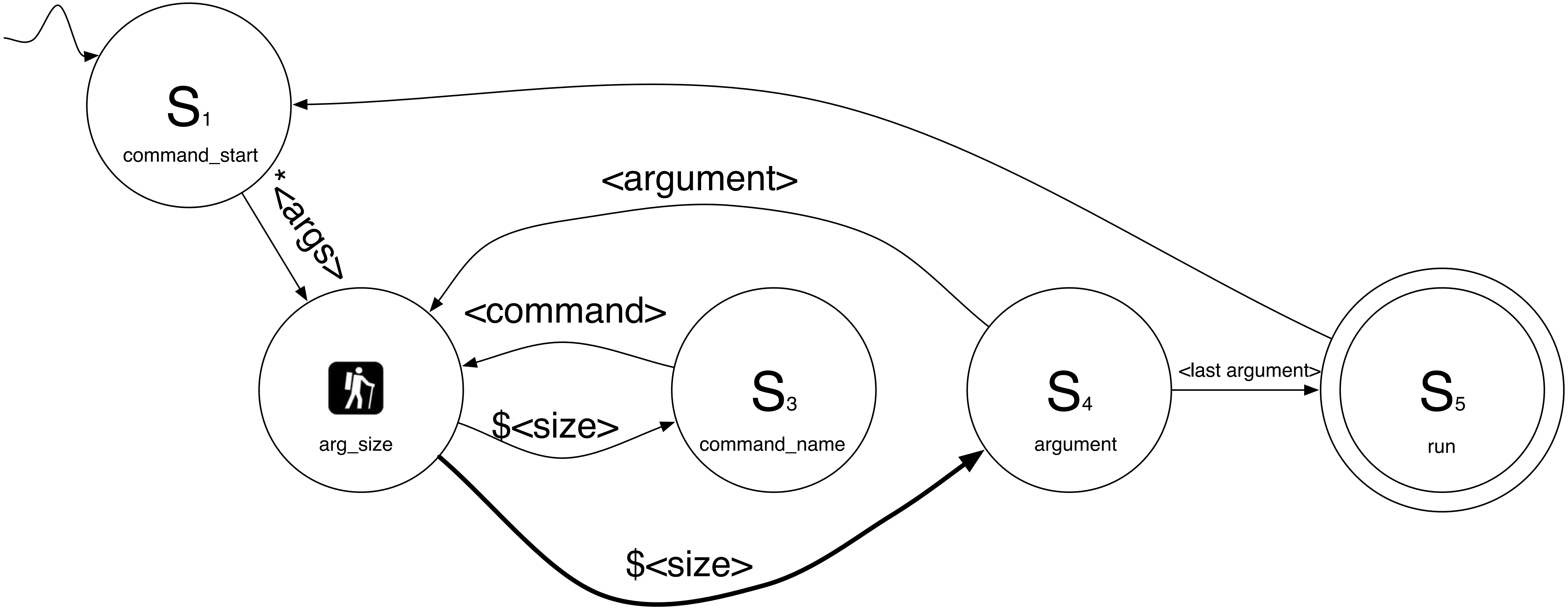
```
*DBG* <0.104.0> got {tcp,#Port<0.3714>,<<"$3\r\n">>} in state arg_size
*DBG* <0.104.0> switched to state command_name
```



