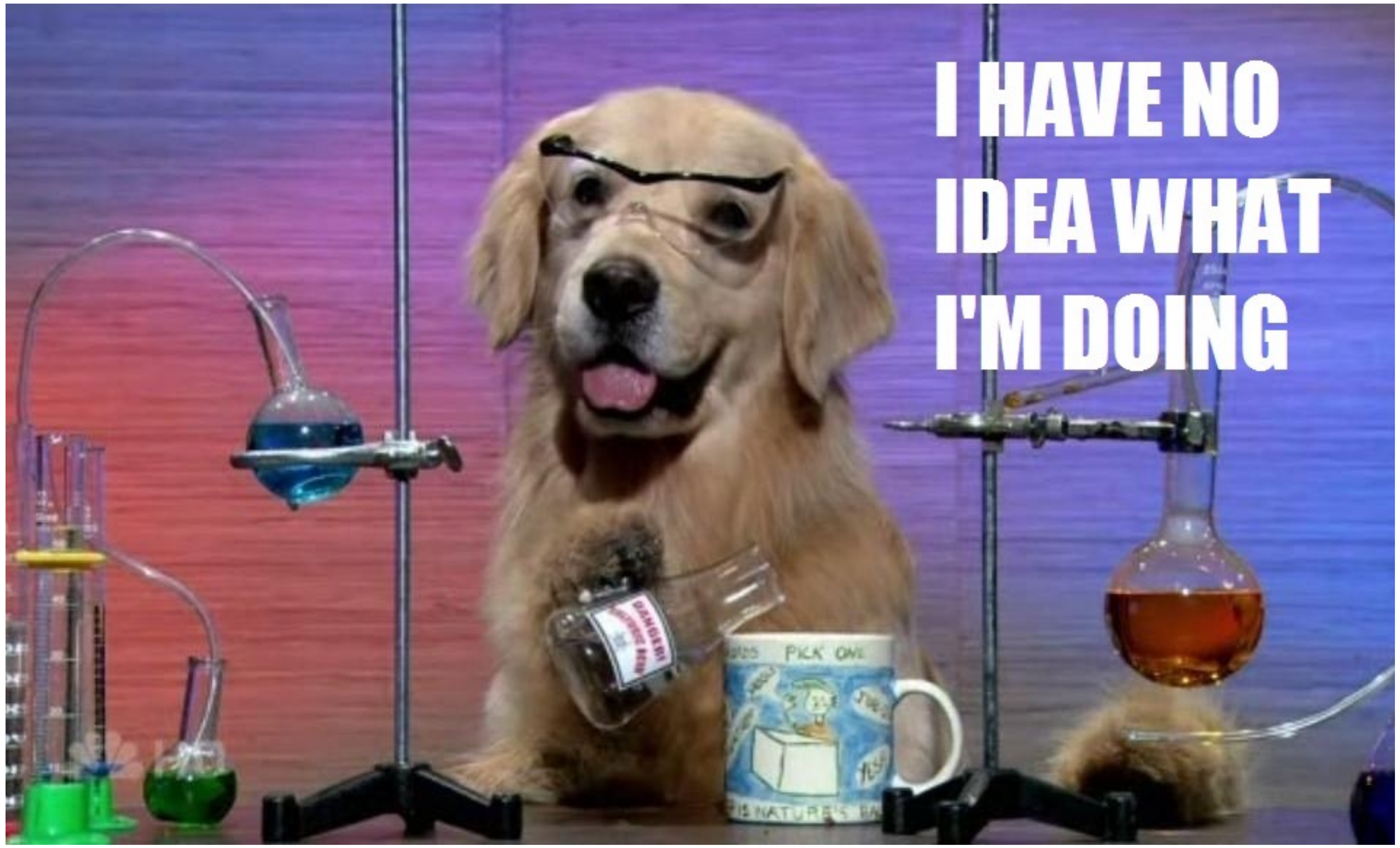
Embedded Elixir with Nerves and All That Jazz (geddit?)

Paul Wilson - @paulanthonywils
http://cultivatehq.com

Cultivate



Never start a talk this way

Cultivate

Cultivate!

(We’re hiring)

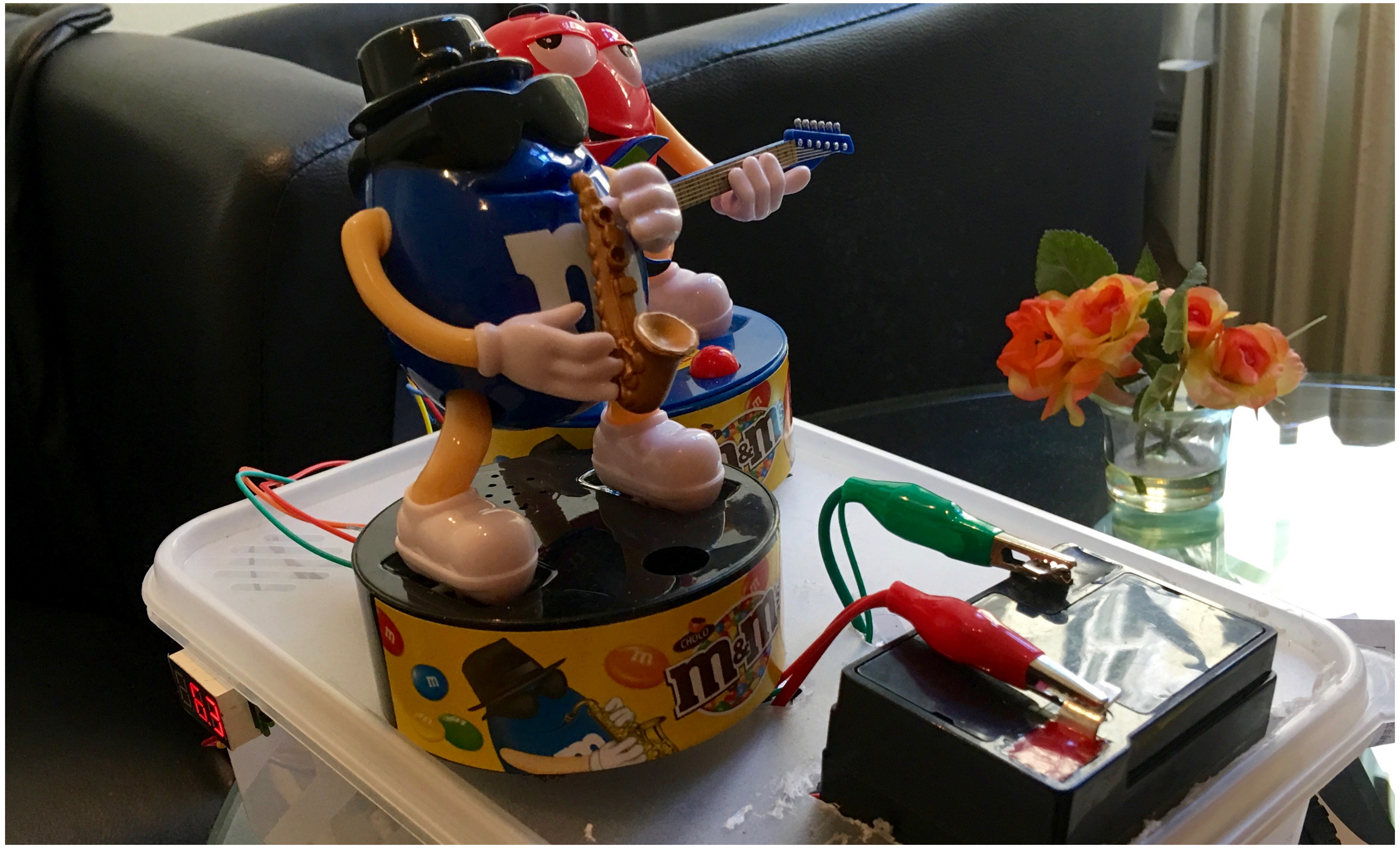
Cultivate

https://www.youtube.com/watch?v=c8ONmQvN3HI

David Henniker

Cultivate

Cultivate



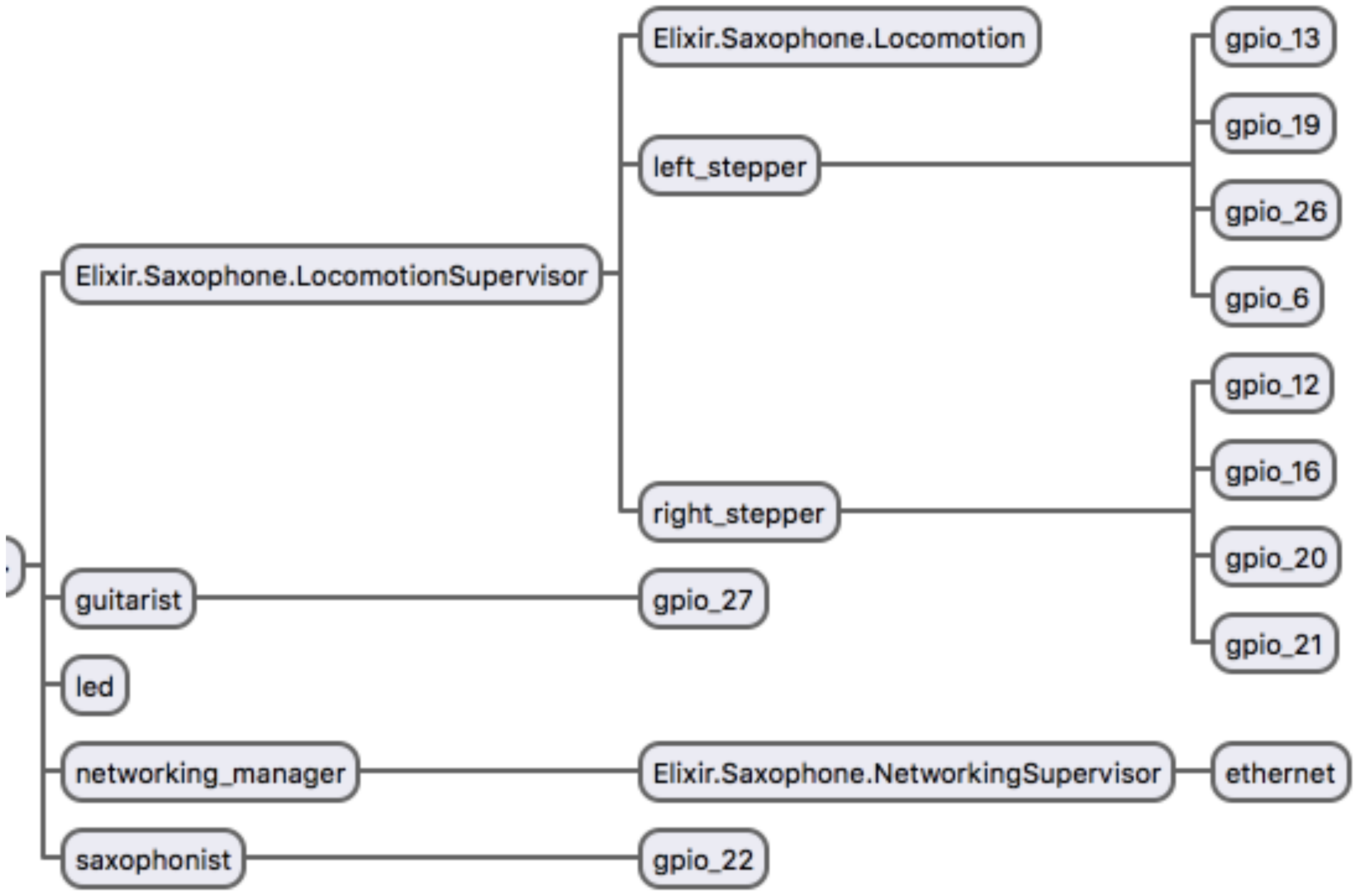
The robot

Cultivate

Why bother?

