/academeez

Redux

1. about Redux

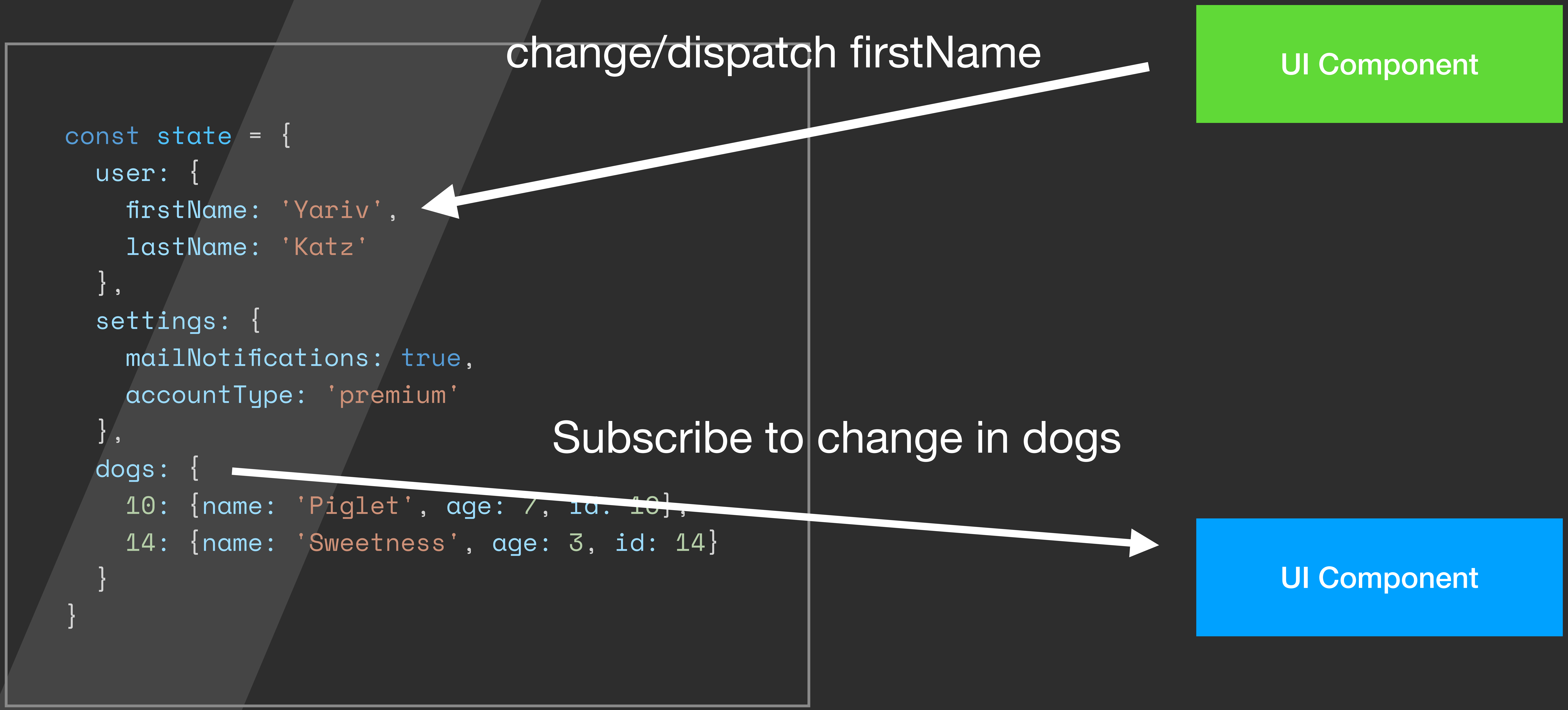
Redux

academeez

- Redux is a popular library for state management.
- State contains data shared between your UI components
- Think of the data as a javascript object
- That object contains sections which holds data
- A UI Component can choose to subscribe to a change in the data