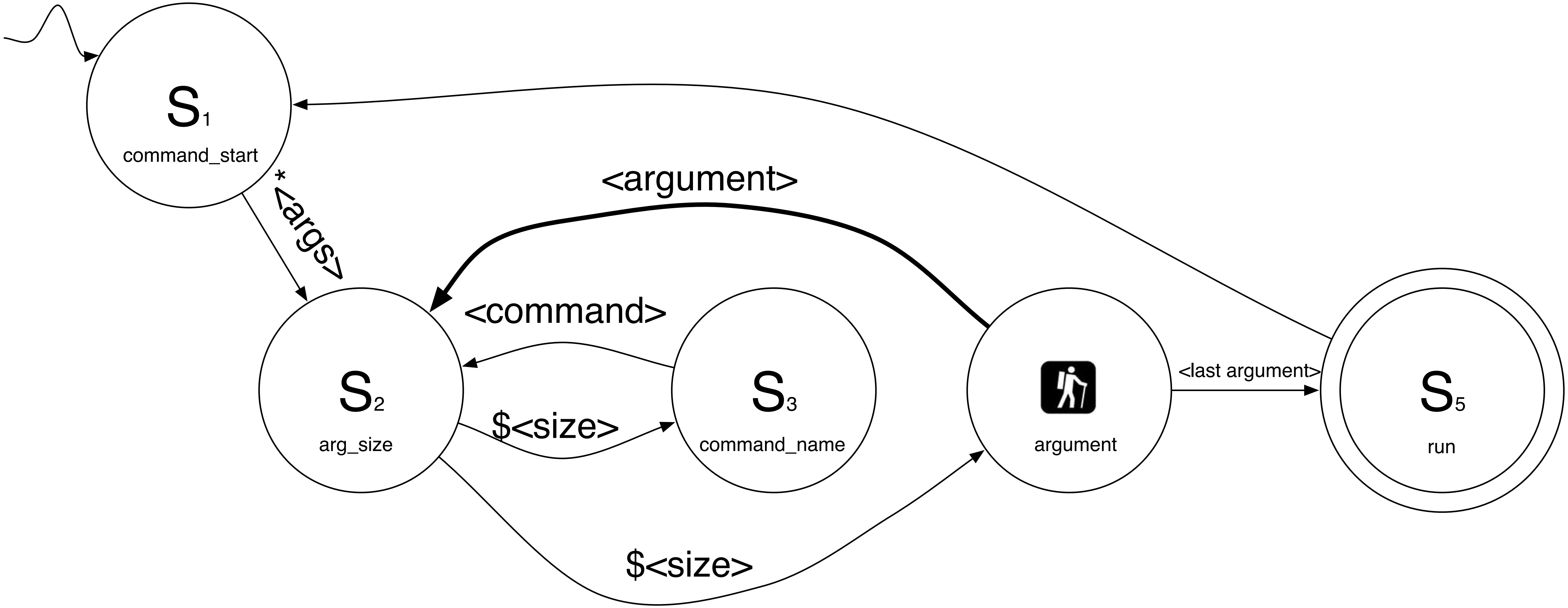
```
*DBG* <0.104.0> got {tcp,#Port<0.3714>,<<"set\r\n">>} in state command_name
*DBG* <0.104.0> switched to state arg_size
```



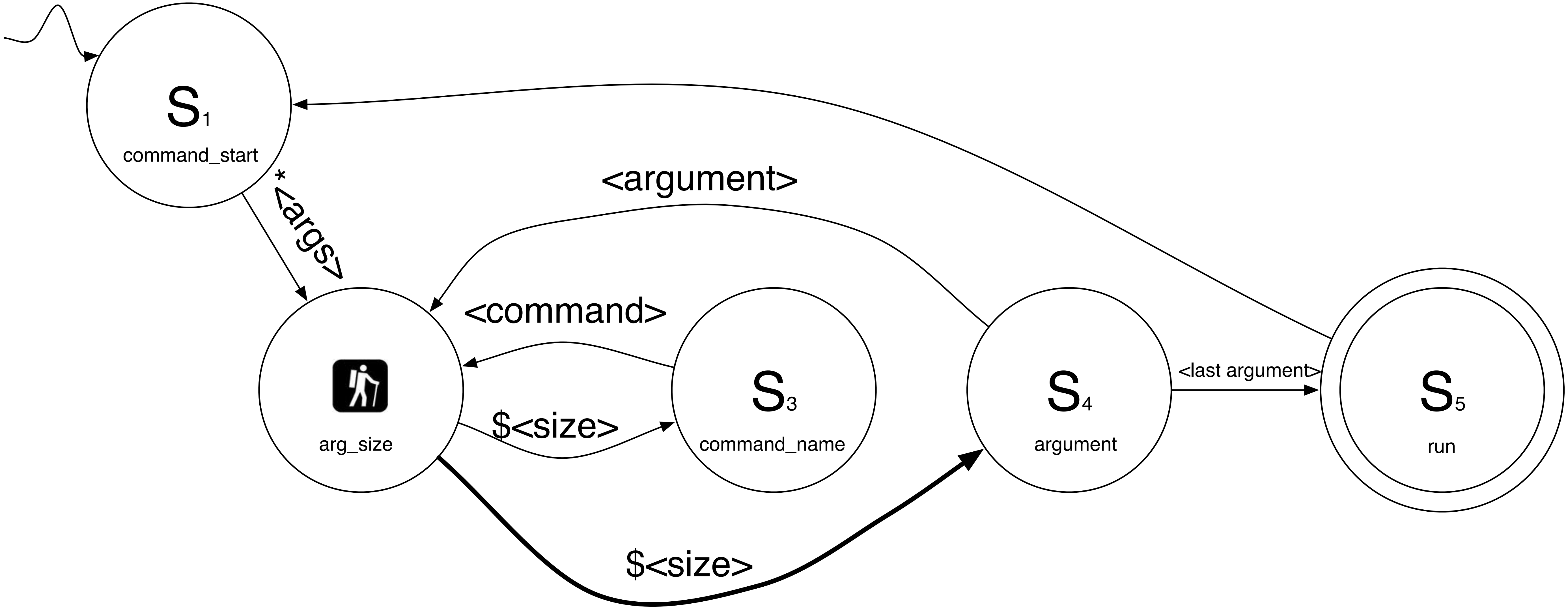
```
*DBG* <0.104.0> got {tcp,#Port<0.3714>,<<"$4\r\n">>} in state arg_size
*DBG* <0.104.0> switched to state argument
```



