/

Building a Multi-Master, Replicated Redis Clone with Erlang/OTP

Chad DePue

Founder of **Inaka**
CTO of **WhisperText**
Enjoy Erlang
Love Redis
@chaddepue

Who is Inaka?

We are an international team of 25 developers, building high performance iOS/Android apps, using Ruby, Erlang, Elixir, Node.

What is Whisper?

Top 10 Social Network

What is 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

I love Redis!

Edis - an Erlang-based server that...

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

Why is Redis Great?

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

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

RPOPLPUSH

work_queue 1 , 2 , 7, 10

RPOPLPUSH

work_queue 1 , 2 , 7, 10

run_queue 18

RPOPLPUSH

work_queue 1 , 2 , 7, 10

run_queue 18

RPOPLPUSH

work_queue 1, 2 , 7

run_queue 10, 18

RPOPLPUSH

work_queue 1, 2 , 7

run_queue 10, 18

Atomic

Redis Design Decisions

- Data must fit in-memory
- Master-slave model
- Scripting Blocks the Thread

Edis Design Decisions

- Disk-backed by default
- Pluggable DB
- Master-Slave or Master-Master
- Extensible with Erlang

Edis Design Decisions

It's important 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.

Edis Use Cases

- Smart Redis Proxy
- Handy Interface on top of existing data
- Reference Protocol Implementation

We're going to ...

- Learn Erlang in 45 seconds
- Review Edis Architecture
- Trace a SET command
- Learn about Replication and MM
- Demo Multi-Master (time permitting)

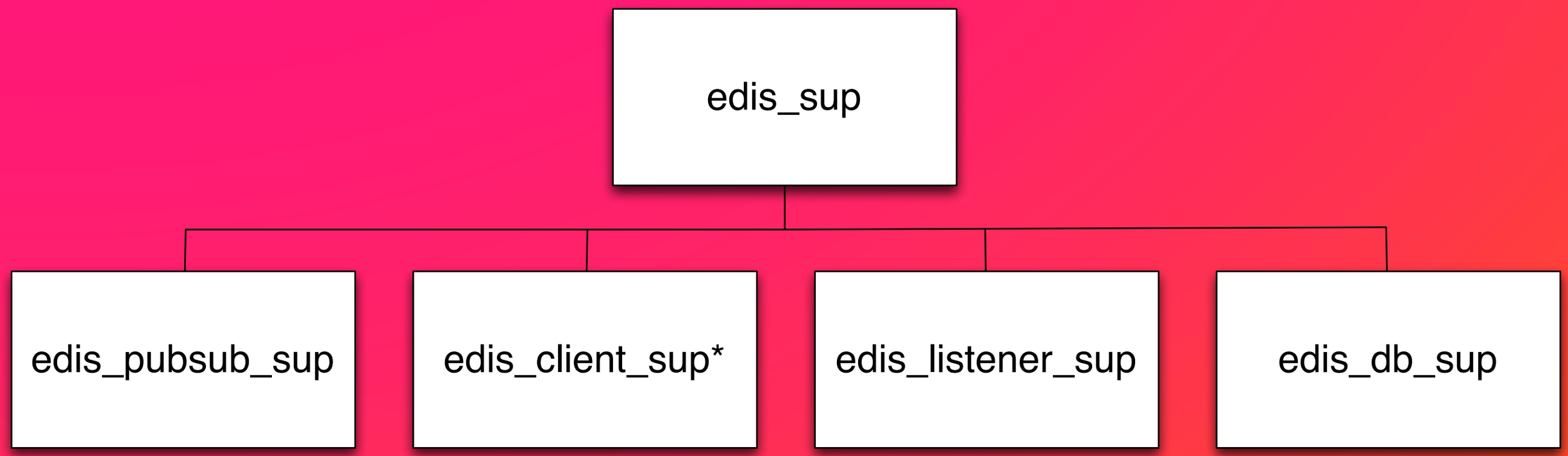
Key terms in Erlang/ OTP in 45 seconds

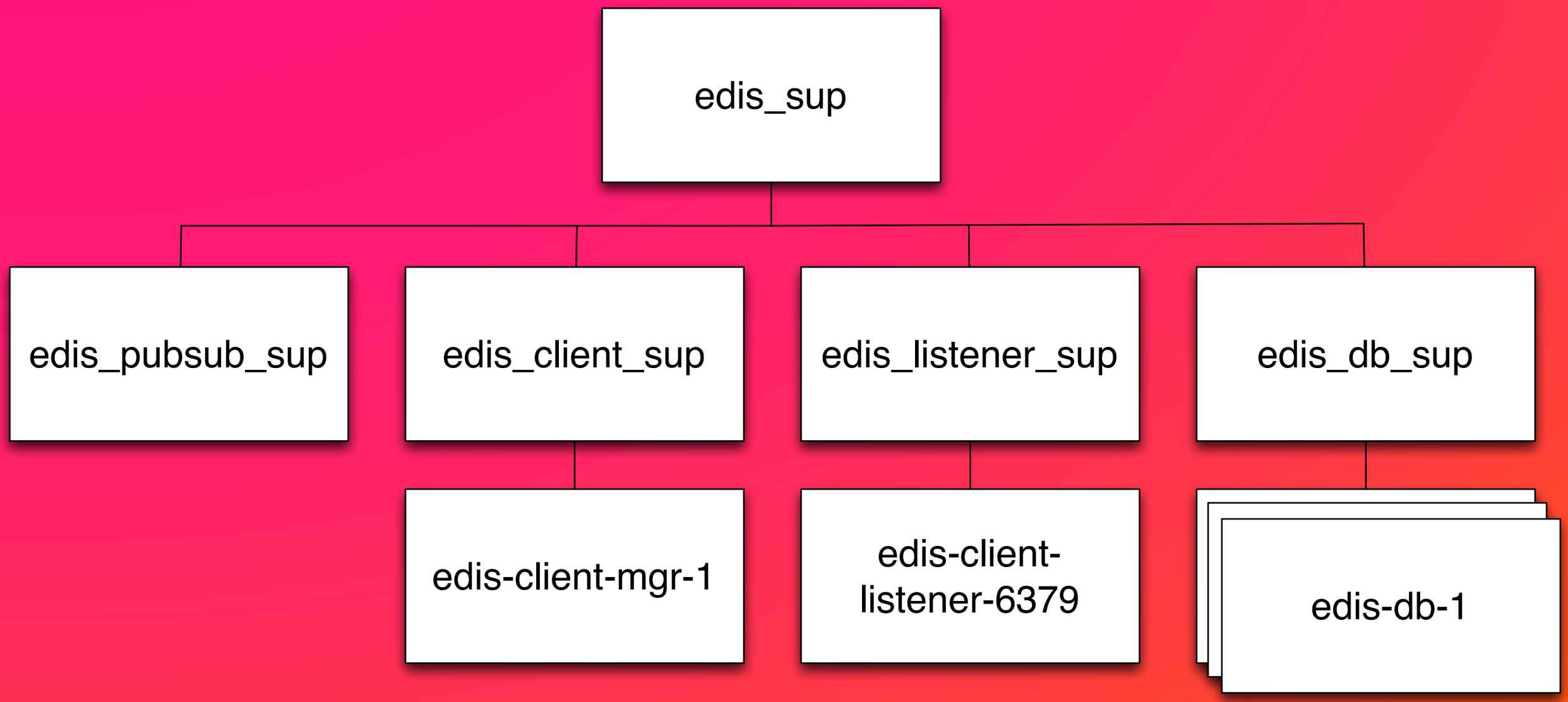
- pattern matching
- variable immutability
- lightweight processes + messages
- semicolon; comma, period.
- OTP provides:
    - gen_server, gen_fsm
    - supervisors (and more)

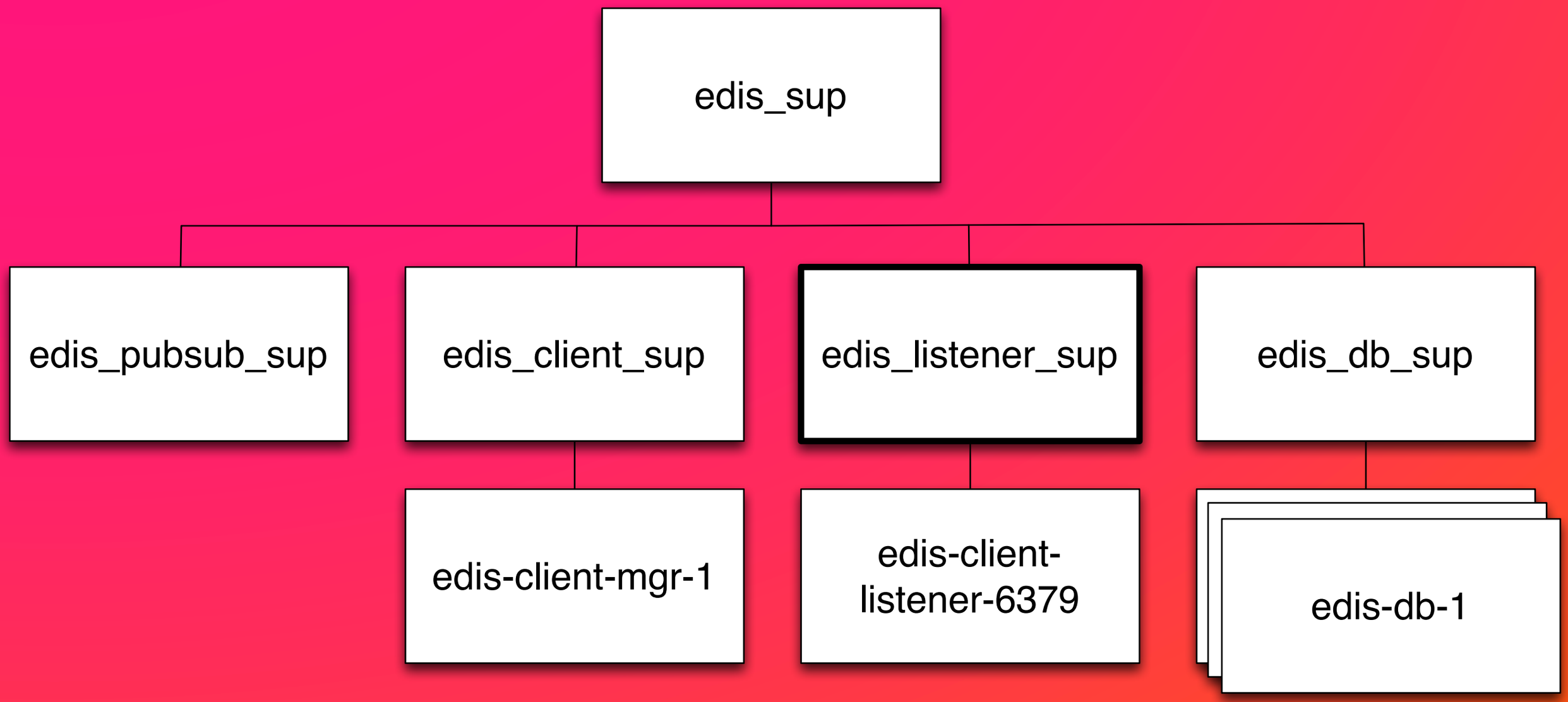
Get All 3 Major Erlang Books

Armstrong Book	(Become a Berliever)
Erlang Programming	(Architecture)
Erlang and OTP in Action	(Practical)

