Vancouver Functional Programmers

BROOKLYN ZELENKA

ELIXIR AND PHOENIX FOR RUBYSISTS

WHAT WE'RE GOING TO COVER

ELIXIR (LANGUAGE)

- Background
- Syntax
  - Compare with Ruby
  - Extra features
- Functional Programming Basics
- Tooling (Mix, Hex, Dialyzer, and ExUnit)
- A *tiny* bit of OTP

PHOENIX (WEB FRAMEWORK)

- Project layout
- Compare with Rails
  - Similarities
  - Differences
  - Extra features
- Live code a simple app

PART ONE

ELIXIR

WHAT IS ELIXIR?

- Runs on BEAM (Erlang virtual machine)
  - Developed by Ericsson
  - Battled tested
    - Major telecom
    - Almost 30 years
- Compiled, dynamically typed language
- Focus on scalability, concurrency, and fault tolerance
- Functional language
- Immutable by default

WHAT ELIXIR IS NOT

- Ruby++
- Friendly syntax for Erlang
- Object-oriented

I WOULDN'T CLASSIFY ELIXIR AS A BETTER RUBY. SURE, RUBY WAS MY MAIN LANGUAGE BEFORE ELIXIR, BUT PART OF MY WORK/ RESEARCH ON CREATING ELIXIR WAS EXACTLY TO BROADEN THIS EXPERIENCE AS MUCH AS POSSIBLE AND GET SOME MILEAGE ON OTHER ECOSYSTEMS SO I DIDN'T BRING A BIASED VIEW TO ELIXIR. IT IS NOT (A BETTER) RUBY, IT IS NOT (A BETTER) ERLANG, IT IS ITS OWN LANGUAGE.

José Valim, Elixir's BDFL

BUT I WANT A FAST, FUNCTIONAL RUBY!

- You should check out *Clojure*
- Once you get over the parentheses, the semantics are much closer
- Legend has it that Matz originally started Ruby as a Lisp
- Great community, tons of libs, Java & JS interop, and so on…

ANYWAY…

FUNCTIONAL PROGRAMMING: 101

- Data-first
- Explicit state rather than data hiding
- Limit and isolate side effects
- Referential transparency
- Composition over inheritance
- Expressions rather than objects & messages
- Everything is an expression
- Not mutually exclusive with OO (see Scala, Swift, and Rust)

WHY CARE ABOUT FUNCTIONAL PROGRAMMING?

- The free lunch is over
- Clean, maintainable systems
- Abstractions
- Even higher level code
  - Focused on meaning and intent, rather than machine instructions
- Highly reusable

ELIXIR IS “WEAKLY TYPED”

- “Weak” is technical, not derogatory
  - Does its own type conversion as needed
    - Nice for integers vs floats
- No built-in ADTs
  - Shameless self-plug: ADTs coming soon in a lib!
- Type annotations
  - @spec add(integer, integer) :: integer

ELIXIR'S BUILT-IN TYPES

- Atoms (similar to Ruby's symbols)
  - :ok
  - :foo
- "pid"s
- Integers
- Floats
- Keyword lists
- Binaries
- Characters
- Character lists
- Strings
- Maps
- Structs
- Dicts

ELIXIR'S TRUTHINESS TABLE

	TRUTHY	FALSEY
TRUE	✓	
FALSE		✓
nil		✓
""	✓	
[]	✓	
anything else	✓	

ERLANG'S LEGACY

- Transmits binary streams <<1,0,1>>
- OTP all the things
- Explicit goal to *match* (or beat) Erlang in terms of performance
- Runs pretty much everywhere

RUBY

```
# Cool stuff ahead
module Foo

   def hello(name = nil, *names)

     case name

     when nil

       “Hello, world!”

     when names.empty?

       “Hello, #{ name }”

     else

       hello(name) + hello(names)

   end

end
```

