An Introduction to Elixir

April 2020 • Mike Zornek


http://mikezornek.com

Elixir is a dynamic, functional language designed for building scalable and maintainable applications. Elixir leverages the **Erlang VM**, often called the **BEAM**, which is known for running low-latency, distributed and fault-tolerant systems.

Origins


José Valim

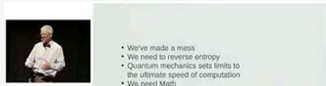

Robert Virding
Creator of Erlang
Mike Williams
Creator of Erlang
Joe Armstrong
Creator of Erlang

Erlang: The Movie

113,329 views • May 3, 2012

1.2K 12 SHARE SAVE ...

Up next AUTOPLAY



"The Mess We're In" by Joe Armstrong

Elixir is a dynamic, functional language designed for building scalable and maintainable applications. Elixir leverages the **Erlang VM**, often called the **BEAM**, which is known for running low-latency, distributed and fault-tolerant systems.

* Code Expectations

Functional Language

```
defmodule Greeter do
  def hello(name) do
    "Hello, " <> name
  end
end

iex> Greeter.hello("Sean")
"Hello, Sean"
```

```
# Other Languages
baz(new_function(other_function()))

# Elixir's Pipe Operator
other_function() |> new_function() |> baz()
```

```
iex> String.split("Elixir rocks")
["Elixir", "rocks"]


iex> "Elixir rocks" |> String.split()
["Elixir", "rocks"]
```

```
iex> String.split("bread;milk;eggs", ";")
["bread", "milk", "eggs"]


iex> "bread;milk;eggs"
|> String.upcase()
|> String.split(";")

["BREAD", "MILK", "EGGS"]
```

Dynamic

No Strict Typing

```
defmodule Greeter do
  def hello(name) do
    "Hello, " <> name
  end
end

iex(3)> Greeter.hello(6)
** (ArgumentError) argument error
    :erlang.byte_size(6)
    iex:3: Greeter.hello/1
iex(3)> Greeter.hello(true)
** (ArgumentError) argument error
    :erlang.byte_size(true)
    iex:3: Greeter.hello/1
```

```
defmodule Greeter do

  @spec hello(String.t()) :: String.t()
  def hello(name) do
    "Hello, " <> name
  end

end

# ** (CompileError) greeter.ex:8: undefined function hello/1
hello(6)
```

Table of Contents

dialyzer

Module

dialyzer

Module Summary

Dialyzer, a DIscrepancy AnaLYZer for ERlang programs.

Description

Dialyzer is a static analysis tool that identifies software discrepancies, such as definite type errors, code that has become dead or unreachable because of programming error, and unnecessary tests, in single Erlang modules or entire (sets of) applications.

Dialyzer starts its analysis from either debug-compiled BEAM bytecode or from Erlang source code. The file and line number of a discrepancy is reported along with an indication of what the discrepancy is about. Dialyzer bases its analysis on the concept of success typings, which allows for sound warnings (no false positives).

Using Dialyzer from the Command Line

Dialyzer has a command-line version for automated use. This section provides a brief description of the options. The same information can be obtained by writing the following in a shell:

```
dialyzer --help
```

Pattern Matching

```
# Algebra

10 = x * 2
```

```
iex(5)> x = 1
1
```

```
iex(5)> x = 1
1
iex(6)> 1 = x
1
```

```
iex(5)> x = 1
1
iex(6)> 1 = x
1
iex(7)> 2 = x
** (MatchError) no match of right hand side value: 1

iex(7)>
```

```
iex(7)> 3 = y
** (CompileError) iex:7: undefined function y/0
```

```
# Tuples
iex> {:ok, value} = {:ok, "Successful!"}
{:ok, "Successful!"}
iex> value
"Successful!"
iex> {:ok, value} = {:error}
** (MatchError) no match of right hand side value:
{:error}
```

