Thinking Functionally

Elixir Taiwan Meetup
December 15, 2020

Jake Morrison jake@cogini.com

Cogini

Agenda

- What is functional programming?
- Shit talking

What is Functional Programming?

- More sophisticated type systems, e.g. Haskell
- "Pure" functions with no side effects
  - Function outputs depend only on inputs, like mathematics
  - No shared state
- Immutable data
- Functions as data
  - Higher-order functions, e.g. map
- Syntax, e.g. pattern matching, list comprehensions

Cogini

Benefits

- Easier to test
- Concurrency
- Easier to deal with faults in production
- All state is in function parameters, so logs are good
- Message passing

Cogini

Functional vs Object Oriented: Types

- OO connects behavior with types, i.e. object methods
- Functional programming uses types for safety
- Modern functional programming languages use type inference to reduce programmer overhead
- “If it will compile, it's correct”

Cogini

Erlang types

- Pattern matching at runtime
- “Let it crash”
- Hot code updates

Cogini

Type checking

- Optional type checking
- Typespecs
- Dialyzer
- Tagged tuples
  - `{:ok, value}` vs `{:error, reason}`
- Gleam https://gleam.run/
  - Types in the language vs types in the runtime, e.g. Typescript

Cogini

Immutable Data

- It's a good thing
  - Debugging multi-threaded C++ code is horrible
- Erlang does not allow mutating variables
  - Elixir allows it as syntax, but it’s fake
  - Actually re-binding
  - "Help, my variables are not varying!"
  - If you are mutating variables, you are probably doing something wrong
    - ✦ Except performance
    - ✦ And algorithms: https://www.amazon.com/Purely-Functional-Data-Structures-Okasaki/dp/0521663504

Cogini

Elixir types

- Structs are simply wrappers on Maps

-
  ```
  defmodule User do
     defstruct name: "John", age: 27
  end
  ```

-
  ```
  iex> %User{}
   %User{age: 27, name: "John"}
   iex> %User{name: "Meg"}
   %User{age: 27, name: "Meg"}
  ```

-
  ```
  iex> is_map(john)
   true
   iex> john.__struct__
   User
  ```

Functional vs Object Oriented: Nouns vs Verbs

- OO: No unbound methods
- FP: Standard algorithms with “meta-programming", lambda functions
- Lambda functions, Ruby "blocks" becoming popular
- Execution in the Kingdom of Nouns: http://steve-yegge.blogspot.tw/2006/03/execution-in-kingdom-of-nouns.html

Cogini

Functional vs Object Oriented: Polymorphism

- OO: Inheritance
- CLOS: multiple dispatch
- Elixir: pattern matching
- Elixir: Protocols

Cogini

Protocols

-
```
defprotocol Blank do
  @doc "Returns true if data is blank/empty”

  def blank?(data)
  end
end
```

Protocols

-
  ```
  defimpl Blank, for: Integer do
    def blank?(_), do: false
  end
  ```

-
  ```
  defimpl Blank, for: List do
    def blank?([]), do: true
    def blank?(_),  do: false
  end
  ```

-
  ```
  defimpl Blank, for: Map do
    # We could not pattern match on %{} because
    # it matches all maps. Check if the size
    # is zero (and size is a fast operation).
    def blank?(map), do: map_size(map) == 0
  end
  ```

Protocols

-
  ```
  defimpl Blank, for: Atom do
    def blank?(false), do: true
    def blank?(nil),      do: true
    def blank?( _ ),      do: false
  end
  ```

-
  ```
  defimpl Blank, for: User do
    def blank?(_), do: false
  end
  ```

Cogini

Protocols: JSON

-
  ```
  iex> IO.puts Poison.Encoder.encode([1, 2, 3], [])
  "[1,2,3]"
  ```

-
  ```
  defimpl Poison.Encoder, for: Person do
    def encode(%{name: name, age: age}, options) do
      Poison.Encoder.BitString.encode("#{name} (#{age})", options)
    end
  end
  ```

