THE ELIXIR PROGRAMING LANGUAGE

```
Functional
|> Concurrent
|> Pragmatic
|> Fun
```


desmotivaciones.es
-¿Por qué lo deseas tanto?
-Porque me dijeron que no lo lograría.

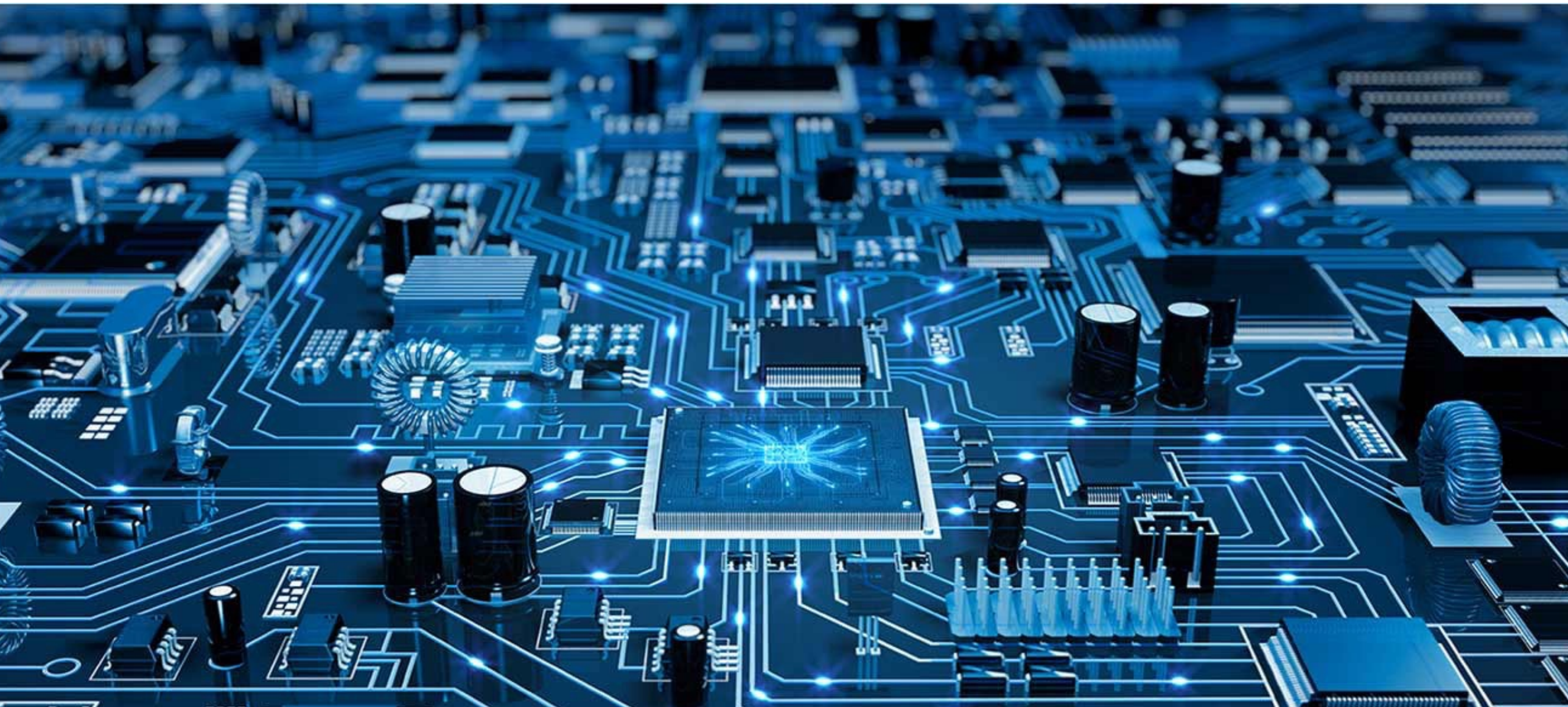
|> AGENDA

Conceptos generales
Programación funcional
- |> ¿Qué es?
- |> Inmutabilidad
- |> Principio de conservación de la información
- |> Funciones de primera clase
- |> Funciones de orden superior
- |> Nuestra amiga la recursión

Fundamentos de Elixir
- |> Historia
- |> iex
- |> Tipos y estructuras de datos
- |> Pattern matching
- |> Funciones anónimas
- |> Módulos

