Eventos de Domínio

Podem Ser Simples

Bernardo Amorim


stone

Eventos

Um *evento* é algo que aconteceu, um *fato*

Exemplos

- **UsuarioCriado**(id: 10, email: "email@exemplo.com")
- **SenhaAtualizada**(id: 10)
- **EmailVerificado**(id: 10, email: "email@exemplo.com")
- **EmailPrincipalAlterado**(id: 10, email: "email@exemplo.com")

O que você quer dizer por event-driven?

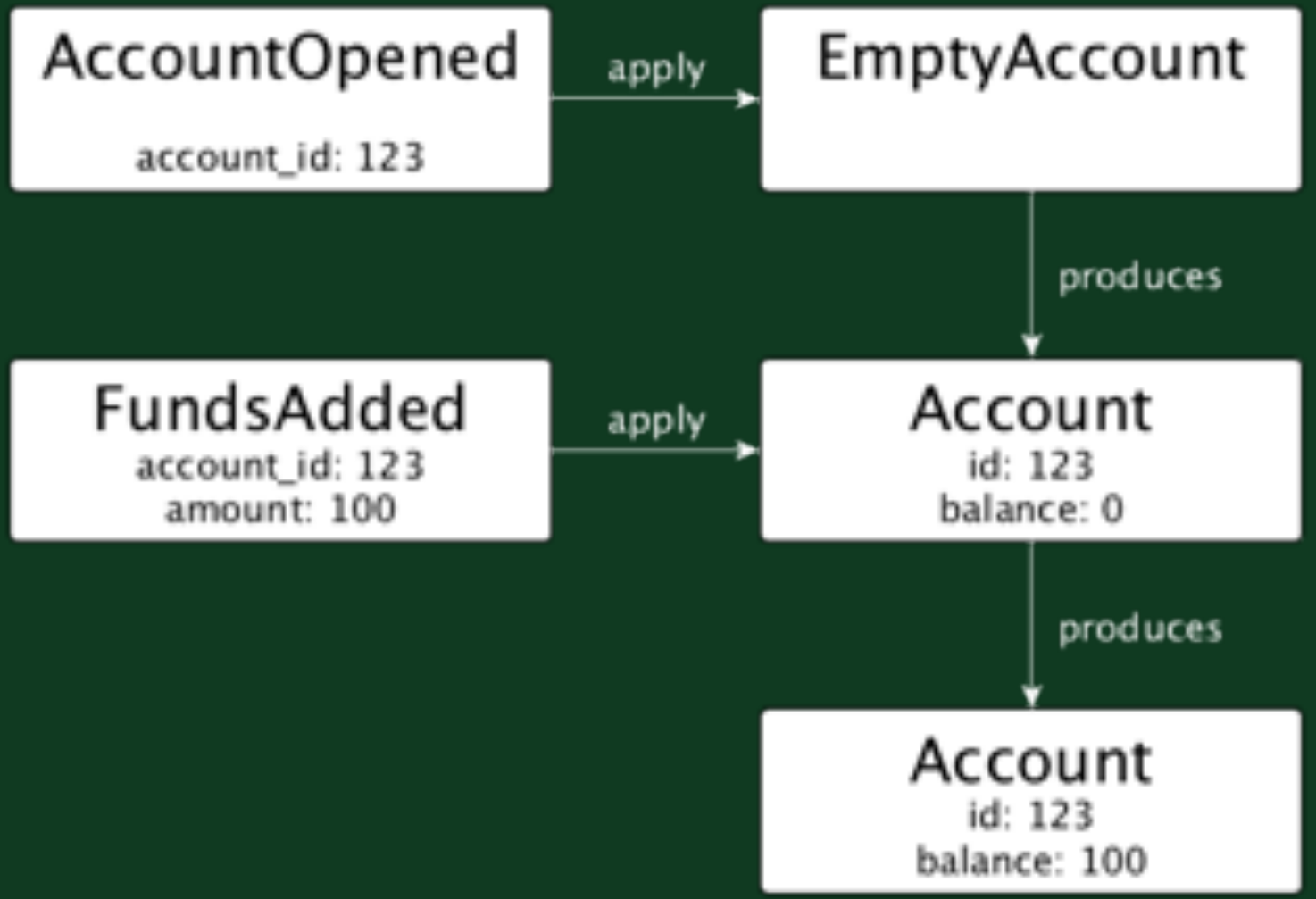
- Martin Fowler

O que você quer dizer por event-driven?