Higher Order Programming

- Functions as data
- Pass a function as a variable into another function
- Using functions to “specialize” common algorithms

Cogini

Higher Order Programming: Map

```
- iex> Enum.map([1, 2, 3], fn x -> x 2 end)
  [2, 4, 6]

- iex> Enum.map(%{1 => 2, 3 => 4}, fn {k, v} -> k v end)
  [2, 12]
```

Higher Order Programming: Fizz Buzz

-
  ```
  defmodule FizzBuzz do
    def fizzbuzz_check(n) do
      case {rem(n, 3), rem(n, 5)} do
        {0, 0} -> "FizzBuzz"
        {0, _} -> "Fizz"
        {_, 0} -> "Buzz"
        {_, _} -> n
      end
    end

    def fizbuzz do
      IO.inspect Enum.map(1..100, fizzbuzz_check/1)
    end
  end
  ```

Cogini

Higher Order Programming: Fold / Reduce

-
```
iex> List.foldl([1, 2, 3], 0, fn x, acc -> x + acc end)
6
```

Higher Order Programming: List Comprehensions

-
  ```
  for a <- list do
      …
  end
  ```

Higher Order Programming: Streams

-
  ```
  nums = Stream.iterate(1, &(&1 + 1))
  fizz = Stream.cycle ["", "", "Fizz"]
  buzz = Stream.cycle ["", "", "", "", "Buzz"]
  fizzbuzz = Stream.zip(fizz, buzz)
  |> Stream.zip(nums)
  |> Stream.map(fn
      {{"",                ""},  number} -> number
      {{fizzword, buzzword}, _number} -> fizzword <> buzzword
  end)
  fizzbuzz |> Stream.take(30) |> Enum.join("\n") |> IO.puts()
  ```
- Laziness

