MVC Software Design Pattern

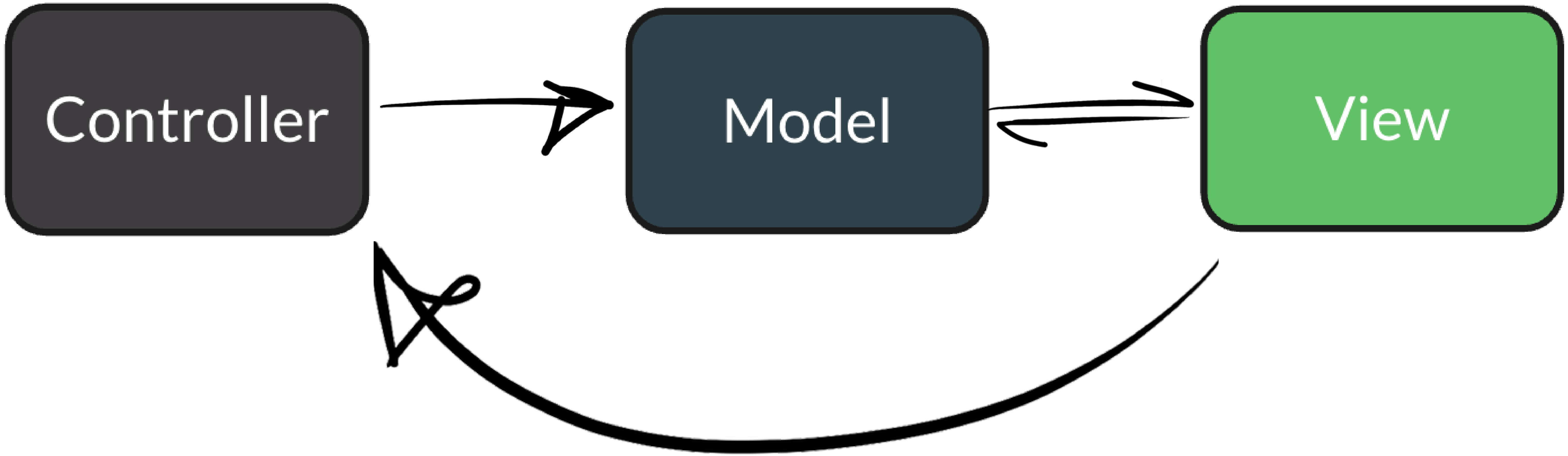
MVC Software Design Pattern

MVC Software Design Pattern

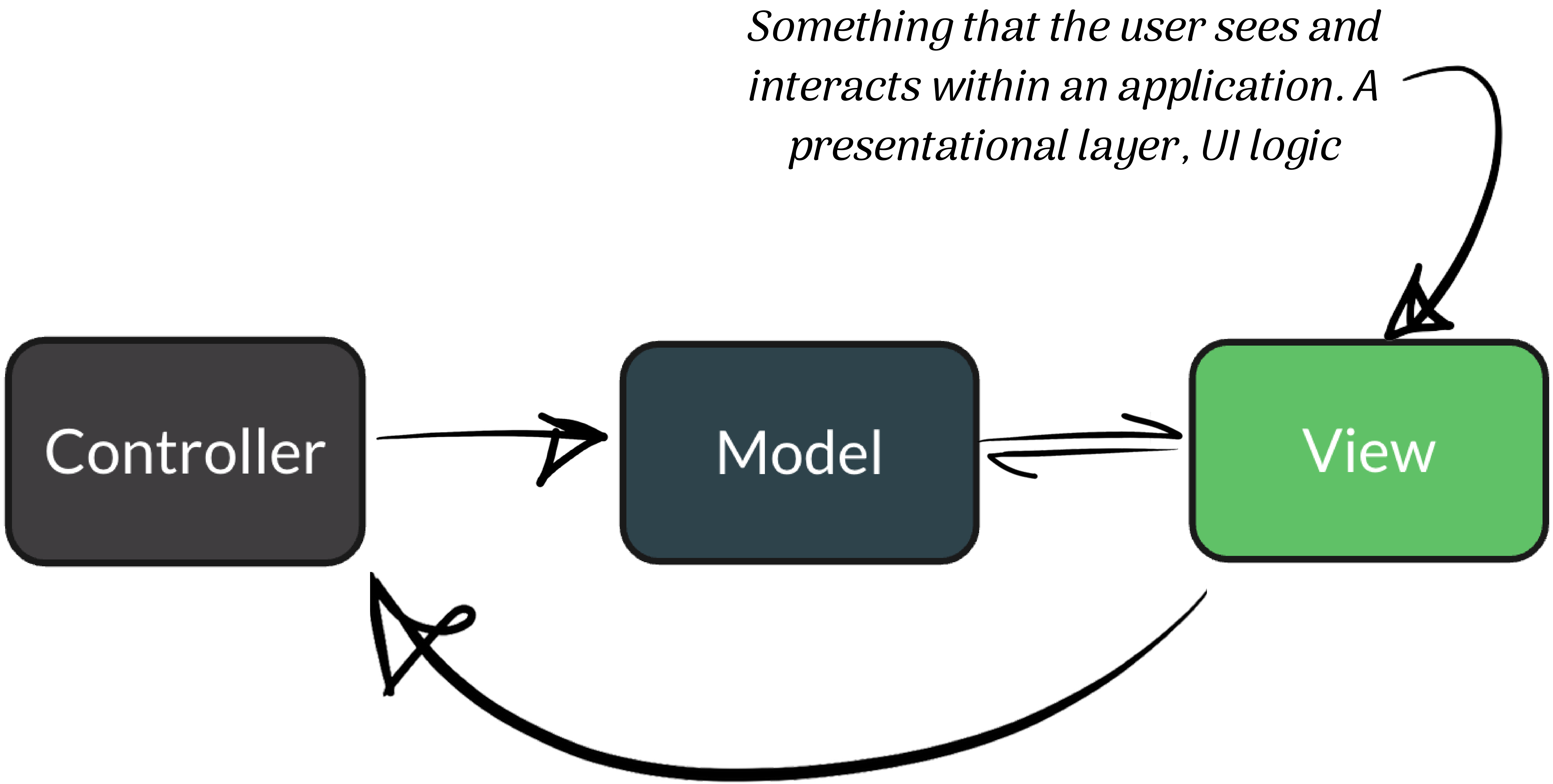
What is it?

History of Redux, MVC pattern, Flux

MVC (Model, View, Controller) Architecture pattern



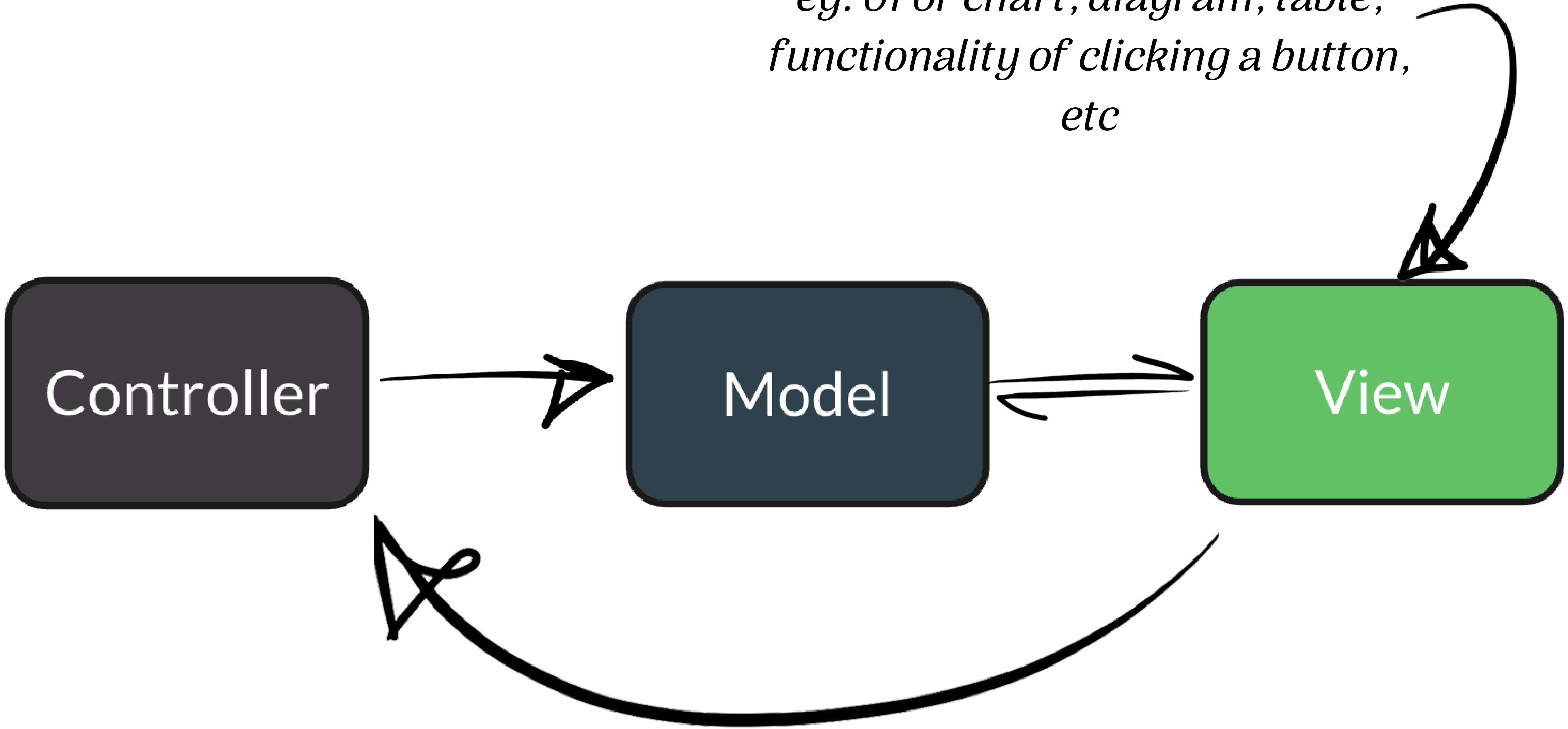
MVC Pattern



Something that the user sees and interacts within an application. A presentational layer, UI logic

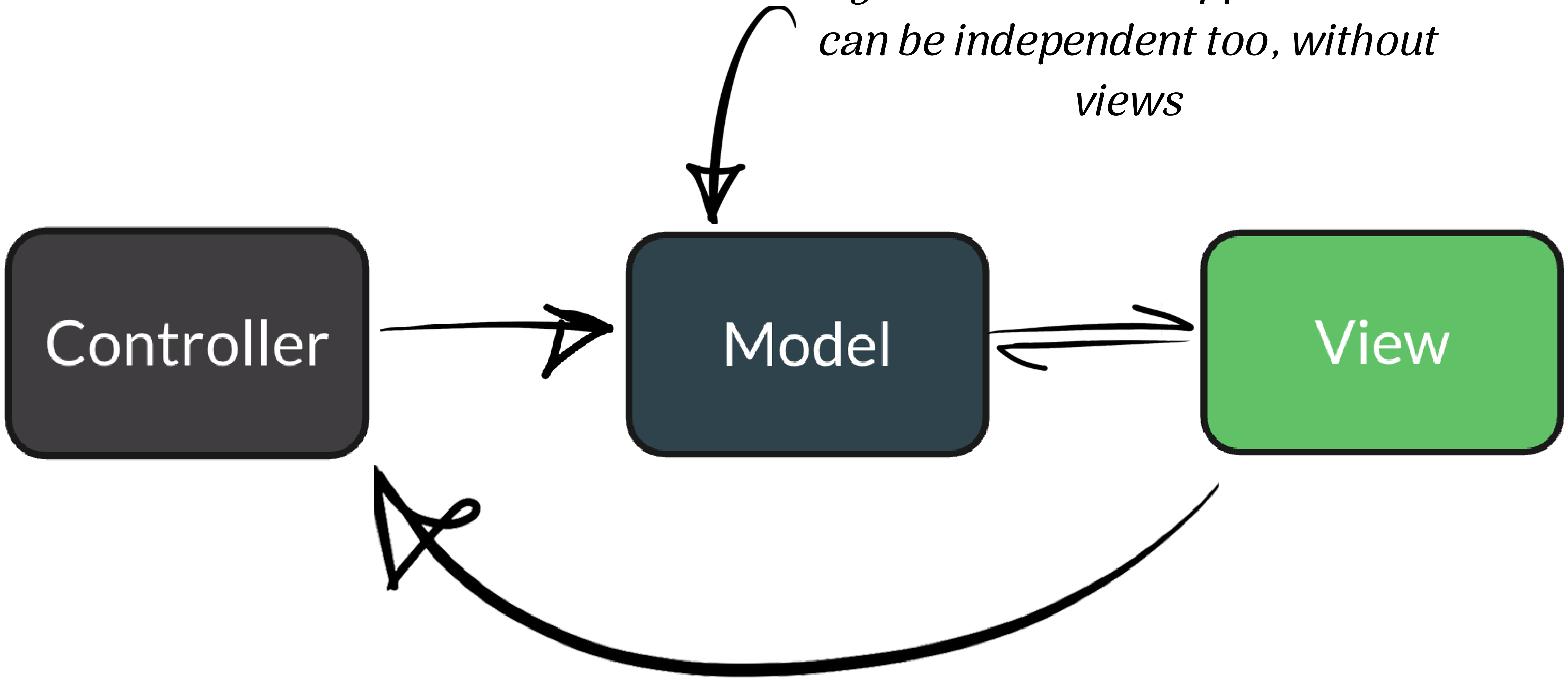
MVC Pattern

Something that the user sees and interacts within an application. A presentational layer, UI logic
eg: UI of chart, diagram, table, functionality of clicking a button, etc



MVC Pattern

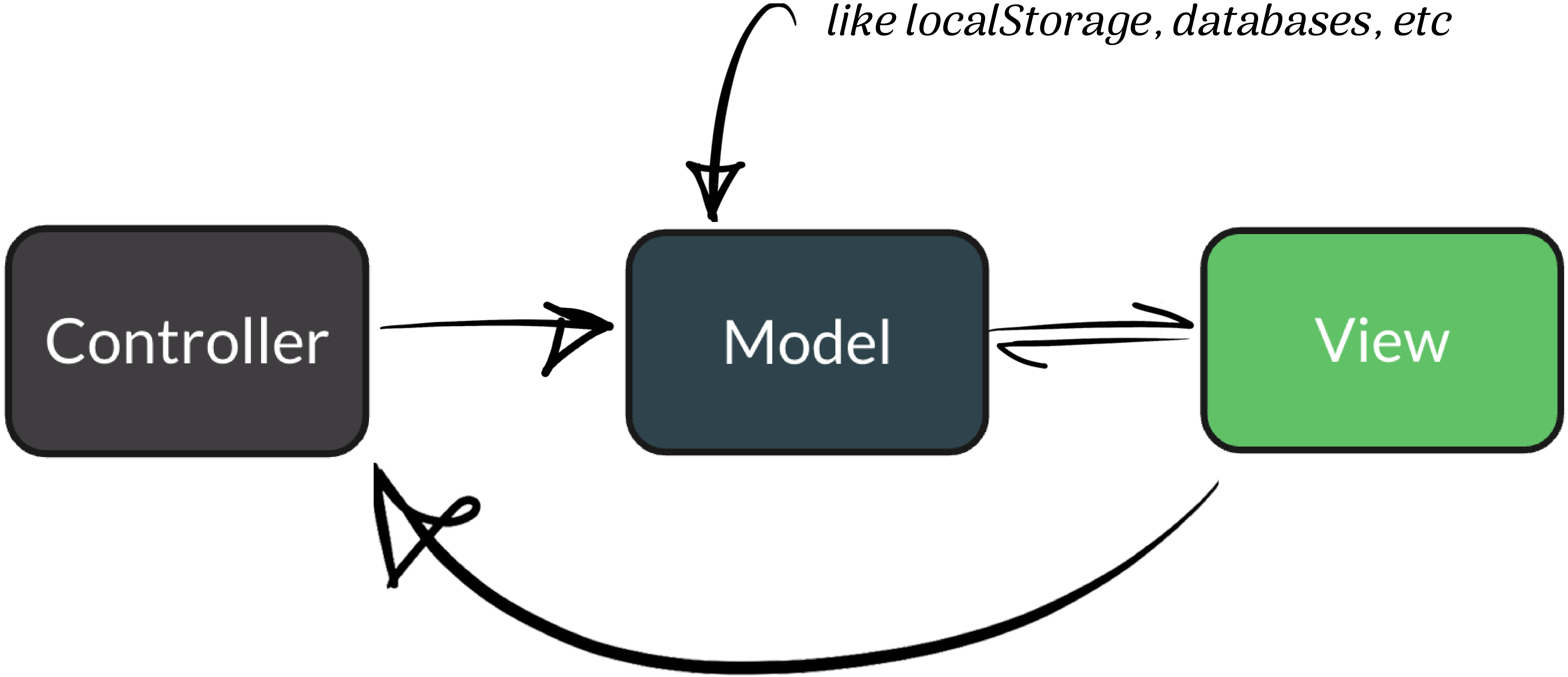
Contains the state/data-related logic that runs the application. It can be independent too, without views