Ejercita tu mente

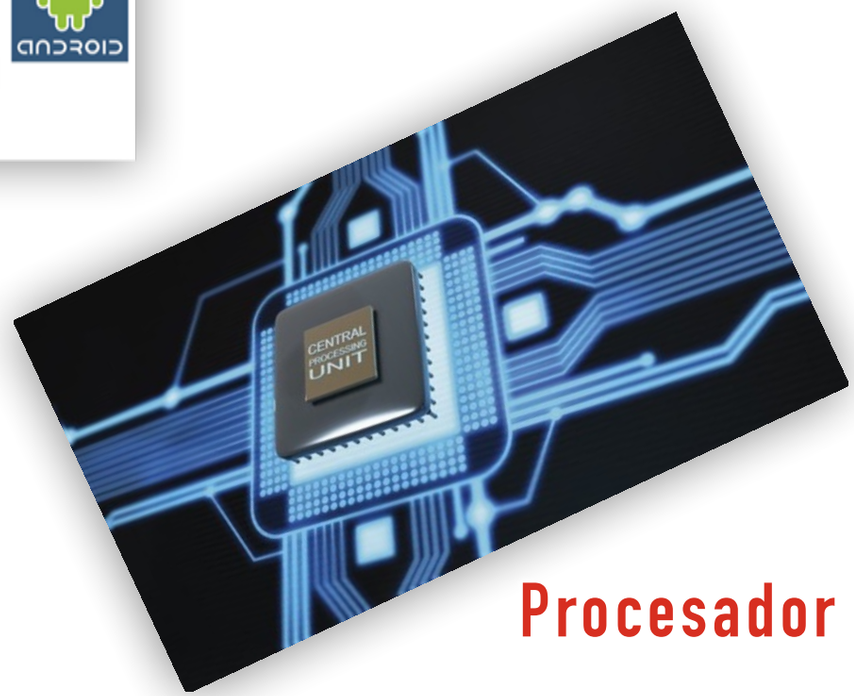
CONCEPTOS GENERALES



|> Organización y arquitectura de computadoras

Memoria

Sistema operativo



Procesador

|> Procesos

$3\mu s$

|> Procesos

$3\mu s$

|> Procesos

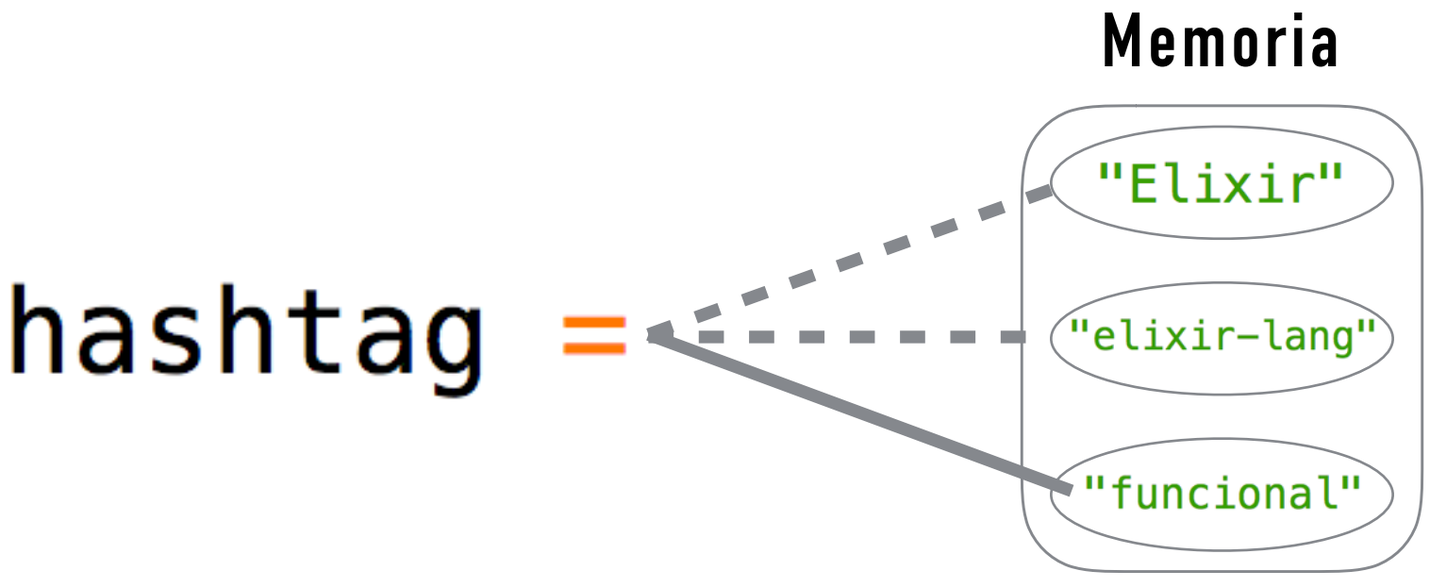