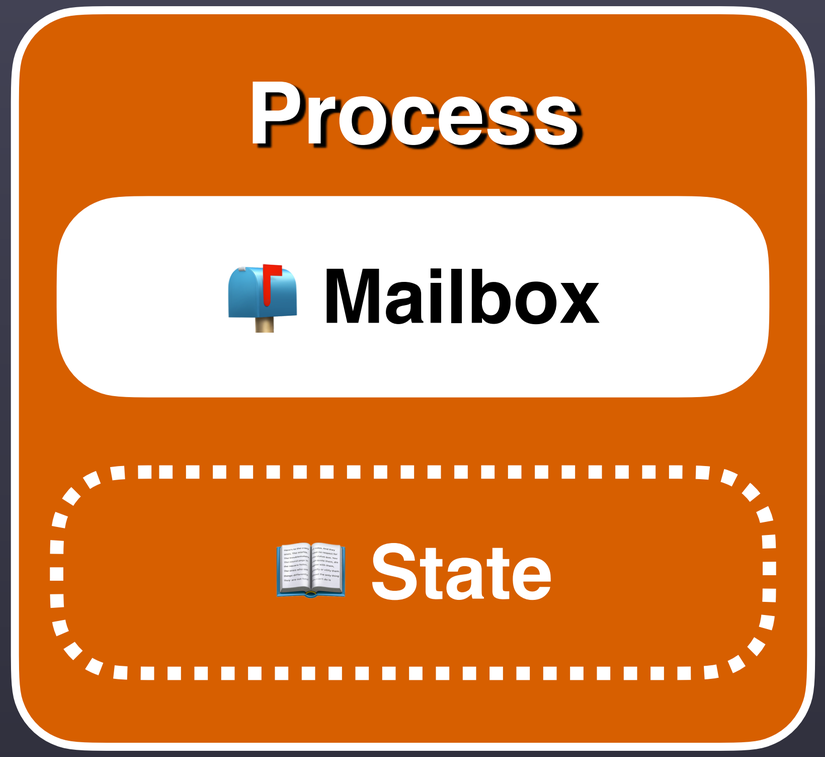
Pattern Matching

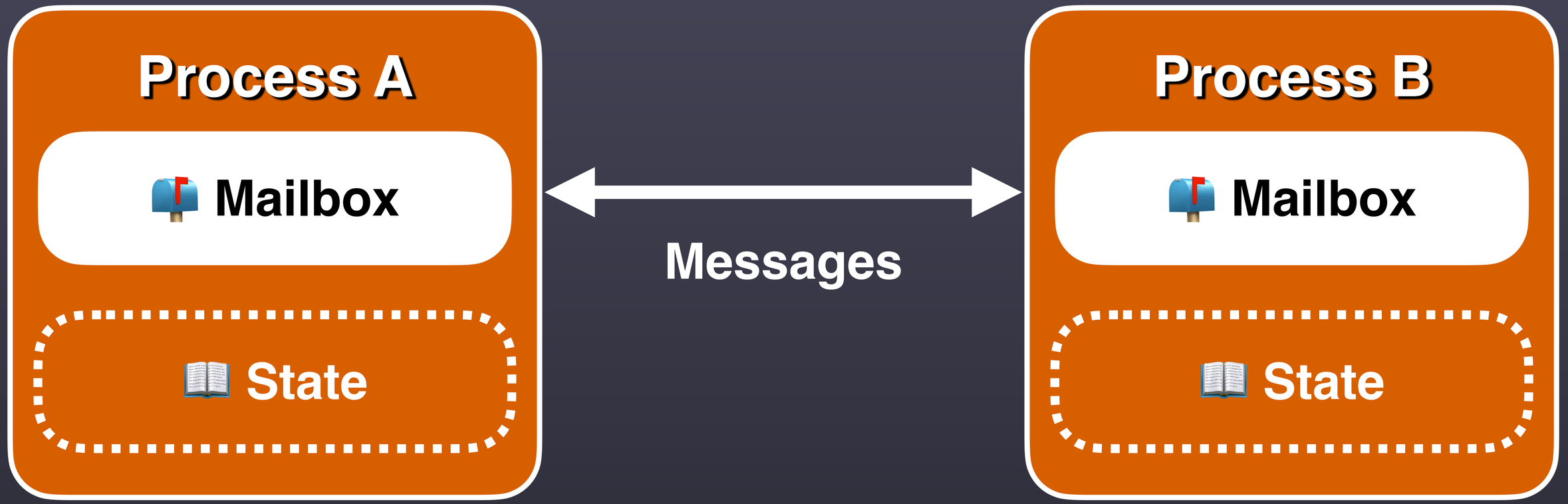
Metaprogramming

```
defmodule Friends.Person do
  use Ecto.Schema

  schema "people" do
    field :first_name, :string
    field :last_name, :string
    field :age, :integer
  end

end
```