MVC Pattern

Contains the state/data-related logic that runs the application. It can be independent too, without views

eg: something that manages state, like localStorage, databases, etc



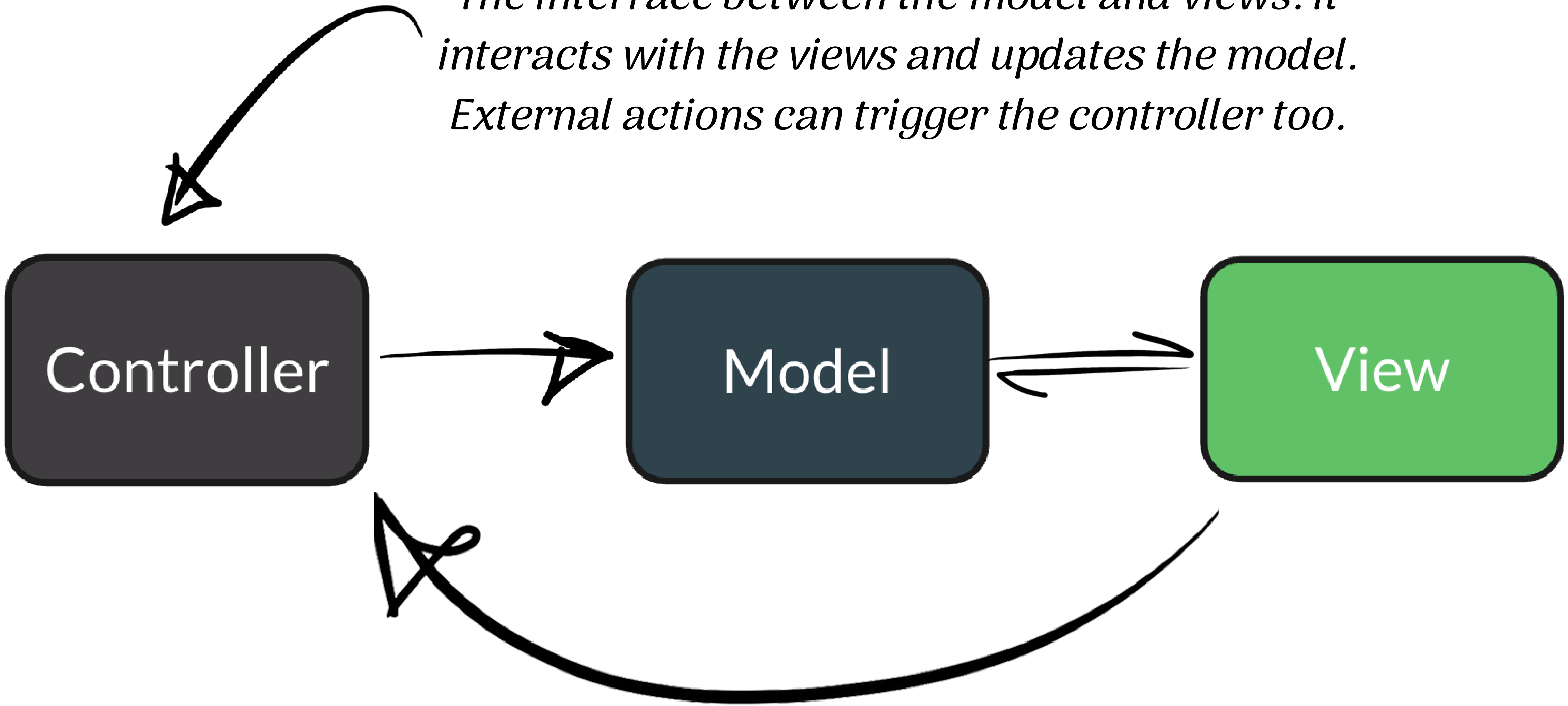
Controller

Model

View

MVC Pattern

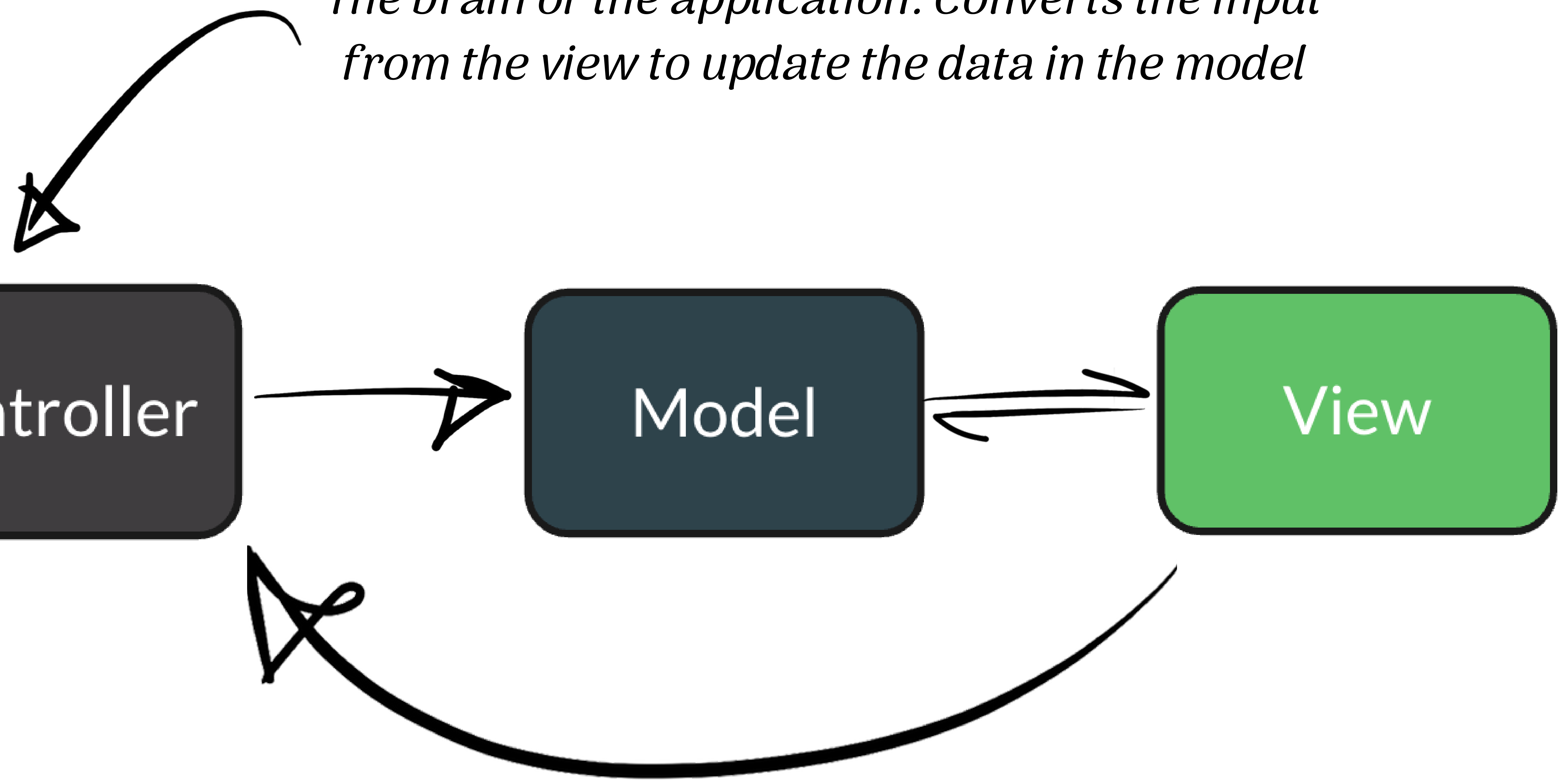
The interface between the model and views. It interacts with the views and updates the model. External actions can trigger the controller too.



MVC Pattern

The interface between the model and views. It interacts with the views and updates the model. External actions can trigger the controller too.

The brain of the application. Converts the input from the view to update the data in the model