$3\mu s$

|> Máquina virtual

PROGRAMACIÓN FUNCIONAL

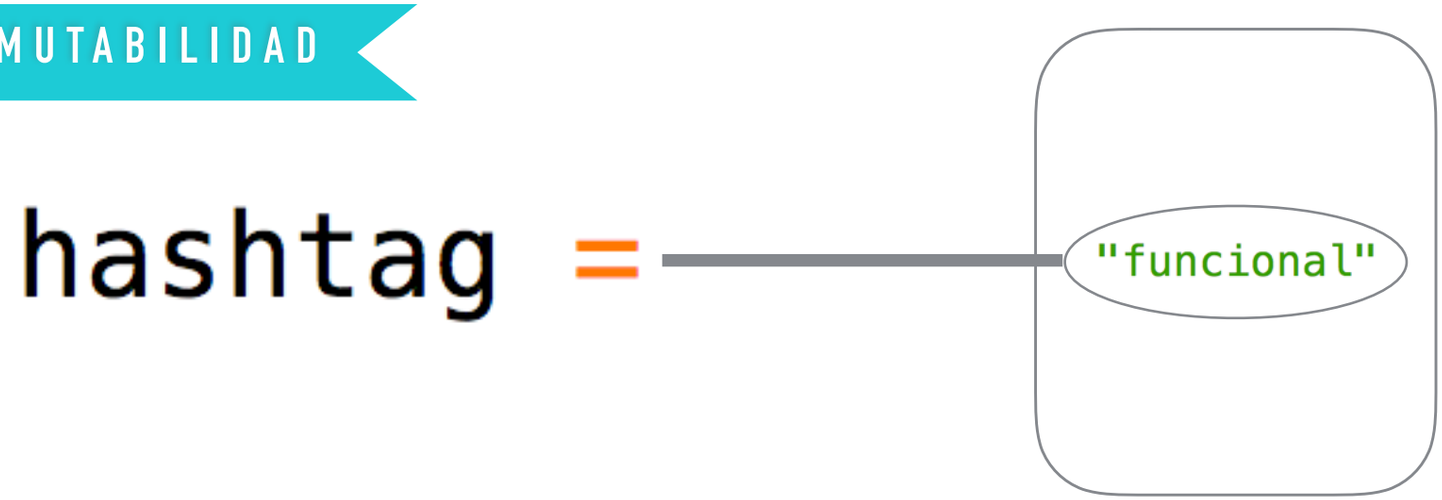
CONCEPTOS

|> Inmutabilidad

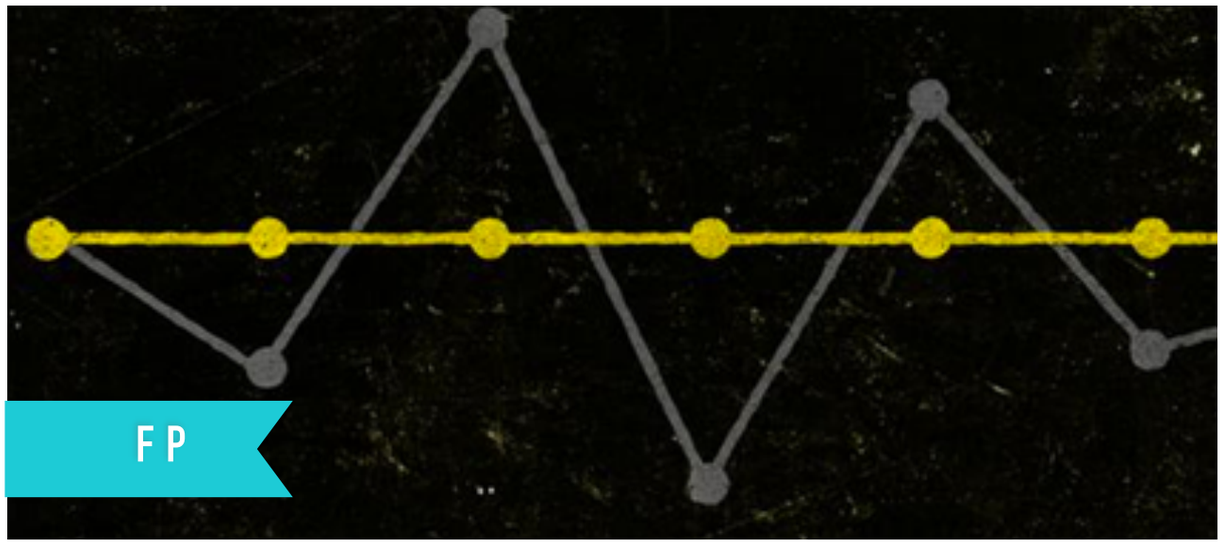



```
hashtag = "Elixir"
hashtag = "elixir-lang"
hashtag = "funcional"
```