- Notificação por evento
- Transferência de estado por eventos
- Event Sourcing
- CQRS
- Event Collaboration

Event Sourcing


AccountOpened
account_id: 123
FundsAdded
account_id: 123
amount: 100
FundsAdded
account_id: 123
amount: 50


AccountOpened
account_id: 123
apply
EmptyAccount
produces
FundsAdded
account_id: 123
amount: 100
apply
Account
id: 123
balance: 0
produces
Account
id: 123
balance: 100

$$S_n = \text{apply}(S_{n-1}, E_n)$$

```
@spec apply(previous_state :: State.t(), event()) :: State.t()
```

$$S_n = \text{reduce}(E, S_0, \text{apply})$$

```
def current_state(events, initial_State) do
  Enum.reduce(events, initial_state, &apply/2)
end
```

Exemplos reais?

State: Files

Events: Commits


A nossa jornada
Event-Driven

README.md

Commanded

Use Commanded to build your own Elixir applications following the CQRS/ES pattern.

Provides support for:

- Command registration and dispatch.
- Hosting and delegation to aggregates.
- Event handling.
- Long running process managers.

Commanded provides a solid technical foundation for you to build on. It allows you to focus on modelling your domain, the most important part of your app, creating a better application at a faster pace.

You can use Commanded with one of the following event stores for persistence:

- EventStore Elixir library using PostgreSQL for persistence
- Greg Young's Event Store.
- In-memory event store included for test use only.

Please refer to the CHANGELOG for features, bug fixes, and any upgrade advice included for each release.

Supporting Commanded

You can help support Commanded by helping to fund its ongoing development, new features, and releases.

- Become a backer or sponsor on OpenCollective.

- Changelog
- Wiki
- What is CQRS/ES?
- Frequently asked questions
- Getting help
- Latest published Hex package & documentation

Modelo de Domínio

```elixir
defmodule CoreBanking.Accounts.Aggregates.Account do
  defstruct [
    :account_id,
    balance: 0
  ]
end

defmodule CoreBanking.Accounts.Events.AccountOpened do
  defstruct [
    :operator_id,
    :account_id,
    :branch_code,
    :owner_id
  ]
end
```

Comando (escrita)

```elixir
defmodule CoreBanking.Accounts.Commands.OpenAccount do
  import Exchema.Notation

  structure(
    operator_id: Exchema.Types.String,
    account_id: CoreBanking.Types.UUID,
    branch_code: Exchema.Types.String,
    owner_id: CoreBanking.Types.UUID
  )
end
```

Processando comandos

```elixir
defmodule CoreBanking.Accounts.Aggregates.Account do
  def execute(%Account{account_id: nil}, %OpenAccount{} = cmd) do
    %AccountOpened{
      account_id: cmd.account_id,
      branch_code: cmd.branch_code,
      operator_id: cmd.operator_id,
      owner_id: cmd.owner_id
    }
  end
end
```

Atualizando o estado

```elixir
defmodule CoreBanking.Accounts.Aggregates.Account do
  def apply(_, %AccountOpened{account_id: id}) do
    %Account{balance: 0, account_id: id}
  end
end
```