Scalable

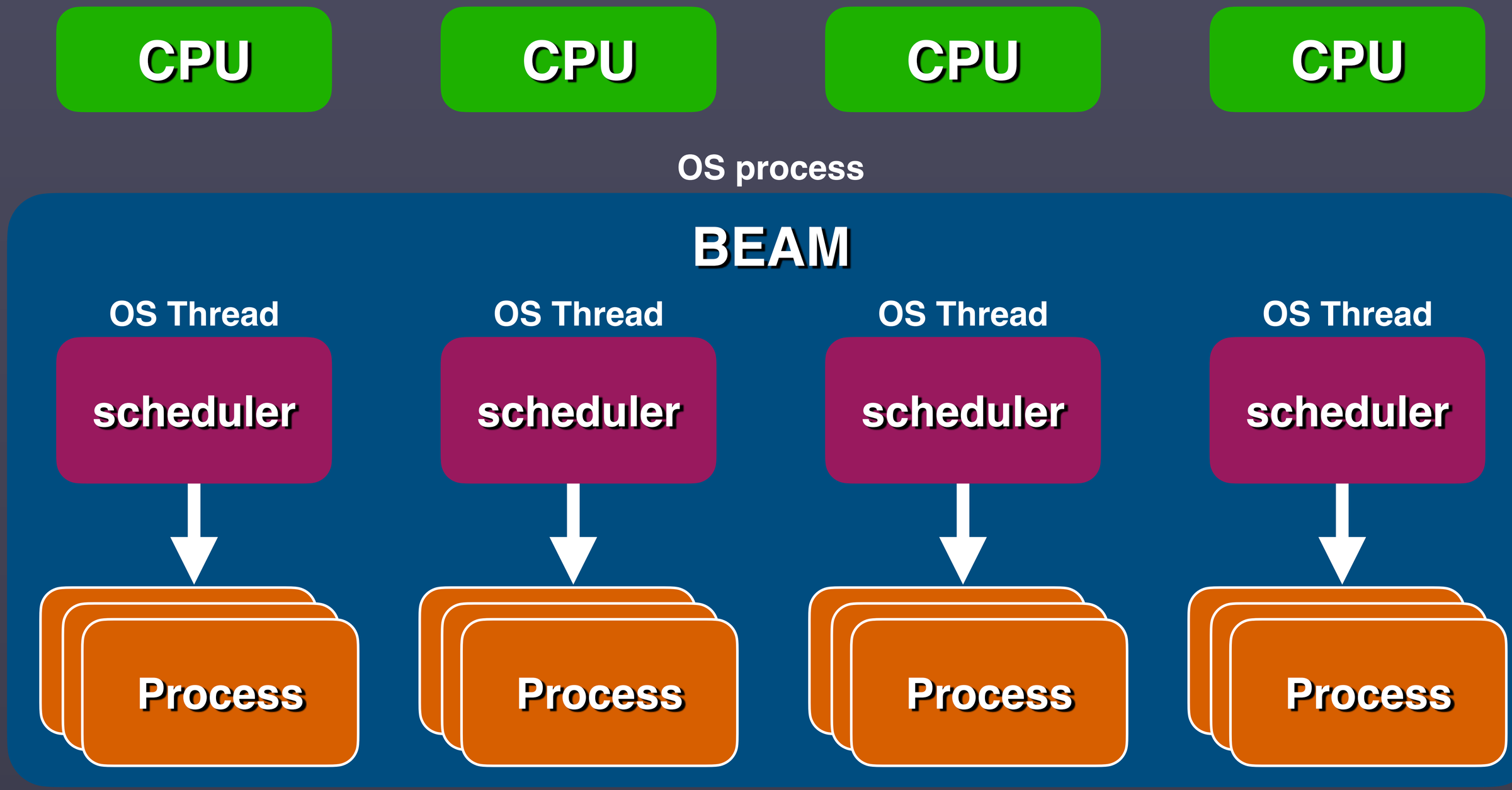

Process

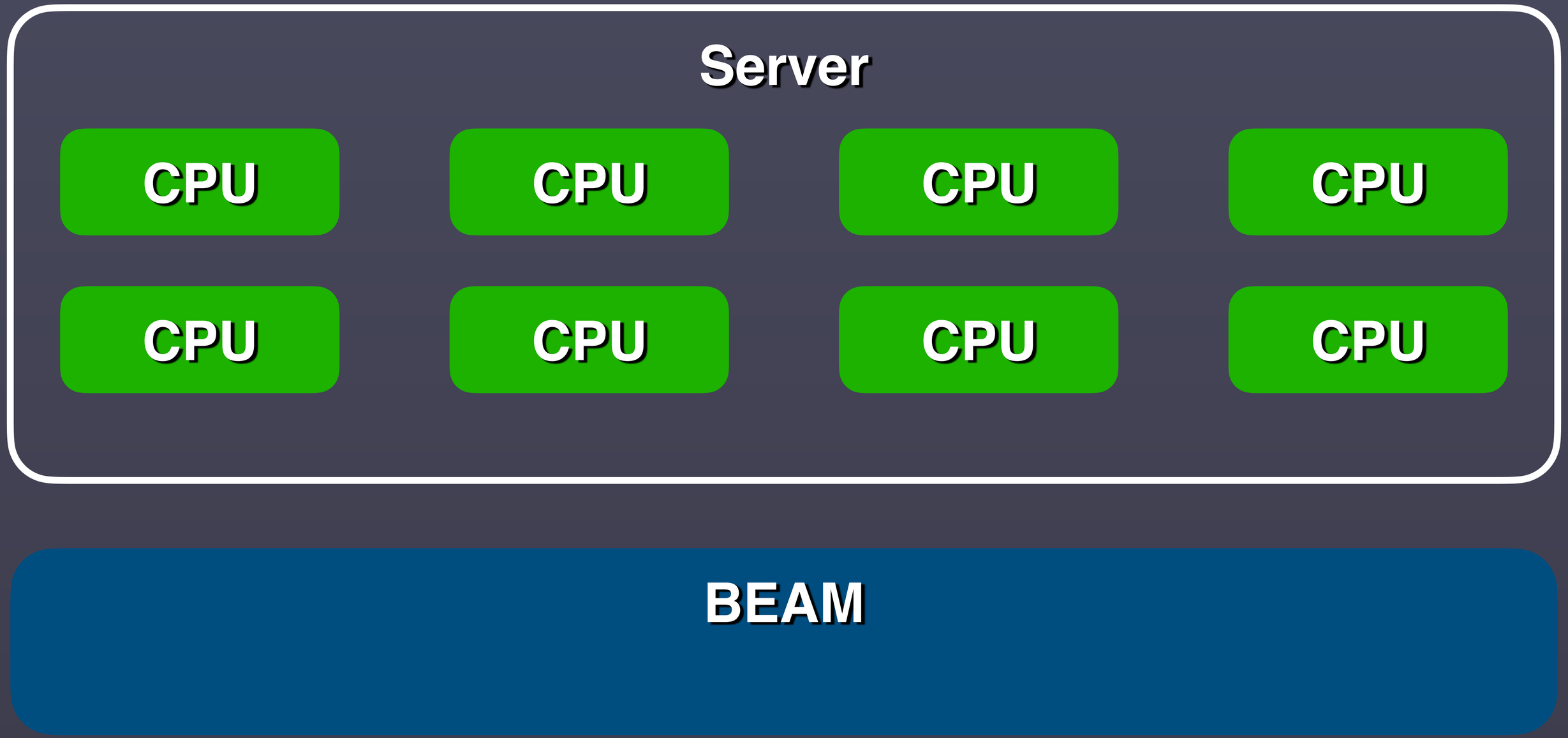

Process
📫 Mailbox
📖 State

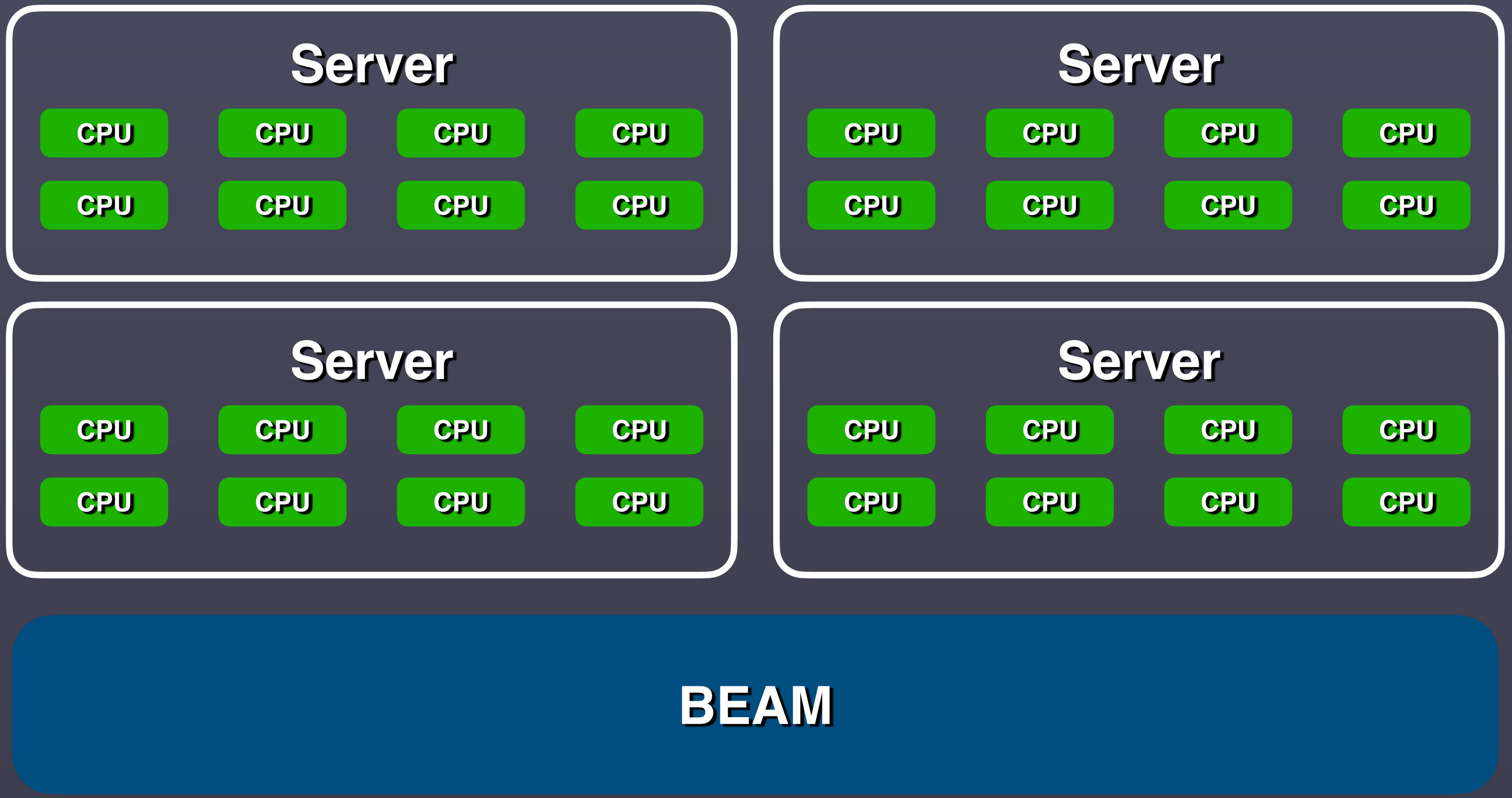

Process A
Mailbox
State
Messages
Process B
Mailbox
State

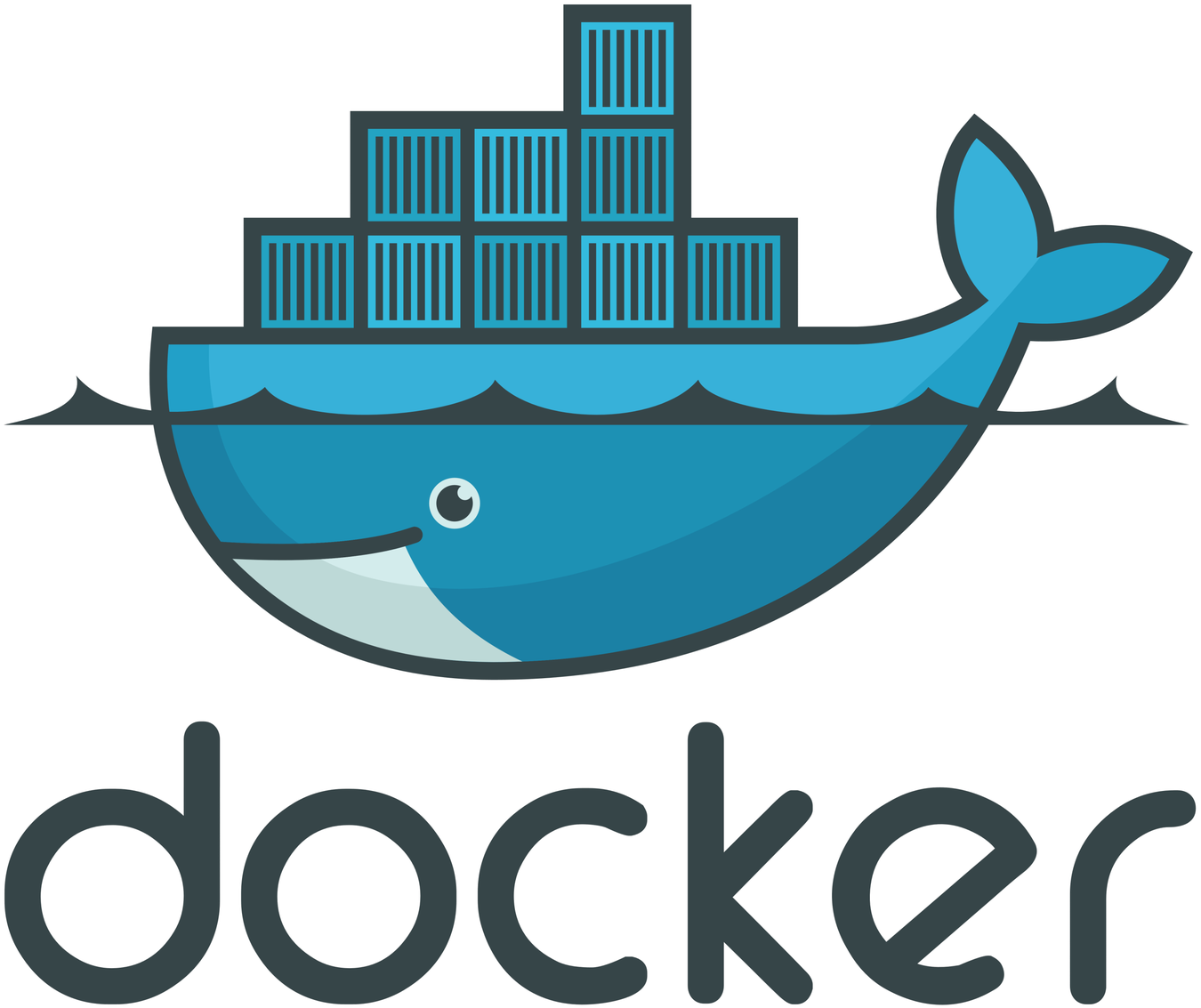
```
defmodule Example do
  def listen do
    receive do
      {:ok, "coffee"} -> IO.puts("Coffee time!")
      {:ok, "tea"} -> IO.puts("Tea, Earl Gray, Hot.")
    end

    listen()
  end
end
```

```
iex> pid = spawn(Example, :listen, [])
#PID<0.108.0>

iex> send(pid, {:ok, "coffee"})
Coffee time!
{:ok, "coffee"}

iex> send(pid, :ok)
:ok
```

