EVENT SOURCING WITH COMMANDED

FIONA MCCAWLEY

GITHUB.COM/FIMAC
@SAUCERLIKE

WHAT the data is, and HOW it got the way it is.

Event Sourced

current_balance	$123.15

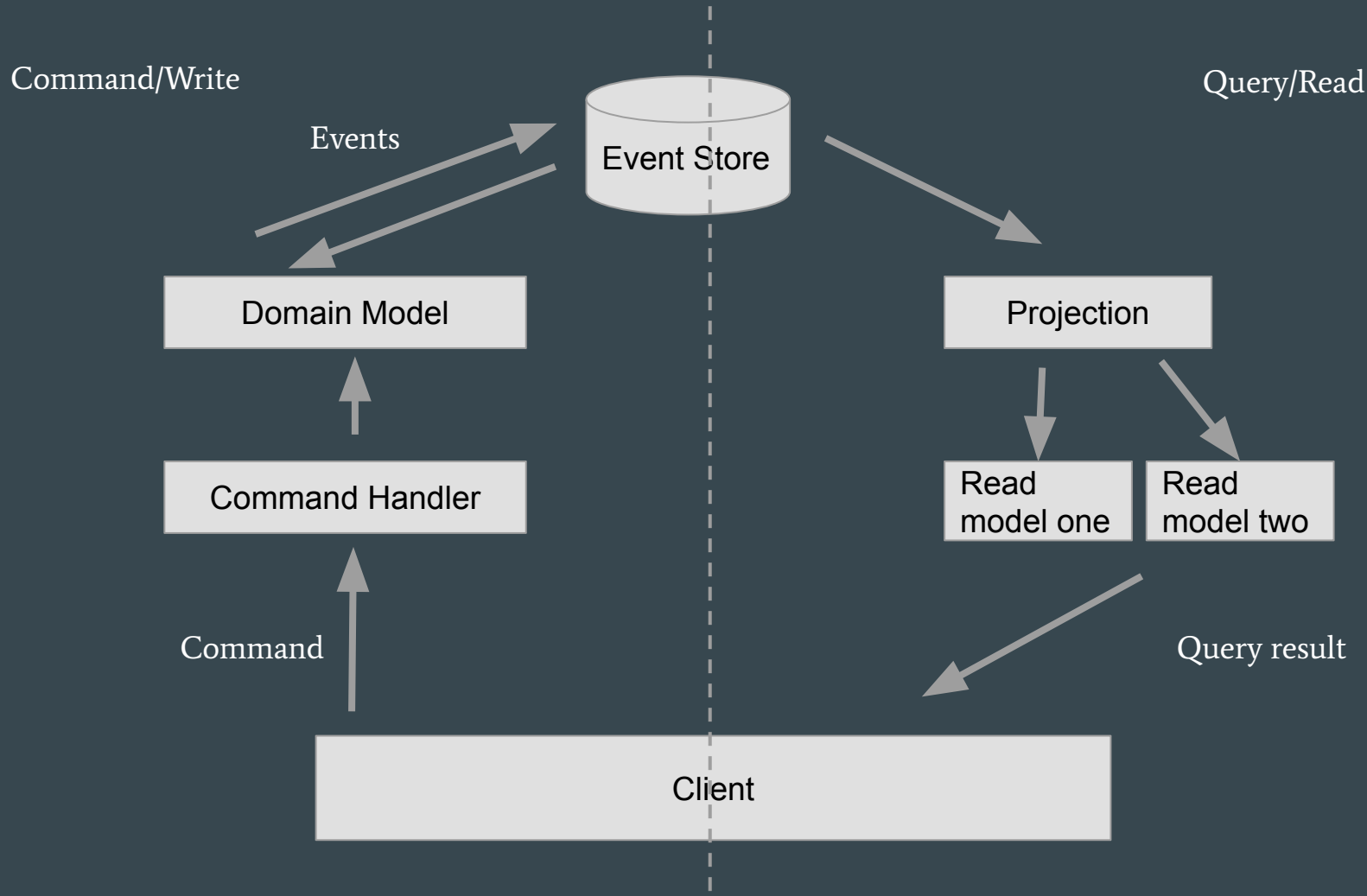
id	event	amount
3	money_withdrawn	$50.00
2	money_deposited	$123.03
1	money_deposited	$50.12

