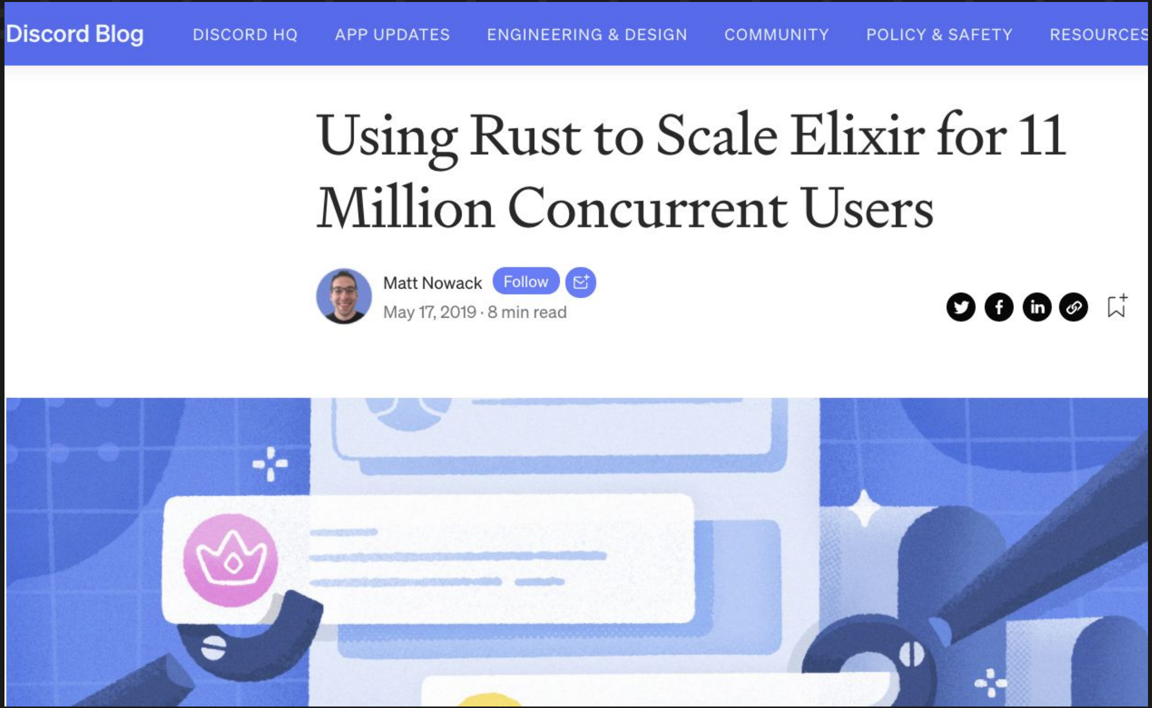
Erlang Solutions

Introduction to Advanced Elixir

???
elixir
???

“Elixir is a functional language; its data structures are immutable. This is great for reasoning about code and supporting the massive concurrency you enjoy when you write Elixir.”