- Fun
- Profit
- Apply OTP

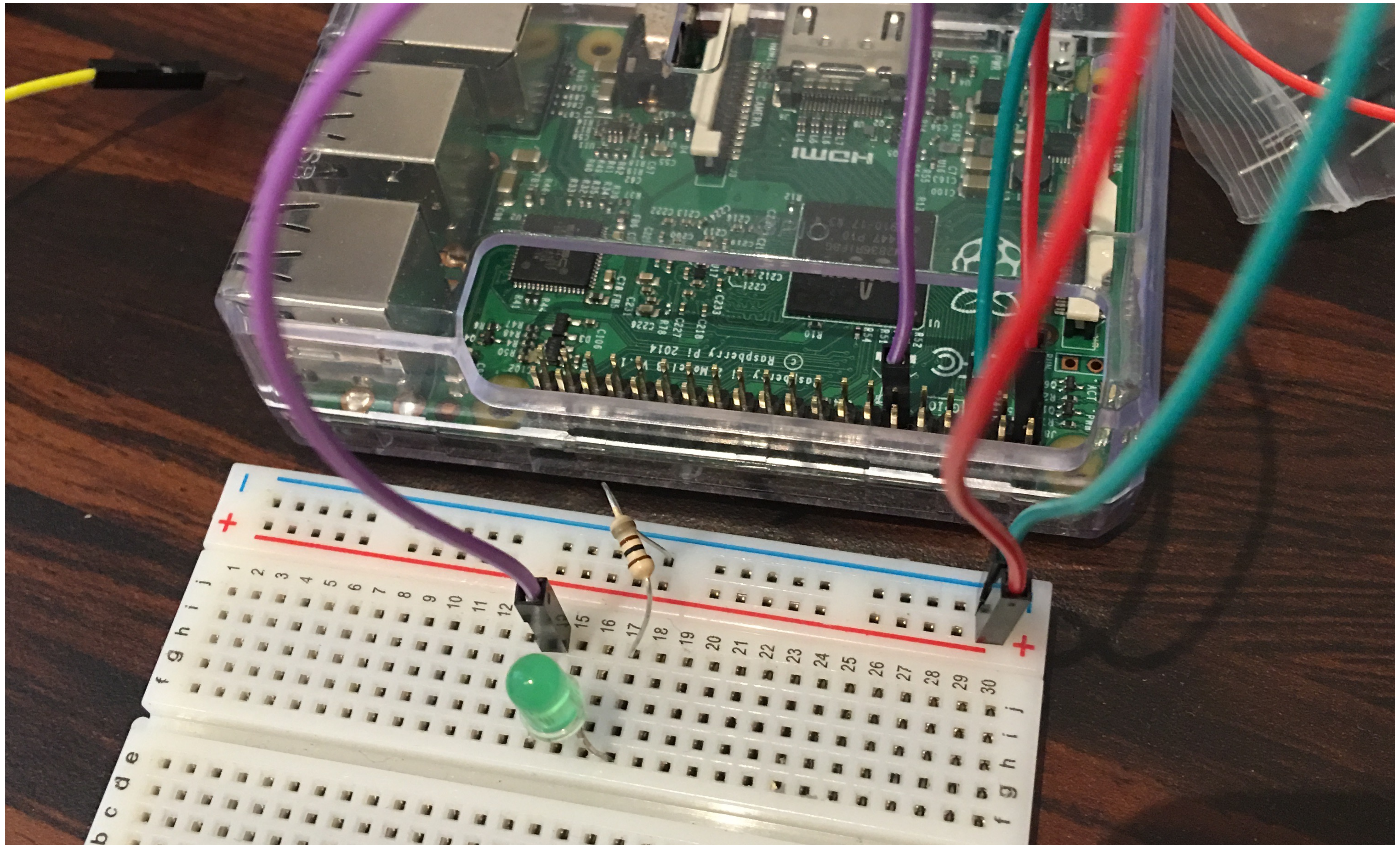
Cultivate



Supervision Tree

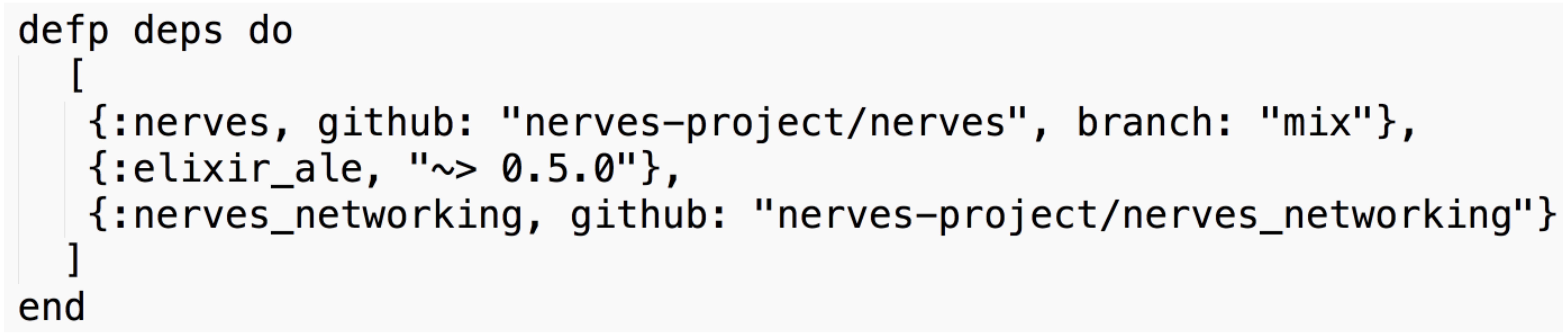
Sax Robot

Cultivate



Trigger the Sax Player

Cultivate

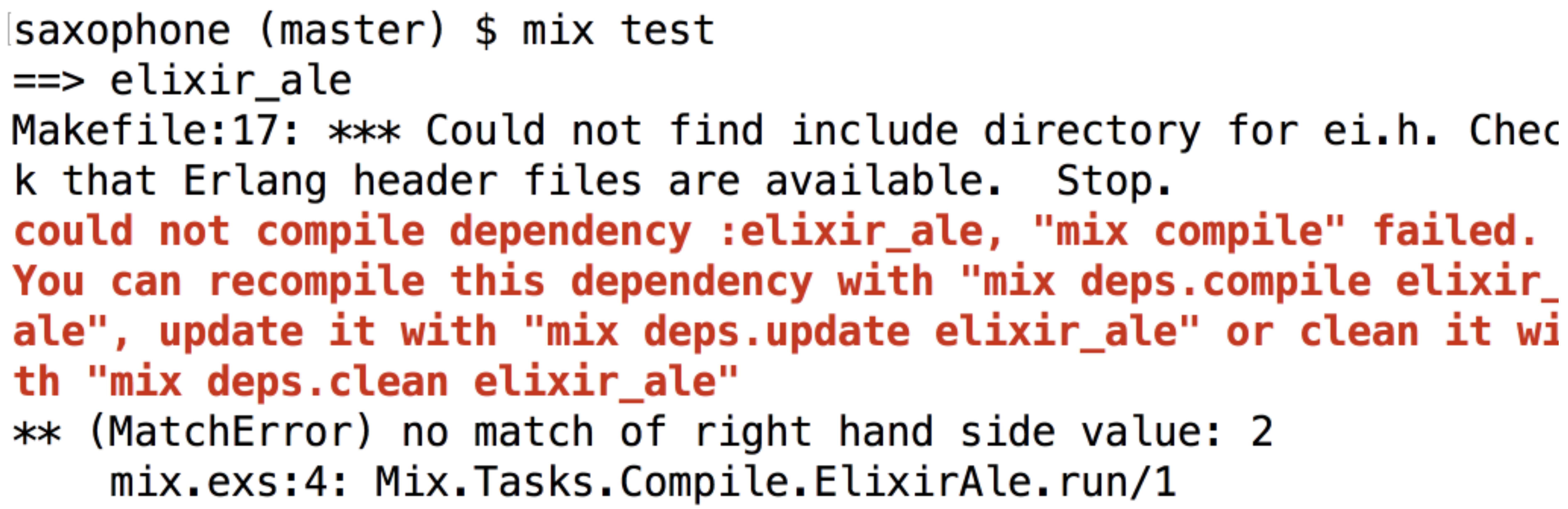


```
defp deps do
  [
   {:nerves, github: "nerves-project/nerves", branch: "mix"},
   {:elixir_ale, "~> 0.5.0"},
   {:nerves_networking, github: "nerves-project/nerves_networking"}
  ]
end
```

Elixir Ale (GPIO) & Nerves.Networking

https://github.com/fhunleth/elixir_ale

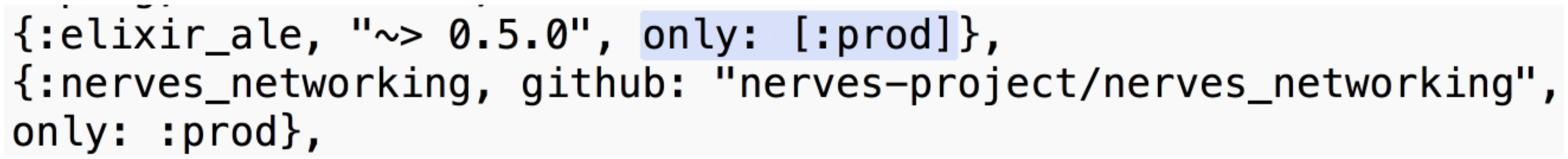
Cultivate



```
saxophone (master) $ mix test
==> elixir_ale
Makefile:17: *** Could not find include directory for ei.h. Chec
k that Erlang header files are available.  Stop.
could not compile dependency :elixir_ale, "mix compile" failed.
You can recompile this dependency with "mix deps.compile elixir_
ale", update it with "mix deps.update elixir_ale" or clean it wi
th "mix deps.clean elixir_ale"
** (MatchError) no match of right hand side value: 2
    mix.exs:4: Mix.Tasks.Compile.ElixirAle.run/1
```

Oh, oh

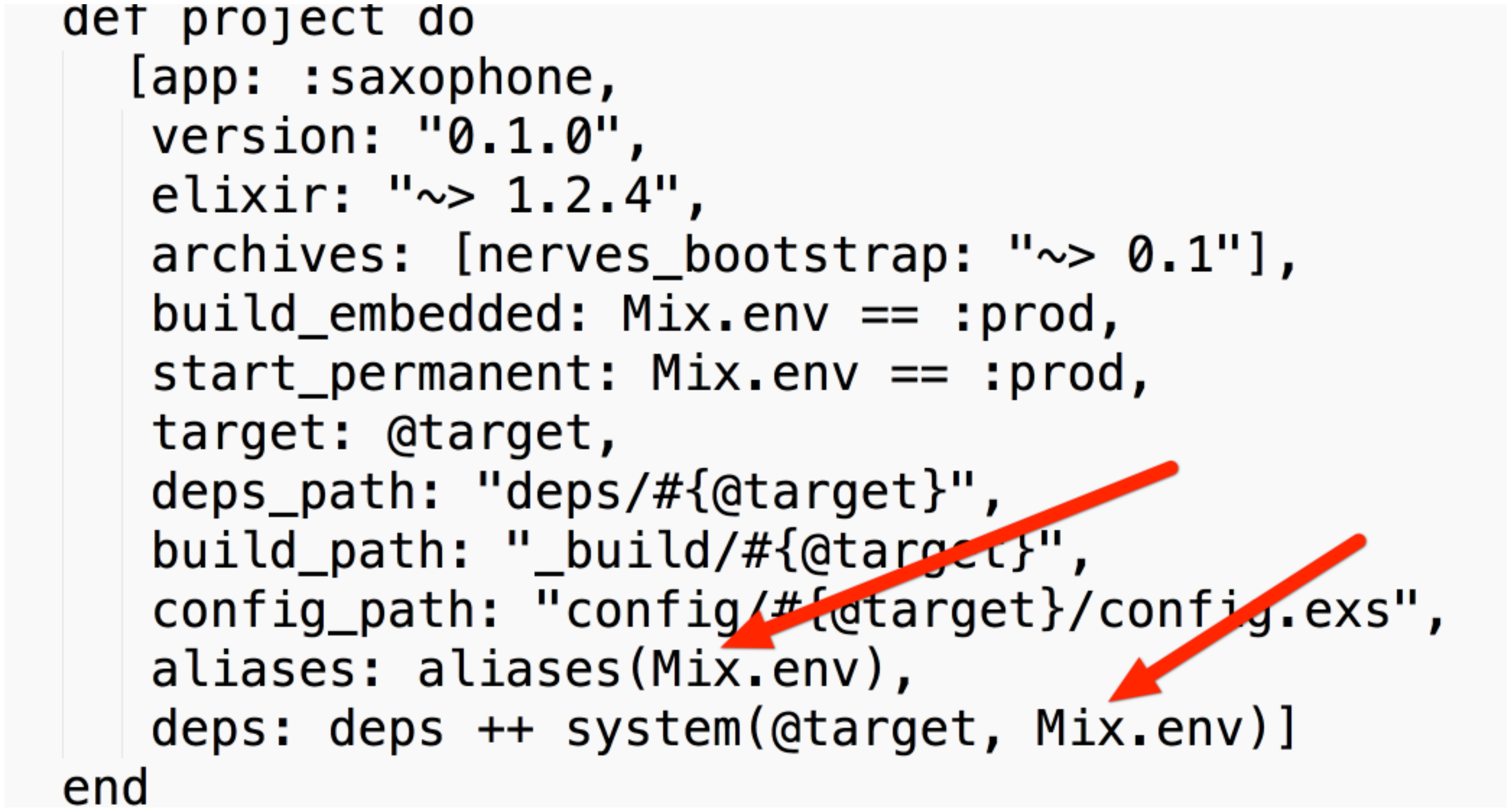
Cultivate



```
{:elixir_ale, "~> 0.5.0", only: [:prod]},
{:nerves_networking, github: "nerves-project/nerves_networking",
only: :prod},
```

Modify mix.exs

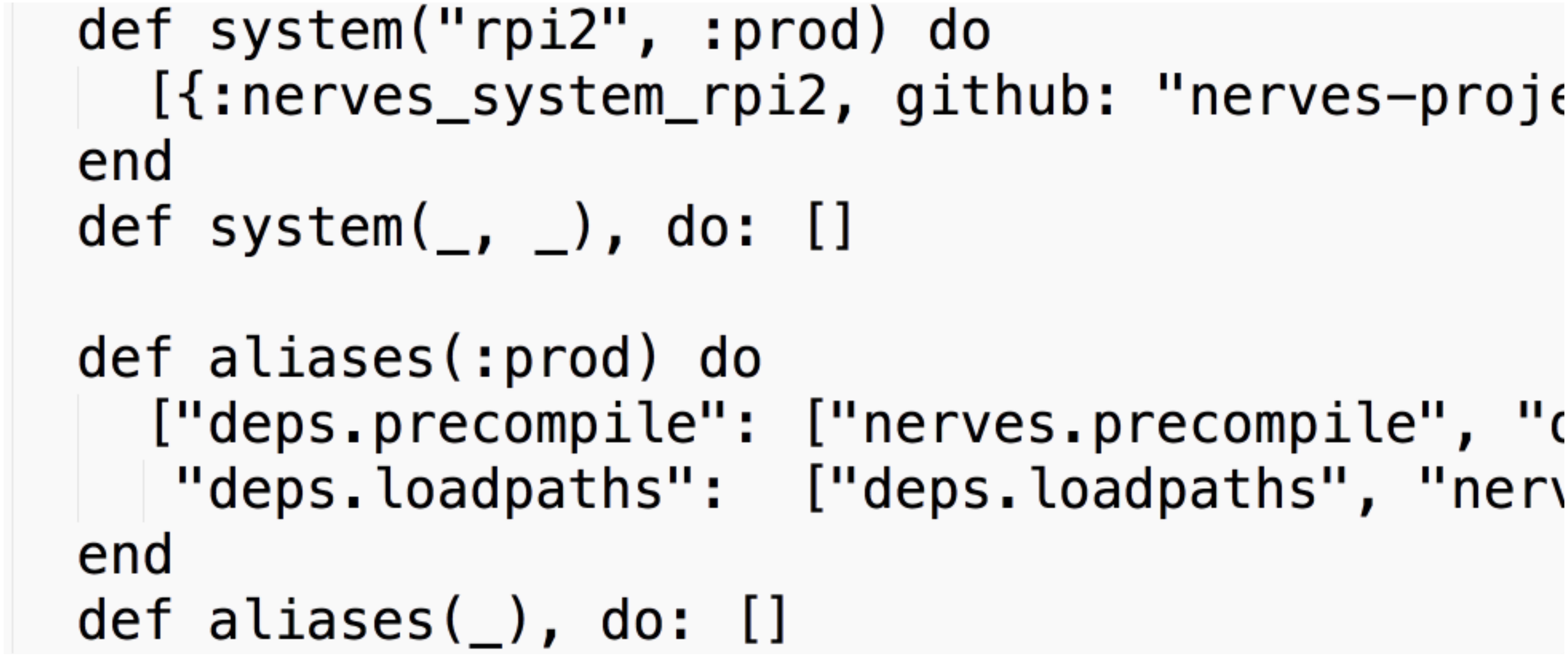
Cultivate



```
def project do
  [app: :saxophone,
   version: "0.1.0",
   elixir: "~> 1.2.4",
   archives: [nerves_bootstrap: "~> 0.1"],
   build_embedded: Mix.env == :prod,
   start_permanent: Mix.env == :prod,
   target: @target,
   deps_path: "deps/#{@target}",
   build_path: "_build/#{@target}",
   config_path: "config/#{@target}/config.exs",
   aliases: aliases(Mix.env),
   deps: deps ++ system(@target, Mix.env)]
end
```

Modify mix.exs

Cultivate



```
def system("rpi2", :prod) do
  [{:nerves_system_rpi2, github: "nerves-proj
end
def system(_, _), do: []

def aliases(:prod) do
  ["deps.precompile": ["nerves.precompile", "
   "deps.loadpaths":  ["deps.loadpaths", "nerv
end
def aliases(_), do: []
```

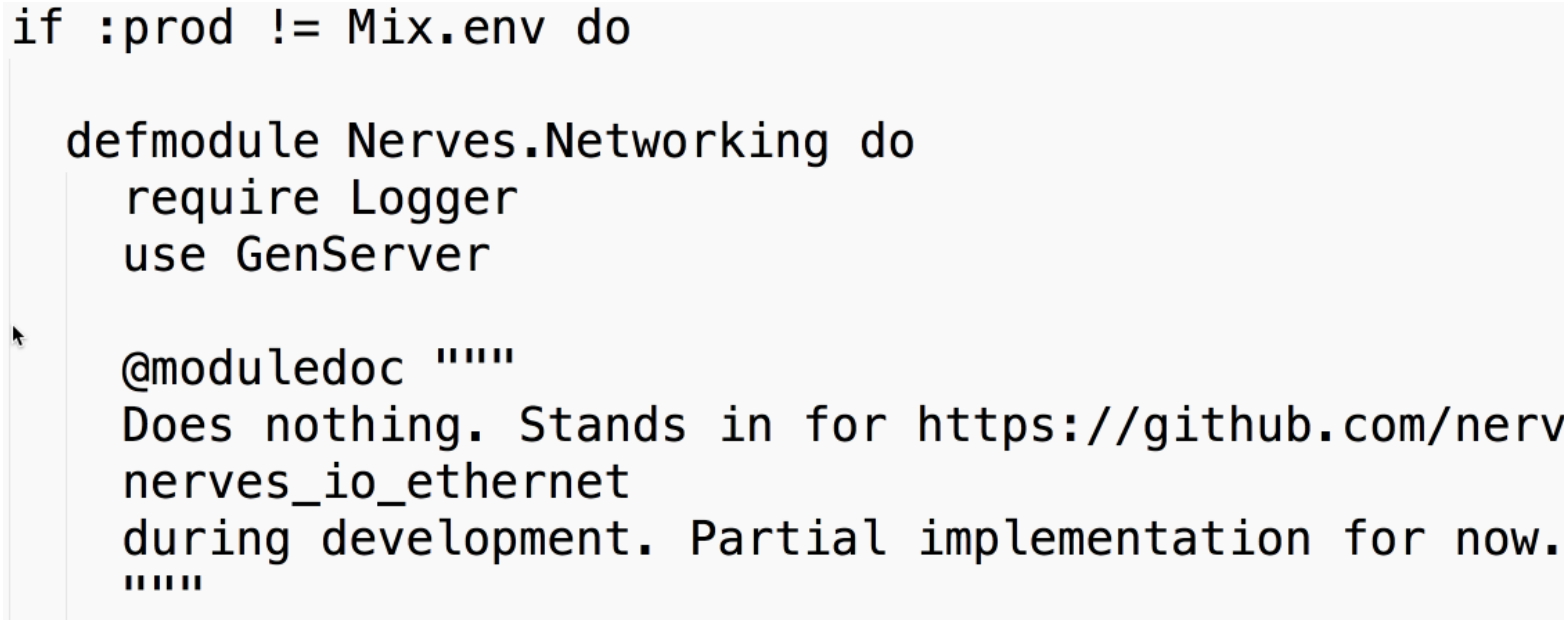
Modify mix.exs

Cultivate

```
defp applications do
  general_apps = [:logger, :runtime_tools]
  case Mix.env do
    :prod -> [:nerves, :nerves_networking, :elixir_ale | general_apps]
    _ -> general_apps
  end
end
```

