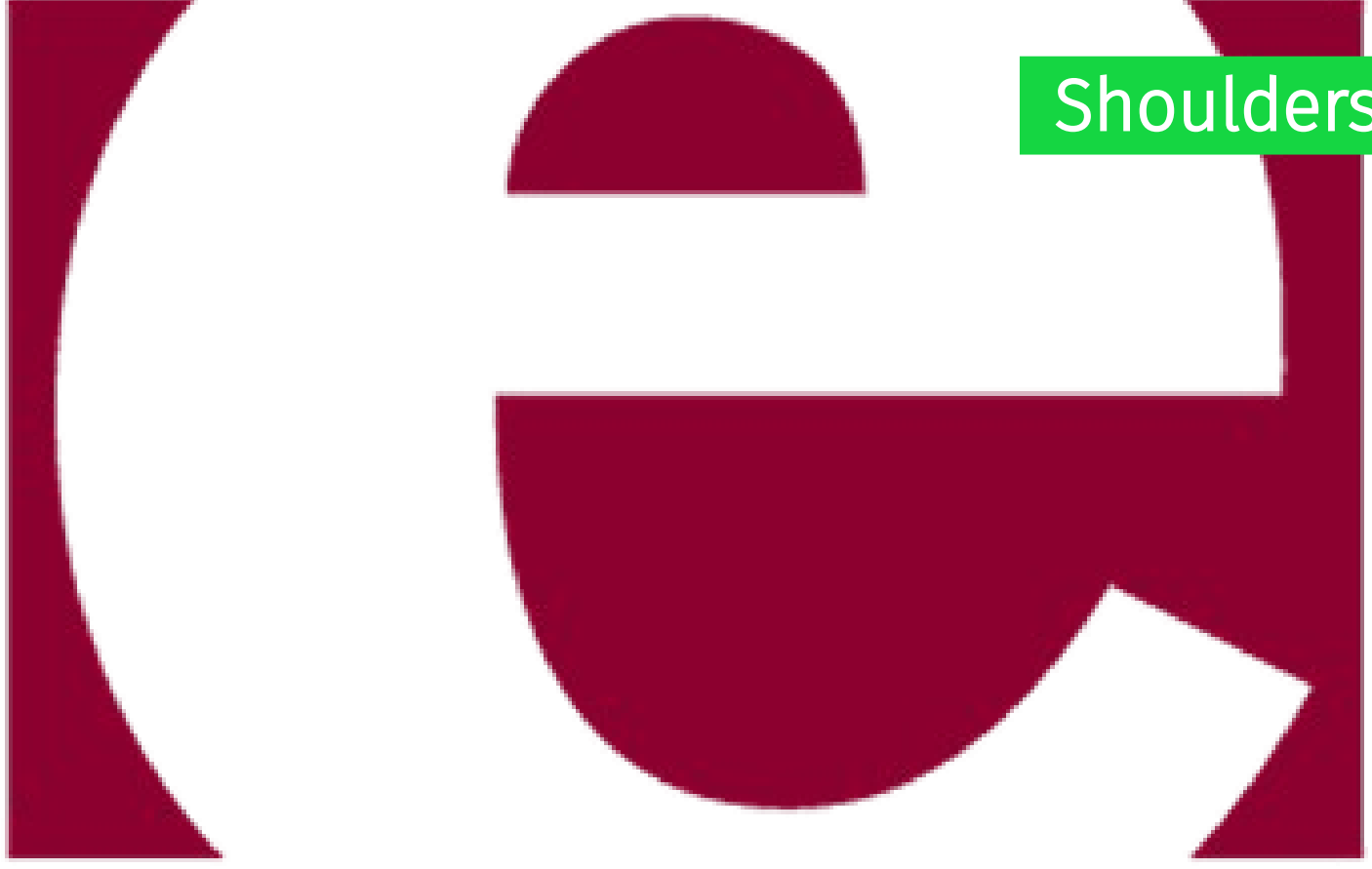
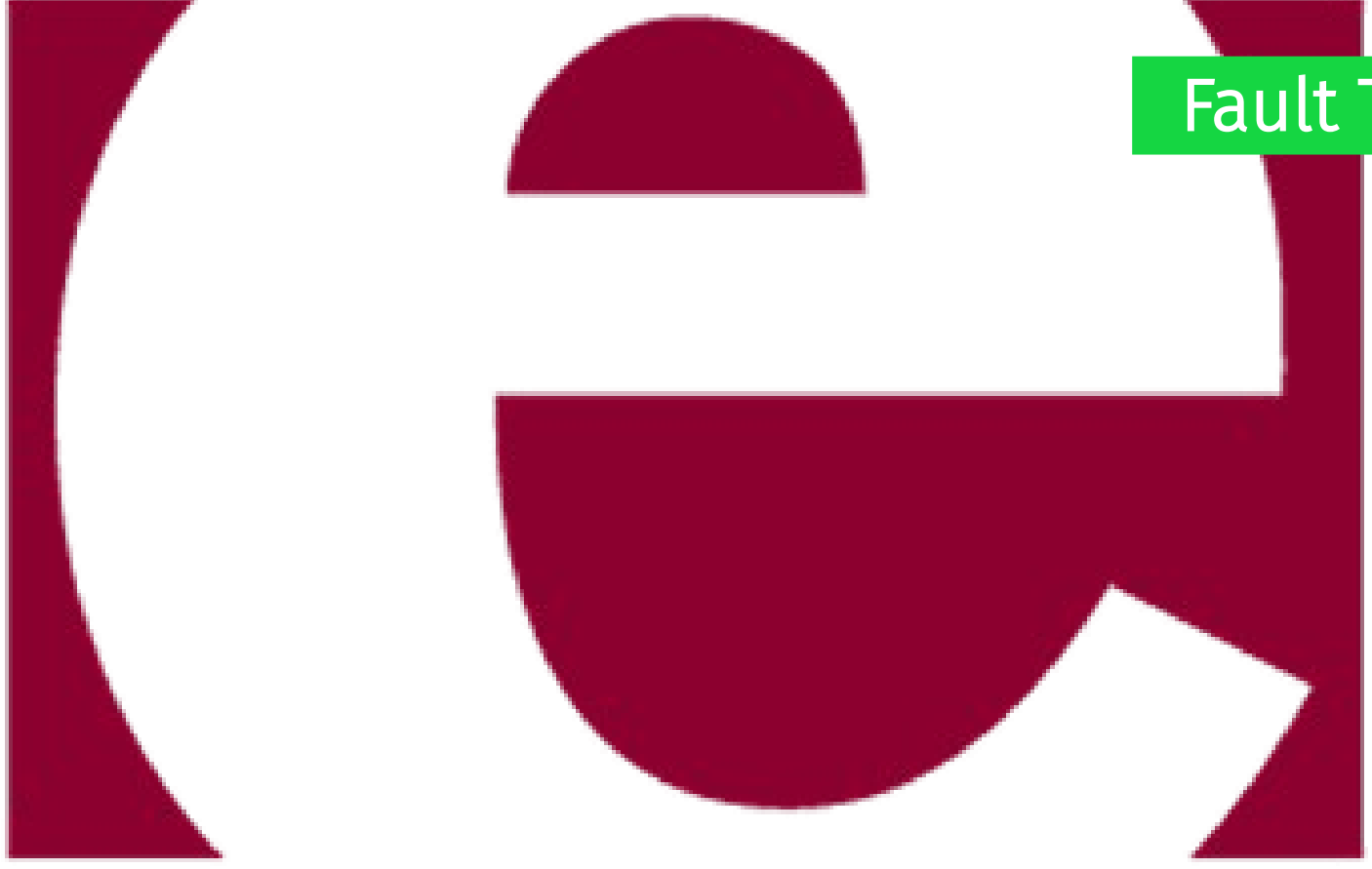
elixir