Roteando e disparando comandos

```elixir
defmodule CoreBanking.Accounts.Router do
  use Commanded.Commands.Router

  alias CoreBanking.Accounts.Aggregates.Account
  alias CoreBanking.Accounts.Commands.OpenAccount

  dispatch([OpenAccount], to: Account)
end

CoreBanking.Accounts.Router.dispatch(%OpenAccount{
  ...
})
```

Modelo de leitura

```elixir
defmodule CoreBanking.ViewModel.Account do
  use Ecto.Schema

  @primary_key {:id, :binary_id, autogenerate: false}

  schema "accounts" do
    field(:operator_id, :string)
    field(:balance, :integer)
    field(:branch_code, :string)
    field(:account_code, :string)
    field(:owner_id, :string)
  end
end
```

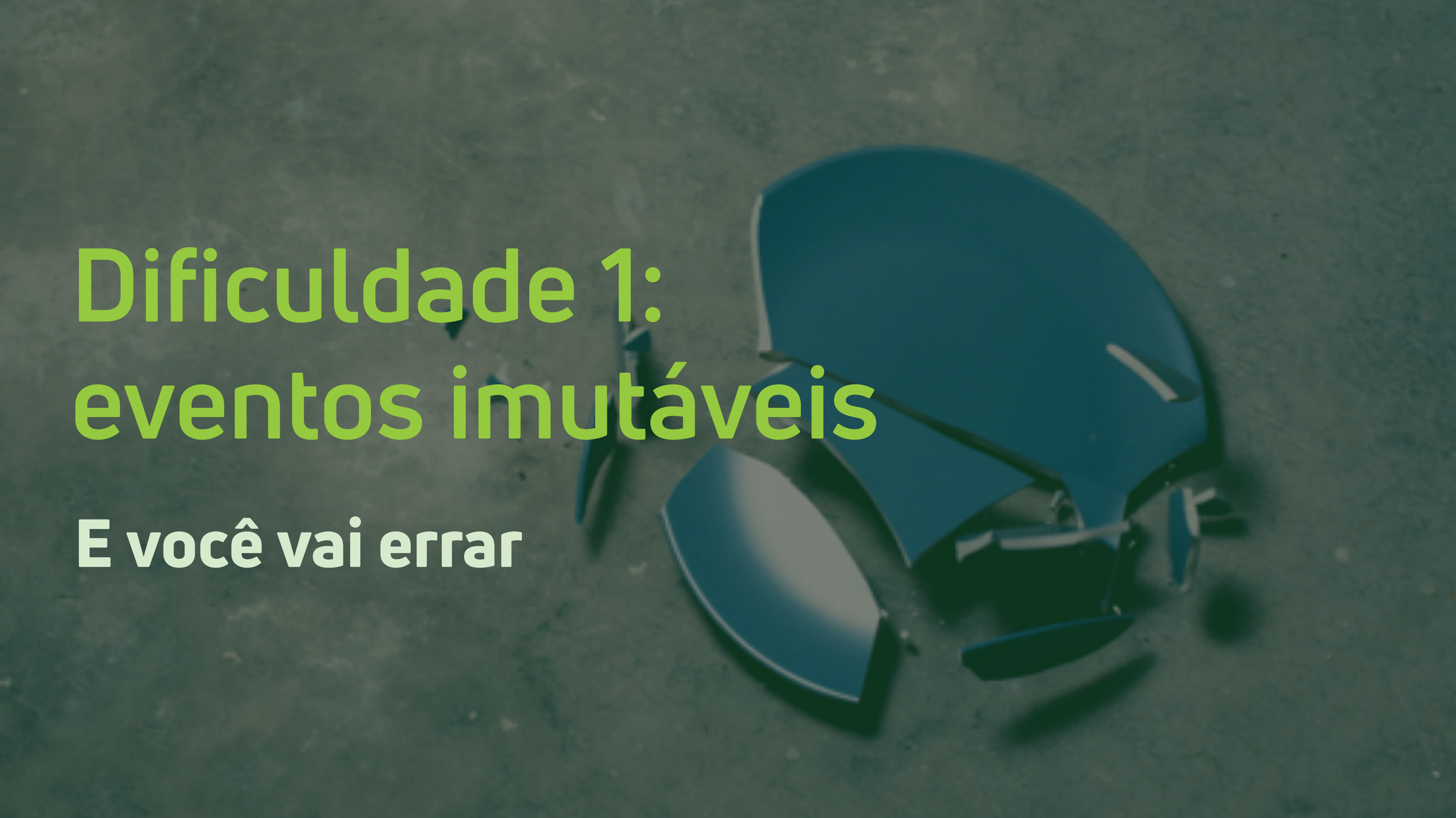
Projetando dados de leitura

```elixir
defmodule CoreBanking.Accounts.Projectors.Account do
  use Commanded.Projections.Ecto, name: "Projectors.Account"

  alias CoreBanking.Accounts.Events.AccountOpened
  alias CoreBanking.ViewModel.Account
  alias Ecto.Multi

  project %AccountOpened{} = evt do
    Multi.insert(multi, :account, %Account{
      operator_id: evt.operator_id,
      id: evt.account_id,
      owner_id: evt.owner_id,
      balance: 0
    })
  end
end
```