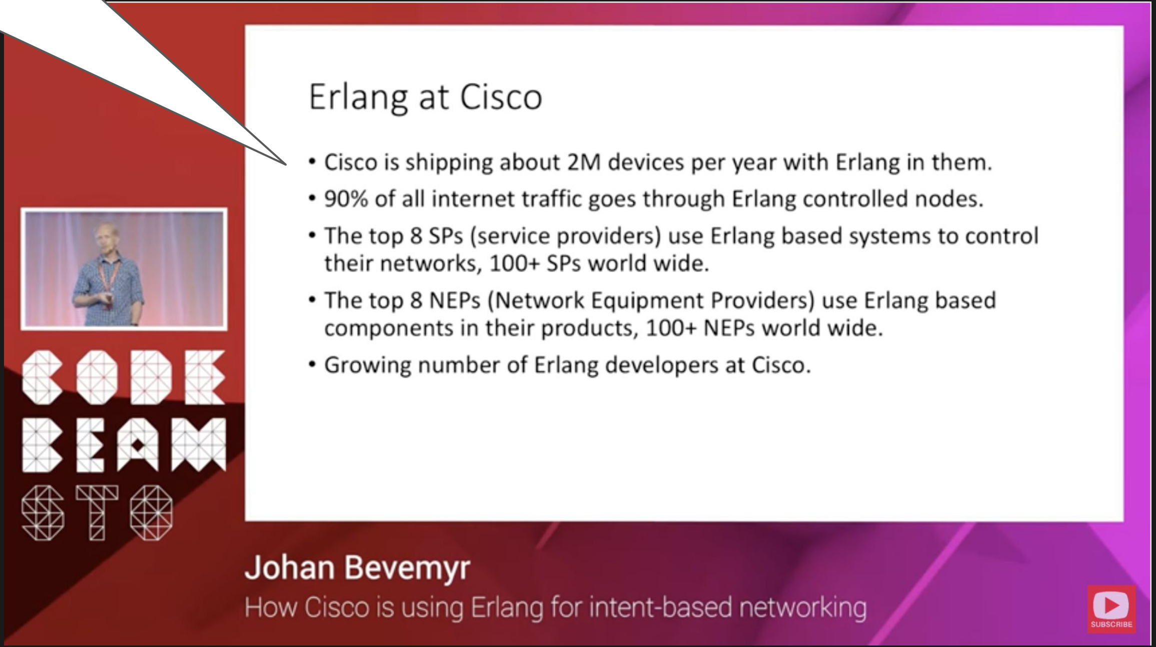
Using Rust to Scale Elixir for 11 Million Concurrent Users

Matt Nowack

May 17, 2019 · 8 min read

https://blog.discord.com/using-rust-to-scale-elixir-for-11-million-concurrent-users-c6f19fc029d3

- Cisco is shipping about 2M devices per year with Erlang in them.
- 90% of all internet traffic goes through Erlang controlled nodes.



Erlang at Cisco

- Cisco is shipping about 2M devices per year with Erlang in them.
- 90% of all internet traffic goes through Erlang controlled nodes.
- The top 8 SPs (service providers) use Erlang based systems to control their networks, 100+ SPs world wide.
- The top 8 NEPs (Network Equipment Providers) use Erlang based components in their products, 100+ NEPs world wide.
- Growing number of Erlang developers at Cisco.

https://www.youtube.com/watch?v=077-XJv6PLQ

Bleacher Report

Key benefits

1. A significant reduction in code complexity
2. A significant decrease in time taken to develop the code
3. 10x reduction in the time it takes to update the site
4. 150 servers were reduced to just 8
5. The system easily handles over 200 million push notifications per day

https://www.erlang-solutions.com/case-studies/bleacher-report-case-study/

“At WhatsApp, we use Erlang for pretty much everything.
We’re essentially running on Erlang.”

https://www.youtube.com/watch?v=6WbuboDwwjw

“ When I started working on Elixir, I personally had the ambition of using it for building scalable and robust web applications. However, I didn’t want Elixir to be tied to the web. My goal was to design an extensible language with a diverse ecosystem. Elixir aims to be a general purpose language and allows developers to extend it to new domains. “

José Valim

Creator of Elixir

Erlang Solutions

“ Given Elixir is built on top of Erlang and Erlang is used for networking and distributed systems, Elixir would naturally be a good fit in those domains too, as long as I didn’t screw things up. The Erlang VM is essential to everything we do in Elixir, which is why compatibility has become a language goal too.“

José Valim

Creator of Elixir

Erlang Solutions

" I also wanted the language to be productive, especially by focusing on the tooling. Learning a functional programming language is a new endeavor for most developers. Consequently their first experiences getting started with the language, setting up a new project, searching for documentation, and debugging should go as smoothly as possible."

José Valim

Creator of Elixir

Business Outcomes from Erlang/Elixir/OTP

2x FASTER development of new services

10x BETTER services, down-time less than 5 minutes/year

10x SAFER services that are very hard to hack or crash

10x MORE users and transactions - within milliseconds

10x LESS costs and energy consumption

Robert Virding, co-creator of Erlang/OTP
"Any sufficiently complicated concurrent program in another language contains an ad hoc informally-specified bug-ridden slow implementation of half of Erlang."