ELIXIR

```
# Cool stuff ahead
defmodule Foo do

   def hello, do: “Hello, world!”

   def hello([head|tail]) do

     hello(head) <> hello(tail)

   end

   def hello(name), do: “Hi, #{ name }”

end
```

hello/0

pattern matching

hello/1 on lists

hello/1 on anything

FUNCTION COMPOSITION

- Pipeline operator
  - |>
- Remember “g o f” from way back in high school?
- Pipeline is backwards
  - ie: The “forward”, or operational order
  - g(f(x)) == (g o f) x == x |> f |> g

THE PIPELINE OPERATOR |>

Ruby: message chaining	Elixir: pipe, or (forward) composition
[1,2,3].sum.divide(5).floor	[1,2,3] |> Enum.sum |> divide(5) |> floor
[1,2,3].sum = 6	Enum.sum([1,2,3]) = 6
6.divide(5) = 1.2	*First* argument divide(6, 5) = 1.2
1.2.floor = 1	Float.floor(1.2) = 1

CLASSES VS MODULES, PROTOCOLS, AND STRUCTS

```
class Foo
  def initialize(bar, quux)
    @bar = bar
    @quux = quux
  end

  def add
    @bar + @quux
  end
end

module Bar
  def add(a, b)
    a + b
  end
end

Foo.new(1, 2).add
```

```
defmodule Foo do
  defstruct bar: nil, quux: nil

  ~~def add(%Foo{bar: b, quux: q}), do: b + q~~
end
```

Interface, not implementation {

```
defprotocol Mathy do
  def add(struct)
  def add(a, b)
end
```

Implementation {

```
defimpl Mathy, for: Foo do
  def add(%Foo{bar: b, quux: q}), do: b + q
end
```

Implementation {

```
defimpl Mathy, for: Bar do
  def add(%Bar{a: a, b: b}), do: a + b
end
```

SOMEWHAT-DIFFERENT-FROM-USUAL MACROS

- Macros are functions that run at compile time, not runtime
- “Code that writes code”
- quote and unquote
- Elixir gives an AST, rather than tokens or syntax
- WARNING: harder to reason about!
    - Only use when a regular function can’t get the job done
    - Can switch behaviours based on environment (prod vs dev vs test)
    - Generate large scaffolds
    - Make cleaner code
    - and more!

- Canonical “unless” example:

```
defmodule MyCoolUnless do
  defmacro unless_m do
    quote do
      if(!unquote(clause), do: unquote(expression))
    end
  end
end
```

```
iex> require MyCoolUnless
iex> Unless.unless_m true, IO.puts “don’t print this"
nil
```

```
iex> Unless.unless_fun true, IO.puts “don’t print this"
“don’t print this”
nil
```