MUTABILIDAD




```
hashtag = "Elixir"
hashtag = "elixir-lang"
hashtag = "funcional"
```



CONSERVACIÓN DE LA INFORMACIÓN

|> Conservación de la información

```
{
    created_at: "2016-04-07T18:45:43",
    text: "Aprendiendo #elixir-lang",
    id: 221643833908269060,
    id_str: "221643833908269057",
    in_reply_to_status_id: null,
    user:  {
        id: 352506187,
        id_str: "352506187",
        name: "Collin",
        screen_name: "Neises",
        location: "",
        followers_count: 23
    }
}
```

Tweet elasticsearch

```
{
    created_at: "2016-04-07T18:45:43",
    body: "Aprendiendo #elixir-lang",
    id: "221643833908269057",
    user:  {
        id: "352506187",
        name: "Collin",
        username: "Neises",
        followers: 23
    }
}
```

Tweet expuesto

|> Funciones

$$f(x) = 2x^2 - 5$$

$x = 5$

$$f(5) = (2)(5^2) - 5$$

$$f(5) = (2)(25) - 5$$

$$f(5) = 50 - 5$$

$$f(5) = 45$$

|> Funciones de primera clase

$$f(x) = 2x^2 - 5$$

$$y = f(x)$$

|> Funciones de orden superior

$$f(x) = \sqrt{x - 2} \qquad g(x) = \frac{1}{x}$$

$$(f \circ g)(x) = f(g(x)) = f(\frac{1}{x}) = \sqrt{\frac{1}{x} - 2}$$

|> Nuestra amiga la recursión

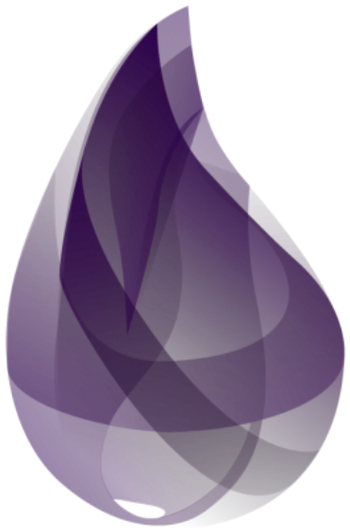
ELIXIR

```
    eval_dot_iex(state, path)
  end
end

defp eval_dot_iex(state, path) do
  try do
    code = File.read!(path)
    env  = :elixir.env_for_eval(state.env, file: path, line: 1)

    # Evaluate the contents in the same environment server_loop will run in
    {_result, binding, env, _scope} =
      :elixir.eval(String.to_char_list(code), state.binding, env)

    %{state | binding: binding, env: :elixir.env_for_eval(env, file: "iex", l
  catch
    kind, error ->
```

|> Historia

José Valim

Erlang



|> iex

```
➜  ~ iex
Erlang/OTP 18 [erts-7.2.1] [source] [64-bit] [smp:
 [dtrace]

Interactive Elixir (1.2.1) - press Ctrl+C to exit
iex(1)> list = [1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8]
iex(2)> Enum.map list, fn item -> item * item end
[1, 4, 9, 16, 25, 36, 49, 64]
iex(3)>
```

