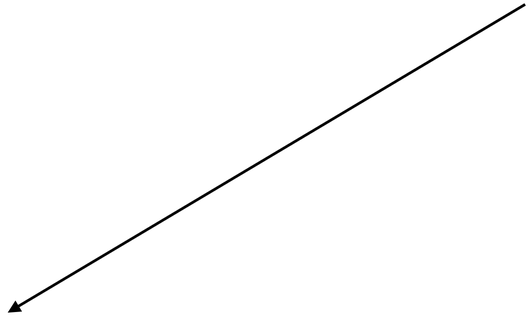
Eventsourcing and CQRS in Elixir

Vasilis Spilka

Make it simple

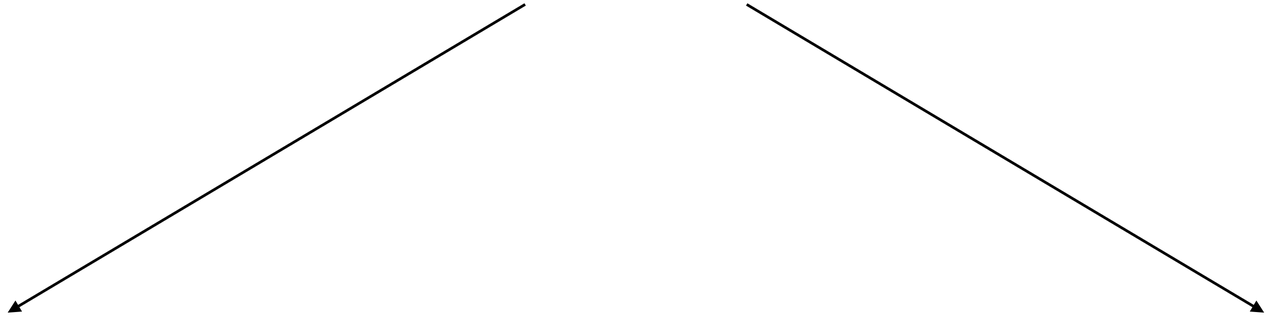
Event-sourcing is not hard, it is just different

Event-



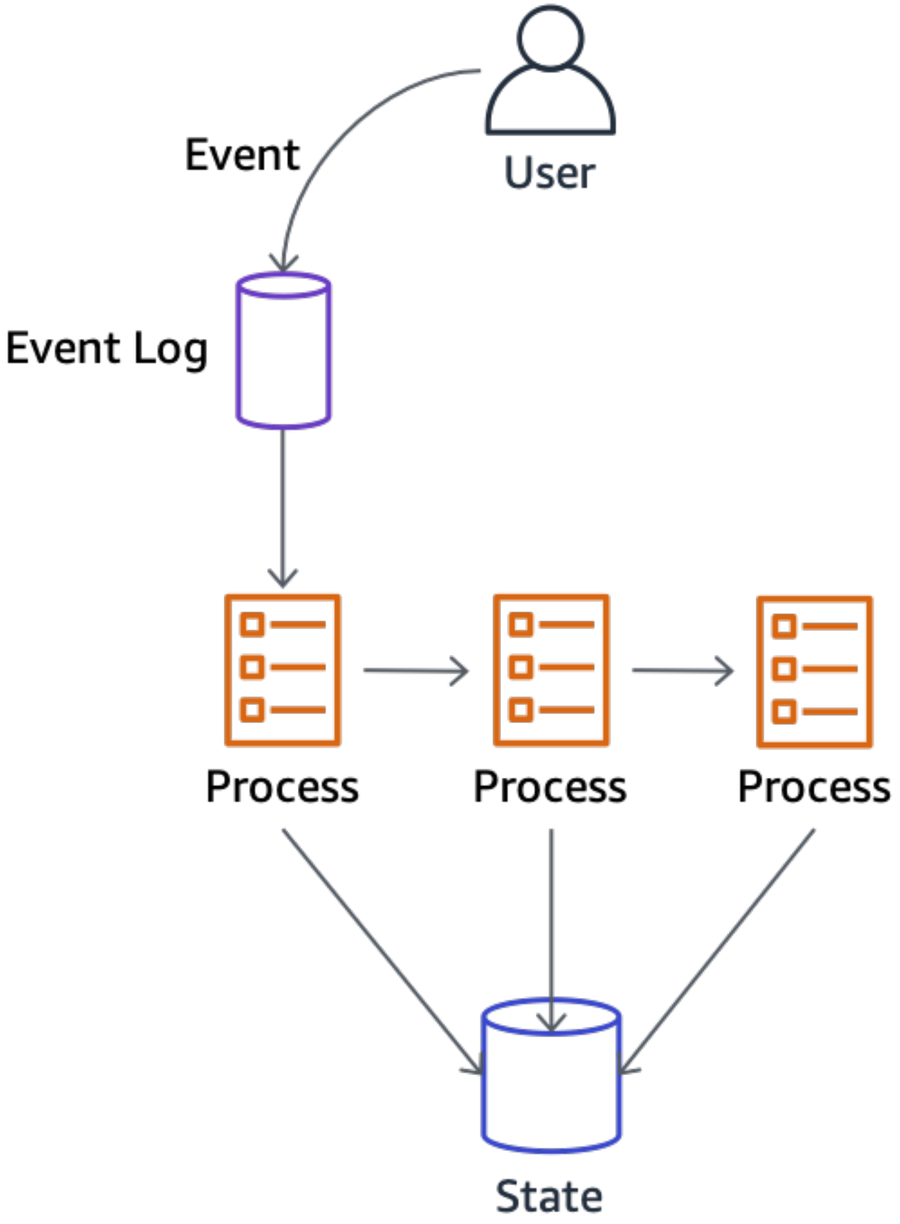
A stored record of something significant that happened in the business (domain).

Event-Sourcing



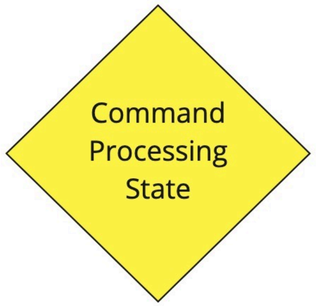
A stored record of something significant that happened in the business (domain).

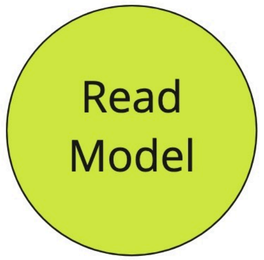
Application state is created (sourced) from the events.


Event
User
Event Log
Process
Process
Process
State

CQRS

Command Query Responsibility Segregation

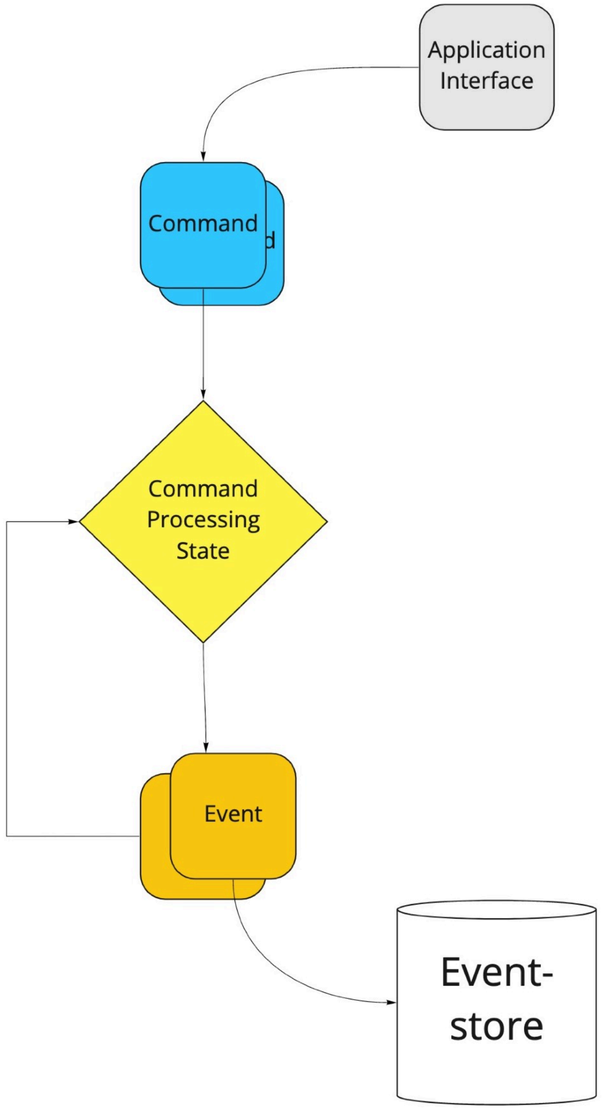




(Query State)

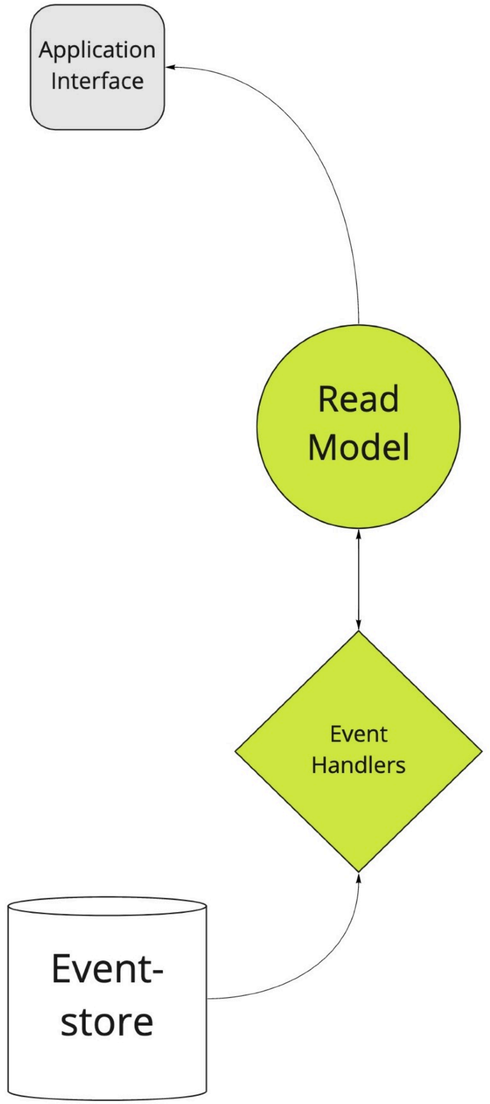
CQRS

Command Query Responsibility Segregation



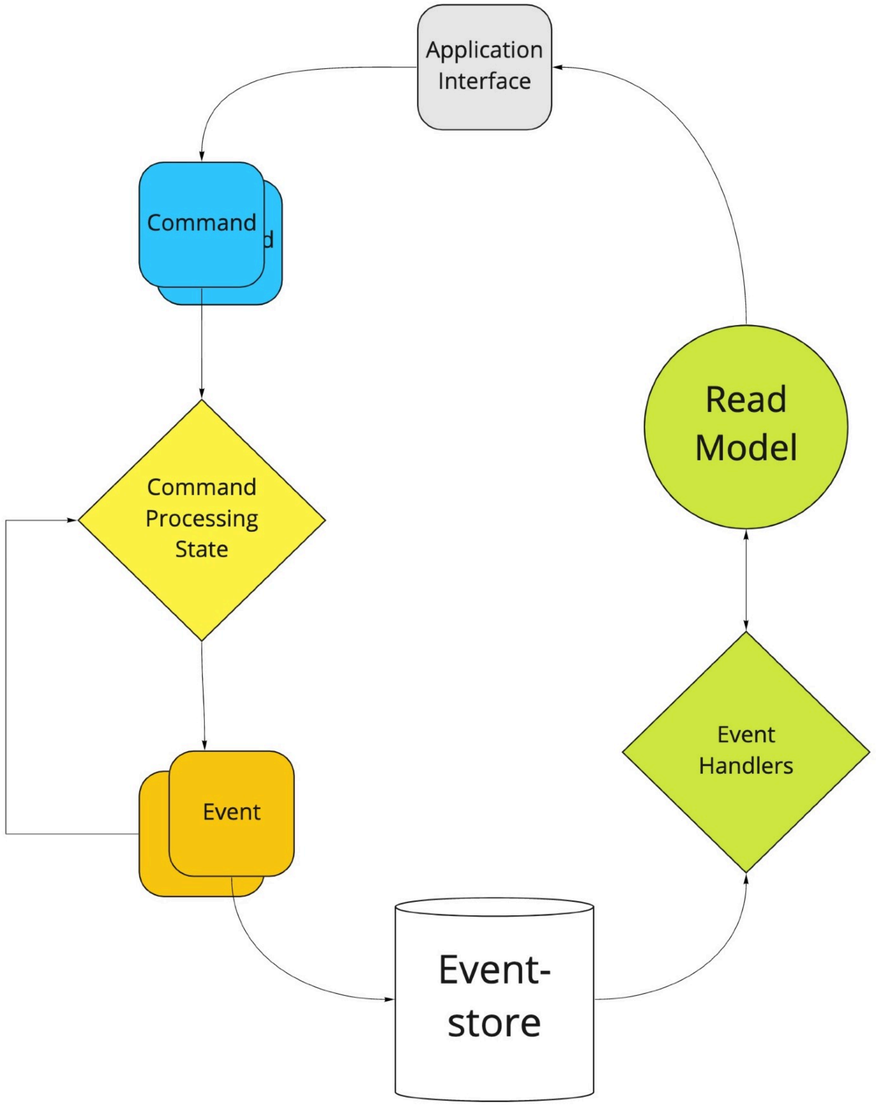
CQRS

Command Query Responsibility Segregation



CQRS

Command Query Responsibility Segregation



Let’s Build a Bank

Let’s Build a Bank

With Commanded!