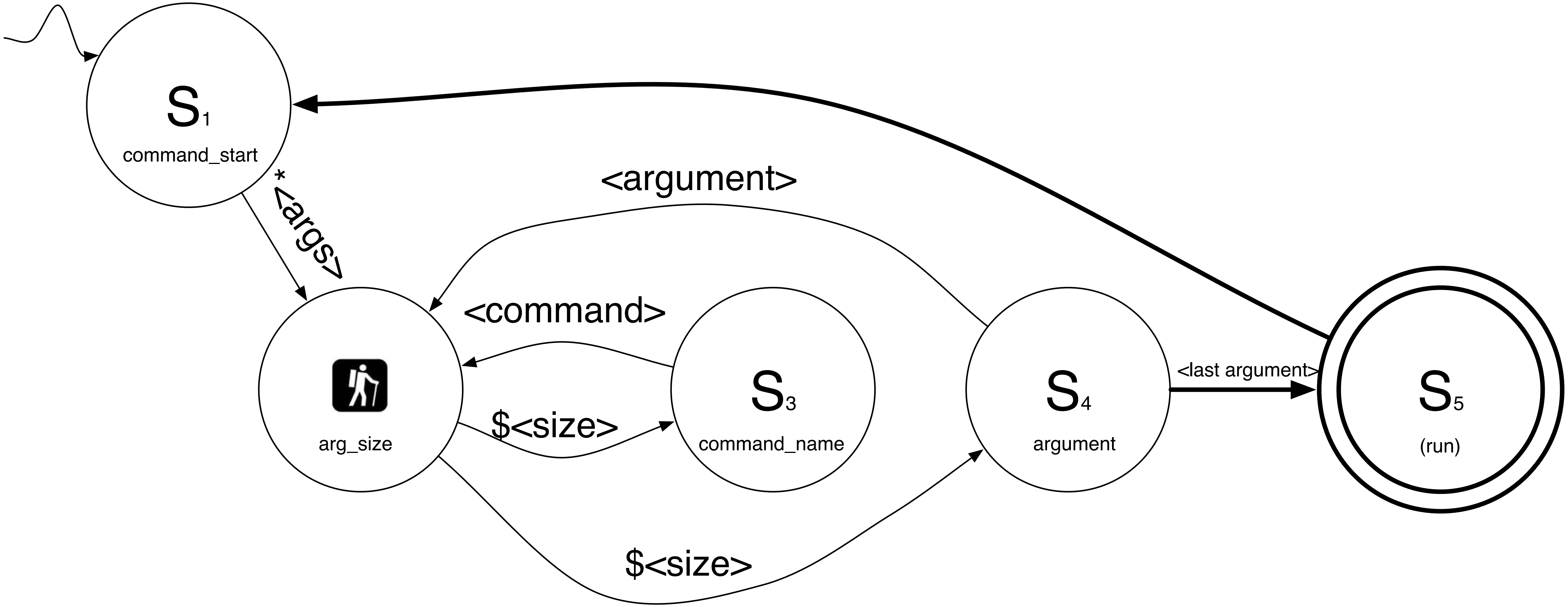
```
*DBG* <0.104.0> got {tcp,#Port<0.3714>,<<"lang\r\n">>} in state argument
*DBG* <0.104.0> switched to state arg_size
```



```
*DBG* <0.104.0> got {tcp,#Port<0.3714>,<<"$5\r\n">>} in state arg_size
*DBG* <0.104.0> switched to state argument
```



```
*DBG* <0.104.0> got {tcp,#Port<0.3714>,<<"erlang\r\n">>} in state argument
*DBG* <0.104.0> switched to state command_start
```

