Erlang Solutions Ltd.

Cowboy in Practice

Richárd Jónás & Csaba Hoch

Agenda

- History
- Cowboy and Ranch architecture
- Features by example
  - HTTP handlers
  - REST handlers, practical considerations
  - WebSockets
  - middleware
- Hands on session
- Q&A

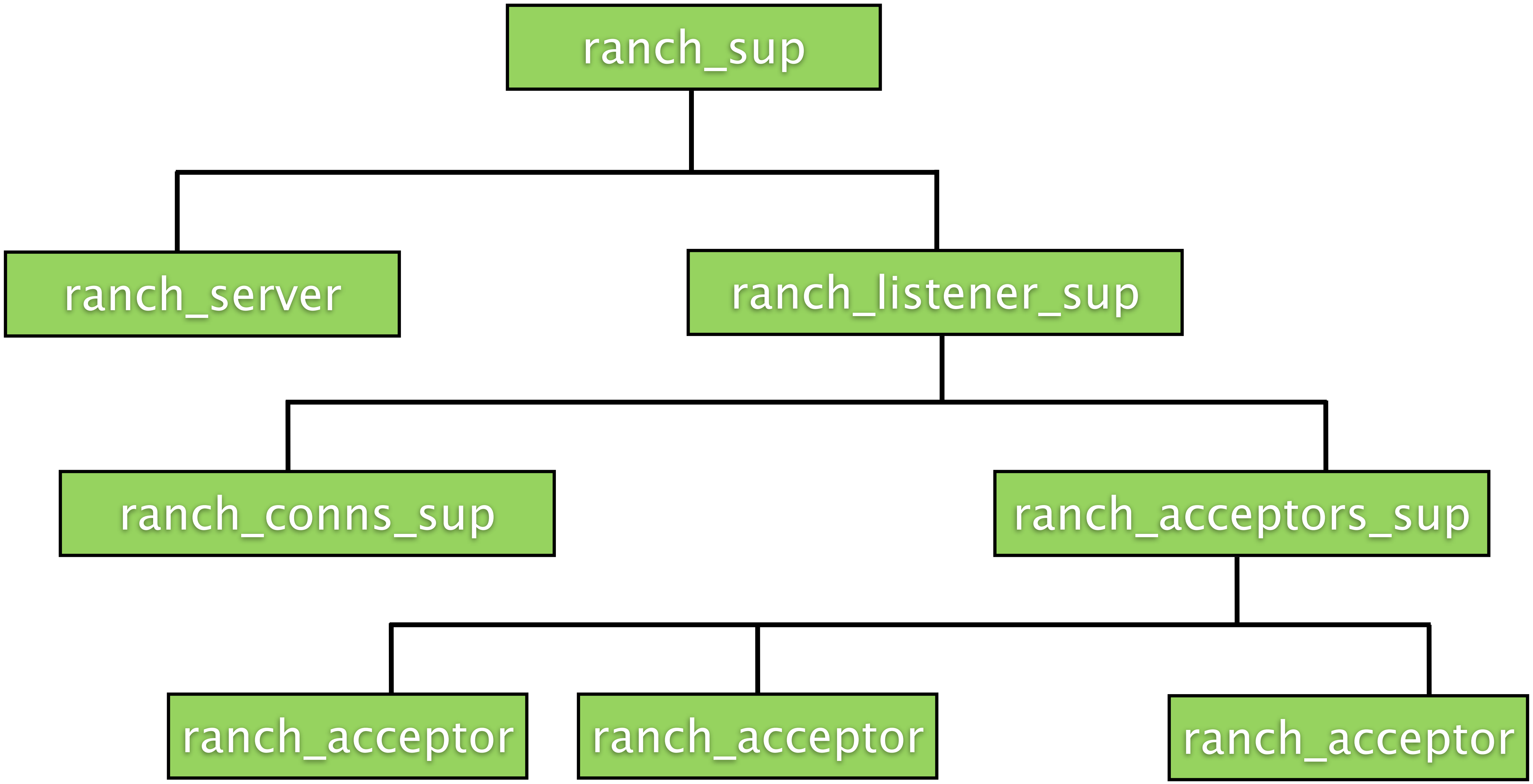
Context

- mochiweb – lightweight embeddable HTTP server by Bob Ippolito
- misultin – same with websockets, practically is dead
- ranch – tcp acceptor pool
- cowboy – embeddable HTTP server based on ranch (Loïc Hoguin from 99s)
- yaws