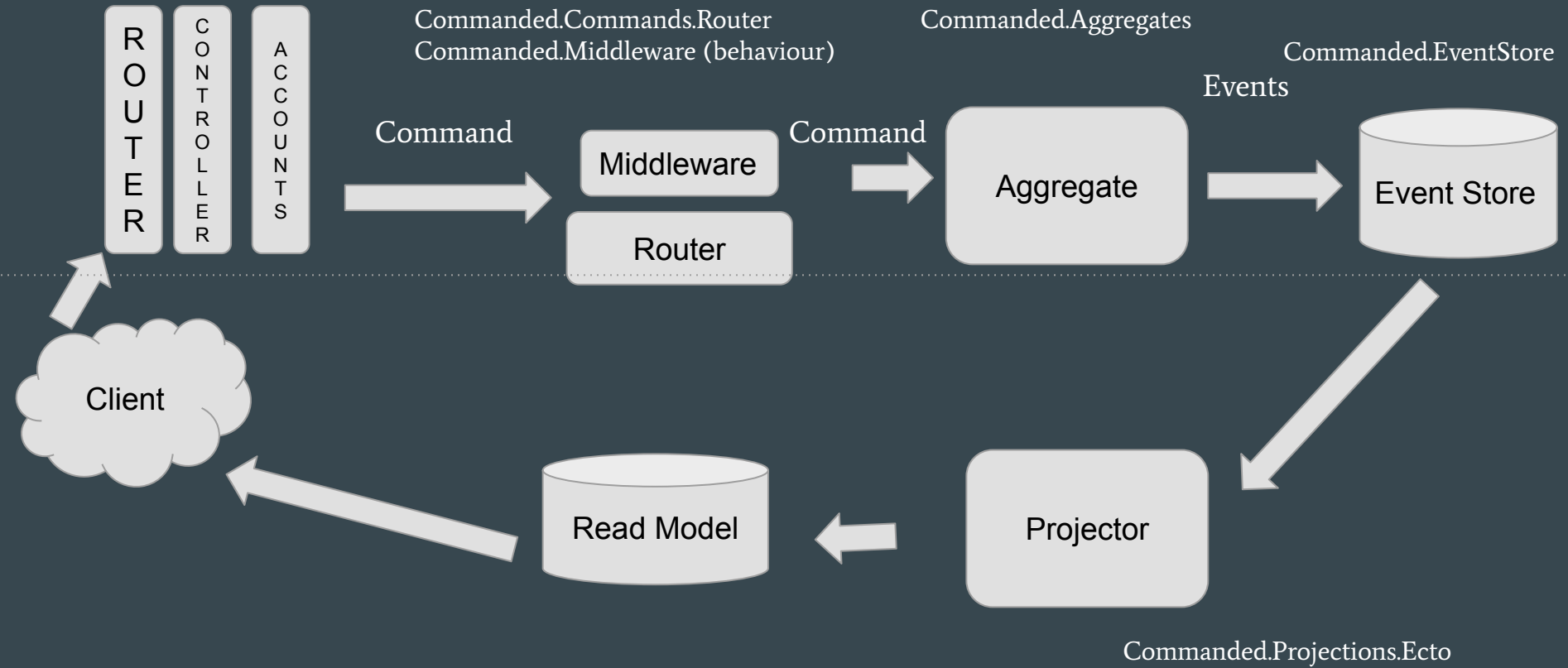
Storing final state only

account_id	current_balance
1	$123.15

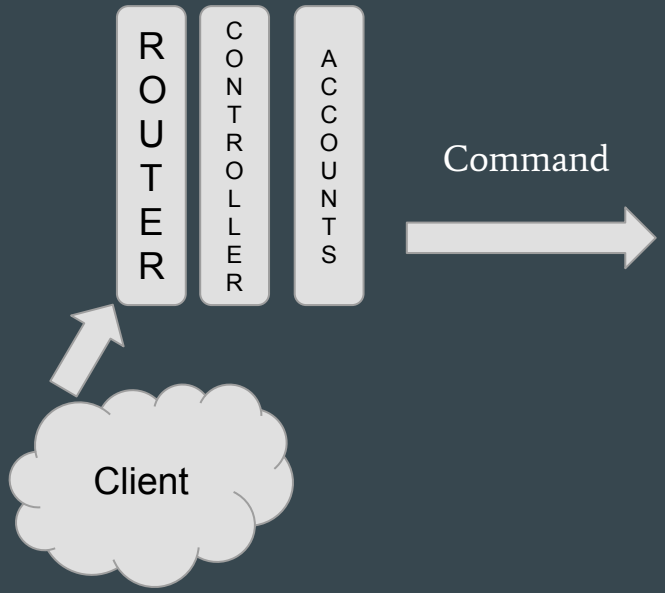
Command Query Responsibility Segregation

“. . . every method should either be a command that performs an action or a query that returns data to the caller, but not both.”