Redux state

- Our redux state might look like this:



```
const state = {
  user: {
    firstName: 'Yariv',
    lastName: 'Katz'
  },
  settings: {
    mailNotifications: true,
    accountType: 'premium'
  },
  dogs: {
    10: {name: 'Piglet', age: 7, id: 10},
    14: {name: 'Sweetness', age: 3, id: 14}
  }
}
```

Store

academeez

- In redux the state is held in an object called **Store**
- There is one **Store** in a redux app
- The store hides that state
- The state can change using a method in the store called **dispatch**

```
export interface Store<S = any, A extends Action = AnyAction> {
  dispatch: Dispatch<A>
  getState(): S
  subscribe(listener: () => void): Unsubscribe
}
```

Reducers

academeez

- Our redux state is divided to sections where each section is managed by a **reducer**
- You might remember the concept of reducer from array
- Reducer example in array:

```
const numbersToSum = [1, 2, 3, 4, 5];
const reducer = (sum, currentValue) => sum + currentValue;

numbersToSum.reduce(reducer); // 15
numbersToSum.reduce(reducer, 5); // 20
```

Reducers

academeez

- Our reducer is a function that gets the sum of previous changes, the current iteration value, and calculates the next sum of values
- A reducer is redux takes the sum of all the state change - meaning the current state
- The reducer than gets the current action that is happening
- The reducer will than calculate the next state

```
export type Reducer<S = any, A extends Action = AnyAction> = (
  state: S | undefined,
  action: A
) => S
```

Reducer in Redux

academeez

- Our state is split to sections (user, settings, dogs)
- Each section will have a reducer that will manage that section
- The reducer will get the sum (the current state section)
- It will take an action
- It will calculate the next state section

Example of reducer in redux

academeez

- A reducer that manage the part of **user** in our state might look like this:

```
interface UserState {
  firstName: string;
  lastName: string
}

interface Action {
  type: string;
  payload: any;
}

// action might be: {type: 'change firstName', payload: 'piglet'}
function userReducer(state: UserState, action: Action): UserState {
  //...
}
```

Action

academeez

- An Action describes something that happened in our app that will cause a change in the state
- An action might be, user pressed a button, server returned response, etc.
- An action in redux is a simple object that describes what happened and contains data needed to change the state.

```
const chageFirstNameAction = {
  type: 'change firstName',
  payload: 'Piglet'
}

const chageLatNameAction = {
  type: 'change lastName',
  payload: 'Chaitovsky'
}
```

Reducer + Action

academeez

- The reducer will get the current state section he manages
- The reducer will get the action that happened
- The reducer will return the next state he manages

```
function userReducer(state: UserState, action: Action): UserState {
  switch(action.type) {
    case 'change firstName':
      return {...state, firstName: action.payload};
    case 'change lastName':
      return { ...state, lastName: action.payload };
    default:
      return state;
  }
}
```

Reducer initial state

academeez

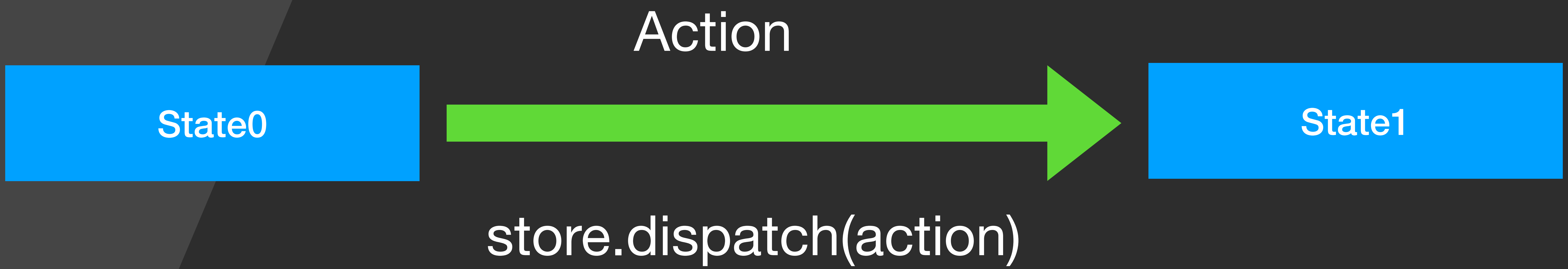
- The **reduce** method can determine an initial state to begin with.
- In Redux when the store is initialised it will call every reducer with a state undefined
- Whatever our reducer will return will be the initial state

```
const initialState = {
  firstName: 'Yariv',
  lastName: 'Katz'
}

function userReducer(state: UserState, action: Action): UserState {
  switch (action.type) {
    case ...
    default:
      return state;
  }
}
```

