Modeling the real world with Elixir and OTP

Aish Raj Dahal

pagerduty

Why are we here?

- Concurrency, Events and the Real World
- Case study of an Elixir use
- Tour of the BEAM and OTP
- Demo time!

Epilogue

How did we get here ?

“Can programming be liberated from the von Neumann style” ?

Elixir is a programming language that leverages the power of Erlang VM (BEAM)

With a simple syntax and a supportive community the language has seen tremendous growth

A lot* of PagerDuty is powered by Elixir

It has always not been this way.

At PagerDuty it started as a language of choice for Rails-native way to talk to Kafka

Why ?

It was high value

It was easy to redo it in a different language

How was it ?

It was not all rainbows and unicorns

The application needed some tuning

But language never got in the way

Since then…

A cambrian explosion in Elixir based services

A good number of critical backend applications are written in Elixir

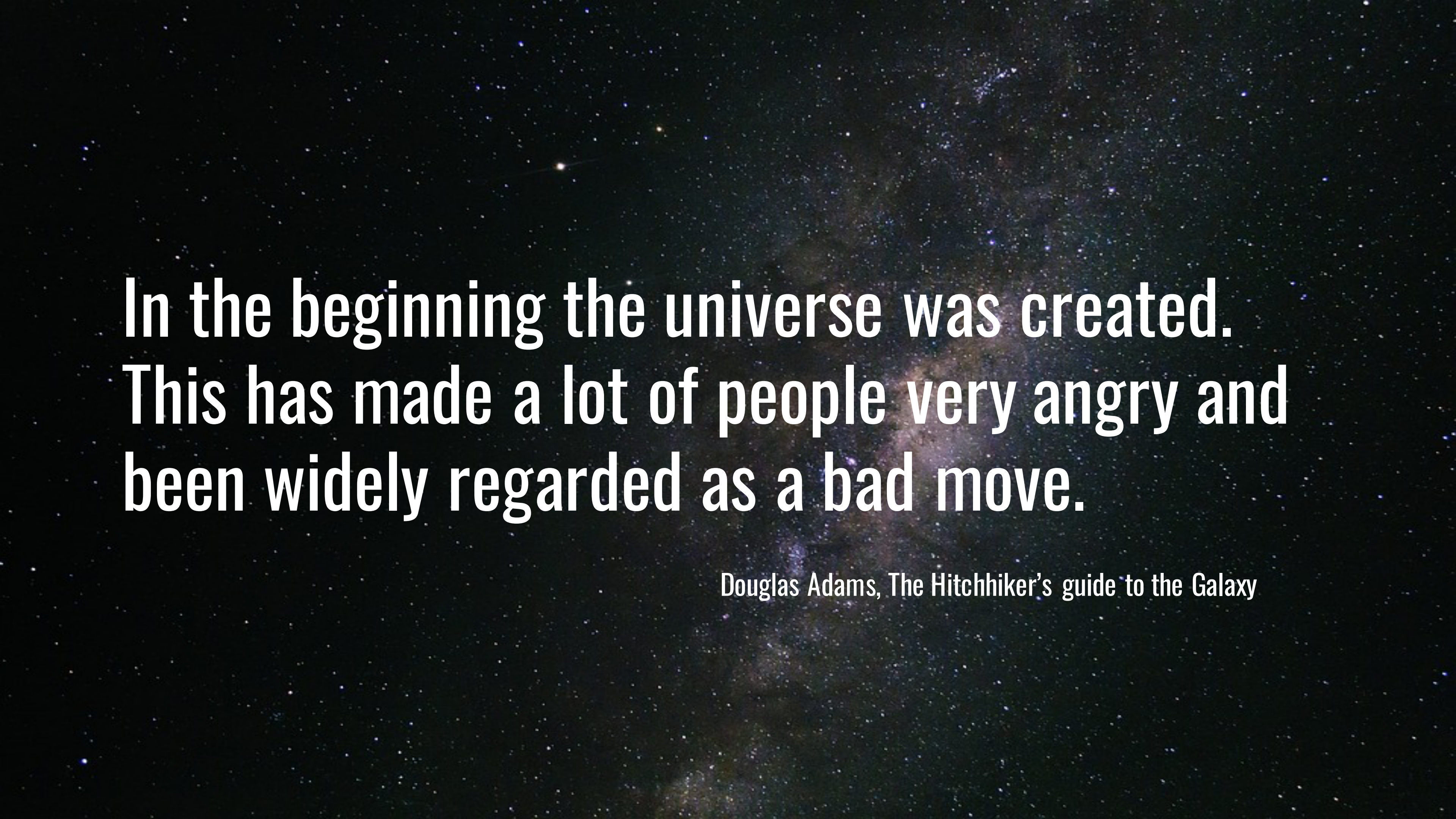
All with the niceness of being built for the Real-World ™

Chapter I

The Real World

What is the Real World™?

Are we talking about
the universe ?