Requirements

Open Accounts

Deposit/Withdraw Money

View account balances

Transfer Money Between Accounts

Keep track of accounting

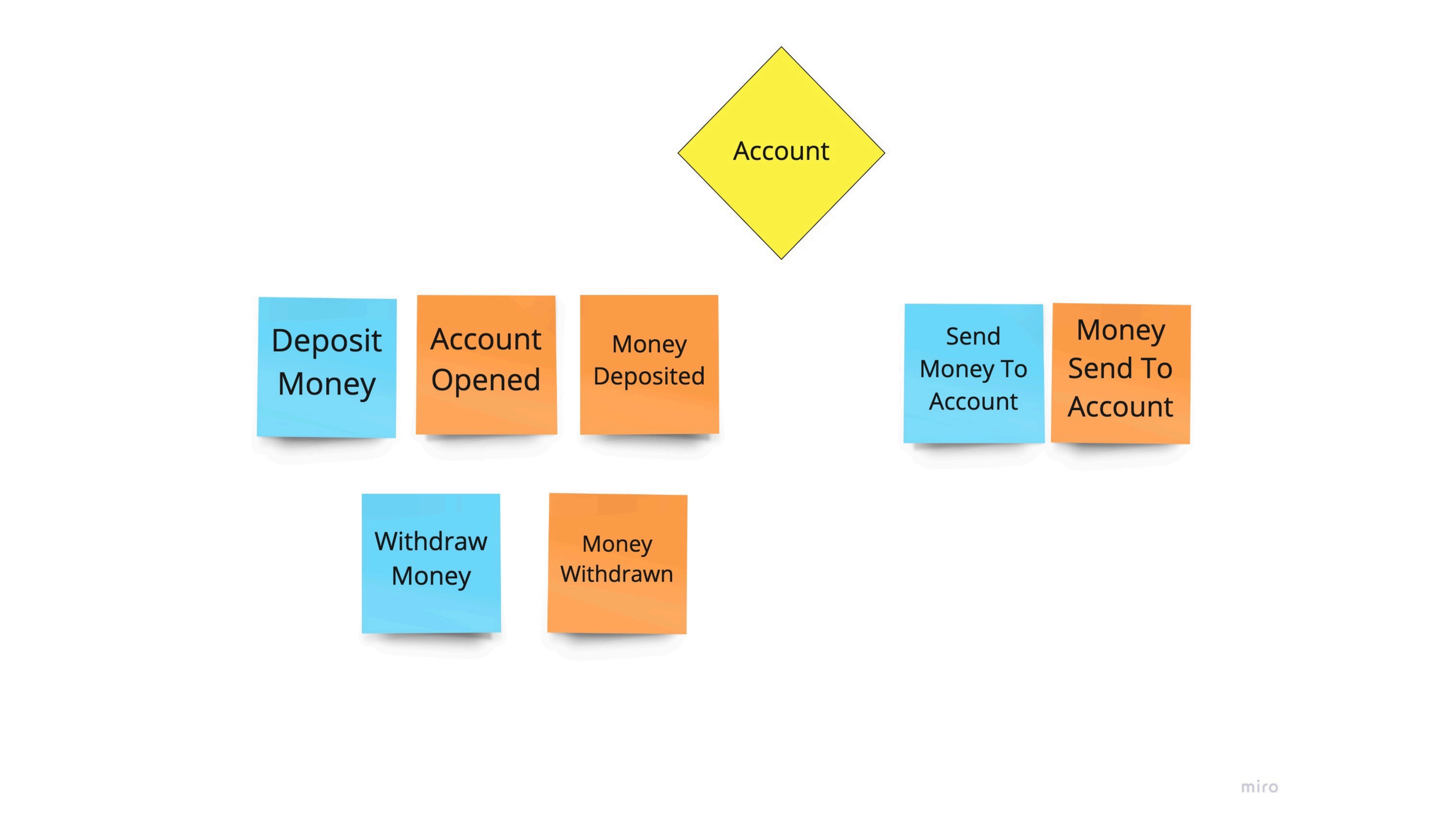
Account Opened

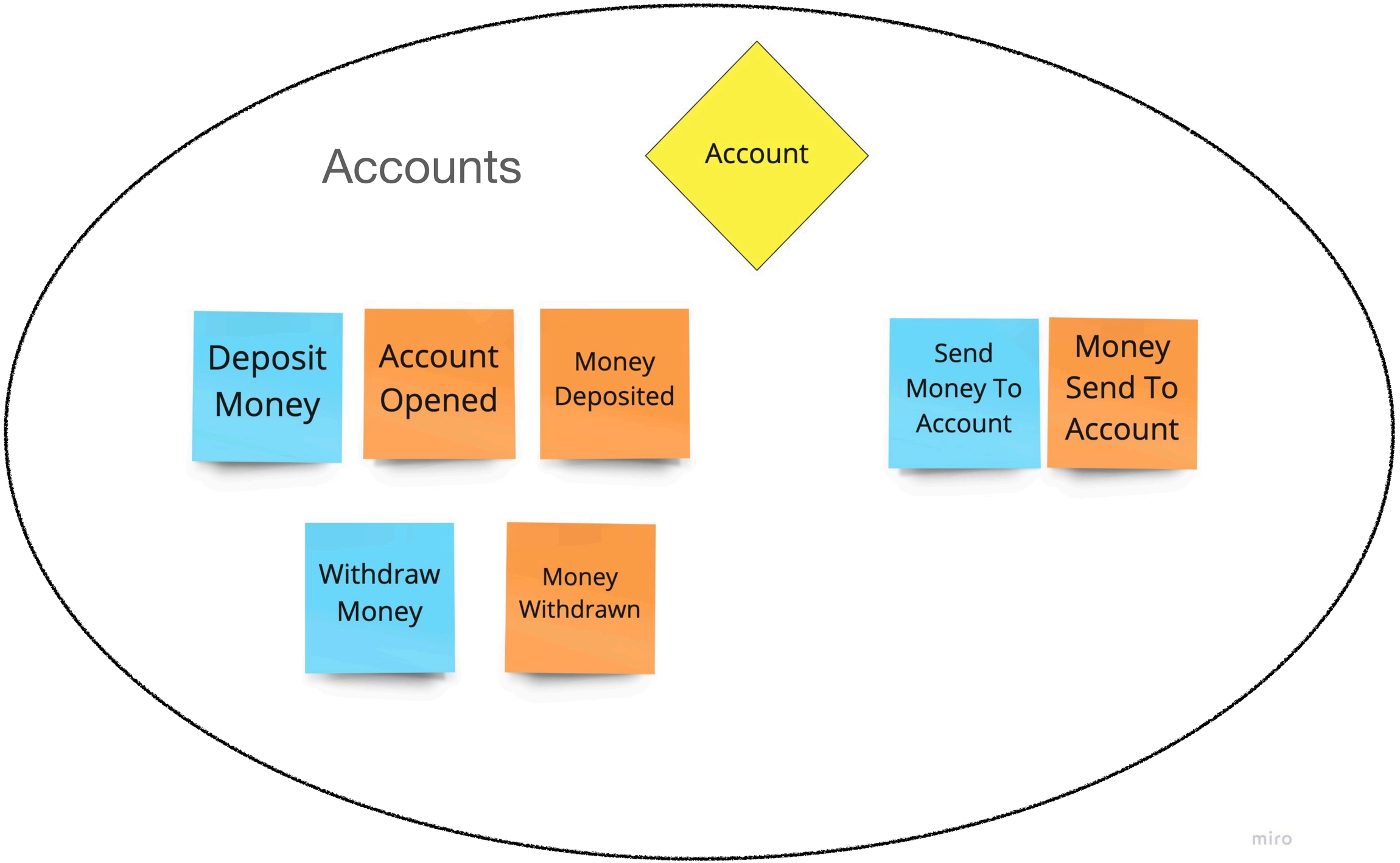
Money Deposited

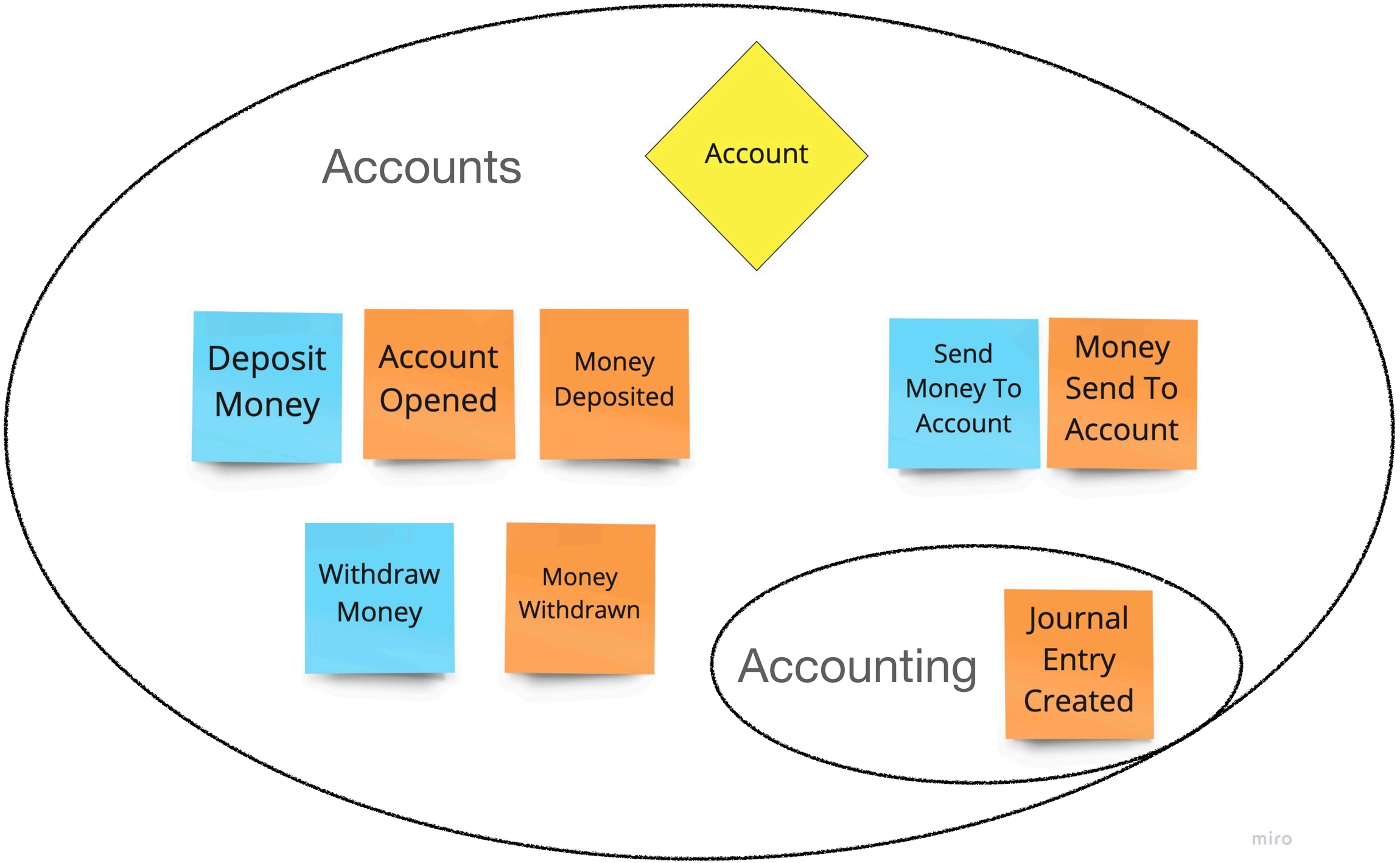
Money Send To Account

Money Withdrawn


Deposit Money
Account Opened
Money Deposited
Send Money To Account
Money Send To Account
Withdraw Money
Money Withdrawn


Account
Deposit Money
Account Opened
Money Deposited
Send Money To Account
Money Send To Account
Withdraw Money
Money Withdrawn


Accounts
Account
Deposit Money
Account Opened
Money Deposited
Send Money To Account
Money Send To Account
Withdraw Money
Money Withdrawn


Accounts
Account
Deposit Money
Account Opened
Money Deposited
Send Money To Account
Money Send To Account
Withdraw Money
Money Withdrawn
Accounting
Journal Entry Created

Common Types

```elixir
defmodule Bank do
  @moduledoc "Banking application"

  @type account_number() :: binary()
  @type amount() :: integer()
  @type account_entries() :: %{account_number() => amount()}
end
```

Accounts Context

Application Interface

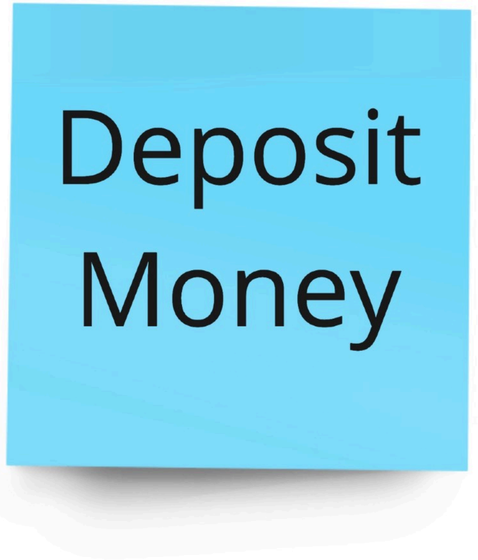
lib/core/accounts.ex

```elixir
defmodule Bank.Core.Accounts do
  @moduledoc "Core context of user Accounts."

  @spec deposit_money(Bank.account_number(), Bank.amount()) ::
  def deposit_money(acc_id, amount) do
    ...
  end

  @spec withdraw_money(Bank.account_number(), Bank.amount()) ::
  def withdraw_money(acc_id, amount) do
    ...
  end

  @spec send_money(Bank.account_number(), Bank.account_number(), Bank.amount()) ::
  def send_money(from_acc_id, to_acc_id, amount) do
    ...
  end

  @spec view_balance(Bank.account_number()) :: Bank.amount()
  def view_balance(account) do
    ...
  end
end
```

Commands

lib/core/commands/deposit_money.ex

```
defmodule Bank.Core.Commands.DepositMoney do
  @type t :: %__MODULE__{
          account_id: Bank.account_number(),
          amount: Bank.amount()
        }

  defstruct [:account_id, :amount]
end
```



lib/core/commands/withdraw_money.ex

```
defmodule Bank.Core.Commands.WithdrawMoney do
  @type t :: %__MODULE__{
          account_id: Bank.account_number(),
          amount: Bank.amount()
        }

  defstruct [:account_id, :amount]
end
```

Events

lib/core/events/money_deposited.ex

```
defmodule Bank.Core.Events.MoneyDeposited do
  @type t :: %__MODULE__{
          account_id: Bank.account_number(),
          amount: Bank.amount()
        }

  defstruct [:account_id, :amount]
end
```

lib/core/events/money_withdrawn.ex

```
defmodule Bank.Core.Events.MoneyWithdrawn do
  @type t :: %__MODULE__{
          account_id: Bank.account_number(),
          amount: Bank.amount()
        }

  defstruct [:account_id, :amount]
end
```

Money Deposited

lib/core/events/account_opened.ex

```
defmodule Bank.Core.Events.AccountOpened do
  @type t :: %__MODULE__{
    account_id: Bank.account_number()
  }

  defstruct [:account_id]
end
```

Money Withdrawn

Account Opened

Journal Entry Event

lib/core/events/journal_entry_created.ex

```elixir
defmodule Bank.Core.Events.JournalEntryCreated do
  alias Bank.Core.Accounting

  @type t :: %__MODULE__{
          journal_entry_uuid: binary(),
          debit: Accounting.account_entries(),
          credit: Accounting.account_entries(),
        }

  defstruct [:journal_entry_uuid, :debit, :credit]
end
```

Journal Entry Event

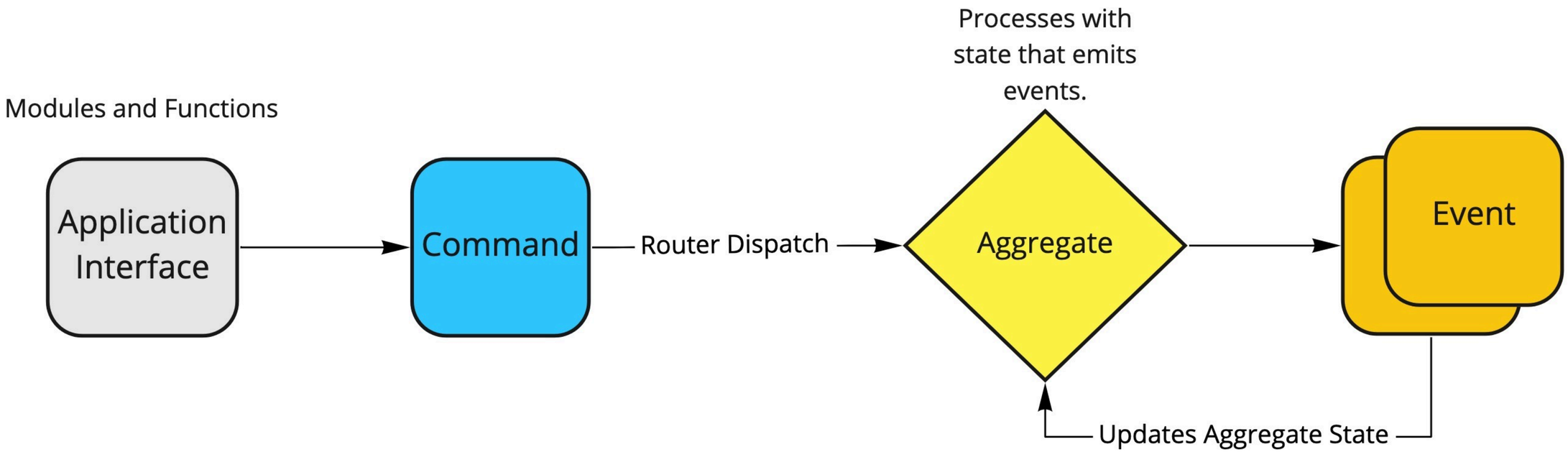
lib/core/events/journal_entry_created.ex

```elixir
defmodule Bank.Core.Events.JournalEntryCreated do
  alias Bank.Core.Accounting

  @type t :: %__MODULE__{
          journal_entry_uuid: binary(),
          debit: Accounting.account_entries(),
          credit: Accounting.account_entries(),
        }

  defstruct [:journal_entry_uuid, :debit, :credit]
end
```


```elixir
%Bank.Core.Events.JournalEntryCreated{
  journal_entry_uuid: "707dbaf5-7afd-45d1-a52c-9fa590169eea",
  credit: %{"000-001" => 200},
  debit: %{"000-0005" => 100, "000-0012" => 100},
}
```

Aggregate

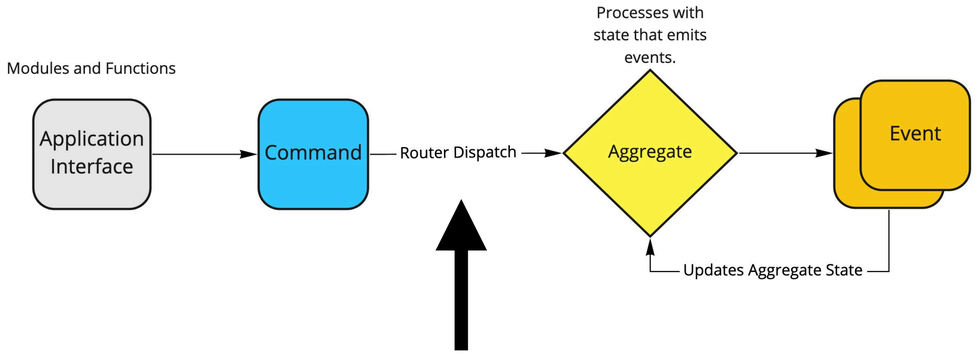
A business object that encapsulate relevant information required to validate commands and record events.