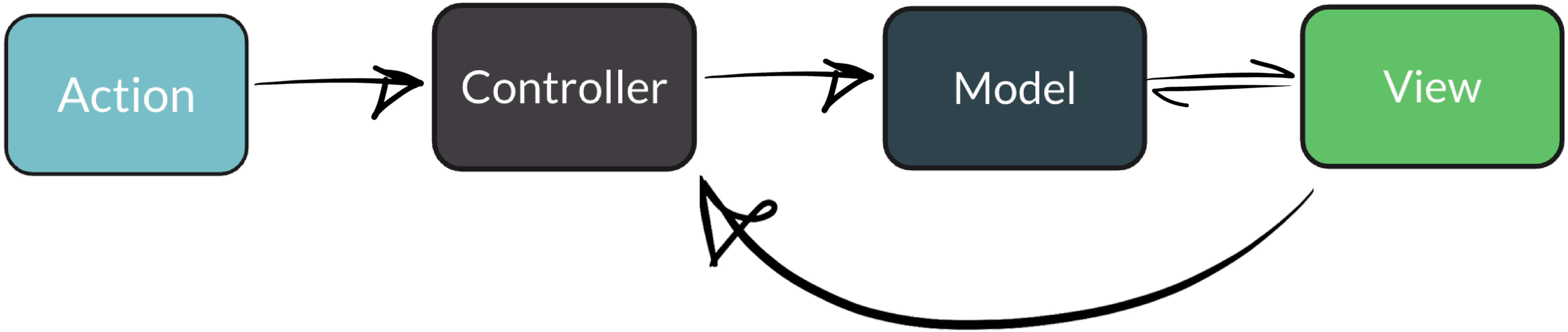


Controller

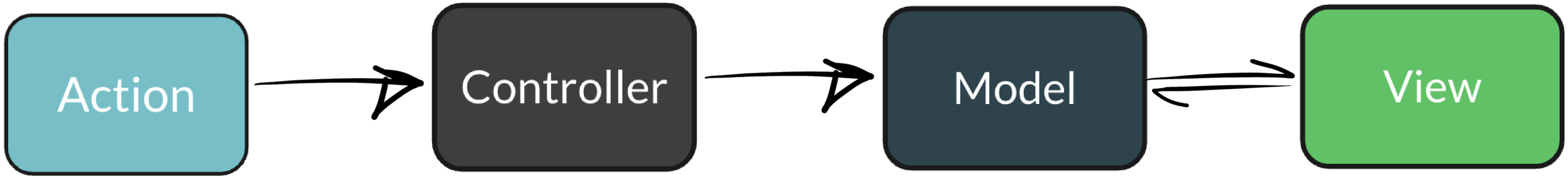
Model

View

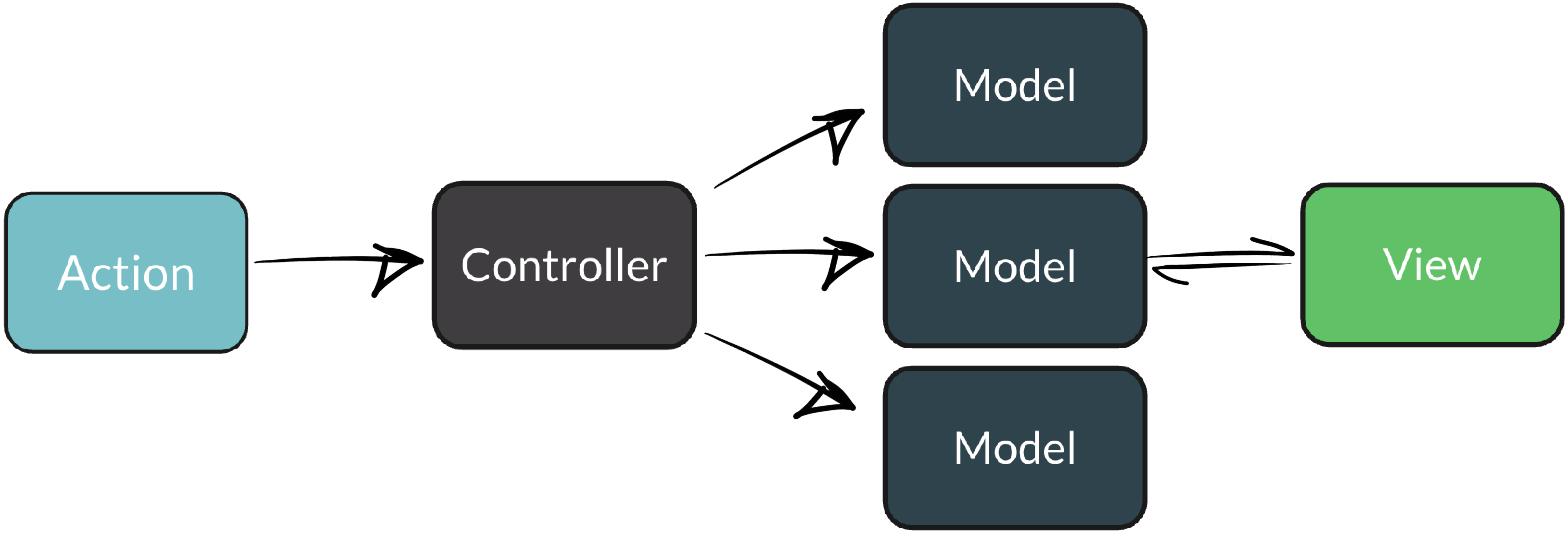
MVC Pattern



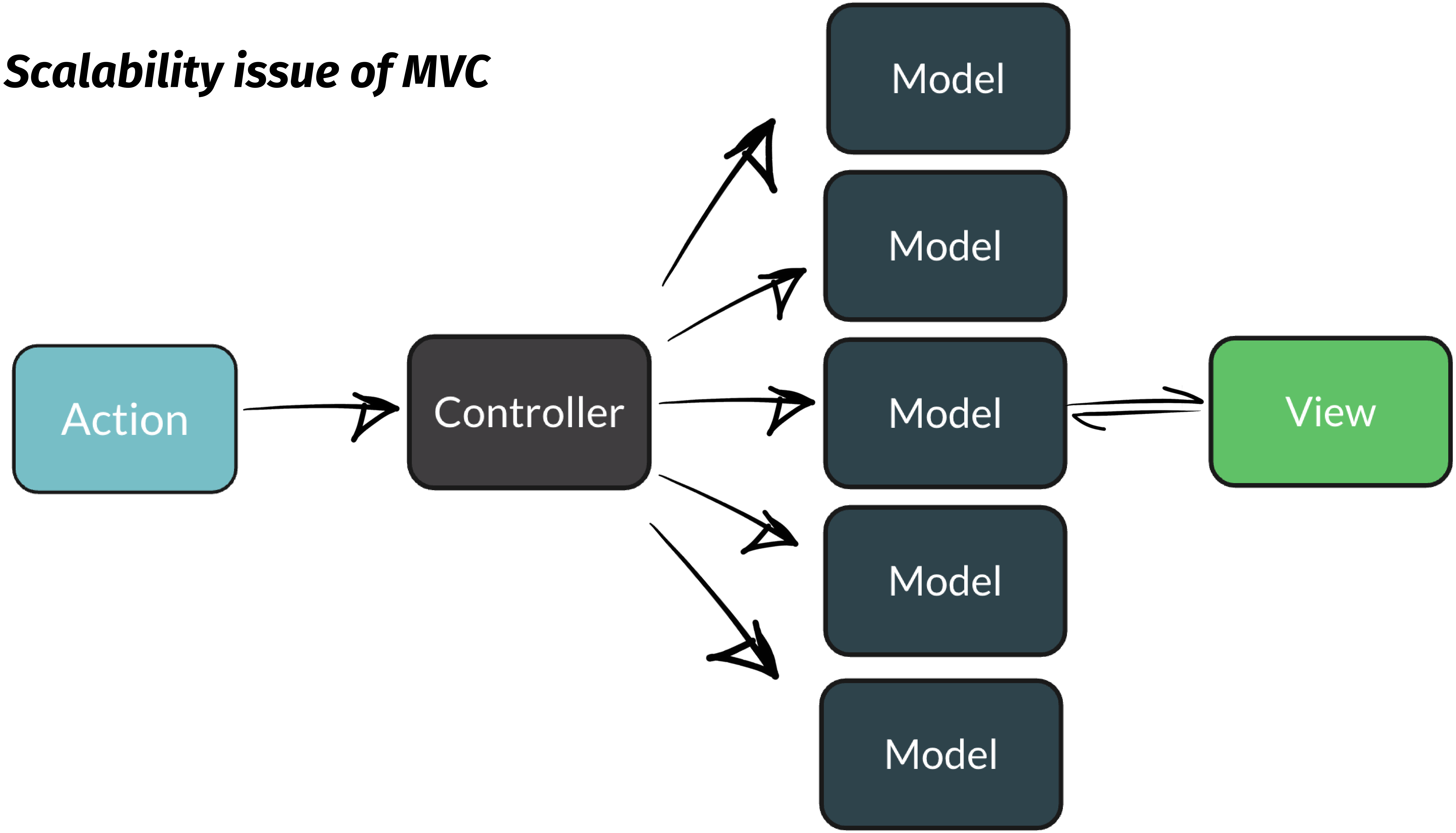
Scalability issue of MVC



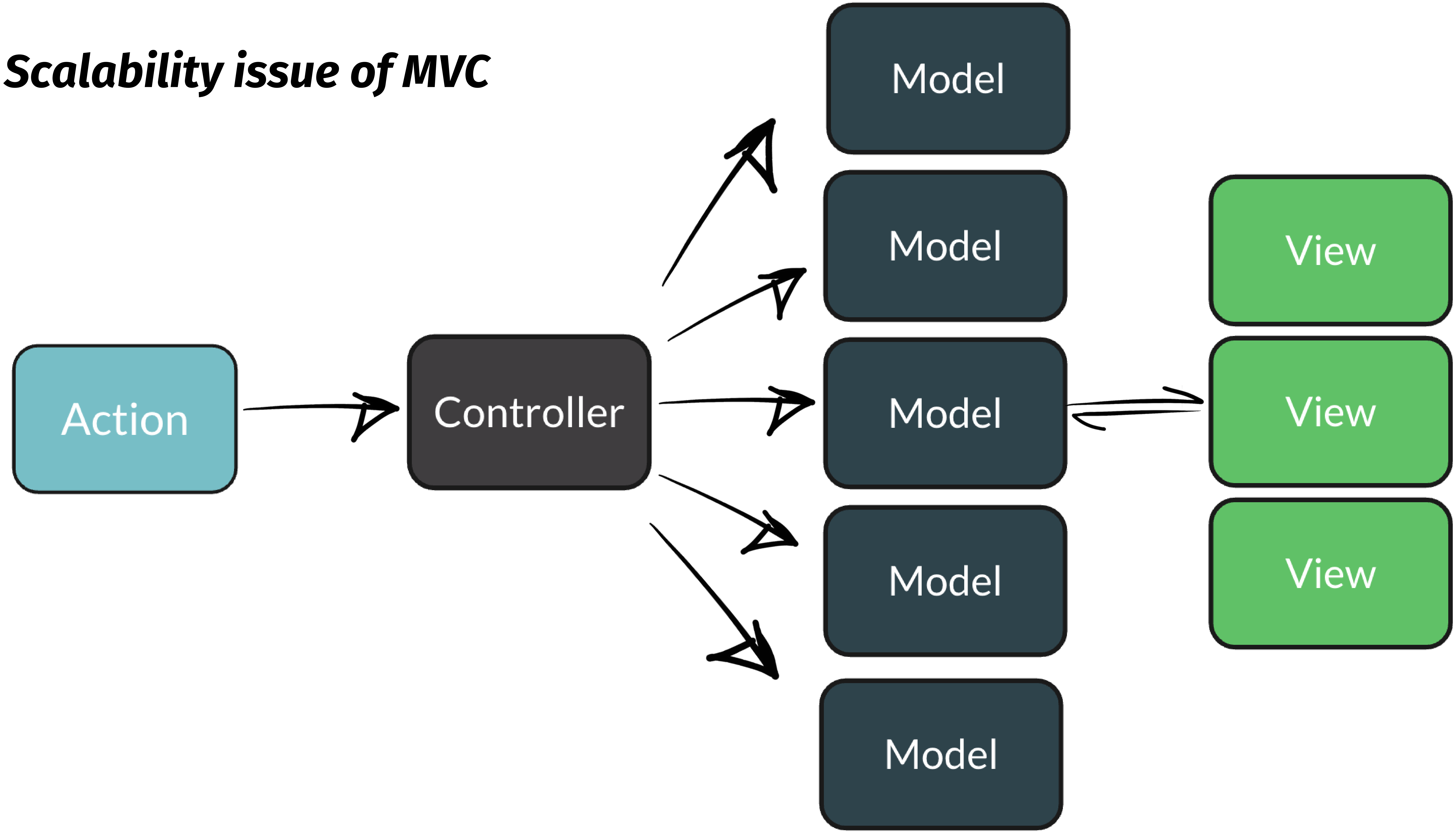
Scalability issue of MVC



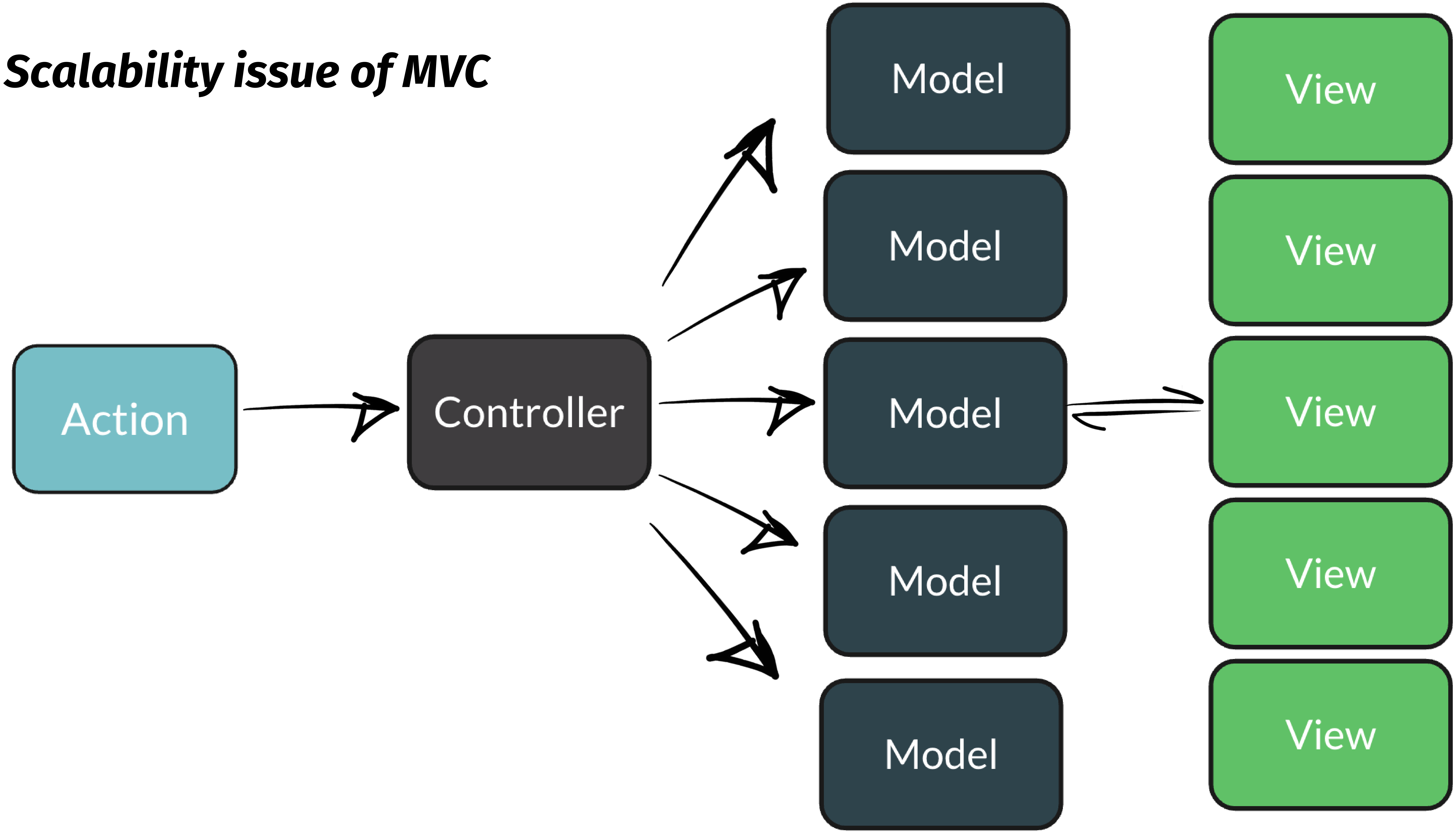
Scalability issue of MVC



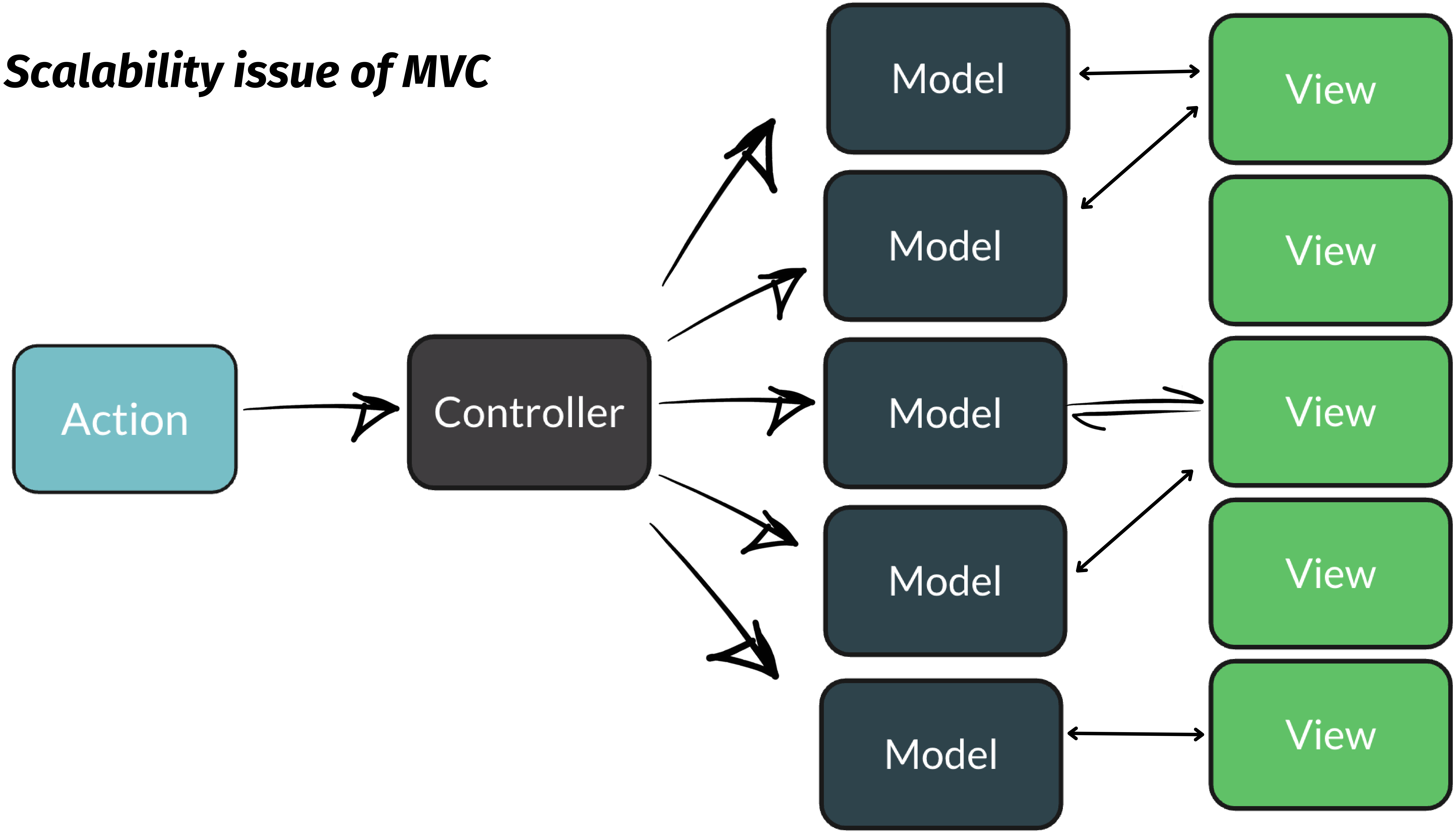
Scalability issue of MVC



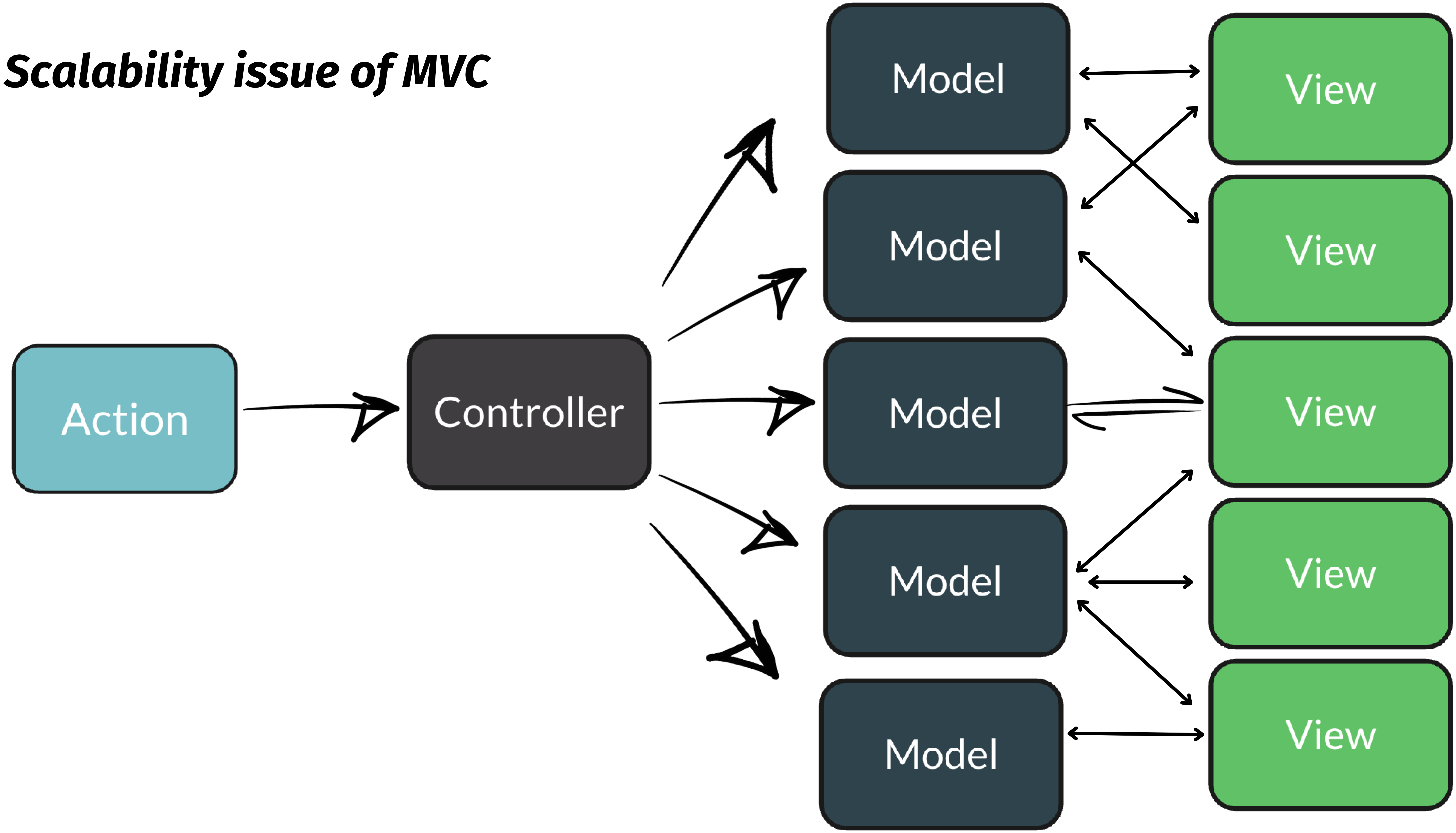
Scalability issue of MVC



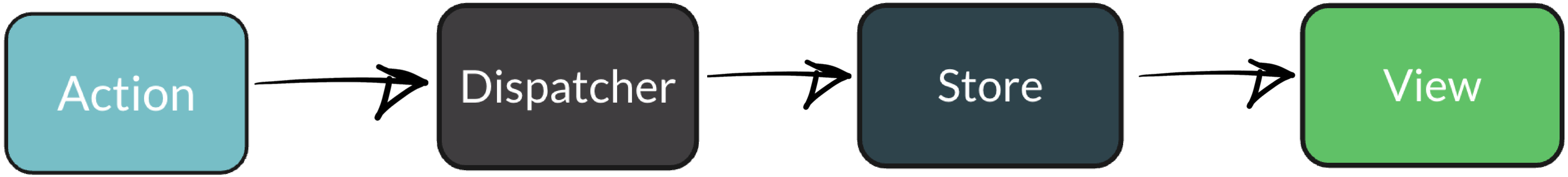
Scalability issue of MVC



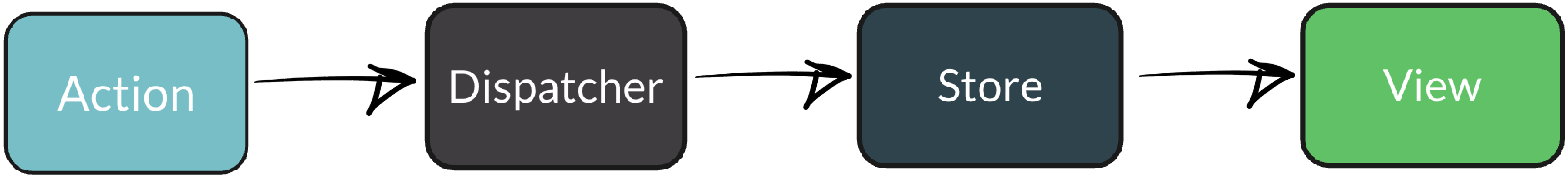
Scalability issue of MVC



Flux Architecture



Flux Architecture



Action

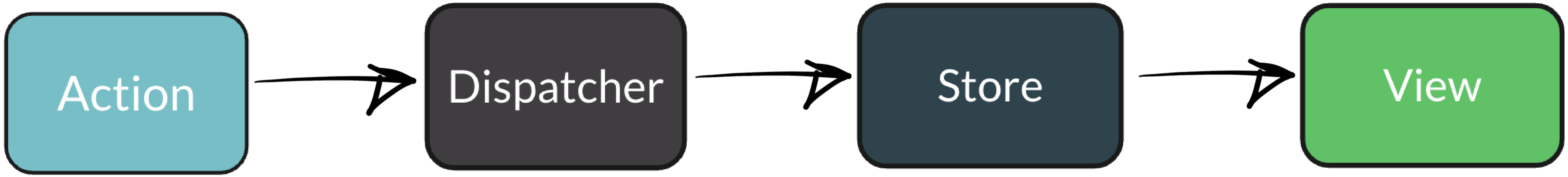
Dispatcher

Store

View

Uni-Directional Flow

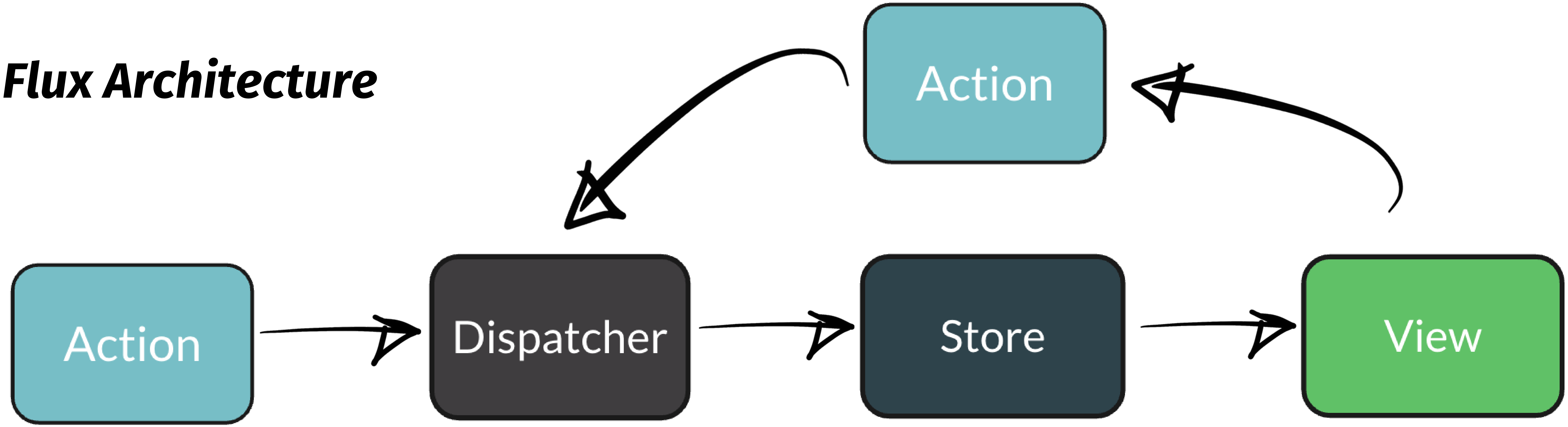
Flux Architecture



Uni-Directional Flow

- *Views react to changes in the store*
- *Stores can only get updated through dispatchers*
- *Dispatchers can only be triggered by actions*
- *Actions can only get triggered by Views*

