A WHIRLWIND TOUR OF ELIXIR

BUFFER "SNACK CHAT"

📪🤿 PIPES, MAILBOXES, & THE ZEN OF JUST LETTING YOUR APP CRASH ☠️🚀

BACKGROUND

BACKGROUND

A BIT OF INFO ABOUT THE TALK AND CONTEXT

BACKGROUND

BROOKLYN ZELENKA

BACKGROUND

BROOKLYN ZELENKA

BACKGROUND

BROOKLYN ZELENKA

- Cofounder of fission
  - Web3 tools for a Web 2.0 world
  - Early access: https://tools.fission.codes
  - Code: https://github.com/fission-suite

BACKGROUND

BROOKLYN ZELENKA

- Cofounder of fission
  - Web3 tools for a Web 2.0 world
  - Early access: https://tools.fission.codes
  - Code: https://github.com/fission-suite
- PLT & VM enthusiast

BACKGROUND

BROOKLYN ZELENKA

- Cofounder of fission
  - Web3 tools for a Web 2.0 world
  - Early access: https://tools.fission.codes
  - Code: https://github.com/fission-suite
- PLT & VM enthusiast
- Ethereum Core Developer

BACKGROUND

BROOKLYN ZELENKA

- Cofounder of fission
  - Web3 tools for a Web 2.0 world
  - Early access: https://tools.fission.codes
  - Code: https://github.com/fission-suite
- PLT & VM enthusiast
- Ethereum Core Developer
- Previously a FT Elixir developer
  - Exceptional
  - Witchcraft Suite
    - https://github.com/witchcrafters
  - Keynotes
    - Elixir.LDN 2017 / Empex 2018 / ElixirConf 2019

BACKGROUND

GOALS

BACKGROUND

GOALS

- Situate Elixir
- Main characteristics & concepts
- Briefly touch on Phoenix

BACKGROUND

WHY ELIXIR?

BACKGROUND

WHY ELIXIR?

- According to the 2019 Stack Overflow developer survey:
  - Developers who work with Rust, WebAssembly, and **Elixir** contribute to **open source at the highest rates**
  - Globally, respondents who use Clojure, F#, **Elixir**, and Rust earn the **highest salaries**

BACKGROUND

FUNCTIONAL PROGRAMMING

BACKGROUND

FUNCTIONAL PROGRAMMING

Alan Turing	Alonzo Church

BACKGROUND

FUNCTIONAL PROGRAMMING

Alan Turing	Alonzo Church
Imperative instructions	Expressions & analytic truths

BACKGROUND

FUNCTIONAL PROGRAMMING

Alan Turing	Alonzo Church
Imperative instructions	Expressions & analytic truths
Global state — “memory”	Manual, localized state — “stateless”

BACKGROUND

FUNCTIONAL PROGRAMMING

Alan Turing	Alonzo Church
Imperative instructions	Expressions & analytic truths
Global state — “memory”	Manual, localized state — “stateless”
Mechanistic, Turing machines	Semantic, mathematical models

ERLANG & ELIXIR

ERLANG & ELIXIR

A BRIEF HISTORY OF THE BEAM & OPEN TELECOM PLATFORM

BEAM

ERLANG

ERLANG

BEAM • ERLANG

HIGH LEVEL

BEAM ● ERLANG

HIGH LEVEL

- Name
  - Agner Krarup Erlang
  - “Ericsson Language”
- High performance 1980s telecom switches
- Prolog-ish syntax
- Same VM as Elixir (BEAM)
- Full introp
- Actor model

```erlang
-module(count_to_ten).
-export([count_to_ten/0]).

count_to_ten() → do_count(0).

do_count(10) → 10;
do_count(N) → do_count(N + 1).
```

BEAM

ELIXIR

BEAM

ELIXIR

BEAM • ELIXIR

HIGH LEVEL

BEAM • ELIXIR

HIGH LEVEL

- Released in 2011
- BDFL José Valim & co
- Several Rails core members
- Runs on Erlang's BEAM VM
- Fully interoperable with Erlang
- Concurrent, fault tolerant, &c

```
defmodule CountToTen do
  def count_to_ten, do: do_count(0)

  defp do_count(10), do: 10
  defp do_count(n), do: do_count(n + 1)
end
```

BEAM • ELIXIR

PHILOSOPHY

BEAM ● ELIXIR

PHILOSOPHY

- Readability
- Consistency
- Fault tolerance
- Light weight processes
- Soft realtime
- Modernity (ex. UTF-8 support out of the box)
- Extension / metaprogarmming

BEAM • ELIXIR

RUBY INFLUENCE 💎

BEAM ● ELIXIR

RUBY INFLUENCE 💎

- Philosophy
  - Friendly
  - Pragmatic
  - Readable (“like English”)
  - Permissive
  - Dynamic
- Superficial aesthetics
  - do blocks
  - Dot syntax
  - Truthiness

BEAM

IN THE WILD

BEAM

IN THE WILD

- WhatsApp (2 million connections on a single node)
- “Every phone call since 1986”
- Discord
- Amazon
- Facebook
- Motorola
- Ericsson (obviously)
- Heroku
- RabbitMQ
- Nerves
- Parallela

BEAM

MOTIVATING CASES

BEAM

MOTIVATING CASES

- Performance free lunch is over
- Networking (ie: internet)
- High availability
- “Nine nines” of uptime
  - 99.9999999%
  - ~32ms total downtime **per year**
- Soft realtime
- Fault tolerant
- Concurrency
- Distributed computing
- Hot code reloading
- Clustering

SALIENT SYNTAX

SALIENT SYNTAX

FIRST THINGS THAT YOU'LL NOTICE

SALIENT SYNTAX

PATTERN MATCHING & MULTIPLE FUNCTION HEADS

```
defmodule WowMath do
  def div(_, 0), do: 0
  def div(a, b), do: a / b
end
```

SALIENT SYNTAX

VARIADIC FUNCTIONS

```elixir
defmodule WowMath do
  # Anonymous function arity
  # add_anon = fn(x,y) -> x + y end
  # add_anon.(1, 2) # add/0

  def add(), do: 10

  # add/1
  def add(a), do: a + 10

  # add/2
  def add(a, b), do: a + b

  # add/3, despite the default value
  def add(a, b, c \\ 10), do: a + b + c
end
```

SALIENT SYNTAX

PIPES

SALIENT SYNTAX

PIPES

- Reorder the flow of functions
- Linearize function calls for readability
- Goes into the first argument position of the next function
- Conceptually there is a “subject”

```
12345
|> Integer.digits() # ⇒ [1, 2, 3, 4, 5]
|> Enum.count() # ⇒ 5
|> Integer.floor_div(3) # ⇒ 1

# Equivalent to
Integer.floor_div(Enum.count(Integer.digits(5)), 3) # ⇒ 1
```

SALIENT SYNTAX

PROTOCOLS

```
defprotocol Semigroup do
  @doc "Sticks two things together"
  def concat(a, b)
end
```

SALIENT SYNTAX

PROTOCOLS

```elixir
defprotocol Semigroup do
  @doc "Sticks two things together"
  def concat(a, b)
end

defimpl Semigroup, for: Integer do
  @doc """
      iex> Semigroup.concat(3, 4)
      7
  """
  def concat(int_a, int_b), do: int_a + int_b
end

defimpl Semigroup, for: List do
  @doc """
      iex> Semigroup.concat([1, 2, 3], [4, 5, 6])
      [1, 2, 3, 4, 5, 6]
  """
  def concat(list_a, list_b), do: list_a ++ list_b
end

defimpl Semigroup, for: Bitstring do
  @doc """
      iex> Semigroup.concat("hi", "there")
      "hi there"
  """
  def concat(string_a, string_b), do: string_a <> string_b
end
```

CONCURRENCY

CONCURRENCY

THE BEAM'S RAISON D'ÊTRE

CONCURRENCY

LIGHTWEIGHT

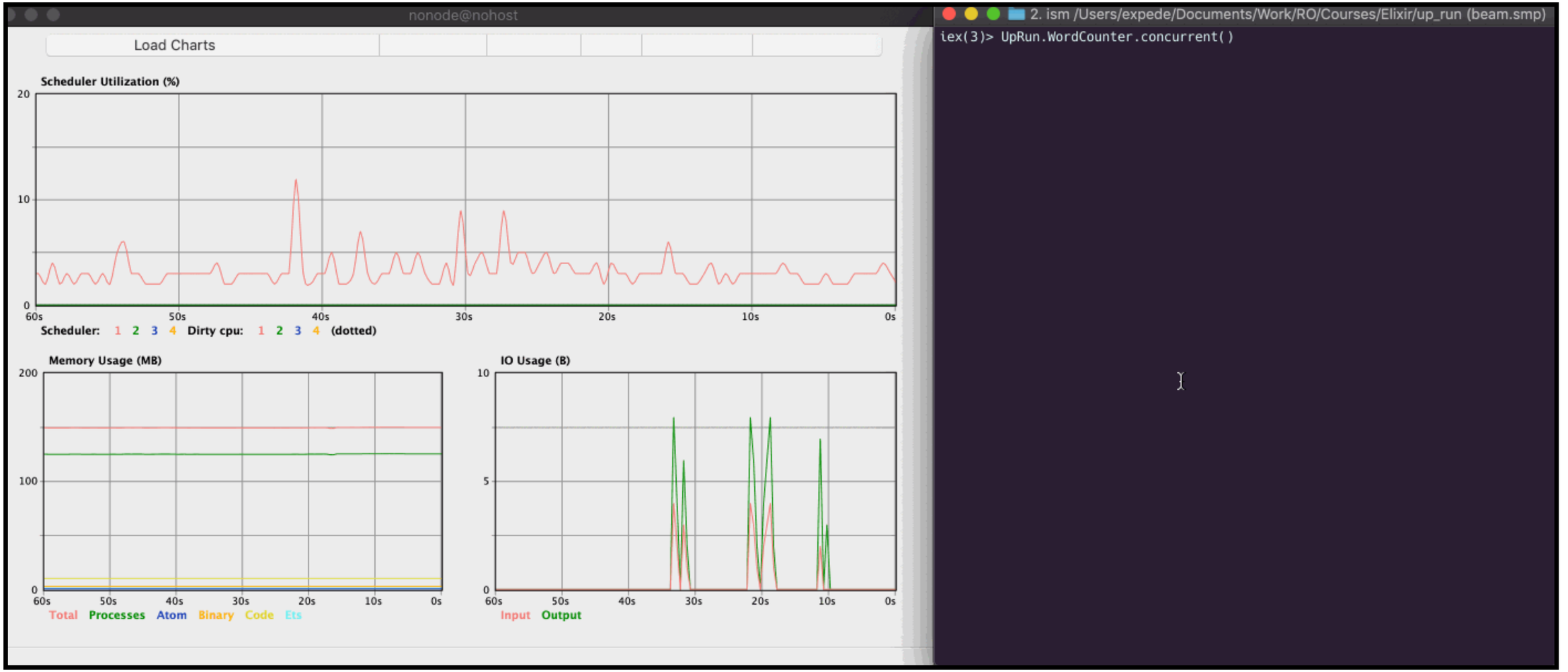
CONCURRENCY

LIGHTWEIGHT

- Lightweight virtual green threads
- Spin up millions of processes easily
- spawn ~ fork