References: Cesarini (2019), CVE (2021), Virding (2008)

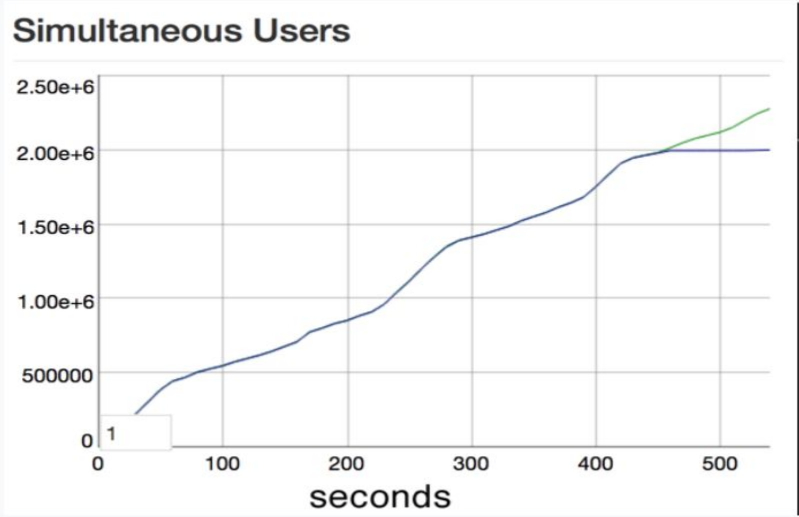
Erlang/Elixir/OTP is the Right Tool for the Job in Fintech!

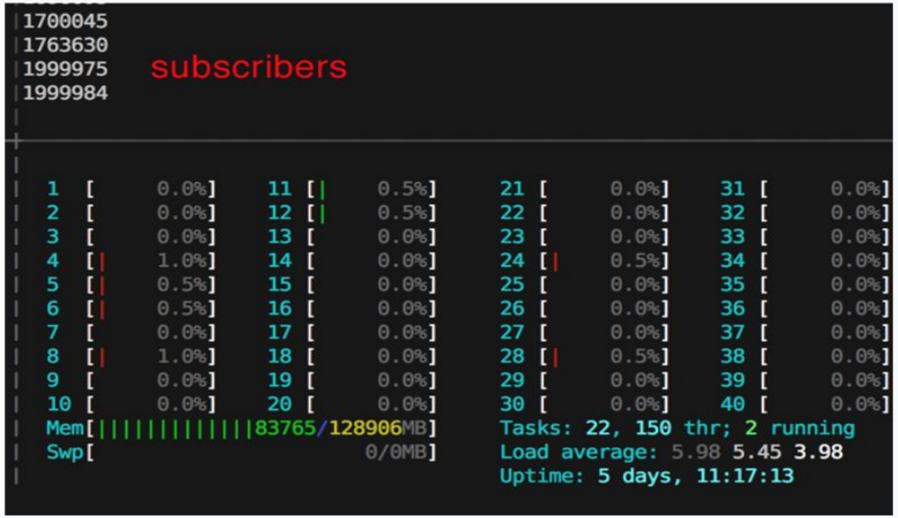
2x FASTER, 10x BETTER, 10x SAFER, 10x MORE for 10x LESS

References: 5 Erlang and Elixir Use Cases In FinTech; Kivra Case Study; Æternity Case Study