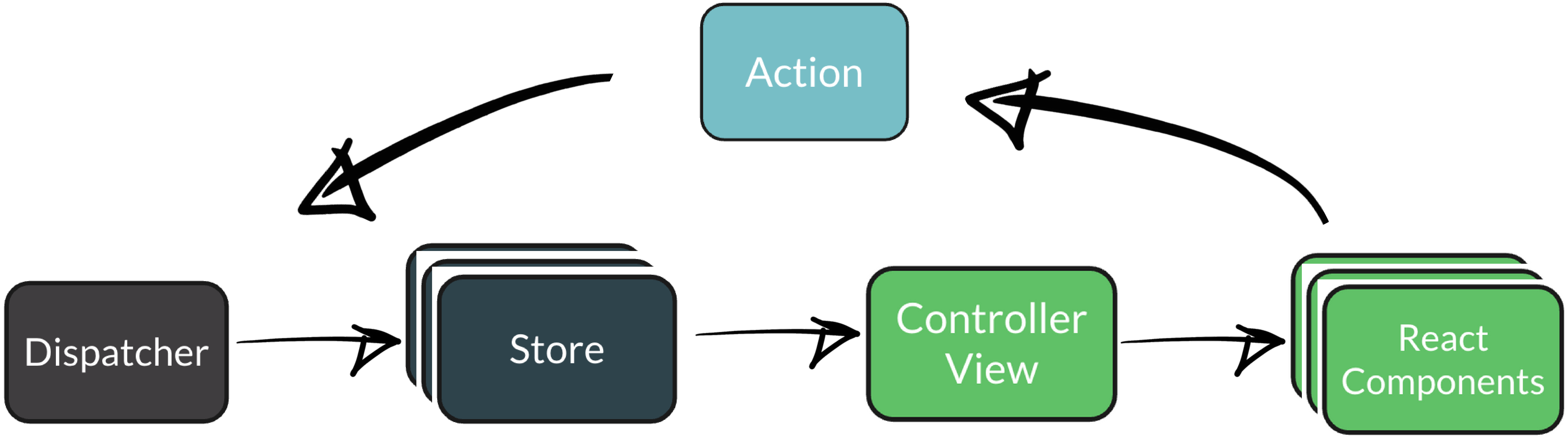
Flux Architecture



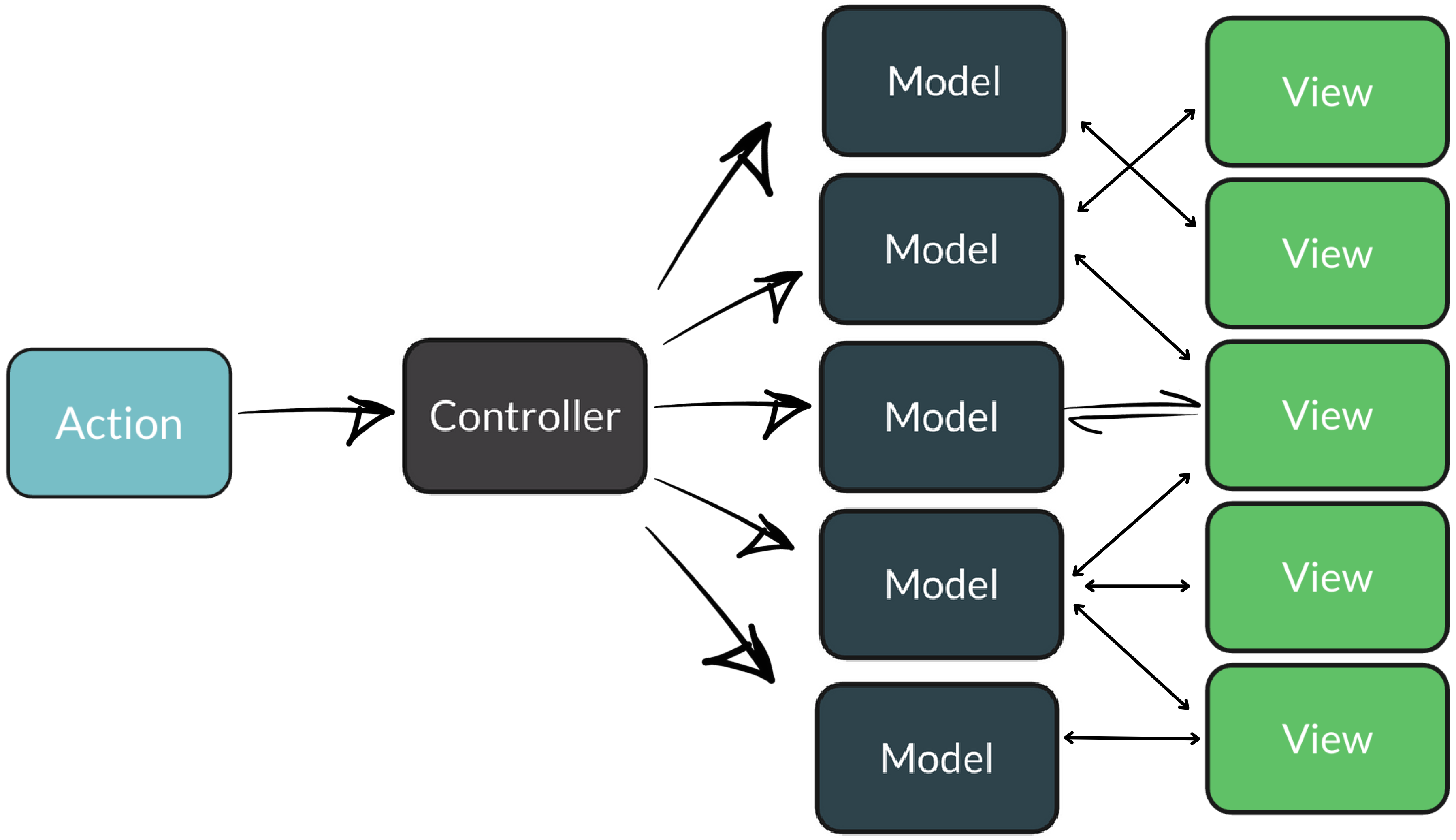
Uni-Directional Flow

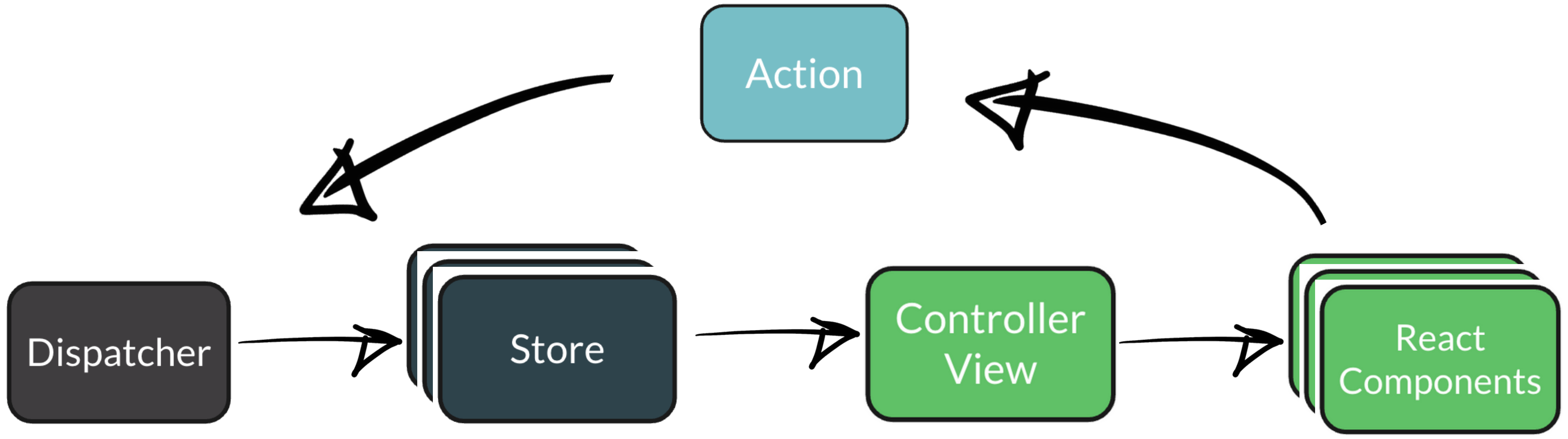
- *Views react to changes in the store*
- *Stores can only get updated through dispatchers*
- *Dispatchers can only be triggered by actions*
- *Actions can only get triggered by Views*