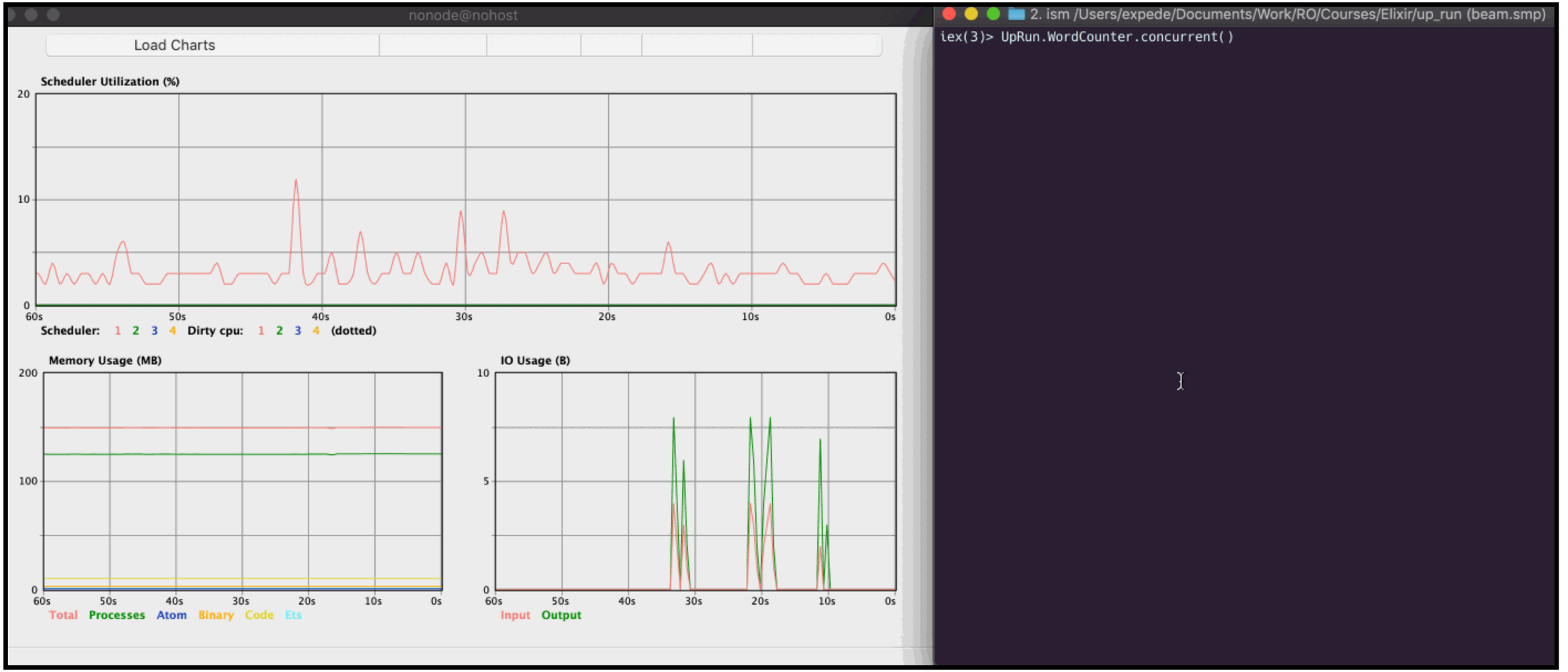
CONCURRENCY

"WAR & PEACE" WORD FREQUENCY



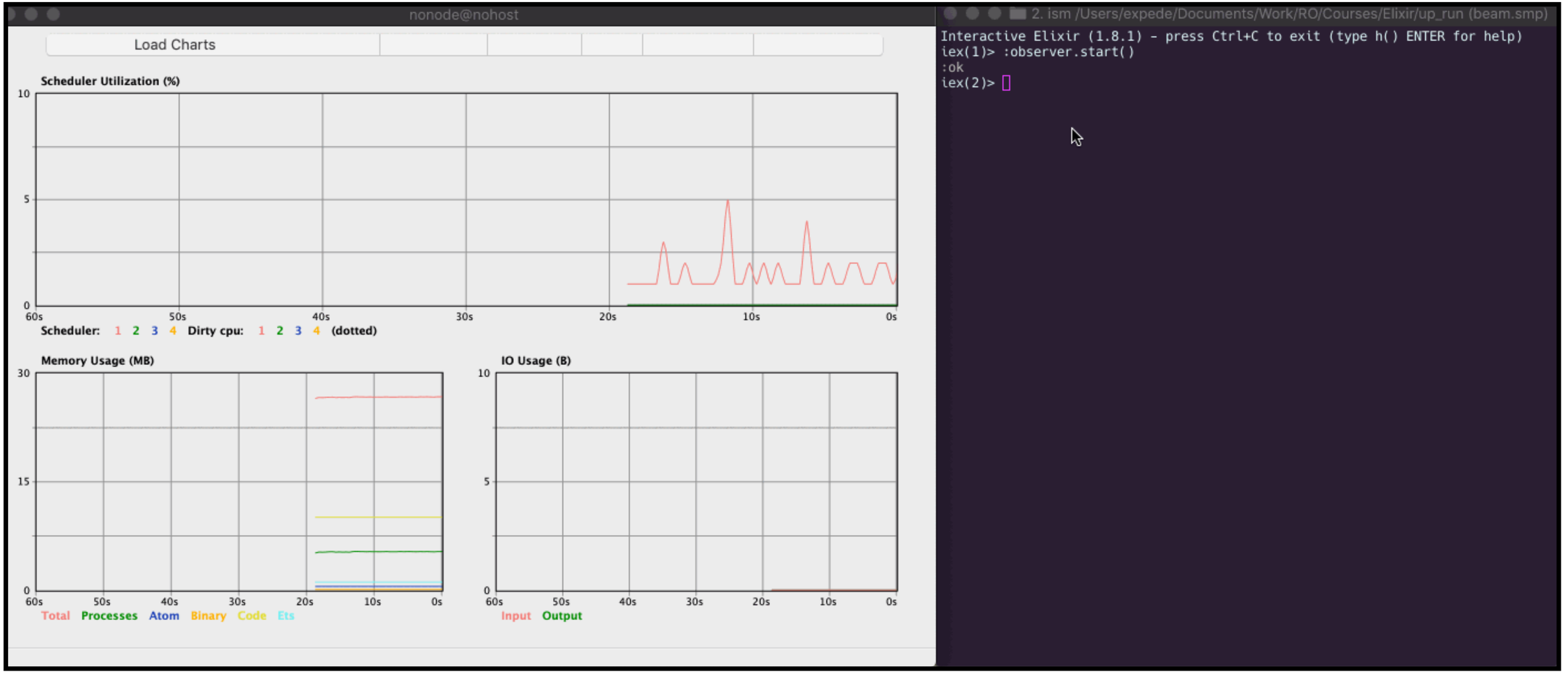
CONCURRENCY

"WAR & PEACE" WORD FREQUENCY



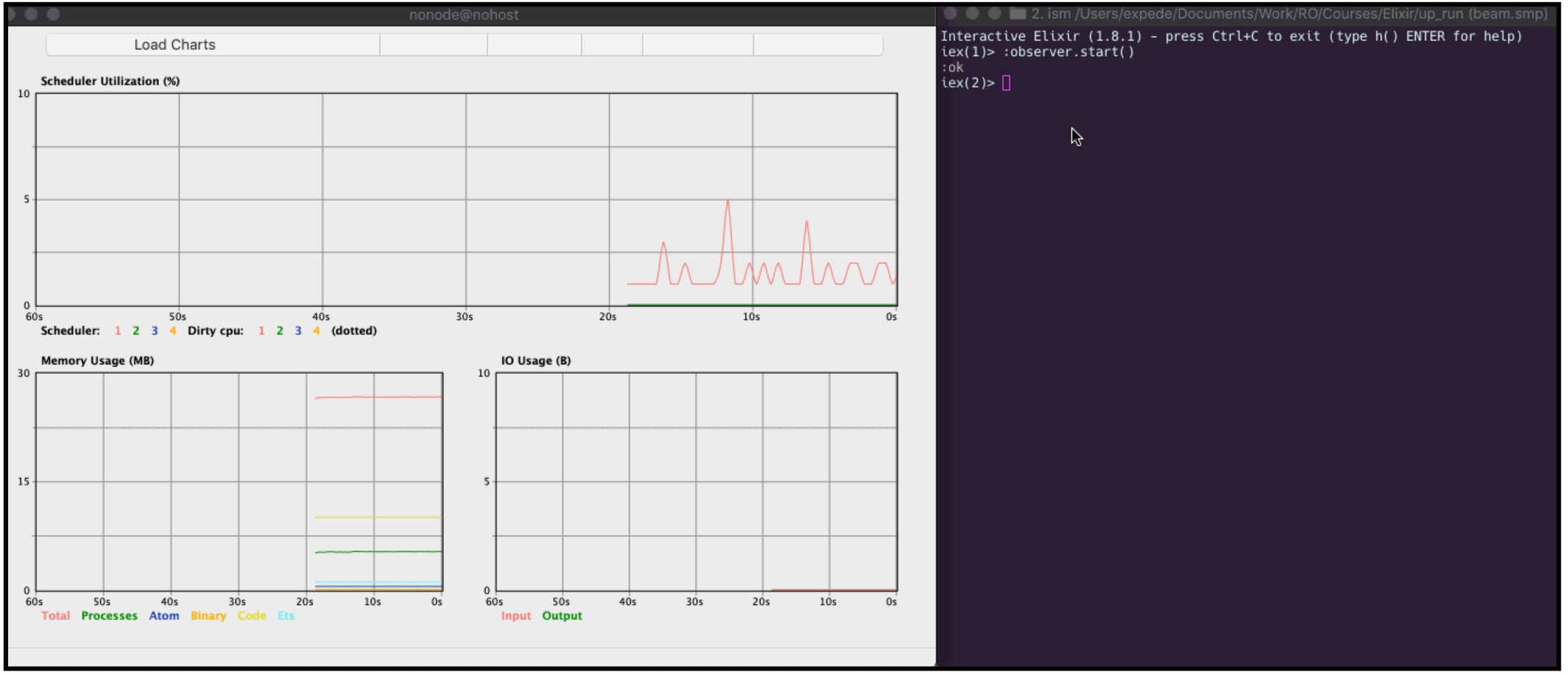
CONCURRENCY

100,000+ CONCURRENT TASKS (ON A LAPTOP)



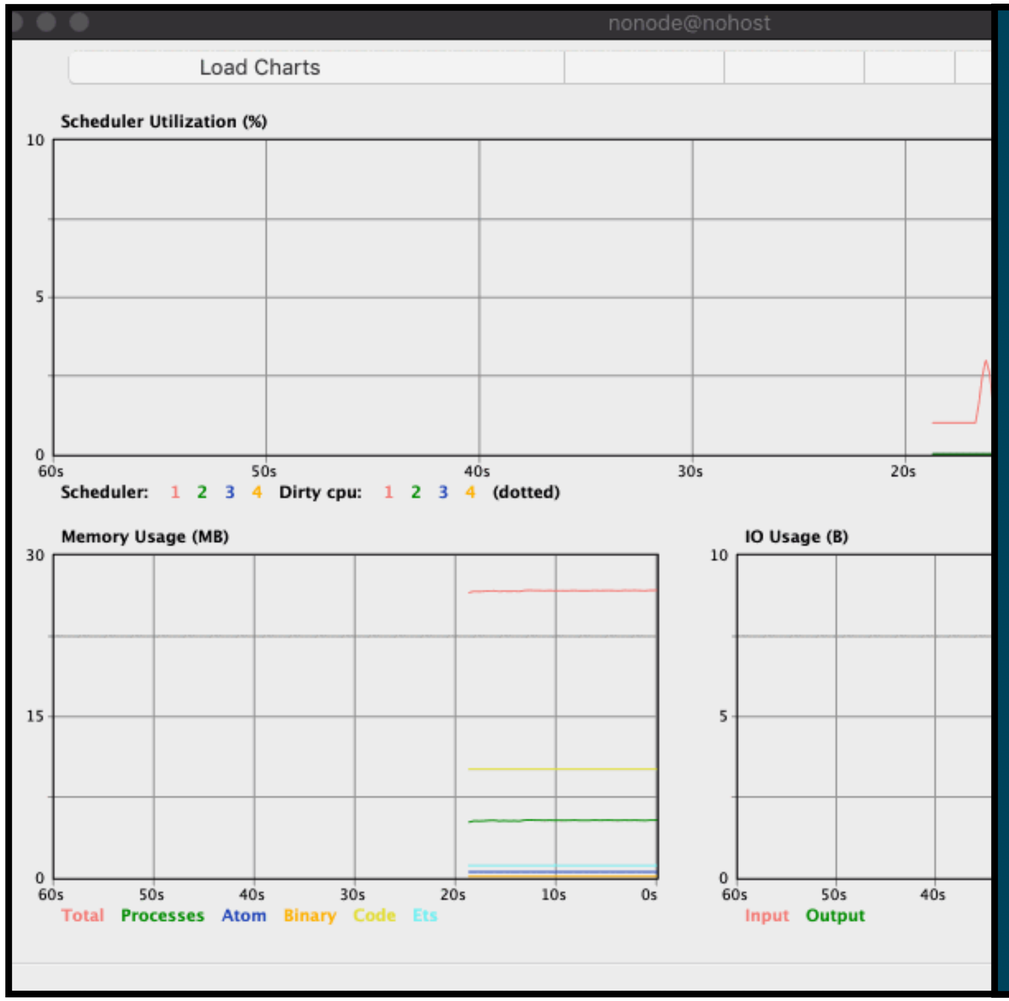
CONCURRENCY

100,000+ CONCURRENT TASKS (ON A LAPTOP)