Envolvendo multiplos aggregates

```elixir
defmodule CoreBanking.Accounts.Sagas.AccountSaga do
  use Commanded.ProcessManagers.ProcessManager,
    name: "AccountSaga",
    router: CoreBanking.Router,
    consistency: :strong

  alias __MODULE__, as: Saga

  alias CoreBanking.Accounts.{
    Commands.RegisterAccount,
    Events.AccountOpened,
    Events.AccountRegistered
  }

  defstruct [
    :branch_code,
    :account_id
  ]

  def interested?(%AccountOpened{} = evt), do: {:start, evt.account_id}
  def interested?(%AccountRegistered{} = evt), do: {:stop, evt.account_id}

  def handle(%Saga{}, %AccountOpened{} = evt) do
    %RegisterAccount{
      account_id: evt.account_id,
      branch_code: evt.branch_code
    }
  end

  def apply(%Saga{} = state, %AccountOpened{} = evt) do
    %Saga{
      state
      | account_id: evt.account_id,
        branch_code: evt.branch_code
    }
  end
end
```

Dificuldades



Dificuldade 1: eventos imutáveis

E você vai errar

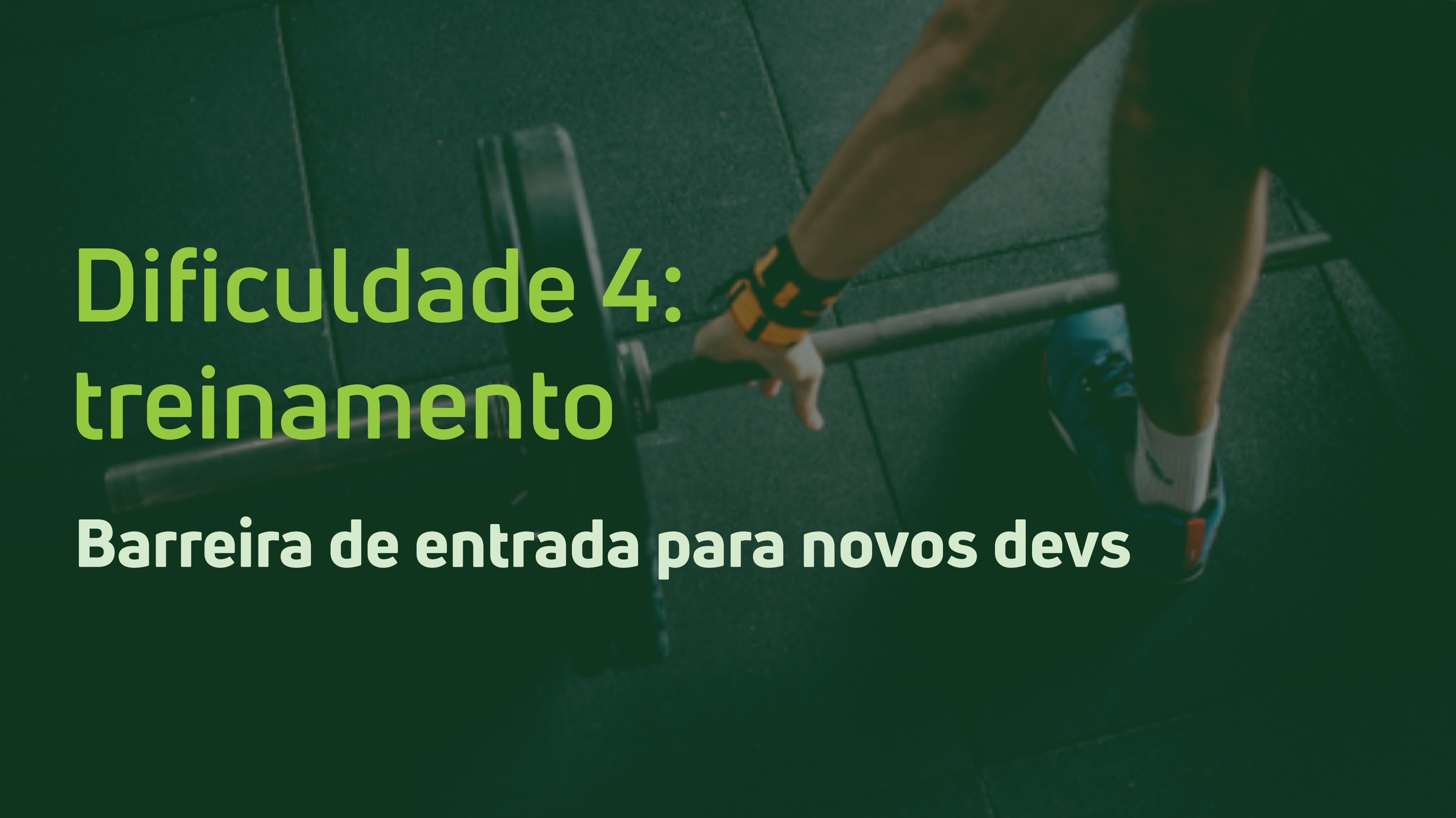
Dificuldade 2: consistência eventual

Saudades Repo.transaction

Dificuldade 3: muitos conceitos juntos

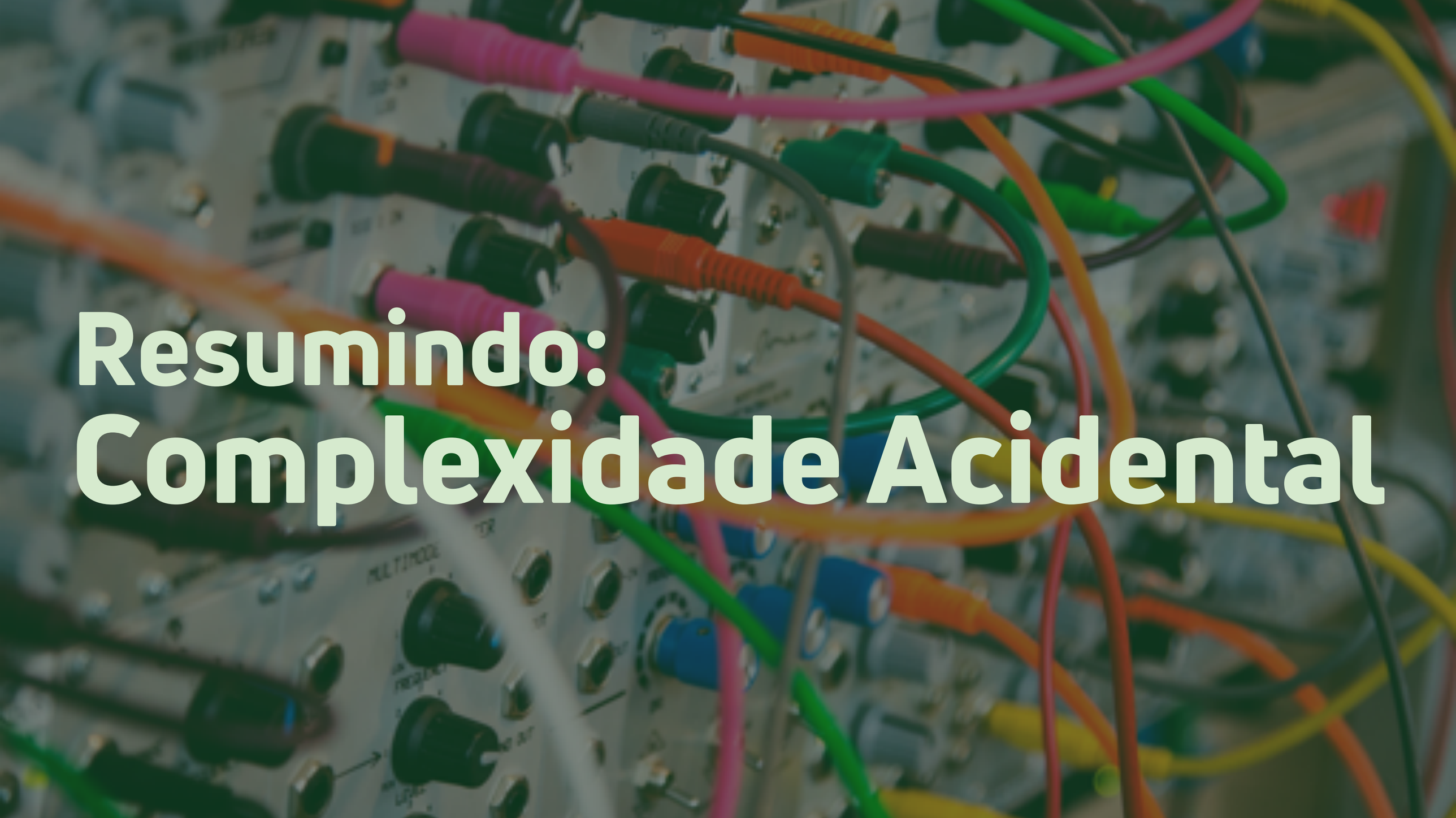
Que às vezes você não precisa


Events
Commands
Aggregates
Projections
Sagas



Dificuldade 4: treinamento

Barreira de entrada para novos devs



Resumindo:
Complexidade Acidental

Voltando atrás

Nem tudo precisa ser Event-Sourced

Máquinas de estado finitas

Apenas 1 modelo

```elixir
defmodule CoreBanking.Account do
  use Ecto.Schema

  @primary_key {:id, :binary_id, autogenerate: true}

  schema "accounts" do
    field(:operator_id, :string)
    field(:balance, :integer, default: 0)
    field(:branch_code, :string)
    field(:account_code, :string)
    field(:owner_id, :string)
    field(:account_opened_at, :naive_datetime_usec)
    field(:account_closed_at, :naive_datetime_usec)
  end
end
```

