FUNCTIONAL PROGRAMMING IN THE WEB WITH ELIXIR AND ELM

RODRIGO NONOSE

SÃO PAULO, 18. NOVEMBER. 2017.

rhnonose

RodrigoNonose


DIGITAL
NATIVES


Staff

DIGITAL NATIVES

Summary

- OO + Its problems
- Introduction to Elm and Elixir
- FP concepts + code examples
- Runtime

DIGITAL NATIVES

Object Orientation

"Original" definition

- Objects contain state
- An object can send messages
- An object can receive messages

DIGITAL NATIVES

Object Orientation Problems

How it was implemented in mainstream languages

- State and Logic in the same place
- State mutability anywhere in the method body
- State mutability outside of the module
- Temporal coupling
- Inheritance doesn't offer what's promised
- Passing objects by reference breaks encapsulation

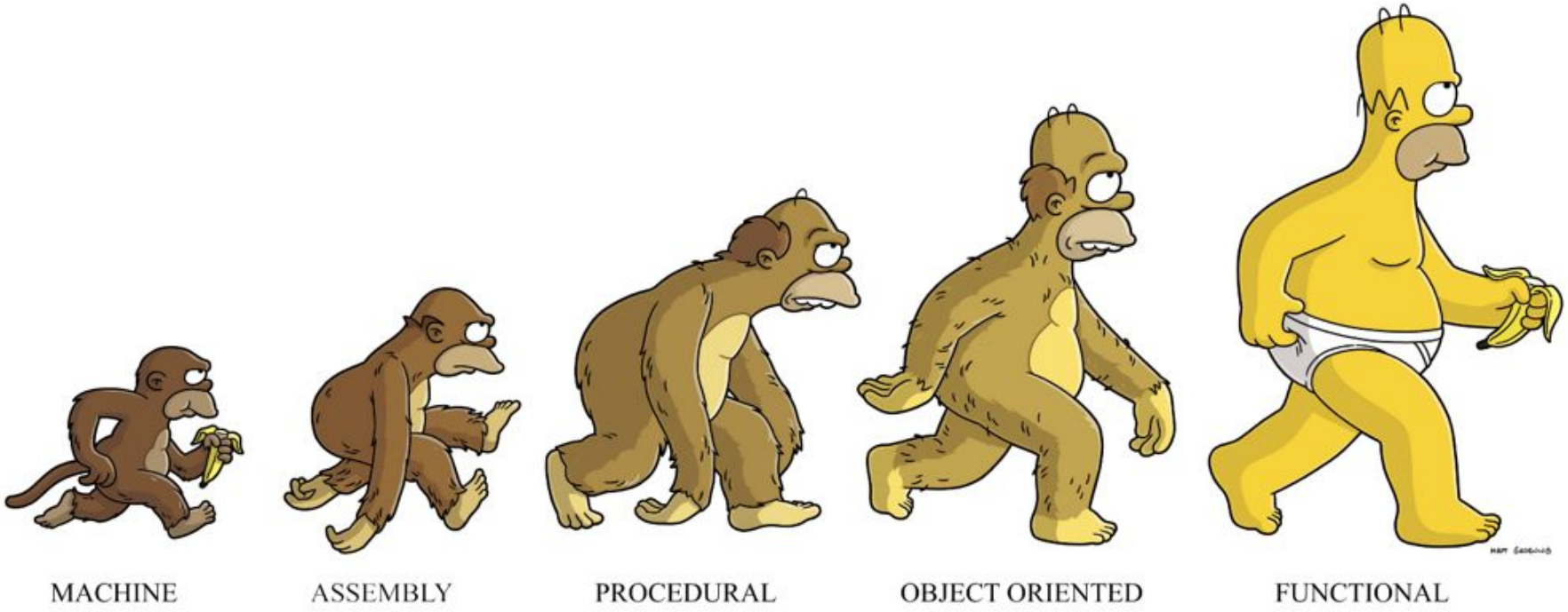
DIGITAL NATIVES

Design Patterns

- Singleton
- Factory
- Factory Method
- Abstract Factory
- Builder
- Object Pool
- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Prototype
- Adapter
- Decorator
- Bridge
- Proxy
- Composite
- Flyweight
- Memento
- Strategy
- Template Method
- Visitor
- ...

DIGITAL NATIVES

Functional Programming



DIGITAL NATIVES

Elm

- Purely functional and statically typed
- Expressive, concise and self-documented
- Immutability and Referential Transparency
- There's no null
- There's no undefined function
- Interoperable with Javascript
- Compiled to Javascript
- Forced semantic versioning

DIGITAL NATIVES

Elixir

- Functional, dinamically typed
- Focus on scalability and maintenability
- Fault-tolerant
- Extensible
- Compiled to Erlang Virtual Machine (BEAM)

DIGITAL NATIVES

Functional Programming

Pure functions

Recursion

First-class functions

High-order functions

DIGITAL NATIVES

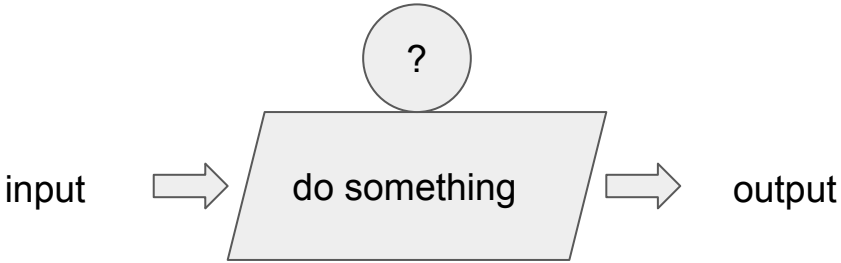
Pure Functions

DIGITAL NATIVES

Pure Functions

DIGITAL NATIVES

Pure Functions



DIGITAL NATIVES

Pure Functions

Javascript

```
1  // with side-effect
2  counter = 0
3  function increment(){
4    counter++;
5  }
6  increment();
7  console.log(counter); // 1
8
9  // without side-effect
10 function pure_increment(counter){
11   return counter + 1;
12 }
13 new_counter = pure_increment(counter);
14 console.log(new_counter) // 2
15 console.log(counter) // 1
```

