Welcome to Elixir

"This is good shit"
— Joe Armstrong

June 20, 2016

Table of Contents

Motivation

- Power of Erlang
- Elixir basics
- Macros
- Tooling and abstractions

Table of Contents

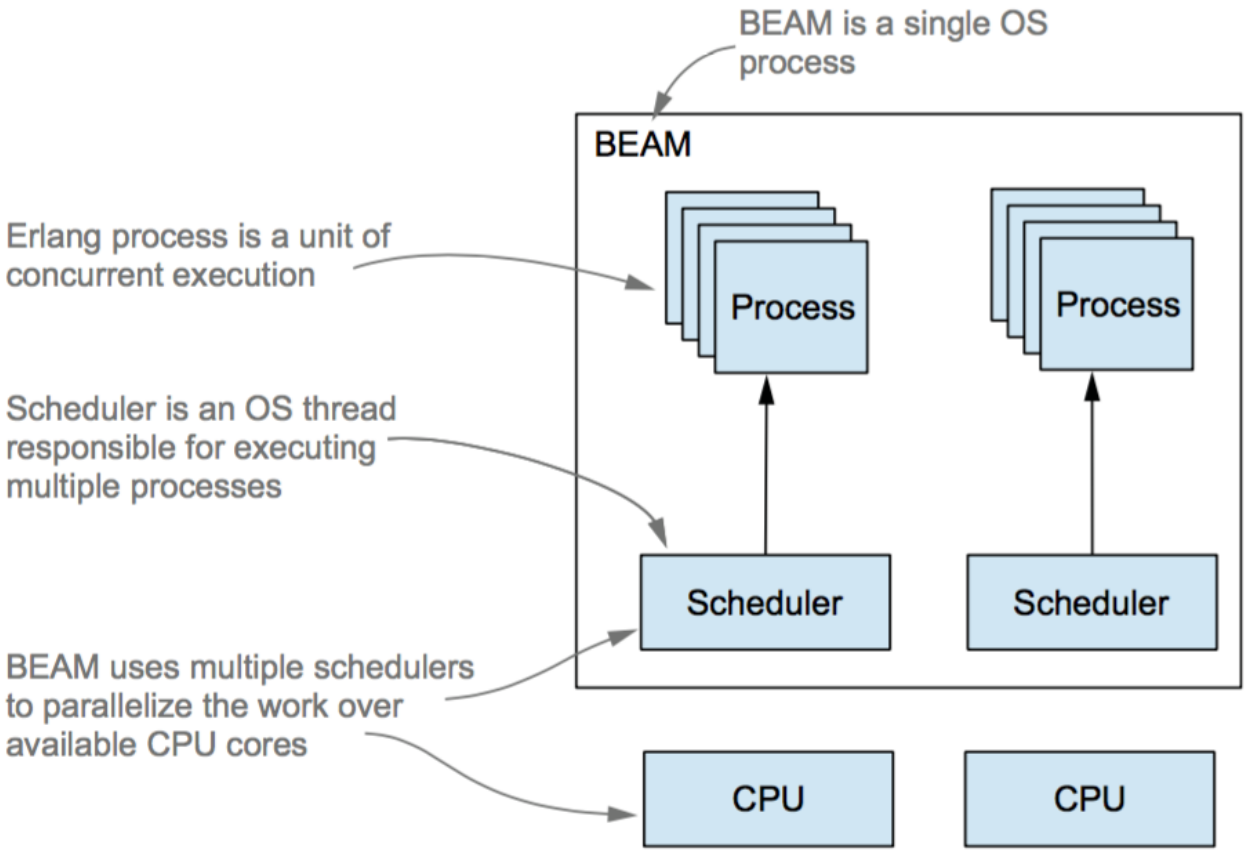
Erlang

- created in mid-1980s
- designed for telecom
- connect multiple systems
- minimal impact of errors
- entire system should never go down