.exs

.ex

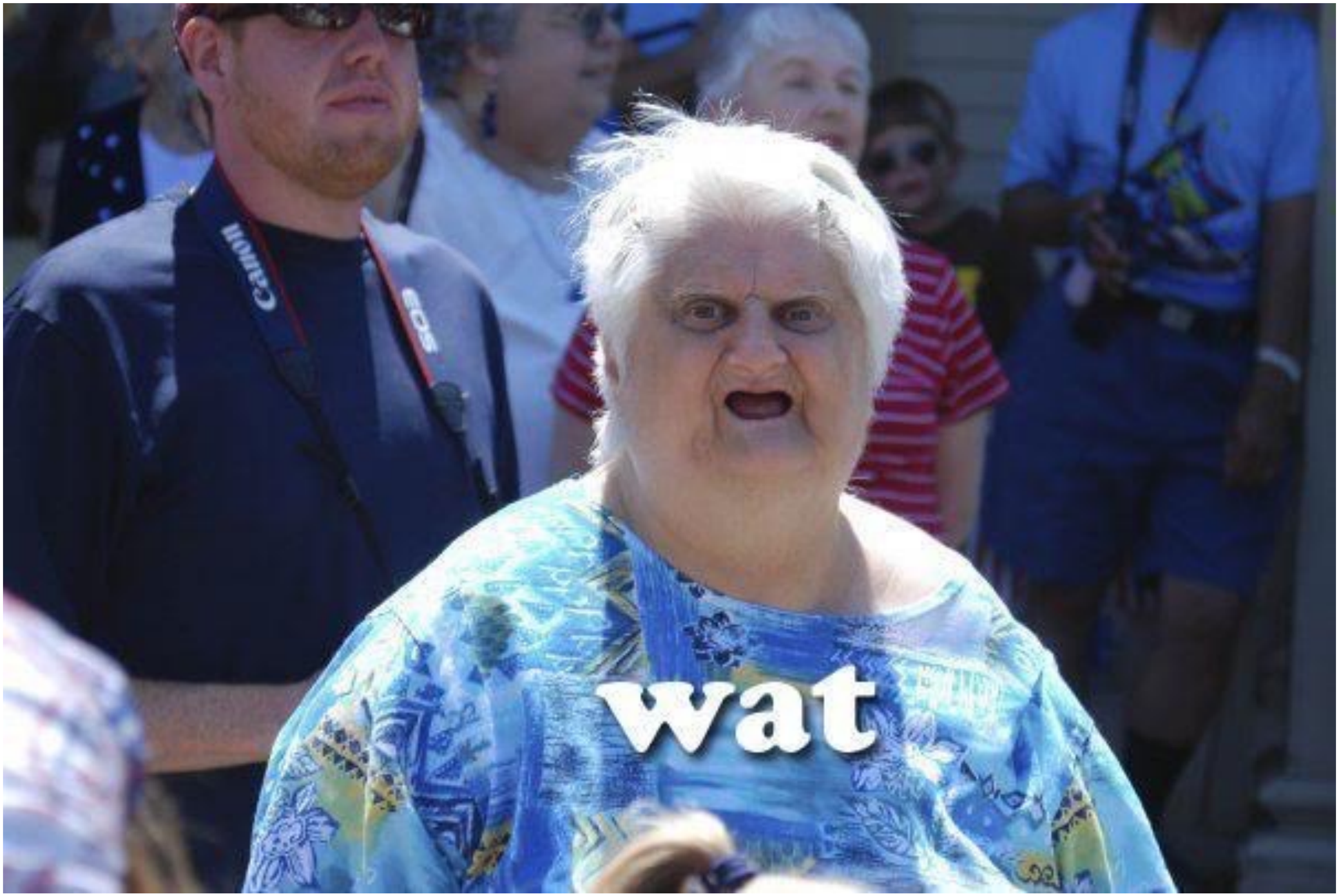
Extensiones

|> Pattern matching

```
x = 1
1 = x
```

```
list = [1, 2, 3, 4]
[1, 2, 3, 4] = list
```

```
successful_feed = { :ok, ["tweet1", "tweet2"] }
{ :ok, ["tweet1", "tweet2"] }
```


wat

|> Pattern matching

$$y = \frac{7x + 3}{5}$$

$$\frac{5}{7x + 3} = \frac{1}{y}$$

|> Funciones anónimas

```elixir
greet = fn name -> "Hi #{name}, how are you doing?" end

greet.("Tona")          # Hi Tona, how are you doing?
```

Función anónima simple

|> Funciones anónimas

```
successful_feed = { :ok, ["tweet1", "tweet2"]}
failed_feed = { :fnf, "Feed not found" }
weird_feed = { :mongo_error, "Internal Server Error" }

inspect_feed = fn
  { :ok, feed } -> IO.inspect(feed)
  { :fnf, msg } -> IO.inspect(msg)
  { _, msg } -> IO.inspect(msg)
end
```