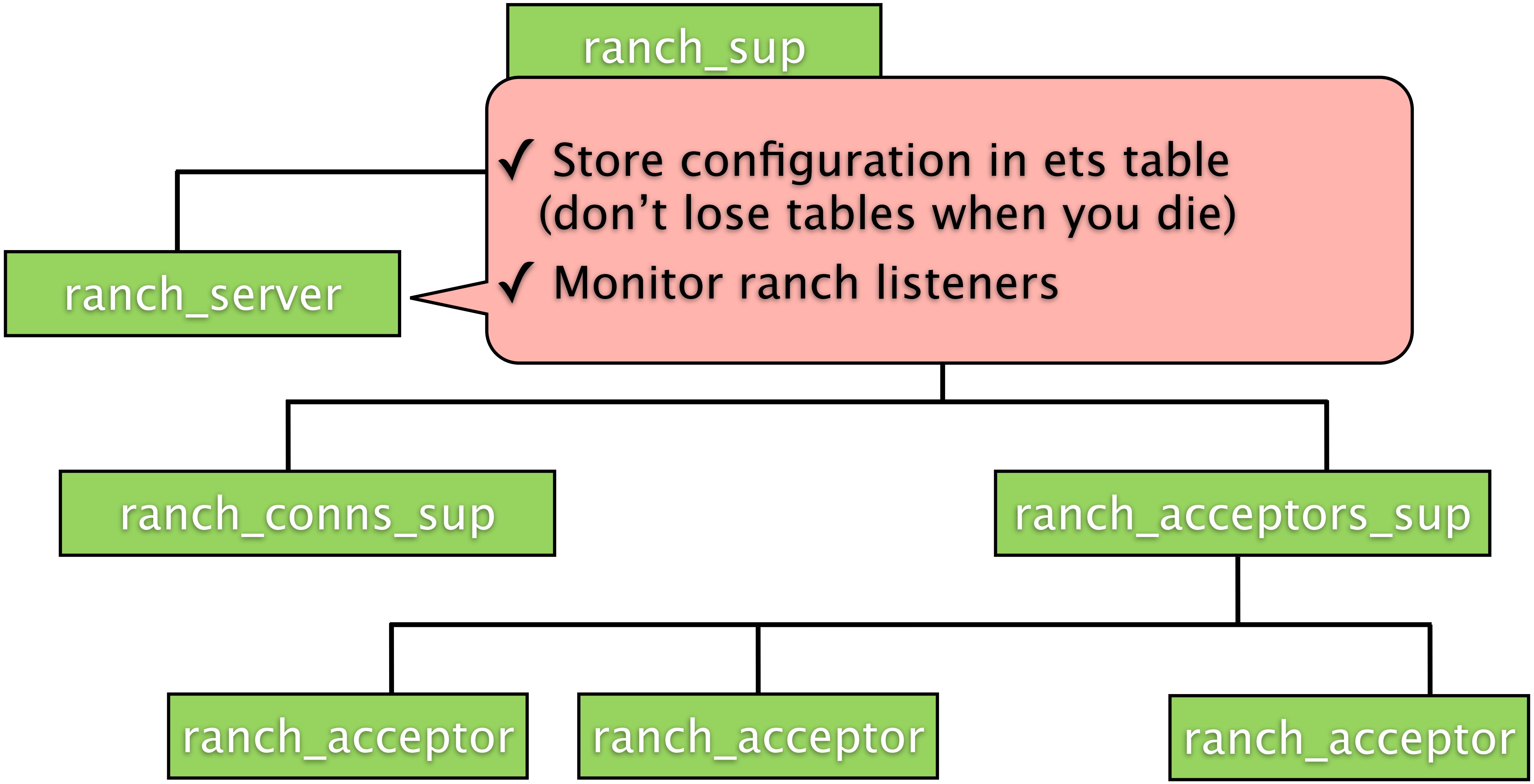
Ranch features

- Callback modules for business logic
- Support tcp and ssl connections
- Support both active and passive connections
- Listen on an existing port (for ssl at least OTP R16 is needed – because of ssl ownership change)
- Listener can be OTP gen_server with a bit of trick