DIGITAL NATIVES

Pure Functions

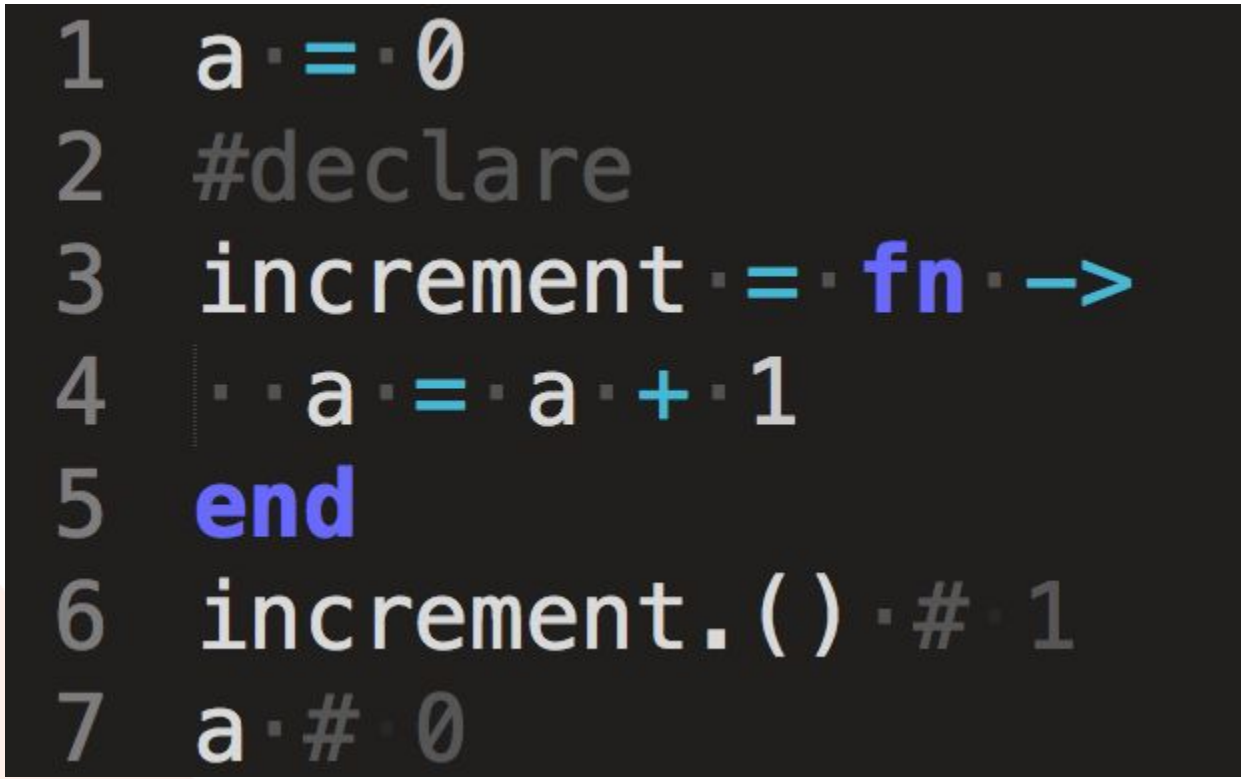
Elm

```
> a = 0
0 : number
> a = a + 1
-- BAD RECURSION ------------------------------------------ repl-temp-000.elm

`a` is defined directly in terms of itself, causing an infinite loop.

2| a = a + 1
   ^
Maybe you are trying to mutate a variable? Elm does not have mutation, so when I
see `a` defined in terms of `a`, I treat it as a recursive definition. Try
giving the new value a new name!

Maybe you DO want a recursive value? To define `a` we need to know what `a` is,
so let's expand it. Wait, but now we need to know what `a` is, so let's expand
it... This will keep going infinitely!

To really learn what is going on and how to fix it, check out:
<https://github.com/elm-lang/elm-compiler/blob/0.18.0/hints/bad-recursion.md>
```