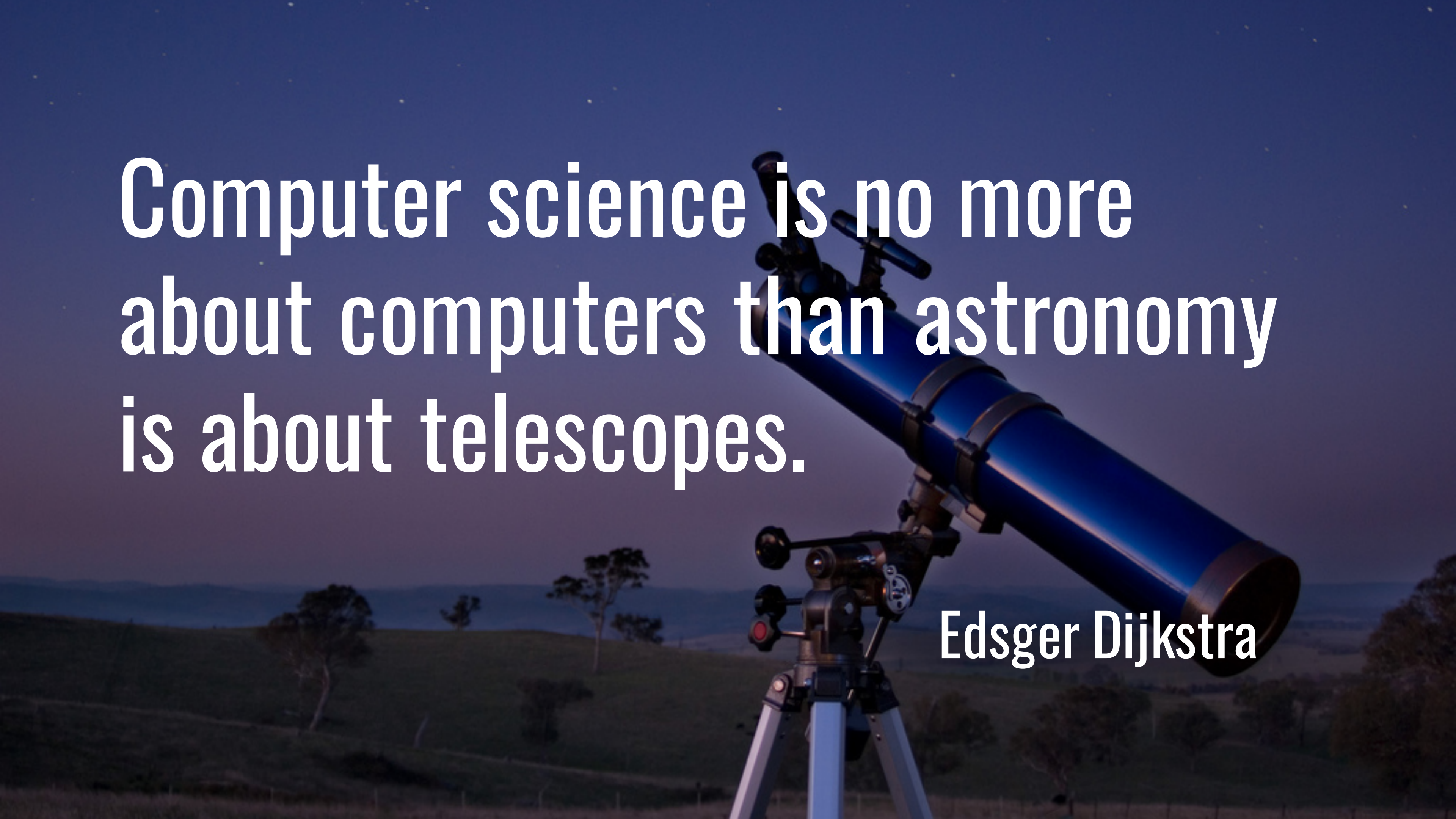
In the beginning the universe was created. This has made a lot of people very angry and been widely regarded as a bad move.

Douglas Adams, The Hitchhiker’s guide to the Galaxy

The Real World ™ is the world we live in

You must care about it because....

When was the last time you programmed a non-multi-core computer ?



Computer science is no more about computers than astronomy is about telescopes.

Edsger Dijkstra

Software you write probably models a real world situation

Software you write probably deals with events

Mental model of the real world:

It is inherently concurrent

It is event-based

Failures are unpredictable

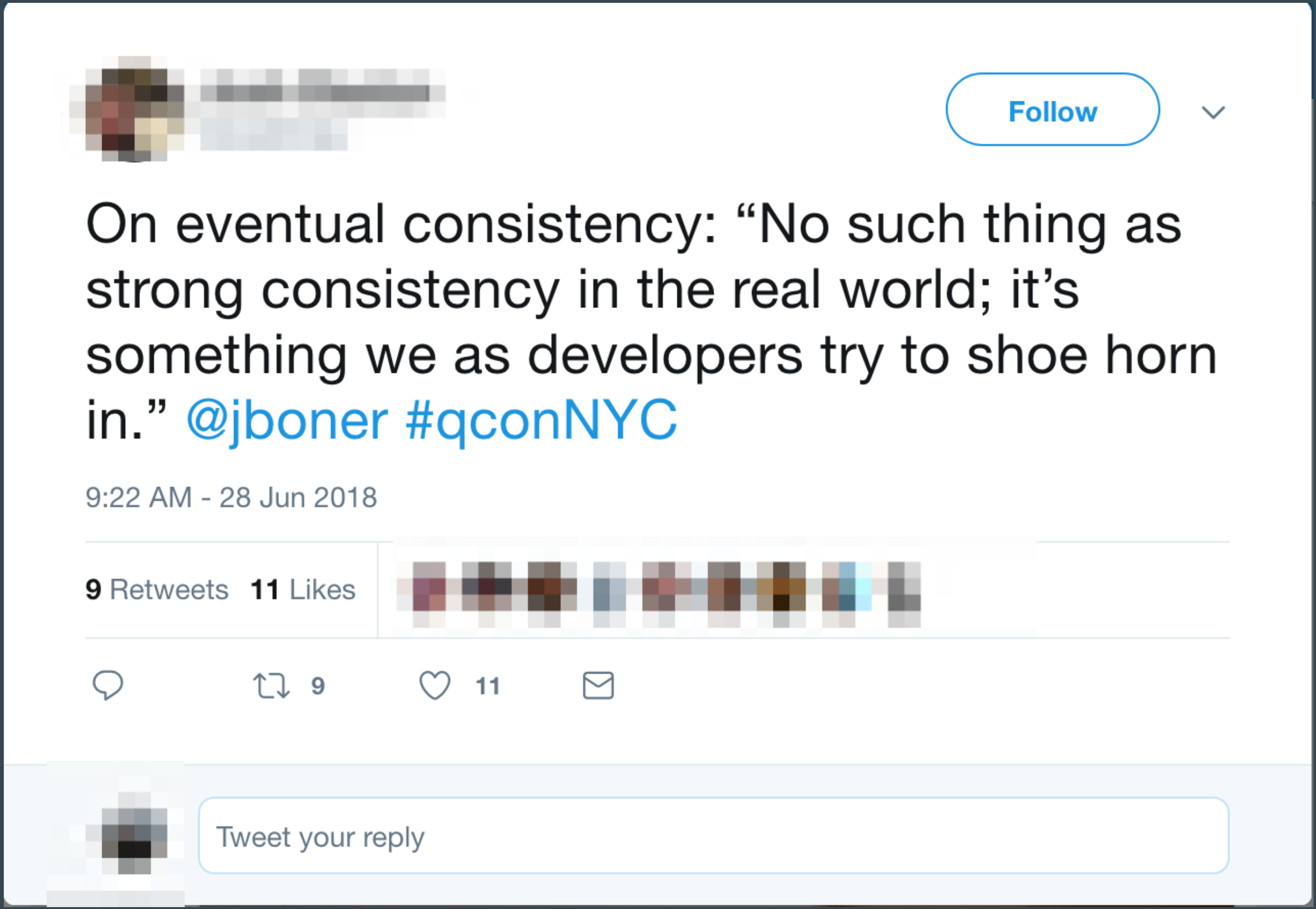
It tends to be “real time”

Software you write for the Real World ™ should deal with these situations.

Which means…

Your software should be concurrent.

Your software should deal with events



Follow

On eventual consistency: “No such thing as strong consistency in the real world; it’s something we as developers try to shoe horn in.” @jboner #qconNYC

9:22 AM - 28 Jun 2018

9 Retweets **11** Likes

9 11

Tweet your reply

Your software should be fault-tolerant.

Your software should respond to the user-input within a reasonable* time.

Erlang solved these problems around three decades ago

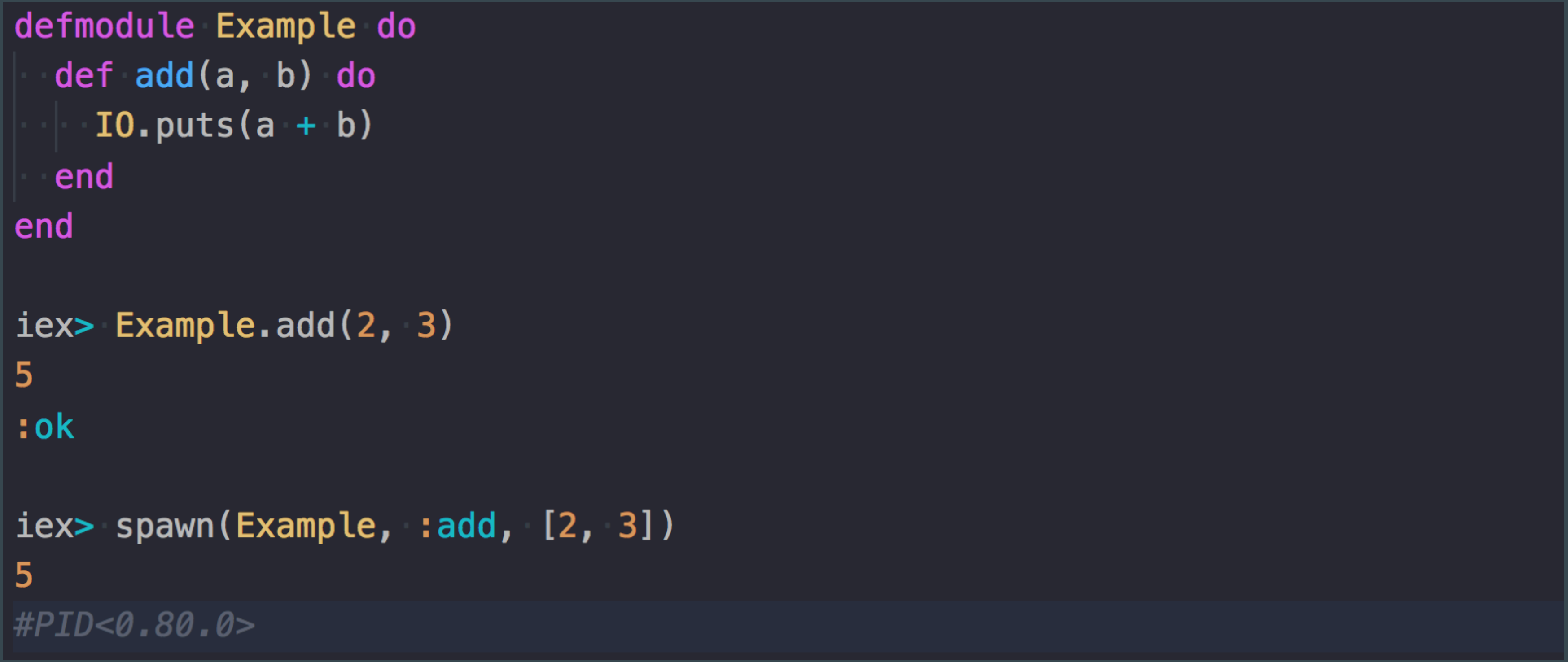
But this talk is not really about the Erlang programming language