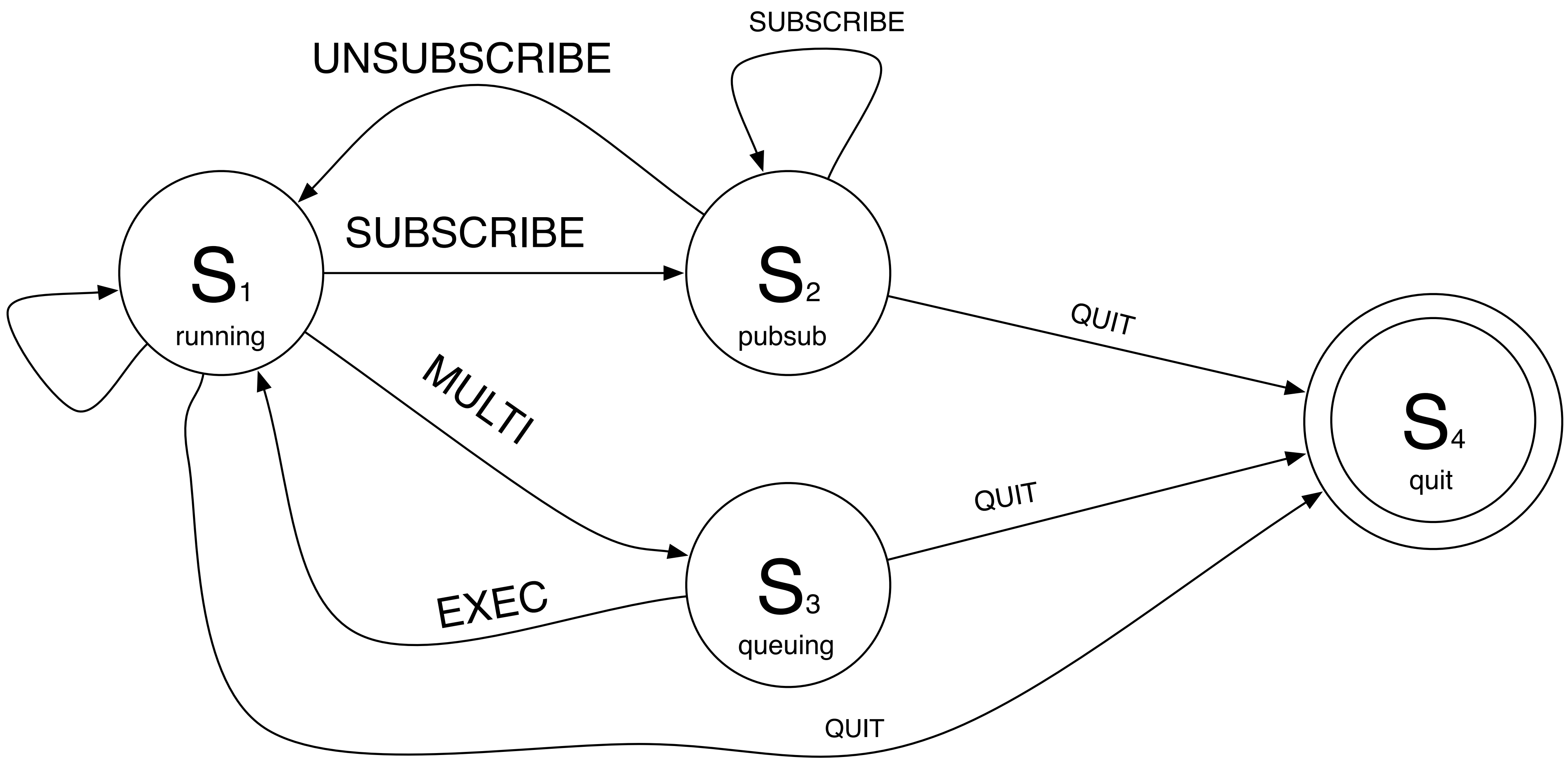


```
*DBG* <0.105.0> got cast {run,<<"SET">>,[<<"lang">>,<<"erlang">>]}
*DBG* <0.105.0> new state {state,#Port<0.3714>,
                        'edis-db-0',0,56068,true,undefined,[],undefined}
```

edis_client.erl

```
    edis_command_runner:run(State#state.command_runner,
         edis_util:upper(Command), []),
    {next_state, command_start, State, hibernate}
```