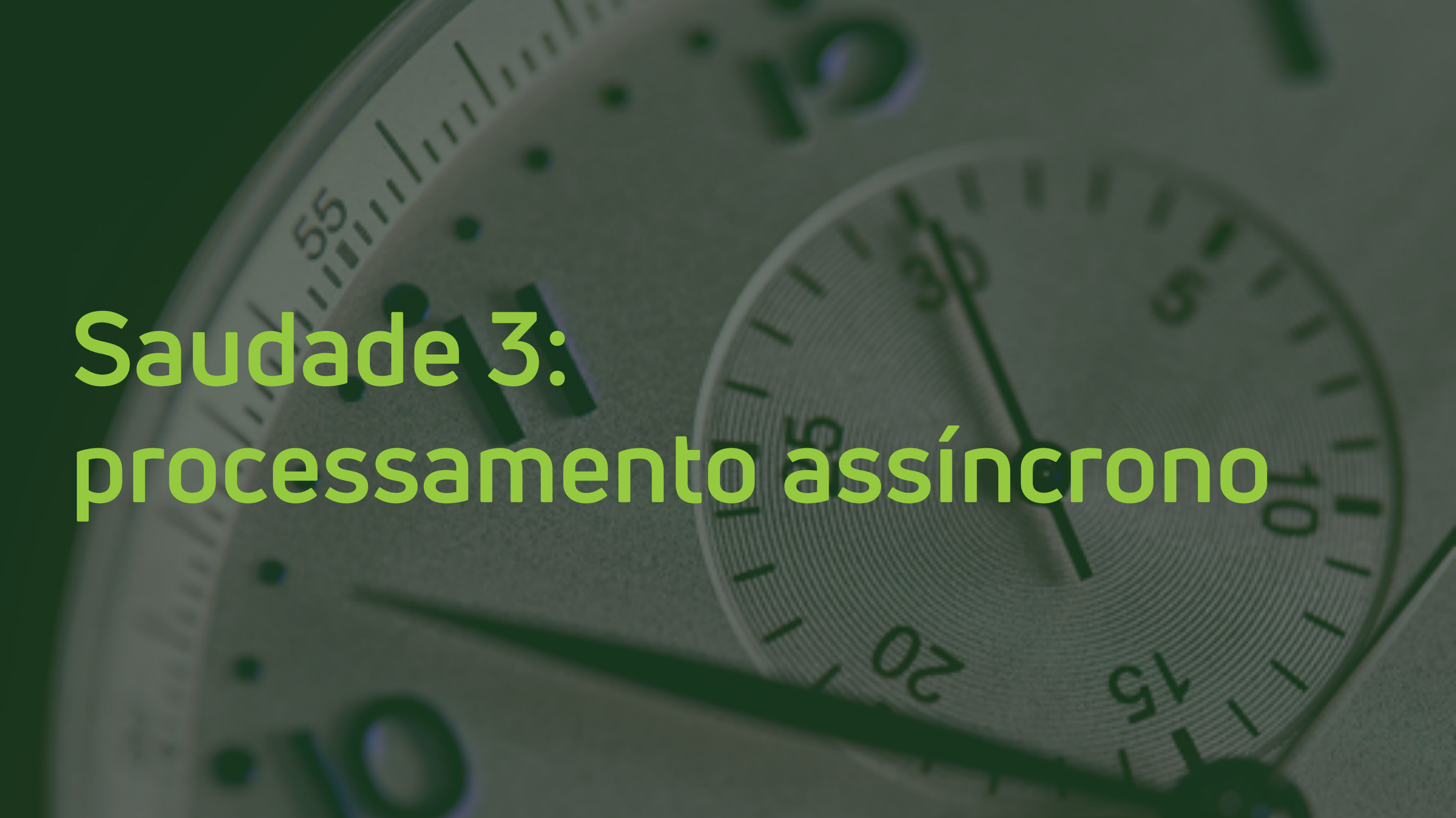
Criando uma conta

```elixir
def open_account(params) do
  %Account{}
  |> cast(params, [:account_code, :owner_id])
  |> put_change(:account_opened_at, NaiveDateTime.utc_now())
  |> unique_constraint(:account_code)
  |> validate_xxx()
end
```