Chapter II

A Whirlwind tour of Elixir, BEAM and OTP

Key ideas in Elixir

Isolated Processes



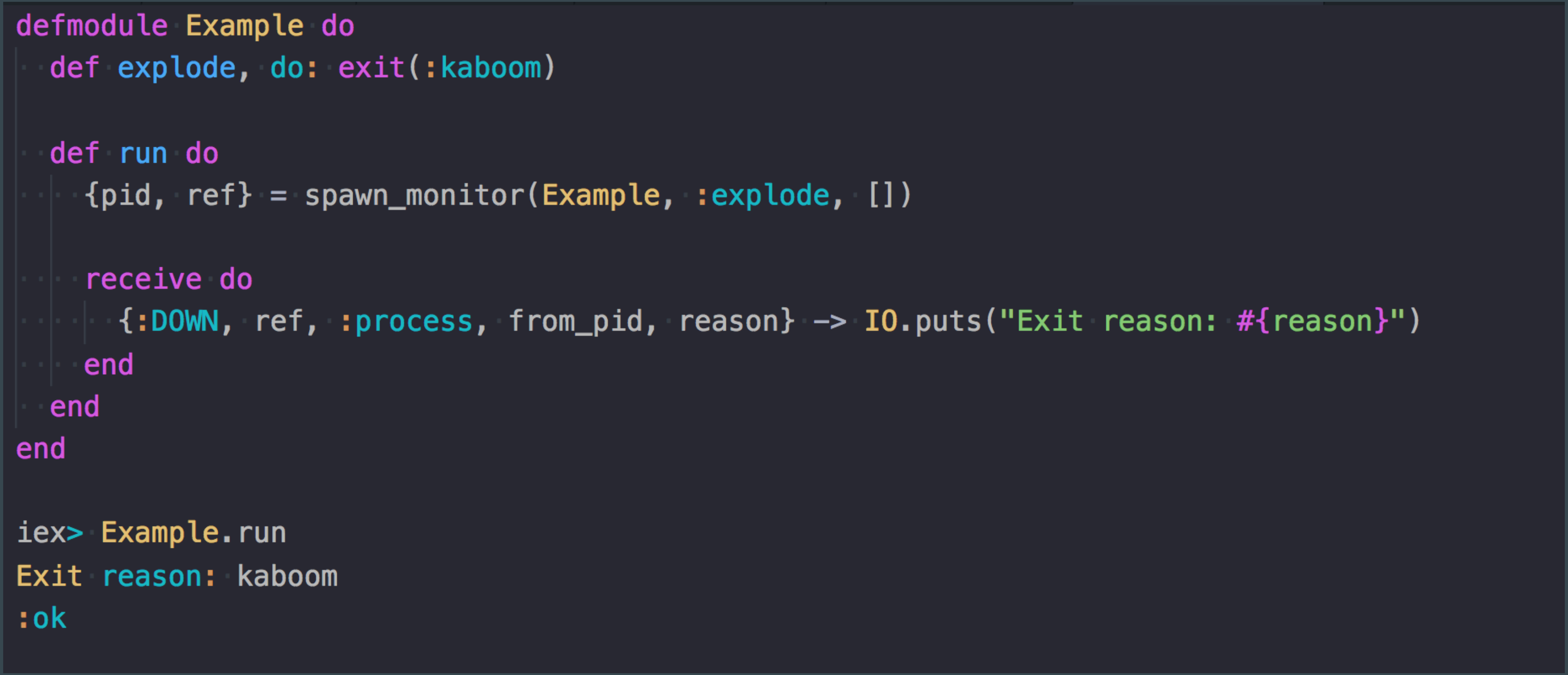
```
defmodule Example do
  def add(a, b) do
    IO.puts(a + b)
  end
end

iex> Example.add(2, 3)
5
:ok

iex> spawn(Example, :add, [2, 3])
5
#PID<0.80.0>
```

Pure Message Passing between processes

```elixir
defmodule Example do
  def listen do
    receive do
      {:ok, "hello"} -> IO.puts("World")
    end

    listen
  end
end

iex> pid = spawn(Example, :listen, [])
#PID<0.108.0>

iex> send pid, {:ok, "hello"}
World
{:ok, "hello"}

iex> send pid, :ok
:ok
```

