academeez

Redux - @ngrx/*

3. @ngrx/store

@ngrx/store

academeez

- The core implementation of Redux is in this library
- Will provide the tools to create
    - Actions
    - Reducers
    - Selectors
    - Store service
- In this lesson we will go over these basic library tools, what is the job of each one in a Redux implementation, and how they help us manage the data

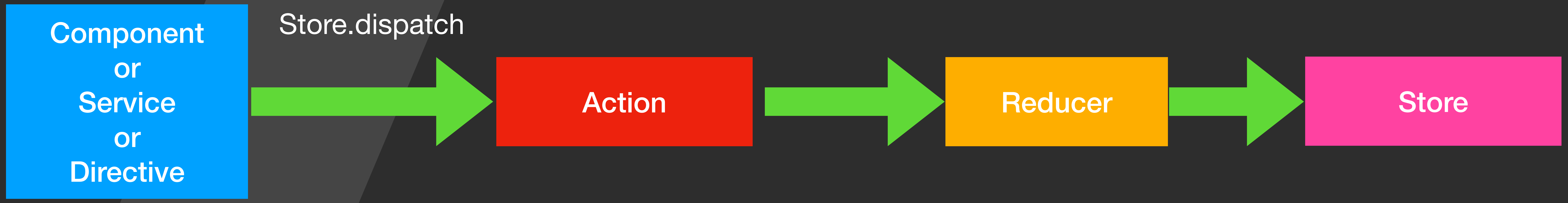
@ngrx/store flow

academeez

- Reducers determine the state sections and how they will change
- Actions describe a change we want to perform in the state
- The store service dispatch an action with store.dispatch
- The reading of data is done with selectors

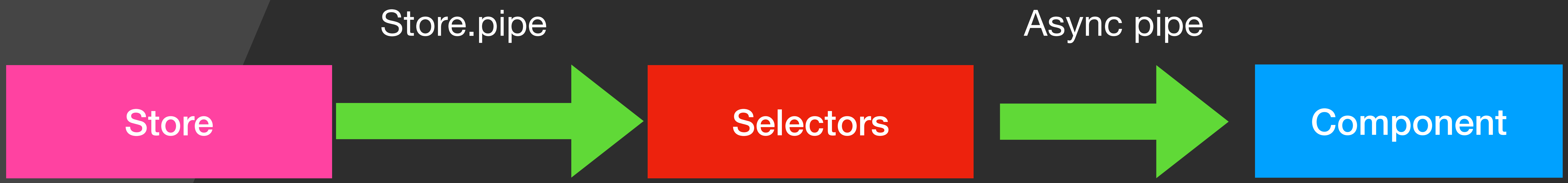
@ngrx/store change data

academeez



@ngrx/store read data

academeez



@ngrx/store - change/read state

academeez

- To understand the flow of an angular app that manage the data with @ngrx/* we will create a small hello world app using ngrx
- A component will display a message save in the ngrx store
- A different component will be able to change the message in the store
- This small ex will allow us to understand the basics of
    - state
    - store
    - actions
    - reducers
    - selectors