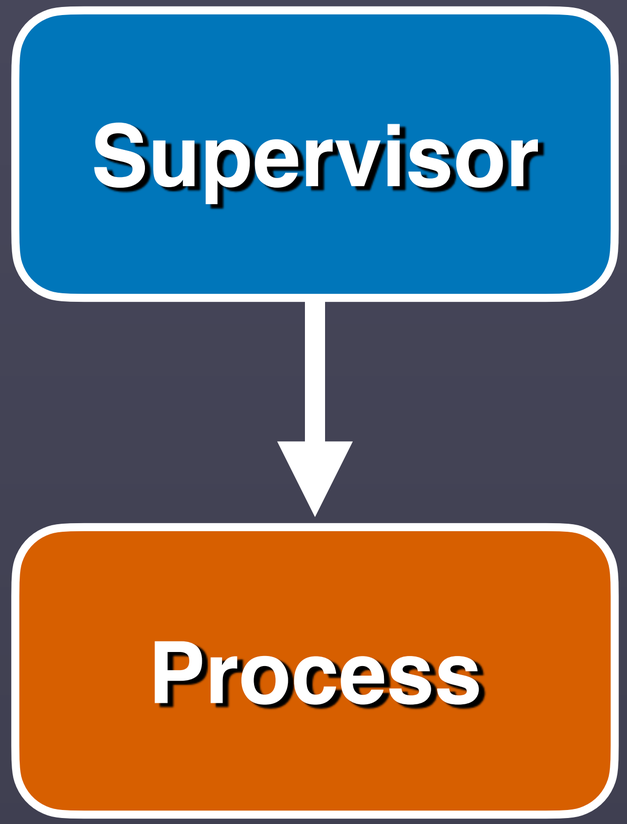

CPU
CPU
CPU
CPU
OS process
BEAM
OS Thread
scheduler
OS Thread
scheduler
OS Thread
scheduler
OS Thread
scheduler
Process
Process
Process
Process