The ability to detect errors in remote processes

A method for determining what error caused a process to crash



```elixir
defmodule Example do
  def explode, do: exit(:kaboom)

  def run do
    {pid, ref} = spawn_monitor(Example, :explode, [])

    receive do
      {:DOWN, ref, :process, from_pid, reason} -> IO.puts("Exit reason: #{reason}")
    end
  end
end

iex> Example.run
Exit reason: kaboom
:ok
```

This means that Elixir is

- Concurrent

This means that Elixir is

- Concurrent
- Fault tolerant

This means that Elixir is

- Concurrent
- Fault tolerant
- Soft real time

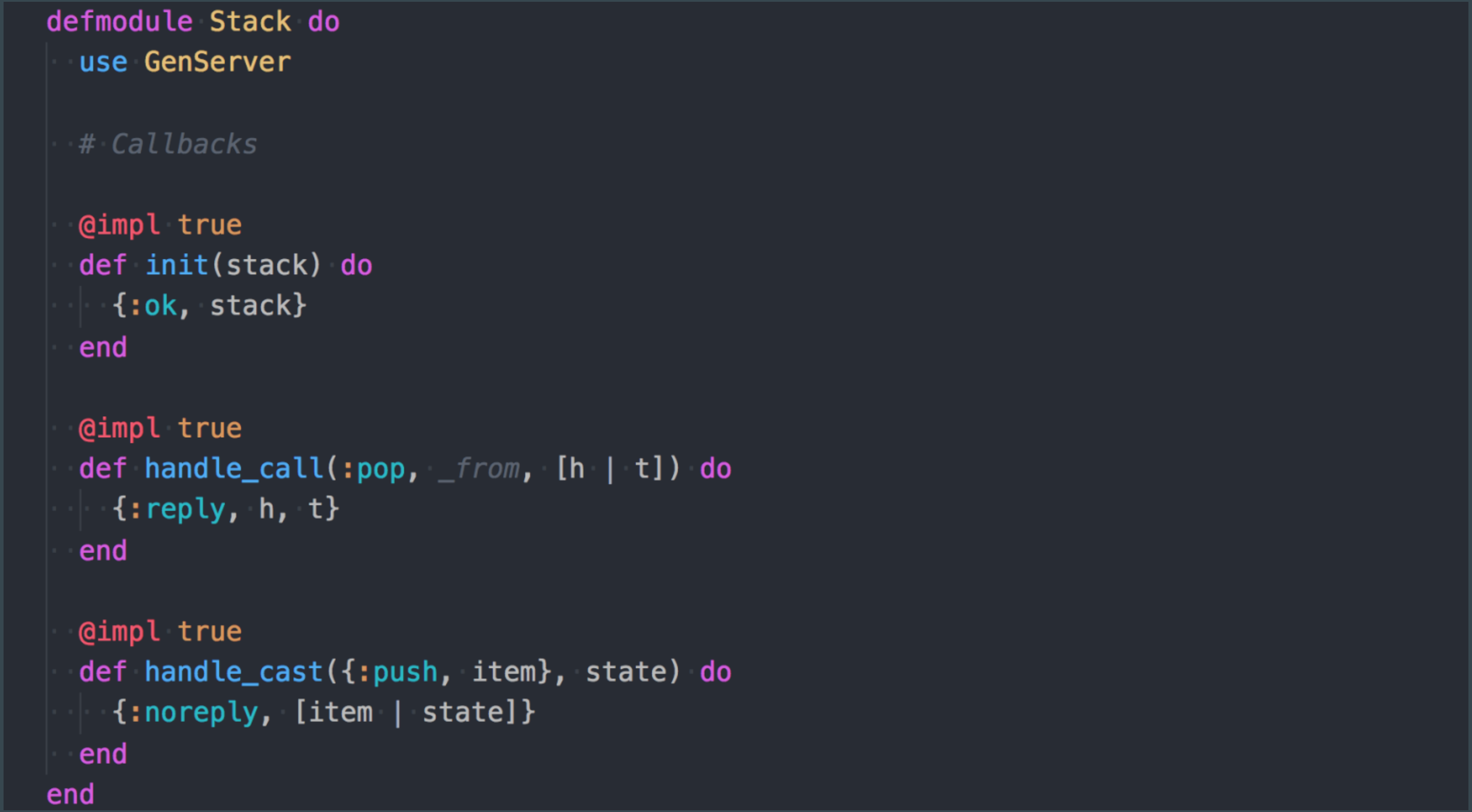
And built in support for things like hot code swap

Detour:
Open Telecom Platform
(OTP)

OTP is a set of libraries and tools that provides fundamental abstractions for BEAM languages