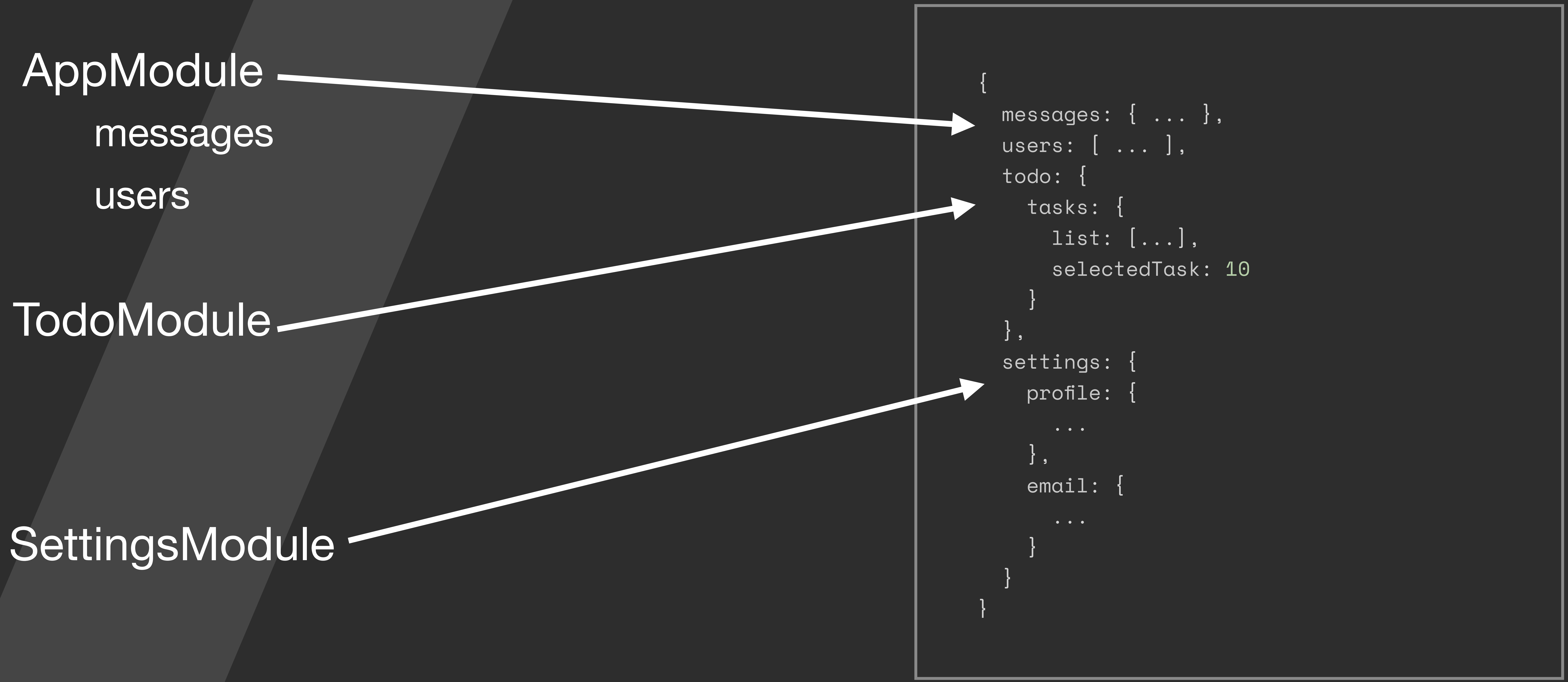
1. State

- The state is an object with different sections
- Each section is well defined using an interface
- Those section interface when combined defines our global state
- A feature module that adds data to @ngrx usually opens a section in the state
- Each section will have a reducer that is in charge of that section
- In the AppModule we have the root sections
- Each module can open a feature section

Example of State by modules

academeez

- Our app contains the following modules where each on will add data to the state



```
{
  messages: { ... },
  users: [ ... ],
  todo: {
    tasks: {
      list: [...],
      selectedTask: 10
    }
  },
  settings: {
    profile: {
      ...
    },
    email: {
      ...
    }
  }
}
```

1. State EX

academeez

- the app module will have a state with multiple section that each one has a reducer in charge of that section
- a feature module that want to store data in the store will open a section for that module
    - that section will contain multiple subsections that each one will have a reducer that is in charge of that section.
- Create a section state that will be called **message** and will contain an hello message

```
{
    message: {
        hello: 'hello world'
    }
}
```

Actions

academeez

- A change in our state must come from an Action
- An action describes the change and pass needed params for the state change
- The action is created using the method **createAction**
- An action has a unique name, and optional params needs for the state change

```
import { createAction, props } from '@ngrx/store';

export const changeMessage = createAction(
  '[message] Change Message',
  props<{message: string}>()
);
```

Reducer

academeez

- The action is passed to the reducers via **store.dispatch(action)**
- The reducer will decide if the state will change and how
- We create the reducer with the method: **createReducer**

```
import { createReducer, on, Action } from '@ngrx/store';
import { changeMessage } from '../actions/message.actions';

const initialState = 'hello world'

const featureReducer = createReducer(
  initialState,
  on(changeMessage, (state, action) => action.message ),
);

export function reducer(state: string, action: Action) {
  return featureReducer(state, action);
}
```

Changing the state

academeez

- To change the state we ask for the **Store** service and call the **dispatch** method passing the action to change the state.

```
export class SendComponent {
  newMessage: string = '';

  constructor(private _store: Store) {}

  send(event) {
    event.preventDefault();
    this._store.dispatch(changeMessage({message: this.newMessage}));
  }
}
```

Reading the state - Selectors

academeez

- We define Selectors which are functions to select from the state
- These functions are memoized to increase performance
- We create the selectors using **createSelector** method

```
import { createSelector } from '@ngrx/store';

export const selectMessage = createSelector(
  (state: any) => state.message
)
```

Reading the state - Component

academeez

- To read from the state we grab the **Store** service which is an observable emitting the current state
- We use **pipe** and the **select** operator to grab from the state the part that interests us using **Selectors**
- We get the data wrapped in Observable so we use the async pipe to use the data in the template - which means we can use OnPush

```
export class RecieveComponent {
  message$ = this._store.select(selectMessage);

  constructor(private _store: Store) { }
}
```

Using Service to pass data is much simpler

- We used to just place the data to a Service and use that service to change the data and read the data, it's much more complex with NGRX, what is the benefit?
- In ngrx the data Is wrapped in Observable which means we can use OnPush, to achieve the same in a Service we would have to manually wrap it in a Subject or Observable
- We split the change to actions and reducers, this way we can collect the action in an array and see the data change along a timeline
    - We achieve predictability
    - Easy testing
    - Easy undo redo
- We split the Selectors to improve reading performance

Predictability

academeez

- The fact that we separate the actions allows us to collect the array of actions and see exactly how the state got to it's current position
- We can examine the actions using a browser extension called **redux dev tools**
- We can install the package **@ngrx/store-devtools** and add the module to the imports array to connect our store to the devtools
- We can now examine our state and the actions that led us to the current state

Summary

- With @ngrx/store our data is managed using redux
  - Actions change the state
  - The reducer decides how the state will change
  - Components can read from the state using selectors
  - Components can change the state using the **store.dispatch**

academeez

Thank You

Next Lesson: 4. @ngrx/effects