Erlang Solutions Ltd.

A True Conversational Web

Robert Virding

Conversational Web

- Two way communication between client and server
- Maintain server state between calls
- Interactive

<u>This is not RESTful!</u>

But desirable!

Fakin' It (not really makin' it)

- Streaming
  - HTTP server push
  - XMLHttpRequest
  - Pushlet
- Long polling

Makin' It

- Websockets
  - Full bi-directional channel
  - TCP socket
- Socket.IO
  - Looks like a websocket
  - Uses feature detection to decide how connection is to be made

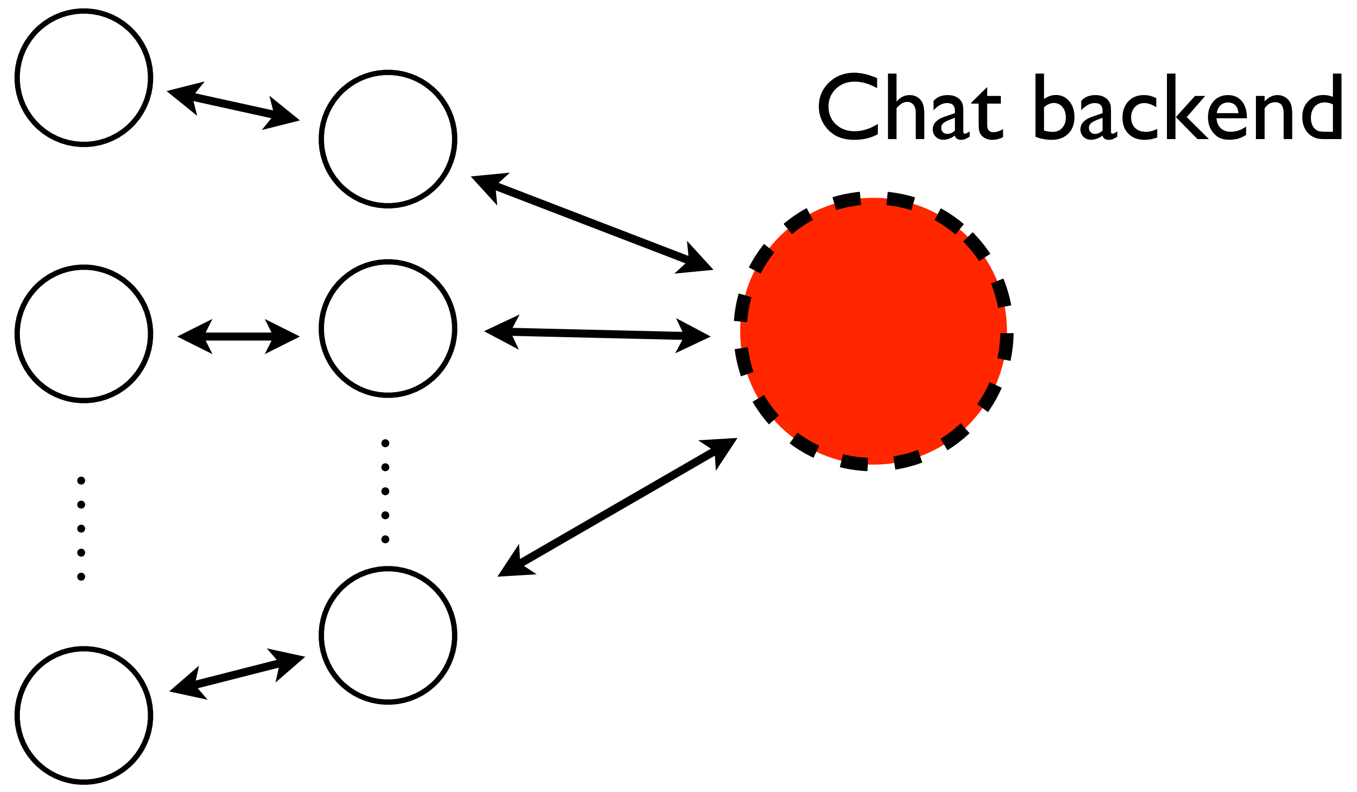
Erlang Concurrency

- Lightweight concurrency
  - The system should be able to handle a large number of processes
  - Process creation, context switching and inter-process communication must be cheap and fast.
- Asynchronous communication
- Process isolation
  - We don't want what is happening in one process to affect any other process.
- Error handling
  - The system must be able to detect and handle errors.