Modify mix.exs

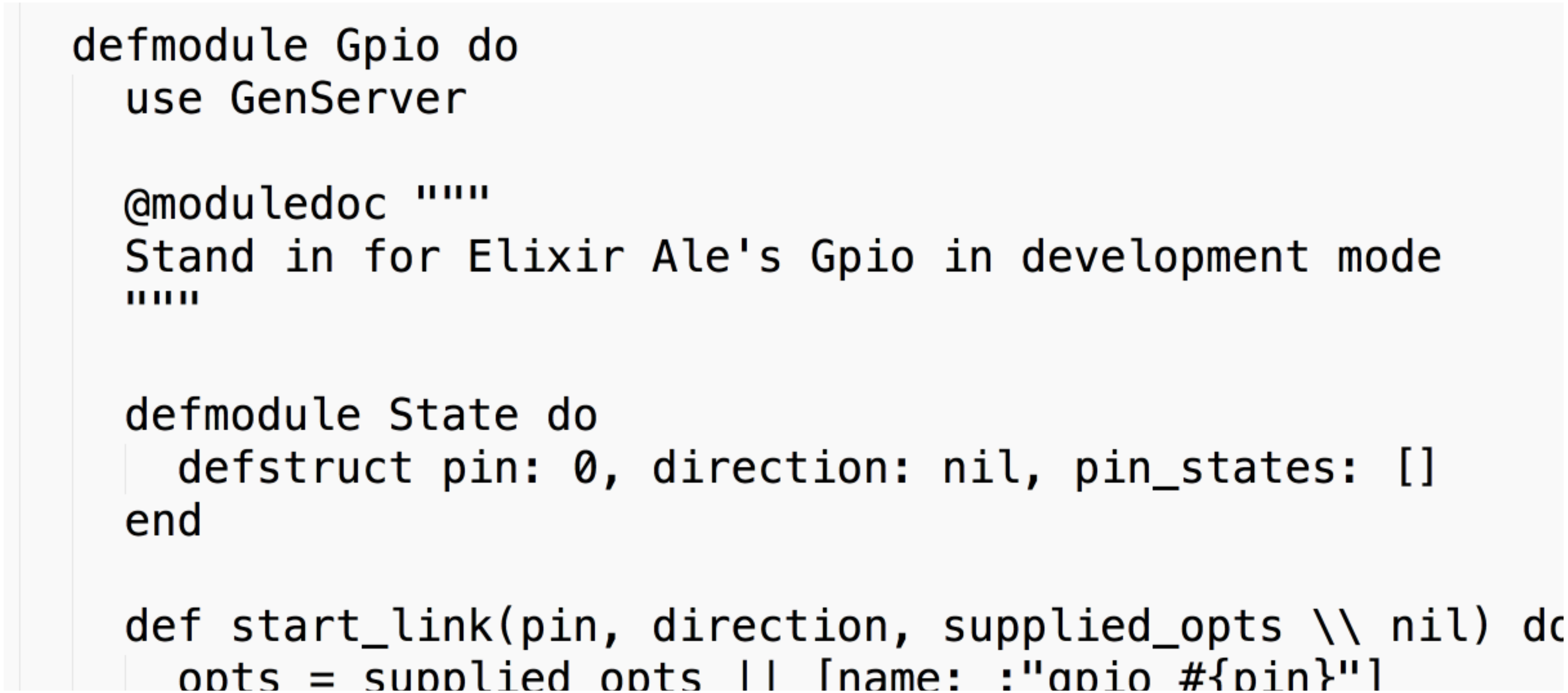
Cultivate



```
if :prod != Mix.env do

  defmodule Nerves.Networking do
    require Logger
    use GenServer

    @moduledoc """
    Does nothing. Stands in for https://github.com/nerv
    nerves_io_ethernet
    during development. Partial implementation for now.
    """
```

Fake modules (dev & test)

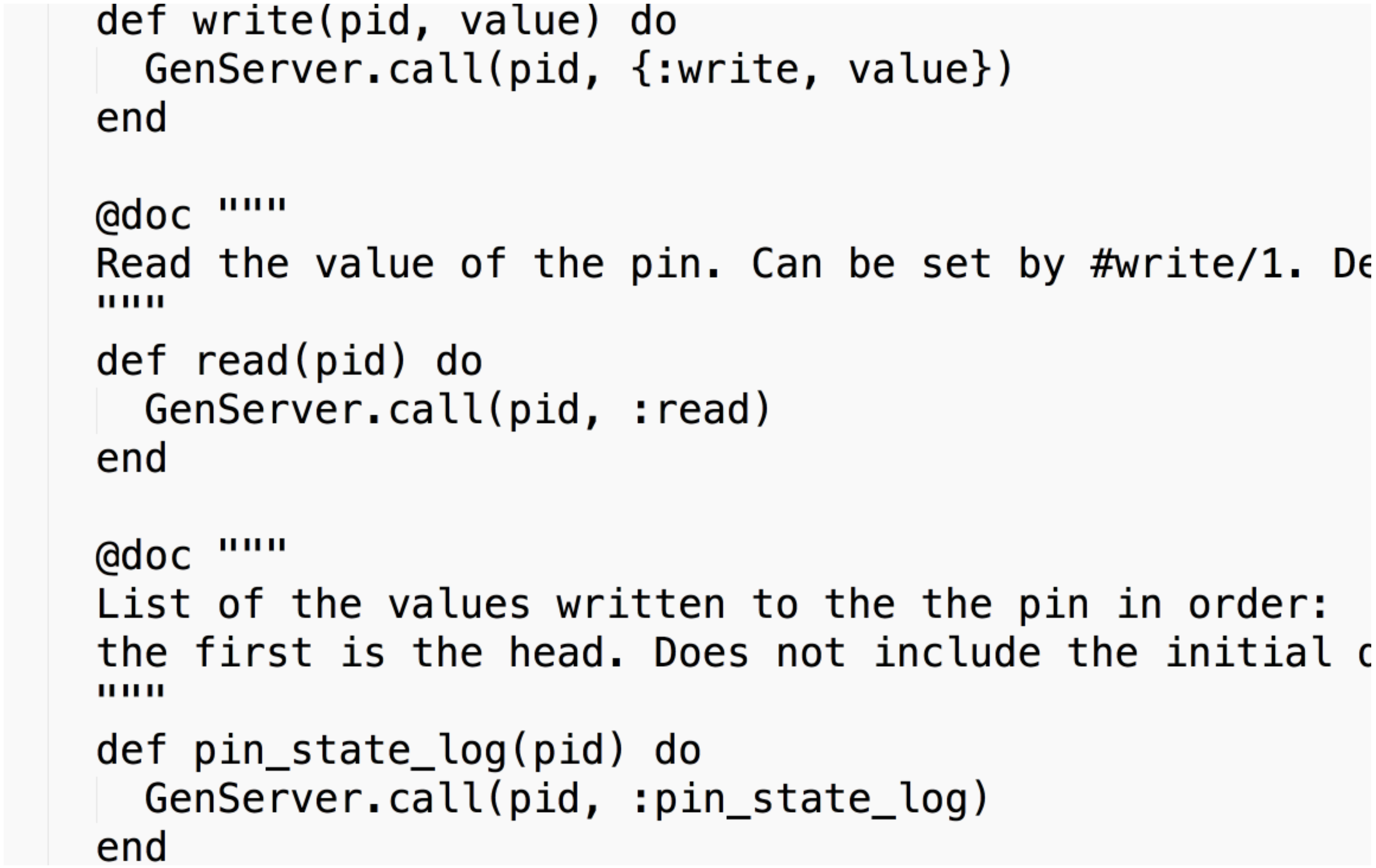
Cultivate



```
defmodule Gpio do
  use GenServer

  @moduledoc """
  Stand in for Elixir Ale's Gpio in development mode
  """

  defmodule State do
    defstruct pin: 0, direction: nil, pin_states: []
  end

  def start_link(pin, direction, supplied_opts \\ nil) dc
    opts = supplied opts || [name: :"gpio #{pin}"]
```

Fake objects (dev & test)

Cultivate


```
def write(pid, value) do
  GenServer.call(pid, {:write, value})
end

@doc """
Read the value of the pin. Can be set by #write/1. De
"""
def read(pid) do
  GenServer.call(pid, :read)
end

@doc """
List of the values written to the the pin in order:
the first is the head. Does not include the initial c
"""
def pin_state_log(pid) do
  GenServer.call(pid, :pin_state_log)
end
```

Extra support for testing

Cultivate

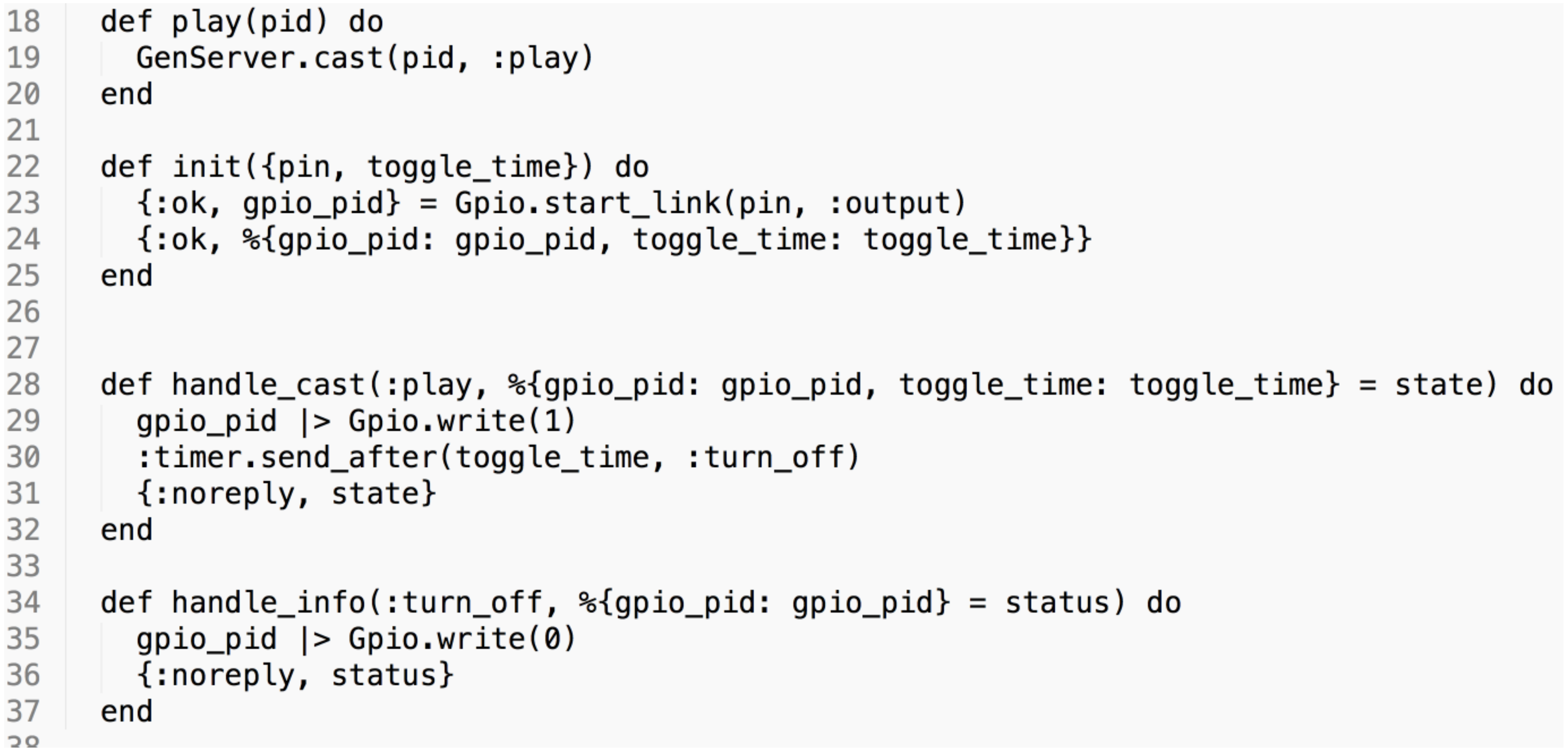
```
use Mix.Config

config :saxophone, :saxophonist, pin: 4, toggle_time: 0
```

/config/rpi2/test.exs

```
  test "play toggles the pin on and off" do
    Saxophonist.play(:saxophonist)
    :timer.sleep(1)
    assert [1, 0] == @gpio |> Gpio.pin_state_log
  end
```

/test/saxophonist_test.exs

Cultivate

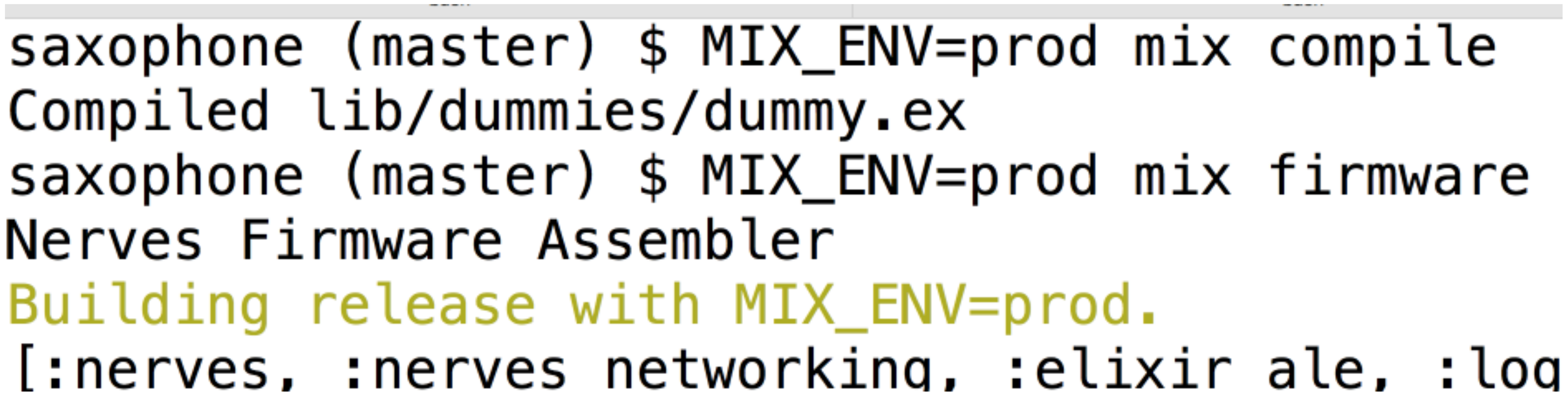


```elixir
def play(pid) do
  GenServer.cast(pid, :play)
end

def init({pin, toggle_time}) do
  {:ok, gpio_pid} = Gpio.start_link(pin, :output)
  {:ok, %{gpio_pid: gpio_pid, toggle_time: toggle_time}}
end


def handle_cast(:play, %{gpio_pid: gpio_pid, toggle_time: toggle_time} = state) do
  gpio_pid |> Gpio.write(1)
  :timer.send_after(toggle_time, :turn_off)
  {:noreply, state}
end

def handle_info(:turn_off, %{gpio_pid: gpio_pid} = status) do
  gpio_pid |> Gpio.write(0)
  {:noreply, status}
end
```

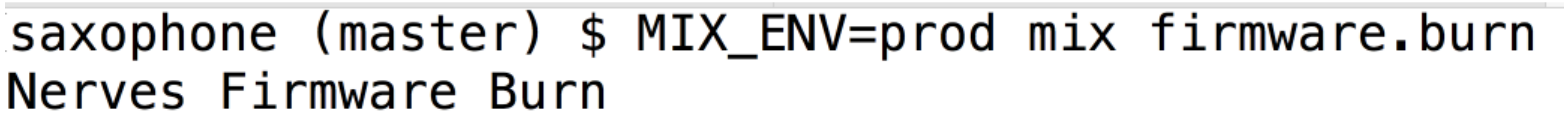
Saxophonist implementation

```
saxophone (master) $ mix test
..........................

Finished in 0.4 seconds (0.2s on load, 0.1s on tests)
26 tests, 0 failures

Randomized with seed 804014
saxophone (master) $
```