Flux Architecture with React



MVC v/s Flux


Action
Controller
Model
Model
Model
Model
Model
View
View
View
View
View


Action
Dispatcher
Store
Controller View
React Components

MVC

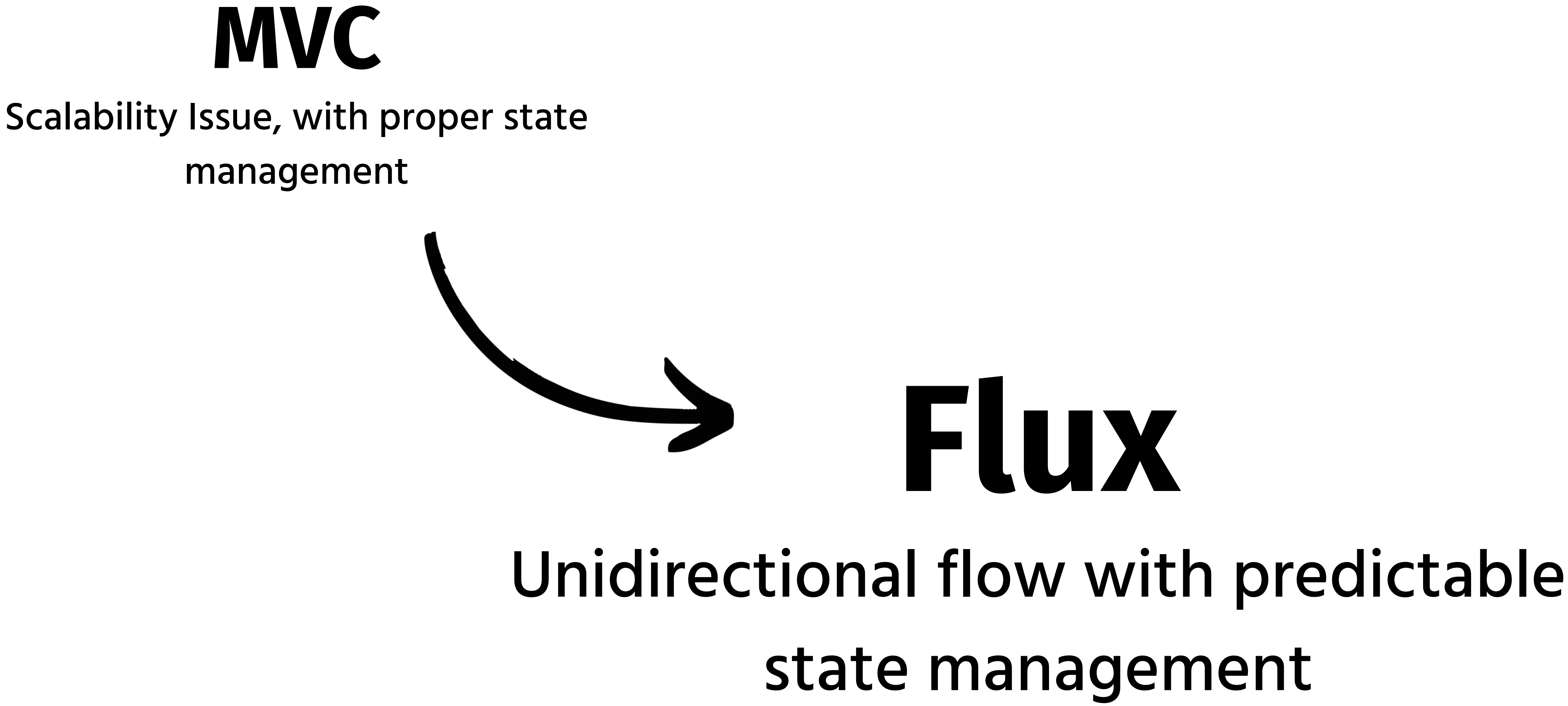
MVC

Scalability Issue, with proper state management

MVC

Scalability Issue, with proper state management

Flux

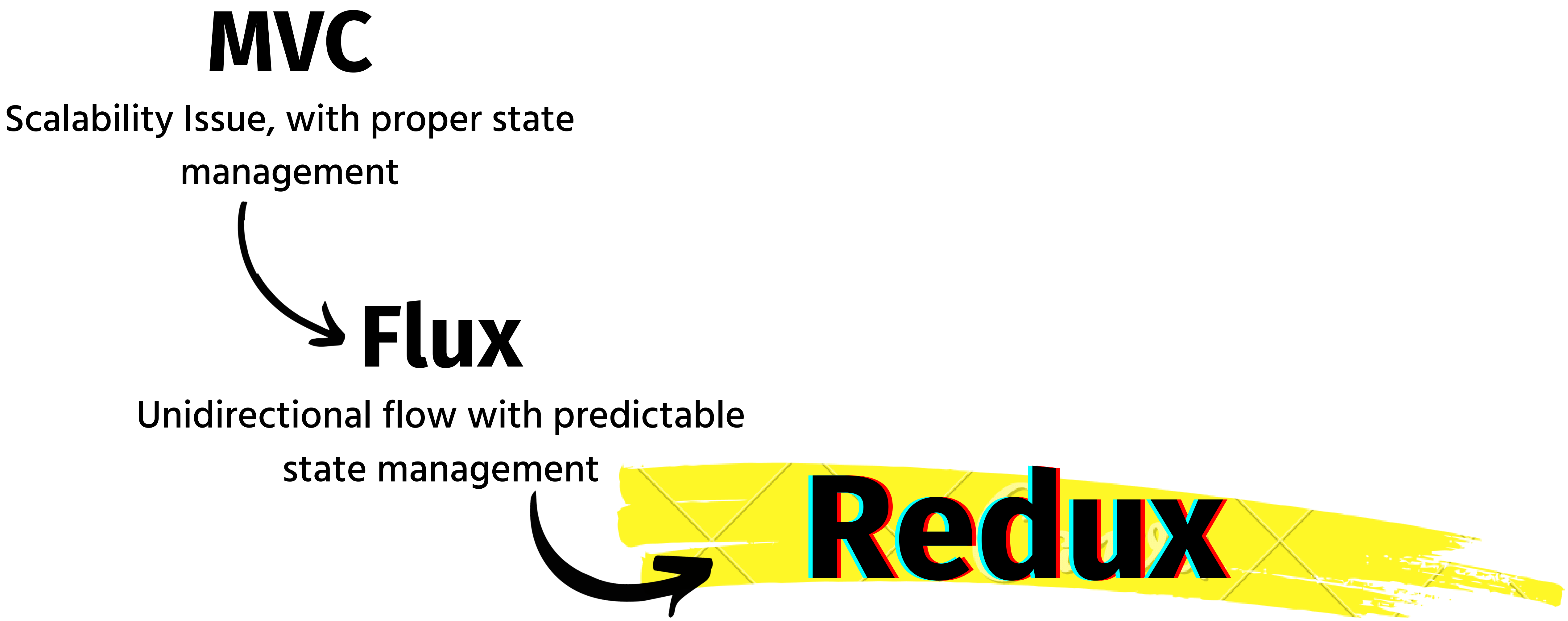


MVC

Scalability Issue, with proper state management

Flux

Unidirectional flow with predictable state management



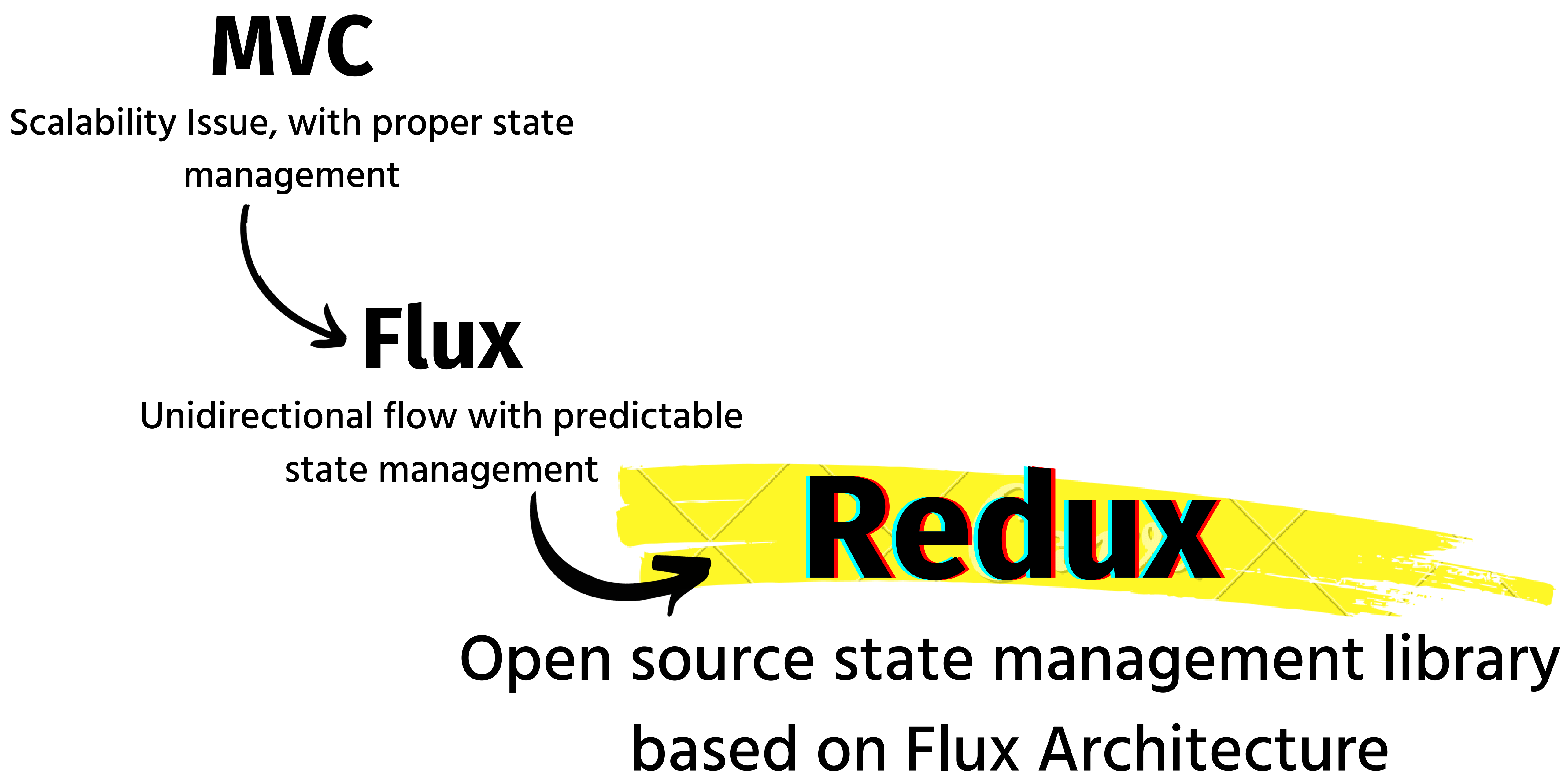
MVC

Scalability Issue, with proper state management

Flux

Unidirectional flow with predictable state management

Redux



MVC

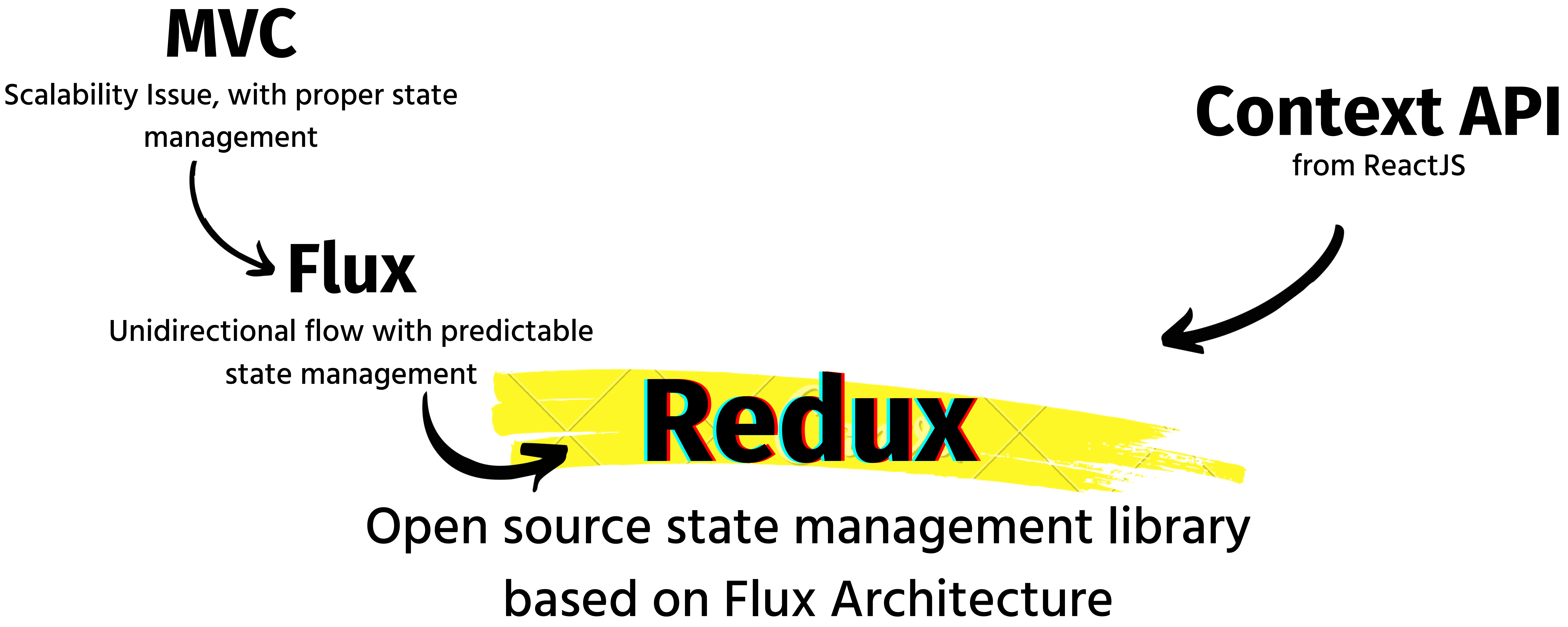
Scalability Issue, with proper state management

Flux

Unidirectional flow with predictable state management

Redux

Open source state management library based on Flux Architecture



MVC

Scalability Issue, with proper state management

Flux

Unidirectional flow with predictable state management

Context API

from ReactJS

Redux

Open source state management library based on Flux Architecture

What is Redux?

What is Redux?

A predictable and global state container/state management library for JavaScript applications, that follows flux pattern

What is Redux?

A predictable and global state container/state management library for JavaScript applications, that follows flux pattern

What is Redux?

A predictable and global state container/state management library for JavaScript applications, that follows flux pattern

JS

What is Redux?

A predictable and global state container/state management library for JavaScript applications, that follows flux pattern

JS

What is Redux?

A predictable and global state container/state management library for JavaScript applications, that follows flux pattern

What is Redux?

A predictable and global state container/state management library for JavaScript applications, that follows flux pattern

What is Redux?

A predictable and global state container/state management library for JavaScript applications, that follows flux pattern

What is Redux?

A predictable and global state container/state management library for JavaScript applications, that follows flux pattern

What is Redux?

A predictable and global state container/state management library for JavaScript applications, that follows flux pattern

JS
A
Vue.js
ember

What is Redux?

A predictable and global state container/state management library for JavaScript applications, that follows flux pattern

and many more...

What is Redux?

A predictable and global state container/state management library for JavaScript applications, that follows flux pattern

Used for "state management", just like,

`useState`

What is Redux?

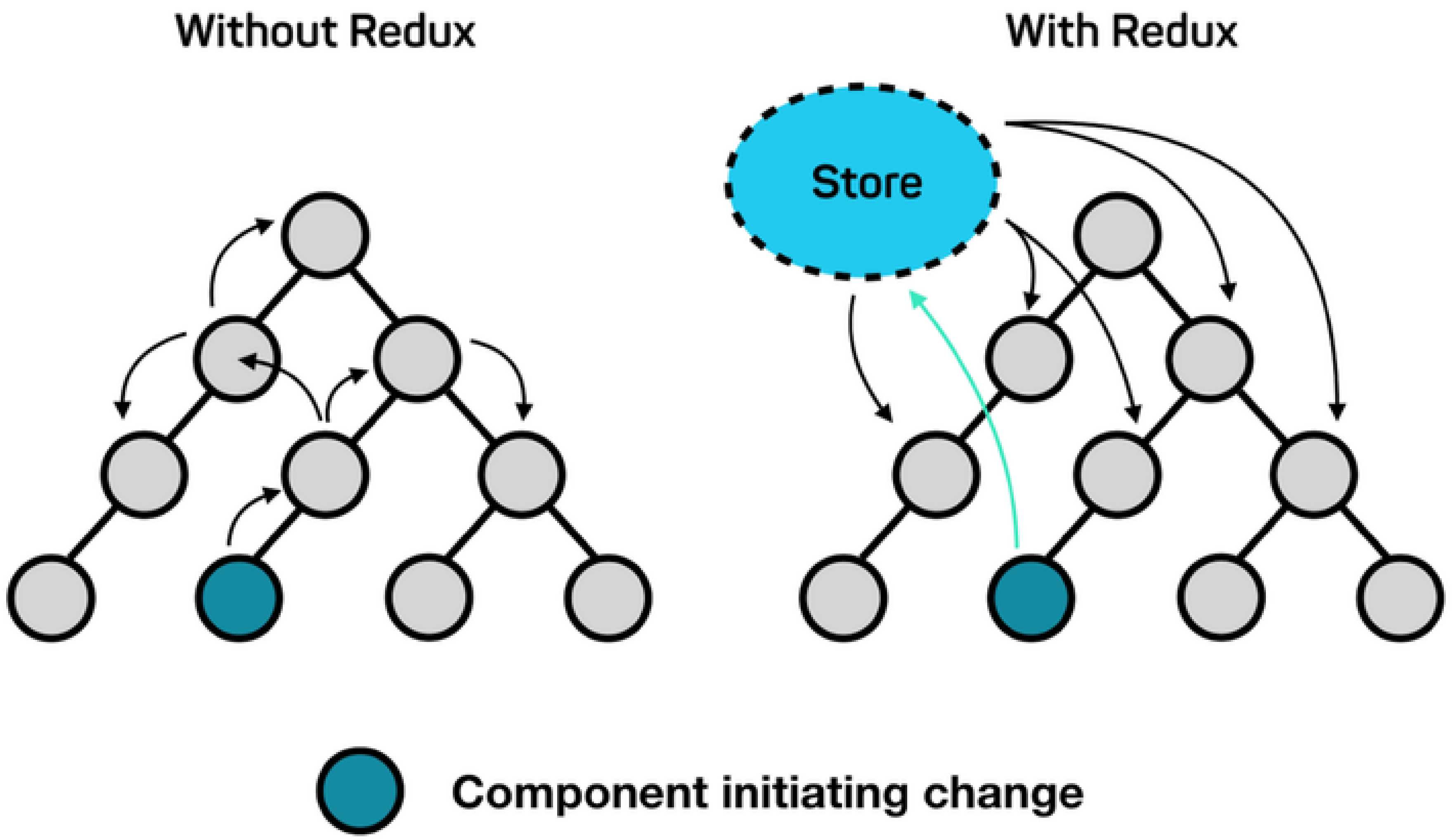
A predictable and global state container/state management library for JavaScript applications, that follows flux pattern

Used for "state management", just like,

`useState` & `useReducer`

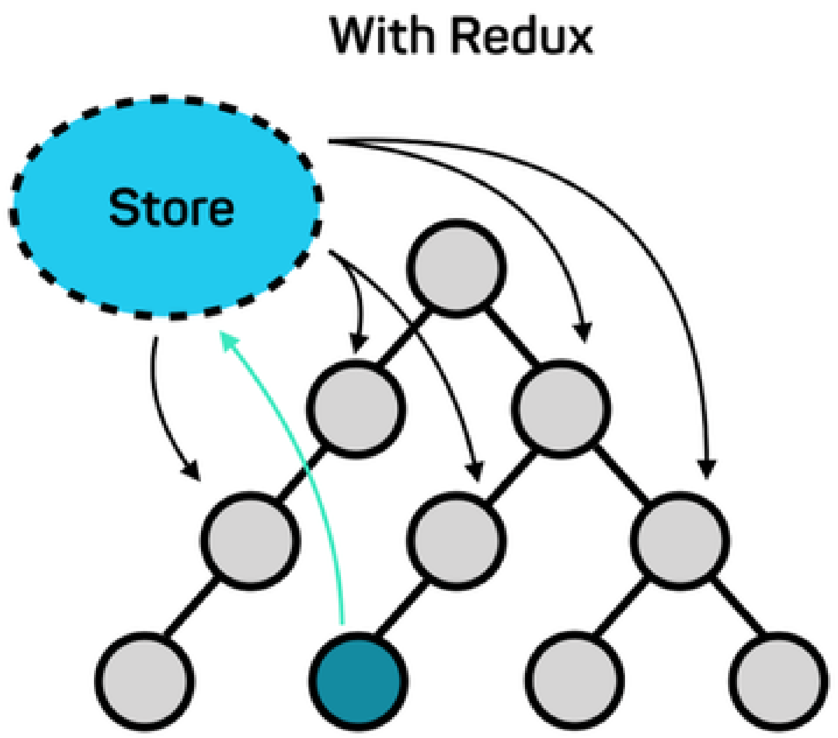
What is Redux?

A predictable and global state container/state management library for JavaScript applications, that follows flux pattern



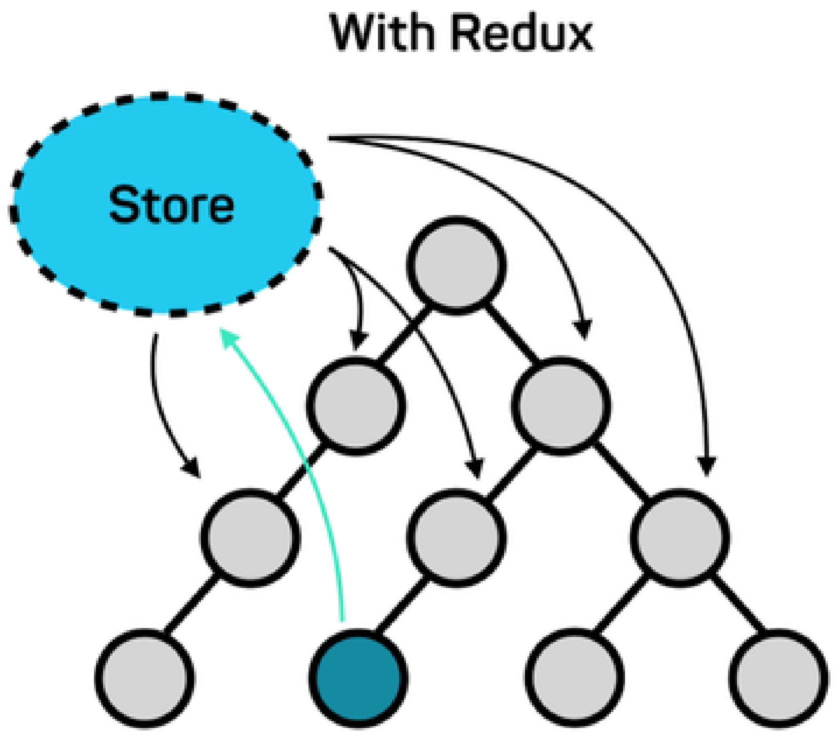
Redux

Isn't this similar to ContextAPI?



Redux

Isn't this similar to ContextAPI?

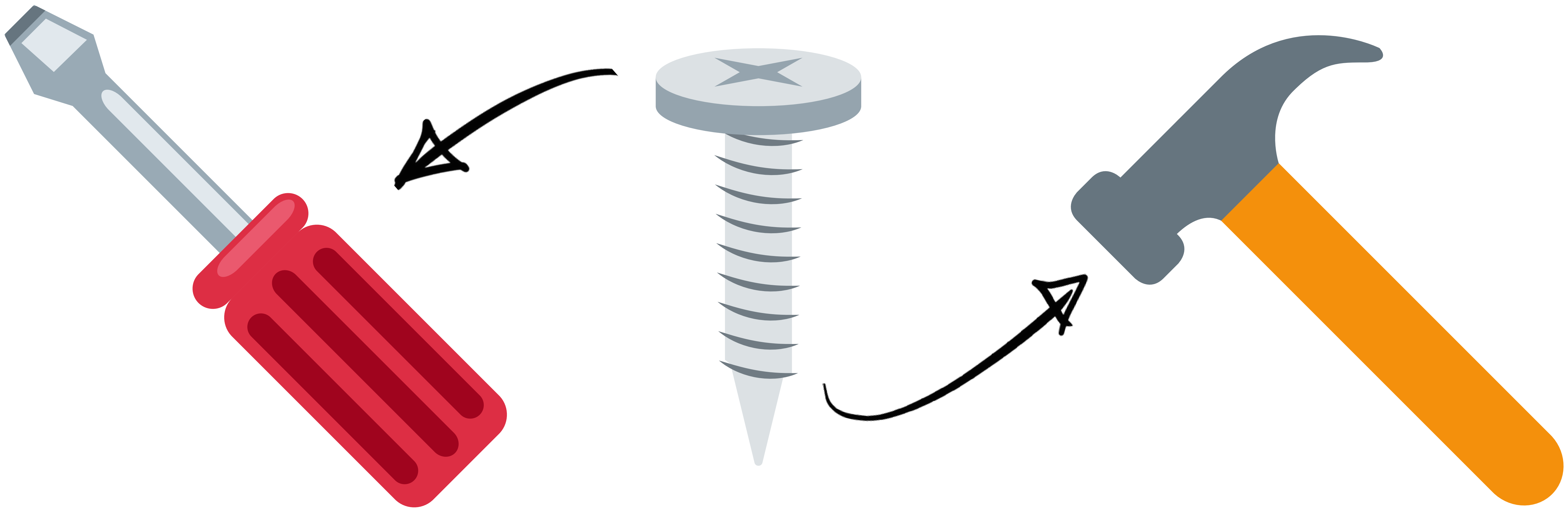


Well... kind of yes, but NO❌

know more about the differences here

Redux

"Different tools for different purpose"



So, why should we use REDUX?