Structure of the App

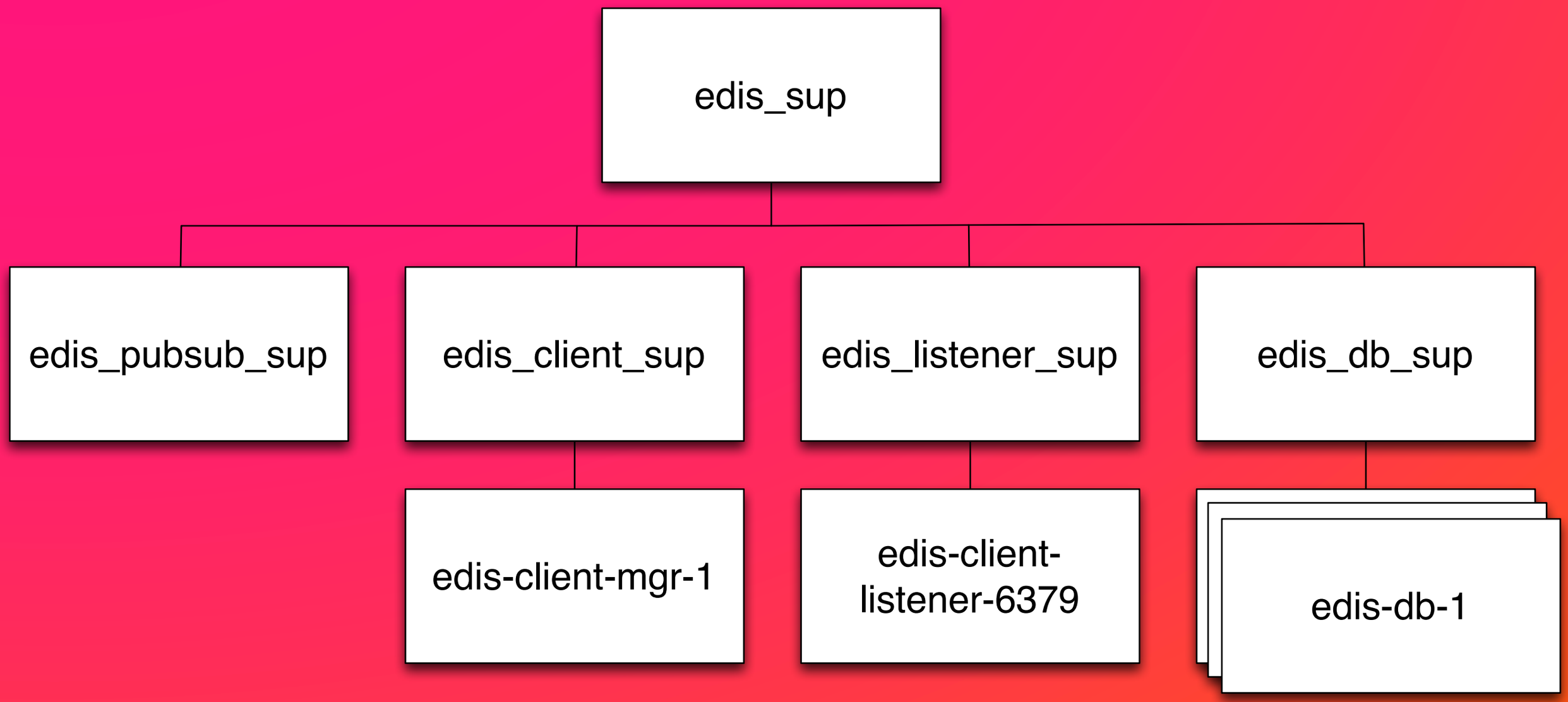

edis_sup
edis_pubsub_sup
edis_client_sup*
edis_listener_sup
edis_db_sup

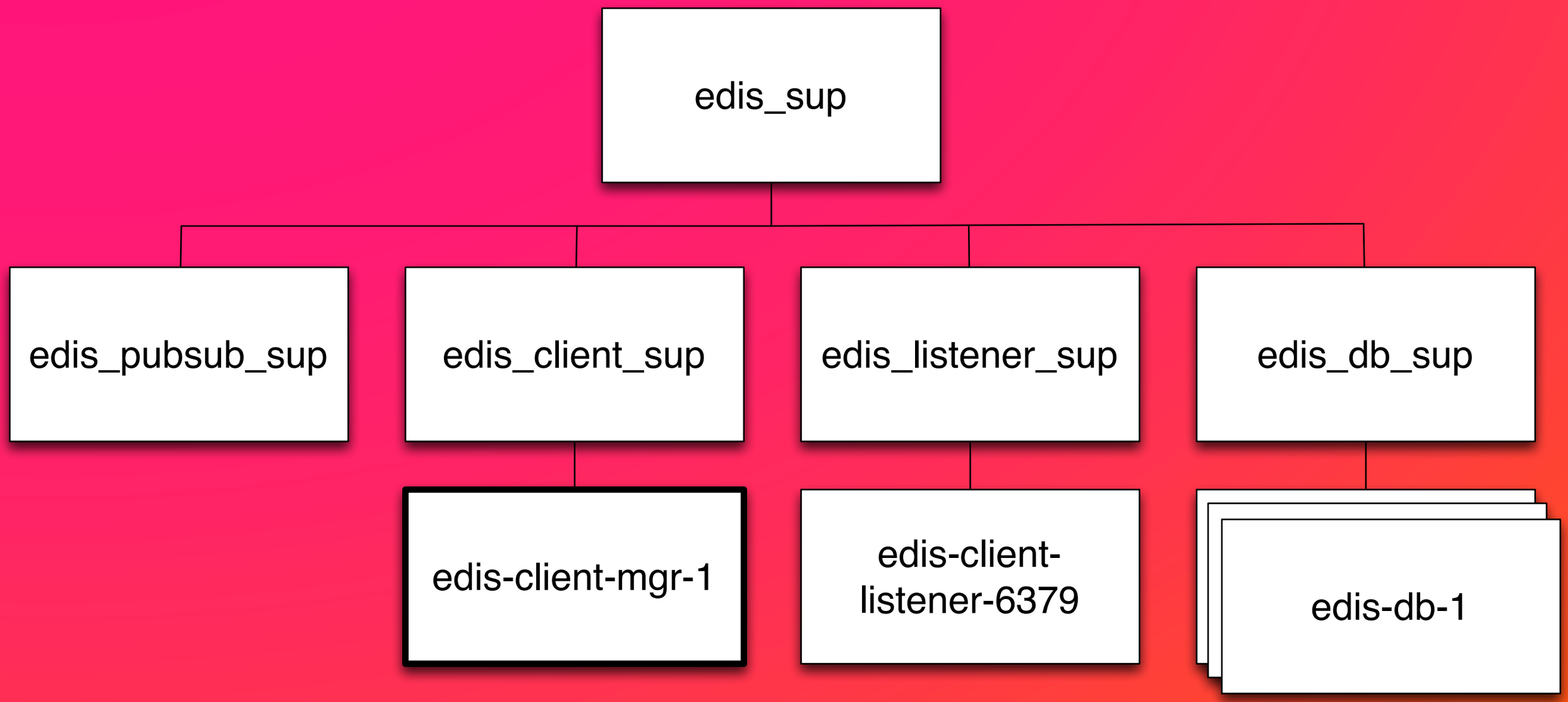

edis_sup
edis_pubsub_sup
edis_client_sup
edis_listener_sup
edis_db_sup
edis-client-mgr-1
edis-client-
listener-6379
edis-db-1

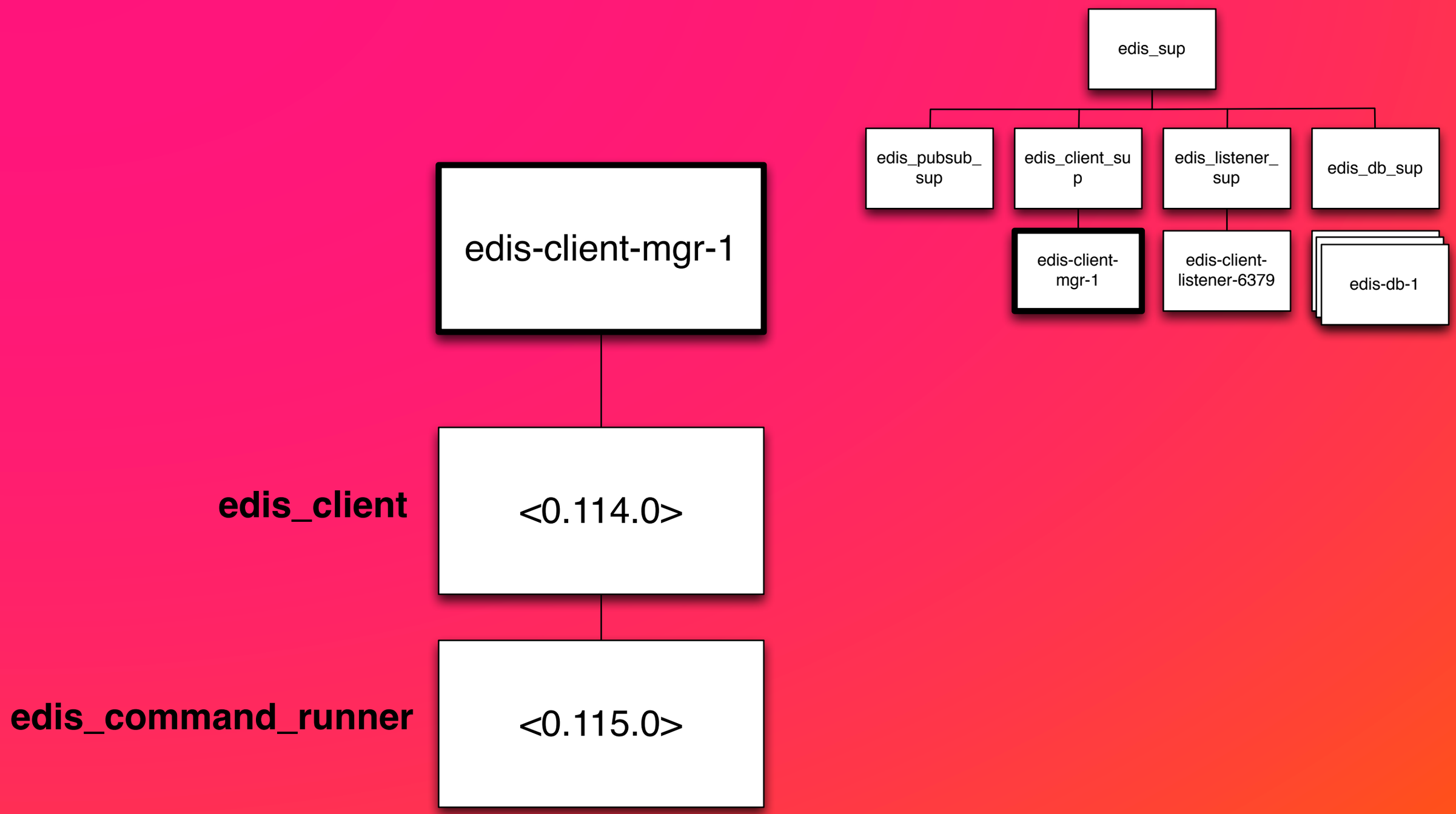

edis_sup
edis_pubsub_sup
edis_client_sup
edis_listener_sup
edis_db_sup
edis-client-mgr-1
edis-client-
listener-6379
edis-db-1

```
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}}.
```

```
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_sup
edis_pubsub_sup
edis_client_sup
edis_listener_sup
edis_db_sup
edis-client-mgr-1
edis-client-
listener-6379
edis-db-1


edis_sup
edis_pubsub_sup
edis_client_sup
edis_listener_sup
edis_db_sup
edis-client-mgr-1
edis-client-
listener-6379
edis-db-1


edis_sup
edis_pubsub_sup
edis_client_sup
edis_listener_sup
edis_db_sup
edis-client-mgr-1
edis-client-listener-6379
edis-db-1
edis-client-mgr-1
edis_client
<0.114.0>
edis_command_runner
<0.115.0>

```
64 socket({socket_ready, Socket}, State) ->
65   % Now we own the socket
66   PeerPort = inet:peername(Socket),
67
68   ok = inet:setopts(Socket, [{active, once},
69                              {packet, line}, binary]),
70   _ = erlang:process_flag(trap_exit, true),
71   {ok, CmdRunner} = edis_command_runner:start_link(Socket),
72   {next_state, command_start,
73      State#state{socket         = Socket,
74                  peerport       = PeerPort,
75                  command_runner = CmdRunner}, hibernate};
```

edis_client.erl

