Sweet Elixir!

Ryan Cromwell

SPARKBOX

Pipeline

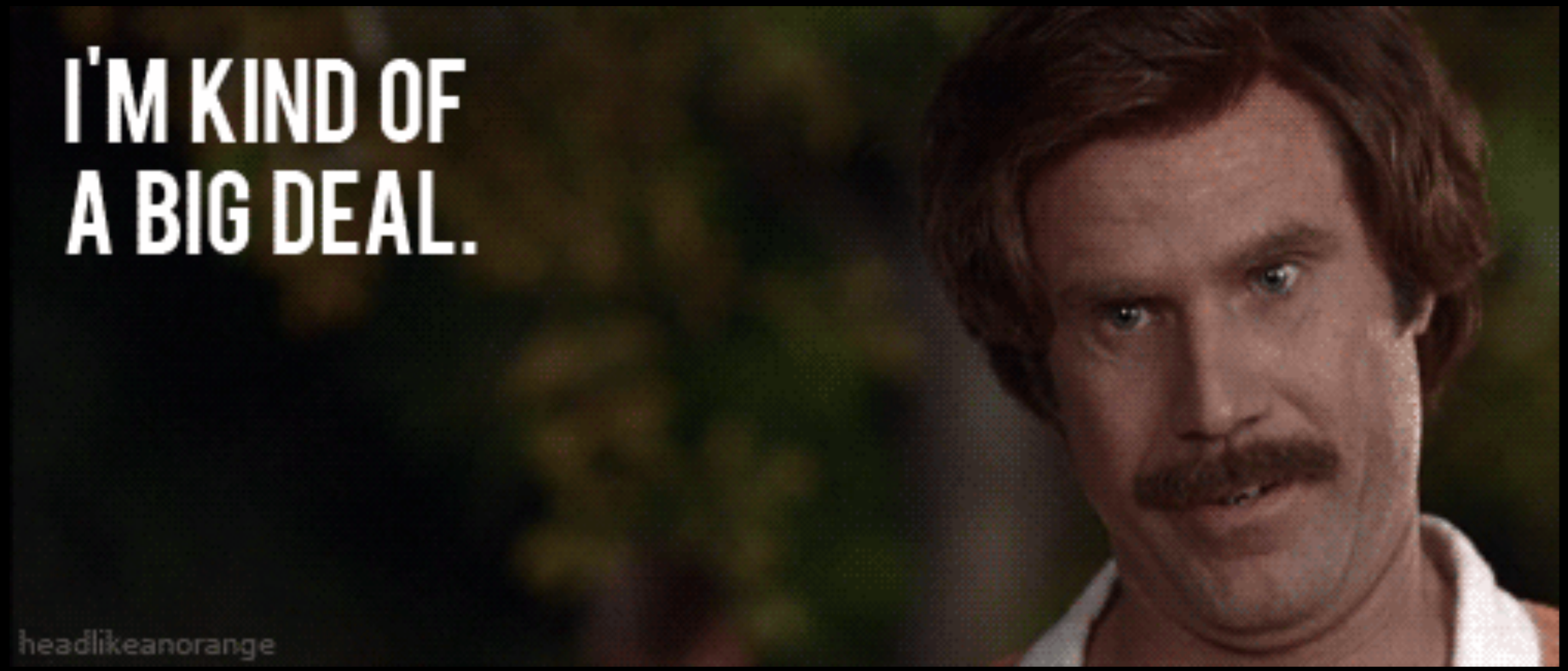
What's Elixir?

- |> Types, Functions, Modules
- |> Pattern Matching
- |> Maps, Structs, & Protocols
- |> Pipelines
- |> Processes
- |> OTP, Phoenix

What's Elixir?

Sweet Elixir!

