Elixir: A Whirlwind Tour

by @colinfdrake

Overview

- Motivation - History, Backstory, etc.
- Elixir Overview
- OTP Overview
- Other Cool Stuff ™

Motivation

What is Elixir?

- A Ruby-inspired "[...] **dynamic**, **functional** language designed for building scalable and maintainable applications"
- Compiled, targeting the Erlang virtual machine

Erlang

- A functional language designed by Ericsson in 1986
- Open sourced as **Open Telecom Platform** (OTP) in 1998
- Influenced by Prolog, Smalltalk, CSP
- Designed for "distributed, fault-tolerant, [...], highly available, non-stop applications"

Erlang Features

- Simple, process-based concurrency ("actor model")
  - Higher-level than OS threads/processes
  - Very lightweight
  - Share *no* state
- Dynamic software updating
- Fault tolerant - *"Let it crash"* philosophy

Erlang IRL

- Amazon SimpleDB (AWS)
- Facebook Messenger
- WhatsApp
- T-Mobile SMS network
- Pinterest (deployed alongside Elixir!)

Erlang

```
X=person (address ("Big street", 23),
          telno ([1, 2, 4, 6, 7, 9]))
```

```
{http_header, _, Hdr, _, Val} ->
    {headers, Hdrs} = lists:keyfind(headers, 1, Req),
    serve(S, Handler,
          lists:keystore(headers, 1, Req,
                         {headers, [{Hdr,Val}|Hdrs]}));
http_eoh ->
    ok = inet:setopts(S, [{packet, raw}]),
    {Status, Hdrs, Resp} =
        try Handler(S, Req)
        catch _:_ -> {500, [], <<>>} end,
    ok = gen_tcp:send(
           S, ["HTTP/1.0 ", integer_to_list(Status), "\r\n",
               [[H, ": ", V, "\r\n"] || {H,V} <- Hdrs],
               "\r\n", Resp]),
    gen_tcp:close(S);
{http_error, Error} ->
    exit(Error);
ok ->
    ok
```

I knew Erlang
before it was Cool*

(* it was never cool)

Elixir Overview

What makes Elixir different?

- Able to target a proven VM with a "simpler" language
- Adds macros, pipelines, protocol-oriented design, ...
- Excellent open source ecosystem
  - Plug, Phoenix, Ecto, HTTPoison, ...

Basics

```elixir
1.0                          # numbers
"Strings"                    # strings
:atoms                       # atoms/keywords
{:ok, "some data"}           # tuples
%{a: "b", c: "d"}            # maps/structs

x = 1                                          # variables
{status, value} = {:ok, "some data"}           # destructuring

# functions, modules, etc.
```

Pattern Matching

```
res = some_unsafe_function

case res do
  {:ok, value} ->
    IO.puts value
  {:error, _} ->
    IO.puts "Uh oh!"
end

# one of *many* cases of pattern matching in the language
```

Pattern Matching

```
# I lied... "=" doesn't mean "assign". It means "match".

x = 1

1 = x
[x, y] = [1, 2]     # 👍

2 = x     # ** (MatchError) no match of right hand side value: 1
          # ** This is a crash!
```

Match Refactoring

```
{status, result} = call_some_unsafe_service

if status == :ok do
  result |> do_cool_stuff |> and_more_stuff
else
  perform_some_fatal_error
end

# Let's rewrite this with pattern matching...
```

Match Refactoring - "Let it Fail"

```
{:ok, result} = call_some_unsafe_service
result |> do_cool_stuff |> and_more_stuff

# Sure... but what happens if doesn't match to :ok?
```

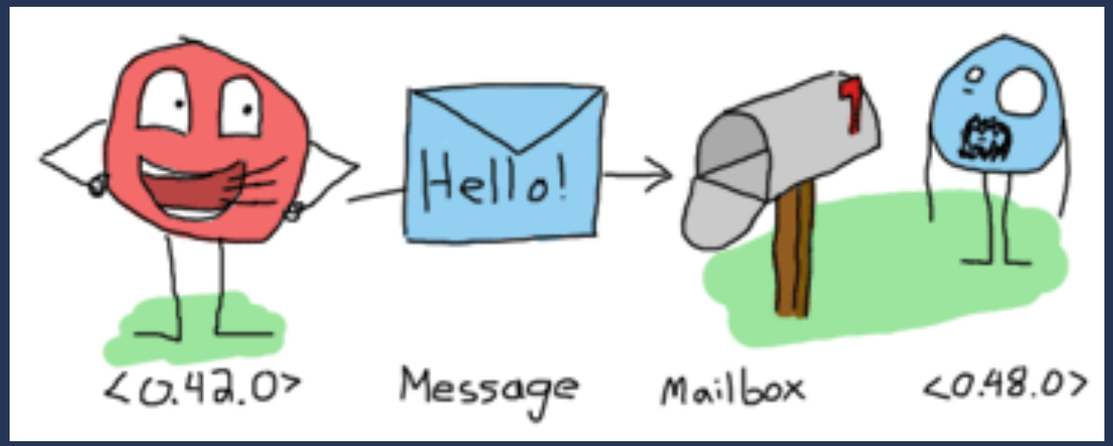

YOLO

OTP Overview

Actors, Agents, Supervisors, ...oh my!