The Road to 2 Million Websocket Connections in Phoenix

Posted on November 3rd, 2015 by Gary Rennie



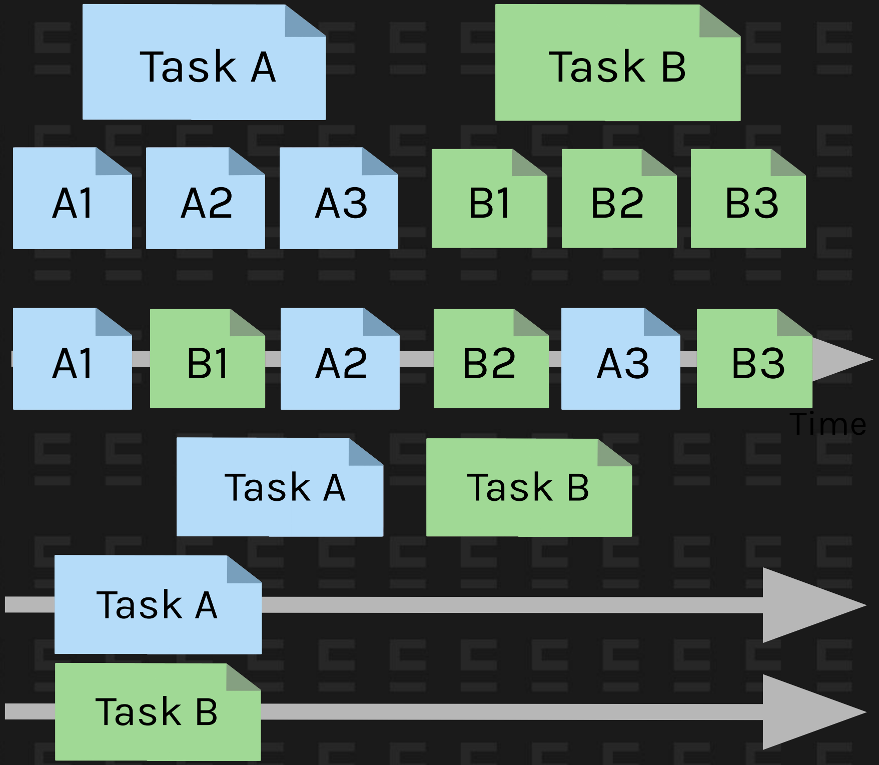


```
1700045
1763630
1999975
1999984

  1  [         0.0%]   11 [|        0.5%]   21 [         0.0%]   31 [         0.0%]
  2  [         0.0%]   12 [|        0.5%]   22 [         0.0%]   32 [         0.0%]
  3  [         0.0%]   13 [         0.0%]   23 [         0.0%]   33 [         0.0%]
  4  [|        1.0%]   14 [         0.0%]   24 [|        0.5%]   34 [         0.0%]
  5  [|        0.5%]   15 [         0.0%]   25 [         0.0%]   35 [         0.0%]
  6  [|        0.5%]   16 [         0.0%]   26 [         0.0%]   36 [         0.0%]
  7  [         0.0%]   17 [         0.0%]   27 [         0.0%]   37 [         0.0%]
  8  [|        1.0%]   18 [         0.0%]   28 [|        0.5%]   38 [         0.0%]
  9  [         0.0%]   19 [         0.0%]   29 [         0.0%]   39 [         0.0%]
  10 [         0.0%]   20 [         0.0%]   30 [         0.0%]   40 [         0.0%]
  Mem[||||||||||||||83765/128906MB]        Tasks: 22, 150 thr; 2 running
  Swp[                     0/0MB]          Load average: 5.98 5.45 3.98
                                           Uptime: 5 days, 11:17:13
```

If you have been paying attention on Twitter recently, you have likely seen some increasing numbers regarding the number of simultaneous connections the Phoenix web framework can handle. This post documents some of the techniques used to perform the benchmarks.

Concurrency

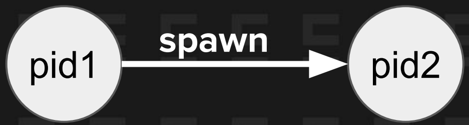


- Concurrency happens when your code is running in different processes
- Control of Concurrency is key to scale
- Concurrent solutions can exploit the underlying system’s parallelism, if present
- Parallelism can speed up execution