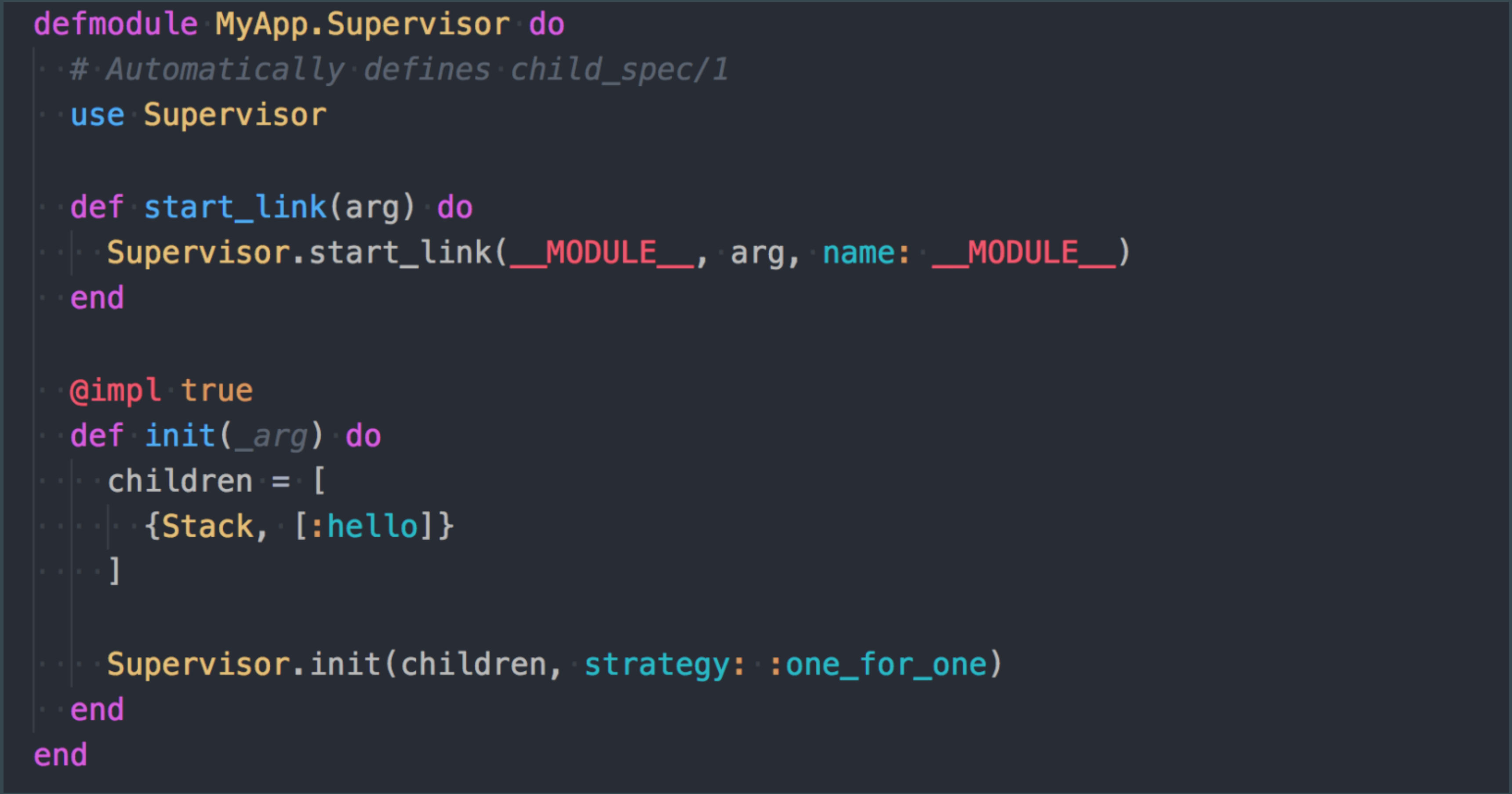
Here are some nice building blocks that OTP provides

GenServer: A generic server



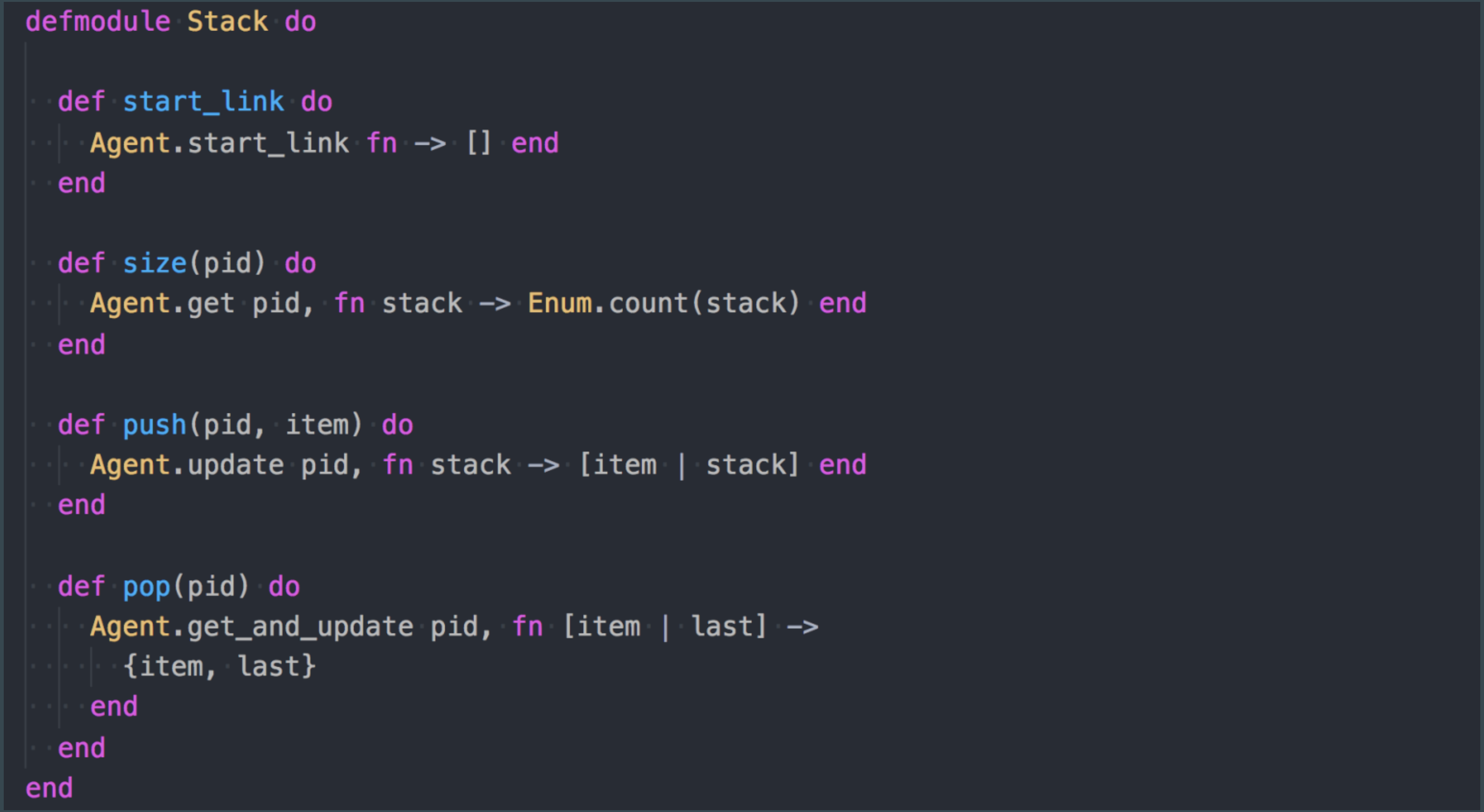
```elixir
defmodule Stack do
  use GenServer

  # Callbacks

  @impl true
  def init(stack) do
    {:ok, stack}
  end

  @impl true
  def handle_call(:pop, _from, [h | t]) do
    {:reply, h, t}
  end

  @impl true
  def handle_cast({:push, item}, state) do
    {:noreply, [item | state]}
  end
end
```

