OTP

Section 6

In this Section, we are going to take a look at...

- Introduction to OTP
- The OTP behavior set
- Application

Introduction to OTP

In this Video, we are going to take a look at...

- What is OTP
- Why should we use OTP
- How is OTP exposed in Elixir

Where Does OTP Come From?

OTP

OTP

Open Telecom Platform

OTP – Open Telecom Platform

Framework

A structure for building **Applications** and having them operate with each-other

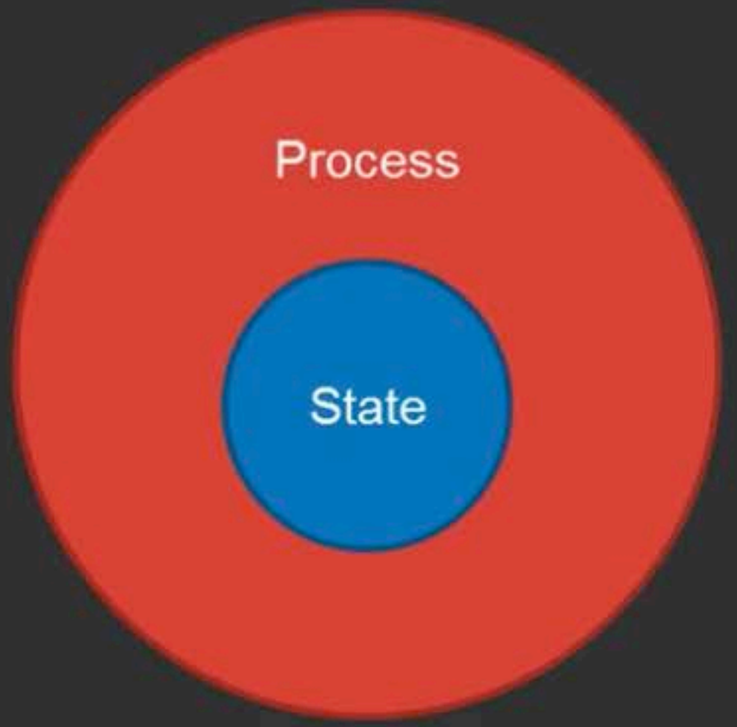
Design Patterns

A set of **Behaviors** that implement common generic usage patterns

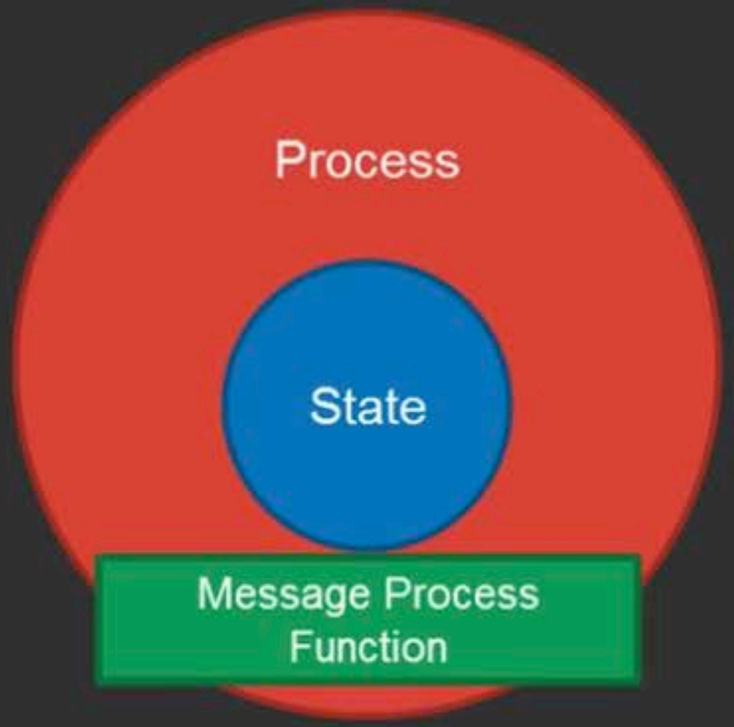
Why?

```elixir
defmodule Counter do
  def create() do
    spawn(__MODULE__, :loop, [0])
  end

  def inc(pid, x) do
    pid |> send({:inc, x})
  end

  def dec(pid, x) do
    pid |> send({:dec, x})
  end

  def get(pid) do
    pid |> send({:get, self()})
    receive do
      counter -> counter
    end
  end

  def loop(counter) do
    receive do
      {:inc, x} -> loop(counter + x)
      {:dec, x} -> loop(counter - x)
      {:get,caller} -> send(caller, counter)
                       loop(counter)
    end
  end
end
```