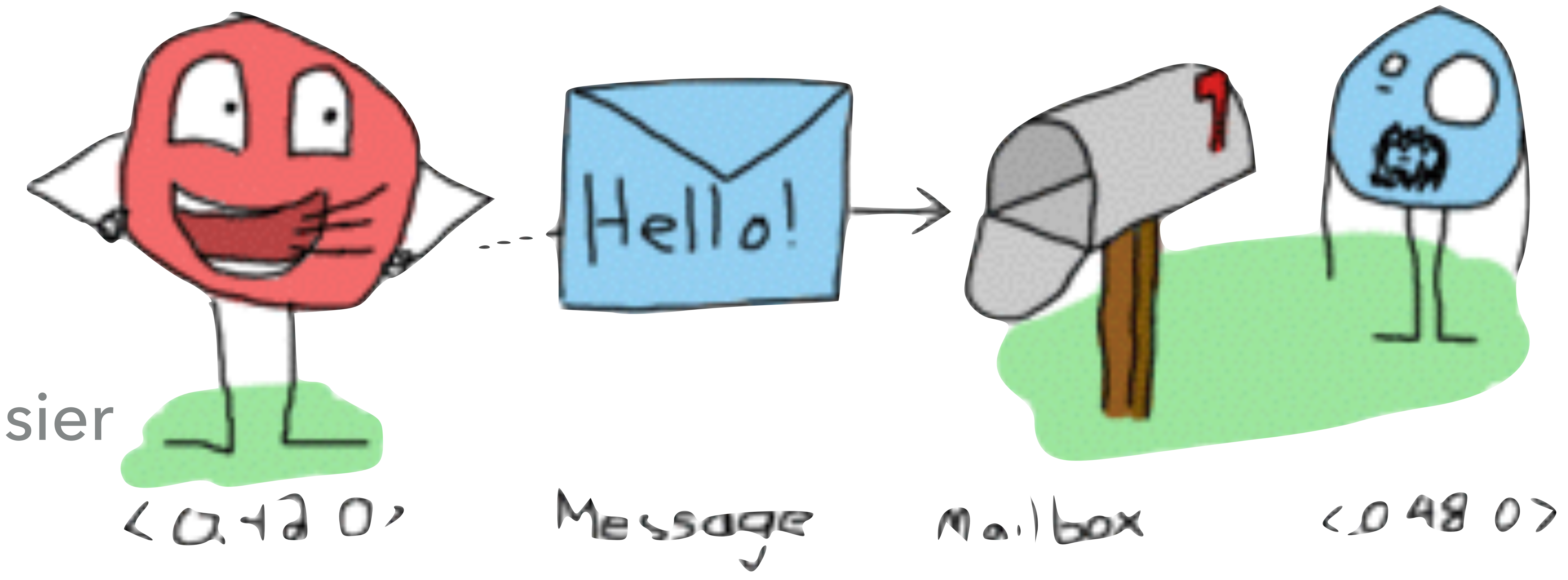
IEX, MIX, HEX, ECTO, DIALYZER, & EXUNIT

- IEx is Elixir's IRB
- Mix is roughly Elixir's Bundler
- Hex is roughly `gem` + RubyGems
- Ecto is a database interface (will see bit more in Phoenix section)
- Dialyzer is a static analysis tool
  - Annotate your code with @spec to ensure that types will line up
  - Will tell you about errors in branching paths, error handling, and so on
- ExUnit
  - Built-in unit testing framework with nice `assert` syntax
  - ex. `assert response.status == 200`