\o/



```
saxophone (master) $ MIX_ENV=prod mix compile
Compiled lib/dummies/dummy.ex
saxophone (master) $ MIX_ENV=prod mix firmware
Nerves Firmware Assembler
Building release with MIX_ENV=prod.
[:nerves, :nerves_networking, :elixir_ale, :log
```



```
saxophone (master) $ MIX_ENV=prod mix firmware.burn
Nerves Firmware Burn
```

MIX_ENV=prod

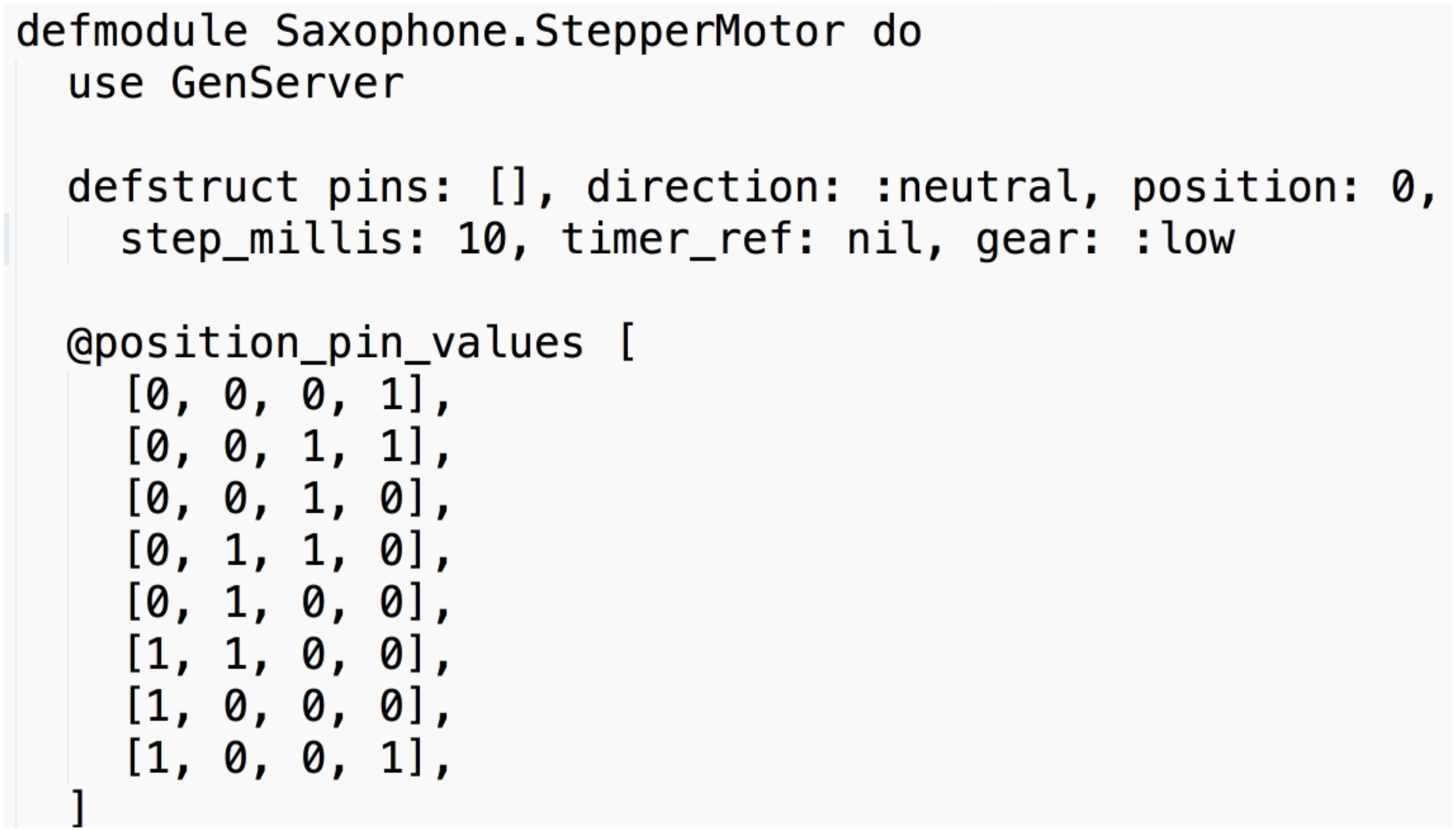
Cultivate

Cultivate

Stepper Motor

28BYJ-48 with ULN2003 Driver Board

Cultivate



```
defmodule Saxophone.StepperMotor do
  use GenServer

  defstruct pins: [], direction: :neutral, position: 0,
    step_millis: 10, timer_ref: nil, gear: :low

  @position_pin_values [
    [0, 0, 0, 1],
    [0, 0, 1, 1],
    [0, 0, 1, 0],
    [0, 1, 1, 0],
    [0, 1, 0, 0],
    [1, 1, 0, 0],
    [1, 0, 0, 0],
    [1, 0, 0, 1],
  ]
```

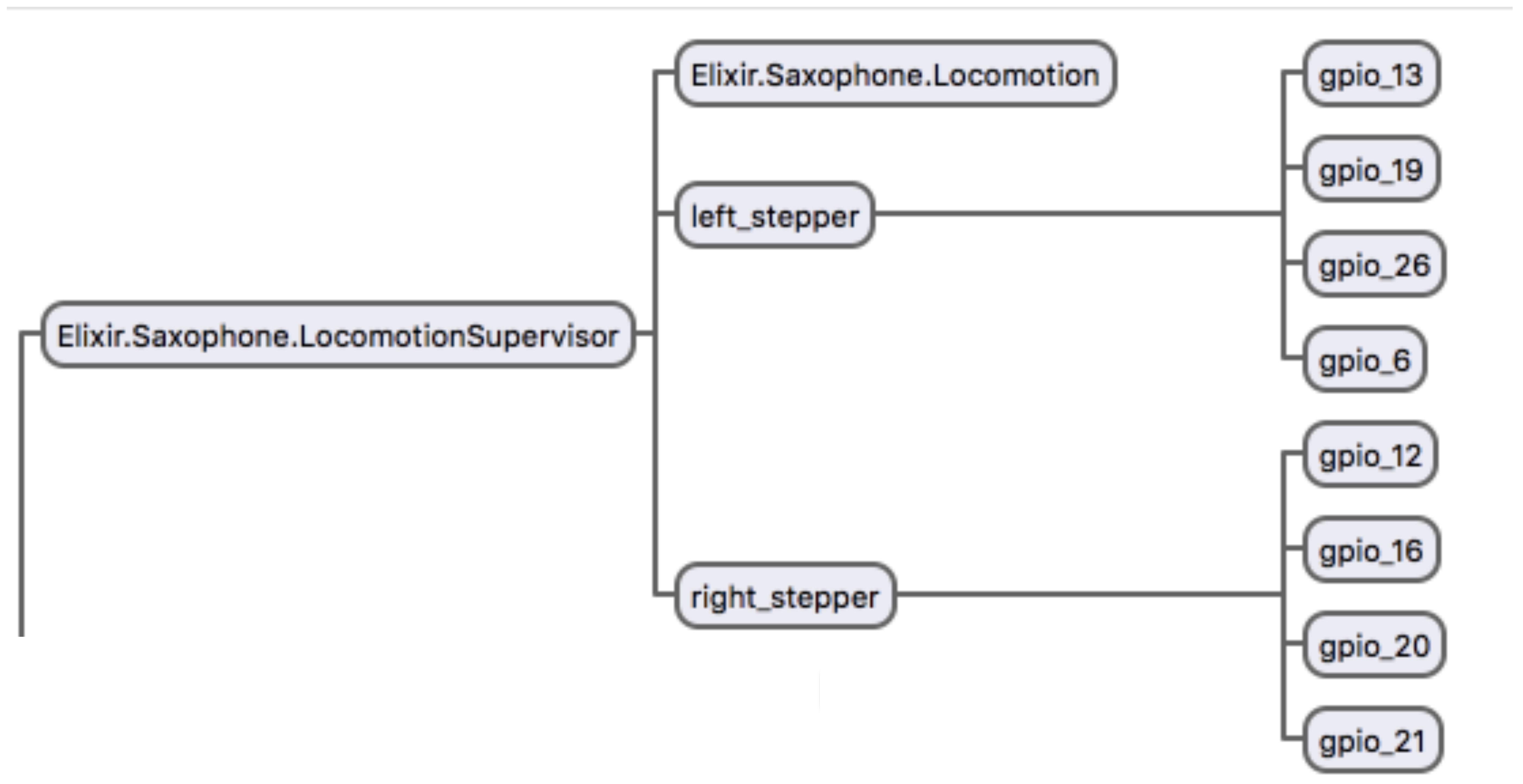
Stepper motor

Cultivate

```elixir
test "cycling back", %{pid: pid} do
  pid |> StepperMotor.set_direction(:back)
  (7..0) |> Enum.each(fn i ->
    send(pid, :step)
    :timer.sleep(1)
    assert StepperMotor.state(pid).position == i
  end)

  assert Gpio.pin_state_log(:gpio_30) == [0, 0, 0,
  1, 0] |> Enum.reverse
  assert Gpio.pin_state_log(:gpio_33) == [1, 1, 0,
  1, 1] |> Enum.reverse
end
```