Why should we use REDUX?

Why should we use REDUX?

Central State Management

Why should we use REDUX?

Central State Management

Why should we use REDUX?

Central State Management

Debugging

Why should we use REDUX?

Central State Management

Debugging

Why should we use REDUX?

Central State Management

Performance Optimization

Debugging

Why should we use REDUX?

Central State Management

Performance Optimization

Debugging

Why should we use REDUX?

Central State Management

Performance Optimization

Separation of Concern (Clean Code)

Debugging

Why should we use REDUX?

Central State Management

Performance Optimization

Debugging

Separation of Concern (Clean Code)

Why should we use REDUX?

Central State Management

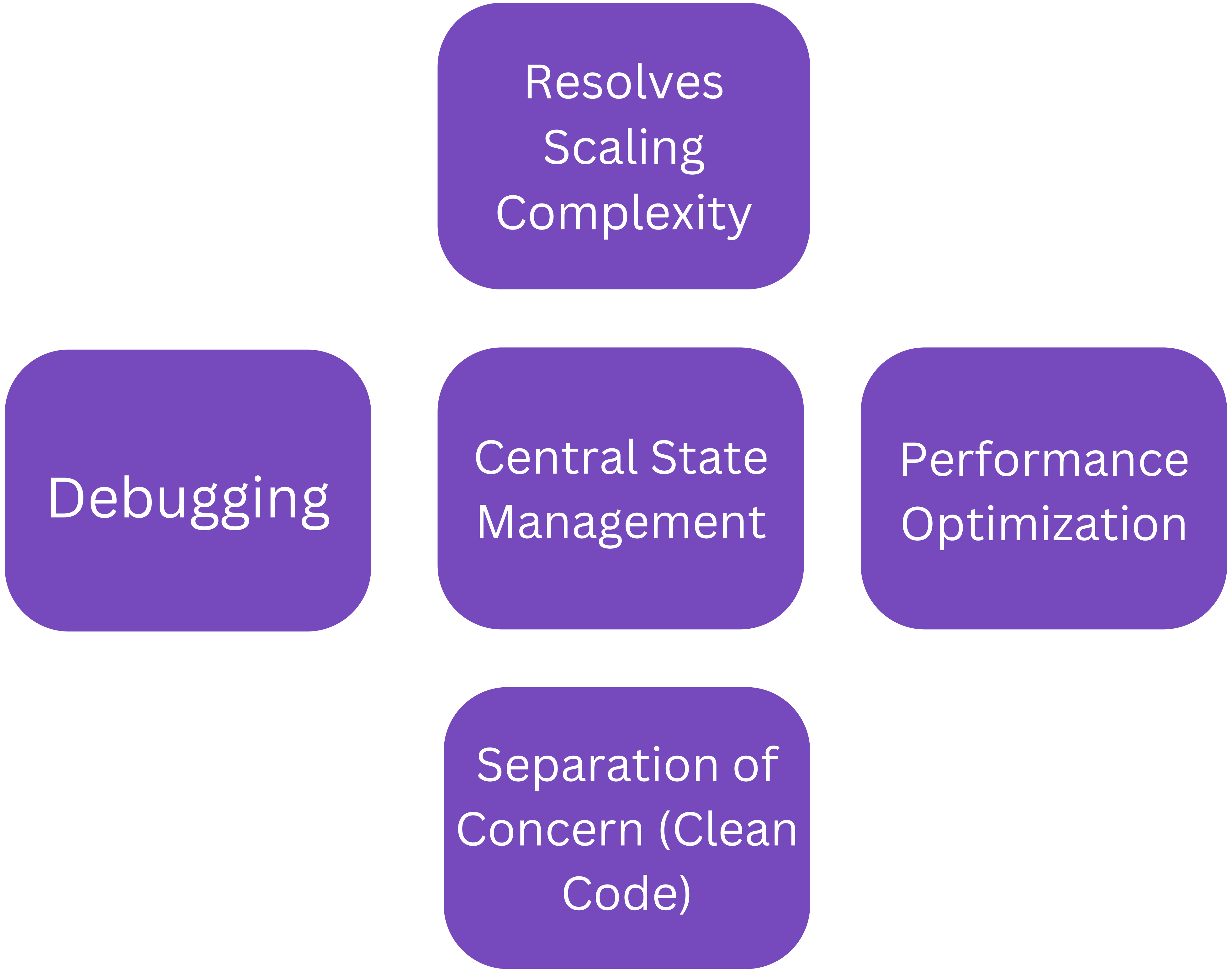
Performance Optimization

Resolves Scaling Complexity

Debugging

Separation of Concern (Clean Code)

Why should we use REDUX?



Resolves Scaling Complexity

Debugging

Central State Management

Performance Optimization

Separation of Concern (Clean Code)

Redux

3 Principles of Redux

Redux

3 Principles of Redux

SINGLE SOURCE OF TRUTH

Redux

3 Principles of Redux

SINGLE SOURCE OF TRUTH

STATE IS READ ONLY

Redux

3 Principles of Redux

SINGLE SOURCE OF TRUTH

STATE IS READ ONLY

CHANGES ARE MADE WITH PURE FUNCTIONS

Redux

3 Principles of Redux

SINGLE SOURCE OF TRUTH

STATE IS READ ONLY

CHANGES ARE MADE WITH PURE FUNCTIONS

Redux

3 Principles of Redux

SINGLE SOURCE OF TRUTH

STATE IS READ ONLY

CHANGES ARE MADE WITH PURE FUNCTIONS

3 Principles of Redux

SINGLE SOURCE OF TRUTH

3 Principles of Redux

SINGLE SOURCE OF TRUTH

"The global state of your application is stored in an object tree within a single store"

3 Principles of Redux

SINGLE SOURCE OF TRUTH

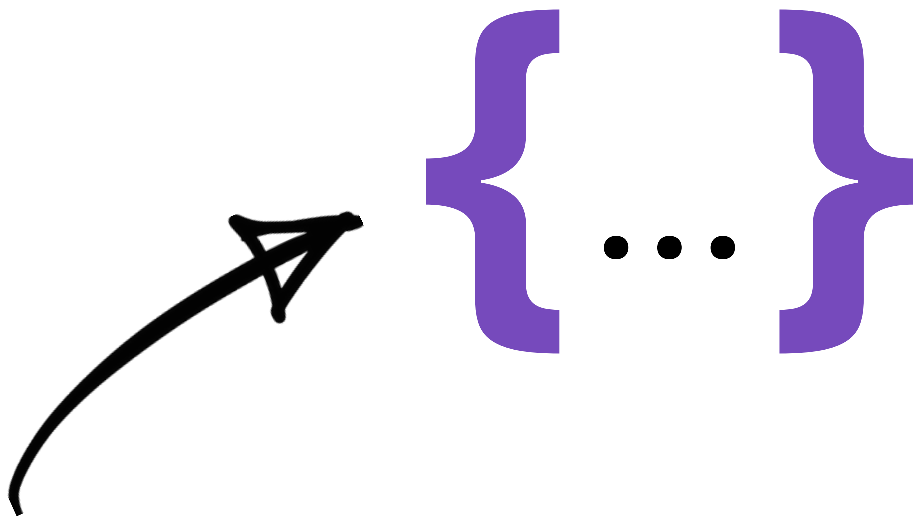
"The global state of your application is stored in an object tree within a single store"

{...}

3 Principles of Redux

SINGLE SOURCE OF TRUTH

"The global state of your application is stored in an object tree within a single store"



A single object that contains all the application data, at one place

3 Principles of Redux

STATE IS READ ONLY

3 Principles of Redux

STATE IS READ ONLY

"The only way to change the state is to emit an action, an object describing what happened."

3 Principles of Redux

STATE IS READ ONLY

"The only way to change the state is to emit an action, an object describing what happened."

```
obj['key'] = 'new_value'
```

3 Principles of Redux

STATE IS READ ONLY

"The only way to change the state is to emit an action, an object describing what happened."

```
obj['key'] = 'new_value'
```

or

```
obj.key = 'new_value'
```

3 Principles of Redux

STATE IS READ ONLY

"The only way to change the state is to emit an action, an object describing what happened."

`obj['key'] = 'new_value'` or `obj.key = 'new_value'`

3 Principles of Redux

STATE IS READ ONLY

"The only way to change the state is to emit an action, an object describing what happened."

`obj['key'] = 'new_value'` or `obj.key = 'new_value'`

3 Principles of Redux

STATE IS READ ONLY

"The only way to change the state is to emit an action, an object describing what happened."

obj['key'] ~~=~~ 'new_value'   or   obj.key ~~=~~ 'new_value'

The state can only be changed/modified, using **ACTIONS** and **REDUCERS**

3 Principles of Redux

CHANGES ARE MADE WITH PURE FUNCTIONS

3 Principles of Redux

CHANGES ARE MADE WITH PURE FUNCTIONS

"To specify how the state tree is transformed by actions we write pure reducers."

3 Principles of Redux

CHANGES ARE MADE WITH PURE FUNCTIONS

"To specify how the state tree is transformed by actions we write pure reducers."

PURE FUNCTIONS?

3 Principles of Redux

CHANGES ARE MADE WITH PURE FUNCTIONS

"To specify how the state tree is transformed by actions we write pure reducers."

PURE FUNCTIONS?

Predictable

3 Principles of Redux

CHANGES ARE MADE WITH PURE FUNCTIONS

"To specify how the state tree is transformed by actions we write pure reducers."

PURE FUNCTIONS?

Predictable

Without side-effects

3 Principles of Redux

CHANGES ARE MADE WITH PURE FUNCTIONS

"To specify how the state tree is transformed by actions we write pure reducers."

PURE FUNCTIONS?

Predictable

Should return the same output,
if the same input is provided

Without side-effects

3 Principles of Redux

CHANGES ARE MADE WITH PURE FUNCTIONS

"To specify how the state tree is transformed by actions we write pure reducers."

PURE FUNCTIONS?

Predictable

Should return the same output,
if the same input is provided

Without side-effects

They should not perform any
operations that are not
related/required for getting
the final output

3 Principles of Redux

CHANGES ARE MADE WITH PURE FUNCTIONS

"To specify how the state tree is transformed by actions we write pure reducers."

PURE FUNCTIONS?

Predictable & Without side-effects

3 Principles of Redux

CHANGES ARE MADE WITH PURE FUNCTIONS

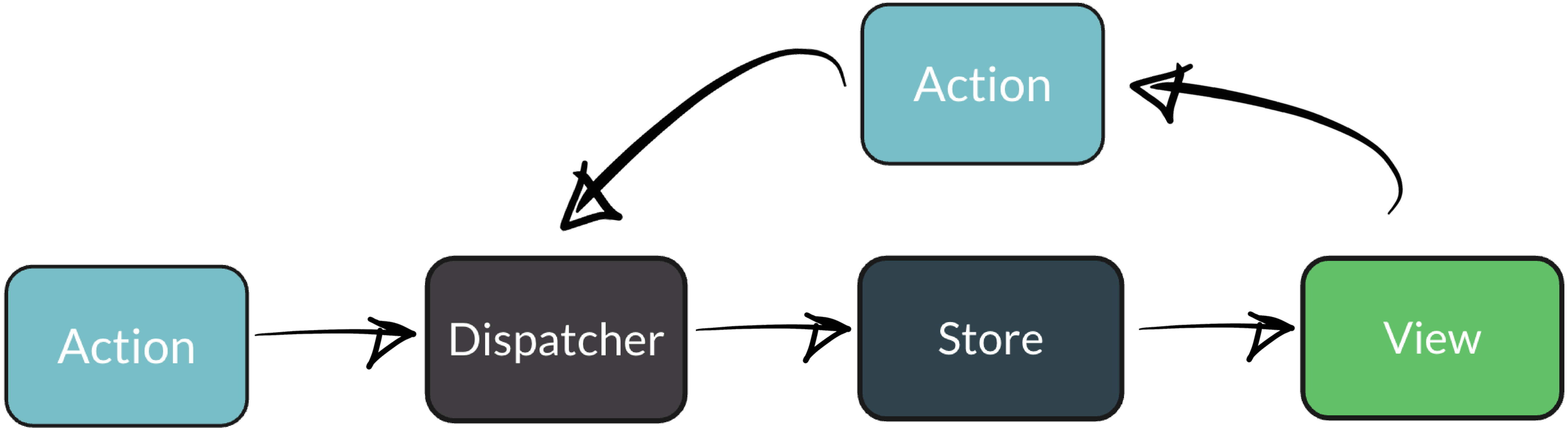
"To specify how the state tree is transformed by actions we write pure reducers."

PURE FUNCTIONS?

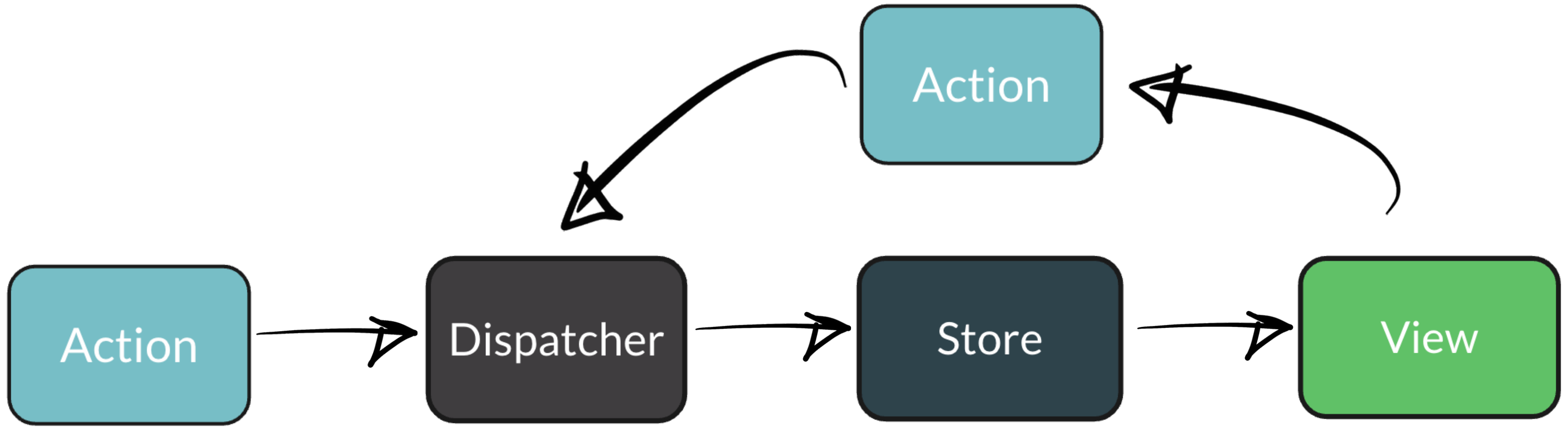
Predictable & Without side-effects

REDUCERS are pure functions that take the previous state, and the action object, and return the new state back