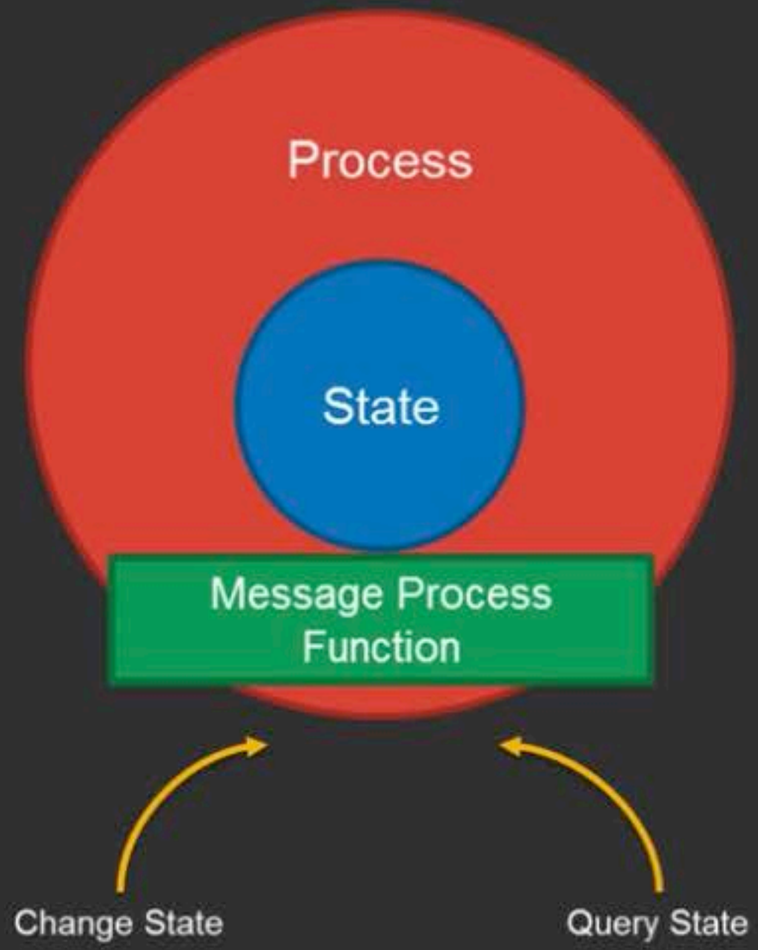
```elixir
defmodule Counter do
  def create() do
    spawn(__MODULE__, :loop, [0])
  end

  def inc(pid, x) do
    pid |> send({:inc, x})
  end

  def dec(pid, x) do
    pid |> send({:dec, x})
  end

  def get(pid) do
    pid |> send({:get, self()})
    receive do
      counter -> counter
    end
  end

  def loop(counter) do
    receive do
      {:inc, x} -> loop(counter + x)
      {:dec, x} -> loop(counter - x)
      {:get,caller} -> send(caller, counter)
                       loop(counter)
    end
  end
end
```



```elixir
defmodule Counter do
  def create() do
    spawn(__MODULE__, :loop, [0])
  end

  def inc(pid, x) do
    pid |> send({:inc, x})
  end

  def dec(pid, x) do
    pid |> send({:dec, x})
  end

  def get(pid) do
    pid |> send({:get, self()})
    receive do
      counter -> counter
    end
  end

  def loop(counter) do
    receive do
      {:inc, x} -> loop(counter + x)
      {:dec, x} -> loop(counter - x)
      {:get,caller} -> send(caller, counter)
                       loop(counter)
    end
  end
end
```

