Processes

Recreate a **simple** GenServer
implementation

Why?

Because:

- Everyone was talking about it
- I didn't get it correctly
- It seemed like the building block of OTP

Who?

People who

- Don't understand Elixir Processes
- Don't understand GenServer behaviour

Processes

DISCLAIMER: From now on, we won't talk about OS but Erlang processes

- Lightweight thread of Erlang VM (1 or 2 KB)
- Run in concurrence of other Processes
- Use all the available CPU resources
- Isolated from each other
  - crash wise
  - data wise
- Can have an internal state
- Can receive messages only

Processes

Starting from scratch

```
spawn(fn ->
  Process.sleep(2000)
  IO.puts(2 * 2)
end)

# Execution:
# ...wait 2 seconds...
# 4
# PID<0.84.0>
```

Cool, but useless. Would be awesome to take some parameters

Processes

Starting from scratch

```
background_job = fn(value) ->
  spawn(fn ->
    Process.sleep(2_000)
    IO.puts(value * 2)
  end)
end

# Execution:
# > background_job.(4)
# ... Wait 2 seconds...
# 8
# PID<0.87.0>
```

Better. We can trigger tasks in background.

What about communication?

Processes

Exchanging messages

```
background_job = fn(value, caller) ->
  spawn(fn ->
    Process.sleep(2_000)
    send(caller, {:done, value * 2})
  end)
end

# Execution:
# > background_job.(4, self())
# PID<0.87.0>
# > receive do
# >   {:done, result} -> IO.puts("Result is #{result}")
# > end
# Result is 8
# :ok
```

Yay! we can now send and receive messages
but what about not dying after each call?

Processes

```
defmodule Multiplicator do
  def start(initial_value) do
    spawn(fn -> loop(initial_value) end)
  end

  defp loop(result) do
    receive do
      {multiplicator, caller} ->
        result = result * multiplicator
        send(caller, {:done, result})
        loop(result)
    end
  end
end

# Execution:
# > pid = Multiplicator.start(1)
# PID<0.87.0>
# > send(pid, {2, self()})
# > receive do
# >   {:done, result} -> IO.puts("Result is #{result}")
# > end
# Result is 2
# > send(pid, {21, self()})
# > receive ....
# Result is 42
```

Now, we are ready!

Let's move to an editor

`https://github.com/kdisneur/processes`

Step 2

```
git checkout 9f788ca
```

What we've seen previously in theory but now with a real use case

Pros

- It works
- Concurrent
- Elixir "by the book"

Cons

- Verbose
- Mix Interface with Implementation details

Step 3

`git checkout 0e32b1f`

We partially fixed the cons from **Step 2**

Pros

- It works
- Concurrent
- Elixir "by the book"
- Clean separation between interface and implementation

Cons

- Verbose

Step 5

```
git checkout a1d6436
```

Generic

- Spawn a new process
- Return a response tuple
- For asynchronous calls, send the message and return an ok atom
- For synchronous calls, send the message, wait for a response and return a value

Specific

- Internal state
- Messages

Step 6

```
git checkout 78e87e7
```

Well done! We just (re)created a basic
GenServer

In a perfect world, we could now add new features more easily (new servers, or news messages)

RECOMMENDATION