Función con múltiples cuerpos

```
iex(7)> inspect_feed.(successful_feed)
["tweet1", "tweet2"]
["tweet1", "tweet2"]
iex(8)> inspect_feed.(failed_feed)
"Feed not found"
"Feed not found"
iex(9)> inspect_feed.(weird_feed)
"Internal Server Error"
"Internal Server Error"
```

|> Funciones anónimas

```elixir
greet = fn name ->
  "Hi #{name}, stop watching anime!!"
end

names = [
  "Jules",
  "Mario",
  "José",
  "Dann",
  "Porro"
]

Enum.map names, greet
```

Función de orden superior

```
[
  "Hi Jules, stop watching anime!!",
  "Hi Mario, stop watching anime!!",
  "Hi José, stop watching anime!!",
  "Hi Dann, stop watching anime!!",
  "Hi Porro, stop watching anime!!"
]
```

Salida

|> Funciones con nombre

```
defmodule Shape do
  def area(:triangle, base, high), do: (base * high) / 2
  def area(:rectangle, base, high), do: base * high
  def area(_, _, _), do: :not_allowed
end
```

Módulo Shape

> Funciones con nombre vs anónimas

```elixir
defmodule Shape do
  def area(:triangle, base, high), do: (base * high) / 2
  def area(:rectangle, base, high), do: base * high
  def area(_, _, _), do: :not_allowed
end
```

Módulo Shape

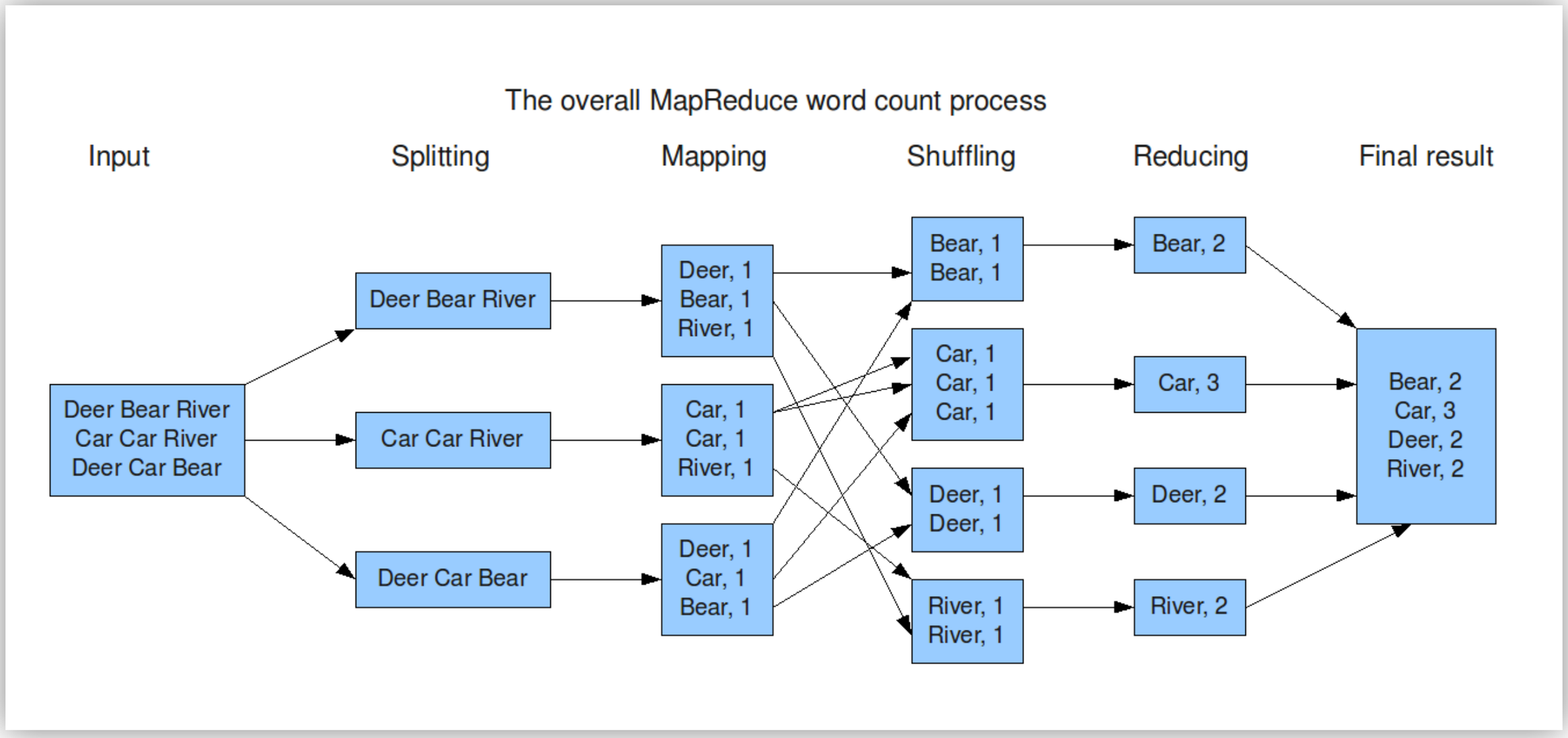
```elixir
area = fn
  (:triangle, base, high) -> (base * high) / 2
  (:rectangle, base, high) -> base * high
  (_, _, _ ) -> :not_allowed
end
```