Processes

- Break out app into processes
  - Contains completely *localized, independent* state
  - Has a mailbox, a queue of messages to respond to
  - Can asynchronously send messages to other mailboxes



Processes - Example Code

```
parent = self()  # PID

# Create a child process...
spawn_link(fn ->
  send parent, {:msg, "hello world"}
end)

# Listen to messages sent to current process...
receive do
  {:msg, contents} -> IO.puts contents
end
```

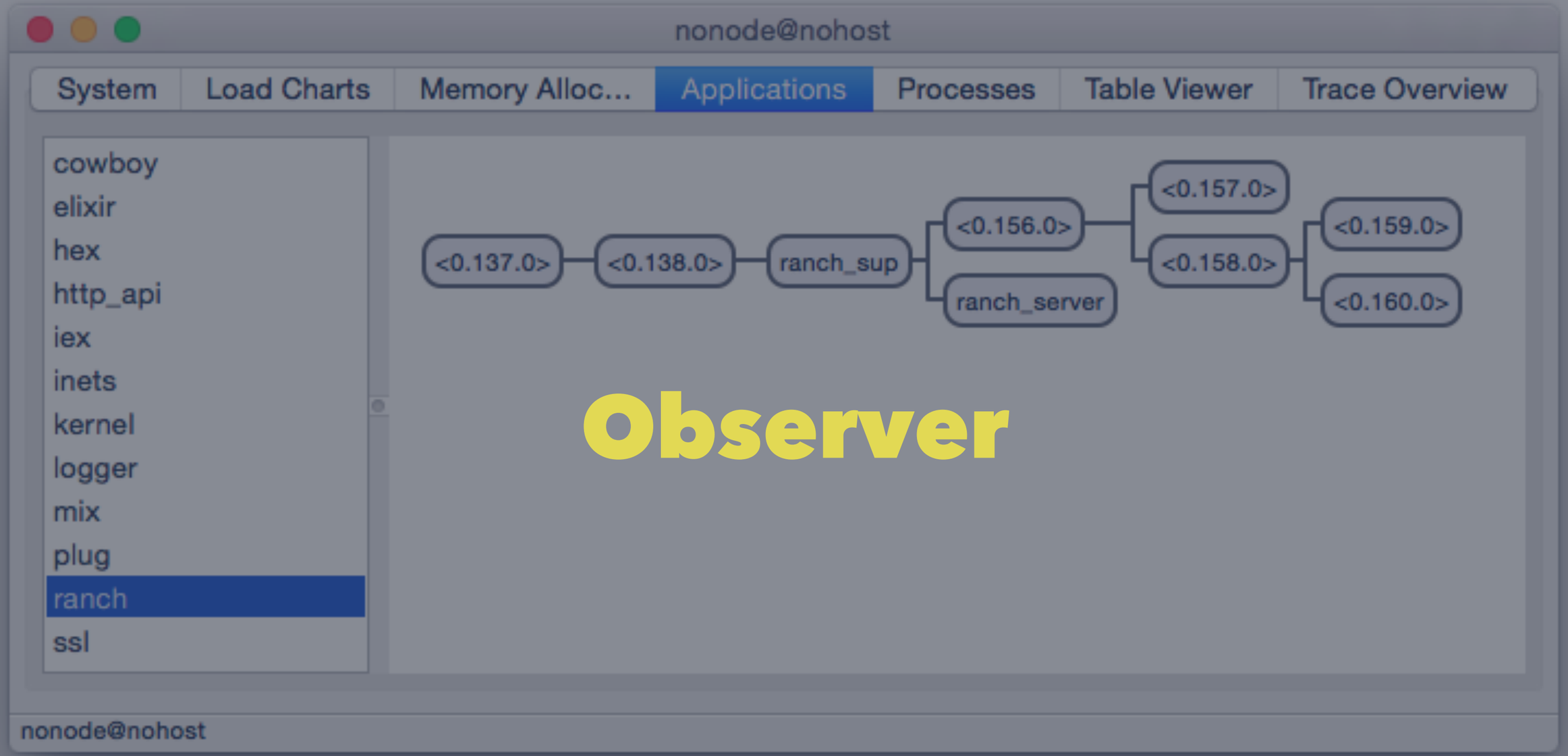
Fault Tolerance: Supervisors

```
import Supervisor.Spec

children = [
  worker(...),
  worker(...),
  supervisor(...)  # could be a tree of processes!
]

Supervisor.start_link(children, strategy: :one_for_one)
```

Other Cool Stuff ™


nonode@nohost
System
Load Charts
Memory Alloc...
Applications
Processes
Table Viewer
Trace Overview
cowboy
elixir
hex
http_api
iex
inets
kernel
logger
mix
plug
ranch
ssl
<0.137.0>
<0.138.0>
ranch_sup
<0.156.0>
ranch_server
<0.157.0>
<0.158.0>
<0.159.0>
<0.160.0>
Observer
nonode@nohost

```elixir
defmodule Chatty.Channels.Rooms do
  use Phoenix.Channel

  def join(socket, "lobby", message) do
    broadcast socket, "user:entered", username: message["message"] || "anon"
    {:ok, socket}
  end

  def join(socket, private_topic, message) do
    {:error, socket, :unauthorized}
  end

  def event("new:message", socket, message) do
    broadcast socket, "new:message", content: message["content"],
                                     username: message["username"]

    socket
  end

end
```

Phoenix

elixir

elixir

elixir

elixir

Questions?