A SIP OF ELIXIR

Artem Chernyak

June 26, 2016

TABLE OF CONTENTS

MOTIVATION

- Power of Erlang
- Elixir basics
- Macros
- Tooling

TABLE OF CONTENTS

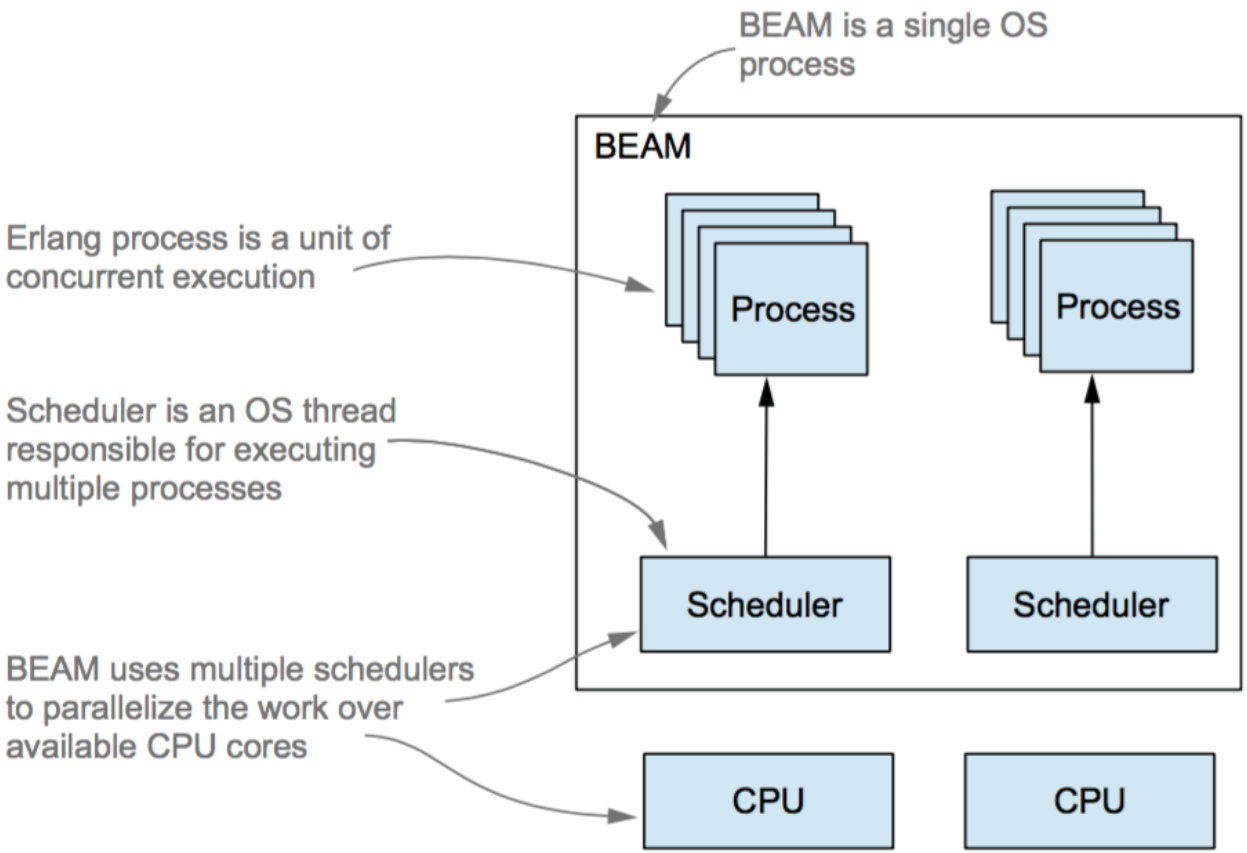
ERLANG

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

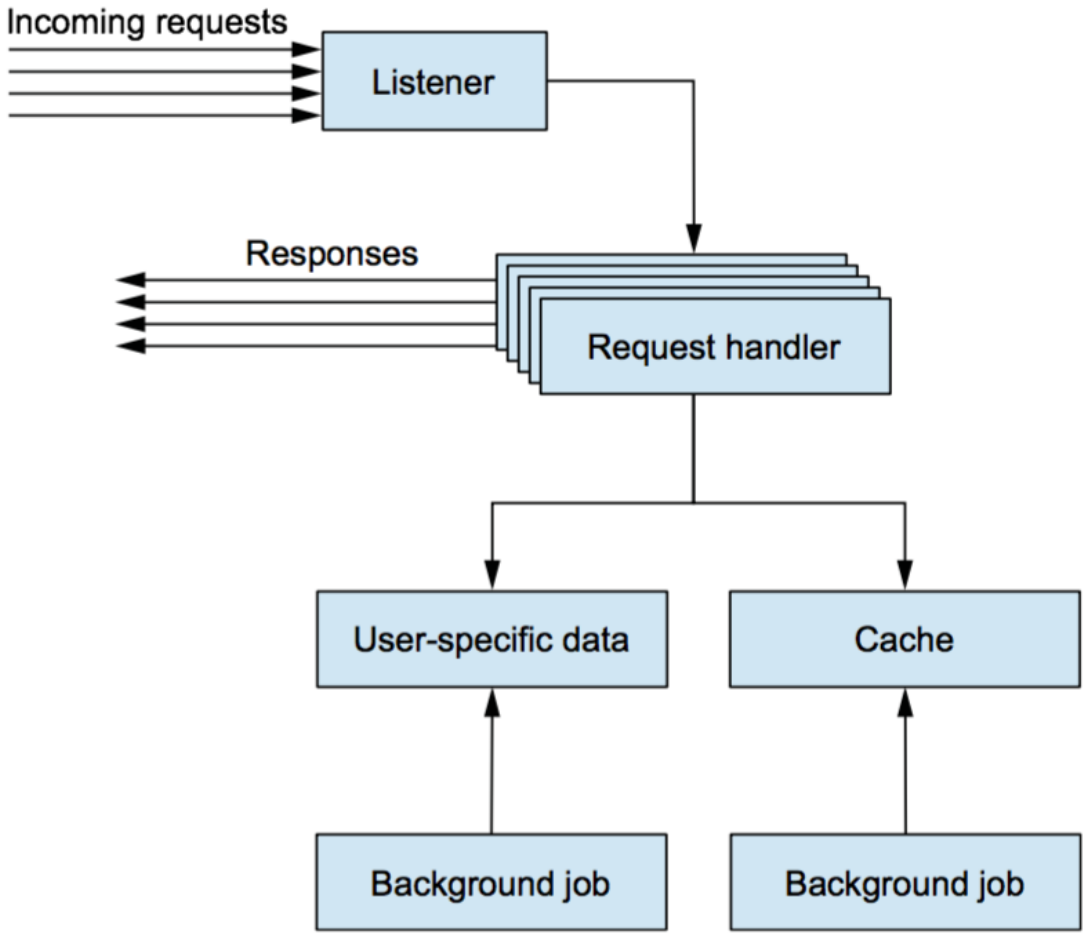
HIGH AVAILABILITY

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

HOW DO THEY DO IT?



SERVER SIDE SYSTEMS



A MODERN SYSTEM

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

OTP

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

```
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

> *Lisp traditionally empowered developers because you can eliminate anything that's tedious through macros, and that power is really what people keep going back for.*

– *Rich Hickey*

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

MACRO EXPENSION

```
unless some_exp do
  block_1
else
  block_2
end
```

```
if some_exp do
  block_2
else
  block_1
end
```

QUOTING

```
quote do: sum [1, 2, 3]
# => { :sum, [], [1, 2, 3] }

quote do: 1 + 1
# => { :+,
#      [context: Elixir, import: Kernal],
#      [1, 1] }
```

SIMPLE MACRO

```
defmacro match?(left, right) do
  quote do
    case unquote(right) do
      unquote(left) ->
        true
      _ ->
        false
    end
  end
end
```

USINE OUR NEW MACRO!

```
list = [{:a,1},{:b,2},{:a,3}]
# => [a: 1, b: 2, a: 3]

Enum.filter list, fn (thing) do
  match?({:a, _}, thing)
end
# => [a: 1, a: 3]

Enum.filter list, match?({:a, _}, &1)
# => [a: 1, a: 3]
```

TABLE OF CONTENTS

IEX

- full REPL