Función anónima

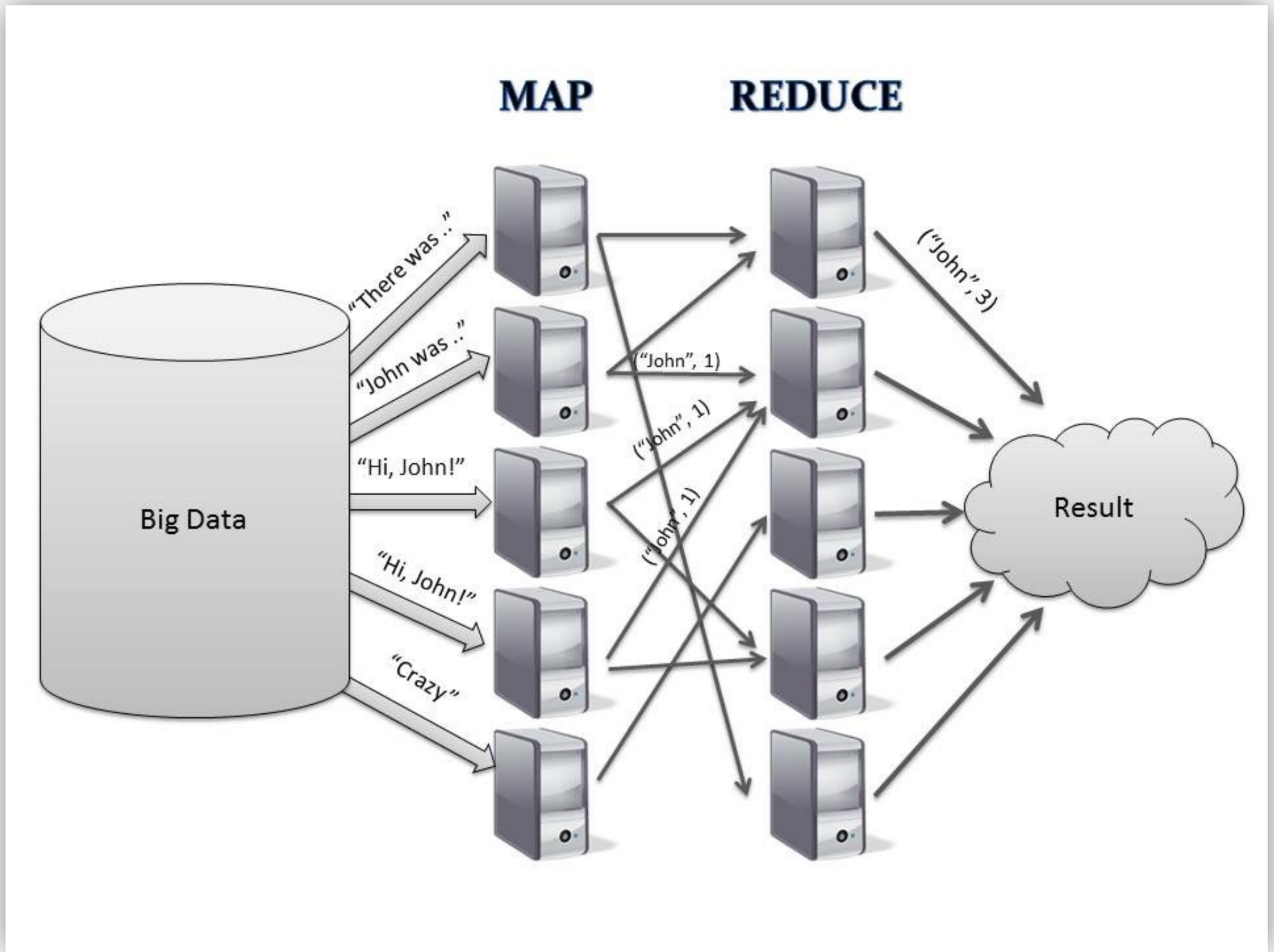
|> Mapas

```
feed_request = %{
  :query => %{
    :match => %{
        :body => ["elixir", "elixir-lang", "fp"],
        :username => ["josevalim", "phoenixcom"]
    },
    :like => %{
      :body => %{
        :_ctn => ["iex", "exs", "ex"]
      }
    },
    :range => %{
      :posted_time => %{
        :_lte => "2016.04.12",
        :_gte => "2016.04.10"
      }
    }
  }
}
```