PART TWO

KILLER FEATURE: CONCURRENCY

THE ACTOR MODEL



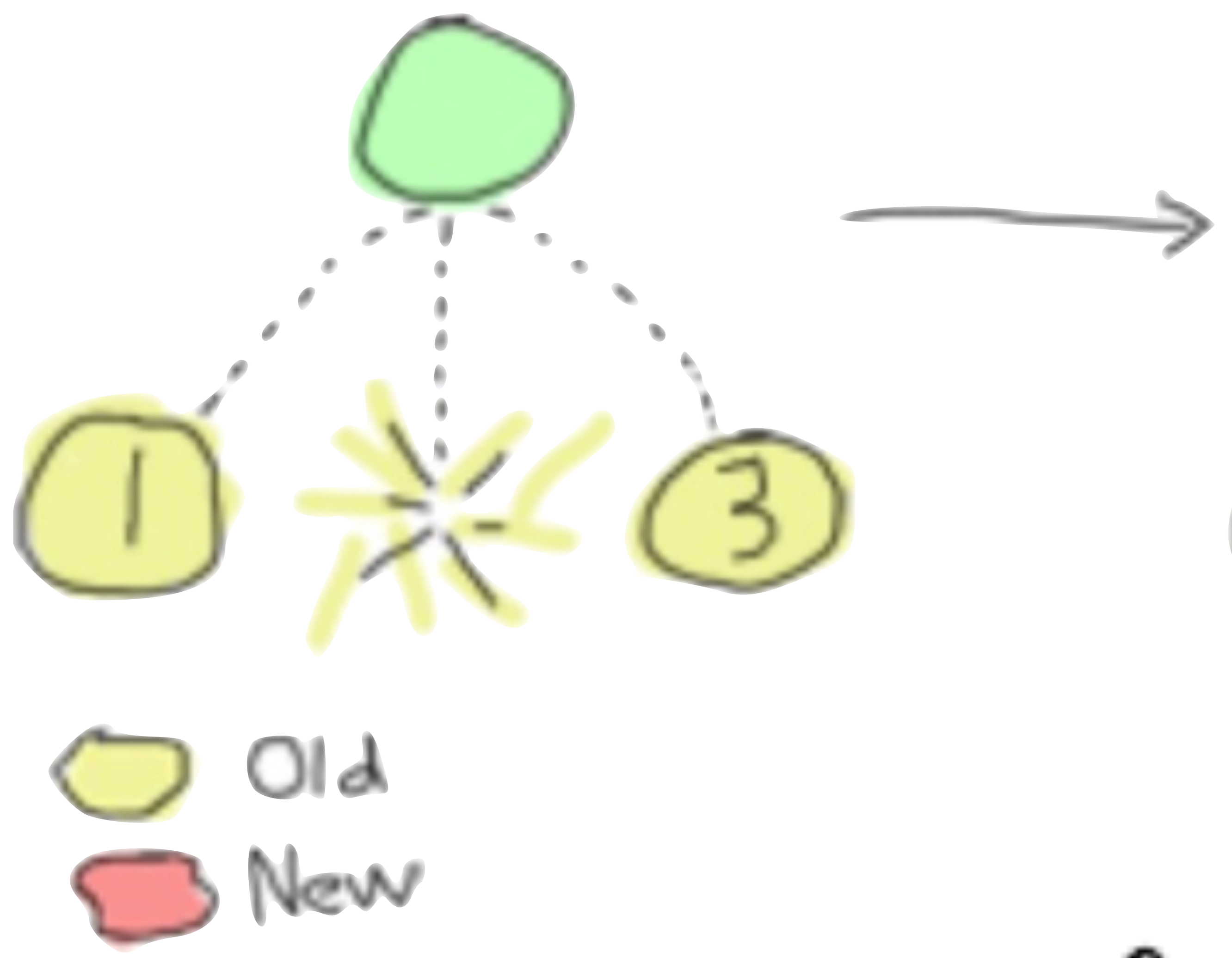
- Concurrency is hard
  - Erlang/Elixir tries to make it easier
- Processes are “actors”
- Mailboxes (queue)
- Do something when receive a message
- Optionally, reply to sender
- Don’t be afraid to “kill your children”