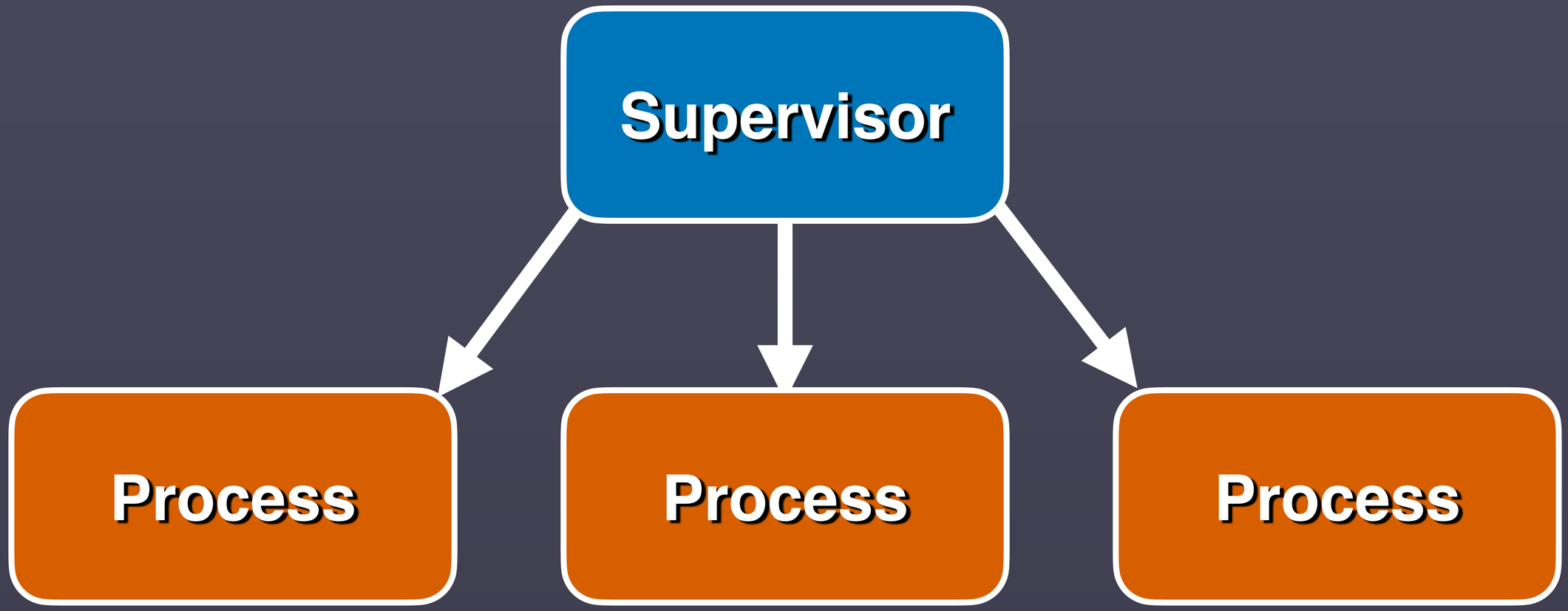

Server
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
BEAM

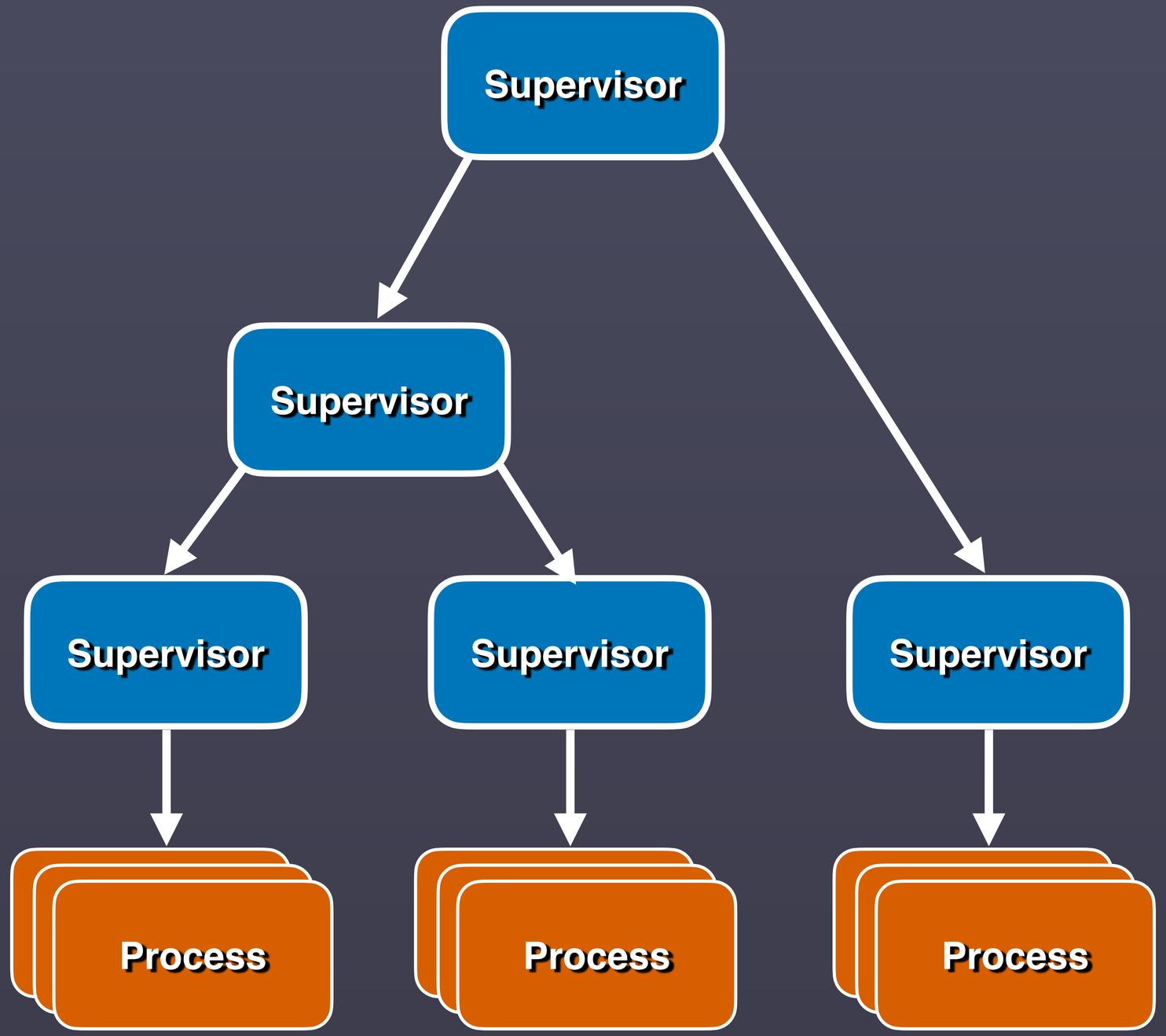

Server
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
Server
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
Server
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
Server
CPU
CPU
CPU
CPU
CPU
CPU
CPU
CPU
BEAM


docker

Fault-Tolerant


Supervisor
Process


Supervisor
Process
Process
Process

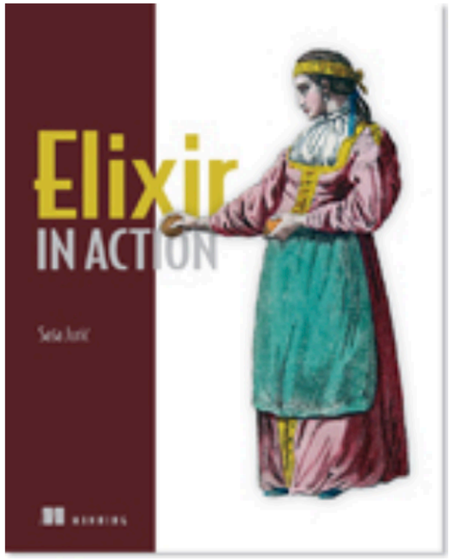

Supervisor
Supervisor
Supervisor
Supervisor
Supervisor
Process
Process
Process

Low-Latency

The Erlangelist

(not only) Erlang related musings

Hi, I'm Saša Jurić, a software developer with many years of professional experience in programming of web and desktop applications using various languages, such as Elixir, Erlang, Ruby, JavaScript, C# and C++. I'm also the author of the Elixir in Action book. In this blog you can read about Elixir, Erlang, and other programming related topics. You can subscribe to the feed, follow me on Twitter or fork me on GitHub.



Posts

Observing low latency in Phoenix with wrk

2016-06-12

Recently there were a couple of questions on Elixir Forum about observed performance of a simple Phoenix based server (see here for example). People reported some unspectacular numbers, such as a throughput of only a few thousand requests per second and a latency in the area of a few tens of milliseconds.