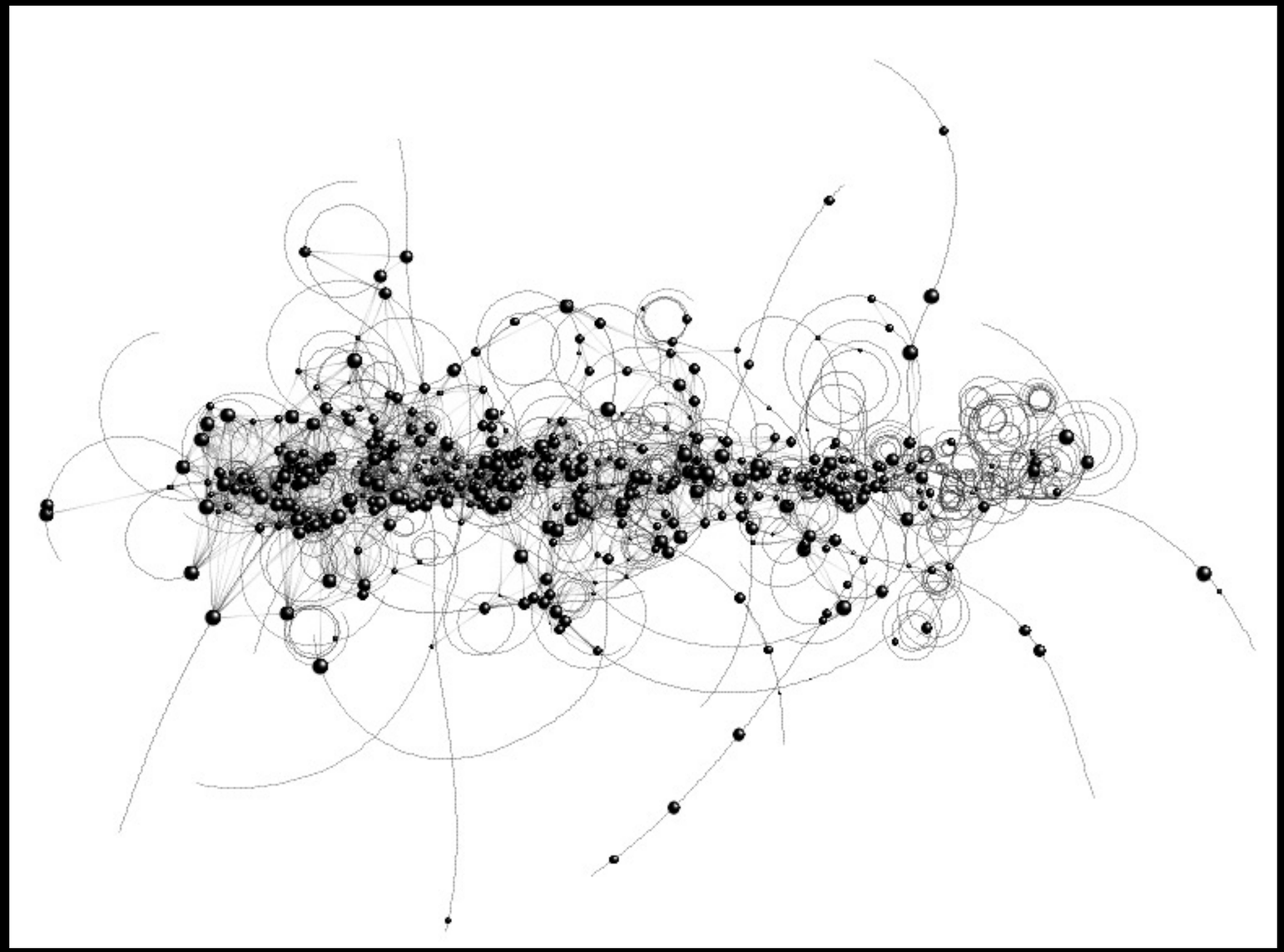
Functional



Erlang VM

Concurrent

Distributed



Setup

Sweet Elixir!

```
#Mac
$> brew update
$> brew install erlang
$> brew install elixir

#Linux
<Your package manager here>

#Windows
PS> cinst elixir
```

Mac

SECTION TITLE

SWEET ELIXIR!

```
$> iex

iex(1)> IO.puts “hello world!”
```

```
$> elixir hello_world.exs
hello world
```

```
elixirc hello_world.ex

$> ls
Elixir.HelloWorld.beam
```

```
$> mix deps.get
$> mix compile
$> mix phoenix.start
```