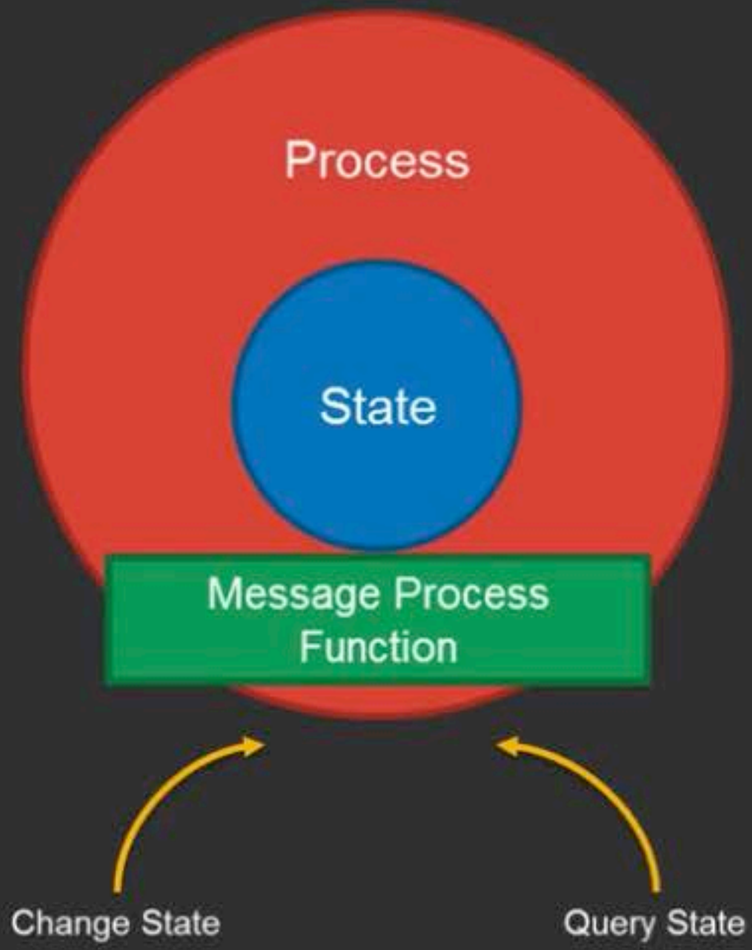


```elixir
defmodule Counter do
  def create() do
    spawn(__MODULE__, :loop, [0])
  end

  def inc(pid, x) do
    pid |> send({:inc, x})
  end

  def dec(pid, x) do
    pid |> send({:dec, x})
  end

  def get(pid) do
    pid |> send({:get, self()})
    receive do
      counter -> counter
    end
  end

  def loop(counter) do
    receive do
      {:inc, x} -> loop(counter + x)
      {:dec, x} -> loop(counter - x)
      {:get,caller} -> send(caller, counter)
                       loop(counter)
    end
  end
end
```