Ranch architecture



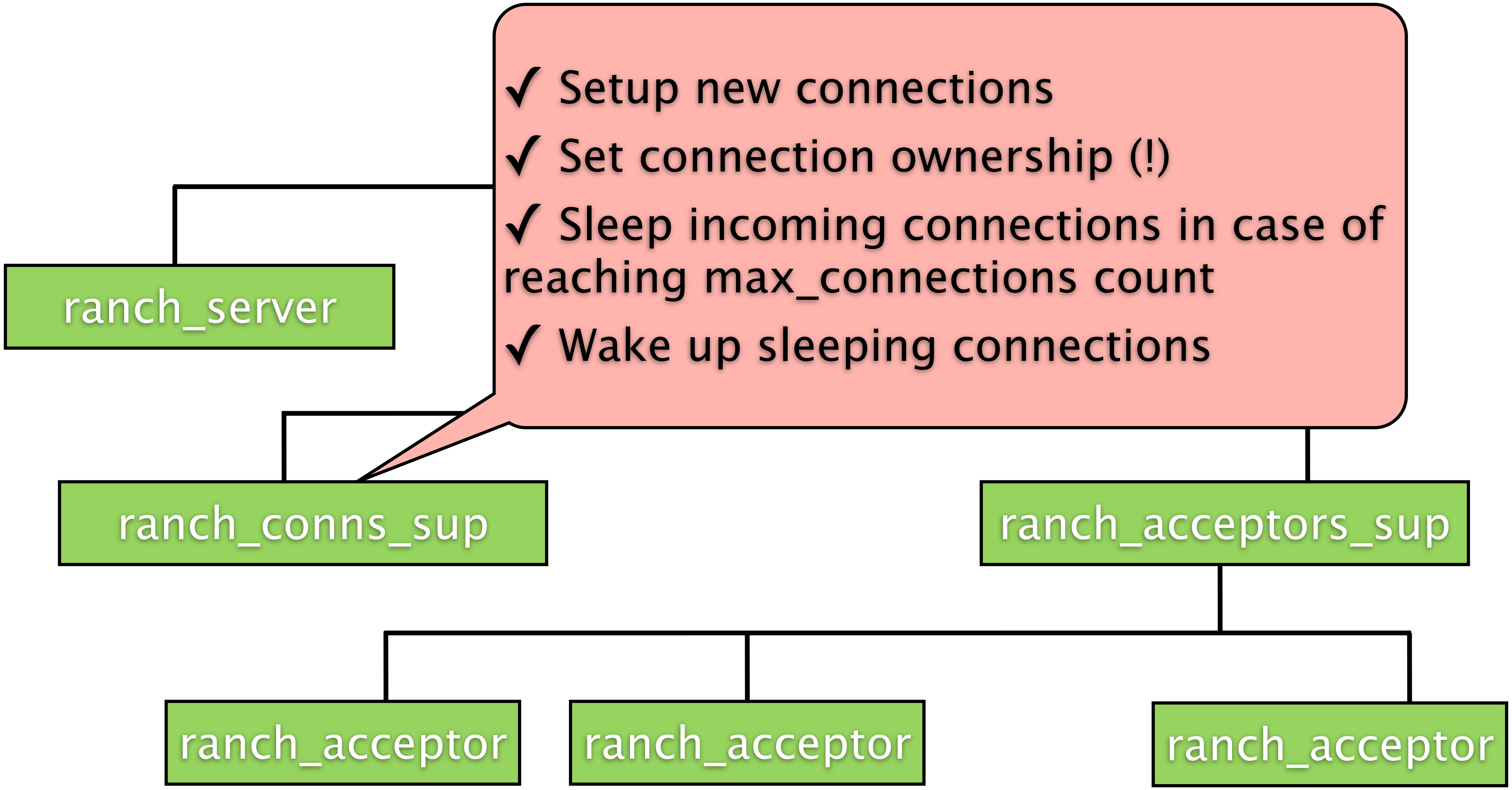
Ranch architecture



✓ Store configuration in ets table (don't lose tables when you die)