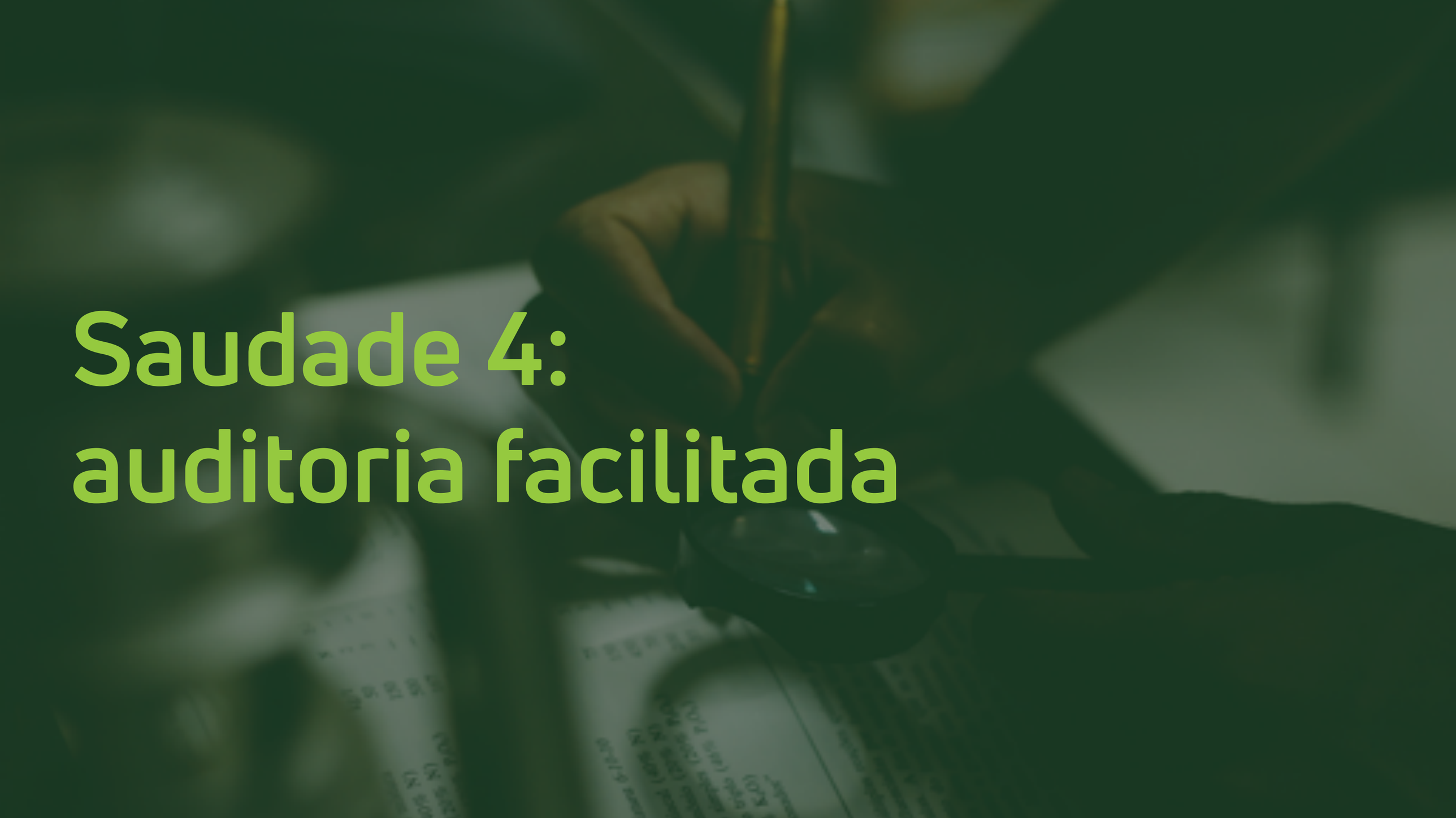

Saudades

Saudade 1: linguagem de domínio

Saudade 2: efeitos desacoplados



Saudade 3: processamento assíncrono



Saudade 4: auditoria facilitada

Explorando alternativas