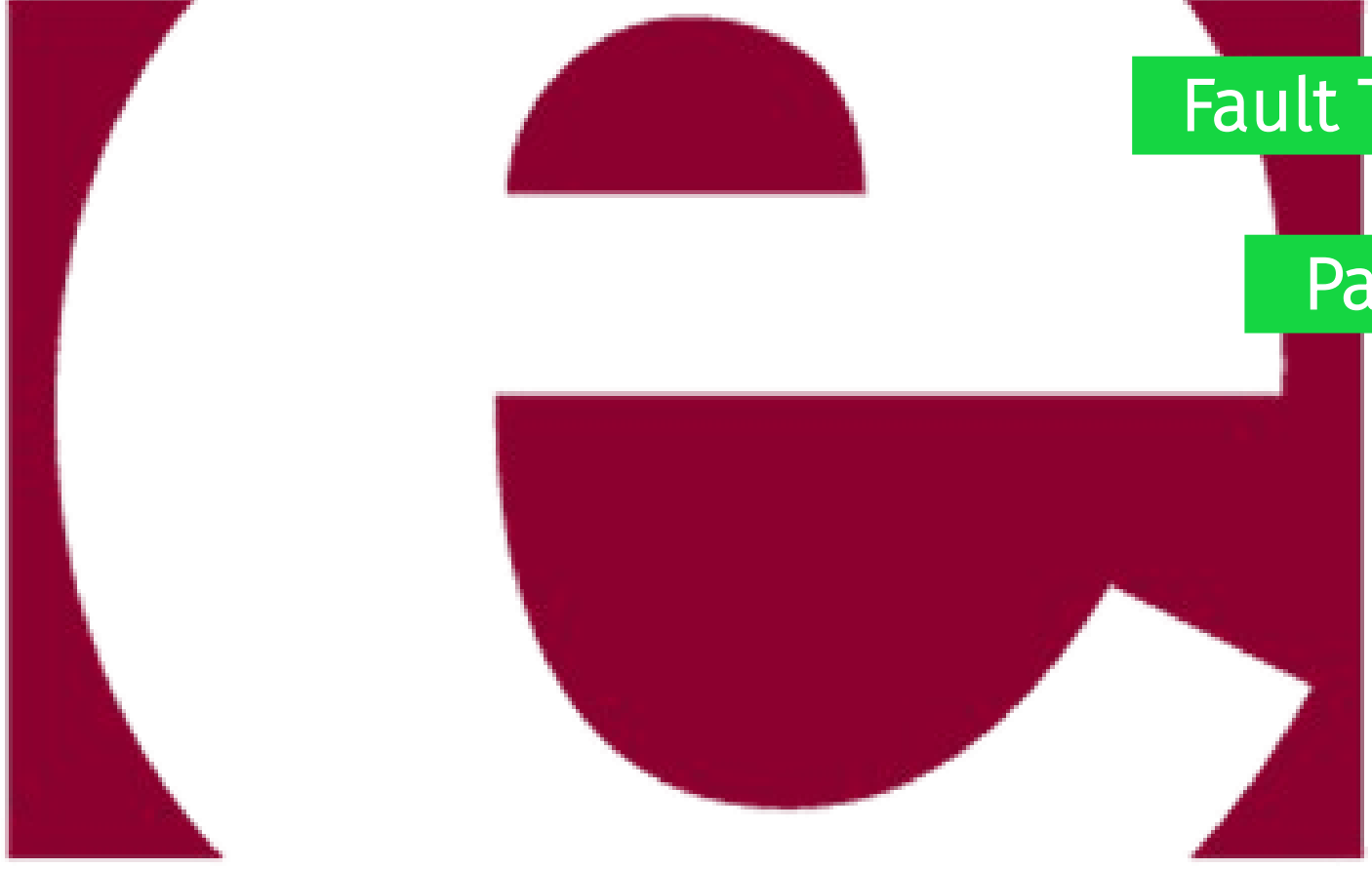
ERLANG

Shoulders of giants



ERLANG

Fault Tolerance

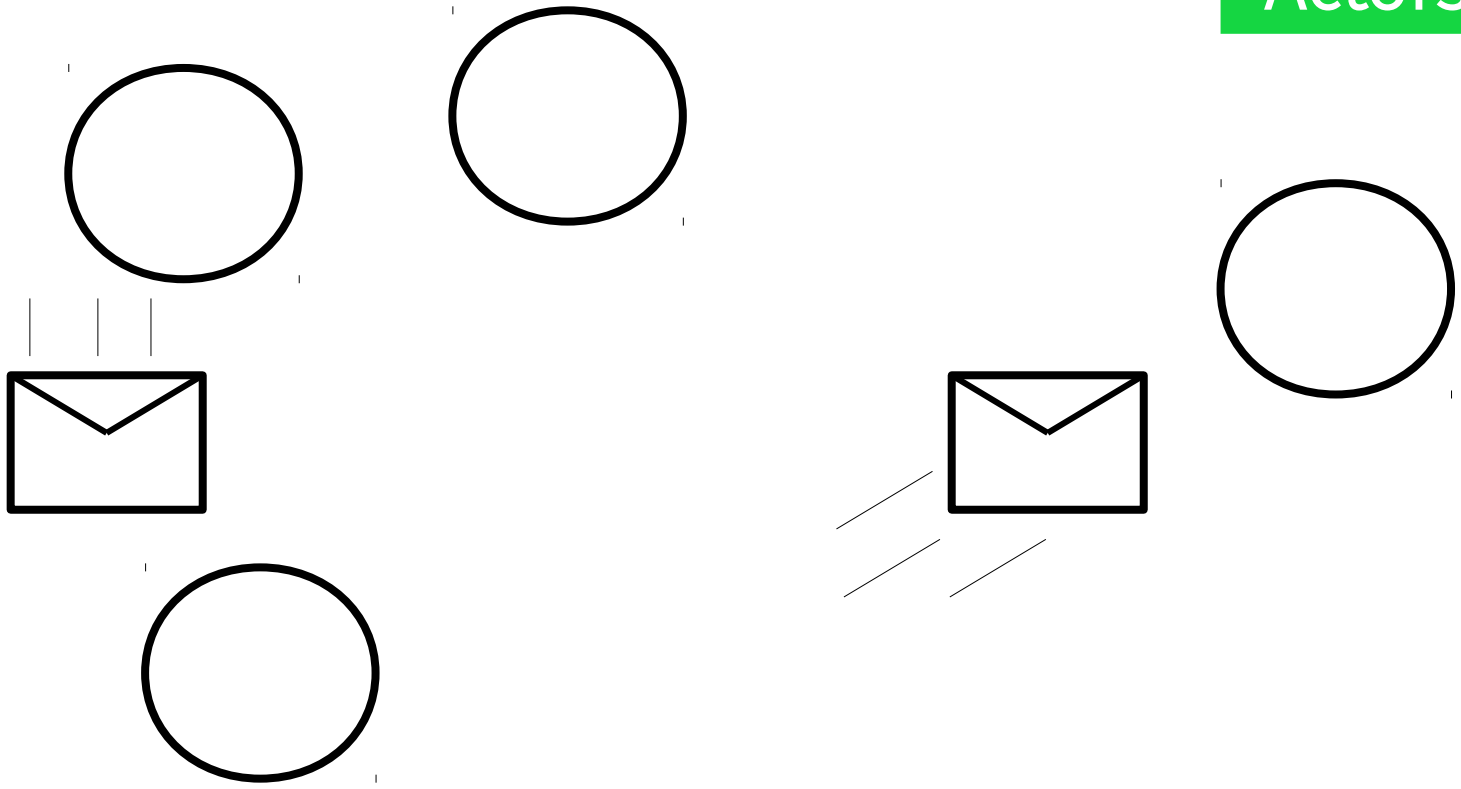


ERLANG

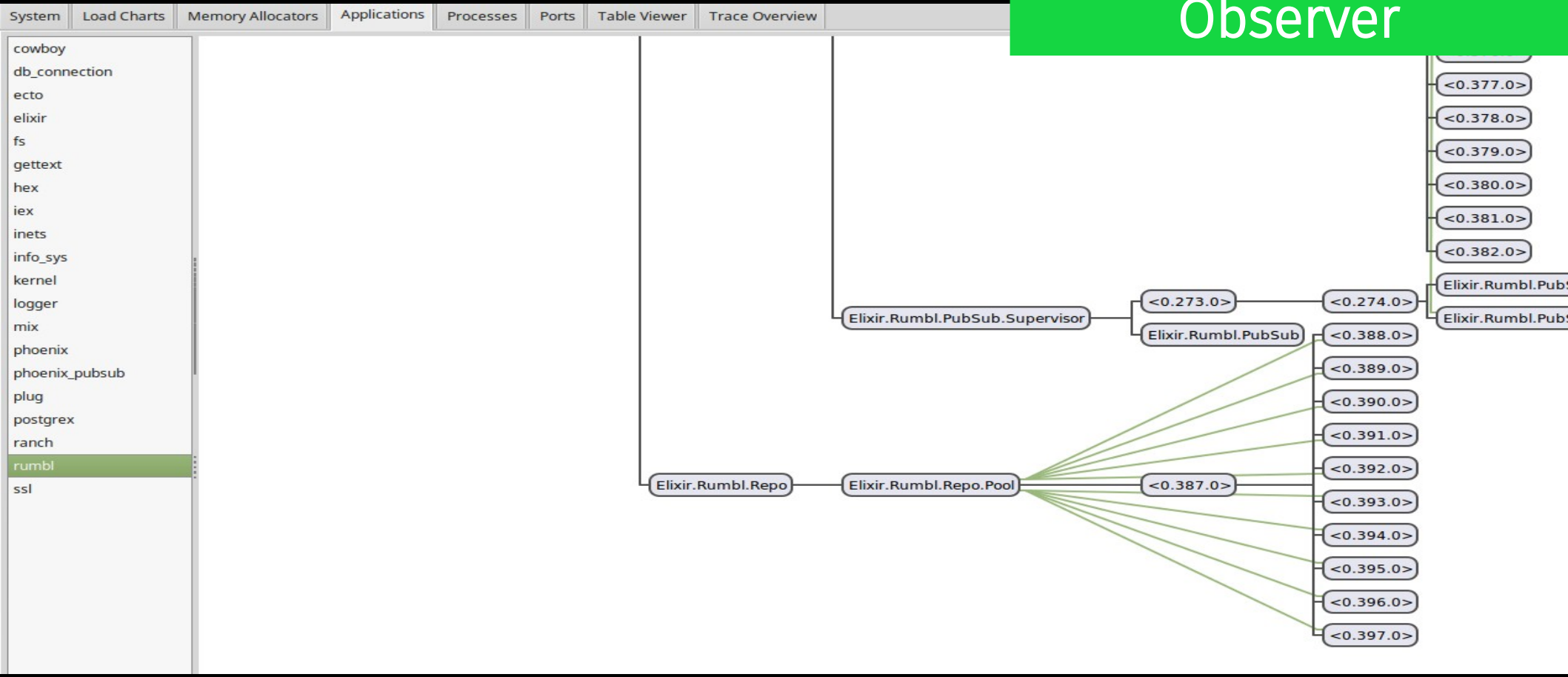
Fault Tolerance

Parallelism

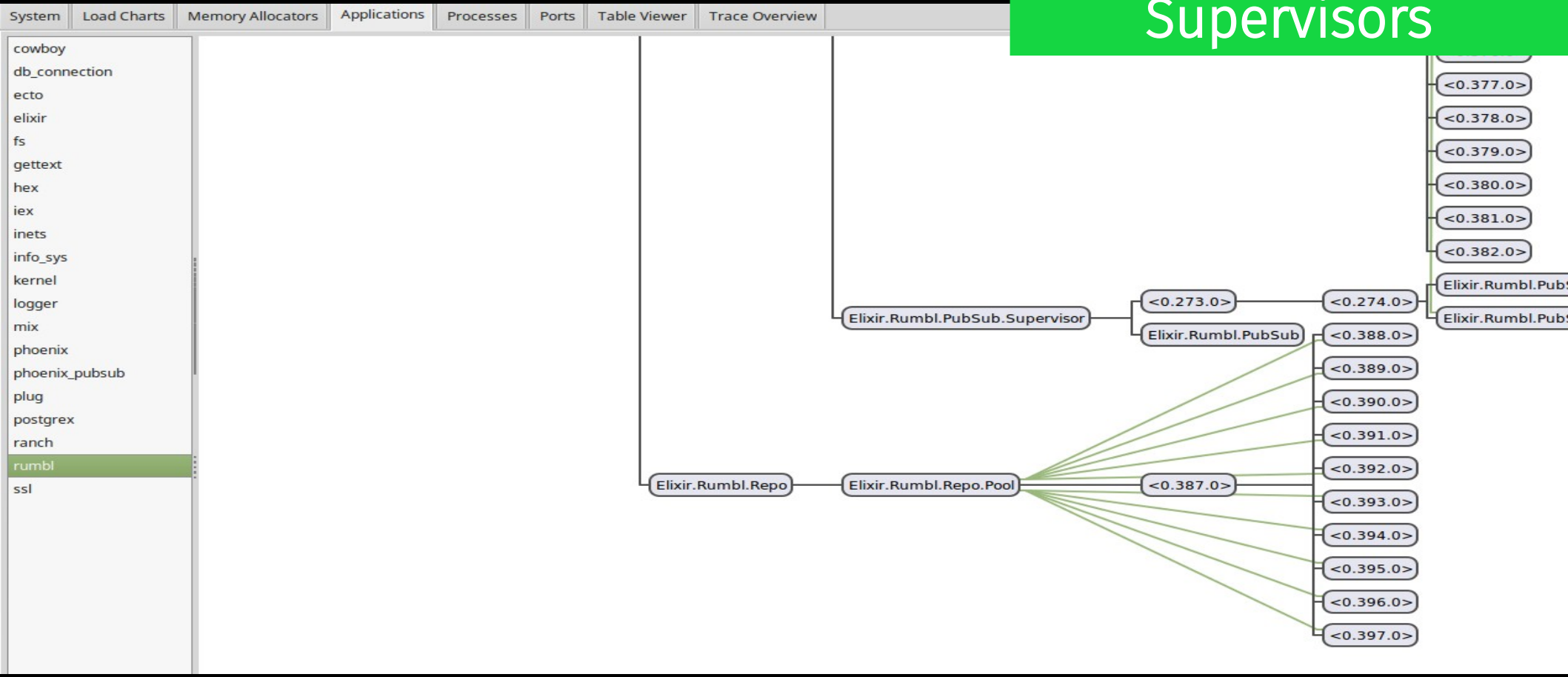
Actors / Processes

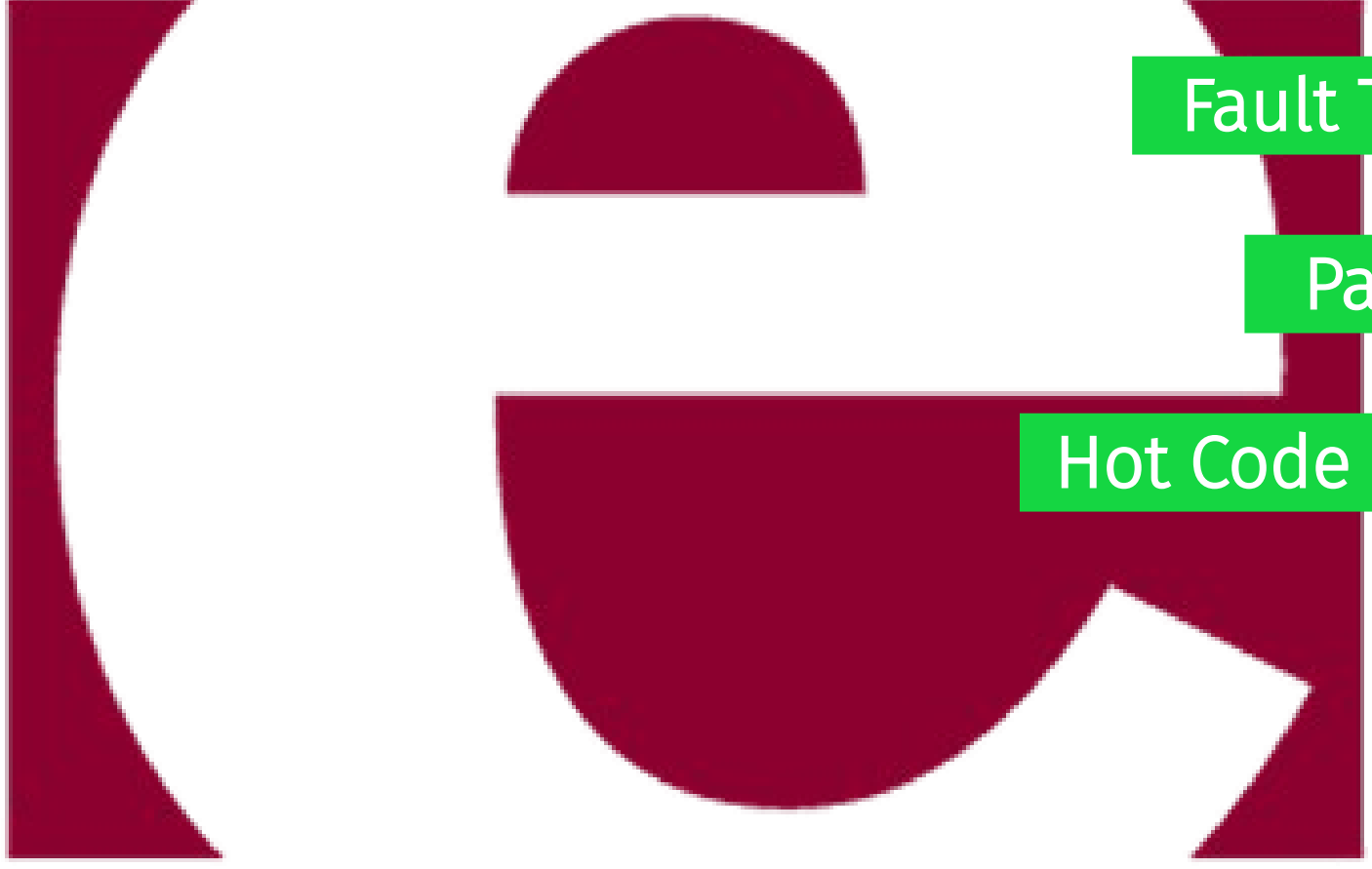


Observer


System
Load Charts
Memory Allocators
Applications
Processes
Ports
Table Viewer
Trace Overview
cowboy
db_connection
ecto
elixir
fs
gettext
hex
iex
inets
info_sys
kernel
logger
mix
phoenix
phoenix_pubsub
plug
postgrex
ranch
rumbl
ssl
Elixir.Rumbl.PubSub.Supervisor
<0.273.0>
Elixir.Rumbl.PubSub
<0.274.0>
<0.377.0>
<0.378.0>
<0.379.0>
<0.380.0>
<0.381.0>
<0.382.0>
Elixir.Rumbl.Repo
Elixir.Rumbl.Repo.Pool
<0.387.0>
<0.388.0>
<0.389.0>
<0.390.0>
<0.391.0>
<0.392.0>
<0.393.0>
<0.394.0>
<0.395.0>
<0.396.0>
<0.397.0>