- easy to load and interact
- full coloring support
- easy extensible with user defined helpers

DOCTEST

```
@doc """
Check if the given argument is nil or not.
Allowed in guard clauses.

## Examples

  iex> nil?(1)
  false
  iex> nil?(nil)
  true
"""
```

MIX

- Great build and deploy tool
- Based around Mix.Tasks
  - Simply define run/1
  - Easy documentation
  - Access to 15 helper methods
- Great documentation

MIX TASKS

```
mix clean              # Clean generated application files
mix compile            # Compile source files
mix deps               # List dependencies and their status
mix deps.clean         # Remove dependencies files
mix deps.compile       # Compile dependencies
mix deps.get           # Get all out of date dependencies
mix deps.unlock        # Unlock the given dependencies
mix deps.update        # Update dependencies
mix do                 # Executes the commands separated by comma
mix escriptize         # Generates an escript for the project
mix help               # Print help information for tasks
mix local              # List local tasks
mix local.install      # Install a task locally
mix local.rebar        # Install rebar locally
mix local.uninstall    # Uninstall local tasks
mix new                # Creates a new Elixir project
mix run                # Run the given expression
mix test               # Run a project's tests
```

HEX

- Easy to use package manager
- Searchable
- Manages dependencies
- Almost 2300 packages available

ITEROPERABILITY WITH ERLANG

```
# Elixir map function
Enum.map

# Erlang map function
:list.map
```