DIGITAL NATIVES

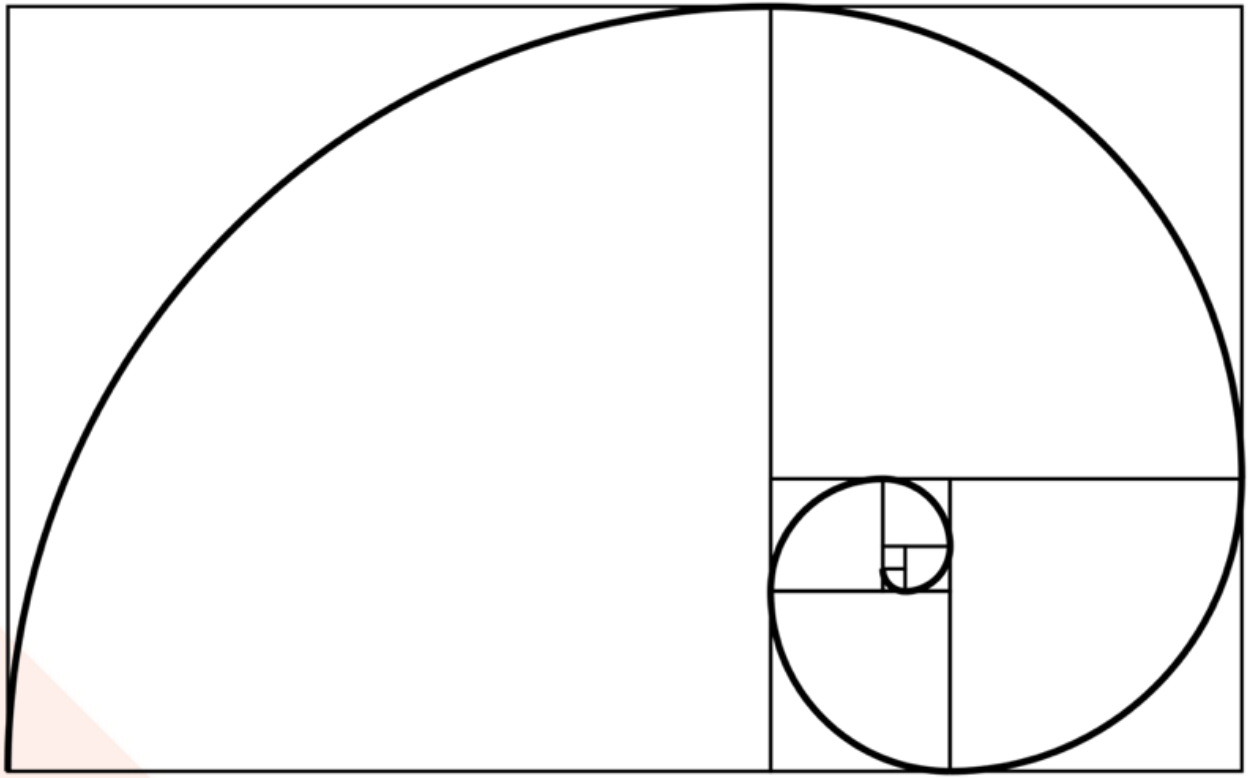
Pure Functions

Elixir



DIGITAL NATIVES

Recursion



DIGITAL NATIVES

Recursion

Java

```java
//iteractive fibonacci
static int fibo_iter(int n) {
    int x = 0, y = 1, z = 1;
    for (int i = 0; i < n; i++) {
        x = y;
        y = z;
        z = x + y;
    }
    return x;
}

//recursive fibonacci
static int fibo_recur(int  n) {
    if ((n == 1) || (n == 0)) {
        return n;
    }
    return fibo_recur(n - 1) + fibo_recur(n - 2);
}
```

DIGITAL NATIVES

Recursion

Elm

```
1  fibo n =
2      case n of
3          0 -> 0
4          1 -> 1
5          n -> (fibo (n - 1)) + (fibo (n - 2))
```

DIGITAL NATIVES

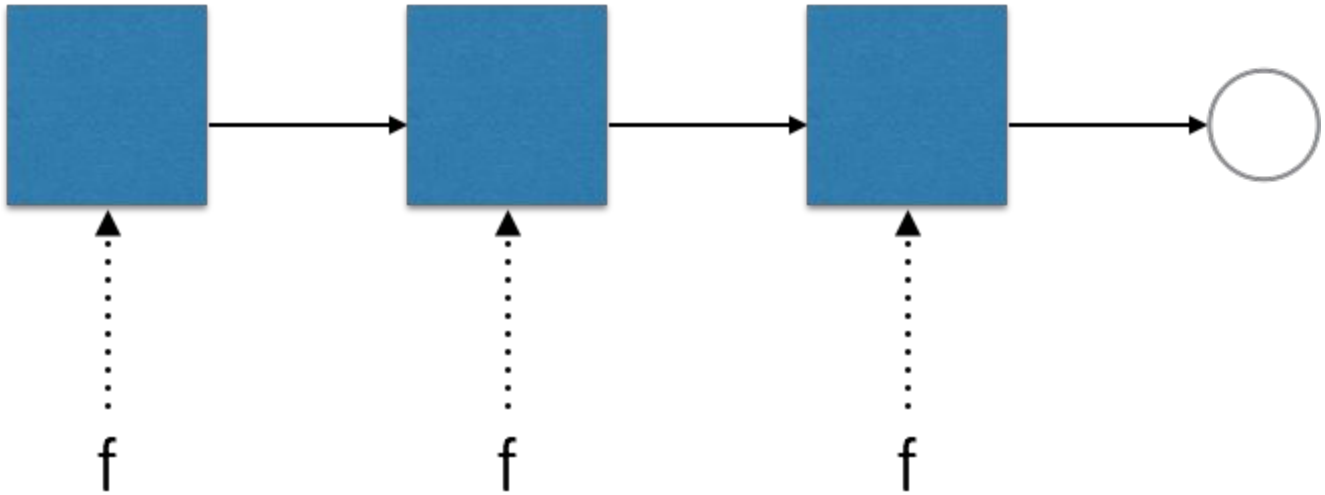
Recursion

Elixir

```elixir
def fibo(0), do: 0
def fibo(1), do: 1
def fibo(n) do
  fibo(n - 1) + fibo(n - 2)
end
```