✓ Monitor ranch listeners

Ranch architecture



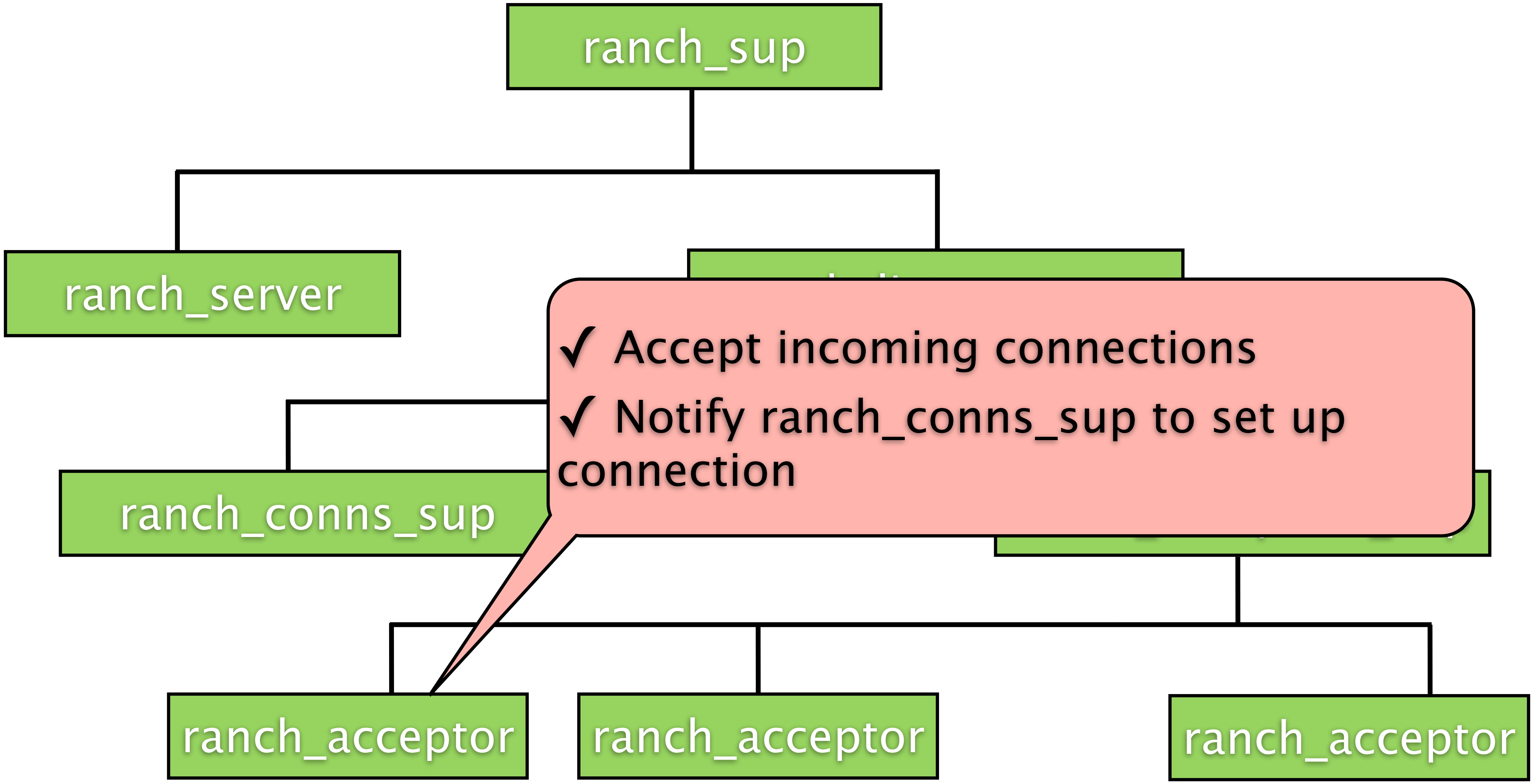
✓ Setup new connections

✓ Set connection ownership (!)