Command/Write
Query/Read
Events
Event Store
Domain Model
Projection
Command Handler
Read model one
Read model two
Command
Query result
Client


ROUTER
CONTROLLER
ACCOUNTS
Commanded.Commands.Router
Commanded.Middleware (behaviour)
Commanded.Aggregates
Commanded.EventStore
Events
Command
Middleware
Router
Command
Aggregate
Event Store
Client
Read Model
Projector
Commanded.Projections.Ecto

Command



POST “/api/accounts”

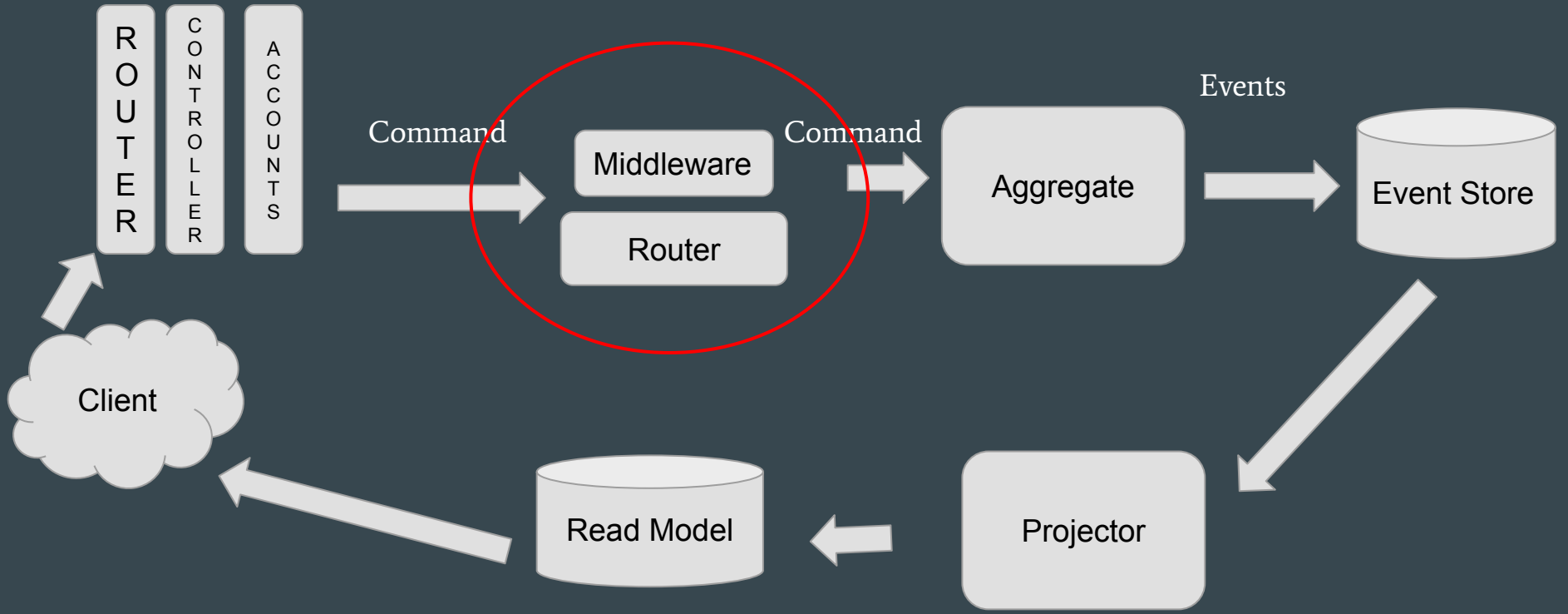
Payload
{"account": {"initial_balance":1000 } }