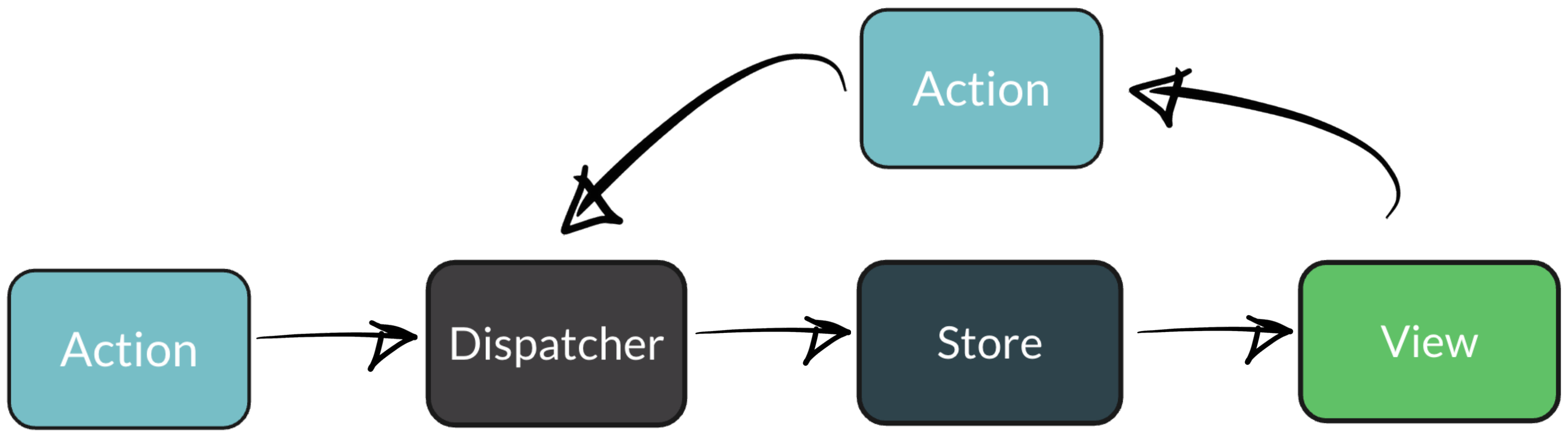
Redux Flow



Redux Flow

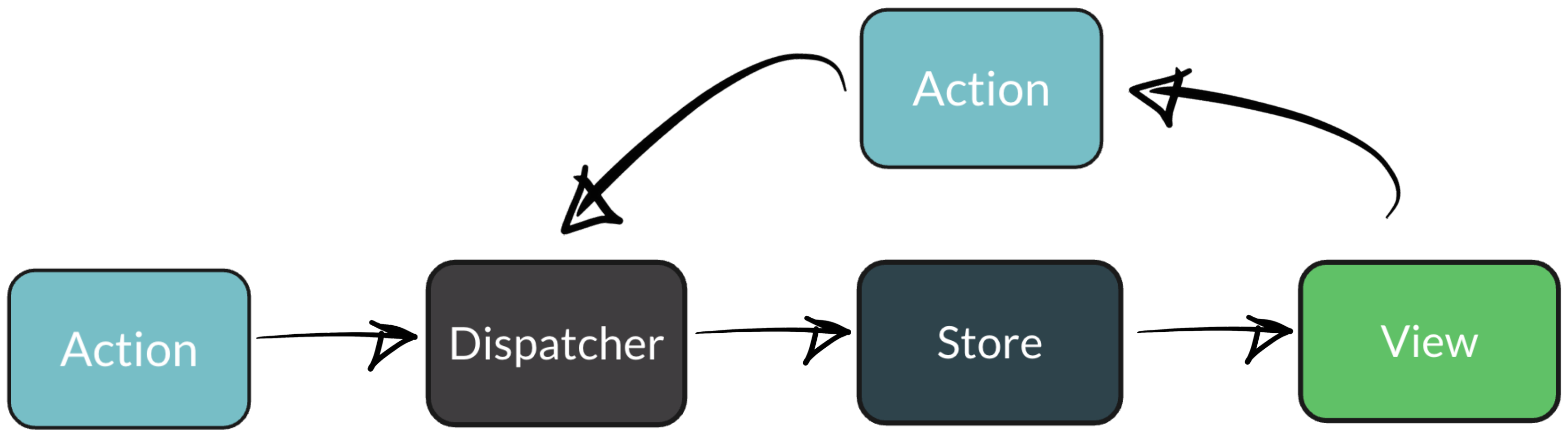


Redux Flow

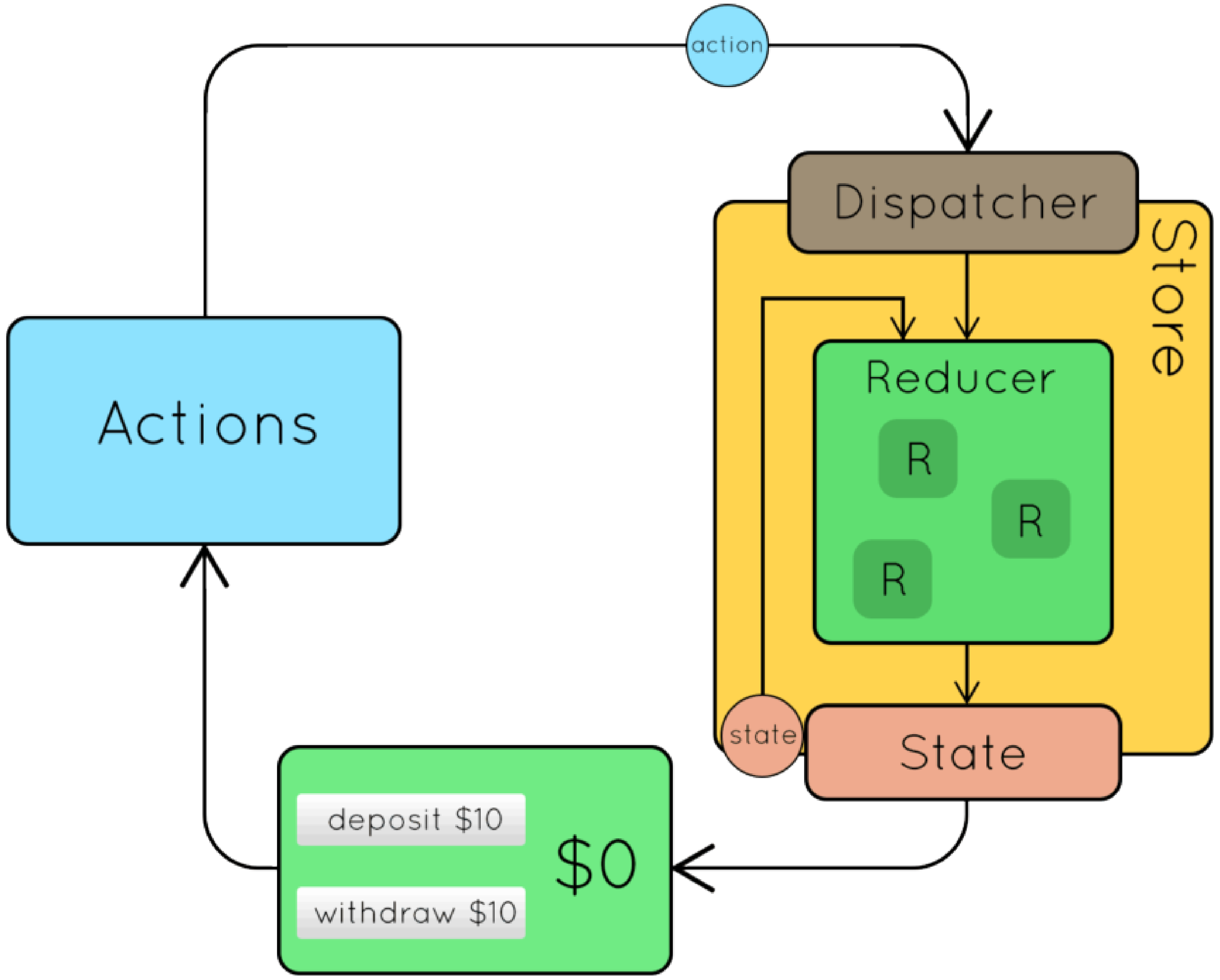


Flux Architecture

Redux Flow



Flux Architecture



Redux Architecture

Core Parts of Redux

Core Parts of Redux

ACTIONS

Redux

Core Parts of Redux

ACTIONS

DISPATCHERS

Redux

Core Parts of Redux

ACTIONS

DISPATCHERS

REDUCERS

Core Parts of Redux

ACTIONS

DISPATCHERS

REDUCERS

STORE

Redux

Core Parts of Redux

ACTIONS

DISPATCHERS

REDUCERS

STORE

Core Parts of Redux

ACTIONS

DISPATCHERS

REDUCERS

STORE

Core Parts of Redux

ACTIONS

Core Parts of Redux

ACTIONS

They are simple plain objects, that send data from the **application** to the **redux store**

Core Parts of Redux

ACTIONS

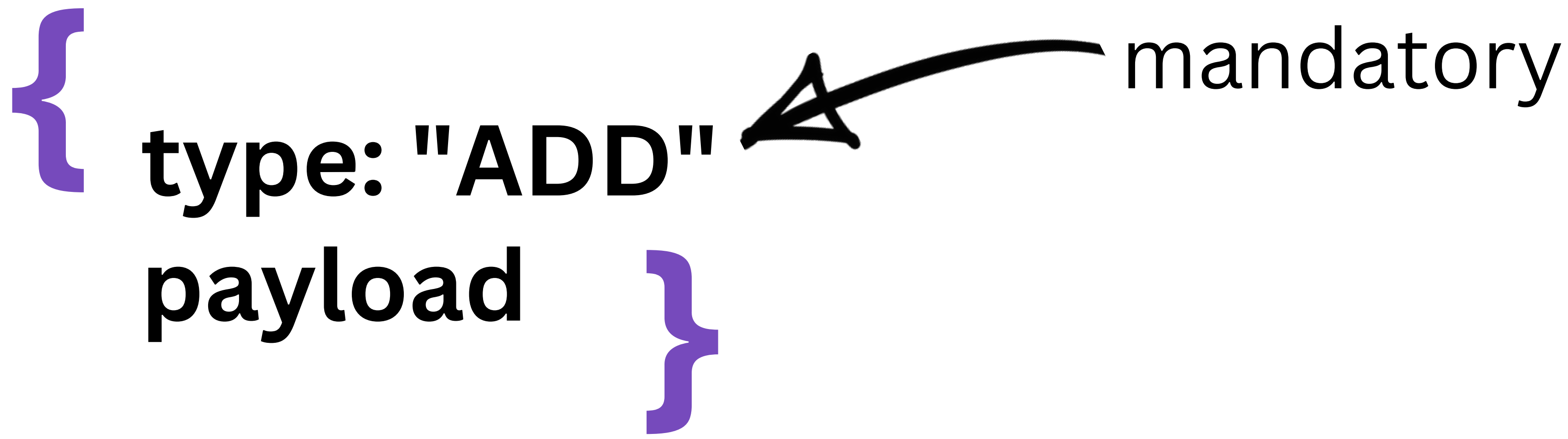
They are simple plain objects, that send data from the **application** to the **redux store**

```
{
  type: "ADD"
  payload
}
```

Core Parts of Redux

ACTIONS

They are simple plain objects, that send data from the **application** to the **redux store**



{

type: "ADD"

payload

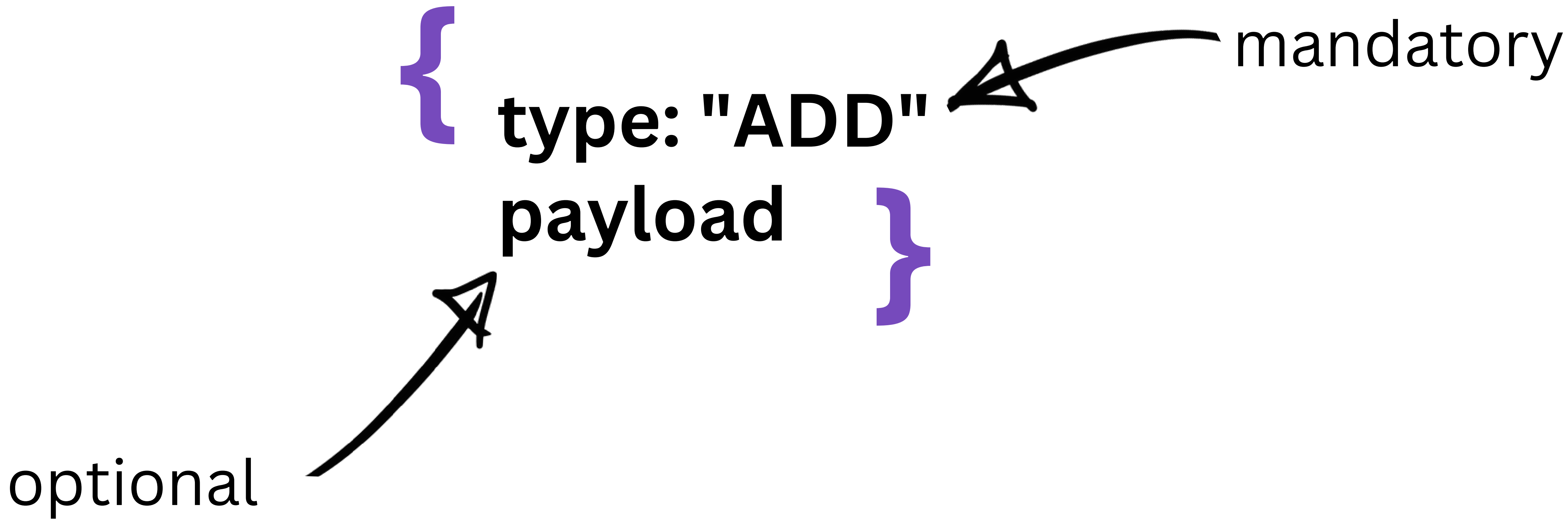
}

mandatory

Core Parts of Redux

ACTIONS

They are simple plain objects, that send data from the **application** to the **redux store**



{

type: "ADD"

payload

}

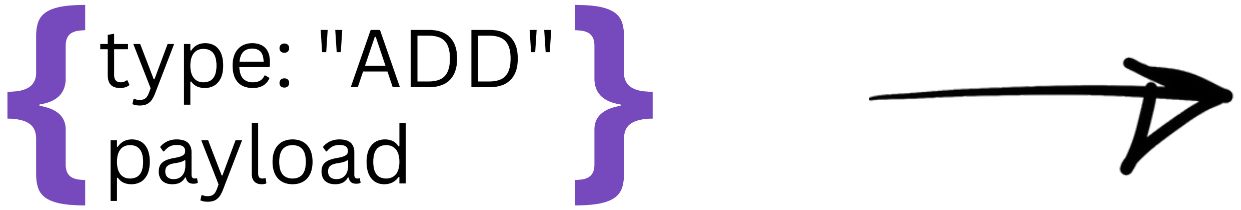
mandatory

optional

Core Parts of Redux

ACTIONS

They are simple plain objects, that send data from the **application** to the **redux store**



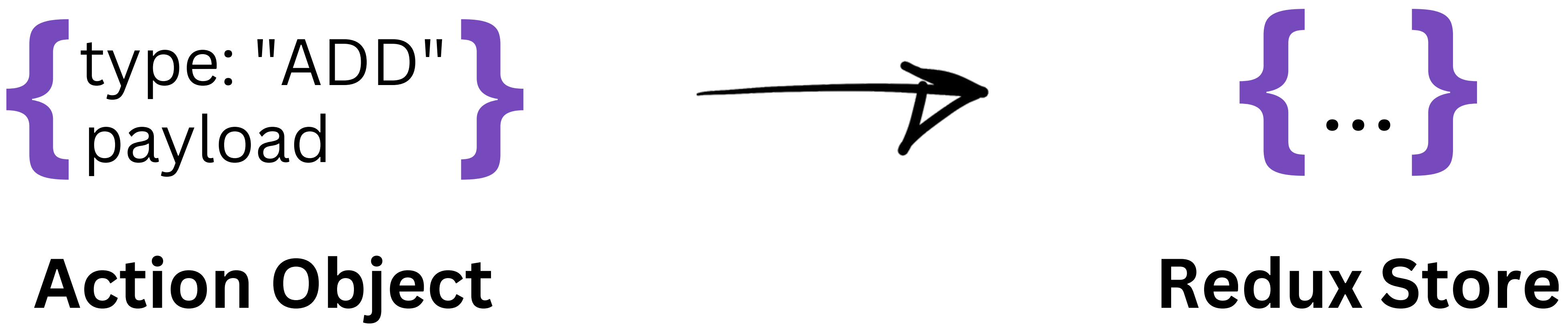
```
{
  type: "ADD"
  payload
}
```

Action Object

Core Parts of Redux

ACTIONS

They are simple plain objects, that send data from the **application** to the **redux store**



{type: "ADD"
payload }

Action Object

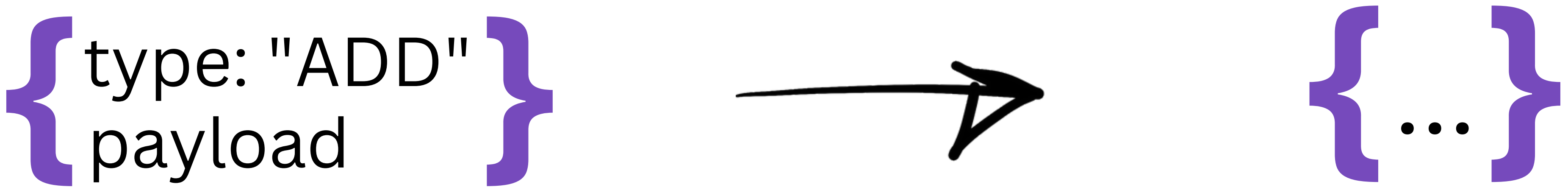
{...}

Redux Store

Core Parts of Redux

ACTIONS

They are simple plain objects, that send data from the **application** to the **redux store**



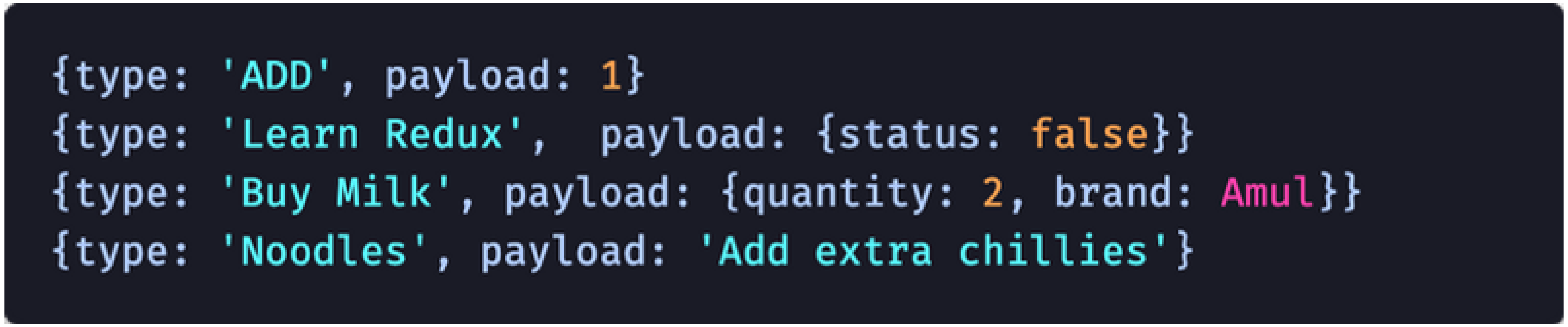
type: "ADD"
payload

...