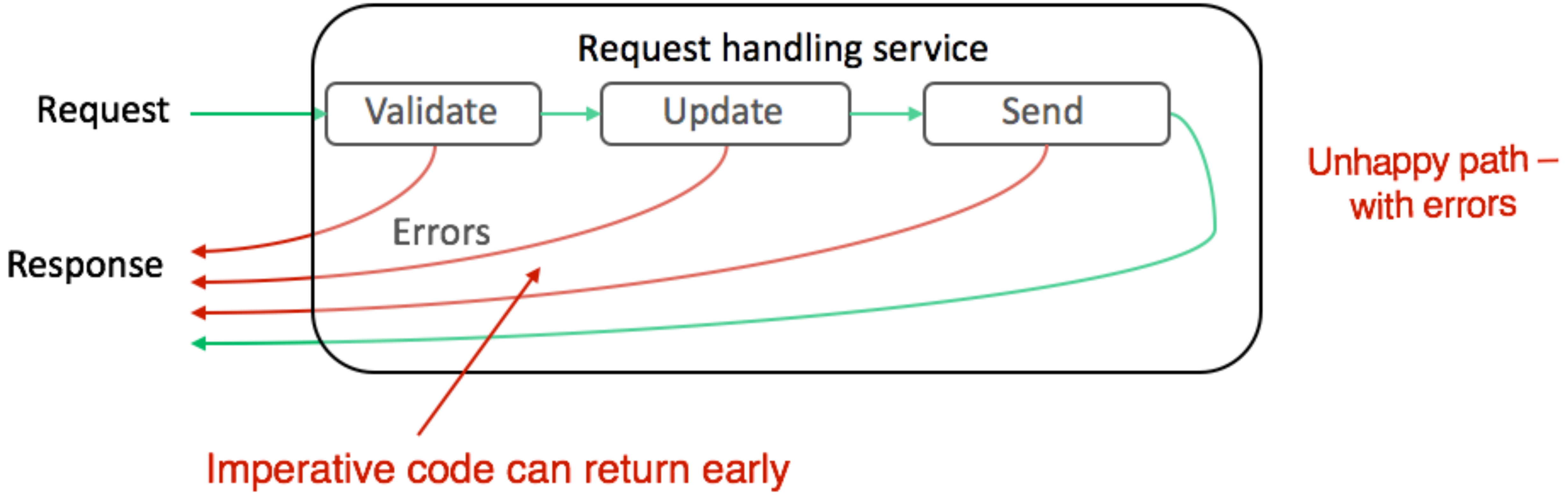
Cogini

Transforming data

- Phoenix: Handling a request is just a series of transformations
  - Take a request as input, transform it into a response
  - Plug “conn”
  - Ecto changesets
- Some dirty stuff in the middle
  - Database
  - Logging
- Error handling
  - Pattern matching
  - Functional core: “`with`” vs “pipe”
- Syntactic sugar: Plug framework