An action/request

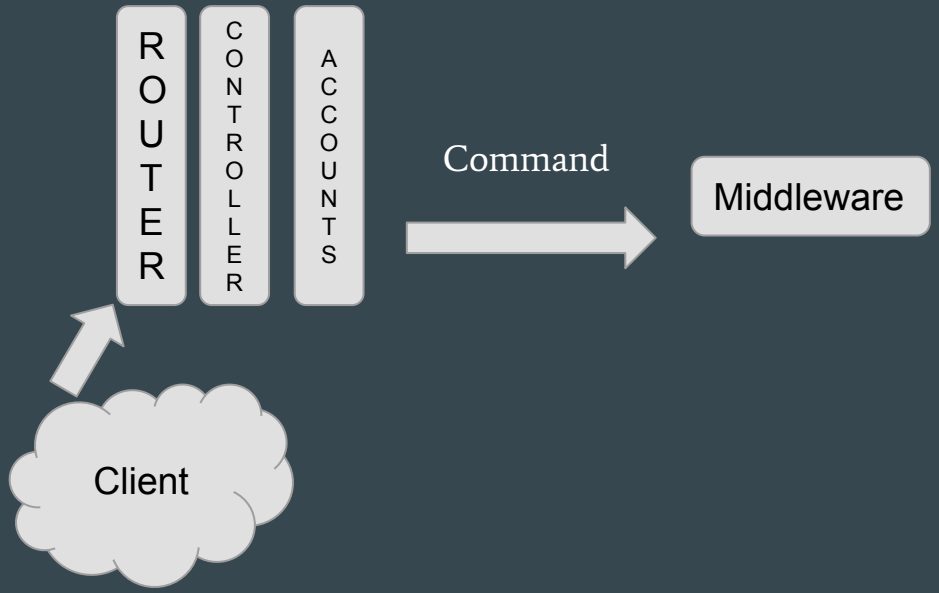
Always in the imperative

Eg. Add funds, Open Account

```
defmodule BankAPI.Accounts.Commands.OpenAccount do
  @enforce_keys [:account_uuid]

  defstruct [:account_uuid, :initial_balance]
end
```

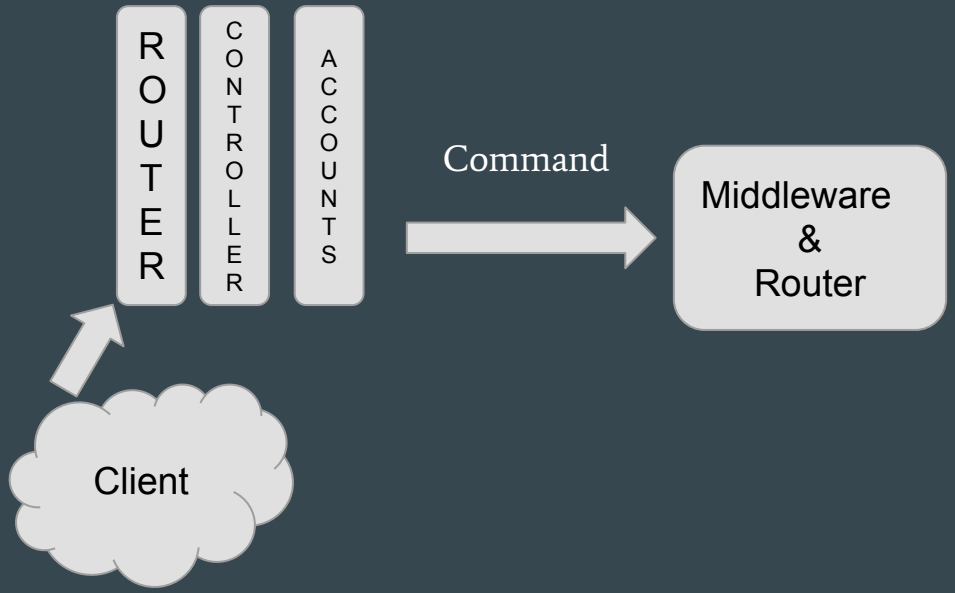

ROUTER
CONTROLLER
ACCOUNTS
Command
Middleware
Router
Command
Aggregate
Events
Event Store
Client
Read Model
Projector