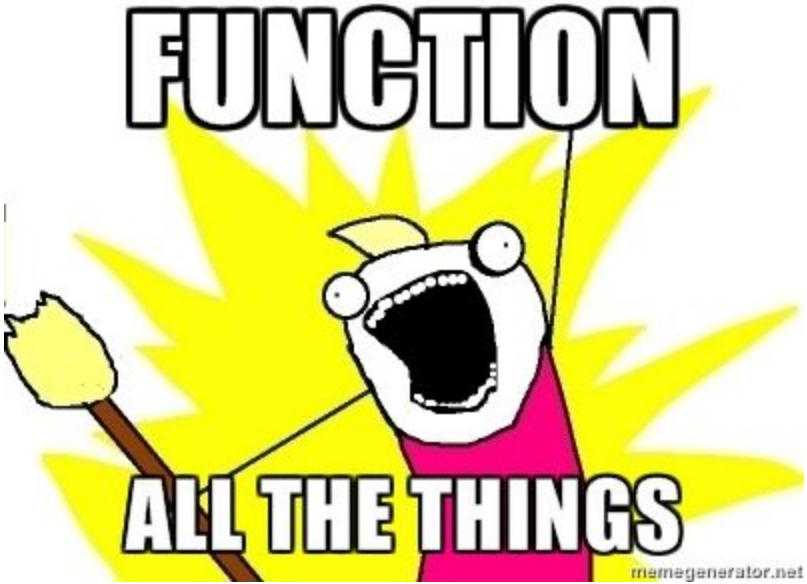
DIGITAL NATIVES

Recursion



DIGITAL NATIVES

First-class Functions



DIGITAL NATIVES

First-class Functions

Elm

```elm
-- declare outer function
apply a b fun =
  fun a b

-- declare function to be passed as parameter
add a b = a + b

-- calls the outer function
apply 1 2 (add) -- 3

-- call function with other parameters
apply "first_name" "last_name" (++) -- "first_namelast_name"
```

DIGITAL NATIVES

First-class Functions

Elixir

```elixir
# declare outer function
apply = fn a, b, fun -> fun.(a, b) end

# declare function to be passed as parameter
add = fn a, b -> a + b end

# calls the outer function
apply.(1, 2, add) # 3

# call function with other parameters
apply.("first_name", "last_name", &<>/2) # "first_namelast_name"
```

DIGITAL NATIVES

High-order Functions

DIGITAL NATIVES

High-order Functions

Java

```java
//"Function" to filter odd numbers
public List<Integer> filterOdds(List<Integer> list) {
  List<Integer> filteredList = new ArrayList<Integer>();

  for(Integer element : list) {
    if(1 == element % 2) {
      filteredList.add(element);
    }
  }
  return filteredList;
}
```

DIGITAL NATIVES

```
public List<Integer> filter(List<Integer> list, Predicate predicate) {
  List<Integer> filteredList = new ArrayList<Integer>();

    for (Integer element : list) {
      if (predicate.evaluate(element)) {
        filteredList.add(element);
      }
    }
    return filteredList;
}

public interface Predicate {
    public boolean evaluate(Integer arg);
}

class IsEven implements Predicate {

    public boolean evaluate(Integer arg) {
      return 0 == arg % 2;
    }
}

class IsOdd implements Predicate {

    public boolean evaluate(Integer arg) {
      return 1 == arg % 2;
    }
}

filter(Arrays.asList(1,2,3,4), new IsEven());
filter(Arrays.asList(1,2,3,4), new IsOdd());
```

DIGITAL NATIVES

High-order Functions

Elixir

```elixir
# anonymous function 1
Enum.filter([1,2,3,4], fn n -> rem(n, 2) == 0 end)
Enum.filter([1,2,3,4], fn n -> rem(n, 2) == 1 end)

# anonymous function 2
Enum.filter([1,2,3,4], &(rem(&1, 2) == 0))
Enum.filter([1,2,3,4], &(rem(&1, 2) == 1))

# existing functions from Integer module
Enum.filter([1,2,3,4], &Integer.is_even/1)
Enum.filter([1,2,3,4], &Integer.is_odd/1)
```

DIGITAL NATIVES

High-order Functions

- Map
- Reduce
- Filter
- Find
- Split

- Count
- Sum
- Reject
- Min
- Max

DIGITAL NATIVES

High-order Functions

Java 8

```
1  //Java 8
2  Arrays
3      .asList("elm", "elixir", "scala", "clojure", "haskell")
4      .stream()
5      .filter(str -> !"elixir".equals(str))
6      .collect(Collectors.toList());
```

DIGITAL NATIVES

Functional Programming (bonus)

Partial application

Piping

Pattern matching

Immutability

DIGITAL NATIVES

Partial Application

Elm

```elm
-- function with 3 parameters
add a b c = a + b + c

-- call with only one parameter
add1 = add 1 -- <function> : number -> number -> number

-- call the returned function
add2 = add1 2 -- <function> : number -> number

-- call the returned function
add2 3 -- 6
```

DIGITAL NATIVES

Piping

Elixir

```elixir
Enum.map(
  String.split(
    String.downcase(
      "ELIXIR-ELM"),
    "_"),
  &String.capitalize/1)
# ["Elixir", "Elm"]
```

DIGITAL NATIVES

Piping

Elixir

```elixir
"ELIXIR-ELM"
|> String.downcase()
|> String.split("-")
|> Enum.map(&String.capitalize/1)
# ["Elixir", "Elm"]
```

DIGITAL NATIVES

Pattern Matching

Elm

```
update msg state =
  case msg of
    Increment value -> { state | counter = counter + value }
    Decrement value -> { state | counter = counter - value }
```

DIGITAL NATIVES

Pattern Matching

Elixir

```elixir
%{name: name, age: age} = person = find_person(1)

def get_full_name(%{first_name: first_name, last_name: last_name}) do
  first_name <> " " <> last_name
end

case some_request() do
  nil -> {:error, "something went wrong."}
  result -> {:ok, result}
end
```

DIGITAL NATIVES

Immutability

DIGITAL NATIVES

OOP vs FP

- More abstractions
- Requires discipline and training
- Hidden and mutable state
- Dependent on state