State change

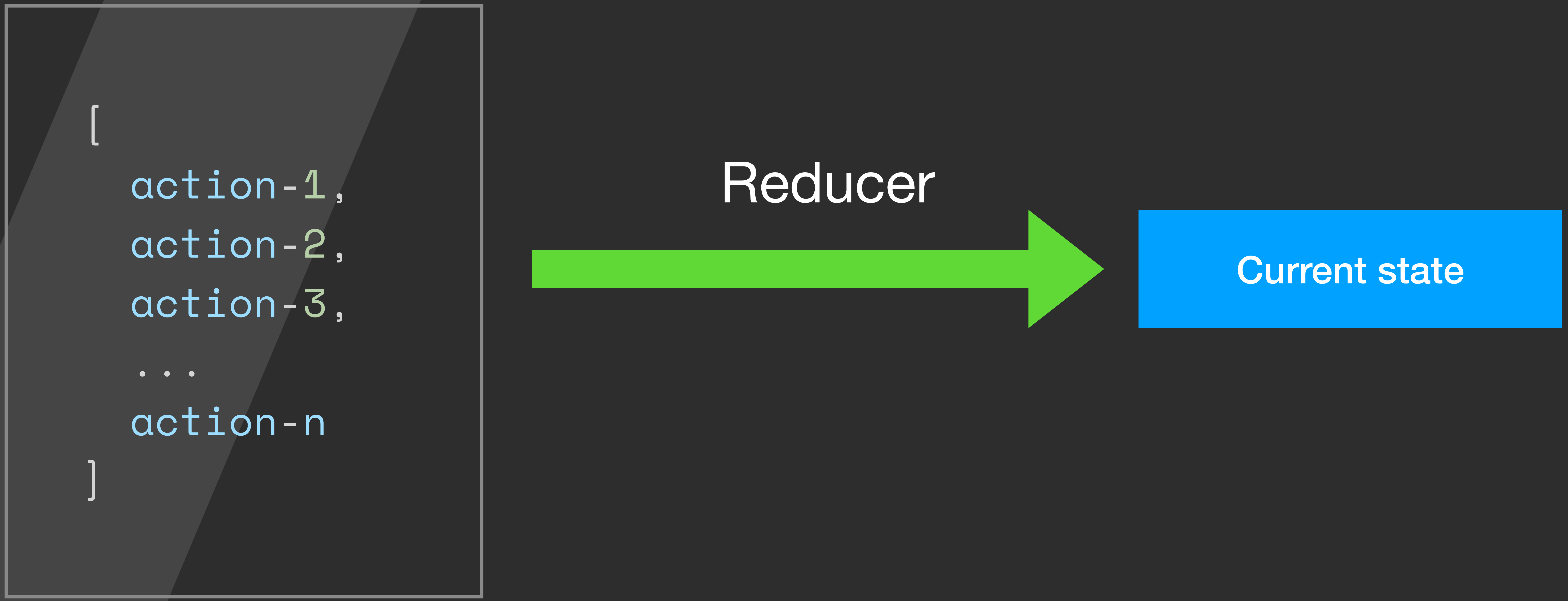
academeez



Reducer

academeez

- The array that the reducer is reducing:

