# Functional Programming With Elixir

Nguyen Anh Tien

# Table of Contents

# What is Elixir ?

- Functional, Concurrent, General Purpose
- Created by José Valim @ Plataformatec
  - Former Rails Core Team member
  - Author of **Crafting Rails 4 Application**
- Run on Erlang VM
- Stable release: 1.3.1
- Used by: Pinterest, Dockyard, Bleacher Report, ...


# What is Elixir ? - Erlang

- Developed by Ericsson, ~1986
- Same characteristic as Elixir
- Used in Telephone Applications
- BEAM VM
  - A virtual machine to run Erlang
  - Interfaces to the “outside” world
- Backed of WhatApp, Amazon’s SimpleDB, ...
  - http://highscalability.com/blog/2014/2/26/the-whatsapp-architecture-facebook-bought-for-19-billion.html
  - https://blog.whatsapp.com/196/1-million-is-so-2011
- Battle-tested !

# Functional vs Imperative Programming

- Modules
- Immutable
- Pure functions
  - No side-effect
- Stateless ?
- Declarative
  - Expressions
- Erlang, Haskell, Clojure, ...

- Objects
- Mutable
- Methods
  - Can have side-effect
- Stateful
- Imperative
  - Statements
- Python, Ruby, Java

# Elixir’s Syntax - Types

| | |
|---|---|
| **Integer** | 1234 0xcafe 0177 0b100 10_000 |
| **Float** | 1.0 3.1415 6.02e23 |
| **Atom** | :foo :me@home :"with spaces" |
| **Tuple** | { 1, 2, :ok, "xy" } |
| **List** | [ 1, 2, 3 ] or [head \| tail] |
| **Keyword List** | [ a: "Foo", b: 123 ] |
| **Map** | %{ key => value, key => value } |
| **Truth** | true, false, nil |
| **Range** | a..b |

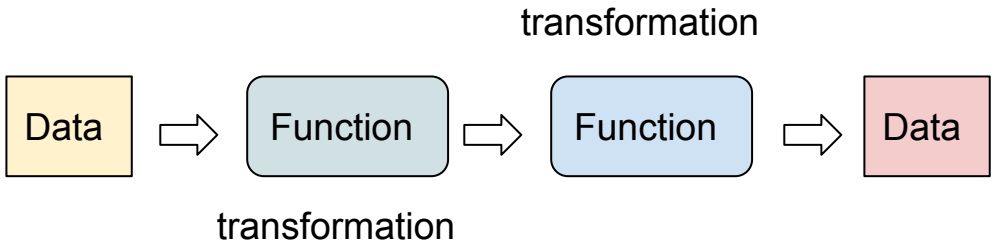
# Elixir’s Syntax - Ruby like

```
1  defmodule LobbyDemo.RegistrationController do
2    use LobbyDemo.Web, :controller
3    alias LobbyDemo.User
4
5    def new(conn, _params) do
6      changeset = User.changeset(%User{})
7      render conn, changeset: changeset
8    end
9
10 end
```

# Elixir’s Syntax

- First class functions - High order functions
  - Functions that can either take other functions as arguments or return them as results
  - Macro
  - Homoiconicity
- Anonymous functions

```
iex(7)> Enum.map([[1, 2], [3, 4], [5, 6]], fn(x) -> Enum.at(x, 1) end)
[2, 4, 6]
```



# Elixir’s Syntax - Pipe operator (1)

- |>
  - passes the result of an expression as the first parameter of another expression.
- First argument: Data to transform
- Function composing
- Normal (bad) code

```
s = String.reverse("abba radar")
s = String.upcase(s)
s = String.split(s)
["RADAR", "ABBA"]
```

# Elixir’s Syntax - Pipe operator (2)

- Nested code

```
s = String.split(String.upcase(String.reverse("abba radar")))
```

- Pipe code

```
"abba radar"
|> String.reverse
|> String.upcase
|> String.split
```

# Elixir’s Syntax - Pattern Matching (1)

- the = operator is actually a match operator -> binding + rebinding

```
iex> x = 1
1
iex> 1 = x
1
iex> 2 = x
** (MatchError) no match of right hand side value: 1
iex> x = 2
2
iex> ^x = 3
** (MatchError) no match of right hand side value: 3
```

# Elixir’s Syntax - Pattern Matching (2)

- Function signature matching

```
defp strip_unsafe_description(model, %{"description" => nil}) do
  model
end

defp strip_unsafe_description(model, %{"description" => description}) do
  {:safe, clean_description} = Phoenix.HTML.html_escape(description)
  model |> put_change(:description, clean_description)
end

defp strip_unsafe_description(model, _) do
  model
end
```

# Elixir’s Syntax - Looping

- Looping through recursion, tail-call

```
def sum_list([head | tail], accumulator) do
  sum_list(tail, head + accumulator)
end

def sum_list([], accumulator) do
  accumulator
end
```

- Iteration
  - Enum.reduce
  - Enum.map
  - Enum.filter
  - ...