```
64 socket({socket_ready, Socket}, State) ->
65   % Now we own the socket
66   PeerPort = inet:peername(Socket),
67
68   ok = inet:setopts(Socket, [{active, once},
69                              {packet, line}, binary]),
70   _ = erlang:process_flag(trap_exit, true),
71   {ok, CmdRunner} = edis_command_runner:start_link(Socket),
72   {next_state, command_start,
73      State#state{socket         = Socket,
74                  peerport       = PeerPort,
75                  command_runner = CmdRunner}, hibernate};
```

edis_client.erl

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

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

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

edis_client.erl

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

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

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

edis_client.erl

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

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

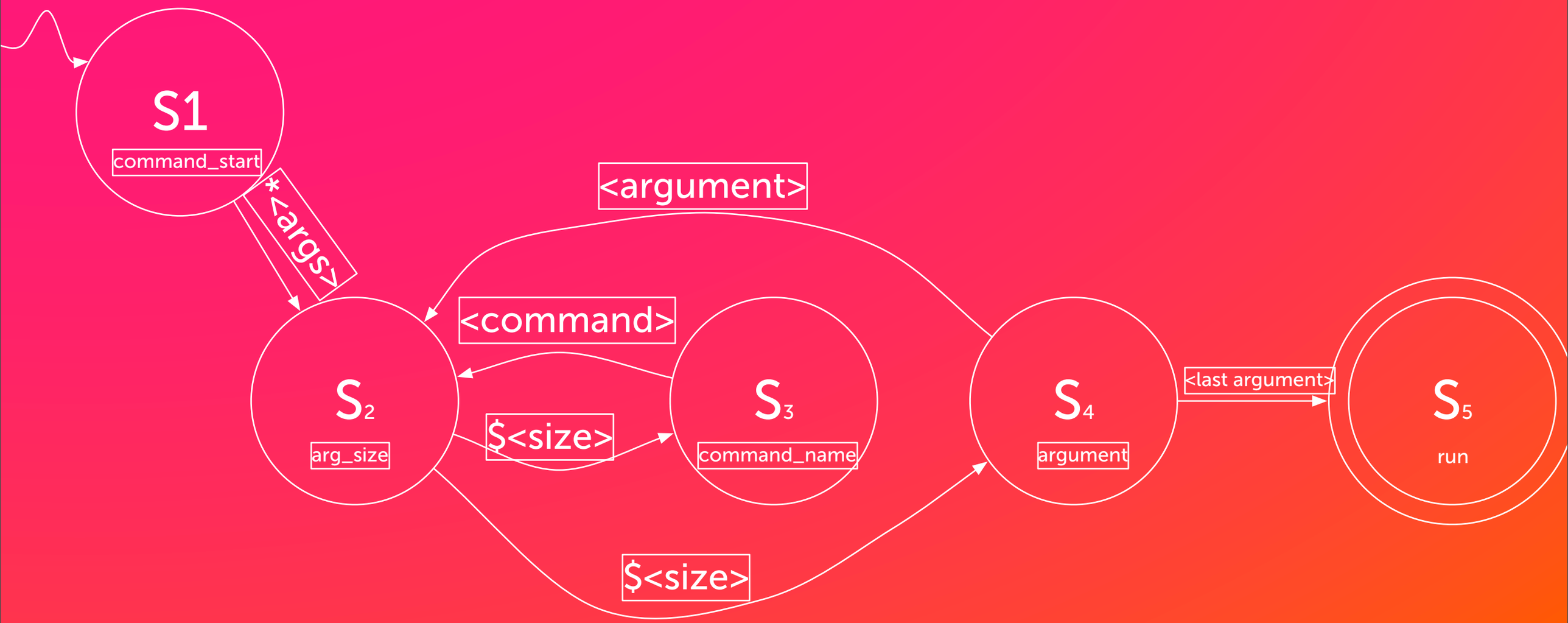