Process Supervisors



```elixir
defmodule MyApp.Supervisor do
  # Automatically defines child_spec/1
  use Supervisor

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

  @impl true
  def init(_arg) do
    children = [
      {Stack, [:hello]}
    ]

    Supervisor.init(children, strategy: :one_for_one)
  end
end
```

Agents: State Wrappers


```elixir
defmodule Stack do

  def start_link do
    Agent.start_link fn -> [] end
  end

  def size(pid) do
    Agent.get pid, fn stack -> Enum.count(stack) end
  end

  def push(pid, item) do
    Agent.update pid, fn stack -> [item | stack] end
  end

  def pop(pid) do
    Agent.get_and_update pid, fn [item | last] ->
      {item, last}
    end
  end
end
```

Sounds like what you will want for the Real World ™

Chapter III

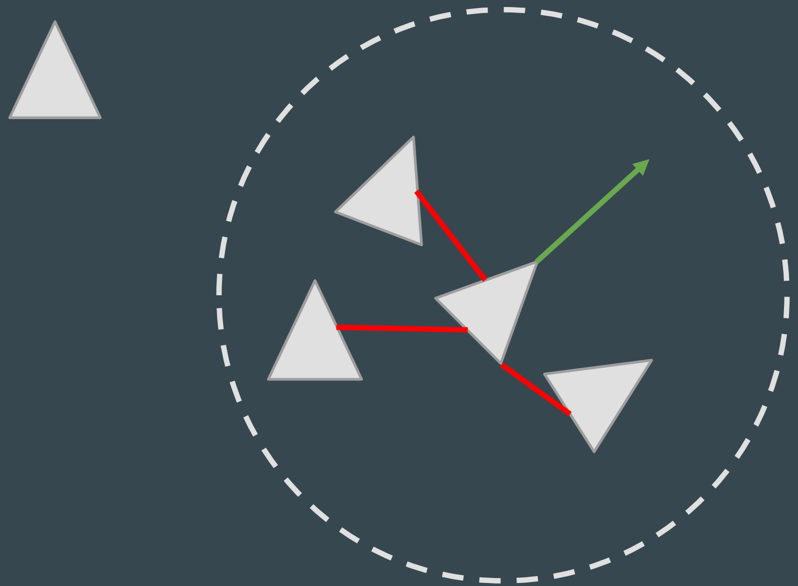
Demo time

Boids

The rules of flight

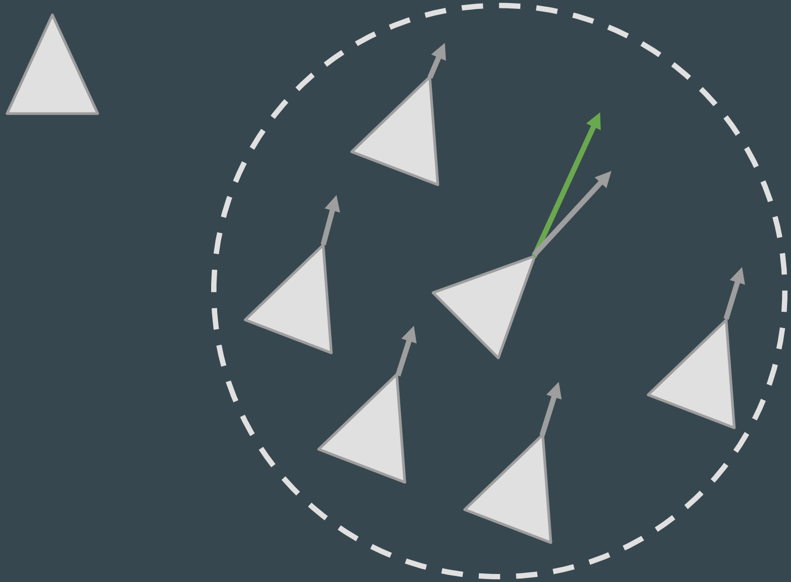
Separation: Steer to avoid crowding local flockmates