# Why Elixir ? Processes

- Processes
  - Lightweight, low cost (~1KB)
  - Isolated, concurrent
  - Thousand of processes
- Communication by message-passing
  - Inbox

```
iex(32)> spawn fn -> IO.puts 1 + 2 end
3
#PID<0.104.0>
```

```
iex(34)> pid = spawn fn -> receive do {:hello, msg} -> IO.puts msg end end
#PID<0.115.0>
iex(35)> send pid, {:hello, "Hi !"}
Hi !
{:hello, "Hi !"}
```

- Connect to other node in another computer
- -> Distributed

# Why Elixir ? Fast (1)

- Data copying
  - -> inefficient ?
- Garbage Collector
  - Processes have separated heap
  - No synchronisation
- **Robert Virding - Hitchhiker's Tour of the BEAM**
  - https://www.youtube.com/watch?v=_Pwlvy3zz9M

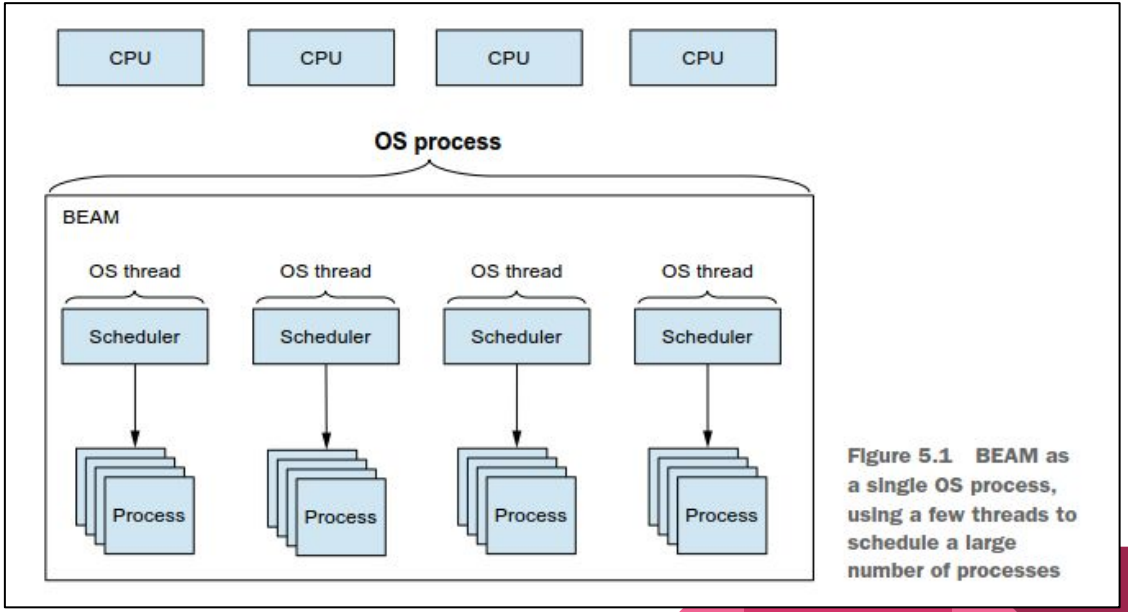



Figure 5.1 BEAM as a single OS process, using a few threads to schedule a large number of processes