CONCURRENCY

100,000+ CONCURRENT TASKS (ON A LAPTOP)




```elixir
Enum.each(0..99, fn n →
  spawn(fn →
    Enum.each(0..999, fn m →
      spawn(fn →
        name = "⚙ #{(n * 100) + m}"
        IO.puts "   Start  #{name}"

        time =
          [10, 500, 1000, 2000, 5000]
          ▷ Enum.random()
          ▷ :rand.uniform()

        Process.sleep(time)

        IO.puts "   Finish #{name} in    #{time}ms"
      end)
    end)
  end)
end)
```

THE ACTOR MODEL

THE ACTOR MODEL

A SANE MODEL OF CONCURRENCY & A CLEANER O.O.

THE ACTOR MODEL

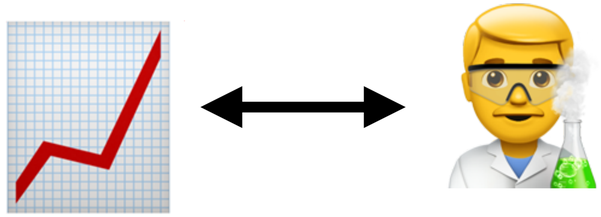
STORY TIME 📖

THE ACTOR MODEL

STORY TIME 📖

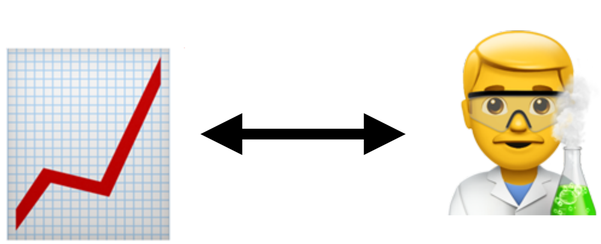
THE ACTOR MODEL

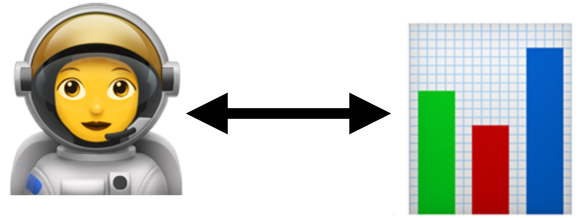
STORY TIME 📖



THE ACTOR MODEL

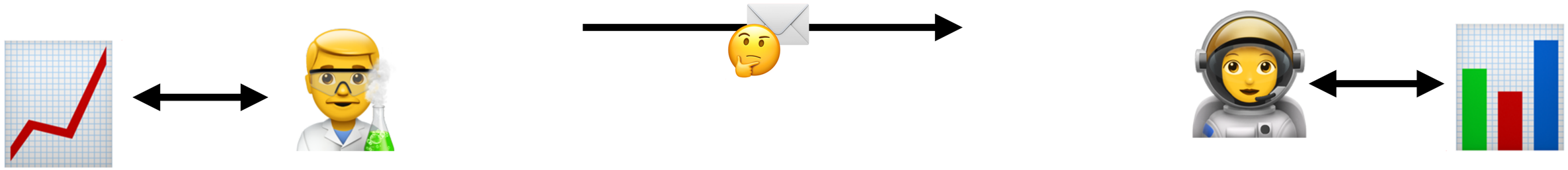
STORY TIME 📖





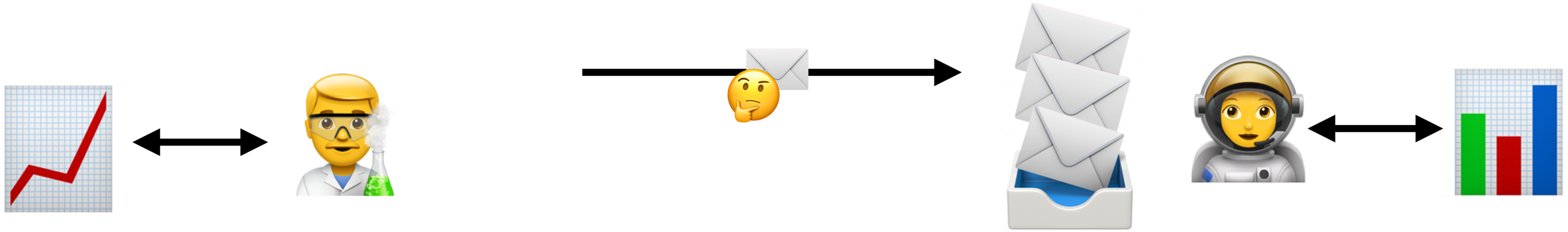
THE ACTOR MODEL

STORY TIME 📖



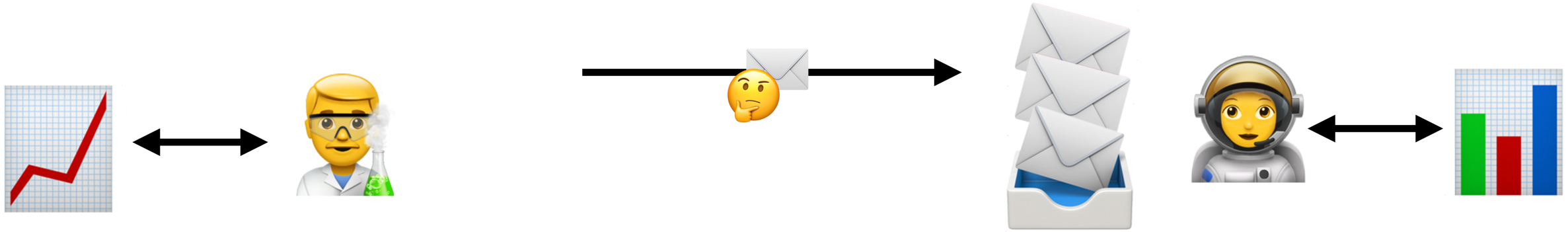
THE ACTOR MODEL

STORY TIME 📖



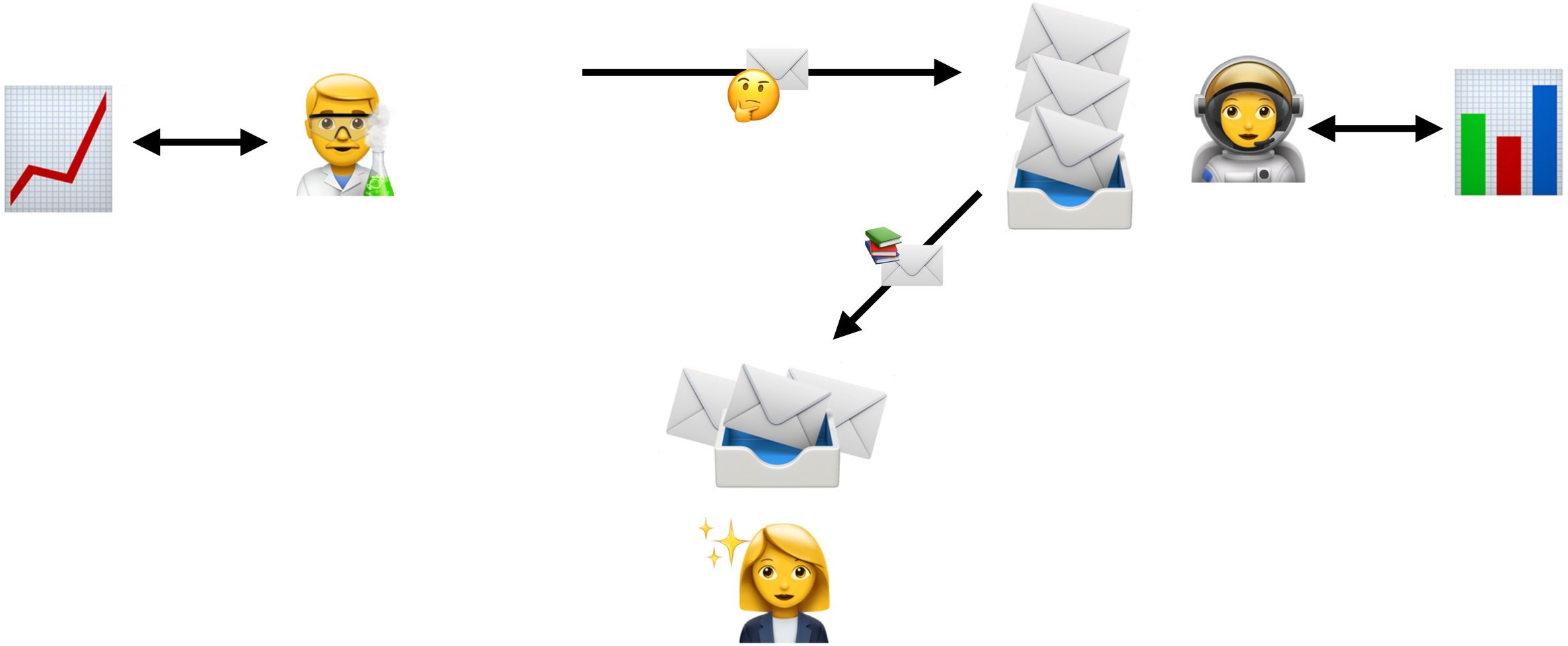
THE ACTOR MODEL

STORY TIME 📖