Router

lib/core/router.ex

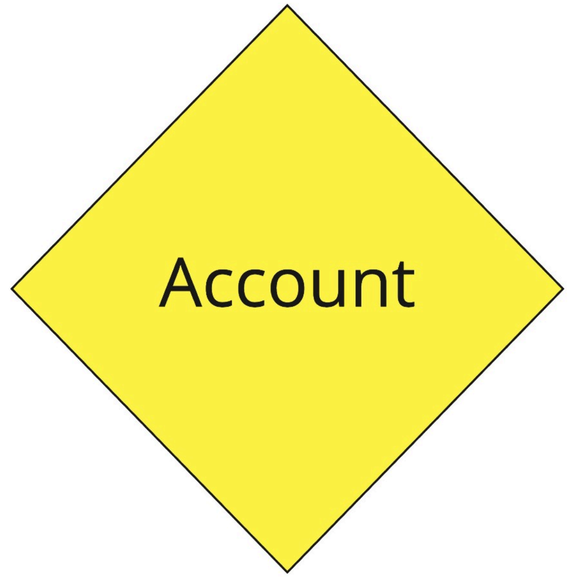
```
defmodule Bank.Core.Router do
  use Commanded.Commands.Router

  alias Bank.Core.Commands

  dispatch(
    [
      Commands.DepositMoney,
      Commands.WithdrawMoney
    ],
    to: Bank.Core.Accounts.Account,
    identity: :account_id
  )
end
```



Accounts Aggregate

lib/core/accounts/account.ex

```
defmodule Bank.Core.Accounts.Account do
  @moduledoc "Account Aggregate."

  @type t() :: %__MODULE__{
          id: Bank.account_number(),
          balance: Bank.amount()
        }
  defstruct [:id, balance: 0]

  @type event() :: struct()
  @type cmd() :: struct()

  @spec execute(t(), command()) :: [event()] | {:error, term}
  def execute(state, cmd) do
    ...
  end

  @spec apply(t(), event()) :: t()
  def apply(state, evt) do
    ...
  end
end
```

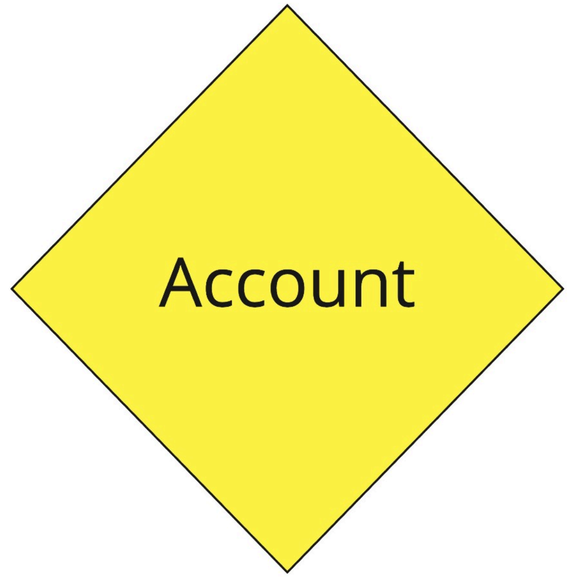


Accounts Aggregate

lib/core/accounts/account.ex

```
defmodule Bank.Core.Accounts.Account do
  @moduledoc "Account Aggregate."

  @type t() :: %__MODULE__{
          id: Bank.account_number(),
          balance: Bank.amount()
        }
  defstruct [:id, balance: 0]

  @type event() :: struct()
  @type cmd() :: struct()

  @spec execute(t(), command()) :: [event()] | {:error, term}
  def execute(state, cmd) do
    ...
  end

  @spec apply(t(), event()) :: t()
  def apply(state, evt) do
    ...
  end
end
```

State

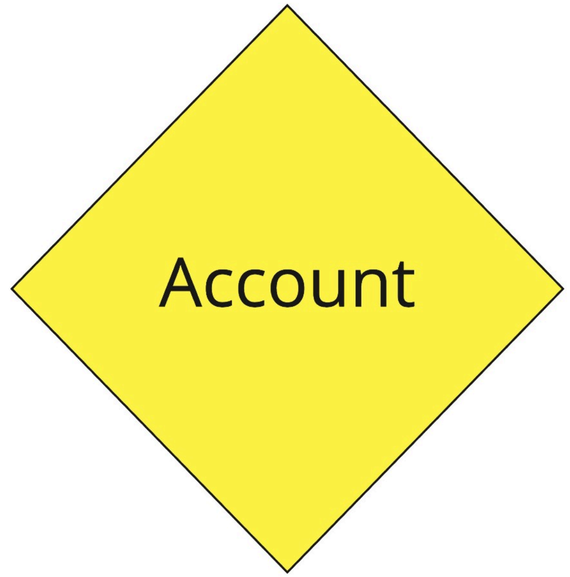


Accounts Aggregate

lib/core/accounts/account.ex

```elixir
defmodule Bank.Core.Accounts.Account do
  @moduledoc "Account Aggregate."

  @type t() :: %__MODULE__{
          id: Bank.account_number(),
          balance: Bank.amount()
        }
  defstruct [:id, balance: 0]

  @type event() :: struct()
  @type cmd() :: struct()

  @spec execute(t(), command()) :: [event()] | {:error, term}
  def execute(state, cmd) do
    ...
  end

  @spec apply(t(), event()) :: t()
  def apply(state, evt) do
    ...
  end
end
```

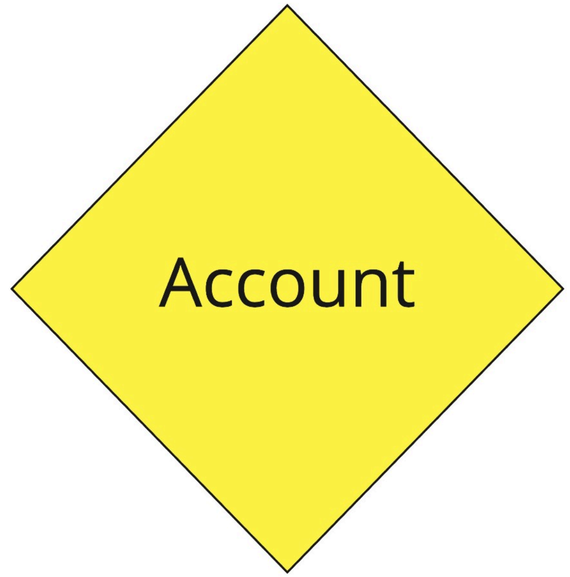
Callbacks



Accounts Aggregate

lib/core/accounts/account.ex

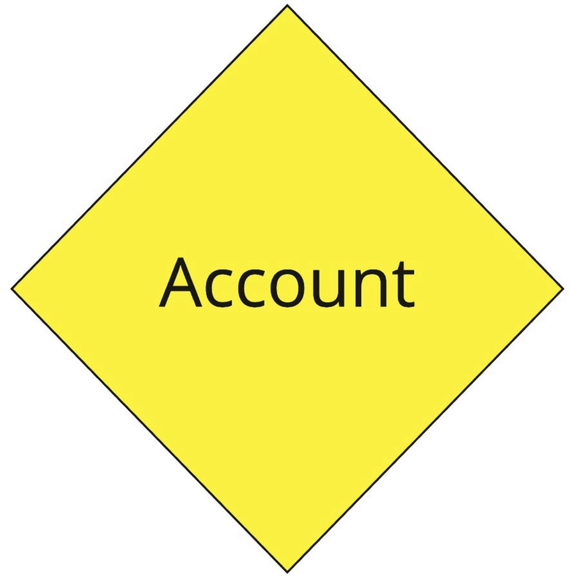
```elixir
defmodule Bank.Core.Accounts.Account do
  @moduledoc "Account Aggregate."

  @type t() :: %__MODULE__{
          id: Bank.account_number(),
          balance: Bank.amount()
        }
  defstruct [:id, balance: 0]

  @type event() :: struct()
  @type cmd() :: struct()

  @spec execute(t(), command()) :: [event()] | {:error, term}
  def execute(state, cmd) do
    ...
  end

  @spec apply(t(), event()) :: t()
  def apply(state, evt) do
    ...
  end
end
```



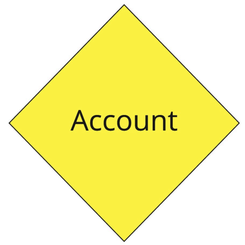
Accounts Aggregate

lib/core/accounts/account.ex

```elixir
defmodule Bank.Core.Accounts.Account do
  @moduledoc "Account Aggregate."

  @type t() :: %__MODULE__{
          id: Bank.account_number(),
          balance: Bank.amount()
        }
  defstruct [:id, balance: 0]

  @type event() :: struct()
  @type cmd() :: struct()

  @spec execute(t(), command()) :: [event()] | {:error, term}
  def execute(state, cmd) do
    ...
  end

  @spec apply(t(), event()) :: t()
  def apply(state, evt) do
    ...
  end
end
```



Accounts Aggregate

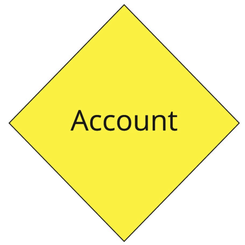


lib/core/accounts/account.ex

```elixir
def execute(%Account{}, %DepositMoney{account_id: "000-000"}),
  do: {:error, :unable_to_create_account}

def execute(%Account{id: nil}, %DepositMoney{} = cmd) do
  [
    %AccountOpened{
      account_id: cmd.account_id
    },
    %MoneyDeposited{
      account_id: cmd.account_id,
      amount: cmd.amount
    },
    %JournalEntryCreated{
      journal_entry_uuid: Ecto.UUID.generate(),
      debit: %{"#{cmd.account_id}" => cmd.amount},
      credit: %{"000-000" => cmd.amount}
    }
  ]
end
```

```elixir
def execute(%Account{id: nil}, _cmd),
  do: {:error, :not_found}

def execute(%Account{}, %DepositMoney{} = cmd) do
  [
    %MoneyDeposited{
      account_id: cmd.account_id,
      amount: cmd.amount
    },
    %JournalEntryCreated{
      journal_entry_uuid: Ecto.UUID.generate(),
      debit: %{"#{cmd.account_id}" => cmd.amount},
      credit: %{"000-000" => cmd.amount}
    }
  ]
end
```

Accounts Aggregate



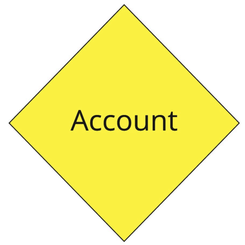
lib/core/accounts/account.ex

```elixir
def execute(%Account{}, %DepositMoney{account_id: "000-000"}),
  do: {:error, :unable_to_create_account}
```

```elixir
def execute(%Account{id: nil}, %DepositMoney{} = cmd) do
  [
    %AccountOpened{
      account_id: cmd.account_id
    },
    %MoneyDeposited{
      account_id: cmd.account_id,
      amount: cmd.amount
    },
    %JournalEntryCreated{
      journal_entry_uuid: Ecto.UUID.generate(),
      debit: %{"#{cmd.account_id}" => cmd.amount},
      credit: %{"000-000" => cmd.amount}
    }
  ]
end
```

```elixir
def execute(%Account{id: nil}, _cmd),
  do: {:error, :not_found}

def execute(%Account{}, %DepositMoney{} = cmd) do
  [
    %MoneyDeposited{
      account_id: cmd.account_id,
      amount: cmd.amount
    },
    %JournalEntryCreated{
      journal_entry_uuid: Ecto.UUID.generate(),
      debit: %{"#{cmd.account_id}" => cmd.amount},
      credit: %{"000-000" => cmd.amount}
    }
  ]
end
```

Accounts Aggregate



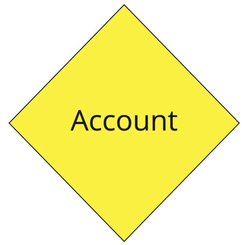
lib/core/accounts/account.ex

```elixir
def execute(%Account{}, %DepositMoney{account_id: "000-000"}),
  do: {:error, :unable_to_create_account}

def execute(%Account{id: nil}, %DepositMoney{} = cmd) do
  [
    %AccountOpened{
      account_id: cmd.account_id
    },
    %MoneyDeposited{
      account_id: cmd.account_id,
      amount: cmd.amount
    },
    %JournalEntryCreated{
      journal_entry_uuid: Ecto.UUID.generate(),
      debit: %{"#{cmd.account_id}" => cmd.amount},
      credit: %{"000-000" => cmd.amount}
    }
  ]
end
```

```elixir
def execute(%Account{id: nil}, _cmd),
  do: {:error, :not_found}
```

```elixir
def execute(%Account{}, %DepositMoney{} = cmd) do
  [
    %MoneyDeposited{
      account_id: cmd.account_id,
      amount: cmd.amount
    },
    %JournalEntryCreated{
      journal_entry_uuid: Ecto.UUID.generate(),
      debit: %{"#{cmd.account_id}" => cmd.amount},
      credit: %{"000-000" => cmd.amount}
    }
  ]
end
```

Accounts Aggregate



lib/core/accounts/account.ex

```elixir
def execute(%Account{}, %DepositMoney{account_id: "000-000"}),
  do: {:error, :unable_to_create_account}

def execute(%Account{id: nil}, %DepositMoney{} = cmd) do
  [
    %AccountOpened{
      account_id: cmd.account_id
    },
    %MoneyDeposited{
      account_id: cmd.account_id,
      amount: cmd.amount
    },
    %JournalEntryCreated{
      journal_entry_uuid: Ecto.UUID.generate(),
      debit: %{"#{cmd.account_id}" => cmd.amount},
      credit: %{"000-000" => cmd.amount}
    }
  ]
end
```

```elixir
def execute(%Account{id: nil}, _cmd),
  do: {:error, :not_found}
```

```elixir
def execute(%Account{}, %DepositMoney{} = cmd) do
  [
    %MoneyDeposited{
      account_id: cmd.account_id,
      amount: cmd.amount
    },
    %JournalEntryCreated{
      journal_entry_uuid: Ecto.UUID.generate(),
      debit: %{"#{cmd.account_id}" => cmd.amount},
      credit: %{"000-000" => cmd.amount}
    }
  ]
end
```