Erlang Concurrency

- Processes are used for many things
  - Concurrency
  - Managing state
- Pattern matching is ubiquitous

Simple chat



- Running at: http://10.0.15.66:8080/chat

TCP client

```
client(Host, Port, Message) ->
    {ok,Socket} = gen_tcp:connect(Host, Port, [{packet,0},{active,false}]),
    gen_tcp:send(Socket, Message),
    Reply = gen_tcp:recv(Socket, 0),
    gen_tcp:close(Socket),
    Reply.
```

- A client process is responsible for starting a connection towards the server
- It sends a package and closes the socket
  - It could have sent more packages
  - It could have waited for the other side to close the connection

TCP server

```
start(Port) ->
    {ok,ListenSocket} =
      gen_tcp:listen(Port, [binary,{packet,0},{active,true}]),
    wait_connect(ListenSocket).

wait_connect(ListenSocket) ->
    {ok,Socket} = gen_tcp:accept(ListenSocket),
     spawn(fun () -> wait_connect(ListenSocket) end),
     request_loop(Socket).

request_loop(Socket) ->
    receive
        {tcp,Socket,Binary} ->
            <do_stuff>
            request_loop(Socket);
        {request,From,Req} ->
            <do other stuff>
            request_loop(Socket)
    end.
```

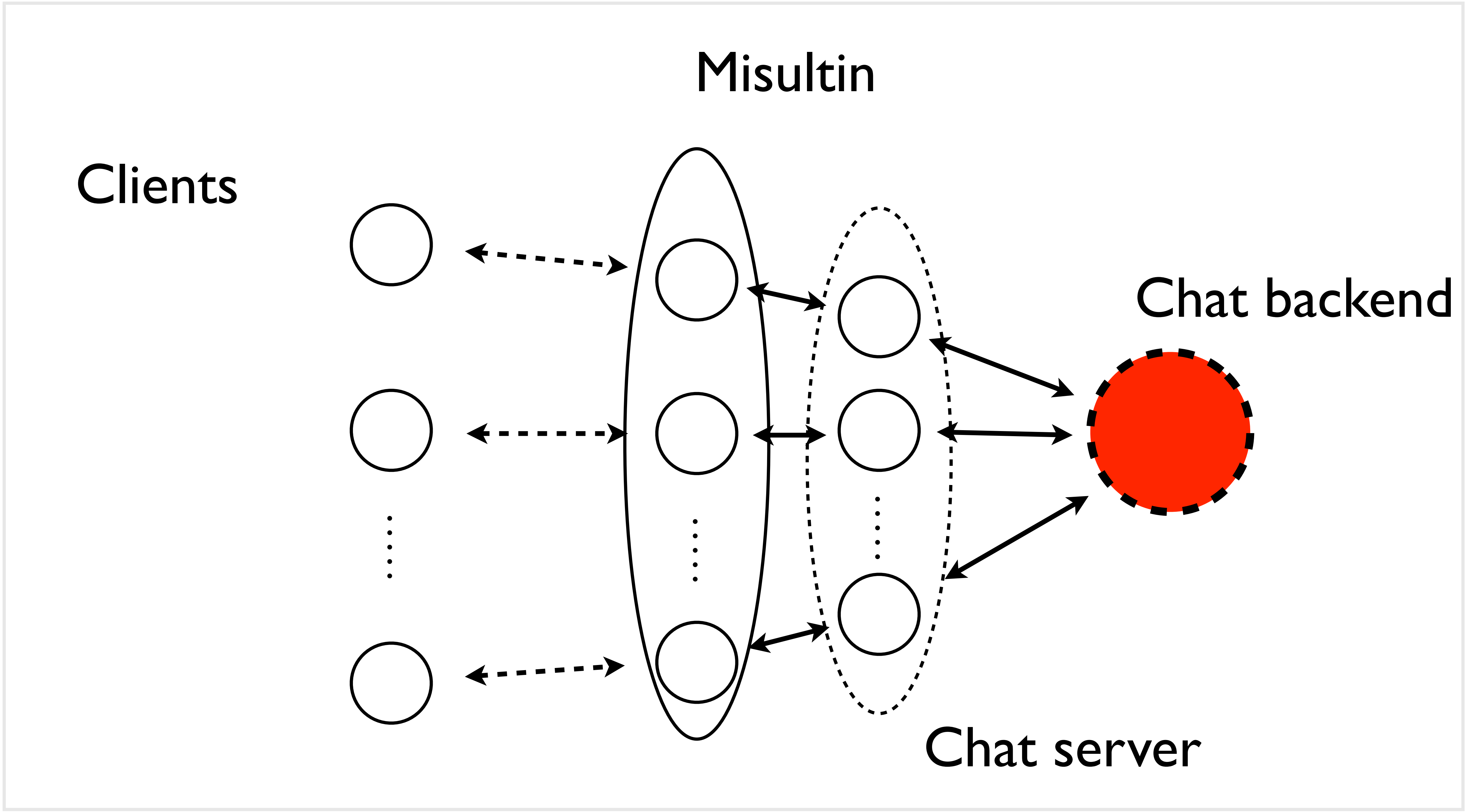
- The server will spawn a new listener for every request