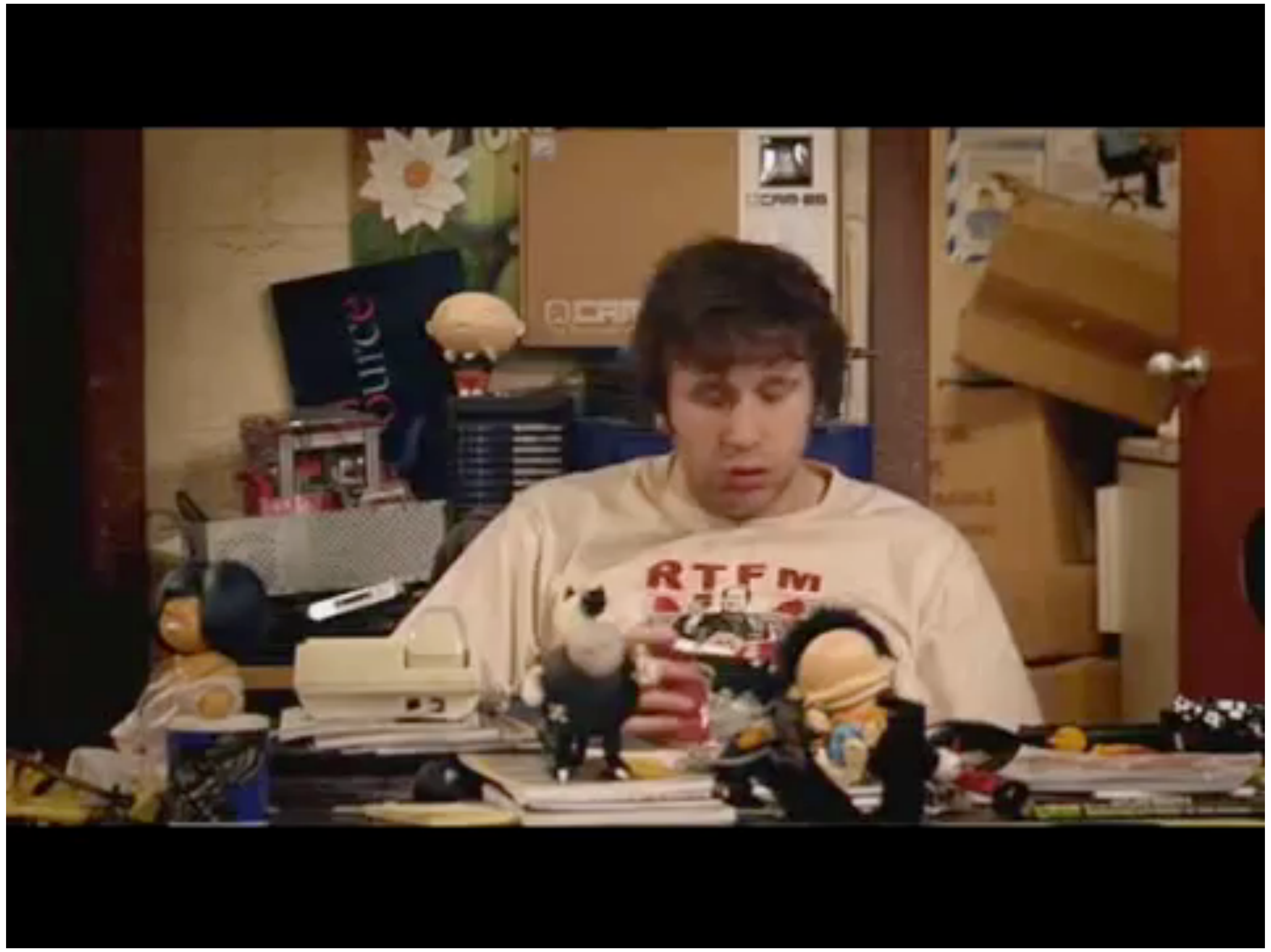
One of the motor tests

Cultivate



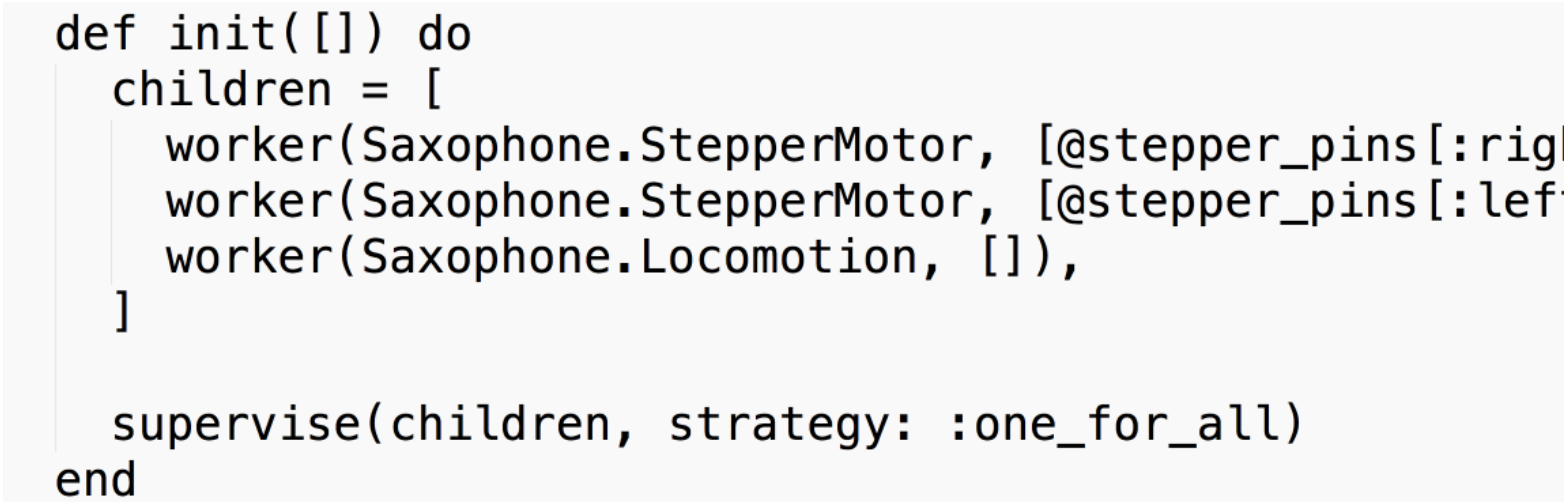
Motor supervison

Cultivate



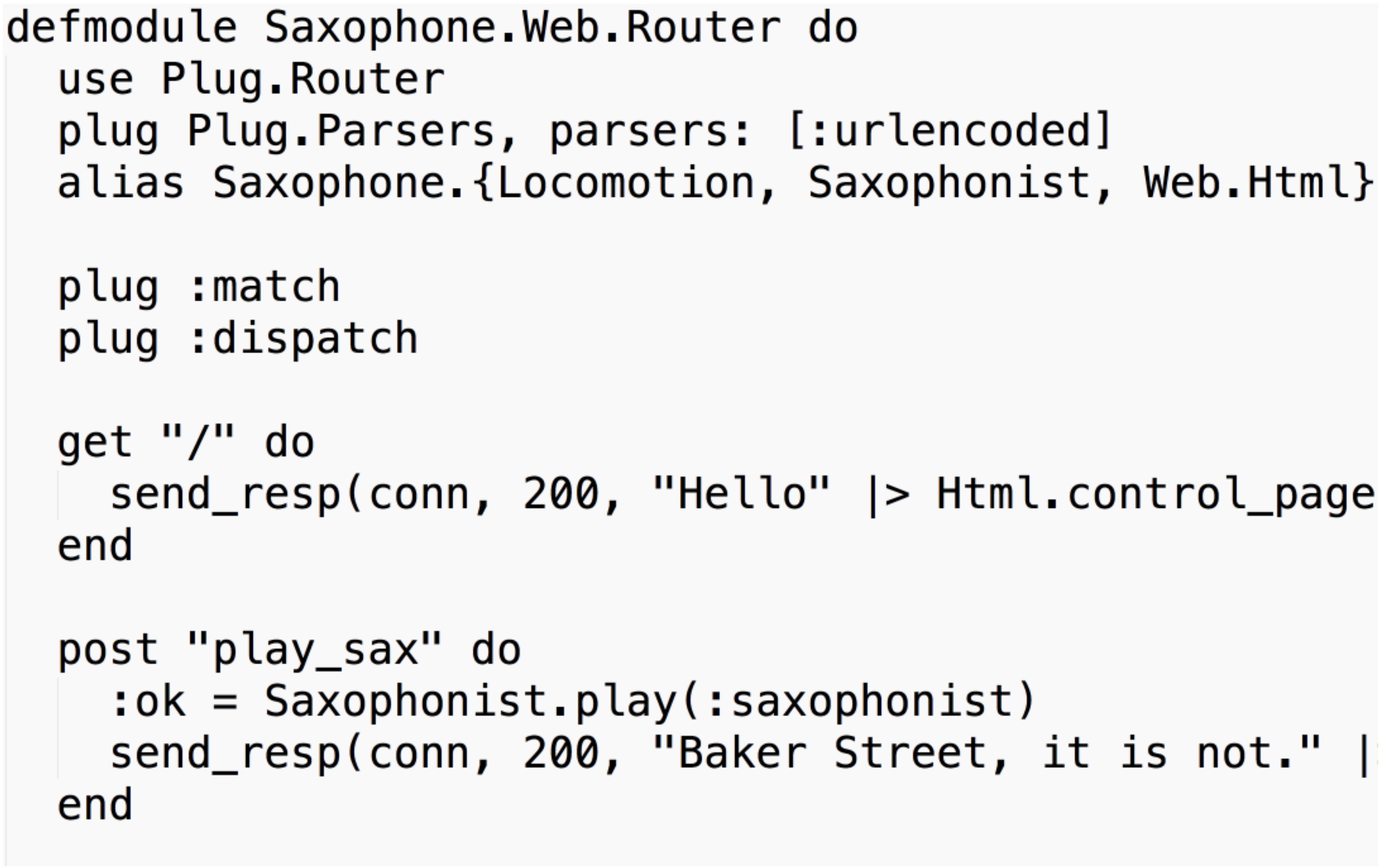
Restart in known good state

Cultivate



```
def init([]) do
  children = [
    worker(Saxophone.StepperMotor, [@stepper_pins[:rig
    worker(Saxophone.StepperMotor, [@stepper_pins[:lef
    worker(Saxophone.Locomotion, []),
  ]

  supervise(children, strategy: :one_for_all)
end
```

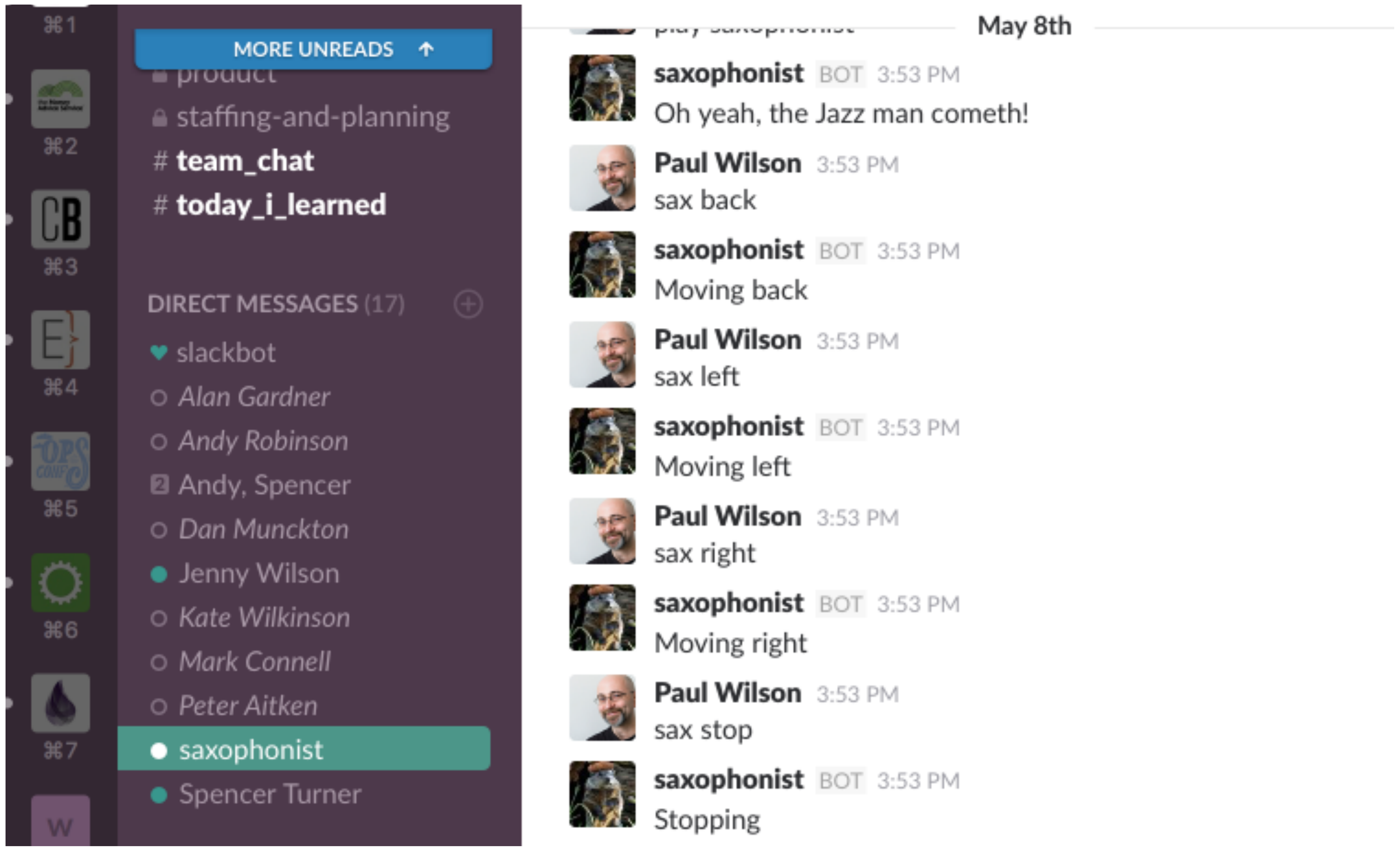
one_for_all - known good state



```
defmodule Saxophone.Web.Router do
  use Plug.Router
  plug Plug.Parsers, parsers: [:urlencoded]
  alias Saxophone.{Locomotion, Saxophonist, Web.Html}

  plug :match
  plug :dispatch

  get "/" do
    send_resp(conn, 200, "Hello" |> Html.control_page
  end

  post "play_sax" do
    :ok = Saxophonist.play(:saxophonist)
    send_resp(conn, 200, "Baker Street, it is not." |
  end
```

Web interface

Cultivate



saxophonist BOT 3:53 PM
Oh yeah, the Jazz man cometh!

Paul Wilson 3:53 PM
sax back

saxophonist BOT 3:53 PM
Moving back

Paul Wilson 3:53 PM
sax left

saxophonist BOT 3:53 PM
Moving left

Paul Wilson 3:53 PM
sax right

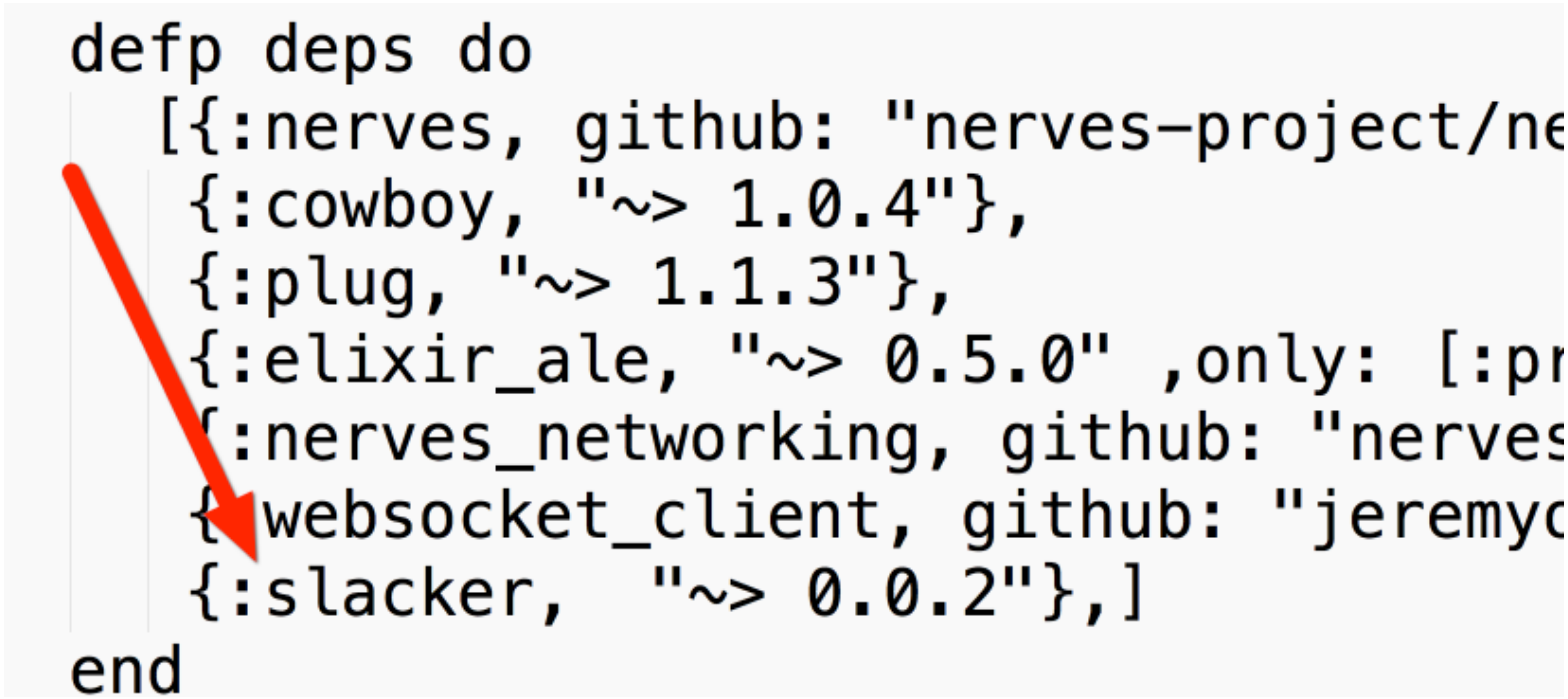
saxophonist BOT 3:53 PM
Moving right

Paul Wilson 3:53 PM
sax stop

saxophonist BOT 3:53 PM
Stopping

Slackbot interface

Cultivate

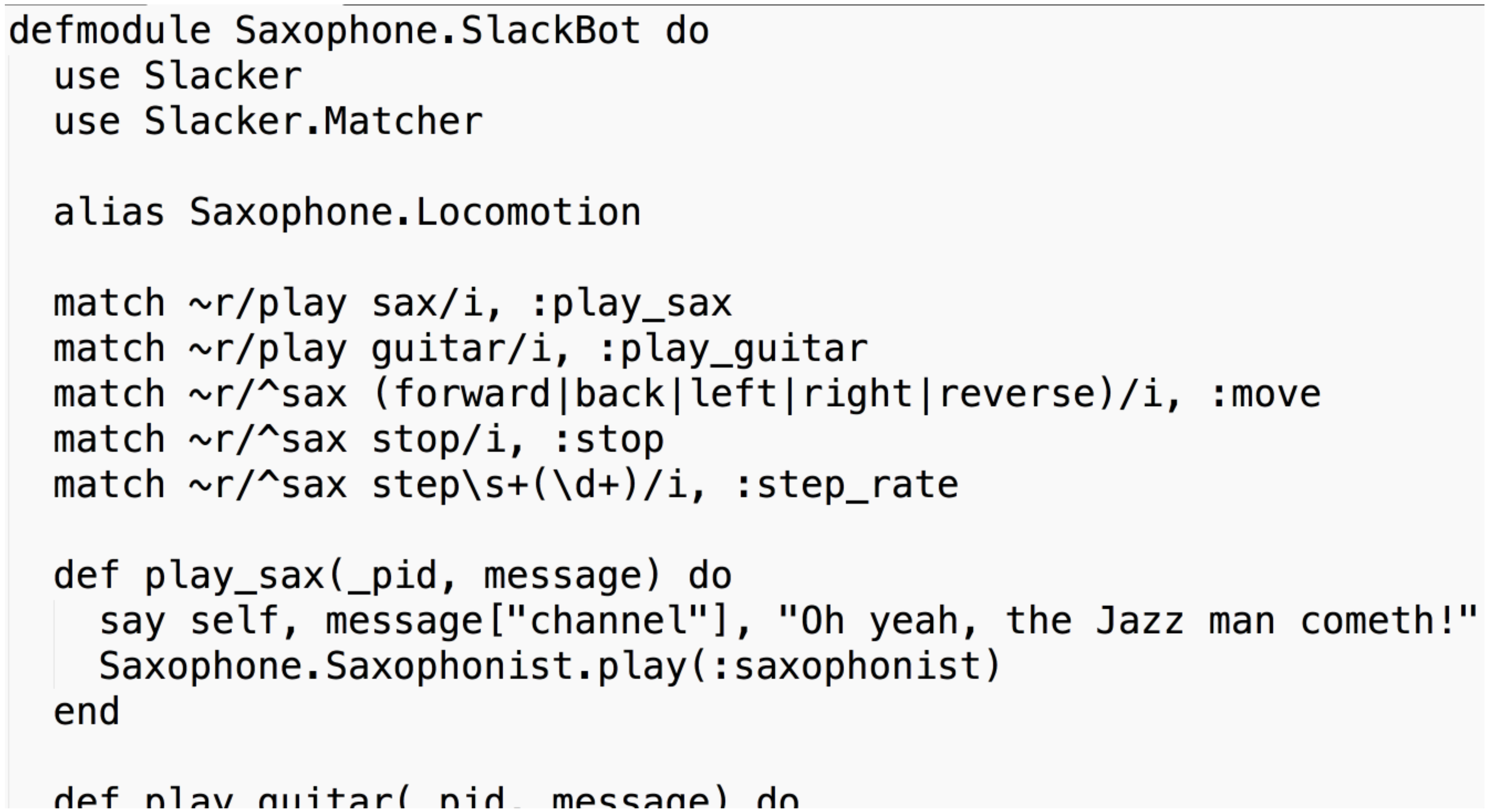


```
defp deps do
  [{:nerves, github: "nerves-project/ne
   {:cowboy, "~> 1.0.4"},
   {:plug, "~> 1.1.3"},
   {:elixir_ale, "~> 0.5.0" ,only: [:pr
   {:nerves_networking, github: "nerves
   {:websocket_client, github: "jeremy
   {:slacker,  "~> 0.0.2"},]
end
```