Accounts Aggregate

lib/core/accounts/account.ex

```elixir
def execute(%Account{balance: balance}, %WithdrawMoney{amount: amount})
    when balance < amount do
  {:error, :insufficient_balance}
end
```

```elixir
def execute(%Account{}, %WithdrawMoney{} = cmd) do
  [
    %MoneyWithdrawn{
      account_id: cmd.account_id,
      amount: cmd.amount
    },
    %JournalEntryCreated{
      journal_entry_uuid: Ecto.UUID.generate(),
      debit: %{"000-000" => cmd.amount},
      credit: %{"#{cmd.account_id}" => cmd.amount}
    }
  ]
end
```

Accounts Aggregate

lib/core/accounts/account.ex

```
def execute(%Account{balance: balance}, %WithdrawMoney{amount: amount})
    when balance < amount do
  {:error, :insufficient_balance}
end
```

```
def execute(%Account{}, %WithdrawMoney{} = cmd) do
  [
    %MoneyWithdrawn{
      account_id: cmd.account_id,
      amount: cmd.amount
    },
    %JournalEntryCreated{
      journal_entry_uuid: Ecto.UUID.generate(),
      debit: %{"000-000" => cmd.amount},
      credit: %{"#{cmd.account_id}" => cmd.amount}
    }
  ]
end
```

Accounts Aggregate

lib/core/accounts/account.ex

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

def apply(state, %MoneyDeposited{} = evt) do
  %{state | balance: state.balance + evt.amount}
end

def apply(state, %MoneyWithdrawn{} = evt) do
  %{state | balance: state.balance - evt.amount}
end

def apply(state, %JournalEntryCreated{}), do: state
```

Accounts Context

Application Interface

lib/core/accounts.ex

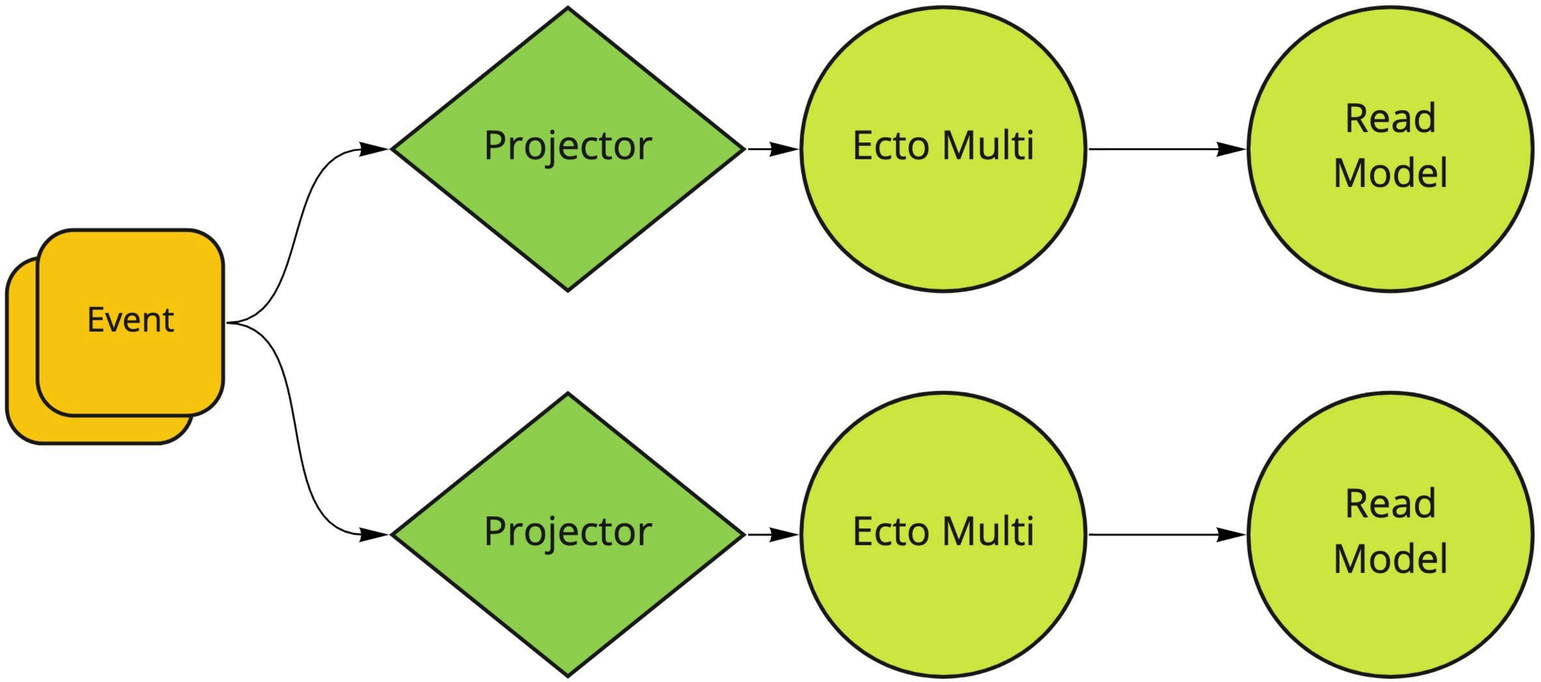
```elixir
@spec deposit_money(Bank.account_number(), Bank.amount()) ::
        {:ok, ExecutionResult.t()} | {:error, term()}
def deposit_money(acc_id, amount) do
  %DepositMoney{account_id: acc_id, amount: amount}
  |> Bank.Core.Application.dispatch(returning: :execution_result)
end

@spec withdraw_money(Bank.account_number(), Bank.amount()) ::
        {:ok, ExecutionResult.t()} | {:error, term()}
def withdraw_money(acc_id, amount) do
  %WithdrawMoney{account_id: acc_id, amount: amount}
  |> Bank.Core.Application.dispatch(returning: :execution_result)
end
```

View Balance

Projector

Processes events and executes an Ecto.Multi as it's side-effect in a single transaction.

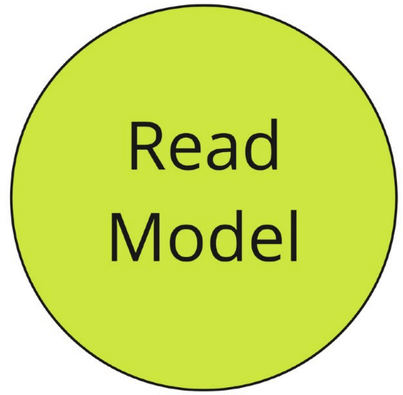
It helps us guarantee that events are handled only once.



Account Entry Schema

lib/core/accounting/account_entry.ex

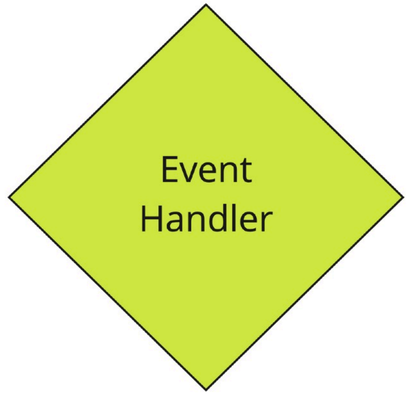
```elixir
defmodule Bank.Core.Accounting.AccountEntry do
  @moduledoc """
  This module defines the AccountEntry schema.
  Used to model single account entries.
  """

  use Ecto.Schema

  @type t() :: %{
          journal_entry_uuid: binary(),
          account: binary(),
          credit: integer(),
          debit: integer()
        }

  schema "accounting_account_entries_v1" do
    field :journal_entry_uuid, :binary_id
    field :account, :string

    field :credit, :integer, default: 0
    field :debit, :integer, default: 0
  end
end
```



Account Entry Projector

lib/core/accounting/account_entry_projector.ex

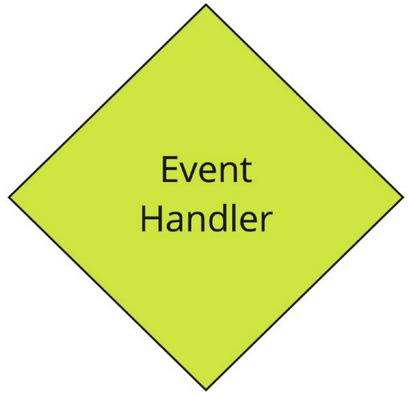
```elixir
defmodule Bank.Core.Accounting.AccountEntryProjector do
  use Commanded.Projections.Ecto,
    name: "Accounting.AccountEntriesProjector",
    application: Bank.Core.Application,
    consistency: :strong

  alias Bank.Core.Accounting.AccountEntry
  alias Bank.Core.Events.JournalEntryCreated

  project(
    %JournalEntryCreated{} = evt,
    _metadata,
    fn multi ->
      Ecto.Multi.insert_all(
        multi,
        :insert_account_entries,
        AccountEntry,
        from_journal_entry(evt)
      )
    end
  )

  @spec from_journal_entry(%JournalEntryCreated{}) :: [AccountEntry.t()]
  defp from_journal_entry(journal_entry) do
    ...
  end
end
```



Account Entry Projector

lib/core/accounting/account_entry_projector.ex

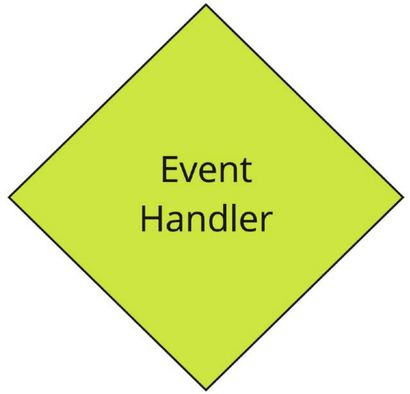
```elixir
defmodule Bank.Core.Accounting.AccountEntryProjector do
  use Commanded.Projections.Ecto,
    name: "Accounting.AccountEntriesProjector",
    application: Bank.Core.Application,
    consistency: :strong

  alias Bank.Core.Accounting.AccountEntry
  alias Bank.Core.Events.JournalEntryCreated

  project(
    %JournalEntryCreated{} = evt,
    _metadata,
    fn multi ->
      Ecto.Multi.insert_all(
        multi,
        :insert_account_entries,
        AccountEntry,
        from_journal_entry(evt)
      )
    end
  )

  @spec from_journal_entry(%JournalEntryCreated{}) :: [AccountEntry.t()]
  defp from_journal_entry(journal_entry) do
    ...
  end
end
```



Account Entry Projector

lib/core/accounting/account_entry_projector.ex

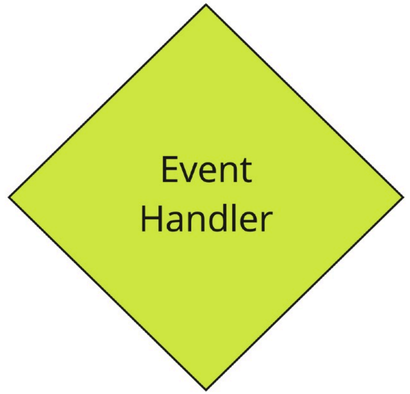
```elixir
defmodule Bank.Core.Accounting.AccountEntryProjector do
  use Commanded.Projections.Ecto,
    name: "Accounting.AccountEntriesProjector",
    application: Bank.Core.Application,
    consistency: :strong

  alias Bank.Core.Accounting.AccountEntry
  alias Bank.Core.Events.JournalEntryCreated

  project(
    %JournalEntryCreated{} = evt,
    _metadata,
    fn multi ->
      Ecto.Multi.insert_all(
        multi,
        :insert_account_entries,
        AccountEntry,
        from_journal_entry(evt)
      )
    end
  )

  @spec from_journal_entry(%JournalEntryCreated{}) :: [AccountEntry.t()]
  defp from_journal_entry(journal_entry) do
    ...
  end
end
```



Account Entry Projector

lib/core/accounting/account_entry_projector.ex

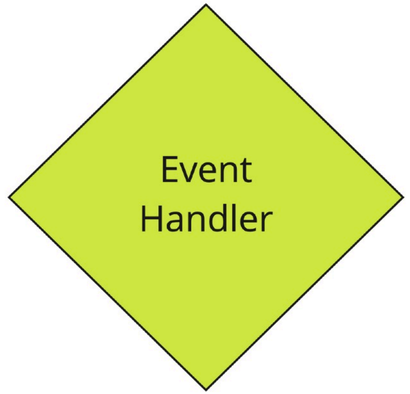
```elixir
defmodule Bank.Core.Accounting.AccountEntryProjector do
  use Commanded.Projections.Ecto,
    name: "Accounting.AccountEntriesProjector",
    application: Bank.Core.Application,
    consistency: :strong

  alias Bank.Core.Accounting.AccountEntry
  alias Bank.Core.Events.JournalEntryCreated

  project(
    %JournalEntryCreated{} = evt,
     metadata,
    fn multi ->
      Ecto.Multi.insert_all(
        multi,
        :insert_account_entries,
        AccountEntry,
        from_journal_entry(evt)
      )
    end
  )

  @spec from_journal_entry(%JournalEntryCreated{}) :: [AccountEntry.t()]
  defp from_journal_entry(journal_entry) do
    ...
  end
end
```



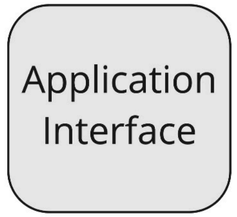
Account Entry Projector

lib/core/accounting/account_entry_projector.ex

```elixir
defmodule Bank.Core.Accounting.AccountEntryProjector do
  use Commanded.Projections.Ecto,
    name: "Accounting.AccountEntriesProjector",
    application: Bank.Core.Application,
    consistency: :strong

  alias Bank.Core.Accounting.AccountEntry
  alias Bank.Core.Events.JournalEntryCreated

  project(
    %JournalEntryCreated{} = evt,
    _metadata,
    fn multi ->
      Ecto.Multi.insert_all(
        multi,
        :insert_account_entries,
        AccountEntry,
        from_journal_entry(evt)
      )
    end
  )

  @spec from_journal_entry(%JournalEntryCreated{}) :: [AccountEntry.t()]
  defp from_journal_entry(journal_entry) do
    ...
  end
end
```



Getting the balance



lib/core/accounting.ex

```elixir
@spec view_balance(Bank.account_number()) :: Bank.amount()
def view_balance(account) do
  Repo.one(
    from e in AccountEntry,
      where: e.account == ^account,
      select: sum(e.debit) - sum(e.credit)
  )
end
```

Lets try it out!