Supervisors



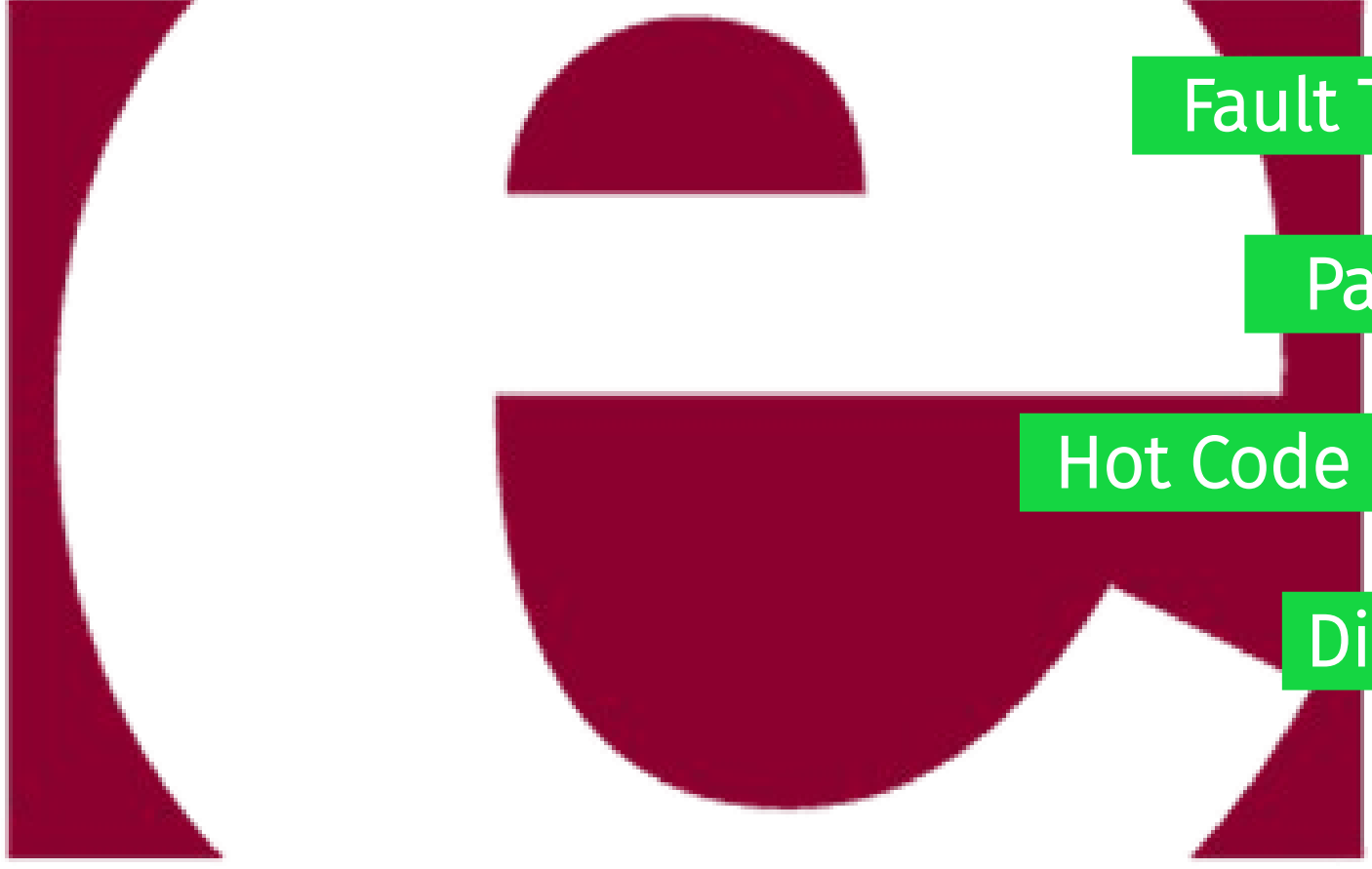


ERLANG

Fault Tolerance

Parallelism

Hot Code Swapping



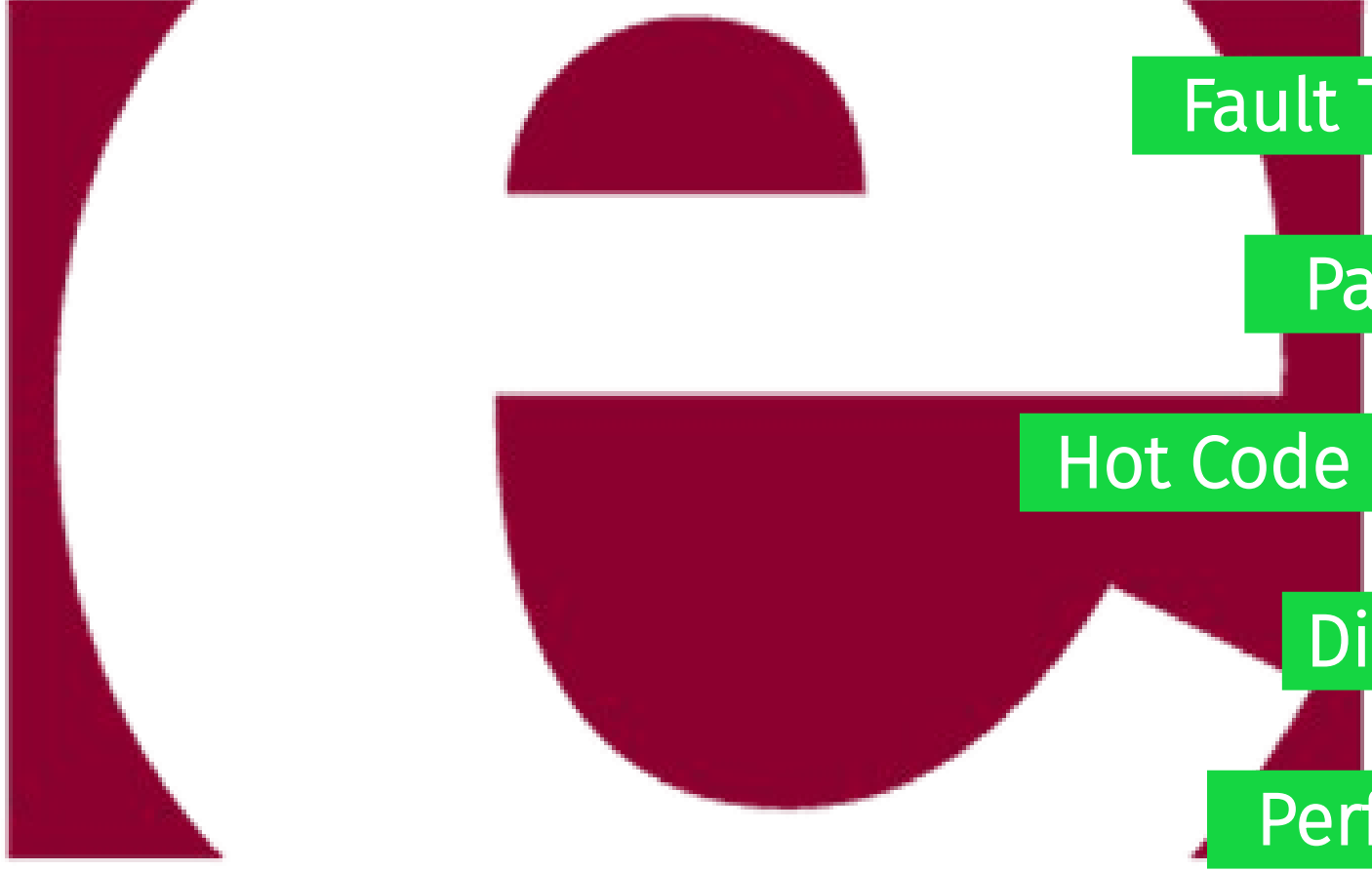
ERLANG

Fault Tolerance

Parallelism

Hot Code Swapping

Distribution



ERLANG

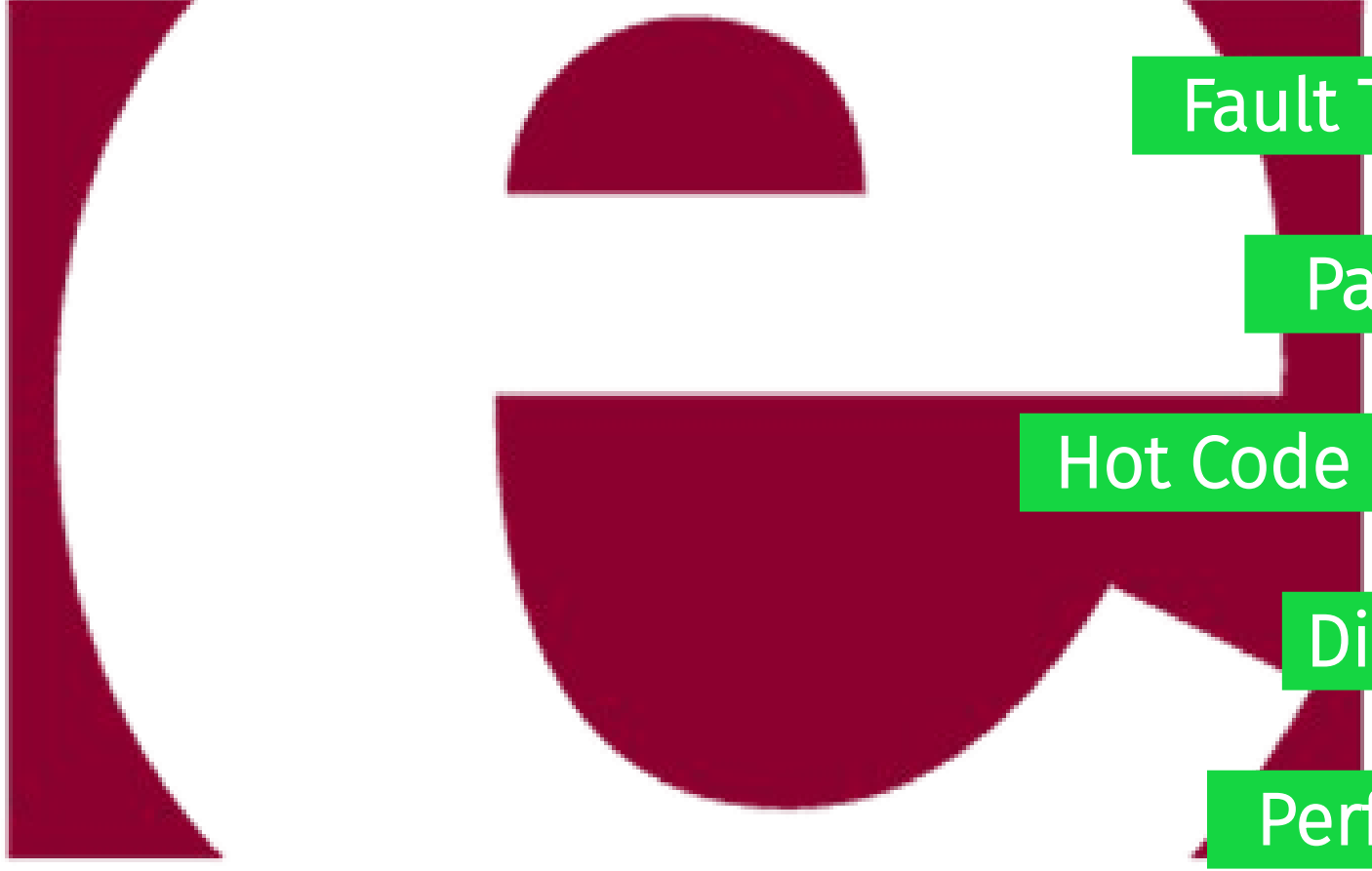
Fault Tolerance

Parallelism

Hot Code Swapping

Distribution

Performance



ERLANG

Fault Tolerance

Parallelism

Hot Code Swapping

Distribution

Performance

Functional Programming

Fault Tolerance

Parallelism

Hot Code Swapping

Distribution

Performance