Erlang Solutions

Processes

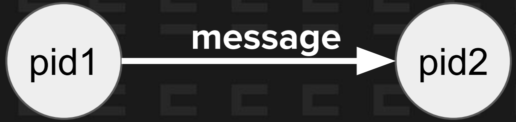
- **Pid1** executes **spawn** (pid = process identifier)
- Returns **pid2**
- **Pid2** runs **module.function(args)** or anonymous function
- The process terminates abnormally when run-time errors occur
- The process terminates normally when there is no more code to execute



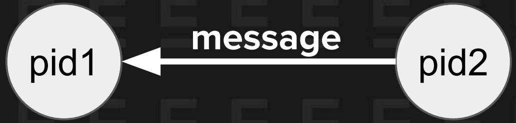
spawn(module, function, args)
spawn(fn -> ... end)

Messages

- **Pid1** sends message to **pid2**
- **Pid1** receives message from **pid2 (or any process)**
- **self()** is the pid of the caller process



Pid1:
send(pid2, message)



Pid1:

```
receive do
  message -> ...
end
```

Erlang Solutions

A concurrency example

- Cash desk service
- The customer wants to buy items
- Requests are sent to the cash desk process (product, amount)
- The cash desk process checks the prices and adds the item’s price to the total bill
- When the customer is done, the total amount to be paid is returned

Cashdesk module (v1)

```elixir
defmodule Cashdesk do
  @prices %{"flour" => 100, "egg" => 45,
"toilet paper" => 500}

  def start do
    spawn(fn -> init() end)
  end

  def init do
    loop(%{total: 0, prices: @prices})
  end

  def buy(cashdesk, product, amount) do
    send(cashdesk, {:buy, self(), product,
amount})
    receive do
      response ->
        response
    after 5000 ->
        :cashdesk_closed
    end
  end
```

```elixir
  def done(cashdesk) do
    send(cashdesk, {:done, self()})
    receive do
      total ->
        total
    end
  end

  defp loop(%{total: total, prices: prices} =
state) do
    receive do
      {:buy, customer, product, amount} ->
        send(customer, :ok)
        pay = amount * prices[product]
        loop(%{state | total: total + pay})
      {:done, customer} ->
        send(customer, total)
        loop(%{state | total: 0})
    end
  end
end
```

Erlang Solutions

Cashdesk module (v2)

```elixir
defp loop(%{total: total, prices: prices} = state) do
  receive do
    {:buy, customer, product, amount} ->
      case prices[product] do
        nil ->
          send(customer, :not_available)
          loop(state)
        price ->
          pay = amount * price
          send(customer, :ok)
          loop(%{state | total: total + pay})
      end
    {:done, customer} ->
      send(customer, total)
      loop(%{state | total: 0})
  end
end
```

Erlang Solutions

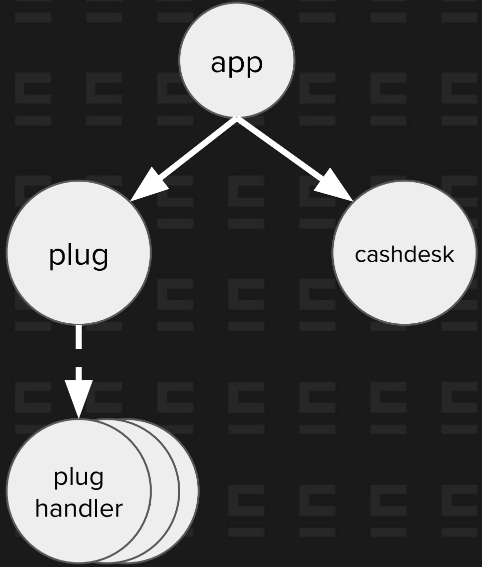
Cashdesk module (v3)

```
  def start do
    pid = spawn(fn -> init() end)
    Process.register(pid, :cashdesk)
  end
...
  def buy(product, amount) do
...
  def done() do
...
```