While such results are decent, a simple server should be able to give us better numbers. In this post I'll try to demonstrate how you can easily get some more promising results. I should immediately note that this is going to be a shallow experiment. I won't go into deeper analysis, and I won't deal with tuning of VM or OS parameters. Instead, I'll just pick a few low-hanging fruits, and rig the load test by providing the input which gives me good numbers. The point of this post is to demonstrate that it's fairly easy to get (near) sub-ms latencies with a decent throughput. Benching a more real-life like scenario is more useful, but also requires

slogsdon / language-web-shootout

Watch 1 | Star 1 | Fork 0

Code | Issues 0 | Pull requests 0 | Actions | Projects 0 | Security | Insights

Web shootout between some recently used languages

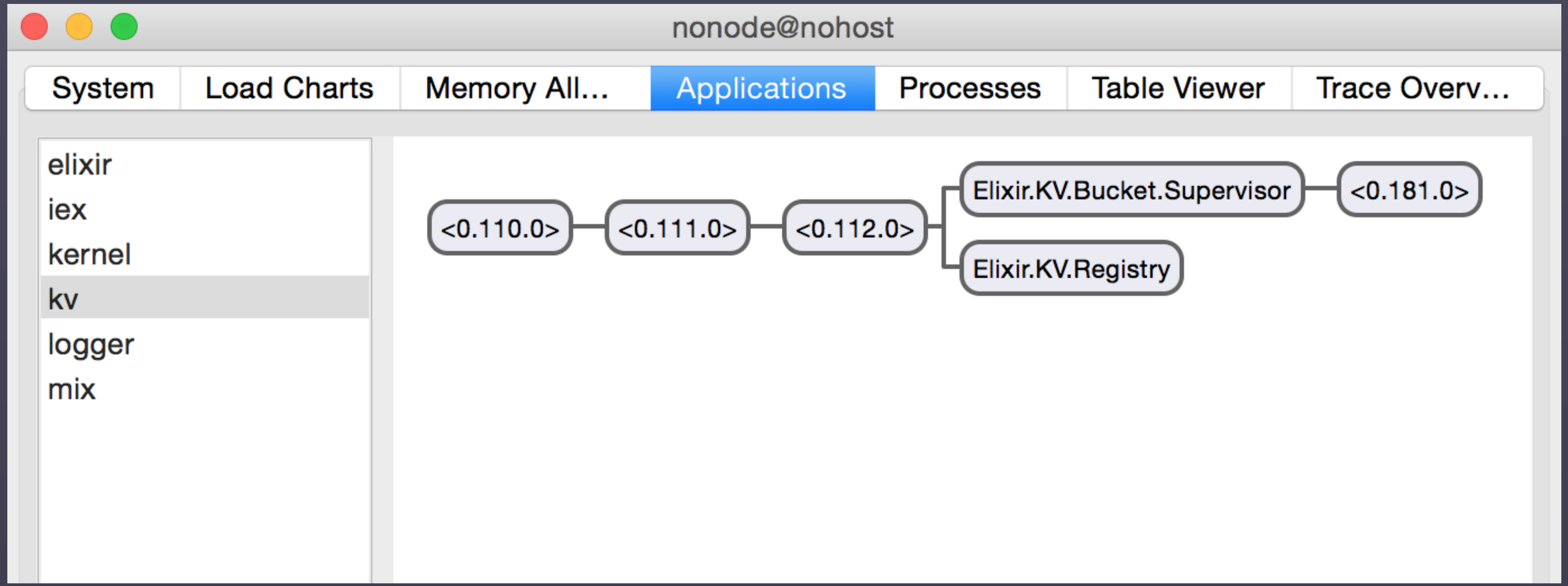
25 commits | 1 branch | 0 packages | 0 releases | 1 contributor | MIT

Branch: master | New pull request | Find file | Clone or download

slogsdon adding lua — Latest commit acb9618 on Oct 12, 2014

clojure	clojure: adding http-kit option	6 years ago
common-lisp	adding common lisp	6 years ago
d	adding d	6 years ago
elixir	removing project readmes	6 years ago
erlang	formatting. first set of results. readme work	6 years ago
go	formatting. first set of results. readme work	6 years ago
haskell	haskell: update to 7.8.3	6 years ago
lua	adding lua	6 years ago

Maintainable


nonode@nohost
System
Load Charts
Memory All...
Applications
Processes
Table Viewer
Trace Overv...
elixir
iex
kernel
kv
logger
mix
<0.110.0>
<0.111.0>
<0.112.0>
Elixir.KV.Bucket.Supervisor
<0.181.0>
Elixir.KV.Registry

```
$ iex -S mix
iex(1)> :observer.start()
```

elixir-lang / elixir

Watch 739 | Star 16.7k | Fork 2.4k

Code | Issues 16 | Pull requests 1 | Actions | Wiki | Security | Insights

Releases | Tags

Tags

v1.10.2 ...

on Feb 26 0745083 zip tar.gz Notes Downloads

v1.10.1 ...

on Feb 10 51c95b6 zip tar.gz Notes Downloads

v1.10.0 ...

on Jan 27 5bd7a90 zip tar.gz Notes Downloads

v1.10.0-rc.0 ...

on Jan 7 105770c zip tar.gz Notes Downloads

v1.9.4 ...

The package manager for the Erlang ecosystem

Find packages

Using with Elixir

Specify your Mix dependencies as two-item tuples like `{:plug, "~> 1.1.0"}` in your dependency list, Elixir will ask if you want to install Hex if you haven't already.

Using with Erlang

Download rebar3, put it in your `PATH` and give it executable permissions. Now you can specify Hex dependencies in your rebar.config like `{deps, [hackney]}`.