edis_client.erl

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

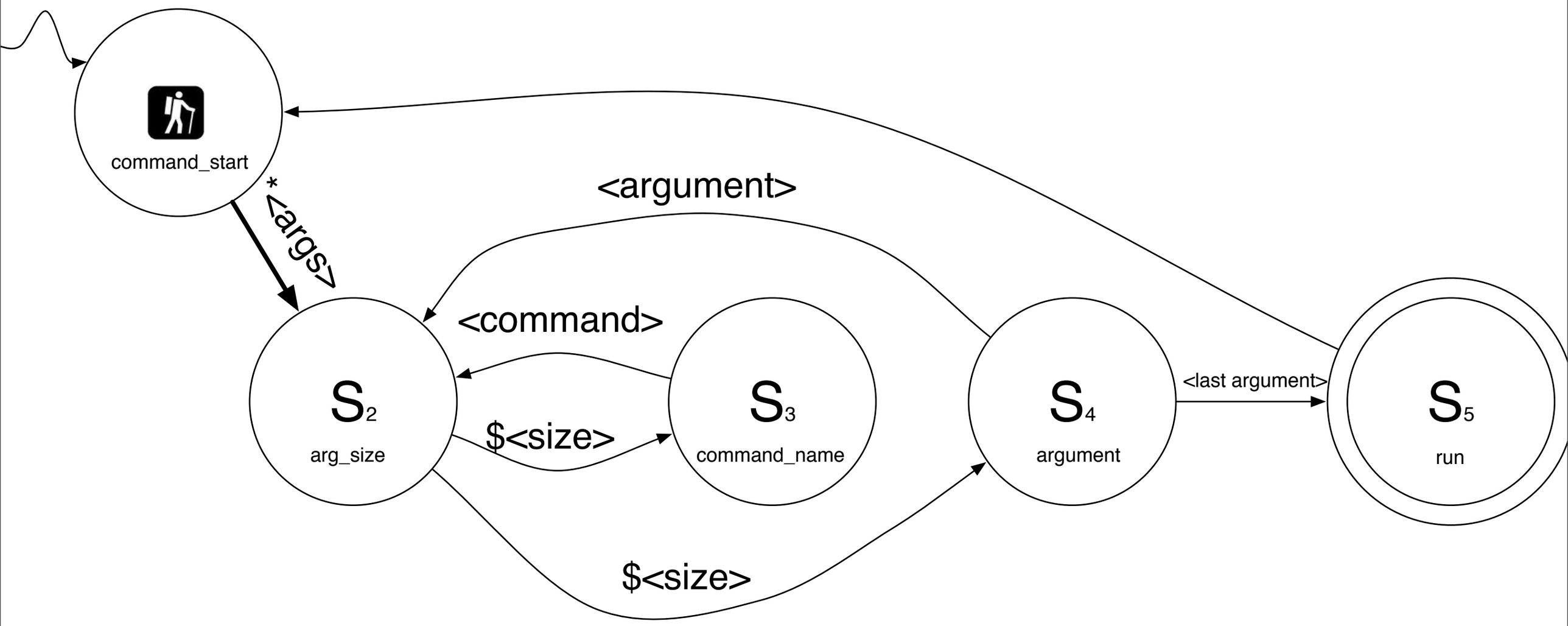
Redis Protocol

"SET lang erlang"

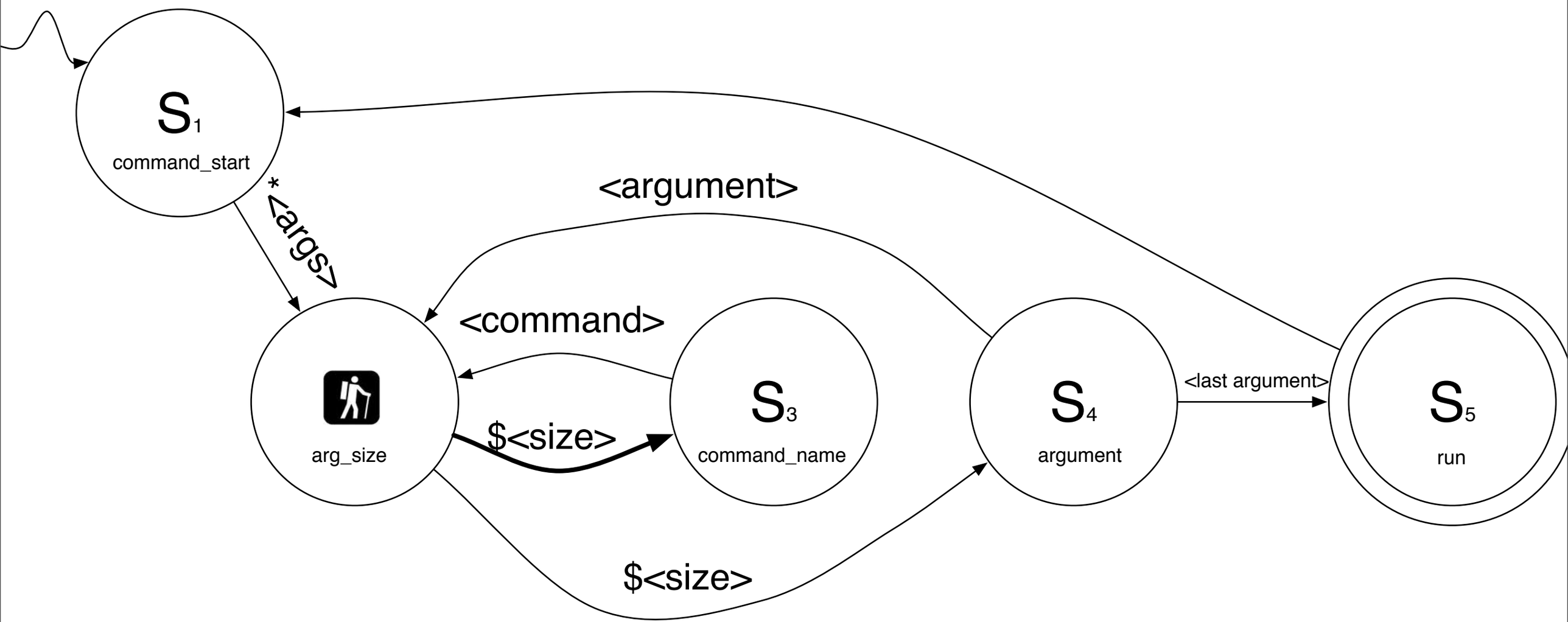
Redis Protocol



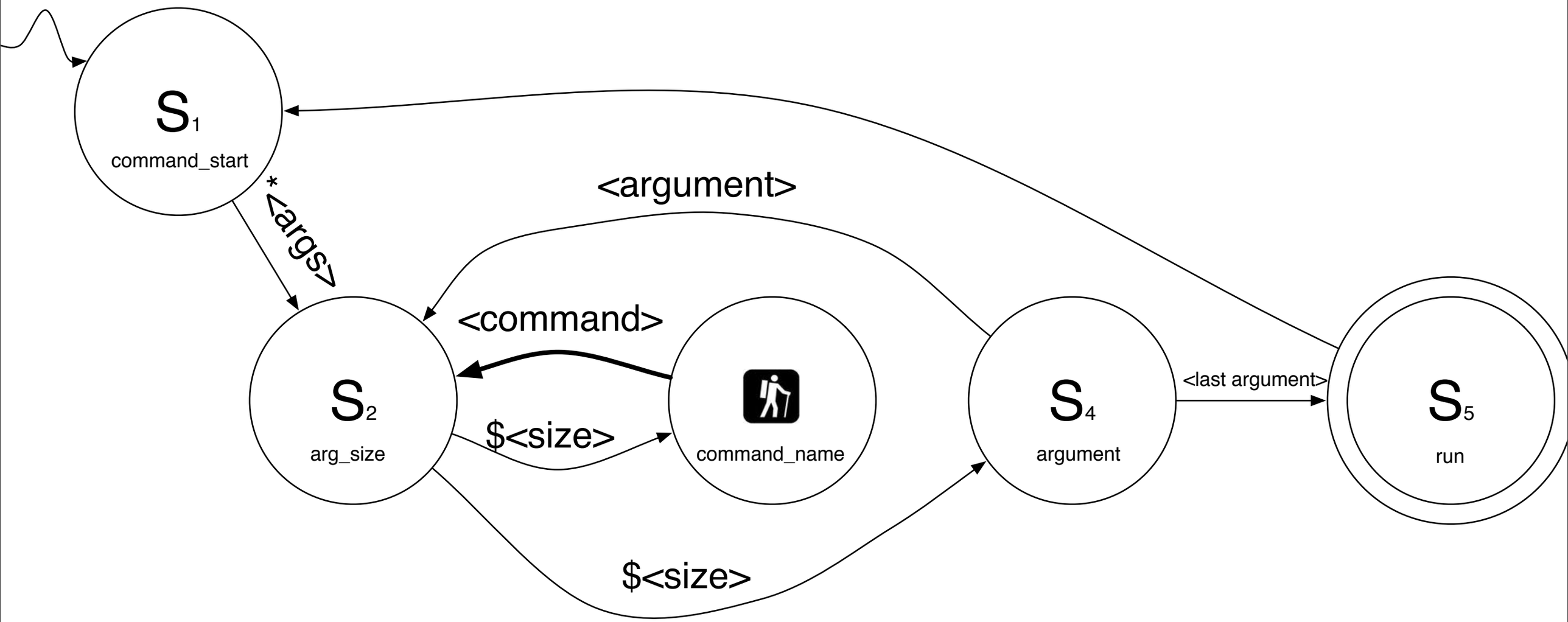
```
*DBG* <0.104.0> got {tcp,#Port<0.3714>,<<"*3\r\n">>} in state command_start
*DBG* <0.104.0> switched to state arg_size
```



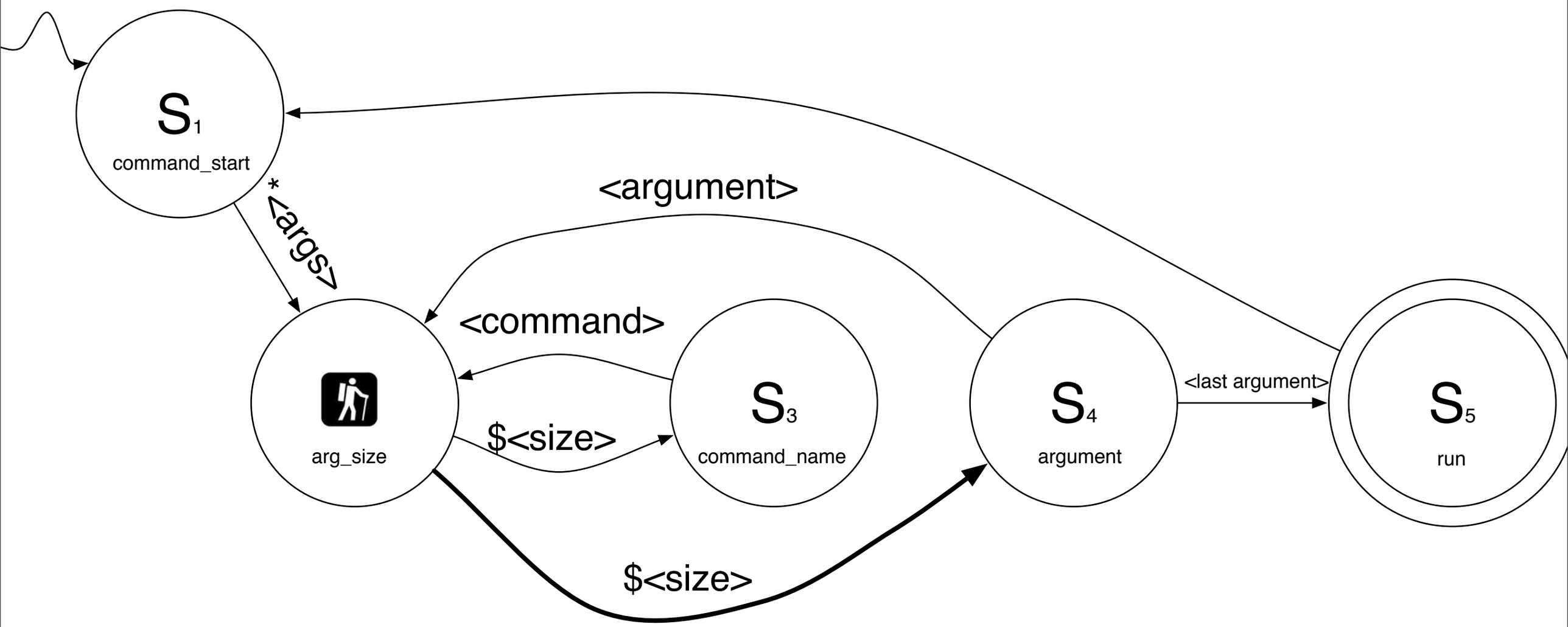
```
*DBG* <0.104.0> got {tcp,#Port<0.3714>,<<"$3\r\n">>} in state arg_size
*DBG* <0.104.0> switched to state command_name
```



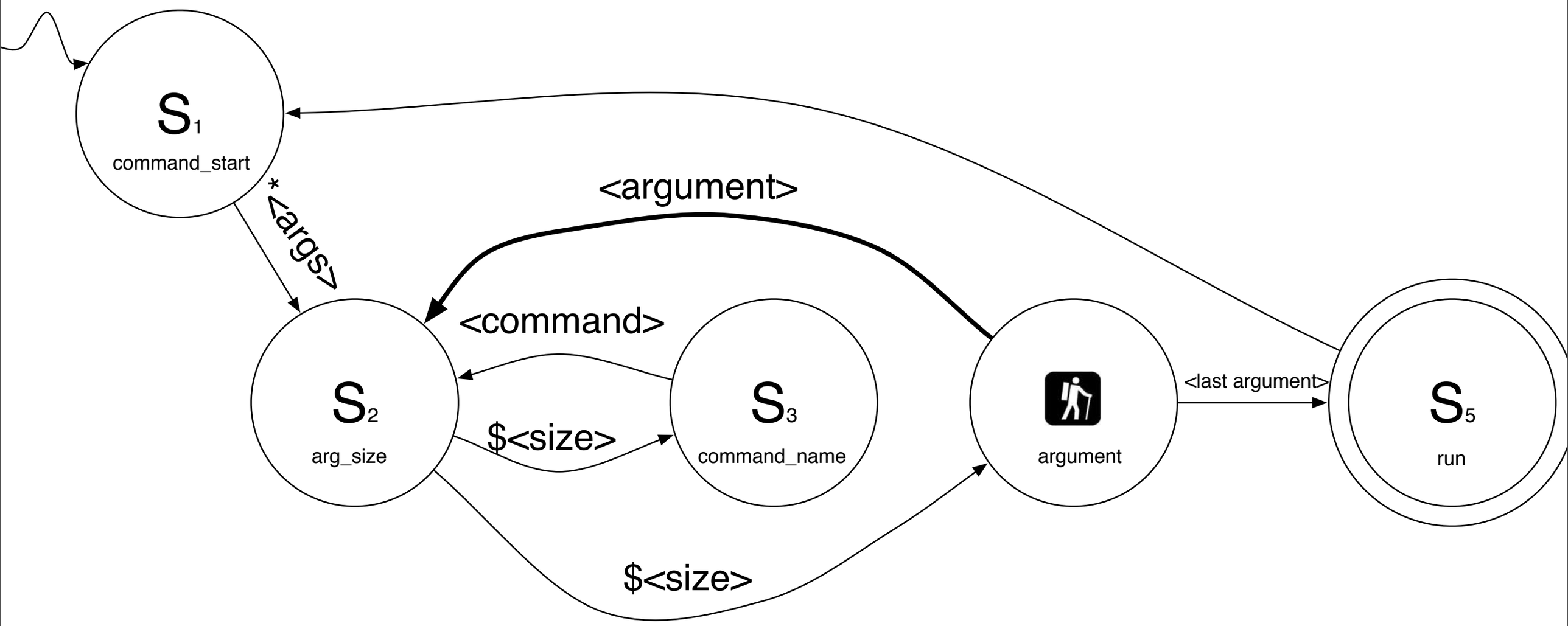
```
*DBG* <0.104.0> got {tcp,#Port<0.3714>,<<"set\r\n">>} in state command_name
*DBG* <0.104.0> switched to state arg_size
```



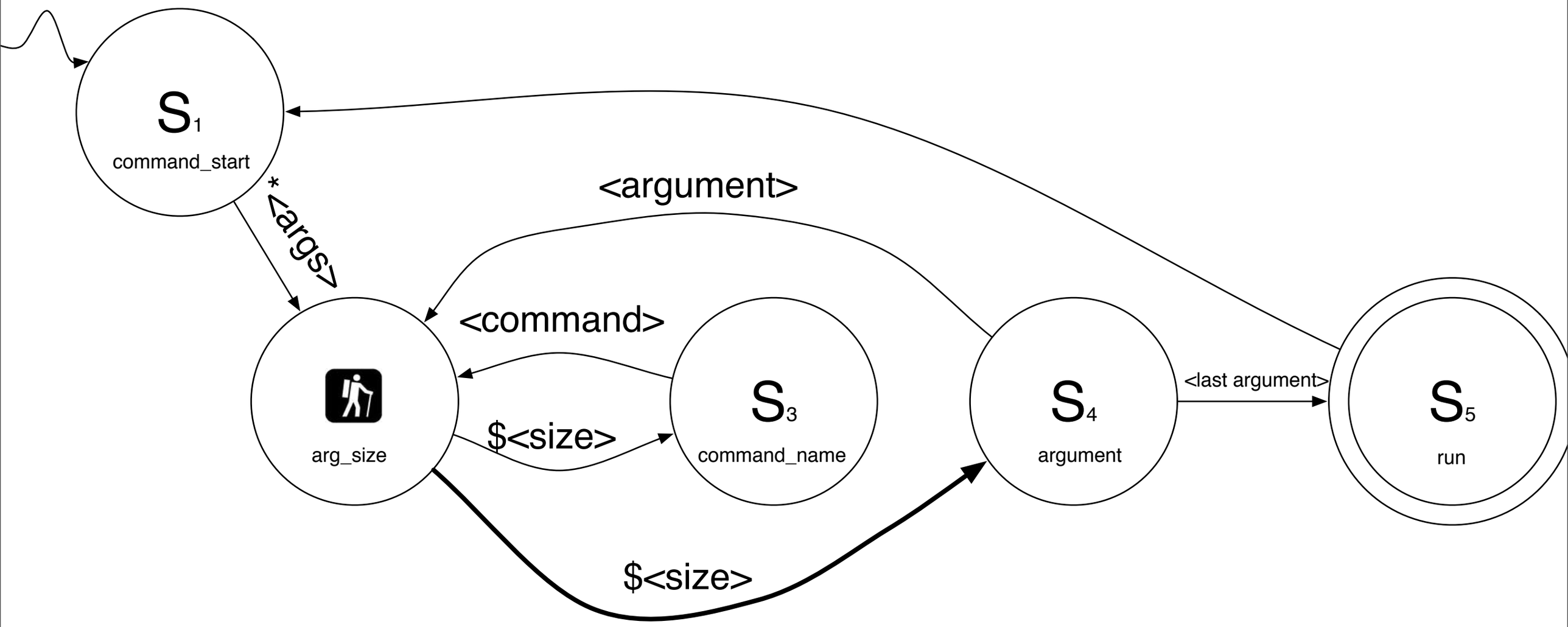
```
*DBG* <0.104.0> got {tcp,#Port<0.3714>,<<"$4\r\n">>} in state arg_size
*DBG* <0.104.0> switched to state argument
```



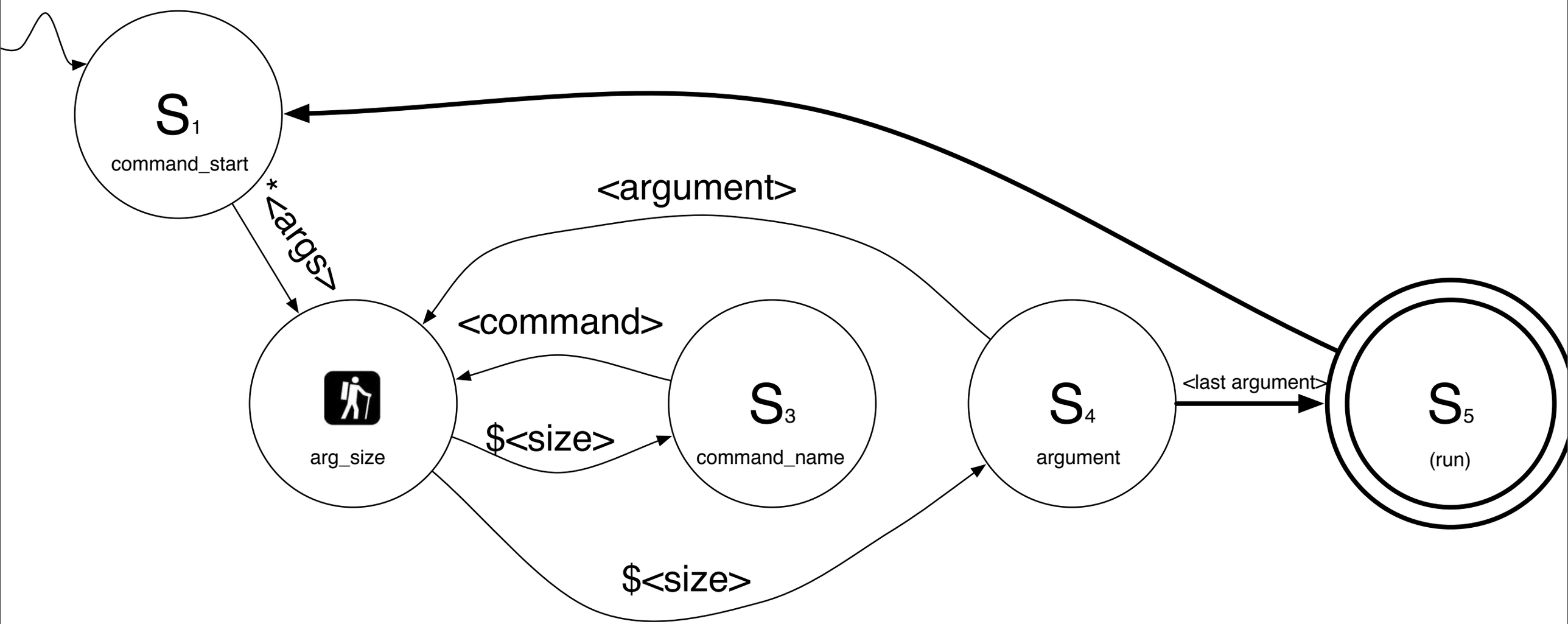
```
*DBG* <0.104.0> got {tcp,#Port<0.3714>,<<"lang\r\n">>} in state argument
*DBG* <0.104.0> switched to state arg_size
```