|> Map-reduce


The overall MapReduce word count process
Input
Splitting
Mapping
Shuffling
Reducing
Final result
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, 1
Bear, 1
Car, 1
Car, 1
Car, 1
Deer, 1
Deer, 1
River, 1
River, 1
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

|> Map-reduce


MAP
REDUCE
Big Data
"There was .."
"John was .."
"Hi, John!"
"Hi, John!"
"Crazy"
("John", 1)
("John", 1)
("John", 1)
("John", 3)
Result

|> Map-reduce

```
username = "@john"

tweets = [
    "Hey @john, what is your name again?",
    "Trying to keep alive! =(",
    "@john do you remember when we were at high school?",
    "@thedevilwearsprada... the best band ever!!",
    "Goats on a boat! Great hit! Listen to it, @john"
]
```

|> Map-reduce

```
Enum.map(tweets, fn tweet -> String.split(tweet, " ") end)
  |> List.flatten
  |> Enum.filter(fn word -> word === username end)
  |> Enum.reduce(0, fn _, acc -> acc + 1 end)
  |> IO.inspect
```

```
[["Hey", "@john", "what", "is", "your", "name", "again?"],
 ["Trying", "to", "keep", "alive!", "=("],
 ["@john", "do", "you", "remember", "when", "we", "were", "at", "high",
  "school?"], ["@thedevilwearsprada...", "the", "best", "band", "ever!!"],
 ["Goats", "on", "a", "boat!", "Great", "hit!", "Listen", "to", "it,", "@john"]]
```

|> Map-reduce

```
Enum.map(tweets, fn tweet -> String.split(tweet, " ") end)
  |> List.flatten
  |> Enum.filter(fn word -> word === username end)
  |> Enum.reduce(0, fn _, acc -> acc + 1 end)
  |> IO.inspect
```

```
["Hey", "@john", "what", "is", "your", "name", "again?", "Trying", "to", "keep",
 "alive!", "=(", "@john", "do", "you", "remember", "when", "we", "were", "at",
 "high", "school?", "@thedevilwearsprada...", "the", "best", "band", "ever!!",
 "Goats", "on", "a", "boat!", "Great", "hit!", "Listen", "to", "it,", "@john"]
```

|> Map-reduce

```
Enum.map(tweets, fn tweet -> String.split(tweet, " ") end)
  |> List.flatten
  |> Enum.filter(fn word -> word === username end)
  |> Enum.reduce(0, fn _, acc -> acc + 1 end)
  |> IO.inspect
```

```
["@john", "@john", "@john"]
```

|> Map-reduce

```elixir
username = "@john"

tweets = [
    "Hey @john what is your name again?",
    "Trying to keep alive! =(",
    "@john do you remember when we were at high school?",
    "@thedevilwearsprada... the best band ever!!",
    "Goats on a boat! Great hit! Listen to it, @john"
]

Enum.map(tweets, fn tweet -> String.split(tweet, " ") end)
  |> List.flatten
  |> Enum.filter(fn word -> word === username end)
  |> Enum.reduce(0, fn _, acc -> acc + 1 end)
  |> IO.inspect
```

|> Map-reduce

```
'use strict';

let username = '@john'

let tweets = [
    'Hey @john what is your name again?',
    'Trying to keep alive! =(',
    '@john do you remember when we were at high school?',
    '@thedevilwearsprada... the best band ever!!',
    'Goats on a boat! Great hit! Listen to it, @john'
]

let user_appearances = 0;

for (let tweet of tweets) {
    let splited_tweet = tweet.split(" ");

    for (let word of splited_tweet) {
        if (word === username) {
            user_appearances++;
        }
    }
}

console.log(user_appearances);
```

|> Ejercita tu mente

$$0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$$

$$F_n = F_{n-1} + F_{n-2}$$

Código fuente en:

GitHub jovannypcg/learning_elixir