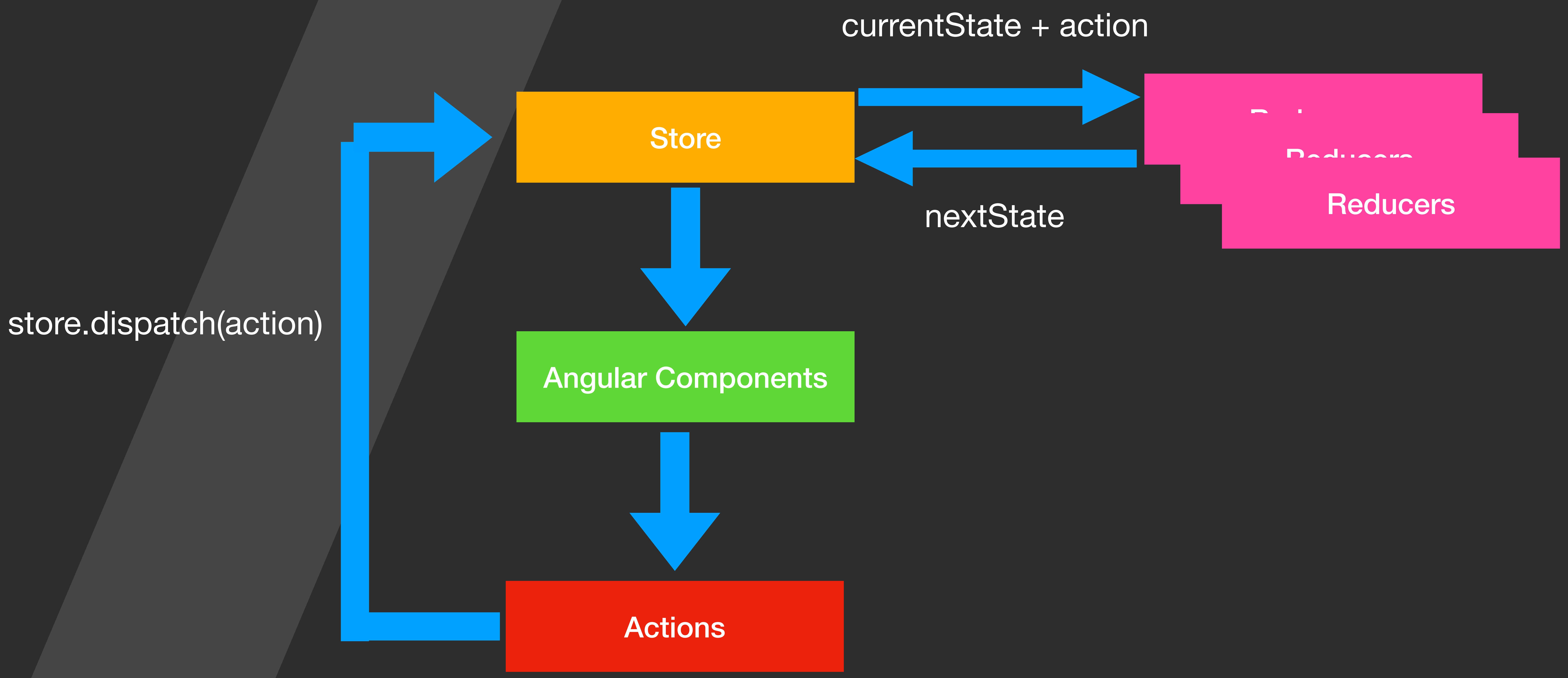


```
[
  action-1,
  action-2,
  action-3,
  ...
  action-n
]
```

How redux works with Angular

academeez

- In an Angular app the flow of redux will look like this diagram



Should I use redux?

- Will you have a lot of data and a lot of components listening to that data change?
- Will you need to return the user to the same state he left
- Do you often need to track the state change
- Do you need community tools for dealing with data change?
- Redux comes with a not easy learning curve, and a bit more code and files to manage state change

Summary

academeez

- If the data shared between components will grow larger, we will benefit from managing that data in a predictable easy to debug way.
- Redux gives us tools to manage our state and state changes
- Components in our angular app will choose to listen to the data relevant to them
- They will get modified with the change in the part of data they choose to listen to

academeez

Thank You

Next Lesson: 2. @ngrx/*