```
iex(2)> Bank.Core.Accounts.deposit_money("111-101", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 100, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 3,
   events: [
     %Bank.Core.Events.AccountOpened{account_id: "111-101"},
     %Bank.Core.Events.MoneyDeposited{account_id: "111-101", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-101" => 100},
       journal_entry_uuid: "1f5e310c-b5b9-4e54-9468-b2c7431df27b"
     }
   ],
   metadata: %{}
 }}
```

```
iex(3)> Bank.Core.Accounts.deposit_money("111-101", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 200, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 5,
   events: [
     %Bank.Core.Events.MoneyDeposited{account_id: "111-101", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-101" => 100},
       journal_entry_uuid: "d1278006-c1f7-464c-9e27-8d2279b9277e"
     }
   ],
   metadata: %{}
 }}
```

```
iex(2)> Bank.Core.Accounts.deposit_money("111-101", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 100, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 3,
   events: [
     %Bank.Core.Events.AccountOpened{account_id: "111-101"},
     %Bank.Core.Events.MoneyDeposited{account_id: "111-101", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-101" => 100},
       journal_entry_uuid: "1f5e310c-b5b9-4e54-9468-b2c7431df27b"
     }
   ],
   metadata: %{}
 }}
iex(3)> Bank.Core.Accounts.deposit_money("111-101", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 200, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 5,
   events: [
     %Bank.Core.Events.MoneyDeposited{account_id: "111-101", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-101" => 100},
       journal_entry_uuid: "d1278006-c1f7-464c-9e27-8d2279b9277e"
     }
   ],
   metadata: %{}
 }}
```

```
iex(2)> Bank.Core.Accounts.deposit_money("111-101", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 100, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 3,
   events: [
     %Bank.Core.Events.AccountOpened{account_id: "111-101"},
     %Bank.Core.Events.MoneyDeposited{account_id: "111-101", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-101" => 100},
       journal_entry_uuid: "1f5e310c-b5b9-4e54-9468-b2c7431df27b"
     }
   ],
   metadata: %{}
 }}
iex(3)> Bank.Core.Accounts.deposit_money("111-101", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 200, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 5,
   events: [
     %Bank.Core.Events.MoneyDeposited{account_id: "111-101", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-101" => 100},
       journal_entry_uuid: "d1278006-c1f7-464c-9e27-8d2279b9277e"
     }
   ],
   metadata: %{}
 }}
```

```
iex(2)> Bank.Core.Accounts.deposit_money("111-101", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 100, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 3,
   events: [
     %Bank.Core.Events.AccountOpened{account_id: "111-101"},
     %Bank.Core.Events.MoneyDeposited{account_id: "111-101", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-101" => 100},
       journal_entry_uuid: "1f5e310c-b5b9-4e54-9468-b2c7431df27b"
     }
   ],
   metadata: %{}
 }}
```

```
iex(3)> Bank.Core.Accounts.deposit_money("111-101", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 200, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 5,
   events: [
     %Bank.Core.Events.MoneyDeposited{account_id: "111-101", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-101" => 100},
       journal_entry_uuid: "d1278006-c1f7-464c-9e27-8d2279b9277e"
     }
   ],
   metadata: %{}
 }}
```

```
iex(2)> Bank.Core.Accounts.deposit_money("111-101", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 100, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 3,
   events: [
     %Bank.Core.Events.AccountOpened{account_id: "111-101"},
     %Bank.Core.Events.MoneyDeposited{account_id: "111-101", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-101" => 100},
       journal_entry_uuid: "1f5e310c-b5b9-4e54-9468-b2c7431df27b"
     }
   ],
   metadata: %{}
 }}
iex(3)> Bank.Core.Accounts.deposit_money("111-101", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 200, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 5,
   events: [
     %Bank.Core.Events.MoneyDeposited{account_id: "111-101", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-101" => 100},
       journal_entry_uuid: "d1278006-c1f7-464c-9e27-8d2279b9277e"
     }
   ],
   metadata: %{}
 }}
```

```
iex(2)> Bank.Core.Accounts.deposit_money("111-101", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 100, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 3,
   events: [
     %Bank.Core.Events.AccountOpened{account_id: "111-101"},
     %Bank.Core.Events.MoneyDeposited{account_id: "111-101", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-101" => 100},
       journal_entry_uuid: "1f5e310c-b5b9-4e54-9468-b2c7431df27b"
     }
   ],
   metadata: %{}
 }}
iex(3)> Bank.Core.Accounts.deposit_money("111-101", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 200, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 5,
   events: [
     %Bank.Core.Events.MoneyDeposited{account_id: "111-101", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-101" => 100},
       journal_entry_uuid: "d1278006-c1f7-464c-9e27-8d2279b9277e"
     }
   ],
   metadata: %{}
 }}
```

```
iex(4)> Bank.Core.Accounts.withdraw_money("111-101", 200)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 0, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 7,
   events: [
     %Bank.Core.Events.MoneyWithdrawn{account_id: "111-101", amount: 200},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"111-101" => 200},
       debit: %{"000-000" => 200},
       journal_entry_uuid: "184f60e4-3cb4-4c96-87db-74c5a536820c"
     }
   ],
   metadata: %{}
 }}
iex(5)> Bank.Core.Accounts.withdraw_money("111-101", 100)
{:error, :insufficient_balance}
iex(6)> Bank.Core.Accounts.withdraw_money("111-102", 100)
{:error, :not_found}
iex(5)> Bank.Core.Accounts.view_balance("111-101")
0
```

```
iex(4)> Bank.Core.Accounts.withdraw_money("111-101", 200)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 0, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 7,
   events: [
     %Bank.Core.Events.MoneyWithdrawn{account_id: "111-101", amount: 200},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"111-101" => 200},
       debit: %{"000-000" => 200},
       journal_entry_uuid: "184f60e4-3cb4-4c96-87db-74c5a536820c"
     }
   ],
   metadata: %{}
 }}
iex(5)> Bank.Core.Accounts.withdraw_money("111-101", 100)
{:error, :insufficient_balance}
iex(6)> Bank.Core.Accounts.withdraw_money("111-102", 100)
{:error, :not_found}
iex(5)> Bank.Core.Accounts.view_balance("111-101")
0
```

```
iex(4)> Bank.Core.Accounts.withdraw_money("111-101", 200)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 0, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 7,
   events: [
     %Bank.Core.Events.MoneyWithdrawn{account_id: "111-101", amount: 200},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"111-101" => 200},
       debit: %{"000-000" => 200},
       journal_entry_uuid: "184f60e4-3cb4-4c96-87db-74c5a536820c"
     }
   ],
   metadata: %{}
 }}
iex(5)> Bank.Core.Accounts.withdraw_money("111-101", 100)
{:error, :insufficient_balance}
iex(6)> Bank.Core.Accounts.withdraw_money("111-102", 100)
{:error, :not_found}
iex(5)> Bank.Core.Accounts.view_balance("111-101")
0
```

```
iex(4)> Bank.Core.Accounts.withdraw_money("111-101", 200)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 0, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 7,
   events: [
     %Bank.Core.Events.MoneyWithdrawn{account_id: "111-101", amount: 200},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"111-101" => 200},
       debit: %{"000-000" => 200},
       journal_entry_uuid: "184f60e4-3cb4-4c96-87db-74c5a536820c"
     }
   ],
   metadata: %{}
 }}
iex(5)> Bank.Core.Accounts.withdraw_money("111-101", 100)
{:error, :insufficient_balance}
iex(6)> Bank.Core.Accounts.withdraw_money("111-102", 100)
{:error, :not_found}
iex(5)> Bank.Core.Accounts.view_balance("111-101")
0
```

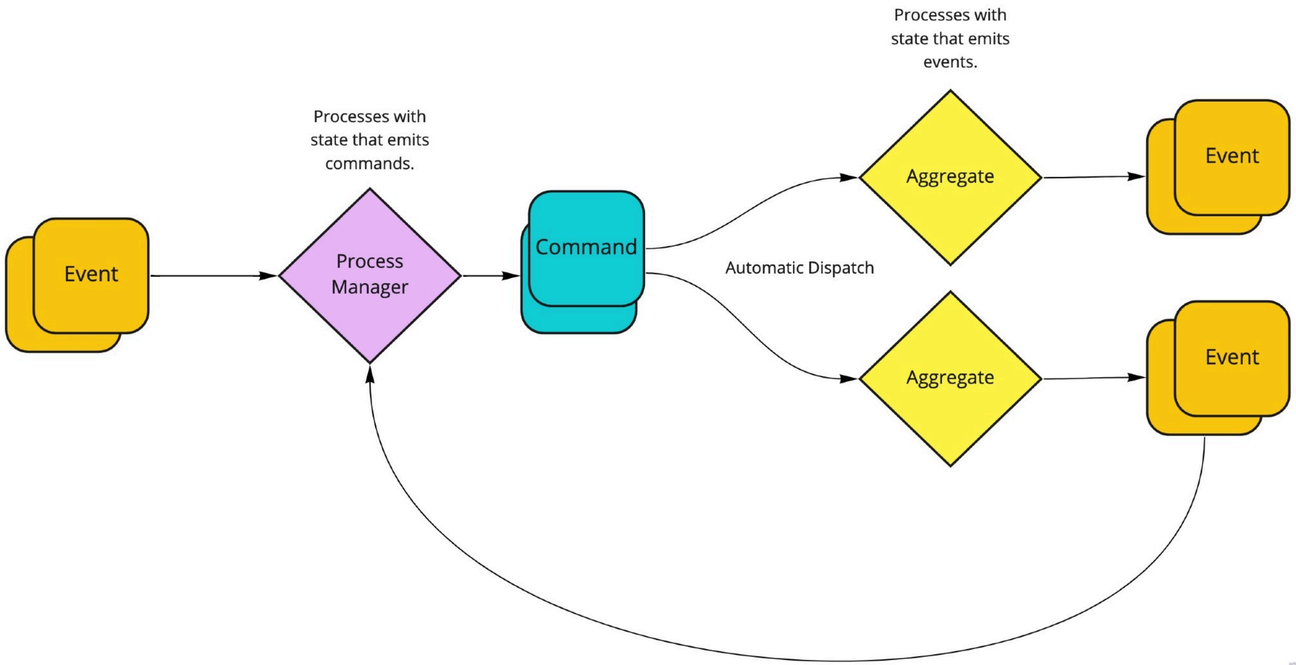
```
iex(4)> Bank.Core.Accounts.withdraw_money("111-101", 200)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 0, id: "111-101"},
   aggregate_uuid: "111-101",
   aggregate_version: 7,
   events: [
     %Bank.Core.Events.MoneyWithdrawn{account_id: "111-101", amount: 200},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"111-101" => 200},
       debit: %{"000-000" => 200},
       journal_entry_uuid: "184f60e4-3cb4-4c96-87db-74c5a536820c"
     }
   ],
   metadata: %{}
 }}
iex(5)> Bank.Core.Accounts.withdraw_money("111-101", 100)
{:error, :insufficient_balance}
iex(6)> Bank.Core.Accounts.withdraw_money("111-102", 100)
{:error, :not_found}
iex(5)> Bank.Core.Accounts.view_balance("111-101")
0
```

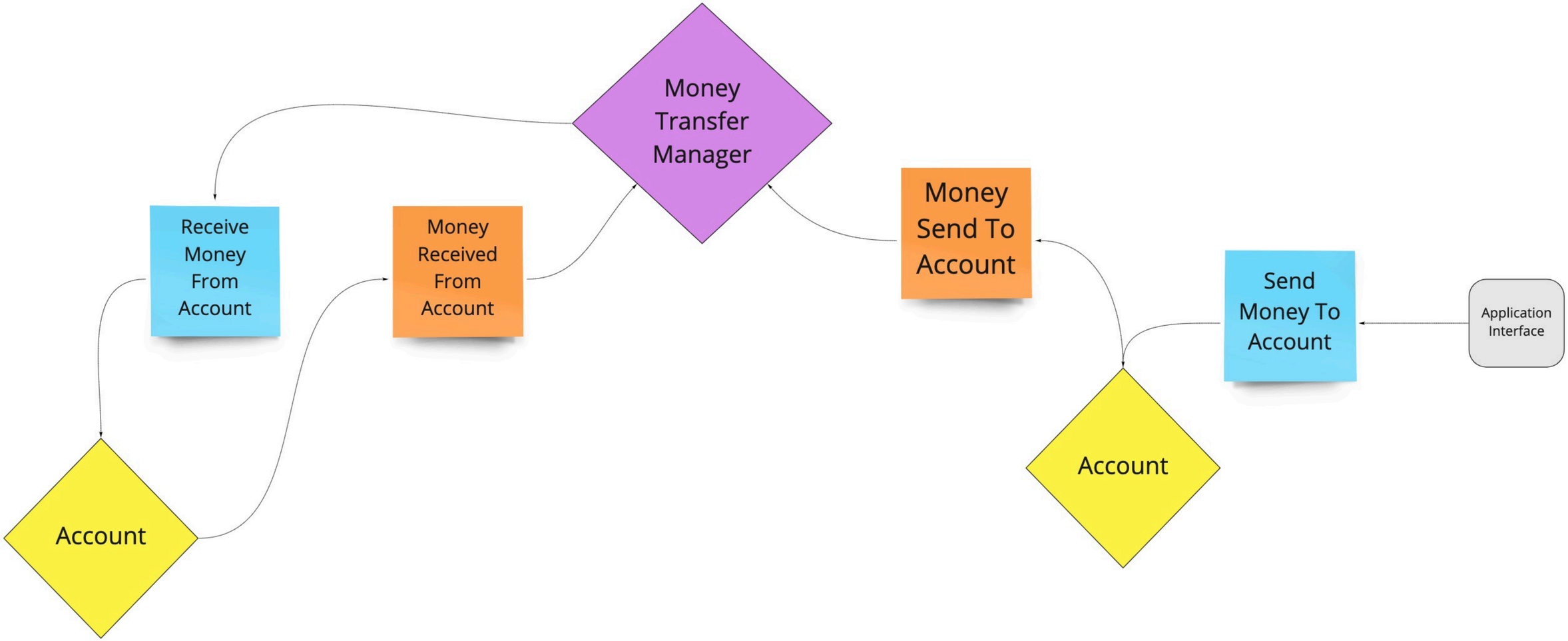
Transferring Money

Process Managers

An actor that updates it’s state from a streams of events.
It creates commands from events that occurred and it’s state.

Multiple processes separated by unique identifier.

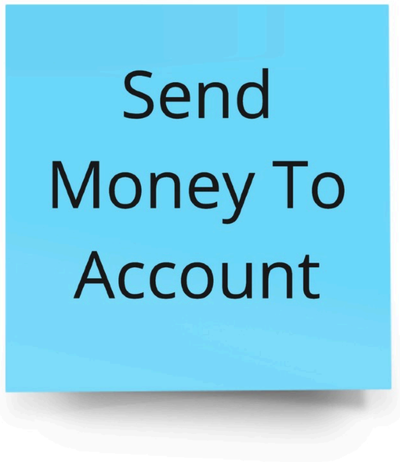



Money Transfer Manager
Receive Money From Account
Money Received From Account
Money Send To Account
Send Money To Account
Application Interface
Account
Account

Send To Account

lib/core/commands/send_money_to_account.ex

```
defmodule Bank.Core.Commands.SendMoneyToAccount do
  @type t :: %__MODULE__{
          from_account_id: Bank.account_number(),
          to_account_id: Bank.account_number(),
          amount: Bank.amount()
        }

  defstruct [
    :from_account_id,
    :to_account_id,
    :amount
  ]
end
```



Send To Account

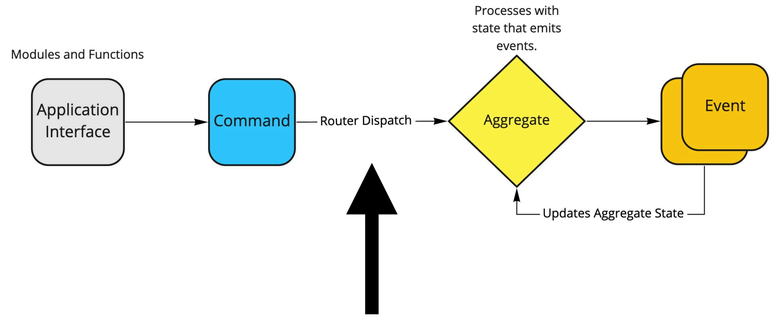
lib/core/events/money_sent_to_account.ex

```elixir
defmodule Bank.Core.Events.MoneySentToAccount do
  @type t :: %__MODULE__{
          transaction_id: binary(),
          from_account_id: Bank.account_number(),
          to_account_id: Bank.account_number(),
          amount: Bank.amount()
        }

  defstruct [
    :transaction_id,
    :from_account_id,
    :to_account_id,
    :amount
  ]
end
```

Send To Account

lib/core/router.ex

```
defmodule Bank.Core.Router do

  ...

  dispatch(
    [Commands.SendMoneyToAccount],
    to: Bank.Core.Accounts.Account,
    identity: :from_account_id
  )
end
```

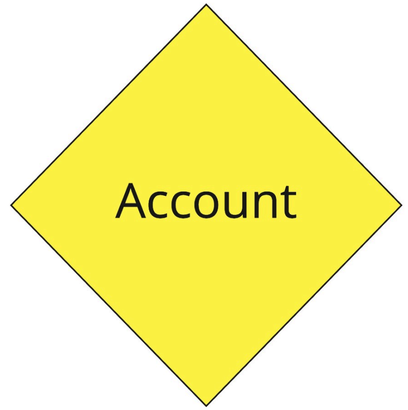


Sending Money

lib/core/accounts/account.ex