Middleware



Macro used for things like validating commands.

```
middleware(BankAPI.Middleware.ValidateCommand)

defmodule BankAPI.Middleware.ValidateCommand do
 @behaviour Commanded.Middleware

 alias Commanded.Middleware.Pipeline

 def before_dispatch(%Pipeline{command:
command} = pipeline) do
       #validation called here.
 end

 def after_dispatch(pipeline), do: pipeline
 def after_failure(pipeline), do: pipeline
```

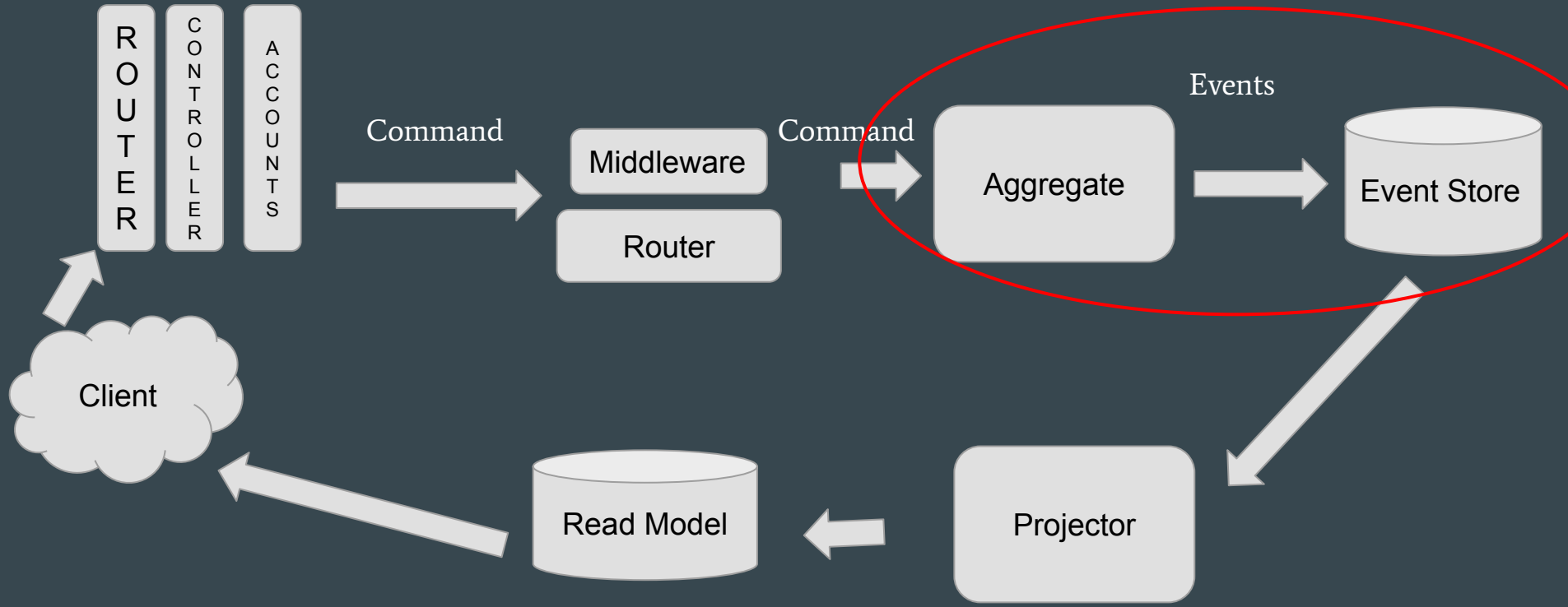


Router

Dispatch commands to the aggregates that will handle them.

```
use Commanded.Commands.Router
```

```
dispatch([OpenAccount],
to: Account,
identity: :account_uuid)
```


ROUTER
CONTROLLER
ACCOUNTS
Command
Middleware
Router
Command
Events
Aggregate
Event Store
Client
Read Model
Projector

```elixir
 def execute(%Account{uuid: nil},
%OpenAccount{account_uuid: ccount_uuid,
initial_balance: initial_balance}) when
initial_balance > 0 do
   %AccountOpened{
     account_uuid: account_uuid,
     initial_balance: initial_balance
   }
 end
 def apply(%Account{} = account,
%AccountOpened{account_uuid: account_uuid,
 initial_balance: initial_balance}) do
   %Account{account | uuid: account_uuid,
current_balance: initial_balance}
 end
```

Aggregate

Like the glue.