Functional Programming

ERLANG

If somebody came to me and wanted to pay me a lot of money to build a large scale message handling system that really had to be up all the time, could never afford to go down for years at a time, I would unhesitatingly choose Erlang to build it in.

Tim Bray

(Director of Web Technologies Sun Microsystems - 2008)

José Valim
@josevalim

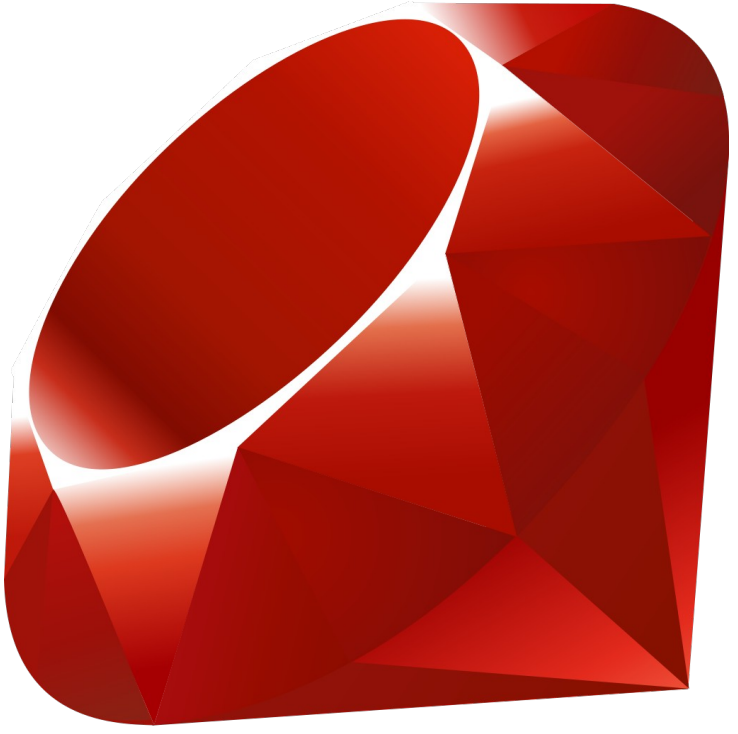
Folgen

Reminder it is 2016. Almost everything you do must be using all CPUs: compiling code, booting, running tests... Easy math on the wins here.