State Machine - Runner



command_runner

```erlang
handle_cast({run, Cmd, Args}, State) ->
  try
    OriginalCommand = #edis_command{cmd = Cmd,
                                    db = State#state.db_index,
                                    args = Args},

    Command = parse_command(OriginalCommand),

    ok = edis_db_monitor:notify(OriginalCommand),

    case {State#state.multi_queue, State#state.subscriptions} of
      {undefined, undefined} -> run(Command, State);
      {undefined, _InPubSub} -> pubsub(Command, State);
      {_InMulti, undefined} -> queue(Command, State);
      {_InMulti, _InPubSub} -> throw(invalid_context)
    end
  catch
```

command_runner

```
handle_cast({run, Cmd, Args}, State) ->
  try
    OriginalCommand = #edis_command{cmd = Cmd,
                                    db = State#state.db_index,
                                    args = Args},

    Command = parse_command(OriginalCommand),

    ok = edis_db_monitor:notify(OriginalCommand),

    case {State#state.multi_queue, State#state.subscriptions} of
      {undefined, undefined} -> run(Command, State);
      {undefined, _InPubSub} -> pubsub(Command, State);
      {_InMulti, undefined} -> queue(Command, State);
      {_InMulti, _InPubSub} -> throw(invalid_context)
    end
  catch
```

command_runner

```
handle_cast({run, Cmd, Args}, State) ->
  try
    OriginalCommand = #edis_command{cmd = Cmd,
                                    db = State#state.db_index,
                                    args = Args},

    Command = parse_command(OriginalCommand),

    ok = edis_db_monitor:notify(OriginalCommand),

    case {State#state.multi_queue, State#state.subscriptions} of
      {undefined, undefined} -> run(Command, State);
      {undefined, _InPubSub} -> pubsub(Command, State);
      {_InMulti, undefined} -> queue(Command, State);
      {_InMulti, _InPubSub} -> throw(invalid_context)
    end
  catch
```

edis_db

```erlang
run(Db, Command, Timeout) ->
  try gen_server:call(Db, Command, Timeout) of
    ok -> ok;
    {ok, Reply} -> Reply;
    {error, Error} ->
      throw(Error)
  catch
    _:{timeout, _} ->
      throw(timeout)
  end.
```

edis_db

```erlang
run(Db, Command, Timeout) ->
  try gen_server:call(Db, Command, Timeout) of
    ok -> ok;
    {ok, Reply} -> Reply;
    {error, Error} ->
      throw(Error)
  catch
    _:{timeout, _} ->
      throw(timeout)
  end.
```

edis_db

```erlang
handle_call(#edis_command{cmd = <<"MSET">>, args = KVs},
                _From, State) ->
  Reply =
    (State#state.backend_mod):write(
     State#state.backend_ref,
      [{put, Key,
        #edis_item{key = Key, encoding = raw,
                   type = string, value = Value}}
                     || {Key, Value} <- KVs]),
  {reply, Reply, stamp([K || {K, _} <- KVs], write, State)};
```

edis_db

```
handle_call(#edis_command{cmd = <<"MSET">>, args = KVs},
                _From, State) ->
  Reply =
    (State#state.backend_mod):write(
     State#state.backend_ref,
      [{put, Key,
        #edis_item{key = Key, encoding = raw,
                   type = string, value = Value}}
                     || {Key, Value} <- KVs]),
  {reply, Reply, stamp([K || {K, _} <- KVs], write, State)};
```

edis_eleveldb_backend.erl

```
write(#ref{db = Db}, Actions) ->
  ParseAction = fun({put, Key, Item}) ->
                     {put, Key, erlang:term_to_binary(Item)};
                   (Action) -> Action
                end,
  eleveldb:write(Db, lists:map(ParseAction, Actions), []).

```

edis_eleveldb_backend.erl

```
write(#ref{db = Db}, Actions) ->
  ParseAction = fun({put, Key, Item}) ->
                     {put, Key, erlang:term_to_binary(Item)};
                   (Action) -> Action
                end,
  eleveldb:write(Db, lists:map(ParseAction, Actions), []).