✓ Sleep incoming connections in case of reaching max_connections count

✓ Wake up sleeping connections

Ranch architecture



✓ Accept incoming connections

✓ Notify ranch_conns_sup to set up connection

Echo server (protocol)

```erlang
-module(echo_protocol).
-behaviour(ranch_protocol).

-export([start_link/4, init/4]).

start_link(Ref, Socket, Transport, Opts) ->
    Pid = spawn_link(?MODULE, init, [Ref, Socket, Transport, Opts]),
    {ok, Pid}.

init(Ref, Socket, Transport, _Opts = []) ->
    ok = ranch:accept_ack(Ref),
    loop(Socket, Transport).

loop(Socket, Transport) ->
    case Transport:recv(Socket, 0, 5000) of
        {ok, Request} ->
            Transport:send(Socket, Request),
            loop(Socket, Transport);
        _ ->
            ok = Transport:close(Socket)
    end.
```

Echo server (main)

```
-module(echo_main).

-export([start/0]).

start() ->
    application:start(ranch),
    ranch:start_listener(echo, 10, ranch_tcp, [{port, 6000}],
                         echo_protocol, []).
```

- Start ranch application
- Register `echo` listener on port 6000 with module `echo_protocol`

Examples on github

- Ranch universal time
  - passive socket
  - active once socket
- https://github.com/jonasrichard/cowboy-examples

Cowboy, why?

- Very fast, using mainly binaries
- Small memory footprint
- Hibernate processes (web socket, streaming)
- Compact http requests (save memory)
- Lazy request body fetching
- Extensible (handlers, middlewares)
- Cool (websocket, http 2.0, spdy)
- 2 million connections

Step 1 - Plan who will handle what

```
Dispatch = cowboy_router:compile(
  [{'_', [
    {"/app/[...]", my_http_handler, []},
    {"/[...]", cowboy_static, [
      {directory, {priv_dir, my_app, []}},
      {mimetypes, {fun mimetypes:path_to_mimes/2, default}}
    ]}
]}]),

cowboy:start_http(my_http, 10, [{port, 8080}],
                  [{env, [{dispatch, Dispatch}]}])
```

Mimetypes application is required to be started!

More about routing

Host match rule	
`“www.erlang.org”`	exact match
`“.erlang.org”`	same as above
`“:subdomain.erlang.org”`	catch binding for subdomain
`“erlang.:_”`	erlang.com, .org, .eu
`“[www.]erlang.org”`	www is optional
`‘_’`	any host

More about routing

Path match rule	
`“/main/controller”`	exact match
`“/[...]”`	matches anything
`“/images/[...]”`	all path inside images
`“/nodes/:nodeId”`	catch path segment
`“/nodes/[:nodeId]”`	optional path segment
`‘_’`	any path

Step 2 - Write handlers (http_handler)

```erlang
-module(my_http_handler).
-behaviour(cowboy_http_handler).
-export([init/3, handle/2, terminate/3]).

init(_Type, Req, _Opts) ->
  {ok, Req, undefined}.

handle(Req, State) ->
  {ok, Req2} = cowboy_req:reply(200,
    [{<<"content-type">>, <<"text/plain">>}],
    <<"Hello, Erlang">>,
    Req),
  {ok, Req2, State}.

terminate(_Reason, _Req, _State) ->
  ok.
```

Step 2 - Write websocket handler

```erlang
-module(my_ws_handler).
-behaviour(cowboy_websocket_handler).
-export([init/3]).
-export([websocket_init/3, websocket_handle/3,
         websocket_info/3, websocket_terminate/3]).

%% Comes from cowboy_http_handler behaviour
%% The other two callbacks are not needed
init({tcp, http}, _Req, _Opts) ->
  {upgrade, protocol, cowboy_websocket}.

websocket_init(_TransportName, Req, _Opts) ->
  {ok, Req, undefined_state}.
```