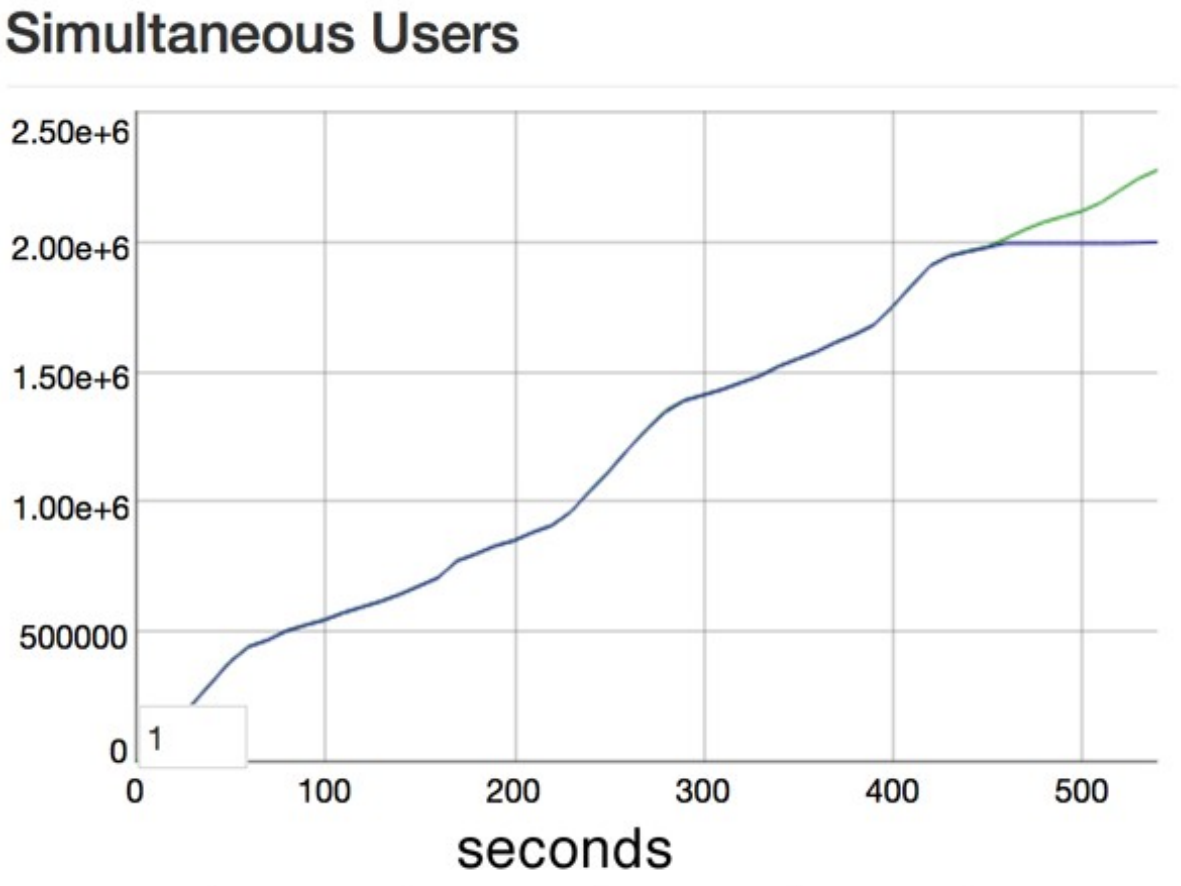
RETWEETS 56 GEFÄLLT 67

07:03 - 29. Feb. 2016

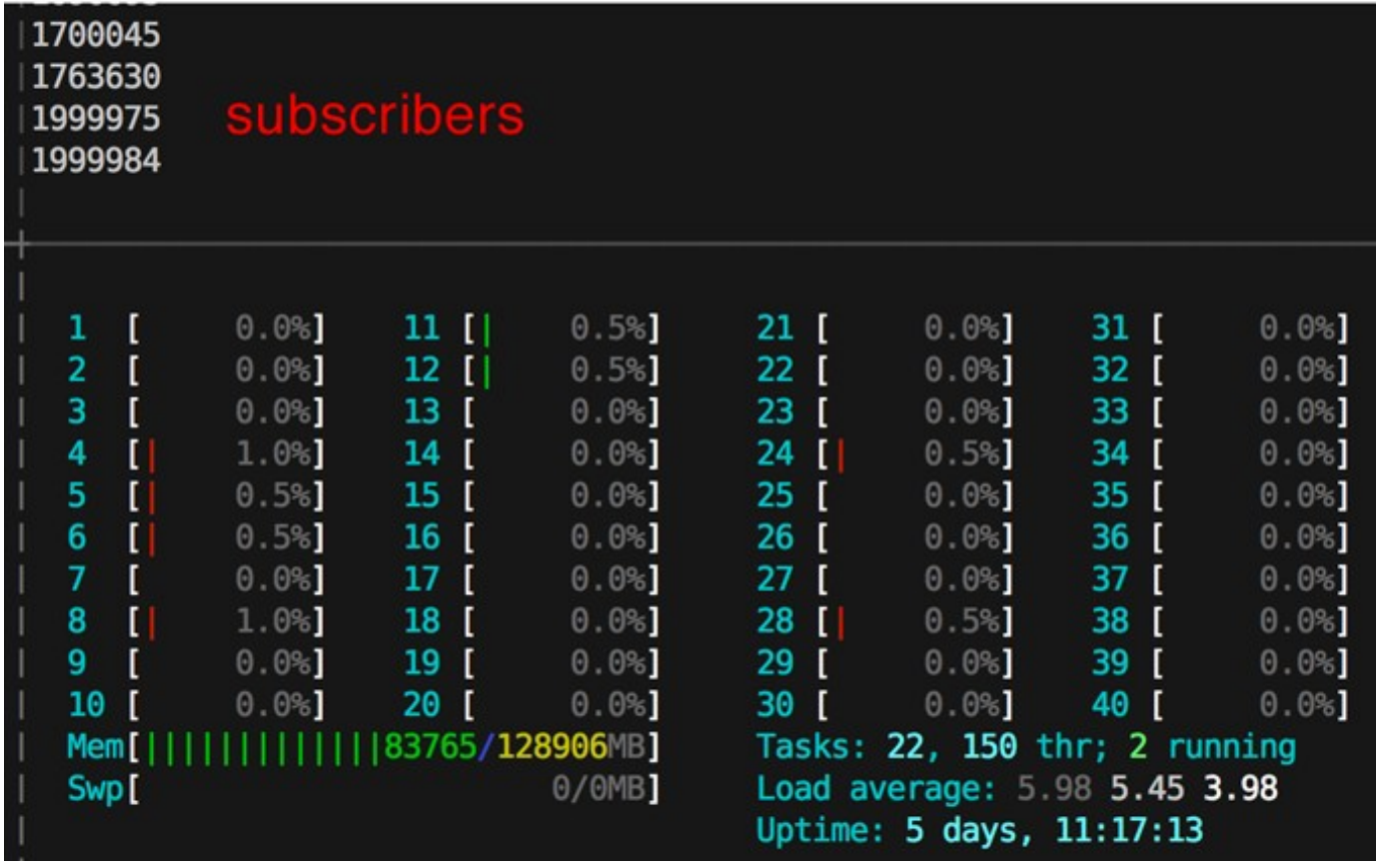

ERLANG



2 Million Websocket Connections



2 Million Websocket Connections



Great Company

SQUARE ENIX

DISCORD

Elixir and Phoenix

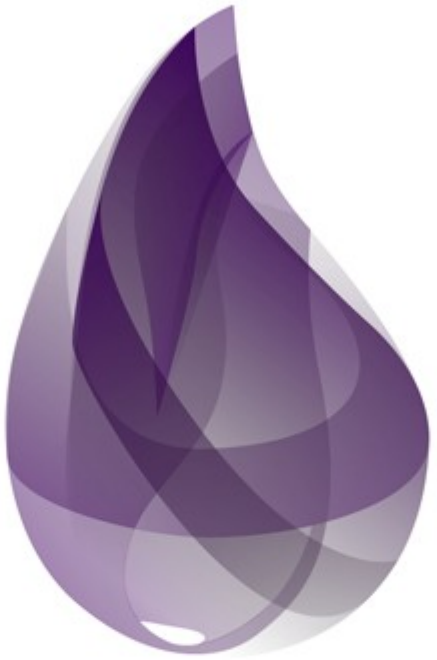
fast, concurrent and explicit

Tobias Pfeiffer
@PragTob
pragtob.info

Elixir and Phoenix

fast, concurrent and explicit

Tobias Pfeiffer
@PragTob
pragtob.info



elixir

```elixir
defmodule MapDemo do
  @doc """
  iex> MapDemo.map [1, 2, 3, 4], fn i -> i + 1 end
  [2, 3, 4, 5]
  """
  def map(list, function) do
    Enum.reverse(do_map([], list, function))
  end

  defp do_map(acc, [], _function) do
    acc
  end

  defp do_map(acc, [head | tail], function) do
    do_map([function.(head) | acc], tail, function)
  end
end
```