Slacker

https://github.com/koudelka/slacker

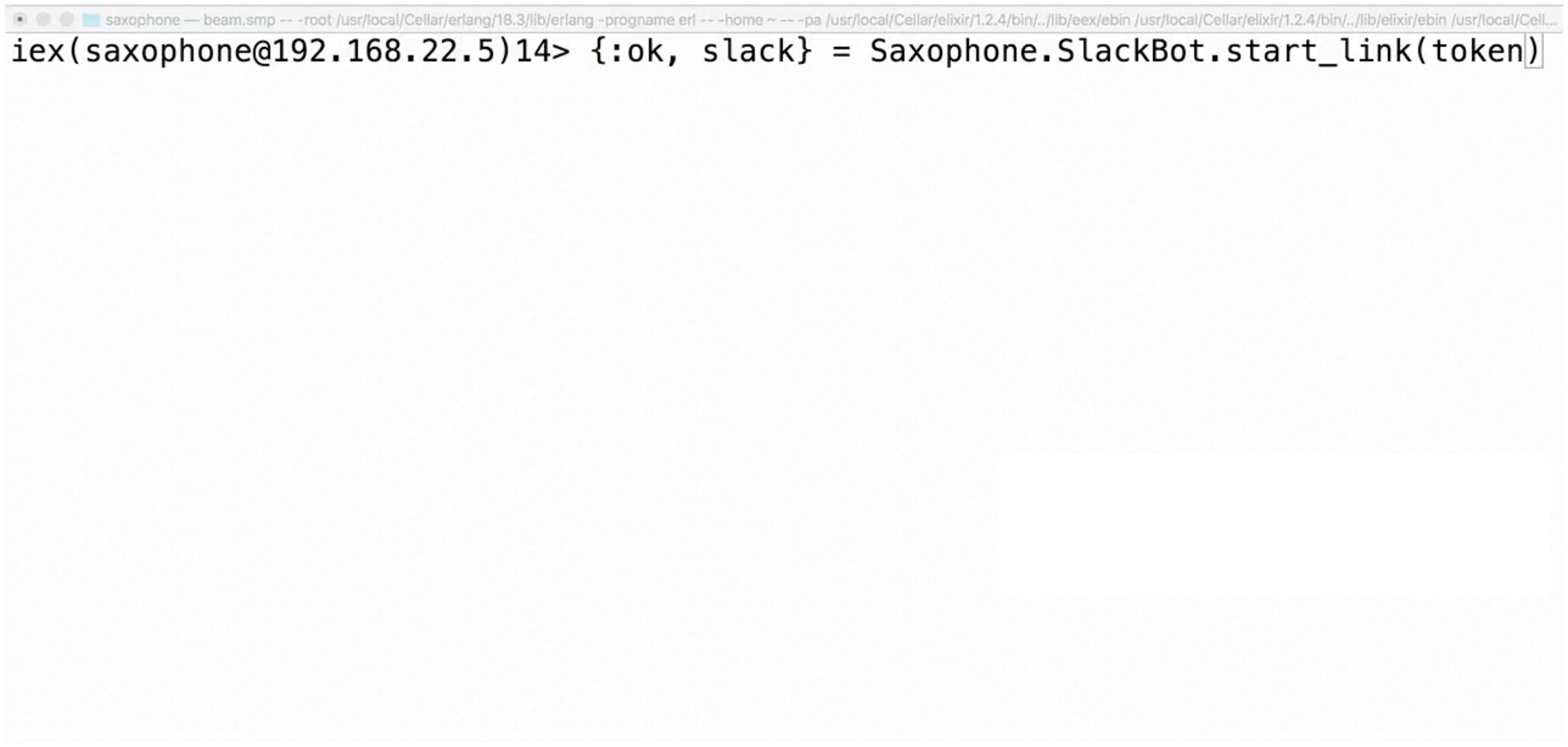
Cultivate



```
defmodule Saxophone.SlackBot do
  use Slacker
  use Slacker.Matcher

  alias Saxophone.Locomotion

  match ~r/play sax/i, :play_sax
  match ~r/play guitar/i, :play_guitar
  match ~r/^sax (forward|back|left|right|reverse)/i, :move
  match ~r/^sax stop/i, :stop
  match ~r/^sax step\s+(\d+)/i, :step_rate

  def play_sax(_pid, message) do
    say self, message["channel"], "Oh yeah, the Jazz man cometh!"
    Saxophone.Saxophonist.play(:saxophonist)
  end

  def play_guitar(_pid, message) do
```

Slackbot code

Cultivate



saxophone — beam.smp -- -root /usr/local/Cellar/erlang/18.3/lib/erlang -progname erl -- -home ~ -- -pa /usr/local/Cellar/elixir/1.2.4/bin/../lib/eex/ebin /usr/local/Cellar/elixir/1.2.4/bin/../lib/elixir/ebin /usr/local/Cell...

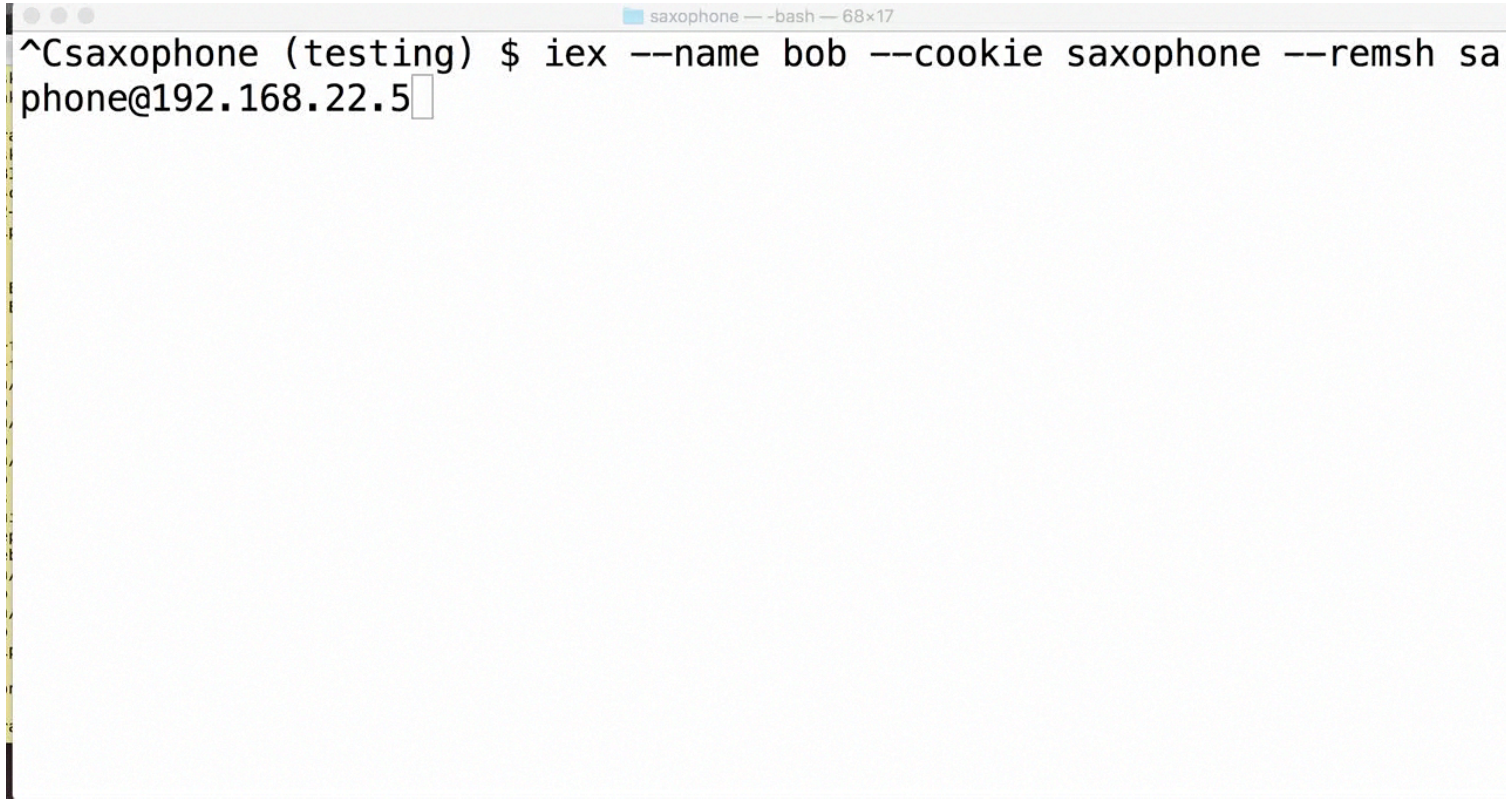
```
iex(saxophone@192.168.22.5)14> {:ok, slack} = Saxophone.SlackBot.start_link(token)
```

Let’s try it!

Cultivate

The 1970s

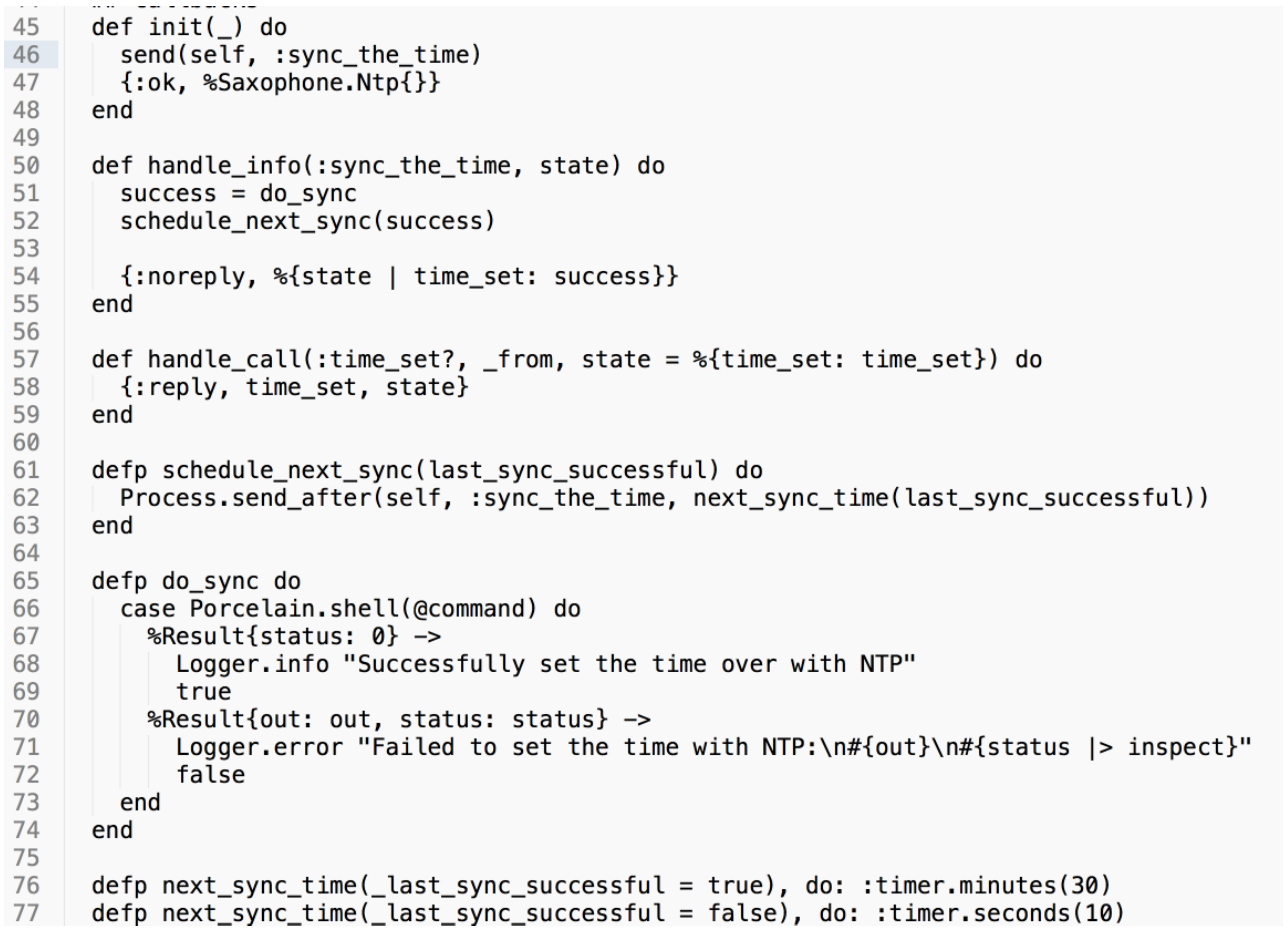
Cultivate



```
^Csaxophone (testing) $ iex --name bob --cookie saxophone --remsh sa
phone@192.168.22.5
```

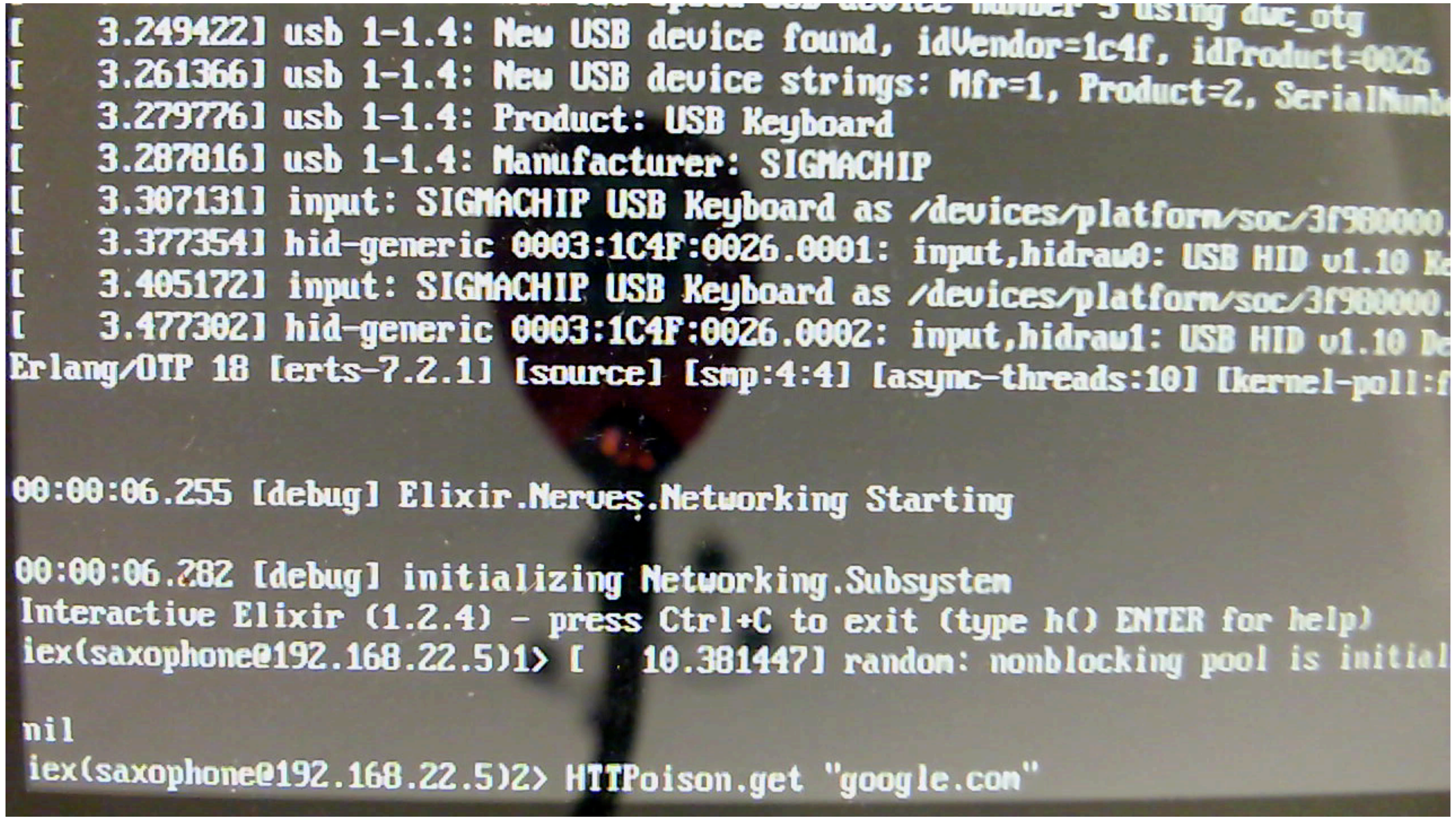
ntpd to the rescue

Cultivate



```elixir
def init(_) do
  send(self, :sync_the_time)
  {:ok, %Saxophone.Ntp{}}
end

def handle_info(:sync_the_time, state) do
  success = do_sync
  schedule_next_sync(success)

  {:noreply, %{state | time_set: success}}
end

def handle_call(:time_set?, _from, state = %{time_set: time_set}) do
  {:reply, time_set, state}
end

defp schedule_next_sync(last_sync_successful) do
  Process.send_after(self, :sync_the_time, next_sync_time(last_sync_successful))
end

defp do_sync do
  case Porcelain.shell(@command) do
    %Result{status: 0} ->
      Logger.info "Successfully set the time over with NTP"
      true
    %Result{out: out, status: status} ->
      Logger.error "Failed to set the time with NTP:\n#{out}\n#{status |> inspect}"
      false
  end
end

defp next_sync_time(_last_sync_successful = true), do: :timer.minutes(30)
defp next_sync_time(_last_sync_successful = false), do: :timer.seconds(10)
```