Takes a command

Returns an event and persists this to the event store.

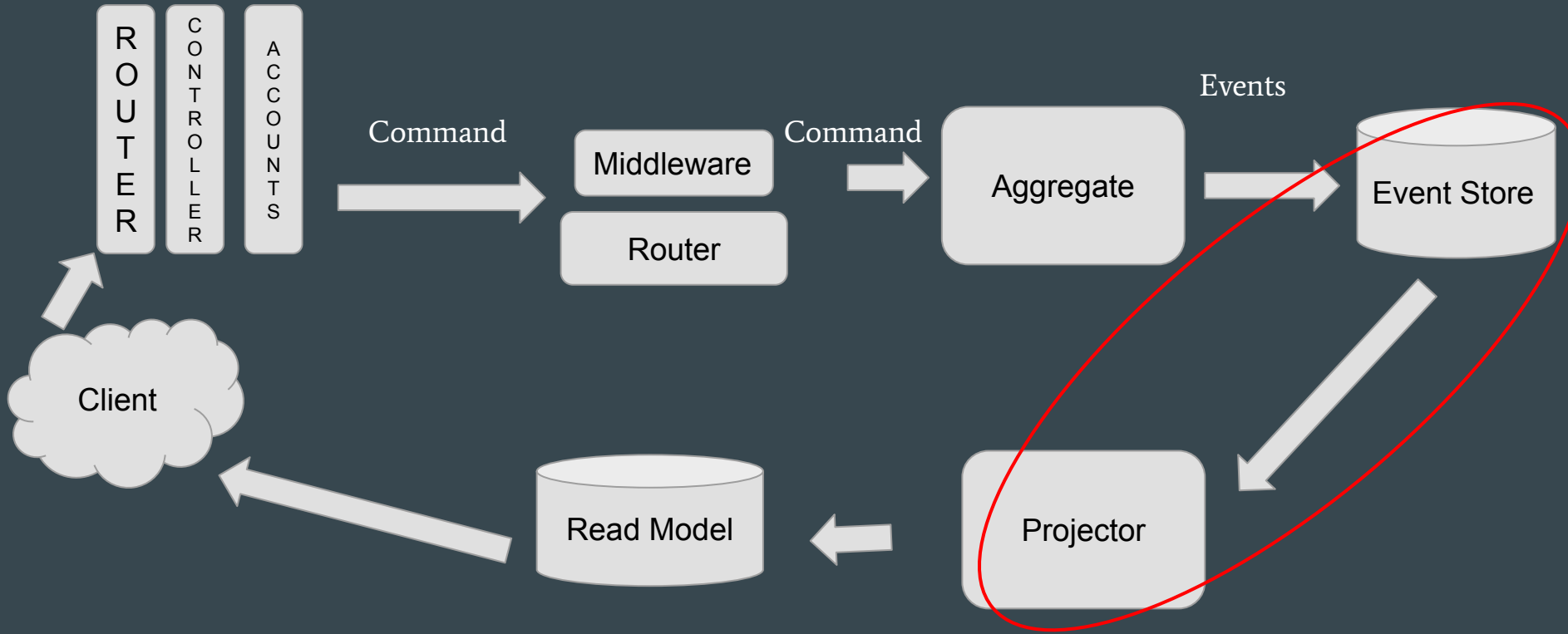
```
Defmodule
BankAPI.Accounts.Events.AccountOpened  do

 defstruct [
   :account_uuid,
   :initial_balance
 ]
end
```