Ruby-like Syntax

```elixir
defmodule MapDemo do
  @doc """
  iex> MapDemo.map [1, 2, 3, 4], fn i -> i + 1 end
  [2, 3, 4, 5]
  """
  def map(list, function) do
    Enum.reverse(do_map([], list, function))
  end

  defp do_map(acc, [], _function) do
    acc
  end

  defp do_map(acc, [head | tail], function) do
    do_map([function.(head) | acc], tail, function)
  end
end
```

First Class Functions

```elixir
defmodule MapDemo do
  @doc """
  iex> MapDemo.map [1, 2, 3, 4], fn i -> i + 1 end
  [2, 3, 4, 5]
  """
  def map(list, function) do
    Enum.reverse(do_map([], list, function))
  end

  defp do_map(acc, [], _function) do
    acc
  end

  defp do_map(acc, [head | tail], function) do
    do_map([function.(head) | acc], tail, function)
  end
end
```

Pattern Matching

```elixir
defmodule MapDemo do
  @doc """
  iex> MapDemo.map [1, 2, 3, 4], fn i -> i + 1 end
  [2, 3, 4, 5]
  """
  def map(list, function) do
    Enum.reverse(do_map([], list, function))
  end

  defp do_map(acc, [], _function) do
    acc
  end

  defp do_map(acc, [head | tail], function) do
    do_map([function.(head) | acc], tail, function)
  end
end
```

Pattern Matching

```elixir
defmodule Patterns do
  def greet(%{name: name, age: age}) do
    IO.puts("Hi there #{name}, what's up at #{age}?")
  end

  def greet(%{name: "Emily"}) do
    IO.puts("Thanks for running Øredev for so long!")
  end

  def greet(%{name: name}) do
    IO.puts("Hi there #{name}")
  end

  def greet(_) do
    IO.puts("Hi")
  end
end
```

```
%{name: "Tobi", age: 30, something: :else}
```

```elixir
defmodule Patterns do
  def greet(%{name: name, age: age}) do
    IO.puts("Hi there #{name}, what's up at #{age}?")
  end

  def greet(%{name: "Emily"}) do
    IO.puts("Thanks for running Øredev for so long!")
  end

  def greet(%{name: name}) do
    IO.puts("Hi there #{name}")
  end

  def greet(_) do
    IO.puts("Hi")
  end
end
```

%{name: "Emily"}

```elixir
defmodule Patterns do
  def greet(%{name: name, age: age}) do
    IO.puts("Hi there #{name}, what's up at #{age}?")
  end

  def greet(%{name: "Emily"}) do
    IO.puts("Thanks for running Øredev for so long!")
  end

  def greet(%{name: name}) do
    IO.puts("Hi there #{name}")
  end

  def greet(_) do
    IO.puts("Hi")
  end
end
```

```
%{name: "Tadeáš"}
```

```elixir
defmodule Patterns do
  def greet(%{name: name, age: age}) do
    IO.puts("Hi there #{name}, what's up at #{age}?")
  end

  def greet(%{name: "Emily"}) do
    IO.puts("Thanks for running Øredev for so long!")
  end

  def greet(%{name: name}) do
    IO.puts("Hi there #{name}")
  end

  def greet(_) do
    IO.puts("Hi")
  end
end
```

["Mop"]

```elixir
defmodule Patterns do
  def greet(%{name: name, age: age}) do
    IO.puts("Hi there #{name}, what's up at #{age}?")
  end

  def greet(%{name: "Emily"}) do
    IO.puts("Thanks for running Øredev for so long!")
  end

  def greet(%{name: name}) do
    IO.puts("Hi there #{name}")
  end

  def greet(_) do
    IO.puts("Hi")
  end
end
```

Docs

```elixir
defmodule MapDemo do
  @doc """
  iex> MapDemo.map [1, 2, 3, 4], fn i -> i + 1 end
  [2, 3, 4, 5]
  """
  def map(list, function) do
    Enum.reverse(do_map([], list, function))
  end

  defp do_map(acc, [], _function) do
    acc
  end

  defp do_map(acc, [head | tail], function) do
    do_map([function.(head) | acc], tail, function)
  end
end
```

Doctests

```elixir
defmodule MapDemo do
  @doc """
  iex> MapDemo.map [1, 2, 3, 4], fn i -> i + 1 end
  [2, 3, 4, 5]
  """
  def map(list, function) do
    Enum.reverse(do_map([], list, function))
  end

  defp do_map(acc, [], _function) do
    acc
  end

  defp do_map(acc, [head | tail], function) do
    do_map([function.(head) | acc], tail, function)
  end
end
```

Meta Programming

```
defmacro plug(plug, opts \\ []) do
  quote do
    @plugs {unquote(plug), unquote(opts), true}
  end
end
```

Polymorphism

```
defprotocol Blank do
  def blank?(data)
end

defimpl Blank, for: List do
  def blank?([]), do: true
  def blank?(_), do: false
end

defimpl Blank, for: Map do
  def blank?(map), do: map_size(map) == 0
end
```

Polymorphism

```
defprotocol Blank do
  def blank?(data)
end

defimpl Blank, for: List do
  def blank?([]), do: true
  def blank?(_), do: false
end

defimpl Blank, for: Map do
  def blank?(map), do: map_size(map) == 0
end
```

Polymorphism

```
defprotocol Blank do
  def blank?(data)
end

defimpl Blank, for: List do
  def blank?([]), do: true
  def blank?(_), do: false
end

defimpl Blank, for: Map do
  def blank?(map), do: map_size(map) == 0
end
```

Implemented in itself!

```
@spec all?(t) :: boolean
@spec all?(t, (element → as_boolean(term))) :: boolean

def all?(enumerable, fun \\ fn x → x end)

def all?(enumerable, fun) when is_list(enumerable) and is_function(fun, 1) do
  do_all?(enumerable, fun)
end
```

Optional Typespecs

```
@spec all?(t) :: boolean
@spec all?(t, (element → as_boolean(term))) :: boolean

def all?(enumerable, fun \\ fn x → x end)

def all?(enumerable, fun) when is_list(enumerable) and is_function(fun, 1) do
  do_all?(enumerable, fun)
end
```

“Interfaces”

```
defmodule Plug do
  @type opts :: tuple | atom | integer | float | [opts]

  @callback init(opts) :: opts
  @callback call(Plug.Conn.t(), opts) :: Plug.Conn.t()
end
```

“Interfaces”

```elixir
defmodule Plug do
  @type opts :: tuple | atom | integer | float | [opts]

  @callback init(opts) :: opts
  @callback call(Plug.Conn.t(), opts) :: Plug.Conn.t()
end
```

“Interfaces”

```elixir
defmodule Plug.Head do
  @behaviour Plug

  alias Plug.Conn

  def init([]), do: []

  def call(%Conn{method: "HEAD"} = conn, []) do
    %{conn | method: "GET"}
  end

  def call(conn, []), do: conn
end
```

“Interfaces”

```elixir
defmodule Plug.Head do
  @behaviour Plug

  alias Plug.Conn

  def init([]), do: []

  def call(%Conn{method: "HEAD"} = conn, []) do
    %{conn | method: "GET"}
  end

  def call(conn, []), do: conn
end
```

“Interfaces”

```elixir
defmodule Plug.Head do
  @behaviour Plug

  alias Plug.Conn

  def init([]), do: []

  def call(%Conn{method: "HEAD"} = conn, []) do
    %{conn | method: "GET"}
  end

  def call(conn, []), do: conn
end
```

Functional Programming?