```
*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
```

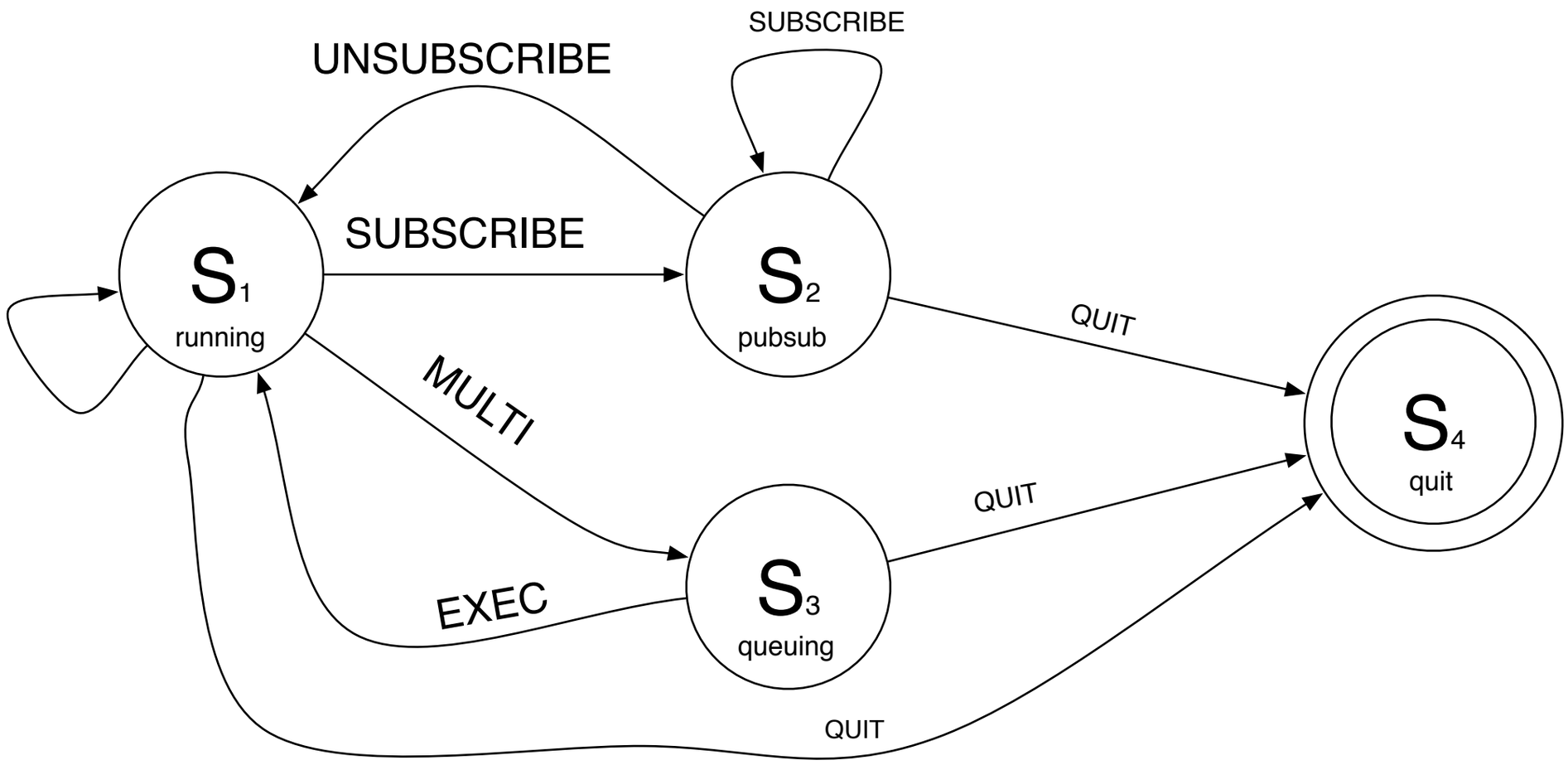