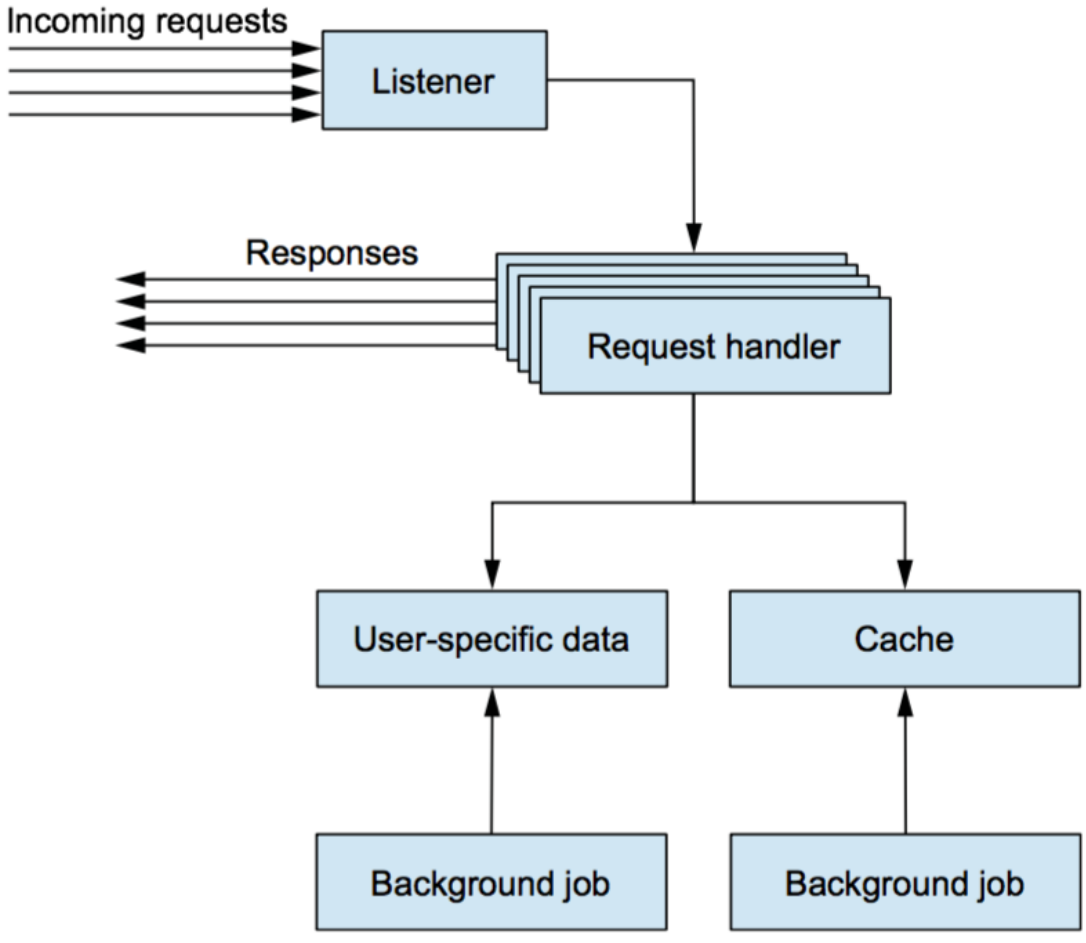
High availability

- fault tolerance
- scalability
- distribution
- responsiveness
- live update

How do they do it?



OTP


Incoming requests
Listener
Responses
Request handler
User-specific data
Cache
Background job
Background job

A modern system

Technical requirements	Server
HTTP server	Nginx and Phusion Passenger
Request processing	Ruby on Rails
Long-running requests	Go
Server-wide state	Redis
Persistable data	Redis and MongoDB
Background jobs	Cron, Bash scripts, and Ruby
Service crash recovery	Upstart

OTP capabilities

Technical requirements	Server
HTTP server	Erlang
Request processing	Erlang
Long-running requests	Erlang
Server-wide state	Erlang
Persistable data	Erlang
Background jobs	Erlang
Service crash recovery	Erlang

Table of Contents

Syntax

"Elixir syntax is like a marriage of DSL friendly Ruby and the powerful hygenic macros of Clojure."
– Devin Torres

The basics you know

```
1 + 1                  # => 2
2 * (3 + 1) / 4        # => 2.0
1 + 2; 1 + 3           # => 4

greeting = "Hello World!"
IO.puts(greeting)
# => Hello World!
# => :ok
```

Modules and function, oh my!

```
defmodule Geometry do
  def rectangle_area(a, b) do
    a * b
  end
end
```

Composing functions

```
def process_xml(model, xml) do
  model
  |> update(xml)
  |> process_changes
  |> persist
end
```

Function arity

```
defmodule Rectangle do
  def area(a), do: area(a, a)
  def area(a, b), do: a * b
end
```

Destructuring

```
def do_something({:ok, value}) do
  # use the value here
end

def do_something({:warning, value}) do
  # warn user before proceeding
end

def do_something({:error, message}) do
  # produce a nice error message to the user
end
```

Typespec

```
% read up on dializer and custom types
defmodule Circle do
  @pi 3.14159

  @spec area(number) :: number
  def area(r), do: r * r * @pi

  @spec circumference(number) :: number
  def circumference(r), do: 2 * r * @pi
end
```

Table of Contents

Macros

- code transformation at compile time
- code that can change semantics of the input
- a lot of elixir functionality is implemented in macros *e.g. def*, *unless*
- should be used sparingly

Table of Contents