Erlang Solutions

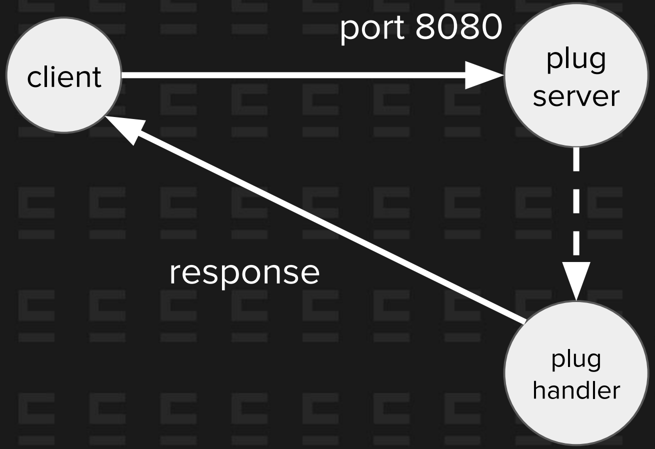
Process supervision

- Application links and monitors the included processes
- Connected processes are notified of crashes and can take action



A mix application

- Client initiates HTTP request to the server listening on port 8080
- Server spawns a process handling client request



Erlang Solutions

A mix application

- HTTP server accepting JSON requests for ‘buy’ and ‘done’
- Create a new application
  - `mix new cashdesk --sup`
  - `cd cashdesk`
- Add dependencies to *mix.exs*

```
defp deps do
  [
    {:plug_cowboy, "~> 2.0"},
    {:jason, "~> 1.2"}
  ]
end
```

- Add processes to the supervision tree in *application.ex*

```
{Plug.Cowboy, scheme: :http, plug: Cashdesk.Router, options: [port: 8080]},
Cashdesk
```

- `mix deps.get`
- `iex -S mix`

Demo

- Ping
- Curl requests (verbose: -v)
  - `curl -X POST -H "Content-Type: application/json" -d '{"product": "egg", "amount": 1}' localhost:8080/buy`
  - `curl localhost:8080/done`
- Invalid amount type
  - `curl -X POST -H "Content-Type: application/json" -d '{"product": "egg", "amount": "a"}' localhost:8080/buy`
- Fix the error with exception handling

```
try do
  pay = amount * price
  send(customer, :ok)
  loop(%{state | total: total + pay})
rescue
  _ ->
    send(customer, :invalid_amount)
    loop(state)
end
```

Erlang Solutions

Unit tests

- Test files (with extension .exs), preferably using similar paths as modules, but under /test folder
- test macro for each test case in module
- assert / refute macros to evaluate test results
- Showing details on failure
- Examples in doctest cases are executed and included in code documentation

```
defmodule CashdeskTest do
  use ExUnit.Case
  doctest Cashdesk

  test "buy" do
    Cashdesk.buy("flour", 2)
    Cashdesk.buy("egg", 1)
    assert Cashdesk.done() == 245
  end
end
```

- `mix test`

Dialyzer

- Code analysis
- Detect potential errors

```
defp deps do
  [
    {:plug_cowboy, "~> 2.0"},
    {:jason, "~> 1.2"},
    {:dialyxir, "~> 1.0", only: [:dev], runtime: false}
  ]
end
```

- `mix deps.get`
- `mix dialyzer`

Distributed Elixir

What is it ?

- Connect several Elixir VMs to work together for the same goal transparently

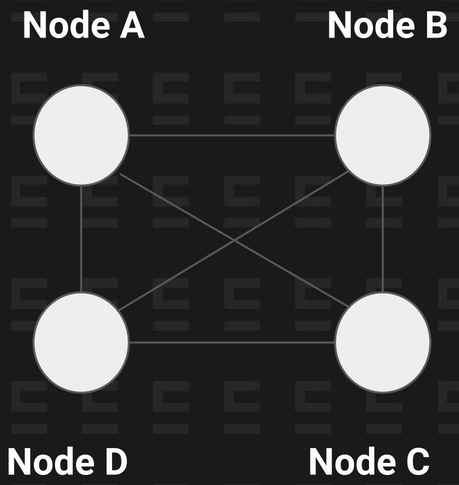
Why ?