```
*DBG* <0.105.0> got cast {run,<<"SET">>,[<<"lang">>,<<"erlang">>]}
*DBG* <0.105.0> new state {state,#Port<0.3714>,
                           'edis-db-0',0,56068,true,undefined,[],undefined}
```

edis_client.erl

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

State Machine - Runner



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
```

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

```
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

```
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

```
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)};
```

```
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), []).
```

```
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

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

edis_command_runner.erl

```
716    case ResType of
717      ok -> tcp_ok(State);
718      string -> tcp_string(Res, State);
719      bulk -> tcp_bulk(Res, State);
720      multi_bulk -> tcp_multi_bulk(Res, State);
721      number -> tcp_number(Res, State);
722      boolean -> tcp_boolean(Res, State);
723      float -> tcp_float(Res, State);
724      sort -> tcp_sort(Res, State);
725      zrange ->
726        [_Key, _Min, _Max, ShowScores, Limit] = C#edis_command.args,
727        tcp_zrange(Res, ShowScores, Limit, State)
728    end.
```

LevelDB

- 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

(LevelDB is not the only backend, currently, in-memory and HanoiDB)

So what is different?

- SAVE, BGSAVE, LASTSAVE: database dependent
- MULTI doesn't support cross-db/non-db
- SLAVEOF not fully supported
- Encoding is inefficient

Master/Slave Replication

Multi-Master Behavior

Replication

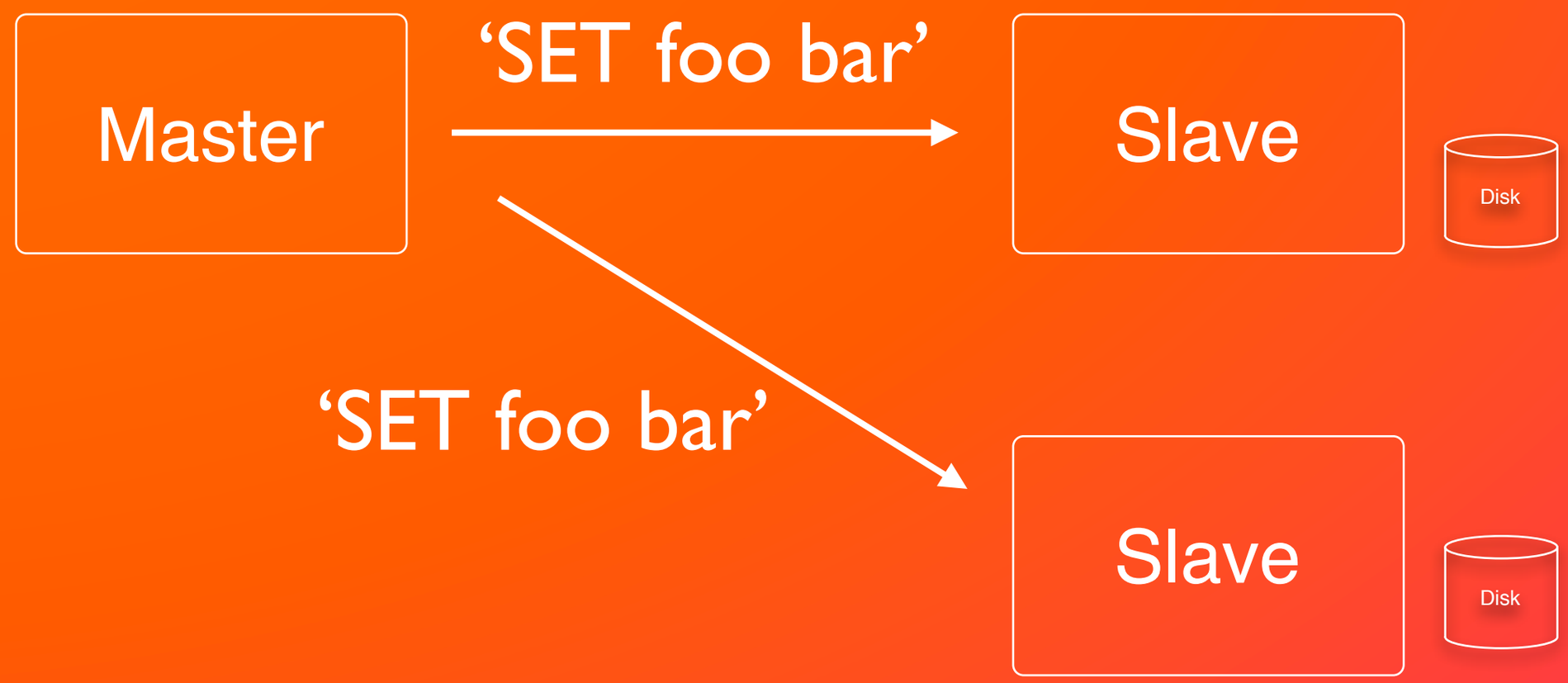
Slave sends SYNC command

Master (may) flush RDB to disk

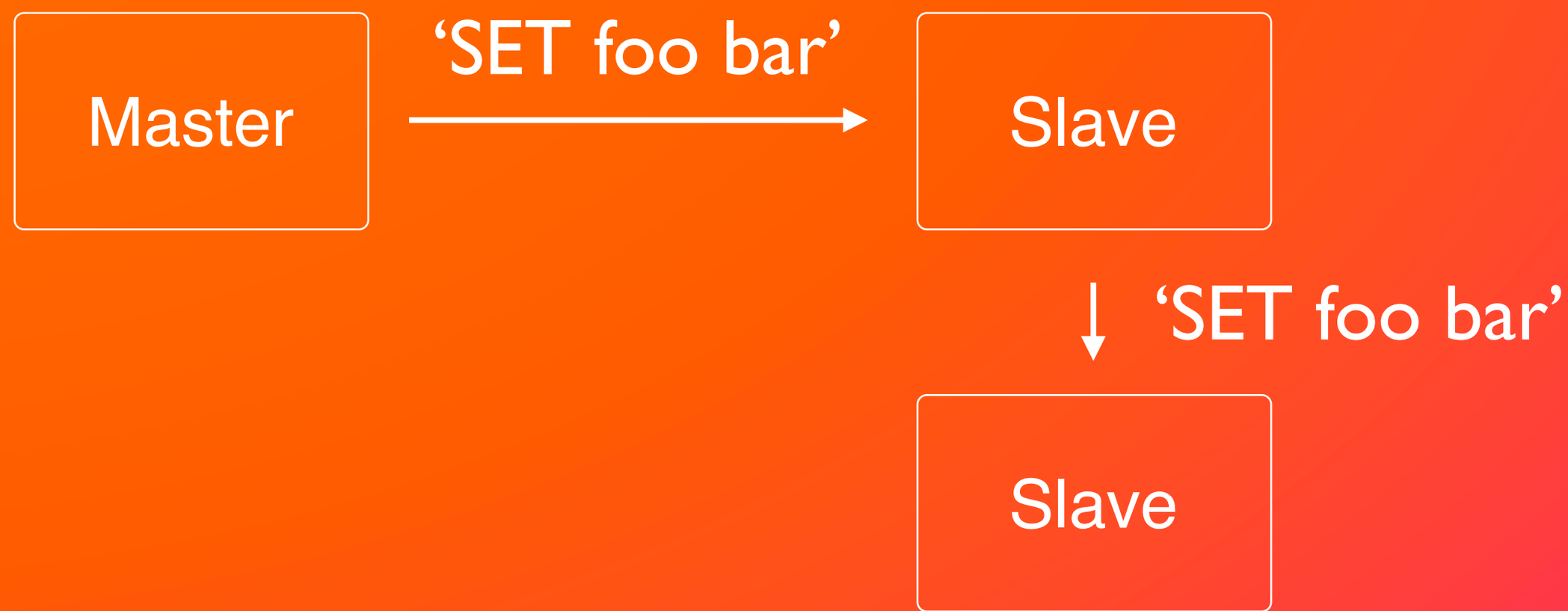
Master sends database file to slave

Slave saves file as its new .rdb file

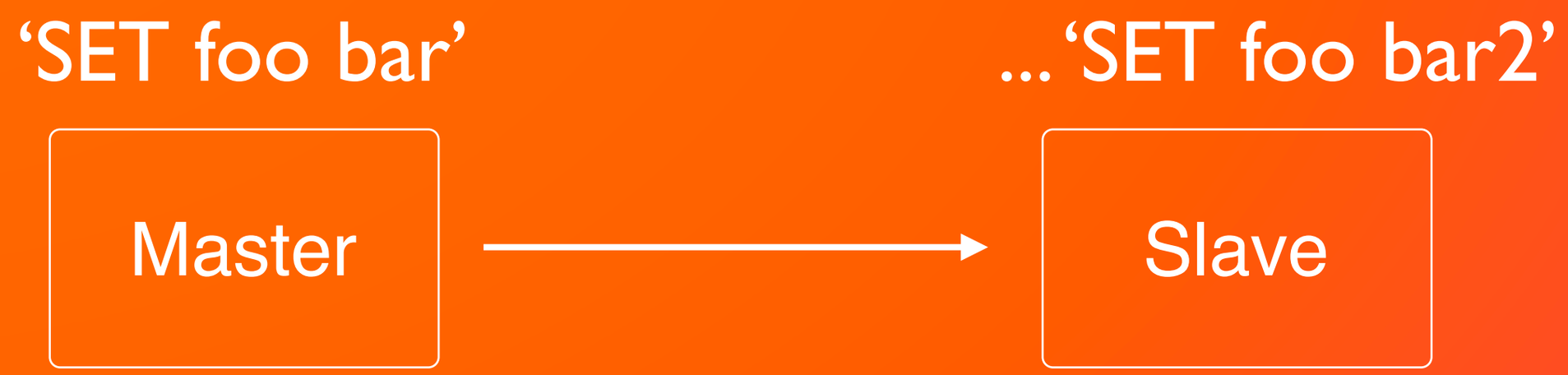
Slave now MONITORs
all commands



Masters can have multiple slaves

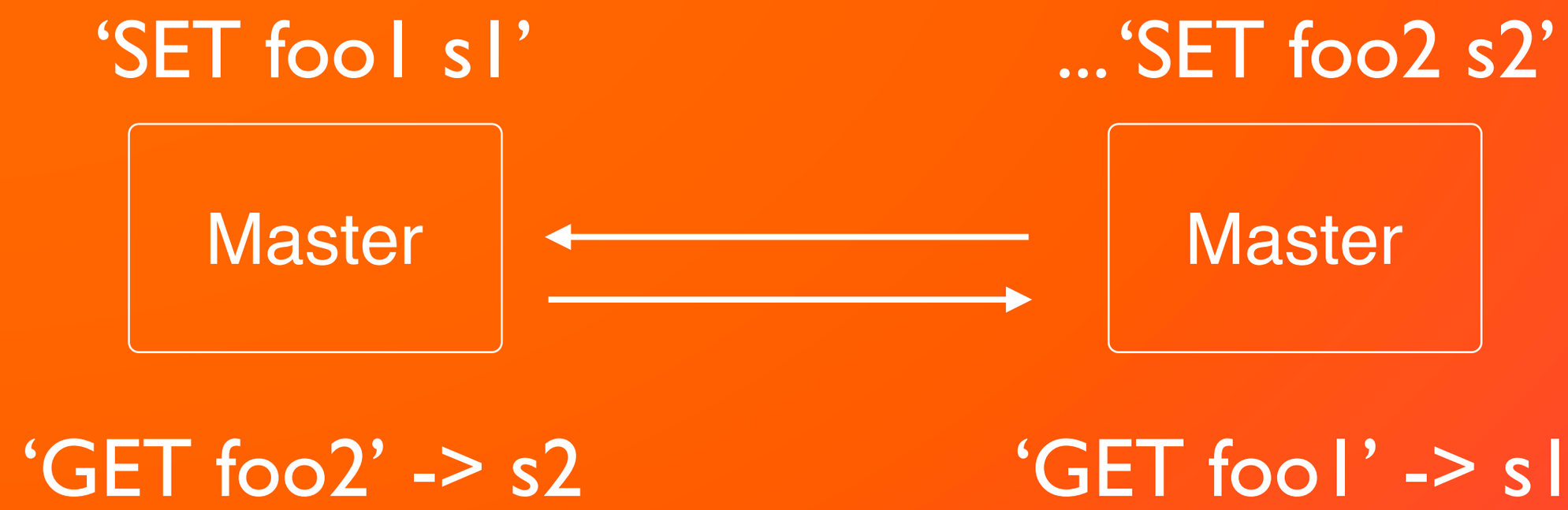


Slaves can have slaves

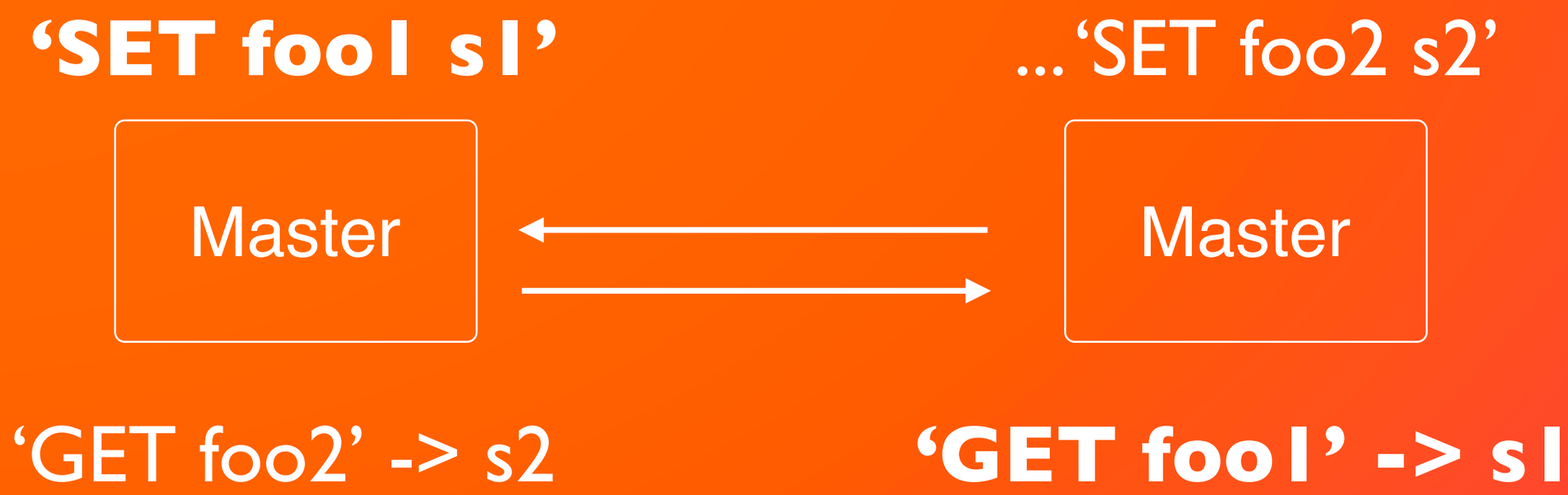


Slaves can accept writes

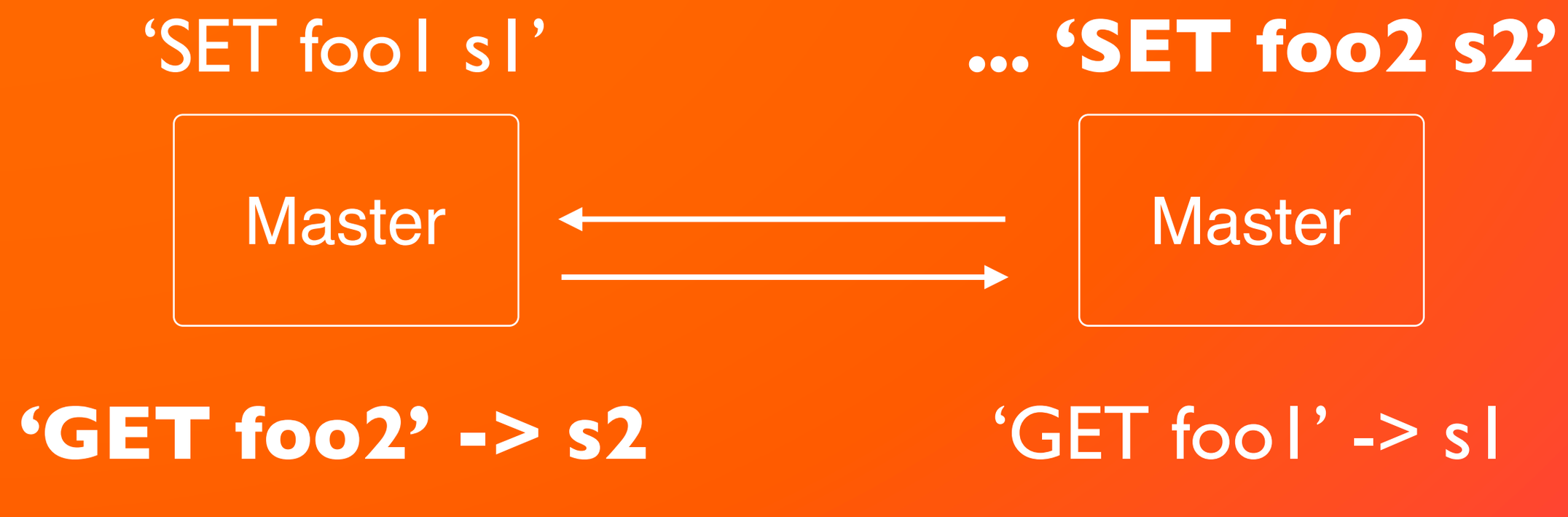
(this can be disabled)



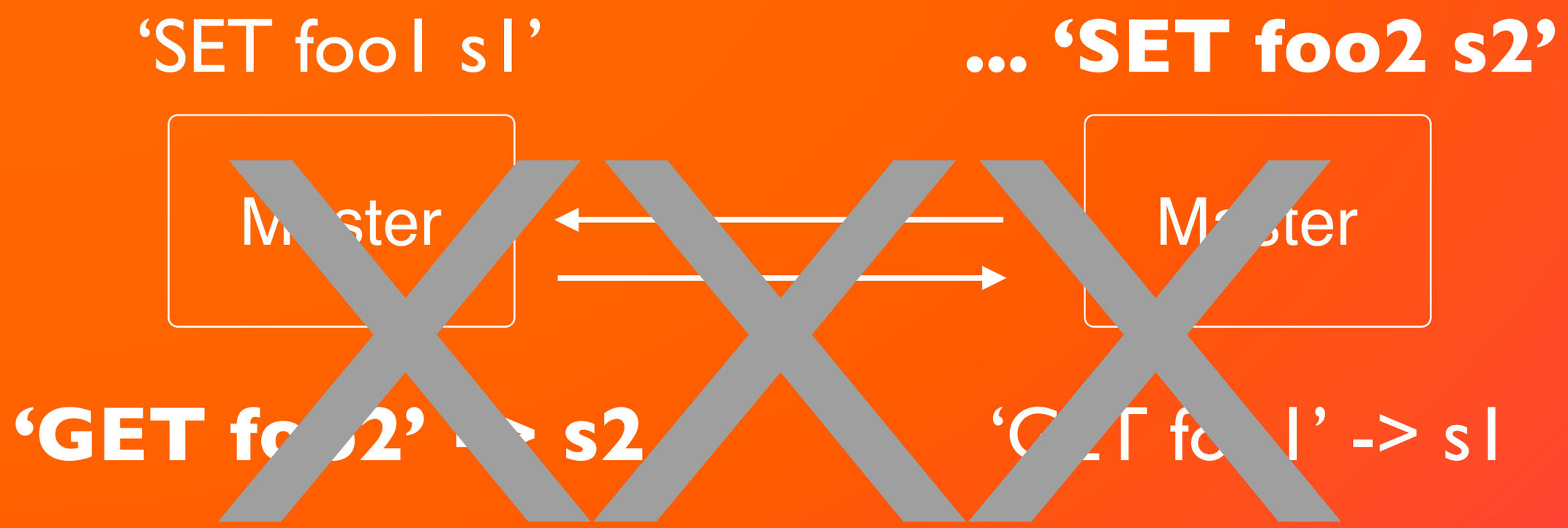
No multi-master



No multi-master



No multi-master



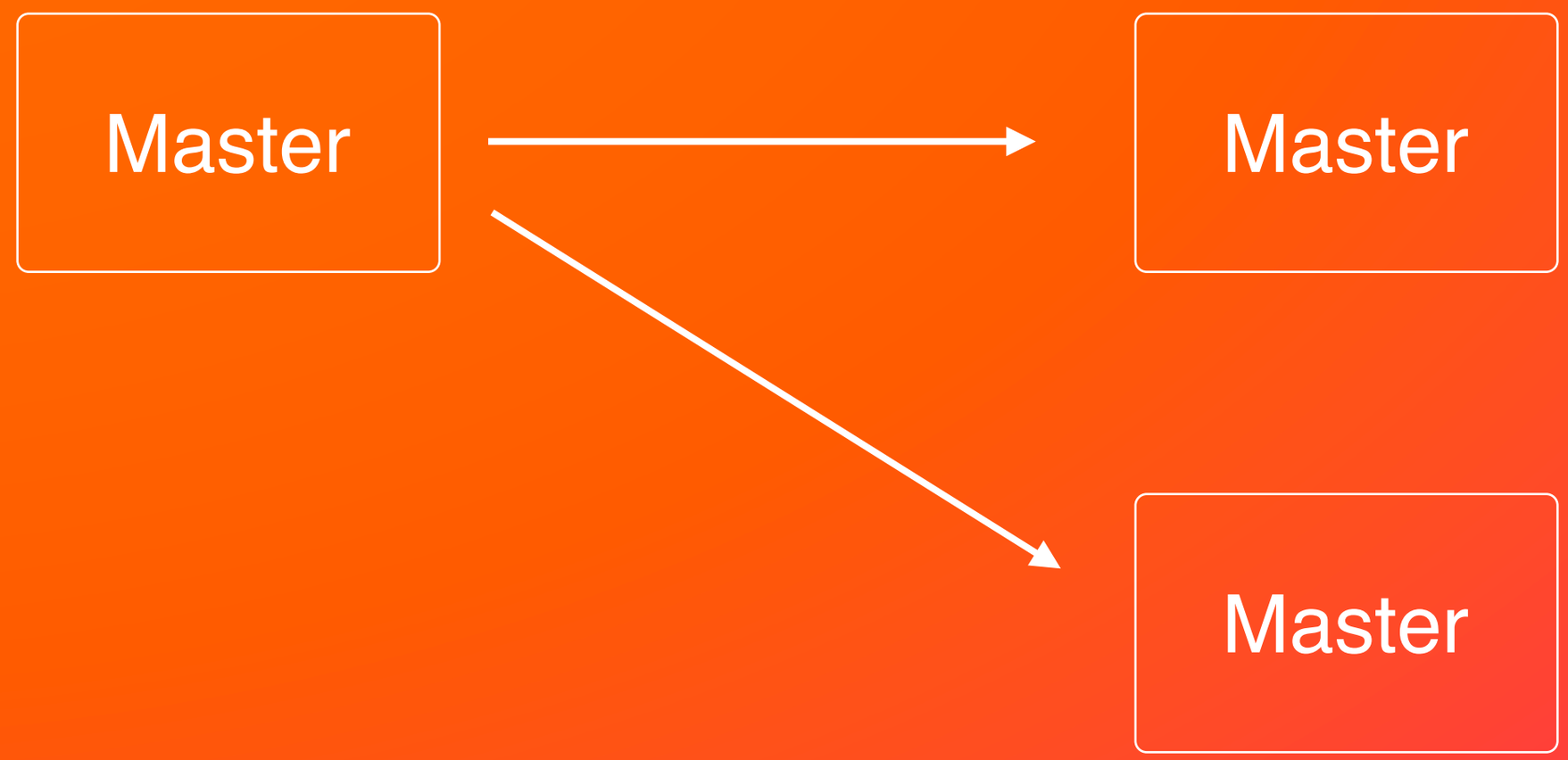
No multi-master in Redis (yet)

So how does Edis Multi-master work?

Uses gen_server:abcast/2

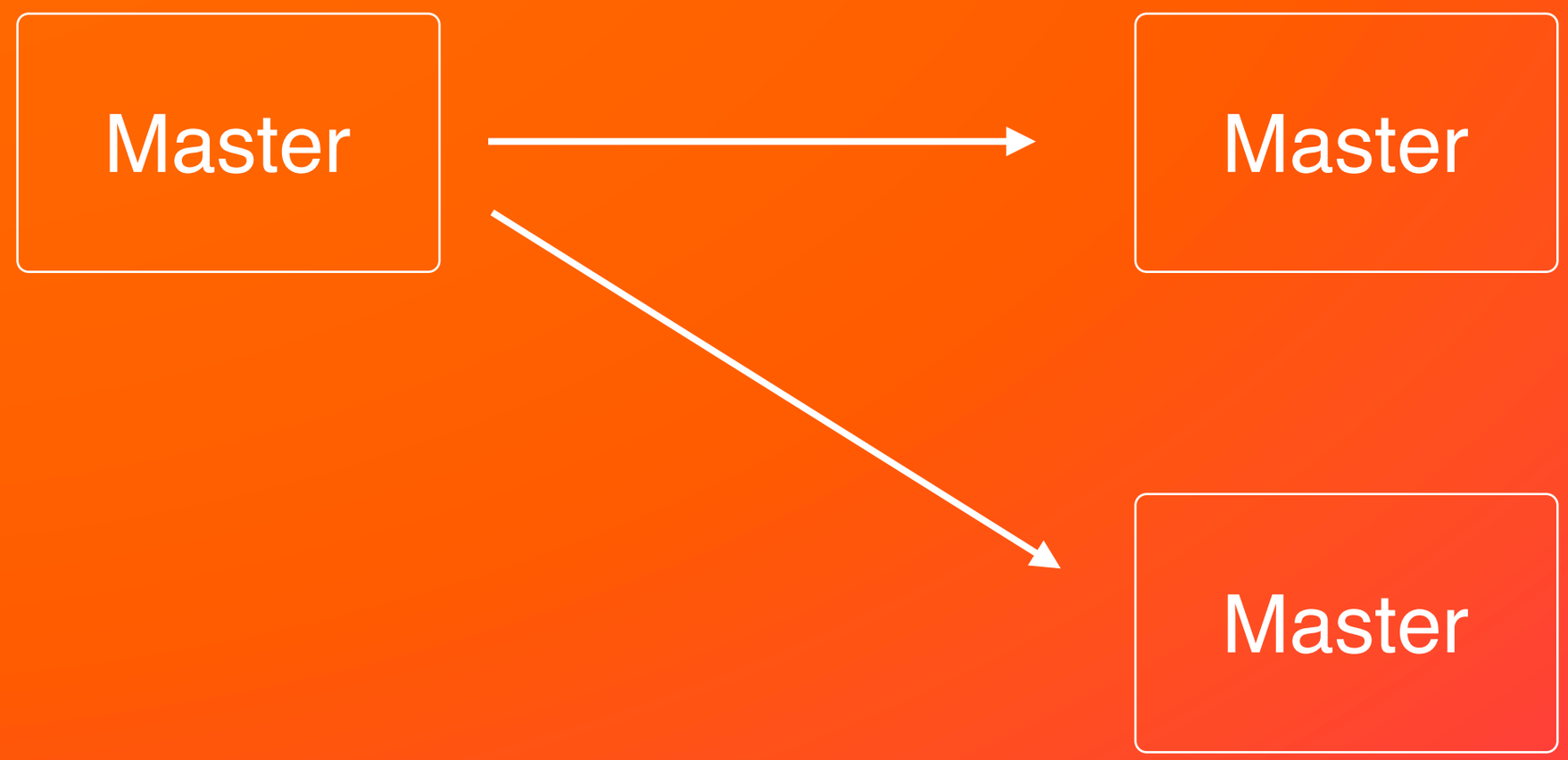
Broadcasts to all nodes

‘SET foo l s l’

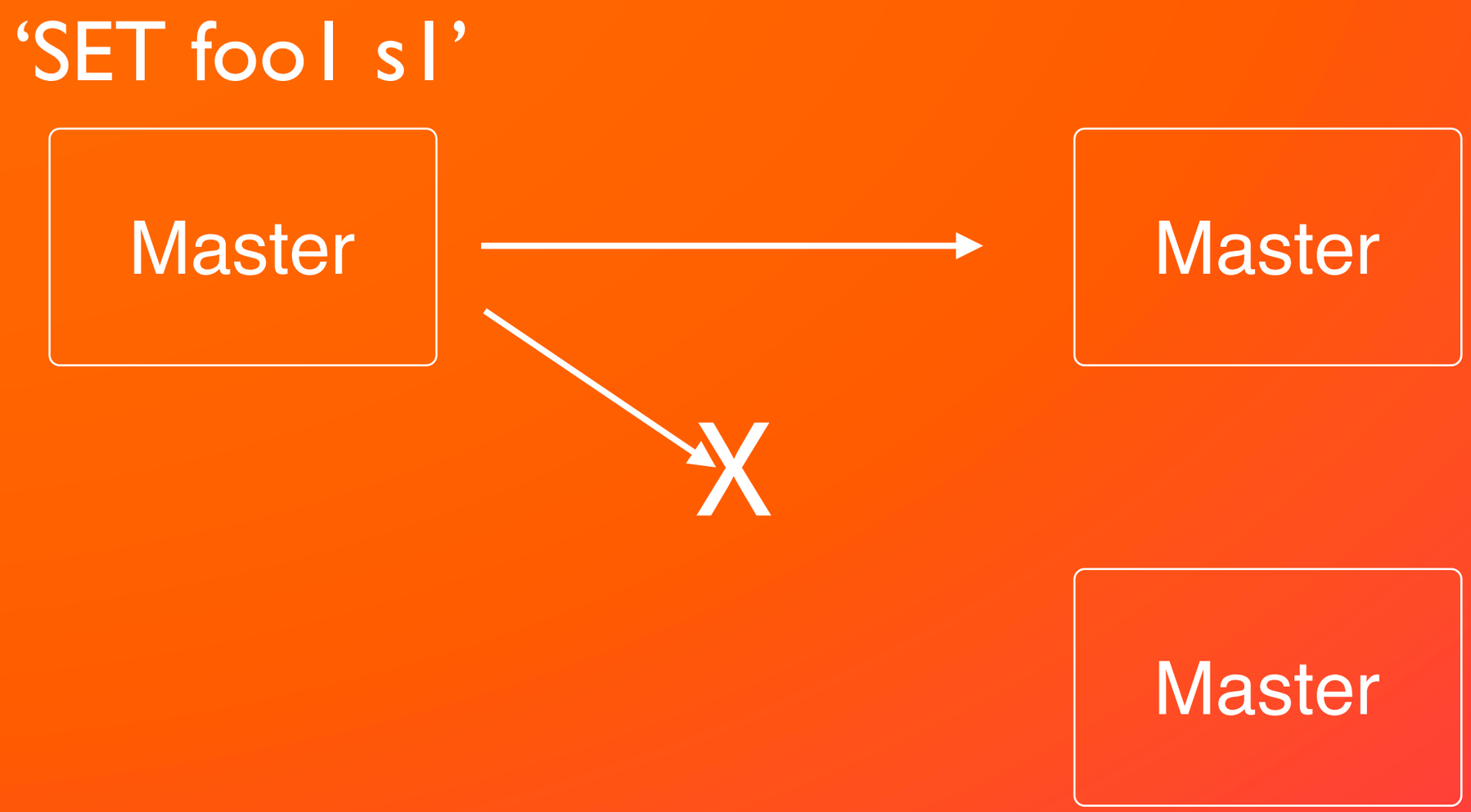


Broadcasts to all nodes

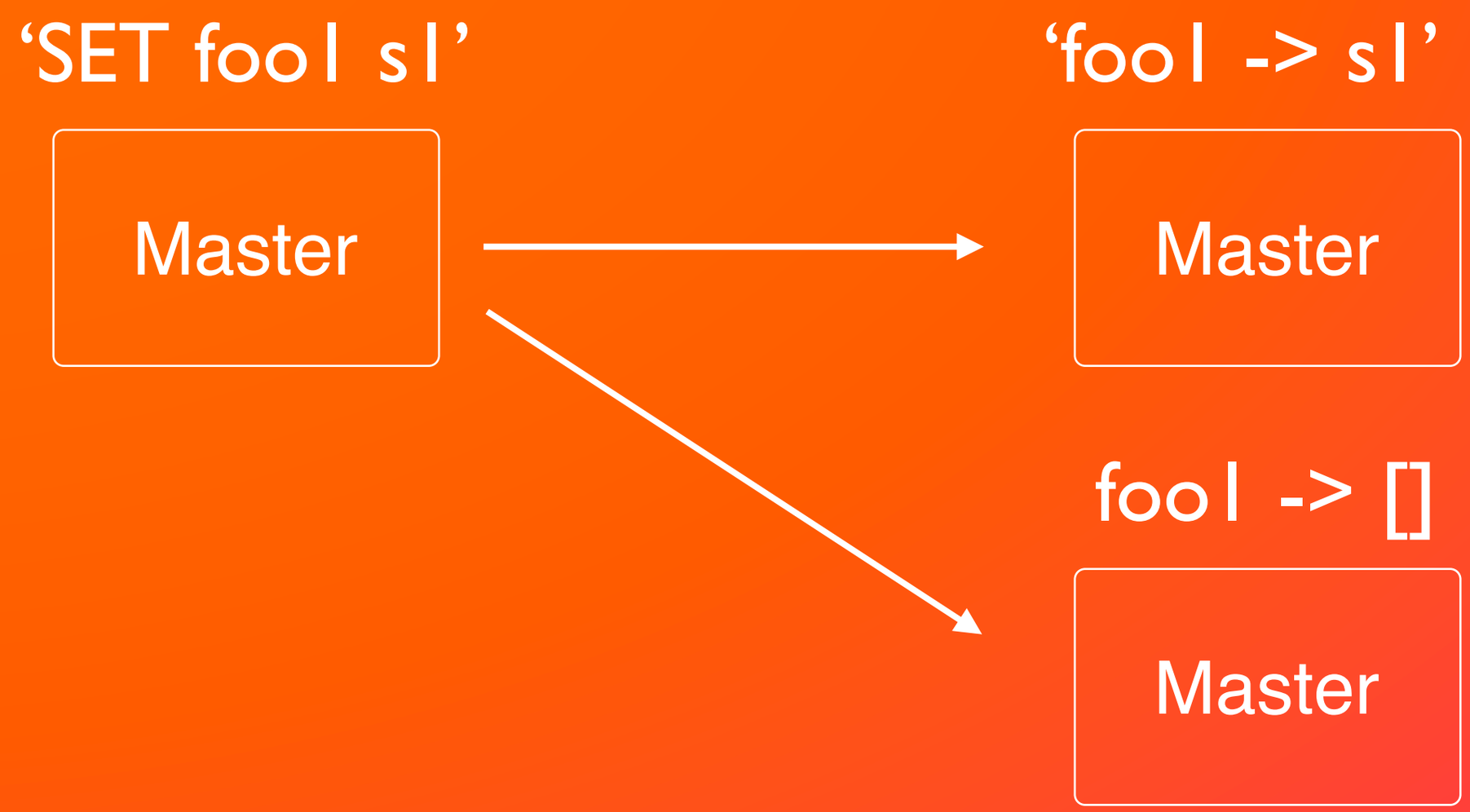
‘SET foo1 s1’



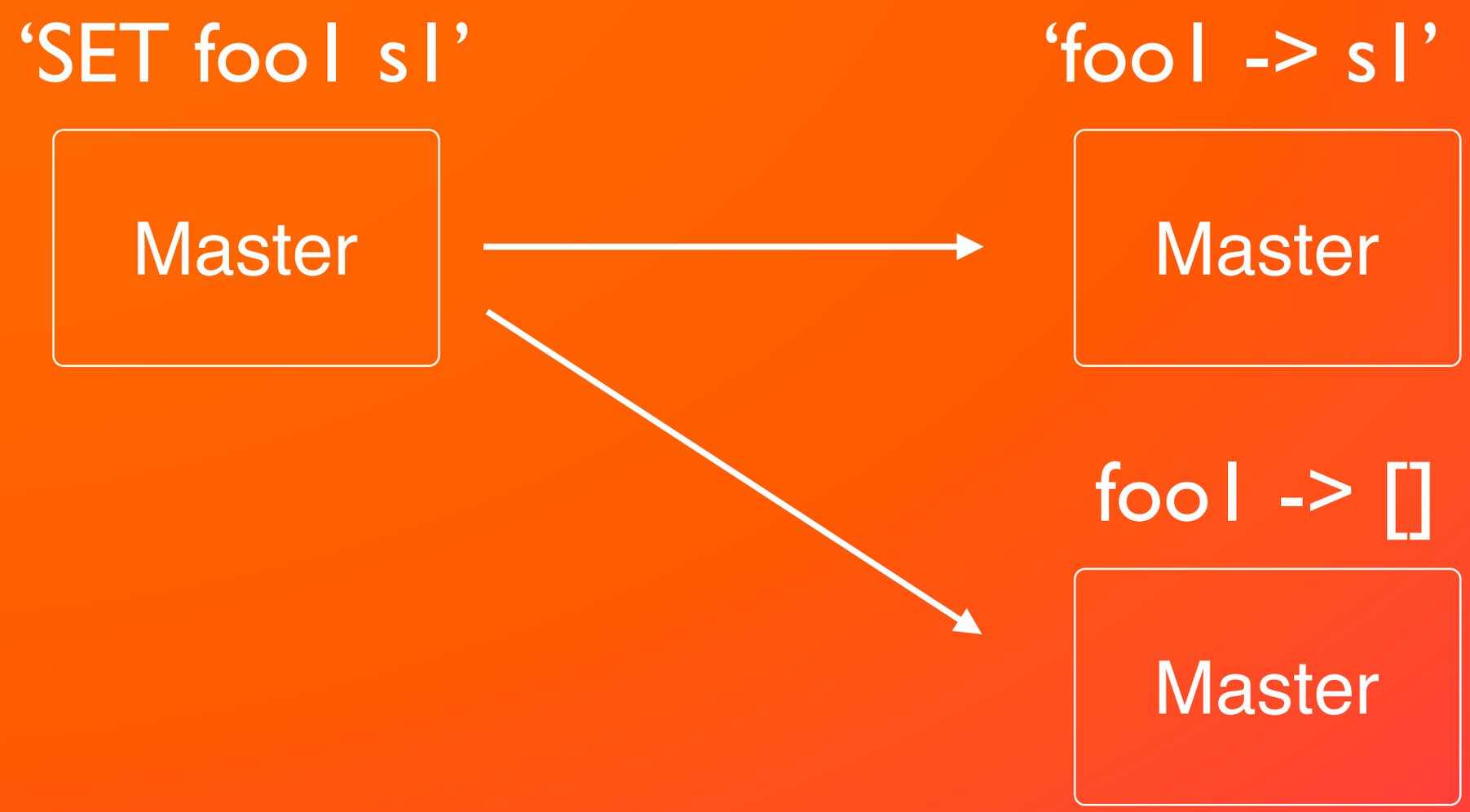