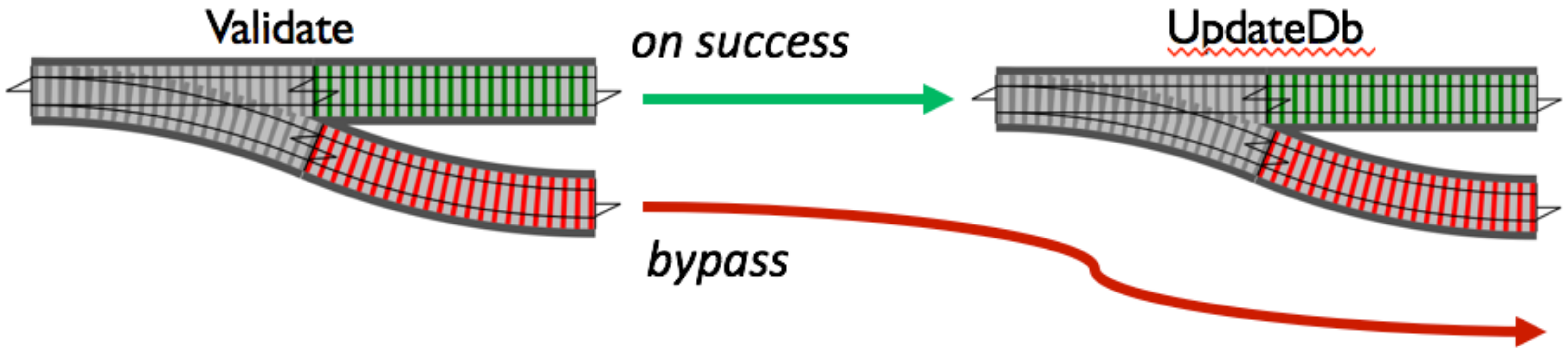
Cogini

Railway Oriented Programming

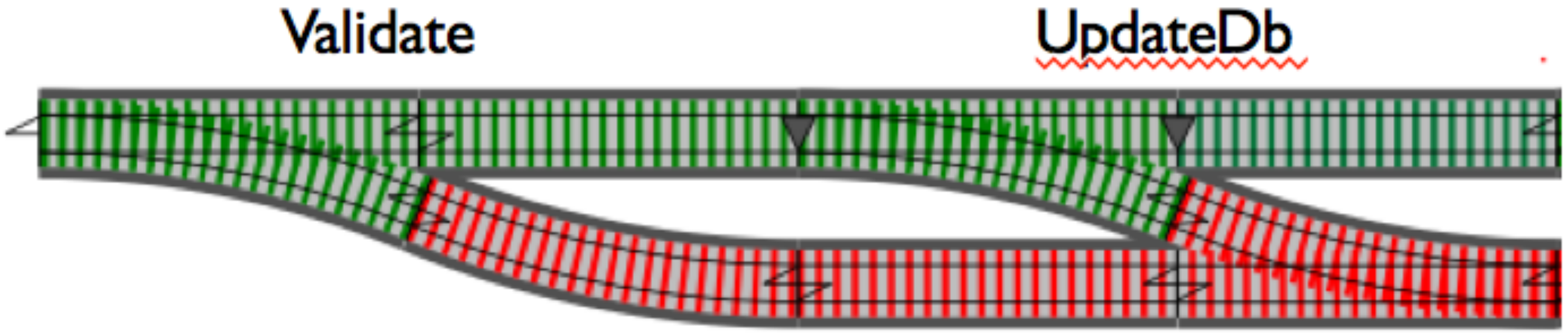


Cogini

Railway Oriented Programming

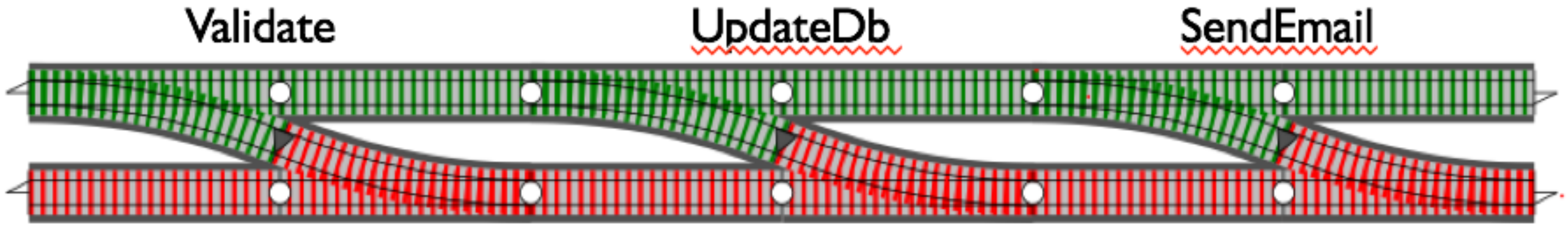


Railway Oriented Programming



-

Railway Oriented Programming



- http://zohaib.me/railway-programming-pattern-in-elixir/

History

- Model the real world
- Common behavior across multiple types
    - Share implementation code
- Code reuse

History

- C++ is better than C, because C got out of control
- C++ was a great way to make Windows GUIs
- C++ is just syntactic sugar
  - https://github.com/drh/cii
- C++ templates and generics, are they object oriented?
- Modules are good

Cogini

Heresy

- Objects have not proven to be a great way of modeling the world
    - Implementation inheritance in a framework vs domain
    - Lack of multiple inheritance in popular languages like Java
    - Relational model is fundamental math, not ORM
- Domain Driven Design
- Domain specific languages (lisp)
- SOLID principles
    - https://medium.com/@andreichernykh/solid-elixir-777584a9ccba

Macros!

- Compile time code generation
- https://littlelines.com/blog/2014/07/08/elixir-vs-ruby-showdown-phoenix-vs-rails
- Rails metaprogramming is insane, macros are easy
- Is it all just code generation plus pattern matching?
  - http://www.gar1t.com/blog/solving-embarrassingly-obvious-problems-in-erlang.html

Cogini

The Age of Concurrency

- Objects are incompatible with concurrency
  - Every object is a bug waiting to happen
  - Singletons
  - Or anything
  - Lock everything?
- Async / await
  - Not really concurrent
  - Node.js: started with callbacks, then promises
  - Syntactic sugar
  - Twisted Python has been doing this for 10 years, and we know how it ends (badly)
- Message passing concurrency model

Cogini

Message Passing

- Erlang is a *truly* object oriented language, unlike all these pretenders
- In Smalltalk, calling a method is sending a message to an object.
- How do objects in the real world communicate? By sending messages.
- So Erlang is the most object oriented language there is.
  - Start a GenServer process
  - Send a message to it, and it will update its state and send a response back
  - Requests are serialized, kept in the mailbox. Only one request is active at a time.
- Don’t do this!
- Model the natural concurrency of your system

Cogini

Questions?

- jake@cogini.com
- @reachfh

Cogini