

# The Power of ~~LOVE~~... Actors

Bernhard Stöcker

Recognizer Group GmbH

*bernhard.stoecker@recognizer.com*

November 2, 2016

# Overview

- 1 What are Actors?
- 2 WhatsApp
- 3 Actors in Elixir
- 4 Actors in Scala

# What are Actors?

Act-or: to act: **"to do something for a particular purpose or to solve a problem"** (From the Cambridge dictionary)

# What are Actors?

Wikipedia:

- A mathematical model of concurrent computation
- Actors can hold and modify private state
- Affects each other through messages only
- In response to a message that it receives, an actor can:
  - Make local decisions
  - Create more actors
  - Send more messages
  - Respond to the incoming message

# WhatsApp???



# WhatsApp

- Every user has an actor representing her
- When sending a message my actor sends a message to all related users
- The users receiving a message ensure the message is delivered.

# Actors in Elixir

```
bernhard@bernhard's-thinkpad ~ $ iex
Erlang/OTP 19 [erts-8.1] [source-4cc2ce3] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-po

Interactive Elixir (1.3.3) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> spawn fn -> IO.puts("Hello World") end
Hello World
#PID<0.83.0>
iex(2)> █
```

# Actors in Elixir

```
bernhard@bernhard's-thinkpad ~ $ iex
Erlang/OTP 19 [erts-8.1] [source-4cc2ce3] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-po

Interactive Elixir (1.3.3) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> func = fn ->
... (1)> receive do
... (1)>   name -> IO.puts("Hello #{name}")
... (1)> end
... (1)> end
#Function<20.52032458/0 in :erl_eval.expr/5>
iex(2)> pid = spawn func
#PID<0.88.0>
iex(3)> send(pid, "Kira")
Hello Kira
"Kira"
iex(4)> █
```



# Actors in Elixir

```
iex(1)> defmodule Value do
... (1)> def current(x) do
... (1)> receive do
... (1)> add -> IO.puts(x + add); current(x + add)
... (1)> end
... (1)> end
... (1)> end
{:module, Value,
 <<70, 79, 82, 49, 0, 0, 5, 40, 66, 69, 65, 77, 69, 120, 68, 99, 0, 0, 0, 148,
 131, 104, 2, 100, 0, 14, 101, 108, 105, 120, 105, 114, 95, 100, 111, 99, 1,
 95, 118, 49, 108, 0, 0, 0, 4, 104, 2, ...>>, {:current, 1}}
iex(2)> pid = spawn fn -> Value.current(0) end
#PID<0.95.0>
iex(3)> send pid, 17
17
17
iex(4)> send pid, 16
33
16
iex(5)> send pid, 9
42
9
iex(6)> █
```

# Actors in Elixir

```
1 ▾ defmodule Stack do
2   use GenServer
3
4   # Callbacks
5
6   def handle_call(:pop, _from, []) do
7     {:reply, nil, []}
8   end
9   def handle_call(:pop, _from, [h | t]) do
10    {:reply, h, t}
11  end
12
13  def handle_call(:top, _from, []) do
14    {:reply, nil, []}
15  end
16  def handle_call(:top, _from, [h | t]) do
17    {:reply, h, [h | t]}
18  end
19
20  def handle_cast({:push, item}, state) do
21    {:noreply, [item | state]}
22  end
23 end
24
```

# Actors in Elixir

```
^Cbernhard@bernhard's-thinkpad ~/Dokumente/Officetalk $ iex
Erlang/OTP 19 [erts-8.1] [source-4cc2ce3] [64-bit] [smp:4:4] [async-threads:0] [hipe-4] [dtrace-0.0]

Interactive Elixir (1.3.3) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> c("stack.ex")
[Stack]
iex(2)> {:ok, pid} = GenServer.start_link(Stack, ["Hello", "World"])
{:ok, #PID<0.89.0>}
iex(3)> pid
#PID<0.89.0>
iex(4)> GenServer.call(pid, :pop)
"Hello"
iex(5)> GenServer.call(pid, :pop)
"World"
iex(6)> GenServer.call(pid, :top)
nil
iex(7)> GenServer.cast(pid, {:push, "Hello Kira"})
:ok
iex(8)> GenServer.call(pid, :top)
"Hello Kira"
iex(9)> 
```

# Verbatim

## Example (Theorem Slide Code)

```
\begin{frame}  
\frametitle{Theorem}  
\begin{theorem}[Mass--energy equivalence]  
$E = mc^2$  
\end{theorem}  
\end{frame}
```

# Figure

Uncomment the code on this slide to include your own image from the same directory as the template .TeX file.

# Citation

An example of the `\cite` command to cite within the presentation:

This statement requires citation [?].

# The End