hex.pm

Types

Sweet Elixir!

```
is_number 1                    #=> true

is_integer 1                   #=> true

is_number 2.1                  #=> true

is_float 2.1                   #=> true

is_integer 2.1                 #=> false
```

```
1 == 1.0                    #=> true

1 === 1.0                   #=> false
```

```
is_boolean true                 #=> true

true or false                   #=> true

true || false                   #=> true
```

```
is_atom :selfConf                    #=> true

is_atom true                         #=> true
```

```
is_list [1,2,3]                 #=> true

length [1,2,3]                  #=> 3

[1,2] ++ [3,4]                  #=> [1,2,3,4]

iex> h Enum

iex> h Stream
```

```
is_tuple {1, “b”, :c}                  #=> true

elem {1, “b”, :c}, 1                   #=> “b”

t = {1, “b”, :c}
set_elem t, 2, :d                      #=> {1, “b”, :d}

Integer.parse “1.0a3”                  #=> {1, “.0a3”}
```

```
is_list ‘hello’                    #=> true

is_list “hello”                    #=> true

is_binary “hello”                  #=> true
```

```
“hello” <> “ world”                    #=> “hello world”

who = “ryan”
“hello #{who}”                         #=> “hello ryan”
```

Anonymous Functions

SWEET ELIXIR!

```
add = fn (x, y) -> x + y end

add.(2,2)                    #=> 4
```

```
do_calc = fn(x,y, calc) -> calc.(x,y) end

do_calc.(2,3, add)                  #=> 5

do_calc.(2,4, &(&1 * &2))           #=> 8
```

Modules

SWEET ELIXIR!

```elixir
defmodule Weather do
  def celsius_to_fahrenheit(celsius) do
    (celsius * 1.8) + 32
  end

  def boiling, do: 100
  def freezing, do: 0
end

Weather.celsius_to_fahrenheit(20)   #=> 68.0
```

```
defmodule USWeather do
  import Weather, only: [freezing: 0,
      celsius_to_fahrenheit: 1]

  alias :math, as Math

  def cold_in_michigan do
    celsius_to_fahrenheit(freezing - 10)
  end
end
```

import
alias

Pattern Matching

SWEET ELIXIR!

```
[head | tail] = [1,2,3,4,5]

head                                    #=> 1

tail                                    #=> [2,3,4,5]
```

```
{a,b,c}= {1,”b”,:c}

a                                   #=> 1

b                                   #=> “b”

c                                   #=> :c
```

```
{a,b,:other} = {1,"b",:c}
```

```
{a,b,:other} = {1,”b”,:c}

** (MatchError) no match of right hand side value: {1, "b", :c}
```

```
calculate = fn expression ->
  case expression do
    {:+, num1, num2} -> num1 + num2
    {:-, num1, num2} -> num1 - num2
    {:*, num1, 0}     -> 0
    {:*, num1, num2} -> num1 * num2
  end
end
```

Control Flow

```
defmodule Countdown do
  def run(from, to) when from >= to do
    run(from, to, from)
  end

  def run(_from, to, current) when to == current do
    IO.puts to
    IO.puts "Done!"
  end

  def run(from, to, current) do
    IO.puts current
    run(from, to, current - 1)
  end
end
```

Function

```
defmodule PersonPrefixer do
  def prefix(p = %{ gender: :male }), do: "Mr."
  def prefix(p = %{ gender: :female }), do: "Mrs."
  def prefix(p), do: ""
end

PersonPrefixer.prefix %{first: “Sandy”, gender: :female}  #=> “Mrs”
```

Maps
Structs
Protocols

SWEET ELIXIR!

Maps

```
speaker = %{
  first: “Ryan”,
  last: “Cromwell”,
  twitter: “@cromwellryan”,
  “home town” => “Dayton, OH”
}

speaker[:twitter] #=> “@cromwellryan”

speaker[“Home Town”] #=> “Dayton, OH”
```

Structs

```
defmodule Kid do
  defstruct name: "ryan", age: 7
end
```

Protocols

```
defprotocol Good do
  @doc "Returns true if data is considered good"
  def good?(data)
end
```

Ad-hoc Polymorphism

```
defimpl Good, for: Kid do
  def good?(%Kid{name: "Ryan"}), do: true
  def good?(_), do: false
end

Good.good? %Kid{name: "Ryan"}                    #> true
Good.good? %Kid{name: "Ben"}                     #> false
```