- Fault tolerance, scaling
- When building distributed systems beware of the ‘Fallacies of distributed computing’:
    - Network is reliable
    - There is no latency
    - Bandwidth is infinite
    - Network is secure
    - Topology does not change
    - There is only one administrator
    - Transport cost is zero
    - The network is homogeneous

Erlang Solutions

Distributed Elixir

- Node is a running Elixir VM ()
- Unique name for each node - short or FQDN
- Epmd (Erlang port mapper daemon) - node name registry - listens 4369 TCP port
- Elixir provides transparency for communication (eg.: link/1 or monitor/1 works across the cluster)
- :global - global name registry
- Comm. is unencrypted, but TLS can be used
- Nodes are connected to each other with full mesh
- Nodes a monitoring each other automatically
- **`Node`** module is responsible for handling other nodes
- Inside VM **`:net_kernel`** module takes care of managing communication among nodes



Erlang Solutions

Distributed Elixir - Demo

- Create 3 nodes - (**`iex --sname node_a; iex --sname node_b; iex --sname node_c`**)
- Connect them (**`:netkernel.connect_node :“node_b@vbox”`**)
- Check the cluster (**`Node.list`**)
- Register shell pid (**`Process.register(self(), :shell)`**)
- Send message to another node **`(Process.send({:shell, :"node_b@vbox"}, {“Hello”, self()}, [:noconnect])`**
- Receive on other node (**`receive do`**
  ```
  {msg, othershell}->
  IO.puts msg
  send(othershell, “Hey there”)
  end )
  ```
- Call function with using :rpc module (**`:rpc.call(:”node_c@vbox”, Node,:list,[]))`**
- Monitor node (**`Node.monitor(:”node_c@vbox”, true)`**) -> message in mailbox when monitored node goes down

Visibility through metrics.

Find here a list of projects to help you monitor your Elixir application
https://github.com/erlef/observability-wg

A Monitoring tool made by Erlang Solutions Budapest office (Thesis projects are welcome)
https://www.erlang-solutions.com/capabilities/wombatoam/

WOMBATOAM

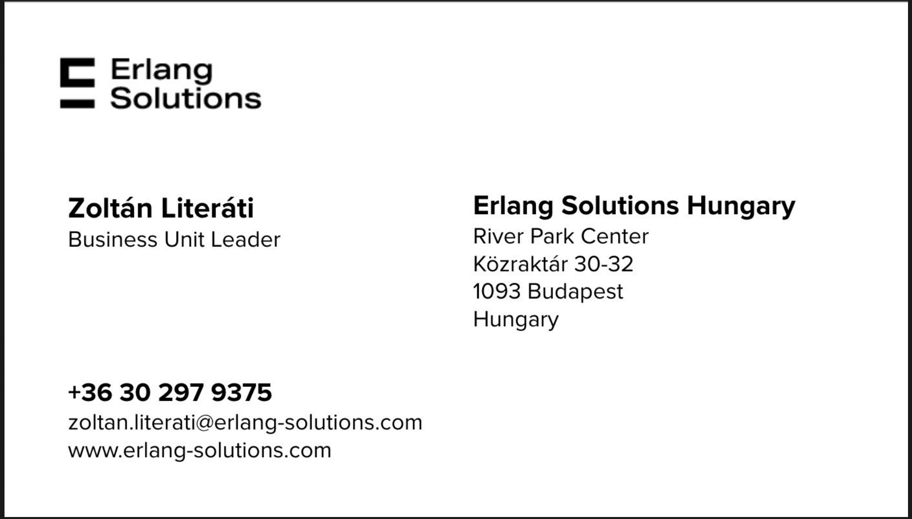
Questions |>

Erlang Solutions

Thank you!

Viktor Gergely
Ferenc Pesti
Zoltán Literáti

www.erlang-solutions.com



Erlang Solutions

Zoltán Literáti
Business Unit Leader

Erlang Solutions Hungary
River Park Center
Közraktár 30-32
1093 Budapest
Hungary

+36 30 297 9375
zoltan.literati@erlang-solutions.com
www.erlang-solutions.com