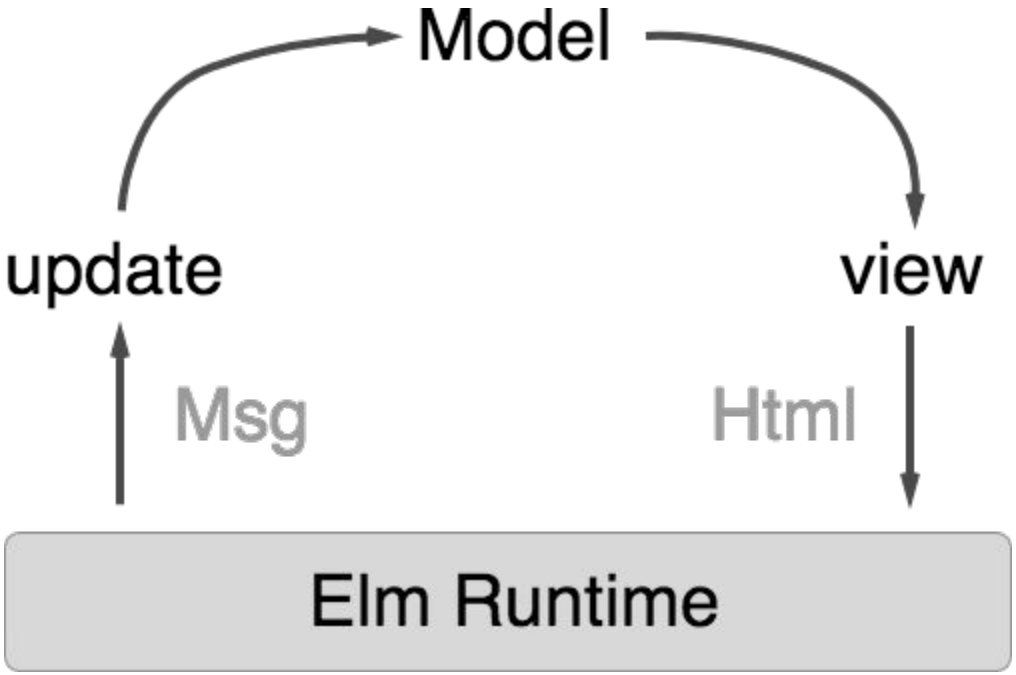
- Less abstractions
- Easier to program without too much experience
- Exposed and immutable state
- Independent on state