Rule I: Separation



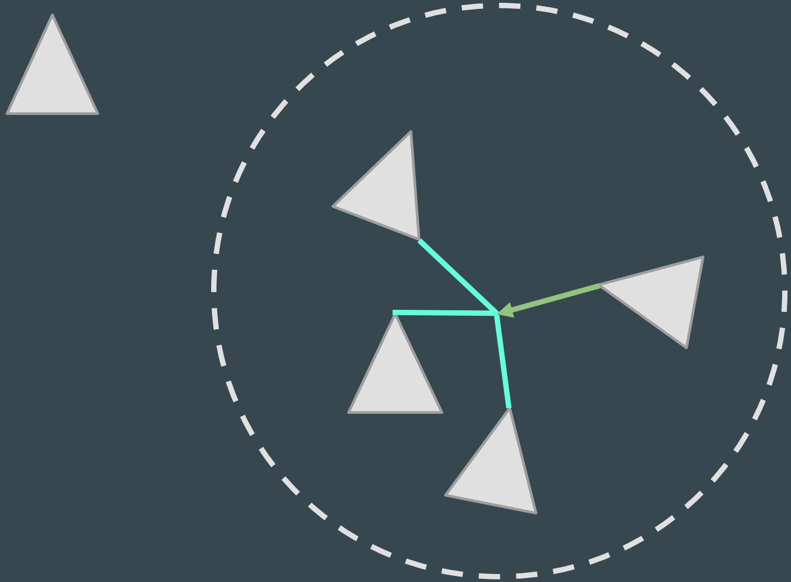
Alignment: Steer towards the average heading of local flockmates

Rule II: Alignment

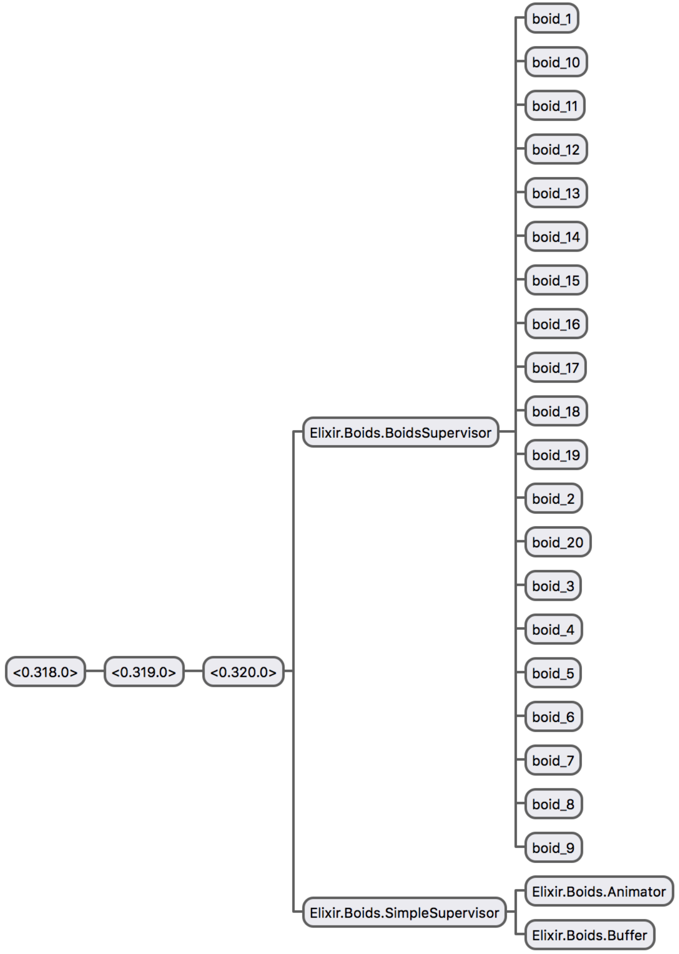


Cohesion: Steer to move toward the average position (center of mass) of local flockmates

Rule III: Cohesion



Modeling this with Elixir/OTP



<0.318.0>
<0.319.0>
<0.320.0>
Elixir.Boids.BoidsSupervisor
boid_1
boid_10
boid_11
boid_12
boid_13
boid_14
boid_15
boid_16
boid_17
boid_18
boid_19
boid_2
boid_20
boid_3
boid_4
boid_5
boid_6
boid_7
boid_8
boid_9
Elixir.Boids.SimpleSupervisor
Elixir.Boids.Animator
Elixir.Boids.Buffer

Modeling the events

Move

Render

Dealing with failures

Restart the boid

Concurrency

Keep your processes light and have many of them.

Epilogue

Make it work, then make it beautiful, then if you really, really have to, make it fast.

Joe Armstrong

That's all Folks!

- @aishrajdahal