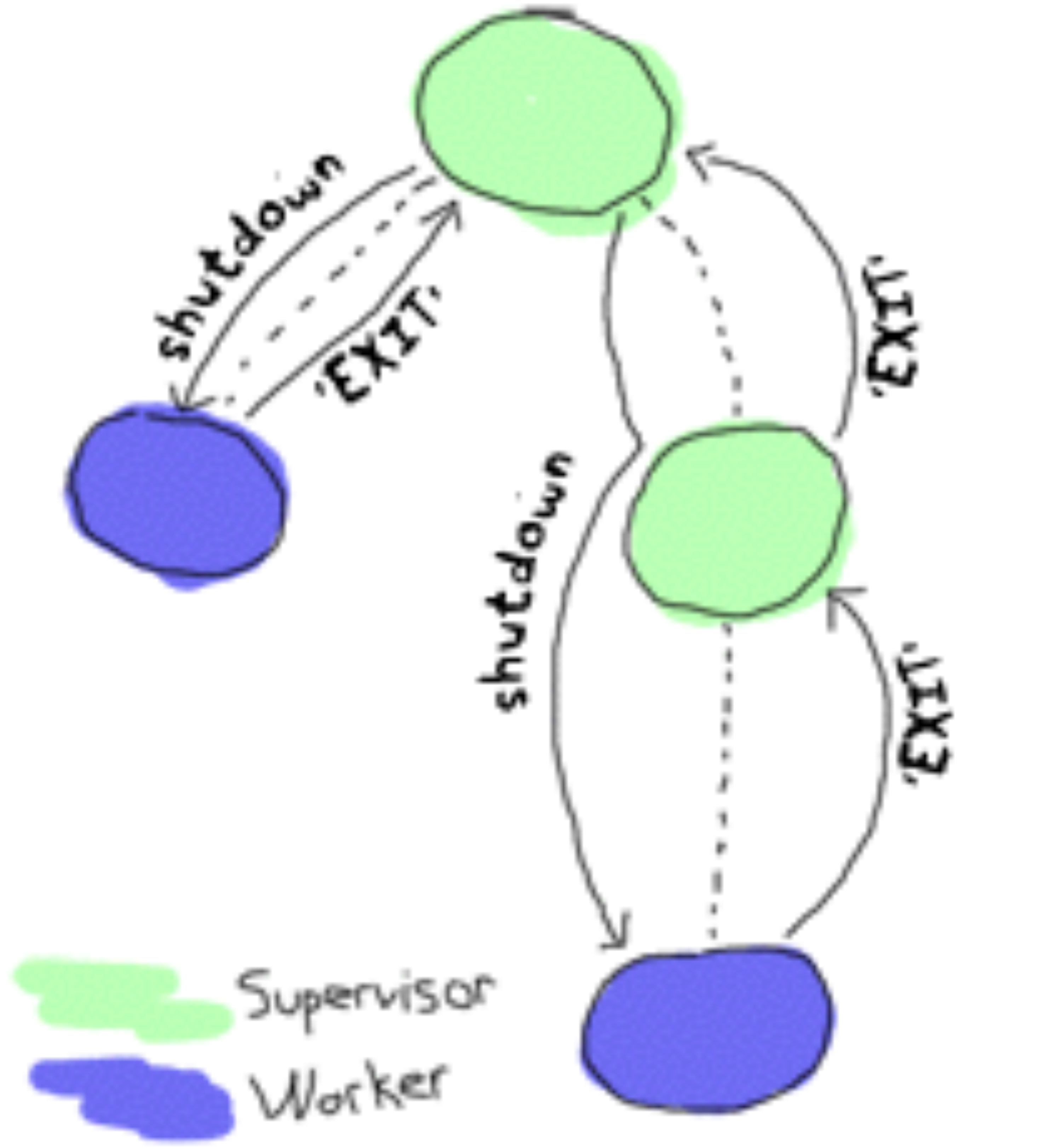
SENDING MESSAGES IS SIMPLE!

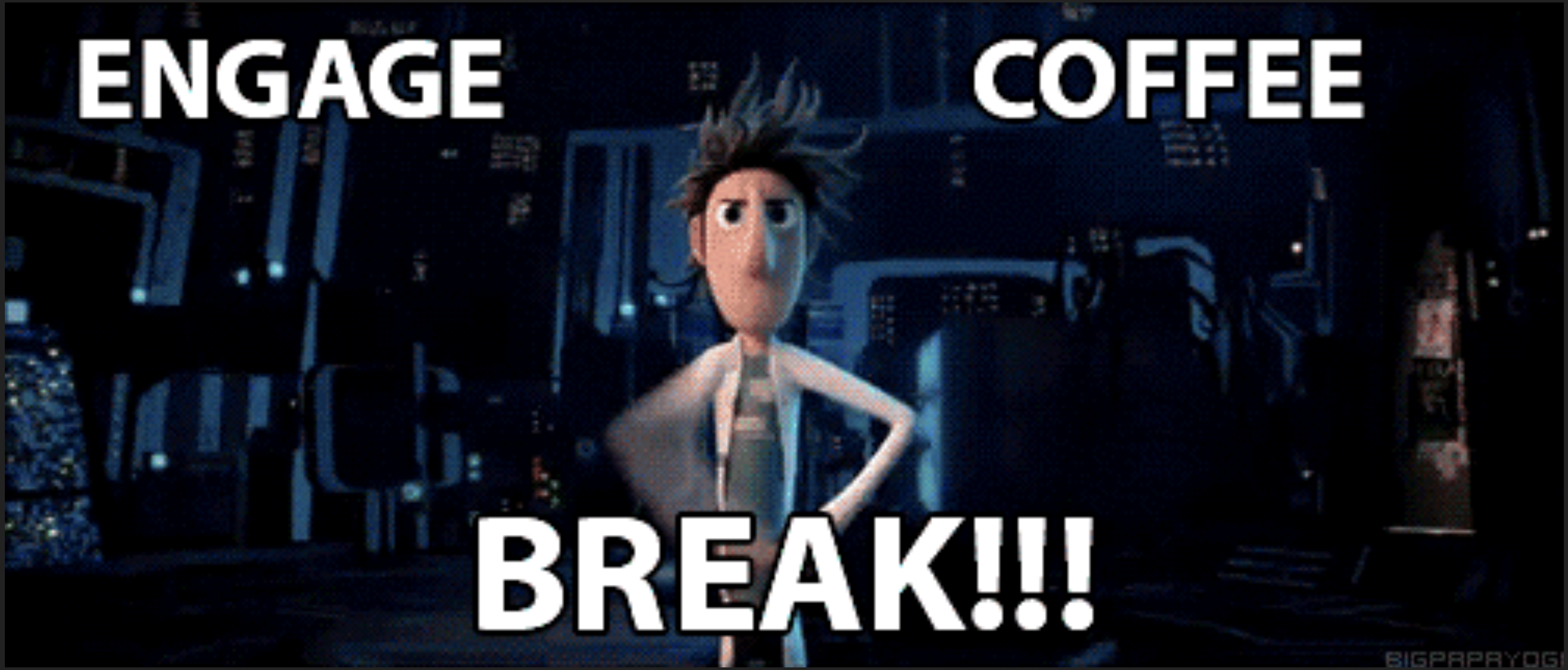
```
iex> a = spawn(Foo, :bar, []) <~Module, function, args
iex> a |> send({self, {3, 2, 1}})
iex> flush

12 3
:ok
```