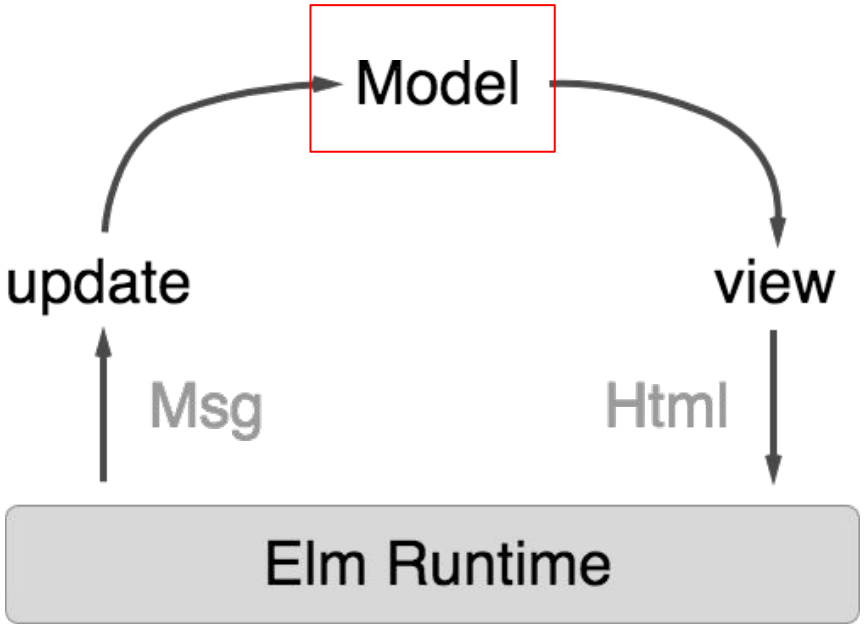
DIGITAL NATIVES

Runtime

DIGITAL NATIVES

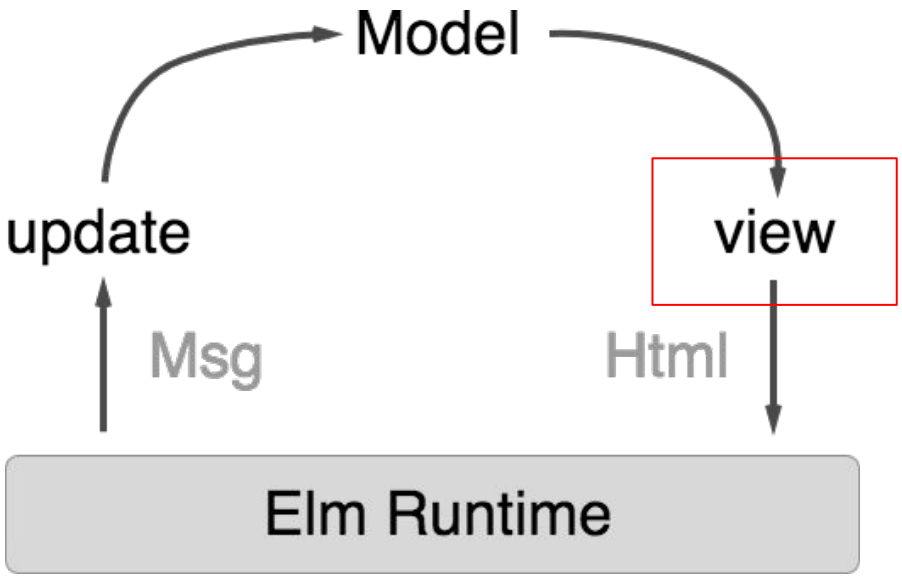
Elm Architecture





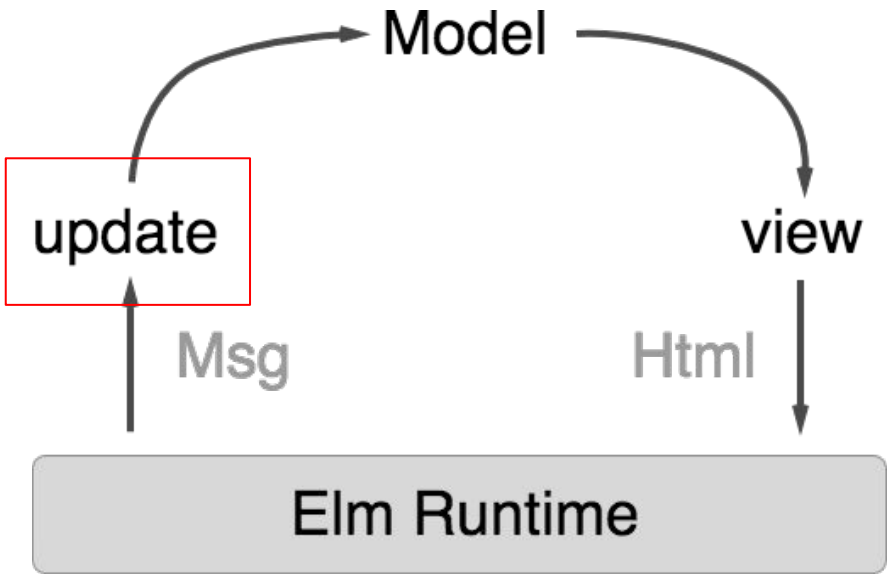
Model:

- State of the application
- Initial state definition is mandatory
- Leverages Elm types (records, union types)



view:

- It's a single pure function
- Receives the current state to be rendered
- Returns HTML and Messages
- Rendering leverages Virtual DOM

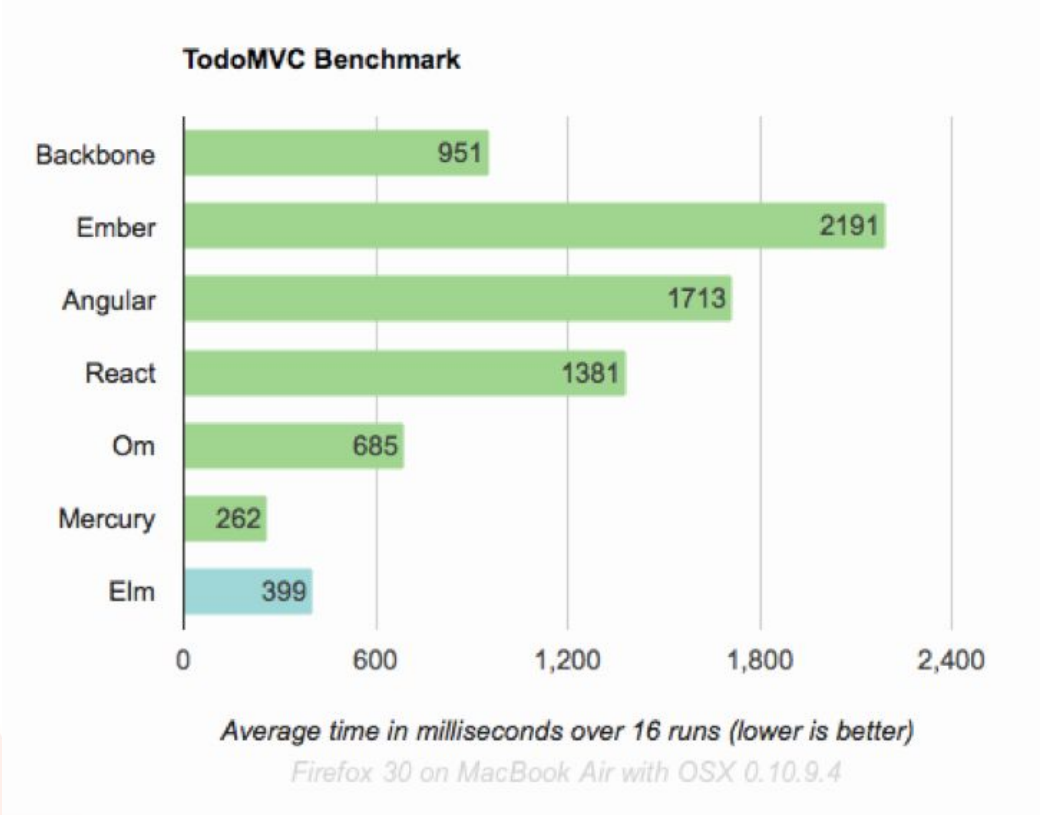


update:

- It's a single pure function
- Receives Message and current state as parameters
- Updates the state according to the message and content
- Returns the updated state