Not Me


Me

Where to call functions

```
2.6.5 :001 > [1, 2, 3, 4].map { |i| i + 1 }
 ⇒ [2, 3, 4, 5]
```

VS

```
iex(2)> Enum.map [1, 2, 3, 4], fn i → i + 1 end
[2, 3, 4, 5]
```

Separate State & Behaviour

Transformation
of Data

Pipe

```
people = DB.find_customers
orders = Orders.for_customers(people)
tax    = sales_tax(orders, 2019)
filing = prepare_filing(tax)
```

Pipe

```
filing = DB.find_customers
         |> Orders.for_customers
         |> sales_tax(2019)
         |> prepare_filing
```

Pipe

```
filing =
  prepare_filing(sales_tax(
    Orders.for_cusstomers(DB.find_customers), 2019))
```

Pipe

```
filing =
  prepare_filing(sales_tax(
    Orders.for_cusstomers(DB.find_customers), 2019))
```

Pipe

```
filing = DB.find_customers
         |> Orders.for_customers
         |> sales_tax(2019)
         |> prepare_filing
```

Immutable Data

```
person = Person.new(attributes)
do_something(person)
insert_in_db(person)
```

Immutable Data

```
person = Person.new(attributes)
person = do_something(person)
insert_in_db(person)
```

Principles vs Power

Same Input,
Same Output

Readability

Refactoring++

Testing++


TM

Transformation

connection
▷ router
▷ controller
▷ model
▷ view

Router

```
defmodule DemoWeb.Router do
  use DemoWeb, :router

  scope "/", DemoWeb do
    pipe_through :browser

    get "/", PageController, :index

    resources "/users", UserController

    resources "/posts", PostController
  end
end
```

Controller

```elixir
defmodule DemoWeb.UserController do
  use DemoWeb, :controller

  def show(conn, %{"id" => id}) do
    user = Accounts.get_user!(id)
    render(conn, "show.html", user: user)
  end
end
```

Context

```
defmodule DemoWeb.UserController do
  use DemoWeb, :controller

  def show(conn, %{"id" => id}) do
    user = Accounts.get_user!(id)
    render(conn, "show.html", user: user)
  end
end
```

Context

```
defmodule Demo.Accounts do
  def get_user!(id) do
    Repo.get!(User, id)
  end
end
```

Schema

```elixir
defmodule Demo.Accounts.User do
  use Ecto.Schema

  schema "users" do
    field :age, :integer
    field :name, :string

    has_many(:posts, Blogging.Post)

    timestamps()
  end
end
```

Controller

```elixir
defmodule DemoWeb.UserController do
  use DemoWeb, :controller

  def show(conn, %{"id" => id}) do
    user = Accounts.get_user!(id)
    render(conn, "show.html", user: user)
  end
end
```

```elixir
defmodule DemoWeb.UserView do
  use DemoWeb, :view

  def display(user) do
    "#{user.name} (#{user.age})"
  end
end
```

```
<h1>Show User</h1>

<ul>

  <li>
    <strong>Name:</strong>
    <%= @user.name %>
  </li>

  <li>
    <strong>Age:</strong>
    <%= @user.age %>
  </li>

</ul>
```


RAILS


TM

Changesets

```elixir
def new_changeset(model, params \\ %{}) do
  model
  |> cast(params, ~w(name username), [])
  |> unique_constraint(:username)
  |> validate_length(:username, min: 1, max: 20)
end

def registration_changeset(model, params) do
  model
  |> new_changeset(params)
  |> cast(params, ~w(password), [])
  |> validate_length(:password, min: 6, max: 100)
  |> put_pass_hash()
end
```

Changesets

```elixir
def new_changeset(model, params \\ %{}) do
  model
  |> cast(params, ~w(name username), [])
  |> unique_constraint(:username)
  |> validate_length(:username, min: 1, max: 20)
end

def registration_changeset(model, params) do
  model
  |> new_changeset(params)
  |> cast(params, ~w(password), [])
  |> validate_length(:password, min: 6, max: 100)
  |> put_pass_hash()
end
```

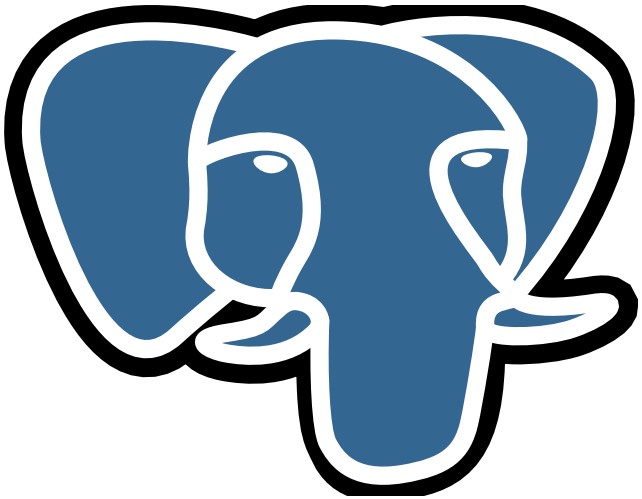
Changesets

```elixir
def new_changeset(model, params \\ %{}) do
  model
  |> cast(params, ~w(name username), [])
  |> unique_constraint(:username)
  |> validate_length(:username, min: 1, max: 20)
end

def registration_changeset(model, params) do
  model
  |> new_changeset(params)
  |> cast(params, ~w(password), [])
  |> validate_length(:password, min: 6, max: 100)
  |> put_pass_hash()
end
```

Changesets

```elixir
def new_changeset(model, params \\ %{}) do
  model
  |> cast(params, ~w(name username), [])
  |> unique_constraint(:username)
  |> validate_length(:username, min: 1, max: 20)
end

def registration_changeset(model, params) do
  model
  |> new_changeset(params)
  |> cast(params, ~w(password), [])
  |> validate_length(:password, min: 6, max: 100)
  |> put_pass_hash()
end
```

The right tool




Flappy Phoenix
Score: 3.7
Game Over!
You scored 3.7
Play again
Liveview!

```elixir
def handle_info(:tick, socket) do
    game = Game.update(socket.assigns.game)

    case game.state do
        :ok ->
            socket = schedule_tick(socket)
            {:noreply, assign(socket, game: game)}

        :end ->
            {:noreply, assign(socket, game: game)}
    end
end

def handle_event("keydown", _key, socket) do
    game = socket.assigns.game

    case game.state do
        :ok -> {:noreply, assign(socket, game: Game.flap(game))}
        _ -> {:noreply, new_game(socket)}
    end
end
```

Liveview!

Liveview!

```
  <div id="bird"
       style="left:<%= @game.bird.x %>vw;
              top:<%= @game.bird.y %>vh;
              transform: rotate(<
%= if @game.bird.velocity > 0, do: -25, else: 25 %>deg);">
    <img src="bird.png" />
  </div>

  <%= for pipe ← @game.pipes do %>
    <div class="pipe"
         style="left:<%= pipe.x %>vw; top:<%= pipe.y %>vh;">
      <img src="pipe.png" />
    </div>
  <% end %>
```

Liveview!

```
  <div id="bird"
       style="left:<%= @game.bird.x %>vw;
              top:<%= @game.bird.y %>vh;
              transform: rotate(<
%= if @game.bird.velocity > 0, do: -25, else: 25 %>deg);">
    <img src="bird.png" />
  </div>

  <%= for pipe ← @game.pipes do %>
    <div class="pipe"
         style="left:<%= pipe.x %>vw; top:<%= pipe.y %>vh;">
      <img src="pipe.png" />
    </div>
  <% end %>
```


Flappy Phoenix
Score: 3.7
Game Over!
You scored 3.7
Play again
Demo!

So we all go and do Elixir and Phoenix now?

Baggage


An exciting land


Thank you!
Tobias Pfeiffer
@PragTob
pragtob.info

Photo Attribution

- CC BY-ND 2.0
  - https://www.flickr.com/photos/mmmswan/8918529543/
- CC BY 2.0
  - https://flic.kr/p/eKGRRJ
- CC BY-NC 2.0
  - https://www.flickr.com/photos/-jule/2728475835/
  - https://flic.kr/p/emoKPd
- CC BY-NC-ND 2.0
  - https://flic.kr/p/eyC7ZT
  - https://www.flickr.com/photos/75487768@N04/14029339573/
  - https://flic.kr/p/bG2r2D
- CC BY-SA 2.0
  - https://commons.wikimedia.org/wiki/File:Heckert_GNU_white.svg
  - https://flic.kr/p/cEJDC3