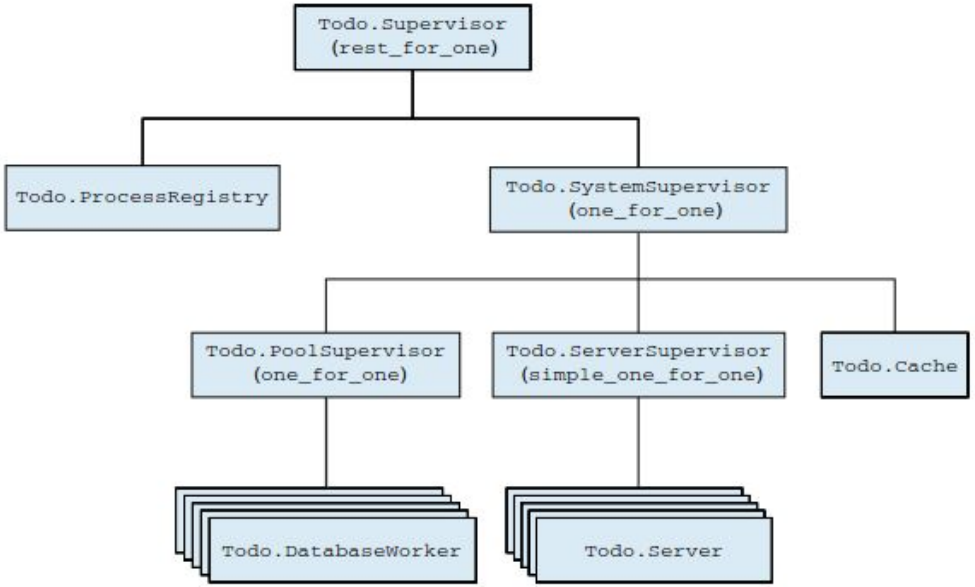
# Why Elixir ? Fast (2)

- Let’s see some benchmark
  - https://github.com/mroth/phoenix-showdown

| Framework | Throughput (req/s) | Latency (ms) | Consistency (σ ms) |
|---|---|---|---|
| Gin | 51483.20 | 1.94 | 0.63 |
| Phoenix | 43063.45 | 2.82 | *(1)* 7.46 |
| Express Cluster | 27669.46 | 3.73 | 2.12 |
| Martini | 14798.46 | 6.81 | 10.34 |
| Sinatra | 9182.86 | 6.55 | 3.03 |
| Express | 9965.56 | 10.07 | 0.95 |
| Rails | 3274.81 | 17.25 | 6.88 |
| Plug *(1)* | 54948.14 | 3.83 | 12.40 |
| Play *(2)* | 63256.20 | 1.62 | 2.96 |

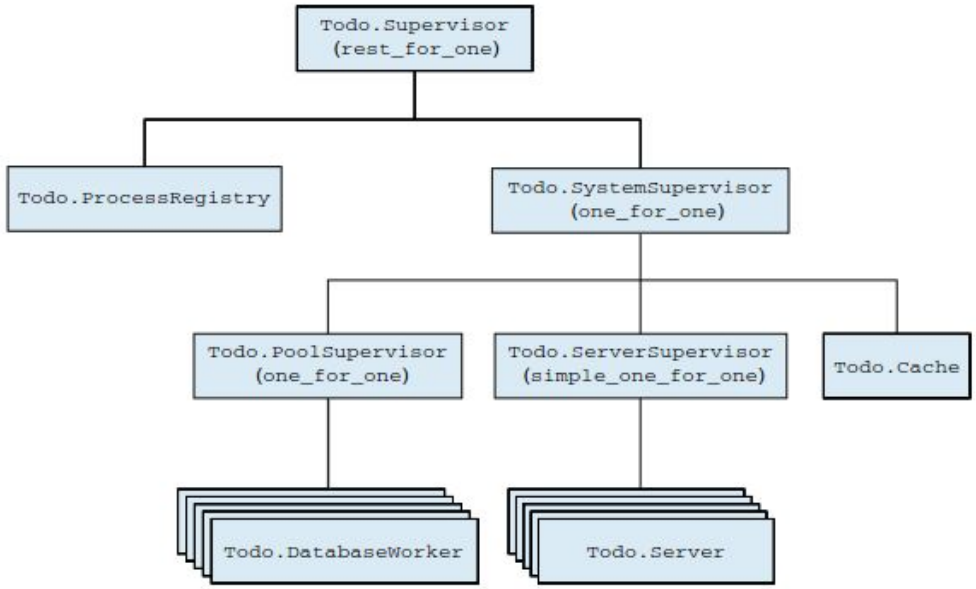
# Why Elixir ? Fault-tolerant (1)

- Todo Server Example
  - Everyone have a Todo
  - Todo: List of todo items
- Use cache when DB isn't reachable
- One user's todo list do not affect other's todo list