```

edis_command_runner.erl

```
run(C = #edis_command{result_type = ResType,
       timeout = Timeout, hooks = Hooks}, State) ->
  Res = edis_db:run(State#state.db, C);
```

edis_command_runner.erl

```erlang
case ResType of
 ok -> tcp_ok(State);
 string -> tcp_string(Res, State);
 bulk -> tcp_bulk(Res, State);
 multi_bulk -> tcp_multi_bulk(Res, State);
 number -> tcp_number(Res, State);
 boolean -> tcp_boolean(Res, State);
 float -> tcp_float(Res, State);
 sort -> tcp_sort(Res, State);
 zrange ->
  [_Key, _Min, _Max, ShowScores, Limit] = C#edis_command.args,
  tcp_zrange(Res, ShowScores, Limit, State)
end.
```

level_db

- Stores arbitrary byte arrays
- Data is stored sorted by key
- Three operations: Put/Get/Delete
- Multiple changes in atomic batch operations
- Data is automatically compressed
- Edis uses the Riak leveldb bindings

Performance

Performance

- Major testing with redis-benchmark
- Custom benchmark code
- All testing with physical servers
- Intel i5 760 quad-core @ 2.8 GHz
- Erlang R14B04

Performance

It's important to remember that edis respects Redis's goals of algorithmic complexity.

If a Redis command is O(log(n)), Edis will have the same O().*

* Except for ZSETS - We don't yet have skiplists in Erlang

Performance (Operations/second)

	Redis	In-Memory Edis	% slower
PING (inline)	120,734	40,741	296%
PING	129,892	32,956	394%
MSET (10 keys)	73,825	6,662	1,108%
SET	135,160	22,051	613%
GET	134,282	23,127	581%
INCR	138,916	24,421	569%
LPUSH	137,990	21,397	645%
LPOP	130,769	22,728	575%
SADD	135,160	21,860	618%
SPOP	132,456	25,707	515%
LRANGE (first 100 elements)	65,362	1,783	3,667%

Performance (Operations/second)

	Redis	In-Memory Edis	% slower
PING (inline)	120,734	40,741	296%
PING	129,892	32,956	394%
MSET (10 keys)	73,825	6,662	1,108%
SET	135,160	22,051	613%
GET	134,282	23,127	581%
INCR	138,916	24,421	569%
LPUSH	137,990	21,397	645%
LPOP	130,769	22,728	575%
SADD	135,160	21,860	618%
SPOP	132,456	25,707	515%
LRANGE (first 100 elements)	65,362	1,783	3,667%

Performance (Operations/second)

	Redis	LevelDB Edis	% slower
PING (inline)	120,734	41,152	293%
PING	129,892	32,419	401%
MSET (10 keys)	73,825	6,058	1,219%
SET	135,160	20,726	652%
GET	134,282	21,463	626%
INCR	138,916	17,930	775%
LPUSH	137,990	226	61,105%
LPOP	130,769	229	57,092%
SADD	135,160	9,003	1,501%
SPOP	132,456	1,298	10,205%
LRANGE (first 100 elements)	65,362	644	10,143%

Performance (Operations/second)

	Redis	LevelDB Edis	% slower
PING (inline)	120,734	41,152	293%
PING	129,892	32,419	401%
MSET (10 keys)	73,825	6,058	1,219%
SET	135,160	20,726	652%
GET	134,282	21,463	626%
INCR	138,916	17,930	775%
LPUSH	137,990	226	61,105%
LPOP	130,769	229	57,092%
SADD	135,160	9,003	1,501%
SPOP	132,456	1,298	10,205%
LRANGE (first 100 elements)	65,362	644	10,143%

Using (and Extending) Edis

Extending with Hooks

- Similar to Riak "post commit" hooks
- In-process
- Only operates on lists (sets/hashes TBD)
- Set in Erlang config-file - no realtime creation
- Not officially merged yet

Extending with Hooks

- Can implement the "Reliable Queue" pattern
- Could interface with RabbitMQ
- Could do "additional resource checks"

What's next

- Performance Improvements
- Support for Master/Slave
- Roadmap for Multi-Master Replication
- Custom command runners
- RabbitMQ Hooks
- Riak backend support

For additional fun...

- Look at the edis source
- Lots of Benchmarks w/ common test
- We're using:
  * -extends directive
  * Parameterized Modules
  * Custom behaviors
  * Ascii Art

Thanks!

- github.com/inaka/edis
- @chaddepue
- chad@inaka.net