```
def execute(%Account{balance: balance}, %SendMoneyToAccount{amount: amount})
    when balance - amount < 0 do
  {:error, :insufficient_balance}
end

def execute(%Account{} = state, %SendMoneyToAccount{} = cmd) do
  transaction_id = Ecto.UUID.generate()

  [
    %MoneySentToAccount{
      transaction_id: transaction_id,
      from_account_id: state.id,
      to_account_id: cmd.to_account_id,
      amount: cmd.amount
    },
    %JournalEntryCreated{
      journal_entry_uuid: Ecto.UUID.generate(),
      debit: %{"#{state.id}" => cmd.amount},
      credit: %{"#{transaction_id}" => cmd.amount}
    }
  ]
end
```

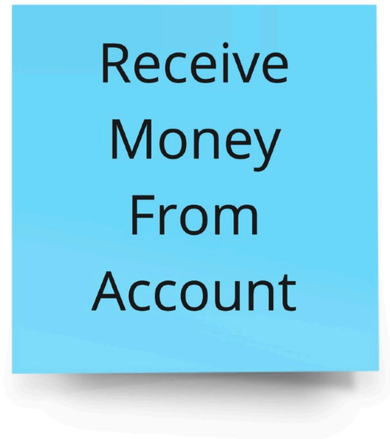
```
def apply(state, %MoneySentToAccount{} = evt) do
  %{state | balance: state.balance - evt.amount}
end
```



Receive from Account

lib/core/commands/receive_money_from_account.ex

```elixir
defmodule Bank.Core.Commands.ReceiveMoneyFromAccount do
  @type t :: %__MODULE__{
          transaction_id: binary(),
          from_account_id: Bank.account_number(),
          to_account_id: Bank.account_number(),
          amount: Bank.amount()
        }

  defstruct [
    :transaction_id,
    :from_account_id,
    :to_account_id,
    :amount
  ]
end
```



Receive from Account

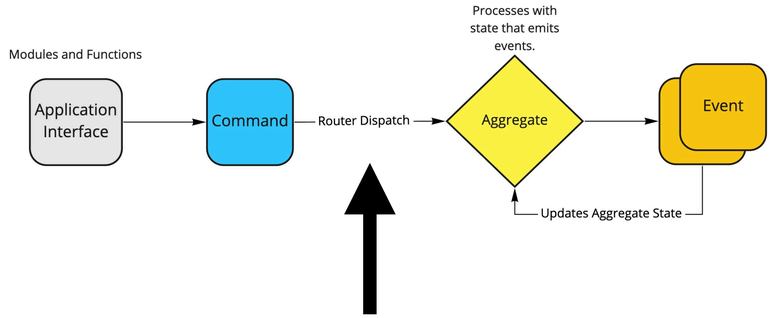
lib/core/events/money_received_from.ex

```
defmodule Bank.Core.Events.MoneyReceivedFromAccount do
  @type t :: %__MODULE__{
          transaction_id: binary(),
          from_account_id: Bank.account_number(),
          to_account_id: Bank.account_number(),
          amount: Bank.amount()
        }

  defstruct [
    :transaction_id,
    :from_account_id,
    :to_account_id,
    :amount
  ]
end
```

Receive from Account

lib/core/router.ex

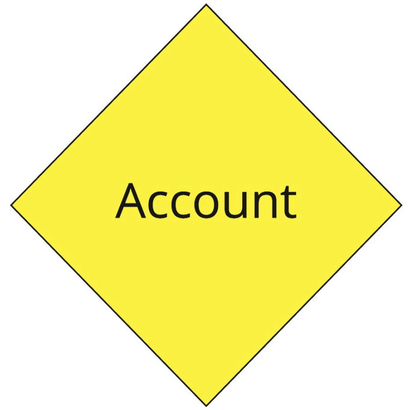
```
defmodule Bank.Core.Router do

  ...

  dispatch(
    [Commands.ReceiveMoneyFromAccount],
    to: Bank.Core.Accounts.Account,
    identity: :to_account_id
  )
end
```



Receiving Money

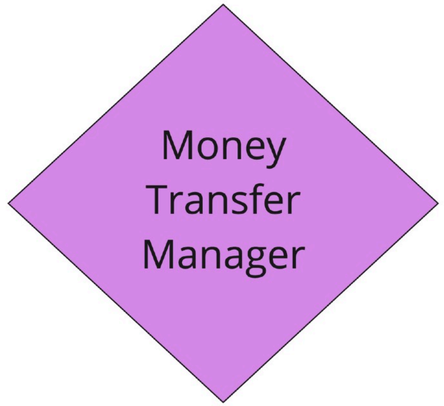
lib/core/accounts/account.ex

```elixir
def execute(%Account{} = state, %ReceiveMoneyFromAccount{} = cmd) do
  [
    %MoneyReceivedFromAccount{
      transaction_id: cmd.transaction_id,
      from_account_id: cmd.from_account_id,
      to_account_id: state.id,
      amount: cmd.amount
    },
    %JournalEntryCreated{
      journal_entry_uuid: Ecto.UUID.generate(),
      credit: %{"#{cmd.transaction_id}" => cmd.amount},
      debit: %{"#{cmd.to_account_id}" => cmd.amount}
    }
  ]
end

def apply(state, %MoneyReceivedFromAccount{} = evt) do
  %{state | balance: state.balance + evt.amount}
end
```



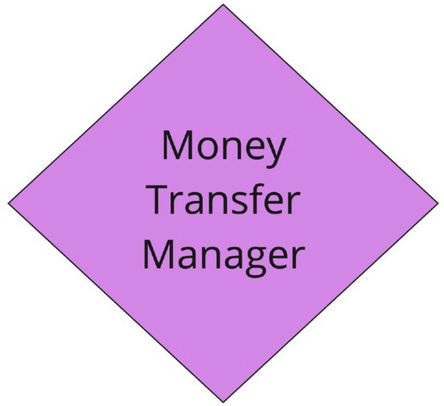
lib/core/accounts/money_transfer_process_manager.ex

```elixir
defmodule Bank.Core.Accounts.MoneyTransferProcessManager do
  use Commanded.ProcessManagers.ProcessManager,
    name: "Bank.Core.Accounts.MoneyTransferProcessManager",
    application: Bank.Core.Application

  alias Bank.Core.Commands.{ReceiveMoneyFromAccount}
  alias Bank.Core.Events.{MoneySentToAccount, MoneyReceivedFromAccount}

  defstruct [:transaction_id]

  def interested?(%MoneySentToAccount{transaction_id: id}), do: {:start, id}
  def interested?(%MoneyReceivedFromAccount{transaction_id: id}), do: {:stop, id}

  def handle(%__MODULE__{}, %MoneySentToAccount{} = evt),
    do: [
      %ReceiveMoneyFromAccount{
        transaction_id: evt.transaction_id,
        from_account_id: evt.from_account_id,
        to_account_id: evt.to_account_id,
        amount: evt.amount
      }
    ]

  def apply(_state, %MoneySentToAccount{} = evt) do
    %__MODULE__{transaction_id: evt.transaction_id}
  end
end
```



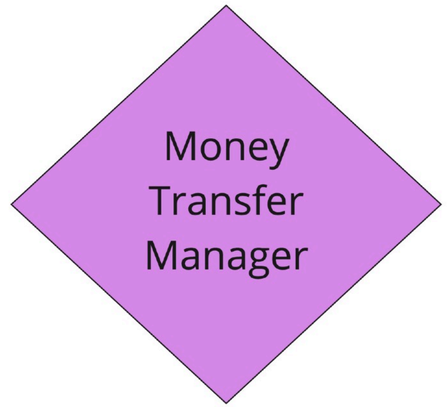
lib/core/accounts/money_transfer_process_manager.ex

```elixir
defmodule Bank.Core.Accounts.MoneyTransferProcessManager do
  use Commanded.ProcessManagers.ProcessManager,
    name: "Bank.Core.Accounts.MoneyTransferProcessManager",
    application: Bank.Core.Application

  alias Bank.Core.Commands.{ReceiveMoneyFromAccount}
  alias Bank.Core.Events.{MoneySentToAccount, MoneyReceivedFromAccount}

  defstruct [:transaction_id]

  def interested?(%MoneySentToAccount{transaction_id: id}), do: {:start, id}
  def interested?(%MoneyReceivedFromAccount{transaction_id: id}), do: {:stop, id}

  def handle(%__MODULE__{}, %MoneySentToAccount{} = evt),
    do: [
      %ReceiveMoneyFromAccount{
        transaction_id: evt.transaction_id,
        from_account_id: evt.from_account_id,
        to_account_id: evt.to_account_id,
        amount: evt.amount
      }
    ]

  def apply(_state, %MoneySentToAccount{} = evt) do
    %__MODULE__{transaction_id: evt.transaction_id}
  end
end
```



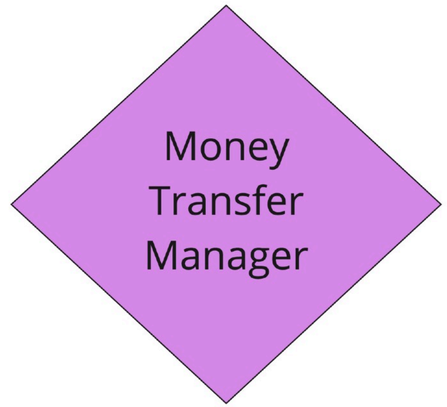
lib/core/accounts/money_transfer_process_manager.ex

```elixir
defmodule Bank.Core.Accounts.MoneyTransferProcessManager do
  use Commanded.ProcessManagers.ProcessManager,
    name: "Bank.Core.Accounts.MoneyTransferProcessManager",
    application: Bank.Core.Application

  alias Bank.Core.Commands.{ReceiveMoneyFromAccount}
  alias Bank.Core.Events.{MoneySentToAccount, MoneyReceivedFromAccount}

  defstruct [:transaction_id]

  def interested?(%MoneySentToAccount{transaction_id: id}), do: {:start, id}
  def interested?(%MoneyReceivedFromAccount{transaction_id: id}), do: {:stop, id}

  def handle(%__MODULE__{}, %MoneySentToAccount{} = evt),
    do: [
      %ReceiveMoneyFromAccount{
        transaction_id: evt.transaction_id,
        from_account_id: evt.from_account_id,
        to_account_id: evt.to_account_id,
        amount: evt.amount
      }
    ]

  def apply(_state, %MoneySentToAccount{} = evt) do
    %__MODULE__{transaction_id: evt.transaction_id}
  end
end
```



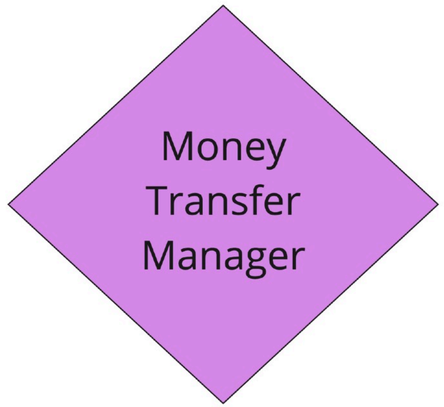
lib/core/accounts/money_transfer_process_manager.ex

```elixir
defmodule Bank.Core.Accounts.MoneyTransferProcessManager do
  use Commanded.ProcessManagers.ProcessManager,
    name: "Bank.Core.Accounts.MoneyTransferProcessManager",
    application: Bank.Core.Application

  alias Bank.Core.Commands.{ReceiveMoneyFromAccount}
  alias Bank.Core.Events.{MoneySentToAccount, MoneyReceivedFromAccount}

  defstruct [:transaction_id]

  def interested?(%MoneySentToAccount{transaction_id: id}), do: {:start, id}
  def interested?(%MoneyReceivedFromAccount{transaction_id: id}), do: {:stop, id}

  def handle(%__MODULE__{}, %MoneySentToAccount{} = evt),
    do: [
      %ReceiveMoneyFromAccount{
        transaction_id: evt.transaction_id,
        from_account_id: evt.from_account_id,
        to_account_id: evt.to_account_id,
        amount: evt.amount
      }
    ]

  def apply(_state, %MoneySentToAccount{} = evt) do
    %__MODULE__{transaction_id: evt.transaction_id}
  end
end
```



lib/core/accounts/money_transfer_process_manager.ex

```elixir
defmodule Bank.Core.Accounts.MoneyTransferProcessManager do
  use Commanded.ProcessManagers.ProcessManager,
    name: "Bank.Core.Accounts.MoneyTransferProcessManager",
    application: Bank.Core.Application

  alias Bank.Core.Commands.{ReceiveMoneyFromAccount}
  alias Bank.Core.Events.{MoneySentToAccount, MoneyReceivedFromAccount}

  defstruct [:transaction_id]

  def interested?(%MoneySentToAccount{transaction_id: id}), do: {:start, id}
  def interested?(%MoneyReceivedFromAccount{transaction_id: id}), do: {:stop, id}

  def handle(%__MODULE__{}, %MoneySentToAccount{} = evt),
    do: [
      %ReceiveMoneyFromAccount{
        transaction_id: evt.transaction_id,
        from_account_id: evt.from_account_id,
        to_account_id: evt.to_account_id,
        amount: evt.amount
      }
    ]

  def apply(_state, %MoneySentToAccount{} = evt) do
    %__MODULE__{transaction_id: evt.transaction_id}
  end
end
```



```
iex(4)> Bank.Core.Accounts.deposit_money("111-102", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 100, id: "111-102"},
   aggregate_uuid: "111-102",
   aggregate_version: 3,
   events: [
     %Bank.Core.Events.AccountOpened{account_id: "111-102"},
     %Bank.Core.Events.MoneyDeposited{account_id: "111-102", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-102" => 100},
       journal_entry_uuid: "e66f12e8-393f-4675-bbba-90ba8a870cac"
     }
   ],
   metadata: %{}
 }}
iex(5)> Bank.Core.Accounts.deposit_money("111-103", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 100, id: "111-103"},
   aggregate_uuid: "111-103",
   aggregate_version: 3,
   events: [
     %Bank.Core.Events.AccountOpened{account_id: "111-103"},
     %Bank.Core.Events.MoneyDeposited{account_id: "111-103", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-103" => 100},
       journal_entry_uuid: "9d43a32d-fa39-4a88-a83c-2770ccf189df"
     }
   ],
   metadata: %{}
 }}
iex(6)> Bank.Core.Accounts.view_balance("111-102")
100
iex(7)> Bank.Core.Accounts.view_balance("111-103")
100
```

```
iex(4)> Bank.Core.Accounts.deposit_money("111-102", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 100, id: "111-102"},
   aggregate_uuid: "111-102",
   aggregate_version: 3,
   events: [
     %Bank.Core.Events.AccountOpened{account_id: "111-102"},
     %Bank.Core.Events.MoneyDeposited{account_id: "111-102", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-102" => 100},
       journal_entry_uuid: "e66f12e8-393f-4675-bbba-90ba8a870cac"
     }
   ],
   metadata: %{}
 }}
iex(5)> Bank.Core.Accounts.deposit_money("111-103", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 100, id: "111-103"},
   aggregate_uuid: "111-103",
   aggregate_version: 3,
   events: [
     %Bank.Core.Events.AccountOpened{account_id: "111-103"},
     %Bank.Core.Events.MoneyDeposited{account_id: "111-103", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-103" => 100},
       journal_entry_uuid: "9d43a32d-fa39-4a88-a83c-2770ccf189df"
     }
   ],
   metadata: %{}
 }}
iex(6)> Bank.Core.Accounts.view_balance("111-102")
100
iex(7)> Bank.Core.Accounts.view_balance("111-103")
100
```