Pipelines

SWEET ELIXIR!

```
Enum.map [1,2,3,4], fn (x) -> x * 2              #=> [2,4,6,8]

[1,2,3,4] |> Enum.map(fn (x) -> x *2)          #=> [2,4,6,8]
```

before

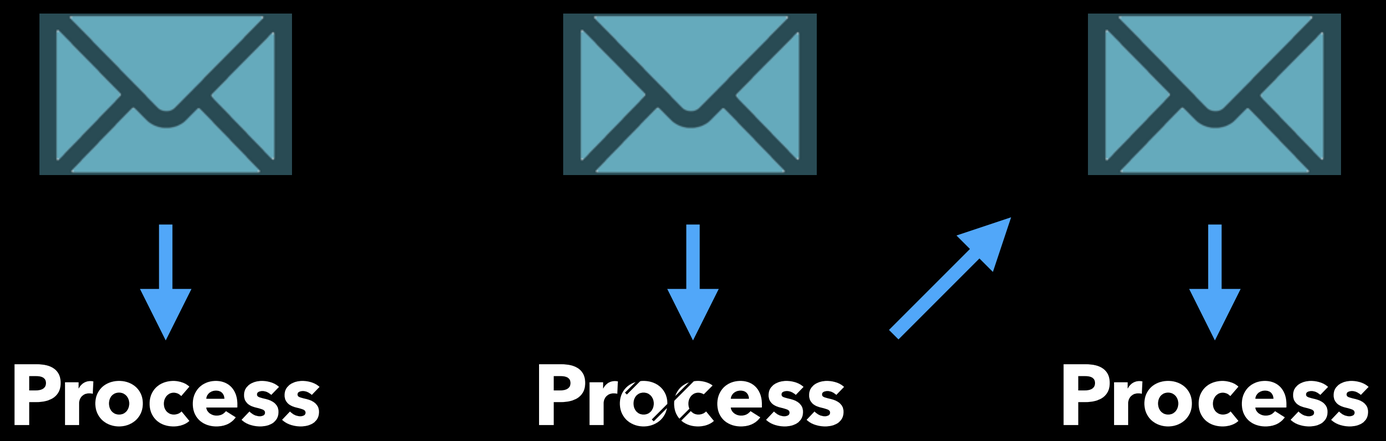
```
lines = String.split file_content, "\n"
lines = Stream.filter lines, &(String.length(&1) > 0)
lengths = Stream.map lines, &(String.split &1)
triangles = Stream.map lengths, &(list_to_tuple &1)
classifications = Stream.map triangles, &( %{sides: &1,
classification: Classifier.classify &1} )

messages = Stream.map classifications, &( "Triangle #{inspect
&1[:sides]} is #{&1[:classification]}" )
result = Enum.join(messages, "\n")

IO.puts result
```

after

```
file_content
   |> String.split("\n")
   |> Stream.filter( &(String.length(&1) > 0) )
   |> Stream.map( &(String.split &1) )
   |> Stream.map( &(list_to_tuple &1) )
   |> Stream.map( &(%{sides: &1, class: Classifier.classify &1}) )
   |> Stream.map( &("Triangle #{inspect &1[:sides]} is #{&1[:class]}")
)
   |> Enum.join("\n")
   |> IO.puts
```

Processes

SWEET ELIXIR!