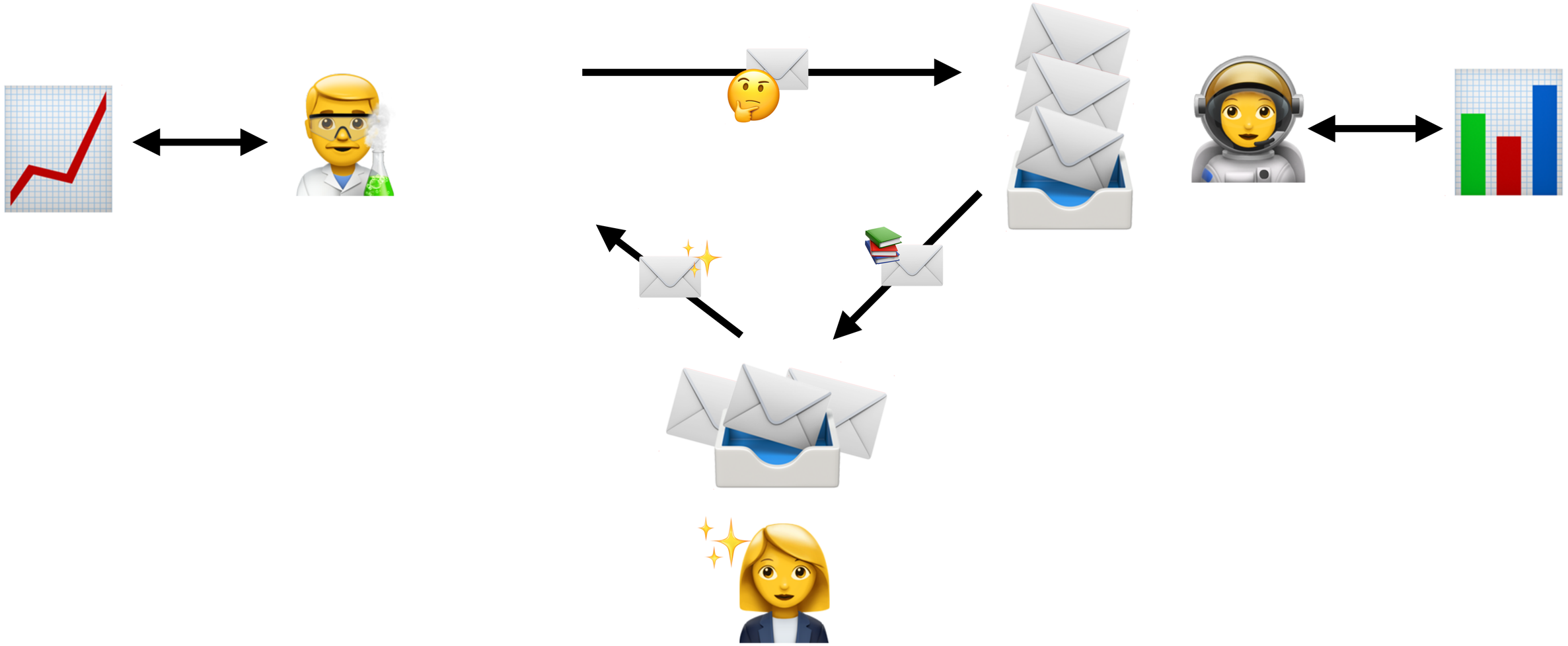
THE ACTOR MODEL

STORY TIME 📖



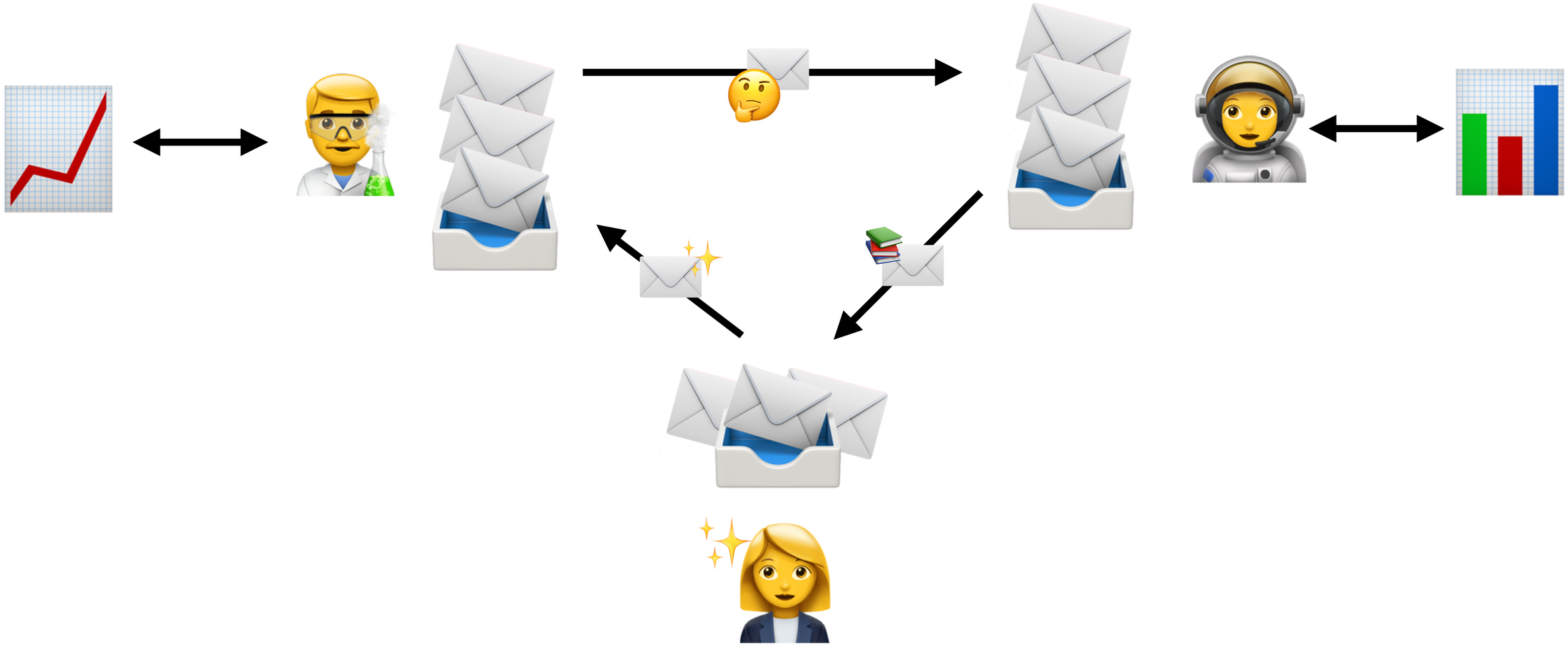
THE ACTOR MODEL

STORY TIME 📖



THE ACTOR MODEL

STORY TIME 📖

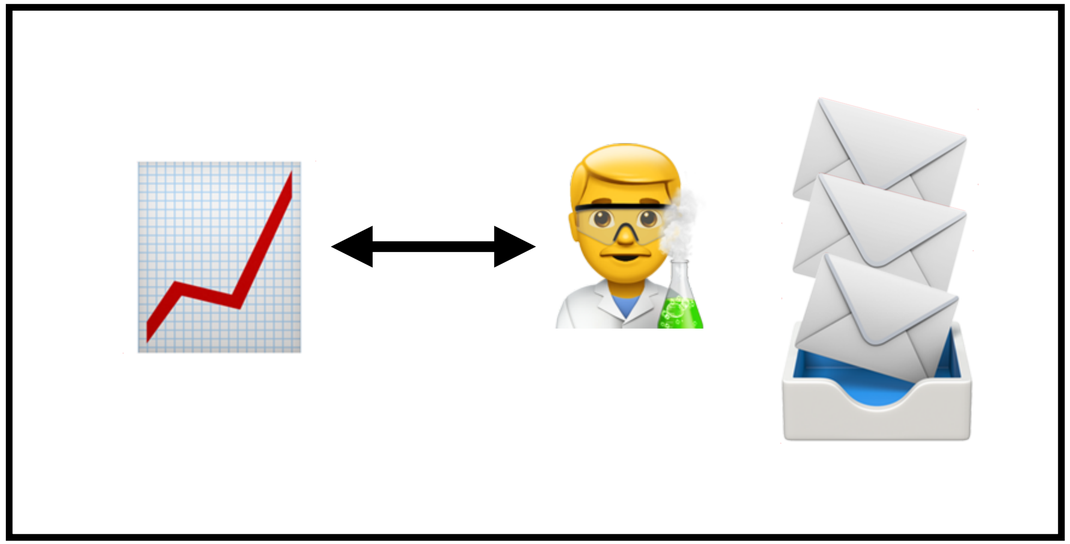


THE ACTOR MODEL

ABILITIES

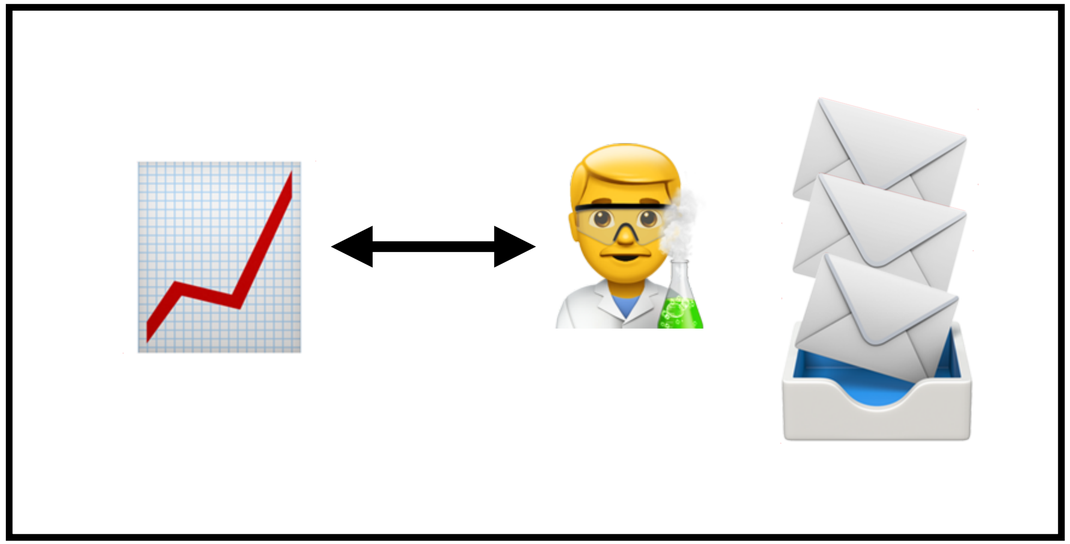
THE ACTOR MODEL

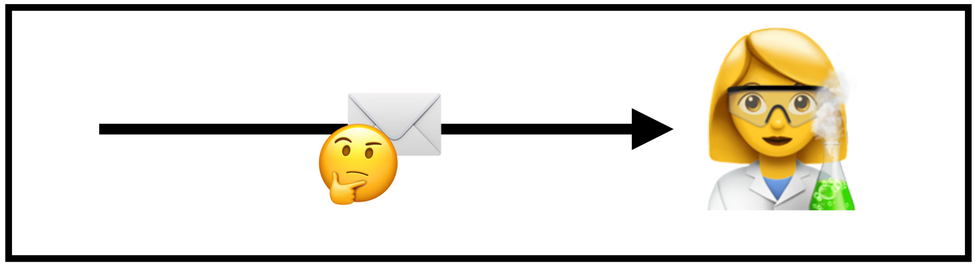
ABILITIES

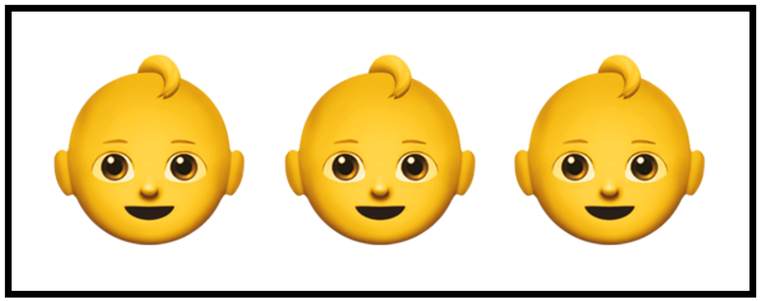


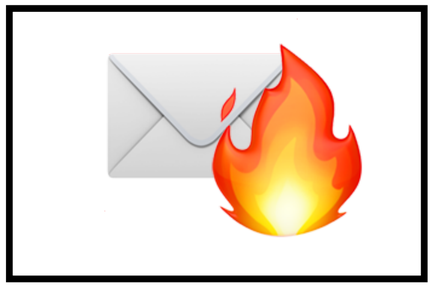
THE ACTOR MODEL

ABILITIES









THE ACTOR MODEL

MAILBOXES

THE ACTOR MODEL

MAILBOXES

- Literally what it sounds like
- Elixir is async, so message passing needs a message queue buffer
- Message queue
- Optional timeout

```
~/Desktop iex
Erlang/OTP 19 [erts-8.3] [source] [64-bit] [smp:4:4] [async-threads:10] [hipe] [kernel-poll:false] [dtrace]

Interactive Elixir (1.4.2) - press Ctrl+C to exit (type h() ENTER for help)
iex(1)> receive do
...(1)>   {:cool, x} -> x
...(1)>   {:nifty, %{y: y}} when y > 2 -> y
...(1)> after
...(1)>    5_000 -> "Timed out"
...(1)>
...(1)> end
```