GETTING STARTED

Fetch dependencies from Hex without creating an account. Hex is usable out of the box in Elixir with Mix and in Erlang with

PUBLISH PACKAGES

Create an account and follow the publishing guide. Your package will be immediately available to all Elixir and Erlang users and the

PRIVATE PACKAGES

Publish private packages by creating an organization. Your private packages will get the same features as public packages such as

Naming Conventions

This document covers some naming conventions in Elixir code, from casing to punctuation characters.

Casing

Elixir developers must use `snake_case` when defining variables, function names, module attributes, and the like:

```
some_map = %{this_is_a_key: "and a value"}
is_map(some_map)
```

Aliases, commonly used as module names, are an exception as they must be capitalized and written in `CamelCase`, like `OptionParser`. For aliases, capital letters are kept in acronyms, like `ExUnit.CaptureIO` or `Mix.SCM`.

Atoms can be written either in `:snake_case` or `:CamelCase`, although the convention is to use the snake case version throughout Elixir.

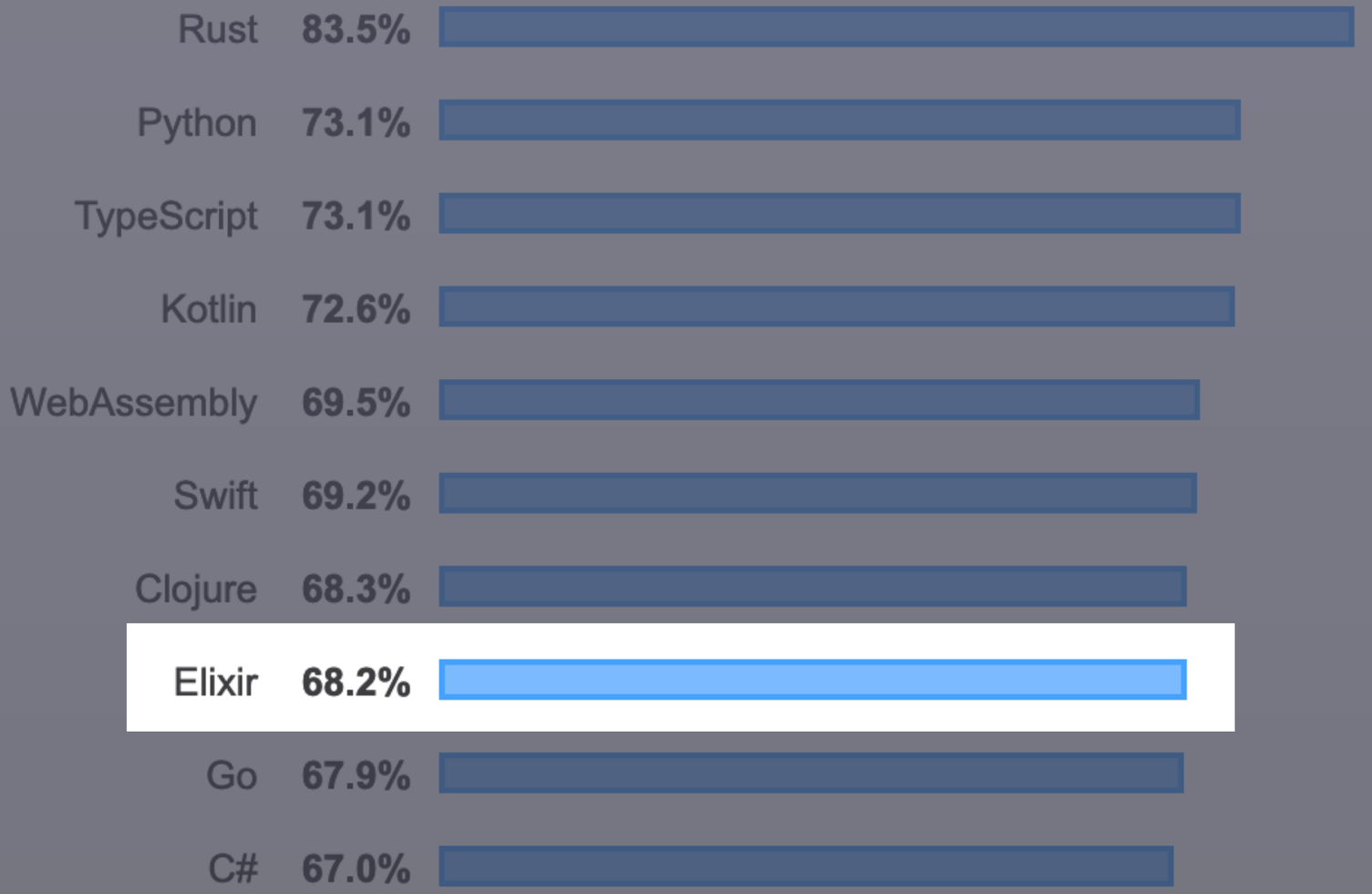
Generally speaking, filenames follow the `snake_case` convention of the module they define. For example, `MyApp` should be defined inside the `my_app.ex` file. However, this is only a convention. At the end of the day, any filename can be used as they do not affect the compiled code in any way.

Underscore (`_foo`)

Elixir relies on underscores in different situations.

Most Loved, Dreaded, and Wanted Languages

Loved | Dreaded | Wanted



Notable Projects

- Building web applications using **Phoenix**.
- Working with databases using **Ecto**.
- Assemble data processing pipelines with **Broadway**.
- Crafting GraphQL APIs using **Absinthe**.
- Deploying embedded software using **Nerves**.

- **Elixir Language Website Guide**
  https://elixir-lang.org/getting-started/introduction.html

- **Elixir School**
  https://elixirschool.com/en/

- **Elixir in Action (Book)**
  https://www.manning.com/books/elixir-in-action

- **The Pragmatic Studio (Videos)**
  https://pragmaticstudio.com/elixir

- **ElixirConf**
  https://www.youtube.com/channel/UC0l2QTnO1P2iph-86HHilMQ/videos

Thanks!

Available For Hire:

http://mikezornek.com/for-hire/

Contact:

@zorn on Micro.Blog + Twitter