Process
Process
Process

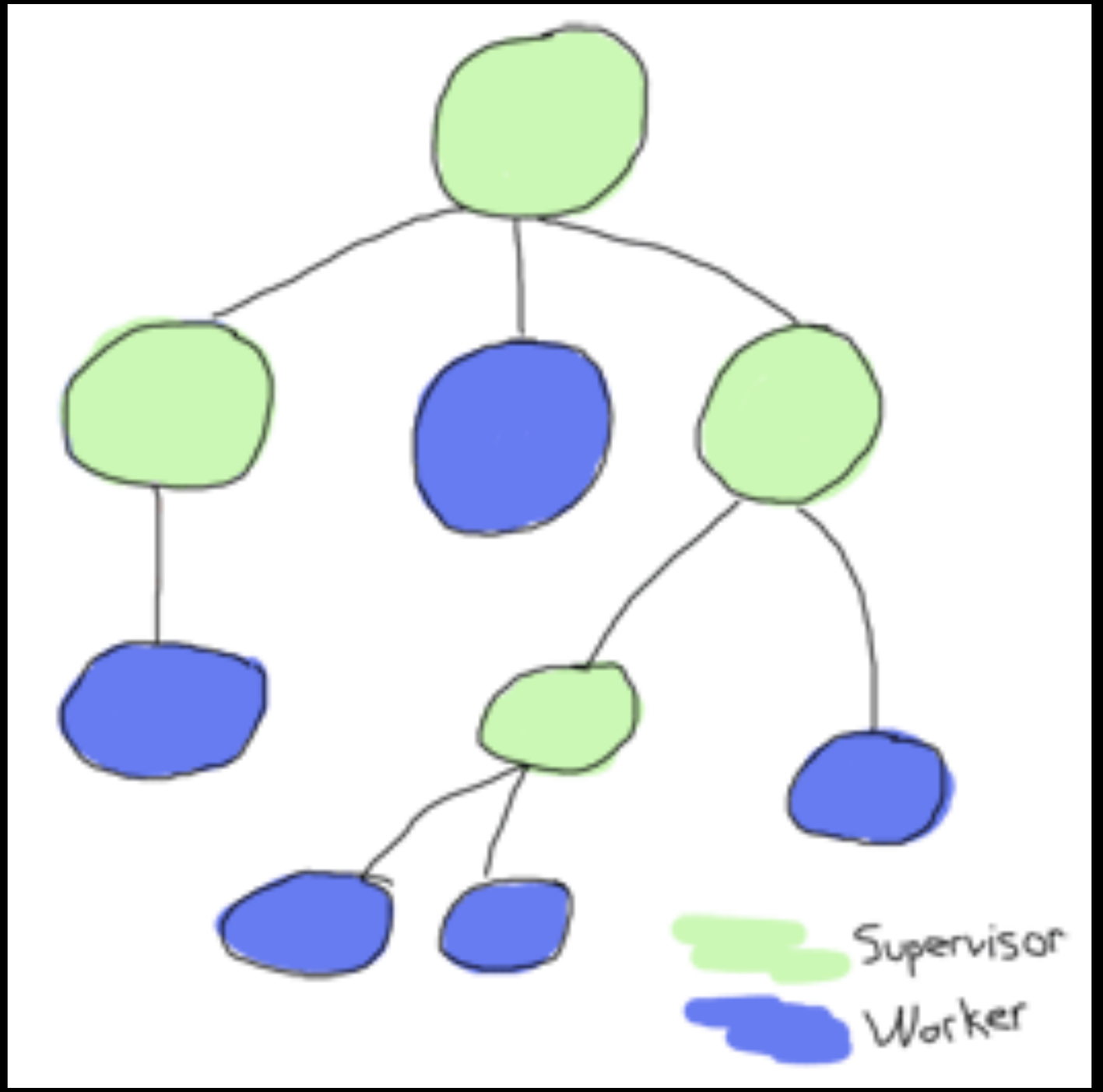
```
pid = spawn fn ->
  receive do
    {sender, :ping} ->
      IO.puts "Got ping"
      send sender, :pong
  end
end
```

Live Coding

OTP

SWEET ELIXIR!

- Supervisor
- GenServer
- GenFSM
- GenEvent
- Application


Supervisor
Worker

phoenix

SWEET ELIXIR!

github.com/phoenixframework/phoenix

```
$> mix phoenix.new your_app /path/to/scaffold/your_app

$> mix do deps.get, compile

$> mix phoenix.start
```

Inspired by Rails

```
defmodule ChatDemoEx.Router do
  use Phoenix.Router
  use Phoenix.Router.Socket, mount: "/ws"

  plug Plug.Static, at: "/static", from: :chat_demo_ex
  get "/", ChatDemoEx.Controllers.Pages, :index, as: :page

  channel "rooms", Chat.Channels.Rooms
end
```

Elixir

```
defmodule Chat.Channels.Rooms do
  use Phoenix.Channel

  def event(socket, "new:message", message) do
    broadcast socket, "new:message", message

    socket
  end
end
```

Javascript

```
chan.send("new:message", {
  username: “cromwellryan”,
  body: “hello internets!”
});
```

LEARN YOU SOME ELIXIR

SWEET ELIXIR!

Resources

- elixir-lang.org
- elixirsips.com
- elixirconf.com (July 25-26)

- PragProg: Programming Elixir

- github.com/cromwellryan/sweetelixir
- http://bit.ly/sweetelixir-short

THANKS!

@cromwellryan

ryan@heysparkbox.com