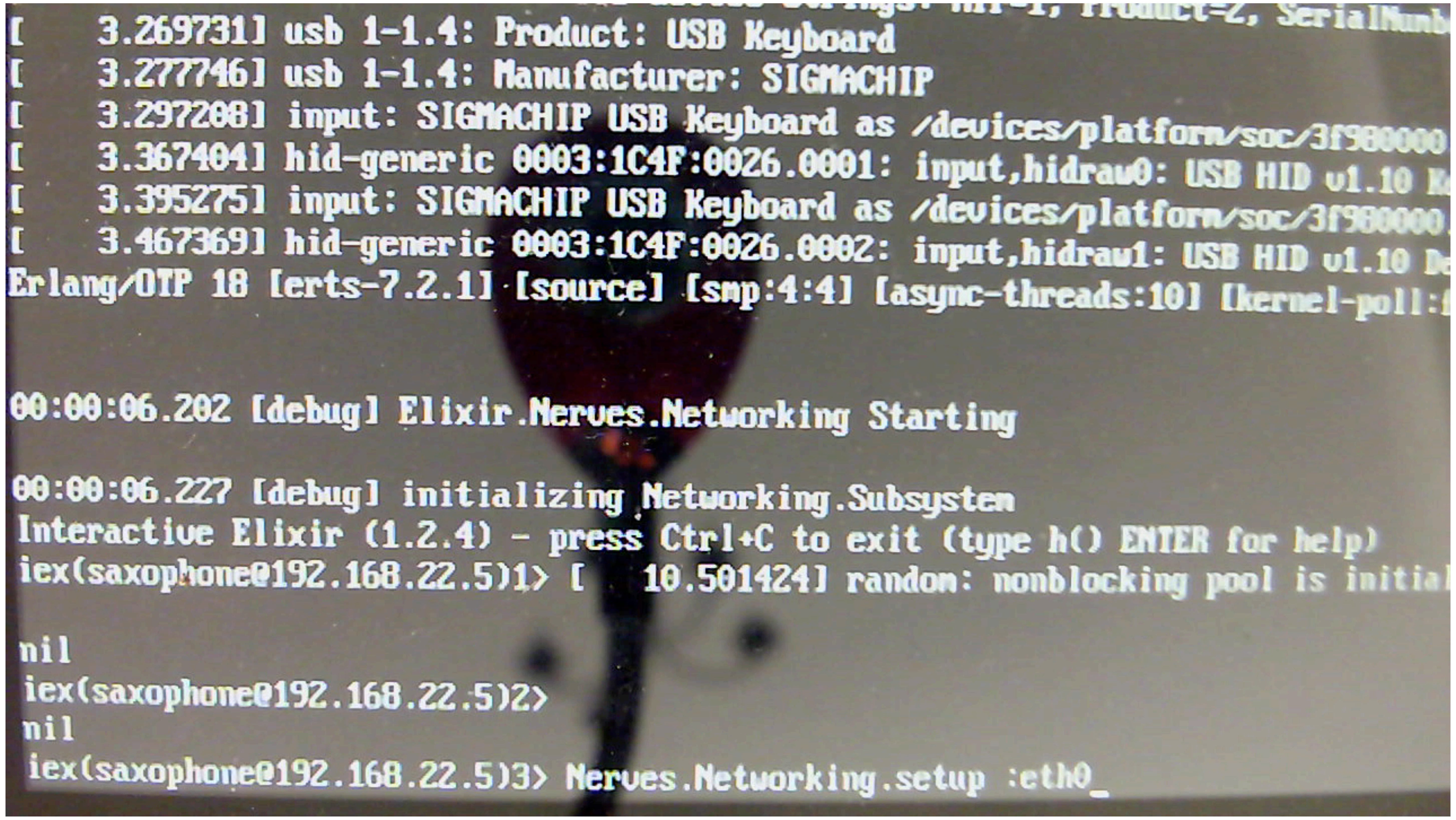
Cultivate



```
[    3.249422] usb 1-1.4: New USB device found, idVendor=1c4f, idProduct=0026
[    3.261366] usb 1-1.4: New USB device strings: Mfr=1, Product=2, SerialNumb
[    3.279776] usb 1-1.4: Product: USB Keyboard
[    3.287816] usb 1-1.4: Manufacturer: SIGMACHIP
[    3.307131] input: SIGMACHIP USB Keyboard as /devices/platform/soc/3f980000
[    3.377354] hid-generic 0003:1C4F:0026.0001: input,hidraw0: USB HID v1.10 Ke
[    3.405172] input: SIGMACHIP USB Keyboard as /devices/platform/soc/3f980000
[    3.477302] hid-generic 0003:1C4F:0026.0002: input,hidraw1: USB HID v1.10 De
Erlang/OTP 18 [erts-7.2.1] [source] [smp:4:4] [async-threads:10] [kernel-poll:f


00:00:06.255 [debug] Elixir.Nerves.Networking Starting

00:00:06.282 [debug] initializing Networking.Subsystem
Interactive Elixir (1.2.4) - press Ctrl+C to exit (type h() ENTER for help)
iex(saxophone@192.168.22.5)1> [   10.381447] random: nonblocking pool is initial

nil
iex(saxophone@192.168.22.5)2> HTTPoison.get "google.com"
```

HTTPoison gotcha

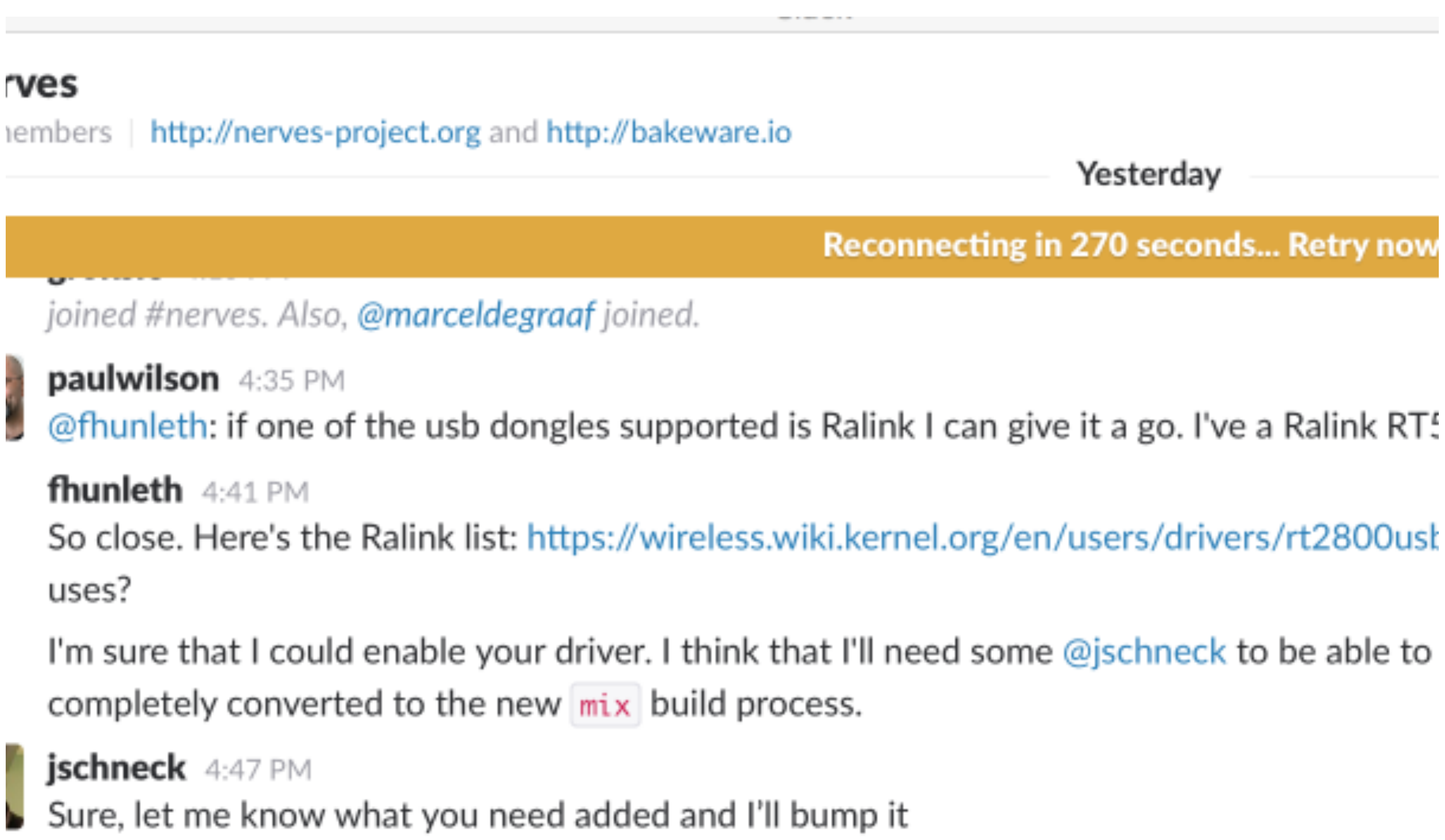
Cultivate



```
                                          Mfr=1, Product=2, SerialNumb
[    3.269731] usb 1-1.4: Product: USB Keyboard
[    3.277746] usb 1-1.4: Manufacturer: SIGMACHIP
[    3.297208] input: SIGMACHIP USB Keyboard as /devices/platform/soc/3f980000
[    3.367404] hid-generic 0003:1C4F:0026.0001: input,hidraw0: USB HID v1.10 K
[    3.395275] input: SIGMACHIP USB Keyboard as /devices/platform/soc/3f980000
[    3.467369] hid-generic 0003:1C4F:0026.0002: input,hidraw1: USB HID v1.10 De
Erlang/OTP 18 [erts-7.2.1] [source] [smp:4:4] [async-threads:10] [kernel-poll:f


00:00:06.202 [debug] Elixir.Nerves.Networking Starting

00:00:06.227 [debug] initializing Networking.Subsystem
Interactive Elixir (1.2.4) - press Ctrl+C to exit (type h() ENTER for help)
iex(saxophone@192.168.22.5)1> [   10.501424] random: nonblocking pool is initial

nil
iex(saxophone@192.168.22.5)2>
nil
iex(saxophone@192.168.22.5)3> Nerves.Networking.setup :eth0_
```

HTTPoison ok

Cultivate


rves
iembers | http://nerves-project.org and http://bakeware.io

Yesterday

Reconnecting in 270 seconds... Retry now

joined #nerves. Also, @marceldegraaf joined.

paulwilson 4:35 PM
@fhunleth: if one of the usb dongles supported is Ralink I can give it a go. I've a Ralink RT

fhunleth 4:41 PM
So close. Here's the Ralink list: https://wireless.wiki.kernel.org/en/users/drivers/rt2800us
uses?

I'm sure that I could enable your driver. I think that I'll need some @jschneck to be able to completely converted to the new `mix` build process.

jschneck 4:47 PM
Sure, let me know what you need added and I'll bump it

Slack connectivity

Cultivate

Ariane 5 Maiden Flight

Flight 501 - 4 June 1996

Cultivate

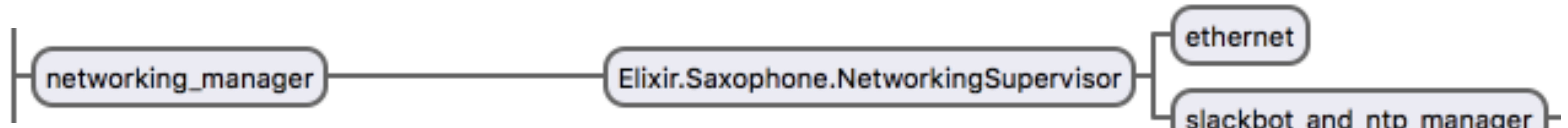
Ariane 5 Failure

- Software error in the Inertial Reference System
- 64 bit to 16 bit caused overflow
- Subsystem crashed entire navigation system
- (Not even needed after takeoff)

Supervision tree considerations

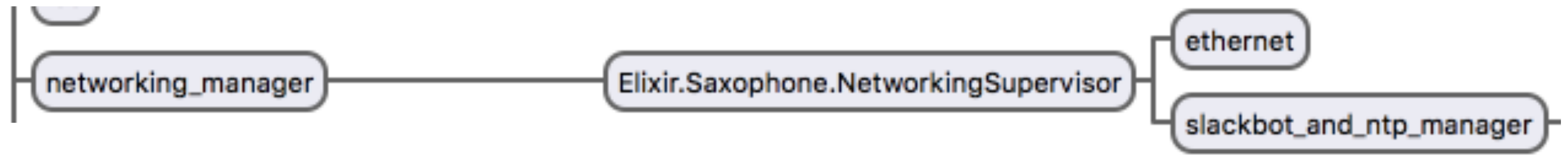
- Ethernet may fail to come up, but we want it to keep trying
- The SlackBot cannot be allowed to try and connect until there is a network connection
- There's no point in connecting to Slack until we've set the time
- SlackBot failure, even continuous, should not bring down the entire application. Just keep retrying.