Event

A fact

In the past

Eg. Funds Added

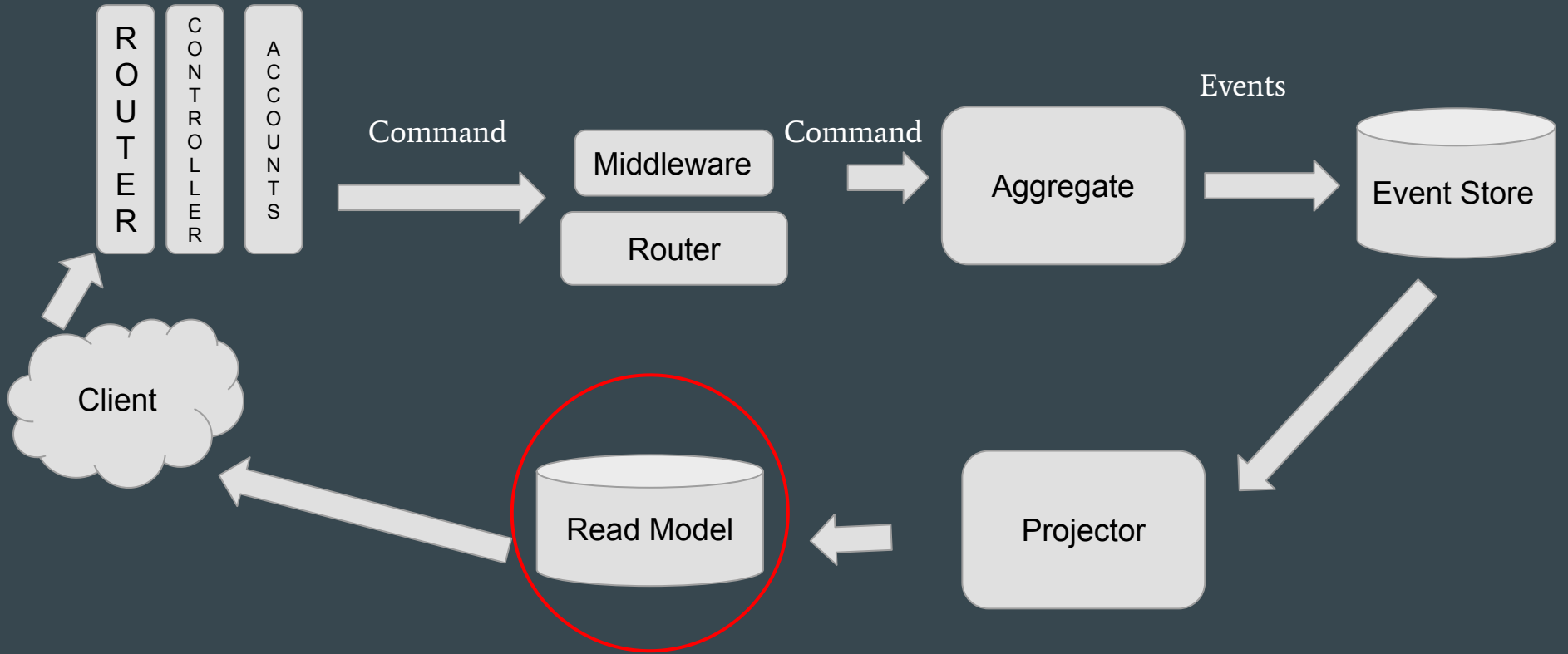

ROUTER
CONTROLLER
ACCOUNTS
Command
Middleware
Router
Command
Aggregate
Events
Event Store
Client
Read Model
Projector

```elixir
defmodule BankAPI.Accounts.Projectors.AccountOpened do
 use Commanded.Projections.Ecto, name:
"Accounts.Projectors.AccountOpened "

 alias BankAPI.Accounts.Events.AccountOpened
 alias BankAPI.Accounts.Projections.Account

 project(%AccountOpened{} = event, _metadata, fn multi
->
   Ecto.Multi.insert(
     multi,
     :account_opened,
     %Account{
       uuid: event.account_uuid,
       current_balance: event.initial_balance,
       status: Account.status().open
     }
   )
 end)
end
```

Projector

Listens for event recorded events and updates the read model.

Does this using the Commanded.Projections.Ecto or Commanded.Event.Handler


ROUTER
CONTROLLER
ACCOUNTS
Command
Middleware
Router
Command
Aggregate
Events
Event Store
Client
Read Model
Projector

Code

Resources

Event Sourcing with Elixir tutorial - https://blog.nootch.net/post/event-sourcing-with-elixir-part-1/

Greg Young on ES & CQRS - https://cqrs.files.wordpress.com/2010/11/cqrs_documents.pdf

Greg Young talk - https://www.youtube.com/watch?v=LDW0QWie21s

Martin Fowler - https://martinfowler.com/eaaDev/EventSourcing.html & https://martinfowler.com/bliki/CQRS.html

Commanded Learnings & Slides for this talk - https://github.com/fimac/commanded_learnings