```
iex(4)> Bank.Core.Accounts.deposit_money("111-102", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 100, id: "111-102"},
   aggregate_uuid: "111-102",
   aggregate_version: 3,
   events: [
     %Bank.Core.Events.AccountOpened{account_id: "111-102"},
     %Bank.Core.Events.MoneyDeposited{account_id: "111-102", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-102" => 100},
       journal_entry_uuid: "e66f12e8-393f-4675-bbba-90ba8a870cac"
     }
   ],
   metadata: %{}
 }}
iex(5)> Bank.Core.Accounts.deposit_money("111-103", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 100, id: "111-103"},
   aggregate_uuid: "111-103",
   aggregate_version: 3,
   events: [
     %Bank.Core.Events.AccountOpened{account_id: "111-103"},
     %Bank.Core.Events.MoneyDeposited{account_id: "111-103", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-103" => 100},
       journal_entry_uuid: "9d43a32d-fa39-4a88-a83c-2770ccf189df"
     }
   ],
   metadata: %{}
 }}
iex(6)> Bank.Core.Accounts.view_balance("111-102")
100
iex(7)> Bank.Core.Accounts.view_balance("111-103")
100
```

```
iex(4)> Bank.Core.Accounts.deposit_money("111-102", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 100, id: "111-102"},
   aggregate_uuid: "111-102",
   aggregate_version: 3,
   events: [
     %Bank.Core.Events.AccountOpened{account_id: "111-102"},
     %Bank.Core.Events.MoneyDeposited{account_id: "111-102", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-102" => 100},
       journal_entry_uuid: "e66f12e8-393f-4675-bbba-90ba8a870cac"
     }
   ],
   metadata: %{}
 }}
iex(5)> Bank.Core.Accounts.deposit_money("111-103", 100)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 100, id: "111-103"},
   aggregate_uuid: "111-103",
   aggregate_version: 3,
   events: [
     %Bank.Core.Events.AccountOpened{account_id: "111-103"},
     %Bank.Core.Events.MoneyDeposited{account_id: "111-103", amount: 100},
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"000-000" => 100},
       debit: %{"111-103" => 100},
       journal_entry_uuid: "9d43a32d-fa39-4a88-a83c-2770ccf189df"
     }
   ],
   metadata: %{}
 }}
iex(6)> Bank.Core.Accounts.view_balance("111-102")
100
iex(7)> Bank.Core.Accounts.view_balance("111-103")
100
```

```
iex(8)> Bank.Core.Accounts.send_money("111-102", "111-103", 50)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 50, id: "111-102"},
   aggregate_uuid: "111-102",
   aggregate_version: 5,
   events: [
     %Bank.Core.Events.MoneySentToAccount{
       amount: 50,
       from_account_id: "111-102",
       to_account_id: "111-103",
       transaction_id: "05c52667-70e6-4e0e-b46e-e6d5dd8aa21c"
     },
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"111-102" => 50},
       debit: %{"05c52667-70e6-4e0e-b46e-e6d5dd8aa21c" => 50},
       journal_entry_uuid: "c0681ab9-cb57-4c6f-8584-d05f059c5d1c"
     }
   ],
   metadata: %{}
 }}
iex(9)> Bank.Core.Accounts.view_balance("111-102")
50
iex(10)> Bank.Core.Accounts.view_balance("111-103")
150
```

```
iex(8)> Bank.Core.Accounts.send_money("111-102", "111-103", 50)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 50, id: "111-102"},
   aggregate_uuid: "111-102",
   aggregate_version: 5,
   events: [
     %Bank.Core.Events.MoneySentToAccount{
       amount: 50,
       from_account_id: "111-102",
       to_account_id: "111-103",
       transaction_id: "05c52667-70e6-4e0e-b46e-e6d5dd8aa21c"
     },
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"111-102" => 50},
       debit: %{"05c52667-70e6-4e0e-b46e-e6d5dd8aa21c" => 50},
       journal_entry_uuid: "c0681ab9-cb57-4c6f-8584-d05f059c5d1c"
     }
   ],
   metadata: %{}
 }}
iex(9)> Bank.Core.Accounts.view_balance("111-102")
50
iex(10)> Bank.Core.Accounts.view_balance("111-103")
150
```

```
iex(8)> Bank.Core.Accounts.send_money("111-102", "111-103", 50)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 50, id: "111-102"},
   aggregate_uuid: "111-102",
   aggregate_version: 5,
   events: [
     %Bank.Core.Events.MoneySentToAccount{
       amount: 50,
       from_account_id: "111-102",
       to_account_id: "111-103",
       transaction_id: "05c52667-70e6-4e0e-b46e-e6d5dd8aa21c"
     },
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"111-102" => 50},
       debit: %{"05c52667-70e6-4e0e-b46e-e6d5dd8aa21c" => 50},
       journal_entry_uuid: "c0681ab9-cb57-4c6f-8584-d05f059c5d1c"
     }
   ],
   metadata: %{}
 }}
iex(9)> Bank.Core.Accounts.view_balance("111-102")
50
iex(10)> Bank.Core.Accounts.view_balance("111-103")
150
```

```
iex(4)> Bank.Core.EventStore.stream_all_forward() |> Enum.take(-4) |> Enum.map(& &1.data)
[
  %Bank.Core.Events.MoneySentToAccount{
    amount: 50,
    from_account_id: "111-102",
    to_account_id: "111-103",
    transaction_id: "4f7bee90-29af-4ebd-9c56-549187a6daa1"
  },
  %Bank.Core.Events.JournalEntryCreated{
    credit: %{"111-102" => 50},
    debit: %{"4f7bee90-29af-4ebd-9c56-549187a6daa1" => 50},
    journal_entry_uuid: "a0d8092d-fc02-433e-a526-fa1d3b272934"
  },
  %Bank.Core.Events.MoneyReceivedFromAccount{
    amount: 50,
    from_account_id: "111-102",
    to_account_id: "111-103",
    transaction_id: "4f7bee90-29af-4ebd-9c56-549187a6daa1"
  },
  %Bank.Core.Events.JournalEntryCreated{
    credit: %{"4f7bee90-29af-4ebd-9c56-549187a6daa1" => 50},
    debit: %{"111-103" => 50},
    journal_entry_uuid: "1c92ce05-1f49-4def-af14-76fd4385acf7"
  }
]
```

```
iex(4)> Bank.Core.EventStore.stream_all_forward() |> Enum.take(-4) |> Enum.map(& &1.data)
[
  %Bank.Core.Events.MoneySentToAccount{
    amount: 50,
    from_account_id: "111-102",
    to_account_id: "111-103",
    transaction_id: "4f7bee90-29af-4ebd-9c56-549187a6daa1"
  },
  %Bank.Core.Events.JournalEntryCreated{
    credit: %{"111-102" => 50},
    debit: %{"4f7bee90-29af-4ebd-9c56-549187a6daa1" => 50},
    journal_entry_uuid: "a0d8092d-fc02-433e-a526-fa1d3b272934"
  },
  %Bank.Core.Events.MoneyReceivedFromAccount{
    amount: 50,
    from_account_id: "111-102",
    to_account_id: "111-103",
    transaction_id: "4f7bee90-29af-4ebd-9c56-549187a6daa1"
  },
  %Bank.Core.Events.JournalEntryCreated{
    credit: %{"4f7bee90-29af-4ebd-9c56-549187a6daa1" => 50},
    debit: %{"111-103" => 50},
    journal_entry_uuid: "1c92ce05-1f49-4def-af14-76fd4385acf7"
  }
]
```

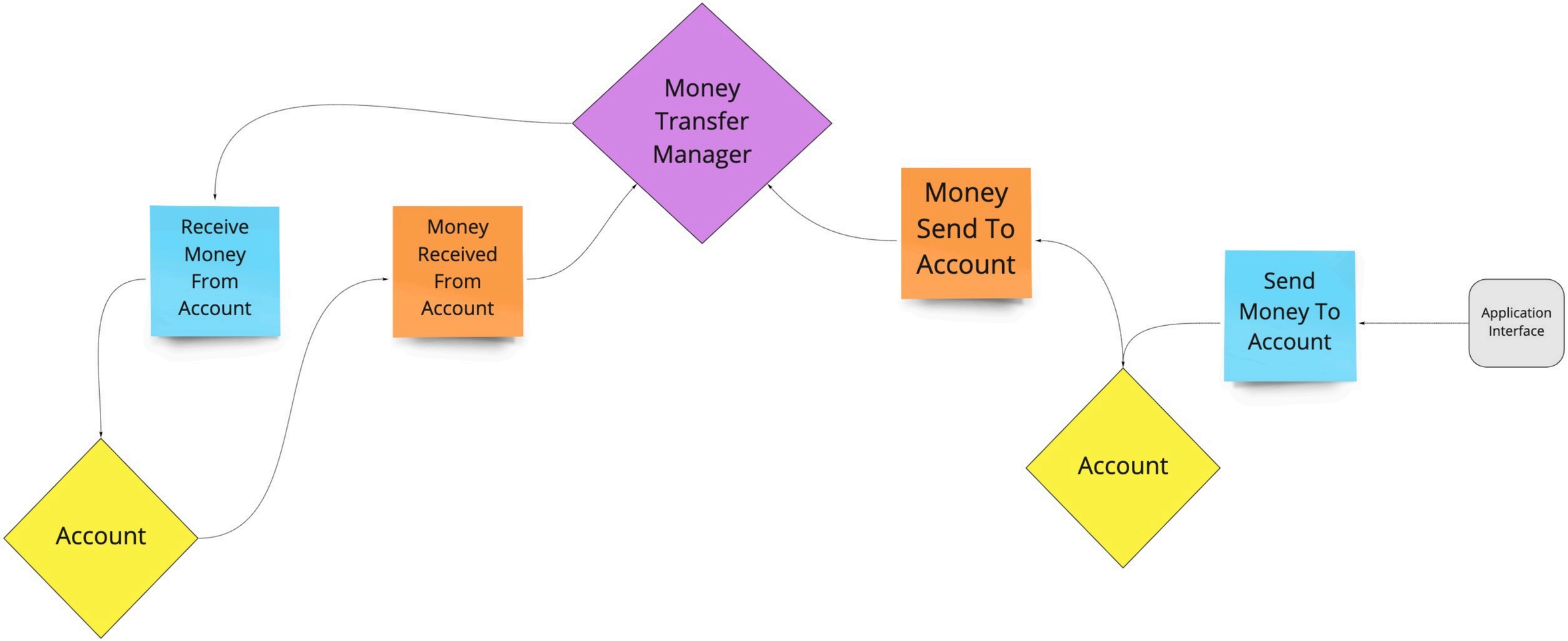
```
iex(4)> Bank.Core.EventStore.stream_all_forward() |> Enum.take(-4) |> Enum.map(& &1.data)
[
  %Bank.Core.Events.MoneySentToAccount{
    amount: 50,
    from_account_id: "111-102",
    to_account_id: "111-103",
    transaction_id: "4f7bee90-29af-4ebd-9c56-549187a6daa1"
  },
  %Bank.Core.Events.JournalEntryCreated{
    credit: %{"111-102" => 50},
    debit: %{"4f7bee90-29af-4ebd-9c56-549187a6daa1" => 50},
    journal_entry_uuid: "a0d8092d-fc02-433e-a526-fa1d3b272934"
  },
  %Bank.Core.Events.MoneyReceivedFromAccount{
    amount: 50,
    from_account_id: "111-102",
    to_account_id: "111-103",
    transaction_id: "4f7bee90-29af-4ebd-9c56-549187a6daa1"
  },
  %Bank.Core.Events.JournalEntryCreated{
    credit: %{"4f7bee90-29af-4ebd-9c56-549187a6daa1" => 50},
    debit: %{"111-103" => 50},
    journal_entry_uuid: "1c92ce05-1f49-4def-af14-76fd4385acf7"
  }
]
```

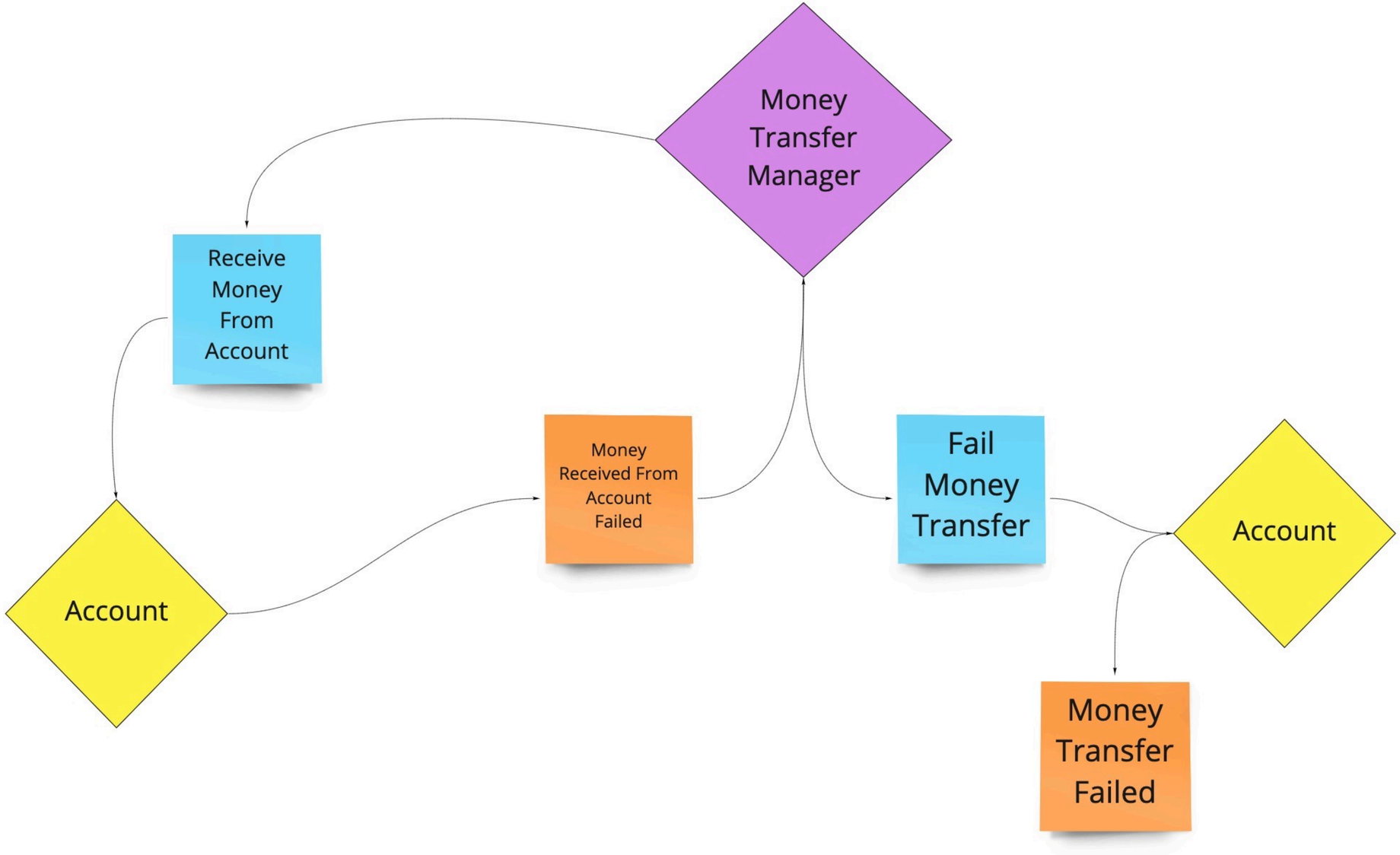
```
iex(10)> Bank.Core.Accounts.send_money("111-102", "111-104", 50)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 0, id: "111-102"},
   aggregate_uuid: "111-102",
   aggregate_version: 7,
   events: [
     %Bank.Core.Events.MoneySentToAccount{
       amount: 50,
       from_account_id: "111-102",
       to_account_id: "111-104",
       transaction_id: "7096c01c-95be-4a5b-a7a3-c1cb42ad0e69"
     },
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"111-102" => 50},
       debit: %{"7096c01c-95be-4a5b-a7a3-c1cb42ad0e69" => 50},
       journal_entry_uuid: "569bb403-daa7-4c9a-bad7-85f2fe396e68"
     }
   ],
   metadata: %{}
 }}
iex(11)> Bank.Core.Accounts.view_balance("111-102")
0
iex(12)> Bank.Core.Accounts.view_balance("111-104")
nil
iex(13)> Bank.Core.Accounts.view_balance("7096c01c-95be-4a5b-a7a3-c1cb42ad0e69")
50
```

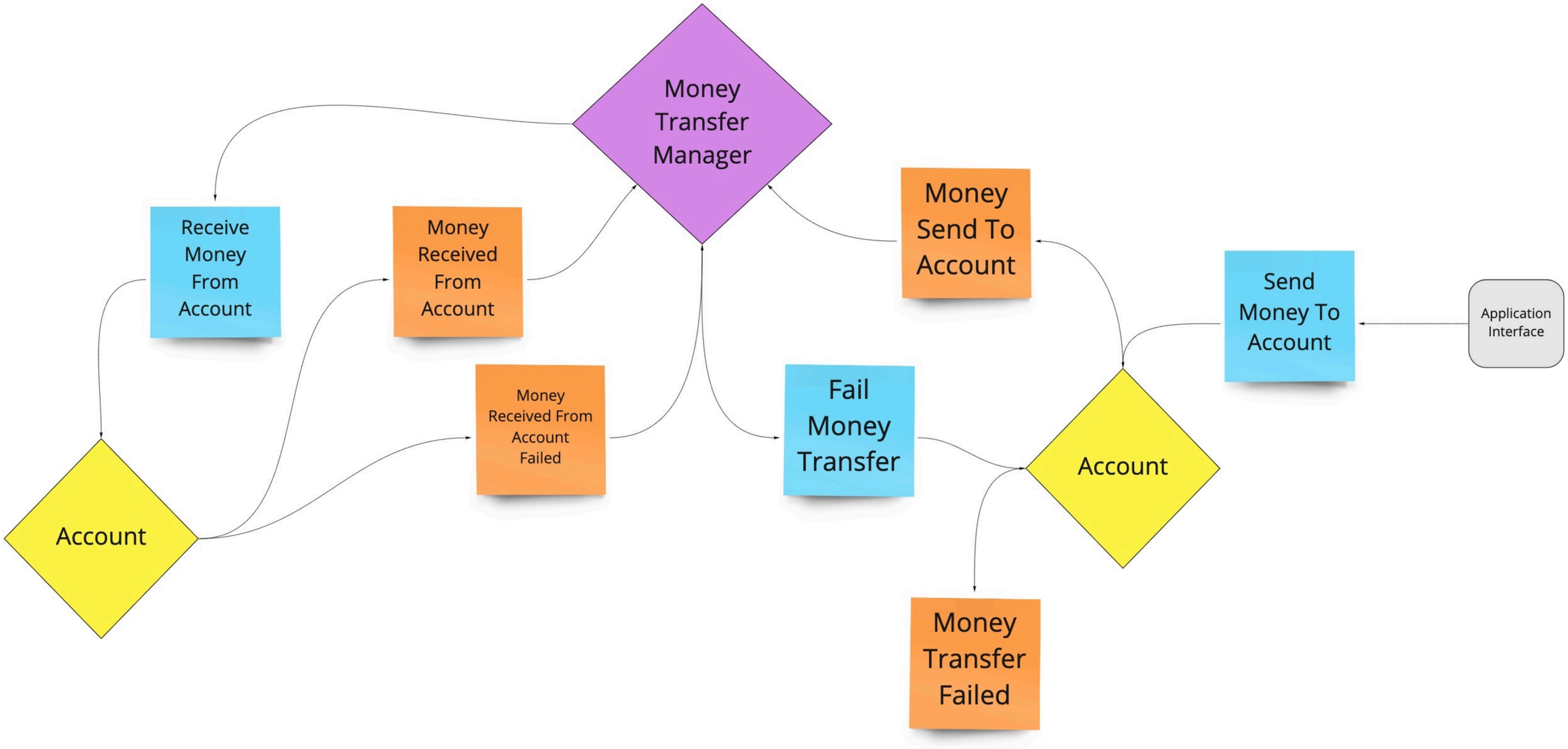
```
iex(10)> Bank.Core.Accounts.send_money("111-102", "111-104", 50)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 0, id: "111-102"},
   aggregate_uuid: "111-102",
   aggregate_version: 7,
   events: [
     %Bank.Core.Events.MoneySentToAccount{
       amount: 50,
       from_account_id: "111-102",
       to_account_id: "111-104",
       transaction_id: "7096c01c-95be-4a5b-a7a3-c1cb42ad0e69"
     },
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"111-102" => 50},
       debit: %{"7096c01c-95be-4a5b-a7a3-c1cb42ad0e69" => 50},
       journal_entry_uuid: "569bb403-daa7-4c9a-bad7-85f2fe396e68"
     }
   ],
   metadata: %{}
 }}
iex(11)> Bank.Core.Accounts.view_balance("111-102")
0
iex(12)> Bank.Core.Accounts.view_balance("111-104")
nil
iex(13)> Bank.Core.Accounts.view_balance("7096c01c-95be-4a5b-a7a3-c1cb42ad0e69")
50
```