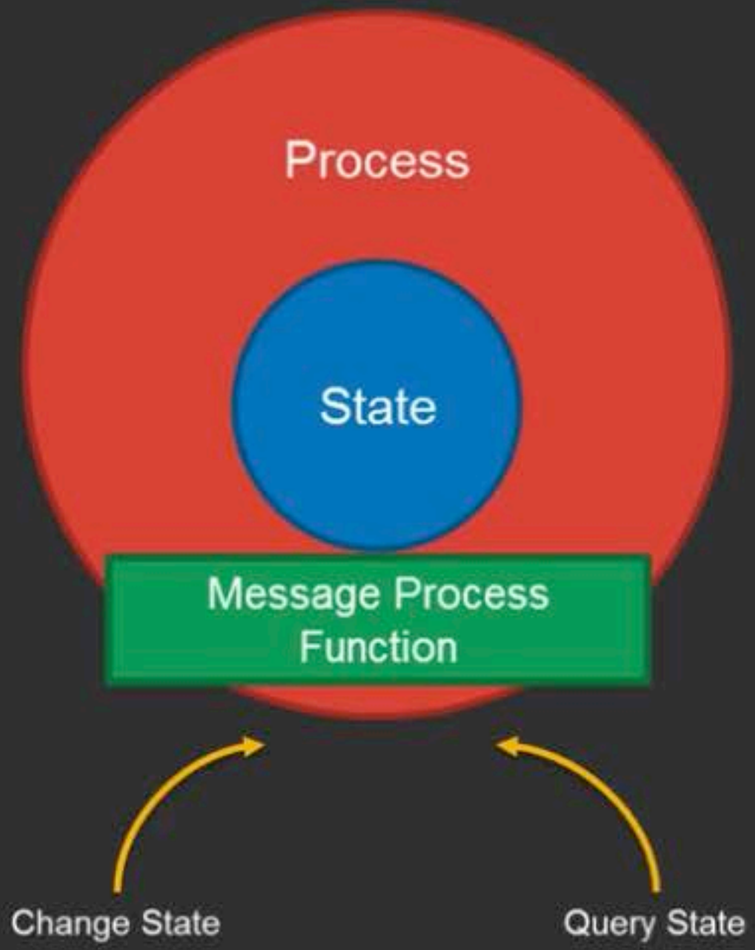
```elixir
defmodule Counter do
  def create() do
    spawn(__MODULE__, :loop, [0])
  end

  def inc(pid, x) do
    pid |> send({:inc, x})
  end

  def dec(pid, x) do
    pid |> send({:dec, x})
  end

  def get(pid) do
    pid |> send({:get, self()})
    receive do
      counter -> counter
    end
  end

  def loop(counter) do
    receive do
      {:inc, x} -> loop(counter + x)
      {:dec, x} -> loop(counter - x)
      {:get,caller} -> send(caller, counter)
                       loop(counter)
    end
  end
end
```

- Timeouts
- Unknown messages
- *Crashes*
- No receiver



```elixir
defmodule Counter do
  def create() do
    spawn(__MODULE__, :loop, [0])
  end

  def inc(pid, x) do
    pid |> send({:inc, x})
  end

  def dec(pid, x) do
    pid |> send({:dec, x})
  end

  def get(pid) do
    pid |> send({:get, self()})
    receive do
      counter -> counter
    end
  end

  def loop(counter) do
    receive do
      {:inc, x} -> loop(counter + x)
      {:dec, x} -> loop(counter - x)
      {:get,caller} -> send(caller, counter)
                       loop(counter)
    end
  end
end
```

GenServer



```elixir
defmodule Counter do
  def create() do
    spawn(__MODULE__, :loop, [0])
  end

  def inc(pid, x) do
    pid |> send({:inc, x})
  end

  def dec(pid, x) do
    pid |> send({:dec, x})
  end

  def get(pid) do
    pid |> send({:get, self()})
    receive do
      counter -> counter
    end
  end

  def loop(counter) do
    receive do
      {:inc, x} -> loop(counter + x)
      {:dec, x} -> loop(counter - x)
      {:get,caller} -> send(caller, counter)
                       loop(counter)
    end
  end
end
```

```elixir
defmodule Counter do
  use GenServer

  def create() do
    GenServer.start_link(__MODULE__, 0)
  end

  def inc(server, x) do
    GenServer.cast(server, {:inc, x})
  end

  def dec(server, x) do
    GenServer.cast(server, {:dec, x})
  end

  def get(server) do
    GenServer.call(server, {:get})
  end

  def handle_cast({:inc, x}, counter) do
    {:noreply, counter + x}
  end

  def handle_cast({:dec, x}, counter) do
    {:noreply, counter - x}
  end

  def handle_call({:get}, _from, counter) do
    {:reply, counter, counter}
  end
end
```

How?

How?

We've been using it the entire time


The Elixir Tool Set
Erlang OTP Features
Behaviours
Mix
ETS
GenServer
Application
Mnesia
Crypto
Building Blocks for **Distributed**, **Scalable**, and **Robust** Applications