Step 2 - Write websocket handler

```
websocket_handle({text, <<"create ", Rst/binary}, Req, State) ->
  {reply, {text, create_user(Rst)}, Req, State};
websocket_handle({text, <<"exit">>}, Req, State) ->
  {shutdown, Req, State};
websocket_handle(_Data, Req, State) ->
  {ok, Req, State}.

websocket_info({logged_in, User}, Req, State) ->
  {reply,
    {text, <<User/binary, <<" logged in">>/binary >>},
    Req, State};
websocket_info(_Info, Req, State) -> {ok, Req, State}.

websocket_terminate(_Reason, _Req, _State) -> ok.
```

REST handlers

- Set allowed methods (GET, HEAD, OPTIONS)
  - options/2 for handling OPTIONS request
- Set provided content type (text/html)
- Set accepted content type (none)
  - register hooks for each content type
- Implement cowboy_http_handler behaviour only
- There is no behaviour for REST, too much optional functions

Step 3 - REST handler

```
-module(my_rest_handler).
-behaviour(cowboy_http_handler).

init(_Transport, _Req, _Opts) ->
  {upgrade, protocol, cowboy_rest}.

content_types_provided(Req, State) ->
  Handlers = [
    {<<"application/json">>, handle_json},
    {<<"text/html">>, handle_html}
  ],
  {Handlers, Req, State}.

content_types_accepted(Req, State) ->
  Types = [
    {{<<"application">>, <<"json">>, '*'}, process_post}],
  {Types, Req, State}.
```

Step 3 - REST handler

```
allowed_methods(Req, State) ->
  M = [<<"GET">>, <<"POST">>, <<"PUT">>, <<"OPTIONS">>],
  {M, Req, State}.

rest_init(Req, _Opts) ->
  {ok, Req, undefined_state}.

rest_terminate(Req, State) -> ok.

options(Req, Opts) ->
  case cowboy_req:header(<<"origin">>, Req) of
    {undefined, _Req} ->
      {halt, Req, State};
    {<<"http://mysite.com">> = Orig, _Req} ->
      Req2 = cowboy_req:set_resp_header(
        <<"access-control-allow-origin">>, Orig),
      {ok, Req2, Opts}
  end.
```

Step 3 - REST handler

```
handle_json(Req, State) ->
  {PathInfo, _} = cowboy_req:path_info(Req),
  case PathInfo of
    [<<"people">>] ->
      {to_json(all_people()), Req, State};
    [<<"people">>, BId] ->
      case get_person(binary_to_integer(BId)) of
        {ok, Person} ->
          {to_json(Person), Req, State};
        {error, _Reason} ->
          {ok, Req2} = cowboy_req:reply(404, Req),
          {halt, Req2, State}
     end
    _ ->
      {ok, Req3} = cowboy_req:reply(404, Req),
      {halt, Req3, State}

end.
```

Request body with `cowboy_req`

`has_body/1`	boolean()
`body_length/1`	Size of the body (compressed size)
`body/1`	< 8MB
`body_qs/1`	< 16KB
`init_stream, stream_body`	For huge request bodies
`multipart_data/1` `multipart_skip/1`	Provide part or eof

Cowboy middleware

- In default there are two middlewares
  - cowboy_router - preparing handler by request path
  - cowboy_handler - controlling lifecycle of a handler
- We can extend the default middleware chain by implementing cowboy_middleware behaviour

```
execute(Req, Env) ->
   {ok, Req, Env},
 | {suspend, module(), atom(), [any()]},
 | {halt, Req},
 | {error, cowboy:http_status(), Req}.
```

Thank you

- Sources:
  https://github.com/jonasrichard/cowboy-examples
- Contact
  - Richárd Jónás
  - richard.jonas.76@gmail.com
  - mail@jonasrichard.hu