```
iex(10)> Bank.Core.Accounts.send_money("111-102", "111-104", 50)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 0, id: "111-102"},
   aggregate_uuid: "111-102",
   aggregate_version: 7,
   events: [
     %Bank.Core.Events.MoneySentToAccount{
       amount: 50,
       from_account_id: "111-102",
       to_account_id: "111-104",
       transaction_id: "7096c01c-95be-4a5b-a7a3-c1cb42ad0e69"
     },
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"111-102" => 50},
       debit: %{"7096c01c-95be-4a5b-a7a3-c1cb42ad0e69" => 50},
       journal_entry_uuid: "569bb403-daa7-4c9a-bad7-85f2fe396e68"
     }
   ],
   metadata: %{}
 }}
iex(11)> Bank.Core.Accounts.view_balance("111-102")
0
iex(12)> Bank.Core.Accounts.view_balance("111-104")
nil
iex(13)> Bank.Core.Accounts.view_balance("7096c01c-95be-4a5b-a7a3-c1cb42ad0e69")
50
```

```
iex(10)> Bank.Core.Accounts.send_money("111-102", "111-104", 50)
{:ok,
 %Commanded.Commands.ExecutionResult{
   aggregate_state: %Bank.Core.Accounts.Account{balance: 0, id: "111-102"},
   aggregate_uuid: "111-102",
   aggregate_version: 7,
   events: [
     %Bank.Core.Events.MoneySentToAccount{
       amount: 50,
       from_account_id: "111-102",
       to_account_id: "111-104",
       transaction_id: "7096c01c-95be-4a5b-a7a3-c1cb42ad0e69"
     },
     %Bank.Core.Events.JournalEntryCreated{
       credit: %{"111-102" => 50},
       debit: %{"7096c01c-95be-4a5b-a7a3-c1cb42ad0e69" => 50},
       journal_entry_uuid: "569bb403-daa7-4c9a-bad7-85f2fe396e68"
     }
   ],
   metadata: %{}
 }}
iex(11)> Bank.Core.Accounts.view_balance("111-102")
0
iex(12)> Bank.Core.Accounts.view_balance("111-104")
nil
iex(13)> Bank.Core.Accounts.view_balance("7096c01c-95be-4a5b-a7a3-c1cb42ad0e69")
50
```

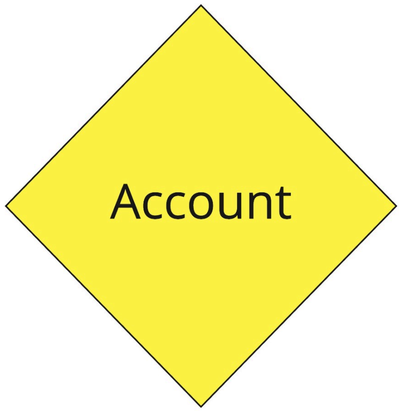

Money Transfer Manager
Receive Money From Account
Money Received From Account
Money Send To Account
Send Money To Account
Application Interface
Account
Account


Money Transfer Manager
Receive Money From Account
Account
Money Received From Account Failed
Fail Money Transfer
Account
Money Transfer Failed


Money Transfer Manager
Receive Money From Account
Money Received From Account
Money Send To Account
Send Money To Account
Application Interface
Money Received From Account Failed
Fail Money Transfer
Account
Account
Money Transfer Failed

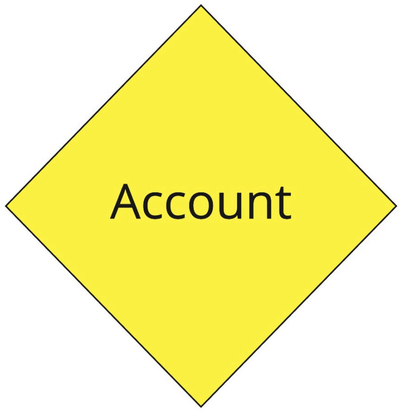
Aggregate

```elixir
def execute(%Account{id: nil}, %ReceiveMoneyFromAccount{} = cmd) do
  %MoneyReceivedFromAccountFailed{
    transaction_id: cmd.transaction_id,
    from_account_id: cmd.from_account_id,
    to_account_id: cmd.to_account_id,
    amount: cmd.amount
  }
end

def apply(state, %MoneyReceivedFromAccountFailed{}), do: state
```



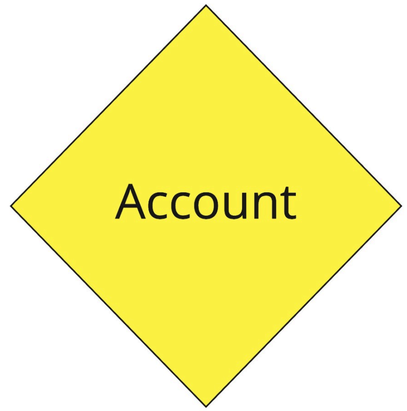
Aggregate

```
def execute(%Account{id: nil}, %ReceiveMoneyFromAccount{} = cmd) do
  %MoneyReceivedFromAccountFailed{
    transaction_id: cmd.transaction_id,
    from_account_id: cmd.from_account_id,
    to_account_id: cmd.to_account_id,
    amount: cmd.amount
  }
end

def apply(state, %MoneyReceivedFromAccountFailed{}), do: state
```



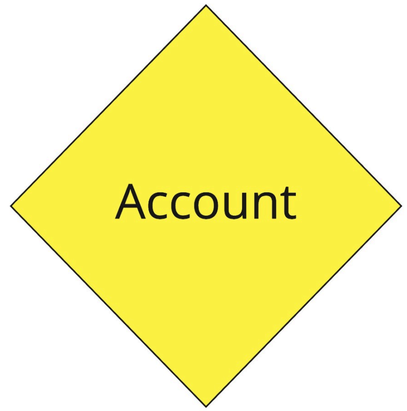
Aggregate

```elixir
def execute(%Account{} = state, %FailMoneyTransfer{} = cmd) do
  [
    %MoneyTransferFailed{
      transaction_id: cmd.transaction_id,
      from_account_id: state.id,
      to_account_id: cmd.to_account_id,
      amount: cmd.amount
    },
    %JournalEntryCreated{
      journal_entry_uuid: Ecto.UUID.generate(),
      credit: %{"#{state.id}" => cmd.amount},
      debit: %{"#{cmd.transaction_id}" => cmd.amount}
    }
  ]
end

def apply(state, %MoneyTransferFailed{} = evt) do
  %{state | balance: state.balance + evt.amount}
end
```



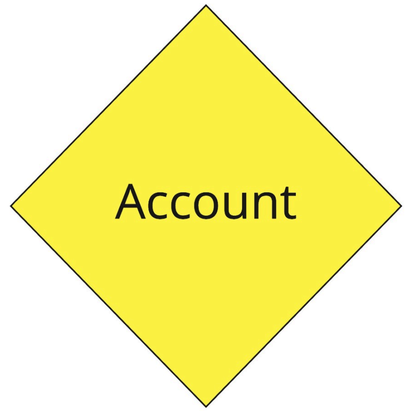
Aggregate

```elixir
def execute(%Account{} = state, %FailMoneyTransfer{} = cmd) do
  [
    %MoneyTransferFailed{
      transaction_id: cmd.transaction_id,
      from_account_id: state.id,
      to_account_id: cmd.to_account_id,
      amount: cmd.amount
    },
    %JournalEntryCreated{
      journal_entry_uuid: Ecto.UUID.generate(),
      credit: %{"#{state.id}" => cmd.amount},
      debit: %{"#{cmd.transaction_id}" => cmd.amount}
    }
  ]
end

def apply(state, %MoneyTransferFailed{} = evt) do
  %{state | balance: state.balance + evt.amount}
end
```

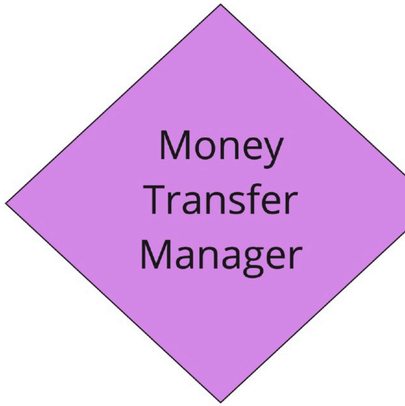


Aggregate

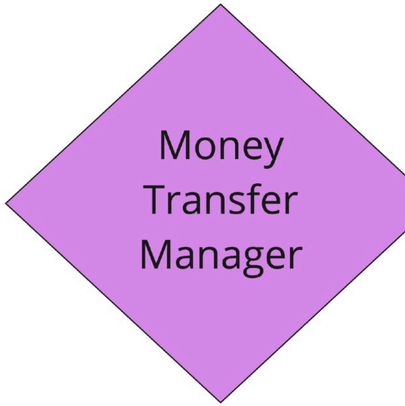
```elixir
def execute(%Account{} = state, %FailMoneyTransfer{} = cmd) do
  [
    %MoneyTransferFailed{
      transaction_id: cmd.transaction_id,
      from_account_id: state.id,
      to_account_id: cmd.to_account_id,
      amount: cmd.amount
    },
    %JournalEntryCreated{
      journal_entry_uuid: Ecto.UUID.generate(),
      credit: %{"#{state.id}" => cmd.amount},
      debit: %{"#{cmd.transaction_id}" => cmd.amount}
    }
  ]
end

def apply(state, %MoneyTransferFailed{} = evt) do
  %{state | balance: state.balance + evt.amount}
end
```



```elixir
defmodule Bank.Core.Accounts.MoneyTransferProcessManager do

  ...

  def interested?(%MoneyReceivedFromAccountFailed{transaction_id: id}), do: {:continue, id}
  def interested?(%MoneyTransferFailed{transaction_id: id}), do: {:stop, id}

  ...

  def handle(%__MODULE__{}, %MoneyReceivedFromAccountFailed{} = evt),
    do: [
      %FailMoneyTransfer{
        transaction_id: evt.transaction_id,
        from_account_id: evt.from_account_id,
        to_account_id: evt.to_account_id,
        amount: evt.amount
      }
    ]
end
```



```elixir
defmodule Bank.Core.Accounts.MoneyTransferProcessManager do

  ...

  def interested?(%MoneyReceivedFromAccountFailed{transaction_id: id}), do: {:continue, id}
  def interested?(%MoneyTransferFailed{transaction_id: id}), do: {:stop, id}

  ...

  def handle(%__MODULE__{}, %MoneyReceivedFromAccountFailed{} = evt),
    do: [
      %FailMoneyTransfer{
        transaction_id: evt.transaction_id,
        from_account_id: evt.from_account_id,
        to_account_id: evt.to_account_id,
        amount: evt.amount
      }
    ]
end
```



And it’s Fixed!!

```
iex(10)> Bank.Core.Accounts.view_balance("111-104")
nil
iex(11)> Bank.Core.Accounts.view_balance("111-102")
50
iex(12)> Bank.Core.Accounts.view_balance("7096c01c-95be-4a5b-a7a3-c1cb42ad0e69")
0
```

Recap

Go out there and build something!

The end…

Go out there and build something!

https://github.com/vasspilka/bank

@vasspilka