Action Object

Redux Store

Ex:

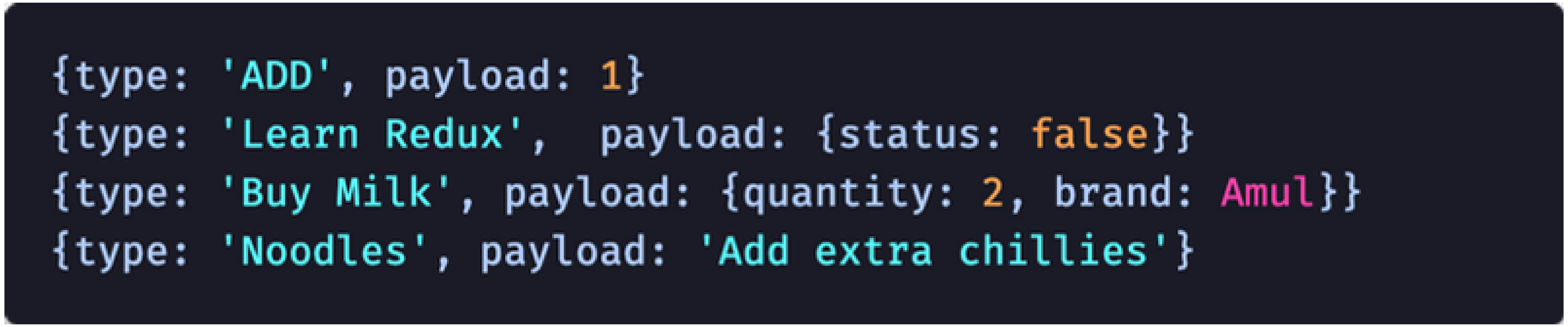


```
{type: 'ADD', payload: 1}
{type: 'Learn Redux',  payload: {status: false}}
{type: 'Buy Milk', payload: {quantity: 2, brand: Amul}}
{type: 'Noodles', payload: 'Add extra chillies'}
```

Core Parts of Redux

ACTIONS

They are simple plain objects, that send data from the **application** to the **redux store**



```
{type: 'ADD', payload: 1}
{type: 'Learn Redux',  payload: {status: false}}
{type: 'Buy Milk', payload: {quantity: 2, brand: Amul}}
{type: 'Noodles', payload: 'Add extra chillies'}
```

"actions only describe what happened, but don't describe how the application's state changes"

Core Parts of Redux

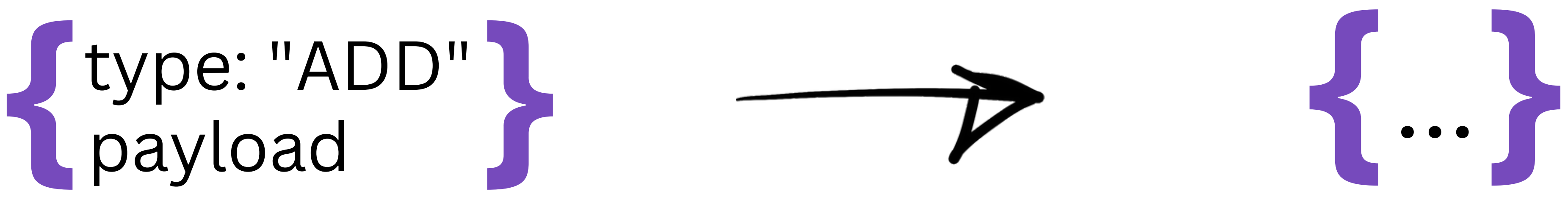
DISPATCHERS

They are the only way to take the **action objects** from the application to the **Redux store**

Core Parts of Redux

DISPATCHERS

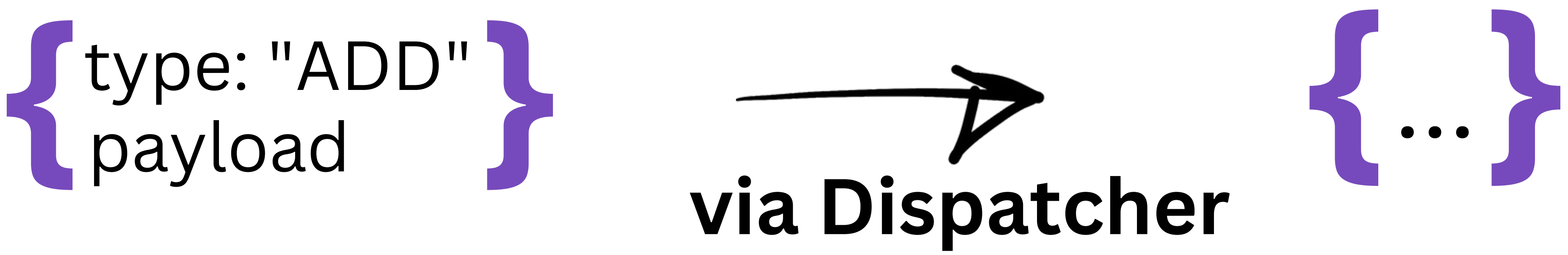
They are the only way to take the **action objects** from the application to the **Redux store**



Core Parts of Redux

DISPATCHERS

They are the only way to take the **action objects** from the application to the **Redux store**



{type: "ADD"
payload}

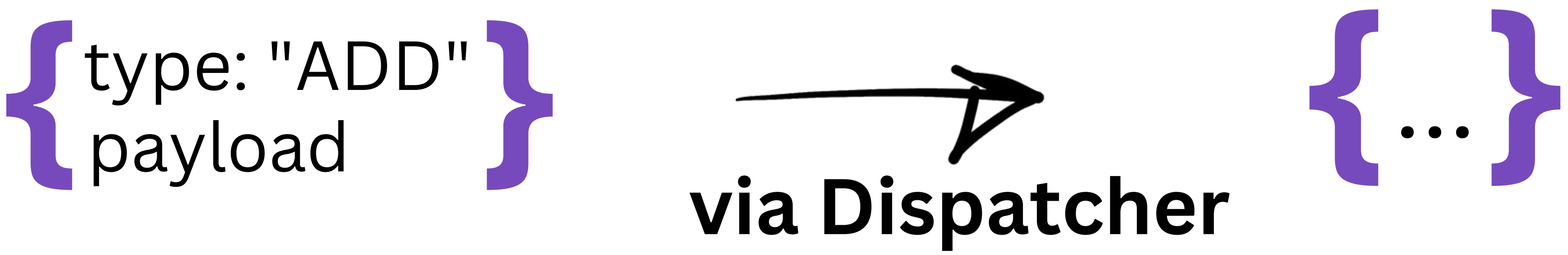
via Dispatcher

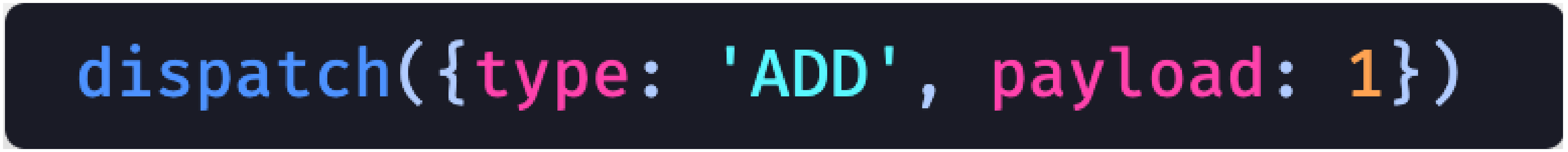
{...}

Core Parts of Redux

DISPATCHERS

They are the only way to take the **action objects** from the application to the **Redux store**





```
dispatch({type: 'ADD', payload: 1})
```

Core Parts of Redux

REDUCERS

A **pure function**, that gets invoked every time an action object is dispatched

Core Parts of Redux

REDUCERS

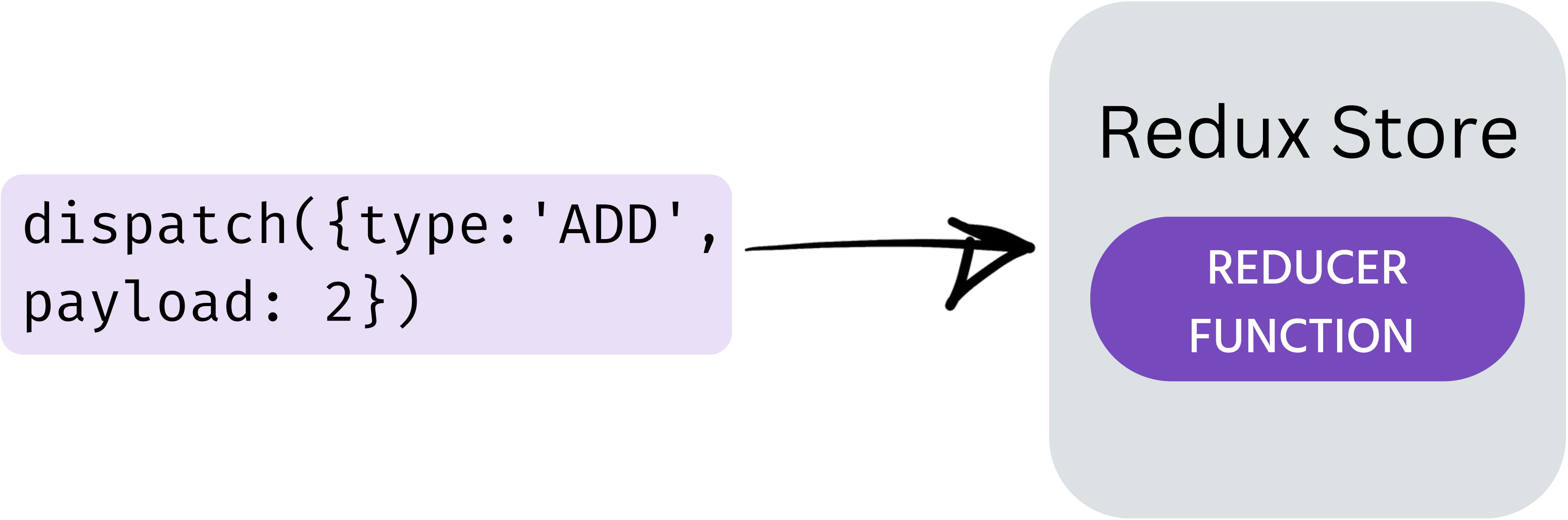
A **pure function**, that gets invoked every time an action object is dispatched

```
dispatch({type:'ADD',
payload: 2})
```

Core Parts of Redux

REDUCERS

A **pure function**, that gets invoked every time an action object is dispatched

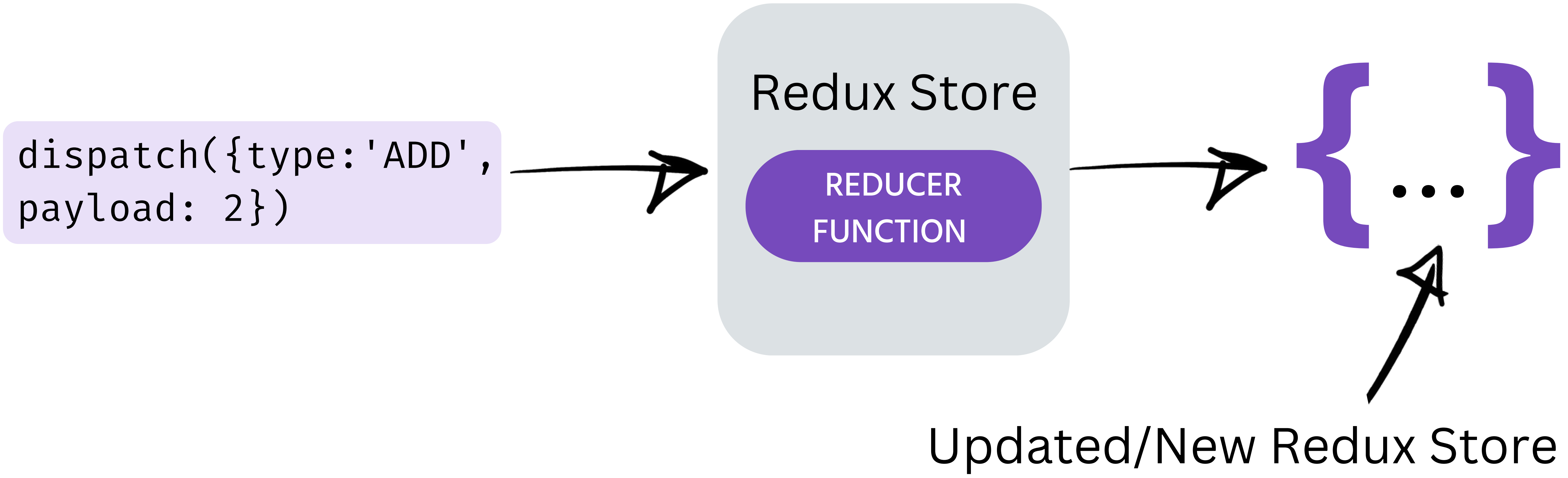


```
dispatch({type:'ADD',
payload: 2})
```

Core Parts of Redux

REDUCERS

A **pure function**, that gets invoked every time an action object is dispatched



```
dispatch({type:'ADD',
payload: 2})
```

Core Parts of Redux

STORE

The whole **state tree** of our application can be stored, inside **an object** inside redux, known as the store

Core Parts of Redux

STORE

The whole **state tree** of our application can be stored, inside **an object** inside redux, known as the store

KEY RESPONSIBILITIES OF THE STORE:

1. To hold the application state

Core Parts of Redux

STORE

The whole **state tree** of our application can be stored, inside **an object** inside redux, known as the store

KEY RESPONSIBILITIES OF THE STORE:

1. To hold the application state
2. Allow access to the application state

Core Parts of Redux

STORE

The whole **state tree** of our application can be stored, inside **an object** inside redux, known as the store

KEY RESPONSIBILITIES OF THE STORE:

1. To hold the application state
2. Allow access to the application state
3. Allow the state to be updated via the dispatch method containing the. action object.

Core Parts of Redux

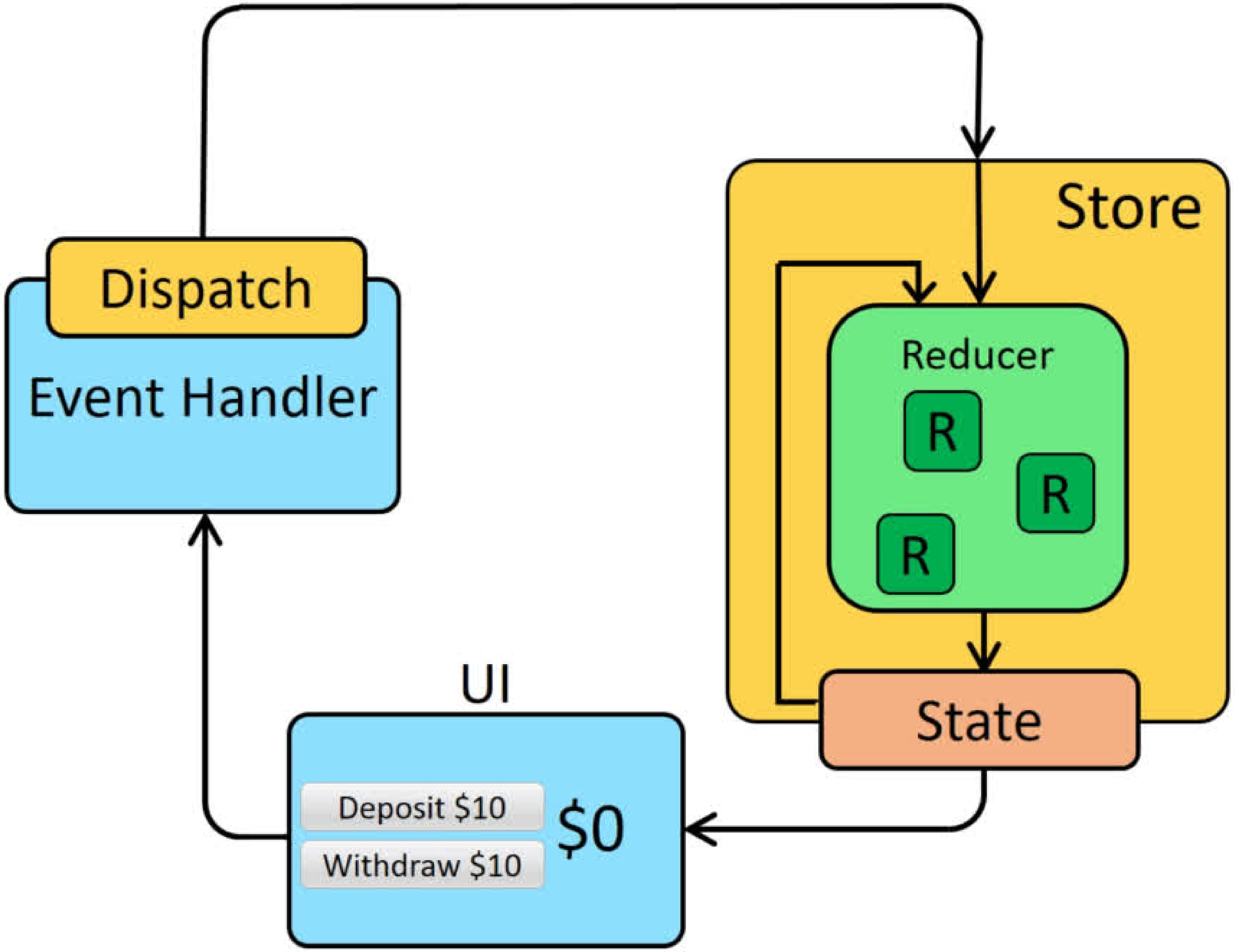
STORE

The whole **state tree** of our application can be stored, inside **an object** inside redux, known as the store

KEY RESPONSIBILITIES OF THE STORE:

1. To hold the application state
2. Allow access to the application state
3. Allow the state to be updated via the dispatch method containing the. action object.
4. Allows the application to listen to changes in the state.

Redux


Store
Reducer
R
R
R
State
Dispatch
Event Handler
UI
Deposit $10
Withdraw $10
$0