SUPERVISION

SUPERVISION

WHEN GOOD CODE GOES BAD 😈

SUPERVISION

LET IT CRASH 🤯

SUPERVISION

LET IT CRASH 🤯

- Don’t defensively program for extreme edge cases 😊
- Does not mean that you can sidestep error handling completely 😡
- Can ignore unknown unknowns
  - High load, concurrent systems
    - 1/1,000,000 bug may happen every few seconds

SUPERVISION

DECLARATIVE CONFIGURATION

SUPERVISION

DECLARATIVE CONFIGURATION

```elixir
defmodule Demo.SimpleSupervisor do
  use Supervisor

  def start_link, do: Supervisor.start_link(__MODULE__, [])

  def init(_) do
    children = [
      worker(Demo.SimpleServer, [])
    ]

    supervise(children, strategy: :one_for_one)
  end
end
```

SUPERVISION

DECLARATIVE CONFIGURATION

```elixir
defmodule Demo.SimpleSupervisor do
  use Supervisor

  def start_link, do: Supervisor.start_link(__MODULE__, [])

  def init(_) do
    children = [
      worker(Demo.SimpleServer, [])
    ]

    supervise(children, strategy: :one_for_one)
  end
end
```

SUPERVISION

RESTART STRATEGIES ♻

SUPERVISION

RESTART STRATEGIES ♻

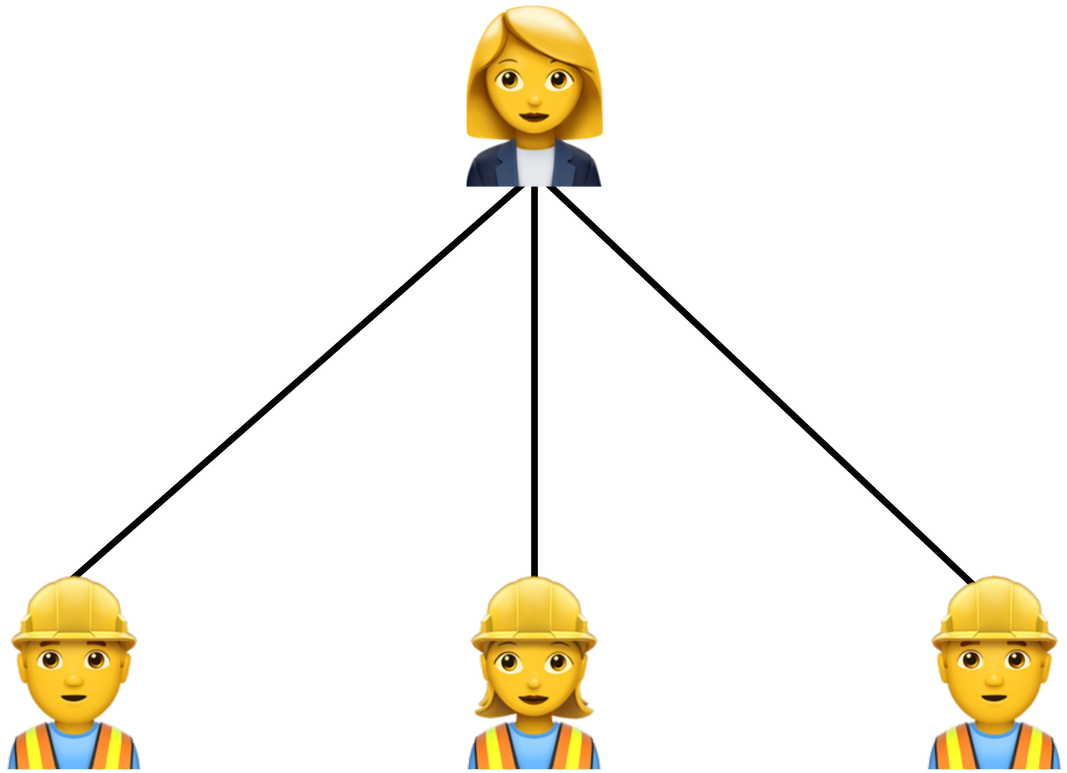
SUPERVISION

RESTART STRATEGIES ♻️

- :one_for_one
- :simple_one_for_one
- :one_for_all
- :rest_for_one

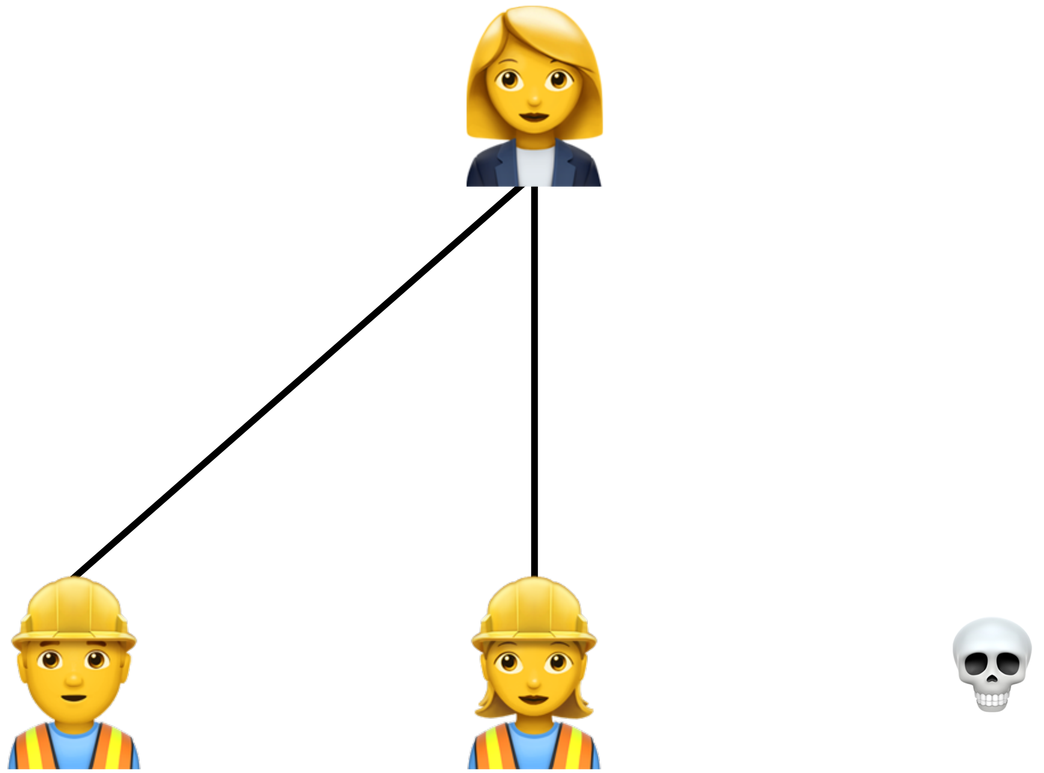
SUPERVISION

:ONE_FOR_ONE



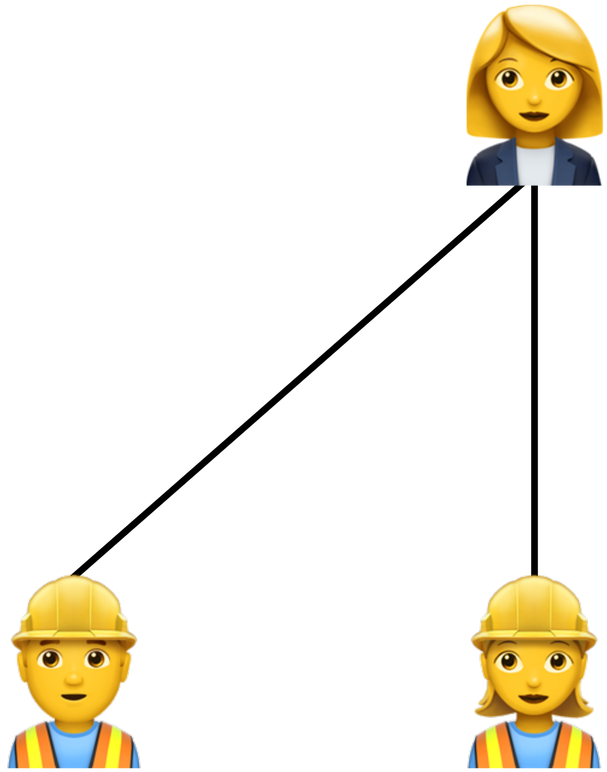
SUPERVISION

:ONE_FOR_ONE



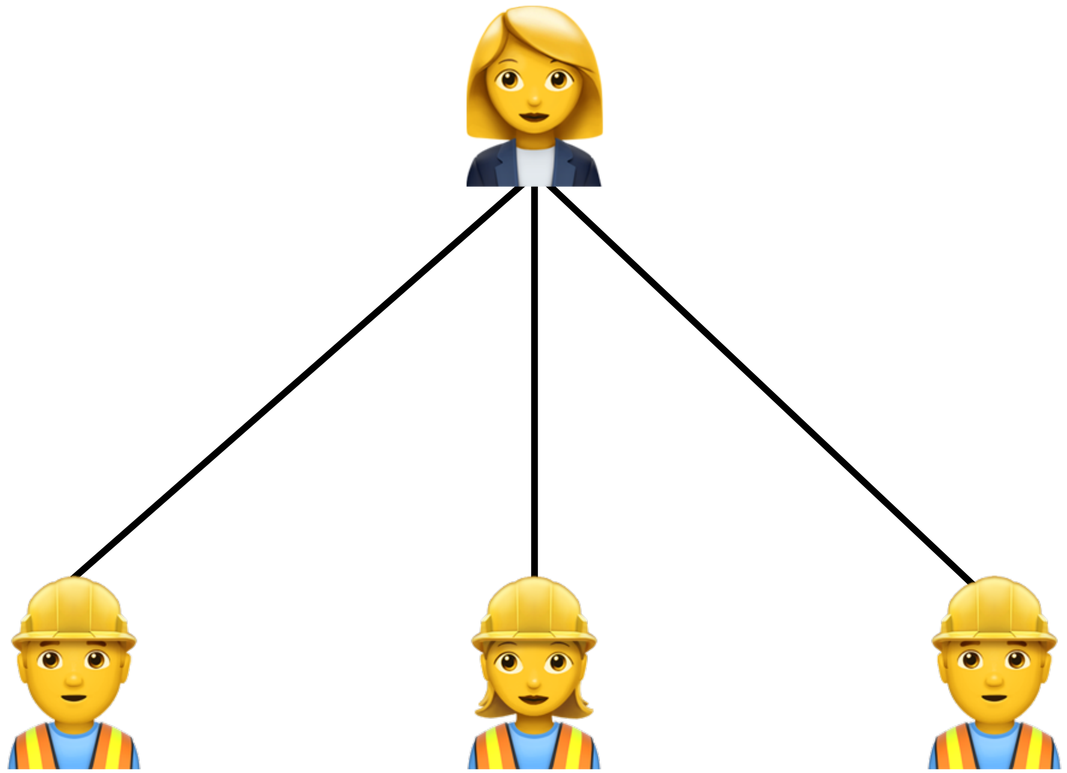
SUPERVISION

:ONE_FOR_ONE



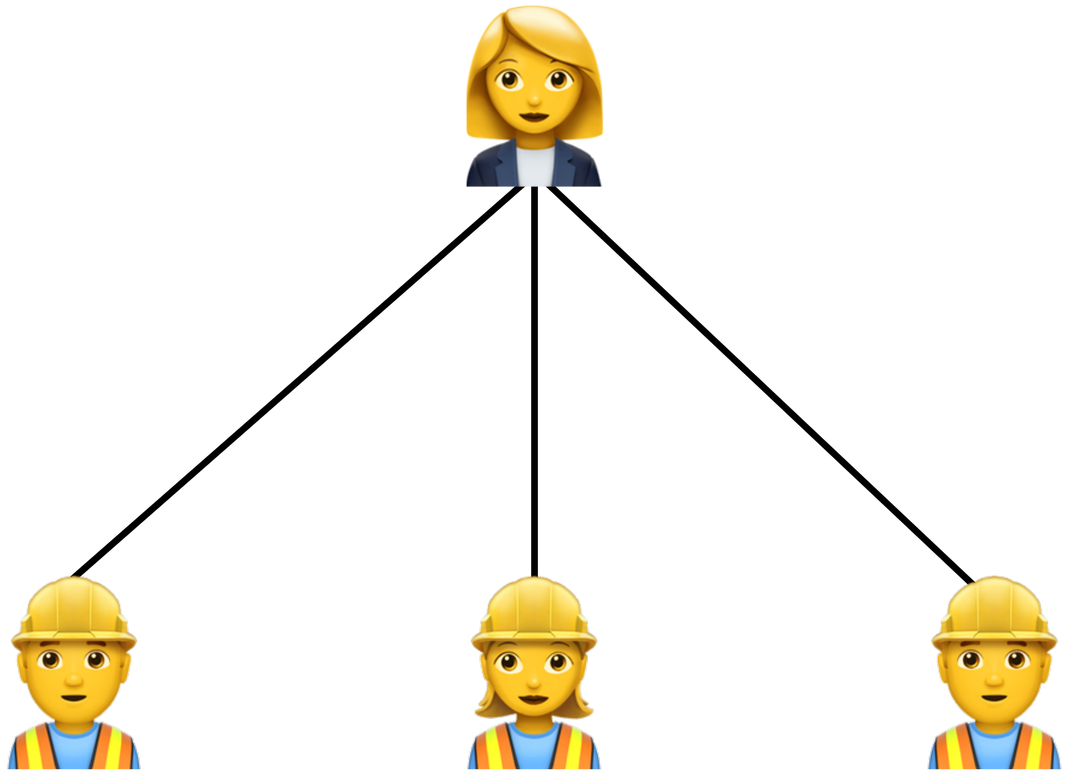
SUPERVISION

:ONE_FOR_ONE



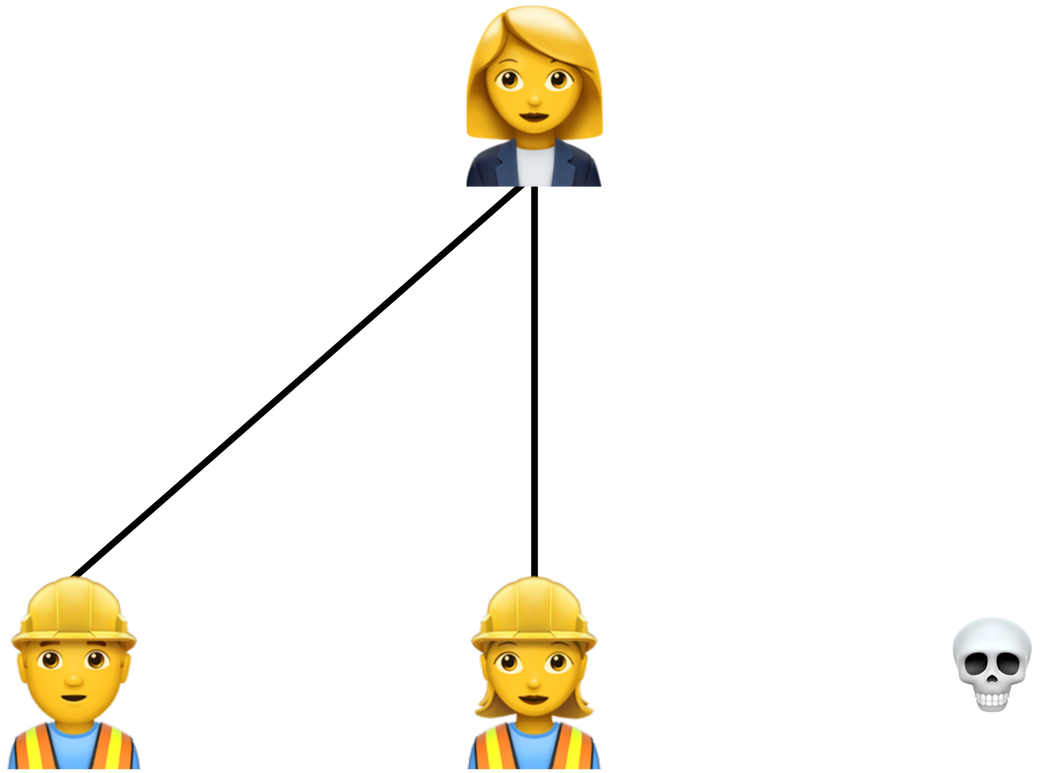
SUPERVISION

:ONE_FOR_ALL



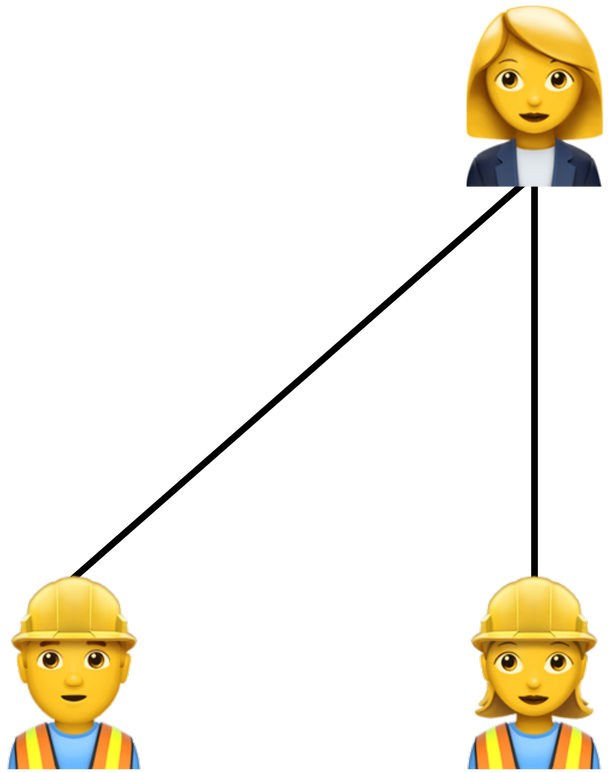
SUPERVISION

:ONE_FOR_ALL