# Why Elixir ? Fault-tolerant (2)

- “Let it crash”
- Link and Monitor
- OTP
- Supervisor Trees - GenServer
- Restart Strategy
  - one_for_one
  - simple_one_for_one
  - rest_for_one



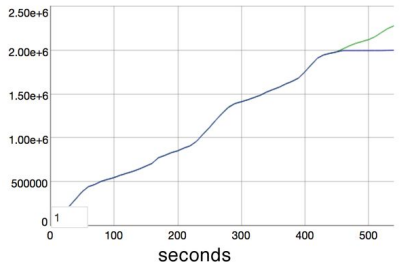
# Elixir’s Ecosystem

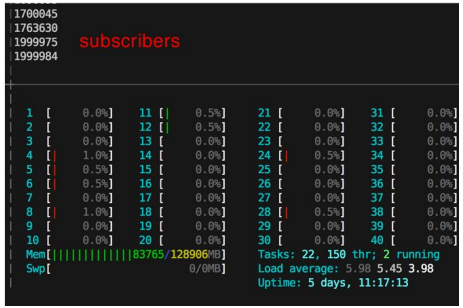
- Hex - **The package manager for the Erlang ecosystem**
- Mix - **Elixir’s build tool**
  - = Rake + Bundler + RubyGems
- Call outside program via Ports or NIF

# Elixir’s Ecosystem

- Productive - Reliable - Fast
- MVC model, Plug based
- Live reload coding
- Websocket integrated
  - http://www.phoenixframework.org/blog/the-road-to-2-million-websocket-connections
  - https://dockyard.com/blog/2016/08/09/phoenix-channels-vs-rails-action-cable
- More and more app migration from Rails to Phoenix
  - https://speakerdeck.com/bcardarella/from-rails-to-phoenix

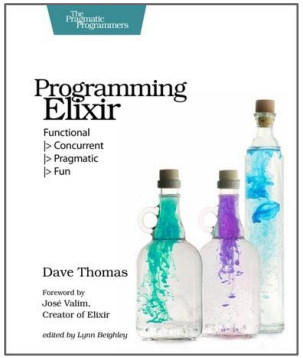
Phoenix Framework

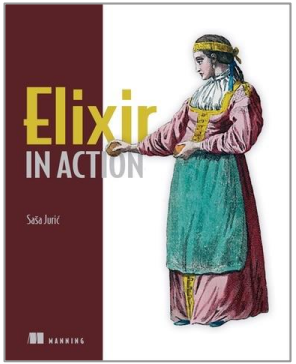


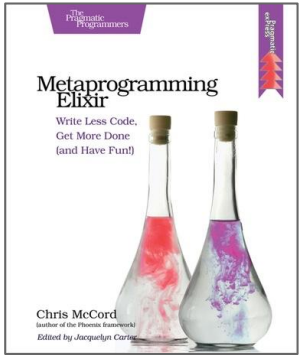



# Learning Resources

- Elixir Getting Started
- Elixir Schools
- ElixirConf @ youtube
- #elixir @ confreaks.tv
- exercism.io
- vietnamrb.slack.com
- Elixir-lang.slack.com
- Erlang stuffs !
- ...

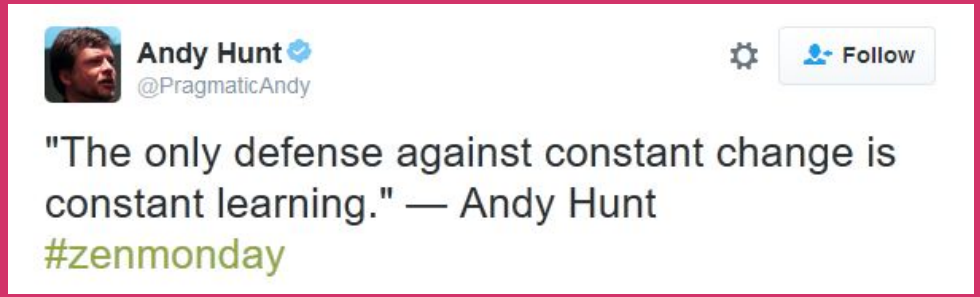



# Q & A

# Thank you !


Andy Hunt
@PragmaticAndy
Follow
"The only defense against constant change is constant learning." — Andy Hunt
#zenmonday