Uses vector clocks to resolve key versions



Does not update disconnected nodes on reconnect



Does not update disconnected nodes on reconnect



However disconnected nodes will detect that keys are out of date since last connection

What is Edis's CAP?

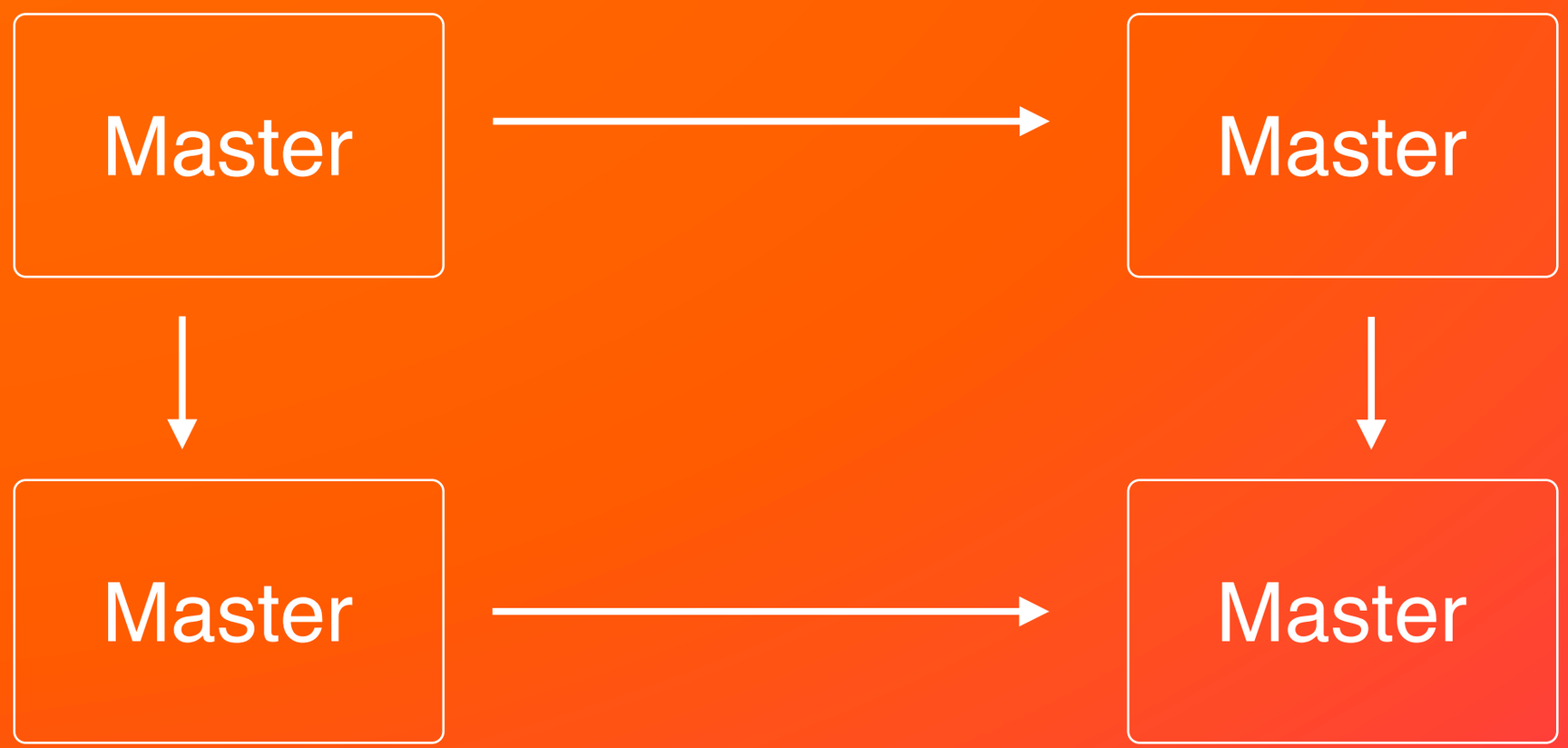
Consistent Available Partition-Tolerant

What is Edis's CAP?

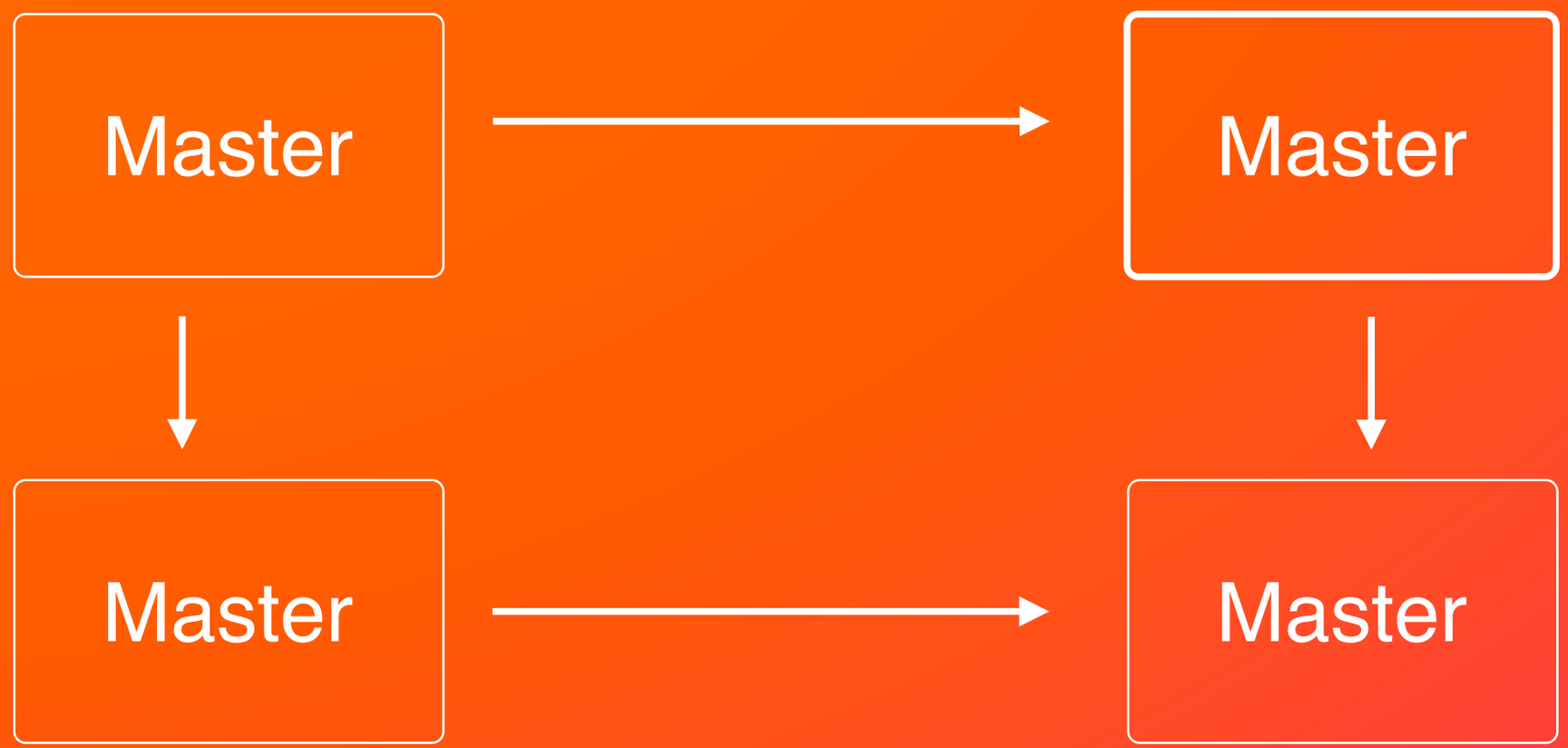
Consistent **Available** **Partition-Tolerant**

What’s next?

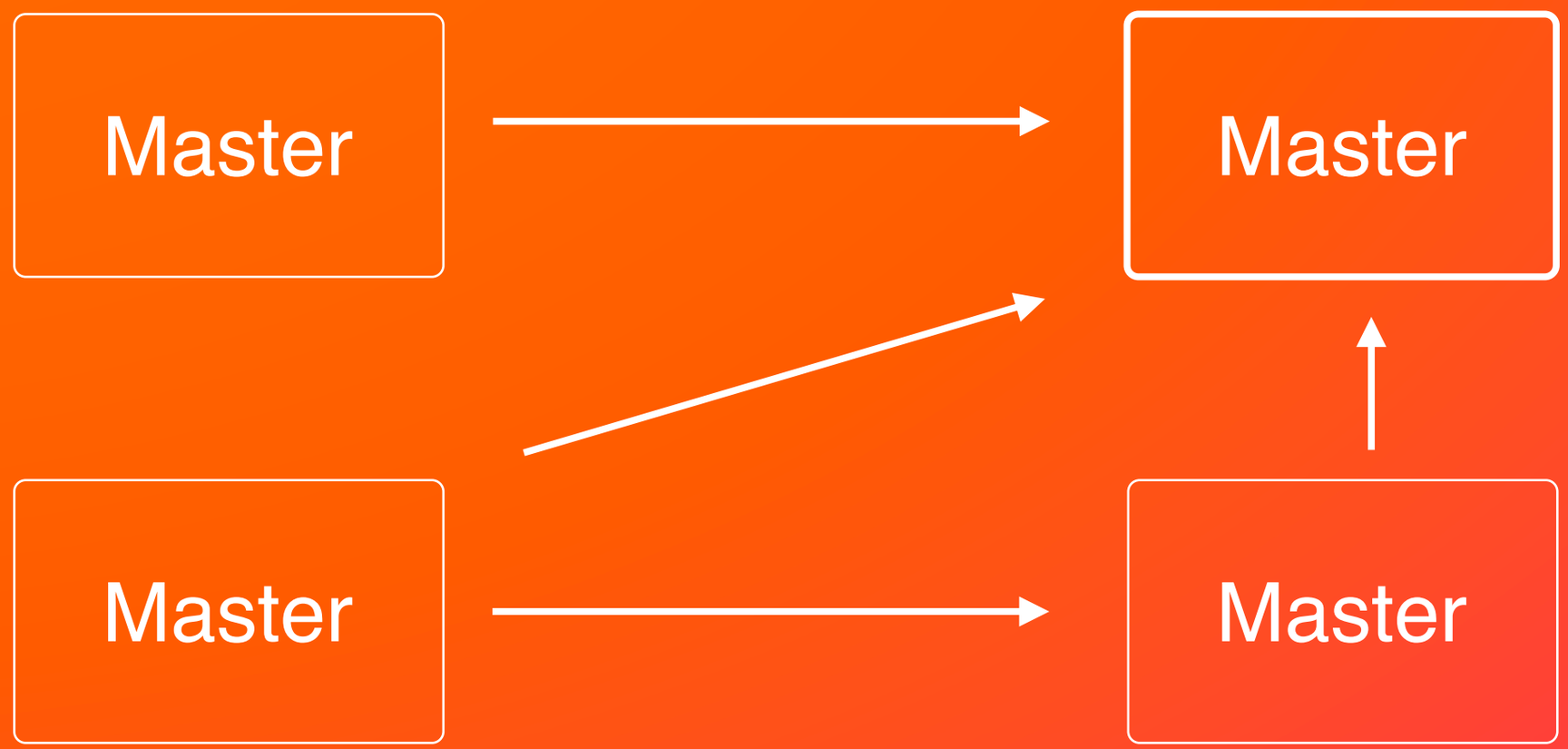
Computation Nodes



Leader Election using gen_leader



Leader Election using gen_leader



now INCR, POP, etc occur on leader only

Replication on reconnect

- Master asks leader for SYNC but disconnects after database received
- New node can either block before replying or background replicate (Currently replication is background)

- Finish Replication for all datatypes
- Post commit hooks
- Redis cluster implementation

How can you help?

- Look at the Edis source
- Lots of Benchmarks w/ common test
- Experiment with it

https://github.com/inaka/edis

Thanks to Joachim Nilsson
for his Uppsala University thesis work
applied to Edis while at Inaka

Thank you / Questions?

http://inaka.net
@chaddepue
chad@inaka.net || chad@whisper.sh

inaka

Other slides

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%