SUPERVISION

:ONE_FOR_ALL

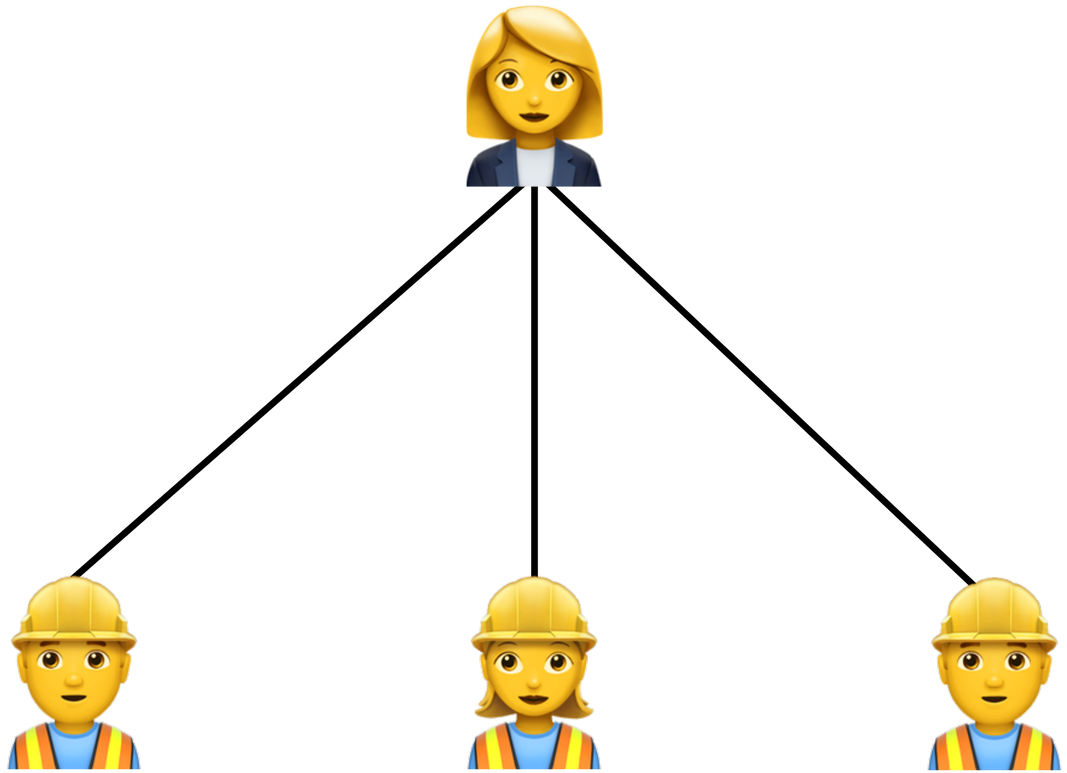


SUPERVISION

:ONE_FOR_ALL

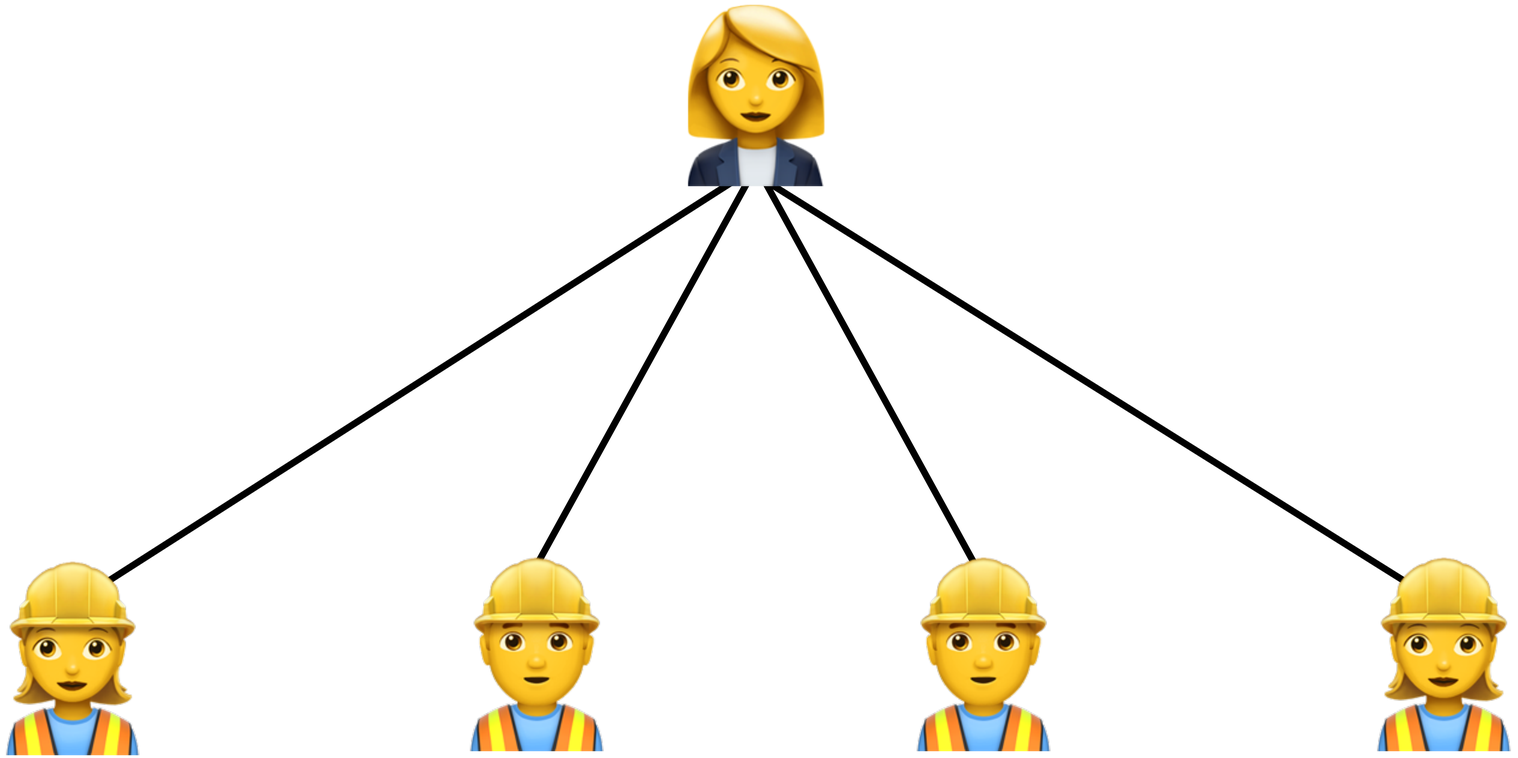
SUPERVISION

:ONE_FOR_ALL



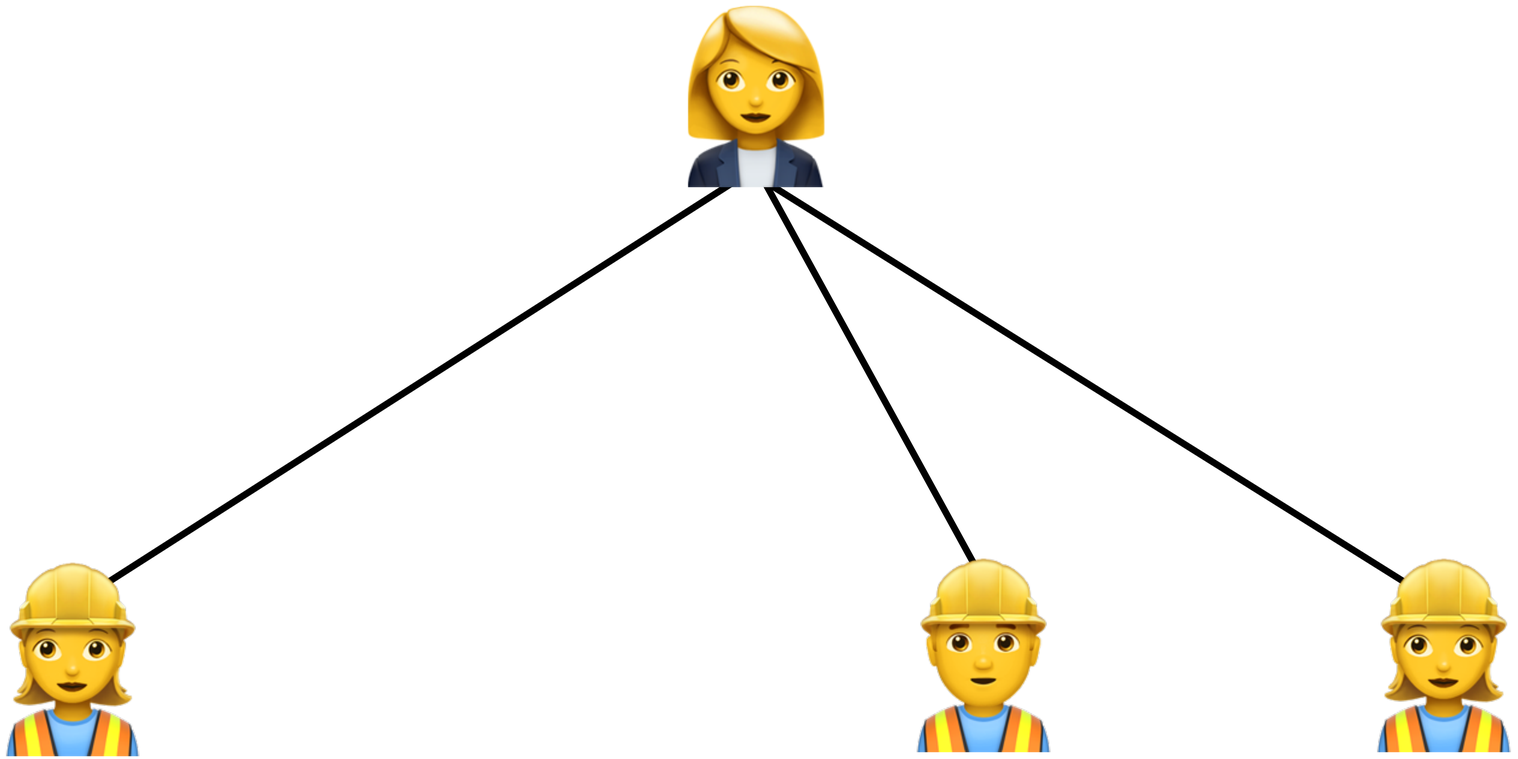
SUPERVISION

:REST_FOR_ONE



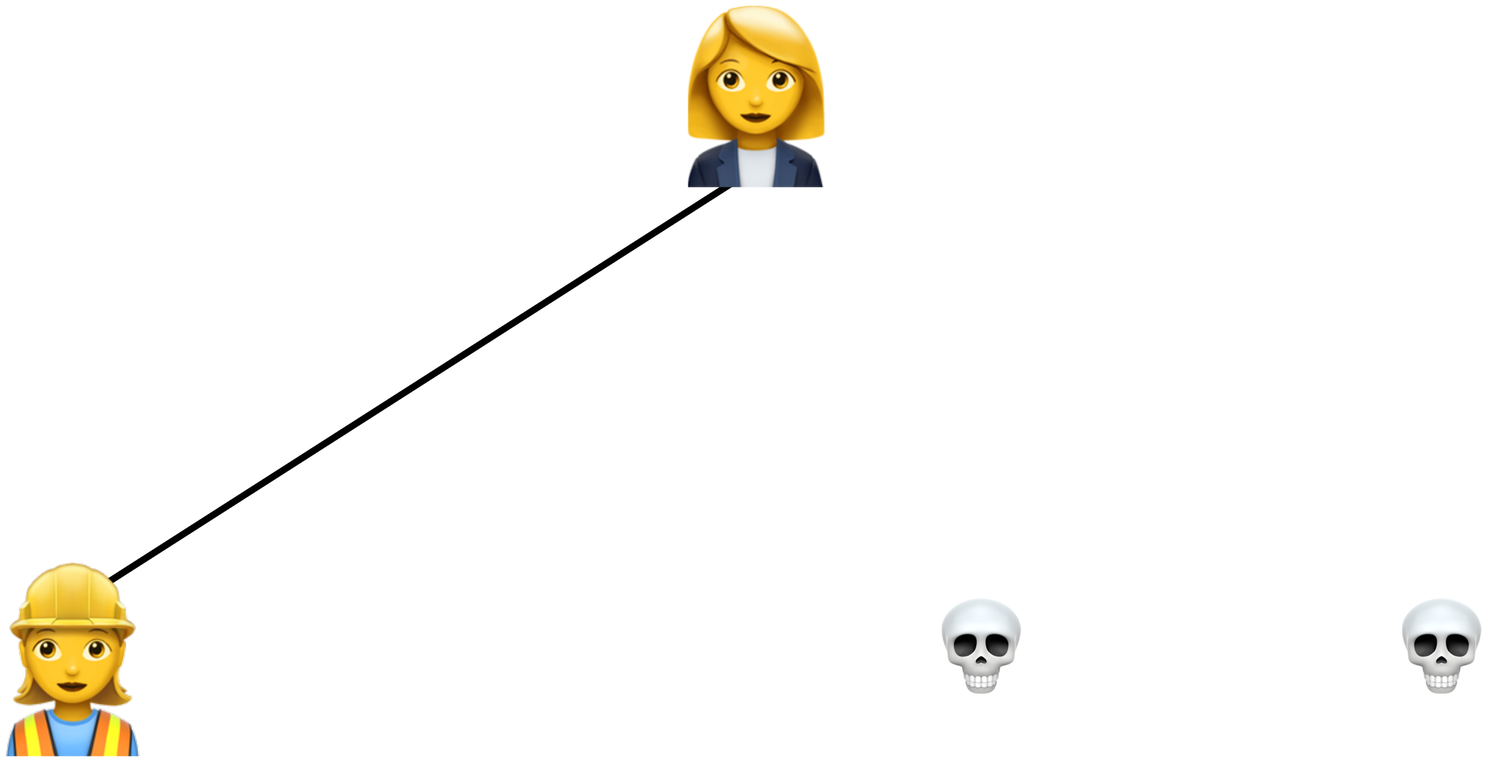
SUPERVISION

:REST_FOR_ONE



SUPERVISION

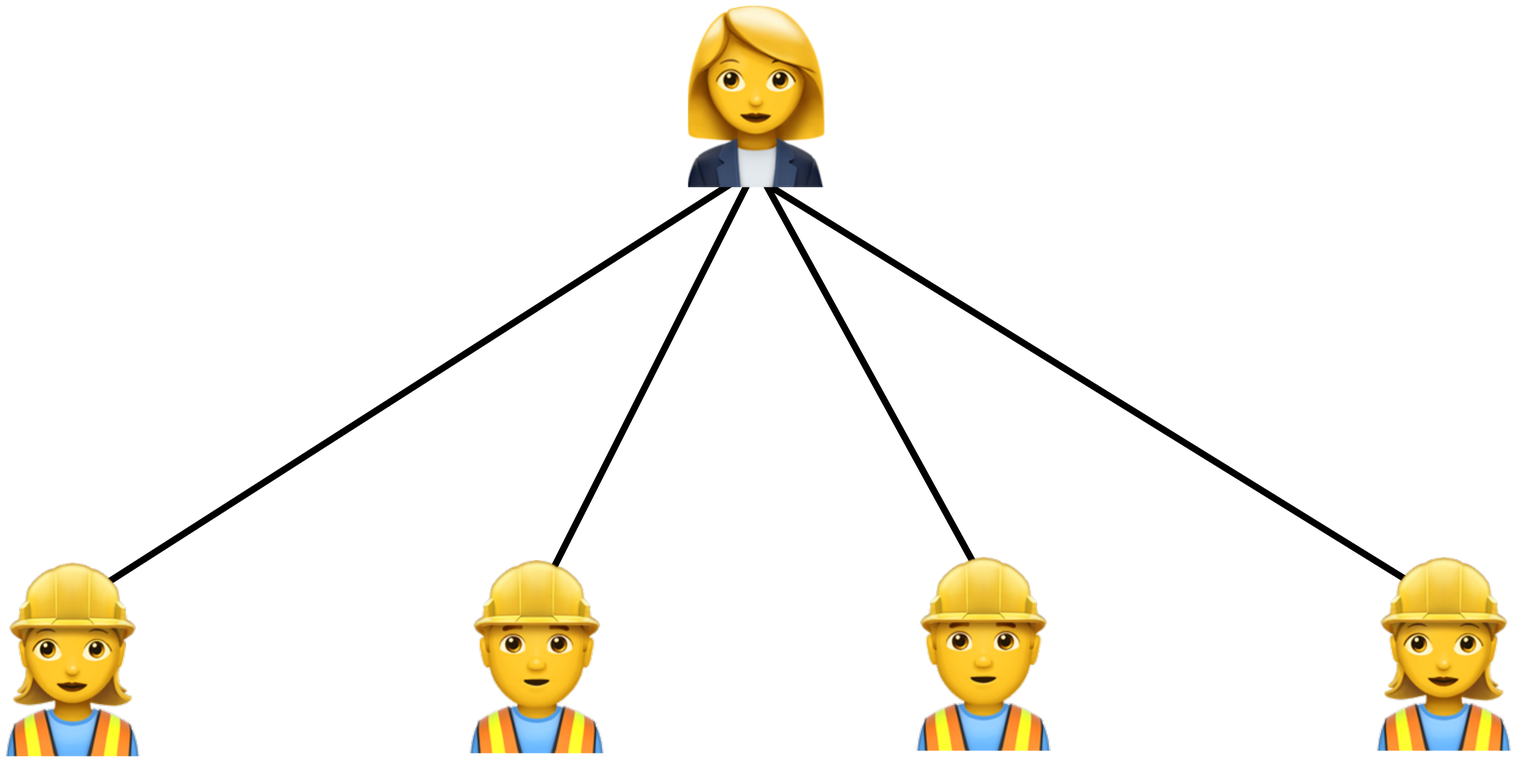
:REST_FOR_ONE



1 2 3 4

SUPERVISION

:REST_FOR_ONE



PHOENIX  

PHOENIX  

A HIGH UPTIME, BATTERIES-INCLUDED WEB FRAMEWORK

PHOENIX

STRENGTHS

PHOENIX

STRENGTHS

- All of the OTP goodness (concurrent, fault tolerant, &c)
- Generally no need for external queuing systems like Sidekiq or RabbitMQ
- Convenient, Rails-y APIs
- Familiar layout & style
- Clean(er) MVC architecture
- “A bunch of libraries”, rather than a monolithic system