OPEN TELECOM PLATFORM (OTP)

- Library, framework, and much more!
- Debugger, databases
- Common patterns, including
  - Supervisors and workers
  - GenServer (Generic Server, behaviours for OTP to call)
  - Sync and async
  - Restart strategies
    - one_for_one, rest_for_all, rest_for_one







PART THREE

SHORT INTERMISSION

PART FOUR

PHOENIX

WHAT IS PHOENIX?

- Server-side web framework
- Soft-realtime features
  - Channels
- Distributed
- Attention to serving web APIs
- Well separated concerns
- Just hit 1.0

Phoenix Framework™

SIMILAR TO RAILS

- MVC (and then some)
- Plugins
- Migrations
- EEx is similar to ERB
  - <%= some stuff %>
- Path helpers
- Router
- Schema
- Generators

MAJOR DIFFERENCES FROM RAILS

- No ActiveRecord
  - Closer to Data Mapper
- Request cycle is clearer
- Soft real time
  - Sockets, etc
- View models
- Schemas are kept in each model

PERFORMANCE

- In Rails, a 200-300ms response time is not uncommon (without a cache)
  - Can get that down to ~50ms range
- With Phoenix, we often see results in the µs (*micro*second) range
- Hear various stats, most roughly 10-40x performance over Rails