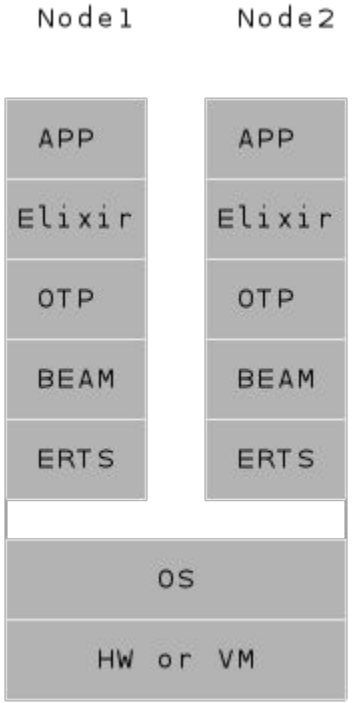
DIGITAL NATIVES

Elm



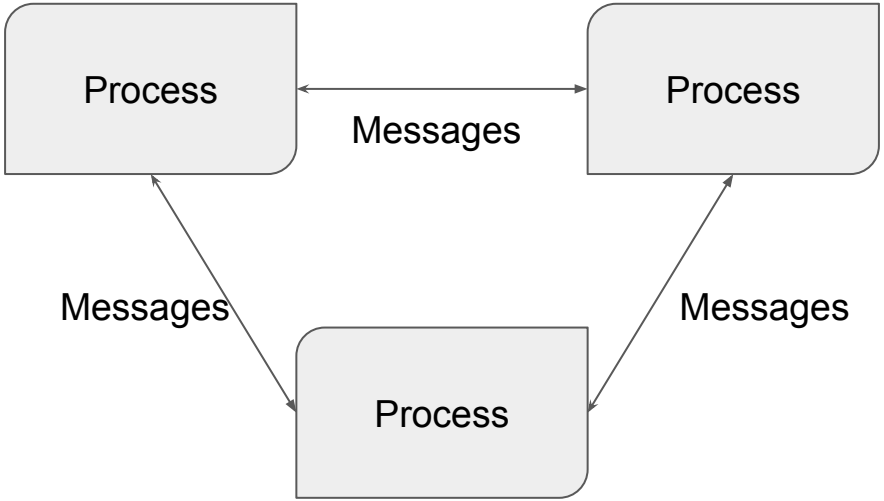
DIGITAL NATIVES

BEAM Architecture



DIGITAL NATIVES

BEAM Architecture



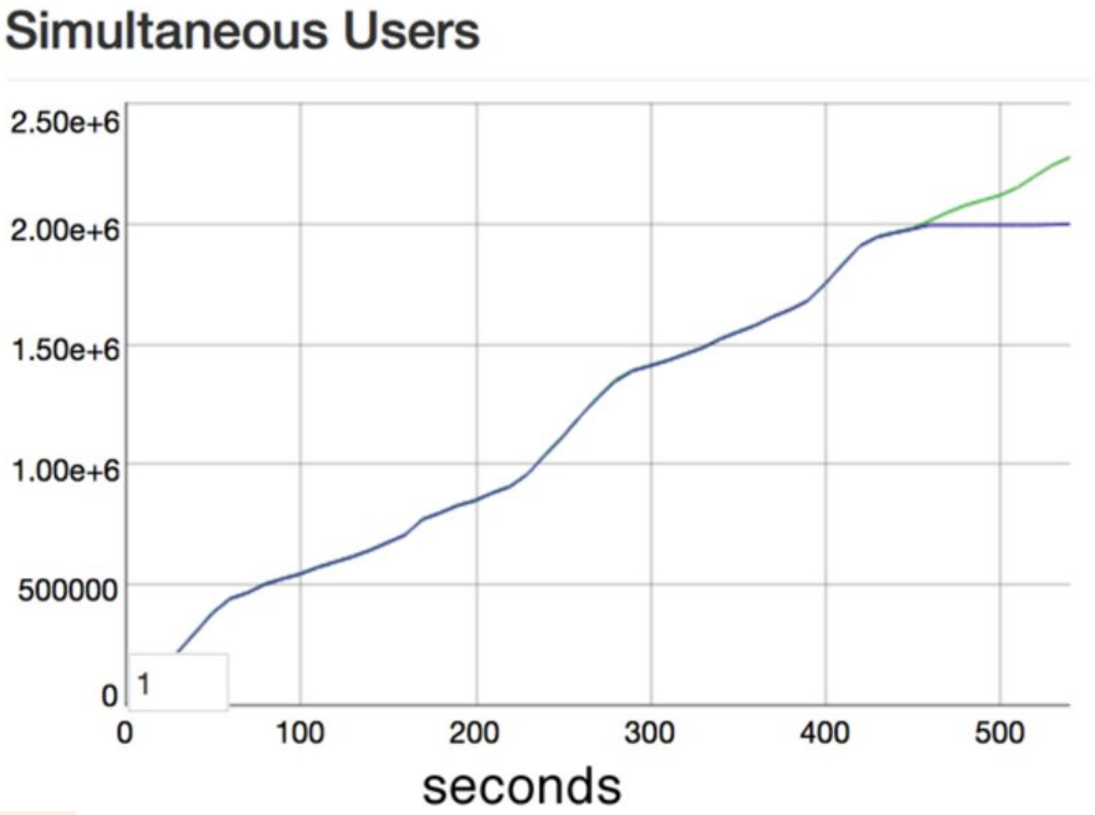
DIGITAL NATIVES

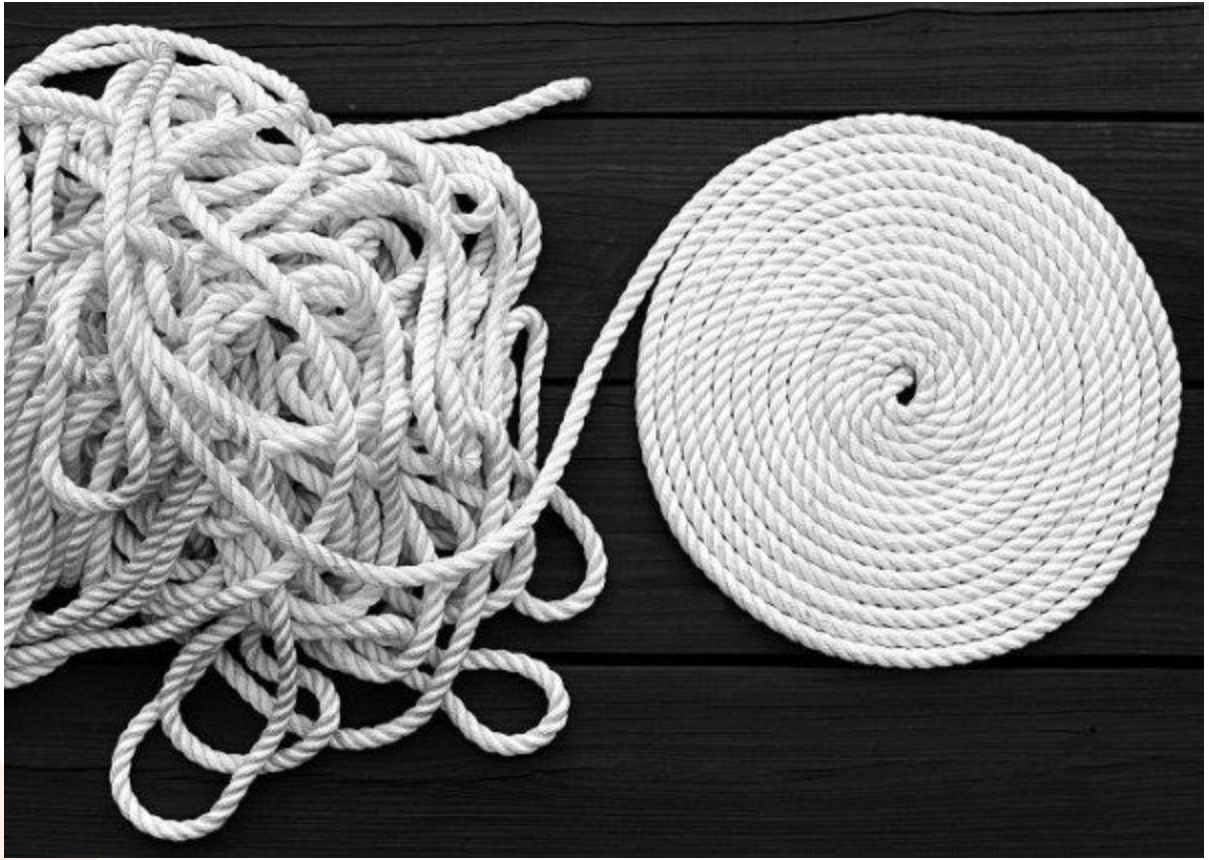
Elixir

Framework	Throughput (req/s)	Latency (ms)	Consistency (σ ms)
Gin	51483.20	1.94	0.63
Phoenix	43063.45	2.82	*(1)* 7.46
Express Cluster	27669.46	3.73	2.12
Martini	14798.46	6.81	10.34
Sinatra	9182.86	6.55	3.03
Express	9965.56	10.07	0.95
Rails	3274.81	17.25	6.88
Plug *(1)*	54948.14	3.83	12.40
Play *(2)*	63256.20	1.62	2.96