Cultivate



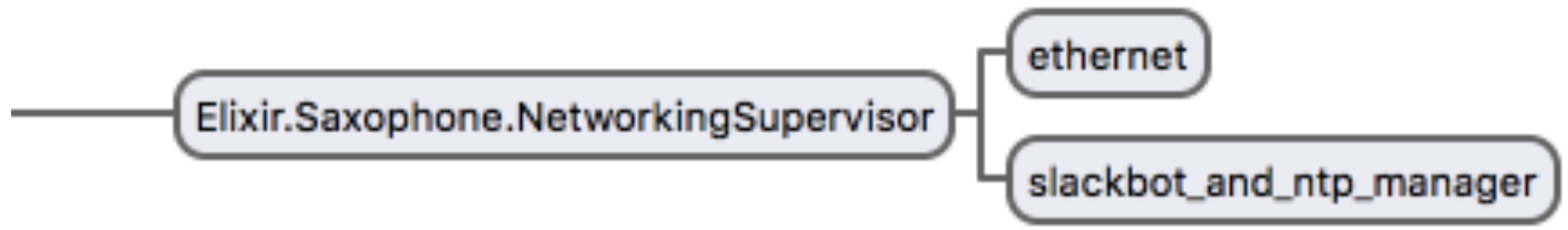
```
def init([]) do
  children = [
    worker(Saxophone.Web.Router, []),
    worker(Gpio, [@led_pin, :output, [name: :led]]),
    worker(Saxophone.Saxophonist, [@sax_pin, @sax_toggle_time, [name: :saxophonist]], id: :s
    worker(Saxophone.Saxophonist, [@guitar_pin, @guitar_toggle_time, [name: :guitarist]], id
    supervisor(Saxophone.LocomotionSupervisor, []),
    worker(Saxophone.GenServerRestarter, [Saxophone.NetworkingSupervisor, :start_link, [],
           @ethernet_retry_time, [name: :networking_manager]]),
    ]
  supervise(children, strategy: :one_for_one)
end
```

Cultivate

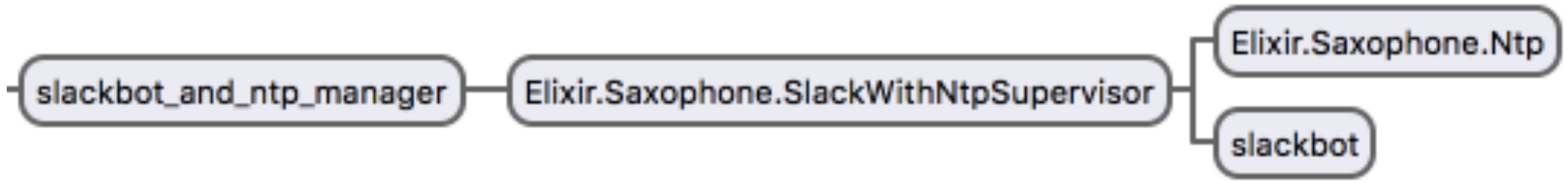


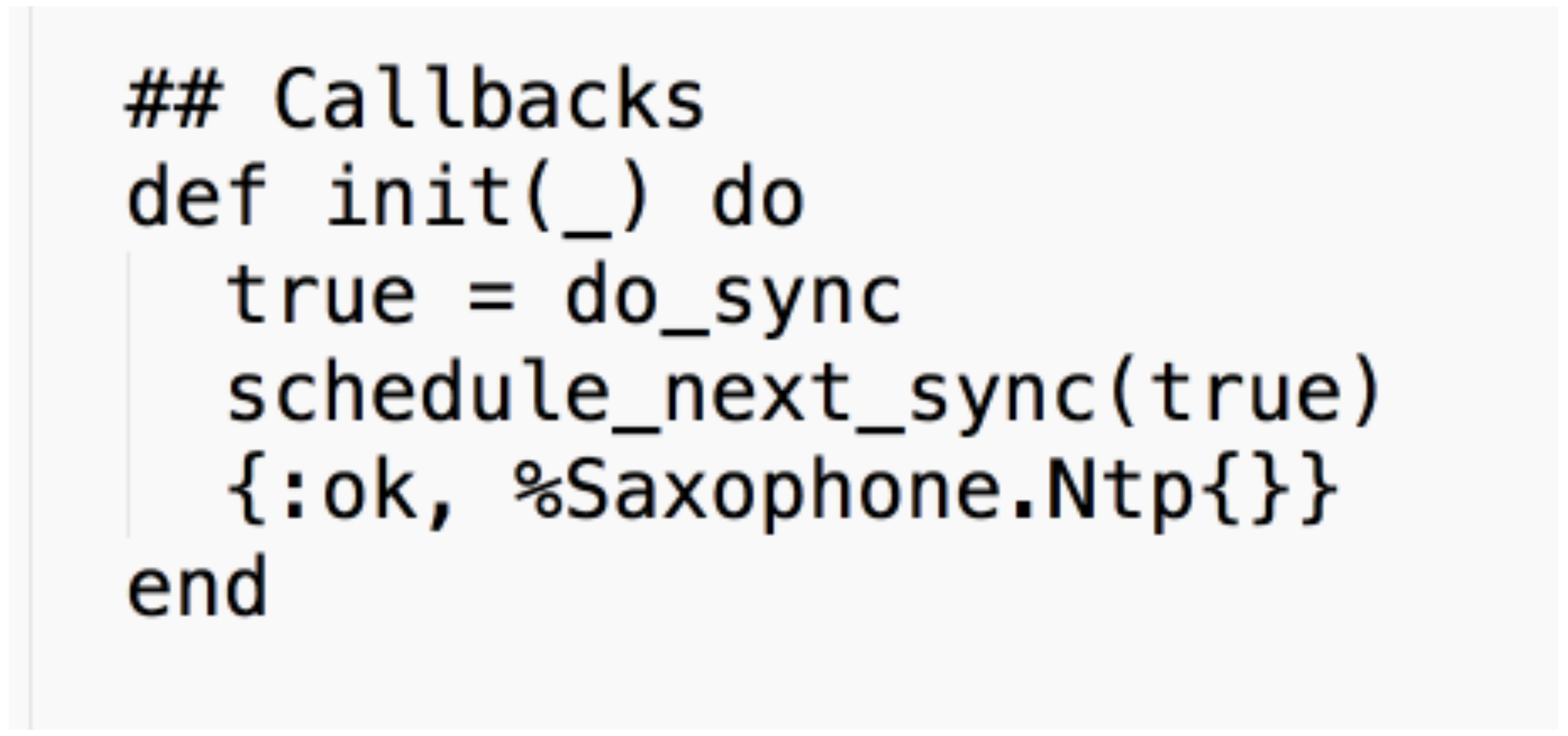
```
defmodule Saxophone.GenServerRestarter do
  use GenServer

  def start_link(module, function, args, retry_interval, restarter_otp_opts \\ [], start_
    GenServer.start_link(__MODULE__,
                          {%{retry_interval: retry_interval,
                            module: module,
                            function: function,
                            args: args}, start_delay},
                          restarter_otp_opts)
  end

  def init({status, start_delay}) do
    Process.send_after(self, :start, start_delay)
    Process.flag(:trap_exit, true)
    {:ok, status}
  end

  def handle_info(:start, state = %{module: module, function: function, args: args}) do
    {:ok, pid} = apply(module, function, args)
    Process.link(pid)
    {:noreply, state}
  end

  def handle_info({:EXIT, _pid, _reason}, status = %{retry_interval: retry_interval}) do
    Process.send_after(self, :start, retry_interval)
    {:noreply, status}
  end
end
```

Cultivate



```
defmodule Saxophone.NetworkingSupervisor do
  use Supervisor

  @ethernet_opts Application.get_env(:saxophone, :ethernet_opts) || []

  @slackbot_retry_time Application.get_env(:saxophone, :slackbot_retry_seconds) |> :timer.seconds
  @slackbot_start_delay Application.get_env(:saxophone, :slackbot_start_delay_seconds) |> :timer.seconds

  def start_link do
    Supervisor.start_link(__MODULE__, [], name: __MODULE__)
  end

  def init(_) do
    children = [
      worker(Nerves.Networking, [:eth0, @ethernet_opts], function: :setup),
      worker(Saxophone.GenServerRestarter, [Saxophone.SlackWithNtpSupervisor,
                                            :start_link,
                                            [],
                                            @slackbot_retry_time,
                                            [name: :slackbot_and_ntp_manager],
                                            @slackbot_start_delay])
    ]

    supervise(children, strategy: :rest_for_one)
  end
end
```

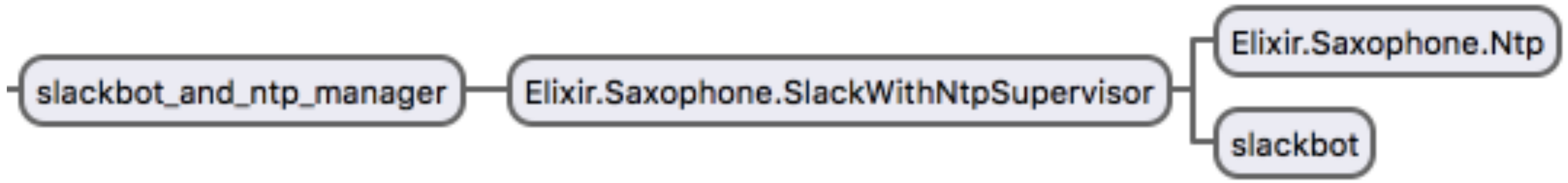




```
## Callbacks
def init(_) do
  true = do_sync
  schedule_next_sync(true)
  {:ok, %Saxophone.Ntp{}}
end
```

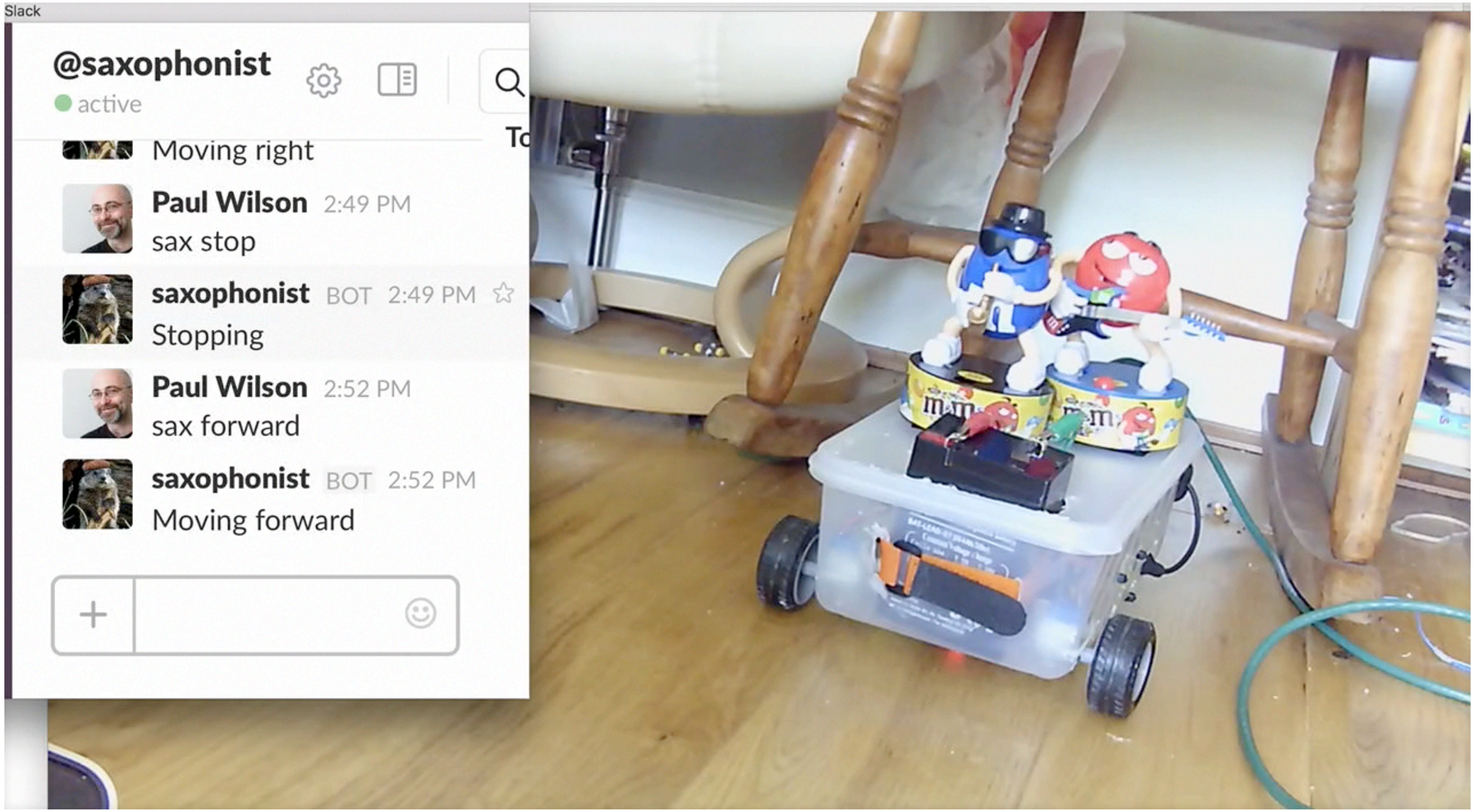
Elixir.Saxophone.Ntp

Cultivate



```elixir
defmodule Saxophone.SlackWithNtpSupervisor do
  use Supervisor

  @slackbot_token  Application.get_env(:saxophone, :slackbot_token)

  def start_link do
    Supervisor.start_link(__MODULE__, [], name: __MODULE__)
  end

  def init(_) do
    children = [
      worker(Saxophone.Ntp, []),
      worker(Saxophone.SlackBot, [@slackbot_token, [name: :slackbot]]),
    ]

    supervise(children, strategy: :rest_for_one)
  end
end

```

Cultivate



Slack

@saxophonist
active

Moving right

Paul Wilson 2:49 PM
sax stop

saxophonist BOT 2:49 PM
Stopping

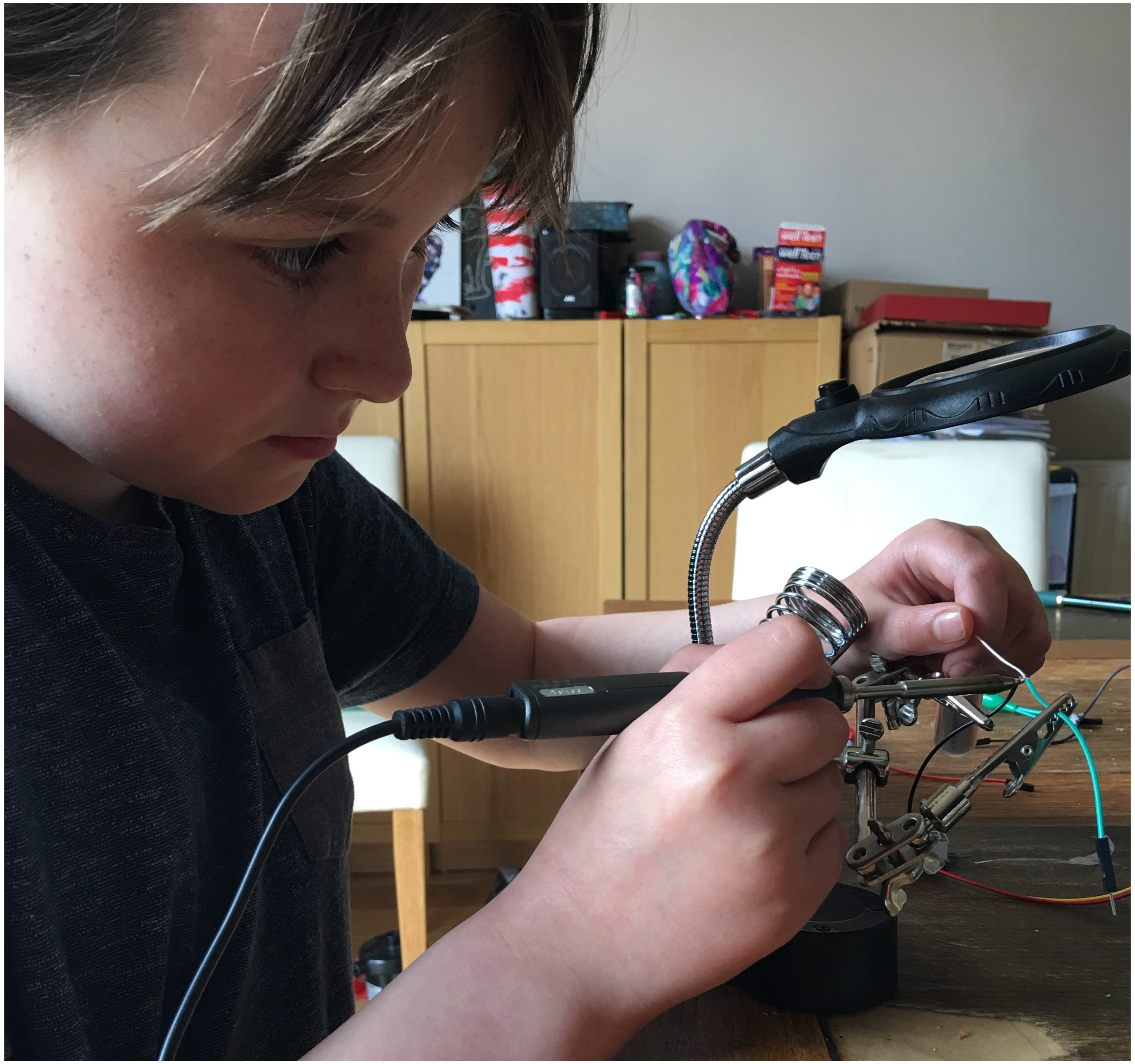
Paul Wilson 2:52 PM
sax forward

saxophonist BOT 2:52 PM
Moving forward

Slack bot

Cultivate

Outsourcing the soldering



Cultivate

More information

- Justin Schneck's keynote at Elixir Conf EU 2016, about an hour ago. Remember?
- http://nerves-project.org
- https://github.com/nerves-project
- https://github.com/paulanthonywilson/saxophone
- Wendy Smoak's Cat Feeder http://wsmoak.net/2016/04/03/cat-feeder-fabrication.html
- Nerves channel on Elixir Slack https://elixir-lang.slack.com/archives/nerves
- http://www.cultivatehq.com/posts/ (soon)

Cultivate