µs

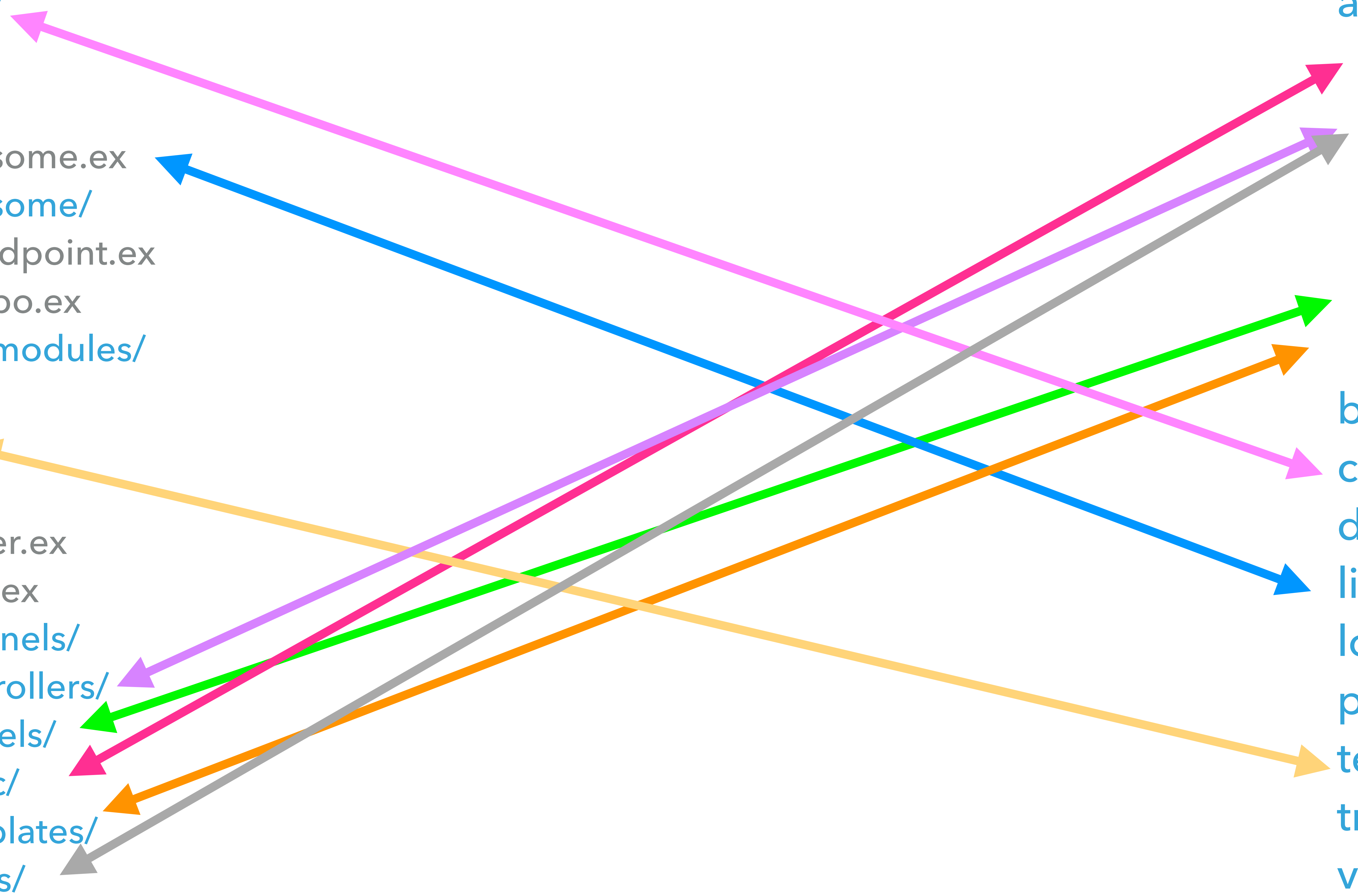
DIRECTORY STRUCTURE

mix phoenix.new awesome

```
awesome/
  README.md
  mix.exs
  package.json
  brunch-config.js
  _build/
  config/
  deps/
  lib/
    awesome.ex
    awesome/
      endpoint.ex
      repo.ex
  node_modules/
  priv/
  test/
  web/
    router.ex
    web.ex
    channels/
    controllers/
    models/
    static/
    templates/
    views/
```

rails new awesome

```
awesome/
  README.rdoc
  Gemfile
  Rakefile
  config.ru
  app/
    assets/
    controllers/
    helpers/
    mailers/
    models/
    views/
  bin/
  config/
  db/
  lib/
  log/
  public/
  test/
  tmp/
  vendor/
```



CHANNELS VS PROCESSES

- Channels are layers
- PubSub on steroids
  - Senders and receivers can switch roles on the topic at any time
  - Don't even have to be Elixir/Erlang processes
    - Could be a Rails server, JS client, Android app, mix & match, and so on
- Have their own routing system
- Fallback
  - Ex. sockets will fall back to polling

PART FIVE

LIVE CODING

“WHAT COULD POSSIBLY GO WRONG?”