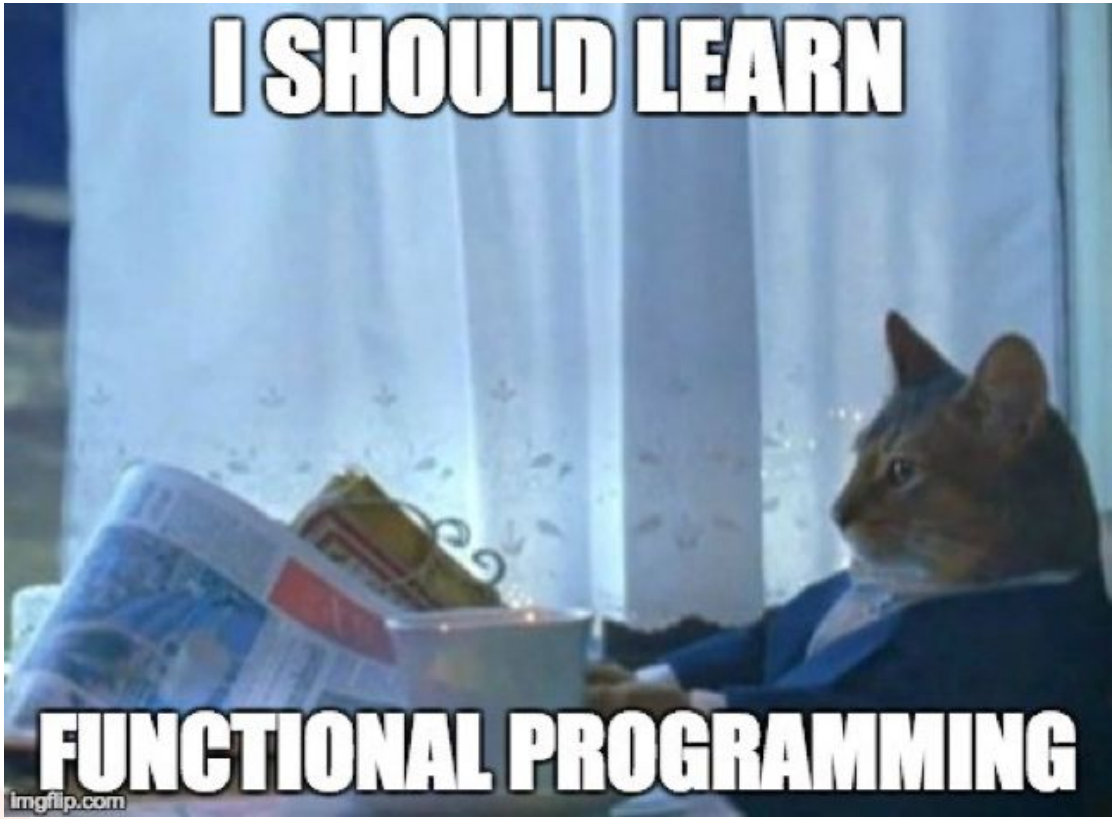
DIGITAL NATIVES

Elixir





DIGITAL NATIVES



Want to move to Europe?
We're Hiring!
digitalnatives.hu/jobs

 rhnonose

 RodrigoNonose

DIGITAL NATIVES

Links and Acknowledgements

- Code Examples: https://github.com/rhnonose/rubyconf_examples
- Elixir website: https://elixir-lang.org/
- Elm website: http://elm-lang.org/
- Beam architecture: https://happi.github.io/theBeamBook/
- Benchmark: https://github.com/mroth/phoenix-showdown
- Github and Twitter icons designed by Dave Gandy from Flaticon
- Design Patterns in Dynamic Languages: http://www.norvig.com/design-patterns/