TCP server

```
start(Port, Count) ->
    {ok,ListenSocket} =
      gen_tcp:listen(Port, [binary,{packet,0},{active,true}]),
    start_servers(Count, ListenSocket).

start_servers(0, _) -> ok;
start_servers(Num, ListenSocket) ->
    spawn(fun () -> wait_connect(ListenSocket) end),
    start_servers(Num-1, ListenSocket).

wait_connect(ListenSocket) ->
    {ok,Socket} = gen_tcp:accept(ListenSocket),
     request_loop(Socket).

request_loop(Socket) ->
    receive
        {tcp,Socket,Binary} ->
            <do_stuff>,
            get_request(Socket);
        {request,From,Req} -> ...
```

- The server will only spawn Count listeners

Conversational simple chat



Chat server process

```
ws_loop(Ws, ChatBackend) ->
    receive
        {browser, Data} ->
            case Data of
                "msg ! " ++ Msg ->
                    ChatBackend ! {self(),{message,Msg}}
                "nick ! " ++ Nick ->
                    ChatBackend ! {self(),{set_nick,Nick}}
                _ -> Ws:send(["status ! received '", Data, "'"])
            end,
            ws_loop(Ws, ChatBackend);
        closed -> closed;
        {chat_server,{message,Msg}} ->
            Ws:send(["output ! ",Msg]),
            ws_loop(Ws, ChatBackend);
        _Ignore -> ws_loop(Ws, ChatBackend)
    after 10000 ->
        Ws:send("clock ! tick " ++ io_lib:fwrite("~p", [time()])),
        ws_loop(Ws, ChatBackend)
    end.
```

- Browser message, send correctly to backend.
- Server message, send to output window
- Push a "tick" to show we are alive

HTTP Request process

```
handle_http(Req, Port) ->
    %% output
    case {Req:get(method),Req:resource([lowercase,urldecode])} of
        {'GET',["chat"]} ->                     %Our chat program
            {ok,File} = file:read_file("./chat.html"),
            [Bef,Aft] = binary:split(File, <<"%%HOST:PORT%%">>),
            Host = proplists:get_value('Host', Req:get(headers)),
            Req:ok([Bef,Host,Aft]);
        {'GET',[File]} ->
            case filelib:is_regular(File) of
                true ->
                    Req:file(File);
                false ->
                    Req:respond(404, [], ["no file: ",File])
            end;
        _ ->
            io:fwrite("hh: ignoring\n")
    end.
```

- Chat request, patch in host and port
- Other requests, try and get file

Chat server process (LFE)

```
(defun ws-loop (ws chat-backend)
  (receive
    ((tuple 'browser data)
     (case data
       ((++ '"msg ! " msg)
        (! chat-backend (tuple (self) (tuple 'message msg))))
       ((++ '"nick ! " nick)
        (! chat-backend (tuple (self) (tuple 'set_nick nick))))
       (_ (call ws 'send (list '"status ! received'" data '"'"))))
     (ws-loop ws chat-backend))
    ('closed 'closed)
    ((tuple 'chat_server (tuple 'message msg))
     (call ws 'send (list '"output ! " msg))
     (ws-loop ws chat-backend))
    (_ (ws-loop ws chat-backend))
    (after 10000
      (call ws 'send (++ '"clock ! tick "
                         (: io_lib fwrite '"~p" (list (time)))))
      (ws-loop ws chat-backend))))
```

- Browser message, send correctly to backend.

- Server message, send to output window
- Push a "tick" to show we are alive

Thank you

Robert Virding robert.virding@erlang-solutions.com