PHOENIX

THE ~~RAILS~~ PHOENIX WAY

PHOENIX

THE ~~RAILS~~ PHOENIX WAY

- Opinionated!
- [M]VC+
- Explicit > automagic
  - …for the parts that you touch
- Safe defaults
- Consistency
- Focus on
  - Developer experience & ease of use
  - Sockets
  - Performance

PHOENIX

EXAMPLE REQUEST

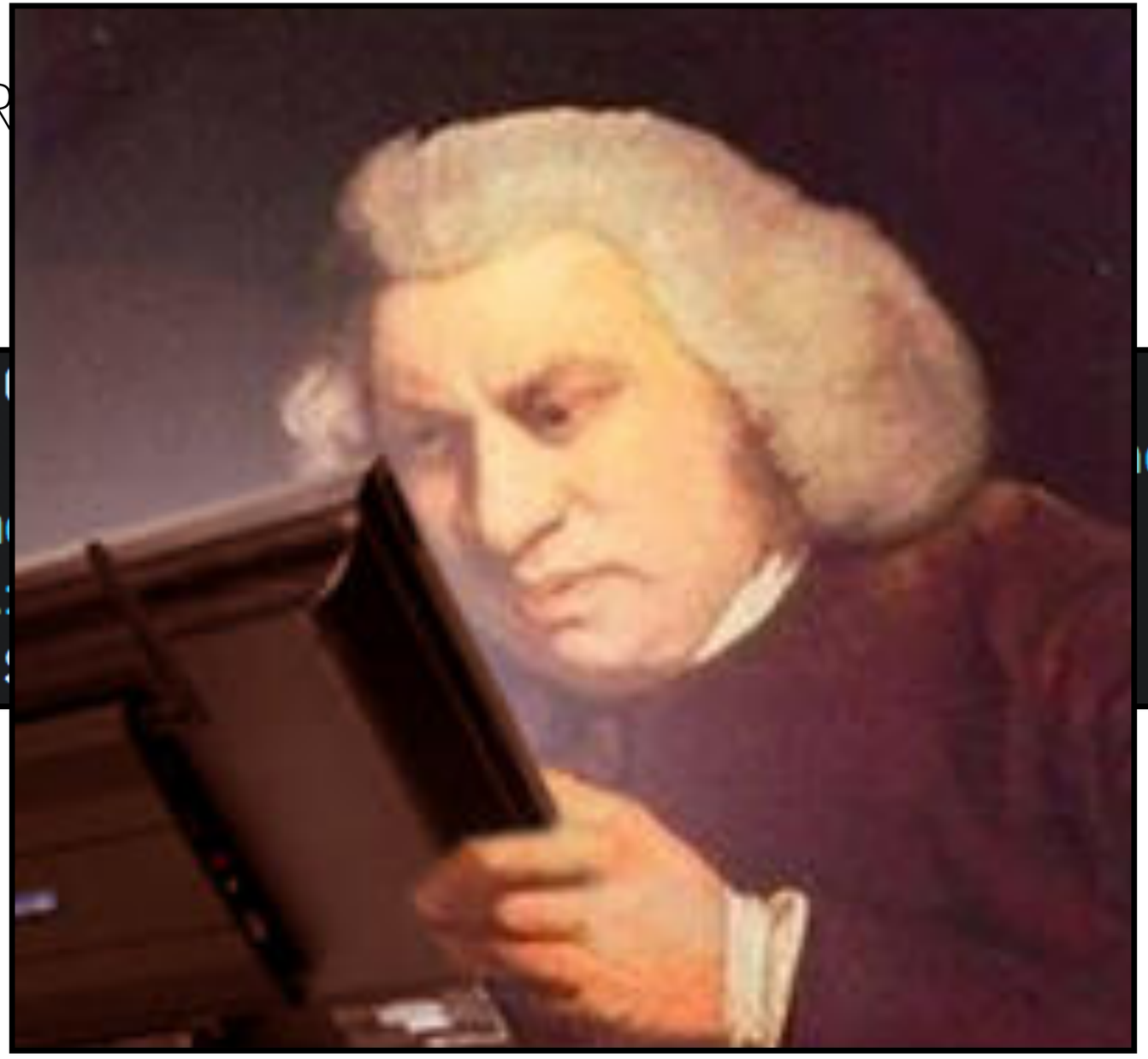
PHOENIX

EXAMPLE REQUEST

```
[info] GET /
[debug] Processing by Todo.PageController.index/2
  Parameters: %{}
  Pipelines: [:browser]
[info] Sent 200 in 244µs
```

PHOENIX

EXAMPLE R



```
[info] (
[debug]                                    dex/2
  Param
  Pipel
[info] S
```

PHOENIX

EXAMPLE R

[info] (

[info] S

[info] Sent 200 in 244µs

PHOENIX

EXAMPLE R

PHOENIX

FAMILIAR STRUCTURE

PHOENIX

FAMILIAR STRUCTURE

```
lib/
├── todo
│   ├── accounts
│   │   └── user.ex
│   ├── application.ex
│   ├── factory.ex
│   ├── lists
│   │   ├── item.ex
│   │   └── list.ex
│   └── repo.ex
├── todo.ex
├── todo_web
│   ├── channels
│   │   ├── event_channel.ex
│   │   └── event_socket.ex
│   ├── controllers
│   │   ├── api
│   │   │   ├── item_controller.ex
│   │   │   ├── list_controller.ex
│   │   │   └── user_controller.ex
│   │   ├── item_controller.ex
│   │   ├── list_controller.ex
│   │   ├── page_controller.ex
│   │   └── user_controller.ex
│   ├── endpoint.ex
│   ├── gettext.ex
│   ├── router.ex
│   ├── templates
│   │   ├── item
│   │   │   ├── delete_button.html.eex
│   │   │   ├── edit.html.eex
│   │   │   ├── image.html.eex
│   │   │   ├── index.html.eex
│   │   │   ├── items.html.eex
│   │   │   ├── new.html.eex
│   │   │   └── show.html.eex
│   │   ├── layout
│   │   │   ├── app.html.eex
│   │   │   └── app.json.eex
│   │   ├── list
│   │   │   ├── delete_button.html.eex
│   │   │   ├── edit.html.eex
│   │   │   ├── index.html.eex
│   │   │   ├── lists.html.eex
│   │   │   ├── members.html.eex
│   │   │   ├── new.html.eex
│   │   │   └── show.html.eex
│   │   ├── page
│   │   │   ├── index.html.eex
│   │   │   └── index.json.eex
│   │   └── user
│   │       ├── avatar.html.eex
│   │       ├── completed_items.html.eex
│   │       └── delete_button.html.eex
│   └── views
│       ├── api
│       │   ├── item_view.ex
│       │   ├── list_view.ex
│       │   └── user_view.ex
│       ├── error_helpers.ex
│       ├── error_view.ex
│       ├── item_view.ex
│       ├── layout_view.ex
└── todo_web.ex
```

PHOENIX

FAMILIAR STRUCTURE

```
lib/
├── todo
│   ├── accounts
│   │   └── user.ex
│   ├── application.ex
│   ├── factory.ex
│   ├── lists
│   │   ├── item.ex
│   │   └── list.ex
│   └── repo.ex
├── todo.ex
└── todo_web
    ├── channels
    │   ├── event_channel.ex
    │   └── event_socket.ex
    ├── controllers
    │   ├── api
    │   │   ├── item_controller.ex
    │   │   ├── list_controller.ex
    │   │   └── user_controller.ex
    │   ├── item_controller.ex
    │   ├── list_controller.ex
    │   ├── page_controller.ex
    │   └── user_controller.ex
    ├── endpoint.ex
    ├── gettext.ex
    ├── router.ex
    ├── templates
    │   ├── item
    │   │   ├── delete_button.html.eex
    │   │   ├── edit.html.eex
    │   │   ├── image.html.eex
    │   │   ├── index.html.eex
    │   │   ├── items.html.eex
    │   │   ├── new.html.eex
    │   │   └── show.html.eex
    │   ├── layout
    │   │   ├── app.html.eex
    │   │   └── app.json.eex
    │   ├── list
    │   │   ├── delete_button.html.eex
    │   │   ├── edit.html.eex
    │   │   ├── index.html.eex
    │   │   ├── lists.html.eex
    │   │   ├── members.html.eex
    │   │   ├── new.html.eex
    │   │   └── show.html.eex
    │   ├── page
    │   │   ├── index.html.eex
    │   │   └── index.json.eex
    │   └── user
    │       ├── avatar.html.eex
    │       ├── completed_items.html.eex
    │       └── delete_button.html.eex
    └── views
        ├── api
        │   ├── item_view.ex
        │   ├── list_view.ex
        │   └── user_view.ex
        ├── error_helpers.ex
        ├── error_view.ex
        ├── item_view.ex
        └── layout_view.ex
└── todo_web.ex
```

PHOENIX

EXAMPLE CONTROLLER

PHOENIX

EXAMPLE CONTROLLER

```elixir
defmodule TodoWeb.API.UserController do
  alias Todo.Accounts.User
  use TodoWeb, :controller

  @spec index(Plug.Conn.t(), map()) :: Plug.Conn.t()
  def index(conn, _params), do: render(conn, "index.json", users: Repo.all(User))

  @spec show(Plug.Conn.t(), map()) :: Plug.Conn.t()
  def show(conn, %{"id" => id}) do
    user =
      User
      |> Repo.get!(id)
      |> Repo.preload([:lists, :completed_items])

    render(conn, "show.json", user: user)
  end

  @spec create(Plug.Conn.t(), map()) :: Plug.Conn.t()
  def create(conn, %{"data" => params}) do
    user =
      User
      |> User.changeset(params)
      |> Repo.insert!()
      |> Repo.preload([:lists, :completed_items])

    conn
    |> put_status(:created)
    |> render("show.json", user: user)
  end

  @spec update(Plug.Conn.t(), map()) :: Plug.Conn.t()
  def update(conn, %{"id" => id, "data" => changes}) do
    user =
      User
      |> Repo.get!(id)
      |> User.changeset(changes)
      |> Repo.update!()

    render(conn, "show.json", user: user)
  end

  @spec delete(Plug.Conn.t(), map()) :: Plug.Conn.t()
  def delete(conn, %{"id" => id}) do
    user =
      User
      |> Repo.get!(id)
      |> Repo.delete!()

    send_resp(conn, 204, "")
  end
end
```

FURTHER READING

FURTHER READING

WHERE TO GO FROM HERE

FURTHER READING

GETTING STARTED & INTEREST READING

FURTHER READING

GETTING STARTED & INTEREST READING

- IEx & Mix
- GenServer
- Broadway
- Phoenix LiveView

- https://elixir-lang.org
- https://hex.pm & https://hexdocs.pm
- http://blog.plataformatec.com.br/tag/elixir/
- https://github.com/expede/up_run
- https://github.com/expede/todo-example
- https://github.com/expede/quick_chat
- Twitter's #MyElixirStatus

THANK YOU, BUFFER! ✨

hello@brooklynzelenka.com
github.com/expede
@expede